Universitat Politècnica de Catalunya
Facultat d’ Enginyeria de Telecomunicacions

Master in Advanced Telecommunications Engineering

Master’s thesis

Graph Neural Networks for Electroencephalogram Analysis

Dimas Ávila Martínez

Supervised by Sergi Abadal

July, 2022

Thanks to Sergi Abadal for supervising this work and the BNN members for helping with the technical stuff, in particular to Rocco and Axel. Also thanks to Neeraj Wagh and John Wei, who guided me to build my first GNN. Next, I would like to thank some people who is unrelated to this work, but they are important in my life and I want their names to be in my thesis. First, to my mother and my father who always supported me to study whatever and wherever I want. To my sister Iris, because she is one of the main reasons why I am an engineer. To my girlfriend Janira, who believed in me and my work even more than myself. And last but not least, to Jorge and Eva, who were with me in primary school, and they are still here while I finish writing my Master's thesis.

Abstract

The aim of this work is to provide a model able to identify Alzheimer's disease and Mild Cognitive Impairment (MCI) in electroencephalogram's (EEGs) recordings. Despite EEGs being one of the most common tests used for neurological disorders, nowadays the diagnose of these diseases is based on the patient's behaviour. This is because expert's accuracy on EEGs visual recognition is estimated to be around 50%. To solve the difficulties of the aforementioned task, this thesis proposes a Graph Neural Network (GNN) model to classify the subjects using only the recorded signals.

To develop the final model, first we proposed several procedures to build graphs from the EEGs signals, exploring different ways of representing the inter-channel connectivity as well as methods for relevant features extraction. For the time being, there are not GNN models proposed for Alzheimer or MCI detection. Hence, we used architectures employed by similar tasks and modified them for our specific domain. Finally, a set of coherent combinations of graph and GNN model is evaluated under the same set of metrics. Moreover, for the best performing combinations, a study of the impact of several hyperparameters is carried out. In order to handle all the possible experiments, we developed a software framework to easily build the different types of graphs, create the models and evaluate their performance.

The best combination of graph building and model design, based on graph attention convolutional layers, leads to a 92.31% of accuracy in the binary classification of healthy subjects and Alzheimer's patients and to a 87.59% of accuracy when also evaluating MCI patients recordings, these are comparable to state of the art results. Although this work is done within a novel field and there exist many possibilities yet to be explored, we conclude that GNNs show super-human capabilities for Alzheimer and MCI detection using EEGs.

Keywords

Graph Neural Network, Deep Learning, Electroencephalogram, EEGs, Alzheimer, Mild Cognitive Impairment

Contents

List of Figures

List of Tables

1. Introduction

1.1 Motivation

Alzheimer disease is a well known neurological disorder. The Alzheimer's Association [2] estimated that 10.7% of the population older than 65 years is affected by it. Unfortunately, the percentage of affected people increases with the age. On the other hand, Mild Cognitive Impairment (MCI) is not so well known. The Texas Department of State Health Services [31] defines the disease as *a borderline condition between normal, age-related memory loss and early Alzheimer's disease. Individuals with MCI have memory problems beyond what is expected for their age with no other clinical signs of dementia*. It also states that individuals with MCI have a greater chance of developing Alzheimer's disease. In 2019 Gillis, Cai *et al.* [11] estimated that 5.3% of the population over 75 years suffered from MCI, and the percentage increased up to 16.3% for ages over 85 years. In this context, developing techniques for the early and reliable detection of Alzheimer's disease (AD) and MCI is important to guarantee the well-being of the ageing population.

Nowadays, expert neuroscientists can only achieve a 50% of precision on AD diagnosis by electroencephalography (EEG) visual recognition. That's why the diagnosis is mainly based on the patient's behaviour, by analyzing their performance in tests that measure the reasoning and memorizing capabilities. The main drawback of the behavioural diagnosis is that is not an easy task to decide if the disease is Alzheimer's or an earlier stage of dementia. But it is decisive to be precise, because patients may need different levels of medical care depending on their condition. For this reason, several machine learning (ML) approaches have been developed to help identifying properly neurological disorders using EEGs.

The ML methods used to classify EEGs are diverse, and they would be further discussed in the state of the art chapter, but mainly they rely on a feature extraction phase and a classifier model, which usually is a Deep Neural Network (DNN). First approaches used Convolutional Neural Networks to process EEG recordings as matrices. Nevertheless, this architecture was not optimal to exploit the spatial dependencies and inter-channel connectivity because they are designed to deal only with grid-like data. Therefore, Graph Neural Networks (GNNs) have been introduced into this field because of the great expressive power of graphs, which are able to model relationships between electrodes in a much more natural way. Despite being many publications on similar fields such as epilepsy or schizophrenia classification using EEGs and GNNs, the Alzheimer's and dementia stages classification with GNNs remains almost unexplored.

1.2 Scope

Within the wide range of applications that EEG analysis has, the scope of this document is to describe the process of building a GNN based model able to classify healthy, Alzheimer's and MCI patients using only EEGs recordings as input data, which stands as an alternative to behavioral tests and human interpretation of EEGs. A report of the results obtained for all the techniques is also provided. Finally, a fair comparison of the results obtained with the ones from the main competitors in the field is given. To do so, the next steps will be followed:

- Explore and discuss feature extraction techniques relevant for EEGs.
- Inspect GNN models and architectures.
- Compare the performance of the models with the corresponding features.

- Describe the framework implemented to accomplish the previous steps.

1.3 Objectives

The main aim of the thesis is to find a GNN model that accurately classifies Alzheimer's diseases, MCI and healthy subjects, possibly exceeding the accuracy of other methods in the state of the art. For that, our main contributions are:

- Review the proposed GNN models for other mental diseases and EEG applications.
- Propose a method to build a graph from EEG signals that accurately represents the intra-channel information and the inter-channel connectivity.
- Provide a GNN model able to accurately solve the classification task using the graphs proposed in the previous step.
- Evaluate and compare the performance of the models and types of graphs proposed.
- Build a software framework to accomplish the previous objectives in an efficient and scalable way.

1.4 Thesis Outline

The remainder of the thesis is organized as follows: First, background about GNN and EEG in a high-level way is provided in Chapter 2. Next, the state of the art in GNN models being used to analyze EEGs and their applications is surveyed in Chapter 3. Coming up next, the methodology followed, starting with a description of the dataset used, the preprocessing of the EEGs, the models creation and how they are evaluated is described in Chapter 4. After the methodology description, the results obtained would be shown with a brief study of the hyperparameter and architectural decisions implications in Chapter 5. Finally, the conclusions containing a discussion of the obtained results and future work are outlined in Chapter 6.

2. Background

2.1 Brain activity measurement approaches

Neurological diseases, such as Alzheimer and MCI, directly affect to the brain's activity, which is manifested by electrical waves that communicate different areas of the brain at a neuron level. To measure this activity, there exist three main approaches:

- **Functional Magnetic Resonance Imaging (fMRI):** It measures the blood flow across the different brain areas. Higher activity is related to an increased flow. This technique is limited by the temporal and spatial resolution. It is not able to capture activity within a short time span, neither is capable of isolating small regions of neurons. In addition, the machinery needed is expensive and it is not available in all the health centers. However, it can produce a 3D brain image and visualize the blood flow and its directions in many brain areas.

- **Electrocorticography (ECoG):** This technique acquires the combined activity of millions of neurons by means of their oscillatory waves. It can deal with much higher temporal and spatial resolutions than fMRI. Nevertheless, it is a very invasive technique because it requires to insert an electrode under the scalp, so surgery is mandatory. This technique is only used with patients that have already scheduled brain surgery, it is not suitable for research purposes.

- **Electroencephalography (EEG):** This technique measures the electrical activity in the cerebral cortex. It is the less invasive method and the most affordable, it only requires a helmet containing a set of electrodes to be placed in the patient's head. In addition, the temporal resolution is much higher than the fMRI, it is in the range of 1 millisecond vs 1 second. On the other hand, the spatial resolution is poor because of the difficulty of precisely determining the region originating each signal. Unlike ECoG, the EEG cannot reach inner brain regions such as the hippocampus, which limits its applications.

The strengths and drawbacks of the aforementioned methods are summarized in Table 1. It can be concluded that the technique more suitable for research purposes is the EEG. Its simplicity and especially the non-invasiveness that characterizes the procedure make possible to record the activity of several subjects and generate datasets significant enough to conduct valid experiments. For these reasons, in the following sections, a deeper look into EEGs and their applications is taken.

Method	Temporal Resolution	Spatial Resolution	Invasiveness	Resources
EEG	High	Poor	Very low	Cheap
fMRI	Poor	Medium	High	Expensive
ECoG	High	High	Very High	Very Expensive%

Table 1: Brain activity measurement techniques

2.2 A deeper look into EEGs

The main idea behind an EEG is to capture the combined electrical signals produced by thousands of millions of neurons at the cerebral cortex level. To achieve that goal, several electrodes are placed around the head of the subject. Each one of the electrodes records a temporal signal measured in µV. Often, the signal obtained by an electrode is referred as channel. The EEG acquisition does not have a unique standard procedure. The methodologies differ in the number of electrodes employed and their locations.

Despite being different procedures, the majority of them are based on measurement system proposed in 1958 by H. H. Jasper [15], known as the 10-20 montage. This montage defines how to place a total of 19 electrodes for a proper EEG recording. The 10-20 name comes from the distance percentage between neighboring electrodes. Later, Oostenveld and Praamstra (2001) [32] updated the system to 21 electrodes, which became the 10-20 international standard. This standard distributed three electrodes for the frontal left, frontal right, central, parietal and occipital lobes and two electrodes for the temporal left and temporal right lobes. Figure 1 illustrates the described location of the electrodes:

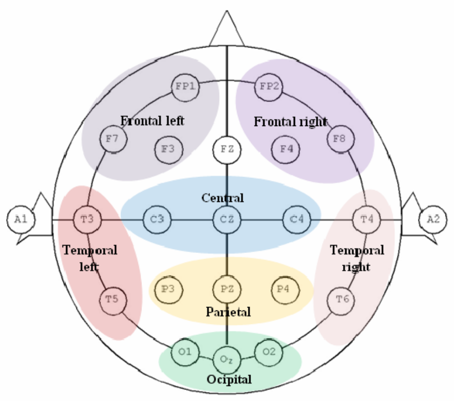


Figure 1: 10-20 international system electrodes distribution. Source: [38].

After this international montage system, several methods were proposed to record EEG. Some of the most used are the 10-10 and the 10-5 systems, which rely on higher number of electrodes by reducing the distance between each one of them. Within these standards, there can be found many modifications that span between 64 and 329 electrodes. ValerJurcak, Daisuke Tsuzuki *et al.* (2007) [24] provide an extensive revision of the most used systems.

The number of electrodes and their locations are not the only parameters that present variability in EEG setups. There are other variables that might change depending on the experiment's applications (see next section) and available resources. For example, the sampling frequency or the bit resolution per sample. Regarding the sampling frequency is standard to use at least 256 Hz, which is at least three times bigger than the highest frequency band of interest, found centered at 70 Hz. A typical range for the sampling frequency is between 256 Hz and 1024 Hz. The samples recorded are usually stored at least with a 16 bit representation, however this decision is up to the researcher of each experiment.

2.3 EEGs applications overview

There is a wide variety of applications in the fields of EEGs analysis. In this section, an overview of them would be provided, to illustrate the possibilities that EEGs offer. Later in the next chapter, a more technical review of the works most related to this thesis would be carried out. To begin with the applications, the area of neurological disorders diagnosis is one of the most explored, with applications like:

- Neeraj Wagh and Yogatheesan Varatharajah (2020) [43] proposed a GNN model for epilepsy classification. For this same purpose, Jialin Wang *et al.* (2020) [44] also proposed a GNN model to solve the task of epilepsy classification.

- Mohammad-Parsa Hosseini *et al.* (2021) [18] provided a review of machine learning for EEG signal processing for many applications of Biongineering such as seizure localization.

- Qi Chang *et al.* (2021) [5] developed a model to classify first-episode schizophrenia, chronic schizophrenia and healthy control.

- Cosimo Ieracitano and Nadia Mammone *et al.* (2020) [20] developed a Convolutional Neural Network (CNN) model to classify neurodegenerative states in dementia, which are Alzheimer and MCI.

It can be appreciated that the potential of EEGs classification goes beyond Alzheimer and MCI detection. Furthermore, there are other interesting applications related to detect the state of the subject or the task being done. Some examples in this field are:

- Andac Demir *et al.* (2021) [5] developed a model to distinguish errors during spelling tasks and for Rapid Serial Visual Presentation (RSVP), a scientific method for studying timing of vision.

- Ruilin Li *et al.* (2020) [28] provided a model to monitor driver's state of awareness, to prevent him for falling asleep while driving.

To finalize with this set of examples, two more futuristic applications are shown:

- Youngchul Kwak, Woo Jin Song and Seong Eun Kim (2020) [26] used multi-level fusion GNNs for developing a Brain Computer Interface to exchange information between the user's intention and the device control signals.

- Biao Sun *et al.* (2021) [37] developed a spatiotemporal GNN for motor imagery classification, in particular, to identify a set of movements of the right hand and feet.

From the above lists of applications examples, it can be concluded that machine learning algorithms are clearly the trend-line for solving EEGs classification tasks. Specifically, deep learning (DL) models have gained a lot of ground in the past years. Specially some particular architectures such as GNNs, CNNs, Recurrent Neural Networks (RNNs) and Multi-layer perceptrons (MLPs) combined with hand-engineered feature extraction methods. In Chapter 4, these architectures would be further detailed and how they are being adapted for EEGs analysis. Moreover, notice that the previous examples use many different approaches of feature extraction to feed into the models, this point would also be discussed.

2.4 Deep Learning for EEG classification: Main approaches

During the overview of EEG applications several architectures of DNNs were mentioned. This section provides a high-level description of them as well as an intuition about why they are suitable for solving EEG classification tasks.

2.4.1 Multi Layer Perceptrons

The first and more basic structure are the MLPs. These networks stack several layers, each one of them containing multiple perceptrons. The layers between the input layer and the final output layer are known as hidden layers. Each perceptron, can be defined by a set of weights, one for each element within the input data array, and a bias. By vertically stacking several perceptrons, a linear layer is obtained. When several layers are combined, each perceptron of one layer is connected to all the perceptrons in the next layer, that is why this layers are known as fully connected layers. The strength of each connection between two perceptrons, is defined by the weight. For a given input, these layers would apply the following transformation:

$$y = xA^T + b \tag{1}$$

where:

x = the input array
A = weights' matrix of dimension input size $\times$ number of weights
b = bias array of dimension 1 $\times$ number of weights

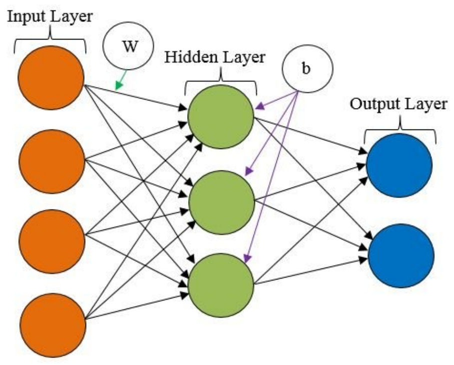


Figure 2: Multi Layer Perceptron. Source [48]

By stacking multiple linear layers and adding activation functions in between, a MLP is created. The last linear layer, contains as many perceptrons as classes in the classification task. Each one of them outputs a logit, which represents the confidence of the networks of a given input belonging to each class. These logits are inside the $(-\infty, \infty)$ interval. However, the final goal is to estimate the probability of a given input to belong to each one of the possible classes. Hence, the softmax function is typically applied. The softmax maps the logits of each class to the interval $[0, 1]$, obtaining the desired probabilities:

$$Softmax(x) = \frac{exp(x_i)}{\sum_{i=1}^{N} exp(x_i)} \tag{2}$$

MLPs can be used as universal function approximators [17] as they learn the weights' matrices and bias during training. Therefore, given an array of features, they learn to decide which of the classes is the most probable.

In the EEGs field, they have being used for classification tasks combined with hand-crafted features extraction techniques. However, they are also used as a building block for more complex networks, letting these networks to perform the feature extraction and using the MLPs as the final classifier. Figure 2 shows a visual description of the architecture. Indeed, they are used in all the architectures that are described below.

2.4.2 Convolutional Neural Networks

CNNs are a huge topic with enormous research about them during the past years. The variability in architectures, applications, types of layers is remarkable. In this section, the minimal background needed to understand their applications in EEG classification is given.

CNNs are architectures specially developed to handle grid-like data, in other words, matrices. For this reason, they are the preferred architecture in the computer vision field. The main idea behind CNNs are the convolutional layers. These layers, apply bi-dimensional filters to extract features from the input data. The power of this architecture comes from the capability of learning the filters' weights during the network training process.

The result of the convolution of the input matrix with a filter, is known as feature map. Each convolutional layer, applies many filters of the same dimensions and then stacks depth-wise each one of the outputs to produce a volume of feature maps. Furthermore, the feature volumes can be forwarded to another convolutional layer containing another set of learned filters, that can be of different size than the ones from the previous layer. By applying this process, the network can learn to extract relevant features that are rotation and scale invariant [35].

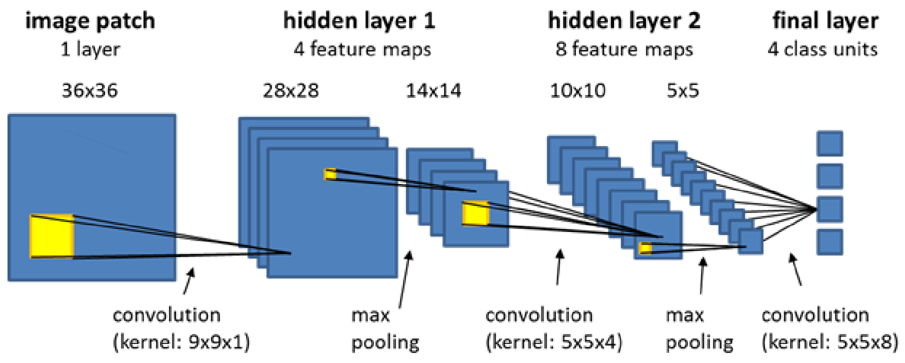


Figure 3: Convolutional Neural Network architecture. Source: [1]

To reduce the large dimensionality of the problem given by the size of the matrices and the large number of filters stacked, pooling layers are used. These layers apply several strategies, such as averaging or maxima selection in a small neighborhood, to reduce the sizes of the feature maps. In addition, batch normalization layers are used to normalize the feature maps. These layers shown several benefits such as speed up the training process and help the gradient flow better through big CNNs [3].

Figure 3 shows a simple CNN architecture. Usually, a convolutional layer followed by a pooling layer and a batch normalization layer are combined into a convolutional block. Then, the output of one block is used as the input of the following one, with an activation function in between. The combination of these blocks has the capability of finding from high-level generic patterns to fine granular patterns.

[1]Image source: `https://docs.ecognition.com/v10.0.2/eCognition_documentation/User%20Guide%20Developer/8%20Classification%20-%20Deep%20Learning.htm`

For the purpose of classification, a MLP is placed on top of the convolutional blocks, using the extracted features from the CNN to decide which is the most likely class for a given input. In the EEG classification field, a CNN is quite a natural choice because of the structure of the data. A matrix can be built by vertically stacking the recordings of each channel, to obtain an input of (numbers of channels, samples per channel) size. The channels can be ordered following EEG montage used during the recording, such as the 10-20 international standard [15]. Moreover, this solution provides flexibility on how the data of each channel is represented.

2.4.3 Recurrent Neural Networks

RNNs are an architecture specially designed to deal with sequences of variable length. The main strength of these networks is to capture temporal patterns in dynamic data. The fully connected layers of an MLP connect each node to the ones in the next layer, but nodes inside the same layer are independent from each other. In RNNs, this is no longer true as each node in the same layer receives two inputs: the first one is the the input at the current time step x_t, whereas the second input received is the output of the previous node, which has processed the input at the previous time step, x_{t-1}. These are known as hidden states. By repeating this process, the network develops memory, because for every input is also considering the previous states. Figure 4 illustrates the basic structure of a recurrent layer:

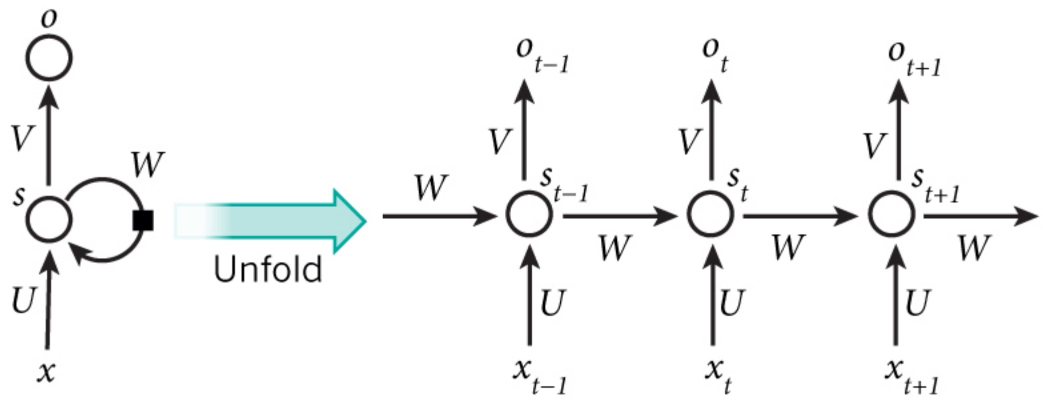


Figure 4: Recurrent Neural Network base architecture. Source: [2]

Consequently, the output of a recurrent layer can be computed as follows:

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \tag{3}$$

$$y_t = \sigma_y(W_y h_t + b_y) \tag{4}$$

where:

[2] Image source: `https://iis-projects.ee.ethz.ch/index.php?title=File:Rnn.jpg`

x_t	= the input array
h_t	= hidden layer vector
W, U, b	= Weights and bias
y_t	= output array
σ_h, σ_y	= activation functions

However, RNNs have a significant problem: forgetting or fading memory [14]. The deeper the network, the harder it becomes preserving long-time relationships. To solve that problem, new architectures were proposed based on gated units. The most popular and widely used units are Long short-Term Memory (LSTM) [16] and Gated Recurrent Units (GRU). In the field of EEG classification tasks, LSTM is the option that stands out. In a nutshell, a LSTM gate controls the information to be kept or forgotten at a given time step. To do so, it relies in four main blocks:

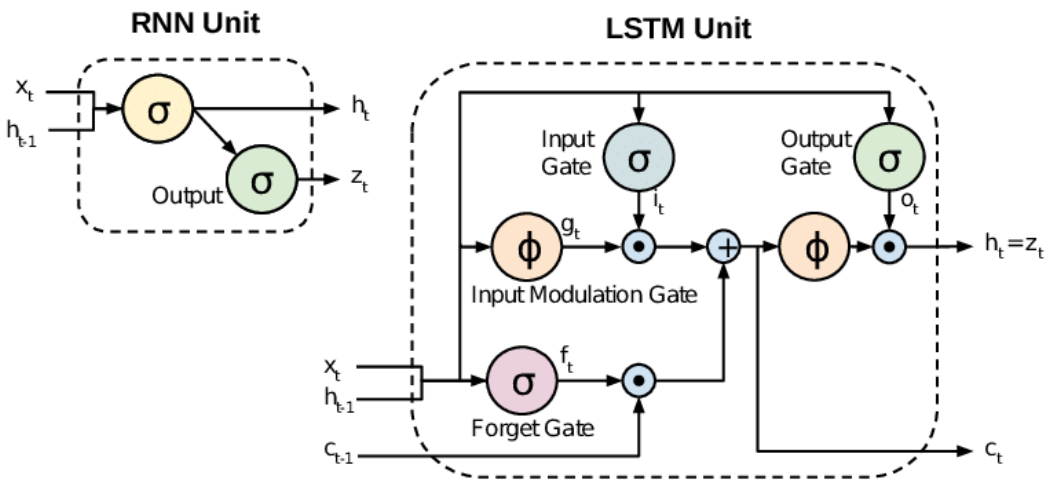


Figure 5: LSTM Gate compared to base RNN unit. Source: [3]

- **Forget gate:** This gate receives the information of the current input and the previous hidden state. Its function is to decide which information should be kept and which shall be forgotten.
- **Input gate:** It receives the previous state and the current input. Its main task is to decide which information must be updated.
- **Cell state:** To compute the cell state, the current state is multiplied point-wise by the output of the forget gate. Then, it is point-wise added to the output of the input gate. That produces the new cell state.
- **Output gate:** This gate is the one responsible of deciding the next hidden state. To do so, the information of the previous hidden state and current input are multiplied by the updated cell state. The output is the hidden state that would be forwarded to the next cell.

Figure 5 provides a visual description of the previously explained blocks. In the field of EEG processing, RNN based on LSTM gates are used to detect temporal patterns along the channels, and then classify

[3]Image source: `https://www.researchgate.net/figure/RNN-simple-cell-versus-LSTM-cell-4_fig2_317954962`

them using an MLP. Moreover, some works combine LSTMs with GNNs to first extract temporal features and then extract spatial relationships [22].

2.5 Graphs and Graph Neural Networks

This section aims to provide the necessary background about graphs and GNNs that may be needed to follow the rest of the work. Notice that this chapter is focused on the applications of GNNs which are relevant for EEG classification. It is not intended to provide a wide review of all the different architectural possibilities and tasks that can be solved with GNNs, which represents a huge design space. For that, we refer the interested reader to [1], [51]. To understand the needs of GNNs and why are they powerful, it is essential to have some knowledge about graphs.

2.5.1 Graphs as data structures

Graphs are data structures formed by two elements: nodes and edges. On one hand, the nodes represent the main objects or entities present in the data to be modeled. On the other hand, edges represent the connectivity between those objects, this is, they express how the nodes are related to each other or even how strong is their connection. The most typical and comprehensive example of a graph is the users of a social network, where the nodes represent the users of the network and can contain information describing the user: the name, age, location, interests, email and so on. In turn, the edges indicate the relationship between users. For example, if $user_1$ follows $user_2$, there would be an edge from $user1$ to $user_2$. If two users are not related in any way, then there would not be any edge between them.

There are many areas and examples where graphs are used, some of them are to model molecules, to describe physical systems, to model the different scenes in an image or describing how web pages are linked. The large variety of examples directly leads many different expressions of graphs. Next, we will go through the most relevant, following the classification given by Jie Zhou, Ganqu Cui, *et al.* (2020) [51]:

- Graphs can be either directed or undirected depending on the edges. If the edges are directed, it means that the connection begins in node X and ends in node Y. This can be the case of users in Twitter, a directed link means that user X is following user Y, but user Y does not follow user X. Undirected graphs represent that if nodes are linked, then they are connected in both directions. This can be the case of Facebook users, when user X and Y become friends, it means that both follow each other.

- Graphs can be homogeneous or heterogeneous. In homogeneous graphs, all nodes and all edges contain the same type of information (same number of features). However, heterogeneous graphs may contain different information in every node and edge. An example would be the recommendation graph for stores like Amazon, where nodes can be users or products and the main relationships are established between users and products.

- Depending on the evolution of the features stored in a graph, they can be static or dynamic. These classification refers to whether the information within the graph evolves over time (dynamic) or not (static). A static graph can be the relationship between users in a social network for a given time stamp. However, users follow or unfollow other users, new people join the social network, and so on. So if we model this evolution, we have a dynamic graph, that represents users interactions over time.

Graphs are defined as $G = (V, E)$, where:

$|V| = N$, denotes the number of nodes
$|E| = N^e$, denotes the number of edges

The connectivity is given by the Adjacency Matrix A. This is a binary matrix of dimensions $N \times N$. If node i is connected to node j, then a 1 is placed at the position (i, j) in the adjacency matrix, 0 otherwise. With that being said, to model a graph from an EEG, it would have to be decided if: there is a sense of direction when connecting signals from two electrodes, also if every electrode must be represented by the same set of features or on the contrary, the features may vary depending on the channel. Finally, if the features representing the electrodes signals shall vary over time or if they can be summarized into a static representation.

2.5.2 GNN applications

Before explaining GNNs, it is convenient to be familiarized with the kind of tasks that they can carry out. They can be grouped in three main categories, depending on the level of interaction with the graph:

- **Node-level tasks:** The GNN performs one of the following functions related to the nodes in the graph:
  - Node classification: The network intends to categorize each node within a set of predefined classes.
  - Node regression: The network predicts a continuous value for a node.
  - Node clustering: Group nodes in several disjoint groups by their similarity.
- **Edge-level tasks:** The GNN performs one of the following functions related to the edges in the graph:
  - Edge classification.
  - Link prediction: Predict possible links between edges.
- **Graph-level tasks.** This group involves the whole graph to perform one of the following tasks:
  - Graph classification.
  - Graph matching: Compare two graphs to determine if they belong or not to the same element.
  - Graph regression.

These tasks have direct implications on the type of output of the network and the loss function used to train it. In this particular scenario, the GNNs are trained to perform Graph classification. Graphs would be built from EEG recordings and the network must try to predict one of the three classes: healthy, Alzheimer or MCI.

2.5.3 Graph Neural Networks

GNNs are deep learning models specifically designed to process graph data. They can be understood as an extension or generalization of CNNs to provide capabilities to process non grid-like data. This generalization of deep learning models to a non-Euclidean domain is known as geometric deep learning [4]. To sum up, the main idea behind GNNs is to extend operators such as convolutional layers or pooling layers to be applied in graphs, which do not have a straight-forward sense of distance or order like in images.

One of the main building blocks of a GNN are the graph convolutional layers. This layers are known as propagation modules. The main function of these modules is to propagate information between nodes in order to each node can aggregate information from its neighbours. To do so, a general framework is implemented known as message passing.

Message passing is the procedure used to combine features from linked neighbouring nodes. This method is not uniquely defined, it has many implementations depending on how messages are computed at each node and how the received messages are aggregated. Following, the base procedure would be defined to have a general understanding of the process.

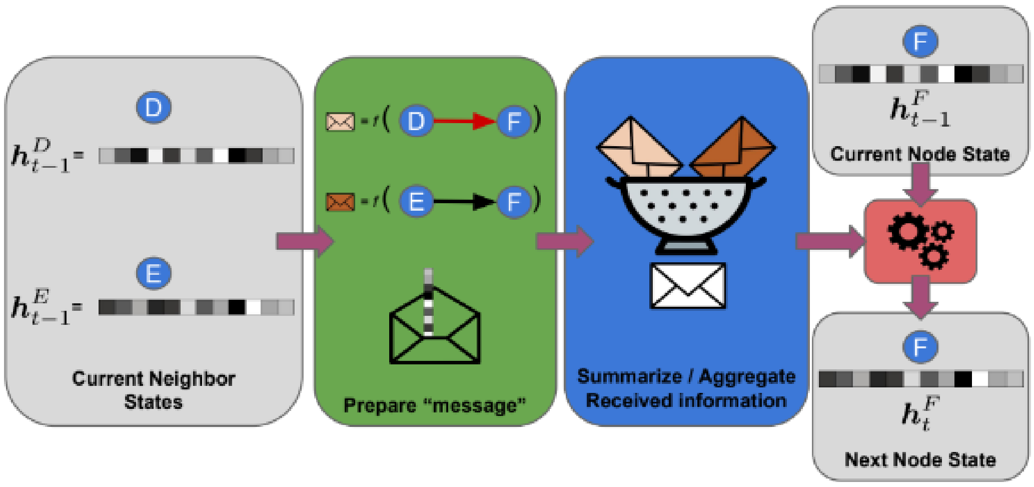


Figure 6: Message Passing general schema. Source: Microsoft Research [4]

The message passing process illustrated in Figure 6 is iterative. It means that at each step, every node produces a message, propagates it and aggregates the received one, and all the nodes do it synchronously. The process of message passing can be summarized in three main steps:

1. **Message computation:** each node in the graph computes its message. The computation is the result of a function that takes into account the state of the current node, the state of the destination node and the edge that links them.

2. **Message aggregation:** Each node combines the information received in all the messages. There are several possibilities for defining the aggregation function, however, it should be permutation-invariant.

3. **Update:** Each node updates its attributes as a function of the aggregated messages.

If all the previous steps are formulated, the following expression is obtained:

$$H_t = \sigma(D_{IN}^{\frac{-1}{2}} A^T D_{IN}^{\frac{-1}{2}} H_{t-1} W_t + b_t) \tag{5}$$

where:

[4]Image source: `https://www.microsoft.com/en-us/research/video/msr-cambridge-lecture-series-an-introduction-to-graph-neural-networks-models-and-applications/`

H	= Matrix of nodes features
A	= Adjacency matrix
D	= Degree Matrix
W	= Trainable weights matrix
σ	= activation function

It can be compared to an image convolution, where at each step the weights of the filter are multiplied by the values of the image and then aggregated in the central pixel. In the case of graphs, at each step the hidden state of the central node and its neighbours is used to create a message. Then the messages received at one node are aggregated. This process allows to propagate node features throughout the graph.

Notice that for the first step, each nodes receives information only of its directly linked nodes. However, for each farther step, it would also receive, indirectly, messages from distant nodes. This phenomenon is known as the increasing receptive field. As a consequence, after several steps, the over-smoothing [6] effect can be produced. When this happens, all the nodes in the graph carry a global averaged information, and the gradients can not flow properly. This is one of the main difficulties when trying to build deep architectures with base graph convolutional layers.

As it was previously mentioned, there exist many different approaches to carry out graph convolutions. For example, there are recurrent convolutions such as the ones implemented in RENet [23]. And also there are graph convolutions that incorporate attention mechanisms, such as the proposed by Petar Veličković, Guillem Cucurull, *et al.* (2017) [40]. As Graph Attention Networks (GAT) are relevant for this work and would be mentioned in the following sections, the next subsection would be dedicated to understand them

2.5.4 Graph Attention Networks

The main idea and strength behind graph attention network is the capability of learning the relationships between nodes, instead of using predefined weights for the edges or computing them based only in structural characteristics of the data such as the degree matrix. In a nutshell, attention is a mechanism to assign different levels of importance to samples in a given sequence. By doing this, the network knows the most relevant features and how are they related. In the particular case of GAT, the features in the nodes are used to learn the edges' weights. To do so, the next procedure is applied:

Given a set of nodes, each one with a vector of features $\mathbf{h} = \{\overrightarrow{h_1}, \overrightarrow{h_2}, ..., \overrightarrow{h_N}\}$. To compute the attention coefficients, a shared attention mechanisms is applied:

$$e_{ij} = a(\mathbf{W}h_i, \mathbf{W}h_j) \tag{6}$$

where:

W	= Trainable weights matrix
a	= attention mechanism
h_i, h_j	= array of node features
e_{ij}	= attention weights

In a high-level, e_{ij} *indicates the importance of node's j features to node i* [40]. The coefficients are then normalized to make them comparable across nodes using a softmax:

$$\alpha_{ij} = softmax_j(e_{ij}) = \frac{exp(e_{ij})}{\sum_{k \in N} exp(e_{ik})} \tag{7}$$

Finally, the weights computed by the attention mechanisms are computed as:

$$\alpha_{ij} = \frac{exp(LeakyReLU(\overrightarrow{a}^T[W\overrightarrow{h_i}||W\overrightarrow{h_j}]))}{\sum_{k\in N_i} exp(LeakyReLU(\overrightarrow{a}^T[W\overrightarrow{h_i}||W\overrightarrow{h_k}]))} \tag{8}$$

where:

a = weight vector parameterized by a feedforward linear layer
$||$ = vector concatenation

In this work, we rely on several implementations of graph attention layers to allow the network learn the relationship and strength of links between nodes. By doing this, it is avoided the need of static engineered connectivity features and the choice is leveraged to the network learning.

2.5.5 Graph Pooling Layers

In CNNs, it is very common to use pooling layers after convolutional layers to reduce the dimensionality of the problem and collapse the information across channels. In the graph domain, pooling methods are used to extract rich latent structures and high level graph representations. In this thesis, pooling methods are not deeply explored, however a minimal background is provided with the most used graph pooling algorithms. Graph pooling can be classified in two main families: hierarchical and direct pooling methods [51]. Direct pooling methods consists on a set of strategies to learn graph-level representations using the information present in the nodes. The most common methods for direct pooling are the following ones:

- **Set2Set:** It is designed to deal with unordered sets. It relies on LSTM gates to produce a graph representation invariant to order (Vinyals *et al.* 2016)[41].
- **SortPooling:** This method does sort the node embeddings according to graph structural criterion. Once they are sorted, the ordered embeddings are fed into a CNN to produce the graph representation (Zhang *et al.*, 2018)[50].

Previous methods do not consider a hierarchical structure within the graphs. However, the hierarchical pooling methods assumes hierarchy and uses several layers to learn the graph representation. Following some of the most popular methods are briefly described:

- **DiffPool:** This method "learns a differentiable soft cluster assignment for nodes at each layer of a deep GNN, mapping nodes to a set of clusters, which then form the coarsened input for the next GNN layer." (Ying *et al.* 2016)[49].
- **gPool:** This method relies on a projection vector to learn projection scores for each node, then selects the top-k highest scores. Unlike DiffPool, that uses a matrix, gPool uses a projection vector, but it does not take into account the graph structure to learn the scores. (Gao *et al.*, 2018)[9].
- **SAGPool:** Applies graph convolution with self-attention taking into account the topology of the graph and the nodes features (Lee *et al.*, 2019)[27].

So far, the most typical architectures of deep learning methods, such as CNNs, LSTMs GNNs, and MLPs have been explained. In GNNs, as the main topics, the principal layers such as convolutional and pooling layers have been extensively described. In addition, an introduction of how these methods are used for EEG classification tasks has been provided. At this point, a solid background has been stated to understand the state of the art methods on EEG classification using GNNs and in Alzheimer's detection.

2.6 Challenges of current AI for EEG

When designing deep learning models for EEG classification tasks, there are several challenges intrinsic to the data. As mentioned before, there are several standards describing possible montages for EEG recording. The variability in the number of electrodes, their location or the used sampling rate introduces an extra challenge into the task.

It is known that the spatial resolution in EEGs is poor. This drawback introduces the fake correlations term. Fake correlations occur when the electrical signals of two channels seem to be correlated, but the correlation comes from neighbouring electrodes affecting the measure of the channel being observed. This introduces a huge channel, because the true dependencies between brain areas turns very complicated to quantify.

Another problem is the relevance of the channels. As visual diagnosis carried out by experts is not reliable, it is difficult to decide which channels are more relevant when trying to detect neurological diseases. Indeed, in GNNs architectures exist the application of channel selection, which tries to answer this question. For this same reason, another challenge is the temporal span of relevant events. EEGs are usually recorded during several minutes, however, brain responses happen in the range of milliseconds. Distinguishing this events and finding their temporal resolution is not an easy task.

Finally, the end-to-end learning is also a challenge. Many state-of-the-art solutions use complex hand-crafted features to train the network, which involve some sort of prior knowledge like the spectral activity of the signals. Nowadays, using the raw signal to train the network is still a challenge in numerous applications. In particular, for GNNs how the graph is built from the EEGs is one of the most relevant decisions, and authors propose many different approaches as we will see in the next chapter.

3. State of the art

As it was stated in Section 2.4, there are many examples in the literature where GNNs are applied to solve a wide variety of tasks related to EEGs. This chapter would focus mainly on two objectives. The first one, to review GNNs architectures that solve neurological diseases classification tasks and also GNNs that analyze EEGs for other applications such as BCI, but present some differentiable characteristic. The second one is to review ML and DL based methods of Alzheimer and MCI classification.

3.1 GNNs for EEG classification related work

The following tables summarize several papers including the task being solved, the model architecture, how graphs are built and the results obtained. First, Table 3.1 summarizes the publications, tasks and results obtained of several EEG classification applications:

ID	Publication	Task	Results
1	EEG-GNN: Graph Neural Networks for Classification of Electroencephalogram (EEG) Signals [8]	Classification of correct / incorrect feedback during spelling task (ErrP dataset). Classification between tasks: emotion elicitation, resting state, motor imagery, execution task (RVSP dataset)	ErrP 76.73% Acc. RSVP: 93.49% Acc
2	EEG-GCNN: Augmenting Electroencephalogram-based Neurological Disease Diagnosis using a Domain-guided Graph Convolutional Neural Network [43]	Classification between healthy subjects and epilepsy patients	0.90 AUC 93.00% Acc
3	Graph Neural Network with Multi-level Feature Fusion for EEG based Brain-Computer Interface [26]	Classification between 5 different motor imagery actions	92.40 % Acc
4	Classification of First-episode Schizophrenia, Chronic Schizophrenia and Healthy Control Based on Brain Network of Mismatch Negativity by Graph Neural Network [5]	Classification between schizophrenia, chronic schizophrenia, and healthy control group	84.17 % Acc
5	A Sequential Graph Convolutional Network with Frequency-domain Complex Network of EEG Signals for Epilepsy Detection [44]	Classification between epilepsy, and healthy control group	89.30 % Acc

Table 2: Summary of publications using GNNs to solve EEG classification tasks

Table 3 summarizes different procedures used to build a graph from the raw EEG recordings. It focuses on the general graph structure, the node features (or embeddings), node linking rules and edge weighting measures.

ID	Graph Structure	Node features	Edge features
1	Static, homogeneous, undirected graph. Fully connected or node linking depending on a policy.	Every node stores the raw features of a EEG channel. 56 nodes for ErrP, 16 nodes for RSVP.	Fully connected with unweighted edges, weighted edges using the Pearson Correlation Coefficient (PCC) (9), only connect nodes closer than a heuristic distance, construct the adjacency matrix applying k-NN over node features.
2	Static, homogeneous, undirected graph. Fully connected.	Power Spectral Density (PSD) (10) computed at 6 frequency bands: delta (1-4Hz), theta (4-7.5Hz), alpha (7.5-13Hz), lower beta (13-16Hz), higher beta (16-30Hz), and gamma (30-40Hz)	Expected spectral coherence (11) added to normalized geodesic distance between each pair of channels.
3	Static, homogeneous, undirected graph. Fully connected.	Each node contains the raw samples from one channel. 64 nodes.	Graph Laplacian (12) computed from Absolute PCC matrix.
4	Static, homogeneous, undirected graph.	Characteristic Path Length (16), Global Clustering Coefficient, Closeness Centrality. Before the MLP classification layer, the following features are concatenated: age, education, IQ and duration of illness.	Two variants: Partial Correlation Coefficient or Phase Lagged Index (PLI).
5	Static, homogeneous, undirected. The connectivity depends on a defined policy.	Each node contains the Fast Fourier Transform (FFT) of one channel.	Node i would be linked to node j if the following rule is true: $(\lvert f_k\rvert - \lvert f_i\rvert)/(k-i) < (\lvert f_j\rvert - \lvert f_i\rvert)/(j-i), \forall(i<k<j)$ being f_i, f_j two different data points in the frequency domain and f_k any data point in between the past two frequencies.

Table 3: Graph building procedures for GNNs found in the literature for EEG analysis.

It is important to mention that the general procedure for building a graph implies windowing the full EEG recordings. The graph construction is carried out at window level, usually these windows have a duration between 5 and 10 seconds, depending on the sampling frequency and the task carried out by the subjects during the recording. These windows are considered as independent samples during the network training. Then, the validation of the model is done at subject level, by comparing the average output of each window belonging to the subject with the full recording ground truth.

As it can be observed in Table 3, there are many different approaches for extracting features, from simply using the raw data for building the nodes, to more sophisticated measures such as the cross spectral coherence. Next, the mathematical expression of the methods explained would be stated, as well as a high-level intuition of why the measures used can be relevant for EEG detection.

- The Pearson Correlation Coefficient (PCC) is a statistical measure of linear correlation between two arrays of data. It is bounded within the $[-1, 1]$ interval. It can be used to know if there is a linear correlation between the samples generated by two different electrodes, if it is the case, a strong connection is assumed. The more correlated the samples of two channels are, the larger the weight of the edge connecting both nodes would be. The PCC of two channels x and y can be estimated as follows:

$$\rho_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{9}$$

where:

x, y = data vectors
$\bar{x}, \bar{y}$ = mean of the vectors

- The PSD describes how the power of a signal is distributed along its frequency components. In EEGs, it can be used to measure the brain activity at several frequency bands. The spectrum of a discrete signal can be computed as:

$$\bar{S}_{xx}(f) = \lim_{N\to\infty} (\Delta t)^2 \left| \sum_{n=-N}^{N} x_n e^{-i2\pi fn\,\Delta t} \right|^2 \tag{10}$$

where:

x = data vector
Δ_t = time increment
f = frequency variable

However, in practice, the spectrum is estimated using the Welch's method [46]. Then, to compute the power within a given band, the spectrum is integrated following the Simpson's integration method between the cut frequencies.

- The spectral coherence between two signals is a statistical measure of relation. In this case, it can be interpreted as a measure of similarity between the spectrum produced by two different electrodes. The coherence is a real scalar function, and to be used as a weight for the edges, the authors employed the expected spectral coherence, defined as:

$$\bar{C}_{xy}(f) = \frac{|E[S_{xy}]|}{\sqrt{E[S_{xx}]E[S_{yy}]}} \tag{11}$$

where:

S_{xx}, S_{yy} = PSD estimates of each channel
S_{xy} = Cross PSD estimates between channels
E = Expectation

- The laplacian graph matrix, in a high-level intuition, represents how smooth are the transitions between features of connected nodes. This matrix is defined as:

$$L = D - A \tag{12}$$

where:

A = Adjacency Matrix
D = Degree matrix

$$A = |P| - I_N \tag{13}$$

$$D_{ii} = \sum_{j-1}^{N} A_{ij} \tag{14}$$

where:

$|P|$ = Absolute Pearson Correlation Matrix
I_N = Identity matrix of size N channels

In practice, the normalized graph laplacian is generally used:

$$L = I_N - D^{-1/2} A D^{-1/2} \tag{15}$$

- The characteristic path length is a measure that comes from network theory. It computes the average shortest path between every pair of nodes. It is one of the most robust and widely used measures over graphs. The characteristic path length gives a measure of how well connected or dense is the network. It can be computed as follows [29]:

$$\bar{d} = \frac{1}{|V|} \sum_{i=1}^{N} \bar{d}(i, j) \tag{16}$$

where

$$\bar{d}(i) = \frac{1}{(|V| - 1)} \sum_{j=1}^{N} d(i, j) \tag{17}$$

$\bar{d}(i)$ = average distance from a node i to all the remaining nodes
$d(i, j)$ = Minimum number of edges traversed from node i to node j

Next, for each one of the graphs described in Table 3, the GNN model used to process them is described. The description would focus on the number of layers, the type of graph convolution used and the dimensions at each layer. This would provide an overview of the architecture of neural networks capable of resolving EEG classification tasks.

ID	GNN Architecture
1	2 x (SAGE Conv. layer [12] + ReLU) Readout function (sum of node representations) MLP + Softmax
2	5 x (GCN Conv. layer [33] + Batch Normalization + Leaky ReLU) Global Mean Pooling MLP(3 x FC layers) + Softmax
3	6 x (Chebyshev Conv. layer [7] + Max Pooling + SoftPlus) Skip Connections from 1st and 2nd conv. layers + MLP Feature concatenation from skip connections and output of convolutional block MLP + L2 Normalization + Softmax
4	4 x (Chebyshev Conv. layer [7] + Top-k-pooling + L2 normalization + ReLU) Flatten Layer Concatenation with quantitative indexes MLP + Softmax
5	k-hops node aggregation [30] 2 x (SG Conv. layer [47] + Max-pooling + ReLU) MLP (2 x FC Layers) + Softmax

Table 4: GNN Architectures for EEG classification found in the literature.

From Table 4, it can be concluded that GNNs for EEG classification have several steps in common. First, a set of graph convolutions (many different definitions of convolution can be used) to extract features from the input data. Depending on the dimensions of the data, several pooling strategies can be applied to reduce it (at graph level or at node level) after convolutional layers. Finally, the features extracted can be aggregated or reshaped to be fed into a MLP and a softmax, that would output the estimated probabilities for each class.

3.2 Alzheimer and MCI classification related work

3.2.1 Classical Machine Learning approach

In 2020, Cosimo Ieracitano, Nadia Mammone, Amir Hussain, and Francesco C. Morabito published a novel multi-model machine learning approach to classify states of dementia in EEGs [20]. The dataset used in the publication was the same that the one used in this work (19 channels per trial following the 10-20 international standard). DNN are not used in this publication; instead, they focus on specific feature extraction based on prior knowledge about how dementia affects the brain activity in the frequency domain. For classifying the features, different classifiers such as Support Vector Machine (SVM), MLP or Logistic Regression (LR) are compared.

First, the dataset is preprocessed: original samples are recorded at 1024Hz and they are downsampled to 256Hz, then a notch filter at 50Hz is applied to remove the possible effects of direct current. Also, a band pass filter in the 0.5Hz to 32Hz band is applied. Then, every recording is windowed in 5s windows (1280 samples per channel, 19 channels in total). The following procedure of feature extraction is performed over these windows independently, not over the whole recording of a single patient.

For the feature extraction phase, they focus on two different techniques. For both techniques, the analysis is focused in five different frequency bands known as: δ [0.5–4 Hz], θ [4–8 Hz], α_1 [8–10 Hz], α_2 [10–13 Hz], β [13–32 Hz].They compare the performance of the classifiers when both types of features are used together and when only one type of features is used for the task. In this case, the methods used in feature extraction are:

- **Continuous Wavelet Transform (CWT):** The CWT is applied independently to each one of the 19 channels. Then the time-frequency signals from all the channels are averaged into a single representation. Using the averaged time-frequency signal, the mean, standard deviation, skewness, kurtosis, and entropy are extracted for every frequency band. This produces a feature vector of 25 samples (5 statistical moments x 5 frequency bands).
- **Bispectrum Analysis (BiS):** The BiS is a method to quantify the non-linear interactions and the deviations from normality between two signals. It is defined as:

$$BiS(f_1, f_2) = E[S(f_1)S(f_2)S^*(f_1 + f_2)] \tag{18}$$

  In this case, the same procedure as in CWT is applied. However, instead of extracting statistical moments, the following measures are computed for each frequency band:

  – Normalized bispectral entropy.
  – Normalized bispectral squared entropy.
  – Sum of logarithmic amplitudes of the bispectrum.
  – Sum of logarithmic amplitudes of diagonal elements of the bispectrum.
  – First order spectral moment of the amplitudes of diagonal elements of the bispectrum.
  – Second-order spectral moment of the amplitudes of diagonal elements of the bispectrum

  Hence, a feature vector of 30 samples is obtained (6 bispectrum features x 5 frequency bands).

For the classification task, the the following classifiers are compared:

- Auto-Encoder (AE): Two different AE versions.
- MLP: Two different MLP versions.
- LR
- SVM

Each one of the previous classifier, is trained in three different scenarios:

- Using only CWT features (25 samples feature vector).
- Using only BiS features (30 samples feature vector).
- Using CWT and BiS features (55 samples feature vector, both types of features merged by concatenation).

Each classifier, using each one of the possible feature combinations, is trained to solve the following tasks:

- AD vs. HC
- AD vs. MCI
- MCI vs. HC
- AD vs. MCI vs. HC

For all the possible combinations above, the result is evaluated with a test dataset. Even though the classifiers are trained using the 5s windows as independent samples, the final evaluation is done at subject level. This implies evaluating every window of a subject and deciding the class by the majority of the labels within the set of windows. To evaluate the performance of each classifier the metrics used are: Precision, Recall, F1-Score and Accuracy (see Section 4.5 for a description of such metrics). The best results obtained for each classification task are summarized in the following table:

Task	**Best Classifier**	**Best Features**	**Test Accuracy**
AD vs. HC	MLP	CWT + BiS	96.95%
AD vs. MCI	MLP	CWT + BiS	90.24%
MCI vs. HC	MLP	CWT + BiS	96.24%
AD vs. HC vs. MCI	MLP	CWT + BiS	89.22%

Table 5: Results of feature extraction in several classification tasks from [20]

It can be concluded that the best combination for every classification task is using both types of features with a MLP as a classifier. In average, the MLP showed to be 5% better than the other classifiers while using the same features. The MLP used consists on 3 FC layers, the input layer (55 units), a single hidden layer (30 units) and the output layer (2/3 layers, depending on the number of classes).

3.2.2 Deep Learning approach

In this section, two publications that use DL approaches to classify AD and MCI would be summarized. The first one still uses a feature extraction stage with hand-crafted methods. The second one bets for an end-to-end DNN to directly classify the raw EEG recordings. Both methods are from the same authors than the previous work and employ the same dataset with the same preprocessing and windowing strategy.

The first publication proposes a CNN to classify dementia stages based on 2-D spectral representation of EEGs [19]. The method proposed consists of a feature extraction stage and a classification stage. The steps followed are:

1. The PSD is estimated for each one of the 19 channels using the modified periodogram (windowed periodogram). The periodogram is estimated in the 0.5-32Hz, taking one sample every 0.2Hz, for a total of 159 samples per channel.

2. The PSD matrix is built by normalizing the features within the [0, 1] interval and vertically stacking channels following the order of the 10-20 montage. The result of this process referred as the PSD image.

3. The PSD images are fed into a CNN, the four types of classification tasks described in Table 5 are performed.

The architecture of the top performing CNN proposed for PSD image classification is the following one is shown in Table 6. It is important to mention that the authors of [19] also evaluated a deeper architecture with more convolutional layers and larger number of filters, but it performed worse than the aforementioned CNN. In this case, the accuracy over the test dataset is provided at window level, so the accuracy is computed considering only the given window and not all the windows from the same recording. The accuracies obtained are: 92.95% AD vs. HC, 84.62% AD vs. MCI, 91.88 MCI% vs. HC and 83.33% AD vs. MCI vs. HC.

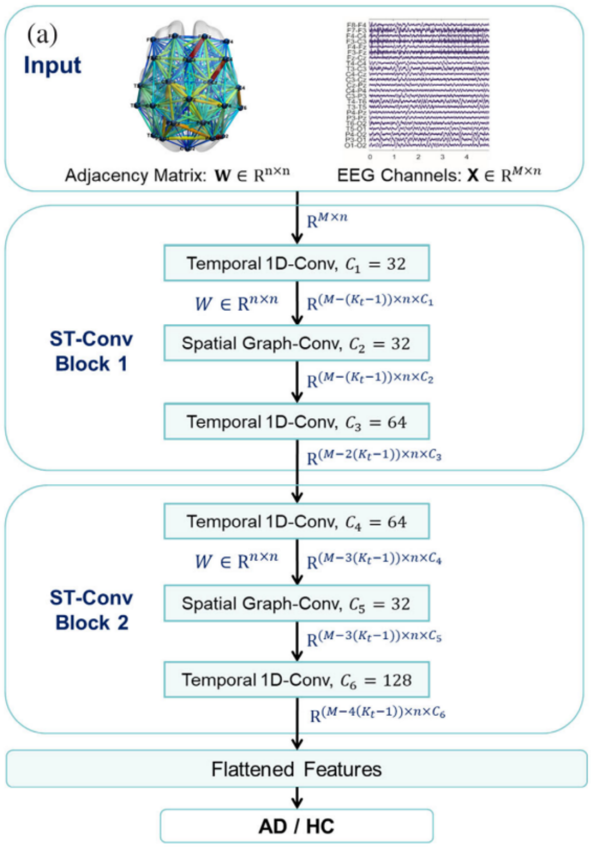
ID	Layer
1	Convolutional layer: 16 filters, 3x3 kernel size
2	ReLU Activation Function
3	Max Pooling: 2x2 kernel size
4	MLP with 3 FC layers: 11376 units, 300 units, 2/3 units + softmax

Table 6: Best CNN architecture in [19]

Later in 2020, the same group proposed an end-to-end CNN to solve the same tasks [21]. The main change with respect to the previous publication is that the CNN is trained to process the raw data (temporal series) of the 5s EEG windows. That means that there is no longer a feature extraction phase to be leveraged by the network. In this case, the CNN architecture is similar to the one described in Table 6, but a second convolutional layer (max pooling and ReLU included) is added. The other difference is the number of filters used per layer: the first one uses 4 filters and the second one uses 8. In this publication, only binary classifications are tested, obtaining the following accuracies over the test dataset: 85.78% AD vs. HC, 69.03% AD vs. MCI, 85.34 MCI% vs. HC.

Very recently, in June 2022, Xiaocai Shan et al. proposed a spatio-temporal GNN model to classify HC and AD subjects [34]. They use a 23 electrode system recording at 100Hz to produce the samples, then they divide the recordings in 0.25 second windows without overlapping. They build a fully connected graph, using the raw temporal samples of each channel as nodes features. For the edge weights, they explore several connectivity measures, but the one producing top results is the Wavelet Coherence (WC).

The model proposed uses spatio-temporal convolutional blocks. These blocks are formed by two different types of layers. The first one is the 1-dimension temporal convolution. This layer learns the weights of a 1-D convolutional filter that processes the raw input data of each channel independently. The second one is the graph spatial convolution, which uses the Chebyshev convolution implementation. Using these two elements, a spatio-temporal convolutional block is defined by stacking them into the following order: temporal convolution, graph spatial convolution and temporal convolution. The final model consists on two of these blocks and a Fully Connected layer acting as the final classifier. The model is visually described in Figure 7.



(a)
Input
Adjacency Matrix: $\mathbf{W} \in \mathbb{R}^{n \times n}$
EEG Channels: $\mathbf{X} \in \mathbb{R}^{M \times n}$
$\mathbb{R}^{M \times n}$
ST-Conv Block 1
Temporal 1D-Conv, $C_1 = 32$
$W \in \mathbb{R}^{n \times n}$
$\mathbb{R}^{(M-(K_t-1)) \times n \times C_1}$
Spatial Graph-Conv, $C_2 = 32$
$\mathbb{R}^{(M-(K_t-1)) \times n \times C_2}$
Temporal 1D-Conv, $C_3 = 64$
$\mathbb{R}^{(M-2(K_t-1)) \times n \times C_3}$
ST-Conv Block 2
Temporal 1D-Conv, $C_4 = 64$
$W \in \mathbb{R}^{n \times n}$
$\mathbb{R}^{(M-3(K_t-1)) \times n \times C_4}$
Spatial Graph-Conv, $C_5 = 32$
$\mathbb{R}^{(M-3(K_t-1)) \times n \times C_5}$
Temporal 1D-Conv, $C_6 = 128$
$\mathbb{R}^{(M-4(K_t-1)) \times n \times C_6}$
Flattened Features
AD / HC

Figure 7: Spatio-temporal GNN for EEG classification. Source: [34]

With the described model and graph configuration, the authors of [34] obtained a 92.3% accuracy in the binary classification of AD vs. HC.

Finally, to sum up this chapter in a few main points:

- There exist GNNs able to perform EEGs classification tasks with high accuracy.
- There exist DL methods to classify AD and MCI with high performance.
- Hand-crafted feature extraction with traditional ML classifiers currently outperform end-to-end DL methods.
- There have not been found GNNs for AD and MCI classification in the literature.

4. Methodology

In this chapter, the method used to conduct the several experiments is presented. First, a description of the dataset available to train and validate the models designed. Then, a description of the several approaches of graph building that have been tested. Next, a description of the models implemented with the corresponding hyperparameter selection. Then, the metrics used for evaluating the models. Finally, the tools and technologies used for developing all the previous steps. All the source code implemented for this work can be found in a GitHub repository [5].

4.1 The Dataset

The dataset available for this work consists of three classes: Healthy Control (HC), Alzheimer's Disease (AD) and Mild Cognitive Impairment (MCI). For every class, there are 63 EEG recordings for each class of subjects. The recording was done following the 10-20 international montage standard, using 19 electrodes recording at 1024Hz. The subjects were recorded during several minutes in resting state, that means that no activity was carried out while recording. The length of each recording can differ by several minutes depending on the subject. A minimal preprocessing had already been applied to the dataset. First, it was downsampled from the original sampling rate to 256Hz. Then, a notch filter at 50Hz was applied to remove the DC component.

The dataset was provided stored in .mat files (the MATLAB binary file extension). These files contained several parameters such as the sampling rate, the recording date and the EEG information. The EEG recording is given in the shape of a matrix of 19 rows (one per each electrode) and $recording\_time \times sampling\_rate$ columns, varying for each subject. Each file was named following the pattern: patient_{id}_{class}.

As proposed in [19, 20, 21] each recording was divided into rectangular and non-overlapping windows of 5 seconds (1280 samples), also referred as epochs. Due to the large variability in the length of the recordings, the dataset obtained after windowing resulted very unbalanced, having more than the double of HC samples than AD samples. The implications of the unbalancing problem would be later discussed, as well as methods to deal with it. Table 7 describes the distribution of the samples:

Class	Number of epochs	Relative Percentage
HC	6461	47.34 %
AD	2756	20.19 %
MCI	4430	32.46 %

Table 7: Dataset classes distribution

After the windowing, each resulting epoch was labeled after its parent recording. That means that if one recording class was AD, all the epochs generated from it would also be labeled as AD. All the epochs were stored as .mat files, to ensure backward compatibility. The naming pattern of the files was respected, but adding an index at the end of the name to differentiate the epochs: patient_{id}_{class}_{epoch_index}. The dataset was divided into three different folders, one per each class. Finally, a .csv file containing the full path of each file and its label was generated to ease the iteration over all the dataset.

All the source code of this project is written in Python. Therefore, to handle the reading and writing operations with MATLAB files, the SciPy [42] IO package was used, which is able to convert the stored

[5] https://github.com/Lokixin/TFM_GNN

data into Numpy [13] arrays. By doing this, the data is kept into its original format and it is compatible with both Python and MATLAB (used to develop the experiments in [19, 20, 21]).

4.2 Building the Graphs

In this section, the procedure to build a graph representation of the data that can be processed by a GNN from the raw recordings is explained. To begin, it must be understood how a graph is built, which implies defining the nodes and its attributes, the edges linking the nodes and the weights (or attributes) for the edges. To do it, Pytorch Geometric (PyG) would be used, which is a library built upon Pytorch framework to be able to develop GNNs and handle graph-like data. In PyG, graphs are defined as objects from the Data class, which requires four components:

1. **Node features:** It is a tensor of shape N rows and number of node features columns. It stores the information of each node within the graph.

2. **Edge index:** It is a $2xN^e$ tensor. The first row contains the index of the origin node, and the second row the index of the destination node. It describes how nodes are linked. For instance: $\begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 4 \end{bmatrix}$ means that there is a bidirectional edge between nodes 1 and 2, and that node 3 is linked to node 4.

3. **Edge attributes:** It is a tensor with shape $N^e \times N^e$. It defines the weights of the edges, the value at i, j defines the weight of the edge connecting node i to node j. It can be 0 if the nodes are not linked.

4. **Labels:** It is a tensor of arbitrary shape. It can contain node-level labels or graph-level labels.

For this work, it is assumed that graphs are fully connected, so the edge index would always be a tensor of 2x361 (19x19 channels). In addition, the target task is graph classification, so the label would always be a scalar $\in [0, 1, 2]$ which are the labels for the 3 classes. This reduces the problem to extracting features for representing the nodes and computing weights for the edges that represent the relationship between nodes information. Figure 8 exemplifies the procedure of building a graph from an EEG recording. Next, the different approaches for computing this tensors are explained.

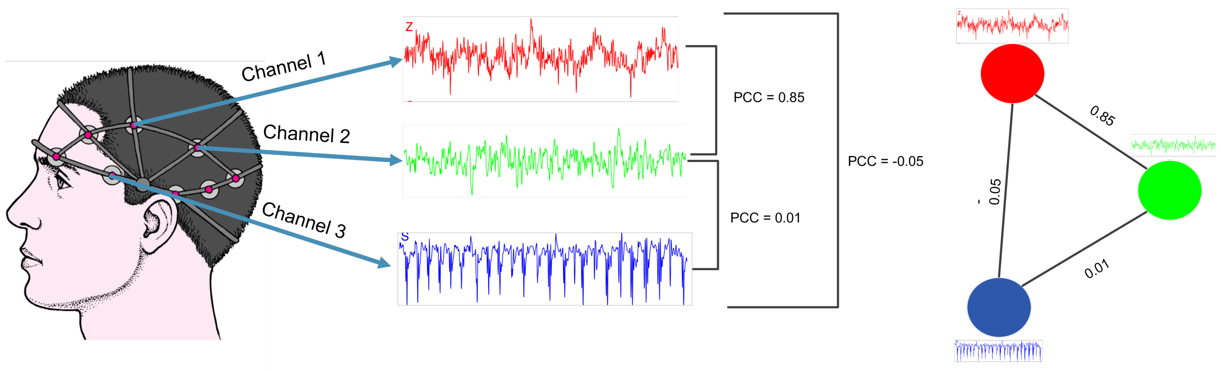


Figure 8: Graph representation from EEG recording

4.2.1 Node features computation

The main idea is to represent every physical electrode used in the EEG recording with a node in the graph. However, there are several ways to achieve that objective. In the state of the art chapter, several methods to extract relevant features to describe the nodes of the graph were discussed. In this work, three main approaches have been implemented and tested. These are:

- **Raw data:** Each node contains the time signal produced by one electrode. Then, the features of each node are a vector of 1280 samples (5s recordings at 256Hz). Then, all of the vectors are vertically stacked to produce a tensor of shape $[19, 1280]$. This approach is used to let the network perform the feature extraction phase and learn meaningful node embeddings. Moreover, it can be used to exploit the temporal dependencies which are lost in other approaches. For this approach, the only operation needed is to transform the EEG matrix into a tensor, so PyG modules can process it. In addition, data can be optionally normalized using L1 norm, to make the model focus on signal variations instead of absolute peaks.
- **Statistical moments:** In this approach, each node is represented by the mean, standard deviation, variance, entropy, skewness and kurtosis of the temporal signal. Consequently, each node contains the moments computed for a given channel of the EEG, resulting in a tensor of shape $[19, 6]$. Using the numpy package, this metrics can be computed for each channel at the same time, leading to fast computations which allows to build online datasets. This approach is more intended to validate GNN models, and perform experiments in a fast and easy way than to achieve high accuracy.
- **Power Spectral Density:** Each node is represented by the average PSD computed as described in (10) in six frequency bands. These bands are δ [1, 4]Hz, θ [4, 7.5]Hz, α [7.5, 13]Hz, β_{low} [13, 16]Hz, β_{high} [16, 30]Hz and γ [30, 40]Hz as it was proposed in [43]. As a result, each node contained 6 features, each one of them representing the average power within one of these bands. Finally, the dimensions of the node feature tensor are $[19, 6]$.

Notice that all of the aforementioned methods work at channel level. It means that the features extracted for one channel depend only on the temporal signal of the same channel. Nevertheless, one of the main advantages of using a GNN is the capability of exploiting the connections between nodes. For this purpose, the following edge attributes computation methods have been implemented.

4.2.2 Edge attributes computation

The main function of the edge attributes is to describe how strong is the connection between nodes. In other words, how much is the state of $Node_i$ affected by the current state of $Node_j$. As fully connected graphs are assumed because there is not a clear rule for connecting nodes in an EEG, the relationship between each possible pair of nodes is computed (including self loops). As a result, for all the following metrics, a $[19, 19]$ tensor is obtained. The methods proposed are:

- **PCC:** It is computed for each pair of channels as described in Equation (10). This correlation measure aims to find temporal dependencies between the channel's signals. There are approaches that use the absolute PCC bounded between $[0, 1]$, however the implementation in this work does not, so the resulting matrix is bounded within the $[-1, 1]$ interval.
- **Average Spectral Coherence:** The weight of the edges are represented by the average spectral coherence between two channels, Equation (11). As there are several examples in the literature that

remark the importance of the frequency domain in EEG classification tasks, this measure quantifies how similar two spectra are. Optionally, all the weights computed can be normalized to the interval $[0, 1]$, because the average spectral coherence is not upper bounded and may produce large weights.

- **Phase Lag Index (PLI):** Unlike PCC that focuses in the amplitude of the signals to compute their relationship, the PLI focuses on the phase difference. The phase difference seems to be a more robust approach, because the amplitude of a channel's signal can be contaminated by neighbouring electrodes, causing fake correlations. The PLI between two channels can be computed as follows [39]:

$$PLI_{ab} = \frac{1}{N} \sum_{t=1}^{N} exp(j\Delta\Phi_{ab}(t)) \tag{19}$$

where:

$\Delta\Phi_{ab}$ = Phase (computed by Hilbert transform) difference between signals a and b.

- **Unweighted:** Graphs can also be unweighted. This approach is useful when attention graph convolutions are implemented. In this scenario, the network learns to compute the weights so there is no need to apply neither of the previous approaches.

For every of the aforementioned methods, there is the option to apply a threshold over the weights. That means that if any weights is lower than a given threshold, it would be set to zero. In practice, this is the same as removing a link between two nodes, because a link with a weight equal to zero, would imply that the node features would not be taken into account during the message passing step.

4.3 The Models

Once the graph building is prepared, it is time to design the models to process them and perform the classification task. The first step to begin with this process was to replicate one of the publications, in particular the one proposed in [43], which is described in Table 4. To do so, their proposed model was implemented and trained with the TUEG dataset, that is available under request [6]. The authors obtained a 0.90 AUC, and we obtained a 0.85 AUC with less training epochs and without tuning hyperparameters. Hence, it was considered a valid justification for the capabilities of GNNs to classify EEG data.

In addition to the replicated model, several more were designed, implemented and tested. Notice that not all the models are designed to deal with all types of graphs, some of them are more oriented to process raw signals and others are designed to deal, for example, with lower dimensional features such as the PSD or the statistical moments. A description of the model's architectures is provided below:

[6]Dataset source: `https://isip.piconepress.com/projects/tuh_eeg/html/downloads.shtml`

Model	Architecture
EEGGraphConvNet	GCNConv(in=1280, out=640) + LeakyReLU + BatchNorm GCNConv(in=640, out=512) + LeakyReLU + BatchNorm GCNConv(in=512, out=256) + LeakyReLU + BatchNorm GCNConv(in=256, out=256) + LeakyReLU + BatchNorm GlobalAddPooling MLP(FC(256, 128), FC(128, 64), FC(64, 2)) + Softmax
EEGGraphConvNetLSTM	LSTM(in_features=1280, out_features=hidden_dim, gates=n_gates) GCNConv(in=hidden_dim, out=320) + LeakyReLU + BatchNorm GCNConv(in=320, out=180) + LeakyReLU + BatchNorm GCNConv(in=180, out=90) + LeakyReLU + BatchNorm GCNConv(in=90, out=50) + LeakyReLU + BatchNorm GlobalAddPooling MLP(FC(50, 32), FC(32, 16), FC(16, 2)) + Softmax
EEGGraphConvNetMini	GCNConv(in=6, out=16) + LeakyReLU + BatchNorm GCNConv(in=16, out=32) + LeakyReLU + BatchNorm GlobalAddPooling MLP(FC(64, 32), FC(32, 16), FC(16, 2)) + Softmax
EEGGraphConvNetAttention	GATConv(in=6, out=12) + LeakyReLU + BatchNorm GATConv(in=12, out=32) + LeakyReLU + BatchNorm GATConv(in=32, out=64) + LeakyReLU + BatchNorm GlobalAddPooling MLP(FC(64, 32), FC(32, 16), FC(16, 2)) + Softmax

Table 8: GNN Architectures evaluated in this work

The in and out arguments used in the convolutional layers refer to the number of features per node. If there is a variable instead of a predefined value, it means that it can be defined when the model is created. The models would be evaluated and discussed in the result section, however following there are several comments on the models described above:

- **EEGGraphConvNet:** This model follows the same structure as the replicated model [43], but incorporates some changes. First, a batch normalization layer is added after each convolution, the original model only had one batch normalization layer after all the convolutions. The second main change is the number of input and output features. This model is intended to work with temporal signal without any preprocessing, so that the input dimension of the first convolution must match with the samples per window of the EEG recordings. As the input dimension is quite large, the model tries to compress the information by reducing the number of features, instead of augmenting them.
- **EEGGraphConvNetLSTM:** This model aims to capture temporal relationships first and then spatial relationships. To do so, a first layer with N LSTM gates is added, which would try to learn temporal patterns. These gates process each channel individually, each temporal signal is treated as an independent input from the LSTM point of view. Next, the embeddings produced by the LSTM gates are fed into the set of graph convolutional layers. These layers combine the information of every layer, and would try to identify spatial relationships between channels.

- **EEGGraphConvNetMini:** This model is a smaller version of the EEGGraphConvNet. It removes the last convolutional layer in order to avoid the oversmoothing effect. Moreover, the number of input features is much smaller, and the convolutional layers expand the amount of features per node. This model is designed to deal with graphs which have already been through a feature extraction phase, such as the PSD or the statistical moments.

- **EEGGraphConvNetAttention:** It is a version of the previous model which changes the GCN layers to GAT layers so the network can learn the graph's weights. In this model, many versions of GAT convolutions are tested to identify which implementation works best for our task. These are discussed in the results chapter.

4.4 Model Training

4.4.1 Dataset Generation

To generate the datasets a custom Pytorch dataset was implemented. The dataset iterates through the .csv file containing the paths of all the EEG recordings available and loading the corresponding file. However, as explained in previous sections, there are several ways to build the graph. For this purpose, two functionalities were implemented. The first, is to load the raw recordings and perform the desired transformation (such as computing the PCC or the PSD) online. However, this process can slow the training because some operations, like computing the PLI, can take a very long time.

To deal with this problem, an offline dataset generation functionality was added. This allows to compute any nodes / edges features for all the subjects' recordings and store them as .mat files following the exact same structure as the original one. Using this option, the dataset can be fast and easily built by selecting two folders: one with the precomputed node features and another with the precomputed edge weights, and they can be combined in any way.

Once the dataset is built, it is divided in three different splits: train, test and validation. The train dataset contains the 70% of the windows, the test dataset a 20% and the validation dataset the remaining 10%. However, due to the unbalancing problem shown in Table 7, only the 60% of the HC samples are used, to avoid over-representing that class. The train dataset is the one used to train the network, so the network's weights are updated as a function of the classification error of this dataset. The test dataset would be used to monitor the performance of the model, but only used in the forward stage, no backpropagation is done with it. Its main function is to detect overfitting if it appears. Finally, the validation set would be used to check that the model is capable to generalize well with unseen data.

After the dataset division, the Dataloaders are implemented. Using a Dataset class allows to load one sample per iteration, but for training the models it is needed to load data in batches of N samples. To achieve that, the PyG Dataloaders are used, which can load batches of a given size of unrelated graphs. Moreover, it gives the option to shuffle the data, which results really handy to ensure that at every batch there are samples from all the different classes. At this point, everything is ready to start forwarding the dataset through the designed models. In the next section, the training configuration including optimizers, loss functions, schedulers and hyperparameters choices is explained.

4.4.2 Training Configuration

The model is trained using a batch of 64 to 128 samples of unrelated graphs. For each batch, the samples are forwarded through the network, which returns the probabilities of a given samples to belong to each one of the possible classes. This probabilities are compared with the ground truth using a defined criterion to compute the error during the training stage. The criterion used for this purpose is the Cross Entropy Loss. This loss is designed for classification problems, it computes the cross entropy between the input (the model's prediction) and the target (the label or ground truth). It is defined as follows:

$$l(x, y) = L = \{l_1, l_2, ..., l_N\}^T, l_n = -\sum_{c=1}^{C} w_c log \frac{exp(x_{n,c})}{\sum_{i=1}^{C} exp(x_{n,i})} y_{n,c} \tag{20}$$

where:

C = number of classes
N = Number of samples in the batch
x = model output
y = target value
w = optional weight for each class

Once the loss is computed, it must be backpropagated through the network, so the error related to each weight is computed. Once this task is finished, an optimization step can start, which is the responsible of updating the weights of the model to minimize the loss function. To do so, the Adam optimizer [25] is employed, which shows to be more robust and converge to better results in this scenario than other methods such as Stochastic Gradient Descent (SGD). The optimizer is initialized with a 0.1 learning rate.

To control the learning rate during the training, an optimizer scheduler is used. In this case, the ReduceLROnPlateau from Pytorch. This scheduler tracks the progress of one metric, in our case the training loss, and reduces the learning rate when the metric being tracked gets stuck during several epochs. There are many parameters that can be configured in the scheduler, the main ones are the following:

- **Patience:** The amount of epochs that the scheduler waits before reducing the learning rate if the metric being tracked does not improve. For training the models 5 epochs of patience were used.
- **Threshold:** The threshold defines the minimum margin of improvement needed. If the metric improves, but less than the threshold, the scheduler would be triggered anyways. A 0.05 threshold was used over the loss decrease.
- **Cooldown:** The amount of epochs to wait before start counting patience epochs again after the learning rate has been reduced. 2 cooldown epochs were given.
- **Factor:** The factor of learning rate reduction after the patience epochs have been completed.
- **Minimum learning rate:** A lower bound for the learning rate that would not be farther reduced event though the metric does not improve.

All the models are trained up to 100 epochs, however the process can be stopped earlier if no progress is observed despite decreasing the learning rate.

4.5 Metrics for evaluating models performance

There exist a wide range of metrics that can be used to assess the performance of the model. The most commonly used is the accuracy, which represents the percentage of samples classified correctly over the total number of samples. Despite of the fact that accuracy guarantees a global overview of the model behavior, in medical terms, this metric lacks of specificity. A clear example is that diagnosing a disease when the subject is healthy, is much less severe than considering the subject healthy when in actually has the disease. However, when computing the accuracy both types of errors weight the same. In addition, the accuracy metric does not consider if a class is over-represented or infra-represented, neither what classes are causing more confusion to the model. To solve this inconveniences and to achieve a more detailed evaluation of the model, the following metrics are proposed:

For the following definitions this nomenclature is followed: TP stands for True Positive, it means that the subject has a disease and it is correctly diagnosed. TN means True Negative, it happens when the subject is healthy and the system classifies it as healthy. FP is False Positive, when the subject is actually healthy but is diagnosed with a disease. And finally, FN means False Negative, is the case when the subject has a disease but it is not diagnosed.

- **Accuracy:** It is the hit rate of the model. Defined as:

$$Accuracy = \frac{\#hits}{total\_samples} \tag{21}$$

- **Balanced accuracy:** This version of the accuracy computes the weighted per class accuracy. If the model is unbalanced, the accuracy may be high but the balanced accuracy would be significantly lower. It is computed as:

$$BalancedAccuracy = \frac{\frac{TP}{TP+FN} + \frac{TN}{TN+FP}}{2} \tag{22}$$

  grouping the summands in the numerator, it also can be defined as:

$$BalancedAccuracy = \frac{Sensitivity + Specificity}{2} \tag{23}$$

- **Precision:** It computes the ratio between the TP and the total positive predictions.

$$Precision = \frac{TP}{TP + FP} \tag{24}$$

- **Recall:** It is a very representative measure because it gives the ratio between the correct diagnoses and the missed diagnoses.

$$Recall = \frac{TP}{TP + FN} \tag{25}$$

- **F1-Score:** It summarized the overall model performance by averaging the precision and recall values.

$$F1Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{26}$$

- **The Area Under the Receiver Operator Curve (AUC):** This metric averages the precision obtained at several values of recall. It is bounded between [0.5, 1], the closer to 1 the better the model is working. It is computed as the area under the Precision vs. Recall curve.

5. Results

This chapter is dedicated to evaluate the performance of the models based on the metrics defined in Section 4.5. The models would be evaluated together with the different approaches of graph building. Hence, every experiment would be carried out with one model, one of the possible node features and one of the edge weight computation options. These combinations would be evaluated for the binary classification task of AD vs. HC. Then, the ones showing the best performance would be evaluated in the classification of AD vs. HC vs. MCI classification.

It is important to notice that not all the possible combinations of model, edge and nodes would be tested. Some models are designed to specifically process a concrete type of node features. Furthermore, only for the best models, the effects of changing some parameters such as the number of filters of convolutional layers, the type of attention, or the pooling methodology would be evaluated.

Even though the models are trained up to 100 epochs, the state of the model which gives the higher test accuracy during the training is saved. Then, these weights are the ones used to evaluate the model with the train, test and validation dataset and obtain the final metrics. Tables 9 and 10 show all the results for all the experiments performed.

Model	**Nodes**	**Edges**				
Replicated	PSD	Spectral Coherence				
Set	**Acc.(%)**	**Bal.Acc.(%)**	**Precision(%)**	**Recall(%)**	**F1(%)**	**AUC**
Train	45.67	50.02	72.83	50.02	31.39	0.74
Test	46.97	50.00	23.48	50.00	31.96	0.75
Val	48.00	50.00	24.00	50.00	32.43	0.73
Model	**Nodes**	**Edges**				
GraphConvNet	Raw	PCC				
Set	**Acc.(%)**	**Bal.Acc.(%)**	**Precision(%)**	**Recall(%)**	**F1(%)**	**AUC**
Train	99.09	99.11	99.05	99.11	99.08	0.99
Test	69.34	69.36	69.34	69.36	69.33	0.74
Val	70.93	70.99	70.89	70.99	70.88	0.80
Model	**Nodes**	**Edges**				
GraphConvNet	Raw	PLI				
Set	**Acc.(%)**	**Bal.Acc.(%)**	**Precision(%)**	**Recall(%)**	**F1(%)**	**AUC**
Train	96.17	96.41	96.20	96.41	96.17	0.97
Test	73.17	73.81	73.43	73.81	73.11	0.81
Val	75.20	75.68	75.87	75.68	75.19	0.83
Model	**Nodes**	**Edges**				
GraphConvNet	Raw	SC				
Set	**Acc.(%)**	**Bal.Acc.(%)**	**Precision(%)**	**Recall(%)**	**F1(%)**	**AUC**
Train	75.14	75.82	76.00	75.82	75.13	0.80
Test	75.49	75.88	76.26	75.88	75.45	0.80
Val	73.33	74.16	74.02	74.16	73.33	0.79
Model	**Nodes**	**Edges**				
GraphConvNetLSTM	Raw	PCC				
Set	**Acc.(%)**	**Bal.Acc.(%)**	**Precision(%)**	**Recall(%)**	**F1(%)**	**AUC**
Train	90.84	90.65	90.92	90.65	90.76	0.97
Test	65.42	65.04	65.15	65.04	65.07	0.70
Val	59.20	59.15	59.13	59.15	59.13	0.65
Model	**Nodes**	**Edges**				
GraphConvNetMini	Moments	PCC				
Set	**Acc.(%)**	**Bal.Acc.(%)**	**Precision(%)**	**Recall(%)**	**F1(%)**	**AUC**
Train	73.71	73.47	73.58	73.47	73.51	0.81
Test	73.35	72.86	73.19	72.86	72.96	0.81
Val	71.73	71.16	71.40	71.16	71.24	0.80
Model	**Nodes**	**Edges**				
GraphConvNetMini	PSD	PCC				
Set	**Acc.(%)**	**Bal.Acc.(%)**	**Precision(%)**	**Recall(%)**	**F1(%)**	**AUC**
Train	86.39	86.26	86.32	86.26	86.29	0.94
Test	85.47	85.38	85.43	85.38	85.40	0.93
Val	84.80	84.59	84.63	84.59	84.61	0.93

Table 9: Results of models with different combinations of node and edge features in AD vs. HC classification

Model	**Nodes**	**Edges**				
GraphConvNetMini	PSD	PLI				
Set	**Acc.(%)**	**Bal.Acc.(%)**	**Precision(%)**	**Recall(%)**	**F1(%)**	**AUC**
Train	86.04	86.03	86.03	86.03	86.03	0.94
Test	87.01	87.02	87.01	87.02	87.01	0.94
Val	83.53	83.36	83.21	83.36	83.28	0.90
Model	**Nodes**	**Edges**				
GraphConvNetMini	PSD	SC				
Set	**Acc.(%)**	**Bal.Acc.(%)**	**Precision(%)**	**Recall(%)**	**F1(%)**	**AUC**
Train	89.41	89.45	89.42	89.45	89.41	0.97
Test	86.61	86.67	86.56	86.67	86.59	0.94
Val	82.93	82.93	83.08	82.93	82.92	0.92
Model	**Nodes**	**Edges**				
GraphConvNetAttention	PSD	Attention				
Set	**Acc.(%)**	**Bal.Acc.(%)**	**Precision(%)**	**Recall(%)**	**F1(%)**	**AUC**
Train	96.99	96.89	96.73	96.89	96.81	0.99
Test	91.57	91.29	90.91	91.29	91.09	0.97
Val	92.31	91.80	91.90	91.80	91.85	0.97

Table 10: Results of models with different combinations of node and edge features in AD vs. HC classification (continued)

For the AD vs. HC vs. MCI classification task only the best model configuration was evaluated, the GraphConvNetAttention. The model architecture remains the same, but it is retrained using data from the three classes and using a lower initial learning rate (0.001 instead of 0.1). The obtained results are shown in Table 11.

Set	**Acc.(%)**	**Bal.Acc.(%)**	**Precision(%)**	**Recall(%)**	**F1(%)**	**AUC**
Train	97.50	97.36	97.29	97.36	97.32	1.00
Test	87.68	86.64	87.07	86.64	86.84	0.96
Val	87.59	86.69	86.60	86.69	86.63	0.97

Table 11: Results GraphConvNetAttention for AD vs. HC vs. MCI classification

5.1 Discussion of results

5.1.1 Proposed Models Comparison

The top results of all the proposed models are summarized in Figure 9, where the accuracy, F1-score and AUC are given to compare the their general performance. Only the results of the best combinations of node and edge features are shown.

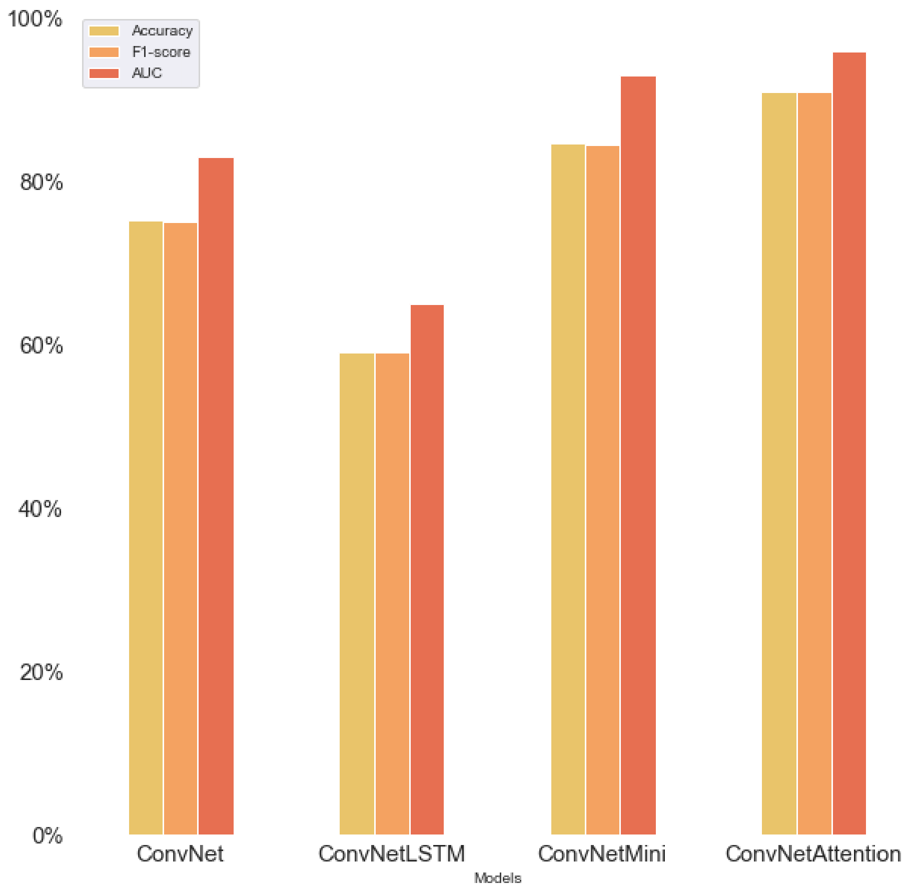


Figure 9: Results comparison between top performing models

It can be clearly observed that the worst model is the ConvNetLSTM, which achieves a 30% lower accuracy than the best one, the ConvNetAttention. Another direct conclusion is that the models using hand-crafted features for the nodes, outperform the ones using the raw temporal signals. This is because the network receives relevant features, instead of having to learn a meaningful representation from the raw data, so the job is easier. However, the different methods used to compute the edges (PLI, SC, PCC) do not seem to have a major impact over the models performance. In this case, letting the network learn the edges weights produces notorious better results. In addition, it can be seen that the accuracy and the F1-score bars are almost identical, which means that the model is balanced and it is not over-represented by any class.

5.1.2 Models training

In addition to the final results, it is important to discuss the behaviour of the models while training. All the models were trained using an Intel i7 10750H CPU and 16GB of RAM, following the procedure described in the methodology section. In addition, all features were precomputed offline, so the differences in training time are only due to the model architecture, and not related to the feature extraction methods. Note that all the computing times could be reduced with proper acceleration techniques [1, 45, 10] and not pose a roadblock in the EEG analysis in general and the present use case in particular. Next, the evolution of loss and accuracy curves during training is shown for all of the proposed models.

The GraphConvNet model took 4 hours to complete the 100 epochs of training, as shown in Figure 10. At epoch 25, it started to show obvious signs of overfitting.

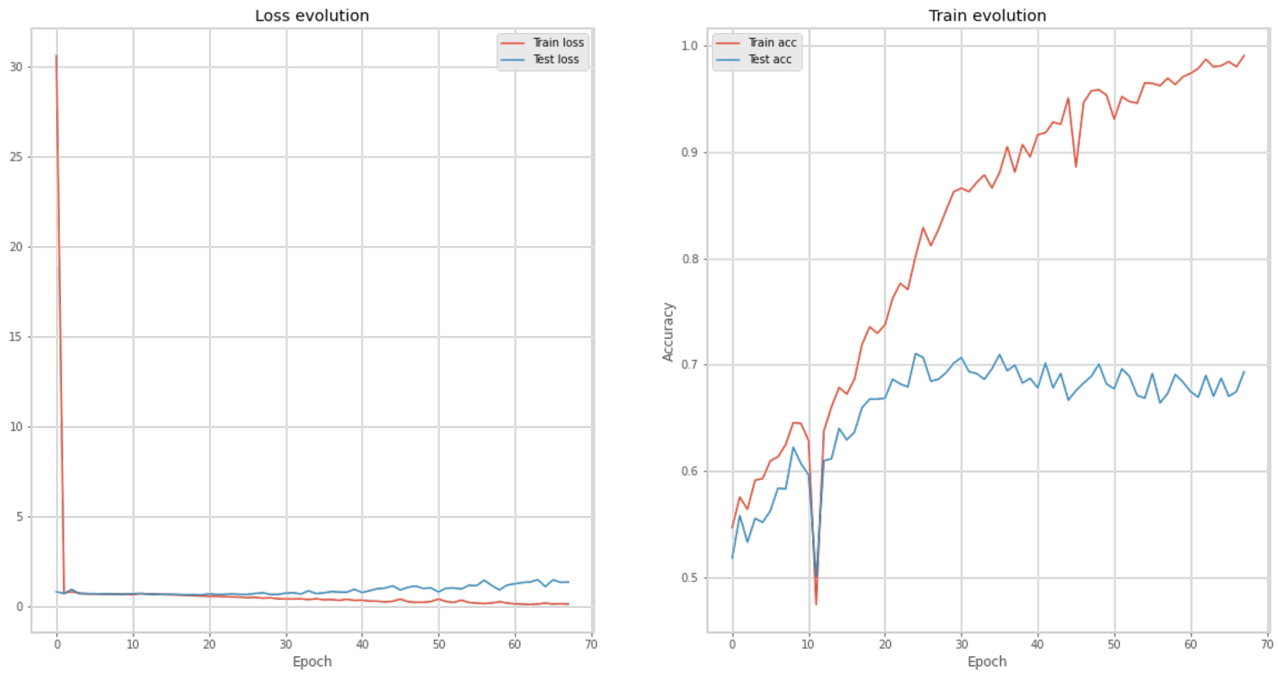


Figure 10: GraphConvNet training evolution.

The GraphConvNetLSTM model took 7 hours to complete 25 epochs of training, where it was stopped because of overfitting and lack of progress. As it is expected from a recurrent model, it was the one that took longer to train by far. These results are shown in Figure 11.

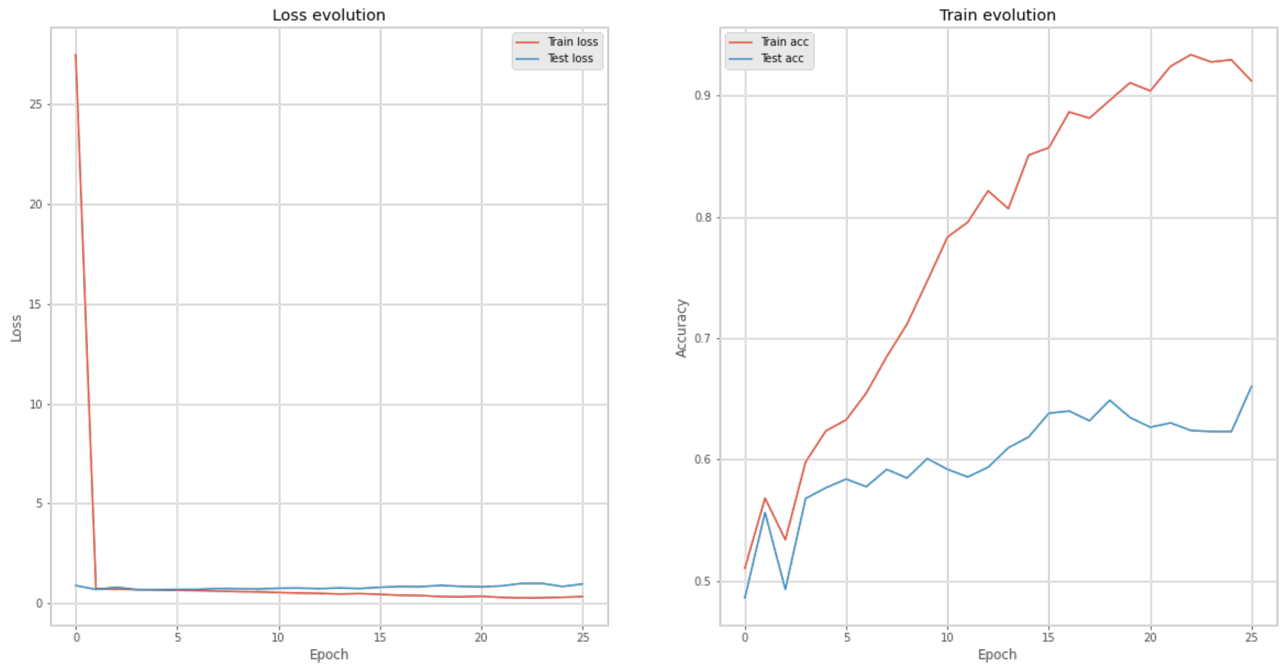


Figure 11: GraphConvNetLSTM training evolution.

Models that process temporal data directly need to perform larger convolutions due to the features dimensionality. As it can be expected, they need more time to be trained because the operations are more expensive. On the contrary, the following models process features with lower dimensionality and have less convolutional layers, so they are remarkably faster.

The GraphConvNetMini architecture only has two convolutional layers. It took only 48 minutes to converge. As the model has less parameters, it is not as prone to overfit as the temporal models. Moreover, it reaches better results as shown in Figure 12.

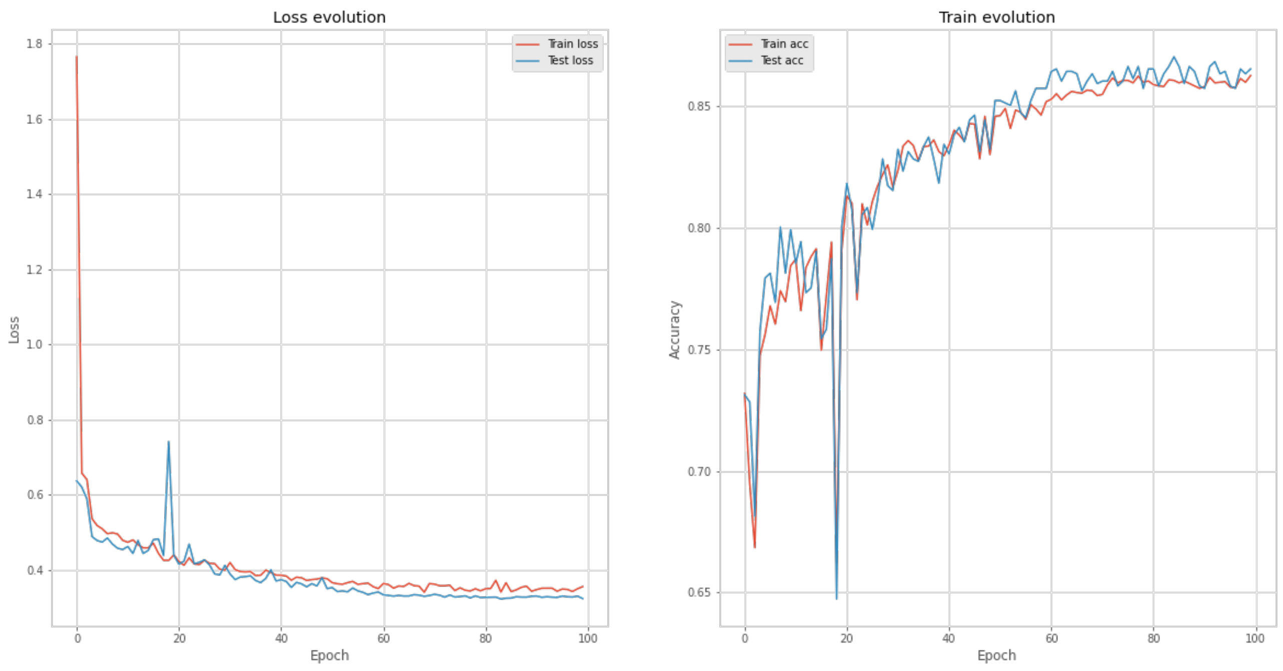


Figure 12: GraphConvNetMini training evolution.

Finally, the GraphConvNetAttention architecture took between 2 and 3 hours to train, depending on the number of heads and the attention implementation. Figure 13 illustrates the evolution of the training.

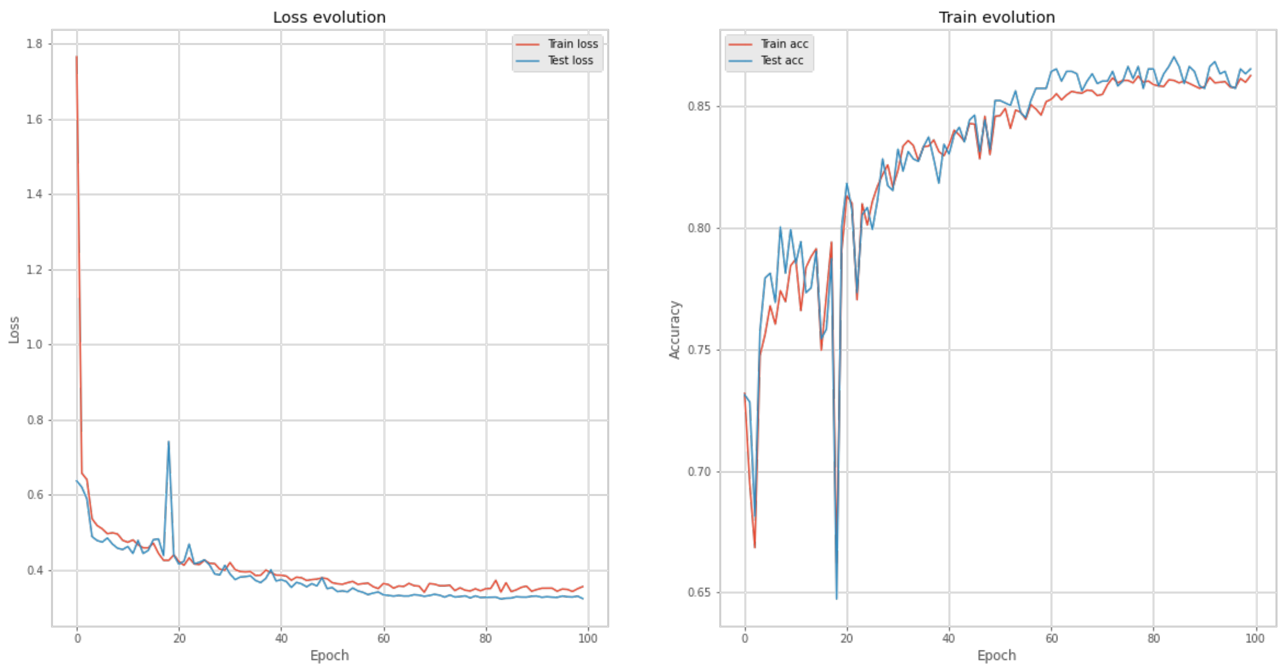


Figure 13: GraphConvNetMini training evolution.

5.1.3 Best model configuration

From Tables 9 and 10 it can be concluded that the best overall model is GraphConvNetAttention. This model receives as input a fully connected unweighted graph (weights are learnt by the network using attention), where nodes are the average PSD at 6 frequency bands. The model general architecture is described in Table 8. The graph attention convolution layers that gave the best performance were the TransformerConv, defined in [36]. For every TransformerConv layer, two attention heads were defined. It is important to remark that the results obtained with this model only differ around 1-2% between the train, test and validation splits. Hence, it is proven that the GraphConvNetAttention is capable of generalizing with unseen data. Next, Table 12 shows a comparison of performance depending on the type of convolutional layer used:

Attn. Layer	Acc.(%)	Bal.Acc.(%)	Precision(%)	Recall(%)	F1(%)	AUC
GAT	86.13	86.27	86.23	86.27	86.13	0.95
GATv2	89.01	88.51	87.94	88.51	88.20	0.97
TransformerConv	92.31	91.80	91.90	91.80	91.85	0.97
SuperGATConv	91.02	90.96	91.14	90.96	91.00	0.96

Table 12: Comparison between convolutional attention layers

All of the aforementioned attention layers rely on attention heads to find different patterns and relationships between nodes features. The number of features is proportional to the number of heads. Table 12 shows a small study about the impact of the number of heads while using the TransformerConv convolutional layer.

Number of heads	Acc.(%)	Bal.Acc.(%)	Precision(%)	Recall(%)	F1(%)	AUC
1	83.20	83.09	82.88	83.09	82.97	0.90
2	92.31	91.80	91.90	91.80	91.85	0.97
3	87.03	86.74	85.84	86.74	86.23	0.95

Table 13: Comparison between number of attention heads

An increment in the number of heads directly implies a larger number of trainable parameters in the network. This is because each head contains a FC layer to learn different projections. However, if the dataset is not large enough, or the number of node features is small, using a larger number of heads may not produce a positive impact on the network's performance.

5.2 Comparison with State of Art Methods

In this section, the results obtained in this work are compared with state-of-the-art methods from [19], [20], [21] and [34] labeled as MLP, CNN-2018, CNN-2020, and ST-Conv here, respectively. They are compared in terms of accuracy and F1-score obtained with the validation datasets. First, the performance on the binary classification task of AD vs. HC is compared in Figure 14.

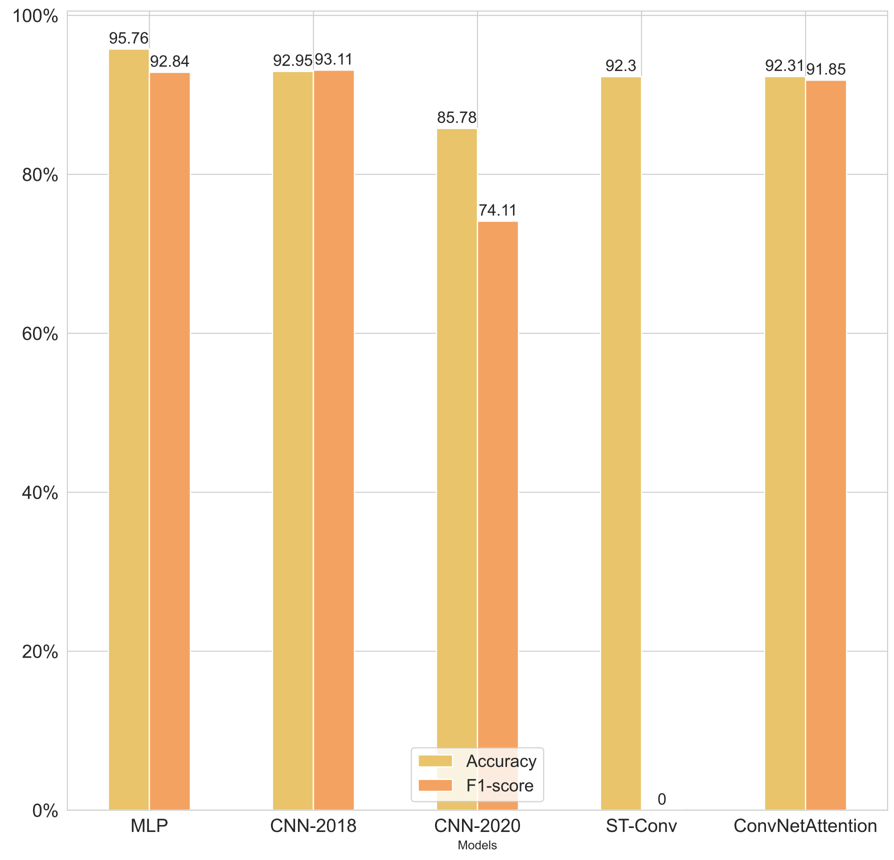


Figure 14: Comparison of the proposed work (ConvNetAttention) with respect to the state of art methods [19, 20, 21, 34] for the binary classification case.

Notice that despite using the same dataset (MLP, CNN-2018, CNN-2020), the splits may change because of random shuffling when producing the train, test and validation splits. Moreover, the ST-Conv model was trained with a different dataset recorded with a different EEG montage. With that being said, the MLP model which uses CWT and BiS features is still the best method. The CNN-2018 uses the PSD of the complete spectrum obtains slightly better results than our proposed model, which uses the PSD of six frequency bands. The model with worst performance is the CNN-2020, but it uses the raw data without any feature extraction previous to the CNN. The ST-Conv model authors do not provide any metric but accuracy, so the F1-Score can not be compared.

The following graphic shows a comparison of the proposed model for the AD vs. MCI vs. HC classification with the state of art models described in [19, 20, 21].

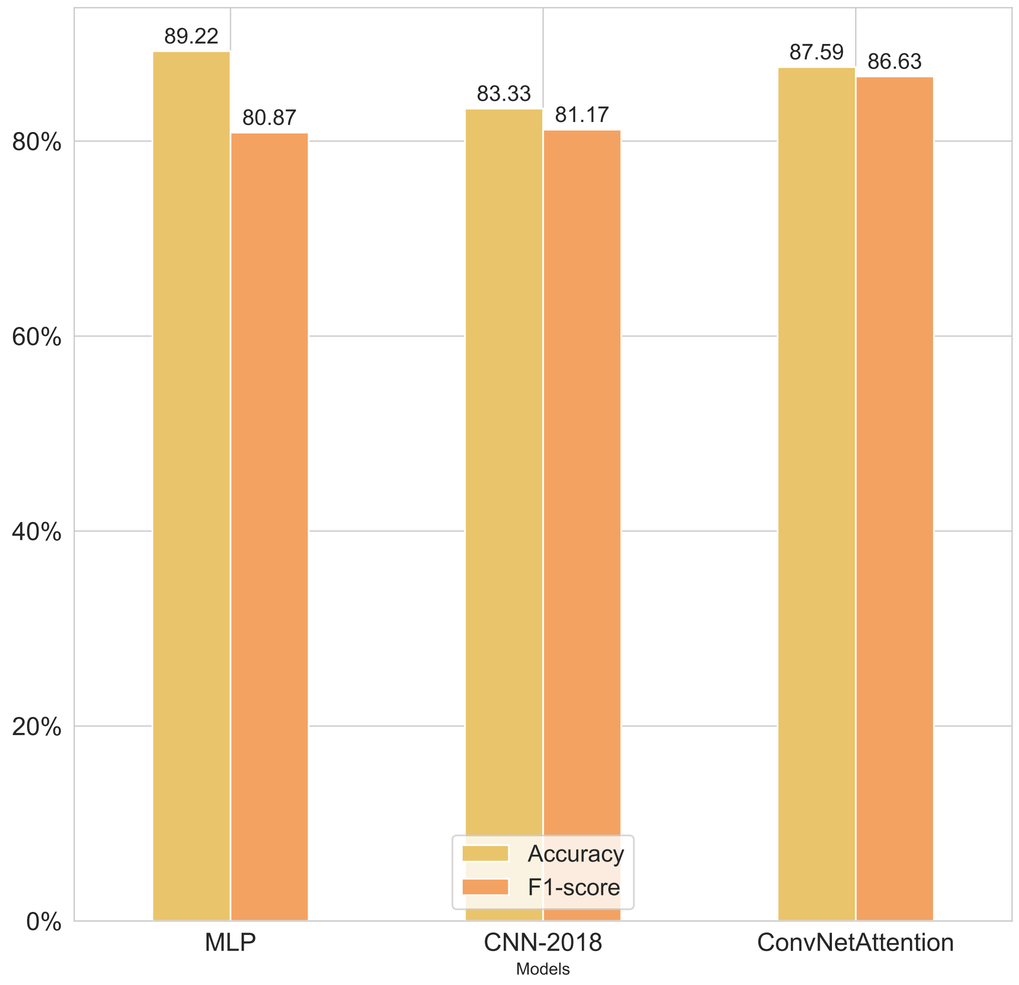


Figure 15: Comparison of the proposed work (ConvNetAttention) with respect to the state of art methods [19, 20, 21] for the AD vs. MCI vs. HC case.

The proposed GNN improved the results obtained by the CNN-2018, which also used PSD features, by a 4%. Compared to the MLP, the accuracy is 2% lower but the F-Score is 6% higher, which indicates that the proposed model may provide a more balanced solution. The CNN-2020 model proposed by [21] and the STConv model by [34] were only used for binary classification tasks, so in these cases such techniques can not be added to the comparison.

6. Conclusions

This master thesis has explored different combinations of graph building and GNNs models for classifying EEGs from AD and MCI patients. It has been demonstrated that GNNs are suitable architectures for detecting neurological diseases from EEG recordings, without the need of extra information about the patient. From the exhaustive comparison of models and feature extraction methods, many interesting conclusions can be stated.

First, the combination of hand-engineered feature extraction methods with neural network models used as classifiers outperforms end-to-end learning methods. This phenomenon can be detected also in many state of the art publications. It is difficult to directly process the raw temporal signals captured during the recordings and achieve acceptable results. The main reason behind this is that there exist prior knowledge on how neurological diseases affect to the brain activity, so features that better represent these alterations would ease the job of the network. Despite neural networks can learn to extract features by their own, they do not possess high-level human knowledge about particular topics such as the Alzheimer's disease. A clear example is the use of the PLI instead of the PCC: researchers know that relationships between channels amplitudes may be fake because of the interference of neighbouring electrodes. However, they know that the phase information of the electrical signal is more reliable. As a result, the PLI, which uses phases information, is preferred over the PCC, that relies on amplitude measure instead. This is a very difficult deduction for a neural network to make. However, if the model is trained with these features, it would benefit from the prior human knowledge about the task.

Following with the feature extraction topic, it can be concluded that information related to the frequency domain is very relevant for detecting neurological diseases. The best model proposed in this work uses the PSD to compute the nodes features, whereas the best overall model in the literature uses CWT and BiS features, all of them coming from the frequency domain. Nevertheless, this does not happen with the edges. Graph attention mechanisms capable of learning relations between nodes have showed to be superior to all the manual edge weight computation methods proposed. This finding is really meaningful, since attention methods can signify a huge step in the field of GNNs explainability. The capability of the network to learn the edges weights can help to understand how signals from different channels relate to each other, and which are the most important features across channels.

Finally, despite getting really close, all results from the literature could not be outperformed. However, a baseline of GNN architectures, node feature extraction methods, edge weight computation, model training, and metrics extraction is provided. This can be the foundation of more optimized models that overpass the current results, not only in AD and MCI classification, also for other neurological diseases.

6.1 Future Work

Due to the wide array of options to be evaluated, there are many topics related to this work that remain to be tested. The first one would be to optimize the current models. This step would include a cross-optimization between the type of attention layers, the global pooling methodologies, and the number of input/output features per layer. In addition, a more in-depth study of training hyperparameters such as the learning rate or the batch size would be beneficial. Following this point, searching better models for the three way AD vs. HC vs. MCI classification would be the next step. This work was initially focused to solve the AD vs. MCI task, and due to lack of time a better model for the three classes task could not be designed.

Another field with huge potential is that of spatio-temporal graphs. Nowadays, many state-of-the-art graph convolutional layers are developed to find not only spatial, but also temporal relationships in graphs. This would be a really interesting topic of research because EEGs are intrinsically temporal signals. To exploit the temporal dependencies, it would be great to explore high density datasets, using higher sampling rates and larger number of electrodes, so the time series are better represented.

7. Bibliography

References

[1] Sergi Abadal, Akshay Jain, Robert Guirado, Jorge López-Alonso, and Eduard Alarcón. Computing graph neural networks: A survey from algorithms to accelerators. *ACM Computing Surveys (CSUR)*, 54(9):1–38, 2021.

[2] Alzheimer's Association. 2022 alzheimer's disease facts and figures.

[3] Nils Bjorck, Carla P Gomes, Bart Selman, and Kilian Q Weinberger. Understanding batch normalization. volume 31. Curran Associates, Inc., 2018.

[4] 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, 7 2017.

[5] Qi Chang, Cancheng Li, Qing Tian, Qijing Bo, Jicong Zhang, Yanbing Xiong, and Chuanyue Wang. Classification of first-episode schizophrenia, chronic schizophrenia and healthy control based on brain network of mismatch negativity by graph neural network. *IEEE transactions on neural systems and rehabilitation engineering : a publication of the IEEE Engineering in Medicine and Biology Society*, 29:1784–1794, 2021.

[6] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:3438–3445, 4 2020.

[7] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in Neural Information Processing Systems*, pages 3844–3852, 6 2016.

[8] Andac Demir, Toshiaki Koike-Akino, Ye Wang, Masaki Haruna, and Deniz Erdogmus. Eeg-gnn: Graph neural networks for classification of electroencephalogram (eeg) signals. 2021.

[9] Hongyang Gao and Shuiwang Ji. Graph u-nets. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2083–2092. PMLR, 09–15 Jun 2019.

[10] Raveesh Garg, Eric Qin, Francisco Muñoz-Martínez, Robert Guirado, Akshay Jain, Sergi Abadal, José L Abellán, Manuel E Acacio, Eduard Alarcón, Sivasankaran Rajamanickam, et al. Understanding the design space of sparse/dense multiphase dataflows for mapping graph neural networks on spatial accelerators. *Proceedings of the IPDPS'22*, 2022.

[11] Cai Gillis, Fariba Mirzaei, Michele Potashman, M. Arfan Ikram, and Nancy Maserejian. The incidence of mild cognitive impairment: A systematic review and data synthesis. *Alzheimer's and Dementia: Diagnosis, Assessment and Disease Monitoring*, 11:248–256, 12 2019.

[12] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 2017-December:1025–1035, 6 2017.

[13] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[14] Michiel Hermans and Benjamin Schrauwen. Training and analysing deep recurrent neural networks. volume 26. Curran Associates, Inc., 2013.

[15] Jasper H.H. The ten-twenty electrode system of the international federation. electroencephalography and clinical neurophysiology. *Electroencephalography and clinical neurophysiology. Supplement*, pages 367–380, 1958.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 11 1997.

[17] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4:251–257, 1 1991.

[18] Mohammad Parsa Hosseini, Amin Hosseini, and Kiarash Ahi. A review on machine learning for eeg signal processing in bioengineering. *IEEE Reviews in Biomedical Engineering*, 14:204–218, 2021.

[19] Cosimo Ieracitano, Nadia Mammone, Alessia Bramanti, Amir Hussain, and Francesco C. Morabito. A convolutional neural network approach for classification of dementia stages based on 2d-spectral representation of eeg recordings. *Neurocomputing*, 323:96–107, 1 2019.

[20] Cosimo Ieracitano, Nadia Mammone, Amir Hussain, and Francesco Carlo Morabito. A convolutional neural network based self-learning approach for classifying neurodegenerative states from eeg signals in dementia. *Proceedings of the International Joint Conference on Neural Networks*, 7 2020.

[21] Cosimo Ieracitano, Nadia Mammone, Amir Hussain, and Francesco Carlo Morabito. A convolutional neural network based self-learning approach for classifying neurodegenerative states from eeg signals in dementia. *Proceedings of the International Joint Conference on Neural Networks*, 7 2020.

[22] Ziyu Jia, Youfang Lin, Jing Wang, Ronghao Zhou, Xiaojun Ning, Yuanlai He, and Yaoshuai Zhao. Graphsleepnet: Adaptive spatial-temporal graph convolutional networks for sleep stage classification. volume 2021-January, 2020.

[23] Woojeong Jin, Meng Qu, Xisen Jin, and Xiang Ren. Recurrent event network: Autoregressive structure inference over temporal knowledge graphs. *EMNLP 2020 - 2020 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pages 6669–6683, 4 2019.

[24] Valer Jurcak, Daisuke Tsuzuki, and Ippeita Dan. 10/20, 10/10, and 10/5 systems revisited: Their validity as relative head-surface-based positioning systems. *NeuroImage*, 34:1600–1611, 2 2007.

[25] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 12 2014.

[26] Youngchul Kwak, Woo Jin Song, and Seong Eun Kim. Graph neural network with multilevel feature fusion for eeg based brain-computer interface. 2020.

[27] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3734–3743. PMLR, 09–15 Jun 2019.

[28] Ruilin Li, Zirui Lan, Jian Cui, Olga Sourina, and Lipo Wang. Eeg-based recognition of driver state related to situation awareness using graph convolutional networks. *Proceedings - 2020 International Conference on Cyberworlds, CW 2020*, pages 180–187, 9 2020.

[29] William S. Lovejoy and Christoph H. Loch. Minimal and maximal characteristic path lengths in connected sociomatrices. *Social Networks*, 25:333–347, 10 2003.

[30] Giannis Nikolentzos, George Dasoulas, and Michalis Vazirgiannis. k-hop graph neural networks. *Neural Networks*, 130:195–205, 7 2019.

[31] Texas Department of State Health Services. Alzheimer's disease questions and answers, 6 2022.

[32] Robert Oostenveld and Peter Praamstra. The five percent electrode system for high-resolution eeg and erp measurements. *Clinical neurophysiology : official journal of the International Federation of Clinical Neurophysiology*, 112:713–719, 2001.

[33] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10843 LNCS:593–607, 3 2017.

[34] Xiaocai Shan, Jun Cao, Shoudong Huo, Liangyu Chen, Ptolemaios Georgios Sarrigiannis, Yifan Zhao, and Correspondence Yifan Zhao. Spatial–temporal graph convolutional network for alzheimer classification based on brain functional connectivity imaging of electroencephalogram. *Human Brain Mapping*, 6 2022.

[35] Qiaoqiao Shi, Wei Li, Fan Zhang, Wei Hu, Xu Sun, and Lianru Gao. Deep cnn with multi-scale rotation invariance features for ship classification. *IEEE Access*, 6:38656–38668, 7 2018.

[36] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjing Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *IJCAI International Joint Conference on Artificial Intelligence*, pages 1548–1554, 9 2020.

[37] Biao Sun, Han Zhang, Zexu Wu, Yunyan Zhang, and Ting Li. Adaptive spatiotemporal graph convolutional networks for motor imagery classification. *IEEE Signal Processing Letters*, 28:219–223, 2021.

[38] Vladas Valiulis. The effect of transcranial magnetic stimulation on brain bioelectrical activity. 2014.

[39] Giuseppe Varone, Wadii Boulila, Michele Lo Giudice, Bilel Benjdira, Nadia Mammone, Cosimo Ieracitano, Kia Dashtipour, Sabrina Neri, Sara Gasparini, Francesco Carlo Morabito, Amir Hussain, and Umberto Aguglia. Signatures of brain network alteration in psychogenic non-epileptic seizures: A rest-eeg study based on power spectral density and phase lag index. *bioRxiv*, page 2021.10.20.464353, 10 2021.

[40] Petar Veličković, Arantxa Casanova, Pietro Liò, Guillem Cucurull, Adriana Romero, and Yoshua Bengio. Graph attention networks. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 10 2017.

[41] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 11 2015.

[42] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[43] Neeraj Wagh and Yogatheesan Varatharajah. Eeg-gcnn: Augmenting electroencephalogram-based neurological disease diagnosis using a domain-guided graph convolutional neural network. 11 2020.

[44] Jialin Wang, Shen Liang, Dake He, Ye Wang, Yingpei Wu, and Yanchun Zhang. A sequential graph convolutional network with frequency-domain complex network of eeg signals for epilepsy detection. *Proceedings - 2020 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2020*, pages 785–792, 12 2020.

[45] Axel Wassington and Sergi Abadal. Prognnosis: A data-driven model to predict gnn computation time using graph metrics. *arXiv preprint arXiv:2206.08258*, 2022.

[46] P. Welch. The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms. *IEEE Transactions on Audio and Electroacoustics*, 15(2):70–73, 1967.

[47] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. *36th International Conference on Machine Learning, ICML 2019*, 2019-June:11884–11894, 2 2019.

[48] P. Yasodha and N. R. Ananthanarayanan. Detecting the ovarian cancer using big data analysis with effective model. *Biomedical Research (India)*, 2018:S309–S315, 2018.

[49] Rex Ying, Christopher Morris, William L. Hamilton, Jiaxuan You, Xiang Ren, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in Neural Information Processing Systems*, 2018-December:4800–4810, 6 2018.

[50] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32, 4 2018.

[51] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 1 2020.

A. Abbreviations

- EEGs: Electroencephalograms
- GNNs: Graph Neural Networks
- MCI: Mild Cognitive Impairment
- fMRI: Functional Magnetic Resonance
- ECoG: Electrocorticography
- CNN: Convolutional Neural Network
- RSVP: Rapid Serial Visual Presentation
- ML: Machine Learning
- DL: Deep Leaning
- DNN: Deep Neural Network
- RNNs: Recurrent Neural Networks
- MLP: Multi Layer Perceptron
- FC: Fully Connected
- GAT: Graph Attention
- CWT: Continuous Wavelet Transform
- BiS: Bispectrum