POLITECNICO DI TORINO

Master of Science in Mathematical Engineering

Master Thesis

Graph neural networks for classification: models and applications

Supervisors:

Prof. Pietro Liò

Prof. Elisa Ficarra

Candidate:

Chiara Sopegno

March 2020

A chi mi ha sostenuta in questi anni

Abstract

Graph neural networks have emerged in the past years as very promising methods for the analysis of graph-structured data. Useful insights can in fact be extracted by allowing learning models to take into account relationships between entities in a graph. The main methods used in the context of graph neural networks are here described and compared, with a focus on the extension of convolutional layers to graph structures. Afterwards, an analysis of how attention mechanisms integrate with graph neural networks is introduced. In this context a new architecture is proposed, with a self-attentive mechanism that allows a graph neural network to attend over its own input in the context of graph classification. An application of these methods to biomedical data is finally presented, with an example in the field of Parkinson's disease classification.

Contents

Chapter 1

Introduction

The field of deep learning has become very popular in the past years, bringing to the creation of neural network models capable of reaching the state-of-the-art performance in many machine learning tasks. The majority of deep learning methods analyses data that naturally lie in euclidean spaces. However, in many learning tasks, the input data inherently contain relationships between their entities. This is for example the case of users in social networks, neurons in brain connectomes or interacting particles in physical systems, all of which constitute systems of naturally interconnected items. By disregarding the connectivity between entities in such networks, relevant information can be lost during the automated learning process. For this reason, new models have been introduced to take into account the underlying structure of this kind of data. The architectures of these models, called graph neural networks, are built directly on graph structures, where nodes represent the analyzed items and edges encode information about the relationships between them.

Graph neural networks have first been introduced by [41] as an extension of recursive neural networks, but recently the field expanded with several models extending the concepts of convolutional neural networks to graph-structured data. Useful insights can in fact be extracted by allowing learning models to take into account relationships between the items in input and capture this information through a message-passing framework between the nodes of the graphs.

At the same time, the mechanism of *attention* in deep learning models has recently started to gain increasing popularity. Initially introduced in the context of neural machine translation, it is starting to be integrated in other fields such as computer vision. The basic idea behind these mechanisms is to allow the models to focus on the most important part of the input during training, while disregarding underlying noise in the signal. An important advantage of such mechanisms lies in the interpretability of the resulting models. It would be therefore desirable to introduce the characteristics of attentive mechanisms also in the context of signals defined on non-euclidean domains.

The contribution of this work is three-fold: firstly, an analysis of the main methods introduced in the context of graph neural networks is presented, with a systematic categorization of convolutional, pooling and attention mechanisms; in second place, a new architecture is proposed, with a self-attentive mechanism that allows a graph neural network to attend over its own input in the context of graph classification; finally, an application of these methods to biomedical data is presented, with examples in the field of Parkinson's disease classification.

The second chapter gives a background of the main methods that have been developed in the field of deep learning in the past years. A first section

describes in a schematic way what convolutional neural networks are, by describing the main concepts and structures. In the second section, instead, an introduction is given to what attention mechanisms are and what contributions they can bring in many different contexts, ranging from natural language processing to computer vision.
The third chapter gives a complete overview of state-of-the-art methods in the field of graph neural networks. Extensions of convolutional and pooling layers to graph-structured data are explained in details and advantages and disadvantages of each method are analyzed. The emerging field of attentive mechanisms in graph domains is then explored, analyzing both node-focused and graph-focused structures. Finally, a new framework for the application of self-attentive strategies when trying to classifying an entire graph is proposed.
The forth chapter focuses on the application of these methods to data concerning patients with Parkinson's disease, especially considering the problem of classifying data coming from the imaging and genomic fields. After a general introduction to the disease and to the data modalities taken into consideration, which include DNA methylation data and SPECT images of the brain, a description of the dataset coming from the PPMI (Parkinson's Progression Markers Initiative) is given. The disease classification problem is then re-framed as a graph classification instance and the application of the described models is analyzed.

Chapter 2

Machine learning background

2.1 Convolutional Neural Networks

Artificial neural networks are machine learning techniques that allow to create models mapping variables between an input space and an output space. Convolutional neural networks (CNNs) are a specific kind of neural networks commonly used in the fields of image and video analysis, as well as in natural language processing. The basic structure of a CNN consists of a sequence of layers of different type [17]:

- *Convolutional layer*: the characteristic layer of a CNN. It takes as input a tensor of numbers (e.g. pixel values of an image) and convolves it with one or more smaller tensors of parameters, called filters. The operation consists in sliding the parameter tensor along the input: in every position, the element-wise multiplication between the filter and the corresponding part of the input is computed, and the resulting values are added together. The final values are then inserted in order in the output tensor, also called feature map. Intuitively, for the case of images, the filters act as feature extractors, detecting edges, shapes, lines and textures, according to the values of their parameters.

- *Activation layer*: the layer that introduces non-linearity inside the neural network model, applying a non-linear function on the feature map. The most used ones are $ReLU$, hyperbolic tangent and sigmoid function.

- *Pooling layer*: the layer that reduces the size of the feature map, by retaining only the most important information in a spatial neighbourhood of its input. The most common pooling strategies usually apply simple mathematical functions, such as summing the elements in the selected patch or taking the average between them.

- *Dense layer*: a standard feed-forward neural network layer, in which every input element is connected with every output.

- *Output layer*: the last layer of a CNN architecture. In the context of multi-class recognition, it has as many outputs as the number of classes. It usually applies the $SoftMax$ function in order to return class probabilities.

Formally, in the context of classification, the structure of the main body of a CNN can be summarized in the following framework, introduced in [6]. If we consider a compact Euclidean domain $\Omega \in \mathbb{R}^d$ and a set of functions in $L^2(\Omega)$, the objective is to find an approximation of a function $y : L^2(\Omega) \rightarrow Y$, where the target space Y is discrete and $|Y|$ is equal to the number of classes. In this context, the main body of a CNN can be described as follows:

$$U_\Theta(f) = (C_{\Gamma(K)} \circ ... \circ P \circ ... \circ C_{\Gamma(2)} \circ C_{\Gamma(1)})(f) \tag{2.1}$$

where $f(x) = (f_1(x), \dots, f_p(x))$ is the p-dimensional input to the network, with $f \in L^2(\Omega)$, C and P denote convolutional and pooling layers respectively, and $\Theta = (\Gamma(1), \dots, \Gamma(K))$ is the hyper-vector of network parameters. The generic convolutional layer $g = C_\Gamma(f)$, with compactly supported filters $\Gamma = (\gamma_{l,l'})$, can be described by the following equation:

$$g_l(x) = \xi\left(\sum_{l'=1}^{p} f_{l'} \star \gamma_{l,l'}(x)\right), \qquad l = 1, \dots, p, \quad l' = 1, \dots, q \tag{2.2}$$

where $x \in \Omega$ are the spatial coordinates, ξ is a non-linearity and:

$$f \star \gamma = \int_\Omega f(x - x')\gamma(x')dx' \tag{2.3}$$

denotes the standard convolution. It has to be noted that in standard CNNs the filters are not actually reversed, so what is applied is a cross-correlation and not a standard convolution. However, the two operations are equivalent when applied with filters that are one the reversed of the other, and since in CNNs the filters are learnt in the process, there's not an actual difference between them. Following (2.2), many feature maps are produced, resulting in a q-dimensional output $g(x) = (g_1(x), \dots, g_p(x))$ over the domain Ω.

For what concerns the pooling layer $g = P(f)$, it can instead be generally described by:

$$g_l(x) = h(\{f_l(x') : x' \in N(x)\}) \quad , \quad l = 1, \dots, q \tag{2.4}$$

where h is a permutation-invariant function applied to a neighbourhood $N(x)$ of x. Examples of permutation-invariant function are the L_p norms, which result in the application of average-pooling for $p = 1$ and max-pooling for $p = \infty$.

It's important to note that (2.1) describes the main structure of a CNN, however sometimes the convolutional layers are followed by normalization layers that allow to improve the performance. Moreover, as previously said, the results given by (2.1) is then fed into one or more final fully-connected layers in order to perform the final classification.

Moving on to the learning process, the training of a convolutional neural network is characterized by different steps. The first step consists in the initialization of the network parameters, which are usually initialized using small random values. Subsequently an iterative process starts, which involves the following steps:

- *Forward propagation*: an input sample (or a batch of them) is shown to the network, which computes the intermediate variables layer by layer and in the last layer outputs the probabilities of class membership $\hat{y}^{(i)}$.
- *Loss evaluation*: given the i-th sample, the probabilities of class membership $\hat{y}^{(i)}$ are compared with the one-hot encoding $y^{(i)}$ of the true label, in order to compute the loss function. A common example is the cross-entropy loss function:
$$L = \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -(y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})) \tag{2.5}$$
- *Back-propagation of the error*: the derivatives of the function L with respect to the network parameters are computed by applying recursively the chain rule. This process provides the local gradient of L in the current parameter configuration $\bar{w}$:
$$\nabla L(\bar{w}) = \left(\frac{\partial L(\bar{w})}{\partial w_1}, \ldots, \frac{\partial L(\bar{w})}{\partial w_m} \right) \tag{2.6}$$
- *Parameters update*: after back-propagating the error, the weights are updated following a gradient descent formula:
$$w_{t+1} = w_t - \eta \nabla L(w_t) \tag{2.7}$$

where η is a hyper-parameter controlling the length of the step.

The training process usually involves many iterations, with the goal of finding a configuration of the parameters space that allows to minimize the loss function. Each complete presentation of the dataset to the network is called *epoch*. In most cases it has been observed that presenting the input samples to the network in batches helps to stabilize the training procedure and improves overall results.

The success of CNNs is primarily due to their ability to exploit the stationarity, locality and intrinsic hierarchical structure of the data [6].
In the visual domain, stationarity refers to the fact that many of the properties of an image are shift-invariant. Therefore, the use of filters that are convolved across all the input allows to recognize identical features independently of their spatial locations. This aspect allows therefore to share weights across neurons, thus diminishing the number of parameters and reducing the risk of overfitting.
Secondly, in images small sets of nearby adjacent pixels are able on their own to encode significant information for the downstream task. This allows the use of filters whose support is much smaller than the size of the input, further diminishing the number of required parameters.
Moreover, by using a multi-layer structure that alternates pooling and convolutional operators, CNNs are able to learn multi-scale latent features in a hierarchical way. This last aspect works particularly well in capturing the intrinsic hierarchical structure of data like images, allowing the detection of different aspects of the image at different layers of the CNN: the first layers will detect edges and simpler shapes, while the last ones will be able to detect entire objects and perform classification.

2.2 Attention Mechanism

The idea behind any attention mechanism is to help a learning model to understand which elements of the input are more relevant for the task at hand. This aspect is generally addressed by learning a vector of attention scores for each element in the input and using them to modulate the flow of information from the input signal.

This idea was first introduced in the context of neural machine translation by [3], to cope with the loss of information in encoder-decoder networks when the input was a long sentence. In fact, while previous methods tried to encode the whole input sentence into a fixed-length vector, the method proposed in [3] uses a sequence of different latent vectors, which are then fed to the attention mechanism in the decoder step. In this way the decoder has the ability to choose adaptively which ones of the learnt embeddings are more significant for constructing the correct translation.

More precisely, if we call $\{h_1, \dots, h_T\} \in \mathbb{R}^d$ the encoded embeddings, the implementation of this mechanism consists in computing a context vector c_i as the weighted sum of the embeddings:

$$c_i = \sum_{j=1}^{T} \alpha_{ij} h_j \tag{2.8}$$

where the attention coefficients α_{ij} are computed based on an alignment model between the input and the output sentence. Each α_{ij} should in fact measure how well the latent embedding h_j is *aligned* with an encoding of the output sentence up until position i, denoted by $s_{i-1} \in \mathbb{R}^d$. The attention coefficients can be computed by:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})} \tag{2.9}$$

where the inputs e_{ij} to the softmax function are the alignment coefficients [3]:

$$e_{ij} = a(s_{i-1}, h_j) \tag{2.10}$$

The alignment model a can be parametrized by a feedforward neural network taking as input a linear combination of s_{i-1} and h_j, in what is called *additive attention*. However, recently some works proposed a *multiplicative attention* method, that uses as alignment model the dot product between the two inputs [46]. It should be noted that the alignment model can be trained together with the whole neural network, thus allowing great flexibility in the model.
Interestingly, in the context of neural machine translation, if we consider the coefficient α_{ij} to be the probability that s_{i-1} is aligned with the embedding h_j, then the context vector c_i can be interpreted as an expected value of the embeddings over the possible alignments [3].

Equation (2.10) applies the alignment model to vectors that are computed from the input and output sentences by using recurrent neural networks. However, it's also possible to apply a *self-attention* mechanism by computing the alignment model directly between different positions of input and output sequences, as proposed in the *Transformer* model by [46]. They propose in fact to separate the output sequence into a set of *queries*, represented by a matrix $Q \in \mathbb{R}^{q \times d}$, and the input one into a set of *keys*, encoded in $K \in \mathbb{R}^{k \times d}$, where q is the number of queries, k the number of keys and d is the shared dimension of queries and keys. The attention scores are computed through the alignment model:

$$A = \text{SoftMax}\left(\frac{QK^T}{\sqrt{d}}\right) \tag{2.11}$$

in a scaled multiplicative attention formulation. If we then consider the *values* $V \in \mathbb{R}^{k \times v}$, we can use the attention scores to modulate their information through $C = AV$, where C is the matrix containing the context vectors. The values are often computed through a transformation of the keys themselves. This formulation allows to avoid costly operations derived from the computation of embeddings through RNNs and yields very good experimental results. Another improvement in attention models proposed by [46] is what is called *multi-head attention*: instead of computing only one alignment model over the queries and keys proposed, it is in fact possible to compute different models, taking as inputs learned linear projections of those same queries and keys. The attention mechanisms are then computed in parallel and the learnt context vectors can be concatenated to form a unique output of the multi-head attention layer. Through this formulation it's possible to increase the expressive capacity of the model [46].

Despite being introduced in the context of neural machine translation, attentive models have also been exploited in the context of computer vision. It is in fact possible to use the attention mechanism to adaptively select a portion of an image, in order to disregard the non-relevant parts and diminish both noise and number of parameters required. A first attempt was made by constructing an RNN model which processed the input by sequentially fixating on different parts of the image and, then, incrementally combined the information by giving different weights to embeddings coming from different regions of the image [35].
More recent approaches integrate instead attention mechanism in convolutional neural networks through the use of learnable attention maps. Traditionally, attention maps have been extracted from intermediate layers of

CNNs after these were fully trained. The objective was to understand the relative importance that features in a layer had in the creation of the following feature maps. However, the use of attention in a post-hoc formulation doesn't let it participate in the learning process in order to improve it.

A method that allows to integrate attention into training is proposed by [22], where attention scores are learnt during training and used to create a convex combination of the feature vectors in a specific layer. Their method for computing attention scores proposes to re-purpose the global image representation, computed by the network itself in its last layers, as a query that attends over the feature vectors of intermediate layers, considered as keys. The basic idea of this mechanism is summarized in figure 2.2.
We can denote the global feature vector as $g \in \mathbb{R}^d$ and by $\mathcal{L}^s = \{l_1^s, \ldots, l_n^s\} \in \mathbb{R}^d$ the set of local feature vectors extracted at a given layer $s \in S$, where n is the total number of spatial locations at layer s and S is a subset of layers of interest. For computing the attention map at layer s, we first need to compute each alignment score $e_i^s \in \mathbb{R}$ between g and each one of the feature vectors l_i^s through $e_i^s = a(g, l_i^s)$, where the alignment model a can be both multiplicative or additive. The alignment scores are then normalized following:

$$\alpha_i^s = \frac{\exp(e_i^s)}{\sum_{j=1}^{n} \exp(e_j^s)} \tag{2.12}$$

in a layer-wise formulation. Equation (2.12) generates the values in the attention map $\mathcal{A} = \{\alpha_1^s, \ldots, \alpha_n^s\}$, which will be used to filter the importance of each local feature vector in layer s. For each layer of interest, a vector representation is produced by:

$$g_a^s = \sum_i \alpha_i^s l_i^s \tag{2.13}$$

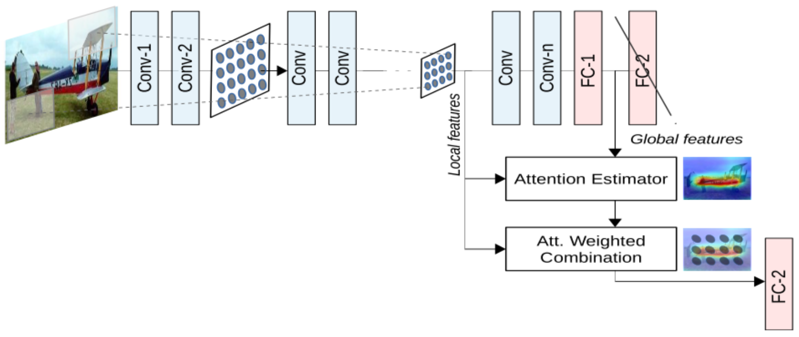


Figure 2.1: Overview of the attention mechanism proposed by [22]

and the concatenation $||_{s \in S} \, g_a^s$ substitutes the vector g as global representation of the image.

This method allows for an integration of global and local features, attending one over the others and thus creating a multi-scale attention framework. The attention scores represent how well a region of the intermediate feature map relates to the global representation of the image itself, and enables the learning process to amplify the influence of the most significant regions, discarding the information coming from noisy regions. Moreover, the use of the attention mechanism at different layers allows to focus on different spatial resolutions.

Chapter 3

Methods

3.1 Graph Neural Networks

Convolutional neural networks represent the state-of-the-art methods for data defined on grid-like Euclidean structures, like images and videos. However, many real-world scenarios are represented using data that are naturally defined in non-Euclidean spaces, such as graphs and manifolds [6]. Some examples of graph-structured data can be found in social networks, transportation networks, brain connectomes and other kinds of biological networks. However, generalizing convolutional neural networks to graphs is not easy, since convolutional and pooling operators are defined on regular grids. The first approaches to applying deep learning to these kind of structured data required a pre-processing phase in which the topological structure of the input was encoded in simpler structures (e.g. vectors of reals). However, for what concerned these methods, the quality of the results was deeply linked with the quality of the encoding and, in the absence of a deep understanding of the data domain, a considerable amount of the topological information

could be lost in the process. Moreover, the hand-engineering of features is simply not feasible when dealing with high-dimensional data.
Graph neural networks have first been presented in [18] and [41] as an extension of recurrent neural networks. Recurrent neural networks are in fact a type of neural network whose inner structure is composed by a directed acyclic graph, so their basic scheme can be extended to more general kinds of graph structures. The method was based on the repeated application of contraction maps, until the states of nodes reached a stable fixed point. Subsequently the state vectors of nodes were fed into a feed-forward neural network to perform classification [41]. More recently instead, graph neural networks have been re-framed as an extension of CNNs to graph domains, by defining convolution and pooling operations to graph structures.

Formally, a graph can be defined as $G = (V, E, W)$, where V is a finite set of n nodes, E is the set of edges between nodes and $W \in \mathbb{R}_+^{V \times V}$ is a weighted adjacency matrix. Edges are described as ordered pairs of nodes $(i, j) \in V \times V$, while the weight matrix is a non-negative matrix whose elements satisfy the conditions:

$$\begin{aligned} W_{ij} &= 0 \quad \text{if} \quad (i,j) \notin E \\ W_{ij} &> 0 \quad \text{if} \quad (i,j) \in E \end{aligned} \tag{3.1}$$

The set of nodes represents the units involved in the graph, for example people in social networks or interacting particles in a physical system. Edges represent instead specific relationships between the defined entities, though they may have different interpretation depending on the context. For what concerns instead the optional weight W_{ij} associated to an edge (i, j), it usually defines the strength of that specific relationship.

The amount of works that has been proposed in the field of deep learning applied to graphs can be generally divided into two categories [41]:

- *node-focused applications*, where the final task is associated with individual properties of each node (examples include semi-supervised node classification, link prediction and node clustering)

- *graph-focused application*, where the model to be constructed is dependent on the whole graph structure (examples include graph classification, graph generation and estimation of properties of graphs)

Since images can be considered as graphs with a fixed euclidean grid, we can re-frame tasks like image classification as an example of graph classification, while object localization and image-segmentation are examples of node-focused applications, where the nodes are the pixels of the image.
In the context of this thesis we will focus on the second class of problems, the one regarding graph focused applications and specifically graph classification.

In [7], graph neural network have been introduced in a CNN-like structure for the first time. First of all, this involves giving a definition of what it means to apply convolutions on graph structures data. The propagation of information across elements is in fact at the basis of any convolutional neural network. Then, even if not strictly necessary, an extension of the idea of pooling layer should also be given, in order to allow the network to also exploit a possible hierarchical structure of the data. Batch normalization and activation layers can instead be trivially extended from the ones used in standard convolutional neural networks. Moreover, the training can be performed by the use of the back-propagation algorithm with chosen task-specific losses.

3.1.1 Graph Convolutional Layers

The first attempt to generalize convolutions to graph structures takes inspiration from harmonic analysis on graphs [7] and it requires to introduce some concepts from spectral graph theory in order to be described.
Given a graph structure $G = (V, E, W)$ with n nodes, we can consider the Hilbert spaces $L^2(V)$ and $L^2(E)$ defined by the following inner products:

$$\langle f, g\rangle_{L^2(V)} = \sum_{i\in V} f_i g_i \tag{3.2}$$

$$\langle F, G\rangle_{L^2(E)} = \sum_{e\in E} w_e F_e G_e \tag{3.3}$$

with $f, g : V \to \mathbb{R}$ and $F, G : E \to \mathbb{R}$. By considering two functions $f \in L^2(V)$ and $F \in L^2(E)$, we can then define the graph gradient as the operator $\Delta : L^2(V) \to L^2(E)$ such that:

$$(\Delta f)_{(i,j)} = f_i - f_j \tag{3.4}$$

and the graph divergence div : $L^2(E) \to L^2(V)$ expressed by:

$$(\mathrm{div}F)_i = \sum_{j:(i,j)\in E} w_{ij} F_{(i,j)} \tag{3.5}$$

It follows that the graph Laplacian $\nabla = -\mathrm{div}\Delta$ is an operator $\nabla : L^2(V) \to L^2(V)$ such that:

$$(\nabla f)_i = \sum_{(i,j)\in E} w_{ij}(f_i - f_j) \tag{3.6}$$

If we apply the graph Laplacian operator to a signal $x = (x_1, ..., x_n)^T \in \mathbb{R}^n$ such that $x \in L^2(V)$, we see that (3.6) can be rewritten in the following matrix formulation:

$$\nabla x = (D - W)x \tag{3.7}$$

where $D \in \mathbb{R}^{n \times n}$ is the diagonal matrix with $D_{ii} = \sum_j w_{ij}$. We can then define the combinatorial Laplacian matrix as $L = D - W$, which is an important operator in graph spectral analysis. Alternatively, the normalized definition of the Laplacian matrix $\mathcal{L} = I - D^{-1/2} W D^{-1/2}$ can be used. Both definitions of the graph Laplacian are generalization of the Laplacian defined in Euclidean spaces.

Given a signal $x \in \mathbb{R}^n$ over the nodes of the graph, we can define the smoothness functional $\|\nabla x\|_W$ as:

$$\|\nabla x\|_W = \sum_i \|\nabla x\|_W (i) = \sum_i \sum_j w_{ij} [x(i) - x(j)]^2 \tag{3.8}$$

where the notation $\|\nabla x\|_W (i)$ denotes the smoothness functional at node i. Given this definition, we have that:

$$u_0 = \underset{x \in \mathbb{R},\ \|x\|=1}{\arg\min} \|\nabla x\|_W^2 = (1/\sqrt{n}) \mathbb{1}_n$$

is the smoothest signal on the graph structure (it is in fact a constant over the nodes), while the other eigenvectors of L are in general given by:

$$u_i = \underset{x \in \mathbb{R},\ \|x\|=1,\ x \perp \{u_0, \ldots, u_{i-1}\}}{\arg\min} \|\nabla x\|_W^2$$

Since L is a symmetric positive semidefinite matrix, it is possible to determine the smoothness of the signal x by reading its coefficients in the basis $\{u_i\}_{i=0}^{n-1}$ of orthonormal eigenvectors. These vectors are also called the Fourier modes of the graph and their associated non-negative eigenvalues $\{\lambda_i\}_{i=0}^{n-1}$ identify the frequencies of the graph.

The graph Fourier transform is defined as the projection onto the eigenbasis of the graph Laplacian operator:

$$\hat{x} = U^T x$$

while its inverse will be given by $x = U\hat{x}$. The coefficients of a signal x in the eigenvector basis are the equivalent of the Fourier coefficients of signals defined on grids. Moreover, the Fourier basis can also be used to diagonalize the Laplacian itself as follows:

$$L = U\Lambda U^T, \quad U = [u_0, \ldots, u_n] \in \mathbb{R}^{n \times n}, \quad \Lambda = \text{diag}\{\lambda_0, \ldots, \lambda_n\} \in \mathbb{R}^{n \times n}$$

Since it's difficult to extend the traditional definition of convolution, based on the existence of a translation operator, on the vertex domain of the graph, the idea introduced in [7] is to define the convolution operation on the spectrum of the weights. The spectral convolution of two signals $x, y \in \mathbb{R}^n$ on a graph G can then be defined as:

$$x \star y = U\left(\left(U^T x\right) \odot \left(U^T y\right)\right) = U\text{diag}(\hat{y_1}, \ldots, \hat{y_n})\hat{x} \tag{3.9}$$

where $\odot$ is the Hadamard product. Equation (3.9) represents the equivalent of the *convolution theorem* defined in Euclidean domains. A signal x can then be filtered following:

$$x' = g_\theta \star x = UG(\Lambda)U^T x \tag{3.10}$$

where $G(\Lambda)$ is a diagonal matrix of learnable filters in the frequency domain. If we assume that the input signal at a specif layer k is given by a matrix $X^{(k)} \in \mathbb{R}^{n \times f_k}$, where f_k is the number of features of each node at that layer, then the rows of the feature matrix at the next layer $k+1$ can be computed as:

$$x_j^{(k+1)} = \xi\left(U \sum_{i=1}^{f_k} G_{i,j}^{(k)} U^T x_i^{(k)}\right), \quad j = 1, \ldots, f_{k+1} \tag{3.11}$$

where ξ is a non-linear function and each $G_{i,j}^{(k)}$ is a diagonal matrix of filters that can be learnt during the back-propagation process. We observe that the resulting feature matrix $X^{(k+1)}$ will have dimensions $n \times f_{k+1}$. The idea is similar to classical CNNs: the input signals are passed through these learnable filters, which aggregate the information, and subsequently the non-linearity ξ is applied to the output.

However, often only the smoothest part of the signal is useful in the analysis, since it is the one that is less likely to carry signal noise. It can therefore be useful to consider only the first d eigenvectors of the Laplacian and substitute U with $U_d = [u_0, \dots, u_d] \in \mathbb{R}^{n \times d}$. The cutoff value d depends on the specif application, on the regularity of the graph and on the sample size. It is important to notice that often the single high frequency eigenvectors do not bring much information, but we can obtain much more information by allowing the network to combine them together.

We have seen that equation (3.11) allows us to define a translation operation in the spectral domain. By defining the convolution on this domain, we are now able to allow weight sharing across different locations of the graph, however the filters used aren't spatially localized. It can be observed that, in the euclidean grid, the spatial localization is translated into smoothness in the spectral domain. So, in order to learn filters that are actually localized in the original spatial domain, it is necessary to learn spectral multipliers that give a smooth signal in the Fourier domain. This can be done also in graphs, by selecting only a subset of multipliers and using interpolation kernels to obtain the rest, thus allowing localization of the filters in the spatial domain. In particular the smooth filters can be described by:

$$\mathrm{diag}(G_{i,j}^{(k)}) = B\alpha_{i,j}^{(k)} \tag{3.12}$$

where $B = (b_{l,m}) = (\beta_m(\lambda_l))$ is a $d \times q_k$ matrix of fixed interpolation kernels, for example cubic splines, and $\alpha_{i,j}^{(k)}$ are the q_k interpolation coefficients.

In order to obtain filters with constant spatial support (i.e. independent of the size of the input n), it is necessary to choose a sampling step $\gamma \sim n$ in the spectral domain, thus giving a constant number $q_k \sim n \cdot \gamma^{-1}$ of coefficients per filter [7].
Another limitation of convolutions in the Fourier domain is the $\mathcal{O}(n^2)$ computational cost, which is due to the multiplication with the Fourier basis, that has to be performed twice in the forward and inverse Fourier transform. Moreover, the filters in (3.11) learnt during the process are dependent on the basis U (or U_d) used, and therefore dependent on the specific structure of the graph [50].

An improvement of the previous model is introduced by [11]. In order to solve both the efficiency problem and the non-localization of filters, a solution is the use of polynomial filters:

$$G_\theta(\Lambda) = \sum_{j=0}^{r-1} \theta_j \Lambda^j \tag{3.13}$$

where r is the polynomial order and $\theta \in \mathbb{R}^r$ is a vector of learnable coefficients. It can be proven that such filters are localized in a spatial neighbourhood of radius $r-1$ from the central node in which they are applied [11]. This is intuitive since the Laplacian is an operator that works on a neighbourhood of radius 1 and any j-th power of the diagonalized Laplacian is thus working on a j-hops neighbourhood [6]. Moreover, the number of parameters to be learnt is also dependent on the size r of the filters and thus independent from the input size.
However, by introducing such filters in (3.11), we still encounter the computational complexity brought by the multiplication with the Fourier basis. This problem can be solved through the use of a Chebyshev expansion.

The Chebyshev polynomials are defined by the following recursive formula:

$$\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_k(x) &= 2xT_{k-1}(x) - T_{k-2}(x) \end{aligned} \tag{3.14}$$

and they form an orthogonal basis for $L^2([-1,1], dy/\sqrt{1-y^2})$, i.e. the space of square integrable functions on $[-1,1]$ with respect to the measure $dy/\sqrt{1-y^2}$ [20]. A generic filter can thus be parametrized by the truncated Chebyshev expansion:

$$G_\theta(\Lambda) = \sum_{j=0}^{r-1} \theta_j \mathcal{T}_j(\tilde{\Lambda}) \tag{3.15}$$

where the Chebyshev coefficients $\theta \in \mathbb{R}^r$ are again learnable and the diagonal Laplacian is rescaled through $\tilde{\Lambda} = 2\Lambda/\lambda_{max} - I_n$, so that its elements lie in $[-1,1]$.

With the filters defined in (3.15), the convolution operation (3.11) on a signal $x \in \mathbb{R}^n$ can be rewritten as follows:

$$\begin{aligned} x' = g_\theta \star x = UG_\theta(\Lambda)U^T x &= \sum_{j=0}^{r-1} \theta_j U \mathcal{T}_j(\tilde{\Lambda}) U^T x \\ &= \sum_{j=0}^{r-1} \theta_j \mathcal{T}_j(\tilde{L}) x \\ &= \sum_{j=0}^{r-1} \theta_j \bar{x}_j \end{aligned} \tag{3.16}$$

where $\tilde{L} = 2L/\lambda_{max} - I_n$ is the rescaled Laplacian and the values $\bar{x}_j = \mathcal{T}_j(\tilde{L})x$ can be computed using the recursive relation:

$$\bar{x}_j = 2\tilde{L}\bar{x}_{k-1} - \bar{x}_{k-2} \tag{3.17}$$

with $\bar{x}_0 = x$ and $\bar{x}_1 = \tilde{L}x$. Thanks to the compactly supported filters, we have defined filters which, to be computed, require a computational complexity of $O(|E|)$, i.e linear in the number of edges of the graph. This means that for sparse graphs the computation can be performed in linear time with respect to the input size n. We can thus see how avoiding an explicit use of the Fourier transform brings a great computational advantage.

The filters defined in equation (3.15) can be further simplified in a first-ordered approximation [23]. In fact, starting from equation (3.16), if we consider the expansion only until $r = 2$, we obtain a function that is linear with respect to $\tilde{L}$. Moreover, if we consider the approximation $\lambda_{max} \sim 2$, we get a filter g_θ leading to the following convolution:

$$g_\theta \star x \approx \theta_0 x + \theta_1 (L - I_n) x = \theta_0 x - \theta_1 D^{-1/2} W D^{-1/2} x \tag{3.18}$$

with parameters that are shared across the different locations of the graph. Since we have taken a further approximation of the Chebyshev expansion, these filters will aggregate the signal only across a 1-hop neighbourhood of the graph. Nevertheless, the consecutive application of different filters will allow to aggregate information from a larger neighbourhood.
If we further introduce the constraint $\theta^* = \theta_0 = -\theta_1$, we obtain the following filter:

$$g_\theta \star x = \theta^* (I_n + D^{-1/2} W D^{-1/2}) x \tag{3.19}$$

However, since the matrix $I_n + D^{-1/2} W D^{-1/2}$ has eigenvalues in $[0, 2]$, we can encounter problems of vanishing/exploding gradients. For this reason, a renormalization of W is introduced:

$$g_\theta \star x = \theta^* (\tilde{D}^{-1/2} \tilde{W} \tilde{D}^{-1/2}) x \tag{3.20}$$

where $\tilde{W} = W + I_n$ and $\tilde{D} = \sum_j \tilde{W}_{ij}$.
If we consider a multi-channel signal $X^{(k)} \in \mathbb{R}^{n \times f_k}$ defined on the graph at layer k of the neural network, we can then compute the signal at the next layer generalizing the previous formula:

$$X^{(k+1)} = \xi(\tilde{D}^{-1/2}\tilde{W}\tilde{D}^{-1/2}X^{(k)}\Theta) \tag{3.21}$$

where $\Theta \in \mathbb{R}^{f_k \times f_{k+1}}$ is the matrix of filter parameters, with f_{k+1} output channels.
It should be noted that, when constructing the new feature matrix with this type of convolutions, we give same importance to the features of the node itself and to the ones of its neighbours. However, in some applications, it might be useful to introduce a trade-off parameter μ in the definition of $\tilde{W}$:

$$\tilde{W} = W + \mu I_n$$

where μ is also learnable through the gradient descent algorithm.
With this formulation we get single-parametric filters that allow to tackle the overfitting problem on neighbourhoods where the distribution of node degrees is wide. This property is crucial when building deeper graph neural network with more than few convolutional layers and also in the case of application to large-scale networks. It is also important to notice that, although equations (3.15) and (3.20) are derived from the spectral convolutions, both of them are in the end defined in the spatial domain, thus allowing for further computational efficiency.

The main problem of the spectral approach defined by (3.11) can be find in the definition of convolution dependent on the Fourier basis: this dependence implies that a model learnt on a specif domain is not easily transferable

to a different one [36]. For this reason different approaches have been developed in order to allow the convolution operator to deal with different sized neighborhoods [51].
One of the methods that overcome this limitation is introduced in [19]. The basic idea of this method is to learn a function able to generate node embeddings only by looking at the neighbourhood of the single node. The method is called GraphSAGE, which stands for *SAmple and aggreGatE*, and a schematic representation can be found in figure 3.1.1. The main difference from previous methods is that it doesn't learn one different embedding for each node, but it learns an aggregation operator over a subset of the neighbouring nodes:

$$x_{\mathcal{N}(v)}^{(k+1)} = s(\{x_u^{(k)}, \forall u \in \mathcal{N}(v)\}) \tag{3.22}$$

where s is an aggregation function and $\mathcal{N}(v)$ is a fixed-size subset of the neighbours of v. Since there's no natural ordering between the neighbours of a node, the aggregation function s needs to be a permutation-invariant function, operating over an unordered set of vectors. The method requires also that $\mathcal{N}(v)$ is sampled again at each iteration, with a uniform distribution over $\{u \in V : (u, v) \in E\}$.
The hidden representation from (3.22) is then concatenated with the vector of node features at layer k:

$$x_v^{(k+1)} = \xi(W_k \cdot [x_v^{(k)} || x_{\mathcal{N}(v)}^{(k+1)}]) \tag{3.23}$$

where W_k is a matrix of parameters to be learnt at layer k. By concatenating the representation of nodes at the previous layer, we create a *skip-connection* layer, where the network is allowed to maintain only the previous representation of the node, in the case it results to be the best option for the task.

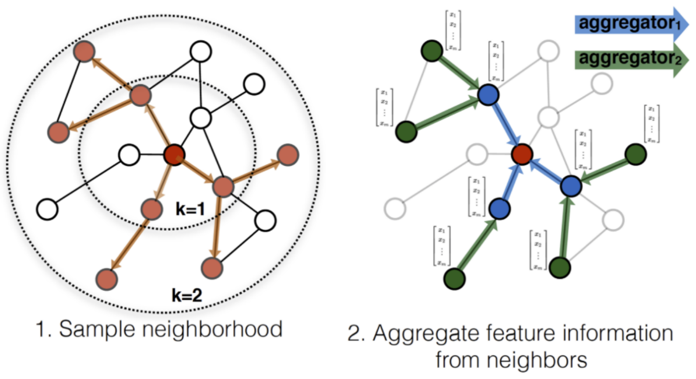


Figure 3.1: A schematic representation of the GraphSAGE method [19]

In [19], different functions s are proposed. The most simple idea is to use the mean over the vectors of the neighbouring set. Alternatively a pooling aggregator can be used, in the form:

$$s(T) = max(\{\sigma(W_{pool}x_u^{(k)} + b, u \in T\}), \quad T \in V$$

where T is the subset of neighbouring nodes.
This method is particularly useful for graphs in which the nodes have initially a big amount of features describing their properties, for example social networks with people's information or citation data with text attributes, but it also allows to exploit the structural information of the graph. Moreover, while previous approaches were useful for the case in which all the graph structure is previously known (e.g. graph-classification or semi-supervised node classification), the convolution layer of GraphSAGE allows to extend the learnt function also in inductive settings, when new nodes are added to the graphs in following stages, as in the case of a new person joining a social network.

3.1.2 Graph Pooling Layers

Extending the idea of pooling layers from CNNs to graph domains is particularly challenging. This is due to two main factors: the lack of a definition of spatial locality and, in the setting of multiple domains, the variable number of nodes in different data samples. In the context of graphs, in fact, there's not a clear definition of what a *patch* is and it is therefore complicated to decide which nodes should be pooled together.
Despite the challenge, a notion of pooling is often necessary when dealing with graph classification. It was therefore initially introduced in the form of a readout layer where *global pooling* is performed: all the nodes embeddings are pooled together and a simple permutation-invariant function is applied to them. This kind of layer is usually inserted after the message-passing and aggregating steps are performed by convolutional layers, and it's usually followed by one or more fully connected layers that allow to perform classification.

The downside of these methods is that they are intrinsically flat and don't allow to capture any hierarchical structure that might be present in the network. This is the reason why the authors of [49] proposed a pooling layer in which a soft cluster assignment is performed and afterwards the node features of each cluster are aggregated together. This structure allows to create a smaller graph structure in the next layer, where the information regarding similar nodes has been aggregated together.
The model is based on learning a cluster assignment matrix through a graph neural network module of convolutional layers. This matrix can be denoted by $S^{(k)} \in \mathbb{R}^{n_k \times n_{n+1}}$, where n_k in the number of nodes at layer k and n_{k+1} is the number of clusters that we want to generate (which coincides with the

number of nodes of the input to the next layer). If we denote as $X^{(k)} \in \mathbb{R}^{n_k \times f_k}$ and $W^{(k)} \in \mathbb{R}^{n_k \times n_k}$ the matrix of node features and the weighted adjacency matrix respectively, at layer k, we can compute the soft assignment as:

$$S^{(k)} = \text{softmax}\left(\text{GNN}(X^{(k)}, W^{(k)})\right) \tag{3.24}$$

where GNN denotes a module of stacked convolutional layers. This clustering module can be trained together with the rest of the neural network in an end-to-end structure. Once the soft assignment is computed, it can be used to aggregate the information of nodes in each cluster following:

$$X^{(k+1)} = S^{(k)^T} X^{(k)} \in \mathbb{R}^{n_{k+1} \times f_k} \tag{3.25}$$

where we can note that the number of node features f_k remains unchanged in the pooling process. Subsequently the graph structure between the new *cluster nodes* can be generated through:

$$W^{(k+1)} = S^{(k)^T} W^{(k)} S^{(k)} \in \mathbb{R}^{n_{k+1} \times n_{k+1}} \tag{3.26}$$

where the elements of $W^{(k+1)}$ denote the strength of the connections between each pair of clusters [49]. This method, called *DiffPool*, introduces a graph coarsening procedure that allows to reduce the size of the graph when going deeper in the neural network and this helps to detect possible hierarchical structures in the graph.

While the previous approach works really well for small graphs, it can encounters problems when processing bigger structures. First of all, because of the soft-clustering procedure, the resulting graph will not be sparse even if the starting one had few connections [50]. Secondly, it requires to store at each step the soft-clustering matrix, which at the beginning of the network can be of the order of $\mathcal{O}(r|V|^2)$, where $r \in (0, 1]$ is the fraction of graph vertices that is kept. For this reason, in [8] and [16], a new methodology is proposed to retain only $\lceil rn \rceil$ nodes, instead of creating $\lceil rn \rceil$ clusters from previous nodes, where n is the number of nodes in input to the layer. This is achieved by simply dropping $n - \lceil rn \rceil$ nodes based on a learnable projection score. If we defined as $p^{(k)} \in \mathbb{R}^{f_k}$ the learnable projection vector and as $y^{(k)} \in \mathbb{R}^n$ the vector of scores, the method can be formalized as follows:

$$\begin{aligned} y^{(k)} &= \frac{X^{(k)} p^{(k)}}{\|p^{(k)}\|} \\ \boldsymbol{i} &= \tau(y^{(k)}, r) \end{aligned} \tag{3.27}$$

where τ is a function that selects only the indices of the top $\lceil rn \rceil$ elements in $y^{(k)}$ and $\boldsymbol{i} \in \mathbb{R}^r$ is a vector denoting the retained indices. The selection step is then performed by:

$$\begin{aligned} X^{(k+1)} &= \left(X^{(k)} \odot ((\tanh(y^{(k)})^T \mathbb{1}_n)\right)_{\boldsymbol{i}} \\ W^{(k+1)} &= (W^{(k)})_{\boldsymbol{i},\boldsymbol{i}} \end{aligned} \tag{3.28}$$

where $\odot$ is the Hadamard product and $(\cdot)_{\boldsymbol{i}}$ is the indexing operation based on vector $\boldsymbol{i}$. The idea behind this method is to relegate the step of the aggregation of information between nodes in the convolutional layers, and just perform a selection of the most important nodes in the pooling layer. This selection, performed just by slicing the feature matrix and the adjacency matrix, allows to maintain sparsity in the graph [8], a characteristic that was missing in *DiffPool*.

3.2 Attention Mechanism in Graph Neural Networks

In real applications, we often have to deal with graph-structured data that can be both really big and noisy. As previously said in 2.2, attention mechanisms allow the network to focus on the specific parts of the input that turn out to be most relevant. This benefit can be leveraged as well with graph structured data, in an attempt to force the model to focus on the most important part of the graphs and thus improve the signal-to-noise ratio [29]. Moreover, if successfully applied, attention provides a tool for interpreting the results given by the network and discovering the underlying dependencies that have been learnt.
Given a set of nodes $\Gamma_V = \{v_0, ..., v_n\} \subset V$, not necessarily composed by all the nodes in V, and a target object s, representing a specific entity in the network, we can generally define attention in graphs as a function $\phi'_s : \Gamma_V \rightarrow [0, 1]$ that maps each node in Γ_V to an attention score and that satisfy the following condition:

$$\sum_{i=0}^{|\Gamma_V|} \phi'_s(v_i) = 1 \tag{3.29}$$

Attention mechanisms in graphs can be generally divided into two classes: attention-based node embeddings and attention-based graph embeddings [29]. In the following paragraphs, we will give a general description of both.

In the case of node embeddings, the target object s is a generic node of a graph v_j and Γ_v is the set of neighbours $\mathcal{N}(v_j)$ of node v_j. The attention mechanism is applied to every node of the graph and it defines the relative importance of every node v_i in $\mathcal{N}(v_j)$ in creating the embedding of node v_j.

The objective is to learn a generic attentive function $\phi : V \to \mathbb{R}^{f_{k+1}}$, that will be applied to every node of the graph at a specific layer k.
The previous mechanism is applied for example in the GAT convolutional layer by [47], where the embedding of a node is computed by attending over the feature vectors of nodes in its 1-hop neighbourhood.
If we define as $\{x_1^{(k)}, \dots, x_n^{(k)}\} \in \mathbb{R}^{f_k}$ the input feature vectors of each node, a linear transformation can be applied to them by a learnable weight matrix $W \in \mathbb{R}^{f_{k+1} \times f_k}$ and subsequently a masked attentional mechanism can be applied to every node, resulting in the following scores:

$$e_{ij} = a(Wx_i^{(k)}, Wx_j^{(k)}) \tag{3.30}$$

with $i \in V$ and $j \in \mathcal{N}(i)$. The attention function a used is an feed-forward neural network parametrized by a vector $\boldsymbol{a} \in \mathbb{R}^{2f_{k+1}}$. The coefficients in (3.30) are then fed to a softmax function, resulting in the following equation for computing the attention scores [47]:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\boldsymbol{a}^T(Wx_i^{(k)}||Wx_j^{(k)})\right)\right)}{\sum_{k \in \mathcal{N}(i)} \exp\left(\text{LeakyReLU}\left(\boldsymbol{a}^T(Wx_i^{(k)}||Wx_k^{(k)})\right)\right)} \tag{3.31}$$

where || denotes the concatenation of vectors.
These attention scores are then used to compute the output of the layer $\{x_1^{(k+1)}, \dots, x_n^{(k+1)}\} \in \mathbb{R}^{f_{k+1}}$ through:

$$x_i^{(k+1)} = \xi\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} W x_j^{(k)}\right) \tag{3.32}$$

where ξ is an appropriate non-linear function. Alternatively, a multi-head formulation can be employed, following:

$$x_i^{(k+1)} = \Big\|_{p=1}^{m} \xi\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^m W^m x_j^{(k)}\right) \tag{3.33}$$

with m representing the number of *attention heads* and $x_i^{(k+1)} \in \mathbb{R}^{mf_{k+1}}$.
The model described in (3.33) helps to bring more stable results, besides being parallelizable and computational efficient. Moreover, differently from many layers described in 3.1.1, it does not depend on knowing the graph structure upfront, while at the same time taking into consideration all the neighbours of a certain node when computing its embedding.

For what concern instead graph embeddings, the goal is to learn a function $\phi : G \to \mathbb{R}^m$ that maps the whole graph to a low-dimensional vector. In the context of attention mechanisms this is done by learning an attention function that gives different important scores to different subregions of the graph. The embedding can then be fed into a standard neural network to perform what is called *attention-based graph classification* [29].
One of the first works that proposed to use attention in the context of graph classification is GAM [28], which is based on attention-guided walks over a selected number of informative nodes. In particular, GAM uses an attention mechanism over the neighbourhood of a node in order to decide which steps to take at each next iteration. This mechanism is trained through a reinforcement learning approach and constitutes a model that is able to learn which are the important parts of the graph. Moreover, in the case of very large graphs, multiple walks can be initialized and performed in parallel.
An attention mechanism has also been introduced in the context of a read-out layer in [33]. In fact, given the node feature vectors after the last convolutional layer $\{x_1^{(K)}, \ldots, x_n^{(K)}\}$, an embedding of the entire graph G can be computed following the attention model:

$$h_G = \xi \left(\sum_{i \in V} \alpha_i^{(K)} \odot \tanh \left(g(x_i^{(K)}) \right) \right) \tag{3.34}$$

where g is a feed-forward neural network, ξ is a non-linear function and $\{\alpha_1^{(K)}, \dots, \alpha_n^{(K)}\}$ represents the attention scores at layer K. Each score is given by:

$$\alpha_i^{(K)} = \sigma\left(a(x_i^{(K)})\right) \tag{3.35}$$

where σ is the SoftMax function and a is a feed-forward neural network.

It has to be noticed that some instances of pooling in graphs can be redefined through an attentive framework [26]. After computing the attention coefficients for nodes in a layer k, these can be used for selecting the most important nodes through a thresholding operation of the form:

$$x_i^{(k+1)} = \begin{cases} \alpha_i^{(k)} x_i^{(k)} & \text{if } \alpha_i^{(k)} > \bar{\alpha} \\ 0 & \text{otherwise} \end{cases} \tag{3.36}$$

where α_i is the attention score of node i at layer k.
In this context, [30] proposes to take into account the graph topological structure also when computing attention scores $\alpha \in \mathbb{R}^n$. This can be done by integrating a convolutional structure, such as (3.20), in the formula, bringing to:

$$\alpha^{(k)} = \sigma(\tilde{D}^{-1/2}\tilde{W}\tilde{D}^{-1/2}X^{(k)}p) \tag{3.37}$$

where $\tilde{D}$ and $\tilde{W}$ are defined as in (3.20), X is the feature matrix at layer k and $p \in \mathbb{R}^{f_k}$ is the only vector of learnable parameter in the layer. The attention scores learnt can then be used to modulate the information coming to the next layer or can be used in (3.36) to create a pooling layer. Alternatively, other convolutional operations can also be used in (3.37).

3.3 Self-Attentive Graph Classifier

The proposed method aims to extend the ideas introduced in [22] to the case of graph inputs. The authors of [22] presented an attentive framework in the context of convolutional neural networks for image classification. The core idea consisted in re-purposing the global representation of the image as a query that could attend over the hidden representations generated at intermediate layers of the network. Since every image classification task can be seen as a specific case of a graph classification problem, the main idea introduced in this work is to extend their concept to the case of graph classification. Therefore, an attentive framework that can be applied to different graph neural network architectures is here proposed.
In the setting of graph classification, the input to the problem is given by a set of labeled graphs $\mathcal{S} = \{(G_i, y_i)\}_{i \in I}$, where each $G_i \in \mathcal{G}$ represents a graph and the corresponding $y_i \in \mathcal{Y}$ is the label representing the class to which the graph belongs. The cardinality of $\mathcal{Y}$ represents the number of different classes in the dataset, while $\mathcal{G}$ is the set of input graphs. The main objective of the learning process is to find an approximation $\hat{f}$ of the theoretical function $f : \mathcal{G} \to \mathcal{Y}$ that maps each graph to its corresponding label.
As previously said, given a generic graph G in input, it will be characterized by its structure (V, E, W) and by a set of node feature vectors, that are summarized in the input feature matrix $X^{(0)} \in \mathbb{R}^{n_0 \times f_0}$, where n_0 is the number of nodes and f_0 the number of features. The more general setting, which includes also vectors of features for every edge, is here excluded.
The starting feature matrix, together with the graph structure, is fed into the neural network, which will process them through a sequence of convolutional and pooling layers. After each layer k, a new feature matrix $X^{(k)} \in \mathbb{R}^{n}$ is

produced, and, while in the case of convolutional layers the graph structure remains the same, after pooling layers also the triplet $(V^{(k)}, E^{(k)}, W^{(k)})$ is recomputed.

At each layer, the rows of $X^{(k)}$ represent the feature vectors of the nodes present in that layer: $\{x_1, \ldots, x_{n_k}\} \in \mathbb{R}^{f_k}$. Before the first layer, these vectors will represent only the information pertaining the related node, but as the layers go on, the node features will represent the aggregated information of a portion of neighbouring nodes in the graph.

When arriving at the last layer, in order to perform the final graph classification, it is necessary to produce a global vector $g \in \mathbb{R}^m$, in which to aggregate the processed information of the whole graph. This vector can be obtain through different kind of *read-out* layers. Commonly used strategies include taking the max or mean values over the features vectors of all the nodes, or otherwise just summing them up. This brings to a global feature vector with number of features m equal to the number f_k of node features in the last convolutional layer of the network. Alternatively, a read-out layer like the one in (3.34) can be exploited.

The proposed method will use the global representation of the input, represented by vector g, as a *query* in the attention mechanism. Keys and values will instead be constituted by the hidden representation of nodes in the intermediate layers (or by linear transformation of them). The basic idea is to use this mechanism at different intermediate layers along the architecture, in order to be able to get information from different scales.

Given a chosen layer l, the global vector g will be used to attend over the set of features vector $\{x_1, \ldots, x_{n_l}\} \in \mathbb{R}^{f_l}$ learnt at that layer. This process will produce attention scores $\{\alpha_i^l\}_{i=1,\ldots,n_l}$ through the application of a general attention function. The scores are computed through the classical attentive framework:

$$\alpha_i^l = \frac{e_i^l}{\sum_{i=1}^{n_l} e_i^l} \quad , \quad e_i = a(g, x_i^l) \tag{3.38}$$

where a is a generic function computing the alignment model. This function will in fact return a value representing how well the feature vector of node i at layer l relates with the global representation of the graph, thus creating a *multi-scale attention mechanism*.

The function a can be a single-layer neural network taking as input a concatenation of the two vectors:

$$a(g, x_i^l) = p^T(g||x_i^l) \tag{3.39}$$

where $p \in \mathbb{R}^{(m+f_l)}$ is the vector of parameters. Alternatively, the additive mechanism used in [3] can be exploited:

$$a(g, x_i^l) = p^T(g + Wx_i^l) \tag{3.40}$$

where, in the case $m \neq f_l$, it's first necessary to transform the input feature vectors through a shared projection into a space of dimension m, with $W \in \mathbb{R}^{m \times f_l}$. The same has to be done also if a multiplicative attention mechanism is used, following:

$$a(g, x_i^l) = g^T(Wx_i^l) \tag{3.41}$$

in a similarity-based computation that focuses on the alignment between vectors. All the three attention strategies can be used interchangeably inside the proposed framework.

Once the attention vectors are computed, they are employed in order to compute an attentive representation of the selected layer l, given by:

$$g_a^l = \sum_{i=1}^{n_l} \alpha_i^l x_i^l \tag{3.42}$$

By applying this mechanism at different layers $l \in L$, with $L \subset \{1, ..., K\}$, we obtain attentive representations $\{g_a^l\}_{l \in L}$ at different resolutions. These vectors can be concatenated together, potentially also with the global representation g, to constitute the final vector which will be fed to a feed-forward neural network for classification. As an alternative, if all the attentive vectors have the same size, they can be summed up together.
With the described framework, a multi-scale attention mechanism has been created in the context of graph classification. It provides a way to re-purpose a representation of the graph itself as an attentional query, and allows for a cyclic integration between local and global features of the graph structure during the learning process.

Chapter 4

Model application

4.1 Parkinson's disease

Parkinson's disease (PD) is one of the most common neurodegenerative disorders among old people. It mostly affects dopamine-producing neurons situated in a basal ganglia structure of the midbrain, called *substantia nigra*. The loss of neurons that produce dopamine causes the degradation of this area and the dysfunction of the cortico-basal ganglia-thalamocortical circuits, which are deeply connected to movement [15].

For this reason, the main signal of the disease is to be found in movement-related symptoms, which develop gradually over the years and may cause tremors, that are usually worse on one side of the body even when both sides are damaged. Other symptoms include stiffened muscles and slowed movements, along with loss of automatic ones, like blinking and smiling, and changes in speech and writing abilities [27].

The causes of PD are mostly unknown, but it is believed that specific genetic mutations and environmental factors, such as nutrition or exposure to

certain pesticides, may play an important role. It is also known that the risk factor is higher in men than in women and that the disease usually develops in elderly people [34].
Parkinson's disease can't be cured, but the symptoms can be controlled and contained for a fairly long period. Medications that increase the level of dopamine can be useful for improving movements and diminishing tremors, while a healthy lifestyle can help to reduce a lot of problems connected to PD, such as constipation, lack of flexibility and high levels of anxiety [27].

4.1.1 DNA Methylation

Over the past years, many efforts have been made to find genetic signatures related to Parkinson's disease and acquired mutations in some specific genes seem to be involved in the onset of sporadic PD. However, it is believed that also epigenetic factors play a role in the development of the disease. Epigenetic mechanisms involve chemical modification of the structure around the DNA strands, without actually modifying the genomic sequence [34].
One of the most studied epigenetic modification is DNA methylation, a process involved in both gene expression and cell differentiation. The methylation of DNA is characterized by the transferring of a methyl group to the fifth carbon of a cytosine residue. It has been observed that this process occurs almost exclusively on cytosines that are followed by a guanine base on the DNA sequence, i.e at the so called CpG sites [37]. A representation of the process is summarized in figure 4.1.2.
Since it has been demonstrated that epigenetic regulation of neural cells is fundamental for neurogenesis and brain development, it is intuitive that an alteration in these mechanisms can be involved in the onset of neurodegenerative diseases [34]. However, it has been proven that also DNA methylation

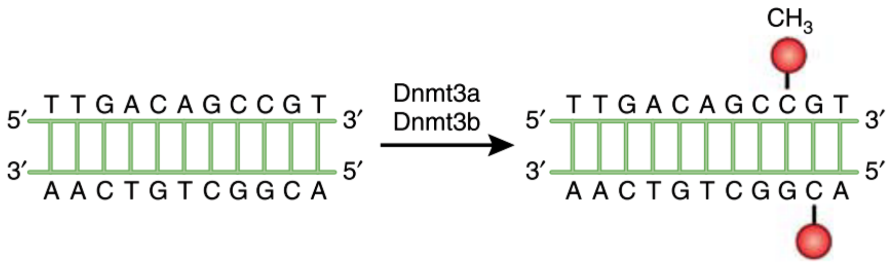

5′ T T G A C A G C C G T 3′
3′ A A C T G T C G G C A 5′
Dnmt3a
Dnmt3b
CH$_3$
5′ T T G A C A G C C G T 3′
3′ A A C T G T C G G C A 5′

Figure 4.1: A schematic representation of DNA methylation catalyzed by DNA methyltransferases (Dnmts) [37]

levels in blood correlate with the presence of Parkinson's disease [10].
There are different ways through which methylation can affect the behaviour of a cell, one of which consists in interfering with gene expression. If in fact the methylated sequence is located in the gene promoter, the methyl group interferes with the binding of the transcription factor and thus inhibits the activation of the related gene [34].
Recent studies have analyzed how DNA methylation relates to aging processes and they brought to the definition of an *epigenetic clock* model. This model allows to estimate the age of different tissues just by looking at the methylation state of a restricted set of CpG sites [21] . These results further support the idea that there has to be a connection between epigenetic biomarkers and neurodegenerative diseases.

4.1.2 SPECT Images

Brain imaging is one of the most commonly used techniques for the diagnosis of Parkinson's disease. In particular, dopamine transporter single-photon emission computed tomography (DaT SPECT) is one of the most effective

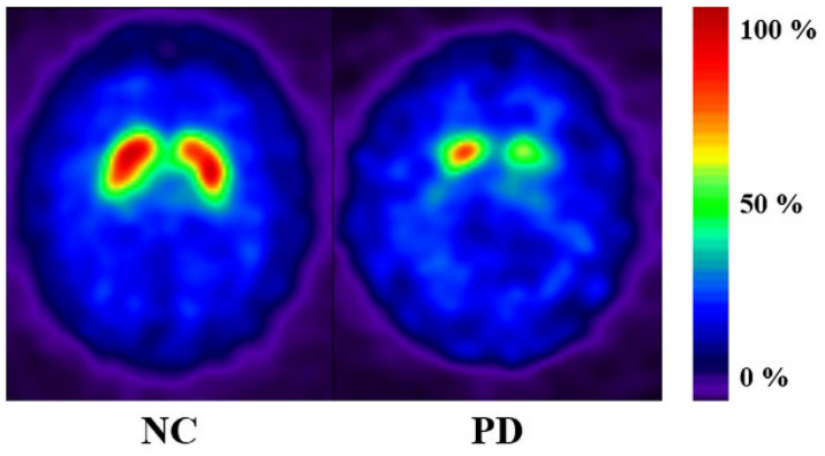


Figure 4.2: Representative slices from the volumetric SPECT images of a healthy patient and one affected by PD, where the color scale represents the amount of dopaminergic activity [42]

methods in detecting deficits inside the nigrostriatal dopamine system [43]. It is in fact proved that, when the degenerative process is in the most advance stage, around 50-60% of dopaminergic neurons in the *substantia nigra* are lost and the content of dopamine is considerably decreased. This happens in particular in the striatum, a cluster of neurons in the subcortical basal ganglia of the forebrain [2].

Given the underlying biological process, SPECT imaging allows the retrieval of 3D information related to the binding of the dopamine transporters (DaTs) with ^{123}I-Ioflupane, which is administered to the patient with an intravenous injection. In particular, it provides a quantitative information about the spatial distribution of dopamine transporters, which is really useful in order to make the correct diagnosis [42].

4.2 Graph neural networks for Parkinson's disease classification

A wide variety of machine learning techniques have been employed in the context of Parkinson's disease classification, with a recent focus in exploring artificial neural networks. These have been in fact employed using different kind of data, ranging from genomic data [48] to acoustic measures [5] and handwritten dynamics [39].

For what concerns the application of deep learning to methylation data, few approaches have been considered until now. An interesting approach consists in a modular framework that uses variational autoencoders for constructing embeddings of the high dimensional data and then performs downstream classification [31]. Another work proposed instead the use of convolutional neural networks on a matrix of one-hot encoding of DNA fragments centered at methylation sites [45]. However, the majority of the studies that applied deep learning to methylation data have been performed in the context of cancer-related data.

Different trials have been made concerning the application of traditional deep learning techniques to SPECT images coming from Parkinson's disease datasets. Convolutional neural networks can be applied directly to the full 3D image, as explored in [9], or to specific smaller voxels selected a priori from the whole volumetric image [24]. Moreover, in [44], a first attempt to integrate methylation and SPECT data has been proposed in the context of Parkinson's disease data.

More recently the field of deep learning has been extended with the introduction of neural networks applied to graph-structured data. To the author's best knowledge, these models haven't been applied either to DNA methylation data or SPECT images yet.

Traditional deep learning frameworks, such as convolutional neural networks, rely on the properties that nearby pixels in an image are correlated between each other and that images have an intrinsic hierarchical structure. Thanks to their architecture, described in section 2.1, these networks are able to take advantage of the spatial information given by the image structure. When dealing with biological data, ranging from gene expression to DNA methylation, the features are often treated as unrelated to each other instead. However, it is known that, in intra-cellular processes, genes and other molecules are highly related to each other, deeply influencing the cell's functions.
The process of gene expression constitutes a good example of this interconnectivity inside the cell. Its core process involves two stages: initially messenger RNA molecules are produced through the process of *transcription* of genes; in a second moment the mRNA undergoes a *translation* process that finally synthesize proteins. The starting gene is said to *code* for the resulting protein and it's known that there are some genes coding for multiple proteins. Some of the resulting proteins play an important role in the process of transcription of other genes, thus generating a network of transcriptional dynamics between genes, usually defined as *gene regulatory network*. Moreover, the proteins produced by the whole gene expression process will interact between each other inside the cell in complex biological pathways, that can be described by a protein-protein interaction (PPI) network .
For this reason, new works are emerging that aims to integrate this biological networks in the structure of deep learning models, as done for instance in [40]. The idea is that the information about gene interactions can be used to impose a biological bias on the model, in the same way as convolu-

tional network impose a spatial bias on image data [12]. Since the process of DNA methylation seems to be deeply related with gene expression, and thus with protein production, in this work we will try to construct a graph neural network that process methylation data after having arranged them on a protein-protein interaction network. This allows to introduce biological knowledge in the model by allowing the model to know which features in the input are biologically connected to each other [4]. The introduction of prior validated knowledge can be particularly useful in the case of high dimensional data with restricted number of samples, which is the case of most dataset containing DNA methylation data.

When considering instead 2-dimensional or 3-dimensional images, convolutional neural networks have proved to be the state-of-the-art method and it's difficult that a graph neural network can achieve better results when applied on the same input. What a standard neural network cannot do, instead, is to work directly with higher-order representation of an image, such as the set of superpixels of which it is composed. The main advantage that can derived from working on superpixels is a reduction in the size of the model, which can be especially important when the size of input images is big [25]. Moreover, in the context of brain imaging, not only the volumetric images are big, but it is also common to have a really small amount of samples for the training of the model.
In this situation, to avoid training a big model, various strategies can be followed. The extraction of patches that might relate to the problem to be solved is one of those, but often requires some biological knowledge beforehand. Alternatively, the dimensionality of the input can be reduced by down-sampling, but this strategy often doesn't bring great results in the medical

context. By extracting superpixels instead, we are able to maintain information on the geometrical structures inside the images [25].

4.3 Dataset overview

The data used in this work were obtained from the Parkinson's Progression Markers Initiative (PPMI) database (*www.ppmi-info.org/data*). The PPMI database contains information from different data modalities, among which clinical data, imaging ones and various types of genetic information.
In the next sections, the purpose will be to apply the described models to SPECT images and DNA methylation data present in the dataset. There are 450 patients in the PPMI cohort for which both methylation samples and SPECT data have been registered. Of the total amount of patients, 317 have been diagnosed with PD, while 133 are healthy patients.

The methylation profiling was performed using Illumina Human Methylation EPIC array on whole-blood extracted DNA samples. Each sample was analyzed at a single-nucleotide resolution and the analysis returned the methylation profiling of 864067 CpG sites across the genome. The methylation dataset is therefore composed by a matrix of dimensions 450×864067, where each element represents the methylation state of a particular CpG site in a specific sample. In particular, for each site, the reported value is a β-value that estimates the methylation level with:

$$\beta = \frac{M}{M+U}$$

where M is the number of cells in the sample where the analyzed CpG site is methylated and U is the number of cells for which the DNA molecule was not

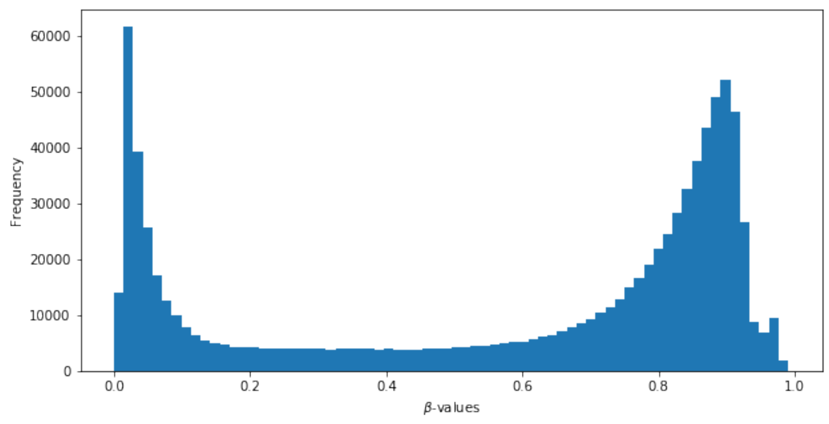


Figure 4.3: Histogram representing frequencies of β-values in a representative sample

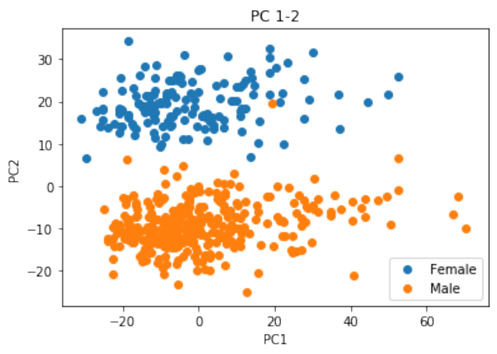
methylated in that position. The β-values are therefore intensity estimates, with $\beta \in [0, 1]$. As it can be noticed from figure 4.3, a big portion of probes tends to assume values at the extremes of the interval.

The SPECT dataset consists instead of 3D grayscale images that have been aligned together and saved in the NifTI-1 data format. Each image is represented by a $91 \times 109 \times 91$ tensor, with values normalized to the range $[0, 1]$.

4.4 Data preprocessing

4.4.1 DNA methylation data

As a first explorative step, an unsupervised analysis over the methylation matrix has been performed through principal component analysis. The projections of the data on the two principal components is shown in figure 4.4.1.



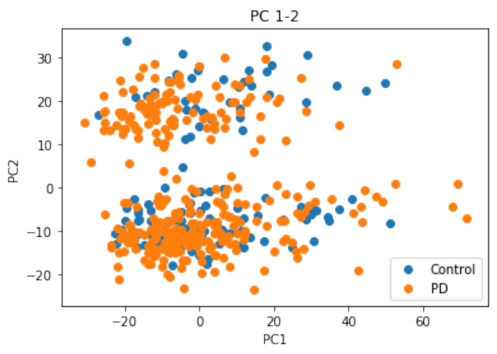


Figure 4.4: Principal component analysis of methylation data

From those projections, it's possible to notice how the data seems to be divided in two clusters, related to the biological sex of the patients. Healthy and control patients seem to be divided equally in the two clusters. For this reason, all the probes related to DNA positions situated on sex chromosomes have been excluded from the analysis.

Moreover, [52] identified a number of probes that, because of the specific region of DNA that they refer to, should be excluded from the analyses for probe selection. We then further pre-filter those probes from the data matrix.

A further selection step is performed considering the final goal of embedding the features in a network of genes: only the CpGs located in a DNA position corresponding to a gene are kept in the analysis.

Subsequently, the 10000 most important probes for the goal of Parkinson's disease classification in the dataset have been selected on the base of univariate ANOVA statistical tests.

As previously said, the goal is to embed the selected features on a biologically meaningful graph. There are many different biological graphs that can be considered, which differ for the quantity of genes considered and the amount

and type of relationship between them. For the purpose of this work we decided to use a the protein-protein interaction graph. In particular, we refer to the interaction data registered in the STRING database [13], which contains information of gene associations recovered from multiple sources, like biological experiments, literature information and computational prediction. We decide to focus in particular on the association that have been biologically proved, in order to avoid to add more uncertainty in the problem.
The resulting graph dataset will be composed by 450 graphs, each with 10000 nodes: each node will represent a location in the DNA (CpG). The edges between nodes will instead be constructed based on the biological interactions between the proteins encoded by the genes present in the CpG location.

4.4.2 SPECT imaging data

For what concerns the volumetric SPECT data, they are first pre-processed using a specific algorithm for superpixels generation. Different algorithms have been proposed for this purpose, but one of the fastest and most effective is the *simple linear iterative clustering* (SLIC) algorithm. SLIC is based on a procedure that generates superpixels by clustering pixels based on their color similarity and proximity in the image [1]. An example of results given by SLIC superpixel is given in figure 4.4.2, where it is applied to a slice of a SPECT image. The SLIC algorithm is performed using the `skimage.segmentation.slic` function on the 3D images, attempting to create around 1000 superpixels for each image.
We then want to construct a graph based on this division in superpixels:

- each superpixel will be associated with a node in the graph;
- the feature vector of each node will be constituted by four features: the

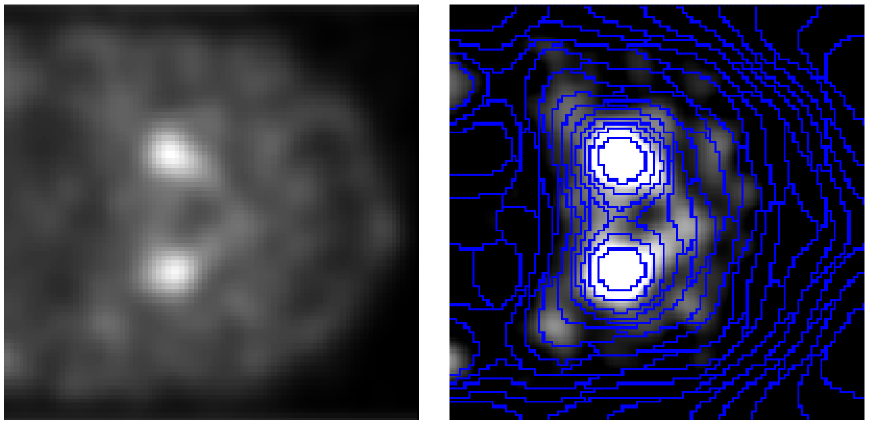


Figure 4.5: Slice of a SPECT image on the left; resulting boundaries given by SLIC algorithm on the right

first one will be the mean between the intensity values of the elements in the superpixel, the other three are the coordinates of the center of mass $z \in \mathbb{R}^3$ of the superpixel itself;

- edges between nodes are formed based on the euclidean distance between centers of mass, based on the following formula [11]:

$$W_{ij} = \exp\left(-\frac{\|z_i - z_j\|^2}{\sigma^2}\right)$$

  with $\sigma = 0.1\pi$. Only for the $k = 20$ nearest nodes the edge will actually be maintained in the final graph.

Starting from an initial input size of $91 \times 109 \times 91 = 902629$, the dimensionality of the input will then be reduced to a set of around 4×1000 node features, while the average amount of edges in the newly created graphs will then be of 20000 edges.

4.5 Experiments and results

The code has been implemented in Python, using the libraries PyTorch [38] and PyTorch Geometric [14] for the neural networks section.
The model constructed for classifying the methylation graphs consists of 3 convolutional layers, with 2 final fully connected layers. The size of the hidden node representations has been set to 4 after some experiments. Skip connections after each convolutional layer are also included to allow the model to be more flexible. The big size of the underlying graph constitutes a limit in the construction of a deeper and more complex architecture.
The network has been trained with the so called *RMSProp* (Root Mean Square Propagation) algorithm, using a standard cross-entropy loss function. The training is performed for 200 epochs, with an early stopping option if the accuracy doesn't improve for a certain amount of epochs. The hyperparameters used during training are summarized in table 4.1.

Training hyperparameters (methylation)	
Learning rate	10^{-3}
Batch size	16
Number of epochs	200
Early stopping (epochs)	50
L^2 weight decay	10^{-1}

Table 4.1: Values of hyperparameters used during training with RMSProp (methylation data classifier)

The experiments have been performed with a stratified 3-fold cross-validation, across different types of convolutional layers:

- *ChebNet* refers to the convolutional operation described in formula (3.16), with $r = 3$;
- *GCN* refers to the convolution described in (3.20);
- *GraphSAGE* refers to the opearation in (3.23);
- *GAT* is instead the attentional convolution in (3.32), with 3 heads.

The results obtained with different layer types are reported in figure 4.6. The results are given in terms of Area Under the ROC (Receiver Operating Characteristic) curve. The plot of a ROC curve summarizes the performance of a binary classification model on the positive class, plotting the True Positive Rate against the False Positive Rate, when varying the threshold parameter of the classifier. The Area Under the ROC Curve (AUC) is a good metric for evaluating the expressive power of a classifier and it's superior to the accuracy in case of imbalanced datasets with binary labels.

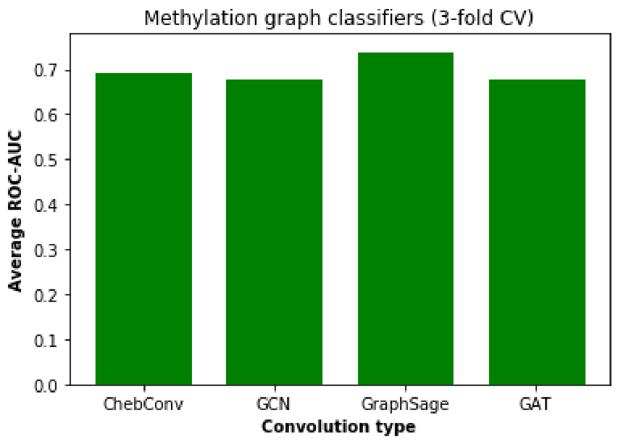


Figure 4.6: Average value of AUC obtained using different types of convolutional layers in the model

We can observe that none of the classifiers provides a high average ROC-AUC value. However, it seems that the one composed with the *GraphSAGE* convolutional layers performs a bit better than the rest, providing an average AUC of 0.74 and an average classification accuracy of 75.8%.
The attentive model proposed in 3.3 has been tested on the methylation graphs, however it does not seem able to capture the information from the methylation signal.
The reasons for the low performances observed in figure 4.6 can be various. First of all, the amount of data available, together with the high dimensionality of the dataset itself and the amount of noise that it contains, makes it difficult to train any model in a 3-fold cross-validation fashion. Only $66,6\%$ of the data is in fact used during training, while the remaining ones equally split between validation and test sets.
In second instance, we made a strong assumption by choosing a specific biological structure for the underlying graph. Different experiment should be performed to analyse if, using different biological graphs, the results might improve. Moreover, the big amount of data made the training of a deeper convolutional network difficult, while at the same time the feature selection step was also really affected by the few samples available for such noisy data.

Differently from the classifier used for the methylation data, the models constructed for classifying the SPECT graphs exploit the framework introduced in section 3.3. They consists of 6 convolutional layers and also in this case three different types of convolution (from *ChebConv*, *GraphSAGE* and *GAT*) are analyzed. The latent feature vectors are this time composed by 10 values, which is also the same amount of variables that will constitute the graph representation g, produced after the fully connected layers attached at the end of the convolutional block. The attentive mechanism described in equation (3.38) is applied every two layers and the attention function used is (3.39). This time the model doesn't include a skip-connection mechanism, as the attentive connections already serve as ways for the information contained in the first layers to be considered directly when building the final vector of features. The optimization algorithm used during training is *Adam*, an adaptation of the classical stochastic gradient descent procedure, and the hyperparameters used are reported in table 4.2.

Training hyperparameters (SPECT)	
Learning rate	$5 \cdot 10^{-3}$
Batch size	16
Number of epochs	100
Early stopping (epochs)	20

Table 4.2: Values of hyperparameters used during training with Adam (SPECT data classifier)

Different experiments have been carried out in the context of the final integration of the attentive vectors:

- *case* 1: the attentive vectors $\{g_a^l\}_{l \in L}$ are averaged together and the resulting vector is used for the final classification;
- *case* 2: the vectors $\{g_a^l\}_{l \in L}$ are concatenated together before performing the final classification;
- *case* 3: the global vector g and the vectors $\{g_a^l\}_{l \in L}$ are averaged together and then used for classification.

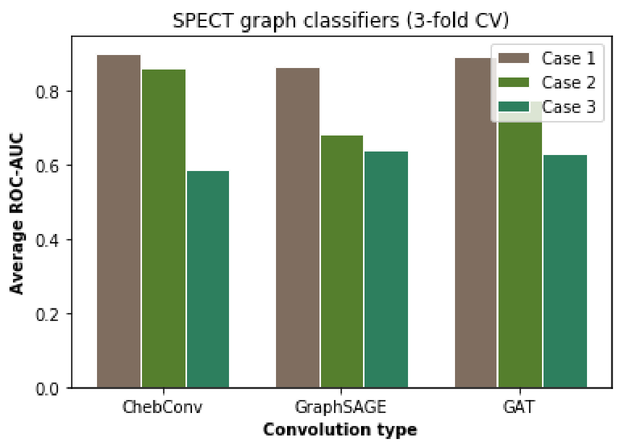


Figure 4.7: Average value of AUC obtained using different kind of convolutional layers in the model, with different integration of attentive vectors

The results of the experiments, analyzed with different convolutional layers, are shown in figure 4.7. They suggest that considering the global vector g together with $\{g_a^l\}_{l \in L}$ is probably bringing more noise into the model, resulting in the worst performances consistently across layer type. The model that instead seems to perform better is the one that compute the average between the features of the attentive vectors at different scales.

Even if more experiments should be performed for what regards a more systematic hyperparameters selection, adapting it to each model separately, we decided to focus on the model that performed better on the previous analysis: the 6-layer model with convolutional layer composed by *ChebConv*. A comparison is carried out to verify what happens if the attentive mechanism is applied after each of the first three layers (early attention) or after each of the last three layers (late attention).

Attention model	**Accuracy (%)**	**ROC-AUC**
Early attention	80.81 ± 2.03	0.887 ± 0.014
Late attention	79.86 ± 1.05	0.844 ± 0.018

Table 4.3: Results from SPECT graph classifier

Interestingly, the results are different from what expected: the attention model seems to work slightly better, expecially in terms of AUC, when applied after the first layers of the network. In the case of convolutional networks applied on euclidean images, [22] noticed how attention models attached to deeper layers brought to better results. This was easily explained by the fact that in images from the natural world, as the one analyzed in the paper (CIFAR-100), the last layers are the ones that convey the most information about objects present in the image. However, in our case the setting is quite different, all the images represent the same object (a patient's brain) and it's more probable that important details for classification can be found at smaller scales. Moreover, the division in superpixels already caused a coarsening of the input data, so that each superpixel and its immediate neighbours already contain aggregated information ready to be used by the network.
More experiments concerning analysis of network depth and variations of attentional functions should be considered in order to further optimize the models.

Chapter 5

Conclusions and future works

In this work an extensive introduction to graph neural networks has been provided, with a description of the main methods developed in this emerging field. After describing how the convolutional operations, typical of CNNs, can be extended to graphs, a particular focus has been given to attentive mechanisms. In the context of graph classification, a new method for computing multi-scale self-attentive features has been introduced. Attention mechanisms have previously been proposed in various structures inside graph neural networks, however no mechanism allowed to integrate attention in a multi-scale fashion, taking into account representations at different scales simultaneously and allowing them to attend one over the other.
The presented methods have been applied to graphs constructed starting from biological data and an analysis of how different models perform on different data has been carried out. The research of new models able to integrate biological knowledge inside the standard machine learning pipelines is a field that is just starting to be explored and may bring many important advantages in the research field. From one point of view, the integration of

biological biases may help to increase both accuracy and interpretability of the models. On the other hand, insights given by interpretable models may serve as an input for generating new biological hypothesis and thus opening new research directions. A sector in which graph neural networks may be useful in the future is represented by the research of important biological pathways correlated with specific diseases and, in this context, attentive mechanisms on graphs could play an important role.

However, when adding biological networks in the analysis, it's important to verify the intrinsic value of the underlying graphs that are selected to be used as bias. An idea for a future work stands in the analysis of how different prior graphs can affect the final results and which ones bring the most valuable knowledge for the task at hand.

For what concerns DNA methylation data, it could be interesting to analyze if different gene interaction graphs, such as gene regulatory networks or co-expression ones, could add more value to the analysis. Moreover, it could be interesting to approach the problem with a multiplex framework, where nodes representing genes are connected by multiple edge-types in a unified multi-layer structure.

In the context of volumetric images, it's important to notice that analysis at different scales may bring different results. Low-levels of abstraction tend to maintain information about the detail of the images, while using bigger superpixels only the global information is preserved. For this reason it can be interesting to integrate different levels of abstraction in the same graph neural network, as proposed by [25], and analyze how this feature would improve the results in the context of biomedical imaging.

It's also important to notice that, in the context of Parkinson's disease, the amount of available multimodal datasets is slowly increasing. Among these data, functional magnetic resonance (fMRI) are a kind of data that is naturally predisposed to be represented as graphs. After the necessary preprocessing steps, specific region of interest in the brain can be encoded as nodes and then, by computing the correlation of signals between these regions, a correlation matrix can be constructed. The final edges will be derived from the correlation matrix, by applying a task specific thresholding operation. The final connectivity graphs, describing functional relationship between regions of the brain, represent the perfect input for graph neural networks [32]. In this setting, an attentive model could potentially bring many insights on the parts of the brain that relates with specific functional problems, like the ones connected to Parkinson's disease.

In the context of data integration in the biomedical field, at least at the current stage, one of the main limitation is given by the fact that not all the data types are available for every patients, so classical data-integration techniques may fail to work cause of the small amount of data in the intersection of datasets. It could therefore be interesting to explore how data encoders, trained separately on different data modalities, would work in a downstream integration task, when presented with new data that haven't been seen before.

Finally, integration with other datasets and exploration of transfer learning techniques might be one of the most promising fields to explore, with the goal of expanding the available knowledge in the context of Parkinson's disease.

Bibliography

[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Susstrunk. Slic superpixels, June 2010.

[2] Moran Artzi, Einat Even-Sapir, Hedva Shacham, Avner Thaler, Avi Urterger, Susan Bressman, Karen Marder, Talma Hendler, Nir Giladi, Dafna Ben Bashat, and Anat Mirelman. Dat-spect assessment depicts dopamine depletion among asymptomatic g2019s lrrk2 mutation carriers. *PLOS ONE*, 12, 04 2017.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[4] Paul Bertin, Mohammad Hashir, Martin Weiß, Geneviève Boucher, Vincent Frappier, and Joseph Paul Cohen. Analysis of gene interaction graphs for biasing machine learning models. *ArXiv*, abs/1905.02295, 2019.

[5] Lucijano Berus, Simon Klancnik, Miran Brezocnik, and Mirko Ficko. Classifying parkinson's disease based on acoustic measures using artificial neural networks. *Sensors*, 19:16, 12 2018.

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

[7] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *CoRR*, abs/1312.6203, 2013.

[8] Catalina Cangea, Petar Velickovic, Nikola Jovanovic, Thomas Kipf, and Pietro Liò. Towards sparse hierarchical graph classifiers. *ArXiv*, abs/1811.01287, 2018.

[9] Hongyoon Choi, Seunggyun Ha, Hyung-Jun Im, Sun Paek, and Dong Lee. Refining diagnosis of parkinson's disease with deep learning-based interpretation of dopamine transporter imaging. *NeuroImage: Clinical*, 16, 09 2017.

[10] Yu-Hsuan Chuang, Kimberly Paul, Jeff Bronstein, Yvette Bordelon, Steve Horvath, and Beate Ritz. Parkinson's disease is associated with dna methylation levels in human blood and saliva. *Genome Medicine*, 9, 12 2017.

[11] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 2016.

[12] Francis Dutil, Joseph Paul Cohen, Martin Weiss, Georgy Derevyanko, and Yoshua Bengio. Towards gene expression convolutions using gene interaction graphs. *ArXiv*, abs/1806.06975, 2018.

[13] Damian Szklarczyk et al. String v11: protein-protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic Acids Res.*, 47(D1):D607–D613, 2019.

[14] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *ArXiv*, abs/1903.02428, 2019.

[15] Adriana Galvan, Annaelle Devergnas, and Thomas Wichmann. Alterations in neuronal activity in basal ganglia-thalamocortical circuits in the parkinsonian state. *Frontiers in Neuroanatomy*, 9:5, 2015.

[16] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *ICML*, 2019.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[18] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734 vol. 2, July 2005.

[19] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

[20] David K. Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *ArXiv*, abs/0912.3848, 2008.

[21] Steve Horvath and Ken Raj. Dna methylation-based biomarkers and the epigenetic clock theory of ageing. *Nature Reviews Genetics*, 19, 04 2018.

[22] Saumya Jetley, Nicholas A. Lord, Namhoon Lee, and Philip H. S. Torr. Learn to pay attention. *ArXiv*, abs/1804.02391, 2018.

[23] Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ArXiv*, abs/1609.02907, 2016.

[24] Ivan S. Klyuzhin, Nikolay Shenkov, Arman Rahmim, and Vesna Sossi. Use of deep convolutional neural networks to predict parkinson's disease progression from datscan spect images. 2018.

[25] Boris Knyazev, Xiao Lin, Mohammed Abdel Rahman Amer, and Graham W. Taylor. Image classification with hierarchical multigraph networks. *ArXiv*, abs/1907.09000, 2019.

[26] Boris Knyazev, Graham W. Taylor, and Mohammed Abdel Rahman Amer. Understanding attention and generalization in graph neural networks. In *NeurIPS*, 2019.

[27] Antonina Kouli, Kelli M. Torsney, and Wei-Li Kuan. Parkinson's disease: Etiology, neuropathology, and pathogenesis. In *Parkinson's Disease: Pathogenesis and Clinical Aspects*. 2018.

[28] John Boaz Lee, Ryan Rossi, and Xiangnan Kong. Graph classification using structural attention. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '18, page 1666–1674, New York, NY, USA, 2018. Association for Computing Machinery.

[29] John Boaz Lee, Ryan A. Rossi, Sungchul Kim, Nesreen Ahmed, and Eunyee Koh. Attention models in graphs: A survey. *ArXiv*, abs/1807.07984, 2018.

[30] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *ICML*, 2019.

[31] Joshua J. Levy, Alexander J. Titus, Curtis L. Petersen, Youdinghuan Chen, Lucas A. Salas, and Brock C. Christensen. Methylnet: An automated and modular deep learning approach for dna methylation analysis. *bioRxiv*, 2019.

[32] Xiaoxiao Li, Nicha C. Dvornek, Yuan Zhou, Juntang Zhuang, Pamela Ventola, and James S. Duncan. Graph neural network for interpreting task-fmri biomarkers. In *MICCAI*, 2019.

[33] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. *CoRR*, abs/1511.05493, 2016.

[34] Ernesto Miranda-Morales, Karin Meier, Ada Sandoval-Carrillo, José Salas-Pacheco, Paola Vázquez-Cárdenas, and Oscar Arias-Carrión. Implications of dna methylation in parkinson's disease. *Frontiers in Molecular Neuroscience*, 10:225, 2017.

[35] Volodymyr Mnih, Nicolas Manfred Otto Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent models of visual attention. In *NIPS*, 2014.

[36] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5425–5434, 2016.

[37] Lisa Moore, Thuc Le, and Guoping Fan. Dna methylation and its basic

function. *Neuropsychopharmacology : official publication of the American College of Neuropsychopharmacology*, 38, 07 2012.

[38] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[39] Pereira R., Silke Weber, Christian Hook, Gustavo de Rosa, and João Papa. Deep learning-aided parkinson's disease diagnosis from handwritten dynamics. 06 2016.

[40] SungMin Rhee, Seokjun Seo, and Sun Kim. Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification. In *IJCAI*, 2017.

[41] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, Jan 2009.

[42] Seong-Jin Son, Mansu Kim, and Hyunjin Park. Imaging analysis of parkinson's disease patients using spect and tractography. *Scientific Reports*, 6(1):38070, 2016.

[43] Sven Suwijn, Caroline Boheemen, Rob de Haan, Gerrit Tissingh, Jan Booij, and Rob Bie. The diagnostic accuracy of dopamine transporter spect imaging to detect nigrostriatal cell loss in patients with parkinson's disease or clinically uncertain parkinsonism: A systematic review. *EJNMMI research*, 5:12, 12 2015.

[44] Devin Taylor, Simeon E. Spasov, and Pietro Lió. Co-attentive cross-

modal deep learning for medical evidence synthesis and decision making. *ArXiv*, abs/1909.06442, 2019.

[45] Qi Tian, Jianxiao Zou, Jianxiong Tang, Yuan Fang, Zhongli Yu, and Shicai Fan. Mrcnn: a deep learning model for regression of genome-wide dna methylation. *BMC Genomics*, 20(2):192, 2019.

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

[47] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *ArXiv*, abs/1710.10903, 2017.

[48] Qianfan Wu, Adel Boueiz, Alican Bozkurt, Arya Masoomi, Allan Wang, Dawn DeMeo, Scott Weiss, and Weiliang Qiu. Deep learning methods for predicting disease status using genomic data. *Journal of biometrics and biostatistics*, 9, 01 2018.

[49] Rex Ying, Jiaxuan You, Cynthia. Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018.

[50] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *ArXiv*, abs/1812.04202, 2018.

[51] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *ArXiv*, abs/1812.08434, 2018.

[52] Wanding Zhou, Peter Laird, and Hui Shen. Comprehensive characterization, annotation and innovative use of infinium dna methylation beadchip probes. *Nucleic acids research*, 45, 10 2016.