Modeling sRGB Camera Noise with Normalizing Flows

SHAYAN KOUSHA

A THESIS SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

GRADUATE PROGRAM IN COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

April, 2022

© Shayan Kousha, 2022

Abstract

Noise modeling and reduction are fundamental tasks in low-level computer vision. They are particularly important for smartphone cameras relying on small sensors that exhibit visually noticeable noise. There has recently been renewed interest in using data-driven approaches to improve camera noise models via neural networks. These data-driven approaches target noise present in the raw-sensor image before it has been processed by the camera's image signal processor (ISP). Modeling noise in the RAW-rgb domain is useful for improving and testing the in-camera denoising algorithm; however, there are situations where the camera's ISP does not apply denoising or additional denoising is desired when the RAW-rgb domain image is no longer available. In such cases, the sensor noise propagates through the ISP to the final rendered image encoded in standard RGB (sRGB). The nonlinear steps on the ISP culminate in a significantly more complex noise distribution in the sRGB domain and existing raw-domain noise models are unable to capture the sRGB noise distribution. We propose a new sRGB-domain noise model based on normalizing flows that is capable of learning the complex noise distribution found in sRGB images under various ISO levels. Our normalizing flows-based approach outperforms other models by a large margin in noise modeling and synthesis tasks. We also show that image denoisers trained on noisy images synthesized with our noise model outperforms those trained with noise from baseline models.

Acknowledgements

Throughout my master's I was lucky to receive a great amount of support from my supervisor, professors, family, and friends.

I would first like to thank my supervisor, Professor Marcus A. Brubaker, for providing support and guidance every step of the way. I have learned so much under his guidance. His valuable feedback and insights into the challenges I have faced allowed me to become a better researcher.

I would like to acknowledge Professor Michael S. Brown for his insightful feedback and ideas. He helped me bring this work to a higher level. I truly appreciate him being on my supervisory committee.

I would also like to thank my friends for their support throughout this process. I would like to single out one of my friends, Ali Maleky. He has always been there for me when I faced challenges and pushed me forward. I also have valued the technical help I received from him throughout this work.

Lastly, I would like to thank my parents and my brother for their constant support and encouragement. I could not have completed this thesis without their support. I am and will be forever grateful.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Modeling and reducing noise are long-standing problems in computer vision and image processing with a rich history (*e.g.*, [11, 19, 20]). While simple models, like simple additive white Gaussian noise (AWGN), have often been used in testing denoising methods, they are well-known not to be realistic and serves only as a rough approximation for real-world camera noise. When realistic noise models are needed, more sophisticated models, such as Poisson-Gaussian [9] or Heteroscedastic Gaussian models [8, 21], are used to model the noise distribution observed on camera sensors. While such models are more realistic than AWGN, they too are often not able to fully capture real camera noise distributions.

In recent years, several data-driven noise models have been proposed that learn the noise distribution directly from large datasets of noisy sensor images (*e.g.*, [24, 2, 4, 10, 22, 34, 35]). These methods primarily focus on modeling the noise present in the raw sensor images. Modeling noise in the RAW-rgb domain is useful as denoising algorithms

are typically applied by the camera's image signal processor (ISP) hardware. Such noise reduction is applied early in the ISP's processing pipeline (often in the Bayer processing stages) before the image is rendered to its final sRGB representtion with tonal and color photo-finishing algorithms which are applied to improve the image's perceptual quality.

It is not unusual, however, for the in-camera denoising to be disabled, not present or insufficient. Further, most cameras do not save in RAW-rgb by default, making sRGB images far more ubiquitous. In such cases, the noisy sensor image is rendered through the camera's ISP, including the photo-finishing algorithms that apply complex, nonlinear operations to manipulate the image's tonal and color values. The resulting noise distribution in the final sRGB is notably more complex than in the unprocessed RAW-rgb space and existing RAW-rgb focused noise models become ineffective for modeling the sRGB noise, as shown in Fig. 1.1.

1.1 Contributions

We focus on modeling and synthesizing sRGB image noise, where the in-camera nonlinear processing has altered the noise characteristics from that of the camera's sensor. We begin with an analysis that shows that existing noise models targeting sensor noise in the RAW-rgb domain are not well suited for sRGB images. We then propose a generative model that combines recent advances in normalizing flows and captures effects of different gain (ISO) settings and camera types on sRGB image noise. We show that our sRGB noise model is superior to several baseline noise models. We further investigate our model's ability to synthesize noise by training a denoiser using noisy images sampled from the model. We

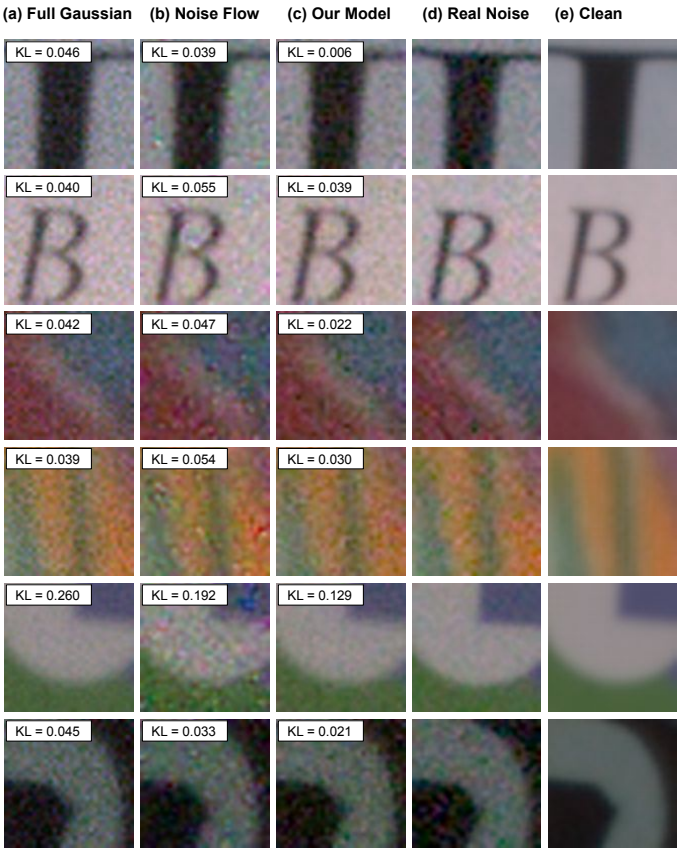


Figure 1.1: Noisy images generated by (a) a full covariance Gaussian model, (b) Noise Flow [2], and our model (c) compared with real images (d). Images are from the SIDD dataset [1].

show that this denoiser achieves significantly higher performance compared to denoisers trained on synthesized data from baseline models.

The proposed model and experiments have been published at IEEE CVPR conference 2022 [18].

Chapter 2

Related work

Additive white Gaussian noise (AWGN) [25, 29, 33] is a simple and intuitive model that has long been used by computer vision researchers to mimic image noise. However, it is well known that real camera noise is non-Gaussian, in part because it fails to capture signal-dependence of the variance. A more realistic model often used by camera manufacturers is the heteroscedastic Gaussian model [21], defined as:

$$\mathbf{N} \sim \mathcal{N}(0, \beta_1 \mathbf{I} + \beta_2), \tag{2.1}$$

where $\beta_1, \beta_2 > 0$ are parameters that model the signal-dependent and signal-independent nature of noise observed on real camera sensors. Some cameras even include the parameters of the manufacturer-calibrated heteroscedastic Gaussian noise model in their saved raw-sensor files encoded in the DNG format [1, 2], although recent work [36] has revealed that such parameters are often not well calibrated. The heteroscedastic Gaussian noise model is relatively simple and has few parameters; however, it is still only an approxima-

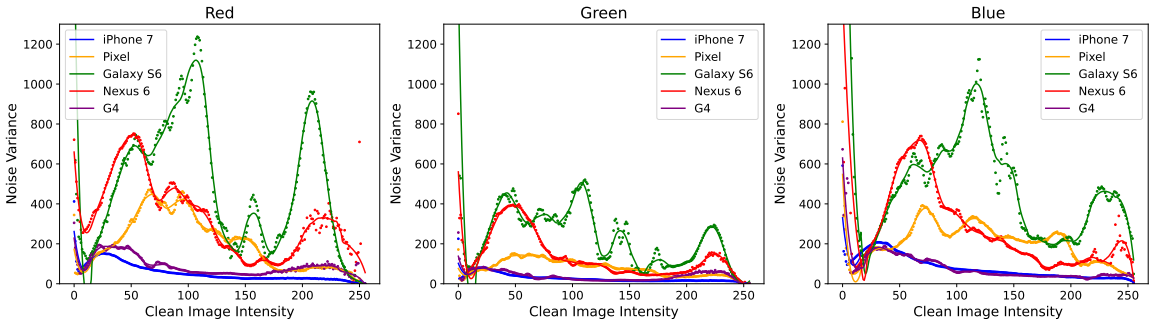


Figure 2.1: sRGB noise variance as a function of clean image intensity on SIDD [1]. These plots exhibit highly variable and unpredictable behavior, in contrast to RAW-rgb where there is typically a linear relation between noise variance and intensity. Consequently, we cannot use traditional image noise models like AWGN and NLF and instead propose a novel sRGB focused noise model based on normalizing flows.

tion of the real sensor noise [32, 1, 13, 8, 28].

Researchers have recently begun exploring data-driven approaches. For example, Abdelhamed et al. [2] proposed the Noise Flow model which combined the domain knowledge of signal and gain dependence with the expressiveness of learning-based generative models based on normalizing flows to capture more complex components of the noise. While the Noise Flow model was effective in simulating RAW-rgb noise, it relied on assumptions that do not apply in the sRGB color space. For example, the Noise Flow model builds off the heteroscedastic Gaussian model, which assumes the noise variance linearly depends on the underlying clean image intensity. However, this assumption no longer applies in the sRGB image domain (see Fig. 2.1) due to subsequent non-linear and potentially content-dependent processing of the image. As a result, in our experiments we show that simply applying the Noise Flow model to sRGB data fails to capture the noise distribution.

There have been relatively few attempts to model noise from sRGB data. Nam et al.

[24] introduced a model that is designed for the specific use case of modeling noise and other degradations caused by JPEG compression. They consider a model with full covariance to capture correlations between color channels. Even though they show to get good results for their task, the base of their model is a Gaussian model which is known to be limiting for noise modeling. More recently, the C2N [14] model attempted to model noise using unpaired clean and noisy images using a generative adversarial network (GAN). However, due to the nature of GANs, evaluating its modeling capabilities directly using methods like likelihood is challenging.

In this thesis, we introduce a data-driven model for image noise in the sRGB domain that conditions on camera settings and the underlying clean image. Like Noise Flow [2], the model is built using normalizing flows [6, 16, 17] but avoids making unrealistic assumptions that apply only to RAW-rgb. The resulting model is shown to have state-of-the-art noise modeling capabilities for sRGB noise. Further, when training a denoiser using noisy samples from the proposed noise model, we show that the resulting denoiser significantly out-performs those trained with existing sRGB noise models.

Chapter 3

Preliminaries

Image noise is a combination of degradations introduced by multiple sources in the imaging process and physical limitations of sources like camera sensors. Many of these sources occur when first capturing the RAW-rgb image by the camera sensors. Subsequently, the RAW-rgb image (including noise) is subject to in-camera imaging process that transforms the image from the scene-referred RAW-rgb color space to the display-referred sRGB color space. Fig. 3.1 summarizes some of the main steps in the in-camera imaging pipeline. The results of the photo-finishing steps, many nonlinear in nature, introduce new sources of noise (*e.g.*, from clipping and sharpening) while amplifying and distorting the original sensor's noise distribution. The noise from these multiple sources can be characterized as:

$$\tilde{\mathbf{I}} = \mathbf{I} + \mathbf{N}, \tag{3.1}$$

where $\tilde{\mathbf{I}}$ is the observed, noisy image, $\mathbf{I}$ is the true, underlying clean image, and $\mathbf{N}$ is the the noise whose distribution may depend on $\mathbf{I}$. In this work, we aim to design a generative

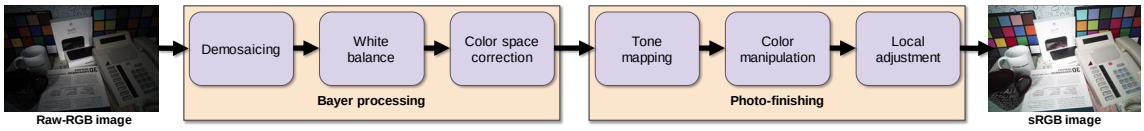


Figure 3.1: A typical camera ISP processing pipeline. This pipeline processes the RAW-rgb image to encode it in the sRGB domain. The nonlinear steps in the photo-finishing stage substantially complicate the noise distribution in the sRGB domain.

model that captures the complexity of noise introduced by all sources.

3.1 Normalizing Flows

Normalizing flows are a family of generative models that have gained popularity in recent years. Due to their formulation, they admit both efficient sampling and exact evaluation of probability density in contrast to other generative models, like GANs and VAEs [3, 17]. Additionally, flow-based models do not suffer from issues like mode and posterior collapse that are commonly faced when training GANs and VAEs.

Below we briefly introduce normalizing flows. A full review is beyond the scope of this thesis, and we refer the reader to recent review articles [17, 27] for a more extensive treatment. A normalizing flow consists of differentiable and bijective functions that learn a transformation $\mathbf{z} = f(\mathbf{x}|\Theta)$ with parameters Θ called a *flow*. A flow transforms data samples $\mathbf{x} \in \mathbb{R}^d$ from a complex distribution, $p_{\mathcal{X}}$, to some base space $\mathbf{z} \in \mathbb{R}^d$ with a known and tractable distribution and probability density function, $p_{\mathcal{Z}}$. Here, as is common, we will assume that $p_{\mathcal{Z}}$ takes the form of an isotropic Gaussian distribution with unit variance. The probability density function in the data space can then be found using the change of

variables formula

$$p_{\mathcal{X}}(\mathbf{x}) = p_{\mathcal{Z}}(f(\mathbf{x}|\Theta)) \left|\det \mathbf{D}f(\mathbf{x}|\Theta)\right|, \tag{3.2}$$

where $\mathbf{D}f(\mathbf{x})$ is the Jacobian matrix of f at $\mathbf{x}$. The result is a model that, given a dataset $D = \{\mathbf{x}_i\}_{i=1}^{M}$, can be trained by using stochastic gradient descent to minimize the negative log likelihood of the data

$$-\sum_{i=1}^{M} \log p_{\mathcal{Z}}(f(\mathbf{x}_i|\Theta)) + \log \left|\det \mathbf{D}f(\mathbf{x}_i|\Theta)\right| \tag{3.3}$$

with respect to the parameters Θ. Further, samples from the model can be drawn by generating samples from the base distribution $\mathbf{z} \sim p_{\mathcal{Z}}$ and then applying the inverse of the flow $\mathbf{x} = f^{-1}(\mathbf{z})$.

Formally, normalizing flows define distributions over continuous spaces. To apply them to quantized data (*e.g.*, as sRGB data which is typically truncated to 256 intensity levels) some attention must be paid to avoid a specific form of overfitting [30] that occurs when fitting continuous density models to discrete data. Specifically, here we use uniform dequantization, which adds uniformly sampled noise to the images during training. More complex forms of dequantization are possible [12].

Constructing expressive, differentiable, and bijective functions is the primary research problem in normalizing flows, and there have been many attempts at this; see [17, 27] for a thorough review. A flow f is typically constructed by the composition of simpler flows—namely, $f = f_1 \circ ... \circ f_{N-1} \circ f_N$—since the composition of bijective functions is itself bijective. Analogous to adding depth in a neural network, composing more flows can increase the complexity of the resulting distribution $p_{\mathcal{X}}$. Individual flows are typically

constructed such that their inverse and Jacobian determinant are easily calculated. Next we review three common forms of bijection that we will use in our model.

Affine Coupling Affine coupling flows [6] are a simple, efficient and widely used form of flow. They work by splitting the input dimensions, $\mathbf{x} = (\mathbf{x}^A, \mathbf{x}^B)$, into two disjoint subsets, $\mathbf{x}^A, \mathbf{x}^B$. Then, one subset, $\mathbf{x}^A$, is unmodified but used to compute scale and translation factors, which are applied to the other subset, $\mathbf{x}^B$. Formally, an affine coupling layer is defined as $\mathbf{y} = (\mathbf{y}^A, \mathbf{y}^B)$, where $\mathbf{y}^A = \mathbf{x}^A$ and

$$\mathbf{y}^B = \mathbf{x}^B \odot f_s(\mathbf{x}^A|\Theta) + f_t(\mathbf{x}^A|\Theta),$$

where $\odot$ is the element-wise product. The functions f_s and f_t compute the scale and translation factors and can be arbitrary, for example, deep neural networks. The inverse of this layer is easily computed as $\mathbf{x}^B = \big(\mathbf{y}^B - f_t(\mathbf{y}^A|\Theta)\big) \oslash f_s(\mathbf{y}^A|\Theta)$. Further, the log determinant of this transformation is efficiently calculated as $\sum \log f_s(\mathbf{x}^A|\Theta)$, where the sum is taken over the output dimensions of f_s.

1x1 Convolution Coupling layers must change the way dimensions are split between layers. This can be done by a random permutations [5, 6] but the Glow model [16] introduced the use of 1x1 convolutions as an invertible transformation. In essence, these layers are full linear transformations applied channel-wise to the inputs. The inverse is simply the inverse linear transformation applied channel-wise and the log determinant term is the number of pixels times the log determinant of the linear transformation.

Neural Spline Similar to Affine coupling flows, neural spline flows [7] split the input dimensions, $\mathbf{x} = (\mathbf{x}^A, \mathbf{x}^B)$, into two disjoint subsets, $\mathbf{x}^A, \mathbf{x}^B$. The subset $\mathbf{x}^A$ remains unmodified but used to compute the parameters required to transform the second subset, $\mathbf{x}^B$. Formally, a neural spline flow layer is defined as $\mathbf{y} = (\mathbf{y}^A, \mathbf{y}^B)$, where $\mathbf{y}^A = \mathbf{x}^A$ and

$$\Theta_W, \Theta_H, \Theta_D = f(\mathbf{x}^A)$$
$$\mathbf{y}^B = g(\mathbf{x}^B | \Theta_W, \Theta_H, \Theta_D)$$

where f is an arbitrary network responsible for computing the widths, heights, and derivatives parameters used by function, g. This function partitions the input, $\mathbf{x}^B$ in to K bins and then define a monotonically-increasing rational-quadratic function within each bin. A rational-quadratic function takes the form of a quotient of two quadratic polynomials and is easily differentiable. They are invertible because they are monotonically-increasing, and their inverse is analytically calculated. Function g is an elementwise transformation. As a result, The Jacobian matrix of this coupling transform is diagonal and the Jacobian determinant is given by $\prod_i g_i'(\mathbf{x}_i^B | \Theta_W, \Theta_H, \Theta_D)$.

Chapter 4

Normalizing Flows for sRGB Noise

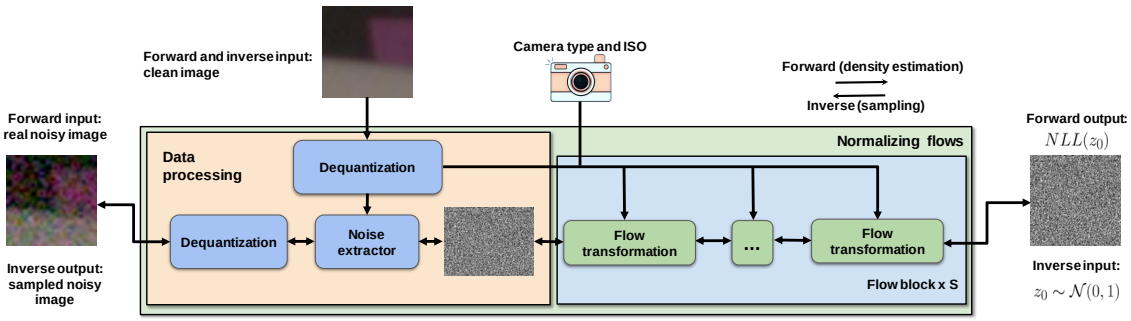


Figure 4.1: Our model consists of two main sections: (1) The data processing section is responsible for processing noisy and clean images. (2) The flow steps are responsible for learning the complex noise distribution.

Here we introduce our model of sRGB noise based on normalizing flows. Our analysis on the SIDD dataset [1], summarized in Fig. 2.1, confirms that real noise in the sRGB domain has a complex structure that is not captured by the standard, heteroscedastic-based noise models and varies significantly between cameras and even color channels. Inspired by this analysis, we introduce new conditional flows in the following sections that allow

the noise model to be conditioned on critical parameters, such as camera model, gain setting and clean intensity.

Figure 4.1 shows the proposed architecture of our model. Here we describe it as a transformation from the data (*i.e.*, a noisy, observed sRGB image $\tilde{\mathbf{I}}$) to the base space $\mathbf{z}$. First, the input images are dequantized with uniform dequantization as mentioned above. Unlike RAW-rgb, which is typically represented as a floating point number, sRGB data is typically quantized to 256 intensity levels. Note that, because both the clean and observed images have been quantized, they both need to be dequantized as well. Next, the clean image, $\mathbf{I}$, is subtracted from the observed image, $\tilde{\mathbf{I}}$, to get the noise image, $\mathbf{N}$. This is then followed by S flow blocks, which are responsible for learning a transformation from the noise to a sample in the base distribution, and vice versa. These flow blocks consist of a combination of one or more invertible 1x1 convolution layer and conditional layers introduced in the next section. In our experiments, we consider different block numbers with some reasonable combinations of the layers. Next, we describe the conditional layers.

Figs. 2.1 and 4.2 show the noise distribution has a complex dependency on both the underlying clean image and the camera and gain settings. However, unlike the RAW-rgb space where we have intuition about the noise distribution, we do not have a clear way of conditioning on these variables in the sRGB domain. As a result, we introduce novel conditional flow transformations for the task of noise modeling to capture this complexity. These transformations can be combined in many ways to form the flow block of our normalizing flow model. We consider several combinations in Section 5 to study the effectiveness of our conditioning approaches.

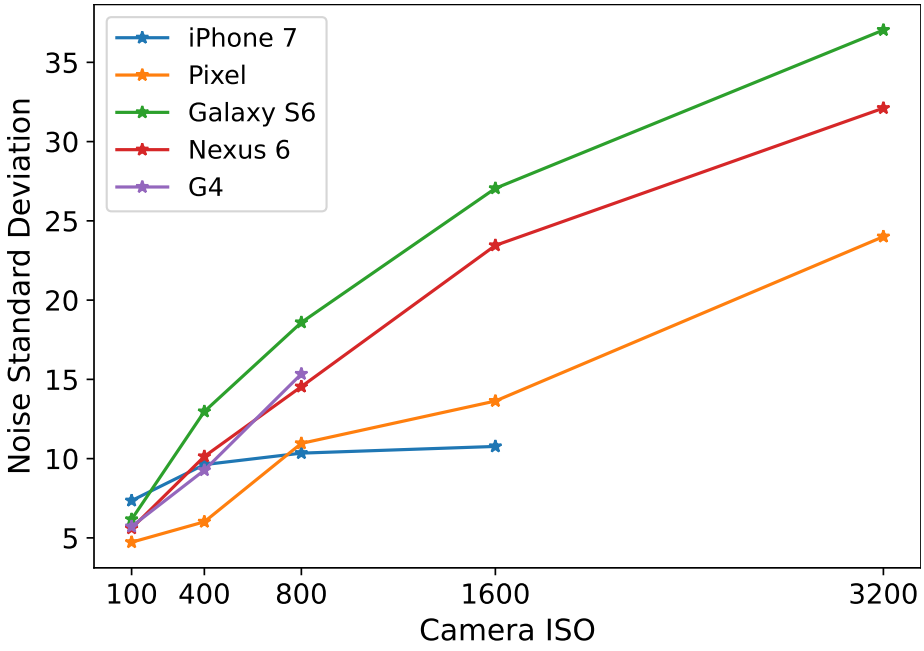


Figure 4.2: Real noise standard deviation changes as the sensitivity of the camera's sensor increases. The camera type is another factor affecting the noise behavior. For example, the noise standard deviation of images captured by Galaxy S6 is the highest under almost all ISO levels. Analysis done on the SIDD [1].

4.1 Conditional Affine Coupling (CAC)

This layer is an extension of the affine coupling layer presented in Section 3.1. To capture the complex dependence of the noise distribution on both the underlying clean image and the camera and gain settings (see Figs. 2.1 and 4.2), we extend the coupling layers to take these values as input. The conditional affine coupling layer is similar to the standard affine coupling layer but differs in the the scale and translation factors. Specifically, the output of the layer is $\mathbf{y} = (\mathbf{y}^A, \mathbf{y}^B)$ with $\mathbf{y}^A = \mathbf{x}^A$ and

$$\mathbf{y}^B = \mathbf{x}^B \odot f_{cs}(\mathbf{x}^A|\mathbf{I}, \mathbf{c}, \mathbf{g}) + f_{ct}(\mathbf{x}^A|\mathbf{I}, \mathbf{c}, \mathbf{g}),$$

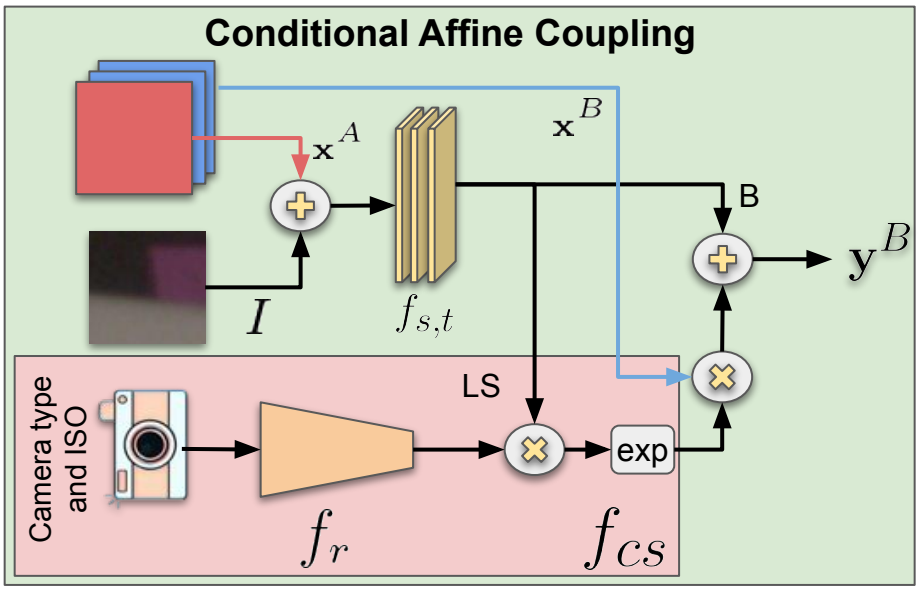


Figure 4.3: Forward pass of the conditional affine coupling layer. $f_{s,t}$ calculates the bias (B) and log_scale (LS) from the clean image and one channel of the input data. A rescaling term is calculated from the camera type and ISO setting and applied to LS by f_{cs} function. Later the rescaled LS is used along with the bias term and the remaining two channels of the input image to calculate $\mathbf{y}^B$. $\mathbf{y}^B$ is one part of the output.

where $\odot$ is the element-wise product, $\mathbf{x} = (\mathbf{x}^A, \mathbf{x}^B)$ is the input, and f_{cs} and f_{ct} are functions that compute the conditional scale and translation based on the input clean image, $\mathbf{I}$, camera one-hot encoding, $\mathbf{c}$, and gain setting one-hot encoding, $\mathbf{g}$. The scale and translation term have the shape as $\mathbf{x}^B$ and are applied elementwise. The inverse and log determinant of this transformation can be easily calculated, analogous to the (unconditional) coupling layer.

As shown in Figure 4.3, to condition on all three variables, namely the clean image, camera type, and ISO level, f_{cs} and f_{ct} functions consider these variables in two steps. First, the clean image is concatenated with one subset of the input data and passed to a CNN model ($f_{s,t}$) to calculate the log scale and bias factors. Then, the camera type and

gain setting are encoded into one-hot vectors separately. In our case, each encoding vector has a size of 5. The one-hot encoding vectors are concatenated and passed to a residual network ($f_r : \mathbb{R}^{10} \rightarrow \mathbb{R}$) to calculate a rescaling factor. This rescaling factor is multiplied by the log scale factor calculated earlier to get the final scaling values. The bias term is returned untouched. These functions have the form:

$$\begin{aligned} \mathbf{LS}, \mathbf{B} &= split(f_{s,t}(\mathbf{x}^A, \mathbf{I})) \\ f_{ct}(\mathbf{x}^A|\mathbf{I}, \mathbf{c}, \mathbf{g}) &= \mathbf{B} \\ f_{cs}(\mathbf{x}^A|\mathbf{I}, \mathbf{c}, \mathbf{g}) &= \exp(\mathbf{LS} * f_r(\mathbf{c}, \mathbf{g})) \end{aligned}$$

where $\mathbf{LS}$ and $\mathbf{B}$ are the log scale and bias factors, respectively.

This setup conditions on the clean image and the other two variables ($\mathbf{c}$ and $\mathbf{g}$) in independent steps. As a result, we have the option to skip one of the steps and only condition on a sub set of the variables.

4.2 Conditional Spline Coupling (CSC)

This layer is an extension of the neural spline layer presented earlier. Similar to our earlier reasoning based on Figures 2.1 and 4.2, we extend the neural spline transformation to condition on the clean image, camera type and gain setting. Here, we outline the procedure to condition on these variables:

As shown in Figure 4.4, the clean image, I, is concatenated channel-wise with $\mathbf{x}^A$ and passed to function f to calculate the parameters of the bins. Then, similar to the conditional affine coupling layer, the camera and gain settings are encoded into one-hot

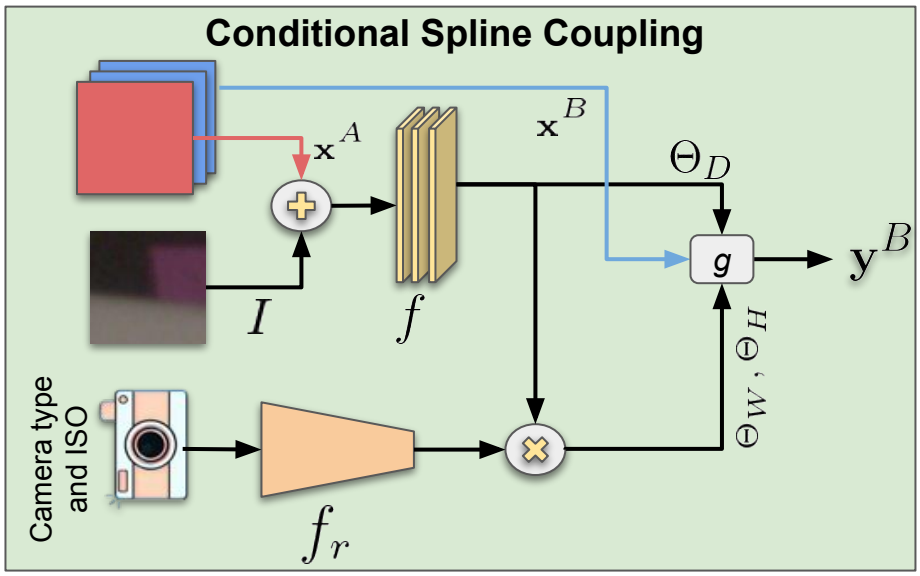


Figure 4.4: Forward pass of the conditional spline coupling layer. Function f calculates the bins' parameters from the clean image and one of the input channels. Later some of these parameters are rescaled using the rescale term calculated by f_r from the one-hot encodings of camera and gain settings. Finally, the updated parameters are passed to function g introduced in the neural spline paper [7]. This function is responsible for calculating the output of this layer.

vectors separately. $f_r : \mathbb{R}^{10} \rightarrow \mathbb{R}$, which is an arbitrary function, takes the resulting vectors as input and outputs a scale factor. In our experiments we use a residual network architecture. The scale factor is later used to rescale the bins' width and height parameters. The conditional spline coupling layer is defined as follows:

$$\Theta_W, \Theta_H, \Theta_D = f(\mathbf{x}^A, I)$$

$$\Theta_W, \Theta_H = (\Theta_W, \Theta_H) * f_r(\mathbf{c}, \mathbf{g})$$

$$\mathbf{y}^B = g(\mathbf{x}^B | \Theta_W, \Theta_H, \Theta_D)$$

where function g is the same function used in the standard neural spline flows. The inverse and log determinant of this layer can be calculated by following the same procedure used by the unconditional neural spline layer.

Similar to the affine coupling layer, the step that conditions on the clean image is independent of the step that conditions on the camera and gain. This allows us to use this layer in multiple ways in our experiments in Section 5.

4.3 Conditional Affine (CA)

The nature of the noise and the subsequent non-linear processing is heavily determined by the underlying noise free image and the specific camera and gain (or ISO) settings used. To account for this, we introduce linear flow layers, which are conditioned on combinations of important variables including the camera, $\mathbf{c}$, gain setting, $\mathbf{g}$, and the underlying clean image, $\mathbf{I}$. This layer comes in three forms depending on which subset of variables it is considering:

Conditioned on Camera and ISO ($CA_{\mathbf{c},\mathbf{g}}$) This layer has the following form:

$$\mathbf{y} = \mathbf{x} \odot f_s(\mathbf{c}, \mathbf{g}) + f_t(\mathbf{c}, \mathbf{g}), \tag{4.1}$$

where $\odot$ is the element-wise product, $\mathbf{x}$ is the input, $\mathbf{y}$ is the output, and f_s and f_t are functions that output the scale and translation factors. The functions f_s and f_t can be arbitrarily complex and have no constraints other than that $f_s \neq 0$. The inverse of this

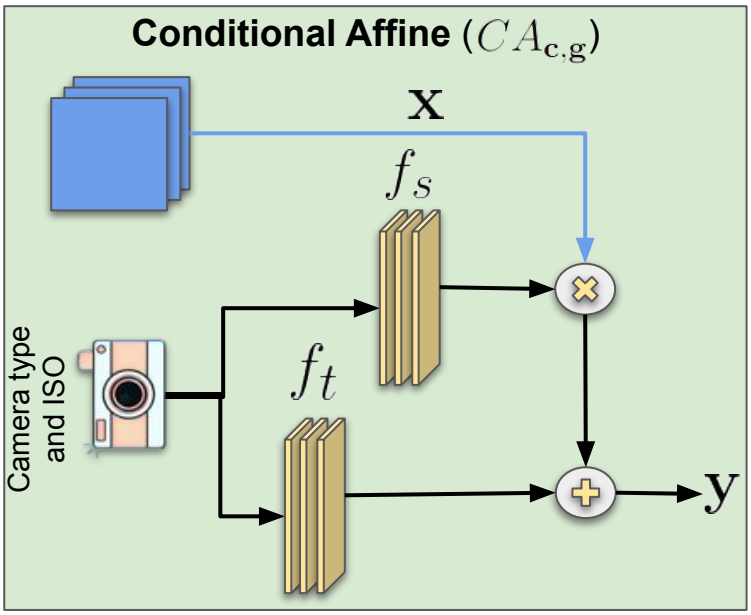


Figure 4.5: Forward pass of the conditional affine layer conditioned on the camera type and gain setting. Functions f_s and f_t are responsible for calculating the scale and bias terms, respectively. These terms are later applied to the input image $\mathbf{x}$ to calculate the output of this layer.

layer is easily calculated and is as follows:

$$\mathbf{x} = (\mathbf{y} - f_t(\mathbf{c}, \mathbf{g})) \oslash f_s(\mathbf{c}, \mathbf{g}), \tag{4.2}$$

where $\oslash$ is element-wise division. The log-determinant is given by $\sum \log f_s(\mathbf{c}, \mathbf{g})$, where the sum is taken over all dimensions of the input.

To condition on camera types and ISO levels, this layer creates a one-hot encoding of the camera-ISO pairs. For example, in our training and testing steps, we use images from five smartphones with five different ISO levels. This means the one-hot encoding of the pairs is a vector of size 25. $f_s : \mathbb{R}^{25} \to \mathbb{R}^3$ and $f_t : \mathbb{R}^{25} \to \mathbb{R}^3$ functions have one

scale and bias parameter for each channel for a given camera-iso representation. Since we have 25 camera-ISO pairs and are working with three-channel sRGB data, each of the two functions has a set of parameters with shape $(25, 3)$. The two functions use the one-hot vectors to index into these parameters. As a result, the output of each function is a vector of size 3 (one value per channel). The values corresponding to each channel are later applied to all pixels in that channel.

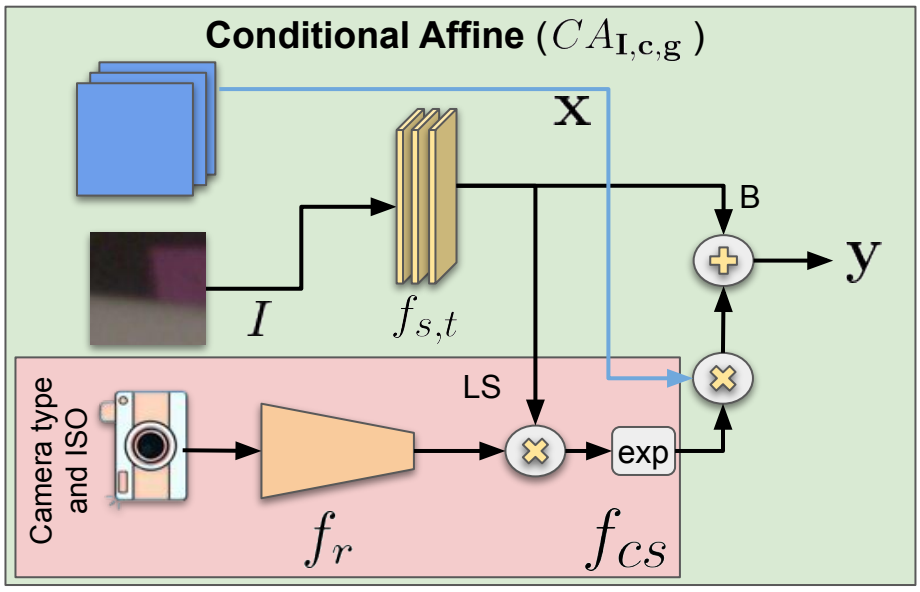


Figure 4.6: Forward pass of the conditional affine layer conditioned on the clean image, camera type and gain signal. $f_{s,t}$ calculates the bias (B) and log_scale (LS) from the clean image. The camera type and ISO setting are used to learn a rescaling factor which is later applied to LS by f_{cs} function. Finally, the rescaled LS is used along with the bias term and the input image to calculate $\mathbf{y}$.

Conditioned on Clean Image, Camera and ISO ($CA_{\mathbf{I},\mathbf{c},\mathbf{g}}$) This layer is similar to the conditional affine coupling layer. The only difference is that this layer does not split the input dimension. It calculates the scale and bias and applies them to all input dimensions.

This layer has the following form:

$$\mathbf{y} = \mathbf{x} \odot f_{cs}(\mathbf{I}, \mathbf{c}, \mathbf{g}) + f_{ct}(\mathbf{I}, \mathbf{c}, \mathbf{g}),$$

where $\mathbf{x}$ and $\mathbf{y}$ are the input and output of this layer, respectively. f_{cs} and f_{ct} are similar to the functions used by the affine coupling layer with not taking $\mathbf{x}^A$ as an input being the only difference. As a result, $f_{s,t}$, which is defined earlier, only uses the clean image to calculated the log scale and bias terms calculated. Later a rescaling factor is generated from the encoding of the camera type and ISO setting and applied to the log scale term. Similar to the CAC layer, the inverse and log determinant of this transformation can be easily calculated.

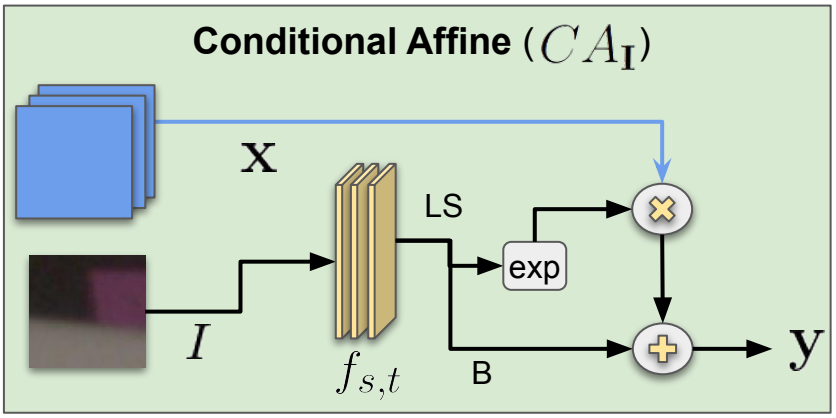


Figure 4.7: Forward pass of the conditional affine layer conditioned on the clean image. $f_{s,t}$ calculates the bias (B) and log_scale (LS) from the clean image. These factors are applied directly to the input image to calculate the output of this layer.

Conditioned on Clean Image ($CA_{\mathbf{I}}$) f_{cs} and f_{ct} of $CA_{\mathbf{I},\mathbf{c},\mathbf{g}}$ use a two step process to calculate the scale and bias terms. First, they calculated the log scale and bias terms.

Then, a rescale term is learned from variables such as gain and applied to the scale factor. These two steps are independent of each other and can be applied separately. In this layer, we only use the first step to learn the log scale and bias terms from the clean image. This layer has the following form:

$$\mathbf{LS}, \mathbf{B} = split(f_{s,t}((\mathbf{x}^A, \mathbf{I}))$$

$$\mathbf{y} = \mathbf{x} \odot \exp(\mathbf{LS}) + \mathbf{B}$$

Since the second step of f_{cs} and f_{ct} is not applied in this layer, information learned from the clean image is directly used to scale and shift the input data with out being rescaled. In Section 5 we investigate whether such direct transfer of information from the underlying clean representation to the noise image is beneficial.

Chapter 5

Experiments

To evaluate our model we use the Smartphone Image Denoising Dataset (SIDD) [1]. The SIDD-Medium split contains 320 noisy-clean image pairs captured under various ISO and lighting conditions taken by five different smartphones. While this dataset provides the data in both RAW-rgb and sRGB domains, here, we use only sRGB images. We extract approximately 3,000 patches of size 32x32 from each image. From these extracted patches, 80% are used for training and the remaining for validation. Patches are randomly distributed to ensure all cameras and ISO settings are fairly represented in both training and validation sets. For training we minimize the negative log likelihood (Eq. 3.3) using the Adam optimizer [15].

5.1 Metrics

To quantitatively evaluate the model we consider two metrics. First, the negative log likelihood per dimension (NLL) on the test set is used as a direct evaluation of density estima-

tion. Second, to better assess the quality of the sampled noise, we use the Kullback-Leibler (KL) divergence which was introduced in [2]. This metric computes the KL divergence between histograms of real and sampled noise. This metric is more sensitive to mismatches in the model's estimated variance than the NLL metric is.

5.2 Baselines

We explored a number of baseline sRGB noise models. We considered three variations of homoscedastic Gaussian noise: 1) AWGN, which assumes independent, isotropic noise at each pixel; 2) diagonal covariance Gaussian, which assumes independent but anisotropic noise at each pixel; and 3) full covariance Gaussian, which allows correlations between color channels. Note the full covariance Gaussian model was previously proposed for sRGB data by [24]. We also implemented a heteroscedastic Gaussian model, often referred to as the noise level function (NLF) and described in Eq. 2.1. While not expected to perform well based on, *e.g.*, Figure 2.1, it is a widely used and well known model of camera noise. Finally, we compared with a direct adaption of the Noise Flow model [2] to sRGB instead of RAW-rgb data. To do this we modified the number of channels that the architecture expected, but otherwise left it unchanged. Noise Flow is known to have a small number of parameters. To make the comparison fair, we create a larger Noise Flow, referred to as Noise Flow-Large, that follows the architecture of Noise Flow with the number of parameters increased to match the number of parameters we have in our model. All models were implemented in PyTorch. All Gaussian baselines were implemented as normalizing flows using combinations of simple linear flows, signal-dependent flows, and

Unconditional	ISO and Cam	Clean	Clean, ISO, and Cam	$NLL\,(\downarrow)$	$D_{KL}\,(\downarrow)$
SCx2	$\text{CA}_{\mathbf{c,g}}$	$\text{CA}_{\mathbf{I}}$	CSCx2	4.238	0.229
-	$\text{CA}_{\mathbf{c,g}}$	$\text{CA}_{\mathbf{I}}$	CSCx2	**3.076**	**0.141**
ACx2	$\text{CA}_{\mathbf{c,g}}$	$\text{CA}_{\mathbf{I}}$	CACx2	4.147	0.183
-	$\text{CA}_{\mathbf{c,g}}$	$\text{CA}_{\mathbf{I}}$	CACx2	**3.602**	**0.104**

Table 5.1: **Unconditional layer experiment.** Models in this table use 1 flow block (S = 1). Removing the unconditional layer and keeping conditional layers untouched improves the performance of the models. This trend is independent of the type of unconditional layer used in these models. Models that use unconditional/conditional affine coupling layers achieve better D_{KL} compared to the models that use unconditional/conditional spline coupling layers. A lower D_{KL} suggests better generated samples.

ISO and Cam	Clean	Clean, ISO, and Cam	NLL	D_{KL}
$\text{CA}_{\mathbf{c,g}}$	$\text{CA}_{\mathbf{I}}$	CACx2	3.602	0.104
$\text{CA}_{\mathbf{c,g}}$	CSCx2	CACx2	**2.901**	0.163
$\text{CA}_{\mathbf{c,g}}$	-	CACx2	3.473	**0.077**

Table 5.2: **Clean image only experiment.** Models in this table use 1 flow block (S = 1). Adding a specialized layer to only condition on the clean image harms D_{KL} but it might improve (reduce) NLL. We consider two specialized layers for conditioning on the clean image, namely $\text{CA}_{\mathbf{I}}$ and CSCx2. They both fail to improve the model performance in terms of D_{KL}.

gain-dependent flows [2] To ensure consistency.

5.3 Architecture search

In this section, we consider multiple architectures for the flow block based on the layers introduced in Section 4. The goal is to understand what combination of layers results in better modeling and generative performance. In our experiments, unless otherwise specified, all coupling layers are preceded by an invertible 1x1 convolution layer. Additionally, the coupling layers come in pairs. As a result, we have "x2" next to the coupling layers in

ISO and Cam	Clean, ISO, and Cam	NLL	D_{KL}
$\text{CA}_{\mathbf{c},\mathbf{g}}$	CACx2	3.473	0.077
$\text{CA}_{\mathbf{c},\mathbf{g}}$	CSCx2	**3.064**	0.159
$\text{CA}_{\mathbf{c},\mathbf{g}}$	$\text{CA}_{\mathbf{I},\mathbf{c},\mathbf{g}}$	3.636	**0.063**
CACx2	$\text{CA}_{\mathbf{I},\mathbf{c},\mathbf{g}}$	3.522	0.099

Table 5.3: **Conditioning on all variables.** Models in this table use 1 flow block (S = 1). The choice of transformation for conditioning on ISO, camera type, and clean image has a significant impact on the overall performance of the normalizing flows model. $\text{CA}_{\mathbf{c},\mathbf{g}}$ seems to be the best choice for conditioning on gain and camera as the models containing this layer achieve the best NLL and D_{KL}.

# Flow Blocks (S)	ISO and Cam	Clean, ISO, and Cam	NLL	D_{KL}
1	$\text{CA}_{\mathbf{c},\mathbf{g}}$	CACx2	3.473	0.077
1	$\text{CA}_{\mathbf{c},\mathbf{g}}$	$\text{CA}_{\mathbf{I},\mathbf{c},\mathbf{g}}$	3.636	0.063
4	$\text{CA}_{\mathbf{c},\mathbf{g}}$	CACx2	**3.072**	**0.044**
4	$\text{CA}_{\mathbf{c},\mathbf{g}}$	$\text{CA}_{\mathbf{I},\mathbf{c},\mathbf{g}}$	3.639	0.060

Table 5.4: **Model depth.** Increasing the number of flow blocks improves the noise modeling capabilities of the NF models. The model in row three achieves the best performance in terms of D_{KL} compared to other models.

our tables.

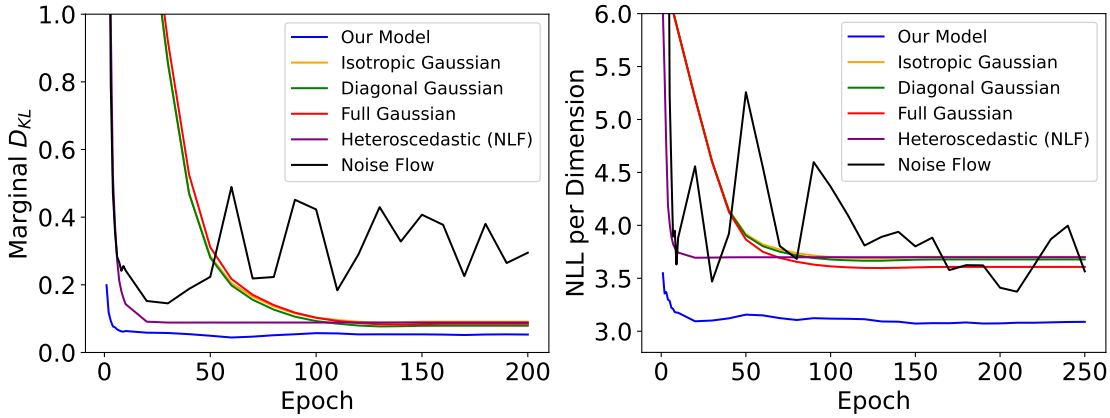
First, we investigate the effects of unconditional layers. Table 5.1 summarizes the results of experiment. The model defined in the first row uses two spline coupling layers as the unconditional step followed by some conditional layers. The model described in row two is similar to the first model with the exception that unconditional layers are eliminated. Even though this model has fewer parameters due to the lack of unconditional layers, it outperforms the first model in both metrics. Eliminating the unconditional layers results in a 1.162 nats/pixel improvement of NLL. The third row replaces the unconditional/conditional spline coupling transformations with unconditional/conditional affine coupling layers. A similar pattern of improvement emerges when this model is compared

with the last row where the unconditional layers are removed. Our last model achieves an NLL of 3.602 which translates to a 0.545 nats/pixel improvement. Additionally, when compared to the second row, the last row reveals that CAC is a better fit in terms of D_{KL} than CSC for conditioning on the clean image, camera type, and gain setting. As a result, we use the model in row four as a base for our next experiment.

The next experiment is designed to show the effects of layers only conditioned on the clean image. The three models shown in Table 5.2 are identical with their choice of layer for conditioning on the clean image being the only difference. The results suggest removing the layer that only conditions on the clean image improves the performance in terms of D_{KL} but might worsen the NLL results. We find D_{KL} to be a better indicator of the quality of the samples and prefer simpler model. Therefore, we conclude that there is no need to have a specialized layer to solely condition on the clean image. As shown in Table 5.2 the model defined in row three achieves a D_{KL} value of 0.077 which is the best result achieved so far. As a result, we include this model in Table 5.3 to help with our next experiment.

The previous experiments show that having an unconditional layer and a layer that only conditions on the clean image is not necessary. The third experiment focuses on the conditional transformations that either utilize ISO and camera type information or use all three variables. Table 5.3 summarizes the experiment and shows $\text{CA}_{\mathbf{c,g}}$_CACx2 model and $\text{CA}_{\mathbf{c,g}}$_$\text{CA}_{\mathbf{I,c,g}}$ with one flow block achieve the best D_{KL} performances with values of 0.077 and 0.063, respectively. Given the architectural design choices we made in Section 4, the models can easily be made deeper by increasing the number blocks.

Table 5.4 shows the performance of these two models when the number of blocks



(a) (b)

Figure 5.1: (a) Marginal KL divergence between the synthetic noise from the models and the real noise samples of the test set. (b) Testing NLL per dimension of our model compared to our baseline models. Our model not only performs by both metrics, it also converges faster.

is 4. The results suggest making a model deeper improves the results. For example, $\text{CA}_{\mathbf{c},\mathbf{g}}$_CACx2 with 4 repeated flow blocks achieves the best NLL with 0.401 nats/pixel improvement over the same model architecture with only one flow block. $\text{CA}_{\mathbf{c},\mathbf{g}}$_CACx2 model with four blocks outperforms the other models in this table in both metrics. It achieves NLL and D_{KL} of 3.072 and 0.044, respectively. Given that this model achieves the best D_{KL} among the models we investigate, we focus on this model for the remaining experiments in the following sections.

5.4 Results

Table 5.5 shows the final test NLL and KL divergence for our model and all the baselines. Figures 5.1a and 5.1b show the KL divergence and NLL during training of all models. The results demonstrate that our model achieves a lower (better) NLL (3.072 vs. 3.311

Model	NLL	D_{KL}	$\#Params$
Isotropic Gaussian	3.703	0.091	50
Diagonal Gaussian	3.678	0.079	150
Full Gaussian	3.608	0.085	525
Heteroscedastic (NLF)	3.642	0.088	72
Noise flow [2]	3.311	0.198	2330
Noise Flow-Large	3.288	0.227	6618
Our model	**3.072**	**0.044**	6160

Table 5.5: Test NLL and marginal D_{KL} for our model and our baselines. The proposed model outperforms the baselines in both metrics by a large margin. Noise Flow-Large and Noise Flow models have the closest NLL to the proposed method, however, their relatively high D_{KL} indicates that these models fail to generate realistic noise samples. The larger Noise Flow model follows the architecture of Noise Flow but has the same number of parameters as our proposed architecture, we see that while it marginally improves performance over the baseline Noise Flow model, it is still significantly outperformed by our proposed architecture. Together the results show that models originally developed for RAW-rgb are unlikely to be successful with sRGB.

nats/pixel for Noise Flow), and converges faster, requiring just a few epochs of training. During training the NLL of the Noise Flow model fluctuates significantly compared to the other baselines. We believe this is due to the assumptions made in the model architecture, specifically the signal-dependent layer, which do not hold in the sRGB domain and further demonstrate the need for a different approach to noise modeling in the sRGB domain.

In terms of marginal KL divergence our model also significantly improves over the baselines. Unlike with NLL, the closest performing baseline in terms of KL divergence was the diagonal Gaussian noise model, with a KL divergence of 0.079. In contrast, our model achieved a KL divergence of 0.044. For comparison, we also considered the recently proposed C2N [14] model, a GAN-based sRGB noise model. Their paper reported a KL divergence of 0.1638. However, we note that KL divergence is sensitive to the choice of histogram bins and other implementation details.

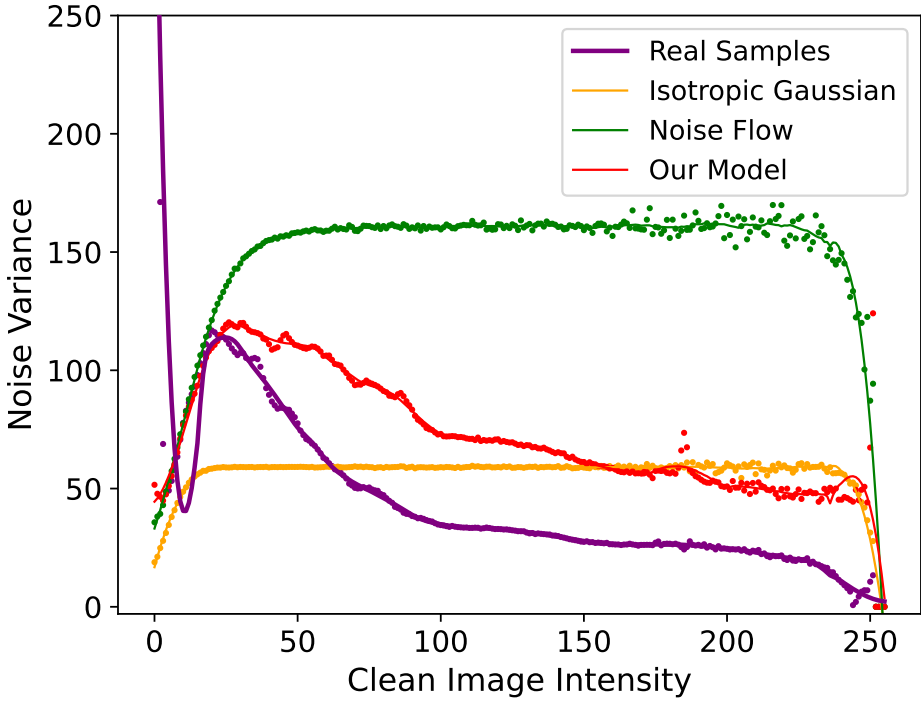


Figure 5.2: Red channel noise variance of real noise samples from iPhone7-ISO 100 setting and samples generated from our model and two of the baselines given clean images corresponding to the real noise samples. Our model success fully shows a similar noise variance trend to the real noise. However, the Noise Flow and Isotropic Gaussian models fail to learn this trend.

Interestingly, Table 5.5 shows that Noise Flow significantly outperformed the other baseline noise models in terms of NLL, but significantly underperformed them in terms of KL divergence. On further investigation, we found that the Noise Flow model was significantly overestimating the variance of the noise in many cases. For instance, Figure 5.2, similar to Figure 2.1, shows the variance of sRGB noise as a function of noise-free image intensity for real data, our model, Noise Flow, and the isotropic baseline for an iPhone7 with ISO level of 100. This graph shows that Noise Flow has badly overestimated the variance, even compared to the isotropic Gaussian model, suggesting that the difficulties

in training seen earlier are preventing it from converging to a reasonable model. In contrast, while our proposed model slightly overestimates the noise as well, it much better captures the structure of the relationship.

Qualitative Comparison. Noise samples can be qualitatively compared as well. Figure 5.3 shows samples of noise from our model and compared with the baseline models at different ISO levels. Samples from our model are generally more visually similar to those of real noisy images, particularly in comparison to the baselines. Noise Flow samples are too noisy and samples from Gaussian models do not exhibit enough variance. For example, the samples generated from full covariance Gaussian at ISO 800 is significantly less noisy compared to the real noisy image.

Modeling Different Cameras and ISO Settings. Figure 5.4 shows the learning of noise characteristics under different cameras and ISO settings. The dotted lines are the true noise standard deviation under each condition. The results show that the noise distribution changes drastically with different cameras and ISO levels. Further, our model is able to successfully capture this behavior to learn a more realistic noise model. The graphs also suggest that, while the model is quick to learn in the first few epochs and exhibits relatively little overfitting with the exception of ISO 3200. However we note that this ISO setting has a very limited number of samples in the training set.

Ablation Studies. Table 5.6 breaks down our model ($\text{CA}_{\mathbf{c,g}}$_CACx2) in its simpler building blocks and summaries their performance. The results show a significant improvement in noise modeling and noise synthesis when the conditional affine coupling

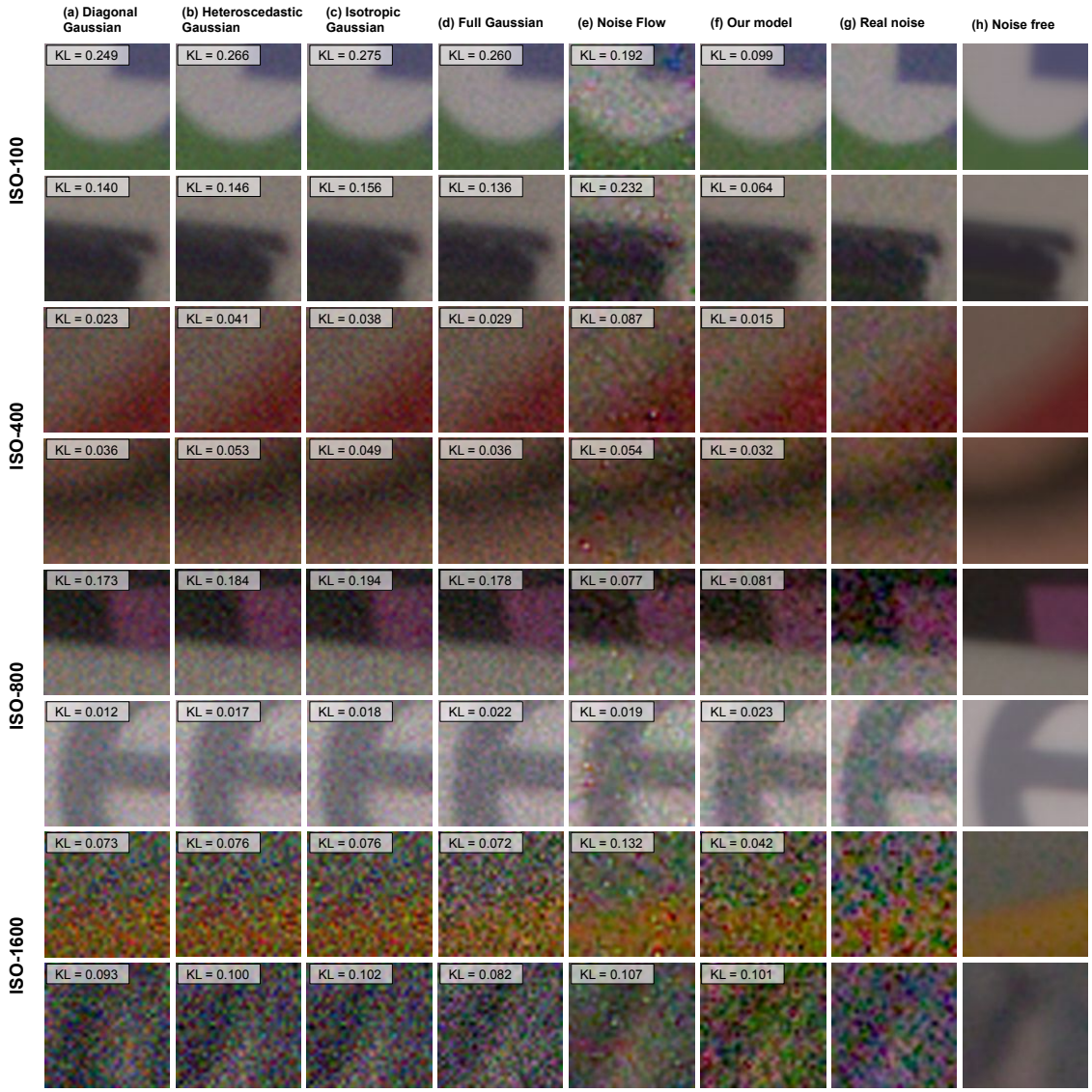
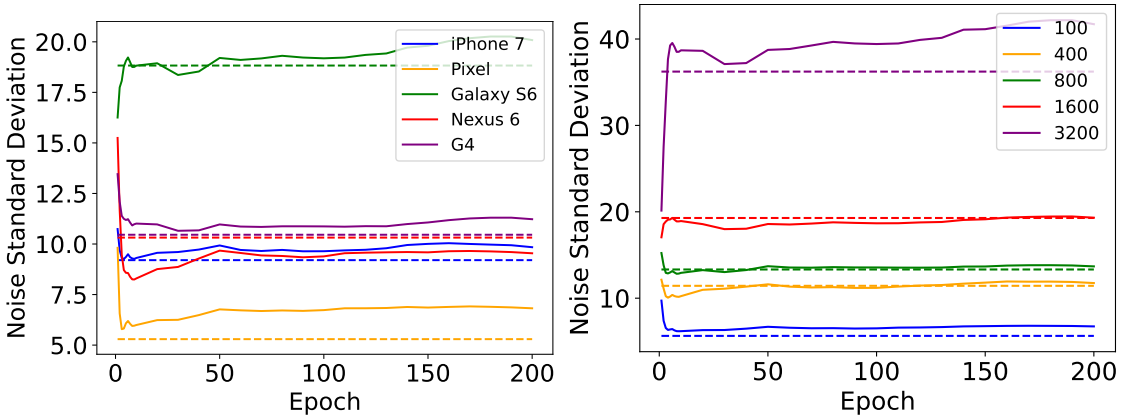


Figure 5.3: Generated samples from our model and all baselines. Samples from our model have noticeable visual similarities with the real noisy samples. Our samples achieve the lowest D_{KL} in almost all cases, showing its ability to generate realistic noise.



(a) (b)

Figure 5.4: Standard deviation of learned noise model during training under (a) different cameras and (b) different ISO levels is close to the real ones. This shows the proposed conditional layers have learned to adjust the distribution based on those settings. Dotted lines are the true values.

(CAC) layer conditions on all the important variables including the clean image, camera type, and ISO settings, rather than conditioning on only one of them. Additionally, we see an improvement by adding the conditional affine ($\text{CA}_{\mathbf{c},\mathbf{g}}$) layer to the conditional affine coupling layers (CAC), showing the importance of having a direct way of transferring knowledge from camera types and ISO levels. Finally, we show the importance of having multiple flow blocks. The architecture of $\text{CA}_{\mathbf{c},\mathbf{g}}$_CACx2 with four flow blocks achieves the best performance in both metrics, NLL and D_{KL}. This is the architecture used in other experiments.

# Flow Blocks (S)	Flow blocks	NLL	D_{KL}
1	$\text{CA}_{\mathbf{c},\mathbf{g}}$	3.678	0.079
1	$\text{CAC}_{\text{ISO only}}$x2	3.726	0.154
1	$\text{CAC}_{\text{camera only}}$x2	3.609	0.216
1	$\text{CAC}_{\text{clean image only}}$x2	3.882	0.295
1	CACx2	3.530	0.104
1	$\text{CA}_{\mathbf{c},\mathbf{g}}$_CACx2	3.473	0.077
2	$\text{CA}_{\mathbf{c},\mathbf{g}}$_CACx2	3.254	0.067
4	$\text{CA}_{\mathbf{c},\mathbf{g}}$_CACx2	**3.072**	**0.044**

Table 5.6: Test NLL and D_{KL} achieved by different flow steps. The symbols $\text{CA}_{\mathbf{c},\mathbf{g}}$ and CAC refer to the conditional affine conditioned on camera type and ISO and conditional affine coupling, respectively. Unless otherwise specified in the subscripts, the layers have the formulation mentioned in Chapter 4. Last row is the architecture we use in other experiments.

5.5 Application: sRGB Denoising

One of the main applications of noise modeling is to generate realistic noise to be used in downstream tasks, such as denoising. Here we explore the training of the standard DnCNN denoiser [34] using samples generated from the learned noise model to further test its noise generation capabilities.

Dataset To train the DnCNN model we use SIDD-Medium. In our experiments, the clean images are from SIDD-Medium and the noisy images are either the real ones provided with the dataset or generated by either the proposed noise model or one of the baseline models. We use noisy and clean images from SIDD-Validation and SIDD-Benchmark for validation and testing purposes, respectively. Refer to Fig. 5.5 for more details about using samples from noise models to train the denoiser.

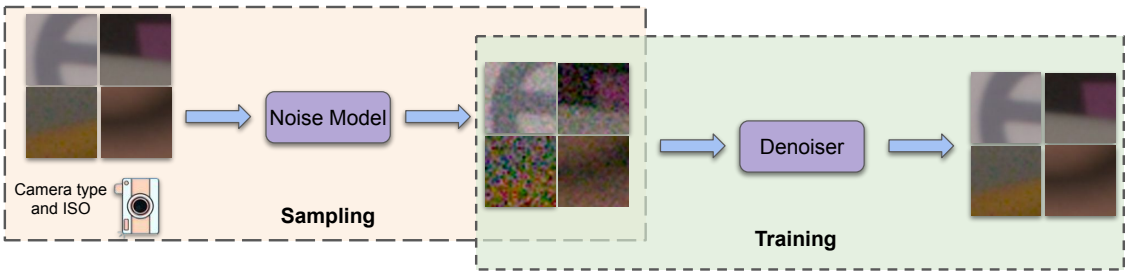


Figure 5.5: **Denoiser training process.** First, multiple clean images, camera types, and ISO settings are passed to the noise model to produce noisy samples. Then, these synthetic noisy samples are used to train the denoiser.

Results Table 5.7 summarizes the result of our denoising experiment. The results show that a DnCNN model trained on the synthetic noises from our model achieves a significantly higher performance in terms of peak signal-to-noise ratio (PSNR) and structural similarity (SSIM) [31] on SIDD-Benchmark compared to the denoisers trained on the samples from baseline models. While the performance of our model does not exceed the performance of a denoiser trained with real data (*e.g.*, as was found with noise models in RAW-rgb [2]), it does significantly shrink the gap.

Figure 5.6 shows denoising results of the DnCNN model trained with real noisy images and multiple noise synthesis strategies including our model and baselines. The figure also includes the input noisy image and the ground truth clean image for reference. The denoiser trained on noise samples from our model tends to produce denoised images that are closer to the ground truth clean images than when trained on samples from the baseline noise models. The model trained on noisy image samples from Noise Flow tends to not remove the noise fully, outputting images which still contain a significant amount of noise (*e.g.*, as in row 6). This is likely caused by the tendency of Noise Flow to significantly overestimate the noise variance as demonstrated in Figure 5.2. Finally, denoisers trained

Model	$PSNR$	$SSIM$
Isotropic Gaussian	32.48	0.855
Diagonal Gaussian	33.34	0.867
Full Gaussian	32.72	0.873
Heteroscedastic Gaussian	32.24	0.849
Noise Flow	33.81	0.894
C2N*	33.76	0.901
Our model	**34.74**	**0.912**
DnCNN-Real	36.51	0.922

Table 5.7: Performance of the DnCNN denoisers trained on samples from each model. These trained denoisers are evaluated on the SIDD Benchmark set. The DnCNN model trained on samples from our model achieves better performance compared to the DnCNN models trained on synthetic noise from baselines. (*) Results are reported from the paper [14].

on Gaussian samples tend to produce overly smooth denoised images, (*e.g.*, as in row 4 of full covariance Gaussian).

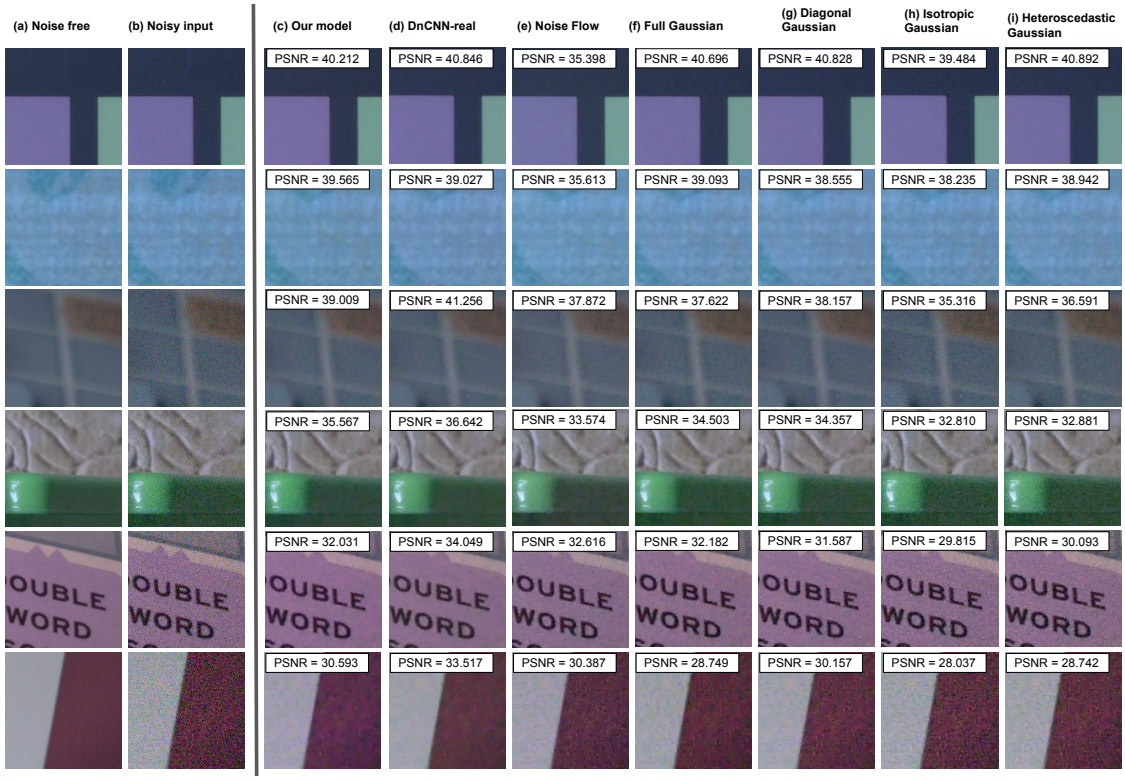


Figure 5.6: Denoising results on SIDD-Validation from denoisers trained on noisy images from (c) our model, (d) real noisy images of SIDD-Validation, and (e, i) all of our baselines.

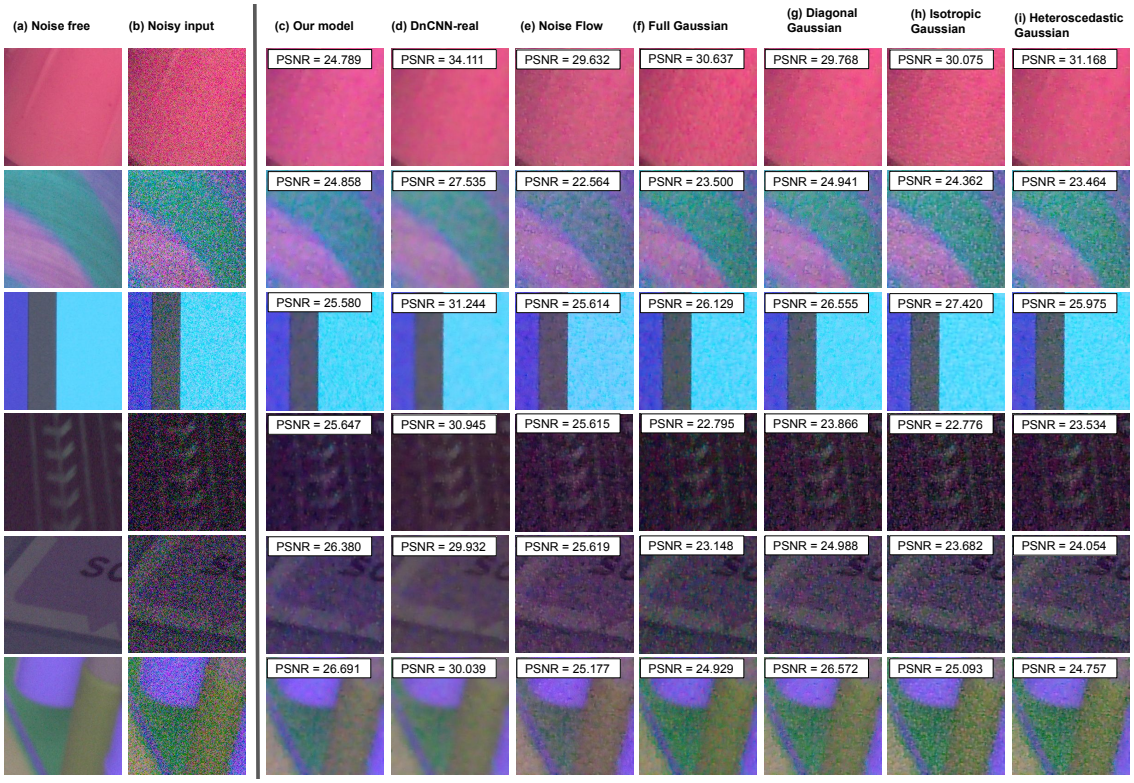


Figure 5.7: Poor denoising results on SIDD-Validation from denoisers trained on noisy images from (c) our model, (d) real noisy images of SIDD-Validation, and (e, i) all of our baselines. This figure shows some denoising failed cases.

Chapter 6

Conclusion

We introduced a noise model specifically tailored to capture image noise in the sRGB domain. Noise Flow [2] was our starting point as it has shown state-of-the-art performance in the RAW-rgb domain. However, initial experiments concluded the performance of this model is not sufficient in the sRGB domain. A subset of the results gathered from these experiments is included in Section 5. Similar experiments showed that other existing RAW-rgb noise models like the heteroscedastic Gaussian noise model are not effective at capturing noise in sRGB. These models are ineffective because image noise in the sRGB domain exhibits significantly more complex noise distributions than those found in the unprocessed RAW-rgb domain.

To address this complexity, we modified the signal dependant and gain layers of Noise Flow to eliminate parts of these layers that were designed based on assumptions that are true in the RAW-rgb domain but do not hold in sRGB. However, the performance gain was not significant. Please note this is not the same as the Noise Flow model we used as

a baseline in the experiments chapter. This was a preliminary experiment and we decided to not include its results in Chapter 5. Our experiments (*e.g*. Fig. 2.1) showed the noise distribution in sRGB is highly complex which is mainly due to the non-linearities introduced by the in-camera pipeline (Fig. 3.1) responsible for transforming captured images from RAW-rgb to sRGB. This phenomenon eliminates our ability to use simple and highly engineered layers such as the signal dependant layer introduced in the Noise Flow paper.

Further experimentation (*e.g*. Fig. 4.2) revealed that the noise and its characteristics such as standard deviation depend significantly on the camera type and gain setting. As a result, we proposed novel conditional layers based on recent normalizing flow architectures. Our layers are mainly based on affine coupling [6] and neural spline flows [7]. However, there are many more layers and flow transformations such as autoregressive flows [23, 26] that can be modified to condition on the variables and considered for the noise modeling task.

Our conditional layers can be combined in many ways to create a normalizing flows model for the task of noise modeling. It is challenging to consider and experiment with all these combinations. As a result, we designed our experiments to allow us reach final model with reasonable performance but the exploration is limited.

We demonstrated the effectiveness of the final noise model directly with NLL and KL divergence metrics and by training image denoisers using image noise synthesized by our model. We showed that our image denoisers significantly outperform other denoisers trained on existing noise models for sRGB.

6.1 Broader impact

Our model is designed in a way that allows us to generate unlimited noise samples. This will help researchers to generate enough training data for tasks such as denoising in fields that gathering data is difficult and costly. Medical imaging is an example of such fields.

6.2 Future work

Our conditional flow transformations focus on three variables, namely, the clean image, camera type, and gain setting. However, there are many more factors such as the scene's light source brightness levels that have significant impacts on the noise distribution. Our proposed transformations can be modified to condition on these new variables.

As mentioned above, there are many recent normalizing flow transformations that we did not consider in this work. These transformations each have interesting properties and behavior that if adjusted carefully, might improve performance when modeling camera noise in sRGB.

Our conditional flow transformations have shown to be effective in modeling the camera noise in sRGB. These transformations need to be combined to create an expressive normalizing flow model. In this work, we only investigated a few combinations. More work is required to expand our experiments to all possible combinations.

Bibliography

[1] Abdelrahman Abdelhamed, Stephen Lin, and Michael S. Brown. A high-quality denoising dataset for smartphone cameras. In *CVPR*, 2018. viii, ix, 3, 4, 5, 12, 14, 23

[2] Abdelrahman Abdelhamed, Marcus A Brubaker, and Michael S Brown. Noise Flow: Noise Modeling with Conditional Normalizing Flows. In *ICCV*, 2019. viii, 1, 3, 4, 5, 6, 24, 25, 29, 35, 39, 48

[3] Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G Willcocks. Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models. *arXiv preprint arXiv:2103.04922*, 2021. 8

[4] Chang Chen, Zhiwei Xiong, Xinmei Tian, and Feng Wu. Deep boosting for image denoising. In *ECCV*, 2018. 1

[5] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. In *ICLR Workshop*, 2015. 10

[6] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. In *ICLR*, 2017. 6, 10, 40

[7] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In *Neurips*, 2019. ix, 11, 17, 40

[8] Alessandro Foi. Clipped noisy images: Heteroskedastic modeling and practical denoising. *Signal Processing*, 89:2609–2629, 12 2009. 1, 5

[9] Alessandro Foi, Mejdi Trimeche, Vladimir Katkovnik, and Karen Egiazarian. Practical poissonian-gaussian noise modeling and fitting for single-image raw-data. *TIP*, 17(10):1737–1754, 2008. 1

[10] Shi Guo, Zifei Yan, Kai Zhang, Wangmeng Zuo, and Lei Zhang. Toward convolutional blind denoising of real photographs. In *CVPR*, 2019. 1

[11] Glenn E Healey and Raghava Kondepudy. Radiometric ccd camera calibration and noise estimation. *TPAMI*, 16(3):267–276, 1994. doi: 10.1109/34.276126. 1

[12] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *ICML*, 2019. 9

[13] Gerald C Holst. *CCD Arrays, Cameras, and Displays*. SPIE Optical Engineering Press, USA, second edition, 1996. 5

[14] Geonwoon Jang, Wooseok Lee, Sanghyun Son, and Kyoung Mu Lee. C2n: Practical generative noise modeling for real-world denoising. In *ICCV*, 2021. vii, 6, 29, 36

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 23

[16] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *NeurIPS*, 2018. 6, 10

[17] Ivan Kobyzev, Simon J.D. Prince, and Marcus A. Brubaker. Normalizing flows: An introduction and review of current methods. *TPAMI*, 43(11):3964–3979, 2021. 6, 8, 9

[18] Shayan Kousha, Ali Maleky, Marcus A. Brubaker, and Michael S. Brown. Modeling sRGB Camera Noise with Normalizing Flows. In *CVPR*, 2022. 3

[19] Darwin T. Kuan, Alexander A. Sawchuk, Timothy C. Strand, and Pierre Chavel. Adaptive noise smoothing filter for images with signal-dependent noise. *TPAMI*, 7 (2):165–177, 1985. 1

[20] Ce Liu, Richard Szeliski, Sing Bing Kang, C. Lawrence Zitnick, and William T. Freeman. Automatic estimation and removal of noise from a single image. *TPAMI*, 30(2):299–314, 2008. 1

[21] Xinhao Liu, Masayuki Tanaka, and Masatoshi Okutomi. Practical signal-dependent noise parameter estimation from a single noisy image. *TIP*, 23(10):4361–4371, 2014. 1, 4

[22] Yang Liu, Saeed Anwar, Liang Zheng, and Qi Tian. Gradnet image denoising. In *CVPR Workshop*, 2020. 1

[23] Shweta Mahajan, Apratim Bhattacharyya, Mario Fritz, Bernt Schiele, and Stefan Roth. Normalizing flows with multi-scale autoregressive priors. In *CVPR*, 2020. 40

[24] Seonghyeon Nam, Youngbae Hwang, Yasuyuki Matsushita, and Seon Joo Kim. A holistic approach to cross-channel image noise modeling and its application to image denoising. In *CVPR*, 2016. 1, 6, 24

[25] Naoya Ohta. A statistical approach to background subtraction for surveillance systems. In *ICCV*, 2001. 4

[26] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Neurips*, 2018. 40

[27] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021. 8, 9

[28] Tobias Plötz and Stefan Roth. Benchmarking denoising algorithms with real photographs. In *CVPR*, 2017. 5

[29] Paul L. Rosin. Thresholding for change detection. In *ICCV*, 1998. 4

[30] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *ICLR*, 2016. 9

[31] Zhou Wang, Alan Bovik, Hamid Sheikh, and Eero Simoncelli. Image quality assessment: from error visibility to structural similarity. *TIP*, 13(4):600–612, 2004. doi: 10.1109/TIP.2003.819861. 35

[32] Kaixuan Wei, Ying Fu, Jiaolong Yang, and Hua Huang. A physics-based noise formation model for extreme low-light raw denoising. In *CVPR*, 2020. 5

[33] Christopher R. Wren, Ali Azarbayejani, Trevor Darrell, and Alexander Pentland. Pfinder: real-time tracking of the human body. *TPAMI*, 19(7):780–785, 1997. 4

[34] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *TIP*, 26(7): 3142–3155, 2017. 1, 34

[35] Kai Zhang, Wangmeng Zuo, and Lei Zhang. Ffdnet: Toward a fast and flexible solution for cnn-based image denoising. *TIP*, 27(9):4608–4622, 2018. 1

[36] Yi Zhang, Hongwei Qin, Xiaogang Wang, and Hongsheng Li. Rethinking noise synthesis and modeling in raw denoising. In *ICCV*, 2021. 4

Appendix A

Inverse Gamma

As mentioned in Chapter 3, RAW-rgb images go through an in-camera imaging pipeline that transforms the image from the RAW-rgb space to the sRGB domain. The steps in this pipeline introduce non-linearities that result in a complex noise distribution in the sRGB space. One of the main nonlinear steps in this pipeline is gamma correction which is an invertible process. Its inverse is commonly used to approximately linearize sRGB data and can easily be implemented as a normalizing flow transformation. Inverse gamma is defined as $\mathbf{y} = \mathbf{x}^{\mathbf{gamma}}$ where the default value of $\mathbf{gamma}$ is 2.2. In this section, we explore the effects this layer has on Noise Flow and our proposed model when added to the data processing step.

Table A.1 shows that using the inverse gamma layer does not improve the modeling and sampling performances. Figure A.1 shows that $\mathbf{gamma}$ gamma converges to 1.2 from the initial value of 2.2 resulting in a near identity transformation for both models.

Model	NLL	D_{KL}	$\#Params$
Noise flow [2]	3.311	0.198	2330
Our model	**3.072**	**0.044**	6160
Inverse gamma + Noise flow	3.322	0.181	2332
Inverse gamma + Our model	3.092	0.061	6162

Table A.1: Models with the inverse gamma transformation added to their data processing step have similar performances as the original models.

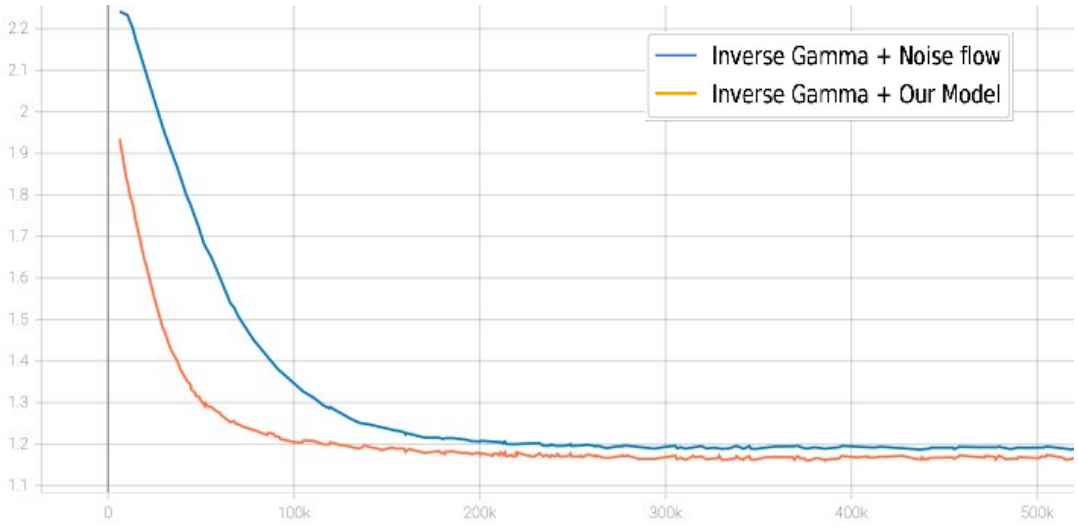


Figure A.1: **gamma** value changes in the training process. **gamma** converges to 1.2 for both models.