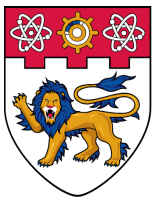


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

On the Experimental Transferability of Spectral Graph Convolutional Networks

Master Thesis
EPFL, Lausanne, Switzerland, June 2020

Submitted to:
Prof. Pierre VANDERGHEYNST
Michaël DEFFERRARD
EPFL

Submitted by :
Axel NILSSON
Under the supervision of:
Prof. Xavier BRESSON
NTU

Contents

Abstract

Spectral Graph Convolutional Networks (GCNs) are generalisations of standard convolutional for graph-structured data using the Laplacian operator. Recent work [26] has shown that spectral GCNs have an intrinsic transferability. This work verifies this by studying the experimental transferability of spectral GCNs for a particular family of spectral graph networks using Chebyshev polynomials.
This work introduces two contributions. First, numerical experiments exhibit good performances on two graph benchmarks [13] [19], on tasks involving batches of graphs, namely graph regression, graph classification and node classification problems. Secondly we study a form of data augmentation through structural edge dropout showing performance improvements for GCNs.
This work contributes to open research with public implementations of all experiments, enabling full reproducibility.

Keywords: Graphs, neural networks, deep learning, transferability, data augmentation, benchmarking.

Acknowledgements

I warmly thank Pr. Xavier Bresson for the opportunity of working with him and welcoming me to the Nanyang Technical University of Singapore. I greatly value his benevolence and implication in the project and am grateful for the fruitful exchanges with the members of his laboratory. I would like to also thank all my supervisors at EPFL supervising the project and making this exchange possible. I am especially grateful to everyone involved in this project, who maintained interest and support during the difficult and uncertain times that we faced with the outbreak of Covid-19. I naturally thank my family for their love and support.

1 Introduction

The mathematical study of graphs can be traced back to Euler in 1973 when the Swiss mathematician resolved the seven bridges of the Könignsberg problem and laid the foundation of what became graph theory and graph topology. In this famous mathematical problem, one can see how the abstract representation of the world that graphs offer, and how it can be applied on a variety of domains.

Graph neural networks (GNNs) [35] has given a way for using deep learning techniques on graphs. With a recent surge of interest in this nascent field, the concept of convolutional neural network (CNN) which has shown great performances on a variety of tasks on Euclidean domains such as computer vision, has been extended to graphs with graphs convolutional networks (GCNs) with spacial [21] and spectral [6] [11] inspiration.
GCNs have showed stellar performance in myriads of domains, among others the representation of social networks to describe communities [21] or fake news detection [32], chemistry [16][12], knowledge graphs [36] [17], physics [10] , recommendation systems [31] [41] and more generally abstract representations of spaces. All of these tasks are noticeably different, and one could think that all relational information can be represented in the form of a graph as they can represent non-Euclidean spaces. Thus, the ability to make predictions about components of graphs or graphs themselves leverages the potential to understand a wide variety of problems.

However, as graphs have an awkward nature without orientation or conformation, some problems remain. Most notably the sparsity of matrices describing graphs is not optimised on most of the current processing hardware which is fuelling the explosion of the capacity of deep learning with CNNs. Recent projects have been building infrastructure to remedy this issue [38] [14] and are enabling the wider use and distribution of GNNs.

In this project we focus on the idea of transferability of spectral GCNs. This is the ability to accurately and consistently predict a feature on a graph domain that is previously unseen by such a model. It has been shown that spectral filters [27] transfer well between graphs and as a natural consequence, so does GCNs [26]. Additional evidence of the transferability of spectral GCNs is the formulation of spectral and spacial GCNs with the message passing paradigm [16]. In spite of that, experimental evidence of the use of spectral GCNs is scarce in the literature. This work supports the idea that spectral GCNs have significant transferability capacity by providing a corpus of experimental evidence showing that the ChebNet [11] performs well across a variety of tasks such as graph classification, graph regression and node classification.

In this report, we will start by formally introducing the notion of graphs, to be able to link the theory of graphs to the one of deep Learning, throughout section 2. This will lead us to be able to formulate a more specific problem statement in section 3. Then, in section 5, we will study the performance of the ChebNet on a series of tasks on sets of undirected, unweighted graphs from different public graph neural network benchmarks [13] [19]. We will use these results to show that the ChebNet has similar, if not better, performances compared to other popular GCNs. Subsequently in section 6, we will look at a novel way of improving the accuracy and the transferability of GCNs by doing structural edge dropout. This is a form of data augmentation, a technique commonly used in other subfields of machine learning but seldom used on graph data and graph deep learning.

In order to have all of the results of this research project to be replicable, an important feature of computational sciences, the code to the majority of the experiments can be found on several open repositories. Throughout the report these will be specified to be easily accessed.

2 Review of spectral graph theory and spectral GCNs

In this section we will first introduce the notion of graphs and their spectral decomposition. Then, we will use these to show how graph signal processing is used for graph deep learning and present the ChebNet [11], a spectral GCN that serves as an experimental reference throughout the rest of the report.

Most of the following material in the following two sections can be linked to books on spectral graph theory [8], seminal papers in the field [5] or courses syllabuses currently used [25].

2.1 Graphs and graph features

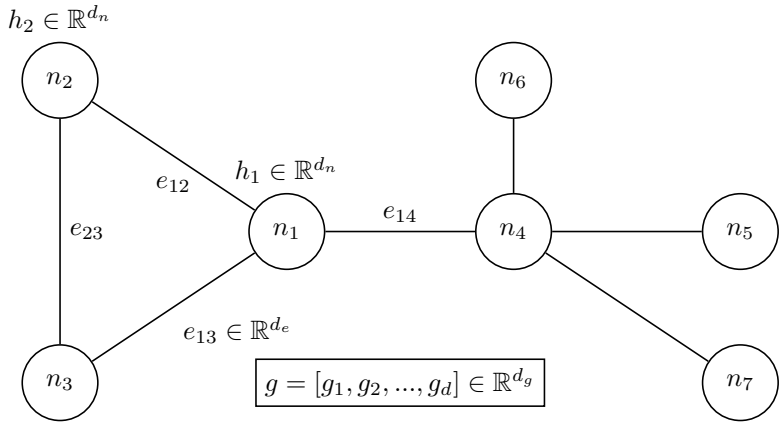


Figure 1: Drawing of an example graph with annotations on nodes and edges.

A graph is noted $\mathcal{G} = \{\mathcal{N}, \mathcal{E}, A\}$, where $\mathcal{N}$ is its set of nodes, $\mathcal{E}$ the set of edges and an adjacency matrix A, also called weight matrix.

The elements of the adjacency matrix A, a_{ij} give information on the type of edge that joins two nodes. In an unweighted graph, a_{ij} acts as a boolean signaling existence or absence of edge between the nodes n_i and n_j. For an undirected graph, the adjacency matrix is symmetric so that $a_{ij} = a_{ji}$. For weighted graph, a_{ij} is a scalar that represents information like distance, connectivity or probability.

A graph can have features associated with each of its components. Actually, the same relational data can be represented by different graphs, depending on the chosen convention, thus features on the graphs holds a significant part of the information of any given graph.

Node features h on the i^{th} node n_i is $h_i = [h_{i_1}, h_{i_2}, ..., h_{i_d}]$ of dimension $\mathbb{R}^{d_n}$. Edge features e_{ij} between two nodes n_i and n_j, is $e_{ij} = [e_{ij_1}, e_{ij_2}, ..., e_{ij_d}]$ of dimension $\mathbb{R}^{d_e}$. The neighbourhood $\mathcal{N}_i$ of a node n_i is the set of all the connected nodes. Graph features are noted g which are feature related to the entire graph like molecule energy of dimension $\mathbb{R}^{d_g}$. These notations are illustrated on a toy graph in Fig.1.

2.2 Spectral graph theory

Introduction to the Graph Laplacian The unnormalised Laplacian matrix Δ_u of a graph $\mathcal{G}$ is a symmetric, positive-semidefinite matrix. It is related to the adjacency matrix by the following formula: $\Delta_u = D - A$ where $D = \text{diag}\left\{\sum_{j \neq i} A_{i,j}\right\}_{i=1}^{N}$ is the degree matrix.

The normalised Laplacian Δ_n, is computed such as:

$$\Delta_n = I_n - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = D^{-\frac{1}{2}} \Delta_u D^{-\frac{1}{2}} \tag{1}$$

The Laplacian is interpreted, or directly related to the smoothness of the graph. Were the signal to be smooth, the Laplacian would be small, and vice versa. The Laplacian is also the solution to the heat equation. The operation of the Laplacian on the feature of a node h_i is the difference between itself and the average of the neighbourhood of the node $\mathcal{N}_i$, such that $(\Delta h)_i = h_i - \frac{1}{d_i} \sum_{j \in \mathcal{N}_i} A_{ij} h_j$.

Spectral decomposition of the Laplacian operator The Laplacian operator, being positive-semidefinite, admits a spectral decomposition such as:

$$\Delta = \Phi^T \Lambda \Phi$$

where $\Phi = [\phi_1, \phi_2, ..., \phi_n]$ are eigenvectors and $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ eigenvalues. This decomposition naturally satisfies $\Delta\Phi_k = \Phi_k\Lambda_k$, k being the k^{th} eigenvalue.

The spectral decomposition of a graph gives an orthogonal basis describing the space of the graph. A visualisation of the eigenfunctions can be seen in Fig.2. One can observe that higher order eigenfunctions corresponds to higher order frequencies on the graph, just as on an euclidian domain.

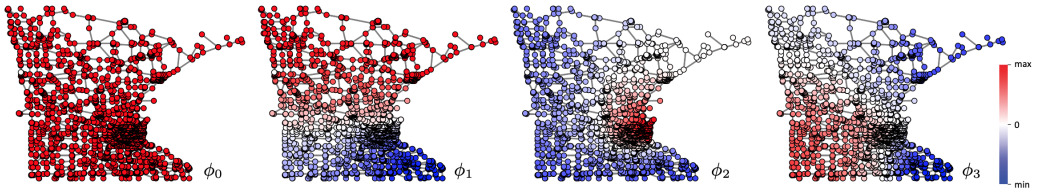


Figure 2: Visualisation of the first four Laplacian eigenfunctions $\phi_0, ..., \phi_3$ on a graph representing the roads of Minnesota. One can see that the eigenfunctions have an orthogonality that is typical of euclidian domains but are impacted by the geometry of the Graph. Extract of figure from [5].

Spectral filtering of graph signals Now let us introduce the notion of Fourier transform on graphs. The basis Φ given by the decomposition of the Laplacian can be used similarly as discrete signal processing. Like signal processing on regular euclidian domains, spectral graph signal processing is dependent of the geometry of the domain.

We can define the Fourier transform $\mathcal{F}$ of a signal h:

$$\mathcal{F}(h) = \Phi^T h = \hat{h}$$

and the inverse Fourier transform, $\mathcal{F}^{-1}$:

$$\mathcal{F}^{-1}(\hat{h}) = \Phi\hat{h} = \Phi\Phi^T h = h$$

This verifies the condition of permutability $\mathcal{F}^{-1} \circ \mathcal{F} = \Phi\Phi^T = I$.

Using the previous equations, we have the convolution theorem with the operator $\star$ between to signals g and h over a domain with eigenvectors Φ by using $\odot$ the Hadaman, or elementwise product:

$$\begin{aligned} g \star h &= \mathcal{F}^{-1}(\mathcal{F}(g) \odot \mathcal{F}(h)) \\ &= \Phi \left(\underbrace{\Phi^T g}_{=\hat{g}} \odot \Phi^T h \right) \\ &= \Phi \text{diag}(\hat{g}_1, ..., \hat{g}_n)\Phi^T h \\ &= \Phi\hat{g}(\Lambda)\Phi^T h \\ &= \hat{g}(\Phi\Lambda\Phi^T)h \\ &= \hat{g}(\Delta)h \end{aligned} \tag{2}$$

Though we have been able to show in Eq.2 that a convolution on a graph domain can be simplified to a multiplication of $\hat{g}$, Δ and h, the computational cost of the eigendecomposition of a matrix is $O(n^2)$.

2.3 Standard convolutional neural networks

Convolutional neural networks (CNNs) are best known in the context of deep learning and computer vision corresponding to a layer with a convolutional kernel/matrix w defined in size, which elements w_i are learned by the network. This kernel is convoluted with the image, meaning translated across

the image and create feature maps by successively applying the kernel to the pixels. Sharing weights, reduces the complexity of models and has shown to improve performances on CNNs by enabling scaling to deeper architectures of CNNs [18].

For graphs, this idea of convolution translates with some difficulty. First of all, in graph there is no ordering of nodes, which means that the notion of direction that exists in euclidian domains simply do not exist. Secondly, the size of the neighbourhood can vary for each node, which means that it is impossible to fix a size of kernel to learn discrete filters.

2.4 Extention to graphs

The original "vanilla" spectral GCN proposed in [6] is as formulated in Eq.3. With d_n the number of features, Φ_k the k^{th} first eigenvectors of Φ and n the number of nodes in a graph, h is $[n \times d_n]$ and h^l is the l^{th} feature map.

$$\begin{aligned} h_i^{\ell+1} &= \xi\left(\sum_{j\in\mathcal{N}_i} \theta_{i,j} \star h_{i,j}^{\ell}\right) \\ &= \xi\left(\sum_{j\in\mathcal{N}_i} \Phi_k \theta_{i,j}^{k} \Phi_k^T h_j^{\ell}\right) \end{aligned} \tag{3}$$

Where $\theta_{i,j}$ is a matrix of (learnable) filters, ξ is a non linear activation function applied element-wise.

This technique has several limitations. First, there is no guarantee of spatial localisation of filters. Second, the filters are basis dependent, which means that they cannot be applied on other basis without significant distortion. In terms of transferability, this means that the filters are altogether theoretically not transferable. In practice it is conceivable that the filter could transfer if there were some correspondence between the domains. Third, the computational complexity of this method is $O(n^2)$ because of the computation of the eigen-decomposition of the graph Laplacian. Lastly, the model need to learn $O(n)$ paramaters per layer.

Though the model presented un [6] has significant drawbacks in practice, it is conceptually a significant milestone for spectral GCNs. Later spectral GCNs models [11] [15] [22] [28] alleviate the pain of matrix decomposition and basis dependence to provide models with lower time and space complexity and higher transferability.

A vanilla 'spatial' graph neural network can be found in [21] a GCN is proposed that uses a single weight matrix θ of size $[1 \times d_n]$ that is multiplied and averaged over all the nodes in the neighbourhood $\mathcal{N}_i$ of a node n_i as in the following equation.

$$h_i^{l+1} = \xi\left(\frac{1}{d_i}\sum_{j\in N_i} A_{ij}\theta^l h_j^l\right)$$

Note that the GCN layer responds to the problems mentioned in the previous section - no ordering of node and varying sizes of neighbourhoods - by using one single parameter. The GCN of [21] performs well on small graphs, the model is too simplistic learning only low pass filters with 1-hop neighbours to have high accuracy on larger graphs or batches of graphs.

Other spacial models [17] [40] build on the same message-passing paradigm but by adding degrees of liberty within the model, show better performances on larger graphs and sets of graphs.

The idea of a distinction between spacial and spectral has been disputed. [16] show that spectral GCNs have valid spacial formulations using the message passing paradigm, [2] come to the conclusion that the distinction arises rather from the inspiration that led to the model building than immutable differences. This has led the spectral methods to typically be wrongly associated with the shortcomings of the vanilla spectral GCN model [6]. In this work we will call spectral methods all that use the Laplacian operator as an essential component of the mathematical formulation of the convolutional layer.

2.5 The ChebNet

The Chebyshev graph convolutional layer proposed in [11] is of spectral inspiration but uses a computed orthonormal basis based on the Chebyshev polynomials. This adds robustness to small coefficient perturbation.

This is the model that will be used and studied throughout the report. Additionally a pooling method is presented and used in [11] but it is never used in this report. Though the performance of the neural network might increase with pooling, not using a pooling layer enables us to better compare the model to other types of convolutional layers.

Compared to other spectral graph neural network the ChebyNet is computationally efficient as it computes recursively the basis T of order k with the following formula:

$$\begin{cases} T_0 = h \\ T_1 = \tilde{\Delta} T_0 \\ T_{k\geq 2} = 2\tilde{\Delta} T_{k-1} - T_{k-2} \end{cases} \tag{4}$$

In order to make the basis orthogonal, a normalisation of the Laplacian is needed. This is done by defining a normalized Laplacian such as:

$$\tilde{\Delta} = 2\lambda_{\text{max}}^{-1}\Delta - I_n \tag{5}$$

So that the eigenvalues $\Lambda = 2\lambda_{\text{max}}^{-1}\Lambda - I$ are in the interval [-1,1], where the Chebyshev polynomes are orthogonal. The effect of this normalisation will be studied and discussed in the following section.

The learned filters, g_θ explicited in Eq.6 are localised in space. The filters are isotropic, limit their capacity of the network to specify, leading for instance to lower performance than a traditional convolutional network (CNNs), which are anisotropic, on MNIST as showed in [11] [23].

$$g_\theta(\tilde{\Delta})h = \sum_{i=0}^{k} \theta_i T_i(\tilde{\Delta})h \tag{6}$$

As one can note GCN [21] is a simplification of the ChebNet with the parameters $k = 2$ and $\lambda_{\text{max}} = 2$ It is interesting that the ChebNet [11] and GCN [21] are sometimes compared as one being spectral and the other being spacial, showing once again that the distinction between spacial and spectral are mostly inspirational.

2.5.1 Considerations on the effect of fixing the largest eigenvalue on the ChebNet

The ChebNet is presented in [11] as a fast GCN with low time and space complexity. In practice, the estimation of λ_{max}, needed for the re-scaling of the Laplacian detailed in Eq.5, for each graph, can be tricky and computationally expensive ($O(n^2)$). Though it is true that this operation needs to be done once for every graph and can be stored with the data as a constant propriety of each graph, it is sometimes impractical to do so without changing the data pipeline, forcing the computation of this value for each graph at each occurrence.

Some [22][23] have found a 'fix' to this problem by fixing the value of $\lambda_{\text{max}} = 2$ for all graphs in all of their experimentation. This is justifiable as it ensures that for any graph, the eigenvalues are within the range of [-1,1] and prevents for a 'worst case'. Experimentally, it reduces computation time. But actually, doing that removes the orthogonality of the Chebyshev basis. Also, the learned filters of the ChebNet described in Eq.6 have a direct relationship to this largest eigenvalue. Thus, fixing an arbitrary number should be somewhat deteriorating the performances of the ChebNet.

2.5.2 Graph embedding layer

To be able to compare the performance of a neural network across graphs of different sizes, the MLP needs to be independent of the number of nodes on a graph. Thus there needs to be an aggregation of the features so that they become independent of the number of nodes. A typical model of that can be seen in Table.11 in appendix A.4. By summing, averaging or taking the maximum value a feature over all nodes, one reduces the dimensions of the featuremaps outputted by convolutional layers $g_\theta(\tilde{\Delta})h$ $[n \times \text{out}]$ to h_θ $[1 \times \text{out}]$.

Aggregating all of the feature maps results in significantly less information for the MLP. Several functions such as max, mean or sum can be used for the aggregation of the feature maps over all nodes. In [40] it is showed that summing is the most expressive function of the ones mentioned previously and therefore the one that degrades the least information from the featuremaps.
In practice, having aggregation functions that combine the information in a surjective way can be a powerful way to improve transferability. Also, the most commonly used aggregation function is "mean" and so to be able to compare models, it will be the default option in this work. Some tinkering with the aggregation function on several of the experimentation in this project never showed superior performance when using the 'sum' instead of 'mean' aggregation function.

3 Problem statement

Transferability for GCNs is a type of generalisation capability. Usually, it designates that if two graphs represent the same underlying phenomenon and the two signals given on the two graphs are similar in some sense, the output of the GCN on both signals should be similar as well. Therefore it is directly related to the capacity of one GCN to learn on one (or set of) graph and make predictions with similar accuracy on unseen graphs of the same kind with consistency.

The innate transferability of spectral GCNs have long been underestimated as they usually were attributed the drawbacks of the vanilla spectral GCN [6]. This has lead to less interest in those compared spacial GCNs, leading to less experimental evidence of their workings in the literature.
The work of Levie et al. [26] [27] debunked the prejudices on spectral GCNs by showing that if two graphs describe the same phenomenon, then a single GCN should have similar repercussions on both graphs. Moreover it emphasises the importance of the underlying manifold that graph discretise, showing that if two graphs discretise the same continuous metric space, then a spectral GCN has approximately the same repercussion on both graphs. These strong claims provides a new way of understanding the performance of spectral GCNs that has not been experimentally assessed before this work. In section 5 we will test a spectral GCN on various datasets and discuss its performance in the scope of the aforementioned claims.

As transferability is a known boundary for some applications of GCNs, some have tried to improve their built-in transferability. One approach is to modify the Laplacian in order to diminish the difference between two domains, making the filters more relevant to the unseen domain. Some state of the art results of domain transfer has been done in [34] by aligning the eigenvalue of the Hamiltonians, or [9] with isospectralization or deforming a shape so that the Laplacian spectrum of each graphs match each other.
In other fields of deep learning, improving generalisation capability is generally done by data augmentation. On graphs however this uncommon though some attempts of trying to predict edge weights to improve the prediction performance have been presented [20], adding new conformations of the same set of nodes [22], which can be understood as specific forms of attention mechanisms [37]. In section 6 we will propose and test experimentally structural edge dropout, a form of data augmentation done by randomly removing edges of graphs during the training phase.

4 Contribution to ChebNet DGL implementation

A significant part of the project was setting up a code-base to do experimentation with the ChebNet. In order to do graph deep learning, an implementation of the ChebNet was done with the Deep Graph Library (DGL)[1] [38]. DGL used in combination with Pytorch [33], leverages high performance by being faster and more memory-friendly than PyTorch Geometric, another graph neural network library.
Though there was initially an implementation of the ChebNet in DGL[2] in the library. This implementation was found to be incorrect. Firstly because the layer was lacking a rectified linear unit (ReLu), which caused the convolutional layer not to perform better when stacked. Secondly because though it was released by the DGL team, it did not use some of the functions optimised for the library.
Other implementations were prior to the use of deep learning frameworks optimised for graphs and used only Tensorflow[3] or Pytorch[4] while others [23] used Pytorch Geometric[5].

A revised implementation for the ChebNet[6] has been proposed and is currently being reviewed to be integrated[7] in the DGL library. This implementation uses the message passing paradigm to leverage the functionalities of DGL and give the fastest implementation of the ChebNet. By using the message-passing functions, DGL optimises in the backend the operations for the sparse matrices that graphs can be represented as.

[1] *http://dgl.ai/*
[2] *docs.dgl.ai/en/latest/modules/dgl/nn/pytorch/conv/chebconv*
[3] *github.com/mdeff/cnn_graph*
[4] *github.com/xbresson/CE7454_2019/tree/master/codes/labs_lecture14/lab01_ChebGCNs*
[5] *github.com/bknyaz/graph_nn*
[6] *github.com/Axeln78/dgl/tree/ChebConv*
[7] *github.com/dmlc/dgl/pull/1460*

5 Benchmarking the ChebNet

The benchmark [13] aims to compare the performance of GCN layers on datasets composed of batches of graphs. This is done by comparing the performance of several models, the most common are: GCN [21], GraphSage [17], MoNet [30], Gated-GCN [4], GAT [37] and GIN [40].

There are several advantages of using the material of the benchmark presented in [13]. Firstly, the datasets are public and were design by the authors to reflect a wide series of tasks. Some, like the SBMs and SuperPixels are generated e.g. are not natural. Therefore, unlimited amounts of sets of graphs can be generated. Secondly, the performance results become comparable to the performance of the other aforementioned models given in the publication.

In practice it is difficult to compare models as the tweaking of some hyper-parameters and changes in architecture can strike a significant difference in performance. By fixing a budget of $100,000$ parameters per model, the authors try to introduce a sense of fairness that compares graph convolutional layers by their design. The graph convolutional layers are put in similar GCNs with a maximum of four convolutional layers (L=4) and three layers of MLP after that.

The benchmark also constraints the learning hyper-parameters for the training of the GCNs. There is batch normalisation with batches of 128 graphs. The learning follows a scheduler with an initial learning rate of $\text{lr} = 0.001$, with a reduce factor of 0.5 and a learning rate scheduler patience of 5 epochs. The minimum learning rate is $1 * 10^{-5}$.

For the ChebNet, the number of parameters vary with the order of the Chebyshev polynomial k. We will chose k accordingly to best performance of the model for each task, which is found to be proportional to the mean number of nodes per graph for each dataset.

Implementation All of the experiments are replicable with the code found on the repository[8]. For a given task (ZINC in this example), the files that were added or changed are as follows:

```
benchmarking_gnns/
├── configs/
│   └── molecules_graph_regression_Cheb_ZINC.json
├── Layers/
│   └── Cheb_layer.py
├── nets/
│   └── molecules_graph_regression/
│       ├── ChebNet.py
│       └── load_net.py
└── script_main_molecules_graph_regression_ZINC.sh
```

All of the additional code needed to add the ChebNet to the 'benchmarking_gnn' repository of [13] will proposed as a contribution to the project.

Results summary In order to be able to easily compare the performance of the ChebNet on the tasks of the benchmark of [13], the accuracies of all of the tasks are summarised in Table.7 in appendix.A.4. We will review the tasks of graph classification with MNIST and CIFAR, graph regression with ZINC and node classification with SBM CLUSTER .

5.1 Graph classification with the SuperPixel datasets

Super-pixels represent small regions of homogeneous intensity in images, and can be extracted with the SLIC technique presented in [1] and used in similar settings in relevant literature [23][24]. The SuperPixel datasets are made by doing just that on the images of the MNIST and CIFAR datasets to turn them into graphs. The resulting graphs vary in sizes, thus the models used for graph classification have a graph embedding layer. Fig.3 shows graphs of the MNIST SuperPixel dataset. For both graph classification tasks, the same model configuration is used with $k = 4$, both can be seen in Table.10 in appendix A.4.

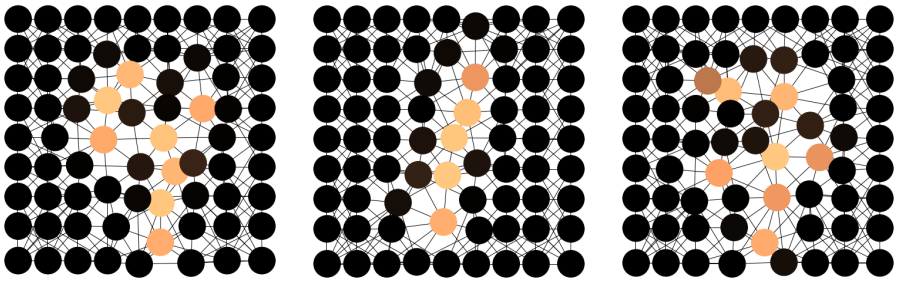
5.1.1 MNIST SuperPixel

Unlike the original MNIST dataset which has only one signal channel corresponding to greyscale intensity, the SuperPixel MNIST uses three channels: mean greyscale intensity and the center of

[8] *github/Axeln78/benchmarking-gnns/tree/ChebNet*

Dataset	Model	# parameters	Accuracy $\lambda_{\max} = 2$	epoch/total
MNIST	L=4	100001	96.2625 ± 0.106 %	46 s/ 0.81 hrs
	L=16	387365	96.3125 ± 0.338 %	95 s/ 1.69 hrs
CIFAR10	L=4	100155	62.2125 ± 0.453 %	60 s/ 0.95 hrs
	L=16	387519	64.4075 ± 0.548 %	100 s/ 1.7 hrs
MNIST	GCN	101365	90.7050 ± 0.218 %	
	GraphSage	104337	97.3400 ± 0.143 %	
CIFAR10	GraphSage	104517	65.7670 ± 0.308 %	
	GatedGCN	104357	67.3120 ± 0.311 %	

Table 1: Summary of the performance of the ChebNet on the SuperPixel datasets. The accuracy is the mean and standard deviation of the classification score of four models trained with four random initialisation. L is the number of convolutional layers for a given model. Additionally two models for each dataset have been added from [13] as elements of comparison



(a) 4 (b) 1 (c) 7

Figure 3: Three sample graphs from the SuperPixel MNIST dataset with their corresponding labels.

mass (x,y) of the the pixels. This gives an information of orientation of the graph to the neural network. The number of nodes per graph varies between 40 to 75.

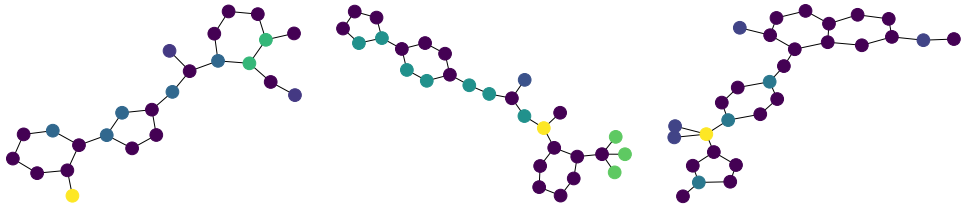
Results The testing accuracy is $96.2625 \pm 0.1056\%$ which, comparatively to the benchmark corresponds to a top-tier performance and would rank fourth compared to all the other models. The training accuracy is 100% for all four trained models, which shows that the model is capable of overfitting the data.

5.1.2 CIFAR10 SuperPixel

This classification task is more difficult than classifying the MNIST dataset. Because the classes like 'plane' or 'dog' are more abstract, and thus harder to represent in a SuperPixel graph. Contrary to the MNIST SuperPixel dataset, a human could not label correctly these SuperPixel graphs with high accuracy. The features on each node is 5-dimensional with RGB information as well as the center of mass of each node on the original image, giving information on the orientation of the graph to the model. There are on average 100 nodes per graph. The used model is represented in Table.10 in appendix A.4.

Results The accuracy being $62.2125 \pm 0.4526\%$, the network lands in the middle compared with the other models of the benchmark and would be ranked fourth. The training accuracy being consistently above 90% and the MNIST graph classification having high accuracy leads to think that this result is linked to the difficulty of the task rather than inability to learn of the model.

For the original CIFAR10 dataset, it is well known that additional data augmentation is necessary to reach performances beyond about 60%. Though the performances are not comparable because of the constraints given by the benchmark, [23] obtains higher accuracy (68,92%) by using the ChebNet in a setting of batches of graphs with different coarsening of the images to graphs.



(a) -0.2070 (b) 0.5230 (c) 0.8350

Figure 4: Three graphs from the ZINC dataset, in caption are the labels or values to predict from the graph. The node colours corresponds to a categorical label for each kind of atom.

5.2 Graph regression with molecular dataset - ZINC

The ZINC dataset is a set of graph representing molecules of varying sizes and connectivity as represented in Fig.4.

This is a graph regression task as the graph neural network has to predict the constrained solubility of each molecule, or one value per graph. The accuracy is determined by the mean average error of the $L1$ loss between the graphs of the test set. Node features are categorical and correspond to the type of atom each node represent. Similarly, the edge features represent the type of bond between two atoms. The edge features are not used in any experimentation. The number of graphs in the training- validation- and test- set are respectively 10000, 1000 and 1000 graphs.

The architecture of the model can be seen in Table.11 in appendix A.4. The chosen order of Chebyshev polynomials is $k = 2$, because the graphs are small (9 to 37 nodes/atoms). A graph embedding layer is added at the end of the convolutional layers taking the mean of the node features. An embedding layer is added at the beginning of the model in order to be able to create an embedding from the categorical labels of the nodes.

Dataset	Model	# parameters	Accuracy	Accuracy $\lambda_{\max} = 2$	epoch/total
ZINC	L=4	101230	**0.3304 ± 0.0210**	0.3408 ± 0.041	135 s/ 3.5 hrs
	L=4 w rsd	101230	0.4099 ± 0.0048		171 s/ 3.2 hrs
	L=16	374710	**0.2680 ± 0.0184**		39 s/ 1.24 hrs
	L=16 w rsd	374710	0.2834 ± 0.0066		38 s/ 1.10 hrs
ZINC	GatedGCN	105735	0.4350 ± 0.011		
	GatedGCN-E-PE	505011	0.2140 ± 0.006		

Table 2: Summary of the performance of different ChebNets on the ZINC Dataset of the benchmark [13]. The accuracy is the mean accuracy and standard deviation of four models trained with four random initialisation. L is the number of convolutional layers and models that use layer-wise residual connections are noted 'w rsd'. Two of the best models from [13] have been added as elements of comparison.

Results Molecular regression task are known to be difficult because the datasets are often severely imbalanced and that the number of pieces of data is limited. This benchmark [13] choses to take an initial random split for the train/val/test set while [19] splits by scaffolding pointing out that random splits yields lower performances.

On this task the ChebNet outperforms the other models. The accuracy for the ChebNet on this dataset can be seen in Table.2. The MAE is 0.330375 ± 0.02106 which places it as the top contender of the benchmark, taking the first place from the Gated GCN [4] models which additionally use edge features.

Fig.5 show the prediction of a trained ChebNet on the train/val/test sets of the dataset. It shows that the distribution of labels is unbalanced, with few molecules with very low solubility. This might explain why the predictions of the trained model are better in the region where there are more examples for a given solubility, a plateau in 5. Moreover the Fig.5 shows fringes between the predicted values. This can be an effect of the $L1$ loss, the disparate distribution of molecules or the

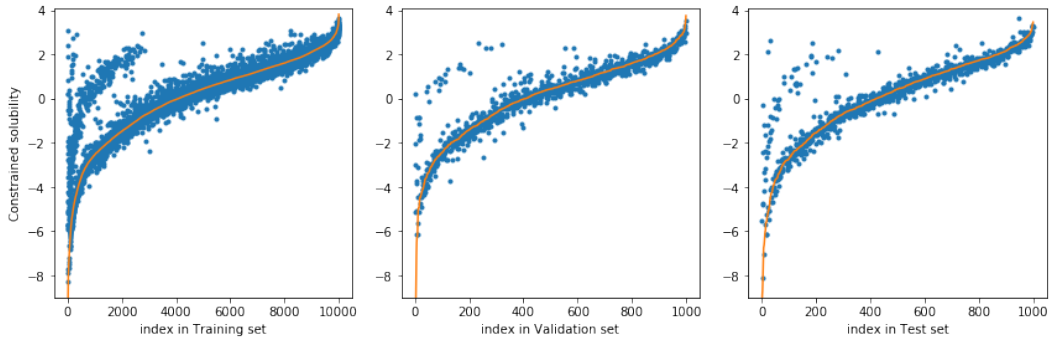


Figure 5: Distribution of labels and predictions on the ZINC Dataset sorted by labels. In orange the labels, and in blue the predictions for each molecule. One can see that the labels of the dataset is imbalanced which lead to the variance of the predictions being greater for molecules with low solubility, as they are underrepresented in the training set.

inherent geometry of the problem. In order to distinguish between those, the code to produce these visuals will be pushed to the original repository[9] to compare the predictions distributions across models.

5.3 Node classification with Stochastic block model - SBM

The stochastic bloc model (SBM) graphs have several clusters of communities. There is an intra-probability p of being connected to another node in the same communities and a extra-probability q of being connected to a node in another communities. SBM have been shown to be a good representation of social networks [29].

With this dataset, two tasks are proposed in [13]. Firstly the CLUSTER task which is a classification task of the label of each node correspondingly to different communities. Secondly the PATTERN task which is a pattern recognition and semi-supervised graph clustering. Unfortunately, experimentation showed that the loss used in [13] was incorrect. After correction, the PATTERN task became too easy and will not be presented in this work.

In the CLUSTER dataset, each graph has 6 communities of 5 to 35 nodes connected together. The weighted cross-entropy loss and accuracy metric are averaged across communities in both CLUSTER so that the model is penalised equally for miss-labelling communities of different sizes.

Although the graphs are different in sizes, the output of the neural network is proportional to the number of nodes. Thus there is no graph embedding layer for this model. The parameter $k = 5$ has been chosen because the graph varies to sometimes be large with 40-190 nodes for CLUSTER. The communities themselves are smaller and the diffusion proprieties should correspond to the size of the communities rather than the full graph. The model can be seen in Table.12 in appendix A.4.

Dataset		# parameters	Accuracy	Accuracy $\lambda_{\max} = 2$	epoch/total
CLUSTER	L=4	102745	**72.8968 ± 0.197**	72.4338 ± 0.213	103 s/ 2.1 hrs
	L=4 w rsd	102745	72.7414 ± 0.211	**73.0887 ± 0.295**	103 s/ 2.1 hrs
	L=16 w rsd	399055	**74.5450 ± 0.306**		115 s/ 1.7 hrs
Gated-GCN	L = 4	104355	60.404 ± 0.419		
	L= 16	502615	73.840 ± 0.326		

Table 3: Summary of the performance of different ChebNets on the ZINC Dataset of the benchmark [13]. The accuracy is the mean accuracy and standard deviation of four models trained with four random initialisation. L is the number of convolutional layers and models that use layer-wise residual connections are noted 'w rsd'. Two versions of the Gated-GCN from [13] are added as elements of comparison on the same dataset.

[9] *github.com/Axeln78/benchmarking-gnns/tree/ZINC_visualization*

Results The results for this task is shown in Table.3. The ChebNet outperforms all other models in the benchmark with its highest accuracy of 72.8968 ± 0.197. It outperform the best model of [13], the Gated-GCN, by 12 percentage points. These are outstanding performance with four layers of convolution $L = 4$, and the corresponding number of parameters. However, the accuracy does not significantly improve when using a deeper model ($L = 16$). This is possibly not because of a shortcoming of the model but rather that the task is difficult as no model show significantly better performance in the benchmark.

5.4 Open Graph Benchmark graph regression tasks

Additionally to the benchmark in [13], graph regression tasks on the Open Graph Benchmark (OGB) [19] has been studied. The OGB benchmark proposes a range of datasets that are derived from nature. In the graph regression tasks, the datasets are all chemistry related with graph representations of molecules [39]. The task that we will consider is the prediction of whether a molecule prevents HIV virus replication. The graphs are of the same nature as those used in ZINC, and could be illustrated with something comparable to Fig.4.

All of the experimentation is replicable by using the code in the following repository[10]. For a given task, the files that were added or changed are as follows:

```
ogb/examples/graphporppred/mol/
├── conv.py
├── gnn_dgl.py
└── main_dgl.py
```

The OGB project maintains a leaderboard[11], where the performance of different models can be compared. The performance of the ChebNet for some of the tasks can be seen in Table.4 and we will compare them to the other models currently on the leaderboard in the following sections.

To be able to compare any results to the OGB benchmark, mostly the same learning parameters are used as in [19]. The training is done with batches of sizes of 128 graphs with batch normalisation. There is fixed number of 100 epochs for training. Models have residual connections at each layers and has no dropout in input or output of the model.

The largest eigenvalue parameter $\lambda_{\text{max}} = 2$ is fixed for all graphs because of the non convergence of some of the matrix decomposition of graphs in the datasets. Moreover, the width of each GCN layers are limited to a maximum of 300 in the OGB benchmark, but because the ChebNet does not work like the other models and makes its own basis, a comparable number of parameters can be obtained by using a width of 150 with $k = 3$ that makes every layer a 450 to 150 wide layers.
The learning rate has been lowered to 0.0001 from 0.001 as it showed a significant increase in performance. Even though it is different from the standards of the benchmark, this change seem justified and comparable because they do not take away the most constraining training parameters.

In [19] the models are tested with and without a virtual node. Adding a virtual node a technique for improving performance of the graph neural network consisting in connecting a 'virtual' node to all nodes of the graph. This gives a node to which all the information of the whole graph is connected and helps making a meaningful representation of the graph in the graph embedding layer. Though it is possible that an additional virtual node would improve the performance of the ChebNet, like it does for the other models, it is beyond the point of assessing the quality of the model in the scope of this work. Therefore no results with a virtual node are given.

The model used to get these performances can be seen in Table.9 in appendix A.3 and the results are the average and standard deviation for the losses of 10 models trained with the aforementioned parameters with randomly initialised weights.

5.4.1 Molhiv

MOLHIV is the smallest of the two OGB datasets with a set of 41'127 graphs. The metric for comparing models is a ROC AUC.

Comparing to the leaderboard, the ChebNet would rank second, after the GatedGCN, which was added to the OGB posterior to publication. Also the ChebNet would rank better than GCN and GIN,

[10] *github.com/Axeln78/ogb*
[11] *ogb.stanford.edu/docs/leader_graphprop/*

Dataset	# graphs	Metric	Accuracy (%)		
			Best Train	Val	Test
MOL-HIV	41'127	ROC-AUC	0.9992	0.8490	**0.7631± 0.0127**
MOL-PCBA	437'929	PRC-AUC	0.5417	0.2387	**0.2317 ± 0.0036**

Table 4: Accuracy of the ChebNet, with corresponding metrics for two tasks of graph regressions on molecule graphs of the OGB datasets. The shown results are the mean and standard deviation for 10 models initialised with random weights.

which are the reference GCNs of the paper. It is interesting to see that the best accuracy during the training phase on the Train set is close to 100% which shows that the model has the capacity to learn with high accuracy on such a setting with batches of graphs.

5.4.2 Molpcba

MOLPCBA is a more complete task thank MOLHIV as it has a larger set of graphs (437'929) and the performances are evaluated with PRC AUC.

For this task, the ChebNet performs better than any other model if we do not consider those using without virtual nodes. Actually, a closer look at the results in [19] reveal that the models exhibits a large improvement in performance by adding a virtual node.

Interestingly, the ChebNet outperforms the Gated-GCN on this task but not the MOLHIV. This is possibly because the sets of data is larger in this task than in MOLPCBA, which could favour the ChebNet over the Gated-GCN.

5.5 Discussion

Putting the ChebNet model up against public benchmarks gives experimental proofs that the model has similar, if not better, performance on four tasks of [13] and two of [19] than the most common GCNs currently used. Most notably the ChebNet shows state of the art performance on molecule graph regression with the tasks of ZINC & MOLPCBA, and semi-supervised node classification with SBM-CLUSTER.

This builds up to strong experimental proof that models considered 'spectral' can outperform models considered 'spacial' on tasks such as graph regression, graph classification and node classification in a semi-supervised manner when the datasets are collections of graphs. Moreover, these experiments made on batches of graphs provides a body of experimental proof for the claim that spectral GCNs have inherent transferability capacities [26].

One can note that both benchmarks used in this work were published in 2020, yet none of them include spectral GCNs as a default comparison. This shows a general misunderstanding of the capabilities of spectral GCNs though some work [22] [26] corroborate the results of this report.

Let us now discuss in further details the results of several tasks with regards to the considerations of [26] in the attempt to better understand the workings of the ChebNet and spectral GCNs. These considerations are that spectral GCNs should have good performance on datasets made out of discretisation of an underlying continuous manifold.

SuperPixels Datasets The performance of the ChebNet on these dataset are average compared to other GCNs, and worse compared to CNNs on the original datasets. This supports the observation of [13] stating that GCNs that learn isotropic filters could not exceed the accuracy of other models learning anisotropic filters.

ZINC That the ChebNet outperform other models is a significant and unexpected result. Even though [22] show that spectral graph convolutional layers could learn on graphs of arbitrary size and structures in similar settings with graphs representing molecules.

Firstly, because as molecules are made of relatively small graphs and that not all atom combination make molecules makes significant differences between each graph. The large variation in structure of the graphs is considered to be a drawback for spectral GCNs because any space describing the molecules would not be densely mapped.

Secondly, the question of whether all the graphs lie in the same underlying continuous space is relevant, as [26] argues it could explain the remarkable performance of the model.
On the one hand, if we consider that all of the graphs live in the continuous three dimentional space that we live in, then it might not be a surprise that the ChebNet is able to generalise proprieties across graphs. If that is the prevalent consideration then this implies that the ChebNet should generalise well for all graphs representing structures that live in the real world. However, unlike in the Super-Pixel experiments, the model does not get any information on the position of the nodes relative to each other. Thus, the model would need to infer a three dimensional positioning of the molecules, which seems improbable.
On the other hand, if we consider that each molecule represent a domain of itself, then each of these continuous domains would be vastly different. Though one could always argue that molecules could be samplings or points on a non-euclidian manifold, it is highly improbable that a representation of this manifold could be mapped by the observation of (10'000) graph molecules. In this case it possible that the sensitivity to specific domains have been overestimated for the ChebNet.

To sum up, none of the aforementioned considerations on the underlying manifold from which the data originates are convincing to the point of explaining the good performances of the ChebNet.

SBM - Cluster The fact that the ChebNet performs significantly better, of about 10 percentage points, than the other models of the benchmark is astonishing as the ChebNet has not been used in the literature for semi-supervised node classification unlike GCN in [21].

If we consider the underlying manifold that could generate the discretisation of CLUSTER, one can find that this is a continuous non-euclidian manifold. The theory of [26] predicts that the a spectral GCN should transfer with minimum error which is supported experimentally in this case by the strong performance of the ChebNet on CLUSTER.

Model depth One observation that arose from this project is that the Chebyshev convolutional layer can be stacked. This is not a generality for GCNs as some well known models like GIN do not improve performance when the convolutional layers are stacked. The experimental data of Table.1,2&3 shows that for different tasks, the deeper ChebNet with 16 convolutional layers outperform the comparative 4 layer model.

The impact of fixing $\lambda_{\max}$ on the ChebNet The effect of fixing or not the largest eigenvalue has been studied throughout this project. Experimentally we see that when $\lambda_{\max} = 2$ improves the performances for SBM CLUSTER but deteriorates it for ZINC. This work provides a body of experimental results that, without confirming, supports the hypothesis proposed in [22] which states that for every order of Chebyshev polynome k the learned filters become less sensitive to the eigenvalues of a graph. In practice this means that the ChebNet could be able to learn filters that are less sensitive to each graph for each order of polynome k.

6 Improving transferability with data augmentation

While data augmentation is used extensively in computer vision and other fields related to deep learning as a way to help neural networks with generalisation, it is not common practice in the graph deep learning community. Augmentation is a type of alteration of the data, which does not influence the label of a piece of data, or called task invariant.
In computer vision, augmentations are usually relatively small deformations or tweaks on an image that does not change significantly the content, or the information encoded in an image. In natural language processing (NLP) augmentation can be done under some constraints [7], usually linked to not changing the meaning of sentences.

However, on graph domain, there is no straightforward way of doing augmentation.
Though it is true that some augmentation techniques can be borrowed from other fields and used on specific datasets, the task invariants remain task dependent. In the case of the MNIST on a lattice dataset, similar augmentation to computer vision can be done changing only node features, but these require a strong understanding of the task. Furthermore, these augmentation would strictly be a replication of the computer vision type, and would not take advantage of the medium that is graphs.

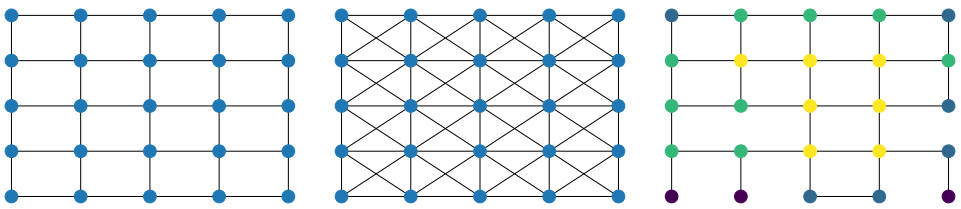
Graphs are the only domain on which structural data augmentation is possible. This is, to modify the connections of graphs, in practice the adjacency matrix A, during the training phase to improve the learning capabilities. However, changing the structure of graphs by modifying the adjacency matrix is equivalent to changing domain as each graph is its own space. Thus it poses the problem of knowing if this change of domain is beneficial or not to a GCN and in which tasks structural augmentation can be done.

6.1 Naive transferability of the ChebNet on the MNIST Dataset

In this section we will use the MNIST dataset to study the 'built-in' transferability of the ChebNet. This is, the ability of the model to predict on unseen domains. This can be seen as an extension of the work in [11], who presented the ChebNet model and showed its performances on the MNIST Dataset.

The MNIST dataset is composed of 28x28 pixels images with one greyscale channel. These images can be transposed to graphs by using lattice graphs, or 2D grids. Fig.6 (a) and (b) shows different configurations of these lattice graphs by choosing to connect to the 4 or 8 nearest neighbours (NN) respectively.

When a ChebNet with an architecture described in Table.8 is trained on MNIST on a 4 NN lattice and achieves 96.97% testing accuracy, the performance of the same model, trained on a 4 NN lattice but tested on a 8NN, the accuracy drops significantly to 20.95%. This gives some form of baseline for the naive transferability of the ChebNet. This naive transferability is poor enough that we should ask ourselves why, and eventually how to improve it.



(a) 4 nearest neighbours (b) 8 nearest neighbours (c) 4 NN with randomly removed edges

Figure 6: Different types of planar lattices. These lattices can all be considered samplings of the planar continuous space. In (c), the node colours are related to their degree.

6.2 Structural edge Removal

In this experiment a Chebnet is trained while doing structural edge dropout. During the training, a random proportion between 0% and 25% of all the edges are randomly removed for each graph, yielding lattices such as illustrated in Fig.6(c).

Lattice of evaluation	ChebNet - 4 NN	ChebNet - 4NN rand. edge rm.
4 NN - Fig.6.a	96.97%	96.04%
8 NN - Fig.6.b	20.95%	**95.56%**
8 NN, $\lambda_{\max} = 2$	**73.49%**	95.85%

Table 5: Accuracy of two ChebNet models, one trained on a 4NN lattice and the other on a 4NN lattice with randomly removed edges, on three different settings for testing. $\lambda_{\max}$ references the re-scaling parameter of Eq.5

The result of the experiment is that the learned filters have a better performance even outside the region of training and on different lattices. When the model trained on randomly removed edges is used on a testing set built on a 8 NN lattice, the accuracy drops from 96.04% to 95.56%. These result are summarised in Table.5.

Two things can be noted from this result. Firstly that by doing the random edge removal during the training, we dramatically improve the transferability of the network between the 4 to 8 NN lattices. Secondly, the testing accuracy on the 4NN lattice is lower, this leads to believe that the learned filters are of a different nature.

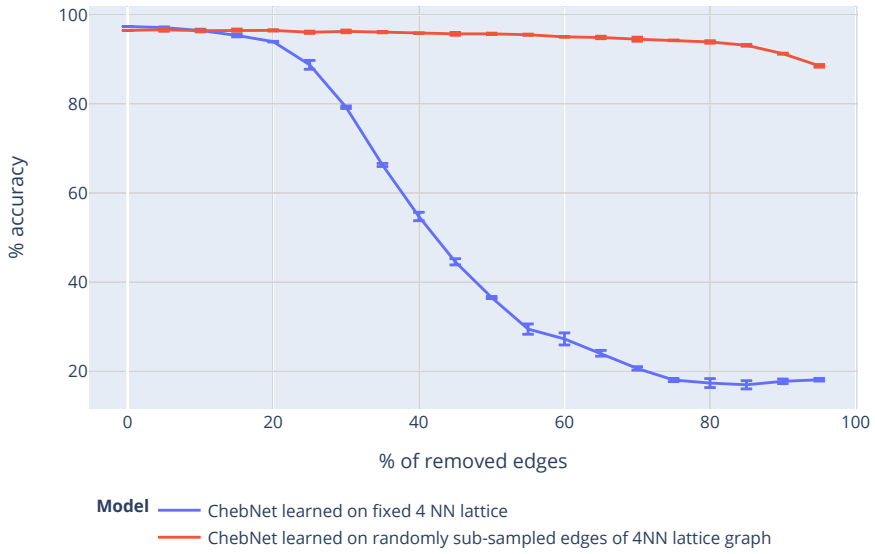


Figure 7: Classification accuracy over the test set of MNIST for two ChebNet models over different proportions of removed edges on the test set, with confidence intervals. One model (blue) is trained on the MNIST dataset on a 4 NN lattice while the other (red) was trained on a 4 NN lattice with structural edge dropout of up to 25% of removed edges during the training phase. The model trained with structural edge dropout (in red) exhibits better transferability capacity than without (in blue), even outside of its region of training.

By studying the accuracy of the two models for an increasing number of removed edges, as displayed in Fig.7 we can get information on the consistency of the GCN. The network trained on a set of graphs has an accuracy that stays consistently over 85% of accuracy outside of the region of training (up to 25% of removed edges). It becomes apparent that the transferability capacity of the ChebNet trained on randomly removed edges improves beyond the training region.

This evidence shows that this type of graph augmentation changes the problem where the built-in transferability of the ChebNet is needed but rather its capacity to learn common features accross batches of graphs. We there see that the ChebNet performs better because the nature of the problem becomes different.

Benchmark experiments To strengthen these claims, additional experimentation has been done on other datasets of the benchmark [13], namely MNIST SUPERPIXEL, CIFAR10 SUPERPIXEL and SBM CLUSTER. In these experiments, a random proportion between 0% to respectively 15% and 30% of edges are removed during the training phase for each dataset. All of the code needed for reproduction of these experiments can be found in the following repository[12]. The files needed for this are structured as follows for the Superpixel task:

```
benchmarking_gnns/
├── main_superpixels_graph_classification_aug.py
└── train/
    ├── augmentation.py
    └── train_superpixels_graph_classification_aug.py
```

Table.6 show that the testing accuracy systematically improve the performance of the ChebNet improves when doing random edge removal.

Dataset \ Edge rm. rate	0% (reference)	15%	30%
MNIST SUPERPIXEL	96.2625 ± 0.1056	96.6050 ± 0.1933	$\mathbf{96.6475 \pm 0.1466}$
CIFAR10 SUPERPIXEL	62.2125 ± 0.4526	65.9650 ± 0.6810	$\mathbf{66.3875 \pm 0.8126}$
SBM CLUSTER	72.7414 ± 0.2110	$\mathbf{73.1050 \pm 0.1369}$	72.5186 ± 0.3642

Table 6: Testing accuracy (in %) of a ChebNet model on different tasks. The GCNs are trained with randomly removed edges (15% and 30%) but the test set is original. In bold are the best performances. One can see that structural edge dropout improves the performances for each of the respective task of graph classification MNIST,CIFAR and graph regression SBM CLUSTER.

For the graph classification tasks of MNIST & CIFAR SuperPixel dataset, the testing accuracy gains on average 0.4 and 4 percentage points, respectively. For CIFAR, the gain in performance of the model is large as it outperforms the deeper version of the ChebNet with 16 layers (Tab.1 or 7). Meaning that the structural edge dropout made a more significant improvement than adding learning capacity to the model.

For SBM CLUSTER the improvement on the accuracy is of about .5 percentage points. Although this is does not seem very impressive, one should note that this is shoulder to shoulder with the best performance of any model with any depth of architecture in the benchmark.

Unfortunately, there is a limit to the possibilities of the structural edge dropout technique as we cannot use it on graph regression tasks such as all those related to the molecular datasets like ZINC, MOLHIV or MOLPCBA. This is because molecules are small, very sparse graphs, and thus removing edges destroys what makes a molecule a molecule. This shows that this data augmentation technique too need some understanding on the underlying data for a given task.

7 Conclusion

This experimental work examine the transferability capacity of spectral GCNs on two graph benchmarks. Numerical experiments show that ChebNet provide state of the art performance on ZINC and CLUSTER and close to the best results on MOLHIV and MOLPCBA.
This supports experimentally recent demonstrations [26] that spectral GCNs have good performance when graphs are discretisation of a continuous manifold in view of the performance of the Chebnet on the SBM CLUSTER dataset.
We show that structural edge dropout, a proposed data augmentation technique, can not only increase the performance but also the transferability of GCNs. Experimentally this is shown by significant performance improvements on MNIST, CIFAR and CLUSTER.

In this work, only one type of spectral GCN is evaluated, complementary work could be to compare the performance of other, more advanced, spectral GCNs [28] [3] [2] on the same tasks to strengthen the claims made in this report. Moreover the proposed structural edge dropout method need to be tested further and compared to feature dropout. The same extends to node structural and feature dropout. Finally all of the observations of this report should be studied for very large graphs, that need to be split during the training phase, providing with erroneous Laplacians.

[12] *github.com/Axeln78/benchmarking-gnns/tree/dev*

A Appendix

A.1 Result Summary for benchmarking gnns

Dataset		# parameters	Accuracy	Acc. $\lambda_{\max} = 2$	epoch/total
MNIST	L=4	100001		96.2625 ± 0.106 %	46 s/ 0.81 hrs
	L=16	387365		96.3125 ± 0.338 %	95 s/ 1.69 hrs
CIFAR10	L=4	100155		62.2125 ± 0.453 %	60 s/ 0.95 hrs
	L=16	387519		64.4075 ± 0.548 %	100 s/ 1.7 hrs
ZINC	L=4	101230	**0.3304 ± 0.0210**	0.3408 ± 0.041	135 s/ 3.5 hrs
	L=4 w rsd	101230	0.4099 ± 0.0048		171 s/ 3.2 hrs
	L=16	374710	**0.2680 ± 0.0184**		39 s/ 1.24 hrs
	L=16 w rsd	374710	0.2834 ± 0.0066		38 s/ 1.10 hrs
CLUSTER	L=4	102745	**72.8968 ± 0.197**	72.4338 ± 0.213	103 s/ 2.1 hrs
	L=4 w rsd	102745	72.7414 ± 0.211	**73.0887 ± 0.295**	103 s/ 2.1 hrs
	L=16 w rsd	399055	**74.5450 ± 0.306**		115 s/ 1.7 hrs

Table 7: Summary of the performance of the different ChebNets on the benchmark. The accuracy is the mean accuracy and standard deviation of each metric defined for each task in [13] of four models trained with four random initialisation. In bold, models performing state of the art performances compared to the other models of the benchmark. L is the number of convolutional layers and 'w rsd' marks the addition of layer-wise residual connections.

A.2 ChebNet architecture for test of naive transferability

Layer type	In features	Out features	k	Activation	Bias
ChebConv	1	32	10	Relu	Yes
ChebConv	32	50	10	Relu	Yes
Dense	39200	512		Relu	Yes
Dropout				$p = 0.5$	
Dense	512	10			Yes

Table 8: LeNet5 architecture ChebNet. This is used on the 28×28 MNIST lattice and the number of nodes (784) $\times$ the number of features (50) is the input size of the MLP (39200).

A.3 Model for benchmarking OGB

Layer type	In features	Out features	k	Activation	Bias
OGB AtomEncoder	9	150			
ChebConv	150	150	3	ReLu	Yes
ChebConv	150	150	3	ReLu	Yes
ChebConv	150	150	3	ReLu	Yes
ChebConv	150	150	3	ReLu	Yes
ChebConv	150	150	3	ReLu	Yes
Graph pooling	any	150		"mean"	
Dense	150	1			Yes

Table 9: ChebNet for OGB - MOLHIV / MOLPCBA

A.4 Models for benchmarking GNNs

Layer type	In features	Out features	k	Activation	Bias
Embedding	3 (5)	77			
ChebConv	77	77	4	ReLu	Yes
ChebConv	77	77	4	ReLu	Yes
ChebConv	77	77	4	ReLu	Yes
ChebConv	77	77	4	ReLu	Yes
Global pooling	any	77		"mean"	
Dense	77	38		ReLu	Yes
Dense	38	19		ReLu	Yes
Dense	19	10			Yes
Total number of parameters				100001 (100155)	

Table 10: ChebNet for SuperPixel MNIST and CIFAR in '( )'

Layer type	In features	Out features	k	Activation	Bias
Embedding	28	106			
ChebConv	106	106	2	ReLu	Yes
ChebConv	106	106	2	ReLu	Yes
ChebConv	106	106	2	ReLu	Yes
ChebConv	106	106	2	ReLu	Yes
Global pooling	any	106		"mean"	
Dense	106	53		ReLu	Yes
Dense	53	26		ReLu	Yes
Dense	26	1			Yes
Total number of parameters				101230	

Table 11: ChebNet for ZINC

Layer type	In features	Out features	k	Activation	Bias
Embedding	7	70			
ChebConv	70	70	5	ReLu	Yes
ChebConv	70	70	5	ReLu	Yes
ChebConv	70	70	5	ReLu	Yes
ChebConv	70	70	5	ReLu	Yes
Dense	70	35		ReLu	Yes
Dense	35	17		ReLu	Yes
Dense	17	6			Yes
Total number of parameters				102745	

Table 12: ChebNet for SBM - CLUSTER

Bibliography

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and Sabine Süsstrunk. SLIC Superpixels Compared to State-of-the-Art Superpixel Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282, November 2012.

[2] Muhammet Balcilar, Guillaume Renton, Pierre Heroux, Benoit Gauzere, Sebastien Adam, and Paul Honeine. Bridging the Gap Between Spectral and Spatial Domains in Graph Neural Networks. *arXiv:2003.11702 [cs, stat]*, March 2020.

[3] Filippo Maria Bianchi, Daniele Grattarola, Cesare Alippi, and Lorenzo Livi. Graph Neural Networks with convolutional ARMA filters. *arXiv:1901.01343 [cs, stat]*, October 2019.

[4] Xavier Bresson and Thomas Laurent. Residual Gated Graph ConvNets. *arXiv:1711.07553 [cs, stat]*, April 2018.

[5] Michael Bronstein, Bruna, Joan, Szlam, Arthur, Bresson, Xavier, and LeCun, Yann. Geometric deep learning - NIPS Tutorial, December 2017.

[6] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral Networks and Locally Connected Networks on Graphs. *arXiv:1312.6203 [cs]*, May 2014.

[7] Chaudhary, Amit. A Visual Survey of Data Augmentation in NLP, May 2020.

[8] Fan R. K. Chung. *Spectral graph theory*. Number no. 92 in Regional conference series in mathematics. Published for the Conference Board of the mathematical sciences by the American Mathematical Society, Providence, R.I, 1997.

[9] Luca Cosmo, Mikhail Panine, Arianna Rampini, Maks Ovsjanikov, Michael M. Bronstein, and Emanuele Rodolà. Isospectralization, or how to hear shape, style, and correspondence. *arXiv:1811.11465 [cs]*, April 2019.

[10] Miles D. Cranmer, Rui Xu, Peter Battaglia, and Shirley Ho. Learning Symbolic Physics with Graph Networks. *arXiv:1909.05862 [astro-ph, physics:physics, stat]*, November 2019.

[11] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. page 9.

[12] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional Networks on Graphs for Learning Molecular Fingerprints. *arXiv:1509.09292 [cs, stat]*, November 2015.

[13] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking Graph Neural Networks. *arXiv:2003.00982 [cs, stat]*, March 2020.

[14] Matthias Fey and Jan Eric Lenssen. Fast Graph Representation Learning with PyTorch Geometric. *arXiv:1903.02428 [cs, stat]*, April 2019.

[15] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Muller. SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 869–877, Salt Lake City, UT, June 2018. IEEE.

[16] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry. *arXiv:1704.01212 [cs]*, June 2017.

[17] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. *arXiv:1706.02216 [cs, stat]*, June 2017.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015.

[19] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv:2005.00687 [cs, stat]*, June 2020.

[20] Anees Kazi, Luca Cosmo, Nassir Navab, and Michael Bronstein. Differentiable Graph Module (DGM) Graph Convolutional Networks. *arXiv:2002.04999 [cs, stat]*, February 2020.

[21] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv:1609.02907 [cs, stat]*, February 2017.

[22] Boris Knyazev, Xiao Lin, Mohamed R. Amer, and Graham W. Taylor. Spectral Multigraph Networks for Discovering and Fusing Relationships in Molecules. *arXiv:1811.09595 [cs, stat]*, November 2018.

[23] Boris Knyazev, Xiao Lin, Mohamed R. Amer, and Graham W. Taylor. Image Classification with Hierarchical Multigraph Networks. *arXiv:1907.09000 [cs]*, July 2019.

[24] Boris Knyazev, Graham W. Taylor, and Mohamed R. Amer. Understanding Attention and Generalization in Graph Neural Networks. *arXiv:1905.02850 [cs, stat]*, October 2019.

[25] LeCun, Yann, Canziani, Alfredo, and Bresson, Xavier. Deep Learning with pytorch - lecture 13, 2020.

[26] Ron Levie, Michael M. Bronstein, and Gitta Kutyniok. Transferability of Spectral Graph Convolutional Neural Networks. *arXiv:1907.12972 [cs, stat]*, July 2019.

[27] Ron Levie, Elvin Isufi, and Gitta Kutyniok. On the Transferability of Spectral Graph Filters. *arXiv:1901.10524 [cs, stat]*, January 2019.

[28] Ron Levie, Federico Monti, Xavier Bresson, and Michael M. Bronstein. CayleyNets: Graph Convolutional Neural Networks with Complex Rational Spectral Filters. *arXiv:1705.07664 [cs]*, October 2018.

[29] Nikhil Mehta, Lawrence Carin, and Piyush Rai. Stochastic Blockmodels meet Graph Neural Networks. *arXiv:1905.05738 [cs, stat]*, May 2019.

[30] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M. Bronstein. Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5425–5434, Honolulu, HI, July 2017. IEEE.

[31] Federico Monti, Michael Bronstein, and Xavier Bresson. Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks. page 11.

[32] Federico Monti, Fabrizio Frasca, Davide Eynard, Damon Mannion, and Michael M. Bronstein. Fake News Detection on Social Media using Geometric Deep Learning. *arXiv:1902.06673 [cs, stat]*, February 2019.

[33] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. page 12.

[34] Arianna Rampini, Irene Tallini, Maks Ovsjanikov, Alex M. Bronstein, and Emanuele Rodolà. Correspondence-Free Region Localization for Partial Shape Similarity via Hamiltonian Spectrum Alignment. *arXiv:1906.06226 [cs]*, June 2019.

[35] F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner, and G. Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, January 2009.

[36] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling Relational Data with Graph Convolutional Networks. *arXiv:1703.06103 [cs, stat]*, October 2017.

[37] Petar Velicˇkovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. GRAPH ATTENTION NETWORKS. page 12, 2018.

[38] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin Lin, Junbo Zhao, Jinyang Li, Alexander Smola, and Zheng Zhang. Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs. *arXiv:1909.01315 [cs, stat]*, September 2019.

[39] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay Pande. MoleculeNet: A Benchmark for Molecular Machine Learning. *arXiv:1703.00564 [physics, stat]*, October 2018.

[40] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? *arXiv:1810.00826 [cs, stat]*, February 2019.

[41] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, July 2018.