Unifying local and non-local signal processing with graph CNNs

Gilles Puy, Srđan Kitić, and Patrick Pérez

Abstract. This paper deals with the unification of local and non-local signal processing on graphs within a single convolutional neural network (CNN) framework. Building upon recent works on graph CNNs, we propose to use convolutional layers that take as inputs two variables, a signal and a graph, allowing the network to adapt to changes in the graph structure. In this article, we explain how this framework allows us to design a novel method to perform style transfer.

1. Introduction

Convolutional neural networks (CNNs) have achieved unprecedented performance in a wide variety of applications, in particular for image analysis, enhancement and editing – *e.g.*, classification [1], super-resolution [2], and colorisation [3]. Yet standard CNNs can only handle signals that live on a regular grid, and each layer of a CNN only performs a local processing. Locality has already been identified as a limitation for classical signal processing tasks where powerful non-local methods have been proposed, such as patch-based methods for inpainting [4] or denoising [5, 6]. Regular CNNs do not allow such a non-local processing. Furthermore, the growing amount of signals collected on irregular grids, such as social, transportation or biological networks, requires extending signal processing from regular to irregular graphs [7].

Any CNN consists of a composition of convolutional and pooling layers. One should thus redefine both convolution and pooling to handle "graph signals". In this work, we use convolutional layers only and hence just concentrate on the generalisation of the convolution. One major challenge in this generalisation is to take into account the possible changes of the graph structure from one signal instance to another: nodes and edges can appear, disappear, and the edge weights can vary. For instance, the connections between the users (graph's vertices) in a social network change over time. It would be cumbersome to retrain a CNN each time a connection changes. In non-local signal processing methods, the situation is even more extreme as the graph is a construct whose edges typically capture similarities between different parts of the signal itself. In this case, the CNN must not be just robust to few variations in the graph structure but fully adapt to these variations. We propose here a solution to this challenge but passing two variables to the CNN: the signal itself, as usual, and the graph structure.

Contributions – We propose a graph CNN framework that takes as inputs two variables: a signal and a graph structure. This permits the adaptation of the CNN to changes in the structure of the graph on which the signal lives, even in the extreme case where this structure changes with the input signal itself. We also propose a unique way of defining convolutions on arbitrary graphs, in particular non-local convolutions, with application to a wide range of many different signal processing applications. Due to space constraint, we only present the use of graph CNNs for image style transfer in this article. We use a local CNN to capture and transfer local style properties of the painting to the photograph. We also use a non-local graph CNN to capture and transfer global style properties of the painting, as well as to preserve the content of the photograph. In addition, we show that this task can be done using only *two random shallow* networks, instead of a trained regular deep CNN [8].

Let us mention that additional experiments in the Appendix demonstrate the effectiveness and versatility of our framework on other kinds of signals (greyscale images, color palettes, and speech signals) and tasks (color transfer and denoising). In particular, the experiments show that it is possible to identify the optimal mixing of local and non-local signal processing techniques by learning.

Gilles Puy, Srđan Kitić and Patrick Pérez are with Technicolor, 975 Avenue des Champs Blancs, 35576 Cesson-Sévigné, France.

2. Graph CNN

2.1. State-of-the-art methods

In this section, we review different existing solutions to generalise CNNs to signal living on graphs. The reader can refer to [9] for a detailed overview. We restrict our attention here to the solutions the most closely connected to ours.

A first approach to redefine convolution for graph signals is to work in the spectral domain, for which we need to define the graph Fourier transform. To introduce this transform, we consider an undirected weighted graph[1] $\mathcal{G}$ with graph Laplacian denoted by $\mathsf{L} \in \mathbb{R}^{n \times n}$. For example, L can be the combinatorial graph Laplacian $\mathsf{L} = \mathsf{D} - \mathsf{W}$, or the normalised one $\mathsf{L} = \mathsf{I} - \mathsf{D}^{-1/2}\mathsf{W}\mathsf{D}^{-1/2}$, where I is the identity matrix and $\mathsf{D} \in \mathbb{R}^{n \times n}$ is the diagonal degree matrix with entries $d_i = \sum_{j=1}^{n} \mathsf{W}_{ij}$ [10]. The matrix L is real symmetric and positive semi-definite. Thus, there exists a set of orthonormal eigenvectors $\mathsf{U} \in \mathbb{R}^{n \times n}$ and real eigenvalues $0 = \lambda_1 \leqslant \ldots \leqslant \lambda_n$ such that $\mathsf{L} = \mathsf{U}\Lambda\mathsf{U}^{\mathsf{T}}$, where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n) \in \mathbb{R}^{n \times n}$. The matrix U is viewed as the graph Fourier basis [7].

For any signal $\boldsymbol{x} \in \mathbb{R}^n$ defined on the vertices of $\mathcal{G}$, $\hat{\boldsymbol{x}} = \mathsf{U}^{\mathsf{T}}\boldsymbol{x}$ is its graph Fourier transform. One way to define convolution on $\mathcal{G}$ with a filter $\boldsymbol{h} \in \mathbb{R}^n$ is by filtering in the graph Fourier domain:

$$\boldsymbol{x} \star \boldsymbol{h} = \mathsf{U}\,(\hat{\boldsymbol{h}} \odot \hat{\boldsymbol{x}}) = \mathsf{U}\mathsf{H}\mathsf{U}^{\mathsf{T}}\boldsymbol{x}, \tag{1}$$

where $\odot$ denotes the entry-wise multiplication and $\mathsf{H} = \mathrm{diag}(\hat{\boldsymbol{h}}) \in \mathbb{R}^{n \times n}$. In the context of graph CNN, it is the approach chosen in [11]. This approach however has several drawbacks: Computing U is often intractable for real-size graphs; Matrix-vector multiplication with U is usually slow (there is no fast graph Fourier transform); This definition does not allow variations in the graph structure as the matrix U is impacted by any such change; The number of filter coefficients to learn is as large as the size of the input signal. The subsequent work [12] solves this last issue by imposing that the filter lives in the span of a kernel matrix $\mathsf{K}^{n \times \tilde{n}}$ with $\tilde{n} \leqslant n$.

To overcome the computational issues of the spectral approach, a known trick in the field of graph signal processing is to define a filter as a polynomial of the graph eigenvalues [13]. Let $\hat{h} \colon \mathbb{R} \to \mathbb{R}$ be a polynomial of degree $m \geqslant 0$: $\hat{h}(t) = \sum_{i=0}^{m} \alpha_i t^i$, with $\alpha_0, \ldots, \alpha_m \in \mathbb{R}$ and consider the filter $\hat{\boldsymbol{h}} = (\hat{h}(\lambda_1), \ldots, \hat{h}(\lambda_n))^{\mathsf{T}}$. One can easily prove that spectral filtering with $\hat{\boldsymbol{h}}$ satisfies

$$\boldsymbol{x} \star \boldsymbol{h} = \mathsf{U}\,(\hat{\boldsymbol{h}} \odot \hat{\boldsymbol{x}}) = \sum_{i=0}^{m} \alpha_i \mathsf{L}^i \boldsymbol{x}. \tag{2}$$

This expression involves only computations in the vertex domain through matrix-vector multiplications with L. As the Laplacian is usually a sparse matrix, filtering a signal with a polynomial filter is fast. This is the approach adopted by [14] and [15] in their construction of graph CNNs. Beyond the computational improvements, the number of coefficients to learn is also reduced: m instead of n. Furthermore, the localisation of the filter in the vertex domain is exactly controlled by the degree of the polynomial [13, 14]. Yet these polynomial filters are not entirely satisfying. Indeed, for, *e.g.*, a graph modelling a regular lattice, polynomial filters are isotropic unlike those in regular CNNs for images – where the underlying graph is a regular lattice. There is no equivalence between regular CNNs and graph CNNs with polynomial filters. Let us also mention the work of [16] where the convolution is defined using a diffusion process on the graph. Due to lack of space, we do not report the exact definition but this one shares similarities with (2) where the normalised transition matrix $\mathsf{P} = \mathsf{D}^{-1}\mathsf{W}$ is substituted for the Laplacian.

In our work, we built upon the work of [17] and [18] to get rid of these shortcomings. The convolutions are directly defined in the vertex domain in a way which allows ones to directly control the computational complexity and the localisation of the filters. Furthermore, these filters do not suffer from the isotropy issue of polynomial filters.

---

[1]A graph $\mathcal{G}$ is a set of n vertices, a set of edges $\mathcal{E}$ and a weighted adjacency matrix $\mathsf{W} = [\mathsf{W}_{ij}] \in \mathbb{R}_+^{n \times n}$, with $\mathsf{W}_{ij} > 0$ iff $(i, j) \in \mathcal{E}$. In this paper, we consider directed graphs unless explicitly stated. The matrix W is thus not symmetric in general.

2.2. Our method

Each layer of our graph CNN implements a function

$$(3) \qquad f \colon (\mathsf{X}, \mathcal{G}) \quad \longmapsto f(\mathsf{X}, \mathcal{G})$$

where $\mathsf{X} \in \mathbb{R}^{n \times m_0}$ is the input signal, $f(\mathsf{X}, \mathcal{G}) \in \mathbb{R}^{n \times m_1}$, and $\mathcal{G}$ is a n-vertex graph on which the columns of X live and which defines how the convolution is done in this layer. The input signal X has size n in the "spatial" dimension – $e.g.$, n pixels for images – and has m_0 channels or feature maps – $e.g.$, $m_0 = 3$ for color images. The output signal has same spatial size n – we do not use any pooling layers in this work – and m_1 feature maps.

2.2.1. Convolution

The convolution we use follows principles also used in, $e.g.$, [19, 20, 17, 18], where the computation done at one vertex is a function of (at least) the values of the signal at this vertex and neighbouring vertices as well as of labels attributed to each edge. We choose here to use the formalism of [18] for our description.

Convolutions in [18] are done in two steps: the extraction of a signal patch around each vertex and a scalar product. We assume here that all vertices have the same number of connections: $|\{j \colon (i, j) \in \mathcal{E}\}| = d$ for all $i \in \{1, \ldots, n\}$. If this is not the case, one can always complete the set of edges and associate to these edges, $e.g.$, a null weight. We also assume that $(i, i) \in \mathcal{E}$ for all $i \in \{1, \ldots, n\}$.

For a given graph $\mathcal{G}$ satisfying the above assumption and with adjacency matrix $\mathsf{W} \in \mathbb{R}^{n \times n}$, we model patch extraction at vertex i with a function

$$p \colon \{1, \ldots, n\} \times \mathbb{R}^n \longrightarrow \mathbb{R}^d$$
$$(4) \qquad \qquad (i, \boldsymbol{x}) \quad \longmapsto (p_1(i, \boldsymbol{x}), \ldots, p_d(i, \boldsymbol{x}))^{\mathsf{T}}$$

where each $p_\ell(i, \boldsymbol{x}) \in \mathbb{R}$, $\ell = 1, \ldots, d$, extracts one entry of the vector $\boldsymbol{x}$, which represents one column of the input signal X. Let $j_1, \ldots, j_d$ be the d indices to which i is connected. The order in which these d entries are extracted by p is determined by "pseudo-coordinates" $u(i, j_k) \in \{1, \ldots, d\}$ attributed to each connected vertex j_k [18]. The nature of these pseudo-coordinates will be given in Section 2.2.2 for local convolution and in Section 2.2.3 for non-local convolution. We define

$$(5) \qquad p_\ell(i, \boldsymbol{x}) = g(\boldsymbol{w}_i, j_k) \; \boldsymbol{x}_{j_k}$$

where $u(i, j_k) = \ell$. The vector $\boldsymbol{w}_i \in \mathbb{R}^n$ is the ith row of W, which contains at most d non-zero entries, and $g \colon \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ is a re-weighting function that gives the possibility to account for each edge weight in the convolution. We noticed that the choice of this function is very important in the definition of the non-local convolutions to achieve good results in our signal processing applications (see its definition in Section 2.2.3). Note that g depends on $\boldsymbol{w}_i$ and j_k in our work while this function depends solely on the pseudo-coordinates in [18]. This is a simple but important modification for our applications.

Convoluting $\boldsymbol{x}$ with a filter $\boldsymbol{h} \in \mathbb{R}^d$ is then defined as in [18]:

$$(6) \qquad (\boldsymbol{x} \star \boldsymbol{h})(i) = \boldsymbol{h}^{\mathsf{T}} p(i, \boldsymbol{x}),$$

for all $i \in \{1, \ldots, n\}$. Finally, the function f in (3) satisfies

$$(7) \qquad f(\mathsf{X}, \mathcal{G}) = \left(s\left(\sum_{j=1}^{m_0} \boldsymbol{x}_j \star \boldsymbol{h}_j^\ell, b^\ell \right) \right)_{\ell = 1, \ldots, m_1}$$

where $s \colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is an element-wise non-linearity, $e.g.$, ReLU defined as $s(\boldsymbol{x}, b) = \text{ReLU}_b(\boldsymbol{x}) = \max\{0, \boldsymbol{x} + b\}$, $\boldsymbol{x}_j \in \mathbb{R}^n$ denotes the jth column-vector of X, $\boldsymbol{h}_j^\ell \in \mathbb{R}^d$, $j = 1, \ldots, m_0$, $\ell = 1, \ldots, m_1$, are filters, and $b^1, \ldots, b^{m_1}$ are biases.

Let us highlight that the size n of the input signal X in the spatial dimension is not fixed in (7). Hence, f can be computed for signals of different sizes using the same filters $\boldsymbol{h}_j^\ell$, exactly as with regular CNNs.

We explain in the next section how one can recover the usual local convolution for images from this definition. We will then continue with the description of the proposed non-local filtering in the Section 2.2.3.

2.2.2. Local convolution

As noticed in [18], the above definition of convolution permits us to recover easily the standard convolution for images (or, similarly, signals on regular lattices) by constructing a local graph from the Cartesian $2D$-coordinates of each pixel in the image.[2] We denote these coordinates $(\alpha(i), \beta(i))$, $i = 1, \ldots, n$. For a filter of size $\sqrt{d} \times \sqrt{d}$, we connect each pixel i to all its local neighbours $j_1, \ldots, j_d$ that satisfies $|\alpha(j_k) - \alpha(i)| \leqslant \lfloor \sqrt{d}/2 \rfloor$ and $|\beta(j_k) - \beta(i)| \leqslant \lfloor \sqrt{d}/2 \rfloor$, for $k = 1, \ldots, d$. We then build the local adjacency matrix W that satisfies $\mathsf{W}_{ij} = 1$ if $(i, j) \in \mathcal{E}$, and 0 otherwise. The pseudo-coordinates are determined using the relative position of each pixel j_k to pixel i. For any pixel of the image, the connected pixels have relative coordinates in $\{(\alpha(i) - \alpha(j_k), \beta(i) - \beta(j_k)), 1 \leqslant k \leqslant d\}$. We thus create a look up table $c : \mathbb{R} \times \mathbb{R} \to \{1, \ldots, d\}$ that associates a unique integer $\ell \in \{1, \ldots, d\}$ to each of these relative coordinates. Then, we define $u(i, j_k) = c(\alpha(i) - \alpha(j_k), \beta(i) - \beta(j_k))$.

With this procedure the pixels are always extracted in the same order, *e.g.*, lexicographically. Finally, (6) is equivalent to the usual convolution when using $g(\cdot) = 1$ in (5).

2.2.3. Non-local convolution

We now describe our proposition to perform more general non-local convolutions, *i.e.*, we give the definition of the pseudo-coordinates and of the function g in (5). In our applications, these convolutions are based on a graph $\mathcal{G}$ that captures some structure that we wish to preserve in the signal X. The exact construction of $\mathcal{G}$ thus differs depending on the application. Yet, we define the pseudo-coordinates and the function g always in the same way, whatever the application.

The weight W_{ij} of the edge between vertices i and j is determined based on a distance between feature vectors $\boldsymbol{f}_i$ and $\boldsymbol{f}_j$ extracted at vertex i and j, respectively. Let $\Delta(\boldsymbol{f}_i, \boldsymbol{f}_j) \geqslant 0$ denote this distance. Let again $j_1, \ldots, j_d$ be the vertices to which vertex i is connected and ordered such that $\Delta(\boldsymbol{f}_i, \boldsymbol{f}_{j_1}) \leqslant \ldots \leqslant \Delta(\boldsymbol{f}_i, \boldsymbol{f}_{j_d})$. We propose to define the pseudo-coordinates as $u(i, j_k) = k$, $k \in \{1, \ldots, d\}$. In other words, the pseudo-coordinates re-order the distances between feature vectors in increasing order. Note that we break any tie arbitrarily.

Finally, we propose to use the following function g in (5):

$$(8) \qquad g(\boldsymbol{w}_i, j) = \frac{\boldsymbol{w}_{ij}}{\sum_{k=1}^{n} \boldsymbol{w}_{ik}} \, d,$$

where $\boldsymbol{w}_{ij} = \mathsf{W}_{ij}$ is the jth entry of $\boldsymbol{w}_i$.

3. Style transfer with graph CNNs

In this section, we substitute $f(\mathsf{X})$ for $f(\mathsf{X}, \mathcal{G})$ in (7) to simplify notations. However, one should not forget that the convolution at each layer is defined by an underlying graph $\mathcal{G}$. This graph will always be defined explicitly in the text.

Style transfer consists in transforming a target image $\mathsf{X}_t \in \mathbb{R}^{n \times 3}$, typically a photograph, to give it the "style" of a source image $\mathsf{X}_s \in \mathbb{R}^{n' \times 3}$, typically a painting. Impressive results have recently been obtained using CNNs [8]. The style transfer method of Gatys *et al.* consists in solving a minimization problem of the form

$$(9) \qquad \mathsf{X}^* \in \operatorname*{argmin}_{\mathsf{X} \in \mathbb{R}^{n \times 3}} \sum_{\ell \in \mathcal{L}_s} \| f_\ell(\mathsf{X})^\intercal f_\ell(\mathsf{X}) - f_\ell(\mathsf{X}_s)^\intercal f_\ell(\mathsf{X}_s) \|_{\mathrm{F}}^2 + \lambda \sum_{\ell \in \mathcal{L}_t} \| f_\ell(\mathsf{X}) - f_\ell(\mathsf{X}_t) \|_{\mathrm{F}}^2 \,,$$

where $f_\ell(\mathsf{X})$ is a matrix with the feature maps at depth ℓ of a multi-layer CNN, $\mathcal{L}_s$ and $\mathcal{L}_t$ are two subsets of depths, and $\|\cdot\|_{\mathrm{F}}$ denotes Frobenius norm. Gatys *et al.* used the very deep VGG-19 network, pre-trained for image classification [21]. The first term encourages the solution X^* to have the style of

---

[2]We consider a regular grid of equispaced pixels.

the painting X_s by matching the Gram matrices of the feature maps, such statistics capturing texture patterns at different scales. The second term ensures that the main structures (the "content") of the original photograph X_t, as captured in feature maps, are preserved in X^*. Note that all the spatial information is lost in the first term that encodes the style, while it is still present in the second term. It was proved shortly after that similar results can be obtained using a deep neural network with all the filter coefficients chosen randomly [22]. Let us also mention that [23] showed that texture synthesis, *i.e.*, when only the first term in (9) is involved, can be done using *multiple (8) one-layer* CNNs with random filters giving each $m = 1024$ feature maps.

We show now that our graph-based CNNs allow us to revisit neural style transfer. We use only *two one-layer* graph CNNs with *random filters* giving each only 50 feature maps. This is a much "lighter" network than the ones used in the literature. The first network, denoted f_1, uses local convolutions (Section 2.2.2) and the second, denoted f_2, uses non-local convolutions (Section 2.2.3) on a graph that captures the structure of the photograph X_t to be preserved. Both f_1 and f_2 have the form (7) with $m_0 = 3$ for the three *Lab* channels of color images and $m_1 = 50$. We also choose $d = 25$ and ReLU for the non-linearity in both cases. The $25 \times 3 \times 50 \times 2 = 7500$ coefficients of the filters $\boldsymbol{h}_j^\ell$ and the $50 \times 2 = 100$ biases b^ℓ in (7) are chosen randomly using independent draws from the standard Gaussian distribution.

The graph in the second CNN f_2 is constructed as follows. For an image of interest X, we construct a feature vector $\boldsymbol{f}_i \in \mathbb{R}^{29}$ at each pixel i of the image by extracting all the pixels' *Lab* values in the neighbourhood of size 3×3 around i as well as the absolute 2D coordinates of the pixel. We then search the $d = 25$ nearest neighbours to $\boldsymbol{f}_i$ in the set $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_n\}$ using the Euclidean distance. Let $\mathcal{D} = \{\|\boldsymbol{f}_i - \boldsymbol{f}_j\|_2\}_{ij}$ be the set of all distances between each $\boldsymbol{f}_i$ and its nearest neighbours. We have $|\mathcal{D}| = 25n$. To avoid that some pixels are too weakly connected to others, which then produces artefacts in the final images, we compute the 80th percentile of the values in $\mathcal{D}$ and saturates all the distances above this percentile to this value. The weights of the adjacency matrix W then satisfy

$$(10) \qquad \mathsf{W}_{ij} = \exp\left(-\|\boldsymbol{f}_i - \boldsymbol{f}_j\|_2^2 / \sigma^2\right), \; \forall (i,j) \in \mathcal{E},$$

with σ equal to the 75th percentile of $\mathcal{D}$.

We capture the style of the painting X_s by computing the Gram matrices

$$(11) \qquad \mathsf{G}_1 = f_1(\mathsf{X}_s)^\intercal f_1(\mathsf{X}_s) \text{ and } \mathsf{G}_2 = f_2(\mathsf{X}_s)^\intercal f_2(\mathsf{X}_s),$$

where the non-local convolution in f_2 is computed using the graph constructed on X_s. The matrix G_1 captures local statistics while G_2 captures non-local statistics. To give the style of X_s to X_t, we now compute a new graph on X_t. We then compute an image $\mathsf{X}^* \in \mathbb{R}^{n \times 3}$ by solving

$$(12) \qquad \min_{\mathsf{X} \in \mathbb{R}^{n \times 3}} \gamma_1 \|f_1(\mathsf{X})^\intercal f_1(\mathsf{X}) - \mathsf{G}_1\|_{\mathrm{F}}^2 + \gamma_2 \|f_2(\mathsf{X})^\intercal f_2(\mathsf{X}) - \mathsf{G}_2\|_{\mathrm{F}}^2 + \gamma_3 \|\mathsf{X}\|_{\mathrm{TV}},$$

where f_2 uses, this time, *the graph constructed on* X_t for the non-local convolutions, $\|\cdot\|_{\mathrm{TV}}$ is the Total Variation norm, and $\gamma_1, \gamma_2, \gamma_3 > 0$. Note that unlike in (9), we do not try to match feature maps but only Gram matrices. Yet the final image X^* retains the structure of X_t thanks to the non-local convolution in f_2.

In practice, we minimise (12) using the L-BFGS algorithm starting from a random initialisation of X. The parameters γ_1, γ_2 are computed so that the gradient coming from the term they respectively influence has a maximum amplitude of 1 at the first iteration of the algorithm. We set $\gamma_3 = 0.01n$. All images used in the experiments have size $n = 256 \times 256$. However, we do not solve (12) directly at this resolution, but in a coarse-to-fine scheme instead: We start by downsampling all images at $n = 32 \times 32$ pixels; Solve (12) at this resolution; Upscale the solution at 64×64 pixels; Restart the same process at this new resolution using the up-scaled image as initialisation; Repeat this process until the final resolution is reached.

We present some results obtained with our graph-based method in Fig. 1. One can notice that the main structure of the photograph X_t perfectly appears in X^* thanks to the presence of the structure-preserving non-local convolutions in f_2. The style is also well transferred thanks to the matching of the Gram matrices G_1 and G_2.

Figure 1. *Examples of style transfer results obtained where the photograph (left) is transformed to have the style of a given painting (top). Using the proposed graph CNN framework, only two one-layer random CNNs are required to extract matched statistics.*

To highlight the role of the graph CNN with non-local convolutions, we repeat exactly the same experiments but using only the non-local graph CNN, *i.e.*, we do not use the regular CNN with local convolutions – the TV regularisation is still present. Fig. 2 shows results for one photograph and different paintings. First, we notice that the main structures of the photograph are well preserved thanks to the graph CNN. Second, the colors of the painting and the relative arrangement of the colors are well transferred. However, we are not able to transfer finer style details like brush strokes. On the contrary, the local CNN is able to capture these finer details which appear in the results with the complete method.

Let us highlight that this experiment already shows that our graph CNN framework can adapt to many changes in the graph structure. Indeed, the graph used in f_2 was built from the painting when computing G_2 while it is built from the photograph when computing X^*.

4. Conclusion

We proposed a graph CNN framework that allows us to unify local and non-local processing of signals on graphs, and showed how to use this framework to perform style transfer. The results already suggest that the proposed convolution adapts correctly to changes in the input graph. Additional experiments in the Appendix demonstrate the versatility of our framework on other kinds of signals
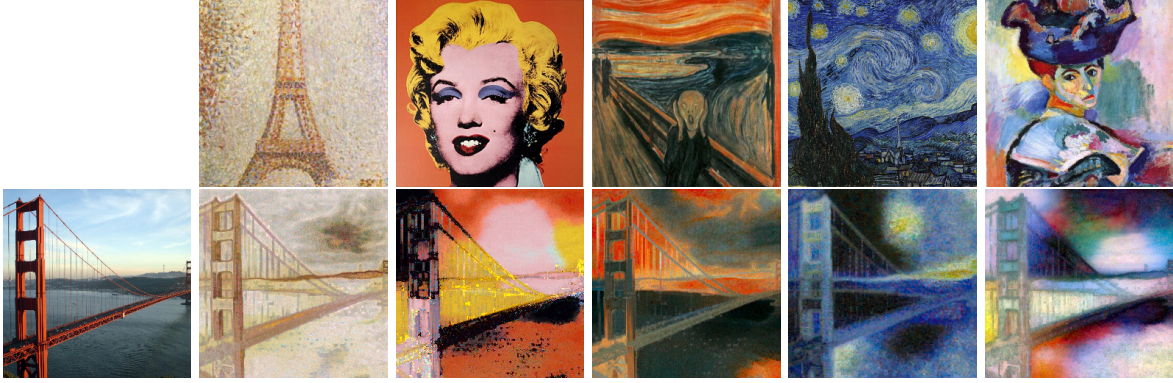
Figure 2. *Examples of style transfer results obtained with our graph CNN method using* only non-local convolutions *where the photograph (left) is transformed to have the style of a given painting (top).*

and tasks. Beyond signal processing, we believe that some of the tools presented here can be useful to other applications involving time-varying graph structures, such as in social networks.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[2] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 3238, no. 2, pp. 295–307, 2016.

[3] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *ECCV*, 2016.

[4] A. Criminisi, P. Pérez, and K. Toyama, "Region filling and object removal by exemplar-based image inpainting," *IEEE Trans. Image Process.*, vol. 13, no. 9, pp. 1200–1212, 2004.

[5] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," in *CVPR*, 2005.

[6] R. Talmon, I. Cohen, and S. Gannot, "Transient noise reduction using nonlocal diffusion filters," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 6, pp. 1584–1599, 2011.

[7] 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. 30, no. 3, pp. 83–98, 2013.

[8] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *CVPR*, 2016.

[9] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond Euclidean data," *arXiv:1611.08097*, 2016.

[10] F. Chung, *Spectral graph theory.* Amer Mathematical Society, 1997, no. 92.

[11] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *ICLR*, 2014.

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

[13] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Appl. Comput. Harmon. Anal.*, vol. 30, no. 2, pp. 129–150, 2011.

[14] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *NIPS*, 2016.

[15] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv:1609.02907*, 2016.

[16] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *NIPS*, 2016.

[17] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *ICML*, 2016.

[18] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model CNNs," *arXiv:1611.08402*, 2016.

[19] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[20] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *Int. Conf. on Learning Representations*, 2016.

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.

[22] K. He, Y. Wang, and J. Hopcroft, "A powerful generative model using random weights for the deep image representation," in *NIPS*, 2016.

[23] I. Ustyuzhaninov, W. Brendel, L. A. Gatys, and M. Bethge, "Texture synthesis using shallow convolutional networks with random filters," *arXiv:1606.00021*, 2016.

[24] O. Frigo, N. Sabater, V. Demoulin, and P. Hellier, "Optimal transportation for example-guided color transfer," in *ACCV*, 2014.

[25] S. Ferradans, N. Papadakis, J. Rabin, G. Peyré, and J.-F. Aujol, "Regularized discrete optimal transport," in *SSVM*, 2013.

[26] H. Hristova, O. L. Meur, R. Cozot, and K. Bouatouch, "Style-aware robust color transfer," in *EXPRESSIVE*, 2015.

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

[28] H. Jégou, M. Douze, and C. Schmid, "Hamming embedding and weak geometry consistency for large scale image search," in *ECCV*, 2008.

[29] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett, "DARPA TIMIT acoustic-phonetic continous speech corpus CD-ROM. nist speech disc 1-1.1," *NASA STI/Recon technical report n*, vol. 93, 1993.

Appendix A - Color transfer

In this appendix, we address another application that is color transfer where the goal is to transfer the "color palette" of a source image onto the one of the target image. First, we describe how we perform this task using graph CNNs. Second, we describe a more traditional approach using optimal transport, such as used in [24], with which we will compare our results.

A.1. Color transfer with graph CNNs

All images are converted to *Lab* and only chrominances a and b are processed – the luminance of target images remains untouched. The first step consists in building a palette representative of the color distribution of each image. This is done by clustering image's chrominances into $n = 64$ clusters using k-means. We thus obtain a source palette $\mathsf{X}_s \in \mathbb{R}^{n \times 2}$ and a target one $\mathsf{X}_t \in \mathbb{R}^{n \times 2}$. The first and second columns of these palettes represent respectively the a and b channels. The pixel values in the ab channels are normalised and shifted to be in the range $[-0.5, 0.5]$.

Similarly to what was done for style transfer, we capture the "statistics" of the reference palette X_s by computing the Gram matrix of the output feature maps of a graph convolutional layer f of the form of (7). We have $m_0 = 2$. We choose $m_1 = 100$ and $d = 10$. The coefficients of the filters $\boldsymbol{h}_j^\ell$ and the biases b^ℓ in (7) are chosen randomly using independent draws from the standard Gaussian distribution. We use ReLU for the non-linearity.

The network f implements non-local convolutions (Section 2.2.3) with a graph constructed as follows. For a palette $\mathsf{X} \in \mathbb{R}^{n \times 2}$, we construct a feature vector $\boldsymbol{f}_i \in \mathbb{R}^2$ for each palette element i that simply contains the ab values for that element. We then search the $d = 10$ nearest neighbours to $\boldsymbol{f}_i$ in the set $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_n\}$ using the Euclidean distance. The weights of the adjacency matrix W representing the graph $\mathcal{G}$ used for convolution satisfy

$$(13) \qquad \mathsf{W}_{ij} = \exp\left(-\|\boldsymbol{f}_i - \boldsymbol{f}_j\|_2^2 / \sigma^2\right),$$

with $\sigma = 0.25$, for connected palette entries i and j.

We capture the color statistics of the source image by computing the Gram matrix

$$(14) \qquad \mathsf{G}_s = f(\mathsf{X}_s)^\intercal f(\mathsf{X}_s),$$

where the graph used in f was built using X_s.

To find a mapping between the colors in the source and target images, we solve the following minimisation problem

$$(15) \qquad \min_{\mathsf{X} \in \mathbb{R}^{n \times 2}} \gamma_1 \|f(\mathsf{X})^\intercal f(\mathsf{X}) - \mathsf{G}_s\|_\mathrm{F}^2 + \gamma_2 \|\mathsf{X} - \mathsf{X}_t\|_\mathrm{F} + \gamma_3 \operatorname{Tr}\left(\mathsf{X}^\intercal \mathsf{L} \mathsf{X}\right),$$

where the graph used for the convolutions in f is now built using X_t, $\mathsf{L} \in \mathbb{R}^{n \times n}$ and $\gamma_1, \gamma_2, \gamma_3 > 0$. We explain the role of each term and give the definition of L in the next paragraph. We solve this problem using the L-BFGS algorithm starting from X_t as initialisation. The parameter γ_1 is computed so that the gradient coming from the term it influences has a maximum amplitude of 1 at the first iteration

of the algorithm. We set $\gamma_2 = 0.1$ and $\gamma_3 = 1/\max_i\{\mathsf{L}_{ii}\}$. Let $\mathsf{X}^* \in \mathbb{R}^{n \times 2}$ be the obtained solution to (15). We transform the color in target image as follows. For each pixel ab value in this image, we find its nearest neighbour in the palette X_t, say the i-th color in the palette. The new color is then obtained by replacing the pixel ab value by the ith entry of the new palette X^*, so that there are only 64 distinct colors in the final images.

The first term in (15) ensures that the colors in X^* are similar to the colors in X_s. The second and third terms in (15) permit us to ensure a consistency in the mapping from the old colors X_t to the new colors X^*. The second term controls the total cost of moving from the old colors to the new colors. Note that this cost is also usually involved in optimal transport methods for color transfer [25, 24]. The role of the third term is to ensure that two similar palette elements in X_t should map to similar palette elements in X^*. The matrix L is the combinatorial Laplacian matrix constructed from a symmetric version $\widetilde{\mathsf{W}}$ of the adjacency matrix W built from X_t: $\widetilde{\mathsf{W}} = (\mathsf{W} + \mathsf{W}^{\mathsf{T}})/2$. The third term in (15) classically promotes smoothness on the weighted symmetrized graph since $\text{Tr}\,(\mathsf{X}^{\mathsf{T}}\mathsf{L}\mathsf{X}) = \frac{1}{2}\sum_{i,j=1}^n \sum_{k=1}^2 \widetilde{\mathsf{W}}_{ij}(\mathsf{X}_{ik} - \mathsf{X}_{jk})^2$. Similar elements in palette X_t should remain similar in transformed palette X^*.

A.2. Optimal transport

Let $\mathsf{C} \in \mathbb{R}^{n \times n}$ be the cost matrix with entries $\mathsf{C}_{ij} = \|(\mathsf{X}_t)_i - (\mathsf{X}_s)_j\|_2$, where $(\mathsf{X}_t)_i \in \mathbb{R}^2$ and $(\mathsf{X}_s)_j \in \mathbb{R}^2$ are the ith and j^{th} rows of X_t and X_s, respectively. This matrix encodes the cost of moving the palette elements in X_t to the palette elements in X_s, and vice-versa. The optimal transport problem is about finding the transport from X_t to X_s that is the least costly:

$$(16) \qquad \min_{\Gamma \in \mathbb{R}^{n \times n}} \langle \mathsf{C}, \Gamma \rangle_{\text{F}} \;\; \text{s.t.} \;\; \begin{cases} 0 \leqslant \mathbf{1}\Gamma \leqslant n^{-1}, & \mathbf{1}\Gamma\mathbf{1}^{\mathsf{T}} = 1, \\ 0 \leqslant \Gamma\mathbf{1}^{\mathsf{T}} \leqslant n^{-1}, & \Gamma \geqslant 0, \end{cases}$$

where $\mathbf{1} = (1,\ldots,1)^{\mathsf{T}} \in \mathbb{R}^n$ and the inequalities on the right hand side hold element-wise. This is the optimal transport problem solved in [24] for color transfer, except that the cost matrix C also incorporates information about the luminance of each image in their work, which hence yields different results. Let $\Gamma^* \in \mathbb{R}^{n \times n}$ be the solution to the above convex problem. We compute the new palette $\mathsf{X}_{ot} \in \mathbb{R}^{n \times 2}$ whose rows read

$$(17) \qquad (\mathsf{X}_{ot})_i = \frac{\sum_{j=1}^n \Gamma^*_{ij}(\mathsf{X}_s)_j}{\sum_{j=1}^n \Gamma^*_{ij}}$$

for $i = 1,\ldots,n$. Remark that the palette X_{ot} is made of colors similar to those in the palette X_s as each palette element of X_{ot} is a convex combination of palette elements in X_s. We finally transform the color in image t as follows. For each pixel ab value in image t, we find the nearest neighbour in the palette X_t, say the ith entry of X_t. The new color is then obtained by replacing the pixel ab value by the ith entry of the new palette X_{ot}.

A.3. Results

We present color transfer results obtained with both methods in Fig. 3. One can notice that our results suffer from fewer artefacts than the ones obtained with optimal transport. One can also refer to the results of [26] on the same images[3] for comparison with a method more evolved than a sole optimal transport. We believe that our results achieve similar visual qualities. Let us also mention that the optimal transport results presented here can certainly be improved thanks to some extra graph-regularisation terms such as used in [25]. Nevertheless, our results show that one can achieve competitive results compared to the state-of-the-art with a completely different approach that uses shallow graph CNNs with random weights.

Beyond the transformed images, it is also interesting to study how each palette is transformed with the different methods. We present in Fig. 4 these different color palettes. We remark that the palette X_{ot} obtained with optimal transport is almost identical to the target palette X_s, at the price of several

---

[3]http://people.irisa.fr/Hristina.Hristova/publications/2015_EXPRESSIVE/indexResults.html

Target image	Our method	Optimal transport	Source colors

Figure 3. *Examples of color transfer results where the photograph on the left is modified to take the color of the photograph on the right. The second image from the left shows the result obtained with our graph CNN method while the second from the right shows the result obtained with optimal transport.*

artefacts in the resulting images. On the contrary, we observe more differences between the palette X^* obtained with our graph CNN method and X_s: a better preservation of the internal structure of X_t is obtained in X^* thanks to the graph regularisation.

Appendix B - Training graph CNNs for denoising

In this appendix, we show that one can train a graph CNN to solve standard signal processing tasks. To demonstrate that the approach can lend itself to different kinds of data, this part is about denoising images and single-channel audio signals.

The ground truth signal is denoted $\boldsymbol{x} \in \mathbb{R}^n$, and its components take values in range $[0, 1]$ (image pixels), and $[-1, 1]$ (audio samples). An associated noisy version satisfies

$$(18) \qquad \boldsymbol{y} = \boldsymbol{x} + \boldsymbol{n},$$

where $\boldsymbol{n} \in \mathbb{R}^n$ is a random vector drawn from the centered Gaussian distribution of standard deviation $\sigma = 0.07$, for images, or $\sigma = 0.12$, for audio.

Figure 4. *Examples of color transfer results where the photograph on the left is modified to take the color of the photograph on the right. The second image from the left shows the result obtained with our graph CNN method while the second from the right shows the result obtained with optimal transport. We present below each image the corresponding color palettes X_t, X^*, X_{ot}, X_s (from left to right).*

In the following experiments, we compare the denoising performance achieved by trained local and non-local graph CNNs.
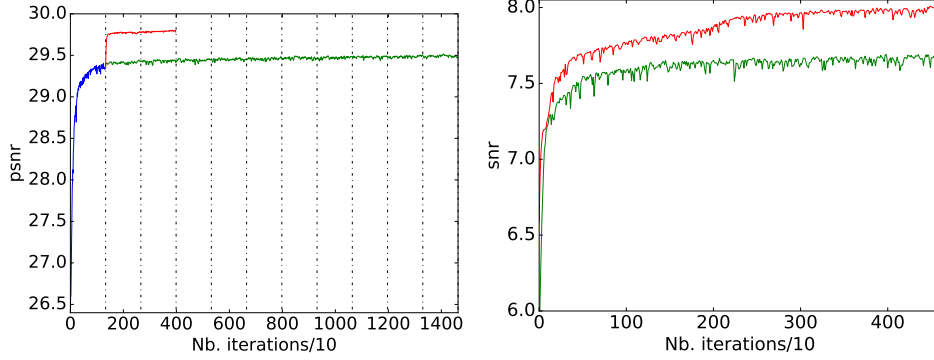
Figure 5. *Left: Evolution of the average PSNR on the image validation set during training. The blue curve corresponds to the pre-training of the local network. The green and red curves correspond to the training of the local and non-local networks, respectively, using the pre-trained network for the initialisation. The vertical dashed-dotted black lines indicate the end of an epoch. Right: Evolution of the average SNR on the audio validation set during training. The green and red curves correspond to the training of the local and non-local networks, respectively.*

B.1. Network structures

For both types of signals, the first and the last layer of denoising CNNs are using local convolutions (Section 2.2.2). We denote these layers by $f_1 \colon \mathbb{R}^n \to \mathbb{R}^{n \times m_1}$ and $f_2 \colon \mathbb{R}^{n \times m_1} \to \mathbb{R}^n$, where only f_1 incorporates a non-linearity, while f_2 is fully linear (without bias). For ease of notation, we slightly abuse the definition of non-linearity in (7), to accommodate the soft-thresholding function

$$(19) \qquad s(x, b_+, b_-) = \left\{ \begin{array}{ll} x - b_+ & \text{if } x \geqslant b_+ \\ 0 & \text{if } x \in (b_-, b_+) \\ x - b_- & \text{if } x \leqslant b_- \end{array} \right. ,$$

where $b_+ \geqslant 0$ and $b_- \leqslant 0$ are parameters learned during training. The first layer f_1 thus takes as input a noisy signal $\boldsymbol{y}$, while f_2 returns its denoised version.

The graph for the non-local convolutions is built on the noisy input signal $\boldsymbol{y}$ and thus changes with each new signal to denoise. It is constructed based on the nearest neighbour search as follows. From the noisy signal, we assemble a feature vector $\boldsymbol{f}_i \in \mathbb{R}^r$ by extracting all the components (*i.e.* pixels or samples) in the predefined local neighbourhood centered around the ith entry of $\boldsymbol{y}$. We then search the d nearest neighbours to $\boldsymbol{f}_i$ in the set $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_n\}$ using the Euclidean distance. The weight W_{ij} of the adjacency matrix W between connected elements i and j satisfies

$$(20) \qquad \mathsf{W}_{ij} = \exp\left(- \alpha \, \max\left\{ \frac{\|\boldsymbol{f}_i - \boldsymbol{f}_j\|_2^2}{r} - \beta, \, 0 \right\} \right),$$

where $\alpha, \beta > 0$ are parameters learned during training. Note that (20) is the weighting function used in NL-means [5].

We compare the denoising performance reached by a CNN using only local convolutions, *i.e.*, a regular CNN, and a graph CNN where we also use non-local convolutions.

Image denoising – The regular CNN uses only f_1 and f_2, and, therefore, implements the function $f_2 \circ f_1$. We choose $d = 9$, hence the local filters have size 3×3, and $m_1 = 18$. The non-local network is built by inserting one non-local layer between f_1 and f_2, which we denote by $f_{\text{nl}} \colon \mathbb{R}^{n \times m_1} \to \mathbb{R}^{n \times m_1}$. This network implements the function $f_2 \circ f_{\text{nl}} \circ f_1$. The neighbourhood used for building the graph through features is of size 7×7 ($r = 49$). The non-local layer f_{nl} is linear (no bias or soft-thresholding) and we interpret its role as an additional non-local denoising of the sparse feature maps given by f_1 before reconstruction of the denoised image by f_2. This layer implements a 2D graph-convolution with a single filter $\boldsymbol{h} \in \mathbb{R}^d \colon f_{\text{nl}}(\mathsf{X}, \mathcal{G}) = (f_1(\boldsymbol{x})_j \star \boldsymbol{h})_{j=1,\ldots,m_1}$. For the local layers, we use $d = 9$. We remark that, while one may expect an equivalent *local* CNN (implementing $f_2 \circ f_\ell \circ f_1$, with f_ℓ

Figure 6. *Noisy image (left, 23.13 dB) denoised with the trained local network (middle left, 28.28 dB) or the trained non-local network (middle right, 28.61 dB) and the original image (right).*

a fully linear 2D local convolution layer of the same size as f_{nl}) to perform better than $f_2 \circ f_1$, our experiments showed that both yield equivalent results.

Audio denoising – The audio denoising CNNs (local and non-local) share the same architecture, defined as $f_2 \circ f_\bullet \circ f_1$. The middle layer $f_\bullet \colon \mathbb{R}^{n \times m_1} \to \mathbb{R}^{n \times m_1}$ is linear (no bias or soft-thresholding) and implements either local (f_1) or non-local (f_{nl}) convolutions[4] as defined in (7). Therefore, the number of filter coefficients to train is equal in both local and non-local cases. We choose filters of size $d = 5$ and $m_1 = 10$. The non-local graph is generated as described before, with a local neighbourhood defined as a signal segment of length $r = 25$.

B.2. Experimental setup

In all layers, one filter is initialised to the constant filter $(1/d, \ldots, 1/d) \in \mathbb{R}^d$, serving to extract and approximately reconstruct the mean of each patch. The remaining filters are initialised using random draws from the centered Gaussian distribution with standard deviation 0.01. We noticed that this initialisation accelerates the learning. The parameters b_+ and b_- (m_1 of each) are initialised at 0.

Training is done using ADAM [27], with a batchsize of size 1, and by minimising the Euclidean distance between the denoised and ground-truth signals. The datasets are divided into three subsets: training, validation and test sets. The validation set is used to monitor the evolution of the PSNR (images) or SNR (audio) during training. The training and test examples are always generated by corrupting the original signal, as defined in (18), with independent draws of $\boldsymbol{n}$.

Image denoising – We train the two image denoising CNNs using the *holidays* dataset which contains 1491 images [28]. The test set is built by choosing 150 images at random among all available images; the validation set is build by choosing 15 instances among the remaining ones; the rest form the training set. All images are cropped to become square and resized to have size $n = 128 \times 128$. First, we pretrain the local network $f_1 \circ f_2$ for 1 epoch and a stepsize of 10^{-3}. Second, we build the non-local network $f_2 \circ f_{\mathrm{nl}} \circ f_1$ using the pre-trained layers f_1 and f_2 and train this whole network using a stepsize of 10^{-4}. Third, starting again from the pre-trained layer f_1 and f_2, we continue training the local network $f_1 \circ f_2$ using a stepsize of 10^{-4}. Note that training the non-local network is computationally more expensive than training the local network as a new non-local graph must be constructed for each new noisy image. We thus limited training of $f_2 \circ f_{\mathrm{nl}} \circ f_1$ to 2 epochs instead of 10 for $f_1 \circ f_2$. For the non-local layer, the default values proposed in [5] are used to initialise the parameters α and β ($1/(0.40\sigma^2)$ and $2\sigma^2$, respectively).

Audio denoising – For audio, we use the *TIMIT* [29] speech dataset, counting 4618 training and 1680 test tracks of varying duration, sampled at 16 kHz. The validation set is built by separating 15 tracks from the training set chosen at random. The audio files are first trimmed to have equal length by extracting a 1 second signal from the original audio tracks. The two parameters in (20) are initialised at $\alpha = 1/\sigma^2$ and $\beta = 0$, and the training is performed on the whole network at once (for both local and non-local CNNs). We run the training procedure for one epoch, with the stepsize 10^{-3}.

---

[4]With slight abuse, we keep the same notation here as in the image denoising case for simplicity.

Regular CNN – $f_2 \circ f_1$ Graph CNN – $f_2 \circ f_{\mathrm{nl}} \circ f_1$
Analysis layer   Synthesis layer   Analysis layer   Synthesis layer
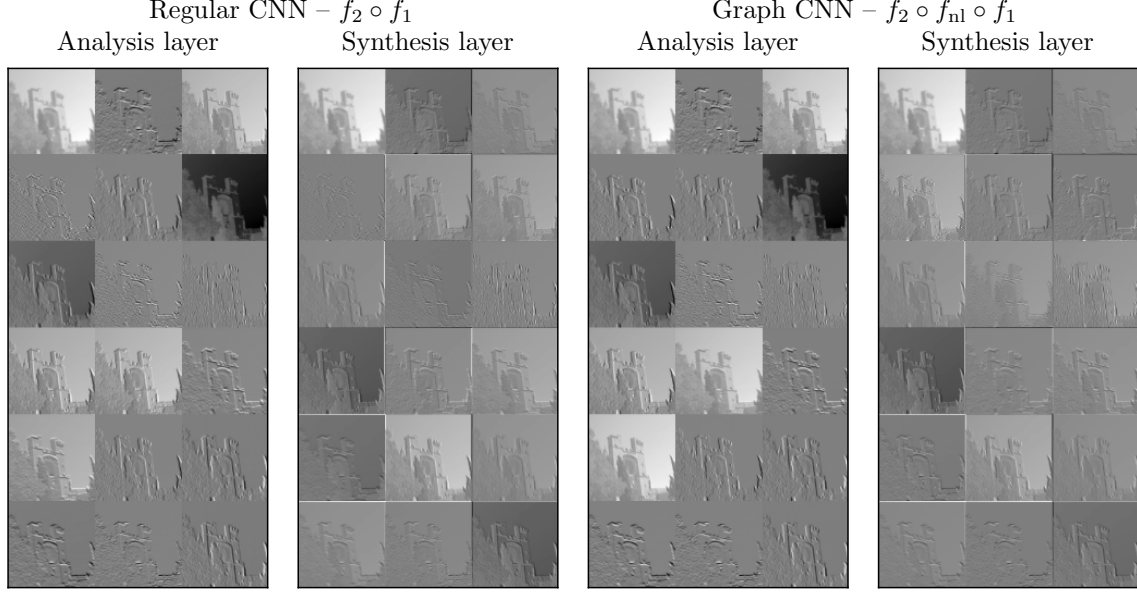

Figure 7. *First and third panels: feature maps obtained at the analysis layer of both trained networks computed from the same image (the non-linearity was not applied). Second and fourth panels: feature maps obtained by applying the adjoint of the filters at the synthesis layer of both trained networks.*

B.3. Results

Figure 5 presents the evolutions of the mean PSNR (for images) and the mean SNR (for audio), on the validation sets during training. One can already notice at this stage that the non-local networks outperform the local ones.

Image denoising – The average PSNR of the image test set is 23.10 dB before denoising. After denoising, we reach a PSNR of 29.13 dB and 29.42 dB with the local and non-local networks, respectively. The non-local network thus improves the denoising performance by 0.29 dB. For comparison, we also denoise the test set by soft-thresholding in the Haar wavelet basis. We first compute the threshold that maximises the PSNR on the validation set and then use it to denoise the test set. We reach an average PSNR of 26.78 dB only on the image test set with this method. We present in Fig. 6 an image before and after denoising with both networks. We notice that the non-local network allows a better recovery of the homogeneous regions in the image.

By analogy with dictionary learning and the terminology used in this field, we call hereafter the first layer f_1 of the CNNs used for denoising, the "analysis layer", and the last layer f_2, the "synthesis layer". The analysis layer filters the input image with a bank of filters and soft-thresholds the result, yielding sparse feature maps. The synthesis layer resynthesises an image from the feature maps using another bank of filters. We present in Fig. 7 some feature maps obtained at the analysis layer – without application of the non-linearity – for both trained networks. We remark that the learned filters act like wavelet filters in both cases. This is a well-known phenomenon observed in dictionary learning or at the first layer of deep CNNs. Our results are thus in line with these observations. In addition, we remark that the introduction of the non-local convolutions did not affect a lot the type of filters learned at the first layer. We also present the feature maps obtained by applying the adjoint of the filters at the synthesis layer of both trained networks. Note that in dictionary learning, the filters at the analysis are exactly the adjoint of the filters at the synthesis. Similarly, here, we notice that the (adjoint of the) synthesis filters act like wavelet filters, just as the analysis filters do. We also remark that the synthesis filters are quite similar in absence and presence of the non-local filtering.

Audio denoising – The average SNR of the audio test set is 0.85 dB before denoising. Denoising by the local CNN increases the average SNR to 8.44 dB, whereas, denoising with the non-local CNN

increases the SNR to 8.79 dB, *i.e.* a 0.35 dB improvement compared to the former. As for images, we also denoise the test set by soft-thresholding in the Haar wavelet basis with a threshold optimised on the validation set. We reach an average PSNR of 5.80 dB only on the test set with this method.