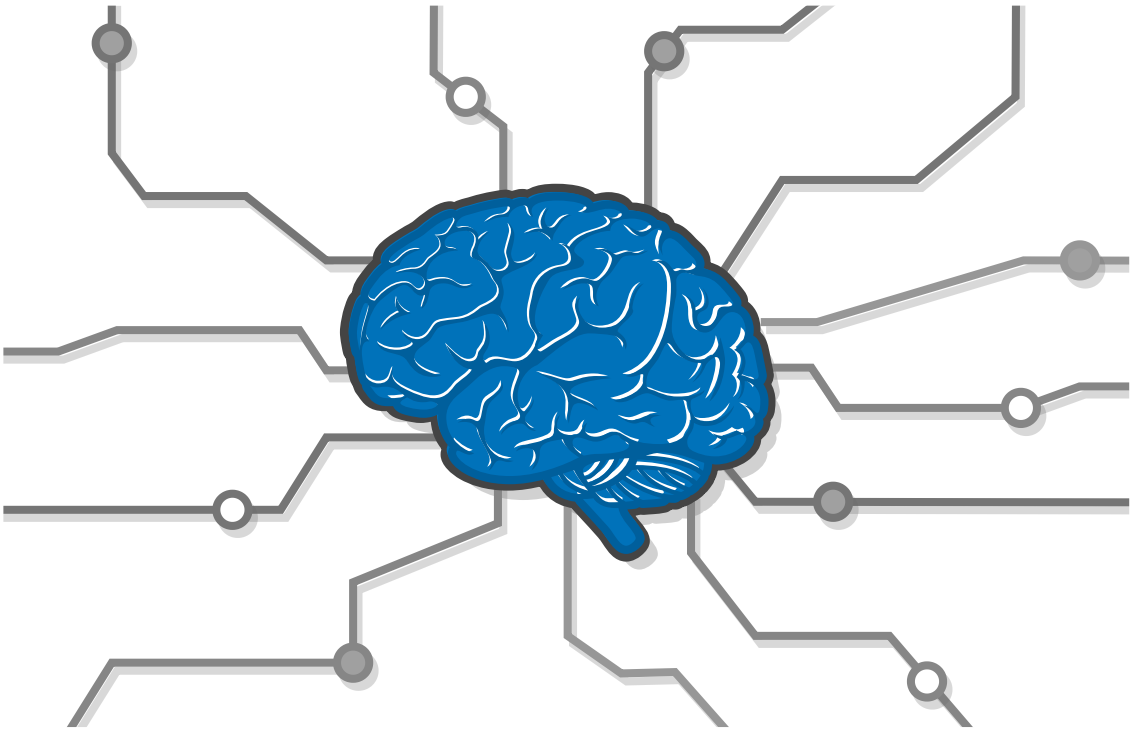


Machine Learning for Brain Activity Analysis

Deep Learning Methods for Improving Essential Hypertension Prediction Based on Magnetoencephalography Data

Master's thesis in Data science and AI and Complex adaptive systems

David Hall
Elias Sundqvist

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Machine Learning for Brain Activity Analysis

Deep Learning Methods for Improving Essential Hypertension Prediction Based on Magnetoencephalography Data

David Hall
Elias Sundqvist

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2022

Machine Learning for Brain Activity Analysis
Deep Learning Methods for Improving Essential Hypertension Prediction Based on Magnetoencephalography Data
David Hall
Elias Sundqvist

Supervisor: Affiliated Scientist Justin Schneiderman, Microtechnology and Nanoscience, Quantum Device Physics Laboratory
Supervisor: Kevin Andersson, Syntronic Research and Development AB
Examiner: Professor Paolo Monti, Electrical Engineering

Master's Thesis 2022
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The brain icon is modified and attribution-free, the rest is original work.

Typeset in LaTeX
Gothenburg, Sweden 2022

Abstract

Cardiovascular disease is a wide-spread problem which is strongly linked to essential hypertension (high blood pressure). Previous research has demonstrated a correlation between arousal-induced elevated blood pressure and invasive measurements of muscle-sympathetic nerve activity (MSNA). Such blood pressure responses are likely to integrate over a lifetime into essential hypertension. Research has also shown that there is a correlation between MSNA and magnetoencephalography (MEG) brain activity during stressful situations. MEG is a non-invasive method for measuring brain activity. Two previous Master's thesis projects have tried to make a classifier from this data with limited success. This Master's thesis project aims to improve classification results by resolving issues in previous theses and by trying new approaches to the problem. The main hypotheses investigated for explaining low classification accuracies in previous attempts are: 1) Inappropriate data split, 2) Lack of data preprocessing/augmentation, 3) Inappropriate model choice. A fourth hypothesis, insufficient data, was considered as potentially being the biggest issue. It was, however, out of the scope of this thesis because of it being infeasible to collect enough data in the given time-frame and because medical professionals would have to be included to perform the measurements. To test the three relevant hypotheses, a more principled data split was used, in combination with cross-validation to decrease the variance. Furthermore, several augmentation methods and deep learning architectures have been implemented and optimized using Bayesian hyperparameter tuning. Beyond the continued use of univariate deep learning models, this thesis introduces the use of multivariate ones. This resulted in test accuracies higher than those reported in previous projects, but with strong indications that the performance is still largely explained by random chance. In conclusion, it seems that the small dataset size is the major reason for the limited success and that future work in this direction should strongly consider incorporating additional data sources for the analysis.

Keywords: Essential Hypertension, Muscle Sympathetic Nerve Activity, Magnetoencephalography, Time Series Classification, Artificial Neural Networks, Long Short-Term Memory, Multilayer Perceptron, Convolutional Neural Network, Transformer, Data Preprocessing, Data Augmentation

Acknowledgements

We would like to express our thanks to Christofer Åkerström at Syntronic Research and Development AB and MedTech West for providing us with this Master's thesis opportunity. We would also like to thank Kevin Andersson from Syntronic and Justin Schneiderman from MedTech West for supervising our work, as well as Professor Paolo Monti at Chalmers University of Technology for being our examiner. Furthermore, we would like to thank Bushra Syeda from Sahlgrenska Academy for giving us important biological data and insight, publishing the initial results that enabled this thesis, and giving us permission to use a figure of hers. We would also like to thank Alma Lund and Felix Ericsson, who worked on a parallel Master's thesis at Syntronic, for interesting discussions. Finally, we want to thank everyone at the Syntronic R&D Gothenburg office for the fierce dart games during our lunch breaks.

David Hall & Elias Sundqvist, Gothenburg, June 2022

Contents

List of Acronyms

ANN Artificial Neural Network. 21, 32, 38, 40, 82, II, IV–VI, IX

BCE Binary Cross-Entropy. IV

CLC Contrasted Linear Combination. 18–20, 48, 49, 56, 57, 64, 65, 73, 82
CNN Convolutional Neural Network. 7–9, 21, 25, 32, 34–36, 58–66, 75–78, 82
Conv1D One-dimensional convolutional layer. 34, 35
CVD Cardiovascular Disease. 1

ECG Electrocardiogram. 28, 29
EEG Electroencephalography. 9, 29, 79
EI Expected improvement. 27
EOG Electrooculogram. 28, 29
ESN Echo State Networks. VIII

FCNN Fully Convolutional Neural Network. 21
FNN Feed Forward Neural Network.

GP Gaussian process. 27
GRU Gated Recurrent Unit. 21

IC Independent Component. 29
ICA Independent Component Analysis. 29
iEEG Intracranial Electroencephalography. 29
ILSTM Incorrect LSTM. 32–34, 36, 40–49, 75–77

LOOCV Leave One Out Cross-Validation. 31, 40, 75
LSTM Long-Short Term Memory. 4, 21–23, 25, 31–33, 40–42

MEG Magnetoencephalography. 1–7, 9–12, 28, 29, 32, 35, 76, 79, 82

Mexh Mexican Hat wavelet. 46, 54, 62, 71

MLP Multilayer Perceptron. 4, 21, 23, 25, 32–34, 36, 50–57, 75–77, VI, VII

Morl Morlet wavelet. 46, 54, 62, 71

MSE Mean Squared Error. 7, IV, VII

MSNA Muscle Sympathetic Nerve Activity. 1, 2, 4–7, 28, 30, 32, 34, 36, 37, 39, 76, 79, 80, 82, IV

PLU Piecewise Linear Unit. III

ReLU Rectified Linear Unit. 32–34, 36, III

RNN Recurrent Neural Network. 21, 22, VIII

RWS Random Window Slicing. 12, 48, 56, 64, 65, 73, 74, 82

S4 Structured State Spaces for Sequence Modeling. 21

SDG Sustainable Development Goal. 80

SGD Stochastic Gradient Descent. V, VI

tSSS Temporal Signal Space Separation. 29

WandB Weights & Biases. 39

Nomenclature

Indices

μ	Index of column data vector in $\mathbf{X}^{(\mu)}$ or $\mathbf{X}$
ν	Index of mini batch matrix
t	Time step index
l	The index of a layer in an ANN.

Matrices

$\mathbf{X}$	The full matrix of column data vectors.
$\mathbf{Y}$	The full matrix of column target vectors.
$\mathbf{X}_{(\nu)}$	Data matrix of mini batch number ν.
$\mathbf{Y}_{(\nu)}$	Target matrix of mini batch number ν.
$\mathbf{W}^{(l)}$	The weight matrix of layer l in an FNN
$\mathbf{\Theta}$	The network parameter matrix of an ANN.

Vectors

$\vec{x}^{(\mu)}$	Column data vector μ
$\mathbf{X}^{(\mu)}$	Column data vector μ in data matrix $\mathbf{X}$
$\mathbf{Y}^{(\mu)}$	Column target vector μ in target matrix $\mathbf{Y}$
$\mathbf{X}^{(\mu)}_{(\nu)}$	Column data vector μ in mini batch data matrix $\mathbf{X}_{(\nu)}$
$\mathbf{Y}^{(\mu)}_{(\nu)}$	Column target vector μ in mini batch target matrix $\mathbf{Y}_{(\nu)}$
$\vec{V}^{(l,\mu)}$	Neuron state vector of layer l in an FNN for data vector μ
$\vec{\theta}^{(l)}$	The threshold vector of layer l in an FNN for data vector μ
$\vec{b}^{(l,\mu)}$	Local field vector of layer l in an FNN for data vector μ

Functions

$\hat{f}_{\boldsymbol{\Theta}}$	The function approximated with an ANN with parameters $\boldsymbol{\Theta}$
$\mathcal{L}$	Loss function
J	Expected loss function
g	Activation function

Variables

x_t	MEG data point at time step t
$\tilde{x}_t$	Approximated MEG data point at time step t
y	Actual MSNA inhibition/non-inhibition label
$\hat{y}$	Predicted MSNA inhibition/non-inhibition label
$l^{(\mu)}$	Actual label of data vector μ
$\delta_{a,b}$	Kronecker delta, $\delta_{a,b} = 1$ if $a = b$. Otherwise, $\delta_{a,b} = 0$

Distributions

$\hat{p}_{\text{data}}$	Data distribution

Constants

p	Total number of data points in the data set
M	Number of classes. In this thesis equal to 2 (MSNA inhibitor and non-inhibitor)
L	Number of layers in an FNN

List of Figures

List of Tables

1

Introduction

This chapter introduces the issue under investigation in this Master's thesis project, namely how to accurately classify muscle sympathetic nerve activity (MSNA) inhibition from magnetoencephalography (MEG) brain activity. The background of the problem at hand, previous work, the aim of this thesis, its delimitations, and its specifications are all covered in the following.

1.1 Background

According to the World Health Organization [1], around a third of all deaths worldwide occur due to cardiovascular disease (CVD).

One of the biggest factors for CVD is essential hypertension (high blood pressure) [2], [3]. Hypertension, from the greek *hyper*, meaning "beyond", and *tension*, which means stretching or straining, refers to the excessive strain on your blood vessels from abnormally high blood pressure [4]. Essential/primary hypertension is a kind of hypertension that originates from genetic or environmental factors, as opposed to secondary hypertension which has identifiable causes such as diseased organs [5].

One likely cause for essential hypertension is a lifetime of arousal-induced elevated blood pressure responses. The nerve activity responsible for regulating blood pressure is the muscle sympathetic nerve activity (MSNA).

1.1.1 Muscle Sympathetic Nerve Activity (MSNA)

The sympathetic nervous system is responsible for your body's fight-or-flight response [6]. It can control the behavior of various organs in your body in response to scary or stressful situations. One functionality of the sympathetic nervous system is the ability to constrict or dilate vasculature [3]. This is also known as vasoconstriction/vasodilation [7]. Constricted vasculature forces blood to flow through a smaller volume, which increases blood pressure.

The sympathetic nervous system regulates vasculature size through electrical activity. This activity is called muscle sympathetic nerve activity (MSNA) [7]. There is always a baseline level of MSNA activity [8]. Going higher than this baseline induces constriction, while going lower induces dilation [3].

For parts of the population, stressful situations induce an increased MSNA response.

If they are exposed to stress often, this can lead to long-term negative health implications. The rest of the population exhibit an inhibition of MSNA activity when stressed, which dilates the vasculature. These individuals are not exposed to the same risk [3]. The knowledge of which of these categories an individual belongs to can enable medical professionals to prescribe appropriate preventative measures to their patients.

The current standard approach for determining whether an individual is an MSNA inhibitor is through an invasive method. MSNA can be measured with microneurography, for example through the electrical activity in the peroneal nerve in the leg (see Figure 1.1). By recording this activity and providing some stressful stimulus, such as an electrical shock in the finger, it is possible to determine whether the MSNA activity is inhibited during stress. The standard procedure is to label those with a MSNA activity reduction by 30% or more after the stimulus as "inhibitors" and the rest as "non-inhibitors" [3]. This procedure works, but is slow and invasive. As with most invasive procedures, it is rarely done unless absolutely necessary. It is therefore worth investigating other approaches for getting this data.

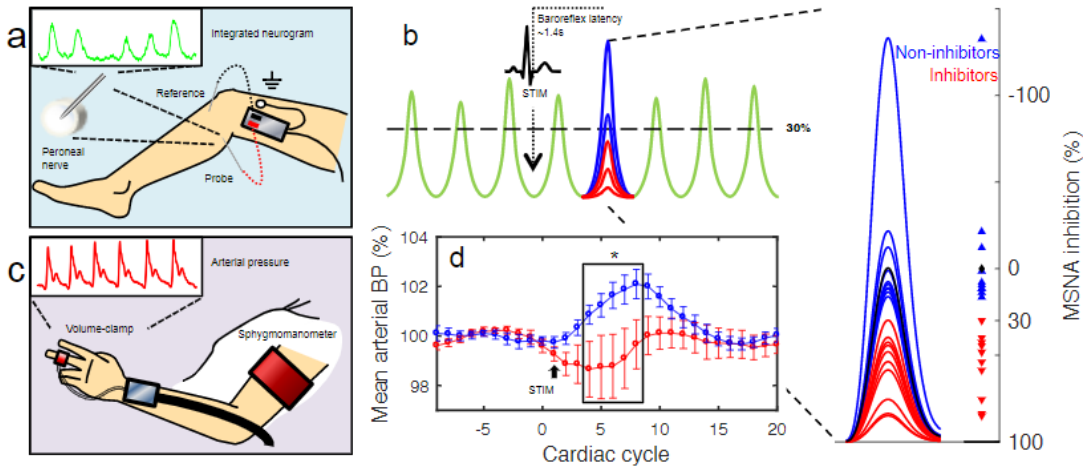
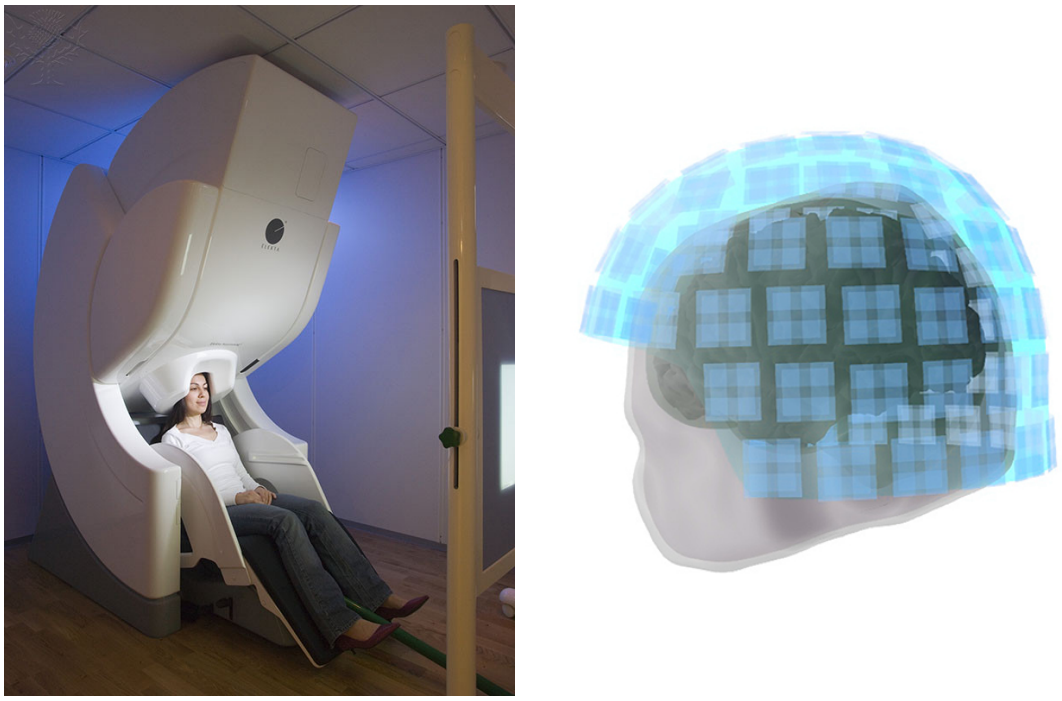


Figure 1.1: Current clinical process for measuring MSNA inhibition. Source: [3] (with permission).

Classifying a patient as an inhibitor or non-inhibitor of MSNA can be done by analyzing brain activity data obtained with magnetoencephalography.

1.1.2 Magnetoencephalography (MEG)

Magnetoencephalography (MEG) is a method for measuring brain activity based on fluctuations in the magnetic field [9]. This is achieved by placing very sensitive magnetic sensors close to the skull, as illustrated in Figure 1.2b. To avoid the interference of background noise, subjects must also be inside a magnetically shielded room when the measurements are performed. Figure 1.2a shows an example of an MEG scanner in such a room. An example of the MEG data recorded from one sensor during the process illustrated in Figure 1.1 can be seen in Figure 1.3.



(a) MEG scanner at the University of Oxford. Source: [10].

(b) Placement of MEG sensors. Source: Adapted from [11].

Figure 1.2: Example of a MEG scanner and the placement of sensors.

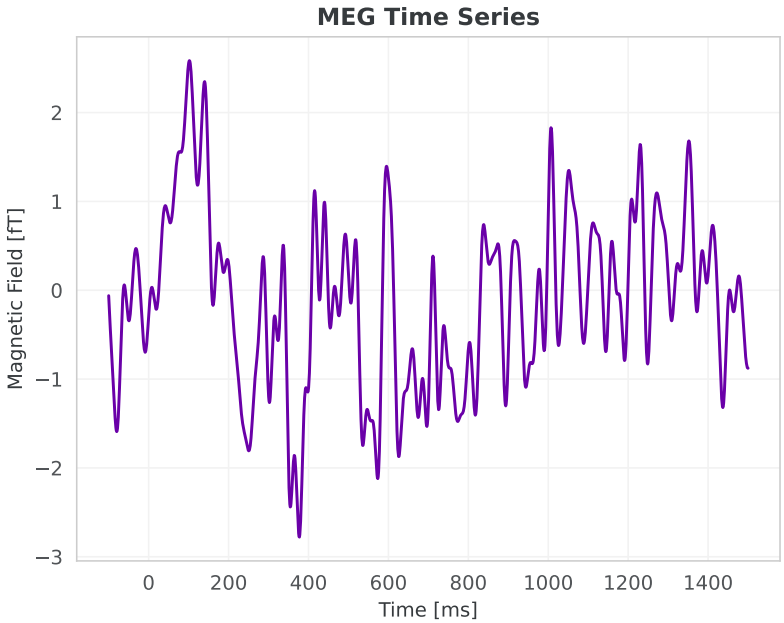


Figure 1.3: Example time series from one MEG sensor during the process carried out in Figure 1.1. The x-axis corresponds to milliseconds relative to the electric stimulus.

In 2018, it was found that there is a correlation between MSNA and MEG activity. In particular, a correlation was found in the Rolandic area of the brain (surrounding the central sulcus [12], see Figure 1.4) around the beta frequency band (13–25 Hz) [3].

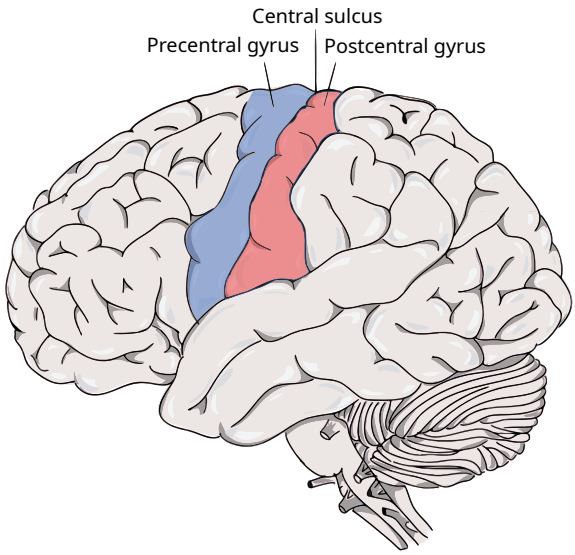


Figure 1.4: Lateral View Of The Brain. Source: Adapted from [13].

1.2 Previous Work

This section describes the previous work on the subject at hand that this thesis builds upon.

1.2.1 Previous Master's Theses

In collaboration with Syntronic and MedTech West, two previous Master's thesis projects have already investigated the use of neural networks for predicting hypertension development from MEG data to an extent. None of the made attempts resulted in classification accuracies of MSNA inhibition sufficiently high enough for clinical use (for example a threshold of 90% on the test set, as suggested by Park, et al.) [14]–[16].

In the first thesis from 2020 [15], the authors investigated whether long short-term memory networks (LSTM), and multilayer perceptron networks (MLP) could classify MSNA inhibition with high accuracy. They also examined whether an autoencoder could be used to increase the accuracy by acting as a high-frequency filter. Neither method resulted in high classification accuracies on the used test set, the highest being 64.5% [15], which means that their models did not generalize well. This is however not surprising given the small data set used. It is also worth noting that cross-validation was not used, which means that the previous results should be taken with a grain of salt.

The second thesis project from 2021 [16] attempted to increase the accuracy of the

models from the previous thesis by filtering out MEG frequencies not in the beta-frequency band since it is correlated to MSNA (See Section 1.1.2). The hypothesis was that individuality could be discarded from the data this way and therefore lead to greater generalization. They however found that this led to a decrease in classification accuracy and concluded that it is not only the beta-band that carries important information regarding MSNA inhibition.

1.2.2 Traditional Statistical Approaches

Besides the two aforementioned Master's theses, there have also been attempts to analyze the MEG data using traditional statistical methods [17]. These approaches have been relatively successful in finding regions where activity in the beta band correlates strongly with MSNA activity.

Two downsides of traditional statistical approaches are that, 1) Currently, according to the research of this thesis, no attempt has been made to make a classifier based on this signal. Despite a high correlation, it is possible that there is a large overlap between the classes, and it may therefore be difficult to make a good decision boundary. 2) The approach requires an averaging over many MEG measurements, requiring around 1 hour per patient to get a result [3].

1.3 Aim

As discussed in [16], a potential reason for the poor MSNA inhibition classification accuracies in the previous attempts is that the dataset only contains data from 19 individuals. It was and still is therefore very difficult to train a model that generalizes well across the larger population. The aim of this project is to explore different ways of preprocessing and augmenting the data, as well as trying new deep learning architectures to hopefully increase generalization performance.

1.4 Delimitations

One limitation of this Master's thesis project is that no more data will be collected. The only data that will be used when training models is what has been given: a dataset with a mere 19 subjects, but with a lot of MEG time series data per individual. In total, the dataset is 31 GB. Having just 19 subjects will, as stated before, make the task of classifying whether an individual is an MSNA inhibitor or not difficult. This is however what is available and there is currently no option available for obtaining more data at this time.

Another limitation is that the analysis will be restricted to the time domain, similar to previous projects, as another Master's thesis project at Syntronic running in parallel will investigate classification in the time-frequency domain.

A third limitation is that architectures developed during this project need to be sufficiently resource-efficient to be able to run on the computers accessible at the office of Syntronic in Gothenburg. For example, neural networks cannot have too

many parameters. The computer that will be used for training has approximately 12 GB of VRAM and about 30 GB of RAM, which limits the number of parameters, especially since another thesis project will use it simultaneously.

A fourth limitation is that the prediction models developed here will not be designed to be user-friendly so that they can be used in clinical settings. The focus of this project is solely to find ways of accurately classifying MSNA inhibitors and non-inhibitors based on time series MEG data.

A final limitation is that the focus of this project will be restricted to analyzing sensor-level data.

2

Theory

This chapter introduces the relevant technical theory underlying the work done in this Master's thesis project.

2.1 Preprocessing of Time Series Data

Preprocessing involves performing different computational operations on a dataset before feeding it to a machine learning model. The purpose of doing this is typically to get the data into a form that the model can work with or to simplify the task. The most common operations are scaling and resampling (down/up-sampling). These operations are provided by many libraries such as *tslearn* [18]. Other common operations are feature extraction (provided by libraries such as TSFEL [19]), and frequency filtering [20, Section 7.2].

The rest of this section describes two preprocessing methods used in this thesis: Noise-removal with autoencoders and feature extraction with wavelets. The former could be of value because of potential external disturbances in the magnetic field measured with MEG. The latter could be of value because of certain MEG signal frequencies having stronger correlations with MSNA, as mentioned in Section 1.1.2.

2.1.1 Removing Noise With Autoencoders

Autoencoders are generative artificial neural networks [21] (see Appendix A.5) and can be used for preprocessing as tools for filtering out noise in data. The autoencoder first encodes the data into a smaller format which only consists of its most crucial characteristics (see the red, middle layers in Figure 2.1 and Figure 2.2). It then decodes the encoded data to the same dimension as the input.

By minimizing for example the mean squared error (MSE, a loss function, see Appendix A.3) between the network output and the input, the autoencoder learns to replicate the input quite well but not perfect. Noise is thus discarded [22].

An autoencoder with linear layers for univariate data (see Figure 2.1) was used to remove noise in the 2020 thesis [15]. However, autoencoders can also be implemented as convolutional neural networks (CNNs) to be used on multivariate data by moving a kernel over its channels (see Figure 2.2). If the kernel size is 1, the CNN autoencoder is equivalent to applying an autoencoder with linear layers on each channel.

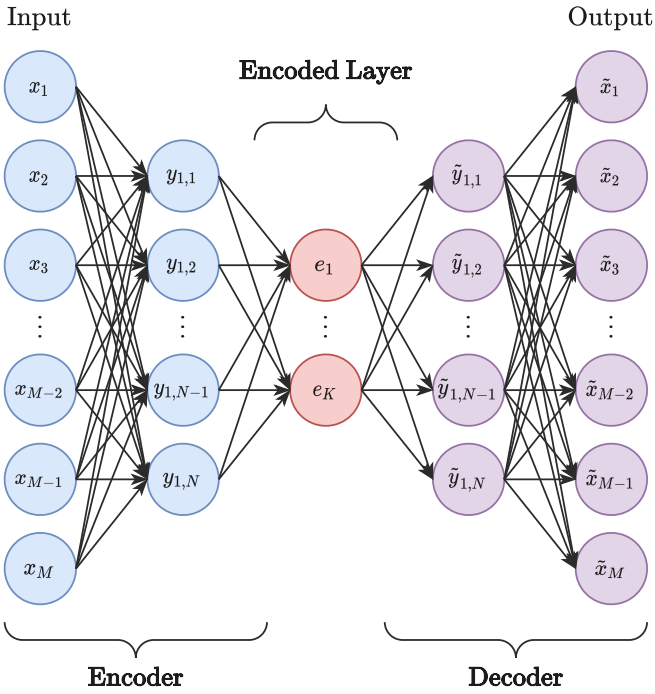


Figure 2.1: A schematic of an autoencoder of depth 2 with univariate input. M denotes the length of the input/output, N denotes the number of neurons in the hidden layers, and K denotes the number of neurons in the encoded layer. Note that: $K < N < M$.

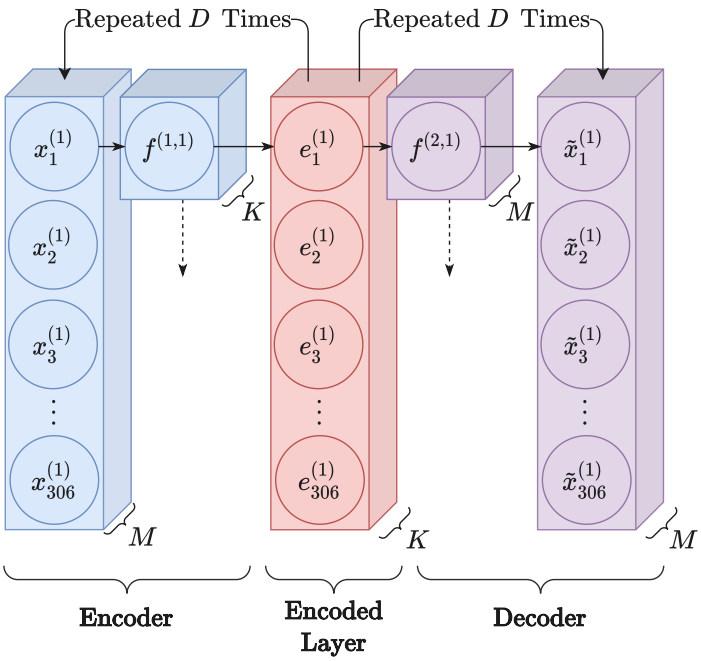


Figure 2.2: A schematic of a CNN autoencoder of depth D with multivariate input (306 channels) and kernel size 1. M denotes the length of the input/output, K denotes the length of the encoded data. Note that: $K < M$.

The size of the encoded layer and the kernel size are not the only hyperparameters that autoencoders can have. Their depths can also be varied. Figure 2.1 shows an autoencoder with linear layers of depth 2, Figure 2.2 shows a CNN autoencoder with an arbitrary depth $D \in \mathbb{Z}^+$.

A plot showing the effect of an autoencoder on a time series from the provided data set for this thesis can be seen in Figure 2.3 below.

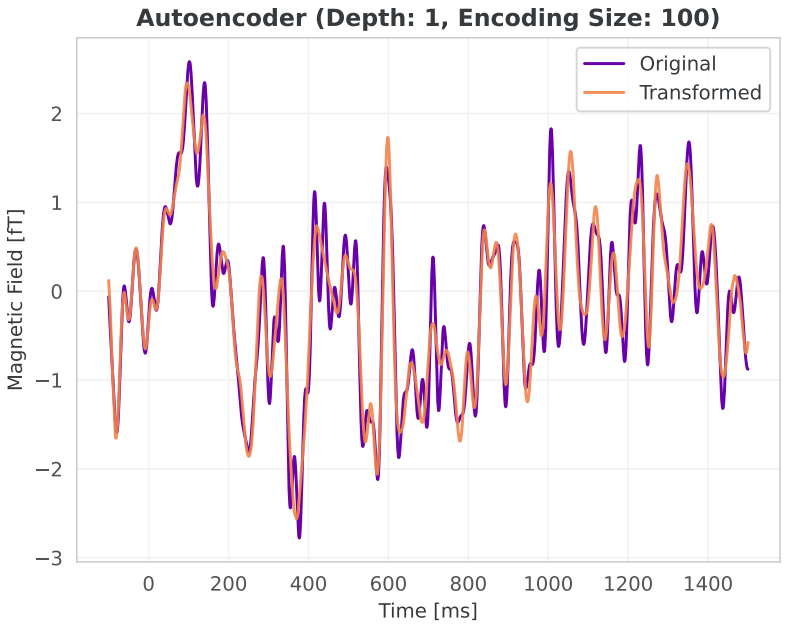


Figure 2.3: A time series and its counterpart after being filtered with an autoencoder of depth 1 and encoding dimension 100.

2.1.2 Feature Extraction With Wavelet Transforms

Wavelets can be used for feature extraction in time series. It has for example been used for this purpose on EEG data [23], which is similar to the MEG data used in this thesis. A wavelet is a wave that only fluctuates for a short amount of time, compared to a big wave such as the sine function, which oscillates infinitely over time [24]. When used for feature extraction, the wavelet in question is convolved over the signal in the time dimension, resulting in a new signal (sequence of wavelet coefficients) [25].

A wavelet $\psi_{ab}(t)$ has two parameters: dilation (a, also called scale [26]) and translation (b). The dilation a determines which frequency will be extracted from the convolution and the translation only results in the wavelet coefficients being shifted one way or another. The so-called mother function of the wavelet is simply the wavelet without the dilation and translation constraints, meaning: $\psi(t)$. Two properties of $\psi(t)$ are that its integral is equal to 0 and that the integral of its square is equal to 1. In the formulas down below, the function f is the target of the wavelet transform and $W_f(a, b)$ is the result of the convolution or, in other words, the wavelet

coefficients [25]. In the context of this thesis, f represents a time series in the MEG data.

$$\psi_{ab}(t) = \frac{1}{\sqrt{a}}\psi\Big(\frac{t-b}{a}\Big), \quad W_f(a,b) = \int_{-\infty}^{\infty} \psi_{ab}(t) f(t) dt \tag{2.1}$$

One example of a wavelet family is complex Morlet wavelets. They are defined as in Equation 2.2. The first factor in Equation 2.2 is a complex sine wave and h denotes its frequency. The second factor is a Gaussian window with a width of σ and n number of cycles [27]. This window essentially turns the otherwise endless sine wave into a short wave, or in other words a wavelet.

$$\psi(t) = e^{2i\pi ht} e^{\frac{-t^2}{2\sigma^2}}, \qquad \sigma = \frac{n}{2\pi h} \tag{2.2}$$

The `pywt` [26] library defines the complex Morlet wavelets slightly differently (see Equation 2.3) with two other parameters B (bandwidth) and C (center frequency, same as h).

$$\psi(t) = \frac{1}{\sqrt{\pi B}} e^{\frac{-t^2}{B}} e^{2i\pi Ct} \tag{2.3}$$

There are also real-valued Morlet wavelets. The one defined in the `pywt` library has a number of cycles $n = 5$ and the Gaussian window width $\sigma = 1$ (see Equation 2.4) [26]. The plot of the wavelet can be seen in Figure 2.4.

$$\psi(t) = e^{\frac{-t^2}{2}} \cos(5t) \tag{2.4}$$

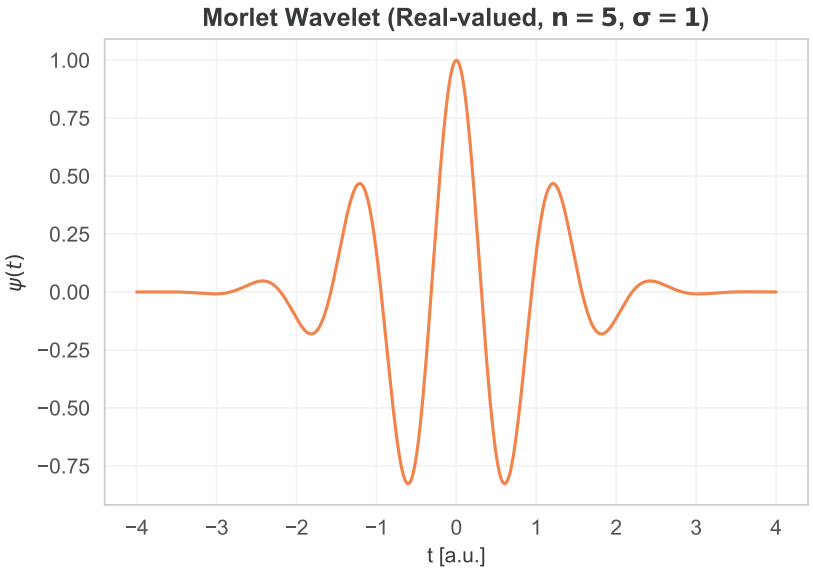


Figure 2.4: A real-valued Morlet wavelet with $n = 5$ and $\sigma = 1$ (see Equation 2.4).

The plot below shows an example of how a time series of MEG data looks after having been transformed with the real-valued Morlet wavelet from Equation 2.4 at 15 Hz.

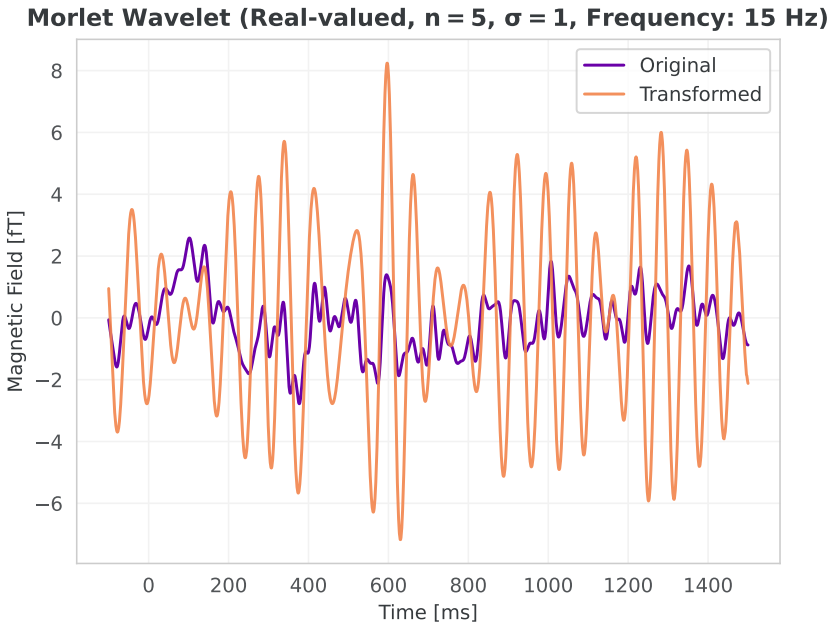


Figure 2.5: A time series and its counterpart at 15 Hz which was extracted with the Morlet wavelet transform.

There is an infinite amount of wavelet types and variations of them. Two examples beyond Morlet are the Haar and Mexican Hat wavelets [24]. Each wavelet is better suited for different problems. It is thus necessary to choose wavelets carefully [28], [29]. The described Morlet wavelet is for example often used on neuroelectrical signals [27], which are similar to the MEG data at hand in this thesis (see Section 1.1.2 and Section 3.1).

2.2 Augmentation of Time Series Data

Data augmentation involves generating synthetic data in order to increase the size of data sets [30]. The available data set is, as mentioned before, quite small. Data augmentation could therefore be beneficial as it would expand the data set. This could enable the deep learning models to generalize better. This section covers some existing augmentation techniques related to time series data that could be used when augmenting the data at hand.

As the dataset expands with variants of the original data, neural networks will learn not to pay attention to the properties that the augmentation techniques give the data. For example, if the data is permuted randomly as explained in Section 2.2.2, they will learn that the order of things does not matter. It is not always clear which properties can be ignored, which is why many data augmentation techniques have been considered and tested in this thesis (see Chapter 4 and Section 5.1.4 for more on the subject).

It is important to note that augmenting data can result in labels changing, which would be detrimental to classification accuracies. Mislabeling data results in so-called "class noise" [31]. In the context of MEG time series, domain knowledge is needed to determine which augmentation techniques presented in this section could result in such noise.

2.2.1 Random Window Slicing (RWS)

A common way of augmenting time series data is to slice it into segments. Each slice keeps the label of the time series it was extracted from, and a network can then be trained on them rather than on the whole time series. This augmentation method is called *window slicing* [32], [33]. In the deterministic case, this is more of a preprocessing technique but the window location can be decided randomly as well. See Figure 2.6 for an example of what this augmentation technique does.

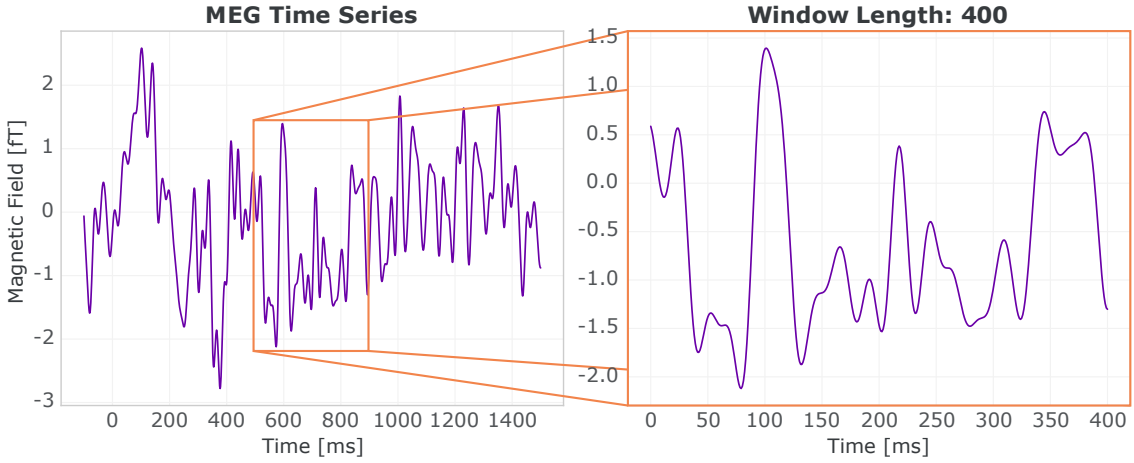


Figure 2.6: A time series and a stochastically extracted slice of length 400.

2.2.2 Permutation

The window slicing data augmentation technique mentioned in the previous subsection leads directly to another method called *permutation*. It involves slicing the time series into pieces and randomly putting them together again, thus forming a new time series [34], [35]. See Figure 2.7 for an example of what this augmentation technique does.

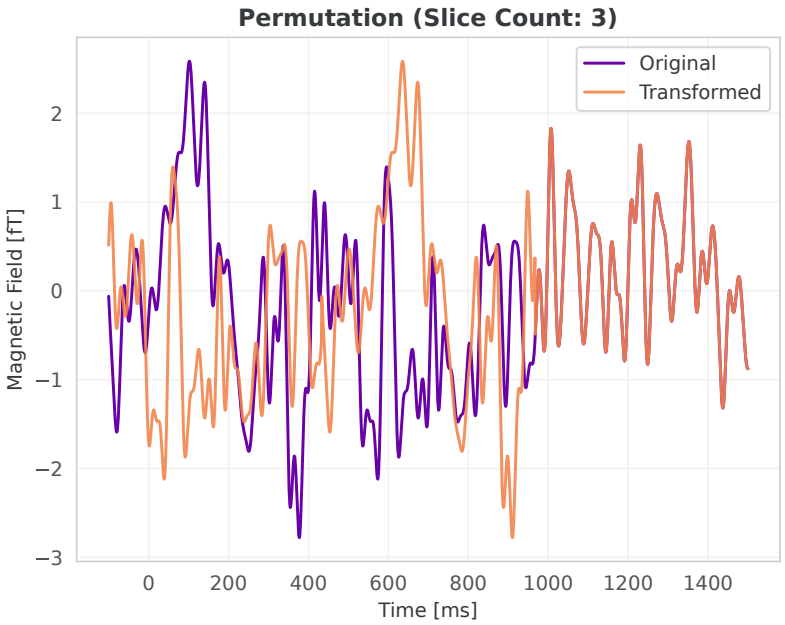


Figure 2.7: A time series and a permutation of it.

2.2.3 Time Warping and Window Warping

Another method is called *time-warping* [34] or simply *warping* and involves speeding up or slowing down segments in the time series. This however results in new time series that are of different lengths. To combat this, this augmentation method can be combined with window slicing in order to make all data the same length. This combination is called *window warping* [32].

Instead of having to slice away data after performing a time-warp as in window warping, a similar method was developed for this thesis. From here on out, this is what is meant by "time warping". First, a sequence of two-dimensional points with coordinates x and y are randomly generated. Both coordinates are within the integer range of the time series in question, meaning $[-100, 1500]$ in the context of this thesis (see Section 3.1.1). The first coordinate, x, defines the start/end of a section in the original time series that will be warped. The second coordinate, y, defines the corresponding start/end of a section in time in the warped time series. Thus, if this time warping procedure is denoted as a function f, this means that $f(x) = y$. The second step clamps the end points of the generated sequence to

the minimum and maximum of the time range. This forces the augmented time series to have the same dimension as the original. After that, a spline is created by interpolating between the generated time warp points (Figure 2.8 shows the spline as a red line). The original time series is then finally warped by mapping the time values of its data points to the y values of the interpolated spline.

The effect of the time-warp on the data can be tweaked with a hyperparameter ω, which will be called *warp factor* ($\omega \in [0, 1]$). The final augmented data $\tilde{\mathbf{X}}$ is computed as in Equation 2.5, where $\mathbf{W}$ is the time-warped data and $\mathbf{X}$ is the original data.

$$\tilde{\mathbf{X}} = \omega\mathbf{W} + (1 - \omega)\mathbf{X} \tag{2.5}$$

Figure 2.9 shows an actual time series from the provided data set and its time warped counterpart.

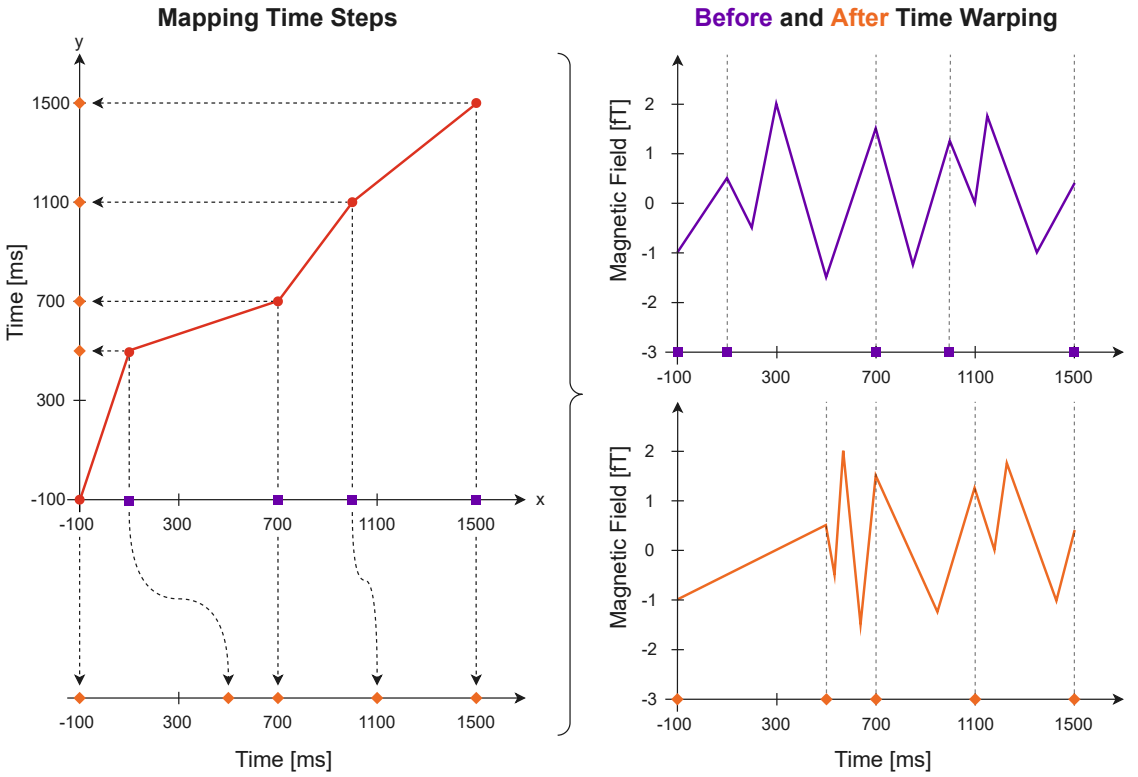


Figure 2.8: Red dots denote the time warp points. Purple squares denote the original time steps corresponding to the time warp points. Orange diamonds represent the warped time steps. Dashed arrows denote the time step mappings, resulting in the orange time series when applied on the purple one.

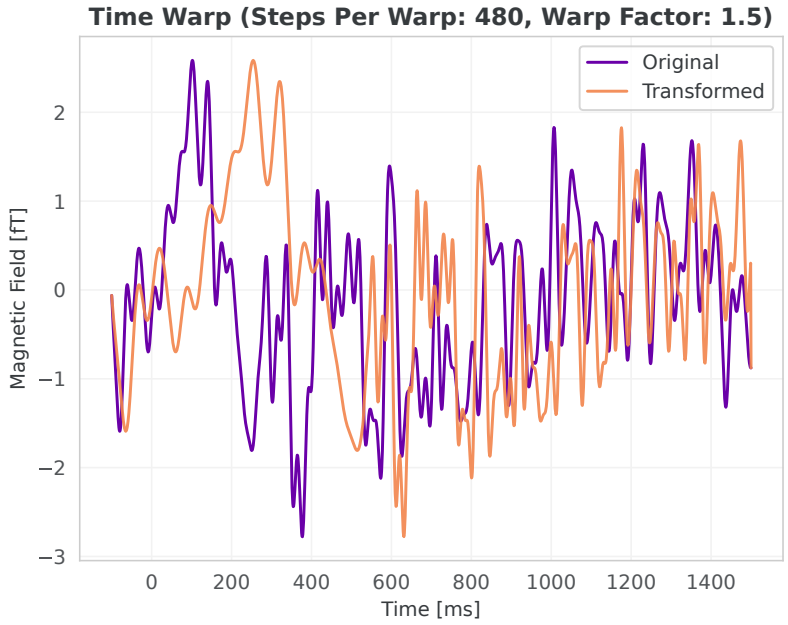


Figure 2.9: A time series and its time-warped counterpart.

2.2.4 Jittering

Time series data can also be augmented by adding noise to it [36]. This method is called *jittering* and can for example be done by adding noise following a Gaussian distribution to each time step of the data [35]. See Figure 2.10 for an example of what this augmentation technique does.

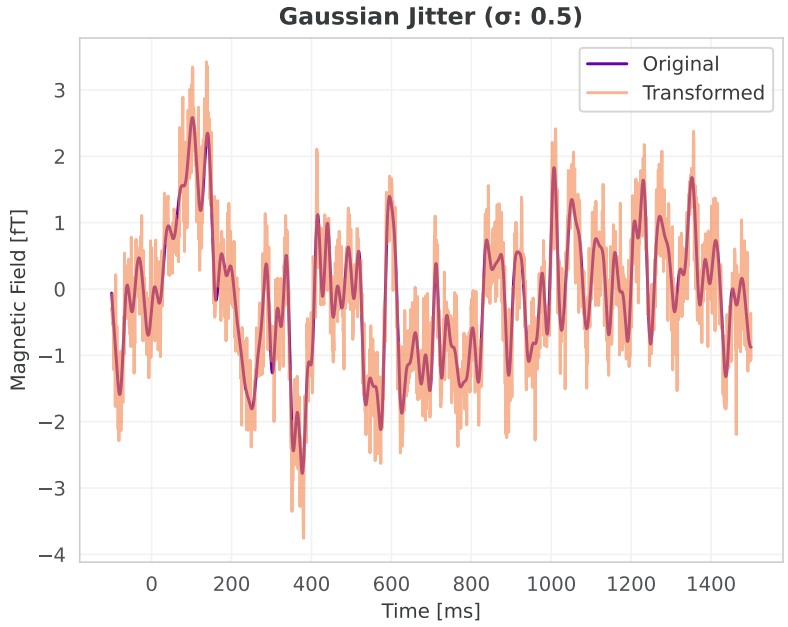


Figure 2.10: A time series and its counterpart jittered with Gaussian noise.

2.2.5 Stochastic Scaling

Scaling is another data augmentation method and involves scaling the value the same at every time step in the time series. The scaling factor can for example be sampled from a Gaussian distribution. This does not affect length of the time series [35]. See Figure 2.11 for an example of what this augmentation technique does.

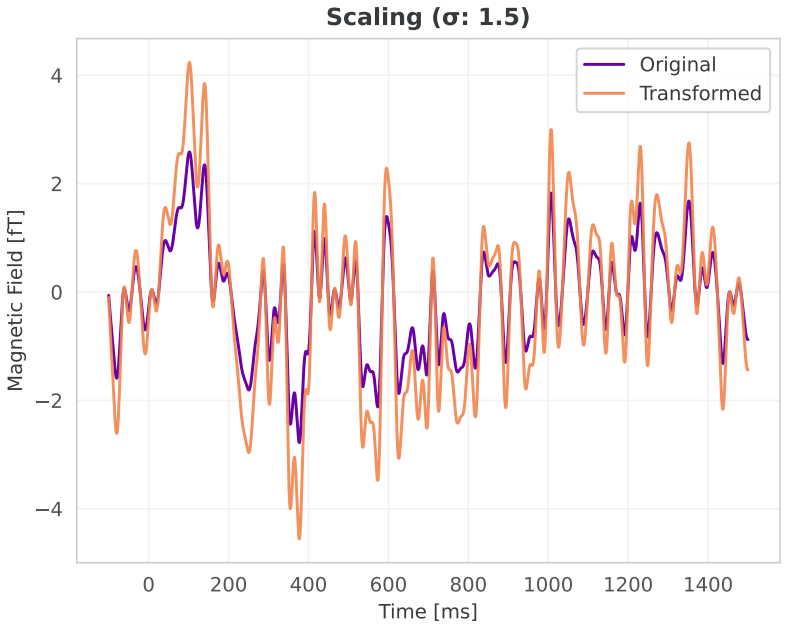


Figure 2.11: A time series and its stochastically scaled counterpart.

2.2.6 Magnitude Warping

An augmentation technique similar to scaling is *magnitude warping* which also involves scaling the magnitude. Here, it is however done according to a cubic spline which is a smooth curve.

First, the number of spline knots is chosen. In the paper by Iwana and Uchida four knots are suggested [37]. The knots are then randomly generated. This can be done in any way, but one example is to randomize their y-values according to a Gaussian distribution ($\mathcal{N}(1, 0.2)$ is suggested in the mentioned paper [37]) and evenly distributing them on the x-axis within the time range of the time series in question.

A cubic spline is then created through interpolation between these knots [37]. It being cubic means that each section between two knots can be described as a cubic function. In order for the spline to be smooth at the knots as well, the derivative of each such function needs to be the same as the neighboring function at the surrounding knots.

An example of a cubic spline generated in this manner is shown in Figure 2.12. This plot is just a sketch but shows how a spline could look. Note that the end points of the spline are not counted as knots. This is because the knots do not have to be evenly distributed on the time axis as in the figure. The end points always have to be fixed to the start and end of the time range in order to make the spline usable in the next step of the procedure. Note that the end points are randomized in the scaling value dimension, just like the knots. The knots are however randomized in the time dimension as well.

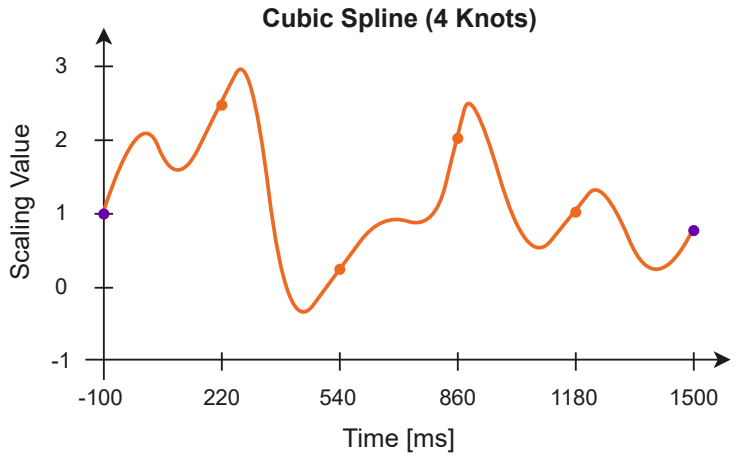


Figure 2.12: An extreme example of a cubic spline. Purple dots are start and end points. Orange dots are knots and their scaling values are distributed to mimic $\mathcal{N}(1, 1.5)$.

Once the smooth spline has been created, it is multiplied with the time series in question to yield an augmented time series [37]. Had the scaling values been generated totally at random at each time step, the result would have essentially been jitter (see Section 2.2.4). Magnitude warping allows for non-uniform scaling in different parts of the time series, without resulting in jitter. See Figure 2.13 for an example of what this augmentation technique does.

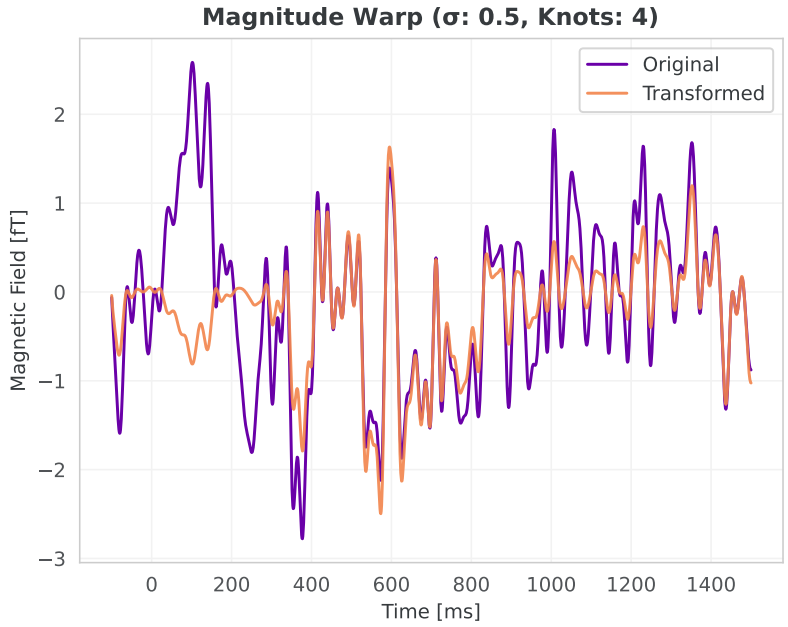


Figure 2.13: A time series and its magnitude warped counterpart.

2.2.7 Contrasted Linear Combination (CLC)

Another novel data augmentation technique (as far as the research for this thesis has shown) will be called "Contrasted Linear Combination" ("CLC" for short). The idea is to augment each time series in a batch based on a randomly weighted linear combination of all time series of the same label in the batch.

The randomly generated weights can be tweaked so that the greater of the original ones stand out more than the smaller ones. This is done by taking the power of the original weight matrix and using it instead. The exponent of this power, c, is here called "contrast", hence the name of this augmentation technique.

Before multiplying the computed weight matrix with the original time series batch $\mathbf{B}$ to yield an augmented version, it is first normalized row-wise so that each row becomes a probability vector. This normalized weight matrix will be denoted with $\mathbf{W}$. The augmented time series batch is thus: $\mathbf{B}_{\text{new}} = \mathbf{WB}$ (see Figure 2.14 for a visualization).

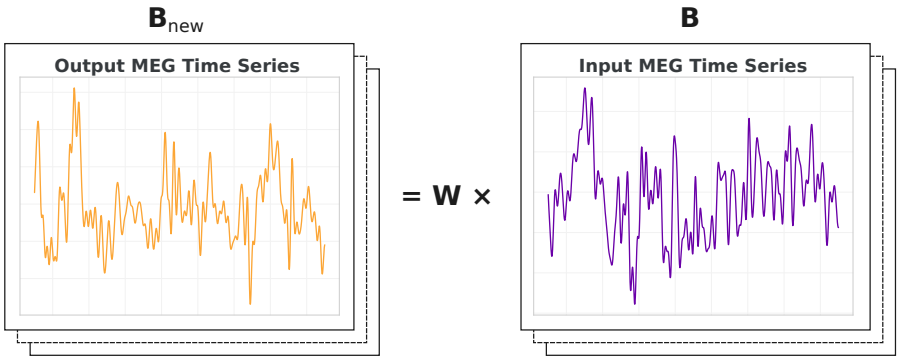


Figure 2.14: Batch $\mathbf{B}$ and its CLC augmented counterpart $\mathbf{B}_{\text{new}}$

The rest of this section derives the mathematics in more detail. First, here is a list of definitions:

$$
\begin{aligned}
c &\coloneqq \text{Weight contrast, } c \in \mathbb{R} \\
T &\coloneqq \text{Number of time steps in the time series} \\
\mathbf{B} &\coloneqq \text{Original batch of time series, } \mathbf{B} \in \mathbb{R}^{N_\mathbf{B} \times T} \\
N_\mathbf{B} &\coloneqq \text{Number of time series in the batch } \mathbf{B} \\
\vec{l} &\coloneqq \text{Vector of labels } l_i, \text{ one per time series in } \mathbf{B} \ (l_i \in \{0,1\},\ \forall i \in [1, N_\mathbf{B}]) \\
d(\mathbf{X}) &\coloneqq \begin{bmatrix} \|\mathbf{X}^{(1)}\| & 0 & \cdots & 0 \\ 0 & \|\mathbf{X}^{(2)}\| & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \|\mathbf{X}^{(N_\mathbf{X})}\| \end{bmatrix}, \begin{cases} \mathbf{X} \in \mathbb{R}^{N_\mathbf{X} \times S} \\ N_\mathbf{X},\ S \in \mathbb{Z}^+ \end{cases}
\end{aligned}
$$

The first step of CLC is the random generation of the weight matrix $\mathbf{A} \in \mathbb{R}^{N_\mathbf{B} \times N_\mathbf{B}}$ so that each of its elements $a_{i,j} \sim \text{Uniform}([0,\ 1)),\ \forall i, j \in [1,\ N_\mathbf{B}]$. This matrix will be used to produce the final weight matrix $\mathbf{W}$ in later steps.

The second step is to create a label matrix $\mathbf{L}$ which elements are either 0 or 1. An element is always 0 if the label of the time series corresponding to the row is not equal to the label of the time series corresponding to the column. Moreover, an element is always 1 if the labels match. The properties of this matrix force each linear combination of time series in the last step of the procedure to only consist of time series of the same label. It would not make sense to mix labels in the linear combinations since the label of the augmented data would then be ambiguous. The label matrix is computed as follows:

$$\mathbf{L} = \vec{l}\,\vec{l}^{\intercal} + (\vec{1} - \vec{l})(\vec{1} - \vec{l})^{\intercal} \tag{2.6}$$

In the third step, the randomly generated weights $\mathbf{A}$ are given the same label properties as $\mathbf{L}$ through matrix multiplication. Let the product be called $\mathbf{U}$:

$$\mathbf{U} = \mathbf{L}\mathbf{A} \tag{2.7}$$

The fourth step introduces the use of the contrast scalar c when taking the power of $\mathbf{U}$, resulting in the contrasted weight matrix $\mathbf{U}^c$. If $c = 1$, it will of course not have an effect on the weight matrix.

The penultimate step of the procedure is to normalize the weight matrix $\mathbf{U}^c$ to obtain the final weight matrix $\mathbf{W}$. This is done by multiplying it with the inverse of the diagonal matrix $d(\mathbf{U}^c)$ (see the general definition of the function d in the list above). This is the case since the diagonal of $d(\mathbf{U}^c)$ are the norms of the rows in $\mathbf{U}^c$:

$$\mathbf{W} = d(\mathbf{U}^c)^{-1}\,\mathbf{U}^c \tag{2.8}$$

This normalization of course results in each row in $\mathbf{W}$ being a probability vector. A row thus defines how much of each time series in $\mathbf{B}$ of the same label to include in the linear combination. $\mathbf{W}$ consists of elements $w_{i,j},\ \forall i, j \in [1,\ N_{\mathbf{B}}]$:

$$\sum_{j=1}^{N_{\mathbf{B}}} w_{i,j} = 1,\ \forall i \in [1,\ N_{\mathbf{B}}] \tag{2.9}$$

The final step of CLC is the augmentation of the time series batch $\mathbf{B}$ itself, yielding the new time series batch $\mathbf{B}_{\text{new}}$ which after computation is ready for use. The computation is done as explained earlier by multiplying $\mathbf{W}$ with $\mathbf{B}$:

$$\mathbf{B}_{\text{new}} = \mathbf{W}\mathbf{B} = d(\mathbf{U}^c)^{-1}\,\mathbf{U}^c\mathbf{B} = d((\mathbf{L}\mathbf{A})^c)^{-1}\,(\mathbf{L}\mathbf{A})^c\mathbf{B} \tag{2.10}$$

2.3 Artificial Neural Networks For Time Series Classification

Artificial Neural Networks (ANNs) are function approximators [22]. The unique feature of ANNs is their ability to learn compositional features of the data. They are typically (but not necessarily) characterized by being differentiable and operating on high-dimensional data.

There are certain types of ANNs that might be better suited for time series classification than others. These are for example some discriminative models (see Appendix A.5) that could be relevant: Multi-Layer Perceptrons (MLP), One-dimensional Convolutional Networks (CNN), Fully CNN (FCNN), Encoder, Multi-Scale CNN, Time Le-Net, Multichannel Deep CNN, Residual Networks, Long Short-Term Memory (LSTM), Gated Recurrent Neural Network (RNN), Gated Recurrent Neural Networks (GRU), S4, and Transformer [21], [22], [38]–[40].

The remainder of this section describes architectures that actually have been used in this project in more detail in order to clarify how the results in Chapter 4 were obtained. For theory on ANNs in general, see Appendix A.

2.3.1 Multilayer Perceptron (MLP)

The canonical ANN architecture is a Multilayer Perceptron (MLP for short, see Figure 2.15). It has an input layer, an arbitrary number of hidden layers, and an output layer. Note that all layers are linear and that non-linearity can be added with non-linear activation functions (see Appendix A.2). There can be an arbitrary amount of so-called neurons (see Appendix A.1) in each hidden layer and output layer [22]. An MLP can be defined mathematically as follows:

$$\vec{h}_0 \leftarrow \vec{x} \tag{2.11}$$

$$\vec{h}_n \leftarrow g(\mathbf{W}_n \vec{h}_{n-1} + \vec{\theta_n}) \tag{2.12}$$

$$\vec{y} \leftarrow \vec{h_N} \tag{2.13}$$

$\vec{x} \coloneqq$ The input vector that is fed to the network.
$N \coloneqq$ The number of layers in the network ($N \in \mathbb{Z}^+$)
$n \coloneqq n \in [0, N]$)
$\vec{h}_n \coloneqq$ The vector of neuron state values in network layer n
$\mathbf{W}_n \coloneqq$ The weight matrix of network layer n
$\vec{\theta_n} \coloneqq$ The threshold vector of the network layer n
$\vec{y} \coloneqq$ The output vector of the network
$g \coloneqq$ Activation function (see Appendix A.2)

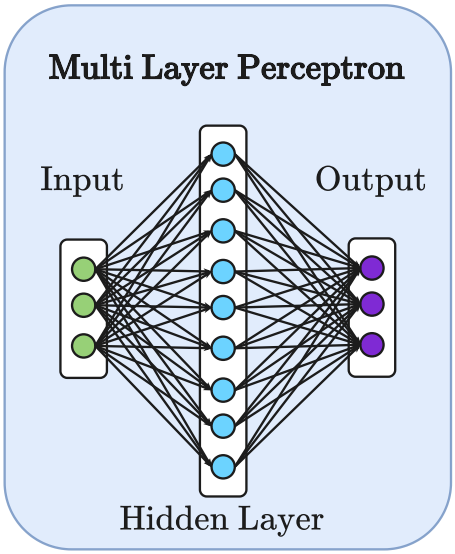


Figure 2.15: Illustration of a multilayer perceptron with one hidden layer.

2.3.2 Long-Short Term Memory (LSTM)

Long-Short Term Memory (LSTM) [41], are recurrent neural networks (RNN), augmented with the ability to selectively store and forget information. The basic LSTM architecture is illustrated in Figure 2.16.

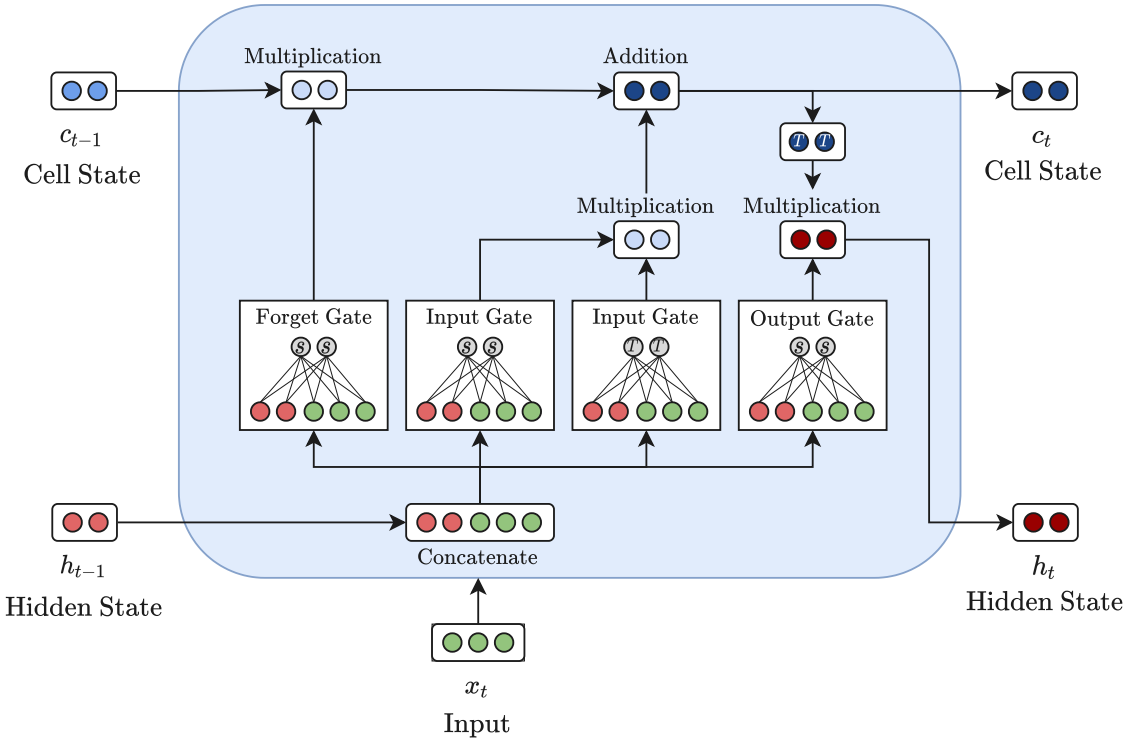


Figure 2.16: Illustration of a LSTM cell.

The basic formula for the forward pass of an LSTM cell is.

$$\vec{h}_0 \leftarrow \vec{0} \tag{2.14}$$
$$\vec{c}_0 \leftarrow \vec{0} \tag{2.15}$$
$$\vec{f}_t \leftarrow \sigma_s(\mathbf{W}_f \vec{x}_t + \mathbf{U}_f \vec{h}_{t-1} + \vec{b}_f) \tag{2.16}$$
$$\vec{i}_t \leftarrow \sigma_s(\mathbf{W}_i \vec{x}_t + \mathbf{U}_i \vec{h}_{t-1} + \vec{b}_i) \tag{2.17}$$
$$\vec{o}_t \leftarrow \sigma_s(\mathbf{W}_o \vec{x}_t + \mathbf{U}_o \vec{h}_{t-1} + \vec{b}_o) \tag{2.18}$$
$$\vec{\tilde{c}}_t \leftarrow \sigma_t(\mathbf{W}_c \vec{x}_t + \mathbf{U}_c \vec{h}_{t-1} + \vec{b}_c) \tag{2.19}$$
$$\vec{c}_t \leftarrow \vec{f}_t \odot \vec{c}_{t-1} + \vec{i}_t \odot \vec{\tilde{c}}_t \tag{2.20}$$
$$\vec{h}_t \leftarrow \vec{o}_t \odot \sigma_h(\vec{c}_t) \tag{2.21}$$

$\vec{x}_t \coloneqq$ The input vector
$\vec{f}_t \coloneqq$ Forget gate activity $\in (0,1)^h$
$\vec{i}_t \coloneqq$ Input gate activity $\in (0,1)^h$
$\vec{o}_t \coloneqq$ Output gate activity $\in (0,1)^h$
$\vec{h}_t \coloneqq$ Hidden state (output vector) $\in (-1,1)^h$
$\vec{\tilde{c}}_t \coloneqq$ Cell input activation vector $\in (-1,1)^h$
$\vec{c}_t \coloneqq$ Cell state vector $\in \mathbb{R}^h$
$\mathbf{W} \coloneqq$ Input Weights $\in \mathbb{R}^{h \times d}$
$\mathbf{U} \coloneqq$ Recurrent Weights $\in \mathbb{R}^{h \times h}$
$\vec{b} \coloneqq$ Bias $\in \mathbb{R}^h$
$\vec{\sigma}_s \coloneqq$ Sigmoid function
$\vec{\sigma}_t \coloneqq$ Hyperbolic tangent function

LSTMs are appropriate when working with sequential data, such as time series, because of their recurrent nature. Because the same operation is applied at each time step, time-shift equivariant properties of the data can relatively easily be discovered by such networks. A downside of LSTMs is that they can be quite slow due to the fact that they process the data sequentially.

LSTM layers take a sequence as input and output a sequence, $\vec{h}_0, \ldots, \vec{h}_T$. To get a single vector as output, a common solution is to only look at the final hidden vector $\vec{h}_T$. For binary classification, this vector can be fed through a MLP with a single-neuron output and sigmoid activation function. This output is interpreted as the probability for class 1.

2.3.3 Transformer

Transformers, introduced by Vaswani et al. in 2017 [40], have quickly grown to be the dominant deep learning models for sequential data.

A transformer consists of an encoder and a decoder, but for classification tasks, an encoder is typically sufficient. An encoder consists of multiple encoder layers, as illustrated in Figure 2.17.

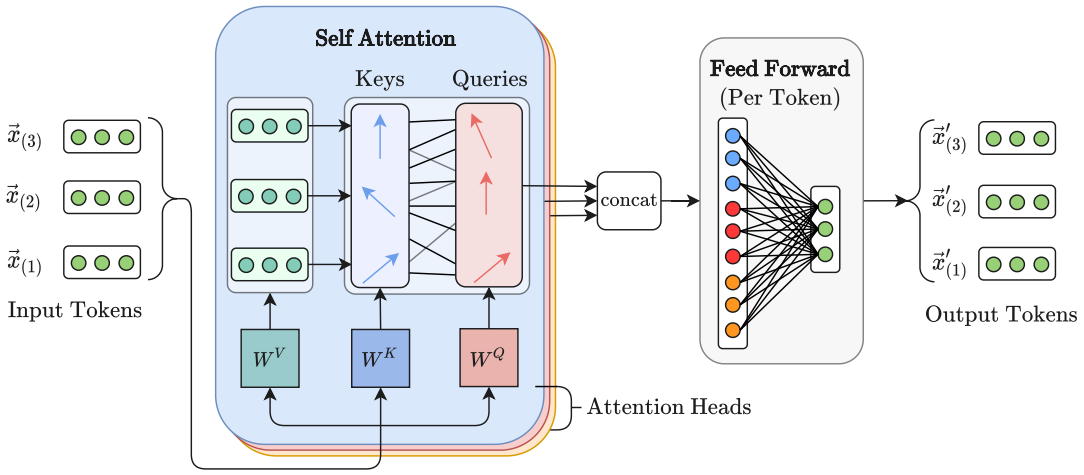


Figure 2.17: Illustration of a Transformer Encoder layer.

The basic formula for the forward pass of a transformer encoder layer is shown in Equation 2.22.

$$\begin{aligned}
\text{MultiHeadSelfAttention}(\mathbf{X}) &= \text{MultiHead}(\mathbf{X}, \mathbf{X}, \mathbf{X}) && (2.22)\\
\text{where MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O && (2.23)\\
\text{where head}_i &= \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) && (2.24)\\
\text{where Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V} && (2.25)
\end{aligned}$$

h := Number of attention heads
d_k := Dimensionality of the key vectors
$\mathbf{W}^O$:= Output matrix
$\mathbf{W}^Q$:= Query matrix
$\mathbf{W}^K$:= Key matrix
$\mathbf{W}^V$:= Value matrix

Transformers do not inherently treat the order of input tokens differently — it is permutation equivariant. If positionality is a desired feature, a position embedding is typically added to the tokens before feeding them into the transformer [40].

Similar to LSTM layers, transformers take a sequence of vectors as input and output another sequence of vectors. There are various ways of getting a single vector output from a transformer, but a simple solution is to average over the output tokens. For binary classification, this vector can be fed through an MLP with a single-neuron output and sigmoid activation function. This output is interpreted as the probability for class 1.

2.3.4 Convolutional Neural Network (CNN)

Convolutional neural networks are among the most popular neural network architectures for classification, since their success on ImageNet in 2012 [42]. They are most commonly known in their two-dimensional form, for classification of images, but they are also relatively common for sequence classification in their one-dimensional form. A one-dimensional CNN layer is illustrated in Figure 2.18.

As Figure 2.18 shows, the CNN layer moves a filter (kernel) along the input and computes a new value at each stride. If multiple filters are used, the layer will have multichannel output, one vector per filter. The CNN layer can be followed by a linear (dense) layer for classification purposes.

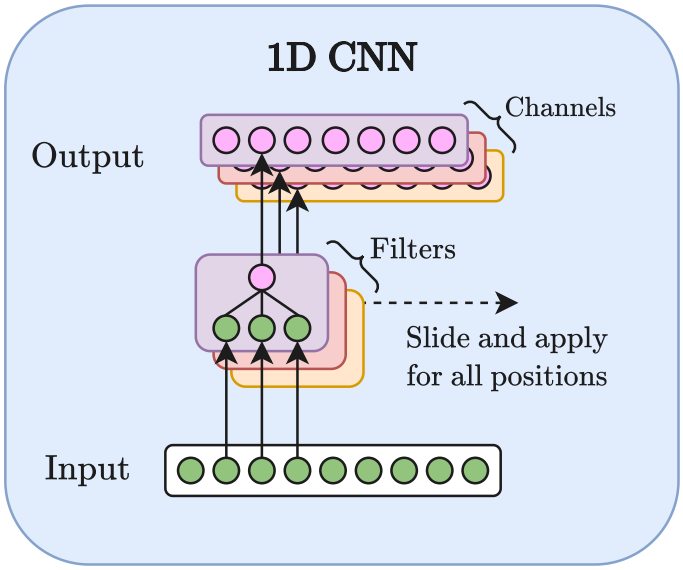


Figure 2.18: Illustration of a one-dimensional CNN.

The forward pass of a one-dimensional convolutional layer l can be defined as in Equation 2.26 [22].

$$V_{i,c} = g\left(\sum_{j=1}^{L_f}\sum_{n=1}^{N_f} w_{j,n,c}x_{s(i-1)+j,n} - \theta_c\right) \tag{2.26}$$

$L_f \coloneqq$ Length of the filters, $L_f \in \mathbb{Z}^+$
$N_f \coloneqq$ Number of filters, $N_f \in \mathbb{Z}^+$
$V_{i,c} \coloneqq$ Output i in channel c
$x_{m,n} \coloneqq$ Value m in channel k input vector $\vec{x}_k$
$w_{j,n,c} \coloneqq$ Weight j of filter n in channel c
$s \coloneqq$ Stride, $s \in \mathbb{Z}^+$
$\theta_c \coloneqq$ The threshold (bias) for channel c
$g \coloneqq$ Activation function (see Appendix A.2)

2.4 Bayesian Hyperparameter Optimization

Bayesian optimization is an algorithm that can be used to find hyperparameters that maximize or minimize different metrics [43]. Say that the objective is to maximize the classification accuracy when searching through combinations of hyperparameters. The objective function $f : \mathcal{X} \to \mathbb{R}$ then maps hyperparameter sets $\mathbf{X} \in \mathcal{X}$ to classification accuracies $c \in \mathbb{R}$. This function is, however, unknown and needs to be approximated. This is done based on Bayes' rule and using so-called surrogate models that map hyperparameter sets $\mathbf{X}$ to probability distributions over accuracies.

Bayesian hyperparameter optimization works by iterating over two basic steps: 1) sampling new hyperparameters $\mathbf{X}$ informed by the surrogate model and 2) updating the surrogate model with the information from the new sample. This can be repeated until the global optimum is found.

The sampling is done by selecting the hyperparameter set $\mathbf{X}$ which maximizes a function $u(\mathbf{X})$, referred to as a utility, or acquisition function [43]. The choice of which acquisition function to use determines the trade-off between exploration (improving the surrogate model) and exploitation (finding hyperparameters $\mathbf{X}$ that optimize f). Exploration is important for avoiding getting stuck in local optima.

The set of observed pairs of hyperparameter sets $\mathbf{X}$ and their corresponding classification accuracies c can be denoted as $\mathcal{D}$. The surrogate models are updated by computing a posterior distribution from $\mathcal{D}$.

One example of a surrogate model is a Gaussian process (GP). The prior of the objective f can in this case be defined as in Equation 2.27, where μ denotes the mean and K denotes the variance. Moreover, the posterior of the objective can be defined as in Equation 2.28 [43].

$$P(f) = \mathrm{GP}(f; \mu, K) \tag{2.27}$$

$$P(f \mid \mathcal{D}) = \mathrm{GP}(f; \mu_{f|\mathcal{D}}, K_{f|\mathcal{D}}) \tag{2.28}$$

An example of an acquisition function is expected improvement (EI), which is defined as in Equation 2.29 [43]. Here, I is the improvement function, $f_{t+1}(\mathbf{X})$ denotes the accuracy for hyperparameter set $\mathbf{X}$ at iteration $t+1$, and $f(\mathbf{X}^*)$ denotes the currently highest observed accuracy, obtained with hyperparameters $\mathbf{X}^*$. Using EI can be thought of as rewarding hyperparameters that improve on the best seen metric value so far, while ignoring the magnitude of any lack of improvement.

$$\mathbb{E}[I(\mathbf{X})] = \mathbb{E}[\max(0, f_{t+1}(\mathbf{X}) - f(\mathbf{X}^*))] \tag{2.29}$$

The hyperparameter set that maximizes the expected improvement (see Equation 2.30) is then chosen for the next iteration [43].

$$\mathbf{X} = \arg\max \mathbb{E}[I(\mathbf{X})] \tag{2.30}$$

The last step before the next iteration of the algorithm is to update $\mathcal{D}$ with the observed pair in the current iteration, meaning: $\mathcal{D} \leftarrow \{\mathcal{D}, (\mathbf{X}_t, c_t)\}$ [43].

3

Methods

This chapter describes how the work in this Master's thesis project has been carried out in different aspects. The preceding data collection and preprocessing is also explained.

3.1 Provided Data

This section describes the provided data; how it was collected, preprocessed and structured before making it available for this thesis.

3.1.1 Data Collection and Storage

MedTech West collected the data from 20 healthy men at the Karolinska Institute in Solna, Sweden, with MEG (see Section 1.1.2). One subject proved to be an outlier since its MSNA activity reduction from the baseline was -132%, which is far from the values of the other patients. The data of one of the patients was corrupt and was therefore discarded from the data set. In total, 102 magnetometers and 204 gradiometers were placed on each subject's head, summing up to 306 sensors [17]. An electrooculogram (EOG) and an electrocardiogram (ECG) were also recorded for preprocessing purposes later on [3].

When collecting the data from a subject, electric stimulations were made on their left finger. There were 72 stimulus trials per subject, consisting of three electrical shocks each. Each shock where induced every other heartbeat (approximately every 1.5–2 seconds). There was a resting period of 30, 45, or 60 seconds in between each trial. Each subject received the same rest sequence [17]. These shocks will henceforth be referred to as pulse 1, 2, and 3.

The recorded MEG data from the 306 sensors each cover a range of [-2000ms, 2000ms] relative to the electric pulse. This has been cropped to [-100ms,1500ms] in order to contain only the most relevant information, as well as ensuring that all data can fit in memory on the available hardware. A breakdown of the available data is illustrated in Figure 3.1.

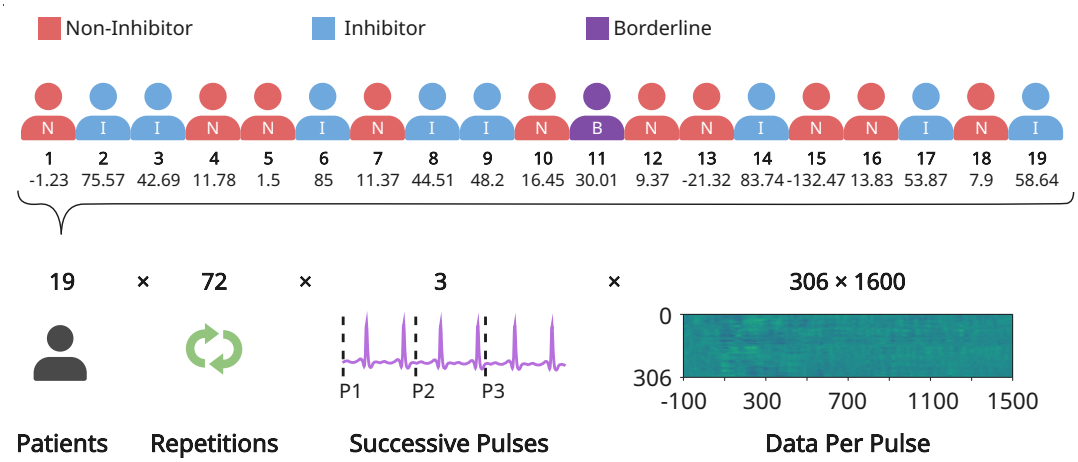


Figure 3.1: Subject data breakdown. The available data comes from 19 subjects. There were 72 stimulus trials per subject, consisting of 3 electrical shocks each. For each shock, time series data was collected by 306 sensors, which after some preprocessing covered the range [-100ms, 1500ms] relative to the electric stimulus.

In the previous theses [15], [16], two versions of the data set were created — one called A which does not contain the borderline subject and one called B which does. The borderline subject is visualized as the purple avatar in Figure 3.1. Both of these data sets have been used during this project like before to see if they result in different prediction accuracies.

The MEG data is stored in a set of `.fiff` files, one for each patient. These are read using *mne* [44], which is a Python package for interfacing with the *fieldtrip* [45] MATLAB toolbox for MEG, EEG, and iEEG analysis. These files contain the preprocessed MEG data in a tabular form, with columns for Epoch, Pulse Number, milliseconds before/after the pulse, as well as the data for each MEG sensor.

3.1.2 Preprocessing

After collecting the MEG data, it was preprocessed in a number of ways: temporal signal space separation (tSSS), head movement compensation, 0.5–40 Hz filtering, Independent Component Analysis (ICA), and manual removal of epochs with residual ocular artifacts based on visual inspection [3], [17].

ICA was performed to remove irrelevant data artifacts such as signals corresponding to eye blinks that were prone to happen at stimulus [17]. ICA was first performed on the raw MEG data in order to split it into statistically independent components (ICs). The MEG ICs that corresponded to artifacts in the EOG and the ECG were filtered away from the MEG data by analyzing the correlations between the respective signals [3]. This process however left some epochs with artifact residuals which were discarded based on visual inspection [17].

3.2 Data Split

This section describes how the data was split in previous years, and what was done differently in this thesis, including the use of cross-validation.

3.2.1 Old and New Data Split

In the previous projects, they used a split where they had a held-out test set of two patients, and used the remaining patients for training and validation. This data was randomly split on a per-time-series basis. This resulted in a data split where the training and validation sets were unique, but where the same patient could contribute with data to both sets. The fact that the same patient could contribute with data to both the training and validation could explain the large discrepancy between validation and test accuracies in the previous theses. The old data split is visualized in Figure 3.2.

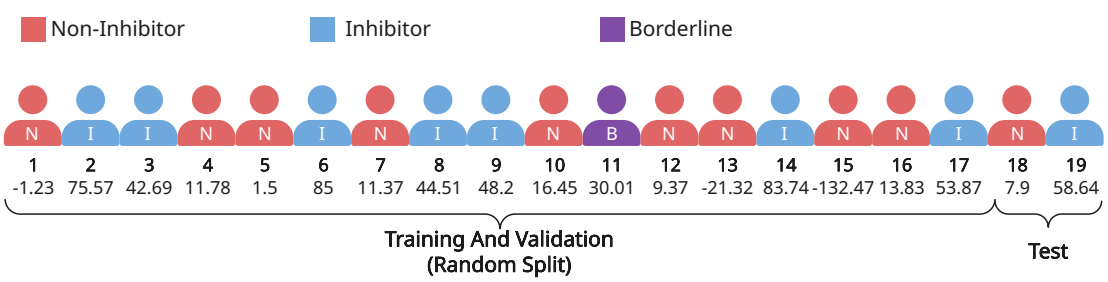


Figure 3.2: Data split used in previous iterations of the project. The numbers in bold are the anonymized patient identification numbers and the numbers below them are the respective MSNA activity reduction percentages.

To circumvent this issue in this project, the split was instead done as illustrated in Figure 3.3. The split between validation and training subjects was however not constant during training since cross-validation was always performed (see Section 3.2.2).

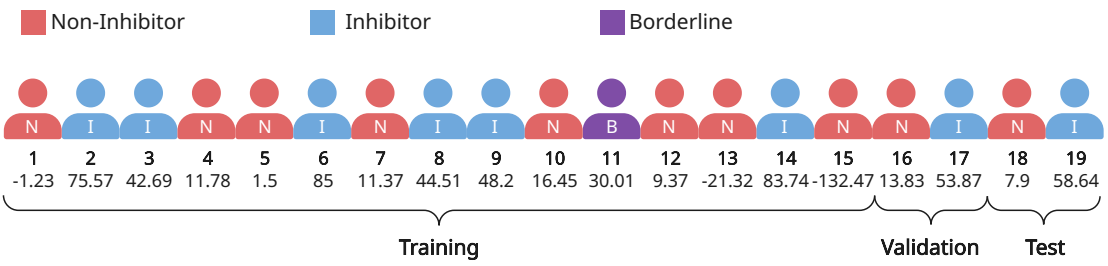


Figure 3.3: Data split used in this project. The numbers in bold are the anonymized patient identification numbers and the numbers below them are the respective MSNA activity reduction percentages.

Lastly, the results presented in Chapter 4 of this thesis cannot be directly compared with the ones in the previous theses since the data split is different here.

3.2.2 Cross-Validation

Although using a split like in Figure 3.3 reduces the bias in estimating how well the model generalizes to unseen patients, it increases the variance of the validation accuracy since the training and validation sets do not include data from the same people. A higher variance leads to less reliable estimates. To combat this issue, cross-validation was introduced. This means that, instead of having a fixed set of patients assigned to the validation set – models were trained multiple times with different patient subsets used for validation in so-called cross-validation folds. The extreme case of leave one out cross-validation (LOOCV) was only used for the reproduction of the 2020 LSTM model because of the low number of hyperparameter combinations. In other cases, it was not feasible to use LOOCV due to time constraints. Because of this, four cross-validation folds were used in general.

3.3 Used Artificial Neural Networks

This thesis project has mainly focused on four ANN model types: LSTM, MLP, CNN, and Transformer. Autoencoders were also used in some instances to remove noisy data. See Section 2.3 and Section 2.1.1 for model specific theory. This section will motivate the use of each model and describe the chosen architectures.

Note that the LSTM and MLP models only work on univariate data, which means that they can only take input from one MEG sensor at a time in this context. The CNN and Transformer models can however handle multivariate data, meaning that they can be fed data from all sensors in one step.

The univariate models are trained on the data available from all sensors and are therefore tasked with treating all sensors the same way. The benefit of this is that many more data points are available for training, while a potential downside is that the model may be forced to classify time series which contain no information about MSNA inhibition. Another downside is that the time series of different sensors could have opposing distributions in terms of which signal corresponds to which class.

The multivariate models have the opposite problem. They always have access to all signals relevant for MSNA inhibition, and (depending on architecture) there is no risk of confusing distributions between sensors. On the other hand, they have 306 times fewer data points to train on.

3.3.1 Long Short-Term Memory (LSTM)

The existing LSTM model from previous years [15], [16] was first recreated in order to be able to compare new methods to the old ones. This had to be done since the results between this thesis and the others are not directly comparable, as explained in Section 3.2.

When recreating the LSTM and analyzing the code from 2020, it was found that the model had been implemented incorrectly. This conclusion was based on that the description of the model did not seem to match the actual implementation. Instead of correctly feeding the LSTM the time series one time step at a time (see Figure 3.4), it was fed the whole time series as one vector and thus only performed one iteration (see Figure 3.5). This LSTM model type was hence not used properly and will therefore be called the “incorrect LSTM” (“ILSTM” for short) from here on out. Despite this, it was decided that the ILSTM would be used in this thesis as well since the correct one was unfeasibly slow given the time-frame. It was also worth investigating whether the model type could benefit from other types of data preprocessing and augmentation methods than the ones presented in the previous theses.

The ReLU activation function was used between the LSTM layers and the sigmoid activation function was used for the output neuron.

Sweeps were performed over the number of sub-layers in the LSTM layers, the hidden state sizes, and the drop out probabilities to hopefully find better versions of the previous ILSTM architecture. For more in-depth theory on LSTMs, see Section

2.3.2.

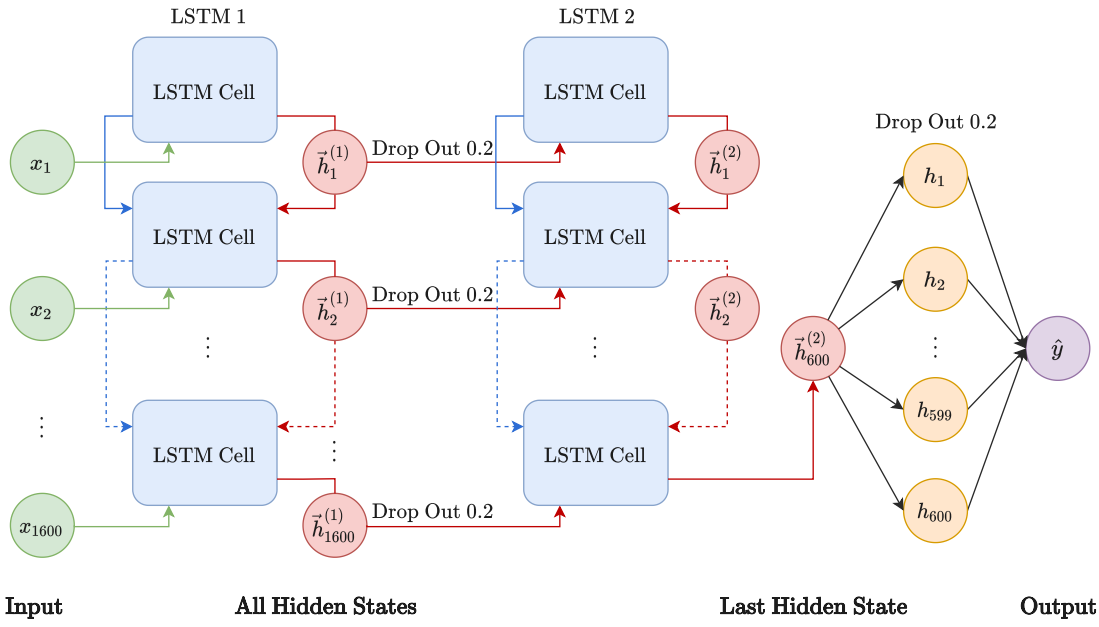


Figure 3.4: The seemingly intended LSTM architecture from the previous theses.

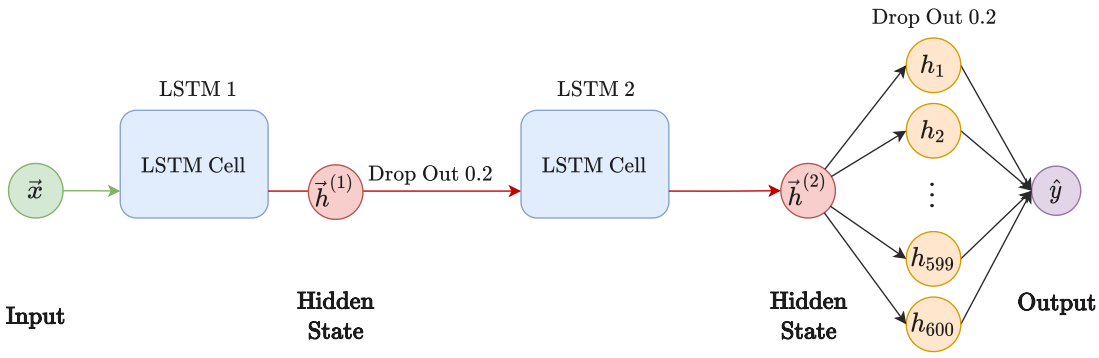


Figure 3.5: The incorrect LSTM (ILSTM) architecture from the previous theses.

3.3.2 Multilayer Perceptron (MLP)

Because of the incorrect properties of the ILSTM model from previous years (see Section 3.3.1), it was thought that the ILSTM layers would work similarly to linear, or dense, layers. Therefore, a model using linear layers instead of LSTM layers was implemented to potentially simplify the architecture and perhaps get the same results. This model thus became a multilayer perceptron (MLP, see Section 2.3.1 for theory). ReLU was used in all layers except for the output neuron which used the sigmoid activation function, similar to the ILSTM model. An example of this

architecture with corresponding hyperparameters to the one in Figure 3.5 is shown in Figure 3.6.

Sweeps were performed over the drop out probabilities and the number of neurons in the hidden layers in order to find relatively good versions of the architecture.

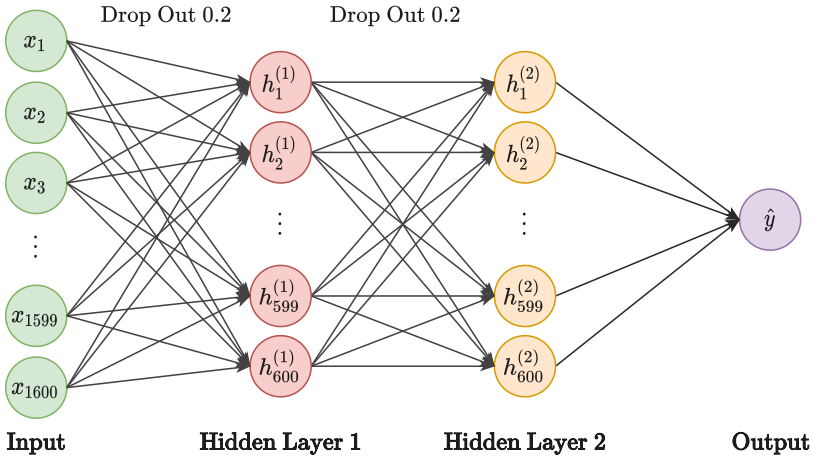


Figure 3.6: The MLP architecture designed to mimic the ILSTM model from the previous theses.

3.3.3 Convolutional Neural Network (CNN)

One motivation for using a one-dimensional convolutional neural network for the classification problem at hand was that it can learn to find local features in a time series by applying its filters (kernels) along it during training. Another motivation was that CNNs are well studied.

Figure 3.7 illustrates the CNN architecture designed in this project. Each convolutional layer (Conv1D) has outputs with different receptive fields that each only cover part of the input. The first convolutional layer in the illustration is followed by drop out and a one-dimensional max pooling layer that extracts the most important information from the convolutional layer's receptive fields. These two layers were made repeatable so that different depths of the architecture could be tested during sweeps.

The max pooling layer is followed by another convolutional layer that looks for features in its output. By then applying global average pooling, the average over the receptive fields is computed for each input channel. Each filter thus essentially votes for class features. These averages are then linearly combined to form an output $\hat{y}$. Each average value a_c represents the found features in its corresponding input time series.

Each convolutional layer uses a ReLU activation function to add non-linearity (see Appendix A.2 for theory). The last layer utilizes the sigmoid activation function to output a probability for MSNA inhibition. Binary cross-entropy is then used as the loss function (see Appendix A.3 for theory).

Sweeps were performed over the depth of the network, the length of the filters L_f,

the number of filters N_f (and thus the number of channels in hidden layers), and the drop out probability.

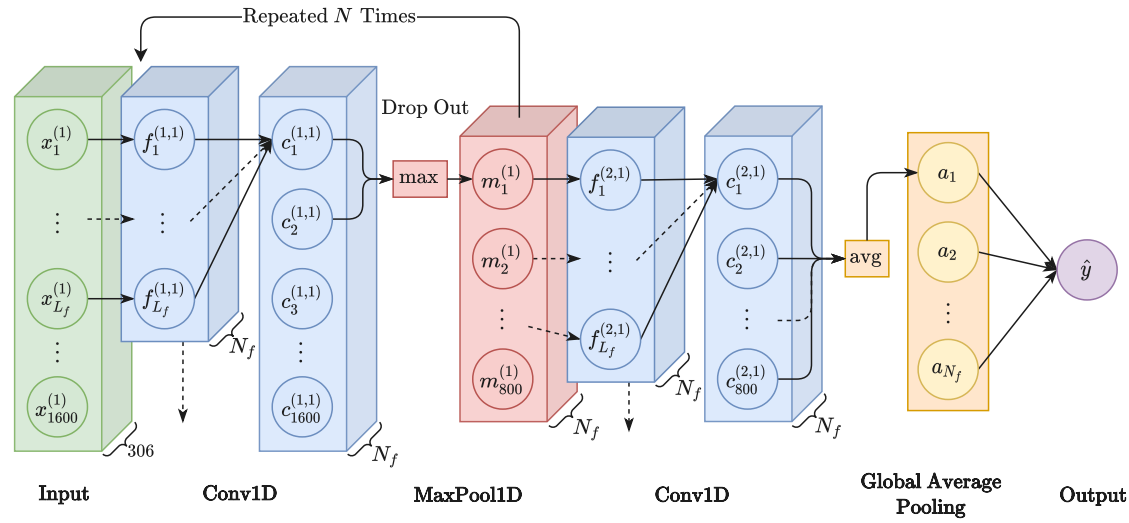


Figure 3.7: The generic CNN model architecture used in this thesis. The time series length is 1600 and there are 306 sensors. N_f denotes the number of convolutional filters. L_f denotes the length of the filters.

3.3.4 Transformer

As an alternative to the CNN, a transformer was also implemented. This decision was primarily informed by two factors. Firstly, when working with the multivariate representation of the dataset there is a substantial risk of overfitting. This is because treating the time series from all the sensors as a single data point effectively reduces the dataset size by a factor of 306. The issue is then further exacerbated by the fact that more information is made available in which spurious correlations could potentially be found. Secondly, according to MedTech West, the MEG sensors in the helmet are not lined up to the exact same brain regions for all patients since their heads and brains are different from each other. The same sensor does therefore not always correspond to activity in the same brain region between patients. Instead of having the model learn to ignore the sensor index, it was reasoned that a transformer, which inherently has this property, could be used.

Figure 3.8 illustrates the transformer architecture used in this project. In this case, the sensor index is the sequence dimension, while time is the channel dimension. The reason for this is to keep the sensor index equivariance property described previously. First, a Conv1D layer is used to project each sensor sequence into tokens of a smaller embedding space. These tokens are then fed through N transformer encoder layers, as explained in Section 2.3.3.

Finally, the tokens are globally averaged and linearly combined with a dense layer. Similar to the global average pooling in the CNN architecture (see Section 3.3.3), the tokens essentially vote for class features. The final layer then utilizes the sigmoid

activation function to output a probability for MSNA inhibition. Binary cross-entropy is then used as the loss function (see Appendix A.3 for theory).

When the kernel (filter) size in the initial convolution layer is 1, all layers are equivariant with respect to the sensor ordering. The final global averaging then ensures that the output is completely invariant with respect to this ordering, meaning that the same prediction will be made regardless of how the sensors are ordered.

With other kernel sizes, this property is lost, but it could be argued that extracting information directly based on position will be less trivial. This means that the architecture might still encourage gradient descent to focus on other information in the data than the positionality.

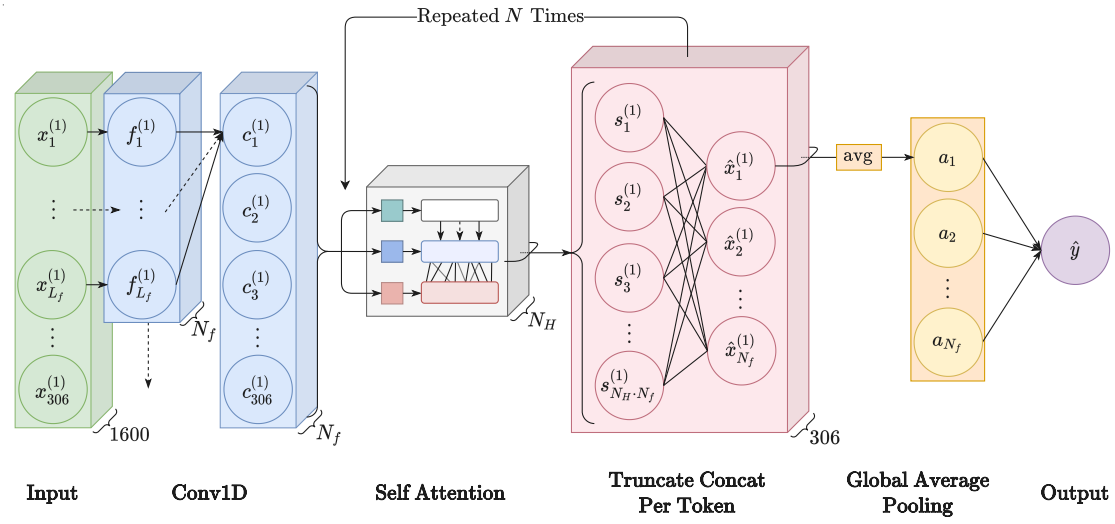


Figure 3.8: An example of the transformer model architecture. The time series length is 1600 and there are 306 sensors. N_f denotes the number of convolutional filters. L_f denotes the length of the filters. N_H denotes the number of attention heads in the self attention layer.

3.3.5 Autoencoder

In order to draw stronger conclusions about the usage of autoencoders in the context of this thesis, different variations were tested than the one used in the 2020 thesis. This was done for each model type described in this chapter to see whether it would be beneficial or not in different cases. The depth of the autoencoder as well as the size of the encoded layer were varied. Such hyperparameter sweeps were performed once per the best found variation of each classification model used in this thesis (ILSTM, MLP, CNN, and transformer). The motivation for this being that different autoencoder architectures might perform better in different contexts.

Autoencoders with linear layers were used in univariate data settings and CNN autoencoders were used in multivariate settings. In both cases and for all depths, the ReLU activation function was used in the encoder part of the autoencoder, just like in the 2020 model. See Section 2.1.1 for theory on autoencoders.

3.4 Training Artificial Neural Network Models

This section describes the process by which artificial neural network models were trained and evaluated in this project.

3.4.1 Pipeline

The model-data pipeline used for every cross-validation fold (see Section 3.2.2) and model is illustrated in Figure 3.9.

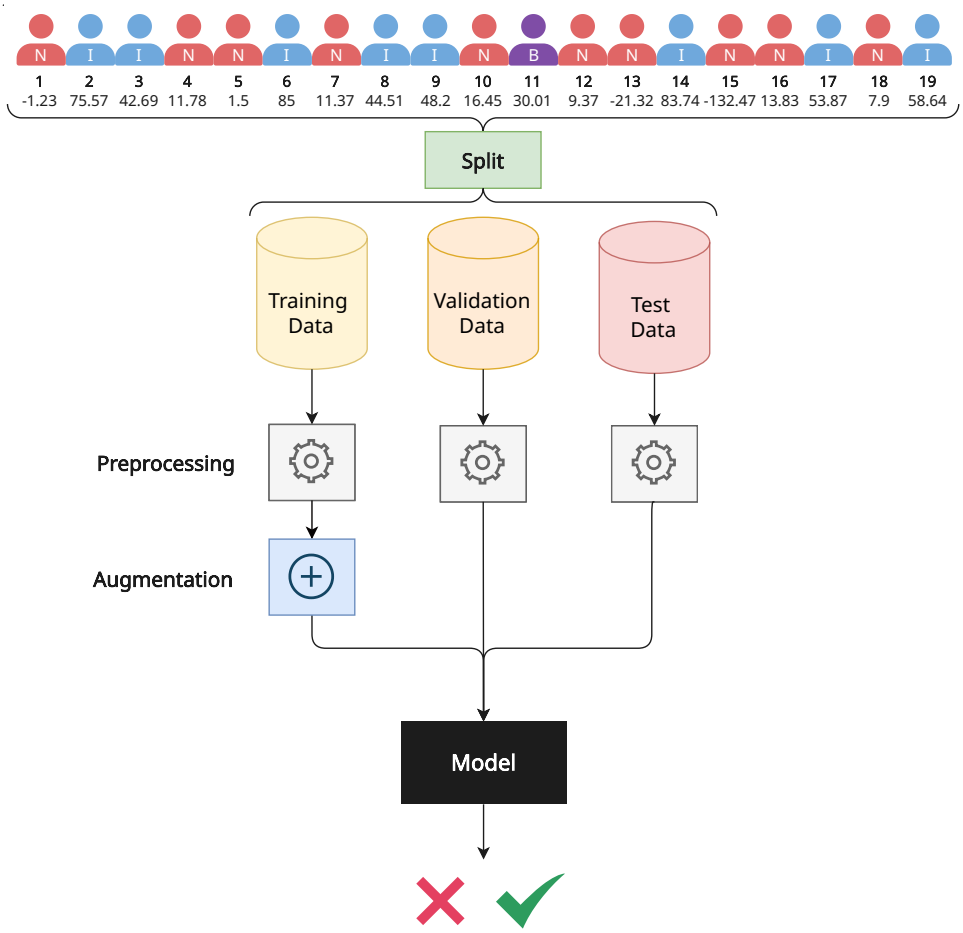


Figure 3.9: Illustration of the model-data pipeline. The cross and check marks denote the model's predictions for MSNA inhibition and non-inhibition. The numbers in bold are the anonymized patient identification numbers and the numbers below them are the respective MSNA activity reduction percentages.

First, the data from all patients was collected and split into three datasets: training, validation, and testing (see Section 3.2). Note that the test set was constant and held-out, as described in Section 3.4.2, for all cross-validation folds. Secondly, if preprocessing was used, it was applied to all dataset splits. Augmentation was then only applied on the training data when used.

The resulting data, whether modified or not, could then be fed for training into the ANN model which is illustrated as a black box in Figure 3.9. The trained model could then be used to make predictions.

Note that this black box could represent other types of machine learning models than ANNs. That is, however, not the focus of this thesis, as mentioned before.

3.4.2 Training, Validation, And Testing

The standard machine learning procedures for training, validation, and testing were followed in this project. The model parameters were optimized based on the training set and the hyperparameters based on the validation set. The test set was held-out and only used to evaluate the models at the end of the project. This in order to avoid manually optimizing based on it, which would result in overfitted models that do not generalize well on new data. It was also done in order to be able to estimate the generalization ability of models. The data was also normalized before use to increase the numerical stability of the models.

To get more reliable statistics about validation accuracy, cross-validation was used, as mentioned in Section 3.2.2. Hyperparameters were tuned to maximize the average validation accuracy over all cross-validation folds.

Testing was done only after being content with the trained model by evaluating the accuracy of it on the previously held-out test set.

3.4.3 Hyperparameter Tuning

Weights & Biases [46] was used for experiment tracking and hyperparameter tuning. In particular, it has a built-in feature for doing Bayesian hyperparameter optimization (see Section 2.4), which was useful for determining optimal architectures and augmentation operations. In the cases where the number of hyperparameters combinations was relatively low, grid search was performed instead in order to use each combination and see how it performed.

To further accelerate the search for good models, early stopping was used. Early stopping ensures that models stop training when they stop improving. This in contrast to stopping after a fixed number of training iterations. The most commonly used stopping criterion is when the validation loss/accuracy has not improved for a set number of epochs. Because of the small dataset size, in this project the criterion that was minimized is the minimum of validation and loss accuracy. This ensured that runs did not stop early just because they happened to be initialized in such a way that they performed ok on the validation set.

3.4.4 Motivations Behind Collected Results

For all models mentioned in Section 3.3, sweeps over their hyperparameter combinations were performed in order to find relatively good architectures (see Section 3.4.3). The variations with the highest validation accuracy averages were selected for subsequent data preprocessing and augmentation sweeps. Only using the best found architecture per model saved a lot of time.

As mentioned before, an autoencoder sweep per model was done to find a relatively good autoencoder architecture with respect to each model. One reason for this was to see if the autoencoder would help on its own to yield higher prediction accuracies. Another reason was to save time in subsequent sweeps by only training each autoencoder once per data pulse and category as well as making the sweep focus on hyperparameters of other types of transforms.

The motivation for presenting results with wavelet preprocessing transforms separately (see Section 2.1.2) was that MedTech West proposed that the transform could be beneficial to the problem at hand. This is because certain brain activity frequencies have stronger correlations with MSNA inhibition. It was therefore decided that it was relevant to show those results without other transforms. The use of the best found autoencoder in the context was the only exception, since it did not hurt to gather results with the addition of an autoencoder as well.

Lastly, sweeps over different transforms were done with the motivation that Bayesian hyperparameter optimization (see Section 2.4) would find relatively good combinations and that it would save time compared to running one sweep per transform. All transforms described in Chapter 2 were included in each transform combination sweep.

3.5 Implementation

Throughout the project, a Python code base was developed to easily carry out and log different types of experiments. This code base is flexible and could be used for other experiments in the future. It can therefore be seen as having intrinsic value beyond the results produced in this particular project.

The primary frameworks used in the implementation have been Numpy [47], Pandas [48], [49], PyTorch [50], PyTorch Lightning [51], Jax [52], and the WandB API [46].

4

Results

This chapter presents the results of this thesis. The results have been obtained as explained in Chapter 3 with the ANN architectures described in Section 3.3. Only the new data split explained in Section 3.2 has been used. Moreover, all presented prediction accuracies are averages over the last epoch accuracy for all cross-validation folds (see Section 3.2.2).

The "best" model or hyperparameter combination will refer to the one that resulted in the highest validation accuracy average. The test accuracies were computed in later stages, as explained in Section 3.4.2.

Each parallel coordinate plot in this chapter visualizes a hyperparameter sweep and emphasizes the best hyperparameter combination and the training, validation, and test accuracies it led to.

Note that when an autoencoder was used for preprocessing, it was used before any other transform.

For a summary of all obtained training, validation, and test accuracies for all model and transform sweeps, see Section 4.5.

4.1 Incorrect Long Short Term Memory (ILSTM)

This section presents results obtained with the incorrect LSTM (ILSTM). The motivation for using this model even though it is incorrectly implemented is found in Section 3.3.1. Specifically, results here were obtained with different architectures, autoencoders, wavelet transforms, and data transform combinations.

4.1.1 Reproduction From Previous Thesis

The prediction accuracies from the reproduction of the 2020 thesis' ILSTM and autoencoder is illustrated with a parallel coordinates plot in Figure 4.1. LOOCV was used to obtain these results. The best combination of hyperparameters is emphasized with a thicker, orange curve. The hyperparameters of the models in question here are shown in Table 4.1. The accuracies of the best hyperparameters is shown in Table 4.2.

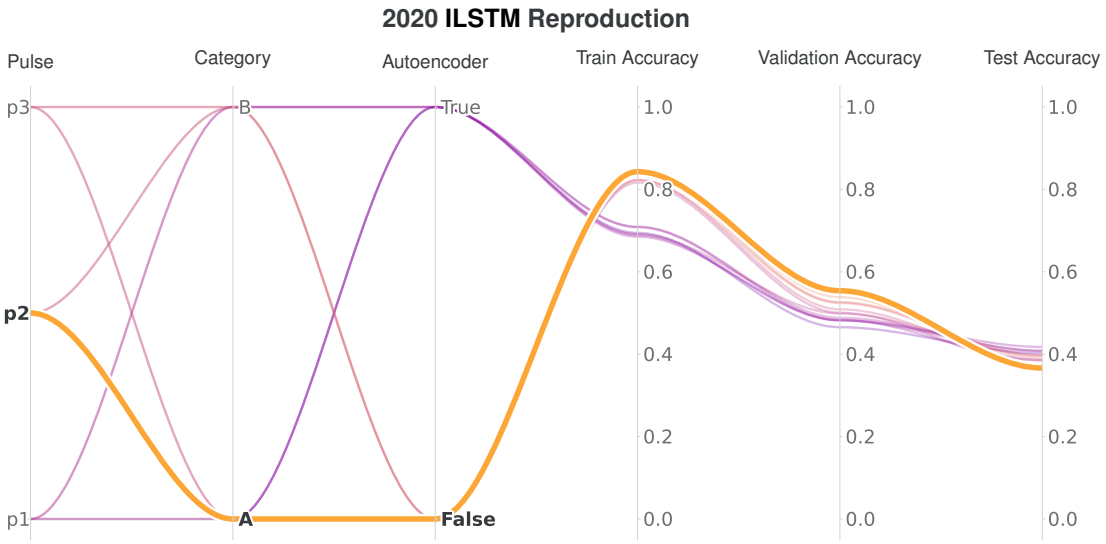


Figure 4.1: Parallel coordinates plot showing the LSTM reproduction sweep results. The thick line corresponds to the best identified configuration.

Table 4.1: The model hyperparameters of the autoencoder and ILSTM from the 2020 thesis [15].

Autoencoder Hyperparameter	**Value**
Encoding dimension	30
Input size	1600
ILSTM Hyperparameter	**Value**
Drop out	0.2
Hidden size	600
Input size	1600
Layer count	1

Table 4.2: Accuracies obtained with the hyperparameters in Table 4.1.

Stage	**Accuracy**
Training	84.30%
Validation	55.40%
Testing	36.65%

4.1.2 Different ILSTM Architectures

The prediction accuracies from a sweep over ILSTM hyperparameters is illustrated in Figure 4.2. Four cross-validation folds were used to obtain these results. The best combination of hyperparameters is emphasized with a thicker, orange curve. Figure 4.3 shows the training and validation accuracies over time. Table 4.3 presents the hyperparameters of the best ILSTM found in this thesis. The accuracies of the best hyperparameters is shown in Table 4.4.

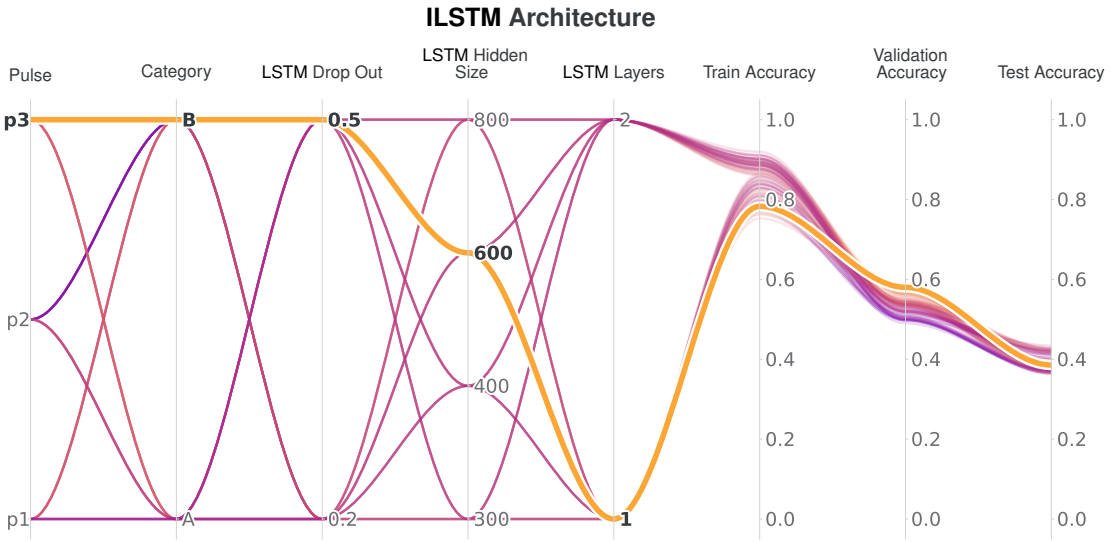
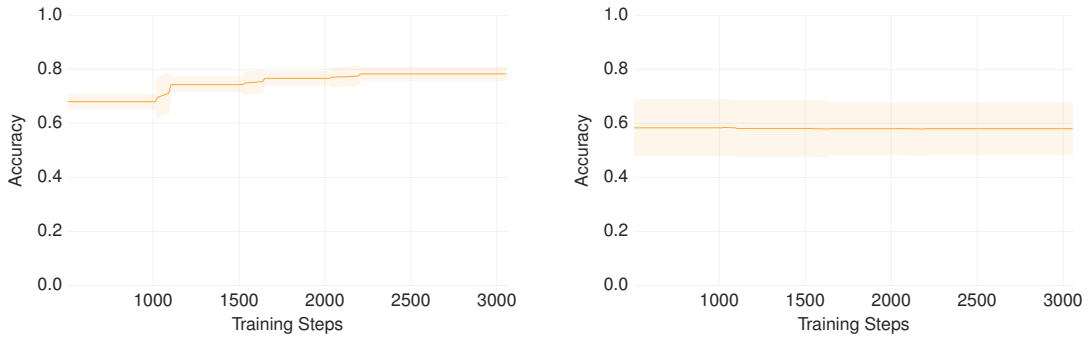


Figure 4.2: Parallel coordinates plot showing the sweep results of ILSTM architectures. The thick line corresponds to the best identified architecture and data combination.



(a) Training Accuracy **(b)** Validation Accuracy

Figure 4.3: Training and validation accuracy for the best identified ILSTM model identified in Figure 4.2 with a 95% confidence interval computed from the cross validation folds.

Table 4.3: The best data and ILSTM hyperparameters.

Data Hyperparameter	**Value**
Category	B
Pulse	3
ILSTM Hyperparameter	**Value**
Drop out	0.5
Hidden size	600
Input size	1600
Layer count	1

Table 4.4: Accuracies obtained with the hyperparameters in Table 4.3.

Stage	**Accuracy**
Training	78.29%
Validation	58.01%
Testing	38.56%

4.1.3 Best ILSTM With Autoencoder Preprocessing

The prediction accuracies from a sweep with the best ILSTM (see Section 4.1.2) and different autoencoder architectures is illustrated in Figure 4.4. Four cross-validation folds were used to obtain these results. The best combination of hyperparameters is emphasized with a thicker, orange curve. Figure 4.5 shows the training and validation accuracies over time. Table 4.5 presents the hyperparameters of the best autoencoder in combination with the best ILSTM. The accuracies of the best hyperparameters is shown in Table 4.6.

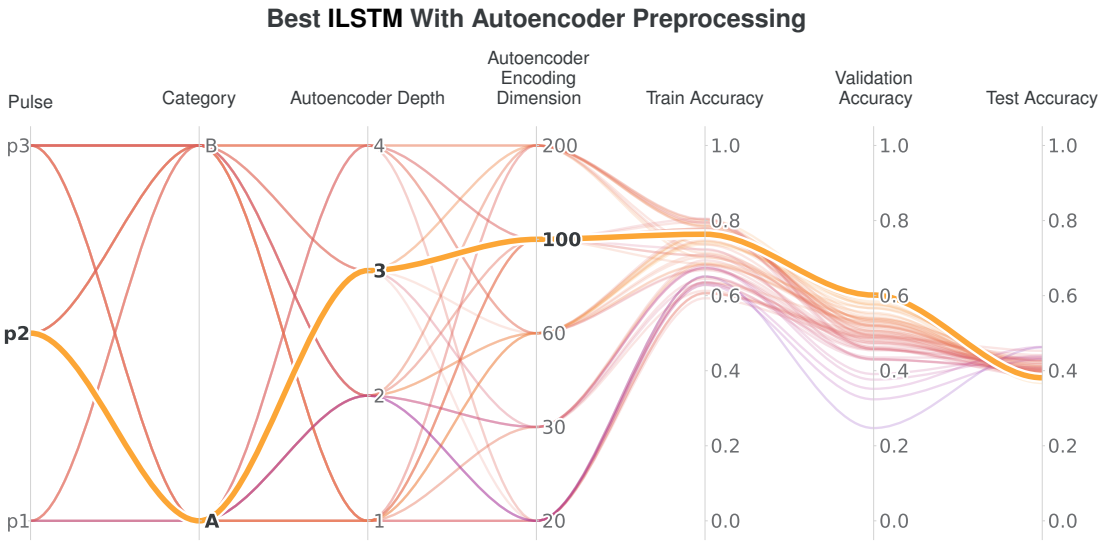
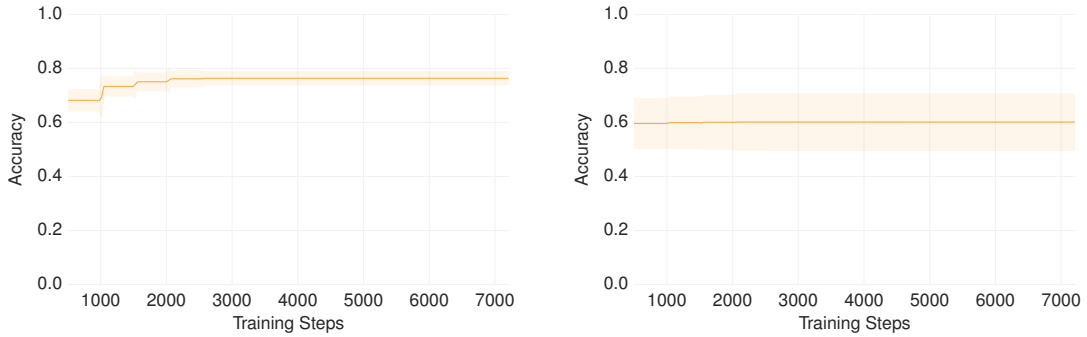


Figure 4.4: Parallel coordinates plot showing sweep results using the best ILSTM model identified in Figure 4.2 with different autoencoder architectures. The thick line corresponds to the best identified autoencoder hyperparameters and data combination.



(a) Training Accuracy **(b)** Validation Accuracy

Figure 4.5: Training and validation accuracy for the best configuration identified in Figure 4.4 with a 95% confidence interval computed from the cross validation folds.

Table 4.5: The best data and autoencoder hyperparameters when used together with the best ILSTM.

Data Hyperparameter	**Value**
Category	A
Pulse	2
Autoencoder Hyperparameter	**Value**
Depth	3
Encoding dimension	100
Input size	1600

Table 4.6: Accuracies obtained with the hyperparameters in Table 4.5.

Stage	**Accuracy**
Training	76.36%
Validation	60.15%
Testing	38.12%

4.1.4 Best ILSTM With Wavelet Preprocessing

The prediction accuracies from a sweep with the best ILSTM and autoencoder (see Section 4.1.2 and Section 4.1.3) as well as different wavelet transforms is illustrated in Figure 4.6. Four cross-validation folds were used to obtain these results. The best combination of hyperparameters is emphasized with a thicker, orange curve. Figure 4.7 shows the training and validation accuracies over time. Table 4.7 presents the best wavelet transform in combination with the best ILSTM. The accuracies of the best hyperparameters is shown in Table 4.8.

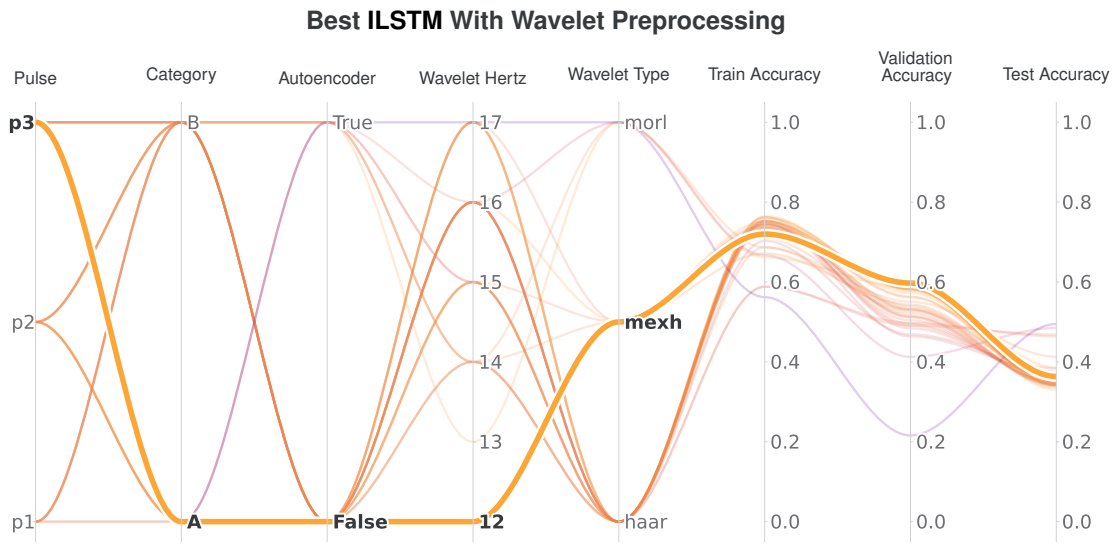
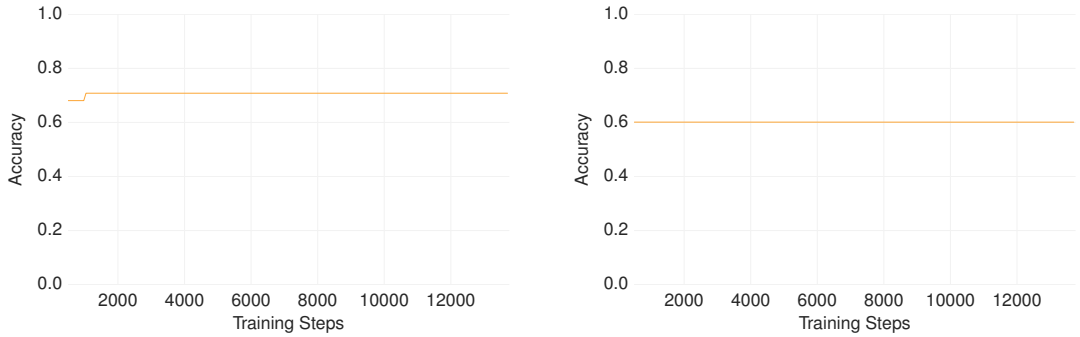


Figure 4.6: Parallel coordinates plot showing sweep results of the best ILSTM model identified in Figure 4.2 and (optionally) the best autoencoder identified in Figure 4.4 with different wavelet preprocessing transforms. The thick line corresponds to the best identified configuration.



(a) Training Accuracy **(b)** Validation Accuracy

Figure 4.7: Training and validation accuracy for the best wavelet configuration identified in Figure 4.6 with a 95% confidence interval computed from the cross validation folds.

Table 4.7: The best data and wavelet transform hyperparameters when used together with the best ILSTM.

Data Hyperparameter	**Value**
Category	A
Pulse	3
Wavelet Transform	**Value**
Frequency [Hz]	12
Wavelet	Mexican Hat

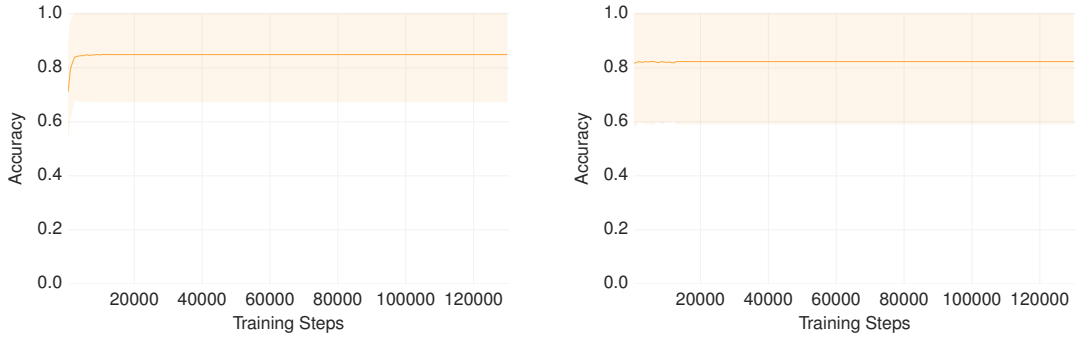
Table 4.8: Accuracies obtained with the hyperparameters in Table 4.7.

Stage	**Accuracy**
Training	72.01%
Validation	59.74%
Testing	36.35%

4.1.5 Best ILSTM With Transform Combinations

The prediction accuracies from a sweep with the best ILSTM and autoencoder (see Section 4.1.2 and Section 4.1.3) as well as different data transforms is illustrated in Figure 4.8. Four cross-validation folds were used to obtain these results. The best combination of hyperparameters is emphasized with a thicker, orange curve. Figure 4.9 shows the training and validation accuracies over time. Table 4.9 presents the best transform combination together with the best ILSTM. The accuracies of the best hyperparameters is shown in Table 4.10.

Figure 4.8: Parallel coordinates plot showing sweep results of the best ILSTM model identified in Figure 4.2 with random data transform combinations. The thick line corresponds to the best identified transform combination. "T" and "F" denote "True" and "False" respectively.



(a) Training Accuracy **(b)** Validation Accuracy

Figure 4.9: Training and validation accuracy for the best transform combination identified in Figure 4.8 with a 95% confidence interval computed from the cross validation folds.

Table 4.9: The best data and transform combination hyperparameters when used together with the best ILSTM. The transforms are listed in the order as they were applied during training/validation/testing.

Data Hyperparameter	**Value**
Category	B
Pulse	3
Autoencoder	**Value**
Depth	3
Encoding dimension	100
Input size	1600
Permutation	**Value**
Slice count	3
Magnitude Warp	**Value**
Knots	6
Standard deviation	0.1468
Time Warp	**Value**
Steps per warp	100
Warp factor	0.7911
Contrasted Linear Combination (CLC)	**Value**
Contrast	2.9930

Table 4.10: Accuracies obtained with the hyperparameters in Table 4.9.

Stage	**Accuracy**
Training	84.88%
Validation	82.29%
Testing	49.67%

4.2 Multilayer Perceptron (MLP)

This section presents results obtained with MLP models. Specifically, results here were obtained with different architectures, autoencoders, wavelet transforms, and data transform combinations.

4.2.1 Different MLP Architectures

The prediction accuracies from a sweep over MLP hyperparameters is illustrated in Figure 4.10. Four cross-validation folds were used to obtain these results. The best combination of hyperparameters is emphasized with a thicker, orange curve. Figure 4.11 shows the training and validation accuracies over time. Table 4.11 presents the hyperparameters of the best MLP found in this thesis. The accuracies of the best hyperparameters is shown in Table 4.12.

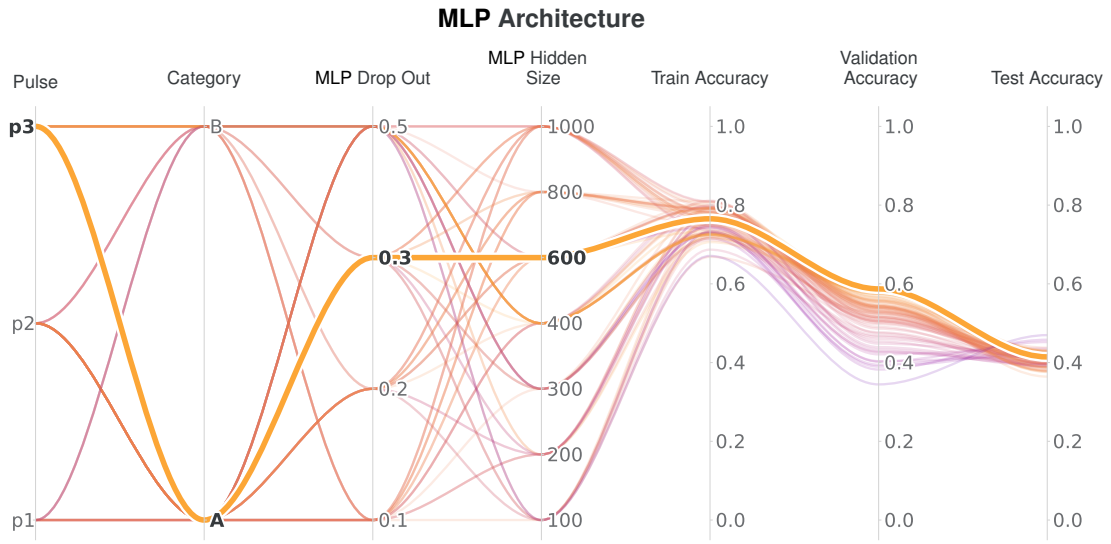
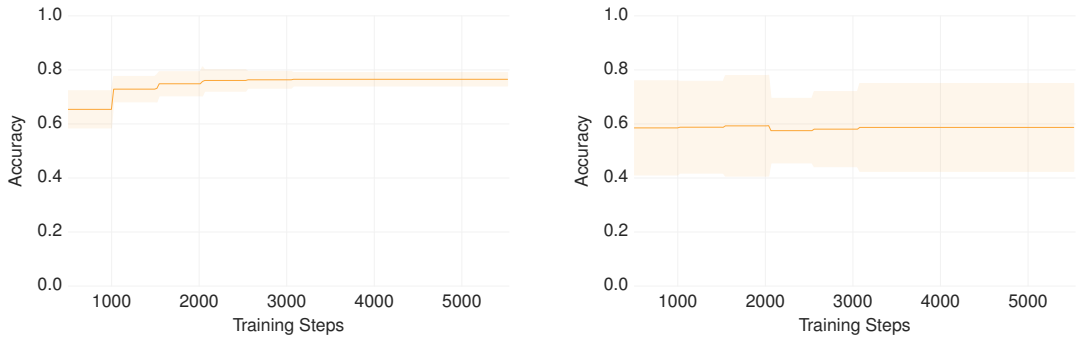


Figure 4.10: Parallel coordinates plot showing the sweep results of MLP architectures. The thick line corresponds to the best identified architecture and data combination.



(a) Training Accuracy **(b)** Validation Accuracy

Figure 4.11: Training and validation accuracy for the best identified MLP identified in Figure 4.10 with a 95% confidence interval computed from the cross validation folds.

Table 4.11: The best data and MLP hyperparameters.

Data Hyperparameter	**Value**
Category	A
Pulse	3
MLP Hyperparameter	**Value**
Drop out	0.3
Hidden size	600
Input size	1600

Table 4.12: Accuracies obtained with the hyperparameters in Table 4.11.

Stage	**Accuracy**
Training	76.53%
Validation	58.71%
Testing	41.44%

4.2.2 Best MLP With Autoencoder Preprocessing

The prediction accuracies from a sweep with the best MLP (see Section 4.2.1) and different autoencoder architectures is illustrated in Figure 4.12. Four cross-validation folds were used to obtain these results. The best combination of hyperparameters is emphasized with a thicker, orange curve. Figure 4.13 shows the training and validation accuracies over time. Table 4.13 presents the hyperparameters of the

best autoencoder in combination with the best MLP. The accuracies of the best hyperparameters is shown in Table 4.14.

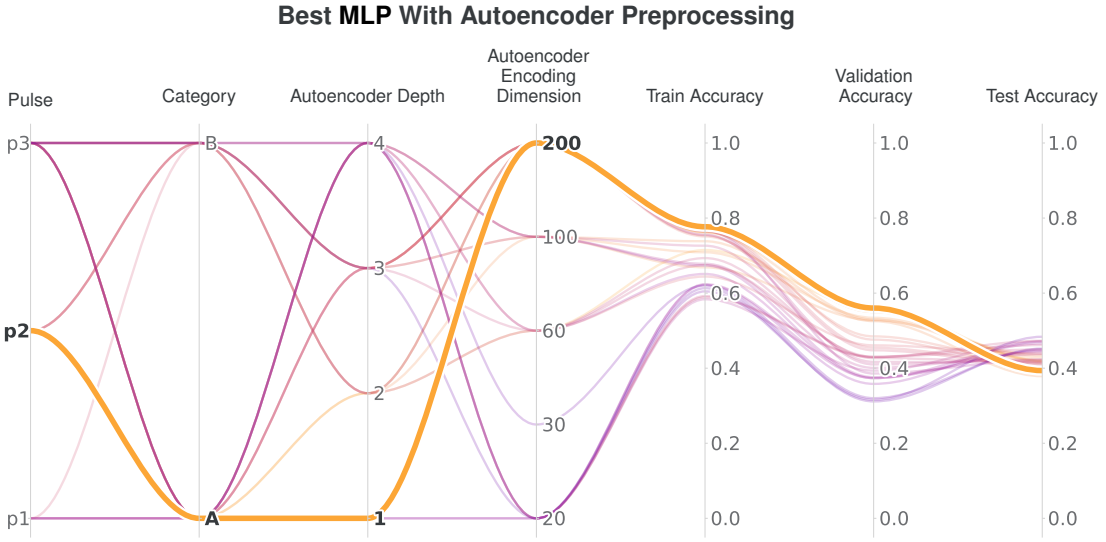
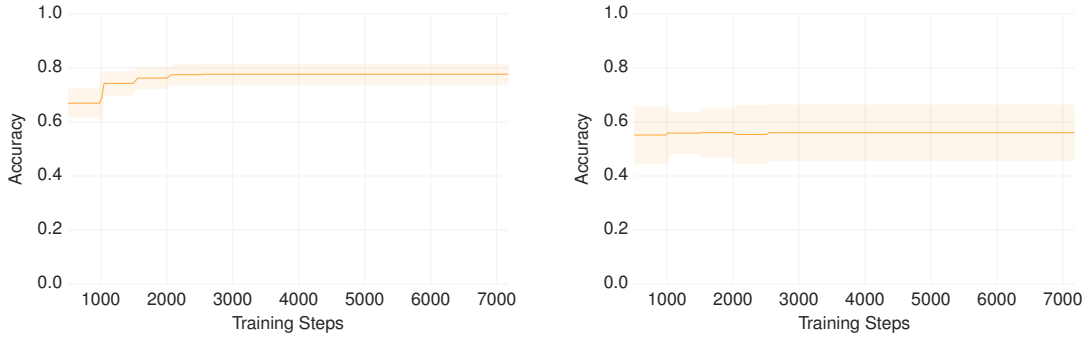


Figure 4.12: Parallel coordinates plot showing sweep results using the best MLP identified in Figure 4.10 with different autoencoder architectures. The thick line corresponds to the best identified Autoencoder hyperparameters and data combination.



(a) Training Accuracy **(b)** Validation Accuracy

Figure 4.13: Training and validation accuracy for the best configuration identified in Figure 4.12 with a 95% confidence interval computed from the cross validation folds.

Table 4.13: The best data and autoencoder hyperparameters when used together with the best MLP.

Data Hyperparameter	**Value**
Category	A
Pulse	2
Autoencoder Hyperparameter	**Value**
Depth	1
Encoding dimension	200
Input size	1600

Table 4.14: Accuracies obtained with the hyperparameters in Table 4.13.

Stage	**Accuracy**
Training	77.68%
Validation	56.02%
Testing	39.36%

4.2.3 Best MLP With Wavelet Preprocessing

The prediction accuracies from a sweep with the best MLP and autoencoder (see Section 4.2.1 and Section 4.2.2) as well as different wavelet transforms is illustrated in Figure 4.14. Four cross-validation folds were used to obtain these results. The best combination of hyperparameters is emphasized with a thicker, orange curve. Figure 4.15 shows the training and validation accuracies over time. Table 4.15 presents the best wavelet transform in combination with the best MLP. The accuracies of the best hyperparameters is shown in Table 4.16.

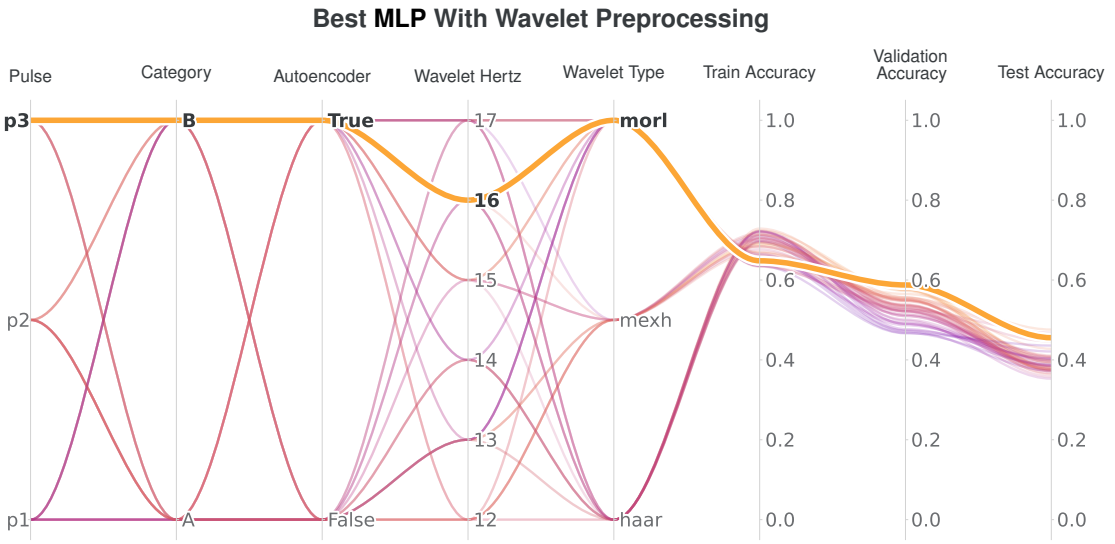
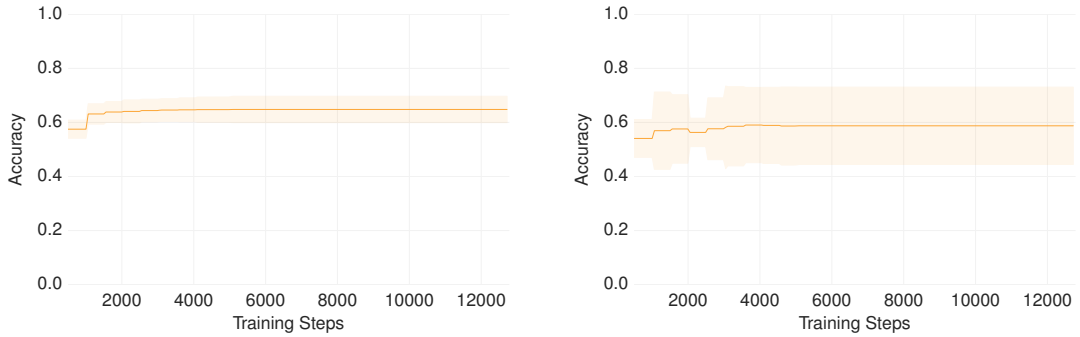


Figure 4.14: Parallel coordinates plot showing sweep results of the best MLP identified in Figure 4.10 and (optionally) the best autoencoder identified in Figure 4.12 with different wavelet preprocessing transforms. The thick line corresponds to the best identified configuration.



(a) Training Accuracy **(b)** Validation Accuracy

Figure 4.15: Training and validation accuracy for the best wavelet configuration identified in Figure 4.14 with a 95% confidence interval computed from the cross validation folds.

Table 4.15: The best data and wavelet transform hyperparameters when used together with the best MLP.

Data Hyperparameter	**Value**
Category	B
Pulse	3
Wavelet Transform	**Value**
Frequency [Hz]	16
Wavelet	Morlet

Table 4.16: Accuracies obtained with the hyperparameters in Table 4.15.

Stage	**Accuracy**
Training	64.81%
Validation	58.76%
Testing	45.56%

4.2.4 Best MLP With Transform Combinations

The prediction accuracies from a sweep with the best MLP and autoencoder (see Section 4.2.1 and Section 4.2.2) as well as different data transforms is illustrated in Figure 4.16. Four cross-validation folds were used to obtain these results. The best combination of hyperparameters is emphasized with a thicker, orange curve. Figure 4.17 shows the training and validation accuracies over time. Table 4.17 presents the best transform combination together with the best MLP. The accuracies of the best hyperparameters is shown in Table 4.18.

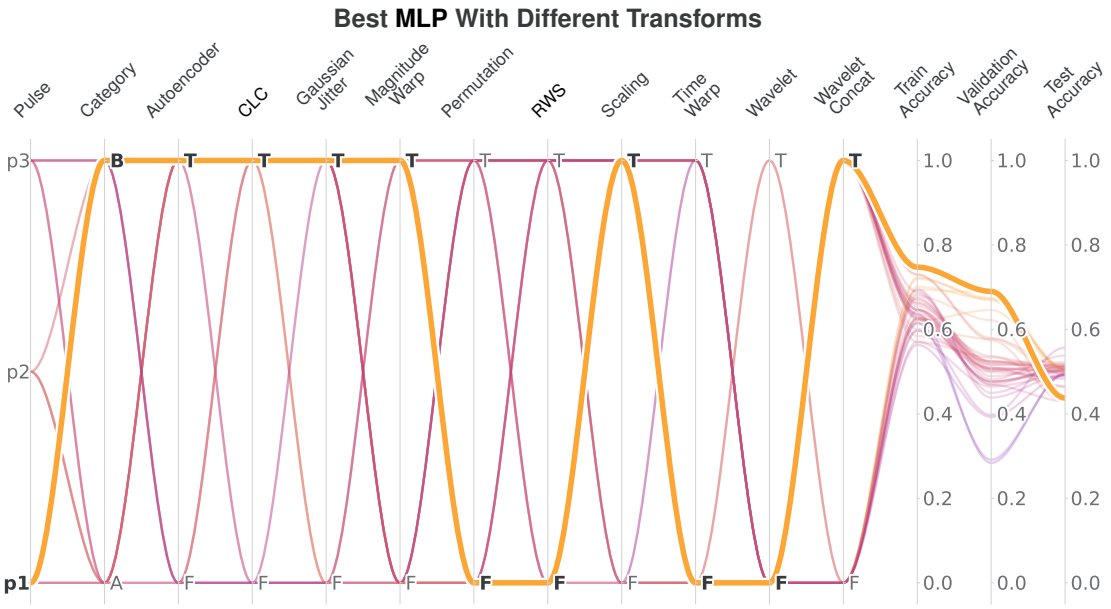
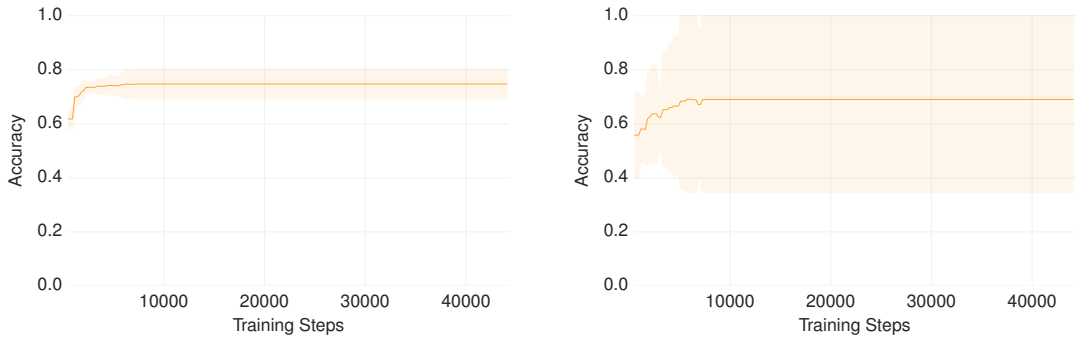


Figure 4.16: Parallel coordinates plot showing sweep results of the best MLP identified in Figure 4.10 with random data transform combinations. The thick line corresponds to the best identified transform combination. "T" and "F" denote "True" and "False" respectively.



(a) Training Accuracy **(b)** Validation Accuracy

Figure 4.17: Training and validation accuracy for the best transform combination identified in Figure 4.16 with a 95% confidence interval computed from the cross validation folds.

Table 4.17: The best data and transform combination hyperparameters when used together with the best MLP. The transforms are listed in the order as they were applied during training/validation/testing.

Data Hyperparameter	**Value**
Category	B
Pulse	1
Autoencoder	**Value**
Depth	1
Encoding dimension	200
Input size	1600
Wavelet Concatenation	**Value**
Wavelet	Haar
Frequencies [Hz]	14, 15
Gaussian Jitter	**Value**
Standard deviation	0.0585
Scaling	**Value**
Standard deviation	0.0890
Magnitude Warp	**Value**
Knots	8
Standard deviation	0.2019
Contrasted Linear Combination (CLC)	**Value**
Contrast	2.4521

Table 4.18: Accuracies obtained with the hyperparameters in Table 4.17.

Stage	**Accuracy**
Training	74.74%
Validation	69.00%
Testing	43.75%

4.3 Convolutional Neural Network (CNN)

This section presents results obtained with CNNs. Specifically, results here were obtained with different architectures, autoencoders, wavelet transforms, and data transform combinations.

4.3.1 Different CNN Architectures

The prediction accuracies from a sweep over CNN hyperparameters is illustrated in Figure 4.18. Four cross-validation folds were used to obtain these results. The best combination of hyperparameters is emphasized with a thicker, orange curve. Figure 4.19 shows the training and validation accuracies over time. Table 4.19 presents the hyperparameters of the best CNN found in this thesis. The accuracies of the best hyperparameters is shown in Table 4.20.

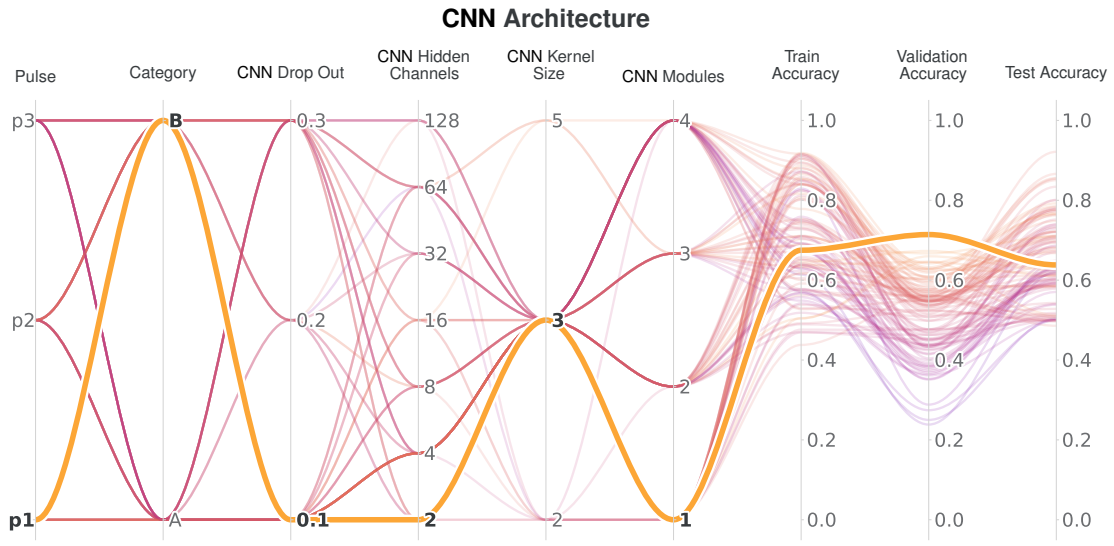
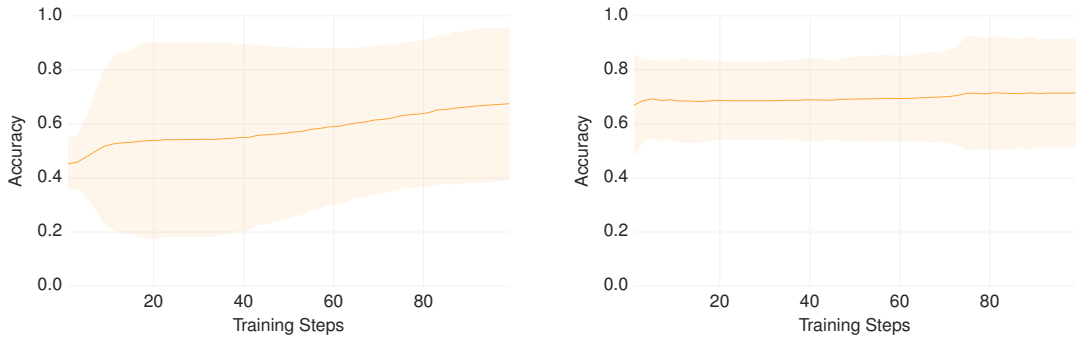


Figure 4.18: Parallel coordinates plot showing the sweep results of CNN architectures. The thick line corresponds to the best identified architecture and data combination.



(a) Training Accuracy **(b)** Validation Accuracy

Figure 4.19: Training and validation accuracy for the best identified CNN model identified in Figure 4.18 with a 95% confidence interval computed from the cross validation folds.

Table 4.19: The best data and CNN hyperparameters.

Data Hyperparameter	**Value**
Category	B
Pulse	1
CNN Hyperparameter	**Value**
Drop out	0.1
Hidden channels	2
In channels	306
Kernel size	3
Module count	1

Table 4.20: Accuracies obtained with the hyperparameters in Table 4.19.

Stage	**Accuracy**
Training	67.39%
Validation	71.42%
Testing	63.81%

4.3.2 Best CNN With Autoencoder Preprocessing

The prediction accuracies from a sweep with the best CNN (see Section 4.3.1) and different autoencoder architectures is illustrated in Figure 4.20. Four cross-validation folds were used to obtain these results. The best combination of hyperparameters is emphasized with a thicker, orange curve. Figure 4.21 shows the training

and validation accuracies over time. Table 4.21 presents the hyperparameters of the best autoencoder in combination with the best CNN. The accuracies of the best hyperparameters is shown in Table 4.22.

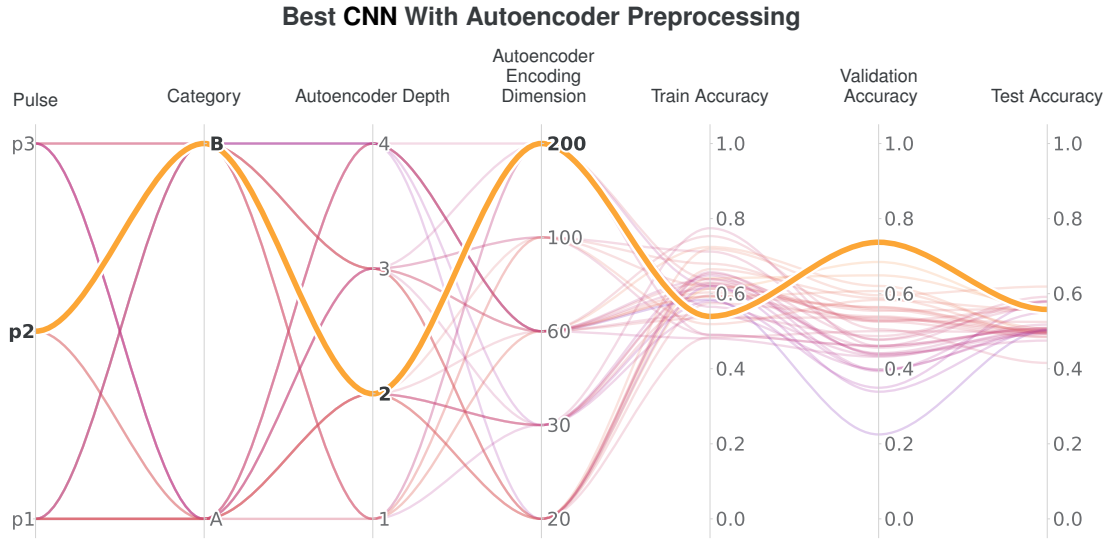
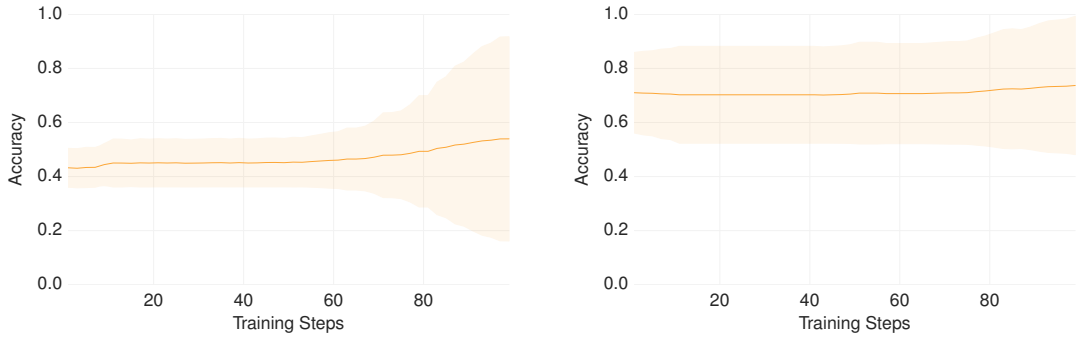


Figure 4.20: Parallel coordinates plot showing sweep results using the best CNN model identified in Figure 4.18 with different autoencoder architectures. The thick line corresponds to the best identified autoencoder hyperparameters and data combination.



(a) Training Accuracy **(b)** Validation Accuracy

Figure 4.21: Training and validation accuracy for the best configuration identified in Figure 4.20 with a 95% confidence interval computed from the cross validation folds.

Table 4.21: The best data and autoencoder hyperparameters when used together with the best CNN.

Data Hyperparameter	**Value**
Category	B
Pulse	2
Autoencoder Hyperparameter	**Value**
Depth	2
Encoding dimension	200
Input size	1600

Table 4.22: Accuracies obtained with the hyperparameters in Table 4.21.

Stage	**Accuracy**
Training	53.95%
Validation	73.70%
Testing	55.81%

4.3.3 Best CNN With Wavelet Preprocessing

The prediction accuracies from a sweep with the best CNN and autoencoder (see Section 4.3.1 and Section 4.3.2) as well as different wavelet transforms is illustrated in Figure 4.22. Four cross-validation folds were used to obtain these results. The best combination of hyperparameters is emphasized with a thicker, orange curve. Figure 4.23 shows the training and validation accuracies over time. Table 4.23 presents the best wavelet transform in combination with the best CNN. The accuracies of the best hyperparameters is shown in Table 4.24.

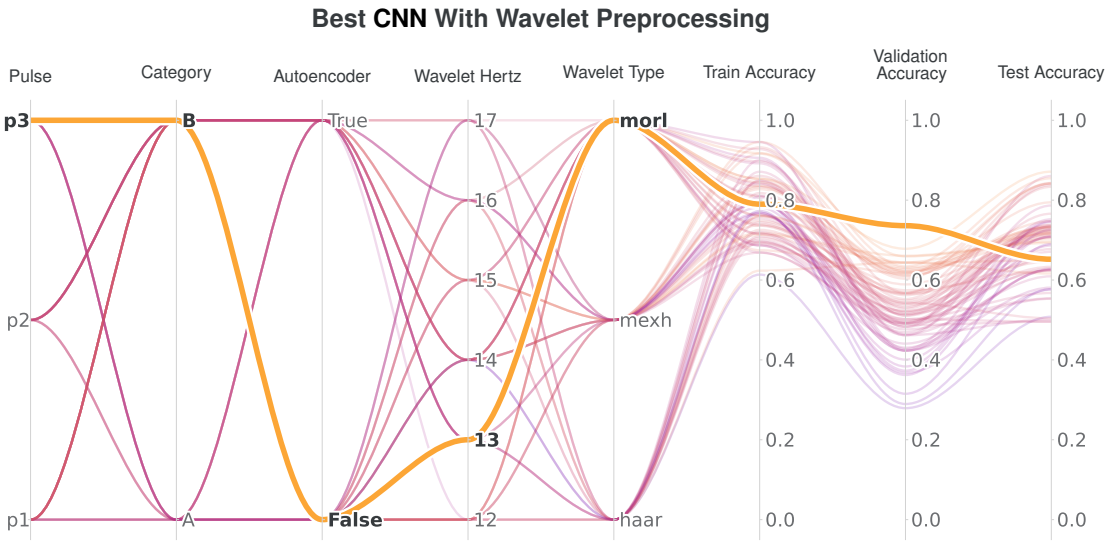
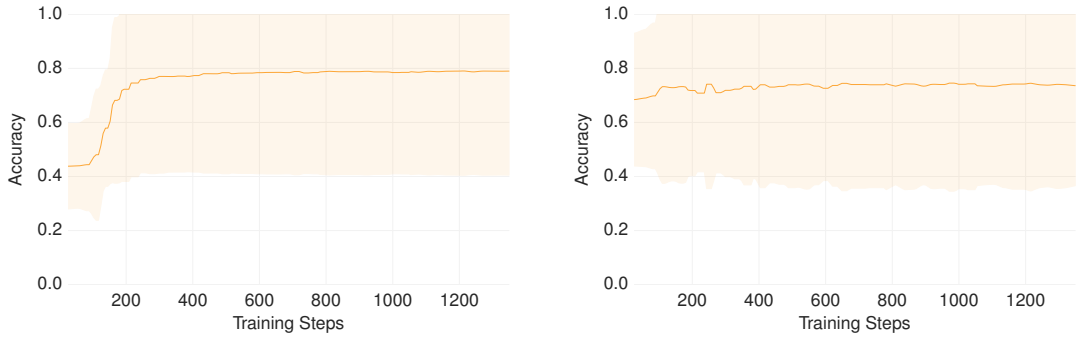


Figure 4.22: Parallel coordinates plot showing sweep results of the best CNN model identified in Figure 4.18 and (optionally) the best autoencoder identified in Figure 4.20 with different wavelet preprocessing transforms. The thick line corresponds to the best identified configuration.



(a) Training Accuracy **(b)** Validation Accuracy

Figure 4.23: Training and validation accuracy for the best wavelet configuration identified in Figure 4.22 with a 95% confidence interval computed from the cross validation folds.

Table 4.23: The best data and wavelet transform hyperparameters when used together with the best CNN.

Data Hyperparameter	**Value**
Category	B
Pulse	3
Wavelet Transform	**Value**
Frequency [Hz]	13
Wavelet	Morlet

Table 4.24: Accuracies obtained with the hyperparameters in Table 4.23.

Stage	**Accuracy**
Training	79.01%
Validation	73.61%
Testing	65.21%

4.3.4 Best CNN With Transform Combinations

The prediction accuracies from a sweep with the best CNN and autoencoder (see Section 4.3.1 and Section 4.3.2) as well as different data transforms is illustrated in Figure 4.24. Four cross-validation folds were used to obtain these results. The best combination of hyperparameters is emphasized with a thicker, orange curve. Figure 4.25 shows the training and validation accuracies over time. Table 4.25 presents the best transform combination together with the best CNN. The accuracies of the best hyperparameters is shown in Table 4.26.

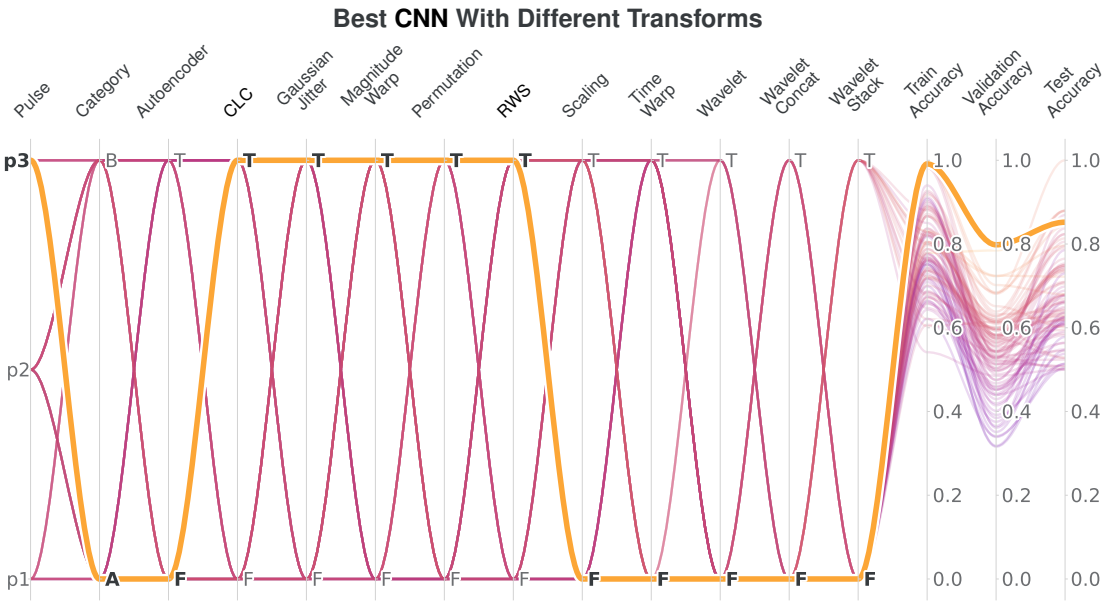
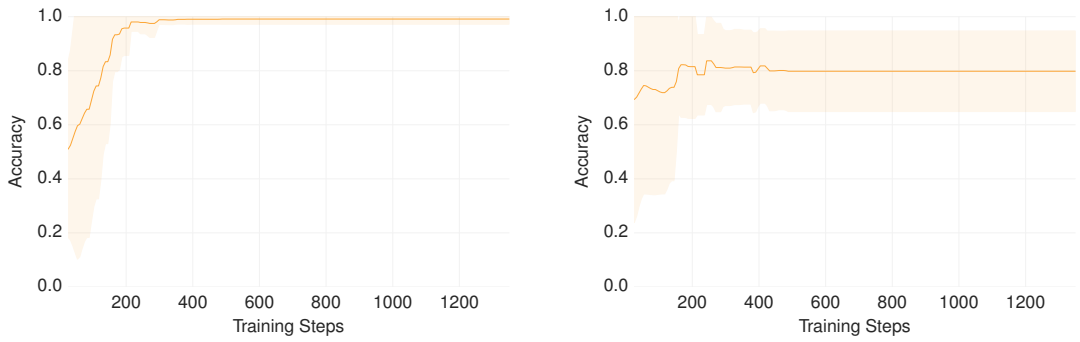


Figure 4.24: Parallel coordinates plot showing sweep results of the best CNN model identified in Figure 4.18 with random data transform combinations. The thick line corresponds to the best identified transform combination. "T" and "F" denote "True" and "False" respectively.



(a) Training Accuracy **(b)** Validation Accuracy

Figure 4.25: Training and validation accuracy for the best transform combination identified in Figure 4.24 with a 95% confidence interval computed from the cross validation folds.

Table 4.25: The best data and transform combination hyperparameters when used together with the best CNN. The transforms are listed in the order as they were applied during training/validation/testing.

Data Hyperparameter	**Value**
Category	A
Pulse	3
Gaussian Jitter	**Value**
Standard deviation	0.4697
Permutation	**Value**
Slice Count	6
Magnitude Warp	**Value**
Knots	4
Standard deviation	0.6937
Contrasted Linear Combination (CLC)	**Value**
Contrast	0.9401
Random Window Slicing (RWS)	**Value**
Window length	800

Table 4.26: Accuracies obtained with the hyperparameters in Table 4.25, averaged over four cross-validation folds.

Stage	**Accuracy**
Training	99.13%
Validation	79.82%
Testing	85.21%

Because of the relatively high accuracies showed in Table 4.26, obtained with the hyperparameters in Table 4.25, the corresponding model and transforms were trained and tested 132 more times. There is high variance in the accuracies between those runs and this visualized in the violin plots in Figure 4.26.

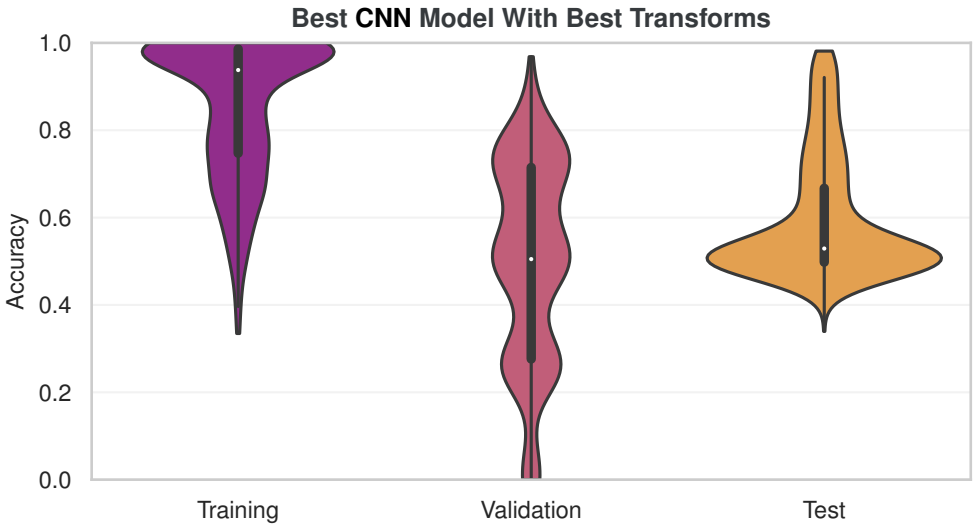


Figure 4.26: Violin plot showing training, validation, and test accuracy distributions over all four cross-validation folds in all 132 repetition runs of the best CNN and the best transform combination seen in Figure 4.24.

4.4 Transformer

This section presents results obtained with transformers. Specifically, results here were obtained with different architectures, autoencoders, wavelet transforms, and data transform combinations.

4.4.1 Different Transformer Architectures

The prediction accuracies from a sweep over transformer hyperparameters is illustrated in Figure 4.27. Four cross-validation folds were used to obtain these results. The best combination of hyperparameters is emphasized with a thicker, orange curve. Figure 4.28 shows the training and validation accuracies over time. Table 4.27 presents the hyperparameters of the best transformer found in this thesis. The accuracies of the best hyperparameters is shown in Table 4.28.

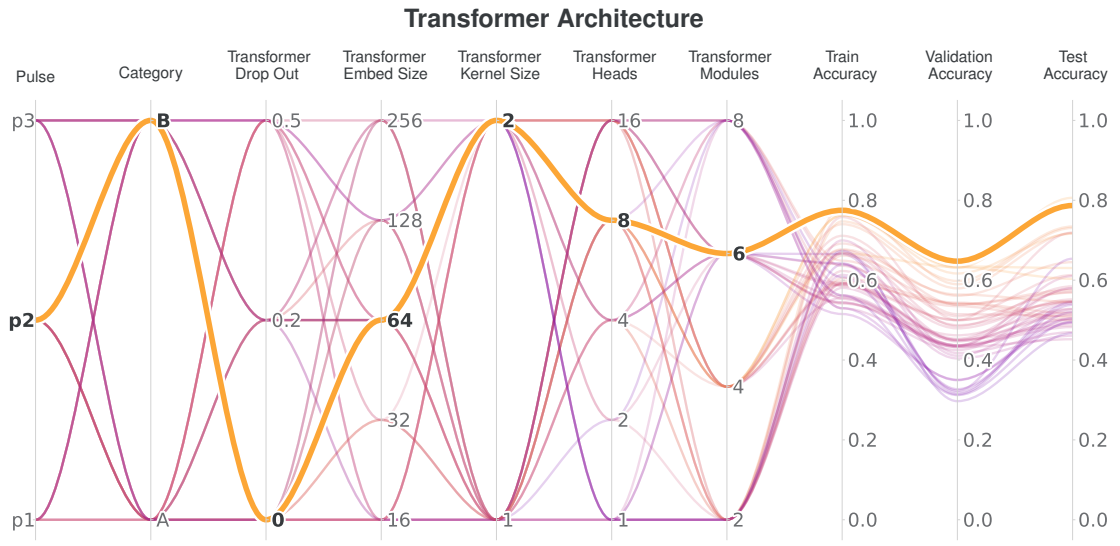
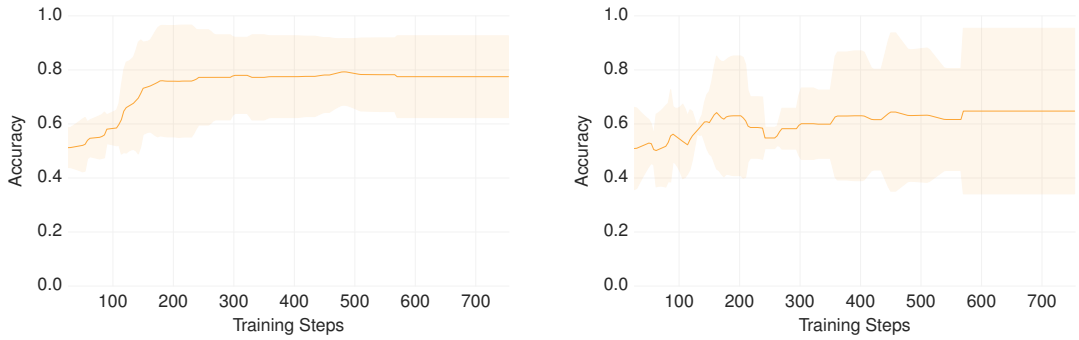


Figure 4.27: Parallel coordinates plot showing the sweep results of transformer architectures. The thick line corresponds to the best identified architecture and data combination.



(a) Training Accuracy **(b)** Validation Accuracy

Figure 4.28: Training and validation accuracy for the best identified transformer model identified in Figure 4.27 with a 95% confidence interval computed from the cross validation folds.

Table 4.27: The best data and transformer hyperparameters.

Data Hyperparameter	**Value**
Category	B
Pulse	2
Transformer Hyperparameter	**Value**
Attention head count	8
Drop out	0
Embed size	64
In channels	1600
Kernel size	2
Module count	6

Table 4.28: Accuracies obtained with the hyperparameters in Table 4.27.

Stage	**Accuracy**
Training	77.50%
Validation	64.76%
Testing	78.68%

4.4.2 Best Transformer With Autoencoder Preprocessing

The prediction accuracies from a sweep with the best transformer (see Section 4.4.1) and different autoencoder architectures is illustrated in Figure 4.29. Four cross-

validation folds were used to obtain these results. The best combination of hyperparameters is emphasized with a thicker, orange curve. Figure 4.30 shows the training and validation accuracies over time. Table 4.29 presents the hyperparameters of the best autoencoder in combination with the best transformer. The accuracies of the best hyperparameters is shown in Table 4.30.

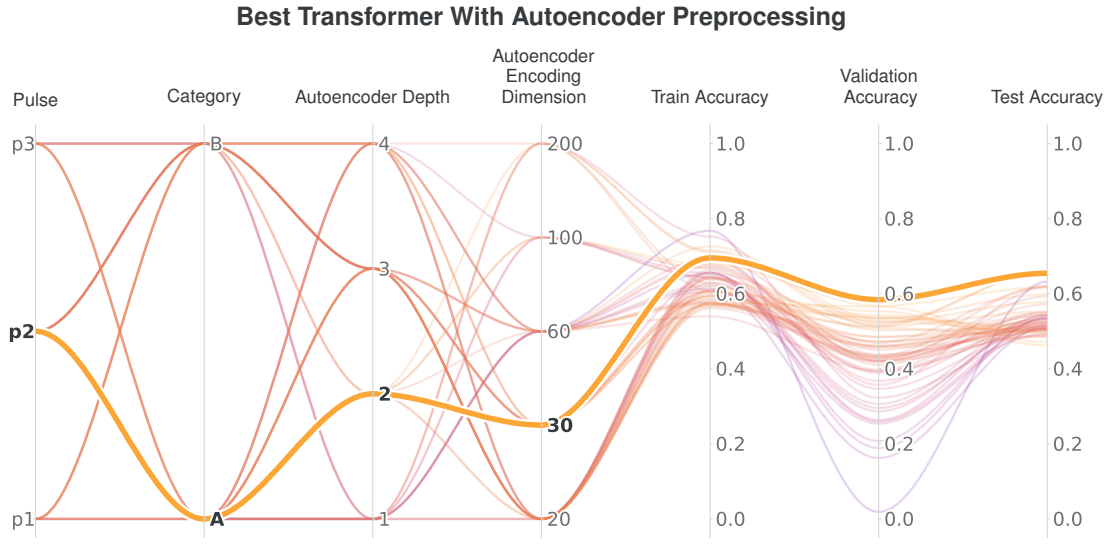
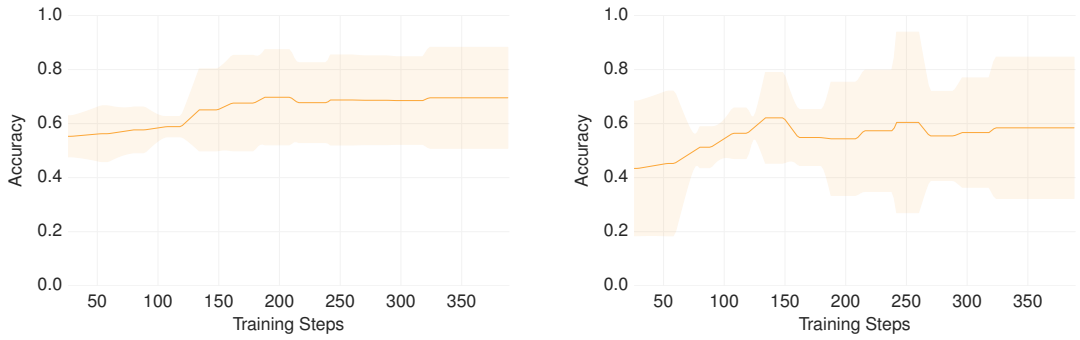


Figure 4.29: Parallel coordinates plot showing sweep results using the best transformer model identified in Figure 4.27 with different autoencoder architectures. The thick line corresponds to the best identified autoencoder hyperparameters and data combination.



(a) Training Accuracy **(b)** Validation Accuracy

Figure 4.30: Training and validation accuracy for the best configuration identified in Figure 4.29 with a 95% confidence interval computed from the cross validation folds.

Table 4.29: The best data and autoencoder hyperparameters when used together with the best transformer.

Data Hyperparameter	**Value**
Category	A
Pulse	2
Autoencoder Hyperparameter	**Value**
Depth	2
Encoding size	30
Input size	1600

Table 4.30: Accuracies obtained with the hyperparameters in Table 4.29.

Stage	**Accuracy**
Training	69.54%
Validation	58.42%
Testing	65.45%

4.4.3 Best Transformer With Wavelet Preprocessing

The prediction accuracies from a sweep with the best transformer and autoencoder (see Section 4.4.1 and Section 4.4.2) as well as different wavelet transforms is illustrated in Figure 4.31. Four cross-validation folds were used to obtain these results. The best combination of hyperparameters is emphasized with a thicker, orange curve. Figure 4.32 shows the training and validation accuracies over time. Table 4.31 presents the best wavelet transform in combination with the best transformer. The accuracies of the best hyperparameters is shown in Table 4.32.

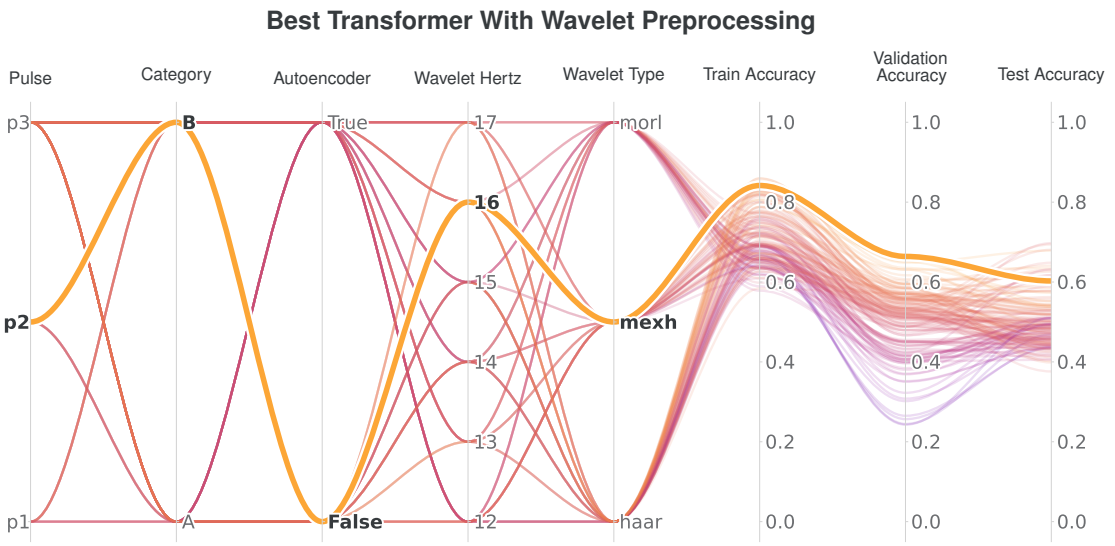
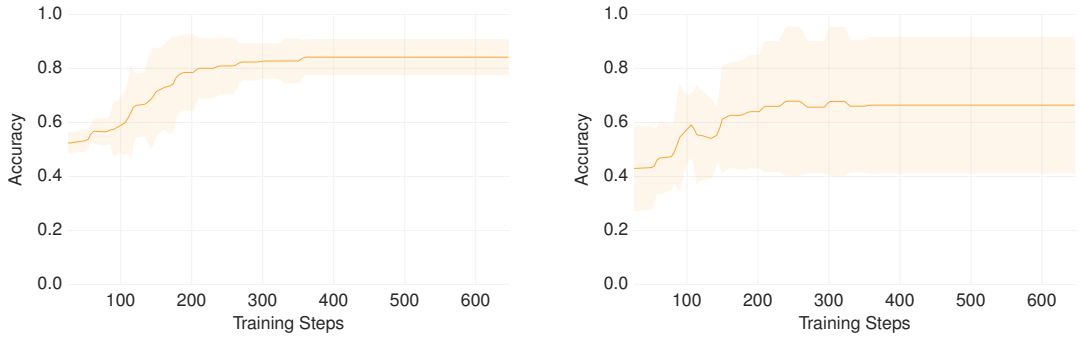


Figure 4.31: Parallel coordinates plot showing sweep results of the best transformer model identified in Figure 4.27 and (optionally) the best autoencoder identified in Figure 4.29 with different wavelet preprocessing transforms. The thick line corresponds to the best identified configuration.



(a) Training Accuracy

(b) Validation Accuracy

Figure 4.32: Training and validation accuracy for the best wavelet configuration identified in Figure 4.31 with a 95% confidence interval computed from the cross validation folds.

Table 4.31: The best data and wavelet transform hyperparameters when used together with the best transformer.

Data Hyperparameter	**Value**
Category	B
Pulse	2
Wavelet Transform	**Value**
Frequency [Hz]	16
Wavelet	Mexican Hat

Table 4.32: Accuracies obtained with the hyperparameters in Table 4.31.

Stage	**Accuracy**
Training	84.17%
Validation	66.41%
Testing	60.27%

4.4.4 Best Transformer With Transform Combinations

The prediction accuracies from a sweep with the best transformer and autoencoder (see Section 4.4.1 and Section 4.4.2) as well as different data transforms is illustrated in Figure 4.33. Four cross-validation folds were used to obtain these results. The best combination of hyperparameters is emphasized with a thicker, orange curve. Figure 4.34 shows the training and validation accuracies over time. Table 4.33 presents the best transform combination together with the best transformer. The accuracies of the best hyperparameters is shown in Table 4.34.

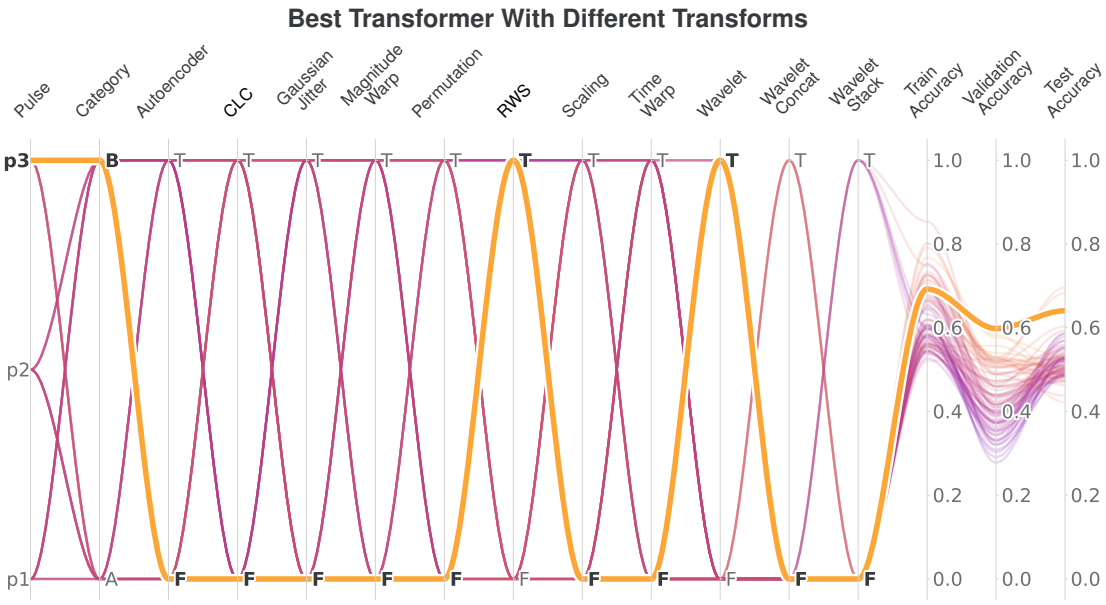
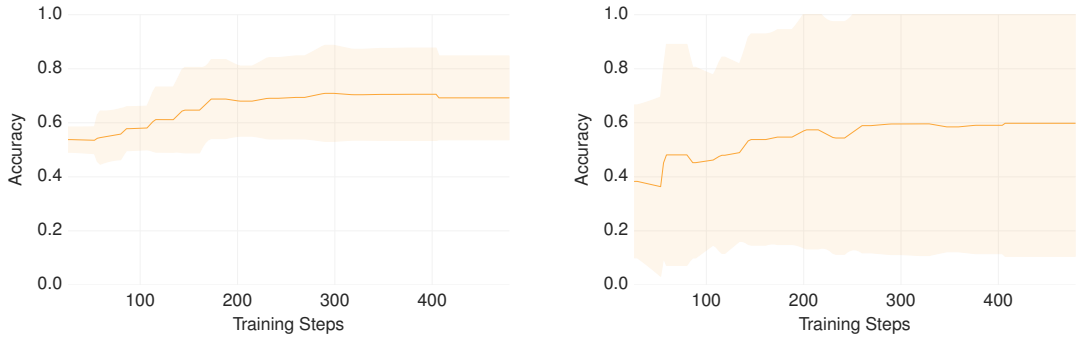


Figure 4.33: Parallel coordinates plot showing sweep results of the best transformer model identified in Figure 4.27 with random data transform combinations. The thick line corresponds to the best identified transform combination. "T" and "F" denote "True" and "False" respectively.



(a) Training Accuracy **(b)** Validation Accuracy

Figure 4.34: Training and validation accuracy for the best transform combination identified in Figure 4.33 with a 95% confidence interval computed from the cross validation folds.

Table 4.33: The best data and transform combination hyperparameters when used together with the best transformer. The transforms are listed in the order as they were applied during training/validation/testing.

Data Hyperparameter	**Value**
Category	B
Pulse	3
Wavelet Transform	**Value**
Frequency [Hz]	15
Wavelet	Mexican Hat
Random Window Slicing (RWS)	**Value**
Window length	800

Table 4.34: Accuracies obtained with the hyperparameters in Table 4.33.

Stage	**Accuracy**
Training	69.23%
Validation	59.82%
Testing	64.06%

4.5 Summary

Table 4.35 shows a summary of all obtained prediction accuracies in the different hyperparameter sweeps from the previous sections of this chapter.

Table 4.35: Summary of the highest obtained prediction accuracies in each performed sweep. The highest accuracies are written in bold.

ILSTM Sweep	**Training**	**Validation**	**Test**
2020 Reproduction (LOOCV)	84.30%	55.40%	36.65%
Architecture	78.29%	58.01%	38.56%
Best With Autoencoder Preprocessing	76.36%	60.15%	38.12%
Best With Wavelet Preprocessing	72.01%	59.74%	36.35%
Best With Different Transforms	84.88%	**82.29%**	49.67%
MLP Sweep	**Training**	**Validation**	**Test**
Architecture	76.53%	58.71%	41.44%
Best With Autoencoder Preprocessing	77.68%	56.02%	39.36%
Best With Wavelet Preprocessing	64.81%	58.76%	45.56%
Best With Different Transforms	74.74%	69.00%	43.75%
CNN Sweep	**Training**	**Validation**	**Test**
Architecture	67.39%	71.42%	63.81%
Best With Autoencoder Preprocessing	53.95%	73.70%	55.81%
Best With Wavelet Preprocessing	79.01%	73.61%	65.21%
Best With Different Transforms	**99.13%**	79.82%	**85.21%**
Transformer Sweep	**Training**	**Validation**	**Test**
Architecture	77.50%	64.76%	78.68%
Best With Autoencoder Preprocessing	69.54%	58.42%	65.45%
Best With Wavelet Preprocessing	84.17%	66.41%	60.27%
Best With Different Transforms	69.23%	59.82%	64.06%

5

Discussion

This chapter discusses and compares the results presented in Chapter 4 as well as the limitations that this project faced, future work, and ethical and societal aspects related to the subject.

5.1 Comparing Results

This section discusses the results presented in the previous chapter, possible interpretations and inferences.

5.1.1 Univariate vs. Multivariate Models

Very broadly speaking, there is a high-level distinction between the ILSTM/MLP runs versus the CNN/Transformer runs, in that the former used univariate data while the latter used multivariate data. In other words, the former models were tasked with classifying MSNA inhibition based on the readings from single MEG sensors, while the latter models could classify the patient based on all the sensor readings for a given pulse.

Intuitively, one would expect the multivariate models to perform better, because they always get access to the relevant information (assuming at least one sensor can pick up information about MSNA inhibition). The univariate models, on the other hand, may have to classify signals from regions where no MSNA information is present, which forces it to overfit to noise or resort to random guessing.

Notably, in the univariate case, the test accuracy was consistently below the performance of random guessing, while most multivariate models at least made predictions that correlated positively with the true labels in the test set.

5.1.2 Confidence, Variance, and Words of Caution

Most of the accuracies reported in Chapter 4 come from an averaging of four cross-validation folds. This is better than just single runs, but does not remove all variance from the results. Furthermore, in the multivariate case the dataset is effectively reduced by a factor of 306, which means that the variance of the accuracy estimates should be 306 times larger.

A rough upper-bound for how spread-out the accuracy distribution is can be seen by

looking at the accuracies of all the runs in the parallel coordinate plots and thinking of them as being samples from a single distribution. In the multivariate case this distribution is very wide.

In practice, the hyperparameters do likely have *some* effect, and the conditional distribution of accuracies for a given hyperparameter selection should have a somewhat lower variance, but exactly how much is difficult to tell without running many additional runs.

Another source for variance in this thesis is that the weights of the models are initialized differently in each cross-validation fold and run. The hyperparameters are thus not the only things that vary.

5.1.3 Model Architecture

The performance was comparable for both univariate models. Neither of them learned anything useful. The training accuracy is a bit higher, while the test accuracy is a bit lower for the ILSTM compared to the MLP. But since the test accuracy is below random chance, not much can be concluded from this.

Between the multivariate Transformer and CNN, the CNN generally got better performance across all metrics. A possible explanation for this could be that sensor index is important, and an architecture that prevents the model from distinguishing between these indices is detrimental to performance.

In general, it is difficult to draw any conclusions about which architecture hyperparameters are the most beneficial due to the high accuracy variance.

5.1.4 Preprocessing and Augmentation Methods

The best models identified typically included some augmentation methods. However, it is not clear whether the performance difference is due to the augmentation, or random factors such as weight initialization and choice of validation patients in the cross-validation folds.

In order to draw stronger conclusions about the effect of a specific hyperparameter or transform, the same architecture could be trained several times, where each run has a different random seed. This would make the weight initialization constant for each cross-validation fold in any given run but make it different in other runs. The average accuracy over all such runs would better represent the general effect of the hyperparameter or transform in question. Doing this was however not feasible in this project because of the given time-frame.

With the mentioned sources for high variance in mind, the rest of this section will discuss the differences in results with regard to different preprocessing and augmentation methods.

Using autoencoders for preprocessing generally decreased accuracies for all models, with some exceptions where there only was a slight increase. It therefore does not seem that beneficial to use an autoencoder in this context.

Wavelet transforms were also used with the different models and their effect on the accuracies varied quite a lot. Any increase in accuracy was not large, making it seem like wavelets are not that useful either.

Random transform combinations, including both preprocessing and augmentation techniques, were also used with the different models. The results from these sweeps are what generally stand out since the difference in accuracy between models on their own and together with transforms are found here. Notably, the best CNNs together with certain transforms got 99.13% in training accuracy, 79.82% in validation accuracy, and 85.21% in test accuracy (see Section 4.3.4). This is a huge change from using the model in question on its own. It is however worth mentioning the high variance again, which is visualized in Figure 4.26.

5.2 Limitations

Even though there are a lot of results in this thesis, there could have been even more. The computer used for training models was a limiting factor in how many results could be obtained, especially since it was shared with another Master's thesis project. The computer stopped working several times and the competition for its resources caused sweeps to sometimes crash. A training schedule for the projects was set up but only after some time had been lost.

The available computing resources was not the only time limiting factor however. The framework developed for running sweeps worked very well towards the end but left relatively little time to actually run sweeps. All results in this thesis were obtained in the last month of the project. Although it took long to develop, the framework did make it easy to run new things however. It made it worth the effort since it made it possible to quickly run new things that were thought of. Due to the human factor however, some mistakes regarding for example sweep configurations and simple bugs in the framework resulted in some sweeps having to be restarted. It is much to ask to implement everything perfectly the first time but had there been fewer bugs in the beginning of gathering results, there would have been more to go on.

Due to privacy concerns since the dataset contains medical data, the dataset could not be transferred onto more high-performance data solutions. The data was therefore always kept on a slow hard drive which was connected directly to the computer used for training. Allocating space for the data in RAM memory allowed for relatively fast computations but the allocation itself was very slow. This was especially noticed during cross-validation when the whole dataset had to be rotated and reallocated.

5.3 Future Work

Given the results in this project, there is clearly more work remaining to be done before a system like this can be deployed in a medical setting.

Achieving good accuracy on the validation and test sets would be a necessary, but not sufficient, requirement. This was not achieved in this project. Despite various attempts at augmenting the data and exploring different methods of utilizing domain knowledge, getting around the small dataset size proved to be too difficult. Especially when applying deep learning techniques, which are known to be data hungry.

It would be advisable for continuations of this work to either focus on getting more real data, or on minimizing the use of deep learning in the classification pipeline, and relying more on the statistical methods as described in [3].

An ideal scenario would be if MedTech West, perhaps in combination with other organizations across the globe, made a concerted effort to gather significantly more data. With a data set on the order of 1000 people, it is much more likely that a practically useful and generally applicable model could be developed.

Barring this possibility, it could be worth investigating other general large MEG datasets. Even if they do not come from the same type of experiment, a large dataset of MEG data could be useful for semi-supervised learning. A model could learn the general structure of MEG from large datasets and utilize this knowledge for downstream tasks such as classifying MSNA inhibition.

Besides this, there is more data available from the experiments on which this project was based than just the MEG data. For instance, there is EEG data available from around 30 people, and there is more "idle" MEG data available from where the patients were not exposed to any stimuli.

Although deep learning is enticing in its powerful ability to extract patterns from large amounts of data, it is often inappropriate for smaller datasets. Without access to data from more people, deep learning might not be the right tool for the job. Processing the data similarly to Bushra et al. [3], computing some statistics from the regions with known correlations, and running basic classification methods on this low-dimensional data may prove much more interpretable, robust, and successful.

5.4 Ethical and Societal Aspects

This section discusses different ethical and societal aspects that have been taken into consideration during this project, as well as some discussion of the broader impact of this work.

5.4.1 Privacy

This project has involved working with medical data collected from a set of people. Personal data in general, and in particular medical data should for many reasons be kept private. This is a widely recognized fact covered by international laws such as GDPR [53].

To ensure that the privacy of the individuals involved in this project is respected, the data has been anonymized so that it is not possible to identify them from it. Additionally, the data has at all times been kept locally at Syntronic and not been exposed to any external party.

5.4.2 Confidence and Interpretability

If the results of this project were more promising, and time had allowed, it would have been valuable to investigate methods for interpretability of these classification models.

In medical settings, black-box classifiers are typically not ideal. The doctor might want a reasonable explanation for why the answer came out as it did. Good interpretability would give both clinicians and patients more confidence in the model predictions.

Another avenue that may be valuable to explore is the creation of confidence intervals for individual predictions, which could be a way for the doctor to gauge how confident the classifier is in its output.

5.4.3 Sustainability

The United Nations has 17 goals for sustainable development (SDGs) [54]. The goals for Good Health and Well-Being and Gender Equality are of particular relevance to this project.

Regarding good health and well-being, this project is part of an effort to understand and prevent essential hypertension and cardiovascular disease, which could lead to people living longer, healthier lives. With a cynical point of view, this would have a negative impact on the Climate Action SDG since it would result in a larger ecological footprint.

Regarding gender equality, or bias in general, it is too early to be concerned about the biases of the models developed in this project. The reason for this is that the models are unable to reliably classify MSNA inhibition. With that said, bias will be important to keep in consideration in future iterations of this work. In medical

situations, it is especially important to be aware of potential biases in the training dataset. Ideally, the final system should perform equally well for everyone regardless of age, ethnicity, and gender. It is therefore crucial that future models are trained on large diverse datasets before being deployed for real-world use.

6

Conclusion

The goal of this Master's thesis was to improve the classification accuracy of MSNA inhibition from MEG data beyond what was achieved in the two previous theses on the subject [15], [16]. This was attempted through fixing some methodological issues of these previous theses, as well as exploring various new ANN architectures, and data preprocessing and augmentation methods.

The main methodological issue that was fixed is the data split between training and validation patients. This made the validation accuracy more aligned with the test accuracy, and hyperparameter optimization based on the validation accuracy became viable.

The main contribution of this thesis was a broad search of architectures, hyperparameters, and data preprocessing and augmentation methods and an analysis of their relative performance.

The main finding from this search is that multivariate models which consider all MEG sensors have more potential to achieve high accuracy than models that classify on the individual sensor level. Even these, however, fail to reliably solve the task on this dataset. The dataset is small in terms of the number of patients, resulting in unreliable high-variance results.

The choice of data preprocessing and augmentation methods had a minor impact on results, but no clear patterns were identified. The best performing model was a CNN with a collection of augmentation methods applied (CLC, Gaussian Jitter, Magnitude Warping, Permutation, and RWS). However, this performance seems to be relatively sensitive to weight initialization, and reproducing the same model does not guarantee similar results.

The main conclusion from this work is that the size of the dataset remains the most significant bottleneck for good performance on this task. Without more data, applying deep learning to the problem is likely to be unsuccessful and the most promising step forward might be to use more traditional statistical approaches or semi-supervised learning techniques.

Despite high variances, this thesis lays some groundwork for MedTech West's future research on preventative measures for essential hypertension and cardiovascular disease.

Bibliography

[1] World Health Organization. "Cardiovascular diseases." (2021), [Online]. Available: https://www.who.int/health-topics/cardiovascular-diseases (visited on 12/10/2021).

[2] S. E. Kjeldsen, "Hypertension and cardiovascular risk: General aspects," *Pharmacological research*, vol. 129, pp. 95–99, Mar. 2018, ISSN: 1096-1186. DOI: 10.1016/J.PHRS.2017.11.003. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/29127059/.

[3] B. R. Syeda, "Bringing meg towards clinical applications," 2018. [Online]. Available: http://hdl.handle.net/2077/56334.

[4] J. Gordon Betts, P. Desaix, E. Johnson, *et al.*, *Anatomy & physiology*. Rice University, 2013, ISBN: 9781938168130.

[5] R. Kivi. "Just the essentials of essential hypertension." (Sep. 17, 2018), [Online]. Available: https://www.healthline.com/health/high-blood-pressure-hypertension/how-reduce-high-blood-pressure#causes (visited on 05/25/2022).

[6] M. N. Alshak and J. M Das, *Neuroanatomy, Sympathetic Nervous System*. StatPearls Publishing, Treasure Island (FL), 2021. [Online]. Available: http://europepmc.org/books/NBK542195.

[7] T. Mano, "Muscle sympathetic nerve activity in blood pressure control against gravitational stress," *Journal of Cardiovascular Pharmacology*, vol. 38, S7–S11, Oct. 2001, ISSN: 0160-2446. DOI: 10.1097/00005344-200110001-00003.

[8] M. J. Joyner, N. Charkoudian, and B. G. Wallin, "Sympathetic nervous system and blood pressure in humans," *Hypertension*, vol. 56, pp. 10–16, 1 Jul. 2010, ISSN: 0194-911X. DOI: 10.1161/HYPERTENSIONAHA.109.140186.

[9] S. Supek and C. J. Aine, *Magnetoencephalography*. Springer, 2016.

[10] Meg Scanner Unit, Univeristy of Oxford, Nightingale Associates. Photograph. Britannica ImageQuest, Encyclopædia Britannica, 25 May 2016. [Online]. Available: https://quest-eb-com.eu1.proxy.openathens.net/search/104_324042/1/104_324042/cite (visited on 04/21/2022).

[11] "Transformation. The positions of the head, skull, brain, and helmet sensors after the transformation." Andersen LM (2018) Group Analysis in MNE-Python of Evoked Responses from a Tactile Stimulation Paradigm: A Pipeline for Reproducibility at Every Step of Processing, Going from Individual Sensor Space Representations to an across-Group Source Space Representation. Front. Neurosci. 12:6. Distributed under the terms of the Creative Commons Attribution

License (CC BY)., Jan. 22, 2018. DOI: 10.3389/fnins.2018.00006. (visited on 04/21/2022).

[12] C. Amrutkar and R. M. Riel-Romero, "Rolandic epilepsy seizure," *StatPearls*, Aug. 2021. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK534845/.

[13] "Brain Lateral View" by Chiara Mazzasette, licensed by CC BY 4.0), May 18, 2021. [Online]. Available: https://med.libretexts.org/Bookshelves/Anatomy_and_Physiology/Human_Anatomy_(OERI)/12%3A_Central_and_Peripheral_Nervous_System/12.03%3A_Brain-_Cerebrum (visited on 04/21/2022).

[14] S. H. Park, J. Choi, and J.-S. Byeon, "Key principles of clinical validation, device approval, and insurance coverage decisions of artificial intelligence," *Korean Journal of Radiology*, vol. 22, p. 442, 3 2021, ISSN: 1229-6929. DOI: 10.3348/kjr.2021.0048.

[15] A. Bakidou and J. N. Odhner, "Deep learning for brain activity analysis," 2020. [Online]. Available: https://hdl.handle.net/20.500.12380/301297.

[16] C. Chau and E. Nordanger, "Guiding AI-based classification: Can conventional functional neuroimaging analysis improve deep learning methods for identifying risk for essential hypertension?," 2021. [Online]. Available: https://hdl.handle.net/20.500.12380/304228.

[17] B. Riaz, J. J. Eskelin, L. C. Lundblad, *et al.*, "Brain structural and functional correlates to defense-related inhibition of muscle sympathetic nerve activity in man," *Scientific Reports 2022 12:1*, vol. 12, pp. 1–13, 1 Feb. 2022, ISSN: 2045-2322. DOI: 10.1038/s41598-022-05910-8. [Online]. Available: https://www.nature.com/articles/s41598-022-05910-8.

[18] R. Tavenard, J. Faouzi, G. Vandewiele, *et al.*, "Tslearn, a machine learning toolkit for time series data," *Journal of Machine Learning Research*, vol. 21, no. 118, pp. 1–6, 2020. [Online]. Available: http://jmlr.org/papers/v21/20-091.html.

[19] M. Barandas, D. Folgado, L. Fernandes, *et al.*, "Tsfel: Time series feature extraction library," *SoftwareX*, vol. 11, p. 100 456, 2020, ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2020.100456. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711020300017.

[20] M. X. Cohen, *Analyzing Neural Time Series Data: Theory and Practice*, illustrated. MIT Press, 2014, 0-600, ISBN: 026231956X, 9780262319560.

[21] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: A review," *Data Mining and Knowledge Discovery*, vol. 33, pp. 917–963, 4 Jul. 2019, ISSN: 1384-5810. DOI: 10.1007/s10618-019-00619-1. [Online]. Available: http://link.springer.com/10.1007/s10618-019-00619-1.

[22] B. Mehlig, "Machine learning with neural networks," *Machine Learning with Neural Networks*, Jan. 2019. DOI: 10.1017/9781108860604. [Online]. Available: http://arxiv.org/abs/1901.05639%20http://dx.doi.org/10.1017/9781108860604.

[23] A. Subasi, "Eeg signal classification using wavelet feature extraction and a mixture of expert model," *Expert Systems with Applications*, vol. 32, pp. 1084–1093, 4 May 2007, ISSN: 0957-4174. DOI: 10.1016/J.ESWA.2006.02.005.

[24] D. B. Percival and A. T. Walden, *Wavelet methods for time series analysis*. Cambridge university press, 2000, vol. 4.

[25] A. N. Akansu and R. A. Haddad, *Multiresolution Signal Decomposition: Transforms, Subbands, and Wavelets*. 2001, p. 499, ISBN: 9780120471416. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780120471416500070.

[26] T. P. Developers. "Continuous wavelet transform (cwt)." (Mar. 11, 2022), [Online]. Available: https://pywavelets.readthedocs.io/en/latest/ref/cwt.html (visited on 04/11/2022).

[27] M. X. Cohen, "A better way to define and describe morlet wavelets for time-frequency analysis," Aug. 21, 2018. DOI: 10.1101/397182. [Online]. Available: https://doi.org/10.1101/397182.

[28] W. K. Ngui, M. S. Leong, L. M. Hee, and A. M. Abdelrhman, "Wavelet analysis: Mother wavelet selection methods," *Applied Mechanics and Materials*, vol. 393, pp. 953–958, 2013, ISSN: 16609336. DOI: 10.4028/WWW.SCIENTIFIC.NET/AMM.393.953.

[29] J. K. Sunkara, "Selection of wavelet basis function for image compression – a review," *ELCVIA Electronic Letters on Computer Vision and Image Analysis*, vol. 18, pp. 1–20, 1 Sep. 2019, ISSN: 1577-5097. DOI: 10.5565/rev/elcvia.1094. [Online]. Available: https://elcvia.cvc.uab.cat/article/view/1094.

[30] G. Forestier, F. Petitjean, H. A. Dau, G. I. Webb, and E. Keogh, "Generating synthetic time series to augment sparse datasets," vol. 2017-November, IEEE, Nov. 2017, pp. 865–870, ISBN: 978-1-5386-3835-4. DOI: 10.1109/ICDM.2017.106. [Online]. Available: https://ieeexplore.ieee.org/document/8215569/.

[31] S. Gupta and A. Gupta, "Dealing with noise problem in machine learning datasets: A systematic review," *Procedia Computer Science*, vol. 161, pp. 466–474, Jan. 2019, ISSN: 18770509. DOI: 10.1016/j.procs.2019.11.146. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1877050919318575.

[32] A. L. Guennec, S. Malinowski, and R. Tavenard, "Data augmentation for time series classification using convolutional neural networks," Sep. 2016. [Online]. Available: https://halshs.archives-ouvertes.fr/halshs-01357973%20https://halshs.archives-ouvertes.fr/halshs-01357973/document.

[33] C. Oh, S. Han, and J. Jeong, "Time-series data augmentation based on interpolation," *Procedia Computer Science*, vol. 175, pp. 64–71, 2020, ISSN: 18770509. DOI: 10.1016/j.procs.2020.07.012.

[34] T. T. Um, F. M. J. Pfister, D. Pichler, *et al.*, "Data augmentation of wearable sensor data for parkinson's disease monitoring using convolutional neural networks *," vol. 17, 2017. DOI: 10.1145/3136755.3136817. [Online]. Available: https://doi.org/10.1145/3136755.3136817.

[35] B. K. Iwana and S. Uchida, "An empirical survey of data augmentation for time series classification with neural networks," *PLOS ONE*, vol. 16, e0254841, 7 Jul. 2021, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0254841.

[36] T. Fields, G. Hsieh, and J. Chenou, "Mitigating drift in time series data with noise augmentation," *Proceedings - 6th Annual Conference on Computational Science and Computational Intelligence, CSCI 2019*, pp. 227–230, Dec. 2019. DOI: 10.1109/CSCI49370.2019.00046.

[37] B. K. Iwana and S. Uchida, "Time series data augmentation for neural networks by time warping with a discriminative teacher," [Online]. Available: https://github.com/uchidalab/time.

[38] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," Dec. 2014. [Online]. Available: https://arxiv.org/abs/1412.3555v1.

[39] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," Oct. 2021. [Online]. Available: https://arxiv.org/abs/2111.00396v1.

[40] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. arXiv: 1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762.

[41] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[43] J. Wu, X. Y. Chen, H. Zhang, L. D. Xiong, H. Lei, and S. H. Deng, "Hyperparameter optimization for machine learning models based on bayesian optimization," *Journal of Electronic Science and Technology*, vol. 17, pp. 26–40, 1 Mar. 2019, ISSN: 1674-862X. DOI: 10.11989/JEST.1674-862X.80904120.

[44] E. Larson, A. Gramfort, D. A. Engemann, *et al.*, *Mne-tools/mne-python: V0.24.1*, 2021. DOI: 10.5281/ZENODO.5748364. [Online]. Available: https://zenodo.org/record/5748364.

[45] R. Oostenveld, P. Fries, E. Maris, and J.-M. Schoffelen, "Fieldtrip: Open source software for advanced analysis of meg, eeg, and invasive electrophysiological data," *Computational intelligence and neuroscience*, vol. 2011, 2011.

[46] L. Biewald, *Experiment tracking with weights and biases*, Software available from wandb.com, 2020. [Online]. Available: https://www.wandb.com/.

[47] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2.

[48] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[49] T. pandas development team, *Pandas-dev/pandas: Pandas*, version latest, Feb. 2020. DOI: 10.5281/zenodo.3509134. [Online]. Available: https://doi.org/10.5281/zenodo.3509134.

[50] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[51] W. Falcon and The PyTorch Lightning team, *PyTorch Lightning*, version 1.4, Mar. 2019. DOI: 10.5281/zenodo.3828935. [Online]. Available: https://github.com/PyTorchLightning/pytorch-lightning.

[52] J. Bradbury, R. Frostig, P. Hawkins, *et al.*, *JAX: Composable transformations of Python+NumPy programs*, version 0.2.5, 2018. [Online]. Available: http://github.com/google/jax.

[53] European Union. "General data protection regulation." (2016), [Online]. Available: https://gdpr.eu/ (visited on 06/27/2022).

[54] United Nations Development Programme. "THE SDGS IN ACTION." (2022), [Online]. Available: https://www.undp.org/sustainable-development-goals (visited on 06/27/2022).

[55] W. S. Mcculloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," 1943.

[56] A. Nicolae, "Plu: The piecewise linear unit activation function," Sep. 2018. DOI: 10.48550/arxiv.1809.09534. [Online]. Available: http://arxiv.org/abs/1809.09534.

[57] Y. Zhou, Z. Zhu, and Z. Zhong, "Learning specialized activation functions with the piecewise linear unit," Apr. 2021. [Online]. Available: http://arxiv.org/abs/2104.03693.

[58] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[59] J. Li, "Regression and classification in supervised learning," ACM Press, 2019, pp. 99–104, ISBN: 9781450372909. DOI: 10.1145/3366650.3366675. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3366650.3366675.

[60] Y. Ho and S. Wookey, "The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling," *IEEE Access*, vol. 8, pp. 4806–4813, 2020, ISSN: 21693536. DOI: 10.1109/ACCESS.2019.2962617.

[61] S. Ruder, "An overview of gradient descent optimization algorithms," Sep. 2016. [Online]. Available: http://arxiv.org/abs/1609.04747.

[62] D. Foster, "Generative deep learning," vol. 6, p. 308, November 2019. [Online]. Available: https://www.oreilly.com/library/view/generative-deep-learning/9781492041931/.

[63] M. M. Bronstein, J. Bruna, T. Cohen, and P. Velickovic, "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges," *CoRR*, vol. abs/2104.13478, 2021. arXiv: 2104.13478. [Online]. Available: https://arxiv.org/abs/2104.13478.

[64] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, pp. 78–87, 10 Oct. 2012, ISSN: 0001-0782. DOI: 10.1145/2347736.2347755. [Online]. Available: https://dl.acm.org/doi/10.1145/2347736.2347755.

A

Theory On Artificial Neural Networks

This chapter covers in-depth theory on concepts related to artificial neural networks such as neurons, activation functions, loss functions, stochastic gradient descent, backpropagation, generative and discriminative models, hypothesis space, and generalization.

A.1 Neurons

The neurons used in artificial neural networks are based on the binary threshold unit neurons (McCulloch-Pitts neurons) originally invented by McCulloch and Pitts in 1943. The ith neuron in their model is defined as a unit having input states $s_j(t)$, input weights $w_{i,j}$, a threshold (also called bias) value θ_i, and an output state $s_i(t+1)$. The model repeats the same computations for a certain amount of time steps which is the reason for the time step index t. The index j is in the range 1 to the length of the model input. The output of a McCulloch-Pitts neuron is either -1 or 1 since the output is computed as the signum of the neuron's *local field*. The local field is shown as the argument to the signum function in Figure A.1, which shows a schematic of a McCulloch-Pitts neuron. The local field is the difference between the weighted sum of the neuron's inputs and its threshold. The use of the signum function explains why the neurons are called binary threshold units here. An example of a model that utilizes McCulloch-Pitts neurons is the Hopfield model [22], [55].

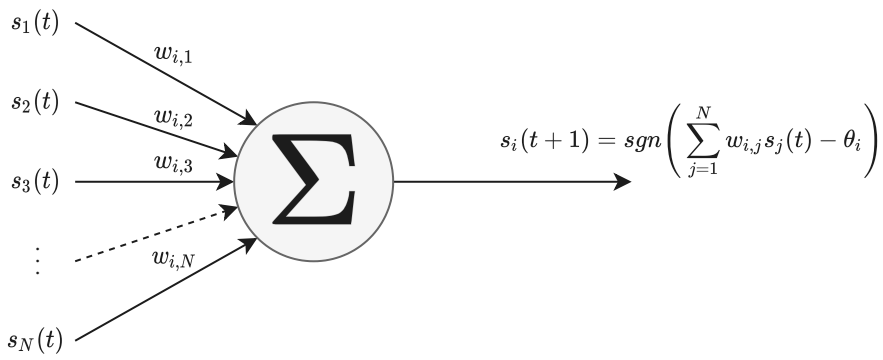


Figure A.1: A McCulloch-Pitts neuron schematic.

The McCulloch-Pitts neuron uses the signum function as mentioned earlier to compute its output. This is however a special case of the general neuron (see Figure A.2) that is used in ANNs where the function applied is arbitrary and is called the *activation function*.

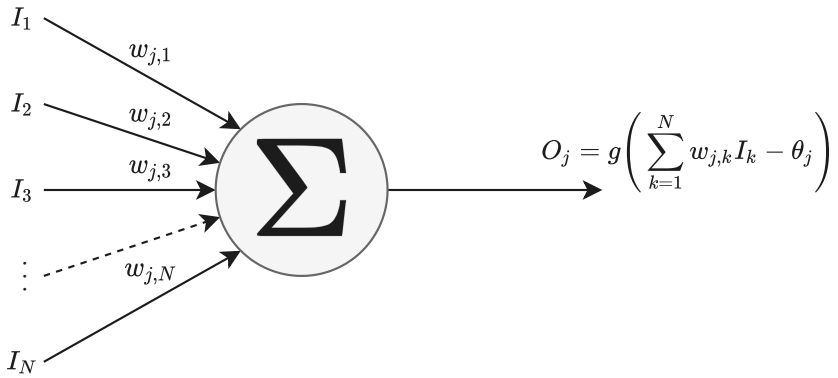


Figure A.2: A general neuron schematic. The activation function is denoted as g.

In the McCulloch-Pitts case, the activation function is discontinuous and discrete. In contrast, in the case of it being continuous, the neurons are allowed to take on continuous state values [22]. Appendix A.2 covers activation functions in more detail.

A.2 Activation Functions

Activation functions are used in each neuron to produce an output based on its local field (see Appendix A.1). There are linear, non-linear, and piecewise linear activation functions.

Linear activation functions can only solve problems that are linearly separable while non-linear ones can solve ones that are not. One example of a non-linear activation function is tangens hyperbolicus (*tanh*), which is shown in Figure A.3. The sigmoid and softmax functions are other examples. A problem with using non-linear activations functions is that they result in the *vanishing gradient problem* as the ANNs get deeper (meaning get more layers). If the gradient of the activation function is smaller than 1 in many layers in the ANN, their product will get very close to 0 and the network will thus not learn anything during backpropagation (see Appendix A.4) [22], [56].

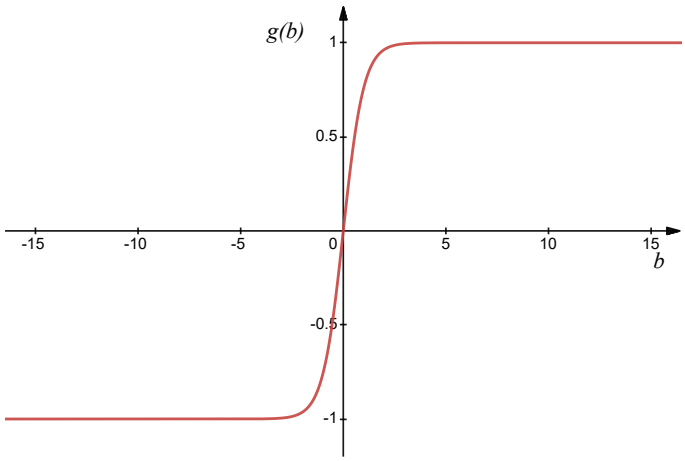


Figure A.3: A non-linear, continuous activation function: Tangens hyperbolicus ($g(b) = tanh(b)$).

One way to mitigate the vanishing gradient problem is to use so-called piecewise linear activation functions instead of non-linear [56], [57]. One example of such an activation function is ReLU (rectified linear unit, $g(b) = \max(0,\ b)$), which is shown in Figure A.4. The derivative of ReLU is constantly 1 for local fields larger than 0. If ReLU is used in every layer, it means that the product of gradients during backpropagation will not approach 0 along such neuron paths in the network as it becomes deeper.

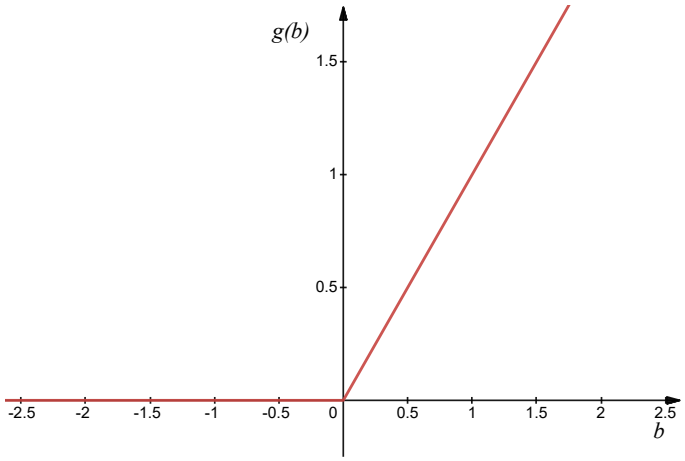


Figure A.4: A piecewise linear activation function: ReLU ($g(b) = max(0, b)$).

Piecewise linear activation functions can of course be more complicated and even mimic non-linear ones. For example, the Piecewise Linear Unit (PLU), suggested by Nicolae at the University of Washington, mimics *tanh*. It is said to be superior to ReLU in certain tasks while not resulting in the vanishing gradient problem that *tanh* induces [56].

A.3 Loss Functions

Artificial neural networks are, as stated before, function approximators. In order to approximate a function, it needs to learn when it classifies its input incorrectly. This is where loss functions enter the stage. They are used to compute how far off the ANN's predictions are from the ground truth [22], [58]. The loss function has to be chosen with the problem at hand in mind; is it a linear regression problem, binary classification problem, or a multi-class classification problem? The difference between regression problems and classification problems is that the former involves predicting continuous quantities while the latter involves predicting discrete class labels [59]. In the case of this thesis, the problem is a binary classification problem and the labels are MSNA inhibitor and non-inhibitor.

An example of a binary classification loss function is binary cross-entropy (BCE) which can be defined as in Equation A.1 for a single input data point $\vec{x}^{(\mu)}$. When training the network on a mini batch $\mathbf{X}_{(\nu)}$ of data points instead, BCE can be defined as in Equation A.2. Here, the loss becomes an average loss over the data points in the batch, resulting in less noisy parameter updates during backpropagation (see Appendix A.4) [22], [60].

$$l(\vec{x}^{(\mu)}, l^{(\mu)}, \boldsymbol{\Theta}) = -\sum_{i=1}^{M} \left[\delta_{i,l^{(\mu)}} \log\left(\hat{f}_{\boldsymbol{\Theta}}(\vec{x}^{(\mu)})_i \right) + (1 - \delta_{i,l^{(\mu)}}) \log\left(1 - \hat{f}_{\boldsymbol{\Theta}}(\vec{x}^{(\mu)})_i \right) \right] \tag{A.1}$$

$$\mathcal{L}(\mathbf{X}_{(\nu)}, \vec{l}_{(\nu)}, \boldsymbol{\Theta}) = \frac{1}{\|\mathbf{X}_{(\nu)}\|} \sum_{\mu=1}^{\|\mathbf{X}_{(\nu)}\|} l\left(\mathbf{X}_{(\nu)}^{(\mu)}, \vec{l}_{(\nu)}^{(\mu)}, \boldsymbol{\Theta} \right) \tag{A.2}$$

$$\begin{aligned}
M &\coloneqq 2 \text{ (Number of classes)} \\
p &\coloneqq \text{The total number of data points} \\
\boldsymbol{\Theta} &\coloneqq \text{The parameter matrix of the network} \\
\vec{x}^{(\mu)} &\coloneqq \text{Data point vector } (\mu \in [1, p]) \\
\mathbf{X}_{(\nu)} &\coloneqq \text{Mini batch } \nu \text{ matrix } \left(\|\mathbf{X}_{(\nu)}\| \in [1, p],\ \nu \in \left[1, \left\lceil \frac{p}{\|\mathbf{X}_{(\nu)}\|} \right\rceil \right] \right) \\
l^{(\mu)} &\coloneqq \text{The label of the } \mu\text{th data point } \ (l^{(\mu)} \in \{0, 1\},\ \forall \mu \in [1, p]) \\
\delta_{a,b} &\coloneqq \begin{cases} 1, & \text{if } a = b \\ 0, & \text{otherwise} \end{cases} \quad \text{(Kronecker delta)} \\
\hat{f}_{\boldsymbol{\Theta}} &\coloneqq \text{The function approximated by the network } (\hat{f}_{\boldsymbol{\Theta}} \mapsto \vec{y},\ \vec{y} \in \mathbb{R}^M)
\end{aligned}$$

Mean Squared Error (MSE) is an example of a loss function used in regression problems. In geometric terms, it is the average squared Euclidean distance between predictions and targets [58]. The MSE loss function can be defined as in Equation A.3 for a mini batch of data points $\mathbf{X}_{(\nu)}$ and targets $\mathbf{Y}_{(\nu)}$. This loss function can for

example be used when training autoencoders (see Section 2.1.1).

$$\mathcal{L}(\mathbf{X}_{(\nu)}, \mathbf{Y}_{(\nu)}, \boldsymbol{\Theta}) = \frac{1}{\|\mathbf{X}_{(\nu)}\|} \sum_{\mu=1}^{\|\mathbf{X}_{(\nu)}\|} \left(\hat{f}_{\boldsymbol{\Theta}}\left(\mathbf{X}_{(\nu)}^{(\mu)}\right) - \mathbf{Y}_{(\nu)}^{(\mu)} \right)^2 \tag{A.3}$$

The general usage of loss functions will be explained in more detail in the context of stochastic gradient descent and backpropagation in Appendix A.4.

A.4 Stochastic Gradient Descent

The expected loss $J(\boldsymbol{\Theta})$ of an ANN can be defined as in Equation A.4. By taking steps downhill its gradient with respect to the network parameters $\boldsymbol{\Theta}$ and updating them along the way to convergence, the parameters are optimized to minimize the loss. In deterministic gradient descent, it is common to get stuck in local minima and thus never reach a global one. Stochastic gradient descent (SGD) on the other hand makes it possible to escape local minima to potentially find a global minimum later on. By randomly estimating the gradient when taking steps along it, the estimation error can allow for steps that are not downhill. Taking a step uphill can obviously result in an escape from a local minima. Whether a global minima is found during the training process thus depends on stochasticity and the training time T ($T \in \mathbb{Z}^+$) [22], [58], [61]. The rest of this section will explain how SGD makes gradient estimations and how the network parameters $\boldsymbol{\Theta}$ are updated during training.

Here, $\mathbf{X}$ denotes the data point matrix, $\mathbf{Y}$ denotes the target matrix, $\hat{p}_{\text{data}}$ denotes the data distribution, and $\mathcal{L}$ denotes the loss function (see Appendix A.3).

$$J(\boldsymbol{\Theta}) = \mathbb{E}_{\mathbf{X},\mathbf{Y}\sim\hat{p}_{\text{data}}}[\mathcal{L}(\mathbf{X}, \mathbf{Y}, \boldsymbol{\Theta})] \tag{A.4}$$

$$= \frac{1}{\|\mathbf{X}\|} \sum_{\mu=1}^{\|\mathbf{X}\|} \mathcal{L}(\mathbf{X}^{(\mu)}, \mathbf{Y}^{(\mu)}, \boldsymbol{\Theta}) \tag{A.5}$$

The gradient of $J(\boldsymbol{\Theta})$ with respect to the network parameters $\boldsymbol{\Theta}$ is defined as in Equation A.6 [58], [61].

$$\nabla_{\boldsymbol{\Theta}} J(\boldsymbol{\Theta}) = \nabla_{\boldsymbol{\Theta}} \left(\frac{1}{\|\mathbf{X}\|} \sum_{\mu=1}^{\|\mathbf{X}\|} \mathcal{L}(\mathbf{X}^{(\mu)}, \mathbf{Y}^{(\mu)}, \boldsymbol{\Theta}) \right) \tag{A.6}$$

$$= \frac{1}{\|\mathbf{X}\|} \sum_{\mu=1}^{\|\mathbf{X}\|} \nabla_{\boldsymbol{\Theta}} \mathcal{L}(\mathbf{X}^{(\mu)}, \mathbf{Y}^{(\mu)}, \boldsymbol{\Theta}) \tag{A.7}$$

$$= \mathbb{E}_{\mathbf{X},\mathbf{Y}\sim\hat{p}_{\text{data}}}[\nabla_{\boldsymbol{\Theta}} \mathcal{L}(\mathbf{X}, \mathbf{Y}, \boldsymbol{\Theta})] \tag{A.8}$$

Notice that the gradient of $J(\mathbf{\Theta})$ is an expectation (see Equation A.8). It can thus be estimated with a subset of the whole dataset. In SGD, a so-called mini batch $\mathbf{X}_{(\nu)}$ is uniformly sampled from $\mathbf{X}$ ($\mathbf{X}_{(\nu)} \subset \mathbf{X}$). This stochastic estimation of the gradient makes it possible to escape local minima as mentioned before [22], [58], [61]. The mini batch estimation of the expected loss $J(\mathbf{\Theta})$ can be defined as in Equation A.9.

$$\nabla_{\mathbf{\Theta}} \hat{J}(\mathbf{\Theta}) = \frac{1}{\|\mathbf{X}_{(\nu)}\|} \sum_{\mu=1}^{\|\mathbf{X}_{(\nu)}\|} \nabla_{\mathbf{\Theta}} \mathcal{L}\Big(\mathbf{X}_{(\nu)}^{(\mu)}, \mathbf{Y}_{(\nu)}^{(\mu)}, \mathbf{\Theta}\Big) \approx \nabla_{\mathbf{\Theta}} J(\mathbf{\Theta}) \tag{A.9}$$

The network parameters $\mathbf{\Theta}$ are then updated by taking a step of length η (learning rate) along the estimated gradient in the direction $\nabla_{\mathbf{\Theta}} \hat{J}(\mathbf{\Theta})$ [58], [61]. The gradient step at time step t ($t \leq T$) will here be denoted as $\delta\mathbf{\Theta}_{(t)}$, as seen in Equation A.10. The update of the parameters at time step t is shown in Equation A.11.

$$\delta\mathbf{\Theta}_{(t)} = -\eta \nabla_{\mathbf{\Theta}} \hat{J}(\mathbf{\Theta}) \tag{A.10}$$

$$\mathbf{\Theta}_{(t+1)} \leftarrow \mathbf{\Theta}_{(t)} + \delta\mathbf{\Theta}_{(t)} \tag{A.11}$$

The learning rate η can be adapted during training to take steps appropriate for the gradient at the current location. This can for example be done by adding *momentum* to the step based on the previous step. A new hyperparameter α, called the *momentum constant* is therefore introduced (affects the intertia, $\alpha \geq 0$). Equation A.12 shows the momentum being added to the step at time step t. Adding momentum results in larger steps in shallow minima and shorter ones where the gradient oscillates rapidly [22]. This thus allows for faster and more stable convergence, hopefully to the global minimum of the loss function in question.

$$\delta\mathbf{\Theta}_{(t)} = -\eta \nabla_{\mathbf{\Theta}} \hat{J}(\mathbf{\Theta}) + \alpha \delta\mathbf{\Theta}_{(t-1)} \tag{A.12}$$

The backpropagation of the loss through an ANN is done using the chain rule to compute the parameter updates [22]. To explain this concept in a simple fashion, consider the multilayer perceptron (MLP, see Section 2.3.1) which is a feed-forward ANN. Equation A.13 shows the update for the weight matrix $\mathbf{W}^{(l)}$ in layer l in the MLP. Note that this example does not include momentum but that it can be added easily. Also note that the time step notation has been hidden to make the derivation less cluttered.

$$\begin{aligned} \delta\mathbf{W}^{(l)} &= -\eta \frac{\partial \hat{J}(\mathbf{\Theta})}{\partial \mathbf{W}^{(l)}} \\ &= -\frac{\eta}{\|\mathbf{X}_{(\nu)\|}} \sum_{\mu=1}^{\|\mathbf{X}_{(\nu)}\|} \frac{\partial \mathcal{L}\Big(\mathbf{X}_{(\nu)}^{(\mu)}, \mathbf{Y}_{(\nu)}^{(\mu)}, \mathbf{\Theta}\Big)}{\partial \mathbf{W}^{(l)}} \end{aligned} \tag{A.13}$$

$$
\begin{aligned}
&\propto \sum_{\mu=1}^{\|\mathbf{X}_{(\nu)}\|} \frac{\partial \mathcal{L}\left(\mathbf{X}_{(\nu)}^{(\mu)}, \mathbf{Y}_{(\nu)}^{(\mu)}, \boldsymbol{\Theta}\right)}{\partial \vec{V}_{(\nu)}^{(L,\mu)}} \odot \frac{\partial \vec{V}_{(\nu)}^{(L,\mu)}}{\partial \mathbf{W}^{(l)}} \\
&\propto \sum_{\mu=1}^{\|\mathbf{X}_{(\nu)}\|} \frac{\partial \mathcal{L}\left(\mathbf{X}_{(\nu)}^{(\mu)}, \mathbf{Y}_{(\nu)}^{(\mu)}, \boldsymbol{\Theta}\right)}{\partial \vec{V}_{(\nu)}^{(L,\mu)}} \odot \frac{\partial \vec{V}_{(\nu)}^{(L,\mu)}}{\partial \vec{V}_{(\nu)}^{(L-1,\mu)}} \odot \frac{\partial \vec{V}_{(\nu)}^{(L-1,\mu)}}{\partial \mathbf{W}^{(l)}} \\
&\propto \sum_{\mu=1}^{\|\mathbf{X}_{(\nu)}\|} \frac{\partial \mathcal{L}\left(\mathbf{X}_{(\nu)}^{(\mu)}, \mathbf{Y}_{(\nu)}^{(\mu)}, \boldsymbol{\Theta}\right)}{\partial \vec{V}_{(\nu)}^{(L,\mu)}} \odot \frac{\partial g\left(\vec{b}_{(\nu)}^{(L,\mu)}\right)}{\partial \vec{b}_{(\nu)}^{(L,\mu)}} \odot \left(\frac{\partial \vec{b}_{(\nu)}^{(L,\mu)}}{\partial \vec{V}_{(\nu)}^{(L-1,\mu)}} \frac{\partial \vec{V}_{(\nu)}^{(L-1,\mu)}}{\partial \mathbf{W}^{(l)}} \right) \\
&\propto \sum_{\mu=1}^{\|\mathbf{X}_{(\nu)}\|} \frac{\partial \mathcal{L}\left(\mathbf{X}_{(\nu)}^{(\mu)}, \mathbf{Y}_{(\nu)}^{(\mu)}, \boldsymbol{\Theta}\right)}{\partial \vec{V}_{(\nu)}^{(L,\mu)}} \odot \frac{\partial g\left(\vec{b}_{(\nu)}^{(L,\mu)}\right)}{\partial \vec{b}_{(\nu)}^{(L,\mu)}} \odot \left(\mathbf{W}^{(L)} \frac{\partial \vec{V}_{(\nu)}^{(L-1,\mu)}}{\partial \mathbf{W}^{(l)}} \right) \\
&\propto \sum_{\mu=1}^{\|\mathbf{X}_{(\nu)}\|} \frac{\partial \mathcal{L}\left(\mathbf{X}_{(\nu)}^{(\mu)}, \mathbf{Y}_{(\nu)}^{(\mu)}, \boldsymbol{\Theta}\right)}{\partial \vec{V}_{(\nu)}^{(L,\mu)}} \odot \frac{\partial g\left(\vec{b}_{(\nu)}^{(L,\mu)}\right)}{\partial \vec{b}_{(\nu)}^{(L,\mu)}} \odot \left(\mathbf{W}^{(L)} \dots \frac{\partial \vec{V}_{(\nu)}^{(l,\mu)}}{\partial \mathbf{W}^{(l)}} \right) \\
&\propto \sum_{\mu=1}^{\|\mathbf{X}_{(\nu)}\|} \frac{\partial \mathcal{L}\left(\mathbf{X}_{(\nu)}^{(\mu)}, \mathbf{Y}_{(\nu)}^{(\mu)}, \boldsymbol{\Theta}\right)}{\partial \vec{V}_{(\nu)}^{(L,\mu)}} \odot \frac{\partial g\left(\vec{b}_{(\nu)}^{(L,\mu)}\right)}{\partial \vec{b}_{(\nu)}^{(L,\mu)}} \odot \left(\mathbf{W}^{(L)} \dots \frac{\partial g\left(\vec{b}_{(\nu)}^{(l,\mu)}\right)}{\partial \vec{b}_{(\nu)}^{(l,\mu)}} \odot \frac{\partial \vec{b}_{(\nu)}^{(l,\mu)}}{\partial \mathbf{W}^{(l)}} \right) \\
&\propto \sum_{\mu=1}^{\|\mathbf{X}_{(\nu)}\|} \frac{\partial \mathcal{L}\left(\mathbf{X}_{(\nu)}^{(\mu)}, \mathbf{Y}_{(\nu)}^{(\mu)}, \boldsymbol{\Theta}\right)}{\partial \vec{V}_{(\nu)}^{(L,\mu)}} \odot \frac{\partial g\left(\vec{b}_{(\nu)}^{(L,\mu)}\right)}{\partial \vec{b}_{(\nu)}^{(L,\mu)}} \odot \left(\mathbf{W}^{(L)} \dots \frac{\partial g\left(\vec{b}_{(\nu)}^{(l,\mu)}\right)}{\partial \vec{b}_{(\nu)}^{(l,\mu)}} \odot \vec{V}_{(\nu)}^{(l-1,\mu)} \right)
\end{aligned}
$$

$L \coloneqq$ The number of layers in the MLP (depth)

$l \in [0,\ L]$

$g \coloneqq$ Element-wise activation function (see Section A.2)

$\vec{b}_{(\nu)}^{(l,\mu)} \coloneqq$ The local field vector in layer l (data point μ, mini batch ν)

$$\vec{b}_{(\nu)}^{(l,\mu)} = \mathbf{W}^{(l)} \vec{V}_{(\nu)}^{(l-1,\mu)} - \vec{\theta}^{(l)}$$

$\vec{V}_{(\nu)}^{(l,\mu)} \coloneqq$ The neuron state vector in layer l (data point μ, mini batch ν)

$$\vec{V}_{(\nu)}^{(l,\mu)} = g\left(\vec{b}_{(\nu)}^{(l,\mu)}\right)$$

The general MLP backpropagation formula shown in Equation A.13 can be concretized with for example the mean squared error loss function (MSE, see Appendix A.3). The gradient of the loss function then looks like Equation A.14.

$$\begin{aligned}\frac{\partial \mathcal{L}(\mathbf{X}_{(\nu)}^{(\mu)}, \mathbf{Y}_{(\nu)}^{(\mu)}, \boldsymbol{\Theta})}{\partial \vec{V}_{(\nu)}^{(L,\mu)}} &= \frac{\partial}{\partial \vec{V}_{(\nu)}^{(L,\mu)}} \left(\frac{1}{\|\mathbf{X}_{(\nu)}^{(\mu)}\|} \left(\hat{f}_{\boldsymbol{\Theta}} \left(\mathbf{X}_{(\nu)}^{(\mu)} \right) - \mathbf{Y}_{(\nu)}^{(\mu)} \right) \cdot \left(\hat{f}_{\boldsymbol{\Theta}} \left(\mathbf{X}_{(\nu)}^{(\mu)} \right) - \mathbf{Y}_{(\nu)}^{(\mu)} \right) \right) \\ &= \frac{\partial}{\partial \vec{V}_{(\nu)}^{(L,\mu)}} \left(\frac{1}{\|\mathbf{X}_{(\nu)}^{(\mu)}\|} \left(\vec{V}_{(\nu)}^{(L,\mu)} - \mathbf{Y}_{(\nu)}^{(\mu)} \right) \cdot \left(\vec{V}_{(\nu)}^{(L,\mu)} - \mathbf{Y}_{(\nu)}^{(\mu)} \right) \right) \\ &= \frac{2}{\|\mathbf{X}_{(\nu)}^{(\mu)}\|} \left(\vec{V}_{(\nu)}^{(L,\mu)} - \mathbf{Y}_{(\nu)}^{(\mu)} \right) \end{aligned} \tag{A.14}$$

The backpropagation derivation in Equation A.13 shows the issue of vanishing gradients, previously mentioned in Appendix A.2. If the network is deep and the activation function gradients are smaller than 1, the parameter update will be very small and the network will virtually learn nothing. As mentioned in Appendix A.2, there are workarounds.

A.5 Generative and Discriminative Models

There are two types of deep learning models: generative and discriminative. The former involves estimating how probable observations are. The latter involves estimating how probable a label is given an observation [62]. Generative models are thus not classifiers on their own but can be used to learn representations of the raw data that potentially are easier to classify with discriminative models [21].

A.5.1 Generative Models

There are two families of generative models related to time series classification: Autoencoders and echo state networks (ESN) [21]. The former was described in Section 2.1.1. The latter are seemingly the same as what Mehlig refers to as reservoir computers [22].

Echo state networks are recurrent neural networks (RNN) and can also be used to create new representations of time series data. The reservoir in an ESN is a dynamical memory which means that the values of all its neurons at time step T represent the input time series for all $t \leq T$ [21], [22]. The reservoir is thus a representation of the data which can also be fed into discriminative models for classification.

A.5.2 Discriminative Models

Discriminative models can be divided into two groups: Feature engineering and end-to-end models. The former involves extracting features, engineered based on domain knowledge, from the data and then feeding them into a discriminative model. End-to-end models however learn the features from the data on their own while also tweaking the parameters of the classifier [21].

Section 2.3 mentions some discriminative models that could be useful for time series classification and describes ones actually used in this thesis in more detail.

A.6 Hypothesis Space and Generalization

Different ANN models have different prior probabilities for approximating a function well enough for it to generalize on unseen data [63]. If the true function was found, it would generalize accurately at all times. The space of possible function approximations is called the *hypothesis space* [64]. A model can only generalize if its hypothesis space intersects with a set of functions that explain the data distribution relatively well. Even then, the intersection might be so small that it is very unlikely to reach generalization. This is visualized in Figure A.5 where one model has a higher probability of generalizing than the other, based on their priors.

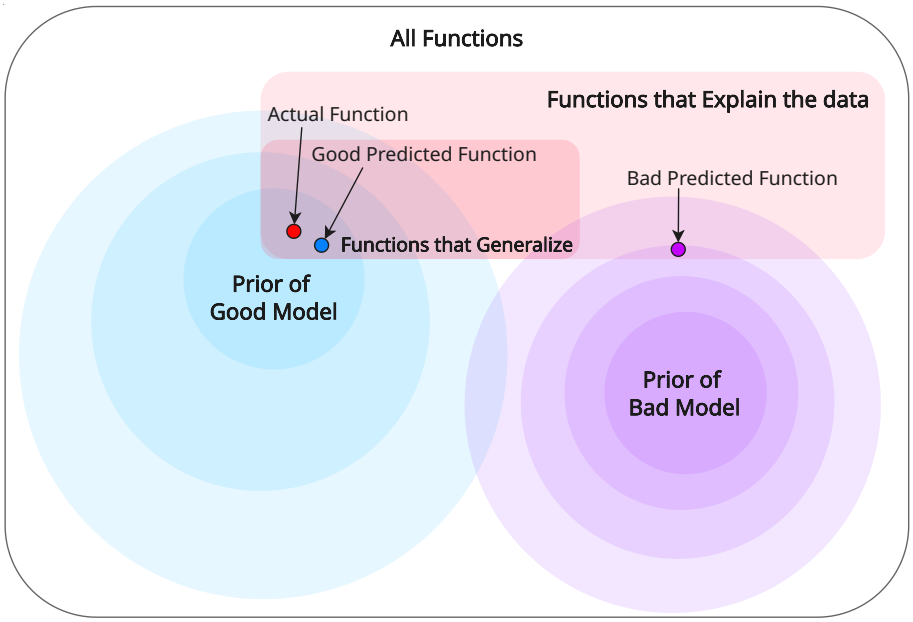


Figure A.5: The blue circles represent the hypothesis space of the "good" model, the purple ones represent the hypothesis space of the "bad" model, and the red area represents the set of functions that describe the data distribution well.

It is clear that the amount of data is not the only factor in reaching generalization. The model architecture plays a role as well since it affects the hypothesis space. However, if the true function or a good approximation is in the hypothesis space, the probability of finding a bad approximation decreases as the data set cardinality increases [64].

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY