UNROLLING OF GRAPH TOTAL VARIATION FOR IMAGE DENOISING

HUY VU

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTERS OF APPLIED SCIENCE

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

APRIL 2021

Abstract

While deep learning (DL) architectures, like convolutional neural networks (CNNs), have enabled effective solutions in image denoising, in general their implementations overly rely on training data and require tuning of a large parameter set. In this thesis, a hybrid design that combines graph signal filtering with deep feature learning is proposed. It utilizes interpretable analytical low-pass graph filters and employs 80% fewer network parameters than a state-of-the-art DL denoising scheme called DnCNN. Specifically, to construct a suitable graph for graph spectral filtering, a CNN is used to learn feature representations per pixel first, then feature distances are computed to establish edge weights. Given a constructed graph, a convex optimization problem for denoising using a graph total variation (GTV) prior is formulated. Via a ℓ_1 graph Laplacian reformulation, its solution is interpreted in an iterative procedure as a graph low-pass filter with an analytical frequency response. For fast filter implementation, this response is realized by a Lanczos approximation. Experimental results show that in the case of statistical mistmatch, this method outperformed DnCNN by up to 3dB in PSNR.

Acknowledgements

First and foremost, I cannot express enough thanks to my supervisor, Professor Gene Cheung, Dept. of Electrical Engineering and Computer Science, York University. I am fortunate to have such a great, dedicated, and patient supervisor. He has given much time and effort to guide many aspects of my journey. Working with him has been a life-changing experience. Thanks to Prof. R. P. Wildes and Prof. J. DeSouza for serving as committee members and nominating my study for a thesis award.

I would like to express my deepest gratitude to my family. To my wife, vợ Loan, thank you for always supporting and staying by my side through the difficult time. Thank you for all of the countless sacrifices you have made to help me get to this point. To my daughter, bé Ái My (Ty), you are my greatest inspiration and motivation. Both of you have made me stronger, more responsible, and more reliable than I could have imagined. I am extremely grateful to my mom, mẹ Hiền. Without your unconditional love and support, none of this would be possible. Last but not least, many thanks to my brother and his family, anh Duy, chị Thảo, bé Thiên Phương (Mon). Your consistent encouragement is very much appreciated.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Graph Signal Processing (GSP) studies the processing of data on irregular kernels described by graphs [1, 2, 3]. Many modern data naturally reside on graph structures, such as transportation networks, online social networks, and networks of wireless temperature sensors. GSP has many potential applications and has received significant interest over the last few years.

Digital images are discrete signals on regular data kernels (2D grids) naturally but can nonetheless benefit from GSP. A neighborhood graph (*e.g.*, 4-/8-connected graph) can capture pairwise inter-pixel similarity or correlation of the two connected nodes, leading to improvements in many image processing tasks, such as image compression [4, 5, 6], image denoising [7, 8, 9] and image deblurring [10] *etc*.

Image denoising is a basic yet challenging problem in image processing, where the main goal is to recover the original image $\mathbf{x}$ given a noise-corrupted

observation $\mathbf{y} = \mathbf{x} + \mathbf{n}$, where $\mathbf{n}$ is an additive noise. Currently, modern image denoising methods can be loosely categorized into model-based and learning-based. In the last few years, thanks to the vast learning capabilities of Convolutional Neural Networks (CNN), learning-based approaches have achieved state-of-the-art performance in image denoising [11, 12, 13]. However, performance of these methods depends heavily on the training data, and often optimizing these methods requires tuning of a large set of network parameters, resulting in large memory requirements. This can be a significant impediment to practical implementation on platforms like mobile phones that require small memory footprints. Additionally, interpreting operations inside a deep neural network is challenging. Specifically, what the convolutional filters in a CNN-based architecture are actually doing remains hard to explain. Unexpected behaviors can take place when using learning-based algorithms as black boxes [14].

On the other hand, model-based approaches require prior assumptions on the characteristics of the sought signal $\mathbf{x}$ for regularization, which lead to more interpretable solutions. Model-based methods [15, 16, 17, 18] rely on signal priors like *total variation* (TV) [15, 18] and sparse representation [16] to regularize an inherently ill-posed problem. Graph-based prior assumptions are also possible. Popular signal priors in GSP include graph total variation (GTV) [10] and graph Laplacian regularizer (GLR) [8, 19]. The solutions of these graph-based approaches are often interpreted as graph low-pass filters of the noisy observations [8, 9, 19].

Since a digital image is a signal on a 2D grid, the key to good graph filtering performance for imaging is the selection of an appropriate underlying graph. The conventional choice for edge weight, $w_{i,j}$, between pixels (nodes) i and j in graph spectral image processing [8, 20, 10] is based on *bilateral filtering* [21]: an exponential kernel, $\exp(\cdot)$, of the negative inter-pixel distance, $\|\mathbf{l}_i - \mathbf{l}_j\|_2^2$, and pixel intensity difference, $(x_i - x_j)^2$, resulting in a non-negative edge weight $0 \leq w_{i,j} \leq 1$. This choice of edge weights means that graph connectivity is dependent on the signal, resulting in a *signal-dependent* signal prior like signal-dependent GLR (SDGLR), $\mathbf{x}^\top \mathbf{L}(\mathbf{x})\mathbf{x}$, which promotes *piecewise smoothness* (PWS) in the reconstructed signal [8, 20]. However, this choice is handcrafted and sub-optimal in general.

To learn a good similarity graph from data, *Deep GLR* (DGLR) [19] proposed a hybrid framework to combine graph-based with learning-based approaches. Specifically, the authors retained GLR's analytical filter response, $f(\lambda)$, but learned suitable feature vector, $\mathbf{f}_i$, per pixel i via a CNN in an end-to-end manner, leading to a robust denoising system that is less prone to overfitting than a pure CNN-based method. Note that the feature vector $\mathbf{f}_i$ is of low dimension, and the network parameters required are relatively few.

However, GLR has a noticeable drawback. The convergence of iterative GLR to reconstruct sharp object boundaries is slower than iterative GTV [10]. On the other hand, GTV is harder to optimize due to the non-differentiable ℓ_1-norm and does not have a natural spectral interpretation. In [10], the authors proposed a notion of frequency for GTV by approximating the GTV

using iterative GLR. Note that before this work, there was no frequency interpretation of GTV. However, there was no tangible benefit of the spectral interpretation of GTV.

This thesis proposes an image denoising algorithm using GTV as a signal prior and CNN for graph construction, similar to DGLR [19]. In addition, the runtime implementation is optimized for speed and memory via algorithm unrolling [22], where filtering in each unrolled algorithm iteration is approximated using Lanczos-based accelerated graph filter [23]—feasible thanks to the spectral interpretation of the GTV-based graph filter.

Compared to classical model-based methods like BM3D [17], the proposed algorithm has better denoising performance thanks to DL architecture's power in feature representation learning. Compared to pure DL-based DnCNN [11], it employs 80% fewer network parameters due to the deployment of analytical graph filters that are also interpretable. Compared to DGLR [19], this algorithm reconstructs more PWS images for the same network complexity. Moreover, in case of statistical mismatch between training and test data, DGTV outperformed DnCNN by up to 3dB in PSNR.

The thesis is structured as follows. Chapter 2 reviews necessary Graph Signal Processing and Deep Learning definitions and concepts and related works in the literature. The proposed image denoising algorithm is described in Chapter 3. Chapter 4 describes fast graph filter implementation methods for speeding up the algorithm. Experimental results and conclusion are presented in Chapter 5 and 6, respectively.

Chapter 2

Background

This chapter provides an introduction of important concepts in graph signal processing (GSP) and deep learning. In addition, literature overviews on image denoising and graph-based image restoration are provided.

2.1 Graph Signal Processing Concepts

Digital Signal Processing (DSP) studies discrete signals, where samples are equally spaced in time (*e.g.*, audio) or space (*e.g.*, 2D image). For example, in a speech or audio signal with N samples per second, the distance between two consecutive samples is $1/N$ seconds for any pair of consecutive samples. Similarly, for a digital image, two neighboring pixels have the same spatial distance due to regular spacings in the image-capturing Bayer-patterned grid. However, there are many practical scenarios where data resides in irregular

sampling kernels, *e.g.*, transportation networks, online social networks, and networks of wireless temperature sensors. A graph is a flexible abstraction that can elegantly describe these irregular kernels. GSP is the study of computational tools for signals residing on irregular kernels described by graphs [1, 2, 3].

2.1.1 Graph Definition

A graph is composed of sets of nodes and edges. An edge reflects pairwise relation between the two connected nodes. Conventionally, an edge weight represents pairwise similarity, correlation or feature distance.

Formally, a graph can be defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{W})$ with a set, $\mathcal{V}$, of nodes and a set, $\mathcal{E}$, of edges. The edges can be *weighted*, *i.e.*, a real value is associated with an edge, or *unweighted*, *i.e.*, all edges have the same weight. Considering a weighted graph, denote by $w_{i,j}$ the weight of an edge $(i, j) \in \mathcal{E}$ connecting nodes i and j. Edge weights in $\mathcal{G}$ form an *adjacency matrix* $\mathbf{W}$, where $W_{i,j} = w_{i,j}$ if $(i, j) \in \mathcal{E}$, and $W_{i,j} = 0$ if $(i, j) \notin \mathcal{E}$. For positive edge weights, larger $w_{i,j}$ indicates stronger similarity between nodes i and j.

An undirected graph is a graph with edges that have no associated directions, *i.e.*, $(i, j) \in \mathcal{E}$ means one can traverse from node i to node j and also from j to i. The associated adjacency matrix $\mathbf{W}$ of an undirected graph is symmetric. On the other hand, in a directed graph, each edge $(i, j) \in \mathcal{E}$ means only that one can traverse from node i to node j. The corresponding adjacency matrix $\mathbf{W}$ is not symmetric, in general.

In imaging, an N-pixel image (or image patch) can be interpreted as a graph signal as follows. Each pixel i of the image is represented by a node $i \in \mathcal{V}$. An edge $(i, j) \in \mathcal{E}$ connecting nodes i and j with weight $w_{i,j}$ represents the similarity between pixels i and j. There are multiple ways to compute these edge weights, which will be covered later in Section 2.1.3. The collection of N pixel values—a vector $\mathbf{x} \in \mathbb{R}^N$ of length N—is a graph signal on the constructed graph.

2.1.2 Graph Frequencies

Given an adjacency matrix $\mathbf{W}$ that represents an undirected graph $\mathcal{G}$, a diagonal *degree matrix* $\mathbf{D}$ is defined as

$$\mathbf{D}_{i,i} = \sum_j w_{i,j}. \tag{2.1}$$

A *combinatorial graph Laplacian matrix* $\mathbf{L}$ [2] is defined as

$$\mathbf{L} = \mathbf{D} - \mathbf{W}. \tag{2.2}$$

Assuming $w_{i,j} \geq 0, \forall (i, j) \in \mathcal{E}$, $\mathbf{L}$ is provably a *positive semi-definite* (PSD) matrix, *i.e.*, all eigenvalues of $\mathbf{L}$ are non-negative [3]. Given $\mathbf{L}$ is real and symmetric, one can eigen-decompose $\mathbf{L}$ into

$$\mathbf{L} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\top, \tag{2.3}$$

where columns of $\mathbf{V}$ are the eigenvectors, and $\mathbf{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_N)$ is a diagonal matrix with real non-negative eigenvalues (also known as graph frequencies) λ_k along its diagonal. $\mathbf{V}^\top$ is called the *graph Fourier transform* (GFT) [2] that transforms a graph signal $\mathbf{x}$ from the nodal domain to the graph frequency domain via $\boldsymbol{\alpha} = \mathbf{V}^\top \mathbf{x}$, similar to well-known classical discrete transforms such as *discrete cosine transform* (DCT). One can actually interpret GFT as a generalization of known transforms such as DCT-II [24]. Specifically, a 1D DCT is a GFT for a specific graph—a line graph with all weights equal to 1.

Other variation operators can also be eigen-decomposed to obtain a set of graph frequencies and frequency components. Popular variants of the Laplacian operator $\mathbf{L}$ include the following. A normalized graph Laplacian, $\mathbf{L}_n = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$, is a symmetric normalized version of $\mathbf{L}$. A random walk graph Laplacian, $\mathbf{L}_r = \mathbf{D}^{-1}\mathbf{L}$, is an asymmetric normalized version of $\mathbf{L}$. A generalized graph Laplacian, $\mathbf{L}_g = \mathbf{L} + \text{diag}(d_{i,i})$, is a graph Laplacian that has self-loops at node i. Each variant has its own unique spectral properties. For example, $\mathbf{L}_n$ and $\mathbf{L}_r$ have the same eigenvalues between $[0, 2]$ (they are similarity transforms of each other). $\mathbf{L}_n$ does not have the constant vector as an eigenvector like $\mathbf{L}$ although they are both symmetric.

2.1.3 Graph Construction

There are multiple ways to define edge weights of a graph $\mathcal{G}$. From a machine learning perspective, a suitable graph can be learned given multiple signal

observations. This is called *graph learning*—a fundamental problem in GSP—where a graph structure that best fits the observations is identified given a criterion or model assumptions [25, 26, 27, 28, 29]. *Graphical lasso* [25] focuses on learning a sparse inverse covariance matrix (precision matrix) assuming a *Gaussian Markov Random Field* (GMRF) model and a sparse graph.

Another popular way to define $\mathcal{G}$ is to construct a similarity graph. In a similarity graph, the edge weights represent pairwise similarities between nodes; *i.e.*, a larger weight $w_{i,j}$ indicates that samples at nodes i and j are more similar. There are different ways to defined $w_{i,j}$. A common method to define the edge weights in images is using the bilateral filter [21, 30]:

$$w_{i,j} = \exp\left(-\frac{\|\mathbf{l}_i - \mathbf{l}_j\|_2^2}{\sigma_l^2}\right) \exp\left(-\frac{\|x_i - x_j\|^2}{\sigma_x^2}\right), \tag{2.4}$$

where $\mathbf{l}_i$ is the location of pixel i on the 2D grid, σ_l and σ_x are two parameters.

Other works, *e.g.*, [8], use a simpler version of the above kernel; the weights $w_{i,j}$ are computed directly from the pixel intensities using a single Gaussian kernel, *i.e.*,

$$w_{i,j} = \exp\left(-\frac{\|x_i - x_j\|^2}{\sigma_x^2}\right). \tag{2.5}$$

In some works that combine GSP with deep learning like [9] and [19], the

edge weights are computed as

$$w_{i,j} = \exp\left(-\frac{\text{dist}(i,j)}{\epsilon^2}\right), \tag{2.6}$$

where $\text{dist}(i,j)$ represents the feature distance between pixels i and j and are computed from features learned by CNNs.

2.1.4 Graph Signal Priors

A signal prior is a mathematical description of a signal assumption. Signal priors are needed to regularize ill-posed (*i.e.*, under-determined) inverse problems, such as denoising.

Traditionally, for an undirected graph $\mathcal{G}$ with positive edge weights, signal $\mathbf{x}$ is considered smooth if it contains mostly low graph frequency components, *i.e.*, GFT coefficients $\alpha_k \approx 0$ for high frequency components k.

A popular graph signal prior for signal $\mathbf{x}$ is *graph Laplacian regularizer* (GLR) [8, 19]. It is a widely used graph signal prior to regularize ill-posed inverse problems like image denoising. GLR is defined as

$$\mathbf{x}^\top \mathbf{L}\mathbf{x} = \sum_{(i,j)\in\mathcal{E}} w_{i,j}(x_j - x_i)^2 = \sum_{k=1}^{N} \lambda_k \alpha_k^2, \tag{2.7}$$

where λ_k's are the eigenvalues of $\mathbf{L}$, and α_k's are the GFT coefficients of signal $\mathbf{x}$. A signal $\mathbf{x}$ is smooth if its GLR is small [31, 32, 8]. In the nodal domain, GLR of signal $\mathbf{x}$ is small when the nodes connected by large edge weights

have similar signal intensities. In the graph frequency domain, GLR of $\mathbf{x}$ is small when the energy of high graph frequencies is small. Thus, minimizing (2.7) means low-pass filtering. Since $\mathbf{L}$ is PSD [33], GLR is lower-bound by 0, $\forall \mathbf{x} \in \mathbb{R}^N$.

Another popular signal prior is *Graph Total Variation* (GTV) [10, 34], defined as

$$\|\mathbf{x}\|_{\text{GTV}} = \sum_{(i,j)\in\mathcal{E}} w_{i,j}|x_j - x_i|. \tag{2.8}$$

Unlike GLR, GTV does not have a straightforward frequency interpretation. Instead, [10] provided a frequency interpretation per iteration using a different definition of graph Laplacian. Minimizing GTV is different from minimizing GLR since GTV is a non-differentiable ℓ_1-norm. GTV can be minimized using Proximal Gradient Descent, a general algorithm to optimize non-smooth functions [34, 35]. GTV is also lower-bounded by 0 when $w_{i,j} \geq 0, \forall (i,j) \in \mathcal{E}$.

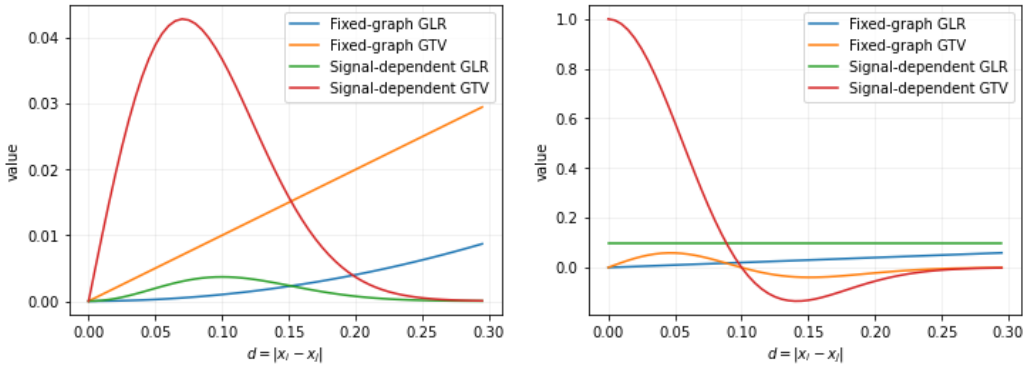

Figure 2.1: Regularizer values (left) and regularizer derivatives (right) as functions of one pair of inter-node sample difference $d = |x_i - x_j|$.

The selection of an underlying graph is important in GSP. One can group ways to define the edge weights between nodes i and j into two groups: signal-independent and signal-dependent. If the underlying graph structure and edge weights are selected independent of the signal, then it is signal-independent. In contrast, in a signal-dependent graph, the edge weights are computed dependent on the target signal. A conventional choice for computing signal-dependent edge weights is based on bilateral filtering [21]. Thus, the graph signal priors GTV and GLR can have two variants—signal-dependent GLR/GTV and signal-independent GLR/GTV—based on the selection of edge weights.

A brief comparison of GTV and GLR. In [10], the authors argued that both signal-dependent GLR and GTV can promote *piecewise smoothness* (PWS) in signal reconstruction, but signal-dependent GTV can promote PWS faster than signal-dependent GLR. For simplicity, consider only one term in a regularizer, for example, $w_{i,j}(x_i - x_j)^2$ in GLR. Figure 2.1(left) shows regularizer values of this one term as functions of the inter-node difference $d = |x_i - x_j|$. When $d \to 0$, clearly all regularizers also approach 0. Moreover, for signal-dependent GLR and GTV, when $d \to \infty$, signal-dependent edge weight $w_{i,j} \to 0$ due to dependency such as (2.5), and thus the regularizers also approach 0. We can make the following general observations from this plot. First, signal-independent GLR and GTV are convex functions, while signal-dependent GLR and GTV are non-convex. Second, to minimize signal-dependent GLR and GTV, inter-sample difference $d = |x_i - x_j|$ must be either

very small or very large. This explains why signal-dependent GLR and GTV promotes PWS in signal reconstruction.

Consider now the regularizer derivatives as functions of d in Figure 2.1(right). In signal-independent GLR and GTV, the derivatives are linear and constant, respectively, for fixed edge weight $w_{i,j}$. For signal-dependent GLR and GTV, we see that as $d \to 0$, the derivative of signal-dependent GLR approaches 0, while the derivative of signal-dependent GTV approaches 1. This means that signal-dependent GTV promotes PWS *faster* than signal-dependent GLR as $d \to 0$. A more detailed comparison of GLR and GTV can be found in [10].

As previously discussed, note that there was no frequency interpretation of GTV before [10].

2.2 Deep Learning Concepts

2.2.1 Neural Networks

Neural networks are machine learning algorithms that are inspired by human brains. In imaging, two popular neural network types are Fully-connected Neural Networks and Convolutional Neural Networks [36].

Fully-connected Neural Networks

This is the simplest type of neural network in terms of architecture design. Figure 2.2 shows an example architecture of a fully-connected neural network. It is composed of layers of connected nodes, where each node is called a neuron. The output of a single neuron is computed as $z = f(\mathbf{W}\mathbf{x} + b)$, where

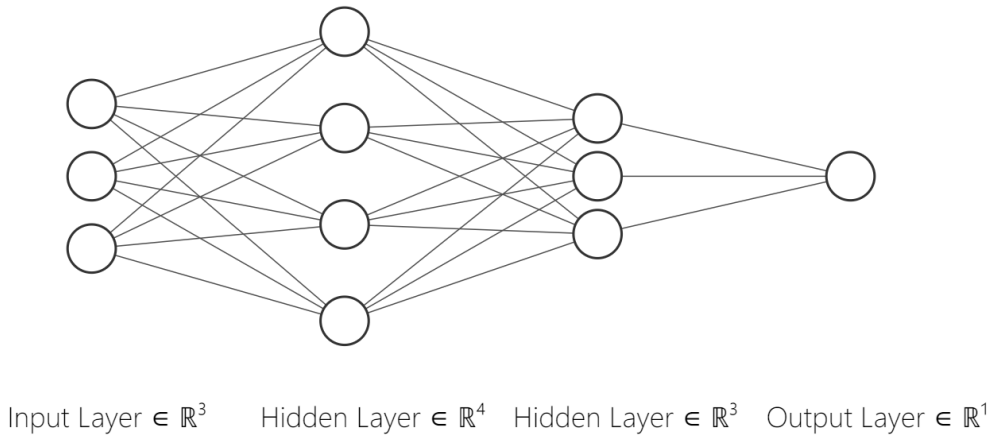

Figure 2.2: A simple fully-connected neural network architecture with two hidden layers.

z is the output of the neuron, $\mathbf{x}$ is the input (the input can be outputs of the neurons in the previous layer), $f(\cdot)$ is a non-linear activation function, $\mathbf{W}$ is a weight vector, and b is a bias term. $\mathbf{W}$ and b are parameters of the network and are trained via learning. A fully-connected neural network means that all nodes in adjacent layers are connected to each other.

Convolutional Neural Networks

A Convolutional Neural Networks (CNN) are a form of neural networks that employ a basic operation in signal processing—convolution. CNNs are used extensively in computer vision and image processing problems with great success. Formal definitions of CNN can be found in [36]. An example of a simple CNN is shown in Figure 2.3. It has convolutional layers, pooling layers and fully-connected layers.

Convolutional Layer. Convolutional layer is the main layer type in

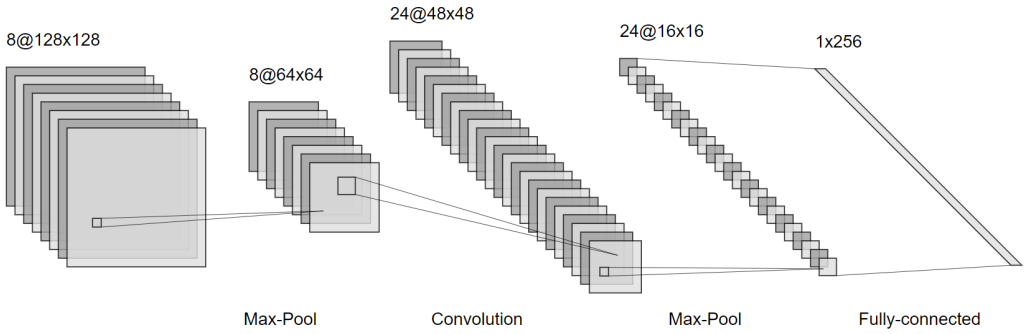

Figure 2.3: A simple CNN architecture.

a CNN. This layer performs a convolution operation on an input, followed by a non-linear activation function. The filter coefficients that are used for convolution in this layer are learned via data training. Convolutional layer's hyperparameters include the filter size and the stride—the amount we slide the filter on the input at a time.

Point-wise Activation Functions. Point-wise activation functions add non-linearity to the network. The activation function is attached to each node of the network and defines output of the node. Some of the popular activation functions are Sigmoid, Tanh and Rectifier Linear Unit (ReLU) [37].

Sigmoid. The sigmoid function is defined as $f(x) = \frac{1}{1+e^{-x}}$. This function is useful when one needs to predict the probability as an output, since its value is in the range $[0, 1]$. The function is differentiable.

Tanh. The Tanh function is similar to the sigmoid function but the outputs are in the range $[-1, 1]$. It is defined as $f(x) = \frac{e^{x}-e^{-1}}{e^{x}+e^{-1}}$.

ReLU. Currently, this is one of the most used activation functions in the

literature. Its formal definition is $f(x) = \max(0, x)$. In other words, $f(x)$ is zero for $x < 0$ and $f(x)$ is x for $x \geq 0$.

Pooling Layer. Typically, a pooling layer performs a downsampling operation on the output of a convolutional layer. Average pooling and max pooling are popular types of pooling, where the average and the maximum value in a window is computed as output, respectively.

Fully-connected layer. This is similar to the fully-connected neural networks. This layer is not always presented in a CNN.

LeNet [38], AlexNet [39], VGG [40], ResNet [41] are a few popular successful CNN architectures. For detailed discussion of these various architectures, see [42].

2.2.2 Loss Functions

A loss function is used to drive the learning of the network via minimization of the loss with respect to the free parameters of the system, *e.g.*, individual numerical values of the convolutional kernels. In image denoising, a popular loss function is the mean square error (MSE). It is defined as follows:

$$\text{MSE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2, \tag{2.9}$$

where N is the number of samples, $\mathbf{y}$ and $\hat{\mathbf{y}}$ are the true output and the predicted output of a model, respectively. MSE measures the average squared difference between the expected and the predicted outputs.

2.2.3 Gradient Descent

Gradient descent is a family of optimization algorithms. It is widely used to minimize a chosen loss function by optimizing the choice of a parameter set during data training. Using gradient information of a defined loss function, gradient descent minimizes the loss function by iteratively moving in the direction of steepest descent. Some well-known variations of gradient descent are as follows.

Batch Gradient Descent. This is the most well-known variation. One first calculates the gradient of the loss function to find the steepest descent direction, then moves a small step toward that direction, and repeat. The step size is controlled by a parameter called the learning rate. One must set the step size carefully; if the step size is too large, the optimization algorithm may not converge. On the other hand, if the step size is too small, it may take a long time until convergence to a locally optimal point. In Batch Gradient Descent (BGD), in each iteration one uses all training data in its calculations, resulting in slow execution speed for large datasets.

Stochastic Gradient Descent. Instead of using all training data each step, Stochastic Gradient Descent (SGD) computes its gradient using only one data sample. Thus, SGD updates more frequently than BGD. Another difference between BGD and SGD is that SGD updates have higher variance. This makes the loss function fluctuate heavily. On the one hand, the fluctuation may enable SGD to jump over local minima if it gets stuck. On the other hand, it also makes the convergence to the exact minimum more

difficult since SGD may overshoot the minimum.

Mini-batch Gradient Descent. This variation is in between BGD and SGD; *i.e.*, it uses a fixed number of samples to compute a gradient.

Momentum. SGD usually oscillates in areas where the loss surface curves much more steeply in one dimension than in others [43]. Momentum is a method that helps accelerate SGD in the relevant direction and reduces the oscillations. It does this by adding a fraction of the update vector of the past time step to the current update vector. The momentum increases for dimensions whose gradients point in the same directions and decreases for dimensions whose gradients change directions. Thus, Momentum leads to faster convergence and reduce oscillation.

Skip-connections. A skip connection connects components of different layers directly. The direct connection helps reduce the vanishing gradient problem during learning via back propagation. Thus, skip connections can enable learning of very deep neural network architectures, *e.g.*, [41].

2.2.4 Regularization

Making an algorithm perform well on both training and test data is a main problem in machine learning. Regularization is a strategy that helps address the problem. Regularization is a mathematical technique for solution to ill-posed problems and prevent overfitting during model estimation [44]. It accomplishes these goals by introducing additional constraints on the problem at hand. For machine learning, models that overfit the training data are

unlikely to generalize. Popular regularization techniques in machine learning include L1 regularization, L2 regularization, and more recently, Dropout [45].

L1 regularization and L2 regularization. A regularization term is added to the loss function. In L1 regularization, an ℓ_1-norm term of the parameter vector is added, and in L2 regularization, an ℓ_2-norm term of the parameter vector is added. These two methods force the optimization algorithm to choose a parameter set that does not have a large ℓ_1/ℓ_2-norm, leading to less complex models.

Dropout. Dropout is a computationally inexpensive yet effective neural network regularization technique. While training, each neuron has a probability $1 - p$ to be set to zero. One can view dropout as a method that approximates training a large number of neural networks with different architectures in parallel. This makes the training process noisy. Therefore, each hidden node in neural networks must be able to perform well since other hidden nodes can be swapped out probabilistically. Dropout reduces the effective capacity of a model. The size of the model usually is increased to offset this effect. Dropout is less effective when extremely few labeled training examples are available [45].

2.2.5 Algorithm Unrolling

One of the biggest drawbacks of deep neural networks is the lack of interpretability. Specifically, what the convolutional filters in a CNN-based architecture are actually doing remains hard to explain. An emerging tech-

nique called algorithm unrolling (or unfolding) [22] helps address this issue.

In general, given an iterative algorithm (*e.g.*, Iterative Shrinkage and Thresholding Algorithm [46] (ISTA), Coordinate Descent method [47]), one can represent each iteration of the algorithm as one layer of a neural network. By concatenating these layers, an architecture similar to deep neural network is formed. Executing the original iterative algorithm for a finite number of iterations is equivalent to forward-passing through the network. The iterative algorithm parameters (*e.g.*, regularizer trading-off parameter) translate to parameters in the neural network, which can be trained from data via back-propagation. Thus, with algorithm unrolling, the trained network is naturally interpretable as a parametric algorithm, resulting in interpretability in conventional deep neural networks. Each layer can now be understood as an algorithm iteration.

Learned ISTA (LISTA) [48] is the earliest work on algorithm unrolling. It unrolls the iterations of ISTA into a deep network and learns parameters of ISTA end-to-end. ISTA is one of the most common method to solve a sparse coding problem [22]. ISTA parameters include a weight parameter and an over-complete dictionary. Each iteration of ISTA composes of one linear operation followed by a non-linear soft-thresholding operation. Thus, mapping each iteration to a network layer and stacking the layers can form a deep network. Forward-passing this network is equivalent to executing ISTA iterations multiple times.

The original motivation of LISTA was to improve the computational

efficiency of ISTA through end-to-end training. However, it also led to benefits of parameters savings and interpretability. Classical iterative algorithms often had significantly fewer number of parameters, compared with deep neural network approaches. Hence, the unrolled networks are efficient in terms of number of parameters. Furthermore, classical iterative algorithms were built analytically, resulting in more understandable algorithms in general.

2.3 Image Denoising

The literature in image denoising is vast, since the basic problem is fundamental and has been studied for decades [49]. Modern image denoising methods can be roughly categorized into model-based and learning-based.

2.3.1 Model-based Image Denoising

Model-based methods [15, 50, 16, 17, 18, 51, 52, 53] rely on assumptions on signal characteristics to regularize an inherently ill-posed problem. Notable methods include the following. Non-linear filters [54] were used to preserve edge information and suppress the noise. In [16], the authors leveraged sparse and redundant representations over trained dictionaries to describe the image content effectively, leading to a simple yet effective denoising algorithm. Non-local methods, like *non-local mean* (NLM) [51], took global information of an image into consideration instead of a group of pixels surrounding a target pixel in local methods. For instance, NLM took weighted average of

all pixels in an image, the weights are the similarities between pixels and the target pixel. Signal priors like *total variation* (TV) simultaneously smoothed away noise in flat regions while preserving edges via minimization of signal variations in the ℓ_1-norm [15, 18, 50, 55].

Other competitive model-based denoising methods include the *shrinkage fields* [52]—its regularization is provided through a Markov random field model, [56] proposed simultaneous sparse coding as a framework to combine sparse signal representations and non-local information. In [53], the authors proposed Weighted Nuclear Norm Minimization (WNNM) to enforce a low-rank prior and adopted it to image denoising.

Although these methods achieved reasonably good results in image denoising, they suffered from several drawbacks [57]. For example, parameters are set manually, and the assumed priors are too simplistic, all of which lead to sub-par performance in real-world scenarios.

2.3.2 Deep Learning-based Image Denoising

In contrast, learning-based methods leverage powerful learning abilities of recent deep learning (DL) architectures such as *convolutional neural networks* (CNN) to compute mappings directly from $\mathbf{y}$ to $\mathbf{x}$, given a large training dataset [11, 12, 13, 58, 59] to overcome the drawbacks of traditional model-based methods [57].

Despite their powerful denoising capabilities, these methods depend heavily on the training data and perform poorly when the statistics between the

testing and training data differ noticeably [19]. Moreover, these methods usually require tuning of a large set of network parameters, leading to large memory footprints. Large memory requirement is a significant impediment to practical implementation on platforms like mobile devices that have limited memory.

2.4 Graph-based Image Restoration

To reduce network parameters, we turn to graph-based techniques. We review graph-based image restoration works in this section. Image restoration is an inverse problem. In image restoration, one seeks to recover the original signal, $\mathbf{x}$, given a noise corrupted and/or degraded (partial) observation, $\mathbf{y}$. Image denoising, interpolation, deblurring, super-resolution are example tasks of image restoration. A widely used linear image formation model is

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{z}, \tag{2.10}$$

where $\mathbf{H}$ is a degradation matrix (*e.g.*, down-sampling, blurring), and $\mathbf{z}$ is an additive noise.

2.4.1 Graph-based Image Denoising

In image denoising, $\mathbf{H} = \mathbf{I}$, and $\mathbf{z}$ is typically assumed to be Gaussian distributed. Image denoising is the most basic image restoration problem.

Graph Laplacian Regularizer. A GLR assumes that the original image $\mathbf{x}$ is smooth with respect to a defined graph $\mathcal{G}$, *i.e.*, $\mathbf{x}$ should have a small value $\mathbf{x}^\top \mathbf{L}\mathbf{x}$, where $\mathbf{L}$ is the graph Laplacian matrix of graph $\mathcal{G}$. In [60], the authors removed impulse noise by applying multi-scale GLR. [61] employed GLR for joint denoising and super-resolution. Generally, to denoise a noisy image or image patch $\mathbf{y}$, one can formulate the following unconstrained quadratic programming (QP) problem using GLR:

$$\mathbf{x}^* = \min_{\mathbf{x}} \|\mathbf{y} - \mathbf{x}\|_2^2 + \mu \mathbf{x}^\top \mathbf{L}\mathbf{x}, \tag{2.11}$$

where μ is a trade-off parameter. For a fixed $\mathbf{L}$, this problem has a closed-form solution $\mathbf{x}^* = (\mathbf{I} + \mu\mathbf{L})^{-1}\mathbf{y}$.

In OGLR [8], the authors derived "optimal" features from a continuous signal model, using which an underlying graph $\mathcal{G}$ is constructed and used for solving the problem (2.11).

Instead of manually chosen features, a subsequent work called Deep GLR [19] (DGLR) learned appropriate features from data via a CNN to define a notion of feature distance and construct $\mathcal{G}$ with edge weights computed from those distances. Specifically, denote by $\mathbf{f}_i \in \mathbb{R}^K$ the feature vector of pixel i learned by a CNN. Then the edge weights $w_{i,j}$'s of a positive undirected graph are computed as

$$w_{i,j} = \exp\left(-\frac{\text{dist}(i,j)}{\epsilon^2}\right), \tag{2.12}$$

where $\text{dist}(i,j)$ is the Euclidean distance between pixels i and j in the K-dimension feature space, *i.e.*, $\text{dist}(i,j) = \sum_{k=1}^{K} \left(\mathbf{f}_i^k - \mathbf{f}_j^k\right)^2$. The problem (2.11) is then solved using the corresponding graph Laplacian $\mathbf{L}$ computed from the constructed graph.

Note that DGLR [19] was the first in the literature to combine graph-based regularization with deep neural networks (DNNs) into a hybrid image denoising scheme. While DGLR [19] requires solving a system of linear equations to denoise an image patch, Deep Analytical Graph Filter [9] (DAGF) employed *GraphBio* [62] —a biorthogonal critically sampled graph wavelet filter—as the analytical graph filter for image denoising. Specifically, DAGF constructed an underlying graph for each target image patch using (2.12), similarly done in DGLR. However, given a constructed graph, instead of solving an unconstrained QP programming problem to reconstruct a patch, DAGF filtered the observed patch using GraphBio—a critically sampled biorthogonal graph wavelet that allowed perfect reconstruction during the synthesis phase.

Recently, graph-convolutional layers were used in [63] to create layers that exploit non-local self-similarity to denoise images. This method also suffered from the lack of interpretability like traditional deep learning methods, *i.e.* what the trained network actually filters out remains unclear. Other recent works in algorithm unrolling [22] include also graph-signal denoising [64]. However, the goal in [64] is to learn the most suitable prior for signal denoising on a fixed given graph, while DGTV focuses on image denoising,

where learning a good underlying graph is a key challenge.

2.4.2 Graph-based Image Deblurring

Image deblurring is more challenging than image denoising [3] since the blurring operator $\mathbf{H}$ in (2.10) may not be known. For example, in [10], Graph Total Variation—a well-known graph signal prior—was used for blind image deblurring, where the blur kernel was not known. The deblurring problem in [10] was formulated as

$$(\hat{\mathbf{x}}, \hat{\mathbf{k}}) = \arg\min_{\mathbf{x},\mathbf{k}} \frac{1}{2}\|\mathbf{x} \otimes \mathbf{k} - \mathbf{b}\| + \beta\|\mathbf{x}\|_{\mathrm{RGTV}} + \mu\|\mathbf{k}\|_2^2, \tag{2.13}$$

where $\mathbf{k}$, $\mathbf{x}$ and $\mathbf{b}$ are respectively the blur kernel, the original signal and the observed blurred signal, β and μ are weight parameters, and $\otimes$ is the convolution operator. Different from the widely used formulation (2.11) of image denoising, the formulation for image deblurring in [10] had another regularization term for blur kernel $\mathbf{k}$. The unknown blur kernel made the problem more challenging. To find the solution of the optimization problem (2.13), the authors alternately fixed either $\mathbf{x}$ or $\mathbf{k}$ and minimized the other term.

Some other well-known graph based image deblurring methods are listed as follows. In [65], the authors proposed a deconvolution algorithm based on the regularized Stein's unbiased risk estimate (SURE), which is a good estimate of the mean squared error. Kernel regression was extended for deblurring

applications [66]. Other graph-based deblurring methods include [67, 68]. In [67], a new cost function, which consists of a new fidelity term and a normalized graph Laplacian regularization term was proposed. The proposed cost function also allowed spectral analysis of their solutions. Inspired by the multi-Wiener SURE-LET deconvolution, [68] proposed a deblurring algorithm for point cloud attributes.

2.4.3 Other Graph-based Image Restorations

A few notable works using graph-based restoration techniques are briefly discussed in this section. [69] proposed a joint demosaicking / rectification framework that composed a 360-degree image from viewpoint images captured multiple fisheye cameras, using GLR for interpolating pixels. The optimization problem in [69] was posed as:

$$\min_{\mathbf{x}} \|\mathbf{H}\mathbf{y} - \mathbf{x}\|_2^2 + \mu \mathbf{x}^\top \mathbf{L}_\mathbf{x} \mathbf{x}, \tag{2.14}$$

where $\mathbf{H}$ is an interpolation matrix and $\mathbf{y}$ is the corrupted observation $\mathbf{y}$. The problem (2.14) can be interpreted as follows. The reconstructed signal $\mathbf{x}$ should be similar to the interpolation $\mathbf{H}\mathbf{y}$ and smooth with respect to a graph specified by $\mathbf{L}_\mathbf{x}$ [69].

[70] also used GLR for image demosaicking, where edge weights were computed from interpolated color images. In [61], the authors formulated the

joint denoising and super-resolution problem as:

$$\min_{\mathbf{x}} \|\mathbf{DHx} - \mathbf{y}\|_2^2 + \lambda \mathbf{x}^\top \mathbf{L}_h \mathbf{x}, \tag{2.15}$$

where $\mathbf{D}$ is a down-sampling matrix, $\mathbf{H}$ is a Gaussian low-pass filtering operator, and $\mathbf{L}_h$ is a suitably derived graph Laplacian matrix. Specifically, $\mathbf{L}_h$ was computed from a high resolution graph, which was estimated from a low resolution dual graph. In this case, $\mathbf{H}$ in (2.10) is a combination of two operators, a down-sampling and a low-pass filter.

Chapter 3

Deep Graph Total Variation for Image Denoising

In this chapter, the proposed algorithm, Deep Graph Total Variation (DGTV), is described in detail. DGTV is an image denoising method that combines classical graph signal filtering with convolutional neural networks feature learning. First, a CNN is used to learn feature representation of an image. Then, edge weights are computed from feature distances, from which an 8-connected graph is constructed. Finally, to denoise an image, a convex optimization problem is formulated and solved with a closed-form solution. The optimization problem has a weight parameter that is also learned by a different CNN.

In Section 3.1, an overview of the DGTV architecture is given. Section 3.2 formulates an image denoising problem using GTV, and Section 3.3 describes

how DGTV learns the features from data.

3.1 Architecture Overview

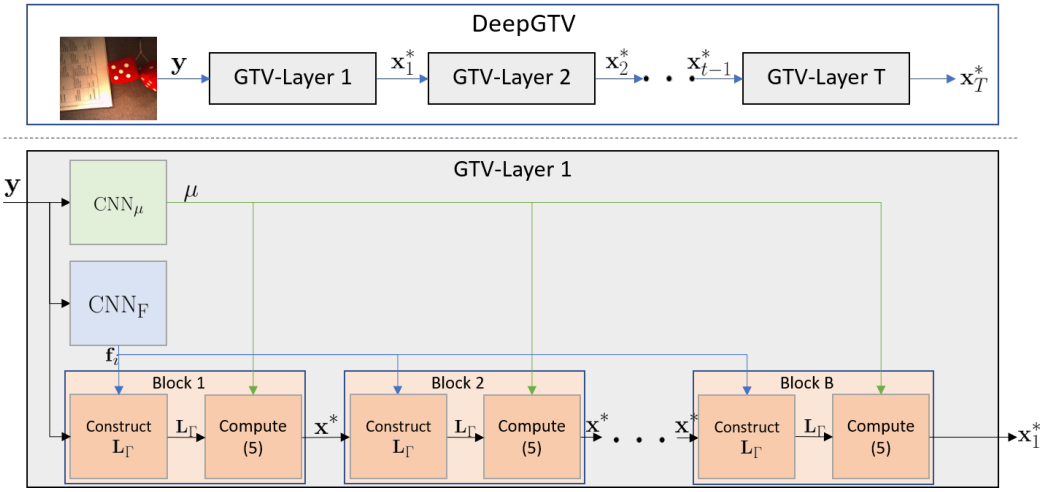

Figure 3.1: Overview of the proposed architecture. Top: Deep GTV composed of multiple GTV-Layers; Bottom: architecture of a GTV-Layer.

The proposed algorithm is called Deep Graph Total Variation (DGTV). It is a combination of convolutional neural networks and traditional graph filtering for image denoising. The network architecture, as shown in Figure 3.1, is composed of T *layers*, each implementing an algorithm iteration. Hence, it can be interpreted as an implementation of algorithm unrolling [22]. Particularly, T layers filter an input image or image patch T times.

Each layer is implemented as a cascade of B *blocks*. At runtime, each block takes as input a noisy input image patch $\mathbf{y}$ and computes a denoised patch $\mathbf{x}$ of the same dimension via graph low-pass filtering. To assist in the

denoising process, in each layer two CNNs are used to compute feature vector $\mathbf{f}_i$ per pixel and a weight parameter μ. Parameters of the CNNs are trained end-to-end offline using training data via stochastic gradient descent [71].

The next section describes the unrolling of a chosen denoising algorithm into layers, then the two employed CNNs.

3.2 Denoising with Graph Total Variation

Consider a linear image formation model for an image patch corrupted by an additive noise:

$$\mathbf{y} = \mathbf{x} + \mathbf{n}, \tag{3.1}$$

where $\mathbf{y} \in \mathbb{R}^N$ is the corrupted observation, $\mathbf{x} \in \mathbb{R}^N$ is the original image patch, and $\mathbf{n} \in \mathbb{R}^N$ is a noise term. The goal of denoising is to estimate $\mathbf{x}$ given only $\mathbf{y}$.

Given a graph $\mathcal{G}$ with edge weights $w_{i,j}$ (to be discussed), an optimization problem using GTV as a signal prior is formulated [10] to reconstruct $\mathbf{x}$ given noisy observation $\mathbf{y}$ as

$$\min_{\mathbf{x}} \|\mathbf{y} - \mathbf{x}\|_2^2 + \mu \sum_{(i,j)\in\mathcal{E}} w_{i,j}|x_i - x_j|, \tag{3.2}$$

where $\mu > 0$ is a parameter trading off the fidelity term and the prior. The optimization problem given in (3.2) is convex and can be solved using iterative

methods, *e.g.*, *proximal gradient* (PG) [34, 35], but does not have a closed-form solution. Moreover, there is no spectral interpretation of the obtained solution.

Instead, as done in [10], the GTV ℓ_1-norm term in (3.2) is transformed into a quadratic term as follows. First, given a signal estimate $\mathbf{x}^o$, a new adjacency matrix $\mathbf{\Gamma}$ with edge weights $\Gamma_{i,j}$ is defined as

$$\Gamma_{i,j} = \frac{w_{i,j}}{\max\{|x_i^o - x_j^o|, \rho\}}, \tag{3.3}$$

where $\rho > 0$ is a chosen parameter to circumvent numerical instability when $|x_i^o - x_j^o| \approx 0$. Assuming $|x_i^o - x_j^o|$ is sufficiently large (*i.e.*, $|x_i^o - x_j^o| > \rho$) and estimate $\mathbf{x}^o$ sufficiently close to signal $\mathbf{x}$, one can see that

$$\Gamma_{i,j}\,(x_i - x_j)^2 = \frac{w_{i,j}}{|x_i^o - x_j^o|}\,(x_i - x_j)^2 \approx w_{i,j}\,|x_i - x_j|\,.$$

This means that using $\mathbf{\Gamma}$, GTV can be expressed in a quadratic form. Specifically, an ℓ_1-*Laplacian matrix* $\mathbf{L}_\Gamma$ can be defined as

$$\mathbf{L}_\Gamma = \mathrm{diag}(\mathbf{\Gamma 1}) - \mathbf{\Gamma}, \tag{3.4}$$

where $\mathbf{1} \in \mathbb{R}^N$ is a length-N vector of all one's. Then, (3.2) can be reformulated using a quadratic regularization term given $\mathbf{L}_\Gamma$; *i.e.*,

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \|\mathbf{y} - \mathbf{x}\|_2^2 + \mu\ \mathbf{x}^\top \mathbf{L}_\Gamma \mathbf{x}. \tag{3.5}$$

This new optimization problem (3.5) no longer has a non-smooth objective. In particular, objective in (3.5) contains two quadratic terms. One can solve it via different methods. A direct way to solve (3.5) is to take the partial derivative with respect to $\mathbf{x}$ to find the solution. The partial derivative is

$$\frac{\partial}{\partial \mathbf{x}} = -2\mathbf{y} + 2\mathbf{x} + 2\mu \mathbf{L}_\gamma \mathbf{x}. \tag{3.6}$$

The convex function is minimized when its derivative is zero. Thus, setting (3.6) to 0 and solving for $\mathbf{x}$ yields

$$\mathbf{x}^* = (\mathbf{I} + \mu\ \mathbf{L}_\Gamma)^{-1}\mathbf{y}. \tag{3.7}$$

Hence, for given estimate $\mathbf{x}^o$ and subsequent Laplacian $\mathbf{L}_\Gamma$, (3.5) has a closed-form solution. Note that (3.5) must be solved multiple times, where in each iteration $b+1$, the solution $\mathbf{x}_b^*$ from the previous iteration b is used to update edge weights (3.3) in $\mathbf{\Gamma}$. In this architecture, each block solves (3.5) once, and a layer composing of B blocks solves (3.2). Cascading T layers forms the DGTV architecture, shown in Figure. 3.1.

Computing the solution (3.7) is computationally expensive, since it requires a matrix inversion operation. Instead, one can write an equivalent system of

linear equations:

$$(\mathbf{I} + \mu\ \mathbf{L}_\Gamma)\mathbf{x}^* = \mathbf{y}, \tag{3.8}$$

and solve for $\mathbf{x}^*$ efficiently without matrix inversion using *conjugate gradient* (CG). CG requires the coefficient matrix $\mathbf{I} + \mu\mathbf{L}_\Gamma$ to be positive definite (PD), sparse and symmetric. Clearly, $\mathbf{I} + \mu\mathbf{L}_\Gamma$ is symmetric and sparse, since graph Laplacian $\mathbf{L}_\Gamma$ is symmmetric and sparse. One can prove it is also PD as follows. We know that the identity matrix $\mathbf{I}$ is PD, *i.e.* $\mathbf{x}^\top\mathbf{I}\mathbf{x} > 0, \forall\mathbf{x} \neq \mathbf{0}$. Further, the combinatorial Laplacian matrix $\mathbf{L}_\Gamma$ is PSD, *i.e.* $\mathbf{x}^\top\mathbf{L}_\Gamma\mathbf{x} \geq 0, \forall\mathbf{x}$. Thus, $\forall\mathbf{x} \neq \mathbf{0}$,

$$\begin{aligned}\mathbf{x}^\top(\mathbf{I} + \mu\mathbf{L}_\Gamma)\mathbf{x} &= \mathbf{x}^\top\mathbf{I}\mathbf{x} + \mathbf{x}^\top\mathbf{L}_\Gamma\mathbf{x} \\ &\geq \mathbf{x}^\top\mathbf{x} > 0.\end{aligned} \tag{3.9}$$

Thus, $\mathbf{I} + \mu\mathbf{L}_\Gamma$ is PD.

Although CG can solve the system of linear equations in (3.8) without matrix inversion, it can still be slow. However, with frequency interpretation of GTV [10], it is possible to achieve faster solution using accelerated graph filter. Chapter 4 will describe these fast implementations in detail.

3.3 Feature and Weight Parameter Learning with Convolutional Neural Networks

In (3.2), a graph $\mathcal{G}$ with edge weights $w_{i,j}$ is assumed. As done in [19, 9], in each layer, a $\text{CNN}_{\mathbf{F}}$ is used to compute an appropriate feature vector $\mathbf{f}_i \in \mathbb{R}^K$ at runtime for each pixel i in an N-pixel patch, using which edge weights are computed to construct a graph $\mathcal{G}$. Specifically, given feature vectors $\mathbf{f}_i$ and $\mathbf{f}_j$ of pixels (nodes) i and j, a non-negative edge weight $w_{i,j}$ between them is computed using a Gaussian kernel, *i.e.*,

$$w_{i,j} = \exp\left(-\frac{\sum_{k=1}^{K}\left(\mathbf{f}_i^k - \mathbf{f}_j^k\right)^2}{\epsilon^2}\right), \tag{3.10}$$

where $\epsilon > 0$ is a parameter. To learn $\text{CNN}_{\mathbf{F}}$ end-to-end, one can compute the partial derivative of the *mean square error* (MSE) between the recovered patch $\mathbf{x}^*$ and the ground-truth patch $\mathbf{x}$ with respect to $\mathbf{L}_\Gamma$. This is feasible since $\mathbf{f}_i$ is used to construct $\mathbf{L}_\Gamma$ and $\mathbf{L}_\Gamma$ appears in (3.7). MSE is defined as

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(x_i^* - x_i)^2. \tag{3.11}$$

Denote by $\delta_i \in \mathbb{R}^N$ the indication vector whose i-th entry is 1 and the

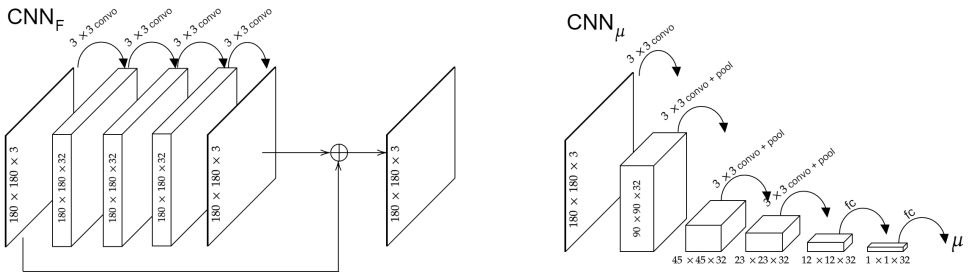

Figure 3.2: Network architecture. Left: architecture of CNN_F. Right: architecture of CNN_μ.

rest are zeros, then the partial derivative of $\mathcal{L}_{\text{MSE}}$ with respect to $\mathbf{L}_\Gamma$ is

$$\begin{aligned}\frac{\partial \mathcal{L}_{\text{MSE}}}{\partial \mathbf{L}_\Gamma} &= \sum_{n=1}^{N} \frac{\partial \mathcal{L}_{\text{MSE}}}{\partial x_i} \cdot \frac{\partial x_i}{\partial \mathbf{L}_\Gamma} \\ &= -\sum_{n=1}^{N} \mu \left(x_i^* - x_i\right) \cdot \left(\mathbf{I} + \mu \mathbf{L}_\Gamma\right)^{-\top} \delta_i \mathbf{y}^\top \left(\mathbf{I} + \mathbf{u}\mathbf{L}_\Gamma\right)^{-\top}, \end{aligned} \tag{3.12}$$

where $A^{-\top} = (A^{-1})^\top$. This partial derivative is back-propagated to update the parameters in $\text{CNN}_\mathbf{F}$. The computed edge weights $w_{i,j}$ compose the adjacency matrix $\mathbf{W}$ of an image patch.

In each layer, another CNN—denoted by CNN_μ— is used to compute weight parameter μ that trades off the fidelity term and the prior in (3.2), for a given noisy patch $\mathbf{y}$. Similar to $\text{CNN}_\mathbf{F}$, μ appears in (3.7), and thus one can learn CNN_μ end-to-end via back-propagation. Figure 3.2 shows the specific architecture of the described CNNs.

Each GTV-Layer in DGTV filters an image patch by solving (3.2), and hence all CNNs in DGTV can be learned in an end-to-end manner by super-

vising only the error between the final restored patch $\mathbf{x}_T^*$ and the ground-truth patch.

Chapter 4

Implementing The Graph Filter

Computing the solution (3.7) requires matrix inversion, which is computationally expensive. Computing the solution in the system of linear equations in (3.8) via conjugate gradient (CG) is faster, but it can still be slow if the dimension of the sought signal $\mathbf{x}^*$ is large. Faster execution can be achieved from the graph spectral filtering perspective. This chapter describes how one can circumvent matrix inversion in (3.7) via different approximation filter methods in the graph frequency domain. In particular, Chebyshev polynomials approximation [72, 73] and Lanczos method [23] are investigated.

4.1 Frequency interpretation of the solution

One can interpret (3.7) as a low pass graph spectral filter by eigen-decomposing $\mathbf{L}_\Gamma = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top$, with frequency response $f(\lambda)$:

$$\mathbf{x}^* = \mathbf{U} f(\mathbf{\Lambda}) \mathbf{U}^\top \mathbf{y} \tag{4.1}$$

$$f(\mathbf{\Lambda}) = \text{diag}\left((1+\mu\lambda_1))^{-1}, \ldots, (1+\mu\lambda_N)^{-1}\right). \tag{4.2}$$

(4.1) states that an input signal $\mathbf{y}$ is transformed to the graph spectral domain $\boldsymbol{\alpha} = \mathbf{U}^\top \mathbf{y}$ via GFT $\mathbf{U}^\top$, where each graph frequency coefficient α_i of frequency i is scaled by $f(\lambda_i) = 1/(1+\mu\lambda_i)$, before transforming back to the nodal domain via inverse GFT $\mathbf{U}$. The frequency response $f(\mathbf{\Lambda})$ in (4.2) is low-pass, because for large graph frequency λ, the frequency response $f(\lambda) = 1/(1+\mu\lambda)$ is smaller. Figure 4.1 shows the filter response $f(\mathbf{\Lambda})$ for different weight parameters μ.

Given frequency response $f(\mathbf{\Lambda})$, one can avoid matrix inversion by using accelerated graph filter implementations. Chebyshev polynomials approximation [72, 73] and Lanczos method [23] are notable existing filter approximation methods in the literature. Note again that accelerated filter implementations are possible in our framework because of the availability of a model-based, analytical frequency response $f(\mathbf{\Lambda})$ for the graph filter (4.1). A chosen accelerated graph filter implementation is applicable regardless of what underlying graph is actually constructed during runtime.

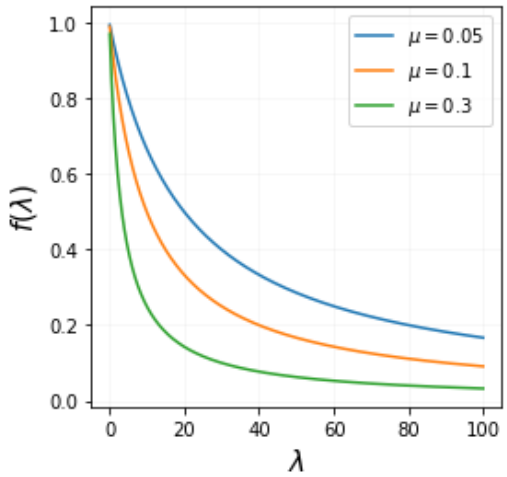

Figure 4.1: Graph filter responses $f(\mathbf{\Lambda})$ for different weight parameters μ.

4.2 Accelerated graph filter via Chebyshev polynomials approximation

As described in [73], one can compute Chebyshev coefficients c_m and polynomials $\tilde{T}_m(\mathbf{L}_\Gamma)$, $m = 0, 1, \ldots$ and use the following equation to approximate the graph filter (4.1)

$$\mathbf{x}^* \approx \frac{1}{2} c_0 \mathbf{I}\mathbf{y} + \sum_{m=1}^{\infty} c_m \tilde{T}_m(\mathbf{L}_\Gamma)\mathbf{y}. \tag{4.3}$$

Denote by $\lambda_{\max}$ the largest eigenvalue of $\mathbf{L}_\Gamma$. Then $\tilde{T}_m(\mathbf{L}_\Gamma)\mathbf{y}$ is generated using recurrence relation

$$\tilde{T}_m(\mathbf{L}_\Gamma)\mathbf{y} = 2\left(\frac{2\mathbf{L}_\Gamma}{\lambda_{\max}} - \mathbf{I}\right)\tilde{T}_{m-1}(\mathbf{L}_\Gamma)\mathbf{y} - \tilde{T}_{m-2}(\mathbf{L}_\Gamma)\mathbf{y}, \tag{4.4}$$

where $T_0(\mathbf{L}_\Gamma)\mathbf{y} = \mathbf{I}$ and $T_1(\mathbf{L}_\Gamma)\mathbf{y} = \mathbf{y}$. The fact that $\tilde{T}_m(\mathbf{L}_\Gamma)\mathbf{y}$ can be computed recursively is the main reason for fast execution time of the Chebyshev polynomials approximation.

The sum is truncated at a defined order M when implemented, resulting in M-th order Chebyshev polynomial approximation. The m-th polynomial $\tilde{T}_m$ can be computed recursively from $\tilde{T}_{m-1}$ and $\tilde{T}_{m-2}$ with time complexity of $\mathcal{O}(M|\mathcal{E}|)$, leading to computational benefits of the Chebyshev polynomial approximation.

4.3 Accelerated graph filter via Lanczos Approximation

While fast, the error of a Chebyshev approximation can be large if the order M is small. This motivates a better filter approximation. Lanczos algorithm originated from the numerical linear algebra literature, where a Hermitian matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is transformed to a tridiagonal matrix $\mathbf{T} = \mathbf{V}^*\mathbf{A}\mathbf{V} \in \mathbb{R}^{m\times m}$, where $m \leq n$ is the pre-specified iteration number. The Lanczos algorithm is conventionally used to compute eigen-pairs of a Hermitian matrix quickly. In the case of filter implementation, Lanczos approximation leverages this algorithm for filter approximation. Specifically, similar to [23], first, an orthonormal basis $\mathbf{V}_M = [\mathbf{v}_1, \ldots, \mathbf{v}_M]$ of the Krylov subspace $K_M(\mathbf{L}_\Gamma, \mathbf{y}) = \text{span}(\mathbf{y}, \mathbf{L}_\Gamma\mathbf{y}, \ldots, \mathbf{L}_\Gamma^{M-1}\mathbf{y})$ is computed using the Lanczos method described in [23]. The following symmetric tridiagonal matrix $\mathbf{H}_M$ relates $\mathbf{V}_M$ and $\mathbf{L}_\Gamma$.

α_m and β_m are scalars computed by the Lanczos algorithm.

$$\mathbf{H}_M = \mathbf{V}_M^* \mathbf{L}_\Gamma \mathbf{V}_M = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_M \\ & & & \beta_M & \alpha_M \end{bmatrix}. \tag{4.5}$$

The computational cost of the algorithm is $\mathcal{O}(M|\mathcal{E}|)$. Then, the following approximation to the graph filter in (3.7) was proposed in [74]:

$$f(\mathbf{L})\mathbf{y} \approx \|\mathbf{y}\|_2 V_M f(\mathbf{H}_M) e_1 := f_M, \tag{4.6}$$

where e_1 is the first unit vector. Due to the eigenvalue interlacing property, the eigenvalues of $\mathbf{H}_M$ are inside the interval $[\lambda_1, \lambda_N]$, and thus $f(\mathbf{H}_M)$ is well-defined. The evaluation of (4.6) is inexpensive since $M \ll N$, leading to accelerated implementation of $f(\mathbf{L})\mathbf{y}$.

The Lanczos method is adopted as the graph filter acceleration for implementation since it has lower approximation error than the Chebyshev alternative for a small order M. Figure 5.1 in Section 5 shows an example of the approximation errors of the two methods.

Chapter 5

Experimental Results

First, experiments were conducted to compare the two graph filter approximation schemes. Then, the proposed denoising method was evaluated against various state-of-the-art image denoising methods.

5.1 Comparison between the two approximation methods

The experiments in this section compare the Lanczos method with the Chebyshev method in approximating the graph filter frequency response in (4.2).

Specifically, the graph filter (4.2) was executed on 1000 image patches of size 36×36 (randomly drawn from the RENOIR dataset [75]) using three filter implementations: i) the original filter (4.2), ii) Lanczos approximation with order M, and iii) Chebyshev approximation with order M.

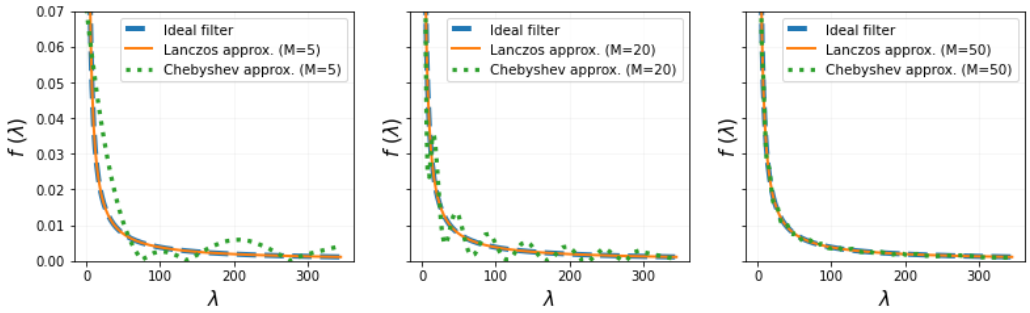

Figure 5.1: Sample frequency response of Lanczos and Chebyshev method at various M.

The average MSE between the outputs of the original filter and the two approximation methods were measured for comparison. The results are shown in Figure 5.1 and 5.2. Figure 5.1 shows a sample frequency response of the low pass filter for $M = 5, 20, 50$, and Figure 5.2 shows the approximation errors as a function of M. Clearly from the plots, the Lanczos approximation outperformed the Chebychev approximation by a large margin, especially for small M. For $M = 10$, Lanczos can reduce MSE of Chebeyshev by about 33%, and for $M = 50$, the difference is about 10%. Since both methods had the same time complexity $\mathcal{O}(M|\mathcal{E}|)$, one can conclude that the Lanczos method was superior in this experiment. Note again that these graph filter approximations are possible thanks to the analytical frequency response derived in (4.2).

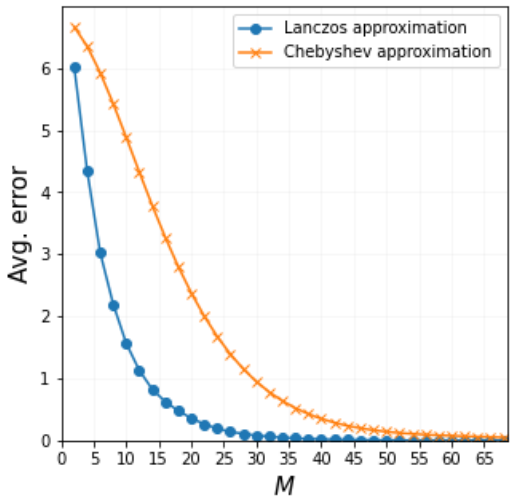

Figure 5.2: Average approximation errors with respect to M.

5.2 Denoising Performance Comparison

The experiments in this section compare the proposed DGTV algorithm against several state-of-the-art image denoising schemes. The competing schemes are: i) a well-known model-based method BM3D [17], ii) a state-of-the-art deep learning method CDnCNN [11], and iii) DGLR [19]—a hybrid of CNN and analytical graph filters derived from a convex optimization formulation regularized using GLR.

These methods were evaluated under two noise types: 1) additive white Gaussian noise, and 2) real low-light captured noise of images in the RENOIR Dataset [75]. For each type of noise, two experiment settings were considered. The first setting was a statistical match setting, where both the training and test datasets were drawn from the same distribution. The second setting

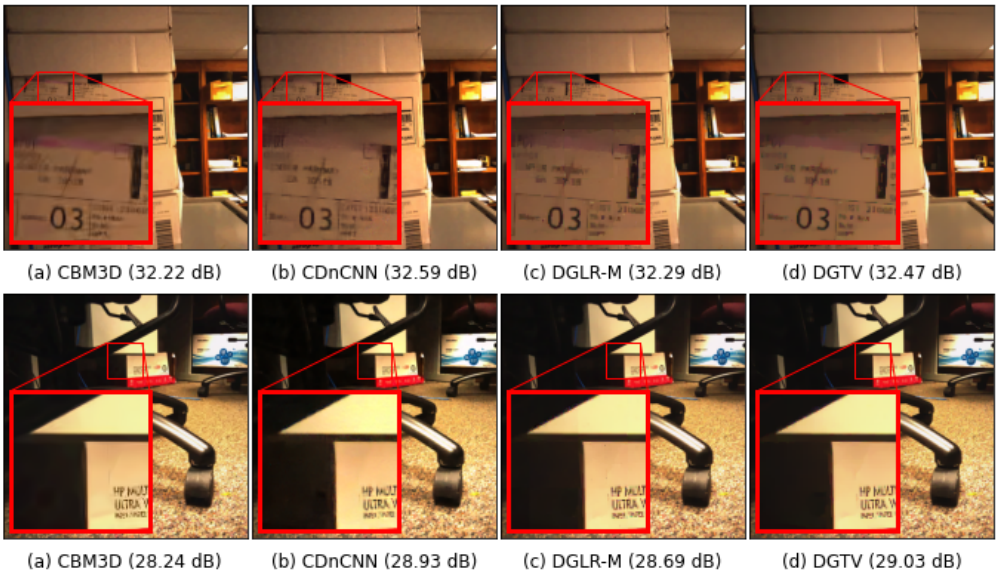

Figure 5.3: Sample results of AWGN denoising. Training and testing on $\sigma = 25$.

was a statistical mismatch setting, where images of the training and test set had different noise characteristics. Statistical mismatching mimics real world image denoising situations. In practice, access to the actual noise generation distribution is not possible. Thus, it is more realistic to assume that the model is trained on a distribution that is different than the actual distribution. Specifically, for type 1 noise, the training set was added with Gaussian noise (standard deviation of $\sigma = 25$) and the test set was added with Gaussian noise (standard deviation of $\sigma = 40$). Essentially, the images of the two datasets were drawn from the same noise distribution but with different parameters. For type 2 noise, the training set was also added with Gaussian noise (standard deviation of $\sigma = 25$), while the test set was the real low-light noise images of the RENOIR dataset. Noted that BM3D does not

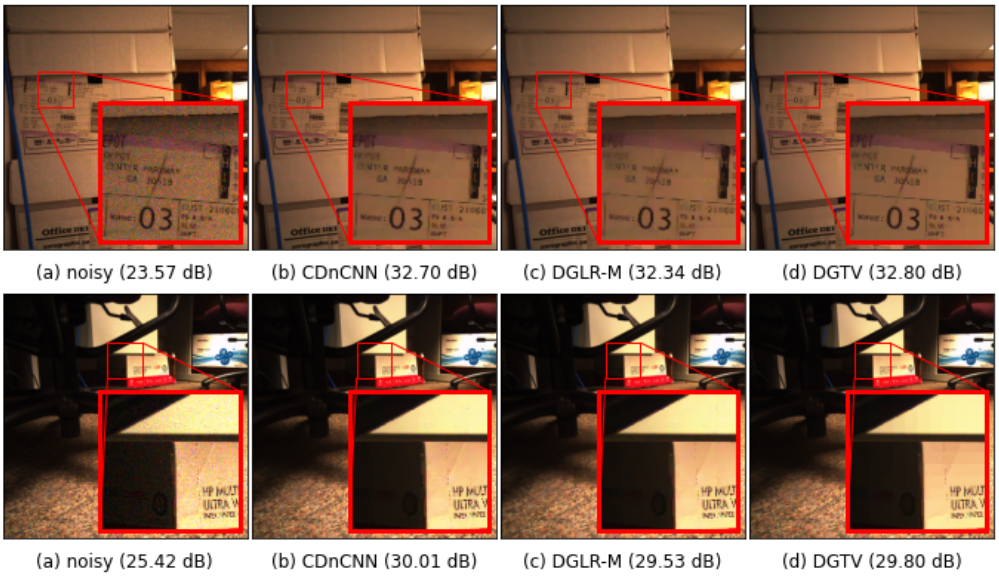

Figure 5.4: Examples of real low-light denoising. Training and testing on real low-light noise.

require training.

It took the proposed DeepGTV about 10 seconds to denoise a 720×720 image. Although the filter approximation helped speed up the executing time, the bottle neck was at the graph construction module. It is possible to reduce the execution time with further implementation optimization.

5.2.1 Dataset Description

A small subset of the RENOIR Dataset [75] was used for experiments. The subset selection criterion is described in Section 4.2 of [19]. This subset contained 10 pairs of high resolution images captured by a smartphone. Each pair contained a clean image and a noisy image (caused by low-light

capturing). Training bias is possible since this dataset contains only indoor scenes. One can adopt DeepGTV to another dataset to avoid the bias. All images were resized to 720×720 pixels, and each was divided into patches of size 36×36. Each patch overlapped its previous patch by 18 pixels. The images were randomly split into two sets, where each set contained five images. The average of peak signal-to-noise ratios (PSNR) and the average of structural similarity index measure (SSIM) [76] of five test images were used for evaluation.

5.2.2 Network architecture and hyperparameters

The CNN architectures were as following. CNN_{F} was a lightweight CNN with four 2D-convolution layers. The first layer had 3 and 32 input and output channels, respectively. The next two layers had 32 input and output channels. The last layer had 32 input channels and $K = 3$ output channels. CNN_{μ} also had four 2D-convolution layers, where the first layer had 3 and 32 input and output channels, respectively, and the rest had 32 input and output channels, followed by a fully-connected (FC) layer with 32 input units and 1 output unit. A max pooling operator was used between two convolution layers, and a Rectified Linear Unit (ReLU) was used after every convolution layer and FC layer.

The architecture was chosen by systematically trying different options. For instance, to choose the number of features K, different K's were tested. When K was large, *e.g.*, $K = 9, 12, 36$, multiple features had similar appearances

when viewed as images. Thus, K was reduced gradually until there was no redundant features, resulting in $K = 3$. Similarly, when the number of layers are too large, the gradient vanished.

Skip connections were not used to address the gradient vanishing problem, since DeepGTV only learns the features for constructing a similarity graph. This requires less learning capability than learning a function that maps from a noisy to a clean image (*e.g.*, in pure DL methods). A shallow network is sufficient and is used instead.

In each experiment, the proposed DGTV model was trained for 50 epochs using stochastic gradient descent (SGD) with batch size of 16. The learning rate was set to 10^{-4}. Other parameters were set empirically as follows: $\epsilon = 0.3, \rho = 0.01$, $B = 6$ and $M = 20$.

5.2.3 Statistical Match Noise Removal

This experiment evaluated the noise removal performance of the competing methods. Two noise types were experimented: 1) both training and test datasets were added with Gaussian noise with standard deviation of $\sigma = 25$, 2) both training and test datasets contained real low-light captured noise. To achieve comparable complexity between DGLR and DGTV, DGLR's network architectures were replaced with the same described architectures of DGTV.

The third column of Table 5.1 shows the results for Additive White Gaussian Noise, and the third column of Table 5.2 shows the results for real low-light noise. Generally, learning-based methods performed better than the

Method	# Parameters	$\left(\sigma = 25\right)$	$\begin{pmatrix}\sigma_{\text{train}} = 25\\ \sigma_{\text{test}} = 40\end{pmatrix}$
BM3D	N/A	30.19 | 0.802	N/A
CDnCNN	0.56M	**30.39** | **0.826**	24.77 | 0.482
DGLR (1 layer)	0.45M	30.24| 0.809	27.27| 0.742
DGLR-M (1 layer)	0.06M	29.66 | 0.774	26.75 | 0.666
DGTV (1 layer)	0.06M	30.29 | 0.818	27.68 | 0.766
DGLR (2 layers)	0.9M	30.36 | 0.820	27.47 | 0.763
DGLR-M (2 layers)	0.12M	30.29 | 0.817	27.19 | 0.731
DGTV (2 layers)	0.12M	30.35 | 0.820	**27.82** | **0.772**

Table 5.1: Number of trainable parameters, average PSNR (left) and SSIM (right) in AWGN removal and statistical mismatch setting.

model-based method. Compared to CDnCNN, DGTV achieved comparable performance (within 0.04dB in PSNR) while employing 80% fewer network parameters. Compared to DGLR-M, DGTV performed better given the same number of layers. Specifically, at 1 layer, DGTV outperformed DGLR-M by 0.63 dB in PSNR when removing AWGN. When removing real low-light noise, DGTV outperformed DGLR-M by 0.11dB in PSNR at 1 layer. Fig. 5.3 and Fig. 5.4 show sample results of this experiment. As expected, DGTV reconstructed piecewise-smooth (PWS) image patches like English letters on light background very well.

Method	# Parameters	$\begin{pmatrix}\text{train = Real noise}\\ \text{test = Real noise}\end{pmatrix}$	$\begin{pmatrix}\text{train = AWGN}\\ \text{test = Real noise}\end{pmatrix}$
CDnCNN	0.56M	31.60 | **0.841**	28.06 | 0.532
DGLR (1 layer)	0.45M	31.21 | 0.832	30.50 | 0.812
DGLR-M (1 layer)	0.06M	31.16 | 0.829	30.37 | 0.814
DGTV (1 layer)	0.06M	31.27 | 0.835	30.69 | 0.814
DGLR (2 layers)	0.9M	**31.62** | **0.841**	30.96 | 0.818
DGLR-M (2 layers)	0.12M	31.44 | 0.833	30.44 | 0.813
DGTV (2 layers)	0.12M	31.57 | 0.840	**31.01** | **0.820**

Table 5.2: Number of trainable parameters, average PSNR (left) and SSIM (right) in Real Low-light noise removal and statistical mismatch setting. In statistical mismatching, training data had AWGN ($\sigma = 25$).

5.2.4 Statistical Mismatch Noise Removal

To demonstrate robustness against statistical mismatch, *i.e.* the training and test data have different statistics, all models were trained on artificial noise with standard deviation of $\sigma = 25$ and tested on different statistic datasets: 1) tested with AWGN with $\sigma = 40$, and 2) tested with real low-light noise. Note again that BM3D does not require training on data, and hence, it was not evaluated in this experiment.

The final column of Table 5.1 and Table 5.2 shows the results of this experiment. From the table, one can see that DGTV performed better than DGLR-M consistently for the same number of layers, though the gap became smaller as the number of layers increased. In particular, at 1 layer, DGTV outperformed DGLR-M by 0.93dB in PSNR for AWGN removal, and by

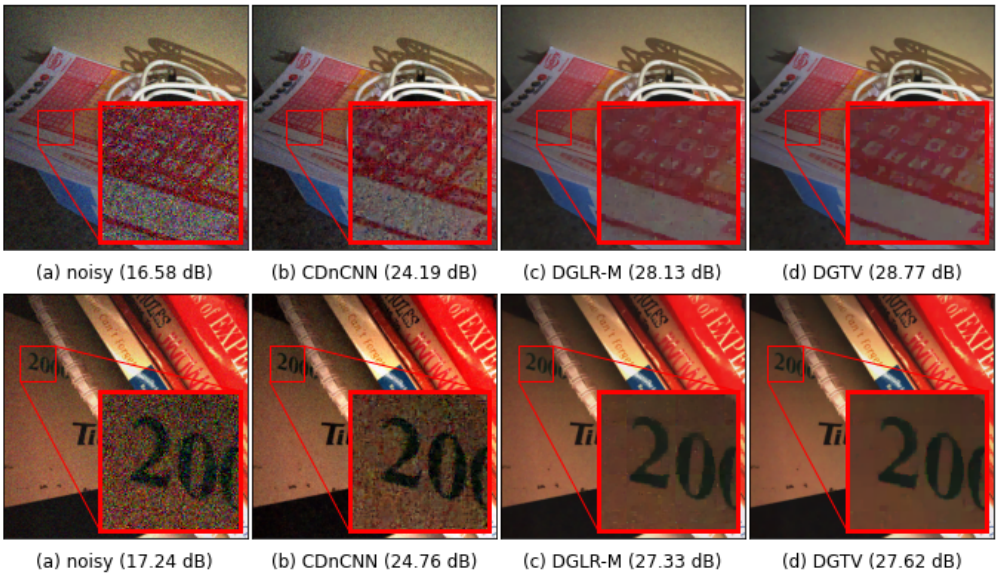

Figure 5.5: Sample result in statistical mismatch situations. Training on $\sigma = 25$ and testing on $\sigma = 40$.

0.32dB for real noise removal. Compared to CDnCNN, DGTV outperformed it by a large margin—approximately 3dB for both noise types. Because DGTV employed 80% fewer network parameters than CDnCNN, one can interpret this result on robustness against statistical mismatch to mean that DGTV implementation is less likely to overfit training data than CDnCNN. Fig. 5.5 and Fig. 5.6 show sample results of this experiment.

Figure 5.6: Sample result in statistical mismatch situations. Training on $\sigma = 25$ and testing on real low-light noise.

Chapter 6

Conclusion

Image denoising is a fundamental problem in the field of image processing and has been studied for decades. Recently, deep neural network architectures have shown great success in image denoising. However, these methods require tuning of a large number of parameters. Further, they overly depend on training data and tend to fail when the training and testing data have different noise statistics.

In this thesis, a new image denoising algorithm called Deep Graph Total Variation (DGTV) is proposed, combining graph signal processing with deep neural networks, resulting in demonstrable advantages compared to pure deep learning approaches. Specifically, a CNN is used to learn feature representations per pixel first. Then, a suitable graph for graph spectral filtering based on the distances of the learned features is constructed. Given a constructed graph, a convex optimization problem for denoising using a graph

total variation (GTV) prior is formulated. The optimization problem has a convex but non-smooth objective originally. The objective is transformed into a convex and smooth version via an ℓ_1 graph Laplacian reformulation. The transformed problem has a closed-form solution per iteration and is interpreted as a graph low-pass filter with an analytical frequency response. However, the solution requires matrix inversion, which is computationally expensive. Thanks to a graph spectral filter interpretation of GTV, matrix inversion can be avoided by using fast filter approximation in the graph spectral domain. Particularly, this solution is realized via Lanczos approximation.

The proposed approach performed as well as pure deep learning approaches in ideal denoising settings, where both training and test data had the same noise distribution. Notably, DGTV achieved the same performance level with DnCNN—a state-of-the-art deep learning approach—while using 80% fewer parameters. Moreover, when the test images had different noise statistics than the training images, DGTV outperformed DnCNN by up to 3dB in PSNR. This suggests that DGTV generalizes better than DnCNN in case of statistical mismatch and is less likely to overfit.

Using a hybrid design of interpretable analytical graph filters and deep feature learning, DeepGTV shows the potential of hybrid methods for solving image processing tasks. DeepGTV can be extended to other image problems beyond simple image denoising, *e.g.*, different data types (3D images, light-field images), different degradation types (blur, low resolution).

For future work, the number of parameters of DGTV can be reduced even

further. Specifically, the CNN_{μ} can be replaced completely with a single variable μ that can be learned from data offline *a priori*. The $\text{CNN}_{\mathbf{F}}$ can also be replaced with more frugal neural network implementations that can result in even smaller parameter sets.

Bibliography

[1] D. I Shuman, S. K Narang, P. Frossard, A. Ortega, and P. Vandergheynst. "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains". In: *IEEE signal processing magazine* 30.3 (2013), pp. 83–98.

[2] A. Ortega, P. Frossard, J. Kovacevic, J. M. F. Moura, and P. Vandergheynst. "Graph Signal Processing: Overview, Challenges, and Applications". In: *Proceedings of the IEEE* 106.5 (2018), pp. 808–828.

[3] G. Cheung, E. Magli, Y. Tanaka, and M. K. Ng. "Graph spectral image processing". In: *Proceedings of the IEEE* 106.5 (2018), pp. 907–930.

[4] W. Hu, G. Cheung, and A. Ortega. "Intra-Prediction and Generalized Graph Fourier Transform for Image Coding". In: *IEEE Signal Processing Letters* 22.11 (2015), pp. 1913–1917.

[5] J. Zeng, G. Cheung, Y. Chao, I. Blanes, J. Serra-Sagristà, and A. Ortega. "Hyperspectral image coding using graph wavelets". In: *2017 IEEE*

International Conference on Image Processing (ICIP). 2017, pp. 1672–1676.

[6] X. Su, M. Rizkallah, T. Maugey, and C. Guillemot. "Graph-based light fields representation and coding using geometry information". In: *2017 IEEE International Conference on Image Processing (ICIP)*. 2017, pp. 4023–4027. DOI: 10.1109/ICIP.2017.8297038.

[7] W. Hu, X. Li, G. Cheung, and O. Au. "Depth map denoising using graph-based transform and group sparsity". In: *2013 IEEE 15th International Workshop on Multimedia Signal Processing (MMSP)*. 2013, pp. 001–006. DOI: 10.1109/MMSP.2013.6659254.

[8] J. Pang and G. Cheung. "Graph Laplacian Regularization for Image Denoising: Analysis in the Continuous Domain". In: *IEEE Transactions on Image Processing* 26.4 (2017), pp. 1770–1785.

[9] W.-T. Su, G. Cheung, R. P. Wildes, and C.-W. Lin. "Graph Neural Net using Analytical Graph Filters and Topology Optimization for Image Denoising." In: *IEEE International Conference on Acoustics, Speech and Signal Processing*. 2020.

[10] Y. Bai, G. Cheung, X. Liu, and W. Gao. "Graph-Based Blind Image Deblurring From a Single Photograph". In: *IEEE Transactions on Image Processing* 28 (2018), pp. 1404–1418.

[11] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang. "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising". In: *IEEE Transactions on Image Processing* 26 (2017), pp. 3142–3155.

[12] R. Vemulapalli, O. Tuzel, and M.-Y. Liu. "Deep Gaussian Conditional Random Field Network: A Model-Based Deep Network for Discriminative Denoising". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 4801–4809.

[13] Y. Tai, J. Yang, X. Liu, and C. Xu. "MemNet: A Persistent Memory Network for Image Restoration". In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), pp. 4549–4557.

[14] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal. "Explaining Explanations: An Overview of Interpretability of Machine Learning". In: *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. 2018, pp. 80–89.

[15] L. I. Rudin, S. Osher, and E. Fatemi. "Nonlinear total variation based noise removal algorithms". In: *Physica D: Nonlinear Phenomena* 60.1 (1992), pp. 259–268. ISSN: 0167-2789. DOI: https://doi.org/10.1016/0167-2789(92)90242-F. URL: http://www.sciencedirect.com/science/article/pii/016727899290242F.

[16] M. Elad and M. Aharon. "Image Denoising Via Sparse and Redundant Representations Over Learned Dictionaries". In: *IEEE Transactions on Image Processing* 15.12 (2006), pp. 3736–3745.

[17] K. Dabov, A. Foi, V. Katkovnik, and K. O. Egiazarian. "Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering". In: *IEEE Transactions on Image Processing* 16 (2007), pp. 2080–2095.

[18] A. Beck and M. Teboulle. "Fast Gradient-Based Algorithms for Constrained Total Variation Image Denoising and Deblurring Problems". In: *IEEE Transactions on Image Processing* 18.11 (2009), pp. 2419–2434.

[19] J. Zeng, J. Pang, W. Sun, and G. Cheung. "Deep Graph Laplacian Regularization for Robust Denoising of Real Images". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2019, pp. 1759–1768.

[20] X. Liu, G. Cheung, X. Wu, and D. Zhao. "Random Walk Graph Laplacian-Based Smoothness Prior for Soft Decoding of JPEG Images". In: *IEEE Transactions on Image Processing* 26.2 (2017), pp. 509–524.

[21] C. Tomasi and R. Manduchi. "Bilateral filtering for gray and color images". In: *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. 1998, pp. 839–846.

[22] V. Monga, Y. Li, and Y. C. Eldar. "Algorithm Unrolling: Interpretable, Efficient Deep Learning for Signal and Image Processing". In: *arXiv:1912.10557* (2019). arXiv: `1912.10557 [eess.IV]`.

[23] A. Susnjara, N. Perraudin, D. Kressner, and P. Vandergheynst. "Accelerated filtering on graphs using Lanczos method". In: *arXiv:1509.04537* (2015). arXiv: 1509.04537 [math.NA].

[24] G. Strang. "The Discrete Cosine Transform". In: *SIAM Rev.* 41.1 (1999), pp. 135–147. ISSN: 0036-1445. DOI: 10.1137/S0036144598336745. URL: https://doi.org/10.1137/S0036144598336745.

[25] J. Friedman, T. Hastie, and R. Tibshirani. "Sparse inverse covariance estimation with the graphical lasso". In: *Biostatistics* 9.3 (July 2008), pp. 432–441. ISSN: 1465-4644, 1468-4357. DOI: 10.1093/biostatistics/kxm045.

[26] S. I. Daitch, J. A. Kelner, and D. A. Spielman. "Fitting a Graph to Vector Data". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. Montreal, Quebec, Canada: Association for Computing Machinery, 2009, pp. 201–208. ISBN: 9781605585161. DOI: 10.1145/1553374.1553400.

[27] B. Cheng, J. Yang, S. Yan, Y. Fu, and T. S. Huang. "Learning With ℓ^1-Graph for Image Analysis". In: *IEEE Transactions on Image Processing* 19.4 (2010), pp. 858–866. DOI: 10.1109/TIP.2009.2038764.

[28] H. E. Egilmez, E. Pavez, and A. Ortega. "Graph Learning From Data Under Laplacian and Structural Constraints". In: *IEEE Journal of Selected Topics in Signal Processing* 11.6 (2017), pp. 825–841. DOI: 10.1109/JSTSP.2017.2726975.

[29] S. Bagheri, G. Cheung, A. Ortega, and F. Wang. "Learning Sparse Graph Laplacian with K Eigenvector Prior via Iterative GLASSO and Projection". In: *arXiv:2010.13179* (2021). arXiv: 2010.13179 [eess.SP].

[30] A. Gadde, S. K. Narang, and A. Ortega. "Bilateral filter: Graph spectral interpretation and extensions". In: *2013 IEEE International Conference on Image Processing*. 2013, pp. 1222–1226. DOI: 10.1109/ICIP.2013.6738252.

[31] J. Pang, G. Cheung, W. Hu, and O. C. Au. "Redefining self-similarity in natural images for denoising using graph signal gradient". In: *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2014 Asia-Pacific*. 2014, pp. 1–8. DOI: 10.1109/APSIPA.2014.7041627.

[32] J. Pang, G. Cheung, A. Ortega, and O. C. Au. "Optimal graph laplacian regularization for natural image denoising". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2015, pp. 2294–2298. DOI: 10.1109/ICASSP.2015.7178380.

[33] F. Chung. *Spectral Graph Theory*. CBMS Regional Conference Series in Mathematics, 1997.

[34] C. Couprie, L. Grady, L. Najman, J.-C. Pesquet, and H. Talbot. "Dual constrained TV-based regularization on graphs". In: *SIAM Journal on Imaging Sciences* 6.3 (July 2013), pp. 246–1273. DOI: 10.1137/120895068.

[35] M. Hidane, O. Lezoray, and A. Elmoataz. "Nonlinear Multilayered Representation of Graph-Signals". In: *Journal of Mathematical Imaging and Vision* 45 (2013), pp. 114–137.

[36] Y. LeCun, Y. Bengio, and G. Hinton. "Deep Learning". In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539.

[37] V. Nair and G. E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel, 2010, pp. 807–814.

[38] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012.

[40] K. Simonyan and A. Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[41] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[42] I. Hadji and R. P. Wildes. *What Do We Understand About Convolutional Networks?* 2018. arXiv: 1803.08834 [cs.CV].

[43] R. S. Sutton. "Two Problems with Backpropagation and Other Steepest-Descent Learning Procedures for Networks". In: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. 1986.

[44] J. Kukacka, V. Golkov, and D. Cremers. *Regularization for Deep Learning: A Taxonomy*. 2017. arXiv: 1710.10686 [cs.LG].

[45] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958.

[46] A. Beck and M. Teboulle. "A fast Iterative Shrinkage-Thresholding Algorithm with application to wavelet-based image deblurring". In: *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*. 2009, pp. 693–696. DOI: 10.1109/ICASSP.2009.4959678.

[47] Y. Li and S. Osher. *Coordinate descent optimization for minimization with application to compressed sensing; a greedy algorithm*. 2009. DOI: 10.3934/ipi.2009.3.487.

[48] K. Gregor and Y. LeCun. "Learning Fast Approximations of Sparse Coding". In: *ICML*. 2010.

[49] B. Goyal, A. Dogra, S. Agrawal, B.S. Sohi, and A. Sharma. "Image denoising review: From classical to state-of-the-art approaches". In: *Information Fusion* 55 (2020), pp. 220–244. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2019.09.003.

[50] S. Osher, M. Burger, D. Goldfarb, J. Xu, and W. Yin. "An Iterative Regularization Method for Total Variation-Based Image Restoration". In: *Multiscale Modeling & Simulation* 4.2 (2005), pp. 460–489. DOI: 10.1137/040605412.

[51] A. Buades, B. Coll, and J. -. Morel. "A non-local algorithm for image denoising". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 2. 2005, 60–65 vol. 2. DOI: 10.1109/CVPR.2005.38.

[52] U. Schmidt and S. Roth. "Shrinkage Fields for Effective Image Restoration". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 2774–2781. DOI: 10.1109/CVPR.2014.349.

[53] S. Gu, L. Zhang, W. Zuo, and X. Feng. "Weighted Nuclear Norm Minimization with Application to Image Denoising". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 2862–2869. DOI: 10.1109/CVPR.2014.366.

[54] J. Shanks, S. Treitel, and J. Justice. "Stability and synthesis of two-dimensional recursive filters". In: *IEEE Transactions on Audio and*

Electroacoustics 20.2 (1972), pp. 115–128. DOI: 10.1109/TAU.1972.1162358.

[55] D. Strong and T. Chan. "Edge-preserving and scale-dependent properties of total variation regularization". In: *Inverse Problems* 19.6 (Nov. 2003), S165–S187. DOI: 10.1088/0266-5611/19/6/059.

[56] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman. "Non-local sparse models for image restoration". In: Sept. 2009, pp. 2272–2279. DOI: 10.1109/ICCV.2009.5459452.

[57] A. Lucas, M. Iliadis, R. Molina, and A. K. Katsaggelos. "Using Deep Neural Networks for Inverse Problems in Imaging: Beyond Analytical Methods". In: *IEEE Signal Processing Magazine* 35.1 (2018), pp. 20–36. DOI: 10.1109/MSP.2017.2760358.

[58] J. Liang and R. Liu. "Stacked denoising autoencoder and dropout together to prevent overfitting in deep neural network". In: *2015 8th International Congress on Image and Signal Processing (CISP)*. 2015, pp. 697–701. DOI: 10.1109/CISP.2015.7407967.

[59] K. Zhang, W. Zuo, and L. Zhang. "FFDNet: Toward a Fast and Flexible Solution for CNN based Image Denoising". In: *IEEE Transactions on Image Processing* PP (Oct. 2017). DOI: 10.1109/TIP.2018.2839891.

[60] X. Liu, D. Zhai, D. Zhao, G. Zhai, and W. Gao. "Progressive Image Denoising Through Hybrid Graph Laplacian Regularization: A Unified

Framework". In: *IEEE Transactions on Image Processing* 23.4 (2014), pp. 1491–1503.

[61] W. Hu, G. Cheung, X. Li, and O. C. Au. "Graph-based joint denoising and super-resolution of generalized piecewise smooth images". In: *2014 IEEE International Conference on Image Processing (ICIP)*. 2014, pp. 2056–2060. DOI: 10.1109/ICIP.2014.7025412.

[62] S. K. Narang and A. Ortega. "Compact Support Biorthogonal Wavelet Filterbanks for Arbitrary Undirected Graphs". In: *IEEE Transactions on Signal Processing* 61.19 (2013), pp. 4673–4685.

[63] D. Valsesia, G. Fracastoro, and E. Magli. "Deep Graph-Convolutional Image Denoising". In: *IEEE Transactions on Image Processing* 29 (2020), pp. 8226–8237.

[64] S. Chen, Y. C. Eldar, and L. Zhao. "Graph Unrolling Networks: Interpretable Neural Networks for Graph Signal Denoising". In: *arXiv:2006.01301* (2020). arXiv: 2006.01301 [eess.SP].

[65] F. Xue, F. Luisier, and T. Blu. "Multi-Wiener SURE-LET Deconvolution". In: *IEEE Transactions on Image Processing* 22.5 (2013), pp. 1954–1968. DOI: 10.1109/TIP.2013.2240004.

[66] H. Takeda, S. Farsiu, and P. Milanfar. "Deblurring Using Regularized Locally Adaptive Kernel Regression". In: *IEEE Transactions on Image Processing* 17.4 (2008), pp. 550–563. DOI: 10.1109/TIP.2007.918028.

[67] A. Kheradmand and P. Milanfar. "A General Framework for Regularized, Similarity-Based Image Restoration". In: *IEEE Transactions on Image Processing* 23.12 (2014), pp. 5136–5151.

[68] K. Yamamoto, M. Onuki, and Y. Tanaka. "Deblurring of point cloud attributes in graph spectral domain". In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016, pp. 1559–1563.

[69] F. Lan, C. Yang, G. Cheung, and J. Z. G. Tan. "Joint Demosaicking / Rectification Of Fisheye Camera Images Using Multi-Color Graph Laplacian Regularization". In: *2020 IEEE International Conference on Image Processing (ICIP)*. 2020, pp. 2596–2600. DOI: 10.1109/ICIP40778.2020.9190859.

[70] C. Hu, L. Cheng, and Y. M. Lu. "Graph-based regularization for color image demosaicking". In: *2012 19th IEEE International Conference on Image Processing*. 2012, pp. 2769–2772. DOI: 10.1109/ICIP.2012.6467473.

[71] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[72] M. Onuki, S. Ono, K. Shirai, and Y. Tanaka. "Fast Singular Value Shrinkage With Chebyshev Polynomial Approximation Based on Signal Sparsity". In: *IEEE Transactions on Signal Processing* 65.22 (2017), pp. 6083–6096.

[73] D. I. Shuman, P. Vandergheynst, D. Kressner, and P. Frossard. "Distributed Signal Processing via Chebyshev Polynomial Approximation". In: *IEEE Transactions on Signal and Information Processing over Networks* 4.4 (2018), pp. 736–751.

[74] E. Gallopoulos and Y. Saad. "Efficient Solution of Parabolic Equations by Krylov Approximation Methods". In: *SIAM Journal on Scientific and Statistical Computing* 13.5 (1992), pp. 1236–1264. DOI: 10.1137/0913071.

[75] J. Anaya and A. Barbu. "RENOIR - A Dataset for Real Low-Light Noise Image Reduction". In: *Journal of Visual Communication and Image Representation* 51 (2018), pp. 144–154.

[76] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. "Image quality assessment: from error visibility to structural similarity". In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.