Analyzing and improving graph neural networks

Guillaume Renton

Normandie Université

THÈSE

Pour obtenir le diplôme de doctorat

Spécialité Informatique

Préparée au sein de l'Université de Rouen Normandie

Analyzing and Improving Graph Neural Networks

Présentée et soutenue par

Guillaume RENTON

Thèse soutenue publiquement le 8 juillet 2021 devant le jury composé de		
Ramel	Jean-Yves	Rapporteur
Llados	Josep	Rapporteur
Fromont	Elisa	Membre du jury
Thome	Nicolas	Membre du jury
Brun	Luc	Membre du jury
Héroux	Pierre	Membre du jury
Gaüzère	Benoït	Membre du jury
Adam	Sébastien	Membre du jury

Thèse dirigée par Sébastien ADAM, laboratoire LITIS

UNIVERSITÉ DE ROUEN NORMANDIE

ED 590 MIIS
Mathématiques, Information, Ingénierie des Systèmes

litis

Remerciements

Tout d'abord, je tiens à remercier mon directeur de thèse, Sébastien Adam, ainsi que mes deux encadrants de thèse, Pierre Héroux et Benoît Gaüzère, pour l'opportunité qu'ils m'ont offert, leur encadrement ainsi que leur soutien et leur patience, notamment dans le contexte particulier de la dernière année de thèse.

Je tiens également à remercier mes deux rapporteurs, Jean-Yves Ramel et Josep Llados, pour avoir pris le temps de lire et d'emmetre des remarques pertinentes sur les travaux proposés dans ce manuscrit. Merci également à Elisa Fromont, Nicolas Thome et Luc Brun d'avoir accepté de prendre de leur temps et de faire partie de ce jury de thèse.

Je souhaite également remercier l'ensemble du LITIS, pour son accueil et sa bienveillance. Je remercie tout particulièrement Yann, Cyprien, Avelina, Andrès, Sen et Achille pour leur présence ainsi que pour les intenses parties de Tarot. Merci également à Maxime, pour les violentes parties de Bloodbowl. Merci également à Muhammet, avec qui cela a été un plaisir de travailler. Et puis globalement tous les membres de l'équipe Apprentissage, que j'ai pu cotoyer à la fois en tant qu'enseignants et en tant que collègues. Enfin, merci à Thierry, qui m'a offert une première opportunité lors de mon stage de Master.

Je remercie également toutes les personnes qui ont été présentes durant ces trois années. Mes parents tout d'abord, pour le soutien qu'ils m'ont accordé tout au long de mes études et de ma vie. Merci également à Alex et Marie pour leur dur labeur pendant mes pauses. Enfin, je tiens à remercier Jérémie, Annouck, Fred, Lucie, Clément, Mathile, Hugo, Simon, Baptiste, Florian, Benjamin, Seb et Dorian pour leur support, les différentes distractions nécéssaires qu'ils m'ont apportés et tout simplement leur présence à mes côtés.

Résumé

Bien que théorisés il y a une quinzaine d'années, l'intérêt de la communauté scientifique pour les réseaux de neurones sur graphes n'a connu un réel essor que très récemment. De tels modèles visent à transposer les capacités d'apprentissage de représentation inhérentes aux réseaux de neurones profonds sur des données de type graphes, via l'apprentissage d'états cachés associés aux nœuds du graphe. Ces états cachés sont calculés et mis à jour en fonction des informations contenues dans le voisinage de chacun des nœuds.

Ce récent intérêt pour les réseaux de neurones sur graphes (GNN) a conduit à une "jungle" de modèles et de méthodes, rendant le domaine de recherche parfois confus. Historiquement, deux principales stratégies ont été explorées : les réseaux spatiaux et les réseaux spectraux. Les réseaux spatiaux, parfois appelés "message passing neural network", sont basés sur le calcul d'un message agrégeant l'information contenue dans le voisinage de chacun des nœuds. Ce message est ensuite utilisé afin de mettre à jour les états cachés des différents nœuds du graphe. Les réseaux spectraux quant à eux sont basés sur la théorie spectrale des graphes et reposent donc sur le Laplacien du graphe. La décomposition en valeurs/vecteurs propres du Laplacien permet notamment de définir une transformée de Fourier sur graphe ainsi qu'une transformée inverse. À partir de ces transformées, différents filtrages peuvent être appliqués sur le graphe, obtenant des résultats similaires au filtrage sur une image ou sur un signal.

Dans cette thèse, nous commençons par introduire une troisième catégorie, appelée spectral-rooted spatial convolution. En effet, certaines méthodes récentes prennent racine dans le domaine spectral tout en évitant le calcul de la décomposition en vecteurs propres du Laplacien. Cette troisième catégorie nous amène à nous poser la question de la différence fondamentale entre les réseaux de neurones spatiaux et réseaux de neurones spectraux. Nous répondons à cette question par la proposition d'un modèle général unifiant les deux stratégies, montrant notamment que les modèles spectraux sont un cas particulier des modèles spatiaux. Ce cadre unifié nous a par ailleurs permis de proposer une analyse spectrale de plusieurs modèles de GNN populaires dans la communauté

scientifique, à savoir GCN, GIN, GAT, Chebnet et CayleyNet. Cette analyse montre que les modèles spatiaux sont limités aux filtrages passe-bas et passe-haut, tandis que les modèles spectraux sont capables de réaliser n'importe quel type de filtres. Ces résultats sont ensuite retrouvés avec la présentation d'un problème jouet, montrant dans un premier temps la limitation des modèles spatiaux à définir des filtres passe-bande et l'importance que peut revêtir l'utilisation de tels filtres.

Ces résultats nous ont amenés à proposer une méthode capable de réaliser n'importe quel type de filtrage, tout en limitant le nombre de paramètres du réseau. En effet, bien que les modèles spectraux soient capables de réaliser tout type de filtrage, l'ajout d'un nouveau filtre requiert l'ajout d'une nouvelle matrice de poids dans le réseau de neurones. Afin de réduire le nombre de paramètres, nous proposons l'adaptation des Depthwise Separable Convolution aux graphes via une méthode intitulée Depthwise Separable Graph Convolution Network. Cette méthode a été évaluée à la fois en apprentissage transductif et en apprentissage inductif, et obtient des résultats supérieurs à l'état de l'art sur les datasets testés.

Enfin, nous proposons une méthode définie dans le domaine spatial afin de prendre en compte les attributs d'arcs. En effet, cette problématique a été peu étudiée par la communauté scientifique, et le nombre de méthodes proposant d'inclure ces informations est très réduit. La nôtre, intitulée Edge Embedding Graph Neural Network, propose de projeter les attributs d'arcs dans un nouvel espace via un premier réseau de neurones, avant d'utiliser les caractéristiques extraites dans un GNN. Cette méthode est évaluée dans une problématique particulière de détection de symboles dans un graphe.

Abstract

Although theorised about fifteen years ago, the scientific community's interest for graph neural networks has only really taken off recently. Those models aim to transpose the representation learning capacity inherent in deep neural network onto graph data, via the learning of hidden states associated with the graph nodes. These hidden states are computed and updated according to the information contained in the neighborhoud of each node.

This recent interest for graph neural networks (GNNs) has led to a "jungle" of models and frameworks, making this field of research sometimes confusing. Historically, two main strategies have been explored : the spatial GNNs on one side and the spectral GNNs on the other side. Spatial GNNs, sometimes also called Message Passing Neural Network, are based on the computation of a message which agregates the information contained in the neighborhoud of each node. On the other side, spectral GNNs are based on the spectral graph theory and thus on the graph Laplacian. The eigendecomposition of the graph Laplacian allows to define a graph Fourier transform and its inverse. From these transforms, different filters can be applied on the graph, leading to similar result than filtering on images or signals.

In this thesis, we begin by introducing a third category, called spectral rooted spatial convolution. Indeed, some recent methods are taking root in the spectral domain while avoiding to compute the eigendecomposition of the graph Laplacian. This third category leads to question about the fundamental difference between spectral and spatial GNNs. We answer this question by proposing a general model unifying both strategies, showing notably that spectral GNNs are a particular case of spatial GNNs. This unified model also allowed us to propose a spectral analysis of some popular GNNs in the scientific communitic, namely GCN, GIN, GAT, ChebNet and CayleyNet. This analysis shows that spatial models are limited to low-pass and high-pass filtering, while spectral models can produce any kind of filters. Those results are then found with the presentation of a toy problem, showing in the first instance the limitation of spatial models to define pass-band filters, and the importance of designing such filters.

Those results have led us to propose a method allowing any kind of filter, while limiting the network's number of parameters. Indeed, even though spectral models are able to design any kind of filtering, each new filter require the add of a new weight matrix in the neural network. In order to reduce the number of parameters, we propose to adapt Depthwise Separable Convolution to graphs through a method called Depthwise Separable Graph Convolution Network. This method is evaluated on both transductive and inductive learning, outperforming state-of-the-arts results.

Finally, we propose a method defined in the spatial domain in order to take into account edge attributes. Indeed, this issue has been little studied by the scientific community, and the number of methods allowing to include edge attributes is very small. Our proposal, called Edge Embedding Graph Neural Network, consists in embedding edge attributes into a new space through a first neural network, before using the extracted features in a GNN. This method is evaluated on a particular problem of symbol detection in a graph.

Contents

Chapter 1

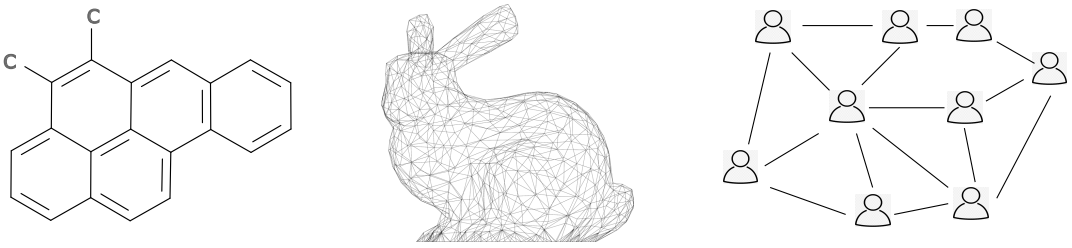
Introduction

1.1 Context of the thesis

Working with data containing structural information has always been a complex task. Yet, structural information exists in many cases, from microscopic to macroscopic through more abstract cases. It can be the structure of a molecule which expresses how different atoms are bonded together to form the molecule. But it can also represent a planetary system or an electricity network. In fact, one could argue that most of the concepts that human can think of can be described as a structure, or at least own a structural information. This document for example, has a structure. It is made of different chapters, each of them divided into multiple sections. The set of chapters, sections and their hierarchy forms the logical structure of the document. This structure can then be presented as a table of contents.

The structural and relational information also directs our thoughts. We don't simply think about a person, an object, an action or a feeling, we associate them through relational structures to make a more complex concept. We don't simply think about an image containing a cat and a mug, we think about a cat pushing a mug to make it fall.

In order to be used in computer science, those data need to be encoded. The most common way to represent data is to use matrices or vectors. Those encoded data can implicitly include structural information. For example, when an image is encoded as a 2 or 3-dimension matrix, this matrix includes a proximity relation between 2 neighboring pixels. For a signal encoded as a vector, the temporal relation between two moments is encoded by their proximity in the vector. Both matrices and vectors allow to encode multiple kinds of structural relations. However, those structural relations are never



Molecules *Meshes* *Social networks*

Figure 1.1: Graph structured data

explicitly encoded, which forbids more complex structure. One representation way that can be used is through graphs.

A graph is a data structure that represents a set of objects called *nodes*, where each object can be related to other ones from the set through a relational information called an *edge*. Each node and each edge is an object, and can thus carry information. Figure 1.1 presents few data represented as graphs.

This thesis dives into this fascinating area of analyzing data represented as graphs. More particularly, it concerns machine learning on graphs. Machine learning is one of the most studied field of Artificial Intelligence (AI). It consists in designing algorithms that aim to *learn* from data. The way data are designed is thus especially important for Machine Learning algorithms.

The interest for Machine Learning has recently exploded with the emergence of Deep Learning algorithms. This emergence has been allowed thanks to the improved computing capacity and the availability of huge datasets. Unlike most of Machine Learning algorithms, Deep Learning aims at learning new representations of information. Those new representations are obtained through a succession of non-linear functions and can be used to predict some properties. Deep learning architectures are based on neural network models and some notable ones will be presented in section 2.4.

Unfortunately, most of machine learning algorithms have been designed to work with vectors or matrices, due to their rich mathematical properties, especially the fact that they evolve in a Euclidean space. Less machine learning models have been dedicated to graphs, and it has also been the case with Deep Learning. Proposing efficient Deep Learning algorithms dedicated to graphs can thus be considered as an important step to improve AI performance and possibilities.

In this thesis, we consider this design of machine learning algorithms which can be applied on structured data represented as graphs. In this context, *Graph Neural Networks* (GNNs) [28, 72] have recently been proposed. GNNs are

models of deep learning that are able to capture the structural information contained in graphs. In this thesis, we focus on the analysis of Graph Neural Networks and we propose some contributions to develop this concept.

1.2 Research questions and main contributions

The contributions of this thesis are guided by the following research questions:

Research question 1: *In the last years, Graph Neural Networks have became one of the hottest topics in machine learning. This has led to a "jungle" of models and frameworks. Our first question is thus: how to classify existing approaches ?*

Our first contribution consists in deepening the classical GNN taxonomy which classifies GNNs into two different categories: spectral and spatial GNNs. Since some models are not easily classified following this taxonomy, we propose to add a third category which contains the models based on spectral-rooted convolution.

Research question 2: *Given this taxonomy, can we merge all these approaches into one single framework ?*

As a second contribution of this thesis, we provide a proof that spectral GNNs are a special case of spatial GNNs. This result allows to present a general framework that bridges the gap between spectral and spatial GNN.

Research question 3: *Since all existing models can be merged into a single framework, one can wonder what is the spectral behavior of models initially defined as spatial ones compared to models defined as spectral ?*

The third contribution of this manuscript consists in a spectral comparative analysis of some notable GNNs, namely: ChebNet, CayleyNet, GCN, GIN and GAT. This analysis highlights the fact that most of popular models and particularly spatially defined models are limited to low-pass filtering.

Research question 4: *Since most of existing models are limited to low-pass filtering while obtaining good results on reference datasets, one can wonder if low-pass filters are sufficient for all problems ?*

In our fourth contribution, two toy problems are designed and used to show that some problems may require band-pass filtering. Those image-based toy

problems have been handcrafted in order to require band-pass filters. Results confirm the limits of existing models on those datasets.

Research question 5: *Following previous results, most of GNNs appear to be limited to low-pass filtering. Models that are able to propose band-pass filtering require a huge number of parameters. Are GNNs able to produce band-pass filtering at low-cost ?*

As our fifth contribution, a new model allowing band-pass filtering at low parameter cost is presented. This method adapts a popular CNN method named Depthwise Separable Convolution to graphs.

Research question 6: *One of the strength of graphs is the versatility of their attributes. Both nodes and edges can carry information. Yet, most of GNN models only use node attributes. Are GNN limited to use the information carried by node attributes ?*

Our final contribution proposes a general model that is able to use edge attributes. This model is then evaluated on a problem of symbol detection in floorplan images.

1.3 Outline of this thesis

This manuscript is divided into 7 chapters. After this general introduction, chapter 2 presents a set of notations and background that are going to be used all along this manuscript. Background notably includes general graph notations and definitions, a presentation of the different kinds of tasks that can occur in machine learning on graphs, and a general introduction about Deep Learning.

Chapter 3 presents a general introduction to Graph Neural Networks. GNNs are a very recent and active field of research. But this activity has led to many different definitions of GNNs, leading to a confusing field. After a presentation of the first use of the "GNN" term, the two different main theories that has led to the actual GNNs are explained. By answering research question 1, a third category called spectral rooted spatial convolution is added. This chapter concludes by presenting the comparison with the Weisfeiler-Lehman isomorphism test, which is commonly used as an evaluation tool of GNNs.

In chapter 4 the dichotomy made between the two approaches that led to GNNs, namely spectral and spatial GNNs is questioned. A proof that spectral GNNs are a special case of spatial GNNs is firstly given. This allows to present a general model that bridges the gap between spectral and spatial GNN and to

thus answer research question 2. Using this result, research question 3 is tackled through a spectral analysis of most popular GNNs that shows that most GNN models are limited to low-pass filtering. This chapter is concluded by proposing an answer to research question 4, highlighting the need of band-pass filtering.

Using conclusions from chapter 3 as well as the general model proposed in chapter 4, a new model allowing band-pass filtering at low parameter cost is presented in chapter 5 to answer research question 5. This method is based on a popular CNN method named Depthwise Separable Convolution, and proposes its adaptation to graphs.

Using both conclusions from chapter 3 and the general model presented in chapter 4, chapter 6 proposes to answer to research question 6 through a general model that is able to use edge attributes. This model is then evaluated on a problem of symbol detection in floorplan images.

Chapter 7 concludes this manuscript and attempts to synthesize the research questions previously asked.

Finally, in appendix A, our first work on deep learning and graph is presented. This work is mainly presented to show how traditional deep learning algorithms such as CNN are limited to work with graphs.

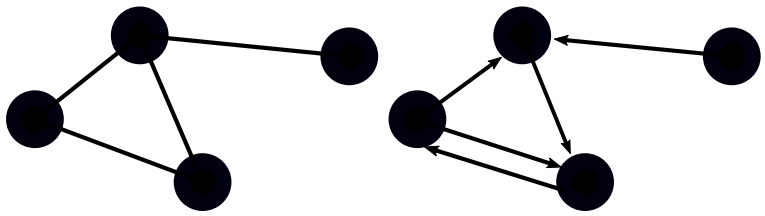
Chapter 2

Background

In this chapter, we briefly introduce several background topics and notations. More background will be added over the document when required. In 2.1, we present definitions and notations related to graphs. Section 2.2 tries to quickly present how machine learning is used with graphs. In section 2.3, we present a set of applications for graphs that can be solved with machine learning.

2.1 Graph definitions and notations

A graph is a pair $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of nodes or vertices of G and $\mathcal{E} \in \mathcal{V} \times \mathcal{V}$ is the set of edges linking those nodes. An edge e is called incident to a node v if there is a node u such as $e = (u, v)$ or $e = (v, u)$. A node v is adjacent to a node u if there is an edge $e = (u, v)$ or $e = (v, u) \in \mathcal{E}$. The set of nodes adjacent to a node v is called the neighborhood of v and is written $\mathcal{N}(v)$.



(a) An undirected graph (b) A directed graph

Figure 2.1: Undirected vs directed graph.

Edges in a graph can have a direction associated or not. In the undirected case, we have $e = (v, u) \Leftrightarrow e = (u, v)$ while in the directed case $e = (v, u)$ is

the edge from v to u, and is different from $e = (u, v)$. Such a graph is called a directed graph. In the case of directed graphs, the neighborhood definition can be extended to oriented neighborhood.

A graph is said to be simple if it has no loop. A loop is an edge linking a node to himself, e.g. $e = (v, v)$.

In this document, graphs are considered as simple and undirected.

In a graph, both nodes and edges can be attributed to carry information. For example, considering molecules, each node representing an atom can be attributed by the name of the chemical element, and each edge can be attributed by its type of bond. In such a case, the graph definition can be extended with two functions $\mu\colon \mathcal{V} \to \mathcal{L}_\mathcal{V}$ and $\xi\colon \mathcal{E} \to \mathcal{L}_\mathcal{E}$, where $\mathcal{L}_\mathcal{V}$ and $\mathcal{L}_\mathcal{E}$ are two sets of attributes. μ (resp. ξ) is a function that associates one attribute from $\mathcal{L}_\mathcal{V}$ (resp. $\mathcal{L}_\mathcal{E}$) to a node (resp. an edge). The graph definition thus becomes $G = (\mathcal{V}, \mathcal{E}, \mu, \xi)$.

In this document, we work with both attributed and non attributed graphs.

When working with non attributed graphs, one way to represent the graph is to use the adjacency matrix $\mathbf{A} \in \mathbb{R}^{|V|\times|V|}$. In this matrix, nodes are ordered and each row and column correspond to a node. This matrix is build so that, if i corresponds to node u, j corresponds to node v, then $\mathbf{A}_{i,j} = 1$ if $(u, v) \in \mathcal{E}$ and 0 otherwise. When working with attributed graphs, this representation has to be extended. In order to integrate edge attributes, the adjacency matrix can be extended to $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|\times\dim(\mathcal{L}_\mathcal{E})}$. A matrix $\mathbf{H} \in \mathbb{R}^{|\mathcal{V}|\times\dim(\mathcal{L}_\mathcal{V})}$ can be added to carry node attributes.

A set is basically an unordered data structure. When a graph is encoded with matrices or tensors (adjacency matrix/tensor, node labeling matrix), it implicitly defines an empirical ordering of the node set by assigning each node to a matrix row or column. However, any other ordering would result in different but valid encoding matrices.

Thus, any function applied over the graph should produce the same output, independently of the arbitrary selected ordering. Specifically, it means that for any permutation matrix $\mathbf{P}$, we want permutation invariance $f(\mathbf{PAP}^T) = f(\mathbf{A})$. When applying a function to nodes, we should have a consistent output for each node, independently of its ordering. In this case, we seek for a permutation equivariant function $f : f(\mathbf{PAP}^T) = \mathbf{P}f(\mathbf{A})$. Intuitively, the permutation applied to the node is applied to the output of the function. This is an important point when designing algorithms for graphs, including machine learning models.

2.2 Machine Learning and Graphs

Artificial intelligence is a field of research that seeks to produce models that perform tasks that would normally require human intelligence. Machine learning is one of the most studied branch of artificial intelligence. It is based on mathematical and statistical approaches to compute models that learn from data.

There are basically 3 types of machine learning frameworks: supervised learning, semi or self supervised learning and unsupervised learning. Supervised learning refers to an algorithm which makes use of labeled data unlike unsupervised learning where each data is not labeled. Semi or self supervised algorithms are the in-between: a generally small part of data are labeled, while the rest is not. In this document, we mainly focus on supervised learning.

Supervised learning is commonly used in two steps, the training and test step. The training step consists in computing a model using statistical or mathematical approaches. The model is fit over a subset of available data, called training set. Once the model is trained, the test step consists in evaluating the computed model over the other subset of data, called test set. A third set, called validation set, is often used to tune models that require hyperparameters or to avoid overfitting. Hyperparameters are parameters that can not be learned during fitting, such as the number of layers in neural networks for example. Overfitting happens when a model stops generalizing and starts to fit too close to the training data.

Historically, most of machine learning models work with handcrafted feature extracted from raw data. Those features were generally defined by an expert of the application domain. Some models also use feature selection to select most interesting features over the data.

This feature extraction process and the mathematical models used have constrained the direct use of machine learning over vectors and matrices for a long time. The graph structure made of variable number of nodes and edges can not be easily taken into account by traditional machine learning models.

In order to exploit the representative power of graphs, the machine learning community has developed different tools. One of them is the graph edit distance (GED) whose definition will be deepen in appendix A. Basically, GED is a distance between two graphs, quantifying the amount of edit operations such as removing, deleting and substituting nodes and edges required to transform one graph into another. This distance can then be used by traditional distance based machine learning algorithms to compute a model.

Another developed tool is graph embedding. Graph embedding aims at casting graphs with different sizes (variable number of nodes and edges) into

fixed size single vectors. It can be seen as some kind of feature extraction on the graph object. Once these features are extracted, usual machine learning methods can be applied. While opening the use of graphs with vector based machine learning algorithms, the embedding operation induces a loss of information which may be hard to control.

To limit this loss information, graph kernels [11] have been proposed since a couple of decades. The strategy here is to apply the kernel trick to the graph space, and thus allow the use of kernel based methods such as SVM to graphs. The kernel trick [4] consists in replacing the access to the data trough scalar products by the output of a similarity measure function $k : \mathcal{G}^2 \to \mathbb{R}$ which is semi definite positive [57]. Given this property, one can show that $k(G, G')$ corresponds to a scalar product in the space $\mathcal{H}$ associated to the kernel :

$$k(G, G') = \langle \Phi(G), \Phi(G') \rangle_{\mathcal{H}}$$

, with $\Phi : \mathcal{G} \to \mathcal{H}$. Note that Φ may be not known explicitly, hence allowing infinite dimension for $\mathcal{H}$. Alleviating the need of an explicit embedding of graphs into a finite space while keeping the connection to high level machine learning algorithms put graph kernel methods to the state of the art during 2010's. However, the definition of k must be done a priori and is application dependant. In addition, its computation is generally high, hence limiting the scalability of this approach.

As we can see, most of approaches of machine learning over graphs consist in transforming data to obtain a suitable data design, in order to subsequently apply machine learning methods over those data. But those transformations can not be achieved without a loss of information, especially on the structural information.

2.3 Different kind of tasks

The representation flexibility of graphs makes this data structure suited for the representation of a huge number of data from different kinds. Thus, they can be used for different tasks.

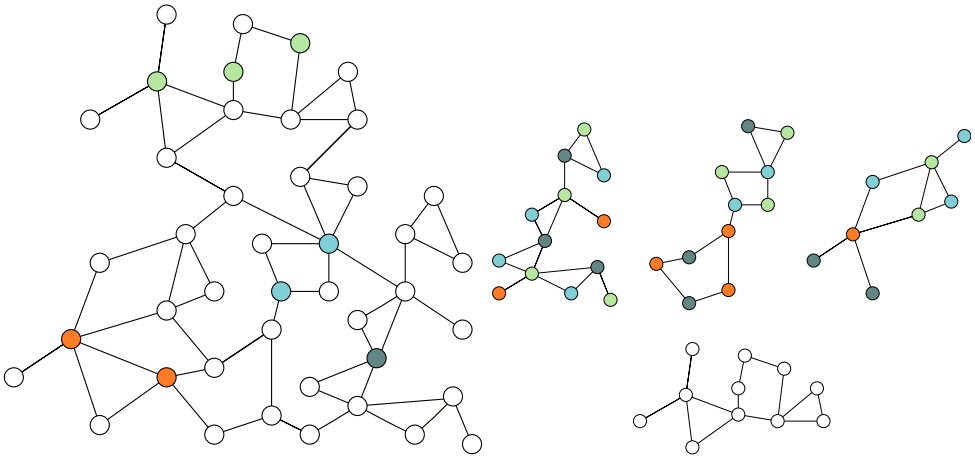
The first task is node classification or regression. In this task, nodes are seen as data themselves, and edges are additional information representing links between those data. To this extent, edges are not mandatory to apprehend the task. This is the case for example for citation datasets (Cora [40], Citeseer [26] or Pubmed [1]). In those datasets, an article is represented as a node, and articles are linked together with edges if they cite or are cited by each other. From those citation graphs, we want to predict one or multiple categories the

article belongs to. The category prediction can be taken by only examining the intrinsic information given by article information. However, taking into account that an article cites or is cited by another brings interesting information that can be used to the prediction.

Two cases can be distinguished for node classification tasks. In a first case, the training dataset is a set of graphs where the class of each node is known, and we want to induce rules to predict the class of nodes in other graphs (test dataset). This first case is called inductive learning and is similar to what is done with supervised learning in Euclidean space. Examples of inductive learning datasets are Reddit and protein-protein interaction [94].

In the second case, we only have one graph or a set of graphs, where only the class of a few nodes is known. Since we can't split the set into a training and a test set, we want to use the features of the known nodes to propagate information to predict the class of nodes whose class is previously unknown. Classification problem associated to Cora, Citeseer and Pubmed datasets are examples of transductive task. It may be important to notice that transductive learning is data specific, meaning that simply adding a single test node may require a complete new training from scratch.

Neither inductive nor transductive learning is specific to graphs or nodes classification, but if inductive learning is the most common in Euclidean spaces, graph structure compels some problems to transductive learning. Figure 2.2 presents the difference between transductive and inductive learning.



(a) Transductive learning (b) Inductive learning

Figure 2.2: Transductive versus inductive learning. Colored nodes are nodes whose class is known while white nodes are nodes whose class has to be predicted.

A second task is graph classification or regression. Unlike node classification, it is the couple of set of nodes and set of edges forming the graph which is the data as a whole and both are inseparable. For example, atoms in a molecule are as important than the bonds between those atoms. Molecules datasets are thus common examples of graph prediction tasks, such as QM9 [27], TOX21 or HIV [85, 5].

Even though those two first tasks are the most common, other tasks exist, such as graph generation, link prediction, or community detection. For example, a task can consist in the creation of a graph representing a set of short sentences, such as the tasks associated to the bAbI[84] dataset. Figure 2.3 shows different examples of tasks with graphs.

Thus, graphs can bring many information and can be the subject of many kind of different tasks. But as we saw previously, working with graphs and machine learning is a tough task.

2.4 Deep Neural Networks

Deep Neural Network can be seen as the succession of multiple neural network layers, where each layer is generally the composition of a parameterized function with a non-linearity function.

Unlike other machine learning methods, Deep Learning computes new representations, which are then used to predict properties of the object.

One of the first definition of a neural network layer is the multi-layer perceptron (MLP) [70]. This model is defined following 2.1.

$$f(\mathbf{h}) = \sigma(\mathbf{h}\mathbf{W} + \mathbf{b}) \tag{2.1}$$

where $\mathbf{h} \in \mathbb{R}^{1\times f_l}$ is a line feature vector of size f_l, $\mathbf{W} \in \mathbb{R}^{f_l \times f_{l+1}}$ is the matrix of parameters (also called the weight matrix), and $\mathbf{b} \in \mathbb{R}^{1\times f_{l+1}}$ is the bias vector. σ is a non linear function and was originally defined as the sigmoid function $S(x) = \frac{1}{1+\exp^{-x}}$, but the $\texttt{ReLU}(x) = \texttt{max}(0, x)$ is commonly preferred. Other non linear functions can be found, such as the `tanh`, the `eLU` or the `LeakyReLU`.

Multiple neural network models have since been proposed. Most notable ones are the Convolutional Neural Network (CNN) and the Recurrent Neural Network (RNN). Each of them is dedicated to particular cases.

On one side, RNNs are defined to work with sequences, such as speech or handwriting. RNNs aim to compute new representations, called hidden states, for each moment of the sequence. Each moment hidden state is computed consecutively by taking into account the hidden states of the previous moment.

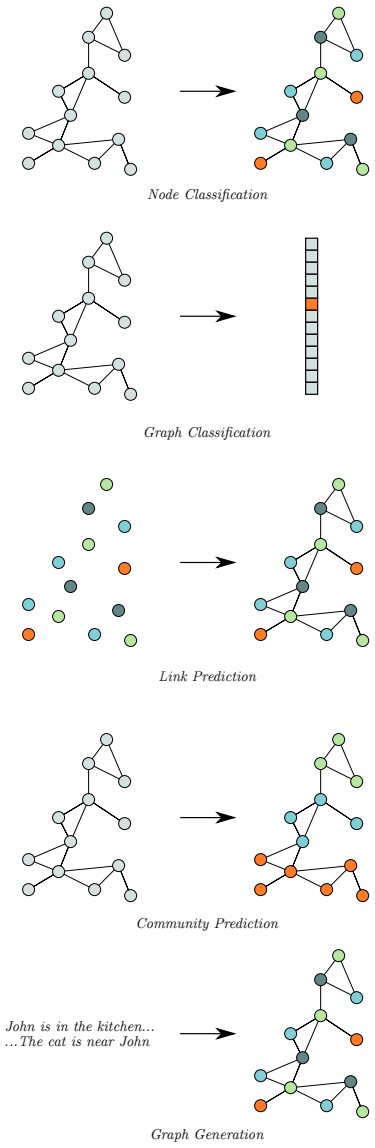


Figure 2.3: Tasks examples with graphs. One can note that node classification and community detection are close, but in practice, node classification refers to supervised learning while community detection refers to unsupervised learning.

Some models, called Bi-directional, also compute hidden states from the following moment.

On the other side, CNNs are defined in order to work with matrices encoding images. CNNs are interesting due to their proximity with Graph Neural Networks. It thus might be interesting to quickly presents them.

A layer of CNN is constituted of a fixed number of convolution filters, where each filter is defined by a fixed size set of learnable weights. Each filter is then applied equivalently on each pixel of the image, computing a new representation for each of those pixels.

One important point with CNN is the context used to compute the new pixel representation. This context is firstly defined by the shape of the filter. A large filter will take more distant context than a tight one.

Another way to take into account a large context is through an accumulation of CNN layers. Indeed, the new pixel representation computed by a CNN depends on the previous representation of its neighborhood, which has itself been obtained depending on a neighborhood. The neighborhood of the neighborhood is thus indirectly used to update each pixel by the second CNN layer. Increasing the number of CNN layers indirectly increases the neighborhood used to update each pixel. The set of neighbors used to update a pixel is called its receptive field.

A third way to increase the shape of receptive fields is through pooling. Pooling consists in the agglomeration of local information represented by a small set of pixels into a single one, through simple operations, such as `max` or `min`.

Deep Learning has been largely used and studied due to its impressive performances on images or sequences encoded as matrices. This success is partially based on the fact that these matrices are designed using a known and invariant topological relationships between its elements, which induces an implicit order on these elements and fixed topological relationships for each pair of element. For instance, a pixel can be defined on the top over a second one without ambiguity. However, graphs does not include such topological information since no order on neighborhood of a given node can be defined in general. Therefore, we can not directly apply CNN and RNN to graphs since they rely on fixed topological relationships. Being able to transpose CNN and/or RNN schemes to graphs will combine the representation learning ability of Deep Learning with the structural information endowed by graphs. This combination may lead to a significant improvement of machine learning in general.

Chapter 3

What is a Graph Neural Network ?

3.1 Introduction

As stated in the introduction chapter, most of machine learning algorithms are designed to work on Euclidean data and are not directly applicable in the graph space because of the lack or regular grid. This statement is particularly true for Deep Learning methods which strongly rely on linear algebra. Yet, Deep Learning has had, over the past decade, a strong impact in various machine learning applications relying on sequences or images such as scene recognition [45] and speech analysis [29].

This explains the recent challenge tackled by the machine learning community which consists in extending the Deep Learning paradigm into the world of graphs. The objective is to revisit Neural Networks to operate on graph data, in order to mix the benefits of representation power of graphs and representation learning ability offered by deep models across layers (see 2.4).

In this context, a large number of Graph Neural Networks (GNNs) have been recently proposed in the literature [72, 27, 14, 13, 30, 15, 79, 23, 87, 43, 51]. GNNs are Neural Networks that rely on the computation of successive hidden representations of nodes using both structure and attributes information carried by the whole graph.

In contrast to conventional Neural Networks, where the architecture of the network is related to the known and invariant topology of the data (e.g. MLPs for feature vectors, RNNs for data sequences, CNNs for images, see Figure

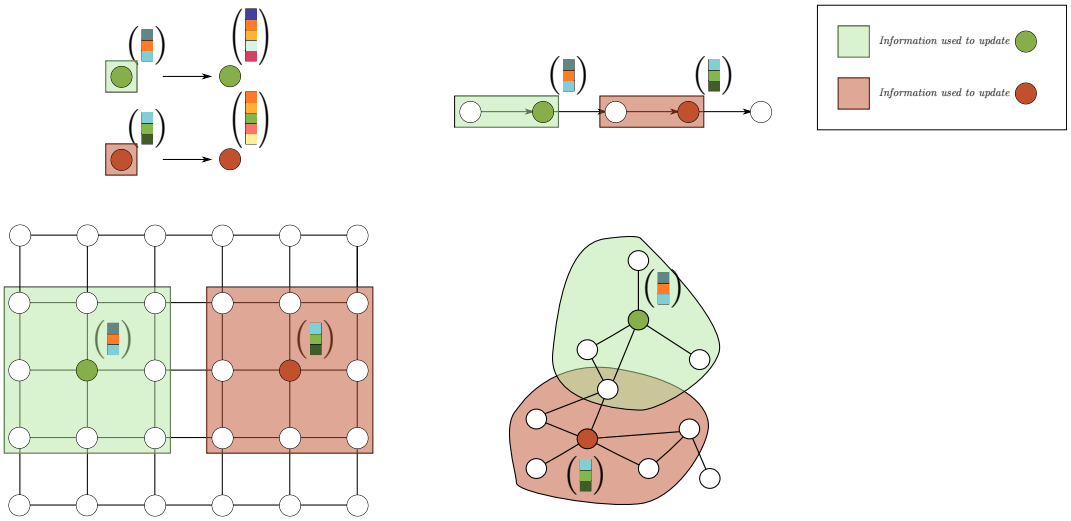


Figure 3.1: Figure showing the topology used by (upper left) a MLP, (upper right) a RNN, (down left) a CNN and (down right) a GNN. Topology is known and invariant for MLP, RNN and CNN, but not for GNN.

3.1), the node states[1] of GNNs must be propagated according to the graph structural information which is *a priori* unknown.

Figure 3.2 presents a typical GNN architecture. The figure highlights that in our point of view, a GNN layer "only" produces a new feature vector for each node. Tasks such as node classification can thus be naturally tackled through a particular GNN decision layer. In the case of tasks such as graph classification, a function that produces a fixed size embedding of the graph is required after GNN layers, before a decision layer applied on this embedding. This function is generally called the `readout` function, and must be permutation invariant (see section 2.1), as well to the number of nodes considered. Typical readout function are the `sum` or the `mean` function, but other solutions exist, such as pooling for example. The `readout` function is out of the scope of this chapter which focuses on GNNs.

In this chapter, our objective is to review the GNN literature. The study of GNN is a very recent but extremely active field of research. Hence, by the time you read this manuscript, new papers concerning GNNs will probably have been uploaded on arXiv[2]. This is why proposing an exhaustive and up

[1]It might be important to note the difference between features and states. Features are properties of nodes while states are a set of value computed by GNNs across layers. However, node states are generally initialized with node features, which might lead to a confusion. In this manuscript, both terms are used similarly.

[2]One hundred submissions at ICLR'21 concern GNNs

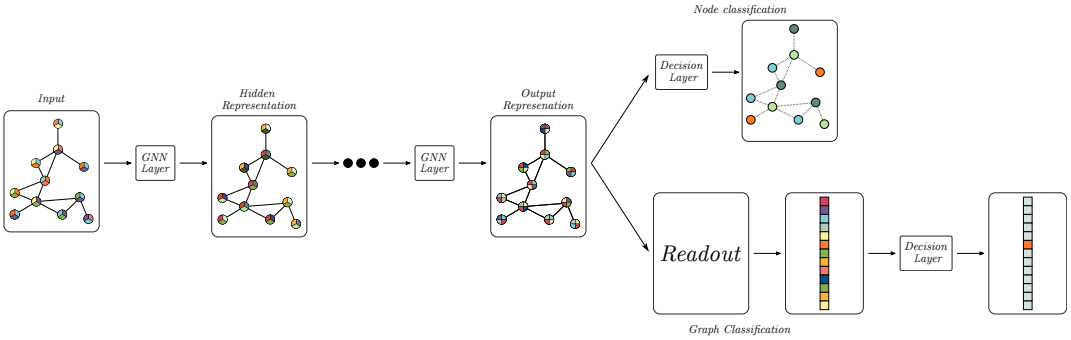


Figure 3.2: Example of GNN architecture for node or graph classification.

to date state of the art of the field is difficult, even if some surveys have already been proposed [93, 47, 86, 92]. In this chapter, rather than being exhaustive, we present the main families of approaches used in reference GNNs. The presentation relies on the classical dichotomy which distinguishes *spatial GNNs* and *spectral GNNs* [86]. Beyond, we propose for this review a third category called *Spectral-Rooted Spatial Convolutions* which gathers recent and efficient methods that take their foundations in the spectral domain, but are applied in the spatial one, without computing the graph Fourier transform. This review is the first contribution of this manuscript. Finally, in section 3.7, we discuss the relation between GNNs and the Weisfeiler-Lehman Isomorphism test (WL test). Indeed, the comparison with the WL test is a commonly used method to evaluate the expressive power of the different GNNs.

3.2 Review of reference GNNs

As shown in figure 3.2, a GNN can be defined as a neural network that computes node embeddings through hidden representations obtained by gathering information from node features, edge features and the graph structure.

As said before, a huge number of contributions have been proposed in this field during the very recent years. Most of them concern new GNN models but one can also find some contributions which describe generalization frameworks (e.g. GN, MPNN, NLNN and MoNet).

GNNs are generally classified into two categories in the literature. The first one concerns methods that rely on the spectral graph theory and that are designed or analyzed in a spectral way. They are called Spectral ConvGNN, Spectral CNN or Graph Convolutional Neural Networks (GCNN) in the literature. Methods such as [15, 43, 51, 53, 23] belong to this category. The second category concerns methods that are designed in a spatial way, trying

to mimic Convolutional Neural Network (CNN) applied on images. Those methods are called Spatial ConvGNN or Spatial GNN in the literature.

Concerning general frameworks, Message Passing Neural Networks (MPNN) [27] is one of the first attempt to propose a general model. Their model notably generalizes [54, 43, 9, 15], and [87, 30, 63] could naturally fit in this model. On the other side [6, 79, 43] are generalized by Mixture Model Network (MoNet) [60] model. Non-Local Neural Network [82] is another general model which is not dedicated to the graph space, but unifies attention models, as GAT. Finally, in [10], the authors propose to generalize MPNN and NLNN into a Graph Network (GN) model.

Figure 3.3 illustrates together the main state of the art GNNs and these frameworks. It shows that the distinction between spectral and spatial approaches is not always clearly defined since some spectral models fit into spatial frameworks. As an example, GCN which is defined by their authors as a spectral model, fits into the MPNN framework, which is a spatial framework. That is why we propose for our review a third category called "spectral-rooted spatial GNN".

In the following, before describing the principles of each category, we first focus on the first Graph Neural Network model proposed in [28] and extended in [72] since it is historically important. Then, we successively describe reference models that operate (i) in the spatial way, (ii) in the spectral way and (iii) at the intersection of both categories.

3.3 The "pioneer" Graph Neural Network Model

Graph Neural Networks (GNN) have been firstly theorized by [28] and then extended by [72]. In those papers, the authors propose to update a hidden representation of each node by aggregating information contained in its own labeling, its neighborhood labeling and the labeling of its incident edges. This defines Graph Neural Networks. More specifically, each node in the graph is updated according to :

$$\mathbf{h}_v^{(l+1)} = f(l_v, l_{\mathcal{N}(v)}, \mathbf{h}_v^{(l)}, e_{\mathcal{N}(v)}), \tag{3.1}$$

where $\mathbf{h}_v^{(l)}$ is the hidden state of the node v at iteration l, l_v encodes the attributes of node v, $\mathcal{N}(v)$ corresponds to the neighborhood of v, and thus $l_{\mathcal{N}(v)}$ and $e_{\mathcal{N}(v)}$ encode respectively the sets of nodes and edges attributes of v's neighborhood. Finally, a decision $\boldsymbol{o}_v$ (classification or regression) is taken for each node v as follows:

$$\mathbf{o}_v = g(\mathbf{h}_v^{\text{final}}, l_v), \tag{3.2}$$

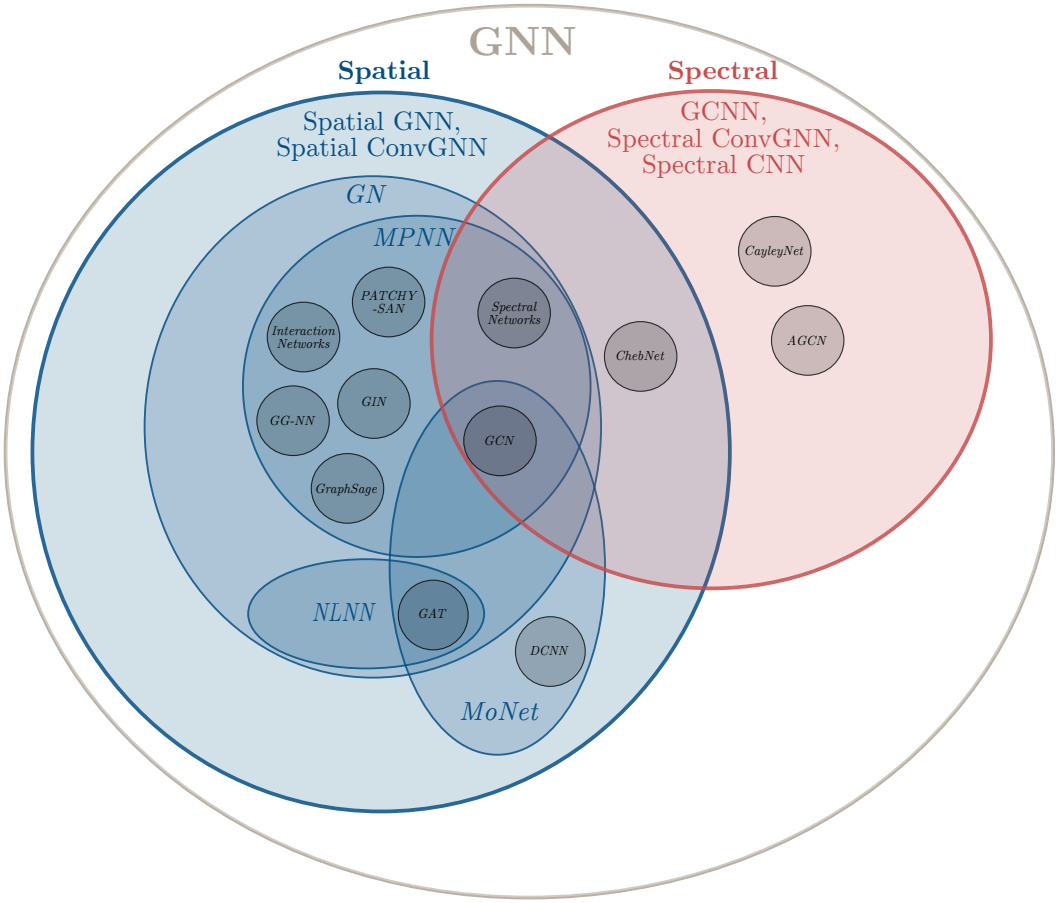


Figure 3.3: Overview of some state of the art GNNs and general models that have been proposed. In black are methods from the literature. In blue are general frameworks that operate in a spatial way. Spectral-rooted spatial methods are those that are classified as both spatial and spectral on this figure.

where g is an arbitrary function of the final node state $\boldsymbol{h}_v^{\text{final}}$ and the original node attribute l_v. Figure 3.4 shows an example of how a node is updated depending on its neighborhood. In GNNs, both functions f and g are generally defined as perceptrons [70].

This GNN in [28] and [72] was designed as a Recurrent Neural Network learned with the backpropagation through time algorithm [59]. The convergence is ensured by constraining f to be a contraction mapping. This first definition has built the foundations of GNNs, especially for the ones in the spatial domain. In the next subsection, we propose a review of this family denoted spatial GNN.

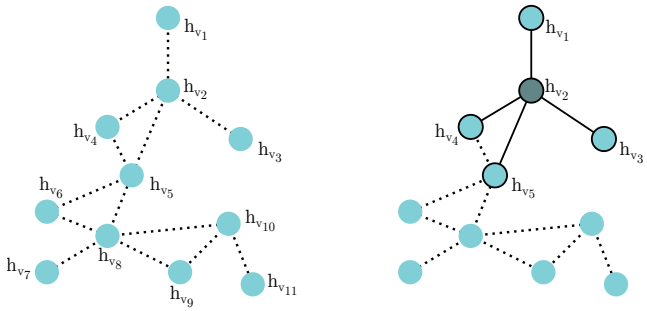


Figure 3.4: Example of how the hidden state of a node is updated depending on its neighborhood. On the left is the original graph. On the right, the hidden state of node v_2 is updated depending on its neighborhood v_1, v_3, v_4, v_5, and the edges between v_2 and its neighbor (in bold).

3.4 Spatial GNNs or Message Passing Neural Networks

Even though we will see in section 4.2 that spatial and spectral methods are closely related, we believe that Message Passing Neural Network (MPNN) [27] is the framework that best describes what is a Graph Neural Network.

An MPNN operates in the spatial domain and can be defined as a two step method. If we focus on one node in particular, denoted as central node, MPNN starts by computing a message corresponding to an aggregation of the central node's neighborhood states. The central node is then updated depending on this computed message and his own state. Formally, if we assume that `aggregate` and `update` are two functions (generally neural networks), and $\mathbf{h}_u^{(l)}$ the state of node u at layer l, then we can define an MPNN as equation 3.3.

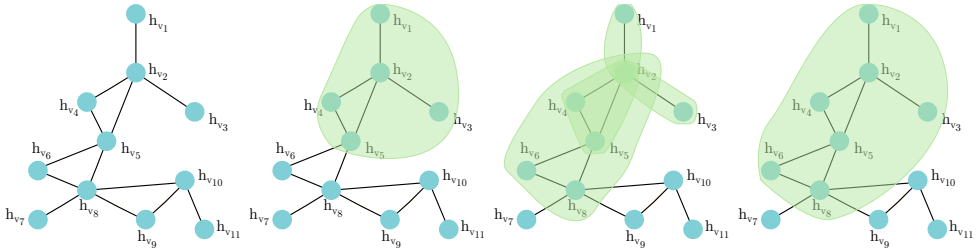
$$\mathbf{h}_u^{(l+1)} = \texttt{update}(\mathbf{h}_u^{(l)}, \texttt{aggregate}(\{\mathbf{h}_v^{(l)}, \forall v \in \mathcal{N}(u)\})) \tag{3.3}$$

Equation 3.3 defines a general way to compute a MPNN. One can notice that this equation does not take into account possible features on edges so the definition is extended by equation 3.4, where $\mathbf{h}_{e_{uv}}$ is the feature vector of edge e_{uv}.

$$\mathbf{h}_u^{(l+1)} = \texttt{update}(\mathbf{h}_u^{(l)}, \texttt{aggregate}(\{\mathbf{h}_v^{(l)}, \mathbf{h}_{e_{uv}}, \forall v \in \mathcal{N}(u)\})) \tag{3.4}$$

A single computation of equation 3.3 or 3.4 only takes into account the direct neighborhood of each node. By iterating this process, a larger structural

information can be included. This is illustrated in figure 3.5. Each iteration can be done either by the same `aggregate` and `update` functions, or different ones. In analogy to deep neural network, each iteration is often referred as a layer while the structural information used to update a node can be seen as the receptive field.



(a) (b) (c) (d)

Figure 3.5: Structural information used depending on the number of layers used. (a) an initial graph, (b) structural information used to update $\mathbf{h}_{v_2}$ with one layer. (c) structural information used to update each of the neighbor of v_2. (d) structural information used to update v_2 with two layers.

In this general definition of MPNN, `aggregate` and `update` functions have to be defined. The simplest way is to define the `aggregate` function as the sum function, and the `update` function as a neural network (Eq. 3.5[3]).

$$\mathbf{h}_u^{(l+1)} = \sigma((\mathbf{h}_u^{(l)} + \sum_{\forall v \in \mathcal{N}(u)} \mathbf{h}_v^{(l)})\mathbf{W}^{(l)} + \mathbf{b}) \tag{3.5}$$

The benefit of this simple definition is the way it can be easily implemented by matrix multiplication. By defining $\mathbf{C} = \mathbf{A} + \mathbf{I}$, where $\mathbf{A}$ is the adjacency matrix and $\mathbf{I}$ is the identity matrix, equation 3.5 can be written as equation 3.6. By adding the identity matrix to the adjacency, a virtual loop on each node is added. The same weights are shared between the central node and its neighborhood.

$$\mathbf{H}^{(l+1)} = \sigma((\mathbf{C}\mathbf{H}^{(l)}\mathbf{W}^{(l)}) + \mathbb{1}\mathbf{b}) \tag{3.6}$$

Where $\mathbf{H}^{(l)} \in \mathbb{R}^{|\mathcal{V}| \times f_l}$, f_l being the dimension of the state vector in layer l, $W \in \mathbb{R}^{f_l \times f_{l+1}}$, $\mathbf{C} \in \mathbb{N}^{|\mathcal{V}| \times |\mathcal{V}|}$ and $\mathbf{b} \in \mathbb{R}^{f_{l+1}}$. Note that each line of $\mathbf{H}^{(l)}$ encodes the state of a node for the layer l : $\mathbf{H}^{(l)}(i,:) = \mathbf{h}_i^{(l)}$.

[3]Note that in this equation, the `aggregate` function takes $\mathbf{h}_u$ as input in addition to u neighborhood.

The drawback of such MPNN formulation is that it applies the same coefficients to all neighbors and to the node itself for each feature. By analogy with a convolution on images, it is equivalent to compute the value of a pixel using the same coefficient for the pixel itself and its neighbors (see figure 3.6(a)). Thus, such an MPNN has an effect of smoothing (low-pass filtering). We refer to this model as "Vanilla GNN" later in the manuscript. One improvement that can be made is to apply different coefficients to the neighborhood on one side and to the node on the other side (see figure 3.6(b)). In this context one can define $\mathbf{C}^0 = \mathbf{I}$ and $\mathbf{C}^1 = \mathbf{A}$ (Eq. 3.7) and apply different parameter matrices $\mathbf{W}^{(l,i)}$ to each $\mathbf{C}^i$ matrix. This improvement allows the MPNN to behave either as a high-pass filter or as a low-pass filter depending on the weights applied to the node (matrix $\mathbf{C}^0$) and its neighborhood (matrix $\mathbf{C}^1$).

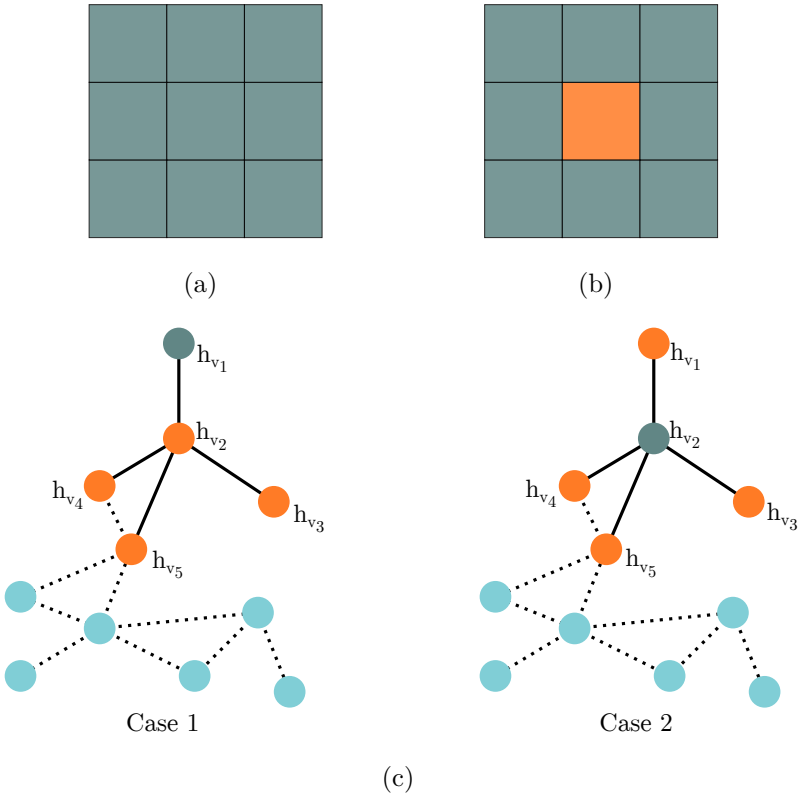


Figure 3.6: Analogy of a MPNN with a convolution on images. (a) shows a filter that would be designed by $\mathbf{C} = \mathbf{A} + \mathbf{I}$ while (b) shows a filter designed by $\mathbf{C}^0 = \mathbf{I}$ and $\mathbf{C}^1 = \mathbf{A}$. Finally, (c) is a simple example presenting the importance of applying different coefficients to the neighborhood and to the node. Indeed, a filter designed as (a) applied on $\mathbf{h}_{v_2}$ would not distinguish case 1 and case 2 while a filter designed as (b) would distinguish them.

$$\mathbf{H}^{(l+1)} = \sigma((\sum_{i=0}^{1} \mathbf{C}^{(i)}\mathbf{H}^{(l)}\mathbf{W}^{(l,i)}) + \mathbf{b}) \tag{3.7}$$

Eq. 3.7 can be extended to consider discrete edge features. Using the extended definition of the adjacency matrix $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|\times\dim(\mathcal{L}_\mathcal{E})}$, one can define $\mathbf{C}^0 = \mathbf{I}$ and $\mathbf{C}^{1:\mathcal{L}_\mathcal{E}} = \mathbf{A}$. The equation thus becomes Equation 3.8. This way, different weight matrices are applied according to edge's label:

$$\mathbf{H}^{(l+1)} = \sigma((\sum_{i=0}^{|\mathcal{L}_\mathcal{E}|} \mathbf{C}^{(i)}\mathbf{H}^{(l)}\mathbf{W}^{(l,i)}) + \mathbf{b}) \tag{3.8}$$

A notable GNN is the Graph Isomorphism Network (GIN) [87] defined by Equation 3.9 :

$$\mathbf{h}_u^{(l+1)} = \texttt{MLP}((1+\epsilon^{(l)}).\mathbf{h}_u^{(l)} + \sum_{v\in\mathcal{N}(u)} \mathbf{h}_v^{(l)}), \tag{3.9}$$

where $\epsilon \in \mathbb{R}$ may be a trainable parameter. As one can see, the GIN definition is a slight improvement from the GNN defined in Equations 3.5 and 3.6. Indeed, instead of defining $\mathbf{C} = \mathbf{A}+\mathbf{I}$, the GIN model defines $\mathbf{C}$ as $\mathbf{C} = \mathbf{A}+(\mathbf{I}+\epsilon)$. The authors proves that a GNN defined as a GIN model is at least as powerful as the Weisfeiler-Lehman Isomorphism Test [83]. The relation between GNNs and Weisfeiler-Lehman Isomorphism Test will be presented in details in subsection 3.7.

To go one step further, some solutions propose to use the attention mechanism [79, 81, 80] in order to assign different weights on the neighborhood depending on each pair of nodes. The rationale is to compute the matrix $\mathbf{C}$ in such a way that $\mathbf{C}_{i,j}$ is an attention factor representing the importance of node v_j when computing a message for node v_i.

$$\mathbf{C}_{i,j} = f\left(\mathbf{h}_i, \mathbf{h}_j, \mathbf{W}_{AT}\right), \tag{3.10}$$

where $\mathbf{W}_{AT}$ encodes the trainable parameters of a neural network and f is a function to be defined.For example, the Graph Attention Network (GAT) [79] defines the $\mathbf{C}$ matrix by first computing a scalar for each pair of node (Equation 3.12), and then apply an element wise function (Equation 3.12) to these values:

$$e_{ij} = \sigma([\mathbf{h}_i^{(l)}\mathbf{W}^{(l)}||\mathbf{h}_j^{(l)}\mathbf{W}^{(l)}])\mathbf{a} \tag{3.11}$$

where $e_{ij} \in \mathbb{R}$, $\mathbf{W}^{(l)} \in \mathbb{R}^{f_l\times f_{l+1}}$ is the weight matrix in layer l, $\mathbf{h}_i^{(l)} \in \mathbb{R}^{1\times f_l}$ and $h_j^{(l)} \in \mathbb{R}^{1\times f_l}$ are the states of nodes v_i and v_j in layer l, σ is a non-linear

function such as the LeakyReLU function, $||$ is the concatenation operator and $\mathbf{a} \in \mathbb{R}^{2f_l}$ is an attention mechanism defined as a weight vector of a neural network. In order to keep the graph structure information, e_{ij} is only computed if $v_j \in \mathcal{N}(v_i)$ and is equal to 0 otherwise. This masked attention mechanism allows to keep the structure of the graph, and do not consider non existing edges in the graph.

Each e_{ij} is then normalized according to Equation 3.12 :

$$\mathbf{C}_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum\limits_{k \in \mathcal{N}(v_i)} \exp(e_{ik})} \tag{3.12}$$

where softmax_j is the normalized exponential function that uses all neighbors of i-th node to normalize edge attention factor of i-th to j-th node.

Multiple attentions can be computed for a single layer which correspond to the multi-head attention framework. In this case, the matrix $\mathbf{C}$ becomes $\mathbf{C} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}| \times n_h}$ where n_h is the number of attention heads used.

If nodes have discrete labels, weights can be shared by the neighbors whose labels are the same [24]. Another method consists in defining an ordering on nodes included within the receptive field of convolution, and sharing the coefficients according to this reordering [63]. The reordering process is called canonical node reordering. A similar sharing approach, based on reordered neighbors, was presented in [35]. The difference is that the reordering is computed according to the absolute correlation of features to the central node. A different spatial-designed method proposed in [6] considers a diffusion process on the graph using random walks. This allows to introduce variability on output signal by applying random walks of different lengths to the different features. A study of different `aggregate` functions has also been proposed in [30] through their GraphSAGE method. The `aggregate` functions studied are the mean aggregator, a LSTM aggregator and a pooling aggregator.

Many other methods fall in the paradigm of spatial GNNs. Once again having an exhaustive look at them is not possible but this subsection has presented the notable ones. In the next subsection, we review notable approaches that rely on the spectral point of view.

3.5 Spectral ConvGNN

Spectral ConvGNNs rely on the spectral graph theory [22]. In this framework, signals on graphs are filtered using the eigendecomposition of graph Laplacian

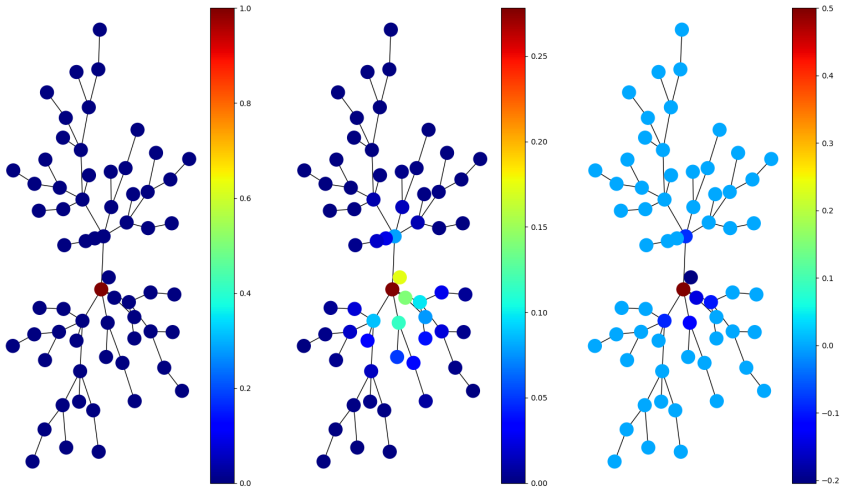


Figure 3.7: Example on filtering applied on the eigenvalues of the graph Laplacian. On the left, the graph before filtering. Each node in the graph has one single feature, which is 0 for each node except the first one which is 1. On the middle, the same graph after a low-pass filtering. On the right, the same graph after a high-pass filtering.

[74]. Graph Laplacian is defined by $\mathbf{L} = \mathbf{D} - \mathbf{A}$ (or $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ for the normalized version), where $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is the adjacency matrix, $\mathbf{D} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is the diagonal degree matrix with entries $\mathbf{D}_{i,i} = \sum_j \mathbf{A}_{j,i}$ and $\mathbf{I}$ is the identity matrix. Since the Laplacian is positive semidefinite, it can be decomposed into $\mathbf{L} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^T$ where $\mathbf{U}$ is a matrix which gathers eigenvectors of $\mathbf{L}$ and $\mathbf{\Sigma} = \text{diag}(\boldsymbol{\lambda})$ where $\boldsymbol{\lambda}$ denotes the vector of the eigenvalues. Note that the Graph Laplacian is a semi positive definite matrix [22], and thus all its eigenvalues are positive.

The graph Fourier transform of any unidimensional signal on graph is defined by $\mathbf{x}_{ft} = \mathbf{U}^\top\mathbf{x}$ and its inverse is given by $\mathbf{x} = \mathbf{U}\mathbf{x}_{ft}$ [22]. By transposing the convolution theorem to graphs, the spectral filtering in the frequency domain can be defined by

$$\mathbf{x}_{filtered} = \mathbf{U}\,\text{diag}(F(\boldsymbol{\lambda}))\mathbf{U}^\top\mathbf{x}, \tag{3.13}$$

where $F(\boldsymbol{\lambda})$ is the desired filter function applied to the eigenvalues $\boldsymbol{\lambda}$. Figure 3.7 shows the effect of two filterings applied over a graph. One can notice that those effects are similar to the ones that can be obtained on images. As a

consequence, a graph convolution layer in spectral domain can be written by a sum of filtered signals followed by an activation function as in [15], namely

$$\mathbf{H}_j^{(l+1)} = \sigma\left(\sum_{i=1}^{f_l} \mathbf{U}\,\mathrm{diag}(\mathbf{F}_{i,j,l})\mathbf{U}^\top \mathbf{H}_i^{(l)}\right), \tag{3.14}$$

for all $j \in \{1, \ldots, f_{l+1}\}$. Here, σ is an activation function such as `ReLU` (Rectified Linear Unit), $\mathbf{H}_i^{(l)} = \mathbf{H}^{(l)}(:, i)$ is the column vector of i-th feature of the l-th layer, $\mathbf{F}_{i,j,l} \in \mathbb{R}^n$ is the corresponding weight vector whose size is the number of eigenvectors (also n, the number of nodes). A spectral ConvGNN based on (3.14) seeks to tune the trainable parameters $\mathbf{F}_{i,j,l}$, as proposed in [36] for the single-graph problem.

A first drawback is the necessity of Fourier and inverse Fourier transform by matrix multiplication of $\mathbf{U}$ and $\mathbf{U}^T$ on one side, and the computation of the eigen decomposition of the graph Laplacian on the other side. Another drawback occurs when generalizing the approach to multi-graph learning problems. Indeed, the k-th element of the vector $\mathbf{F}_{i,j,l}$ weights the contribution of the k-th eigenvector to the output. Those weights are not shareable between graphs of different sizes, which means a different length of $\mathbf{F}_{i,j,l}$ is needed. Moreover, even though the graphs have the same number of nodes, their eigenvalues will be different if their structures differ. As a consequence, a given weight $\mathbf{F}_{i,j,l}$ may correspond to different eigenvalues in different graphs.

This means one weight is associated to different eigenvalues in different graphs. In the Euclidean signal analogy, it means one weight corresponds to different frequency components for different samples. As expected, the number of eigenvalues and their values are different. Since eigenvalues of each graph are different and not aligned, sharing eigenvectors relative to each eigenvalue is not relevant. For instance in Fig. 3.8, bigger graph's 7th eigenvalue is around 4.2 while smaller graph's 7th eigenvalue is around 1.2.

To overcome these issues, a few spatially-localized filters have been defined such as cubic B-spline parameterization [15] and polynomial parameterization [23]. With such approaches, trainable parameters are defined by:

$$\mathbf{F}_{i,j,l} = \mathbf{B}\left[\mathbf{W}_{i,j}^{(l,1)}, \ldots, \mathbf{W}_{i,j}^{(l,S)}\right]^\top, \tag{3.15}$$

where $\mathbf{B} \in \mathbb{R}^{n \times S}$ is an initially designed matrix and $\mathbf{W}^{(l,s)}$ is the trainable matrix for the l-th layer's s-th convolution kernel, $\mathbf{W}_{i,j}^{(l,s)}$ is the (i,j)-th entry of $\mathbf{W}^{(l,s)}$ and S is the desired number of convolution kernels. Each column in $\mathbf{B}$ is designed as a function of eigenvalues, namely $\mathbf{B}_{i,j} = (f_j(\lambda_i))$.

In the polynomial case, each column of $\mathbf{B}$ encodes powers of eigenvalues, starting at 0 ($\mathbf{B}_{i,0} = \lambda_i^0$) and ending at $(S-1)$-th power, eg. $\mathbf{B}_{i,S} = \lambda_i^{S-1}$. In

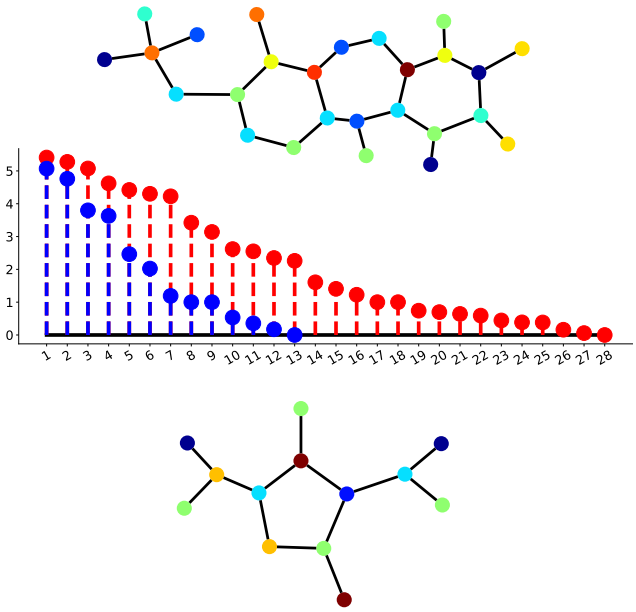


Figure 3.8: Filter Transferability Problem. The respective eigenvalues of a 13-node graph (bottom graph, eigenvalues in blue) versus a 28-node graph (top graph, eigenvalues in red). The number and values of eigenvalues are different. Since the eigenvalue of each graph are different, sharing eigenvectors relative to each eigenvalue is not relevant. Respective eigenvalues are shown in the middle, in red for the former and in blue for the latter. As expected, the number of eigenvalues and their values are different. Since the frequency profiles of each graph are different and not aligned, sharing eigenvectors relative to each eigenvalue is not relevant. For instance, bigger graph's 7th eigenvalue is around 4.2 while smaller graph's 7th is around 1.2.

the cubic B-spline case, the $\mathbf{B}$ matrix encodes the cubic B-spline coefficients [15].

A very recent ConvGNN named CayleyNet proposes to parameterize trainable coefficients by $\mathbf{F}_{i,j,l} = [g_{i,j,l}(\lambda_1, h), ..., g_{i,j,l}(\lambda_n, h)]^\top$, where h is a scale parameter to be learned, λ_n is the n-th eigenvalue, and g is a spectral filter function defined as follows in [51]:

$$g(\lambda, h) = c_0 + 2Re\left(\sum_{k=1}^{r} c_k \left(\frac{h\lambda - \mathbf{i}}{h\lambda + \mathbf{i}}\right)^k\right) \tag{3.16}$$

where $\mathbf{i}^2 = -1$, $Re(\cdot)$ is the function returning the real part, c_0 is a real trainable coefficient, and for $k = 1, \ldots, r$, c_k are the complex trainable coefficients.

The CayleyNet parameterization can also be formulated as in Equation (3.15). CayleyNet model will be used later in this manuscript.

3.6 Spectral-rooted Spatial Convolutions

As said before, some methods have recently been proposed to get rid of the computation burden of graph Fourier and inverse graph Fourier transforms, while still taking their foundations in the spectral domain. These solutions rely on the approximation of a spectral graph convolution proposed in [32], based on the Chebyshev polynomial expansion of the scaled graph Laplacian. Accordingly, the first two Chebyshev kernels are $\mathbf{C}^{(1)} = \mathbf{I}$ and $\mathbf{C}^{(2)} = 2\mathbf{L}/\lambda_{\max} - \mathbf{I}$ and the remaining kernels are defined by

$$\mathbf{C}^{(k)} = 2\mathbf{C}^{(2)}\mathbf{C}^{(k-1)} - \mathbf{C}^{(k-2)}. \tag{3.17}$$

Researchers have shown that any desired filter can be written as a linear combination of these kernels [32]. ChebNet is the first method that used these kernels in ConvGNN [23].

One major extension and simplification of the Chebyshev polynomial expansion method is Graph Convolution Network (GCN) [43]. GCN uses the subtraction of the second Chebyshev kernels from the first one under the assumption of $\lambda_{\max} = 2$ and $\mathbf{L}$ is the normalized graph Laplacian. The fact that $\lambda_{\max} = 2$ for the normalized graph Laplacian has been proven in [22].

However, instead of using this subtracted kernel, they used re-normalization trick and defined the final single kernel by:

$$\mathbf{C} = \widetilde{\mathbf{D}}^{-1/2}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-1/2}, \tag{3.18}$$

with $\widetilde{\mathbf{D}}_{i,i} = \sum_j \widetilde{\mathbf{A}}_{i,j}$ and $\widetilde{\mathbf{A}} = (\mathbf{A} + \mathbf{I})$ the adjacency matrix with added self-connections.

This approach influenced many other contributions. The method described in [91] directly uses this convolution but changes the network architecture by adding a fully connected layer as the last layer. The MixHop algorithm [3] uses the 2nd or 3rd powers of the same convolution.

The methods described in this section are quite different from pure spatial and pure spectral convolutions. They are not designed by using eigenvalues, but are implicitly designed as a function of structural information (adjacency, Laplacian) and perform convolution in spatial domain as any spatial convolution. However, their frequency profiles are stable for different arbitrary graphs as any spectral convolution. This aspect will be theoretically and experimentally illustrated in the next chapter.

3.7 Assessing the expressive power of GNNs

A last interesting point of view about the GNNs is their relation with the Weisfeiler-Lehman (WL) isomorphism test [83]. This has been studied in [87, 56, 62, 25], and to a lesser extent in [30]. The WL isomorphism test starts by assigning to each node the same label (or a label according to the node attributes). Then, iteratively, each node is assigned to a tuple containing the aggregation of node's old compressed label and its neighborhood compressed labels. Then, a new label is assigned to each node depending on the tuple previously computed. The new label is generally computed with a hash function in order to have unique labels. The algorithm says that if after $|\mathcal{V}|$ iterations or after convergence the nodes of two graphs do not have the same labeling, then they are not isomorphic. It has however been proven through counter-examples that 2 graphs with the same labeling are not necessarily isomorphic. An iteration of WL test can be represented by Equation 3.19 and one can easily notice the similarity with MPNN formulation (Eq.3.5).

$$\mathbf{h}_u^{(l+1)} = \texttt{hash}(\mathbf{h}_u^{(l)}, \texttt{aggregate}(\{\mathbf{h}_v^{(l)}, \forall v \in \mathcal{N}(u)\})) \tag{3.19}$$

It is possible to extend the previous algorithm to tuples of nodes, by computing a new label for each tuple of k nodes instead of each node. We then talk about k-WL test, and the WL test becomes the 1-WL test. Apart from the 1-WL and 2-WL case, it has been proven that the $(k+1)$-WL test is strictly stronger than the k-WL test [16], in the way that $k+1$-WL test is able to distinguish two non-isomorphic graphs where the k-WL test fails to detect the difference.

To get rid of the limitation induced by the fact the 2-WL test is not more powerful than 1-WL test, some variations of this test have been proposed. One commonly used is the Folklore WL (FWL) test, which is defined such as 1-WL = 1-FWL, but for $k \geq 2$, there is $(k+1)$-WL $\approx k$-FWL.

Notable GNNs has thus been directly designed in order to be as powerful as some WL test. For example, GIN [87], which have already been presented in this chapter, has been designed in order to be as powerful as 1-WL test. In [62] and [56], GNNs as powerful as 3-WL test have been designed. However, this representation power comes with an exponentially growing memory and computational complexity.

The equivalence to k-WL test or k-FWL has thus become a way to assess the expressive power of GNNs. A GNN proven to be as powerful than the 3-WL test would be stronger than a GNN proven to be as powerful than the 2-WL test. However, this does not allows a comparison between two GNNs that are as powerful to the same k-WL test. Also, and probably more important, this

only compares how the structural information is used. Features on edges are not taken into account in those comparisons.

3.8 Conclusion

This chapter has presented the Graph Neural Network paradigm. GNNs are a class of neural networks dedicated to graphs. It consists in computing node embeddings through hidden representations obtained by gathering information from node features, edge features and the graph structure. This embedding is then used either directly to produce an output for each node, or in combination with a `readout` function to produce an output for the whole graph. Firstly presented in [28, 72], GNNs have recently been the center of a huge interest by the machine learning community, as shown by the number of submissions concerning GNNs in different conferences such as ICLR.

The development of GNNs has then been made following two different ways in parallel. On one side, spectral graph theory has lead to graph neural networks that operate in the spectral domain, such as CayleyNet. On the other side, GNNs that rely on spatial domain have also been proposed, with methods such as MPNN. Those developments have lead to two distinct categories of GNNs: the spatial and the spectral approaches.

We also introduced a third category of GNNs, the Spectral-Rooted Spatial Convolutions. This category corresponds to GNNs that are based on the spectral domain, but applied in a spatial way. This third category led us to question ourselves on what is the fundamental difference between Spectral and Spatial approaches.

We propose one answer to this question in the following chapter, where we claim and prove that most of spectral and spatial GNNs can be written as one general model, and that there is thus no fundamental differences between Spectral and Spatial GNNs.

Chapter 4

Graph Neural Networks: Are they Spectral or Spatial ?

4.1 Introduction

In previous chapter, we presented a general introduction on Graph Neural Networks (GNN). This introduction has presented a major dichotomy made between two different kind of GNNs: Spectral GNN and Spatial GNN. This chapter was concluded by asking what was the fundamental difference between Spectral and Spatial, and we claimed that the dichotomy generally made had in fact no reason to be made. This chapter aim to prove this fact.

Thus, in section 4.2, we present our second contribution in this manuscript that consists in bridging the gap between spectral and spatial domains for GNNs. Through a new general framework, we demonstrate in this section the equivalence of convolution processes regardless if they are designed in the spatial or the spectral domain.

Taking advantage of this result, we present in section 4.3 our third contribution which consists in a spectral analysis of existing graph convolutions for five popular GNNs, known as GCN [43], ChebNet [23], CayleyNet [51], GIN [87] and Graph Attention Networks (GAT) [79].

This analysis shows that most of influential GNNs are in fact limited to low-pass filtering. To measure the generalization capability of the GNN model, we present in section 4.4 two toy problems that illustrate once again the fact that most of GNNs models are not able to produce band-pass filtering.

4.2 Bridging the gap between Spatial and Spectral GNN

In this section, we firstly provide a theoretical analysis demonstrating that parameterized spectral ConvGNNs can be implemented as spatial ConvGNNs when they use a fixed frequency profile matrix B.

4.2.1 Theoretical analysis

Theorem 1. *Spectral ConvGNN parameterized with fixed frequency profiles matrix $\boldsymbol{B}$ of entries $\boldsymbol{B}_{i,j} = F_j(\lambda_i)$, defined as*

$$\boldsymbol{H}_j^{(l+1)} = \sigma\Big(\sum_{i=1}^{f_l} \boldsymbol{U}\,\mathrm{diag}\left(B\left[\boldsymbol{W}_{i,j}^{(l,1)},\ldots,\boldsymbol{W}_{i,j}^{(l,S)}\right]^\top\right)\boldsymbol{U}^\top \boldsymbol{H}_i^{(l)}\Big), \tag{4.1}$$

is a particular case of spatial ConvGNN, defined as

$$\boldsymbol{H}^{(l+1)} = \sigma\Big(\sum_s \boldsymbol{C}^{(s)}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l,s)}\Big), \tag{4.2}$$

with the convolution kernel set to

$$\boldsymbol{C}^{(s)} = \boldsymbol{U}\,\mathrm{diag}(F_s(\boldsymbol{\lambda}))\,\boldsymbol{U}^\top, \tag{4.3}$$

where the columns of $\boldsymbol{U}$ are the eigenvectors of the studied graph, σ is the activation function, $\boldsymbol{H}^{(l)} \in \mathbb{R}^{n\times f_l}$ is the l-th layer's feature matrix with f_l features, $\boldsymbol{H}_i^{(l)}$ is the i-th column of $\boldsymbol{H}^{(l)}$, $\boldsymbol{B} \in \mathbb{R}^{n\times S}$ is an apriori designed matrix for each graph's eigenvalues, and $F_s(\boldsymbol{\lambda})$ is the s-th column of $\boldsymbol{B}$. $\boldsymbol{W}^{(l,s)}$ and S are defined in (3.15).

Proof. First, let us expand the matrix $\mathbf{B}$ and rewrite it as the sum of its columns, denoted $F_1(\boldsymbol{\lambda}),\ldots,F_S(\boldsymbol{\lambda}) \in \mathbb{R}^n$:

$$\mathbf{H}_j^{(l+1)} = \sigma\left(\sum_{i=1}^{f_l} \mathbf{U}\,\mathrm{diag}\Big(\sum_{s=1}^{S} \mathbf{W}_{i,j}^{(l,s)} F_s(\boldsymbol{\lambda})\Big)\mathbf{U}^\top \mathbf{H}_i^{(l)}\right). \tag{4.4}$$

Now, we distribute $\mathbf{U}$ and $\mathbf{U}^\top$ over the inner summation:

$$\mathbf{H}_j^{(l+1)} = \sigma\left(\sum_{s=1}^{S}\sum_{i=1}^{f_l} \mathbf{U}\,\mathrm{diag}\Big(\mathbf{W}_{i,j}^{(l,s)} F_s(\boldsymbol{\lambda})\Big)\mathbf{U}^\top \mathbf{H}_i^{(l)}\right). \tag{4.5}$$

Then, we take out the scalars $\mathbf{W}_{i,j}^{(l,s)}$ of the diag operator:

$$\mathbf{H}_j^{(l+1)} = \sigma\left(\sum_{s=1}^{S}\sum_{i=1}^{f_l}\mathbf{W}_{i,j}^{(l,s)}\mathbf{U}\,\mathrm{diag}(F_s(\boldsymbol{\lambda}))\mathbf{U}^\top\mathbf{H}_i^{(l)}\right). \tag{4.6}$$

Let us define a convolution operator $\mathbf{C}^{(s)} \in \mathbb{R}^{n\times n}$ as:

$$\mathbf{C}^{(s)} = \mathbf{U}\,\mathrm{diag}(F_s(\boldsymbol{\lambda}))\mathbf{U}^\top. \tag{4.7}$$

Using (4.6) and (4.7), we have thus:

$$\mathbf{H}_j^{(l+1)} = \sigma\left(\sum_{i=1}^{f_l}\sum_{s=1}^{S}\mathbf{W}_{i,j}^{(l,s)}\mathbf{C}^{(s)}\mathbf{H}_i^{(l)}\right). \tag{4.8}$$

Then, each term of the sum over s corresponds to a matrix $\mathbf{H}^{(l+1)} \in \mathbb{R}^{n\times f_{l+1}}$ with

$$\mathbf{H}^{(l+1)} = \sigma\left(\mathbf{C}^{(1)}\mathbf{H}^{(l)}\mathbf{W}^{(l,1)} + \cdots + \mathbf{C}^{(S)}\mathbf{H}^{(l)}\mathbf{W}^{(l,S)}\right), \tag{4.9}$$

with $\mathbf{H}^{(l)} = [\mathbf{H}_1^{(l)}, \ldots, \mathbf{H}_{f_l}^{(l)}]$. We get by grouping the terms:

$$\mathbf{H}^{(l+1)} = \sigma\left(\sum_{s=1}^{S}\mathbf{C}^{(s)}\mathbf{H}^{(l)}\mathbf{W}^{(l,s)}\right), \tag{4.10}$$

which corresponds to (4.2). Therefore, (4.1) corresponds to (4.2) with $\mathbf{C}^{(s)}$ defined as (4.7). □

This theorem is general, since it covers many well-known spectral ConvGNNs, such as non-parametric spectral graph convolution [36], polynomial parameterization [23], cubic B-spline parameterization [15] and CayleyNet [51].

From Theorem 1, designing a graph convolution either in spatial or in spectral domain is equivalent. Therefore, Fourier calculations are not necessary when convolutions are parameterized by an initially designed matrix B. Using that relation, it is not difficult to show the spatial equivalence of non-parametric spectral graph convolution defined in (3.14). It can be written in spatial domain with $B = I$ in (3.15). It thus corresponds to (4.2) where each convolution kernel is defined by $\mathbf{C}^{(s)} = \mathbf{U}_s\mathbf{U}_s^\top$, where $\mathbf{U}_s$ is the s-th eigenvector.

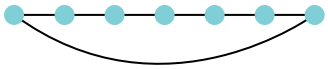


Figure 4.1: Example of a circular graph made of 7 nodes.

4.3 Spectral Analysis of Existing Graph Convolutions

Using the results of the preceding section, this section aims at providing a deeper understanding of the graph convolution process through an analysis of existing GNNs in the spectral domain. To the best of our knowledge, no one has led such an analysis concerning graph convolutions in the literature. In this section, we show how it can be done on four well-known graph convolutions: ChebNet [23], CayleyNet [51], GCN [43] and GAT [79]. This analysis is led using the following corollary of Theorem 1.

Corollary 1.1. *The frequency profile of any given graph convolution kernel* $\boldsymbol{C}^{(s)}$ *can be defined in spectral domain by the vector*

$$\digamma_s(\boldsymbol{\lambda}) = \text{diag}^{-1}(\boldsymbol{U}^\top \boldsymbol{C}^{(s)} \boldsymbol{U}). \tag{4.11}$$

Proof. By using (4.3) from Theorem 1, we can obtain a spatial convolution kernel $\mathbf{C}^{(s)}$ whose frequency profile is $\digamma_s(\boldsymbol{\lambda})$. Since the eigenvector matrix is orthogonal (i.e., $\mathbf{U}^{-1} = \mathbf{U}^\top$), we can extract $\digamma_s(\boldsymbol{\lambda})$, which yields (4.11). □

We denote the matrix $\digamma_s = \mathbf{U}^\top \mathbf{C}^{(s)} \mathbf{U}$ as the full frequency profile of the convolution kernel $\mathbf{C}^{(s)}$, and $\digamma_s(\boldsymbol{\lambda}) = \text{diag}(\digamma_s)$ as the standard frequency profile of the convolution kernel. The full frequency profile includes all eigenvector-to-eigenvector pairs contributions. Standard frequency profile just includes each eigenvector's self-contribution.

To show the frequency profiles of some well-known graph convolutions, we used three graphs. The first one corresponds to a 1D signal encoded as a regular circular line graph with 1001 nodes. A circular graph can be seen as a signal whose last moment is followed by the first moment. Figure 4.1 presents an example of a circular graph. The second and third ones are the Cora and Citeseer reference datasets, which consist of one single graph with respectively 2708 and 3327 nodes [88]. Basically, each node of these graphs is labeled by a vector, and edges are unlabeled and undirected. These two graphs will be described in details in Chapter 5.

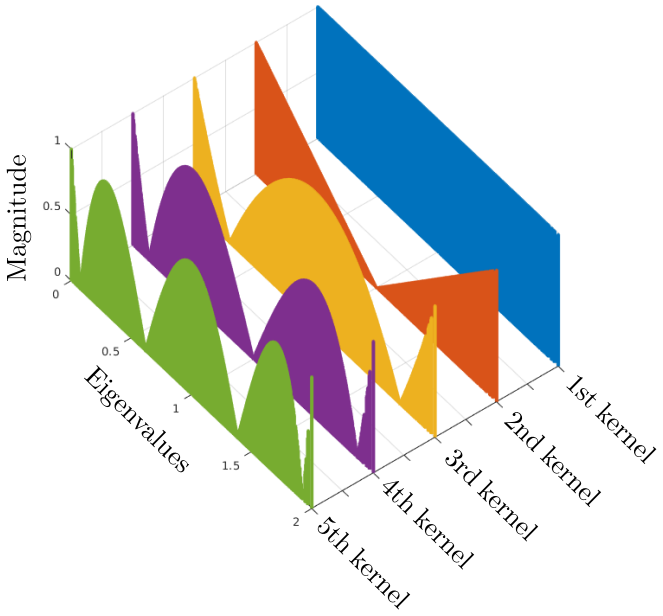


Figure 4.2: Standard frequency profiles of first 5 Chebyshev convolutions.

ChebNet

After computing the kernels of ChebNet by (3.17), Corollary 1.1 can be used to obtain their frequency profiles.

Theorem 2. *The frequency profile of the first Chebyshev convolution kernel for any undirected arbitrary graph defined by* $C^{(1)} = I$ *can be defined by :*

$$\digamma_1(\boldsymbol{\lambda}) = 1, \tag{4.12}$$

Proof. It is needless to say that if the identity matrix is used as convolution kernel, it just directly transmits the inputs to the outputs without any modification. This process is called all-pass filter. Mathematically, we can calculate the full frequency profile for kernel I by using Corollary 1.1

$$\digamma_1 = \mathbf{U}^\top \mathbf{I} \mathbf{U} = \mathbf{U}^\top \mathbf{U} = \mathbf{I}, \tag{4.13}$$

Hence, $\mathbf{U}$ contains the eigenvectors which are orthogonal to each other, $\mathbf{U}^\top \mathbf{U} = I$. In the full frequency profile, all the elements out of the diagonal must be zero and all the elements on the diagonal must be 1. So we can parameterize diagonal of the full frequency profile by $\boldsymbol{\lambda}$ and reach the standard frequency profile as follows:

$$\digamma_1(\boldsymbol{\lambda}) = \mathrm{diag}(\mathbf{I}) = 1, \tag{4.14}$$

□

Theorem 3. *The frequency profile of the second Chebyshev convolution kernel for any undirected arbitrary graph given by* $\boldsymbol{C}^{(2)} = 2\boldsymbol{L}/\lambda_{\max} - \boldsymbol{I}$ *can be defined by:*

$$\boldsymbol{F}_2(\boldsymbol{\lambda}) = \frac{2\boldsymbol{\lambda}}{\lambda_{\max}} - 1, \tag{4.15}$$

Proof. We can calculate the $\mathbf{C}^{(2)}$ kernel full frequency profile using Corollary 1.1:

$$\boldsymbol{F}_2 = \mathbf{U}^\top \left(\frac{2}{\lambda_{\max}}\mathbf{L} - \mathbf{I}\right)\mathbf{U}, \tag{4.16}$$

Since $\mathbf{U}^\top\mathbf{I}\mathbf{U} = \mathbf{I}$, (4.16) can be rearranged as:

$$\boldsymbol{F}_2 = \frac{2}{\lambda_{\max}}\mathbf{U}^\top\mathbf{L}\mathbf{U} - \mathbf{I}, \tag{4.17}$$

Since $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_n]$ are the eigenvalues of the graph laplacian $\mathbf{L}$, they must conform to the following condition:

$$\mathbf{L}\mathbf{U} = \mathbf{U}\,\mathrm{diag}(\boldsymbol{\lambda}) \tag{4.18}$$

$$\mathbf{U}^\top\mathbf{L}\mathbf{U} = \mathrm{diag}(\boldsymbol{\lambda}) \tag{4.19}$$

Replacing (4.19) into (4.17), we get:

$$\boldsymbol{F}_2 = \frac{2}{\lambda_{\max}}\,\mathrm{diag}(\boldsymbol{\lambda}) - \mathbf{I}, \tag{4.20}$$

This full frequency profile consists of two parts. One is the negative identity matrix. The second part is also a diagonal matrix whose values are the eigenvalues. So we can parameterize the full frequency matrix diagonal to show the standard frequency profile as follows:

$$\boldsymbol{F}_2(\boldsymbol{\lambda}) = \mathrm{diag}(\boldsymbol{F}_2) = \frac{2\boldsymbol{\lambda}}{\lambda_{\max}} - 1, \tag{4.21}$$

□

Theorem 4. *The frequency profile of third and following Chebyshev convolution kernels for any undirected arbitrary graph can be defined by :*

$$\boldsymbol{F}_k = 2\boldsymbol{F}_2\boldsymbol{F}_{k-1} - \boldsymbol{F}_{k-2}, \tag{4.22}$$

Proof. Given the third and following Chebyshev kernel defined by $\mathbf{C}^{(k)} = 2\mathbf{C}^{(2)}\mathbf{C}^{(k-1)} - \mathbf{C}^{(k-2)}$ and using Corollary 1.1, its frequency profile is:

$$\mathcal{F}_k = \mathbf{U}^\top \left(2\mathbf{C}^{(2)}\mathbf{C}^{(k-1)} - \mathbf{C}^{(k-2)}\right)\mathbf{U}, \tag{4.23}$$

Expanding (4.23), we get:

$$\mathcal{F}_k = 2U^\top\mathbf{C}^{(2)}\mathbf{C}^{(k-1)}\mathbf{U} - \mathbf{U}^\top\mathbf{C}^{(k-2)}\mathbf{U}, \tag{4.24}$$

Since $\mathbf{U}\mathbf{U}^\top = \mathbf{I}$, we can insert the product $\mathbf{U}\mathbf{U}^\top$ into (4.24). We have thus:

$$\mathcal{F}_k = 2\mathbf{U}^\top\mathbf{C}^{(2)}\mathbf{U}\mathbf{U}^\top\mathbf{C}^{(k-1)}\mathbf{U} - \mathbf{U}^\top\mathbf{C}^{(k-2)}\mathbf{U} \tag{4.25}$$

$$\mathcal{F}_k = 2\left(\mathbf{U}^\top\mathbf{C}^{(2)}\mathbf{U}\right)\left(\mathbf{U}^\top\mathbf{C}^{(k-1)}\mathbf{U}\right) - \mathbf{U}^\top\mathbf{C}^{(k-2)}\mathbf{U} \tag{4.26}$$

$$\tag{4.27}$$

Since $\mathcal{F}_n = \mathbf{U}^\top\mathbf{C}^{(n)}\mathbf{U}$, it follows that (4.25) and (4.22) are identical.

□

Hence $\mathcal{F}_1$ and $\mathcal{F}_2$ are diagonal matrices, rest of the kernels frequency profiles become diagonal matrices in (4.22). So we can write standard frequency profiles of third and following Chebyshev convolution kernels as follows;

$$\mathcal{F}_k(\boldsymbol{\lambda}) = 2\mathcal{F}_2(\boldsymbol{\lambda})\mathcal{F}_{k-1}(\boldsymbol{\lambda}) - \mathcal{F}_{k-2}(\boldsymbol{\lambda}), \tag{4.28}$$

In summary, the first two kernel frequency profiles of ChebNet are $\mathcal{F}_1(\boldsymbol{\lambda}) = 1$ and $\mathcal{F}_2(\boldsymbol{\lambda}) = 2\boldsymbol{\lambda}/\lambda_{\max} - 1$. Since $\lambda_{\max} = 2$ for any normalized graph Laplacian [22], we get $\mathcal{F}_2(\boldsymbol{\lambda}) = \boldsymbol{\lambda} - 1$. The third one and following kernel frequency profiles can also be computed using $\mathcal{F}_k(\boldsymbol{\lambda}) = 2\mathcal{F}_2(\boldsymbol{\lambda})\mathcal{F}_{k-1}(\boldsymbol{\lambda}) - \mathcal{F}_{k-2}(\boldsymbol{\lambda})$, leading to $\mathcal{F}_3(\boldsymbol{\lambda}) = \boldsymbol{\lambda}^2 - 4\boldsymbol{\lambda} + 1$ for example for the third kernel. The resulting 5 frequency profiles are shown in Figure 4.2 (in absolute value). Since the full frequency profiles consist of zeros outside the diagonal, they are not illustrated.

Analyzing the frequency profile of ChebNet, one can argue that the convolutions mostly cover the spectrum. However, none of the kernels focuses on some certain parts of the spectrum. As an example, the second kernel is mostly a low-pass and high-pass filter and stops the middle band, while the third one passes very high, very low and middle bands, but stops almost first and third quarter of the spectrum. Therefore, if the relation between input-output pairs can be figured out by just a low-pass, high-pass or some specific band-pass filter, a high number of convolution kernels is needed. However, in the literature, only 2 or 3 kernels are generally used for experiments [23, 43].

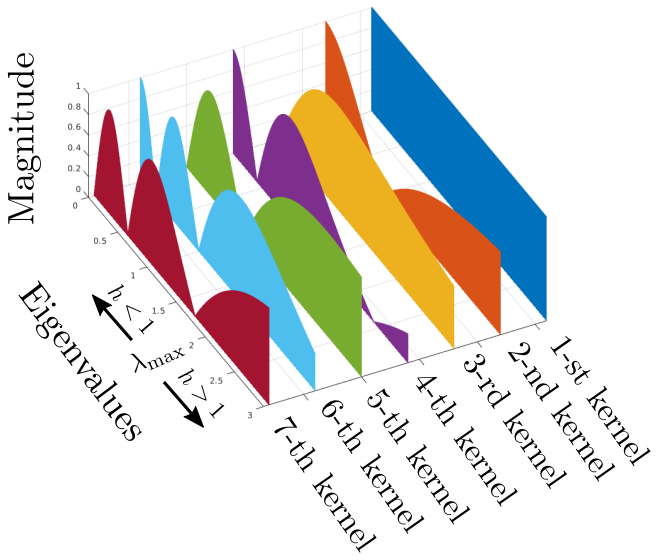


Figure 4.3: Standard frequency profiles of first 7 CayleyNet convolutions.

CayleyNet

CayleyNet uses spectral graph convolutions whose frequency profiles can be changed by scaling eigenvalues [51]. The frequency profile is defined by a complex rational function of eigenvalues, scaled by a trainable parameter h in (3.16).

CayleyNet uses $F_{i,j,l} = [g_{i,j,l}(\lambda_1, h), ..., g_{i,j,l}(\lambda_n, h)]^\top$ in (3.14) where function g was defined in [51] by:

$$g(\lambda, h) = c_0 + 2Re\left(\sum_{k=1}^{r} c_k \left(\frac{h\lambda - \mathbf{i}}{h\lambda + \mathbf{i}}\right)^k\right) \tag{4.29}$$

where $\mathbf{i}^2 = -1$, $Re(\cdot)$ is the function that returns the real part of a given complex number, c_0 is a trainable real coefficient, and for $k = 1..r$, c_ks are the complex trainable coefficients. We can write $h\lambda - \mathbf{i}$ in Euler form by $\sqrt{h^2\lambda^2 + 1}.e^{\mathbf{i}.\,\mathrm{atan2}(-1,h\lambda)}$ and for $h\lambda + \mathbf{i}$ by $\sqrt{h^2\lambda^2 + 1}.e^{\mathbf{i}.\,\mathrm{atan2}(1,h\lambda)}$. By this substitution, (4.29) becomes:

$$g(\lambda, h) = c_0 + 2Re\left(\sum_{k=1}^{r} c_k e^{\mathbf{i}.k.(\mathrm{atan2}(-1,h\lambda) - \mathrm{atan2}(1,h\lambda))}\right) \tag{4.30}$$

where atan2(y, x) is the inverse tangent function which finds the angle (in range of $[-\pi, \pi]$) of a point given its y and x coordinates. For further simplification, let's introduce the $\theta(\cdot)$ function defined by:

$$\theta(x) = \text{atan2}(-1, x) - \text{atan2}(1, x) \tag{4.31}$$

Since c_k are complex numbers, we can write them as a sum of real and imaginary part $c_k = a_k + \mathbf{i}b_k$. Then (4.30) can be rewritten as follows:

$$g(\lambda, h) = c_0 + 2Re\left(\sum_{k=1}^{r}(a_k + \mathbf{i}b_k)e^{\mathbf{i}.k.\theta(h\lambda)}\right) \tag{4.32}$$

We can replace $e^{\mathbf{i}.k.\theta(h\lambda)}$ with its polar coordinate equivalence form $\cos(k.\theta(h\lambda))+ \mathbf{i}.\sin(k.\theta(h\lambda))$. When we remove the imaginary components because of $Re(.)$ function, (4.32) becomes:

$$g(\lambda, h) = c_0 + 2\left(\sum_{k=1}^{r} a_k.\cos(k.\theta(h\lambda)) - b_k.\sin(k.\theta(h\lambda))\right) \tag{4.33}$$

In this definition, there is no complex coefficient. There are c_0 and a_k, b_k, $\forall k = 1 \dots r)$ that are real coefficients to be tuned by training. Because they are coefficients, we can write $a_k = a_k/2$ and $b_k = b_k/2$. So the coefficient 2 in front of the summation has no meaning. By using the form in (4.33), we can parametrize CayleyNet by the parametrization matrix $B \in \mathbb{R}^{n\times 2r+1}$, as in (3.15), by:

$$[g(\lambda_0, h), \dots, g(\lambda_n, h)]^\top = B[c_0, a_1, b_1, \dots, a_r, b_r]^\top \tag{4.34}$$

where $\mathbf{B}_s$ is the s-th column vector of matrix $\mathbf{B}$, and must fulfill the following conditions:

$$\mathbf{B}_s = \digamma_s(\boldsymbol{\lambda}) = \begin{cases} 1 & s = 1 \\ \cos(\frac{s}{2}.\theta(h\boldsymbol{\lambda})) & s \in \{2, 4, \dots, 2r\} \\ -\sin(\frac{s-1}{2}.\theta(h\boldsymbol{\lambda})) & s \in \{3, 5, \dots, 2r+1\} \end{cases} \tag{4.35}$$

We can see CayleyNet as a spectral graph convolution which uses $2r + 1$ convolution kernels. First kernel is an all-pass filter, and the frequency profiles of remaining kernels ($\digamma_s(\boldsymbol{\lambda})$) are created using sine and cosine functions, with a parameter h used to scale the eigenvalues in (4.35). Considering (4.3) in Theorem 1, we can write CayleyNet's convolutions ($\mathbf{C}^{(s)}$) in spatial domain. CayleyNet includes the tuning of this scaling parameter in the training pipeline. Note that because of the function definition in (4.31), $\theta(h\lambda)$ is not linear w.r.t. λ. Therefore, $\digamma_s$ cannot be a perfect sinusoidal w.r.t. λs.

CayleyNet can be defined through the frequency profile matrix $\mathbf{B}$. Using this representation, CayletNet can be seen as multi-kernel convolutions with real-valued trainable coefficients. According to this analysis, CayleyNet uses $2r+1$ graph convolution kernels, with r being the number of complex coefficients [51]. The first 7 kernel's frequency profiles are illustrated in Figure 4.3. The scale parameter h affects the x-axis scaling but does not change the global shape. When $h = 1$, frequency profiles can be defined within the range $[0, 2]$ (because $\lambda_{\max} = 2$ in all three test graphs). If $h = 1.5$, the frequency profile can be defined till $1.5\lambda_{\max} = 3$ in Fig. 4.3 and rescale axis label from $[0, 3]$ to $[0, 2]$ in original range.

Learning the scaling of eigenvalues may seem advantageous. However, it induces extra computational cost in order to calculate the new convolution kernel. To limit this cost, an approximation is computed using a fixed number of Jacobi iterations [51]. In addition, similarly to ChebNet, CayleyNet does not have any band specific convolutions, even when considering different scaling factors.

GCN

Theorem 5. *The frequency profile of GCN convolution kernel is defined by:*

$$\boldsymbol{C}_{GCN} = \widetilde{\boldsymbol{D}}^{-1/2}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-1/2}, \tag{4.36}$$

and can be written as:

$$F_{GCN}(\boldsymbol{\lambda}) = 1 - \frac{p}{p+1}\boldsymbol{\lambda} \tag{4.37}$$

where $\boldsymbol{\lambda}$ is the eigenvalues of the normalized graph laplacian and the given graph is an undirected regular graph whose node degrees are all equal to p.

Proof. Since $\widetilde{\mathbf{D}}_{i,i} = \sum_j \widetilde{\mathbf{A}}_{i,j}$ and $\widetilde{\mathbf{A}} = (\mathbf{A} + \mathbf{I})$, we can rewrite (4.36) as:

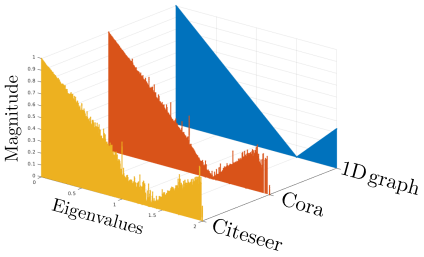
$$\mathbf{C}_{GCN} = (\mathbf{D} + \mathbf{I})^{-1/2}(\mathbf{A} + \mathbf{I})(\mathbf{D} + \mathbf{I})^{-1/2}, \tag{4.38}$$

Under the assumption that all node degrees are equal to p, we can write the diagonal degree matrix by $\mathbf{D} = p\mathbf{I}$. Then, (4.38) can be rewritten as follows:

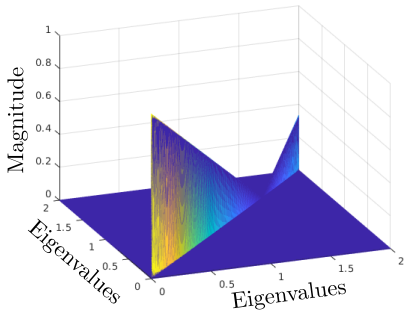
$$\mathbf{C}_{GCN} = ((p+1)\mathbf{I})^{-1/2}(\mathbf{A} + \mathbf{I})((p+1)\mathbf{I})^{-1/2}, \tag{4.39}$$

which is equivalent to:

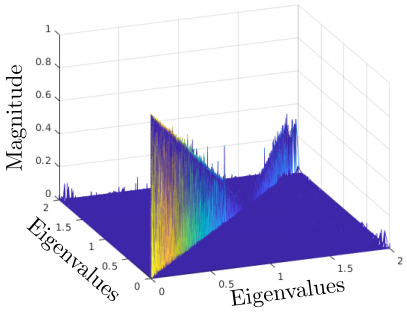
$$\mathbf{C}_{GCN} = \frac{\mathbf{A} + \mathbf{I}}{p+1}, \tag{4.40}$$



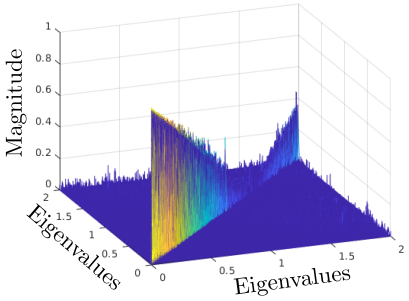
(a) Standard frequency profiles



(b) Full frequency profile on 1D regular line graph



(c) Full frequency profile on Cora



(d) Full frequency profile on Citeseer

Figure 4.4: Frequency profiles of GCN on different graphs.

Using Corollary 1.1, one can express the frequency profile of $\mathbf{C}_{GCN}$ in matrix form by:

$$\mathcal{F}_{GCN} = \frac{1}{p+1}\mathbf{U}^\top\mathbf{A}\mathbf{U} + \frac{1}{p+1}\mathbf{I} \tag{4.41}$$

Since $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_n]$ are the eigenvalues of the normalized graph Laplacian $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, those must conform to the following condition:

$$\left(\mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\right)\mathbf{U} = \mathbf{U}\,\mathrm{diag}(\boldsymbol{\lambda}), \tag{4.42}$$

According to the assumption $\mathbf{D} = p\mathbf{I}$, it conforms to $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2} = \frac{\mathbf{A}}{p}$, so (4.42) can be written as:

$$\mathbf{U} - \frac{\mathbf{A}\mathbf{U}}{p} = \mathbf{U}\,\mathrm{diag}(\boldsymbol{\lambda}), \tag{4.43}$$

Then $\mathbf{A}\mathbf{U}$ is expressed as:

$$\mathbf{A}\mathbf{U} = p\mathbf{U} - p\mathbf{U}\,\mathrm{diag}(\boldsymbol{\lambda}), \tag{4.44}$$

Replacing $\mathbf{AU}$ in (4.41), we obtain:

$$\mathcal{F}_{GCN} = \frac{1}{p+1}\mathbf{U}^\top \left(p\mathbf{U} - p\mathbf{U}\,\mathrm{diag}(\boldsymbol{\lambda})\right) + \frac{1}{p+1}\mathbf{I} \tag{4.45}$$

Since $\mathbf{U}^\top\mathbf{U} = \mathbf{I}$, we have then:

$$\mathcal{F}_{GCN} = \frac{p\mathbf{I} - p\,\mathrm{diag}(\boldsymbol{\lambda}) + \mathbf{I}}{p+1} \tag{4.46}$$

(4.46) can be simplified to:

$$\mathcal{F}_{GCN} = \mathbf{I} - \frac{p}{p+1}\,\mathrm{diag}(\boldsymbol{\lambda}) \tag{4.47}$$

which is equal to the matrix form defined in (4.37). □

This demonstration shows that the GCN frequency profile acts as a low-pass filter. When the given graph is a circular undirected graph, all node degrees are equal to $p = 2$, leading to a frequency profile defined by $1 - 2\boldsymbol{\lambda}/3$. Since the normalized graph laplacian eigenvalues are in the range $[0..2]$, the filter magnitude linearly decreases until the third quarter of the spectrum (cut-off frequency) where it reaches zero. Then it linearly increases until the end of the spectrum. This explains the shape of the frequency profile of GCN convolutions for 1D regular graph observed on Figure 4.4.

However, this conclusion cannot explain the perturbations on the GCN frequency profile. To analyse that point, let's neglect the assumption of $\mathbf{D} = p\mathbf{I}$ and rewrite (4.38) by expanding left and right $(\mathbf{D}+\mathbf{I})^{-1/2}$ terms:

$$\mathbf{C}_{GCN} = (\mathbf{D}+\mathbf{I})^{-1} + (\mathbf{D}+\mathbf{I})^{-1/2}\mathbf{A}(\mathbf{D}+\mathbf{I})^{-1/2}, \tag{4.48}$$

Given (4.48), we can see that the GCN kernel consists of two parts, $\mathbf{C}_{GCN} = \mathbf{C}^{(1)} + \mathbf{C}_2$, where first part is given by $\mathbf{C}^{(1)} = (\mathbf{D}+I)^{-1}$ and the second one is $\mathbf{C}^{(2)} = (\mathbf{D}+\mathbf{I})^{-1/2}\mathbf{A}(\mathbf{D}+\mathbf{I})^{-1/2}$. With the element wise multiplication operator $\odot$, we can write the $\mathbf{C}^{(2)}$ part as:

$$\mathbf{C}^{(2)} = \mathbf{A} \odot \sqrt{1/(\mathbf{d}+1)}.\sqrt{1/(\mathbf{d}+1)}^\top, \tag{4.49}$$

where $\mathbf{d}$ is the column degree vector $\mathbf{d} = \mathrm{diag}(\mathbf{D})$.

With the same notation, we can rewrite the Chebyshev second kernel assuming that $\lambda_{\max} = 2$:

$$\mathbf{C}^{(2)}_{CHEB} = -\mathbf{A} \odot \sqrt{1/\mathbf{d}}.\sqrt{1/\mathbf{d}}^\top, \tag{4.50}$$

(4.49) and (4.50) show that negative $\mathbf{C}^{(2)}_{GCN}$ is an approximation of the second Chebyshev kernel if vector d consists of same values, as it was assumed in Theorem 5. When $\mathbf{d}$ vector is composed of different values, $\sqrt{1/\mathbf{d}}.\sqrt{1/\mathbf{d}}^{\top}$ matrix and $\sqrt{1/(\mathbf{d}+1)}.\sqrt{1/(\mathbf{d}+1)}^{\top}$ matrix are not proportional for each coordinate. To obtain $\mathbf{C}^{(2)}_{GCN}$ from $\mathbf{C}^{(2)}_{CHEB}$, we need to use different coefficients for each coordinate of the kernel. If the difference between node degrees is important, these coefficients have a strong influence, and $\mathbf{C}^{(2)}_{GCN}$ may be very different from $\mathbf{C}^{(2)}_{CHEB}$. Conversely, if the node degrees are quite uniform, these coefficients may be neglected. This phenomenon is the first cause of perturbation on GCN frequency profile.

The first part ($\mathbf{C}_1$) of the GCN kernel is more interesting. Actually, it is just a diagonal matrix which shows the contribution of each node in the convolution process. Instead of looking for some approximations of known frequency profiles such as those of Chebyshev kernels, we can write its frequency profile directly. Using Corollary 1.1, we can express the frequency profile of $\mathbf{C}_1$ in matrix form by:

$$F_{\mathbf{C}_1} = (\mathbf{U}^{\top}\mathbf{C}_1\mathbf{U}), \tag{4.51}$$

where $\mathbf{U}$ consists of eigenvectors. By taking advantage of having a diagonal kernel $\mathbf{C}_1$, we can express each component of full frequency profile as:

$$F_{\mathbf{C}_1}(i,j) = \sum_{k=1}^{n} \left(\frac{1}{1+\mathbf{d}_k} \mathbf{U}_{i,k}\mathbf{U}_{j,k} \right), \tag{4.52}$$

where n is the number of nodes in the graph, d_k is degree of k-th node, $\mathbf{U}_{i,k}$ is the k-th element of i-th eigenvector. As eigenvectors $\mathbf{U}_i$ and $\mathbf{U}_j$ are orthogonal for $i \neq j$, their dot product is 0. However, in (4.52), the weighting coefficient $\frac{1}{1+\mathbf{d}_k}$ is not constant over all the dimensions the eigenvectors. There is no guarantee that $F_{\mathbf{C}_1}(i,j)$ is null. This is an other reason which explains that the GCN frequency profile has many non-zero elements out of the diagonal.

In addition, it is also clear that the standard frequency profile of $\mathbf{C}_1$ (diagonal of $F_{\mathbf{C}_1}$ where $i = j$ in (4.52)) is not smooth. Indeed, diagonal elements of $F_{\mathbf{C}_1}$ can be written as a weighted sum of squared eigenvalues, which again is weighted by $1/(1+\mathbf{d}_k)$. The L2-norm of an eigenvector is 1, so sum of squared eigenvectors elements has to be 1 as well. But in this case, since $1/(1+\mathbf{d}_k)$ are not necessarily constant over all the dimensions of eigenvectors, the diagonal of the matrix may have some perturbations. This point constitutes another explanation on the fact that the GCN standard frequency profile is not smooth.

On the other hand, under the assumption that node degrees distribution is uniform, we can derive the following approximation:

$$p \approx \overline{d} = \frac{1}{n} \sum_{k=1}^{n} \mathbf{d}_k \tag{4.53}$$

We can then write an approximation of the GCN frequency profile as a function of the average node degree by replacing p with $\overline{d}$ in (4.47) and obtain the final approximation:

$$F_{GCN}(\boldsymbol{\lambda}) \approx 1 - \frac{\overline{d}}{\overline{d}+1}\boldsymbol{\lambda} \tag{4.54}$$

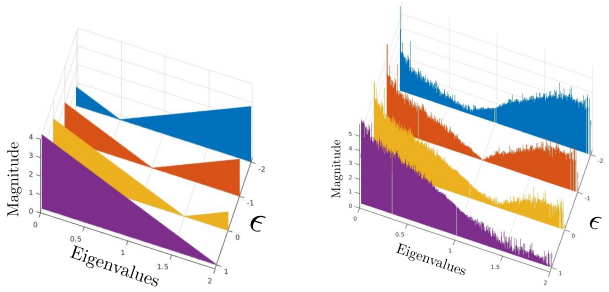
We can theoretically show the cut-off frequency where GCN kernel's frequency profile reach 0 by:

$$\lambda_{\text{cut}} \approx \frac{\overline{d}+1}{\overline{d}} \tag{4.55}$$

Theoretically, if all nodes degree are different, standard frequency profile will not be smooth and will include some perturbations. In addition, full frequency profile will be composed of non-zero components.

Analyzing experimentally the behavior of GCN [43] in the spectral domain first implies to compute the convolution kernel as given in (3.18). Then, the spectral representation of the obtained convolution matrix can be back-calculated using Corollary 1.1. This result leads to the frequency profiles illustrated in Figure 4.4 for the three different graphs. The three standard frequency profiles have almost the same low-pass filter shape corresponding to a function composed of a decreasing part on the three first quarters of the eigenvalues range, followed by an increasing part on the remaining range. This observation is coherent with the theoretical analysis. Hence, kernels used in GCN are transferable across the three graphs at hand. In Figure 4.4, the cut-off frequency of the 1-D linear circular graph is exactly 1.5, while it is about 1.35 for Citeseer. This observation can be explained by the fact that when considering a 1-D linear circular graph, all nodes have a degree equal to 2, hence $\lambda_{\text{cut}} = 1.5$. Since the average node degree in Citeseer is 2.77, therefore $\lambda_{\text{cut}} \approx 1.36$.

Concerning the full frequency profiles, there is no contribution outside the diagonal for the regular line graph (Figure 4.4 b). Conversely, some off-diagonal values are not null for Citeseer and Cora. Again, this observation confirms the theoretical analysis.



(a) GIN on 1D (b) GIN on CiteSeer

Figure 4.5: Frequency profiles of GIN on 1D and CiteSeer graph with $\epsilon = 1, 0, -1, -2$

Since GCN frequency profile does not cover the whole spectrum, such an approach is not able to learn relations that can be represented by high-pass or band-pass filtering. Hence, even though it gives very good results on a single graph node classification problem in [43], it may fail for problems where discriminant information lies in particular frequency bands. Therefore, such an approach can be considered as problem specific.

GIN

Graph Isomorphism Network (GIN) defined in [87] has single convolution support defined as follows;

$$C_{GIN} = A + (1 + \epsilon)I \tag{4.56}$$

where ϵ is trainable parameter which makes the support trainable (GIN-ϵ) and classified as spatial-designed trainable-support graph convolution. However, another version named GIN-0 also defined in the same paper where they assigned $\epsilon = 0$ as fixed value in [87], makes $C_{GIN} = A + I$, thus the convolution becomes fixed-support and identical with Vanilla GNN defined in Section 3.4.

Proposition 1. *$C_{GIN} = A + (1+\epsilon)I$ frequency response is $\Phi_{GIN}(\boldsymbol{\lambda}) = p(\frac{1+\epsilon}{p} + 1 - \boldsymbol{\lambda})$ for regular graphs, whose node degrees are p.*

GIN has attracted a lot of interests from the community, mostly because of its simple convolution mechanism. It has a single convolution support and its theoretical frequency response is given in the following theorem :

Theorem 6. *The theoretical frequency response of GIN support can be approximated as*

$$\Phi(\boldsymbol{\lambda}) \approx \overline{p}\left(\frac{1+\epsilon}{\overline{p}} + \mathbf{1} - \boldsymbol{\lambda}\right) \tag{4.57}$$

where ϵ is a trainable scalar. In addition to its simple convolution mechanism, another interesting point with GIN is the fact that it has been proven to be as discriminative as the WL-test.

Proof. When all node degrees are p, it yields $D = pI$, $L = I - A/p$ or $A = pI - pL$. When we substitute new equations of A and D into C_{GIN}, we get

$$C_{GIN} = (p + 1 + \epsilon)I - pL. \tag{4.58}$$

It should meet the following condition if the given frequency response is true.

$$(p + 1 + \epsilon)I - pL = U\text{diag}(p + \epsilon + \mathbf{1} - p\boldsymbol{\lambda})U^\top \tag{4.59}$$

We can obtain the following equation by $p + \epsilon + \mathbf{1} = (p + 1 + \epsilon)I$ substitution.

$$(p + 1 + \epsilon)I - pL = (p + 1 + \epsilon)UIU^\top - pU\text{diag}(\boldsymbol{\lambda})U^\top \tag{4.60}$$

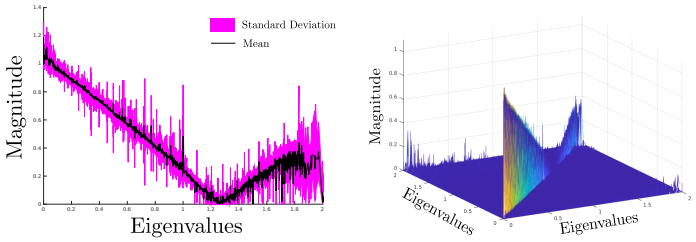
Since $UIU^\top = I$ and $U\text{diag}(\boldsymbol{\lambda})U^\top = L$, the condition in (4.59) is satisfied. □

Proof. Even in regular graph, the theoretical frequency response of GIN is not identical and it depends on the node degree, thus it is not spectral-designed. In addition, we can see the GIN convolution support as the sum of two matrices where the second one $(1+\epsilon)I$ is diagonalizable by eigenvectors U of graph Laplacian by $\Phi = \mathbf{1} + \epsilon$. Thus the second part of GIN support is spectral. However, the first part, which is adjacency A, cannot be diagonalizable by U. Since the convolution support is not diagonalizable, we cannot write exact frequency response of GIN convolution but just an approximation of Proposition 1, assuming by the average node degree of the graph is $\overline{p}$ in (4.61).

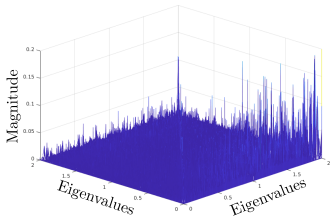
$$\Phi_{GIN}(\boldsymbol{\lambda}) \approx \overline{p}\left(\frac{1+\epsilon}{\overline{p}} + \mathbf{1} - \boldsymbol{\lambda}\right) \tag{4.61}$$

□

When the given graph is a regular graph where each node degree is the same (2 for 1D graph case), theoretical frequency responses become certain as seen in Figure 4.5(a). When $\epsilon = 2$, 1D graph's ($\overline{p} = 2$) frequency responses of GIN the same than GCN within a scaling factor. However in realistic graphs, both GIN and GCN are not spectral-designed, their frequency responses differ for



(a) Standard frequency profile (b) Mean of full frequency profile



(c) Standard deviation of full frequency profile

Figure 4.6: Frequency profiles of randomly generated 250 GAT convolutions using Cora graph.

different graphs. As Theorem 5 and Theorem 6 say, GCN's and GIN's frequency responses depend on the average node degree. In opposite to GCN, $\overline{p}$ acts here as scaling factor on GIN's frequency response.

In order to create some variations between low-pass to high-pass, having trainable parameter in GIN's convolution support seems advantageous. But, since it is not spectral-designed, there is no guaranty that it works the same way for extremely diverse graph datasets. Besides, its low-pass shape (where ϵ is high) is a linearly decreasing function, so it is not strong low-pass where generally natural graph problems needs. Using more stacked layer may be a solution. In addition, this convolution cannot focus on some certain bands if the problem needs.

GAT

Graph Attention Networks (GATs) rely on trainable convolutions kernels [79] (See chapter 3.4). For this reason, frequency profiles cannot be directly computed similarly to GCN or ChebNet ones. Thus, instead of back-calculating the kernels, we perform simulations and evaluate the potential kernels of

attention mechanism for given graphs. Hence, we show the frequency profiles of those simulated potential kernels.

In [79], 8 different attention heads are used. Assuming that each attention head matrix is a convolution kernel, multi-attention systems can be seen as multi-kernel convolutions. The difference is that convolution kernels are not a priori defined but are functions of node feature vectors and trainable parameters $\mathbf{a}$ and $\mathbf{W}$; see (3.11). To show the potential output of GATs on the Cora graph (1433 features for each node), we produce 250 random pairs of $\mathbf{W} \in R^{1433\times 8}$ and $\mathbf{a} \in R^{16\times 1}$, which correspond to the convolution kernels trained by GATs. The σ function in (3.11) is a `LeakyReLU` activation with a 0.2 negative slope as in [79].

The mean and standard deviation of the frequency profiles for these simulated GAT kernels are shown in Figure 4.6. As one can see, the mean standard frequency profile has a similar shape as those of GCN (Figure 4.4). However, variations on the frequency profile induce more variations on output signal when compared to GCN.

The full frequency profile is not symmetric. According to Figure 4.6, variations are mostly on the right side of the diagonal in the full frequency profile. This is related to the fact that these convolution kernels are not symmetric. However, the variation on frequency profile might not be sufficient in problems that need some specific band-pass filters.

In this section, we proved that most of state-of-the-art GNN models are limited to low-pass filtering, despite presenting great results in many applicative problems. In order to support our point, we provide in following section 4.4 an experimental setup that proves this fact.

4.4 Why Spectral properties of GNNs are important ?

In this chapter, the fact that most of models of GNNs are limited to low pass filtering has been claimed multiple times. In this section, we aim to show the reader the importance of spectral properties. To this end, we firstly wonder what kind of filters the different GNNs are able to learn. In a second part, we propose an analysis of the GNNs ability to classify graph according to their signal.

4.4.1 Which Filters Can the GNN Models Learn?

In this section, we seek to measure of the ability of GNN models to learn some specific filtering process. This study is very important in order to understand the learning capability of existing GNN models. Since the problem may need various types of filtering, the best GNN model has to be able to learn any kind of filtering.

For this purpose, we conduct an empirical analysis on a real image with resolution of 100×100 and its corresponding 2D regular 4-neighborhood grid graph. The input of the GNN is the adjacency matrix of size 10000×10000 and the pixel intensities given in a 10000-length vector. We create three different spectral filters that correspond to low-pass, band-pass and high-pass effects and apply these filters to the given input image. Our selection of spectral filters are defined by $\Phi_1(\rho) = \exp(-100\rho^2)$, $\Phi_2(\rho) = \exp(-1000(\rho - 0.5)^2)$ and $\Phi_3(\rho) = 1 - \exp(-10\rho^2)$ for low-pass, band-pass and high-pass filters respectively, where $\rho^2 = u^2 + v^2$ and u and v are the normalized frequencies on each direction for a given image resolution. Used input image and its filtering results can be found in Figure 4.7.

Since we do not use pixel positions, neither as node feature nor as edge feature, we create these spectral filters to be learned in a directional agnostic way. Therefore, the problem can be viewed as a single graph node regression problem, where we train the GNN models to minimize the square error between its output and targeted filtered image.

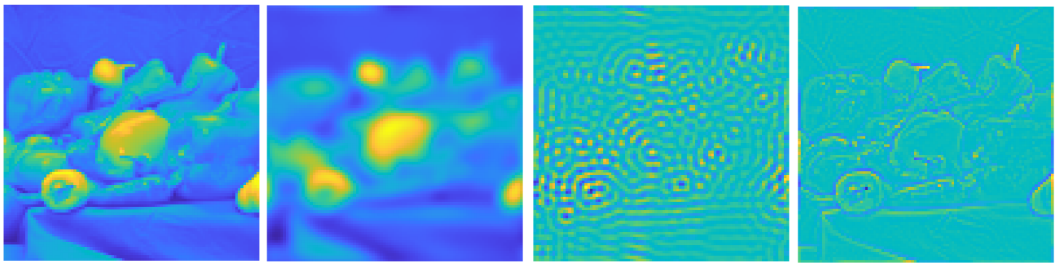


Figure 4.7: Input image, and its filtering results by Φ_1, Φ_2 and Φ_3 respectively

In order to assess ChebNet, GCN, GIN and GAT, we use a 3-layer GNN architecture whose input is a one-length feature (intensity of the pixel) and the number of neurons in hidden layers is respectively 32, 64 and 64; the output layer is an MLP that projects the final node representation onto the single output for each node. We used roughly 30k trainable parameter in ChebNet with 5 supports. For the other methods, we tuned the hidden neuron numbers in order to be sure that they have a similar number of trainable parameters.

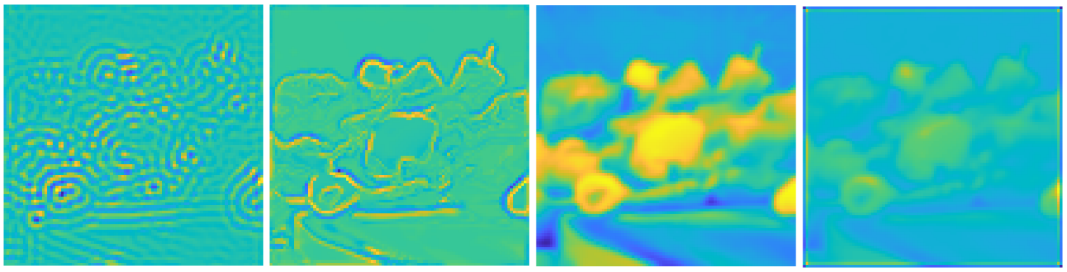


Figure 4.8: The output of GNNs trained with band-pass task. Images are taken from ChebNet, GIN, GAT and GCN respectively.

Table 4.1: Sum of squared errors. All models have roughly 30k trainable parameters.

Prediction Target	GCN	GIN	GAT	ChebNet
Low-pass filter (Φ_1)	15.55	11.01	10.50	3.44
Band-pass filter (Φ_2)	79.72	63.24	79.68	17.30
High-pass filter (Φ_3)	29.51	14.27	29.10	2.04

Since the aim is not assessing the generalization performance, we do not use any regularization or dropout to address overfitting, but simply force the GNN to learn the input-output relation. We keep the iterations till there is no improvement for consecutive 100 iterations or maximum 3000 iterations.

Table 4.1 gives the sum of squared errors between target and the output of the trained model. One can see that ChebNet constantly outperformed GCN, GIN and GAT for all tasks. For learning low-pass filtering, the rest of the models did better compared to the high-pass and band-pass tasks. That is the fact that GCN, GIN and GAT have the ability to act as low-pass filters. In addition to do better on the low-pass task, GIN also did relatively better on the high-pass task as well. It is obvious that GIN can work as high pass if the ϵ parameter is selected negative (see Theorem 6). It turns out that the trained values of ϵ in GIN for each layer are -5.27, -2.21 and -0.47 for the high-pass task.

Thanks to the spectral-designed convolution supports in ChebNet, it could learn high-pass and low-pass tasks very well. However for band-pass tasks, even though it is the best in this category too, it still has large errors compared to the high-pass and low-pass tasks. This is due to the fact that the selected band-pass filter is very narrow, because the coefficient -1000 in the formulation of

Table 4.2: ChebNet's sum of squared errors on band-pass tasks with respect to S kernels and L stacked layers. All models have roughly 30k trainable parameters.

	S=2	S=3	S=5	S=7	S=10
L=1	65.06	56.25	47.12	39.59	28.15
L=2	55.85	45.96	26.81	18.92	13.33
L=3	50.13	36.60	17.30	9.84	7.32
L=4	44.88	24.89	11.90	8.91	6.96

Φ_2 makes the used ChebNet (with 5 convolution supports and 3 layers) unable to adapt this stiff (not smooth) filter function. Moreover, since ChebNet has no band specific convolutions, band specific output can be produced if the number of kernels increases (going wider) and/or the model goes deeper. To clarify this point, we conducted another test for band-pass task on ChebNet to show the effect of going deeper in the model and going wider (increase the convolution support) while keeping the trainable parameters fixed. These results are given in Table 4.2.

According to Table 4.2, the ability of ChebNet to learn the given frequency response becomes better with respect to the number of convolution supports and number of layers. However, this result is not surprising where it is proved that any frequency response can be written by a weighed sum of enough number of Chebyshev polynomials ([32]). When we train the ChebNet, it just finds these coefficients to create the target frequency response by minimizing the error. However, the interesting point is the incapability of GCN, GIN and GAT methods to even create reasonable approximations of these targeted filter effects. For instance, it can be seen in Figure 4.8 that ChebNet performed well to produce the desired band-pass output. However, GAT and GCN produce just a different kind of low-pass filtering result instead of band-pass, while GIN at least can find edges (high-pass component) thank to its trainable parameter ϵ. We also tested the deeper network for GCN, GAT and GIN as well and have not seen any significant improvement when we use deeper network.

4.4.2 Can GNN classify graphs according to its signal ?

In this section, we measure the generalization ability of GNN for graph classification problem where graph classes depend on the signal that the graphs carry. We generate 5000 images of 100×100 pixels composed of

random generated frequency patterns obtained by a sinusoidal function with a frequency in the range [1-5]. We labelled the image as negative if the pattern's frequency is in the ranges [2-2.5] or [4-4.5]. The rest of the frequency patterns are labeled as belonging to the positive class. Then, we randomly rotate and translate the image pattern, add white noise (with std=0.2) and normalize each image independently. From each image, we randomly sample 200 points in the 100×100 image plane and we divide the image into 200 regions by watershed algorithm [58], where each sampled point is the marker. From this preprocessing, we generate 5000 graphs, each graph having 200 nodes. Each node corresponds to a watershed region in the image, and if the two regions have intersection on the image plane, we assume these two nodes are connected by an edge in the graph. We set the average intensity value in each region as a 1-length node feature. Even though we know the region center position, we do not use it in order to make the problem harder. Sampled generated image, randomly selected points and their watershed regions, and the graph can be found in Figure 4.9 for a 30-node illustration.

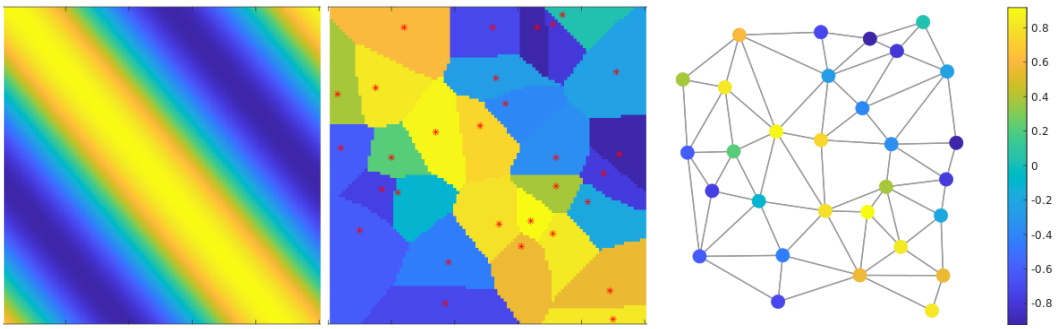


Figure 4.9: Sample graph in Band-Pass graph dataset. Random rotated and translated image pattern with frequency of 1, random sampled points and their watershed regions, and graph represent the connected region and average region intensity value respectively.

We divided the dataset into train/valid/test subsets, with respectively 3000, 1000 and 1000 graphs. We resampled the same number of positive and negative examples, such that the dataset is balanced. We used 3 layers of GNN followed by a mean readout layer and finally two fully connected layers which have 10 and 1 neuron respectively. Since the problem is a binary graph classification problem, we used binary cross entropy loss and no regularization. We roughly use 30K parameters in each model. The dropout ratio has been applied to all GNN layer's inputs and optimized with respect to the validation set performance.

Table 4.3: Test set accuracy and binary cross entropy loss.

	MLP	GCN	GIN	GAT	ChebNet
Accuracy	50	77.90	87.60	85.30	98.2
Loss	0.69	0.454	0.273	0.324	0.062

The results are found on Table 4.3. Since the node distributions are all the same in the graphs (because the graph nodes were independently normalized), MLP cannot do better than a random classifier. GCN does not perform well, probably because of its low-pass nature. Since GAT and GIN are better than GCN according to the spectral ability, they got a better accuracy than GCN. Finally, ChebNet with 5 convolution supports clearly outperforms the rest of them with a huge margin. These results show that models able to catch a particular band of frequencies obtain the best results, whereas only low-pass based methods like GCN perform only slightly better than MLP. Therefore, this toy example confirms our theoretical analysis.

To conclude, we have shown that if the model is able to perform different filtering operation, it can classify the graphs according to frequency of its signal.

4.5 Conclusion

This chapter has claimed and proved in section 4.2 that spectral GNN are a special case of spatial GNNs. Even though this fact is commonly accepted, no proof has formally been proposed in our knowledge. This chapter thus fills this gap.

This proof has enabled us to provide a spectral analysis of some of the most influential methods, namely : ChebNet, CayleyNet, GCN, GIN and GAT. This analysis as highlighted one common limitation of GNNs : they are not able to produce specific band-pass filters, and most of them are generally limited to low-pass filters. Another limitation that can be pointed out is the fact that, despite few propositions in the spatial domain, edge attributes are rarely mentioned.

We also provided an analysis to show the importance of spectral properties. This was done by firstly presenting the type of filters that GNNs can learn. Table 4.4 present a summary of the analyzed models. In a second time, we presented a toy problem that requires band-pass filtering, highlighting once

Table 4.4: Summary of the studied GNN models.

	Design	Support Type	Convolution Matrix	Frequency Response
MLP	Spectral	Fixed	$C = I$	$\Phi(\boldsymbol{\lambda}) = \mathbf{1}$
GCN	Spatial	Fixed	$C = \tilde{D}^{-0.5}\tilde{A}\tilde{D}^{-0.5}$	$\Phi(\boldsymbol{\lambda}) \approx \mathbf{1} - \boldsymbol{\lambda}\overline{p}/(\overline{p}+1)$
GIN	Spatial	Trainable	$C = A + (1+\epsilon)I$	$\Phi(\boldsymbol{\lambda}) \approx \overline{p}\left(\frac{1+\epsilon}{\overline{p}} + \mathbf{1} - \boldsymbol{\lambda}\right)$
GAT	Spatial	Trainable	$C^{(s)}_{v,u} = e_{v,u} / \sum_{k \in \tilde{\mathcal{N}}(v)} e_{v,k}$	NA
CayleyNet[a]	Spectral	Trainable	$C^{(1)} = I$ $C^{(2r)} = Re(\rho(hL)^r)$ $C^{(2r+1)} = Re(\mathbf{i}\rho(hL)^r)$	$\Phi_1(\boldsymbol{\lambda}) = \mathbf{1}$ $\Phi_{2r}(\boldsymbol{\lambda}) = \cos(r\theta(h\boldsymbol{\lambda}))$ $\Phi_{2r+1}(\boldsymbol{\lambda}) = -\sin(r\theta(h\boldsymbol{\lambda}))$
ChebNet	Spectral	Fixed	$C^{(1)} = I$ $C^{(2)} = 2L/\lambda_{\max} - I$ $C^{(s)} = 2C^{(2)}C^{(s-1)} - C^{(s-2)}$	$\Phi_1(\boldsymbol{\lambda}) = \mathbf{1}$ $\Phi_2(\boldsymbol{\lambda}) = 2\boldsymbol{\lambda}/\lambda_{\max} - \mathbf{1}$ $\Phi_s(\boldsymbol{\lambda}) = 2\Phi_2(\boldsymbol{\lambda})\Phi_{s-1}(\boldsymbol{\lambda}) - \Phi_{s-2}(\boldsymbol{\lambda})$

[a] $\rho(x) = (x - \mathbf{i}I)/(x + \mathbf{i}I)$

again the unability of most GNNs to produce band-pass filtering. This gives us an hint on the direction to take in order to improve GNNs.

In the next chapters, we will propose two methods to tackle on one hand the filtering limitation, and on the other hand the edge attribute limitation.

Chapter 5

Depthwise Separable Graph Convolution Network

5.1 Introduction

Having defined and explored the Graph Neural Network paradigm in chapter 3, the chapter ended 4 with two limitations for Graph Neural Network. The first is that most existing GNNs are limited to loss-pass filtering.

Indeed, either our theoretical or experimental analysis concluded that the frequency responses of GIN, GAT and GCN were limited to low-pass filtering. On the other side, spectral designed methods such as ChebNet and CayleyNet produce convolutions filters that, once combined, are able to cover the whole spectrum. They are however unable to produce specific band-pass filters. Another drawback of those methods is that they require a large amount of convolution filters to cover the whole spectrum. The increase of convolution filters thus also increases the number of parameters required by the model.

In this chapter, we propose a method that allows to design enough convolution kernels to cover as much as possible the frequency spectrum, while containing the number of parameters thanks to the Depthwise Separable Convolution framework. The resulting method is called Depthwise Separable Graph Convolutional Network (DSGCN).

This DSGCN method is presented in 5.2, and is then evaluated in 5.3. The evaluation is conducted on both transductive and inductive task, obtaining state-of-the-art results. Moreover, a limited version of the proposed approach is also evaluated on the transductive task. This version is limited to low-pass filtering, and obtain equivalent results to popular GNNs, proving once again that those GNNs are limited to low-pass filtering.

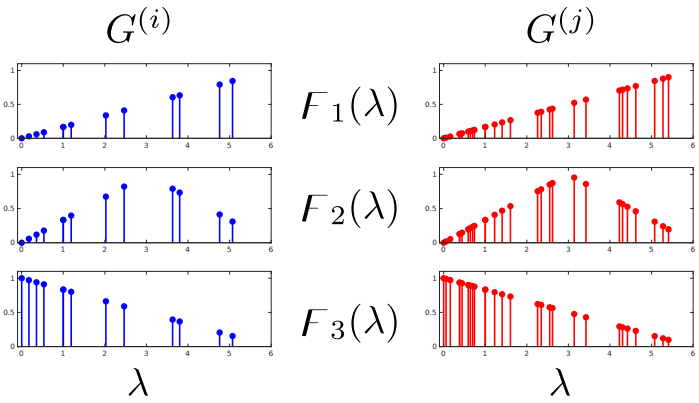


Figure 5.1: Three designed convolution kernel frequency profiles as a function of graph eigenvalues ($\boldsymbol{\lambda}$) of two sample graphs $G^{(i)}$ and $G^{(j)}$ by $F_1(\boldsymbol{\lambda}) = \frac{\boldsymbol{\lambda}}{6}, F_2(\boldsymbol{\lambda}) = 1 - \frac{|\boldsymbol{\lambda}-3|}{3}$ and $F_3(\boldsymbol{\lambda}) = 1 - \frac{\boldsymbol{\lambda}}{6}$. There are three shared coefficients. Each coefficient encodes the contribution of corresponding frequency profiles. First row refers mostly to high frequencies, middle row to middle frequencies and last row to low frequencies.

5.2 Depthwise Separable Graph Convolutions

Instead of designing the spatial convolution kernels $C^{(s)}$ of Eq. (4.2) by functions of graph adjacency and/or graph Laplacian, we propose in this section to use S convolution kernels that have custom-designed standard frequency profiles. These designed frequency profiles are functions of eigenvalues, such as $[F_1(\boldsymbol{\lambda}), \ldots, F_S(\boldsymbol{\lambda})]$ where $F_s : \mathbb{R}_+^{|\mathcal{V}|} \rightarrow \mathbb{R}_+^{|\mathcal{V}|}$. In this proposal, the number of kernels and their frequency profiles are hyperparameters, and should be defined initially, which is a drawback of the method. Given these frequency profiles, we can back-calculate corresponding spatial convolution matrices $\mathbf{C}^{(s)}$ for $s = 1 \ldots S$ using Eq. (4.3) in Theorem 1. Then these spatial convolution matrices can be used in any GNN following the scheme defined in equation 3.7.

To obtain problem-agnostic graph convolutions, the set of all designed convolution's frequency profiles has to cover most of the spectrum range and each kernel frequency profile must focus on some particular frequency range. As a didactic example, we show in Figure 5.1 an example of desired spectral convolutions frequency profile for $S = 3$ and its application on two different graphs.

In order to figure out arbitrary relations of input-output pairs, multiple convolution kernels have to be efficiently designed. However, increasing the number S of convolution kernels increases the number of trainable parameters linearly. Hence, the total number of multi-support ConvGNN parameters is

given by $\sum_{l=0}^{L} f_l f_{l+1}$ where L is the number of layers and f_l is the feature length of the l-th layer. To overcome this issue, we propose Depthwise Separable Graph Convolution Network (DSGCN).

5.2.1 Depthwise Separable Convolution

Our method is an adaptation of Depthwise Separable Convolution (DSC) applied on graphs. In this section, we present the original DSC method applied on images. Color images are generally encoded by $L \times H \times 3$ tensors. Convolution operator aggregates the information of neighborhood pixels. Therefore, 3 dimension (3D) convolution kernels computing a $N \times N$ output are generally used. In order to produce multiple output channels, multiple 3D convolution kernels have to be designed. The number of parameters thus depends on the number of input channels (f_i) and the number of output channels (f_o). For example, the number of parameters for a 3×3 convolution kernel is equals to $f_o * (3 * 3 * f_i)$ as presented in Figure 5.2. In this figure, 3 filters of shape 3×3 are applied on an image with 3 input channels. This leads to $3 * (3 * 3 * 3) = 81$ parameters. Traditional CNN architectures such as VGG16 [75] use layers with a number of output channels between 64 and 256. Such an architecture leads to a number of parameters between 36 864 and 589 824 for each layer. Multiplying layers of convolutions and the number of output channels will thus increase substantially the number of parameters.

In order to reduce the number of parameters, the Depthwise Separable Convolution (DSC) framework has already been used in computer vision problems ([20, 71]). DSC consists in two steps. First is the separation step which consists in considering each of the channel as an independent image. Then, 2D convolution kernel can be applied over each of those independent images to produce different outputs. However, the hypothesis that each channel is independent is generally not true considering colored images. Indeed, it's the combination of the three RGB features of each pixel which constitutes the color information. Second, to overcome this drawback, the depthwise step consists in applying convolution kernel along the channel dimension which can be seen as the depth dimension of an image. This step allows to combine the different channels for each pixel.

Formally, in this framework, a different 2D convolution kernel is applied over each input channel independently. Thus, f_i 2D convolution kernels are computed, where f_i is the number of input channels, which produces f_i intermediate channels. This step requires to apply each convolution kernel f_i times, and thus lead to $f_i * 3 * 3$ parameters, for a 3×3 convolution kernel. Then, a $1 \times 1 \times f_i$ convolution kernel is applied on the concatenation of the

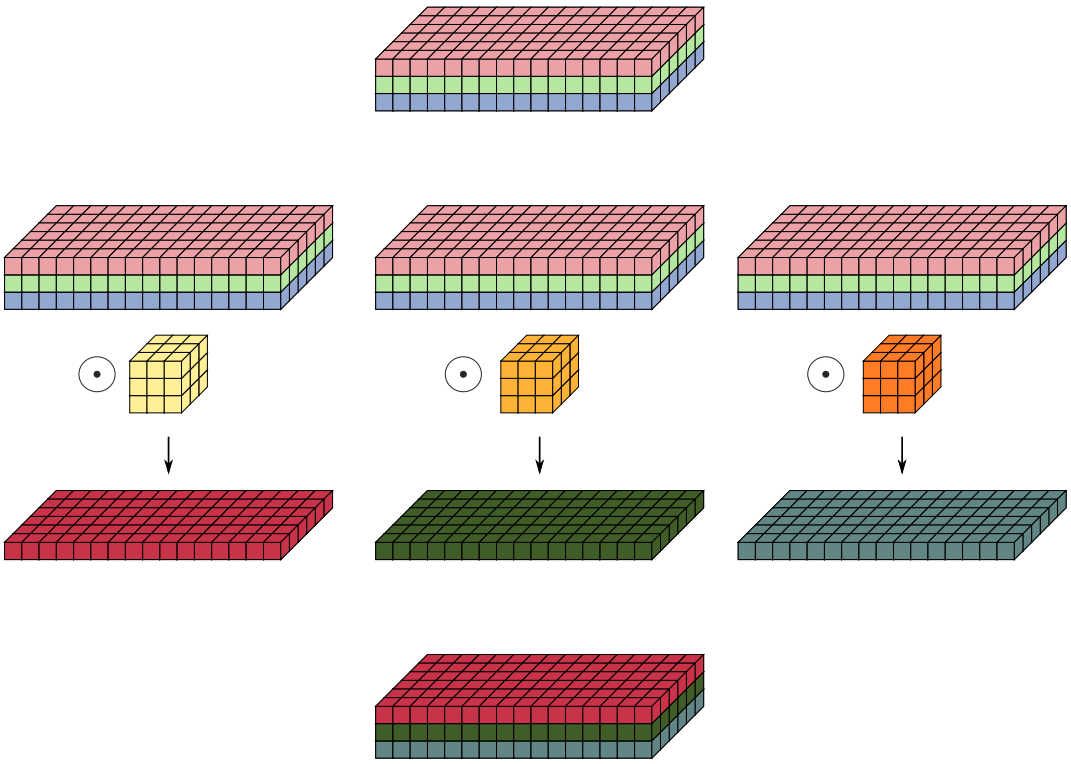


Figure 5.2: Standard convolution applied on a 3 channel image to produce a 3 channel output. This requires $3 * (3 * 3 * 3) = 81$ parameters.

intermediate channels to produce one output channel. This step corresponds to the depthwise convolution as described before. To produce f_o output channels, f_o depthwise convolution kernels applied over the intermediate channels have to be designed. Since each depthwise convolution kernel requires f_i parameters, this step leads to $f_o * f_i$ parameters. The combination of this two steps leads to a number of parameters of $f_i * (3*3) + f_o * f_i$. On condition that $f_o > 1$, the number of parameters is thus reduced with Depthwise Separable Convolution compared to standard convolution. Transposing Depthwise Separable Convolution to VGG16 leads to a number of parameters between 4672 and 67 840

5.2.2 Depthwise Separable Convolution on Graph

We propose to apply the principle of Depthwise Separable Convolution to Graph Neural Networks in order to produce multiple convolution kernels whose frequency profiles cover most of the spectrum range with a lower number of parameters than usually required. To the best of our knowledge, applying

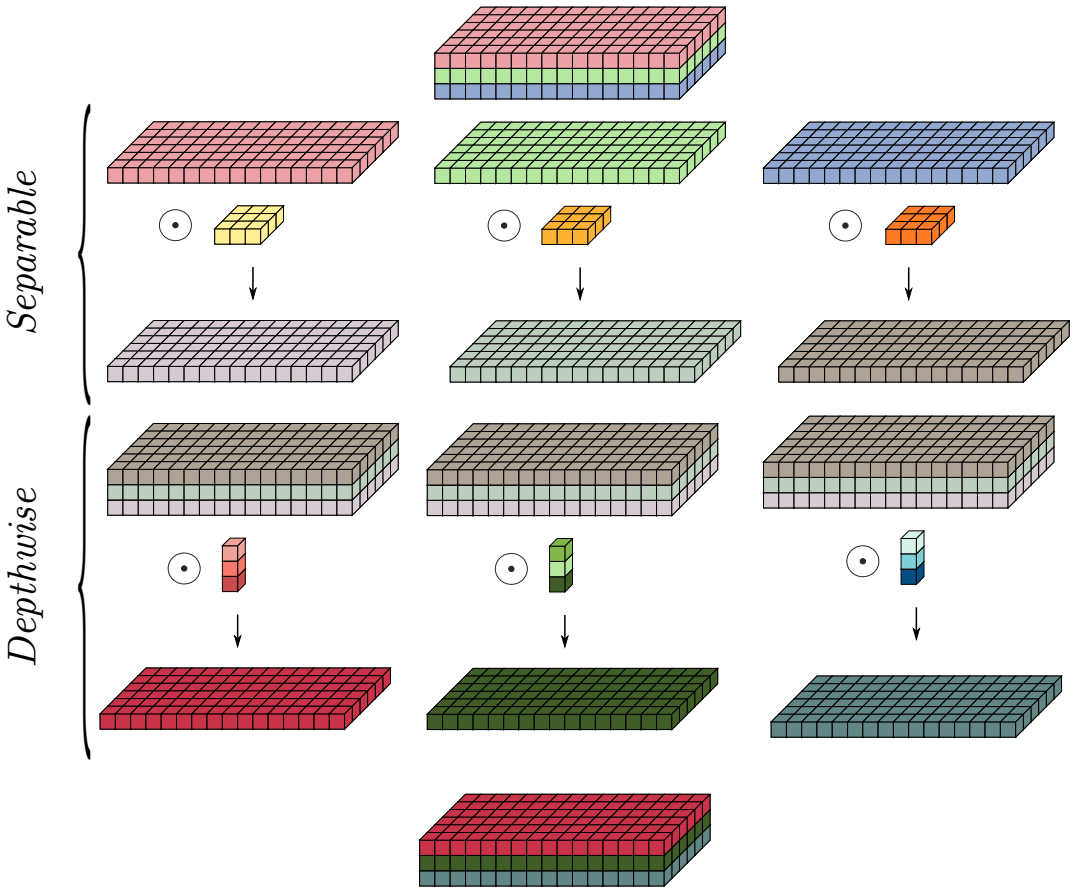


Figure 5.3: Depthwise Separable convolution applied to a 3 channel image to produce a 3 channel output. This require $3 * 3 + 3 * 9 = 36$ parameters

Depthwise Separable Convolution to graph neural networks has never been proposed in the literature.

One way to make the analogy between graph and images is to consider pixel as nodes in a graph. In this case, RGB values of a node are considered as its features and pixels convoluted together are considered as neighbors (central pixel of the convolution is a neighbor of each other pixel). From this point of view, separable convolutions consist in considering independently each feature of the node. One way to apply separable convolution and thus to reduce the number of parameters would be to filter each feature independently. Mathematically, this can be applied by designing one vector of parameters following Equation 5.1 :

$$\mathbf{H}^{(l+1)} = \sigma(\sum_{s=1}^{S}(\mathbb{1}\mathbf{w}^{(s,l)}) \circ (\mathbf{C}^{(s)}\mathbf{H}^{(l)}) + \mathbf{b}), \tag{5.1}$$

here $\mathbf{H}^{(l)} \in \mathbb{R}^{|\mathcal{V}| \times f_l}$, f_l being the dimension of the hidden state vector in layer l, $\mathbf{w}^{(s,l)} \in \mathbb{R}^{1 \times f_l}$ is the parameters vector, $\mathbf{C} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ and $\mathbf{b} \in \mathbb{R}^{f_{l+1}}$. In this equation, $\circ$ is the Hadamard product. The Separable Graph Convolution thus requires $S * f_l$ parameters per layer. This is an important reduction of the number of parameters compared to standard GNN, of which we remind the definition with Equation 5.2 and that would require $S * f_l * f_{l+1}$ parameters.

$$\mathbf{H}^{(l+1)} = \sigma((\sum_{i=0}^{1} \mathbf{C}^{(i)}\mathbf{H}^{(l)}\mathbf{W}^{(l,i)}) + \mathbf{b}), \tag{5.2}$$

where $\mathbf{H}^{(l)} \in \mathbb{R}^{|\mathcal{V}| \times f_l}$, f_l being the dimension of the state vector in layer l, $\mathbf{W} \in \mathbb{R}^{f_l \times f_{l+1}}$, $\mathbf{C} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ and $\mathbf{b} \in \mathbb{R}^{f_{l+1}}$.

However, once again, the hypothesis that features are independent doesn't hold. In order to take this into account, we propose to also add a Depthwise step. This step consists in computing a new feature depending on the features independently computed. We can then compute f_{l+1} features which leads to a single matrix of parameters of shape $f_l \times f_{l+1}$. Applying both steps of Separable and Depthwise thus leads to Equation 5.3 :

$$\mathbf{H}^{(l+1)} = \sigma\left(\left(\sum_{s=1}^{S}(\mathbb{1}\mathbf{w}^{(s,l)}) \circ (\mathbf{C}^{(s)}\mathbf{H}^{(l)})\right)\mathbf{W}^{(l)}\right). \tag{5.3}$$

In this formula, note that there is only one trainable matrix $\mathbf{W}^{(l)}$ in each layer. Other trainable variables $\mathbf{w}^{(s,l)} \in \mathbb{R}^{1 \times f_l}$ encode feature contributions for each convolution kernel and layer. The number of trainable parameters for this case becomes $S * f_l + f_l * f_{l+1}$ per layer. Compared to standard GNNs, whose number of parameters per layer was $S * f_l * f_{l+1}$, this is still an important reduction of the number of parameters, on the condition that $S + f_{l+1} < S * f_{l+1}$. Previously, adding a new kernel increases the number of parameters by $\sum_{l=0}^{L} f_l f_{l+1}$. Using separable convolutions, this number is only increased by $\sum_{l=0}^{L} f_l$. This modification is particularly interesting when the number of features in hidden layers is high. On the other hand, the variability of the model also decreases. If the data has a smaller number of features, using this approach might not be optimal. Detailed illustration of the proposed Depthwise Separable Graph Convolution process is presented in Figure 5.4. The following section presents the experimental evaluation of our method DSGCN.

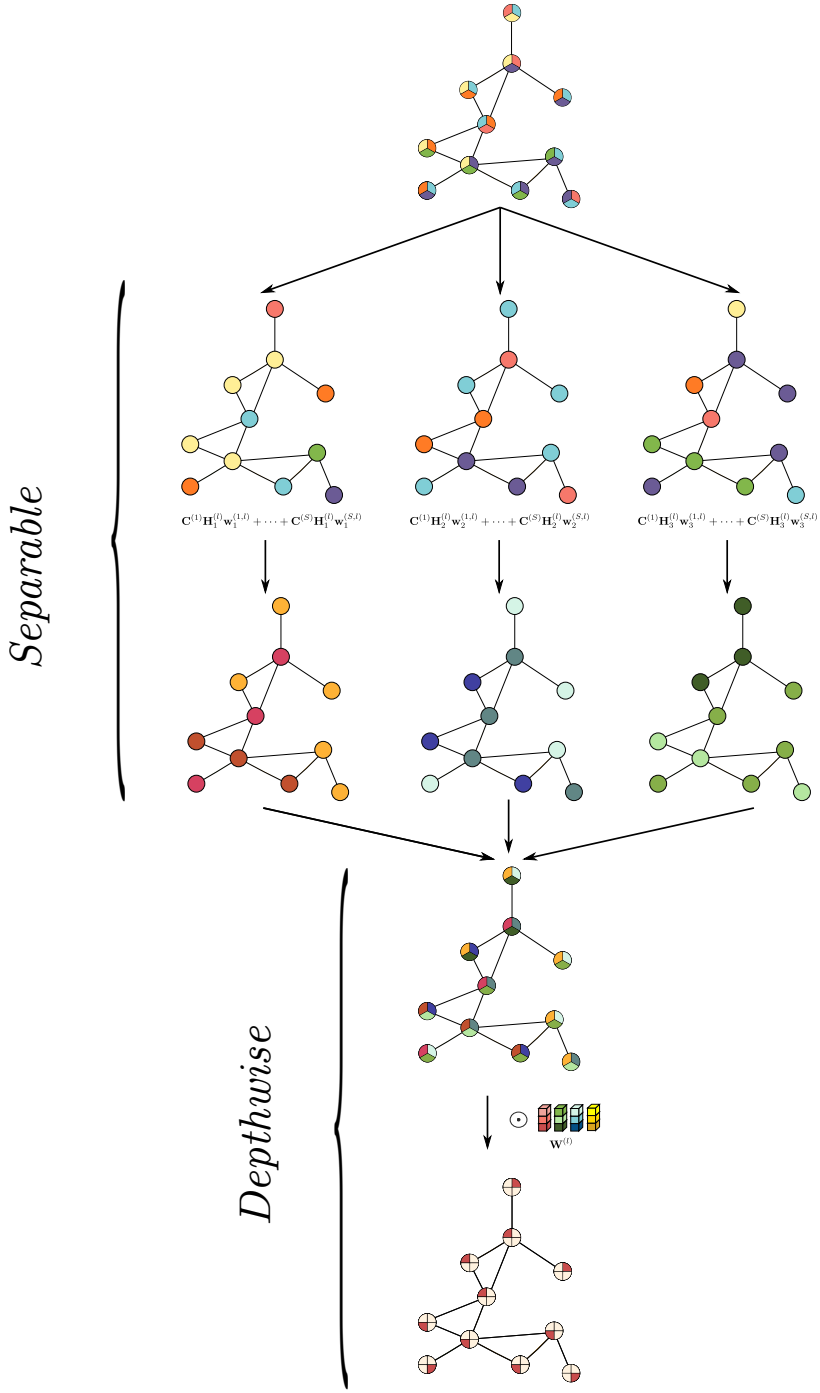


Figure 5.4: Detailed schematic of Depthwise Separable Graph Convolution Layer. Each input feature is computed independently in the Separable step. Then, 4 new features are computed through the Depthwise step.

Table 5.1: Summary of the transductive datasets used in our experiments. Each dataset consists of one single graph

	Cora	Citeseer	PubMed
# Nodes	2708	3327	19717
# Edges	5429	4732	44338
# Features	1433	3703	500
# Classes	7	6	3
# Training Nodes	140	120	60
# Validation Nodes	500	500	500
# Test Nodes	1000	1000	1000

5.3 Experimental evaluation of DSGCN

In this section, we describe the experiments carried out to evaluate the proposed approach on both transductive and inductive problems. In the first case, we target a single graph node classification task while in the second case, both multi-graph node classification task and entire graph classification task are considered. For all the experiments, we compare our algorithm to state-of-the-art approaches.

5.3.1 Transductive Learning Problem

Datasets Experiments on transductive problems were led on the three datasets summarized in Table 5.1. These datasets are well-known paper citation graphs. Each node corresponds to a paper. If one paper cites another one, there is an unlabeled and undirected edge between the corresponding nodes. Binary features on the nodes indicate the presence of specific keywords in the corresponding paper. The task is to attribute a class to each node (i.e., paper) of the graph using for training the graph itself and a very limited number of labeled nodes. Labeled data ratio is 5.1%, 3.6% and 0.3% for Cora, Citeseer and PubMed respectively. We use predefined train, validation and test sets as defined in [88] and follow the test procedure of [43, 79] for fair comparisons.

Models To evaluate the performance of convolutions designed in the spectral domain independently from the architecture design, a single hidden layer is used for all models, as in [43] for GCN. This choice, even sub-optimal, enables a deep understanding of the convolution kernels. For these evaluations, a set of convolution kernels is experimented:

- A low-pass filter defined by $F_1(\boldsymbol{\lambda}) = (1 - \boldsymbol{\lambda}/\lambda_{\max})^\eta$ where η impacts the cut-off frequency

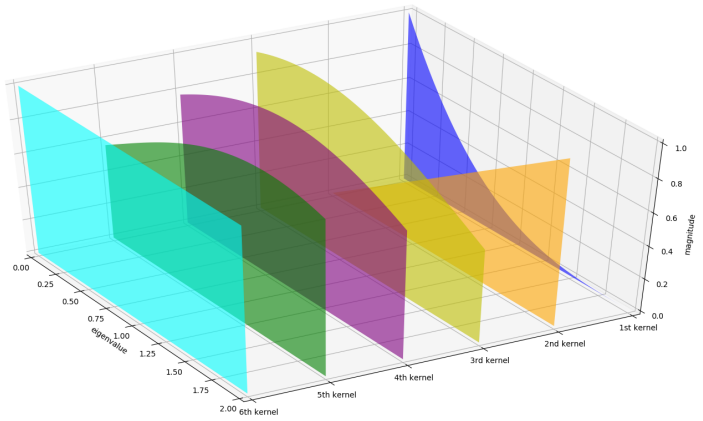


Figure 5.5: Visualization of the set of convolution kernels used in experimentation. For F_1, η is fixed to 3. For F_3, F_4, F_5, γ is fixed to 0.25.

- A high-pass filter defined by $F_2(\boldsymbol{\lambda}) = \boldsymbol{\lambda}/\lambda_{\max}$
- Three band-pass filters defined by:
  - $F_3(\boldsymbol{\lambda}) = \exp(-\gamma(0.25\lambda_{\max} - \boldsymbol{\lambda})^2)$
  - $F_4(\boldsymbol{\lambda}) = \exp(-\gamma(0.5\lambda_{\max} - \boldsymbol{\lambda})^2)$
  - $F_5(\boldsymbol{\lambda}) = \exp(-\gamma(0.75\lambda_{\max} - \boldsymbol{\lambda})^2)$
- An all-pass filter defined by $F_6(\boldsymbol{\lambda}) = 1$

Figure 5.5 presents the different filters used in our experimentations.

We firstly consider a model composed of only F_1. This choice comes from the fact that state-of-the-art GNNs are sort of low-pass filters (see Section 4.3) and perform well on the datasets of Table 5.1. Hence, it is interesting to evaluate our framework with F_1. For the experiments, the value of η are tuned for each dataset, using the validation loss value and accuracy, yielding $\eta = 5$ for Cora and Citeseer, and $\eta = 3$ for PubMed. Since there is only one convolution kernel, depthwise separable convolutions are not necessary for this model. Therefore, this model can be seen as similar to those from [23, 43] but using a different convolution kernel. This approach is denoted as LowPassConv in the results section (Section 5.3.1).

Beyond this low-pass model, we also evaluate different combinations of the $F_i(\boldsymbol{\lambda})$ through the depthwise separable schema defined in Section 5.2.

Table 5.2: Used kernels frequency profiles and architecture of models for each transductive dataset. DSG refers to Depthwise Separable Graph convolution layer, G to Graph convolution layer, D to Dense layer

Dataset	Architecture
Cora	$F_1(\lambda) = (1 - \lambda/\lambda_{max})^5$ $F_3(\lambda) = \exp(-0.25(0.25\lambda_{max} - \lambda)^2)$ $F_4(\lambda) = \exp(-0.25(0.5\lambda_{max} - \lambda)^2)$ $F_5(\lambda) = \exp(-0.25(0.75\lambda_{max} - \lambda)^2)$ DSG160-DSG7
Citeseer	$F_1(\lambda) = (1 - \lambda/\lambda_{max})^5$, $F_6(\lambda) = 1$ DSG160-DSG6
Pubmed	$F_1(\lambda) = (1 - \lambda/\lambda_{max})^3$, $F_2(\lambda) = \lambda/\lambda_{max}$ DSG16-DSG3

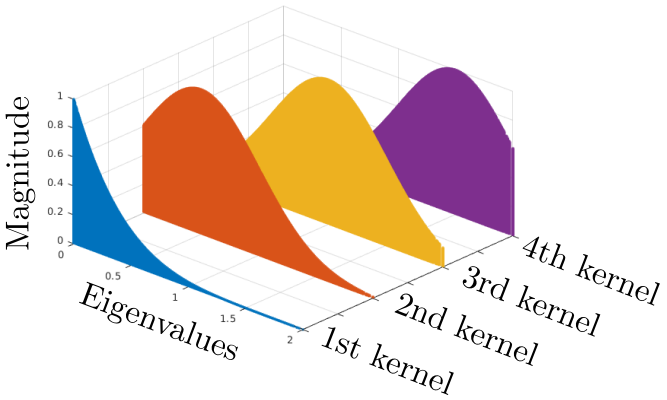


Figure 5.6: Designed convolution's frequency profiles for Cora dataset.

For experiments involving $\{F_3(\lambda), F_4(\lambda), F_5(\lambda)\}$, the bandwidth parameter γ was tuned using train and validation sets. Table 5.2 details the best models found on the validation set. As an example, for Cora dataset, 4 kernels are used by a DSGCN with 160 neurons: $F_1(\lambda)$, $F_3(\lambda)$, $F_4(\lambda)$, $F_5(\lambda)$. As an illustration, Figure 5.6 provides the standard frequency profiles of this designed convolution on Cora dataset. The models of Table 5.2 are denoted as DSGCN in the following.

Results Obtained results on transductive learning are given in Table 5.3. We compare the performance of the proposed LowPassConv and DSGCN to state-of-the-art methods. We first can see that our low-pass convolution kernel (LowPassConv) obtains comparative performance with existing methods. This result confirms our theoretical analysis which states that GCN and GAT mostly

Table 5.3: Comparison of methods on the transductive learning problems using publicly defined train, validation and test sets classification. Accuracies on test set are reported with their standard deviations on 20 random runs.

Method	Cora	Citeseer	Pubmed
MLP	0.551	0.465	0.714
Planetoid [88]	0.757	0.647	0.744
MoNet [60]	0.817 ± 0.005	-	0.788 ± 0.003
ChebNet [23]	0.812	0.698	0.744
CayleyNet [51]	0.819 ± 0.007	-	-
DPGCNN [61]	0.833 ± 0.005	0.726 ± 0.008	-
GCN [43]	0.819 ± 0.005	0.707 ± 0.004	0.789 ± 0.003
GAT [79]	0.830 ± 0.007	0.725 ± 0.007	0.790 ± 0.007
LowPassConv	0.827 ± 0.006	0.717 ± 0.005	0.794 ± 0.005
DSGCN	**0.842 ± 0.005**	**0.733 ± 0.008**	**0.819 ± 0.003**

correspond to low-pass filters (Section 4.3). Second, DSGCN outperforms state-of-the-art methods thanks to the flexibility provided by the different filters. Moreover, the depthwise separable strategy allows to limit the number of parameters, hence controlling the overfitting. These results show the potential of not being restricted to low-pass filters.

It is worth noting that the good results obtained by low-pass approaches show that these three classification tasks are mainly low-pass specific problems. Hence the flexibility provided by DSGCN has only a limited effect on the final performance when applied to this kind of problems. However, some problems may not be low pass specific, but properties to predict may be associated to high frequencies. On one hand, traditionnal GNNs may fail since they are limited to only low pass filtering, and thus can not predict correctly these properties. On the other hand, using DSGCN approach allows to use different filters, each one covering a particular range of the spectrum. Then, DSGCN may lead to a significative different performance compared to the ones obtained by traditionnal GNNs.

5.3.2 Inductive Learning Problem

Inductive Learning problems are common in chemoinformatics and bioinformatics. In an inductive setting, a given instance is represented by a single graph. Thus, models are trained and tested on different graph sets.

In the graph neural networks literature, there is a controversy concerning the transferability of spectral designed convolutions from learning graphs to unseen graphs. Some authors consider that convolutions cannot be transferred

Table 5.4: Summary of inductive learning datasets used in this paper.

	PPI	PROTEINS	ENZYMES
Type	Node Class.	Graph Class.	Graph Class.
# Graph	24	1113	600
# Avg.Nodes	2360.8	39.06	32.63
# Avg.Edges	33584.4	72.82	62.14
# Features	50	3 label	3 label + 18 cont.
# Classes	2 (121 criterias)	2	6
# Training	20 graphs	9-fold	9-fold
# Validation	2 graphs	1-fold	1-fold
# Test	2 graphs	None	None

[60], while very recent theoretical [52] and empirical [44] works show the contrary. In this subsection, we target to bring an answer to this controversy by experimenting our proposal on inductive learning problems.

Datasets Inductive experiments are led on 3 datasets: a multi-graph node classification dataset called Protein-to-Protein Interaction (PPI) [94] and on two graph classification datasets called PROTEINS and ENZYMES [41]. The protocols used for the evaluations are those defined in [79] for PPI and [89, 17, 78, 87] for PROTEINS and ENZYMES datasets.

The PPI dataset is a multi-label node classification problem on multi-graphs. Each node has to be classified either True or False for 121 different criteria. Each node is described by a 50-length continuous feature vector. The PPI dataset includes 24 graphs, with a train/validation/test standard splitting.

The PROTEINS and ENZYMES datasets are graph classification datasets. There are 2 classes in PROTEINS and 6 classes in ENZYMES. In PROTEINS dataset, there are three different types of nodes and one continuous feature. But we do not use this continuous feature on nodes. In ENZYMES dataset, there are 18 continuous node features and three different kinds of node types. In the literature, some methods use all provided continuous node features while others use only node label. This is why ENZYMES results are given using either all features (denoted by ENZYMES-allfeat) or only node labels (denoted by ENZYMES-label).

Since there is no standard train, validation and test sets split for PROTEINS and ENZYME, the results are given using a 10-fold cross-validation (CV) strategy under a fixed predefined epoch number. The CV only uses training and validation set. Specifically, after obtaining 10 validation curves corresponding to 10 folds, we first take average of validation curves across the 10 folds and then select the single epoch that achieved the maximum averaged

validation accuracy. This procedure is repeated 20 times with random seeds and random division of dataset. Mean accuracy and standard deviation are reported. This is the same protocol than [89, 78, 87, 17].

A summary of these three datasets is given in Table 5.4.

Models For PPI, 7 depthwise graph convolution layers compose the model. Each layer has 800 neurons, except the output layer which has 121 neurons, each one classifying the node either True or False. All layers use a ReLU activation except the output layer, which is linear. No dropout or regularization of the binary cross-entropy loss function is used. All graph convolutions use three spectral designed convolutions: a low-pass convolution given by $F_1(\boldsymbol{\lambda}) = \exp(-\boldsymbol{\lambda}/10)$, a high-pass one given by $F_2(\boldsymbol{\lambda}) = \boldsymbol{\lambda}/\lambda_{\max}$ and an all-pass filter given by $F_3(\boldsymbol{\lambda}) = 1$.

For graph classification problems (PROTEINS, ENZYMES-label and ENZYMES-allfeat), depthwise graph convolution layers are not needed since these datasets have a reduced number of features. Thus, it is tractable to use all multi-support graph convolution layers instead of the depthwise schema. In these cases, our models firstly consist of a series of graph convolution layers. Then, a global pooling (i.e., graph readout) is applied in order to aggregate extracted features at graph level. For this pooling, we use a concatenation of mean and max global pooling operator, as used in [17]. Finally, a dense layer (except for ENZYMES-label) is applied, before the output layer as in [87].

All details about the architecture and designed convolutions can be found in Table 5.5.

Results

Table 5.6 compares the results obtained by the models described above and state-of-the-art methods. A comparison with the same models but without graph information, a Multi-Layer Perceptron (MLP) that corresponds to $C^{(1)} = I$ is also provided to discuss if structural data includes some information or not. To the best of our knowledge, such an analysis is not provided in the literature. Finally, results obtained by the same architecture with GCN kernel is also provided.

As one can see in Table 5.6, the proposed method obtains competitive results on inductive datasets. For PPI, DSGCN clearly outperforms state-of-the-art methods with the same protocol, reaching a micro-F1 percentage of 99.09% and an accuracy of 99.45%. For this dataset, MLP accuracy is low since the percentage of micro-F1 is 46.2 (random classifier's micro-F1 being 39.6%). This means that the problem includes significant structural information. Using

Table 5.5: Kernels frequency profiles and model architecture for each inductive dataset. meanmax refers to global mean and max pooling layer.
Same legend as Table 5.2.

Dataset	Architecture
PPI	$F_1(\boldsymbol{\lambda}) = \exp(-\boldsymbol{\lambda}/10)$ $F_2(\boldsymbol{\lambda}) = \boldsymbol{\lambda}/\lambda_{\max}$, $F_3(\boldsymbol{\lambda}) = 1$ DSG800-DSG800-DSG800-DSG800- DSG800-DSG800-DSG121
PROTEINS	$F_1(\boldsymbol{\lambda}) = 1 - \boldsymbol{\lambda}/\lambda_{\max}$, $F_2(\boldsymbol{\lambda}) = \boldsymbol{\lambda}/\lambda_{\max}$ G200-G200-meanmax-D100-D2
ENZYMES-label	$F_1(\boldsymbol{\lambda}) = 1$, $F_2(\boldsymbol{\lambda}) = \boldsymbol{\lambda}_s - 1$ $F_3(\boldsymbol{\lambda}) = 2\boldsymbol{\lambda}_s^2 - 4\boldsymbol{\lambda}_s + 1$, $\boldsymbol{\lambda}_s = 2\boldsymbol{\lambda}/\lambda_{\max}$ G200-G200-G200-G200-meanmax-D6
ENZYMES-allfeat	$F_1(\boldsymbol{\lambda}) = 1$, $F_2(\boldsymbol{\lambda}) = \exp(-\boldsymbol{\lambda}^2)$ $F_3(\boldsymbol{\lambda}) = \exp(-(\boldsymbol{\lambda} - 0.5\lambda_{\max})^2)$ $F_4(\boldsymbol{\lambda}) = \exp(-(\boldsymbol{\lambda} - \lambda_{\max})^2)$ G200-G200-meanmax-D100-D6

the GCN kernel, which operates as a low-pass convolution (see Section 4.3), the accuracy increases to 0.592, but again not comparable with state-of-the-art accuracy.

For the PROTEINS dataset, one can see that MLP ($C^{(1)} = I$) reaches an accuracy that is quite comparable with state-of-the-art GNN methods. Hence, MLP reaches a 74.03% validation accuracy while the proposed DSGCN reaches 77.28%, which is the best performance among GNNs. This means that PROTEINS problem includes very few structural information to be exploited by GNNs.

ENZYMES dataset results are very interesting in order to understand the importance of continuous features and their processing through different convolutions. As one can see in Table 5.6, there are important differences of performance between the results on ENZYMES-label and ENZYMES-allfeat. When node labels are used alone, without features, MLP accuracy is very poor and nearly acts as a random classifier. When using all features, MLP outperforms GCN and even some state-of-the-art methods. A first explanation is that methods are generally optimized for just node label but not for

Table 5.6: Comparison of methods on inductive learning problems using publicly defined data split for PPI dataset and 10-fold CV for PROTEINS and ENZYMES datasets. PPI results are the test set results reported by micro-F1 metric percentage. Others are CV results reported by accuracy percentage. Results denoted by * were reproduced from original source codes but denoted feature set.

Method	PPI	PROTEINS	ENZYMES	
	All Features	Node Label	Node Label	All Features
GraphSAGE [31]	76.8	-	-	-
GAT [79]	97.3 ± 0.20	-	-	-
GaAN [90]	98.7 ± 0.20	-	-	-
Hierarchical [17]	-	75.46	64.17	-
Diffpool [89]	-	76.30	62.50	66.66*
ChebNet [44]	-	75.50 ± 0.40	58.00 ± 1.40	-
Multigraph [44]	-	76.50 ± 0.40	61.70 ± 1.30	68.00 ± 0.83
GIN [87]	-	76.20 ± 0.86	-	-
GFN [78]	-	76.56 ± 0.30*	60.23 ± 0.92*	70.17 ± 0.86
MLP ($C^{(1)} = I$)	46.2 ± 0.56	74.03 ± 0.92	27.83 ± 2.51	76.11 ± 0.87
GCN (3.18)	59.2 ± 0.52	75.12 ± 0.82	51.33 ± 1.23	75.16 ± 0.65
DSGCN	**99.09 ± 0.03**	**77.28 ± 0.38**	**65.13 ± 0.65**	**78.39 ± 0.63**

continuous features. Another one is that the continuous features already include information related to the graph structure since they are experimentally measured. Hence, their values are characteristic of the node when it is included in the given graph. Since GCN is just a low-pass filter, it removes some important information on higher frequency and decreases the accuracy. Thanks to the multiple convolutions proposed in this paper, our GNN DSGCN clearly outperforms other methods on the ENZYMES dataset.

5.4 Conclusion

In this chapter, we proposed DSGCN, a method that uses the Depthwise Separable Convolution framework in order to compute multiple convolution kernels while limiting the increase of parameters. On the other hand, we proposed to use handcrafted convolution kernels in order to cover most of the spectrum range.

We evaluated our method on both transductive and inductive tasks. Our method obtained state of the art results for each dataset on both categories. Moreover, results obtained with low-pass filtering on transductive datasets confirmed our theoretical analysis in Chapter 3 that most of GNNs are limited

to low-pass filtering. Indeed, when our method is limited to low-pass filtering, results obtained are similar to other GNNs methods. Globally good results also induce the fact that those tasks are principally low-pass specific.

Nevertheless, the proposed approach has some drawbacks. First, it needs eigenvalues and eigenvectors of the graph Laplacian. If the graph has more than 20k nodes, computing these values is not tractable. Second, we did not propose yet any automatic procedure to select the best frequency profile of convolution. Hence, the proposed approach needs expertise to find the appropriate graph kernels. Third, although our theoretic complexity is the same than GCN or ChebNet, in practice our convolutions are more dense than GCN, which makes it slower in practice since it cannot take advantage of sparse matrix multiplications. Last, if edge type can be handled by designing convolution for each type, the proposed method does not handle continuous edge features and directed edges.

In conclusion, even though spectral and spatial GNNs are closely related, there is still an interest to separate them. Indeed, each of them has its advantages and drawbacks, and some applications will be more suited to spectral GNNs while other will take advantage of spatial GNNs. Our DSGCN method is part of the spectral GNNs, and still share some of their drawbacks while improving their advantage.

In introduction, we reminded two major limitations of GNNs. This chapter has proposed a solution to the first one, and next chapter aim to tackle the second limitation, which is the use of edge features.

Chapter 6

Edge Embedding Graph Neural Network

6.1 Introduction

Chapter 4 was concluded by pointing out two major limitations of GNNs: the fact that most GNNs are limited to low-pass filtering, and their inability to exploit edge attributes. A solution to tackle the first limitation was proposed in chapter 5. This chapter proposes a solution to the second limitation : the use of edge attributes.

Such a problem is important since many real-world problems rely on graphs that include edge features. One can cite bond type for molecules, type of relationship in social networks or quantified adjacency relations in Region Adjacency Graphs.

Paradoxically, few existing GNN methods are able to efficiently take into account edge features. Hence, if discrete edge attributes can be easily incorporated into general frameworks of GNNs as it will be presented in section 6.2, it becomes much more difficult when those attributes are multiple and continuous. Being able to effectively take into account edge attributes can thus be an interesting point in order to gather more information about the graphs which may lead to increase the performance of GNNs.

In this chapter we present the Edge Embedding Graph Neural Network (EEGNN) model, a general GNN model that is able to exploit any type and any number of edge attributes, independently of the node attributes, while keeping a reasonable size of the model.

This contribution has been guided by a real-world application which consists in detecting symbols in floorplan images. This problematic consists

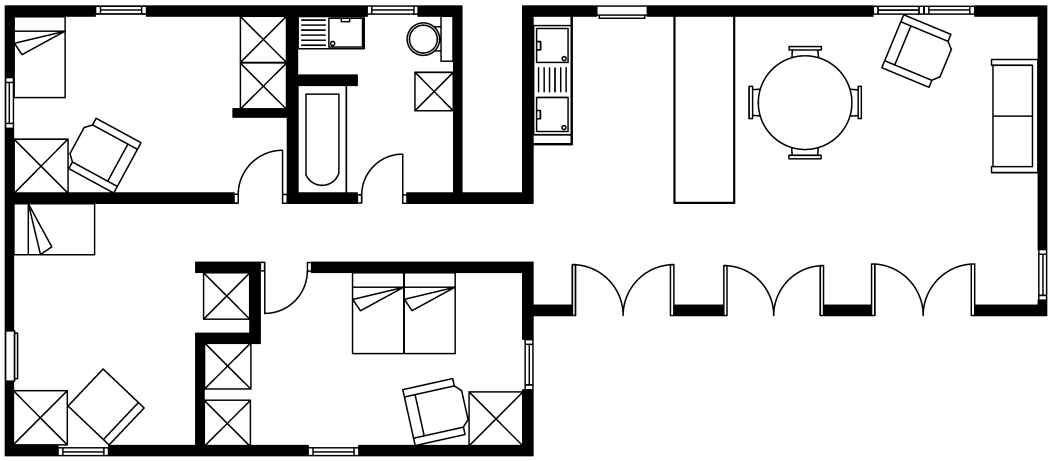


Figure 6.1: Example of a floorplan image

in processing floorplan images as depicted in Fig 6.1 to automatically detect symbols used by architects to define a particular piece of furniture such as a table, chair, bed... This processing may add an high level information to the floorplan compared to the hardly exploitable information encoded by pixels themselves. To implement this detection, we strongly believe that using a graph approach constitutes an advantage with respect to classical CNN approaches. Indeed, floorplan images are not natural images and are mostly formed of binary pixels composing geometric shapes. Therefore, using a structural approach may be more adapted to this problematic. In the proposed framework, and floorplan images are firstly converted into graphs (see Figure 6.2) where nodes represent symbol parts while edges represent neighborhood relationships between these symbol parts. Each node can be attributed by a vector describing the area associated to this symbol, and each edge is labeled by a continuous value quantifying the strength of each neighborhood relationship. See section 6.4.1 for more details about this transformation of floorplan images to graphs.

Using such a representation, detecting a given symbol consists in finding a set of nodes, where each node is a part of the symbol. This problem can be described in less or more complex ways. First, it can be seen as a node classification problem, where each node of a same symbol must be associated to the same class. However, such an approach may neglect the fact that it is the set of nodes which forms the symbol. Therefore, some irregularities may appear, such as a disconnected set of nodes belonging to the same class. To overcome this problem, we can formulate it as a link prediction problem where the tasks consists in deciding if two nodes belongs to the same symbol instance or not.

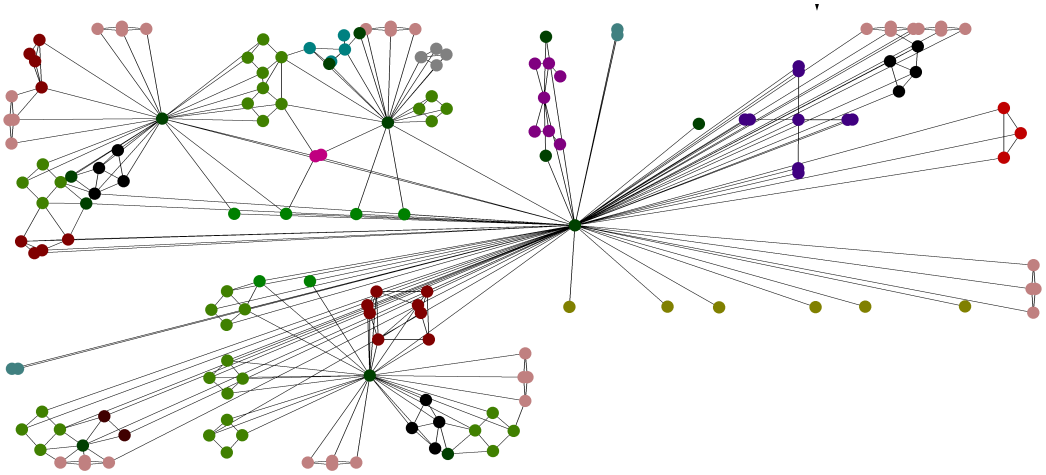


Figure 6.2: Example of a floorplan image converted into a graph

Then, a mix of this two approaches leads to a third problem formulation where the two tasks are combined together. We can note that this hierarchy of tasks leads us very close to the subgraph isomorphism problem, where the task now is to identify a subgraph, ie. a symbol, within a target graph, ie. the floorplan. More details about these different tasks are presented in section 6.4.2, together with an analysis of their results in section 6.5 Considering this problem of symbol detection in floorplan images, one can note that the information carried by edge label is essential to solve the problem. Therefore, we first propose in this chapter to explore state of the art GNNs methods dealing with edge attributes, section 6.2. This review leads to the observation that most of methods limit the use of edge attributes to discrete ones or to a fixed latent dimension size.

Therefore, we propose in section 6.3 the EEGNN model which alleviates drawbacks identified in SOTA methods. The symbol detection problem is presented in section 6.4. This method is then evaluated on our symbol detection tasks together with SOTA methods in section 6.5.

Obtained results show the importance of structure on one side through the comparison with a baseline model, and the importance of edge attributes with our method on the other side.

6.2 Graph Neural Networks and edges

Given the previous definition of GNNs : $\mathbf{H}^{(l+1)} = \sigma(\sum_{s=1}^{S}(\mathbf{C}^{(s)}\mathbf{H}^{(l)})\mathbf{W}^{(l,s)})$, we can note that the matrix $\mathbf{A}_{\mathrm{e}}$ encoding the edge attributes is not explicitly included. Thus, edge attributes are not taken into account within this

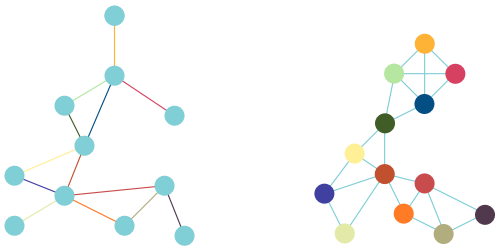


Figure 6.3: A graph (on the left) and its associated line graph (on the right)

formulation and in general in the GNN framework. However, edge attributes can bring relevant information about the data encoded by the graph. Therefore, it can be interesting to take into account this data in order to build more expressive and more general predictive models.

Multiple solutions have been proposed to use edge attributes. First, if only edges are attributed, one solution would be to use the line-graph, also called edge-dual graph or conjugate graph $L(G) = (L(\mathcal{V}), L(\mathcal{E}))$ of the graph $G = (\mathcal{V}, \mathcal{E})$. The line-graph is a transformation of graph G where each edge $e \in \mathcal{E}$ is a node in graph $L(G)$, and for each pair of edges in G that have a node in common there is a vertex in $L(G)$ linking the corresponding nodes. Edge features in G are then transformed into node features in $L(G)$, which makes GNNs able to take into account the edge features as node features. Figure 6.3 presents an example of a graph and its associated line graph.

However this solution is not satisfying, because the node features in G are transposed in the edge features of $L(G)$, leading to the same problem than before. This solution is thus usable if only edges are attributed.

As presented in section 3.4, in the discrete case, one solution commonly used is to apply one convolution kernel with one weight matrix per edge attribute (Equation 6.1 which was previously defined as Equation 3.8). It consists in splitting the adjacency matrix $\mathbf{A}$ into a tensor $\mathbf{C} \in \mathbb{R}^{|V|\times|V|\times|\mathcal{L}_\mathcal{E}|}$ where $\mathbf{C}^{(i)} = \mathbf{C}(:,:,i)$ encodes the adjacency matrix of edges labeled by the ith value of edge attribute. An example of such a transformation is presented in Figure 6.4.

$$\mathbf{H}^{(l+1)} = \sigma((\sum_{i=1}^{|\mathcal{L}_\mathcal{E}|} \mathbf{C}^{(i)}\mathbf{H}^{(l)}\mathbf{W}^{(l,i)}) + \mathbf{b}^{(l)}) \tag{6.1}$$

The limitation of this method lies in the fact that it is restricted to discrete edge attributes. In addition, in the case where $|\mathcal{L}_\mathcal{E}|$ is too high, it will require a huge number of parameters hence increasing the complexity of the model. Finally, in this model, each edge attribute is computed independently.

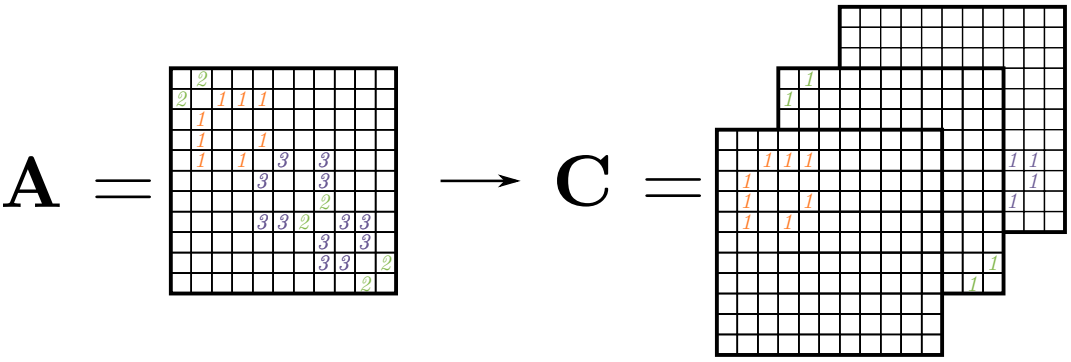


Figure 6.4: Example on how edges represented on the adjacency matrix $\mathbf{A}$ can be used to produce a $\mathbf{C}$ tensor.

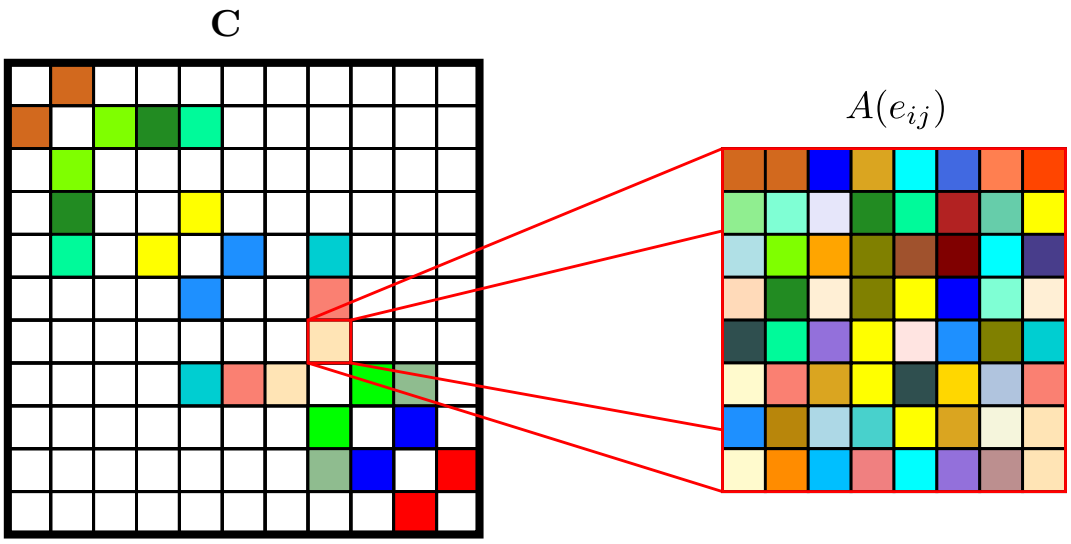


Figure 6.5: Example on how edges are embedded into a $f_l \times f_l$ matrix with the edge network method.

To take into account continuous edge attributes, [27] proposes to define a neural network which learns a $f_l \times f_l$ matrix representation for each edge attribute vector of size $\dim(\mathcal{L}_\mathcal{E})$, where f_l is the dimension of each node's hidden representation of layer l. This neural network consists in learning the matrix $\mathbf{W}^A \in \mathbb{R}^{\dim(\mathcal{L}_\mathcal{E}) \times f_l \times f_l}$ which is applied to the extended adjacency matrix $\mathbf{A}_\text{e} \in \mathbb{R}^{|V| \times |V| \times \dim(\mathcal{L}_\mathcal{E})}$. $\mathbf{A}_\text{e}\mathbf{W}^A \in \mathbb{R}^{|V| \times |V| \times f_l \times f_l}$ thus represents the learned edge embedding. Figure 6.5 shows an example of the matrix representation that can be learned.

This new edge representation can then be used to compute an enhanced representation of each node's hidden representation :

$$\hat{\mathbf{h}}_i^{(l+1)} = \sum_{j \in \mathcal{N}(i)} \mathbf{h}_j^{(l)} \mathbf{A}_{\mathrm{e}} \mathbf{W}^{A(l)}(i,j), \forall i \in |V| \tag{6.2}$$

Compared to vanilla GNN, Equation 6.2 extends the convolution operator by learning coefficients which allows to weight the contribution of each neighbor feature according to the embedding of $e_{i,j}$ computed by $\mathbf{A}_{\mathrm{e}}\mathbf{W}^{A}$. This thus leads to a better expressiveness of the neural network. Once applied over each node, the new hidden representation $\hat{\mathbf{H}}^{(l)}$, where $\hat{\mathbf{H}}^{(l)}(i,:) = \hat{\mathbf{h}}_i^{(l)}$, can be multiplied by the classical matrix $\mathbf{W}^{(l)}$ to compute layer $l+1$ as in Equation 6.3

$$\hat{\mathbf{H}}^{(l+1)} = \sigma(\hat{\mathbf{H}}^{(l)}\mathbf{W}^{(l)}), \tag{6.3}$$

where $\hat{\mathbf{H}}^{(l)} \in \mathbb{R}^{|V| \times f_l}$ is the result of the extended convolution operator.

This proposition is very interesting in the way it redefines the convolution operator to enhance the capability and the expressiveness of GNNs. However, this improvement comes with a cost. The number of parameters which compose the $\mathbf{W}^{A}$ weight matrix is $f_l * f_l$, which is really expensive, considering that f_l is generally included between 100 and 200.

In [73], authors focus on molecular graphs with their method Edge Attention-based multi-relational Graph Convolutional Networks (EAGCN). Designing edge based graph neural network operating on molecular graphs is interesting due to the fact that the chemical bonds linking atoms hold an important information about the molecule. Molecular graph edges are generally attributed by at least the kind of chemical bond, also denoted as bond order, linking two atoms together (simple, double, triple, aromatic...), but other attributes may also be encoded. In [73], 5 different edge attributes are used: the bond order, the atom pair types, the aromaticity, the conjugation and if the edge belongs to a ring or not. Each attribute indexed by k can be represented by a matrix $\mathbf{A}_k \in \mathbb{R}^{|V| \times |V| \times f_{e_k}}$, where f_{e_k} is the dimension of kth edge attribute. However, in [73], all edge attributes are defined as discrete values (there is no continuous value), and are thus encoded by one hot vectors.

Instead of embedding directly the edges as in Edge Network, EAGCN propose to compute an attention matrix for each attribute. These attention matrices will learn a weight for each combination of edge and attribute. This weight quantifies the contribution of incident node of each edge for the update of node hidden states, and this independently for each attribute. This method computes one attention matrix $\mathbf{A}_{att,k}$ for each edge attribute k. Each attention matrix is built depending on two informations: the different values of each edge attribute which are encoded by $\mathbf{A}_k$ and a learned weight matrix $\mathbf{W}_k^E \in \mathbb{R}^{f_{e_k} \times 1}$ (Equation 6.4). The mask function is used to ensure that no entry in the

attention matrix is added if there is no edge between the corresponding nodes. The softmax_j operator simply applies a softmax function to each line j of the matrix given as input. This operation acts as a normalization step to avoid order of magnitude differences between attention weights of the different edges.

$$\mathbf{A}_{att,k} = \text{softmax}_j(\text{mask}(\mathbf{A}_k \mathbf{W}_k^E)) \tag{6.4}$$

Then this attention matrix is used as a convolution kernel in a classical message passing framework as in Equation 6.5. Since there are K attention matrices, each one defined independently for each edge attribute k, the different embedding must be combined into an unique node hidden representation. The authors of [73] propose either to concatenate (Equation 6.5) or to compute a weighted sum (Equation 6.6) of the K representations.

$$\mathbf{H}^{(l+1)} = ||\sigma(\mathbf{A}_{att,k}\mathbf{H}^{(l)}\mathbf{W}_k^{(l)})|1 \leq k \leq K|| \tag{6.5}$$

$$\mathbf{H}^{(l+1)} = \sum_{k=1}^{K} \beta_k \sigma(\mathbf{A}_{att,k}\mathbf{H}^{(l)}\mathbf{W}_k^{(l)}) \tag{6.6}$$

The edge attention method allows to weight each edge and to select the most relevant ones. This is done without hugely increasing the number of parameters hence limiting the complexity of the model. However, the experiment in [73] only consider edges having discrete attributes. The method is not tested on edges having continuous attributes despite the fact that there is no restriction on processing those kind of attributes. In addition, the architecture of EAGCN depends on K, which may vary with the graph. Hence, a particular EAGCN architecture has to be designed explicitly for each kind of graphs, making this approach problem specific.

6.3 Proposed model

To circumvent the explosion of parameters of Edge Network and the specificity of EAGCN architecture, we propose to learn hidden representations where the dimension is controlled.

For this purpose, we extend [27] by embedding edge features into a new vector of size f_e, where f_e is the size of the edge hidden representation. Each edge $e \in \mathbb{R}^{\dim(\mathcal{L}_\mathcal{E})}$ is thus embedded into a new space $\mathbb{R}^{f_e}$. As a result, the adjacency matrix $\mathbf{A}_\text{e} \in \mathbb{R}^{|V|\times|V|\times\dim(\mathcal{L}_\mathcal{E})}$ is also embedded into a new space $\mathbb{R}^{|V|\times|V|\times f_e}$. This embedding then becomes our convolution support $\mathbf{C}$. This embedding is computed by a neural network $\mathbf{W}^A \in \mathbb{R}^{\dim(\mathcal{L}_\mathcal{E})\times f_e}$ following

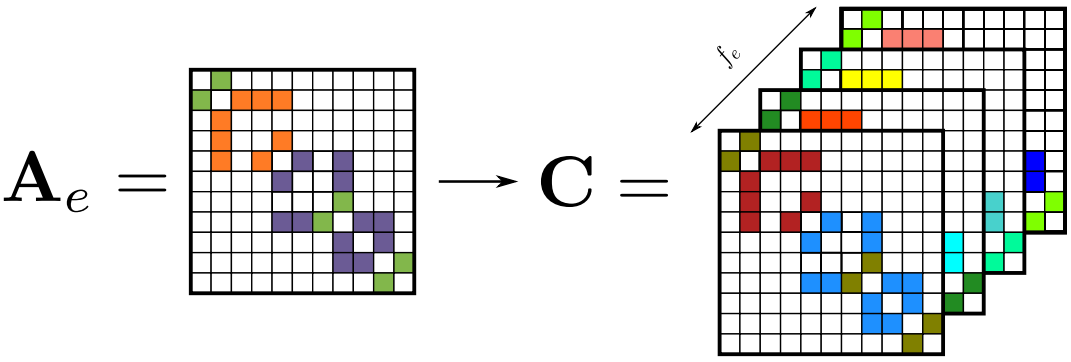


Figure 6.6: Example on how edges are embedded by our EEGNN method to learn new convolution kernels depending on the edge attributes.

Equation 6.7. To summarize, our method aims to learn f_e new convolution kernels according to the embedding of each edge e_{ij}. Example of our method is presented on Figure 6.6.

In order to keep the graph topology, it is important that the application of a neural network to $\mathbf{A}_\mathrm{e}$ does not include new edges. Thus, 0 values in $\mathbf{A}_\mathrm{e}$ must remain 0 after applying a neural network or a non null value will correspond to the creation of an non-existing edge. With the aim of applying a neural network $f(\mathbf{A}_\mathrm{e}(i,j)) = \sigma(\mathbf{A}_\mathrm{e}(i,j)\mathbf{W}^A + \mathbf{b})$, we have to ensure that $f(\mathbf{A}_\mathrm{e}(i,j)) = 0$ if $\mathbf{A}_\mathrm{e}(i,j) = 0$. A neural network includes 3 operations: the matrix multiplication, the bias add and the activation function. The matrix multiplication is not a problem, since $\mathbf{A}_\mathrm{e}(i,j)\mathbf{W}^A = 0$ if $\mathbf{A}_\mathrm{e}(i,j) = 0$. However, the use of bias is not compatible with our requirement, since $0 + b = b$. We then decide to remove biases to our model. The activation function used is $\mathrm{ReLU} = \max(0, x)$, which ensures that $f(0) = 0$. Finally, the neural network for learning edge hidden representation is defined by Equation 6.7.

$$\mathbf{C} = \mathrm{ReLU}(\mathbf{A}_\mathrm{e}(i,j)\mathbf{W}^A) \tag{6.7}$$

This learned edge hidden representation can then be used as multiple convolution kernels in a classical GNN framework as in Equation 6.8.

$$\mathbf{H}^{(l+1)} = \sigma((\sum_{i=0}^{f_e} \mathbf{C}^{(i)}\mathbf{H}^l\mathbf{W}^{(l,i)}) + \mathbf{b}^{(l)}) \tag{6.8}$$

This proposition has multiple advantages. Firstly, it doesn't require a huge number of parameters. Compared to Edge Network that requires $\dim(\mathcal{L}_\mathcal{E}) * f_l * f_l$ parameters, our method only requires $\dim(\mathcal{L}_\mathcal{E}) * f_e$ parameters. EEGNN thus requires fewer parameters as long as $f_e < f_l * f_l$. Another advantage of our

method is that it is not specific to discrete edge attributes and do not depend on those edge attributes. This allows our method to be used in a more general way than EAGCN. The last advantage of our method is that it is able to learn new convolution kernels.

The major drawback of our method is the fact that f_e is an hyperparameter that has to be defined a priori. The second drawback is that even though the number of parameters added is inferior to Edge Network, $\dim(\mathcal{L}_\mathcal{E}) * f_e$ parameters are still added, increasing the complexity of the model. However, to contain this complexity, our model can be computed as it was done with DSGCN. Instead of having f_e weight matrices $\mathbf{W}$, we only have one, and edge attributes are used to compute a weight that will correspond to a contribution coefficient, similarly to DSGCN.

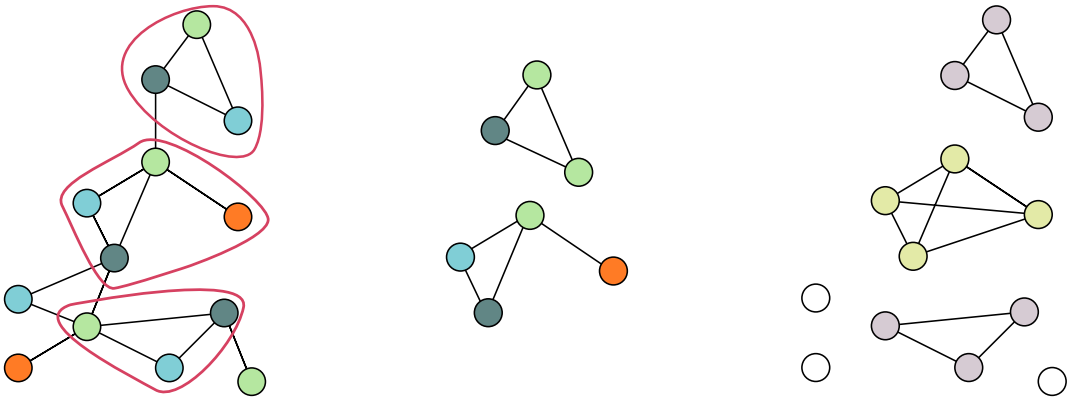
$$\mathbf{H}^{(l+1)} = \sigma(((\sum_{i=0}^{f_e}(\mathbf{A}_\mathrm{e}\mathbf{W}^A)(:,:,i)\mathbf{H}^{(l)})\mathbf{W}^{(l)}) + \mathbf{b}) \tag{6.9}$$

Next section presents the applicative problem used to evaluate the EEGNN model.

6.4 Symbol detection in floorplan images

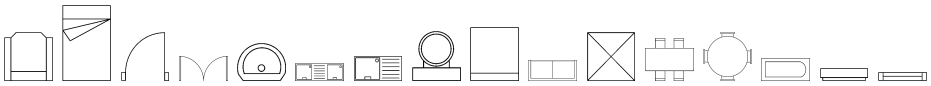
As stated in introduction, the work described in this chapter has been led in the context of an application targeting symbol detection in document images. In this section, the different problematic of this application, and the proposed solutions, are presented. In a first part, the conversion of floorplan images into Region Adjacency Graphs (RAGs) is presented. Within those graphs, each node corresponds to a white region in the original image, and each edge indicates an adjacency relationship between two regions encoded by incident nodes. Nodes are attributed using Zernike moments shape descriptors [76], and edges are characterized using the distance between centers of gravity of connected components. Figure 6.2 presents an example of the floorplan image converted into a graph.

In a second part, we focus on the symbol detection problem [50] in such a graph which consists in detecting each occurrence of a set of symbols (see Figure 6.8) called pattern graphs in a graph called target graph. Figure 6.7 presents the problem of symbol detection in a graph. One could see the similarity with the subgraph isomorphism problem, and this similarity will be discussed in conclusion. Main difference being that, in the case of symbol detection, there is a dataset with localized symbols, while in the subgraph isomorphism case, the subgraph is a second input with the target graph. To tackle the symbol



(a) Target graph (b) Pattern graphs (c) Output graph

Figure 6.7: Inputs and outputs of symbol detection problem. (a) shows an example target graph while (b) present different pattern graph to be detected. The excepted detection is obtained on (c).



(a) (b) (c) (d) (e) (f) (g) (h) (i) (j) (k) (l) (m) (n) (o) (p)

Figure 6.8: Symbol models to be detected

detection problematic, two strategies are studied: as a node classification task and as a link prediction task. However, as it will be presented, neither of the strategies solve the problem alone. It is thus the combination of both steps, either computed independently or computed simultaneously through a multi task model, that allows to tackle the symbol detection problem.

6.4.1 From images to graphs

Before presenting our strategies for detecting symbols, we present in this subsection the graph generation method. Region Adjacency Graphs (RAGs) are well suited to encode symbols and technical drawings since they allow to model the adjacency relationships between the regions extracted by a segmentation process. Working on technical documents, the digital images are mainly binary images where white components correspond to the background and black components to drawings. Segmenting such kind of images can be achieved using component labeling [18]. However, aiming at finely modeling adjacency relationship between two regions, a binary image can be firstly thinned [8].

The obtained image is then morphologically the same than the initial image of the document but the thickness of the drawing components is reduced to a single pixel. Using this image, each white component is mapped to a node of the graph. Then, the skeleton branches represent frontiers and adjacency relationships between two regions. An edge is then built between two nodes representing regions separated by a skeleton branch. Figure 6.9 illustrates the overall process of transforming a binary image to a RAG.

To enrich such a description, attributes have to be assigned to each node and edge. Many features have been proposed to characterize shapes and spatial relations [77]. Among them, Zernike Moments (ZM) [76] yield interesting results for pattern recognition tasks when invariance to affine transforms and robustness to degradations are required. Hence, a feature vector corresponding to a set of ZM is assigned to each vertex in order to characterize shapes. A continuous attribute for each edge connecting nodes representing adjacent regions (source and target) is defined by the *relative distance* between their gravity centers, computed with respect to the overall area of the two regions:

$$e_{source,target} = \frac{d_e(g_{source}, g_{target})}{\sqrt{\text{Area}(source) + \text{Area}(target)}} \tag{6.10}$$

where d_e denotes the Euclidean distance between gravity centers.

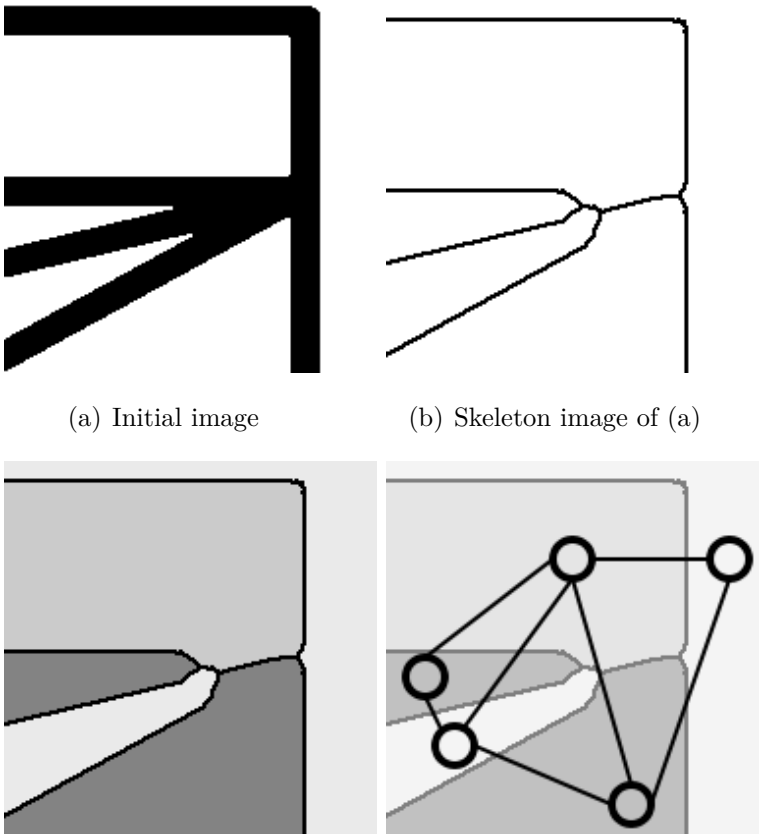
A Graph-based representation $G = (\mathcal{V}, \mathcal{E}, \mu, \xi)$ of a document is finally defined where $\mathcal{V}$ encodes the white connex regions and $\mathcal{E}$ corresponds to adjacency relationships between regions. Labeling function on nodes $\mu : \mathcal{V} \rightarrow \mathbb{R}^{24}$ encodes the morphology of each region according to its ZM and labeling function $\xi : \mathcal{E} \rightarrow \mathbb{R}$ expresses the geometrical properties of an adjacency relationship, as defined in Eq. 6.10.

6.4.2 Learning to detect symbols

In this section, two different ways to learn to detect symbols in a graph are presented. Indeed, from the graph point of view, this particular task can be divided into two distinct parts: a first node classification part, that allows to detect different class of symbols, and a link prediction part, that allows to detect different occurrences of the same symbol.

Symbol detection as node classification task

A first idea would be to consider this problem through a node classification problem. In the target graph, nodes belonging to a symbol occurrence would



(a) Initial image (b) Skeleton image of (a)

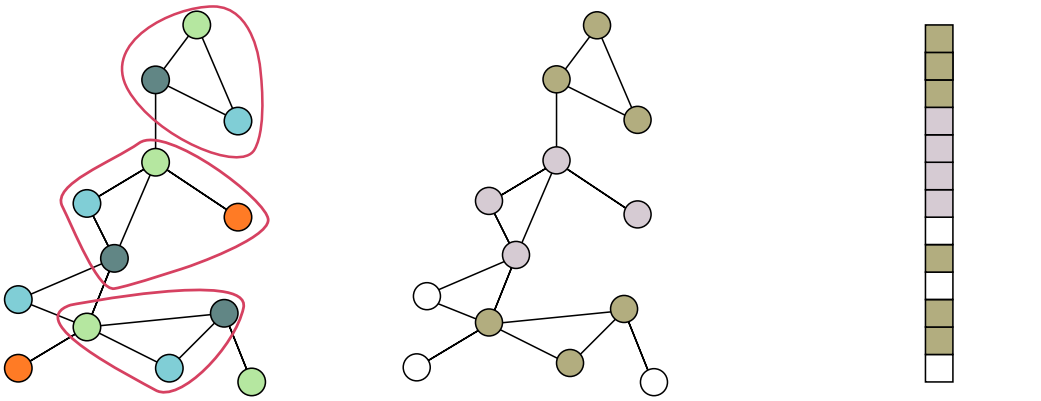
(c) Components labels of (b) (d) Corresponding RAG

Figure 6.9: From initial image to Region Adjacency Graph.

be activated while the rest would be unactivated. However, this kind of representation might not be strong enough to detect or distinguish multiple instances of the pattern graph in the target graph. Indeed, in the case where multiple instances of the symbol exist, the node classification solution is not able to detect if two nodes belong to two different instances or if they belong to the same instance. The kind of result obtained by node classification is illustrated in figure 6.10.

Symbol detection as link prediction task

On the other hand, in order to detect different occurrences of the same symbol, a way is to consider link prediction. The idea is to differentiate two occurrences of the same pattern by predicting if two nodes belong to the same occurrence. This prediction is made by activating the edge between the two nodes if they

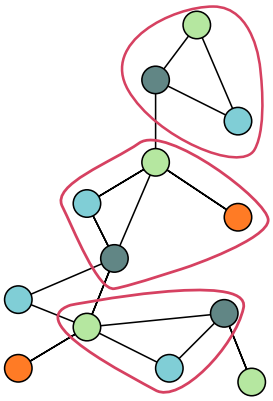


(a) Target graph (b) Output graph (c) Output graph

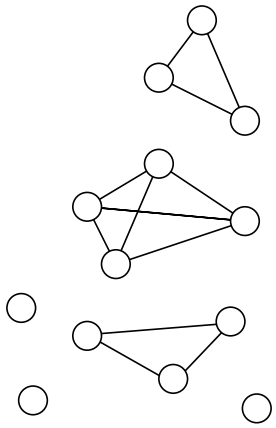
Figure 6.10: Inputs and outputs of symbol detection problem. (a) shows an example target graph where the symbols to detect are surrounded. The detection in node classification is obtained on (b).

belong to the same occurrence, and unactivating it if they don't. Theoretically, a clique made of nodes and activated edges is obtained, which correspond to an occurrence of the searched pattern. By relaxing the constraint of the occurrence being a clique, an occurrence can then be a connected component in the graph. A connected component made of nodes and activated edges then correspond to an occurrence of the searched pattern. Ultimately, this connected component is a clique in which activated edges mean all nodes belong to the same symbol occurrence. We study the use of pairwise classification [7] to produce a $n \times n$ binary matrix $\mathbf{M}$ for a graph having n nodes. $\mathbf{M}$ is a symmetric matrix, with $\mathbf{M}(i, j)$ equals to 1 if v_i and v_j represent parts belonging to the same symbol occurrence, and 0 otherwise. Such a matrix can be seen as the adjacency matrix of a non connected graph, composed of all the nodes describing the connected components of the original image. The symbols are then represented by the different connected parts of the whole graph, each of them being a clique. In Figure 6.11, we take back our example in Figure 6.7 to show an example of a desired output.

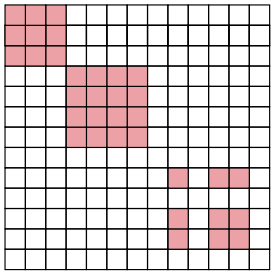
Finally, one can see that the ability of detecting multiple occurrences of the same symbols has been obtained at the cost of detecting different symbols. In order to obtain a complete model, one solution would be to combine both tasks into a multi-task model. Next section presents the experimental result obtained for each task.



(a) Target graph



(b) Output graph



(c) Output Matrix

Figure 6.11: Inputs and outputs of our proposition. (a) shows an example target graph where the symbols to detect are surrounded. The output graph is obtained on (b) and (c) is the corresponding matrix to be predicted.

6.5 Experimental Results

In this section, the results obtained by the different approaches introduced in 6.4.2 are presented. Three different cases are evaluated: the symbol detection as a node classification task, the symbol detection as a link prediction task, and the symbol detection as a combination of node classification and link prediction tasks. The different evaluated models and the protocols for each case are firstly presented. Results obtained for each case are then presented.

6.5.1 Evaluated models and protocols

For each task, we evaluated 4 different models. The first model is a neural network that aims at classifying each node using only its own features, and then discarding all information encoded within the graph representation. This model serves as a baseline in order to measure the information brought by the structural information. It is made of three dense layers, with the third one used for decision.

The second model is a Graph Neural Network (GNN) made of three layers. Similarly to the baseline, the third layer is used for decision. With this model, we are only able to use the existence of an edge between a pair of nodes, but not the label associated to this edge.

The third and fourth experimented models use respectively the Edge Network (EN) and the Edge Embedding (EE) defined in Section 6.3. Those models use both the existence of an edge and its label.

In order to make a fair comparison of the four models, we make them as similar as possible. Thus, both the neural networks of the baseline and the three GNN produce 200 features for each node. For our edge embedding model, we studied 4 different values for f_e: 2, 4, 8 and 16. Since $f_e = 8$ obtained the best results on validation and test, in each experiment, we only show results for this value.

For the protocol, the 200 graphs are firstly randomly divided into 3 sets. 160 graphs are used for training, 20 for validation and 20 for test. Node and edge attributes are normalized to have a value either between 0 and 1 or -1 and 1, depending on the minimum value of the attribute. Each model is then trained using the Adam optimizer with an initial learning rate of 5.10^{-3}.

The node classification task is trained for 100 epochs using the crossentropy loss (see Eq.6.11) and the best model in validation is used for test. The evaluated metric in the node classification task is the node classification accuracy.

$$\mathcal{L}_{node} = -\sum_{p_i} p_i \log(p_i) \tag{6.11}$$

In the link prediction task, classes are hugely imbalanced. To tackle this imbalance, the focal loss [55] is used (see Eq. 6.12) in the link prediction problem.

$$\mathcal{L}_{link} = -\sum_{p_i} \alpha_i (1 - p_i)^{\gamma} \log(p_i) \tag{6.12}$$

Hyperparameter of focal loss used are $\gamma = 2$ and $\alpha = 0.3$, where α weights both classes (class 0 is weighted by α and class 1 is weighted by $(1 - \alpha)$) and γ is an hyperparameter that controls the contribution of each sample in the loss computation depending on how hard is the sample to be classified. A hard sample correctly classified will thus have more importance than an easy sample correctly classified. $\alpha = 0.3$ allows to weight the class 0, which is over-represented, with a lower contribution than the class 1, which is under-represented. $\gamma = 2$ is a standard value for focal loss. In the link prediction task, the evaluated metric is the ratio of detected symbols. A symbol is defined as detected as a connected component of the graph consisting of all the nodes describing a symbol occurrence and that does not contain any extra node.

In the last case, the combination of both node classifcation and link predicton tasks is studied. The network is trained for 300 epochs, with a combination of both standard crossentropy and focal loss (see Eq.6.13). The computed loss is thus the mean of both losses. The computed metric is the ratio of detected symbols, where a symbol is considered as detected as a connected component of the graph consisting of all the nodes correctly classified describing a symbol occurrence and that does not contain any extra node. This task is thus harder than before, due to the node classification condition.

$$\mathcal{L}_{combine} = \frac{\mathcal{L}_{node} + \mathcal{L}_{link}}{2} \tag{6.13}$$

In order to evaluate the robustness of each model, a Gaussian noise is added on the node features for the test dataset. We ran those experiments using a 10-fold cross-validation in order to limit split bias. Mean and standard deviation of accuracy for each model are reported for each experiment

6.5.2 Results in node classification task

Results in node classification are firstly tested. Results obtained following the protocol previously presented in section 6.5.1 are reported in Table 6.1.

As one can see, results for GNN, EN and EE are pretty similar and close to 100%, widely increasing results obtained by the base model. However, adding noise allows to show disparities between the different models. Indeed, EN and EE models perform better than GNN when noise is added. Moreover, when Gaussian noise variance is set to 0.2, our model EE obtains better results than EN.

Table 6.1: Results obtained (in percentage of accuracy of nodes classification) by the different experimented models.

Model	**Gaussian Noise Variance**			
	0.0	0.05	0.1	0.2
MLP	95.57 ± 0.83	60.37 ± 1.82	39.99 ± 0.98	23.54 ± 0.79
GNN	99.54 ± 0.17	99.27 ± 0.30	96.36 ± 0.82	79.67 ± 3.03
EN	99.67 ± 0.12	99.54 ± 0.18	98.19 ± 0.36	85.77 ± 1.94
EE	99.64 ± 0.12	99.53 ± 0.17	98.08 ± 0.57	88.33± 2.16

6.5.3 Results in link prediction task

In a second step, results obtained in link prediction are given in Table 6.2.

Results show that integrating edge features definitively improves the symbol detection when no noise is added. However, in the case of EN, those results drop quickly and obtain worst results with Gaussian noise variance above 0.1. The poor results obtained by the different models can be explained with two arguments. First, the edge prediction task is a very imbalanced task. Most of pairs of nodes are made of nodes that do not belong to the same symbol, leading to more than 96% of pairs that are labeled as 0. Second, the importance of the noise. Since each feature is normalized, adding a Gaussian noise of variance 0.2 leads to an important noising.

Table 6.2: Results obtained following the previous metric by the different experimented models.

Model	**Gaussian Noise Variance**			
	0.0	0.05	0.1	0.2
MLP	45.65 ± 4.51	0.46 ± 0.32	0.0 ± 0.0	0.0 ± 0.0
GNN	90.13 ± 1.28	75.12 ± 4.37	44.22 ± 4.57	17.18 ± 3.62
EN	92.59 ± 1.02	76.54 ± 2.00	39.88 ± 3.25	12.31 ± 1.81
EE	92.22 ± 1.11	79.63 ± 3.03	49.72 ± 5.15	21.21 ± 2.71

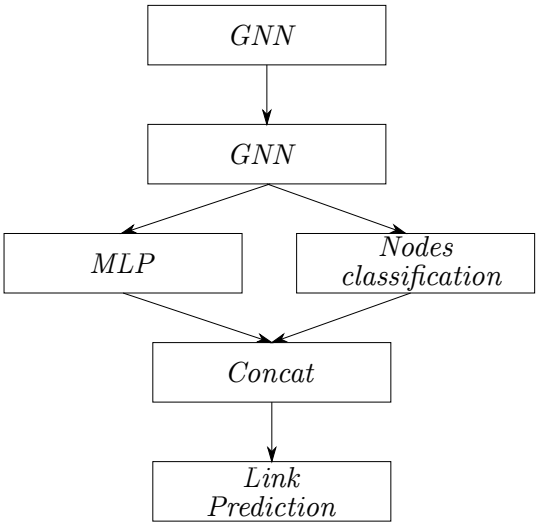


Figure 6.12: General model used for multi task prediction

6.5.4 Results in both node classification and link prediction

In this last experiment, we examine the effect of combining both link prediction and node classification in a multi task model. A single model is thus used to predict both links and classes. In addition, the node classification result is used to improve the link prediction task by concatenating the class prediction to the hidden feature vector. Figure 6.12 presents the general model used. Apart from this, the same four models that were presented in Section 6.4 are compared.

Results are shown in Table 6.3.

Table 6.3: Results obtained by the different experimented models using the symbol detection accuracy.

Model	**Gaussian Noise Variance**			
	0.0	0.05	0.1	0.2
MLP	40.69 ± 3.19	1.04 ± 0.46	0.08 ± 0.10	0.0 ± 0.0
GNN	86.30 ± 2.31	66.09 ± 3.12	37.51 ± 2.55	12.42 ± 1.48
EN	90.99 ± 0.64	75.29 ± 2.65	37.64 ± 4.29	12.10 ± 1.66
EE	87.22 ± 1.74	70.88 ± 2.53	41.00 ± 3.37	14.22 ± 3.02

Conclusions drawn from these results are similar to the previous ones: EN obtained best results but are less robust to noise. Conversely, our method performs below EN with unnoised data, but performs slightly better with noised data, the gap increasing with the noise intensity.

6.6 Conclusion

In this chapter introduction and in chapter 4, the lack of method using efficiently edge attributes was pointed out. This chapter has proposed to fill this gap with a new GNN method called EEGNN, which has been evaluated through a symbol detection problem.

Floorplan images are firstly transformed into Region Adjacency Graph by the approach. In order to achieve the symbol detection problem, the approach divide the problem in two parts: a node classification task that allows to detect different classes of symbols, and a link prediction part that allows to detect different occurrences of the same symbol.

Three different models of graph neural network have been compared with a baseline that does not use any structural information. The experimental evaluation is then divided into 3 parts: a first part which only considers node classification, a second that only considers link prediction, and a third that considers a model that combines both node classification and link prediction. Each GNN models presents good results largely improving result obtained with the baseline model.

Even imperfect, we argue that the obtained results provide a first empirical evidence that machine learning can be used to solve a subgraph isomorphism problem when learning data are available.

In future works, it could be interesting to consider using graph pooling strategy to compute a new graph where a node corresponds to a whole symbol. Another perspective would be to use Siamese networks to detect a single symbol specified in input, as presented in figure 6.13.

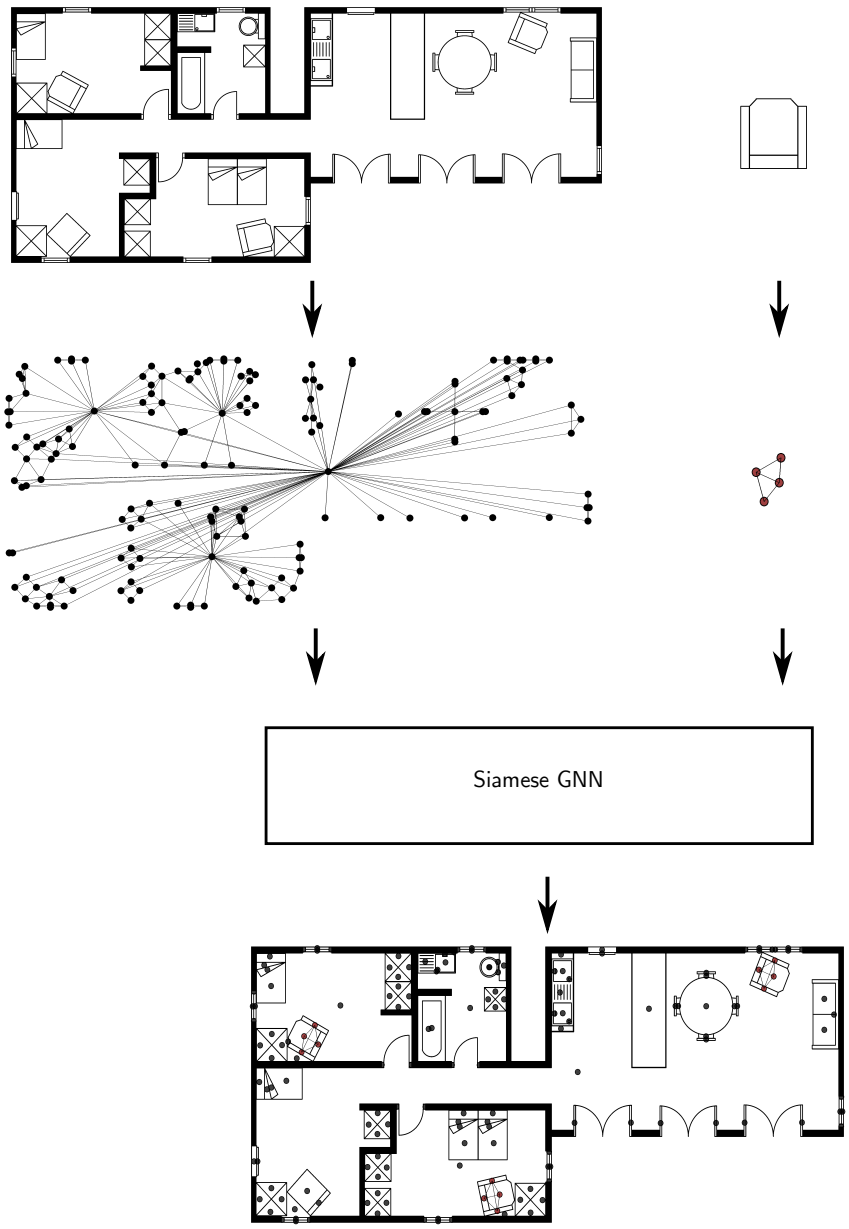


Figure 6.13: Example on how Siamese networks could be used to detect a single symbol.

Chapter 7

Conclusion

7.1 Review of the contributions

The main subject of this thesis has been the analysis of Neural Networks dedicated to graphs, called Graph Neural Networks (GNN), and how we can improve those methods. Having introduced and defined the different backgrounds and notations in chapter 2, chapters 3 to 6 have proposed a set of contributions to answer the research questions raised in chapter 1. We can now conclude this manuscript by giving a synthesis of these contributions.

Research question 1: *In the last years, Graph Neural Networks have became one the hottest topics in machine learning. This has led to a "jungle" of models and frameworks. Our first question is thus: how to classify existing approaches ?*

The answer of this question has been given in chapter 3. After a presentation of the first use of the "GNN" term, the two different main theories that have led to the actual GNNs have been explained. However, this taxonomy was not sufficient to classify every models. We thus added a third category, which contains models based on spectral-rooted spatial convolution.

Research question 2: *Given this taxonomy, can we merge all these approaches into one single model ?*

This question has been addressed in chapter 4, where we proved that Spectral GNNs are a special case of Spatial GNNs, leading to GNNs being defined as Spatial. This result allowed us to provide a general model of GNNs, based on the definition of convolution supports. Those convolution supports can either be defined in a spatial way, with the adjacency matrix or the graph

Laplacian for example, or in a spectral way with the eigen decomposition of the graph Laplacian, or an approximation of this eigen decomposition.

Research question 3: *Since all existing models can be merged into a single framework, one can wonder what is the spectral behavior of models defined as spatial compared to models defined as spectral ?*

One interesting point of the general model previously defined is the fact that it allowed to answer this research question by analyzing the frequency profile of convolution supports, whether they are originally defined in a spatial way or a spectral way. This analysis has been done in second part of chapter 4

Five notable GNNs have been compared, namely ChebNet, CayleyNet, GCN, GIN and GAT. In those analysis, the main result obtained is the fact that GCN, GIN and GAT are limited to low-pass filtering. Some models such as ChebNet or CayleyNet are able to generate high-pass and band-pass filters, however they are not able to produce specific band-pass filters.

Research question 4: *Since most of existing models are limited to low-pass filtering while obtaining good results on reference datasets, one can wonder if low-pass filters are sufficient for all problems ?*

We addressed this question in the last part of chapter 4. In this section, two toy problems inspired from image analysis have been proposed. This allowed to easily propose problems that require band-pass filtering. Results showed the limits of GNNs to design some kind of filters, and the importance for GNNs to be able to design those filters. In order to obtain efficient models, GNNs should be able to produce any kind of filtering, as a CNN would.

Research question 5: *Following previous results, most of GNNs appear to be limited to low-pass filtering. Models that are able to propose band-pass filtering require a huge number of parameters. Are GNNs able to produce band-pass filtering at low-cost ?*

We proposed in chapter 5 to adapt the Depthwise Separable Convolution to GNNs. Depthwise Separable Convolution was originally used with CNNs, in order to produce filters at lower parameter costs. The adaptation has led to our method DSGCN.

Our DSGCN method has been tested on both transductive and inductive tasks, and has presented very good results on both tasks. Moreover, the global good results obtained by popular GNN are comparable to our DSGCN method limited to low-pass filtering, proving that those methods are also limited to low-

pass filtering on one hand, and once again that applicative problems generally require only low-pass filtering.

The proposed approach has however some drawbacks. First is that it still needs to compute eigenvalues and eigenvectors of the graph Laplacian, which is untractable when the graph has too much nodes. The second drawback is that convolution filters are handcrafted, which means that efficient convolution filters on one applicative problem may totally fail on another one. It thus require to design filters according to the problem, which is antinomic with deep learning, where convolution filters are learned. Finally, the last drawback is that continuous edge features are not taken into account by our method.

Research question 6: *One of the strength of graphs is the versatility of their attributes. Both nodes and edges can carry information. Yet, most of GNN models only use node attributes. Are GNN limited to use the information carried by node attributes ?*

In chapter 6 a new model called EEGNN has been proposed. In our proposed approach, edge attributes are embedded into a new feature space in a general way. The Depthwise Separable architecture presented previously is also applicable, allowing to reduce the number of parameters. Our model has then been evaluated on a problem of symbol detection in floorplan images and obtained promising results.

7.2 Perspectives

Being able to efficiently take into account more information carried by a graph has been the main subject of this thesis. Recently, a set of benchmark datasets called Open Graph Benchmark [38] has been proposed. Comparing the GNN models proposed in this thesis on unified datasets to add new analysis capacity is one of our shot-term perspective.

In order to improve GNNs, one important point might be to learn to produce any kind of filtering. Indeed, we saw that one major limitation of most of notable GNNs was that they were limited to low-pass filtering. In addition to an expressive power being limited, another problematic induced by low-pass filtering is the limitation in terms of number of GNN layers. Indeed, accumulating layers limited to low-pass filtering causes oversmoothing. Since each layer smoothe the signal, each node hidden representation tends to be more and more similar to its neighborhood with the addition of layer, leading to the same representation for each node in the graph. Being able to learn any kind of filtering might thus allow to accumulate more layers, leading to

deeper architectures. Deeper architecture would then allow to take into account information carried by distant nodes, which might be interesting depending on the problem. Deep architecture and learnable convolution filters are some of CNNs strengths, and applying them to GNNs could then be a path to follow.

Another way to keep improving GNNs is through the edge representation. Our proposed method only consider edge attributes to update the edge representation, in a MLP way. However, one could consider updating the edge representation in the same way node representations are updated in GNNs, meaning updating the edge representation through the actual edge representation on one side, and the connected nodes on the other side.

In our last contribution, we proposed a model that implicitly solve the subgraph isomorphism problem. This NP-hard problem is one of the classical combinatorial optimization challenge, but there are other ones. We are convinced that in the future, machine learning will be used combinatorial optimization problems. Using machine learning to solve such problems is an exciting idea and even though some traditional combinatorial optimization problems such as the traveling salesman problem (TSP) are studied by the machine learning community, this has not been the case for most of NP-hard problems.

Appendix A

Deep Learning for Graph Edit Distance Approximation

A.1 Introduction

As seen previously, graphs are a powerful tool to represent data. However, in machine learning, most of approaches in literature take as input numerical data, and use mathematical properties of euclidean spaces to build models. This is the case for deep learning for example.

Because graph space does not have a scalar product, a dissimilarity measure between graph is necessary to use machine learning common methods. The graph edit distance is one approach commonly used to answer this problem.

But the computation of the graph edit distance is an NP-complete problem, and is thus untractable when the size of graphs increase. In this chapter, we present a method to efficiently compute an approximation of the graph edit distance using a deep learning approach.

A.1.1 Graph Edit Distance

The Graph Edit Distance is a metric between two graphs $G_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $G_2 = (\mathcal{V}_2, \mathcal{E}_2)$, where $\mathcal{V}_1$ (resp. $\mathcal{V}_2$) is the set of nodes in G_1 (resp. G_2) and $\mathcal{E}_1$ (resp. $\mathcal{E}_2$) is the set of edges in G_1 (resp. G_2). This measure is computed by evaluating the cost of transforming G_1 into G_2. This transformation is obtained by a sequence of elementary operations which can have 3 different types: substitution, insertion and deletion. Each of these operation types can be applied over nodes or edges, reaching a total of 6 elementary operations. To each of those elementary operations is associated a cost, quantifying the

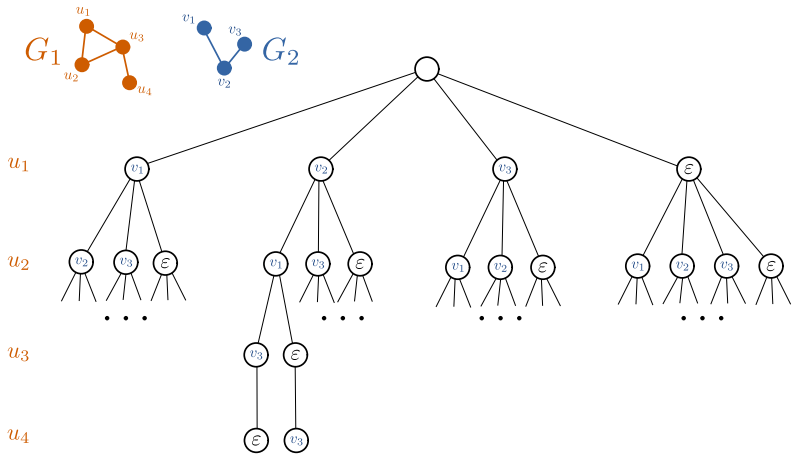


Figure A.1: Partial tree representing the set of edit paths between G_1 and G_2. A^* algorithm allows to browse this tree to find the edit path with the minimal cost.

transformation made on the graph. There is an infinite number of operation sequences transforming a graph G_1 into a graph G_2, each of them defining an edit path. The optimal edit path is the edit path whose cost is minimal among the set of edit paths. The edit distance is the cost associated to the optimal edit path.

The use of A^* algorithm is one of the first approach used to compute Graph Edit Distance [33]. This method builds and browses the solution tree to get one optimal edit path as well as the corresponding cost. More recently, the GED as also been computed as a binary linear programming problem, including [39, 48, 49]. However, getting an optimal edit path is an NP-complete problem, and induces a high complexity. For example, this complexity is in $O(n^m)$ for A^*, where n is the number of nodes in G_1 and m the number of nodes in G_2. This complexity limits the use of those methods to small graphs [2]. To tackle this problem, algorithms of lower complexity computing an approximation of the graph edit distance have been proposed. One of the first approximation approach has been proposed in [69]. In this method, authors transform the computation of the graph edit distance in a linear affectation problem, which can be solved in polynomial time. It is thus possible to process larger graphs.

Obviously, reducing the complexity is made at the expense of the precision of value obtained. The problem of a search of a compromise between approximation quality and complexity then arises, as shown in [2].

In this section, we propose a new method to approximate the graph edit distance, based on the use of recent deep learning methods to get a better accuracy with a low complexity. Section A.1.2 presents different methods used

in our approach. Section A.2 describes our approach. Finally, section A.3 presents the experimental evaluation.

A.1.2 GED approximation

One of the first proposed method for graph edit distance approximation is the Bipartite Graph Matching [69]. This method brings together graph edit distance and a node matching problem. The search for an optimal edit path is a quadratic assignment problem, simplified by [69] into a linear assignment problem to get an approximation.

We are looking for an optimal matching, in the meaning of a minimal cost matching between nodes. Formally, we are looking for a function $\varphi : \mathcal{V}_1 \to \mathcal{V}_2$, which, for a node u_k, matches it to a node $v_{\varphi(k)}$ and which minimizes the global cost of those matchings. For this, a matching cost has to be defined. Here, it is defined as the cost of transforming a node u_i into a node v_j. To simulate node suppression and insertion, empty nodes called ε-nodes are added. Thus, suppressing the node u_i is defined as a matching between node u_i and ε-node. On the other hand, insertion of node v_j is defined as a matching between ε-node and node v_j. Thus, m ε-nodes are added to G_1 and n ε-nodes are added to G_2, increasing both sizes of $\mathcal{V}_1$ and $\mathcal{V}_2$ to $n + m$.

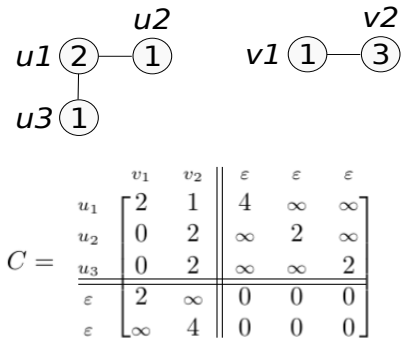
It may be important to notice that both nodes and edges can have features, and that the cost computation must take into account those features. The substitution cost between two nodes can for example be the euclidean distance between the two feature vectors of the nodes.

To take into account structural information in graphs, the minimal cost of matching incident edges of both nodes is also computed.

In equation A.1, $u_i, u_k \in \mathcal{V}_1$, $v_j, v_{\varphi_k} \in \mathcal{V}_2$, a_{ik} is the edge between u_i and u_k, $b_{j\varphi_k}$ is the edge between node v_j and v_{φ_k} and φ_k is the matching function. Finally, $\mathscr{S}(n+m)$ is the set of $(n+m)!$ possible permutations.

$$C(i,j) = c(u_i \to v_j) + \min_{\substack{(\varphi_1,\dots,\varphi_{n+m}) \\ \in \mathscr{S}(n+m)}} \sum_{k=1}^{n+m} c(a_{ik} \to b_{j\varphi_k}) \tag{A.1}$$

Finally, a matrix C is computed, of size $(n+m) \times (n+m)$ (cf Figure A.2), so we can take into account each cost for each node. The shape of the matrix allows to match each node u_i to a unique node v_j, with u_i and v_j which can possibly be ε-nodes. To ensure that each ε-node can only be associated to one unique node, the computed cost between this node and any other ε-node is fixed to ∞. In a pretty natural way, the cost of matching two ε-nodes together is set to 0, this matching inducing no graph transformation. Figure A.2 presents a C



$$C = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ \hline \varepsilon \\ \varepsilon \end{array} \begin{array}{c} \begin{array}{cc|ccc} v_1 & v_2 & \varepsilon & \varepsilon & \varepsilon \end{array} \\ \left[\begin{array}{cc||ccc} 2 & 1 & 4 & \infty & \infty \\ 0 & 2 & \infty & 2 & \infty \\ 0 & 2 & \infty & \infty & 2 \\ \hline 2 & \infty & 0 & 0 & 0 \\ \infty & 4 & 0 & 0 & 0 \end{array}\right] \end{array}$$

Figure A.2: C matrix example for two graphs. Nodes of the graph are attributed. The matching cost of two nodes is the difference between their attributes. The cost of matching two edges is 0. Cost of inserting/deleting nodes/edges is 1.

matrix example for two graphs. We can see that this matrix can be divided into 4 parts: a substitution part (up left), a deletion part (up right), an insertion part (down left) and an empty part (down right).

$$C = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \end{array} \begin{array}{c} \begin{array}{ccccccc} v_1 & v_2 & v_3 & \varepsilon_1 & \varepsilon_2 & \varepsilon_3 & \varepsilon_4 \end{array} \\ \left[\begin{array}{ccccccc} 1 & 0 & 1 & 3 & \infty & \infty & \infty \\ 1 & 0 & 1 & \infty & 3 & \infty & \infty \\ 2 & 1 & 2 & \infty & \infty & 4 & \infty \\ 0 & 1 & 0 & \infty & \infty & \infty & 2 \\ 2 & \infty & \infty & 0 & 0 & 0 & 0 \\ \infty & 3 & \infty & 0 & 0 & 0 & 0 \\ \infty & \infty & 2 & 0 & 0 & 0 & 0 \end{array}\right] \end{array}$$

The problem of node matching is thus a linear matching problem, where the costs of matching one node to another are encoded in the C matrix (Eq. A.1). This problem can be solved with the Hungarian algorithm (also called Munkres algorithm) [46]. With the obtained matching, it is possible to deduce an edit path which transforms G_1 into G_2 . From this edit path, we can easily compute its associated cost, which corresponds to the approximation made by the algorithm. Many propositions have been made to improve the algorithm, with greedy methods to reduce computation times as in [66], or iterative methods to increase accuracy as in [12].

It may be important to notice that the edit path obtained is not necessary optimal, and thus the obtained cost is not necessary optimal. Indeed, the way the matrix is built does only take into account partial structural information of the graph. The matching and the edit path are thus optimal only for the linear assignment problem. The cost associated to the edit path obtained is thus an upper bound of the graph edit distance.

[67] also proposes to compute a lower bound, by dividing by two the costs of edit operations over edges. [68] use this lower bound with the upper bound as a feature vector, which is used as input of an SVR to learn to predict a graph edit distance more accurately, getting encouraging results. But this prediction is limited by two points. First, having only two features to predict an accurate value might no be enough. Second, the approximation obtained with BP-GED is sometimes very inaccurate.

Even though those works are encouraging, there is, on our knowledge, no work using machine learning to approximate the graph edit distance.

In the next section, we present our new approach to approximate the graph edit distance with deep learning, by the use of the C matrix.

A.2 Deep Learning based method for GED approximation

We can suppose that learning with only two features does not allow to efficiently generalize. Our idea here is to directly extract features from the matrix C. However, using this matrix induces two problems when using learning methods with it.

First of all, graphs are, by definition, variable in size. The matrix C is thus also variable in size. This particularity prevents the use of most of learning methods, which requires the use of fixed size vectors.

Second comes from the fact that there is no order in the set of nodes of a graph. Thus, there are multiple C matrices for any pair of graphs. Each of those matrices corresponds to one permutation of node. In such a case, we need the prediction to be identical for a pair of graphs, independently of the node permutation.

To solve the first problem, we use tools from convolutional neural networks. Indeed, some frameworks require to work on varying size images. In those cases, one proposed way is to use Spatial Pyramid Pooling [34] for example.

A solution for the second problem is explained in next section.

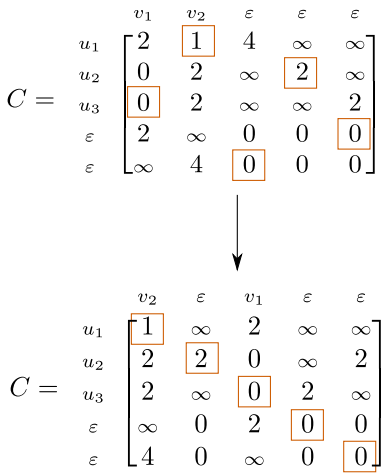


Figure A.3: Permutation matrix example

A.2.1 Network Input

Nodes ordering in a graph is not fixed, and many different node orders can express the same graph. Thus different C matrices can be computed for the same pair of graphs. Those matrices contain the same information, apart from one permutation. To be less sensitive to those permutations, we permute the matrix C. This permutation is made in a way that matches made by Munkres algorithm. Matrix C is thus permuted following permutation matrix P defined by equation A.2. This solves only partially the permutation problem, and a more efficient method could be used in future works.

$$P(i,j) = \begin{cases} 1 & \text{si } \varphi(i) = j \\ 0 & \text{sinon} \end{cases} \tag{A.2}$$

Figure A.3 shows an example of permuted matrix, following example in figure A.2. Matrices thus built and permuted are then used by our learning method to estimate the graph edit distance between two graphs. This matrix being of variable size, convolutional layers are used, those layers being not limited to a fixed size input.

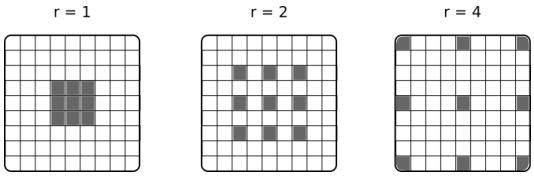


Figure A.4: Visual example of dilated convolutions.

A.2.2 Convolutional Network

Convolutional networks do use convolutional filters with learned weights. Those filters are generally of fixed size (3×3, 5×5) to limit the number of parameters to learn. Those filters are combined with pooling operations to artificially increase the context taken into account by those filters. This context is called receptive fields. To increase this receptive fields, in small windows (2×2), pooling aggregates the set of values in the windows and computes one value, which can be the max value of the window, or the minimum, the mean... The obtained matrix is thus a shorter version of the original matrix, where the reduction ratio depends on the window size.

The reduction of the matrix size is not the only way to increase the size of the receptive field. Another way is to use dilated convolutions. The idea of dilated convolution is to use sparse filters, that are bigger but with same amount of free parameters to learn. A dilation rate is used for this. This method is based on the "à trou" algorithm [37], and has been used for example in semantic segmentation [19]. Figure A.4 shows an example of dilated convolutions.

Furthermore, cost matrices have very variable sizes, and can be very small. Using the dilated convolution seems more judicious than pooling, to prevent to reduce the size of an already small matrix.

Finally, the architecture of our model uses the same configuration than the first layers of VGG-16 [75]. This architecture has shown a good capacity of feature extraction. Our networks thus use the 6 first convolutional layers of VGG-16, where we replace the pooling by dilation. The network thus has 2 first layers of 64 filters of size 3×3 and dilation 1. 2 layers with 128 filters of size 3×3 and dilation 2 follows. Finally, 2 layers with 256 filters of size 3×3 and dilation 4 finalize the features extraction part.

The purpose of our work is in a first time to evaluate the ability to extract relevant features from the cost matrix. The network architecture is not optimized to solve the problem, which could be an interesting perspective.

Our model architecture thus does not change the shape of the matrix. If the input matrix has a shape of $(n + m) \times (n + m)$, then the output of the feature extraction part is a tensor of shape $(n+m) \times (n+m) \times 256$. However, in

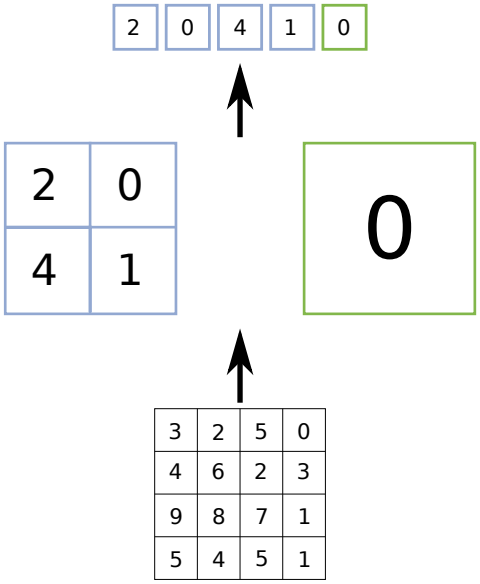


Figure A.5: Spatial Pyramid Pooling applied on a 4×4 matrix with $N = 1$ and $N = 2$

order to achieve regression, we have to obtain a fixed-size matrix. The solution adopted here is the Spatial Pyramid Pooling

A.2.3 Spatial Pyramid Pooling

The principle of the Spatial Pyramid Pooling is to apply pooling not on a fixed size windows but on multiple windows of variable sizes. For this purpose, the input matrix is divided into N^2 parts. Thus, the shape of each window is a ratio of the shape of the input matrix. For each of these windows, a value is extracted. As for traditional pooling, this value can be the maximum, the minimum, the mean or any feature of the window. Here, we are looking for a minimal cost, we then used the minimum value.

This operation is made for multiple values of N, to extract features for each filter. We decided to use 2 different values for N in our experiments: 1 and 2, corresponding to 5 values per filter. The last layer having 256 filters, 1280 features are extracted by the Spatial Pyramid Pooling. Figure A.5 shows an example of SPP.

A regression step is finally performed by dense layers applied on the features extracted by Spatial Pyramid Pooling. Two dense layers of 21 neurons are used followed by one dense layer of one neuron, corresponding to the predicted value.

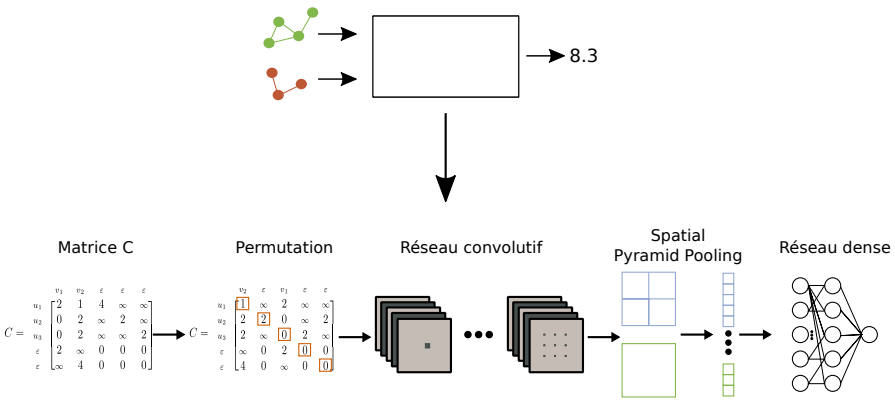


Figure A.6: Complete process of our method.

The network is trained using the Adam optimizer [42]. Figure A.6 shows the complete process of our method and its global architecture.

The complete network, including dilated convolutions, Spatial Pyramid Pooling and dense layers, has 1,172,217 parameters and has been developed with Keras [21]. The next section presents the experiments that were conducted to evaluate the performance of the proposed model.

A.3 Experiments

This section presents the different experiments made to evaluate the performances of the proposed approach. The experimental protocol with the metrics are firstly presented. We then evaluate the results obtained on 2 graph datasets (Letter and Fingerprint [64]). The results are compared to those obtained by BP-GED [69] and SVR [68], a method that is also based on machine learning.

A.3.1 Experimental protocol and metrics

The experimental protocol is the following. 1000 graphs are randomly extracted from the initial dataset. For each pair of graphs, the exact graph edit distance is computed, using the A^* algorithm. The cost matrix C is also computed and permuted, following the strategy defined in A.2.1. The set of 5.10^5 pairs of graphs is then splitted into 3 sets: a training set with 40% of the data, a validation set with 10% of the data, and a test set with 50% of the data. The validation set is used to ensure that our model does not overfit during training.

Figure A.7: Letter A with different noises.

In the literature, two metrics are used to evaluate the results: the mean relative error, expressed as a percentage in equation A.3, and the mean squared error, defined in equation A.4. In both equations, d_{a_i} is the approximate distance of the i^{th} pair of graphs, and d_{e_i} is the exact distance of the i^{th} pair of graphs.

$$MRE = 100 * \frac{1}{N} \sum_{i=1}^{N} \frac{|d_{a_i} - d_{e_i}|}{d_{e_i}} \tag{A.3}$$

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (d_{a_i} - d_{e_i})^2 \tag{A.4}$$

A.3.2 Letter

Letter [64] is a graph dataset representing capital letters made of line segments. Each letter is noised. Figure A.7 shows multiple examples of noised letters. Each node corresponds to one vertex of the letter, attributed with x and y coordinates. Edges represent the existence of a line segment between two vertices and are not attributed.

The cost to substitute two nodes is computed as the euclidean distance of the coordinate of each node, while the cost to substitute two edges is 0. Following [65], the cost to delete or insert nodes is fixed to 0.9 while the cost to delete or insert an edge is fixed to 1.7.

Table A.1 shows the obtained results on the Letter dataset with MRE and MSE for 3 methods : BP-GED, SVR and our approach. Figure A.8 shows those results graphically. For a better readability, the log_{10} of the results is used.

	BP-GED	SVR	Proposed approach
MRE	13.11	7.76	4.2
MSE	3.07	0.86	0.32

Table A.1: Comparison of BP-GED, SVR and our approach on LETTER

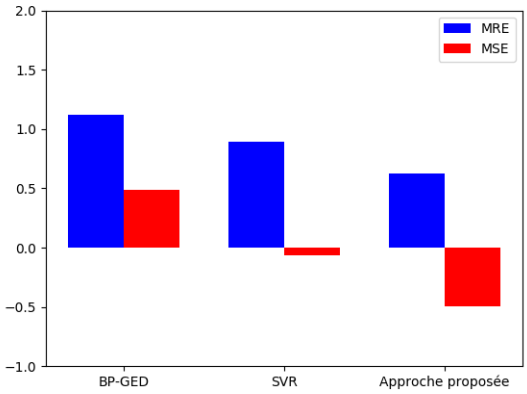


Figure A.8: Obtained results by BP-GED, SVR and our approach on LETTER. log_{10} of MRE and MSE is computed for a better readability

Figure A.9: Finger print used to generate the dataset

As one can see, our approach is clearly better, in MRE as in MSE. The use of matrix C thus have helpful information that are correctly extracted by our approach, leading to a better accuracy.

A.3.3 Fingerprint

Fingerprint [64] is a dataset made of graphs representing finger prints (Figure A.9). It has been obtained by a skeletonization of finger print images. Each branch and vertex of the obtained skeleton is a node, and edges represent the links between the nodes.

Nodes have no attributes. The matching of two nodes is thus only penalized by the cost for matching their respective edges. Edges are characterized by their angle. Thus, the cost of matching two edges depend on the distance between both angles, to the nearest 2π. Insertion and deletion costs are fixed to 0.525 for nodes and 0.125 for edge, following [65].

	BP GED	SVR	Proposed approach
MRE	68.39	7.56	1.15
MSE	16.15	0.10	0.006

Table A.2: Comparison between BP-GED, SVR and our approach on Fingerprint.

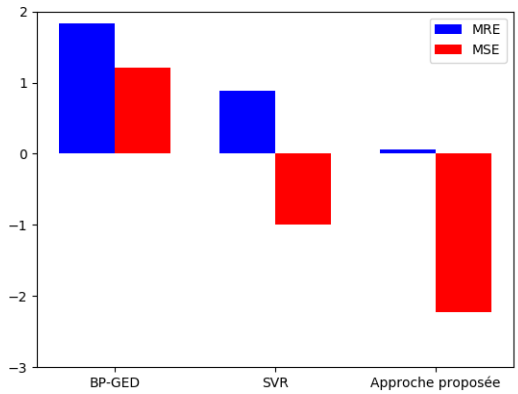


Figure A.10: Obtained results by BP-GED, SVR and our approach on Fingerprint. log_{10} of MRE and MSE is computed for a better readability.

Table A.2 presents the results obtained with MRE and MSE for 3 methods: BP-GED, SVR and our approach. Figure A.10 shows those results in a graphical way. For better readability, log_{10} of the results is shown.

We can see that both learning methods allows a clear increase of accuracy compared to BP-GED on this dataset. This might be explained by the fact that most of the features are contained into edges, which is not easily taken into account by BP-GED. Our approach also obtain better results than SVR. As for the Letter dataset, this might be explained by the fact the matrix C embed more information than the couple made of upper and lower bounds.

A.4 Conclusion

In this chapter, we presented a new approach to approximate the graph edit distance. Our method is based on the use of convolutional networks and Spatial Pyramid Pooling. The use of both methods allows the feature extraction from the cost matrix, despite its variable size.

Our approach is evaluated on two different datasets. It presents good performance on both datasets, with better results than the only other method based on machine learning. Those results could be improved, by a network optimization for example. However, the proposed approach does not provide an edit path, which could be interesting for some applications.

Bibliography

[1] Pubmed. https://www.ncbi.nlm.nih.gov/pubmed/. Accessed: 2018-11-28.

[2] Zeina Abu-Aisheh, Benoit Gaüzère, Sébastien Bougleux, Jean-Yves Ramel, Luc Brun, Romain Raveaux, Pierre Héroux, and Sébastien Adam. Graph edit distance contest: Results and future challenges. *Pattern Recognition Letters*, 100:96–103, 2017.

[3] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. *arXiv preprint arXiv:1905.00067*, 2019.

[4] Mark A Aizerman. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and remote control*, 25:821–837, 1964.

[5] Han Altae-Tran, Bharath Ramsundar, Aneesh S Pappu, and Vijay Pande. Low data drug discovery with one-shot learning. *ACS central science*, 3(4):283–293, 2017.

[6] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1993–2001, 2016.

[7] Pranjal Awasthi and Reza B Zadeh. Supervised clustering. In *Advances in neural information processing systems*, pages 91–99, 2010.

[8] G. Sanniti Di Baja and E. Thiel. Skeltonization algorithm running on path-based distance maps. *Image and Vision Computing*, 14:47–57, 1996.

[9] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and

physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.

[10] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[11] Karsten Borgwardt, Elisabetta Ghisu, Felipe Llinares-López, L O'Bray, and Bastian Rieck. Graph kernels. *Foundations and Trends in Machine Learning*, 13(5-6):531–712, 2020.

[12] Sébastien Bougleux, Benoit Gaüzere, and Luc Brun. Graph edit distance as a quadratic program. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 1701–1706. IEEE, 2016.

[13] Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

[14] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[15] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[16] Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

[17] Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287*, 2018.

[18] C.-J. C. Fu Chang and C.-J. Lu. A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision and Image Understanding*, 93:206–220, 2004.

[19] LC Chen, G Papandreou, I Kokkinos, K Murphy, and AL Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. arxiv 2016. *arXiv preprint arXiv:1606.00915*, 2016.

[20] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[21] François Chollet et al. keras, 2015.

[22] Fan RK Chung and Fan Chung Graham. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

[23] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[24] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[25] Floris Geerts. On the expressive power of linear algebra on graphs. *Theory of Computing Systems*, 65(1):179–239, 2021.

[26] C Lee Giles, Kurt D Bollacker, and Steve Lawrence. Citeseer: An automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*, pages 89–98. ACM, 1998.

[27] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

[28] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 2, pages 729–734. IEEE, 2005.

[29] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[30] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[31] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[32] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[33] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[35] Yotam Hechtlinger, Purvasha Chakravarti, and Jining Qin. A generalization of convolutional neural networks to graph-structured data. *arXiv preprint arXiv:1704.08165*, 2017.

[36] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[37] Matthias Holschneider, Richard Kronland-Martinet, Jean Morlet, and Ph Tchamitchian. A real-time algorithm for signal analysis with the help of the wavelet transform. In *Wavelets*, pages 286–297. Springer, 1990.

[38] 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 preprint arXiv:2005.00687*, 2020.

[39] Derek Justice and Alfred Hero. A binary linear programming formulation of the graph edit distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(8):1200–1214, 2006.

[40] Andrew Kachites, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning.

[41] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016. `http://graphkernels.cs.tu-dortmund.de`.

[42] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[43] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[44] Boris Knyazev, Xiao Lin, Mohamed R Amer, and Graham W Taylor. Spectral multigraph networks for discovering and fusing relationships in molecules. *arXiv preprint arXiv:1811.09595*, 2018.

[45] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[46] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[47] John Boaz Lee, Ryan A Rossi, Sungchul Kim, Nesreen K Ahmed, and Eunyee Koh. Attention models in graphs: A survey. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 13(6):1–25, 2019.

[48] Julien Lerouge, Zeina Abu-Aisheh, Romain Raveaux, Pierre Héroux, and Sébastien Adam. Graph edit distance: a new binary linear programming formulation. *arXiv preprint arXiv:1505.05740*, 2015.

[49] Julien Lerouge, Zeina Abu-Aisheh, Romain Raveaux, Pierre Héroux, and Sébastien Adam. Exact graph edit distance computation using a binary linear program. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 485–495. Springer, 2016.

[50] Julien Lerouge, Maroua Hammami, Pierre Héroux, and Sébastien Adam. Minimum cost subgraph matching using a binary linear program. *Pattern Recognition Letters*, 71:45–51, 2016.

[51] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109, Jan 2019.

[52] Ron Levie, Elvin Isufi, and Gitta Kutyniok. On the transferability of spectral graph filters. *arXiv preprint arXiv:1901.10524*, 2019.

[53] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. Adaptive graph convolutional neural networks. *arXiv preprint arXiv:1801.03226*, 2018.

[54] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[55] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[56] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *arXiv preprint arXiv:1905.11136*, 2019.

[57] James Mercer. Xvi. functions of positive and negative type, and their connection the theory of integral equations. *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character*, 209(441-458):415–446, 1909.

[58] Fernand Meyer. Topographic distance and watershed lines. *Signal processing*, 38(1):113–125, 1994.

[59] W Thomas Miller, Paul J Werbos, and Richard S Sutton. *Neural networks for control*. MIT press, 1995.

[60] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017.

[61] Federico Monti, Oleksandr Shchur, Aleksandar Bojchevski, Or Litany, Stephan Günnemann, and Michael M. Bronstein. Dual-primal graph convolutional networks, 2018.

[62] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[63] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023, 2016.

[64] Kaspar Riesen and Horst Bunke. Iam graph database repository for graph based pattern recognition and machine learning. In Niels da Vitoria Lobo, Takis Kasparis, Fabio Roli, James T. Kwok, Michael Georgiopoulos,

Georgios C. Anagnostopoulos, and Marco Loog, editors, *Structural, Syntactic, and Statistical Pattern Recognition*, pages 287–297, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[65] Kaspar Riesen and Horst Bunke. Classification and clustering of vector space embedded graphs. In *Emerging topics in computer vision and its applications*, pages 49–70. World Scientific, 2012.

[66] Kaspar Riesen, Miquel Ferrer, and Horst Bunke. Approximate graph edit distance in quadratic time. *IEEE/ACM transactions on computational biology and bioinformatics*, 17(2):483–494, 2015.

[67] Kaspar Riesen, Andreas Fischer, and Horst Bunke. Computing upper and lower bounds of graph edit distance in cubic time. In *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pages 129–140. Springer, 2014.

[68] Kaspar Riesen, Andreas Fischer, and Horst Bunke. Estimating graph edit distance using lower and upper bounds of bipartite approximations. *International Journal of Pattern Recognition and Artificial Intelligence*, 29(02):1550011, 2015.

[69] Kaspar Riesen, Michel Neuhaus, and Horst Bunke. Bipartite graph matching for computing the edit distance of graphs. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pages 1–12. Springer, 2007.

[70] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[71] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4510–4520. IEEE Computer Society, 2018.

[72] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *Trans. Neur. Netw.*, 20(1):61–80, January 2009.

[73] Chao Shang, Qinqing Liu, Ko-Shin Chen, Jiangwen Sun, Jin Lu, Jinfeng Yi, and Jinbo Bi. Edge attention-based multi-relational graph convolutional networks. *arXiv*, pages arXiv–1802, 2018.

[74] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30(3):83–98, 2013.

[75] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[76] M. Teague. Image analysis via the general theory of moments. *Journal of the Optical Society of America*, 70(8):920–930, 1980.

[77] O. R. Terrades, S. Tabbone, and E. Valveny. A review of shape descriptors for document analysis. In *Proceedings of the ninth International Conference on Document Analysis and Recognition*, pages 227–231, 2007.

[78] Yizhou Sun Ting Chen, Song Bian. Dissecting graph neural networks on graph classification. *CoRR*, abs/1905.04579, 2019.

[79] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 1(2), 2017.

[80] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. Kgat: Knowledge graph attention network for recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 950–958, 2019.

[81] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, pages 2022–2032, 2019.

[82] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7794–7803, 2018.

[83] B Weisfeiler and A Leman. The reduction of a graph to canonical form and the algebgra which appears therein. *NTI, Series*, 2, 1968.

[84] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

[85] 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. *Chemical science*, 9(2):513–530, 2018.

[86] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[87] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[88] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning, ICML'16*, 2016.

[89] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pages 4800–4810, 2018.

[90] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. In *Conference on Uncertainty in Artificial Intelligence, UAI*, 2018.

[91] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[92] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[93] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

[94] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, 2017.