Bachelor thesis

Artificial Intelligence

A Deep Learning approach to Noise Tagging

Author:
Siebe Geurts
S4749197

First supervisor:
J. Thielen (PhD)
Artificial Intelligence Department
jordy.thielen@donders.ru.nl

Second supervisor:
Dr. M.W. Tangermann
Artificial Intelligence Department
michael.tangermann@donders.ru.nl

July 22, 2021

Abstract

Brain-computer interfaces (BCI) are systems that allow direct communication between the brain and a computer. A specific type of BCI is a code-modulated visual evoked potential (cVEP) based spelling BCI. These cVEP BCIs can use different approaches to decoding and classifying brain signals. Reconvolution, linear discriminant analysis (LDA) and deep learning are some of these approaches. In my research, I implement my own adaptation of a deep learning model and compare it against a baseline model which is LDA, and I compare both deep learning and LDA against a simple event type reconvolution. The deep learning approach (96.0% accuracy) is shown to be significantly better ($p < .05$) than the simple event type reconvolution (56.8% accuracy). Furthermore, on trial classification the deep learning approach (96.0% accuracy) does not perform significantly different ($p > .05$) from LDA (95.7% accuracy). On epoch classification deep learning (65.3% accuracy) performs significantly better ($p < .05$) than LDA (63.2% accuracy). This proves deep learning to be a useful option for cVEP BCIs with many ways to go further in research.

Contents

Chapter 1

Introduction

1.1 Brain-computer interfaces

A brain computer interface (BCI) is an interface that enables a user to control a device directly only using their brain signals. BCIs can be used to allow communication or control for users through a separate channel than muscle control. This is done by measuring brain activity and decoding it into the intentions of the user, which are then used as input for a computer. One practical way to measure brain signals for BCIs is by using electroencephalography (EEG). With EEG the user has electrodes placed on their scalp that measure the electrical activity of the brain.

1.2 Code-modulated visual evoked potentials

Communication BCIs have been making use of visual evoked potentials (VEPs) since 1992 when Sutter showed an ALS patient communicating through a BCI-controlled keyboard that recorded electrocortical activity, which was measured using intracranial electrodes [13]. A grid was shown with symbols, each individual symbol flashed with a different pattern to evoke different potential sequences for every symbol. Through these unique potential sequences, the likely attended letter was selected. The patterns Sutter used were based on binary m-sequences, these are pseudo-random bit sequences with low auto-correlation.

Since then VEPs have been used with different approaches where the majority of the BCI systems are based on frequency-modulated VEPs, which means encoding different symbols with different frequency patterns.

The approach used by Sutter was different from fVEPs because he used code-modulated VEPs (cVEPs), which means encoding different symbols with pseudo-random code patterns instead of frequency patterns. The codes used in these approaches are useful because they are optimized to be orthogonal. This means that the codes are optimized to be the least similar. The

practice of encoding symbols with code patterns for evoking cVEPs is called noise tagging. In this noise tagging paradigm the patterns of flashes are represented by binary sequences where a flash is represented by a 1, and a non-flash is represented by a 0. To be able to compare performances, a metric called information transfer rate (ITR) is used.

The ITR is a metric commonly used by BCIs which measures the amount of information conveyed. It does this based on a trial using the accuracy of the classification, time taken to classification and number of classes, which means the number of different classification options. This is a useful tool to compare the performance of different BCI systems.

Models based on cVEPs reached high ITRs such as the model by Bin et al. in 2011 with an average ITR of 108 bit/min [1], or the model presented by Spüler in 2012 which reached an average ITR of 144 bit/min which was the highest reported at that time.

1.3 Reconvolution

From using cVEPs emerged the reconvolution model presented by Thielen et al. [17]. Reconvolution is an approach that involves template generation and matching. It works by decomposing cVEPs into estimations of single event brain responses, so a response to a flash is estimated. This brain response is then used to generate templates for unseen bit sequences by assuming linearity in the brain. This means that for example, when there are two flashes happening after each other, that you can model the brain's reaction by overlapping the responses and adding them together. So using this assumption, the templates for unseen bit sequences can be generated by applying the responses. When a subject would now observe a symbol with a specific code sequence on the grid, it could be matched with the generated templates for different symbol codes and thus, a classification can be done on which symbol was observed. This model showed an average ITR of 48 bits/min [17].

1.3.1 Gold codes

The code sequences used in the reconvolution model are modulated Gold codes named after Robert Gold [4]. These modulated Gold codes are made by combining specific m-sequences. This leads to Gold codes being optimized in having the least auto-correlation and cross-correlation.

The current reconvolution model makes use of modulated Gold codes. Here the Gold codes are modulated by xor-ing them using a double frequency bit-clock. The effect of this was that the bit sequence now reflects sequences of only short flashes (i.e. 10, 100) and long flashes (i.e. 11, 110).

1.3.2 Trial versus epoch

Classification for bit sequences, such as (modulated) Gold codes, can be done on two different levels. There is trial level classification and there is epoch level classification. In the case of reconvolution, the classification is done on trial level which means that the entire bit sequence is used to make a classification on which symbol was observed.

Contrary to taking the entire bit sequence, epoch level classification uses only a single bit to make a classification on whether the subject was observing a flash or no flash. This means that instead of getting a symbol as output, a bit is presented as output which in turns is used in combination with other epochs to estimate a bit sequence which then, in turn, is used for trial classification.

1.4 Deep learning

A different approach to decoding brain activity than reconvolution, is by making use of deep neural networks (DNNs). This is called deep learning (DL) and it is a method of machine learning where often large data sets are processed with a multilayered neural network to extract high level features from the data. This is useful because the network can learn to recognise features and classify new data based on the previous data. DL approaches allow computers to learn complicated concepts or features by building them out of simpler ones [5]. Neural networks are inspired by the way the brain functions. They consist of a network of units that can be activated with the right input from either outside the network or from other units in the network. In the case of DNNs the architecture of the networks consists of different computational layers where the units activate according to their weights with a (non-linear) activation function. The way these weights are computed is by training the DNN on data and letting it learn its own representation of the data. There are many different kinds of layers one can use when building such a network, but some of the most commonly used layers are: dense layers, where every unit of one layer is connected to every unit of another layer, convolutional layers, where feature detection is done by using filters to create a feature map, and recurrent layers, which represents a memory by maintaining states from previous iterations [6].

The use of convolutional networks was adapted in different BCI paradigms where other event-related potentials are classified such as P300s. A P300 is a positive peak at around 300 ms in an event-related potential. EEGNet [7], DeepConvNet and ShallowConvNet [12] are some networks used for classifying event-related potentials, where EEGNet, in particular, showed usability of the architecture across multiple BCI data sets.

In 2019, Nagel et al. created a DNN named EEG2Code that makes use of cVEPs [10]. This network in particular reached a very high 1237 bits/min

ITR, making use of multiple convolutional layers. DL cVEP classification, and specifically EEG2Code, is as far as we know the current highest performer in spelling BCIs in terms of ITR.

1.5 Aim of the project

For my research I wanted to expand on the field of DL-based BCIs. I wanted to do this by researching the comparison between DL and reconvolution, which leads to my research question: "Can we improve the standard reconvolution approach with a DL model by means of ITR?" To answer this question, I partially replicated the research done by Nagel et al. in 2019 [10] by adapting their EEG2Code architecture and applying it to the data set used by Thielen et al. in 2021 [15]. I used LDA as a baseline model to compare the performance of my adaptation of EEG2Code and to adjust the network's parameters to fit the new data.

Besides this, I compared the trial classification of reconvolution against the trial classification of both LDA and the EEG2Code adaptation. This allows the DNN to be compared to the reconvolution method.

This research is relevant because of the replication of EEG2Code and its comparison to reconvolution. Reconvolution is a novel approach that was not yet approached in a DL way and it could benefit from many of DL's benefits.

For example, as Fahimi et al. in their 2019 research [3] showed, the usage of transfer learning in a DL BCI. The research showed that transfer learning is possible with a deep convolutional network which means that the learned representation of one subject was successfully used with a different subject. This could improve the usability of BCIs because there is less or no training time needed for switching between different users.

Another DL technique usable in BCIs is recurrency, as shown by Tortora et al. in 2020 [18]. Recurrency adds a model of memory to DNNs. As Tortora et al. suggest in movement-based BCI research, the usage of a memory model in BCIs is potentially more effective for rehabilitation devices. Since both movement and communication make use of time dependent information, this research showing promise with recurrency means that recurrency could potentially be an improvement on communication BCIs. This recurrency improvement was also shown by Maddula et al. in 2017 [8], where they used a recurrent convolutional neural network for classifyin P300 brain signals.

Besides these, reconvolution currently exists as a sequential pipeline of multiple steps which include learning the temporal responses, a spatial filter and classifier. Because of the multi-step approach, there are also multiple different optimizations happening. A DL end-to-end approach could combine these into a single optimization. Also, because DL has shown its poten-

tial in many domains including complex issues such as image classification, machine translation and natural language processing, such a model would be very adaptable to any future improvements and progress in the field of DL over the current standard reconvolution. As Minar et al. stated: "Every now and then, new and new deep learning techniques are being born, outperforming state-of-the-art machine learning and even existing deep learning techniques."[9] Therefore, I think using DL to try to improve on the standard reconvolution approach could be a very fruitful endeavor.

Chapter 2

Methods

2.1 Data

In this study I used the data set [15] from the paper by Thielen et al. in 2021 [16]. This data set contains the EEG data from 30 subjects performing an offline spelling experiment performed by Thielen et al. In the experiment participants were presented with a screen with 20 symbols. The symbols were ordered in a 4 X 5 grid on the screen as depicted in Figure 2.1. After a 1 second cue, the participants had to focus on the cued symbol for 31.5 seconds while every symbol started to flash with a pseudo-random pattern. A flash in this context is a white background for the symbol and a non-flash is a black background for the symbol. Each participant performed 100 of these trials. Since there were 20 symbols, every symbol was used 5 times per subject to result in the 100 trials. The screen had a framerate of 60 Hz.

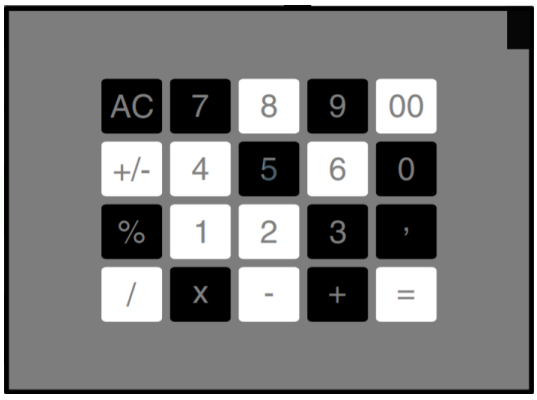

Figure 2.1: The screen showing 20 symbols in a 4 x 5 grid. Image from Thielen et al. 2021 [16].

The data recorded from the experiment is comprised of EEG recordings of 30 participants aged 19 - 62 years with an average of 25 years old. The

recordings were made with eight electrodes positioned at the channels Fz, T7, T8, POz, O1, Oz, O2 and Iz, at a sampling frequency of 512 Hz, which was later downsampled to 120 Hz. The channels POz, O1, Oz, O2 and Iz were chosen because of their location on the occipital lobe where the visual cortex is located. They are shown in Figure 2.2.

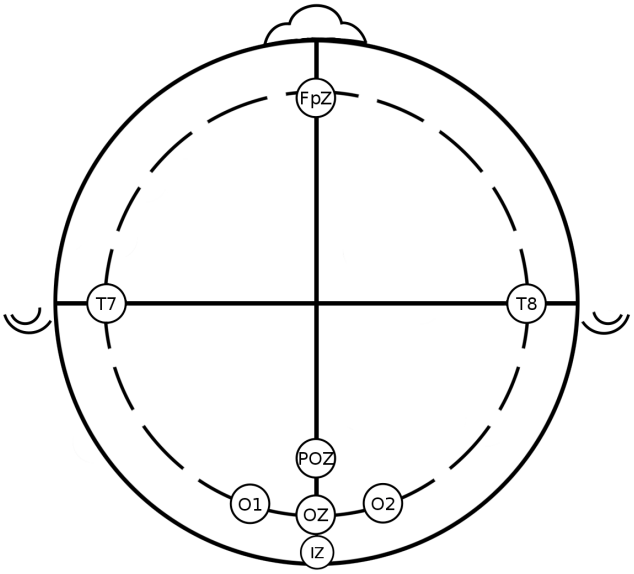

Figure 2.2: The channels shown here are the channels used for the EEG data. The most important channels are located on the occipital lobe at the visual cortex. Created by Maks Kraanenborg.

Besides EEG data, the data set contains both the flashing patterns of the symbols as bit codes where a 1 represents a flash and a 0 represents a non-flash, and the target symbol of their respective trials.

The dimensionality of data is as follows:

- EEG data: $\mathbf{X} \in \mathbb{R}^{c \times n \times k}$

  where c is 8 channels, n is 3780 samples and k is 100 trials.

- Code patterns: $\mathbf{V} \in \mathbb{R}^{l \times s}$

  where l is 252 bits and s is 20 symbols.

- Symbols: $\mathbf{Y} \in \mathbb{R}^{k \times 1}$

  where k is 100 trials.

2.2 Gold codes

The flash code patterns are based on Gold codes as developed by Robert Gold in 1967 [4]. Gold codes are binary sequences that have a small auto-correlation and cross-correlation within a set. The codes used by Thielen are modulated Gold codes where the resulting bit sequences consist of short (10, 100) and long (110, 1100) events. These are then used as short flashes and long flashes. The codes created are $2 \times (2^6 - 1) = 126$ bits long which represent 2.1 seconds. Each sequence was presented 15 times in a row which leads to 31.5 seconds per trial. In the data, the sequences are 252 bits long because of the difference between the sampling frequency and the frame rate of the screen. This was adjusted by using the data with a stepsize of 2 because $\frac{f_s}{f_r} = 2$ where sampling rate f_s is 120 and screen refresh rate $f_r = 60$.

2.3 Preprocessing

During the preprocessing stage of my research, the data was processed with a high-pass Butterworth filter of the second order at 2 Hz and a low-pass Butterworth filter of the sixth order at 30 Hz. Because of the framerate being 60 Hz, the data was downsampled from 512 Hz to 120 Hz being a multiple of 60.

2.4 Slicing

The EEG data was sliced into epochs of 32 samples which equals 267 ms of data per window. An epoch here is made up of 32 samples following an event being a flash or no flash so each epoch captures the response to this event. Since the response takes longer than the time needed to present a new event, there is overlap between epochs. To be explicit, because of the 60 Hz frame rate, it takes 1/60 seconds for a new event to occur which is 17 ms. Therefore, an epoch has $267 - 17 = 250$ ms overlap with the following epoch.

The shape of the epochs is $\mathbf{X} \in \mathbb{R}^{c \times t}$ where channels $c = 8$ and window $t = 32$. This is represented in Figure 2.3.

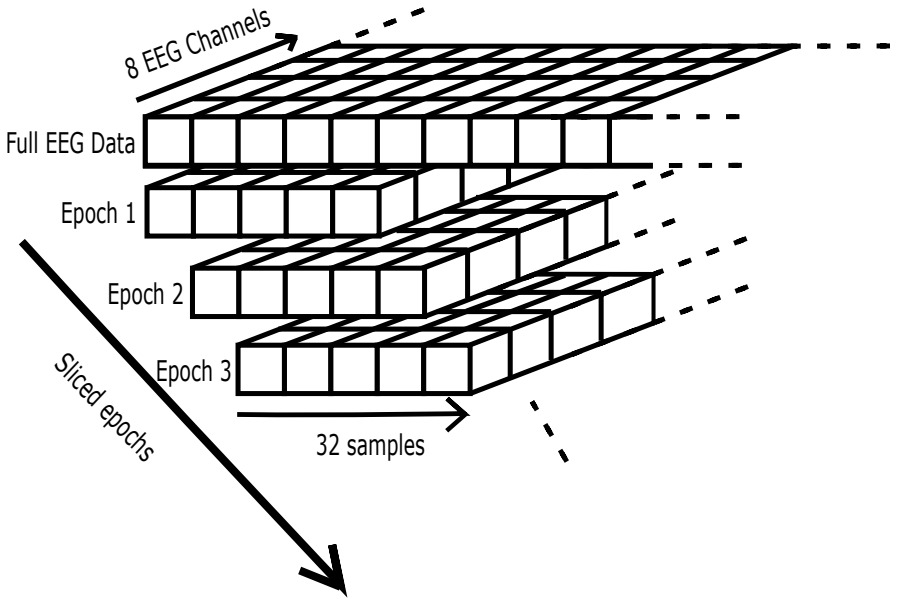

Figure 2.3: The EEG data with 8 channels are sliced into epochs by taking epochs of 32 samples by 8 channels through sliding over the data. Note that here the 32 samples are represented by 5 blocks for visibility.

This number of samples was chosen based on the reconvolution paper by Thielen et al. [17]. In this paper, the transient brain responses to a flash were shown to mostly be contained in the first 250 ms of the brain's response. This would translate to epochs of 30 samples in my implementation. However, I wanted to use epochs with their number of samples being a power of two, which would fit the network in a better manner. Therefore, I chose to use 32 samples instead of 30 samples.

In order to train, test and validate the network, the data set was split into 3 parts. The train data consisted of 80% of the data, the test data consisted of 10% of the data and the validation data also consisted of 10%. This leads to the training data consisting of 80 trials with each 1858 epochs, meaning the models are trained on 14860 epochs total. 80 trials equals $80 * 31.5 = 2520$ seconds of training data so 42.0 minutes. Calculating the same for both test and validation data leads to them consisting of 10 trials of 1858 epochs made from $10 * 31.5 = 315$ seconds of data so 5.3 minutes.

The train data was used for training the network, the test data was used to test the network during training to prevent over-fitting and check how well the network generalizes. Finally, the validation data was used as hold out set which was only used after the network was trained. This validation data was used for the final tests regarding performance.

2.5 Hardware

I performed the entire research on a device with an AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx, 2100 MHz, 4 cores, 8 logical processors. The code is provided on Gitlab *https://gitlab.socsci.ru.nl/S.Geurts/bachelorthesis*.

For the research, I used Python version 3.7.4, with PyTorch package [11] version 1.6.0 for the implementation of the deep neural network. The implementation of LDA was made with the package scikit-learn version 0.21.3. The underlying data representation was made with the NumPy package version 1.16.5 and the statistical functions were used from the SciPy package version 1.3.1. Finally, the visualization of data are created with the matplotlib package version 3.1.1.

2.6 Epoch classification

In the following sections about the model and LDA, the classification is done on epoch level. This means that the classification is on whether an epoch contains the response to a flash (1) or a non-flash (0). This can be used to classify trials, which is explained under Section 2.9.

2.7 EEG2Code

The implementation of the network is based on the EEG2Code model by Nagel and Spüler [10]. Their implementation was done in Python with the Keras library. The adaptation I made was done with the Pytorch package for practical reasons. I adapted the EEG2Code model to be able to use the data set by Thielen et al. by adjusting the parameters of the layers. The adapted network takes in data with the shape $\mathbf{X} \in \mathbb{R}^{b \times i \times c \times t}$, where batch size $b = 256$, input channels $i = 1$, EEG channels $c = 8$ and window size $t = 32$.

The network outputs data with shape $\mathbf{Y} \in \mathbb{R}^m$ with classes $m = 2$. The output can be interpreted as consisting of 2 numbers which are the probability of a non-flash versus the probability of a flash.

In a binary classification, having one output unit would be sufficient. I, however, chose for 2 output units to allow an extension to a multi class classification where $m > 2$, which could be used for an adaptation to the reconvolution approach.

The way the data changes through the network is depicted in Figure 2.4. Below that, I will give a more in-depth explanation of the individual layers. I trained the network on batches of 256 epochs at a time. The loss was calculated with cross-entropy loss and the optimizer chosen was the Adam optimizer with the learning rate of 0.001.

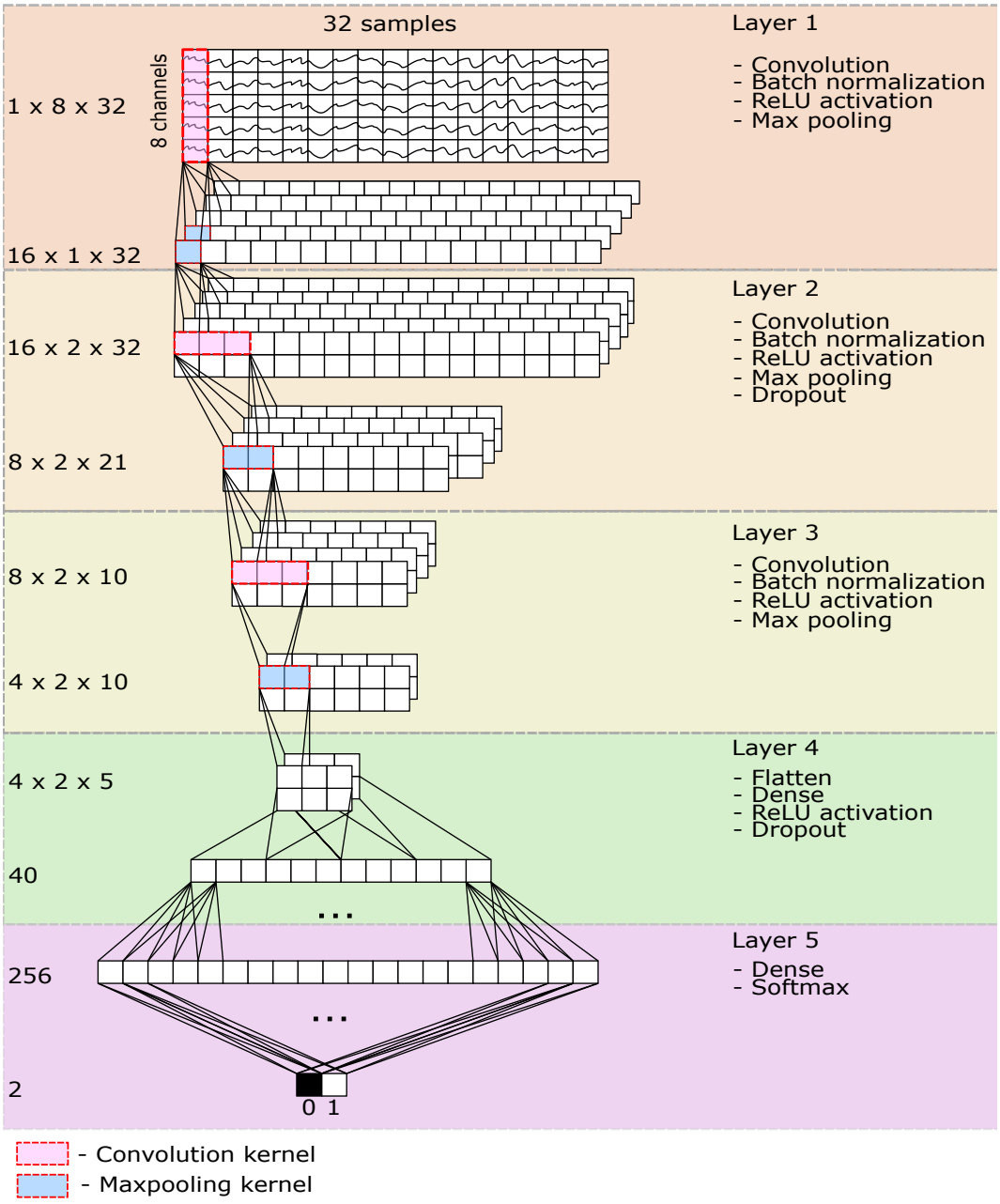

Figure 2.4: The architecture of the model is presented here. The functions contained in the layers are shown on the right. The way the shape of the data changes through the network is shown on the left side with a visual representation in the middle. Note that for the sake of readability, the dimensions of the data are not exactly the same as the visualized data dimensions. The exact dimensions of the data are represented in the text on the left. The visual shows the effect of convolution, shown in red, the effect of max pooling, shown in blue, and the effect of flattening and the dense layer shapes. The final 2 output units represent non-flash (0) and flash (1).

2.7.1 First layer

The first layer functions as a spatial filter which performs convolution on the spatial channels over time. The input shape for this layer is $\mathbf{X} \in \mathbb{R}^{b \times i \times c \times t}$, where batch size $b = 256$, input channels $i = 1$, spatial EEG channels $c = 8$ and temporal data, which is the time window, is $t = 32$. For parameters, there is one input channel because the EEG data consists of one channel. The output channels are set to 16, which translates to the convolution outputting 16 feature maps. The kernel size is set to 8×1, what follows from this is that it functions as a 1-D convolution. The kernel does the convolution only over the time dimension because all the 8 EEG channels are used by the kernel at the same time. The final parameters are padding, which is set to $(0, 0)$, and stride, set to $(1, 1)$. This leads to the shape of the data being the same as the input except for the input channels becoming $i = 16$ and the spatial channels becoming $c = 1$ so $\mathbf{X} \in \mathbb{R}^{b \times i \times c \times t}$ with batch size $b = 256$, input channels $i = 16$, spatial channels $c = 1$ and temporal data $t = 32$.

After convolution, the data is centered and normalized through batch normalization. The batch normalization function takes in the 16 feature maps as input channels and since the function should work in a constant manner without learning its parameters, I set the boolean *affine = False*. This does not change the shape of the data.

The next step in the first layer is applying a rectified linear activation unit (ReLU) to the data. This is a function that returns the input if it is above 0 and in case the input is below 0, it returns 0, so $f(x) = max(0, x)$. Since this is an activation function, it does not change the shape of the data.

The final step in this layer is to apply max pooling to the input, which downsamples the input. This is done by using a kernel to slide over the input and extracting the most salient features, which have the highest values. The parameters for this function are defined as a kernel size of $(2, 1)$, a stride of $(1, 1)$ and a padding of $(1, 0)$ to make sure the output of this function does not become too small in dimensionality. This, however, also increases the previously single spatial channel to 2 spatial channels. This happens because the max pooling is done over the spatial channels, which was reduced to 1 while the function outputs 2 channels. So after max pooling $c = 1$ became $c = 2$. The output of the complete layer has the shape $\mathbf{X} \in \mathbb{R}^{b \times o \times c \times t}$, where batch size $b = 256$, output channels $o = 16$, spatial channels $c = 2$ and time window $t = 32$.

2.7.2 Second layer

The second layer functions as a temporal filter where convolution is done over the spatially filtered time samples. The input shape of this layer is equal to the output of layer 1, meaning $\mathbf{X} \in \mathbb{R}^{b \times i \times c \times t}$, where batch size

$b = 256$, input channels $i = 16$, spatial channels $c = 2$ and time window $t = 32$. The convolution done in this layer has 16 input channels, which are the feature maps created in the previous layer. The output channels are set to be 8, meaning the 16 feature maps create 8 new feature maps during the convolution. A kernel size of $(1, 12)$ was chosen to keep the convolution in a single dimension while looking at 12 samples at a time which equals 100 ms of EEG data. The padding for the convolution was set to $(0, 0)$. After the convolution the shape of the data became $\mathbf{X} \in \mathbb{R}^{b \times i \times c \times t}$, where batch size $b = 256$, input channels $i = 8$, spatial channels $c = 2$ and time window $t = 21$.

In the same manner as the first layer, the next steps are batch normalization and ReLU activation. Besides the input channels for batch normalization being 8 instead of 16, there are no different parameters than the first layer.

The next step is again max pooling. The kernel size was here chosen as $(1, 2)$ with padding set to $(0, 0)$. This leads to the function transforming the shape of the data to $\mathbf{X} \in \mathbb{R}^{b \times i \times c \times t}$, where batch size $b = 256$, input channels $i = 8$, spatial channels $c = 2$ and time window $t = 10$.

This layer ends in a dropout function with $p = 0.5$. This is a function that randomly deactivates units to prevent over-fitting. The $p = 0.5$ parameter means that half of the units are deactivated at a time. This regularization function does not alter the shape of the data, so it stays as described: the output of this layer is $\mathbf{X} \in \mathbb{R}^{b \times o \times c \times t}$, where batch size $b = 256$, output channels $o = 8$, spatial channels $c = 2$ and time window $t = 10$.

2.7.3 Third layer

In the third layer, a second temporal convolution is performed. The input shape of this layer being as described: $\mathbf{X} \in \mathbb{R}^{b \times i \times c \times t}$, where batch size $b = 256$, input channels $i = 8$, spatial channels $c = 2$ and time window $t = 10$.

For the convolution itself, the input channels is set to 8 with output channels set to 4. This means that the 8 feature maps from the previous layer are transformed into 4 feature maps during convolution. The kernel size was set to $(1, 5)$ and this time a padding was added of $(0, 2)$ to prevent the data from being reduced too much during convolution.

Similar to the previous layers, batch normalization, ReLU activation and max pooling were performed. The batch normalization input parameter was changed to 4 from 8 but besides this, the parameters are the same as layer 2. After max pooling, the time window is again reduced which leads to the output for the entire layer being $\mathbf{X} \in \mathbb{R}^{b \times o \times c \times t}$, where batch size $b = 256$, output channels $o = 4$, spatial channels $c = 2$ and time window $t = 5$.

2.7.4 Fourth layer

The fourth layer is different from the previous layers, as no convolution is performed here. The main function of this layer is a dense layer. The input for this layer is $\mathbf{X} \in \mathbb{R}^{b \times i \times c \times t}$, where batch size $b = 256$, input channels $i = 4$, spatial channels $c = 2$ and time window $t = 5$. However, for the dense layer to receive the right data shape, the data is first flattened. This leads to the input shape now being $\mathbf{X} \in \mathbb{R}^{b \times f}$, where batch size $b = 256$ and the flattened data $f = 40$.

What follows is a dense layer with parameters input and output features set to 40 and 256, respectively. This means that this layer is made up of 256 fully connected units. This transforms the shape of the data to $\mathbf{X} \in \mathbb{R}^{b \times d}$, where batch size $b = 256$ and the dense output $d = 256$.

This dense output is then fed through a ReLU activation. After this, another dropout function is performed, its parameter set to $p = 0.5$ again for further regularization. The output of the fourth layer is $\mathbf{X} \in \mathbb{R}^{b \times d}$, where batch size $b = 256$ and the dense output $d = 256$.

2.7.5 Fifth layer

The fifth and final layer contains another dense layer where the 256 units from the previous layer are fully connected with 2 output units which represent the binary classification of flash versus non-flash. This is done with the parameters input and output features set to 256 and 2, respectively. This transforms the input shape of $\mathbf{X} \in \mathbb{R}^{b \times d}$, where batch size $b = 256$ and the dense output $d = 256$, to output shape $\mathbf{X} \in \mathbb{R}^{b \times d}$, where batch size $b = 256$ and the dense output $d = 2$. Here $d = m$, because the number of dense units d is equal to the number of classes m which is 2.

Finally to get the output of probabilities for the 2 classes, the output of the dense layer had to be normalized and this was done with a softmax function. This function transforms the 2 outputs values to both be between 0 and 1, and makes it so they add up to 1. This makes the output be interpretable as probabilities. All this leads to the output of the model being $\mathbf{X} \in \mathbb{R}^{b \times m}$, where batch size $b = 256$ and the class probabilities $m = 2$.

From this output I take resulting probability vector $\mathbf{z}$, which is then used for epoch classification.

2.8 Linear discriminant analysis

The DL model is analyzed against the baseline model which is linear discriminant analysis (LDA). LDA is a supervised classification method that predicts labels for new data points by learning a linear discriminant function. This function separates 2 classes of data points as well as possible by projecting the data onto a lower-dimensional space which maximizes

between-class covariance and minimizes within-class variance. LDA makes some assumptions about the data which are:

- The data distribution is Gaussian for both classes, which means that covariance matrices can be used to describe them.
- The covariance matrices of both classes are equal. This means that if one covariance matrix is known, the covariance matrices for other classes are also known.

The explanation I give here is heavily based on the work by Tharwat et al. [14] and by Bishop [2]. In these articles, they explain that LDA calculates its discriminant function in 3 steps.

The first step is calculating the separability between the different classes, called the between-class covariance. This is defined by Equation 2.1.

$$\mathbf{S}_B = (\mathbf{m}_0 - \mathbf{m}_1)(\mathbf{m}_0 - \mathbf{m}_1)^\top \tag{2.1}$$

To explain this formula, it calculates the between-class variance matrix $\mathbf{S}_B$, which in my case is a matrix with dimensions $\mathbf{S}_B \in \mathbb{R}^{f \times f}$, where $f = 256$ is the number of features. These are 256 because I have flattened the data set by multiplying 8 channels with 32 time samples: $8 * 32 = 256$. To get this matrix I take the class means, $\mathbf{m}_0$ and $\mathbf{m}_1$. These class means are vectors containing the mean for every feature computed from all data points belonging to the class so they both have shape $\mathbf{m}_k \in \mathbb{R}^{f \times 1}$, where $f = 256$ features for classes k. The formula calculates the separation distance between the classes.

The second step to LDA is calculating the distance between the mean and the samples of each class, which is called the within-class covariance. This is defined by Equation 2.2

$$\mathbf{S}_W = \frac{1}{2}(\mathbf{S}_0 + \mathbf{S}_1) \tag{2.2}$$

This formula calculates the within-class covariance matrix $\mathbf{S}_W$ by calculating covariance matrices $\mathbf{S}_0$ and $\mathbf{S}_1$ which are the calculated by Equation 2.3 for both class 0 and class 1. Both $\mathbf{S}_0$ and $\mathbf{S}_1$ are matrices with shape $\mathbf{S}_k \in \mathbb{R}^{f \times f}$, where $f = 256$ features for k classes. Because of this, the within-class covariance matrix $\mathbf{S}_W$ has shape $\mathbf{S}_W \in \mathbb{R}^{f \times f}$ with $f = 256$ features.

$$\mathbf{S}_k = \frac{1}{N-1}\mathbf{X}_k\mathbf{X}_k^\top \tag{2.3}$$

This is the sample covariance matrix for each class. Here N is the number of data points in class k. $\mathbf{X}_k$ is the data matrix with in my case dimensions $\mathbf{X}_k \in \mathbb{R}^{f \times f}$ with $f = 256$.

The third step to LDA is calculating the transformation vector $\mathbf{w} \in \mathbb{R}^f$, with $f = 256$ features, by making use of Fisher's criterion as shown in Equation 2.4.

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} \tag{2.4}$$

Here $J(\mathbf{w})$ is optimized by maximizing $\mathbf{S}_B$ and minimizing $\mathbf{S}_W$. To find the maximum for this $J(\mathbf{w})$ differentiation can be done with respect to $\mathbf{w}$ as in Equation 2.5.

$$\mathbf{w} = \mathbf{S}_W^{-1}(\mathbf{m}_0 - \mathbf{m}_1) \tag{2.5}$$

What this leads to is a projection of the data onto a lower dimensional space that separates the different class as well as possible. The result of this is a parameter $\mathbf{w}$ which can be used in a discriminant function $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$. The last parameter needed for this is a scalar b which is the bias. The bias can be calculated as in Equation 2.6.

$$b = -\frac{1}{2}\mathbf{w}^T(\mathbf{m}_0 + \mathbf{m}_1) \tag{2.6}$$

With $\mathbf{m}_1$ and $\mathbf{m}_2$ being the means of the 2 classes. To reach an output probability result $\mathbf{z}$ in similar manner as DL does, a softmax function is applied to the output of the function $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$. This yields probability vector $\mathbf{z}$.

The $\mathbf{z}$ probabilities from both the DL network and LDA are then used for classification on epoch level, so $\mathbf{z}$ leads to a classification for flash (1) or non-flash (0).

2.9 Trial classification

After epoch classification with either DL or LDA, the probability $\mathbf{z}$ per epoch can be used to predict a code sequence.

For the model I did this by letting the model predict $\mathbf{z}$ for every epoch in the validation set, then concatenating them into sequences to get predicted probabilities for codes.

For LDA, I passed the validation set through the discriminant function learned from the training data and the resulting probabilities I concatenated per epoch into probability sequences.

Then I used these code sequences to classify per trial to which symbol the subject was looking, by calculating the Pearson's correlation. This was done between the predicted code sequence $\mathbf{z}$ and each of the code sequences from $\mathbf{V}$. Here $\mathbf{V}$ are the code sequences used to tag each of the symbols in the experiment. Pearson's correlation is defined as in Equation 2.7

$$\hat{y} = \max_i \frac{\mathbf{z}^T \mathbf{V}_i}{\sqrt{\mathbf{z}^T \mathbf{z} * \mathbf{V}_i^T \mathbf{V}_i}} \tag{2.7}$$

Here, the probability vector is $\mathbf{z}$ and every code sequence $\mathbf{V}_i$ where i is the number of symbol codes. Using this correlation coefficient, I computed the most likely observed code $\hat{y}$. This leads to a classification of which symbol was observed. This classification strategy was also used for the reconvolution approach which is described in the following section.

2.10 Reconvolution

Reconvolution is mostly adapted from the model presented by Thielen et al. in 2015 [17].

Reconvolution is a 2 step approach where, in the first step, full brain responses to code sequences are decomposed into brain responses to individual events. The current reconvolution model uses duration events which are defined as either a short flash (10, 100) or a long flash (11, 110). This makes the assumption that brain processes behave both linearly and non-linearly. It assumes linearity since it assumes the response of a sequence of flashes is built up from adding the responses to the time-shifted individual flashes. However, since the model uses a different response for a long event and does not just shift and add the response of 2 short events to get the response, it assumes the brain handles both differently so non-linearly.

My implementation of reconvolution uses a different event type, namely simple events. This means that the only event used is a flash (1) as opposed to short flash (10, 100) or a long flash (11, 110). In the further explanation of reconvolution, the simple event type reconvolution is used. The simple event reconvolution model only assumes linearity of the brain and does not assume non-linearity of the brain.

The second step in reconvolution is the generation of templates. What happens here is that the response to the individual event is used to generate template responses to each of the unseen code sequences.

To explain the first step, the decomposition of the full sequence for a single channel for a single trial is defined as Equation 2.8.

$$\tilde{\mathbf{x}}(t) = \sum_{\tau=1}^{L} l(t)\mathbf{r}(t-\tau) \tag{2.8}$$

Here, $\tilde{\mathbf{x}}(t)$ is the full response at time t; $l(t)$ is a function that indicates whether there was a flash at time t. The value of $l(t)$ is 1 if there is a flash at time t, otherwise it is 0. $\mathbf{r}(t)$ is the response to a flash at time t. Finally, L is the length of $\mathbf{r}$.

This equation can be written in a matrix notation as in Equation 2.9.

$$\tilde{\mathbf{x}}(t) = \mathbf{M}\mathbf{r} \tag{2.9}$$

In this equation $\mathbf{M}$ is a structure matrix that has a value of 1 at all rows t in column 1 where at time t a flash happened where in the columns 2 to L, the ones shift down to $t + L$. This results in a matrix of diagonal ones as in Figure 2.5.

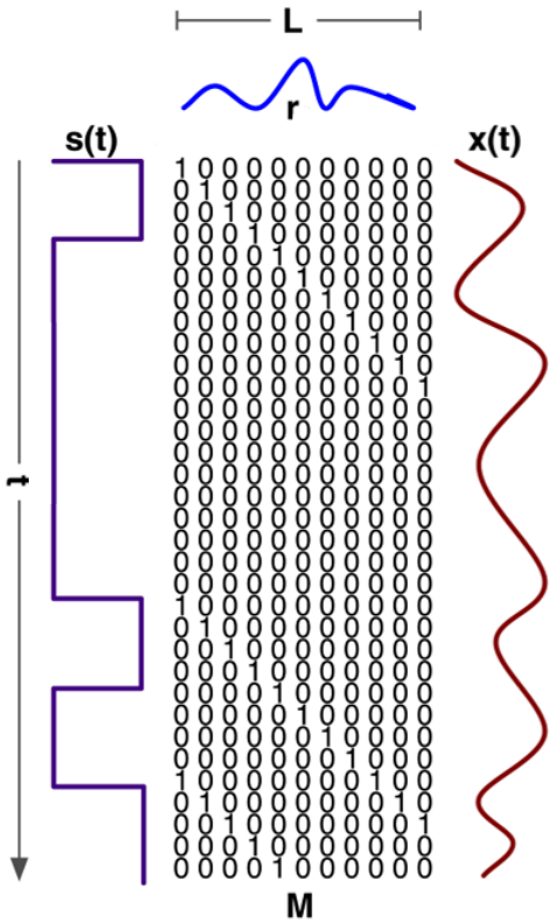

Figure 2.5: The structure matrix $\mathbf{M}$ for the event response $\mathbf{r}$ is created with the code sequence $\mathbf{s}(t)$. Whenever $\mathbf{s}(t)$ has an event, the structure matrix contains a 1 in the first column which is repeated down while shifting to the right once every column. Image adapted from Thielen et al. [17].

To then generalize the model, multiple single-channel single trials can be concatenated which leads to a linear regression problem defined in Equation 2.10.

$$\mathbf{X} = [\mathbf{x}_1 ... \mathbf{x}_k] = [\mathbf{M}_1 ... \mathbf{M}_k]\mathbf{r} = \mathbf{M}\mathbf{r} \tag{2.10}$$

Here k single channel single trials are concatenated in $\mathbf{X}$. $\mathbf{M}$ is the concatenation of all resulting $\mathbf{M}_k$ from the k single channel single trials. To find the response $\mathbf{r}$, the linear equation has to be solved, which can be done by adjusting the formula to Equation 2.11.

$$\mathbf{r} = \mathbf{M}^{+}\mathbf{X} \tag{2.11}$$

Where $\mathbf{M}^{+}$ is the pseudo inverse of $\mathbf{M}$. After this is calculated, the response to individual events is learned which can now be used for template generation in step 2.

Step 2 of the reconvolution process is generating templates $\mathbf{T}_V$ where V represents the code sequences that are in the codebook. This is done by multiplying the structure matrix for the codes in V by the response vector $\mathbf{r}$ to get the templates $\mathbf{T}_V$. These templates are calculated as in Equation 2.12.

$$\mathbf{T}_V = \mathbf{M}_V\mathbf{r} \tag{2.12}$$

Similar to LDA and DL, the final step is using Pearson's correlation to classify the code sequence as in Equation 2.13

$$\hat{y} = \max_V \frac{\mathbf{X}^T\mathbf{T}_V}{\sqrt{\mathbf{X}^T\mathbf{X} * \mathbf{T}_V^T\mathbf{T}_V}} \tag{2.13}$$

Here $\mathbf{X}$ is the input data, $\mathbf{T}_V$ is the template response for code V and these are maximized through this formula to find the best fitting template.

This is reconvolution method as used in the 2015 paper by Thielen et al. The implementation I used for my research and comparison with other models was the adapted version from their 2021 paper. In this new iteration, the estimation of the response vector is performed within a canonical correlation analysis which also introduces the usage of multiple channels into reconvolution. In the next section, this will be explained.

2.11 Canonical correlation analysis

Reconvolution only uses a single spatial channel so to still be able to use multiple channels in reconvolution, a spatial filter needs to be added. This spatial filter is obtained by the use of canonical correlation analysis (CCA). This spatial filter reduces multiple channels, as in the case of this research there are 8 spatial channels, to a single channel. The filter functions by adding a weight to every channel and multiplies the data from the channels with their respective weights after which they are combined into 1 channel. The adding of weights is relevant because it places value on the channels such that more important and informative channels weigh heavier than the

less informative channels. As mentioned in the previous section, CCA also returns a response vector with the associated response to events.

The way CCA works is explained through Equation 2.14.

$$\underset{\mathbf{w},\mathbf{r}}{max}\rho(\mathbf{w}^T\mathbf{X}, \mathbf{r}^T\mathbf{M}) = \frac{\mathbf{w}^T\mathbf{X}\mathbf{M}^T\mathbf{r}}{\sqrt{\mathbf{w}^T\mathbf{X}\mathbf{X}^T\mathbf{w} * \mathbf{r}^T\mathbf{M}\mathbf{M}^T\mathbf{r}}} \tag{2.14}$$

It maximizes a spatial filter $\mathbf{w}$ and response vector $\mathbf{r}$ by applying them to both the concatenated structure matrices $\mathbf{M}$ and the input data $\mathbf{X}$. This results in a spatial filter that can be applied to single trials and a response vector that can be used to predict template responses with structure matrices for the codes. From this, template matching is done in the same manner as described in the previous section which results in a classification for best fitting template and thus, the attended symbol.

2.12 Evaluation

To evaluate each of these approaches, the accuracy was calculated. This was done for all 30 subjects on LDA and DL on epoch level, and for all 30 subjects on LDA, DL and reconvolution on trial level. The accuracy was calculated as a percentage of correct classifications.

To determine which results are significant, I used a Wilcoxon signed-rank test. This is a non-parametric statistical test that compares pairs. I chose for the Wilcoxon signed-rank test because the data does not follow a normal distribution since the accuracies cannot go above 100% and the accuracies tend to be on the higher end.

For epoch level significance, the compared pairs are the 30 subject accuracies for LDA and the 30 subject accuracies for the DL model. For trial level significance, there are 3 pairs of 30 accuracies, namely LDA-DL, LDA-reconvolution and DL-reconvolution. This led to 3 Wilcoxon tests.

Because more than 2 models are compared in significance on trial level, a correction needs to be done. For this, I chose the Bonferroni correction.

Besides these significance tests, I will evaluate the performance of each model by means of ITR. ITR is defined as in Equation 2.15 and Equation 2.16.

$$B = logN + PlogP + (1-P)log\frac{1-P}{N-1} \tag{2.15}$$

$$ITR = B/T \tag{2.16}$$

Here B is the average bits of information in each selection, P is the selection accuracy and N is number of possible classifications. Finally, to reach the ITR, B is divided by T which is the time it takes to reach a classification in minutes.

Chapter 3

Results

Whether the reconvolution approach can be improved by using DL is dependent on their performances. To first identify the performance of the DL model, a comparison is made with LDA on both epoch and trial level. The results were gathered through training and testing on all 30 subjects. After training both models with the training set and testing the DL model with the test set, the final accuracies for both approaches were calculated from the validation set. This was done for both epoch and trial level.

To also compare the performance of reconvolution with the two other approaches, the trial level comparison was expanded with the accuracies for the reconvolution model.

3.1 Performance

3.1.1 Epoch classification

The comparison between the 2 models is shown below. Table 3.1 shows the mean and standard deviation and Figure 3.1 visualizes the descriptions. When performing the Wilcoxon signed-rank test on the accuracies of LDA and DL, it showed $p = 0.003$. Since $p < .05$, this indicates that the sets are significantly different, so it is safe to assume that the DL model performs significantly better on epoch level. The ITR on epoch level for the LDA with an accuracy of 63.2% translates to 11.5 bits/min, which follows from $N = 2$ classes, $P = 0.632$ selection accuracy and $T = 267.8$ from the length of one epoch + 1/60 inter-stimulus interval. The ITR for the DL with an accuracy of 65.4% translates to 15.5 bits/min, from $N = 2$ classes, $P = 0.654$ selection accuracy and $T = 267.8$ from the length of one epoch + 1/60 inter-stimulus interval.

Table 3.1: The mean and standard deviation of the LDA and DL models.

Epoch Classification	LDA	DL
Mean Accuracy (%)	63.2	65.3
Standard Deviation (%)	±3.6	±5.1
ITR (bits/min)	11.5	15.5
Wilcoxon (p-value)	0.003	

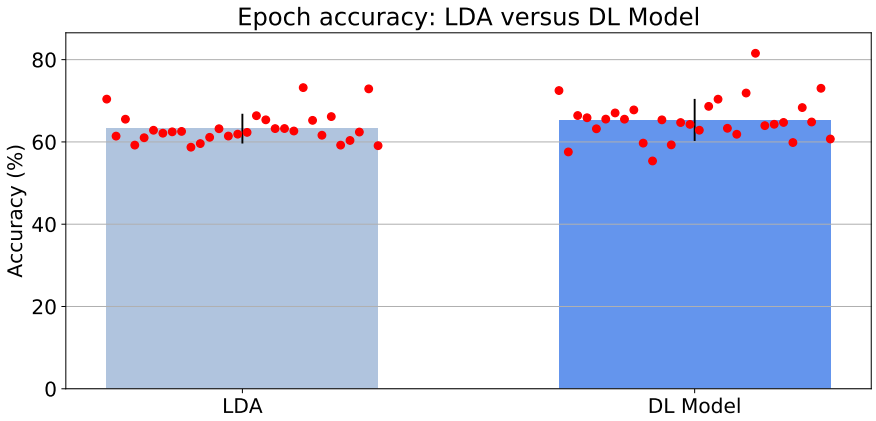

Figure 3.1: Graph shows the mean accuracies for LDA and DL with the standard deviation as a line. The actual data points are scattered over the bar graphs to show the distribution.

3.1.2 Trial classification

The comparison on trial level is done with 3 pairs. Its description can be found in Table 3.2, the visualization of the description can be seen in Figure 3.2.

This comparison is also done with a Wilcoxon signed-rank test. However, a correction has to be made because of the multiple p-values which was done with a Bonferroni correction. This counteracts the issue with multiple comparisons. As can be seen in Table 3.2, LDA with an accuracy of 95.7% and DL with an accuracy of 96.0% seem to no be significantly different from each other ($p > .05$). LDA showed a significantly different ($p < .05$) performance from reconvolution which had an accuracy of 56.8%. DL also performed significantly different ($p < .05$) from reconvolution. This means that both LDA and DL perform better than reconvolution.

The ITR on trial level is calculated for all 3 models with the same $N = 20$ classes and the same $T = 0.542$ time until classification which follows from 31.5 seconds trial + 1-second inter-trial interval where T is in minutes. The

only thing different for the models is the selection accuracy P. For LDA, the trial ITR with $P = 0.957$ leads to 7.2 bits/min. For DL, the trial ITR with $P = 0.960$ leads to 7.2 bits/min and for reconvolution, the trial ITR with $P = 0.568$ leads to 2.8 bits/min.

Table 3.2: The mean and standard deviation of the LDA and DL models.

Trial Classification	LDA	DL	Reconvolution
Mean Accuracy (%)	95.7	96.0	56.8
Standard Deviation (%)	± 6.2	± 6.2	± 24.2
ITR (bits/min)	7.2	7.2	2.8
Wilcoxon (p-value) Bonferroni adjusted			
LDA	-	0.952	$< .001$
DL	0.952	-	$< .001$
Reconvolution	$< .001$	$< .001$	-

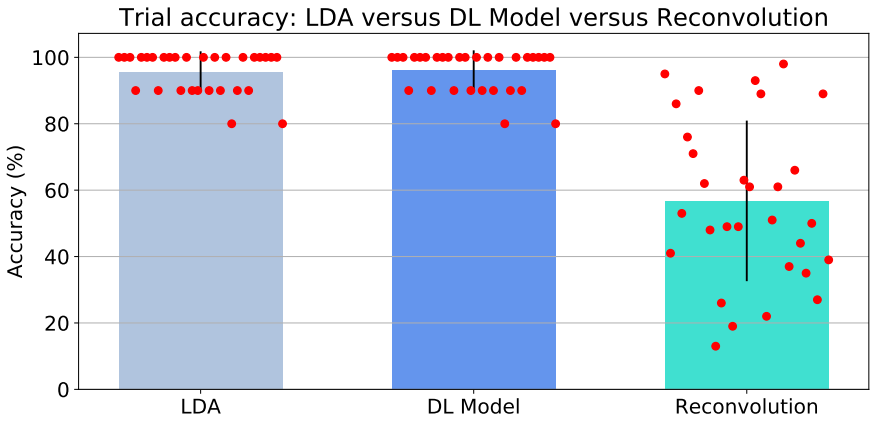

Figure 3.2: Graph shows the mean accuracies for LDA, DL and reconvolution models with the standard deviation as a line. The actual data points are scattered over the bar graphs to show the distribution.

3.2 Training

Training curves, which include the epoch accuracy and loss for both training data and testing data, for the DL model are shown in Figure 3.3. These curves were created by iterating 25 times through the model where after each iteration, the model weights were adjusted. In the graphs, it can be seen that most of the time, the DL already starts at a somewhat high accuracy

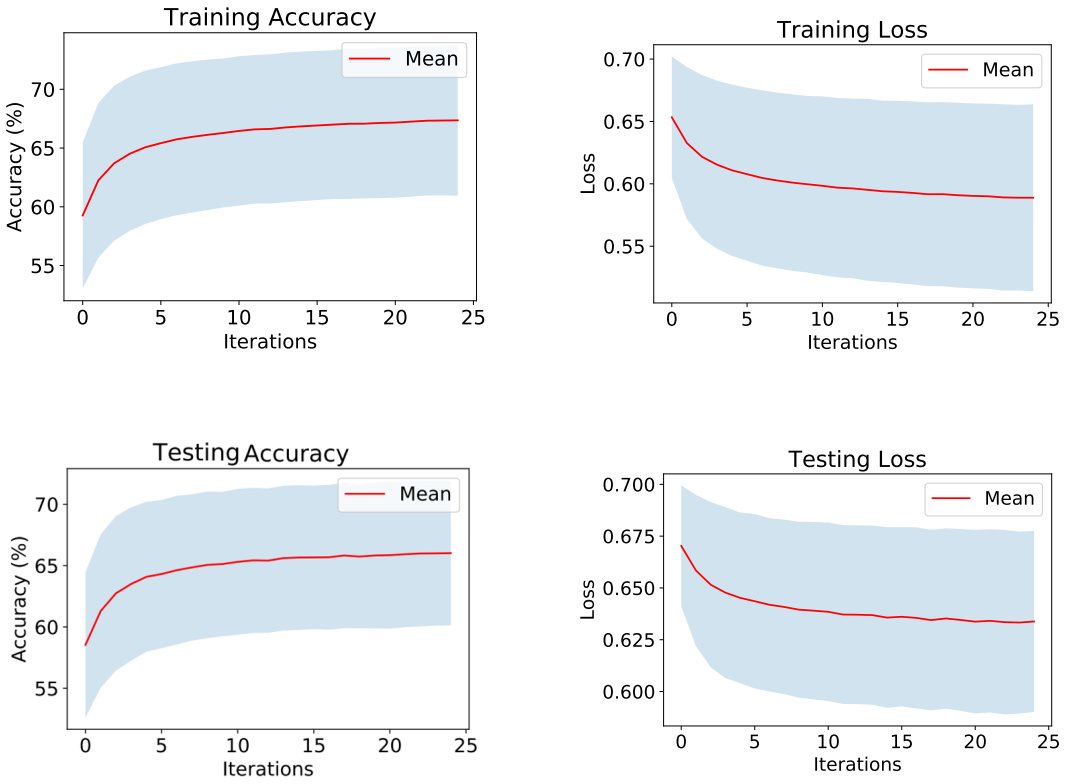

Figure 3.3: The training accuracy and loss, together with the test accuracy and loss. Trained with 25 iterations. The means are plotted in red with the standard deviation of the curves over the 30 subjects in blue/gray.

after which it increases a few percent but not that much since it is already quite good.

3.3 Summary

What the results show is that at epoch level, the DL model is significantly better than the LDA. Besides this, it shows that at trial classification level the DL model is not significantly different from the LDA, but both DL and LDA are significantly better than reconvolution.

Chapter 4

Discussion

In this research I have tried to pave the way for DL to be introduced into the reconvolution framework by comparing the performances and showing that DL is significantly better at binary classification than both the baseline LDA model and the single event reconvolution model. For this, I implemented and adapted a DL model named EEG2Code originally made by Nagel et al. in 2019 [10], which I tested on a new data set gathered by Thielen et al. for their research in 2021 [16]. The original EEG2Code network was compared against a baseline ridge regression, for which I chose an alternative being LDA. This comparison was performed on 2 different levels, being epoch and trial level. This research shows that DL can at the very least, improve simple event type reconvolution by means of ITR.

To compare my adaptation of EEG2Code to the original EEG2Code as proposed by Nagel et al. in 2019 [10], I want to look at the performance and ITR. In the Nagel et al. paper the offline experiment results show an accuracy of 74.9% and an ITR of 701.3 bits/min, while my model showed an accuracy of 96.0% and an ITR of 7.2 bits/min. The accuracy is higher in my adaptation of the model but the ITR is many times lower. Since ITR is about amount of information conveyed, the EEG2Code model seems to be better. This is, however, a somewhat biased comparison because of the differences in testing. Regarding the results for my model in ITR being very low, this is a consequence of the usage of the entire trial plus inter-trial time meaning 32.5 seconds. This influences the ITR through the formula to result in very low bits/min. Nagel et al. used a classification time of 1/60 seconds against my 32.5 seconds. A lower classification time could lead to a higher ITR if the selection accuracy does not decrease too much from a lower classification time. To make a better comparison, future work could look into the time needed to classify a trial. A curve could be generated in which different classification times are set against their selection accuracy to see where the optimum ITR lies. If the trial is reduced to for example 2.1 seconds, which is precisely one bit-sequence, and the accuracy would

remain the same, then it would result in an trial ITR for the DL model of 75 bits/min. Comparing to the 7.2 bits/min from this research it looks like a much better model even though the model is the same.

Despite the comparison with EEG2Code being biased, the ITR is calculated in the same way for every model I tested, it does serve as comparison against each of the models in this research.

To delve further into the comparison between LDA and DL, and one of its key differences, it would be interesting to look at the impact different amounts of training data would make. Since DL typically needs large data sets to learn its parameters, a smaller data set could improve training time for a DL but could also show that the analytical approach of LDA outperforms DL. On the opposite side, the non-linear approach of DL could prove to outperform LDA given enough data to train on which does create a longer training time. The pay-off between training time and accuracy could be something worth researching in future work.

Based on the results I gathered throughout the research, I came to the conclusion that it could be beneficial to the validity of this research if some steps are taken to improve the comparisons performed. One of these steps could be using k-fold cross-validation while training the different models. This means that the data is split into k different parts and then the model is trained and tested multiple times but each time the test data is a different part of the k parts. This makes sure that all the data is used an equal amount in training and testing which makes sure that there is no bias because of the specific test part used.

Besides expanding the research I performed, going further with it could prove very interesting. A potentially very insightful extension of this research would be to delve into what the DL network learned by taking a look at the feature maps and representations in the model. Because of the data driven approach, the DL model learns its own features from the data which could include novel features that we have not yet seen before. This could potentially be relevant for the neuroscientific aspect of BCIs. Visualization of feature maps could be handled in a similar way to how Shirrmeister et al. did it with their ConvNet [12]. In the case of the learned spatial filter, this could be visualized through a topographic plot of the EEG channels with the weight of each of the used channels as gradients over the scalp. For the temporal filters, input-feature unit-output correlation maps could be created where the unit outputs can be visualized for different input features. This could show sensitivity to novel features in brain response data that can further the neuroscientific field and could potentially be used in BCIs.

A different but also very fitting next step for follow-up research would be to adapt the DL model to duration event classification such as reconvolution does. In this research the reconvolution model has been adapted to a simple event type classification while it currently uses duration event types. Using simple event types makes it underperform in comparison with its du-

ration event type variant. This could explain the low performance of the reconvolution model in my research. Since DL outperforms reconvolution in this instance, it could suggest that it could also outperform reconvolution when adapted to duration event type classification. The way I implemented my model is already very flexible and only minor changes would need to be made to the final layer and the way the data is labeled.

To go a little further with this, the way this could be done is by extending the final dense layer which currently outputs data with shape $\mathbf{y} \in \mathbf{R}^m$ with classes $m = 2$. This should be adjusted to reflect the duration event types which is changing to $m = 3$ so 3 output units. These 3 output units reflect the short flash, long flash and non-flash. Current reconvolution does not make use of the non-flash event. My DL implementation does make use of it which is why 3 output units are used. The data itself would then have to be labeled differently to reflect the 3 possible targets. Since these events would be non flash (0), short flash (10) and long flash (11), the epochs should be labeled 0 for non flash, 1 for a short flash and 2 for a long flash. In this case, the second 1 from the long flash (11) would be set to 0, so (11) would become (20), to keep the events separated in the right way. Then after training the network on this, the epoch classification would yield probabilities for each of the 3 events because softmax also works on multi-class activation.

To then go from epoch classification to trial classification would have to be done differently from binary classification. One way to classify trials when epoch output is 3 probabilities, is by creating a matrix. This matrix consists of for every time point t, the 3 probabilities as follows:

$$\begin{bmatrix} 0.1 & 0.7 & 0.2 \\ 0.0 & 0.3 & 0.7 \\ 0.6 & 0.2 & 0.2 \\ : & : & : \end{bmatrix}$$

Here the first row denotes the probability for non-flash, short flash and long flash at the first time point. The second row is for the second time point, and so on.

Then in a similar manner as reconvolution does, creating a structure matrix for every template where there are only zeros except for where there was an event:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ : & : & : \end{bmatrix}$$

Here the first row denotes that there was a short flash at the first time point, the second row denotes that there is a long flash at the second time point, and so on. This would be for one of the templates, so there would be 20 structure matrices.

The next step consists of combining the probability matrix with every template structure matrix, which would lead to 20 matrices as such:

$$\begin{bmatrix} 0 & 0.7 & 0 \\ 0 & 0 & 0.7 \\ 0.6 & 0 & 0 \\ : & : & : \end{bmatrix}$$

To then pick the right template, you can simply add the probabilities per matrix and choose the matrix template that has the highest probability of all templates. For this small example matrix it would be: $0.7 + 0.7 + 0.6 = 2.0$. Doing this for every matrix gives 20 values and the maximum value leads to classification of the symbol corresponding to that template.

This research and its possible future extensions could provide an adaptable basis for improving the field of BCIs. Showing DL to be useful in the cVEP paradigm opens the door to extend the usefulness of these models by applying DL based improvements.

Besides the possibility of outperforming analytical methods due to the non-linear approach of DL, these improvements could include improving usage between different users with transfer learning, as shown by Fahimi et al. in 2019 [3]. As shown by Madulla et al in 2017 [8] and Tortora et al. in 2020 [18], using recurrency could improve the usability of P300 and movement BCIs as well so applying recurrency to cVEP BCIs could potentially show improvements as well.

Chapter 5

Conclusions

In conclusion, I have shown DL to be a useful alternative to reconvolution when considering simple event types. Besides this, I have proven that DL on its own can be at least as good as LDA at trial classification and better than LDA at epoch classification. This shows that DL can be used for cVEP based BCIs.

To reflect on the research question: "Can we improve the standard reconvolution approach with a DL model by means of ITR?" For the reconvolution model with simple event types, the answer is yes, it can be improved with a DL model. To answer the question with regards to the standard duration event type reconvolution model, there would need to be an extension on this research. So the result is a partial answer to the research question. I think that with this research the opportunity to expand on DL based cVEP BCIs is created and that there are many ways to go from here.

Chapter 6

Acknowledgements

I would like to express my gratitude to my supervisors, Jordy Thielen and Michael Tangermann. I am very grateful for having been able to work on such an interesting project. Even though it was a difficult time during the pandemic, I believe they have done great in supervising my research. From giving feedback, to sharing the tips and tricks of the field, to overall supporting me and my research, I believe they went above and beyond the standard and I am very grateful for it. Besides my supervisors, I would sincerely like to thank my thesis group: David Teulings, Elise Lems and Julia Janssen. Without you, I would not have come this far. Besides the social interaction, giving each other feedback and working together, I have received the necessary mental support from you which kept me going. Thank you.

Bibliography

[1] Guangyu Bin, Xiaorong Gao, Yijun Wang, Yun Li, Bo Hong, and Shangkai Gao, *A high-speed bci based on code modulation vep*, Journal of neural engineering **8** (2011), 025015.

[2] Christopher M. Bishop, p. 186–193, Springer, 2006.

[3] Fatemeh Fahimi, Zhuo Zhang, Wooi Boon Goh, Tih-Shi Lee, Kai Keng Ang, and Cuntai Guan, *Inter-subject transfer learning with an end-to-end deep convolutional neural network for eeg-based bci*, Apr 2019.

[4] R. Gold, *Optimal binary sequences for spread spectrum multiplexing (corresp.)*, IEEE Transactions on Information Theory **13** (1967), no. 4, 619–621.

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

[6] Martin Isaksson, *Four common types of neural network layers*, Jun 2020.

[7] Vernon J Lawhern, Amelia J Solon, Nicholas R Waytowich, Stephen M Gordon, Chou P Hung, and Brent J Lance, *EEGNet: a compact convolutional neural network for EEG-based brain–computer interfaces*, Journal of Neural Engineering **15** (2018), no. 5, 056013.

[8] Ramesh Maddula, Joshua Stivers, Mahta Mousavi, Sriram Ravindran, and Virginia de Sa, *Deep recurrent convolutional neural networks for classifying p300 bci signals.*, GBCIC **201** (2017).

[9] Matiur Rahman Minar and Jibon Naher, *Recent advances in deep learning: An overview*, CoRR **abs/1807.08169** (2018).

[10] Sebastian Nagel and Martin Spüler, *World's fastest brain-computer interface: Combining eeg2code with deep learning*, PLOS ONE **14** (2019), no. 9, 1–15.

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala, *Pytorch: An imperative style, high-performance deep learning library*, Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035.

[12] 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** (2017), no. 11, 5391–5420.

[13] Erich E. Sutter, *The brain response interface: communication through visually-induced electrical brain responses*, Journal of Microcomputer Applications **15** (1992), no. 1, 31–45, Special Issue on Computers for Handicapped People.

[14] Alaa Tharwat, Tarek Gaber, Abdelhameed Ibrahim, and Aboul Ella Hassanien, *Linear discriminant analysis: A detailed tutorial*, Ai Communications **30** (2017), 169–190,.

[15] Jordy Thielen, Pieter Marsman, Jason Farquhar, and Peter Desain, *From full calibration to zero training for a code-modulated visual evoked potentials brain computer interface*, 2021.

[16] Jordy Thielen, Pieter Marsman, Jason Farquhar, and Peter Desain, *From full calibration to zero training for a code-modulated visual evoked potentials brain computer interface*, (2021).

[17] Jordy Thielen, Philip van den Broek, Jason Farquhar, and Peter Desain, *Broad-band visually evoked potentials: Re(con)volution in brain-computer interfacing*, PLOS ONE **10** (2015), no. 7, 1–22.

[18] Stefano Tortora, Stefano Ghidoni, Carmelo Chisari, Silvestro Micera, and Fiorenzo Artoni, *Deep learning-based BCI for gait decoding from EEG with LSTM recurrent neural network*, Journal of Neural Engineering **17** (2020), no. 4, 046011.