Deep Graph-Convolutional Image Denoising

Deep Graph-Convolutional Image Denoising

Diego Valsesia, *Member, IEEE,* Giulia Fracastoro, *Member, IEEE,* Enrico Magli, *Fellow, IEEE*

***Abstract*—Non-local self-similarity is well-known to be an effective prior for the image denoising problem. However, little work has been done to incorporate it in convolutional neural networks, which surpass non-local model-based methods despite only exploiting local information. In this paper, we propose a novel end-to-end trainable neural network architecture employing layers based on graph convolution operations, thereby creating neurons with non-local receptive fields. The graph convolution operation generalizes the classic convolution to arbitrary graphs. In this work, the graph is dynamically computed from similarities among the hidden features of the network, so that the powerful representation learning capabilities of the network are exploited to uncover self-similar patterns. We introduce a lightweight Edge-Conditioned Convolution which addresses vanishing gradient and over-parameterization issues of this particular graph convolution. Extensive experiments show state-of-the-art performance with improved qualitative and quantitative results on both synthetic Gaussian noise and real noise.**

Keywords*—Graph neural networks, image denoising, graph convolution

I. Introduction

Denoising is a staple among image processing problems and its importance cannot be overstated. Despite decades of work and countless methods, it still remains an active research topic because its purpose goes far beyond generating visually pleasing pictures. Denoising is fundamental to enhance the performance of higher-level computer vision tasks such as classification, segmentation or object recognition, and is a building block in the solution to various problems [1]–[4]. The recent successes achieved by convolutional neural networks (CNNs) extended to this problem as well and have brought a new generation of learning-based methods that is redefining the state of the art. However, it is important to learn the lessons of past research on the topic and integrate them with the new deep learning techniques. In particular, classic denoising methods, such as BM3D [5] and non-local means [6], showed the importance of exploiting non-local self-similar patterns. However, the convolution operation underpinning all CNNs architectures [7]–[10] is unable to capture such patterns because of the locality of the convolution kernels. Only very recently, some works started addressing the integration of non-local information into CNNs [11]–[14].

This paper presents a denoising neural network, called GCDN, where the convolution operation is generalized by means of graph convolution, which is used to create layers with hidden neurons having non-local receptive fields that successfully capture self-similar information. Graph convolution is a generalization of the traditional convolution operation when the data are represented as sitting over the vertices of a graph. In this work, every pixel is a vertex and the edges in the graph are dynamically computed from the similarities in the feature space of the hidden layers of the network. This allows us to exploit the powerful representational features of neural networks to discover and use latent self-similarities. With respect to other CNNs integrating non-local information for the denoising task, the proposed approach has several advantages: i) it creates an adaptive receptive field for the pixels in the hidden layers by dynamically computing a nearest-neighbor graph from the latent features; ii) it creates dynamic non-local filters where feature vectors that may be spatially distant but close in a latent vector space are aggregated with weights that depend on the features themselves; iii) the aggregation weights are estimated by a fully-learned operation, implemented as a subnetwork, instead of a predefined parameterized operation, allowing more generality and adaptability. Starting from the Edge-Conditioned Convolution (ECC) definition of graph convolution, we propose several improvements to address stability, over-parameterization and vanishing gradient issues. Finally, we also propose a novel neural network architecture which draws from an analogy with an unrolled regularized optimization method.

A preliminary version of this work appeared in [15]. There are several differences with the work in this paper. The architecture of the network is improved by drawing an analogy with proximal gradient descent methods, and it is significantly deeper. Moreover, we propose several solutions to address the ECC overparameterization and computational issues. Finally, we also present an in-depth analysis of the network behavior and greatly extended experimental results.

Notation and paper outline

We denote vectors and matrices by lowercase and upper case boldface characters, respectively. Unless otherwise specified we use column vectors. The i-th column of matrix $\mathbf{H}$ is denoted as $\mathbf{H}_i$. The i, j entry of matrix $\mathbf{H}$ is denoted as $\mathbf{H}_{i,j}$. Sets are represented with calligraphic letters.

This paper is structured as follows. Sec. II provides some background material on graph-convolutional neural networks and state-of-the-art denoising approaches. Sec. III describes the proposed method. Sec. IV analyzes the characteristics of the proposed method and experimentally compares it with state-of-the-art approaches. Finally, Sec. V draws some conclusions.

The authors are with Politecnico di Torino – Department of Electronics and Telecommunications, Italy. email: {name.surname}@polito.it. This research has been partially funded by the SmartData@PoliTO center for Big Data and Machine Learning technologies. We thank Nvidia for donating a Quadro P6000 GPU.

II. Related work

A. Graph neural networks

Inspired by the overwhelming success of deep neural networks in computer vision, a significant research effort has recently been made in order to develop deep learning methods for data that naturally lie on irregular domains. One case is when the data domain can be structured as a graph and the data are defined as vectors on the nodes of this graph. Extending CNNs from signals with a regular structure, such as images and video, to graph-structured signals is not straightforward, since even simple operations such as shifts are undefined over graphs.

One of the major challenges in this field is defining a convolution-like operation for this kind of data. Convolution has a key role in classical CNNs, thanks to its properties of locality, stationarity, compositionality, which well match prior knowledge on many kinds of data and thus allow effective weight reuse. For this reason, defining an operation with similar characteristics for graph-structured data is of primary importance in order to obtain effective graph neural networks. The literature has identified two main classes of approaches to tackle this problem, namely spectral or spatial. In the former case [16]–[18], the convolution is defined in the spectral domain through the graph Fourier transform [19]. Fast polynomial approximations [17] have been proposed in order to obtain an efficient implementation of this operation. Graph-convolutional neural networks (GCNN) with this convolution operator have been successfully applied in problems of semi-supervised node classification and link prediction [18], [20]. The main drawback of these methods is that the graph is supposed to be fixed and it is not clear how to handle the cases where the structure varies. The latter class of approaches overcomes this issue by defining the convolution operator in the spatial domain [21]–[26]. In this case, the convolution is performed by local aggregations, i.e. a weighted combination of the signal values over neighboring nodes. Since in this case the operation is defined at a neighborhood level, the convolution remains well-defined even when the graph structure varies. Many of the spatial approaches present in the literature [23]–[25] perform local aggregations with scalar weights. Instead, [21] proposes to weight the contributions of the neighbors using edge-dependent matrices. This makes the convolution a more general function, increasing its descriptive power. For this reason, in this paper we employ the convolution operator proposed in [21]. However, in order to obtain an efficient operation, we introduce several approximations that reduce its computation complexity, memory occupation, and mitigate vanishing gradient issues that arise when trying to build very deep architectures.

B. Image denoising

The literature on image denoising is vast, as it is one of most classic problems in image processing. Focusing on the recent developments, we can broadly define two categories of methods: model-based approaches and learning-based approaches.

Model-based approaches traditionally focused on defining hand-crafted priors to carefully capture the salient features of natural images. Early works in this category include total variation minimization [27], and bilateral filtering [28]. Non-local means [6] introduced the idea of non-local averaging according to the similarity of local neighborhood. Similar approaches have been proposed in [29], [30], where an iterative non-local means procedure has been proposed. The popular BM3D [5] expanded on the idea by collaborative filtering of the matched patches. WNNM [31] used nuclear norm minimization to enforce a low-rank prior. Finally, some works recently introduced graph-based regularizers [32] to enforce a measure of smoothness of the signal across the edges of a graph of patch or pixel similiarities. Many of the most successful model-based approaches are non-local, i.e., they exploit the concept of self-similarity among structures in the image beyond the local neighborhood. This shows the importance of exploiting non-locality in image denoising.

Learning-based approaches use training data to learn a model for natural images. The popular K-SVD algorithm [33] learns a dictionary in which natural patches have a sparse representation, and therefore casts image denoising as a sparse coding problem on this learned dictionary. The TNRD method [34] uses a nonlinear reaction diffusion model with trainable filters. An early work with neural networks [35] used a multilayer perceptron discriminatively trained on synthetic Gaussian noise and showed significant improvements over model-based methods. More recently, CNNs have achieved remarkable performance. Zhang et al. [7] showed that the residual structure and the use of batch normalization [36] in their DnCNN greatly helps the denoising task. Following the DnCNN, many other architectures have been proposed, such as RED [8], MemNet [9] and a CNN working on wavelet coefficients [10]. While the convolutional layers of a CNN constitute an excellent prior for images [37], they are limited by the local nature of the convolution operation, which is unable to increase the receptive field of a neuron-pixel to model non-local image features. This means that CNNs are unable to exploit the self-similar patterns that were proven to be highly successful in model-based methods. Very recently, a few works started addressing this issue by trying to incorporate non-local information in a CNN. NN3D [11] uses a global post-processing stage based on a non-local filter after the output of a denoising CNN. This stage performs block matching and filtering over the whole image denoised by the CNN. This is clearly suboptimal as the non-local information does not contribute to the training of the CNN. UNLNet [12] introduces a trainable non-local layer which collaboratively filters image blocks. However, performance is limited by the selection of matching blocks from the noisy input image instead of the feature space, and ultimately UNLNet does not improve over the performance of the simpler DnCNN. N^3Net [13] introduces a continuous nearest-neighbor relaxation to create a non-local layer. Finally, NLRN [14] proposes a non-local module that uses the distances among hidden feature vectors of a search window around the pixel of interest to aggregate such vectors and return the output features of the pixel. However, there are significant differences with respect to the work in this paper. First, they use all the pixels in the search window instead of only a number of nearest neighbors, which means

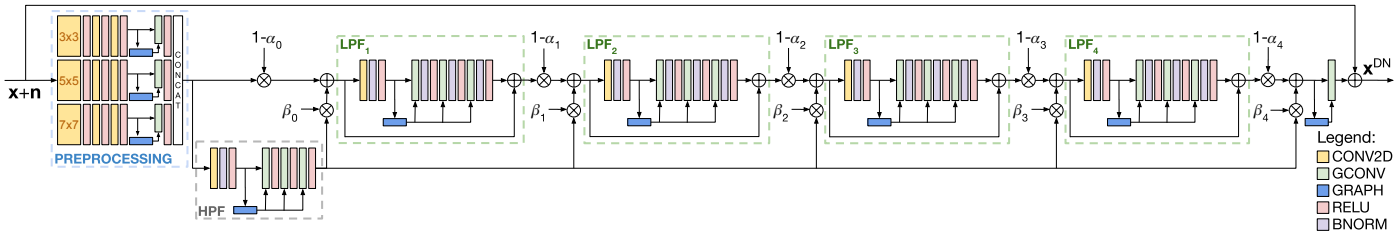


Figure 1. GCDN architecture.

that their receptive field cannot dynamically adapt to the content of the image. Then, while in both works the feature aggregation weights are dynamically computed from the features themselves, NLRN uses an explicitly-parameterized function with learnable parameters, in contrast to this work where the function is fully learned as a dedicated sub-network. These choices increase the adaptivity of the proposed non-local operations, which result in better performance around edges.

III. Proposed denoiser

A. Overview

An overview of the proposed graph-convolutional denoiser network (GCDN) can be seen in Fig. 1. The structure will be explained more in detail in Sec. III-D where an analogy is drawn between unrolled proximal gradient descent with a graph total variation regularizer and the proposed network architecture. At a first glance, the network has a global input-output residual connection whereby the network learns to estimate the noise rather than successively clean the image. This has been shown [7] to improve training convergence for the denoising problem.

The main feature of the proposed network is the use of graph-convolutional layers where the graphs are dynamically computed from the feature space. The graph-convolutional layer, described in Sec. III-B, creates a non-local receptive field for each pixel-neuron, so that pixels that are spatially distant but similar in the feature space created by the network can be merged.

An important block of the proposed network is the preprocessing stage at the input. It can be noticed that the first layers of the network are classic 2D convolutions rather than graph convolutions. This is done to create an embedding over a receptive field larger than a single pixel and stabilize the graph construction operation, which would otherwise be affected by the input noise. The preprocessing stage has three parallel branches that operate on multiple scales, in a fashion similar to the architectures in [38] and [39]. The multiscale features are extracted by a sequence of three convolutional layers with filters of size 3×3, 5×5, and 7×7, depending on the branch. After a final graph-convolutional layer, the features are concatenated.

The remaining network layers are grouped into an HPF block and multiple LPF blocks, named after the analogy with highpass and lowpass graph filters described in Sec. III-D. These blocks have an initial 3×3 convolutional layer followed by three graph-convolutional layers sharing the same graph

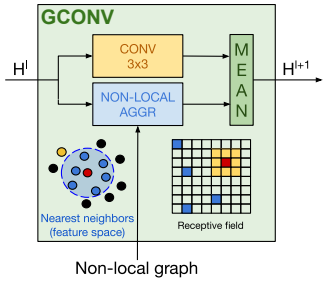


Figure 2. Graph-convolutional layer. The operation has a receptive field with a local component (3×3 2D convolution) and a non-local component (pixels selected as nearest neighbors in the feature space).

constructed from the output of the convolutional layer. All layers are interleaved by Batch Normalization operations [36] and leaky ReLU nonlinearities. Notice that the LPF blocks have themselves a residual connection to help backpropagation, as in ResNet architectures [40]. The final layer is a graph-convolutional layer mapping from feature space to the image space.

B. Graph-convolutional layer

The operation performed by the graph-convolutional layer is summarized in Fig. 2. The two inputs to the graph-convolutional layer are the feature vectors $\mathbf{H}^l \in \mathbb{R}^{F^l \times N}$ associated to the N image pixels at layer l and the adjacency matrix of a graph connecting image pixels. In this work, we use a directed graph without self-loops. The graph is constructed as a K-nearest neighbor graph in the feature space. For each pixel, the Euclidean distances between its feature vector and the feature vectors of pixels inside a search window are computed and an edge is drawn between the pixel and the K pixels with smallest distance. Using this method, we obtain a K-regular graph $\mathcal{G}^l(\mathcal{V}, \mathcal{E}^l)$, where $\mathcal{V}$ is the set of vertices with $|\mathcal{V}| = N$ and $\mathcal{E}^l \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. We also assume that the edges of $\mathcal{G}^l$ are labeled, i.e. there exists a function $\mathcal{L} : \mathcal{E}^l \to \mathbb{R}^{F^l}$ that assigns a label to each edge. In this work, we define the edge labeling function as the difference between the two feature vectors, i.e. $\mathcal{L}(i,j) = \mathbf{H}^l_j - \mathbf{H}^l_i = \mathbf{d}^{l,j \to i}$. A classic 3×3 local convolution processes the local neighborhood to provide its estimate of the output feature vector for the current pixel, while the feature vectors of the non-local pixels connected by the graph are aggregated by means of the edge-conditioned

convolution (ECC) [21]. Notice that the 8 local neighbors of the pixel are excluded from the search of non-local neighbors as they are already used by the local convolution. The non-local aggregation is computed as:

$$\begin{aligned}\mathbf{H}_i^{l+1,\mathrm{NL}} &= \sum_{j\in\mathcal{S}_i^l} \gamma^{l,j\to i} \frac{\mathcal{F}^l_{\mathbf{w}^l}\left(\mathbf{d}^{l,j\to i}\right)\mathbf{H}_j^l}{|\mathcal{S}_i^l|} \\ &= \sum_{j\in\mathcal{S}_i^l} \gamma^{l,j\to i} \frac{\mathbf{\Theta}^{l,j\to i}\mathbf{H}_j^l}{|\mathcal{S}_i^l|}, \end{aligned} \tag{1}$$

where $\mathcal{F}^l_{\mathbf{w}^l} : \mathbb{R}^{F^l} \to \mathbb{R}^{F^{l+1}\times F^l}$ is a fully-connected network that takes as input the edge labels and outputs the corresponding weight matrix $\mathbf{\Theta}^{l,j\to i} = \mathcal{F}^l_{\mathbf{w}^l}\left(\mathcal{L}(i,j)\right) \in \mathbb{R}^{F^{l+1}\times F^l}$, $\mathbf{w}^l$ are the weights parameterizing network $\mathcal{F}^l$, and $\mathcal{S}_i^l$ is the set of neighbors of node i in the graph $\mathcal{G}^l$. The scalar $\gamma^{j\to i}$ is an edge-attention term computed as:

$$\gamma^{l,j\to i} = \exp\left(-\|\mathbf{d}^{l,j\to i}\|_2^2/\delta\right) \tag{2}$$

where δ is a cross-validated hyper-parameter. This term is reminiscent of the edge attention mechanism from the graph neural network literature [41] and it serves the purpose of stabilizing training by underweighting the edges that connect nodes with distant feature vectors. Note that this term could, in principle, be learned by the $\mathcal{F}$ network but we found that decoupling it and making it explicitly dependent on feature distances in an exponential way, accelerated and stabilized training. Also notice that in Sec. IV we show that this term alone, i.e. without weight matrices $\mathbf{\Theta}$, is not powerful enough to reach good performance. Moreover, it is worth mentioning that the edge weights $\mathbf{\Theta}$ and the edge-attention term γ depend only on the edge labels. This means that two pairs of nodes with the same edge labels will have the same weights, resulting in a behaviour similar to weight sharing in classical CNNs.

Finally, we combine the feature vector estimated by the non-local aggregation with the one produced by the local convolution to provide the output features as follows

$$\mathbf{H}_i^{l+1} = \frac{\mathbf{H}_i^{l+1,\mathrm{NL}} + \mathbf{H}_i^{l+1,\mathrm{L}}}{2} + \mathbf{b}^l,$$

where $\mathbf{H}_i^{l+1,\mathrm{L}}$ is the output of the 3×3 local convolution for the node i and $\mathbf{b}^l \in \mathbb{R}^{F^l}$ is the bias.

The advantages of the ECC with respect to other definitions of graph convolution are trifold: i) the edge weights depend on the edge label, ii) it allows to compute an affine transformation along every edge, and iii) the edge weight function is highly general since it does not have a predefined structure. By making the edge weights depend on the input features, the ECC implements an adaptive filter which can be more complex than the non-adaptive local filters. Moreover, the second advantage is due to the fact that $\mathbf{\Theta}^{l,j\to i}$ is an edge-dependent matrix, making the convolution operation more general than other non-local aggregation methods using scalar edge weights. Among such methods we can find GCN [18], GIN [23], MoNet [24], and FeastNet [25]. Finally, the $\mathcal{F}$ function is a general function which can be learned to be the optimal one for the denoising task by the function approximation capability of the subnetwork implementing it. This is in contrast with other methods where the function predicting the edge weights is fixed with some learnable parameters. For example, FeastNet [25] employs scalar edge weights computed using the following function

$$f(\mathbf{H}_i^l, \mathbf{H}_j^l) \propto \exp\left(\mathbf{u}^\intercal \mathbf{H}_i^l + \mathbf{v}^\intercal \mathbf{H}_j^l + c\right),$$

where $\mathbf{u}, \mathbf{v} \in \mathbb{R}^{F^l}$ and $c \in \mathbb{R}$ are learnable parameters. Instead, MoNet [24] employs a Gaussian kernel as follows

$$f(\mathbf{H}_i^l, \mathbf{H}_j^l) = \exp\left(-\frac{1}{2}(\mathbf{d}^{l,j\to i} - \boldsymbol{\mu})^\intercal \mathbf{\Sigma}^{-1}(\mathbf{d}^{l,j\to i} - \boldsymbol{\mu})\right),$$

where $\mathbf{\Sigma} \in \mathbb{R}^{F^l\times F^l}$ and $\boldsymbol{\mu} \in \mathbb{R}^{F^l}$ are learnable parameters. Also NLRN [14] uses a Gaussian kernel to perform non-local aggregations. We can consider this operation as a graph convolution where each pixel is connected to all the other pixels in its search window and the edge weights are defined as follows

$$f(\mathbf{H}_i^l, \mathbf{H}_j^l) = \frac{\exp\left(\mathbf{H}_i^{lT}\mathbf{W}_\theta^\intercal \mathbf{W}_\phi \mathbf{H}_j^l\right)}{\sum_{j\in\mathcal{S}_i}\exp\left(\mathbf{H}_i^{lT}\mathbf{W}_\theta^\intercal \mathbf{W}_\phi \mathbf{H}_j^l\right)}\mathbf{W}_g,$$

where $\mathbf{W}_\theta, \mathbf{W}_\phi \in \mathbb{R}^{t\times F^l}$ and $\mathbf{W}_g \in \mathbb{R}^{F^{l+1}\times F^l}$ are learnable parameters.

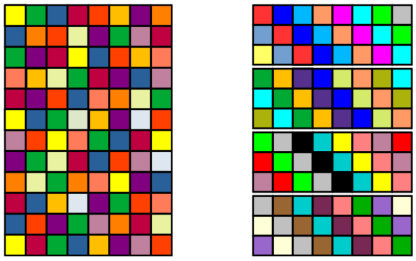


Figure 3. Circulant approximation of a fully-connected layer.

C. Lightweight Edge-Conditioned Convolution

As seen in the previous section, the function $\mathcal{F}$ has a key role in the ECC because it defines the weights for the neighborhood aggregation. In the original definition of ECC [21], the function $\mathcal{F}$ is implemented as a two-layer fully connected network. This definition raises some relevant issues. In the following, we will describe in detail these issues and present two possible solutions.

1) Circulant approximation of dense layer: The first issue is related to the risk of over-parameterization. The dimension of the input of the $\mathcal{F}$ network is F^l, while the dimension of its output is $F^{l+1} \times F^l$. This means that the number of weights of the network depends cubically on the number of features. Therefore, the number of parameters quickly becomes excessively large, resulting in vanishing gradients or overfitting.

To address the over-parameterization problem we propose to use a partially-structured matrix for the last layer, instead of an

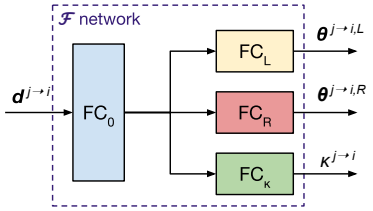


Figure 4. $\mathcal{F}$ network. FC$_0$ is a fully-connected layer followed by a leaky ReLU non-linearity. The FC$_R$, FC$_L$, FC$_\kappa$ do not have any output non-linearities.

unstructured one. We impose that this matrix is composed of multiple stacked partial circulant matrices, i.e., matrices where only a few shifted versions of the first row are used instead of all the possible ones of the full square matrix. Fig. 3 shows the structure of the approximated matrix. Using this approximation, the only free parameters are in the first row of each partial circulant matrix. If only m shifts per partial circulant matrix are allowed, we reduce the number of parameters by a factor m. Thus, if the unstructured dense matrix has $F^l F^{l+1} \times F^l$ parameters, with the proposed approximation the number of parameters drops to $\frac{F^l F^{l+1}}{m} \times F^l$. Similar approaches to approximate fully connected layers have already been studied in the literature [42], [43]. In particular, [42] shows that imposing a partial circulant structure does not significantly impact the final performance in a classification problem. Indeed, there are connections with results stating that random partial circulant matrices implement stable embeddings almost as well as fully random matrices [44], [45].

2) Low-rank node aggregation: The second issue related to the $\mathcal{F}$ network regards memory occupation and computations. In order to perform the ECC operation, we have to compute a weight matrix $\boldsymbol{\Theta}^{l,j\to i}$ for each edge j of every neighborhood $\mathcal{N}_i$ of every image in the batch. If we consider a K-regular graph and a batch of B images with N pixels each, the memory occupation needed to store all the matrices $\boldsymbol{\Theta}^{l,j\to i}$ as single-precision floating point tensors is equal to $B \times N \times K \times F^{l+1} \times F^l \times 4$ bytes and this quantity can easily become unmanageable. To give an idea of the required amount of memory, let us consider an example with $B = 16$, $N = 1024$, $K = 8$, $F^l = F^{l+1} = 66$, then the memory required to store all the matrices $\boldsymbol{\Theta}^{l,j\to i}$ for only one graph-convolutional layer is around 2 GB.

In order to solve this issue, we propose to impose a low-rank approximation for $\boldsymbol{\Theta}^{l,j\to i}$. Let us consider the singular value decomposition of a matrix

$$\mathbf{A} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Psi}^\intercal = \sum_s \lambda_s \boldsymbol{\phi}_s \boldsymbol{\psi}_s^\intercal,$$

where $\boldsymbol{\phi}_s$ and $\boldsymbol{\psi}_s$ are the left and right singular vectors and λ_s the singular values. We can obtain a low-rank approximation of rank r by keeping only the r largest singular values and setting the others to zero. Therefore, the approximation is reduced to a sum of r outer products. Inspired by this fact, we define $\boldsymbol{\Theta}^{l,j\to i}$ as follows

$$\boldsymbol{\Theta}^{l,j\to i} = \sum_{s=1}^{r} \kappa_s^{j\to i} \boldsymbol{\theta}_s^{j\to i,L} \boldsymbol{\theta}_s^{j\to i,R^\intercal}, \tag{3}$$

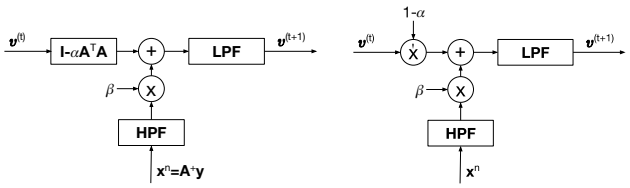
where $\boldsymbol{\theta}_s^{j\to i,L} \in \mathbb{R}^{F^l}$, $\boldsymbol{\theta}_s^{j\to i,R} \in \mathbb{R}^{F^{l+1}}$, $\kappa_s^{j\to i} \in \mathbb{R}$ and $1 \leq r \leq F^l$. Notice that the approximation in (3) ensures that the rank is at most r rather than exactly enforcing a rank-r structure, because we do not impose orthogonality between $\boldsymbol{\theta}_s^{j\to i,L}$ and $\boldsymbol{\theta}_s^{j\to i,R}$, even though random initialization makes them quasi-orthogonal. Using this approximation, we can redefine the $\mathcal{F}$ network in such a way that it outputs $\boldsymbol{\theta}_s^{j\to i,L}, \boldsymbol{\theta}_s^{j\to i,R}, \kappa_s^{j\to i}$ for $s = 1, 2, \ldots, r$. In particular, we redefine the second layer of the $\mathcal{F}$ network: instead of having a single fully connected layer that outputs the entire matrix $\boldsymbol{\Theta}^{l,j\to i}$, we have three parallel fully connected layers that separately output $\boldsymbol{\theta}_s^{j\to i,L}$, $\boldsymbol{\theta}_s^{j\to i,R}$ and $\kappa_s^{j\to i}$, as shown in Fig. 4. The advantage of this approximation is that we only need to store $\boldsymbol{\theta}_s^{j\to i,L}$, $\boldsymbol{\theta}_s^{j\to i,R}$ and $\kappa_s^{j\to i}$ instead of the entire matrix $\boldsymbol{\Theta}^{l,j\to i}$, drastically reducing the memory occupation to $B \times N \times K \times r(F^l + F^{l+1} + 1) \times 4$ bytes. If we consider the example presented above and set $r = 10$, the memory requirement drops from 2 GB to 700 MB. Another advantage of this approximation is that it also leads to a significant reduction of the computation burden, because we never have to actually compute all the matrices $\boldsymbol{\Theta}^{l,j\to i}$. In fact, the neighborhood aggregation can be reduced as follows

$$\begin{aligned}\mathbf{H}_i^{l+1,\mathrm{NL}} &= \sum_{j\in\mathcal{S}_i^l} \gamma^{l,j\to i} \frac{\boldsymbol{\Theta}^{l,j\to i}\mathbf{H}_j^l}{|\mathcal{S}_i^l|} \\ &= \sum_{j\in\mathcal{S}_i^l} \gamma^{l,j\to i} \frac{\sum_{s=1}^r \kappa_s^{j\to i} \boldsymbol{\theta}_s^{j\to i,L} \boldsymbol{\theta}_s^{j\to i,R^\intercal} \mathbf{H}_j^l}{|\mathcal{S}_i^l|},\end{aligned} \tag{4}$$

where the computational cost of the full operation on the first line is $O(F^l F^{l+1})$, instead the cost of the decoupled operation on the second line is $O(r(F^l + F^{l+1}))$. Finally, this approximation also helps to reduce the number of parameters of the last layer of the $\mathcal{F}$ network since the output has size $r(F^l + F^{l+1} + 1)$ instead of $F^{l+1}F^l$.

When we employ the new structure of the $\mathcal{F}$ network, we need to pay special attention to the weight initialization. In particular, we have to carefully define the variance of the random weight initialization of the three parallel layers to avoid scaling problems. We define $\mathbf{W}_0$ as the weight matrix of the first layer of the $\mathcal{F}$ network, and $\mathbf{W}^L$, $\mathbf{W}^R$ and $\mathbf{W}^\kappa$ as the weight matrices of the three parallel fully connected layers. Let us suppose that $\mathbf{W}^0$ has been initialized using Glorot initialization [46], i.e., $\mathbf{W}^0_{u,v} \sim \mathcal{N}\left(0, \frac{1}{F^l}\right)$ with $u, v = 1, \ldots, F^l$. Let us also assume that $\mathbf{W}^L_{u,v} \sim \mathcal{N}(0, \sigma_L^2)$, $\mathbf{W}^R_{u,v} \sim \mathcal{N}(0, \sigma_R^2)$, and $\mathbf{W}^\kappa_u \sim \mathcal{N}(0, \sigma_\kappa^2)$. We first normalize the input of the $\mathcal{F}$ network, i.e., $\|\mathbf{d}^{l,j\to i}\|_2^2 = F^l$. Then, we obtain

$$\begin{aligned}\mathbb{E}[\boldsymbol{\theta}_{s,u}^{j\to i,L}] &= 0, \quad \mathrm{Var}[\boldsymbol{\theta}_{s,u}^{j\to i,L}] = F^l\sigma_L^2; \\ \mathbb{E}[\boldsymbol{\theta}_{s,u}^{j\to i,R}] &= 0, \quad \mathrm{Var}[\boldsymbol{\theta}_{s,u}^{j\to i,R}] = F^l\sigma_R^2; \\ \mathbb{E}[\kappa_s^{j\to i}] &= 0, \quad \mathrm{Var}[\kappa_s^{j\to i}] = F^l\sigma_\kappa^2;\end{aligned}$$



(a) Linear inverse problem (b) Denoising

Figure 5. Single iteration. LPF is graph lowpass filter, HPF is a graph highpass filter.

where $s = 1, \ldots, r$. Finally, the aggregation formula in Eq. (4) leads to the following result:

$$\mathbb{E}\left[\mathbf{H}_{i,u}^{l+1,\mathrm{NL}}\right] = 0, \quad \mathrm{Var}\left[\mathbf{H}_{i,u}^{l+1,\mathrm{NL}}\right] = \frac{1}{2} r F^{l^4} \sigma_L^2 \sigma_R^2 \sigma_\kappa^2, \quad (5)$$

with $u = 1, \ldots, F^{l+1}$. In Eq. (5), we can observe that the variance of $\mathbf{H}_{i,u}^{l+1,\mathrm{NL}}$ depends on the fourth power of the number of features. This term can easily become extremely large, therefore it is important to set σ_L^2, σ_R^2 and σ_κ^2 in such a way that they can balance it. In this work, we set $\sigma_L^2 = \sigma_R^2 = \frac{1}{F^{l^2}}$ and $\sigma_\kappa^2 = \frac{2}{r}$. This allows us to obtain $\mathrm{Var}\left[\mathbf{H}_{i,u}^{l+1,\mathrm{NL}}\right] = 1$ with $u = 1, \ldots, F^{l+1}$.

D. Analogy with unrolled graph smoothness optimization

The neural network architecture presented in Sec. III-A can be seen as a generalization of few iterations of an unrolled proximal gradient descent optimization method, which is widely used to solve linear inverse problems in the form of

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{n} \quad (6)$$

being $\mathbf{x}$ the clean image, $\mathbf{A}$ a forward model (e.g., a degradation such as blurring, downsampling, compressed sensing, etc.) and $\mathbf{n}$ a noise term. A well-known technique to recover $\mathbf{x}$ from $\mathbf{y}$ is to cast the problem as a least-squares minimization problem with a regularization term that models some prior knowledge about the image. One such regularizer is graph smoothness. Considering a graph with Laplacian matrix $\mathbf{L}$ where edges connect pixels that are deemed correlated according to some criterion, the graph smoothness $\mathbf{x}^\intercal\mathbf{L}\mathbf{x}$ is the graph equivalent of the total variation measure, indicating how much $\mathbf{x}$ varies across the edges of the graph. Natural images where the graph connects the local neighborhood typically have lowpass behavior, resulting in a low graph smoothness value. Reconstruction is therefore cast as:

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\mathrm{argmin}} \left[\frac{1}{2}\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 + \frac{\beta}{2}\mathbf{x}^\intercal\mathbf{L}\mathbf{x}\right] \quad (7)$$

The functional in Eq. (7) is in the form of a sum of two terms $(f(\mathbf{x}) + g(\mathbf{x}))$ and can be minimized by means of proximal gradient descent [47] which alternates a gradient descent step over f and a proximal mapping over g:

$$\begin{aligned}\mathbf{x}^{(t+1)} &= \mathrm{prox}_g\left(\mathbf{x}^{(t)} - \alpha\nabla_{\boldsymbol{\nu}} f\right) \\ &= \mathrm{prox}_g\left((\mathbf{I} - \alpha\mathbf{A}^\intercal\mathbf{A})\mathbf{x}^{(t)} + \alpha\mathbf{A}^\intercal\mathbf{y}\right) \\ \mathrm{prox}_g(\boldsymbol{\mu}) &= \underset{\mathbf{z}}{\mathrm{argmin}}\left[\|\mathbf{z} - \boldsymbol{\mu}\|_2^2 + \frac{\beta}{2}\mathbf{z}^\intercal\mathbf{L}\mathbf{z}\right].\end{aligned}$$

Solving for the proximal mapping operator results in the following update equation:

$$\mathbf{x}^{(t+1)} = (\mathbf{I} + \beta\mathbf{L})^{-1}\left[(\mathbf{I} - \alpha\mathbf{A}^\intercal\mathbf{A})\mathbf{x}^{(t)} + \alpha\mathbf{A}^\intercal\mathbf{y}\right]. \quad (8)$$

In order to match the framework of residual networks, let us define the least-squares solution $\mathbf{x}^n = \mathbf{A}^+\mathbf{y} = (\mathbf{A}^\intercal\mathbf{A})^{-1}\mathbf{A}^\intercal\mathbf{y}$ and perform a change of variable whereby the optimization estimates the residual of the least squares solution, i.e., $\boldsymbol{\nu}^{(t)} = \mathbf{x}^n - \mathbf{x}^{(t)}$. Hence, we can rewrite Eq. (8) as:

$$\begin{aligned}&\mathbf{x}^n - \boldsymbol{\nu}^{(t+1)} = \\ &(\mathbf{I} + \beta\mathbf{L})^{-1}\left[(\mathbf{I} - \alpha\mathbf{A}^\intercal\mathbf{A})\left(\mathbf{x}^n - \boldsymbol{\nu}^{(t)}\right) + \alpha\mathbf{A}^\intercal\mathbf{y}\right].\end{aligned}$$

Finally, the following update equation can be derived:

$$\boldsymbol{\nu}^{(t+1)} = (\mathbf{I} + \beta\mathbf{L})^{-1}\left[(\mathbf{I} - \alpha\mathbf{A}^\intercal\mathbf{A})\,\boldsymbol{\nu}^{(t)} + \beta\mathbf{L}\mathbf{x}^n\right]. \quad (9)$$

This update can be visualized as in Fig. 5a and is composed of two major operations involving the signal prior:

1) $\mathbf{L}\mathbf{x}^n$: the graph Laplacian can be seen as a graph highpass filter applied to $\mathbf{x}^n$;
2) $(\mathbf{I} + \beta\mathbf{L})^{-1}$: this term can be seen as a graph lowpass filter. In order to see this, let us use the matrix inversion lemma as $(\mathbf{I} + \beta\mathbf{L})^{-1} = \left(\mathbf{I} + \beta\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^H\right)^{-1} = \mathbf{I} - \mathbf{U}\left(\beta^{-1}\boldsymbol{\Lambda}^{-1} + \mathbf{I}\right)^{-1}\mathbf{U}^H = \mathbf{U}\left[\mathbf{I} - \left(\beta^{-1}\boldsymbol{\Lambda}^{-1} + \mathbf{I}\right)^{-1}\right]\mathbf{U}^H$, where $\mathbf{U}$ is the graph Fourier transform. The term $\mathbf{I} - \left(\beta^{-1}\boldsymbol{\Lambda}^{-1} + \mathbf{I}\right)^{-1}$ is a diagonal matrix whose entries are equal to $\frac{1}{\beta\lambda_i + 1}$ where λ_i are the eigenvalues of the graph Laplacian, and the lowpass behavior is due to decreasing value of such entries for increasing λ.

For the denoising problem, we can set $\mathbf{A} = \mathbf{I}$ and obtain the update shown in Fig.5b. The network architecture proposed in Sec. III-A draws from this derivation by unrolling a finite number of Eq. (9) iterations and generalizing the lowpass and highpass filters with learned graph filters interleaved by nonlinearities.

We remark that this is not the only possible analogy between unrolled iterative model-based methods and neural network architectures. For instance, alternative analogies could be drawn from iterative non-local means approaches [48]. However, all analogies are limited by the fact that the nonlinearities between network layers and the training process can in principle make the network learn any arbitrary function without bearing similarity to the original algorithm inspiring the architecture. Nevertheless, it is interesting to notice that in Sec. IV we experimentally show that the learned filters actually show an approximate highpass and lowpass behavior, following the predictions of our analogy.

IV. Experimental Results

A. Training details

The training protocol follows the one used in [7]. The network is trained with patches of size 42×42 randomly extracted from 400 images from the train and test partitions of the Berkeley Segmentation Dataset (BSD) [49], withholding the 68 images in the validation set for testing purposes (BSD68 dataset). The loss function is the mean squared error (MSE) between the denoised patch output by the network and the ground truth. Each model is trained for approximately 800000 iterations with a batch size of 8. The Adam optimizer [50] has been used with an exponentially decaying learning rate between 10^{-4} and 10^{-5}. The behavior of the graph-convolutional layer is slightly different between training and testing for efficiency reasons. During training all pairwise distances are computed among the feature vectors corresponding to the pixels in the patch. On the other hand, testing is "fully convolutional", as every pixel has a search window centered around it and neighbors are identified as the closest pixels in such search window. The search window size is 43×43, roughly comparable to the patch size used in training. This procedure is slightly suboptimal as some pixels might suffer from border effects during training (their search windows are not centered around them) but it is advantageous in terms of speed and memory requirements. Reflection padding is used for all 2D convolutions to avoid border effects. The δ parameter in the edge attention term in Eq. (2) is set to a value equal to 10, after a few experiments showing no significant differences among tested values on the validation data. The number of features used in all convolutional layers is 132, except for the three parallel branches of the preprocessing stage which have 44 features. The number of circulant rows in the circulant approximation of dense layers in the $\mathcal{F}$ network is $m = 3$. The low-rank approximation uses $r = 11$ terms. During training, we noticed that the proposed lightweight ECC presented in Sec. III-C is extremely useful. In fact, without it, the network suffered from vanishing gradient problems even with a significantly lower number of layers.

B. Feature analysis

In this section we study the properties of the features in the hidden layers of the network.

1) Adaptive receptive field: We first analyze the characteristics of the receptive field of a single pixel. The receptive field of a pixel at the output of a graph-convolutional layer with respect to its input is defined as the set of pixels used in the computation of its output feature vector. Observing Fig. 2 we can see it includes the pixel itself and the 8 local neighbors due to the local 3×3 convolution as well as the K non-local neighbors identified by the graph construction. In Fig. 6 we show two examples of the receptive field of a single pixel at the output of each graph-convolutional layer in an LPF block with respect to the input of the block. Since the proposed network employs multiple graph-convolutional layers, the receptive field does not only expand radially as in classical CNNs, but its shape depends on the structure of the graph. Instead, in Fig. 7 we show the receptive field of a single pixel at the output of each of the layers in the HPF block and in the first LPF block with respect to the output of the preprocessing block. We can clearly see that the receptive field is adapted to the characteristics of the image: if we consider a pixel in a uniform area, its receptive field will mostly contain pixels that belong to similar regions; instead if we consider a pixel on an edge, its receptive field will be mainly composed of other edge pixels. This is beneficial to the denoising task as it allows to exploit self-similarity and it descends from the use of a nearest neighbor graph, connecting each pixel to other pixels with similar features. Notice that differently from algorithms performing block matching in the pixel space, we compute distances between feature vectors which can capture more complex image characteristics. This can be seen in Fig. 8 where we compute the Euclidean distances between the feature vector of the central pixel and the feature vectors of the other pixels in the search window. We notice that the distances reflect the type of edge that includes the central pixel, e.g., a pixel sitting on a horizontal edge will detect as closest other pixels sitting on horizontal edges. This is due to the visual features learned by the network and would not happen in pixel-space matching. Thanks to the adaptability of the receptive field, graph convolution can be interpreted as a generalization of the block matching operation performed in other non-local denoising methods, such as BM3D [5].

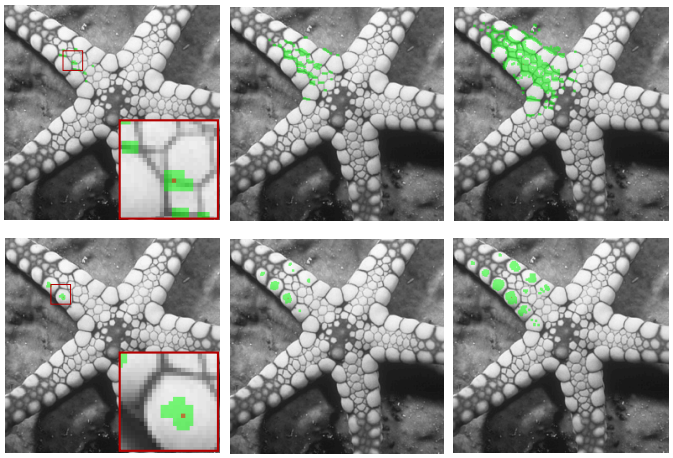


Figure 6. Receptive field (green) of a single pixel (red) at the output of the three graph-convolutional layers in the LPF_1 block with respect to the input of the first graph-convolutional layer in the block. Top row: gray pixel on an edge. Bottom row: white pixel in a uniform area.

2) Filter analysis: We also study the behavior of the LPF and HPF operators. In particular, we are interested in validating the analogy made in Sec. III-D. We compute the discrete Fourier transform (DFT) of the feature maps at the output of these operators. As an example, Fig. 9 shows the log-magnitude of the coefficients of three feature maps at the output of the HPF block and of the first LPF block. The energy of the DFT coefficients of the LPF feature maps is concentrated in the low frequencies, thus showing a lowpass behavior. Instead, the coefficients of the HPF feature maps show a typical highpass behavior, having the energy concentrated

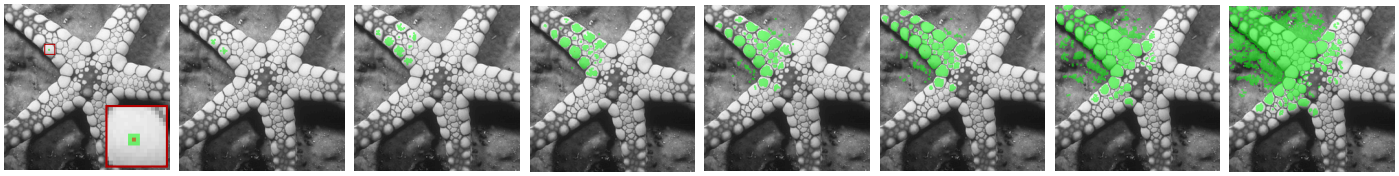


Figure 7. Receptive field (green) of a single pixel (red) at the output of each layer (convolutional or graph-convolutional) in the HPF and LPF_1 blocks (left to right, in the same order as a forward pass), with respect to the output of the preprocessing block.

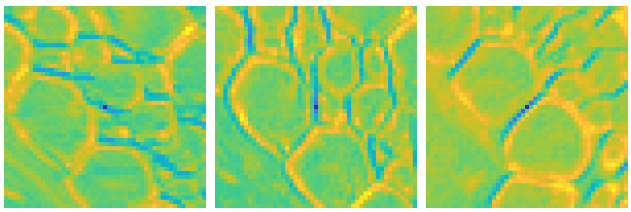


Figure 8. Euclidean distances between feature vectors of the central pixel and all the pixels in the search window (input of first graph-convolutional layer of LPF_1). Left to right: pixel on a horizontal edge, pixel on a vertical edge, pixel on a diagonal edge. Blue represents lower distance.

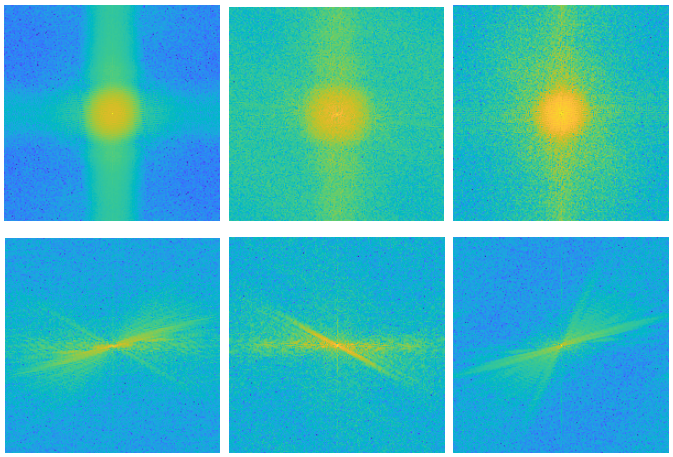


Figure 9. Log-magnitude of discrete Fourier transform of three feature maps at the output of the LPF_1 block (top) and HPF block (bottom). Blue is lower magnitude.

along few directions. This substantiates our claim that the learned convolutional layers actually approximate nonlinear highpass and lowpass operators.

3) Edge prediction: Lastly, we measure how much the true graph constructed by pixel or patch similarities on the noiseless image is successfully predicted by the graph constructed from the feature vectors in the hidden layers when a noisy input is used. In order to construct the true graph of the image, we first compute the average pixel value of a 5×5 window centered at the considered pixel, for every pixel in the image, and then we use the obtained values to compute a nearest neighbor graph with Euclidean distances. We then compare the true graph with the graph computed in the hidden layers of the network for a noisy input. Fig. 10 shows the percentage of edges of the feature-space graph also found in the true graph,

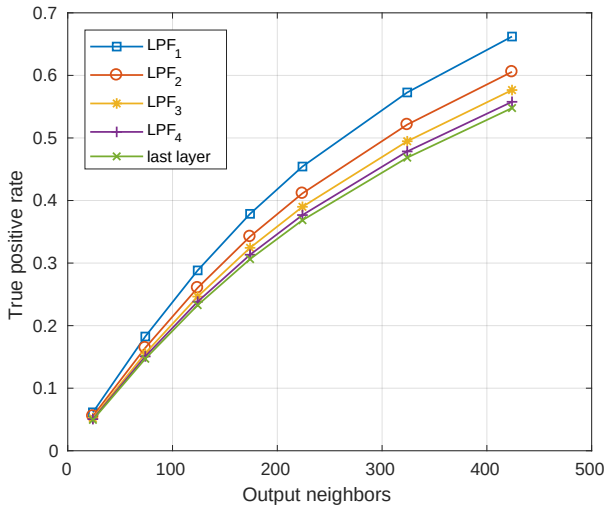


Figure 10. Accuracy of edge prediction from hidden layers ($\sigma = 25$).

as a function of the number of neighbors considered for the true graph. We can notice that the accuracy of the prediction decreases in layers closer to the output. This is due to the fact that we use a residual network that estimates the noise instead of approximating the clean image. In fact, the network learns to successively remove the latent correlations in the feature space, and as a consequence, the graph becomes more random in the later layers.

C. Ablation studies

We study the impact of various design parameters on denoising performance. First, Table I shows the PSNR on the Set12 testing set as function of the number of neighbors used by the graph convolution operation for several values of the noise standard deviation σ. Each model has been independently trained for the specified number of neighbors. It can be noticed that increasing the number of neighbors improves the denoising performance up to a saturation point, and then the performance slightly decreases. This shows that there an optimal neighborhood size and that it is important to employ only a small number of neighbors, in order to select only pixels with similar characteristics. This is in contrast with the NLRN method which uses all the pixels in the search window.

Then, we study the relative impact on performance of the edge aggregation matrices $\boldsymbol{\Theta}$ in Eq. (1) with respect to using

Table I. PSNR (DB) V. NON-LOCAL NEIGHBORHOOD SIZE (SET12)

σ	0-NN	4-NN	8-NN	12-NN	16-NN	20-NN
15	32.91	33.09	33.11	33.13	33.14	33.13
25	30.50	30.70	30.74	30.75	30.78	30.78
50	27.28	27.52	27.58	27.58	27.60	27.59

Table II. EDGE ATTENTION V. ECC + EDGE ATTENTION (8-NN). PSNR (DB).

σ	Edge attention only	Proposed
25	30.53	30.74

only the edge attention scalar γ. Table II reports the PSNR achieved on Set12 by the proposed method with the non-local aggregation performed as in Eq. (1) and a variant where the aggregation is computed as:

$$\mathbf{H}_i^{l+1,\mathrm{NL}} = \sum_{j \in \mathcal{S}_i^l} \gamma^{j \to i} \mathbf{H}_j^l.$$

Both methods use a non-local graph with 8 nearest neighbors. We can notice that the edge attention term alone achieves a worse PSNR with respect to GCDN by approximately 0.2 dB, even though it improves over a model without non-local neighbors (see Table I for the corresponding 0-NN value). This shows the advantage of using a trainable affine transformation, such as $\boldsymbol{\Theta}$ in Eq. (1), instead of a scalar weight function with a predefined structure.

Finally, we remark that we do not compare with respect to the full ECC without the approximations introduced in Sec. III-C because it suffers from vanishing gradient problems, rendering training unstable even for a much smaller number of layers, and it would be computationally prohibitive.

D. Comparison with state of the art

In this section we compare the proposed network with state-of-the-art models for the Gaussian denoising task of grayscale images. We train an independent model for each noise standard deviation, which is assumed to be known a priori for all methods. We fix the number of neighbors for the proposed method to 16. The reference methods can be classified into model-based algorithms such as BM3D [5], WNNM [31], TNRD [34] and recent deep-learning methods such as DnCNN [7], N^{3}Net [13] and NLRN [14]. In particular, among the deep-learning methods, N^{3}Net and NLRN propose non-local approaches. All results have been obtained running the pre-trained models provided by the authors, except for N^{3}Net at $\sigma = 15$ which is unavailable. Table III reports the PSNR and SSIM values obtained for the Set12, BSD68 and Urban100 standard test sets. It can be seen that the proposed method achieves state-of-the art performance and works especially well at low to medium levels of noise. This can be explained by a higher difficulty in constructing a meaningful graph from the noisy image at higher noise levels. We also notice that the proposed method achieves strong results on the Urban dataset. This dataset contains higher resolution images with respect to the other two and is mainly composed of photos of buildings and other regular structures where exploiting self-similarity is very important. In addition, it is also worth mentioning that the proposed method provides a better visual quality. In many cases, the proposed method has a higher SSIM score, even if NRLN has better performance in terms of PSNR. This can also be noticed in Fig. 11, which shows a visual comparison on an image from the Urban100 dataset. In general, the images produced by the proposed algorithm present sharper edges and smoother content in uniform areas. We can notice that many areas in the photos from Urban100 have approximately piecewise smooth characteristics. It is well known that image processing algorithms based on graphs are well suited for piecewise smooth content (see, e.g., [51], [52] in the context of compression and [32] for denoising). To further show this point, we study the performance of the proposed method for denoising of depth maps, e.g., generated by time-of-flight cameras. The OGLR algorithm [32] based on a graph smoothness regularizer achieved state-of-the-art results among model-based algorithms for this specific task where it is essential to preserve edge sharpness while simultaneously smoothing the flat areas. Table IV reports the PSNR and SSIM results achieved on a standard set of depth maps[1]. It can be seen that the proposed method outperforms both NLRN and OGLR, even at high levels of noise. Also, we can notice that OGLR displays competitive performance at low noise levels, but its visual quality significantly degrades when in presence of stronger noise. Fig. 12 shows a visual comparison where it can be seen that GCDN produces sharper edges while also providing a very smooth background.

E. Real image denoising

Real image noise is generally more challenging than synthetic Gaussian noise. There are multiple contributions such as quantization noise, shot noise, fixed-pattern noise [1], [53], dark current, etc. that make it overall signal-dependent. It has been observed [54]–[56] that deep learning methods trained on synthetic Gaussian noise perform poorly in presence of real noise. However, suitable retraining with real data generally improves their performance. Zeng et al. [56] propose to introduce a regularizer based on the graph Laplacian in the deep learning framework in order to achieve robust real image denoising. In this section, we study the behavior of the proposed network in a blind denoising setting with real noisy images acquired by smartphones. We retrain the proposed method, NLRN and DnCNN on the SIDD dataset [55] composed of 30000 high-resolution images acquired by smartphone cameras at varying illumination and ISO levels. The authors provide clean and carefully registered ground truths for all the available scenes, so that it is possible to perform a supervised training. We create training and testing subsets from the sRGB images in the SIDD dataset by selecting a range of noise levels. Our training set is composed of 3500 crops of size 512×512 whose RMSE with respect to the ground truth is below 15. The testing set is composed of 25 random crops of size 512×512 with noise in the same range as the training set. Table V reports the results for CBM3D [57], DnCNN, NLRN and the proposed GCDN. Notice that CBM3D is not a blind method, so we provide

[1] http://vision.middlebury.edu/stereo/data/.

Table III. NATURAL IMAGE DENOISING RESULTS. METRICS ARE PNSR (DB) AND SSIM.

Dataset	Noise σ	BM3D [5]	WNNM [31]	TNRD [34]	DnCNN [7]	N^3Net [13]	NLRN [14]	**GCDN**
Set12	15	32.37 / 0.8952	32.70 / 0.8982	32.50 / 0.8958	32.86 / 0.9031	- / -	**33.16** / 0.9070 ± 1.06/0.0222	33.14 / **0.9072** ± 1.05/0.0214
	25	29.97 / 0.8504	30.28 / 0.8557	30.06 / 0.8512	30.44 / 0.8622	30.55 / -	**30.80** / **0.8689** ± 1.19/0.0305	30.78 / 0.8687 ± 1.20/0.0295
	50	26.72 / 0.7676	27.05 / 0.7775	26.81 / 0.7680	27.18 / 0.7829	27.43 / -	**27.64** / **0.7980** ± 1.31/0.0430	27.60 / 0.7957 ± 1.33/0.0443
BSD68	15	31.07 / 0.8717	31.37 / 0.8766	31.42 / 0.8769	31.73 / 0.8907	- / -	**31.88** / 0.8932 ± 2.48/0.0380	31.83 / **0.8933** ± 2.47/0.0383
	25	28.57 / 0.8013	28.83 / 0.8087	28.92 / 0.8093	29.23 / 0.8278	29.30 / -	**29.41** / 0.8331 ± 2.63/0.0564	29.35 / **0.8332** ± 2.66/0.0570
	50	25.62 / 0.6864	25.87 / 0.6982	25.97 / 0.6994	26.23 / 0.7189	26.39 / -	**26.47** / 0.7298 ± 2.78/0.0882	26.38 / **0.7389** ± 2.70/0.0877
Urban100	15	32.35 / 0.9220	32.97 / 0.9271	31.86 / 0.9031	32.68 / 0.9255	- / -	33.42 / 0.9348 ± 2.56/0.0324	**33.47** / **0.9358** ± 2.58/0.0326
	25	29.70 / 0.8777	30.39 / 0.8885	29.25 / 0.8473	29.97 / 0.8797	30.19 / -	30.88 / 0.9003 ± 2.63/0.0460	**30.95** / **0.9020** ± 2.65/0.0476
	50	25.95 / 0.7791	26.83 / 0.8047	25.88 / 0.7563	26.28 / 0.7874	26.82 / -	27.40 / **0.8244** ± 2.52/0.0711	**27.41** / 0.8160 ± 2.53/0.0758

Table IV. DEPTH MAP DENOISING RESULTS. METRICS ARE PNSR (DB) AND SSIM.

σ	Method	*aloe*	*art*	*baby*	*cones*	*dolls*	*laundry*	*moebius*	*reindeer*	Average
15	GCDN	40.74 / **0.9873**	40.66 / **0.9886**	41.64 / **0.9917**	39.29 / **0.9832**	**40.70 / 0.9830**	**41.97 / 0.9842**	**42.07 / 0.9877**	**42.62 / 0.9915**	**41.21 / 0.9872** ± 1.06/0.0035
	NLRN	40.50 / 0.9844	40.48 / 0.9858	**41.76** / 0.9899	39.50 / 0.9814	40.69 / 0.9800	41.96 / 0.9814	42.01 / 0.9848	42.44 / 0.9880	41.17 / 0.9845 ± 1.02/0.0034
	OGLR	**40.82** / 0.9801	**40.77** / 0.9821	40.90 / 0.9806	**39.65** / 0.9774	40.41 / 0.9756	41.32 / 0.9764	41.48 / 0.9793	41.72 / 0.9823	40.88 / 0.9792 ± 0.66/0.0025
25	GCDN	**37.12 / 0.9771**	**37.15 / 0.9788**	**37.50 / 0.9814**	35.88 / **0.9697**	**37.05 / 0.9705**	**38.62 / 0.9730**	**38.39 / 0.9786**	**38.80 / 0.9836**	**37.56 / 0.9766** ± 0.98/0.0051
	NLRN	37.08 / 0.9720	37.01 / 0.9734	37.37 / 0.9797	**36.09** / 0.9661	37.01 / 0.9646	38.42 / 0.9679	38.33 / 0.9723	38.65 / 0.9786	37.50 / 0.9718 ± 0.89/0.0055
	OGLR	36.67 / 0.9592	36.68 / 0.9649	36.29 / 0.9594	35.51 / 0.9545	36.41 / 0.9541	37.44 / 0.9541	37.17 / 0.9575	37.86 / 0.9655	36.75 / 0.9587 ± 0.73/0.0046
50	GCDN	**33.37 / 0.9522**	**33.18 / 0.9536**	32.23 / 0.9468	**31.61 / 0.9379**	32.37 / **0.9417**	34.07 / **0.9526**	**33.73 / 0.9567**	34.35 / **0.9672**	**33.11 / 0.9511** ± 0.96/0.0101
	NLRN	33.23 / 0.9444	32.86 / 0.9448	**32.42 / 0.9534**	31.53 / 0.9304	**32.40** / 0.9347	**34.15** / 0.9459	33.58 / 0.9475	**34.37** / 0.9603	33.07 / 0.9452 ± 0.96/0.0095
	OGLR	32.24 / 0.9121	31.92 / 0.9129	31.23 / 0.9027	30.21 / 0.8926	31.44 / 0.8999	32.85 / 0.9051	32.46 / 0.9093	32.99 / 0.9191	31.92 / 0.9067 ± 0.93/0.0084

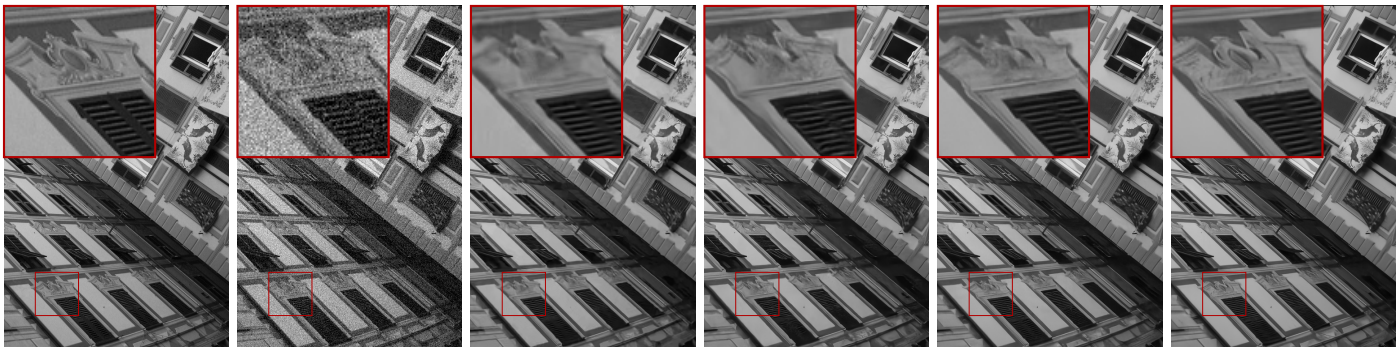


Figure 11. Extract from Urban100 scene 13, $\sigma = 25$. Left to right: ground truth, noisy (20.16 dB), BM3D (30.40 dB), DnCNN (30.71 dB), NLRN (31.41 dB), GCDN (**31.53 dB**).

Table V. REAL IMAGE DENOISING (SIDD DATASET)

	CBM3D	DnCNN	NLRN	GCDN
PSNR	38.73 ± 2.95 dB	39.98 ± 3.17 dB	41.24 ± 2.64 dB	**41.48** ± 2.15 dB
SSIM	0.9587 ± 0.0138	0.9605 ± 0.0158	0.9652 ± 0.0144	**0.9697** ± 0.0132

an estimate of the noise standard deviation, as computed by a noise estimation algorithm [58]. We can notice that the proposed method achieves better results and this is confirmed by the visual comparison in Fig. 13.

V. CONCLUSIONS

In this paper, we presented a graph-convolutional neural network targeted for image denoising. The proposed graph-convolutional layer allows to exploit both local and non-local similarities, resulting in an adaptive receptive field. We showed that the proposed architecture can outperform state-of-the-art denoising methods, achieving very strong results on piecewise smooth images. Finally, we have also considered a real image denoising setting, showing that the proposed method can provide a significant performance gain. Future work will focus on extending the proposed architecture to other inverse problems, such as super-resolution [59], [60].

REFERENCES

[1] J. Lukáš, J. Fridrich, and M. Goljan, "Digital camera identification from sensor pattern noise," *IEEE Transactions on Information Forensics and Security*, vol. 1, no. 2, pp. 205–214, 2006.

[2] D. Valsesia, G. Coluccia, T. Bianchi, and E. Magli, "Compressed fingerprint matching and camera identification via random projections," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 7, pp. 1472–1485, July 2015.

[3] Y. Romano, M. Elad, and P. Milanfar, "The little engine that could: Regularization by denoising (red)," *SIAM Journal on Imaging Sciences*, vol. 10, no. 4, pp. 1804–1844, 2017.

[4] Y. Sun, J. Liu, and U. Kamilov, "Block coordinate regularization by denoising," in *Advances in Neural Information Processing Systems*, 2019, pp. 382–392.

[5] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-D transform-domain collaborative filtering," *IEEE Transactions on Image Processing*, vol. 16, no. 8, pp. 2080–2095, 2007.

[6] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2005, pp. 60–65.

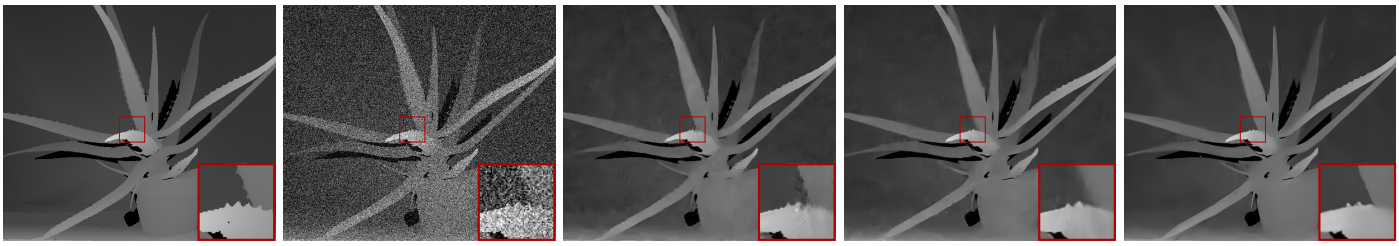


Figure 12. *aloe* depthmap denoising, $\sigma = 50$. Left to right: ground truth, noisy (14.16 dB), OGLR (32.24 dB), NLRN (33.23 dB), GCDN (**33.37 dB**).

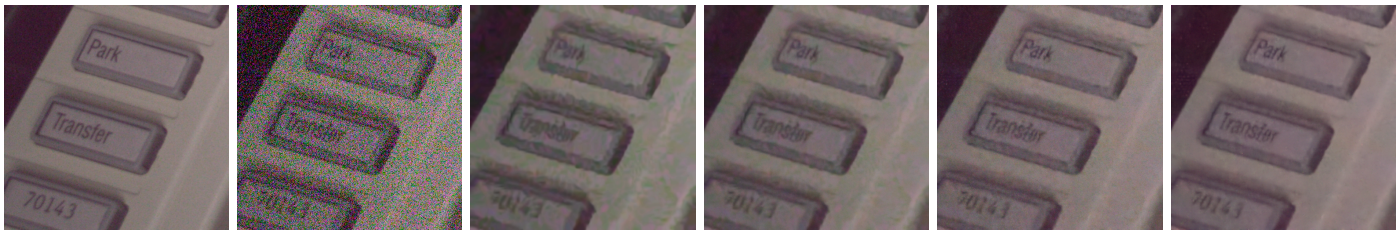


Figure 13. Real image denoising. Left to right: ground truth, noisy (23.69 dB), CBM3D (36.90 dB), DnCNN (38.76 dB), NLRN (40.78 dB), GCDN (**41.21 dB**). Image intensity rescaled for visualization.

[7] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a Gaussian denoiser: residual learning of deep CNN for image denoising," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142–3155, 2017.

[8] X. Mao, C. Shen, and Y.-B. Yang, "Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections," in *Advances in Neural Information Processing Systems*, 2016, pp. 2802–2810.

[9] Y. Tai, J. Yang, X. Liu, and C. Xu, "Memnet: A persistent memory network for image restoration," in *IEEE International Conference on Computer Vision*, 2017, pp. 4539–4547.

[10] W. Bae, J. J. Yoo, and J. C. Ye, "Beyond deep residual learning for image restoration: Persistent homology-guided manifold simplification," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 1141–1149.

[11] C. Cruz, A. Foi, V. Katkovnik, and K. Egiazarian, "Nonlocality-reinforced convolutional neural networks for image denoising," *IEEE Signal Processing Letters*, vol. 25, no. 8, pp. 1216–1220, Aug 2018.

[12] S. Lefkimmiatis, "Universal denoising networks: a novel CNN architecture for image denoising," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3204–3213.

[13] T. Plötz and S. Roth, "Neural nearest neighbors networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 1087–1098.

[14] D. Liu, B. Wen, Y. Fan, C. C. Loy, and T. S. Huang, "Non-local recurrent network for image restoration," in *Advances in Neural Information Processing Systems*, 2018, pp. 1673–1682.

[15] D. Valsesia, G. Fracastoro, and E. Magli, "Image denoising with graph-convolutional neural networks," in *IEEE International Conference on Image Processing*, 2019, pp. 2399–2403.

[16] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[17] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3844–3852.

[18] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017.

[19] D. Shuman, S. 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," *IEEE Signal Processing Magazine*, vol. 3, no. 30, pp. 83–98, 2013.

[20] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European Semantic Web Conference*, 2018, pp. 593–607.

[21] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 29–38.

[22] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *ACM Transactions on Graphics*, vol. 38, no. 5, pp. 1–12, 2019.

[23] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations*, 2019.

[24] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5115–5124.

[25] N. Verma, E. Boyer, and J. Verbeek, "Feastnet: Feature-steered graph

convolutions for 3d shape analysis," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2598–2606.

[26] D. Valsesia, G. Fracastoro, and E. Magli, "Learning localized generative models for 3d point clouds via graph convolution," in *International Conference on Learning Representations*, 2019.

[27] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: nonlinear phenomena*, vol. 60, no. 1-4, pp. 259–268, 1992.

[28] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *IEEE International Conference on Computer Vision*, 1998, pp. 839–846.

[29] S. P. Awate and R. T. Whitaker, "Higher-order image statistics for unsupervised, information-theoretic, adaptive, image filtering," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2005, pp. 44–51.

[30] ——, "Unsupervised, information-theoretic, adaptive image filtering for image restoration," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 28, no. 3, pp. 364–376, 2006.

[31] S. Gu, L. Zhang, W. Zuo, and X. Feng, "Weighted nuclear norm minimization with application to image denoising," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 2862–2869.

[32] J. Pang and G. Cheung, "Graph Laplacian regularization for image denoising: analysis in the continuous domain," *IEEE Transactions on Image Processing*, vol. 26, no. 4, pp. 1770–1785, 2017.

[33] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *IEEE Transactions on Image Processing*, vol. 15, no. 12, pp. 3736–3745, Dec 2006.

[34] Y. Chen and T. Pock, "Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1256–1272, 2016.

[35] H. C. Burger, C. J. Schuler, and S. Harmeling, "Image denoising: Can plain neural networks compete with BM3D?" in *IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 2392–2399.

[36] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[37] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Deep image prior," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9446–9454.

[38] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[39] N. Divakar and R. Venkatesh Babu, "Image denoising via CNNs: an adversarial approach," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 80–87.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[41] L. Gong and Q. Cheng, "Exploiting edge features for graph neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9211–9219.

[42] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S.-F. Chang, "An exploration of parameter redundancy in deep networks with circulant projections," in *IEEE International Conference on Computer Vision*, 2015.

[43] J. Wu, "Compression of fully-connected layer in neural network by kronecker product," in *International Conference on Advanced Computational Intelligence*, 2016, pp. 173–179.

[44] A. Hinrichs and J. Vybíral, "Johnson-lindenstrauss lemma for circulant matrices," *Random Structures & Algorithms*, vol. 39, no. 3, pp. 391–398, 2011.

[45] D. Valsesia and E. Magli, "Binary adaptive embeddings from order statistics of random projections," *IEEE Signal Processing Letters*, vol. 24, no. 1, pp. 111–115, Jan 2017.

[46] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International Conference on Artificial Intelligence and Statistics*, vol. 9, 2010, pp. 249–256.

[47] P. L. Combettes and J.-C. Pesquet, "Proximal splitting methods in signal processing," in *Fixed-point algorithms for inverse problems in science and engineering*. Springer, 2011, pp. 185–212.

[48] P. Milanfar, "A tour of modern image filtering: New insights and methods, both practical and theoretical," *IEEE Signal Processing Magazine*, vol. 30, no. 1, pp. 106–128, Jan 2013.

[49] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *IEEE International Conference on Computer Vision*, vol. 2, 2001, pp. 416–423.

[50] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[51] W. Hu, G. Cheung, A. Ortega, and O. C. Au, "Multiresolution graph fourier transform for compression of piecewise smooth images," *IEEE Transactions on Image Processing*, vol. 24, no. 1, pp. 419–433, Jan 2015.

[52] G. Fracastoro, D. Thanou, and P. Frossard, "Graph transform optimization with application to image compression," *IEEE Transactions on Image Processing*, vol. 29, pp. 419–432, 2019.

[53] G. C. Holst, *CCD arrays, cameras, and displays*. JCD Publishing SPIE Press, 1992.

[54] T. Plotz and S. Roth, "Benchmarking denoising algorithms with real photographs," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1586–1595.

[55] A. Abdelhamed, S. Lin, and M. S. Brown, "A high-quality denoising dataset for smartphone cameras," in *IEEE Conference on Computer Vision and Pattern Recognition*, June 2018.

[56] J. Zeng, J. Pang, W. Sun, and G. Cheung, "Deep graph laplacian regularization for robust denoising of real images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2019, pp. 0–0.

[57] K. Dabov, A. Foi, V. Katkovnik, and K. O. Egiazarian, "Color image denoising via sparse 3d collaborative filtering with grouping constraint in luminance-chrominance space." in *IEEE International Conference on Image Processing*, 2007, pp. 313–316.

[58] G. Chen, F. Zhu, and P. A. Heng, "An efficient statistical method for image noise level estimation," in *IEEE International Conference on Computer Vision*, 2015, pp. 477–485.

[59] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *European conference on computer vision*, 2014, pp. 184–199.

[60] A. Bordone Molini, D. Valsesia, G. Fracastoro, and E. Magli, "Deepsum: Deep neural network for super-resolution of unregistered multitemporal images," *IEEE Transactions on Geoscience and Remote Sensing*, 2019.