DEEP GRAPH CONVOLUTIONAL NETWORKS FOR IMAGE DENOISING

NIKOLA JANJUŠEVIĆ, NPJ226

December 18, 2020

Abstract. This report walks through the implementation of the Graph Convolutional Denosising Network (GDCN) as detailed in [2]. The mentioned work proposes a novel image denoising deep neural-network that dynamically builds K-regular graphs in feature space to exploit non-local self-similarity of the underlying signal. It is an addition to the ever growing body of work on non-local methods in signal processing, and has achieved state of the art-performance on popular image denoising benchmarks (both real and synthetic). Background is given to understand the mechanics of the algorithm's novelty. Some additional insights are given in the implementation of the method. Small scale experiments are done to verify the benefit of the graph signal processing.

1. Introduction

Denoising is perhaps the most fundamental problem in signal processing. Although it has received decades, if not centuries, of attention, it remains a hot topic in today's literature as a proving ground for new ideas and algorithms. This is largely because the denoising problem presents one of the simplest observation model, $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{n}$, where $\mathbf{n}$ is sample of noise following some distribution, and the observation operator $\mathbf{A}$ (in this case) is the identity. We refer to the problem of obtaining the signal $\mathbf{x}$ from observation $\mathbf{y}$ as an *inverse problem*, which in image-processing includes densoising, deblurring, super-resoluiton, and more. Thus, for the most part, any inverse-problem solving algorithm must also be a good denoiser. To further simplify the problem, as will be done in this implementation, we often look at evaluating our algorithms on synthetic white Gaussian noise, where $\mathbf{n} \sim \mathcal{N}(0, \sigma_n^2)$. Though additive Gaussian white noise (AWGN) is by no means a sophisticated model for real world noise (especially camera noise), it is a convenient benchmark which is widespread in the literature. See [**real˙denoising˙refs**] for extending modern denoising algorithms to real-world noise models.

In this report we present an implementation of a state-of-the-artdeep-learning based denoising method [2] for images. Recently, learning based algorithms have come to dominate not only computer-vision tasks but also image-processing tasks such as denoising. This leap in performance was generated by the advent (and later efficient implementation) of convolutional neural networks (CNNs). Much of the CNNs success can be attributed to its bias towards exploiting local information (via convolution). Further gains in the performance of deep-learning methods are increasingly being sought by exploiting both local and non-local information of the input signal. These "non-local" methods come in many different flavors, mainly differing in their definitions of non-local neighbors and aggregation operators.

The remainder of this report is outlined as follows. In Section 2 we first introduce the concepts and notation necessary for understanding the graph signal processing used in the GCDN [2]. We then go over the differences between the non-local operations of GCDN and other popular (learned) non-local methods. In Section 3 we detail our implemented GCDN. Finally, in Section 4 we show some experiments from the implementation.

2. Background and Related Works

2.1. **Convolutions on Graphs.** The emerging field of graph signal processing (GSP) seeks to extend classical signal processing knowledge and techniques to signals defined on irregular domains (non-uniform grids, etc.). In doing so a more general formulation of common operations, such as convolution, must be obtained, which may also bring benefits to signal processing on regular domains, such as images. Lets first introduce some notation:

Notation: Consider a multi-channel signal (ex. RGB image) with C channels and N pixels, $\mathbf{x} \in \mathbb{R}^{CN}$.

- $\mathbf{x}[i] \in \mathbb{R}^C$, feature vector corresponding to pixel i.
- $\mathbf{x}^k \in \mathbb{R}^N$, channel k of $\mathbf{x}$.
- $\mathcal{G}(\mathcal{V}, \mathcal{E})$, directed graph with vertices $\mathcal{V}$ and edges $\mathcal{E}$.
- $\mathcal{V}$, set of vertices (pixels). Cardinality $|\mathcal{V}| = N$.
- $\mathcal{E}$, set of edges. $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$.
- $\mathcal{N}(i) = \{j | (i,j) \in \mathcal{E}\}$, set of neighbors of vertex i.
- $\mathcal{L} : \mathcal{E} \to \mathbb{R}^M$, label mapping.
- $\mathcal{L}[i,j] \in \mathbb{R}^M$, label for the edge connecting vertex j to i.

2.1.1. *Frequency vs. Spatial Domain Definitions of Graph Convolution.* Convolution is a staple of signal processing as it defines any linear shift-invariant (LSI) system. The operator is diagonalized by the Fourier basis and as such can equivalently characterized in both the frequency and spatial domain. For extending the concept of convolutions to signals defined on graphs, several formulations have been proposed. A graph-laplacian matrix based on the graph's adjacency matrix can be used to yield a spectral decomposition. However, the formation and application of this operator can be expensive, which is undesirable for situations with dynamically changing graphs. An alternative approach is to define graph convolution in the spatial domain as an aggregation over neighboring vertices. This more general formulation loses the nice properties of spectral theory gains simplicity, flexibility, and speed.

2.1.2. *Edge Conditioned Convolutions.* The GCDN architecture uses the graph convolution formulation from [1] known as edge conditioned convolution (ECC). Suppose we have a multi-channel signal with C_in channels and N pixels, $\mathbf{x} \in \mathbb{R}^{C_{\text{in}}N}$ and an associated graph with vertices, edges, and edge-labels, $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{L})$, $|\mathcal{V}| = N$. Further, consider the mapping $\mathcal{F} : \mathbb{R}^M \to \mathbb{R}^{C_{\text{out}} \times C_{\text{in}}}$. We define the edge conditioned convolution of $\mathbf{x}$ by $\mathcal{F}$ as

$$\begin{aligned} \mathbf{y}[i] &= \sum_{j \in \mathcal{N}(i)} \mathcal{F}(\mathcal{L}[i,j]) x[j], \quad i = 1, \ldots, N \qquad (1) \\ &= \sum_{j \in \mathcal{N}(i)} \mathbf{H}_{i,j} x[j] \in \mathbb{R}^{C_{\text{out}}}. \end{aligned}$$

We call $\mathcal{F}$ the label-operator mapping and $\mathbf{H}_{i,j}$ the (edge-conditioned) feature operator from vertex j to i. ECC is performing a node-aggregation in which the contribution of an input node in the neighborhood is determined by an associated label-generated operator.

In the 1D case ($C_{\text{in}} = C_{\text{out}} = 1$), we note that ECC can reduce to the linear convolution, $\mathbf{y} = \mathbf{h} * \mathbf{x}$. First, we define the graph such that only immediate neighbors are connected, and edge-labels as $\mathcal{L}[i,j] = \delta[i-j]$, where δ is the discrete impulse function. Then, define the label-operator mapping as $\mathcal{F}(x) = \mathbf{h}^\top \mathbf{x}$. Together, we arrive at $\mathbf{y}[i] = \sum_{j=-M/2}^{M/2-1} \mathbf{h}[i-j]\mathbf{x}[j]$, the famed discrete convolution formula.

3. Implemented Denoiser

3.0.1. *Overview.* An overview of the GCDN architecture is presented in Figure 1. The network employs a residual learning strategy by adding the input signal to the output of the very last stage. The main path consists of a "multi-scale" preprocessing block (3 parallel paths processing the input with different filter sizes, $3, 5, 7$). After the preprocessing stage, the signal goes through several "LPF" blocks which themselves involve a residual connection between input and output. Another residual connection in the main path is given from the preprocessing block, through a "HPF" block, and connected to the output of each "LPF" block. The heavy use of residual connections (and leaky-ReLUs) is reportedly for the purpose of mitigating vanishing gradients. Furthermore, the architecture's inspiration is said

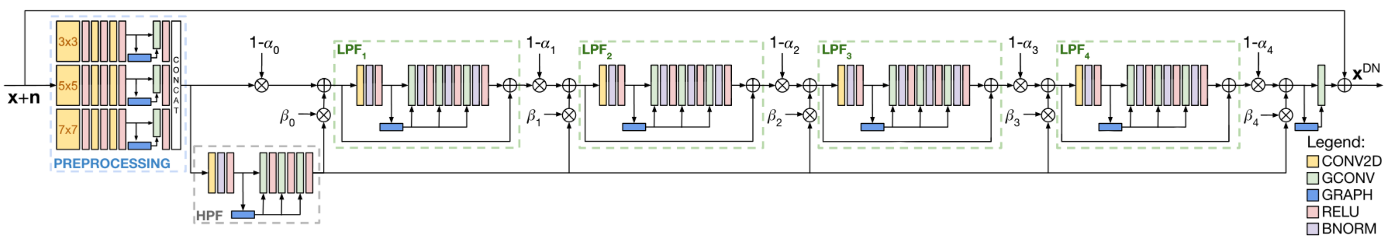

FIGURE 1. Graph Convolutional Densoising Network [2] architecture. All ReLU blocks are leakly-ReLUs with negative slope of 0.2. α_i, β_i are scalars learned for each iteration.

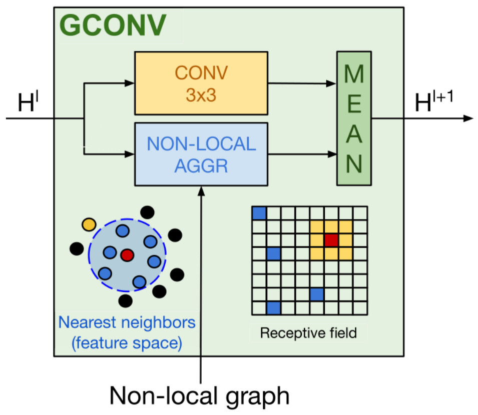

FIGURE 2. Graph convolutional layer [2]. A local 2D convolution is averaged with a non-local edge-conditioned convolution given an input graph. The input graph explicitly excludes neighbors within each pixels local area, defined by the size of the local convolution's filters.

to derive from an algebraic manipulation of an unrolled proximal-gradient descent algorithm with a graph-laplacian regularizer [2].

The defining feature of the network resides within each of these blocks: the graph convolutional layer. These layers perform edge-conditioned convolutions with the input signal provided an input graph. The graphs are formed as K-regular graphs via a K nearest neighbors (KNN) algorithm in the feature domain. The edge-labels of said graph are defined as the vector from node j to i, $\mathcal{L}[i,j] = \mathbf{z}[j] - \mathbf{z}[i]$, where $\mathbf{z}$ is an input latent signal. As the computation of the KNN graph can be computationally expensive, the same graph is reused through a single network block – although the labels still change within blocks.

3.1. Graph Convolutional Layer.

The graph convolutional layer is defined as the average of a (local) convolutional operation and a non-local graph convolution operation, as represented pictorially in Figure 2.

The authors of GCDN [2] employ an augmented definition of ECC,

$$z[i]^{(\ell+1),\mathrm{NL}} = \sum_{j\in\mathcal{N}(i)} \gamma[i,j]\mathcal{F}^{(\ell)}_{\mathbf{w}^{(\ell)}}(\mathcal{L}[i,j])\mathbf{z}[j]^{(\ell)} \tag{2}$$

$$= \sum_{j\in\mathcal{N}(i)} \gamma[i,j]\Theta_{i,j}\mathbf{z}[j]^{(\ell)}. \tag{3}$$

This formulation is consistent with our previous formulation up to the label dependent scale factor,

$$\gamma[i,j] = \exp\left(-\|L[i,j]\|^2/\delta\right).$$

In fact, this formulation is exactly equivalent to the original ECC definition except that we explicitly model a dependence of proximity in the feature domain (label-domain). The graph convolutional layer is then defined as

$$\mathbf{z}^{(\ell+1)} = \frac{\mathbf{z}^{(\ell+1),\mathrm{L}} + \mathbf{z}^{(\ell+1),\mathrm{NL}}}{2} + \mathbf{b}^{(\ell)}. \tag{4}$$

The label-operator mapping $\mathcal{F}$ is implemented by a 2-layer multi-layer perceptron (MLP) for each graph-conv layer, with some caveats for memory consideration and over-parameterization, as described in the following section.

3.2. **Lightweight ECC.** Consider a signal inside the network of M channels and N pixels, which is to be transformed into another signal of M channels. In the graph convolutional layer, we propose that for every pixel, its K neighbors each have their own feature-operator of size $M \times M$. For processing relatively small image of size $N = 256$, with feature vectors $M = 128$ and $K = 8$ nearest neighbors, this single layer would require 64Gb to store just the associated feature-operators. Further more, the feature-operators are generated by a single feature vector, and thus largely over-parameterized. The GCDN network thus employs circulant approximations of dense layers in its label-operator mappings $\mathcal{F}$, and low-rank feature-operator matrices to tackle the over parameterization and memory problems respectively.

Circulant approximation of dense layers: The circulant approximation of dense matrices is applied in both layers of the MLPs that define $\mathcal{F}$. It can be viewed as replacing a matrix multiplication with a 1D (valid) convolution where the filters are of the size of the input and the input is padded with m values (from a circular shift). We then say that $\mathcal{F}$ is approximated with m-circulant matrices. Clearly, this approximation is able to reduce the number of parameters in $\mathcal{F}$ by a factor of m.

Low-rank edge conditioned convolution: We can further reduce the computational and memory burdens of ECC by explicitly factorizing the feature-operators $\Theta[i, j]$ as a sum of r outer-products, yielding an operator of at most rank r. This is implemented by having three parallel branches in the second layer of $\mathcal{F}$ MLP that produce factors, $\Theta^L, \kappa, \Theta^R$ to create an SVD-like factorization: $\Theta = \Theta^L \,\mathbf{diag}(\kappa)\Theta^R$.

Further care is taken to insure proper initialization of the label-operator mapping for smoother training.

4. Experiments

The Pytorch implementation of GCDN is available online[1].

4.1. **Training Setup.** The network was trained on patches of size 32×32 from the Berkley Segmentation Dataset, BSD400 [**bsd**], and evaluated on the associated test set, BSD68. Validation was performed on the Set12 dataset of famous signal processing images. A batch size of 4 is used and the networks are trained for roughly $40k$ iterations – 20% of the total training time reported in the original paper. Despite this, training still took 36 hours on two GPUs for each network. Other implementation details, such as rank and circulant approximation, match that of the original paper. During training, nearest neighbor computations are taken over the whole patch. During inference however, a sliding window of 32×32 is used to compute the KNN for each pixel. This method of inference allows the receptive field of the network to adaptive in fantastic ways across the input images.

4.2. **Ablation Study.** An ablation study was performed to demonstrate the efficacy of the graph convolutions towards denoising performance. Despite the relatively short training in comparison to the original paper, the performance between networks with and without graph-convolutions appears to separate quite early. However, as common with deep neural networks, we see that the wider networks are not able to achieve superior performance so early in training. Table **??** shows the results of this ablation study, with a consistent gain in performance by using graph convolutions in the network.

4.3. **Adaptive Receptive Field.** In this section we marvel at the adaptive receptive field generated by the first LPF1 block one of the trained networks. The KNN graph is generated by the learned features representations. After passing through several graph convolutional layers, the receptive filed is able to grow along with semantically connected objects in the seen (such as edges of the same texture) instead of simply radially outward as in a vanilla CNN. Figure 3 such an adaptive receptive field.

[1]`github.com/nikopj/DGCN`

Width	Nearest Neighbors	Test PSNR (dB)
27	None	28.59
27	8	28.67
81	None	28.20
81	8	28.57

TABLE 1. Ablation study involving network width (number of features) and use of graph-convolutions. Test PSNR is reported on BSD68 with noise standard deviation of $\sigma_n = 25$ on a scale of 255.

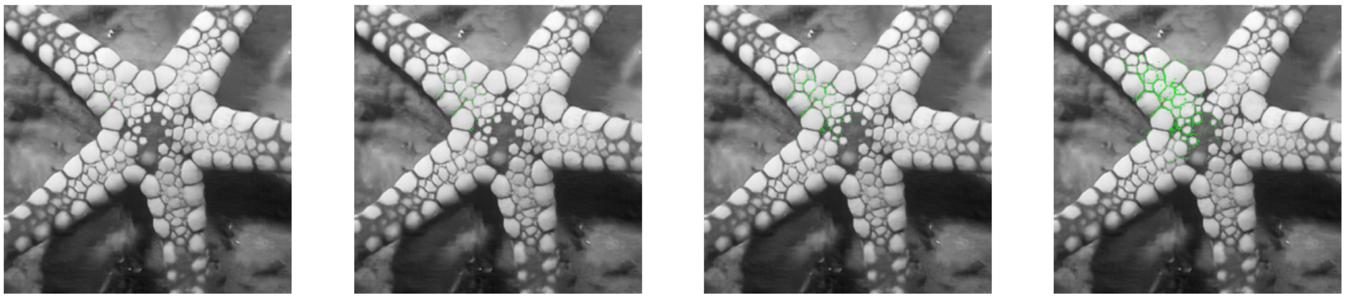

FIGURE 3. Adaptive receptive field of the GCDN in its intermediate layers. Red pixel denotes the center pixel and the green denotes pixels in its receptive field as it goes deeper through the network, left to right.

5. Conclusion

The GCDN network [2] was implemented in Pytorch. Many issues were faced in implementation regarding memory and compute power. This may be due to some semi-naive implementations of nearest neighbor searches, or simply the use of python for such high-performance tasks. Further improvements can be made by making use of a lower-level packages for faster operations (see Facebook's FAISS). Despite this, a personal goal of obtaining a visual representation of the GDCN's adaptive receptive field was obtained, and the efficacy of the proposed method was validated.

References

[1] Martin Simonovsky and Nikos Komodakis. "Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs". In: *CVPR*. 2017. URL: `https://arxiv.org/abs/1704.02901`.

[2] Diego Valsesia, Giulia Fracastoro, and Enrico Magli. "Image Denoising with Graph-Convolutional Neural Networks". In: *2019 26th IEEE International Conference on Image Processing (ICIP)*. 2019.