Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

Tom Welch
tom.welch-honnart@etu.u-paris.fr

Philippe Gratias-Quiquandon
pgratias-22@telecom-paris.fr

Louis-Ferdinand Marignier
louis-ferdinand.marignier@telecom-sudparis.eu

1 Introduction

1.1 CNN

The effectiveness of convolutional neural networks in their field of application is well established. But when considering these fields: image processing, computer vision, natural language processing... we realize that they all share one fundamental characteristics, they can be represented in the form of structured data. In fact, the use of convolutional filters is made possible by this type of structure. These very filters enable networks to capture the local characteristics of data. Yet, despite this success with grid-structured data, many real-world problems involve complex relational structures that are better represented with graphs. Graph data encompasses a wide range of domains including social networks, biological systems, and transportation infrastructures. Traditional CNNs are not inherently designed to handle the irregularities and dynamic nature of graph data. To bridge this gap, one would naturally seek designs resembling CNNs on graphs, but the lack of structure stands as a hurdle for them to come to fruition. In this report, we present a method developed by Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst trying to address this problem.[3].

1.2 The idea behind the article

The above-mentioned issue can be addressed through graph neural networks (GNNs), which have emerged as promising extensions of CNNs, facilitating the processing and analysis of graph-structured information. The authors propose a new approach to the generalization of graph neural networks based on spectral graph theory. Therefore, it overcomes the difficulties inherent to defining convolution and pooling operations on irregular domains.

Their approach introduces several significant contributions. By grounding CNNs in spectral graph theory, the authors redefine convolution operations on graphs using the graph Laplacian, which enables the analysis of graph signals within the frequency domain. They design spectral filters that are strictly localized within a radius of K hops from a central vertex, thereby improving both the interpretability and efficiency of feature extraction. Additionally, the methodology achieves linear evaluation complexity relative to the filter's support size and the number of edges, making it highly scalable for large and sparse graphs. This efficiency is attained by circumventing the computationally demanding eigenvalue decomposition (EVD) typically necessary in spectral methods. Moreover, the authors introduce an innovative pooling strategy that restructures graph vertices into a binary tree format, analogous to pooling in one-dimensional signals, which facilitates hierarchical feature aggregation.

We shall review the construction details of all the steps mentioned above, show our implementation and experiments, and finally try to highlight some of the shortcomings this method suffers from.

2 The method

2.1 Spectral Graph Theory

From now on, we will consider data represented as a connected graph. That is, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$, $\mathcal{V}$ is a set whose elements are called vertices, and $\mathcal{E}$ is a set of unordered pairs $\{v_1, v_2\}$ of vertices, whose elements are called edges. If we have $|\mathcal{V}| = n$, then W is the $n \times n$

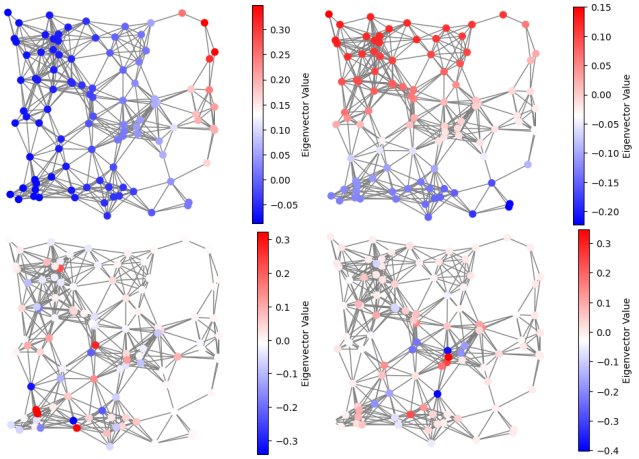

Figure 1: Different eigenfunctions of a random graph. Indices of eigenvalues : top-left 1, top-right 2, bottom-left 75, bottom-right 90.

(symmetric) weighted adjacency matrix ; $W_{ij} > 0$ if there is an edge between v_i and v_j. And if we define the diagonal matrix D by $D_{ii} = \sum_j W_{ij}$, we then can define the central tool of the theory, the graph Laplacian:

$$L = D - W = I_n - D^{-1/2} W D^{-1/2}$$

As a side note, $D^{-1/2}$ is defined as long as there is no vertex that is not linked to another vertex, even though the matrix L is defined no matter what using the first formula. The Laplacian matrix L is symmetric and positive semidefinite, ensuring a complete set of orthonormal eigenvectors $\{u_l\}_{l=0}^{n-1}$ known as graph Fourier modes, and corresponding non-negative eigenvalues $\{\lambda_l\}_{l=0}^{n-1}$ representing graph frequencies. As we can see in Figure 1, higher eigenvalues correspond to higher frequencies.

The Graph Fourier Transform (GFT) extends the traditional Fourier Transform to graph-structured data by projecting a signal $x \in \mathbb{R}^n$ (which can also be seen as a function $x : \mathcal{V} \to \mathbb{R}$) onto the basis formed by the eigenvectors of the Laplacian. Specifically, the GFT of x is defined as $\hat{x} = U^T x$, where $U = [u_0, \ldots, u_{n-1}] \in \mathbb{R}^{n \times n}$ is the matrix of eigenvectors, and its inverse transform reconstructs the signal as $x = U\hat{x}$. This transformation enables the application of fundamental signal processing operations, such as filtering, within the spectral domain of the graph. U gives a basis in which the Laplacian is diagonalized: $L = U\Lambda U^T$, with $(\Lambda)_{ii} = \lambda_i$.

Conventional convolution relies on the ability to translate a filter across the input data, a property inherent to regular grids. However, in graph domains, the absence of a meaningful translation operator complicates the direct application of convolutions. To overcome this issue, the convolution operator on a graph is defined in the Fourier domain. Specifically, the convolution of a signal x with a filter y on a graph is expressed as: $x *_{\mathcal{G}} y = U\left(\left(U^T x\right) \odot \left(U^T y\right)\right)$, where $\odot$ is the Hadamard product [3]. This operation relies on manipulating frequency components rather than spatially translating filters across nodes. It inherently accounts for the graph's structure through the eigenvectors and eigenvalues of the Laplacian. For a general filter g_θ depending on a learnable parameter θ we have:

$$y = g_\theta(L)x = g_\theta(U\Lambda U^T)x = U g_\theta(\Lambda)U^T x \qquad (1)$$

When considering this formula, one can see that we first transform x into $\hat{x}$, then apply the filter, and finally go back to the spatial domain.
This filter must satisfy certain properties to mimic a normal CNN. The first being its need to be localized, i.e., not take into account all vertices of the graph. To ensure this, the authors proposed the following form:

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

If one wanted to center this filter on the vertex v_i, one would simply consider: $g_\theta(L)\delta_i$. Hence, the value of this filter at the vertex v_j is $(g_\theta(L)\delta_i)_j = (g_\theta(L))_{ij} = \sum_{k=0}^{K-1} \theta_k (L^k)_{ij}$.
To see that this expression corresponds to our requirement, one would need a result from spectral graph theory which states if two vertices v_i, v_j are strictly further then K hops apart, then $(L^k)_{ij} = 0$ [5]. To get an intuition as to why this holds, we need some insights into L^k. If we develop $(D - W)^k$, we'll get terms with highest power being W^k and D^k, those two matrices precisely captures interactions up to k-hops away, while the mixed terms $D^{k-m}W^m$ integrate both the degree scaling and the connectivity information up to m-hops. This precisely means that the filter g_θ is $nK-$localized. To even further convince us about this, we can apply L^k to a delta function at a specific node and see the propagation, results are show in Figure 2 (we use a regular graph for clearer display).

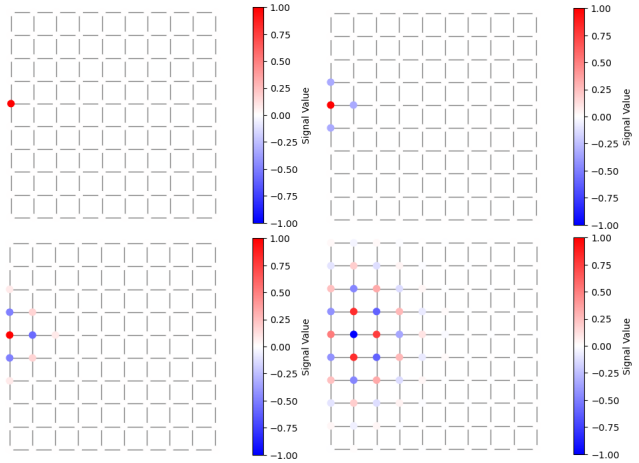

Figure 2: Propagation of a Dirac throughout L^k : top-left L^0, top-right L^1, bottom-left L^2, bottom-right L^5.

2.2 Computational complexity

Computing the filtered signal with Equation (1) naively leads to a complexity of $O(n^2)$ for matrix-vector multiplications, assuming that we have already computed the eigendecomposition of L. However, obtaining U and Λ requires $O(n^3)$ operations. Therefore, the overall complexity becomes $O(n^3)$, which is computationally infeasible for large graphs. To avoid this problem, one idea is to circumvent the computation of U and Λ. Therefore, we compute directly $y = g_\theta(L)x$ with $g_\theta(L) = \sum_{k=0}^{K-1} \theta_k L^k$. This method results in a complexity of $O(K|\mathcal{E}|)$ due to the sparsity of L. This idea is equivalent to filtering in the spectral domain, precisely due to the equality (1).

Furthermore, the authors decided to use Chebyshev polynomials to ensure filter stability. Indeed, L^k may not be well-conditioned while $T_k(\tilde{L})$, the k-th polynomial, has a magnitude in $[-1; 1]$ where $\tilde{L}$ is defined below.

The filter was thus defined as follows:

$$g_\theta(L) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})$$

where $\tilde{L}$ is the scaled Laplacian $\tilde{L} = \frac{2L}{\lambda_{\max}} - I_n$ (to once again make sure $T_k(\tilde{L})$ has a magnitude in $[-1; 1]$). A filtered signal is now defined by:

$$y = g_\theta(L)x = \sum_{k=0}^{K-1} \theta_k \bar{x}_k = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x$$

where $\bar{x}_k$ is computed using the recurrence formula. $\bar{x}_0 = x$, $\bar{x}_1 = \tilde{L}x$, and $\bar{x}_k = 2\tilde{L}\bar{x}_{k-1} - \bar{x}_{k-2}$. The computation of λ_{max} is not expensive thanks to the implicitly restarted Lanczos method [2].

One natural question that arises is: is this equivalent to the idea of computing the different powers of L? The answer is yes. To see why, let us remark that the last filter can be written : $[\bar{x}_0, \ldots, \bar{x}_{K-1}]\,\theta$, which is linear. Then let's recall that the set of Chebyshev polynomials $\{T_k(x)\}$ and the set of monomials $\{X^k\}$ both form bases for the space of polynomials up to degree $K-1$. Then, the two filters are the same but expressed in a different basis; they are indeed equivalent.

2.3 Gradient and backpropagation

In the context of Graph Convolutional Networks (GCNs), efficient computation of gradients is crucial for effective training, especially when dealing with large-scale and complex graph-structured data. The authors present a formulation for computing the output feature maps and the associated gradients necessary for backpropagation within spectral GCNs. The core operation in a GCN layer involves generating output feature maps from input feature maps through spectral filtering. Specifically, the $j-$th output feature map $y_{s,j}$, for a sample s is defined as:

$$y_{s,j} = \sum_{i=0}^{F_{\text{in}}} g_{\theta_{i,j}}(L)x_{s,i}$$

If E represents the loss function we chose, we then need the two following derivatives:

$$\begin{cases} \dfrac{\partial E}{\partial \theta_{i,j}} = \displaystyle\sum_{s=1}^{S} \frac{\partial E}{\partial y_{s,j}} \frac{\partial y_{s,j}}{\partial \theta_{i,j}} = \sum_{s=1}^{S} [\bar{x}_0, \ldots, \bar{x}_{K-1}]^T \frac{\partial E}{\partial y_{s,j}} \\ \dfrac{\partial E}{\partial x_{s,i}} = \displaystyle\sum_{j=1}^{F_{\text{out}}} \frac{\partial E}{\partial y_{s,j}} \frac{\partial y_{s,j}}{\partial x_{s,i}} = \sum_{j=1}^{F_{\text{out}}} g_{\theta_{i,j}}(L) \frac{\partial E}{\partial y_{s,j}} \end{cases}$$

where S in the number of samples. Once again the sparsity of L tremendously alleviates the computation complexity of these operations. Sadly, there remains a dense matrix-vector multiplication, that is $\frac{\partial E}{\partial \theta_{i,j}}$. In the forward pass, we had the previous complexity, $O(K|\mathcal{E}|)$. For each output feature map $y_{s,j}$, the summation involves

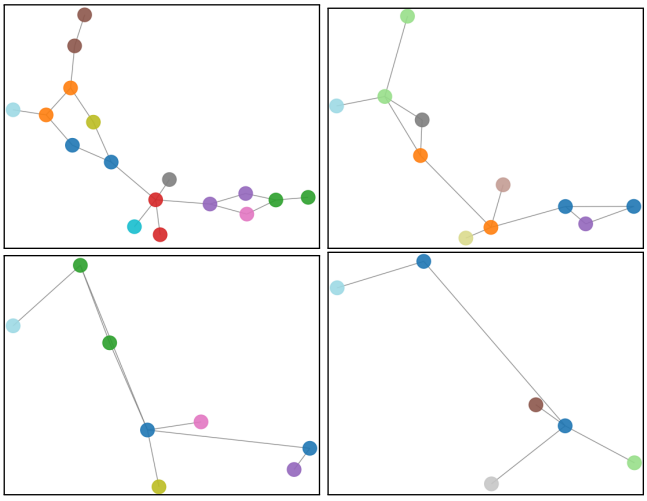

Figure 3: Progressive coarsening of a color, same color nodes are grouped during the next iteration. The number of nodes is halved each time starting from 17.

F_{in} spectral filtering operations, which raises the complexity to $O\left(K|\mathcal{E}|F_{\text{in}}\right)$. For each sample s, computing all F_{out} output feature maps across all samples S in the mini-batch : $O\left(K|\mathcal{E}|F_{\text{in}}F_{\text{out}}S\right)$. This term is dominant compared to the complexity of the backward pass.

2.4 Graph coarsening and pooling

Pooling operations are integrated to CNNs as they enable the extraction of hierarchical features by progressively reducing the spatial dimensions of the input data. Extending this concept to GCNs introduces unique challenges due to the irregular nature of graph data.
The authors address these challenges by employing a multi-level clustering approach, specifically utilizing the Graclus multilevel clustering algorithm, to perform pooling operations that preserve local geometric structures. Graclus employs a greedy matching strategy to create successive coarser versions of the graph, effectively approximately halving its size at each coarsening level. This approach facilitates multi-scale clustering, allowing the GCN to operate at different resolutions and capture hierarchical features. An example of graph coarsening using this method is shown in Figure 3.

A problem then arises, that of the storage of these coarsened graphs. A table containing all matched vertices would not be satisfactory in terms of memory usage, speed and parallelization. To get around this problem, the authors noted that this set graphs can be represented as a binary tree. The idea is to start from the last graph and assign a random number to each and every vertex. They then all have two parents in the previous graph; we add a fake node if the vertex only has one parent. The value carefully set for those nodes does not impact the filtering process and typically consists of 0 for classical ReLU, max-pooling combination. An important point is that nodes with adjacent numbers are aggregated together. Allowing to perform max-pooling as if the signal was 1 dimensional.

Let's go through a detailed example to get an intuition of the method, with only one iteration. We start with the graph $\mathcal{G}_0$ on Figure 4. Graclus then gives the graph $\mathcal{G}_1$.

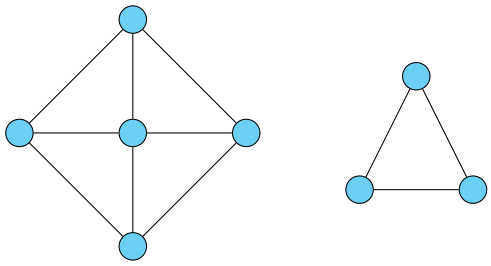

Figure 4: $\mathcal{G}_0$ on the left, and $\mathcal{G}_1$ on the right.

Then we number arbitrary our last graph, and build the binary tree according to what Graclus returned, see *Figure* 5. But we strive for a binary tree, that's where

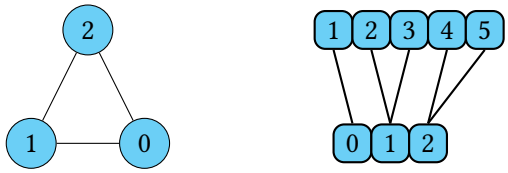

Figure 5: Graph numbering and partial tree

fake nodes come into play (only the 0-th node is fake here) to get the final binary tree, as shown on figure Figure 6.

The pooling is then simply given by:
$z = [\max(x_0, x_1), \max(x_2, x_3) \max(x_4, x_5)]$, and knowing the fact that 0 is a fake node, it becomes:
$z = [x_1, \max(x_2, x_3) \max(x_4, x_5)]$.

3 Experiments with Cora Dataset

In this section, we tried applying the previously enounced concepts to the Cora Dataset. Since, the authors solely

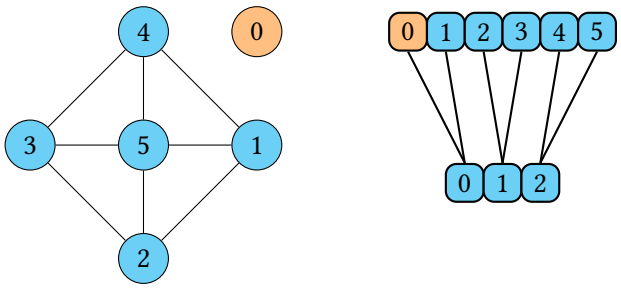

Figure 6: Complete binary tree with fake node

limited their scope to the MNIST and 20NEWS datasets, we took the initiative of testing on another dataset to make sure our results would coincide with those announced. We attempted to evaluate whether this method gave satisfactory results and pace, or at least comparable to those obtained in 2016.

Cora is a dataset containing 2,708 nodes, each node represents an article and an edge between two articles means a citation. In particular, each node has a 0/1-valued feature word vector which represents the presence or not of a word inside the article given a dictionary. The problem we solve here is that of classification: there are 7 classes of article corresponding to their theme. Given a node, its feature vector, and its edges, what is his class?

Following the article, we understand that we have to invocate three main components: the convolution using Chebyshev polynomials, the coarsening using the Graclus algorithm, and finally the max-pooling following Graclus. We complete the neural network with a fully connected layer incorporating an output of size 7 to the classes.

We tried multiple architectures, the final one is represented Figure 7. Here is an explanation of what is happening inside this architecture and why made this choice.

3.1 Description of the implementation

We start from our graph with feature vectors and we apply the Chebyshev convolution on it. As a reminder, we use Chebyshev polynomials to apply a filter on a graph and therefore "diffuse" values to its neighbors. In particular, the hyperparameter K we defined in 2.2 can be interpreted as the number of hops; the "field of view" of a filtered node.

After the convolution, the feature vector will be diffused to each node, allowing the network to take into account neighbor's feature vectors, which is the principle of convolution.

After that, we apply a ReLU and Graclus to find clusters and perform the graph coarsening. We then apply the max-pooling method to reduce the size of the clusters, and finally find the class corresponding to each clusters. Therefore, we also technically needed to register in the code which node correspond to which cluster and realize a mapping at the end to get the correct output.

At first, we tried a more complex architecture (2 Chebyshev convolutions followed by max-poolings), which was clearly overfitting on the training dataset. We consequently decided to lighten the model by simply using one Chebyshev convolution and one max-pooling, similarly to Figure 7. We implemented this structure using the torch geometric library [4] in Python.

3.2 Results and comparisons

We can find on [1] that at the time of the article, in 2016, the state-of-the-art performance was up to 75.7%, which gave us hope for our model's performance to belong to the same order of magnitude.

We trained our model with a learning rate of 0.01, a weight decay of 5.10^{-4}, and 100 epochs. The results are shown in Figure 8. The final accuracy for the test dataset is 79.2% which is coherent with our expectations. Concerning the computing process, it was quite fast as we trained it on one of our CPUs and took 1.5 minute. We also see observe a fast convergence as the plateau reached after only 20 epochs.

Though, one problem is that the training accuracy immediately goes to 1, but this may simply be related to the dataset. Indeed, the Cora dataset is a very common and constitute a simple example. We should try on larger training datasets to rule out the risk of overfitting due to lack of data. However, these results are still satisfactory and comparable to state-of-the-art performance. They also show the parsimony of the model.

4 Shortcomings

Despite the useful tools it introduces and the theoretical insights it provides, the present work nevertheless retains caveats we shall not oversee. Indeed, central aspects of its approach, regarding the very feasibility

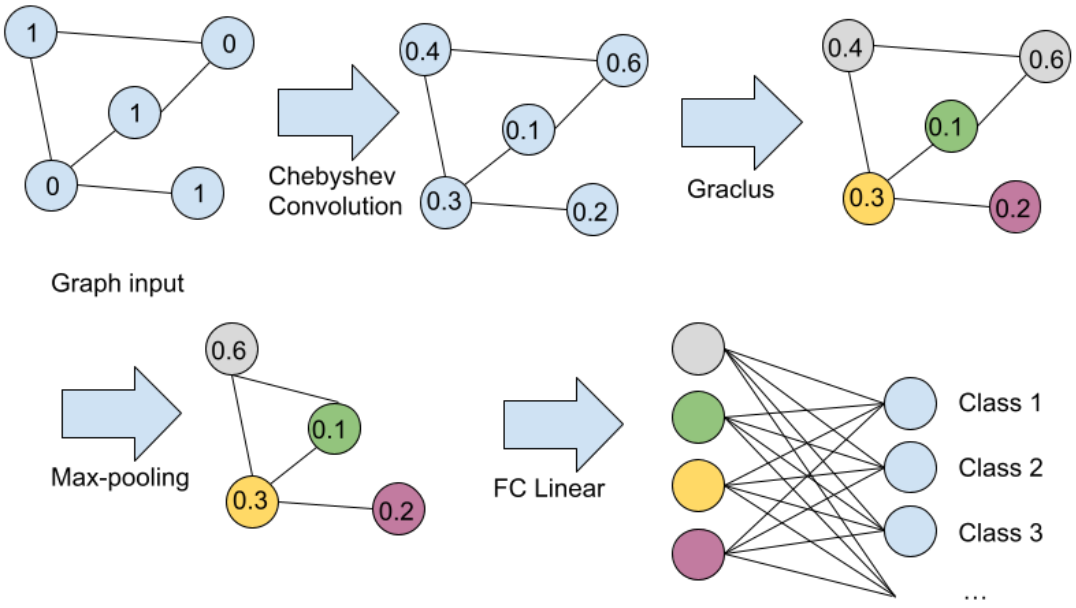

Figure 7: Illustration of the architecture used. The values here are given as an example for a better understanding.

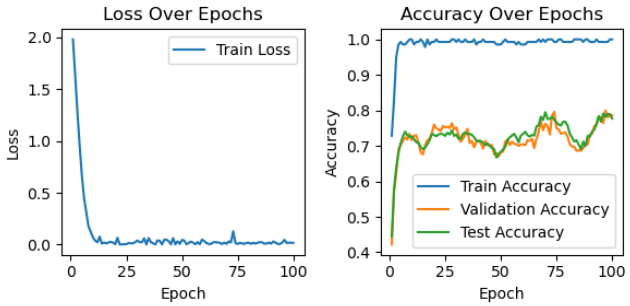

Figure 8: Loss and accuracies during the training

or the optimality of its methods, might rely on non-generalizable, disputable or even sometimes brittle hypothesis. We shall thereafter review those limitations.

4.1 Generalizability

First and foremost, several hurdles stand in the way of an implementation on data from complex real-world scenarios like dynamic social networks or large-scale biological data. To begin with, the sparsity hypothesis of L, upon which the decrease in complexity relies, that is $|\mathcal{E}| \ll n^2$ might not apply to all graphs. For instance, a complete graph constitutes an obvious loophole. The article supposes that most real-world application would come with a sparse matrix L. However, the computational efficiency it permits when compared to previous approaches might not apply to dense matrices and may result in high computational costs. Along with this constraint, Indeed, it should also be noted that regardless of the sparsity of the Lagrangian, this method demands significant GPU memory resources due to recursive computations and large matrix operations, posing challenges for large-scale datasets. Some tools could help alleviate these costs. Those include algebraic multigrid techniques on graphs [6], Kron reduction [7], and Lanczos algorithm [8]. Yet, despite their worthiness, they were left for future works.

Secondly, another limitation lies in the assumption of a static graph structure. The model does not provide mechanisms to learn or adapt the graph during training, which could hinder its performance in dynamic environments such as many of those commonly studied by the means of graphs, social networks or neuroscientific data for instance. Furthermore, for such ubiquitous data often displaying directed or disjoint graphs, the isotropy of spectral filters introduces limitations in tasks requiring directional sensitivity, as the filters lack the ability to encode orientation information.

One might also notice the reliance on localized filters

within a fixed radius (K hops) may limit the model's ability to capture long-range dependencies in certain graph configurations.

4.2 Evaluation

Final and crucial phase of a work, the performance evaluation of a new method heavily determines its worth in the eyes of the scientific community. In this case, the evaluation was carried out thanks to two wildly utilized datasets, MNIST and 20NEWS. In these two cases, the authors report notable performances, thorough put into perspective with state-of-the-art algorithms.
Nevertheless, the scope of these two datasets may very well be excessively narrow to draw meaningful conclusions regarding as they remain relatively simple and may not prove its generalizability to more complex real-world scenarios like dynamic social networks or large-scale biological data, since those may suffer varying degrees of sparsity, scale, and domain-specific characteristics.
One might nowadays take interest in comparisons with modern graph-based learning techniques or spatial-based GNNs, which define convolutions directly on the graph's spatial structure.

5 Conclusion

The three authors of the present article offer a significant advancement in the generalization of CNNs to graph-structured data through a spectral graph theoretical approach. By addressing the challenges of convolution and pooling operations on irregular domains, the proposed method offers a scalable and efficient solution for graph-based learning tasks.
We showed that this model is also appropriate for the Cora dataset. Despite its rather simplistic nature, the architecture we utilized proved light and powerful and was able to rival with state-of-the-art algorithms dating back to the time of the publication.
However, this approach is not without its limitations, particularly concerning scalability to dense graphs. Future research may focus on designing a dynamic architecture for graphs that could support online inputs, enhancing its flexibility, and conducting comprehensive evaluations across diverse datasets to fully seize the potential of spectral graph CNNs.

References

[1] 2024. Node Classification on Cora - Papers with Code. https://paperswithcode.com/sota/node-classification-on-cora Accessed: 2024-11-30.

[2] SciPy Community. 2024. scipy.sparse.linalg.eigsh. https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.eigsh.html Accessed: 2024-12-01.

[3] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2017. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. arXiv:1606.09375 [cs.LG] https://arxiv.org/abs/1606.09375

[4] Matthias Fey and Jan Eric Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. arXiv:1903.02428 [cs.LG] https://arxiv.org/abs/1903.02428

[5] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. 2009. Wavelets on Graphs via Spectral Graph Theory. arXiv:0912.3848 [math.FA] https://arxiv.org/abs/0912.3848

[6] Dorit Ron, Ilya Safro, and Achi Brandt. 2010. Relaxation-based coarsening and multiscale graph organization. arXiv:1004.1220 [cs.DS] https://arxiv.org/abs/1004.1220

[7] David I Shuman, Mohammad Javad Faraji, and Pierre Vandergheynst. 2016. A Multiscale Pyramid Transform for Graph Signals. arXiv:1308.4942 [cs.IT] https://arxiv.org/abs/1308.4942

[8] Ana Susnjara, Nathanael Perraudin, Daniel Kressner, and Pierre Vandergheynst. 2015. Accelerated filtering on graphs using Lanczos method. arXiv:1509.04537 [math.NA] https://arxiv.org/abs/1509.04537