CONVOLUTIONAL NEURAL NETWORKS VIA NODE-VARYING GRAPH FILTERS

Fernando Gama[†]*, Geert Leus*[‡]*, Antonio G. Marques*[∗] *and Alejandro Ribeiro*[†]

† Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, USA
‡ Department of Microelectronics, Delft University of Technology, Delft, The Netherlands
∗ Department of Signal Theory and Communications, King Juan Carlos University, Madrid, Spain

ABSTRACT

Convolutional neural networks (CNNs) are being applied to an increasing number of problems and fields due to their superior performance in classification and regression tasks. Since two of the key operations that CNNs implement are convolution and pooling, this type of networks is implicitly designed to act on data described by regular structures such as images. Motivated by the recent interest in processing signals defined in irregular domains, we advocate a CNN architecture that operates on signals supported on graphs. The proposed design replaces the classical convolution not with a node-invariant graph filter (GF), which is the natural generalization of convolution to graph domains, but with a *node-varying* GF. This filter extracts different local features without increasing the output dimension of each layer and, as a result, bypasses the need for a pooling stage while involving only local operations. A second contribution is to replace the node-varying GF with a *hybrid node-varying* GF, which is a new type of GF introduced in this paper. While the alternative architecture can still be run locally without requiring a pooling stage, the number of trainable parameters is smaller and can be rendered independent of the data dimension. Tests are run on a synthetic source localization problem and on the `20NEWS` dataset.

***Index Terms*—** Convolutional neural networks, network data, graph signal processing, node-varying graph filters.

1. INTRODUCTION

Convolutional neural networks (CNNs) have shown remarkable performance in a wide array of inference and reconstruction tasks [1], in fields as diverse as pattern recognition, computer vision and medicine [2–4]. The objective of CNNs is to find a computationally feasible architecture capable of reproducing the behavior of a certain unknown function. Typically, CNNs consist of a succession of layers, each of which performs three simple operations – usually on the output of the previous layer – and feed the result into the next layer. These three operations are: 1) convolution, 2) application of a nonlinearity, and 3) pooling or downsampling. Because the classical convolution and downsampling operations are defined for regular (grid-based) domains, CNNs have been applied to act on data modeled by such a regular structure, like time or images.

However, an accurate description of modern datasets such as those in social networks or genetics [5, 6] calls for more general irregular structures. A framework that has been gaining traction to tackle these problems is that of graph signal processing (GSP) [7–9]. GSP postulates that data can be modeled as a collection of values associated with the nodes of a graph, whose edges describe pairwise relationships between the data. By exploiting the interplay between the data and the graph, traditional signal processing concepts such as the Fourier transform, sampling and filtering have been generalized under the GSP framework to operate on a broader array of datasets [10–12].

Motivated by the success of CNNs and the need to deal with irregular domains, recent efforts have been made to extend CNNs to work with data (signals) defined on manifolds and graphs [13]. Since in the GSP literature the notion of convolution is generalized to that of node-invariant graph filters (GFs) –matrix polynomials of the graph Laplacian–, existing CNN works operating on graph signals have replaced classical convolutions with such node-invariant GFs [14]. Nonetheless, how to generalize pooling remains elusive. Attempts using hierarchical multilayer clustering algorithms have been made [15], but clustering is usually a computationally intensive operation [16].

This paper proposes a new architecture for CNNs operating on graph signals upon replacing convolutions with *node-varying* GFs, which are more flexible local graph-signal operators described in [17]. This not only introduces additional degrees of freedom, but also avoids the pooling stage and, as a result, the need to compute a cluster for each of the layers disappears. A second architecture is also proposed, that replaces convolutions with a *hybrid node-varying* GF, a new graph-signal operator introduced in this paper that can be viewed as an intermediate design between node-varying and classical GFs. Our node-varying GF based architectures are able to extract different local features at varying resolutions, do not increase the dimension of the output of each layer, and can be implemented using only local exchanges.

Paper outline: Sec. 2 reviews traditional CNNs and GSP and introduces the definition of node-varying and node-invariant GFs. Sec. 3 presents the new local graph CNN architectures using node-varying GFs. Sec. 4 runs tests on a synthetic source localization problem and on the `20NEWS` dataset.

2. PRELIMINARIES: CNN AND GSP

Let $\mathbf{x} \in \mathcal{X}$ be the input data or signal, defined on a field $\mathcal{X}$, and let $\mathbf{y} \in \mathcal{Y}$ be the output data, defined on a field $\mathcal{Y}$. Let $f : \mathcal{X} \to \mathcal{Y}$ be a function such that $\mathbf{y} = f(\mathbf{x})$. Generically, the objective of CNNs is to design a function $\hat{f} : \mathcal{X} \to \mathcal{Y}$ such that a problem-dependent loss function $\mathcal{L}(\mathbf{y}, \hat{f}(\mathbf{x}))$ is minimized. Standard choices for such a loss are the cross-entropy (for classification) or the mean square error (for regression). The function $\hat{f}$ is built from a concatenation of L layers $\hat{f} = f_L \circ \cdots \circ f_2 \circ f_1$ where each layer is a function $f_\ell : \mathcal{X}_{\ell-1} \to \mathcal{X}_\ell$, $\ell = 1, \ldots, L$ with $\mathcal{X}_0 = \mathcal{X}$ and $\mathcal{X}_L = \mathcal{Y}$. Each one of these layers is computed from three basic operations $\mathbf{x}_\ell = f_\ell(\mathbf{x}_{\ell-1}) = \mathcal{P}_\ell\{\rho_\ell(\mathcal{A}_\ell(\mathbf{x}_{\ell-1}))\}$, where $\mathcal{A}_\ell : \mathcal{X}_{\ell-1} \to \mathcal{X}'_\ell$ is a linear function, $\rho_\ell : \mathcal{X}'_\ell \to \mathcal{X}'_\ell$ is a nonlinear function, and $\mathcal{P}_\ell : \mathcal{X}'_\ell \to \mathcal{X}_\ell$ is the pooling operator, and where $\mathbf{x}_0 = \mathbf{x}$ and $\mathbf{x}_L = \hat{\mathbf{y}} = \hat{f}(\mathbf{x})$ is the estimated output after L layers. It is noted that this

Supported by USA NSF CCF 1717120 and ARO W911NF1710438, and Spanish MINECO TEC2013-41604-R and TEC2016-75361-R.

architecture is computationally straightforward since it is comprised of simple operations, and it is also amenable to be efficiently trained by means of a back-propagation algorithm [18].

In a CNN, the first operation of each layer is a *convolution* with a filter $\mathcal{A}_\ell(\mathbf{x}_{\ell-1}) = \mathbf{a}_\ell * \mathbf{x}_{\ell-1}$. Filter $\mathbf{a}_\ell$ has small support so that it acts as a computationally efficient local feature extractor by relating only a few nearby values of the signal. In order to extract several different features within the same region, a collection of F_ℓ filters $\{\mathbf{a}_{\ell,k}\}_{k=1}^{F_\ell}$ is used, resulting in a F_ℓ-times increase in the dimension of the output. To illustrate this with an example, consider that $\mathbf{x} = \mathbf{x}_0$ is an image of size 16×16, $\mathcal{X} = \mathcal{X}_0 = \mathbb{R}^{16\times 16}$ and that, in the first layer, $\{\mathbf{a}_{1,1}, \ldots, \mathbf{a}_{1,4}\}$ is a collection of $F_1 = 4$ filters of support 2×2 pixels. Then, $\mathcal{X}_1' = \mathbb{R}^{16\times 16\times 4}$ and $\mathcal{A}_1 : \mathbb{R}^{16\times 16} \to \mathbb{R}^{16\times 16\times 4}$ with $\mathcal{A}_1(\mathbf{x}) = \{\mathbf{a}_{1,1} * \mathbf{x}, \ldots, \mathbf{a}_{1,4} * \mathbf{x}\}$.

The second operation is to apply a (pointwise) nonlinear function $\rho_\ell(\cdot)$ to the output of the linear step to yield $\rho_\ell(\mathcal{A}_\ell(\mathbf{x}_{\ell-1})) \in \mathcal{X}_\ell'$. The objective behind applying these nonlinearities at each layer is to create a structure flexible enough to reproduce general nonlinear behaviors. Typical choices for ρ_ℓ include rectified linear units (ReLUs) $\max\{0, x\}$ and the absolute value $|x|$ [19]. Continuing with the previous example, now that the output of the convolution layer is $\mathcal{A}_1(\mathbf{x}_0) \in \mathbb{R}^{16\times 16\times 4}$, we apply a ReLU so that $\rho_\ell : \mathbb{R}^{16\times 16\times 4} \to \mathbb{R}^{16\times 16\times 4}$ with $[\rho_\ell(\mathcal{A}_1\mathbf{x}_0)]_{i,j,k} = \max(0, [\mathcal{A}_1(\mathbf{x}_0)]_{i,j,k})$ for $i, j = 1, \ldots, 16$ and $k = 1, \ldots, 4$.

The third operation is pooling, whose objective is twofold; i) given that each convolution operation increases the number of features, pooling keeps the output dimension under control; and ii) since it is desirable to analyze the data at different resolution levels, pooling reduces the distance between datapoints that were originally far away (with the reduction being more significant as more layers are added). It is noted that a better way to aggregate data in non-bandlimited signals is to do max-pooling or average-pooling instead of traditional downsampling [19]. Returning to the ongoing example, assume that we consider max-pooling of size 2. Then, $\mathcal{P}_1 : \mathbb{R}^{16\times 16\times 4} \to \mathbb{R}^{8\times 8\times 4}$ so that $\mathcal{X}_1' = \mathbb{R}^{16\times 16\times 4}$ and $\mathcal{X}_1 = \mathbb{R}^{8\times 8\times 4}$ and where each element of $\mathbf{x}_1$ is obtained from computing the maximum value of $\rho_1(\mathcal{A}_1(\mathbf{x}_0))$ within pixel masks of size 2×2.

As already explained, those three operations are subsequently repeated by concatenating layers. The idea is to change the representation of the data by progressively trading samples for features [20]. The target representation should be more useful for the specific task at hand as measured by the loss function $\mathcal{L}$. The last step is typically a *readout* layer implementing a (linear) map from $\mathcal{X}_{L-1}$ to $\mathcal{Y}$.

Remark: Albeit fairly typical, modifications to the described CNN architecture have been developed. These range from using outputs of different layers as input to the next layer [21], to assuming that the useful output is collected at every layer instead of the last one [2], to adding fully-connected layers after reaching the all-feature vector [22]. Also, avoiding the pooling stage has been discussed [23].

2.1. Graph signals and filters

In this paper, we consider each datapoint in the dataset to be modeled as a graph signal. To be specific, let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ be a graph with a node set $\mathcal{V}$ with cardinality N, a set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, and a weight function $\mathcal{W} : \mathcal{E} \to \mathbb{R}$. A graph signal is then a mapping $\mathbf{x} : \mathcal{V} \to \mathbb{R}$ that assigns a real number to each node and can be conveniently represented as a vector $\mathbf{x} \in \mathbb{R}^N$, with element $[\mathbf{x}]_k$ being the signal value at node k. Modeling a dataset as a graph signal allows for arbitrary pairwise relationships between the elements of the datapoint (i.e. between the elements of the vector). This relationship is brought to the fore by means of a graph shift operator (GSO) $\mathbf{S} \in \mathbb{R}^{N\times N}$ which is the matrix that relates the signal with the underlying graph support. More specifically, $\mathbf{S}$ is such that $[\mathbf{S}]_{ij} \neq 0$ only if $(i, j) \in \mathcal{E}$ or if $i = j$. This means that $\mathbf{Sx}$ is a local computation that can be carried out by operating only on the neighborhood. Examples of GSOs are the adjacency matrix, the graph Laplacian and their normalized counterparts [8, 9].

The GSO is the key to define the graph Fourier transform (GFT) and the different types of GFs. Assuming first that $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^H$ is a normal matrix diagonalized by a unitary matrix $\mathbf{V}$, the GFT of a signal $\mathbf{x}$ is defined as $\tilde{\mathbf{x}} = \mathbf{V}^H\mathbf{x}$. Moreover, node-invariant and node-varying GFs are defined, respectively, as [17]

$$\mathbf{H}_{\rm ni} := \textstyle\sum_{t=0}^{T-1} h_t\mathbf{S}^t, \qquad \mathbf{H}_{\rm nv} := \textstyle\sum_{t=0}^{T-1} \text{diag}(\mathbf{h}_t)\mathbf{S}^t, \quad (1)$$

where T is the order of the filter, and $\{h_t\}_{t=0}^{T-1}$ and $\{\mathbf{h}_t\}_{t=0}^{T-1}$ are the filter coefficients. Furthermore, if $\mathbf{h}_t \in \mathbb{R}^N$ is set such that $[\mathbf{h}_t]_k = h_t$ for all k and t, then filter $\mathbf{H}_{\rm nv}$ reduces to $\mathbf{H}_{\rm ni}$.

Two interesting properties of the GFs in (1) are: i) they are linear operators that account for the structure of the graph via $\mathbf{S}$, and ii) since $\mathbf{S}$ is a local (one-hop) operator and the output of either $\mathbf{H}_{\rm nv}$ or $\mathbf{H}_{\rm ni}$ can be viewed as a linear combination of successive applications of $\mathbf{S}$ to the input, it follows that $\mathbf{H}_{\rm nv}$ or $\mathbf{H}_{\rm ni}$ are local operators as well. The main difference is that while $\mathbf{S}$ takes into account information within the one-hop neighborhood of the nodes, the operators in (1) consider information that is within their $T-1$ neighborhood [17].

2.2. CNNs using node-invariant GFs

Recent efforts have been made towards extending CNNs to operate on graph signals in the hope of carrying over their excellent performance to a broader class of problems (see [13] for a general survey). The existing works typically set the GSO as the graph Laplacian matrix and, more importantly, replace the classical convolutions with node-invariant GFs [cf. $\mathbf{H}_{\rm ni}$ in (1)]. The main reason for this is that node-invariant GFs allow for the generalization of the convolution theorem to graph signals in the sense that filtering in the (node) domain implies multiplication in the frequency domain given by the GFT. To see why this is the case, consider the graph signal $\mathbf{y} = \mathbf{H}_{\rm ni}\mathbf{x}$, recall the eigendecomposition of the GSO $\mathbf{S}$, and note that since $\mathbf{H}_{\rm ni}$ is a matrix polynomial on $\mathbf{S}$, its eigenvectors are also $\mathbf{V}$. With these considerations, after applying the GFT to the input-output equation $\mathbf{y} = \mathbf{H}_{\rm ni}\mathbf{x}$ we have that $\tilde{\mathbf{y}} = \text{diag}(\tilde{\mathbf{h}})\tilde{\mathbf{x}}$ with $\text{diag}(\tilde{\mathbf{h}}) := \sum_{t=0}^{T-1} h_t\mathbf{\Lambda}^t$ being the filter's frequency response.

Building on this interpretation, [14] designed the filter coefficients to be used at each layer in the spectral domain. To avoid the (expensive) computation of eigendecompositions, a Chebyshev approximation which operates in the node domain using a low-order node-invariant GF was adopted in [15]. While convolutions have been replaced with node-invariant GFs and point-wise nonlinearities with node-wise nonlinearities applied locally at each node of the graph, there is no consensus on how pooling must be implemented. The suggestion in [14] was to use multiscale hierarchical algorithms to create a collection of related graphs with less and less nodes. In that context, [15] adopted the Graclus algorithm [24] and suggested an innovative pooling system by means of a binary tree partition. It is noted that clustering is in itself an ill-posed problem and that there exist several criteria for determining *good* clusters [25, 26]. Moreover, it is usually a computationally intensive operation [16, 27, 28].

3. CNN ARCHITECTURE USING NODE-VARYING GF

Starting from the CNN architecture described in Sec. 2, we propose a new architecture for CNNs that at each layer ℓ: 1) replaces convolutions with node-varying GFs [cf. $\mathbf{H}_{\rm nv}$ in (1)]; 2) applies a local

node-wise nonlinearity; and 3) does not apply a pooling stage, thus avoiding the computation of clusters for each of the layers.

To motivate the proposed design, recall that the idea in the convolution stage is to get several features per region and, for that, F_ℓ filters are employed. This naturally increases the dimension of the signal by a factor of F_ℓ and pooling becomes necessary to prevent a geometric growth of the size of the data. That is, there is a trade-off between the availability of multi-resolution features extracted from the data and the size of the information passed onto the next layer. Our proposed architecture tries to extract local features at different locations of the graph without increasing dimensionality. Being more specific, by adopting the node-varying GF in (1), each node gains the ability to weight their local neighborhood differently, and because nodes within a neighborhood weight differently their respective neighborhoods, each of them acts as a different feature within the region. Since the output of a node-varying GF is another graph signal, then the dimensionality of the data at each layer is not increased while local features are captured respectively by each node. The data analysis at different resolutions comes naturally with the adoption of this kind of filters and is adjusted by the length of the filters on each layer. Concretely, by applying a filter of length T_1 each node gathers information of up to the $T_1 - 1$ neighborhood; then, in the following layer, when another filter of length T_2 is applied, then nodes actually disseminate information up to the $T_2 - 1$ neighborhood from the previous layer, so that the total information processed goes up to the $T_1 + T_2 - 2$ neighborhood. Therefore, as the local graph CNN goes deeper, it gathers more global information.

3.1. CNN via hybrid node-varying GFs

A key aspect of any CNN architecture is the number of parameters that need to be optimized in the training phase [21]. Based on this criterion, it is observed that adopting a node-varying GF results in a number of parameters proportional to the number of nodes, the length of the filter at each layer and the number of layers $\sum_{l=1}^{L} NT_l$. This might be an undesirable characteristic of the architecture, especially for high-dimensional datasets. In order to overcome this, we propose an alternative design where the convolution is replaced with a *hybrid node-varying GF*.

To define this new type of GF, start by considering a *tall* binary matrix $\mathbf{C}_\mathcal{B} \in \{0,1\}^{N \times B}$ with exactly one non-zero entry per row. Define now the reduced vector of filter coefficients as $\mathbf{h}_{\mathcal{B},t} \in \mathbb{R}^B$. Then a hybrid node-varying GF is a graph signal operator of the form

$$\mathbf{H}_{\text{hv}} := \textstyle\sum_{t=0}^{T-1} \text{diag}(\mathbf{C}_\mathcal{B} \mathbf{h}_{\mathcal{B},t}) \mathbf{S}^t. \tag{2}$$

Clearly the GF above is linear, accounts for the structure of the graph, and can be implemented locally. The name "hybrid" is due to the fact that i) if $B = N$ and $\mathbf{C}_\mathcal{B} = \mathbf{I}$, then $\mathbf{H}_{\text{hv}}$ is equivalent to $\mathbf{H}_{\text{nv}}$; and ii) if $B = 1$, then $\mathbf{H}_{\text{hv}}$ reduces to $\mathbf{H}_{\text{ni}}$.

While basis expansion models other than $\mathbf{h}_t = \mathbf{C}_\mathcal{B} \mathbf{h}_{\mathcal{B},t}$ could have been used, $\mathbf{C}_\mathcal{B}$ was selected to be binary to facilitate intuition and keep implementation simple. In particular, the columns of $\mathbf{C}_\mathcal{B}$ can be viewed as membership indicators that map nodes into different groups. With this interpretation, $\{[\mathbf{h}_{\mathcal{B},t}]_b\}_{t=0}^{T-1}$ represents the common filter coefficients that each node of the bth group will use. This demonstrates that the selection of the method to group the nodes offers a new degree of freedom for the design of (2) and the corresponding CNN. Different from the multi-resolution clustering algorithms associated with the pooling stage, this algorithm performs a single grouping. In the simulations presented in the next section, the grouping implicit in $\mathbf{C}_\mathcal{B}$ is carried out in two steps. First, we form the set $\mathcal{B} = \{v_1, \ldots, v_B\}$ containing the B nodes with the highest degree (ties are broken uniformly at random) and set $[\mathbf{C}_\mathcal{B}]_{v_b,b} = 1$ for all $b = 1, ..., B$. Second, for all the nodes that do not belong to $\mathcal{B}$ we set the membership matrix as

$$[\mathbf{C}_\mathcal{B}]_{ij} = 1 \text{ if } j \in \text{argmax}_{b: v_b \in \mathcal{B}} \{\mathcal{W}(i, v_b)\}\ ,\ i \notin \mathcal{B}, \tag{3}$$

where $\mathcal{W}(i, v_b)$ is the edge weight. That is, for each of the nodes not in $\mathcal{B}$ we copy the filter coefficients of the node in $\mathcal{B}$ that exercises the largest influence. As before, ties are broken uniformly at random. CNN schemes with region-dependent filters have been used in the context of images using regular convolutions [29, 30]. The regional features computed at each layer are kept separate and only the last stages (involving fully connected layers) merge them. The use of node-varying graph filters proposed in this paper, not only changes the definition of the convolution, but also merges the regional features at every layer.

Algorithm 1 (Hybrid) Node-varying GF CNN.

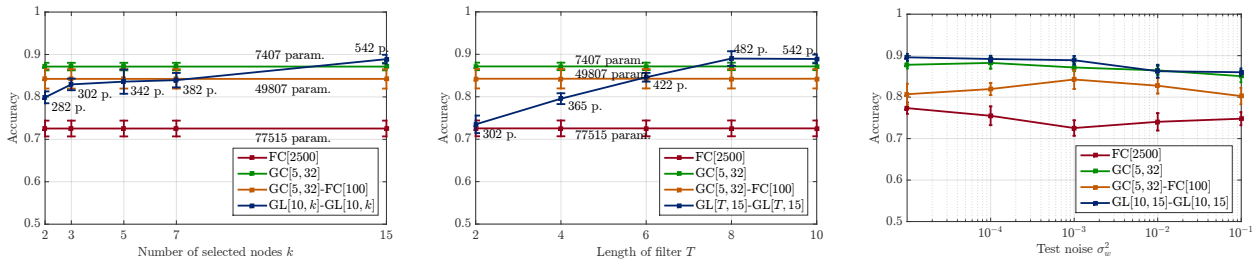
Input: $\{\mathbf{x}\}$: test dataset, $\{(\mathbf{x}', \mathbf{y}')\}$: train dataset
$\mathbf{S}$: GSO, $\{T_1, \ldots, T_{L-1}\}$: degrees of layer
B: number of nodes to select for weights
Output: $\{\hat{\mathbf{y}}\}$: estimates

procedure NVGF_CNN($\{\mathbf{x}\}$,$\{(\mathbf{x}', \mathbf{y}')\}$,$\mathbf{S}$,$\{T_1, \ldots, T_{L-1}\}$,$B$)
  Create set $\mathcal{B}$ by selecting B nodes with highest degree
  Compute $\mathbf{C}_\mathcal{B}$ ▷ See (3)
  Create the $L-1$ layers:
  for $\ell = 1 : L-1$ **do**
    Create B filter taps $\{\mathbf{h}_{\mathcal{B},0}, \ldots, \mathbf{h}_{\mathcal{B},T_\ell - 1}\}$
    Obtain $\mathbf{H}_\ell = \sum_{t=0}^{T_\ell - 1} \text{diag}(\mathbf{C}_\mathcal{B} \mathbf{h}_{\mathcal{B},t}) \mathbf{S}^t$ ▷ See (2)
    Apply non-linearity $\rho_\ell(\mathbf{H}_\ell \cdot)$
  end for
  Create readout layer $\mathcal{A}_L$ ·
  Learn $\{\mathbf{h}_{\mathcal{B},0}, \ldots, \mathbf{h}_{\mathcal{B},T_{\ell-1}}\}_{\ell=1}^{L-1}$ and $\mathcal{A}_L$ from $\{(\mathbf{x}', \mathbf{y}')\}$
  Obtain $\hat{\mathbf{y}} = \hat{\mathbf{f}}(\mathbf{x})$ using trained coefficients
end procedure

CNN architecture: Adopting the hybrid node-varying GF for the first stage of each layer of our CNN implies that the total number of parameters to be learned is $\sum_{l=1}^{L} BT_l$, which is independent of N and guarantees that the proposed architecture scales well for high-dimensional data. Lower values of B will decrease the number of training parameters, while limiting the ability of extracting features of the filter. All in all, the architecture of the proposed CNN is given by Algorithm 1. We observe that, except for the final readout layer, all computations are carried out in a local fashion making the CNN amenable to a distributed implementation. Finally, let us note that while some problems inherently live in a constant-dimension submanifold and make the choice of a constant B possible, some other problems might have a lower dimension that still grows with N but in a sublinear fashion. Therefore, while B might not be independent of N, it could still be chosen as a sublinear function of N [31, 32].

4. NUMERICAL TESTS

In this section, we run tests on the proposed CNN architecture and compare it with the one developed in [15]. In general, we observe that our CNN achieves similar performance but with at least one order of magnitude less of parameters. In the first testcase we consider a synthetic dataset of a source localization problem in which different diffused graph signals are processed to determine the single node that originated them. In the second testcase we use the `20NEWS` dataset and a `word2vec` embedding underlying graph to



(a) (b) (c)

Fig. 1: Accuracy in the source localization problem. Results were averaged across 10 different realizations. For clarity of figures, error bars represent $1/4$ of the estimated variance. The number of parameters of each architecture is also shown. (a) As a function of the number of selected nodes k. Accuracy gets better as more and more nodes are selected to extract features. (b) As a function of the length of the filter T. Accuracy improves when filters are longer. It is observed that after length 8 for which 2 layers are able to obtain all the relevant information in the graph (15 nodes), accuracy does not improve substantially. (c) As a function of the noise in the test set. The proposed architecture is fairly robust to noise since accuracy drops approximately 5% across 5 orders of magnitude.

Architecture	Parameters	Accuracy
FC[2500]	$77,515$	72.6%
GC[5, 32]	$7,407$	87.2%
GC[5, 32]-FC[100]	$49,807$	84.3%
GL[10, 15]-GL[10, 15]	542	88.9%

Table 1: Source localization results for $N = 15$ nodes.

classify articles in one out of 20 different categories [33]. For both problems, denote as GC$[T, k]$ a graph CNN using Chebyshev polynomial approximation of order T with k features; as FC$[k]$ a fully connected layer CNN with k hidden units; and as GL$[T, k]$ the proposed CNN where the degree-based hybrid node-varying GF is of order T with $B = k$ nodes selected. A ReLU nonlinearity is applied at each layer and all architectures include a readout layer. We note that the total parameter count includes this last readout layer as well as bias parameters typically used before applying the ReLU nonlinearity. For the training stage in both problems, an ADAM optimizer with learning rate 0.005 was employed [34], for 20 epochs and batch size of 100.

Testcase 1: Source localization. Consider a connected Erdős-Rényi (ER) graph with N nodes and edge probability $p_{ER} = 0.4$ and let $\mathbf{W}$ denote its adjacency matrix. With $\boldsymbol{\delta}_c$ representing a graph signal taking the value 1 at node c and 0 elsewhere, the signal $\mathbf{x} = \mathbf{W}^t\boldsymbol{\delta}_c$ is a diffused version of the sparse input $\boldsymbol{\delta}_c$ for some unknown $0 \leq t \leq N-1$. The objective is to determine the source c that originated the signal $\mathbf{x}$ irrespective of time t. To that end, we create a set of N_{train} labeled training samples $\{(c', \mathbf{x}')\}$ where $\mathbf{x}' = \mathbf{W}^t\boldsymbol{\delta}_{c'}$ with both c' and t chosen at random. Then we create a test set with N_{test} samples in the same fashion, but we add i.i.d. zero-mean Gaussian noise $\mathbf{w}$ with variance σ_w^2, so that the signals to be classified are $\mathbf{W}^t\boldsymbol{\delta}_c + \mathbf{w}$. The goal is to use the training samples to design a CNN that determines the source (node) c that originated the diffused.

For a graph with $N = 15$ nodes we test four architectures: (a) FC[2500], (b) GC$[5, 32]$, (c) GC$[5, 32]$-FC$[100]$ and (d) GL$[10, 15]$-GL$[10, 15]$. The GSO employed is the adjacency matrix $\mathbf{S} = \mathbf{W}$. A dropout of 0.5 is included in the training phase. The test set is of size $N_{\text{test}} = 200$. Results are listed in Table 1. Note that these results are obtained by averaging 10 different realizations of the problem. We observe that the performance of our CNN is similar to that of GC$[5, 32]$ but with ten times less parameters.

Additionally, we run tests changing the values of several of the parameters of the architecture. In Fig. 1a we observe the accuracy obtained when varying the number of selected nodes. It is noted that selecting less nodes implies that less features are extracted. This impacts negatively the accuracy. Nonetheless, even an accuracy level of 80% is achieved with as few as 282 parameters, which is a better performance than using a fully connected layer with 2500 hidden units which requires 100 times more parameters. The dependence of the accuracy on the length of the filter T can be observed in Fig. 1b. We note a linear increase in accuracy that saturates around $T = 8$. This is the length for which, when using two layers, the information corresponding to the whole graph can be aggregated. Finally, in Fig. 1c we show the performance of all four architectures as a function of the noise on the test set. We observe that both GC$[5, 32]$ and the proposed architecture achieve similar accuracies with a fairly robust performance, since the accuracy dropped only 5% within a 5 order magnitude change in the noise. Values shown are mean accuracies obtained after averaging 10 realizations and the error bars represent $1/4$ of the estimated variance from these realizations.

Architecture	Parameters	Accuracy
GC$[5, 32]$	$1,920,212$	60.75%
GL$[5, 1500]$	$67,521$	60.34%

Table 2: Results for classification on `20NEWS` dataset on a `word2vec` graph embedding of $N = 3,000$ nodes.

Testcase 2: `20NEWS` dataset. Here we consider the classification of articles in the `20NEWS` dataset which consists of $18,846$ texts ($11,314$ of which are used for training and $7,532$ for testing) [33]. The graph signals are constructed as in [15]: each document x is represented using a normalized bag-of-words model and the underlying graph support is constructed using a 16-NN graph on the `word2vec` embedding [35] considering the $3,000$ most common words. The GSO adopted is the normalized Laplacian. No dropout is used in the training phase. The architectures used are GC$[5, 32]$ and GL$[5, 1500]$. Accuracy results are listed in Table 2, demonstrating that both architectures achieve similar accuracies, but with our CNN requiring 100 times less parameters.

5. CONCLUSIONS

A CNN architecture to operate on graph signals was proposed. The convolution stage was replaced by a node-varying GF, and no pooling stage was implemented. Extraction of different features was achieved by the adoption of a node-varying GF and resolution levels were adjusted via the length of the filter. The convolutional layers of the resulting CNN could be implemented locally. To prevent the number of parameters to grow with the size of the data, we proposed a hybrid node-varying GF where nodes were grouped and the same filter coefficients were used within a particular group. Results on the `20NEWS` dataset showed a performance similar to that of existing CNNs implementing node-invariant GFs but with 100 times less parameters to train. A synthetic source localization problem was used to asses numerically the sensitivity of the estimation performance with respect to the number of groups and the degree of the filter.

6. REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 85–117, 2015.

[2] J. Bruna and S. Mallat, "Invariant scattering convolution networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1872–1886, Aug. 2013.

[3] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *2010 IEEE Int. Symp. Circuits and Syst.*, Paris, France, 30 May-2 June 2010, IEEE.

[4] H. Greenspan, B. van Ginneken, and R. M. Summers, "Deep learning in medical imaging: Overview and future promise of an exciting new technique," *IEEE Trans. Med. Imag.*, vol. 35, no. 5, pp. 1153–1159, May 2016.

[5] D. Lazer et al., "Life in the network: The coming age of computational social science," *Science*, vol. 323, no. 5915, pp. 721–723, Feb. 2009.

[6] E. H. Davidson et al., "A genomic regulatory network for development," *Science*, vol. 295, no. 5560, pp. 1669–1678, Feb. 2002.

[7] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs," *IEEE Trans. Signal Process.*, vol. 61, no. 7, pp. 1644–1656, Apr. 2013.

[8] A. Sandyhaila and J. M. F. Moura, "Discrete signal processing on graphs: Frequency analysis," *IEEE Trans. Signal Process.*, vol. 62, no. 12, pp. 3042–3054, June 2014.

[9] D. I Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 83–98, May 2013.

[10] S. Chen, R. Varma, A. Sandryhaila, and J. Kovačević, "Discrete signal processing on graphs: Sampling theory," *IEEE Trans. Signal Process.*, vol. 63, no. 24, pp. 6510–6523, Dec. 2015.

[11] A. G. Marques, S. Segarra, G. Leus, and A. Ribeiro, "Sampling of graph signals with successive local aggregations," *IEEE Trans. Signal Process.*, vol. 64, no. 7, pp. 1832–1843, Apr. 2016.

[12] S. Segarra, A. G. Marques, G. Leus, and A. Ribeiro, "Reconstruction of Graph Signals Through Percolation from Seeding Nodes," *IEEE Trans. Signal Process.*, vol. 64, no. 16, pp. 4363–4378, Aug. 2016.

[13] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric Deep Learning: Going Beyond Euclidean Data," *arXiv:1611.08097v1 [cs.CV]*, 24 Nov. 2016.

[14] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs," *arXiv:1312.6203v3 [cs.LG]*, 21 May 2014.

[15] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," *arXiv:1606.09375v3 [cs.LG]*, 5 Feb. 2017.

[16] G. Carlsson and F. Mémoli, "Characterization, stability and convergence of hierarhical clustering methods," *J. Mach. Learning Res.*, vol. 11, pp. 1425–1470, Apr. 2010.

[17] S. Segarra, A. G. Marques, and A. Ribeiro, "Optimal graph-filter design and applications to distributed linear network operators," *IEEE Trans. Signal Process.*, vol. 65, no. 15, pp. 4117–4131, Aug. 2017.

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[19] T. Wiatowski and H. Bölcskei, "A mathematical theory of deep convolutional neural networks for feature extraction," Website, 23 March 2017.

[20] J.-H. Jacobsen, E. Oyallon, S. Mallat, and A. W. M. Smeulders, "Multiscale hierarchical convolutional networks," *arXiv:1703.04140v1 [cs.LG]*, 12 March 2017.

[21] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," *arXiv:1608.06993v4 [cs.CV]*, 27 Aug. 2017.

[22] C.-C. J. Kuo, "The cnn as a guided multilayer recos transform," *IEEE Signal Process. Mag.*, vol. 34, no. 3, pp. 81–89, May 2017, lecture notes.

[23] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv:1412.6806v3 [cs.LG]*, 13 Apr. 2015.

[24] I. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors: A multilevel approach," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 11, pp. 1944–1957, Nov. 2007.

[25] D. Horta and R. J. G. B. Campello, "Comparing hard and overlapping clusterings," *J. Mach. Learning Res.*, vol. 16, pp. 2949–2997, Dec. 2015.

[26] F. Gama, S. Segarra, and A. Ribeiro, "Hierarchical Overlapping Clustering of Network Data Using Cut Metrics," *IEEE Trans. Signal, Inform. Process. Networks*, vol. PP, no. 99, 24 May 2017.

[27] J. H. Ward Jr., "Hierarchical grouping to optimize an objective function," *J. Amer. Statist. Assoc.*, vol. 58, no. 301, pp. 236–244, March 1963.

[28] D. Defays, "An efficient algorithm for a complete link method," *Comput. J.*, vol. 20, no. 4, pp. 364–366, Apr. 1977.

[29] K. Gregor and Y. LeCun, "Emergence of complex-like cells in a temporal product network with local receptive fields," *arXiv:1006.0448v1 [cs.NE]*, 2 June 2010.

[30] G. B. Huang, H. Lee, and E. Learned-Miller, "Learning hierarchical representations for face verification with convolutional deep belief networks," in *2012 IEEE Conf. Comput. Vision, Pattern Recognition*, Providence, RI, 16-21 June 2012, IEEE Comput. Soc.

[31] M. B. Wakin, D. L. Donoho, H. Choi, and R. G. Baraniuk, "The multiscale structure of non-differentiable image manifolds," in *SPIE Optics + Photonics*, San Diego, CA, July 2005, SPIE.

[32] N. Verma, S. Kpotufe, and S. Dasgupta, "Which spatial partition trees are adaptive to intrinsic dimension?," *arXiv:1205.2609v1 [stat.ML]*, 9 May 2012.

[33] T. Joachims, "Analysis of the rocchio algorithm with tfidf for text categorization," Computer Science Technical Report CMU-CS-96-118, Carnegie Mellon University, 1996.

[34] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *arxiv:1412.06980v9 [cs.LG]*, 30 Jan. 2017.

[35] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv:1301.3781v3*, 7 Sep. 2013.