IMAGE COLOR CORRECTION, ENHANCEMENT, AND EDITING

MAHMOUD NASSER MOHAMMED AFIFI

A DISSERTATION SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

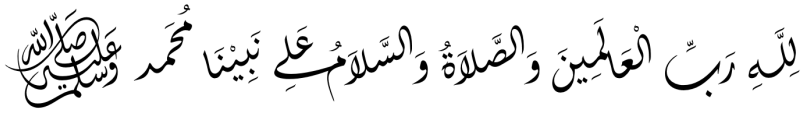
April 2021

Abstract

This thesis presents methods and approaches to image color correction, color enhancement, and color editing. To begin, we study the color correction problem from the standpoint of the camera's image signal processor (ISP). A camera's ISP is hardware that applies a series of in-camera image processing and color manipulation steps, many of which are nonlinear in nature, to render the initial sensor image to its final photo-finished representation saved in the 8-bit standard RGB (sRGB) color space. As white balance (WB) is one of the major procedures applied by the ISP for color correction, this thesis presents two different methods for ISP white balancing. Afterwards, we discuss another scenario of correcting and editing image colors, where we present a set of methods to correct and edit WB settings for images that have been improperly white-balanced by the ISP. Then, we explore another factor that has a significant impact on the quality of camera-rendered colors, in which we outline two different methods to correct exposure errors in camera-rendered images. Lastly, we discuss post-capture auto color editing and manipulation. In particular, we propose auto image recoloring methods to generate different realistic versions of the same camera-rendered image with new colors. Through extensive evaluations, we demonstrate that our methods provide superior solutions compared to existing alternatives targeting color correction, color enhancement, and color editing.

To my wife, kids, parents, and aunt

Acknowledgment



Praise be to Allah, Lord of the Worlds. Prayer and peace be upon the Prophet Muhammad. I owe a great debt to my supervisor Michael S. Brown for his close guidance, support, and valuable advice. Sincere thanks to Marcus Brubaker, Richard Wildes, Brian Price, Scott Cohen, Konstantinos Derpanis, Francois Bleibel, Jonathan Barron, Chloe LeGendre, Yun-Ta Tsai for their help and support throughout this work. I thank my labmates at York, Abhijith Punnappurath, Abdelrahman Abdelhamed, Abdullah Abuolaim, Hakki Can Karaimer, and Hoang Le for their valuable discussions. I would like to express my deepest gratitude to my family. I thank my parents and my aunt for their love, support, and prayers. My wife, Zeinab, for providing me her unending love, patience, companionship, and encouragement to keep pushing ahead in my research when doubt crept in. My last thanks go to my kids, Malika and Yaseen, my most important contribution of all.

Table of Contents

List of Tables

List of Figures

List of Acronyms

AE	Auto Exposure
AC	Auto-Color Function
AIM	Advances in Image Manipulation
AS	Adobe Standard
AT	Auto-Tone Function
AWB	Auto White Balance
BCP	Deep Bright-Channel Prior
BIN	Batch-instance normalization
BN	Batch Normalization
BTF	Brightness Transfer Function
Cat	Category
CAM	Color Adaptation Matrix
CC	Color Constancy
CCC	Convolutional Color Constancy
CFA	Color Filter Array
NIS	Natural Image Statistics
CIE	International Commission on Illumination
CLAHE	Contrast-Limited Adaptive Histogram Equalization

CNN	Convolutional Neural Network
conv	Convolutional
CPU	Central Processing Unit
CRF	Camera Response Function
CS	Camera Standard
CST	Color Space Transformation
dB	Decibel
DCE	Deep Curve Estimation
DNG	Digital Negative
DNN	Deep Neural Network
DS	Deep Specialized
DSLR	Digital Single-Lens Reflex
DoD	Distribution of Color Distributions
DPE	Deep Photo Enhancer
EGAN	Enlighten GAN
EMD	Earth Mover's Distance
EMoR	Empirical Model of Response
EV	Exposure Value
fc	Fully Connected
FC4	Fully Convolutional Color Constancy with Confidence-Weighted Pooling
FFCC	Fast Fourier Color Constancy
GAN	Generative Adversarial Network
GE	Gray Edges
GPU	Graphics Processing Unit
GT	Ground Truth

GUI	Graphical User Interface
GW	Gray World
HDR	High Dynamic Range
HE	Histogram Equalization
Histo	Histogram
HQEC	High-Quality Exposure Correction
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IoU	Intersection-over-Union
ISP	Image Signal Processor
JPEG	Joint Photographic Experts Group
KinD	Kindling the Darkness
KL	Kullback-Leibler
KNN	K Nearest Neighbor
LDR	Low Dynamic Range
LCD	Liquid Crystal Display
LIME	Low-Light Image Enhancement
LOL	LOw-Light
LReLU	Leaky Rectified Linear Units
LUT	Lookup Table
MAE	Mean Angular Error
MB	Megabyte
MoR	Model of Response
MSE	Mean Squared Error
NPE	Naturalness Preserved Enhancement
nm	Nanometers

NUS	National University of Singapore
PCA	Principal Component Analysis
PCC	Polynomial Color Correction
PI	Perceptual Index
PS	Adobe Photoshop
PSNR	Peak Signal-to-Noise Ratio
pxl-acc	Pixel-Wise Accuracy
RAM	Random-Access Memory
RBF	Radial Basis Function
Rec.	Reconstructed
ReHisto	Recoloring Histogram
ReLU	Rectified Linear Units
RHT	HDR Transformation
RMSE	Root Mean Squared Error
RNet	RetinexNet
RPC	Root-Polynomial Color Correction
SDK	Software Development Kit
SIIE	Sensor-Independent Illuminant Estimation
SIFT	Scale-Invariant Feature Transform
SoG	Shades of Gray
SPB	MIT Scene Parsing Benchmark
SPD	Spectral Power Distribution
sRGB	Standard RGB
SSIM	Structrual Similarity
SURF	Speeded-Up Robust Feature

SVD	Singular Value Decomposition
SVR	Support Vector Machine for Regression
t-SNE	t-Distributed Stochastic Neighbor Embedding
TTL	Through-the-Lens
U CC	Unsupervised Color Constancy
UPE	Deep Underexposed Photo Enhancer
UPI	Unprocessing Images
WB	White Balance
wGE	Weighted Gray Edges
WVM	Weighted Variational Model

Part I

Introduction and Prior Work

1 Introduction

The human visual system has the ability to filter out the color cast caused by the dominating scene illumination [180, 260]. This explains, in part, why an apple appears red under sunlight, incandescent light, and fluorescent light—even though these illuminations are significantly different in terms of their spectral profile. Camera sensors, however, do not have this ability and as a result, computational photography, or more specifically, computational color constancy (CC), is applied onboard cameras. In a photography context, this procedure is typically termed "white balance" (WB).

Most computational CC algorithms aim to achieve WB by correcting a scene's illumination to be ideal white light (i.e., the camera's sensor RGB responses to an achromatic object should lie along the achromatic "white line", that is R=G=B). In the literature, the scene illumination is usually assumed to be global and uniform (i.e., a single illuminant in the scene) [137]. Under this assumption, the WB correction process is carried out using a simple 3×3 diagonal matrix in order to undo the effect of the estimated illuminant [137]. One often overlooked issue with WB is that it is applied early in the processing chain directly to the sensor's RGB image—referred to as a raw image (or raw-RGB image). Cameras have a number of processing steps that convert the raw-RGB sensor response to the final output image. These collective steps result in a final output that is saved in a standard RGB (sRGB) image [191, 316].

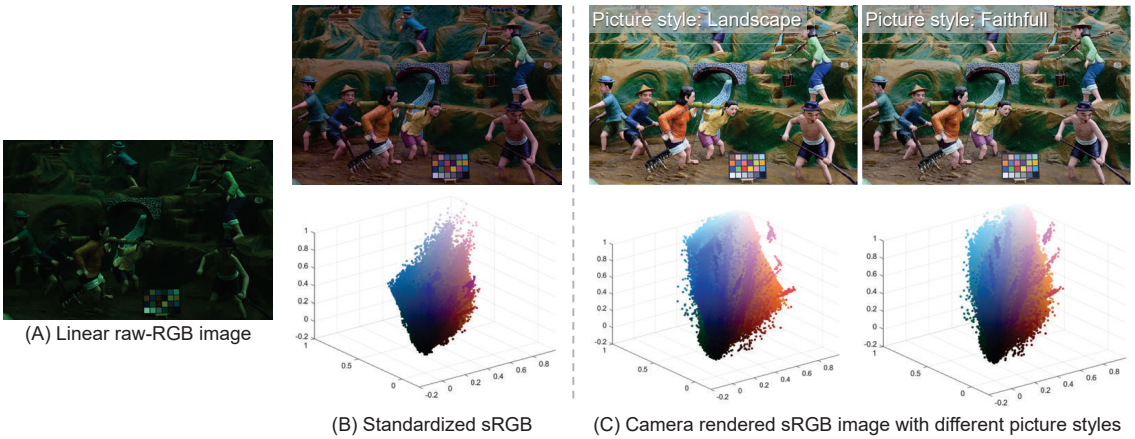


Figure 1.1: This figure shows how the standardized sRGB [31] differs from what the cameras produce. (A) A raw-RGB image captured by Canon EOS-1Ds Mark III from NUS dataset [77]. (B) Rendered sRGB image using a true sRGB encoding with only a single nonlinear gamma encoding applied [31]. (C) Camera rendered sRGB images with different picture styles that include camera-specific nonlinear color rendering. For each sRGB image, we show the RGB histogram in the second row. It is clear that each camera is applying its own proprietary color manipulation.

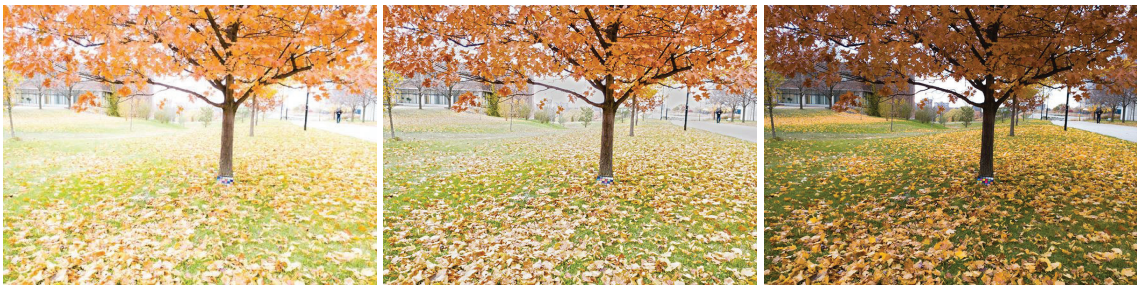
While sRGB color space specifies a nonlinear gamma encoding as part of its encoding regime, cameras apply several other nonlinear operators that are not specified in the sRGB standard. These proprietary nonlinear color manipulations (also called photo-finishing or camera style) are typically unique to a particular make and model of a camera. Moreover, they are often tied to various camera settings used at the time the photo was captured. That means, for instance, the *Landscape* photo-finishing of a Canon camera applies a different color rendering than that applied by a Nikon camera even with a similar setting and imaging the exact same scene. Figure 1.1 shows an example of a raw-RGB image rendered to an sRGB image using a single gamma operation [31]. This image is compared to other images from cameras using their onboard picture styles. The camera sRGB-rendered

images have notably different color distributions from the image's color distribution rendered by the sRGB [31]. Also, both camera rendered images have slightly different color distributions.

Given camera rendered final colors of photographs, it is nearly impossible to restore original linear scene-referred values (i.e., the raw-RGB image) without careful camera calibration to model the processing that was applied. Consequently, if the initial WB is computed incorrectly when capturing an image, it is challenging to undo afterwards in post-capture stage due to the nonlinear photo-finishing operations applied on the camera. In fact, WB errors are not the only camera errors that are hard to correct in post-capture stage. Exposure errors, for example, have a significant impact on the photographic quality of camera-rendered images; correcting such exposure errors in post-capture stage is more challenging than performing the correction in the linear sensor raw-RGB space as display-referred color spaces often have a smaller tonal range. Color correction and enhancement not only have crucial importance in photography aesthetics but also have a significant impact on different computer vision tasks, such as image retrieval, object tracking, texture classification, image forensics, and skin detection/classification [52, 89, 133, 189, 252, 390].

Figure 1.2 shows an example of post-capture color enhancement applied to an over-exposed photograph. As can be seen, the enhancement applied directly to the 8-bit sRGB display-referred image achieves a less pleasing result comparing with processing and re-rendering the linear raw-RGB space.

Unfortunately, there is a misconception in the computer vision and image processing community that a simple inverse gamma operation can undo the nonlinear operations applied on such camera-rendered sRGB images. Instead, to properly undo the onboard color rendering, a detailed reverse engineering per camera is required. There is an entire research branch in computer vision, termed radiometric calibration, with the sole purpose to reverse



(A) Input over-exposed image

(B) Corrected in the 8-bit sRGB nonlinear space

(C) Corrected in raw-RGB linear space

Figure 1.2: Correcting over-exposed photographs. (A) Input over-exposed image. (B) Correction is applied in the 8-bit sRGB color space. (C) Correction is applied in raw-RGB linear space.

the nonlinear processing applied within the camera pipeline (representative examples include [69,146,202,203,238–240,384]). Radiometric calibration is typically used for low-level computer vision tasks that require a linear response to scene radiance (e.g., photometric stereo, image deblurring, HDR imaging). When the necessary radiometric calibration data is available, it can be used to undo the photo-finishing in an sRGB image and effectively perform low-level computer vision tasks (e.g., WB correction) as demonstrated by [202]. However, performing radiometric calibration requires a tedious calibrating procedure and as a consequence, radiometric calibration data is rarely available for most users.

To demonstrate the problem with the misconception that camera images only have a nonlinear gamma applied, we provide a visual example. Figure 1.3 shows two input images with incorrect WB rendered to sRGB with different camera picture styles. For each image, we apply the simple gamma linearization (i.e., we apply the inverse gamma specified by the sRGB encoding standard). After the gamma linearization, we apply a diagonal WB correction using the exact white values extracted from a color chart placed in the scene. Eventually, we re-apply the gamma operation to get the final sRGB image.

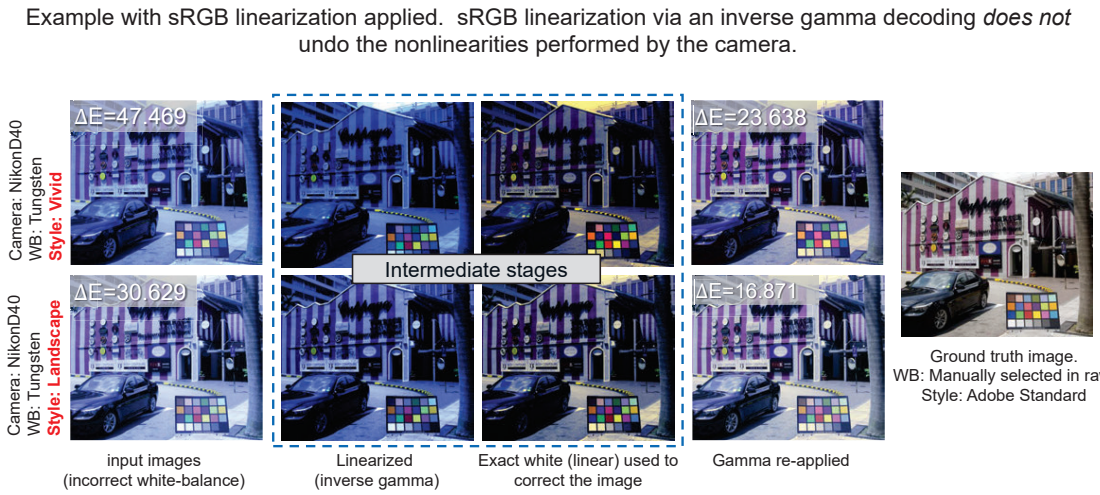


Figure 1.3: Shown are two input images with incorrect WB rendered in sRGB with different picture styles. Intermediate steps with the *inverse gamma* applied using the exact white values are shown. There are noticeable differences between the corrected images and the ground truth image that has been rendered to sRGB with the correct WB applied. This example is intended to show that trying to correct an sRGB image by first applying a simple gamma linearization is not sufficient. This strategy is a misconception commonly purported in the computer vision and image processing literatures.

There are noticeable differences between our corrected images and the ground truth image which is rendered with a correct WB applied to the original raw-RGB image before the photo-finishing step.

While color correction and enhancement in photographs are more effective in scene-referred linear spaces (e.g., camera sensor raw-RGB space) than color processing in post-capture stage, post-capture color editing techniques can achieve impressive results in color editing to modify the photographic style of the captured image. One of the main goals of color editing is to transfer a new "style and feel" to the captured image. Photo filters that are widely offered by social media and smartphone applications achieve this goal by

Figure 1.4: Example of static photo filter and dynamic color transfer results. In this example, we recolored the input image in (A) using: Instagram photo filters, as shown in (B), and a color transfer method [304] that accepts an additional target image to guide the recoloring process, as shown in (C).

applying a pre-defined set of lookup tables (LUTs) to transfer a new style or feel to the input captured image. In the literature, there is a large body of work proposed for dynamic color mapping [108], where instead of relying on static LUTs, these methods propose more sophisticated approaches to achieve more accurate and compelling results of color mapping based on any arbitrary target colors. That is, these methods offer a more fixable way for color transfer comparing to popular LUT-based photo filters (see Fig. 1.4).

1.1 Contributions

This thesis provides five research contributions summarized as follows. First, we propose two methods for CC in camera raw-RGB images. Specifically, we propose two different lightweight deep learning models for CC. Unlike other CC deep learning models, our proposed methods are sensor-independent meaning that they can be deployed to new camera models without a need for retraining or calibration.

Second, we propose the first framework that targets correcting improperly white-

balanced camera-rendered images in post-capture. Our framework is enabled by generating a large dataset (over 60K images) of improperly white-balanced images rendered in the sRGB color space. We show that this framework, along with our dataset, can also be exploited to emulate WB errors in camera-rendered photographs and thus can be used to augment training data for different computer vision tasks.

Third, three different methods for post-capture WB editing and manipulation are proposed. This part focuses in providing the user the freedom to edit the WB setting through interactive tools in post-capture stage. This WB editing and manipulation are required to match user preferences which do not always match accurate WB solutions.

Fourth, we propose two different methods for general color enhancement in photographs. Specifically, the goal of these methods is to enhance the colors of images rendered with exposure errors. As discussed earlier, correcting camera exposure errors in display-referred sRGB images is a challenging task due to the small tonal range of such sRGB images. This fact motivated us to propose a method for scene-referred image reconstruction. We show that this reconstruction can improve color enhancement of low-light and under-exposed camera-rendered images. Additionally, we generate a large dataset (over 24K images) with different exposure settings with broader exposure ranges. This generated dataset allows us to propose a deep learning method for directly correcting colors of under- and over-exposed photographs without a need to reconstruct such scene-referred images.

Fifth, we discuss two different color editing techniques. Unlike our WB manipulation methods which provide global color editing in photographs, this part of our thesis aims at local image recoloring. Specifically, we propose model color distributions of several semantic objects in order to achieve object-aware image recoloring. We then extend this idea by proposing a generative adversarial network (GAN) to control colors in images. We show that our method can achieve auto image recoloring without a need for target images

or any user interaction.

1.2 Thesis Outline

This thesis consists of eight parts. In Part I, we discuss the primary color rendering operations applied onboard cameras to render final sRGB photograph colors (Chapter 2). Chapter 2 also discusses existing methods to linearize the sRGB image (i.e., from sRGB to linear RGB images) and their limitations. Next, Chapter 3 reviews prior work for color correction and manipulation. This survey discuss methods proposed for color correction (CC and WB), color enhancement in photographs rendered with exposure errors, and post-capture color manipulation. Part II of this thesis includes two chapters (Chapters 4 and 5) that present two different sensor-independent illuminant estimation methods. Afterwards, we discuss our framework for correcting improperly white-balanced images in Part III. We first outline the details of our WB correction framework in Chapter 6. Then, we show how this framework can be extended to improve the robustness of computer vision tasks against WB errors in photographs (Chapter 7). Part V includes two chapters (Chapters 8 and 9) that present our methods for enhancing colors of under- and over-exposed photographs. The last contribution of this thesis is discussed in Part VI, where our image recoloring methods are presented in Chapters 10 and 11. Thesis conclusions and future work are discussed in Part VII (Chapter 12). Lastly, bibliography and appendices are given in Parts VIII and IX, respectively.

2 Preliminaries

The sRGB color space is the primary color space used to save images captured by digital cameras. As a result, sRGB images are the primary format used by many computer vision systems. In the color science community, sRGB is known as an "output-referred" or "display-referred" color space as it is intended for use on display devices (monitors, LCD screens and even printers). In this chapter, we present an overview of the formation of sRGB images through the lens of the camera imaging pipeline. To begin, we present a brief review of different standard color spaces. We then overview the in-camera image processing pipeline that is responsible for converting sensor raw-RGB images to the corresponding display-referred sRGB images. Lastly, we discuss the possibility of converting the sRGB colors back to the original linear format.

2.1 Standard Color Spaces

Colors are not physical characteristics of objects. Colors are words we use to describe sensations that arise from our perception of objects based on both the material characteristic of an object (e.g., surface reflectance and specularity) and the incoming visible light, which is within a certain band of the electromagnetic spectrum, called the visual spectrum[1] for typical human eyes [101].

[1]The visual spectrum is roughly from 380 to 780 nanometers (nm).

When the human eye receives visible light spectra, there are three spectral sensors in the retina, known as cone cells, responsible for the color vision process. The cones were named according to the ordering of peak wavelengths, where the three cones are: (i) L: long cone (560-580 nm), (ii) M: medium cone (530-540 nm), and (iii) S: short cone (420-440 nm) [107]. When these cones receive a visible light, their responses can be modeled by the following equation:

$$c_i = \int_{\gamma} \rho(\lambda) s_i(\lambda) d\lambda, \tag{2.1}$$

where γ is the visible spectrum, $s(\cdot)$ is the spectral sensitivity of the i^{th} cone cell at wavelength λ, and $\rho(\cdot)$ is the incoming spectral power distribution (SPD) emitted or reflected from an object—this combines both the object's reflective properties and the scene's illumination. The final color is perceived after mixing the received colors by each cone, see Fig. 2.1.

According to Eq. 2.1, two different SPDs can be perceived identically due to the accumulation effect of the three cones. This phenomenon is referred to as metamerism and color samples that are perceived identically under the same lighting conditions are called metamers. From Eq. 2.1, we can also notice that any color can be matched by a linear combination of the three independent responses [145].

Even before the three spectral sensitivities were physiologically discovered, a set of psycho-physical experiments were carried out in order to establish a standard color space [377]. These experiments determined the color mapping functions of a human observer through a mixing of relative amounts of three standard "primary" colors. This formed the basis of the CIE RGB color space—the term CIE refers to the Commission Internationale de l'Eclairage in French, also known as The International Commission on Illumination in English. One problem with the CIE RGB color space is that the proposed RGB primaries did not span the full visible range. As a consequence, some of the primaries were mixed in

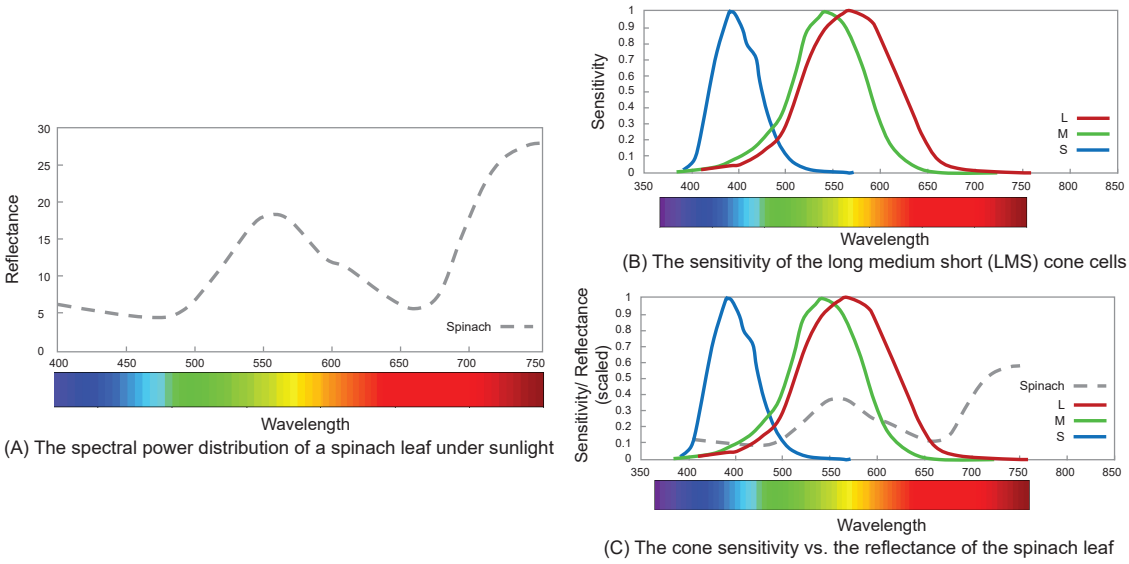


Figure 2.1: In this example, we show the spectral power distribution (SPD) of a spinach leaf captured under sunlight (A). In the retina, there are three cone cells, namely long (L), medium (M), and long (L) cone cells. Each cone is sensitive for only a certain wavelength range that match a range of colors as shown in (B). The reason of why the spinach leaf appears greenish is that the reflected visible light of it mostly matches the medium cone cell, as shown in (C).

with negative values until matching the target colors, see Fig. 2.2-(A). To overcome this issue in CIE RGB color space, the CIE derived a new color space from the CIE RGB color space with no negative points, as shown in Fig. 2.2-(B). This color space was called 1931 CIE XYZ space [82] and is now widely accepted as a canonical device-independent color space [24]. The variables X, Y, Z were used as they do not correspond to known color sensations. This canonical color space is also used to define standard illuminants based on their SPDs. For example, the standard CIE illuminant A represents incandescent light; while standard CIE illuminant series D is defined for natural daylight light. The latter includes different standard illuminants, such as D50 (horizon light) and illuminant D65

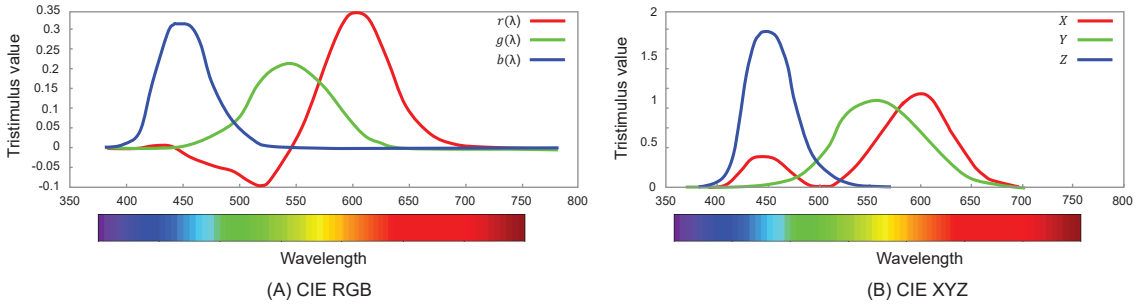
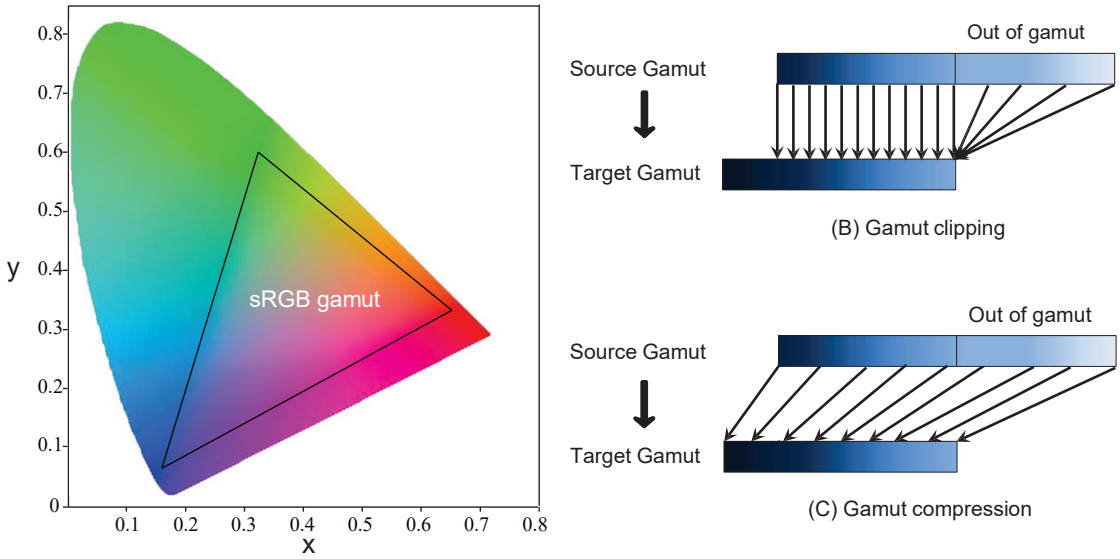


Figure 2.2: The tristimulus values through the visible spectrum of: (A) CIE RGB [377] and (B) CIE XYZ spaces.

(noon light).

The CIE 1931 XYZ color space has remained the dominant color space used by the vast majority of imaging devices. Many other color spaces common in academic and engineering research are derived directly from the CIE XYZ. These include NTSC, YUV, YIQ, CIE $\mathtt{L^*a^*b^*}$, sRGB, and Adobe RGB color spaces. Among the existing color spaces, the 24-bit sRGB color model is the most dominant in consumer electronic systems. This color space was introduced in 1996 by Hewlett-Packard (HP) and Microsoft as a standardized and universal color space for all devices (e.g., monitors, printers, scanners, and digital cameras), through representing each color channel by 8 bits within the sRGB gamut [31]. The sRGB gamut is shown in Fig. 2.3-(A). Converting the CIE XYZ values to the corresponding sRGB tristimulus values, according to the standard conversion introduced in 1996, can be performed by the following steps:

- Converting from CIE XYZ values (X, Y, Z) to the corresponding linear sRGB values



(A) CIE 1931 color space chromaticity and the sRGB gamut

Figure 2.3: (A) The CIE 1931 x-y chromaticity diagram (i.e., a 2D projection of the CIE 1931 XYZ values onto the plane represented by $X+Y+Z=1$) and the sRGB gamut. (B) and (C) show two gamut mapping approaches, namely gamut clipping and compression, respectively. This figure is adapted from [154].

$(sR_l,\ sG_l,\ sB_l)$ using the following equation:

$$\begin{bmatrix} sR_l \\ sG_l \\ sB_l \end{bmatrix} = \mathbf{T}_{XYZ2sRGB} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}, \tag{2.2}$$

where $\mathbf{T}_{XYZ2sRGB}$ is a nonsingular transformation matrix that maps between the CIE XYZ and the corresponding linear sRGB values.

- Converting from the linear sRGB values to the nonlinear sRGB values based on the following equation:

$$sR = \begin{cases} sR_l \times 12.92 & \text{, if } sR_l \leqslant 0.00304 \\ \left(sR_l^{(1.0/2.4)} \times 1.055\right) - 0.055 & \text{, otherwise.} \end{cases} \tag{2.3}$$

The above equation is applied to the other linear color channels (i.e., sG_l and sB_l) to get the final sRGB values. The result of this equation is represented by 24 bits for each color (8-bits/channel) with a close fit to the 2.2 gamma curve. The idea behind fitting the 2.2 gamma curve, also known as gamma encoding, is to exploit the nonlinearity of the human perceptual system whose sensitivity to brightness can be approximately represented by a power function—the human perceptual system is more sensitive to darker tones than lighter ones [309]. In this way, the usage of 8 bits per channel is optimized in order to represent a wide range of different perceptual colors.

One important benefit of adhering to the standardized approach of converting CIE XYZ values to their corresponding nonlinear sRGB colors is the ability of computing the inverses of this operation to get the original CIE XYZ values—meaning that any two pixels with the same sRGB value should have the same CIE XYZ value, and consequently they represent, "in theory", exactly the same perceptual scene color. This conversion can simply be performed using the following equations:

$$sR_l = \begin{cases} \left(sR/255\right)/12.92 & \text{, if } sR/255 \leqslant 0.03928 \\ \left(\left(sR/255 + 0.055\right)/1.055\right)^{2.4} & \text{, otherwise,} \end{cases} \tag{2.4}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \mathbf{T}^{-1}_{XYZ2sRGB} \begin{bmatrix} sR_l \\ sB_l \\ sG_l \end{bmatrix}. \tag{2.5}$$

Despite the success of the idea of having a standardized color space and the adoption of many devices of it (e.g., cameras, displays), no device follows this standard convention in reality, as will be explained in the next section.

2.2 Camera Imaging Pipeline

Digital cameras apply a series of processing routines to convert a captured raw-RGB sensor image to the final sRGB output image. These routines are part of the image signal processor (ISP) hardware on the camera. While each camera manufacturer has its own customized ISP, researchers have developed a reasonably representative ISP model that includes the main components of camera imaging pipelines [191, 316]. Figure 2.4 shows these main components. In the following part of this section, we will explain each component in detail.

Reading and Pre-processing the Raw Image The first input to the camera pipeline is the mosaiced raw image. This image contains digital values that are linear with respect to the amount of physical light irradiance that fell on the camera sensor for some given exposure. The sensor's photodiodes are covered with a color filter array (CFA) that is arranged on a square grid. The format of this CFA can vary based on the camera manufacturer, but the mosaiced Bayer pattern is widely used by digital cameras.

Most sensors have a number of defective pixels. Subsequently, defective pixels (e.g., dead/dark pixels or bright pixels, also called hot pixels) are masked out using a pixel mask. The values for these corrupted pixels are interpolated based on their neighboring values. Due to thermal noise on the sensor, a pixel receiving no light might still output positive values which is known as the black level. Cameras perform a black-level subtraction and then normalize pixel values based on the maximum value (also known as the saturation

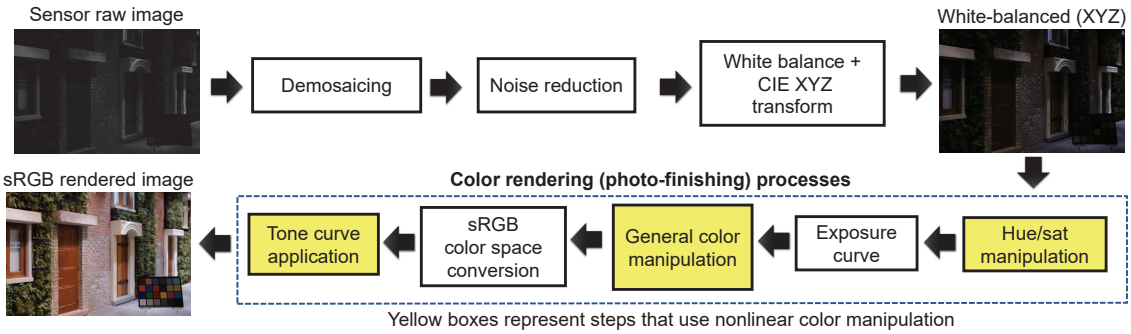


Figure 2.4: This figure shows a simplified the camera processing pipeline, where a set of image processing/low-level computer vision operations are applied early in the pipeline (e.g., demosaicing, denoising, white balancing). Afterwards, a number of nonlinear color manipulations (highlighted in yellow) are applied to obtain the sRGB output image. Note that this arrangement and the details of the shown camera pipeline steps may differ based on the camera manufacturer and model producing different colors by each camera capturing the same scene based on its model and settings.

level) to be in the [0-1] range using 10-16 bits. Due to the effect of the lens distortion within many cameras, flat-field correction is performed to reduce the vignetting effect.

Demosaicing After the black-level and vignetting corrections, camera ISP performs an interpolation process to produce R, G, B raw digital values per pixel. This process is commonly referred to as demoasicing or debayering. This process can be performed using a simple nearest neighbor interpolation or using more sophisticated algorithms, such as pixel grouping [188], interpolation using alternating projections [149], or even using deep learning models [243]. This process produces a full three channel raw-RGB output.

Noise Reduction This step aims to reduce any noise that naturally occurs on the camera sensor. A review of noise reduction is outside the scope of this thesis, but it should be

noted that there is a large body of literature dedicated to this topic (for surveys on image denoising, see [267, 276]).

White Balance and Color Space Conversion At this stage, WB correction is applied to the demosaiced raw-RGB image. Some cameras allow the user to select a preset WB from the camera's settings, or more commonly an auto WB (AWB) routine is used. AWB involves estimating the scene illuminant (represented as a vector $\in \mathbb{R}^3$) and then applying the WB correction (typically is performed using a diagonal matrix based on the estimated illuminant vector). In the next chapter, we will discuss further details on the WB process. Based on the estimated illuminant color, the correlated color temperature is computed (more details are given in Appendix B) and a colorimetric conversion is then applied[2] to map the white-balanced raw-RGB values to a canonical perceptual color space—namely, the CIE 1931 XYZ space. Usually this is performed using a 3×3 full color space transformation matrix (CST). Note that if the WB is applied incorrectly, the image will have a strong color cast and the resulting CIE XYZ values will be wrong.

Hue/Saturation Manipulation Up to this stage, the processed image in the camera pipeline is in a canonical perceptual color space (CIE XYZ or one of its derivatives) and is directly associated to the physical scene image (i.e., scene-referred). However, this link is broken starting from this stage due to the camera-specific nonlinear operations that aim to produce a "pleasing" representation of the captured scene. The hue/saturation manipulation is one of these nonlinear operations and it is usually implemented as a 3D LUT.

[2]Some camera manufacturers omit this colorimetric conversion and convert white-balanced sensor colors to the sRGB space directly.

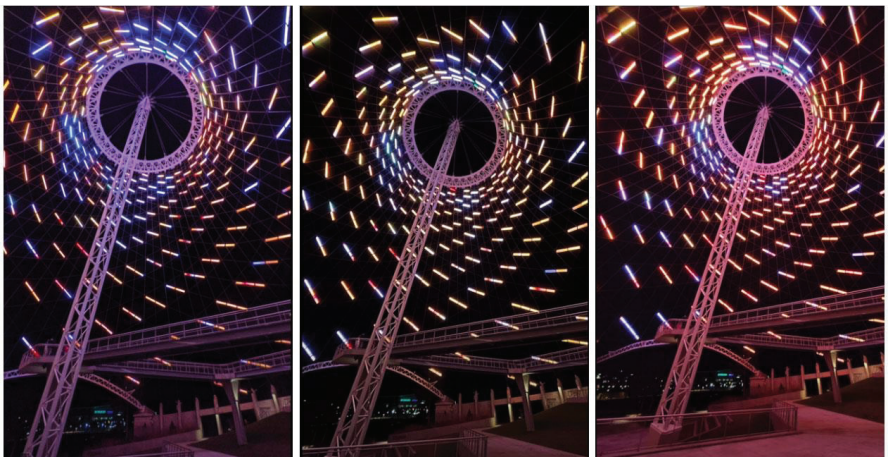


Figure 2.5: Three images captured the same scenes using different camera models, however each camera produced different colors. Photo credit: Max Tech on YouTube.

Exposure Compensation Beside the physical exposure control (i.e., shutter speed and aperture size), digital exposure can be applied to the pixel intensities using a simple linear gain.

General Color Manipulation Each camera has its own color manipulation function that is usually represented by a 3D LUT and it is applied in order to get more visually pleasing colors.

sRGB Color Space Conversion At this stage, a 3×3 full matrix (e.g., $\mathbf{T}_{XYZ2sRGB}$) is used to convert pixel values from the previous stage to the final sRGB color space. During this stage, a gamut mapping operation is performed to map the out-of-gamut pixels to the sRGB gamut. The simple approach is gamut clipping [275], illustrated in Fig. 2.3-(B), while gamut compression can be used for a better mapping, illustrated in Fig. 2.3-(C).

Tone Curve Application Before rendering the final image, a camera-specific tone map operation is applied. This tone map may have different effects based on the selected camera style before capture. It is worth noting that this tone mapping operation may include local operations that are dynamically changed based on the current scene context [134,155,280]. Because the sRGB encoding standards recommends a 2.2 gamma encoding, it is often erroneously assumed that the gamma encoding is the tone curve. However, virtually no camera applies merely this simple gamma encoding [155, 191, 202, 280, 316].

Lastly, it is important to emphasize that the parameters, the arrangement, and the details of the previously described camera pipeline steps may differ based on the camera manufacturer and model producing different colors by each camera capturing the same scene based on its model and settings. Figure 2.5 shows three images of the same scene captured by three different cameras. It is apparent that each image has different colors.

2.3 From sRGB to Linear RGB

As mentioned earlier, the camera imaging pipeline contains several set of nonlinear operations applied to generate a more visually pleasing sRGB image. Applying the standard inverse gamma operation (Eq. 2.4) is a long standing misconception found on wiki pages and provided by even well-known computing environments and libraries, such as Matlab and OpenCV, as a solution to linearize *any* sRGB image. This is a serious problem not only for computing the CIE XYZ values, but also for converting the sRGB colors to any color space derived from the CIE XYZ (e.g., CIE $\mathtt{L^*a^*b^*}$). Moreover, many of the photometric-based post-processing procedures, applied to sRGB images, are subject to a considerable amount of error [284]. For these reasons, there is a large body of radiometric calibration literature to obtain a more accurate reconstruction of the linear RGB image.

By definition, radiometry refers to quantitative measurements of electromagnetic radi-

ation (either the light source radiance or the surface irradiance). Radiometric calibration aims to invert the nonlinear operations applied to the sRGB-rendered images to reconstruct the image irradiance $\mathbf{I}$ weighted by the spectral sensitivities of the R, G, B filters on the camera [148]. In other words, the goal of radiometric calibration is to model the camera response function (CRF), $f_{\text{CRF}} : \mathbf{I} \rightarrow \mathbf{I}_{\texttt{sRGB}}$, and the model of response (MoR), $f_{\text{CRF}}^{-1} : \mathbf{I}_{\texttt{sRGB}} \rightarrow \mathbf{I}$.

Most of the conventional radiometric calibration algorithms approximate the CRF to linearize the sRGB image. This linearization process, however, does not reconstruct the original raw-RGB image [202]. Instead, its goal is to invert the nonlinear functions applied onboard the camera without considering the effects of the other components in the camera imaging pipeline (e.g., WB, color space conversion, or gamut mapping); see Fig. 2.6. As a response, we organize this section into two parts. First, we discuss the *full* reconstruction process of the raw-RGB image. Second, we review the main strategies of the existing radiometric calibration methods.

2.3.1 Raw-RGB Image Reconstruction

A simplified sRGB image formation model can be represented by the following equation:

$$\mathbf{I}_{\texttt{sRGB}} = f_{\text{CRF}}(\mathbf{I}). \tag{2.6}$$

We can formulate the problem of reconstructing the raw-RGB image as:

$$\mathbf{I} = f_{\text{CRF}}^{-1}(\mathbf{I}_{\texttt{sRGB}}). \tag{2.7}$$

Due to the complexity of the nonlinear camera pipeline's processes, it is non-trivial to find f_{CRF}^{-1} without having a prior knowledge of the camera model and the capture settings. To perform proper radiometric calibration, it is necessary to capture many images of a color rendition chart, or other calibration device, under a controlled environment. One

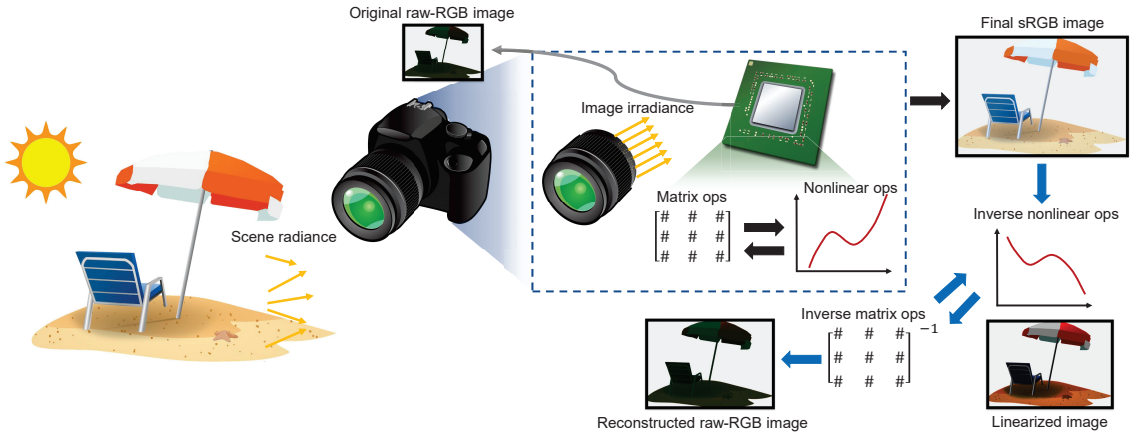


Figure 2.6: Reconstruction of raw-RGB images requires inverting the operations performed in the sRGB image formation in order to reconstruct a demosaiced form of the image irradiance. The linearization process aims only to undo the nonlinear effects applied onboard the camera to obtain a linearized representation of the image.

approach could be to measure the camera responses inside a white sphere to a varying incoming light in order to model the correspondence between scene radiance and measured pixel values [351]. Simpler solutions can be given by capturing a color rendition chart under uniform lighting conditions [147].

Grossberg and Nayar [148] found that real-world cameras have a bounded space of CRFs. Thus, they proposed an empirical model of response (EMoR) obtained based on 201 real camera responses. In their work, they approximated the CRF as a nonlinear response of the camera without considering other factors (e.g., WB) through a brightness transfer function (BTF) f_{BTF} that maps the image from some linear form to the final pixel brightness in the sRGB space. Their EMoR is represented as a principle component analysis (PCA)-based model of f_{BTF}^{-1}. Specifically, the EMoR is formatted by the following equation:

$$\hat{f}_{\text{BTF}}^{-1} = \mu(f_{\text{BTF}}^{-1}) + \mathbf{Lb}, \tag{2.8}$$

where $\mu(\cdot)$ represents the mean of the BTFs of different real-world cameras, $\mathbf{L}$ is a matrix whose columns contains the first g eigenvectors, and $\mathbf{b} \in \mathbb{R}^g$ is the PCA coefficient vector. This EMoR representation gives the ability to approximate the complete BTF using a fewer number of parameters (e.g., $g = 5$ used in [229, 239]).

Kim *et al.* [202,238] proposed a new in-camera imaging model in order to reconstruct an accurate raw-RGB image from the given sRGB image. In their model, they can effectively reconstruct the MoR by formulating the problem as follows:

$$\mathbf{I}_{\texttt{sRGB}} = f_{\text{CRF}}(\mathbf{I}) = f_{\text{BTF}}(h(\mathbf{I}\ \texttt{diag}(\boldsymbol{\ell}^*)\mathbf{T}_{\text{CAM}})), \tag{2.9}$$

where $h(\cdot)$ is a nonlinear 3D gamut mapping function, $\texttt{diag}(\boldsymbol{\ell}^*)$ is a 3×3 diagonal matrix for WB correction, and $\mathbf{T}_{\text{CAM}}$ is a 3×3 full color transformation matrix that converts from the camera space to the linear RGB space—this matrix combines the CAM and the XYZ-RGB conversion matrix. This formulation allowed them to obtained an accurate reconstruction of raw-RGB images by calibrating the camera model in order to define $\mathbf{T}_{\text{CAM}}^{-1}$, $\texttt{diag}(\boldsymbol{\ell}^*)^{-1}$, h^{-1}, and f_{BTF}^{-1}. This calibration process was performed in a three-stage manner. First, they estimated f_{BTF} based on the PCA model of camera responses [148] from non-saturated pixels that are unaffected by the gamut mapping process. Second, they calibrated the $\mathbf{T}_{\text{CAM}}$ and $\texttt{diag}(\boldsymbol{\ell}^*)$ matrices from the linearized sRGB values after applying the calibrated f_{BTF}^{-1}. Third, h^{-1} is modeled by a non-parametric model based on a point interpolation using radial basis functions (RBFs). Xiong *et al.* [384] extended this idea to provide a distribution of the possible raw-RGB image colors using a probabilistic model with an uncertainty prediction. A more recent deep learning-based solution was proposed by Nam and Kim [280]. Their model relies on the image's scene context and color distribution in order to reconstruct the original raw-RGB image.

Despite the accurate reconstruction, these methods require the presence of the camera models used to capture the sRGB image in order to either calibrate the camera [202, 238]

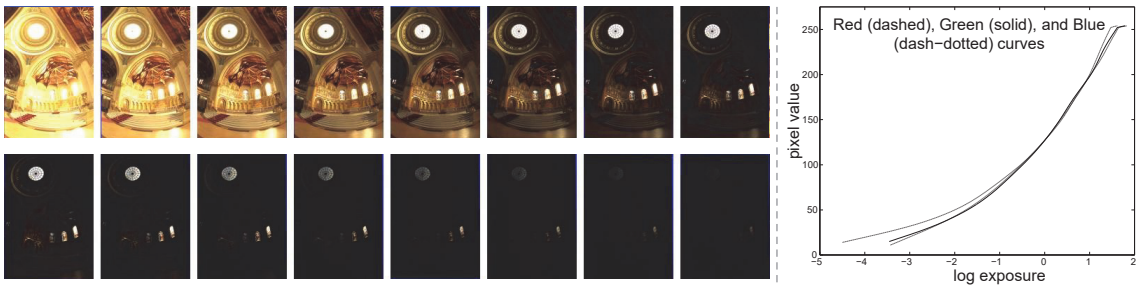
or train a reconstruction model [280].

Recently, Jiang *et al.* [181] showed that the camera imaging pipeline can be represented as a set of affine transformation matrices to map the raw-RGB image to the sRGB image. They achieved this by clustering the sensor data into various classes based on their spatial and color information followed by learning the transformation matrix for each class.

Nguyen and Brown [286] proposed to embed a small memory overhead to keep the necessary metadata in order to reconstruct the original raw-RGB image. This metadata includes information for tone mapping, WB, color space transform, saturation pixels, and the color manipulation for gamut mapping. To keep the overhead small, some assumptions and approximations were made. First, they assumed the tone mapping operation affects only the chromatic colors, so they employed only the V channel of the HSV color space to reduce the required information for the reconstruction process. The 3D sRGB color histogram was approximated by scattered points in order to undo the gamut mapping operation. Lastly, the original values of the saturated pixels are saved to avoid the problem of overexposed pixels. At the end, their approach effectively encoded useful metadata for reconstructing the raw-RGB image from the sRGB-JPEG image. Unfortunately, such metadata is not provided by the existing camera models [311].

2.3.2 Radiometric Calibration

The settings required to reconstruct the original raw-RGB image are tedious and impractical in many scenarios. As a result, most of the conventional radiometric methods try to model BTFs instead of the complete CRFs. In other words, they aim to undo the effect of the BTF without the need for a calibration object in order to reconstruct a radiometrically linear representation $\mathbf{I}_l$ of the sRGB image $\mathbf{I}_{\texttt{sRGB}}$ rather than the original raw-RGB image $\mathbf{I}$. Now, the problem is usually formulated as the following equation [201]:



(A) Same scene captured with different exposure times

(B) Estimated response functions

Figure 2.7: Reconstructed response functions from multiple images captured with different exposures using Canon 35mm SLR camera, adapted from [90]. (A) Aligned sRGB images captured with different exposure times. (B) Estimated response functions for Canon 35mm SLR camera by Debevec and Malik [90].

$$\mathbf{I}_{\texttt{sRGB}} = f_{\text{BTF}}(\varphi \mathbf{I}_l \boldsymbol{\ell}), \tag{2.10}$$

where φ is the exposure value used during capturing $\mathbf{I}_l$. One solution is to use multiple aligned sRGB images of the same scene with different known exposure values under a constant lighting condition to construct a matrix of the "brightness" values. This matrix can be used to reconstruct the linearized image by fitting the brightness/exposure data [90,261]. Figure 2.7 shows an example of the nonparametric recovered BTF obtained by Debevec and Malik [90]. Based on the idea of having varying exposure, many modifications were proposed (e.g., iterative polynomial-based solving [268], spatially varying optical mask [281], prior-based probabilistic model [294]).

Another solution is based on obtaining a set of images of the same scene under different lighting conditions by employing the EMoR [148] to estimate the PCA coefficients using pixels with the same lighting conditions in the scene [201]. The vignetting function also can be integrated into Eq. 2.10 as well, while assuming a radial symmetry vignetting to estimate the radiometric response function from a sequence of sRGB images [203].

The main limitation of these methods is the need to capture a set of images under certain settings. Also, since the BTF can be a scene-dependent nonlinear function, the assumption of a fixed response function per channel is not sufficient [148]. As a solution to all these problems, scene geometric calibrations were performed for a large set of training images to benefit from the estimated normal vectors of scene surfaces [96,270]. This enables the ability to estimate the response function of a given sRGB-rendered image. However, these methods require a large amount of training data that is pre-geometrically calibrated.

2.4 Summary

In this chapter, we have provided an overview of the sRGB color space and its formation. We have explained the difference between the standardized approach for generating sRGB colors and the existing camera imaging pipeline operations. We have shown that the current camera models apply a sequence of nonlinear operations in order to make the captured scene more pleasing regardless of the effects on the relation between the image colors and the real scene colors. These nonlinear operations make it hard to reconstruct the original linear colors. Accordingly, a full radiometric calibration is required. However, we have discussed how full radiometric calibration requires tedious image processing steps or the embedding of necessary metadata to help in the reconstruction process. We have also reviewed representative examples of radiometric calibration methods that aim to linearize the sRGB image in a more efficient way than the simple inverse gamma operation. We have shown that these models did not consider the effect of main components of the camera imaging pipeline. Additionally, they require certain conditions to work properly.

3 Prior Work

This chapter reviews prior work proposed for color correction and editing in photographs. Specifically, we discuss in more detail the image white balancing process, which is one of the major procedures that are responsible for color correction and manipulation on board cameras (Sec. 3.1). Then, we will elaborate on why correcting colors in the post-capture stage is more challenging; especially, if the camera-rendered images have some errors in WB (Sec. 3.2). Afterward, we will discuss other factors that directly contribute to the quality of camera-rendered image colors. In particular, we will discuss exposure errors in cameras and how such errors have a significant impact on the final rendered colors by cameras (Sec. 3.3). Lastly, we will briefly review post-capture image color editing techniques (Sec. 3.4).

3.1 Image White Balancing

WB is applied as an approximation to color constancy (CC), described earlier in Chapter 1, that is the term given to the human visual system's ability to perceive an object's color as the same when viewed under different illumination [398]. Camera sensors lack this ability and unprocessed raw-RGB camera images contain noticeable color cast due to the scene's illumination. WB, or more generally computational CC, is a fundamental processing step applied onboard cameras to compensate for scene illumination.

We can formally describe WB in terms of the image formation process. Let $\mathbf{I} =$

$\{\mathbf{I}_r, \mathbf{I}_g, \mathbf{I}_b\}$ denote an image captured in the linear raw-RGB space. The value of each color channel $c = \{\mathrm{R}, \mathrm{G}, \mathrm{B}\}$ for a pixel located at x in $\mathbf{I}$ is given by the following equation [46]:

$$\mathbf{I}_c(x) = \int_{\gamma} \rho(x, \lambda) R(x, \lambda) S_c(\lambda) d\lambda, \tag{3.1}$$

where γ is the visible spectrum, $\rho(\cdot)$ is the illuminant spectral power distribution, $R(\cdot)$ is the captured objects' body reflectance (i.e., diffuse reflection component), and $S(\cdot)$ is the sensor response function at wavelength λ. According to this simple model, the surface appears the same from all viewing directions—assuming there is no specular reflection. The problem can be simplified more by assuming a single uniform illuminant. Hence, the problem can be written as:

$$\mathbf{I}_c = \mathbf{R}_c \times \boldsymbol{\ell}_c, \tag{3.2}$$

where $\boldsymbol{\ell}_c$ is the color channel c of this single illuminant (see Fig. 3.1. We assume that black-level subtraction has been applied to $\mathbf{I}$. Now, the problem can be solved by a simple linear model (i.e., a 3×3 diagonal matrix) to make $\boldsymbol{\ell}_{\mathrm{R}} = \boldsymbol{\ell}_{\mathrm{G}} = \boldsymbol{\ell}_{\mathrm{B}}$ (i.e., white illuminant).

Typically, $\boldsymbol{\ell}$ is unknown and should be estimated from the *linear raw-RGB images*. Illumination estimation is one of the fundamental processes performed onboard cameras as a part of their WB feature. Illumination estimation methods predict the color of the scene's illumination from a captured image in the form of an R, G, B vector in the sensor's raw-RGB color space.

The straightforward way to do this is to capture an image of an object that acts as a pure reflector—for example, an achromatic (i.e., white or neutral) object. Under ideal white light, the camera's sensor response to the achromatic object should lie along the achromatic "white line"; that is, R=G=B. The scale of the R, G, B values depend on the intensity of the reflected light from the object. Under non-white illuminations, the camera's response to a pure reflector would *not* lie along the achromatic line and the R, G,

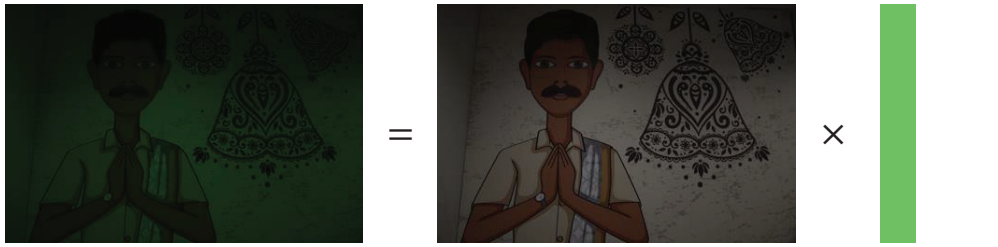


Figure 3.1: Problem formulation of color constancy with the assumption of a single uniform lighting condition. The shown image is from the INTEL-TAU dataset [215].

B responses would, therefore, represent the measurement of the illumination in the sensor's color space.

In this section, we present a survey on the existing WB methods from raw-RGB images. Then, we examine the ability of existing WB techniques to correct improperly white-balanced sRGB-rendered images.

The WB process consists of two steps: (i) estimating the color of the illumination in the camera's sensor space and (ii) correcting the image based on the estimated illumination.

3.1.1 Illuminant Estimation

In practice, we do not always have neutral patterns in our scenes and the color of the illumination must instead be estimated directly from captured images. This illuminant estimation is a challenging problem, because it is fundamentally under-constrained: an infinite family of white-balanced images and global color casts can explain the same observed image. Illuminant estimation is, therefore, often framed in terms of inferring the most likely illuminant color given some observed image and some prior knowledge of the spectral properties of the camera's sensor.

We can categorize illumination estimation methods into two different categories: (A)

single-illuminant scene estimation and (B) multi-illuminant scene estimation.

While there are few attempts proposed for illuminant estimation of multi-illuminant scene (e.g., [43, 48, 100, 114, 174, 198, 321]), the majority of prior work adopted the single illuminant assumption. Generally, single-illuminant scene estimation methods fall roughly into four categories: (i) statistical methods, (ii) physics-based methods, and (iii) learning-based methods.

Statistical-Based Methods

Statistical-based methods operate using statistics from an image's color distribution and spatial layout. Most statistical-based methods are based on one or more assumptions in order to apply a set of generic statistics to estimate the illuminant color vector.

The *gray world* (GW) assumption [60], for instance, assumes that the mean of image irradiance is achromatic (i.e., "gray"). That is, the algorithm computes the mean of the given raw-RGB image in order to estimate the illuminant color (i.e., $\overline{\mathbf{I}_c} \propto \boldsymbol{\ell}_c$). Smoothing the camera responses can be applied to compute an initial local averaging before employing the gray world assumption [137]. This smoothing operation is usually performed using a Gaussian filter. This method is called general gray world (GGW). Potential improvements can be achieved by using a weighted gray world, such that the mean of each color channel is adapted based on its standard deviation [218, 295].

From another perspective, other methods assumed the presence of white objects with larger intensity values than other pixels in the captured scene. This assumption is called the *white patch* hypothesis which can be implemented by computing the maximum response of each color channel (i.e., max-RGB) [58]. This hypothesis is extended later in the bright pixels algorithm (BP) [187] by considering the gamut of the bright pixels instead of the simple max-RGB method. Finlayson and Trezzi [120] showed that the max-RGB and

gray world algorithms are special cases of a more generic algorithm for computational CC, referring to it as shades of gray (SoG), which assumes the mean of the Minkowski p-norm of the scene is shades of gray. Specifically, the gray world and max-RGB algorithms can be represented by the following equation:

$$\frac{\|\mathbf{I}_c\|_p}{N^p} \propto \boldsymbol{\ell}_c, \tag{3.3}$$

where $\|\cdot\|_p$ is the p-norm and N is the total number of pixels in $\mathbf{I}$. For $p = 1$, Eq. 3.3 represents the gray world algorithm, while if $p = \infty$, Eq. 3.3 computes the max-RGB. For $p \in]1, \infty[$, the equation refers to the SoG algorithm. They found that the best results obtained with $p = 6$.

Unlike the previous assumptions which rely merely on the color information, *gray edges* (GE) assumption assumes that the mean reflectance differences in a scene is achromatic [362]. In this context, Eq. 3.3 can be modified to be:

$$\frac{\|\bigtriangledown \texttt{blur}(\mathbf{I}_c, \sigma_b)\|_p}{N^p} \propto \boldsymbol{\ell}_c, \tag{3.4}$$

where $\bigtriangledown$ denotes the gradient magnitude of the Gaussian blurred version of $\mathbf{I}_c$ with standard deviation σ_b. Similarly to the weighted gray world, the weighted gray edges (wGE) algorithm [139] assigns weights to the image's edges based on their photometric properties (i.e., shadow edges, material edges, etc.) to improve the accuracy.

In contrast, Cheng *et al.* [77] showed that relying only on the color distribution of the image is sufficient to estimate the illuminant color without the need for any spatial information in the image. This work shows that the reason behind the spatial-based methods' success is the ability of obtaining large color differences from edges in the image's scene content. Hence, they showed that the scene illuminant can be obtained from the vector which maximizes the variance of the projected dark and bright pixels into one dimension (i.e., the first PCA vector).

The advantages of the statistical-based methods can be summarized as follows: (i) simplicity, (ii) speed, and (iii) few number of parameters—mostly less than three parameters that are usually fine-tuned for each camera model.

Despite the considerable merits of the statistical-based methods, their results are not always satisfactory. Zakizadeh *et al.* [396] experimentally showed that most statistical-based methods have a systematic failure due to the reliance on the statistics of scene content. In particular, they showed that there are certain types of images that are consistently difficult for different statistical-based methods including GW, max-RGB, SoG, and GE. There are a few attempts to improve the accuracy of such statistical-based methods by introducing a post-correction transformation to correct the "bias" error produced by these methods (e.g., [23, 111, 112]). However, such methods requires a large set of training data with corresponding ground-truth illuminant colors to *learn* the bias-correction function. With the reliance on labeled training data, we can think of these post-correction methods as a learning-based mechanism. We will discuss prior learning-based CC methods later in this chapter.

Physics-Based Methods

Physics-based methods usually depend on a more complex model than the Lambertian model (described in Eq. 3.2) to estimate the illumination vector based on the physical interaction between the illumination source and the scene's surfaces. According to the dichromatic reflection model [334], also known as neutral interface reflection assumption, the value of each color channel $c = \{\mathrm{R}, \mathrm{G}, \mathrm{B}\}$ for a pixel located at x in $\mathbf{I}$ is given by two components, as described in the following equation:

$$\mathbf{I}_c(x) = \int_\gamma m_b(x)\rho(x,\lambda)R(x,\lambda)S_c(\lambda)d\lambda + m_s(x)\rho(x,\lambda)R^{'}(x,\lambda)S_c(\lambda)d\lambda, \qquad (3.5)$$

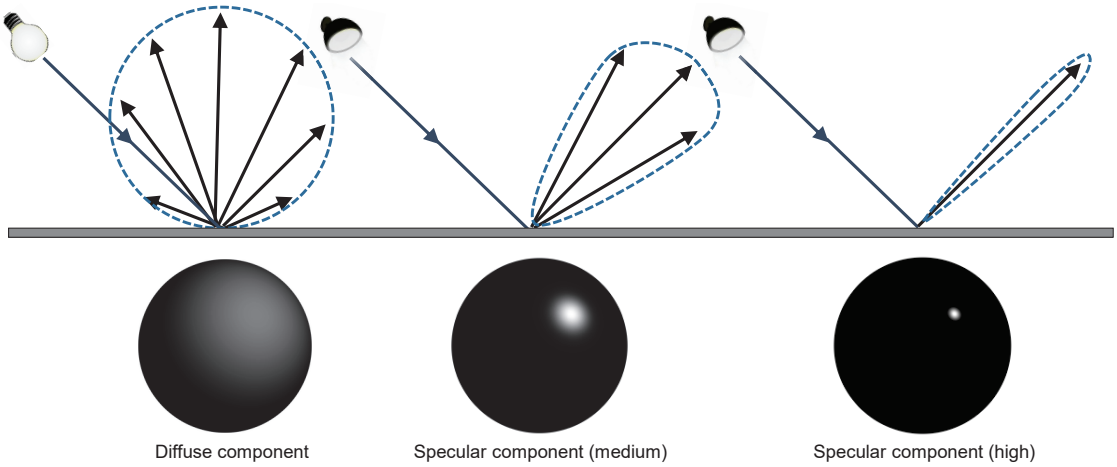


Figure 3.2: Lambertian model [46] vs. the dichromatic reflection model [334]. Lambertian model considers only the diffuse component of the object, while the dichromatic reflection model considers both the specular and diffuse components.

where $m_b(\cdot)$ and $m_s(\cdot)$ are independent scale factors depending on: (i) the angle of the viewpoint, (ii) direction of the light source, (iii) and surface orientation, and $R^{'}(\cdot)$ is the surface specular reflectance. Figure 3.2 illustrates the diffuse reflectance component, $R(\cdot)$, and specular reflectance component, $R^{'}(\cdot)$.

Based on this model, the color of specular highlights is the best cue for the illuminant color in the scene. The simplest approach is to estimate the pixel values at $m_b(\cdot) = 0$ (i.e., specular regions should be brighter than non-specular regions in the image). This gives us the max-RGB algorithm [58]. This simple assumption, however, is not accurate enough to estimate the specular pixels. Tan *et al.* [356] found that the correlation between the pixel intensities and chromaticity is not linear for the specular component. Thus, the specular pixels can be represented by a curve in the chromaticity-intensity space. On the other hand, the diffuse pixels construct a vertical line (i.e., the diffuse chromaticity and the total image intensity values are independent).

Li and Lee [232] proposed to search through a predefined set of light sources in order to undo the effect of the specular component. This cancellation condition can be satisfied if the specular component is projected on a perpendicular plane to the scene light source. To that end, they optimized the following minimization function:

$$\arg\min_{\iota,\varrho} \left(\texttt{Area}\left(\iota, \varrho\right) \omega\left(\iota, \varrho\right) \right), \tag{3.6}$$

where $\iota = G/R$ and $\varrho = G/B$, $\texttt{Area}(\cdot)$ refers to the projected area on the plane spanned by $\boldsymbol{\upsilon}_1\left(\iota, \varrho\right)$ and $\boldsymbol{\upsilon}_2\left(\iota, \varrho\right)$ vectors, and $\omega(\cdot)$ is a learned bias function that maximizes the corrected image with a low perceptual error from the ground truth image. The perceptual error was calculated using $\triangle$E, where $\triangle$E is defined as the Euclidean norm between the CIELAB color value of each pixel in the corrected image and the CIELAB color value of the corresponding pixel in the ground truth image.

Although these methods are assumed, in theory, to be more accurate, they are difficult to computed in practice [167].

Learning-Based Methods

Learning-based methods rely on training data with examples where the "true" scene illumination is known (e.g., by placing a neutral object in the scene) and use various strategies to estimate or predict the illumination for new unseen images. In the following part of this section, we review examples from learning-based models, such as gamut mapping, probabilistic models, and deep learning models.

Gamut mapping depends on finding a prior knowledge to guide a fixed model for estimating a set of plausible scene illuminants. These illuminants are determined based on their ability of mapping the entire color distribution of the testing image inside the "canonical" gamut (i.e., the prior). This idea was first introduced by Forsyth [122], where it was found that the gamut of 180 color chips under white lighting conditions can be represented

as a convex hull in the color space representing the canonical gamut under the ideal light. The final estimated illuminant vector is generated from this set of plausible scene illuminants. Finlayson and Hordley [110] showed that the median of these candidate illuminants works better than averaging them.

An efficient modification of the gamut mapping idea can be implemented by defining a set of all possible illuminants and the associated gamut for each one to constrain the solution space. The main idea lies in the fact that the solution space is bounded in terms of illuminant chromaticity, and can be represented by a finite set of chromaticity gamuts. Given the input image's gamut, a brute-force search through the predefined chromaticity gamuts can be used to solve the problem. This is how Finlayson *et al.* [117] proposed a gamut-based constrained solution by minimizing the error between the chromaticity of the given image and the predefined chromaticity gamuts in order to estimate the illuminant in the scene.

Similar to the physical-based approaches, Bianco and Schettini [55] relied on physical characteristics of skin colors to guide their method to estimate the illuminant vector of a given raw-RGB image. They found that skin colors can be clustered in the color space. Specifically, the skin "canonical" gamut can be computed from measured samples of skin tones in $\mathrm{Y}^{'}\mathrm{CbCr}$ color space with a roughly constant value of luma. In their experiments, Bianco and Schettini used 697 samples with $\mu(\mathrm{Y}^{'}) = 0.5$. This skin canonical gamut is represented as a convex hull of these skin samples. Based on this new canonical gamut, the feasible solution is the illuminant vector that can completely map the gamut of the given image's skin pixels inside the skin canonical gamut.

Instead of relying on non-parametric models, a more efficacious way can treat the problem as a *probability-based* problem to find the most likely illuminant parameters given the observed data (i.e., the given raw-RGB image's colors). This can be implemented by

computing the posterior probability for each illuminant vector $\hat{\boldsymbol{\ell}}_i$ using Bayesian estimation, such that:

$$\mathtt{p}(\hat{\boldsymbol{\ell}}_i|\mathbf{I}) \propto \left|\mathtt{diag}(\hat{\boldsymbol{\ell}}_i)^{-1}\right|^N \mathtt{p}(\mathtt{diag}(\hat{\boldsymbol{\ell}}_i)^{-1}\ \mathbf{I})\mathtt{p}(\hat{\boldsymbol{\ell}}_i), \tag{3.7}$$

where $\mathtt{p}(\cdot)$ denotes the probability, $\mathtt{diag}(\hat{\boldsymbol{\ell}}_i)^{-1}$ is the correction matrix that reconstructs the image without the effect of $\hat{\boldsymbol{\ell}}_i$[1], and $\mathbf{I}$ is represented as $3 \times N$ matrix, where N is the total number of pixels in the given raw-RGB image. In Eq. 3.7, the determinant term $\left|\mathtt{diag}(\hat{\boldsymbol{\ell}}_i)^{-1}\right|^N = \prod_c 1/\boldsymbol{\ell}_i(c)^N$ is used for normalization [325].

By assuming that the probability distribution of the illuminant $\mathtt{p}(\hat{\boldsymbol{\ell}}_i)$ is a uniform distribution (i.e., constant), the posterior probability can be computed using maximum likelihood estimation (e.g., [116, 324]).

To define the prior of the object's reflectances (i.e., $\mathbf{R} = \mathtt{diag}(\boldsymbol{\ell})^{-1}\ \mathbf{I}$), and $\hat{\boldsymbol{\ell}}_i$ in the case of the Bayesian-based models, different approaches were adapted. Brainard et al. [57], for instance, assumed that the prior of the object reflectances can be represented by a Gaussian distribution. Another solution proposes to use the estimated illuminants of other algorithms (e.g., statistical-based methods) as a proxy for the ground truth illuminants to get the color distribution under a "white" illumination [324, 325]. For the sake of accuracy, Gehler *et al.* [132] obtained more precise priors by collecting a set of raw-RGB images with an achromatic surface as a reference object.

Since then, many raw-RGB images have been publicly available, and consequently, more accurate learning-based algorithms were presented. For instance, Cheng *et al.* [79] proposed a fast framework that comprises a bank of regression trees to rectify the initial estimation of four different statistical-based methods (GW, max-RGB, histogram-based color palette, and histogram-based dominant color) followed by computing the median of

[1]To simplify, we represent the correction matrix using the inverse of the diagonal matrix, but more details are given in Sec. 3.1.2

these corrected illuminant vectors. Gijsenij *et al.* [135] suggested incorporating semantic information, represented by the edge distributions, in order to select the proper statistical-based method. During the training stage, they clustered the training data into groups and evaluate different statistical-based methods on each cluster. When testing, the given image is assigned to the closest cluster and the best illuminant estimation method is used.

Recently, several researchers have introduced convolutional neural network (CNN)-based solutions to solve the problem, due to the impressive results obtained using the *deep neural neural networks* (DNNs) in many computer vision problems [92]. Even before the evolution of DNNs in the recent years, there were a few attempts of training shallow networks in order to estimate the scene illuminant (e.g., one hidden layer was trained by Funt *et al.* [125] and two hidden layers were used by Cardei *et al.* [65]).

As the CC problem usually is posed as a regression problem, the majority of the CNNs models were trained to predict the parameters of the global illuminant vector (e.g., [44, 171, 248]). There are a few methods (e.g., [289]) that approximates the solution as a classification problem to benefit from the well-established CNN-based frameworks for image classification tasks.

The straightforward adaptation of the existing CNN architectures can be performed by fine-tuning one of the pre-trained models (e.g., the pre-trained AlexNet [209] using ImageNet dataset [91]) after replacing the last fully connected (fc) layer with a new fc layer (e.g., a new fc with three neurons, each of which represents a color channel value of the estimated illuminant color). Lou *et al.* [248] suggested to tackle the problem using the deep learning power by feeding the network a set of images with the corresponding ground truth illuminant vectors. They used `L2` loss function to fine-tune their AlexNet-based network outperforming several of the previous statistical-based methods.

Seoung and Kim [289] approximated the solution space by a set of clustered illumi-

nant vectors to derive benefit from the CNN models for the image classification problem. They adopted the AlexNet architecture by fine-tuning the network's weights to classify the given raw-RGB image based on these clusters. The final estimated illuminant vector is represented by a weighted summation of the cluster centers using the score of the softmax layer.

Finlayson and Hordley [115] showed that the illuminant vector $\boldsymbol{\ell} \in \mathbb{R}^3$ can be represented by two component in the uv log-chrominance space in which the problem of estimating $\boldsymbol{\ell}$ can be reformulated by two unknown values instead of three. Thus, Barron [44] represented the raw-RGB image by a 2D histogram of its log-chrominance components to treat the problem as a localization problem (correcting the image's colors now can be representing by translating the 2D histogram in the log-chrominance space). Based on this idea, Barron and Tasi [45] later proposed to detect the illuminant "location" in the histogram space through a learnable convolutional filter in the frequency domain (read Chapter 5 for more details).

Unlike Barron's methods [44, 45] which depend only on the histogram feature, Shi *et al.* [338] relied on $\mathbf{I}_{\text{u}}$ and $\mathbf{I}_{\text{v}}$ in order to train their CNN for a regression/classification task. They designed a novel architecture, called deep specialized network (DS-Net), consisting of two interacting networks—one network for regression and the second one for classification. The first network, they called it hypotheses network (HypNet), was designed to estimate two hypotheses of the illuminant vector in the UV space with two output branches: branch (A) and branch (B). Each branch estimates an illuminant vector; we denote them as $\hat{\boldsymbol{\ell}}_{\text{A}}$ and $\hat{\boldsymbol{\ell}}_{\text{B}}$. The selection of the best candidate is performed by the second network called selection network (SelNet). Such network is responsible for selecting the best candidate from the suggested illuminant vectors produced by HypNet (i.e., classifying the estimated illuminant to pick the best). This network is trained separately after training HypNet,

where it receives $\mathbf{I}_{\mathrm{u}}$, $\mathbf{I}_{\mathrm{v}}$, $\hat{\boldsymbol{\ell}_{\mathrm{A}}}$, and $\hat{\boldsymbol{\ell}_{\mathrm{B}}}$. Due to the huge number of the networks' parameters, DS-Net was designed to accept only 44×44 patches. The final response is generated by applying median pooling on the local illuminant vectors estimated for the image's patches.

Similar to DS-Net [338], Bianco *et al.* [54] designed a three-stage framework. At the first stage, the given raw-RGB image is divided into a grid of 32×32 patches to predict the local illuminant vector using a CNN model. The second stage was designed to classify the captured scene as a multi-illuminant scene or a single-illuminant scene. At that point, the angular error between each pair of the local illuminant vectors is computed followed by a thresholding process to determine whether the scene has a single illuminant or more. In the case of a global illuminant, the third stage contains a support vector machine for regression (SVR) model, with RBF kernel, which was trained based on the angular error between the response and the ground truth illuminant vectors.

A major challenge of such patch-based methods is determining the usefulness of these patches—some local patches reflect useful information about the scene illuminant, while others do not. Hu *et al.* [171] suggested to estimate a feature map, they referred to it as "the confidence map", that can be learned from the semantic context of the local patches to know which patch is more reliable than others. To be able to learn such confidence maps, they proposed a fully CNN that estimates a four-channel output, such that the first three channels represent the downscaled pixel-wise local estimated illuminants, while the last channel represents the confidence weights of this patch. This confidence map is used to produce the weighted estimated local illuminants. They used the angular error as a loss function that penalizes the network based on the angle between the aggregated weighted local illuminants and the ground truth illuminant. We can interpret their method as a local weighted GW with learnable weights (i.e., the confidence map) and learnable local illuminants. They found that objects with a bounded range of innate colors (e.g.,

faces) have higher confidence weights compared to other objects. Interestingly, this finding matches the previous work in [55] that relies on faces as a cue to estimate the scene illuminant.

Noticeably, adopting CNN architectures that had been originally designed for other tasks is commonly used for the illuminant estimation problem (for example, AlexNet was used by [171, 248, 289]). Such architectures were basically designed to filter out the unnecessary information from the given image producing a strong feature vector (i.e., deep features). Typically, this deep feature extraction process is carried out through a stacked set of traditional conv filters followed by fc layers with learnable weights. Then, a regression/classification model is fed by these deep features in order to tackle a certain problem. One of the important reasons for adopting this conv-based architecture is to effectively extract structural features from the spatial information of the image.

3.1.2 Chromatic Adaptation Transform

Chromatic adaption transform (CAT) is an essential process in color balancing to map image colors, captured under scene illumination source, to the corresponding colors under a different illumination source. Usually, CAT is employed for WB correction to eliminate undesirable color casts, so that neutral objects that perceptually appear white in reality are rendered white (i.e., R=G=B) in the final output image regardless the lighting conditions. According to the von Kries coefficient law [107], transforming the color response under one illuminant to another can be achieved using a simple scaling operation. In other words, a 3×3 diagonal matrix is sufficient to normalize the illumination's colors by mapping them to the achromatic line in the camera's raw-RGB color space [78, 191]. Here, the von Kries coefficient law is applied in the raw-RGB color space. We refer to it as the *standard*

approach that can be represented by the following equation [137]:

$$\mathbf{I}_{\text{corr}} = \texttt{diag}\left(\boldsymbol{\ell}^{*}\right)\ \mathbf{I}_{\text{in}}, \tag{3.8}$$

where $\mathbf{I}_{\text{in}}$ and $\mathbf{I}_{\text{corr}}$ are the raw-RGB input and corrected images represented as a $3\times N$ matrices, respectively, and $\texttt{diag}\left(\boldsymbol{\ell}^{*}\right)$ is a 3×3 diagonal matrix constructed as follows:

$$\texttt{diag}\left(\boldsymbol{\ell}^{*}\right) = \texttt{diag}\left(\left[\frac{\hat{\boldsymbol{\ell}}_{(\texttt{G})}}{\hat{\boldsymbol{\ell}}_{(\texttt{R})}}, \frac{\hat{\boldsymbol{\ell}}_{(\texttt{G})}}{\hat{\boldsymbol{\ell}}_{(\texttt{G})}}, \frac{\hat{\boldsymbol{\ell}}_{(\texttt{G})}}{\hat{\boldsymbol{\ell}}_{(\texttt{B})}}\right]\right), \tag{3.9}$$

where $\hat{\boldsymbol{\ell}}$ is the estimated illuminant vector. If the color space of $\mathbf{I}_{\text{in}}$ is the CIE XYZ space, this approach is referred as XYZ scaling [293]. It is worth noting that the best solution using this method can be obtained if $\hat{\boldsymbol{\ell}}$ is defined manually by picking a known neutral color in the scene (e.g., achromatic colors in a color rendition chart).

The modern models, such as von Kries transform [107], apply WB correction in post-adaptation cone responses related to biological vision (i.e., tristimulus responses of the L, M, S cone in the human eye [346]). Note that adopting von Kries coefficient law in such new spaces is referred as "wrong von Kries" [107, 347].

In particular, these models transform the colors from the CIE XYZ space into another space in which the diagonal model is assumed to work better—these models assume that the WB correction can be performed in either the CIE XYZ space or a transformed space from the CIE XYZ space, which is not the case in the existing camera imaging pipelines. According to these models, the correction process can be represented as [50, 282]:

$$\mathbf{I}_{\text{corr(XYZ)}} = \left(\mathbf{E}^{-1}\texttt{diag}\left(\boldsymbol{\ell}^{**}\right)\ \mathbf{E}\right)\ \mathbf{I}_{\text{in(XYZ)}}, \tag{3.10}$$

where $\mathbf{I}_{\text{in(XYZ)}}$ and $\mathbf{I}_{\text{corr(XYZ)}}$ are the input and corrected images in the CIE XYZ space [347], respectively, $\mathbf{E}$ is a nonsingular 3×3 CAT matrix, and $\texttt{diag}\left(\boldsymbol{\ell}^{**}\right)$ is a diagonal matrix containing the scaling factors. The scaling vector $\boldsymbol{\ell}^{**}$ is given by the following equations:

Table 3.1: Examples of chromatic adaption transform (CAT) models.

CAT model	**CAT entries**
XYZ scaling	$\begin{bmatrix} 1.00000 & 0.00000 & 0.00000 \\ 0.00000 & 1.00000 & 0.00000 \\ 0.00000 & 0.00000 & 1.00000 \end{bmatrix}$
von Kries [107]	$\begin{bmatrix} 0.40024 & 0.70760 & -0.08081 \\ -0.22630 & 1.16532 & 0.04570 \\ 0.0000 & 0.0000 & 0.91822 \end{bmatrix}$
Bradford [219]	$\begin{bmatrix} 0.89510 & 0.26640 & -0.16140 \\ -0.75020 & 1.71350 & 0.03670 \\ 0.03890 & -0.06850 & 1.02960 \end{bmatrix}$
Sharp [113, 119]	$\begin{bmatrix} 1.26940 & -0.09880 & -0.17060 \\ -0.83640 & 1.80060 & 0.03570 \\ 0.02970 & -0.03150 & 1.00180 \end{bmatrix}$
CMCCAT2000 [228]	$\begin{bmatrix} 0.79820 & 0.33890 & -0.13710 \\ -0.59180 & 1.55120 & 0.04060 \\ 0.00080 & 0.23900 & 0.97530 \end{bmatrix}$

$$\boldsymbol{\ell}^{**} = \left[\frac{\boldsymbol{\ell}_{\mathtt{r(R)}}}{\boldsymbol{\ell}'_{\mathtt{e(R)}}}, \frac{\boldsymbol{\ell}_{\mathtt{r(G)}}}{\boldsymbol{\ell}'_{\mathtt{e(G)}}}, \frac{\boldsymbol{\ell}_{\mathtt{r(B)}}}{\boldsymbol{\ell}'_{\mathtt{e(B)}}}\right], \tag{3.11}$$

$$[\boldsymbol{\ell}_{\mathtt{r(R)}}, \boldsymbol{\ell}_{\mathtt{r(G)}}, \boldsymbol{\ell}_{\mathtt{r(B)}}] = [\boldsymbol{\ell}_{\mathtt{r(X)}}, \boldsymbol{\ell}_{\mathtt{r(Y)}}, \boldsymbol{\ell}_{\mathtt{r(Z)}}] \quad \mathbf{E}^T,$$

$$\left[\boldsymbol{\ell}'_{\mathtt{e(R)}}, \boldsymbol{\ell}'_{\mathtt{e(G)}}, \boldsymbol{\ell}'_{\mathtt{e(B)}}\right] = \left[\hat{\boldsymbol{\ell}}_{\mathtt{(X/Y)}}, 1, \hat{\boldsymbol{\ell}}_{\mathtt{(Z/Y)}}\right] \quad \mathbf{E}^T,$$

where $\boldsymbol{\ell}_{\mathtt{r(XYZ)}}$ is a 1×3 row vector of a standard illuminant in CIE XYZ space (e.g., CIE standard illuminant D65).

Table 3.1 shows examples of CAT matrices proposed in the literature. Von Kries transform [107], for example, is based on assuming the independent gain control of the LMS cone responses, and as a consequence, the XYZ scaling is based on the ratio of the LMS cone responses of the illumination sources, where $\mathbf{E}_{\text{vonKries}}$ is defined to convert the CIE XYZ values to the corresponding values in the LMS. This conversion is described in the following equation:

$$\begin{bmatrix} \mathtt{L} \\ \mathtt{M} \\ \mathtt{S} \end{bmatrix} = \begin{bmatrix} 0.40024 & 0.70760 & -0.08081 \\ -0.22630 & 1.16532 & 0.04570 \\ 0.0000 & 0.0000 & 0.91822 \end{bmatrix} \begin{bmatrix} \mathtt{X} \\ \mathtt{Y} \\ \mathtt{Z} \end{bmatrix}. \tag{3.12}$$

Note that the entries of $\mathbf{E}_{\text{vonKries}}$ in Eq. 3.12 are normalized to CIE standard illuminant D65.

Usually, advanced CAT matrices (e.g., CMCCAT2000 [228]) were derived based on minimizing the error of illumination mapping (e.g., from CIE C to CIE D65) using the corresponding-color datasets [347]. Such corresponding-color datasets include pairs of CIE XYZ color tristimulus values under two different illumination sources based on physical stimulus [253, 290, 310]. Figure 3.3 shows an example of a corresponding-color dataset.

The Bradford transform [219] is a widely used CAT matrix. The Bradford transform was derived empirically using a set of corresponding-colors of 58 dyed wool samples with different CC under CIE A and CIE D65.

Another CAT matrix is the Sharp adaption transform [113]. The idea of the Sharp transform is based on the narrow cone space of the Bradford sensors—namely, the linear combination of the XYZs after applying the Bradford transform. Bradford sensors are

more de-correlated and represented by a narrowed cone space compared to the relative LMS cone responses, as shown in Fig. 3.4. Accordingly, Finlayson *et al.* [113, 119] found that there is a potential improvement in the performance of CAT models, if the data is pre-processed by applying Sharp transform $\mathbf{E}_{sharp}$, such that:

$$\mathbf{C}_{\boldsymbol{\ell}_1} = \mathbf{C}_{\boldsymbol{\ell}_2}\mathbf{E}_{\mathtt{sharp}}\mathtt{diag}\left(\boldsymbol{\ell}^{**}\right), \tag{3.13}$$

where $\mathbf{C}_{\boldsymbol{\ell}_1}$ and $\mathbf{C}_{\boldsymbol{\ell}_2}$ are $n\times 3$ matrices of the CIE XYZ values under the first illumination $\boldsymbol{\ell}_1$ and second illumination $\boldsymbol{\ell}_2$, respectively. The sharpening transform can be obtained through eigenvector decomposition of the matrix $\mathbf{S}$ given in the following equation:

$$\mathbf{S} = (\mathbf{C}_{\boldsymbol{\ell}_2}^T\mathbf{C}_{\boldsymbol{\ell}_2})^{-1}\mathbf{C}_{\boldsymbol{\ell}_2}^T\mathbf{C}_{\boldsymbol{\ell}_1}, \tag{3.14}$$

$$\mathbf{S} = \mathbf{E}_{\mathtt{sharp}}\ \mathtt{diag}\left(\boldsymbol{\ell}^{**}\right)\ \mathbf{E}_{\mathtt{sharp}}^{-1}. \tag{3.15}$$

Another example of CAT matrices is CMCCAT2000, which is derived using an iterative optimization process to minimize the error between predicted and observed colors over a set of eight color datasets in the CIELAB color space [228].

Several studies in the literature were carried out to evaluate different CATs [292, 347, 348]. These studies aimed to find the best CAT that obtains the smallest color differences between transformed source and target color values.

To the best of our knowledge, there is no clear criterion for what the best CAT is. For example, Sharp adaptation transform outperforms Bradford transform using particular datasets, while Bradford performs better in another set of color pairs [119]. Süsstrunk and Finlayson [347] found that CMCCAT2000 outperforms the Sharp transform. In another experiment, it is found that Bradford transform is exhibited the lowest error compared to von Kries, XYZ scaling, and CMCCAT2000 using 8,190 color patches containing different pairs of illuminants (D50-A, D50-D65, and D65-A) [292].

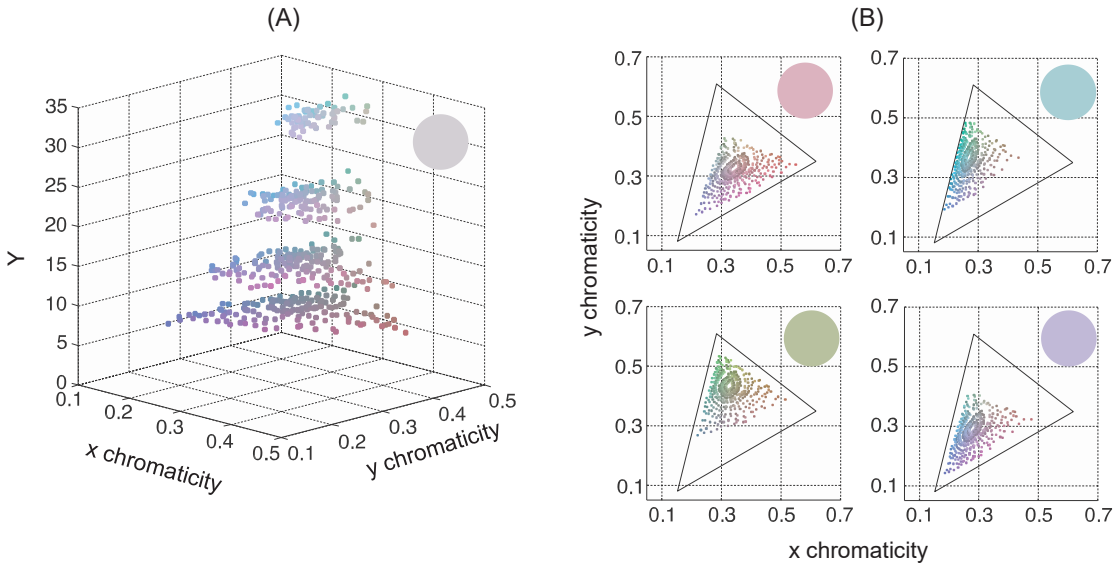


Figure 3.3: Example of Munsell chip collection, adapted from [291]. (A) Munsell chips under a neutral illuminant in the CIE xyY space. Each symbol's color represents the reflected color of each chip. (B) shows the projected chip chromaticities under four illuminations.

Note that these CAT models are applied to the image's CIE XYZ values. Computing the CIE XYZ values from the raw-RGB image, however, is performed through a 3×3 full CST which is computed based on an accurate predetermination of the correct scene illuminant value and color temperature; meaning that in order to get correct CIE XYZ values, the scene illuminant should be determined first—a chicken-and-egg problem. Thus, existing camera imagining pipelines apply the WB correction to the raw-RGB image (i.e., the standard approach) before converting it to the CIE XYZ space [192].

In the case of multi-illuminant scenes, Yang and Shevell's study [387] shows that in the case of two different illuminants, CC is improved if only the specular highlight cues of both illuminants are consistent. They found, however, that CC is reduced if the scene's objects have cues from two distinct illuminants. Accordingly, the chromatic adaption methods

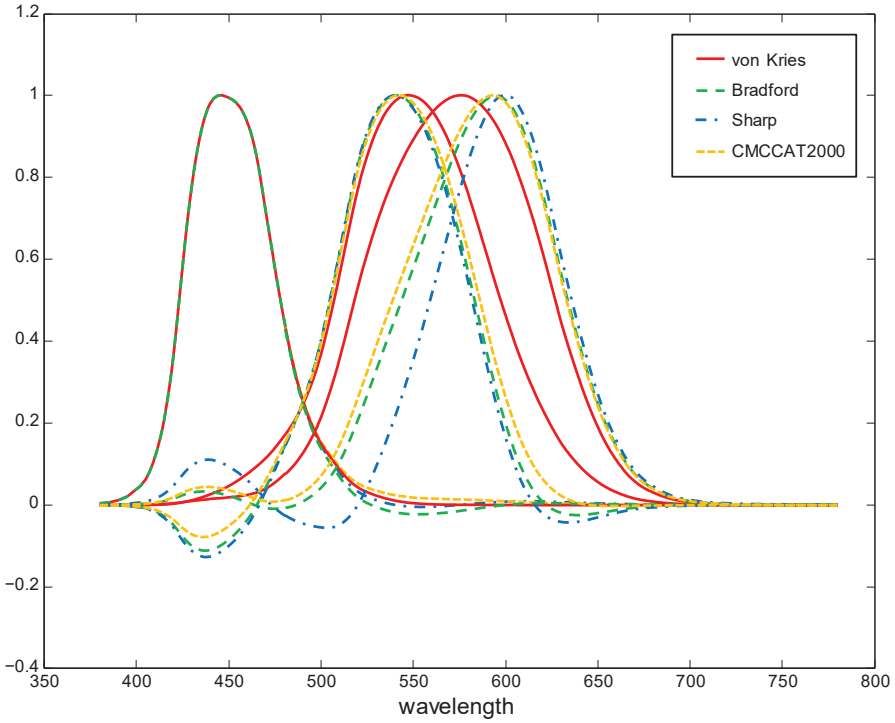


Figure 3.4: Normalized von Kries [107], Bradford [219], Sharp [119] and CMCCAT2000 [228] sensors. This figure is adapted from [119].

mostly assume a single light source (i.e., correct only the dominant light source).

In the literature, considering multiple light sources is performed in two different ways. The first approach is to correct the raw-RGB image for each estimated illuminant separately, then applying a blending post-processing process to produce the final "corrected" image. This approach was adopted by Cheng *et al.* [76] who assessed the user preference and found that "warmer" (reddish) results are more preferable for the outdoor-illuminant scenes captured under two distinct illuminants (i.e., warm and cold tones). They achieved that by blending two images, each of which is corrected using the standard approach, with a blending factor for the "cold" image tone $\in \{0.25, 0.5\}$.

The second approach is applying a pixel-wise diagonal-based correction for multi-

illuminant scenes. This pixel-wise estimation is mostly approximated by patch-wise estimation, followed by post-processing procedures to get approximated pixel-wise illuminant vectors [48, 174]. For instance, Hsu *et al.* [169] proposed a method to correct images with two mixed illuminants ($\boldsymbol{\ell}_{(1)}$ and $\boldsymbol{\ell}_{(2)}$) by assuming that the illuminant vectors are specified by the user and the captured scene has a small number of material colors (i.e., a sparse set of colors). According to their case, the image formation can be expressed by the following equation:

$$\mathbf{I}_c(x) = \mathbf{R}_c \times (\beta_1 \boldsymbol{\ell}_{(1)c} + \beta_2 \boldsymbol{\ell}_{(2)c}), \tag{3.16}$$

where β_1 and β_2 are unknown scalar factors representing the influence on the pixel located at x of light sources $\boldsymbol{\ell}_{(1)}$ and $\boldsymbol{\ell}_{(2)}$, respectively. Thus, the 3×3 diagonal correction matrix's elements $\boldsymbol{\ell}^*$ can be expressed as follows:

$$\boldsymbol{\ell}_c^* = \frac{\beta_1 + \beta_2}{\beta_1 \boldsymbol{\ell}_{(1)c} + \beta_2 \boldsymbol{\ell}_{(2)c}} = \frac{1}{\beta * \boldsymbol{\ell}_{(1)c} + (1 - \beta *) \boldsymbol{\ell}_{(2)c}}, \text{ with } \quad \beta * = \frac{\beta_1}{\beta_1 + \beta_2}. \tag{3.17}$$

To estimate the value of $\beta*$, they first estimate potential material colors in the captured scene by adopting a greedy voting approach. This voting approach works in a 32×32 projected chromaticity space (i.e., R/B, B/R) with log spacing by assigning each pixel to the nearest bin. That is, given an estimated material color $\hat{\mathbf{R}}$ and the associated pixels to this material in $\mathbf{I}$, Eq. 3.16 can be solved. Despite the impressive results obtained by this method, it is constrained by specific conditions.

3.1.3 Research Directions

Following up on the main research areas discussed in this section, we can summarize promising research directions as: (i) lightweight learning-based CC and (ii) sensor-independent learning-based CC.

Table 3.2: Comparison between number of parameters required by examples of statistical-based and CNN-based methods.

Method	**Statistical-based methods**				**CNN-based methods**				
	GW [60]	SoG [120]	GE [362]	PCA [77]	DS-Net [338]	SCC [9]	AlexNet-FC4 [171]	DOCC [165]	Quasi U CC [51]
Number of parameters	0	2	2	1	∼17 millions	∼14 millions	∼4 millions	∼4 millions	∼80 millions

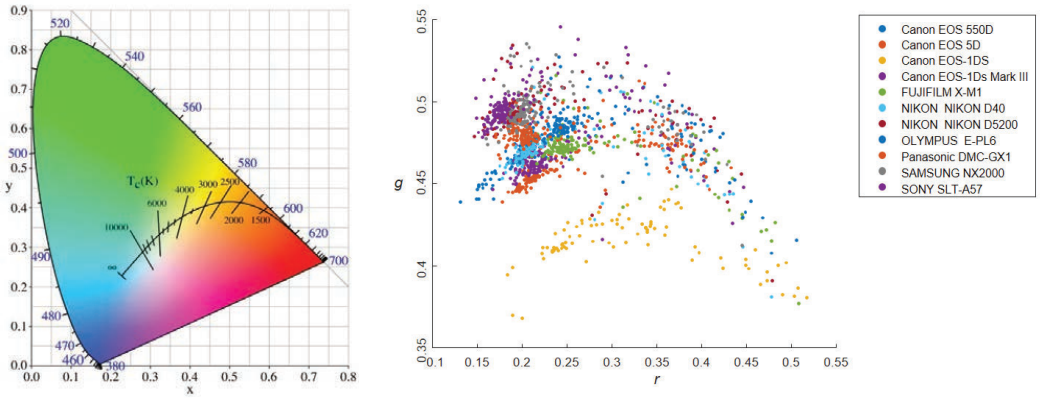
Lightweight Learning-Based CC

As discussed earlier, statistical-based illuminant estimation methods operate using statistics from an image's color distribution and spatial layout to estimate the scene illuminant. These methods are fast and easy to implement; however, their results are not always satisfactory. On the other hand, learning-based methods rely on training data with examples where the illumination is known (e.g., by placing a neutral object in the scene) and use various strategies to estimate or predict the illumination. In recent years, learning-based methods employing deep-learning techniques have shown state-of-the-art performance. Learning-based methods, however, suffer from a substantial increase in complexity, with deep network architectures requiring millions of parameters (see Table 3.2).

In the absence of specialized chips or GPUs, the computational and memory requirements associated with running these methods onboard the camera are still prohibitive. As a result, cameras currently rely on simple statistical methods even though these methods are not as accurate as their learning-based counterparts. A promising research direction could include improving the accuracy of such simple statistical-based methods by learning a post-process enhancement mechanism.

Sensor-Independent Learning-Based CC

Learning-based illuminant estimation models outperform statistical-based methods by training sensor-specific models on training examples provided with the labeled images with ground-truth illumination. These training images are captured with the sensor make and model being trained. The obvious drawback of these methods is that they do not generalize well for arbitrary camera sensors without re-training/fine-tuning on samples captured by testing camera sensor. The reason behinds this is in Eq. 3.1, where the sensor's spectral sensitivity function has a direct contribution to the captured colors by cameras. Figure 3.5-(A) shows the black body locus (also called Planckian locus) in an ideal device-independent space chromaticity diagram (i.e., CIE xy 1931 chromaticity) for a wide range of temperatures. This Planckian curve, however, does not be represented similarly in different camera sensor spaces (see Fig. 3.5-[B]) due to the differences in the spectral sensitivity functions. A promising research direction, thus, is to develop a sensor-independent learning-based illuminant estimation method that is explicitly designed to generalize well for unseen camera sensors without the need to re-train/tune our model. One could think of mapping camera raw-RGB sensor responses to a perceptual color space. As discussed in Chapter 2, this process is applied onboard digital cameras to map the captured sensor-specific raw-RGB image to a standard device-independent "canonical" space (e.g., CIE XYZ) [191,316]. Usually this conversion is performed using a 3×3 matrix and requires an accurate estimation of the scene illuminant [64]. It is important to note that this mapping to CIE XYZ requires that white-balance procedure first be applied. As a result, it is not possible to use CIE XYZ as the canonical color space to perform illumination estimation. Work by Nguyen et al. [283] studied several transformations to map responses from a source camera sensor to a target camera sensor, instead of mapping to a perceptual space. In their study, a color rendition reference chart is captured by both source and target camera sensors in order to



(A) Planckian locus in the CIE xy 1931 chromaticity diagram

(B) Planckian locus in the *rg* chromaticity space of different camera sensors

Figure 3.5: This figure shows the ideal path that a black body color takes in a chromaticity space as the black body temperature changes (i.e., Planckian locus). (A) Planckian locus in the device-independent CIE xy 1931 chromaticity space. (B) Planckian locus in the rg chromaticity space of different camera sensors, where $r = \mathrm{R}/(\mathrm{R}+\mathrm{G}+\mathrm{B})$ and $g = \mathrm{G}/(\mathrm{R}+\mathrm{G}+\mathrm{B})$.

compute the raw-to-raw mapping function. Learning a mapping transformation between responses of two different sensors is also adapted in [127]. However, the work in [127, 283] has no mechanism to map an unseen sensor to a canonical working space without explicit calibration.

Recently, few-shot and multi-domain learning techniques [264, 380] have been proposed to reduce the effort of re-training camera-specific learned color constancy models. These methods require only a small set of labeled images for a new camera unseen during training. Another strategy has been proposed to white balance the input image with several illuminant color candidates and learn the likelihood of properly white-balanced images [163]. Such a Bayesian framework requires prior knowledge of the target camera model's illuminant colors to build the illuminant candidate set. Despite promising results, these methods

all require labeled training examples from the target camera model: raw images paired with ground-truth illuminant colors. As mentioned earlier, collecting such training examples is a tedious process, as certain conditions must be satisfied—i.e., for each image to have a single uniform lighting and a calibration object to be present in the scene [77].

3.2 Color Correction for Camera-Rendered Images

Following up our discussion on image white balancing, we now assume that the given image was rendered with an incorrect WB setting in the sRGB color space (see Fig. 3.6-[A] for example). As can be seen, the shown image has a strong color cast due to the WB error. As shown in Fig. 3.6-(A), there are two highlighted achromatic regions in the scene: (i) a patch from a white bridge and (ii) a neutral patch from the color rendition chart. The same scene was rendered to sRGB with the correct WB in Fig. 3.6-(I). As shown, because the WB was applied correctly in the proper space (i.e., raw-RGB), both the scene achromatic regions lie on the white line (i.e., R=G=B).

One solution to correct the colors of such images that were rendered with WB errors is by estimating the target color distribution of the given image using a trained CNN for relevant problems, such as image colorization. Although the plausible colors are produced by the recent CNN-based colorization methods (e.g., [405]), they usually consider only the spatial information, regardless of the input image's color distribution, in order to produce the output image. Consequently, the estimated colors are consistent and too far from the ground truth images regardless of the level of degradation of the given image's color distribution. Figure 3.7 shows the results obtained by employing a colorization method to correct camera-rendered images with WB errors. As can be seen, the results of colorization is promising in terms of colorizing a given image; however, they are still far from the ground truth images.

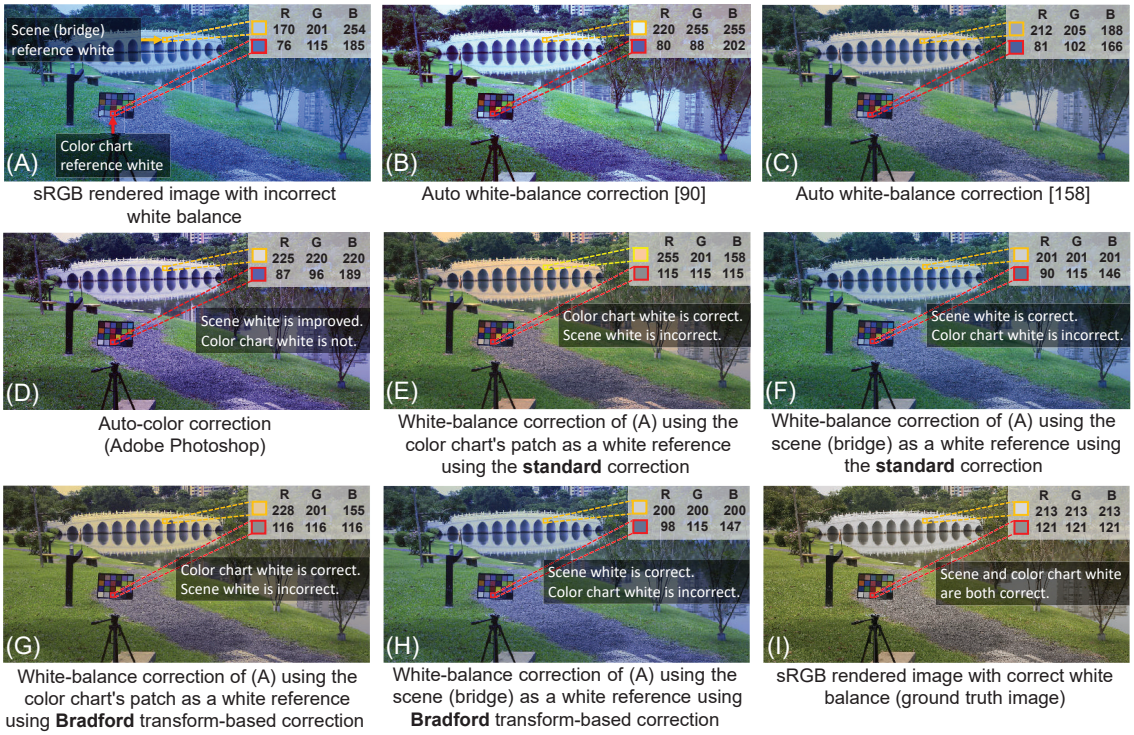


Figure 3.6: (A) An sRGB image rendered with an incorrect WB applied. There are two achromatic regions highlighted in red and yellow. (B) and (C) show results obtained using two different AWB algorithms [173, 352]. (D) shows the result of auto-color correction from Adobe Photoshop. (E) and (F) show standard WB correction applied to the sRGB-rendered image using different reference white points. (G) and (H) show the Bradford transform-based correction as described in Eq. 3.21 using the same reference white points. (I) Ground truth sRGB image with the correct WB applied.

Another possible direction is considering simple histogram-based operations, such as applying histogram stretching for each color channel [371]. The results of the stretching operation can be blended with the results of statistical-based methods to give room for improvements [352]. However, these simple operations are inadequate to deal with the high degradation resulting from the nonlinear camera imaging operations applied after an

Figure 3.7: (A) Input images with a wrong WB rendered in the sRGB color space. (B) Colorized images using Zhang *et al.*'s CNN-based method [405]. (C) Ground truth image with an accurate WB correction rendered in the sRGB color space. Note that the input for the colorization method was the L^* channel of the input image.

incorrect WB setting, as shown in Fig. 3.6-(B). Even the color correction functions provided by different commercial software packages cannot properly work with such cases. As an example, Adobe Photoshop has two functions for color correction—namely, auto-color and auto-tone. The auto-tone function automatically adjusts the black and white points of the given sRGB image's tone curve. The process includes clipping parts of the shadow and highlights, and mapping the darkest and lightest values of each color channel to pure black and white [1]. The auto-color function adjusts both the contrast and colors of the given sRGB image by searching through the image's color to identify shadows, midtones, and highlights in order to map the midtones to 128 gray levels followed by clipping the shadows and highlights by 0.5% [1]. Fig. 3.6-(D) shows the results of Adobe Photoshop's auto-color. As shown, the auto-color function reduces the color cast in Fig. 3.6-(A), but in the perspective of WB, the result still needs more improvement.

We can think of utilizing the rich amount of research done for illuminant estimation

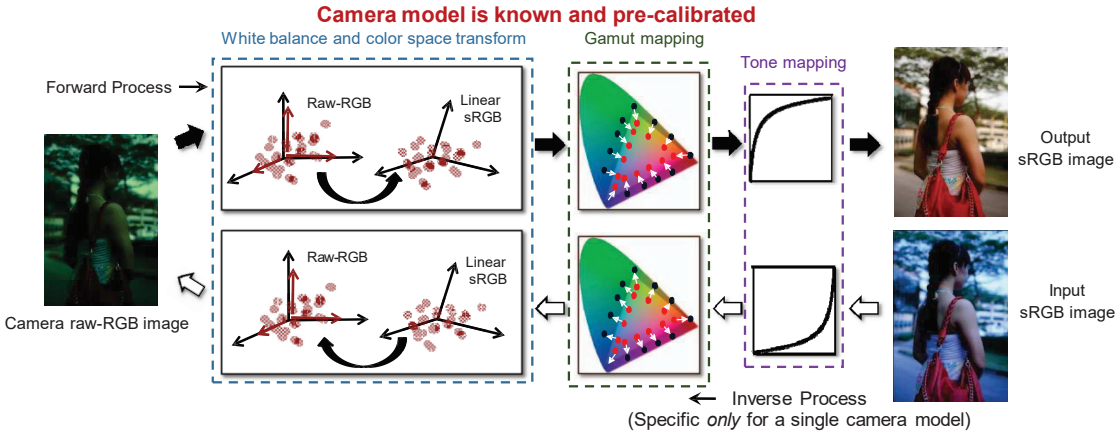


Figure 3.8: An accurate WB correction for sRGB-rendered images can be obtained with calibrated camera models, in which the full model of response (MoR) is defined and the original raw-RGB image reconstruction is applicable. This figure shows the inverse process of the in-camera pipeline after calibrating the required parameters and settings of the imaging process. This calibration is done by using training images captured by the same camera used to photograph the testing image. This figure is adapted from [202].

and color correction in the linear space by extending the existing methods to work in the sRGB color space. Here, we refer to the sRGB image as $\mathbf{I}$ instead of $\mathbf{I}_{\texttt{sRGB}}$ for simplicity. Mathematically, if the given image is rendered in the sRGB color space, the value of each color channel, according to the Lambertian model in Eq. 3.2, is now represented as

$$\mathbf{I}_c = f_{\text{CRF}}(\mathbf{R}_c \times \boldsymbol{\ell}_c). \tag{3.18}$$

Currently, the available accurate solutions to calculate f_{CRF} and f_{CRF}^{-1} require applying a serious radiometric calibration (e.g., [202]) or embedding metadata onboard cameras (e.g., [286]) to reconstruct the original raw-RGB image. After reconstruction, the diagonal WB correction can be applied, then the corrected image is converted back to the sRGB color space. Figure 3.8 shows an example of Kim *et al.*'s in-camera model to tackle the

problem of post color correction of improperly white-balanced rendered sRGB images. Although this solution gives accurate results, this process is impractical in many scenarios, as it requires calibrating the camera model used in photographing the image.

One possible solution is to use another color space, instead of the raw-RGB or sRGB color spaces, to apply the WB correction. For example, if we aim to apply the correction in the CIE XYZ color space, the problem now can be reformulated as

$$\mathbf{I}_c = f^{-1}(\mathbf{I}_{(XYZ)c} \times \boldsymbol{\ell}_c), \tag{3.19}$$

where $f : [\mathtt{R}, \mathtt{G}, \mathtt{B}] \rightarrow [\mathtt{X}, \mathtt{Y}, \mathtt{Z}]$ is a nonlinear function that "linearizes" the sRGB color triplet to get the corresponding CIE XYZ values [101]. We would emphasize that the f function is defined only if the image was rendered to the sRGB color space using the standardized sRGB that assumes the gamma value of 2.2 with no picture style applied [31]. Otherwise, f is undefined and a radiometric calibration process is required.

Here, most of the conventional radiometric calibration solutions are not applicable as a result of the absence of any information related to the camera model or the ability to re-capture several images for the same scene as required by most of the radiometric calibration methods. The simplest solution is to adopt the inverse of the gamma operation defined in the standardized sRGB conversion [31] as an approximation to the true nonlinear function applied on the image.

If we defined the f function as a single gamma operation, the standard approach for correction can be updated according to

$$\mathbf{I}_{\mathtt{corr}} = f^{-1}\left(\mathtt{diag}\left(\boldsymbol{\ell}^*\right) \;\; f\left(\mathbf{I}_{\text{in}}\right)\right). \tag{3.20}$$

Note that the values of the illuminant should also be in the CIE XYZ space by obtaining the illuminant vector after converting the sRGB image to the CIE XYZ color space.

Applying modern CAT models models can also be adjusted to deal with sRGB images according to

$$\mathbf{I}_{\texttt{corr}} = f^{-1}\left(\left(\mathbf{E}^{-1}\texttt{diag}\left(\boldsymbol{\ell}^{**}\right)\ \mathbf{E}\right)\ f\left(\mathbf{I}_{\text{in}}\right)\right). \tag{3.21}$$

Figure 3.6-(G) and (H) show results obtained by the Matlab function for WB correction using the Bradford transform [219] with the inverse of the gamma operations defined in [31]. There are noticeable differences between the corrected images and the ground truth image.

Now, we examine applying the standard diagonal approach for WB correction while intentionally ignoring the nonlinear color manipulations applied onboard cameras. As an example, Huo *et al.* [173] proposed to convert the sRGB image into the YUV space to correct its colors iteratively using the standard diagonal approach after estimating a set of estimated gray pixels at each iteration; see Fig. 3.6-(C).

Instead of estimating the illuminant vector, the best solution (i.e., the ground truth) can be obtained if $\hat{\boldsymbol{\ell}}$ is defined manually by picking a known neutral color in the scene. As shown in Fig. 3.6-(A), there are two reference white points in the scene: (i) a gray patch in the color rendition chart $p(1)$ and (ii) a white patch from the bridge in the scene $p(2)$. In Fig. 3.6-(E), we applied Eq. 3.8 directly to correct the input sRGB image in Fig. 3.6-(A) using the color rendition chart's patch as a reference white. By plugging the values of $\texttt{diag}\left(\boldsymbol{\ell}^{*}\right)$ into Eq. 3.8, we can correct the reference achromatic point $p(1)$ as follows:

$$p(1) = \begin{bmatrix} 1.5132 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.6216 \end{bmatrix} \begin{bmatrix} 76 \\ 115 \\ 185 \end{bmatrix} = \begin{bmatrix} 115 \\ 115 \\ 115 \end{bmatrix}. \tag{3.22}$$

However, applying the same diagonal correction matrix to the second reference white point $p(2)$ results in incorrect WB (i.e., $p(2) = [255, 201, 158]$). Figure 3.6-(F) shows

another attempt using the bridge scene region as a reference white. As exhibited, the same problem appears in the color rendition chart's reference point.

Based on this discussion, a practical possible solution for WB sRGB-rendered images taken by uncalibrated or unknown camera models is to adopt the current chromatic adaption methods with a single estimated illuminant vector with or without the gamma-based linearization process. This can lead in some cases to out-of-gamut colors, because such chromatic adaption methods are meant to work in linear spaces. As a consequence, applying them to the improperly white-balanced sRGB images usually leads to undesirable results—even with attempts to linearize the sRGB image using the simple inverse of the gamma operation, as shown in Fig. 3.9-(D), or one of the applicable radiometric calibration methods, as shown in Fig. 3.9-(E).

3.2.1 Research Directions

Based on the previous discussion, it is obvious that there is a need for a solution to correct image colors that were rendered with WB errors. Another research direction is to allow the user to interactively manipulate the WB settings in the post-capture stage (i.e., applying accurate chromatic adaptation to different color temperatures in the sRGB color space). This post-capture WB editing may act as a beneficial tool to satisfy different user preferences (which may not always match camera AWB correction). Editing the WB settings in camera-rendered images also allow the user to manipulate the colors of captured images of challenging scenes, such as multi-illuminant scenes, where camera raw diagonal WB correction cannot produce correct colors for the entire scene.

Figure 3.9: Example of applying the "linearization" process before the WB correction for sRGB images. (A) Input sRGB image is linearized using the simple inverse gamma operation [31] in (B) and the recent radiometric calibration for face images [229] in (C). The achromatic region in the provided color rendition chart (hidden after selecting the achromatic region manually) was used to correct images in (B) and (C) by applying WB correction using Bradford transform [219], as shown in (D) and (E), respectively. (F) Ground-truth image.

3.3 Exposure Errors in Camera-Rendered Images

As discussed in Chapter 1, exposure settings have a significant impact on the quality of the final rendered colors. This section of the thesis discusses potential exposure errors that may occur in the camera's ISP when capturing an image. We then present a brief review of related methods for correcting images rendered with exposure errors.

Photographic exposure refers to the amount of received light to camera sensor. The exposure used at capture time directly affects the overall brightness of the final rendered photograph. Digital cameras control exposure using three main factors: (i) capture shutter speed, (ii) f-number, which is the ratio of the focal length to the camera aperture diameter, and (iii) the ISO value to control the amplification factor of the received pixel signals. In photography, exposure settings are represented by exposure values (EVs), where each EV refers to different combinations of camera shutter speeds and f-numbers that result in the same exposure effect—also referred to as 'equivalent exposures' in photography.

Digital cameras can adjust the exposure value of captured images for the purpose of varying the brightness levels. This adjustment can be controlled manually by users or performed automatically in an auto-exposure (AE) mode. When AE is used, cameras adjust the EV to compensate for low/high levels of brightness in the captured scene using through-the-lens (TTL) metering that measures the amount of light received from the scene [301].

Exposure errors can occur due to several factors, such as errors in measurements of TTL metering, hard lighting conditions (e.g., very low lighting and backlighting), dramatic changes in the brightness level of the scene, and errors made by users in the manual mode. Such exposure errors are introduced early in the capture process and are thus hard to correct after rendering the final 8-bit image. This is due to the highly nonlinear operations

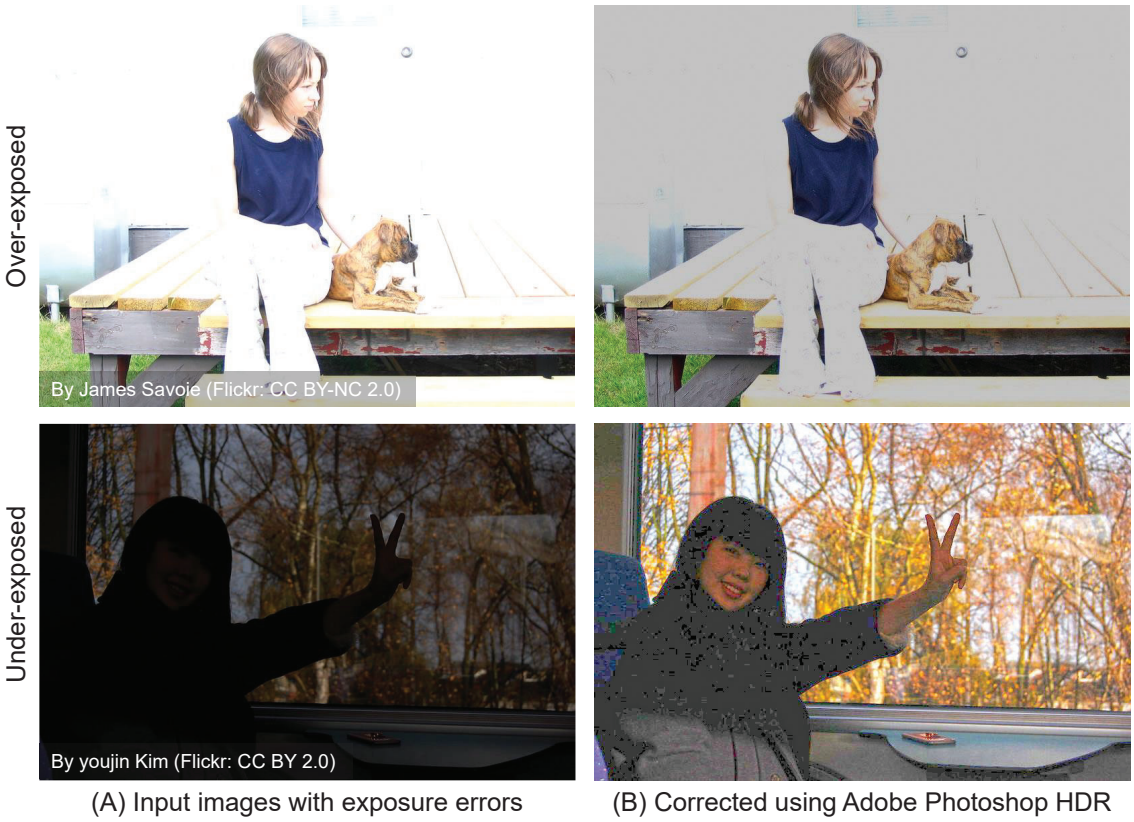


Figure 3.10: Examples of camera-rendered images with exposure errors. (A) Input images. (B) Corrected images using Adobe Photoshop HDR tool [88].

applied by the ISP afterwards to render the final 8-bit sRGB image [191].

Figure 3.10-(A) shows typical examples of images with exposure errors. In Fig. 3.10, exposure errors result in either very bright colors, due to overexposure, or very dark colors, caused by underexposure errors, in the final rendered images. Correcting images with such errors is a challenging task even for well-established image enhancement software packages, see Fig. 3.10-(B). Although both over- and underexposure errors are common in photography, most prior work is mainly focused on correcting underexposure errors [152, 369, 375, 404, 407] or generic image quality enhancement [75, 134].

As our focus is on correcting exposure errors in camera-rendered 8-bit per channel sRGB

images, we refer the reader to [72, 155, 170, 235] for representative examples for rendering linear raw-RGB images captured with low light or exposure errors.

3.3.1 Exposure Correction

Traditional methods for exposure correction and contrast enhancement rely on image histograms to re-balance image intensity values [68, 142, 223, 307, 411]. Alternatively, tone curve adjustment is used to correct images with exposure errors. This process is performed by relying either solely on input image information [395] or trained deep learning models [150, 272, 298, 394]. The majority of prior work adopts the Retinex theory [220] by assuming that improperly exposed images can be formulated as a pixel-wise multiplication of target images, captured with correct exposure settings, by illumination maps. Mathematically, these methods formulate the problem as follows:

$$\mathbf{I} = \mathbf{S} \odot \tilde{\mathbf{I}}, \tag{3.23}$$

where $\mathbf{I}$ is the sRGB camera-captured image, which was captured with some exposure errors or under low-lighting conditions, $\mathbf{S}$ is an unknown illumination map, and $\tilde{\mathbf{I}}$ is the reflectance image that was captured with correct exposure settings under normal-lighting conditions. Thus, the goal of most Retinex-based methods is to predict the illumination map, $\mathbf{S}$, to recover the well-exposed target images. Representative Retinex-based methods include [152,185,220,266,370,402,404] and the most recent deep learning ones [369,375,407]. Most of these methods, however, are restricted to correcting underexposure errors [152, 369, 375, 385, 388, 404, 407, 410] due to the fact that over-exposed images usually require introducing new content in the corrupted input images, which usually have missing contents as being over-exposed (see Fig. 3.10).

3.3.2 HDR Restoration and Image Enhancement

Over-exposed images can be corrected by restoring the high dynamic range (HDR) of the captured-image by reconstructing scene radiance HDR values from one or more low dynamic range (LDR) input images. Prior work either require access to multiple LDR images [106, 190, 265] or use a single LDR input image, which is converted to an HDR image by hallucinating missing information [104, 274]. We experimentally found that, however, this single-HDR reconstruction approach is not able to properly deal with images with over-exposure errors (experimental evaluations are given in Chapter 12).

Ultimately, these reconstructed HDR images are mapped back to LDR for perceptual visualization. This mapping can be directly performed from the input multi-LDR images [63, 90], the reconstructed HDR image [389], or directly from the single input LDR image without the need for radiance HDR reconstruction [75, 134]. There are also methods that focus on general image enhancement that can be applied to enhancing images with poor exposure. In particular, work by [175, 176] was developed primarily to enhance images captured on smartphone cameras by mapping captured images to appear as high-quality images captured by a DSLR.

3.3.3 Datasets

Paired datasets are crucial for supervised learning for image enhancement tasks. Existing paired datasets for exposure correction focus only on low-light underexposed images. Representative examples include Wang et al.'s dataset [369] and the low-light (LOL) paired dataset [375]. One could think of HDR datasets as an alternative option to train models for exposure correction. However, most of the available HDR datasets have a limited number of scenes that can negatively affect the generalization of deep learning models. For example, Funt et al.'s HDR dataset [126] has only 105 HDR scenes. A relatively large HDR

dataset is the HDR+ dataset [155], which has 3,640 bursts of artificially aligned linear raw-RGB images and the corresponding HDR "ground-truth" image. These images were *not* intentionally captured with exposure errors, similarly to image quality enhancement datasets, such as the DSLR Photo Enhancement dataset (DPED) [175]. Moreover, the ground-truth images in the HDR+ dataset were generated by Google Camera's HDR+ algorithm [134, 155]. Thus, it is also arguable that supervision training on this dataset would result in models that learn to mimic procedures applied in [134, 155], rather than learning the underlying mechanisms to correct exposure errors.

3.3.4 Research Directions

Both under- and over-exposure greatly affect the colors in the image and the overall visual appeal. The problem becomes more challenging when the image is rendered in 8-bit format by unknown camera model. A promising research direction towards enhancing colors in images captured with exposure errors is to consider an accurate image linearization process, that models the commonly applied camera pipeline procedures, to reconstruct scene-referred images. With an accurate image linearization, one could expect that low-image enhancement algorithms could achieve better results compared to applying the same algorithms to the 8-bit sRGB camera-rendered images. Another promising research direction is to consider over-exposure errors in images. This direction of research requires generating a dataset of *both* exposure errors – namely, under- and over-exposure errors – in camera-rendered images.

3.4 Post-Capture Color Editing

As discussed in Chapter 2, camera ISPs apply a set of nonlinear color manipulations as a part of the color rendering process. Once the image is rendered, post-capture color editing

(A) Input image (B) Target image (C) Recolored

Figure 3.11: Examples of color transfer. In this example, we use the color transfer method proposed by Reinhard et al. [318]. (A) Input image. (B) Target image. (C) Recolored image.

can be performed using photo editing techniques and filters. One of the fundamental research directions related to color editing is "color transfer", which aims at transferring the colors of a given input image to share the same "feel" with another target image colors [108].

Figure 3.11 shows a typical example of transferring colors from a target image to the input image. As it can be seen, color transfer produces similar effect to image filters provided in photo editing applications and capturing filters provided in smartphone cameras (e.g., [391]). In contrast, recent color transfer methods can achieve more compelling results with the ability to accurately adjusted based on the target image.

Color transfer methods can be categorized into the following categories: (i) geometry-based methods, (ii) statistical-based methods, (iii) learning-based methods, and (iv) color palette-based methods.

3.4.1 Geometry-Based Methods

This category of color transfer methods (e.g., [357, 386]) aims to find semantic similarities between input and target images to achieve realistic color mapping. These methods usually rely on feature detection methods, such as the scale-invariant feature transform (SIFT) [249] or the speeded-up robust feature (SURF) [47], to extract features of both images—namely, the input and target images. Then, a matching technique (e.g., [49]) is used to determine a set of candidate correspondences. Once the candidate correspondences are defined, one simple approach is to build an LUT for color mapping.

One drawback of these methods is that they mainly rely on the accuracy of finding the candidate correspondences. In many cases, however, it is hard to find reasonable feature correspondences between objects present in both input and target images.

3.4.2 Statistical-Based Methods

Statistical-based methods, on the other hand, aims at transferring statistical properties from the target image to the input image. A simple statistical-based method was proposed by Reinhard et al. [318], where color mapping was attained by transferring simple statistical moments (mean and standard deviation) between each channel of both images. Specifically, given two images $\mathbf{I}_{\text{in}}$ and $\mathbf{I}_{\text{target}}$, transferring the colors of $\mathbf{I}_{\text{target}}$ to $\mathbf{I}_{\text{in}}$ to produce a recolored image $\mathbf{I}_{\text{recol}}$ can be computed as follows [318]:

$$
\begin{aligned}
\mathbf{I'}_{\text{in}_{L^*}} &= \mathbf{I}_{\text{in}_{L^*}} - \mu_{\text{in}_{L^*}}, \quad \mathbf{I'}_{\text{in}_{a^*}} = \mathbf{I}_{\text{in}_{a^*}} - \mu_{\text{in}_{a^*}}, \\
\mathbf{I'}_{\text{in}_{b^*}} &= \mathbf{I}_{\text{in}_{b^*}} - \mu_{\text{in}_{b^*}}, \\
\mathbf{I}_{\texttt{recol}_{L^*}} &= \left(\sigma_{\text{target}_{L*}}/\sigma_{\text{in}_{L^*}}\right)\ \mathbf{I'}_{\text{in}_{L^*}} + \mu_{\text{target}_{L^*}}, \\
\mathbf{I}_{\texttt{recol}_{a^*}} &= \left(\sigma_{\text{target}_{a*}}/\sigma_{\text{in}_{a^*}}\right)\ \mathbf{I'}_{\text{in}_{a^*}} + \mu_{\text{in}_{a^*}}, \\
\mathbf{I}_{\texttt{recol}_{b^*}} &= \left(\sigma_{\text{target}_{b*}}/\sigma_{\text{in}_{b^*}}\right)\ \mathbf{I'}_{\text{in}_{b^*}} + \mu_{\text{in}_{b^*}},
\end{aligned}
\tag{3.24}
$$

where $\mathbf{I}_{L^*}$, $\mathbf{I}_{a^*}$, and $\mathbf{I}_{b^*}$ are the CIE $\mathtt{L}^*$, $\mathtt{a}^*$, $\mathtt{b}^*$ components of an image $\mathbf{I}$, respectively, μ is the arithmetic mean, and σ is the standard deviation. Despite the various methods proposed to improve the baseline color transfer method [318] described in Eq. 3.24 (e.g., [109, 161, 287, 305, 306, 381]), all statistical-based methods aims to find a better histogram mapping between images.

For instance, Pitié et al. [305,306] proposed to map the entire 3-dimensional histogram of the source image using a 3D rotation matrix computed iteratively. Nguyen et al. [287] he showed that WB both images (input and target) provides a fast alignment mechanism for color mapping. Thus, a heuristic technique based on image white balancing was proposed by iteratively computing a linear transformation matrix for histogram mapping. Though such statistical-based methods have a less restriction compared to geometry-based methods, the results are not always realistic as such methods do not consider semantic information in both images.

3.4.3 Learning-Based Methods

Despite the efficiency of using color histograms for color mapping, recent deep learning methods mostly use images as an input without an explicit reliance on color histograms. These methods do not only transfer colors between images, but also consider the texture information [129,130,179,186,251,337,360]. That is, the goal of these methods is to achieve image "style" transfer. Such learning-based style transfer methods can be categorized to: (i) image-optimization-based methods and (ii) model-optimization-based methods [184].

The majority of neural style transfer work belongs to the first category, where an online optimization process is performed in order to transfer the style of a target image to the input image (e.g., [129, 131, 322, 337]). The first deep learning endeavor to produce images with artistic-style is Deep Dream [273]. Deep Dream works by reversing a CNN trained

for image classification using image-optimization. That is, the optimization process begins with a full image of noise and it is updated through the optimization process in order to make the network predicts a certain output class.

Inspired by Deep Dream, Gatys et al. [129] proposed to transfer the style of target image to the source image by transferring statistics of the target image representations from intermediate layers of a pre-trained network (i.e., deep features) to the corresponding input image deep features' statistics. In order to retain the source image content in the output image, while transferring only the style from the target image, the optimization process minimizes a dual objective function. This objective function mainly includes a content loss (i.e., similarity between input and output images' contents) and style loss (i.e., similarity between target and output images' styles). The former can be computed using the squared Euclidean distance between deep features of the input and output images. The latter, however, requires a mechanism to model the visual texture to encourage the network to transfer the style of the target image to the source image. To that end, the Gram matrix is used to encode the correlations between deep features of different layers in the pre-trained classification network [129].

In spite of the impressive results achieved by the image-optimization-methods (see Fig. 3.12), these methods are usually inefficient for interactive applications. For example, Gaty et al.'s optimization method [129] takes $\sim$4 minutes to process a single image using a single GTX 1080 GPU. On the other hand, model-optimization-based methods for neural style transfer offers a more efficient solution. However, these model-optimization-based methods are mostly limited to transfer a single style per model – i.e., a new model should be re-trained for any new target style – (representative examples include [186, 359, 361, 401]). There are a few attempts to achieve multiple-style transfer per model [230] or arbitrary-style transfer per-model [231]. The quality of the results produced by these methods,

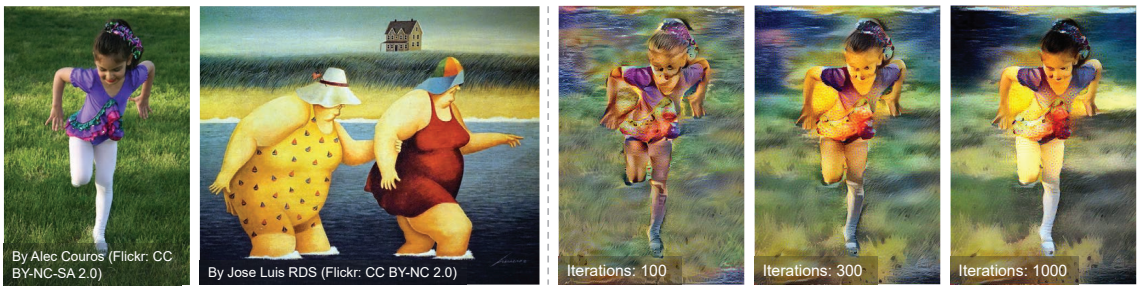


Figure 3.12: Examples of neural style transfer. In this example, we show the results of the method proposed by Gaty et al. [318]. (A) Input image. (B) Target image. (C) Results of optimization after different number of iterations.

however, mostly depends on the matching degree of the semantic content and/or the colors between both images—namely, the input and target images [160].

3.4.4 Color Palette-Based Methods

Color palette-based methods offer another type of color mapping, where the target image is replaced by a target color palette. Color palette is a compact representation of the main colors of a given image [70]. Palette-based methods assume that colors of a given image $\mathbf{I}$ can be approximately represented by a linear combination of a small number of colors (color palettes) as follows:

$$\mathbf{I}^{(p)} = \sum_{i=1}^{k} \mathbf{w}_i^{(p)} \mathbf{C}_i, \tag{3.25}$$

where $\mathbf{C}_i$ is the i^{th} color in the color palette and k is the total number of colors in this color palette, $\mathbf{w}_i^{(p)}$ is a weighting factor that represent the contribution of the color $\mathbf{C}_i$ in forming the color in the original image $\mathbf{I}$ at pixel p. As shown in Eq. 3.25, by defining the unknowns $\mathbf{C}$ and $\mathbf{w}$, the user can interactively change the palette colors $\mathbf{C}_i$ $(i \in 1, ..., k)$ to recolor the input image I. Palette-based methods (e.g., [29, 30, 70, 354, 355]) introduced different ways in order to find the most effective set of colors in the color palette and

utilized regularized color decomposition optimization techniques to compute the weighting factors.

Similar to other color mapping techniques, color palette-based approaches may introduce artifacts (e.g., color bleeding) in the recolored images based on the changes in the target color palette.

3.4.5 Research Directions

By definition, color transfer requires a target image/color palette in order to transfer the colors of this target image to the input image. These image exemplars, however, have a direct impact on the quality of the color transfer process [108]. This motivated a few methods towards achieving automatic color transfer (e.g., [172, 216]). These methods, however, are usually restricted to deal with certain type of images (e.g., outdoor images [216]). Prior work shows the significant role of WB on changing the global colors of images [287]. A potential research direction is to use WB to achieve image color manipulation in the post-capture stage. However, as discussed earlier, WB editing, or chromatic adaptation generally, is applied on linear images (i.e., raw or CIE XYZ images). The challenge would be to achieve accurate WB manipulation on camera-rendered images. Another interesting research direction could be auto image recoloring, where different recolored versions of the input image are produced automatically without any user interaction required. This auto recoloring should consider semantic content present in the image in order to produce realistic recolored images.

3.5 Summary

We have provided a survey of different approaches for color correction and editing. We began by reviewing existing methods for image white balancing including illuminant esti-

mation and chromatic adaptation techniques for scene-referred linear images. Furthermore, we have explained why the current solutions cannot deal with improperly white-balanced camera-rendered images due to the nonlinearity applied on board cameras. Afterward, we briefly discussed other factors that have a significant effect on the quality of camera-rendered image colors. Specifically, we have discussed exposure errors in photographs and reviewed existing methods to enhance images rendered with low-light conditions/exposure errors. Then, we have reviewed post-capture color transfer methods. This chapter also has discussed promising research directions for color correction and editing, which motivated our work presented in the next chapters of this thesis.

Part II

Computational Color Constancy

4 Sensor-Independent Color Constancy

The previous chapter introduced a learning-based procedure to improve CC. While learning-based methods for illuminant estimation (especially the modern deep neural networks) achieve state-of-the-art results, it is currently necessary to train a separate DNN for each type of camera sensor. This means when a camera manufacturer uses a new sensor, it is necessary to re-train an existing DNN model with training images captured by the new sensor. This chapter addresses this problem by introducing a novel sensor-independent illuminant estimation framework [1]. Our method learns a sensor-independent *working space* that can be used to canonicalize the RGB values of any arbitrary camera sensor. Our learned space retains the linear property of the original sensor raw-RGB space and allows unseen camera sensors to be used on a single DNN model trained on this working space. We demonstrate the effectiveness of this approach on several different camera sensors and show it provides performance on par with state-of-the-art methods that were trained per sensor. The source code of this work is available on GitHub: https://github.com/mahmoudnafifi/SIIE.

[1] This work was published in [13]: Mahmoud Afifi and Michael S. Brown. Sensor-Independent Illumination Estimation for DNN Models. In British Machine Vision Conference (BMVC), 2019.

4.1 Introduction

Recall that in Chapter 3, we have described the computational CC in terms of the physical image formation process as follows. Let $\mathbf{I} = \{\mathbf{I}_r, \mathbf{I}_g, \mathbf{I}_b\}$ denote an image captured in the linear raw-RGB space. The value of each color channel $c = \{\mathrm{R}, \mathrm{G}, \mathrm{B}\}$ for a pixel located at x in $\mathbf{I}$ is given by the following equation [46]:

$$\mathbf{I}_c(x) = \int_{\gamma} \rho(x, \lambda) R(x, \lambda) S_c(\lambda) d\lambda, \tag{4.1}$$

where γ is the visible light spectrum (approximately 380nm to 780nm), $\rho(\cdot)$ is the illuminant spectral power distribution, $R(\cdot)$ is the captured scene's spectral reflectance properties, and $S(\cdot)$ is the camera sensor response function at wavelength λ. The problem can be simplified by assuming a single uniform illuminant in the scene as follows:

$$\mathbf{I}_c = \boldsymbol{\ell}_c \times \mathbf{R}_c, \tag{4.2}$$

where $\boldsymbol{\ell}_c$ is the scene illuminant value of color channel c. A standard approach to this problem is to use a linear model (i.e., a 3×3 diagonal matrix) to such that $\boldsymbol{\ell}_{\mathrm{R}} = \boldsymbol{\ell}_{\mathrm{G}} = \boldsymbol{\ell}_{\mathrm{B}}$ (i.e., white illuminant).

Typically, $\boldsymbol{\ell}$ is unknown and should be defined to obtain the true objects' body reflectance values $\mathbf{R}$ in the input image $\mathbf{I}$. The value of $\boldsymbol{\ell}$ is specific to the camera sensor response function $S(\cdot)$, meaning that the same scene captured by different camera sensors results in different values of $\boldsymbol{\ell}$. Figure 4.1 shows an example.

Illuminant estimation methods aim to estimate the value $\boldsymbol{\ell}$ from the sensor's raw-RGB image. Recently, DNN methods have demonstrated state-of-the-art results for the illuminant estimation task. These approaches, however, need to train the DNN model per camera sensor. This is a significant drawback. When a camera manufacture decides to use a new sensor, the DNN model will need to be retrained on a new image dataset captured by

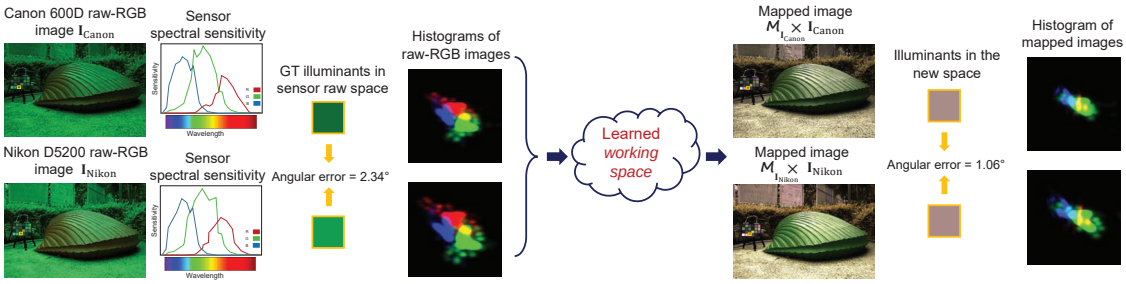


Figure 4.1: A scene captured by two different camera sensors results in different ground truth illuminants due to different camera sensor responses. We learn a device-independent *working space* that reduces the difference between ground truth illuminants of the same scenes.

the new sensor. Collecting such datasets with the corresponding ground-truth illuminant raw-RGB values is a tedious process. As a result, many AWB algorithms deployed on cameras still rely on simple statistical-based methods even though the accuracy is not comparable to those obtained by the learning-based methods.

Contribution In this chapter, we introduce a sensor-independent learning framework for illuminant estimation. The idea is similar to the color space conversion process applied onboard cameras that maps the sensor-specific RGB values to a perceptual-based color space – namely, CIE XYZ. The color space conversion process estimates a color space transform (CST) matrix to map white-balanced sensor-specific raw-RGB images to CIE XYZ [191, 316]. This process is applied onboard cameras *after* the illuminant estimation and white-balance step, and relies on the estimated scene illuminant to compute the CST matrix [64]. Our solution, however, is to learn a new space that is used *before* the illuminant estimation step. Specifically, we design a novel unsupervised deep learning framework that learns how to map each input image, captured by arbitrary camera sensor, to a non-perceptual sensor-independent *working space*. Mapping input images to this space, allows

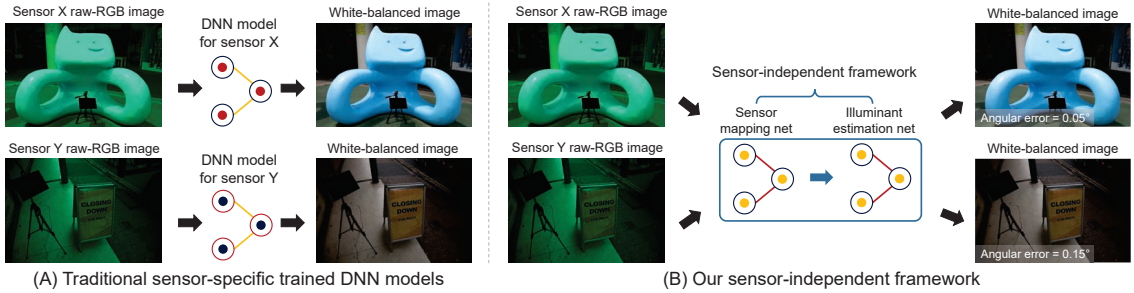


Figure 4.2: (A) Traditional learning-based illuminant estimation methods train or fine-tune a model per camera sensor. (B) Our method can be trained on images captured by different camera sensors and generalizes well for unseen camera sensors. Shown images are rendered in the sRGB color space by the camera imaging pipeline in [191] to aid visualization.

us to train our model using training sets captured by different camera sensors achieving good accuracy and generalizing well for unseen camera sensors as shown in Fig. 4.2.

4.2 Proposed Method

Figure 4.3 provides an overview of our sensor-independent illuminant estimation (SIIE) framework. Our SIIE accepts thumbnail (150×150 pixels) linear raw-RGB images, captured by an arbitrary camera sensor and estimates scene illuminant RGB vectors in the same space of input images. We decided to make SIIE accepting thumbnail images instead of full-sized images because such thumbnail images are often already produced by camera ISPs, and processing these downsized images mostly produces illuminant estimation accuracy that is on par with the results using full-sized images, but with less memory and computational power [45].

We rely on color distribution of input thumbnail image $\mathbf{I}$ to estimate an image-specific transformation matrix that maps the input image to our working space. This mapping

allows us to accept images captured by different sensors and estimate scene illuminant values in the original space of input images.

We begin with the formulation of our problem followed by a detailed description of our framework components and the training process. Note that we will assume input raw-RGB images are represented as $3\times n$ matrices, where $n = 150\times150$ is the total number of pixels in the thumbnail image and the three rows represent the R, G, and B values.

4.2.1 Problem Formulation

We propose to work in a new learned space for illumination estimation. This space is sensor-independent and retains the linear property of the original raw-RGB space. To that end, we introduce a learnable 3×3 matrix $\boldsymbol{\mathcal{M}}$ that maps an input image $\mathbf{I}$ from its original sensor-specific space to a new working space. We can reformulate Eq. 4.2 as follows:

$$\boldsymbol{\mathcal{M}}^{-1}\boldsymbol{\mathcal{M}}\mathbf{I} = \texttt{diag}(\boldsymbol{\mathcal{M}}^{-1}\boldsymbol{\mathcal{M}}\boldsymbol{\ell})\mathbf{R}, \tag{4.3}$$

where $\texttt{diag}(\cdot)$ is a diagonal matrix and $\boldsymbol{\mathcal{M}}$ is a learned matrix that maps arbitrary sensor responses to a sensor-independent space.

Given a mapped image $\mathbf{I}_m = \boldsymbol{\mathcal{M}}\mathbf{I}$ in our learned space, we aim to estimate the mapped vector $\boldsymbol{\ell}_m = \boldsymbol{\mathcal{M}}\boldsymbol{\ell}$ that represents the scene illumination values of $\mathbf{I}_m$ in the new space. The original scene illuminant (represented in the original sensor raw-RGB space) can be reconstructed by the following equation:

$$\boldsymbol{\ell} = \boldsymbol{\mathcal{M}}^{-1}\boldsymbol{\ell}_m. \tag{4.4}$$

4.2.2 RGB–uv Histogram Block

Prior work has shown that the illumination estimation problem is related primarily to the image's color distribution [44, 45]. Accordingly, we use the image's color distribution as

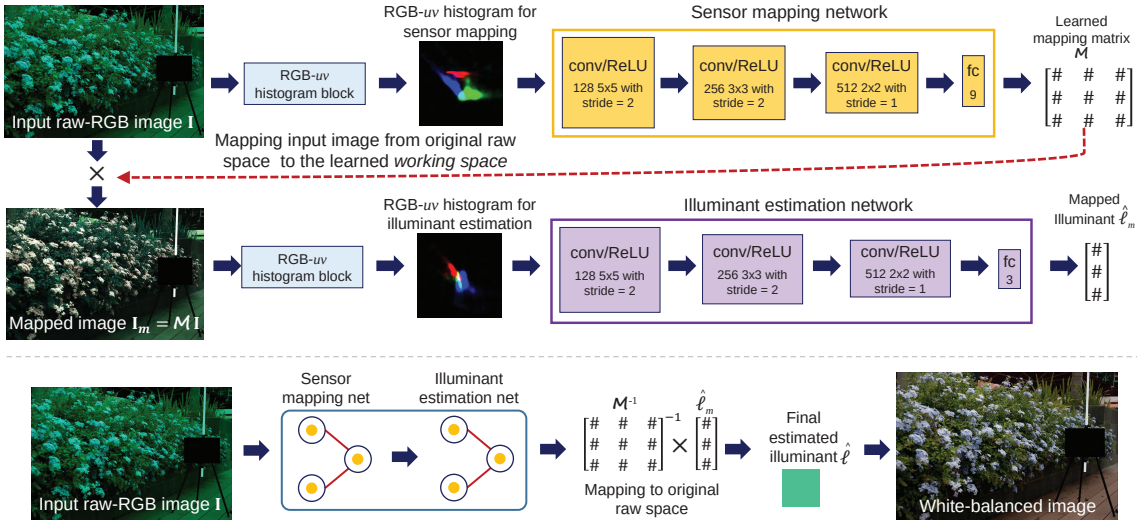


Figure 4.3: Our proposed sensor-independent illuminant estimation (SIIE) framework consists of two networks: (i) a sensor mapping network and (ii) an illuminant estimation network. Our networks are trained jointly in an end-to-end manner to learn an image-specific mapping matrix (resulting from the sensor mapping network) and scene illuminant in the learned space (resulting from the illuminant estimation network). The final estimated illuminant is produced by mapping the result illuminant from our learned space to the input image's camera-specific raw space.

an input for our SIIE. Representing the image using a full 3D RGB histogram requires significant amounts of memory – for example, a 256^3 RGB histogram requires more than 16 million entries. Down-sampling the histogram – for example, to 64-bins – still requires a considerable amount of memory. Thus, we use a projected histogram feature. Our histogram feature is inspired by prior work [44, 45], in which we construct our feature in the log-chrominance space [98,115], which represents the color distribution of an image $\mathbf{I}$ as an $m \times m \times 3$ tensor that is parameterized by uv. We refer to this as an RGB-uv histogram.

We use two learnable parameters to control the contribution of each color channel in the generated histogram and the smoothness of histogram bins. Specifically, our RGB-uv histogram block represents the color distribution of an image $\mathbf{I}$ as a three-layer histogram $\mathbf{H}(\mathbf{I})$ represented as an $m \times m \times 3$ tensor. The produced histogram $\mathbf{H}(\mathbf{I})$ is parameterized by uv and computed as follows:

$$
\begin{aligned}
\mathbf{I}_{y}(i) &= \sqrt{\mathbf{I}^2_{\mathrm{R}(i)} + \mathbf{I}^2_{\mathrm{G}(i)} + \mathbf{I}^2_{\mathrm{B}(i)}}, \\
\mathbf{I}_{u1(i)} = \log\left(\frac{\mathbf{I}_{\mathrm{R}(i)}}{\mathbf{I}_{\mathrm{G}(i)}} + \epsilon\right) &, \mathbf{I}_{v1(i)} = \log\left(\frac{\mathbf{I}_{\mathrm{R}(i)}}{\mathbf{I}_{\mathrm{B}(i)}} + \epsilon\right), \\
\mathbf{I}_{u2(i)} = \log\left(\frac{\mathbf{I}_{\mathrm{G}(i)}}{\mathbf{I}_{\mathrm{R}(i)}} + \epsilon\right) &, \mathbf{I}_{v2(i)} = \log\left(\frac{\mathbf{I}_{\mathrm{G}(i)}}{\mathbf{I}_{\mathrm{B}(i)}} + \epsilon\right), \\
\mathbf{I}_{u3(i)} = \log\left(\frac{\mathbf{I}_{\mathrm{B}(i)}}{\mathbf{I}_{\mathrm{R}(i)}} + \epsilon\right) &, \mathbf{I}_{v3(i)} = \log\left(\frac{\mathbf{I}_{\mathrm{B}(i)}}{\mathbf{I}_{\mathrm{G}(i)}} + \epsilon\right), \\
\mathbf{H}\left(\mathbf{I}\right)_{(u,v,c)} &= \left(s_c \sum_i \mathbf{I}_{y(i)} \exp\left(-\left|\mathbf{I}_{uc(i)} - u\right| / \sigma_c^2\right) \exp\left(-\left|\mathbf{I}_{vc(i)} - v\right| / \sigma_c^2\right)\right)^{1/2},
\end{aligned}
\tag{4.5}
$$

where $i = \{1, ..., n\}$, $c \in \{1, 2, 3\}$ represents each color channel in $\mathbf{H}$, ϵ is a small positive constant added for numerical stability, and s_c and σ_c are learnable scale and fall-off parameters, respectively. The scale factor s_c controls the contribution of each layer in our histogram, while the fall-off factor σ_c controls the smoothness of the histogram's bins of each layer. The values of these parameters (i.e., s_c and σ_c) are learned during the training phase.

4.2.3 Network Architecture

As shown in Fig. 4.3, our framework consists of two networks: (i) a sensor mapping network and (ii) an illuminant estimation network. The input to each network is the RGB-uv histogram feature produced by our histogram block. The sensor mapping network accepts an RGB-uv histogram of a thumbnail raw-RGB image $\mathbf{I}$ in its original sensor space, while the illuminant estimation network accepts RGB-uv histograms of the mapped image $\mathbf{I}_m$ to our learned space. In our implementation, we use $m = 61$ and each histogram feature is represented by a $61 \times 61 \times 3$ tensor.

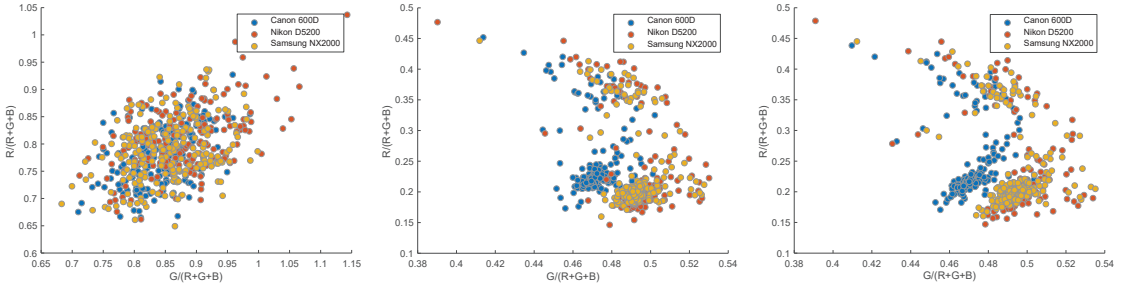
We use a simple network architecture for each network. Specifically, each network consists of three conv/ReLU layers followed by a fully connected (fc) layer. The kernel size and stride step used in each conv layer are shown in Fig. 4.3.

In the sensor mapping network, the last fc layer has nine neurons. The output vector $\mathbf{v}$ of this fc layer is reshaped to construct a 3×3 matrix $\mathbf{V}$, which is used to build $\boldsymbol{\mathcal{M}}$ as described in the following equation:

$$\boldsymbol{\mathcal{M}} = \frac{1}{\|\mathbf{V}\|_1 + \epsilon} |\mathbf{V}|, \tag{4.6}$$

where $|\cdot|$ is the modulus (absolute magnitude), $\|\cdot\|_1$ is the matrix 1-norm, and $\epsilon = 10^{-8}$ is added for numerical stability. The modulus step is necessary to avoid negative values in the mapped image $\mathbf{I}_m$, while the normalization step is used to avoid having extremely large values in $\mathbf{I}_m$. Note the values of $\boldsymbol{\mathcal{M}}$ are image-specific, meaning that its values are produced based on the input image's color distribution in the original raw-RGB space.

There are three neurons in the last fc layer of the illuminant estimation network to produce illuminant vector $\hat{\boldsymbol{\ell}}_m$ of the mapped image $\mathbf{I}_m$. Note that the estimated vector $\hat{\boldsymbol{\ell}}_m$ represents the scene illuminant in our learned space. The final result is obtained by mapping $\hat{\boldsymbol{\ell}}_m$ back to the original space of $\mathbf{I}$ using Eq. 4.4.



(A) Estimated illuminants in the *working space* of three different camera sensors

(B) Estimated illuminants after mapping to original raw-RGB space using image-specific learned maps

(C) Ground truth illuminants in original sensor-specific raw-RGB spaces

Figure 4.4: Raw-RGB images capture the same set of scenes using three different cameras taken from the NUS 8-Cameras dataset [77]. (A) Estimated illuminants resulted from the illuminant estimation network in our learned *working space*. (B) Estimated illuminants after mapping to the original raw-RGB space. This mapping is performed by multiplying each illuminant vector by the inverse of the learned image-specific mapping matrix (resulting from the sensor mapping network). (C) Corresponding ground truth illuminants in the original raw-RGB space of each image.

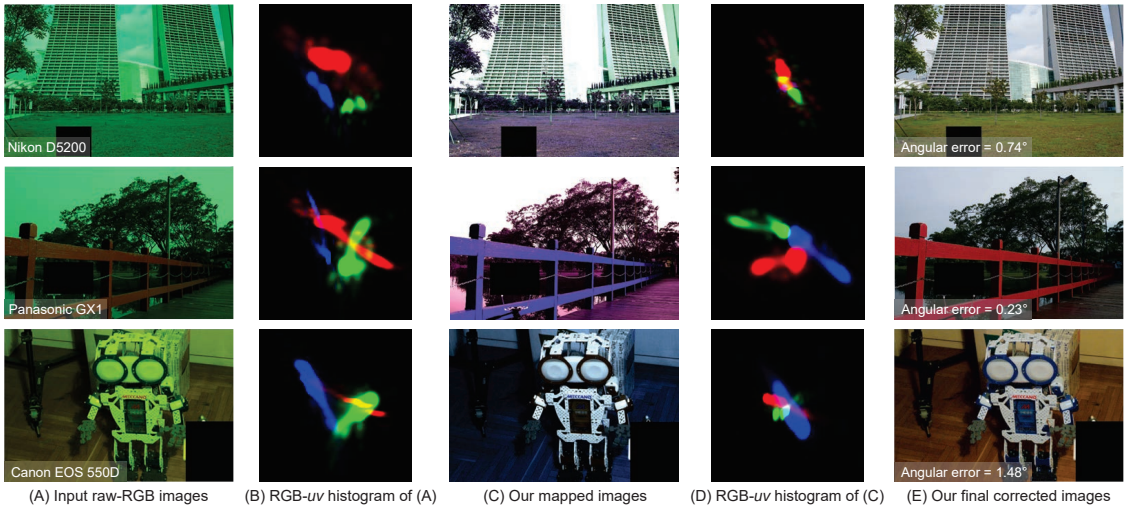


Figure 4.5: Example of our generated RGB-uv histograms. (A) Input raw-RGB images. (B) Generated histograms of images in (A). (C) After mapping images in (A) to the learned space. (D) Generated histograms of images in (C). (E) After correcting images in (A) based on our estimated illuminants. Shown images are rendered in the sRGB color space by the camera imaging pipeline in [191] to aid visualization.

4.2.4 Training

We jointly train our sensor mapping and illuminant estimation networks in an end-to-end manner using the adaptive moment estimation (Adam) optimizer [204] with a decay rate of gradient moving average $\beta_1 = 0.85$, a decay rate of squared gradient moving average $\beta_2 = 0.99$, and a mini-batch with eight observations at each iteration. We initialized both network weights with Xavier initialization [141]. The learning rate was set to 10^{-5} and decayed every five epochs.

We adopt the recovery angular error (referred to as the angular error) as our loss function [168]. The angular error is computed between the ground truth illuminant $\boldsymbol{\ell}$ and our estimated illuminant $\hat{\boldsymbol{\ell}}_m$ after mapping it to the original raw-RGB space of training

image $\mathbf{I}$. The loss function can be described by the following equation:

$$\mathcal{L}(\hat{\boldsymbol{\ell}}_m, \boldsymbol{\mathcal{M}}) = \cos^{-1}\left(\frac{\boldsymbol{\ell} \cdot \left(\boldsymbol{\mathcal{M}}^{-1}\hat{\boldsymbol{\ell}}_m\right)}{\|\boldsymbol{\ell}\|\|\boldsymbol{\mathcal{M}}^{-1}\hat{\boldsymbol{\ell}}_m\|}\right), \tag{4.7}$$

where $\|\cdot\|$ is the Euclidean norm, and $(\cdot)$ is the vector dot-product.

As the values of $\boldsymbol{\mathcal{M}}$ are produced by the sensor mapping network, there is a possibility of producing a singular matrix output. In this case, we add small offset $\mathcal{N}(0,1)\times 10^{-4}$ to each parameter in $\boldsymbol{\mathcal{M}}$ to make it invertible.

At the end of the training process, our framework learns an image-specific matrix $\boldsymbol{\mathcal{M}}$ that maps an input image taken by an arbitrary sensor to the learned space. Figure 4.4 shows an example of three different camera responses capturing the same set of scenes. As shown in Fig. 4.4-(A), the estimated illuminants of these sensors are bounded in the learned space. These illuminants are mapped back to the original raw-RGB sensor space of the corresponding input images using Eq. 4.4. As shown in Fig. 4.4-(B) and Fig. 4.4-(C), our final estimated illuminants are close to the ground truth illuminants of each camera sensor. Figure 4.5 shows examples of the generated histograms of input images in original raw-RGB space and our learned space.

4.3 Experimental Results

In our experiments, we used all cameras of three different datasets, which are: (i) NUS 8-Camera [77], (ii) Gehler-Shi [132], and (iii) Cube+ [41] datasets. In total, we have 4,014 raw-RGB images captured by 11 different camera sensors.

We followed the leave-one-out cross-sensor validation scheme to evaluate our method. Specifically, we excluded all images captured by one camera for testing and trained a model with the remaining images. This process was repeated for all cameras.

We also tested our SIIE on the Cube dataset. In this experiment, we used a trained

model on images from the NUS and Gehler-Shi datasets, and excluded all images from the Cube+ dataset.

The calibration objects (i.e., X-Rite color chart or SpyderCUBE) were masked out in both training and testing processes.

Unlike results reported by existing learning methods which use three-fold cross-validation for evaluation, our reported results were obtained by models that were *not* trained on any example of the testing camera sensor.

In Tables 4.1–4.2, the mean, median, best 25%, and the worst 25% of the angular error between our estimated illuminants and ground truth are reported. The best 25% and worst 25% are the mean of the smallest 25% angular error values and the mean of the highest 25% angular error values, respectively. We highlight learning methods (i.e., models trained/tuned for the testing sensor) with gray in the shown tables. It is notable that our SIIE performs better than all statistical-based methods and outperforms some sensor-specific learning methods. We obtain results on par with the state-of-the-art results in the NUS 8-Camera dataset (Table 4.1). We would like to emphasize that these state-of-the-art results are obtained by *sensor-specific* methods and reported using three-fold cross-validation on images taken by the same sensor.

In Table 4.3, we show our results on each camera of the NUS 8-Camera dataset. We report the mean, median, best 25%, and the worst 25% of the angular error between our estimated illuminants and ground truth.

We also examined our trained models on the Cube+ challenge [39]. This challenge introduced a new testing set of 363 raw-RGB images captured by Canon EOS 550 D (the same camera model used in the original Cube+ dataset [41]). In our results, we did not include any image from the testing set in the training/validation processes. Instead, we used the same models trained for the evaluation on the other datasets (Tables 4.1–4.2).

Table 4.1: Angular errors on the NUS 8-Cameras [77] and Gehler-Shi [132] datasets. Methods highlighted in gray are trained/tuned for each camera sensor (i.e., sensor-specific models). The lowest errors are highlighted in yellow.

NUS 8-Cameras Dataset	**Mean**	**Med.**	**Best**	**Worst**
Method			**25%**	**25%**
White-Patch [58]	9.91	7.44	1.44	21.27
Pixel-based Gamut [136]	5.27	4.26	1.28	11.16
GW [60]	4.59	3.46	1.16	9.85
Edge-based Gamut [136]	4.40	3.30	0.99	9.83
SoG [120]	3.67	2.94	0.98	7.75
Bayesian [132]	3.50	2.36	0.78	8.02
Local Surface Reflectance [128]	3.45	2.51	0.98	7.32
2nd-order GE [362]	3.36	2.70	0.89	7.14
1st-order GE [362]	3.35	2.58	0.79	7.18
Quasi-U CC [51]	3.00	2.25	-	-
Corrected-Moment [111]	2.95	2.05	0.59	6.89
PCA-based B/W Colors [77]	2.93	2.33	0.78	6.13
Grayness Index [313]	2.91	1.97	0.56	6.67
Color Dog [40]	2.83	1.77	0.48	7.04
APAP using GW [23]	2.40	1.76	0.55	5.42
CCC [44]	2.38	1.69	0.45	5.85
Effective Regression Tree [79]	2.36	1.59	0.49	5.54
Deep Specialized Net [338]	2.24	1.46	0.48	6.08
Meta-AWB w 20 tuning images [263]	2.23	1.49	0.49	5.20
SqueezeNet-FC4	2.23	1.57	0.47	5.15
AlexNet-FC4 [171]	2.12	1.53	0.48	4.78
FFCC – thumb, 2 channels [45]	2.06	1.39	0.39	4.80
FFCC – full, 4 channels [45]	1.99	1.31	0.35	4.75
Quasi-unsupervised CC (tuned) [51]	1.97	1.41	-	-
Avg. result for sensor-independent	4.26	3.25	0.99	9.43
Avg. result for sensor-dependent	2.40	1.64	0.50	5.75
SIIE (Ours)	2.05	1.50	0.52	4.48

Gehler-Shi Dataset	**Mean**	**Med.**	**Best**	**Worst**
Method			**25%**	**25%**
White-Patch [58]	7.55	5.68	1.45	16.12
Edge-based Gamut [136]	6.52	5.04	5.43	13.58
GW [60]	6.36	6.28	2.33	10.58
1st-order GE [362]	5.33	4.52	1.86	10.03
2nd-order GE [362]	5.13	4.44	2.11	9.26
SoG [120]	4.93	4.01	1.14	10.20
Bayesian [132]	4.82	3.46	1.26	10.49
Pixels-based Gamut [136]	4.20	2.33	0.50	10.72
Quasi-U CC [51]	3.46	2.23	-	-
PCA-based B/W Colors [77]	3.52	2.14	0.50	8.74
NetColorChecker [248]	3.10	2.30	-	-
Grayness Index [313]	3.07	1.87	0.43	7.62
Meta-AWB w 20 tuning images [263]	3.00	2.02	0.58	7.17
Quasi-unsupervised CC [51] (tuned)	2.91	1.98	-	-
Corrected-Moment [111]	2.86	2.04	0.70	6.34
APAP using GW [23]	2.76	2.02	0.53	6.21
Bianco et al.'s CNN [53]	2.63	1.98	0.72	3.90
Effective Regression Tree [79]	2.42	1.65	0.38	5.87
FFCC - thumb, 2 channels [45]	2.01	1.13	0.30	5.14
CCC [44]	1.95	1.22	0.35	4.76
Deep Specialized Net [338]	1.90	1.12	0.31	4.84
FFCC - full, 4 channels [45]	1.78	0.96	0.29	4.62
AlexNet-FC4 [171]	1.77	1.11	0.34	4.29
SqueezeNet-FC4 [171]	1.65	1.18	0.38	3.78
Avg. result for sensor-independent	5.10	4.03	1.91	10.77
Avg. result for sensor-dependent	2.62	1.75	0.50	5.95
SIIE (Ours)	2.77	1.93	0.55	6.53

Table 4.2: Angular errors on the Cube and Cube+ datasets [41]. Methods highlighted in gray are trained/tuned for each camera sensor (i.e., sensor-specific models). The lowest errors are highlighted in yellow.

Cube Dataset	**Mean**	**Med.**	**Best**	**Worst**
Method			**25%**	**25%**
White-Patch [58]	6.58	4.48	1.18	15.23
GW [60]	3.75	2.91	0.69	8.18
SoG [120]	2.58	1.79	0.38	6.19
2nd-order GE [362]	2.49	1.60	0.49	6.00
1st-order GE [362]	2.45	1.58	0.48	5.89
APAP using GW [23]	1.55	1.02	0.28	3.74
Color Dog [40]	1.50	0.81	0.27	3.86
Meta-AWB (20) [263]	1.74	1.08	0.29	4.28
Avg. result for sensor-independent	3.57	2.47	0.64	8.30
Avg. result for sensor-dependent	1.54	0.92	0.26	3.85
SIIE (Ours)	1.98	1.36	0.40	4.64

Cube+ Dataset	**Mean**	**Med.**	**Best**	**Worst**
Method			**25%**	**25%**
White-Patch [58]	9.69	7.48	1.72	20.49
GW [60]	7.71	4.29	1.01	20.19
Color Dog [40]	3.32	1.19	0.22	10.22
SoG [120]	2.59	1.73	0.46	6.19
2nd-order GE [362]	2.50	1.59	0.48	6.08
1st-order GE [362]	2.41	1.52	0.45	5.89
APAP using GW [23]	2.01	1.36	0.38	4.71
Color Beaver [205]	1.49	0.77	0.21	3.94
Avg. result for sensor-independent	4.98	3.32	0.82	11.77
Avg. result for sensor-dependent	2.04	1.02	0.25	5.58
SIIE (Ours)	2.14	1.44	0.44	5.06

Table 4.3: Our results (angular errors) on each camera of the NUS 8-Camera [77].

	NUS 8-Cameras Dataset							
Camera	**Canon EOS**	**Canon EOS**	**Fujifilm**	**Nikon**	**Olympus**	**Panasonic**	**Samsung**	**Sony**
	1Ds MrkIII	**600D**	**XM1**	**D5200**	**EPL6**	**GX1**	**NX2000**	**SLT-A57**
Mean	2.07	1.99	2.08	2.06	2.26	1.82	1.71	2.29
Median	1.59	1.43	1.46	1.51	1.73	1.41	1.32	1.78
Best 25%	0.48	0.56	0.56	0.55	0.63	0.49	0.41	0.54
Worst 25%	4.51	4.43	4.63	4.44	4.70	3.83	3.71	5.16

Table 4.4: Angular and reproduction angular errors [121] on the Cube+ challenge [39]. The methods are sorted by the median of the errors (shown in bold), as ranked in the challenge [39]. Methods highlighted in gray are sensor-specific models. We show our results w/wo training on Cube+ dataset. The lowest errors over all methods are highlighted in yellow.

Cube+ challenge (angular error)	**Mean**	**Med.**	**Best**	**Worst**
Method			**25%**	**25%**
GW [60]	4.44	**3.50**	0.77	9.64
1st-order GE [362]	3.51	**2.3**	0.56	8.53
V Vuk et al., [39]	6	**1.96**	0.99	18.81
Y Qian et al., (1) [39]	2.48	**1.56**	0.44	6.11
K Chen et al., [39]	1.84	**1.27**	0.39	4.41
Y Qian et al., (3) [39]	2.27	**1.26**	0.39	6.02
FFCC [45]	2.1	**1.23**	0.47	5.38
A Savchik et al., [39]	2.05	**1.2**	0.41	5.24
SIIE (ours) trained wo/ Cube+	2.89	**1.718**	0.71	7.061
SIIE (ours) trained w/ Cube+	2.1	**1.23**	0.47	5.38

Cube+ challenge (reproduction error)	**Mean**	**Med.**	**Best**	**Worst**
Method			**25%**	**25%**
GW [60]	5.74	**4.60**	1.12	12.21
1st-order GE [362]	4.57	**3.22**	0.84	10.75
V Vuk et al., [39]	6.87	**2.1**	1.06	21.82
Y Qian et al., (1) [39]	6.87	**2.09**	0.61	8.18
K Chen et al., [39]	2.49	**1.69**	0.52	6.00
Y Qian et al., (3) [39]	2.93	**1.64**	0.50	7.78
FFCC [45]	2.48	**1.59**	0.58	7.27
A Savchik et al., [39]	2.65	**1.51**	0.50	6.85
SIIE (ours) trained wo/ Cube+	3.97	**2.31**	0.86	10.07
SIIE (ours) trained w/ Cube+	2.8	**1.54**	0.58	7.27

Table 4.5: This table shows the angular and reproduction angular errors [121] obtained on the Cube+ challenge [39] using our trained models. We did not use any example from the Cube+ challenge testing set in the training/validation sets. The reported results in Table 4.4 are highlighted in green.

Cube+ challenge	**Mean**	**Med.**	**Best**	**Worst**	**Cube+ challenge**	**Mean**	**Med.**	**Best**	**Worst**
Method			**25%**	**25%**	**Method**			**25%**	**25%**
Trained wo/ Canon EOS 550 D (Cube+)	2.89	1.72	0.71	7.06	Trained wo/ Canon EOS 550 D (Cube+)	3.97	2.31	0.86	10.07
Trained wo/ Canon 1Ds MkIII (NUS)	1.98	1.22	0.43	4.89	Trained wo/ Canon 1Ds MkIII (NUS)	2.65	1.59	0.54	6.54
Trained wo/ Canon 600D (NUS)	1.96	1.31	0.44	4.72	Trained wo/ Canon 600D (NUS)	2.59	1.69	0.53	6.248
Trained wo/ Fujifilm XM1 (NUS)	2.31	1.61	0.52	5.36	Trained wo/ Fujifilm XM1 (NUS)	3.08	2.19	0.67	7.1
Trained wo/ Nikon D5200 (NUS)	1.97	1.22	0.47	4.75	Trained wo/ Nikon D5200 (NUS)	2.62	1.73	0.57	6.29
Trained wo/ Olympus EPL6 (NUS)	2.4	1.92	0.58	5.21	Trained wo/ Olympus EPL6 (NUS)	3.23	2.59	0.76	6.97
Trained wo/ Panasonic GX1 (NUS)	2.21	1.44	0.65	5.14	Trained wo/ Panasonic GX1 (NUS)	2.89	1.86	0.74	6.86
Trained wo/ Samsung NX2000 (NUS)	2.02	1.38	0.38	4.92	Trained wo/ Samsung NX2000 (NUS)	2.7	1.89	0.48	6.51
Trained wo/ Sony SLT-A57 (NUS)	2.1	1.23	0.47	5.38	Trained wo/ Sony SLT-A57 (NUS)	2.8	1.54	0.58	7.27
Trained wo/ Sony Canon 5D (Gehler-Shi)	2.02	1.27	0.432	4.927	Trained wo/ Sony Canon 5D (Gehler-Shi)	2.69	1.68	0.54	6.59

Table 4.4 shows the angular error and reproduction angular errors [121] obtained by our models and the top-ranked methods that participated in the challenge. Additionally, we show results obtained by other methods [60, 120].

We report results of two trained models using our method. The first one was trained without examples from Cube+ camera sensor (i.e., trained on all camera models in NUS and Gehler-Shi datasets). The second model was originally trained to evaluate our method on one camera of the NUS 8-Cameras dataset (i.e., trained on 7 out of the 8 camera models in NUS 8-Cameras dataset, the Cube+ camera model, and the Gehler-Shi camera models). The latter model is provided to demonstrate the ability of our method to use different camera models beside the target camera model during the training phase.

Table 4.5 provides our results obtained on the Cube+ challenge [39] using different trained models. The models were originally trained for evaluation on NUS 8-Camera [77], Gehler-Shi [132], and Cube+ [41] datasets using the leave-one-out cross-sensor validation scheme. We did not use any example from the Cube+ challenge testing set in the training/validation phases.

We further tested our trained models on the INTEL-TUT dataset [37], which includes DSLR and mobile phone cameras that are not included in the NUS 8-Camera, Gehler-Shi, and Cube+ datasets. Table 4.6 shows the obtained results by the proposed method trained on DSLR cameras from the NUS 8-Camera, Gehler-Shi, and Cube+ datasets.

Finally, we show qualitative examples in Fig. 4.6. For each example, we show the mapped image $\mathbf{I}_m$ in our learned intermediate space. In the shown figure, we rendered the images in the sRGB color space by the camera imaging pipeline in [191] to aid visualization.

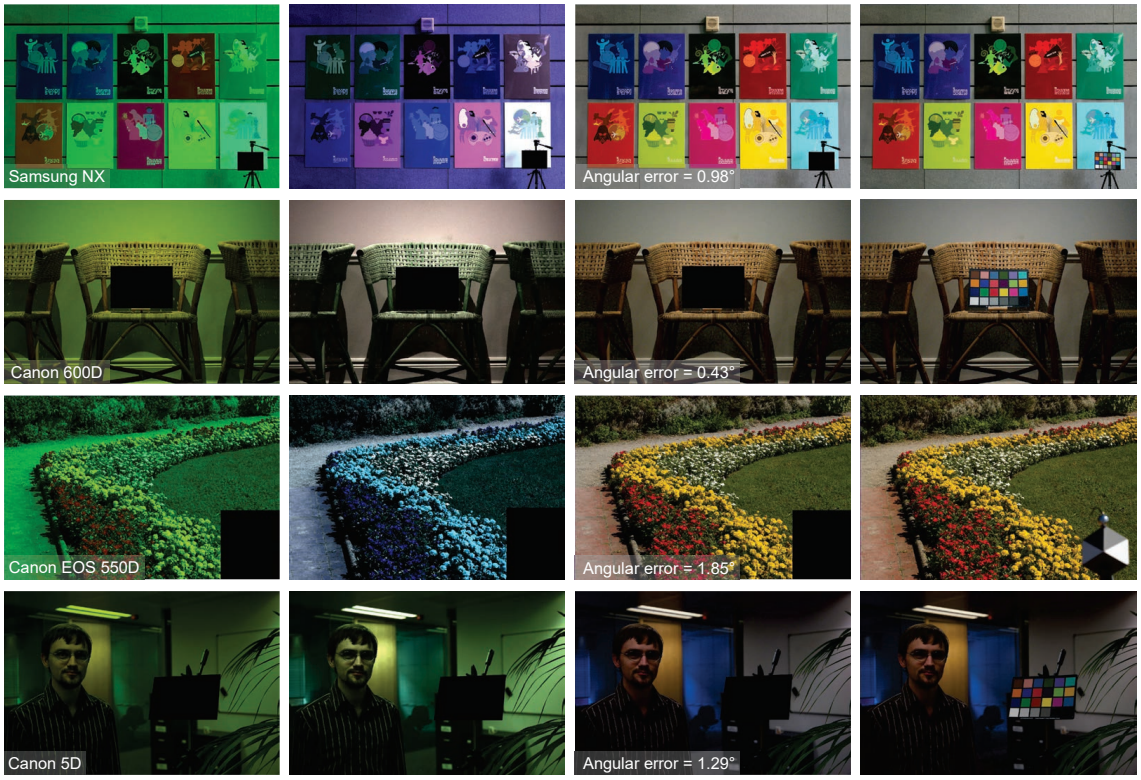


Figure 4.6: Qualitative results of our method. (A) Input raw-RGB images. (B) After mapping images in (A) to the learned space. (C) After correcting images in (A) based on our estimated illuminants. (D) Corrected by ground truth illuminants. Shown images are rendered in the sRGB color space by the camera imaging pipeline in [191] to aid visualization.

Table 4.6: Angular errors on the INTEL-TUT dataset [37]. Methods highlighted in gray are trained/tuned for each camera sensor (i.e., sensor-specific models). The lowest errors are highlighted in yellow.

INTEL-TUT Dataset	GW [60]	SoG [120]	2nd-order GE [362]	PCA-based B/W Colors [77]	1st-order GE [362]	APAP [23]	Ours trained on NUS and Cube+	Ours trained on NUS and Gehler-Shi
Mean	4.77	4.99	4.82	4.65	4.62	4.30	3.76	3.82
Median	3.75	3.63	2.97	3.39	2.84	2.44	2.75	2.81
Best 25%	0.99	1.08	1.03	0.87	0.94	0.69	0.81	0.87
Worst 25%	10.29	11.20	11.96	10.75	11.46	11.30	8.40	8.65

4.4 Summary

We have proposed a deep learning framework for illuminant estimation. Unlike other learning-based methods, our method is a sensor-independent and can be trained on images captured by different camera sensors. To that end, we have introduced an image-specific learnable mapping matrix that maps an input image to a new sensor-independent space. Our method relies only on color distributions of images to estimate scene illuminants. We adopted a compact color histogram that is dynamically generated by our new RGB-uv histogram block. Our method achieves good results on images captured by new camera sensors that have not been used in the training process.

5 Sensor-Independent Convolutional Color Constancy

In this chapter, we present "Cross-Camera Convolutional Color Constancy" (C5)[1], a learning-based method, trained on images from multiple cameras, that accurately estimates a scene's illuminant color from raw images captured by a new camera previously unseen during training. C5 is a hypernetwork-like extension of the convolutional color constancy (CCC) approach: C5 learns to generate the weights of a CCC model that is then evaluated on the input image, with the CCC weights dynamically adapted to different input content. Unlike prior cross-camera color constancy models, which are usually designed to be agnostic to the spectral properties of test-set images from unobserved cameras, C5 approaches this problem through the lens of transductive inference: additional unlabeled images are provided as input to the model at test time, which allows the model to calibrate itself to the spectral properties of the test-set camera during inference. C5 achieves state-of-the-art accuracy for cross-camera color constancy on several datasets, is fast to evaluate (∼7 and ∼90 ms per image on a GPU or CPU, respectively), and requires little memory (∼2

[1]Work was done while Mahmoud was an intern at Google; this work is under review in IEEE International Conference on Computer Vision (ICCV) 2021. A preprint version is available in [12]: Mahmoud Afifi, Jonathan T. Barron, Chloe LeGendre, Yun-Ta Tsai, and Francois Bleibel. Cross-Camera Convolutional Color Constancy. arXiv preprint 2020.

Input query image & additional images | Result of C5 | Input query image & additional images | Result of C5

Figure 5.1: Our C5 model exploits the colors of additional images captured by the new camera model to generate a specific color constancy model for the input image. The shown images were captured by *unseen* DSLR and smartphone camera models [215] that were not included in the training stage.

MB), and, thus, is a practical solution to the problem of calibration-free automatic white balance for mobile photography. The source code of this work is available on GitHub: https://github.com/mahmoudnafifi/C5.

5.1 Introduction

As discussed in Chapter 3, one simple heuristic applied to the color constancy problem is the GW assumption: that colors in the world tend to be neutral gray and that the color of the illuminant can, therefore, be estimated as the average color of the input image [60]. This GW method and its related techniques have the convenient property that they are *invariant* to much of the spectral sensitivity differences among camera sensors and, therefore, very well-suited to the cross-camera task. If camera A's red channel is twice as sensitive as camera B's red channel, then a scene captured by camera A will have an average red intensity that is twice that of the scene captured by camera B, and so GW will produce identical output images (though this assumes that the spectral response of A and B are identical up to a scale factor, which is rarely the case in practice).

However, current state-of-the-art learning-based methods for color constancy rarely ex-

hibit this property, because they often learn things like the precise distribution of likely illuminant colors (a consequence of black-body illumination and other scene lighting regularities) and are, therefore, sensitive to any mismatch between the spectral sensitivity of the camera used during training and that of the camera used at test time.

As discussed in Chapter 4, there is often significant spectral variation across camera models (as shown in Fig. 5.2), this sensitivity of existing methods is problematic when designing practical white-balance solutions. Training a learning-based algorithm for a new camera requires collecting hundreds, or thousands, of images with ground-truth illuminant color labels (in practice: images containing a color chart), a burdensome task for a camera manufacturer or platform that may need to support hundreds of different camera models. However, the GW assumption still holds surprisingly well across sensors—if given several images from a particular camera, one can do a reasonable job of estimating the range of likely illuminant colors (as can also be seen in Fig. 5.2).

Contribution This chapter presents C5, a cross-camera convolutional color constancy model. Our model addresses this problem of high-accuracy cross-camera color constancy through the use of two concepts. First, our system is constructed to take as input not just a single test-set image, but also a small set of additional images from the test set, which are arbitrarily-selected, unlabeled, and not white balanced. This allows the model to calibrate itself to the spectral properties of the test-time camera during inference. We make no assumptions about these additional images except that they come from the same camera as the "target" test set image and they contain some content (not all black or white images). In practice, these images could simply be randomly chosen images from the photographer's "camera roll", or they could be a fixed set of ad hoc images of natural scenes taken once by the camera manufacturer—because these images do not need to

be annotated, they are abundantly available. Second, our system is constructed as a *hypernetwork* [153] around an existing color constancy model. The target image and the additional images are used as input to a deep neural network whose output is the weights of a smaller color constancy model, and those generated weights are then used to estimate the illuminant color of the target image. Our approach is also closely related to the work on domain adaptation [87,327] and transfer learning [296], both of which attempt to enable learning-based models to cope with differences between training and test data. Our system is trained using labeled (and unlabeled) images from multiple cameras, but at test time our model is able to look at a set of (unlabeled) test set images from a new camera. Our hypernetwork is able to infer the likely spectral properties of the new camera that produced the test set images (much as the reader can infer the likely illuminant colors of a camera from only looking at aggregate statistics, as in Fig. 5.2) and produce a small model that has been dynamically adapted to produce accurate illuminant estimates when applied to the target image. By learning the weights of an FFCC-like model using other test-set images, C5 is able to dynamically adapt to the domain of unseen camera models, thereby allowing a single learned color constancy model to be applied to a wide variety of disparate datasets (see Fig. 5.1). By leveraging the fast convolutional approach already in-use by FFCC, C5 is able to retain the computational efficiency and low memory footprint of FFCC, while achieving state-of-the-art results compared to other camera-independent color constancy methods.

5.2 Methodology

We call our system "cross-camera convolutional color constancy" (C5), because it builds upon the existing "convolutional color constancy" (CCC) model [44] and its successor "fast Fourier color constancy" (FFCC) [45], but embeds them in a multi-input hypernetwork to

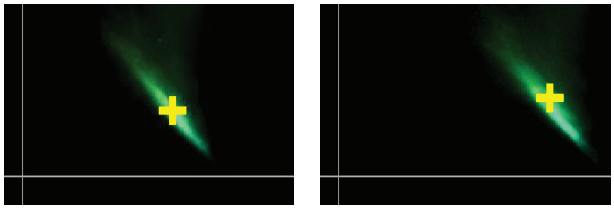


Figure 5.2: A visualization of uv log-chroma histograms ($u = \log(g/r)$, $v = \log(g/b)$) of images from two different cameras averaged over many images (green), as well as the uv coordinate of the mean of ground-truth illuminants over the entire scene set (yellow) [77]. The "positions" of these histograms change significantly across the two camera sensors because of their different spectral sensitivities, which is why many color constancy models generalize poorly across cameras.

enable accurate cross-camera performance. These CCC/FFCC models work by learning to perform localization within a log-chroma histogram space, such as those shown in Fig. 5.2. Here, we present a convolutional CC model that is a simplification of those presented in the original work [44] and its FFCC follow-up [45]. This simple convolutional model will be a fundamental building block that we will use in our larger neural network.

The image formation model behind CCC/FFCC (and most CC models) is that each pixel of the observed image is assumed to be the element-wise product of some "true" white-balanced image (or equivalently, the observed image if it were imaged under a white illuminant) and some illuminant color:

$$\forall_k \, \mathbf{c}^{(k)} = \mathbf{w}^{(k)} \circ \boldsymbol{\ell}, \tag{5.1}$$

where $\mathbf{c}^{(k)}$ is the observed color of pixel k, $\mathbf{w}^{(k)}$ is the true color of the pixel, and $\boldsymbol{\ell}$ is the color of the illuminant, all of which are 3-vectors of RGB values. CC algorithms traditionally use the input image $\{\mathbf{c}^{(k)}\}$ to produce an estimate of the illuminant $\hat{\boldsymbol{\ell}}$ that is then divided (element-wise) into each observed color to produce an estimate of the true

color of each pixel $\{\hat{\mathbf{w}}^{(k)}\}$.

CCC defines two log-chroma measures for each pixel, which are simply the log of the ratio of two color channels:

$$u^{(k)} = \log\left(c_g^{(k)}/c_r^{(k)}\right), \quad v^{(k)} = \log\left(c_g^{(k)}/c_b^{(k)}\right). \tag{5.2}$$

As discussed in Chapter 3 Finlayson and Hordley showed that this log-chrominance representation of color means that illuminant changes (i.e. element-wise scaling by $\boldsymbol{\ell}$) can be modeled simply as additive offsets to this uv representation [115]. We then construct a 2D histogram of the log-chroma values of all pixels:

$$\mathbf{N}_0(u,v) = \sum_k ||\mathbf{c}^{(k)}||_2 \left[\left|u^{(k)} - u\right| \leq \epsilon \wedge \left|v^{(k)} - v\right| \leq \epsilon\right]. \tag{5.3}$$

This is simply a histogram over all uv coordinates of size (64×64) written out using Iverson brackets, where ϵ is the width of a histogram bin, and where each pixel is weighted by its overall brightness under the assumption that bright pixels provide more actionable signal than dark pixels. We use a histogram bin width $\epsilon = (b_{\texttt{max}} - b_{\texttt{min}})/n$, where $b_{\texttt{max}}$ and $b_{\texttt{min}}$ are the histogram boundary values. In our experiments, we set $b_{\texttt{min}}$ and $b_{\texttt{max}}$ to -2.85 and 2.85, respectively. As was done in FFCC, we construct two histograms: one of pixel intensities $\mathbf{N}_0$, and one of gradient intensities $\mathbf{N}_1$ (constructed analogously to Equation 5.3).

These histograms of log-chroma values exhibit a useful property: element-wise multiplication of the RGB values of an image by a constant results in a *translation* of the resulting log-chrominance histograms. The core insight of CCC is that this property allows CC to be framed as the problem of "localizing" a log-chroma histogram in this uv histogram-space [44]—because every uv location in N corresponds to a (normalized) illuminant color $\boldsymbol{\ell}$, the problem of estimating $\boldsymbol{\ell}$ is *reducible* (in a computability sense) to the problem of estimating a uv coordinate. This can be done by discriminatively training

a "sliding window" classifier much as one might train, say, a face-detection system: the histogram is convolved with a (learned) filter and the location of the argmax is extracted from the filter response, and that argmax corresponds to uv value that is (the inverse of) an estimated illumination location.

We adopt a simplification of the convolutional structure used by FFCC [45]:

$$\mathbf{P} = \text{softmax}\left(\mathbf{B} + \sum_i \left(\mathbf{N}_i * \mathbf{F}_i\right)\right), \tag{5.4}$$

where $\{\mathbf{F}_i\}$ and $\mathbf{B}$ are filters and a bias, respectively, which have the same shape as $\mathbf{N}_i$ (unlike FFCC, we do not include a "gain" multiplier, as it did not result in a uniformly improved performance). Each histogram $\mathbf{N}_i$ is convolved with each filter $\mathbf{F}_i$ and summed across channels (a "conv" layer) and $\mathbf{B}$ is added to that summation, which collectively biases inference towards uv coordinates that correspond to common illuminants, such as black body radiation. As was done in FFCC, this convolution is accelerated through the use of FFTs, though, unlike FFCC, we use a non-wrapped histogram and, thus, non-wrapped filters and bias. This sacrifices speed for simplicity and accuracy and avoids the need for the complicated "de-aliasing" scheme used by FFCC which is not compatible with the convolutional neural network structure that we will later introduce.

The output of the softmax $\mathbf{P}$ is effectively a "heat map" of what illuminants are likely, given the distribution of pixel and gradient intensities reflected in $\mathbf{N}$ and in the prior $\mathbf{B}$, from which, we extract a "soft argmax" by taking the expectation of u and v with respect to $\mathbf{P}$:

$$\hat{\ell}_u = \sum_{u,v} u\mathbf{P}(u,v)\,, \quad \hat{\ell}_v = \sum_{u,v} v\mathbf{P}(u,v). \tag{5.5}$$

Equation 5.5 is equivalent to estimating the mean of a fitted Gaussian, in the uv space, weighted by $\mathbf{P}$. Because the absolute scale of $\boldsymbol{\ell}$ is assumed to be irrelevant or unrecoverable in the context of CC, after estimating $(\hat{\ell}_u, \hat{\ell}_v)$, we produce an RGB illuminant estimate $\hat{\boldsymbol{\ell}}$

that is simply the unit vector whose log-chroma values match our estimate:

$$\hat{\boldsymbol{\ell}} = \left(\exp\left(-\hat{\ell}_u\right)/z,\, 1/z,\, \exp\left(-\hat{\ell}_v\right)/z\right), \tag{5.6}$$

$$z = \sqrt{\exp\left(-\hat{\ell}_u\right)^2 + \exp\left(-\hat{\ell}_v\right)^2 + 1}. \tag{5.7}$$

A convolutional color constancy model is then trained by setting $\{\mathbf{F}_i\}$ and $\mathbf{B}$ to be free parameters which are then optimized to minimize the difference between the predicted illuminant $\hat{\boldsymbol{\ell}}$ and the ground-truth illuminant $\boldsymbol{\ell}^*$.

5.2.1 Architecture

With our baseline CCC/FFCC-like model in place, we can now construct our cross-camera convolutional color constancy model (C5), which is a deep architecture in which CCC is a component. Both CCC and FFCC operate by learning a single fixed set of parameters consisting of a single filter bank $\{\mathbf{F}_i\}$ and bias $\mathbf{B}$. In contrast, in C5 the filters and bias are parameterized as the output of a deep neural network (parameterized by weights θ) that takes as input not just log-chrominance histograms for the image being color-corrected (which we will refer to as the "query" image), but also log-chrominance histograms from several other randomly selected input images (but with no ground-truth illuminant labels) from the test set. By using a generated filter and bias from additional images taken from the query image's camera (instead of using a fixed filter and bias as was done in previous work) our model is able to automatically "calibrate" its CCC model to the specific sensor properties of the query image. This can be thought of as a hypernetwork [153], wherein a deep neural network emits the "weights" of a CCC model, which is itself a shallow neural network. This approach also bears some similarity to a Transformer approach, as a CCC model can be thought of as "attending" to certain parts of a log-chroma histogram, and so our neural network can be viewed as a sort of self-attention mechanism [363]. See Fig. 5.3

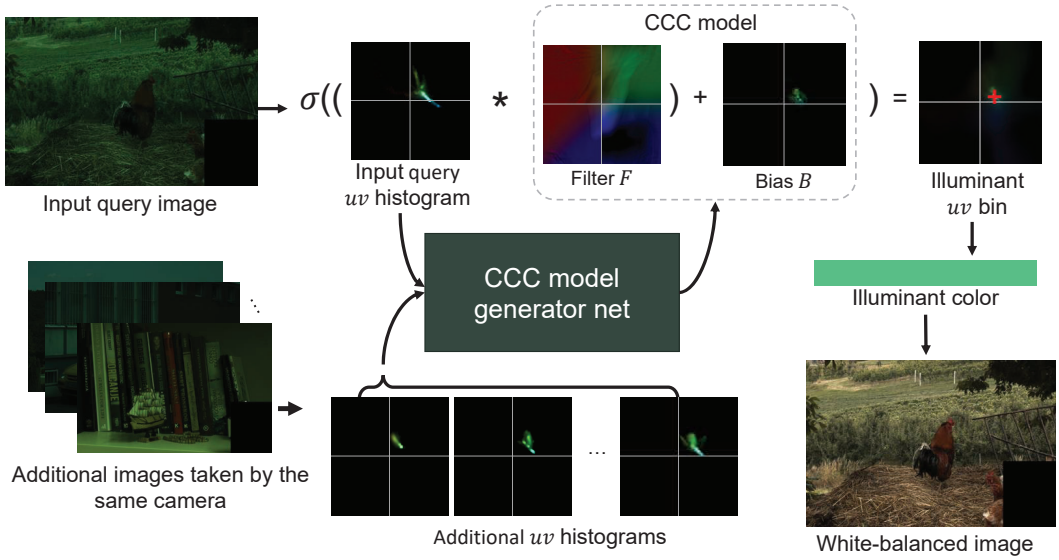


Figure 5.3: An overview of our C5 model. The uv histograms for the input query image and a variable number of additional input images taken from the same sensor as the query are used as input to our neural network, which generates a filter bank $\{\mathbf{F}_i\}$ (here shown as one filter) and a bias B, which are the parameters of a conventional CCC model [44]. The query uv histogram is then convolved by the generated filter and shifted by the generated bias to produce a heat map, whose argmax is the estimated illuminant [44].

for a visualization of this data flow.

At the core of our model is the deep neural network that takes as input a set of log-chroma histograms and must produce as output a CCC filter bank and bias map. For this we use a multi-encoder-multi-decoder U-Net-like architecture [323]. The first encoder is dedicated to the "query" input image's histogram, while the rest of the encoders take as input the histograms corresponding to the additional input images. To allow the network to reason about the set of additional input images in a way that is insensitive to their ordering, we adopt the permutation invariant pooling approach of Aittala et al. [26]: we use max pooling *across* the set of activations of each branch of the encoder. This "cross-

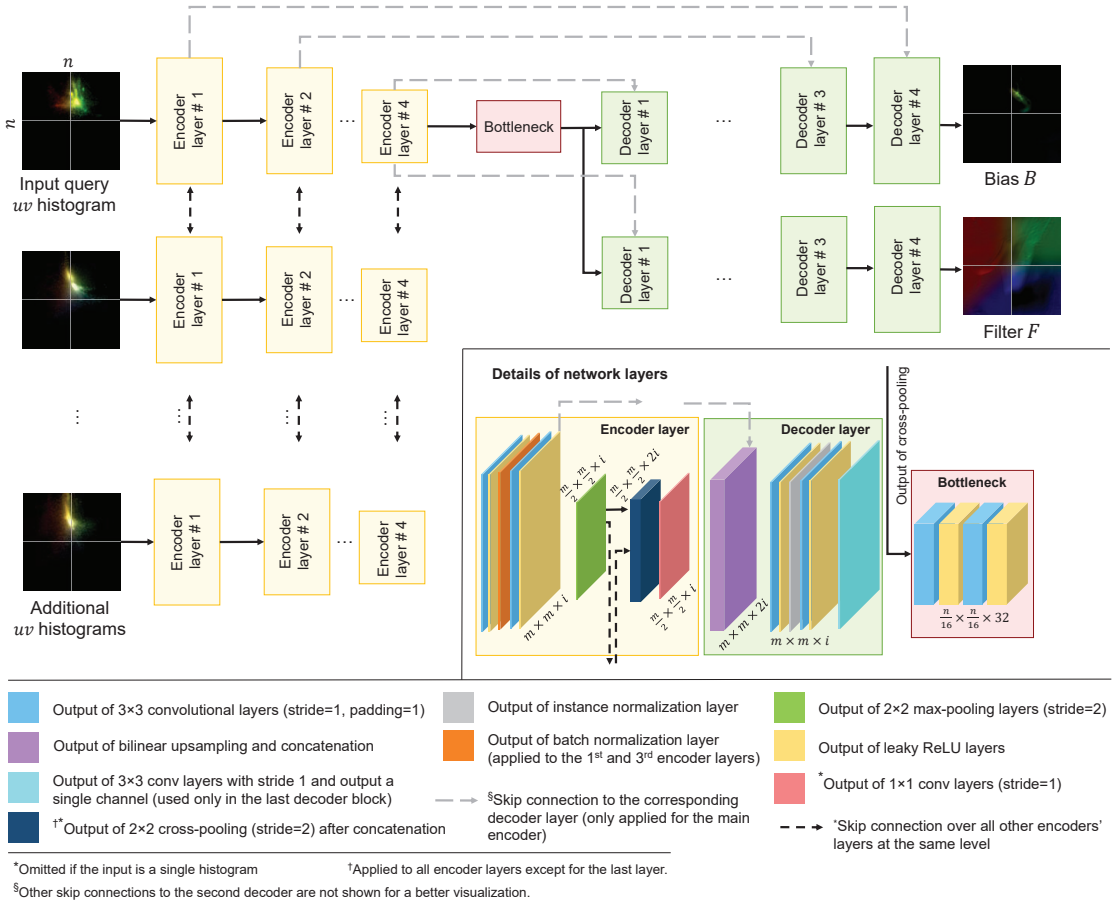


Figure 5.4: An overview of neural network architecture that emits CCC model weights. The uv histogram of the query image along with additional input histograms taken from the same camera are provided as input to a set of multiple encoders. The activations of each encoder are shared with the other encoders by performing max-pooling across encoders after each block. The cross-pooled features at the last encoder layer are then fed into two decoder blocks to generate a bias and filter bank of an CCC model for the query histogram. Each scale of the decoder is connected to the corresponding scale of the encoder for query histogram with skip connections.

pooling" gives us a single set of activations that are reflective of the set of additional input images, but are agnostic to the particular ordering of those input images. At inference time, these additional images are needed to allow the network to reason about how to use them in challenging cases. Due to the "cross-pooling," the activations of the encoders for the additional images depend on the query image, and so cannot be pre-computed for a given sensor. The cross-pooled features of the last layer of all encoders are then fed into two decoder blocks. Each decoder produces one component of our CCC model: a bias map B and two filters, $\{\mathbf{F}_0, \mathbf{F}_1\}$ (which correspond to pixel and edge histograms $\{\mathbf{N}_0, \mathbf{N}_1\}$, respectively). As per the traditional U-Net structure, we use skip connections between each level of the decoder and its corresponding level of the encoder with the same spatial resolution, but only for the encoder branch corresponding to the query input image's histogram. Each block of our encoder consists of a set of interleaved 3×3 conv layers, leaky ReLU activation, batch normalization, and 2×2 max pooling, and each block of our decoder consists of $2\times$ bilinear upsampling followed by interleaved 3×3 conv layers, leaky ReLU activation, and instance normalization. When passing our 2-channel (pixel and gradient) log-chroma histograms to our network, we augment each histogram with two extra "channels" comprising of only the u and v coordinates of each histogram, as in CoordConv [244]. This augmentation allows a convolutional architecture on top of log-chroma histograms to reason about the absolute "spatial" information associated with each uv coordinate, thereby allowing a convolutional model to be aware of the absolute color of each histogram bin. See Figure 5.4 for a detailed visualization of our architecture.

5.2.2 Training

Our model is trained by minimizing the angular error [168] between the predicted unit-norm illuminant color $\hat{\boldsymbol{\ell}}$ and the ground-truth illuminant color $\boldsymbol{\ell}^*$, as well as an additional

loss that regularizes the CCC models emitted by our network. Our loss function $\mathcal{L}(\cdot)$ is:

$$\mathcal{L}\left(\boldsymbol{\ell}^*, \hat{\boldsymbol{\ell}}\right) = \cos^{-1}\left(\frac{\boldsymbol{\ell}^* \cdot \hat{\boldsymbol{\ell}}}{\|\boldsymbol{\ell}^*\|}\right) + S\left(\{\mathbf{F}_i(\theta)\}, \mathbf{B}(\theta)\right) , \tag{5.8}$$

where $S(\cdot)$ is a regularizer that encourage the network to generate smooth filters and biases, which reduces over-fitting and improves generalization:

$$\begin{aligned} S\left(\{\mathbf{F}_i\}, \mathbf{B}\right) = \lambda_B(\|\mathbf{B} * \nabla_u\|^2 + \|\mathbf{B} * \nabla_v\|^2) \\ +\lambda_F \sum_i (\|\mathbf{F}_i * \nabla_u\|^2 + \|\mathbf{F}_i * \nabla_v\|^2) , \end{aligned} \tag{5.9}$$

where ∇_u and ∇_v are 3×3 horizontal and vertical Sobel filters, respectively, and λ_F and λ_B are multipliers that control the strength of the smoothness for the filters and the bias, respectively. This regularization is similar to the total variation smoothness prior used by FFCC [45], though here we are imposing it on the filters and bias generated by a neural network, rather than on a single filter bank and bias map. We set the multiplier hyperparameters λ_F and λ_B to 0.15 and 0.02, respectively (see Sec. 5.3.2 for ablation studies). In addition to regularizing the CCC model emitted by our network, we additionally regularize the weights of our network themselves, θ, using L2 regularization (i.e., "weight decay") with a multiplier of 5×10^{-4}. This regularization of our network serves a different purpose than the regularization of the CCC models emitted by our network—regularizing $\{\mathbf{F}_i(\theta)\}$ and $\mathbf{B}(\theta)$ prevents over-fitting by the CCC model emitted by our network, while regularizing θ prevents over-fitting by the model *generating* those CCC models.

Training is performed using the Adam optimizer [204] with hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, for 60 epochs. We use a learning rate of 5×10^{-4} with a cosine annealing schedule [247] and increasing batch-size (from 16 to 64) [262, 344] which improve the stability of training. When training our model for a particular camera model, at each iteration we randomly select a batch of training images (and their corresponding ground-truth illuminants) for use as query input images, and then randomly select m additional input images

Figure 5.5: An example of the image mapping used to augment training data. From left to right: a raw image captured by a Fujifilm X-M1 camera; the same image after white-balancing in CIE XYZ; the same image mapped into the Nikon D40 sensor space; and a real image captured by a Nikon D40 of the same scene for comparison [77].

for each query image from the training set for use as additional input images.

5.3 Experiments and Discussion

Similar to our work in Chapter 4, we used downsized raw images after applying the black-level normalization and masking out the calibration object to avoid any "leakage" during the evaluation. Excluding histogram computation time (which is difficult to profile accurately due to the expensive nature of scatter-type operations in deep learning frameworks), our method runs in ∼7 milliseconds per image on a NVIDIA GeForce GTX 1080, and ∼90 milliseconds on an Intel Xeon CPU Processor E5-1607 v4 (10M Cache, 3.10 GHz). Because our model exists in log-chroma histogram space, the uncompressed size of our entire model is ∼2 MB, small enough to easily fit within the narrow constraints of limited compute environments such as mobile phones.

5.3.1 Data Augmentation

Many of the datasets we use contain only a few images per distinct camera model (e.g. the NUS dataset [77]) and this poses a problem for our approach as neural networks generally require significant amounts of training data. To address this, we use a data augmentation procedure in which images taken from a "source" camera model are mapped into the color space of a "target" camera. To perform this mapping, we first white balance each raw source image using its ground-truth illuminant color, and then transform that white-balanced raw image into the device-independent CIE XYZ color space [82] using the CST matrices provided in each DNG file [4]. Then, we transform the CIE XYZ image into the target sensor space by inverting the CST of an image taken from the target camera dataset. Instead of randomly selecting an image from the target dataset, we use the correlated color temperature of each image and the capture exposure setting to match source and target images that were captured under roughly the same conditions. This means that "daytime" source images get warped into the color space of "daytime" target images, etc., and this significantly increases the realism of our synthesized data.

After mapping the source image to the target white-balanced sensor space, we randomly sample from a cubic curve that has been fit to the rg chromaticity of illuminant colors in the target sensor. Lastly, we apply a chromatic adaptation to generate the augmented image in the target sensor space. This chromatic adaptation is performed by multiplying each color channel of the white-balanced raw image, mapped to the target sensor space, with the corresponding sampled illuminant color channel value; see Figure 5.5 for an example. Additional details can be found in Appendix B. This augmentation allows us to generate additional training examples to improve the generalization of our model. More details are provided in Sec. 5.3.3.

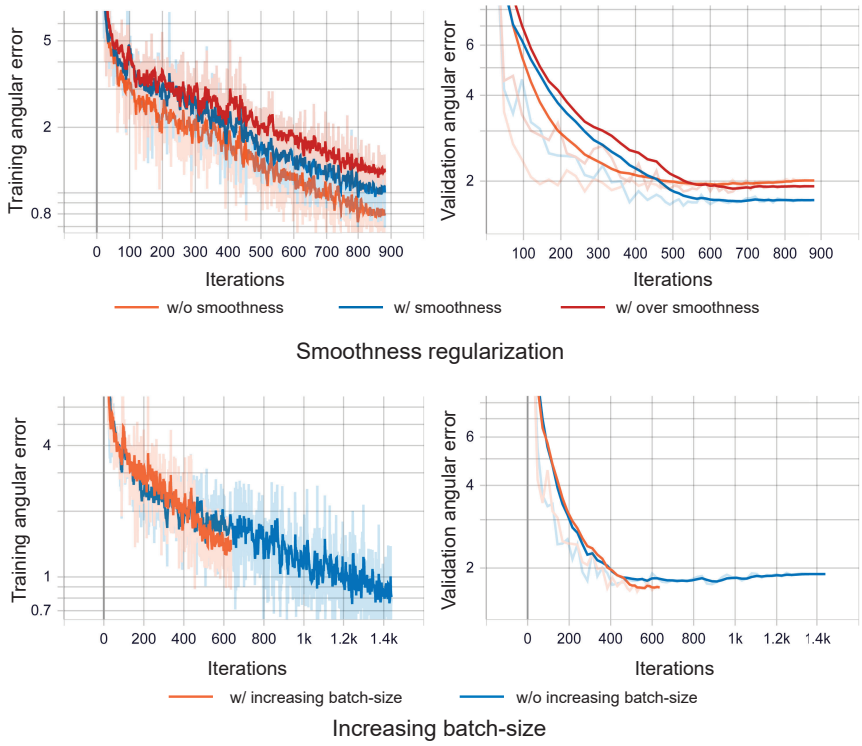


Figure 5.6: The impact of smoothness regularization and of increasing the batch size during training on training/validation accuracy. We show the training/validation angular error of training our network on the Gehler-Shi dataset [132] for camera-specific color constancy. We set $\lambda_F = 0.15$, $\lambda_B = 0.02$ for the experiment labeled with 'w/ smoothness', while we used $\lambda_F = 1.85$, $\lambda_B = 0.25$ for the experiment labeled with 'over smoothness' and $\lambda_F = 0$, $\lambda_B = 0$ for the 'w/o smoothness' experiments.

5.3.2 Ablations Studies

In the following ablation experiments, we used the Cube+ dataset [41] as our test set and trained our network with seven encoders (i.e., $m = 7$) using the following training sets: the NUS dataset [77], the Gehler-Shi dataset [132], and the augmented images after excluding any scene/sensors of the test set. Table 5.1 shows the results obtained by models trained

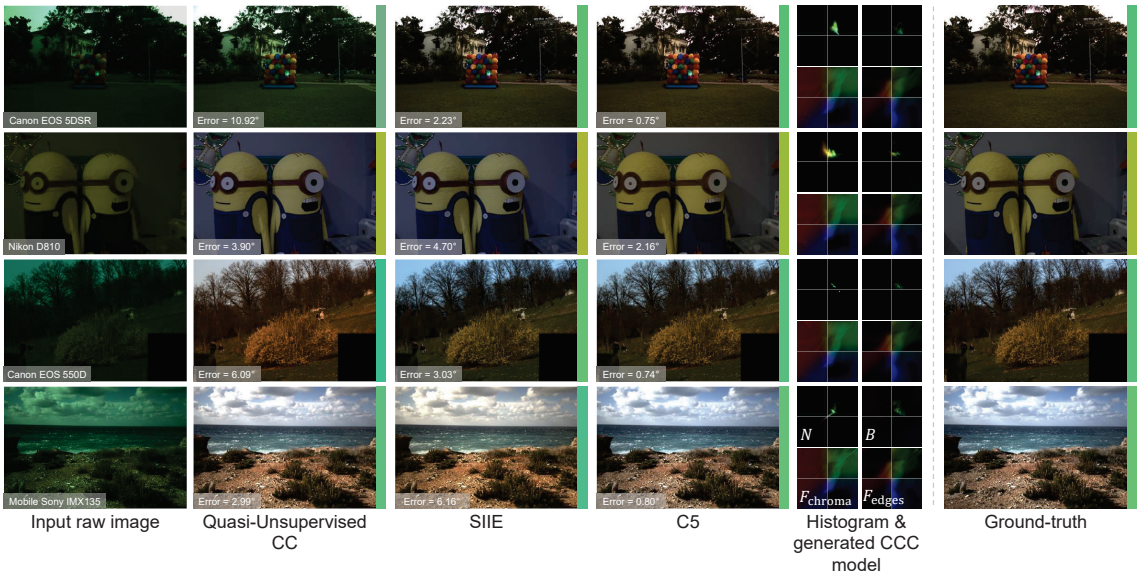


Figure 5.7: In this figure, we visualize the performance of our C5 model alongside other camera-independent models: "quasi-unsupervised CC" [51] and SIIE (Chapter 4). Despite not having seen any images from the test-set camera during training, C5 is able to produce accurate illuminant estimates. The intermediate CCC filters and biases produced by C5 are also visualized.

using different histogram sizes, using different values of the smoothness factors λ_B and λ_F, with and without increasing the batch-size during training, and with and without the histogram gradient intensity and the extra uv augmentation channels. Each experiment was repeated ten times and the arithmetic mean and standard deviation of each error metric are reported.

Figure 5.6 shows the effect of the smoothness regularization and increasing the batch-size during training on a small training set. We use the first fold of the Gehler-Shi dataset [132] as our validation set and the remaining two folds are used for training. In the figure we plot the angular error on the training and validation sets. Each model was trained for 60 epochs as a camera-specific color constancy model (i.e., $m = 1$ and without

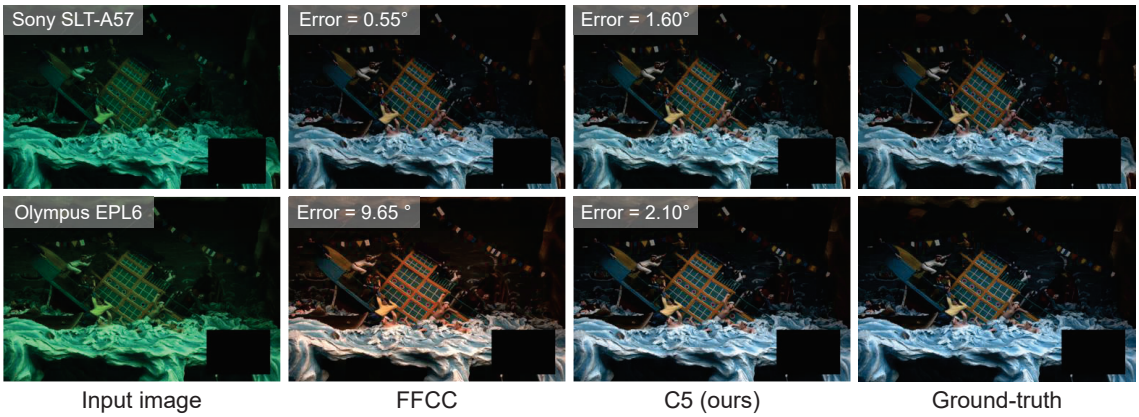


Figure 5.8: In this figure, we compare our C5 model against FFCC [45] on cross-sensor generalization using test-set Sony SLT-A57 images from the NUS dataset [77]. If FFCC is trained and tested on images from the same camera, it performs well, as does C5 (top row). But if FFCC is instead tested on a different camera, such as the Olympus EPL6, it generalizes poorly, while C5 retains its performance (bottom row).

using additional training camera models). As can be seen in Fig. 5.6, the smoothness regularization improves the generalization on the test set and increasing the batch size helps the network to reach a lower optimum.

5.3.3 Results and Comparisons

We validate our model using four public datasets consisting of images taken from one or more camera models: the Gehler-Shi dataset (568 images, two cameras) [132], the NUS dataset (1,736 images, eight cameras) [77], the INTEL-TAU dataset[2] (7,022 images, three cameras) [215], and the Cube+ dataset (2,070 images, one camera) [41] which has a separate 2019 "Challenge" test set [39].

As done in Chapter 4, we measure performance by reporting the error statistics com-

[2]This is an updated version of the INTEL-TUT used in Chapter 4 for evaluation.

monly used by the community: the mean, median, trimean, and arithmetic means of the first and third quartiles ("best 25%" and "worst 25%") of the angular error between the estimated illuminant and the true illuminant. To evaluate our model's performance at generalizing to new camera models not seen during training, we adopt a leave-one-out cross-validation evaluation approach: for each dataset, we exclude all scenes and cameras used by the test set from our training images.

To evaluate the improvement of using the additional input images, we report multiple versions of our model in which we vary m, the number of the additional images (and encoders) used ($m = 1$ means that only the query image is used as input).

As our method randomly selects the additional images, each experiment is repeated ten times and we reported the arithmetic mean of each error metric.

For a fair comparison with FFCC [45], we trained FFCC using the same leave-one-out cross-validation evaluation approach. Results can be seen in Tables 5.2–5.4 and qualitative comparisons are shown in Figs. 5.7 and 5.8. Even when compared with prior sensor-independent techniques [51] and our previous SIIE method (Chapter 4), C5 achieves state-of-the-art performance when using ($m \geq 7$) images, as demonstrated in Tables 5.2–5.4.

When evaluating on the two Cube+ [39,41] test sets and the INTEL-TAU [215] dataset in Tables 5.2 and 5.3, we train our model on the NUS [77] and Gehler-Shi [132] datasets. When evaluating on the Gehler-Shi [132] and the NUS [77] datasets in Table 5.4, we train C5 using the INTEL-TAU dataset [215], the Cube+ dataset [41], and one of the Gehler-Shi [132] and the NUS [77] datasets after excluding the testing dataset.

The one deviation from this procedure is for the NUS result labeled "CS", where for a fair comparison with our SIIE method, proposed in Chapter 4, we report our results with their cross-sensor (CS) evaluation, in which we only excluded images of the test camera, and repeated this process over all cameras in the dataset.

For experiments labeled "w/aug" in Table 5.2–5.4, we augmented the data used to train the model, adding 5,000 augmented examples generated as described in Sec. 5.3.1. In this process, we used only cameras of the training sets of each experiment as "target" cameras for augmentation, which has the effect of mixing the sensors and scene content from the training sets only. For instance, when evaluating on the INTEL-TAU [215] dataset, our augmented images simulate the scene content of the NUS [77] dataset as observed by sensors of the Gehler-Shi [132] dataset, and vice-versa.

Unless otherwise stated, in our experiments varying m, the additional input images are randomly selected, but from the same camera model as the test image. This setting is meant to be equivalent to the real-world use case in which the additional images provided as input are, say, a photographer's previously-captured images that are already present on the camera during inference. However, for the "Cube+ Challenge" table, we provide an additional set of experiments in which the set of additional images are chosen according to some heuristic, rather than randomly. We identified the 20 test-set images with the lowest variation of uv chroma values ("dull images"), the 20 test-set images with the highest variation of uv chroma values ("vivid images"), and we show that using vivid images produces lower error rates than randomly-chosen or dull images. This makes intuitive sense, as one might expect colorful images to be a more informative signal as to the spectral properties of previously-unobserved camera. We also show results where the additional images are taken from a different camera than the test-set camera, and show that this results in error rates that are higher than the $m = 1$ case, as one might expect.

We did not include the "gain" multiplier, originally proposed in FFCC [45], in the main method section as it did not result in a consistent improved performance over all error metrics and datasets. Here, we report results with and without using the gain multiplier map. This gain multiplier map can be generated by our network by adding an additional

decoder network with skip connections from the query encoder. This modification increases our model size from 1.74 MB to 1.97 MB using $m = 7$. Based on this modification, our convolutional structure can now be described as:

$$\mathbf{P} = \text{softmax}\left(\mathbf{B} + \mathbf{G} \circ \sum_i \left(\mathbf{N}_i * \mathbf{F}_i\right)\right), \tag{5.10}$$

where $\{\mathbf{F}_i\}$, $\mathbf{B}$, and $\mathbf{G}$ are filters, a bias map $\mathbf{B}(i,j)$, and the gain multiplier map $\mathbf{G}(i,j)$, respectively. We also change the smoothness regularizer to include the generated gain multiplier as follows:

$$\begin{aligned} S\left(\{\mathbf{F}_i\}, \mathbf{B}, \mathbf{G}\right) = \lambda_B(\|\mathbf{B} * \nabla_u\|^2 + \|\mathbf{B} * \nabla_v\|^2) \\ +\lambda_G(\|\mathbf{G} * \nabla_u\|^2 + \|\mathbf{G} * \nabla_v\|^2) \\ +\lambda_F \sum_i (\|\mathbf{F}_i * \nabla_u\|^2 + \|\mathbf{F}_i * \nabla_v\|^2), \end{aligned} \tag{5.11}$$

where ∇_u and ∇_v are 3×3 horizontal and vertical Sobel filters, respectively, and λ_F, λ_B, λ_G are scalar multipliers to control the strength of the smoothness of each of the filters, the bias, and the gain, respectively. The results of using the additional gain multiplier map are reported in Table 5.5.

We further trained and tested our C5 model using the INTEL-TAU dataset evaluation protocols [215]. Specifically, the INTEL-TAU dataset introduced two different evaluation protocols: (i) the cross-validation protocol, where the model is trained using a 10-fold cross-validation scheme of images taken from three different camera models, and (ii) the camera invariance evaluation protocol, where the model is trained on a single camera model and then tested on another camera model. This camera invariance protocol is equivalent to the CS evaluation used in Chapter 4, as the models are trained and tested on the same scene set, but with different camera models in the training and testing phases. See Table 5.6 for comparison with other methods using the INTEL-TAU evaluation protocols. In

Table 5.6, we also show the results of our C5 model trained on the NUS and Gehler-Shi datasets with augmentation (i.e., our camera-independent model) for completeness.

Our C5 model achieves reasonable accuracy when used as a camera-specific model. In this scenario, we trained our model on training images captured by the same test camera model with a single encoder (i.e., $m = 1$). We found that $n = 128$, using the gain multiplier map $G(i, j)$, achieves the best camera-specific results. We report the results of our camera-specific models in Table 5.7.

5.4 Summary

We have presented C5, a cross-camera convolutional color constancy method. By embedding the existing state-of-the-art CCC model [44, 45] into a multi-input hypernetwork approach, C5 can be trained on images from multiple cameras, but at test time synthesize weights for a CCC-like model that is dynamically calibrated to the spectral properties of the previously-unseen camera of the test-set image. Extensive experimentation demonstrates that C5 achieves state-of-the-art performance on cross-camera color constancy for several datasets. By enabling accurate illuminant estimation without requiring the tedious collection of labeled training data for every particular camera, we hope that C5 will accelerate the widespread adoption of learning-based white balance by the camera industry.

Table 5.1: Results of ablation studies. The shown results were obtained by training our network on the NUS [77] and the Gehler-Shi datasets [132] with augmentation, and testing on the Cube+ dataset [41]. In this set of experiments, we used seven encoders (i.e., six additional histograms). Note that none of the training data includes any scene/sensor from the Cube+ dataset [41]. For each set of experiments, we highlight the lowest errors in yellow.

	Mean	**Med.**	**B. 25%**	**W. 25%**	**Tri.**
	Histogram bin size, n				
$n = 16$	2.28±0.01	1.81±0.03	0.65±0.01	4.72±0.02	1.91±0.02
$n = 32$	2.02±0.01	1.44±0.01	0.44±0.01	4.66±0.01	1.86±0.03
$n = 64$	1.87±0.00	1.27±0.01	0.41±0.01	4.36±0.01	1.40±0.01
$n = 128$	2.03±0.00	1.42±0.01	0.40±0.00	4.70±0.01	1.54±0.01
	Smoothness factors, λ_B and λ_F ($n = 64$)				
$\lambda_B = 0, \lambda_F = 0$	2.07±0.01	1.42±0.01	0.47±0.01	4.67±0.01	1.57±0.01
$\lambda_B = 0.005, \lambda_F = 0.035$	1.95±0.00	1.31±0.01	0.40±0.00	4.57±0.01	1.47±0.01
$\lambda_B = 0.02, \lambda_F = 0.15$	1.87±0.00	1.27±0.01	0.41±0.01	4.36±0.01	1.40±0.01
$\lambda_B = 0.10, \lambda_F = 0.75$	2.11±0.00	1.55±0.01	0.48±0.00	4.70±0.01	1.66±0.01
$\lambda_B = 0.25, \lambda_F = 1.85$	2.23±0.00	1.61±0.01	0.53±0.00	5.04±0.01	1.77± 0.01
	Increasing batch size ($n = 64$)				
w/o increasing	1.93±0.00	1.29±0.01	0.42±0.00	4.52±0.02	1.43±0.01
w/ increasing	1.87±0.00	1.27±0.01	0.41±0.01	4.36±0.01	1.40±0.01
	Gradient histogram and uv channels ($n = 64$)				
w/o gradient histogram	2.30±0.01	1.53±0.01	0.45±0.01	5.51±0.02	1.71±0.02
w/o uv	2.03±0.01	1.45±0.01	0.44±0.01	4.63±0.02	1.56±0.01
w/ uv and gradient histogram	1.87±0.00	1.27±0.01	0.41±0.01	4.36±0.01	1.40±0.01

Table 5.2: Angular errors on the Cube+ dataset [41] and the Cube+ challenge [39]. Lowest errors are highlighted in yellow. m is the number of additional test-time images used as input, and "w/aug." indicates if our data augmentation procedure is used. See the text for additional details on model variants. C5 yields state-of-the-art performance.

Cube+ Dataset	**Mean**	**Med.**	**B. 25%**	**W. 25%**	**Tri.**	**Size (MB)**
GW [60]	3.52	2.55	0.60	7.98	2.82	-
1st-order GE [362]	3.06	2.05	0.55	7.22	2.32	-
2nd-order GE [362]	3.28	2.34	0.66	7.44	2.58	-
SoG [120]	3.22	2.12	0.43	7.77	2.44	-
Cross-dataset CC [206]	2.47	1.94	-	-	-	-
Quasi-U CC [51]	2.69	1.76	0.49	6.45	2.00	622
SIIE (Chapter 4)	2.14	1.44	0.44	5.06	-	10.3
FFCC [45]	2.69	1.89	0.46	6.31	2.08	0.22
C5 ($m = 1$)	2.60	1.86	0.55	5.89	2.10	0.72
C5 ($m = 3$)	2.28	1.50	0.59	5.19	1.74	1.05
C5 ($m = 5$)	2.23	1.52	0.56	5.11	1.71	1.39
C5 ($m = 7$)	2.10	1.38	0.49	4.97	1.56	1.74
C5 ($m = 7$, w/aug.)	1.87	1.27	0.41	4.36	1.40	1.74
C5 ($m = 9$, w/aug.)	1.92	1.32	0.44	4.44	1.46	2.09

Cube+ Challenge	**Mean**	**Med.**	**B. 25%**	**W. 25%**	**Tri.**
GW [60]	4.44	3.50	0.77	9.64	-
1st-order GE [362]	3.51	2.30	0.56	8.53	-
Quasi-U CC [51]	3.12	2.19	0.60	7.28	2.40
SIIE (Chapter 4)	2.89	1.72	0.71	7.06	-
FFCC [45]	3.25	2.04	0.64	8.22	2.09
C5 ($m = 1$)	2.70	2.00	0.61	6.15	2.06
C5 ($m = 7$)	2.55	1.63	0.54	6.21	1.79
C5 ($m = 9$)	2.24	1.48	0.47	5.39	1.62
C5 ($m = 9$, another camera model)	2.97	2.47	0.78	6.11	2.52
C5 ($m = 9$, dull images)	2.35	1.58	0.46	5.57	1.70
C5 ($m = 9$, vivid images)	2.19	1.39	0.43	5.44	1.54

Table 5.3: Angular errors on the INTEL-TAU dataset [215]. Lowest errors are highlighted in yellow. m is the number of additional test-time images used as input, and "w/aug." indicates if our data augmentation procedure is used. See the text for additional details on model variants. C5 yields state-of-the-art performance.

INTEL-TAU	**Mean**	**Med.**	**B. 25%**	**W. 25%**	**Tri.**
GW [60]	4.7	3.7	0.9	10.0	4.0
White-Patch [58]	7.0	5.4	1.1	14.6	6.2
1st-order GE [58]	5.3	4.1	1.0	11.7	4.5
SoG [120]	4.0	2.9	0.7	9.0	3.2
PCA-based B/W Colors [77]	4.6	3.4	0.7	10.3	3.7
wGE [140]	6.0	4.2	0.9	14.2	4.8
Quasi-U CC [51]	3.12	2.19	0.60	7.28	2.40
SIIE (Chapter 4)	3.42	2.42	0.73	7.80	2.64
FFCC [45]	3.42	2.38	0.70	7.96	2.61
C5 ($m = 1$)	2.99	2.18	0.66	6.71	2.36
C5 ($m = 7$)	2.62	1.85	0.54	6.05	2.00
C5 ($m = 7$, w/aug.)	2.49	1.66	0.51	5.93	1.83
C5 ($m = 9$, w/aug.)	2.52	1.70	0.52	5.96	1.86

Table 5.4: Angular errors on the Gehler-Shi dataset [132], and the NUS dataset [77]. Lowest errors are highlighted in yellow. m is the number of additional test-time images used as input, "w/aug." indicates if our data augmentation procedure is used, and "CS" refers to cross-sensor as used in Chapter 4. See the text for additional details on model variants. C5 yields state-of-the-art performance.

Gehler-Shi Dataset	**Mean**	**Med.**	**B. 25%**	**W. 25%**	**Tri.**
PCA-based B/W Colors [77]	3.52	2.14	0.50	8.74	2.47
ASM [27]	3.80	2.40	-	-	2.70
Woo *et al.* [376]	4.30	2.86	0.71	10.14	3.31
Grayness Index [313]	3.07	1.87	0.43	7.62	2.16
Cross-dataset CC [206]	2.87	2.21	-	-	-
Quasi-U CC [51]	3.46	2.23	-	-	-
SIIE (Chapter 4)	2.77	1.93	0.55	6.53	-
FFCC [45]	2.95	2.19	0.57	6.75	2.35
C5 ($m = 1$)	2.98	2.05	0.54	7.13	2.25
C5 ($m = 7$, w/aug.)	2.36	1.61	0.44	5.60	1.74
CS ($m = 9$, w/aug.)	2.50	1.99	0.53	5.46	2.03

NUS Dataset	**Mean**	**Med.**	**B. 25%**	**W. 25%**	**Tri.**
PCA-based B/W Colors [77]	2.93	2.33	0.78	6.13	2.42
Grayness Index [313]	2.91	1.97	0.56	6.67	2.13
Cross-dataset CC [206]	3.08	2.24	-	-	-
Quasi-U CC [51]	3.00	2.25	-	-	-
SIIE (CS) (Chapter 4)	2.05	1.50	0.52	4.48	
FFCC [45]	2.87	2.14	0.71	6.23	2.30
C5 ($m = 1$)	2.84	2.20	0.69	6.14	2.33
C5 ($m = 7$, w/aug.)	2.68	2.00	0.66	5.90	2.14
CS ($m = 9$, w/aug.)	2.54	1.90	0.61	5.61	2.02
C5 ($m = 9$, CS)	1.77	1.37	0.48	3.75	1.46

Table 5.5: Results of using the gain multiplier, G. For each experiment, we used $m = 7$ and $n = 64$, and trained our network using the same training data explained earlier with augmentation. Lowest errors are highlighted in yellow.

	Cube+ [41]				**Cube+ Challenge** [39]				**INTEL-TAU** [215]				**Gehler-Shi** [132]			
	Mean	**Med.**	**B. 25%**	**W. 25%**	**Mean**	**Med.**	**B. 25%**	**W. 25%**	**Mean**	**Med.**	**B. 25%**	**W. 25%**	**Mean**	**Med.**	**B. 25%**	**W. 25%**
w/o G	1.87	1.27	0.41	4.36	2.40	1.58	0.52	5.76	2.49	1.66	0.51	5.93	2.36	1.61	0.44	5.60
w/ G	1.83	1.24	0.42	4.25	2.34	1.45	0.46	5.86	2.63	1.81	0.55	6.18	2.36	1.72	0.48	5.4

Table 5.6: Results using the INTEL-TAU dataset evaluation protocols [215]. We also show the results of camera-independent methods, including our camera-independent C5 model. Lower errors for each evaluation protocol are highlighted in yellow. The best results are bold-faced.

INTEL-TAU [215]	**Mean**	**Med.**	**B. 25%**	**W. 25%**	**Tri.**
Camera-specific (10-fold cross-validation protocol [215])					
Bianco et al.'s CNN [53]	3.5	2.6	0.9	7.4	2.8
C3AE [214]	3.4	2.7	0.9	7.0	2.8
BoCF [213]	2.4	1.9	0.7	5.1	2.0
FFCC [45]	2.4	1.6	0.4	5.6	1.8
VGG-FC^4 [171]	**2.2**	1.7	0.6	**4.7**	1.8
C5 ($m = 7, n = 128$), w/ augmentation	2.33	**1.55**	**0.45**	5.57	**1.71**
Camera-specific (camera invariant protocol [215])					
Bianco et al.'s CNN [53]	3.4	2.5	0.8	7.2	2.7
C3AE [214]	3.4	2.7	0.9	7.0	2.8
BoCF [213]	2.9	2.4	0.9	6.1	2.5
VGG-FC^4 [171]	2.6	2.0	0.7	5.5	2.2
C5 ($m = 9$), w/aug.	2.45	1.82	0.53	5.46	1.95
Camera-independent					
GW [60]	4.7	3.7	0.9	10.0	4.0
White-Patch [58]	7.0	5.4	1.1	14.6	6.2
1st-order GE [58]	5.3	4.1	1.0	11.7	4.5
2nd-order GE [58]	5.1	3.8	1.0	11.3	4.2
SoG [120]	4.0	2.9	0.7	9.0	3.2
PCA-based B/W Colors [77]	4.6	3.4	0.7	10.3	3.7
wGE [140]	6.0	4.2	0.9	14.2	4.8
Quasi-U CC [51]	3.12	2.19	0.60	7.28	2.40
SIIE (Chapter 4)	3.42	2.42	0.73	7.80	2.64
C5 ($m = 7$), w/aug.	2.49	1.66	0.51	5.93	1.83

Table 5.7: Results of our C5 trained as a camera-specific model with a single encoder (i.e., $m = 1$). In these experiments, we trained our model using a three-fold cross-validation of each dataset, except for the Cube+ challenge [39], where we report our results after training our model on the Cube+ dataset [41]. We also show the results of other camera-specific color constancy methods reported in past papers. Lowest angular errors are highlighted in yellow.

Cube+ Dataset [41]	**Mean**	**Med.**	**B. 25%**	**W. 25%**	**Tri.**
Color Dog [40]	3.32	1.19	0.22	10.22	-
APAP [23]	2.01	1.36	0.38	4.71	-
Meta-AWB w/ 20 tuning images [264]	1.59	1.02	0.30	3.85	1.15 -
Color Beaver [205]	1.49	0.77	0.21	3.94	-
SqueezeNet-FC^4 [171]	1.35	0.93	0.30	3.24	1.01
FFCC [45]	1.38	0.74	0.19	3.67	0.89
MDLCC [380]	1.24	0.83	0.26	2.91	0.92
C5 ($n = 128$), w/ G	1.39	0.79	0.24	3.55	0.93

Cube+ Challenge [39]	**Mean**	**Med.**	**B. 25%**	**W. 25%**	**Tri.**
V Vuk et al., [39]	6.00	1.96	0.99	18.81	2.25
A Savchik et al., [329]	2.05	1.20	0.40	5.24	1.30
Y Qian et al., (1) [312]	2.48	1.56	0.44	6.11	-
Y Qian et al., (2) [312]	2.27	1.26	0.39	6.02	1.35
FFCC [45]	2.1	1.23	0.47	5.38	-
MHCC [163]	1.95	1.16	0.39	4.99	1.25
K Chen et al., [39]	1.84	1.27	0.39	4.41	1.32
C5 ($n = 128$), w/ G	1.72	1.07	0.36	4.27	1.15

Part III

Addressing Camera White-Balance Errors

6 Correcting Improperly White-Balanced Images

Virtually all consumer cameras have a set of pre-defined WB settings for common illuminations that the user can manually select (e.g., sunlight, incandescent, fluorescent). A problem that often arises is when the sRGB images are rendered with the incorrect WB. This is generally attributed to the incorrect WB setting being erroneously selected by the user or due to errors by the camera illuminant estimation module. As previously discussed, incorrectly white-balanced images can be extremely difficult to correct in the sRGB-rendered image due to the nonlinear color manipulation applied after the essential WB step—*even* when the correct WB settings or a scene reference white can be identified. This part of the thesis discuss this problem – namely, correcting improperly white-balanced images – and studies the impact of WB errors on the performance of different computer vision tasks.

6.1 Introduction

Recall that we showed earlier in Fig. 3.6-(E) and Fig. 3.6-(F) attempts at using the diagonal WB corrections using the color rendition chart's patch and bridge scene region as reference white, respectively. We can see that in both cases, only the selected reference white region

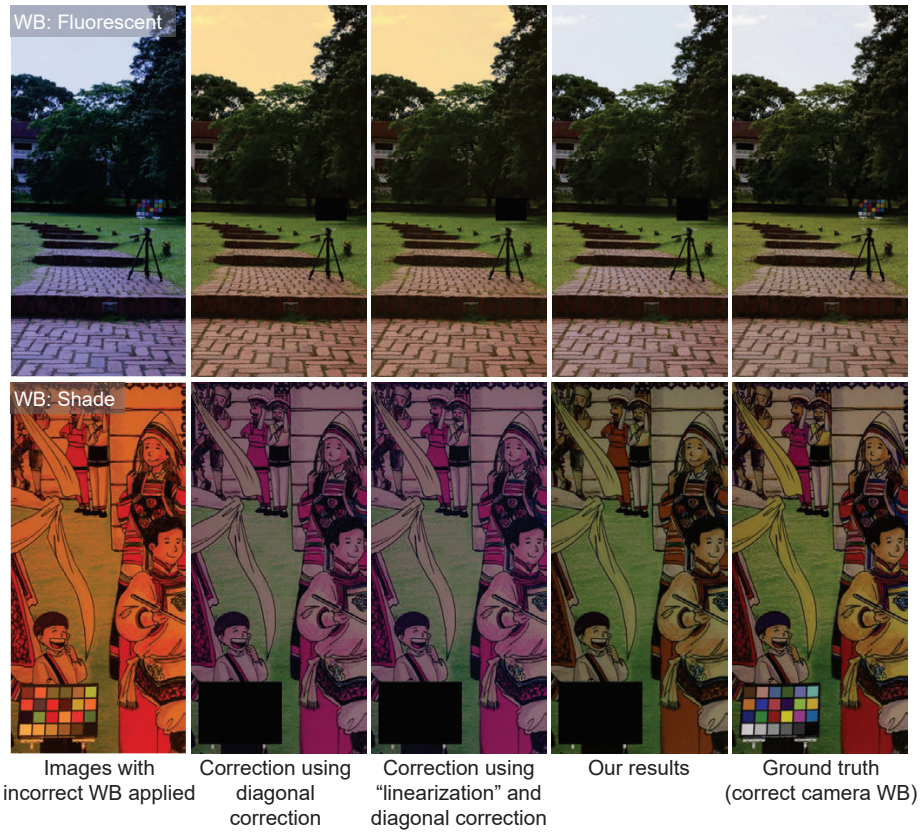


Figure 6.1: This figure shows two incorrectly white-balanced images produced by different cameras. Shown are attempts to correct the images using (1) a linear WB correction, (2) by first applying an erroneously linearization (i.e., a 2.2 gamma [31, 101]), and (3) our results. Also shown is the correct output produced by the camera.

is corrected, while the other region remains incorrect. As we clarify that WB is applied early in the processing chain, attempting to correct it using a diagonal matrix will not work. Matlab suggests using an optional pre-linearization step using a 2.2 gamma [31,101]. However, it has long been known that a 2.2 gamma does not reflect the true nature of the camera specific rendering function. There are currently *no solutions* that directly address this problem and most images like Fig. 3.6-(A) are simply discarded.

Contribution In this chapter, we propose a data-driven approach to correct images that have been improperly white-balanced[1]. As part of this effort, we have generated a new dataset of over 65,000 images from different cameras that have been rendered into a camera's output sRGB image with each camera's different pre-defined WB and photo-finishing settings. The latter is also referred to as picture styles in photography. Each incorrect white-balanced image in the dataset has a corresponding correct white-balanced sRGB image rendered to a standard picture style. Given an improperly white-balanced camera image, we outline a straightforward KNN strategy that is able to find similar incorrectly white-balanced images in the dataset. Based on these similar example images, we describe how to construct a nonlinear color correction transform that is used to remove the color cast. This idea of using a training set to search for nearest neighbors is commonly used in one-shot and prototype learning [345, 365]. Our approach gives good results – see Fig. 6.1, and generalizes well to camera makes and models not found in the training data. In addition, our solution requires a small memory overhead (less than 24 MB) and is computationally fast (less than 1.5 seconds for a 12 mega-pixel image). To the best of our knowledge, this is the first work to explicitly address correcting WB for improperly white-balanced sRGB-rendered images. The dataset and source code of this work are available on GitHub: https://github.com/mahmoudnafifi/WB_sRGB.

[1]This work was published in [20]: Mahmoud Afifi, Brian Price, Scott Cohen, and Michael S. Brown. When Color Constancy Goes Wrong: Correcting Improperly White-Balanced Images. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

6.2 Methodology

6.2.1 Method Overview

We begin with an overview of our approach followed by specific implementation details. Our method is designed with the additional constraints of fast execution and a small memory overhead to make it suitable for incorporation as a mobile app or as a software plugin. Figure 6.2 overviews our framework. Given an incorrectly white-balanced input image, denoted as $\mathbf{I}_{\text{in}}$, our goal is to compute a mapping $\mathbf{M}$ that can transform the input colors to appear as if the WB was correctly applied.

Our method relies on a large set of n training images expressed as $\mathbf{I}_{\text{t}} = \{\mathbf{I}_{\text{t}}^{(1)}, ..., \mathbf{I}_{\text{t}}^{(n)}\}$ that have been generated using the *incorrect* WB settings. Each training image has a corresponding *correct* white-balanced image (or ground truth image), denoted as $\mathbf{I}_{\text{gt}}^{(i)}$. Note that multiple training images may share the same target ground truth image. Section 6.2.2 details how we generated this dataset.

For each pair of training image $\mathbf{I}_{\text{t}}^{(i)}$ and its ground truth image $\mathbf{I}_{\text{gt}}^{(i)}$, we compute a *nonlinear* color correction matrix $\mathbf{M}^{(i)}$ that maps the incorrect image's colors to its target ground truth image's colors. The details of this mapping are discussed in Section 6.2.3.

Given an input image, we search the training set to find images with similar color distributions. This image search is performed using compact features derived from input and training image histograms as described in Section 6.2.4. Finally, we obtain a color correction matrix $\mathbf{M}$ for our input image by blending the associated color correction matrices of the similar training image color distributions, denoted as $\mathbf{M}_{\text{s}}$. This is described in Section 6.2.5.

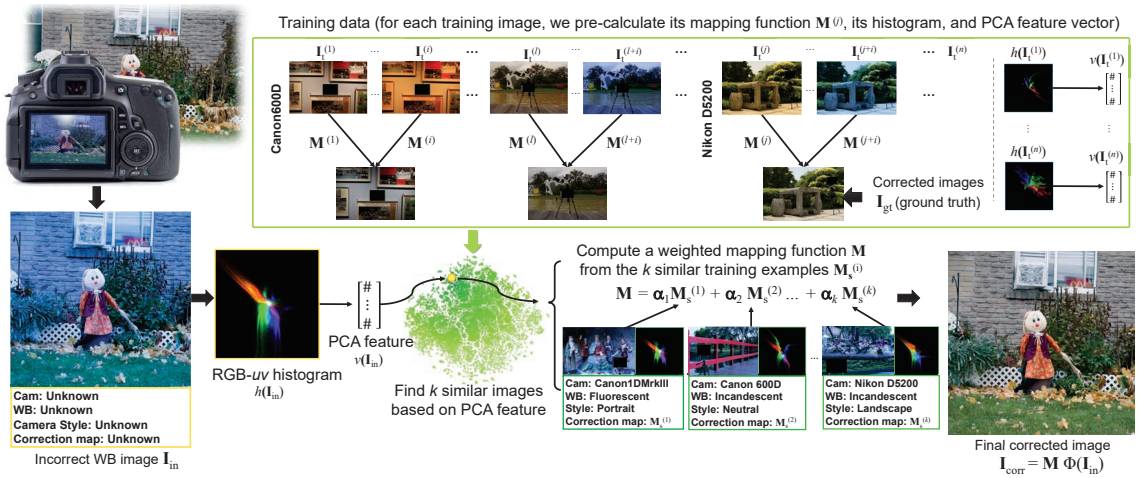


Figure 6.2: An overview diagram of our overall procedure. For the input sRGB image and our training data, we first extract the histogram feature of the input image, followed by generating a compact PCA feature to find the most similar k nearest neighbors to the input image in terms of colors. Based on the retrieved similar images, a color transform $\mathbf{M}$ is computed to correct the input image.

6.2.2 Dataset Generation

Our training images are generated from two publicly available illumination estimation datasets: the NUS dataset [77] and the Gehler dataset [132]. Images in these datasets are captured using digital single-lens reflex (DSLR) cameras with a color rendition chart placed in the scene that provides ground truth reference for illumination estimation. Since these datasets are intended for use in illumination estimation, as discussed in Chapters 4 and 5, they are captured in raw-RGB format. Because the images are in the camera's raw format, we can convert them to sRGB output emulating different WB settings and picture styles on the camera. To do this, we use the Adobe Camera Raw software development kit (SDK) to render sRGB images using different WB presets in the camera. Adobe Camera Raw accurately emulates the camera imaging pipeline and produces results virtually identical

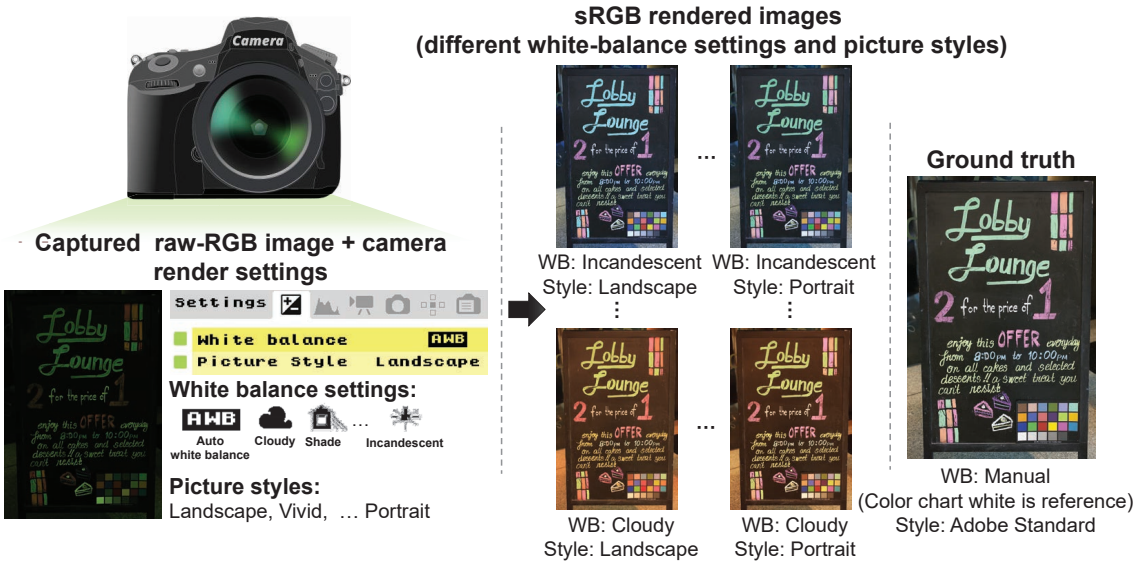


Figure 6.3: Example of rendering an sRGB training image. Working directly from the raw-RGB camera image, we render sRGB output images using the camera's pre-defined WB settings and different picture styles. A target WB sRGB image is also rendered using the color rendition chart in the scene to provide the ground truth.

to what the in-camera processing would produce.

In addition, each incorrect WB can be rendered with different camera picture styles (e.g., vivid, standard, neutral, landscape). Depending on the make and model of the camera, a single raw image can be rendered to more than 25 different camera-specific sRGB images. These images make up our training images $\{\mathbf{I}_{\text{t}}^{(1)}, ..., \mathbf{I}_{\text{t}}^{(n)}\}$.

To produce the correct target image, we manually select the "ground truth" white from the middle gray patches in the color rendition chart, followed by applying a camera-independent rendering style—namely, *Adobe Standard*. This provides the target ground truth sRGB image $\mathbf{I}_{\text{gt}}^{(i)}$. Figure 6.3 illustrates an example of a raw image from the NUS dataset and the corresponding sRGB images rendered with different WB settings and picture styles. In the end, we generated 62,535 images from these data sets.

6.2.3 Color Correction Transform

After generating our training images, we have n pairs of images representing an incorrectly white-balanced image $\mathbf{I}_{\text{t}}^{(i)}$ and its *correct* white-balanced image $\mathbf{I}_{\text{gt}}^{(i)}$. These are represented as $3\times N$ matrices, where N is the total number of pixels in the image and the three rows represent the red, green, and blue values in the camera's output sRGB color space.

We can compute a color correction matrix $\mathbf{M}^{(i)}$, which maps $\mathbf{I}_{\text{t}}^{(i)}$ to $\mathbf{I}_{\text{gt}}^{(i)}$, by minimizing the following equation:

$$\underset{\mathbf{M}^{(i)}}{\arg\min} \left\| \mathbf{M}^{(i)} \; \Phi\left(\mathbf{I}_{\text{t}}^{(i)}\right) - \mathbf{I}_{\text{gt}}^{(i)} \right\|_{\text{F}}, \tag{6.1}$$

where $\|.\|_{\text{F}}$ is the Frobenius norm and Φ is a kernel function that projects the sRGB triplet to a high-dimensional space.

We have examined several different color transformation mappings and found the polynomial kernel function proposed by Hong et al. [166] provided the best results for our task (more details are given in Sec. 6.3.2). Based on [166], Φ:[R, G, B]$^T \rightarrow$ [R, G, B, RG, RB, GB, R^2, G^2, B^2, RGB, 1]^T and $\mathbf{M}^{(i)}$ is represented as a 3×11 matrix.

The color chart in the images was masked out during this process to avoid any bias that may occur from having the same object with a wide range of colors present in the scene. Note that spatial information is not considered when estimating the $\mathbf{M}^{(i)}$.

6.2.4 Image Search

Since our color correction matrix is related to the image's color distribution, our criteria for finding similar images are based on the color distribution. We also seek compact representation as these features represent the bulk of information that will need to be stored in memory.

We rely on a non-learnable version of the RGB-uv histogram used in Chapter 4. Specifically, we construct a histogram feature from the log-chrominance space, which represents the color distribution of an image $\mathbf{I}$ as an $m \times m \times 3$ tensor that is parameterized by uv. This histogram is generated by the function $h(\mathbf{I})$ described by the following equations:

$$\begin{aligned}
\mathbf{I}_{y(i)} &= \sqrt{\mathbf{I}^2_{\mathrm{R}(i)} + \mathbf{I}^2_{\mathrm{G}(i)} + \mathbf{I}^2_{\mathrm{B}(i)}}, \\
\mathbf{I}_{u1(i)} &= \log\left(\mathbf{I}_{\mathrm{R}(i)}\right) - \log\left(\mathbf{I}_{\mathrm{G}(i)}\right), \\
\mathbf{I}_{v1(i)} &= \log\left(\mathbf{I}_{\mathrm{R}(i)}\right) - \log\left(\mathbf{I}_{\mathrm{B}(i)}\right), \\
\mathbf{I}_{u2} &= -\mathbf{I}_{u1} \ , \mathbf{I}_{v2} = -\mathbf{I}_{u1} + \mathbf{I}_{v1}, \\
\mathbf{I}_{u3} &= -\mathbf{I}_{v1} \ , \mathbf{I}_{v3} = -\mathbf{I}_{v1} + \mathbf{I}_{u1}.
\end{aligned} \tag{6.2}$$

$$\mathbf{H}\left(\mathbf{I}\right)_{(u,v,C)} = \sum_i \mathbf{I}_{y(i)} \left[\left|\mathbf{I}_{uC(i)} - u\right| \leqslant \frac{\varepsilon}{2} \wedge \left|\mathbf{I}_{vC(i)} - v\right| \leqslant \frac{\varepsilon}{2}\right], \tag{6.3}$$

$$h(\mathbf{I})_{(u,v,C)} = \sqrt{\frac{\mathbf{H}\left(\mathbf{I}\right)_{(u,v,C)}}{\sum_{u'} \sum_{v'} \mathbf{H}\left(\mathbf{I}\right)_{(u',v',C)}}}, \tag{6.4}$$

where $i = \{1, ..., N\}$, $\mathrm{R}, \mathrm{G}, \mathrm{B}$ represent the color channels in $\mathbf{I}$, $C \in \{1, 2, 3\}$ represents each color channel in the histogram, and ε is the histogram bin's width. Taking the square root after normalizing $\mathbf{H}$ increases the discriminatory ability of our projected histogram feature [34, 44].

Lastly, the histogram is normalized to represent the color distribution in the new chrominance coordinates (u_c, v_c).

For the sake of efficiency, we apply a dimensionality reduction step in order to extract a compact feature representing each RGB-uv histogram. We found that the PCA linear transformation is adequate for our task to map the vectorized histogram $\mathrm{vec}(h\left(\mathbf{I}\right)) \in \mathbb{R}^{m \times m \times 3}$ to a new lower-dimensional space. The PCA feature vector is computed as follows:

$$v(\mathbf{I}) = \left(\mathrm{vec}\left(h\left(\mathbf{I}\right)\right) - \mathbf{b}\right)^T \mathbf{W}, \tag{6.5}$$

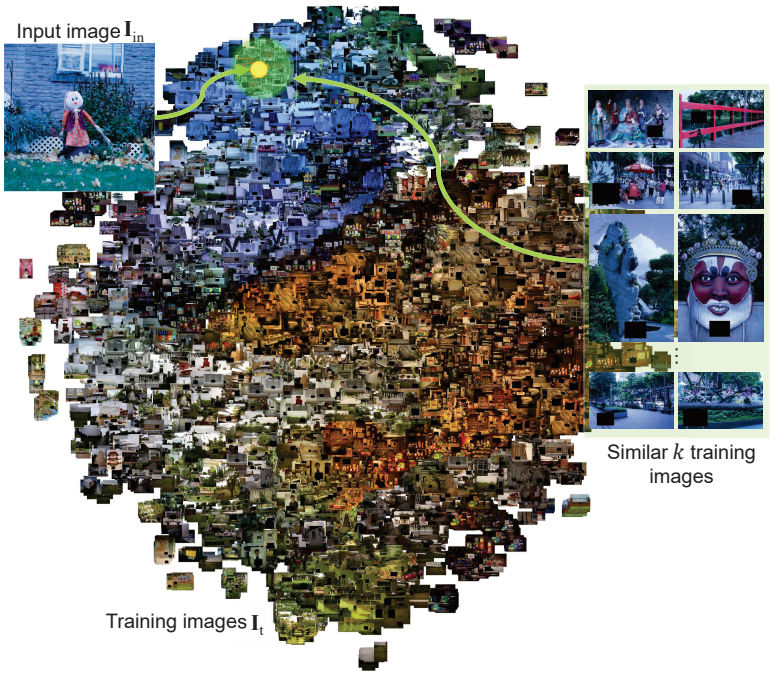


Figure 6.4: Visualization of the training images based on their corresponding PCA feature vectors. In this figure t-SNE [258] is used to aid visualization of the training space. Shown is an example input image and several of the nearest images retrieved using the PCA feature.

where $v(\mathbf{I}) \in \mathbb{R}^c$ is the PCA feature vector containing c principal component (PC) scores, $c \ll m \times m \times 3$, $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_c], \mathbf{w} \in \mathbb{R}^{m \times m \times 3}$ is the PC coefficient matrix computed by the singular value decomposition, and $\mathbf{b} \in \mathbb{R}^{m \times m \times 3}$ is the mean histogram vector. As a result, each training image $\mathbf{I}_t^{(i)}$ can be represented by a small number of PC scores $v\left(\mathbf{I}_t^{(i)}\right)$. The input image is finally represented by $v(\mathbf{I}_{in})$. The L_2 distance is adapted to measure the similarity between the PCA feature vectors. Figure 6.4 visualizes the training images based on their corresponding PCA features.

6.2.5 Final Color Correction

Given a new input image, we compute its PCA feature and search the training dataset for images with similar features. We extract the set of color correction matrices $\mathbf{M}_\mathrm{s}$ associated with the k similar images. The final correction matrix $\mathbf{M}$ is then computed as a weighted linear combination of the correction matrices $\mathbf{M}_\mathrm{s}$ as follows:

$$\mathbf{M} = \sum_{j=1}^{k} \boldsymbol{\alpha}_j \mathbf{M}_\mathrm{s}^{(j)}, \tag{6.6}$$

where $\boldsymbol{\alpha}$ is a weighting vector represented as a RBF:

$$\boldsymbol{\alpha}_j = \frac{\exp\left(-\mathbf{d}_j^2/2\sigma^2\right)}{\sum_{k'=1}^{k} \exp\left(-\mathbf{d}_{k'}^2/2\sigma^2\right)},\ j \in [1, ..., k], \tag{6.7}$$

where σ is the radial fall-off factor used in Eq. 6.7 and $\mathbf{d}$ represents a vector containing the L_2 distance between the given input feature and the similar k training features.

As shown in Fig. 6.2, the final color transformation is generated based on correction transformations associated with training images taken from different cameras and render styles. By blending the mapping functions from images produced by a wide range of different cameras and their different photo-finishing styles, we can interpret $\mathbf{M}$ correction as mapping the input image to a *meta-camera*'s output composed from the most similar images to the input.

Lastly, the corrected image $\mathbf{I}_\mathrm{corr}$ is produced by the following equation:

$$\mathbf{I}_\mathrm{corr} = \mathbf{M}\ \Phi\left(\mathbf{I}_\mathrm{in}\right). \tag{6.8}$$

6.2.6 Implementation Details

Our Matlab implementation requires approximately 0.54 seconds to compute the histogram feature. Once the PCA histogram feature is computed, the correction process takes an average of 0.73 seconds; this process includes the PCA feature extraction, the brute-force

search of the k nearest neighbors, blending the correction matrix, and the final image correction. All the reported runtimes were computed on an Intel® Xeon® E5-1607 @ 3.10 GHz machine and for a 12 mega-pixel image. The accelerated GPU implementation runs on average in 0.12 seconds using GTX 1080 GPU.

Our method requires 23.3 MB to store 62,535 feature vectors, mapping matrices, the PCA coefficient matrix, and the mean histogram vector using single-precision floating-point representation without affecting the accuracy.

In our implementation, each PCA feature vector was represented by 55 PC scores (i.e., $c = 55$), the PC coefficient matrix $\mathbf{W}$ was represented as a $(60 \times 60 \times 3) \times 55$ matrix (i.e., $m = 60$), and the mean vector $\mathbf{b} \in \mathbf{R}^{60 \times 60 \times 3}$. We used a fall-off factor $\sigma = 0.25$ and $k = 25$.

6.3 Experimental Results

6.3.1 Proposed Dataset

As described in Sec. 6.2.2, we have generated a dataset of 65,416 sRGB-rendered images that were divided into two sets: intrinsic set (Set 1) and extrinsic set (Set 2). Table 6.1 shows more details of the camera makes and models used for each set. The size of the original dataset is ∼1.14 TB, which was down-sampled by bicubic interpolation to 48.7 GB. For Set 1, the average image width and height are 890.1 and 687.2 pixels, respectively. For Set 2, the average width and height are 1,219.5 and 1,129.9 pixels, respectively.

For both sets, we have used the following WB presets: Incandescent, Fluorescent, Daylight, Cloudy, and Shade. The corresponding color temperatures of these presets are: 2850 Kelvin (K), 3800K, 5500K, 6500K, and 7500K. For Set 1, we have used the following camera picture styles: Adobe Standard, Faithful, Landscape, Neutral, Portrait, Standard, Vivid, Soft, D2X (mode 1, 2, and 3), and ACR (4.4 and 3.7). For Set 2, we have used the

Table 6.1: Camera models used in the proposed dataset. The intrinsic set (Set 1) comprises 62,535 sRGB images (48.7 GB) for seven different cameras. The extrinsic set (Set 2) comprises 2,881 sRGB images (5.43 GB) for one DSLR camera and four different mobile phone cameras. For each image in the dataset, there is a corresponding ground truth sRGB image rendered with a correct WB in Adobe Standard.

	Intrinsic set (Set 1)							
Camera	Canon EOS-1Ds Mark III	Canon EOS 600D	Fujifilm X-M1	Nikon D40	Nikon D5200	Canon 1D	Canon 5D	Total
# of images	10,721	9,040	5,884	10,826	8,953	2,284	14,827	62,535
Size	11.00 GB	8.27 GB	4.78 GB	3.4 GB	10.3 GB	1.27 GB	9.68 GB	48.7 GB
	Extrinsic set (Set 2)							
Camera	Olympus E-PL6		Mobile phone cameras: Galaxy S6 Edge, iPhone 7, LG G4, and Google Pixel					Total
# of images	1,874		1,007					2,881
Size	3.5 GB		1.93 GB					5.43 GB

following camera picture styles: (for the Olympus camera) Muted, Portrait, Vivid, Adobe Standard, and (for the mobile cameras) the camera's "embedded style".

6.3.2 Hyperparameters Selection

In order to select the number of nearest neighbors, k, and the most appropriate color correction transform, we have evaluated the accuracy of different color correction approaches

Table 6.2: Different kernel functions used to study the most suitable color correction matrix for our problem. The first column represents the dimensions of the output vector of the corresponding kernel function in the second column. The terms PCC and RPC stand for polynomial color correction and root polynomial color correction, respectively.

Dimensions	**Kernel function output**
3 (linear)	$[R, G, B]^T$ (identity)
9 (PCC) [118]	$[R, G, B, R^2, G^2, B^2, RG, RB, GB]^T$
11 (PCC) [166]	$[R, G, B, RG, RB, GB, R^2, G^2, B^2, RGB, 1]^T$
19 (PCC) [118]	$[R, G, B, RG, RB, GB, R^2, G^2, B^2, R^3, G^3, B^3, RG^2, RB^2, GB^2, GR^2, BG^2, BR^2, RGB]^T$
34 (PCC) [118]	$[R, G, B, RG, RB, GB, R^2, G^2, B^2, R^3, G^3, B^3, RG^2, RB^2, GB^2, GR^2, BG^2, BR^2, RGB, R^4, G^4, B^4, R^3G, R^3B, G^3R, G^3B, B^3R, B^3G, R^2G^2, G^2B^2, R^2B^2, R^2GB, G^2RB, B^2RB]^T$
6 (RPC) [118]	$[R, G, B, \sqrt{RG}, \sqrt{GB}, \sqrt{RB}]^T$
13 (RPC) [118]	$[R, G, B, \sqrt{RG}, \sqrt{GB}, \sqrt{RB}, \sqrt[3]{RG^2}, \sqrt[3]{RB^2}, \sqrt[3]{GB^2}, \sqrt[3]{GR^2}, \sqrt[3]{BG^2}, \sqrt{BR^2}, \sqrt[3]{RGB}]^T$
22 (RPC) [118]	$[R, G, B, \sqrt{RG}, \sqrt{GB}, \sqrt{RB}, \sqrt[3]{RG^2}, \sqrt[3]{RB^2}, \sqrt[3]{GB^2}, \sqrt[3]{GR^2}, \sqrt[3]{BG^2}, \sqrt{BR^2}, \sqrt[3]{RGB}, \sqrt[4]{R^3G}, \sqrt[4]{R^3B}, \sqrt[4]{G^3R}, \sqrt[4]{G^3B}, \sqrt[4]{B^3R}, \sqrt[4]{B^3G}, \sqrt[4]{R^2GB}, \sqrt[4]{G^2RB}, \sqrt[4]{B^2RG}]^T$

between an incorrectly white-balanced camera-rendered image and its correctly white-balanced camera-rendered target image.

This study was conducted for quality assessment rather than performance. Accordingly, we have used the RGB-uv histogram features with bandwidth $m = 180$ without dimensionality reduction applied.

By taking the square root after normalizing $\mathbf{H}$, it makes the classic Euclidean L_2 distance applicable as a symmetric similarity metric to measure the similarity between two distributions [34]. Consistently with the PCA feature similarity measurement, the Hellinger

distance [308] was used as a similarity metric in this study. The Hellinger distance [308] between two histograms $h(\mathbf{I}_1)$ and $h(\mathbf{I}_2)$ can be represented as $\left(1/\sqrt{2}\right)\ \mathrm{L}_2\left(h(\mathbf{I}_1), h(\mathbf{I}_2)\right)$.

We tested 8 different color correction matrices on Set 1 using three-fold validation. The matrices are: 3×3 full color correction matrices, 3×9, 3×11, 3×19, 3×34 polynomial color correction (PCC) matrices, and 3×6, 3×13, 3×22 root polynomial color correction (RPC) matrices [118, 166]. For each color correction matrix, we have tested different values of k. Note that the 3×3 color correction matrix is computed using Eq. 6.1 with $\Phi\left(\mathbf{I}_{\mathrm{t}}^{(i)}\right) = \mathbf{I}_{\mathrm{t}}^{(i)}$— that is, the kernel function is an identity function.

Table 6.2 shows the kernel functions used to generate each color correction matrix. Figure 6.5 shows the results obtained using four different error metrics. As shown, the 3×11 color correction matrix, described by Hong et al. [166], provided the best results for our task. Also, it is shown that the accuracy increases as the value of k increases. At a certain point, however, increasing k negatively affects the accuracy.

In this set of experiments, adopting the RGB-*uv* histogram features requires approximately 14.9 GB of memory having ∼41K training samples represented as single-precision floating-point values, and runs in 44.6 seconds to correct a 12 mega-pixel image on average. This process includes the RGB-*uv* histogram feature extraction, the brute-force search of the k nearest neighbors, blending the correction matrix, and the final image correction. In Sec. 6.2.4, we extract a compact feature representing each RGB-*uv* histogram. This compact representation improves the performance (requiring less than 1.5 seconds on CPU to correct a 12 mega-pixel image) and achieves on-par accuracy compared to employing the original RGB-*uv* histogram features.

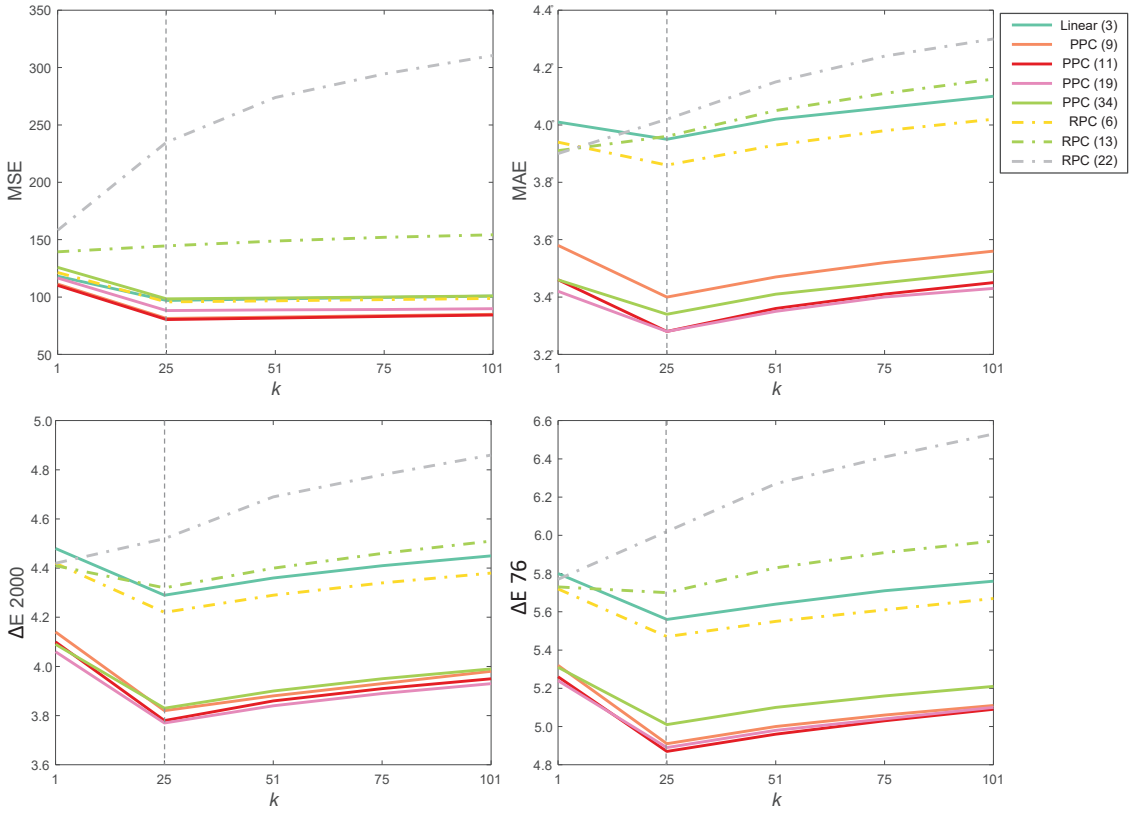


Figure 6.5: A study of the accuracy obtained using different color correction matrices, which are: (i) linear 3×3 full matrix, (ii) 3×9, (iii) 3×11, (iv) 3×19, (v) 3×34 polynomial color correction (PCC) matrices [118,166], (vi) 3×6, (vii) 3×13, and (viii) 3×22 root-polynomial color correction (RPC) matrices [118]. The horizontal axis represents the number of nearest neighbors, k, and the vertical axis represents the error between the corrected images and the ground truth images using different error metrics.

6.3.3 Results and Comparisons

In this section, our proposed approach is compared with common approaches that are currently used to correct improperly white-balanced rendered sRGB images in the proposed dataset. We report and discuss both quantitative and qualitative results. Failure cases of

our algorithm are also shown. Finally, we provide a time analysis of the proposed approach.

We have used Set 1 of our dataset for training and evaluation using three-fold validation, such that the three folds are disjointed in regards to the imaged scenes, meaning if the scene (i.e., original raw-RGB image) appears in a fold, it is excluded from the other folds. The color rendition chart is masked out in the image and ignored during training and testing.

Quantitative results

We compared our results against a diagonal WB correction that is computed using the center gray patch in the color checker chart placed in the scene. We refer to this as the *exact achromatic* reference point, as it represents a true neutral point found in the scene. This exact white point represents the best results that an illumination estimation algorithm could achieve when applied to our input in order to determine the diagonal WB matrix.

For the sake of completeness, we also compared our results against a "linearized" diagonal correction that applies an inverse gamma operation [31,101], then performed WB using the exact reference point, and then reapplied the gamma to produce the result in the sRGB color space. We also include results using Adobe Photoshop corrections—specifically, the auto-color function (AC) and auto-tone function (AT).

We also perform comparisons against the diagonal correction using representative examples of illuminant estimation methods—this means the diagonal matrix is automatically computed and not derived from a selected achromatic patch in the scene. As mentioned in Chapter 2, illumination estimation methods are intended to be applied on raw-RGB images; however, in this case we apply it on the sRGB-rendered image. We use five well-known statistical methods for illumination estimation—namely, the GW [60], GE [362], wGE [139], max-RGB [58], and SoG [120]. The Minkowski norm (p) was set to 5 for

GE, wGE, and SoG. For GE, we computed the results using the first and second differentiations. For each method, we calculated the diagonal correction with and without the pre-linearization process. We could not find learning-based models trained on sRGB images for illuminant estimation except for the fully convolutional color constancy with confidence-weighted pooling (FC4) model [171]. Specifically, we use the FC4 trained model on sRGB-rendered images of Gehler dataset [132] provided by the authors. Other learning-based illuminant estimation methods were excluded, since they were trained on the linear RGB space and re-training them in the sRGB space would use the ERs as ground truth illuminants that were already included in our comparisons. Tables 6.3 and 6.4 shows the obtained results of the exact achromatic reference point correction and other illuminant estimation algorithms. Table 6.5 shows our results against the Adobe Photoshop functions for color corrections—namely, auto color and auto tone.

Tables 6.3–6.5 show that our proposed method consistently outperforms the other approaches in all metrics.

Qualitative results and User Study

Qualitative visual results for Set 1 and Set 2 are shown in Fig. 6.7 and Fig. 6.8. It is arguable that our results are most visually similar to the ground truth images. To confirm this independently, we have conducted a user study of 35 participants (18 males and 17 females), ranging in age from 21 to 46. Each one was asked to choose the most visually similar image to the ground truth image between the results of our method and the diagonal correction with the exact reference point. Experiments were carried out in a controlled environment. The monitor was calibrated using a Spyder5 colorimeter.

Participants were asked to compare 24 pairs of images, such that for each quartile, based on the MSE of each method, 4 images were randomly picked from Set 1 and Set

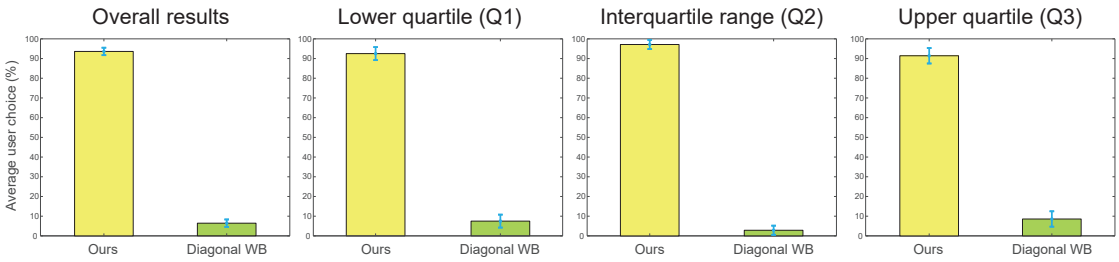


Figure 6.6: The results of a user study with 35 people in which users are asked which output is most visually similar to the ground truth image. An equal number of images are selected randomly from the different quartiles. The outcome of the user study is shown via interval plots, with error bars shown at a 99% confidence interval.

2. That means the selected images represent the best, median, and worst results of each method and for each set. On average, 93.69% of our results were chosen as the most similar to the ground truth images. Figure 6.6 illustrates that the results of this study are statistically significant with p-value < 0.01.

6.4 Summary

This chapter has proposed the first method to explicitly address the problem of correcting a camera image that has been improperly white-balanced. This situation occurs when a camera's AWB fails or when the wrong manual WB setting is used. The proposed method is enabled by a dataset we generated with over 65,000 pairs of incorrectly white-balanced images and their corresponding correctly white-balanced image. Given an improperly white-balanced camera image, we outlined a simple KNN strategy that is able to find similar incorrectly white-balanced images in the dataset. Based on these similar examples images, we described how to construct a nonlinear color correction transform that is used to remove the color cast. The proposed approach requires a small memory overhead (less

Figure 6.7: Comparisons between the proposed approach and other techniques on **Set 1** (first four rows) and **Set 2** (last two rows). (A) Input image in sRGB. (B) Results of Adobe Photoshop (Ps) color correction functions. (C) Results of diagonal correction using the *exact* reference point obtained directly from the color chart. (D) Our results. (E) Ground truth images. In (B) and (C) we pick the best result between the auto-color (AC) and auto-tone (AT) functions and between the sRGB (sRGB) and "linearized" sRGB (LRGB) [31] based on $\triangle$E values, respectively.

than 25MB) and is computationally fast ($\sim$1 second for a 12 mega-pixel image).

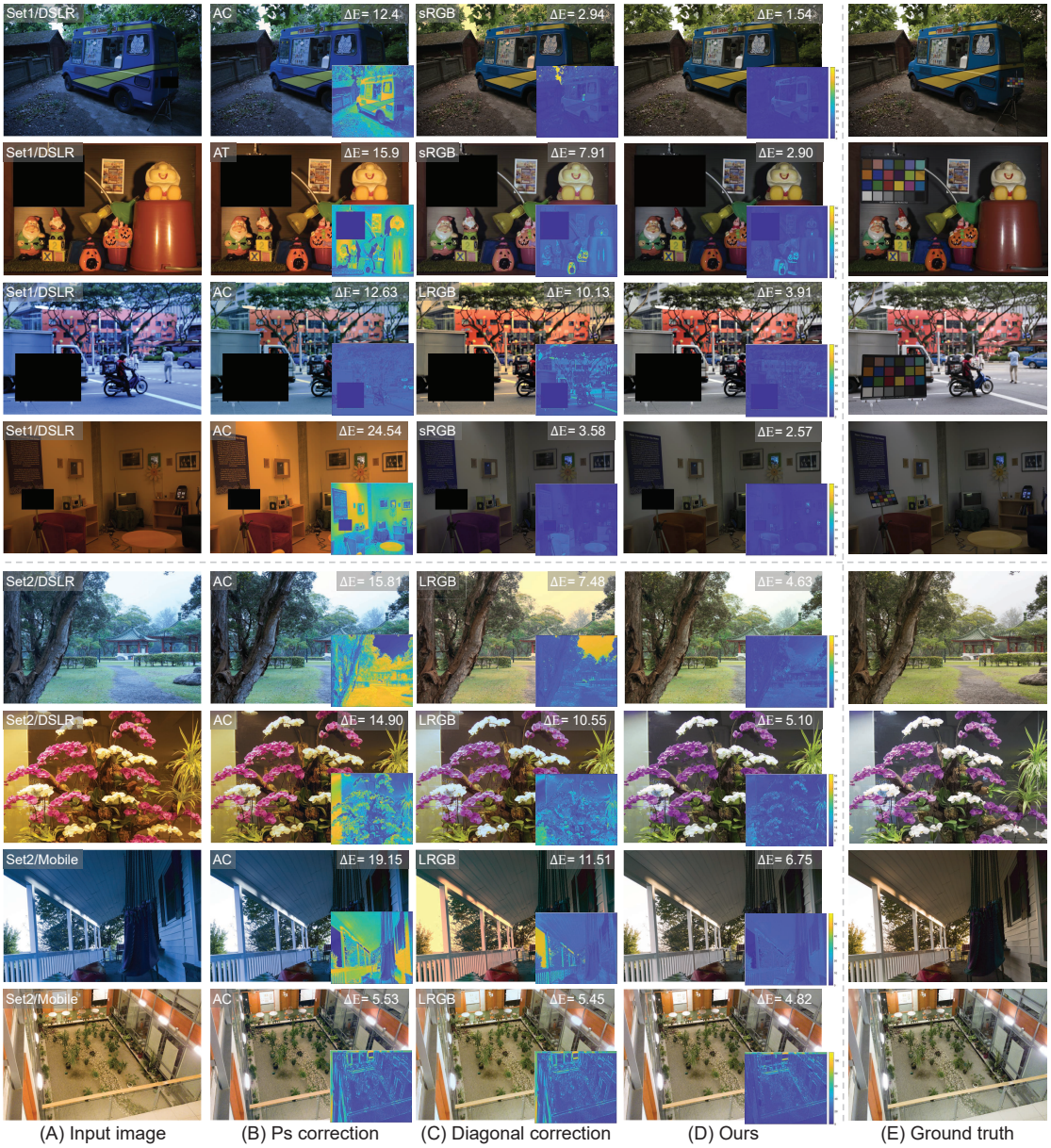


Figure 6.8: Additional qualitative comparisons between our method and other techniques on **Set 1** (first four rows) and **Set 2** (last four rows).

Table 6.3: Comparisons between our method with diagonal WB correction using an *exact achromatic* reference point (ER). We also show results obtained by different illuminant estimation methods. The diagonal correction is applied directly to the sRGB images, denoted as (sRGB) and "linearized" RGB [31, 101], denoted as (LRGB). The terms Q1, Q2, and Q3 denote the first, second (median), and third quartile, respectively. The terms MSE and MAE stand for mean square error and mean angular error, respectively. The top results are indicated with yellow and boldface.

Method	**MSE**				**MAE**				**△E 2000**			
	Mean	**Q1**	**Q2**	**Q3**	**Mean**	**Q1**	**Q2**	**Q3**	**Mean**	**Q1**	**Q2**	**Q3**
Intrinsic set (Set 1): DSLR multiple cameras (62,535 images)												
GW (sRGB) [60]	282.76	70.50	180.81	380.89	7.23°	4.14°	6.40°	9.51°	7.99	5.08	7.47	10.40
GW (LRGB) [60]	285.51	73.37	184.72	384.67	7.97°	4.36°	6.91°	10.53°	8.48	5.38	7.91	11.03
GE-1 (sRGB) [362]	193.99	43.38	119.93	267.84	6.81°	3.22°	5.49°	9.31°	6.86	3.92	6.30	9.17
GE-1 (LRGB) [362]	190.59	42.51	116.75	261.6	6.54°	3.22°	5.37°	8.78°	6.82	3.91	6.20	9.06
GE-2 (sRGB) [362]	208.20	44.82	126.83	285.31	7.03°	3.30°	5.67°	9.62°	7.06	4.01	6.44	9.44
GE-2 (LRGB) [362]	204.93	43.86	123.19	279.15	6.76°	3.31°	5.54°	9.12°	7.02	4.01	6.35	9.32
wGE (sRGB) [139]	225.87	42.76	122.39	300.48	7.15°	3.20°	5.60°	9.69°	7.07	3.78	6.29	9.50
wGE (LRGB) [139]	223.30	42.40	119.47	294.22	6.90°	3.17°	5.42°	9.26°	7.05	3.76	6.21	9.43
max-RGB (sRGB) [58]	285.07	47.20	160.6	397.79	8.48°	3.49°	7.03°	12.35°	8.05	3.97	7.25	11.34
max-RGB (LRGB) [58]	280.86	46.21	156.09	390.99	8.17°	3.41°	6.76°	11.83°	7.96	3.92	7.12	11.21
SoG (sRGB) [120]	171.30	38.31	104.85	235.57	6.06°	2.99°	5.08°	8.21°	6.19	3.52	5.72	8.25
SoG (LRGB) [120]	169.33	38.10	102.21	231.50	5.96°	3.03°	5.03°	7.91°	6.24	3.56	5.71	8.27
FC4 (sRGB) [171]	426.31	118.30	282.05	561.92	7.91°	4.57°	7.33°	10.38°	9.78	6.12	9.14	12.65
FC4 (LRGB) [171]	179.55	33.89	100.09	246.50	6.14°	2.62°	4.73°	8.40°	6.55	3.54	5.90	8.94
ER (sRGB)	135.77	20.20	71.74	196.15	4.63°	1.99°	3.56°	6.14°	4.69	2.25	4.00	6.68
ER (LRGB)	130.01	19.73	68.54	183.65	4.29°	1.85°	3.35°	5.70°	4.59	2.24	3.89	6.51
Ours	**77.79**	**13.74**	**39.62**	**94.01**	**3.06°**	**1.74°**	**2.54°**	**3.76°**	**3.58**	**2.07**	**3.09**	**4.55**

Table 6.4: Comparisons between our method with diagonal WB correction using an *exact achromatic* reference point (ER). We also show results obtained by different illuminant estimation methods. The diagonal correction is applied directly to the sRGB images, denoted as (sRGB) and "linearized" RGB [31, 101], denoted as (LRGB). The terms Q1, Q2, and Q3 denote the first, second (median), and third quartile, respectively. The terms MSE and MAE stand for mean square error and mean angular error, respectively. The top results are indicated with yellow and boldface.

Method	**MSE**				**MAE**				**△E 2000**			
	Mean	**Q1**	**Q2**	**Q3**	**Mean**	**Q1**	**Q2**	**Q3**	**Mean**	**Q1**	**Q2**	**Q3**
Extrinsic set (Set 2): DSLR and mobile phone cameras (2,881 images)												
GW (sRGB) [60]	500.18	173.69	332.75	615.40	8.89°	5.82°	8.32°	11.33°	10.74	7.92	10.29	13.12
GW (LRGB) [60]	469.86	163.07	312.28	574.85	8.61°	5.44°	7.94°	10.93°	10.68	7.70	10.13	13.15
GE-1 (sRGB) [362]	791.10	235.37	524.94	1052.38	12.90°	7.98°	12.41°	17.50°	13.09	9.17	12.98	16.68
GE-1 (LRGB) [362]	779.27	225.36	510.03	1038.71	12.55°	7.55°	11.87°	17.00°	12.98	9.06	12.86	16.57
GE-2 (sRGB) [362]	841.83	239.58	542.07	1114.86	13.16°	7.94°	12.55°	17.76°	13.31	9.20	13.20	17.09
GE-2 (LRGB) [362]	831.01	231.42	530.52	1099.75	12.84°	7.64°	12.13°	17.45°	13.22	9.00	13.13	17.01
wGE (sRGB) [139]	999.95	236.46	587.55	1350.41	13.80°	8.08°	12.99°	18.80°	14.05	9.13	13.80	18.56
wGE (LRGB) [139]	990.20	230.38	577.62	1345.52	13.52°	7.76°	12.62°	18.56°	14.00	9.00	13.70	18.56
max-RGB (sRGB) [58]	791.99	263.00	572.23	1087.14	13.47°	8.44°	12.93°	18.50°	13.01	9.12	13.44	17.15
max-RGB (LRGB) [58]	780.63	256.40	560.58	1073.22	13.18°	8.15°	12.57°	18.12°	12.93	9.02	13.36	17.08
SoG (sRGB) [120]	429.35	147.05	286.84	535.72	9.54°	5.72°	8.85°	12.65°	10.01	7.09	9.85	12.69
SoG (LRGB) [120]	393.85	137.21	267.37	497.40	8.96°	5.31°	8.26°	11.97°	9.81	6.87	9.67	12.46
FC4 (sRGB) [171]	662.53	304.88	524.42	817.57	8.92°	5.94°	8.03°	10.84°	12.12	8.94	11.79	14.76
FC4 (LRGB) [171]	505.30	142.46	307.77	635.35	10.37°	5.31°	9.26°	14.15°	10.82	7.39	, 10.64	13.77
ER (sRGB)	422.31	110.70	257.76	526.16	7.99°	4.36°	7.11°	10.57°	8.53	5.52	8.38	11.11
ER (LRGB)	385.23	99.05	230.86	475.72	7.22°	3.80°	6.34°	9.54°	8.15	5.07	7.88	10.68
Ours	**171.09**	**37.04**	**87.04**	**190.88**	**4.48°**	**2.26°**	**3.64°**	**5.95°**	**5.60**	**3.43**	**4.90**	**7.06**

Table 6.5: Comparisons between our method with the Adobe Photoshop functions: auto-color (AC) and auto-tone (AT). The terms Q1, Q2, and Q3 denote the first, second (median), and third quartile, respectively. The terms MSE and MAE stand for mean square error and mean angular error, respectively. The top results are indicated with yellow and boldface.

Method	**MSE**				**MAE**				**△E**			
	Mean	**Q1**	**Q2**	**Q3**	**Mean**	**Q1**	**Q2**	**Q3**	**Mean**	**Q1**	**Q2**	**Q3**
Intrinsic set (Set 1): DSLR multiple cameras (62,535 images)												
Photoshop-AC	780.52	157.39	430.96	991.28	7.96°	3.43°	5.59°	10.58°	10.06	5.75	8.92	13.30
Photoshop-AT	1002.93	238.33	606.74	1245.51	7.56°	3.08°	5.75°	10.83°	11.12	6.55	10.54	14.68
Ours	**77.79**	**13.74**	**39.62**	**94.01**	**3.06°**	**1.74°**	**2.54°**	**3.76°**	**3.58**	**2.07**	**3.09**	**4.55**
Extrinsic set (Set 2): DSLR and mobile phone cameras (2,881 images)												
Photoshop-AC	745.49	240.58	514.33	968.27	10.19°	5.25°	8.60°	14.13°	11.71	7.56	11.41	15.00
Photoshop-AT	953.85	386.7	743.84	1256.94	11.91°	7.01°	10.70°	15.92°	13.12	9.63	13.18	16.5
Ours	**171.09**	**37.04**	**87.04**	**190.88**	**4.48°**	**2.26°**	**3.64°**	**5.95°**	**5.60**	**3.43**	**4.90**	**7.06**

7 Deep Neural Networks Performance With White-Balance Errors

Color correction has a crucial importance not only in photography aesthetics but also for computer vision tasks. In this chapter, we explore how strong color casts caused by incorrectly applied WB negatively impact the performance of DNNs targeting image segmentation and classification[1]. In addition, we discuss how existing image augmentation methods used to improve the robustness of DNNs are not well suited for modeling WB errors. To address this problem, a novel augmentation method is proposed that can emulate accurate color constancy degradation. We also explore pre-processing training and testing images with a recent WB correction algorithm to reduce the effects of incorrectly white-balanced images. We examine both augmentation and pre-processing strategies on different datasets and demonstrate notable improvements on the CIFAR-10, CIFAR-100, and ADE20K datasets. The test set and source code of this work are available on GitHub: https://github.com/mahmoudnafifi/WB_color_augmenter.

[1]This work was published in [14]: Mahmoud Afifi and Michael S. Brown. What Else Can Fool Deep Learning? Addressing Color Constancy Errors on Deep Neural Network Performance. In IEEE International Conference on Computer Vision (ICCV), 2019.

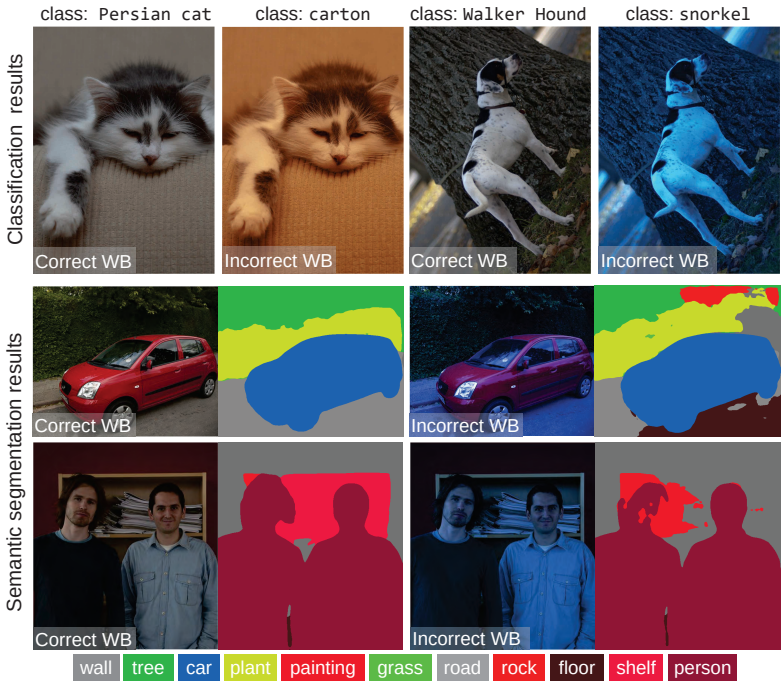


Figure 7.1: The effect of correct/incorrect computational color constancy (i.e., white balance) on (top) classification results by ResNet [159]; and (bottom) semantic segmentation by RefineNet [237].

7.1 Introduction

There is active interest in local image manipulations that can be used to fool DNNs into producing erroneous results. Such "adversarial attacks" often result in drastic misclassifications. We examine a less explored problem of *global* image manipulations that can result in similar adverse effects on DNNs' performance. In particular, we are interested in the role of computational color constancy, which makes up the WB routine on digital cameras.

We focus on computational color constancy because it represents a common source of global image errors found in real images. As discussed in Chapter 6, when WB is applied incorrectly on a camera, it results in an undesirable color cast in the captured image.

Images with such strong color casts are often discarded by users. As a result, online image databases and repositories are biased to contain mostly correctly white-balanced images. This is an implicit assumption that is not acknowledged for datasets composed of images crawled from the web and online. However, in real-world applications, it is unavoidable that images will, at some point, be captured with the incorrect WB applied. Images with incorrect WB can have unpredictable results on DNNs trained on white-balanced biased training images, as demonstrated in Fig. 7.1.

Contribution We examine how errors related to computational color constancy can adversely affect DNNs focused on image classification and semantic segmentation. In addition, we show that image augmentation strategies used to expand the variation of training images are not well suited to mimic the type of image degradation caused by color constancy errors. To address these problems, we introduce a novel augmentation method that can accurately emulate realistic color constancy degradation. We also examine our WB correction method (discussed in Chapter 6) to pre-process testing and training images. Experiments on CIFAR-10, CIFAR-100, and the ADE20K datasets using the proposed augmentation and pre-processing correction demonstrate notable improvements to test image inputs with color constancy errors.

7.2 Related Work

Adversarial Attacks DNN models are susceptible to adversarial attacks in the form of *local* image manipulation (e.g., see [83, 144, 211, 350]). These images are created by adding a carefully crafted imperceptible perturbation layer to the original image [144, 350]. Such perturbation layers are usually represented by *local* non-random adversarial noise [28, 144, 271, 350, 382] or *local* spatial transformations [379]. Adversarial examples are able

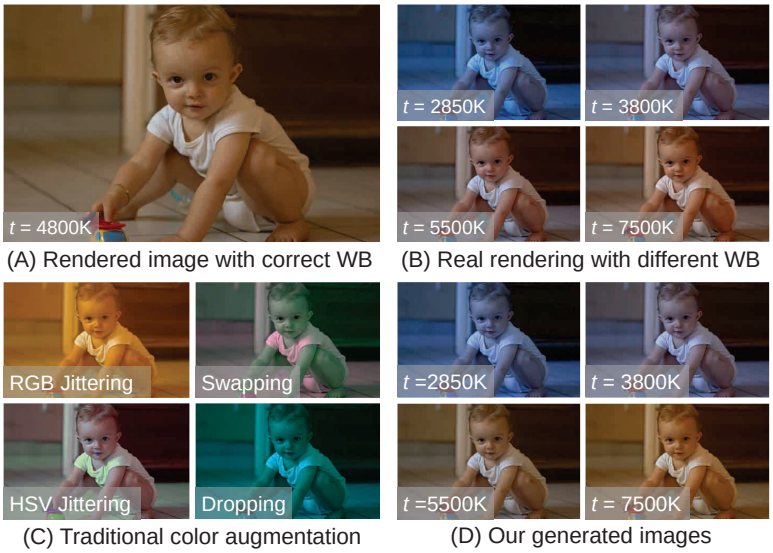


Figure 7.2: (A) An sRGB image from a camera with the correct WB applied. (B) Images from the same camera with the incorrect WB color temperatures (t) applied. (C) Images generated by processing image (A) using existing augmentation methods—the images clearly do not represent those in (B). (D) Images generated from (A) using our proposed method detailed in Sec. 7.4.

to misguide pre-trained models to predict either a certain wrong response (i.e., targeted attack) or any wrong response (i.e., untargeted attack) [28, 66, 245]. While incorrect color constancy is not an explicit attempt at an adversarial attack, the types of failures produced by this *global* modification act much like an untargeted attack and can adversely affect DNNs' performance.

Data Augmentation To overcome limited training data and to increase the visual variation, image augmentation techniques are applied to training images. Existing image augmentation techniques include: geometric transformations (e.g., rotation, translation, shearing) [85,156,156,299], synthetic occlusions [408], pixel intensity processing (e.g., equal-

ization, contrast adjustment, brightness, noise) [85, 364], and color processing (e.g., RGB color jittering and PCA-based shifting, HSV jittering, color channel dropping, color channel swapping) [71,85,97,190,209,225,277,317,319]. Traditional color augmentation techniques randomly change the original colors of training images aiming for better generalization and robustness of the trained model in the inference phase. However, existing color augmentation methods often generate unrealistic colors which rarely happen in reality (e.g., green skin or purple grass). More importantly, the visual appearance of existing color augmentation techniques does not well represent the color casts produced by incorrect WB applied onboard cameras, as shown in Fig. 7.2. As demonstrated in [32, 67, 95], image formation has an important effect on the accuracy of different computer vision tasks. Recently, a simplified version of the camera imaging pipeline was used for data augmentation [67]. This augmentation method in [67], however, explicitly did not consider the effects of incorrect WB due to the subsequent nonlinear operations applied after WB. To address this issue, we propose a camera-based augmentation technique that can synthetically generates images with realistic WB settings.

DNN Normalization Layers Normalization layers are commonly used to improve the efficiency of the training process. Such layers apply simple statistics-based shifting and scaling operations to the activations of network layers. The shift and scale factors can be computed either from the entire mini-batch (i.e., batch normalization [178]) or from each training instance (i.e., instance normalization [360]). Recently, batch-instance normalization (BIN) [279] was introduced to ameliorate problems related to styles/textures in training images by balancing between batch and instance normalizations based on the current task. Though the BIN is designed to learn the trade-off between keeping or reducing original training style variations using simple statistics-based operations, the work in [279]

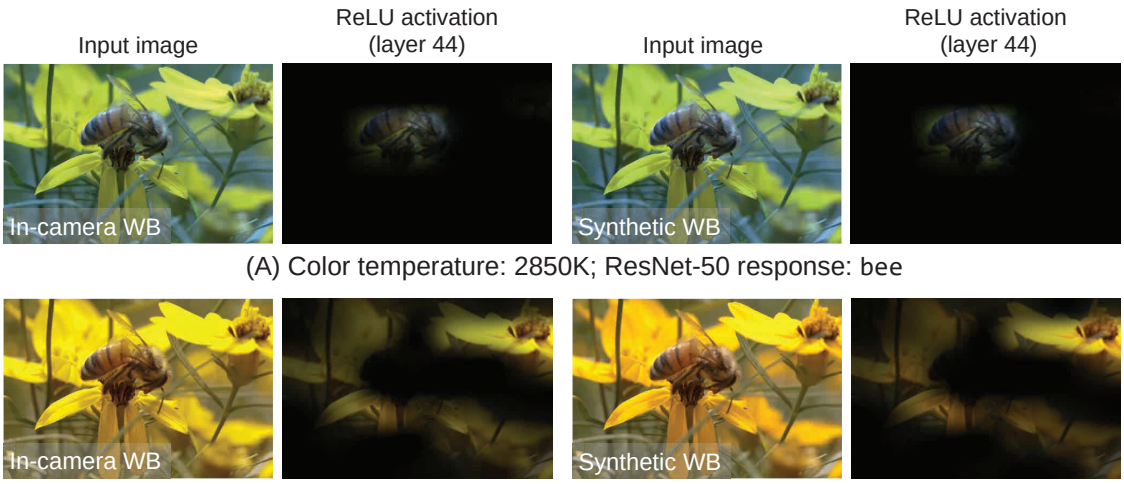


Figure 7.3: Image rendered with two different color temperatures (denoted by t) using in-camera rendering and our method. (A) Image class is `bee`. (B) Image class is `rapeseed`. Classification results were obtained by ResNet-50 [159].

does not provide any study regarding incorrect WB settings. The augmentation and pre-processing methods proposed in our work directly target training and testing images and do not require any change to a DNNs architecture or training regime.

7.3 Effects of WB Errors on DNNs

We begin by studying the effect of incorrectly white-balanced images on pre-trained DNN models for image classification and semantic segmentation. As a motivation, Fig. 7.3 shows two different WB settings applied to the same image. Figure 7.3 shows that the DNN's attention for the same scene is considerably altered by changing the WB setting.

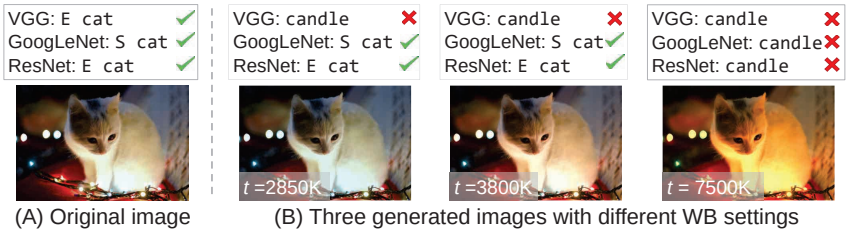
For quantitative evaluations, we adopted several DNN models trained for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 [91] and the ADE20K Scene Parsing Challenge 2016 [409]. Generating an entirely new labeled testing set composed

(A) Original images (B) Generated images with different WB settings

Figure 7.4: Examples from ImageNet validation set [91] (first row) and ADE20K validation set [409] (second row). (A) Original images. (B) Images with different WB settings produced by our method.

of images with incorrect WB is an enormous task—ImageNet classification includes 1,000 classes and pixel-accurate semantic annotation requires ∼60 minutes per image [320]. In lieu of a new testing set, we applied our method which emulates WB errors to the validation images of each dataset. Our method will be detailed shortly in Sec. 7.4. Figure 7.4 shows examples of the generated images with different WB settings used in our study.

Classification We apply our method to ImageNet's validation set to generate images with five different color temperatures and two different photo-finishing styles for a total of ten WB variations for each validation image; 899 grayscale images were excluded from this process. In total, we generated 491,010 images. We examined the following six well-known DNN models, trained on the original ImageNet training images: AlexNet [209], VGG-16 & VGG-19 [342], GoogLeNet [349], and ResNet-50 & ResNet-101 [159]. Table 7.1 shows the accuracy drop for each model when tested on our generated validation set (i.e., with different WB and photo-finishing settings) compared to the original validation set. In most cases, there is a drop of ∼10% in accuracy. Figure 7.5 shows an example of the impact of incorrect WB.



(A) Original image (B) Three generated images with different WB settings

Figure 7.5: Pre-trained models are negatively impacted by incorrect WB settings. (A) Original image. (B) Generated images with different WB color temperatures (denoted by t). Classification results of: VGG-16 [342], GoogLeNet [349], and ResNet-50 [159] are written on top of each image. The terms `E` and `S` stand for `Egyptian` and `Siamese`, respectively.

Semantic Segmentation We used the ADE20K validation set for 2,000 images, and generated ten images with different WB/photo-finishing settings for each image. At the end, we generated a total of 20,000 new images. We tested the following two DNN models trained on the original ADE20K training set: DilatedNet [74, 392] and RefineNet [237]. Table 7.2 shows the effect of improperly white-balanced images on the intersection-over-union (IoU) and pixel-wise accuracy (pxl-acc) obtained by the same models on the original validation set. While DNNs for segmentation fare better than the results for classification, we still incur a drop of over 2% in performance.

7.4 Emulating WB Errors

In this section, we outline our method for emulating WB errors. Our WB emulator heavily relies on our framework presented in Chapter 6. Given an sRGB image, denoted as $\mathbf{I}_{t_{\text{corr}}}$, that is assumed to be white-balanced with the correct color temperature, our goal is to modify $\mathbf{I}_{t_{\text{corr}}}$'s colors to mimic its appearance as if it were rendered by a camera with

Table 7.1: Adverse classification performance on ImageNet [91] due to the inclusion of incorrect WB versions of its validation images. The models were trained on the original ImageNet training set. The reported numbers denote the changes in the top-1 accuracy achieved by each model.

Model	**Effect on top-1 accuracy**
AlexNet [209]	-0.112
VGG-16 [342]	-0.104
VGG-19 [342]	-0.102
GoogLeNet [349]	-0.107
ResNet-50 [159]	-0.111
ResNet-101 [159]	-0.109

Table 7.2: Adverse semantic segmentation performance on ADE20K [409] due to the inclusion of incorrect WB versions of its validation images. The models were trained on ADE20K's original training set. The reported numbers denote the changes in intersection-over-union (IoU) and pixel-wise accuracy (pxl-acc) achieved by each model on the original validation.

Model	**Effect on IoU**	**Effect on pxl-acc**
DilatedNet [74, 392]	-0.023	-0.024
RefineNet [237]	-0.031	-0.026

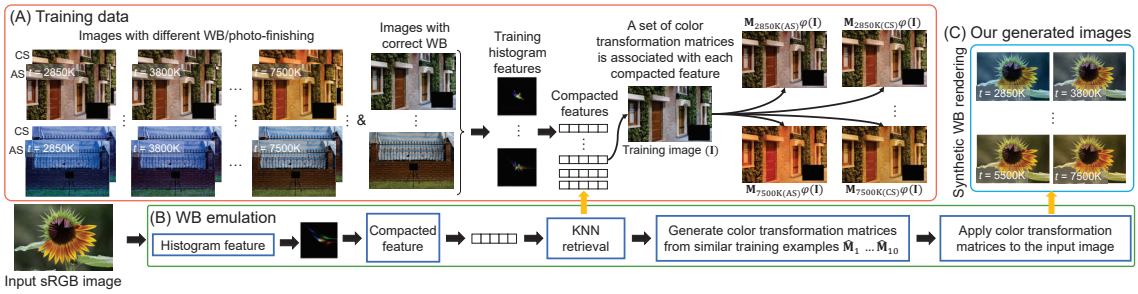


Figure 7.6: Our WB emulation framework. (A) A dataset of 1,797 correctly white-balanced sRGB images (proposed in Chapter 6); each image has ten corresponding sRGB images rendered with five different color temperatures and two photo-finishing styles, Camera Standard (CS) and Adobe Standard (AS). For each white-balanced image, we generate its compact histogram feature and ten color transformation matrices to the corresponding ten images. (B) Our WB emulation pipeline (detailed in Sec. 7.4). (C) The augmented images for the input image that represent different color temperatures (denoted by t) and photo-finishing styles.

different (incorrect) color temperatures, t, under different photo-finishing styles. Since we do not have access to $\mathbf{I}_{t_{\text{corr}}}$'s original raw-RGB image, we cannot re-render the image from raw-RGB to sRGB using a standard camera pipeline. Instead, we have modified our data-driven method discussed in Chapter 6 to mimic this manipulation directly in the sRGB color space. Figure 7.6 provides an overview of our modified framework.

7.4.1 Dataset

Our method relies on our dataset of sRGB images generated in Chapter 6. Recall that this dataset contains images rendered with different WB settings and photo-finishing styles. We have also a ground truth sRGB image (i.e., rendered with the "correct" color temperature) associated with each training image. In our WB emulation framework, we used 17,970

images from this dataset (1,797 correct sRGB images each with ten corresponding images rendered with five different color temperatures and two different photo-finishing styles, Camera Standard and Adobe Standard). The five color temperatures are: 2850 Kelvin (K), 3800K, 5500K, 6500K, and 7500K. In addition, each image was rendered using different camera photo-finishing styles.

7.4.2 Color Mapping

Next, we compute a mapping between the correct white-balanced sRGB image to each of its ten corresponding images. We follow the same procedure of the KNN WB method (Chapter 6) and use the kernel function, φ, to project RGB colors into a high-dimensional space. Then, we perform polynomial data fitting on these projected values as described in Chapter 6. Afterwards, we compute a color transformation matrix between each pair of correctly white-balanced image and its corresponding target image rendered with a specific color temperature and photo-finishing. In the end, we have *ten* matrices associated with each image in our training data.

7.4.3 Color Feature

As shown in Fig. 7.6, when augmenting an input sRGB image to have different WB settings, we search our dataset for similar sRGB images to the input image. This search is not based on scene content, but on the color distribution of the image (i.e., the RGB-uv projected color histogram feature used in Chapter 6).

7.4.4 KNN Retrieval

Given a new input image, we extract its compacted color feature $\mathbf{v}$ (Eq. 6.5), and then search for training examples with color distributions similar to the input image's color

distribution. Similarly to our framework in Chapter 6, the L_2 distance is adopted as a similarity metric between $\mathbf{v}$ and the training compacted color features. Afterwards, we retrieve the color transformation matrices associated with the nearest k training images. The retrieved set of matrices is represented by $\mathbf{M}_s = \{\mathbf{M}_s^{(j)}\}_{j=1}^{j=k}$, where $\mathbf{M}_s^{(j)}$ represents the color transformation matrix that maps the j^{th} white-balanced training image colors to their corresponding image colors rendered with color temperature t.

7.4.5 Transformation Matrix

After computing the distance vector between $\mathbf{v}$ and the nearest training features, we compute a weighting vector $\boldsymbol{\alpha}$ to blend between the associated transformation matrices of the nearest neighbor training examples (as described in Eqs. 6.7 and 6.6). Lastly, the "re-rendered" image $\hat{\mathbf{I}}_t$ with color temperature t is computed as in Eq. 6.8.

7.5 Experiments

Robustness Strategies Our goal is to improve the performance of DNN methods in the face of test images that may have strong global color casts due to computational color constancy errors. Based on the KNN framework (Chapter 6) and the modified framework discussed in Sec. 7.4, we examine three strategies to improve the robustness of the DNN models.

(1) The first strategy is to apply a WB correction to each testing image in order to remove any unexpected color casts during the inference time. Note that this approach implicitly assumes that the training images are correctly WB. In our experiments, we used the KNN WB method (Chapter 6) to correct the test images, because it currently achieves the state-of-the-art on white balancing sRGB rendered images. We examined adapting the simple diagonal-based correction – which is applied by traditional WB methods that are

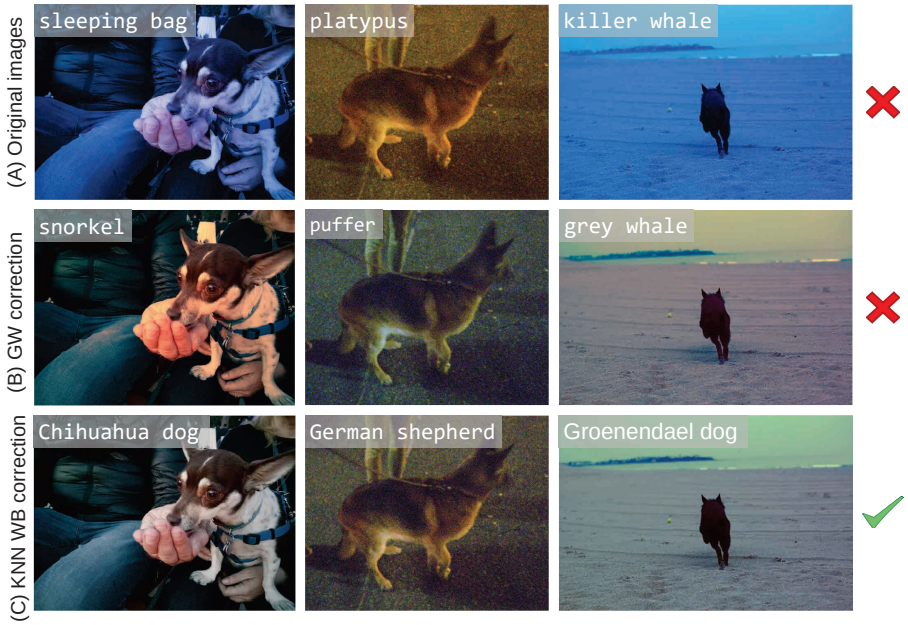


Figure 7.7: (A) Images with different categories of "dogs" rendered with incorrect WB settings. (B) Corrected images using GW [60]. (C) Corrected images using the KNN WB method (Chapter 6). Predicted class by AlexNet is written on top of each image. Images in (A) and (B) are misclassified.

intended to be applied on raw-RGB images (e.g., GW [60]) – but found that they give inadequate results when applied on sRGB images, as also demonstrated in (Chapter 6). In fact, applying diagonal-based correction directly on the training image is similar to multiplicative color jittering. This is why we need to use a nonlinear color manipulation (e.g., polynomial correction estimated by our method in Chapter 6) for more accurate WB correction for sRGB images. An example of the difference is shown in Fig. 7.7.

It is worth mentioning that the training data used by the KNN WB method (Chapter 6) has five fixed color temperatures (2850K, 3800K, 5500K, 6500K, 7500K), all with color correction matrices mapping to their corresponding correct WB. In most cases, one of these five fixed color temperatures will be visually similar to the correct WB. Thus, if the

KNN WB method is applied to an input image that is already correctly white-balanced, the computed transformation will act as an identity.

(2) The second strategy considers the case that some of the training images may include some incorrectly white-balanced images. We, therefore, also apply the WB correction step to all the training images as well as testing images. This again uses the KNN WB method (Chapter 6) on both testing and training images.

(3) The final strategy is to augment the training dataset based on our method described in Sec. 7.4. Like other augmentation approaches, there is no pre-processing correction required. The assumption behind this augmentation process is that the robustness of DNN models can be improved by training on augmented images that serve as exemplars for color constancy errors.

Testing Data Categories Testing images are grouped into two categories. In Category 1 (Cat-1), we expand the original testing images in the CIFAR-10, CIFAR-100, and ADE20K datasets by applying our method to emulate camera WB errors (described in Sec. 7.4). Each test image now has ten (10) variations that share the same ground truth labels. We acknowledge this is less than optimal, given that the same method to modify the testing image is used to augment the training images. However, we are confident in the proposed method's ability to emulate WB errors that we feel Cat-1 images represents real-world examples. With that said, we do not apply strategies 1 and 2 to Cat-1, as the KNN WB method is based on a similar framework used to generate the testing images. For the sake of completeness, we also include Category 2 (Cat-2), which consists of new datasets generated directly from raw-RGB images. Specifically, raw-RGB images are rendered using the full in-camera pipeline to sRGB images with in-camera color constancy errors. As a result, Cat-2's testing images exhibit accurate color constancy errors but con-

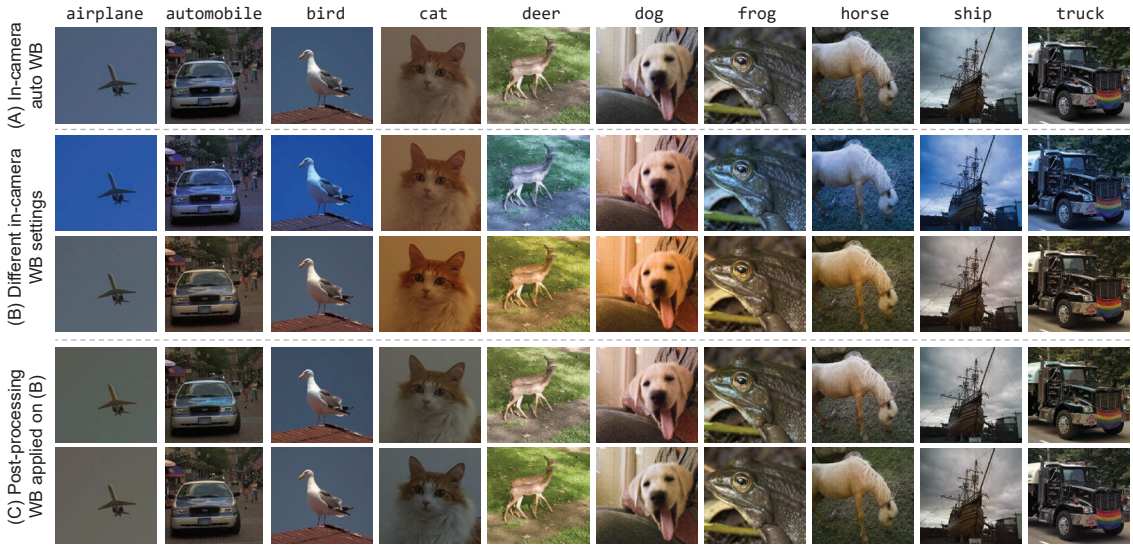


Figure 7.8: Examples of sRGB images used in Cat-2 (i.e., the external testing set of in-camera rendered images). We used this set to evaluate trained models on CIFAR-10 dataset [208]. Class labels of CIFAR-10 dataset are written on top of each column. (A) Images were rendered using the in-camera auto WB setting. (B) Images were rendered with different WB settings. (C) Pre-processing WB correction (Chapter 6) is applied to images in (B).

tain fewer testing images for which we have provided the ground truth labels. Figure 7.8 shows examples from our external testing set.

7.5.1 Experimental Setup

We compare the three above strategies with two existing and widely adopted color augmentation processes: RGB color jittering and HSV jittering.

Our Method The nearest neighbor searching was applied using $k = 25$. The proposed WB augmentation model runs in 7.3 sec (CPU) and 1.0 sec (GPU) to generate *ten* 12-

mega-pixel images. The reported runtime was computed using Intel® Xeon® E5-1607 @ 3.10 GHz CPU and NVIDIA™ Titan X GPU.

Existing Color Augmentation To the best of our knowledge, there is no standardized approach for existing color augmentation methods. Accordingly, we tested different settings and selected the settings that produce the best results.

For RGB color jittering, we generated ten images with new colors by applying a random shift $x \sim \mathcal{N}(\mu_x, \sigma^2)$ to each color channel of the image. For HSV jittering, we generated ten images with new colors by applying a random shift x to the hue channel and multiplying each of the saturation and value channels by a random scaling factor $s \sim \mathcal{N}(\mu_s, \sigma^2)$. We found that $\mu_x = -0.3$, $\mu_s = 0.7$, and $\sigma = 0.6$ give us the best compromise between having color diversity with low color artifacts during the augmentation process.

7.5.2 Network Training

For image classification, training new models on the ImageNet dataset requires unaffordable efforts—for instance, ILSVRC 2012 consists of ∼1 million images and would be ∼10 million images after applying any of the color augmentation techniques. For that reason, we perform experiments on CIFAR-10 and CIFAR-100 datasets [208] due to a more manageable number of images in each dataset.

We trained SmallNet [299] from scratch on CIFAR-10. We also fine-tuned AlexNet [209] to recognize the new classes in CIFAR-10 and CIFAR-100 datasets. As the CIFAR dataset contains 32×32 pixels images, SmallNet was implemented to accept images with these dimensions. In order to fine-tune AlexNet, we rescale all images to 227×227 pixels to fit with the input size of the architecture. For SegNet, the input size was 360×480 pixels. For semantic segmentation, we fine-tuned SegNet [38] on the training set of the ADE20K dataset [409].

We train each model on: (i) the original training images, (ii) the KNN WB method (Chapter 6) applied to the original training images, and (iii) original training images with the additional images produced by color augmentation methods. For color augmentation, we examined RGB color jittering, HSV jittering, and our WB augmentation. Thus, we trained five models for each CNN architecture, each of which was trained on one of the mentioned training settings.

Training was performed using mini-batch stochastic gradient descent with momentum. In our experiments, we used 0.9 momentum. The `L2` regularization factor was set to 0.0005. The mini-batch size was 512 images for SmallNet and AlexNet. For SegNet, the mini-batch size was 4 images due to the GPU memory limitation.

The cross entropy loss was used for image classification (i.e., SmallNet and AlexNet). For image semantic segmentation (i.e., SegNet), we adopted the weighted pixel-wise entropy loss as be suggested by [38]. The assigned weights for each class were computed using the median frequency balancing [103].

The learning rate λ was as follows. For AlexNet's conv1–fc7 layers, we used $\lambda = 10^{-4}$. For AlexNet's fc8 layer, we used $\lambda = 10^{-4} \times 20$. SmallNet and SegNet were trained using $\lambda = 10^{-3}$.

For fair comparisons, we trained each model for the same number of iterations. Specifically, the training was for $\sim$29,000 and $\sim$550,000 iterations for image classification and semantic segmentation tasks, respectively. We adjusted the number of epochs to make sure that each model was trained on the same number of mini-batches for fair comparison between training on augmented and original sets. Note that by using a fixed number of iterations to train models with both original training data and augmented data, we did not fully exploit the full potential of the additional training images when we trained models using additional augmented data.

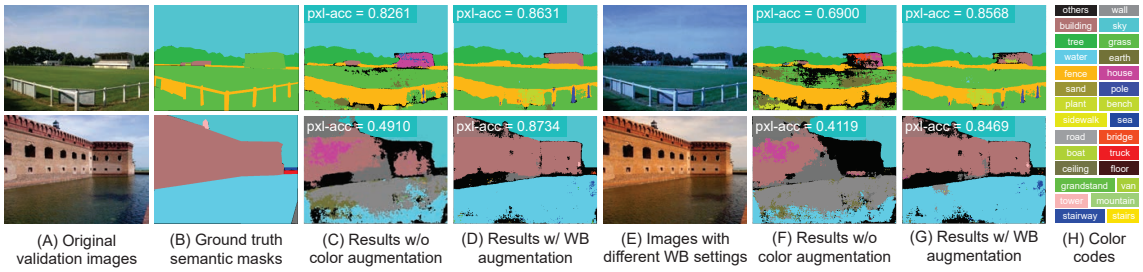


Figure 7.9: Results of SegNet [38] on the ADE20K validation set [409]. (A) Original validation image. (B) Ground truth semantic mask. (C) & (D) Results of trained model wo/w color augmentation using image in (A), respectively. (E) Image with a different WB. (F) & (G) Results w/o and with color augmentation using image in (E), respectively. (H) Color codes. The term 'pxl-acc' refers to pixel-wise accuracy.

7.5.3 Results on Cat-1

Cat-1 tests each model using test images that have been generated by our method described in Sec. 7.4.

Classification We used the CIFAR-10 testing set (10,000 images) to test SmallNet and AlexNet models trained on the training set of the same dataset. We also used the CIFAR-100 testing set (10,000 images) to evaluate the AlexNet model trained on CIFAR-100. After applying our WB emulation to the testing sets, we have 100,000 images for each testing set of CIFAR-10 and CIFAR-100. The top-1 accuracies obtained by each trained model are shown in Table 7.3. The best results on our expanded testing images, which include strong color casts, were obtained using models trained on our proposed WB augmented data.

Interestingly, the experiments show that applying WB correction (Chapter 6) on the training data, in most cases, improves the accuracy using both the original and expanded

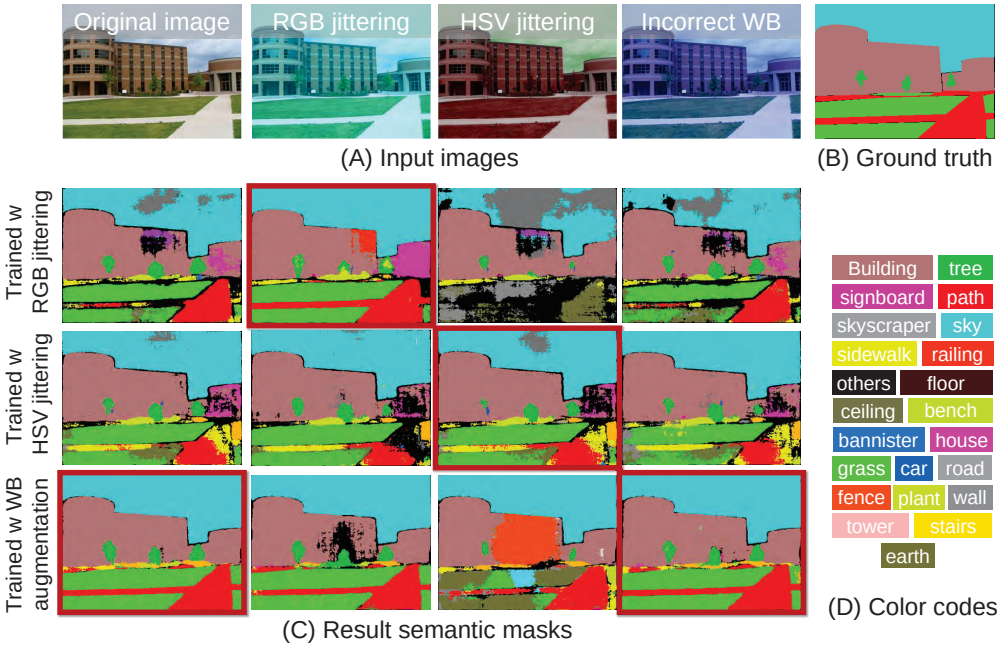


Figure 7.10: (A) Original image. (B) Ground truth semantic mask. (C) Generated by RGB and HSV jittering, and our WB emulation method. (D) color codes. Result masks are obtained by training on augmented data using RGB/HSV jittering and our WB emulation method. The best results are shown in red borders.

test sets. DNNs that were trained on WB augmented training images achieve the best improvement on the original testing images compared to using other color augmenters.

Semantic Segmentation We used the ADE20K validation set using the same setup explained in Sec. 7.3. Table 7.4 shows the obtained pxl-acc and IoU of the trained SegNet models. The best results were obtained with our WB augmentation; Figure 7.9 shows qualitative examples.

It is worth pointing out that when we utilize a certain augmentation technique, we implicitly help the model to expect inputs with similar conditions to what the color augmenter generates. When the testing images having *unrealistic* color manipulations generated by

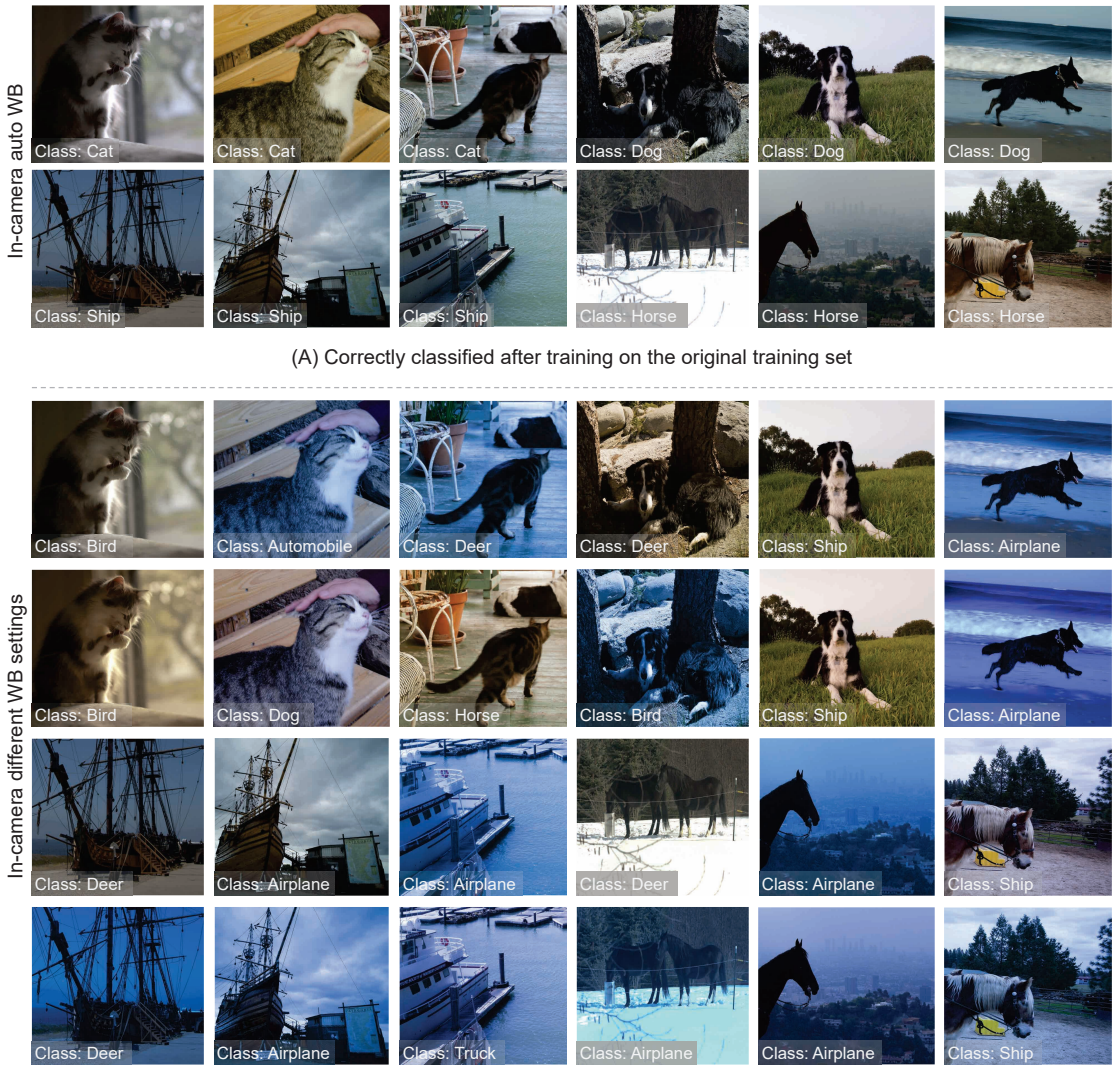


Figure 7.11: (A) Correctly classified images rendered with in-camera auto WB. (B) Misclassified images rendered with *in-camera* different WB. Note that all images in (B) are correctly classified by the same model (AlexNet [209]) trained on WB augmented data.

RGB/HSV jittering, we found that trained models on augmented data by these techniques (i.e., RGB/HSV jittering) are more robust than models trained on other types of images (e.g., original or WB augmented training images). However, for images with color casts

caused by different WB settings, the trained model with our WB augmentation has more resistance than other models; Figure 7.10 shows an example.

7.5.4 Results on Cat-2

Cat-2 data requires us to generate and label our own testing image dataset using raw-RGB images. To this end, we collected 518 raw-RGB images containing CIFAR-10 object classes from the following datasets: HDR+ Burst Photography dataset [155], MIT-Adobe FiveK dataset [62], and Raise dataset [86]. We rendered all raw-RGB images with different color temperatures and two photo-finishing styles using the Adobe Camera Raw SDK. Adobe Camera Raw accurately emulates the ISP onboard a camera and produces results virtually identical to what the in-camera processing would produce (Chapter 6). Images that contain multiple objects were manually cropped to include only the interesting objects—namely, the CIFAR-10 classes. At the end, we generated 15,098 rendered testing images that reflect real in-camera WB settings. We used the following testing sets in our experiments:

(i) In-camera auto WB contains images rendered with the AWB correction setting in Adobe Camera Raw, which mimics the camera's AWB functionality. AWB does fail from time to time; we manually removed images that had a noticeable color cast. This set of images is intended to be equivalent to testing images on existing image classification datasets.

(ii) In-camera WB settings contains images rendered with the different color temperatures and photo-finishing styles. This set represents testing images that contain WB color cast errors.

(iii) WB pre-processing correction applied to set (ii) contains images of set (ii) after applying the KNN WB correction (Chapter 6). This set is used to study the potential improvement of applying a pre-processing WB correction in the inference phase.

Table 7.5 shows the top-1 accuracies obtained by SmallNet and AlexNet on the external testing sets. The experiments show the accuracy is reduced by ∼6% when the testing set is images that have been modified with incorrect WB settings compared with their original accuracies obtained with "properly" white-balanced images using the in-camera AWB. We also notice that the best accuracies are obtained by applying either a pre-processing WB on both training/testing images or our WB augmentation in an end-to-end manner. Examples of misclassified images are shown in Fig. 7.11.

7.6 Summary

This chapter has examined the impact on computational color constancy errors on DNNs for image classification and semantic segmentation. A new method to perform augmentation that accurately mimics WB errors was introduced. We show that both pre-processing WB correction and training DNNs with our augmented WB images improve the results for DNNs targeting CIFAR-10, CIFAR-100, and ADE20K datasets. We believe our WB augmentation method will be useful for other tasks targeted by DNN where image augmentation is sought (see Appendix C).

Table 7.3: [Cat-1] Results of SmallNet [299] and AlexNet [209] on CIFAR dataset [208]. The shown accuracies obtained by models trained on: original training, "white-balanced", and color augmented sets. The testing was performed using: original testing set and testing set with different synthetic WB settings (denoted as diff. WB). The results of the baseline models (i.e., trained on the original training set) are highlighted in green, while the best result for each testing set is shown bold. We highlight best results obtained by color augmentation techniques in yellow. Effects on baseline model results are shown in parentheses.

Cat-1	**SmallNet [299] on CIFAR-10 [208]**	
Training set	Original	Diff. WB
Original training set	0.799	0.655
"White-balanced" set	0.801 (+0.002)	0.683 (+0.028)
HSV augmented set	0.801 (+0.002)	0.747 (+0.092)
RGB augmented set	0.780 (-0.019)	0.765 (+0.11)
WB augmented set (ours)	**0.809 (+0.010)**	**0.786 (+0.131)**
Cat-1	**AlexNet [209] on CIFAR-10 [208]**	
Original training set	**0.933**	0.797
"White-balanced" set	0.932 (-0.001)	0.811 (+0.014)
HSV augmented set	0.923 (-0.010)	0.864 (+0.067)
RGB augmented set	0.922 (-0.011)	0.872 (+0.075)
WB augmented set (ours)	0.926 (-0.007)	**0.889 (+0.092)**
Cat-1	**AlexNet [209] on CIFAR-100 [208]**	
Original training set	**0.768**	0.526
"White-balanced" set	0.757 (-0.011)	0.543 (+0.017)
HSV augmented set	0.722 (-0.044)	0.613 (+0.087)
RGB augmented set	0.723 (-0.045)	0.645 (+0.119)
WB augmented set (ours)	0.735 (-0.033)	**0.670 (+0.144)**

Table 7.4: [Cat-1] Results of SegNet [38] on the ADE20K validation set [409]. The shown intersection-over-union (IoU) and pixel-wise accuracy (pxl-acc) were achieved by models trained using: original training, "white-balanced", and color augmented sets. The testing was performed using: original testing set and testing set with different synthetic WB settings (denoted as diff. WB). Effects on results of SegNet trained on the original training set are shown in parentheses. Highlight marks are as described in Table 7.3.

	IoU	
Cat-1	Original	Diff. WB
Original training set	0.208	0.180
"White-balanced" set	**0.210 (+0.002)**	0.197 (+0.017)
HSV augmented set	0.192 (-0.016)	0.185 (+0.005)
RGB augmented set	0.195 (-0.013)	0.190 (+0.010)
WB augmented set (ours)	0.202 (-0.006)	**0.199 (+0.019)**
Cat-1	**pxl-acc**	
Original training set	0.603	0.557
"White-balanced" set	**0.605 (+0.002)**	0.579 (+0.022)
HSV augmented set	0.583 (-0.020)	0.536 (-0.021)
RGB augmented set	0.544 (-0.059)	0.534 (-0.023)
WB augmented set (ours)	0.597 (-0.006)	**0.581 (+0.024)**

Table 7.5: [Cat-2] Results of SmallNet [299] and AlexNet [209]. The shown accuracies were obtained using trained models on the original training, "white-balanced", and color augmented sets. Effects on results of models trained on the original training set are shown in parentheses. Highlight marks are as described in Table 7.3.

Cat-2	**SmallNet**		
Training Set	In-cam AWB	In-cam Diff. WB	WB pre-processing
Original training set	0.467	0.404	0.461
"White-balanced" set	**0.496 (+0.029)**	0.471 (+0.067)	**0.492 (+0.031)**
HSV augmented set	0.477 (+0.001)	0.462 (+0.058)	0.481 (+0.02)
RGB augmented set	0.474 (+0.007)	0.475 (+0.071)	0.470 (+0.009)
WB augmented set (ours)	0.494 (+0.027)	**0.496 (+0.092)**	0.484 (+0.023)
Cat-2	**AlexNet**		
Original training set	0.792	0.734	0.772
"White-balanced" set	0.784 (-0.008)	0.757 (+0.023)	0.784 (+0.012)
HSV augmented set	0.790 (+0.002)	0.771 (+0.037)	0.779 (+0.007)
RGB augmented set	0.791 (-0.001)	0.779 (+0.045)	0.783 (+0.011)
WB augmented set (ours)	**0.799 (+0.007)**	**0.788 (+0.054)**	**0.787 (+0.015)**

Part IV

Post-Capture White-Balance Editing

8 Interactive White-Balance Editing

Interactive WB editing allows the user to choose WB settings based on preference rather than the AWB estimation. This interactive mechanism is often performed by allowing the user to manually select different regions in a photo as examples of the illumination for WB correction (e.g., clicking on achromatic objects), or by using a color temperature slider to adjust the WB settings. Such interactive editing is possible only with images saved in a raw image format. This is because raw images have no photo-rendering operations applied and photo-editing software is able to apply WB and other photo-finishing procedures to render the final image. Interactively editing WB in camera-rendered images is significantly more challenging, as discussed earlier, because the camera hardware has already applied WB to the image and subsequent nonlinear photo-processing routines. These nonlinear rendering operations make it difficult to change the WB post-capture. The goal of the following chapters, including this one, is to allow interactive WB manipulation of camera-rendered images.

The work in this chapter builds on Chapter 6. We introduce a new framework[1] that is able to link the nonlinear color-mapping functions, introduced in Chapter 6, directly to the user's selected colors to allow interactive WB manipulation. In addition, our framework is more efficient in terms of memory and run-time (99% reduction in memory and 3× speed-

[1]This work was published in [16]: Mahmoud Afifi and Michael S. Brown. Interactive White Balancing for Camera-Rendered Images. In Color and Imaging Conference, 2020.

up). Lastly, we describe how our framework can leverage a simple illumination estimation method (i.e., gray-world) to perform auto-WB correction that is on a par with the WB correction results achieved by our method in Chapter 6. The source code of this work is available on GitHub: https://github.com/mahmoudnafifi/WB_sRGB.

8.1 Introduction

There is photo-editing software (e.g., Adobe Lightroom [199], Skylum [3], Affinity Photo [2]) that enables interactive WB manipulation. Instead of applying AWB, these methods allow the user to select a pixel's RGB values in the image of achromatic scene materials to serve as the estimated illumination color vector. In some scenes, there may be more than one illuminant present and the users can choose which illumination they prefer to correct (see Fig. 8.1). This interactive WB editing, however, is possible only for photos saved in a raw image format. This is because raw images have no photo-finishing applied—instead, the photo-editing software mimics the onboard camera rendering using the user-supplied parameters.

The goal of this chapter is to allow interactive WB manipulation for camera-rendered images. As previously mentioned, WB manipulation in camera-rendered images is challenging due to the nonlinear operations applied by the camera hardware. Our work presented in Chapter 6 showed that even when an exact achromatic reference scene point is known in a camera-rendered image, the conventional diagonal correction method cannot sufficiently remove the color casts caused by WB errors (see Fig. 8.1). To address this issue, we have proposed in Chapter 6 an effective nonlinear polynomial color correction function in lieu of the convention diagonal WB matrix. The work in Chapter 6 used a histogram feature computed from an input image to determine which nonlinear correction function to use to correct a camera-rendered image that had the wrong WB applied. The work in Chapter

Figure 8.1: (A) A camera-rendered image with the wrong WB applied. The user selects two reference colors in the scene representing achromatic scene materials. (B) The results correction using the conventional diagonal WB correction matrix. Due to the nonlinear camera-rendering, the conventional approach is not sufficient to correct the WB. (C) Our method's AWB on (A). (D) Our method's results using the user-supplied reference colors.

6 provided no mechanism to link the nonlinear WB color-mapping functions to the user's selected pixel values.

Contribution We build upon the idea in Chapter 6 and propose a new framework that allows interactive WB editing in camera-rendered images. Our approach works by associating color-cast vectors with rectification functions that output nonlinear color-mapping functions to correct the camera-rendered image's WB. This allows the user to supply a color vector from the image that results in a nonlinear color mapping to modify the image's WB based on the specified color vector. Our framework requires only 1% of the memory used in

our work in Chapter 6 and runs 3× faster, allowing interactive functionality (see Fig. 8.1). In addition, we show that our framework can also be used to perform AWB correction for camera-rendered images that were incorrectly white-balanced with results on a par with our work in Chapter 6.

8.2 Methodology

Given an input image, $\mathbf{I}$, that is rendered with an incorrect or undesired WB setting, the goal is to generate a new output image, $\mathbf{I}_{\texttt{corr}}$, that represents the input $\mathbf{I}$ as it would appear if re-rendered with a new (presumably correct or desired) WB setting. We will refer to the target "ground truth" white-balanced image as $\mathbf{G}$. In the remaining part of this chapter, each image is represented as $3 \times N$ matrix of the $\texttt{R}$, $\texttt{G}$, $\texttt{B}$ triplets, where N is the total number of pixels in the image.

As discussed in previous chapters, a traditional diagonal-based solution relies on determining a 3D vector $\boldsymbol{\gamma} = [\boldsymbol{\gamma}(\texttt{R}), \boldsymbol{\gamma}(\texttt{G}), \boldsymbol{\gamma}(\texttt{B})]^\top$ that represents the scene illuminant color. Since in our application, we are applying a correction not of the scene illumination but instead to a color cast present in the camera-rendered image with the wrong WB applied, we will refer to $\boldsymbol{\gamma}$ as a color-cast vector instead of an illumination vector. In our case, this color-cast vector is provided manually by the user or based on an algorithm when in AWB mode. Given a color-cast vector, the image is assumed to be corrected using the following equation:

$$\mathbf{I}_{\texttt{corr(diag)}} = \mathit{diag}\,(\boldsymbol{\ell})\ \mathbf{I}, \tag{8.1}$$

where $\mathit{diag}\,(\cdot)$ is a 3×3 diagonal matrix of the *color-cast-correction* vector, $\boldsymbol{\ell}$. The vector $\boldsymbol{\ell} = [\boldsymbol{\gamma}(\texttt{G})/\boldsymbol{\gamma}(\texttt{R}),\ \ \texttt{1},\ \ \boldsymbol{\gamma}(\texttt{G})/\boldsymbol{\gamma}(\texttt{B})]^\top$ and represents a simple modification of the color-cast vector.

Due to the nonlinearity applied to camera-rendered images, this simple scaling operation cannot properly correct WB errors. We proposed in Chapter 6 to replace the diagonal correction matrix with a nonlinear color-mapping function that could deal with the nonlinearities in the input $\mathbf{I}$. This function was computed in the following form:

$$\mathbf{I}_{\texttt{corr(poly)}} = r\left(\mathbf{M}\right)\ \phi\left(\mathbf{I}\right), \tag{8.2}$$

where ϕ is a kernel mapping function [166] used in Chapter 6 (i.e., $\phi : \phi\left([\texttt{R},\ \texttt{G},\ \texttt{B}]^\top\right) \rightarrow [\texttt{R}$, $\texttt{G}$, $\texttt{B}$, $\texttt{RG}$, $\texttt{RB}$, $\texttt{GB}$, $\texttt{R}^2$, $\texttt{G}^2$, $\texttt{B}^2$, $\texttt{RGB}$, $1]^\top$), $\mathbf{M} \in \mathbb{R}^{33}$ is a vectorized form of the polynomial mapping matrix, and $r\left(\cdot\right)$ is a reshaping function that constructs the 3×11 matrix from the vectorized version, $\mathbf{M}$. As done in Chapter 6, this polynomial matrix can be computed in a closed-form via standard least squares methods.

Though this nonlinear color mapping achieves superior results compared to the diagonal-based correction, it lack a correlation to a color-cast vector. In the following section, we describe how to efficiently associate these color-mapping functions with color vectors that can be used for WB correction in a camera-rendered image $\mathbf{I}$. Fig. 8.2 provides an overview of our framework.

8.2.1 Training Phase

The first step of our method is to have a large number of training examples that exhibits a wide range of WB errors in camera-rendered images. We used the Rendered WB dataset proposed in Chapter 6, which contains ∼65,000 pairs of improperly white-balanced camera-rendered (sRGB) images and their corresponding ground truth white-balanced sRGB images. All images have WB settings applied in the sensor raw space followed by an emulation of in-camera nonlinear rendering operations to get the final camera-rendered images. This dataset consists of two sets: (i) the training set, referred to as Set 1, and (ii) the testing

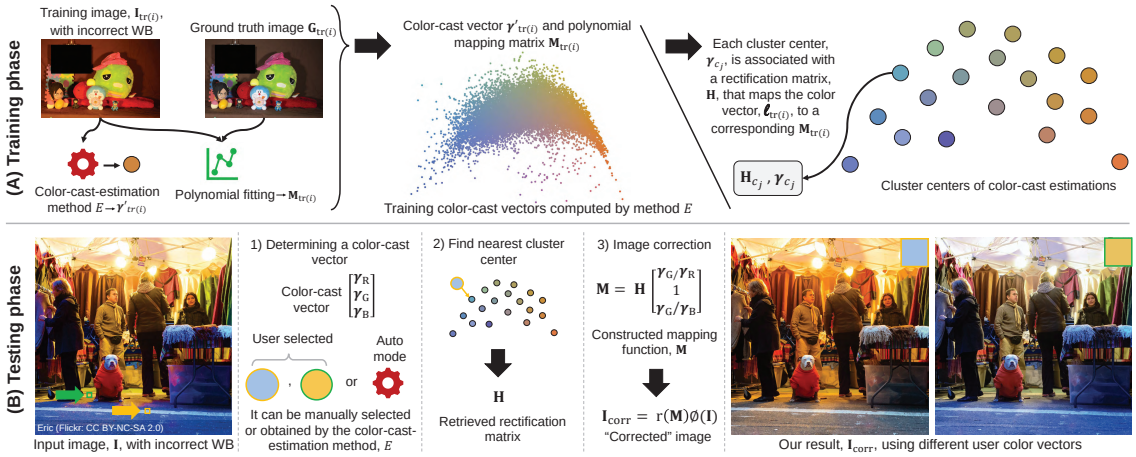


Figure 8.2: Overview of our proposed method. (A) [Training Phase] For each image in the training dataset, we estimate a color-cast vector using an off-the-shelf-illumination estimation algorithm E (e.g., GW [60]). A nonlinear color-mapping function that maps this training image to a correctly white-balanced image is also computed. The color-cast vectors of each training image are clustered. A *rectification function* is computed for each cluster that returns a color-mapping function based on a color-cast vector. (B) [Testing Phase] When applying our method, the user either manually provides a color-cast vector or uses the method E to predict a color-cast vector. Using this color-cast vector, the most similar cluster in the training data is found and its rectification function is used to compute a mapping function $\mathbf{M}$ that is applied to correct the image.

set, referred to as Set 2. Fig. 8.3 shows example images taken from the Rendered WB dataset.

We used all training images in Set 1 to compute the correction functions $\mathbf{M}$ described in Eq. 8.2. For each pair, i, of an improperly white-balanced image, $\mathbf{I}_{\mathtt{tr}(i)}$, and the corresponding ground truth image, $\mathbf{G}_{\mathtt{tr}(i)}$, we compute our 33-dimensional vectorized polyno-

Figure 8.3: This figure shows examples from our Rendered WB dataset (proposed in Chapter 6) used in order to generate our training rectification functions. (A) Three examples of the same scene rendered with different incorrect WB settings. (B) The ground truth white-balanced image.

mial mapping matrix $\mathbf{M}_{\mathtt{tr}(i)}$ as follows:

$$\underset{\mathbf{M}_{\mathtt{tr}(i)}}{\arg\min} \left\| r\left(\mathbf{M}_{\mathtt{tr}(i)}\right) \;\; \phi\left(\mathbf{I}_{\mathtt{tr}(i)}\right) - \mathbf{G}_{\mathtt{tr}(i)} \right\|_{\mathrm{F}}, \tag{8.3}$$

where $\|.\|_{\mathrm{F}}$ is the Frobenius norm. Afterwards, each training image, $\mathbf{I}_{\mathtt{tr}(i)}$, is associated with a color-cast vector $\boldsymbol{\gamma}'_{\mathtt{tr}(i)}$, which will be later used to compute a color-cast-correction vector $\boldsymbol{\ell}_{\mathtt{tr}(i)}$. This color-cast vector can be computed using any off-the-shelf illuminant estimation algorithm, $E : E\left(\mathbf{I}_{\mathtt{tr}(i)}\right) \rightarrow \boldsymbol{\gamma}'_{\mathtt{tr}(i)}$.

We cluster the training data based on their color-cast vectors into k clusters. In our experiments, we used k-means++ [36] with a cosine similarity distance metric and set k to 50. Each cluster, noted as $\mathbf{c}$, also has a number of $\mathbf{M}_i$ mapping functions associated with it, where $i \in \mathbf{c}$. Instead of storing all of these mapping functions, we derive a single

mapping function, termed a *rectification function*, that can estimate the color-mapping function based on the polynomial matrix $\mathbf{M}$. This is described in the following section.

8.2.2 Rectification Function

Our rectification function is inspired by a bias-correction method proposed to rectify scene illuminant estimation errors [111]. Specifically, we propose a rectification function, $\mathbf{H}$, that maps a color-cast-correction vector, $\boldsymbol{\ell}$, directly to a nonlinear correction matrix as follows:

$$\mathbf{M} = \mathbf{H}\ \boldsymbol{\ell}, \tag{8.4}$$

where $\mathbf{M}$ is a 33×1 vectorized polynomial matrix computed to map the colors of an incorrectly white-balanced image, $\mathbf{I}$, into the corresponding colors of the correctly white-balanced image, $\mathbf{G}$, $\mathbf{H}$ is our 33×3 rectification matrix, and $\boldsymbol{\ell}$ is the color-cast-correction vector computed from the color-cast vector as described earlier. This type of correction function was used by Finlayson [111] to correct biases made by illumination estimation algorithms. In [111], an estimated illumination vector would be mapped to a new illumination vector that was closer to the ground truth based on the training dataset. In our case, the function $\mathbf{H}$ maps $\boldsymbol{\ell}$ to its corresponding matrix $\mathbf{M}$, allowing us to connect a color-cast vector directly to a mapping function.

Working from Eq. 8.4, we compute a rectification matrix for each cluster, denoted as $\mathbf{H}_{\mathbf{c}(j)}$ for cluster j. Let n be the number of training examples belonging to each cluster $\mathbf{c}(j)$, s.t. $j \in [1, 2...k]$. For each cluster, we minimize the following equation to compute its rectification matrix:

$$\underset{\mathbf{H}_{\mathbf{c}(j)}}{\arg\min} \left\| \sum_{i \in \mathbf{c}(j)} \mathbf{H}_{\mathbf{c}(j)}\ \boldsymbol{\ell}_i - \mathbf{M}_i \right\|_{\mathrm{F}}, \tag{8.5}$$

where $\boldsymbol{\ell}_i$ is a 3×1 color-cast-correction vector of the i^{th} training example in cluster $\mathbf{c}(j)$, $\mathbf{M}_i \in \mathbb{R}^{33}$ is a vectorized polynomial matrix associated with the i^{th} training example in

cluster $\mathbf{c}(j)$, and $\mathbf{H}_{\mathbf{c}(j)}$ is a 33×3 rectification matrix assigned to cluster $\mathbf{c}(j)$. Eq. 8.5 essentially estimates a single $\mathbf{H}$ per cluster that minimizes the error over all $\boldsymbol{\ell}_i$ associated with this cluster.

After this procedure, each color-cast cluster is now represented by the mean color-cast vector, $\boldsymbol{\gamma}_{c_j}$, of all color-cast vectors that belong to it. For each cluster, we store the corresponding rectification matrix, $\mathbf{H}_{\mathbf{c}(j)}$, to be used in the testing phase. This model requires only 0.04 MB memory to encode our 50 rectification functions, compared to $\sim$25 MB required by our method in Chapter 6 ($\sim$99% reduction in memory requirements).

8.2.3 Testing Phase

Our method can easily be used in an AWB mode. To do this, given an input image $\mathbf{I}$, the same illuminant estimation algorithm, E, used in the training phase to compute the illuminant vector, $\boldsymbol{\gamma}^{'}$, is applied. Afterwards, we search among the pre-computed color-cast cluster centers to find the closest cluster $\mathbf{c}(h)$ to our testing color vector. Then, the rectification function of the closest cluster is retrieved and used along with the computed color-cast-correction vector, $\boldsymbol{\ell}^{'}$, to correct the testing image as described in the following equation:

$$\mathbf{I}_{\texttt{corr}} = r\left(\mathbf{H}_{\mathbf{c}(h)}\ \boldsymbol{\ell}^{'}\right)\ \phi\left(\mathbf{I}\right). \tag{8.6}$$

In the user-interactive mode, we use the color-cast vector supplied by the user of some achromatic reference point in the image instead of the estimated color vector $\boldsymbol{\gamma}^{'}$. Our method requires $\sim$0.5 seconds to correct a 12-mega-pixel image on an Intel® Xeon® E5-1607 @ 3.10 GHz machine, compared to 1.5 seconds required by our method in Chapter 6 using a similar machine. Our approach can significantly improve the results of diagonal-based methods (e.g., [60, 120, 171, 362]) to correct improperly white-balanced images.

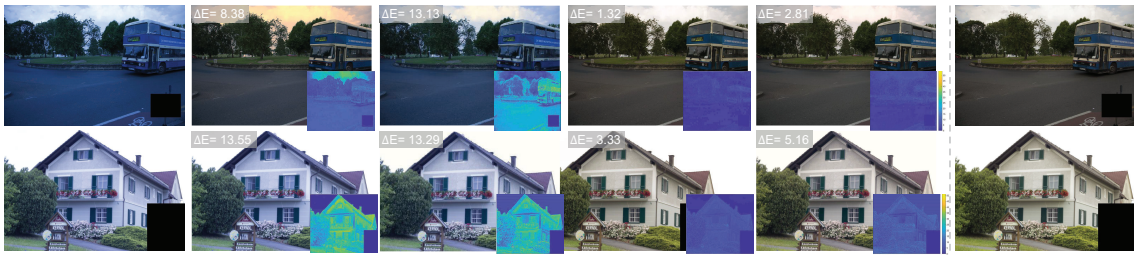


Figure 8.4: Qualitative results on our Rendered WB dataset (Chapter 6) and the rendered version of the Cube+ dataset [41]. (A) Input images. (B) Diagonal correction using the GW method [60]. (C) Diagonal correction using the quasi-unsupervised color constancy (quasi-U CC) method [51]. (D) Results of the KNN WB method (Chapter 6). (E) Results of applying our rectification function to initial estimation of GW. (F) Ground truth images.

8.3 Experimental Results

We evaluate our method extensively through quantitative and qualitative comparisons with existing solutions for AWB and user-interactive WB correction. Sec. 8.3.1 provides quantitative evaluation of our method, while qualitative comparisons are provided in Sec. 8.3.2.

8.3.1 Quantitative Evaluation

We evaluated our AWB correction results using ∼13,000 testing images from the Rendered WB testing set (Set 2) (Chapter 6). In addition, we rendered the Cube+ dataset [41] in the same way of rendering our dataset in Chapter 6. We also used this rendered version of the Cube+ dataset in our evaluation. As mentioned in Sec. 8.2, any off-the-shelf illumination estimation method E can be used. We tested different illuminant estimation methods in our AWB framework. Specifically, we utilized the following methods: the GW [60], the

Figure 8.5: Our method allows the user to manually select achromatic points in order to improve the results. (A) Input images. (B) Results of the GW method [60]. (C) Results of the quasi-unsupervised color constancy (quasi-U CC) method [51]. (D) Results of the KNN WB method (Chapter 6). (E) Our results using the GW's estimated illuminant colors. (F) Our results using manually selected achromatic reference points (see the shown arrows).

SoG [120], and the FC4 [171] methods. In each experiment, we use the training data of our Rendered WB dataset (Chapter 6) to compute our rectification functions, as described in Sec. 8.2.2. For each of the illuminant estimation methods, we compare our results with the diagonal correction with and without linearizing the testing images. The linearizing process was performed using the standard de-gamma linearization operation [31, 101]. We include this linearization process in our comparisons as it is a common misconception that a simple gamma linearization can remove the nonlinearity applied by cameras.

For the sake of completeness, we also compare our results with our recent nonlinear method for post-capture KNN WB correction proposed in Chapter 6). Table 8.1 shows the first, second, and third quantile and the mean of the error values obtained by each method. We followed the evaluation metrics used in Chapter 6, which are: (i) MSE, (ii)

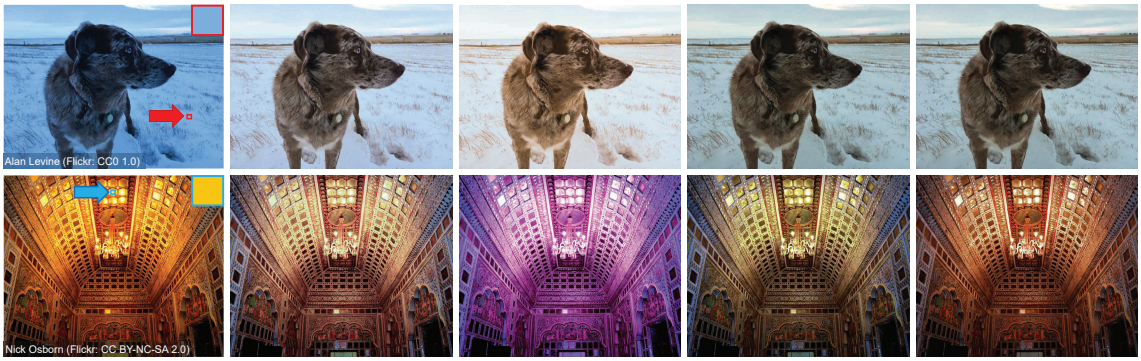


Figure 8.6: Comparison with Adobe Lightroom WB correction. (A) Input image. (B) and (C) Adobe Lightroom's auto WB and manual WB correction results, respectively. (D) and (E) Our auto and manual WB correction results, respectively.

MAE, and (iii) $\triangle\mathbf{E}$ 2000 [336].

As shown in Table 8.1, our rectification function significantly improves the results of diagonal-based methods and achieves results on a par with the WB-sRGB method on both testing sets. As can be seen from the results, our method reduces the MSE by ∼55%, the MAE by ∼32%, and the $\triangle$E 2000 by ∼35 % on average compared to the gamma linearization process, which reduces the MSE by ∼15%, the MAE by ∼5%, and the $\triangle$E 2000 by ∼5% on average.

8.3.2 Qualitative Evaluation

We qualitatively evaluated our method against different methods, including a commercial photo-editing software, for auto and user-interactive WB correction. Figure 8.4 shows a comparison between our results and the diagonal correction of two illuminant estimation methods—namely, the GW method [60] and the quasi-unsupervised color constancy method [51]. As shown, our AWB is superior to diagonal WB and achieves similar results

to our KNN WB method proposed in Chapter 6.

In contrast to the KNN WB method (Chapter 6), our method allows interactive correction to improve the results by manually adjusting the color-cast color. Figure 8.5 shows that this manual adjustment feature produces arguably visually superior results compared to the KNN WB method.

We further compared our method against Adobe Lightroom, as it is one of the most common photo-editing software programs that provide the same manual WB correction feature. As can be seen in Fig. 8.6, our method produces perceptibly superior results in comparison with Adobe Lightroom's results for both auto and manual WB correction.

8.4 Summary

We have introduced an interactive WB method for use on camera-rendered images which allows the user to directly specify scene points to be used for white balancing. Previously, this type of interaction could be performed only by photo-editing software operating on raw images. We have enabled this feature for camera-rendered images that already have a white-balance correction applied as well as additional photo-finishing. Our method works by efficiently computing nonlinear color-correction mappings based on user-supplied color-cast vectors directly from the camera-rendered image. We also showed how our method can easily perform auto-WB correction in a camera-rendered image. Our method enables a new photo-editing feature for color manipulation.

Table 8.1: Quantitative results on the Rendered WB testing set (Set 2) and the rendered version of the Cube+ dataset [41]. We applied our rectification function (RF) to the estimated illuminants of different methods. Our RF results are highlighted in gray. The term "linearized" refers to applying the standard gamma linearization [31, 101] to images before estimating and correcting images. The terms Q1, Q2, and Q3 denote the first, second (median), and third quartile, respectively. The terms MSE and MAE stand for mean square error and mean angular error, respectively. The best results are highlighted in yellow and boldface.

Method	MSE				MAE				△E 2000			
	Mean	Q1	Q2	Q3	Mean	Q1	Q2	Q3	Mean	Q1	Q2	Q3
Testing set of the Rendered WB dataset (Set 2): DSLR and mobile phone cameras (2,881 images)												
GW [60]	500.18	173.69	332.75	615.40	8.89°	5.82°	8.32°	11.33°	10.74	7.92	10.29	13.12
SoG [120]	429.35	147.05	286.84	535.72	9.54°	5.72°	8.85°	12.65°	10.01	7.09	9.85	12.69
FC4 [171]	662.53	304.88	524.42	817.57	8.92°	5.94°	8.03°	10.84°	12.12	8.94	11.79	14.76
GW (linearized) [60]	469.86	163.07	312.28	574.85	8.61°	5.44°	7.94°	10.93°	10.68	7.70	10.13	13.15
SoG (linearized) [120]	393.85	137.21	267.37	497.40	8.96°	5.31°	8.26°	11.97°	9.81	6.87	9.67	12.46
FC4 (linearized) [171]	505.30	142.46	307.77	635.35	10.37°	5.31°	9.26°	14.15°	10.82	7.39	, 10.64	13.77
GW [60] + our RF	207.13	46.71	111.89	230.10	5.35°	2.89°	4.59°	6.84°	6.74	4.45	6.15	8.45
SoG [120] + our RF	256.10	55.93	132.09	266.61	6.25°	3.28°	5.20°	8.20°	7.27	4.77	6.61	9.05
FC4 [171] + our RF	303.99	50.01	118.88	298.44	6.61°	2.99°	4.99°	8.40°	7.28	4.30	6.22	9.33
KNN WB (Chapter 6)	**171.09**	**37.04**	**87.04**	**190.88**	**4.48°**	**2.26°**	**3.64°**	**5.95°**	**5.60**	**3.43**	**4.90**	**7.06**
Rendered Cube+ dataset with different WB settings (10,242 images)												
GW [60]	312.62	55.16	159.63	358.02	6.85°	3.08°	5.76°	9.70°	9.01	5.35	8.38	12.08
SoG [120]	269.31	21.92	90.37	312.02	6.69°	2.3°	4.63°	9.62°	7.70	3.40	6.38	11.07
FC4 [171]	410.01	79.26	219.05	505.71	6.7°	3.26°	5.45°	8.7°	10.4	6.51	9.73	13.43
GW (linearized) [60]	244.59	32.58	121.42	300.99	6.37°	2.51°	5.13°	9.09°	8.05	4.18	7.25	11.08
SoG (linearized) [120]	275.33	17.16	67.49	309.97	6.66°	2.08°	4.17°	9.58°	7.57	3.00	5.73	11.05
FC4 (linearized) [171]	371.9	79.15	213.41	467.33	6.49°	3.34°	5.59°	8.59°	10.38	6.6	9.76	13.26
GW [60] + our RF	**159.88**	21.94	54.76	125.02	4.64°	2.12°	3.64°	5.98°	6.2	3.28	5.17	7.45
SoG [120] + our RF	226.83	20.01	58.61	165.03	5.33°	2.1°	3.83°	6.97°	6.61	3.17	5.38	8.56
FC4 [171] + our RF	175.73	**17.8**	**43.65**	**114.65**	4.67°	1.89°	**3.10°**	5.48°	5.7	**2.95**	4.63	7.05
KNN WB (Chapter 6)	194.98	27.43	57.08	118.21	**4.12°**	**1.96°**	3.17°	**5.04°**	**5.68**	3.22	**4.61**	**6.70**

9 Deep White-Balance Editing

In this chapter, we introduce a deep learning approach[1] to realistically edit an sRGB image's white balance. As discussed in previous chapters, cameras capture sensor images that are rendered by their ISPs to a sRGB color space encoding. The ISP rendering begins with a white-balance procedure that is used to remove the color cast of the scene's illumination. The ISP then applies a series of nonlinear color manipulations to enhance the visual quality of the final sRGB image. In Chapter 6, we showed that sRGB images that were rendered with the incorrect white balance cannot be easily corrected due to the ISP's nonlinear rendering. The work in Chapter 6 proposed a KNN solution based on tens of thousands of image pairs. In this chapter, we propose to solve this problem with a DNN architecture trained in an end-to-end manner to learn the correct white balance. Our DNN maps an input image to two additional white-balance settings corresponding to indoor and outdoor illuminations. Our solution not only is more accurate than the KNN approach in terms of correcting a wrong white-balance setting but also provides the user the freedom to edit the white balance in the sRGB image to other illumination settings. The source code of this work is available on GitHub: https://github.com/mahmoudnafifi/Deep_White_Balance.

[1]Work done while the author was an intern at Samsung AI Center – Toronto; This work was published in [15]: Mahmoud Afifi and Michael S. Brown. Deep White-Balance Editing. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2020.

Figure 9.1: Our deep white-balance editing framework produces compelling results and generalizes well to images outside our training data (e.g., image above taken from an Internet photo repository). Top: input image captured with a wrong WB setting. Bottom: our framework's AWB, Incandescent WB, and Shade WB results. Photo credit: *M@tth1eu* Flickr–CC BY-NC 2.0.

9.1 Introduction

While the goal of WB is intended to normalize the effect of the scene's illumination, ISPs often incorporate aesthetic considerations in their color rendering based on photographic preferences. Such preferences do not always conform to the white light assumption and can vary based on different factors, such as cultural preference and scene content [45, 76, 170, 333].

Most digital cameras provide an option to adjust the WB settings during image capturing. However, once the WB setting has been selected and the image is fully processed by the ISP to its final sRGB encoding it becomes challenging to perform WB editing without access to the original unprocessed raw-RGB image. This problem becomes even more

difficult if the WB setting was wrong, which results in a strong color cast in the final sRGB image.

The ability to edit the WB of an sRGB image not only is useful from a photographic perspective but also can be beneficial for computer vision applications, such as object recognition, scene understanding, and color augmentation [42,138]. Our study in Chapter 7 showed that images captured with an incorrect WB setting produce a similar effect of an untargeted adversarial attack for DNN models.

Contribution We present a novel deep learning framework that allows realistic post-capture WB editing of sRGB images. Our framework consists of a single encoder network that is coupled with three decoders targeting the following WB settings: (1) a "correct" AWB setting; (2) an indoor WB setting; (3) an outdoor WB setting. The first decoder allows an sRGB image that has been incorrectly white-balanced image to be edited to have the correct WB. This is useful for the task of post-capture WB correction. The additional indoor and outdoor decoders provide users the ability to produce a wide range of different WB appearances by blending between the two outputs. This supports photographic editing tasks to adjust an image's aesthetic WB properties. We provide extensive experiments to demonstrate that our method generalizes well to images outside our training data and achieves state-of-the-art results for both tasks.

9.2 Methodology

9.2.1 Problem formulation

Given an sRGB image, $\mathbf{I}_{\text{WB}^{(\text{in})}}$, rendered through an unknown camera ISP with an arbitrary WB setting $\text{WB}^{(\text{in})}$, our goal is to edit its colors to appear as if it were re-rendered with a target WB setting $\text{WB}^{(t)}$.

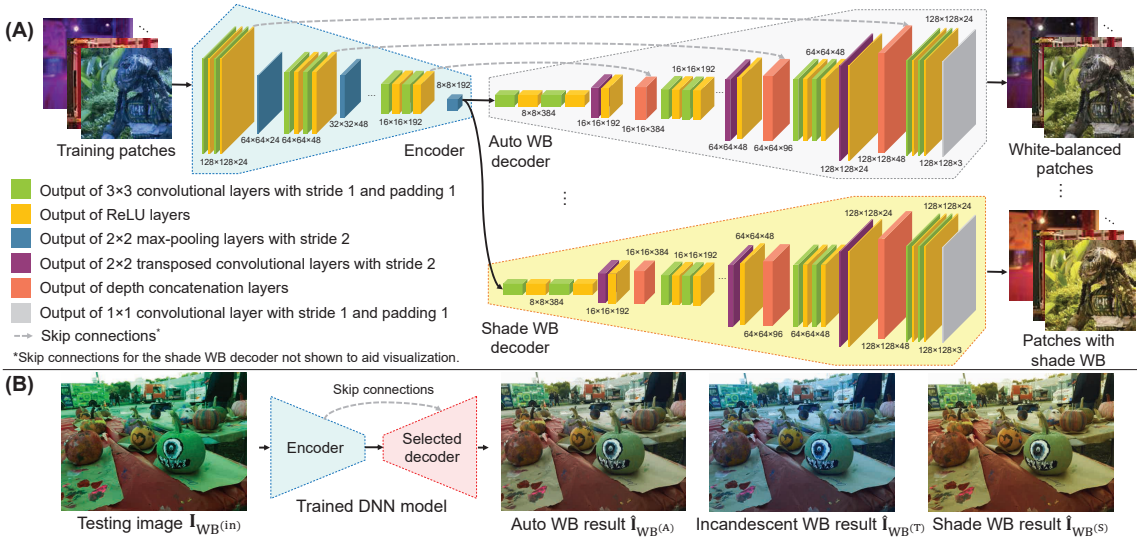


Figure 9.2: Proposed multi-decoder framework for sRGB WB editing. (A) Our proposed framework consists of a single encoder and multiple decoders. The training process is performed in an end-to-end manner, such that each decoder "re-renders" the given training patch with a specific WB setting, including AWB. For training, we randomly select image patches from the Rendered WB dataset (Chapter 6). (B) Given a testing image, we produce the targeted WB setting by using the corresponding trained decoder.

As mentioned in Sec. 9.1, our task can be accomplished accurately if the original unprocessed raw-RGB image is available. If we could recover the unprocessed raw-RGB values, we can change the WB setting $\mathrm{WB}^{(\mathrm{in})}$ to $\mathrm{WB}^{(t)}$, and then re-render the image back to the sRGB color space with a software-based ISP. This ideal process can be described by the following equation:

$$\mathbf{I}_{\mathrm{WB}^{(t)}} = G\left(F\left(\mathbf{I}_{\mathrm{WB}^{(\mathrm{in})}}\right)\right), \tag{9.1}$$

where $F : \mathbf{I}_{\mathrm{WB}^{(\mathrm{in})}} \rightarrow \mathbf{D}_{\mathrm{WB}^{(\mathrm{in})}}$ is an unknown reconstruction function that reverses the camera-rendered sRGB image $\mathbf{I}$ back to its corresponding raw-RGB image $\mathbf{D}$ with the current $\mathrm{WB}^{(\mathrm{in})}$ setting applied and $G : \mathbf{D}_{\mathrm{WB}^{(\mathrm{in})}} \rightarrow \mathbf{I}_{\mathrm{WB}^{(t)}}$ is an unknown camera rendering

Figure 9.3: We consider the runtime performance of our method to be able to run on limited computing resources (∼1.5 seconds on a single CPU to process a 12-megapixel image). First, our DNN processes a downsampled version of the input image, and then we apply a global color mapping to produce the output image in its original resolution. Shown input image is rendered from the MIT-Adobe FiveK dataset [62].

function that is responsible for editing the WB setting and re-rendering the final image.

9.2.2 Method Overview

Our goal is to model the functionality of $G\left(F\left(\cdot\right)\right)$ to generate $\mathbf{I}_{\mathrm{WB}^{(t)}}$. We first analyze how the functions G and F cooperate to produce $\mathbf{I}_{\mathrm{WB}^{(t)}}$. From Eq. 9.1, we see that the function F transforms the input image $\mathbf{I}_{\mathrm{WB}^{(\mathrm{in})}}$ into an intermediate representation (i.e., the raw-RGB image with the captured WB setting), while the function G accepts this intermediate representation and renders it with the target WB setting to an sRGB color space encoding.

Due to the nonlinearities applied by the ISP's rendering chain, we can think of G as a hybrid function that consists of a set of sub-functions, where each sub-function is responsible for rendering the intermediate representation with a specific WB setting.

Our ultimate goal is *not* to reconstruct/re-render the original raw-RGB values, but rather to generate the final sRGB image with the target WB setting $\mathrm{WB}^{(t)}$. Therefore, we can model the functionality of $G\left(F\left(\cdot\right)\right)$ as an encoder/decoder scheme. Our encoder f transfers the input image into a latent representation, while each of our decoders (g_1,

g_2, ...) generate the final images with a different WB setting. Similar to Eq. 9.1, we can formulate our framework as follows:

$$\hat{\mathbf{I}}_{\mathrm{WB}^{(\mathrm{t})}} = g_t \left(f \left(\mathbf{I}_{\mathrm{WB}^{(\mathrm{in})}} \right) \right), \tag{9.2}$$

where $f : \mathbf{I}_{\mathrm{WB}^{(\mathrm{in})}} \rightarrow \boldsymbol{\mathcal{Z}}$, $g_t : \boldsymbol{\mathcal{Z}} \rightarrow \hat{\mathbf{I}}_{\mathrm{WB}^{(\mathrm{t})}}$, and $\mathcal{Z}$ is an intermediate representation (i.e., latent representation) of the original input image $\mathbf{I}_{\mathrm{WB}^{(\mathrm{in})}}$.

Our goal is to make the functions f and g_t independent, such that changing g_t with a new function g_y that targets a different WB y, does not require any modification in f, as it is the case in Eq. 9.1.

In our work, we target three different WB settings: (i) $\mathrm{WB}^{(\mathrm{A})}$: AWB – representing the correct lighting of the captured image's scene; (ii) $\mathrm{WB}^{(\mathrm{T})}$: Tungsten/ Incandescent – representing WB for indoor lighting; and (iii) $\mathrm{WB}^{(\mathrm{S})}$: Shade – representing WB for outdoor lighting. This gives rise to three different decoders (g_A, g_T, and g_S) that are responsible for generating output images that correspond to AWB, Incandescent WB, and Shade WB.

The Incandescent and Shade WB are specifically selected based on the color properties. This can be understood when considering the illuminations in terms of their correlated color temperatures. For example, Incandescent and Shade WB settings are correlated to 2850 Kelvin (K) and $7500K$ color temperatures, respectively. This wide range of illumination color temperatures consider the range of pleasing illuminations [210, 302]. Moreover, the wide color temperature range between Incandescent and Shade allows the approximation of images with color temperatures within this range by interpolation. The details of this interpolation process are explained in Sec. 9.2.5. Note that there is no fixed correlated color temperature for the AWB mode, as it changes based on the input image's lighting conditions.

9.2.3 Multi-Decoder Architecture

An overview of our DNN's architecture is shown in Fig. 9.2. We use a U-Net architecture [323] with multi-scale skip connections between the encoder and decoders. Our framework consists of two main units: the first is a 4-level encoder unit that is responsible for extracting a multi-scale latent representation of our input image; the second unit includes three 4-level decoders. Each unit has a different bottleneck and transpose convolutional (conv) layers. At the first level of our encoder and each decoder, the conv layers have 24 channels. For each subsequent level, the number of channels is doubled (i.e., the fourth level has 192 channels for each conv layer).

9.2.4 Training Phase

Training Data We adopt our Rendered WB dataset produced in Chapter 6 to train and validate our model. This dataset includes ∼65,000 sRGB images rendered by different camera models and with different WB settings, including the Shade and Incandescent settings. For each image, there is also a corresponding ground truth image rendered with the correct WB setting (considered to be the correct AWB result). This dataset consists of two subsets: training set (Set 1) and testing set (Set 2). The training set (Set 1) is divided into three folds – two for training and one for validation. Specifically, we randomly selected 12,000 training images from two folds and 2,000 validation images from the remaining fold. For each training image, we have three ground truth images rendered with: (i) the correct WB (denoted as AWB), (ii) Shade WB, and (iii) Incandescent WB. The validation results on Set 1 of the Rendered WB dataset are reported in Sec. 9.3.

Data Augmentation We also augment the training images by rendering an additional 1,029 raw-RGB images, of the same scenes included in our Rendered WB dataset (Chapter

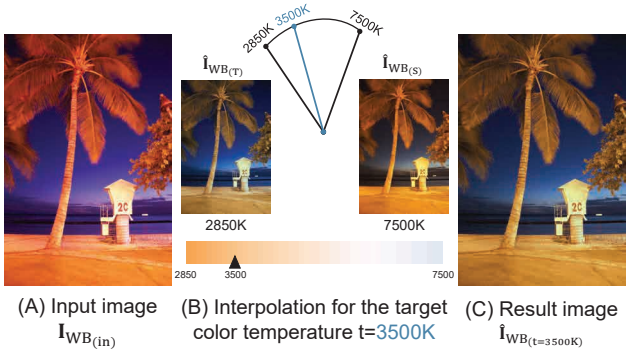


Figure 9.4: In addition to our AWB correction, we train our framework to produce two different color temperatures (i.e., Incandescent and Shade WB settings). We interpolate between these settings to produce images with other color temperatures. (A) Input image. (B) Interpolation process. (C) Final result. Shown input image is taken from the rendered version of the MIT-Adobe FiveK dataset.

6), but with random color temperatures. At each epoch, we randomly select four 128×128 patches from each training image and their corresponding ground truth images for each decoder and apply geometric augmentation (rotation and flipping) as an additional data augmentation to avoid overfitting.

Loss Function We trained our model to minimize the L_1-norm loss function between the reconstructed and ground truth patches:

$$\sum_{i} \sum_{p=1}^{3hw} \left| \mathbf{P}_{\mathrm{WB}^{(i)}}(p) - \mathbf{C}_{\mathrm{WB}^{(i)}}(p) \right|, \tag{9.3}$$

where h and w denote the patch's height and width, and p indexes into each pixel of the training patch $\mathbf{P}$ and the ground truth camera-rendered patch $\mathbf{C}$, respectively. The index $i \in \{\mathrm{A}, \mathrm{T}, \mathrm{S}\}$ refers to the three target WB settings. We also have examined the squared L_2-norm loss function and found that both loss functions work well for our task.

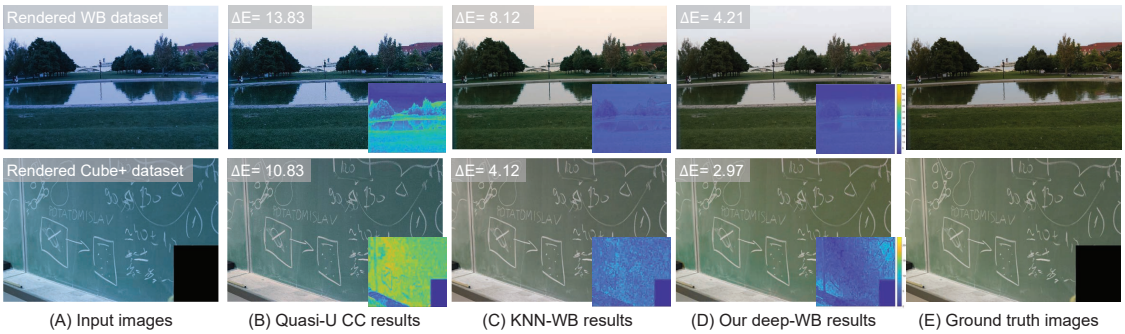


Figure 9.5: Qualitative comparison of AWB correction. (A) Input images. (B) Results of quasi-U CC [51]. (C) Results of KNN WB (Chapter 6). (D) Our results. (E) Ground truth images. Shown input images are taken from our Rendered WB dataset (Chapter 6) and the rendered version of Cube+ dataset [41] (see Chapter 8 for more details about the rendered version of Cube+ dataset).

Training Parameters We initialized the weights of the conv layers using He's initialization [158]. The training process is performed for 165,000 iterations using the Adam optimizer [204], with a decay rate of gradient moving average $\beta_1 = 0.9$ and a decay rate of squared gradient moving average $\beta_2 = 0.999$. We used a learning rate of 10^{-4} and reduced it by 0.5 every 25 epochs. The mini-batch size was 32 training patches per iteration.

9.2.5 Testing Phase

Color mapping procedure Our DNN model is a fully convolutional network and is able to process input images in their original dimensions with the restriction that the dimensions should be multiples of 2^4, as we use 4-level encoder/decoders with 2×2 max-pooling and transpose conv layers. However, to ensure a consistent run time for any sized input images, we resize all input images to a maximum dimension of 656 pixels. Our DNN is applied on this resized image to produce image $\hat{\mathbf{I}}_{\mathrm{WB}^{(i)}\downarrow}$ with the target WB setting $i \in \{\mathrm{A}, \mathrm{T}, \mathrm{S}\}$.

We then compute a color mapping function between our resized input and output image. As done in Chapter 6, we computed a polynomial mapping matrix $\mathbf{M}$ that globally maps values of $\psi\left(\mathbf{I}_{\mathrm{WB}^{(\mathrm{in})}\downarrow}\right)$ to the colors of our generated image $\hat{\mathbf{I}}_{\mathrm{WB}^{(i)}\downarrow}$, where $\psi(\cdot)$ is a polynomial kernel function that maps the image's RGB vectors to a higher 11-dimensional space. This mapping matrix $\mathbf{M}$ can be computed in a closed-form solution, as demonstrated in Chapters 6 and 7.

Once $\mathbf{M}$ is computed, we obtain our final result in the same input image resolution using the following equation:

$$\hat{\mathbf{I}}_{\mathrm{WB}^{(\mathrm{i})}} = \mathbf{M}\psi\left(\mathbf{I}_{\mathrm{WB}^{(\mathrm{in})}}\right). \tag{9.4}$$

Figure 9.3 illustrates our color mapping procedure. Our method requires ∼1.5 seconds on an Intel Xeon E5-1607 @ 3.10GHz machine with 32 GB RAM to process a 12-megapixel image for a selected WB setting.

We note that an alternative strategy is to compute the color polynomial mapping matrix directly [332]). We conducted preliminary experiments and found that estimating the polynomial matrix directly was less robust than generating the image itself followed by fitting a global polynomial function. The reason is that having small errors in the estimated polynomial coefficients can lead to noticeable color errors (e.g., out-of-gamut values), whereas small errors the estimated image were ameliorated by the global fitting.

Editing by User Manipulation Our framework allows the user to choose between generating result images with the three available WB settings (i.e., AWB, Shade WB, and Incandescent WB). Using the Shade and Incandescent WB, the user can edit the image to a specific WB setting in terms of color temperature, as explained in the following.

To produce the effect of a *new* target WB setting with a color temperature t that is not produced by our decoders, we can interpolate between our generated images with the

Incandescent and Shade WB settings. We found that a simple linear interpolation was sufficient for this purpose. This operation is described by the following equations:

$$\hat{\mathbf{I}}_{\mathrm{WB}^{(t)}} = b\,\hat{\mathbf{I}}_{\mathrm{WB}^{(\mathrm{T})}} + (1-b)\,\hat{\mathbf{I}}_{\mathrm{WB}^{(\mathrm{S})}}, \tag{9.5}$$

where $\hat{\mathbf{I}}_{\mathrm{WB}^{(\mathrm{T})}}$ and $\hat{\mathbf{I}}_{\mathrm{WB}^{(\mathrm{S})}}$ are our produced images with Incandescent and Shade WB settings, respectively, and b is the interpolation ratio that is given by $\frac{1/t-1/t(S)}{1/t(T)-1/t(S)}$. Figure 9.4 shows an example.

9.3 Results

Our method targets two different tasks: post-capture WB correction and manipulation of the sRGB rendered images to a specific WB color temperature. We achieve the state-of-the-art results for several different datasets for both tasks. We first describe the datasets used to evaluate our method in Sec. 9.3.1. We then discuss our quantitative and qualitative results in Sec. 9.3.2 and Sec. 9.3.3, respectively. We also perform an ablation study to validate our problem formulation and the proposed framework.

9.3.1 Datasets

As previously mentioned, we used Set 1 of our Rendered WB dataset (Chapter 6) for training and validation. For testing, we used three datasets not part of training or validation. Two of these additional datasets are as follows: (1) Set 2 of the Rendered WB dataset (2,881 images), and (2) the sRGB rendered version of the Cube+ dataset (10,242 images) [41] (see Chapter 8 for more details). Datasets (1) and (2) are used to evaluate the task of AWB correction. For the WB manipulation task, we used the rendered Cube+ dataset and (3) the rendered version of the MIT-Adobe FiveK dataset (29,980 images) [62]. The rendered version of the MIT-Adobe FiveK dataset was generated similarly to the ren-

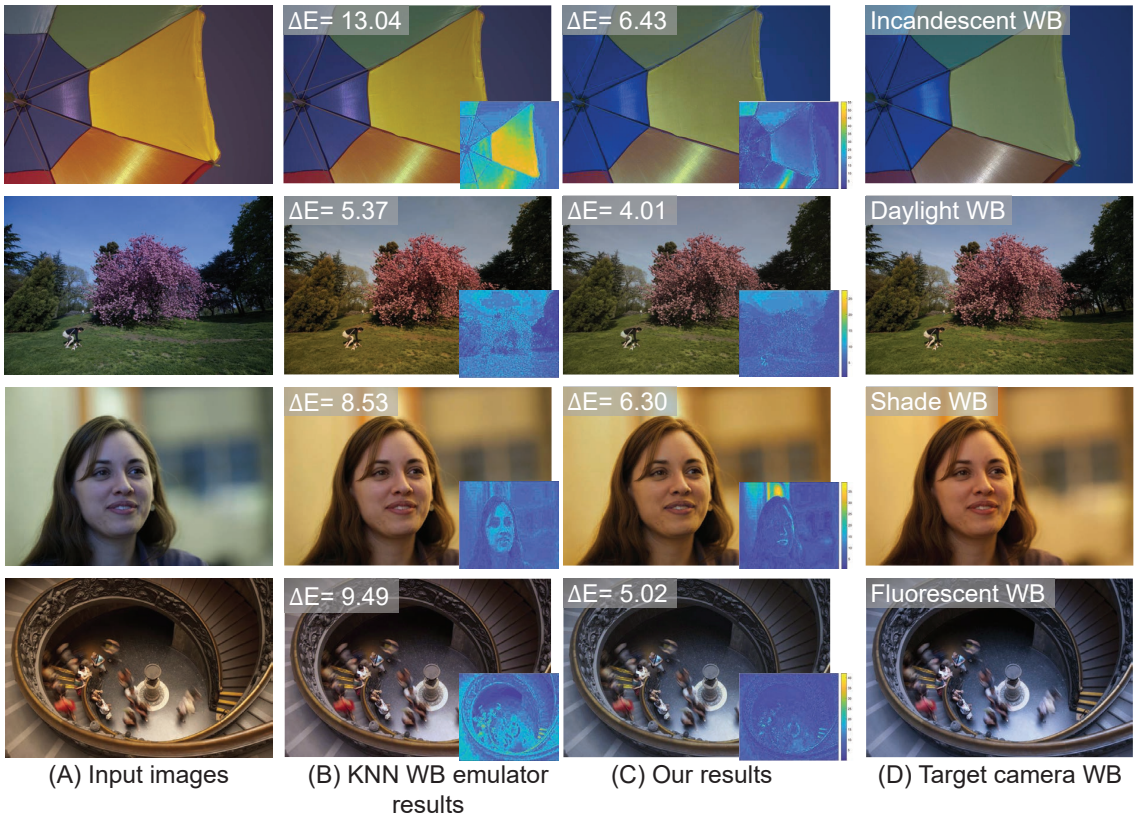


Figure 9.6: Qualitative comparison of WB manipulation. (A) Input images. (B) Results of KNN WB emulator (Chapter 7). (C) Our results. (D) Ground truth camera-rendered images with the target WB settings. In this figure, the target WB settings are Incandescent, Daylight, Shade, and Fluorescent. Shown input images are taken from the rendered version of the MIT-Adobe FiveK dataset.

dering process used to render the Rendered WB dataset (Chapter 6). Specifically, each raw-RGB image was rendered to the sRGB color space with different WB settings. This rendering process was performed in an accurate emulation of advanced camera rendering procedures.

Figure 9.7: Qualitative results of our method. (A) Input images. (B) AWB results. (C) Incandescent WB results. (D) Fluorescent WB results. (E) Shade WB Results. Shown input images are rendered from the MIT-Adobe FiveK dataset [62].

9.3.2 Quantitative Results

For both tasks, we follow the same evaluation metrics used Chapter 6. Specifically, we used the following metrics to evaluate our results: MSE, MAE, and $\triangle$E 2000 [336]. For each evaluation metric, we report the mean, lower quartile (Q1), median (Q2), and the upper quartile (Q3) of the error.

WB Correction We compared the proposed method with the KNN WB approach (Chapter 6). We also compared our results against the traditional WB diagonal-correction using recent illuminant estimation methods [51, 171]. We note that methods [51, 171] were not designed to correct nonlinear sRGB images. These methods are included, because it is often purported that such methods are effective when the sRGB image has been "linearized" using a decoding gamma.

Table 9.1 reports the error between corrected images obtained by each method and

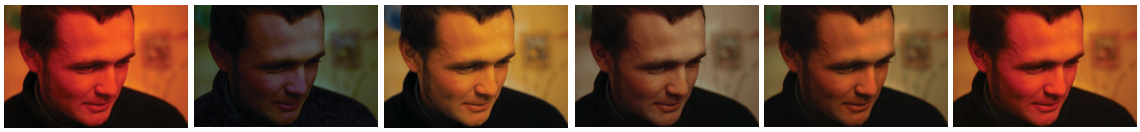


Figure 9.8: (A) Input image. (B) Result of quasi-U CC [51]. (C) Result of KNN WB (Chapter 6). (D)-(F) Our deep-WB editing results. Photo credit: *Duncan Yoyos* Flickr–CC BY-NC 2.0.

Figure 9.9: Strong color casts due to WB errors are hard to correct. (A) Input image rendered with an incorrect WB setting. (B) Result of Photoshop auto-color correction. (C) Result of Samsung S10 auto-WB correction. (D) Result of Google Photos auto-filter. (E) Result of iPhone 8 Plus built-in Photo app auto-correction. (F) Our AWB result using the proposed deep-WB editing framework. Photo credit: *OakleyOriginals* Flickr–CC BY 2.0.

the corresponding ground truth images. In Table 9.1, we show results on the validation set (i.e., Set 1 in the Rendered WB dataset) and three testing sets, for a total of 13,123 unseen sRGB images rendered with different camera models and WB settings. For the diagonal-correction results, we pre-processed each testing image by first applying the 2.2 gamma linearization [31, 101], and then we applied the gamma encoding after correction. As we can see, our method generalizes well for the testing sets, achieving state-of-the-art results compared to the other approaches in all evaluation metrics. We have on par results with the state-of-the-art method (Chapter 6) on the validation set.

WB Manipulation The goal of this task is to change the input image's colors to appear as they were rendered using a target WB setting. We compare our result with the work in Chapter 7 that proposed a KNN WB emulator that mimics WB effects in the sRGB space. We used the same WB settings produced by the KNN WB emulator. Specifically, we selected the following target WB settings: Incandescent ($2850K$), Fluorescent ($3800K$), Daylight ($5500K$), Cloudy ($6500K$), and Shade ($7500K$). As our decoders were trained to generate only Incandescent and Shade WB settings, we used Eq. 9.5 to produce the other WB settings (i.e., Fluorescent, Daylight, and Cloudy WB settings).

Table 9.2 shows the obtained results using our method and the KNN WB emulator. Table 9.2 demonstrates that our method outperforms the KNN WB emulator (Chapter 7) over a total of 40,222 testing images captured with different camera models and WB settings using all evaluation metrics.

9.3.3 Qualitative Results

In Fig. 9.5 and Fig. 9.6, we provide a visual comparison of our results against the most recent work proposed for WB correction [51] and our work in Chapters 6 and 7, respectively. On top of each example, we show the $\triangle$E 2000 error between the result image and the corresponding ground truth image (i.e., rendered by the camera using the target setting). It is clear that our results have the lower $\triangle$E 2000 and are the most similar to the ground truth images.

Figure 9.7 shows additional examples of our results. As shown, our framework accepts input images with arbitrary WB settings and re-renders them with the target WB settings, including the AWB correction.

We tested our method with several images taken from the Internet to check its ability to generalize to images typically found online. Figure 9.8 and Fig. 9.9 show examples. As

it is shown, our method produces compelling results compared with other methods and commercial software packages for photo editing, even when input images have strong color casts.

9.3.4 Comparison With a Vanilla U-Net

As explained earlier, our framework employs a single encoder to encode input images, while each decoder is responsible for producing a specific WB setting. Our architecture aims to model Eq. 9.1 in the same way cameras would produce colors for different WB settings from the same raw-RGB captured image.

Intuitively, we can re-implement our framework using a multi-U-Net architecture [323], such that each encoder/decoder model will be trained for a single target of the WB settings.

In Table 9.3, we provide a comparison between our proposed framework against vanilla U-Net models. We train our proposed architecture and three U-Net models (each U-Net model targets one of our WB settings) for 88,000 iterations. The results validate our design and make evident that our shared encoder not only reduces the required number of parameters but also gives better results.

9.4 Summary

We have presented a deep learning framework for editing the WB of sRGB camera-rendered images. Specifically, we have proposed a DNN architecture that uses a single encoder and multiple decoders, which are trained in an end-to-end manner. Our framework allows the direct correction of images captured with wrong WB settings. Additionally, our framework produces output images that allow users to manually adjust the sRGB image to appear as if it was rendered with a wide range of WB color temperatures. Quantitative and qualitative results demonstrate the effectiveness of our framework against recent data-driven methods.

Table 9.1: AWB results using our Rendered WB dataset (Chapter 6) and the rendered version of the Cube+ dataset [41]. We report the mean, first, second (median), and third quartile (Q1, Q2, and Q3) of MSE, MAE, and $\triangle\mathbf{E}$ 2000 [336]. For all diagonal-based methods, gamma linearization [31, 101] is applied. The top results are indicated with yellow and boldface.

Method	MSE				MAE				△E 2000			
	Mean	Q1	Q2	Q3	Mean	Q1	Q2	Q3	Mean	Q1	Q2	Q3
Rendered WB dataset: Set 1 (21,046 validation images)												
FC4 [171]	179.55	33.89	100.09	246.50	6.14°	2.62°	4.73°	8.40°	6.55	3.54	5.90	8.94
Quasi-U CC [51]	172.43	33.53	97.9	237.26	6.00°	2.79°	4.85°	8.15°	6.04	3.24	5.27	8.11
KNN WB (Chapter 6)	**77.79**	13.74	**39.62**	**94.01**	**3.06°**	**1.74°**	**2.54°**	**3.76°**	**3.58**	**2.07**	**3.09**	**4.55**
Ours	82.55	**13.19**	42.77	102.09	3.12°	1.88°	2.70°	3.84°	3.77	2.16	3.30	4.86
Rendered WB dataset: Set 2 (2,881 images)												
FC4 [171]	505.30	142.46	307.77	635.35	10.37°	5.31°	9.26°	14.15°	10.82	7.39	10.64	13.77
Quasi-U CC [51]	553.54	146.85	332.42	717.61	10.47°	5.94°	9.42°	14.04°	10.66	7.03	10.52	13.94
KNN WB (Chapter 6)	171.09	37.04	87.04	190.88	4.48°	2.26°	3.64°	5.95°	5.60	3.43	4.90	7.06
Ours	**124.97**	**30.13**	**76.32**	**154.44**	**3.75°**	**2.02°**	**3.08°**	**4.72°**	**4.90**	**3.13**	**4.35**	**6.08**
Rendered Cube+ dataset with different WB settings (10,242 images)												
FC4 [171]	371.9	79.15	213.41	467.33	6.49°	3.34°	5.59°	8.59°	10.38	6.6	9.76	13.26
Quasi-U CC [51]	292.18	15.57	55.41	261.58	6.12°	1.95°	3.88°	8.83°	7.25	2.89	5.21	10.37
KNN WB (Chapter 6)	194.98	27.43	57.08	118.21	4.12°	1.96°	3.17°	5.04°	5.68	3.22	4.61	6.70
Ours	**80.46**	**15.43**	**33.88**	**74.42**	**3.45°**	**1.87°**	**2.82°**	**4.26°**	**4.59**	**2.68**	**3.81**	**5.53**

Table 9.2: Results of WB manipulation using the rendered version of the Cube+ dataset [41] and the rendered version of the MIT-Adobe FiveK dataset [62]. We report the mean, first, second (median), and third quartile (Q1, Q2, and Q3) of MSE, MAE, and $\triangle$**E** 2000 [336]. The top results are indicated with yellow and boldface.

Method	**MSE**				**MAE**				**△E 2000**			
	Mean	**Q1**	**Q2**	**Q3**	**Mean**	**Q1**	**Q2**	**Q3**	**Mean**	**Q1**	**Q2**	**Q3**
Rendered Cube+ dataset (10,242 images)												
KNN WB emulator (Chapter 7)	317.25	50.47	153.33	428.32	7.6°	3.56°	6.15°	10.63°	7.86	4.00	6.56	10.46
Ours	**199.38**	**32.30**	**63.34**	**142.76**	**5.40°**	**2.67°**	**4.04°**	**6.36°**	**5.98**	**3.44**	**4.78**	**7.29**
Rendered MIT-Adobe FiveK dataset (29,980 images)												
KNN WB emulator (Chapter 7)	249.95	41.79	109.69	283.42	7.46°	3.71°	6.09°	9.92°	6.83	3.80	5.76	8.89
Ours	**135.71**	**31.21**	**68.63**	**151.49**	**5.41°**	**2.96°**	**4.45°**	**6.83°**	**5.24**	**3.32**	**4.57**	**6.41**

Table 9.3: Average of mean square error and $\triangle$**E** 2000 [336] obtained by our framework and the traditional U-Net architecture [323]. Shown results on Set 2 of our Rendered WB dataset (Chapter 6) for AWB and the rendered version of the Cube+ dataset [41] for WB manipulation (see Chapter 8 for more details regarding this rendered version of the Cube+ dataset). The top results are indicated with yellow and boldface.

Method	**AWB**		**WB editing**	
	MSE	**△E 2000**	**MSE**	**△E 2000**
Multi-U-Net [323]	187.25	6.23	234.77	6.87
Ours	**124.47**	**4.99**	**206.81**	**6.23**

10 Color Temperature Tuning

In this chapter, we propose an imaging framework[1] that renders a small number of "tiny versions" of the original image (e.g., 0.1% of the full-size image), each with different WB color temperatures. Rendering these tiny images requires minimal overhead from the camera pipeline. These tiny images are sufficient to allow color mapping functions to be computed that can map the full-sized sRGB image to appear as if it was rendered with any of the tiny images' color temperatures. Moreover, by blending the color mapping functions, we can map the output sRGB image to appear as if it was rendered through the full pipeline with *any color temperature*. These mapping functions can be stored as a JPEG comment with less than 6 KB overhead. We demonstrate that this capture framework can significantly outperform any existing solution targeting post-capture WB editing. The source code of this work is available on GitHub: https://github.com/mahmoudnafifi/ColorTempTuning.

10.1 Introduction

As discussed in previous chapters, WB is applied to the raw-RGB sensor image and aims to remove the color cast due to the scene's illumination, which is often described by its

[1]This work was published in [22]: Mahmoud Afifi, Abhijith Punnappurath, Abdelrahman Abdelhamed, Hakki Can Karaimer, Abdullah Abuolaim, and Michael S. Brown. Color Temperature Tuning: Allowing Accurate Post-Capture White-Balance Editing. In Color and Imaging Conference, 2019.

correlated color temperature. An image's WB temperature can either be specified by a manual setting (e.g., Tungsten, Daylight), or be estimated from the image using the camera's AWB function.

After the WB step, the camera pipeline applies several nonlinear camera-specific photo-finishing operations to convert the image from the raw-RGB color space to sRGB. These nonlinear operations make it very challenging to modify the WB post-capture. This is particularly troublesome if the WB setting was incorrect, resulting in the captured image having an undesirable color cast. Existing methods for post-capture WB manipulation attempt to reverse the camera pipeline and map the sRGB colors back to raw-RGB. Not only does this process necessitate careful camera calibration, but also it requires reapplying the pipeline to get back to the sRGB space after modifying the WB in the raw-RGB space.

In our work in Chapter 6 and Chapter 8, we proposes to white balance improperly white-balanced sRGB-rendered images by estimating polynomial mapping functions from a large set of training data. Our work in this chapter is close to the work discussed in Chapters 6 and 8 in the sense that we also use a set of polynomial mapping functions to manipulate the WB of sRGB-rendered images. In contrast, our work here embeds the required mapping functions within the final rendered images during the rendering process.

Contributions We advocate an image capture framework to enable accurate post-capture WB manipulation *directly* in the sRGB space without having to revert to the raw-RGB space. Our proposed approach is to create a tiny downsampled version of the raw-RGB image and render it through the camera pipeline multiple times using a set of pre-selected WB settings. We can then use these tiny sRGB images to compute nonlinear color mapping functions that can transform the full-sized sRGB output image to appear as if it was rendered through the pipeline with the color temperature of any of the tiny images. More

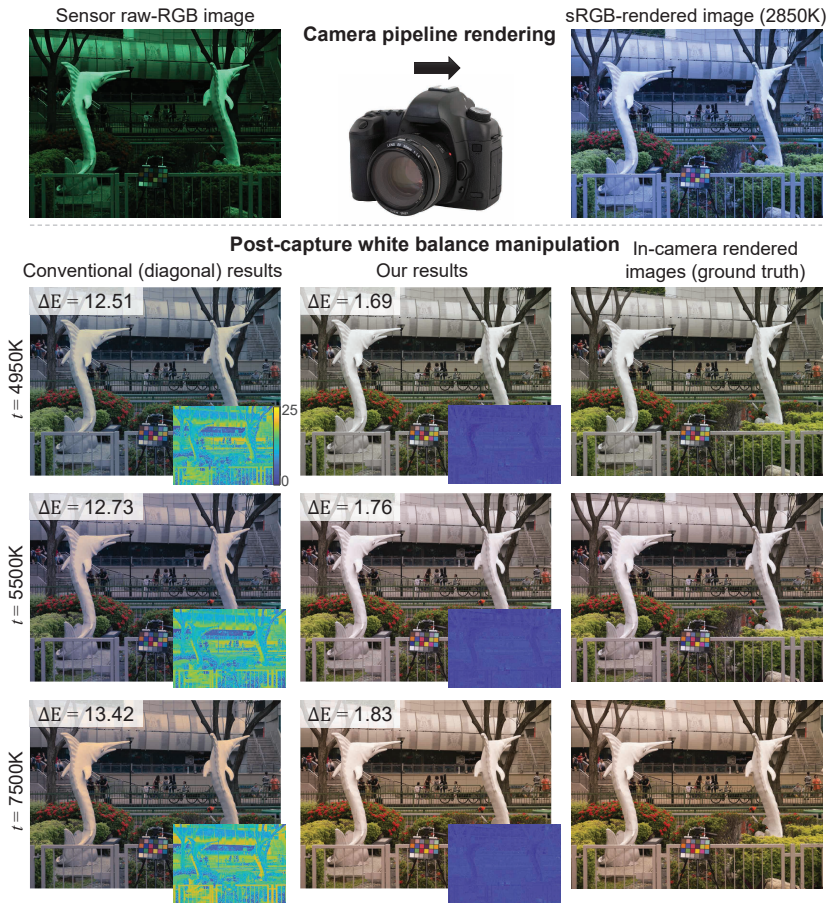


Figure 10.1: An sRGB image rendered using our imaging framework with the *Tungsten* WB setting (i.e., 2850K). Using data embedded within the rendered image, our method can modify this sRGB image's WB post-capture to *any* target WB color temperature (e.g., 4950K, 5500K, 7500K) producing results (column 2) almost identical to what the actual camera pipeline would have generated (column 3). Error maps (△E) insets, and average (△E) demonstrate that our method is far superior to the conventional post-capture WB manipulation (column 1).

importantly, blending these mapping functions provides the ability to navigate the full parameter space of WB settings—that is, we can produce a full-sized sRGB output image

with *any* color temperature that is almost identical to what the camera pipeline would have produced. Figure 10.1 shows an example. The mapping functions themselves can be efficiently computed and easily stored in the sRGB-rendered image (e.g., as a JPEG comment with less than 6 KB overhead).

10.2 Methodology

Our proposed method is based on the computation of a set of nonlinear color mapping functions between a set of tiny downsampled camera-rendered sRGB images. We first discuss, in Sec. 10.2.1, how these tiny images are rendered. Sec. 10.2.2 describes how to compute the mapping functions. Finally, in Sec. 10.2.3, we elaborate on how these mappings enable us to explore the parameter space of WB settings, and allow for accurate post-capture WB manipulation. An overview of the image framework is shown in Fig. 10.2-(A).

In this chapter, we represent each image as a $3 \times P$ matrix that contains the image's RGB triplets, where P is the total number of pixels in the image.

10.2.1 Rendering Tiny Images

The first step of our imaging framework is to create a tiny downsampled copy of the raw-RGB image $\mathbf{I}$. The tiny version is denoted as $\mathbf{X}$ and, in our experiments, is only 150×150 pixels, as compared to, say, a 20-megapixel full-sized image $\mathbf{I}$. Our tiny raw-RGB image $\mathbf{X}$, which is approximately 0.1% of the full-size image, can be stored in memory easily. The full-sized raw-RGB image $\mathbf{I}$ is first rendered through the camera pipeline with some WB color temperature to produce the full-sized sRGB output image $\mathbf{O}$. This color temperature is either obtained from a manually selected WB setting or estimated by the camera's AWB. The tiny image $\mathbf{X}$ is then processed by the camera pipeline N times, each time with a

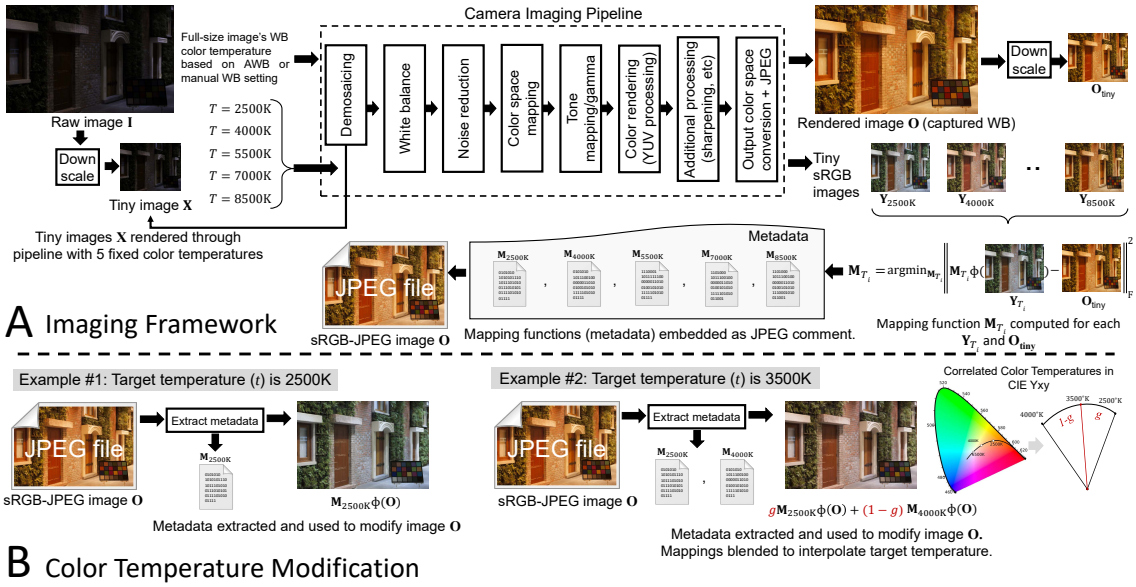


Figure 10.2: (A) Our proposed image capture framework. The raw-RGB image $\mathbf{I}$ is downsampled to produce the tiny raw-RGB image $\mathbf{X}$. Image $\mathbf{X}$ is then rendered through the pipeline N times, each time with a different pre-selected WB preset T_i applied, to produce the tiny sRGB-rendered images $\{\mathbf{Y}_{T_i}\}_{i=1}^{N}$. Mapping functions $\{\mathbf{M}_{T_i}\}_{i=1}^{N}$ are computed from these tiny images $\{\mathbf{Y}_{T_i}\}_{i=1}^{N}$ and a downsampled version of the sRGB output image $\mathbf{O}$ denoted as $\mathbf{O}_{\text{tiny}}$. The mappings $\{\mathbf{M}_{T_i}\}_{i=1}^{N}$ are stored inside the JPEG comment field of the sRGB output image $\mathbf{O}$. (B) Using the metadata to modify an sRGB image. Example 1 shows a case where the sRGB image $\mathbf{O}$ is mapped to one of the color temperatures in the metadata. The corresponding color mapping function can be extracted and used to modify the image. Example 2 shows an example where the target color temperature is not in the metadata. In this case, the target temperature mapping function is interpolated by blending between the two closest mapping functions in the metadata.

different WB color temperature setting $\{T_i\}_{i=1}^{N}$. These settings $\{T_i\}_{i=1}^{N}$ can correspond to the camera's preset WB options, such as Tungsten, Daylight, Fluorescent, Cloudy, and

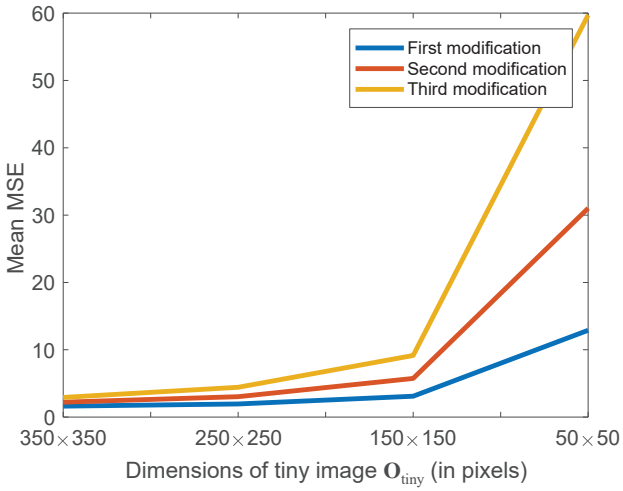


Figure 10.3: The effect of different downsampling sizes on our results. In this figure, we also show the mean squared error (MSE) between our results and the target in-camera ground truth images after updating our metadata for three successive WB modifications with different target color temperatures. The details of the update process are discussed in Sec. 10.2.3.

Shade, or their color temperature values, such as 2850K, 3800K, 5500K, 6500K, and 7500K, or any other chosen set of color temperatures. The resulting tiny sRGB images processed by the camera pipeline are denoted as $\{\mathbf{Y}_{T_i}\}_{i=1}^{N}$ corresponding to the WB settings $\{T_i\}_{i=1}^{N}$.

10.2.2 Mapping Functions

The full-sized sRGB output image $\mathbf{O}$ is downsampled to have the same dimensions as $\{\mathbf{Y}_{T_i}\}_{i=1}^{N}$ and is denoted as $\mathbf{O}_{\text{tiny}}$. For each tiny image $\{\mathbf{Y}_{T_i}\}_{i=1}^{N}$, we compute a nonlinear mapping function $\mathbf{M}_{T_i}$, which maps $\mathbf{O}_{\text{tiny}}$ to $\mathbf{Y}_{T_i}$, by solving the following minimization problem:

$$\arg\min_{\mathbf{M}_{T_i}} \| \mathbf{M}_{T_i} \, \Phi\left(\mathbf{O}_{\text{tiny}}\right) - \mathbf{Y}_{T_i} \|_{\text{F}}^{2}, \tag{10.1}$$

where $\Phi : \mathbb{R}^3 \rightarrow \mathbb{R}^u$ is a kernel function that transforms RGB triplets to a u-dimensional space, where $u > 3$, and $\| \, . \, \|_{\text{F}}$ denotes the Frobenius norm. For each image $\mathbf{Y}_{T_i}$, this equation finds an $\mathbf{M}_{T_i}$ that minimizes the errors between the RGB colors in the downsampled image $\mathbf{O}_{\text{tiny}}$ and its corresponding image $\mathbf{Y}_{T_i}$.

In general, relying on kernel functions based on high-degree polynomials can hinder generalization; however, in our case, the mapping function is computed specifically for a pair of images. Hence, a kernel function with a higher degree is preferable. In our experiments, we adopted a polynomial kernel function given in [118], where $\Phi : \mathbb{R}^3 \rightarrow \mathbb{R}^{34}$. Hence, each mapping function is represented by a 3×34 matrix. Once the mapping functions are computed, the set of downsampled images $\mathbf{X}$, $\{\mathbf{Y}_{T_i}\}_{i=1}^{N}$, and $\mathbf{O}_{\text{tiny}}$ is no longer needed and can be discarded.

For all of our experiments in this chapter, we rendered tiny images using five (5) color temperature values, 2500K, 4000K, 5500K, 7000K, and 8500K, and computed the corresponding mapping functions $\mathbf{M}_{T_i}$. The five functions require less than 6 KB of metadata to represent and can be saved inside the final JPEG image $\mathbf{O}$ as a comment field.

10.2.3 Color Temperature Manipulation

Once the mapping functions have been computed, we can use them to post-process the sRGB output image $\mathbf{O}$ to appear as if it was rendered through the camera pipeline with any of the WB settings $\{T_i\}_{i=1}^{N}$. This process can be described using the following equation:

$$\mathbf{O}_{\text{modified}} = \mathbf{M}_{T_i} \; \Phi \left(\mathbf{O} \right), \tag{10.2}$$

where $\mathbf{O}_{\text{modified}}$ is the full-resolution sRGB image as if it was "re-rendered" with the WB setting T_i. This is demonstrated in Example 1 of Fig. 10.2-(B).

More importantly, by blending between the mapping functions, we can post-process the sRGB output image $\mathbf{O}$ to appear as if it was processed by the camera pipeline using

any color temperature value, and not just the settings $\{T_i\}_{i=1}^{N}$ the tiny images $\{\mathbf{Y}_{T_i}\}_{i=1}^{N}$ were rendered out with.

Given a new target WB setting with a color temperature t, we can interpolate between the nearest pre-computed mapping functions to generate a new mapping function for t as follows:

$$\mathbf{M}_t = g\mathbf{M}_a \ + (1-g)\mathbf{M}_b, \tag{10.3}$$

$$g = \frac{1/t - 1/b}{1/a - 1/b}, \tag{10.4}$$

where $a,\ b \in \{T_i\}_{i=1}^{N}$ are the nearest pre-computed color temperatures to t, such that $a < t < b$, and $\mathbf{M}_a$ and $\mathbf{M}_b$ are the corresponding mapping functions computed for temperatures a and b, respectively. The final modified image $\mathbf{O}_{\text{modified}}$ is generated by using $\mathbf{M}_t$ instead of $\mathbf{M}_{T_i}$ in Eq. 10.2. Example 2 of Fig. 10.2-(B) demonstrates this process.

Recomputing the Mapping Functions

Our metadata was computed for the original sRGB-rendered image $\mathbf{O}$. After $\mathbf{O}$ has been modified to $\mathbf{O}_{\text{modified}}$, this metadata has to be updated to facilitate future modifications of $\mathbf{O}_{\text{modified}}$. The update is performed so as to map $\mathbf{O}_{\text{modified}}$, with color temperature t, to our preset color temperatures $\{T_i\}_{i=1}^{N}$. To that end, each pre-computed mapping function $\{\mathbf{M}_{T_i}\}_{i=1}^{N}$ is updated based on the newly generated image $\mathbf{O}_{\text{modified}}$ as follows:

$$\underset{\mathbf{M}_{T_i}}{\arg\min} \parallel \mathbf{M}_{T_i}\Phi\left(\mathbf{M}_t\Phi\left(\mathbf{O}_{\text{tiny}}\right)\right) - \mathbf{M}_{T_i(\text{old})}\Phi\left(\mathbf{O}_{\text{tiny}}\right) \parallel_{\text{F}}^{2}, \tag{10.5}$$

where $\mathbf{M}_{T_i}$ is the new mapping function and $\mathbf{M}_{T_i(\text{old})}$ is the old mapping function for the i^{th} WB setting before the current WB modification.

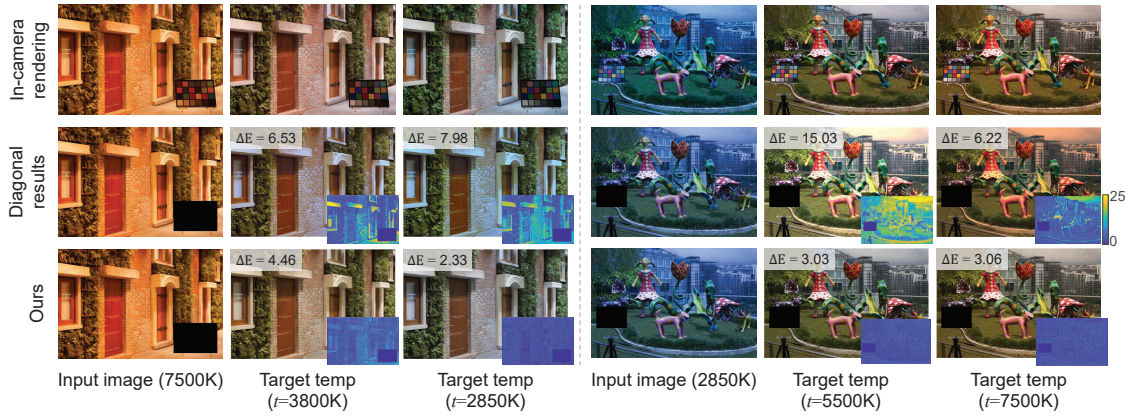


Figure 10.4: First row: in-camera sRGB images rendered with different color temperatures. Second row: results obtained by diagonal manipulation using the *exact* achromatic patch from the color chart. Third row: our results. ΔE error of each result is reported and shown as an error map.

10.3 Results

We evaluated our proposed method on six cameras from the NUS dataset [77]. Images in this dataset were captured using digital single-lens reflex (DSLR) cameras with a color chart placed in the scene. All images in the dataset have been saved in the raw-RGB format, and so we can convert them to sRGB format using the conventional in-camera pipeline [191]. Specifically, we rendered out sRGB images with five different color temperature values, which are: 2850K, 3800K, 5500K, 6500K, and 7500K, corresponding approximately to the common WB presets available on most cameras—namely, Tungsten, Fluorescent, Daylight, Cloudy, and Shade. For our chosen six cameras totaling 1,340 images, this process yields $1340 \times 5 = 6700$ sRGB-rendered images. These images can be considered as ground truth since these are produced by emulating the in-camera pipeline. See Fig. 10.1, for example. We also render the same images using our proposed framework with the color chart masked

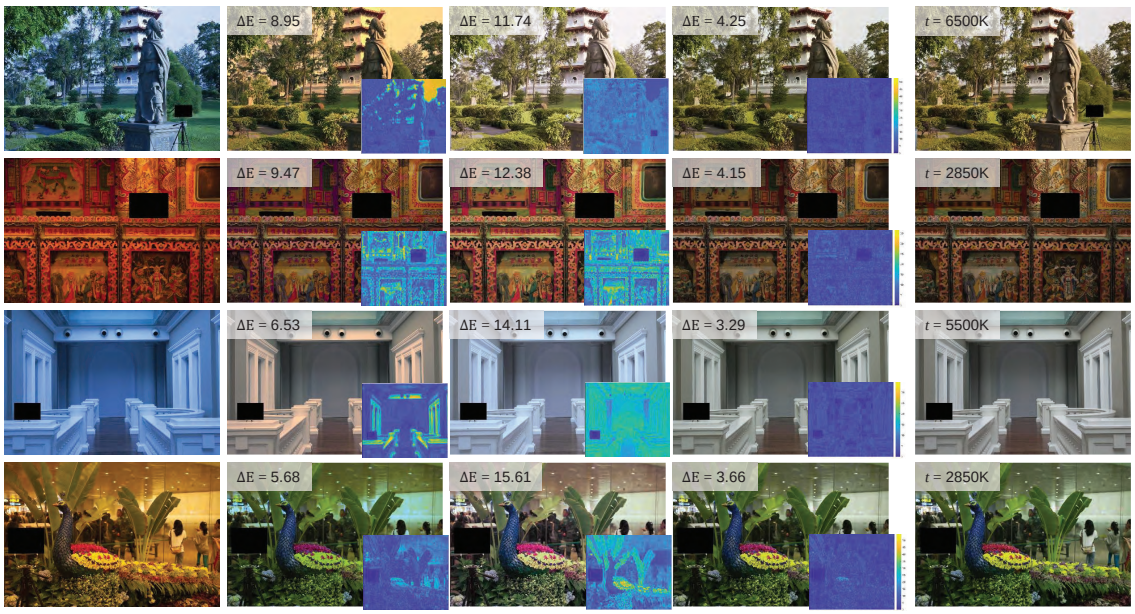


Figure 10.5: (A) Input sRGB-rendered image. (B) Diagonal manipulation result. (C) Adobe Lightroom result. (D) Our result. (E) In-camera sRGB image rendered with the target color temperature t. The average ΔE error is shown for each image.

out. During this rendering process, mapping functions corresponding to our pre-selected color temperature values (i.e., 2500K, 4000K, 5500K, 7000K, and 8500K) are computed and stored as metadata corresponding to each image.

For each image generated with a particular WB preset, we choose the other four WB settings as the target WB values. Following the procedure described in Sec. 10.2.3, we then process the input image using our pre-computed mapping functions to appear as though it was rendered through the camera pipeline with the four target WB settings. For example, given an input image originally rendered out with the WB Tungsten preset, we generate four WB modified versions with the following WB settings: Fluorescent, Daylight, Cloudy, and Shade. In this manner, we generate $6700 \times 4 = 26800$ WB modified images using

Table 10.1: Quantitative results on the NUS dataset [77]. We compare our results against diagonal WB manipulation, denoted as Diag, using an *exact achromatic* reference point obtained from the color chart in the scene. The diagonal manipulation is applied directly on the sRGB images, and on the "linearized" sRGB [31, 101]. The terms Q1, Q2, and Q3 denote the first, second (median), and third quartile, respectively. The terms MSE and MAE stand for mean square error and mean angular error, respectively. The best results are indicated in boldface and highlighted in yellow.

Method	MSE				MAE				Δ E 2000				Δ E 76			
	Mean	Q1	Q2	Q3	Mean	Q1	Q2	Q3	Mean	Q1	Q2	Q3	Mean	Q1	Q2	Q3
Diag WB	1196.93	455.21	825.38	1441.58	5.75	2.24	4.85	8.03	8.98	4.88	7.99	11.73	13.53	7.18	11.84	17.98
Diag WB w linearization	1160.16	428.18	771.78	1400.49	5.49	2.19	4.56	7.61	8.61	4.62	7.58	11.09	12.87	6.82	11.22	16.99
Ours	**75.58**	**35.51**	**61.05**	**98.68**	**2.04**	**1.42**	**1.86**	**2.45**	**3.09**	**2.33**	**3.00**	**3.74**	**4.39**	**3.21**	**4.26**	**5.30**

our proposed approach. These modified images can be compared with their corresponding ground truth images.

We adopt four commonly used error metrics for quantitative evaluation: (i) MSE, (ii) MAE, (iii) ΔE 2000 [336], and (iv) ΔE 76. In Table 10.1, the mean, lower quartile (Q1), median (Q2), and the upper quartile (Q3) of the error between the WB modified images and the corresponding ground truth images are reported. It can be observed that our method consistently outperforms the diagonal manipulation, both with and without the commonly used pre-linearization step using the 2.2 inverse gamma [101], in all metrics.

Figure 10.3 shows the effect of different downsampling sizes on the post-processing WB modification compared with rendering the original raw-RGB image with each target color temperature (i.e., ground truth). In this example, we randomly selected 300 raw-RGB images from the NUS dataset. Each image is rendered using our method, and then

modified to different target WB settings. This process was repeated three times to study the error propagation of our post-processing modifications.

Representative qualitative results from the NUS dataset are shown in Figs. 10.4 and 10.5. As can be observed, our method produces better results compared to the sRGB diagonal WB manipulation and Adobe Lightroom.

10.4 Summary

We have described an imaging framework that enables users to accurately modify the WB color temperature of an sRGB-rendered image. Such functionality is currently not possible with the conventional imaging pipeline. With our method, images typically discarded due to the camera having the wrong WB setting, can be easily fixed by changing to the correct color temperature. In addition, our approach enables editing for aesthetic manipulation of the image's appearance. Our approach requires a minor modification to the existing imaging pipeline and produces metadata that can be easily stored in an image file (e.g. JPEG) with only 6 KB of overhead.

Part V

Color Enhancement

11 Color Enhancement Through the CIE XYZ Space

Previous chapters (Chapters 4–10) discussed methods for computational CC and image white balancing in sensor-raw and sRGB color spaces. This part of the thesis focuses on other degradation factors that can effect the colors of captured photographs. Specifically, we focus on low-light images that either are captured using under-exposure settings or have scenes with low-lighting conditions. We further discuss a less explored area of research, where we propose a method to correct colors of over-exposed images in the next chapter.

The work in this chapter treats the problem of enhancing low-light captured images from a different angle, where we employ a scene-referred image state of the cameras ISP. Cameras currently allow access to two image states: (i) a minimally processed linear raw-RGB image state (i.e., raw sensor data) or (ii) a highly-processed nonlinear image state (e.g., sRGB). There are many computer vision tasks that work best with a linear image state, such as image dehazing and deblurring. Unfortunately, the vast majority of images are saved in the nonlinear image state. Because of this, a number of methods have been proposed to "unprocess" nonlinear images back to a raw-RGB state. However, existing unprocessing methods have a drawback because raw-RGB images are sensor-specific. As a result, it is necessary to know which camera produced the sRGB output and use a method or network tailored for that sensor to properly unprocess it. This chapter addresses this

limitation by exploiting another camera image state that is not available as an output, but it is available inside the camera pipeline[1]. In particular, cameras apply a colorimetric conversion step to convert the raw-RGB image to a device-independent space based on the CIE XYZ color space before they apply the nonlinear photo-finishing. Leveraging this canonical image state, we propose a deep learning framework, CIE XYZ Net, that can unprocess a nonlinear image back to the canonical CIE XYZ image. This image can then be processed by any low-level computer vision operator and re-rendered back to the nonlinear image. We demonstrate the usefulness of the CIE XYZ Net on enhancing low-light images and show significant gains that can be obtained by this processing framework. The source code and dataset of this work are available on GitHub: Code and dataset are publicly available at https://github.com/mahmoudnafifi/CIE_XYZ_NET.

11.1 Introduction

As discussed earlier, an a camera's ISP hardware processes the initial captured sensor image in a pipeline fashion, with routines being applied one after the other. The ISP used by consumer cameras performs operations as two distinct stages. First, a "front-end" stage applies linear operations, such as white balance and color adaptation, to convert the sensor-specific raw-RGB image to a device-independent color space (e.g., CIE XYZ or its wide-gamut representation, ProPhoto) [221]. The image states associated with the front-end process are called a *scene-referred* image because the image remains related directly to initial recorded sensor values related to the physical scene. Next, a "photo-finishing"

[1]This work is under review in IEEE Transactions on Pattern Analysis and Machine Intelligence. A preprint version is available in [11]: Mahmoud Afifi, Abdelrahman Abdelhamed, Abdullah Abuolaim, Abhijith Punnappurath, Michael S. Brown. CIE XYZ Net: Unprocessing Images for Low-Level Computer Vision Tasks. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2021 – to appear.

Figure 11.1: We propose a cycle framework that can unprocess sRGB images back to the linear CIE XYZ color space and re-render the CIE XYZ images into the nonlinear sRGB color space. (A) The input camera-rendered sRGB image. (B) Our image decomposition (left: residual photo-finishing layer, right: scene-referred CIE XYZ reconstruction). (C) The ground-truth scene-referred CIE XYZ image. (D) Our re-rendering result from the reconstructed CIE XYZ image. To aid visualization, CIE XYZ images are scaled by a factor of two. Input image is taken from the MIT-Adobe FiveK dataset [62].

stage is performed that applies nonlinear steps and local operators to produce a visually pleasing photograph. For example, selective color manipulation is often applied to enhance skin tone or make the overall colors more vivid, while local tone manipulation increases local contrast within the image. After the photo-finishing stage, the image is encoded in an output color space (e.g., sRGB, AdobeRGB, or Display P3). The image states associated with the photo-finishing process are referred to as *display-referred* as they are encoded for visual display. Cameras currently allow access only to either the minimally processed scene-referred image state (i.e., raw-RGB image) or the final display-referred image state (e.g., sRGB, AdobeRGB, or Display P3). Unfortunately, these two image states are not ideal for low-level computer vision tasks.

The raw-RGB image state preserves the linear relationship of incident scene radiance. This linear image formation makes raw-RGB images suitable for a wide range of low-level computer vision tasks, such as image deblurring, image dehazing, image denoising, and various types of image enhancement [59, 285, 353, 397]. However, the drawback of raw-RGB is that the physical color filter arrays that make up the sensor's Bayer pattern are sensor-specific. This means raw-RGB values captured of the same scene but with different sensors are significantly different [283]. This often requires learning-based methods to be trained per sensor or camera make and model (e.g., [13, 59, 95, 171, 280]).

The more common display-referred image state (in this chapter, assumed to be in the sRGB color space) also has drawbacks. While this image state is the most widely used and is suitable for display, cameras apply their own proprietary photo-finishing to enhance the visual quality of the image. This means images captured of the same scene but using different camera models (and sometimes the same camera but with different settings) will produce images that have significantly different sRGB values [191, 202, 285].

As previously discussed, the front-end processor of a typical camera ISP performs a colorimetric conversion to map the raw-RGB image to a standard perceptual colorspace—namely, CIE 1931 XYZ [191]. While there exists no formal image encoding for this image state, it is possible to convert existing raw-RGB images stored in digital negative (DNG) format to this intermediate state by applying a software camera ISP (e.g., [7, 191]). This provides a mechanism to standardize all images into a canonical linear scene-referred image state and is the impetus of our work.

Contribution We propose a method to decompose non-linear sRGB images into two parts: 1) a canonical linear scene-referred image state in the CIE XYZ color space and 2) a residual image layer that resembles additional non-linear and local photo-finishing

operations. Through such decomposition strategy, we learn a model that can accurately map back and forth between non-linear sRGB and linear CIE XYZ images. An example is shown in Fig. 11.1. Unlike raw-RGB, the CIE XYZ color space is *device-independent*, and as a result, helps with model generalization. Furthermore, CIE XYZ images can be encoded as standard three-channel images that can be easily handled by existing computer vision frameworks. We show that our proposed model maps images back to the CIE XYZ color space more accurately compared to alternative approaches. In addition, we perform experiments on low-light image enhancement to show that employing our proposed CIE XYZ model provides the performance boost anticipated from using linear images (additional computer vision applications of our CIE XYZ model are provided in Appendix A).

11.2 Related Work

In this section, we review various methods proposed for linearization of camera-rendered images. More details about camera imaging pipeline and linearization methods are provided in Chapter 2.

11.2.1 Camera-Rendered Image Linearization

To obtain a linear image from its camera-rendered version, we need to reverse the nonlinear camera-rendering stage in the pipeline. Many methods have been proposed to model a parametric relationship that maps from the camera-rendered image (i.e., sRGB image) back to its raw-RGB version (e.g., [285]). However, raw-RGB space is camera-dependent and requires having a separate model per camera. As discussed in Chapter 2, other approaches involve simple linearization by inverting the global tone mapping and the gamma compression followed by applying a linearization matrix to obtain a linear sRGB or CIE

XYZ image [59]. Such approaches are too simple and do not account for the local processing or dynamic range adjustments. Unlike prior approaches, instead of trying only to obtain a linear image, our approach is to decompose the nonlinear image into globally processed and locally processed layers. The locally processed layer represents local color processing, such as local tone mapping. Then, we learn a global mapping from the globally processed image to the linear image. Another line of research targeting the problem of image linearization is radiometric calibration [69, 238] (see as Chapter 2 for more details). Unlike our approach, radiometric calibration methods do not target a specific, well-defined color space, and do not address the problem of local processing.

11.3 Methodology

This section describes our overall framework, including network architecture, dataset generation, and training details.

11.3.1 Formulation

Inside a camera imaging pipeline, a raw-RGB image $\mathbf{x}_{\text{raw}} \in \mathbb{R}^{h \times w}$ undergoes a sequence of processing stages to be transformed to the final output sRGB image $\mathbf{x}_{\text{srgb}} \in \mathbb{R}^{h \times w \times 3}$, where h and w represent the image height and width, respectively.

As mentioned earlier, the raw-RGB image $\mathbf{x}_{\text{raw}}$ is in a camera-dependent color space that is linear with respect to scene light irradiance falling on the sensor. One of the early steps in the camera processing pipeline is to convert the camera-dependent color space to a device-independent color space—namely, CIE XYZ. Based on this observation, instead of modeling the whole pipeline back to the raw-RGB image, we choose to model an intermediate representation of the image in the CIE XYZ color space $\mathbf{x}_{\text{xyz}} \in \mathbb{R}^{h \times w \times 3}$ that is still linear with respect to scene irradiance, but is in a canonical color space. We

are interested in the on-camera rendering procedures that map the CIE XYZ images into the final display-referred (i.e., photo-finished) sRGB color space. This operation can be described as

$$\mathbf{x}_{\text{srgb}} = \mathcal{F}(\mathbf{x}_{\text{xyz}}). \tag{11.1}$$

In our method, instead of relying on a single function to model the pipeline stages between sRGB and CIE XYZ, we decompose this mapping into two parts: 1) *global processing*, denoted collectively as $\mathcal{F}_{\text{glob}}(\cdot)$, that is globally applied to all image pixels and 2) *local processing*, denoted collectively as $\mathcal{F}_{\text{loc}}(\cdot)$, that represents local photo-finishing operations, such as local tone mapping and selective color adjustments.

Such design is largely motivated by the fact that actual camera ISPs perform both global and local image processing. Global processing can be easily modeled by a polynomial color mapping, and hence, we train a CNN to estimate such polynomial function coefficients. Local processing is more challenging to model, and hence, we chose to model it as a residual fully-convolutional neural network. Breaking the process into two parts, global and local, has the added advantage that it enables image enhancement methods to selectively manipulate either part independently, and improves the performance of various photo-finishing tasks, as we will demonstrate in our experiments.

Our forward pipeline from $\mathbf{x}_{\text{xyz}}$ to $\mathbf{x}_{\text{srgb}}$ can be represented as a cascade of the global and the local processes. The global processing stage is represented as

$$\mathbf{M}_{\text{fwd}} = \mathcal{F}_{\text{glob}}(\mathbf{x}_{\text{xyz}}), \tag{11.2}$$

$$\mathbf{x}_{\text{glob}} = \psi\left(\mathbf{M}_{\text{fwd}}\ \phi(\mathbf{x}_{\text{xyz}})\right), \tag{11.3}$$

where $\mathbf{M}_{\text{fwd}} \in \mathbb{R}^{3\times 6}$ is a global transformation matrix and $\mathbf{x}_{\text{glob}}$ is the globally processed image layer. The operator $\phi(\cdot)$ reshapes the image to be $6 \times n$ where n is the number of pixels in the image and each pixel is transformed from three to six dimensions: $[R, G, B] \rightarrow$

$[R, G, B, R^2, G^2, B^2]$, while the operator ψ reshapes the image from $3 \times n$ back to $h \times w \times 3$. We chose $\mathbf{M}_{\text{fwd}}$ to be nonlinear to capture global color processing operations, such as gamma compression.

As most consumer cameras locally process the captured scene-referred images to improve the quality of final rendered images [155], such global color processing may not be able to effectively model the function $\mathcal{F}$. To that end, we use a residual learning mechanism where we model the residual layer $\mathbf{x}_{\text{res}}$ between the locally and globally processed layers of the image as follows:

$$\mathbf{x}_{\text{res}} = \mathcal{F}_{\text{loc}}(\mathbf{x}_{\text{glob}}), \tag{11.4}$$

$$\mathbf{x}_{\text{srgb}} = \mathbf{x}_{\text{glob}} + \mathbf{x}_{\text{res}}. \tag{11.5}$$

Now, the decomposition process applies the inverse process of Eqs. 11.2 – 11.5 as follows:

$$\mathbf{x}_{\text{res}} = \mathcal{G}_{\text{loc}}(\mathbf{x}_{\text{srgb}}), \tag{11.6}$$

$$\mathbf{x}_{\text{glob}} = \mathbf{x}_{\text{srgb}} - \mathbf{x}_{\text{res}}, \tag{11.7}$$

$$\mathbf{M}_{\text{inv}} = \mathcal{G}_{\text{glob}}(\mathbf{x}_{\text{glob}}), \tag{11.8}$$

$$\mathbf{x}_{\text{xyz}} = \psi\left(\mathbf{M}_{\text{inv}}\ \phi(\mathbf{x}_{\text{glob}})\right), \tag{11.9}$$

where $\mathcal{G}_{\text{loc}}(\cdot)$ represents the inverse of residual local processing layer and $\mathcal{G}_{\text{glob}}(\cdot)$ is constrained to produce a global transformation matrix $\mathbf{M}_{\text{inv}} \in \mathbb{R}^{3\times 6}$ that represents the inverse global processing stage.

Our ultimate goal is to allow the manipulation of the reconstructed CIE XYZ image by arbitrary image restoration/enhancement algorithms between the inverse and forward pipeline stages (see Fig. 11.2). It is, however, non-trivial to infer the inverse functions $\mathcal{G}_{\text{loc}}^{-1}(\cdot)$ and $\mathcal{G}_{\text{glob}}^{-1}(\cdot)$ to render back the reconstructed image, as its values may be changed by the image restoration or enhancement algorithms. To that end, we model each of $\mathcal{F}_{\text{glob}}(\cdot)$, $\mathcal{F}_{\text{loc}}(\cdot)$, $\mathcal{G}_{\text{glob}}(\cdot)$, and $\mathcal{G}_{\text{loc}}(\cdot)$ by a neural network.

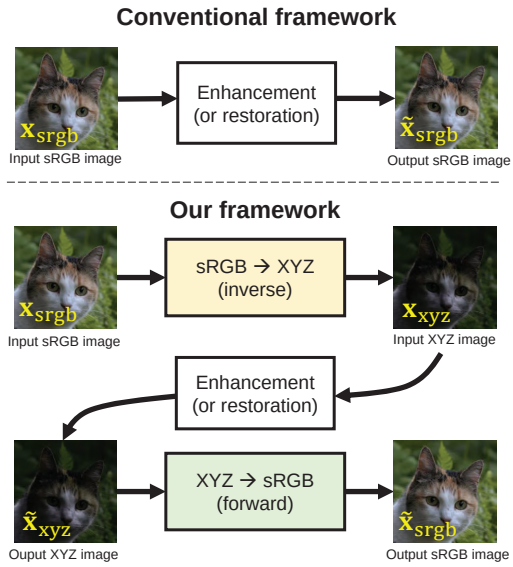


Figure 11.2: An illustration of using our inverse and forward image processing pipelines in an sRGB image restoration/enhancement framework.

11.3.2 Network Design

Imitating this division of the camera imaging pipeline, we build our network architecture to include two sub-networks for modeling both the global and local processing parts for the forward and inverse directions of the imaging pipeline. As shown in Fig. 11.3, we start with the inverse pipeline where the first part is a fully-convolutional neural network (CNN) that models the local processing applied to an input non-linear image (i.e., sRGB image) by predicting the residual image $\mathbf{x}_{\text{res}}$ (Eq. 11.6). Once the local processing layer is predicted, it can be subtracted from the input image $\mathbf{x}_{\text{srgb}}$ to get the globally processed image $\mathbf{x}_{\text{glob}}$ (Eq. 11.7). Then, $\mathbf{x}_{\text{glob}}$ is fed to another sub-network that predicts a global transformation $\mathbf{M}_{\text{inv}}$ that inverts $\mathbf{x}_{\text{glob}}$ back to the linear CIE XYZ image $\mathbf{x}_{\text{xyz}}$ (Eq. 11.9). With this inverse pipeline, we decompose the input image $\mathbf{x}_{\text{srgb}}$ into two image layers, $\mathbf{x}_{\text{res}}$ and $\mathbf{x}_{\text{glob}}$, which represent local and global processing, respectively, and finally output the

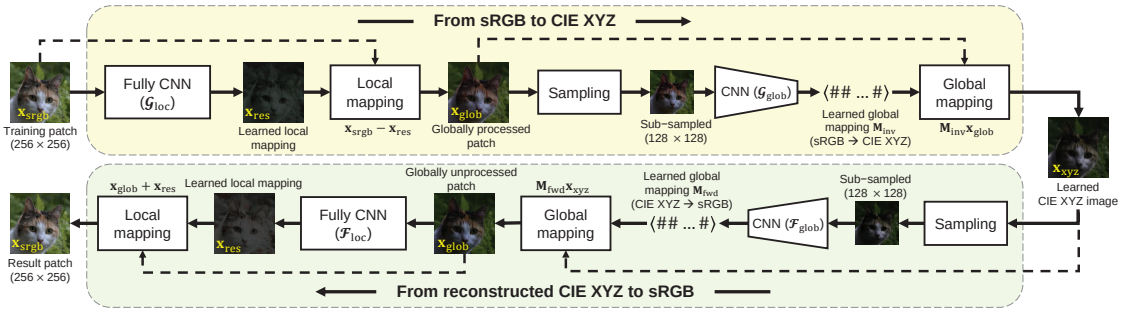


Figure 11.3: Our CIE XYZ image pipeline. The upper part is the inverse pipeline that unprocesses an sRGB image into a CIE XYZ image. The lower part is the forward pipeline that processes a CIE XYZ image into its equivalent sRGB image. The full framework is trainable end-to-end. The CIE XYZ images are scaled 2x to aid visualization.

linear CIE XYZ image $\mathbf{x}_{\text{xyz}}$.

As discussed in Section 11.1, there are computer vision tasks, such as image restoration, that are best processed in a linear image state. A use case of the framework is to convert the input image *imxyz*, process the *imxyz* image, and then render the image back. In this scenario, after decomposing an image and applying an image restoration task to the linear XYZ image, we now need to merge these image layers back to produce the fully processed sRGB image. To model this forward pass of our pipeline, as shown in Fig. 11.3, we use two sub-networks. The first sub-network predicts a global transformation $\mathbf{M}_{\text{fwd}}$ that maps $\mathbf{x}_{\text{xyz}}$ to $\mathbf{x}_{\text{glob}}$ (Eq. 11.3). The second sub-network predicts the residual local processing $\mathbf{x}_{\text{res}}$ that needs to be applied to $\mathbf{x}_{\text{glob}}$ to obtain the final sRGB image $\mathbf{x}_{\text{srgb}}$ (Eq. 11.5). This framework is illustrated in Fig. 11.2 and compared to the conventional way of directly processing the sRGB image.

In order to allow the networks $\mathcal{G}_{\text{glob}}(\cdot)$ and $\mathcal{G}_{\text{loc}}(\cdot)$ to separate the globally and locally processed image layer without having ground truth for both $\mathbf{x}_{\text{glob}}$ and $\mathbf{x}_{\text{res}}$, we apply a scaling factor to the output of the local processing networks $\mathcal{G}_{\text{glob}}(\cdot)$, in both inverse

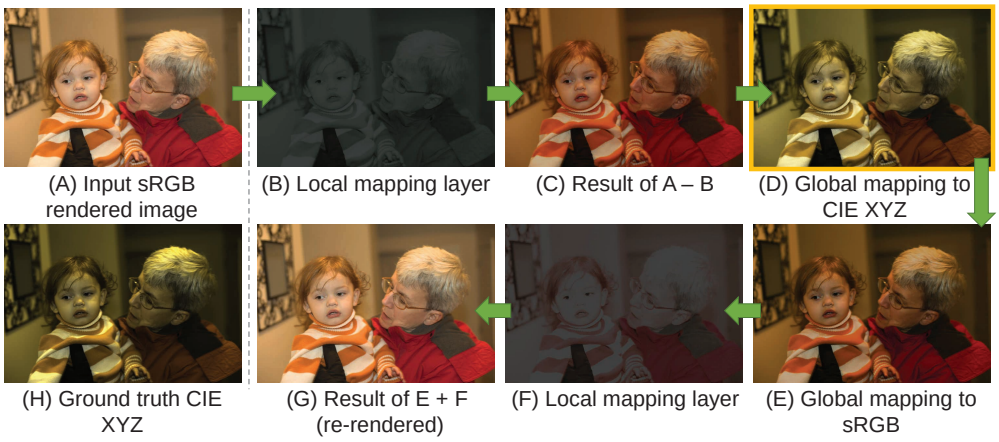


Figure 11.4: Our inverse pipeline decomposites a given camera-rendered sRGB image into a local processed layer and the corresponding CIE XYZ image, while our forward pipeline maps the reconstructed CIE XYZ image to the sRGB color space in an inverse way of our decomposition. The shown image is taken from our testing set. To aid visualization, CIE XYZ images are scaled by a factor of two.

and forward passes, such that the values of $\mathbf{x}_{\text{res}}$ are much smaller than $\mathbf{x}_{\text{glob}}$. In our experiments, we set this scaling factor to 0.25. Figure 11.4 shows an example of the output of each sub-network.

It is challenging to evaluate our global and local processing modules separately; mainly because camera ISP global and local processing modules are typically proprietary and not accessible, and hence, we cannot obtain ground truth data for evaluation.

11.3.3 Loss Function

The objective of the whole network is to minimize the mean absolute error: 1) between the predicted XYZ image $\hat{\mathbf{x}}_{\text{xyz}}$ and its ground truth $\mathbf{x}^*_{\text{xyz}}$ in the inverse pipeline and 2) between the predicted sRGB image $\hat{\mathbf{x}}_{\text{srgb}}$ and its ground truth $\mathbf{x}^*_{\text{srgb}}$ in the forward pipeline, as

follows:

$$\lambda \left| \hat{\mathbf{x}}_{\text{xyz}} - \mathbf{x}^{*}_{\text{xyz}} \right| + \left| \hat{\mathbf{x}}_{\text{srgb}} - \mathbf{x}^{*}_{\text{srgb}} \right|, \tag{11.10}$$

where λ is a weighting factor that we use to deal with the fact that XYZ images generally have lower intensity compared to sRGB images; so this weight can balance the learning behavior between the forward and inverse pipelines. In our experiments, we set $\lambda = 1.5$.

11.3.4 Sub-Networks Architecture

Our local processing sub-networks ($\mathcal{F}_{\text{loc}}$ and $\mathcal{G}_{\text{loc}}$) each consist of 15 blocks of 3×3 convolutional (conv)–LReLU layers. Each conv layer has 32 output channels, with stride of 1 and padding of 1. The last layer of these sub-networks has a single conv layer with three output channels, followed by a tanh operator. As our global processing sub-networks are not fully convolutional, we use a fixed size of input by introducing a differentiable subsampling module that uniformly subsamples 128×128 color values of the processed image by the previous sub-network. Our global sub-network includes five blocks of 3×3 conv–LReLU–2×2 max pooling layers. The conv layers have stride and padding of 1, while the max pooling layers have a stride factor of 2 with no padding. Then, we added a fully connected layer with 1024 output neurons, followed by a dropout layer with a factor of 0.5. The last layer of our global sub-network has a fully connected layer with 18 output neurons to formulate our 3×6 polynomial mapping function.

Our entire framework is a light-weight model with a total of 2,697,578 learnable parameters (∼11MB of memory) for both sRGB-to-XYZ and XYZ-to-sRGB models, and it is fully differentiable for end-to-end training.

11.3.5 Dataset

To train our proposed model, we need a dataset of sRGB images with their corresponding linear images in the CIE XYZ color space. To do so, we start from raw-RGB images taken from the MIT-Adobe FiveK [62]. We then process the raw-RGB images twice to obtain both the sRGB and XYZ versions of each image. For processing raw-RGB images into the XYZ color space, we used the camera pipeline from [7]. This pipeline provides an access to the CIE XYZ values after processing the sensor raw-RGB using the color space transformation (CST) matrices provided with the raw-RGB image. To obtain the camera-like sRGB images, we followed the same procedure explained in Chapter 6 to generate our sRGB dataset for improperly white-balanced images. Specifically, we used the Adobe Camera RAW SDK, which accurately emulates the nonlinearity applied by consumer cameras [20]. In contrast to the dataset generated in Chapter 6, we used the illuminant color estimated by each camera's ISP in the AWB mode.

The MIT-Adobe FiveK [62] dataset contains images captured with different cameras. As a result, the CIE XYZ and sRGB images are rendered with different processing profiles according to the metadata from each camera.

Our method's CIE XYZ fidelity evaluations are tied to the used camera models in the MIT-Adobe FiveK [62]. However, this does not take away from the advantages that the standard canonical CIE XYZ space offers over other linear spaces as a target space for our supervision learning. Our assumption is that by training on images rendered by a broad range of ISP emulations for different camera models, we can learn to unprocess generic processing applied by most cameras in order to achieve a better linearization. Our dataset includes ∼1,200 pairs of sRGB and camera CIE XYZ images. Our dataset will be publicly available upon acceptance.

11.3.6 Training

We divided our dataset into a training set of 971 pairs, a validation set of 50 pairs, and a testing set of 244 pairs. We trained our framework in an end-to-end manner on patches of size 256×256 pixels randomly extracted from our training set, with a mini-batch of size 4. We applied random geometric augmentation (i.e., scaling and reflection) to the extracted patches.

Our framework was trained in an end-to-end manner for 300 epochs using Adam optimizer [204] with gradient decay factor $\beta_1 = 0.9$ and squared gradient decay factor $\beta_2 = 0.999$. We used a learning rate of 10^{-4} with a drop factor of 0.5 every 75 epochs. We added an L_2 regularization with a weight of $\lambda_{\text{reg}} = 10^{-3}$ to our loss in Eq. 11.10 to avoid overfitting.

11.4 Experimental Results

In this section, we first validate the effectiveness of our proposed model in mapping from camera-rendered sRGB images to CIE XYZ, and processing CIE XYZ images back to sRGB. Next, we demonstrate our method's utility on low-light image enhancement task. We refer the reader to Appendix A for additional applications.

11.4.1 From Camera-Rendered sRGB to CIE XYZ, and Back

We first verify our network's ability to unprocess sRGB images to CIE XYZ. We also demonstrate our ability to reconstruct from CIE XYZ back to sRGB. We test our mapping to sRGB using our reconstructed CIE XYZ results as a starting point, and also using the ground-truth CIE XYZ images.

We compared our method with three existing methods. First, we compare with the

standard CIE XYZ mapping [31, 101], which applies a simple 2.2 gamma tone curve. Second, we compare with the recent *unprocessing technique (UPI)* from [59]. The UPI unprocessing technique is non-trainable and inverts the camera ISP, step-by-step, through a series of transformations, such as gamma expansion and inverting color correction matrices. This unprocessing module is then used to generate realistic training data for the task for image denoising. It is only the denoising module of [59] that is a CNN, and that is trainable. For a fair comparison, we compare our results with results of UPI obtained at the CIE XYZ stage.

Third, we compare with CycleISP [397], a recent network architecture that aims at simulating the camera ISP mapping between raw and sRGB stages. Since CycleISP is targeting a camera-specific sRGB-to-raw mapping, we had to introduce some slight modifications to their architecture to make it suitable to our objective, i.e., mapping between sRGB and CIE XYZ. In particular, we omitted their noise injection and color correction modules and modified their RGB2RAW and RAW2RGB networks to have 3-channel inputs and outputs. We retrained the CycleISP with the same training settings used to train our model for a fair comparison.

Table 11.1 shows peak signal-to-noise ratio (PSNR) results averaged over 244 unseen testing images from the MIT-Adobe FiveK dataset [62]. The terms Q1, Q2, and Q3 refer to the first, second (median), and third quantile, respectively, of the PSNR values obtained by each method. For the standard XYZ, the results of mapping from the reconstructed CIE XYZ images back to sRGB are not reported because standard XYZ uses an invertible transform. The sRGB reconstruction error from the UPI model [59] is high due to the fact that the tone mapping is not perfectly invertible. It can be observed from the results that we outperform all competing methods by a sound margin. Qualitative comparisons are provided in Fig. 11.5.

Table 11.1: Results (in terms of PSNR) of camera-rendered sRGB ↔ CIE XYZ mapping. We compare our results against the standard XYZ mapping (the 2.2 gamma tone curve) [31,101], the recent unprocessing technique (UPI) [59], and CycleISP [397]. Average PSNR (dB) results are reported on 244 unseen testing pairs (camera-rendered sRGB and corresponding CIE XYZ images) from the MIT-Adobe FiveK dataset [62]. We show results of mapping from both reconstructed (Rec.) CIE XYZ images and ground truth (GT) CIE XYZ images to the corresponding camera-rendered sRGB images. Highest PSNR values are shown in boldface and highlighted in yellow.

Method	**sRGB → XYZ**				**Rec. XYZ → sRGB**				**GT XYZ → sRGB**			
	Avg.	**Q1**	**Q2**	**Q3**	**Avg.**	**Q1**	**Q2**	**Q3**	**Avg.**	**Q1**	**Q2**	**Q3**
Standard [31, 101]	21.84	16.88	20.91	25.24	-	-	-	-	22.22	19.19	21.79	24.37
Unprocessing [59]	22.19	19.31	22.12	24.75	37.72	37.78	40.56	41.88	18.04	15.67	17.79	20.02
CycleISP [397]	28.29	23.63	28.08	31.98	34.78	30.60	34.20	37.22	20.91	18.36	21.42	24.31
Ours	**29.66**	**23.77**	**29.57**	**34.71**	**43.82**	**41.43**	**43.94**	**46.58**	**27.44**	**23.57**	**28.32**	**30.88**

As shown in Table 11.1, the mapping to sRGB from reconstructed CIE XYZ is better than mapping from ground-truth CIE XYZ. For our method, this behavior is expected because the forward model is trained on the reconstructed CIE XYZ, not the ground truth. Also, for the UPI method, as it is based on matrix inversion, the mapping from the reconstructed CIE XYZ makes the transformation more accurate than mapping from the ground truth.

Lastly, we did experiment with different choices for the global mapping polynomial to validate the polynomial kernel used in our method (see Table 11.2) We found that our chosen ($[R, G, B, R^2, G^2, B^2]$) polynomial yielded the best results.

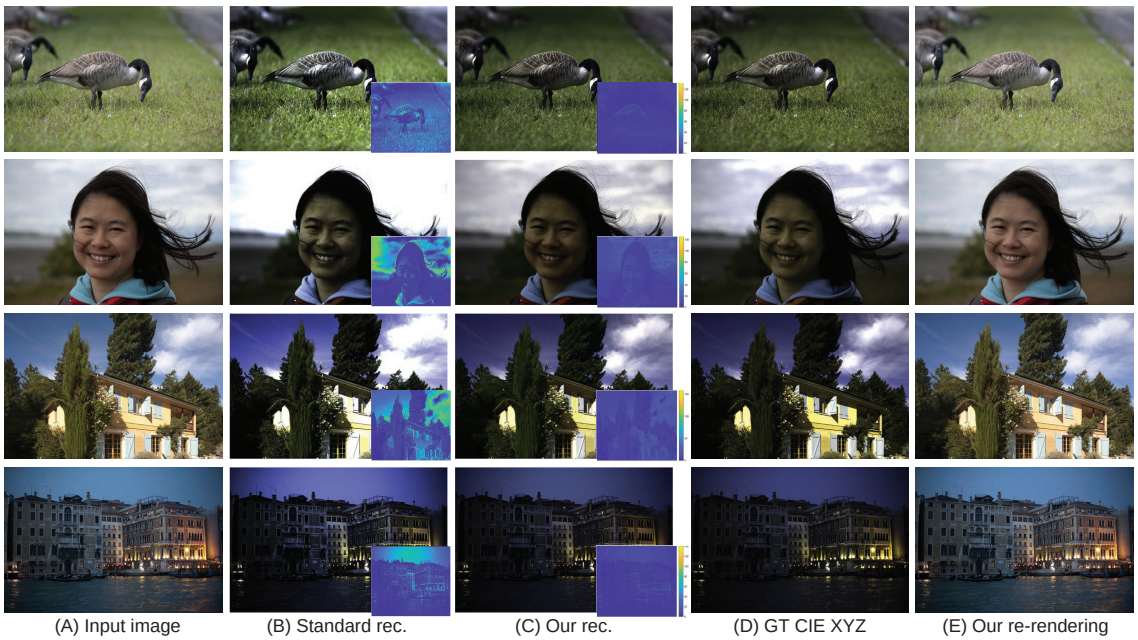


Figure 11.5: Qualitative comparisons for CIE XYZ reconstruction and rendering. (A) The input sRGB rendered image. (B) Standard display-referred CIE XYZ reconstruction [31, 101]. (C) Our reconstruction. (D) The ground-truth scene-referred CIE XYZ image. (E) Our re-rendering result from the reconstructed CIE XYZ image. To aid visualization, CIE XYZ images are scaled by a factor of two. Input images are taken from the MIT-Adobe FiveK dataset [62].

11.5 Comparison with U-Net Baseline

We compare our proposed network against a U-Net-based baseline. This baseline consists of two U-Net-like [323] models trained in an end-to-end manner using the same training settings used to train our network (i.e., epochs, training patches, and loss function). Each U-Net model consists of a 3-level encoder/decoder with skip connections. The output channels of the first conv layer in the encoder unit has 28 channels. The two U-Net models have a total of 2,949,246 learnable parameters, compared to 2,697,578 learnable parameters

Figure 11.6: Our XYZ reconstruction provides a wider range of tonal values compared to the standard CIE XYZ mapping [31,101]. (A) The input sRGB image. (B) Standard XYZ reconstruction [31,101]. (C) Our XYZ reconstruction. The input image is taken from [367].

Table 11.2: Effect of different polynomial terms on global mapping results (in terms of PSNR) for mapping camera-rendered sRGB to CIE XYZ mapping; and mapping from both reconstructed (Rec.) CIE XYZ images and ground truth (GT) CIE XYZ images to the corresponding camera-rendered sRGB images. Highest PSNR values are shown in boldface and highlighted in yellow.

Polynomial terms	**sRGB → XYZ**				**Rec. XYZ → sRGB**				**GT XYZ → sRGB**			
	Avg.	**Q1**	**Q2**	**Q3**	**Avg.**	**Q1**	**Q2**	**Q3**	**Avg.**	**Q1**	**Q2**	**Q3**
Linear $[R, G, B]$	26.85	22.07	26.72	31.58	**44.23**	41.06	**44.53**	**47.72**	23.06	19.49	22.97	26.18
$[R, G, B, R^2, G^2, B^2]$ (our choice)	**29.66**	**23.77**	**29.57**	**34.71**	43.82	**41.43**	43.94	46.58	**27.44**	**23.57**	**28.32**	**30.88**
$[R, G, B, R^2, G^2, B^2, RG, RB, GB]$	27.89	23.47	27.04	33.28	39.64	38.18	41.27	43.47	25.04	21.32	25.43	30.01

in our network, and they were trained to map from sRGB to XYZ and from XYZ back to sRGB, similar to our model.

Table 11.3 shows the results obtained by the U-Net baseline and our network on our testing set.

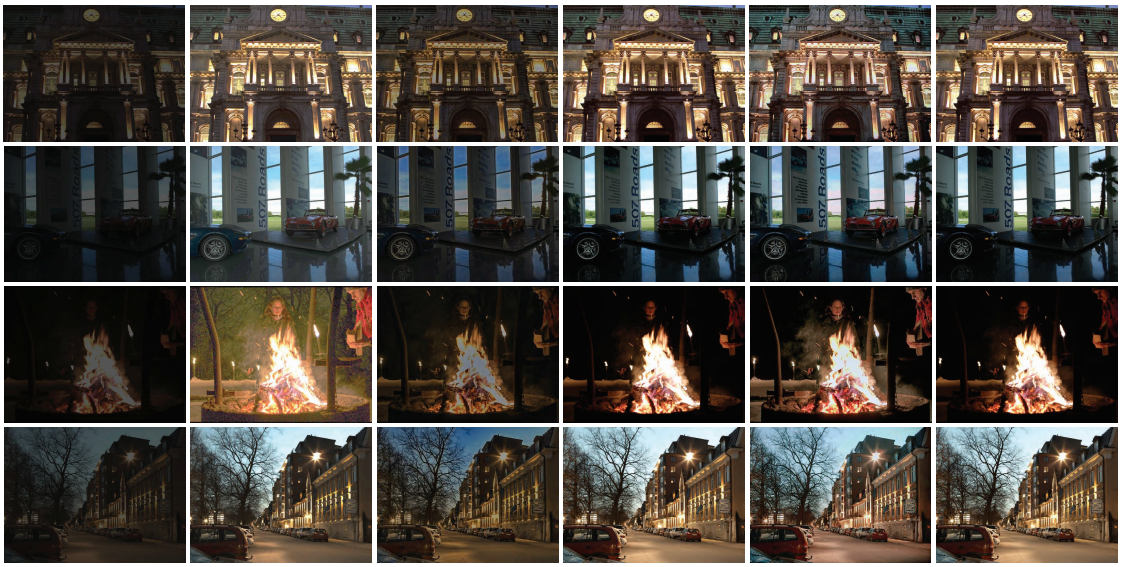


Figure 11.7: (A) The input sRGB rendered image. (B) Adobe Photoshop HDR results. (C) Deep Photo Enhancer results [75]. (D) HDR result of [104]. (E) Our re-rendered images after photo-finishing enhancement. (F) Ground-truth images. Input images are taken from [104].

Figure 11.8: Example from the under-exposure testing set [369]. (A) Input image. (B) Our enhanced rendered image. (C) Expert-retouched image.

Table 11.3: Comparison between our network and two U-Net models trained in an end-to-end manner to map from sRGB to CIE XYZ and back. Both networks, ours and the two U-Net models, have approximately the same number of learnable parameters and both were trained using the same training settings. The best PSNR (dB) values are shown in boldface and highlighted in yellow.

Method	**sRGB → XYZ**				**Rec. XYZ → sRGB**			
	Avg.	**Q1**	**Q2**	**Q3**	**Avg.**	**Q1**	**Q2**	**Q3**
U-Net [323]	20.05	16.84	19.76	22.78	43.39	40.56	43.40	45.91
Ours	**29.66**	**23.77**	**29.57**	**34.71**	**43.82**	**41.43**	**43.94**	**46.58**

Figure 11.9: (A) The input image. (B) Image enhancement in sRGB. (C) Image enhancement in standard XYZ reconstruction. (D) Our enhanced re-rendering. (E) Expert enhancement. The enhancement is based on fusion of "multi-exposed" images [265] and local details enhancement [297]. The image is from the under-exposure testing set [369].

11.5.1 Low-Light Image Enhancement

Many photographers prefer to edit photographs in the linear raw-RGB sensor space rather than the nonlinear 8-bit sRGB space, due to the fact that raw-RGB images provide higher tonal values compared to sRGB camera-rendered images [330]. Similar to the raw-RGB space, the CIE XYZ space is linear scene-referred with higher tonal values compared to

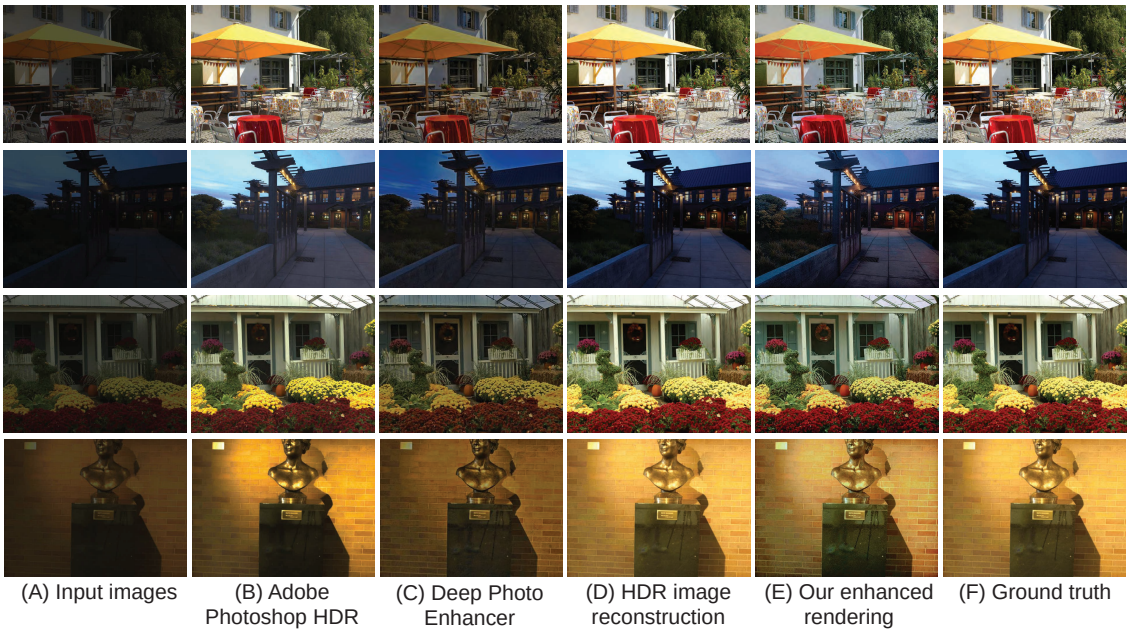


Figure 11.10: Low-light image enhancement application. (A) Input sRGB rendered image. (B) Adobe Photoshop HDR results. (C) Deep Photo Enhancer results [75]. (D) HDR result of [104]. (E) Our re-rendered images after photo-finishing enhancement. (F) Ground truth images. Input images are taken from [104].

the final sRGB space. Thus, we can also benefit from our linear CIE XYZ space for image enhancement tasks.

In this set of experiments, we present a set of simple operations that can achieve results on par with recent methods designed for low-light image enhancement. Specifically, we apply the following set of heuristic operations to perform low-light image enhancement. As our reconstructed XYZ image has a wider range of tonal values (see Fig. 11.6), we apply a set of synthetic digital gains to simulate multi-exposure settings. This simulation does not introduce any new information that did not exist in the original image; however, it allows us to better explore the range of tonal values provided in our reconstructed image—we can think of this operation as an ISO gain that is applied on board cameras to amplify the

Figure 11.11: Qualitative comparison for low-light image enhancement task. Images are taken from the under-exposure testing set [369]. (A) Input image. (B) Adobe Photoshop HDR results. (C) Results of deep photo enhancer [75]. (D) Our enhanced rendered image. (E) Expert-retouched image.

captured image signal. To that end, we multiply the reconstructed image by four different factors. These factors can be tuned in an interactive manner based on each image, but we preferred to fix these hyperparameters over all experiments. In particular, we multiplied our reconstructed XYZ image by (0.1, 1.4, 2.7, 4.0) to generate four different versions of our reconstructed XYZ image. Following this, we apply an off-the-shelf exposure-fusion algorithm [265] to create the modified XYZ layer. To enhance the local details, we apply a local details enhancement method [297] on our forward local sRGB reconstructed layer. Figure 11.7 shows examples of our results. As can be seen, we achieve on par results with state-of-the-art methods designed specifically for the given image enhancement task.

Table 11.4: Quantitative results of the photo-finishing enhancement application using 500 under-exposure images provided in [369].

Method	**PSNR**
White-Box [170]	18.57
Distort-and-Recover [298]	20.97
HDRNet [134]	21.96
Deep photo enhancer [75]	22.150
DeepUPE [369]	23.04
Enhanced in sRGB	16.92
Enhanced in rec. standard XYZ	18.41
Our enhanced re-rendering	21.03

We further evaluated this simple pipeline on 500 under-exposed images taken from [369]. Figure 11.8 shows a qualitative example. We show a quantitative comparison in Table 11.4. Applying digital gain to our reconstructed space provides better results compared to using the standard XYZ reconstruction or the nonlinear sRGB space. This is due to the fact that our reconstructed images have a better linearization with a high tonal range; see Fig. 11.9. We provide additional results in Figs. 11.10 and 11.11.

11.6 Summary

We have proposed a method and DNN model that can map back and forth between nonlinear sRGB and linear CIE XYZ images more accurately compared to alternative approaches. Our method is based on learning a decomposition of sRGB images into a globally processed and locally processed image layers. The learned globally processed image

layer is then used to learn a mapping to the device independent CIE XYZ color space. By utilizing the decomposed image layers produced by our method, we show that our model can be used to perform low-light image enhancement.

12 Correcting Colors in Exposure Errors

The previous chapter presented a linearization method that could be used to enhance low-light and under-exposed image colors. As discussed earlier, exposure problems are categorized as either: (i) overexposed, where the camera exposure was too long, resulting in bright and washed-out image regions, or (ii) underexposed, where the exposure was too short, resulting in dark regions. *Both* under- and overexposure greatly reduce the contrast and visual appeal of an image. Prior work mainly focuses on underexposed images or general image enhancement. In contrast, this chapter presents a method targeting both over- and underexposure errors in photographs[1]. We formulate the exposure correction problem as two main sub-problems: (i) *color enhancement* and (ii) *detail enhancement*. Accordingly, we propose a coarse-to-fine DNN model, trainable in an end-to-end manner, that addresses each sub-problem separately. A key aspect of our solution is a new dataset of over 24,000 images exhibiting the broadest range of exposure values to date with a corresponding properly exposed image. Our method achieves results on par with existing state-of-the-art methods on underexposed images and yields significant improvements for images suffering from overexposure errors. The source code and dataset of this work are

[1]Work done while the author was an intern at Samsung AI Center – Toronto; this work is under review. A preprint version is available in [18]: Mahmoud Afifi, Konstantinos G Derpanis, Björn Ommer, and Michael S. Brown. Learning Multi-Scale Photo Exposure Correction. To appear In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2021.

available on GitHub: https://github.com/mahmoudnafifi/Exposure_Correction.

12.1 Introduction

As discussed in Chapter 3, digital cameras adjust cature exposure to control the overall brightness levels in the image. This adjustment can be controlled manually or performed automatically in the AE mode, where cameras uses TTL metering that measures the amount of light received from the scene to adjust the EV to compensate for high/low level of brightness in the captured scene [301].

We have also showed that in Chapter 3 prior work mainly focuses on underexposed images or general image enhancement. In contrast to the majority of prior work, our work is the first deep learning method to explicitly correct *both* overexposed and underexposed photographs with a single model.

Our method is enabled by generating a large dataset of images with exposure errors. Unlike existing datasets for exposure correction, our dataset is rendered with a wide range of exposure errors to cover both cases of exposure errors—i.e., over- and under-exposure errors. Figure 12.2 shows a comparison between our dataset and the LOL dataset in terms of the number of images and the variety of exposure errors in each dataset. The LOL dataset covers a relatively small fraction of the possible exposure levels, as compared to our introduced dataset. Our dataset is based on the MIT-Adobe FiveK dataset [62] and is accurately rendered by adjusting the high tonal values provided in camera sensor raw-RGB images to realistically emulate camera exposure errors.

Contributions We propose a coarse-to-fine deep learning method for exposure error correction of *both* over- and underexposed sRGB images. Our approach formulates the exposure correction problem as two main sub-problems: (i) color and (ii) detail enhancement.

Figure 12.1: Photographs with over- and underexposure errors and the results of our method using a *single* model for exposure correction. These sample input images are taken from outside our dataset to demonstrate the generalization of our trained model.

We propose a coarse-to-fine DNN model, trainable in an end-to-end manner, that begins by correcting the global color information and subsequently refines the image details. In addition to our DNN model, a key contribution to the exposure correction problem is a new dataset containing over 24,000 images rendered from raw-RGB to sRGB with different exposure settings with broader exposure ranges than previous datasets. Each image in our dataset is provided with a corresponding properly exposed reference image. Lastly, we present an extensive set of evaluations and ablations of our proposed method with comparisons to the state of the art. We demonstrate that our method achieves results on par with previous methods dedicated to underexposed images and yields significant improvements on overexposed images. Furthermore, our model generalizes well to images outside our dataset.

12.2 Our Dataset

To train our model, we need a large number of training images rendered with realistic over- and underexposure errors and corresponding properly exposed ground truth images. As discussed in Chapter 3, such datasets are currently not publicly available to support exposure correction research. For this reason, our first task is to create a new dataset.

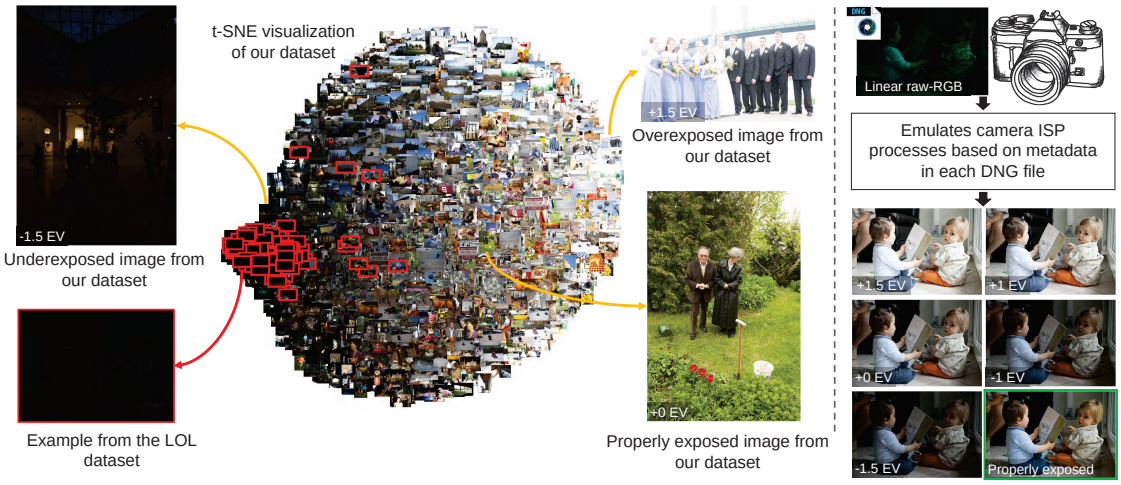
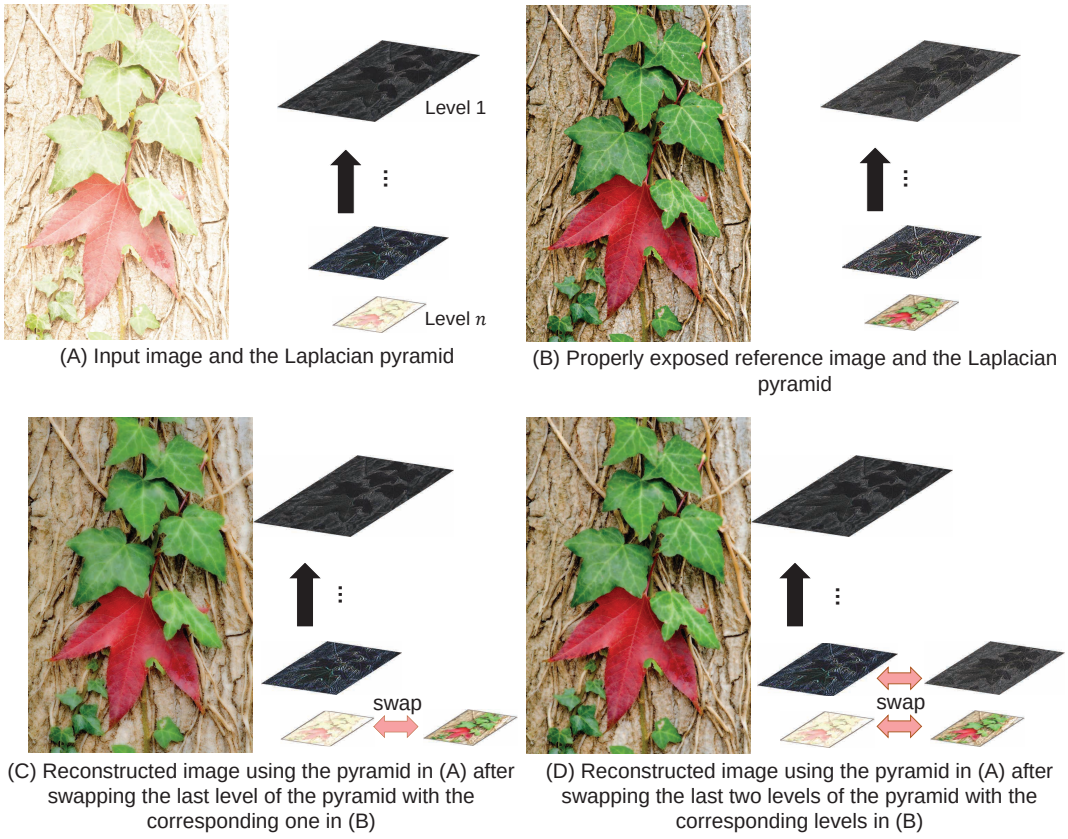


Figure 12.2: Our dataset contains images with different exposure error types and their corresponding properly exposed reference images. Shown is a t-SNE visualization [258] of all images in our dataset and the low-light (LOL) paired dataset (outlined in red) [375]. Notice that LOL covers a relatively small fraction of the possible exposure levels, as compared to our introduced dataset. Our dataset was rendered from linear raw-RGB images taken from the MIT-Adobe FiveK dataset [62]. Each image was rendered with different relative exposure values (EVs) by an accurate emulation of the camera ISP processes.

As done in Chapter 11, our dataset is rendered from the MIT-Adobe FiveK dataset [62], which has 5,000 raw-RGB images and corresponding sRGB images rendered manually by five expert photographers [62].

For each raw-RGB image, we use the Adobe Camera Raw SDK [8] to emulate different EVs as would be applied by a camera [331]. We render each raw-RGB image with the AWB settings and with different digital EVs to mimic real exposure errors. Specifically, we use the relative EVs -1.5, -1, $+0$, $+1$, and $+1.5$ to render images with underexposure errors, a zero gain of the original EV, and overexposure errors, respectively. The zero-gain relative EV is equivalent to the original exposure settings applied onboard the camera



(A) Input image and the Laplacian pyramid

(B) Properly exposed reference image and the Laplacian pyramid

(C) Reconstructed image using the pyramid in (A) after swapping the last level of the pyramid with the corresponding one in (B)

(D) Reconstructed image using the pyramid in (A) after swapping the last two levels of the pyramid with the corresponding levels in (B)

Figure 12.3: Motivation behind our coarse-to-fine exposure correction approach. Example of an overexposed image and its corresponding properly exposed image shown in (A) and (B), respectively. The Laplacian pyramid decomposition allows us to enhance the color and detail information sequentially, as shown in (C) and (D), respectively.

during capture time.

As the ground truth images, we use images that were manually retouched by an expert photographer (referred to as Expert C in [62]) as our target correctly exposed images, rather than using our rendered images with +0 relative EV. The reason behind this choice is that a significant number of images contain backlighting or partial exposure errors in the original exposure capture settings. The expert adjusted images were performed in ProPhoto RGB color space [62] (rather than raw-RGB), which we converted to a standard

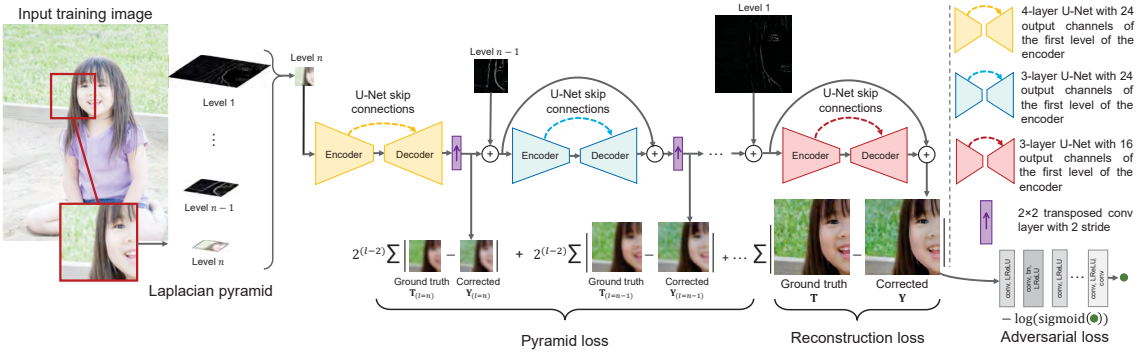


Figure 12.4: Overview of our image exposure correction architecture. We propose a coarse-to-fine deep network to progressively correct exposure errors in 8-bit sRGB images. Our network first corrects the global color captured at the final level of the Laplacian pyramid and then the subsequent frequency layers.

8-bit sRGB color space encoding.

In total, our dataset contains 24,330 8-bit sRGB images with different digital exposure settings. We discarded a small number of images that had misalignment with their corresponding ground truth image. These misalignments are due to different usage of the DNG crop area metadata by Adobe Camera Raw SDK and the expert. Our dataset is divided into three sets: (i) training set of 17,675 images, (ii) validation set of 750 images, and (iii) testing set of 5,905 images. The training, validation, and testing sets, use different images taken from the FiveK dataset. This means the training, validation, and testing images do not share any images in common. Figure 12.2 shows examples of our generated 8-bit sRGB images and the corresponding properly exposed 8-bit sRGB reference images. We acknowledge that digital exposure used to produce our dataset does not consider the noise characteristics that could change based on the camera exposure settings.

12.3 Methodology

Given an 8-bit sRGB input image, $\mathbf{I}$, rendered with the incorrect exposure setting, our method aims to produce an output image, $\mathbf{Y}$, with fewer exposure errors than those in $\mathbf{I}$. As we simultaneously target both over- and underexposed errors, our input image, $\mathbf{I}$, is expected to contain regions of nearly over- or under-saturated values with corrupted color and detail information. We propose to correct color and detail errors of $\mathbf{I}$ in a sequential manner. Specifically, we process a multi-resolution representation of $\mathbf{I}$, rather than directly dealing with the original form of $\mathbf{I}$. We use the Laplacian pyramid [61] as our multi-resolution decomposition, which is derived from the Gaussian pyramid of $\mathbf{I}$.

12.3.1 Coarse-to-Fine Exposure Correction

Let $\mathbf{X}$ represent the Laplacian pyramid of $\mathbf{I}$ with n levels, such that $\mathbf{X}_{(l)}$ is the $l^{\texttt{th}}$ level of $\mathbf{X}$. The last level of this pyramid (i.e., $\mathbf{X}_{(n)}$) captures low-frequency information of $\mathbf{I}$, while the first level (i.e., $\mathbf{X}_{(1)}$) captures the high-frequency information. Such frequency levels can be categorized into: (i) global color information of $\mathbf{I}$ stored in the low-frequency level and (ii) image coarse-to-fine details stored in the mid- and high-frequency levels. These levels can be later used to reconstruct the full-color image $\mathbf{I}$.

Figure 12.3 motivates our coarse-to-fine approach to exposure correction. Figures 12.3-(A) and (B) show an example overexposed image and its corresponding well-exposed target, respectively. As observed, a significant exposure correction can be obtained by using only the low-frequency layer (i.e., the global color information) of the target image in the Laplacian pyramid reconstruction process, as shown in Fig. 12.3-(C). We can then improve the final image by enhancing the details in a sequential way by correcting each level of the Laplacian pyramid, as shown in Fig. 12.3-(D). Practically, we do not have access to

the properly exposed image in Fig. 12.3-(B) at the inference stage, and thus our goal is to predict the missing color/detail information of each level in the Laplacian pyramid.

Inspired by this observation and the success of coarse-to-fine architectures for various other computer vision tasks (e.g., [93, 217, 256, 335]), we design a DNN that corrects the global color and detail information of $\mathbf{I}$ in a sequential manner using the Laplacian pyramid decomposition. The remaining parts of this section explain the technical details of our model (Sec. 12.3.2), including details of the losses (Sec. 12.3.3), inference phase (Sec. 12.3.5), and training (Sec. 12.3.6).

12.3.2 Coarse-to-Fine Network

Our image exposure correction architecture sequentially processes the n-level Laplacian pyramid, $\mathbf{X}$, of the input image, $\mathbf{I}$, to produce the final corrected image, $\mathbf{Y}$. The proposed model consists of n sub-networks. Each of these sub-networks is a U-Net-like architecture [323] with untied weights. We allocate the network capacity in the form of weights based on how significantly each sub-problem (i.e., global color correction and detail enhancement) contributes to our final result. Figure 12.4 provides an overview of our network. As shown, the largest (in terms of weights) sub-network in our architecture is dedicated to processing the global color information in $\mathbf{I}$ (i.e., $\mathbf{X}_{(n)}$). This sub-network (shown in yellow in Fig. 12.4) processes the low-frequency level $\mathbf{X}_{(n)}$ and produces an upscaled image $\mathbf{Y}_{(n)}$. The upscaling process scales up the output of our sub-network by a factor of two using strided transposed convolution with trainable weights. Next, we add the first mid-frequency level $\mathbf{X}_{(n-1)}$ to $\mathbf{Y}_{(n)}$ to be processed by the second sub-network in our model. This sub-network enhances the corresponding details of the current level and produces a residual layer that is then added to $\mathbf{Y}_{(n)} + \mathbf{X}_{(n-1)}$ to reconstruct image $\mathbf{Y}_{(n-1)}$, which is equivalent to the corresponding Gaussian pyramid level $n-1$. This refinement-upsampling process proceeds

until the final output image, $\mathbf{Y}$, is produced. Our network is fully differentiable and thus can be trained in an end-to-end manner.

12.3.3 Losses

We train our model end-to-end to minimize the following loss function:

$$\mathcal{L} = \mathcal{L}_{\text{rec}} + \mathcal{L}_{\text{pyr}} + \mathcal{L}_{\text{adv}}, \tag{12.1}$$

where $\mathcal{L}_{\text{rec}}$ denotes the reconstruction loss, $\mathcal{L}_{\text{pyr}}$ the pyramid loss, and $\mathcal{L}_{\text{adv}}$ the adversarial loss. The individual losses are defined next.

Reconstruction Loss: We use the $\mathtt{L}_1$ loss function between the reconstructed and properly exposed reference images. This loss can be expressed as follows:

$$\mathcal{L}_{\text{rec}} = \sum_{p=1}^{3hw} \left| \mathbf{Y}(p) - \mathbf{T}(p) \right|, \tag{12.2}$$

where h and w denote the height and width of the training image, respectively, and p is the index of each pixel in our corrected image, $\mathbf{Y}$, and the corresponding properly exposed reference image, $\mathbf{T}$, respectively.

Pyramid Loss: To guide each sub-network to follow the Laplacian pyramid reconstruction procedure, we introduce dedicated losses at each pyramid level. Let $\mathbf{T}_{(l)}$ denote the $l^{\mathtt{th}}$ level of the Gaussian pyramid of our reference image, $\mathbf{T}$, after upsampling by a factor of two. We use the standard pyramid upsampling operation [61]. Our pyramid loss is computed as follows:

$$\mathcal{L}_{\text{pyr}} = \sum_{l=2}^{n} 2^{(l-2)} \sum_{p=1}^{3h_l w_l} \left| \mathbf{Y}_{(l)}(p) - \mathbf{T}_{(l)}(p) \right|, \tag{12.3}$$

where h_l and w_l are twice the height and width of the $l^{\mathtt{th}}$ level in the Laplacian pyramid of the training image, respectively, and p is the index of each pixel in our corrected image

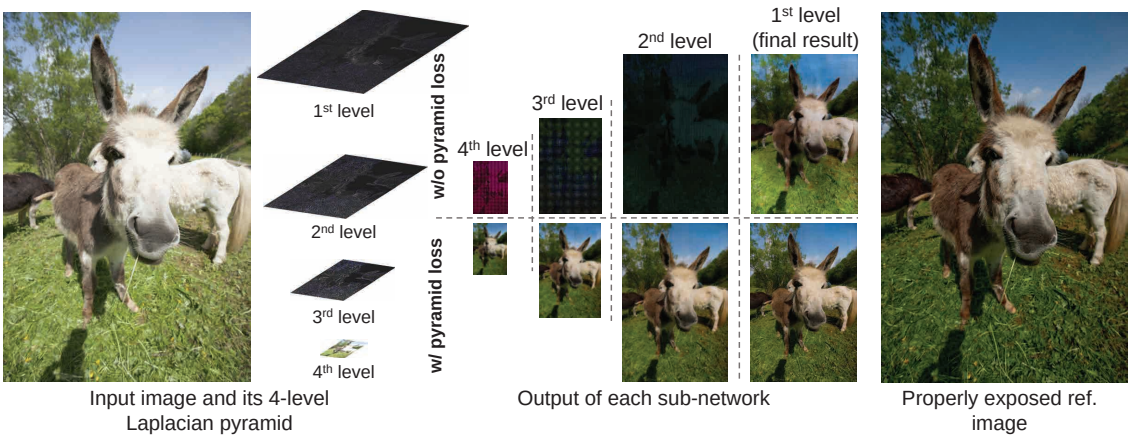


Figure 12.5: Multiscale losses. Shown are the output of each sub-net trained with and without the pyramid loss (Eq. 12.3).

at the $l^{\mathtt{th}}$ level $\mathbf{Y}_{(l)}$ and the properly exposed reference image at the same level $\mathbf{T}_{(l)}$, respectively. The pyramid loss not only gives a principled interpretation of the task of each sub-network but also results in less visual artifacts compared to training using only the reconstruction loss, as can be seen in Fig. 12.5. Notice that without the intermediate pyramid losses, the multi-scale reconstructions, shown in Fig. 12.5 (right-top), deviate widely from the intermediate Gaussian targets compared to using the pyramid loss at each scale, as shown in Fig. 12.5 (right-bottom).

Adversarial Loss: To perceptually enhance the reconstruction of the corrected image output in terms of realism and appeal, we also consider an adversarial loss as a regularizer. This adversarial loss term can be described by the following equation [143]:

$$\mathcal{L}_{\text{adv}} = -3hwn \log\left(\mathcal{S}\left(\mathcal{D}\left(\mathbf{Y}\right)\right)\right), \tag{12.4}$$

where $\mathcal{S}$ is the sigmoid function and $\mathcal{D}$ is a discriminator DNN that is trained together with our main network.

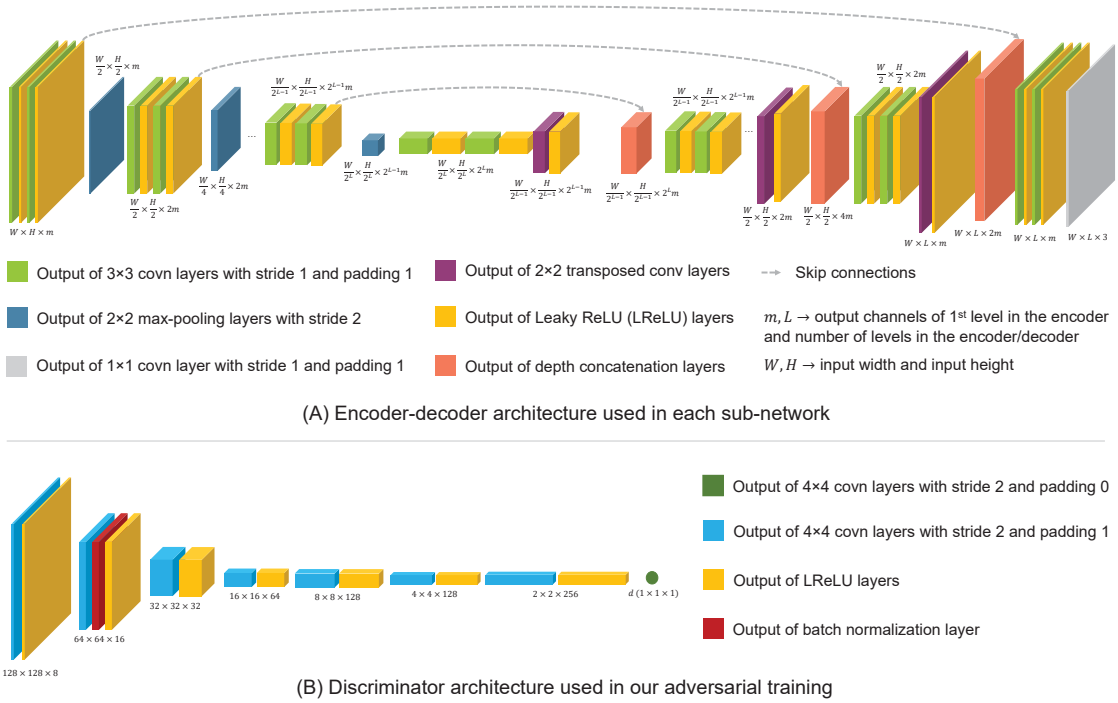


Figure 12.6: Details of the architectures used in our work. (A) Encoder-decoder architecture [323] used to design our sub-networks in the main network. (B) Discriminator architecture.

12.3.4 Network Details

Our main network consists of four sub-networks with ∼7M parameters trained in an end-to-end manner. The largest network capacity is dedicated to the first sub-network with decreasing amounts of capacity as we move from coarse-to-fine scales. Each sub-network accepts a different representation of the input image extracted from the Laplacian pyramid decomposition. The first sub-network is a four-layer encoder-decoder network with skip connections (i.e., U-Net-like architecture [323]). The output of the first conv layer has 24 channels. Our first sub-network has ∼4.4M learnable parameters and accepts the low-frequency band level of the Laplacian pyramid, i.e., $\mathbf{X}_{(4)}$. The result of the first sub-network

is then upscaled using a $2\times2\times3$ transposed conv layer with three output channels and a stride of two. This processed layer is then added to the first mid-frequency band level of the Laplacian pyramid (i.e., $\mathbf{X}_{(3)}$) and is fed to the second sub-network.

The second sub-network is a three-layer encoder-decoder network with skip connections. It has 24 channels in the first conv layer of the encoder, with a total of ∼1.1M learnable parameters. The second sub-network processes the upscaled input from the first sub-network and outputs a residual layer, which is then added back to the input to the second sub-network followed by a $2\times2\times3$ transposed conv layer with three output channels and a stride of two. The result is added to the second mid-frequency band level of the Laplacian pyramid (i.e., $\mathbf{X}_{(2)}$) and is fed to the third sub-network, which generates a new residual that is added back again to the input of this sub-network.

The third sub-network has the same design as the second network. Finally, the result is added to the high-frequency band level of the Laplacian pyramid (i.e., $\mathbf{X}_{(1)}$) and is fed to the fourth sub-network to produce the final processed image.

The final sub-network is a three-layer encoder-decoder network with skip connections and has ∼482.2K learnable parameters, where the output of the first conv layer in its encoder has 16 channels. We provide the details of the main encoder-decoder architecture of each sub-network in Fig. 12.6-(A).

In the adversarial training of our network, we use a light-weight discriminator network with ∼1M learnable parameters. We provide the details of the discriminator in Fig. 12.6-(B). Notice that unlike our main network, we resize all input image patches to have 256×256 pixels before being processed by the discriminator. The output of the last layer in our discriminator is a single scalar value which is then used in our loss during the optimization.

Figure 12.7: We evaluate the results of input images against all five expert photographers' edits from the FiveK dataset [62].

12.3.5 Inference Stage

Our network is fully convolutional and can process input images with different resolutions. While our model requires a reasonable memory size (∼7M parameters), processing high-resolution images requires a high computational power that may not always be available. Furthermore, processing images with considerably higher resolution (e.g., 16-megapixel) than the range of resolutions used in the training process can affect our model's robustness with large homogeneous image regions. This issue arises because our network was trained on a certain range of effective receptive fields, which is very low compared to the receptive fields required for images with very high resolution. To that end, we use the bilateral guided upsampling method [73] to process high-resolution images. First, we resize the input test image to have a maximum dimension of 512 pixels. Then, we process the downsampled version of the input image using our model, followed by applying the fast upsampling technique [73] with a bilateral grid of $22 \times 22 \times 8$ cells. This process allows us to process a 16-megapixel image in ∼4.5 seconds on average. This time includes ∼0.5 seconds to run our network on an NVIDIA® GeForce GTX 1080™ GPU and ∼4 seconds on an Intel® Xeon®

Figure 12.8: Qualitative results of correcting images with exposure errors. Shown are the input images from our test set, results from the DPED [175], results from the Deep UPE [75], our results, and the corresponding ground truth images.

E5-1607 @ 3.10 GHz machine for the guided upsampling process. Note the runtime of the guided upsampling step can be significantly improved with a Halide implementation [315].

12.3.6 Training Details

In our implementation, we use a Laplacian pyramid with four levels (i.e., $n = 4$) and thus we have four sub-networks in our model—an ablation study evaluating the effect on the number of Laplacian levels, including a comparison with a vanilla U-Net architecture, is presented in Sec. 12.4. We trained our model on patches randomly extracted from training images with different dimensions. We begin our training without $\mathcal{L}_{\text{adv}}$ on 176,590 patches with dimensions of 128×128 pixels extracted randomly from our training images for 40 epochs. The mini-batch size is set to 32. The learning rate is decayed by a factor of 0.5 after the first 20 epochs. Then, we continue training on another 105,845 patches with dimensions of 256×256 pixels for 30 epochs with a mini-batch size of eight. At this stage, we train our main network without $\mathcal{L}_{\text{adv}}$ for 15 epochs and continue training for another 15 epochs with $\mathcal{L}_{\text{adv}}$. The learning rates for the main network and the discriminator network are decayed by a factor of 0.5 every 10 epochs. Finally, we fine-tune the trained networks on another 69,515 training patches with dimensions of 512×512 pixels for 20 epochs with a mini-batch size of four and a learning rate decay of 0.5 applied every five epochs.

We use the Adam optimizer [204] to minimize our loss function in Eq. 12.1. Inspired by previous work [255], we initially train without the adversarial loss term $\mathcal{L}_{\text{adv}}$ to speed up the convergence of our main network. Upon convergence, we then add the adversarial loss term $\mathcal{L}_{\text{adv}}$ and fine-tune our network to enhance our initial results.

We use He et al.'s method [158] to initialize the weights of our encoder and decoder conv layers, while the bias terms are initialized to zero. We minimize our loss functions using the Adam optimizer [204] with a decay rate $\beta_1 = 0.9$ for the exponential moving averages of the gradient and a decay rate $\beta_2 = 0.999$ for the squared gradient. We use a learning rate of 10^{-4} to update the parameters of our main network and a learning rate of 10^{-5} to update our discriminator's parameters.

We discard any training patches that have an average intensity less than 0.02 or higher than 0.98. We also discard homogeneous patches that have an average gradient magnitude less than 0.06. We randomly left-right flip training patches for data augmentation.

In the adversarial training, we optimize both the main network and the discriminator in an iterative manner. At each optimization step, the learnable parameters of each network are updated to minimize its own loss function. The discriminator is trained to minimize the following loss function [143]:

$$\mathcal{L}_{\text{dsc}} = r\left(\mathbf{T}\right) + c\left(\mathbf{Y}\right), \tag{12.5}$$

where $r\left(\mathbf{T}\right)$ refers to the discriminator loss of recognizing the properly exposed reference image $\mathbf{T}$, while $c\left(\mathbf{Y}\right)$ refers to the discriminator loss of recognizing our corrected image $\mathbf{Y}$. The $r\left(\mathbf{T}\right)$ and $c\left(\mathbf{Y}\right)$ loss functions are given by the following equations:

$$r\left(\mathbf{T}\right) = -\log\left(\mathcal{S}\left(\mathcal{D}\left(\mathbf{T}\right)\right)\right), \tag{12.6}$$

$$c\left(\mathbf{Y}\right) = -\log\left(1 - \mathcal{S}\left(\mathcal{D}\left(\mathbf{Y}\right)\right)\right), \tag{12.7}$$

where $\mathcal{S}$ denotes the sigmoid function and $\mathcal{D}$ is the discriminator network described in Fig. 12.6-(B).

12.4 Ablation Studies

12.4.1 Loss Function

Our loss function includes three main terms. The first term is the standard reconstruction loss (i.e., $\mathtt{L}_1$ loss). The second and third terms consist of the pyramid and adversarial losses, respectively, which are introduced to further improve the reconstruction and perceptual quality of the output images. In the following, we discuss the effect of these loss terms.

Table 12.1: Results of our ablation study on 500 images randomly selected from our validation set. We show the effects of: (i) the pyramid loss, $\mathcal{L}_{\text{pyr}}$, and (ii) the number of Laplacian levels, n, in the main network. For each experiment, we show the values of the PSNR and SSIM [373]. The best PSNR/SSIM values are indicated with bold for each experiment.

	Pyramid loss $\mathcal{L}_{\text{pyr}}$		Number of levels n		
	w/o	w/	$n = 1$	$n = 2$	$n = 4$
PSNR	18.041	**18.385**	16.984	17.442	**18.385**
SSIM	0.746	**0.749**	0.723	0.734	**0.749**

Pyramid Loss Impact

In this ablation study, we aim to quantitatively evaluate the effect of the pyramid loss on our final results.

We train two light-weight models of our main network with and without our pyramid loss term. Each model has four 3-layer U-Nets with a total of ∼4M learnable parameters, where the number of output channels of the first encoder in each U-Net is set to 24.

The training is performed on a sub-set of our training data for ∼150,000 iterations on 80,000 128×128 patches, ∼100,000 iterations on 40,000 256×256 patches, and ∼25,000 iterations on 25,000 512×512 patches. Table 12.1 shows the results on 500 randomly selected images from our validation set. The results show that the pyramid loss not only helps in providing a better interpretation of the task of each sub-network but also improves the final results.

Adversarial Loss Impact

In Tables 12.2–12.4, we show quantitative results of our method with and without the adversarial loss term. Our trained model with the adversarial loss term achieves better perceptual quality (i.e., lower perceptual index (PI) values [56]) than training without the adversarial loss term.

Figure 12.9 shows qualitative comparisons of our results with and without the adversarial loss. As shown, the network trained without the adversarial training tends to produce darker images with slightly unrealistic colors in some cases, while the adversarial regularization improves the perceptual quality of our results.

12.4.2 Number of Laplacian Pyramid Levels

We repeat the same experimental setup described in Sec. 12.4.1 with a varying number of Laplacian pyramid levels (sub-networks). Specifically, we train a network with $n = 1$ levels—this network is equivalent to a vanilla U-Net-like architecture [323]. Additionally, we train another network with $n = 2$ (i.e., two sub-networks).

For a fair comparison, we fix the total number of parameters in each model by changing the number of filters in the conv layers. Specifically, we set the number of output channels of the first layer in the encoder to 48 for the trained model with $n = 1$, while we decrease it to 34 for the two-sub-net model (i.e., $n = 2$) to have approximately the same number of learnable parameters. Thus, the trained model in Sec. 12.4.1, used to study the pyramid loss impact, and the additional two trained models have approximately the same number of parameters.

Table 12.1 shows the results obtained by each model on the same random validation image subset used to study the pyramid loss impact in Sec. 12.4.1. Figure 12.10 shows a qualitative comparison. As can be seen, the best quantitative and qualitative results are

obtained using the four-sub-net model (i.e., $n = 4$ levels).

12.5 Empirical Evaluation

We compare our method against a broad range of existing methods for exposure correction and image enhancement. We first present quantitative results and comparisons in Sec. 12.5.1, followed by qualitative comparisons in Sec. 12.5.2.

12.5.1 Quantitative Results

To evaluate our method, we use our test set, which consists of 5,905 images rendered with different exposure settings, as described in Sec. 12.2. Specifically, our test set includes 3,543 well-exposed/overexposed images rendered with +0, +1, and +1.5 relative EVs, and 2,362 underexposed images rendered with −1 and −1.5 relative EVs.

We adopt the following three standard metrics to evaluate the pixel-wise accuracy and the perceptual quality of our results: (i) PSNR, (ii) SSIM [373], and (iii) perceptual index (`PI`) [56]. The `PI` is given by:

$$\texttt{PI} = 0.5(10 - \texttt{Ma} + \texttt{NIQE}), \tag{12.8}$$

where both `Ma` [254] and `NIQE` [269] are *no-reference* image quality metrics.

For the pixel-wise error metrics – namely, PSNR and SSIM – we compare the results not only against the properly exposed rendered images by Expert C but also with *all* five expert photographers in the MIT-Adobe FiveK dataset [62]. Though the expert photographers may render the same image in different ways due to differences in the camera-based rendering settings (e.g., white balance and tone mapping), a common characteristic over all rendered images by the expert photographers is that they all have fairly proper exposure settings [62] (see Fig. 12.7). For this reason, we evaluate our method against the *five*

expert rendered images as they all represent satisfactory exposed reference images.

We also evaluate a variety of previous non-learning and learning-based methods on our test set for comparison: histogram equalization (HE) [142], contrast-limited adaptive histogram equalization (CLAHE) [411], the weighted variational model (WVM) [124], the low-light image enhancement method (LIME) [151, 152], HDR CNN [104], DPED models [175], deep photo enhancer (DPE) models [75], the high-quality exposure correction method (HQEC) [404], RetinexNet [375], deep underexposed photo enhancer (UPE) [369], and the zero-reference deep curve estimation method (Zero-DCE) [150]. To render the reconstructed HDR images generated by the HDR CNN method [104] back into LDR, we tested both the deep reciprocating HDR transformation method (RHT) [389], and Adobe Photoshop's (PS) HDR tool [88].

Tables 12.2–12.4 summarizes the quantitative results obtained by each method. As shown in the top portion of the table, our method achieves the best results for overexposed images under all metrics. In the underexposed image correction setting, our results (middle portion of table) are on par with the state-of-the-art methods. Finally, in contrast to most of the existing methods, the results in the bottom portion of the table show that our method can effectively deal with *both* types of exposure errors.

Generalization We further evaluate the generalization ability of our method on the following standard image datasets used by previous low-light image enhancement methods: (i) LIME (10 images) [152], (ii) NPE (75 images) [370], (iii) VV (24 images) [366], and DICM (44 images) [222]. Note that in these experiments, we report results of our model trained on our training set without further tuning or re-training on any of these datasets. Similar to previous methods, we use the `NIQE` perceptual score [269] for evaluation. Table 12.5 compares results by our method and the following methods: LIME [151, 152], WVM [124], RetinexNet (RNet) [375], "kindling the darkness" (KinD) [407], enlighten

GAN (EGAN) [183], and deep bright-channel prior (BCP) [224]. As can be seen in Table 12.5, our method generally achieves perceptually superior results in correcting low-light 8-bit images of other datasets.

12.5.2 Qualitative Results

We compare our method qualitatively with a variety of previous methods. Note we show results using the model trained with the adversarial loss term, as it produces perceptually superior results (see the perceptual metric results in Tables 12.2–12.4).

Figure 12.8 shows our results on different overexposed and underexposed images. As shown, our results are arguably visually superior to the other methods, even when input images have hard backlight conditions, as shown in the second row in Fig. 12.8 (right).

Generalization We also ran our model on several images from Flickr that are outside our introduced dataset, as shown in Figs. 12.1 and 12.11. As with the images from our introduced dataset, our results on the Flickr images are arguably superior to the compared methods. Additional qualitative comparisons using images taken from Flickr are shown in Fig. 12.12.

12.5.3 Limitations

Our method produces unsatisfactory results in regions that have insufficient semantic information, as shown in Fig. 12.13. For example, the input image shown in the first row in Fig. 12.13 is completely saturated and contains almost no details in the region of the man's face. We can see that our network cannot constrain the color inside the face region due to the lack of semantic information. In that way, one can control the output results to reduce such color bleeding problems. It also can be observed that our method may introduce noise when the input image has extreme dark regions, as shown in the second

example in Fig. 12.13. These challenging conditions prove difficult for other methods as well.

12.6 Potential Applications

In this section, we highlight two potential applications of our method: (i) photo editing and (ii) image preprocessing.

Photo Editing The main potential application of the proposed method is to post-capture correct exposure errors in images. This correction process can be performed in a fully automated way or can be performed in an interactive way with the user. Specifically, we introduce a scale vector $\mathbf{S} = [S_1, S_2, S_3, S_4]^\top$ that can be used to independently scale each level in the pyramid $\mathbf{X}$ in the inference stage. The scale vector $\mathbf{S}$ is introduced to produce different visual effects in the final result $\mathbf{Y}$. In particular, this scaling operation is performed as a pre-processing of each level in the pyramid $\mathbf{X}$ as follows: $\mathbf{S}_{(l=i)}\mathbf{X}_{(l=i)}$, s.t. $i \in \{1, 2, 3, 4\}$. The values of the scale vector $\mathbf{S}$ can be interactively controlled by the user to edit our network results. Figure 12.14 shows different results obtained by our network in an interactive way through our graphical user interface (GUI). Our GUI can be used as a photo editing tool to apply different visual effects and filters on the input images.

Image Preprocessing Our method can also improve the results of computer vision tasks by using it as a pre-processing step to correct exposure errors in input images. Figure 12.15 shows example applications. In these examples, we show results of face and facial landmark detection of the work in [400] and image semantic segmentation results obtained by the work in [236, 237]. As shown, the results of face detection and semantic segmentation are improved by pre-processing the input images using our method. In future work, we plan to

investigate the impact of our exposure correction method on a variety of computer vision tasks.

12.7 Summary

We proposed a single coarse-to-fine deep learning model for overexposed and underexposed image correction. We employed the Laplacian pyramid decomposition to process input images in different frequency bands. Our method is designed to sequentially correct each of the Laplacian pyramid levels in a multi-scale manner, starting with the global color in the image and progressively addressing the image details.

Our method is enabled by a new dataset of over 24,000 images rendered with the broadest range of exposure errors to date. Each image in our introduced dataset has a reference image properly rendered by a well-trained photographer with well-exposure compensation. Through extensive evaluation, we showed that our method produces compelling results compared to available solutions for correcting images rendered with exposure errors and it generalizes well. We believe that our dataset will help future work on improving exposure correction for photographs.

Figure 12.9: Comparisons between our results with (w/) and without (w/o) the adversarial loss for training. The PSNR, structural similarity index measure (SSIM) [373], and perceptual index (PI) [56] are shown for each result. Notice that higher PSNR and SSIM values are better, while lower PI values indicate better perceptual quality. The input images are taken from our test set.

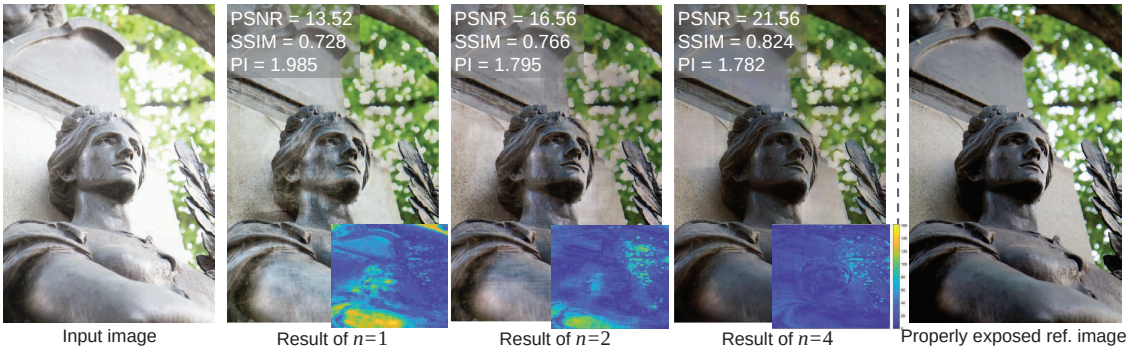


Figure 12.10: Comparison of results by varying the number of Laplacian pyramid levels. Notice that higher PSNR and SSIM values are better, while lower PI values indicate better perceptual quality. The input image is taken from our validation set.

Figure 12.11: Comparisons with commercial software packages. The input images are taken from Flickr.

Figure 12.12: Comparison with the recent Zero-DCE method [150] using images taken from Flickr.

Table 12.2: Quantitative evaluation on our introduced over-exposure test set. **The best results are highlighted with green and bold. The second- and third-best results are highlighted in yellow and red, respectively.** We denote methods designed for underexposure correction in gray. Non-deep learning methods are marked by $*$.

Method	Expert A		Expert B		Expert C		Expert D		Expert E		Avg.		PI ↓
	PSNR ↑	SSIM ↑	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	
+0, +1, and +1.5 relative EVs (3,543 properly exposed and overexposed images)													
HE [142] $*$	16.140	0.686	16.277	0.672	16.531	0.699	16.643	0.669	17.321	0.691	16.582	0.683	2.351
CLAHE [411] $*$	13.934	0.568	14.689	0.586	14.453	0.584	15.116	0.593	15.850	0.612	14.808	0.589	2.270
WVM [124] $*$	12.355	0.624	13.147	0.656	12.748	0.645	14.059	0.669	15.207	0.690	13.503	0.657	2.342
LIME [151, 152] $*$	09.627	0.549	10.096	0.569	9.875	0.570	10.936	0.597	11.903	0.626	10.487	0.582	2.412
HDR CNN [104] w/ RHT [389]	13.151	0.475	13.637	0.478	13.622	0.497	14.177	0.479	14.625	0.503	13.842	0.486	4.284
HDR CNN [104] w/ PS [88]	14.804	0.651	15.622	0.689	15.348	0.670	16.583	0.685	18.022	0.703	16.076	0.680	2.248
DPED (iPhone) [175]	12.680	0.562	13.422	0.586	13.135	0.581	14.477	0.596	15.702	0.630	13.883	0.591	2.909
DPED (BlackBerry) [175]	15.170	0.621	16.193	0.691	15.781	0.642	17.042	0.677	18.035	0.678	16.444	0.662	2.518
DPED (Sony) [175]	16.398	0.672	17.679	0.707	17.378	0.697	17.997	0.685	18.685	0.700	17.627	0.692	2.740
DPE (HDR) [75]	14.399	0.572	15.219	0.573	15.091	0.593	15.692	0.581	16.640	0.626	15.408	0.589	2.417
DPE (U-FiveK) [75]	14.314	0.615	14.958	0.628	15.075	0.645	15.987	0.647	16.931	0.667	15.453	0.640	2.630
DPE (S-FiveK) [75]	14.786	0.638	15.519	0.649	15.625	0.668	16.586	0.664	17.661	0.684	16.035	0.661	2.621
HQEC [404] $*$	11.775	0.607	12.536	0.631	12.127	0.627	13.424	0.652	14.511	0.675	12.875	0.638	2.387
RetinexNet [375]	10.149	0.570	10.880	0.586	10.471	0.595	11.498	0.613	12.295	0.635	11.059	0.600	2.933
Deep UPE [369]	10.047	0.532	10.462	0.568	10.307	0.557	11.583	0.591	12.639	0.619	11.008	0.573	2.428
Zero-DCE [150]	10.116	0.503	10.767	0.502	10.395	0.514	11.471	0.522	12.354	0.557	11.0206	0.5196	2.774
Our method w/o $\mathcal{L}_{\text{adv}}$	**18.976**	**0.743**	**19.767**	**0.731**	**19.980**	**0.768**	**18.966**	**0.716**	**19.056**	**0.727**	**19.349**	**0.737**	2.189
Our method w/ $\mathcal{L}_{\text{adv}}$	18.874	0.738	19.569	0.718	19.788	0.760	18.823	0.705	18.936	0.719	19.198	0.728	**2.183**

Table 12.3: Quantitative evaluation on our introduced under-exposure test set set. **The best results are highlighted with green and bold. The second- and third-best results are highlighted in yellow and red, respectively.** We denote methods designed for underexposure correction in gray. Non-deep learning methods are marked by $*$.

Method	Expert A		Expert B		Expert C		Expert D		Expert E		Avg.		PI ↓
	PSNR ↑	SSIM ↑	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	
−1 and −1.5 relative EVs (2,362 underexposed images)													
HE [142] $*$	16.158	0.683	16.293	0.669	16.517	0.692	16.632	0.665	17.280	0.684	16.576	0.679	2.486
CLAHE [411] $*$	16.310	0.619	17.140	0.646	16.779	0.621	15.955	0.613	15.568	0.608	16.350	0.621	2.387
WVM [124] $*$	17.686	0.728	19.787	**0.764**	18.670	0.728	18.568	0.729	18.362	0.724	18.615	0.735	2.525
LIME [151, 152] $*$	13.444	0.653	14.426	0.672	13.980	0.663	15.190	0.673	16.177	0.694	14.643	0.671	2.462
HDR CNN [104] w/ RHT [389]	14.547	0.456	14.347	0.427	14.068	0.441	13.025	0.398	11.957	0.379	13.589	0.420	5.072
HDR CNN [104] w/ PS [88]	17.324	0.692	18.992	0.714	18.047	0.696	18.377	0.689	19.593	0.701	18.467	0.698	**2.294**
DPED (iPhone) [175]	18.814	0.680	21.129	0.712	20.064	0.683	**19.711**	0.675	19.574	0.676	19.858	0.685	2.894
DPED (BlackBerry) [175]	19.519	0.673	**22.333**	0.745	20.342	0.669	19.611	0.683	18.489	0.653	**20.059**	0.685	2.633
DPED (Sony) [175]	18.952	0.679	20.072	0.691	18.982	0.662	17.450	0.629	15.857	0.601	18.263	0.652	2.905
DPE (HDR) [75]	17.625	0.675	18.542	0.705	18.127	0.677	16.831	0.665	15.891	0.643	17.403	0.673	2.340
DPE (U-FiveK) [75]	19.130	0.709	19.574	0.674	19.479	0.711	17.924	0.665	16.370	0.625	18.495	0.677	2.571
DPE (S-FiveK) [75]	**20.153**	0.738	20.973	0.697	**20.915**	0.738	19.050	0.688	17.510	0.648	19.720	0.702	2.564
HQEC [404] $*$	15.801	0.692	17.371	0.718	16.587	0.700	17.090	0.705	17.675	0.716	16.905	0.706	2.532
RetinexNet [375]	11.676	0.607	12.711	0.611	12.132	0.621	12.720	0.618	13.233	0.637	12.494	0.619	3.362
Deep UPE [369]	17.832	0.728	19.059	0.754	18.763	0.745	19.641	**0.737**	**20.237**	**0.740**	19.106	0.741	2.371
Zero-DCE [150]	13.935	0.585	15.239	0.593	14.552	0.589	15.202	0.587	15.893	0.614	14.9642	0.5936	3.001
Our method w/o $\mathcal{L}_{\text{adv}}$	19.432	0.750	20.590	0.739	20.542	**0.770**	18.989	0.723	18.874	0.727	19.685	**0.742**	2.344
Our method w/ $\mathcal{L}_{\text{adv}}$	19.475	**0.751**	20.546	0.730	20.518	0.768	18.935	0.715	18.756	0.719	19.646	0.737	2.342

Figure 12.13: Failure examples of correcting (top) overexposed and (bottom) underexposed images. The input images are taken from Flickr.

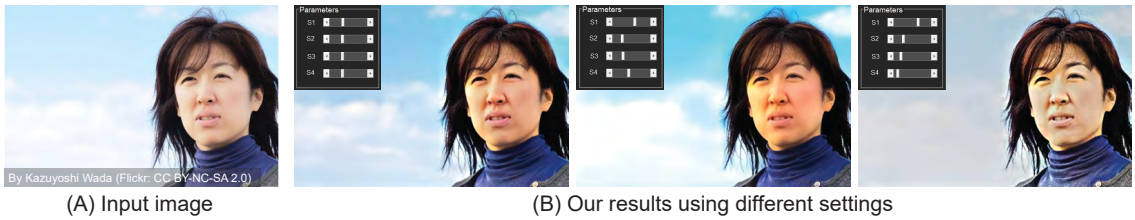
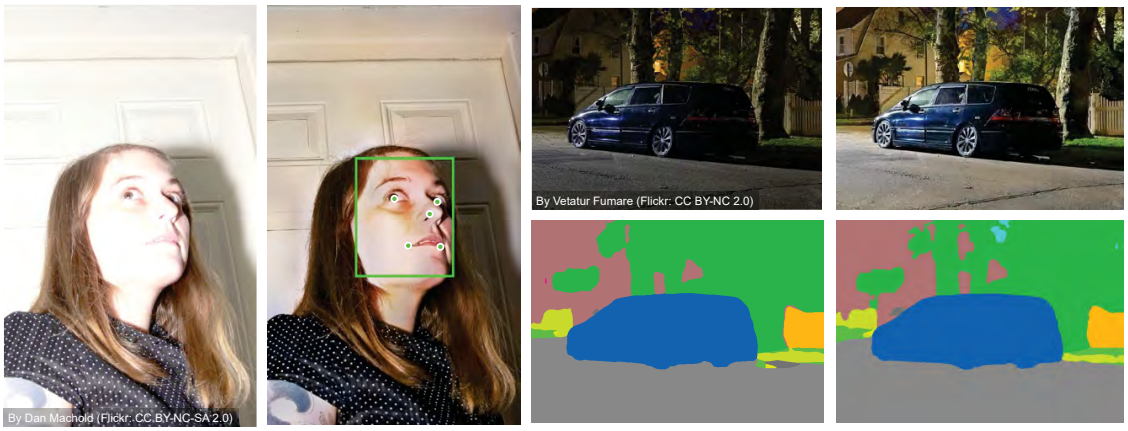


Figure 12.14: Our GUI photo editing tool. (A) Input image. (B) Our results using different pyramid level scaling settings set by the user in an interactive way. The input image is taken from Flickr.

Table 12.4: Quantitative evaluation on our introduced test set (both under- and over-exposed images). **The best results are highlighted with green and bold. The second- and third-best results are highlighted in yellow and red, respectively.** We denote methods designed for underexposure correction in gray. Non-deep learning methods are marked by $*$.

Method	Expert A		Expert B		Expert C		Expert D		Expert E		Avg.		PI ↓
	PSNR ↑	SSIM ↑	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	
Combined over and underexposed images (5,905 images)													
HE [142] $*$	16.148	0.685	16.283	0.671	16.525	0.696	16.639	0.668	17.305	0.688	16.580	0.682	2.405
CLAHE [411] $*$	14.884	0.589	15.669	0.610	15.383	0.599	15.452	0.601	15.737	0.610	15.425	0.602	2.317
WVM [124] $*$	14.488	0.665	15.803	0.699	15.117	0.678	15.863	0.693	16.469	0.704	15.548	0.688	2.415
LIME [151, 152]	11.154	0.591	11.828	0.610	11.517	0.607	12.638	0.628	13.613	0.653	12.150	0.618	2.432
HDR CNN [104] w/ RHT [389]	13.709	0.467	13.921	0.458	13.800	0.474	13.716	0.446	13.558	0.454	13.741	0.460	4.599
HDR CNN [104] w/ PS [88]	15.812	0.667	16.970	0.699	16.428	0.681	17.301	0.687	18.650	0.702	17.032	0.687	2.267
DPED (iPhone) [175]	15.134	0.609	16.505	0.636	15.907	0.622	16.571	0.627	17.251	0.649	16.274	0.629	2.903
DPED (BlackBerry) [175]	16.910	0.642	18.649	0.713	17.606	0.653	18.070	0.679	18.217	0.668	17.890	0.671	2.564
DPED (Sony) [175]	17.419	0.675	18.636	0.701	18.020	0.683	17.554	0.660	17.778	0.663	17.881	0.676	2.806
DPE (HDR) [75]	15.690	0.614	16.548	0.626	16.305	0.626	16.147	0.615	16.341	0.633	16.206	0.623	2.417
DPE (U-FiveK) [75]	16.240	0.653	16.805	0.646	16.837	0.671	16.762	0.654	16.707	0.650	16.670	0.655	2.606
DPE (S-FiveK) [75]	16.933	0.678	17.701	0.668	17.741	0.696	17.572	0.674	17.601	0.670	17.510	0.677	2.621
HQEC [404] $*$	13.385	0.641	14.470	0.666	13.911	0.656	14.891	0.674	15.777	0.692	14.487	0.666	2.445
RetinexNet [375]	10.759	0.585	11.613	0.596	11.135	0.605	11.987	0.615	12.671	0.636	11.633	0.607	3.105
Deep UPE [369]	13.161	0.610	13.901	0.642	13.689	0.632	14.806	0.649	15.678	0.667	14.247	0.640	2.405
Zero-DCE [150]	11.643	0.536	12.555	0.539	12.058	0.544	12.964	0.548	13.769	0.580	12.5978	0.5494	2.865
Our method w/o $\mathcal{L}_{\text{adv}}$	**19.158**	**0.746**	**20.096**	**0.734**	**20.205**	**0.769**	**18.975**	**0.719**	**18.983**	**0.727**	**19.483**	**0.739**	2.251
Our method w/ $\mathcal{L}_{\text{adv}}$	19.114	0.743	19.960	0.723	20.080	0.763	18.868	0.709	18.864	0.719	19.377	0.731	**2.247**



(A) Failure case of face and facial landmark detection

(B) Face and facial landmark detection after our correction

(C) Semantic segmentation result on original image

(D) Semantic segmentation result on our corrected image

Figure 12.15: Applying our method as a pre-processing step can improve results of different computer vision tasks. (A) False negative result of face and facial landmark detection due to the overexposure error in the input image. (B) Our corrected image and the results of face and facial landmark detection. (C) Underexposed input image and its semantic segmentation mask. (D) Our corrected image and its semantic segmentation mask. We use the cascaded convolutional networks proposed in [400] for face and facial landmark detection. For image semantic segmentation, we use RefineNet [236, 237]. The input images are taken from Flickr.

Table 12.5: Perceptual quality evaluation. Summary of `NIQE` scores [269] on different *low-light* image datasets. In these dataset, there are no ground-truth images provided for full-reference quality metrics (e.g., PSNR). Highlights are in the same format as Table 12.2

Method	LIME [152]	NPE [370]	VV [366]	DICM [222]	Avg.
NPE [370] $*$	3.91	3.95	2.52	3.76	3.54
LIME [152] $*$	4.16	4.26	2.49	3.85	3.69
WVM [124] $*$	3.79	3.99	2.85	3.90	3.63
RNet [375]	4.42	4.49	2.60	4.20	3.93
KinD [407]	**3.72**	3.88	-	-	3.80
EGAN [183]	**3.72**	4.11	2.58	-	3.50
DBCP [224]	3.78	**3.18**	-	3.57	3.48
Ours w/o $\mathcal{L}_{\text{adv}}$	3.76	3.20	**2.28**	2.55	2.95
Ours w/ $\mathcal{L}_{\text{adv}}$	3.76	**3.18**	**2.28**	**2.50**	**2.93**

Part VI

Image Recoloring

13 Recoloring Based on Object Color Distributions

This part of the thesis focuses on an auto color editing, where our goal is to manipulate an image's RGB color values to produce a new appearance that conveys a different "look and feel" of the image. This procedure of manipulating an image's color in this manner is often referred to as *image recoloring*. We are interested in achieving realistic auto recolor images with minimal user interaction. To that end, we will outline two methods for auto image recoloring.

In this chapter, we present a method to perform automatic image recoloring based on the distribution of colors associated with objects present in an image[1]. For example, when recoloring an image containing a sky object, our method incorporates the observation that objects of class 'sky' have a color distribution with three dominant modes for blue (daytime), yellow/red (dusk/dawn), and dark (nighttime). Our work leverages recent deep-learning methods that can perform reasonably accurate object-level segmentation. By using the images in datasets used to train deep-learning object segmentation methods, we are able to model the color distribution of each object class in the dataset. Given a new input image and its associated semantic segmentation (i.e., object mask), we perform color

[1]This work was published in [21]: Mahmoud Afifi, Brian Price, Scott Cohen, and Michael S. Brown. Image Recoloring Based on Object Color Distributions. In Eurographics – Short Paper, 2019.

Figure 13.1: (A) An input image and its semantic segmentation (object mask) obtained by RefineNet [237]. (B) Recolored images produced by Photoshop's variation tool. (C) Recolored images from our method that considers the color distribution of objects in the image.

transfer to map the input image color histogram to a set of target color histograms that were constructed based on the learned color distribution of the objects in the image. We show that our framework is able to produce compelling color variations that are often more interesting and unique than results produced by existing methods. The source code of this work is available on GitHub: https://github.com/mahmoudnafifi/Image_recoloring.

13.1 Introduction

Image recoloring aims at transferring the colors of a given input image to share the same colors and "feel" with some target colors. Color manipulation for this purpose is achieved in different ways, such as color transfer (e.g., [108]), appearance transfer (e.g., [216]), and style transfer (e.g., [250]). Image editing software, such as Photoshop, provides tools for automatic image recoloring as a way to provide users with interesting variations on an input image. Figure 13.1 shows a typical case of image recoloring, where an input image is manipulated automatically to produce several recolored variations.

The vast majority of existing recoloring methods require a target image that is specified

by the user (e.g., [108,130,250]). There are several methods that perform automatic transfer (e.g., [172,226]). To the best of our knowledge, these existing methods do not explicitly consider the objects present in the image in the recoloring process. Such object-level semantic information can be useful in guiding the recoloring effort to produce interesting and plausible variations on the input image. For example, if the input image has a *sky object*, the recolored image should exploit observations from the training data that a sky object can be blue, but not green. Moreover, if the original input image already has a sky object with a blue color appearance, we can use a dissimilar color associated with the sky object class (e.g., a reddish appearance) to provide more variety in the recolored results.

Contribution We propose a data-driven framework to automatically recolor an input image that incorporates information about the color distributions of objects present in the image. Specifically, we show how to model the color distributions of different object classes from thousands of images with labeled objects. Given a new input image and its object segmentation, we outline a procedure to produce a diverse set of recolored images. We show that our results produce compelling examples that provide more interesting variations than existing methods.

13.2 Methodology

Figure 13.2 shows a diagram of our procedure. We start by describing the data preparation followed by the details of our method.

13.2.1 Training Data

Our method requires a large source of images with labeled object masks in order to build a color distribution for each class (type) of objects. To this end, we use the MIT Scene

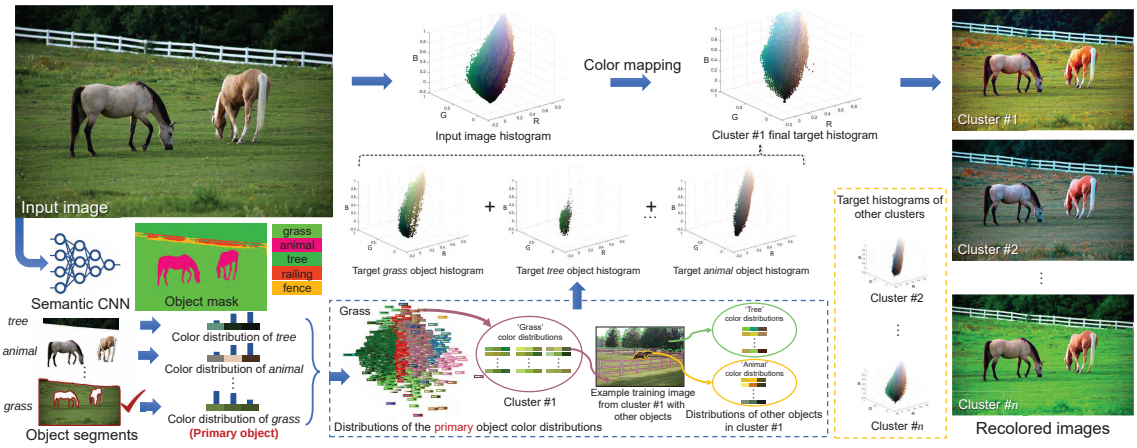


Figure 13.2: Our image recoloring framework. We use a deep-learning method to obtain the input image's semantic segmentation as an object mask. We generate a color distribution for each object as a color palette and select one object in the image as the primary object. Next, we visit each cluster in the training images of the primary object class. For each cluster, we select an object instance with the most *dissimilar* colors to our input object's colors and add its colors to our target histogram. Within this same cluster, we search for the other (non-primary) objects found in the input image and use the most dissimilar instance in the target histogram. Lastly, color transfer is applied to map the input image to our generated target histogram. This is repeated for each cluster, producing several variations on the input image. The term w_{obj} refers to each object's pixel ratio in the input image.

Parsing Benchmark (SPB) [409], which contains 20,210 images with pixel masks for 150 different object classes.

For each object class b, we extract the RGB pixel values for all object instances for this class in the training images. The object class is represented as a set of color distributions $\mathcal{C}_b = \{\mathbf{c}_b^{(m)}\}_{m=1}^{M}$, where M is the number of training images containing an instance of the object of class b. An individual object instance's color distribution $\mathbf{c}_b^{(i)}$ is modeled as a

color palette, with k colors, generated by the method proposed in [70]. The color palette representation also maintains the ratio of the number of pixels associated with each color in the palette.

We use the set $\mathcal{C}_b$ to build a "distribution of color distributions" (DoD) for each object class b. The DoD of an object b is generated by calculating the earth mover's distance (EMD) [326] between each pair of color palettes $\mathbf{c}_b^{(i)}$ and $\{\mathbf{c}_b^{(j)}\}_{j=1}^{M}$ of object instances, where $i = [1, ..., M]$ and $j \neq i$. Based on the EMDs between all color distributions in $\mathcal{C}_b$, we generate n clusters of the color distributions using an agglomerative (bottom-up) hierarchical clustering with Ward's minimum variance [374].

By clustering all individual objects in the training images of object class b based on their color palettes, each object instance associated with a cluster shares a similar overall color appearance. Figure 13.3 shows a visualization of this clustering procedure. Figure 13.4 shows more examples of the DoD of different object classes to provide an idea the different color appearances present in te DoDs. Within each cluster, we maintain a list of all other objects that appeared in the training images. This latter point is important as it gives us a way to find other objects that have appeared in the images for a particular object class b.

13.2.2 Recoloring Procedure

Given an input image $\mathbf{I}$, we compute its semantic object mask $\mathbf{M}$ using RefineNet [237]. Note that this mask can be noisy (i.e., RefineNet reports ~ 0.79 pixel-wise accuracy rate on the SPB dataset). The objects in $\mathbf{I}$ are ranked based on each object's pixel ratio in $\mathbf{I}$ (w_{obj}) and the ratio of the training samples with that object class (r_{obj}) in the training dataset. Both of these terms are normalized to range between 0 and 1. We select the primary object p as the object with the maximum score $w_{\text{obj}_p} + r_{\text{obj}_p}$.

For the primary object and all other objects in **I**, we compute their color distributions (i.e., a color palette with associated pixel weights). We then visit each cluster of the primary object's class as described in Sec. 13.2.1. Within a cluster, we find the most dissimilar object instance based on the EMD of the primary object's color palette and each example in the cluster. The colors of this dissimilar object instance are added to the target histogram. Within the same cluster, this procedure is repeated for all other objects present in the image. Again, we seek the most dissimilar color palette to the input image's object within the cluster using the EMD metric. Figure 13.5 provides an illustrative example. These dissimilar colors from the training data are added to the target histogram. If an object found in the input image does not exist in a cluster, we copy the input object's colors to the target histogram. Note that the construction of the target histogram was performed by a weighted summation based on each object's pixel ratio in **I**. We have purposely chosen to use objects with the most dissimilar colors to provide notably different recolored images; however, we note the criteria for selecting target object instances can be adjusted to employ different strategies.

Once a target histogram has been constructed, we map the input histogram to the target histogram. This mapping produces the recolored result. In our experiments, we used the method by Pitie and Kokaram [304] to transfer the input image's color histogram to the target color histogram. Then, we scale any out-of-gamut pixels to fall in the range [0-255].

The procedure described above is repeated for each cluster in the primary object's class, producing n output images. Alg. 1 provides pseudocode for our procedure.

13.3 Results

In our experiments, the number of clusters used in the DoD is set to $n = 20$, the input and target color histogram bins are resized to resolution $32 \times 32 \times 32$, and our color palettes are fixed to have $k = 20$ colors. We show color palettes with three colors in Fig. 13.2 and Fig. 13.3 to simplify the visualization.

Figure 13.6 shows comparisons between results of our method and three other methods. The first method [216] was proposed to transfer the appearance of outdoor images to different scene appearances specified by a set of attributes selected by the user. The second method [226] is an auto color/style transfer method which employs deep features extracted from a pre-trained CNN on the ImageNet dataset [91] in order to produce content-aware color stylization. The third method [250] is a CNN-based method which requires a reference image in order to transfer its style to the input image. The results show that our method produces compelling results without requiring any user interaction or reference images. Additional qualitative results are shown in Figs. 13.7 and 13.8.

13.4 Summary

We have proposed an automated data-driven method to generate recolored images. Our method leverages semantic object information of the input image and these objects' associated color distributions that have been modeled from thousands of training images. We demonstrate the effectiveness of the proposed method on several input images and compare our results with other strategies to produce color variations.

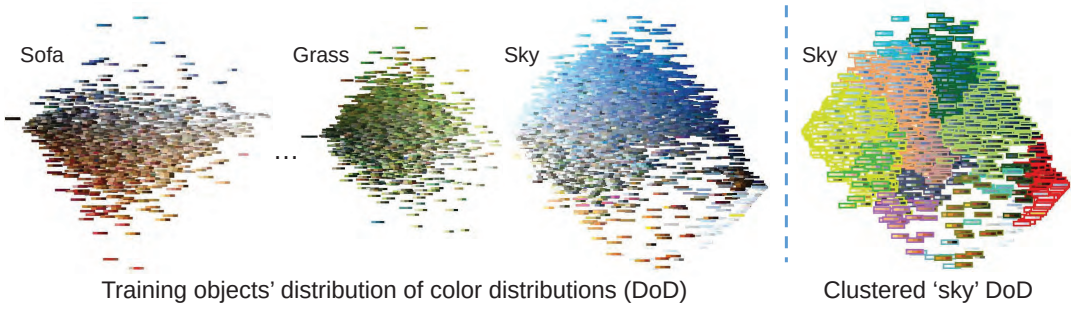


Figure 13.3: For each object class in our training data, we extract each instance of an object of that class that appears in the training images and represent its color appearance as a distribution of k colors in the form of a color palette. We then construct a distribution of color distributions for each object class, termed a DoD (distribution of color distributions). The DoD is computed by clustering the color palettes into n clusters.

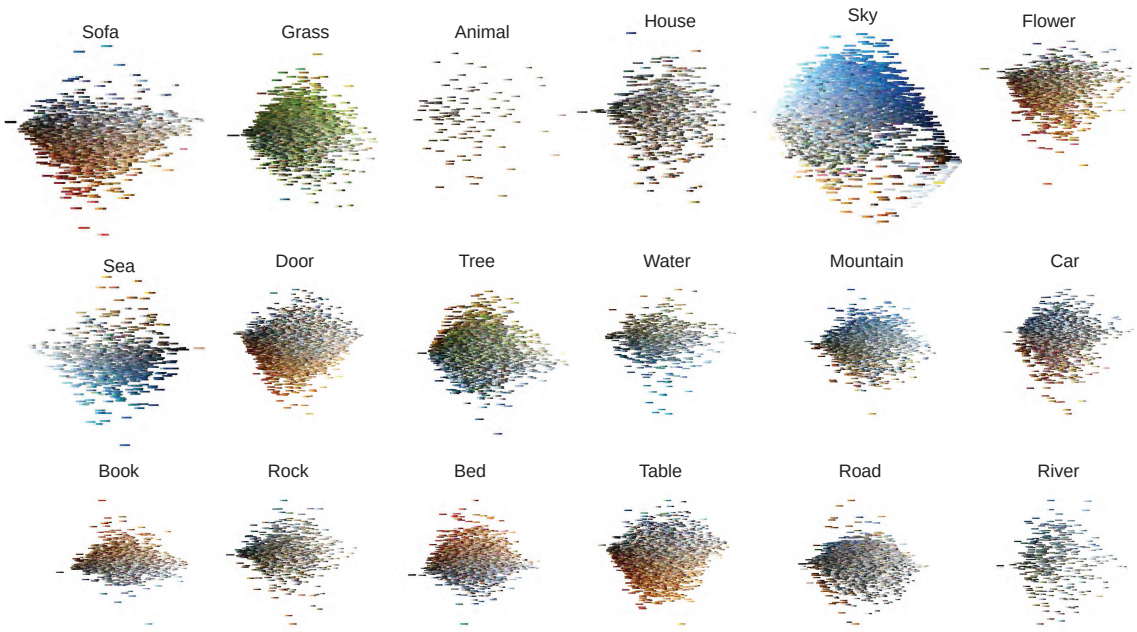


Figure 13.4: Examples of the distribution of color distributions (DoD) of different object classes.

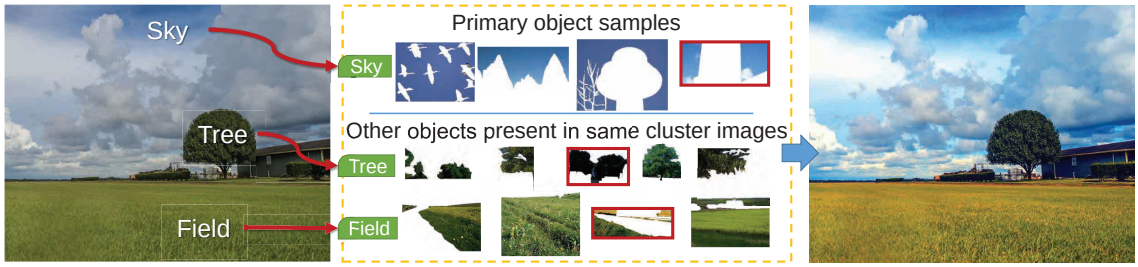


Figure 13.5: For each cluster of the primary object, we search for the most dissimilar training samples (highlighted with red borders) of the input image's semantic objects.

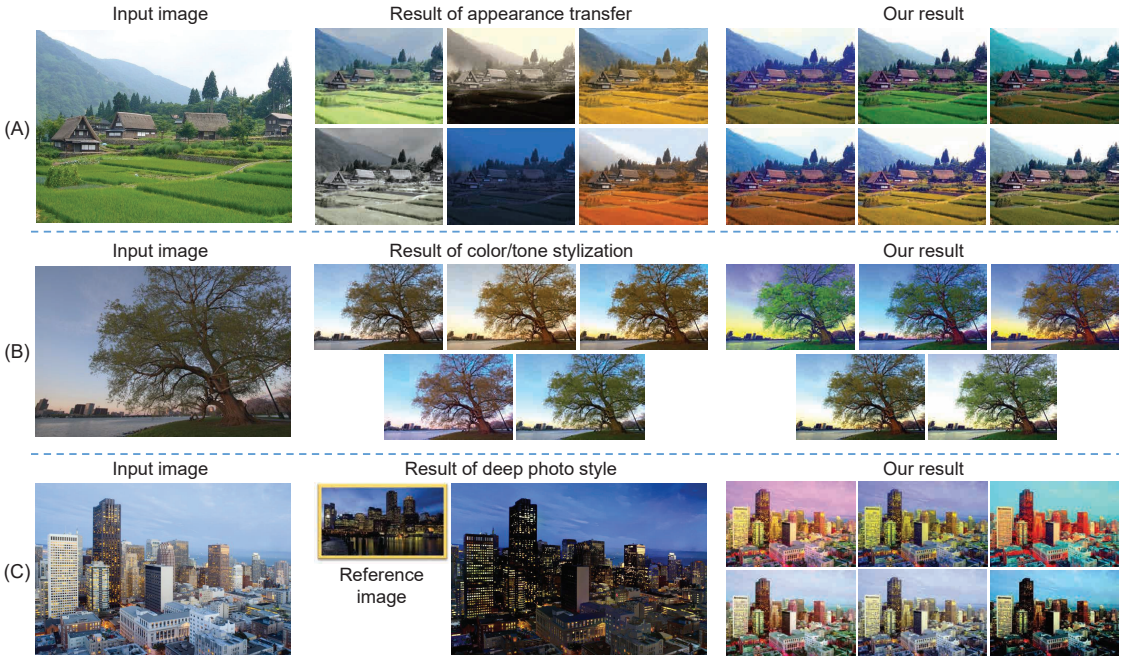


Figure 13.6: Comparisons with existing style transfer methods. For each input image (first column), we show results of other methods (second column) and our results (third column). The other methods are: (A) appearance transfer [216], (B) auto content-aware color and tone stylization [226], and (C) a reference-based deep style transfer [250].

Algorithm 1 Given an input image $\mathbf{I}$ and its object mask $\mathbf{M}$, we generate n recolored images $\{\mathbf{R}_i\}_{i=1}^{n}$.

$p \leftarrow$ select the primary object

for each object j in $\mathbf{M}$, **do**:

$\mathbf{c}_{\text{obj}_j}^{\text{input}} \leftarrow$ get color palette (CP) of object j in $\mathbf{I}$

end

for each training cluster i of primary object p, **do**:

$\mathbf{H}^{(i)} \leftarrow \{\}$ `//Initialize empty target histogram`

for each object j in $\mathbf{M}$, **do**:

if $\mathcal{C}_{\text{obj}_j}^{\text{clust}(i)}$is emtpy, **then**

$\mathbf{S}_{\text{obj}_j} \leftarrow \mathbf{I}_{\text{obj}_j}$ `//Original input object pixels`

else

$\mathbf{d}_{\text{obj}_j} \leftarrow$ calculate EMD between $\mathbf{c}_{\text{obj}_j}^{\text{input}}$ and CPs in $\mathcal{C}_{\text{obj}_j}^{\text{clust}(i)}$

$\mathbf{S}_{\text{obj}_j} \leftarrow$ retrieve training sample with max value in $\mathbf{d}_{\text{obj}_j}$

end

$\mathbf{H}_{\text{obj}_j}^{(i)} \leftarrow$ generate color histogram of $\mathbf{S}_{\text{obj}_j}$.

$\mathbf{H}^{(i)} \leftarrow$ add $\mathbf{H}_{\text{obj}_j}^{(i)}$ to $\mathbf{H}^{(i)}$.

end

$\mathbf{R}_i \leftarrow$ color mapping $\left(\mathbf{H}^{\text{input}}, \mathbf{H}^{(i)}\right)$

end

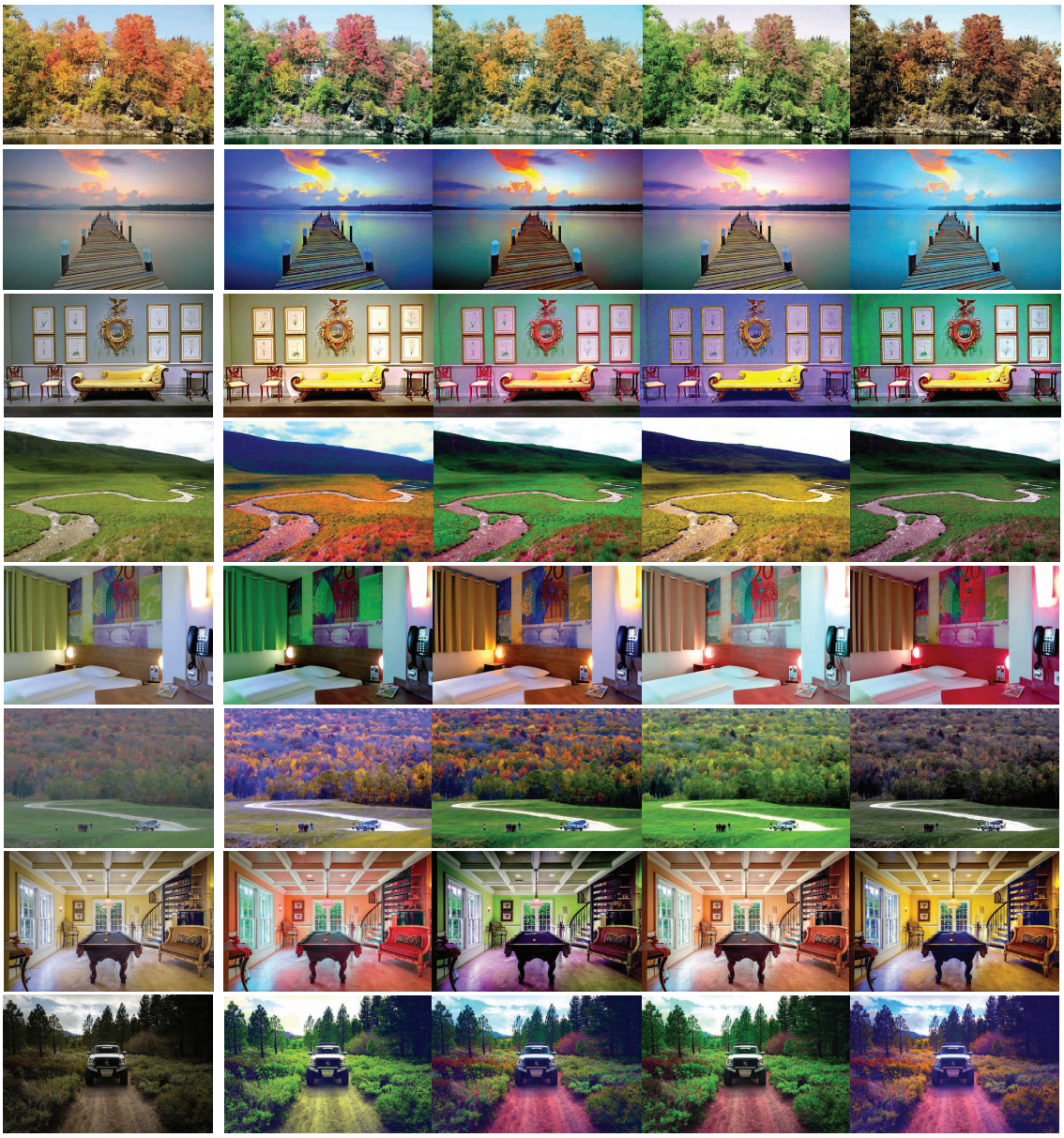


Figure 13.7: Qualitative results of the proposed method. (A) Input image. (B) Four examples of our recolored images.

Figure 13.8: Additional qualitative results of the proposed method. (A) Input image. (B) Four examples of our recolored images.

14 Controlling Colors via Color Histograms

In Chapter 13, we have presented an auto recoloring framework based on object color distribution. In this chapter, we propose a generative adversarial network (GAN)-based method for image recoloring. Our method is not only designed to recolor real images but also to control colors of GAN-generated images. While GANs can successfully produce high-quality images, they can be challenging to control. Simplifying GAN-based image generation is critical for their adoption in graphic design and artistic work. This goal has led to significant interest in methods that can intuitively control the appearance of images generated by GANs. In this chapter, we present HistoGAN[1], a color histogram-based method for controlling GAN-generated images' colors. We focus on color histograms as they provide an intuitive way to describe image color while remaining decoupled from domain-specific semantics. Specifically, we introduce an effective modification of the recent StyleGAN architecture [197] to control the colors of GAN-generated images specified by a target color histogram feature. We then describe how to expand HistoGAN to recolor real

[1]This work is under review and a preprint version is available in [17]: Mahmoud Afifi, Marcus A. Brubaker, Michael S. Brown. HistoGAN: Controlling Colors of GAN-Generated and Real Images via Color Histograms. To appear In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2021.

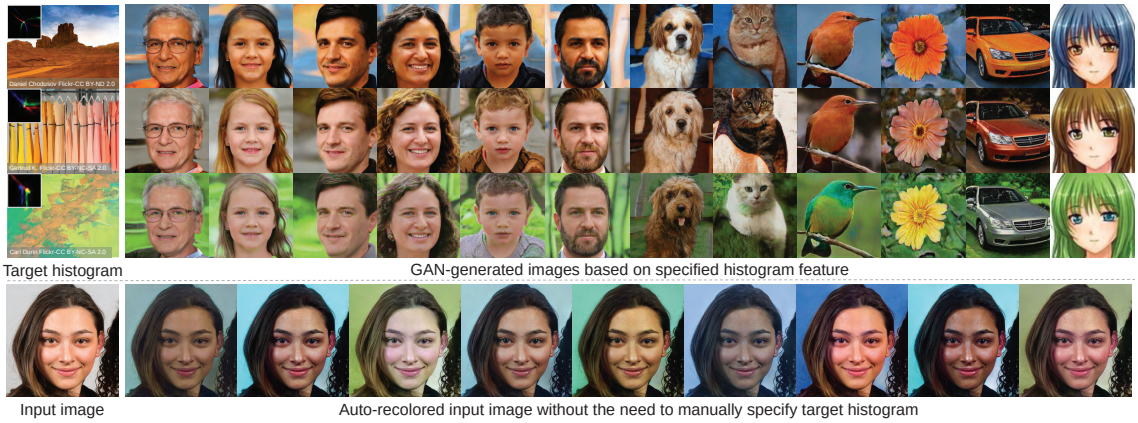


Figure 14.1: HistoGAN is a generative adversarial network (GAN) that learns to manipulate image colors based on histogram features. Top: GAN-generated images with color distributions controlled via target histogram features (left column). Bottom: Results of ReHistoGAN, an extension of HistoGAN to recolor real images, using sampled target histograms.

images. For image recoloring, we jointly train an encoder network along with HistoGAN. The recoloring model, ReHistoGAN, is an unsupervised approach trained to encourage the network to keep the original image's content while changing the colors based on the given target histogram. We demonstrate that this histogram-based approach offers a better way to control GAN-generated and real images' colors while producing more compelling results compared to existing alternative strategies. The source code and dataset of this work are available on GitHub: https://github.com/mahmoudnafifi/HistoGAN.

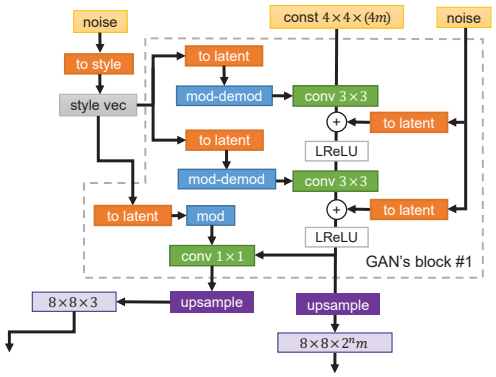
14.1 Introduction

Color histograms are an expressive and convenient representation of an image's color content. Color histograms are routinely used by conventional color transfer methods (e.g., [108, 287, 318, 381]). These color transfer methods aim to manipulate the colors

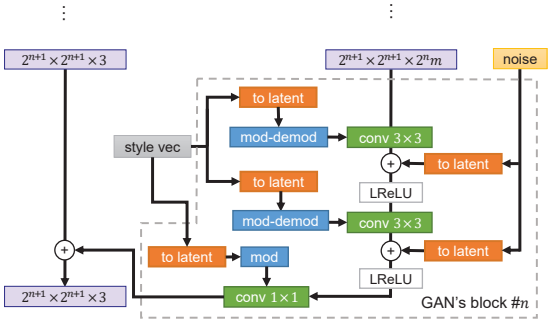
in an input image to match those of a target image, such that the images share a similar "look and feel". As discussed in Chapter 3 there are various forms of color histograms used in the color transfer literature to represent the color distribution of an image, such as a direct 3D histogram [108,318,381], color palettes [70,403] or color triads [341]. Despite the effectiveness of color histograms for color transfer, recent deep learning methods almost exclusively rely on image-based examples to control colors. While image exemplars impact the final colors of generative adversarial network (GAN)-generated images and deep recolored images, they also affect other style attributes, such as texture information and tonal values [129, 130, 179, 186, 251, 337, 360]. Consequently, the quality of the results produced by these methods often depends on the semantic similarity between the input and target images, or between a target image and a particular domain [160, 337].

In this chapter, our attention is focused explicitly on controlling only the color attributes of images—this can be considered a sub-category of image style transfer. Specifically, our method does not require shared semantic content between the input/GAN-generated images and a target image or guide image. Instead, our method aims to assist the deep network through color histogram information only. With this motivation, we first explore using color histograms to control the colors of images generated by GANs.

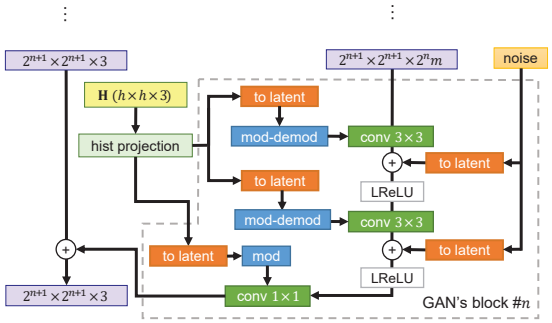
Controlling Color in GAN-Generated Images GANs are often used as "black boxes" that can transform samples from a simple distribution to a meaningful domain distribution without an explicit ability to control the details/style of the generated images [35, 143, 195, 242, 314]. Recently, methods have been proposed to control the style of the GAN-generated images. For example, StyleGAN [196, 197] proposed the idea of "style mixing", where different latent style vectors are progressively fed to the GAN to control the style and appearance of the output image. To transfer a specific style in a target image to



(A) Simplified version of StyleGAN's first block



(B) Simplified version of StyleGAN's last block



(C) HistoGAN block

Figure 14.2: We inject our histogram into StyleGAN [197] to control the generated image colors. (A) and (B) are simplified versions of the StyleGAN's first and last blocks. We modified the last two blocks of the StyleGAN by projecting our histogram feature into each block's latent space, as shown in (C). The parameter m controls the capacity of the model.

GAN-generated images, an optimization process can be used to project the target image to the generator network's latent space to generate images that share some properties with the target image [5, 197]. However, this process requires expensive computations to find the latent code of the target image. Another direction is to jointly train an encoder-generator network to learn this projection [80, 233, 303]. More recently, methods have advocated different approaches to control the output of GANs, such as using the normalized

flow [6], latent-to-domain-specific mapping [80], deep classification features [340], few-shot image-to-image translation [328], and a single-image training strategy [335]. Despite the performance improvements, most of these methods are limited to work with a single domain of both target and GAN-generated images [233, 303].

We seek to control GAN-generated images using color histograms as our specified representation of image style. Color histograms enable our method to accept target images taken from *any* arbitrary domain. Figure 14.1-top shows GAN-generated examples using our method. As shown in Fig. 14.1, our generated images share the same color distribution as the target images without being restricted to, or influenced by, the semantic content of the target images.

Recoloring Real Images In addition to controlling the GAN-generated images, we seek to extend our approach to perform image recoloring within the GAN framework. In this context, our method accepts a real input image and a target histogram to produce an output image with the fine details of the input image but with the same color distribution given in the target histogram. Our method is trained in a fully unsupervised fashion, where no ground-truth recolored image is required. Instead, we propose a novel adversarial-based loss function to train our network to extract and consider the color information in the given target histogram while producing realistic recolored images. One of the key advantages of using the color histogram representation as our target colors can be shown in Fig. 14.1-bottom, where we can *automatically recolor* an image without directly having to specify a target color histogram. As discussed in Chapter 3, auto-image recoloring is a less explored research area with only a few attempts in the literature (e.g., [33, 94, 216]).

14.2 HistoGAN

We begin by describing the histogram feature used by our method (Sec. 14.2.1). Afterwards, we discuss the proposed modification to StyleGAN [197] to incorporate our histogram feature into the generator network (Sec. 14.2.2). Lastly, we explain how this method can be expanded to control colors of real input images to perform image recoloring (Sec. 14.2.3).

14.2.1 Histogram feature

The histogram feature used by HistoGAN is borrowed from the color constancy literature (see Chapters 4–6) and is constructed to be a differentiable histogram of colors in the log-chroma space due to better invariance to illumination changes [102, 115]. The feature is a 2D histogram of an image's colors projected into a log-chroma space. This 2D histogram is parameterized by uv and conveys an image's color information while being more compact than a typical 3D histogram defined in RGB space. A log-chroma space is defined by the intensity of one channel, normalized by the other two, giving three possible options of how it is defined. Instead of selecting only one such space, all three options can be used to construct three different histograms which are combined together into a histogram feature, $\mathbf{H}$, as an $h \times h \times 3$ tensor.

As explained in Chapters 4–6, the histogram is computed from a given input image, $\mathbf{I}$, by first converting it into the log-chroma space. For instance, selecting the $\mathbf{R}$ color channel as primary and normalizing by $\mathbf{G}$ and $\mathbf{B}$ gives:

$$\mathbf{I}_{uR}(\mathbf{x}) = \log\left(\frac{\mathbf{I_R}(\mathbf{x}) + \epsilon}{\mathbf{I_G}(\mathbf{x}) + \epsilon}\right), \mathbf{I}_{vR}(\mathbf{x}) = \log\left(\frac{\mathbf{I_R}(\mathbf{x}) + \epsilon}{\mathbf{I_B}(\mathbf{x}) + \epsilon}\right), \tag{14.1}$$

where the $\mathbf{R}, \mathbf{G}, \mathbf{B}$ subscripts refer to the color channels of the image $\mathbf{I}$, $\epsilon = 10^{-8}$ is a small constant added for numerical stability, $\mathbf{x}$ is the pixel index, and (uR, vR) are the uv

coordinates based on using $\mathbf{R}$ as the primary channel. The other components $\mathbf{I}_{uG}$, $\mathbf{I}_{vG}$, $\mathbf{I}_{uB}$, $\mathbf{I}_{vB}$ are computed similarly by projecting the $\mathbf{G}$ and $\mathbf{B}$ color channels to the log-chroma space. In Chapter 6, the RGB-uv histogram is computed by thresholding colors to a bin and computing the contribution of each pixel based on the intensity $\mathbf{I}_y(\mathbf{x}) = \sqrt{\mathbf{I}^2_{\mathbf{R}}(\mathbf{x}) + \mathbf{I}^2_{\mathbf{G}}(\mathbf{x}) + \mathbf{I}^2_{\mathbf{B}}(\mathbf{x})}$. In order to make the representation differentiable, we replaced the thresholding operator with a kernel weighted contribution to each bin in Chapter 4. The final unnormalized histogram is computed as:

$$\mathbf{H}(u, v, c) \propto \sum_{\mathbf{x}} k(\mathbf{I}_{uc}(\mathbf{x}), \mathbf{I}_{vc}(\mathbf{x}), u, v)\mathbf{I}_y(\mathbf{x}), \tag{14.2}$$

where $c \in \{\mathrm{R}, \mathrm{G}, \mathrm{B}\}$ and $k(\cdot)$ is a pre-defined kernel. While a Gaussian kernel was originally used in Chapter 4, we found that the inverse-quadratic kernel significantly improved training stability in our current task. The inverse-quadratic kernel is defined as:

$$k(\mathbf{I}_{uc}, \mathbf{I}_{vc}, u, v) = \left(1 + \left(|\mathbf{I}_{uc} - u| / \tau\right)^2\right)^{-1} \times \left(1 + \left(|\mathbf{I}_{vc} - v| / \tau\right)^2\right)^{-1}, \tag{14.3}$$

where τ is a fall-off parameter to control the smoothness of the histogram's bins. Finally, the histogram feature is normalized to sum to one, i.e., $\sum_{u,v,c} \mathbf{H}(u, v, c) = 1$.

14.2.2 Color-controlled Image Generation

Our histogram feature is incorporated into an architecture based on StyleGAN [197]. Specifically, we modified the original design of StyleGAN (Fig. 14.2-[A] and [B]) such that we can "inject" the histogram feature into the progressive construction of the output image. The last two blocks of the StyleGAN (Fig. 14.2-[B]) are modified by replacing the fine-style vector with the color histogram feature. The histogram feature is then projected into a lower-dimensional representation by a "histogram projection" network (Fig. 14.2-[C]). This network consists of eight fully connected layers with a leaky ReLU (LReLU) activation function [257]. The first layer has 1,024 units, while each of the remaining seven

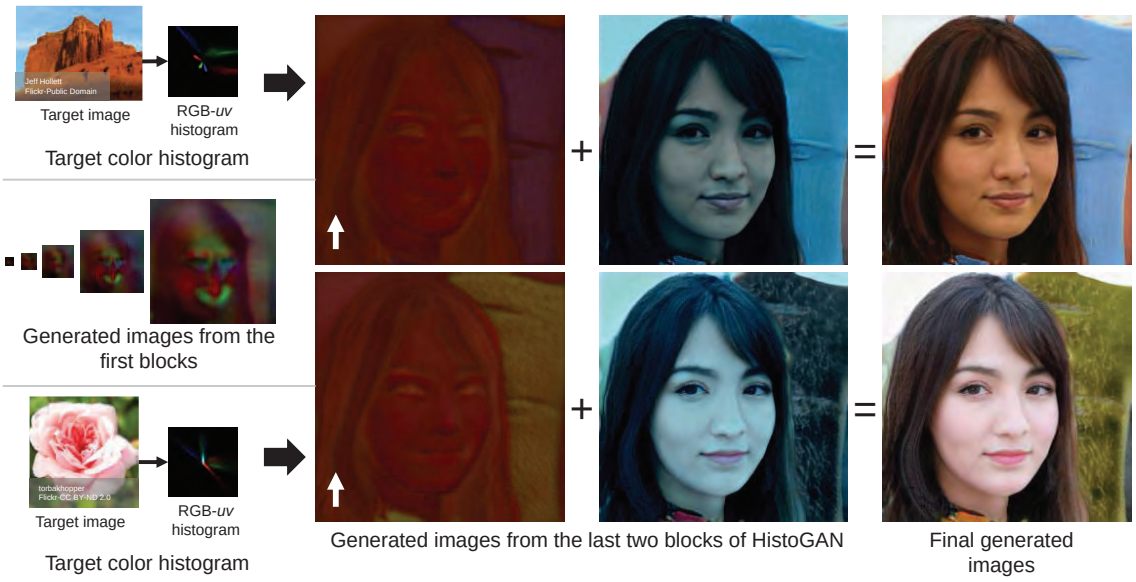


Figure 14.3: Progressively generated images using the HistoGAN modifications.

layers has 512. The "to-latent" block, shown in orange in Fig. 14.2, maps the projected histogram to the latent space of each block. This "to-latent" block consists of a single fc layer with $2^n m$ output neurons, where n is the block number, and m is a parameter used to control the entire capacity of the network.

To encourage generated images to match the target color histogram, a color matching loss is introduced to train the generator. Because of the differentiability of our histogram representation, the loss function, $C(\mathbf{H}_g, \mathbf{H}_t)$, can be any differentiable metric of similarity between the generated and target histograms $\mathbf{H}_g$ and $\mathbf{H}_t$, respectively. For simplicity, we use the Hellinger distance defined as:

$$C\left(\mathbf{H}_g, \mathbf{H}_t\right) = \frac{1}{\sqrt{2}} \left\| \mathbf{H}_g^{1/2} - \mathbf{H}_t^{1/2} \right\|_2, \tag{14.4}$$

where $\|\cdot\|_2$ is the standard Euclidean norm and $\mathbf{H}^{1/2}$ is an element-wise square root.

This color-matching histogram loss function is combined with the discriminator to give

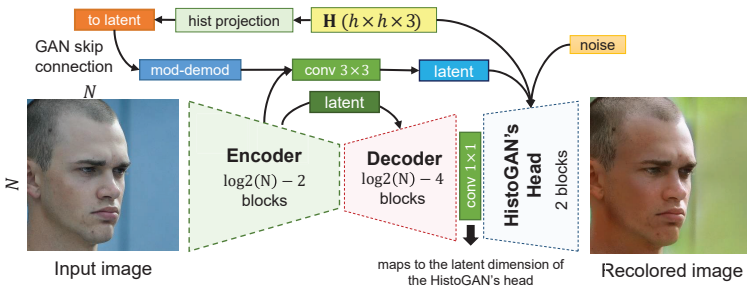


Figure 14.4: Our Recoloring-HistoGAN (ReHistoGAN) network. We map the input image into the HistoGAN's latent space using an encoder-decoder network with skip connections between each encoder and decoder blocks. Additionally, we pass the latent feature of the first two encoder blocks to our GAN's head after processing it with the histogram's latent feature.

the generator network loss:

$$\mathcal{L}_g = D\left(\mathbf{I}_g\right) + \alpha C\left(\mathbf{H}_g, \mathbf{H}_t\right), \tag{14.5}$$

where $\mathbf{I}_g$ is the GAN-generated image, $D\left(\cdot\right)$ is our discriminator network that produces a scalar feature given an image (see Appendix D for more details), $\mathbf{H}_t$ is the target histogram feature (injected into the generator network), $\mathbf{H}_g$ is the histogram feature of $\mathbf{I}_g$, $C\left(\cdot\right)$ is our histogram loss function, and α is a scale factor to control the strength of the histogram loss term.

As our histogram feature is computed by a set of differentiable operations, our loss function (Eqs. 14.4 and 14.5) can be optimized using SGD. During training, different target histograms $\mathbf{H}_t$ are required. To generate these for each generated image, we randomly select two images from the training set, compute their histograms $\mathbf{H}_1$ and $\mathbf{H}_2$, and then randomly interpolate between them. Specifically, for each generated image during training, we generate a random target histogram as follows:

$$\mathbf{H}_t = \delta\mathbf{H}_1 + \left(1 - \delta\right)\mathbf{H}_2, \tag{14.6}$$

Figure 14.5: Results of training ReHistoGAN with and without the variance loss term described in Eq. 14.9.

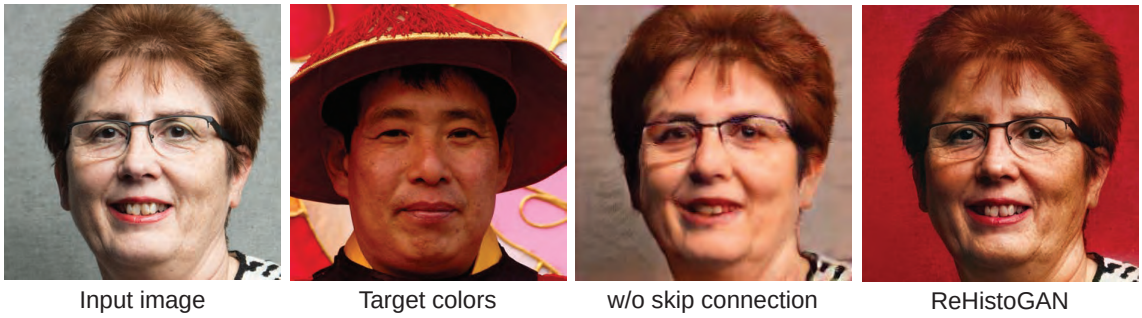


Figure 14.6: Results of image recoloring using the encoder-GAN reconstruction without skip connections and our ReHistoGAN using our proposed loss function.

where $\delta \sim U(0,1)$ is sampled uniformly.

With this modification to the original StyleGAN architecture, our method can control the colors of generated images using our color histogram features. Figure 14.3 shows the progressive construction of the generated image by HistoGAN. As can be seen, the outputs of the last two blocks are adjusted to consider the information conveyed by the target histogram to produce output images with the same color distribution represented in the target histogram.

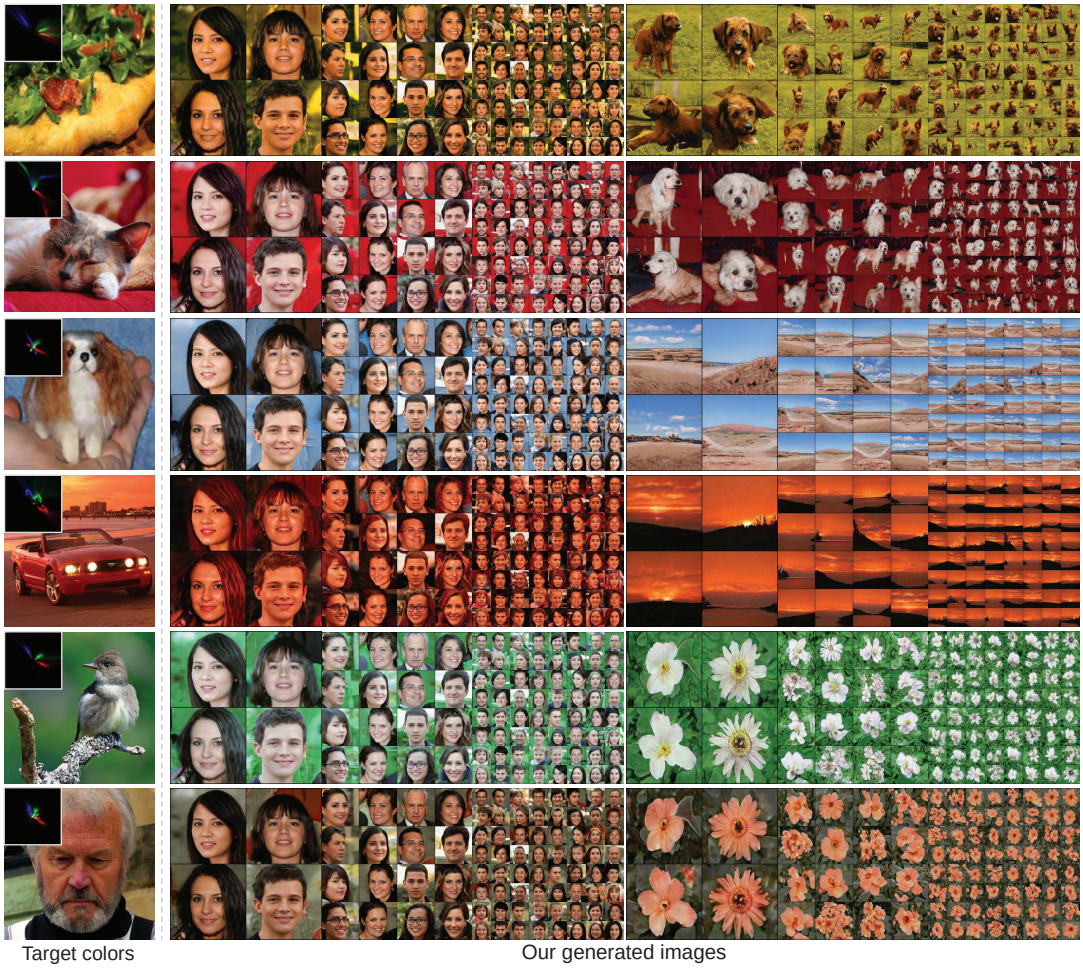


Figure 14.7: Images generated by HistoGAN. For each input image shown in the left, we computed the corresponding target histogram (shown in the upper left corner of the left column) and used it to control colors of the generated images in each row.

14.2.3 Image Recoloring

We can also extend HistoGAN to recolor an input image, as shown in Fig. 14.1-bottom. Recoloring an existing input image, $\mathbf{I}_i$, is not straightforward because the randomly sampled noise and style vectors are not available as they are in a GAN-generated scenario. As shown in Fig. 14.3, the head of HistoGAN (i.e., the last two blocks) are responsible for

controlling the colors of the output image. Instead of optimizing for noise and style vectors that could be used to generate a given image $\mathbf{I}_i$, we propose to train an encoding network that maps the input image into the necessary inputs of the head of HistoGAN.

With this approach, the head block can be given different histogram inputs to produce a wide variety of recolored versions of the input image. We dub this extension the "Recoloring-HistoGAN" or ReHistoGAN for short. The architecture of ReHistoGAN is shown in Fig. 14.4. The "encoder" has a U-Net-like structure [323] with skip connections. To ensure that fine details are preserved in the recolored image, $\mathbf{I}_r$, the early latent feature produced by the first two U-Net blocks are further provided as input into the HistoGAN's head through skip connections.

The target color information is passed to the HistoGAN head blocks as described in Sec. 14.2.2. Additionally, we allow the target color information to influence through the skip connections to go from the first two U-Net-encoder blocks to the HistoGAN's head. We add an additional histogram projection network, along with a "to-latent" block, to project our target histogram to a latent representation. This latent code of the histogram is processed by weight modulation-demodulation operations [197] and is then convolved over the skipped latent of the U-Net-encoder's first two blocks. We modified the HistoGAN block, described in Fig. 14.2, to accept this passed information (see Appendix D for more information). The leakage of the target color information helps ReHistoGAN to consider information from both the input image content and the target histogram in the recoloring process.

We initialize our encoder-decoder network using He's initialization [158], while the weights of the HistoGAN head are initialized based on a previously trained HistoGAN model (trained in Sec. 14.2.2). The entire ReHistoGAN is then jointly trained to minimize

the following loss function:

$$\mathcal{L}_r = \beta R\left(\mathbf{I}_i, \mathbf{I}_r\right) + \gamma D\left(\mathbf{I}_r\right) + \alpha C\left(\mathbf{H}_r, \mathbf{H}_t\right) \tag{14.7}$$

where $R\left(\cdot\right)$ is a reconstruction term, which encourages the preservation of image structure and α, β, and γ are hyperparameters used to control the strength of each loss term (see Appendix D for associated ablation study). The reconstruction loss term, $R\left(\cdot\right)$, computes the L1 norm between the second order derivative of our input and recolored images as:

$$R\left(\mathbf{I}_i, \mathbf{I}_r\right) = \left\|\mathbf{I}_i * \mathbf{L} - \mathbf{I}_r * \mathbf{L}\right\|_1 \tag{14.8}$$

where $*\mathbf{L}$ denotes the application of the Laplacian operator. The idea of employing the image derivative was used initially to achieve image seamless cloning [19, 300], where this Laplacian operator suppressed image color information while keeping the most significant perceptual details. Intuitively, ReHistoGAN is trained to consider the following aspects in the output image: (i) having a similar color distribution to the one represented in the target histogram, this is considered by $C\left(\cdot\right)$, (ii) being realistic, which is the goal of $D\left(\cdot\right)$, and (iii) having the same content of the input image, which is the goal of $R\left(\cdot\right)$.

Our model trained using the loss function described in Eq. 14.7 produces reasonable recoloring results. However, we noticed that, in some cases, our model tends to only apply a global color cast (i.e., shifting the recolored image's histogram) to minimize $C\left(\cdot\right)$. To mitigate this behavior, we added variance loss term to Eq. 14.7. The variance loss can be described as:

$$V(\mathbf{I}_i, \mathbf{I}_r) = -w \sum_{c \in \{\mathbf{R},\mathbf{G},\mathbf{B}\}} \left|\sigma\left(\mathbf{I}_{ic} * \mathbf{G}\right) - \sigma\left(\mathbf{I}_{rc} * \mathbf{G}\right)\right|, \tag{14.9}$$

where $\sigma\left(\cdot\right)$ computes the standard deviation of its input (in this case the blurred versions of $\mathbf{I}_i$ and $\mathbf{I}_r$ using a Gaussian blur kernel, $\mathbf{G}$, with a scale parameter of 15), and $w = \left\|\mathbf{H}_t - \mathbf{H}_i\right\|_1$ is a weighting factor that increases as the target histogram and the input

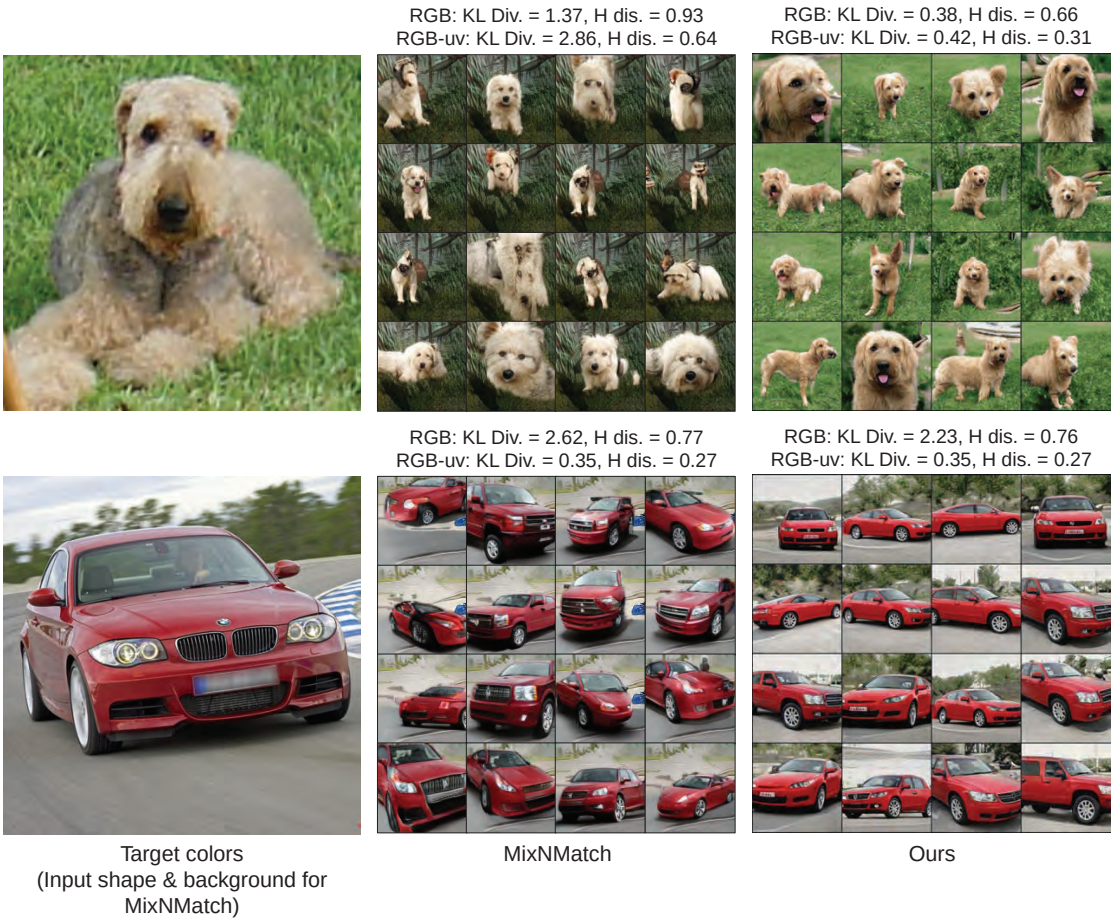


Figure 14.8: Comparison with the MixNMatch method [233]. In the shown results, the target images are used as input shape and background images for the MixNMatch method [233].

image's histogram, $\mathbf{H}_t$ and $\mathbf{H}_i$, become dissimilar and the global shift solution becomes more problematic.

The variance loss encourages the network to avoid the global shifting solution by increasing the differences between the color variance in the input and recolored images. The reason behind using a blurred version of each image is to avoid having a contradiction between the variance loss and the reconstruction loss—the former aims to increase the

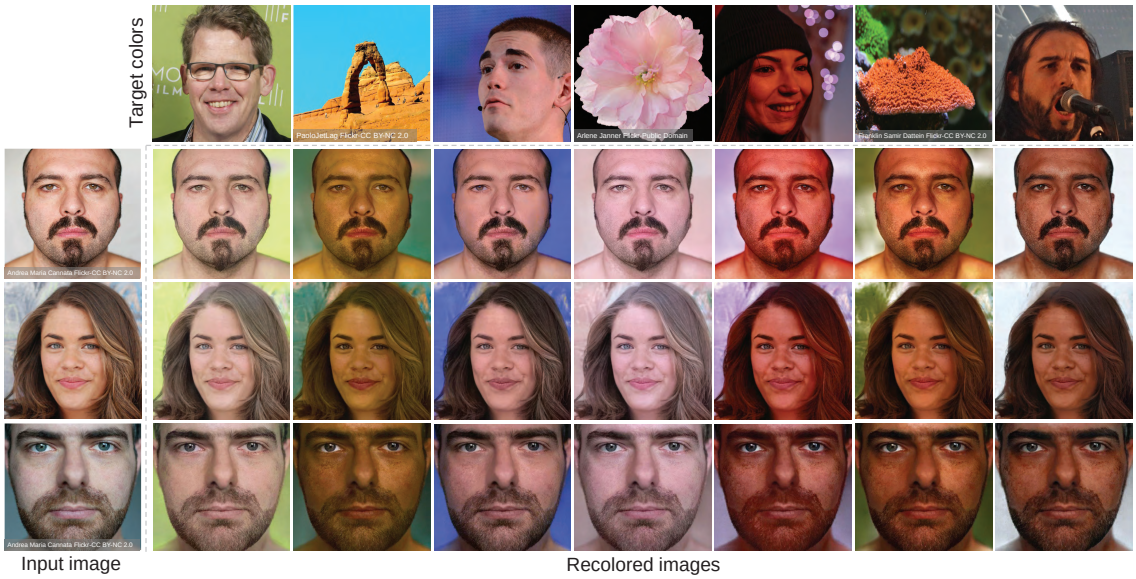


Figure 14.9: Results of our ReHistoGAN. The shown results are after recoloring input images (shown in the left column) using the target colors (shown in the top row).

differences between the variance of the *smoothed* colors in each image, while the latter aims to retain the similarity between the fine details of the input and recolored images. Figure 14.5 shows recoloring results of our trained models with and without the variance loss term.

We train ReHistoGAN with target histograms sampled from the target domain dataset, as described earlier in Sec. 14.2.2 (Eq. 14.6).

A simpler architecture was experimented with initially, which did not make use of the skip connections and the end-to-end fine tuning (i.e., the weights of the HistoGAN head were fixed). However, this approach gave unsatisfactory result, and generally failed to retain fine details of the input image. A comparison between this approach and the above ReHistoGAN architecture can be seen in Fig. 14.6.

Our ReHistoGAN processes a single 256×256 image in $\sim$0.5 seconds using a single GTX 1080 GPU. To process images with higher resolutions, we use the guided upsam-

Figure 14.10: Comparison with the high-resolution daytime translation (HiDT) method [33].

Figure 14.11: Comparisons between our ReHistoGAN and other image color/style transfer methods, which are: Reinhard et al., [318], Xiao et al., [381], Pitié and Kokaram [304], Nguyen et al., [287], Gatys et al., [130], and Sheng et al., [337].

pling procedure [73] (as described in Chapter 12) with additional ∼21 seconds using an unoptimized implementation of the guided upsampling.

14.3 Results and Discussion

Image Recoloring This section discusses our results and comparisons with alternative methods proposed in the literature for controlling color. Due to hardware limitations, we

Figure 14.12: Additional results for image recoloring. We recolor input images, shown in the right by feeding our network with the target histograms of images shown in the top.

used a lightweight version of the original StyleGAN [197] by setting m to 16, shown in Fig. 14.2. We begin by presenting our image generation results, followed by our results on image recoloring. Additional results, comparisons, and discussion are also available in Appendix D.

Image Generation Figure 14.7 shows examples of our HistoGAN-generated images. Each row shows samples generated from different domains using the corresponding in-

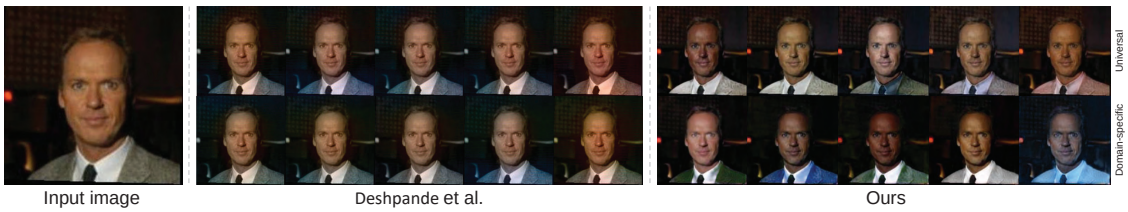


Figure 14.13: Comparisons between our ReHistoGAN and the diverse colorization method proposed by Deshpande et al., [94]. For our ReHistoGAN, we show the resutls of our domain-specific and universal models.

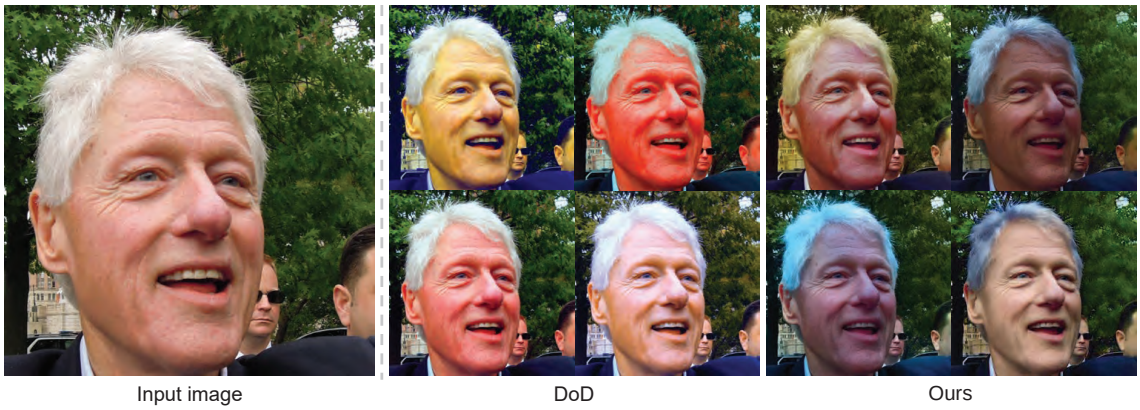


Figure 14.14: Automatic recoloring comparison with our method in Chapter 13.

put target colors. For each domain, we fixed the style vectors responsible for the coarse and middle styles to show our HistoGAN's response to changes in the target histograms. Qualitative comparisons with the recent MixNMatch method [233] are provided in Fig. 14.8.

To evaluate the potential improvement/degradation of the generated-image diversity and quality caused by our modification to StyleGAN, we trained StyleGAN [197] with $m = 16$ (i.e., the same as our model capacity) without our histogram modification. We evaluated both models on different datasets, including our collected set of landscape images. For each dataset, we generated 10,000 256×256 images using the StyleGAN and our HistoGAN. We

evaluated the generated-image quality and diversity using the Frechét inception distance (FID) metric [164] using the second max-pooling features of the Inception model [349].

We further evaluated the ability of StyleGAN to control colors of GAN-generated images by training a regression deep neural network (ResNet [159]) to transform generated images back to the corresponding fine-style vectors. These fine-style vectors are used by the last two blocks of StyleGAN and are responsible for controlling delicate styles, such as colors and lights [196, 197].

The training was performed for each domain separately using 100,000 training StyleGAN-generated images and their corresponding "ground-truth" fine-style vectors. In the testing phase, we used the trained ResNet to predict the corresponding fine-style vectors of the target image—these target images were used to generate the target color histograms for HistoGAN's experiments. We then generated output images based on the predicted fine-style vectors of each target image. In the evaluation of StyleGAN and HistoGAN, we used randomly selected target images from the same domain.

The Hellinger distance and KL divergence were used to measure the color errors between the histograms of the generated images and the target histogram; see Table 14.1.

Figure 14.9 shows examples of image recoloring using our ReHistoGAN. A comparison with the recent high-resolution daytime translation (HiDT) method [33] is shown in Fig. 14.10. Additional comparisons with image recoloring and style transfer methods are shown in Fig. 14.11. Arguably, our ReHistoGAN produces image recoloring results that are visually more compelling than the results of other methods for image color/style transfer. As shown in Fig. 14.11, our ReHistoGAN produces realistic recoloring even when the target image is from a different domain than the input image, compared to other image style transfer methods (e.g., [130, 337]). Additional results are shown in Fig. 14.12.

Another strategy for image recoloring is to learn a diverse colorization model. That is,

Figure 14.15: Results of using our ReHistoGAN for a diverse image colorization.

the input image is converted to grayscale, and then a trained method for diverse colorization can generate different colorized versions of the input image. In Fig. 14.13, we show a qualitative comparison with the diverse colorization method proposed by Deshpande et al., [94].

Lastly, we provide a qualitative comparison with our auto-recoloring method presented in Chapter 13 in Fig. 14.14. In the shown example, our target histograms were dynamically generated by sampling from a pre-defined set of histograms and applying a linear interpolation between the sampled histograms (see Eq. 14.6).

What is Learned? Our method learns to map color information, represented by the target color histogram, to an output image's colors with a realism consideration in the recolored image. Maintaining realistic results is achieved by learning proper matching between the target colors and the input image's semantic objects (e.g., grass can be green, but not blue). To demonstrate this, we examine a trained ReHistoGAN model for an image colorization task, where the input image is grayscale. The input of a grayscale image means that our ReHistoGAN model has no information regarding objects' colors in the input image. Figure 14.15 shows outputs where the input has been "colorized". As can be seen, the output images have been colorized with good semantic-color matching based

on the image's content.

14.4 Summary

We have presented HistoGAN, a simple, yet effective, method for controlling colors of GAN-generated images. Our HistoGAN framework learns how to transfer the color information encapsulated in a target histogram feature to the colors of a generated output image. To the best of our knowledge, this is the first work to control the color of GAN-generated images directly from color histograms. Color histograms provide an abstract representation of image color that is decoupled from spatial information. This allows the histogram representation to be less restrictive and suitable for GAN-generation across arbitrary domains.

We have shown that HistoGAN can be extended to control colors of real images in the form of the ReHistoGAN model. Our recoloring results are visually more compelling than currently available solutions for image recoloring. Our image recoloring also enables "auto-recoloring" by sampling from a pre-defined set of histograms. This allows an image to be recolored to a wide range of visually plausible variations. HistoGAN can serve as a step towards intuitive color control for GAN-based graphic design and artistic endeavors.

Table 14.1: Comparison with StyleGAN [197]. The term 'w/ proj.' refers to projecting the target image colors into the latent space of StyleGAN. We computed the similarity between the target and generated histograms in RGB and projected RGB-uv color spaces. For each dataset, we report the number of training images. Note that StyleGAN results shown here *do not* represent the actual output of [197], as the used model here has less capacity ($m = 16$).

Dataset	StyleGAN [197]						HistoGAN (ours)				
	FID		RGB hist. (w/ proj.)		RGB-uv hist. (w/ proj.)		FID	RGB hist. (w/ proj.)		RGB-uv hist. (w/ proj.)	
	w/o proj.	w/ proj.	KL Div.	H dis.	KL Div.	H dis.		KL Div.	H dis.	KL Div.	H dis.
Faces (69,822) [196]	9.5018	14.194	1.3124	0.9710	1.2125	0.6724	**8.9387**	**0.9810**	**0.7487**	**0.4470**	**0.3088**
Flowers (8,189) [288]	10.876	15.502	1.0304	0.9614	2.7110	0.7038	**4.9572**	**0.8986**	**0.7353**	**0.3837**	**0.2957**
Cats (9,992) [84]	**14.366**	21.826	1.6659	0.9740	1.4051	0.5303	17.068	**1.0054**	**0.7278**	**0.3461**	**0.2639**
Dogs (20,579) [200]	**16.706**	30.403	1.9042	0.9703	1.4856	0.5658	20.336	**1.3565**	**0.7405**	**0.4321**	**0.3058**
Birds (9,053) [367]	3.5539	12.564	1.9035	0.9706	1.9134	0.6091	**3.2251**	**1.4976**	**0.7819**	**0.4261**	**0.3064**
Anime (63,565) [81]	**2.5002**	9.8890	0.9747	0.9869	1.4323	0.5929	5.3757	**0.8547**	**0.6211**	**0.1352**	**0.1798**
Hands (11,076) [10]	2.6853	2.7826	0.9387	0.9942	0.3654	0.3709	**2.2438**	**0.3317**	**0.3655**	**0.0533**	**0.1085**
Landscape (4,316)	24.216	29.248	0.8811	0.9741	1.9492	0.6265	**23.549**	**0.8315**	**0.8169**	**0.5445**	**0.3346**
Bedrooms (303,116) [393]	10.599	14.673	1.5709	0.9703	1.2690	0.5363	**4.5320**	**1.3774**	**0.7278**	**0.2547**	**0.2464**
Cars (16,185) [207]	21.485	25.496	1.6871	0.9749	0.7364	0.4231	**14.408**	**1.0743**	**0.7028**	**0.2923**	**0.2431**
Aerial Scenes (36,000) [259]	**11.413**	14.498	2.1142	0.9798	1.1462	0.5158	12.602	**0.9889**	**0.5887**	**0.1757**	**0.1890**

Part VII

Conclusions and Future Work

15 Conclusions and Future Work

15.1 Concluding Remarks

This thesis has presented a number of works targeting research related to color correction, enhancement, and editing. The thesis discussed color correction and editing from the standpoint of the camera's ISP. We first reviewed different standard color spaces and the details of camera ISPs in Chapter 2. We then presented a survey of prior work for color constancy, enhancement, and editing in Chapter 3. Afterwards, we discussed methods for computational color correction onboard cameras in Chapters 4 and 5. Specifically, we have proposed two sensor-independent learning-based methods for illuminant estimation.

In Chapter 6, we have presented the first method to correct WB errors appears in camera-rendered photographs. Through an extensive evaluation, we showed that our method outperforms existing solutions in correcting such WB errors in camera-rendered images. We further discussed a modification to our WB method (presented in Chapter 6) to augment training data used by DNNs for different computer vision tasks in Chapter 7. We showed that our WB augmenter improves the robustness of DNN methods for image classification and image semantic segmentation against WB errors.

We have discussed a set of methods designed for color editing based on WB editing in Chapters 8–10. In particular, Chapter 8 extends our method proposed in Chapter 6

to allow for user interactive WB editing in post-capture stage. Then, we have proposed a DNN framework for WB editing in Chapter 9. Lastly, we discussed a simple modification to camera ISPs in Chapter 10 to achieve accurate WB editing in camera-rendered images.

We also discussed camera exposure errors and how they can significantly affect the quality of colors in camera-rendered photographs. To that end, we have proposed two methods to enhance under- and over-exposed images in Chapters 11 and 12. Our first method in Chapter 11 was built on the idea of image linearization, where we discussed the benefit of accurately unprocess camera photo-finishing to enhance low-light images. In Chapter 12, we proposed a post-capture DNN enhancement method to correct colors of under- and over-exposed photographs.

Lastly, we have proposed two methods for auto color editing in camera-rendered photographs, where we have presented a data-driven approach for auto recoloring (Chapter 13) and a GAN-based method for controlling colors in synthetic images produced by GANs and real photographs (Chapter 14).

15.1.1 Broader Impact

We discussed several methods in this thesis that target improving the quality of photographs. When compared with other alternatives, our methods introduce more flexible, practical, and accurate image color correction, enhancement, and editing solutions. While the impact of our work is often evaluated from an aesthetic point of view, color correction and image enhancement can be used as a pre-processing step to enhance crucial computer vision applications. These applications may include skin cancer diagnosis [194], diabetic retinopathy detection/classification [343], image forensics [89], and object tracking [390]. The importance of color correction for computer vision tasks was discussed in Chapter 7, where we presented, through a thorough evaluation, an experimental analysis of color cor-

rection's impact on image classification and image semantic segmentation tasks. Chapter 7 also discussed the idea of WB augmentation, which can improve the accuracy of other critical computer vision tasks.

We have also proposed methods for image recoloring that can produce realistic versions of images after changing their original colors. This recoloring tool can be used as a data augmenter to automatically recolor (or augment) training images to improve the robustness and accuracy of other deep learning techniques for different tasks. These recoloring techniques, though, like two sides of a coin, carry both the negative and the positive sides, where image recoloring may result in changes in images intended to deceive the observer (for example, day-to-night image translation or alter evidence). Future work is required to ensure that the authenticity of images is verifiable after such modifications have been performed.

15.2 Future Research Directions

The work in this thesis focused on color processing and discussed two main factors that significantly affect the quality of colors in camera-rendered photographs, namely: (i) WB settings and (ii) exposure settings. The work presented in this thesis was modeled based on our knowledge of camera ISP pipeline design. This knowledge allows us to generate accurate datasets to develop our methods. Through our discussion on WB, we showed that WB editing is not only required to correct improperly white-balanced images but can also help to satisfy user preference, especially in challenging scenes (e.g., multi-illuminant scenes). We further discussed another direction of color editing by modeling/learning a prior knowledge of semantic-color matches to achieve a realistic auto image recoloring based on the image's content. We then extended this data-driven idea to attain more compelling results using GANs. Through our discussion in previous chapters, we can summarize the

future research directions into the following points: (i) deep learning in camera ISPs, (ii) multi-illuminant CC, (iii) user preferences in WB, (iv) enhancing over-exposed images, (v) and universal image recoloring.

15.2.1 Deep Learning in Camera ISPs

As discussed in Chapter 2, there is a sequence of processing steps applied onboard camera ISPs to render the final sRGB image. Each module of camera ISP is responsible for a particular task to enhance the perceptual quality of the captured image. Due to the astounding results achieved by deep learning on a broad range of computer vision tasks, researchers proposed several methods to replace specific modules in camera ISPs with DNNs showing impressive results compared to traditional approaches (e.g., [12, 134, 234, 243, 311, 372]). More recently, a few methods proposed to replace the entire camera ISP with a single DNN model to be trained in an end-to-end fashion (e.g., [177,332]). While this idea is promising—in the sense that a single end-to-end trained model would diminish the effort required to tune the parameters of each camera ISP's module to get the final desirable results—it has some drawbacks. First, the computational power required by currently used DNNs usually exceeds the memory and computing limits in camera hardware units. Second, such DNNs are often treated as black boxes making it hard to have insights into the reason behinds any failure cases. Third, although employing deep learning offers image quality as good as that is produced by traditional camera ISPs without the tedious manual parameter tuning process required to deploy traditional camera ISP, training such DNNs usually requires collecting paired training data, which may require more effort than the camera ISP's parameter tuning procedure.

With that said, DNNs not only allow the generation of good-quality sRGB images, they can also be designed to have additional benefits that are hard to be attempted by

traditional camera ISPs. For example, the recent work in [383] proposed an invertible DNN to emulate almost the entire camera ISP's tasks (i.e., denoising, white balancing, etc.), where the DNN used to replace the camera ISP consists of invertible blocks, which means the entire rendering process can be reversed to reconstruct the original linear raw images.

Following the idea of using invertible DNNs to emulate the camera rendering procedure [383], there is an interesting research direction to not only utilize deep learning to replace camera ISPs but also to have a more effective diagnosis of failure cases. One may think of designing a single deep learning framework with different invertible sub-networks, each of which is designed for a specific rendering task. That is, each step in the rendering (e.g., denoising, white balancing, color mapping, etc.) can be inverted or modified in the post-capture stage. Another interesting research direction related to this point would be designing a learning-based camera ISP that learns user preferences and updates the ISP color rendering based on the post-capture edits performed by the user.

15.2.2 Multi-Illuminant Color Constancy

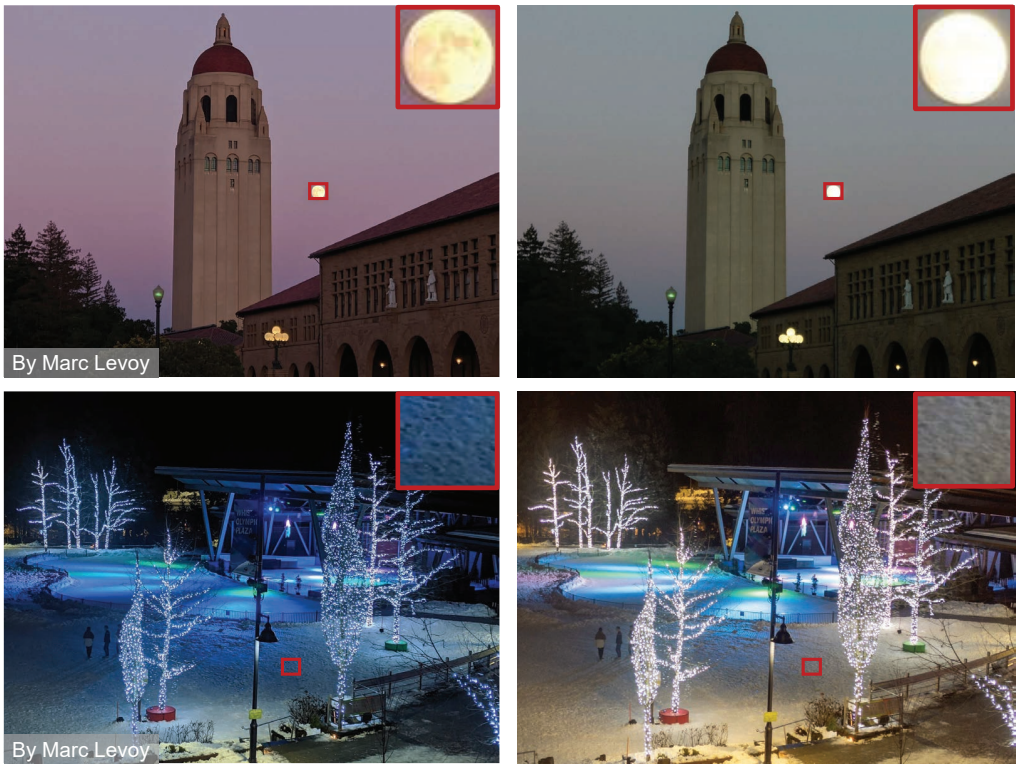
We have discussed and proposed several methods for color correction in order to achieve CC in photographs in Chapters 3–6. These methods, including our methods (Chapters 4–6), target scenes with a single uniform lighting. In many real scenarios, however, scenes have mixed light sources and, thus, these methods are expected to correct colors only for one of these illuminants and ignore other illuminants. For that reason, we propose the WB editing methods in Chapters 8–10 to circumvent the issue by allowing the user to manually edit WB settings in the post-capture.

Though this interactive approach gives a reasonable way to deal with such cases, auto multi-illuminant scene color constancy would offer a more comfortable and effective so-

lution. In the literature, as discussed in Chapter 3, there are a few attempts proposed to deal with multi-illuminant scenarios. These methods, however, could work only under certain conditions and generally are not accurate enough. While deep learning solutions are expected to achieve more accurate results for multi-illuminant color constancy, there are no paired multi-illuminant datasets that represent realistic scenarios of scenes with mixed lighting for supervised learning. A promising research direction is to generate realistic multi-illuminant datasets to accelerate the adoption of learning-based multi-illuminant color constancy by the research community and the camera industry.

15.2.3 User Preferences in White Balancing

As discussed in Chapter 3, WB aims to normalize the color cast caused by scene illumination, such that achromatic object's RGB vectors should lie along the achromatic white line in the 3D color space (i.e., R=G=B). In many cases, however, what we observe in real scenes do not match this goal. Figure 15.1 shows two examples, where the real appearance of the scene does not match the result of accurate WB correction using manually selected achromatic reference point. As can be seen, the WB procedure changes the colors to make achromatic objects look white. Though this makes intuitive sense, as any WB algorithm would remove any color casts caused by scene illuminant, these results seem incorrect to many users as it shows completely different colors than what is observed in reality. Potential research directions may include performing a comprehensive user study to define user preferences under diverse lighting conditions. Based on this analysis of user preferences, one could consider user preferences in the ground-truth labels of color constancy datasets to train DNN models that achieve more compelling WB results.



(A) Real appearance of the scene

(B) Manual WB using an achromatic reference point

Figure 15.1: Image white balancing does not always match what we see in reality. (A) Real scene appearance. (B) WB result. In each example, we rendered the DNG file of each image using Adobe Camera Raw in Photoshop. For WB results, an achromatic reference point were manually selected from each raw image. The red boxes show the selected reference point used for each image. Photo credit: Marc Levoy.

15.2.4 Enhancing Over-Exposed Images

In Chapter 12, we have discussed the first work that employs deep learning towards explicitly enhancing over-exposed images and showed that most of the prior work focuses only on low-light image enhancement and ignores the over-exposure errors. In many cases, over-exposed images may have no information in several regions due to camera exposure

errors. Thus, a DNN should hallucinate the missing content in order to correct this image. This hallucination should have different treatments based on the semantic objects in the photographs—for example, completing a corrupted grass region with a reasonable inpainting quality can be accepted, while the situation is different for other objects, such as faces that require a more careful image completion. A potential research direction may include semantic-aware inpainting methods to correct over-exposed images.

15.2.5 Universal Image Recoloring

We discussed a GAN-based DNN method for image recoloring in Chapter 14. While our method offers domain-specific recoloring models, we show in Appendix D that by training on a large dataset, we can train a universal model for image recoloring. The results of this universal recoloring model, however, are less realistic than the domain-specific trained models. Another direction of future research work may include a pre-classification stage to determine the class of each image before recoloring in order to utilize a proper recoloring model for each image.

Part VIII

Bibliography

References

[1] Adobe Photoshop user guide. `https://helpx.adobe.com/ca/photoshop/user-guide.html`. Accessed: 2021-01-15.

[2] Affinity Photo – Professional image editing software. https://affinity.serif.com. Accessed: 2021-01-15.

[3] Skylum. https://skylum.com. Accessed: 2021-01-15.

[4] Digital negative (DNG) specification. Technical report, Adobe Systems Incorporated, 2012. Version 1.4.0.0.

[5] Rameen Abdal, Yipeng Qin, and Peter Wonka. Image2StyleGAN: How to embed images into the stylegan latent space? In *IEEE International Conference on Computer Vision (ICCV)*, 2019.

[6] Rameen Abdal, Peihao Zhu, Niloy Mitra, and Peter Wonka. StyleFlow: Attribute-conditioned exploration of StyleGAN-generated images using conditional continuous normalizing flows. *arXiv preprint arXiv:2008.02401*, 2020.

[7] Abdelrahman Abdelhamed, Stephen Lin, and Michael S Brown. A high-quality denoising dataset for smartphone cameras. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[8] Adobe. Color and camera raw. `https://helpx.adobe.com/ca/photoshop-elements/using/color-camera-raw.html`. Accessed: 2021-01-15.

[9] Mahmoud Afifi. Semantic white balance: Semantic color constancy using convolutional neural network. *arXiv preprint arXiv:1802.00153*, 2018.

[10] Mahmoud Afifi. 11K hands: Gender recognition and biometric identification using a large dataset of hand images. *Multimedia Tools and Applications*, 78(15):20835–20854, 2019.

[11] Mahmoud Afifi, Abdelrahman Abdelhamed, Abdullah Abuolaim, Abhijith Punnappurath, and Michael S Brown. CIE XYZ Net: Unprocessing images for low-level computer vision tasks. *arXiv preprint arXiv:2006.12709*, 2020.

[12] Mahmoud Afifi, Jonathan T Barron, Chloe LeGendre, Yun-Ta Tsai, and Francois Bleibel. Cross-camera convolutional color constancy. *arXiv preprint arXiv:2011.11890*, 2020.

[13] Mahmoud Afifi and Michael S Brown. Sensor-independent illumination estimation for DNN models. In *British Machine Vision Conference (BMVC)*, 2019.

[14] Mahmoud Afifi and Michael S Brown. What else can fool deep learning? addressing color constancy errors on deep neural network performance. In *ICCV*, 2019.

[15] Mahmoud Afifi and Michael S Brown. Deep white-balance editing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[16] Mahmoud Afifi and Michael S Brown. Interactive white balancing for camera-rendered images. In *Color and Imaging Conference*, 2020.

[17] Mahmoud Afifi, Marcus A Brubaker, and Michael S Brown. Histogan: Controlling colors of gan-generated and real images via color histograms. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[18] Mahmoud Afifi, Konstantinos G Derpanis, Björn Ommer, and Michael S Brown. Learning multi-scale photo exposure correction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[19] Mahmoud Afifi and Khaled F Hussain. MPB: A modified Poisson blending technique. *Computational Visual Media*, 1(4):331–341, 2015.

[20] Mahmoud Afifi, Brian Price, Scott Cohen, and Michael S Brown. When color constancy goes wrong: Correcting improperly white-balanced images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[21] Mahmoud Afifi, Brian L Price, Scott Cohen, and Michael S Brown. Image recoloring based on object color distributions. In *Eurographics 2019 – Short Papers*, 2019.

[22] Mahmoud Afifi, Abhijith Punnappurath, Abdelrahman Abdelhamed, Hakki Can Karaimer, Abdullah Abuolaim, and Michael S Brown. Color temperature tuning: Allowing accurate post-capture white-balance editing. In *Color and Imaging Conference*, 2019.

[23] Mahmoud Afifi, Abhijith Punnappurath, Graham Finlayson, and Michael S Brown. As-projective-as-possible bias correction for illumination estimation algorithms. *Journal of the Optical Society of America A*, 36(1):71–78, 2019.

[24] George A Agoston. *Color Theory and Its Application in Art and Design*. Springer, 2013.

[25] Eirikur Agustsson and Radu Timofte. NTIRE 2017 challenge on single image super-resolution: Dataset and study. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2017.

[26] Miika Aittala and Frédo Durand. Burst image deblurring using permutation invariant convolutional neural networks. In *European Conference on Computer Vision (ECCV)*, 2018.

[27] Arash Akbarinia and C Alejandro Parraga. Colour constancy beyond the classical receptive field. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(9):2081–2094, 2017.

[28] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6:14410–14430, 2018.

[29] Yağiz Aksoy, Tunç Ozan Aydin, Marc Pollefeys, and Aljoša Smolić. Interactive high-quality green-screen keying via color unmixing. *ACM Transactions on Graphics (TOG)*, 36(4):152:1–152:12, 2016.

[30] Yağiz Aksoy, Tunç Ozan Aydin, Aljoša Smolić, and Marc Pollefeys. Unmixing-based soft color segmentation for image manipulation. *ACM Transactions on Graphics (TOG)*, 36(2):1–19, 2017.

[31] Matthew Anderson, Ricardo Motta, Srinivasan Chandrasekar, and Michael Stokes. Proposal for a standard default color space for the Internet—sRGB. In *Color and Imaging Conference*, 1996.

[32] Alexander Andreopoulos and John K Tsotsos. On sensor bias in experimental methods for comparing interest-point, saliency, and recognition algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(1):110–126, 2012.

[33] Ivan Anokhin, Pavel Solovev, Denis Korzhenkov, Alexey Kharlamov, Taras Khakhulin, Aleksei Silvestrov, Sergey Nikolenko, Victor Lempitsky, and Gleb Sterkin. High-resolution daytime translation without domain labels. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[34] Relja Arandjelovic and Andrew Zisserman. Three things everyone should know to improve object retrieval. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[35] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv preprint arXiv:1701.07875*, 2017.

[36] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Annual ACM-SIAM Symposium on Discrete Algorithms*, 2007.

[37] Çağlar Aytekin, Jarno Nikkanen, and Moncef Gabbouj. A data set for camera-independent color constancy. *IEEE Transactions on Image Processing*, 27(2):530–544, 2018.

[38] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017.

[39] Nikola Banić and Karlo Koščević. Illumination estimation challenge. `https://www.isispa.org/illumination-estimation-challenge`. Accessed: 2021-01-15.

[40] Nikola Banic and Sven Loncaric. Color dog-guiding the global illumination estimation to better accuracy. In *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISAPP)*, 2015.

[41] Nikola Banić and Sven Lončarić. Unsupervised learning for color constancy. *arXiv preprint arXiv:1712.00436*, 2017.

[42] Kobus Barnard, Vlad Cardei, and Brian Funt. A comparison of computational color constancy algorithms: methodology and experiments with synthesized data. *IEEE Transactions on Image Processing*, 11(9):972–984, 2002.

[43] Kobus Barnard, Graham Finlayson, and Brian Funt. Color constancy for scenes with varying illumination. *Computer Vision and Image Understanding*, 65(2):311–321, 1997.

[44] Jonathan T Barron. Convolutional color constancy. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[45] Jonathan T Barron and Yun-Ta Tsai. Fast fourier color constancy. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[46] Ronen Basri and David W Jacobs. Lambertian reflectance and linear subspaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2):218–233, 2003.

[47] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded up robust features. In *European Conference on Computer Vision (ECCV)*, 2006.

[48] Shida Beigpour, Christian Riess, Joost Van De Weijer, and Elli Angelopoulou. Multi-illuminant estimation with conditional random fields. *IEEE Transactions on Image Processing*, 23(1):83–96, 2014.

[49] Jeffrey S Beis and David G Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1997.

[50] S Bianco and R Schettini. Two new von kries based chromatic adaptation transforms found by numerical optimization. *Color Research & Application*, 35(3):184–192, 2010.

[51] Simone Bianco and Claudio Cusano. Quasi-unsupervised color constancy. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[52] Simone Bianco, Claudio Cusano, Paolo Napoletano, and Raimondo Schettini. Improving CNN-based texture classification by color balancing. *Journal of Imaging*, 3(3):1–15, 2017.

[53] Simone Bianco, Claudio Cusano, and Raimondo Schettini. Color constancy using CNNs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2015.

[54] Simone Bianco, Claudio Cusano, and Raimondo Schettini. Single and multiple illuminant estimation using convolutional neural networks. *IEEE Transactions on Image Processing*, 26(9):4347–4362, 2017.

[55] Simone Bianco and Raimondo Schettini. Color constancy using faces. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[56] Yochai Blau, Roey Mechrez, Radu Timofte, Tomer Michaeli, and Lihi Zelnik-Manor. The 2018 PIRM challenge on perceptual image super-resolution. In *European Conference on Computer Vision (ECCV) Workshops*, 2018.

[57] David H Brainard and William T Freeman. Bayesian color constancy. *Journal of the Optical Society of America A*, 14(7):1393–1411, 1997.

[58] David H Brainard and Brian A Wandell. Analysis of the retinex theory of color vision. *Journal of the Optical Society of America A*, 3(10):1651–1661, 1986.

[59] Tim Brooks, Ben Mildenhall, Tianfan Xue, Jiawen Chen, Dillon Sharlet, and Jonathan T Barron. Unprocessing images for learned raw denoising. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[60] Gershon Buchsbaum. A spatial processor model for object colour perception. *Journal of the Franklin Institute*, 310(1):1–26, 1980.

[61] Peter Burt and Edward Adelson. The Laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540, 1983.

[62] Vladimir Bychkovsky, Sylvain Paris, Eric Chan, and Frédo Durand. Learning photographic global tonal adjustment with a database of input/output image pairs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[63] Jianrui Cai, Shuhang Gu, and Lei Zhang. Learning a deep single image contrast enhancer from multi-exposure images. *IEEE Transactions on Image Processing*, 27(4):2049–2062, 2018.

[64] Hakki Can Karaimer and Michael S Brown. Improving color reproduction accuracy on cameras. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[65] Vlad C Cardei, Brian Funt, and Kobus Barnard. Estimating the scene illumination chromaticity by using a neural network. *Journal of the Optical Society of America A*, 19(12):2374–2386, 2002.

[66] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, 2017.

[67] Alexandra Carlson, Katherine A Skinner, and Matthew Johnson-Roberson. Modeling camera effects to improve deep vision for real and synthetic data. In *European Conference on Computer Vision (ECCV)*, 2018.

[68] Turgay Celik and Tardi Tjahjadi. Contextual and variational contrast enhancement. *IEEE Transactions on Image Processing*, 20(12):3431–3441, 2011.

[69] A. Chakrabarti, Ying Xiong, Baochen Sun, T. Darrell, D. Scharstein, T. Zickler, and K. Saenko. Modeling radiometric uncertainty for vision with tone-mapped color images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2185–2198, 2014.

[70] Huiwen Chang, Ohad Fried, Yiming Liu, Stephen DiVerdi, and Adam Finkelstein. Palette-based photo recoloring. *ACM Transactions on Graphics (TOG)*, 34(4):139:1–139:11, 2015.

[71] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference (BMVC)*, 2014.

[72] Chen Chen, Qifeng Chen, Jia Xu, and Vladlen Koltun. Learning to see in the dark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[73] Jiawen Chen, Andrew Adams, Neal Wadhwa, and Samuel W Hasinoff. Bilateral guided upsampling. *ACM Transactions on Graphics (TOG)*, 35(6):1–8, 2016.

[74] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFS. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2018.

[75] Yu-Sheng Chen, Yu-Ching Wang, Man-Hsin Kao, and Yung-Yu Chuang. Deep photo enhancer: Unpaired learning for image enhancement from photographs with GANs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[76] Dongliang Cheng, Abdelrahman Kamel, Brian Price, Scott Cohen, and Michael S Brown. Two illuminant estimation and user correction preference. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[77] Dongliang Cheng, Dilip K Prasad, and Michael S Brown. Illuminant estimation for color constancy: Why spatial-domain methods work and the role of the color distribution. *Journal of the Optical Society of America A*, 31(5):1049–1058, 2014.

[78] Dongliang Cheng, Brian Price, Scott Cohen, and Michael S Brown. Beyond white: Ground truth colors for color constancy correction. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[79] Dongliang Cheng, Brian Price, Scott Cohen, and Michael S Brown. Effective learning-based illuminant estimation using simple features. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[80] Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. StarGAN V2: Diverse image synthesis for multiple domains. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[81] Spencer Churchill. Anime face dataset. `https://www.kaggle.com/splcher/animefacedataset`. Accessed: 2021-01-15.

[82] C CIE. Commission internationale de l'eclairage proceedings, 1931. *Cambridge University, Cambridge*, 1932.

[83] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning (ICML)*, 2017.

[84] Chris Crawford and NAIN. Cat dataset. `https://www.kaggle.com/crawford/cat-dataset`. Accessed: 2021-01-15.

[85] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

[86] Duc-Tien Dang-Nguyen, Cecilia Pasquini, Valentina Conotter, and Giulia Boato. Raise: A raw images dataset for digital image forensics. In *ACM Multimedia Systems Conference*, 2015.

[87] Hal Daume III and Daniel Marcu. Domain adaptation for statistical classifiers. *Journal of Artificial Intelligence Research (JAIR)*, 26(1):101–126, 2006.

[88] Lisa DaNae Dayley and Brad Dayley. *Photoshop CS5 Bible*. John Wiley & Sons, 2010.

[89] Tiago José De Carvalho, Christian Riess, Elli Angelopoulou, Helio Pedrini, and Anderson de Rezende Rocha. Exposing digital image forgeries by illumination color classification. *IEEE Transactions on Information Forensics and Security*, 8(7):1182–1194, 2013.

[90] Paul E Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *Computer Graphics and Interactive Techniques*, 1997.

[91] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[92] Li Deng and Dong Yu. Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, 7(3–4):197–387, 2014.

[93] Emily L Denton, Soumith Chintala, arthur szlam, and Rob Fergus. Deep generative image models using a Laplacian pyramid of adversarial networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

[94] Aditya Deshpande, Jiajun Lu, Mao-Chuang Yeh, Min Jin Chong, and David Forsyth. Learning diverse image colorization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[95] Steven Diamond, Vincent Sitzmann, Stephen Boyd, Gordon Wetzstein, and Felix Heide. Dirty pixels: Optimizing image classification architectures for raw sensor data. *arXiv preprint arXiv:1701.06487*, 2017.

[96] Mauricio Díaz and Peter Sturm. Radiometric calibration using photo collections. In *IEEE International Conference on Computational Photography (ICCP)*, 2011.

[97] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[98] Mark S Drew, Graham D Finlayson, and Steven D Hordley. Recovery of chromaticity image free from shadows via illumination invariance. In *IEEE International Conference on Computer Vision (ICCV) Workshops*, 2003.

[99] S. R. Dubey, S. Chakraborty, S. K. Roy, S. Mukherjee, S. K. Singh, and B. B. Chaudhuri. diffGrad: An optimization method for convolutional neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 31(11):4500–4511, 2020.

[100] Marc Ebner. Color constancy using local color shifts. In *European Conference on Computer Vision (ECCV)*, 2004.

[101] Marc Ebner. *Color Constancy*. John Wiley & Sons, 2007.

[102] Eva Eibenberger and Elli Angelopoulou. The importance of the normalizing channel in log-chromaticity space. In *IEEE International Conference on Image Processing (ICIP)*, 2012.

[103] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[104] Gabriel Eilertsen, Joel Kronander, Gyorgy Denes, Rafał Mantiuk, and Jonas Unger. HDR image reconstruction from a single exposure using deep CNNs. *ACM Transactions on Graphics (TOG)*, 36(6):178:1–178:15, 2017.

[105] Majed El Helou, Ruofan Zhou, Johan Barthas, and Sabine Süsstrunk. VIDIT: Virtual image dataset for illumination transfer. *arXiv preprint arXiv:2005.05460*, 2020.

[106] Yuki Endo, Yoshihiro Kanamori, and Jun Mitani. Deep reverse tone mapping. *ACM Transactions on Graphics (TOG)*, 36(6):177:1–177:10, 2017.

[107] Mark D Fairchild. *Color Appearance Models*. John Wiley & Sons, 2013.

[108] H Sheikh Faridul, Tania Pouli, Christel Chamaret, Jürgen Stauder, Erik Reinhard, Dmitry Kuzovkin, and Alain Trémeau. Colour mapping: A review of recent methods, extensions and applications. In *Computer Graphics Forum*, 2016.

[109] Graham Finlayson, Han Gong, and Robert B Fisher. Color homography: Theory and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(1):20–33, 2019.

[110] Graham Finlayson and Steven Hordley. Improving gamut mapping color constancy. *IEEE Transactions on Image Processing*, 9(10):1774–1783, 2000.

[111] Graham D Finlayson. Corrected-moment illuminant estimation. In *IEEE International Conference on Computer Vision (ICCV)*, 2013.

[112] Graham D. Finlayson. Colour and illumination in computer vision. *Interface Focus*, 8(4):1–8, 2018.

[113] Graham D Finlayson, Mark S Drew, and Brian V Funt. Spectral sharpening: Sensor transformations for improved color constancy. *Journal of the Optical Society of America A*, 11(5):1553–1563, 1994.

[114] Graham D Finlayson, Brian V Funt, and Kobus Barnard. Color constancy under varying illumination. In *IEEE International Conference on Computer Vision (ICCV)*, 1995.

[115] Graham D Finlayson and Steven D Hordley. Color constancy at a pixel. *Journal of the Optical Society of America A*, 18(2):253–264, 2001.

[116] Graham D Finlayson, Steven D Hordley, and Paul M Hubel. Colour by correlation: A simple, unifying approach to colour constancy. In *IEEE International Conference on Computer Vision (ICCV)*, 1999.

[117] Graham D Finlayson, Steven D Hordley, and Ingeborg Tastl. Gamut constrained illuminant estimation. *International Journal of Computer Vision*, 67(1):93–109, 2006.

[118] Graham D Finlayson, Michal Mackiewicz, and Anya Hurlbert. Color correction using root-polynomial regression. *IEEE Transactions on Image Processing*, 24(5):1460–1470, 2015.

[119] Graham D Finlayson and Sabine Süsstrunk. Performance of a chromatic adaptation transform based on spectral sharpening. In *Color and Imaging Conference*, 2000.

[120] Graham D Finlayson and Elisabetta Trezzi. Shades of gray and colour constancy. In *Color and Imaging Conference*, 2004.

[121] Graham D Finlayson, Roshanak Zakizadeh, and Arjan Gijsenij. The reproduction angular error for evaluating the performance of illuminant estimation algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(7):1482–1488, 2017.

[122] David A Forsyth. A novel algorithm for color constancy. *International Journal of Computer Vision*, 5(1):5–35, 1990.

[123] Damien Fourure, Rémi Emonet, Elisa Fromont, Damien Muselet, Alain Trémeau, and Christian Wolf. Mixed pooling neural networks for color constancy. *IEEE International Conference on Image Processing (ICIP)*, 2016.

[124] Xueyang Fu, Delu Zeng, Yue Huang, Xiao-Ping Zhang, and Xinghao Ding. A weighted variational model for simultaneous reflectance and illumination estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[125] Brian Funt, Vlad Cardei, and Kobus Barnard. Learning color constancy. In *Color and Imaging Conference*, 1996.

[126] Brian Funt and Lilong Shi. The rehabilitation of maxRGB. In *Color and Imaging Conference*, 2010.

[127] Shao-Bing Gao, Ming Zhang, Chao-Yi Li, and Yong-Jie Li. Improving color constancy by discounting the variation of camera spectral sensitivity. *Journal of the Optical Society of America A*, 34(8):1448–1462, 2017.

[128] Shaobing Gao, Wangwang Han, Kaifu Yang, Chaoyi Li, and Yongjie Li. Efficient color constancy with local surface reflectance statistics. In *European Conference on Computer Vision (ECCV)*, 2014.

[129] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[130] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[131] Leon A Gatys, Alexander S Ecker, Matthias Bethge, Aaron Hertzmann, and Eli Shechtman. Controlling perceptual factors in neural style transfer. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[132] Peter Vincent Gehler, Carsten Rother, Andrew Blake, Tom Minka, and Toby Sharp. Bayesian color constancy revisited. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[133] Theo Gevers and Arnold WM Smeulders. PicToSeek: Combining color and shape invariant features for image retrieval. *IEEE Transactions on Image Processing*, 9(1):102–119, 2000.

[134] Michaël Gharbi, Jiawen Chen, Jonathan T Barron, Samuel W Hasinoff, and Frédo Durand. Deep bilateral learning for real-time image enhancement. *ACM Transactions on Graphics (TOG)*, 36(4):118:1–118:12, 2017.

[135] Arjan Gijsenij and Theo Gevers. Color constancy using natural image statistics and scene semantics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(4):687–698, 2011.

[136] Arjan Gijsenij, Theo Gevers, and Joost Van De Weijer. Generalized gamut mapping using image derivative structures for color constancy. *International Journal of Computer Vision*, 86(2-3):127–139, 2010.

[137] Arjan Gijsenij, Theo Gevers, and Joost Van De Weijer. Computational color constancy: Survey and experiments. *IEEE Transactions on Image Processing*, 20(9):2475–2489, 2011.

[138] Arjan Gijsenij, Theo Gevers, and Joost Van De Weijer. Computational color constancy: Survey and experiments. *IEEE Transactions on Image Processing*, 20(9):2475–2489, 2011.

[139] Arjan Gijsenij, Theo Gevers, and Joost Van De Weijer. Improving color constancy by photometric edge weighting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(5):918–929, 2012.

[140] Arjan Gijsenij, Theo Gevers, and Joost Van De Weijer. Improving color constancy by photometric edge weighting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(5):918–929, 2012.

[141] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

[142] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[143] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2014.

[144] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[145] Hermann Grassmann. Zur theorie der farbenmischung. *Annalen der Physik*, 165(5):69–84, 1853.

[146] M.D. Grossberg and S.K. Nayar. What is the Space of Camera Response Functions? In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2003.

[147] Michael D Grossberg and Shree K Nayar. Determining the camera response from images: What is knowable? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(11):1455–1467, 2003.

[148] Michael D Grossberg and Shree K Nayar. Modeling the space of camera response functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1272–1282, 2004.

[149] Bahadir K Gunturk, Yucel Altunbasak, and Russell M Mersereau. Color plane interpolation using alternating projections. *IEEE Transactions on Image Processing*, 11(9):997–1013, 2002.

[150] Chunle Guo, Chongyi Li, Jichang Guo, Chen Change Loy, Junhui Hou, Sam Kwong, and Runmin Cong. Zero-reference deep curve estimation for low-light image enhancement. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[151] Xiaojie Guo. LIME: A method for low-light image enhancement. In *ACM International Conference on Multimedia*, 2016.

[152] Xiaojie Guo, Yu Li, and Haibin Ling. LIME: Low-light image enhancement via illumination map estimation. *IEEE Transactions on Image Processing*, 26(2):982–993, 2017.

[153] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[154] Lin Haiting. A new in-camera color imaging model for computer vision. *Ph. D. Thesis, National University of Singapore*, 2013.

[155] Samuel W Hasinoff, Dillon Sharlet, Ryan Geiss, Andrew Adams, Jonathan T Barron, Florian Kainz, Jiawen Chen, and Marc Levoy. Burst photography for high dynamic range and low-light imaging on mobile cameras. *ACM Transactions on Graphics (TOG)*, 35(6):1–12, 2016.

[156] Søren Hauberg, Oren Freifeld, Anders Boesen Lindbo Larsen, John Fisher, and Lars Hansen. Dreaming more data: Class-dependent distributions over diffeomorphisms for learned data augmentation. In *Artificial Intelligence and Statistics*, 2016.

[157] Kaiming He, Jian Sun, and Xiaoou Tang. Single image haze removal using dark channel prior. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(12):2341–2353, 2010.

[158] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[159] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[160] Mingming He, Jing Liao, Dongdong Chen, Lu Yuan, and Pedro V Sander. Progressive color transfer with dense semantic correspondences. *ACM Transactions on Graphics (TOG)*, 38(2):1–18, 2019.

[161] Mingming He, Jing Liao, Lu Yuan, and Pedro V Sander. Neural color transfer between images. *arXiv preprint arXiv:1710.00756*, 2017.

[162] Majed El Helou, Ruofan Zhou, Sabine Süsstrunk, Radu Timofte, Mahmoud Afifi, Michael S Brown, Kele Xu, Hengxing Cai, Yuzhong Liu, Li-Wen Wang, et al. AIM 2020: Scene relighting and illumination estimation challenge. In *European Conference on Computer Vision (ECCV) Workshops*, 2020.

[163] Daniel Hernandez-Juarez, Sarah Parisot, Benjamin Busam, Ales Leonardis, Gregory Slabaugh, and Steven McDonagh. A multi-hypothesis approach to color constancy. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[164] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[165] Yannick Hold-Geoffroy, Kalyan Sunkavalli, Sunil Hadap, Emiliano Gambaretto, and Jean-François Lalonde. Deep outdoor illumination estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[166] Guowei Hong, M Ronnier Luo, and Peter A Rhodes. A study of digital camera colorimetric characterisation based on polynomial modelling. *Color Research & Application*, 26(1):76–84, 2001.

[167] Steven D Hordley. Scene illuminant estimation: Past, present, and future. *Color Research & Application*, 31(4):303–314, 2006.

[168] Steven D Hordley and Graham D Finlayson. Re-evaluating colour constancy algorithms. In *ICPR*, 2004.

[169] Eugene Hsu, Tom Mertens, Sylvain Paris, Shai Avidan, and Frédo Durand. Light mixture estimation for spatially varying white balance. In *ACM Transactions on Graphics (TOG)*, volume 27, pages 70:1–70:8, 2008.

[170] Yuanming Hu, Hao He, Chenxi Xu, Baoyuan Wang, and Stephen Lin. Exposure: A white-box photo post-processing framework. *ACM Transactions on Graphics (TOG)*, 37(2):26:1–26:17, 2018.

[171] Yuanming Hu, Baoyuan Wang, and Stephen Lin. FC4: Fully convolutional color constancy with confidence-weighted pooling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[172] H-Z Huang, S-H Zhang, RR Martin, and S-M Hu. Learning natural colors for image recoloring. In *Computer Graphics Forum*, 2014.

[173] Jun-yan Huo, Yi-lin Chang, Jing Wang, and Xiao-xia Wei. Robust automatic white balance algorithm using gray color points in images. *IEEE Transactions on Consumer Electronics*, 52(2):541–546, 2006.

[174] Md Akmol Hussain and Akbar Sheikh Akbari. Color constancy algorithm for mixed-illuminant scene images. *IEEE Access*, 6:8964–8976, 2018.

[175] Andrey Ignatov, Nikolay Kobyshev, Radu Timofte, Kenneth Vanhoey, and Luc Van Gool. DSLR-quality photos on mobile devices with deep convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[176] Andrey Ignatov, Nikolay Kobyshev, Radu Timofte, Kenneth Vanhoey, and Luc Van Gool. WESPE: Weakly supervised photo enhancer for digital cameras. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2018.

[177] Andrey Ignatov, Luc Van Gool, and Radu Timofte. Replacing mobile camera isp with a single deep learning model. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2020.

[178] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[179] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[180] Dorothea Jameson and Leo M Hurvich. Essay concerning color constancy. *Annual Review of Psychology*, 40(1):1–24, 1989.

[181] Haomiao Jiang, Qiyuan Tian, Joyce Farrell, and Brian A Wandell. Learning the image processing pipeline. *IEEE Transactions on Image Processing*, 26(10):5032–5042, 2017.

[182] Jun Jiang, Dengyu Liu, Jinwei Gu, and Sabine Süsstrunk. What is the space of spectral sensitivity functions for digital color cameras? *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2013.

[183] Yifan Jiang, Xinyu Gong, Ding Liu, Yu Cheng, Chen Fang, Xiaohui Shen, Jianchao Yang, Pan Zhou, and Zhangyang Wang. EnlightenGAN: Deep light enhancement without paired supervision. *arXiv preprint arXiv:1906.06972*, 2019.

[184] Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, Yizhou Yu, and Mingli Song. Neural style transfer: A review. *IEEE Transactions on Visualization and Computer Graphics*, 26(11):3365–3385, 2019.

[185] Daniel J Jobson, Ziaur Rahman, and Glenn A Woodell. A multiscale retinex for bridging the gap between color images and the human observation of scenes. *IEEE Transactions on Image Processing*, 6(7):965–976, 1997.

[186] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision (ECCV)*, 2016.

[187] Hamid Reza Vaezi Joze, Mark S Drew, Graham D Finlayson, and Perla Aurora Troncoso Rey. The role of bright pixels in illumination estimation. In *Color and Imaging Conference*, 2012.

[188] Chuan kai Lin. Pixel grouping. `https://sites.google.com/site/chklin/demosaic`. Accessed: 2021-01-15.

[189] Praveen Kakumanu, Sokratis Makrogiannis, and Nikolaos Bourbakis. A survey of skin-color modeling and detection methods. *Pattern Recognition*, 40(3):1106–1122, 2007.

[190] Nima Khademi Kalantari and Ravi Ramamoorthi. Deep high dynamic range imaging of dynamic scenes. *ACM Transactions on Graphics (TOG)*, 36(4):144–1, 2017.

[191] Hakki Can Karaimer and Michael S Brown. A software platform for manipulating the camera imaging pipeline. In *European Conference on Computer Vision (ECCV)*, 2016.

[192] Hakki Can Karaimer and Michael S Brown. Improving color reproduction accuracy on cameras. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[193] Hakki Can Karaimer and Michael S Brown. Beyond raw-RGB and sRGB: Advocating access to a colorimetric image state. *Color and Imaging Conference*, 2019.

[194] Hakki Can Karaimer, Iman Khodadad, Farnoud Kazemzadeh, and Michael S Brown. A customized camera imaging pipeline for dermatological imaging. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2019.

[195] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[196] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[197] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[198] Rei Kawakami, Katsushi Ikeuchi, and Robby T Tan. Consistent surface color for texturing large objects in outdoor scenes. In *IEEE International Conference on Computer Vision (ICCV)*, 2005.

[199] Scott Kelby. *The Adobe Photoshop Lightroom 5 Book for Digital Photographers*. Pearson Education, 2014.

[200] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Fei-Fei Li. Novel dataset for fine-grained image categorization: Stanford dogs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2011.

[201] Seon Joo Kim, Jan-Michael Frahm, and Marc Pollefeys. Radiometric calibration with illumination change for outdoor scene analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[202] Seon Joo Kim, Hai Ting Lin, Zheng Lu, S. Süsstrunk, S. Lin, and Michael S. Brown. A new in-camera imaging model for color computer vision and its application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(12):2289–2302, 2012.

[203] Seon Joo Kim and M. Pollefeys. Robust radiometric calibration and vignetting correction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(4):562–576, 2008.

[204] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[205] Karlo Koščević, Nikola Banić, and Sven Lončarić. Color beaver: Bounding illumination estimations for higher accuracy. In *VISIGRAPP*, 2019.

[206] Samu Koskinen12, Dan Yang, and Joni-Kristian Kämäräinen. Cross-dataset color constancy revisited using sensor-to-sensor transfer. In *British Machine Vision Conference (BMVC)*, 2020.

[207] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3D object representations for fine-grained categorization. In *IEEE International Conference on Computer Vision (ICCV) Workshops*, 2013.

[208] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, 2009.

[209] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2012.

[210] Arie Andries Kruithof. Tubular luminescence lamps for general illumination. *Philips Technical Review*, 6:65–96, 1941.

[211] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *Technical report, Google, Inc.*, 2016.

[212] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, et al. The open images dataset V4. *International Journal of Computer Vision*, pages 1–26, 2020.

[213] Firas Laakom, Nikolaos Passalis, Jenni Raitoharju, Jarno Nikkanen, Anastasios Tefas, Alexandros Iosifidis, and Moncef Gabbouj. Bag of color features for color constancy. *IEEE Transactions on Image Processing*, 29:7722–7734, 2020.

[214] Firas Laakom, Jenni Raitoharju, Alexandros Iosifidis, Jarno Nikkanen, and Moncef Gabbouj. Color constancy convolutional autoencoder. *Symposium Series on Computational Intelligence*, 2019.

[215] Firas Laakom, Jenni Raitoharju, Alexandros Iosifidis, Jarno Nikkanen, and Moncef Gabbouj. Intel-TAU: A color constancy dataset. *arXiv preprint arXiv:1910.10404*, 2019.

[216] Pierre-Yves Laffont, Zhile Ren, Xiaofeng Tao, Chao Qian, and James Hays. Transient attributes for high-level understanding and editing of outdoor scenes. *ACM Transactions on Graphics (TOG)*, 33(4):149:1–149:11, 2014.

[217] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. Deep Laplacian pyramid networks for fast and accurate super-resolution. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[218] Hong-Kwai Lam, OS Au, and Chi-Wah Wong. Automatic white balancing using standard deviation of RGB components. In *International Symposium on Circuits and Systems*, 2004.

[219] King Man Lam. Metamerism and Colour Constancy. *Ph. D. Thesis, University of Bradford*, 1985.

[220] Edwin H Land. The retinex theory of color vision. *Scientific American*, 237(6):108–129, 1977.

[221] Hoang Le, Mahmoud Afifi, and Michael S Brown. Improving color space conversion for camera-captured images via wide-gamut metadata. In *Color and Imaging Conference*, 2020.

[222] Chulwoo Lee, Chul Lee, and Chang-Su Kim. Contrast enhancement based on layered difference representation. In *IEEE International Conference on Image Processing (ICIP)*, 2012.

[223] Chulwoo Lee, Chul Lee, and Chang-Su Kim. Contrast enhancement based on layered difference representation of 2D histograms. *IEEE Transactions on Image Processing*, 22(12):5372–5384, 2013.

[224] H. Lee, K. Sohn, and D. Min. Unsupervised low-light image enhancement using bright channel prior. *IEEE Signal Processing Letters*, 27:251–255, 2020.

[225] Hsin-Ying Lee, Jia-Bin Huang, Maneesh Singh, and Ming-Hsuan Yang. Unsupervised representation learning by sorting sequences. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[226] Joon-Young Lee, Kalyan Sunkavalli, Zhe Lin, Xiaohui Shen, and In So Kweon. Automatic content-aware color and tone stylization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[227] Changjun Li, Guihua Cui, Manuel Melgosa, Xiukai Ruan, Yaoju Zhang, Long Ma, Kaida Xiao, and M Ronnier Luo. Accurate method for computing correlated color temperature. *Optics Express*, 2016.

[228] Changjun Li, M Ronnier Luo, Bryan Rigg, and Robert WG Hunt. CMC 2000 chromatic adaptation transform: CMCCAT2000. *Color Research & Application*, 27(1):49–58, 2002.

[229] Chen Li, Stephen Lin, Kun Zhou, and Katsushi Ikeuchi. Radiometric calibration from faces in images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[230] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal style transfer via feature transforms. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[231] Yijun Li, Ming-Yu Liu, Xueting Li, Ming-Hsuan Yang, and Jan Kautz. A closed-form solution to photorealistic image stylization. In *European Conference on Computer Vision (ECCV)*, 2018.

[232] Ying-Yi Li and Hsien-Che Lee. Auto white balance by surface reflection decomposition. *Journal of the Optical Society of America A*, 34(10):1800–1809, 2017.

[233] Yuheng Li, Krishna Kumar Singh, Utkarsh Ojha, and Yong Jae Lee. MixNMatch: Multifactor disentanglement and encoding for conditional image generation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[234] C. H. Liang, Y. A. Chen, Y. C. Liu, and W. Hsu. Raw image deblurring. *IEEE Transactions on Multimedia*, Early Access:1–12, 2020.

[235] Orly Liba, Kiran Murthy, Yun-Ta Tsai, Tim Brooks, Tianfan Xue, Nikhil Karnad, Qiurui He, Jonathan T Barron, Dillon Sharlet, Ryan Geiss, Samuel W Hasinoff, Yael Pritch, and Marc Levoy. Handheld mobile photography in very low light. *ACM Transactions on Graphics (TOG)*, 38(6):1–16, 2019.

[236] Guosheng Lin, Fayao Liu, Anton Milan, Chunhua Shen, and Ian Reid. RefineNet: Multi-path refinement networks for dense prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Early Access:1–15, 2019.

[237] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian Reid. RefineNet: Multi-path refinement networks for high-resolution semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[238] Haiting Lin, Seon Joo Kim, Sabine Susstrunk, and Michael S Brown. Revisiting radiometric calibration for color computer vision. In *IEEE International Conference on Computer Vision (ICCV)*, 2011.

[239] S. Lin, Jinwei Gu, S. Yamazaki, and Heung-Yeung Shum. Radiometric calibration from a single image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004.

[240] Stephen Lin and Lei Zhang. Determining the radiometric response function from a single grayscale image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

[241] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision (ECCV)*, 2014.

[242] Huidong Liu, Xianfeng Gu, and Dimitris Samaras. Wasserstein GAN with quadratic transport cost. In *IEEE International Conference on Computer Vision (ICCV)*, 2019.

[243] Lin Liu, Xu Jia, Jianzhuang Liu, and Qi Tian. Joint demosaicing and denoising with self guidance. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[244] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks

and the coordconv solution. *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[245] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. In *International Conference on Learning Representations (ICLR)*, 2017.

[246] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[247] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[248] Zhongyu Lou, Theo Gevers, Ninghang Hu, Marcel P Lucassen, et al. Color constancy by deep learning. In *British Machine Vision Conference (BMVC)*, 2015.

[249] David G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[250] F. Luan, S. Paris, E. Shechtman, and K. Bala. Deep photo style transfer. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[251] Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. Deep photo style transfer. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[252] Atima Lui, Nyalia Lui, Mahmoud Afifi, and Ariadne Bazigos. System and method for color matching, 2020. US Patent 10,571,336.

[253] M Ronnier Luo and Peter A Rhodes. Corresponding-colour datasets. *Color Research & Application*, 24(4):295–296, 1999.

[254] Chao Ma, Chih-Yuan Yang, Xiaokang Yang, and Ming-Hsuan Yang. Learning a no-reference quality metric for single-image super-resolution. *Computer Vision and Image Understanding*, 158:1–16, 2017.

[255] Liqian Ma, Xu Jia, Qianru Sun, Bernt Schiele, Tinne Tuytelaars, and Luc Van Gool. Pose guided person image generation. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[256] Ruijun Ma, Haifeng Hu, Songlong Xing, and Zhengming Li. Efficient and fast real-world noisy image denoising by combining pyramid neural network and two-pathway unscented Kalman filter. *IEEE Transactions on Image Processing*, 29(1):3927–3940, 2020.

[257] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning (ICML)*, 2013.

[258] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[259] Emmanuel Maggiori, Yuliya Tarabalka, Guillaume Charpiat, and Pierre Alliez. Can semantic labeling methods generalize to any city? The inria aerial image labeling benchmark. In *International Geoscience and Remote Sensing Symposium (IGARSS)*, 2017.

[260] Laurence T Maloney. Physics-based approaches to modeling surface color perception. *Color Vision: From Genes to Perception*, pages 387–416, 1999.

[261] S Mann and R Picard. Being undigital with digital cameras. *MIT Media Lab Perceptual*, 1:1–7, 1994.

[262] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018.

[263] Steven McDonagh, Sarah Parisot, Zhenguo Li, and Gregory Slabaugh. Meta-learning for few-shot camera-adaptive color constancy. *arXiv preprint arXiv:1811.11788*, 2018.

[264] Steven McDonagh, Sarah Parisot, Fengwei Zhou, Xing Zhang, Ales Leonardis, Zhenguo Li, and Gregory Slabaugh. Formulating camera-adaptive color constancy as a few-shot meta-learning problem. *arXiv preprint arXiv:1811.11788*, 2018.

[265] Tom Mertens, Jan Kautz, and Frank Van Reeth. Exposure fusion: A simple and practical alternative to high dynamic range photography. In *Computer Graphics Forum*, 2009.

[266] Laurence Meylan and Sabine Susstrunk. High dynamic range image rendering with a retinex-based adaptive filter. *IEEE Transactions on Image Processing*, 15(9):2820–2830, 2006.

[267] Peyman Milanfar. A tour of modern image filtering: New insights and methods, both practical and theoretical. *IEEE Signal Processing Magazine*, 30(1):106–128, 2013.

[268] Tomoo Mitsunaga and Shree K Nayar. Radiometric self calibration. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1999.

[269] Anish Mittal, Rajiv Soundararajan, and Alan C Bovik. Making a "completely blind" image quality analyzer. *IEEE Signal Processing Letters*, 20(3):209–212, 2012.

[270] Zhipeng Mo, Boxin Shi, Sai-Kit Yeung, and Yasuyuki Matsushita. Radiometric calibration for Internet photo collections. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[271] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[272] Sean Moran, Pierre Marza, Steven McDonagh, Sarah Parisot, and Gregory Slabaugh. DeepLPF: Deep local parametric filters for image enhancement. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[273] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks. `https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html`. Accessed: 2021-01-15.

[274] Kenta Moriwaki, Ryota Yoshihashi, Rei Kawakami, Shaodi You, and Takeshi Naemura. Hybrid loss for learning single-image-based HDR reconstruction. *arXiv preprint arXiv:1812.07134*, 2018.

[275] Ján Morovč. To develop a universal gamut mapping algorithm. *Ph. D. Thesis, University of Derby*, 1998.

[276] Mukesh C Motwani, Mukesh C Gadiya, Rakhi C Motwani, and Frederick C Harris. Survey of image denoising techniques. In *Proceedings of GSPX*, 2004.

[277] Yair Movshovitz-Attias, Takeo Kanade, and Yaser Sheikh. How useful is photo-realistic rendering for visual learning? In *European Conference on Computer Vision (ECCV)*, 2016.

[278] Junichi Nakamura. *Image sensors and signal processing for digital still cameras*. CRC press, 2017.

[279] Hyeonseob Nam and Hyo-Eun Kim. Batch-instance normalization for adaptively style-invariant neural networks. *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[280] Seonghyeon Nam and Seon Joo Kim. Modelling the scene dependent imaging in cameras with a deep neural network. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[281] Shree K Nayar and Tomoo Mitsunaga. High dynamic range imaging: Spatially varying pixel exposures. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2000.

[282] Yoshinobu Nayatani, Kotaro Takahama, and Hiroaki Sobagaki. Formulation of a nonlinear model of chromatic adaptation. *Color Research & Application*, 6(3):161–171, 1981.

[283] R. M. H. Nguyen, D. K. Prasad, and M. S. Brown. Raw-to-raw: Mapping between image sensor color responses. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[284] Rang Ho Man Nguyen and Michael S Brown. Why you should forget luminance conversion and do something better. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[285] Rang MH Nguyen and Michael S Brown. Raw image reconstruction using a self-contained sRGB-JPEG image with only 64 KB overhead. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[286] Rang MH Nguyen and Michael S Brown. Raw image reconstruction using a self-contained sRGB–JPEG image with small memory overhead. *International Journal of Computer Vision*, 126(6):637–650, 2018.

[287] Rang MH Nguyen, Seon Joo Kim, and Michael S Brown. Illuminant aware gamut-based color transfer. In *Computer Graphics Forum*, 2014.

[288] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics & Image Processing*, 2008.

[289] Seoung Wug Oh and Seon Joo Kim. Approaching the computational color constancy as a classification problem through deep learning. *Pattern Recognition*, 61:405–416, 2017.

[290] Claudio Oleari. Corresponding color datasets and a chromatic adaptation model based on the OSA-UCS system. *Journal of the Optical Society of America A*, 31(7):1502–1514, 2014.

[291] Maria Olkkonen, Thorsten Hansen, and Karl R Gegenfurtner. Categorical color constancy for simulated surfaces. *Journal of Vision*, 9(12):6–6, 2009.

[292] Dejana Đorđević, Aleš Hladnik, and Andrej Javoršek. Performance of five chromatic adaptation transforms using large number of color patches. *Acta graphica: znanstveni časopis za tiskarstvo i grafičke komunikacije*, 20(1-4):9–19, 2010.

[293] Dejana Đorđević, Andrej Javoršek, and Aleš Hladnik. Comparison of chromatic adaptation transforms used in textile printing sample preparation. *Coloration Technology*, 126(5):275–281, 2010.

[294] Chris Pal, Richard Szeliski, Matthew Uyttendaele, and Nebojsa Jojic. Probability models for high dynamic range imaging. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004.

[295] Bin Pan, Zhiguo Jiang, Haopeng Zhang, Xiaoyan Luo, and Junfeng Wu. Improved grey world color correction method based on weighted gain coefficients. In *Optoelectronic Imaging and Multimedia Technology*, 2014.

[296] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[297] Sylvain Paris, Samuel W Hasinoff, and Jan Kautz. Local Laplacian filters: Edge-aware image processing with a Laplacian pyramid. *ACM Transactions on Graphics (TOG)*, 30(4):68:1–68:12, 2011.

[298] Jongchan Park, Joon-Young Lee, Donggeun Yoo, and In So Kweon. Distort-and-recover: Color enhancement using deep reinforcement learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[299] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

[300] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *SIGGRAPH*. 2003.

[301] Bryan Peterson. *Understanding exposure: How to shoot great photographs with any camera*. AmPhoto Books, 2016.

[302] Andrius Petrulis, Linas Petkevičius, Pranciškus Vitta, Rimantas Vaicekauskas, and Artūras Žukauskas. Exploring preferred correlated color temperature in outdoor

environments using a smart solid-state light engine. *The Journal of the Illuminating Engineering Society*, 14(2):95–106, 2018.

[303] Stanislav Pidhorskyi, Donald A Adjeroh, and Gianfranco Doretto. Adversarial latent autoencoders. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[304] F. Pitié and A. Kokaram. The linear Monge-Kantorovitch linear colour mapping for example-based colour transfer. In *European Conference on Visual Media Production*, 2007.

[305] Francois Pitié, Anil C Kokaram, and Rozenn Dahyot. N-dimensional probability density function transfer and its application to color transfer. In *IEEE International Conference on Computer Vision (ICCV)*, 2005.

[306] François Pitié, Anil C Kokaram, and Rozenn Dahyot. Automated colour grading using colour distribution transfer. *Computer Vision and Image Understanding*, 107(1-2):123–137, 2007.

[307] Stephen M Pizer, E Philip Amburn, John D Austin, Robert Cromartie, Ari Geselowitz, Trey Greer, Bart ter Haar Romeny, John B Zimmerman, and Karel Zuiderveld. Adaptive histogram equalization and its variations. *Computer Vision, Graphics, and Image Processing*, 39(3):355–368, 1987.

[308] David Pollard. *A User's Guide to Measure Theoretic Probability*. Cambridge University Press, 2002.

[309] Charles Poynton. *Digital Video and HD: Algorithms and Interfaces*. Elsevier, 2012.

[310] Ralph W Pridmore. Theory of corresponding colors as complementary sets. *Color Research & Application*, 30(5):371–381, 2005.

[311] Abhijith Punnappurath and Michael S Brown. Learning raw image reconstruction-aware deep image compressors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP(99):1–8, 2019.

[312] Yanlin Qian, Ke Chen, and Huanglin Yu. Fast fourier color constancy and grayness index for ISPA illumination estimation challenge. *International Symposium on Image and Signal Processing and Analysis (ISPA)*, 2019.

[313] Yanlin Qian, Jarno Nikkanen, Joni-Kristian Kämäräinen, and Jiri Matas. On finding gray pixels. *arXiv preprint arXiv:1901.03198*, 2019.

[314] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[315] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2013.

[316] Rajeev Ramanath, Wesley E Snyder, Youngjun Yoo, and Mark S Drew. Color image processing pipeline. *IEEE Signal Processing Magazine*, 22(1):34–43, 2005.

[317] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[318] Erik Reinhard, Michael Adhikhmin, Bruce Gooch, and Peter Shirley. Color transfer between images. *IEEE Computer Graphics and Applications*, 21(5):34–41, 2001.

[319] Alexander Jung Revision. Imgaug library. Accessed: 2021-01-15.

[320] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *European Conference on Computer Vision (ECCV)*.

[321] Christian Riess, Eva Eibenberger, and Elli Angelopoulou. Illuminant color estimation for real-world mixed-illuminant scenes. In *IEEE International Conference on Computer Vision (ICCV) Workshops*, 2011.

[322] Eric Risser, Pierre Wilmot, and Connelly Barnes. Stable and controllable neural texture synthesis and style transfer using histogram losses. *arXiv preprint arXiv:1701.08893*, 2017.

[323] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. *International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2015.

[324] Charles Rosenberg, Martial Hebert, and Sebastian Thrun. Color constancy using KL-divergence. In *IEEE International Conference on Computer Vision (ICCV)*, 2001.

[325] Charles Rosenberg, Alok Ladsariya, and Tom Minka. Bayesian color constancy with non-gaussian models. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2004.

[326] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[327] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. *European Conference on Computer Vision (ECCV)*, 2010.

[328] Kuniaki Saito, Kate Saenko, and Ming-Yu Liu. COCO-FUNIT: Few-shot unsupervised image translation with a content conditioned style encoder. In *European Conference on Computer Vision (ECCV)*, 2020.

[329] A Savchik, E Ershov, and S Karpenko. Color cerberus. *International Symposium on Image and Signal Processing and Analysis (ISPA)*, 2019.

[330] Jeff Schewe. *The digital negative: Raw image processing in Lightroom, Camera Raw, and Photoshop*. Peachpit Press, 2015.

[331] Jeff Schewe and Bruce Fraser. *Real World Camera Raw with Adobe Photoshop CS5*. Pearson Education, 2010.

[332] Eli Schwartz, Raja Giryes, and Alex M Bronstein. DeepISP: Toward learning an end-to-end image processing pipeline. *IEEE Transactions on Image Processing*, 28(2):912–923, 2018.

[333] Michael Scuello, Israel Abramov, James Gordon, and Steven Weintraub. Museum lighting: Why are some illuminants preferred? *Journal of the Optical Society of America A*, 21(2):306–311, 2004.

[334] Steven A Shafer. Using color to separate reflection components. *Color Research & Application*, 10(4):210–218, 1985.

[335] Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. SinGAN: Learning a generative model from a single natural image. In *IEEE International Conference on Computer Vision (ICCV)*, 2019.

[336] Gaurav Sharma, Wencheng Wu, and Edul N Dalal. The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations. *Color Research & Application*, 30(1):21–30, 2005.

[337] Lu Sheng, Ziyi Lin, Jing Shao, and Xiaogang Wang. Avatar-Net: Multi-scale zero-shot style transfer by feature decoration. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[338] Wu Shi, Chen Change Loy, and Xiaoou Tang. Deep specialized network for illuminant estimation. In *European Conference on Computer Vision (ECCV)*, 2016.

[339] Yichang Shih, Sylvain Paris, Frédo Durand, and William T Freeman. Data-driven hallucination of different times of day from a single outdoor photo. *ACM Transactions on Graphics (TOG)*, 32(6):1–11, 2013.

[340] Assaf Shocher, Yossi Gandelsman, Inbar Mosseri, Michal Yarom, Michal Irani, William T Freeman, and Tali Dekel. Semantic pyramid for image generation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[341] Maria Shugrina, Amlan Kar, Sanja Fidler, and Karan Singh. Nonlinear color triads for approximation, learning and direct manipulation of color distributions. *ACM Transactions on Graphics (TOG)*, 39(4):97–1, 2020.

[342] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[343] Asim Smailagic, Anupma Sharan, Pedro Costa, Adrian Galdran, Alex Gaudio, and Aurélio Campilho. Learned pre-processing for automatic diabetic retinopathy detection on eye fundus images. In *International Conference on Image Analysis and Recognition*, 2019.

[344] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[345] Jake Snell, Kevin Swersky, and Richard S Zemel. Prototypical networks for few-shot learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[346] Andrew Stockman. Physiologically-based color matching functions. In *Color and Imaging Conference*, 2008.

[347] Sabine Süsstrunk and Graham D Finlayson. Evaluating chromatic adaptation transform performance. In *Color and Imaging Conference*, 2005.

[348] Sabine E Susstrunk, Jack M Holm, and Graham D Finlayson. Chromatic adaptation performance of different RGB sensors. In *Color Imaging: Device-Independent Color, Color Hardcopy, and Graphic Arts VI*, volume 4300, pages 172–184, 2000.

[349] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[350] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.

[351] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer Science & Business Media, 2010.

[352] Shen-Chuan Tai, Tzu-Wen Liao, Yi-Ying Chang, and Chih-Pei Yeh. Automatic white balance algorithm through the average equalization and threshold. In *International Conference on Information and Digital Technologies (ICIDT)*, 2012.

[353] Yu-Wing Tai, Xiaogang Chen, Sunyeong Kim, Seon Joo Kim, Feng Li, Jie Yang, Jingyi Yu, Yasuyuki Matsushita, and Michael S Brown. Nonlinear camera response

functions and image deblurring: Theoretical analysis and practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(10):2498–2512, 2013.

[354] Jianchao Tan, Jose Echevarria, and Yotam Gingold. Efficient palette-based decomposition and recoloring of images via RGBXY-space geometry. *ACM Transactions on Graphics (TOG)*, 37(6):1–10, 2018.

[355] Jianchao Tan, Jyh-Ming Lien, and Yotam Gingold. Decomposing images into layers via RGB-space geometry. *ACM Transactions on Graphics (TOG)*, 36(1):1–14, 2016.

[356] Robby T Tan, Katsushi Ikeuchi, and Ko Nishino. Color constancy through inverse-intensity chromaticity space. In *Digitally Archiving Cultural Objects*, volume 21, pages 323–351. 2008.

[357] Mehrdad Panahpour Tehrani, Akio Ishikawa, Shigeyuki Sakazawa, and Atsushi Koike. Iterative colour correction of multicamera systems using corresponding feature points. *Journal of Visual Communication and Image Representation*, 21(5-6):377–391, 2010.

[358] Radu Timofte, Shuhang Gu, Jiqing Wu, Luc Van Gool, Lei Zhang, Ming-Hsuan Yang, Muhammad Haris, et al. NTIRE 2018 challenge on single image super-resolution: Methods and results. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2018.

[359] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *International Conference on Machine Learning (ICML)*, 2016.

[360] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[361] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[362] Joost Van De Weijer, Theo Gevers, and Arjan Gijsenij. Edge-based color constancy. *IEEE Transactions on Image Processing*, 16(9):2207–2214, 2007.

[363] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[364] VSR Veeravasarapu, Constantin Rothkopf, and Ramesh Visvanathan. Adversarially tuned scene generation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[365] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2016.

[366] Vassilios Vonikakis. Busting image enhancement and tone-mapping algorithms. `https://sites.google.com/site/vonikakis/datasets`. Accessed: 2021-01-15.

[367] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The Caltech-UCSD birds-200-2011 dataset. 2011.

[368] John Walker. Colour rendering of spectra. `https://www.fourmilab.ch/documents/specrend/`. Accessed: 2021-01-15.

[369] Ruixing Wang, Qing Zhang, Chi-Wing Fu, Xiaoyong Shen, Wei-Shi Zheng, and Jiaya Jia. Underexposed photo enhancement using deep illumination estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[370] Shuhang Wang, Jin Zheng, Hai-Miao Hu, and Bo Li. Naturalness preserved enhancement algorithm for non-uniform illumination images. *IEEE Transactions on Image Processing*, 22(9):3538–3548, 2013.

[371] Su Wang, Yewei Zhang, Peng Deng, and Fuqiang Zhou. Fast automatic white balancing method by color histogram stretching. In *International Congress on Image and Signal Processing*, 2011.

[372] Yuzhi Wang, Haibin Huang, Qin Xu, Jiaming Liu, Yiqun Liu, and Jue Wang. Practical deep raw image denoising on mobile devices. In *European Conference on Computer Vision*, 2020.

[373] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[374] Joe H Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.

[375] Chen Wei, Wenjing Wang, Wenhan Yang, and Jiaying Liu. Deep retinex decomposition for low-light enhancement. In *British Machine Vision Conference (BMVC)*, 2018.

[376] S. Woo, S. Lee, J. Yoo, and J. Kim. Improving color constancy in an ambient light environment using the phong reflection model. *IEEE Transactions on Image Processing*, 27(4):1862–1877, 2017.

[377] William David Wright. A re-determination of the trichromatic coefficients of the spectral colours. *Transactions of the Optical Society*, 30(4):141–164, 1929.

[378] Gunter Wyszecki and Walter Stanley Stiles. *Color science*, volume 8. Wiley New York, 1982.

[379] Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, Mingyan Liu, and Dawn Song. Spatially transformed adversarial examples. *arXiv preprint arXiv:1801.02612*, 2018.

[380] Jin Xiao, Shuhang Gu, and Lei Zhang. Multi-domain learning for accurate and few-shot color constancy. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[381] Xuezhong Xiao and Lizhuang Ma. Color transfer in correlated color space. In *International Conference on Virtual Reality Continuum and Its Applications*, 2006.

[382] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Yuyin Zhou, Lingxi Xie, and Alan Yuille. Adversarial examples for semantic segmentation and object detection. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[383] Yazhou Xing, Zian Qian, and Qifeng Chen. Invertible image signal processing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[384] Ying Xiong, K. Saenko, T. Darrell, and T. Zickler. From pixels to physics: Probabilistic color de-rendering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[385] Ke Xu, Xin Yang, Baocai Yin, and Rynson WH Lau. Learning to restore low-light images via decomposition-and-enhancement. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[386] Kenji Yamamoto, Tomohiro Yendo, Toshiaki Fujii, Masayuki Tanimoto, and David Suter. Color correction for multi-camera system by using correspondences. In *SIGGRAPH Research posters*. 2006.

[387] Joong Nam Yang and Steven K Shevell. Surface color perception under two illuminants: The second illuminant reduces color constancy. *Journal of Vision*, 3(5):4–4, 2003.

[388] Wenhan Yang, Shiqi Wang, Yuming Fang, Yue Wang, and Jiaying Liu. From fidelity to perceptual quality: A semi-supervised approach for low-light image enhancement. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[389] Xin Yang, Ke Xu, Yibing Song, Qiang Zhang, Xiaopeng Wei, and Rynson WH Lau. Image correction via deep reciprocating HDR transformation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[390] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *ACM Computing Surveys (CSUR)*, 38(4):1–45, 2006.

[391] Jonghwa Yim, Jisung Yoo, Won-joon Do, Beomsu Kim, and Jihwan Choe. Filter style transfer between photos. In *European Conference on Computer Vision (ECCV)*, 2020.

[392] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations (ICLR)*, 2015.

[393] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

[394] Runsheng Yu, Wenyu Liu, Yasen Zhang, Zhi Qu, Deli Zhao, and Bo Zhang. DeepExposure: Learning to expose photos with asynchronously reinforced adversarial learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[395] Lu Yuan and Jian Sun. Automatic exposure correction of consumer photographs. In *European Conference on Computer Vision (ECCV)*, 2012.

[396] Roshanak Zakizadeh, Michael S Brown, and Graham D Finlayson. A hybrid strategy for illuminant estimation targeting hard images. In *IEEE International Conference on Computer Vision (ICCV) Workshops*, 2015.

[397] Syed Waqas Zamir, Aditya Arora, Salman Khan, Munawar Hayat, Fahad Shahbaz Khan, Ming-Hsuan Yang, and Ling Shao. CycleISP: Real image restoration via improved data synthesis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[398] Daniele Zavagno. Seeing black and white by a gilchrist. *Perception*, 448:1108–1110, 2007.

[399] Kai Zhang, Wangmeng Zuo, and Lei Zhang. Learning a single convolutional super-resolution network for multiple degradations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[400] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, 2016.

[401] Lvmin Zhang, Yi Ji, Xin Lin, and Chunping Liu. Style transfer for anime sketches with enhanced residual U-Net and auxiliary classifier GAN. In *Asian Conference on Pattern Recognition (ACPR)*, 2017.

[402] Qing Zhang, Yongwei Nie, and Wei-Shi Zheng. Dual illumination estimation for robust exposure correction. In *Computer Graphics Forum*, 2019.

[403] Qing Zhang, Chunxia Xiao, Hanqiu Sun, and Feng Tang. Palette-based image recoloring using color decomposition optimization. *IEEE Transactions on Image Processing*, 26(4):1952–1964, 2017.

[404] Qing Zhang, Ganzhao Yuan, Chunxia Xiao, Lei Zhu, and Wei-Shi Zheng. High-quality exposure correction of underexposed photos. In *ACM International Conference on Multimedia*, 2018.

[405] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European Conference on Computer Vision (ECCV)*, 2016.

[406] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[407] Yonghua Zhang, Jiawan Zhang, and Xiaojie Guo. Kindling the darkness: A practical low-light image enhancer. In *ACM International Conference on Multimedia*, 2019.

[408] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. *arXiv preprint arXiv:1708.04896*, 2017.

[409] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[410] Minfeng Zhu, Pingbo Pan, Wei Chen, and Yi Yang. EEMEFN: Low-light image enhancement via edge-enhanced multi-exposure fusion network. In *AAAI Conference on Artificial Intelligence*, 2020.

[411] Karel Zuiderveld. Contrast limited adaptive histogram equalization. In *Graphics Gems IV*, pages 474–485, 1994.

Part IX

Appendices

A Applications of CIE XYZ Net

In Chapter 11, we demonstrated the usefulness of the CIE XYZ Net on low-light image enhancement. Here, we show additional low-level vision tasks that can benefit from our CIE XYZ reconstruction. Specifically, we discuss how we can reconstruct raw-like images from the reconstructed CIE XYZ images. Further, we show significant gains that can be obtained by this processing framework for additional computer vision tasks.

A.1 Raw-RGB Image Reconstruction

One of the advantages of accurately reconstructing scene-referred images is the ability to map the reconstructed images further into a sensor raw-RGB space. Specifically, we can synthetically generate raw-RGB images in any target sensor space by capturing an image with a color rendition calibration chart placed in the scene. The captured image is saved in both the camera's sensor raw-RGB space and the camera-rendered sRGB color space. As the CIE XYZ space is defined for correctly white-balanced colors, we first correct the white balance of the raw-RGB image using the color rendition chart. We then reconstruct the XYZ image using our XYZ network and compute a 3×3 matrix to map our reconstructed image into the sensor space. We refer to this matrix as the XYZ$\rightarrow$raw matrix.

Note that in order to achieve an accurate mapping from the CIE XYZ space to the sensor raw space, this matrix should be calibrated under different illuminant conditions.

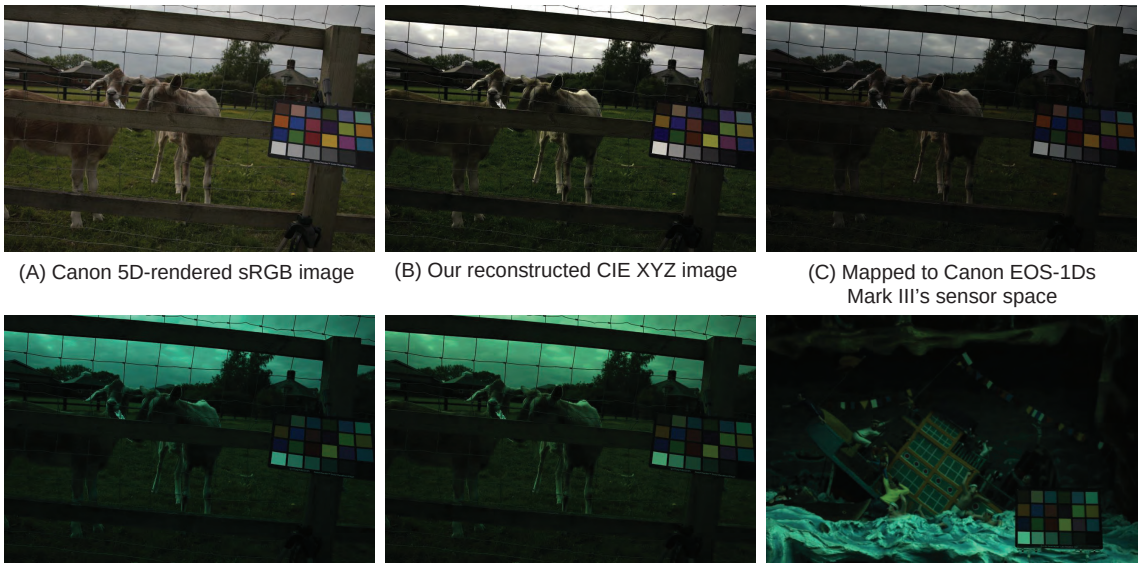


Figure A.1: Sensor raw-RGB image reconstruction. (A) An sRGB image rendered by Canon 5D from Gehler-Shi [132]. (B) Our reconstructed CIE XYZ image. (C) Our reconstructed raw image in the raw-RGB space of the Canon EOS-1Ds Mark III. (D) Two generated raw-RGB images with different illuminant responses in the Canon EOS-1Ds Mark III's sensor space. (E) A real raw-RGB image captured by the Canon EOS-1Ds Mark III taken from the eight-camera NUS dataset [77]. To aid visualization, the shown images are scaled by a factor of two.

For simplicity, we here compute a single global matrix to approximate this mapping. This XYZ→raw matrix is then used to map any arbitrary image into this sensor space by first reconstructing the corresponding XYZ image, followed by mapping it into the sensor space. The assumption here is that as our method achieves superior linearization to the available solutions (see Table 11.1), this calibration process would result in a better sRGB→raw-RGB mapping.

To validate this assumption, we compare between the raw-RGB reconstruction based

on our reconstruction against the raw-RGB reconstruction that is computed based on the standard XYZ mapping [31, 101]. We examine the data augmentation task for illuminant estimation. Scene illuminant estimation is a well-studied problem in computer vision literature. Briefly, we can describe this problem as follows. Given a linear raw-RGB image $\mathbf{I}_{\text{raw}}$ captured by a specific camera sensor, the goal is to determine a 3D vector $\boldsymbol{\ell}$ that represents the illuminant color in the captured scene. Recent work achieves promising results using deep learning to estimate the illuminant vector $\boldsymbol{\ell}$ by training deep models that can be later used in the inference phase to estimate illumination colors of given testing images captured by the same sensor used in the training stage [132].

There is currently a challenge in the available datasets for the illuminant estimation task, which is the limited number of available training images captured by the same sensor—for example, the eight-camera NUS dataset [77], one of the common datasets used for illuminant estimation, has 200 images on average for each camera sensor. In this experiment, we examine our raw-like reconstructed images to serve as a data augmenter to train deep learning models for illuminant estimation. Specifically, we train a simple deep learning model to estimate the scene illuminant of a given raw-RGB image captured by Canon EOS-1Ds Mark III [77].

The model is designed to accept a 150×150 raw-RGB image (similar to prior work that proposed to use thumbnail images for the illuminant estimation task [13, 45]). The model includes a sequence of conv, LReLU, BN, and fc layers. In particular, the model consists of two conv–LReLU–conv–BN–LReLU blocks, followed by a conv–LReLU–FC–LReLU–dropout–FC–LReLU–FC block. All conv layers have 3×3 filters with a different number of output channels and stride steps. The first, second, and third conv layers have 64 output channels, while the fourth and fifth conv layers have 128 and 256 output channels, respectively. The stride steps were set to 2 for the first three conv layers. For

the last two conv layers, we used a stride step of 3. The first two FC layers have 256 output neurons, while the last FC layer has 3 output neurons. We trained each model for 50 epochs to minimize the angular error between the estimated illuminant vector and the ground truth illuminant. The training process was performed with a learning rate of 10^{-4} and mini-batch of 32 using the Adam optimizer [204] with a decay rate of gradient moving average 0.9 and a decay rate of squared gradient moving average 0.999.

There are only 256 original raw-RGB images captured by Canon EOS-1Ds Mark III in the NUS dataset [77]. For each image, there is a ground-truth scene illuminant vector extracted from the color rendition chart. During training and testing processes, the color chart is masked out in each image to avoid any bias. To augment the data, we first computed the 3×3 XYZ→raw calibration matrix as described earlier for our XYZ reconstruction and the standard XYZ mapping. This reconstruction process was performed using a single image captured by the Canon EOS-1Ds Mark III camera with a color rendition chart.

Afterwards, we used 3,752 white-balanced camera-rendered sRGB images captured by ten different camera models other than our Canon EOS-1Ds Mark III. These images were taken from the Rendered WB dataset [20]. Each sRGB image is converted to the CIE XYZ space using our method and the standard XYZ mapping, followed by mapping each reconstructed image to the Canon EOS-1Ds Mark III sensor space using the calibration matrix computed for each XYZ reconstruction method, respectively.

As the calibration matrices map from the reconstructed XYZ space to the white-balanced sensor raw-RGB space, we can apply illuminant color casts, randomly selected from the ground-truth illuminant vectors provided in the Canon EOS-1Ds Mark III's set, to synthetically generate additional training data to train the deep model. Figure A.1 shows an example. This process is inspired by previous work in [123, 248], which randomly selected illuminant vectors from the ground-truth set and applied chromatic adaptation to

augment the training set. These methods, however, use the same images (256 images in the case of the Canon EOS-1Ds Mark III's set) without introducing new image content to the trained model.

We randomly selected 50 testing images from the original 256 images provided in the NUS dataset for the Canon EOS-1Ds Mark III camera. We fixed this testing set over all experiments and excluded these images from any training processes. Table A.1 shows the angular error of the trained model using the following training sets: (i) real training data, (ii) reconstructed raw-like images using the standard XYZ mapping, (iii) real training data and reconstructed raw-like images using the standard XYZ mapping, (iv) reconstructed raw-like images using our XYZ reconstruction, and (v) real training data and reconstructed raw-like images using our XYZ reconstruction. As can be seen, the best results were obtained by using our raw-like reconstruction and real training data. Notice that training only on our raw-like reconstruction gives better results compared with the results obtained by training on real data or reconstructed raw-like images using the standard XYZ mapping. Additional training details are given in the supplementary materials.

A.1.1 Additional Applications

A hazy image is expressed using a linear model as $\mathbf{I}(\mathbf{x}) = \mathbf{J}(\mathbf{x})t(\mathbf{x}) + \mathbf{A}(1 - t(\mathbf{x}))$ [157], where $\mathbf{I}$ is the observed intensity, $\mathbf{J}$ is the scene radiance, $\mathbf{A}$ is the global atmospheric light, and t is the medium transmission describing the portion of the light that is not scattered and reaches the camera. Just as with motion deblurring, this linear relationship is broken by the camera's photo-finishing stages. Therefore, it is desirable to perform dehazing on linearized images. In Fig. A.2, we show the result of dehazing an sRGB image versus dehazing our linear CIE XYZ image and then re-rendering to sRGB. The improvement in visual quality can be clearly observed from the zoomed-in regions.

Table A.1: Angular error of illuminant estimating using the image set captured by the Canon EOS-1Ds Mark III in the NUS dataset [77]. We compare the results obtained by training a deep neural network on real raw-RGB training images, reconstructed (rec.) raw-RGB training images based on the standard XYZ reconstruction, and our CIE XYZ reconstruction. The best results are shown in bold.

Training data	Mean	Median	Best 25%	Worst 25%
Real	4.15	3.89	1.13	7.85
Rec. (standard)	3.37	3.03	1.05	6.68
Real and rec. (standard)	2.72	2.60	0.72	4.99
Rec. (ours)	3.00	2.61	0.83	5.37
Real and rec. (ours)	**2.41**	**2.03**	**0.65**	**4.66**

Another application of our CIE XYZ reconstructed images is chromatic adaptation. When we work in our reconstructed space (i.e., XYZ), we have a sound interpretation of post-capture white-balance editing using standard white points (e.g., D65, D50) and standard chromatic adaptation transforms (e.g., Bradford CAT [219], Sharp CAT [113]), which are originally designed to work in the camera CIE XYZ space. Fig. A.3 shows examples of our enhanced rendering with applying chromatic adaptation [113] in our reconstructed XYZ space.

Additional potential applications are shown in Fig. A.4. In the first row of Fig. A.4, we show super-resolution results obtained directly by working in the sRGB space and in our reconstructed CIE XYZ space followed by applying our re-rendering process. The last row of Fig. A.4 shows an arguably better color transfer result by applying the color transfer process in our reconstructed space.

Figure A.2: Dehazing is one of the potential applications that can benefit from our unprocessing method. (A) Input image taken from Flickr (by Mike Rivera, CC BY-NC-SA 2.0). (B) Dehazing applied in sRGB space. (C) Dehazing applied to our CIE XYZ image. (D) Our final result in sRGB space. In this example, we used the dehazing method from [157].

Figure A.3: (A) Input sRGB rendered image. (B) Our re-rendered images after enhancement. In this example, we applied chromatic adaptation to three different reference white points. Input image is taken from the under-exposure set [369] of the MIT-Adobe FiveK dataset [62].

Lastly, our rendering network can be used as an alternative way to produce aesthetic photographs from raw-RGB DNG files, as shown in Fig. A.5. In this example, we first used the DNG metadata to map the raw-RGB values into the CIE XYZ space. Then, we used our rendering network and a local Laplacian filter to generate the shown output images.

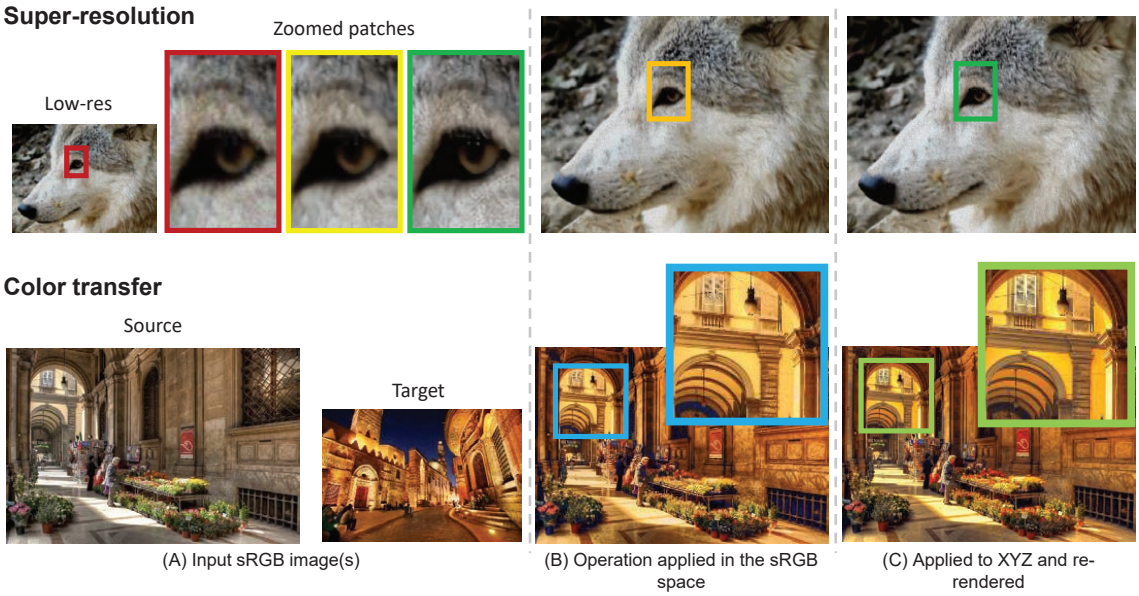


Figure A.4: Additional potential applications of our method. (A) The input sRGB image. (B) Super-resolution and color transfer applied in the sRGB space. (C) Super-resolution and color transfer applied in our reconstructed CIE XYZ space followed by re-rendering. In this example, we used the deep learning super-resolution model proposed in [399] and the color transfer method in [304]. The input image in the first row is taken from the DIV2K dataset [25, 358], while the second input image is taken from Flickr–CC BY-NC 2.0 (by Chris Ford and Giuseppe Moscato, respectively).

Figure A.5: Our rendering network generalizes well for unseen CIE XYZ input images and produces pleasing results that are close to Adobe Lightroom's quality. (A) Input smartphone camera CIE XYZ image. (B) Standard rendering [31,101]. (C) Our rendering. (D) Our rendering after enhancing the local layer using the local Laplacian filter [297]. (E) Adobe Lightroom rendering. To aid visualization, CIE XYZ images are scaled by a factor of two. Input image is taken from the HDR+ burst photography dataset [155].

B Data Augmentation for Color Constancy

In this appendix, we describe in detail the data augmentation procedure described in Chapter 5. We begin with the steps used to map a color temperature to the corresponding CIE XYZ value. We then elaborate on the process required to map from camera sensor raw to the CIE XYZ color space. Afterwards, we describe the details of the scene retrieval process mentioned in Chapter 5. Finally, we discuss experiments performed to evaluate our data augmentation and compare it with other color constancy augmentation techniques used in the literature.

B.1 From Color Temperature to CIE XYZ

According to Planck's radiation law [378], the SPD of a blackbody radiator at a given wavelength range $[\lambda, \partial\lambda]$ can be computed using the color temperature q as follows:

$$S_\lambda d_\lambda = \frac{f_1 \lambda^{-5}}{\exp\left(f_2/\lambda q\right) - 1} \partial\lambda, \tag{B.1}$$

where, $f_1 = 3.741832^1 0^{-16}\ Wm^2$ is the first radiation constant, $f_2 = 1.4388^{10-2} mK$ is the second radiation constant, and q is the blackbody temperature, in Kelvin. [227,368]. Once the SPD is computed, the corresponding CIE tristimulus values can be approximated in the following discretized form:

$$X = \Delta\lambda \sum_{\lambda=380}^{\lambda=780} x_\lambda S_\lambda, \tag{B.2}$$

where the value of x_λ is the standard CIE color match value [82]. The values of Y and Z are computed similarly. The corresponding chromaticity coordinates of the computed XYZ tristimulus are finally computed as follows:

$$\begin{aligned} x &= X/(X+Y+Z), \\ y &= Y/(X+Y+Z), \\ z &= Z/(X+Y+Z). \end{aligned} \tag{B.3}$$

B.2 From Raw to CIE XYZ

Most DSLR cameras provide two pre-calibrated matrices, C_1 and C_2, to map from the camera sensor space to the CIE 1931 XYZ 2-degree standard observer color space. These pre-calibrated CST matrices are usually provided as a low color temperature (e.g., Standard-A) and a higher correlated color temperature (e.g., D65) [4].

Given an illuminant vector $\boldsymbol{\ell}$, estimated by an illuminant estimation algorithm, the CIE XYZ mapping matrix associated with $\boldsymbol{\ell}$ is computed as follows [64]:

$$C_{T_{\boldsymbol{\ell}}} = C_2 + (1-\alpha)\, C_1, \tag{B.4}$$

$$\alpha = (1/q_{\boldsymbol{\ell}} - 1/q_1)/(1/q_2 - 1/q_1), \tag{B.5}$$

where q_1 and q_2 are the correlated color temperature associated to the pre-calibrated matrices C_1 and C_2, and $q_{\boldsymbol{\ell}}$ is the color temperature of the illuminant vector $\boldsymbol{\ell}$. Here, $q_{\boldsymbol{\ell}}$ is unknown, and unlike the standard mapping from color temperature to the CIE XYZ space (Sec. B.1), there is no standard conversion from a camera sensor raw space to the corresponding color temperature. Thus, the conversion from the sensor raw space to the CIE XYZ space is a chicken-and-egg problem—computing the correlated color temperature $q_{\boldsymbol{\ell}}$ is necessarily to get the CST matrix $C_{q_{\boldsymbol{\ell}}}$, while knowing the mapping from a camera

sensor raw to the CIE XYZ space inherently requires knowledge of the correlated color temperature of a given raw illuminant.

This problem can be solved by a trial-and-error strategy as follows. We iterate over the color temperature range of 2500K to 7500K. For each color temperature q_i , we first compute the corresponding CST matrix C_{q_i} using Eqs. B.4 and B.5. Then, we convert q_i to the corresponding xyz chromaticity triplet using Eqs. B.1–B.3.

Afterwards, we map the xyz chromaticity triplet to the sensor raw space using the following equation:

$$\boldsymbol{\ell}_{\mathtt{raw}(q_i)} = C_{q_i}^{-1} \lambda_{\mathtt{xyz}(q_i)}. \tag{B.6}$$

We repeated this process for all color temperatures and selected the color temperature/CST matrix that achieves the minimum angular error between $\boldsymbol{\ell}$ and the reconstructed illuminant color in the sensor raw space.

The accuracy of our conversion depends on the pre-calibrated matrices provided by the manufacturer of the DSLR cameras. Other factors that may affect the accuracy of the mapping includes the precision of the standard mapping from color temperature to XYZ space defined by [82], and the discretization process in Eq. B.2.

B.3 Raw-to-raw mapping

Here, we describe the details of the mapping mentioned in Chapter 5. Let A=$\{\mathbf{a}_1, \mathbf{a}_2, ...\}$ represent the "source" set of demosaiced raw images taken by different camera models with the associated capture metadata. Let $T = \{\mathbf{t}_1, \mathbf{t}_2, ...\}$ represent our "target" set of metadata of captured scenes by the target camera model. Here, the capture metadata includes exposure time, aperture size, ISO gain value, and the global scene illuminant color in the camera sensor space. We also assume that we have access to the pre-calibration CST

matrices for each camera model in the sets A and T (available in most DNG files of DSLR images [4]).

Our goal here is to map all raw images in A, taken by different camera models, to the target camera sensor space in T. To that end, we map each image in A to the device-independent CIE XYZ color space [82]. This mapping is performed as follows. We first compute the correlated color temperature, $q^{(i)}$, of the scene illuminant color vector, $\boldsymbol{\ell}^{(i)}_{\texttt{raw}(A)}$, of each raw image, $I^{(i)}_{\texttt{raw}(A)}$, in the set A (see Sec. B.2). Then, we linearly interpolate between the pre-calibrated CST matrices provided with each raw image to compute the final CST mapping matrix, $C_{q^{(i)}}$, [64]. Afterwards, we map each image, $I^{(i)}_{\texttt{raw}(A)}$, in the set A to the CIE XYZ space. Note that here we represent each image I as matrices of the color triplets (i.e., $I = \{\mathbf{c}^{(k)}\}$), where k is the total number of pixels in the image I. We map each raw image to the CIE XYZ space as follows:

$$I^{(i)}_{\texttt{xyz}(A)} = C_{q^{(i)}} D_{\boldsymbol{\ell}^{(i)}} I^{(i)}_{\texttt{raw}(A)}, \tag{B.7}$$

where $D_{\boldsymbol{\ell}^{(i)}}$ is the white-balance diagonal correction matrix constructed based on the illuminant vector $\boldsymbol{\ell}^{(i)}_{\texttt{raw}(A)}$.

Similarly, we compute the inverse mapping from the CIE XYZ space back to the target camera sensor space based on the illuminant vectors and pre-calibration matrices provided in the target set T. The mapping from the source sensor space to the target one in T can be performed as follows:

$$I^{(i)}_{\texttt{raw}(T)} = D^{-1}_{\jmath^{(i)}} M^{-1}_{q^{(i)}} I^{(i)}_{\texttt{xyz}(A)}, \tag{B.8}$$

where $\jmath^{(i)}_{\texttt{raw}(T)}$ is the corresponding illuminant color to the correlated color temperature, $q^{(i)}$, in the target sensor space (i.e., the ground-truth illuminant for image $I^{(i)}_{\texttt{raw}(T)}$ in the illuminant estimation task), and $M^{-1}_{q^{(i)}}$ is the CST matrix that maps from the target sensor space to the CIE XYZ space.

The described steps so far assume that the spectral sensitivities of all sensors in A and T satisfy the Luther condition[1] [278]. Prior studies, however, showed that this assumption is not always satisfied, and this can affect the accuracy of the pre-calibration matrices [182, 193]. According to this, we rely on Eqs. B.7 and B.8 only to map the original colors of captured objects in the scene (i.e., white-balanced colors) to the target camera model. For the values of the global color cast, $J^{(i)}_{\texttt{raw}(T)}$, we do not rely on $M^{-1}_{q^{(i)}}$ to map $\boldsymbol{\ell}^{(i)}_{\texttt{raw}(A)}$ to the target sensor space of T. Instead, we follow a K-nearest neighbor strategy to get samples from the target sensor's illuminant color space.

B.4 Scene Sampling

As described in Chapter 5, we retrieve metadata of similar scenes in the target set T for illuminant color sampling. This sampling process should consider the source scene capture conditions to sample suitable illuminant colors from the target camera model space—i.e., having indoor illuminant colors as ground-truth for outdoor scenes may affect the training process. To this end, we introduce a retrieval feature $v^{(i)}_A$ to represent the capture settings of the image $I^{(i)}_{\texttt{raw}(A)}$. This feature includes the correlated color temperature and auxiliary capture settings. These additional capture settings are used to retrieve scenes captured with similar settings of $I^{(i)}_{\texttt{raw}(A)}$.

Our feature vector is defined as follows:

$$v^{(i)}_A = [q^{(i)}_{\texttt{norm}}, h^{(i)}_{\texttt{norm}}, p^{(i)}_{\texttt{norm}}, e^{(i)}_{\texttt{norm}}], \tag{B.9}$$

where $q^{(i)}_{\texttt{norm}}$, $h^{(i)}_{\texttt{norm}}$, $p^{(i)}_{\texttt{norm}}$, and $e^{(i)}_{\texttt{norm}}$ are the normalized color temperature, gain value, aperture size, and scaled exposure time, respectively. The gain value and the scaled exposure

[1]The Luther condition is satisfied when camera spectral sensitivities are linearly related to the CIE XYZ space.

time are computed as follows:

$$h^{(i)} = \texttt{BLN}^{(i)}\texttt{ISO}^{(i)}\,, \tag{B.10}$$

$$e^{(i)} = \sqrt{2^{\texttt{BLE}^{(i)}}}l^{(i)}\,, \tag{B.11}$$

where BLE, BLN, ISO, and l are the baseline exposure, baseline noise, digital gain value, and exposure time (in seconds), respectively.

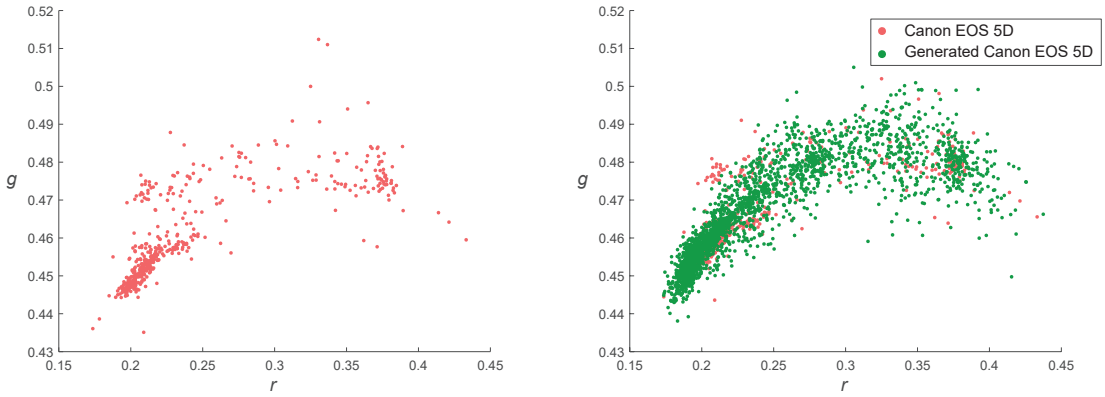
Illuminant Color Sampling A naive sampling from the associated illuminant colors in T does not introduce new illuminant colors over the Planckian locus of the target sensor. For this reason, we first fit a cubic polynomial to the rg chromaticity of illuminant colors in the target sensor T. Then, we compute a new r chromaticity value for each query vector as follows:

$$r_v = \sum_{j \in K} w_j r_j + x\,, \tag{B.12}$$

where $w_j = \exp(1 - d_j)/\sum_k^K \exp(1 - d_k)$ is a weighting factor, $x = \lambda_r \mathcal{N}(0, \sigma_r)$ is a small random shift, λ_r is a scalar factor to control the amount of divergence from the ideal Planckian curve, σ_r is the standard deviation of the r chromaticity values in the retrieved K metadata of the target camera model, T_K, and d_j is the normalized L2 distance between $v_{S(i)}$ and the corresponding $j^{\texttt{th}}$ feature vector in T_K, respectively. In our experiments, we retrieved the nearest 15 sample in from target camera model to our retrieval feature $v_A^{(i)}$ (i.e., K includes 15 samples from the target camera model). The CST matrix M (Eq. B.8) is constructed by linearly interpolating between the corresponding CST matrices associated with each sample in T_K using w_j. After computing r_v, the corresponding g chromaticity value is computed as:

$$g_v = [r_v, r_v^2, r_v^3][\xi_1, \xi_2, \xi_3]^\top + y\,, \tag{B.13}$$

where $[\xi_1, \xi_2, \xi_3]$ are the cubic polynomial coefficients, y is a random shift, and σ_g is the standard deviation of the g chromaticity values in T_K. In our experiments, we set $\lambda_r = 0.7$



Real illuminant *rg* color distribution taken by Canon EOS 5D

Generated illuminants follow the distribution of the real sensor's illuminant colors

Figure B.1: Synthetic illuminant samples of Canon EOS 5D camera model in the Gehler-Shi dataset [132]. The shown generated illuminant colors are then applied to sensor-mapped raw images, originally were taken by different camera models, for augmentation purpose (Chapter 5).

and $\lambda_g = 1$. The final illuminant color $\jmath^{(i)}_{\texttt{raw}(T)}$ can be represented as follows:

$$\jmath^{(i)}_{\texttt{raw}(T)} = [r_v, g_v, 1 - r_v - g_v]^\top . \tag{B.14}$$

To avoid any bias towards the dominant color temperature in the source set, A, we first divide the color temperature range of the source set A into different groups with a step of 250K. Then, we uniformly sample examples from each group to avoid any bias towards specific type of illuminants. Figure B.1 shows examples of the sampling process. As shown, the sampled illuminant chromaticity values follow the original distribution over the Planckian curve, while introducing new illuminant colors of the target sensors that were not included in the original set. Finally, we apply random cropping to introduce more diversity in the generated images. Figure B.2 shows examples of synthetic raw-like images of different target camera models.

Figure B.2: Example of camera augmentation used to train our network. The shown left raw image is captured by Nikon D5200 camera [77]. The next three images are the results of our mapping to different camera models.

B.5 Evaluation

In prior work, several approaches for training data augmentation for illuminant estimation have been attempted [123, 248]. These approaches first white-balance the training raw images using the associated ground-truth illuminant colors associated with each image. Afterwards, illuminant colors are sampled from the "ground-truth" illuminant colors over the entire training set to be applied to the white-balanced raw images. These sampled illuminant colors can be taken randomly from the ground-truth illuminant colors [123] or after clustering the ground-truth illuminant colors [248]. These methods, however, are limited to using the same set of scenes as is present in the training dataset. Another approach for data augmentation has been proposed in Appendix A by mapping sRGB white-balanced images to a learned normalization space that is is learned based on the CIE XYZ space. Afterwards, a pre-computed global transformation matrix is used to

Table B.1: A comparison of different augmentation methods for illuminant estimation. All results were obtained by using training images captured by the Canon EOS 5D camera model [132] as the source and target sets for augmentation. Lowest errors are highlighted in yellow.

Training set	**Mean**	**Med.**	**B. 25%**	**W. 25%**
Original set	1.81	1.12	0.35	4.43
Augmented (clustering & sampling) [248]	1.68	0.97	0.25	4.31
Augmented (sampling) [123]	1.79	1.09	0.33	4.34
Augmented (ours)	1.55	0.98	0.28	3.68

map the images from this normalization space to the target white-balanced raw space. In contrast, the augmentation method described in Chapter 5 uses an accurate mapping from the camera sensor raw space to the CIE XYZ using the pre-calibration matrices provided by camera manufacturers.

In the following set of experiments, we use the baseline model FFCC [45] to study the potential improvement of our chosen data augmentation strategy and alternative augmentation techniques proposed in [123, 248]. Additionally, we include the results of our augmentation discussed in Appendix A. We use the Canon EOS 5D images from in the Gehler-Shi dataset [132] for comparisons. For our test set, we randomly select 30% of the total number of images in the Canon EOS 5D set. The remaining 70% of images are used for training. We refer to this set as "real training set", which includes 336 raw images.

Note that, except for the augmentation proposed in Appendix A, none of these methods apply a sensor-to-sensor mapping, as they use the raw images of the "real training set" as the source and target set for augmentation. For this reason and for a fair comparison,

Table B.2: A comparison of techniques for generating new sensor-mapped raw-like images that were originally captured by different sensors than the training camera model. The term 'synthetic' refers to training FFCC [45] without including any of the original training examples, while the term 'augmented' refers to training on synthetic and real images. The best results are bold-faced. Lowest errors of synthesized and augmented sets are highlighted in red and yellow, respectively.

Training set	**Mean**	**Med.**	**B. 25%**	**W. 25%**
Synthetic (Appendix A)	4.17	3.06	0.78	9.39
Augmentation (Appendix A)	2.64	1.95	0.45	5.97
Synthetic (ours)	2.44	1.89	0.42	5.40
Augmented (ours)	**1.75**	**1.28**	**0.35**	**4.15**

we provide the results of two different set of experiments. In the first experiment, we use the CIE XYZ images taken by the Canon EOS 5D sensor as our source set A, while in the second experiment, we use a different set of four sensors rather than the Canon EOS 5D sensor. The former is comparable to the augmentation methods used in [123, 248] (see Table B.1), while the latter is comparable to the augmentation approach proposed in Appendix A, which performs "raw mapping" in order to introduce new scene content in the training data (see Table B.2). The shown results obtained by generating 500 synthetic images by each augmentation method, including our augmentation approach. As shown in Tables B.1 and B.2, our augmentation approach achieves the best improvement of the FFCC results.

In order to study the effect of the CIE XYZ mapping used by our augmentation approach, we trained FFCC [45] on a set of 500 synthetic raw images of the target camera

Table B.3: Results of FFCC [45] trained on synthetic raw-like images after they are mapped to the target camera model. In this experiment, the raw images are mapped from the Canon EOS-1Ds Mark III camera sensor (taken from the NUS dataset [77]) to the target Canon EOS 5D camera in the Gehler-Shi dataset [132]. The shown results were obtained with and without the intermediate CIE XYZ mapping step to generate the synthetic training set. Lowest errors are highlighted in yellow.

Synthetic training set	**Mean**	**Med.**	**B. 25%**	**W. 25%**
w/o CIE XYZ	3.30	2.55	0.60	7.21
w/ CIE XYZ	3.04	2.36	0.56	6.58

model—namely, the Canon EOS 5D camera model in the Gehler-Shi dataset [132]. These synthetic raw images were originally captured by the Canon EOS 1Ds Mark III camera sensor (in the NUS dataset [77]), then these images are mapped to the target sensor using our augmentation approach. Table B.3 shows the results of FFCC trained on synthetic raw images with and without the intermediate CIE XYZ mapping step (Eqs. B.7 and B.8). As shown, using the CIE XYZ mapping achieves better results, which are further improved by increasing the scene diversity of the source set by including additional scenes from other datasets, as shown in Table B.2.

For a further evaluation, we use our approach to map images from the Canon EOS 5D camera's set (the same set that was used to train the FFCC model) to different target camera models. Then, we trained and tested a FFCC model on these mapped images. This experiment was performed to gauge the ability of our data augmentation approach to have similar negative effects on camera-specific methods that were trained on a different camera model. To that end, we randomly selected 150 images from the Canon EOS 5D

Table B.4: Results of FFCC [45] trained on the Canon EOS 5D camera [132] and tested on images taken by different camera models from the NUS dataset [77] and the Cube+ challenge set [41]. The synthetic sets refer to testing images generated by our data augmentation approach, where these images were mapped from the Canon EOS 5D set (used for training) to the target camera models.

Testing sensor	**Real camera images**			**Synthetic camera images**		
	Mean	**Med.**	**Max**	**Mean**	**Med.**	**Max**
Canon EOS 1D [132]	3.88	2.66	16.32	4.68	3.80	22.83
Fujifilm XM1 [77]	4.22	3.05	47.87	2.91	2.06	38.93
Nikon D5200 [77]	4.45	3.45	36.762	3.36	2.10	41.23
Olympus EPL6 [77]	4.35	3.56	19.89	3.28	2.27	38.81
Panasonic GX1 [77]	2.83	2.03	16.58	3.24	2.29	17.07
Samsung NX2000 [77]	4.41	3.73	17.69	3.44	2.64	18.79
Sony A57 [77]	3.84	3.02	19.38	3.04	1.34	39.67
Canon EOS 550D [41]	3.83	2.49	46.55	3.14	1.98	36.30

sensor set, which was used to train the FFCC model, as our source image set A. Then, we mapped these images to different target camera models using our approach. That means that the training and our synthetic testing set share the same scene content. We report the results in Table B.4. We also report the testing results on real image sets captured by the same target camera models. As shown in Table B.4, both real and synthetic sets negatively affect the accuracy of the FFCC model (see Table B.1 for results of the FFCC on a testing set taken by the same training sensor).

C White-Balance Augmentation for Image Relighting

Image relighting has multiple applications both in research and in practice, and is recently witnessing an increased interest. A single-image relighting method would allow aesthetic enhancement applications, such as photo montage of images taken under different illuminations, and illumination retouching without human expert work. Very importantly, in computer vision research image relighting can be leveraged for data augmentation, enabling the trained methods to be robust to changes in light source position or color temperature. It could also serve for domain adaptation, by normalizing input images to a unique set of illumination settings that the down-stream computer vision method was trained on.

We employed our white-balance techniques for correction (Chapter 6) and data augmentation (Chapter 7) to develop our image relighting framework, Norm-Relighting-U-Net and illuminant setting estimation network[1]. Our frameworks achieved the Running-Up Award over all tasks in the AIM 2020 challenge for Scene relighting [162]. The source code of our method is available in GitHub: https://github.com/mahmoudnafifi/image_relighting.

[1]This work was published in [162]: Majed El Helou, Ruofan Zhou, Sabine Süsstrunk, Radu Timofte, Mahmoud Afifi, Michael S Brown, et al. AIM 2020: Scene relighting and illumination estimation challenge. In European Conference on Computer Vision (ECCV) Workshops, 2020.

C.1 Challenge Tasks

The AIM2020 challenge includes three tasks, which are: (i) one-to-one relighting, (ii) one-to-any relighting, and (iii) Illumination settings estimation. In this section, we provide details of each task, the dataset and evaluation protocol used in this challenge.

One-to-One Relighting The relighting task is pre-determined and fixed for all validation and test samples. In other words, the objective is to manipulate an input image from one pre-defined set of illumination settings (namely, North, 6500K) to another pre-defined set (East, 4500K). The images are in 1024×1024resolution, both input and output, and nothing other than the input image is provided.

Any-to-Any Relighting This track is a generalization of the first track. The objective is to relight an input image (both color temperature and light source position manipulation) from any arbitrary illumination settings to any arbitrary illumination settings. The latter settings are dictated by a second input guide image, as in style transfer applications. The participants were allowed to make use of their solutions to the first two tracks to develop a solution for this track. The images are in 512×512 resolution to ease computations, as this track is very challenging.

Illumination Setting Estimation The goal of this track is to estimate, from a single input image, the illumination settings that were used in rendering it. Given the input image, the output should estimate the color temperature of the illuminant as well as the orientation, i.e. the position of the light source. The input images are also 1024×1024 and no other input is given than the 2D image.

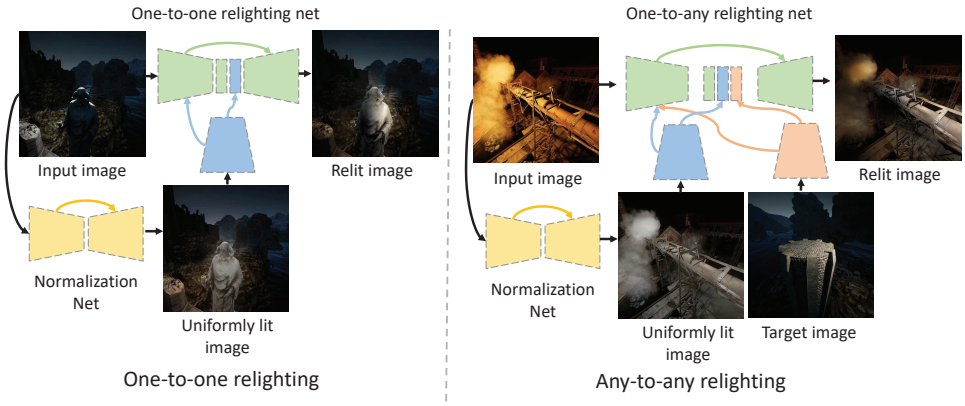


Figure C.1: Overview of our Norm-Relighting-U-Net used for one-to-one and any-to-any relighting tasks.

Data We used the VIDIT dataset [105, 162], which is a well-controlled setup to provide full-reference evaluation through a set of virtual scenes. The VIDIT dataset contains 300 virtual scenes used for training, where every scene is captured 40 times in total: from 8 equally-spaced azimuthal angles, each lit with 5 different illuminants. Every image is 1024×1024, but the images are downsampled by a factor of 2, with bicubic interpolation over 4×4 windows.

Evaluation Protocol To evaluate the image relighting results, the PSNR and SSIM [373] metrics are used. For the final ranking, the Mean Perceptual Score (MPS) is used. The MPS is defined as the average of the normalized SSIM and LPIPS [406] scores, themselves averaged across the entire test set. This MPS can be described by the following equation

$$0.5(S + (1 - L)), \tag{C.1}$$

where S is the SSIM score, and L is the LPIPS score. While the evaluation of the illuminant setting estimation task is based on the accuracy of predictions following this formula for

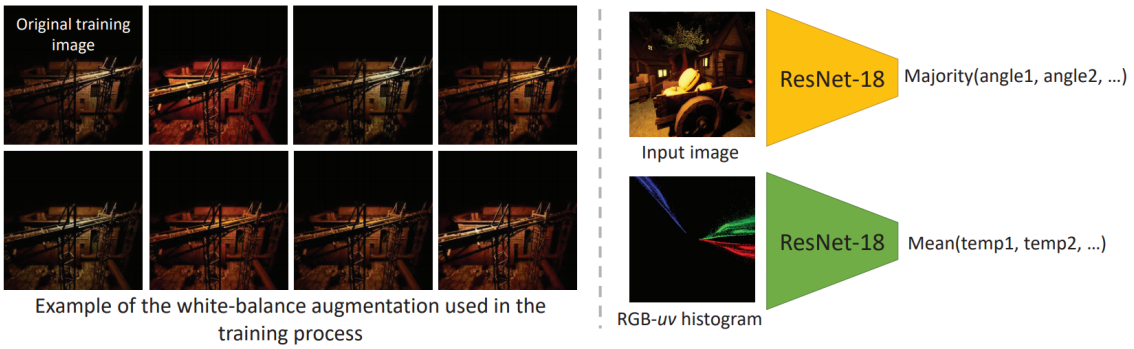


Figure C.2: Overview of our Illuminant-ResNet, with the white-balance augmentation.

the loss:

$$\sqrt{\sum_{i=0}^{N-1}\left(\frac{|\hat{\phi}_i - \phi_i| mod 180}{180}\right)^2 + (\hat{T}_i - T)^2}, \tag{C.2}$$

where $\hat{\phi}_i$ is the predicted angle (0-360) for test sample i, ϕ_i is the ground-truth value for that sample, $\hat{T}_i$ is the temperature prediction for test sample i, and T_i is the ground-truth value for that sample. The temperature T_i takes values equal to [0, 0.25, 0.5, 0.75, 1], which correspond to the color temperature values [2500K, 3500K, 4500K, 5500K, 6500K].

C.2 Norm-Relighting-U-Net

For image relighting tasks, we adopt a U-Net architecture [323] as the main backbone of our framework. Our framework consists of two networks: (i) the normalization net, which is responsible for producing uniformly lit white-balanced images, and (ii) the relighting network, which performs the image relighting task. We apply an instance normalization [360] after each stage in the encoder of the normalization network, while we used batch normalization for the encoder of the relighting network. The relighting network is fed by the input image and the latent representations of the guide image and the uniformly lit image produced by our normalization network. We used our white-balance augmenter,

described in Chapter 7, to augment the training data used for the normalization network. To produce the ground-truth of the normalization network, we use the training data, which provide us with a set of images taken from each scene under different lighting directions. Specifically, we first white-balance all images using our KNN WB method described in Chapter 6. Then, we compute the average image over all images of each scene set. We trained two models: the first model is trained on 256×256 random patches; the second model is trained on 256×256 resized images. The final result is generated by taking the mean of the two generated relit images. The details of our framework for each task are shown in Fig. C.1.

C.3 Illuminant-ResNet

For the illuminant setting task, we treat the task as two independent classification tasks: (i) illuminant temperature classification and (ii) illuminant angle classification. We adopt the ResNet-18 model [159] trained on ImageNet [91]. The last fully connected layer is replaced with a new layer with n neurons, where n is the number of output classes for each task. The Adam optimizer [204] is used with cross entropy loss. For angle classification, we applied our white-balance augmenter described in Chapter 7 to augment the training data. For temperature classification, we use image histogram features instead of the 2D input image. Specifically, we feed the network with 2D RGB-uv projected histogram features (described in Chapter 6), instead of the original training images. This histogram-based training, rather than image-based, improves the model's generalization. Figure C.2 shows an overview of the team's solution, including the white-balance augmentation process.

Table C.1: Results of Image Relighting Challenge for the one-to-one relighting task. The MPS, used to determine the final ranking.

Team	**MPS** ↑	SSIM ↑	LPIPS ↓	PSNR ↑
CET_CVLab [162]	0.6451 (1)	0.6362 (1)	0.3460 (3)	16.8927 (6)
lyl [162]	0.6436 (2)	0.6301 (3)	0.3430 (2)	16.6801 (8)
YorkU (ours)	0.6216 (3)	0.6091 (4)	0.3659 (5)	16.8196 (7)
IPCV_IITM [162]	0.5897 (4)	0.5298 (7)	0.3505 (4)	17.0594 (3)
DeepRelight [162]	0.5892 (5)	0.5928 (6)	0.4144 (7)	17.4252 (1)
Withdrawn [162]	0.5603 (6)	0.5236 (8)	0.4029 (6)	16.5136 (9)
Hertz [162]	0.5339 (7)	0.5666 (6)	0.4989 (8)	16.9234 (4)
Image Lab [162]	0.3746 (8)	0.3769 (9)	0.6278 (9)	16.8949 (5)

C.4 Results

There were 20 teams participated in the AIM challenge for relighting and illuminant setting estimation [162]. Tables C.1–C.3 shows the results for each task in the challenge. As can be seen, our method achieves the second/third rank over all tasks.

Table C.2: Results of Image Relighting Challenge for the any-to-any relighting task. The MPS, used to determine the final ranking.

Team	**MPS** ↑	SSIM ↑	LPIPS ↓	PSNR ↑
NPU-CVPG [162]	0.6484 (1)	0.6353 (1)	0.3386 (3)	18.5436 (2)
YorkU (ours)	0.6428 (2)	0.6195 (2)	0.3338 (2)	18.2384 (4)
IPCV_IITM [162]	0.6424 (3)	0.6042 (3)	0.3194 (1)	19.3559 (1)
lyl [162]	0.6213 (4)	0.5881 (4)	0.3455 (4)	17.6314 (5)
AiRiA_CG [162]	0.5258 (5)	0.4451 (5)	0.3936 (5)	18.3493 (3)
RGETH [162]	0.3465 (6)	0.4123 (6)	0.7192 (6)	10.4483 (6)

Table C.3: Results of Image Relighting Challenge for the Illumination setting estimation task. The loss is computed based on the angle and color temperature predictions, as described in Eq. C.2.

Team	**Loss** ↓	AngLoss ↓	TempLoss ↓
AiRiA_CG [162]	0.0875 (1)	0.0722 (3)	0.0153 (1)
YorkU (ours)	0.0887 (2)	0.0639 (2)	0.0248 (2)
Image Lab [162]	0.0984 (3)	0.0513 (1)	0.0471 (5)
debut_kele [162]	0.1431 (4)	0.1125 (4)	0.0306 (3)
RGETH [162]	0.1708 (5)	0.1347 (5)	0.0361 (4)

D Additional Details of HistoGANs

In this appendix, we provide additional details and results of our work described in Chapter 14. We first discuss additional details of our networks in Sec. D.1. Then, we explain the training details in Sec. D.2. Afterwards, Sec. D.3 presents ablation experiments carried out to validate our choice of hyperparameters and loss terms. In Sec. D.4, we present our experiments performed to train a "universal" recoloring model to recolor images taken from arbitrary domains. Sec. D.5 discusses failure cases of our method. Such failure cases can often be mitigated by applying simple post-processing. The post-processing details in Sec. D.6. Sec. D.6 also discuss post-processing to deal with high-resolution images.

D.1 Details of Our Networks

Our discriminator network, used in all of our experiments, consists of a sequence of $\log_2(N) - 1$ residual blocks, where N is the image width/height, and the last layer is an fully connected (fc) layer that produces a scalar feature. The first block accepts a three-channel input image and produce m output channels. Then, each block i produces $2m_{i-1}$ output channels (i.e., duplicate the number of output channels of the previous block). The details of the residual blocks used to build our discriminator network are shown in Fig. D.1.

Figure D.2 provides the details of our encoder, decoder and GAN blocks used in our

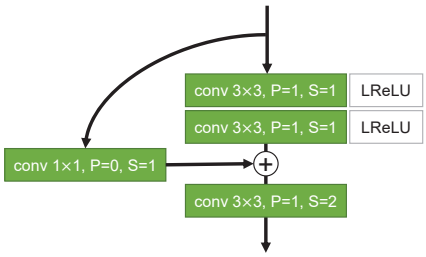


Figure D.1: Details of the residual discriminator block used to reconstruct our discriminator network. The term P and S refer to the padding and stride used in each layer.

ReHistoGAN (used for image recoloring). As shown, we modified the last two blocks of our HistoGAN's to accept the latent feature passed from the first two blocks of our encoder. This modification helps our HistoGAN's head to consider both information of the input image structure and the target histogram in the recoloring process.

D.2 Training Details

We train our networks using an NVIDIA TITAN X (Pascal) GPU. For HistoGAN training, we optimized both the generator and discriminator networks using the diffGrad optimizer [99]. In all experiments, we set the histogram bin, h, to 64 and the fall-off parameter of our histogram's bins, τ, was set to 0.02. We adopted the exponential moving average of generator network's weights [196, 197] with the path length penalty, introduced in StyleGAN [197], every 32 iterations to train our generator network. Due to the hardware limitation, we used mini-batch of 2 with accumulated gradients every 16 iteration steps and we set the image's dimension, N, to 256. We set the scale factor of the Hellinger distance loss, α, to 2 (see Sec. D.3 for an ablation study).

As mentioned in Chapter 14, we trained our HistoGAN using several domain datasets, including: human faces [196], flowers [288], cats [84], dogs [200], birds [367], anime faces

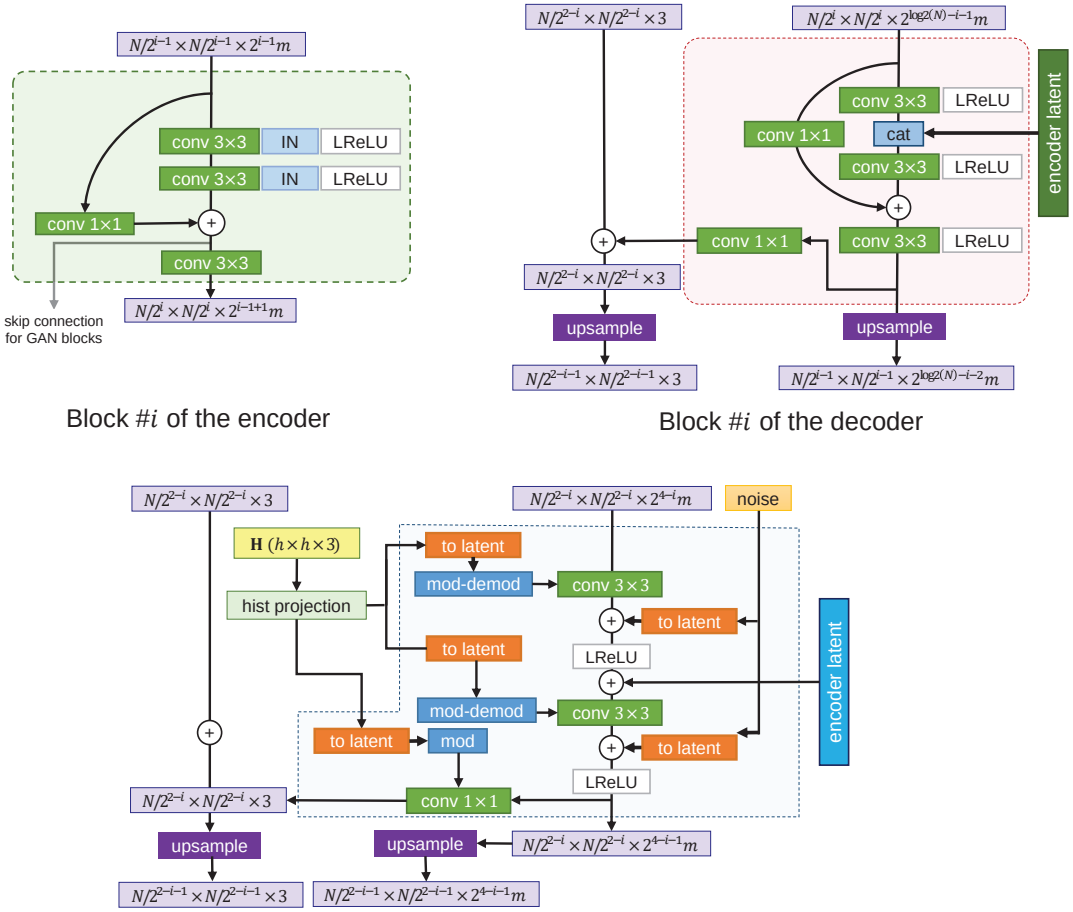


Figure D.2: Details of our ReHistoGAN network. We modified the last two blocks of our HistoGAN by adding a gate for the processed skipped features from the first two blocks of our encoder.

[81], human hands [10], bedrooms [393], cars [207], and aerial scenes [259]. We further trained our HistoGAN using 4,316 landscape images collected from Flickr. The collected images have one of the following copyright licenses: no known copyright restrictions, Public Domain Dedication (CC0), or Public Domain Mark.

To train our ReHistoGAN, we used the diffGrad optimizer [99] with the same mini-batch size used to train our HistoGAN. We trained our network using the following hy-

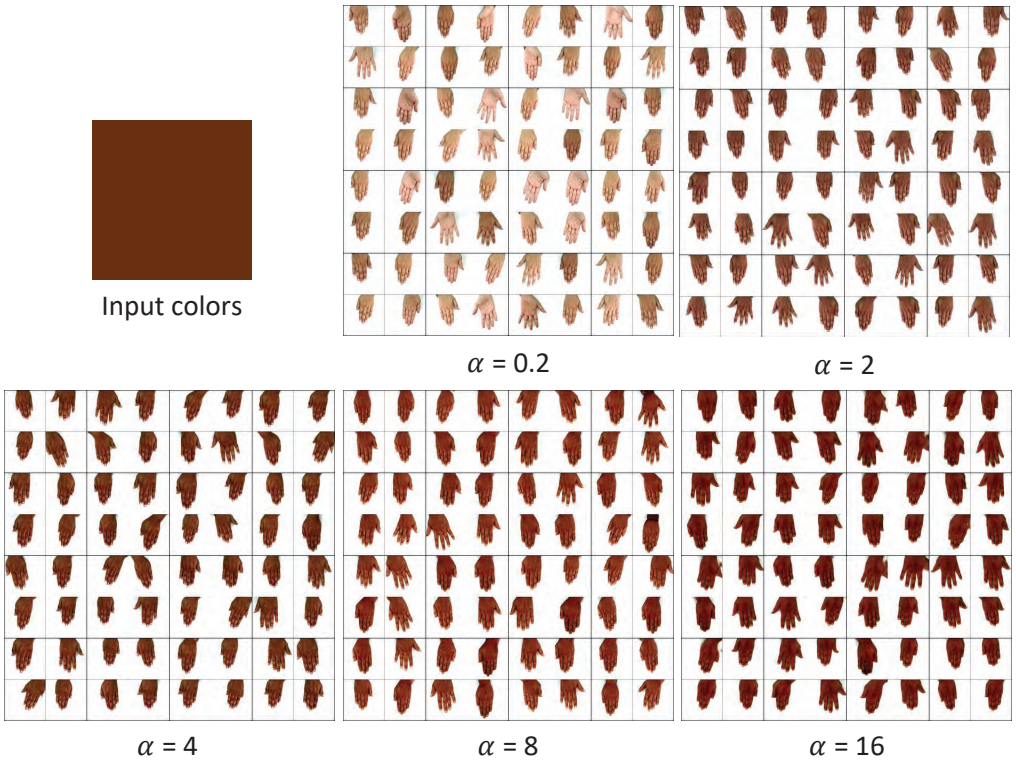


Figure D.3: Results obtained by training our HistoGAN in hand images [10] using different values of α.

perparameters $\alpha = 2$, $\beta = 1.5$, $\gamma = 32$ for 100,000 iterations. Then, we continued training using $\alpha = 2$, $\beta = 1$, $\gamma = 8$ for additional 30,000 iterations to reduce potential artifacts in recoloring.

D.3 Ablation Studies

We carried out a set of ablation experiments to study the effect of different values of hyperparameters used in Chapter 14. Additionally, we show results obtained by variations in our loss terms.

We begin by studying the effect of the scale factor, α, used in the loss function to train

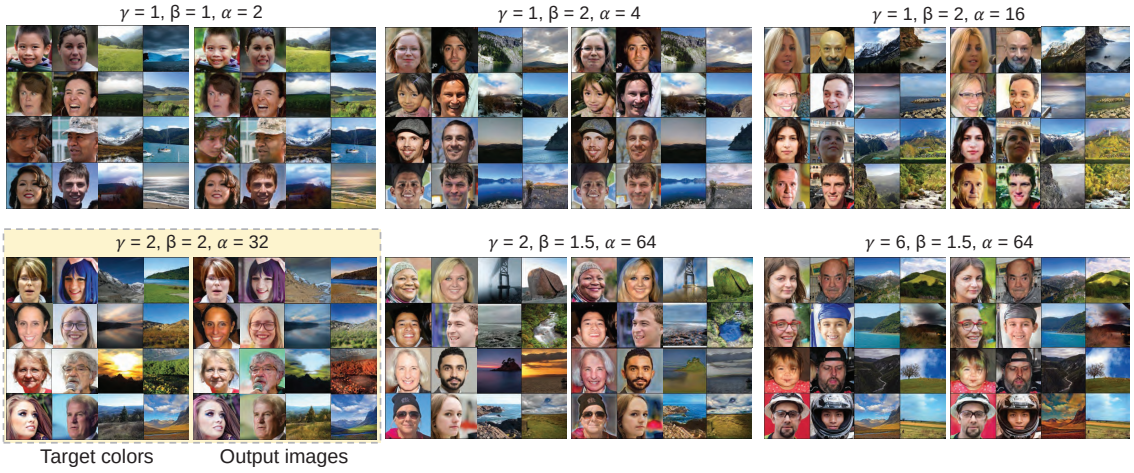


Figure D.4: Results of recoloring by training our recoloring network using different values of α, β, and γ hyperparameters. The highlighted results refer to the settings used to produce the reported results in Chapter 14 and this appendix.

our HistoGAN. This scale factor was used to control strength of the histogram loss term. In this set of experiments, we used the 11K Hands dataset [10] to be our target domain and trained our HistoGAN with the following values of α: 0.2, 2, 4, 8, and 16. Table D.1 shows the evaluation results using the Frechét inception distance (FID) metric [164], the KL divergence, and Hellinger distance. The KL divergence and Hellinger distance were used to measure the similarity between the target histogram and the histogram of GAN-generated images. Qualitative comparisons are shown in Fig. D.3

Figure D.4 shows examples of recoloring results obtained by trained ReHistoGAN models using different combination values of α, β, γ. As can be seen, a lower value of the scale factor, α, of the histogram loss term results in ignoring our network to the target colors, while higher values of the scale factor, γ, of the discriminator loss term, make our method too fixated on producing realistic output images, regardless of achieving the recoloring (i.e., tending to re-produce the input image as is).

Figure D.5: Results of two different kernels used to compute the reconstruction loss term.

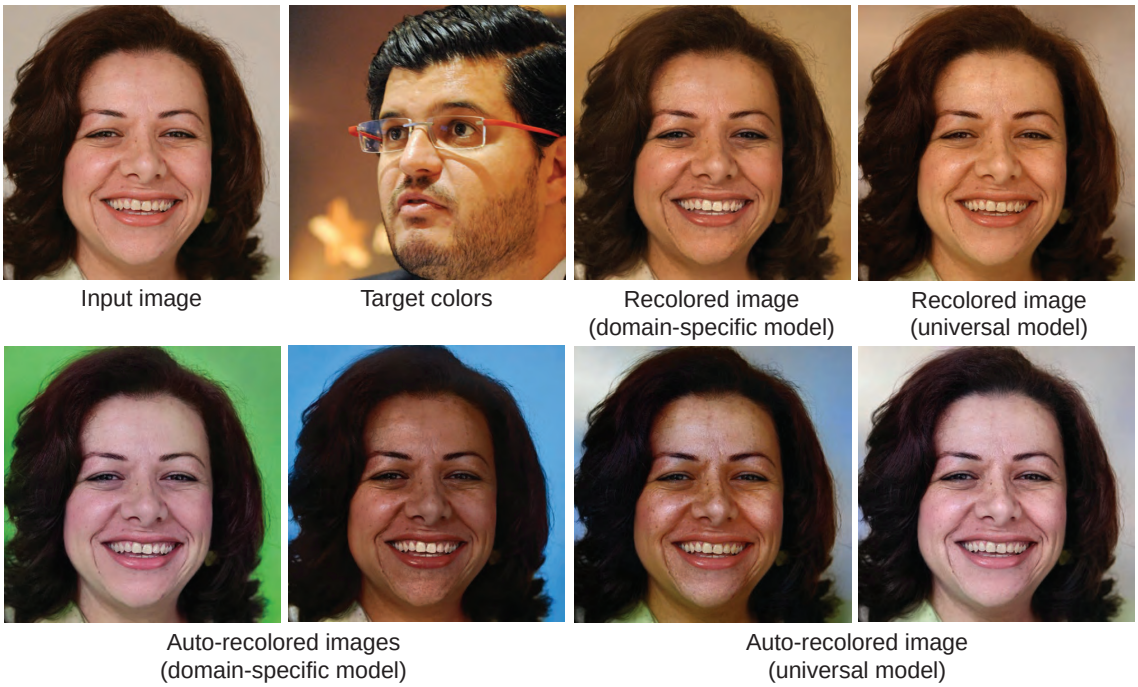


Figure D.6: Results of domain-specific and universal ReHistoGAN models. We show results of using a given target histogram for recoloring and two examples of the auto recoloring results of each model.

In the recoloring loss, we used a reconstruction loss term to retain the input image's spatial details in the output recolored image. Our reconstruction loss is based on the derivative of the input image. We have examined two different kernels, which are: the vertical and horizontal 3×3 Sobel kernels (i.e., the first-order directional derivative approx-

Table D.1: Results of our HistoGAN using different values of α. In this set of experiments, we used the Hands dataset [10] as our target domain. The term FID stands for the Frechét inception distance metric [164]. The term KL Div. refers to the KL divergence between the histograms of the input image and generated image, while the term H. dis. refers to Hellinger distance.

α	FID	RGB-uv hist.	
		KL Div.	H dist.
0.2	1.9950	0.3935	0.3207
2	**2.2438**	**0.0533**	**0.1085**
4	6.8750	0.0408	0.0956
8	9.4101	0.0296	0.0822
16	15.747	0.0237	0.0743

imation) and the 3×3 Laplacian kernel (i.e., the second-order isotropic derivative). We found that training using both kernels give reasonably good results, while the Laplacian kernel produces more compelling results in most cases; see Fig. D.5 for an example.

D.4 Universal ReHistoGAN Model

As the case of most GAN methods, our ReHistoGAN targets a specific object domain to achieve the image recoloring task. This restriction may hinder the generalization of our method to deal with images taken from arbitrary domains. To deal with that, we collected images from a different domain, aiming to represent the "universal" object domain.

Specifically, our training set of images contains ∼2.4 million images collected from different image datasets. These datasets are: collection from the Open Images dataset [212],

Figure D.7: Auto recoloring using our universal ReHistoGAN model.

the MIT-Adobe FiveK dataset [62], the Microsoft COCO dataset [241], the CelebA dataset [246], the Caltech-UCSD birds-200-2011 dataset [367], the Cats dataset [84], the Dogs dataset [200], the Cars dataset [207], the OxFord Flowers dataset [288], the LSUN dataset [393], the ADE20K dataset [409], and the FFHQ dataset [196]. We also added Flickr images collected using the following keywords: `landscape`, `people`, `person`, `portrait`, `field`, `city`, `sunset`, `beach`, `animals`, `living room`, `home`, `house`, `night`, `street`, `desert`, `food`. We have excluded any grayscale image from the collected image set.

We trained our "universal" model using $m = 18$ on this collected set of 2,402,006 images from several domains. The diffGrad optimizer [99] was used to minimize the same generator

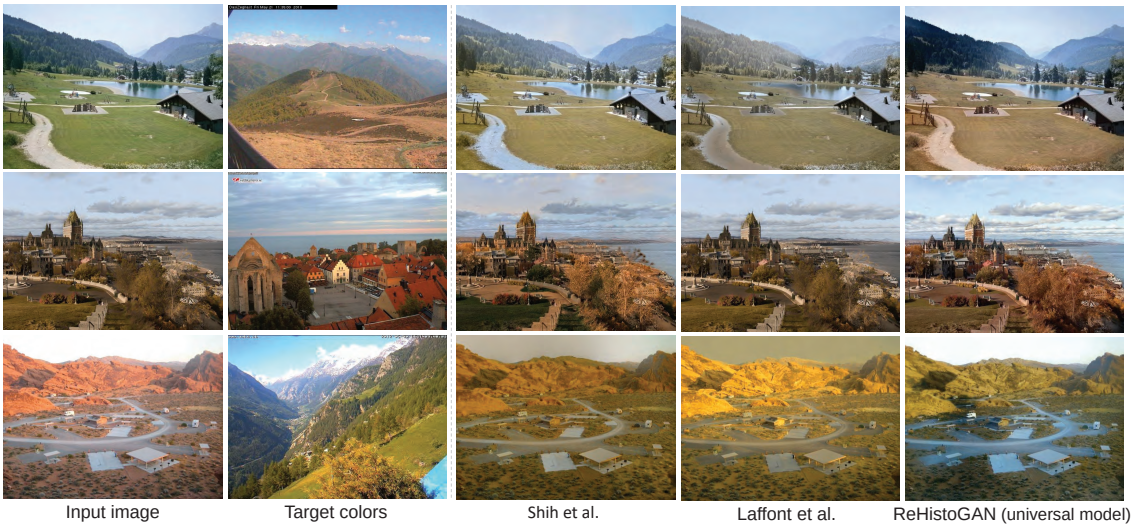


Figure D.8: Comparisons between our universal ReHistoGAN, and the methods proposed by Shih et al., [339] and Laffont et al., [216] for color transfer.

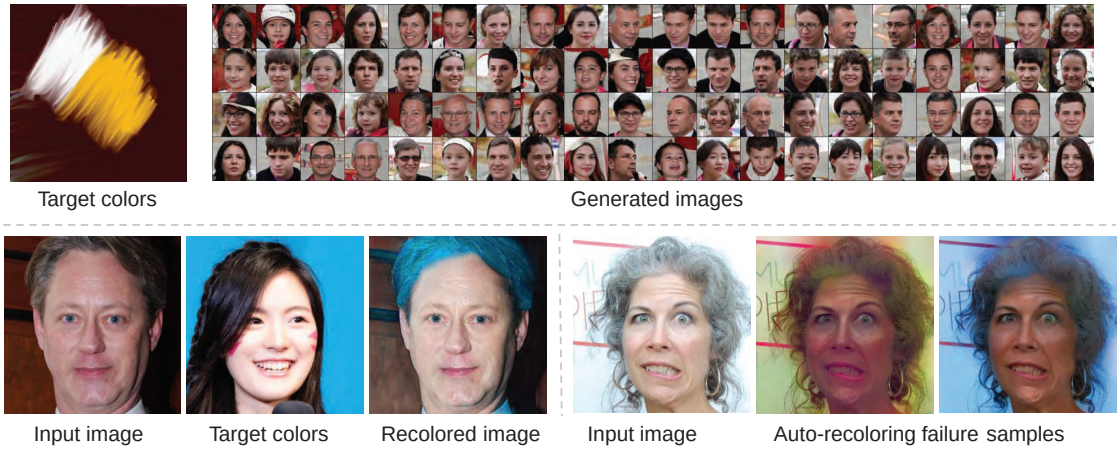


Figure D.9: Failure cases of HistoGAN and ReHistoGAN. Our HistoGAN fails sometimes to consider all colors of target histogram in the generated image. Color bleeding is another problem that could occur in ReHistoGAN's results, where our network could not properly allocate the target (or sampled) histogram colors in the recolored image.

Figure D.10: To reduce potential color bleeding artifacts, it is possible to apply a post-color transfer to our initial recolored image colors to the input image. The results of adopting this strategy are better than applying the color transfer to the input image in the first place. Here, we use the color transfer method proposed by Pitié and Kokaram [304] as our post-color transfer method. We also show the results of directly applying Pitié and Kokaram's [304] method to the input image.

loss described in Chapter 14 using the following hyperparameters $\alpha = 2$, $\beta = 1.5$, $\gamma = 32$ for 150,000 iterations. Then, we used $\alpha = 2$, $\beta = 1$, $\gamma = 8$ to train the model for additional 350,000 iterations. We set the mini-batch size to 8 with an accumulated gradient every 24 iterations. Figure D.6 show results of our domain-specific and universal models for image recoloring. As can be seen, both models produce realistic recoloring, though the universal model tends to produce recolored images with less vivid colors compared to our domain-specific model. Additional examples of auto recoloring using our universal model are shown in Fig. D.7.

In Fig. D.8, we show qualitative comparisons of the recoloring results using our universal ReHistoGAN and the method proposed in [216].

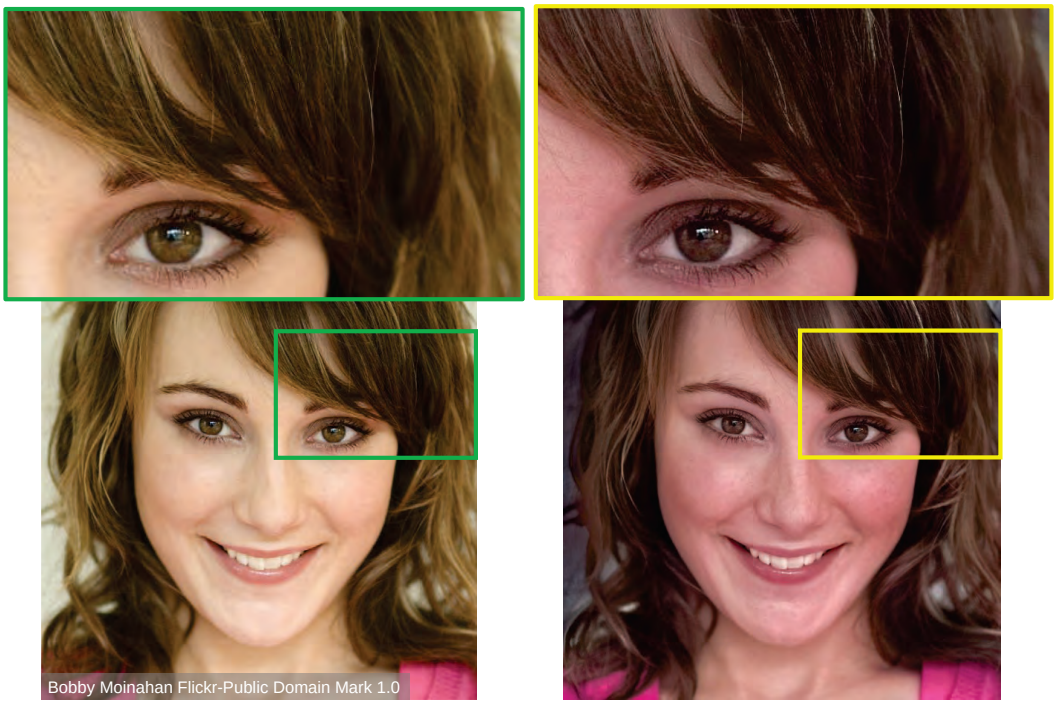


Figure D.11: We apply the bilateral guided upsampling [73] as a post-processing to reduce potential artifacts of dealing with high-resolution images in the inference phase. In the shown example, we show our results of recoloring using an input image with 2048×2048 pixels.

D.5 Limitations

Our method fails in some cases, where the trained HistoGAN could not properly extract the target color information represented in the histogram feature. This problem is due to the inherent limitation of the 2D projected representation of the original target color distribution, where different colors are mapped to the same chromaticity value in the projected space. This is shown in Fig. D.9-top, where the GAN-generated images do not have all colors in the given target histogram. Another failure case can occur in image

recoloring, where the recolored images could have some color-bleeding artifacts due to errors in allocating the target/sampled histogram colors in the recolored image. This can be shown in Fig. D.9-bottom

D.6 Post-Processing

As discussed in Sec. D.5, our method produces, in some times, results with color bleeding, especially when the target histogram feature has unsuitable color distribution for the content of the input image. This color-bleeding problem can be mitigated using a post-process color transfer between the input image and our initial recoloring. Surprisingly, this post-processing mapping produces results better than adopting the mapping in the first place—namely, applying the color transfer mapping without having our intermediate recoloring result.

Figure D.10 shows an example of applying Pitié, and Kokaram's method [304] as a post-processing color transfer to map the colors of the input image to the colors of our recolored image. In the shown figure, we also show the result of using the same color transfer method – namely, Pitié and Kokaram's method [304] – to transfer the colors of the input image directly to the colors of the target image. As shown, the result of using our post-process strategy has a better perceptual quality.

Note that except for this figure (i.e., Fig. D.10), we *did not* adopted this post-processing strategy to produce the reported results in Chapter 14 or this appendix. We discussed it here as a solution to reduce the potential color bleeding problem for completeness.

As our image-recoloring architecture is a fully convolutional network, we can process testing images in any arbitrary size. However, as we trained our models on a specific range of effective receptive fields (i.e., our input image size is 256), processing images with very high resolution may cause artifacts. To that end, we follow the post-processing ap-

proach used in Chapter 12 to deal with high-resolution images (e.g., 16-megapixel) without affecting the quality of the recolored image.

Specifically, we resize the input image to 256×256 pixels before processing it with our network. Afterward, we apply the bilateral guided upsampling [73] to construct the mapping from the resized input image and our recoloring result. Then, we apply the constructed bilateral grid to the input image in its original dimensions. Figure D.11 shows an example of our recoloring result for a high-resolution image (2048×2048 pixels). As can be seen, our result has the same resolution as the input image with no artifacts.

E List of Publications

Patents Applications

1. **Mahmoud Afifi** and Michael S. Brown. Sensor-Independent Illuminant Estimation for Deep Learning Models. **US Patent Application**, 2021.

2. **Mahmoud Afifi**, Michael S. Brown, Brian Price, and Scott Cohen. Image White Balancing. **US Patent Application**, 2020.

3. Michael S. Brown, **Mahmoud Afifi**, Abdelrahman Abdelhamed, Hakki Can Karaimer, Abdullah Abuolaim, and Abhijith Punnappurath. System and Method of Processing of a Captured Image to Facilitate Post-processing Modification. **Worldwide Patent Application**, 2020.

4. **Mahmoud Afifi**, Michael S. Brown, Konstantinos Derpanis, and Björn Ommer. Network for Correcting Overexposed and Underexposed Images. **US Patent Application**, 2020 – to appear.

5. **Mahmoud Afifi** and Michael S. Brown. Apparatus and Method for White Balance Editing. **US Patent Application**, 2020 – to appear.

Publications

1. **Mahmoud Afifi**, Marcus A. Brubaker, and Michael S. Brown. HistoGAN: Controlling Colors of GAN-Generated and Real Images via Color Histograms. In IEEE Conference on Computer Vision and Pattern Recognition (**CVPR**), 2021 – to appear.

2. **Mahmoud Afifi**, Konstantinos G. Derpanis, Björn Ommer, and Michael S. Brown. Learning Multi-Scale Photo Exposure Correction. In IEEE Conference on Computer Vision and Pattern Recognition (**CVPR**), 2021 – to appear.

3. **Mahmoud Afifi**, Abdelrahman Abdelhamed, Abdullah Abuolaim, Abhijith Punnappurath, and Michael S. Brown. CIE XYZ Net: Unprocessing Images for Low-Level Computer Vision Tasks. IEEE Transactions on Pattern Analysis and Machine Intelligence (**TPAMI**), 2021 – to appear.

4. **Mahmoud Afifi** and Michael S. Brown. Deep White-Balance Editing. In IEEE Conference on Computer Vision and Pattern Recognition (**CVPR**), 2020.

5. Majed El Helou, Ruofan Zhou, Sabine Süsstrunk, Radu Timofte, **Mahmoud Afifi**, Michael S. Brown, et al.. AIM 2020: Scene Relighting and Illumination Estimation Challenge. In European Conference on Computer Vision Workshops (**ECCVW**), 2020 (**Runner-Up Award**).

6. **Mahmoud Afifi** and Michael S. Brown. Interactive White Balancing for Camera-Rendered Images. In Color and Imaging Conference (**CIC**), 2020.

7. **Mahmoud Afifi**, Jonathan T. Barron, Chloe LeGendre, Yun-Ta Tsai, and Francois Bleibel. Cross-Camera Convolutional Color Constancy. arXiv preprint arXiv:2011.11890,

2020 – submitted to **ICCV** 2021.

8. **Mahmoud Afifi** and Michael S. Brown. What Else Can Fool Deep Learning? Addressing Color Constancy Errors on Deep Neural Network Performance. In International Conference on Computer Vision (**ICCV**), 2019.

9. **Mahmoud Afifi**, Brian Price, Scott Cohen, and Michael S. Brown. When Color Constancy Goes Wrong: Correcting Improperly White-Balanced Images. In IEEE Conference on Computer Vision and Pattern Recognition (**CVPR**), 2019.

10. **Mahmoud Afifi**, Abhijith Punnappurath, Abdelrahman Abdelhamed, Hakki Can Karaimer, Abdullah Abuolaim, and Michael S. Brown. Color Temperature Tuning: Allowing Accurate Post-Capture White-Balance Editing. In Color and Imaging Conference (**CIC**), 2019 (**Best Paper Award**).

11. **Mahmoud Afifi**, Brian Price, Scott Cohen, and Michael S. Brown. Image Recoloring Based on Object Color Distributions. In **Eurographics - Short Papers**, 2019.

12. **Mahmoud Afifi** and Michael S. Brown. Sensor-Independent Illumination Estimation for DNN Models. In British Machine Vision Conference (**BMVC**), 2019.

13. **Mahmoud Afifi**, Abhijith Punnappurath, Graham Finlayson, and Michael S. Brown. As-Projective-As-Possible Bias Correction for Illumination Estimation Algorithms. Journal of the Optical Society of America A (**JOSA A**), 2019.