EEG data encoding for classification with deep learning models

Emil Højrup Vinther

Supervisor: Paolo Burelli
Co-Supervisor: Laurits Dixen

Thesis submitted to the IT University of Copenhagen

June 2024

Abstract

Being able to correctly classify electroencephalogram (EEG) data is of importance for diagnosing neural disorders such as Alzheimer's and Epilepsy. It is also an important research question in the field of Brain Computer Interface (BCI) applications, with the promise of helping impaired individuals control prosthetic devices. EEG is a non-invasive method for capturing brain activity with electrodes placed on the skull. Convolutional Neural Networks (CNNs) are the most popular deep learning architecture for decoding the EEG signal into a classification scheme that can be used by BCI devices. These models are good at capturing local features of the EEG signal but have no inherent way of capturing long term dependencies in the data. The transformer model architecture is at the core of the most popular Large Language Models (LLMs) such as GPT and BERT, but have also shown good performance in image classification tasks. The self-attention mechanism is good at capturing long term dependencies in a data sequence and therefore potentially a better alternative to CNN models for EEG classification. It is, however, not immediately clear how to transform EEG data into a representation that can be used effectively by a transformer model. In this thesis I investigate strategies for encoding EEG data into a representation that can effectively be used for EEG classification. The three major strategies I investigate are: 1. using the concept of permutation patterns from the field of permutation entropy to encode EEG data into "tokens" from a finite permutation pattern vocabulary, 2. combining a transformer model with a Graph Convolutional Network (GCN) module and 3. collapsing the temporal and spatial convolutions in the ShallowFBCSPNet and Conformer models into one spatiotemporal convolution. For the first two strategies I was not able to create well performing model implementations, but by collapsing the the temporal and spatial convolutions into a spatiotemporal convolution I find the ShallowFBCSPNet and Conformer models to perform at least as well or better than the original models, with fewer parameters and a less complex model architecture.

Contents

List of Figures

Introduction

The code for this project can be found at: `https://github.itu.dk/evin/evin_thesis`

1.1 EEG classification

Electroencephalography (EEG) is a method for measuring electrical activity from the brain using electrodes placed on the skull. By placing electrodes on the skull in accordance with a standardised positioning system it is possible to measure the electrical field generated primarily by the synchronised activity of pyramidal neurons in cortical brain regions (the outer layer on top of the cerebrum) (see fig 1.1). Being able to correctly classify and find patterns in this data is of importance for diagnosing neural disorders such as Alzheimer's and Epilepsy and an important research question in the field of Brain Computer Interface (BCI) applications. A BCI is a communication and control system that provides interaction between a user and a computer, based on the brain's electrical signals when doing a specific task. These patterns can be recorded with EEG signals and used in a BCI system [17].

1.2 EEG and machine learning

Several different traditional machine learning models have been used for EEG classification. Among the most successful methods of extracting features from the EEG data is by using Common Spatial Patterns (CSP). This can then be combined with Linear Discriminant Analysis (LDA) or Support Vector Machines (SVM) for classification [14]. These models have shown good performance on motor imagery classification on tasks such as determining whether a subject thinks about moving their left hand, right hand, tongue etc. These models have, however, been outperformed by deep learning models such as EEGNet, ConvNet and ShallowFBCSPNet. These deep learning models use a Convolutional Neural Network (CNN) architecture and have shown very good temporal

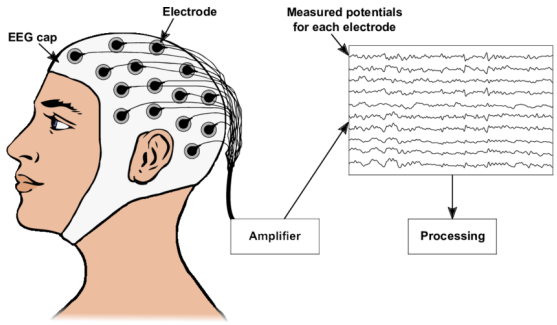

Figure 1.1: Sketch of an EEG recording session. The EEG cap is placed on the skull of a test subject. Brainwaves are captured by the electrodes on the cap, amplified and recorded on a computer. Figure taken from [8].

feature detection but lack the ability to detect long term dependencies which is crucial for time-series data like EEG [12].

1.3 Transformer models

The transformer architecture for deep learning models has gathered a lot of attention since the architecture was first proposed in 2017 [13]. This is especially the case in the context of Natural Language Processing (NLP). The transformer architecture is at the core of all of the most popular large language models like GPT and BERT. Transformers have also been applied outside the original domain of NLP. Most notably in computer vision where the Vision Transformer (ViT) has shown performance comparable to the more traditional Convolutional Neural Network (CNN) architectures [4]. This raises the question whether transformer models can be used for EEG classification as well. And, if so, how to best harness the power of the transformer architecture.

1.4 Project outline

The goal of this thesis is to find an efficient and effective way of encoding EEG data into a representation that can be used by a transformer model. To find such an encoding I have used two different approaches: 1. Exploring new encoding approaches, and 2. Seeking to improve an existing encoding strategy. The methodologies and results of both approaches are detailed in this report, even though only the second approach has been successful. The following is a brief outline of the methods and results from both approaches.

1.4.1 Exploring new encoding approaches for EEG data

Encoding with permutation patterns

EEG is multivariate time series data, meaning that it has both a temporal and a spatial dimension (the placement of the electrodes on the skull). To best capture the temporal information I experimented with a type of manual feature extraction based on the concept of permutation patterns. The idea behind this approach was that it would make it possible to encode the essence of the temporal characteristics of a segment of EEG data into a compact format with a finite vocabulary, which is what a transformer model expects.

Combining graph convolutions and transformers

I also experimented with combining a graph convolution module with a transformer. The idea behind this approach was that a graph convolutional module would be able to capture the non-euclidean spatial relationships between the channels and the transformer could handle the temporal interactions. This model was also slightly better than a model consisting only of a transformer, but still performed far from state of the art models.

1.4.2 Improving existing models

My third strategy was to modify the separate temporal and spatial convolution CNN architecture used by both the ShallowFBCSPNet and Conformer models. I recreated the ShallowFBCSPNet model collapsing the spatial and temporal layer combined into one. This almost halved the model size, made it conceptually simpler and achieved the same performance as the implementation with separate temporal and spatial convolutional layers. I also implemented the Conformer model using both the traditional two layers approach and the approach of combining the layers into one. This also resulted in a simpler model with fewer parameters and better performance for the model using a combined spatiotemporal layer.

1.5 Structure of the thesis

The thesis is structured as follows: I first conduct a literature review giving some background context and a more in depth explanation of EEG data and which machine learning models are usually used for EEG classification. In this chapter I explain how CNNs, transformers and graph convolutional networks work and give in depth examples of how these architectures have been used for EEG classification and what their strengths and weaknesses are.

After that I introduce the datasets I have used for testing the models I have developed. This is followed by a description of the methods I have used. Then the results from running the models on the datasets are presented, followed by a discussion of these results, which other paths could have been taken and

the implication this work might have on further research in the field of EEG classification with deep learning models.

Literature review

2.1 Overview of the literature review

In the following I give a more detailed explanation of what EEG data is and present some of the major strategies for EEG classification using machine learning. I give an introduction to the current state of the research on EEG classification, explain how CNNs, transformers and graph convolutions work, and give concrete examples of how these architectures have been used for EEG classification. I argue that traditional machine learning models are lightweight and can have a decent performance level, but are incapable of capturing complex relationships in the EEG data. CNNs are the most used deep learning architecture for EEG classification. I exemplify this approach by explaining the ShallowFBCSPNet model architecture.

CNNs generally performs better than traditional machine learning models but are not able to capture long term dependencies in the data. I then argue that transformer models have the potential to capture these long term dependencies and explain how the S3T model uses transformers for EEG classification.

Different model architectures can also be combined and I therefore also present the Conformer model, which combines CNNs and transformers. This model has state of the art performance and uses the ShallowFBCSPNet architecture for encoding the EEG data into a format that the transformer can effectively utilise.

Transformers might be good at capturing long term dependencies but graph convolutions might be better at capturing the non-euclidean spatial relationships between the electrodes. I therefore give an explanation of how graph convolutions work and how they might be useful for EEG classification.

2.2 Introduction to EEG data

EEG data is collected by electrodes placed on the skull. It consists of measurements of the voltage received by each electrode sampled a number of times a second depending on the sampling rate. EEG data is multivariate time series

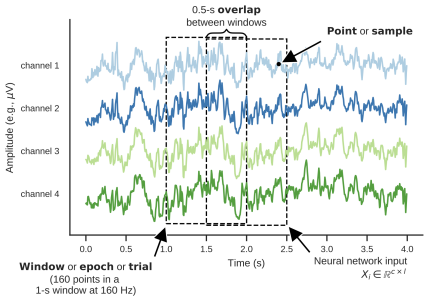

Figure 2.1: EEG data example. EEG data consists of a series of voltage measurements across several channels sampled at a given sampling frequency and split up into trials (also called windows or epochs). Picture taken from [9].

data, which means that it consists of time series data from multiple sources. A typical EEG dataset consists of a series of trials T where a subject is performing a task specified by the experiment. Each trial has a number of channels C that each contain the series of voltage samples S from the corresponding electrode. The shape of the EEG data is then (T, C, S) (see fig 2.1).

As such EEG is a type of multivariate time series data that has both a spatial and a temporal dimension [15]. EEG is easy to use and cheap. It has good temporal resolution but poor spatial resolution compared to other methods, and a low signal to noise ratio. Therefore preprocessing steps are often needed to minimise the effect of non-physiological noise and physiological artefacts such as eye movement. The typical pipeline for machine learning classification of EEG data is to preprocess the data, then use a feature extraction method to find the most important information in the signal and, lastly, a classification step that takes in the extracted features and outputs a classification label [9].

2.3 Status of machine learning models for EEG classification

2.3.1 Traditional models

The goal of using machine learning (ML) for EEG data is to identify brain patterns related to a specific activity without having to rely on traditional statistical methods. Because EEG data is chaotic and non-stationary it is necessary to have an efficient feature extraction approach. One of the most used and effective methods is based on common spatial patterns (CSP). CSP is a technique that transforms the EEG signals into a new space. This is done using spatial filtering techniques where the variance of one group is optimised while the variance of a second group is reduced [9]. For the classification step support vector machines (SVM) and linear discriminant analysis (LDA) are among the most common techniques. LDA transforms high-dimensional data into low-

dimensional data. LDA is computationally efficient and the combination of CSP for feature extraction and LDA for classification has good performance for many EEG classification tasks [9].

2.3.2 Deep learning models

The development of deep learning models for BCI applications has undergone a significant improvement in recent years due to a number of high-quality EEG datasets having been made publicly available. Where a linear model such as CSP or LDA sets up a hyperplane that separates the feature space into regions corresponding to different classes or conditions, deep learning (DL) models are able to capture complex non-linear dependencies in the data. DL is a subset of ML that focuses on models based on many-layered neural networks [18]. The most used DL architectures for EEG classification are CNNs but in recent years long short-term memory (LSTM), recurrent neural networks (RNN) and autoencoders (AE) have also been implemented [5]. To understand these models it is crucial to understand their building blocks. I therefore start with an introduction to convolutions, self-attention, graphs, the ideas behind them and how they are used in models like CNNs, transformers and Graph Neural Netowrks (GNN).

2.4 CNNs

Image classification poses a problem for classical fully connected networks because such models would grow extremely large even for small images. Having a weight for each pixel in the image means that the model size grows quadratically with an increase in the pixel size. CNNs were developed to address this challenge. Some of the advantages of using convolutional layers is that they reduce the number of input nodes, tolerate small shifts in where important features are in the input and that they take advantage of the correlations in complex images.

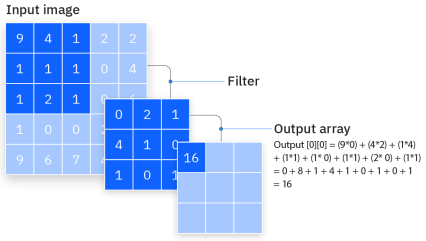

Figure 2.2: Convolution operation performed on an image of size (5×5) *with kernel (filter) dimension* (3×3)*. The resulting feature map (output array) is calculated as the sum of the pixel values multiplied with their corresponding kernel values. Image taken from [6].*

2.4.1 Convolutions

The key idea of convolutional layers is to connect patches of the input image to neurons in a hidden layer. The way this is done is by sliding a series of kernels (also called filters) k with dimensionality (m, n) across the image. An output feature map is created by multiplying the pixel values covered with by the kernel with the corresponding kernel value and summing up these values (see fig 2.2).

The mathematical formula for the cross-correlation operation (which is what most convolutional layer use in practice) for the output feature map O at a given location (i, j) is

$$O(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n)$$

where I is the image, K is the kernel and (m, n) are the kernel dimensions. The kernel does not have to be square and in the context of EEG data a kernel with dimensions $(1, w)$ would effectively be a temporal convolution only considering data with a window length of w in the time dimension for creating the feature map. Correspondingly a kernel with dimensions $(channels, 1)$ would result in a spatial convolution that only takes data from all the electrodes at a given timepoint into account.

The hyperparameter stride determines how much the kernel moves each step. A stride of $(2, 1)$ would mean that the kernel moves two pixels horizontally and one pixel vertically in the context of images. The kernel values are learnt during model training which means that each kernel can learn different low level features of the data.

2.4.2 Pooling

Another popular operation in CNNs is pooling, which is used to further downsample the feature map data. Similar to a convolution layer a pooling layer slides a kernel across the image with a given stride. The most common operations is to either select the max value from the feature map under the kernel region (max pooling) or the average value (average pooling) as the output to a new feature map. In essence the pooling operation select the spots in the input where the kernel best matches the input. The purpose of pooling is to reduce the dimensionality of the data (for computational efficiency) and to achieve translation invariance by making the model less sensitive to small changes in the input.

2.4.3 The ShallowFBCSPNet model

Schirrmeister et al. [11] have combined multiple convolution and pooling layers to create a deep learning EEG classification model named ShallowFBCSPNet. This model is inspired by the Filter Bank Common Spatial Patterns (FBCSP) model which combines CSP with filter banks which is an array of bandpass filters that separate the input signal into multiple components. The FBCSP

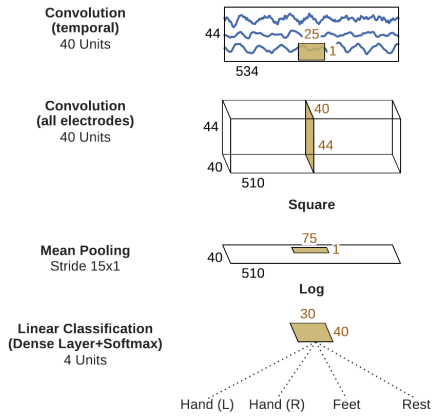

Figure 2.3: The ShallowFBCSPNet architecture. The model consists of two convolutional layers: a temporal convolution and a spatial convolution, a mean pooling layer and a linear classification layer. The temporal convolution layer has $k = 40$ kernels of shape (25,1), meaning that it only covers one channel at a time resulting in k feature maps. The spatial convolution has one kernel with shape $(channels, 1)$ which collapses the channel dimension into 1. After that mean pooling is applied to compress the data along the temporal dimension and a linear classification layer converts the feature vectors into classification predictions. Figure taken from [11].

model won the original BCI Competition IV dataset 2a [1] with an accuracy of 68% but is outperformed by the ShallowFBCSPNet model with an accuracy of 73.7%.

The ShallowFBCSPNet model architecture consists of two convolutional layers, a mean pooling layer and a linear classification layer (see fig 2.3). The authors call the first layer a temporal layer. In this layer k kernels move across the temporal dimension with a kernel of shape (25,1) and a stride of 1. They call this a temporal convolution since the kernel only covers one channel at a time effectively only considering the temporal dimension. The size of the input data is multiplied by the number of kernels $k = 40$ resulting in k feature maps that represent the features the given kernel captures from the data. In the second convolutional layer only spatial dimension is covered by the kernel, which has the shape $(channels, 1)$ and a stride of 1. This layer only has one kernel and therefore effectively collapses the spatial dimension into 1 [11].

[1] I use this as one of the datasets for testing the different models and preprocessing steps I implemented. A detailed description of the dataset is given in the Datasets section.

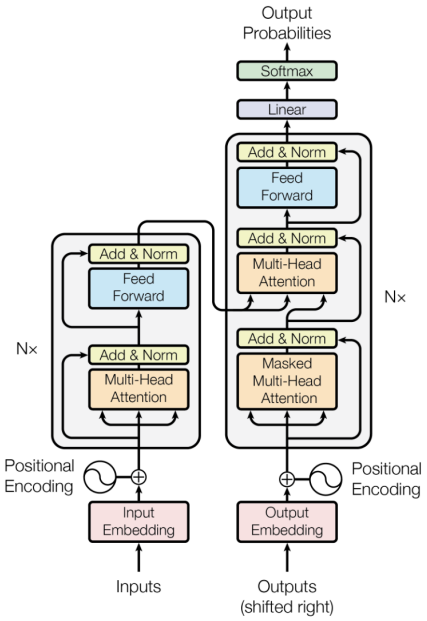

Figure 2.4: The original transformer architecture for translation tasks. The model consists of a encoder part (left) and a decoder part (right). The encoder transforms an input sequence of subword tokens into a continuous representation and the decoder then transforms this continuous representation into a new output sequence. Many different alternative architectures have been proposed and in this thesis I mainly focus on the multi-head attention mechanism which is at the heart of the model architecture. Image taken from [13].

2.5 Transformers

Where CNNs were originally developed for image classification before the architecture was applied for EEG classification, the transformer model architecture was developed for use in Natural Language Processing (NLP). The goal of the original transformer model was to translate a sequence of words in one language into a sequence of words in another language. To handle this task the model has an encoder-decoder structure, where the encoder transforms an input sequence of subword tokens into a continuous representation and the decoder then transforms this continuous representation into a new output sequence [13]. Since the original paper came out many different architectures have been proposed such as encoder-only, encoder-encoder and decoder-only models with different strengths and applications, but they all have multi-head attention at the core of the model (see fig 2.4).

Self-attention (also called scaled dot-product attention) is an operation that captures dependencies and relationships within the input sequence, and multi-

head attention is simply the idea of concatenating multiple self-attention layers that run in parallel.

The self-attention operations do not work directly on the input sequence of words, but requires the words are first broken down into subword tokens. These tokens are then transformed into token embedding vectors, which makes it possible for two tokens with similar semantic meaning to be positioned close to each other in the multidimensional embedding space. The token embeddings can either be pre-learned or be learned along with the other model parameters. The idea of self-attention is to adjust the initial semantic embeddings into contextual embeddings that encode a richer contextual meaning of the tokens based on the tokens' relationship with the other tokens in the input sequence. In other words the self-attention mechanism calculates which tokens are relevant to updating the embedding of the other tokens in the input sequence [18].

To update the embeddings of the tokens a Query (Q), Key (K) and Values (V) matrix is derived from the embedded token matrix by a linear transformation with different weights for the Q, K and V matrices. Q, K and V are calculated as $Q = XW^Q, K = XW^K, V = XW^V$, where W^Q, W^K, W^V are the learned weight matrices. In other words the embedded tokens are multiplied with W^Q, W^K and W^V matrices that consist of learnable parameters creating three new matrices (Q, K, V) with lower dimensionality than the input embedding matrix X.

The similarity between the token and any other token (including itself) is calculated by taking the dot product between the query and key vectors. The values of this operation between two tokens can be thought of as measuring how much the tokens attend to each other, with larger values meaning more attention. These values are then transformed into a weighted distribution by applying the Softmax function to the values resulting in an attention pattern. The grid of all possible dot products between Q and K can be represented as QK^T. For numerical stability this grid is often divided by the square root of the dimensionality of the key query space. The attention pattern is therefore calculated as

$$Softmax(\frac{QK^T}{\sqrt{d_k}})$$

These attention weights serve as coefficients for the values (V), which carry the actual information content of each token. Self-attention is then calculated as

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V$$

By taking a weighted sum of the value vectors based on the attention weights, the self-attention mechanism creates a new contextually updated embedding for each token (see fig 2.5).

2.5.1 Transformers for EEG classification

Several deep learning architectures that use the core ideas of transformer models for EEG data classification have been proposed. Common among them is that

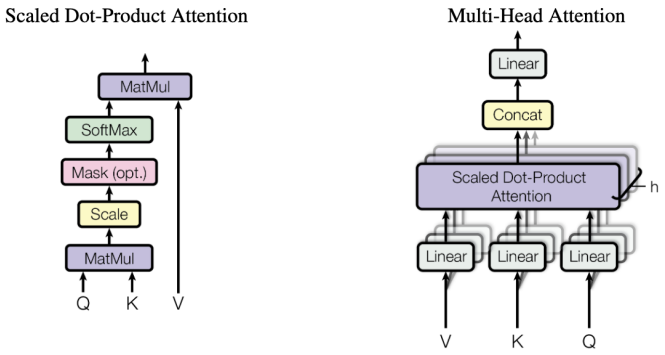

Figure 2.5: Scaled dot-product attention (left) and multi-head attention (right). Scaled dot-product attention (also called self-attention) is calculated as $Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V$*. The masking operation is optional and not covered here. Multi-head attention consists of concatenating multiple self-attention layers that run in parallel. The feedforward linear layers are typically placed after the concatenation of the attention heads. Figure taken from [13].*

they are not simply applying a transformer encoder layer directly on the EEG data but using one or more encoder layers before the self attention layer. This is done to transform the data into a manageable format for the transformer. In the following I give an overview of some of the best performing deep learning model architectures for EEG classification. I explain their model architectures and discuss how they attempt to deal with the challenges EEG data pose for transformer models.

2.5.2 Spatiotemporal Tiny Transformer (S3T)

The authors of "Transformer-based Spatial-Temporal Feature Learning for EEG Decoding" [16] justify their use of transformer models for EEG classification on the basis of a critique of CNNs and RNNs: CNNs are very dependent on the selection of kernels. A too large kernel hinders the models ability to extract deep features of the data, and a too small kernel makes it difficult for the individual neuron to incorporate relevant context. RNNs on the other hand are capable of extracting long term dependencies but only acts on the previous memory and the present. Not the future, which is also important in EEG data.

2.5.3 The S3T model architecture

Instead of CNNs and RNNs the authors leverage the power of transformer models for EEG classification. They are, however, not relying solely on the transformer model architecture. Instead they use CSP for feature extraction. This

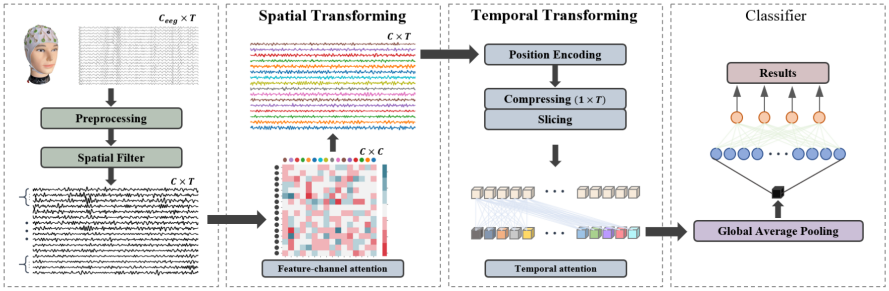

Figure 2.6: The spatiotemporal Tiny Transformer (S3T) model architecture. After a preprocessing step the EEG data is filtered using CSP and multple one-versus-rest tests. The feature channel attention module then weights the importance of the channels. Then the data is compressed and sliced along the time dimension and then sent through the temporal attention module and a fully connected layer for classification. Figure taken from [16].

feature extraction step is important because the non-invasive nature of EEG introduces a lot of noise. In the model the authors propose CSP is first used to find the most distinguishing features of the EEG data. Then the output of multiple one-versus-rest test on these feature vectors are stacked to form the input to the first transformer module. The first of two transformer modules is a feature-channel attention block that gives weights to different channels ensuring that the model gives more attention to the most relevant channels and ignore others. The output from this module is transformed using positional encoding and then compressed into one channel dimension $(1 \times T)$ and divided into many small slices in the time dimension. Then the data is sent through a second transformer module and then a fully connected layer for classification (see fig 2.6).

2.5.4 Experimental results for S3T

The model was tested on the BNCI Competition IV dataset 2a and 2b. Dataset 2a consists of motor imagery tasks covering the imagination of moving the left hand or the right hand, both feet or the tongue. Dataset 2b only contains the the right and left hand tasks [2]. The authors report an average validation accuracy of 91.3% on dataset 2a and an average validation accuracy of 84.2% on dataset 2b. This makes the models performance comparable with state of the art models but with less parameters. The model is, however, sensitive to the kernel size of the positional encoding and the slice size in the temporal slicing [16].

[2]This is the dataset I have used for testing my models and I give a detailed explanation of it in the Dataset section.

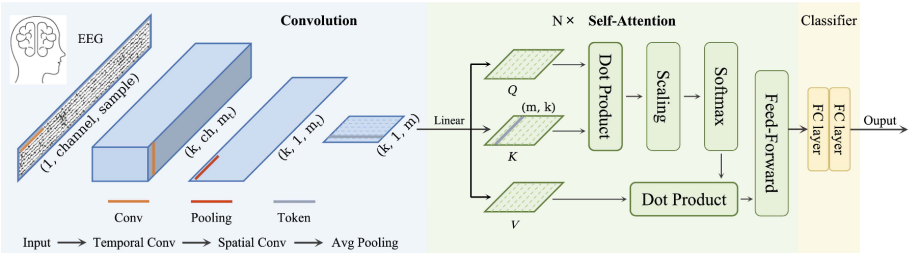

Figure 2.7: The conformer model architecture. The model consists of three modules: a convolutional, a transformer and a classifier module. The convolutional module transforms the EEG data into a series of feature vectors which are treated as embedded tokens by the transformer module. This module consists of N heads of self-attention that tries to find long term dependencies in the data. The output is sent to the classifier module which consists of two fully connected layers that transform the data into label predictions. Figure taken from [12].

2.6 Combining CNNs and transformers - The Conformer model

In the paper "EEG Conformer: Convolutional Transformer for EEG Decoding and Visualization" [12] Song et al. propose a model architecture that combines CNNs and transformers. They argue that this combination can take advantage of both types of models' advantages and compensate for their respective weaknesses for EEG data prediction. CNNs are good at learning local features but lack the ability to capture long term dependencies. Transformers, on the other hand, are excellent at capturing long term dependencies but not as effective as CNNs in capturing local features.

2.6.1 The Conformer model architecture

The model they propose consists of three major components: a convolutional module, a transformer module and a classifier module (see fig 2.7). The convolutional module contains two convolutional layers that perform a temporal and a spatial convolution similar to the convolutional layers in the ShallowFBCSPNet model. Instead of mean pooling as in the ShallowFBCSPNet this model applies average pooling along the temporal dimension to further compress the data. The effect of the convolutional module is that a new feature vector is created for each sample in each channel in the temporal convolution. The channel dimension is then collapsed into the feature vector. The shape of the output from the convolutional layer is then $(k, 1, m)$ where m is the temporal dimension compressed by the first convolutional layer and the average pooling layer, and k is the number of filters in the first convolutional layer. The feature vectors for multiple samples are then combined into an average. The result is a series of m feature vectors that are treated as embedded tokens.

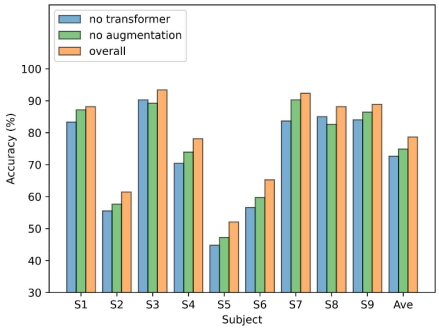

Figure 2.8: Ablation study on the Conformer model. The study reveals that both data augmentation and the transformer module have a non trivial positive impact on the performance of the Conformer model. The ablation study is done on the BNCI Competition IV dataset 2a consisting of four motor imagery tasks: the imagination of moving the left hand, the right hand, both feet or the tongue. Figure taken from [12].

These tokens are then the input to the transformer module which is a standard implementation of the transformer architecture having N self attention heads without positional encoding. The output of this module is sent to the classifier module. Here two fully connected layers transform the data into label prediction probabilities for the c classes in the dataset.

2.6.2 Experimental results for the Conformer model

The authors report performance comparable with state of the art models. Among the results is an 84.63% average validation accuracy on the BNCI Competition IV dataset 2b. The good results are to an extent due to the authors use of data augmentation. To create more data for the model to train on they used segmentation and reconstruction in the time domain. Their ablation study (fig 2.8) where they recreate the experiment without data augmentation and without the transformer module respectively reveals that both data augmentation and the transformer module have a non trivial positive impact on the performance of the model. However it also reveals that the CNN module is by far the most important building block of their model [12].

2.7 Graph Neural Networks

Finding a good way of letting a model extract long term temporal dependencies is only one part of the problem of using the EEG data effectively. The other part is using the spatial data in a way that captures as much relevant information as possible. The interactions between the brain activity capture by different electrodes is non-euclidean, meaning that the voltage measured by one electrode

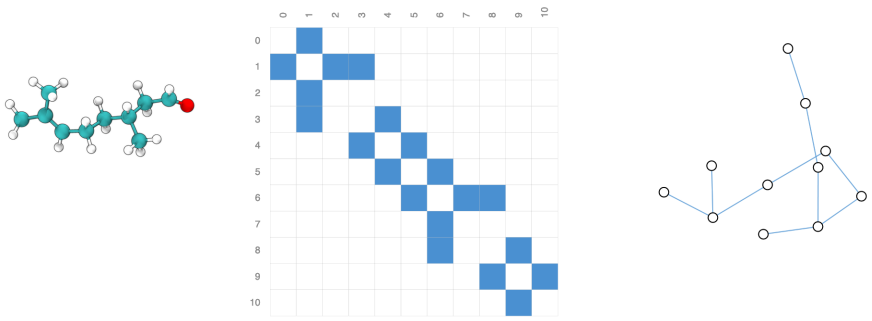

Figure 2.9: Graph representation of the Citronellal molecule. The molecule (left) can be represented with a graph (right) where each node represent an atom and the edges represent covalent bonds between atoms. The adjacency matrix representation of the graph (middle) has an entry for each edge between two nodes. Figure taken from [10].

might be more correlated with another electrode placed on the opposite side of the skull than with its nearest neighbours on the EEG cap. Furthermore the placement of the electrodes in relation to each other is not inherently captured by the raw EEG data. Two channels c_i and c_{i+1} are treated as being adjacent by a spatial convolution as described above but they may not be correlated in their output or close to each other in physical space. One way of handling this problem is by treating the spatial dimension as a graph [7].

Graph Neural Networks (GNNs) are gaining popularity in many different deep learning tasks. Graphs are a powerful way to represent many types of real world phenomena, since most real world objects are defined by their connections to other things. Graphs provide a way of representing objects and their interactions with other objects at the same time. A graph consists of nodes (the entities) and edges (the relations between entities). Molecular structure is one example of a structure that can effectively be modelled using a graph. Molecules consist of atoms and covalent bonds and in a graph representation the nodes represent atoms and the edges represent covalent bonds between the atoms. It is common to represent such a graph by an adjacency matrix by ordering the nodes in a $n_{nodes} \times n_{nodes}$ matrix and fill an entry in the matrix if the two nodes share an edge (see fig 2.9).

Stated formally a graph G is formulated as $G = (V, E)$ where V is the set of nodes and E is the set of edges. N is the number of nodes in a graph. The neighbourhood of a node $v \in V$ is the subset of nodes in the graph with an edge pointing to v. Let $e = (v, u) \in E$ denote an edge pointing from u to v, then the neighbourhood of node v is $N(v) = \{u \in V | (v, u) \in E\}$. The adjacency matrix A is a mathematical representation of a graph $A \in R^{N \times N}$ with $A_{ij} = c > 0$ if $(v_i, v_j) \in E$ and $A_{ij} = 0$ if $(v_i, v_j) \notin E$ [10].

2.7.1 Graph convolutions

Graph Convolutional Networks (GCNs) extend the idea of performing convolutions to graphs. Where a CNN extract the interactions between variables into a global hidden state, GCNs assume that the state of a node depends on the states of its neighbours in the graph [15]. GCNs operate on the graph via the adjacency matrix. Given a set of nodes and associated features a GCN perform neighbourhood aggregation to derive node embeddings. This operation is also called message passing. Similar to how a CNN aggregates information based on adjacent pixels, a GCN aggregates information based on the neighbouring nodes [7].

The neighbourhood relationships between the nodes can be represented using the graph Laplacian L, which is obtained bu subtracting the adjacency matrix A from the degree matrix D, $L = D - A$. The degree of a node v is the number of edges pointing to v, and the degree matrix D can be constructed as: $D_v = \sum_u A_{vu}$, where A_{vu} is the entry in A at row v, column u.

A GCN uses a filter Θ with learnable parameters similar to a kernel in CNNs, and applies it to the feature matrix X and graph Laplacian L. One very popular GCN operation is the Chebychev convolution defined as

$$X' = \sum_{k=1}^{K} Z^{(k)} \cdot \Theta^{(k)}$$

where X' is the transformed feature matrix, K is the number of k-hops, Θ is the filter matrix of learnable parameters and $Z^{(k)}$ is recursively defined as

$$Z^{(1)} = X$$

$$Z^{(2)} = \hat{L} \cdot X$$

$$Z^{(k)} = 2 \cdot \hat{L} \cdot Z^{(k-1)} - Z^{(k-2)}$$

and $\hat{L}$ is the normalised graph Laplacian $\hat{L} = \frac{2L}{\lambda_{max}} - I$ [3]. The hyperparameter K determines how many steps (hops) to be considered when updating a node. $k = 1$ corresponds to only including the nodes direct neighbours, $k = 2$ also includes the neighbours of the nodes neighbours etc [7]. In this way a GCN can learn the relationships between nodes in a graph, which can be applied too any data that can be represented as a graph.

2.7.2 The graph structure of EEG data

EEG data is a type of multivariate time series data, meaning that it consists of multiple interrelated variables that depend on time (the series of measurements from multiple electrodes). It is a basic assumption that the variables in multivariate time series data depend on one another as well as the variable's historical values. To exploit these latent spatial dependencies the channels in the EEG data can be seen as nodes in a graph and their hidden dependency relationships as edges in the graph [15].

Representing the EEG data this way might make it possible to capture the non-euclidean spatial interactions between channels by transforming the data with graph convolutions. The problem is that most multivariate time series data (EEG included) does not have an explicit graph structure, which is required by a graph convolution layer. Wu et al. therefore propose a graph learning layer that extracts a graph adjacency matrix adaptively based on the data [15]. This makes it possible to simultaneously learn the graph structure of the data and use this graph structure to extract spatial dependencies in the data into embeddings.

2.8 Summary of literature review

To correctly classify EEG data is a very complex task. This is both due to the complex nature of the source of the data (electrical signals from the brain), the amount of noise in the signal and the data structure of EEG data. Since EEG is a type of multivariate time series data, a good classification model should be able to capture both spatial and temporal features from the data. At the same time the model should not be overtly complicated in comparison with similar performing models. The model types I have described above each have their strengths and weaknesses for this task. Traditional ML models are generally simple and interpretable but are not able to capture complex patterns in the data. CNNs are the most popular type of deep learning models for EEG classification. They generally perform well but have no inbuilt way of capturing long term dependencies in the data or non-euclidean spatial relationships between channels. Transformer models have the potential to capture long term dependencies in the data, and graph convolutional models have the potential to capture the non-euclidean relationships between channels. In the following I explore some the possible ways of harnessing the power of these models for EEG classification.

Datasets

I have used the BNCI Competition IV dataset 2a and dataset 2b for evaluating the performance of my models [2]. The datasets are publicly available and provided by Graz University of Technology. They are designed to benchmark BCI EEG classification algorithms. The purpose of the datasets is to be able to train and benchmark models that can classify motor imagery. This means determining which body part the subject imagines moving. The datasets are among the most commononly used datasets for benchmarking EEG classification models.

3.1 The BNCI Competition IV 2a dataset

In dataset 2a the tasks were moving their left hand (class 1), right hand (class 2), both feet (class 3), and tongue (class 4) The EEG data was collected for 9 subjects each participating in two sessions (on separate days) each comprised of 6 runs with 48 trials per run. The subjects were sitting in front of a computer screen. At the beginning of a trial (t=0s), a fixation cross appeared on the screen. After two seconds (t=2s) an arrow pointing either left, right, up or down was shown corresponding to the classes left hand, right hand, tongue and feet respectively. The arrow stayed on the screen for 1.25 seconds. The subjects were tasked with carrying out the motor imagery task from the arrow appeared on screen until the fixation cross disappeared at t = 6 seconds. The dataset is balanced, having 288 trials per session, giving a total of $9 \times 288 \times 2 = 5.184$ trials.

The EEG data was recorded using 22 Ag/AgCl electrodes placed according to the international 10-20 system (see fig 3.1). The signals were recorded monopolarly with the left mastoid serving as reference and the right mastoid as ground. The data was collected with a sampling rate of 250Hz and bandpass-filtered between 0.5 Hz and 100 Hz. The sensitivity of the amplifier was set to 100 μV. An additional 50 Hz notch filter was enabled to suppress line noise.

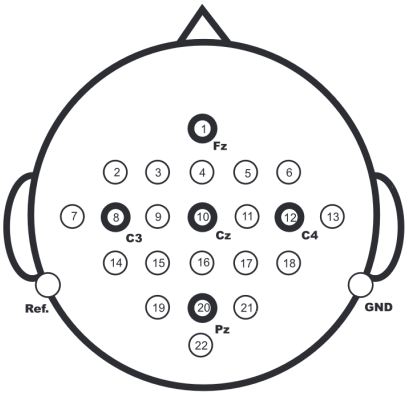

Figure 3.1: EEG electrode montage. The 22 Ag/AgCl electrodes were placed according to the international 20-10 system. The signals were recorded with the left mastoid serving as reference and the right mastoid as ground. Figure taken from [2].

3.2 The BNCI Competition IV 2b dataset

The BNCI Competition IV 2b dataset is recorded similar to dataset 2a but with only two classes of tasks (imagining moving the right or left arm). This dataset is also perfectly balanced, having 2.592 total trials with 1.296 trials per class. Both datasets are split up into a training and a test dataset based on the two sessions (day 1 and day 2), which means that half the data in each dataset is reserved for testing.

Methods

4.1 Overview of methods used

This section is divided into two: the first part details the methods I have used when trying to find a new approach to encoding EEG data for use in transformer models, the second part is an explanation of how I have combined the temporal and spatial convolutions in the ShallowFBCSPNet model architecture into one spatiotemporal convolution and used this to also modify the Conformer model architecture.

4.2 Finding a new approach to encoding EEG data

The work of trying to harness the power of transformers for EEG classification has been an iterative process and I have tried out multiple approaches during the process. In the following I will describe the four major approaches I have used and explain the reasoning behind implementing them. The four approaches are: 1, a transformer only benchmarking model, 2, preprocessing using permutation patterns, 3, preprocessing using a simplified version of permutation patterns and 4, combining GNNs and transformers. I also used a combination of CSP and LDA to get a baseline of what performance any deep learning model should be able to achieve on the datasets.

4.2.1 LDA + CSP

When benchmarking different deep learning models it is useful to explore the baseline performance of a traditional machine learning model. Traditional machine learning models are generally faster and simpler than deep learning models, so there is only reason to use a deep learning model if it outperforms traditional models. I chose a combination of CSP and LDA (as described in the literature review) as the baseline model. Them model was trained using k-fold cross validation (with $k = 10$) and 8 CSP components.

4.2.2 Transformer only model

To be able to evaluate the usefulness of the permutation pattern inspired preprocessing approaches described below, I first implemented a simple transformer only model. The model is based on a standard implementation of the transformer model as described above. One of the challenges of using transformers on EEG data is that a transformer model expects the input data to have the shape $(batch, sequence\_length, embedding)$. As described above EEG data has the shape $(batch, channels, samples)$. Of course the ordering of the dimensions does not matter as you can simply rearrange the data to have samples first and channels last $(batch, samples, channels)$. This would have the effect of treating the samples as a sequence of tokens and the channel dimension as the series of embedding vectors for each sample. One could argue that treating the channels as embedding vectors can be justified because they encode a spatial context for the sample. But the issue with this approach is that the embedding dimension in a traditional transformer implementation is supposed to be an encoding of the similarities between tokens in an abstract embedding space. In NLP one can either use pretrained embeddings or the embeddings can be learned along with the other model parameters. In either case the embeddings are an added feature dimension and not part of the original training data. One solution would be to flatten the channels and samples into one dimension $(channels \times samples)$. There are several downsides to this approach. One is that the spatial information is lost because the one trial now only consists of one series of samples with the individual channels appended after each other. This also has consequences for the interpretation of the temporal dimension since two samples s_1 and s_2 coming from channels 1 and 2 that were recorded at the same timestamp t now appear at two different timestamps t and $t + sequence\_length$. An even more concerning drawback to this approach is that the size of the data grows exponentially in the size of the embedding dimension. This means that the self-attention calculation quickly becomes impractical even for relatively small embeddings.

Another challenge is that a standard implementation of an embedding layer rest on the assumption that the input tokens come from a fixed size vocabulary. Since the EEG signal is measured in floating point numbers, the implicit vocabulary is then practically infinite. This model I report on in the results simply used the approach of treating the channel dimension as the embedding dimension. The model was trained with two layers each containing 2 attention heads, a dropout rate of 0.7 and an embedding size of 22.

4.2.3 Preprocessing using permutation patterns

The challenges that arise from trying to give a transformer model EEG data as input is what inspired the idea of using permutation patterns to encode the EEG data into a representation that more closely resembles the structure of an input sequence of tokens. Permutation patterns is a concept from the field of permutation entropy which studies the complexity or regularity of time series

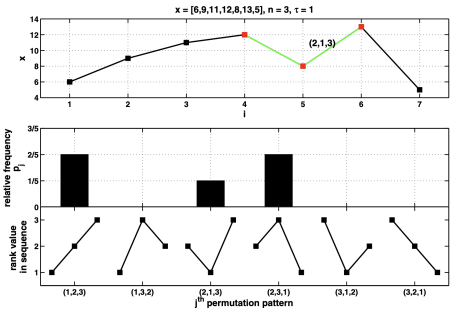

Figure 4.1: Permutation patterns for a sequence x with $n = 3$. The first row contains the time series sequence x. The sequence x_4, x_5, x_6 with the values $12, 8, 13$ is represented as the rank sequence $(2, 1, 3)$ where the ranks are the indices of the values in ascending sorted order. The last row contains all the $3!$ possible permutations of length 3. In the middle are the relative frequencies of the patterns in the sequence. Figure taken from [1].

data. The idea is that a given sequence of time series data $x_i, ...x_{i+n-1}$ of length n can be represented as a rank sequence $r_i, ...r_{i+n-1}$, where the ranks are the indices of the values in ascending sorted order (see fig 4.1). This means that for a given sequence length n there are $n!$ possible distinct permutation patterns [1].

This representation filters out the original values but still encodes the temporal pattern of the time series data. My idea has been to use these possible patterns as the vocabulary for a transformer and transform the EEG data into permutation pattern "tokens". The reasoning behind this idea is that encoding the EEG data in this way might be able to simultaneously compress the data and extract only the essential pattern of the data instead of the specific values. This would transform the EEG data from a continuous stream of floating point measurements into a series of "tokens" in a finite vocabulary defined by the possible permutations for a given window length [1].

I implemented this idea as a preprocessing step where a window length (the number of samples to encode as a "token") and a stride (the number of samples to move when sliding across the data) is first specified. Then all possible permutation patterns for the given window length are calculated and stored as a vocabulary list. A kernel with the specified window length and stride then moves through the EEG data and encodes the data under the kernel into a permutation pattern. The index for this pattern in the vocabulary is then appended to a new sequence as a "token" (see fig 4.2) .

Since the vocabulary size is the factorial of the window length, window sizes above 7 are impractical and defeats the purpose of having a limited vocabulary (since $8! = 40.320$). I ran the experiment with all possible combinations of

[1] As EEG data is, of course, not entirely continuous but consists of individual samples collected with a given sampling frequency.

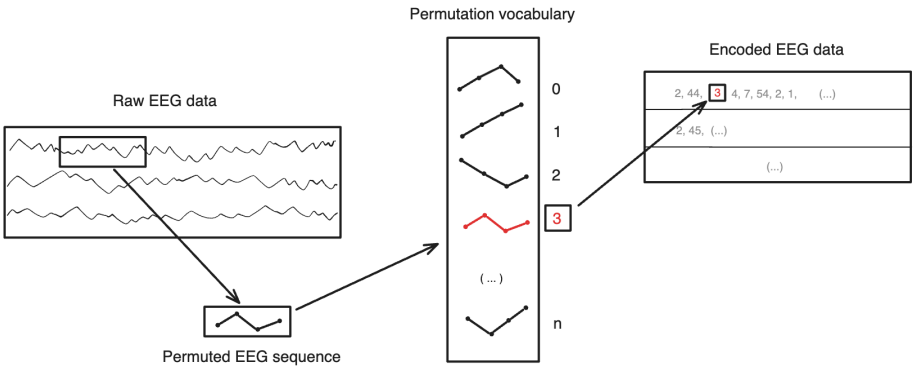

Figure 4.2: EEG encoding using permutation patterns. A window length is specified in advance. The permutation pattern vocabulary is then computed with all possible $n!$ permutation patterns for the specified window length. A kernel with the window length and a specified stride then slides through the raw EEG data one channel at a time, transforming the raw EEG data under the kernel into its permutation pattern representation. This pattern is then looked up in the permutation pattern vocabulary and its index in the vocabulary is appended to the encoded EEG representation.

window sizes ranging from 4 to 9 (inclusive) and strides ranging from 6 to 11 (inclusive). The transformer model architecture for these experiments are largely the same as for the transformer only experiments. Since the data now consists of integers instead of floating point values it was possible to add a learnable embedding layer. But, as described above, a transformer model is not able to handle this extra dimension. To address this issue I flattened the channel dimension before creating the embedding dimension. The model was trained with two layers each containing 2 attention heads, a dropout rate of 0.7 and an embedding size of 22.

4.2.4 Preprocessing using a simplified version of permutation patterns

To address the issue of large window sizes leading to impractically large vocabularies and to be able to try out larger window sizes, I implemented a simplified version of the permutation pattern preprocessing strategy described above. Instead of representing a sequence by as a rank sequence this strategy simply assigns each sample a 0 or 1 corresponding to the sample having a lower or higher value compared to the previous sample respectively.

The vocabulary is then all possible binary combinations for the given window length n which has the size 2^n. The vocabulary is then significantly smaller than the pure permutation pattern approach and the data is compressed even more since the internal rank between samples is discarded. The model was trained

with two layers each containing 2 attention heads, a dropout rate of 0.5 and an embedding size of 10.

4.2.5 Combining GNNs and transformers (the GraphFormer model)

Instead of the manual feature extraction approaches to discretizing the EEG data a better approach could be to combine a transformer model with another deep learning model that can extract useful feature from the EEG data. This approach is similar to the Conformer model as described above. The spatial feature extraction in the Conformer model only applies one kernel and assumes that the spatial relationship between channels is euclidean since it assumes that channels placed next to each other in the channel dimension of the EEG data are spatially close and therefore interact more with each other than channels far from each other in the channel dimension.

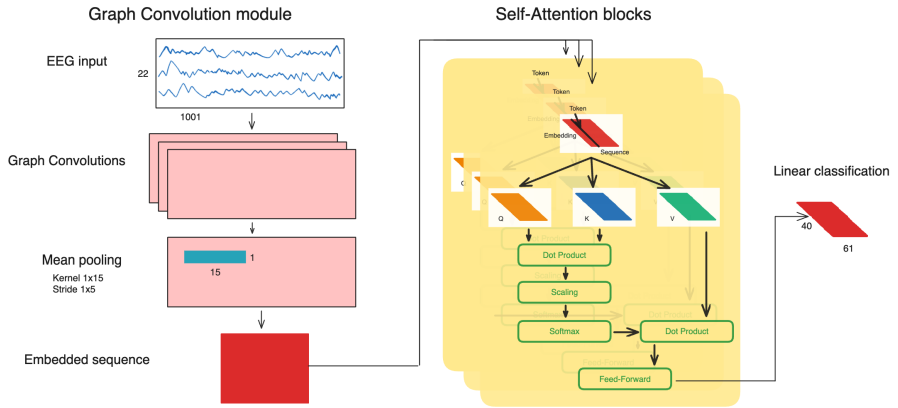

Figure 4.3: The GraphFormer model architecture. The GraphFormer consists of a graph convolution module, a transformer module and a linear classifier. The graph convolution module consists of 3 graph convolution layers with input and output shape identical to the EEG input, and a mean pooling layer with kernel shape $(1, 15)$ *and stride* $(1, 5)$*. The output from the graph convolution module is passed through 10 self-attention heads before it is passed through a linear classification layer.*

Graph convolutional layers do not assume that the spatial dimension is euclidean. These models are good at capturing non-euclidean interactions between the channels and combining a transformer model with a graph convolutional layer could potentially result in a model that can simultaneously capture the complex spatial patterns and long term dependencies in the EEG data.

I implemented this with a series of n Chebychev convolutional layers with the same input and output shape as the original EEG data. Each of these layers consists of a Chebychev convolution, a ReLU activation function and a

dropout layer. The adjacency matrix A is computed using a learnable graph construction layer following [15]. After the graph convolutional layers a mean pooling layer is applied with kernel shape $(1, 15)$ and stride $1, 5)$. The output from this layer is treated as an embedded sequence of tokens with the channels as the embedding dimension. This sequence is the input to the transformer module which applies k self-attention blocks. The output from the transformer module is passed through a linear classification layer with the different class probabilities as output (see fig 4.3). The model was trained with two layers each containing 2 attention heads, a dropout rate of 0.5, an embedding size of 22, 3 graph convolution blocks, K=2, and average pool kernel of shape (15,5).

4.3 Modifying the ShallowFBCSPNet architecture

4.3.1 The CollapsedShallowNet model

Both the ShallowFBCSPNet model and the Conformer model use the idea of first applying a temporal convolution and then a spatial convolution to capture both the temporal and spatial features of the EEG data. In the temporal convolution the kernel is only covering one channel at a time and in the spatial convolution the kernel only covers one time step. The temporal and spatial feature extraction is thus split in two. The temporal convolution is not able to learn the spatial features and the spatial convolution can not learn the temporal features. The reasoning behind this split is that this implicitly regularizes the the overall convolution by forcing a separation of the linear transformation into two. Schirrmeister et al. argue that this might lead to better performance due to the large number of channels compared to image classification tasks with only 3 channels (rgb) [11].

By applying k two dimensional kernels with shape $(channels, window)$, where $channels$ is the number of electrodes in the EEG data and $window$ is the length of the temporal window in the ShallowFBCSPNet and Conformer models it is possible to create one spatiotemporal convolutional layer that is able to learn the temporal and spatial features simultaneously. The input shape is $(1, channel, sample)$. The output shape from this convolution is the same as applying both a spatial and a temporal convolution since the kernel covering all channels with a stride of 1 gives an output feature map of shape $(1, 1, t)$ where $t = (samples - window) + 1$. There are k feature maps and the output shape is therefore $(k, 1, t)$ (see fig 4.4). I call this model CollapsedShallowNet.

The output of the spatiotemporal layer is passed through the ELU activation function, and batch normalisation is applied before the data is sent through a mean pooling layer with a kernel shape of $(1, 75)$ and stride of $(1, 75)$. The kernel shape is the same as in the ShallowFBCSPNet and Conformer models, but I found a stride of $(1, 75)$ to have slightly better performance, better prevent overfitting and resulting in fewer total model parameters. After the mean pooling layer a dropout layer with 50% dropout probability is applied to reg-

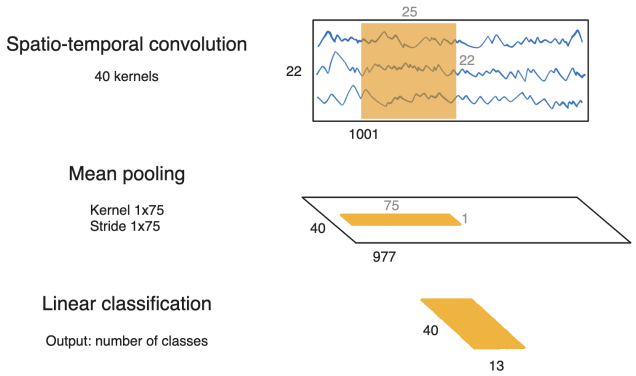

Figure 4.4: The CollapsedShallowNet architecture. The model consists of one spatiotemporal convolutional layer, a mean pooling layer and a linear classification layer. The spatiotemporal convolution has 40 kernels with a kernel shape of $(22, 25)$ *and a stride of* 1*. The output shape from the spatiotemporal layer is* $(40, 977)$*, where 40 is the number of kernels and 977 is the number of samples (1001) minus the kernel length + 1. The mean pooling layer has a kernel of shape* $(1, 75)$ *and a stride of* $(1, 75)$*. The output from the mean pooling layer is sent through a linear classification layer with the predicted probabilities for the different classes as output.*

ularize the model, preventing overfitting. The final layer is a fully connected linear layer with shape $((k \times t_p), num_classes)$, where k is the number of kernels in the spatiotemporal layer, t_p is the temporal dimension after mean pooling, and $num_classes$ is the number of classes in the dataset.

If this model architecture achieves performance comparable to the ShallowFBCSPNet model it would be an attractive alternative to the two layer temporal and spatial convolution approach adopted by both the ShallowFBCSPNet and models like the Conformer. Combining the temporal and spatial convolutions into one layer makes the model conceptually clearer, reduces the number of parameters and might lead to better performance because the same layer can take both temporal and spatial features into account at the same time. The model was trained with a dropout rate of 0.4, 40 kernels, kernel shape of (22,25) and mean pool kernel of (1,75).

4.3.2 The CollapsedConformer model

Since the Conformer model implements the ShallowFBCSPNet architecture, it is also possible to substitute the temporal and spatial convolutions with one spatiotemporal convolution in this model. I did this using the same architecture as in the CollapsedShallowNet model but without the classification layer. Instead the output from the mean pooling layer is treated as a sequence of em-

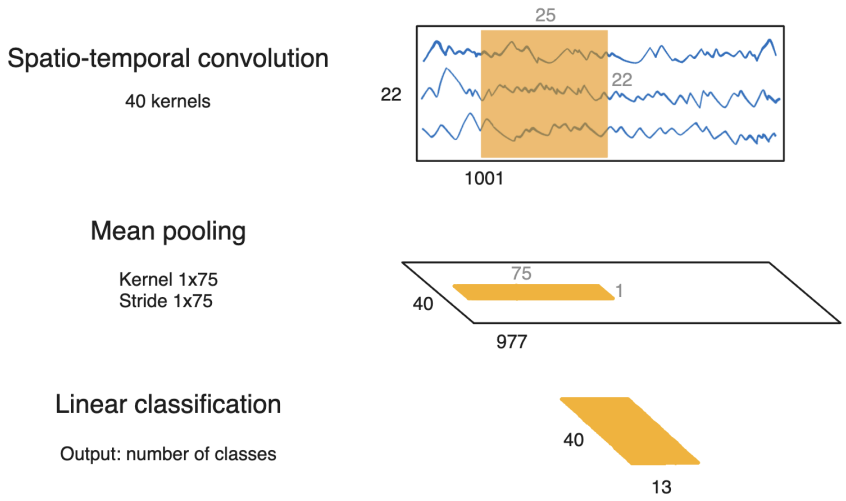

Figure 4.5: The CollapsedConformer model architecture. The model consists of two modules: a CNN block implementing the CollapsedShallowNet architecture excluding the classification layer, and a transformer module consisting of multiple self-attention blocks. The output of the transformer module is sent through a linear classification layer with the predicted probabilities for the different classes as output.

bedded tokens and sent into the the transformer module containing multiple self-attention blocks (see fig 4.5). As with the ShallowFBCSPNet model this implementation simplifies the model conceptually and reduces the number of trainable parameters. The model was trained with 6 layers each containing 10 attention heads, a dropout rate of 0.6, 40 kernels, a kernel shape of (22,25) and mean pool kernel of (1,75).

4.4 Experimental details

For the datasets using permutation patterns and using a simplified version of permutation patterns the data was downsampled to a third of the original length using max pooling. For all other models the dataset was normalised using z-score normalisation, defined as $X_{normalised} = \frac{X-\mu}{\sigma}$, where X is the original data, μ is the mean of the data and σ is the standard deviation. All models were implemented in PyTorch in Python 3.11.7 and trained on a NVIDIA GeForce GTX 1080 Ti GPU. The models were trained using the Adam optimiser with a learning rate of 0.0002 and weight decay of 0.0001.

Results

To be consistent all deep learning models were tested with the same setup. The models were trained for 900 epochs (1800 steps) with a learning rate of 0.0001 and a batch size of 64. I used cross entropy as the loss function for all models since the Conformer model does not work with binary cross entropy in the braindecode implementation I used. Since the classes are balanced this should not be a problem. Cross entropy loss is calculated as

$$L = -\frac{1}{N_b} \sum_{i=1}^{N_b} \sum_{c=1}^{M} y \log(\hat{y})$$

where N_b is the number of trials in a batch, M is the number of classes, y is the ground truth and $\hat{y}$ is the predicted label. The training datasets were split up into training and validation sets with 80% and 20% of the data respectively. In the following I report the test accuracy and present the model's training and validation validation accuracy curves along with their training and validation cross entropy loss curves. The accuracy is given by the sum of true positives and true negatives divides by the sum of all positives and negative. For the models using permutation patterns and simplified permutation patterns I only report on the models with the best performing combination of window length and stride. The test accuracy scores are reported in Table 5.1. The results section is divided into: 1. a section on the results for models I explored for finding a new approach to EEG encoding and 2. a section on the modified versions of ShallowFBCSPNet and Conformer models, following the division in the methods chapter.

5.1 Results for new encoding approaches

5.1.1 CSP + LDA results

The CSP + LDA model achieved 59.59% accuracy on dataset 2a and 78.67% accuracy on dataset 2b.

Table 5.1: Test results for all models on dataset 2a and 2b.

Dataset	Model	Test accuracy
Dataset 2a	Transformer only	29.17%
	PP model	24.5%
	Simple PP model	28.2%
	GraphFormer	27.28%
	CSP + LDA	46.1%
	ShallowFBCSPNet	58.12%
	CollapsedShallowNet	59.88%
	ConformerCopy	63.12%
	CollapsedConformer	**64.27%**
Dataset 2b	Transformer only	50.85%
	PP model	50.77%
	Simple PP model	50.39%
	GraphFormer	54.01%
	CSP + LDA	71.99%
	ShallowFBCSPNet	73.77%
	CollapsedShallowNet	76.47%
	ConformerCopy	73.47%
	CollapsedConformer	**78.47%**

5.1.2 Transformer only results

The transformer only model achieved 29.17% test accuracy on dataset 2a where 25% is the baseline accuracy of pure chance. On dataset 2b it had a test accuracy of 50.85% where 50% is the baseline of pure chance. During training on both datasets the training accuracy is rapidly increasing and stabilising close to 100% and the loss curve is decreasing rapidly and stabilising after 25 epochs (50 steps). The validation cross entropy loss curve is generally increasing with some fluctuation and there is little to no increase in validation accuracy, suggesting that the model is overfitting to the training data (see fig 5.1 and 5.2).

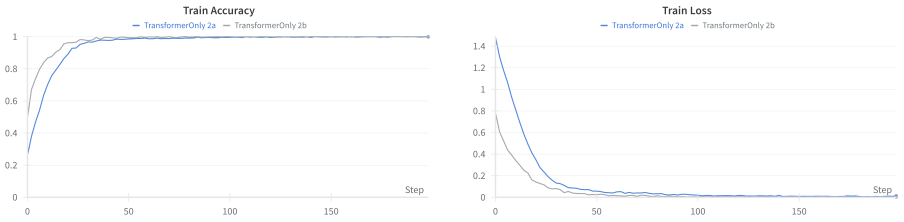

Figure 5.1: Train accuracy and cross entropy loss for the TransformerOnly model on datasets 2a and 2b. The model was trained for 100 epochs and the train accuracy and cross entropy loss is rapidly increasing and decreasing respectively, plateauing after 25 epochs (50 steps).

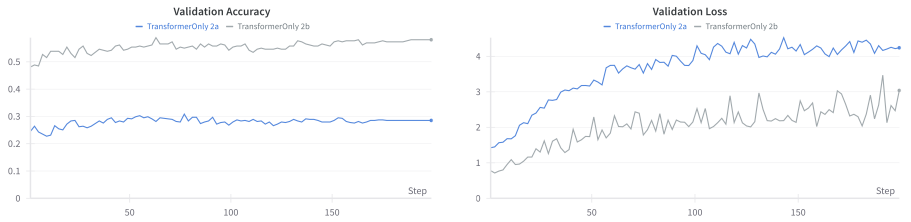

Figure 5.2: Validation accuracy and cross entropy loss for the TransformerOnly model on datasets 2a and 2b. The validation accuracy is generally increasing from near 50% to near 55% on dataset 2b. On dataset 2a the validation accuracy curve is almost entirely flat. On both datasets the validation cross entropy loss is generally increasing with some fluctuation throughout the training.

5.1.3 Preprocessing using permutation patterns (PP model) results

The PP model achieved 24.5% test accuracy on dataset 2a where 25% is the baseline accuracy of pure chance. On dataset 2b it had a test accuracy of 50.77% where 50% is the baseline of pure chance. On both datasets the training accuracy curves have a steep initial increase followed by a gradual slope stabilising close to 100% training accuracy and the training cross entropy loss curves follows an inverse pattern of the training accuracy curves. The validation cross entropy loss curve is generally increasing with some fluctuation and there is little to no increase in validation accuracy, suggesting that the model is overfitting to the training data (see fig 5.3 and 5.4).

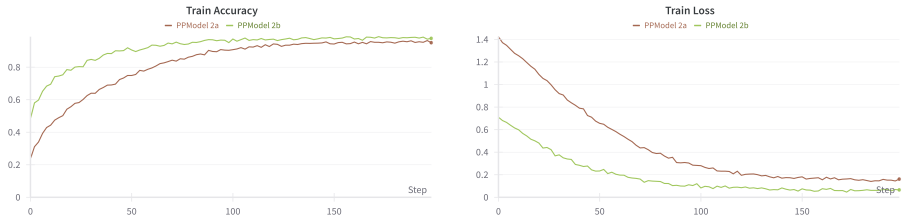

Figure 5.3: Train accuracy and cross entropy loss for the PP model on datasets 2a and 2b. The model was trained for 100 epochs and the training accuracy curves have a steep initial increase followed by a gradual slope plateauing close to 100% training accuracy. The training cross entropy loss curves follows an inverse pattern of the training accuracy curves plateauing after around 75 epochs (150 steps).

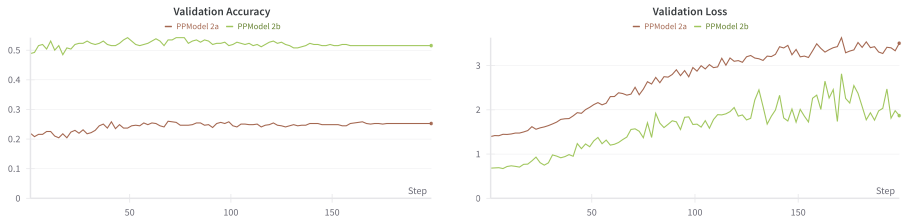

Figure 5.4: Validation accuracy and cross entropy loss for the PP model on datasets 2a and 2b. The validation cross entropy loss curve is generally increasing with some fluctuation and there is little to no increase in validation accuracy, suggesting that the model is overfitting to the training data.

5.1.4 Preprocessing using a simplified version of permutation patterns (Simple PP model) results

The Simple PP model achieved 28.2% test accuracy on dataset 2a where 25% is the baseline accuracy of pure chance. On dataset 2b it had a test accuracy of 50.39% where 50% is the baseline of pure chance. The train accuracy is generally increasing with the rate of improvement declining after 25 epochs on dataset 2b but almost flat on dataset 2a. Both the train and validation cross entropy loss curves are flat on both datasets. This suggests that the model is underfitting because the preprocessing strategy is compressing the data too much and leaves out important features of the data, making the model unable to learn (see fig 5.5 and 5.6).

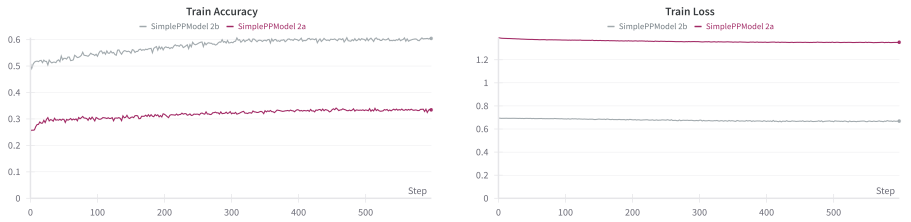

Figure 5.5: Train accuracy and cross entropy loss for the Simple PP model on datasets 2a and 2b. The model was trained for 100 epochs. The train accuracy is generally increasing with the rate of improvement declining after 25 epochs (50 steps) on dataset 2b but flat on dataset 2a, The cross entropy train loss is flat on both datasets.

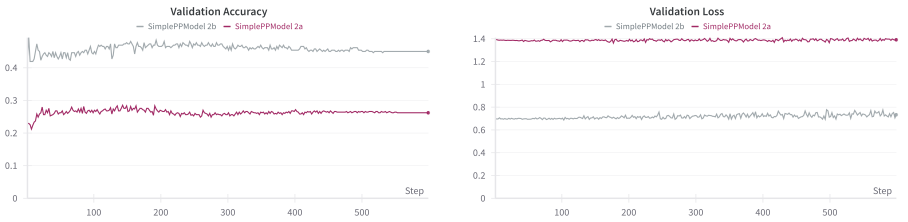

Figure 5.6: Validation accuracy and cross entropy loss for the Simple PP model on datasets 2a and 2b. The validation accuracy is and validation cross entropy loss is almost entirely flat on both datasets, suggesting underfitting.

5.1.5 GraphFormer results

The Simple PP model achieved 27.28% test accuracy on dataset 2a where 25% is the baseline accuracy of pure chance. On dataset 2b it had a test accuracy of 54.01% where 50% is the baseline of pure chance. On both datasets the training accuracy is rapidly increasing and plateauing close to 100% after 15 epochs. The validation cross entropy loss curve generally increasing with some fluctuation and with little to no increase in validation accuracy, suggesting that the model is overfitting to the training data (see fig 5.7 and 5.8).

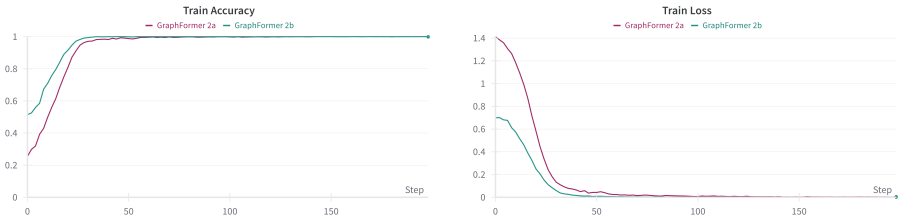

Figure 5.7: Train accuracy and train cross entropy loss for the GraphFormer model on datasets 2a and 2b. The training accuracy is rapidly increasing and stabilising close to 100% after 15 epochs (30 steps) on both datasets.

5.2 Results for collapsed ShallowFBCSPNet and Conformer models

To be able to directly compare the performance of the ShallowFBCSPNet and Conformer models against the versions of the models with a combined spatiotemporal convolution I have reimplemented the models and made a version of the models with a separate spatial and temporal convolution and one with a one layer spatiotemporal convolution.

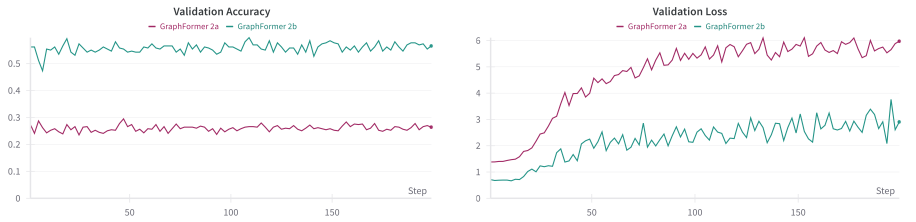

Figure 5.8: Validation accuracy and cross entropy loss for the GraphFormer model on datasets 2a and 2b. The validation accuracy is almost entirely flat on both datasets, while the validation cross entropy loss is generally increasing with some fluctuation throughout the training on both datasets.

5.2.1 ShallowFBCSPNet and CollapsedShallowNet results

The ShallowFBCSPNet model achieved 58.12% and 73.77% test accuracy on dataset 2a and 2b respectively. The test accuracy for the CollapsedShallowNet model on dataset 2a and 2b was 59.88% and 76.47% respectively. Both the ShallowFBCSPNet and the CollapsedShallowNet model's training accuracy curves are increasing throughout the training with an initially steep slope that becomes more gradual after 100 epochs. On dataset 2a the training accuracy curve for the ShallowFBCSPNet model has a more rapid improvement than the CollapsedShallowNet and is above 80% after 900 epochs, where the training accuracy for the training accuracy for the CollapsedShallowNet is around 65% at the end of the training. The training cross entropy loss curves follow the inverse pattern of the training accuracy curves. The validation accuracy curves on dataset 2a also show a difference between the two models. The ShallowFBCSPNet model has a more rapid initial improvement and plateauing around 60% validation accuracy after 100 epochs and the CollapsedShallowNet has a less rapid initial improvement plateauing to around 60% after 300 epochs. The validation accuracy curves are never plateauing completely and might have increased more if the models had been trained for more epochs, but the validation cross entropy loss is plateauing after 600 epochs for both models (see fig 5.9 and 5.10).

The training accuracy and cross entropy loss curves follow the same pattern on dataset 2b. The ShallowFBCSPNet validation accuracy on dataset 2b shows an initial rapid improvement and is plateauing around 76% after 100 epochs, where the validation accuracy for the CollapsedShallowNet shows a less rapid initial improvement stabilising around 79% after 400 epochs. Both models' validation cross entropy loss curves are initially decreasing following the inverse shape of the validation accuracy curves. Both models' validation loss curves show some fluctuation but more so for the ShallowFBCSPNet model (see fig 5.11 and 5.12).

The larger gap between the train and validation accuracy for the ShallowFBCSPNet model compared to the CollapsedShallowNet model suggests that the ShallowFBCSPNet model is more prone to overfitting, while the lower train ac-

curacy for the CollapsedShallowNet model suggests that this model is limited in its ability to find patterns in the data.

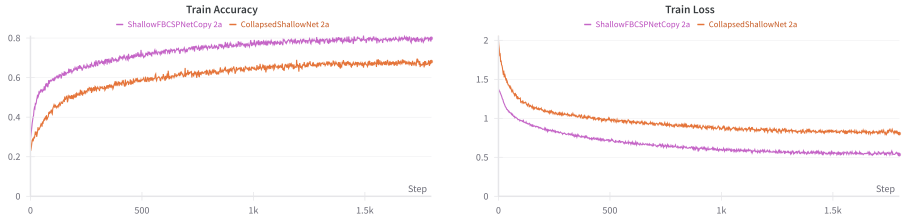

Figure 5.9: Train accuracy and train cross entropy loss for the ShallowFBCSPNet and CollapsedShallowNet models on dataset 2a. The models were trained for 900 epochs. The training accuracy curve for the ShallowFBCSPNet model has a more rapid improvement than the CollapsedShallowNet and is above 80% after 900 epochs, where the training accuracy for the training accuracy for the CollapsedShallowNet is around 65% at the end of the training. The training cross entropy loss curves follow the inverse pattern of the training accuracy curves.

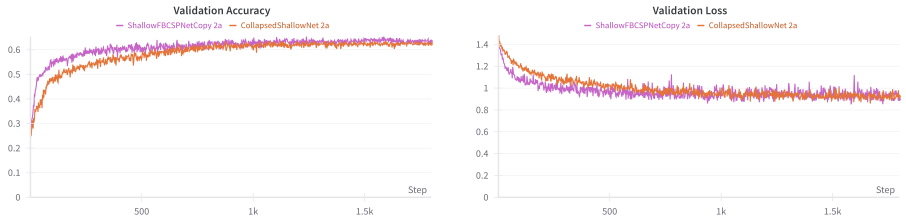

Figure 5.10: Validation accuracy and validation cross entropy loss for the ShallowFBCSPNet and CollapsedShallowNet models on dataset 2a. The ShallowFBCSPNet model has a more rapid initial improvement and plateauing around 60% validation accuracy after 100 epochs and the CollapsedShallowNet has a less rapid initial improvement plateauing to around 60% after 300 epochs.

5.2.2 ConformerCopy and CollapsedConformer results

The ConformerCopy model achieved 63.12% and 73.47% test accuracy on dataset 2a and 2b respectively. The test accuracy for the CollapsedConformer model on dataset 2a and 2b was 64.27% and 78.47% respectively. The training accuracy and cross entropy loss curves for the two models follow a similar path on dataset 2a. The training accuracy curves have an initial steep slope transitioning into a more gradual increase after 200 epochs and plateauing around 70% after 800 epochs, with the training cross entropy loss following the inverse pattern. The validation accuracy curves for both models on dataset 2a show an initial steep slope turning into a more gradual slope after 200 epochs and

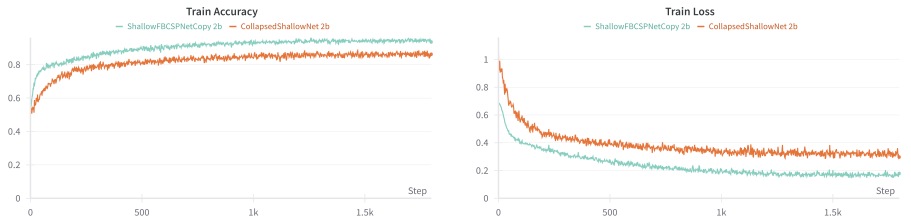

Figure 5.11: Train accuracy and train cross entropy loss for the ShallowFBCSPNet and CollapsedShallowNet models on dataset 2b. The models were trained for 900 epochs. The training accuracy curve for the ShallowFBCSPNet model has a more rapid improvement than the CollapsedShallowNet and is above just below 90% after 900 epochs, where the training accuracy for the training accuracy for the CollapsedShallowNet is around 84% at the end of the training. The training cross entropy loss curves follow the inverse pattern of the training accuracy curves.

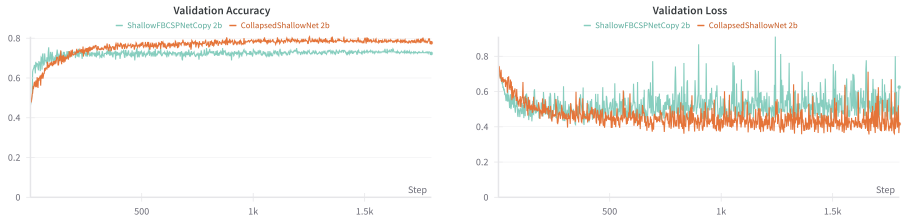

Figure 5.12: Validation accuracy and validation cross entropy loss for the ShallowFBCSPNet and CollapsedShallowNet models on dataset 2b. The ShallowFBCSPNet validation accuracy shows an initial rapid improvement and is plateauing around 76% after 100 epochs, where the validation accuracy for the CollapsedShallowNet shows a less rapid initial improvement stabilising around 79% after 400 epochs (800 steps). Both models' validation cross entropy loss curves are initially decreasing following the inverse shape of the validation accuracy curves. Both models' validation loss curves show some fluctuation but more so for the ShallowFBCSPNet model.

stabilising after 600 epochs with the validation accuracy for the ConformerCopy model stabilising around 67% and the CollapsedConformer stabilising around 65%. The validation cross entropy loss curves are initially decreasing rapidly but stabilising around 0.8 after 300 epochs for both models.

On dataset 2b the training accuracy curve for both models initially increase rapidly for the first 20 epochs and then increase with a more gradual slope until the end of the training with the CollapsedConformer training accuracy curve being around 1% above the ConformerCopy training accuracy curve after 300 epochs. The training cross entropy loss curves are decreasing rapidly for the

first 50 epochs and then decreasing more gradually. The validation accuracy curve on dataset 2b is initially improving rapidly for both models, turning to a more gradual slope after 30 epochs with the accuracy for the CollapsedConformer stabilising around 2% higher than the ConformerCopy accuracy after 600 epochs. The validation cross entropy loss for the CollapsedConformer is initially decreasing rapidly and then stabilising around 0.4 after 200 epochs with some fluctuation. The validation loss for the ConformerCopy also decreases rapidly for the first 50 epochs and stabilising around 0.5 with some more fluctuation than for the CollapsedConformer model.

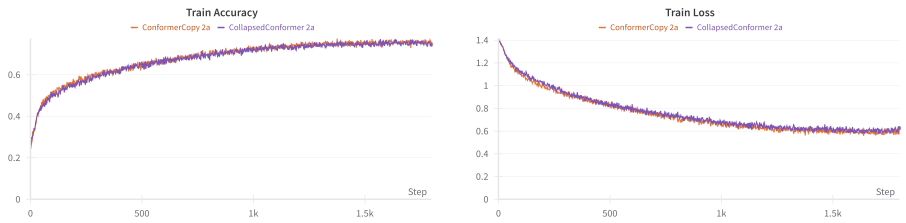

Figure 5.13: Train accuracy and train cross entropy loss for the ConformerCopy and CollapsedConformer models on dataset 2a. The models were trained for 900 epochs. The training accuracy and cross entropy loss curves for the two models follow a similar path having an initial steep slope transitioning into a more gradual increase after 200 epochs (400 steps) and plateauing around 70% after 800 (1600 steps) epochs, with the training cross entropy loss following the inverse pattern.

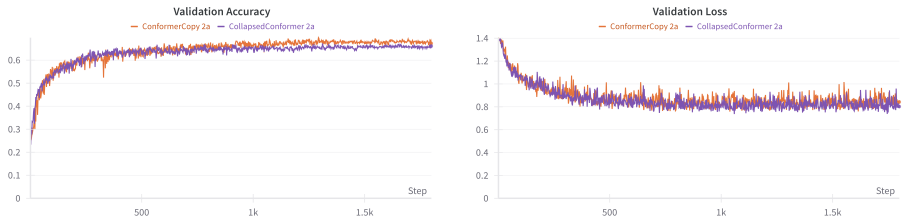

Figure 5.14: Validation accuracy and validation cross entropy loss for the ConformerCopy and CollapsedConformer models on dataset 2a. The validation accuracy curves for both models on dataset 2a show an initial steep slope turning into a more gradual slope after 200 epochs (400 steps) and stabilising after 600 epochs (1200 steps) with the validation accuracy for the ConformerCopy model stabilising around 67% and the CollapsedConformer stabilising around 65%. The validation cross entropy loss curves are initially decreasing rapidly but stabilising around 0.8 after 300 epochs (600 steps) for both models

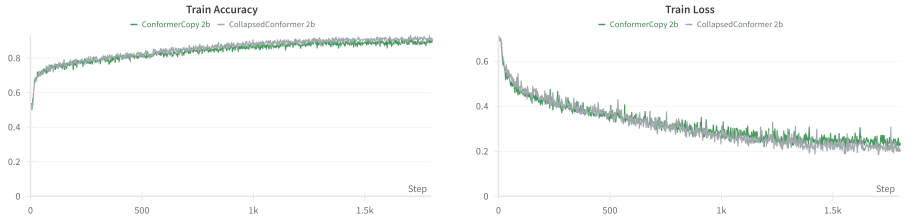

Figure 5.15: Train accuracy and train cross entropy loss for the ConformerCopy and CollapsedConformer models on dataset 2b. The models were trained for 900 epochs. The training accuracy curve for both models initially increase rapidly for the first 20 epochs (40 steps) and then increase with a more gradual slope until the end of the training with the CollapsedConformer training accuracy curve being around 1% above the ConformerCopy training accuracy curve after 300 epochs (600 steps). The training cross entropy loss curves are decreasing rapidly for the first 50 epochs (100 steps) and then decreasing more gradually.

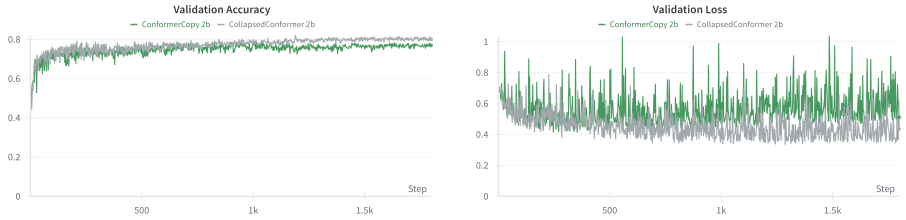

Figure 5.16: Validation accuracy and validation cross entropy loss for the ConformerCopy and CollapsedConformer models on dataset 2b. The validation accuracy curve is initially improving rapidly for both models, turning to a more gradual slope after 30 epochs (60 steps) with the accuracy for the CollapsedConformer stabilising around 2% higher than the ConformerCopy accuracy after 600 epochs (1200 steps). The validation cross entropy loss for the CollapsedConformer is initially decreasing rapidly and then stabilising around 0.4 after 200 epochs (400 steps) with some fluctuation. The validation loss for the ConformerCopy also decreases rapidly for the first 50 epochs (100 steps) and stabilising around 0.5 with some more fluctuation than for the CollapsedConformer model.

Discussion

6.1 Recap of results

Without any prior encoding the TransformerOnly model was unable to learn generalisable features of the data and overfitted significantly. Neither the permutation pattern approach to preprocessing the EEG data or the simplified version of this approach significantly improved the transformers classification performance if at all. The GraphFormer model which combines graph convolutional layers with a transformer model also overfitted heavily with no ability to learn meaningful patterns in the data. As expected, my implementations of the ShallowFBCSPNet and Conformer models performed well on both datasets and outperformed the CSP + LDA traditional ML model. The alternative version of the two models with the temporal and spatial convolutions combined into one spatiotemporal convolution (the CollapsedShallowNet and CollapsedConformer models) achieved higher test accuracy scores on both datasets and were less prone to overfitting than the original versions of the models.

6.1.1 Possible causes of poor model performance

Permutation pattern approaches

The permutation pattern approach to preprocessing EEG data and especially the simplified version of the approach initially seemed to show some promising results when training on a subset of dataset 2b with containing only trials for 3 different subjects. With the right combination of window size and stride many of the models were able to achieve around 60% validation accuracy (I did not use the test data when developing the models to avoid train-test contamination). One of the reasons this performance did not carry over when expanding the training to the full dataset might be that the models overfitted to the validation set because the validation set was too small or not representative of the general distribution in the data. It might also be due to excessive hyperparameter tuning leading to the models performing well on the validation dataset but not learning general features of the data. A better approach would have been to use the full

dataset from the beginning and use cross-validation to ensure that the models were validated on multiple different validation sets. This would, however, have resulted in significantly longer training times during development.

Even if the performance on the smaller dataset had carried over to the full dataset, a validation accuracy of 60% on dataset 2b is still far from state of the art performance for deep learning models and significantly lower than a traditional ML model like CSP + LDA which had a test accuracy of 71.99%. The preprocessing transformation of the EEG data prevents the models from overfitting to the same degree as the TransformerOnly model. This might be because the preprocessing compresses the data in a way that makes it more difficult for the models to fit to random noise in the data. On the other hand, especially for the simplified permutation pattern approach, the compression seems to be too strong, making the models unable to find any patterns in the data at all.

On a more general level the permutation pattern approaches are a type of manual feature engineering. This can be a good strategy in many ML contexts, since it makes it possible to use domain-specific knowledge to transform the raw data into more meaningful and relevant features. Manual feature engineering may also lead to better interpretability. But there are also some important drawbacks. The feature engineering may result in important features being discarded and risks overlooking complex relationships in the data. Furthermore manual feature engineering is inflexible compared to automatic feature extraction techniques, where the parameters for the feature extraction can be learned along with the other model parameters.

In comparison with the Conformer model all of these potential drawbacks of manual feature engineering probably contribute to the poor model performance. The CNN module in the Conformer model also compress the EEG data into a more compact format, but the compression is less heavy and the feature extraction parameters are trainable. Additionally, using convolutional layers for feature extraction is more general than the permutation pattern approach in the sense that it is able create any representation that the permutation pattern approach can create but not the other way around. Since the kernels in the convolutional layers are learnable they could in principle learn to encode the data in the exact same way as the manual feature engineering.

The GraphFormer

There are several possible reasons why the GraphFormer model was essentially unable to learn generalisable features of the data. One possible reason is that the graph convolution module is not compressing the data enough before it is sent to the transformer module. As described above, fitting a transformer model directly on raw EEG data leads to heavy overfitting. The graph convolutional module is only able to capture additional relationships between the channels but only relies on a mean pooling layer to reduce the transformers tendency to overfit. Another reason could be that having multiple graph convolutional layers can lead to over-smoothing of the data, where the node features become

more and more similar after each layer, making it more difficult for the model to distinguish different features in the data.

6.2 Combining spatial and temporal convolutions

ShallowFBCSPNet and CollapsedShallowNet models

As described in the literature review, Schirrmeister et al. argue that having separate temporal and spatial convolutions might lead to better performance than combining the two layers because it introduces some implicit regularisation. This is not what I find when implementing the ShallowFBCSPNet and Conformer models with and without combining the temporal and spatial convolutions into one spatiotemporal convolution. In my results the models with a combined spatiotemporal convolutional layer have performance similar to- or better than the models with a temporal and spatial split. The difference in performance between the ShallowFBCSPNetCopy and the CollapsedShallowNet (58.12% vs 59.88% and 73.77% vs 76.47% on dataset 2a and 2b respectively) is not large enough to conclude that the CollapsedShallowNet is consistently better than the ShallowFBCSPNetCopy model. On the other hand, none of my experiments showed better performance for the ShallowFBCSPNetCopy than the CollapsedShallowNet, and when also taking model size and conceptual complexity into account I will argue that the CollapsedShallowNet is the best alternative. The ShallowFBCSPNetCopy has a total of 41.242 trainable parameters, where the CollapsedShallowNet only has 23.162 trainable parameters, which results in less training time and computational resources needed. The training accuracy curves for the two models suggests that the ShallowFBCSPNetCopy model is more prone to overfitting to the training data than the CollapsedShallowNet considering that it has almost 20% higher training accuracy but lower validation accuracy on dataset 2a. The CollapsedShallowNets relatively low training accuracy scores might, on the other hand, suggest that this model is less capable of finding complex patterns in the data, which could be an issue in more complex tasks than the motor imagery classification I have used to compare the models.

Conformer and CollapsedConformer models

Similar to the difference in results between the ShallowFBCSPNetCopy and the CollapsedShallowNet models, the difference in performance between the ConformerCopy and the CollapsedConformer is not entirely conclusive. But the results suggest that the CollapsedConformer is at least as good as the ConformerCopy. On dataset 2b the CollapsedConformer model even outperforms the ConformerCopy with 5% which is a remarkable gain in performance if the result on multiple tests. To evaluate if the difference in performance is truly

significant, multiple tests would have to be run with both models and compared using a statistical test of significance such as a paired t-test.

As with the CollapsedShallowNet, the CollapsedConformer is also less conceptually complex than the original model. The model also has fewer trainable parameters (317.786 vs 303.546 for the CollapsedConformer), but with the transformer module being responsible for most of the trainable parameters, the relative difference in total number of parameters is less significant.

The CollapsedConformer generally has a higher validation accuracy and less fluctuating validation cross entropy loss curve during training than the ConformerCopy. The training accuracy and cross entropy loss curves for the two models follow roughly the same path. In contrast to the difference between the ShallowFBCSPNetCopy and the CollapsedShallowNet models, there does not seem to be a difference in the two models' overfitting tendency or ability to find complex patterns in the data.

6.3 Transformers in EEG classification

In the introduction I argue that, from an analytical perspective, transformers have the potential to be very effective at EEG classification tasks because they are parallelizable and capable of capturing long term dependencies in the data. My results have shown that this greatly depends on how the EEG data is encoded before it is passed to a transformer model. Without any encoding of the EEG, a transformer model is unable to learn any meaningful patterns in the data. The manual feature extraction approaches I experimented with are not improving the performance of the transformer by much, if at all.

Combining a transformer with a CNN module as in the Conformer architecture results in a model with good performance and the conformer module outperforms the CNN module by itself. But, as the ablation study on the Conformer also reveal, the transformer module is only responsible for a relatively little increase in performance, and the CNN module is still the core of the model's performance.

The biggest difference in performance between the Conformer and the CNN models was on dataset 2a. This might suggest that adding a transformer is especially helpful in more complex tasks. It could also be because the transformer benefits more from having more training data than the CNN models (dataset 2a being twice the size of dataset 2b).

Using transformers for as the decoding part of a classification model is not the only possible way of applying transformers to EEG classification tasks. In the literature review I described how the S3T model architecture involved two transformer modules; one for classification and one to weight how much attention to give to the different channels. There are still many unexplored EEG classification models using transformer models. As is evident from the differences in number of trainable parameters described above, transformers are larger and take more time to train compared to CNNs. This means that when introducing a transformer module in a model architecture one should always weight

the possible gain in accuracy with the added complexity of the model.

6.4 CNNs in EEG classification

CNNs are still the most prevalent deep learning model architecture for EEG classification. As described above, even in a model like the Conformer that successfully applies transformers to EEG classification, the CNN module is the backbone of the model's performance. Compared with transformer models, CNNs are lightweight, requiring significantly fewer trainable parameters. At the same time, combining a simple CNN module with a transformer can result in improved performance, especially on BCI datasets with more motor imagery tasks, or more training data. A spatiotemporal convolutional layer can be effective at capturing both temporal and spatial features of the EEG data but still assumes that the relationships between the channels is euclidean.

6.5 Graph convolutions in EEG decoding

My attempt at combining graph convolutions with transformers for EEG classification was unsuccessful. This does not mean that such an architecture is necessarily doomed to fail. There are many unexplored modifications to this architecture. Among them is using alternative convolutions than the Chebychev convolution, or inserting a different module er encoding step before the graph convolution. One approach could be to extend the Conformer model with a graph convolution layer before or after the CNN module. I experimented with both the latter two architectures. Both models had decent performance but worse than without applying the layers. Instead of superimposing a graph convolutional layer on an already functioning model architecture, building such a model from the ground up with attention to how the different hyperparameters influence each other could result in a model with better performance.

The Conformer model is already relatively complex in itself, and adding a graph convolutional layer adds to this complexity, but the argument that graph convolutions can be used to capture non-euclidean relationships between the channels persists. Several EEG classification models that successfully applies graph convolutions already exist [7]. Transformers CNNs and GCNs all have their strengths and weaknesses in the context of EEG classification and it is plausible that it is possible to create a model that utilises the strengths of all these architectures at the same time.

6.6 Limitations of the study

As mentioned above, I did not test the different models enough times to conduct a statistical test that could determine whether the differences in model performance I have found are statistically significant. This makes the claim that the

models using one spatiotemporal convolutional layer instead of separate temporal and spatial convolutions are at least as good as the latter models not fully supported. All results are reported for the best combination of hyperparameters I have been able to find, but I have not conducted a full hyperparameter search covering all reasonable combinations of hyperparameters.

Both the Conformer and ShallowFBCSPNet model are implemented as part of the open-source python EEG toolbox Braindecode[1]. Since these implementations are part of a larger framework with dependencies across the whole framework I have found it easier to implement both models from scratch following the architectures and hyperparameter settings as they are laid out in the original papers. This has made it possible to compare the version of the models using a combined spatiotemporal convolution directly with a corresponding model that only differs in whether the models have two separate temporal and spatial layers or one combined spatiotemporal layer.

The two datasets I used for evaluating the different models are very similar, both in terms of setup, the tasks the trails consist of and the equipment used for recording the EEG data. To determine whether combining the temporal and spatial convolutions into one layer is generally an improvement over the current Conformer and ShallowFBCSPNet models would require to also evaluate the different models on other datasets also outside the field of motor imagery classification. Since Schirrmeister et al. argue that the reason for separating the two layer is due to the many channels in EEG data compared to images, testing the models on datasets with 66 EEG channels or more would be necessary to determine if this is correct.

6.7 Future research directions

The ShallowFBCSPNet model is a simplified version of the Deep ConvNet architecture proposed in the same paper [11]. Comparing an implementation of the Deep ConvNet with and without a combined spatiotemporal convolution would therefore be an obvious extension of this study.

Testing the models on datasets with different tasks, number of electrodes and sampling rates would give insight into whether the combined spatiotemporal approach is generally better than splitting the convolution into two.

In the original paper the Conformer model was trained per subject instead of on all subjects at the same time. Comparing the models' performance when training per subject might give insight into whether the one-layer spatiotemporal approach is better than the original models at finding subject specific patterns versus more general patterns or the other way around. Furthermore, comparing how the different models perform on the datasets if data augmentation is used could give insight into the robustness and overfitting tendency of the models.

The spatiotemporal convolutional layer captures both temporal and spatial features of the EEG data. Visualising the EEG data after it has been transformed by the spatiotemporal convolution could give insights into both how the

[1] `https://braindecode.org/stable/index.html`

model works but also what the important features of the brain's processing are when doing the e.g. the motor imagery tasks in dataset 2a and 2b.

The idea of combining a transformer model with graph convolutions was not explored fully. It is very likely that there are better ways of combining these architectures than the GraphFormer model I implemented.

Conclusion

Being able to correctly classify EEG data has the potential of aiding the diagnosis of Alzheimers and epilepsy. It also has applications in brain computer interface (BCI) technologies such as controlling prosthetic devices.

EEG is a highly complex type of data. The brain itself is an incredibly complex system, and the non-evasive nature of EEG recording of brainwaves introduces a lot of noise in the data. In addition to this, EEG is a type of multivariate time series data, meaning that it consists of multiple time series measurements recorded from different positions on the skull. EEG data therefore has both a temporal and a spatial dimension, and finding good ways to classify EEG signals requires building models that can effectively extract both the relevant spatial and temporal features from the data. In other words it requires encoding the data in a way that captures the spatiotemporal essence of the data into a representation that can be used for classifying the data.

In this project I have explored several different EEG encoding techniques for EEG classification, all on the same two publicly available motor imagery datasets. I found that using transformer models directly on EEG data resulted in a model that was unable to learn any general features of the data. The manual feature extraction approach of using permutation patterns for preprocessing EEG data into a format resembling a finite vocabulary did not improve the performance of a transformer model by much, if at all. Combining a transformer model with graph convolutions also resulted in poor model performance.

The ShallowFBCSPNet and the Conformer are two well performing deep learning models for EEG classification that share the approach of applying a temporal and a convolutional layer to extract spatiotemporal features from the EEG data. I found that combining these two layers into one spatiotemporal convolution not only led to conceptually simpler models with fewer trainable parameters but also made the ShallowFBCSPNet model less prone to overfitting, while achieving performance similar to- or better than the original models.

The results suggests that other models that implement the idea of separate temporal and spatial convolutions, such as the Deep ConvNet model, could potentially benefit from using one spatiotemporal convolution instead of separate temporal and spatial convolutions as well.

To conclusively determine whether the combined spatiotemporal convolution significantly improves the ShallowFBCSPNet and Conformer models' performance would require training and testing the models multiple times in order to conduct a statistical test of significance.

It would also be useful to compare the models with separate temporal and spatial convolutions with the models with a combined spatiotemporal convolution on datasets with a larger number of electrodes, other types of tasks or with different amounts of training data. This could help in determining whether the results I have found are specific to the types of datasets I have been exploring or is a general pattern.

I did not succeed in creating a model that utilises the potential of both graph convolutions and transformers. Finding a way to combine these architectures (possibly also combined with CNNs) is still potentially powerful way to capture both the long term dependencies and non-euclidean relationships between channels in the EEG data. Future work could explore better ways of combining these model architectures.

Appendix

A.1 Statement on the use of generative AI

No part of this report was written by a generative AI tool. I have, however, used ChatGPT for research. Especially in the early stage where I was trying to learn the basics of deep learning and PyTorch. I found it useful for debugging tensor shape errors and to get an understanding of how some of the modules in PyTorch work. ChatGPT almost always supplies you with code, unless you are very clear in telling it not to in the prompt. I have tried using some of it, but in my experience it almost never works as intended and it would have been easier to read the documentation instead. In the experimental phase of the project I started using GitHub Copilot. I mostly used it when updating my run.py and configs.yaml file when changing which models and parameters to use. It was also helpful for assisting in generating documentation for the code.

Bibliography

[1] Sebastian Berger, Gerhard Schneider, Eberhard Kochs, and Denis Jordan. Permutation Entropy: Too Complex a Measure for EEG Time Series? *Entropy*, 19(12):692, December 2017.

[2] C Brunner, R Leeb, G R Muller-Putz, and A Schlogl. BCI Competition 2008 – Graz data set A.

[3] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering, February 2017. arXiv:1606.09375 [cs, stat].

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, June 2021. arXiv:2010.11929 [cs].

[5] Khondoker Murad Hossain, Md Ariful Islam, Shahera Hossain, Anton Nijholt, and Md Atiqur Rahman Ahad. Status of deep learning for EEG-based brain–computer interface applications. *Frontiers in Computational Neuroscience*, 16, January 2023. Publisher: Frontiers.

[6] IBM. What are Convolutional Neural Networks? | IBM.

[7] Dominik Klepl, Min Wu, and Fei He. Graph Neural Network-based EEG Classification: A Survey, December 2023. arXiv:2310.02152 [cs, q-bio].

[8] Sebastian Nagel. *Towards a home-use BCI: fast asynchronous control and robust non-control state detection*. PhD Thesis, December 2019.

[9] Yannick Roy, Hubert Banville, Isabela Albuquerque, Alexandre Gramfort, Tiago H. Falk, and Jocelyn Faubert. Deep learning-based electroencephalography analysis: a systematic review, January 2019. arXiv:1901.05498 [cs, eess, stat].

[10] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A Gentle Introduction to Graph Neural Networks. *Distill*, 6(9):e33, September 2021.

[11] Robin Tibor Schirrmeister, Jost Tobias Springenberg, Lukas Dominique Josef Fiederer, Martin Glasstetter, Katharina Eggensperger, Michael Tangermann, Frank Hutter, Wolfram Burgard, and Tonio Ball. Deep learning with convolutional neural networks for EEG decoding and visualization. *Human Brain Mapping*, 38(11):5391–5420, November 2017. arXiv:1703.05051 [cs].

[12] Yonghao Song, Qingqing Zheng, Bingchuan Liu, and Xiaorong Gao. EEG Conformer: Convolutional Transformer for EEG Decoding and Visualization. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 31:710–719, 2023.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need.

[14] Shang-Lin Wu, Chun-Wei Wu, Nikhil R. Pal, Chih-Yu Chen, Shi-An Chen, and Chin-Teng Lin. Common spatial pattern and linear discriminant analysis for motor imagery classification. In *2013 IEEE Symposium on Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB)*, pages 146–151, April 2013.

[15] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. Connecting the Dots Multivariate Time Series Forecasting with Graph Neural Networks, May 2020. arXiv:2005.11650 [cs, stat].

[16] Jin Xie, Jie Zhang, Jiayao Sun, Zheng Ma, Liuni Qin, Guanglin Li, Huihui Zhou, and Yang Zhan. A Transformer-Based Approach Combining Deep Learning Network and Spatial-Temporal Information for Raw EEG Classification. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 30:2126–2136, 2022.

[17] Mahsa Zeynali, Hadi Seyedarabi, and Reza Afrouzian. Classification of EEG signals using Transformer based deep learning and ensemble models. *Biomedical Signal Processing and Control*, 86:105130, September 2023.

[18] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. Dive into Deep Learning. 2023.