Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

Michaël Defferrard Xavier Bresson Pierre Vandergheynst

Department of Electrical Engineering
EPFL, Lausanne, Switzerland
{michael.defferrard,xavier.bresson,pierre.vandergheynst}@epfl.ch

July 1, 2016

Abstract

Convolutional neural networks (CNNs) have greatly improved state-of-the-art performances in a number of fields, notably computer vision and natural language processing. In this work, we are interested in generalizing the formulation of CNNs from low-dimensional regular Euclidean domains, where images (2D), videos (3D) and audios (1D) are represented, to high-dimensional irregular domains such as social networks or biological networks represented by graphs. This paper introduces a formulation of CNNs on graphs in the context of spectral graph theory. We borrow the fundamental tools from the emerging field of signal processing on graphs, which provides the necessary mathematical background and efficient numerical schemes to design localized graph filters efficient to learn and evaluate. As a matter of fact, we introduce the first technique that offers the same computational complexity than standard CNNs, while being universal to any graph structure. Numerical experiments on MNIST and 20NEWS demonstrate the ability of this novel deep learning system to learn local, stationary, and compositional features on graphs, as long as the graph is well-constructed.

1 Introduction

Convolutional neural networks [20] offer an efficient architecture to extract highly meaningful statistical patterns in large-scale and high-dimensional datasets. The key success of CNNs is its advanced ability to learn local stationary structures and compose them to form multi-scale hierarchical patterns. Precisely, CNNs extract the local stationarity property of the input data or signals by revealing local features that are shared across the data domain. These similar features are identified with localized convolutional filters or kernels, which are learned from the data. Convolutional filters are shift- or translation-invariant filters, meaning they are able to recognize identical features independently of their spatial locations. Localized kernels or compactly supported filters refer to filters that extract local features independently of the input data size, with a support size that can be much smaller than the input size.

The compositional local stationarity property of CNNs can be efficiently implemented on regular grids, like image and sound domains, as convolutional, downsampling and pooling operators are mathematically well-defined on such discretized Euclidean spaces. This has led to the breakthrough of CNNs in image, video, and sound recognition tasks [19] as these data lie on low-dimensional regular lattices. But not all data lie on regular grids. User data on social networks, gene data on biological

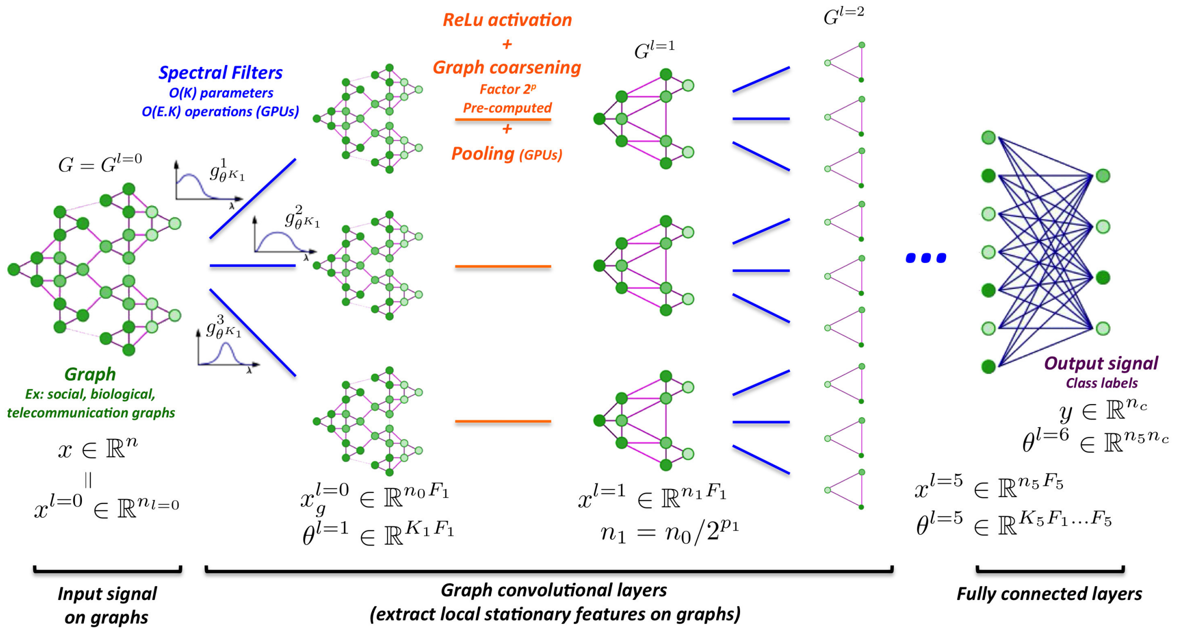

ReLu activation
+
Graph coarsening
Factor 2^p
Pre-computed
+
Pooling (GPUs)
Spectral Filters
O(K) parameters
O(E.K) operations (GPUs)
$G = G^{l=0}$
$G^{l=1}$
$G^{l=2}$
$g^1_{\theta K_1}$
$g^2_{\theta K_1}$
$g^3_{\theta K_1}$
Graph
Ex: social, biological, telecommunication graphs
$x \in \mathbb{R}^n$
$x^{l=0} \in \mathbb{R}^{n_{l=0}}$
$x_g^{l=0} \in \mathbb{R}^{n_0 F_1}$
$\theta^{l=1} \in \mathbb{R}^{K_1 F_1}$
$x^{l=1} \in \mathbb{R}^{n_1 F_1}$
$n_1 = n_0/2^{p_1}$
$x^{l=5} \in \mathbb{R}^{n_5 F_5}$
$\theta^{l=5} \in \mathbb{R}^{K_5 F_1 \dots F_5}$
Output signal
Class labels
$y \in \mathbb{R}^{n_c}$
$\theta^{l=6} \in \mathbb{R}^{n_5 n_c}$
Input signal on graphs
Graph convolutional layers (extract local stationary features on graphs)
Fully connected layers

Figure 1: Architecture of the proposed CNNs on graphs. Notation: l is the coarsening level, x^l are the downsampled signals at layer l, $\mathcal{G}^l$ is the coarser graph, $g_{\theta K_l}$ are the spectral filters at layer l, x_g^l are the filtered signals, p_l is the coarsening exponent, n_c is the number of classes, y is the output signal, and θ^l is the number of parameters to learn at l.

regulatory networks, log data on telecommunication networks, or text documents on words' embedding are important examples of data lying on irregular or non-Euclidean domains. Unfortunately, CNNs cannot be directly applied to such complex domains.

A generalization to irregular grids is not straightforward as the standard operators of convolution, downsampling, and pooling, are only defined for regular grids. This makes this extension challenging, both theoretically and implementation-wise. This work leverages the recent works of [12, 8, 4, 15, 14, 32] to introduce an efficient implementation of CNNs on irregular domains represented here as graphs. Graphs are universal representations of heterogeneous pairwise relationships between possibly high-dimensional data. Graphs can encode complex geometric structures, and can be studied with strong mathematical tools such as spectral graph theory [7], a blend between graph theory and harmonic analysis.

The major bottleneck of generalizing CNNs to graphs, and one of the primary goals of this work, is the definition of graph localized filters which are efficient to evaluate and learn. We will make use of recent tools developed in the context of graph signal processing (GSP) [32] to achieve such goals. Precisely, the main contributions of this work are summarized below:

1. **Spectral formulation.** A graph spectral theoretical formulation of CNNs on graphs built on established tools in GSP.

2. **Strictly localized filters.** Enhancing [4, 15], the proposed spectral filters are provable to be strictly localized in a ball of radius K, i.e. K hops from the central vertex.

3. **Low computational complexity.** The evaluation complexity of our filters is linear w.r.t. the filters support's size K and the number of edges $|\mathcal{E}|$. Importantly, as most real-world graphs are highly sparse, we have $|\mathcal{E}| \ll n^2$ and $|\mathcal{E}| = kn$ for the widespread k-nearest neighbor (NN) graphs, leading to a linear complexity w.r.t the input data size. Moreover, this method avoids the Fourier basis altogether, thus the expensive eigenvalue decomposition (EVD) necessary to

compute it as well as the need to store the basis, a matrix of size n^2. That is especially relevant when working with limited GPU memory. Besides the data, our method only requires to store the Laplacian, a sparse matrix of $|\mathcal{E}|$ non-zero values.

4. **Efficient pooling.** We propose an efficient pooling strategy on graphs which, after a rearrangement of the vertices as a binary tree structure, is analog to pooling of 1D signals.

5. **Experimental results.** We present many experiments that ultimately show that our formulation is (1) a useful model, (2) computationally efficient and (3) superior both in accuracy and complexity to the pioneer spectral graph CNNs introduced in [4, 15]. We also show that our graph formulation performs similarly as a classical CNNs on MNIST and study the impact of various graph constructions on classification performance.

Eventually, generic and efficient implementations of CNNs on graphs, as proposed here and related works [12, 8, 24, 23, 4, 15], is essential to pave the way to the development of a new class of deep learning techniques for graph-based data.

2 Related Works

2.1 Graph Signal Processing

GSP is an emerging field that aims at bridging the gap between signal processing techniques like wavelet analysis [22] and graph theory such as spectral graph analysis [3, 21]. A goal is to generalize fundamental analysis operations for signals from regular grids to irregular structures embodied by graphs. We refer the reader to [32] for an introduction of the field. Standard operations on grids such as convolution, translation, filtering, dilatation, modulation or downsampling do not extend directly to graphs and thus require new mathematical definitions while keeping the original intuitive concepts. In this context, the authors of [14, 9, 11] revisited the construction of wavelet operators on graphs. In [33, 28], the authors designed a technique to perform mutli-scale pyramid transforms on graphs. The works of [36, 26, 27] redefined uncertainty principles on graphs, and showed that intuitive concepts may be lost, but can also produce enhanced localization principles for signals on graphs. In [13], it was shown how to carry out lasso-based signal regularization on graphs, and studied the intertwined relationships between smoothness and sparsity on graphs. In [35], the authors investigated compressed sensing recovery conditions for graph spectral signal analysis.

In the future, we expect to further benefit from newly developed tools in GSP to enhance our mathematical formulation of CNNs on graphs. This paper introduces the mathematical and computational foundations of such models, while future developments may consider to enhance the building blocks, namely graph filtering and coarsening.

2.2 CNNs on Non-Euclidean Domains

Extending CNNs to non-Euclidean domains has been a recent line of work. A preliminary work was proposed with the so-called local reception fields [12, 8], successfully applied to image recognition. The main idea is to group together features based upon a measure of similarity such as to select a limited number of connections between two successive layers. While this model reduced the number of parameters by exploiting the locality assumption, it did not attempt to exploit any stationarity property of data, i.e. no weight-sharing strategy.

A first attempt to design CNNs on graphs was introduced in [4] by adopting a spatial approach. The idea was to construct a multiresolution graph by exploiting graph neighbourhood structures, and learned a deep neural network. However, such approach did not consider convolution operations, and thus no stationarity property.

ShapeNet in [24, 23] is a generalization of CNNs to 3D-meshes, a class of smooth low-dimensional non-Euclidean spaces. The authors used geodesic polar coordinates to define convolution operations on mesh patches, and formulated a deep learning architecture which allows comparison across different manifolds. They obtained state-of-the-art results for 3D shape recognition tasks.

In [6] and [30], the authors investigated the construction of Haar wavelet transforms on graphs using a deep hierarchical architecture. They applied the method to object recognition on sphere, and to sparse reconstruction of faces.

Finally, [4] and [15] introduced a generalization of CNNs on graphs using graph counterparts of grid convolution and downsampling. They were able to learn convolutional filters on graphs using the convolution theorem [22] that defines convolutions as linear operators that diagonalize in the Fourier basis (represented by the eigenvectors of the Laplacian operator). They were thus the first to introduce a spectral formulation to extend CNNs on graphs. They also introduced a strategy to learn the graph structure from the data, and applied their model on image recognition, text categorization and bioinformatics. This approach does however not scale up due to the necessary multiplications by the graph Fourier basis, a matrix of n^2 coefficients, where n is the data dimensionality. Despite the cost of computing this matrix, which requires an EVD on the graph Laplacian, the dominant cost is the need to multiply the data by this matrix twice (forward and inverse Fourier transforms) at a cost of $\mathcal{O}(n^2)$ operations per forward and backward pass, a computational bottleneck already identified by the authors. Besides, as they rely on smoothness (via a spline parametrization) in the Fourier domain to bring localization in the vertex domain, their model does not provide a precise control over the local support of their kernels, which is essential to learn localized filters. Our technique leverages on this work, and we will show how to overcome these limitations and beyond.

3 Proposed Technique

Generalizing CNNs to graphs requires three fundamental steps. First step is the design of localized convolutional filters on graphs. Second step is a graph coarsening procedure that groups together similar features on graphs. And the last step is the graph pooling operation that trades spatial resolution for higher filter resolution.

3.1 Learning Fast Localized Spectral Filters

The goal of this section is to define fast convolutional localized filters on graphs. There are two strategies to define convolutional localized filters; either from a spatial approach or from a spectral approach. By construction, spatial approaches provide filter localization via the finite size of the kernel. However, although graph convolution directly in the spatial domain is conceivable, it also faces the challenge of matching local neighbourhoods, as pointed out in [4]. Consequently, there is no unique mathematical definition of spatial translations on graphs from a spatial perspective. On the other side, a spectral approach provides a well-defined translation operator on graphs via convolutions with a Kronecker delta function implemented in the spectral domain [32]. However, a filter defined in the spectral domain is not naturally localized and translations are costly with $\mathcal{O}(n^2)$ operations due to multiplications with the graph Fourier basis. Both limitations can be overcome with a special choice of filter parametrization, as explained below.

Graph Fourier Transform. We are interested in processing signals defined on undirected and connected graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$, where $\mathcal{V}$ is a finite set of vertices with $|\mathcal{V}| = n$, $\mathcal{E}$ is a set of edges and W is a weighted adjacency matrix encoding the connection weight between two vertices. A signal $x : \mathcal{V} \to \mathbb{R}$ defined on the nodes of the graph may be regarded as a vector $x \in \mathbb{R}^n$ where x_i is the value of x at the i^{th} node. An essential operator in spectral graph analysis is the graph Laplacian [7], which combinatorial definition is $L = D - W \in \mathbb{R}^{n \times n}$ where $D \in \mathbb{R}^{n \times n}$ is the diagonal degree matrix with $D_{ii} = \sum_j W_{ij}$, and its normalized definition is $L = I_n - D^{-1/2} W D^{-1/2}$ where I_n is the identity matrix. As L is a real symmetric positive semidefinite matrix, it has a complete set of orthonormal eigenvectors $\{u_l\}_{l=0}^{n-1} \in \mathbb{R}^n$, known as the graph Fourier modes, and their associated ordered real nonnegative eigenvalues $\{\lambda_l\}_{l=0}^{n-1}$, identified as the frequencies of the graph. The Laplacian is indeed diagonalized by the Fourier basis $U = [u_0, \dots, u_{n-1}] \in \mathbb{R}^{n \times n}$ such that $L = U \Lambda U^T$ where $\Lambda = \mathrm{diag}([\lambda_0, \dots, \lambda_{n-1}]) \in \mathbb{R}^{n \times n}$. We can now define the graph Fourier transform (GFT) of a spatial signal $x \in \mathbb{R}^n$ as $\hat{x} = U^T x \in \mathbb{R}^n$, and its inverse as $x = U\hat{x}$ [32]. Similarly to the Fourier transform on Euclidean spaces, the GFT enables the formulation of fundamental operations such as filtering.

Spectral filtering of graph signals. As we cannot express a meaningful translation operator in the vertex domain, the convolution operator on graph $*_\mathcal{G}$ is defined in the Fourier domain such that $x *_\mathcal{G} y = U((U^T x) \odot (U^T y))$, where $\odot$ is the element-wise Hadamard product. It follows that a signal x is filtered by g_θ as

$$y = g_\theta(L)x = g_\theta(U \Lambda U^T)x = U g_\theta(\Lambda) U^T x. \tag{1}$$

A non-parametric filter, i.e. a filter whose parameters are all free, would be defined as

$$g_\theta(\Lambda) = \mathrm{diag}(\theta), \tag{2}$$

where the parameter $\theta \in \mathbb{R}^n$ is a vector of Fourier coefficients.

Polynomial parametrization for localized filters. There are however two limitations with non-parametric filters: (1) they are not localized in space and (2) their learning complexity is in $\mathcal{O}(n)$, the dimensionality of the data. These issues can be overcome with the use of Laplacian-based polynomial spectral filters:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k, \tag{3}$$

where the parameter $\theta \in \mathbb{R}^K$ is a vector of polynomial coefficients. The value at vertex j of the filter g_θ centered at vertex i is given by $(g_\theta(L)\delta_i)_j = (g_\theta(L))_{i,j} = \sum_k \theta_k (L^k)_{i,j}$, where the kernel is translated via a convolution with a Kronecker delta function $\delta_i \in \mathbb{R}^n$. By [14, Lemma 5.2], $d_\mathcal{G}(i,j) > K$ implies $(L^K)_{i,j} = 0$, where $d_\mathcal{G}$ is the shortest path distance, i.e. the minimum number of edges connecting two vertices on the graph. Consequently, spectral filters represented by K^{th}-order polynomials of the Laplacian are exactly K-localized. Besides, their learning complexity is $\mathcal{O}(K)$, the support size of the filter, and thus the same complexity as in standard CNNs.

Recursive formulation for fast filtering. While we have shown how to learn localized filters with K parameters, the cost to filter a signal x as $y = U g_\theta(\Lambda) U^T x$ is still high with $\mathcal{O}(n^2)$ operations because of the multiplications with the Fourier basis U. A solution of this problem is to parametrize $g_\theta(L)$ as a polynomial function that can be computed recursively from L, as K multiplications by a sparse L costs $\mathcal{O}(K|\mathcal{E}|) \ll \mathcal{O}(n^2)$. One such polynomial, traditionally used in GSP to approximate kernels (like wavelets), is the Chebyshev expansion [14]. Another option, the Lanczos algorithm [34], which constructs an orthonormal basis of the Krylov subspace $\mathcal{K}_K(L,x) = \mathrm{span}\{x, Lx, \dots, L^{K-1}x\}$, seems attractive because of the coefficients' independence. It is however more convoluted and thus left as a future work.

Recall that the Chebyshev polynomial $T_k(x)$ of order k may be generated by the stable recurrence relation $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_0 = 1$ and $T_1 = x$. These polynomials form an orthogonal basis for $L^2([-1,1], dy/\sqrt{1-y^2})$, the Hilbert space of square integrable functions with respect to the measure $dy/\sqrt{1-y^2}$. A filter can thus be parametrized as a truncated expansion of order $K-1$ such that

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda}), \tag{4}$$

where the parameter $\theta \in \mathbb{R}^K$ is a vector of Chebyshev coefficients and $T_k(\tilde{\Lambda}) \in \mathbb{R}^{n \times n}$ is the Chebyshev polynomial of order k evaluated at $\tilde{\Lambda} = 2\Lambda/\lambda_{max} - I_n$, a diagonal matrix of scaled eigenvalues that lie in $[-1,1]$.

The filtering operation can then be written as $y = g_\theta(L)x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x$, where $T_k(\tilde{L}) \in \mathbb{R}^{n\times n}$ is the Chebyshev polynomial of order k evaluated at the scaled Laplacian $\tilde{L} = 2L/\lambda_{max} - I_n$. Denoting $\bar{x}_k = T_k(\tilde{L})x \in \mathbb{R}^n$, we can use the recurrence relation to compute $\bar{x}_k = 2\tilde{L}\bar{x}_{k-1} - \bar{x}_{k-2}$ with $\bar{x}_0 = x$ and $\bar{x}_1 = \tilde{L}x$. The entire filtering operation

$$y = g_\theta(L)x = [\bar{x}_0, \ldots, \bar{x}_{K-1}]\theta \tag{5}$$

then costs $\mathcal{O}(K|\mathcal{E}|)$ operations.

Learning filters. The j^{th} output feature map of the sample s is given by

$$y_{s,j} = \sum_{i=1}^{F_{in}} g_{\theta_{i,j}}(L)x_{s,i} \in \mathbb{R}^n, \tag{6}$$

where the $x_{s,i}$ are the input feature maps and the $F_{in} \times F_{out}$ vectors of Chebyshev coefficients $\theta_{i,j} \in \mathbb{R}^K$ are the layer's trainable parameters. When training multiple convolutional layers with the backpropagation algorithm, one needs the two gradients:

$$\frac{\partial E}{\partial \theta_{i,j}} = \sum_{s=1}^{S} [\bar{x}_{s,i,0}, \ldots, \bar{x}_{s,i,K-1}]^T \frac{\partial E}{\partial y_{s,j}} \qquad \text{and} \qquad \frac{\partial E}{\partial x_{s,i}} = \sum_{j=1}^{F_{out}} g_{\theta_{i,j}}(L) \frac{\partial E}{\partial y_{s,j}}, \tag{7}$$

where E is the loss energy over a mini-batch of S samples. Each of the above three computations boils down to K sparse matrix-vector multiplications and one dense matrix-vector multiplication for a cost of $\mathcal{O}(K|\mathcal{E}|F_{in}F_{out}S)$ operations. These can be efficiently computed on parallel architectures by leveraging tensor operations. Eventually, $[\bar{x}_{s,i,0}, \ldots, \bar{x}_{s,i,K-1}]$ only need to be computed once.

3.2 Graph Coarsening

Local stationarity property of data is extracted via localized convolutional kernels. We are now interested to extract the multi-scale hierarchical composition property of data. In standard CNNs, this is efficiently achieved via grid subsampling and pooling, which trades spatial resolution with feature resolution reducing the learning complexity without compromising the system performances. In contrast with regular domains, the subsampling operation on graphs or graph coarsening is not mathematically sound. It requires to construct meaningful neighborhoods on graphs where similar vertices are clustered together. Doing this for multiple layers is equivalent to construct a multi-scale clustering of the graph that preserves local geometric structures. It is however well-known that graph clustering is a NP-hard problem [5] and that approximations must be used. While there exist many clustering techniques (e.g. the popular spectral clustering [21]), we are most interested in multilevel clustering algorithms, where each level produces a coarser graph which corresponds to the data domain seen at a different resolution. Moreover, clustering techniques that reduce the size of the graph by a factor two at each level offers a precise control on the coarsening and pooling size. In this work, we make use of the coarsening phase of the Graclus multilevel clustering algorithm [10], which has been shown to be extremely efficient at clustering a large variety of graphs. We briefly review it below. Algebraic multigrid techniques on graphs [29] and the Kron reduction [33] are two methods worth exploring in future works.

Graclus [10], built on Metis [17], uses a greedy algorithm to compute successive coarser versions of a given graph and is able to minimize several popular spectral clustering objectives. We chose the normalized cut [31], which is an excellent clustering energy. Graclus' greedy rule consists, at any given coarsening level, in picking an unmarked vertex i and matching it with one of its unmarked neighbors j that maximizes the local normalized cut $W_{ij}(1/d_i + 1/d_j)$. The two matched vertices are then marked and the coarsened weights are set as the sum of their weights. The matching is repeated until all nodes have been explored. This is an extremely fast coarsening scheme which divides the number of nodes by approximately two (there may exist a few singletons, non-matched nodes) from one level to the next coarser level.

3.3 Fast Pooling of Graph Signals

As standard CNNs, pooling operations reduce the spatial resolution allowing higher filter resolution. Pooling operations, such as max pooling or average pooling, are carried out many times during the optimization, and thus must be as efficient as possible. After the graph coarsening operation in the previous section, the graph and its coarsened versions have an unstructured arrangement of the vertices. Hence, if the pooling is directly applied then it would need a table to store all matched vertices, which would result in memory consuming, slow, and not (easily) parallelizable pooling operations.

It is however possible to structure the arrangement of the vertices, and achieve a graph pooling operation as efficient as a 1D grid pooling. We proceed as follows. After coarsening, each node has either two parents, if it was matched at the finer level, or one, if it was not, i.e the node was a singleton. From the coarsest to finest level, fake nodes, i.e. disconnected nodes, are added to pair with the singletons such that each node has two children. This structure is a balanced binary tree: (1) regular nodes (and singletons) either have two regular nodes (e.g. level 1 vertex 0 in Figure 2) or (2) one singleton and a fake node as children (e.g. level 2 vertex 0 in Figure 2), and (3) fake nodes always have two fake nodes as children (e.g. level 1 vertex 1 in Figure 2). Input signals are initialized with a neutral value at the fake nodes, e.g. 0 when using a ReLU activation with max pooling. Because these nodes are disconnected, filtering does not impact the initial neutral value. While those fake nodes do artificially increase the dimensionality thus the computational cost, we found that, in practice, the number of singletons left by Graclus is quite low compared to the number of vertices.

Arbitrarily ordering the nodes at the coarsest level, then propagating this ordering to the finest levels, i.e. node k has nodes $2k$ and $2k+1$ as children, produces a regular ordering in the finest level. Regular in the sense that adjacent nodes are hierarchically merged at coarser levels. Pooling such a rearranged graph signal is analog to pooling a regular 1D signal. Figure 2 shows an example of the whole coarsening and pooling process. This regular arrangement makes the pooling operation very efficient and satisfies parallel architectures such as GPUs as memory accesses are local, i.e. matched nodes do not have to be fetched.

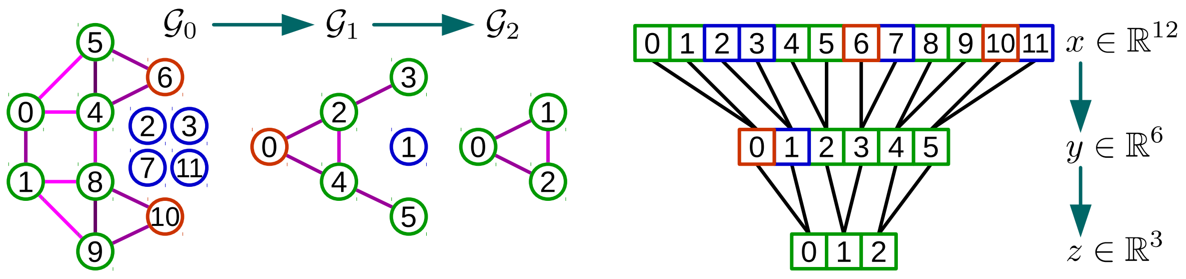


Figure 2: **Example of Graph Coarsening and Pooling.** Let us carry out a max pooling of size 4 (or two poolings of size 2) on a signal $x \in \mathbb{R}^8$ living on $\mathcal{G}_0$, the finest graph given as input. Note that it originally possesses $n_0 = |\mathcal{V}_0| = 8$ vertices, arbitrarily ordered. For a pooling of size 4, two coarsenings of size 2 are needed: Graclus first gives $\mathcal{G}_1$ of size $n_1 = |\mathcal{V}_1| = 5$, then $\mathcal{G}_2$ of size $n_2 = |\mathcal{V}_2| = 3$, the coarsest graph. Sizes are thus set to $n_2 = 3$, $n_1 = 6$, $n_0 = 12$ and fake nodes (in blue) are added to $\mathcal{V}_1$ (1 node) and $\mathcal{V}_0$ (4 nodes) to pair with the singeltons (in orange), such that each node has exactly two children. Nodes in $\mathcal{V}_2$ are then arbitrarily ordered and nodes in $\mathcal{V}_1$ and $\mathcal{V}_0$ are ordered consequently. At that point the arrangement of vertices in $\mathcal{V}_0$ permits a regular 1D pooling on $x \in \mathbb{R}^{12}$ such that $z = [\max(x_0, x_1), \max(x_4, x_5, x_6), \max(x_8, x_9, x_{10})] \in \mathbb{R}^3$, where the signal components x_2, x_3, x_7, x_{11} are set to a neutral value.

4 Numerical Experiments

All experiments were performed with TensorFlow, an open-source library for numerical computation using data flow graphs, especially suited for deep learning [1]. It features various backends, notably CUDA to compute on Nvidia GPUs. All computations are carried on an Nvidia Tesla K40c GPU.

In the sequel, we refer to filters defined by (2) as *Non-Param*, i.e. the spatially non-localized filters, and the proposed filters defined by (4) as *Chebyshev*. Filters referred to as *Spline* are defined by

$$g_\theta(\Lambda) = B\theta, \tag{8}$$

where $B \in \mathbb{R}^{n\times K}$ is the cubic B-spline basis and the parameter $\theta \in \mathbb{R}^K$ is a vector of control points, as proposed in [4, 15]. We use for all techniques the advanced Graclus coarsening algorithm introduced in Section 3.2 rather than the simple agglomerative method of [4, 15]. The motivation is to compare the quality of learned filters, not the coarsening algorithms.

We use the following notation when describing network architectures: FCk denotes a fully connected layer with k hidden units, Pk denotes a (graph or classical) pooling layer of size and stride k, Ck denotes a convolutional layer with k feature maps and GCk denotes a graph convolutional layer with $F_{out} = k$ feature maps (with Chebyshev filters if not specified otherwise). All FCk, Ck and GCk layers are followed by a ReLU activation $\max(x, 0)$. The final layer is always a softmax regression and the loss energy E is the cross-entropy loss with an ℓ_2 regularization on the weights of all FCk layers. Mini-batches are of size $S = 100$.

4.1 Revisiting Standard CNNs on MNIST

To validate our model, we first apply it to the Euclidean case with the benchmark MNIST classification problem [20]. In this situation, the graph is simply a k-NN graph of the Euclidean 2D grid. This is an important sanity check for our model, which must be able to extract features on any graph, including the regular 2D grid on which images reside.

Model	Architecture	Accuracy
Classical CNN	C32-P4-C64-P4-FC512	99.33
Proposed graph CNN	GC32-P4-GC64-P4-FC512	99.14

Table 1: Classification accuracies for the proposed model and a classical CNN on MNIST.

We recall that MNIST is a dataset of 70,000 digit numbers represented on a 2D grid of size 28×28, such that data points lie on a space of 784 dimensions. For our graph model, we construct a 8-NN graph of the 2D Euclidean grid, which produces a graph of $n = |\mathcal{V}| = 976$ nodes (784 pixels and 192 fake nodes as explained in Section 3.3) and $|\mathcal{E}| = 3198$ edges.

Table 1 confirms the ability of our model to learn localized filters as it achieves a performance very close to the classical CNNs with the same LeNet-5-like architecture. The LeNet-5-like network architecture and the following hyper-parameters are borrowed from the TensorFlow MNIST tutorial[1]: dropout probability of 0.5, regularization weight of 5×10^{-4}, initial learning rate of 0.03, learning rate decay of 0.95, momentum of 0.9. Convolutional layer have filters of size 5×5 while graph convolution layers have the same support of $K = 5^2 = 25$. All models were trained for 20 epochs.

4.2 Text Categorization with 20NEWS

To demonstrate the versatility of our model to work with graphs generated from unstructured data, we applied our technique to the text categorization problem with the 20NEWS dataset.

[1]https://www.tensorflow.org/versions/r0.8/tutorials/mnist/pros

Model	Accuracy
Linear SVM	65.90
Multinomial Naive Bayes	68.51
Softmax	66.28
FC2500	64.64
FC2500-FC500	65.76
GC32	68.26

Table 2: Classification accuracies for the proposed model and a other algorithms on 20NEWS.

20NEWS consists of 18,846 (11,314 for training and 7,532 for testing) text documents associated with 20 classes [16]. We extracted the 10,000 most common words from the 93,953 unique words in this corpus. Each document is represented using the bag-of-words model, normalized across words. To test our model, we constructed a 16-NN graph of the word2vec [25] embedding of those words, which produced a graph of $n = |\mathcal{V}| = 10,000$ nodes and $|\mathcal{E}| = 132,834$ edges. All CNN models were trained for 20 epochs by the Adam optimizer [18] with a learning rate of 0.1, without regularization nor dropout. The filter support was set to $K = 5$.

Table 2 shows that CNNs on graphs provide decent performances. While it does not outperform the kernel-based method of multinomial naive Bayes classifier on this small dataset, it does defeat fully connected networks, which require much more parameters to be learned. More importantly, these results verify the validity of the statistical assumptions made for the data, that are locality and stationarity, and which are at the core of the design of any CNN technique. While we know that these data properties are true for low-dimensional Euclidean data like audios, images and videos, we also show experimentally that they are also satisfied for text documents as long as the graph is properly constructed. Section 4.5 will study the influence of the graph quality.

4.3 Numerical Comparison between Spectral Filters

In this section, we compare the proposed spectral filters to the non-parametric filters, and those proposed in [4, 15] on the MNIST and 20NEWS datasets. Table 3 reports that the proposed kernel parametrization outperforms [4, 15] as well as the non-parametric filters, which are not localized and require $\mathcal{O}(n)$ parameters to learn.

		Accuracy		
Dataset	Architecture	Non-Param (2)	Spline (8) [4, 15]	Chebyshev (4)
MNIST	GC10	95.75	97.26	97.48
MNIST	GC32-P4-GC64-P4-FC512	96.28	97.15	99.14

Table 3: Comparison of accuracy results for different types of spectral filters. Spline and Chebyshev filters use $K = 25$.

4.4 Computational Efficiency

Figure 3 validates the low computational complexity of the proposed CNN technique on graphs. The training time of our model scales as $\mathcal{O}(n)$, while [4, 15] scales as $\mathcal{O}(n^2)$. Moreover, Figure 4 gives a sense of how the validation accuracy as well as the loss energy converges w.r.t. the three filter definitions. Finally, Table 4 compares training time on CPU and GPU. The fact that we observe the same order of speedup as classical CNNs exemplifies the natural parallelization opportunity offered by our model. That is possible because our method relies solely on matrix multiplications which are efficiently implemented by cuBLAS, the linear algebra routines provided by NVIDIA.

		Time (ms)		
Model	Architecture	CPU	GPU	Speedup
Classical CNN	C32-P4-C64-P4-FC512	210	31	6.77x
Proposed graph CNN	GC32-P4-GC64-P4-FC512	200	25	8.00x

Table 4: Time to process a mini-batch of 100 MNIST images. That is the total training time divided by the number of gradient steps.

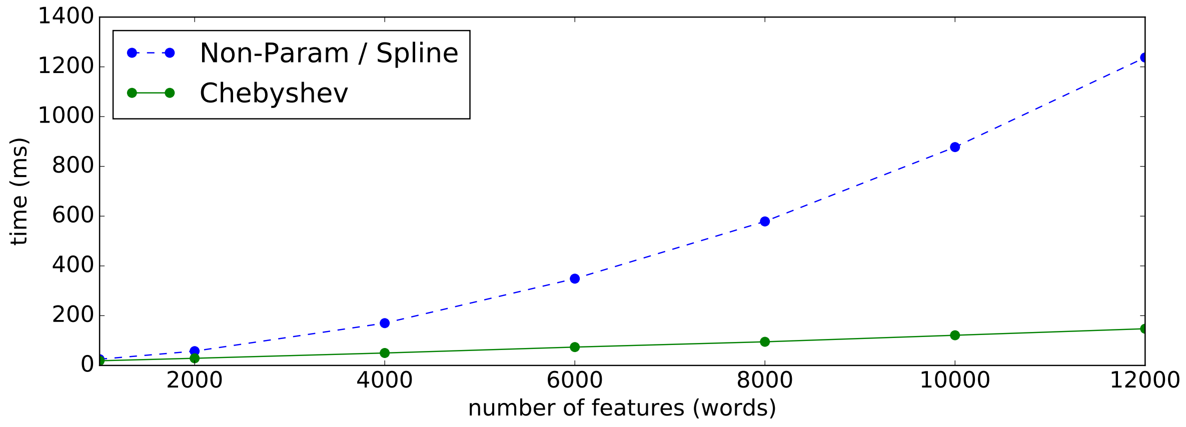
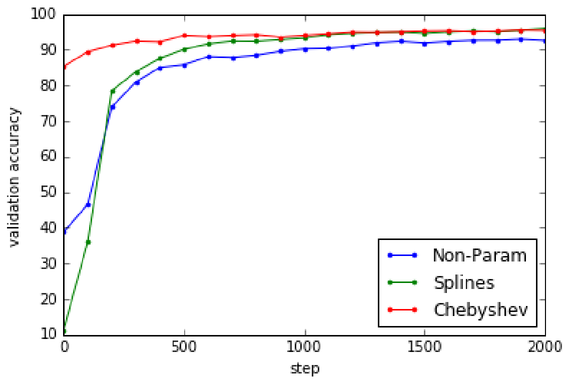
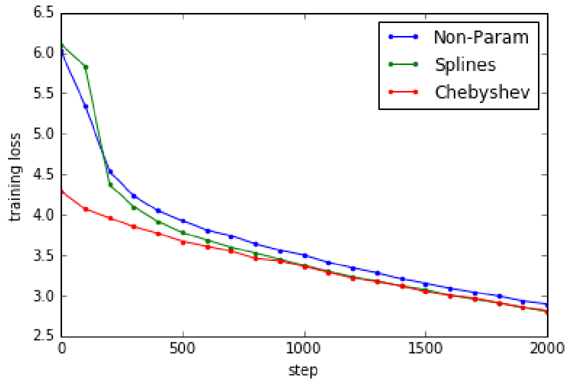


Figure 3: Training time over a mini-batch of 100 20NEWS documents w.r.t. n, the number of selected words, for the three considered filters. The architecture is GC32 with $K = 5$ for Spline and Chebyshev.



(a)



(b)

Figure 4: Plots of accuracy (a) and energy loss (b) for the first 2000 iterations.

4.5 Influence of Graph Quality

For the proposed method to be successful, the statistical assumptions of locality, stationarity, and compositionality regarding the data must be fulfilled on the graph where the data resides. Therefore, the classification performance, the quality of learned filters, all critically depends on the quality of the graph. For data lying on Euclidean space, experiments in Section 4.1 show that a simple k-NN graph of the grid is good enough to recover almost exactly the performance of standard CNNs. We also noticed that the value of k does not have a strong influence on the results. We can witness the importance of a graph satisfying the data assumptions by comparing its performance with a random graph. Table 5 reports a large drop of accuracy when using a random graph, that is when the data structure is lost and the convolutional layers are not useful anymore to extract meaningful features.

Architecture	8-NN on 2D Euclidean grid	random
GC32	97.40	96.88
GC32-P4-GC64-P4-FC512	99.14	95.39

Table 5: Classification accuracies with different graph constructions on MNIST.

While images naturally lie on Euclidean 2D spaces, represented by graphs of regular grids, this is not the case for text documents. For our method to be applied to this class of data, a feature graph does have to be built. We investigate here 5 16-NN similarity graphs of words. The simplest option is to represent each word as its corresponding column in the bag-of-words matrix. Another approach is to learn embeddings on the corpus with word2vec [25] or to use pre-trained embeddings on Google News (given by the authors). As the dataset gets larger (in number of samples and dimensionality), it is often not an option to compute the distance between all features, such that an approximate nearest neighbors algorithm shall be used. We used the LSHForest [2] on the learned word2vec embedding. Table 6 reports the classification results. It shows that the graph built on the learned word2vec embedding is the best at capturing the local and stationarity properties of text documents. It is worth noticing that the approximate k-NN graph constructed from it is almost as bad as the random graph, meaning that it may be a better strategy to learn a good low-dimensional embedding first and then construct an exact k-NN graph from this embedding.

	word2vec			
bag-of-words	pre-trained	learned	approximate	random
67.50	66.98	68.26	67.86	67.75

Table 6: Classification accuracies of GC32 with different graph constructions on 20NEWS.

5 Conclusion and Future Work

We have introduced an efficient implementation of CNNs on non-Euclidean manifolds represented by graphs using tools from GSP. Experiments have shown the ability of the model to extract local and stationary features through the graph convolutional layers.

Compared with the first works of spectral graph CNNs introduced in [4, 15], our model provides a strict control over the local support of filters, is computationally more efficient by avoiding an explicit use of the Graph Fourier basis, and experimentally shows a better test accuracy. Besides, we addressed the three concerns raised by [15]. First, we introduced a model whose computational complexity is linear with the dimensionality of the data. Second, we confirmed that the quality of the input graph is of paramount importance. Along this line, the idea of [15] to learn the input graph from data is highly meaningful. And a natural and future approach would be to alternate the learning of the CNN parameters and the graph, as a virtuous circle. Third, we showed that the statistical assumptions of local stationarity and compositionality made by the model are verified for text documents as long as the graph is well constructed.

Future works will investigate two directions. On one hand, we will explore applications of this generic model to important fields where the data naturally lie on non-artificially constructed graphs such as social networks, biological networks, or telecommunication networks. It makes actually more sense to apply this generalized CNN framework to natural network-based data, rather than artificially created networks which quality may vary as seen in the experiments. On the other hand, we will also improve this framework with tools developed in GSP, including graph wavelet operators [14, 9, 11, 6, 30], spectral graph coarsening techniques [33], and improved localization properties [36, 26, 27].

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.

[2] M. Bawa, T. Condie, and P. Ganesan. LSH Forest: Self-Tuning Indexes for Similarity Search. In *International Conference on World Wide Web*, pages 651–660, 2005.

[3] M. Belkin and P. Niyogi. Towards a Theoretical Foundation for Laplacian-based Manifold Methods. *Journal of Computer and System Sciences*, 74(8):1289–1308, 2008.

[4] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral Networks and Deep Locally Connected Networks on Graphs. *arXiv:1312.6203*, 2013.

[5] T.N. Bui and C. Jones. Finding Good Approximate Vertex and Edge Partitions is NP-hard. *Information Processing Letters*, 42(3):153–159, 1992.

[6] X. Chen, X. Cheng, and S. Mallat. Unsupervised Deep Haar Scattering on Graphs. In *Neural Information Processing Systems*, pages 1709–1717, 2014.

[7] F. R. K. Chung. *Spectral Graph Theory*, volume 92. American Mathematical Society, 1997.

[8] A. Coates and A.Y. Ng. Selecting Receptive Fields in Deep Networks. In *Neural Information Processing Systems (NIPS)*, pages 2528–2536, 2011.

[9] R.R. Coifman and S. Lafon. Diffusion Maps. *Applied and Computational Harmonic Analysis*, 21(1):5–30, 2006.

[10] I. Dhillon, Y. Guan, and B. Kulis. Weighted Graph Cuts Without Eigenvectors: A Multilevel Approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 29(11):1944–1957, 2007.

[11] M. Gavish, B. Nadler, and R. Coifman. Multiscale Wavelets on Trees, Graphs and High Dimensional Data: Theory and Applications to Semi Supervised Learning. In *International Conference on Machine Learning (ICML)*, pages 367–374, 2010.

[12] K. Gregor and Y. LeCun. Emergence of Complex-like Cells in a Temporal Product Network with Local Receptive Fields. In *arXiv:1006.0448*, 2010.

[13] D. Hammond, K. Raoaroor, L. Jacques, and P. Vandergheynst. Image Denoising with Nonlocal Spectral Graph Wavelets. In *SIAM Conference on Imaging Science*, 2010.

[14] D. Hammond, P. Vandergheynst, and R. Gribonval. Wavelets on Graphs via Spectral Graph Theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[15] M. Henaff, J. Bruna, and Y. LeCun. Deep Convolutional Networks on Graph-Structured Data. *arXiv:1506.05163*, 2015.

[16] T. Joachims. A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. *Carnegie Mellon University, Computer Science Technical Report*, CMU-CS-96-118, 1996.

[17] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing (SISC)*, 20(1):359–392, 1998.

[18] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980*, 2014.

[19] Y. LeCun, Y. Bengio, and G. Hinton. Deep Learning. *Nature*, 521(7553):436–444, 2015.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. In *Proceedings of the IEEE, 86(11)*, pages 2278–2324, 1998.

[21] U. Von Luxburg. A Tutorial on Spectral Clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[22] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic press, 1999.

[23] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst. ShapeNet: Convolutional Neural Networks on Non-Euclidean Manifolds. *arXiv:1501.06297*, 2015.

[24] J. Masci, D. Boscaini, M.M. Bronstein, and P. Vandergheynst. Geodesic Convolutional Neural Networks on Riemannian Manifolds. In *Workshop on 3D Representation and Recognition (3dRR)*, 2015.

[25] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Estimation of Word Representations in Vector Space. In *International Conference on Learning Representations*, 2013.

[26] B. Pasdeloup, R. Alami, V. Gripon, and M. Rabbat. Toward an Uncertainty Principle for Weighted Graphs. In *Signal Processing Conference (EUSIPCO)*, pages 1496–1500, 2015.

[27] N. Perraudin, B. Ricaud, D. Shuman, and P. Vandergheynst. Global and Local Uncertainty Principles for Signals on Graphs. *arXiv:1603.03030*, 2016.

[28] I. Ram, M. Elad, and I. Cohen. Generalized Tree-based Wavelet Transform. *IEEE Transactions on Signal Processing,*, 59(9):4199–4209, 2011.

[29] D. Ron, I. Safro, and A. Brandt. Relaxation-based Coarsening and Multiscale Graph Organization. *SIAM Iournal on Multiscale Modeling and Simulation*, 9:407–423, 2011.

[30] R. Rustamov and L.J. Guibas. Wavelets on Graphs via Deep Learning. pages 998–1006, 2013.

[31] J. Shi and J. Malik. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 22(8):888–905, 2000.

[32] 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*, 30(3):83–98, 2013.

[33] D.I. Shuman, M.J. Faraji, and P. Vandergheynst. A Multiscale Pyramid Transform for Graph Signals. *IEEE Transactions on Signal Processing*, 64(8):2119–2134, 2016.

[34] A. Susnjara, N. Perraudin, D. Kressner, and P. Vandergheynst. Accelerated Filtering on Graphs using Lanczos Method. *preprint arXiv:1509.04537*, 2015.

[35] N. Tremblay, G. Puy, R. Gribonval, and P. Vandergheynst. Compressive Spectral Clustering. In *International Conference on Machine Learning (ICML)*, 2016.

[36] M. Tsitsvero and S. Barbarossa. On the Degrees of Freedom of Signals on Graphs. In *Signal Processing Conference (EUSIPCO)*, pages 1506–1510, 2015.