UNIVERSITAT DE BARCELONA

FUNDAMENTAL PRINCIPLES OF DATA SCIENCE MASTER'S THESIS

The Brain Coding of Multidimensional Time Series

Author:
Alejandro ASTRUC LOPEZ

Supervisor:
Dr. Ignasi COS AGUILERA

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Fundamental Principles of Data Science

in the

Facultat de Matemàtiques i Informàtica

June 30, 2024

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

The Brain Coding of Multidimensional Time Series

by Alejandro ASTRUC LOPEZ

Electroencephalography (EEG) is a widely used technique in the study of brain function. A series of electrodes are placed on the scalp measuring an electric signal from a population of neurons over time. In this work, we will focus on the classification of EEG data. The conventional approach to analyse and classify EEG data employed feature extraction methods. However, deep learning techniques have started to be applied to this task. Among the different architectures, Graph Neural Networks (GNNs) have gained especial attention as EGG data contains complex spatio-temporal relations of high dimensionality, that can be interpreted as a graph. Given the potential of GNNs, we will propose a series of models and try to classify and separate different EEGs into three classes of motivation. The data comes from Cos, Deco, and Gilson, Unpublished, a study focusing on the influence of social motivation during a decision making task. Once the models are trained, we will discus their performance and compare them with the results from the aforementioned study.

Acknowledgements

To Dr. Ignasi Cos, my supervisor, for the data, his expertise, and help during the development of this master thesis.

To Enrique Alonso, for his advise and overlook of the latest stages of the text.

To my family, for their full support, care, patience and unwavering trust.

To Gabriel Plaza, for their presence helping me stay focused and collected.

Chapter 1

Introduction

Electroencephalography (EEG) is a widely used technique in the study of brain function. A series of electrodes are placed on the scalp (either individually or embedded in a cap or net). Each electrode gathers an electric signal from a population of neurons over time. The signal represents an average of thousands of neurons of which only large synchronous activity is reflected. This limited spatial resolution is one of the main challenges of EEG. Some other challenges associated with EEG is the low signal-to-noise ratio, non-stationarity and the high dimensionality of the data.

EEG has seen a variety of applications: It has allowed for the discovery of neural oscillations such as alpha waves, beta waves, etc, which have been associated with wakefulness, mental activity, and so on. EEG has also found a role in the medical monitoring of patients as well as informing medical diagnosis (Wikipedia contributors, 2024). In this work we will focus on the classification of EEG data.

The conventional approach to analyse and classify EEG data employed feature extraction methods, some related to the aforementioned neural oscillations (e.g., power spectral density), with the purpose of characterising the brain state. As such, this kinds of manually extracted features pose a series of limitations ranging from selection biases to subjectivity, and generalisation. These limitations have motivated the use of automated feature extraction methods, introducing deep learning architectures to the framework.

Some of the architectures used to analyse this type of data are convolutional neural networks (CNNs), long short-term memory (LSTM) networks, etc., but most recently Graph Neural Networks (GNNs) have gained especial attention. EGG data contains complex spatio-temporal relations of high dimensionality, that can be interpreted as a graph. As a consequence, GNNs have the potential to automatically infer the meaningful features (and patterns) necessary to make accurate predictions (Klepl, Wu, and He, 2023).

Given the potential of GNNs, we will try to classify and separate different EEGs into three classes of motivation. The data comes from Cos, Deco, and Gilson, Unpublished, a study focusing on the influence of social motivation during a decision-making task . The original analysis focuses on the aforementioned conventional approach identifying the two most relevant sets of features: electrodes for power and interactions between pairs of electrodes for correlation. These features underline the relevance of GNNs in the analysis. We will train a series of GNN models on the data as a first step to explore the possibilities offered by GNNs in this classification task.

Once we have trained these models, we will discuss its limitations and possible improvements, proposing several ways of continuing this work.

1.1 Overview of Convolutional and Graph Neural Networks

Convolutional neural networks are a type of artificial neural network often used to solve image recognition tasks through the encoding of image features using kernels in a convolutional layer of the network architecture. Kernels are a set of shared weighs across the convolutional layer that are displaced over the previous layer, tiling it while allowing an overlap in its application. CNNs have made impressive achievements in image recognition and have also been expanded to other areas such as natural language processing, quickly becoming one of the most significant networks in the deep learning field.

One of the main properties of CNNs is the incorporation of a notion of locality in its architecture. The two main layers employed in the encoding of local information are the convolutional layers and the pooling layers. Convolutional layers apply kernels to the output of a layer and tile the whole input by displacing and reapplying the kernel. The output is a measure of resemblance between the kernel and each step of the tiling. Pooling layers work in a similar fashion, tiling the input by displacing a fixed size window that aggregates the output of the previous layer, reducing its dimensionality.

The effective outcome of stacking convolutional layers and pooling layers is that each neuron now has a receptive field.[1] Instead of receiving inputs from all the neurons in the previous layer, as it is the case for fully connected neural networks, each neuron in a CNN receives inputs from only a subset of neurons from the previous layer. As a result, the deeper in the network a neuron is, the larger the area of the original input it perceives.

All in all, the achievements of the CNN architectures arise from its ability to leverage **local connectivity**, shared weights and its use of multiple layers, characteristics which also show up in graphs. Despite of this occurrence, CNNs have a key limitation which arises from its inductive bias and which does not allow them to deal with graph data: its assumption that the input data is **Euclidean**. CNNs can only operate in regular Euclidean data like images, i.e., 2D grids, or text, i.e., 1D sequences. (Zhou et al., 2018) However, in many ways, the main modality in which we receive data from nature are graphs, which also easily subsume Euclidean data: for instance, each pixel of a 2D grid can be thought as a node which is connected to adjacent nodes, thus encoding pixel closeness. (Veličković, 2023) One such example of graph data which cannot be thought as Euclidean data is that coming from an electroencephalographies (EEGs). In this case, the electrodes capture data coming from around a curved surface, a human's head, and the features read in each point are non-locally related through electrical signals traveling through the brain's connections. Other typical examples of graph data include chemical molecules, social networks or transportation networks.

Graph neural networks (GNNs) encompass and generalize the concept of CNNs to graphs. They allow us to encode and capture **non-local**, **long-distances relationships** in our data, as well as data coming from **non-Euclidean**, **curved spaces**. In the next chapter, we will explain how GNNs work and their limitations.

[1] This concept relates to neurons function in the visual cortex function. Neurons are sensitive to certain areas of the visual field and to directions of movement. Neurons which are closer together in the visual cortex have similar receptive fields.

Chapter 2

Graph Convolutions

An observation is given by a graph $\mathcal{G} = (V, E)$, with V the set of vertices, $E \subseteq V \times V$ the set of edges of that graph, and with edges being nothing more than ordered pairs of vertices in the case directed graphs, or unordered pairs otherwise.

Furthermore, each vertex $v \in V$ has a corresponding features vector attached, $x_v \in \mathbb{R}^p$. The features of all nodes can be conveniently expressed through the node feature matrix: if $|V| = n$ is the number of vertices of the graph, the node feature matrix is just a matrix $X \in \mathbb{R}^{n \times p}$ obtained from stacking the node features in rows, that is:

$$X = [x_1, ..., x_n]^T$$

.

To represent the edges in E, we can use the adjacency matrix $A \in \mathbb{R}^{n \times n}$, defined by

$$A_{uv} = \begin{cases} 1 & \text{if } (u, v) \in E \\ 0 & \text{if } (u, v) \notin E \end{cases}$$

Using the aforementioned representation for edges requires an **arbitrary** node ordering over V imposed by us. Hence, any graph neural network should not take this ordering into account in order to learn from the data. That is, given any permutation matrix P, for which, by definition, $P^{-1} = P^T$, any GNN must satisfy the following two rules:

$$f(PX, PAP^T) = f(X, A) \quad \text{(GNN invariance)}$$
$$F(PX, PAP^T) = PF(X, A) \quad \text{(GNN equivariance)}$$

where f returns a graph-level output, i.e., an univariate output, and F returns a node-level output, i.e., a multivariate n-dimensional output.

The edges of the input graph(s) allow for the natural implementation of the locality constraint(s) in GNNs. Just like CNNs operate around an Euclidean neighbourhood of each pixel of an image, **GNNs operate over the neighbourhood of the nodes of a graph**. The standard way to define the smallest neighbourhood of the graph is via the 1-hop neighbours of that node, that is, those nodes that are connected to that node through an edge:

$$\mathcal{N}_v = \{u \,|\, (v, u) \in E \text{ or } (u, v) \in E\}$$

and we can define the multiset of all neighbourhood features as

$$X_{\mathcal{N}_v} = \{\{x_u \,|\, u \in \mathcal{N}_v\}\}$$

We then define our local function ϕ, which takes into account the neighbourhood of an edge, and from which we can obtain a graph-level multivariate output function F, i.e.

$$h_v = \phi(x_v, X_{\mathcal{N}_v}), \qquad F(X) = [h_1, \dots h_n]^T$$

If ϕ is invariant, a trivial computation shows F is equivariant.

The neighbourhood of a node v can be extended to the k-hop neighbours of that node, $\mathcal{N}_v^k$, and the corresponding K-hop neighbours information can be defined using the local function as $h_v = \sum_{k=1}^{K} \phi(x_v, X_{\mathcal{N}_v^k})$ or using the concatenation of the outputs for each k (Demir et al., 2021).

The remaining choice is how to choose ϕ, which is usually called the **diffusion function**, the **propagation function** or the **message passing function**. It is currently a very active area in research, but most GNNs can be classified in three categories depending on how they treat the information coming from the nodes' neighbours. These three categories are **convolutional**, **attentional** or **message-passing GNNs**, and their information encoding, as given in Bronstein et al., 2021, is:

$$h_v = \phi\left(x_v, \bigoplus_{u \in \mathcal{N}_v} c_{uv}\psi(x_u)\right) \qquad \text{(Convolutional GNNs)}$$

$$h_v = \phi\left(x_v, \bigoplus_{u \in \mathcal{N}_v} a(x_v, x_u)\psi(x_u)\right) \qquad \text{(Attentional GNNs)}$$

$$h_v = \phi\left(x_v, \bigoplus_{u \in \mathcal{N}_v} \psi(x_v, x_u)\right) \qquad \text{(Message-passing GNNs)}$$

where ϕ and ψ are artificial neurons, such as $\phi(x) = \text{ReLu}(Wx + b)$, leaky ReLu or sigmoid functions, and $\bigoplus$ is any aggregator which is invariant under permutations such as the sum, the mean or the max function. The general idea is to build the rest of the network on the output of the embedding layers. The types of GNNs are ordered by increasing expressive power, but also by decreasing interpretability, scalability and learning stability. Each category is also more general than the previous one, subsuming it in a more general expression. It must be said that these categories are quite general, and each of them have many different examples in the literature.

Given any GNN, we can learn different tasks over the input graph data. These include, but are not limited to, the following three tasks:

1. **Node classification.** The aim is to predict targets for each node $v \in V$. Thus, our output is equivariant, and the problem is equivalent to learning a shared classifier for all nodes directly of h_v. A typical example of this is classifying protein functions in a given protein-protein interaction network. This example was first tackled using a message-passing GNN named GraphSage (Hamilton, Ying, and Leskovec, 2017), which is a GNN specialized in dealing with large graphs.

2. **Graph classification.** If we predict targets for the whole graph, we need an invariant output. Thus, we first reduce all the h_v into a common representation,

$\bigoplus_{v \in V} h_v$, and we then have to learn a classifier over this common representation. A typical example of this is estimating pharmacological properties of molecules, such as drug toxicity (Jiang et al., 2021).

3. **Link prediction.** The aim is to predict properties of edges (v, u), or even if a given edge exists. In this case, the classifier is learnt over the concatenation of features $h_v || h_u$, together with any edge-level features that may exist. A typical example of this is drug repurposing, which can be though as predicting treatments between drugs and diseases, or equivalently edges between drug nodes and disease nodes (Morselli Gysi et al., 2021).

This work pertains to the second type of tasks, graph classification, but could be further extended to encompass the third task, link prediction. We will implement and adaptive modification of our models that implicitly performs a link prediction task as well as a graph classification task.

2.1 GCN convolution

In our case we will use GCN convolutions from Kipf and Welling, 2016, as they are implemented in the torch_geometric library:

$$\mathbf{X}' = \hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\mathbf{X}\mathbf{\Theta},$$

or in its node-wise formulation:

$$\mathbf{x}_i' = \mathbf{\Theta}^\top \sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{e_{j,i}}{\sqrt{\hat{d}_j \hat{d}_i}} \mathbf{x}_j$$

where $x_j \in \mathbb{R}^\tau$ are the node features, $\Theta \in \mathbb{R}^{\tau \times \tau}$ is the matrix of trainable weights from the graph convolution, τ is the remaining time dimensionality after the time aggregation of the previous layers, $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ (I identity) adjacency matrix with inserted self-loops, $\hat{D}_{ii} = \sum_{j=0} \hat{A}_{ij}$ the diagonal degree, $\hat{d}_i = 1 + \sum_{j \in \mathcal{N}(i)} e_{j,i}$ where $e_{j,i}$ denotes the edge weight from source node j to target node i (one by default).

2.2 Self-adaptive Adjacency Matrix

The first use of a Self-Adaptive Adjacency Matrix comes from Wu et al., 2019, where they were performing a traffic forecasting task, with spatio-temporal graph modeling. They found that the performance of the model was improved both when a previous adjacency matrix was known and when the model had to deduce it entirely. The general formulation for this self-adaptive adjacency matrix is:

$$\mathbf{A}_{adap} = SoftMax(ReLu(\mathbf{E}_1 \mathbf{E}_2)^T),$$

where $\mathbf{E}_1$ is the source node embedding and $\mathbf{E}_2$ is the target node embedding. However, we will not use this definition. Instead we will use a layer inspired on this type of self-adaptive adjacency matrix. We will further describe it in the methodology.

Chapter 3

Methodology

3.1 Data

The group of subjects studied in Cos, Deco, and Gilson, Unpublished, were eleven right-handed individuals (4M and 7F; M. age = 55yrs, SD = 5.8). The participants were in an electrically shielded room during the experiments. The EEG was taken from 60 points on the scalp embedded on a cap. The signals were filtered with a bandpass of 0.1-100 Hz and digitized at a rate of 500 Hz. The samples span a time of 120 0ms with a resolution of 1 ms.

The participants' EGGs were taken during a variety of decision-making tasks between 2 reaching movements. According to the different experimental factors, the realisations were grouped in 12 blocks of 108 trials each. The factor of study in this case is the three motivated states:

0) Solo. It corresponds to blocks 1, 2, 7 and 8. We will call instances, or samples, from this motivated state, class 0.

1) Easy. It corresponds to blocks 3, 4, 9 and 10. We will call instances, or samples, from this motivated state, class 1.

2) Hard. It corresponds to blocks 5, 6, 11 and 12. We will call instances, or samples, from this motivated state, class 2.

The generation of this motivation states, was achieved by means of social pressure. The objective was to introduce a subliminal bias based on the illusion of competing (in terms of their aiming skill) against other participants of varying aptitudes. Solo means without contender, Easy and Hard refer to how good the fictional contender was at performing the task.

The preprocesing of the data was already performed in Cos, Deco, and Gilson, Unpublished. There are channels and trials where the data was discarded and replaced by nans. In chapter 4 we will first explore the data to see which channels and trials were discarded, how it affects each class and how it might bias our models.

3.2 Problem setup

The data at hand is made up of preprocessed signals coming from EEG scans of eleven subjects under different circumstances.

Given a subject, each sample is a collection of n signals from the respective nodes, also called channels. They can be seen as a multidimensional time series, i.e., a time evolving n dimensional vector:

$$\begin{gathered}\{X_s^t \in \mathbb{R}^n\}, \\ t \in \{1, 2, \ldots, T\}, \\ s \in \{1, \ldots, N\}.\end{gathered}$$

Other way to see each sample is as a matrix:

$$\begin{gathered}\{M_s \in \mathbb{R}^{n \times T}\}, \\ s \in \{1, \ldots, N\}.\end{gathered}$$

Or even as a graph. A graph pair $G = (V, E)$, where V is the set of vertices, in this case corresponding to each channel ($|V| = n$), and E is the set of unordered pairs $\{v_i, v_j\}$ called edges. In our case the edge structure is supposed to capture the underlying interactions between the brain regions that are being monitored in the EEG. Each signal would be a time dependent feature of each vertex of the graph. Then for each sample $s \in \{1, \ldots, N\}$:

$$\begin{gathered}x_k \in \mathbb{R}^\tau, \\ \tau \in \{1, 2, \ldots, T\}, \\ k \in \{1, \ldots, n\},\end{gathered}$$

that is, a set of node feature matrices:

$$\begin{gathered}\{X_s = [x_1, ..., x_n]^T\}_{s=1}^N \\ X_s \in \mathbb{R}^{n \times \tau}\end{gathered}$$

where τ represents the reduced dimensionality of the time component when aggregated. And every sample has an associated label $y_s \in \{0, 1, 2\}$. Our data can be expressed as $\{X_s, y_s\}_{s=1}^N$.

These three ways of looking at the data are equivalent but suggest different strategies to model it. The architecture of our models takes inspiration on these perspectives. As a matrix, M_s can be taken as series of 1D images as seen by a 1D convolution. The resulting output of reduced dimensionality in time is then interpreted by the graph convolutions as feature vectors associated to graph nodes. In this way we reduce the size of the future vectors while selecting from the relevant features in the convolutional layers.

3.3 Models

EEG data is an example of multidimensional time series, but these signals can also be interpreted as a collection of one dimensional images, hence susceptible to be modeled using a CNN. The motivation behind our approach is to take the most advantage out of the time component of the data, letting the model extract the features necessary for the task. There are many strategies to deal with the time aspect of graph structured data but we wanted to keep the raw signal. Thus, in order to design a first model we started from a simple time aggregation of the different incoming channels.

Next, we iterated over this initial model adding graph structure (i.e., introducing GCN layers), and experimenting with different variations. Further on, we will go into each of them. The idea is to incorporate non-local interactions between the different signals through graph convolutions. Finally we add an additional layer that allows for the adjacency matrix to be input dependent, i.e., adaptable.

The code for the models and the different functions used to plot and analyse the data can be found on: https://github.com/45truc/TFM.

3.3.1 TimeAggNet

This first base model is composed of 1D convolutions, max. pooling layers, followed by ReLu activations, and a final linear classifier with a softmax activation. All models have the same last classification layer, and their output is the probability that the sample belongs to each class.

In this model all channels share the same 1D convolutions layers, that is, the same convolution is applied in parallel to each signal. All time signals are aggregated into a singular real value for each channel, and this internal representation is what the classification layer receives.

The model is represented on Figure 3.1. In summary:

- For this network each channel is independent of each other, it is blind to inter channel relations. In the end it just aggregates the time component of the data, hence its name.
- This model not only serves as a base for other more complex models, but also allows to gauge the impact in performance of incorporating graph convolutions into its architecture.
- The model has 17 parameters from the 1D convolutions, 126 parameters from the classifier that add up to a total of 143 parameters.

Here is the model description in terms of *PyTorch* layers:

```
  TimeAggNet(
(time_convs): ModuleList(
  (0): Conv1d(1, 1, kernel_size=(10,), stride=(8,), padding=(1,))
  (1): MaxPool1d(kernel_size=5, stride=5, padding=0, dilation=1, ceil_mode=False)
  (2): Conv1d(1, 1, kernel_size=(3,), stride=(3,))
  (3): MaxPool1d(kernel_size=3, stride=3, padding=1, dilation=1, ceil_mode=False)
  (4): Conv1d(1, 1, kernel_size=(4,), stride=(1,))
)
```

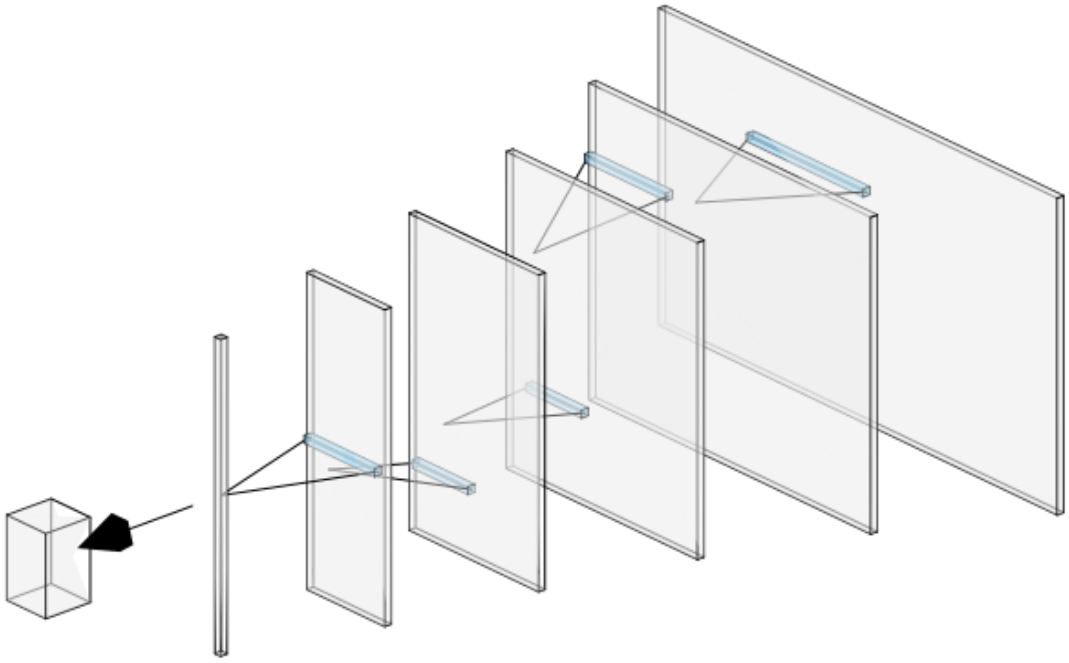

FIGURE 3.1: The TimeAggNet model constituted by 3 1D convolutions with 2 max. pooling layers in between, each with a ReLu activation and a final layer for the classifier, a fully connected layer that receives the time aggregated layer of n channels and outputs 3 nodes corresponding to each class with a softmax activation. Both layer and kernel size are in log scale. The image was generated with with a web tool from LeNail, 2019.

```
  (class_layer): Linear(in_features=42, out_features=3, bias=True)
)
```

3.3.2 TimeGraphNet

This is the first modification to TimeAggNet to incorporate graph convolutions. Its architecture is almost as that represented on Figure 3.1, but it incorporates 2 GCN layers, one after layer 2 and another after layer 4. These allow for modeling channel interactions at different timescales, $\tau_1 = 30$ and $\tau_2 = 4$. The graph was assumed to be fully connected and the time reduced signals were taken as node features.

The model has 17 parameters from the 1D convolutions, 916 parameters from the GCNs and 126 parameters from the classifier that add up to a total of 1059 parameters.

Here is the model in terms of *PyTorch* and *PyTorch_Geometric* layers:

```
 TimeGraphNet(
(time_convs): ModuleList(
  (0): Conv1d(1, 1, kernel_size=(10,), stride=(8,), padding=(1,))
  (1): MaxPool1d(kernel_size=5, stride=5, padding=0, dilation=1, ceil_mode=False)
  (2): Conv1d(1, 1, kernel_size=(3,), stride=(3,))
  (3): MaxPool1d(kernel_size=3, stride=3, padding=1, dilation=1, ceil_mode=False)
  (4): Conv1d(1, 1, kernel_size=(4,), stride=(1,))
)
(gcn1): GCNConv(30, 30)
```

```
  (gcn2): GCNConv(4, 4)
  (class_layer): Linear(in_features=42, out_features=3, bias=True)
)
```

3.3.3 DeepTimeGraphNet

This is a deeper variation of the previous model, with more layers, and smaller kernels. It also employs more parameters in the GCN layers.

The model has 17 parameters from the 1D convolutions, 2644 parameters from the GCNs and 126 parameters from the classifier that add up to a total of 2787 parameters.

Here is the model in terms of *PyTorch* and *PyTorch_Geometric* layers:

```
    DeepTimeGraphNet(
  (time_convs): ModuleList(
    (0): Conv1d(1, 1, kernel_size=(2,), stride=(2,))
    (1): MaxPool1d(kernel_size=3, stride=3, padding=0, dilation=1, ceil_mode=False)
    (2): Conv1d(1, 1, kernel_size=(4,), stride=(2,), padding=(1,))
    (3): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv1d(1, 1, kernel_size=(4,), stride=(2,), padding=(1,))
    (5): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv1d(1, 1, kernel_size=(4,), stride=(2,), padding=(1,))
    (7): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv1d(1, 1, kernel_size=(3,), stride=(1,))
  )
  (gcn1): GCNConv(50, 50)
  (gcn2): GCNConv(12, 12)
  (class_layer): Linear(in_features=42, out_features=3, bias=True)
)
```

3.3.4 SimpleTimeGraphNet

In this iteration we take the same architecture as for TimeGraphNet but simplify it by only employing one GCN layer. This would correspond to the second GCN layer from TimeGraphNet, that is, now $\tau = 4$.

The model has 17 parameters from the 1D convolutions, 16 parameters from the GCNs and 126 parameters from the classifier that add up to a total of 159 parameters.

Here is the model in terms of *PyTorch* and *PyTorch_Geometric* layers:

```
    SimpleTimeGraphNet(
  (time_convs): ModuleList(
    (0): Conv1d(1, 1, kernel_size=(10,), stride=(8,), padding=(1,))
    (1): MaxPool1d(kernel_size=5, stride=5, padding=0, dilation=1, ceil_mode=False)
    (2): Conv1d(1, 1, kernel_size=(3,), stride=(3,))
    (3): MaxPool1d(kernel_size=3, stride=3, padding=1, dilation=1, ceil_mode=False)
    (4): Conv1d(1, 1, kernel_size=(4,), stride=(1,))
  )
  (gcn1): GCNConv(4, 4)
  (class_layer): Linear(in_features=42, out_features=3, bias=True)
)
```

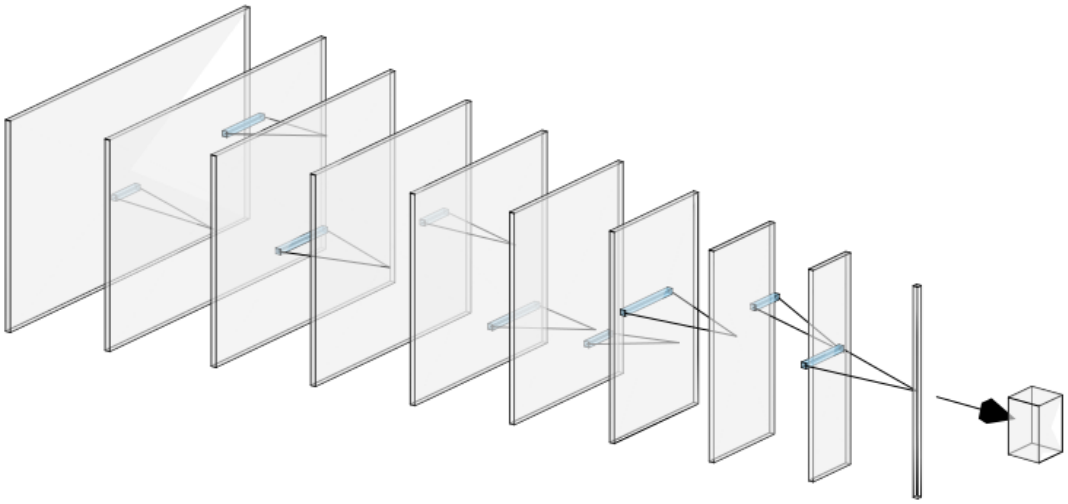

FIGURE 3.2: The DeepTimeGraphNet model made up of 5 1D convolutions, with 4 pooling layers in between, each with ReLu activations, after layers 4 and 6 there are GCN layers that do not reduce time dimensionality, (they are not represented on the diagram) and a final fully connected layer for classifying. Both layer and kernel size are in log scale. The image was generated with a web tool from LeNail, 2019.

3.3.5 Adaptive Modification

In order to improve on the assumption of a fully connected graph in the GCN layers, we incorporated a new layer that allows for an input dependant construction of the adjacency matrix: the AdaptiveGCNLayer. The main idea is to train a series of weights that construct the adjacency matrix adapted to the input. This is the code description of the layer:

It first initialises a weight matrix is: $W \in \mathbb{R}^{\tau \times \tau}$ and the process to calculate the adjacency matrix

$$A = XWX^T + I$$

where X is the node feature matrix, i.e., a sample that has undergone some time aggregation $X \in \mathbb{R}^{n \times \tau}$. Then A is normalised before being fed to a GCN. This layer adds τ^2 parameters to de τ^2 parameters of he GCN layer (that is, a total of $2\tau^2$ parameters).

The models that have been modified to replace the original GCN layers by this adaptive GCN have and added suffix "-Adap" to their name.

3.4 Experiments

For the training of the models we performed a 80% train 20% test split of each subject's data. Each model was trained on each subject during 250 epochs using the Adam optimizer with a learning rate of 0.001. The class labels are one hot encoded and the loss function of choice was mean squared error (MSE), which is a proper scoring rule. This last choice makes the output of the models a calibrated probability.

When the models did not correctly converge they were retrained until they converged. By not correctly converge we mean that there was no significant decrease in the loss function nor improvement of train accuracy.

The metrics we have extracted are: accuracy, recall, precision, and f1-score. The convention and standards of the field would require for the realisation of multiple random train test splits or validation train validation splits. In our case time was a limiting factor so only one instance of each model was realised. We leave this step for future work.

Because of this, we will use a traditional deep learning approach (Raschka, 2018). For the errors we will use a 95% confidence interval using a normal approximation based on the test set:

$$\bar{x} \pm z \times \text{SE}$$

in our case for a 95% confidence $z = 1.959963984540054$, i.e., the number of standard deviations (SE) a value lays away from the mean:

$$\text{SE} = \sqrt{\frac{1}{n}\text{ACC}_{\text{test}}\left(1 - \text{ACC}_{\text{test}}\right)},$$

n being the size of the test. And so:

$$\text{ACC}_{\text{test}} \pm z\sqrt{\frac{1}{n}\text{ACC}_{\text{test}}\left(1 - \text{ACC}_{\text{test}}\right)}.$$

For the case where we would have different repetitions with different random seeds we would instead use:

$$\overline{ACC}_{\text{test}} \pm t \times \text{SE},$$

with $\text{SE} = \frac{\text{SD}}{\sqrt{r}}$, where r is the number of rounds or repetitions with a different seed, t is the value of a Student's t-distribution for our confidence and $r - 1$ degrees of freedom $\overline{ACC}_{\text{test}} = \frac{1}{r}\sum_{j=1}^{r} ACC_{\text{test},j}$, the mean and:

$$\text{SD} = \sqrt{\frac{\sum_j \left(ACC_{\text{test},j} - \overline{ACC}_{\text{test}}\right)^2}{r-1}}$$

.

Chapter 4

Results

In this section we will begin by further characterising the analysed data. Then we will focus on the results from the training of the different models to finally compare our findings with those in Cos, Deco, and Gilson, Unpublished.

4.1 Data exploration

When exploring the data some of the blocks were missing entirely. These are:

- Blocks 4 and 9 for subject 1.
- Blocks 2 and 9 for subject 4.
- Blocks 5 and 11 for subject 6.

Moreover some of the trials from the remaining blocks were empty as well. The effects on the number of samples of each class is summarised on Table 4.1. The class unbalance in some of the classes might have an impact on the classifiers and negatively bias them towards the underrepresented classes. More specifically in subjects 1, 4, and 6. For the case of subject 4 class 1 is underrepresented, for subject 6 class 2 is underrepresented and subject 1 class 0 is over-represented.

	% of class 0	% of class 1	% of class 2	Total samples
Subject 0	33	33	33	1296
Subject 1	36	32	32	1017
Subject 2	33	33	33	1296
Subject 3	33	33	33	1296
Subject 4	40	20	40	1080
Subject 5	33	33	33	1296
Subject 6	40	40	20	1080
Subject 7	33	33	33	1296
Subject 8	33	33	33	1296
Subject 9	33	33	33	1296
Subject 10	33	33	33	1296

TABLE 4.1

As for the channels, they seem to be missing at random from each block and subject. This can be seen in Figure 4.1, or more in detail in Appendix A, Table **??**.

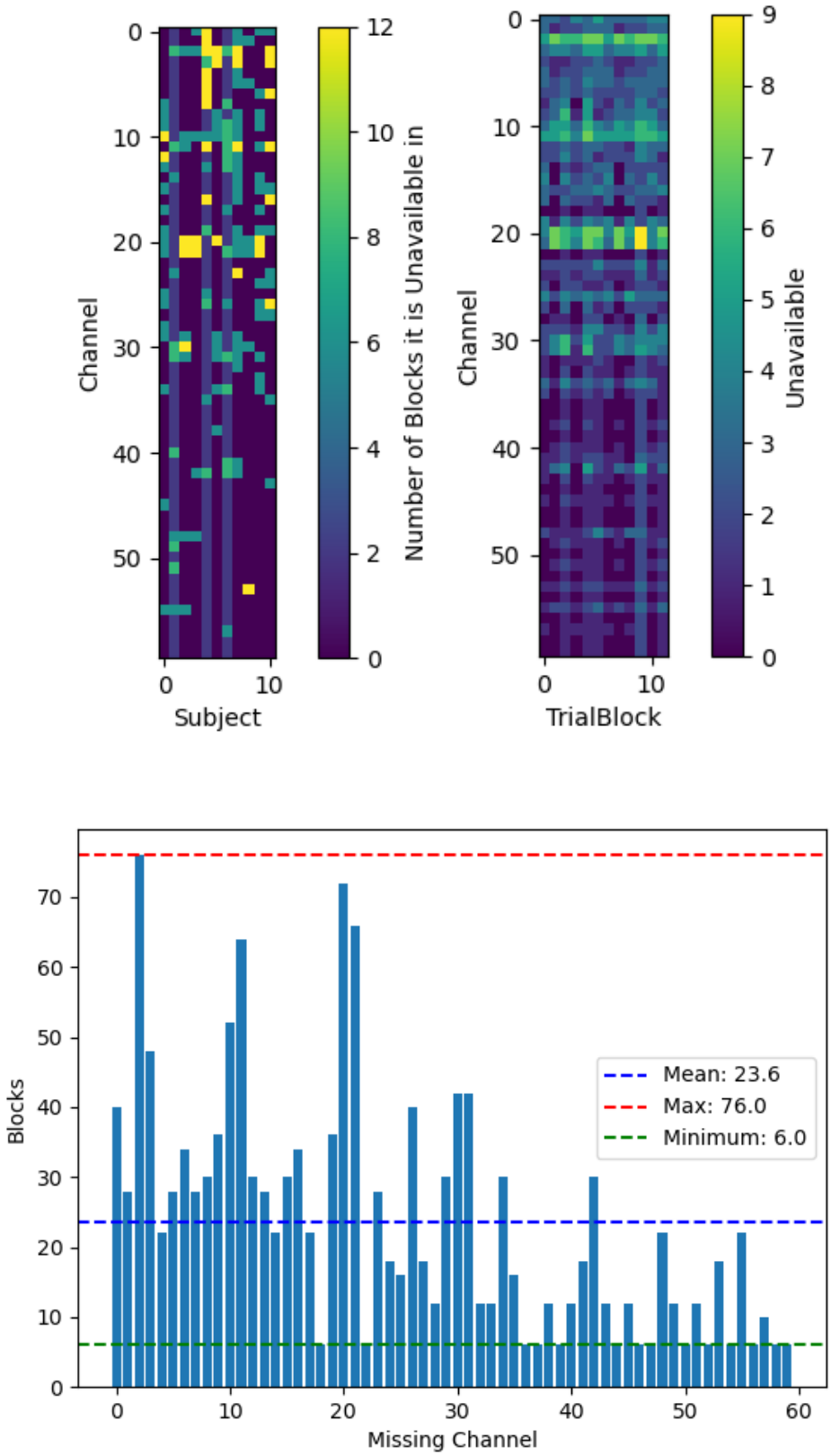

FIGURE 4.1: Missing channels across subjects, within all blocks of the whole data.

4.2 Models

In this section we will begin by looking at the training of the models addressing difficulties and resulting metrics. We will continue with the test results and finally compare our results with those in Cos, Deco, and Gilson, Unpublished in the EEG classification task.

4.2.1 Training

After training the different models over all subjects and doing the average we obtain Table 4.2. At first, TimeAggNet seems to be the best at fitting the data, however there is more variability in the fitting results for the rest of the models, as they are more complex and contain more parameters. While TimeAggNet achieves more consistent results across subjects, DeepTimeGraphNet and DeepTimeGraphNetAdap, its adaptive version, achieve the best fits of 80-90% accuracy for some of the subjects (see Appendix B, Table B.1).

	Loss	Accuracy	Recall	Precision	F1
TimeAggNet	0.112	**0.768**	**0.766**	**0.768**	**0.765**
TimeGraphNet	0.117	0.756	0.752	0.758	0.753
TimeGraphNetAdap	0.117	0.756	0.752	0.758	0.753
DeepTimeGraphNet	0.122	0.732	0.719	0.746	0.716
DeepTimeGraphNetAdap	0.122	0.732	0.719	0.746	0.716
SimpleTimeGraphNet	0.129	0.715	0.709	0.715	0.709
SimpleTimeGraphNetAdap	0.129	0.715	0.709	0.715	0.709

TABLE 4.2: Average loss and metrics on the training set for all models across all subjects. Highest values in bold.

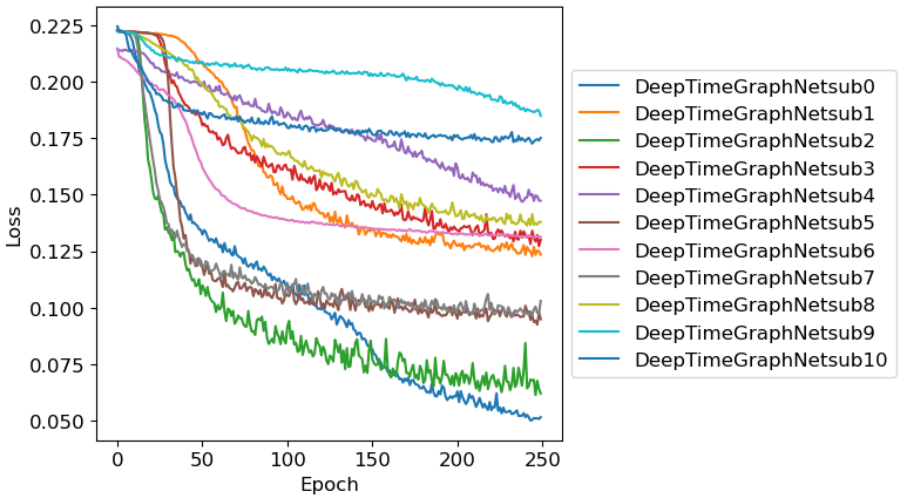

FIGURE 4.2: Training loss for the DeepTimeGraphNet over epochs for all subjects.

Training the models over the whole set of subjects gave rise to a series of problems. Some of the models did not converge at all during training. Moreover, more complex models, with more variables, were the ones that took more attempts to train so that they would converge. For the case of the DeepTimeGraphNet and DeepTimeGraphNetAdap models we have plotted the training loss on Figure 4.2 and 4.3 respectively.

The behaviour for both DeepTimeGraphNet and DeepTimeGraphNetAdap is the same across subjects. They fit accurately for some of the subjects while failing to converge to a low minimum for subjects 1, 10, 4 especially. There might be two main reasons, either the models need modification in architecture or training to better fit the subjects or the intended bias elicited by means of social pressure was not achieved in the same way for all subjects. That is, models are able to perceive only how this social pressure affects a subset of the subjects.

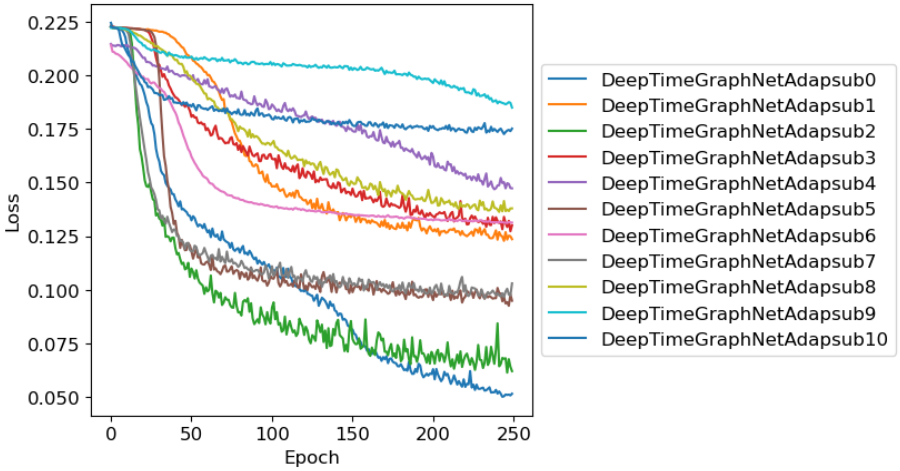

FIGURE 4.3: Training loss for the DeepTimeGraphNetAdap over epochs for all subjects.

4.2.2 Testing

The average test results across subjects are shown in Table 4.3. As mentioned above (section 4.1), three of the subjects have unbalance classes. This is already reflected on the average performance of the models, as the metrics for class 2, underrepresented for two subjects, are lower. This is the case as well for the class 1 as it is underrepresented in one of the subjects. In order to test this, we have also calculated the average excluding these subjects in Table 4.4.

Now, looking at the average data excluding subjects with unbalanced classes we see that the performance of the models across classes is similar. This means that, for further improvement our models should incorporate measures such as a weighted loss in order to avoid biasing against the underrepresented classes.

With respect to the model comparison, the aggregated data seems to suggest that our baseline, TimeAggNet, was the best model from the beginning, however aggregated data is not enough to back this claim. For a coherent model comparison we will need to check the distributions of the different metrics, as the differences might not be significant[1]. If we refer to Figure 4.4, we see that the different mean statistics are similar in value. TimeAggNet and TimeGraphNet show similar distributions with less variability for the latter. TimeGraphNet has some instances of low performance that bring down the average. DeepTimeGraphNet in its adaptive version both have instances of the highest accuracy scores but these are compensated by worse results on other subjects. The SimpleTimeGraphNet net performs similarly to TimeAggNet and TimeGraphNet which makes sense as their architectures resemble. As explained before the more complex models (like DeepTimeGraphNet) took

[1]The whole of the test metrics results for each subject can be found in Appendix A.

	Accuracy	Recall class 0	Recall class 1	Recall class 2
TimeAggNet	**0.709**	**0.742**	0.678	0.695
TimeGraphNet	0.700	0.713	0.655	**0.722**
TimeGraphNetAdap	0.671	0.672	0.661	0.660
DeepTimeGraphNet	0.678	0.696	0.671	0.635
DeepTimeGraphNetAdap	0.661	0.655	0.676	0.632
SimpleTimeGraphNet	0.664	0.663	0.593	0.716
SimpleTimeGraphNetAdap	0.684	0.706	**0.679**	0.637

	Precision class 0	Precision class 1	Precision class 2
TimeAggNet	0.718	0.699	**0.706**
TimeGraphNet	**0.719**	0.686	0.690
TimeGraphNetAdap	0.687	0.684	0.641
DeepTimeGraphNet	0.690	**0.709**	0.605
DeepTimeGraphNetAdap	0.690	0.660	0.633
SimpleTimeGraphNet	0.673	0.662	0.656
SimpleTimeGraphNetAdap	0.707	0.671	0.620

	F1 class 0	F1 class 1	F1 class 2
TimeAggNet	**0.727**	**0.688**	0.699
TimeGraphNet	0.714	0.669	**0.705**
TimeGraphNetAdap	0.677	0.670	0.647
DeepTimeGraphNet	0.684	0.685	0.618
DeepTimeGraphNetAdap	0.669	0.664	0.626
SimpleTimeGraphNet	0.664	0.616	0.681
SimpleTimeGraphNetAdap	0.700	0.674	0.627

TABLE 4.3: Average test metrics for the different models across subjects. Highest values in bold.

more time and more attempts to correctly converge, but when they did we obtained some of the best accuracies. For example, on subject 0 (see Appendix A, Table A.2), the best results come from DeepTimeGraphNet, while for subject 4, results are not that far from TimeAggNet. Figure 4.4 shows that the models while having similar mean performance are not equally matched, moreover, DeepTimeGraphNet and DeepTimeGraphNetAdap have the potential to be the most accurate models with some modifications to their training or to their architecture. Two reasons for their under-performance on some subjects could be attributed to: an inconvenient initialisation of the weights or a difficult propagation of the gradient in the training process. The first issue could be identified by starting from the same partially trained model that for some of the subjects seems to be quite effective. The second issue could be mitigated by the addition of residual channels so the gradient propagates more effectively, or even changing to their activation functions such us leaky ReLu.

In a similar manner, if we look at the class metrics distributions on Figure 4.6, there seems to be no bias against the classes with the exception of some outliers that correspond to DeepTimeGraphNet on subject 6 and SimpleTimeGraphNet on subject 4. We will need to check for patterns on the performance of the models for the different

	Accuracy	Recall class 0	Recall class 1	Recall class 2
TimeAggNet	**0.716**	**0.729**	0.717	0.703
TimeGraphNet	0.712	0.724	0.684	0.728
TimeGraphNetAdap	0.679	0.659	0.694	0.684
DeepTimeGraphNet	0.686	0.655	0.697	0.706
DeepTimeGraphNetAdap	0.686	0.651	**0.731**	0.676
SimpleTimeGraphNet	0.673	0.653	0.628	0.738
SimpleTimeGraphNetAdap	0.705	0.678	0.718	0.719

	Precision class 0	Precision class 1	Precision class 2
TimeAggNet	0.717	**0.724**	**0.710**
TimeGraphNet	0.737	0.716	0.690
TimeGraphNetAdap	0.687	0.712	0.652
DeepTimeGraphNet	0.711	0.706	0.657
DeepTimeGraphNetAdap	0.723	0.697	0.659
SimpleTimeGraphNet	0.674	0.696	0.676
SimpleTimeGraphNetAdap	**0.740**	0.705	0.678

	F1 class 0	F1 class 1	F1 class 2
TimeAggNet	0.722	**0.720**	0.706
TimeGraphNet	**0.728**	0.698	**0.707**
TimeGraphNetAdap	0.670	0.700	0.664
DeepTimeGraphNet	0.679	0.698	0.678
DeepTimeGraphNetAdap	0.683	0.709	0.660
SimpleTimeGraphNet	0.659	0.650	0.700
SimpleTimeGraphNetAdap	0.706	0.711	0.696

TABLE 4.4: Average test metrics for the different models across subjects excluding subjects 1, 4, and 6. Highest values in bold.

subjects.

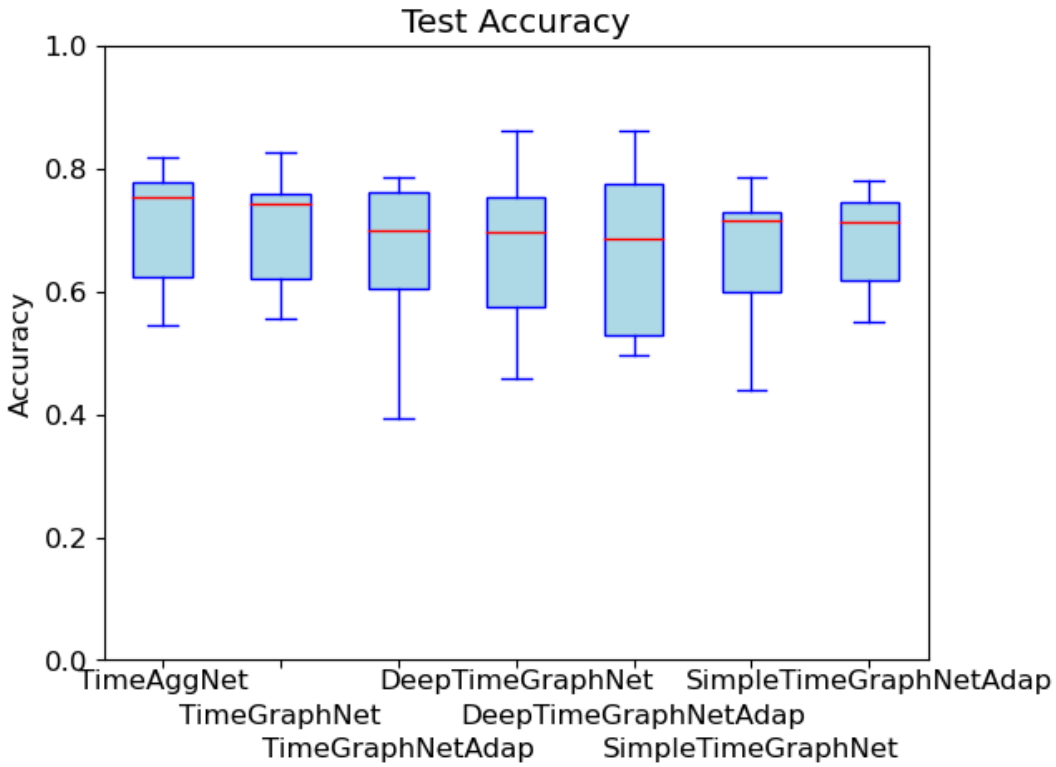

FIGURE 4.4: Test accuracy distributions for the different models across all subjects.

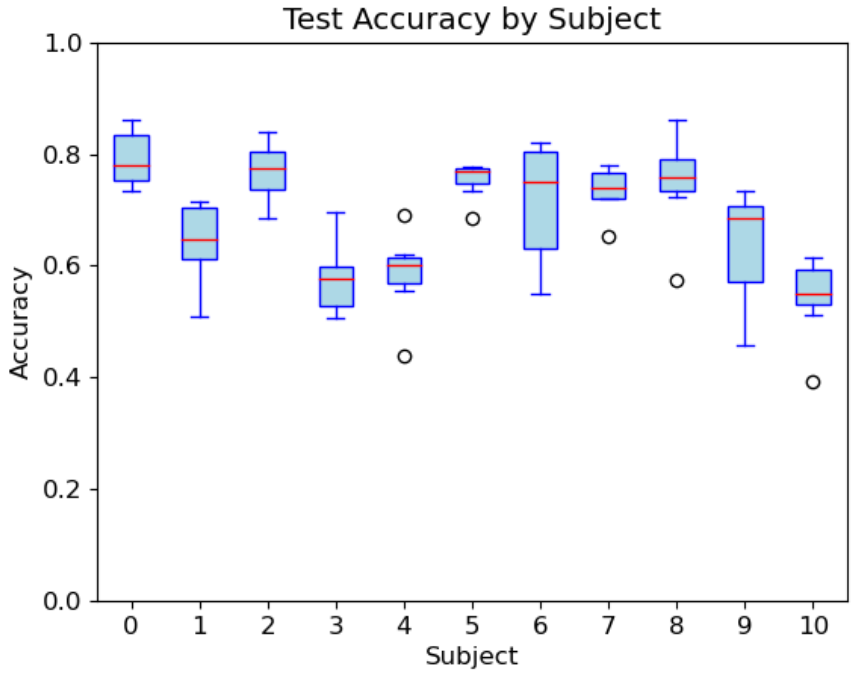

FIGURE 4.5: Accuracy distributions of all models grouped by subject.

On Figure 4.5 we see how the accuracies obtained for each subject are quite distinct. The variability in the test metrics between subjects could come from many sources, maybe the motivation state was not as strongly elicited on all of them, or the models do not generalise well among the subjects or some of the subjects show specific confounders, variability in electrode positioning or even each brain. For the case of how effectively each motivation state was generated on each subject, we can refer to the performance of each subject on the tasks. We can confirm that this motivation state was induced across subjects, however the way this motivation is translated into the state of the brain may be quite different from one subject to the other, this social pressure will not have the same effect on the brain. Even considering this, the performance on some of the subjects (0, 2, 5, and 8) is remarkably high, indicating that they are able to identify with high accuracy the resulting effect on the brain state for at least some of the participants.

The convention and standards of the field would require for the realisation of multiple random train-test splits or train-validation splits. In our case, time was a limiting factor [2] so only one instance of each model (except on subject 0) was realised, thus the error-bars have been calculated using a normal approximation (see section 3.4). On Figure 4.7 we have a comparison of the error-bars obtained with 1 and 4 repetitions for different random seeds in the train test split. For some cases the error-bars with a single repetition overestimate the error and for other it is underestimated. This serves us to illustrate that calculating the error-bars with a single repetition is not that far from a more precise calculation.

If we now focus on the effects of the adaptive modification of the adjacency matrix, we can observe on Figure 4.8 that there is no direct improvement of the resulting metrics against the default fully connected graph representation. In order to have a deeper look we plotted the accuracies of each default-modified pair over the different subjects obtaining Figure 4.9, 4.10, and 4.11. The three show no clear improvement from the adaptive modification, except with SimpleTimeGraphNet that shows some improvement with some of the subjects. As explained before, we observe a similar pattern, the adaptive modifications are more complex models with more parameters and may be subject to the disadvantages that difficult their training.

Taking into account the results from training (Table 4.2) and test (Table 4.3) we can state that the models did not overfit the training data, as the variation from training to test is relatively small.

[2] Training all models over the whole data with a 80% training data took more than 60 h, provided the models converge adequately which was not the case. Models that needed more repetitions to converge correctly added extra running time of around 30 h.

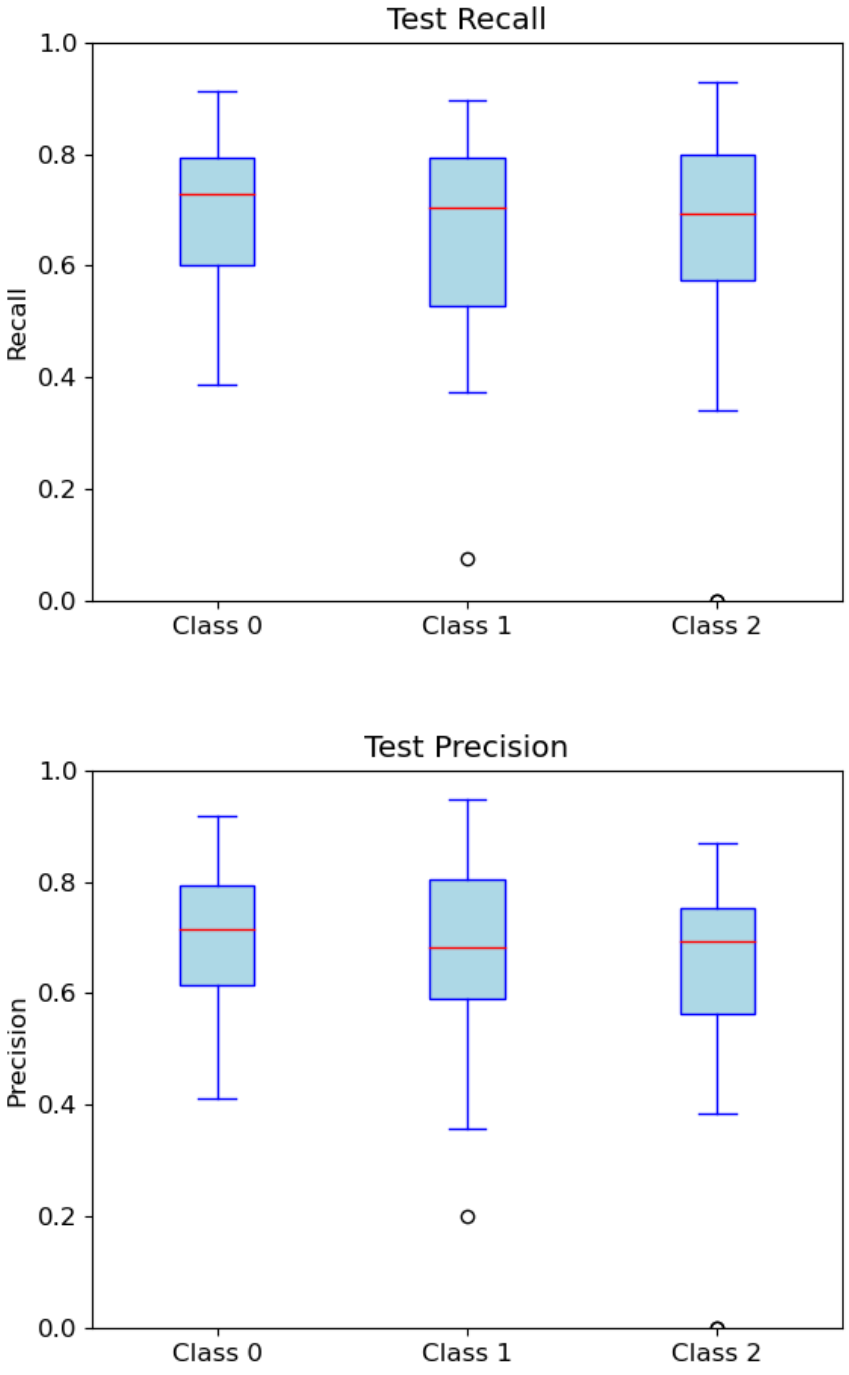

FIGURE 4.6: Recall and precision test metrics grouped by class for all models over all subjects.

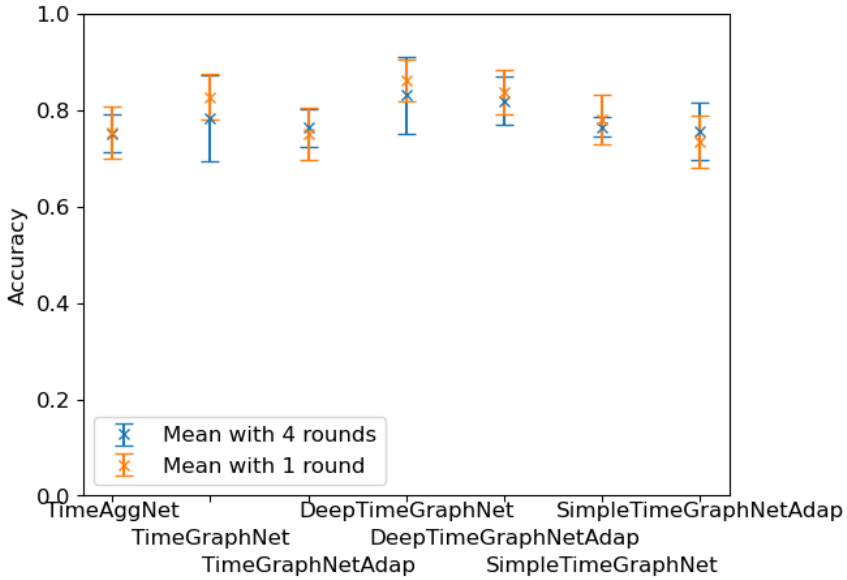

FIGURE 4.7: Average accuracies for subject 0 with a single and 4 repetitions for all models.

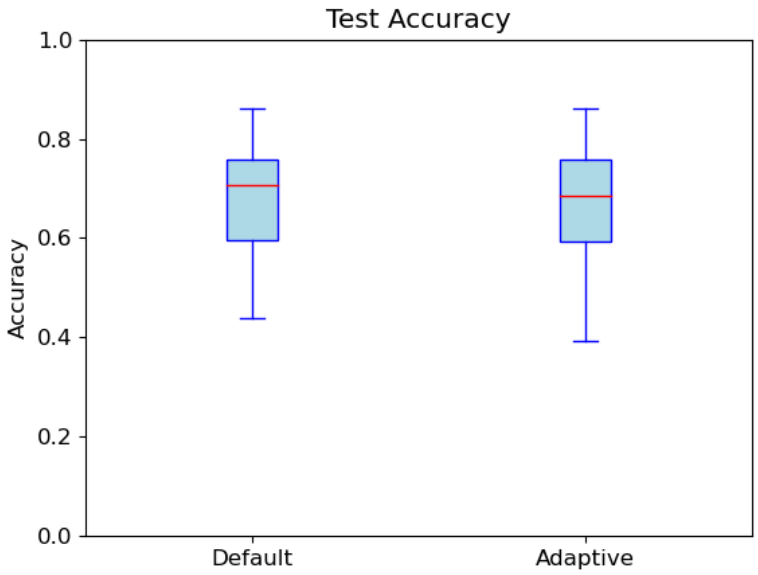

FIGURE 4.8: Test accuracy distribution over all graph models and subjects grouped into default models and adaptive modifications.

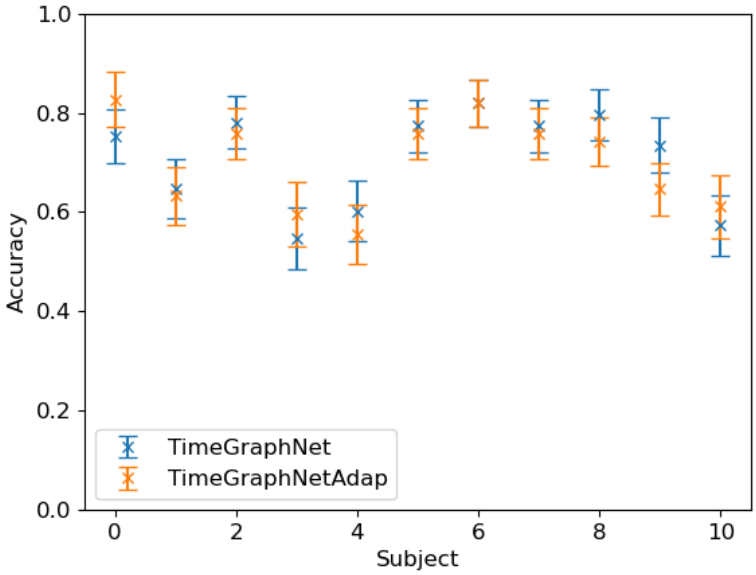

FIGURE 4.9: Test accuracies for the TimeGraphNet and TimeGraphNetAdap models on the different subjects.

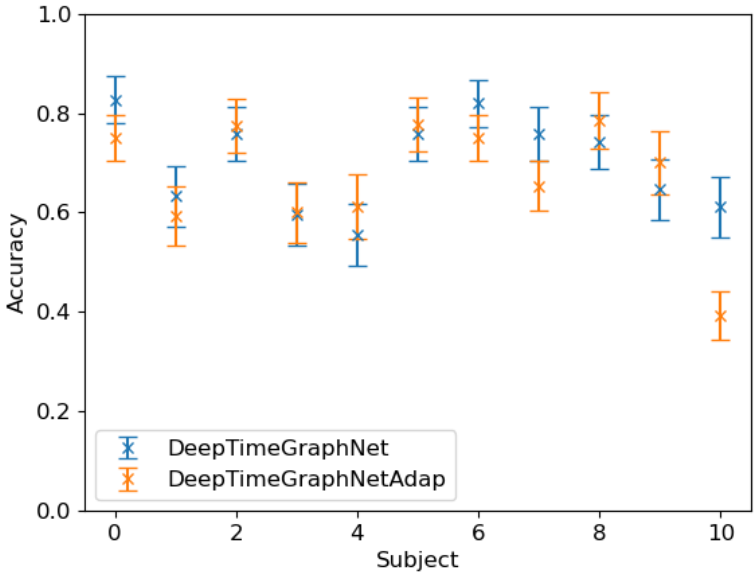

FIGURE 4.10: Test accuracies for the DeepTimeGraphNet and DeepTimeGraphNetAdap models on the different subjects.

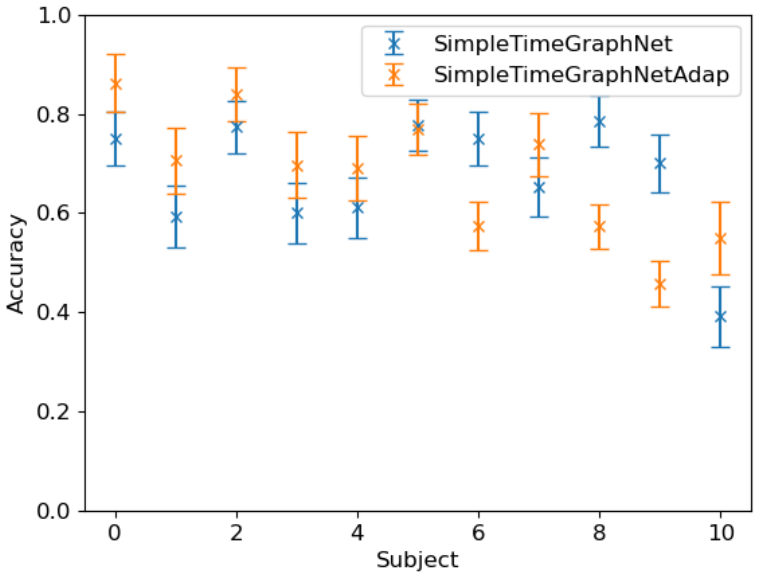

FIGURE 4.11: Test accuracies for the SimpleTimeGraphNet and SimpleTimeGraphNetAdap models on the different subjects.

4.3 Comparison with previous study

In Cos, Deco, and Gilson, Unpublished they perform a series of experiments trying to separate the different motivation states. In their approach they use different frequency bands along two classical machine learning classifiers. On average across subjects, the test accuracy they achieve is around 80% on the highest frequency band they filtered (γ[40-80Hz]). This is summarized in Figure 4.12 where the scores for the multinomial linear regression (MLR) and a 1-nearest-neighbor (1NN) classifiers are shown.

At first it could seem like our models performed worse than those presented in Cos, Deco, and Gilson, Unpublished, however, our models have been trained on the unfiltered signal. This means that our models where able to automatically extract the relevant information from the original signal. Moreover, on some of the subjects DeepTimeGraphNet had remarkably high performance above the presented on Cos, Deco, and Gilson, Unpublished. This patterns where some subjects are easier to classify than others appears on the original analysis as well.

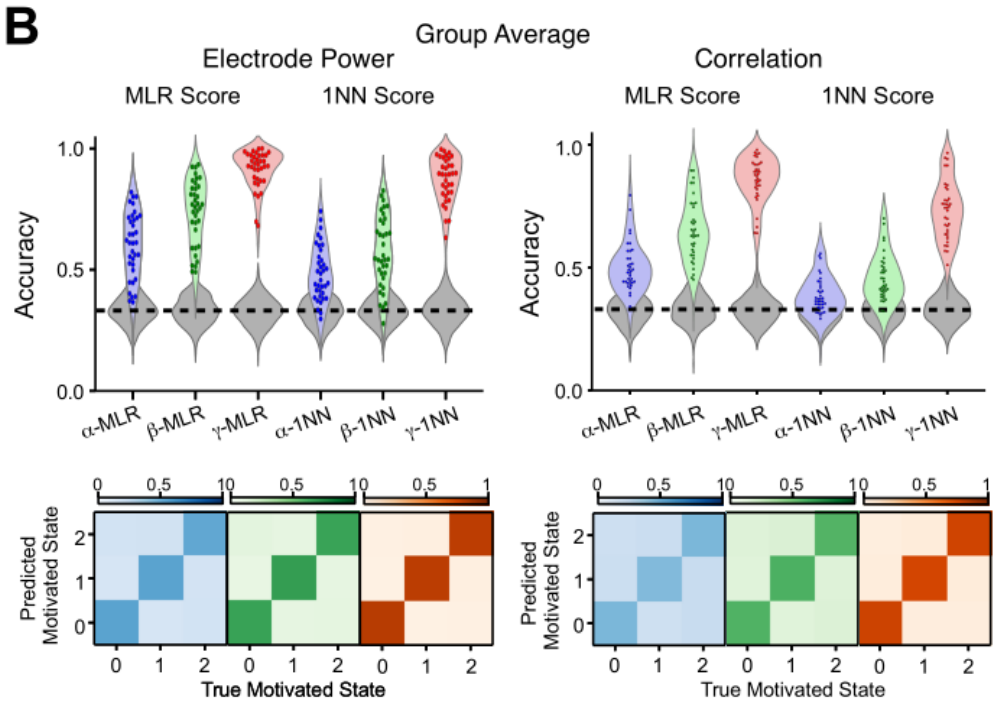

FIGURE 4.12: Figure from Cos, Deco, and Gilson, Unpublished, showing the accuracy as a function of the set of best hand extracted features for each frequency band filtered; for the multinomial linear regression (MLR) and a 1-nearest-neighbor (1NN) classifiers.

Further comparison would need to train and test our models on the frequency filtered signal, facilitating the task and allowing for further inspection of their behaviour.

Chapter 5

Conclusions

5.1 Summary

From the data exploration we have that three of the subjects have unbalanced classes, and that the missing channels across the trials do not follow any pattern.

The training of the different models shows the best average performance to be that of TimeAggNet. However the more complex models (DeepTimeGraphNet and its adaptive modification) showed more variability achieving the highest accuracies for some subjects while not converging adequately for others. The main source of this variability, along with fitting issues, might be the different ways the social pressure affects each subject and their brain state; for our models, some are easier to classify than others.

The average test results again show better metrics for TimeAggNet (mean test accuracy of 0.716) but with a deeper look at the distributions, more complex models have better results but more variability within subjects, and again DeepTimeGraphNet and its adaptive modification show some of the best accuracies (80-90% on test). Again, if we look a the distribution of accuracies per subject, we find some to be significantly harder for our models than others. When looking at the effects of the previous class unbalance, we see it negatively affected the recall and precision of the underrepresented classes. However, for the rest of the subjects, and on average, there was no obvious bias. It appears the adaptive modifications did not show better results than the default models. And finally, comparing training and test metrics, we can affirm there was no overfitting.

Given the results of Cos, Deco, and Gilson, Unpublished on the EEG classification, we find lower though not so distant average accuracies. However, they were using filtered signals, meaning that our models were capable of extracting the more relevant features. Further comparison should be done now, training and testing the models on the filtered signal.

All in all, we can conclude:

i) Our models can automatically extract relevant features from the unfiltered signal for the motivation state classification task.

ii) The accuracy of our models varies more across subjects the more complex they are, but allow for some of the highest test accuracies.

iii) The encoding of the motivation state on the EEG signals is not similar enough across subjects for our models to obtain consistent performance.

iv) The self-adaptive modification we used offers no clear advantage over a fully connected graph representation.

5.2 Limitations

The first set of limitations comes from the lack generality of the models. For instance, each subject had a model of different size input and newly initialised trained on their data. The input size from subject to subject varies in size and information because they do not have the same active channels. This becomes a limitation because our graph embeddings consist of the concatenation of the node features. Moreover EGG have low spatial resolution, that is, even following a standardised pattern to allocate the electrodes within the cap, and then on the scalp, the precision as to which region of the brain is being monitored is not high. In addition to this, each brain contains a lot of variability, i.e., there are many degrees of freedom. This is the main limitation in trying to apply transfer learning in this field. For example, even if we achieve a similar motivational state on the experimental setting the encoding of this motivation state on the brain is quite different.

A second set of limitations are specific to the concrete development of our models. Given the previous variability among subjects, during the different iterations and model development, we introduced an inductive bias. Due to the big amount of data, and the long time the models take to converge, the models were first tested on subject 0 during development. This subject has the best results however it shows that similar results can only be achieved (in the same conditions) with some of the subjects.

The last sets of limitations stems from the need of further proving of the models. As mentioned before, the frequency bands filtering was not explored. Further validation of the models could be improved by more receptions of the train test partition.

5.3 Future work

First, we could focus on limitations regarding class unbalance, by experimenting with weighted losses to correct the issue. We could explore the frequency domain, either by transforming the signals into frequencies at discretised time windows or filtering the original signal to identify which is most useful for the task, and better compare performance with the results from Cos, Deco, and Gilson, Unpublished. We could palliate the convergence problems encountered by DeepTimeGraphNet and its adaptive modification by tuning the hyper-parameters of the optimiser and experimenting with the initialization of the models. We can also modify the graph convolution layers, trying for other types and implementations and incorporating other more effective self-adaptive strategies. Moreover, we could build an adjacency matrix based on the most relevant correlations between nodes found in Cos, Deco, and Gilson, Unpublished. We could also focus on the generalisation limitations exploring node pooling layers readout functions and other strategies to have a graph embedding suitable for transfer learning and application across subjects.

Appendix A

Tables of Metrics

	Accuracy	Recall class 0	Recall class 1	Recall class 2
TimeAggNet	0.75 ±0.04	0.81	0.72	0.74
TimeGraphNet	0.78 ±0.09	0.82	0.75	0.80
TimeGraphNetAdap	0.76 ±0.04	0.79	0.74	0.76
DeepTimeGraphNet	0.83 ±0.08	0.87	0.80	0.82
DeepTimeGraphNetAdap	0.82 ±0.05	0.84	0.78	0.84
SimpleTimeGraphNet	0.77 ±0.02	0.76	0.76	0.77
SimpleTimeGraphNetAdap	0.76 ±0.06	0.80	0.78	0.70

	Precision class 0	Precision class 1	Precision class 2
TimeAggNet	0.77	0.89	0.61
TimeGraphNet	0.79	0.84	0.72
TimeGraphNetAdap	0.77	0.82	0.71
DeepTimeGraphNet	0.81	0.91	0.78
DeepTimeGraphNetAdap	0.79	0.89	0.80
SimpleTimeGraphNet	0.79	0.80	0.72
SimpleTimeGraphNetAdap	0.76	0.78	0.73

	F1 class 0	F1 class 1	F1 class 2
TimeAggNet	0.79	0.79	0.67
TimeGraphNet	0.80	0.79	0.75
TimeGraphNetAdap	0.78	0.77	0.73
DeepTimeGraphNet	0.84	0.85	0.80
DeepTimeGraphNetAdap	0.81	0.83	0.82
SimpleTimeGraphNet	0.77	0.78	0.74
SimpleTimeGraphNetAdap	0.78	0.78	0.71

TABLE A.1: Average test results for subject 0 with 4 repetitions.

	Accuracy	Recall class 0	Recall class 1	Recall class 2
TimeAggNet	0.754	0.784	0.805	0.671
TimeGraphNet	0.827	0.830	0.874	0.776
TimeGraphNetAdap	0.750	0.693	0.805	0.753
DeepTimeGraphNet	0.862	0.852	0.897	0.835
DeepTimeGraphNetAdap	0.838	0.807	0.839	0.871
SimpleTimeGraphNet	0.781	0.682	0.793	0.871
SimpleTimeGraphNetAdap	0.735	0.773	0.828	0.600

	Precision class 0	Precision class 1	Precision class 2
TimeAggNet	0.767	0.769	0.722
TimeGraphNet	0.859	0.817	0.805
TimeGraphNetAdap	0.772	0.824	0.667
DeepTimeGraphNet	0.862	0.907	0.816
DeepTimeGraphNetAdap	0.855	0.948	0.740
SimpleTimeGraphNet	0.870	0.831	0.685
SimpleTimeGraphNetAdap	0.716	0.783	0.699

	F1 class 0	F1 class 1	F1 class 2
TimeAggNet	0.775	0.787	0.695
TimeGraphNet	0.844	0.844	0.790
TimeGraphNetAdap	0.731	0.814	0.707
DeepTimeGraphNet	0.857	0.902	0.826
DeepTimeGraphNetAdap	0.830	0.890	0.800
SimpleTimeGraphNet	0.764	0.812	0.767
SimpleTimeGraphNetAdap	0.743	0.804	0.646

TABLE A.2: Test results for subject 0.

	Accuracy	Recall class 0	Recall class 1	Recall class 2
TimeAggNet	0.647	0.756	0.524	0.627
TimeGraphNet	0.632	0.793	0.444	0.610
TimeGraphNetAdap	0.593	0.744	0.556	0.424
DeepTimeGraphNet	0.706	0.768	0.730	0.593
DeepTimeGraphNetAdap	0.510	0.610	0.508	0.373
SimpleTimeGraphNet	0.701	0.793	0.651	0.627
SimpleTimeGraphNetAdap	0.716	0.866	0.667	0.559

	Precision class 0	Precision class 1	Precision class 2
TimeAggNet	0.785	0.611	0.521
TimeGraphNet	0.739	0.560	0.545
TimeGraphNetAdap	0.735	0.530	0.455
DeepTimeGraphNet	0.818	0.667	0.603
DeepTimeGraphNetAdap	0.633	0.464	0.393
SimpleTimeGraphNet	0.802	0.683	0.587
SimpleTimeGraphNetAdap	0.798	0.656	0.647

	F1 class 0	F1 class 1	F1 class 2
TimeAggNet	0.770	0.564	0.569
TimeGraphNet	0.765	0.496	0.576
TimeGraphNetAdap	0.739	0.543	0.439
DeepTimeGraphNet	0.792	0.697	0.598
DeepTimeGraphNetAdap	0.621	0.485	0.383
SimpleTimeGraphNet	0.798	0.667	0.607
SimpleTimeGraphNetAdap	0.830	0.661	0.600

TABLE A.3: Test results for subject 1.

	Accuracy	Recall class 0	Recall class 1	Recall class 2
TimeAggNet	0.781	0.795	0.724	0.824
TimeGraphNet	0.758	0.727	0.747	0.800
TimeGraphNetAdap	0.773	0.830	0.644	0.847
DeepTimeGraphNet	0.838	0.864	0.747	0.906
DeepTimeGraphNetAdap	0.827	0.761	0.793	0.929
SimpleTimeGraphNet	0.715	0.830	0.494	0.824
SimpleTimeGraphNetAdap	0.685	0.636	0.621	0.800

	Precision class 0	Precision class 1	Precision class 2
TimeAggNet	0.795	0.768	0.778
TimeGraphNet	0.877	0.739	0.687
TimeGraphNetAdap	0.820	0.789	0.720
DeepTimeGraphNet	0.826	0.878	0.819
DeepTimeGraphNetAdap	0.893	0.863	0.752
SimpleTimeGraphNet	0.652	0.827	0.729
SimpleTimeGraphNetAdap	0.683	0.643	0.723

	F1 class 0	F1 class 1	F1 class 2
TimeAggNet	0.795	0.746	0.800
TimeGraphNet	0.795	0.743	0.739
TimeGraphNetAdap	0.825	0.709	0.778
DeepTimeGraphNet	0.844	0.807	0.860
DeepTimeGraphNetAdap	0.822	0.826	0.832
SimpleTimeGraphNet	0.730	0.619	0.773
SimpleTimeGraphNetAdap	0.659	0.632	0.760

TABLE A.4: Test results for subject 2.

	Accuracy	Recall class 0	Recall class 1	Recall class 2
TimeAggNet	0.546	0.545	0.529	0.565
TimeGraphNet	0.596	0.602	0.563	0.624
TimeGraphNetAdap	0.600	0.648	0.598	0.553
DeepTimeGraphNet	0.696	0.682	0.782	0.624
DeepTimeGraphNetAdap	0.508	0.443	0.609	0.471
SimpleTimeGraphNet	0.512	0.420	0.448	0.671
SimpleTimeGraphNetAdap	0.577	0.602	0.563	0.565

	Precision class 0	Precision class 1	Precision class 2
TimeAggNet	0.600	0.511	0.533
TimeGraphNet	0.646	0.590	0.558
TimeGraphNetAdap	0.564	0.642	0.603
DeepTimeGraphNet	0.723	0.654	0.726
DeepTimeGraphNetAdap	0.557	0.477	0.506
SimpleTimeGraphNet	0.536	0.506	0.500
SimpleTimeGraphNetAdap	0.654	0.544	0.539

	F1 class 0	F1 class 1	F1 class 2
TimeAggNet	0.571	0.520	0.549
TimeGraphNet	0.624	0.576	0.589
TimeGraphNetAdap	0.603	0.619	0.577
DeepTimeGraphNet	0.702	0.712	0.671
DeepTimeGraphNetAdap	0.494	0.535	0.488
SimpleTimeGraphNet	0.471	0.476	0.573
SimpleTimeGraphNetAdap	0.627	0.554	0.552

TABLE A.5: Test results for subject 3.

	Accuracy	Recall class 0	Recall class 1	Recall class 2
TimeAggNet	0.602	0.738	0.400	0.573
TimeGraphNet	0.556	0.487	0.425	0.667
TimeGraphNetAdap	0.611	0.588	0.375	0.729
DeepTimeGraphNet	0.690	0.738	0.450	0.750
DeepTimeGraphNetAdap	0.583	0.600	0.375	0.656
SimpleTimeGraphNet	0.440	0.475	0.075	0.562
SimpleTimeGraphNetAdap	0.620	0.625	0.425	0.698

	Precision class 0	Precision class 1	Precision class 2
TimeAggNet	0.536	0.485	0.753
TimeGraphNet	0.481	0.447	0.660
TimeGraphNetAdap	0.618	0.500	0.636
DeepTimeGraphNet	0.615	0.621	0.791
DeepTimeGraphNetAdap	0.558	0.385	0.692
SimpleTimeGraphNet	0.413	0.200	0.495
SimpleTimeGraphNetAdap	0.556	0.459	0.753

	F1 class 0	F1 class 1	F1 class 2
TimeAggNet	0.621	0.438	0.651
TimeGraphNet	0.484	0.436	0.663
TimeGraphNetAdap	0.603	0.429	0.680
DeepTimeGraphNet	0.670	0.522	0.770
DeepTimeGraphNetAdap	0.578	0.380	0.674
SimpleTimeGraphNet	0.442	0.109	0.527
SimpleTimeGraphNetAdap	0.588	0.442	0.724

TABLE A.6: Test results for subject 4.

	Accuracy	Recall class 0	Recall class 1	Recall class 2
TimeAggNet	0.773	0.807	0.782	0.729
TimeGraphNet	0.758	0.795	0.747	0.729
TimeGraphNetAdap	0.777	0.773	0.816	0.741
DeepTimeGraphNet	0.769	0.670	0.793	0.847
DeepTimeGraphNetAdap	0.685	0.602	0.736	0.718
SimpleTimeGraphNet	0.735	0.750	0.701	0.753
SimpleTimeGraphNetAdap	0.777	0.761	0.770	0.800

	Precision class 0	Precision class 1	Precision class 2
TimeAggNet	0.755	0.840	0.729
TimeGraphNet	0.707	0.802	0.775
TimeGraphNetAdap	0.773	0.855	0.708
DeepTimeGraphNet	0.756	0.945	0.661
DeepTimeGraphNetAdap	0.654	0.790	0.622
SimpleTimeGraphNet	0.688	0.859	0.688
SimpleTimeGraphNetAdap	0.788	0.827	0.723

	F1 class 0	F1 class 1	F1 class 2
TimeAggNet	0.780	0.810	0.729
TimeGraphNet	0.749	0.774	0.752
TimeGraphNetAdap	0.773	0.835	0.724
DeepTimeGraphNet	0.711	0.862	0.742
DeepTimeGraphNetAdap	0.627	0.762	0.667
SimpleTimeGraphNet	0.717	0.772	0.719
SimpleTimeGraphNetAdap	0.775	0.798	0.760

TABLE A.7: Test results for subject 5.

	Accuracy	Recall class 0	Recall class 1	Recall class 2
TimeAggNet	0.819	0.838	0.802	0.818
TimeGraphNet	0.819	0.762	0.864	0.836
TimeGraphNetAdap	0.750	0.787	0.790	0.636
DeepTimeGraphNet	0.574	0.912	0.630	0.000
DeepTimeGraphNetAdap	0.685	0.787	0.704	0.509
SimpleTimeGraphNet	0.787	0.800	0.778	0.782
SimpleTimeGraphNetAdap	0.551	0.850	0.630	0.000

	Precision class 0	Precision class 1	Precision class 2
TimeAggNet	0.838	0.802	0.818
TimeGraphNet	0.792	0.814	0.868
TimeGraphNetAdap	0.708	0.800	0.745
DeepTimeGraphNet	0.471	0.864	0.000
DeepTimeGraphNetAdap	0.618	0.838	0.609
SimpleTimeGraphNet	0.790	0.829	0.729
SimpleTimeGraphNetAdap	0.511	0.622	0.000

	F1 class 0	F1 class 1	F1 class 2
TimeAggNet	0.838	0.802	0.818
TimeGraphNet	0.777	0.838	0.852
TimeGraphNetAdap	0.746	0.795	0.686
DeepTimeGraphNet	0.621	0.729	0.000
DeepTimeGraphNetAdap	0.692	0.765	0.554
SimpleTimeGraphNet	0.795	0.803	0.754
SimpleTimeGraphNetAdap	0.638	0.626	0.000

TABLE A.8: Test results for subject 6.

	Accuracy	Recall class 0	Recall class 1	Recall class 2
TimeAggNet	0.773	0.693	0.793	0.835
TimeGraphNet	0.758	0.750	0.655	0.871
TimeGraphNetAdap	0.654	0.545	0.816	0.600
DeepTimeGraphNet	0.738	0.568	0.793	0.859
DeepTimeGraphNetAdap	0.723	0.580	0.759	0.835
SimpleTimeGraphNet	0.719	0.773	0.494	0.894
SimpleTimeGraphNetAdap	0.781	0.682	0.793	0.871

	Precision class 0	Precision class 1	Precision class 2
TimeAggNet	0.782	0.726	0.816
TimeGraphNet	0.767	0.679	0.822
TimeGraphNetAdap	0.600	0.634	0.750
DeepTimeGraphNet	0.833	0.657	0.768
DeepTimeGraphNetAdap	0.785	0.647	0.763
SimpleTimeGraphNet	0.694	0.652	0.792
SimpleTimeGraphNetAdap	0.857	0.711	0.796

	F1 class 0	F1 class 1	F1 class 2
TimeAggNet	0.735	0.758	0.826
TimeGraphNet	0.759	0.667	0.846
TimeGraphNetAdap	0.571	0.714	0.667
DeepTimeGraphNet	0.676	0.719	0.811
DeepTimeGraphNetAdap	0.667	0.698	0.798
SimpleTimeGraphNet	0.731	0.562	0.840
SimpleTimeGraphNetAdap	0.759	0.750	0.831

TABLE A.9: Test results for subject 7.

	Accuracy	Recall class 0	Recall class 1	Recall class 2
TimeAggNet	0.796	0.830	0.770	0.788
TimeGraphNet	0.742	0.830	0.644	0.753
TimeGraphNetAdap	0.785	0.841	0.724	0.788
DeepTimeGraphNet	0.573	0.591	0.586	0.541
DeepTimeGraphNetAdap	0.862	0.875	0.839	0.871
SimpleTimeGraphNet	0.723	0.659	0.828	0.682
SimpleTimeGraphNetAdap	0.758	0.784	0.690	0.800

	Precision class 0	Precision class 1	Precision class 2
TimeAggNet	0.802	0.779	0.807
TimeGraphNet	0.716	0.789	0.736
TimeGraphNetAdap	0.771	0.797	0.788
DeepTimeGraphNet	0.650	0.537	0.541
DeepTimeGraphNetAdap	0.917	0.869	0.804
SimpleTimeGraphNet	0.817	0.615	0.806
SimpleTimeGraphNetAdap	0.812	0.723	0.739

	F1 class 0	F1 class 1	F1 class 2
TimeAggNet	0.816	0.775	0.798
TimeGraphNet	0.768	0.709	0.744
TimeGraphNetAdap	0.804	0.759	0.788
DeepTimeGraphNet	0.619	0.560	0.541
DeepTimeGraphNetAdap	0.895	0.854	0.836
SimpleTimeGraphNet	0.730	0.706	0.739
SimpleTimeGraphNetAdap	0.798	0.706	0.768

TABLE A.10: Test results for subject 8.

	Accuracy	Recall class 0	Recall class 1	Recall class 2
TimeAggNet	0.735	0.705	0.805	0.694
TimeGraphNet	0.646	0.534	0.736	0.671
TimeGraphNetAdap	0.700	0.557	0.770	0.776
DeepTimeGraphNet	0.458	0.466	0.471	0.435
DeepTimeGraphNetAdap	0.496	0.557	0.552	0.376
SimpleTimeGraphNet	0.685	0.659	0.828	0.565
SimpleTimeGraphNetAdap	0.712	0.580	0.805	0.753

	Precision class 0	Precision class 1	Precision class 2
TimeAggNet	0.674	0.805	0.728
TimeGraphNet	0.644	0.719	0.582
TimeGraphNetAdap	0.754	0.798	0.595
DeepTimeGraphNet	0.482	0.477	0.416
DeepTimeGraphNetAdap	0.516	0.490	0.478
SimpleTimeGraphNet	0.652	0.686	0.727
SimpleTimeGraphNetAdap	0.718	0.824	0.615

	F1 class 0	F1 class 1	F1 class 2
TimeAggNet	0.689	0.805	0.711
TimeGraphNet	0.584	0.727	0.623
TimeGraphNetAdap	0.641	0.784	0.673
DeepTimeGraphNet	0.474	0.474	0.425
DeepTimeGraphNetAdap	0.536	0.519	0.421
SimpleTimeGraphNet	0.655	0.750	0.636
SimpleTimeGraphNetAdap	0.642	0.814	0.677

TABLE A.11: Test results for subject 9.

	Accuracy	Recall class 0	Recall class 1	Recall class 2
TimeAggNet	0.573	0.670	0.529	0.518
TimeGraphNet	0.612	0.727	0.506	0.600
TimeGraphNetAdap	0.392	0.386	0.379	0.412
DeepTimeGraphNet	0.550	0.545	0.506	0.600
DeepTimeGraphNetAdap	0.550	0.580	0.724	0.341
SimpleTimeGraphNet	0.512	0.455	0.437	0.647
SimpleTimeGraphNetAdap	0.615	0.602	0.678	0.565

	Precision class 0	Precision class 1	Precision class 2
TimeAggNet	0.562	0.597	0.564
TimeGraphNet	0.681	0.595	0.554
TimeGraphNetAdap	0.442	0.359	0.385
DeepTimeGraphNet	0.558	0.595	0.510
DeepTimeGraphNetAdap	0.607	0.492	0.604
SimpleTimeGraphNet	0.488	0.594	0.482
SimpleTimeGraphNetAdap	0.688	0.584	0.585

	F1 class 0	F1 class 1	F1 class 2
TimeAggNet	0.611	0.561	0.540
TimeGraphNet	0.703	0.547	0.576
TimeGraphNetAdap	0.412	0.369	0.398
DeepTimeGraphNet	0.552	0.547	0.551
DeepTimeGraphNetAdap	0.593	0.586	0.436
SimpleTimeGraphNet	0.471	0.503	0.553
SimpleTimeGraphNetAdap	0.642	0.628	0.575

TABLE A.12: Test results for subject 10.

Subject	Active Channels	Total
1	0, 1, 2, 3, 4, 5, 6, 11, 14, 16, 18, 22, 23, 24, 27, 30, 31, 32, 33, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 46, 47, 48, 49, 50, 51, 52, 53, 54, 56, 57, 58, 59	42
2	0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 32, 33, 34, 35, 36, 37, 38, 39, 41, 42, 43, 44, 45, 46, 47, 50, 52, 53, 54, 56, 57, 58, 59	49
3	0, 1, 3, 4, 5, 6, 7, 8, 9, 12, 13, 14, 15, 16, 17, 18, 19, 22, 23, 24, 25, 26, 27, 28, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 49, 50, 51, 52, 53, 54, 56, 57, 58, 59	50
4	1, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 43, 44, 45, 46, 47, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59	53
5	8, 9, 12, 14, 15, 17, 18, 20, 21, 22, 23, 24, 27, 28, 29, 30, 31, 32, 33, 34, 36, 37, 38, 39, 40, 41, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59	43
6	0, 1, 4, 5, 6, 7, 11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25, 26, 27, 28, 32, 33, 35, 36, 37, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59	49
7	0, 1, 3, 4, 5, 6, 8, 14, 15, 16, 18, 19, 20, 22, 23, 24, 25, 27, 28, 29, 30, 32, 33, 35, 36, 37, 38, 39, 40, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 58, 59	45
8	6, 7, 17, 18, 22, 24, 25, 26, 27, 28, 33, 34, 35, 36, 37, 38, 39, 40, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59	35
9	0, 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 22, 23, 24, 25, 26, 27, 28, 29, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 54, 55, 56, 57, 58, 59	54
10	1, 2, 3, 4, 5, 7, 10, 12, 13, 16, 18, 22, 23, 25, 28, 29, 30, 32, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59	43
11	1, 4, 5, 7, 8, 9, 10, 12, 13, 14, 17, 18, 20, 21, 22, 25, 28, 29, 30, 31, 32, 33, 34, 36, 37, 38, 39, 40, 41, 42, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59	46

TABLE A.13: Lists of used channels per subject.

Appendix B

Training Metrics

	Loss	Accuracy	Recall	Precision	F1
TimeAggNet	0.090	0.813	0.813	0.814	0.813
TimeGraphNet	0.084	0.834	0.834	0.835	0.834
TimeGraphNetAdap	0.084	0.834	0.834	0.835	0.834
DeepTimeGraphNet	0.051	0.899	0.899	0.899	0.898
DeepTimeGraphNetAdap	0.051	0.899	0.899	0.899	0.898
SimpleTimeGraphNet	0.073	0.847	0.847	0.848	0.847
SimpleTimeGraphNetAdap	0.073	0.847	0.847	0.848	0.847

TABLE B.1: Train results for subject 0.

	Loss	Accuracy	Recall	Precision	F1
TimeAggNet	0.124	0.768	0.765	0.765	0.764
TimeGraphNet	0.126	0.724	0.721	0.723	0.720
TimeGraphNetAdap	0.126	0.724	0.721	0.723	0.720
DeepTimeGraphNet	0.124	0.758	0.754	0.754	0.754
DeepTimeGraphNetAdap	0.124	0.758	0.754	0.754	0.754
SimpleTimeGraphNet	0.132	0.724	0.722	0.723	0.722
SimpleTimeGraphNetAdap	0.132	0.724	0.722	0.723	0.722

TABLE B.2: Train results for subject 1.

	Loss	Accuracy	Recall	Precision	F1
TimeAggNet	0.081	0.848	0.848	0.848	0.848
TimeGraphNet	0.089	0.841	0.841	0.843	0.841
TimeGraphNetAdap	0.089	0.841	0.841	0.843	0.841
DeepTimeGraphNet	0.062	0.886	0.886	0.886	0.886
DeepTimeGraphNetAdap	0.062	0.886	0.886	0.886	0.886
SimpleTimeGraphNet	0.120	0.759	0.759	0.759	0.757
SimpleTimeGraphNetAdap	0.120	0.759	0.759	0.759	0.757

TABLE B.3: Train results for subject 2.

	Loss	Accuracy	Recall	Precision	F1
TimeAggNet	0.154	0.669	0.669	0.671	0.669
TimeGraphNet	0.154	0.691	0.691	0.692	0.690
TimeGraphNetAdap	0.154	0.691	0.691	0.692	0.690
DeepTimeGraphNet	0.130	0.732	0.732	0.733	0.732
DeepTimeGraphNetAdap	0.130	0.732	0.732	0.733	0.732
SimpleTimeGraphNet	0.177	0.570	0.570	0.570	0.570
SimpleTimeGraphNetAdap	0.177	0.570	0.570	0.570	0.570

TABLE B.4: Train results for subject 3.

	Loss	Accuracy	Recall	Precision	F1
TimeAggNet	0.157	0.656	0.629	0.657	0.637
TimeGraphNet	0.158	0.661	0.623	0.674	0.635
TimeGraphNetAdap	0.158	0.661	0.623	0.674	0.635
DeepTimeGraphNet	0.147	0.705	0.688	0.707	0.695
DeepTimeGraphNetAdap	0.147	0.705	0.688	0.707	0.695
SimpleTimeGraphNet	0.190	0.552	0.501	0.549	0.504
SimpleTimeGraphNetAdap	0.190	0.552	0.501	0.549	0.504

TABLE B.5: Train results for subject 4.

	Loss	Accuracy	Recall	Precision	F1
TimeAggNet	0.087	0.824	0.824	0.826	0.825
TimeGraphNet	0.104	0.777	0.777	0.780	0.778
TimeGraphNetAdap	0.104	0.777	0.777	0.780	0.778
DeepTimeGraphNet	0.095	0.801	0.801	0.806	0.803
DeepTimeGraphNetAdap	0.095	0.801	0.801	0.806	0.803
SimpleTimeGraphNet	0.083	0.833	0.833	0.836	0.834
SimpleTimeGraphNetAdap	0.083	0.833	0.833	0.836	0.834

TABLE B.6: Train results for subject 5.

	Loss	Accuracy	Recall	Precision	F1
TimeAggNet	0.089	0.832	0.836	0.823	0.829
TimeGraphNet	0.077	0.860	0.859	0.859	0.859
TimeGraphNetAdap	0.077	0.860	0.859	0.859	0.859
DeepTimeGraphNet	0.131	0.650	0.536	0.802	0.493
DeepTimeGraphNetAdap	0.131	0.650	0.536	0.802	0.493
SimpleTimeGraphNet	0.101	0.795	0.783	0.784	0.783
SimpleTimeGraphNetAdap	0.101	0.795	0.783	0.784	0.783

TABLE B.7: Train results for subject 6.

	Loss	Accuracy	Recall	Precision	F1
TimeAggNet	0.087	0.824	0.824	0.824	0.824
TimeGraphNet	0.090	0.818	0.817	0.817	0.817
TimeGraphNetAdap	0.090	0.818	0.817	0.817	0.817
DeepTimeGraphNet	0.103	0.775	0.775	0.776	0.775
DeepTimeGraphNetAdap	0.103	0.775	0.775	0.776	0.775
SimpleTimeGraphNet	0.095	0.786	0.786	0.785	0.785
SimpleTimeGraphNetAdap	0.095	0.786	0.786	0.785	0.785

TABLE B.8: Train results for subject 7.

	Loss	Accuracy	Recall	Precision	F1
TimeAggNet	0.080	0.843	0.843	0.843	0.843
TimeGraphNet	0.084	0.839	0.839	0.840	0.839
TimeGraphNetAdap	0.084	0.839	0.839	0.840	0.839
DeepTimeGraphNet	0.138	0.711	0.711	0.715	0.712
DeepTimeGraphNetAdap	0.138	0.711	0.711	0.715	0.712
SimpleTimeGraphNet	0.124	0.736	0.736	0.741	0.736
SimpleTimeGraphNetAdap	0.124	0.736	0.736	0.741	0.736

TABLE B.9: Train results for subject 8.

	Loss	Accuracy	Recall	Precision	F1
TimeAggNet	0.124	0.730	0.730	0.726	0.727
TimeGraphNet	0.160	0.637	0.637	0.636	0.636
TimeGraphNetAdap	0.160	0.637	0.637	0.636	0.636
DeepTimeGraphNet	0.185	0.542	0.543	0.540	0.541
DeepTimeGraphNetAdap	0.185	0.542	0.543	0.540	0.541
SimpleTimeGraphNet	0.151	0.672	0.672	0.672	0.671
SimpleTimeGraphNetAdap	0.151	0.672	0.672	0.672	0.671

TABLE B.10: Train results for subject 9.

	Loss	Accuracy	Recall	Precision	F1
TimeAggNet	0.162	0.643	0.643	0.644	0.641
TimeGraphNet	0.167	0.632	0.632	0.633	0.631
TimeGraphNetAdap	0.167	0.632	0.632	0.633	0.631
DeepTimeGraphNet	0.175	0.588	0.588	0.592	0.586
DeepTimeGraphNetAdap	0.175	0.588	0.588	0.592	0.586
SimpleTimeGraphNet	0.176	0.594	0.593	0.600	0.588
SimpleTimeGraphNetAdap	0.176	0.594	0.593	0.600	0.588

TABLE B.11: Train results for subject 10.

Appendix C

Code

For the implementation of the models we used *PyTorch* (Paszke et al., 2019) and *PyTorch Geometric* (Fey and Lenssen, 2019). For plotting we used *Matplotlib* (Hunter, 2007), and *scikit-learn* (Pedregosa et al., 2011) for the metrics and train test splits. The code for this study can be found at: `https://github.com/45truc/TFM`.

Bibliography

Bronstein, Michael M. et al. (2021). "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges". In: *CoRR* abs/2104.13478. arXiv: 2104.13478. URL: https://arxiv.org/abs/2104.13478.

Cos, Ignasi, Gustavo Deco, and Matthieu Gilson (Unpublished). "Behavioural and Cortical Correlates of Social Motivation: Decision-Making between Precision-Reachings". unpublished.

Demir, Andac et al. (2021). "EEG-GNN: Graph Neural Networks for Classification of Electroencephalogram (EEG) Signals". In: *CoRR* abs/2106.09135. arXiv: 2106.09135. URL: https://arxiv.org/abs/2106.09135.

Fey, Matthias and Jan Eric Lenssen (2019). "Fast Graph Representation Learning with PyTorch Geometric". In: *CoRR* abs/1903.02428. arXiv: 1903.02428. URL: http://arxiv.org/abs/1903.02428.

Hamilton, William L., Rex Ying, and Jure Leskovec (2017). "Inductive Representation Learning on Large Graphs". In: *CoRR* abs/1706.02216. arXiv: 1706.02216. URL: http://arxiv.org/abs/1706.02216.

Hunter, J. D. (2007). "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3, pp. 90–95. DOI: 10.1109/MCSE.2007.55.

Jiang, Dejun et al. (2021). "Could graph neural networks learn better molecular representation for drug discovery? A comparison study of descriptor-based and graph-based models". In: *Journal of Cheminformatics* 13.1. DOI: 10.1186/s13321-020-00479-8.

Kipf, Thomas N. and Max Welling (2016). "Semi-Supervised Classification with Graph Convolutional Networks". In: *CoRR* abs/1609.02907. arXiv: 1609.02907. URL: http://arxiv.org/abs/1609.02907.

Klepl, Dominik, Min Wu, and Fei He (Oct. 2023). "Graph Neural Network-based EEG Classification: A Survey". In: *arXiv e-prints*, arXiv:2310.02152, arXiv:2310.02152. DOI: 10.48550/arXiv.2310.02152. arXiv: 2310.02152 [q-bio.NC].

LeNail, Alexander (2019). "NN-SVG: Publication-Ready Neural Network Architecture Schematics". In: *Journal of Open Source Software* 4.33, p. 747. DOI: 10.21105/joss.00747. URL: https://doi.org/10.21105/joss.00747.

Morselli Gysi, Deisy et al. (May 2021). "Network medicine framework for identifying drug-repurposing opportunities for COVID-19". In: *Proceedings of the National Academy of Science* 118.19, e2025581118, e2025581118. DOI: 10.1073/pnas.2025581118. arXiv: 2004.07229 [q-bio.MN].

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *CoRR* abs/1912.01703. arXiv: 1912.01703. URL: http://arxiv.org/abs/1912.01703.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Raschka, Sebastian (2018). "Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning". In: *CoRR* abs/1811.12808. arXiv: 1811.12808. URL: http://arxiv.org/abs/1811.12808.

Veličković, Petar (Apr. 2023). "Everything is connected: Graph neural networks". In: *Current Opinion in Structural Biology* 79, p. 102538. ISSN: 0959-440X. DOI: 10.1016/j.sbi.2023.102538. URL: http://dx.doi.org/10.1016/j.sbi.2023.102538.

Wikipedia contributors (2024). *Electroencephalography — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Electroencephalography&oldid=1228461979. [Online; accessed 17-June-2024].

Wu, Zonghan et al. (May 2019). "Graph WaveNet for Deep Spatial-Temporal Graph Modeling". In: *arXiv e-prints*, arXiv:1906.00121, arXiv:1906.00121. DOI: 10.48550/arXiv.1906.00121. arXiv: 1906.00121 [cs.LG].

Zhou, Jie et al. (2018). "Graph Neural Networks: A Review of Methods and Applications". In: *CoRR* abs/1812.08434. arXiv: 1812.08434. URL: http://arxiv.org/abs/1812.08434.