Deep Learning Techniques for Trust Assessment of Social Human-Robot Interactions

Research Project

Universidad Politécnica de Madrid

by

Simon Hommel

Course:	Research Project
Semester:	Winter Term 2024
Submission date:	January 10th, 2025
Supervisor:	Prof. Dr. Laura Melgar
Leading Professor:	Prof. Dr. Javier Bajo

Abstract

This research study investigates advanced deep learning techniques for trust level estimation in social human-robot interactions using Electroencephalogram (EEG) data. Building upon the foundational work of paper [1] that utilized conventional machine learning methods, this study systematically evaluates state-of-the-art neural network architectures, including feed-forward networks, Recurrent Neural Network (RNN)s, and EEGNet. By focusing on classifying trust levels across experimental phases spanning trust-building, transparency, violation, and repair, the research addresses key challenges in trust modeling, such as class imbalance, temporal dependencies, and noise in EEG signals.

The pre-processing techniques which are used to enhance data quality, include artifact minimization through Independent Component Analysis and feature extraction via Fast Fourier Transform (FFT). The corresponding groundwork for the pre-processing steps is described in [1]. EEGNet, a compact and domain-specific architecture, demonstrated superior performance by efficiently capturing spatio-temporal patterns in the raw EEG signals. Results reveal that EEGNet not only outperformed traditional machine learning models but also surpassed other deep learning approaches like feed-forward networks or RNNs in accuracy. The integration of data augmentation techniques further improved model robustness and generalization, addressing the dataset limitations mainly rooted in the inherent imbalance in the trust level classes.

Despite these advancements, challenges such as inconsistencies in time-aligned labeling and potential overfitting due to limited data were tried to be carefully mitigated through methodological adaptations. This study underscores the role of domain-specific architectures, robust pre-processing, and data handling techniques in EEG-based trust classification. It contributes to the broader understanding of trust dynamics in human-robot interaction, with potential applications in collaborative robotics, healthcare, and other social environments. Future directions include exploring transformer-based models for enhanced temporal dependency modeling, cross-subject transfer learning for improved generalizability, and expanding and enhancing the dataset to support more complex architectures.

Contents

List of Figures

List of Tables

1. Introduction and Motivation

The rapid evolution of deep learning has paved the way for advancements across various fields, enabling more accurate and efficient analyses of complex data. Leveraging different deep learning techniques for improvement and optimization has become increasingly essential, especially as datasets grow in size and complexity. In the context of this research project, the focus is placed on evaluating and enhancing the performance of a baseline work with advanced deep learning architectures. This investigation is based upon the previous work described in the paper "Trust Assessment with EEG Signals in Social Human-Robot Interaction" [1], which outlines structured and conventional machine learning approaches for trust classification.
This report aims to systematically explore and apply state-of-the-art deep learning techniques, such as Convolutional Neural Network (CNN)s, RNNs, and Transformer models, each selected for their unique ability to handle specific data patterns. By benchmarking these approaches, the aim is to gain insights into the strengths and limitations of those architectures. Furthermore, techniques like data augmentation and hyperparameter tuning are explored to improve the models' ability to generalize effectively.

In summary, this study contributes to deep learning research by providing a comparative analysis of several methods to understand the factors influencing performance outcomes of the analysis task. This research seeks to advance the findings presented in the base study, further optimizing and exploring for potential applications in real-world scenarios.
Since there is limited availability of implementation details in the paper [1], own code and a repository which holds all necessary information and files which were created over the course of the research project, was created.
The implementation details and source code can be found on github: https://github.com/Sim089n/EEG-ResearchProject.

1.1. Human Robot Interaction

The baseline for the presented work is the published work in [1]. Hereby the main research objective is to test the human participant's ability and sensitivity in terms of trusting a robot while playing a game. While playing the game the human is only interacting with the robot as a second player. This interaction and behaviour of the human participant is especially important for the future study and application of robots in different fields. Thus meaning that robots which interact and work with people in social jobs or environments need higher stability and reliability in their actions when it comes to building trust to co-workers and affected people around them. The role of robots changed drastically over the last decades which can be mainly explained through the advancement of new technologies and higher computational resources when it comes to learning and teaching robots. Nowadays robots already support, collaborate, transport, or entertain humans in very different and diverse ways. This way interacting with robots is no longer just a matter for factory works which use robots for industrial purposes but it becomes more and more important in the day-to-day lives of many humans.

The main objective of Human Robot Interaction (HRI) is strongly correlated and connected with Human Computer Interaction (HCI) including robotics, artificial intelligence, the philosophy of technology, psychology, and design [4]. The main part of this field is the investigation of the interaction of humans with social robots. This usually includes physically embodied robots, with their unusual and unique embodiment. It also needs to be accounted for the fact that social robots are often perceived as social actors bearing cultural meaning and having an impact on existing and future communities and societies. Furthermore it is important to investigate and understand how to design embodiment in terms of software and hardware. This is important since this has effects on people and the interactions they can have with such a robot. To break it down, a robot's embodiment sets physical constraints on the ways in which it can sense and act in the world [4]. This becomes especially evident when the senses of a human are directly compared to a robot. The ability to feel through touch sensors or to see through a camera enables a robot to perceive and interact with and within our world. If the robot would not be able to feel it is not possible for it to control or manipulate things in our world which need a highly sensitive and finetuned interaction style. Since the world is permanently changing, the fields in which robots can be used will gain in variety resulting in a research field which is becoming more and more interdisciplinary. Therefore different stakeholders like engineers, psychologists, designers, anthropologists, sociologists, philosophers and other domain specific experts normally need to work together. To be successful in creating and deploying robotic systems the study of the human–robot interaction is necessary and one key pillar for an applicable system. It requires collaboration from a variety of fields to develop hardware and software. But most importantly the analysis of the behavior of humans when interacting with robots in different social contexts is one of the key challenges. This is why experiments like the one described and introduced in [1] are important and will gain more and more importance in the future as well. To classify if a person puts trust in a robots abilities is crucial for all future fields of application.

1.2. Electroencephalogram

Before deeper technical details of the used data and models are discussed a look at the way how the data is generated is needed. Connected with this it is important to with which kind of data is dealt with. Since the dataset consists of monitored EEG data a closer look at the method, which is used in medical diagnostics and neurological research to measure the accumulated electrical activity of the brain, is needed. This is done by recording voltage fluctuations on the surface of the head. The electroencephalogram is the graphical representation of these fluctuations. Therefore electrodes at the head of the participants are needed to measure the brain regions which are interesting for the research objective. Figure 1.1 provides an overview of the 10-20 system which is a widely used schema to attach the electrodes at the skull of a human, see [2]. The green marked points represent the points where the electrodes in the experiment of paper [1] were attached.
The potential fluctuations are caused by physiological processes of individual brain cells, which contribute to the brain's information processing. The processing in a human brain works mainly through the electrical state changes of the brain's neurons which are the processing units. Depending on their specific spatial arrangement, the potentials generated by individual neurons add up so that potential changes distributed over the entire head can be measured.

1.3. Key Aspects

There are some key aspects which should be regarded as important when dealing with EEG data. Especially for clinical evaluation at least twelve channels from different electrode combinations should be present. In the experiment conducted in [1] there are 14 channels which are used for monitoring the participants brain activity. The spatial resolution of the EEG amounts to several centimetres with a certain measurement insecurity. For measuring the voltage on the scalp which normally lies between 5 to 100 μV a sensitive measuring amplifier is required.
Since electrical voltages are always measured between two points they have to be arranged in a way that the measurement is possible, correct and accurate. The electrodes for the EEG are each attached in a specific system, according to which a distinction is made between different types of leads. Here the best known system is the 10-20 system, see 1.1. The EEG evaluation is traditionally carried out

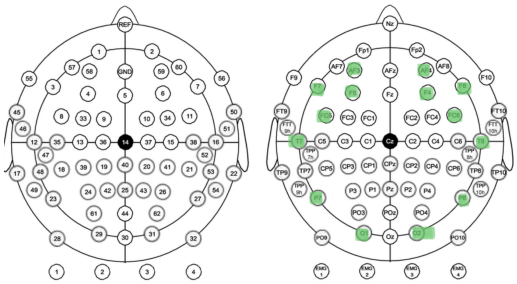


Figure 1.1.: The schema describes the arrangement of the electrodes on the head of an EEG participant based on the 10-20 system, [2].

through pattern recognition by the trained evaluator who in the most cases is a trained professional like a doctor or other medical personal. Through the analysis of the frequency bands it is possible to find out whether there are unusual patterns or not. For example, a very asynchronous pattern of all frequency bands indicates strong emotional stress or loss of voluntary control, while increased slow waves with few fast waves indicate a sleep or dozing state. More information regarding the different frequency bands and their meaning can be found in chapter 2.1.2.

2. Related Work

The base for the work which is discussed in this research project is build by the paper Trust Assessment with EEG Signals in Social Human-Robot Interaction [1]. The corresponding dataset is available together with the paper and deals with the question on how to measure and classify trust levels in human-robot interaction. This is one key question when it comes to improving the acceptance and application of robots in several domains. Especially in social tasks and environments robots could play a major role in the future.

2.1. Current State of the Art

The next subsections feature important knowledge which is needed to understand the work of [1] and the corresponding problem statement. Therefore the characteristics of the dataset are explained, the applied procedure of [1] is reviewed and some general information which are important to understand the structure of EEG data are given.

2.1.1. Dataset

The dataset includes the EEG measurements of the 21 participants as one Comma-separated values (csv)-file. The experiment was set up in the way that the robot had to explain a word either by verbal communication or through descriptive movements. During the game the participants were engaged through a simple reward system which resulted in either earning or losing candy based on their performance. The dataset was gathered using Emotiv headsets as the monitoring device for the brain activities. Each csv-file includes timestamps and measurements from 14 sensors placed across different regions of the scalp. The sensor labels in the header are as follows: EEG.AF3, EEG.F7, EEG.F3, EEG.FC5, EEG.T7, EEG.P7, EEG.O1, EEG.O2, EEG.P8, EEG.T8, EEG.FC6, EEG.F4, EEG.F8, EEG.AF4, and Time [1]. Aggregated the set includes 3,651,124 data points for each of the 14 sensors. This means for each participant an average of 173,863 data points is available whereas the minimum value throughout the 21 participants is 128,505 and the maximum is 249,631. The frame rate for the monitoring process is 128Hz. Based on those measurements it is possible to get insights into the electrical activity of the brain to analyze cognitive processes and emotional responses during various activities and situations during the game played with the robot. All the entries in the csv-file correspond to an electrical voltage in microvolt.
In order to make use of the data it has to be pre-processed through using the information given in the details.xlsx file. This file includes a start time and an end time, which represent distinct stages of the experimental game which is played between robot and human. It should be noted that all the other timestamps which aren't inside the start and end of each phase should be considered as breaks. This could for example include the filling out the questionnaires which are done after each phase of the game. The one which was used by the scientists to obtain the trust values can be seen here: A.1. Based on this MDMT questionnaire the corresponding trust scores for each participant after each phase of the game/experiment are calculated. Averaged over all participants threshold values are calculated to

label each trust score after each phase as either low or high. The following phases are annotated and directly cited from the website of the Benchmark EEG data:

1. Trust Building: This phase involves friendly initial interactions for establishing trust between participants and the robot.
2. Situational Awareness: This phase continues to build up trust by showing situation awareness of the robot, e.g. by complimenting on the participant's fashion choice.
3. Transparency: Trust is maintained by increased openness and clarity in communicating about the robot's abilities.
4. Trust Violation: Trust is compromised during this phase by deliberately misleading the participant and making it impossible to answer correctly.
5. Trust Repair: The robot shows efforts to repair trust by apologizing for the behavior in the previous stage.

Based on those different phases especially the difference in trust level between the Transparency and Trust Violation phase are interesting to observe in terms of the classification problem.

2.1.2. Frequency Bands

In the following section the different frequency bands of an EEG measurement are described. This is important because those characteristics define which frequencies are important for the trust level estimation and which pre-processing steps can be executed to extract those frequency band dependent information. The information are obtained from paper [5].

2.1.2.1. Delta Waves

Delta waves are characterized through a low frequency from 0.1 to <4 Hz. They are typical for the mostly dreamless deep sleep phases but can also occur in people with brain injuries or in pathological states such as coma. If delta waves are occurring during wakefulness it might indicate neurological problems such as cerebral haemorrhage, cerebral infarction or also brain tumour. The most important meaning of the delta waves is therefore that if they are dominant the participant is either having a very slow brain activity, is sleeping very deeply or undergoes regenerative processes of the brain. Respectively this means that the frequency band of the delta waves does not capture attributes which are correlated to trust.

2.1.2.2. Theta Waves

The Theta waves occur in the frequency spectrum between 4 to 8 Hz. They describe a state between wakefulness and sleep which is normally described as dozing or being in the early phase of falling asleep. But also during deep meditation phases Theta waves can occur in a higher amount. They are also common in states of heightened creativity or deep relaxation. In pathological states, they can occur in brain dysfunctions or attention deficits.

- Relaxation
- dozing
- meditation
- creative states

- early sleep phases

Corresponding to the meaning of the theta waves the trust measurement or the estimation of trust level can not be estimated by analyzing the bandwidth between 4 and 8 Hz.

2.1.2.3. Alpha Waves

The frequencies between 8 to 12 Hz characterise the Alpha waves which are most pronounced when a person is awake and relaxed. They are typically closing their eyes but are not sleeping. Alpha waves are strongly associated with a calm state of consciousness especially while meditating or daydreaming. But they often disappear when one opens it's eyes or is concentrating intensely on something. If this happens they turn into beta waves. The meaning of those waves can be clearly described as a description of a relaxed waking state or relaxed attention state.

2.1.2.4. Beta Waves

Beta waves are brainwave patterns characterized by a frequency range of 12 to 30 Hz, commonly associated with alertness, active mental activity, and concentration. These waves dominate during typical cognitive tasks, such as thinking, analyzing, and interacting with the environment. While beta activity reflects mental engagement and focus, heightened beta waves may also indicate states of anxiety or restlessness. In essence, beta waves are connected with alertness, concentration, and cognitive functioning. As already mentioned in the previous chapter, alpha waves can become beta waves if the state of relaxation is disrupted by concentration or awareness.

2.1.2.5. Gamma Waves

Gamma waves are the highest-frequency brainwaves, ranging from about 30 to 100 Hz, associated with intensive mental activity, learning, memory, and the processing of sensory input. These waves become particularly active when the brain engages in complex information processing, such as during cognitive challenges or moments of conscious recognition and experience. Due to their low amplitude, gamma waves are often more challenging to detect in EEG readings. Their primary significance lies in high-level cognitive functions, memory processing, consciousness, and sensory integration.

2.1.2.6. Sharp Waves

Sharp waves are single, rapid discharges observable in EEG readings, marked by steep, short-lived wave forms with high amplitude. Since this is an event happening during the time of brain activity there is no corresponding frequency band which can be stated for the sharp waves phenomenon. They are typically linked to pathological conditions, such as epilepsy, though they may also appear in healthy individuals during certain sleep phases or periods of quiet, inactive states. Sharp waves are thus indicative of either pathological states, as in epilepsy, or as features of deep sleep or inactivity.

2.1.2.7. Sharp-Slow Wave Complex

The sharp-slow wave complex consists of a sharp wave (a pointed, brief discharge described in the section before) followed immediately by a slow wave. These patterns are frequently observed during epileptic seizures and hold diagnostic relevance for epilepsy and other related neurological disorders. The sharp-slow wave complex is primarily significant for its association with epileptic activity or other pathological brain conditions.

2.1.2.8. Spike Waves

Spike waves are sequences of rapid peaks (spikes) followed by slower waves. They are characteristic of epileptic seizures, particularly in absence seizures (petit mal), and are typically high in amplitude and periodic, with a frequency of around 3 Hz. The presence of spike-wave complexes in EEG readings signifies abnormal electrical activity in the brain, commonly associated with epilepsy and especially indicative of absence seizures.

2.1.3. Procedure

To analyze the EEG data, the following processing steps were taken in the reference paper [1]:

1. The data were pre-filtered to remove the DC offset using a high-pass cutoff frequency of 0.16 Hz.
2. The FFT was computed on the data.
3. The EMOTIV Cortex API was employed to minimize artifacts within the EEG signal.
4. A Hann window function was applied to achieve accurate frequency resolution.
5. The continuous EEG time series data were transformed through FFT to specifically analyze the alpha frequency band power.
6. Power spectral density was calculated to determine the power level within each frequency band.
7. For each participant, the power of the alpha frequency band was calculated for each sensor and then averaged across all 14 channels.
8. To extract features, a 1-second window (128 data points) was defined, on which features like mean, peak, median, standard deviation, and kurtosis were calculated.
9. Labels were assigned by using the average results from the MDMT questionnaires as thresholds, with section 3 averaging 5.34 and section 4 averaging 4.64.
10. Finally, the Univariate Feature Selection method was applied to reduce the feature space by eliminating the median.

Following those steps the pre-processing will be replicated in the approach 3 section. For all those steps the paper [1] does not provide any code or basis which someone could built-up upon so the replication will be done using newly generated code.

2.1.3.1. Statistical tests

The Shapiro-Wilk test was applied to assess whether the population distribution of the sample was normal. For the results on test phases 3 and 4, the test indicated that the data were not normally distributed. That is why a Wilcoxon signed-rank test was executed next since given the lack of a normal distribution, a paired t-test was not suitable. Instead, a Wilcoxon signed-rank test was conducted, which is a non-parametric alternative for dependent samples that are not normally distributed. This test evaluates the null hypothesis that the central tendencies of the two dependent samples in the population are equivalent.

2.1.4. Applied Machine Learning Models

After pre-processing the data the paper introduces three major classical machine learning techniques which are applied to the pre-processed data. Binary classification into “low trust” and “high trust” was

performed based on features derived from the alpha frequency band power. For each participant, the alpha band power contribution of each sensor was calculated and then averaged across 14 channels to obtain representative values. To extract features, a sliding window of 1 second, comprising 128 data points, was applied. From the resulting data segments, the following statistical features were computed: mean, peak, median, standard deviation, and kurtosis. The classification labels ("low trust" or "high trust") were determined using the results of the MDMT questionnaires. Specifically, the average MDMT score served as a threshold to differentiate between the two categories. For each computed kurtosis value, the result indicated a slightly negative value, suggesting that the distribution was platykurtic, with lighter tails and a flatter peak compared to a normal distribution. Neglecting the median, the feature vectors for training consist of mean, peak, standard deviation, and kurtosis. The following three techniques where applied:

- Support Vector Machine
- k-Nearest neighbour classifier
- Random Forest

A support vector machine tries to search for a hyperplane in an n-dimensional space. In this case there are 14 dimension, which are the EEG channels of our measurements or the parameters of mean, peak, standard deviation, and kurtosis after pre-processing. The hyperplane discriminatively splits the data points in two classes and therefore solves the binary classification problem for labeling those data points as low or high trust. The goal is to maximize the distance between the support vectors of the data points and the corresponding found hyperplane.
The k-Nearest Neighbour (k-NN) classifier which is used in the paper [1] is a simple, non-parametric algorithm used for the binary trust level classification problem. It classifies data points based on the majority class of their k-nearest neighbors in the feature space, where k is a user-defined parameter. To choose which value for k should be set an elbow plot can be useful. While k-NN is intuitive and effective for small datasets, it can be computationally intensive and sensitive to irrelevant features or noisy data. Therefore its application to the EEG data might not yield the best results for the problem. This is also why in the section of 3.1 instead of the k-nearest neighbour algorithm, the k-means algorithm is used since it is more scaleable and robust. The applied Random Forest is an ensemble learning method that combines multiple decision trees to improve classification performance. Each of the trees is trained on a random subset of the EEG data and selects a random subset of features at each split. This reduces the risk for overfitting and increases diversity among the trees. The final prediction is determined by aggregating the outputs of all trees, typically using a majority vote for classification. To learn a Random Forest training each decision tree independently on its sample is required. Afterwards the best feature splits are selected to minimize the error. Through this ensemble learning it is possible to leverage the collective strengths of individual trees.

2.2. Alternative Methods and Approaches

Since traditional machine learning techniques might not be able to capture all spatio-temporal relationships in the data, more complex alternative methods are explored. Hereby especially the use of deeper neural networks or transformer architectures with attention mechanisms could leverage the information density of the EEG data to extract more complex patterns.

2.2.1. Application of EEG-GPT

The paper from Kim et. al "EEG-Generative Pretrained Transformer (GPT): Exploring Capabilities of Large Language Models for EEG Classification and Interpretation" [6] presents an innovative approach analyzing Electroencephalography (EEG) data by leveraging Large Language Model (LLM)s like GPT. The authors investigate the feasibility of LLMs in interpreting EEG data directly, with a focus on classifying cognitive states and understanding brain activity patterns. They propose EEG-GPT, a model adapted from LLM architectures, to handle the unique temporal and spatial features of EEG signals.

They first outline the complexities of EEG data, emphasizing its multi-dimensionality, with aspects like spatial (sensor locations on the scalp) and temporal (signal changes over time) dimensions. Traditional EEG analysis relies on feature extraction methods such as wavelet transforms and Fourier analysis, which are effective but require significant domain-specific knowledge. This was also done in the paper of [1] where the corresponding feature vectors were created and extracted through a FFT. The authors propose that adapting LLMs known for their prowess with sequential data and extensive contextual understanding, could be a transformative way to interpret EEG data with less manual feature engineering like FFTs.

To achieve this, EEG-GPT incorporates pre-processing stages designed specifically for EEG data. There the signal data is transformed into a format compatible with transformer-based models. They adapt these LLMs to handle the specific challenges in EEG data, such as noise, signal variability, and individual differences in EEG patterns. By training the model with large, annotated EEG datasets, the study aims to evaluate if LLMs can achieve comparable or even superior performance to traditional models in recognizing patterns associated with different cognitive states (e.g., attention, relaxation, and cognitive load). Here the first major disadvantage with regards to the available EEG data arises. Since there is data from 21 patients over a time period of around 20 minutes each, it is necessary to take into account that the dataset which is available might not be large enough to properly train a transformer-based model. This is subject to discussion but should be accounted for when working with complex architectures.

The report shows promising results in classifying cognitive states from EEG data, with performance metrics indicating that it can successfully capture key patterns and even subtle distinctions in brain states. Additionally, the model demonstrates robustness in handling noisy data and variability between subjects, showcasing the adaptability of LLMs to complex bio-signal data beyond traditional text.

One of the most compelling findings is that EEG-GPT allows for a certain level of interpretability, a key concern in neuroscience and medical applications. Unlike black-box approaches, the transformer's attention mechanisms provide insight into which parts of the EEG signals contribute most to the classification decisions. This is especially interesting when it comes to the target of building transparent AI systems in healthcare. This is important in this domain because doctors and medical personal need an understanding of the technical processes involved to be able to draw the right conclusions and therefore make the right decisions.

2.2.2. A neurological approach to classify trust through EEG signals

This papers is very similar to the paper which builds the baseline for this work [1]. [7] aims to calibrate trust quantitatively by analyzing the neural signal to address this perspective. In [7] a 'word elicitation' experiment is chosen. Here a list of words associated with trust and mistrust has been shown to the participants and the participants are expected to think about the trust/mistrust situation corresponding to those words. During this process, neural signals have been collected as EEG data, and the power spectrum density analysis is done in the frequency domain. In this study, classification analysis is performed with the two labels 'trust' and 'mistrust' on the power spectrum density of alpha, beta, and gamma frequency range. A machine learning model was developed and several machine learning classification algorithms were used on the model to find out the best algorithm for the dataset. An 72% average accuracy was achieved with the Decision Tree classification algorithm. Hereby the decision tree was one of the 5 most popular machine learning classification algorithms which were chosen:

- Logistic Regression (LR)
- Linear Discriminant analysis (LDA)
- KNearest Neighbour (KNN)
- Decision Tree (DT)
- Gaussian Naïve Bayes (GNB) algorithm

For this approach **no implementation details** are available.

2.2.3. A Classification Model for Sensing Human Trust in Machines

In [8] two approaches for measuring trust are introduced working on a EEG dataset. For this paper **no implementation details** are available. The first approach considers a general set of psycho-physiological features across all participants as the input variables and trains a classifier-based model for each participant, resulting in a trust-sensor model based on the general feature set. The second approach considers a customized feature set for each individual and trains a classifier-based model using that feature set, resulting in improved mean accuracy but at the expense of an increase in training time. This work represents the first use of real-time psycho-physiological measurements for the development of a human trust sensor.

2.2.4. ChronoNet

The introduced ChronoNet in [9] is a deep Recurrent Neural Network for abnormal EEG identification. There is an existing code implementation on https://github.com/Sharad24/Epileptic-Seizure-Detection which is not from the authors of the original paper.
One of the first steps in interpreting an EEG session or measurement file is to identify whether the brain activity is abnormal or not. To solve this task, the paper [9] proposes a RNN architecture termed **ChronoNet** which is designed to work efficiently with EEG data. ChronoNet is formed by stacking multiple 1D convolution layers followed by deep Gated Recurrent Unit (GRU) layers where each 1D convolution layer uses multiple filters of exponentially varying lengths. The stacked GRU layers are densely connected in a feed-forward manner. ChronoNet directly takes time-series EEG as input and learns meaningful representations of brain activity patterns.

2.2.5. Investigation of human trust by identifying stimulated brain regions

The research objective described in [10] investigates human trust through a word elicitation study by identifying stimulated brain regions on the specific brainwaves using an EEG. Similar to the baseline paper [1] the power spectrum is computed and a coherence analysis is executed. Results confirm that the frontal lobes in the alpha and beta waves have active connectivity in the trust behaviour of a human and that the temporal lobes in the gamma waves have active connectivity in the mistrust.

2.2.6. EEGNet: Convolutional Neural Network for EEG-based brain–computer interfaces

The paper [3] "EEGNet: a compact convolutional neural network for EEG-based brain–computer interfaces" by Lawhern et al. introduces EEGNet, a compact CNN designed for EEG-based brain-computer interfaces. EEGNet has proven to be effective in decoding movement-related EEG data and is particularly suitable for tasks with limited data. This approach could be adapted to trust-level classification as it is capable of extracting spatio-temporal features from EEG signals.
One main advantage is, that the EEGNet model is already pre-implemented in the **braindecode** API which can be found on gitlab https://github.com/braindecode/braindecode. This method is further described in section 3.2.3.

2.2.7. EEG Emotion Recognition Based on Multi-Channel CNNs

The study [11] "EEG Emotion Recognition Based on Multi-Channel Convolutional Neural Network" by Yang et al. leverages a multi-channel CNN for emotion recognition using EEG data. It draws advantages from the effectiveness of spatial convolutional layers for capturing correlations between EEG channels, which could also be useful for trust level classification, as emotions and trust may involve similar cognitive processing areas in the brain. The publication does not offer any reference for an implementation approach.

2.2.8. 4D Attention-based Neural Network for EEG Emotion Recognition

The paper [12] "4D Attention-based Neural Network for EEG Emotion Recognition" introduces an innovative method for classifying emotions using EEG signals. To capture the complex information inherent in EEG signals the paper proposes a four-dimensional attention-based neural network **4D-aNN** that leverages spatial, spectral, and temporal domains of EEG data. The raw EEG signals are transformed into four-dimensional representations encompassing spatial, spectral, and temporal information. This transformation enables the model to process data across different brain regions, frequency bands, and time sequences, capturing the multifaceted nature of EEG signals. The 4D-aNN employs spectral and spatial attention mechanisms to adaptively assign weights to various brain regions and frequency bands. These attention mechanisms allow the network to focus on the most informative parts of the EEG data. Mathematically, attention mechanisms can be represented as weight matrices that modulate the importance of input features. For instance, given an input feature matrix X and an attention weight matrix A , the attended output Y can be computed as:

$$Y = A \times X \tag{2.1}$$

where $\times$ denotes matrix multiplication. In the context of spectral and spatial attention, these weight matrices adjust the influence of different frequency bands and brain regions, respectively. The model

incorporates a CNN to process the spectral and spatial aspects of the 4D representations. CNNs are adept at capturing local patterns and hierarchies in data, making them suitable for analyzing the complex spatial and spectral features present in EEG signals. The convolution operation in a CNN is defined as:

$$(X * W)(i, j) = \sum_m \sum_n X(i + m, j + n) \cdot W(m, n) \tag{2.2}$$

where X is the input feature map, W is the filter (kernel), and (i, j) denotes the spatial position. To capture temporal dependencies within the EEG data, the authors integrate a temporal attention mechanism into a bidirectional Long-Short Term Memory (LSTM) network which are a type of RNN capable of learning long-term dependencies in sequential data. The temporal attention mechanism further refines the focus on relevant time steps, enhancing the model's ability to capture critical temporal features associated with different trust and emotion states. The model and it architecture are described in detail in the paper [12] but there is no implementation given.

2.2.9. Transformer for EEG-based Emotion Recognition

Transformers, which have been highly successful in various domains, are also applicable to EEG data. The paper [13] "Transformer for EEG-based Emotion Recognition" by Song et al. uses transformers for emotion recognition and shows that they are particularly effective for capturing complex patterns over long EEG sequences. Given the success of transformers with time-series data, this approach could be highly relevant to the problem of classifying trust levels. One drawback of transformer models is, that they require a large corpus of data whereas the given dataset described in 2.1.1 might be too small in size in order to fully train a transformer model. A major advantage is that for the paper [13] which introduces a transformer architecture for EEG data classification an implementation is publicaly available on github https://github.com/eeyhsong/EEG-Transformer.

2.2.10. Cross-Subject Transfer Learning for EEG-based Emotion Recognition

Since the EEG data is collected from multiple subjects, the paper [14] "Cross-Subject Transfer Learning for EEG-based Emotion Recognition" by Li et al. on transfer learning offers useful techniques for adapting models across subjects. Variability between individuals is a challenge in EEG analysis, and transfer learning methods may improve model robustness for generalizing across different subjects. Especially since participants vary in their behaviour because of e.g. different levels of experience on interacting with a novel robot, the type and level of trust which the robot can induce into the opposing human is different in every human participant. Emotions are highly subjective which results in a more complex challenge classifying those emotions based on EEG data. The paper offers interesting techniques and methods on how to transfer the learned paradigms across subjects. Unfortunately no implementation details or code is publicly available which increases the difficulty of recreating or applying the presented approaches.

2.2.11. Deep Learning with Convolutional Neural Networks for EEG Decoding

[15] The paper [15] "Deep Learning with Convolutional Neural Networks for EEG Decoding and Visualization" by Schirrmeister et al. explores the application of deep CNNs to decode and visualize movement-related information from EEG data. Traditionally, EEG analysis has relied on manual feature extraction methods, which are designed to detect specific spectral power modulations. This study investigates end-to-end learning approaches that eliminate the need for handcrafted features by

allowing CNNs to learn directly from raw EEG signals. Various CNN architectures are presented to decode information from EEG data. They include batch normalization and exponential linear units, to enhance decoding performance. Additionally, one major advancement is the introduction of a cropped training strategy. This involves training the network on multiple overlapping segments of the data to improve robustness and accuracy. Furthermore innovative visualization techniques are presented to interpret the features learned by the CNNs. This revealed that networks automatically identify and utilize spectral power information in the alpha, beta, and high gamma frequency bands.
Originally the models and techniques presented in the paper were used to classify and identify classes of movements through analysing the EEG measurements. Applying the models to the data given through [1] with the goal of classifying trust levels could therefore lead to promising results, depending on whether the network is able to extract the features needed to identify changes in the different frequency bands over the course of the training. This study demonstrates the potential of deep CNNs for end-to-end EEG decoding with the major benefit of skipping the manual feature engineering in the pre-processing steps. Another main advantage is that there is an already existing implementation of the presented models in the **braindecode** API. This open-source Python toolbox for decoding raw EEG brain data originated from the paper [15] and is developed through researchers from the University of Freiburg, Germany. It offers the possibilities to train custom and pre-implemented models which advances the use of deep learning models especially for raw EEG data analysis.
All the analyzed methods which were useful to check whether they are suitable or not for the trust level estimation are listed in the table 2.1 below:

Table 2.1.: Comparison of Alternative Methods and Approaches

Concept/Model	Advantages	Disadvantages
EEG-GPT (Kim et al., 2024) [6]	Reduces manual feature engineering, leverages transformers for temporal and spatial patterns, offers interpretability via attention mechanisms.	Requires large, annotated datasets, computationally intensive, limited research on trust-level tasks.
Traditional Machine Learning - A neurological approach (e.g., Firoz et al., 2022) [7]	Simpler implementation, effective with small datasets, established methods like FFT for feature extraction.	Requires significant domain expertise for feature engineering, limited adaptability to new patterns.
Classification Model for Sensing Human Trust in Machines [8]	Real-time measurement of human trust using Improved mean accuracy with the customized feature set approach for individual participants	Increased training time for the customized feature-based model, limited scalability
ChronoNet (Roy et al., 2018) [9]	Combines RNNs and CNNs for efficient temporal and spatial feature extraction, optimized for EEG time-series data.	Requires careful hyperparameter tuning, less interpretable than attention-based models.

Investigation of human trust by identifying stimulated brain regions [10]	Identifies specific brain regions involved in trust and mistrust. detailed insights into brainwave connectivity	trained on limited data
EEGNet (Lawhern et al., 2018) [3]	Compact architecture suitable for limited datasets, proven effectiveness for brain-computer interfaces.	Focused on movement-related data; adaptation needed for trust classification.
Attention Mechanisms (Xiao et al., 2021) [12]	Combines spatial, spectral, and temporal information from EEG signals. By employing spectral, spatial, and temporal attention mechanisms, the model adaptively focuses on the most informative brain regions, frequency bands, and time intervals.	Computationally intensive, may require large datasets for effective training. Generalizability to custom datasets or real-world scenarios may require further investigation. Lack of interpretability.
Cross-Subject Transfer Learning (Li et al., 2018) [14]	Adapts models across subjects, addressing individual variability, enhances robustness and generalization.	Additional complexity in training, performance depends on the quality of source-target mappings.
EEG Emotion Recognition with Multi-Channel CNNs [11]	parallel combined RNN and CNN improves accuracy. Captures spatial and temporal features.	High computational complexity, dependent on high quality resolution of EEG data
Transformer-Based Emotion Recognition (Wang et al., 2022) [13]	Effective for long EEG sequences, captures complex spatio-temporal patterns, adapts well to large datasets.	Limited application to trust, data and computational requirements.
Deep Learning with CNNs for EEG Decoding (Schirrmeister et al., 2017) [15]	Effective for EEG decoding and visualization, capable of extracting spatio-temporal features from EEG data, widely adopted EEGNet model.	Primarily designed for motor imagery tasks, may require adaptation for emotion or trust classification tasks.

3. Approach

The following pages include the methodical approach on how to process and examine the underlying problem statement. Firstly recreating the state of the art regarding the classification results of [1] was one major step in terms of further investigation what was already done through [1] and which potential problems could also arise through that. Over the cause different concepts and techniques will be explained and presented with the goal of classifying or inferring levels of trust.

3.1. Recreating the State of the Art

To analyze and get a first impression of the problem statement and the dataset which is used, the presented state of the art of the paper [1] is recreated. The dataset consists of the 21 csv-files which hold the measurements of the participants. Each file has a different length since the experiments where indeed conducted after the same scheme but the EEG measurement with a frequency of 128 Hz is providing a lot of new data points even if the experiments only differ in lengths of some seconds. There are some major questions which arise when looking at the data regarding potential pre-processing steps following the described pattern of the paper [1]:

1. Pre-filtered to remove the Direct Current (DC) offset using high-pass of 0.16 Hz.
2. Minimize artifacts within the EEG signal.
3. Apply Hanning window function
4. Compute FFT
5. Compute Power spectral density for each frequency band
6. Extract features mean, peak, median, standard deviation, and kurtosis
7. Assign labels based on results from the MDMT questionnaires
8. Univariate Feature Selection to reduce the feature space

The details description of the pre-processing steps can be referenced either in the paper [1] or in the section 2. The problem regarding the pre-processing steps involve the following questions and concerns:

- All steps which were taken to pre-process the data are indeed described to some level but they are not detailed enough to validate whether the chosen methods for the recreation of the results are completely similar to the ones chosen in the reference paper [1]. That way inaccuracies can already be introduced in the pre-processing step resulting in abnormalities or deviations in the recreated results.
- The dataset involves a details.csv file where all the participants questionnaire results after each phase of the experiment are listed. The questionnaire which can be referenced in the appendix A.1 includes 16 buzzwords which are related to the behaviour of the robot during the stage of the experiment. Here the discussion should be considered if the keyword accurately portray and

transmit the feeling and experience which the participant had while interacting with the robot. On the other hand more detailed questions after each stage would lead to longer waiting and answering times which could also limit and influence the accuracy regarding the trust relationship with the robot.

- To analyze the data of the EEG measurements properly it is necessary to match the labels which come from the results of the questionnaire with the corresponding time stamps of the EEG measurements. The sampling rate of each of the 14 sensors is 128 Hz which corresponds to 128 single measurements per second. Therefore a problem arises when looking at the level of detail regarding the time stamps mentioned in the details file. Here only hour and minute are mentioned for the features **start_clock** and **finish_clock**. As an example can serve the first entry of the details file which is characterizing the first phase of the experiment for the first participant with the **ID** equal to **1**. There the **start_clock** equals **13:46** and the **finish_clock** equals **13:50**. Compared to the csv-file which contains the raw data of the experiment one can see that the time stamps are not in the format **HH:MM** but rather in the form of **HH:MM:SS.MS** i.e. 13:47:24.798036. This deviation with regards to the accuracy of the time monitoring imposes a greater risk for including potential time stamps in our training dataset which might not be part of one of the phases of the experiment anymore due to the fact that there is a huge interval with potential values which could either lie in the time range of one of the phases or not. As an example the end time of phase one of participant one **13:50** could include all values from **13:50:00.000000** to **13:50:59.999999** which with a sampling rate of 128 Hz would mean that there are 59×128 values which could potentially be part of the phase or not. This would mean there can be 7552 values for each break in between phases where an uncertainty is evident whether they belong to the experiment or the break data points. Since there are 4 break points in between phases there could be $7552 \times 4 = 30,208$ sampling points for which no clear separation would be possible. Projecting this on the amount of sampling points included in csv-file 1 of participant one with 170,898 measurement points this would mean that 17.7% of the measurement entries can not be grouped in to **break** or **non-break** by comparing the time stamps of the raw data and the details-file.
- Regarding the time stamps mentioned in the details csv-file it needs to be pointed out that besides the lack of detail regarding the given time stamps also wrong time stamps are include in the file. Evident is this when comparing the first entry of the details csv-file with the first entry of the raw measurement data of participant one for phase one. The **start_clock** of the details csv-file shows a time point of **13:46** whereas the first entry of the raw data measurement has a time stamp of **13:47:24.798036**. This is especially a huge deviation when looking at the sampling rate of the measurements of 128 Hz. A deviation of **00:01:24.798036** would result in $84 \times 128 = 10,752$ measurement point for which no further information is given. Either the experiment was started later than given in the details csv-file or the first 10,752 values are missing. Either way the level of accuracy which results from providing the data in this way will introduce levels of uncertainty to the models and techniques applied to the data in the next steps. This level of uncertainty can only be analyzed when more context regarding the experiment is given and is therefore not possible as part of the research project.

All the mentioned inconsistencies and issues which were mentioned can not be resolved with the given information regarding experimental context and dataset information since the authors of [1] do not provide further detail on this. That is why the steps which were taken in order to recreate the

results include those mentioned uncertainties introduced through the issues mentioned above.

3.1.1. Pre-processing

The pre-processing steps described in section 3.1 were followed throughout the recreation of the results of the original paper [1].

3.1.1.1. Filtering

In the paper [1] it is described that the data was pre-filtered to remove the DC offset. A 0.16 Hz high-pass cutoff was used to achieve this. For the implementation of the pre-filtering the function 3.1.1.1 was created. It is used in the highpass filter function 3.1.1.1 to obtain the filtered data.

```
# Define a high-pass filter with 0.16 Hz cutoff
def butterworth_highpass(cutoff, fs, order=5):
    nyquist = 0.5 * fs
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='high', analog=False)
    return b, a
```

In the 3.1.1.1 function the cutoff frequency (**0.16 Hz**) and the sampling frequency **fs** are passed. The order is fixed to 5. From the **scipy** library the class function **butter** is loaded and provided with the order, the normal cutoff frequency and the desired filter type. The function creates a digital 5th-order butterworth highpass filter which is characterized through the output of the numerator **b** and denominator **a** polynomials.

```
# Apply high-pass filter to the EEG data
def highpass_filter(data, cutoff, fs, order=5):
    b, a = butterworth_highpass(cutoff, fs, order=order)
    return filtfilt(b, a, data)
```

In the 3.1.1.1 function the corresponding outputs of 3.1.1.1 are used in the imported **scipy** library class function **filtfilt** together with the data which needs to be processed. It takes the numerator and denominator coefficient vector and applies a linear digital filter once forward and once backwards to the data. As a result an array is obtained which still has the same shape as the corresponding data object which was used as input.

3.1.1.2. Minimize Artifacts

Artifacts can occur during the experiment due to movements of the eyes or bio-electric changes in the muscles or in the heart. Originally the authors of [1] used the EMOTIV Cortex API to minimize those artifacts in the EEG signal. EMOTIV is a manufacturer of EEG measurement instruments with the corresponding proximity software. Since this software is not available for recreating the results a **FastICA** was applied. **FastICA** is a technique to decompose a multivariate signal into additive, independent components. The underlying assumptions are that the source signals are statistically independent and non-Gaussian. This is also part of the reason why the **FastICA** dimensionality reduction meaningfully separates important components from noise and artifacts.

```
ica = FastICA(n_components=len(eeg_channels))
ica_data = ica.fit_transform(filtered_data)
# Convert the result back to a DataFrame
ica_df = pd.DataFrame(ica_data, columns=eeg_channels)
```

In this case the **sklearn.decomposition** class with its subclass **FastICA** was utilized through the **fit_transform** function to obtain the reduced results.

3.1.1.3. Windowing

To obtain a frequency resolution higher in quality a hann window function was applied to the cleaned output data from the previous steps [1]. The **hann** class of the scipy.signal.windows library was used to create the hann window of the length of the dataframe obtained from the **ICA**, see 3.1.1.4. The window is defined in the following way (see [16]):

$$w(n) = 0.5 - 0.5\cos\left(\frac{2\pi n}{M-1}\right), \quad 0 \leq n \leq M-1 \tag{3.1}$$

In this case it is used to smooth values and obtain a better frequency resolution.

3.1.1.4. Fast Fourier Transformation and Power Spectral Density

As far as to this point all pre-processing steps were conducted in the continuous time domain. To access the relevant frequency bands which were introduced in 2.1.2 it is necessary to convert the time series EEG data through the application of a FFT into the frequency domain. This way the involved frequencies in the signal can be assessed. Hereby the FFT is an algorithm which computes the Discrete Fourier Transformation (DFT) in a more efficient way. The formula for computing the DFT is the following:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}, \quad k = 0, 1, \ldots, N-1 \tag{3.2}$$

Hereby $X[k]$ represents the frequency component at index k, $e^{-j\frac{2\pi}{N}kn}$ is the complex exponential representing the basis functions of the Fourier transform and j is the imaginary unit. This way it is possible to transform a signal from time domain to the respective frequency domain. The function which is used for this is shown in the code snippet 3.1.1.4. The **numpy** class **fft.fftfreq** is used to extract the sample frequencies bin centers in cycles per unit of the sample spacing. Additionally the **fft** function is used to compute the one-dimensional DFT with the FFT algorithm to obtain the array of FFT-values. These are the Fourier Transform coefficients of the signal after applying a window function and represent the frequency-domain representation of the signal.

```
frequencies = {}
for channel in ica_df.columns:
    freqs, psd_values = compute_powerspectraldensity(np.array(ica_df[channel]), fs,
    window)
    power_in_bands = {band: bandpower(psd_values, freqs, band_range) for band,
    band_range in bands.items()}
    frequencies[channel] = power_in_bands
    # Convert the power in bands to a DataFrame for easier viewing
```

```
    # the power is given in micro Volts squared for each of the frequency bands
band_power_df = pd.DataFrame(frequencies).T
```

In the provided function, the **psd_values** variable represents the Power Spectral Density, also abbreviated with PSD, of the input signal, computed for the given fft_values. The psd_values calulation is done by computing the squared magnitude of the FFT coefficients, which corresponds to the power at each frequency component. Furthermore the values are normalized by the signal length n to ensure the power is appropriately scaled.

```
# Apply FFT to each channel and compute power spectral density
def compute_powerspectraldensity(data, fs, window):
    n = len(data)
    freq = np.fft.fftfreq(n, 1/fs)[:n//2]
    fft_values = fft(data * window)[:n//2]
    psd_values = np.abs(fft_values)**2 / n
    return freq, psd_values
```

The frequencies bin centers and the psd_values are returned by the function to calculate the bandpower in the next step. Here function 3.1.1.4 takes as an input the dictionary of the defined frequency bands as described in 2.1.2 and the frequencies bin centers and the psd_values. With a logical **AND** operation the band frequencies are summed up and the corresponding bandpower is calculated.

```
def bandpower(psd, freqs, band):
    band_freqs = np.logical_and(freqs >= band[0], freqs <= band[1])
    return np.sum(psd[band_freqs])
```

The corresponding bandpowers are then assigned to the frequencies of each channel. This results in a new dataframe which holds the processed EEG data of bandpowers in the frequency domain.

3.1.1.5. Manual Feature Extraction

The code 3.1.1.5 is part of a feature extraction pipeline that processes EEG data to derive meaningful metrics for further analysis and the corresponding machine learning applications mentioned in [1]. It uses the previously defined function **calculate_alpha_contribution** 3.1.1.5 to calculate specific features, focusing on the contribution and statistical characteristics of the alpha frequency band 2.1.2.3. The first step in this section involves calculating the average contribution of the alpha band using the **calculate_alpha_contribution** function. This function takes a dataframe containing the power of different frequency bands for each EEG channel and computes the total spectral power across all bands for each of the 14 channels. It then calculates the proportion of the alpha band power relative to this total and averages the results across all channels, yielding a single numerical value that summarizes the prominence of the alpha band activity across the EEG data. This average contribution is printed to provide an interpretable metric that reflects the cognitive state of the subject.

```
alpha_contribution_avg = calculate_alpha_contribution(band_power_df)
# Output the result
print(f"Average Alpha Band Contribution: {alpha_contribution_avg:.4f}")
alpha_features = extract_window_features(ica_df.values, fs, bands['Alpha'],
    n_channels)
```

```
# Convert the features to a DataFrame for better readability
feature_columns = ['Mean', 'Peak', 'Median', 'Std', 'Kurtosis']
alpha_features_df = pd.DataFrame(alpha_features, columns=feature_columns)
```

Following this, the code extracts additional features specific to the alpha band by calling the **extract_window_features** function A.3. This function operates on windowed segments of EEG data, dividing the input into one-second non-overlapping segments. For each window, it computes the power spectral density using the **compute_alpha_psd_window** function A.4, which applies Welch's method to estimate the power distribution over frequencies. The **scipy.signal** library import of the **welch** method was utilized. The alpha band power is isolated by summing the Power Spectral Density values corresponding to the 8–12 Hz range. Once the alpha band power is determined for all EEG channels within a window, statistical features such as the mean, peak, median, standard deviation, and kurtosis are calculated across the channels. These features encapsulate the distribution and variability of alpha activity. This provides a robust representation of the EEG signal regarding the alpha waves for each temporal segment. The extracted features are aggregated into an array and converted into a dataframe, and stored in a csv-file with appropriate column names: **Mean**, **Peak**, **Median**, **Standard Deviation** and **Kurtosis**. This feature extraction process is essential for transforming the raw EEG signals into structured data, capturing the characteristics that are need for classification or regression tasks.

```
def calculate_alpha_contribution(band_power_df, band_name='Alpha'):
    # Calculate the total power for each sensor by summing the power of all bands
    total_power = band_power_df.sum(axis=1)
    # Calculate the contribution of the Alpha band for each sensor
    alpha_contribution = band_power_df[band_name] / total_power
    # Average the Alpha band contribution across all 14 channels
    average_alpha_contribution = alpha_contribution.mean()
    return average_alpha_contribution
```

3.1.1.6. Label Assignment

The corresponding label's which are obtained through the questionnaire in between the different phases of the experiment need to be assigned to the corresponding row of the dataframe including the mean, median, peak standard deviation and kurtosis. The shown code in A.2 gives the implementation details on how to assign the labels of the details csv-file to the rows. To assign the labels to the right participant it is required to keep track of the currently processed file ID because this ID represents the number of the participant. The details csv-file serves as a lookup table to assign the labels accordingly. Furthermore it is necessary to keep track of the time while going through a dataframe labeling the rows. Therefore a time variable **sum_time** is defined which holds the current time point of the dataframe which needs to be labeled. Due to the uncertainties which are described in section 3.1 there is the possibility that some of the rows can be labeled wrongly. The nature of the dataframe which is obtained after the pre-processing and feature extraction steps is the following:

- each row corresponds to a time period of one second
- each row contains mean, median, peak standard deviation and kurtosis
- each row has an ID starting at zero and continuing till the ID equals the number of row which is the end of the file

- each dataframe corresponds to one full experiment with one participant
- each dataframe contains therefore all five phases of the experiment and the corresponding breaks after each phase

Important to point out is, that the break times in between the stages of the experiment should not be labeled. Those time periods are not considered as representative for the characteristics of the trust level which should be recognized. For labeling the phases of the experiment, the **start_clock** and **finish_clock** datetime objects can be used. Thus it is possible to directly label the corresponding intervals with the trust score of the participants questionnaire results. The resulting time difference will then be added to the **sum_time** variable and since the breaks for filling out the questionnaire are stated to be one minute long, there will be 60 seconds of time difference added at the end of the labeling process. It checks whether each row's timestamp falls within the predefined time ranges for Phase 3 or Phase 4 and assigns labels accordingly. The phases 3 and 4 are especially important because they characterize the change in trust level caused throught the planned behaviour of the robot during the experiment, see 2.1.1. If a timestamp is within the time range for Phase 3, the code evaluates the trust score from the details dataframe. If the trust score is labeled as "high," it assigns a label of 1 for binary classification or the actual trust score if a regression model is used. Hereby the introduced flag **label_regression** is used to decide whether the numerical values of the results of the questionnaire should be used or if the binary labels for the classification problem should be assigned. Similarly if the trust score is labeled as "low", the label is set to 0 for binary classification or the actual trust score for regression. For rows where the trust score is not clearly labeled which are the phases 1, 2 and 5, a custom threshold of 4.8 is applied to determine whether the label should be 1 (high) or 0 (low) for the binary classification problem. For regression models the numerical values of the questionnaires are assigned. A similar approach is applied to Phase 4. If the flag **flag_all_phases** indicating the inclusion of all phases is set to true, this labeling process is extended to Phases 1, 2, and 5 as well. The processed rows are saved to a csv-file with a specific name depending on whether the labeling includes all phases or only Phases 3 and 4, and whether the labels are intended for regression models or binary classification.
As important addition, two flags are implemented which change the behaviour of the provided code with regards to its execution. The **preprocessing_needed** flag is a binary flag which can be set to **True** or **False** depending on if pre-processing was already done or not. If the needed labeled csv files are already existing it is not needed and therefore the flag can be set to **False**. Thus all the pre-processing steps are skipped and the model creation and training is started immediately based on the existing csv files. Furthermore this concept can be also applied in a reversed fashion with the **create_models** flag. Hereby the model definition and training is not executed when the flag is set to **False**. Otherwise if it is set to **True** the model creation is executed normally.

3.1.2. Model Definition

As described in [1] the three techniques which were used for the classification task were the Support Vector Machine (SVM), k-Nearest Neighbour and Random Forests, see 2.1.4. The actual implementation of the models is not entirely specified, so that an own implementation was created and used to create and fit the models to the dataset. In the following section, the implementation and definition of a SVM and a k-means approach will be analyzed and discussed in detail. The Random Forest was not recreated due to the lack of time and in order to focus new methods. Furthermore the in paper [1] used k-nearest neighbour approach was substituted with a k-means clustering approach.
Firstly the labeled Alpha band feature data is loaded from the respective csv files. These features represent characteristics extracted from EEG data Alpha band (8–13 Hz frequency range). The files are iteratively loaded and saved as a dataframe, whereas the first column serves as the index of the rows. Rows with missing labels are removed with ensuring only fully labeled data points, and no points which characterize breaks during the experiment, are included for analysis. To keep track of participants, the identifiers from the filenames are extracted. All processed dataframes are concatenated into a single dataset creating a unified dataset encompassing all participants. The label column in the dataset represents the target variable, denoting distinct trust levels. For the binary classification problem only **high** and **low** exist as classes. For regression the labels are continuous numbers. The distribution of these labels is visualized using the **value_counts** function and a bar plot, providing an immediate overview of class imbalance which is encountered, 3.1. Here it becomes visible that based on the

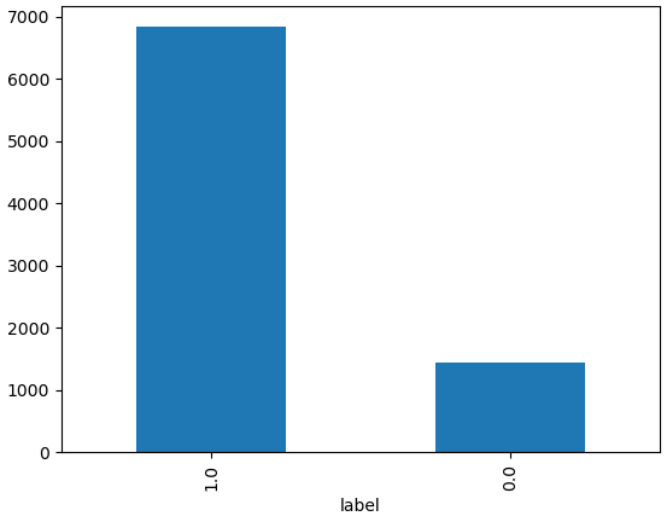


Figure 3.1.: Amount of feature vectors labeled with low (0.0) and high (1.0) trust showing the class imbalance of the dataset for the labeled phases 3 and 4 of the experiment.

threshold values which are presented in [1] and obtained from the results of the questionnaire a huge imbalance is occurring. The mean values of the results of the trust level after phase 3 and 4 as threshold lead to almost 7000 feature vectors being labeled as **high** trust and around 1500 feature vectors being labeled as **low** trust. An even higher imbalance occurs when labeling all the phases of the experiment with low and high trust, 3.2. Hereby a threshold value of **4.8** was used because it lies between median and mean of the trust scores of all participants of the corresponding phases of the experiment. The featuure vectors labeled with high trust accumulate to 17,500 and the feature vectors with low trust to

2600 instances. This imposes a strong bias which is having a huge influence on the training of models

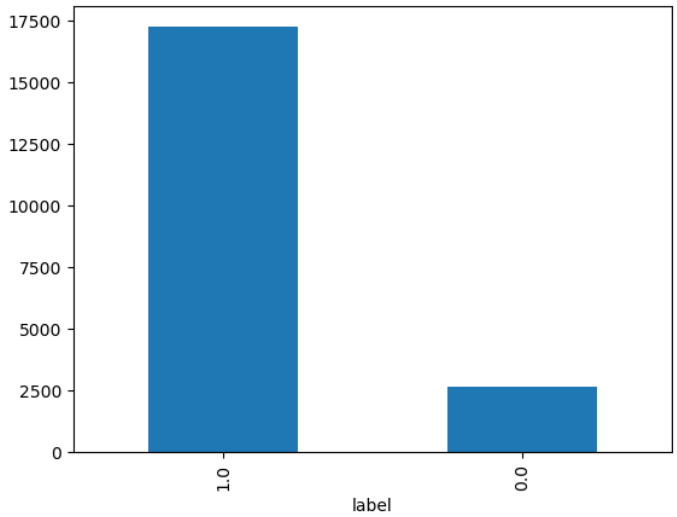


Figure 3.2.: Amount of feature vectors labeled with low (0.0) and high (1.0) trust showing the class imbalance of the dataset for the labeling of all phases of the experiment.

and their use afterwards.
To properly fit the data to the model all features **X** and labels **y** are separated from each other. The features include statistical measures such as mean, peak, standard deviation, and kurtosis. Initially also the median was calculated and introduced as a feature values. In the baseline paper [1] it was not used for the model creation and therefore is for the recreation of the results neglected as well. The labels **y** represent the trust levels that the model will aim to predict. A summary of class distributions per participant paints a similar picture as the value count of the dataset did, A.5. It is visible that the majority of the data for one participant is either completely consisting of **high** trust labels or that the **high** trust labels build the majority of the label distribution for one participant.
Addressing this inherent class imbalance is critical for reliable machine learning models. That is why the Synthetic Minority Oversampling Technique (SMOTE) is applied. It works by generating synthetic data points for the minority class, which in this case is the class of features which are labeled as **low** trust. It does so by interpolating between existing minority class instances and their nearest neighbours in the feature space. This effectively balances the class distribution without simply duplicating existing data, which could lead to overfitting. Before applying SMOTE, the data is split into training and test sets using the **sklearn.model_selection** function **train_test_split**. The split is stratified to ensure that the class proportions in the training and test sets are representative of the original dataset. Features are then scaled using a **MinMaxScaler** to normalize values between 0 and 1. Normalization is crucial for the application of SVMs because they are sensitive to feature magnitudes. This results from the characteristic of the algorithm because SVMs rely on distance-based computations to find the best fitting hyperplane in the n-dimensional space. The resulting training set after the augmentation process with SMOTE is balanced. The learning process is therefore more effective without bias towards the majority class.
The SVM model is instantiated with specific hyperparameters, including $C = 0.5$ and $class\_weight =$ "*balanced*". The parameter **C** controls the trade-off between maximizing the margin and minimizing classification error. A smaller **C** value results in a wider margin at the cost of some misclassifications,

while a larger **C** prioritizes correct classification of training points, potentially leading to overfitting. In this case the parameter value was chosen on the basis of the configuration of the specifications in [1] where the value was also set to 0.5. The *class_weight* = "*balanced*" parameter automatically adjusts the weights of the classes inversely proportional to their frequencies in the training set, further addressing imbalance.
As shortly described in 2.1 a SVM operates on the principle of finding the optimal hyperplane that separates data points from different classes. In this case the classes low and high trust level. The hyperplane is defined by maximizing the margin, which is the distance between the hyperplane and the nearest data points of each class. Those distances and the resulting vectors are referred to as **support vectors**. The support vectors are critical to determining the hyperplane and influence its orientation. For non-linearly separable data, SVMs employ kernel functions, such as linear, polynomial, or radial basis function, to transform the data into a higher-dimensional space where a linear separation becomes possible. In correspondence to the problem statement and the configuration which was introduced in [1] the radial basis function was used as a kernel. It is the default kernel defined in the **sklearn.svm.SVC** class but can also be changed to linear, polynomial or sigmoid kernels.
To reduce the risk of overfitting, Cross-validation is performed using Stratified K-Fold with five splits. This ensures that each fold maintains the class distribution. It provides a robust evaluation of the model's performance because for each fold, the training and test subsets are prepared, the model is trained on the training data, and predictions are made on the test data. The general procedure is the following, see [17]:

1. Shuffle the dataset randomly
2. Split the dataset into k groups
3. For each unique group:
    a Take the group as a hold out or test dataset
    b Take the remaining groups as a training dataset
    c Fit a model on the training set and evaluate it on the test set
    d Retain evaluation score and discard the model
4. Summarize ability of the model using the sample of model evaluation scores

This ensures that the variance and distribution of data points is selected in a way that the evaluation of the model is more robust an reliable in terms of estimating the model's ability to generalize on unseen data. Performance metrics, including accuracy, precision, and recall, are calculated for each fold and averaged at the end.
After specifying cross-validation, the SVM is trained on the entire training set, including synthetic samples obtained through SMOTE, and evaluated on the original test set. Several metrics are computed to assess the model's performance:

1. Balanced Accuracy: Accounts for imbalanced datasets by averaging recall scores for each class. Hereby the true positive rate (sensitivity) and the true negative rate (specificity) are used to calculate the metric 3.3:

$$\text{Balanced Accuracy} = \frac{\text{Sensitivity} + \text{Specificity}}{2} \tag{3.3}$$

2. Precision: Measures the proportion of correctly identified positive instances among all predicted positives. Here TP are the True Positives and FP are the False Positives.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.4}$$

3. Recall (Sensitivity): Measures the proportion of actual positives correctly identified. TP are again the True Positives and FN are the False Negatives.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.5}$$

4. F1 Score: Combines precision and recall into a single metric which is the harmonic mean of both metrics.

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.6}$$

5. Confusion Matrix: Provides detailed insights into classification performance by showing true positives, true negatives, false positives, and false negatives for each class. One example of a confusion matrix can be found in A.1.

The results, including the confusion matrix, are printed, offering a clear picture of how well the model performs on the test set. The confusion matrix is especially valuable for diagnosing specific errors, such as misclassifications between similar classes.
Additionally to the SVM a k-means clustering algorithm for the dataset was implemented. Hereby the the aim is to look for similar data patterns and to see if they correspond to the real classes of high trust and low trust instances. The k-means clustering algorithm provides a straight forward method when trying to replicate the results from the paper [1]. The paper [1] uses the k-nearest neighbour algorithm which is not studied here. Instead the k-means clustering algorithm was chosen because it is a powerful unsupervised learning approach to the problem. Labels are therefore technically not required. This give the advantages that it is more scalable and could also be used as a pre-processing step to reduce dimensionality. Furthermore the clusters formed by k-means can provide insights into the structure of the data, revealing underlying patterns or groupings which is more difficult with the k-nearest neighbour approach since it focuses more on classification based on proximity to the corresponding neighbours of a data point. The first step is data pre-processing using a **StandardScaler** from the **sklearn.preprocessing** module. This method standardizes features by scaling them to have a mean of 0 and a standard deviation of 1. It is a crucial step for the k-means algorithm because it is sensitive to the scale of the input features. The training dataset is fit to the scaler, transforming it into a standardized version. Similarly, the test set X_test is scaled based on the training set's parameters which ensures consistency and prevents data leakage during model evaluation. For the utilization of the k-means clustering the **sklearn.cluster** class **KMeans** was used. The parameters which need to be defined for initializing the k-means algorithm include the following:

- init="random": Random initialization for centroids, ensuring unbiased starting points.
- n_init=10: Specifies 10 initializations of the algorithm, selecting the solution with the lowest inertia
- max_iter=300: Sets maximum number of iterations for convergence
- random_state=42: Ensures reproducibility of results

To estimate which number of clusters fits the problem best the k-means algorithm is investigated with the elbow method. This involves iteratively fitting the k-means algorithm for cluster sizes ranging from 1 to 10 and computing the sum of squared errors (SSE) for each configuration, 3.1.2.

$$\text{SSE} = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2 \tag{3.7}$$

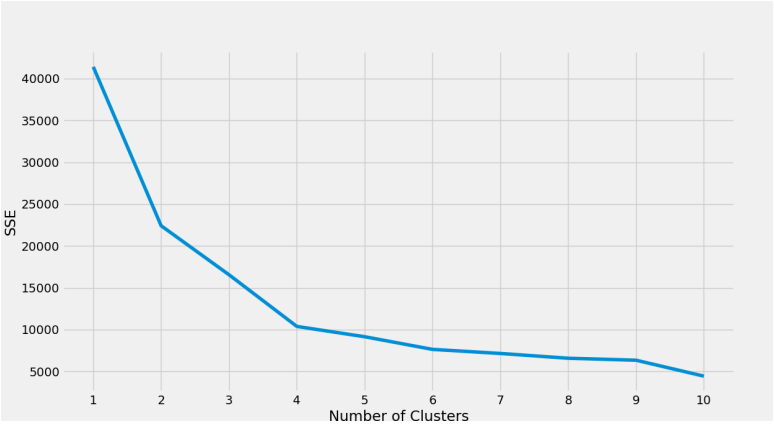


Figure 3.3.: Elbow plot of the the iteratively fitted k-means clustering to the dataset indicating that 4 clusters would be the optimal number of clusters for the problem.

SSE measures the compactness of clusters by summing squared distances between data points and their corresponding centroids. Hereby C_i represents the set of points in the i-th cluster and x is a data point in cluster C_i. μ_i is the centroid of cluster C_i and $\|x - \mu_i\|^2$ is the squared Euclidean distance between a data point x and its cluster centroid μ_i. The plot of SSE versus the number of clusters is also called the **elbow plot**, 3.3. It helps to identify the "elbow point" where additional clusters offer diminishing returns in SSE reduction.

Even though it is already inherited that the number of clusters should be two, given the nature and definition of the problem, handeling two classes with low and high trust level, analyzing the results of the elbow plot paint a different picture. Thus, the optimal number clusters is shown to be four which already imposes the question if the classification on two classes is the most accurate and goal driven approach.

```
sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(X_train_scaled_features)
    sse.append(kmeans.inertia_)

plt.style.use("fivethirtyeight")
plt.plot(range(1, 11), sse)
plt.xticks(range(1, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
```

For the primary analysis, k-means clustering is performed with 2 clusters because the original baseline of [1] also performs classification on the basis of two trust level classes. The model is trained on the scaled training features, and cluster assignments are predicted for both training and test data. To evaluate the performance of the obtained clustering a contingency matrix will be created with the help of the class of the **metrics.cluster** library. It computes a contingency table comparing true labels

(y_test) and predicted cluster assignments (y_prediction_test). Since clustering algorithms assign arbitrary labels, the Hungarian algorithm is employed to align cluster labels with ground truth. This results in maximizing the contingency table's diagonal counts. The results are evaluated using the following metrics and statistical measures:

- Accuracy: proportion of correctly classified instances, weighted to handle class imbalance.
- Precision, recall, and F1 score evaluate the balance between false positives and false negatives for weighted averages across classes, see 3.1.2
- Confusion matrix summarizes the distribution of correctly and incorrectly classified samples
- Silhouette score: quantifies the cohesion and separation of clusters, with higher values indicating better-defined clusters

Additionally misclassified instances are identified by comparing the true labels (y_test) to the realigned cluster labels (aligned_labels). A dataframe stores these instances, enabling further inspection and refinement of feature extraction or pre-processing steps. To further explore clustering behavior, especially for investigating the outcome of the elbow analysis a second k-means clustering with 4 clusters is done. While the process remains largely identical, the cross-tabulated results for four clusters provide additional insights into data separability and model behavior. This adds to the advancement of the understanding of the underlying high dimensionality of the data, to examine more advanced methods.
Studying the data with clustering algorithms allows for more detailed and advanced analyses that open up options for the future, beyond comparison with the low trust and high trust classes. For example, similar data patterns could be examined to identify their relationship with the ground truth of the given classes, as these are subjective because they are obtained from questionnaires filled in by hand by each participant. Furthermore those patterns which are generated through the clustering approaches could be used to compare participants directly or groups of participants who have common or uncommon characteristics. To leverage the full potential of the clustering approaches different data augmentation strategies could be applied. Furthermore the methods work on the pre-processed data where adjustments or the approach to try those techniques on the raw EEG data could also result in more advanced performance.

3.2. Advanced Models

Main part of the research project is to explore different deep learning techniques, which are suitable for the classification and potential regression task imposed by the question on how to estimate the trust level of humans interacting with a robot. To achieve this different approaches can be chosen based on the the complexity and their characteristics:

1. Start with EEGNet as a baseline, as it's lightweight and tailored for EEG data, [3].
2. Experiment with attention-based LSTM or GRU architectures if temporal dependencies seem significant.
3. Explore transformers if enough data is available, as those are effective for capturing long-term dependencies in sequential data.

Based on this, different model architectures can be further investigated:

- CNN: Apply 1D or 2D CNN layers to capture spatial relationships in the EEG signals
- RNN or LSTM: Include RNN layers (GRUs or LSTMs) to capture temporal dependencies within the EEG signal.
- Attention Mechanism: Use attention to emphasize critical time points or frequency bands that are more indicative of trust levels.
- Transformer-based Models: Explore transformer-based architectures, especially if the dataset is sufficiently large, as they can capture long-range dependencies.

3.2.1. Binary Classification Feed Forward Network

Before looking into more complex architectures for the classification task a simple feed-forward binary classification neural network is investigated. The neural network is constructed with PyTorch and is designed to predict a binary outcome based on extracted statistical features, such as mean, peak, standard deviation, and kurtosis, from the created csv files containing labeled data. Those files were created using the pre-processing steps described in section 3.1.1 which involves windowing, applying a FFT and calculating the statistical features. In this case the network is trained and evaluated on the already manipulated data coming from the pre-processing module. For the direct processing of the raw EEG data a more complex architecture would be needed. In this case the binary classification network described in the next lines should be compared directly to the usage of the traditional machine learning models introduced in the paper [1] and in section 3.1.
For the implementation of the network and the corresponding evaluation different libraries are used, including **PyTorch**, **scikit-learn**, and **pandas**. **PyTorch** is utilized for constructing and training the neural network, while **scikit-learn** provides preprocessing tools, performance metrics, and data balancing techniques. **Pandas** handles csv data import and data manipulation.

To create the architecture of the network a class named **TrustClassifier** is defined, which inherits from PyTorch's **nn** module. This custom neural network consists of three fully connected linear layers. The first layer maps the input features to 64 hidden units, the second reduces the dimensionality to 32, and the final layer produces a single output unit. Between the layers, the Rectified Linear Unit (ReLU) activation function is applied, which is defined by the equation:

$$f(x) = \max(0, x) \tag{3.8}$$

This activation introduces non-linearity to the model, allowing it to learn complex patterns. The output layer uses a sigmoid activation function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.9}$$

which maps the output to the range [0,1], making it suitable for binary classification tasks like the binary trust level classification.
Before feeding data to the neural network the csv files are loaded and saved as a pandas dataframe. Furthermore rows with missing labels are discarded. The concatenated dataframe contains the entire dataset. Afterwards it is split into features X and labels y. To ensure that each feature has a mean of zero and a standard deviation of one a **StandardScaler** is applied:

$$X\prime = \frac{X - \mu}{\sigma} \tag{3.10}$$

where μ is the mean and σ is the standard deviation of the feature.

Since PyTorch works with tensors all the label and feature dataframe is convert to such. This is especially important when utilizing the Graphical Processing Unit (GPU) acceleration on tensor operations during model training. Furthermore the data needs to be split into an 80% training set and a 20% test set. Therefore the pre-implemented **random_split** function is used. To streamline the training process, the data is loaded in batches of 16 samples using the PyTorch dataloader class. Hereby shuffling is enabled for the training set to reduce bias.
The training process is governed by the binary cross-entropy loss function where the true label and the predicted probability are compared with each other. It penalizes incorrect predictions more heavily when the confidence in those incorrect predictions is very high. The exact definition of the binary-cross entropy loss is described in equation 3.16. The optimizer which is employed is Adam, a variant of stochastic gradient descent that adapts learning rates individually for each parameter, computed as:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{3.11}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{3.12}$$

$$\theta_t = \theta_{t-1} - \frac{\alpha m_t}{\sqrt{v_t} + \epsilon} \tag{3.13}$$

where g_t is the gradient at time step t, α is the LR, and β_1, β_2 are decay rates. The resulting update rule is dscribed through equation 3.13. For the training a loop iterates for a certain number of specified epochs. During each epoch, the model parameters are updated using backpropagation. This involves the computation of the gradients, followed by the updating of the weights. During the training process the training loss, accuracy, precision, recall, and F1-score are tracked across epochs.
The goal of the evaluation phase is to test the model on unseen data. Therefore the test set which was obtained through the split of the whole dataset is used. All the inferred predictions for the test data are gathered and compared to the true labels. Based on this comparison performance metrics can be computed for assessing the model's generalizability and robustness, This includes the accuracy, precision, recall, and F1-score. Additionally, balanced accuracy is calculated to mitigate the effect of class imbalance which is evident in the used dataset 3.3. The balanced accuracy unifies the aim to accurately detect positives (high trust) and negatives (low trust) instances. The computed training metrics can also be visualized across epochs, showing if the model is improving it's performance and convergence over the course of the training or not. Based on these plots it can be accessed whether the model overfits, underfits, or generalizes well.

3.2.2. Recurrent Neural Networks

To distinguish between states of low and high trust a neural network-based classification system for the corresponding EEG data can be created in the form of a recurrent neural network. The primary goal of this is to leverage the advantages of this deep learning technique to create a robust model capable of capturing intricate patterns within the data, which can then be used to infer trust levels. The structure of the model and the preprocessing pipeline need to reflect the complexities involved in working with EEG signals, which are inherently noisy and susceptible to variability between individuals. Furthermore the underlying class imbalance needs to be addressed which can be done through data augmentation techniques like SMOTE. Through employing a multi-layer perceptron architecture enhanced with attention mechanisms and residual connections different experiments and test trails are investigated.
The model architecture is encapsulated within the **EEGTrustClassifier** class, which is derived from the **nn** module of pytorch. Therefore it is possible to establish it as a fully customizable neural network. The input to this classifier consists of four features — Mean, Peak, Std, and Kurtosis — extracted from the EEG data through the pre-processing steps pointed out in chapter 3.1. The choice of these features can be explained with the necessity of capturing EEG signal characteristics like variations in spectral power, waveform irregularities, and amplitude fluctuations. These features are passed through an input layer consisting of a fully connected linear transformation that maps the four input dimensions to a higher-dimensional space of 128 units. This transformation is followed by batch normalization, an essential component that standardizes the activations within the network, improving convergence and mitigating issues related to internal covariate shift. The use of the SiLU (Sigmoid Linear Unit) activation function further enhances this layer by introducing non-linearity, thereby enabling the model to capture complex, non-linear relationships in the data. The core of the classifier is composed of six consecutive residual blocks, each comprising a fully connected layer, batch normalization, dropout, and a residual connection. It is inspired from ResNet architectures, which are efficient in mitigating the vanishing gradient problem. In the absence of residual connections, deep neural networks may suffer from degradation, whereby additional layers contribute to diminishing accuracy. Mathematically, residual connections introduce identity mappings such that the output of each block can be expressed as:

$$x_{l+1} = F(x_l, W_l) + x_l \tag{3.14}$$

where $F(x_l, W_l)$ represents the output of the layer's transformation, x_l denotes the input to the layer, and W_l represents the weight matrix. This formulation ensures that even if the learned transformation approaches zero, the identity path preserves the information flow, allowing gradients to propagate without attenuation. Dropout is incorporated with a rate of 0.4, serving as a regularization mechanism that randomly deactivates neurons during training to prevent overfitting. This stochastic behavior forces the network to develop redundant representations, enhancing generalization to unseen data. An important aspect of this architecture is the inclusion of an attention mechanism following the residual blocks. Attention mechanisms are widely employed in neural networks to dynamically focus on the most relevant features while suppressing less informative ones. In this context, the attention layer computes a set of weights through a linear transformation of the output feature map, followed by a softmax operation to ensure normalization. The resultant attention weights modulate the feature map, amplifying important features that may correspond to discriminative EEG patterns linked to trust levels. Formally, attention can be described by:

$$\alpha_i = \frac{\exp(e_i)}{\sum_j \exp(e_j)} \tag{3.15}$$

where e_i denotes the raw output of the attention layer. This mechanism enhances the interpretability of the network by allowing it to selectively emphasize features that drive the classification decisions. The final stage of the network consists of two additional fully connected layers that progressively reduce the dimensionality from 128 to 64 and then to 32, ultimately feeding into a single output neuron. This neuron features a sigmoid activation function to produce a probability score indicating the likelihood of high trust, thereby framing the task as a binary classification problem. The binary cross-entropy loss function, also abbreviated as BCELoss, is used to quantify the discrepancy between predicted and true labels, defined as, 3.16:

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{3.16}$$

where y_i and $\hat{y}_i$ represent the true and predicted labels for the i-th sample, respectively. Before feeding the data to the network different data preprocessing steps need to be followed. It begins with loading the needed labeled EEG data from created csv-files which result from the pre-processing steps, 3.1.1. Hereby it needs to be pointed out the similarly as in the introduced models in the paper [1] and in the binary classification feed-forward network the csv files with the statistical parameter are used. Those files are created through the pre-processing steps described in 3.1.1. Meaning that this network processes an already manipulated state of the data. Processing the raw EEG data is possible with more complex architectures like the EEGNet described in section 3.2.3.1. Afterwards the feature vectors which do not contain labels are filtered out to ensure the integrity of the dataset. The already addressed issue of the class imbalance is an issue for the EEG data classification. It is addressed through SMOTE in a similar way as it is applied in section 3.1.2. It generates synthetic samples by interpolating between existing data points within the minority class, mitigating the bias that could arise from an overrepresentation of one class, in this case the high trust instances. This equalizes the class distribution, fostering a balanced learning environment which is important when training and evaluating the RNN. Afterwards the features are standardized using sklearn's StandardScaler, ensuring that all input dimensions share a common scale. Standardization is essential for neural networks as it prevents the dominance of features with larger magnitudes, facilitating smoother gradient updates.
The training process is defined through mini-batch stochastic gradient descent strategy in combination with the Adam optimizer. This optimizer is known for its adaptive LR properties. A cosine annealing LR scheduler is employed to dynamically adjust the LR over epochs, reflecting a decaying sinusoidal pattern that promotes convergence by periodically lowering the LR to small values. This way it is possible to escape saddle points, contributing to improved generalization. Performance metrics, including accuracy, precision, recall, and F1-score, are computed at each epoch, enabling a comprehensive assessment of model performance. To illustrate those metrics over the course of training they are visualized through tracking them throughout the training process and plotting them afterwards. This way it is possible to see trends and possible hurdles when it comes to optimizing the models parameters. Evaluation is executed on a hold-out test set which provides insights into the model's generalization capabilities. By stratifying the train-test split, class proportions are preserved, minimizing the otherwise occurring sampling bias. The balanced accuracy score, precision, recall, and F1-score provide a holistic evaluation, capturing not only the overall correctness but also the model's sensitivity to minority class predictions.

3.2.3. EEGNet

One of the architectures which is studied in this research project is the EEGNet introduced by [3]. It is a compact CNN architecture developed to analyze EEG data for brain-computer interaction applications. The design of EEGNet is centered around efficiently capturing both temporal and spatial features of EEG signals while maintaining a lightweight model with relatively few trainable parameters, making it ideal for applications where data is limited. Furthermore it can produce neurophysiologically interpretable features which can be analyzed later on.

3.2.3.1. Architecture

The general architecture of the network is separated into three main blocks:

- Block 1: perform two convolutional steps in sequence. First fit F1 2D convolutional filters with the filter length chosen to be half the sampling rate.
- Block 2: use a Separable Convolution, which is a Depthwise Convolution followed by F2 (1, 1) Pointwise Convolutions.
- Classification block: features are passed directly to a softmax classification with N units, being the number of classes in the data

Those main blocks can be also visualized through the schema illustrated in 3.4. Each of the three primary stages, is tailored to extract different aspects of the EEG signals. The first stage of EEGNet

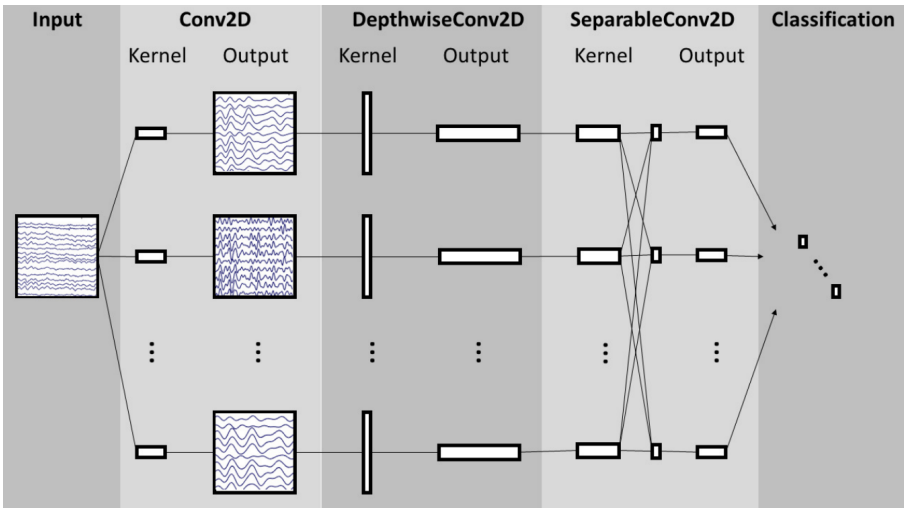


Figure 3.4.: The schema describes the architecture of the EEGNet which was introduced in [3].

applies a convolution across the temporal domain of the EEG signals. This layer is designed to extract frequency-based features from the raw EEG time series. By using 1D convolutions along the time axis, the network learns to recognize patterns that correspond to oscillatory activity or event-related potentials. This layer can be interpreted as a set of temporal filters that enhance relevant frequency components while suppressing noise. In the second stage a depthwise convolution is introduced, which is applied across the 14 EEG channels. Unlike standard convolutions that mix information from all input channels, depthwise convolutions operate independently on each channel. This allows the network to learn spatial patterns specific to each EEG channel, preserving the spatial information essential for understanding brain activity across different regions. The depthwise convolution layer captures spatial relationships between electrodes without significantly increasing computational complexity. The third

stage consists of a separable convolution, which combines depthwise and pointwise convolutions. This layer first performs a depthwise convolution to extract spatial patterns, followed by a 1x1, pointwise convolution that integrates information across different channels. Separable convolutions reduce the number of parameters compared to traditional convolutions, improving computational efficiency while maintaining the model's ability to capture complex spatial-temporal patterns in the data. Additionally the separable convolutions decouple the relationship within and across feature maps. This is done by first learning a kernel summarizing each feature map individually and then optimally merging the outputs afterwards [3]. After each convolutional layer, batch normalization is applied to stabilize learning and accelerate convergence. The batch normalization layer is important for the backward propagation process of the training process. This layer will be included in the network after the activation function of a hidden layer so that the batch normalization layer is able to normalize the activations of each previous layer. This also introduces two new learnable parameters into our network which are called beta β and γ. This means that for each hidden layer one batch normalization layer will be added whereby two parameters are additionally added to the training objective of the network. Activation functions such as Exponential Linear Units or ReLU introduce non-linearity, allowing the network to learn complex mappings between EEG signals and target outputs. Furthermore pooling layers are incorporated to downsample the feature maps, reducing their dimensionality and ensuring that the network focuses on the most salient features. Dropout is applied to prevent overfitting by randomly deactivating a subset of neurons during training, which improves the model's generalization capabilities. The final stages of EEGNet consist of fully connected dense layers that aggregate the learned features for classification. Depending on the brain-computer interface (BCI) task, the output layer uses a softmax or sigmoid activation to generate class probabilities, allowing the network to perform different tasks such as classification and regression of corresponding stages of the experiment with regards to measuring trust.

EEGNet's architecture is highly parameter-efficient, which is critical in the sense that EEG datasets

Block	Layer	# filters	size	# params	Output	Activation	Options
1	Input				(C, T)		
	Reshape				(1, C, T)		
	Conv2D	F_1	(1, 64)	$64 * F_1$	(F_1, C, T)	Linear	mode = same
	BatchNorm			$2 * F_1$	(F_1, C, T)		
	DepthwiseConv2D	D * F_1	(C, 1)	$C * D * F_1$	(D * F_1, 1, T)	Linear	mode = valid, depth = D, max norm = 1
	BatchNorm			$2 * D * F_1$	(D * F_1, 1, T)		
	Activation				(D * F_1, 1, T)	ELU	
	AveragePool2D		(1, 4)		(D * F_1, 1, T // 4)		
	Dropout*				(D * F_1, 1, T // 4)		$p = 0.25$ or $p = 0.5$
2	SeparableConv2D	F_2	(1, 16)	$16 * D * F_1 + F_2 * (D * F_1)$	(F_2, 1, T // 4)	Linear	mode = same
	BatchNorm			$2 * F_2$	(F_2, 1, T // 4)		
	Activation				(F_2, 1, T // 4)	ELU	
	AveragePool2D		(1, 8)		(F_2, 1, T // 32)		
	Dropout*				(F_2, 1, T // 32)		$p = 0.25$ or $p = 0.5$
	Flatten				(F_2 * (T // 32))		
Classifier	Dense	N * (F_2 * T // 32)			N	Softmax	max norm = 0.25

Figure 3.5.: The table describes the architecture of the EEGNet which was introduced in [3].

often suffer from small sizes and high inter-subject variability. Additionally the dataset used here is highly imbalanced which is also introducing uncertainties. The use of depthwise and separable convolutions significantly reduces the number of trainable parameters, making the network faster to train and less prone to overfitting. The architecture is modular and can be adapted to different input sizes, number of EEG channels and EEG recording configurations. This adaptability makes EEGNet suitable for various tasks and challenges, ensuring broad applicability across different research domains and clinical settings. In the paper [3] EEGNet has demonstrated strong performance across diverse BCI paradigms. For example, in P300-based paradigms, EEGNet effectively extracts event-related

potentials that appear as peaks in the EEG signal in response to target stimuli. In motor imagery tasks, the network identifies oscillatory changes in sensorimotor rhythms associated with imagined movements. EEGNet's ability to generalize across these paradigms without significant architectural modifications highlights its robustness and versatility making it also suitable for classifying trust levels.

3.2.3.2. Implementation

For the implementation, importing essential libraries required for processing EEG data and building deep learning models is needed. numpy and pandas handle numerical computations and data manipulation, while torch provides the deep learning framework necessary to define, train, and evaluate neural networks. skorch bridges the gap between PyTorch and scikit-learn, enabling seamless integration and model training. matplotlib facilitates data visualization, and sklearn provides key metrics for model evaluation. The core library which is used in this case is **braindecode**, [15]. It supplies pre-built EEG-specific models, such as EEGNetv4, optimized for classifying EEG signals.

Furthermore a custom function to compute the balanced accuracy is implemented. This metric averages the recall for each class and is particularly useful when dealing with imbalanced datasets. Mathematically, balanced accuracy is described in equation 3.3. where the sensitivity and the specificity are processed together so that the balanced accuracy is able to account for positive and negative predictions. The function takes the model's predictions and true labels, flattens them to ensure compatibility, and computes this score using **sklearn's** balanced_accuracy_score function.
Another key function is the create_windows function which is needed for segmenting continuous EEG data into overlapping or non-overlapping windows. EEG data is typically recorded at high temporal resolution, and windowing allows the model to analyze time segments rather than individual time points. In this case one time window of one second normally contains 128 points because the measurement frequency is set to 128 Hz. The function reshapes the data into a three-dimensional array, with dimensions representing the number of samples, EEG channels, and time points. This transformation is crucial for training CNNs, as it provides the necessary spatial and temporal context. The labels are adjusted to match the window size, ensuring each segment of data is associated with the correct label of either high or low trust or the numerical trust value for the regression task.

The main data processing loop iterates through EEG recordings stored as csv files reading each file using pandas and then loads corresponding label files, which annotate the data with class labels. In this case the raw data with the corresponding 14 EEG channels is loaded since the main objective of using the **braindecode** models is to utilize them to extract information from the raw data skipping the feature extraction and preprocessing step described in 3.1. Labels, representing different trust levels, are repeated for 128 consecutive rows, corresponding to one second of EEG data at a sampling rate of 128 Hz. These labels are applied to the raw EEG data including discarding any rows without labels. The EEG channels, such as AF3, F7, and F3, are extracted to create the feature matrix which is used for training. In this case all the features correspond to one EEG channel whereas for the conventional machine learning methods the pre-processed feature vectors include the statistical parameters of a window, see 3.1. Once data from all participants is pre-processed it is concatenated to form a unified array. This array, representing all EEG data, is split into training, validation, and test sets using stratified sampling, ensuring that each class is proportionally represented across the splits. Typically, 10% of the data is reserved for testing, while 20% of the remaining training data is allocated for validation. Stratification helps to maintain consistent class distribution, preventing data leakage and ensuring reliable model evaluation.

The EEG data is reshaped into the format required by **PyTorch**, with dimensions corresponding to:

$$(\textit{number of samples, number of channels, number of time points}) \tag{3.17}$$

Furthermore the arrays are converted into tensors, which are essential for GPU acceleration during training. Labels are also converted to long tensors to ensure compatibility with the cross-entropy function used as a loss function. To address class imbalance, the script computes the frequency of each class in the training data. Class weights are determined by taking the inverse of the class frequencies:

$$w_i = \frac{1}{n_i} \tag{3.18}$$

where w_i is the weight for class i and n_i is the number of samples in class i. These weights are passed to the loss function, ensuring that underrepresented classes contribute more to the overall loss, thereby mitigating bias towards majority classes.
The model is instantiated using the EEGNetv4 from **braindecode** for EEG data classification. EEGNet leverages depthwise and separable convolutions, reducing the number of parameters while maintaining high classification accuracy. For further explanaitions see 3.2.3.1. The architecture is defined by:

$$Y = X * K_1 * K_2 \tag{3.19}$$

where X is the input EEG signal, K_1 is the depthwise convolution kernel, and K_2 is the pointwise convolution kernel. This two-step convolution process allows the model to extract spatial and temporal features efficiently.

3.2.3.3. Loss Function

A cross-entropy loss function is employed to optimize the model. If trust levels are categorical which is the case with regard to the classification problem of classifying high and low trust level instances the Cross-entropy loss is used. It quantifies the difference between predicted and true class distributions, calculated as:

$$\mathcal{L} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \tag{3.20}$$

where y_i is the true label (one-hot encoded), and $\hat{y}_i$ is the predicted probability for class i. The inclusion of class weights modifies this formula to:

$$\mathcal{L} = -\sum_{i=1}^{C} w_i y_i \log(\hat{y}_i) \tag{3.21}$$

ensuring that misclassifications of underrepresented classes incur higher penalties.

3.2.3.4. Optimizer

Training is conducted using the Adam optimizer, a variant of Stochastic Gradient Descent (SGD) that adapts learning rates based on first and second moments of gradients. The Adam update rule is defined by equation 3.13 where m_t and v_t represent the moving averages of gradients and squared gradients, g_t is the current gradient, and α is the LR. Adam is used because it can accelerate convergence while maintaining stability. That is the reason why it is especially suitable for deeper and more complex neural networks.

The training process tracks balanced accuracy through an **EpochScoring** callback, which evaluates the model after each epoch. Additionally, a cosine annealing LR scheduler gradually reduces the LR following a cosine curve:

$$\alpha_t = \alpha_{\min} + \frac{1}{2}(\alpha_{\max} - \alpha_{\min}) \left(1 + \cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right) \tag{3.22}$$

where α_t is the LR at epoch t, and T_{cur} represents the current epoch. This scheduling prevents the model from stagnating in local minima. It reduces the LR if the validation loss of the training process does not improve for a certain number of epochs. When no improvement over a period of n epochs occurs the scheduler will reduce the LR by the factor x.
The classifier is trained for 200 epochs with a batch size of 16, after which the model is saved for future inference. The number of epochs and all other hyperparameter can be adjusted with the goal of finding the right combination of hyperparameters yielding the best results. On the test set, predictions are generated and compared against true labels using a confusion matrix, visualized through **braindecode**'s plotting utilities. The overall accuracy and balanced accuracy are computed to assess the model's performance on unseen data. Training performance is visualized through plots tracking loss, accuracy, and misclassification rates over epochs. These plots provide insights into the model's convergence and highlight potential overfitting or underfitting issues. All the metrics for the different combinations of hyperparamters can be seen in section 4 where the results and consequences are discussed.

4. Results

In this section all different concepts and techniques which were implemented are analyzed based on their yielded results. First, the results obtained from the models which were used in the referenced paper [1] are presented. In the next step the custom models working on the statistical features yield after the pre-processing steps, are analyzed in more detail. At the end a detailed description of the obtained results from the implementation of EEGNet for the trust level estimation is provided. Hereby two main approaches need to be distinguished: the usage of the model in the context of the classification task and the usage as a regressor for the trust level estimation.
All presented training runs were executed on a MacBook M2 Pro with 32Gb Random-Access Memory (RAM). There was no GPU used for training purposes but since more complex models like the EEGNet has a higher number of trainable parameters, computational complexity increases further. Therefore a GPU or a GPU-cluster is highly recommended for further investigations or the use of more complex models.

4.1. Results State of the Art

As mentioned in the chapter 3.1 two of the three models created in the baseline paper [1] were recreated. The corresponding implementation yielded the following results for the first evaluation on the test set after fitting the models on the training data correspondingly 4.1:
What is observable is that the accuracy of the SVM appears to be satisfying. This is the case because

Table 4.1.: First Fit of the k-means clustering and SVM

SVM Results	
Accuracy SVM Test:	0.82374
Precision SVM Test:	0.67855
Recall SVM Test:	0.8237
KMeans Clustering Results	
Silhouette Score:	0.938616
Accuracy k-means:	0.8224
Precision k-means:	0.7067
Recall k-means:	0.8224
F1 Score k-means:	0.7459
Inertia k-means:	23697.108
Cluster Centers:	[[-4.995e-02, -4.9282e-02, -4.862e-02, -3.907e-02, 1.1931e-03], [1.211e+01, 1.195e+01, 1.179e+01, 9.477e+00, -2.89e-01]]
Number of Iterations:	13

the accuracy metric does not account for the imbalance in the dataset. The relatively high precision

value of 0.68 indicates that the model makes fewer false positive predictions which means that less instances which have a true label of **low** trust are predicted as **high** trust. Since the best achievable value is 1.0 there is still quite some improvement needed to further reduce the false positive predictions. A high recall means that the model is identifying most of the true positives, so **high** trust instances labeled as such, and true negatives, so **low** trust instances labeled as such, in the dataset.
In the reference paper [1] additionally to the SVM a Random Forest and K-Nearest Neighbors approach is selected. Here the clustering approach which is utilized in addition to the SVM is the k-means clustering that was not implemented or tested by authors of [1]. For the k-means clustering the silhouette score indicates a very good clustering since this metric is defined as how similar an object is to its own cluster compared to other clusters. It ranges from -1 to 1, where:

- +1 indicates that the sample is well-clustered and therefore far from neighboring clusters.
- 0 means the sample is on the border between two clusters.
- -1 indicates that the sample is likely in the wrong cluster.

The definition of the metric for one point is the following:

$$s(i) = \frac{dist_{inter}(i) - dist_{intra}(i)}{\max(dist_{intra}(i), dist_{inter}(i))} \tag{4.1}$$

Hereby $dist_{intra(i)}$ is the average distance from point i to all other points in the same cluster and $dist_{inter(i)}$ is the average distance from point i to points in the nearest neighboring cluster. To compute the score for the whole test dataset it is necessary to sum up all point-wise scores and to divide it through the number of points N

$$S = \frac{1}{N} \sum_{i=1}^{N} s(i) \tag{4.2}$$

Also the accuracy, recall and precision values for the fitted model seem to be good. Still, the accuracy metric does not take into account the imbalance of the dataset. The F1-score is also decent with a value of 0.746 it is still considered good since the highest possible value of it is 1.0 which indicates a perfect precision and recall. The lowest possible value is 0, if precision and recall are zero.

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.3}$$

Another important metric for the assessment of the quality of a clustering is the inertia. It is defined as the sum of squared distances of samples to their closest cluster center. That way it is possible to measure the internal coherence of the clusters. The value of 23697.108 indicates that data points are far from their respective cluster centers. This suggests that the created clusters are not very well-defined. The aim of the k-means algorithm is to lower the inertia as far as possible, meaning that the inertia metric serves as a objective function for the whole algorithm, [18]. That way it needs be argued that the obtained clustering is despite the good scores in accuracy, recall, precision and F1-score not accurate. The inertia is signalizing that the single points contained in the clusters are far to different in their characteristics leading to higher inertia values. In addition for this model creation process no augmentation for reducing the class imbalance was applied which is also leading to distorted results regarding the evaluation metrics.
In the next iteration, the SMOTE augmentation approach was applied for the model generation. Furthermore a stratified k-fold with five folds was applied to further reduce the risk of overfitting and

Table 4.2.: Metrics obtained of the SVM with data augmentation

SVM Training:	
Average Training Accuracy:	0.546
Average Training Precision:	0.635
Average Training Recall:	0.221
SVM Test Classification Results:	
Accuracy SVM Test:	0.547
Precision SVM Test:	0.778
Recall SVM Test:	0.317
F1-score SVM Test:	0.318

the influence of the class imbalance. The following results were obtained:

The table 4.2 presents the metrics obtained from the implementation of SVM with the SMOTE technique both for the training and test sets. The following confusion matrix 4.3 which illustrates the true positives, false positives, false negatives and true negatives.

Table 4.3.: confusion matrix for the SVM with data augmentation

Table 4.4.: Confusion Matrix SVM

	predicted 0.0	**predicted 1.0**
true 0.0	391	43
true 1.0	1654	397

The crosstab function 4.5 from the pandas library gives the possibility to illustrate how many instances of which class are represented in the different clusters.

Table 4.5.: Crosstab of the clusters of the SVM with data augmentation

Table 4.6.: Crosstab SVM of 2 Clusters

label	**cluster0**	**cluster1**
0.0	1	433
1.0	7	2044

The following metrics and key information for the k-means clustering were created after executing the training and the evaluation of the model, 4.7.

In this case the balance accuracy was used to access the model performance. Here one can see that the accuracy drops to 54.6% for the SVM when using the right metric which accounts for the imbalance in the dataset. Furthermore precision and recall also drop. Especially recall drops from previously 0.824% to 0.22 during training. The results for the evaluation phase look similar with an accuracy value of 0.547, a precision of 0.778 and a recall of 0.318. Additionally the F1-score paints a similar picture as the metrics before. With a value of 0.318 it needs to be pointed out that the performance of the model does not equal the one of the models introduced in paper [1]. This applies to the SVM as well as to the created k-means clustering (see 4.7, 4.8 and 4.9) which is also underperforming in comparison with the models introduced In the paper. Here especially the inertia and the calculated accuracy are an indicator for the deviation. It should also be mentioned that the difference in terms

Table 4.7.: k-means clustering results with data augmentation

KMeans Clustering Results:	
Silhouette Score:	0.665
Accuracy k-means:	0.499
Precision k-means:	0.703
Recall k-means:	0.823
F1 Score k-means:	0.746
Inertia k-means:	22471.281
Cluster Centers:	[[1.27e+01, 1.287e+01, 1.046e+01, -2.382e-01],]-4.81e-02, -4.86e-02,-3.95e-02, 8.99e-04]]
Number of Iterations: 15	

Table 4.8.: Confusion Matrix k-means clustering

	predicted 0.0	**predicted 1.0**
true 0.0	1	433
true 1.0	7	2044

Table 4.9.: Crosstab k-means of 4 Clusters

label	**cluster0**	**cluster1**	**cluster2**	**cluster3**
0.0	0	129	6	299
1.0	5	644	48	1354

of using the metric of accuracy and balanced accuracy deviate strongly in their values which proves the point that the class imbalance of the dataset has a huge influence on the actual outcome of the models.
In addition to the metrics a overview over the wrong classified instances is put out in the end describing the difference in the number of wrongly classified instances of the original dataset and the number of wrongly classified instances of the samples which were added through the SMOTE augmentation. Interestingly it becomes evident over the numerous of executed trials that the majority of the wrongly classified instances originated from the original dataset and are not artificially created through SMOTE.
To illustrate the difference in clustering quality of using 2 or 4 clusters, another clustering with 4 classes was applied. In this case the classes still seem to be very imbalanced regarding their occurrence of high and low trust values. Furthermore the distribution in the different classes is very imbalance regarding of the number of values in each cluster. This becomes evident when comparing cluster 0, which has 5 samples whereas cluster 3 contains 1653 samples.

4.1.1. Experiment with threshold 3.5 for phases 1, 2 and 5

After iterating the first runs for the creation of the models, different experiments were conducted including the experiment of labeling the feature vectors for the experiment phases 1, 2 and 5 with a different threshold value then before. Hereby the threshold value determines whether a feature vector is label as high or low trust depending on the result of the questionnaire of the corresponding phase. Instead of using the value of 4.8 as in the initial trial a value of 3.5 was used. The value 3.5 was selected because it is the exact middle value of the scale of the questionnaire. There the answer options for the

participants ranged from zero to seven, which in this case yields a threshold value of 3.5. The following results are obtained while training and evaluation

Table 4.10.: Metrics SVM with data augmentation in experiment with threshold 3.5

SVM Training:	
Average Training Accuracy:	0.5797
Average Training Precision:	0.6794
Average Training Recall:	0.3018
SVM Test Classification Results:	
Accuracy SVM Test:	0.564
Precision SVM Test:	0.815
Recall SVM Test:	0.378
F1-score SVM Test:	0.436

Table 4.11.: Confusion Matrix SVM with data augmentation in experiment with threshold 3.5

Table 4.12.: Confusion Matrix SVM

	predicted 0.0	**predicted 1.0**
true 0.0	651	145
true 1.0	3571	1606

Table 4.13.: crosstab SVM with data augmentation in experiment with threshold 3.5

Table 4.14.: Crosstab of 2 Clusters

label	**cluster0**	**cluster1**
0.0	786	10
1.0	5108	69

Hereby the SVM model demonstrates moderate performance during training and evaluation phases. With an average training accuracy of 57.97% and a precision of 67.94%, the model showed a reasonable ability to correctly classify high trust instances (positives). However the recall was considerably lower at 30.19%, indicating that the model struggled to identify a significant portion of the true high trust instances. This discrepancy between precision and recall suggests that while the model was adept at avoiding false positives, so true low trust instance predicted as high trust, it frequently misclassified true high trust instances as low trust, leading to a higher number of false low trust predictions. The classification results further confirmed this trend, with an accuracy of 56.40%, precision of 81.55%, and recall of 37.78%. The high precision but low recall indicates that while the model was confident in its high trust predictions, it missed many actual high trust instances. The F1-score of 43.64% and the confusion matrix of the SVM classification additionally reflect the imbalance between precision and recall. The model correctly predicted 651 instances as low trust (true negatives) but misclassified 145 low trust instances as high trust (false positives). On the other hand, it correctly identified 1,606 high trust instances (true positives) but failed to recognize 3,571 true positive instances, contributing to the model's low recall. This pattern indicates that the SVM model may have a conservative bias, favoring the negative low trust class and hesitating to classify instances as high trust.
In comparison, the k-means clustering results reveal a different outcome on the dataset. The silhouette

Table 4.15.: k-means clustering results with data augmentation and threshold 3.5

KMeans Clustering Results Experiment with threshold 3.5:	
Silhouette Score:	0.471
Accuracy k-means:	0.4996
Precision k-means:	0.7680
Recall k-means:	0.8569
F1 Score k-means:	0.8028
Inertia k-means:	56782.1282
Cluster Centers:	[[-7.22e-02, -7.43e-02, -6.784e-02, 4.51e-03], [7.75e+00, 7.979e+00, 7.28e+00, -4.84e-01]]
Number of Iterations:	23

Table 4.16.: confusion matrix k-means clustering results with data augmentation and threshold 3.5

Table 4.17.: Confusion Matrix k-means

	predicted 0.0	**predicted 1.0**
true 0.0	10	786
true 1.0	69	5108

Table 4.18.: crosstab k-means clustering results with data augmentation and threshold 3.5

label	**cluster0**	**cluster1**	**cluster2**	**cluster3**
0.0	559	205	3	29
1.0	3242	1725	23	187

score of 0.471 suggests moderate clustering performance, reflecting reasonable separation between clusters. A good silhouette score normally lies over a value of 0.7. The k-means model achieved an accuracy of 49.96%, which is slightly lower than that of the SVM, but it exhibited higher recall at 85.69%. This suggests that the clustering approach was more effective at capturing a broader range of true positive instances (high trust), although its precision was lower at 76.80%. The F1 score of 80.28% indicates a more balanced trade-off between precision and recall compared to the SVM model. The confusion matrix demonstrates that while the model misclassified 786 low trust instances as high trust, it accurately identified 5,108 high trust instances. This resulted in fewer false negatives, so high trust instances labeled as low trust, compared to the SVM model, but at the cost of more false positives, so low trust instances labeled as high trust. This behavior suggests that k-means clustering had a tendency to over-predict the high trust category, classifying a larger portion of instances as high trust even when they belonged to the low trust category. This is also observed when looking at the results of the binary classification feed-forward neural network in section 4.2. Hereby the cause is a different one resulting from the missing augmentation approach the underlying class imbalance of the dataset leads to an overestimation of the high trust labels leading to the same over-prediction. THE inertia value of 56782.13, reflecting the sum of squared distances from points to their assigned cluster centers, suggests that the created clusters including those points do not have a high intra class similarity which results in a significant imbalance, with the majority of high trust instances concentrated in cluster 1. Expanding the analysis to four clusters revealed a more nuanced distribution, but the overall pattern remained consistent, with cluster 1 capturing the bulk of high trust instances. An important observation is that 858 instances (14.36%) were misclassified by k-means, suggesting that while the clustering model

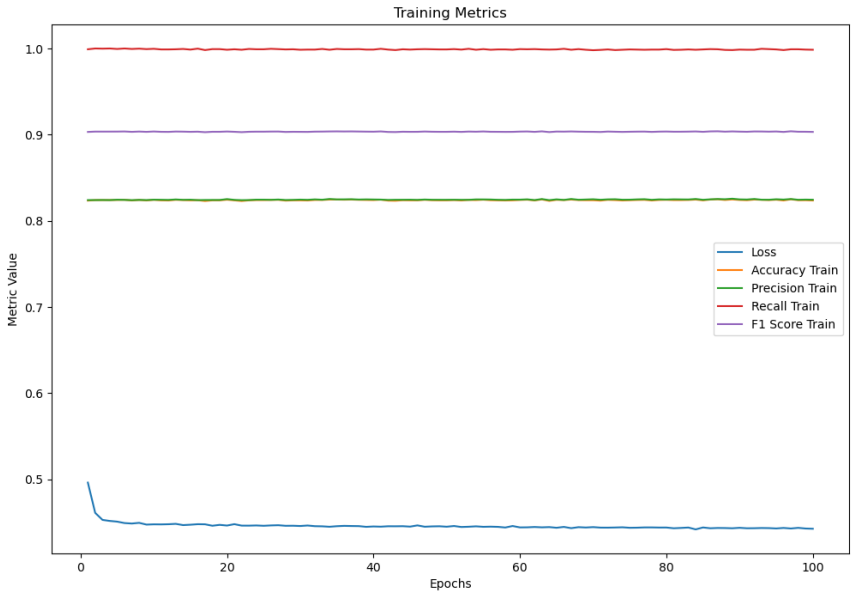
captured major trends, there was still a notable proportion of errors. The results of both models highlight again the contrasting strengths and weaknesses of SVM and k-means clustering. The SVM model demonstrated high precision but low recall, leading to conservative classification with a bias toward the negative class. Conversely, k-means clustering exhibited higher recall but lower precision, indicating a tendency to over-predict the positive class. If minimizing false negatives and capturing as many high trust instances as possible is paramount, k-means may be preferable. However, if avoiding false positives is more critical, the SVM model might be a better fit.

4.2. Results Feed Forward Network

The following section covers the results of different approaches and tests regarding the model creation and evaluation of the binary classification feed-forward neural network. Different training setups and the influence of the augmentation procedure are made evident in the following subsections.

4.2.1. 100 epochs without data augmentation

The first training run involves training the defined architecture on the corresponding dataset for 100 epochs without applying any augmentation approach. This consequently means that the dataset still remains unbalanced resulting in the following evolution of the metrics over the training process, 4.1.



The graph describes the development of the metrics of the cause of the training process of the feed-forward binary classification network.

Figure 4.1.: Training Binary Classification feed-forward network 100 epochs

The figure 4.1 displays the training metrics of the feed-forward neural network over the course of 100 training epochs. The x-axis represents the number of epochs, while the y-axis shows the values of various performance metrics, including loss, accuracy, precision, recall, and F1 score during the training. The blue line represents the training loss, which starts relatively high and decreases rapidly during the initial epochs before stabilizing at a lower value. This indicates that the network is learning quickly at the beginning of training, and the loss converges to a stable point as the training progresses, suggesting that the model has likely reached a state of diminishing returns in terms of further reducing the loss. As a consequence this would mean that training the network over a longer horizon does not improve the loss further. The other lines, representing accuracy, precision, recall, and F1 score, remain relatively stable and high throughout the training process. The recall, depicted by the red line, hovers near 1.0, indicating that the model is almost perfectly identifying the positive class during training. Similarly, the F1 score and precision maintain high values, reflecting that the model's predictions are consistently accurate and balanced between precision and recall. The stability of these metrics across

all epochs suggests that the model is not experiencing overfitting, as there is no indication of significant fluctuations or divergence in the performance metrics.
While indicating a theoretically good performance on the training data the near-perfect recall and high precision, accuracy, and F1 scores during training might also indicate that the model has just memorized the training data rather than generalizing well to unseen data. This results in poor performance on validation and test datasets. This argument is supported when looking at the test metrics logged during the evaluation procedure:
Test Balanced Accuracy: 0.50
Average Test Precision: 0.831
Average Test Recall: 1.0
Average Test F1 Score: 0.9076

The test accuracy lies by 0.5 or 50% which is indicating that the model acts more or less randomly on the unseen data. The precision, recall and F1-scores still remain pretty high. This combination makes the significant class imbalance problem in the test dataset evident. This discrepancy occurs in this case because the model correctly identifies instances of the majority class (high trust instances) but fails to perform well on the minority class (low trust instances). This could also happen vice versa if the model predicts the minority class well but fails on the majority class. Thus, the high recall indicates that the model correctly classifies all true positives which are the instances labeled as high trust. But the poor accuracy reflects the misclassification of the negative class, so the instances labeled as low trust. This means the model is predicting almost everything as the high trust class to ensure it captures all high trust instances, which leads to poor overall accuracy because it misclassifies many low trust instances as high trust. As an example, if 90% of the data belongs to the high trust class and 10% to the low trust class, a model that predicts low trust for almost all instances will achieve perfect recall but will incorrectly classify most of the high trust cases. Hence, accuracy drops significantly, even though precision and F1 score appear high, reflecting good performance on the smaller low trust class. Since this implies that the model is predicting the high trust class for nearly all instances it consequently leads to the following test metric scores:

- a high recall of 1.0 meaning the model correctly identifies all high trust instances.
- a good precision of 0.831 meaning among the instances predicted as high trust, a high proportion is indeed correct.
- a high F1-score of 0.9076 meaning that since the F1-score is the harmonic mean of precision and recall, the high precision and perfect recall yield a strong F1 score.
- a poor balanced accuracy of 0.50 meaning the model completely fails to classify low trust instances, resulting in a recall of 0 for the low trust class, dragging down the averageprediction quality on the test dataset.

The model overpredicts the high trust class which skews the evaluation metrics, masking poor performance on the low trust class. The high recall and F1 score reflect the model's success in detecting the high trust class but overlook the model's inability to correctly handle the low trust class. This proves that without the corrective action of applying an augmentation approach which creates new samples of the minority class or undersamples the majority class the model's general performance becomes misleadingly optimistic.
This consequently also leads to the question whether the low trust or the high trust class is more critical. In real-world scenarios where the low trust class might become critical when it comes to

researchers wanting to know when a human no longer trusts a robot, missing low trust predictions could lead to consequences regarding their outcome. Furthermore systems where trust is critical for maintaining the ethical and right usage need to consequently know when trust is lost and when further actions are needed regarding the improvement of the human-robot interactions.
Additionally, the consistently low and stable loss may also indicate underfitting if the loss plateaus at a higher value than expected. However, given the near-perfect performance metrics, underfitting seems less likely in this case. The lack of divergence between loss and performance metrics could hide potential issues if the training data is too simple or not representative of the problem domain, leading to an overly optimistic view of the model's capabilities. Another potential issue is the underlying class imbalance. If the recall is consistently high, it could mean the model is favoring the majority class, and the metrics might not reflect the true performance on minority classes. This imbalance might not appear in the plot but could manifest in poor generalization to diverse datasets. This issue is the most likely one since augmentation approaches, as SMOTE, were initially not applied to the dataset.

4.3. Results Recurrent Neural Network

4.3.1. 20 epochs without data augmentation

The first training phase without data augmentation produced a test accuracy of 50%, with a high precision of 82.9% and a recall of 100%. This combination resulted in an F1-score of 90.66%, suggesting that the model demonstrated near-perfect sensitivity, accurately identifying all positive instances. However, this recall could indicate an overfitting issue, where the model becomes overly confident in classifying positive high trust samples, potentially at the expense of misclassifying low trust instances. The absence of data augmentation may have limited the diversity of training examples, leading the model to memorize patterns rather than generalize effectively across different data variations. This facilitates the same problem as seen in section 4.2 where the model also overly confident classifies the majority of the test dataset as high trust instances. Test Accuracy: 0.50
Average Test Precision: 0.829
Average Test Recall: 1.0
Average Test F1 Score: 0.9066

4.3.2. First attempt: 20 epochs with data augmentation

In the first experiment incorporating data augmentation, the results shifted dramatically. The test accuracy improved slightly to 54%, and precision rose to 95.86%, but recall dropped to 9.52%, yielding a much lower F1-score of 17.33%. This imbalance between precision and recall indicates that while the model became highly confident in its predictions for the positive class, it failed to detect the instances of the negative class (low trust). This leads to the conclusion that the model has strong bias towards the majority class. The dramatic decline in recall suggests that the model struggled to adapt to the augmented data. This lack of generalization is a common consequence of class imbalance, where the model prioritizes precision.
Test Accuracy: 0.54
Average Test Precision: 0.9586
Average Test Recall: 0.09524
Average Test F1 Score: 0.1733

The following observations and conclusions can be made based on this:

1. Loss Stagnation: The loss curve remains relatively flat and does not show a meaningful decline over the epochs. This indicates that the model is not learning much from the data.
2. Low Recall and F1-Score: Both metrics are extremely low and fluctuating, meaning the model struggles to identify one of the classes likely due to class imbalance or poor feature representation.
3. High Precision: Precision is consistently higher than recall, suggesting the model is biased towards the majority class and avoids predicting the minority class.
4. Flat Accuracy Curve: Accuracy does not improve significantly, which might result from the high class imbalance.

4.3.3. Second attempt: 20 epochs with data augmentation

A closer look at the second attempt with data augmentation, assessed over 20 epochs, reveals a gradual improvement in the model's performance during training, although persistent challenges remained, see 4.19. In the early epochs, the loss began at 0.6907 with an initial training accuracy of 54.18%. Precision at this stage was moderate at 67.3%, but recall was notably low at 16.27%, leading to an F1-score of 26.2%. Over subsequent epochs, the training loss decreased incrementally, and accuracy exhibited slight gains, reaching 57.93% by epoch 20. Precision steadily improved, ultimately reaching 71.57%, while recall increased only marginally, peaking at 26.34%. The final F1-score at the conclusion of training stood at 38.5%, reflecting incremental but insufficient progress in balancing precision and recall.

Epoch	Loss	Accuracy	Precision	Recall	F1 Score
1	0.6907	0.5418	0.6730	0.1627	0.2620
2	0.6809	0.5540	0.6910	0.1950	0.3044
3	0.6795	0.5573	0.6860	0.21099	0.3228
4	0.6777	0.5664	0.6992	0.2332	0.3497
5	0.6773	0.5599	0.7091	0.2028	0.3150
6	0.6753	0.5688	0.6969	0.2430	0.3607
7	0.6763	0.5600	0.7170	0.1997	0.3124
8	0.6746	0.5631	0.7340	0.1979	0.3118
9	0.6749	0.5697	0.7260	0.2240	0.3424
10	0.6758	0.5609	0.7229	0.1977	0.3105
11	0.6745	0.5673	0.7242	0.2174	0.3344
12	0.6744	0.5661	0.7302	0.20982	0.3259
13	0.6732	0.5652	0.7443	0.1985	0.3130
14	0.6725	0.5770	0.7346	0.2427	0.3649
15	0.6727	0.5757	0.7269	0.2427	0.3639
16	0.6740	0.5679	0.7437	0.2075	0.3244
17	0.6733	0.5730	0.7145	0.2443	0.3640
18	0.6729	0.5670	0.7504	0.2038	0.3205
19	0.6746	0.5731	0.7114	0.2458	0.3654
20	0.6722	0.5793	0.7157	0.2634	0.3850

Table 4.19.: Training metrics 2nd training RNN with data augmentation

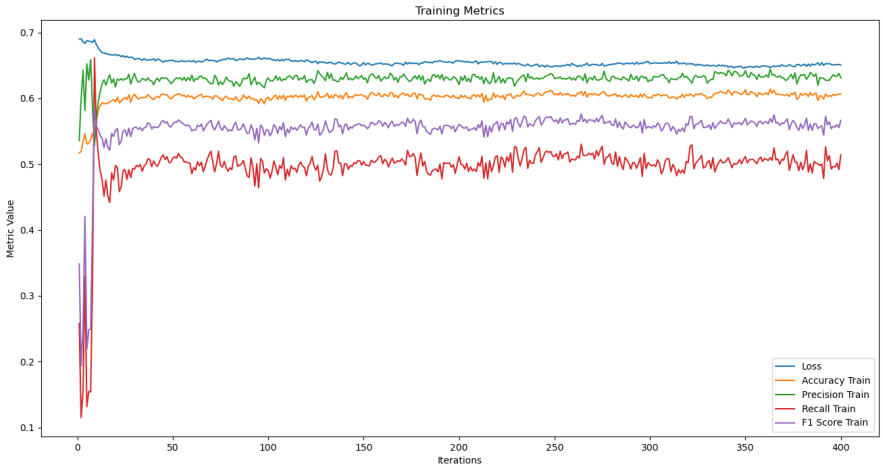
Despite some improvements in training metrics, the model's performance on the test set continued to reflect the underlying difficulties, see 4.20. The test accuracy remained at 54%, with a precision of

Test Accuracy:	0.54
Average Test Precision:	0.895
Average Test Recall:	0.166
Average Test F1 Score:	0.279

Table 4.20.: Test metrics 2nd training RNN with data augmentation

89.52%, but recall stagnated at 16.57%, producing an F1-score of 27.97%. This divergence between high precision and low recall indicates that the model effectively identified certain high trust instances

but consistently failed to generalize this ability across the dataset. The stagnation in recall suggests that the model struggled to adapt its decision boundary to capture the low trust class, reinforcing the notion that class imbalance heavily influenced the model's predictions.



The graph describes the development of the metrics over the course of the training process of the RNN.

Figure 4.2.: Metrics Training RNN

A key observation throughout the experiments is the relatively flat loss curve, which failed to show significant declines across epochs. This stagnation implies that the model's capacity to extract meaningful features from the data was limited. This results in the suggestion that either the architecture lacked sufficient complexity to capture intricate patterns or that the input features themselves did not adequately represent the problem space. Additionally, the flat accuracy curve further validates the idea that the model failed to learn effectively from the data, resulting in minimal improvements over time. The consistently low recall across multiple trials highlights a recurring challenge faced by the model in detecting the minority class of low trust instances. This issue stems from the imbalance in the training dataset, where the dominant class overshadows the minority class, leading the model to favor predictions that align with the majority. While data augmentation is intended to mitigate such issues by introducing variability through new samples into the training data, the results indicate that augmentation alone was insufficient to address the underlying imbalance. The persistence of low recall despite high precision reflects a behaviour that prioritizes minimizing false positives while tolerating a significant number of false negatives. Consequently this means having less low trust instances being predicted as high trust is more important even if it means having more high trust instances being wrongly labeled as low trust. Implementing more advanced augmentation techniques or utilizing oversampling methods to rebalance the dataset could be other ways of trying to find augmentation approaches which might yield better results. Additionally, exploring more complex architectures, such as EEGNet or attention mechanisms, could improve the model's ability to capture long-term dependencies and subtle features within the data. Since there was a limited time frame to conduct this work no further augmentation approaches are discussed or further analyzed which therefore imposes potential future research areas for this problem. More complex and advanced architectures like the EEGNet and their results are further described in section 3.2.3.1 (architecture) and hereafter 4.4.

4.3.4. 400 Epochs RNN Training

As an additional experiment, the RNN was trained for 400 epochs to analyze the impact of a enlarged training horizon. The graph A.2 illustrates the training metrics throughout training the model. The metrics displayed include loss, accuracy, precision, recall, and F1-score. The loss starts at a relatively high value and shows a sharp decline in the early iterations, which is characteristic of the initial phase of training when the model quickly learns basic patterns. However, after the initial drop, the loss decreases more gradually and stabilizes, indicating convergence. The balanced accuracy 3.3 increases rapidly during the early iterations, which aligns with the observed decrease in loss. It then continues to improve at a slower pace and stabilizes as training further progresses. Precision and recall, while initially fluctuating significantly, show more stable behavior after the early iterations. Precision demonstrates a consistent increase and recall also stabilizes after the initial fluctuations. The trends suggest that the model becomes better balanced in its predictions as training progresses. The F1-score follows a pattern similar to that of precision and recall. It initially increases sharply, reflecting the overall improvement in the model's classification performance, and stabilizes over the following epochs/iterations. Overall, the model does not seem to underfit because both the loss and accuracy improve during training. Furthermore the training results indicate that the model has achieved a balance in learning after the execution over 400 epochs.
The test metrics obtained after testing the trained model on the test dataset are the following:

- Test Accuracy: 0.57
- Average Test Precision: 0.881
- Average Test Recall: 0.372
- Average Test F1 Score: 0.523

Those results indicate that the model achieves a moderate test accuracy of 0.57, suggesting limited predictive capability in distinguishing between low and high trust values. The precision score of 0.88 demonstrates that the model is effective in minimizing false positives. However, the recall of 0.37 highlights a significant limitation in capturing the full spectrum of true positive cases, so correctly predicted high trust instances. The F1-score of 0.52 reflects the balance between precision and recall but remains relatively low. Overall the results suggest that the model struggles to generalize effectively across the test set. The disparity between precision and recall indicates potential issues such as lack of data diversity, overfitting during training or the underlying complexity and imbalance of the dataset.

4.4. Results EEGNet

This section covers the results which were obtained training the EEGNet for multiple epochs. In summary five training runs were completed lasting around twelve hours per run on a MacBook M2 Pro with 32Gb of RAM. It needs to be pointed out that no GPU which resulted in less computational resources. Therefore no hyperparameter optimization was done which should be part of a future research project building upon the knowledge gains provided in this study. The EEGNet was utilized for classification as well as for regression purposes. In the case of classification the two classes of high and low trust instances exist whereas the trust scores of the questionnaire serve as the direct label for the regression problem. For the classifier as well as for the regressor five training runs were executed resulting in a total of 10 runs to investigate the performance of the EEGNet on the provided dataset. The overview over all results, logged metrics and other details can be found in the corresponding tables in the appendix A.7.

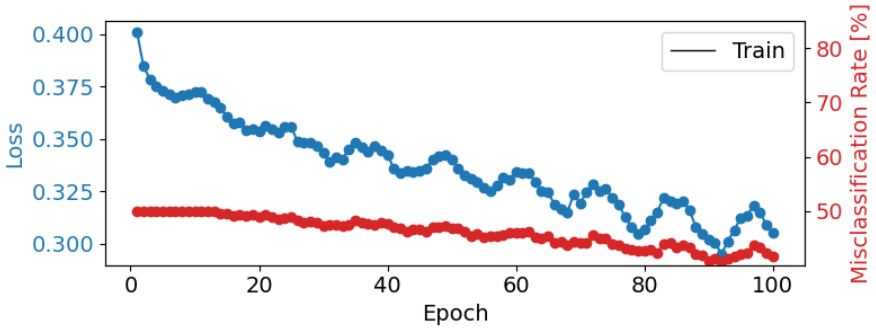
4.4.1. Binary Classification

The binary classification problem is tackled using the EEGNet as model to build a classifier which is able to differentiate between the two classes of low trust and high trust level. As mentioned in section 3.2.3.1 **braindecode** is used to define the model. Here the number of outputs **n_outputs** needs to be set to **2**. The number of input channels, **n_chans**, for the network corresponds to the number of raw EEG channels which is in this case **14**. The sampling rate **n_times** which is needed for windowing passes the number of time samples of the input window to the model. In this case this corresponds to **128**. The last specification which is set is the definition of the last convolutional layer which specifies the final length of convolutional filters. Here **final_conv_length** is set to **auto** which means that no specific integer value for the final length is given. This model specification remains the same throughout the five different training procedures. The class **classifier** of the **braindecode** API provides the opportunity to define the classifier which should be trained. In this case the following parameters are important for the training runs because some of the specifications are varied across the different trials:

- model - is the EEGNetv4 model architecture which is used to train the classifier
- criterion - specifies the loss function used for training. Here mainly the **CrossEntropyLoss** is used.
- default optimizer - specifies the optimizer used for training. Throughout the five training's variations occur but the Adam optimizer is most frequently used
- LR - specifies the hyperparameter LR which is varied across the experiments.
- batch_size - specifies the size of the mini batches used for training. In this case batch size 16 is used for all training's.
- callbacks - callbacks are functions which are executed during training. In this case the callback functions calculate the metrics logged during training.
- warm_start - Whether each fit call should lead to a re-initialization of the module (cold start) or whether the module should be trained further. By default set to false - so cold start.
- train_split - a callable function is passed to further define the train/validation split. If none there is no split defined and the model only works with the defined training dataset.

4.4.1.1. First Training

For the first training run the following specification were chosen:



Loss and misclassification rate of the EEGNet training over 100 epochs

Figure 4.3.: 1st Training EEGNet classification 100 epochs

The EEGNetv4 model was trained over 100 epochs using the 14 channel dataset and the sampling rate of 128 Hz, corresponding to 2 output classes. Cross Entropy Loss was used as the criterion, and the model was optimized with stochastic gradient descent at a LR of 0.001. No warm start or train split was specified. During the training process, the initial loss at epoch 1 was 0.4451, which gradually decreased over the course of the run. By epoch 10, the training loss had dropped to 0.4173, indicating early improvement. The trend of decreasing loss persisted, with fluctuations occurring around the middle of the run. By epoch 20 the loss had reduced to 0.4082, and by epoch 50, it had further declined to 0.4016. The final epochs demonstrated a steady performance, with the lowest loss recorded at epoch 99 at 0.3978, see A.7 subsection A.7.1. The duration per epoch began at approximately 46.6 seconds and varied throughout the training process. Some epochs, particularly around the midpoint, exhibited increased duration's, such as epoch 15, which lasted 73.6 seconds. However, the duration's stabilized towards the latter part of the training, averaging around 44 seconds per epoch. Despite the successful completion of the training phase, the testing phase resulted in failures caused due to an error in the evaluation part of the script resulting in conflicting dimensions of the output tensors.
The graph 4.3 depicts the results of another training run over 100 epochs, with loss shown on the left y-axis and misclassification rate on the right y-axis. The key parameters were chosen to be exactly the same as in the first run. This run was executed to validate the results of the first run with the addition of plotting the evaluation metrics. The x-axis represents the number of epochs. The blue curve illustrates the training loss, while the red curve represents the misclassification rate. At the beginning of the training process, the initial loss is relatively high at around 0.40, and the misclassification rate starts at approximately 70%. As the training progresses, the loss steadily decreases, showing a general downward trend with periodic fluctuations. By the midpoint of the training, the loss stabilizes around 0.325, continuing to decline gradually until it reaches approximately 0.30 by epoch 100. This results is significantly better then the ones from the first run. The misclassification rate follows a similar pattern, steadily declining throughout the training period. Although the rate decreases more slowly compared to the loss, it demonstrates consistent improvement, eventually dropping to around 45% by the end of the run. The fluctuations in the loss curve suggest moments of instability or temporary overfitting, but the overall trend indicates continuous learning. The consistent decrease in the misclassification rate reflects the model's improving accuracy over time. However, the final misclassification rate remains relatively high, which may indicate potential issues in generalization.

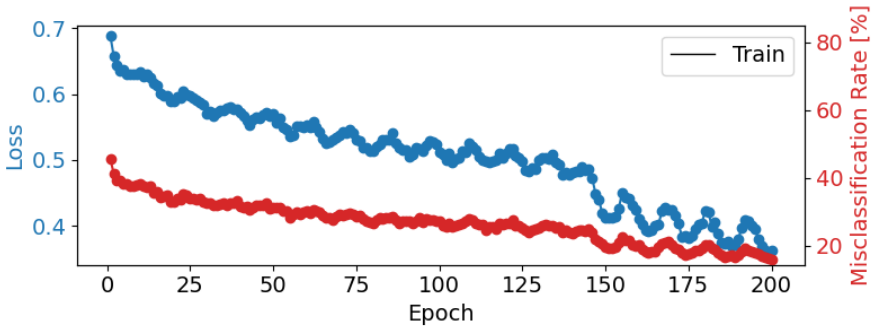
4.4.1.2. Second Training

The second training run for the EEGNetv4 model was also executed over 100 epochs and shows a steady and gradual decrease in training loss. At the start of the training process, the initial loss is recorded at 0.4333, which again indicates a significant high error rate. However, the pattern of learning is similar as in the first run with the loss decreasing significantly in the first few epochs, reaching 0.4097 by the second epoch and continuing this trend through the initial stages. By epoch 10, the loss stabilizes around 0.4026. Throughout the training process, the loss continues to exhibit a downward trajectory, though the rate of decline slows as the epochs progress. Around the midpoint of training, specifically at epoch 50, the loss is reduced to 0.3898, showing a consistent pattern of improvement. In the later stages of the run, the loss fluctuates slightly but remains consistently below the earlier levels, ultimately concluding at 0.3817 by epoch 100. Similar to the first run the consistent reduction in training loss gives the suggestion that the model is able to learn and improve. Furthermore there are no significant signs of overfitting or plateauing. This suggests that the LR and other hyperparameters are already reasonably well-tuned for this particular configuration but remain to be investigated in the future since hyperparameter optimization brings a lot of potential for improvement. The final performance metrics show that the model achieves an accuracy of 86.94%, which at first glimpse indicates strong classification performance. However, the balanced accuracy of 49.98% points out again the discrepancy between accuracy and the overall error rate, which occurs due to the underlying class imbalance.

4.4.1.3. Third Training

The third training also demonstrates a steady and progressive improvement in balanced accuracy and a consistent reduction in loss over the 200 epochs recorded, see A.7.3. The model starts with a relatively low balanced accuracy of 54.45% in the first epoch, while the training loss begins at 0.6877. As training progresses, balanced accuracy consistently improves, reaching 82.14% by the final epoch, and the loss gradually decreases to 0.3915, see A.7.3. The major changes for this run were the increase of the number of epochs which the model was trained for, the introduction of class weights for the cross-entropy loss function to further mitigate the problem of class imbalance and the application of the Adam optimizer instead of the trivial stochastic gradient descent. Furthermore callbacks were implemented to track additional metrics like the balanced accuracy. Lastly a scheduler was implemented to be able to adjust the LR during training. That is why the LR is also logged during training. The scheduling of the LR appears to contribute positively to the model's performance. Dynamic LRs help the model escape potential local minima and allows for continued improvements in accuracy, particularly in later stages of training. Notable improvements in accuracy occur after certain drops in the LR, such as in epochs 55, 102, and 161, suggesting the effectiveness of lower learning rates in fine-tuning the model. There are periods where the accuracy plateaus or even experiences minor decreases, such as around epochs 19 to 30 and again between epochs 85 to 110. However, these plateaus are eventually followed by further improvements, reflecting a certain resilience of the model. By the final stretch of epochs, the model stabilizes at higher accuracy levels. One remarkable feature of this training run is the significant jump in performance after epoch 140, where accuracy climbs from approximately 76% to over 80% in a relatively short period compared to previous periods. This sharp increase suggests that the model has reached a point of better generalization. Compared to the more trivial models of the binary classification model and the created RNN the model likely extracts more meaningful features from the data which could be traced back to the spatial context which can be captured from the convolutional layers. Compared to previous runs, this training run shows improved performance, with the highest

balanced accuracy of 82.14% surpassing earlier results. This improvement suggests the effectiveness of the chosen architecture, loss function, and optimization strategies.



Misclassification rate (red) and loss (blue) over the course of the EEGNet classifier training over 200 epochs.

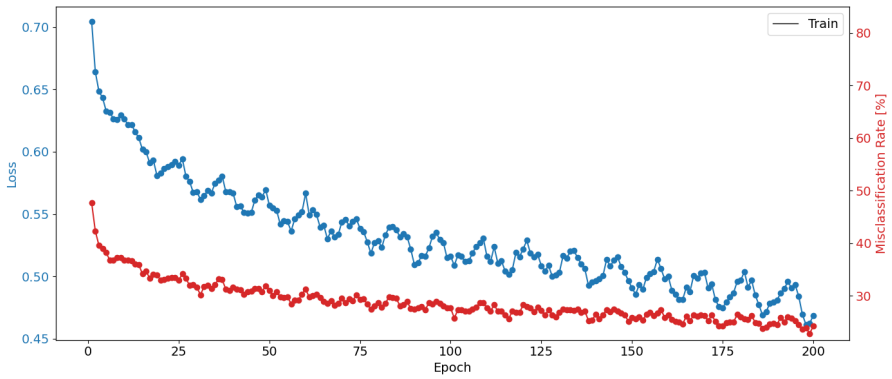
Figure 4.4.: EEGNet classifier 3rd training 200 epochs

The figure 4.4 created from the training run also shows the progression in terms of performance compared to previous runs. The key metrics are the loss (left y-axis, blue) and and the misclassification rate (right y-axis, in red). The x-axis represents the number of epochs. The loss starts at around 0.68 in the initial epochs. There is a steady, consistent decline in loss as training progresses and by the 200th epoch, the loss reaches just below 0.35. The pattern of the course of the training features slightly oscillating patterns, resulting from the adjusted learning rates. However, the overall trend is downward, demonstrating continuous model optimization. The misclassification rate starts at around 50% but as training advances, the misclassification rate steadily drops, with minor fluctuations. By the final epochs, the misclassification rate is reduced to around 18%. Similar to the loss curve, the misclassification rate shows oscillations but consistently trends downward over time.

4.4.1.4. Forth Training

A similar behaviour over the course of the training as in run three is also observable in run four. The corresponding metrics are shown in table A.7.4. The biggest change comparing the training run to the previous ones is that now a **train_split** is directly Incorporated in the classifier definition. The predefined_split method from takes the validation set and returns it as a tuple to **train_split**. Additionally the metrics valid_acc and valid_loss are added to the log. Here both, the training and validation performance as well as the fluctuations in validation accuracy are still observable. The initial epoch shows relatively low training balanced accuracy 52.42% and high loss 70.14%. By the second epoch, the model experiences a sharp drop in validation accuracy from 85.71% to 49.67%. Over subsequent epochs, the model gradually improves, with training balanced accuracy rising consistently alongside a steady decrease in training loss. At around the 20th epoch, the model's performance stabilizes, with validation accuracy fluctuating between approximately 60% and 80%. This pattern suggests that while the model generalizes well, there are occasional instances where it struggles to maintain high validation performance. This could be a cause of the nature of the EEG data, where subtle variations have significant impact and the dimensionality and information density could lead to such occasional behaviour. The LR scheduling process also appears to positively influence this training. As evidence the periods of improvement following adjustments in the LR can be stated. Notably, epochs 40 to 60 show an upward trend in validation accuracy, reaching as high as 87.78%. This

period coincides with a decrease in the LR, supporting the hypothesis. Beyond epoch 60, validation accuracy still continues to improve including some occasional dips. The highest training balanced accuracy, recorded at 73.20%, aligns closely with the highest validation accuracy observed near the end of the training run. This convergence indicates that the model is approaching its optimal state. The final epochs of the training show a strong ability to generalize because the model consistently achieves validation accuracies between 70% and 88%. In comparison to the previous run the final testing accuracy of 76.84% and balanced Accuracy of 79.17% are lower indicating that the model might not be able to deal with test dataset as well as the previous model did. But in general this training run also showcases a stable and accurate model for the binary classification task. The second run incorporating the Adam optimizer and LR scheduler also shows that the model benefits from those made adjustments.



The graph shows the progression of training loss (blue) and misclassification rate (red) over the course of the training over 200 epochs.

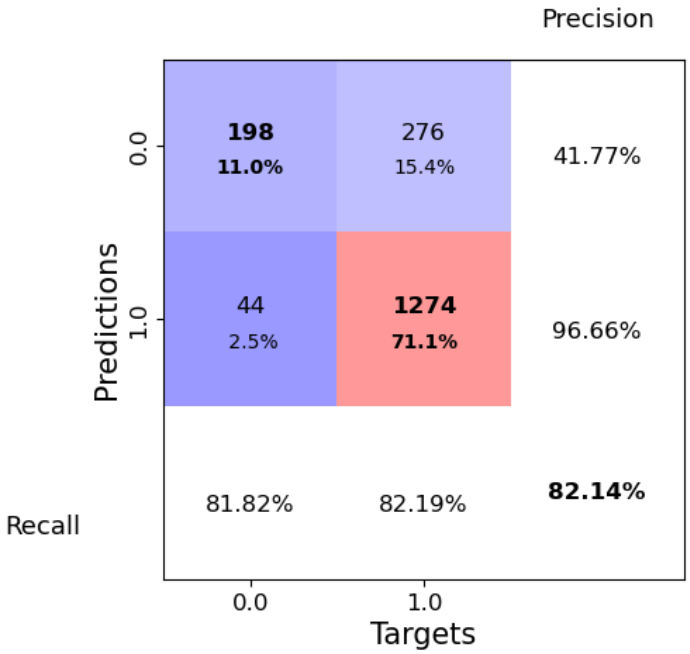
Figure 4.5.: EEGNet classification 200 epochs 4th training

The figure 4.5 also shows the progression of training with respect to the tracked metrics of training loss and misclassification rate. The blue curve represents the loss, while the red curve corresponds to the misclassification rate. The model initially struggles to perform well, as expected during the early phases of training. However, as the epochs progress, both the loss and misclassification rate decrease steadily, suggesting that the model is gradually learning. By around epoch 50, the loss drops below 0.60, and the misclassification rate stabilizes near 50%. Although the improvement rate slows down, the downward trend continues throughout the training process. The misclassification rate shows less variability compared to the loss curve. These oscillations in the loss may indicate that the model is still undergoing fine adjustments as it converges. By epoch 200, the loss approaches 0.45, and the misclassification rate falls below 30%, reflecting the same results as mentioned in table A.7.4.

4.4.1.5. Fifth Training

The fifth training run was executed with the same specifications introduced in run four, see A.7.5. As in all previous runs the model shows a progressive improvement in balanced accuracy and decreasing training loss over all 200 epochs. The initial stages of the training process reveal relatively low balanced accuracy, starting at 0.5237. By epoch 200, the balanced accuracy reaches 0.7581, indicating a significant improvement in the model performance. The consistent reduction in training loss, from 0.7043 to 0.4686, further proves that the model is learning and fitting the data more effectively with ongoing training. The model's perfromance on the unseen validation set varies a lot over the cause

of training and therefore the validation accuracy is fluctuating throughout the training run. By the later epochs, validation accuracy stabilizes around in the range of 0.76 to 0.86, suggesting that the model achieves better generalization performance over time. The highest recorded validation accuracy of 0.8717 is reached at epoch 83. Throughout the training run, the LR scheduler is used to adjust the LR corresponding to improvements or no improvements in the loss. This is again effective in mitigating overfitting while promoting continued learning. Notably, the LR tends to decrease to values as low as 0.0000, allowing the model to make finer adjustments. The duration per epoch fluctuates over the course of the training run. Initially, the time required for each epoch ranges around 190 seconds, but by the later stages of training this value increases up to 294 seconds per epoch. The reason for this lies in the Central Processing Unit (CPU) usage and the corresponding free computing resources on the RAM. If computational overhead is created through computations with large data batches this can therefore result in more time needed for training processes. In comparison to previous training runs, the results of training five reflect a improvement in both training and validation metrics. The balanced accuracy in earlier experiments typically plateaued around 0.65 to 0.70, whereas the current training run consistently surpasses this threshold,. The reduced gap between training loss and validation loss in the later stages of training indicates reduced overfitting. That means that the model is better aligned with the general structure of the data. Some irregularities still persist since periodic spikes in validation loss, particularly in epochs where validation accuracy drops below 0.60 is still evident. The model may still face challenges in handling outliers or difficult samples in the validation set. Future work could use this as a set up point to try different augmentation or also regularization techniques to mitigate this.



Confusion Matrix of the predicted classes for the test data. In the right lower corner are the true positives, so correctly labeled high trust instances. And in the upper left corner are the correctly classified low trust instances.

Figure 4.6.: Confusion Matrix for 5th EEGNet classifier training 200 epochs

The confusion matrix shown in figure 4.6 provides an overview of the model's performance in terms of

correctly and incorrectly classified samples after 200 epochs of training. It is based upon the predictions for the test data set used to evaluate the perfromance of the model after training. It indicates that the model correctly classified 1274 high trust samples (true positives) and 198 low trust samples (true negatives). However, there are 276 false negatives, meaning that these samples were incorrectly predicted as low trust when they were actually high trust instances. Similarly, the model misclassified 44 low trust samples as high trust instances (false positives). The precision for class 1 (high trust) lies by 96.66%, indicating that the model is highly accurate when predicting high trust cases, while the precision for class 0 is 41.77%, suggesting a lower ability to correctly identify low trust cases. The recall values are relatively balanced between the two classes, with class 1 (high trust) having a recall of 82.19% and class 0 showing a recall of 81.82%. The overall F1-score, calculated from the precision and recall, reflects a reasonable balance between the two metrics, as demonstrated by the combined metric of 82.14% in bold at the bottom right.

4.4.2. Regression

For training the regressor the **braindecode** API was used as well. Here the model specification for the EEGNet is the same as for the binary classification task, with the difference of outputting only one class in the end. This class is technically no class because it only holds one real valued number which is the output from the model. This one value is the predicted trust value for the current sample. The definition of the regressor is again varied in the five different trainings executed. The following parameters are important for the specification:

- model - is the EEGNetv4 model architecture which is used to train the regressor
- criterion - specifies the loss function used for training. Th loss function is changed in some training runs but mainly the MSE loss with **torch.nn.MSELoss** is used.
- optimizer - specifies the optimizer used for training. Throughout the five training's variations occur but the AdamW optimizer is most frequently used
- optimizer__lr - specifies the hyperparameter LR
- train_split - a callable function is passed to further define the train/validation split. If none there is no split defined and the model only works with the defined training dataset. In this case the **skorch** method **predefined_split(val_dataset)** is used.
- batch_size - specifies the size of the mini batches used for training. In this case batch size 16 is used for all training's.
- callbacks - callbacks are functions which are executed during training. In this case the callback functions calculate the metrics logged during training.

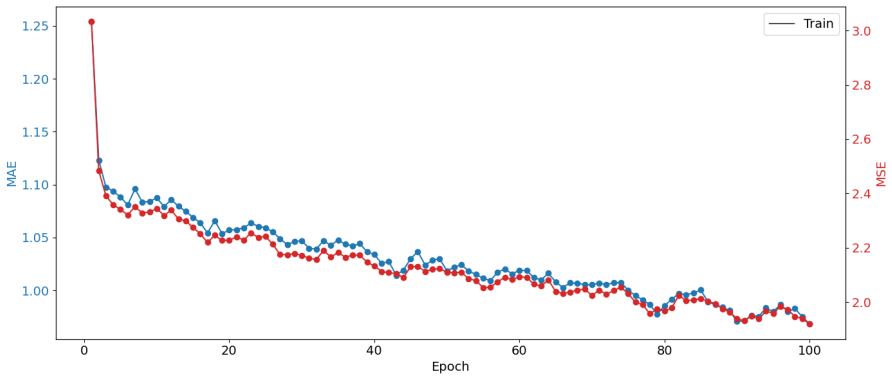
The used AdamW optimizer varies from the Adam optimizer in the way that it includes a weight decay as a regularization technique. This adds a penalty term to the loss function to prevent overfitting. Thus the updating steps for adjusting the weights becomes different compared to Adam. In general all of the different parameters which define the regressor are varied across the training runs. Some, like the train_split remain consistent over all executed runs.

4.4.2.1. First Training

For the first run no metrics were output directly to the console, that is why a detailed analysis per epoch is not possible, see A.8.1. For the first training run the following configuration was used:

- criterion = MSELoss
- optimizer = torch.optim.AdamW
- optimizer__lr = 0.001
- train_split=predefined_split(val_dataset)
- batch_size=16

Figure 4.7 shows the progression of the two performance metrics MAE and MSE, over the course of 100 epochs. The x-axis represents the number of epochs, while the left y-axis tracks the MAE values and the right y-axis corresponds to the MSE values. The plot depicts the MAE curve in blue and the MSE curve in red. On first sight both metrics exhibit a general downward trend as training goes on. At the start of the training process, the MAE and MSE values are relatively high. This is because the model initially struggles to accurately predict target values. Following this there is a sharp decline in both metrics during the early epochs, suggesting that the model quickly learns to reduce errors and improve performance. As the number of epochs increases, the rate of improvement slows down, and the curves begin to stabilize. Throughout the later epochs, the MAE and MSE curves continue to show slight fluctuations, but the overall trend indicates consistent improvement. The convergence of the two metrics towards lower values at the end of the training period suggests relatively good performance by the 100th epoch. The close alignment between the MAE and MSE curves throughout the training process highlights that the model's predictions are becoming more refined and stable over time. Hereby the goal is to always effectively balance complexity and regularization which lead to a robust and reliable training outcome. Another figure which shows the training progress features the



The graph describes the progress of the MAE and MSE over the course of the training process for 100 epochs.

Figure 4.7.: 1st training EEGNet regression 100 epochs showing MAE and MSE

metrics of Root Mean Squared Error (RMSE) and the coefficient of determination, see A.6. Hereby the RMSE is depicted in orange and the coefficient of determination is depicted in green. Both are show a opposite behaviour over the cause of training with the RMSE being further reduced and the coefficient of determination further raised. In the end the following results are obtained on the test data:

- MAE: 0.88
- MSE: 1.88
- RMSE: 1.37

- Coefficients of Determination: 0.16

The MAE of 0.88 reflects the average absolute difference between the model's predictions and the actual target values. This means that, on average, the model's predictions deviate from the true values by 0.88. It provides an interpretable error measure in the same units as the target variable, meaning that the predicted trust score can vary by 0.88 which is with regard to the scale from 0 to 7 a quite significant deviation. The MSE of 1.88 is another measure of error, but it penalizes larger errors more than smaller ones by squaring the differences between predictions and true values. This value suggests that, on average, the squared difference is 1.88. The RMSE of 1.37 is the square root of the MSE. RMSE also expresses the error in the same units as the target variable meaning that in the range of 0 to 7 a deviation in the predicted values of 1.37 occurs. The coefficient of determination of 0.16 suggests that the model explains only 16% of the variance in the target variable, meaning 84% of the variability remains unaccounted for. This indicates a relatively weak predictive performance. So the model captures some pattern in the data but a significant portion of the variability is not well explained by it. While this might seem low, whether it is acceptable depends on the context and the specific problem. Regarding the task of EEG-based regression data can be noisy and complex. So even a low value for the coefficient of determination can still provide valuable insights. EEG data is often subject to significant variability due to biological and environmental factors. Thus it becomes more challenging to achieve high values.

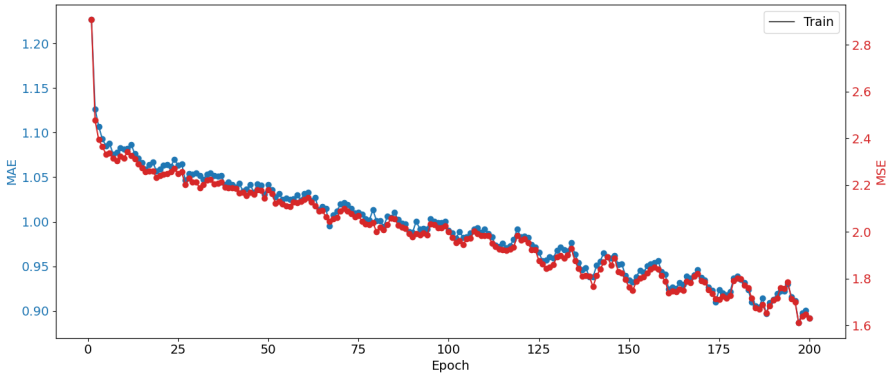
4.4.2.2. Second Training

The second training run features only one adjustment which is the increase of the number of epochs the model is trained for. In this case the training lasts over 200 epochs, providing the model with a longer horizon to converge. The table A.8.2 shows the metrics progression over the cause of the training. What becomes evident in the table is a clear progression in the performance over the course of 200 epochs. The model's training loss consistently decreases, indicating that it is effectively learning to map the raw EEG data to the target regression values. This trend is further supported by the validation loss, which also shows a general downward trajectory also including some fluctuations. The validation loss provides suggests that the model is generalizing well and is not overfitting to the training data. Though it is worth noting that the validation loss begins to plateau around epoch 150. This could be an indication that the model is approaching its optimal performance. The reported values for the coefficient of determination provide a measure of the model's ability to explain the variance in the target data. The training R-squared (coefficient o determination) starts at a low value and steadily increases. This is consistent with the observed trends in the training and validation losses. The validation R-squared also shows a general upward trend which is also visible in figure A.8.2 where the evolution of the RMSE and the coefficient of determination is shown. Notably the increase to 200 epochs yields better results then the previous training on 100 epochs:

- MAE: 0.83
- MSE: 1.41
- RMSE: 1.19
- Coefficients of Determination: 0.35

Figure 4.8 presents two line graphs, one for MAE and the other for MSE, plotted against the number of training epochs. Both graphs exhibit what was already analyzed looking at table A.8.2. The MAE curve shows a rapid initial decrease, indicating that the model quickly learns to predict well. The

MSE curve follows a similar trend to the MAE curve, with a steep initial drop and a plateauing phase around epoch 150. However, the MSE curve is generally higher than the MAE curve, indicating that the model is more susceptible to larger errors. In general the second training run yields better results



The graph describes the progress of the MAE and MSE over the course of the second training process for 200 epochs.

Figure 4.8.: 2nd training EEGNet regression 200 epochs showing MAE and MSE

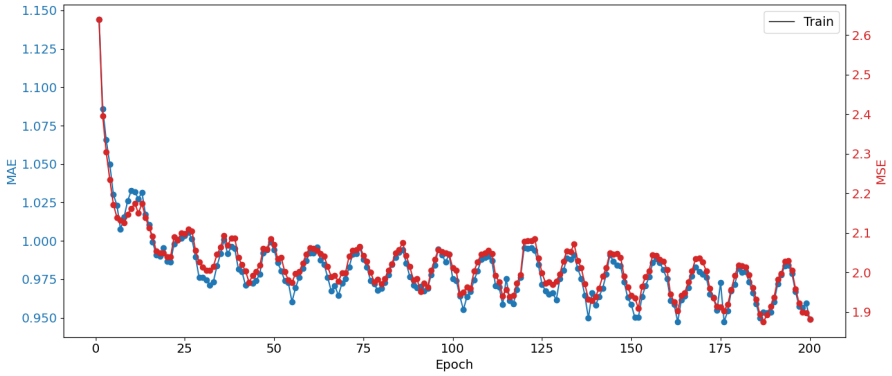
than the first one, indicating that the model benefits from a longer training horizon.

4.4.2.3. Third Training

The provided results table in A.8.3 shows the performance of the model over 200 epochs logging the metrics training loss, validation loss, R-squared (coefficient of determination), MAE, and RMSE recorded for each epoch. For this training run the only adjustment made was to set the LR initially to the value 0.01 which is ten times bigger then the previous LR of 0.001. This model also exhibits a decreasing training loss over the epochs, which is similar to the previous training's. However in this case the losses seem to decrease with a flatter gradient and only over the first epochs. Both training and validation loss remain relatively high, which indicates that the model is underfitting the data and may not have sufficient capacity to learn the underlying patterns. This can be illustrated through figure 4.9 which shows the progress of the MAE and MSE metrics over time. It appears that the adjusted learning rate leads to an oscillation of the metrics in the later stages of training. Those fluctuations and plateaus indicate that the model's LR needs adjustment or further optimization. This is the main result of this experiment, since the LR was adjusted for this run, it becomes evident that this influences convergence and model behaviour during training. There is still a slight downwards trend in the graph's of MAE and MSE visible which is not very significant in terms of the actual increment of its improvement.

Furthermore there is a noticeable gap between the training and validation curves for both MAE and MSE. This gap suggests a degree of overfitting. The model seems to be learning the training data too well, potentially memorizing noise or specific patterns. Thus it hinders its ability to generalize to unseen data. After the training stage the model was tested on a test set which yielded the following results:

- MAE: 0.89
- MSE: 1.84
- RMSE: 1.36



The graph describes the progress of the MAE and MSE over the course of the third training process for 200 epochs. For this training the LR was adjusted to 0.01 instead of the previously used 0.001.

Figure 4.9.: 3rd training EEGNet regression 200 epochs adjusted LR

- Coefficients of Determination: 0.15

Those results suggest that the model's predictive accuracy is limited. The MAE of 0.89 suggests that, on average, the model's predictions deviate by 0.89 from the actual values, which is in the context of the scale for 0 to 7 a significant error. The MSE of 1.84 highlights a higher sensitivity to larger errors, reflecting the presence of some significant deviations between predictions and actual values. The RMSE on the test data lies by 1.36 represents the standard deviation of prediction errors which is regarding the scale of the questionnaire imposing a noticeable variance. The coefficient of determination at 0.15 indicates that the model explains only 15% of the variance in the target variable. As stated in the analysis of the first training run, the coefficient of determination is not the focus point because EEG data is highly complex and the coefficient of determination might fail to be increased with high complexity in the data. But it indicates that the performance of the model struggles under these conditions.
Finally, the choice of hyperparameters used in the model and training process can significantly impact its performance. This third training run gives enough evidence for this statement since even with a learning rate scheduler and the Adam optimizer which adjusts learning rates for the parameters individually, the changed LR has a huge influence on the outcome.

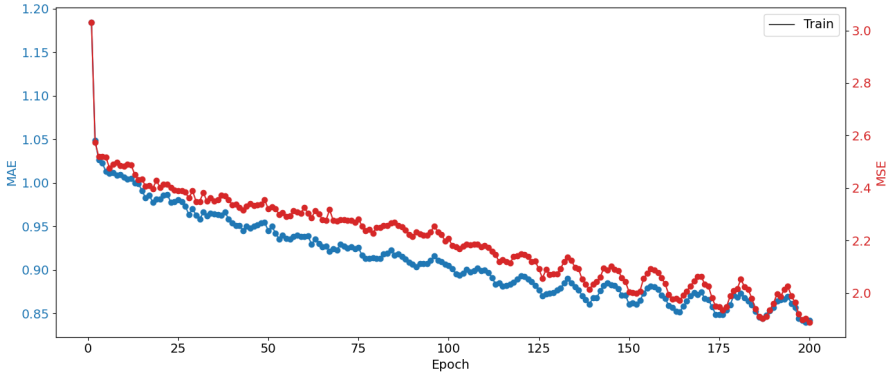
4.4.2.4. Forth Training

The forth training features the change of the loss function to the Huber Loss, 4.4.

$$L_\delta(a) = \begin{cases} \frac{1}{2}(a)^2 & \text{if } |a| \leq \delta, \\ \delta(|a| - \frac{\delta}{2}) & \text{if } |a| > \delta, \end{cases} \tag{4.4}$$

The part of $a = y - \hat{y}$ describes the difference between label y and prediction $\hat{y}$ whereas δ is a threshold parameter which controls the transition between the quadratic and linear behavior of the loss. The major advantage of the Huber loss is, that it penalizes through this incorporated linear and quadratic behaviour small differences quadratically with $|a| \leq \delta$ and larger differences linearly with $|a| > \delta$. It combines the robustness to outliers with the penalization of small errors. Here as well as for the other training's a horizon of 200 epochs for the training is selected and the progression is shown in the following table A.8.4. The Huber Loss was chosen to monitor the impact of a loss function which is

able to handle the impact of outliers well. This especially is interesting in the context of the application of SMOTE. The reason for this is that SMOTE creates artificial samples which might also negatively influence the consistency of the dataset when outliers are created. Figure 4.10 shows the evolution of the metrics of MSE and MAE over the course of the training. Additionally the other logged metrics are illustrated in the figure A.9. The general trends of MSE and MAE show a decreasing progression as the number of epochs increases. Since the also errors appear to stabilize toward the later epochs it can be suggested that the model is converging. In the initial Epochs from 0 to 50 a sharp decline in both MSE and MAE, reflecting rapid learning and adjustment. This follows a gradual decrease at a slower rate in epochs 50 to 150. In the final epochs only a minimal improvement is observable which indicates diminishing returns from further training. The convergence point therefore lies around epoch 150 where the lines for both MSE and MAE begin to flatten. The consistent reduction in errors shows



The graph describes the progress of the MAE and MSE over the course of the training process for 200 epochs. For this training run the loss function was changed to the Huber Loss.

Figure 4.10.: 4th training EEGNet regression 200 epochs Huber Loss

the model is learning effectively and the use of Huber loss likely helps reduce the impact of outliers. After training the model it was tested on the test dataset which yielded the following results:

- MAE: 0.83
- MSE: 1.91
- RMSE: 1.38
- Coefficients of Determination: 0.16

The MAE indicates that on average, the model's predictions deviate from the true values by 0.83 units. This gives a direct and interpretable measure of prediction error which results in the fact that this is a significant error since the value range for the trust levels ranges from 0 to 7. The MSE of 1.91 describes the average squared deviation from the true values. It penalizes the larger errors more which in this case means that a significant amount of predictions actually have a larger error. The value of 1.38 for the RMSE normally suggests moderate error magnitude. The error values are on first sight relatively low, indicating the model performs reasonably well. However, error alone doesn't tell the whole story—context since the target variable scale is crucial. Taking this into account the model does make significant errors when predicting the values. Furthermore a value of 0.16 for the coefficient of determination indicates that only 16% of the variance in the data is explained by the model. This is relatively low, suggesting that the model struggles to capture patterns in the data effectively. Even though the errors of MAE, MSE and RMSE appear moderate, the low R-squared

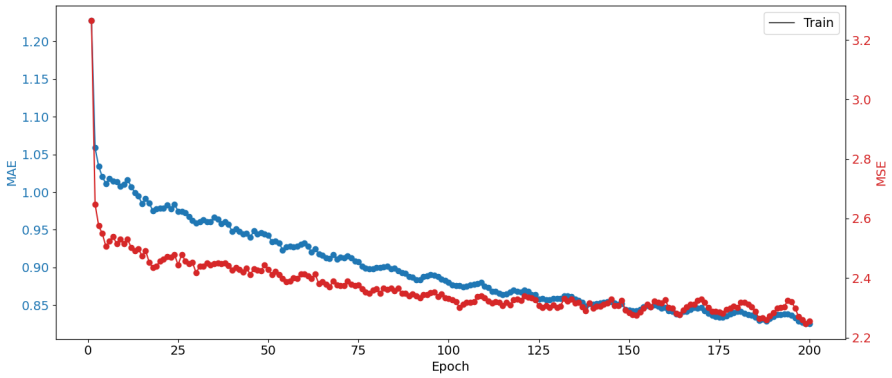
highlights that the model may not generalize well. Furthermore the data has a high degree of noise and non-linear relationships that the model isn't capturing well. This leaves a similar conclusion as for the first training iterations: The model might be too simple or lacks complexity to capture the underlying data patterns.

4.4.2.5. Fifth Training

The fifth training again features the change of the loss function. In the first training iterations the MSELoss and in the 4th training the Huber Loss was used. Now the relatively trivial L1 loss is used, which is defined in the following way 4.4.2.5:

$$L1 = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{4.5}$$

Hereby N is the number of data points, y_i is the true value of the i-th data point and $\hat{y}_i$ is the predicted value of the i-th data point. It therefore calculates the average absolute difference between the true values and the predicted values with the loss being always positive no matter if the model underestimates or overestimates the trust level with regard to the true value.
This training also involved 200 epochs, logging the metrics like MAE, MSE, RMSE, the coefficient of determination, the training loss and the validation loss, see A.8.5. After the training was executed the model wa tested on a test set yielding a MAE of 0.81, a MSE of 2.27, a RMSE of 1.51, and a Coefficient of Determination of -0.02. Additionally, the provided graph 4.11 illustrates the training process, plotting the MAE and MSE over 200 epochs. The MAE of 0.81 indicates that on average, the



The graph describes the progress of the MAE and MSE over the course of the training process for 200 epochs. For this training run the loss function was changed to the L1 Loss.

Figure 4.11.: 5th training EEGNet regression 200 epochs L1 Loss

predictions deviate by 0.81 units from the true trust levels. This error magnitude appears relatively low on the first sight but given the range of the target variable it has a significant impact on the outcome. The regression task involves predicting one trust value for a sample of the measurement. If this is used to classify the participants based on the regression outcome an error of 0.81 on the scale from 0 to 7 might be decisive whether the participant is assigned the correct class. The MSE places greater emphasis on larger errors. With a result of 2.27 it suggests the presence of some larger deviations in the predictions. The RMSE, at 1.51 further confirms those prediction errors. The Coefficient of Determination (R-squared) value of -0.02 indicates that the model fails to explain any meaningful variance in the target variable. The coefficient of determination is defined in the range from 0 to 1

whereas 0 means the model is not predicting the outcome and 1 means that the model is perfectly predicting the outcome. Important to point out is that the coefficient of determination is not bound in this range and can also become negative if for example the residual error is getting higher. A negative value means that the model's performance is slightly worse than that of a naive baseline predicting the mean of the target variable. This result suggests significant shortcomings in the model's ability to generalize to unseen data and calls into question the model's effectiveness in capturing the underlying relationships in the data.
The graph 4.11 provides additional insights and illustrates the MAE and MSE metrics logged during training. All metrics are included in table A.8.5 with the corresponding other graphs for the other metrics being illustrated in figure A.10. It exhibits a sharp decline in the initial epochs, indicating a rapid learning which is already characteristical for the training's done before. Over the subsequent epochs, the error metrics continue to decrease more gradually, suggesting the model is still optimizing its parameters but at a slower rate. The stability of the error curves towards the later epochs indicates that the model has converged. Despite the convergence observed in the training process, the poor test set performance points to potential overfitting or data/model limitations. Overfitting may have occurred if the model captured noise or incorrect patterns in the training data that did not generalize to the test set. Alternatively, the EEGNet architecture or the feature representations used may lack the capacity to adequately model the relationship between the EEG data and trust levels. Furthermore, the relatively low performance metrics might result from the raw EEG data being noisy resulting in a much more complex learning for the model to capture the important patterns and information..

4.5. Comparison of Results

For the comparison of the trained models the following initial configuration was used for the classification model 4.21 and adjusted throughout the trials:

Table 4.21.: Initial Training Configuration Classification

Parameter	Value
Criterion	`CrossEntropyLoss`
Optimizer	`torchoptim.SGD`
Learning Rate	`0.001`
warm_start	`default false`
train_split	`none`

The initial configuration of the regression approach looks slightly different because of the use of other loss functions and optimizer, 4.22:

Table 4.23 provides a detailed comparison of the ten training runs conducted on EEGNet, divided into five classification and five regression experiments. For classification the results for the five training setups with varying configurations are shown. In the first training run, errors were encountered, and no performance metrics were recorded. In the second run, the network achieved a balanced accuracy of 86.94% and an accuracy of 49.98%. The third training iteration included adjustments, such as the use of cross-entropy loss with class weights and the Adam optimizer, leading to a balanced accuracy of 80.11% and an accuracy of 86.76%. The fourth run implemented the same adjustments as the third but with a defined train split, resulting in slightly reduced performance metrics: 76.84% balanced accuracy and 79.17% accuracy. In the fifth training configuration, the same adjustments were retained, and balanced

Table 4.22.: Initial Training Configuration Regression

Parameter	Value
Criterion	`torch.nn.MSELoss`
Optimizer	`torch.optim.AdamW`
Learning Rate	`0.001`
Train Split	`predefined_split(val_dataset)`
Batch Size	`16`
Callbacks	`callbacks`

accuracy and accuracy improved to 82.14% and 82.01%, respectively. However, for all classification runs, no measures for loss or coefficient of determination were reported. For the regression tasks,

Table 4.23.: Comparison Training Results EEGNet Classification and Regression

Training	Epochs	Adjustments	bal acc	acc	MAE	MSE	RMSE	Coef of Det
Classification EEGNet								
1st	100	-	error	error	-	-	-	-
2nd	100	-	86.94%	49.98%	-	-	-	-
3rd	200	CELoss with weights Adam optimizer	80.11%	86.76%	-	-	-	-
4th	200	CELoss with weights Adam optimizer defined train split	76.84%	79.17%	-	-	-	-
5th	200	CELoss with weights Adam optimizer defined train split	82.14%	82.01%	-	-	-	-
Regression EEGNet								
1st	100	-	-	-	0.88	1.88	1.37	0.16
2nd	200	-	-	-	0.83	1.41	1.19	0.35
3rd	200	LR=0.01	-	-	0.89	1.84	1.36	0.15
4th	200	Huber Loss	-	-	0.83	1.91	1.38	0.16
5th	200	L1 Loss	-	-	0.81	2.27	1.51	-0.02

the table provides MAE, MSE, RMSE and coefficient of determination values across five runs. The first regression training achieved a MAE of 0.88, MSE of 1.88, RMSE of 1.37, and a coefficient of determination of 0.16. In the second training run, where the number of epochs was doubled to 200, the MAE decreased to 0.83, MSE dropped to 1.41, and RMSE reduced to 1.19, showing significant improvement, while the coefficient of determination increased to 0.35. In the third regression run, a LR adjustment to 0.01 was introduced. However, the loss metrics worsened slightly, with MAE increasing to 0.89, MSE to 1.84, RMSE to 1.36, and a slight reduction in the coefficient of determination to 0.15. The fourth training applied the Huber loss, resulting in an MAE of 0.83, MSE of 1.91, and RMSE of 1.38, which was marginally worse than previous runs, while the coefficient of determination remained at 0.16. The fifth regression training utilized L1 loss and yielded the poorest results among all regression runs, with an MAE of 0.81, MSE of 2.27, RMSE of 1.51, and a negative coefficient of determination (-0.02), indicating that the model underperformed on the regression task.

Overall, the classification experiments show improvements with the inclusion of adjustments such as cross-entropy loss and optimizer changes, while the regression experiments illustrate varying performance depending on the loss function and training configurations. Despite some success in minimizing errors and improving accuracy or coefficients, the inconsistencies in results suggest the need for further experimentation and parameter tuning for both classification and regression tasks.

5. Discussion

To sum up the results of this study provide significant insights into the effectiveness of various deep learning and machine learning approaches for trust level estimation in social human-robot interaction using the provided EEG data of [1]. The experimental outcomes indicate that advanced models, particularly the EEGNet, outperform simpler architectures in capturing the nuances of trust dynamics which are included in the data. There is a difference in performance regarding the classification and regression approach using the EEGNet. Whilst the classification training with two trust level classes yields good results after finding an appropriate configuration of the model and classifier specifications, the regression results are moderate regarding the performance on the test dataset and during the course of the training.
In general the findings align with the hypothesis that leveraging domain-specific neural network architectures yields better performance on EEG datasets, which are inherently complex due to their temporal and spatial characteristics. A notable observation from the results is the comparative robustness of EEGNet. Its ability to effectively handle limited datasets, a challenge in EEG-based research, underscores the suitability of compact convolutional architectures for this domain. EEGNet's architecture is specifically designed for brain-computer tasks like this. It demonstrated consistent accuracy across classification tasks, outperforming both feed-forward networks and traditional machine learning methods like SVM classifier or the analysis of the clusters created with k-means. This reinforces the importance of leveraging architectures tailored for EEG data, especially when data is scarce or noisy. On the other hand, RNNs, particularly those with LSTM units, showcased decent performance in capturing temporal dependencies within EEG signals. The results highlight the ability of RNNs to model the sequential nature of EEG data effectively. This accounts particularly when augmented with data preprocessing techniques such as artifact removal and frequency filtering. While the computational requirements of RNNs were higher than simpler architectures, the trade-off is justified by their improved performance. Traditional machine learning approaches, while useful as a baseline, fell short in terms of accuracy and generalization. Especially since the dataset is strongly imbalanced, including much more high trust instances than low trust instances. Furthermore the SVMs classifier and k-means clustering struggled with the high dimensionality of the EEG data and the general complexity of the dataset. Additionally these methods require extensive feature engineering, which introduces additional complexity and potential biases. This especially becomes evident when looking at all the pre-processing steps needed in order to obtain processable features. All the pre-processing steps introduced in paper [1] were recreated and implemented as described in section 3.1.1.

Every single processing step adds further uncertainties and insecurities to the data processed by the traditional machine learning models, the feed-forward binary classification neural network and the created RNN. There could be rounding errors in calculating the power spectral density or insecurities when transforming the temporal data through the FFT into the frequency domain. On the other hand techniques like FFT-based preprocessing and power spectral density analysis are instrumental in enhancing classification accuracy for the traditional models by ensuring that only the most relevant features were fed into the models since noisy and high dimensional inputs would have been too complex

for the models to filter and process. Furthermore there are insecurities when it comes to the structure of the dataset which are described in section 3.1. Hereby the inaccurate time stamps of the details.csv file impose the greatest uncertainties. By contrast, the end-to-end learning approach of neural networks, particularly EEGNet, bypassed the need for manual feature extraction and proved to be more effective in capturing the underlying patterns in the data. This results from the fact that the EEGNet and the other models incorporated in the **braindecode** API are able to work with the highly complex and noisy raw EEG data. This way the models are able to process and use all the encapsulated information which is included in the micro-volt measurements of each EEG channel. On the other hand it becomes evident that also the EEGNet struggles at some points with the complexity and noise of the raw data. Especially the usage of the architecture in the regression task does not yield very good results since the MSE, MAE and RMSE indicate a relatively high discrepancy to the real trust scores throughout the five executed training processes. It needs to be argued if the EEGNet in its used configuration is suitable for handling the regression task with the underlying data.
A huge challenge throughout the study of different techniques and methods was the always underlying class imbalance of the dataset which influences the outcomes. The data augmentation approach which was used is the SMOTE technique. This data augmentation techniques helps to balance the dataset to mitigate the existing class imbalance. It generates synthetic samples by interpolating between existing minority class samples which are the low trust samples. A major drawback is, that this can create synthetic data points that overlap with the other high trust class, leading to ambiguity and making it harder for the model to differentiate between classes. Furthermore the model's may overfit to the synthetic samples, especially because the minority class of low trust instances was already sparse. This can result in reduced generalization on unseen data. Additionally it might create synthetic samples that are far from the true distribution of the low trust class, introducing outliers that potentially confuse the model. Also the interpolation process assumes that the feature space between the low trust samples is linearly distributed, which may not always be true. This distortion can lead to less meaningful decision boundaries which distort the decision making process.
The study's limitations, including the relatively small dataset and potential inconsistencies in labeling due to time alignment issues, must be acknowledged. Furthermore the results and analysis of the findings can be enhanced when utilizing more computational resources in order to conduct a full hyperparameter search on the used models. This becomes essential when wanting to improve performance and convergence. Training three of the EEGNet regression model illustrates this in a very practical way. Here the LR was adjusted to a higher value leading to oscillating behaviour of the metrics. This happens when the learning rate is too high and the optimizer is overshooting the minimum of the loss function during updates. Instead of converging smoothly, the loss oscillates or even increases as the optimization process fails to settle into a stable minimum. This example proves that all adjustments can have a certain influence on the model training and creation process which results in the necessity of conducting a hyperparameter search in the future. Despite these challenges and short comings of some performances of some of the models, the findings provide a compelling argument for prioritizing domain-specific architectures like EEGNet for trust classification tasks. Future work could explore the integration of attention mechanisms, as seen in transformer-based models, to further enhance the interpretability and accuracy of EEG-based trust assessments.

In conclusion, the results identify EEGNet as the best approach for trust level estimation, combining robust performance with computational efficiency. Hereby the classification approach yields superior results compared to the regression approach. Especially training three and training five incorporating the Adam optimizer yield good classification results on the test data. Therefore the classification

results seem to appear more stable in their convergence and ability to handle unseen data after training. The regressor models also converge but maintain an insecurity which is too high for accurately and consequently estimating the right trust score. This is particularly the case because the scale of the questionnaire trust scores only ranges from 0 to 7 which makes an error of 0.8 to 1.4 significant regarding the outcome. All this being mentioned, the trust scores result from a manually evaluated and created questionnaire which includes subjective behaviour of the participants resulting in the general question if the format and setup of the underlying experiment is suitable for the objective or if the process of it should be adapted to yield more robust results. While RNNs also performed on a decent basis, their higher computational costs and sensitivity to hyperparameter tuning make them less practical for real-world applications. Traditional machine learning models, though informative as a baseline, were outperformed by the deep learning models, reinforcing the need for specialized architectures in EEG-based trust estimation. On the other hand, the traditional clustering and classification methods used provide a good baseline for qualitatively analyzing the distribution and visual patterns of the underlying data structure. Further adaptions are needed to improve the performance of those models since the balanced accuracy on the test data always ranged around the value of 0.5. This in comparison with the findings of [1] is a worse result than obtained in the original paper. To add upon this fact, it does not become evident if the accuracy measures used in the baseline paper [1] are taking into account the highly imbalanced nature of the dataset. All these findings contribute to the growing body of research in human-robot interaction and pave the way for further advancements in trust modeling using neural networks. The **braindecode** API is providing further additional models like the HybridNet which can be utilized on the same dataset comparing them to the introduced models in this study.

6. Outlook

The conducted study successfully compares different concepts of obtaining a classifier or regressor model for accurately inferring trust levels of humans interacting with robots. As robots become more and more important for humans in their day to day life, it is crucial for the deployment of such systems that humans interacting and relying on those robots trust them with their abilities. The conducted experiment in [1] involves the interaction style of playing a game together. Thereby it is questionable if the EEG measurements obtained from this kind of experiment are representative for tasks which involve more complex behaviour patterns like social interaction with patients in a clinic, for example. One of the main improvements could therefore be made in evolving the scope of the experiment to other domain, more complex areas of application or different tasks when interacting with the opposing humans. This way the trained models in this study could be validated on EEG data of different scenarios which could lead to more robust and accurate results. Furthermore it could lead to new knowledge discovery in terms of the suitability of the models for the different domains. An example could be that a specifically trained model on EEG measurements obtained from humans playing games with a robot is not suitable for estimating the trust a factory worker puts into an industrial robot used for welding or transporting parts.
Additionally to the scope and conditions in which the experiment is carried out and the data is obtained, other further points of research regarding the models and methods can be pointed out. Firstly this study does not deal with the optimization of the hyperparamters of the studied models. This is mainly because hyperparamter search is computationally very expensive. Since in this context the time period in which the study was conducted did not leave the possibility of executing the hyperparameter search on a GPU cluster, only some experiments on a local machine were executed. In future works which build up on this study it should be aimed for optimizing the model configuration to obtain higher accuracy and performance gains. In this context it needs to be mentioned that a hyperparameter search normally takes into account several parameters with different levels of values which need to be cross-validated for each possible combination of parameter values. Furthermore each combination should be used several times to obtain a statistical valid and robust result. A small example of a hyperparameter search including the LR, batch size and activation function with each parameter having three different levels can be:

- LR: 0.1, 0.01 and 0.001
- batch size: 16, 32 and 64
- activation function: sigmoid, ReLU and SiLU

This configuration would lead to 27 different combinations. If for each combination a total amount of ten full training runs is necessary, there would be 270 full training runs which would need to be completed. Such an amount of training process is only executable in a reasonable amount of time using GPUs.

Another open point which could lead to improvement would be to adjust the custom models of the feed-forward Neural Network and the created Recurrent Neural Network in a way to be able to handle raw EEG data inputs into the model. This would lead the possibility to execute a forward pass on the feature and information rich voltage measurements of the different EEG channels. It remains to be seen if the two models would increase in accuracy when working with the raw data but it certainly could solve the currently existing learning problem of the two models by having a higher density and variety in information.
The **braindecode** framework gives a lot of opportunities to create models and train them on custom datasets and architectures. As part of the study the HybridNet architecture was analyzed and implemented on a rudimentary. That is why a detailed exploration of potential configuration was not conducted in this study. However, this approach offers potential for classification as well as for the regression approach and should be evaluated rigorously. Furthermore also other pre-implemented models of **braindecode** should be adapted to the problem and studied in more detail. Models like Deep4Net or EEGResNet could be beneficial for obtaining a more robust framework for estimating trust levels.
In addition more detailed experiments with the classifiers can be planned and executed. There is for example the possibility to adjust the threshold values for the label generation after a different method. Now the threshold values for the phase 3 and 4 of the experiment are the mean values of the questionnaire results of all participants for the corresponding phase. It can be argued if it would be more accurate to take the median value of the results or adjust the value based on the underlying standard deviation or variance. That way the problem of the resulting class imbalances could be resolved or reduced. The effect of the different annotations procedures of the feature vectors and there resulting consequences on the training and validation process and results should be monitored and analyzed in detail afterwards.
Another point of interest which could lead to different results and applicable methodologies is experimenting with different concepts or ways to do the manual feature extraction in the pre-processing section. Here the windowing, the FFT and the kind of statistical measure which is calculated have an influence on which information are kept and which information are surpressed through the transformations which are done.
Since the class distribution is highly imbalanced in the dataset which is used for training the models, it is recommended to try different augmentation approaches to reduce the influence of the class imbalance in different ways. The currently used SMOTE approach yields estimated feature vectors for the minority class but having only estimated values in this highly sensible domain could lead to new insecurities which lead to inaccuracy. Therefore it is beneficial to compare and try different approaches while monitoring their effects on the model performances.
At last one objective for future comparisons and advancements should be to get a more detailed look into the procedures of the baseline paper to validate whether the results which are obtained in [1] are achievable with the machine learning models used. Especially since the corresponding and provided metrics of the paper seem to be based upon the accuracy without taking into account the imbalance of the dataset. A more detailed overview on how the results were obtained and which potential mismatches exist could lead to a better common understanding of the problem and the underlying structure of the data.
In summary, all the analyzed and studied concepts still leave room for further investigations especially regarding the performance enhancements of the models. The **braindecode** API already provides a valuable base on which EEG data analysis can be driven forward. But it needs to be further studied

if the provided models can be adjusted in ways so that they gain improvements in their suitability to the problem of estimating trust levels from humans in robots. Hereby the challenge of estimating trust in intelligent machines and agents will gain in importance over the next years so that future findings could contribute to a higher quality of the development of robotic systems used directly in interactions with humans.

List of abbreviations and formula symbols

EEG Electroencephalogram

RAM Random-Access Memory

CPU Central Processing Unit

LR Learning Rate

MSE Mean Squared Error

MAE Mean Absolute Error

RMSE Root Mean Squared Error

HCI Human Computer Interaction

HRI Human Robot Interaction

GPT Generative Pretrained Transformer

SMOTE Synthetic Minority Oversampling Technique

SSE sum of squared errors

SVM Support Vector Machine

DC Direct Current

CNN Convolutional Neural Network

RNN Recurrent Neural Network

FFT Fast Fourier Transform

DFT Discrete Fourier Transformation

LLM Large Language Model

SVM Support Vector Machine

csv Comma-separated values

LSTM Long-Short Term Memory

GRU Gated Recurrent Unit

BCI brain-computer interface

ReLU Rectified Linear Unit

GPU Graphical Processing Unit

SGD Stochastic Gradient Descent

Bibliography

[1] G. Campagna and M. Rehm, "Trust Assessment with EEG Signals in Social Human-Robot Interaction," (Singapore), pp. 33–42, Springer Nature Singapore, 2024.

[2] M.-H. Lee, O.-Y. Kwon, Y.-J. Kim, H.-K. Kim, Y.-E. Lee, J. Williamson, S. Fazli, and S.-W. Lee, "Eeg dataset and openbmi toolbox for three bci paradigms: an investigation into bci illiteracy," *GigaScience*, vol. 8, 01 2019.

[3] V. J. Lawhern, A. J. Solon, N. R. Waytowich, S. M. Gordon, C. P. Hung, and B. J. Lance, "Eegnet: a compact convolutional neural network for eeg-based brain–computer interfaces," *Journal of Neural Engineering*, vol. 15, p. 056013, July 2018.

[4] C. Bartneck, F. E. T. Belpaeme, M. K. T. Kanda, and S. Sabanovic, "Human-robot interaction – an introduction. (2nd edition)," 2024. Accessed: 2025-01-01.

[5] M. Abo-Zahhad, S. Ahmed, and S. N. Seha, "A new eeg acquisition protocol for biometric identification using eye blinking signals," *International Journal of Intelligent Systems and Applications (IJISA)*, vol. 07, pp. 48–54, 05 2015.

[6] J. W. Kim, A. Alaa, and D. Bernardo, "EEG-GPT: Exploring Capabilities of Large Language Models for EEG Classification and Interpretation," *arXiv [q-bio.QM]*, 2024.

[7] K. F. Firoz, Y. Seong, and S. Oh, "A neurological approach to classify trust through EEG signals using machine learning techniques," pp. 1–6, 2022.

[8] K. Akash, W.-L. Hu, N. Jain, and T. Reid, "A Classification Model for Sensing Human Trust in Machines Using EEG and GSR," *ACM Trans. Interact. Intell. Syst.*, vol. 8, Nov. 2018. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[9] S. Roy, I. Kiral-Kornek, and S. Harrer, "ChronoNet: A Deep Recurrent Neural Network for Abnormal EEG Identification," May 2018. arXiv:1802.00308 [eess].

[10] S. Oh, Y. Seong, S. Yi, and S. Park, "Investigation of human trust by identifying stimulated brain regions using electroencephalogram," *ICT Express*, vol. 8, no. 3, pp. 363–370, 2022.

[11] Y. Yang, Q. Wu, M. Qiu, Y. Wang, and X. Chen, "Emotion recognition from multi-channel eeg through parallel convolutional recurrent neural network," in *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7, 2018.

[12] G. Xiao, M. Ye, B. Xu, Z. Chen, and Q. Ren, "4d attention-based neural network for eeg emotion recognition," 2021.

[13] Z. Wang, Y. Wang, C. Hu, Z. Yin, and Y. Song, "Transformers for eeg-based emotion recognition: A hierarchical spatial information learning model," *IEEE Sensors Journal*, vol. PP, pp. 1–1, 03 2022.

[14] J. Li, H. Hua, Z. Xu, L. Shu, X. Xu, F. Kuang, and S. Wu, "Cross-subject eeg emotion recognition combined with connectivity features and meta-transfer learning," *Computers in Biology and Medicine*, vol. 145, p. 105519, 2022.

[15] R. T. Schirrmeister, J. T. Springenberg, L. D. J. Fiederer, M. Glasstetter, K. Eggensperger, M. Tangermann, F. Hutter, W. Burgard, and T. Ball, "Deep learning with convolutional neural networks for eeg decoding and visualization," *Human Brain Mapping*, aug 2017.

[16] SciPy Developers, *SciPy Signal Processing - Hann Window*, 2024. Accessed: 2024-12-09.

[17] J. Brownlee, "A gentle introduction to k-fold cross-validation," 2024. Accessed: 2024-12-13.

[18] A. Pandey, "The role of inertia in k-means clustering," 2024. Accessed: 2025-01-03.

A. Appendix

A.1. Questionnaire

MDMT Version Date: 2019-04-01

Please rate the robot using the scale from 0 (Not at all) to 7 (Very). If a particular item does not seem to fit the robot in the situation, please select the option that says “Does Not Fit.”

	Not at all 0	1	2	3	4	5	6	Very 7	Does Not Fit
Reliable	○	○	○	○	○	○	○	○	○
Sincere	○	○	○	○	○	○	○	○	○
Capable	○	○	○	○	○	○	○	○	○
Ethical	○	○	○	○	○	○	○	○	○
Predictable	○	○	○	○	○	○	○	○	○
Genuine	○	○	○	○	○	○	○	○	○
Skilled	○	○	○	○	○	○	○	○	○
Respectable	○	○	○	○	○	○	○	○	○
Someone you can count on	○	○	○	○	○	○	○	○	○
Candid	○	○	○	○	○	○	○	○	○
Competent	○	○	○	○	○	○	○	○	○
Principled	○	○	○	○	○	○	○	○	○
Consistent	○	○	○	○	○	○	○	○	○
Authentic	○	○	○	○	○	○	○	○	○
Meticulous	○	○	○	○	○	○	○	○	○
Has integrity	○	○	○	○	○	○	○	○	○
	Not at all 0	1	2	3	4	5	6	Very 7	Does Not Fit

A.2. Code Label Assignment

```
# do the labeling of each row depending on the details.csv file
details = pd.read_csv('data/details/details.csv', sep = ';')
print(details.head())
details_participant_ID = details[details['id'] == participant_ID] # df with details of the participant
if details_participant_ID.empty:
    print(f"No data found for participant ID: {participant_ID}")
else:
    print(f"Filtered data for participant {participant_ID}:")
print(details_participant_ID)
details_participant_ID.info()
sum_time = 0
for index, row in details_participant_ID.iterrows():
    try:
        # keep track of the time
        start_time = datetime.strptime(details.iloc[index]['start_clock'], '%H:%M')
        end_time = datetime.strptime(details.iloc[index]['finish_clock'], '%H:%M')
        time_diff_phase_in_seconds = (end_time - start_time).total_seconds()
        sum_time += time_diff_phase_in_seconds

        current_phase = row['phase']
        # label the feature rows in alpha_features_df with the labels of phase 3 and 4 from details.csv
        # if the time of the row is between the start and end time of phase 3, label it as phase 3
        # if the time of the row is between the start and end time of phase 4, label it as phase 4
        # if the time of the row is not between the start and end time of phase 3 and 4, label it not at all
        for index_alpha, row_alpha in alpha_features_df.iterrows():
            timestamp_alpha = alpha_features_df.index[index_alpha]
            if sum_time - time_diff_phase_in_seconds <= timestamp_alpha <= sum_time:
                if current_phase == 3: # if we are in phase 3
                    if details.iloc[index]['trust_score_binary'] == "high": # we check if trust is labeled high
                        if label_regression == True: # set the scorea as a label for training a regression model
                            alpha_features_df.at[index_alpha, 'label'] = details.iloc[index]['trust_score']
                        else:
                            alpha_features_df.at[index_alpha, 'label'] = 1
                    elif details.iloc[index]['trust_score_binary'] == "low": # if trust is labeld low
                        if label_regression == True: # set the scorea as a label for training a regression model
                            alpha_features_df.at[index_alpha, 'label'] = details.iloc[index]['trust_score']
                        else: # otherwise set it to the corresponding binary label
                            alpha_features_df.at[index_alpha, 'label'] = 0
                    else : # this is the case where the trust_score is not labeled correctly so we set a own threshold
                        if details.iloc[index]['trust_score'] >= 4.8:
```

```
                            if label_regression == True: # set the scorea as a label for training a regression model
                                alpha_features_df.at[index_alpha, 'label'] = details.iloc[index]['trust_score']
                            else: # otherwise set it to the corresponding binary label
                                alpha_features_df.at[index_alpha, 'label'] = 1
                        else: # if the trust score is below 4.8 we need to set it to 0 as a binary label
                            if label_regression == True: # set the scorea as a label for training a regression model
                                alpha_features_df.at[index_alpha, 'label'] = details.iloc[index]['trust_score']
                            else: # otherwise set it to the corresponding binary label
                                alpha_features_df.at[index_alpha, 'label'] = 0
                    elif current_phase == 4:
                        if details.iloc[index]['trust_score_binary'] == "high":
                            if label_regression == True: # set the scorea as a label for training a regression model
                                alpha_features_df.at[index_alpha, 'label'] = details.iloc[index]['trust_score']
                            else: # otherwise set it to the corresponding binary label
                                alpha_features_df.at[index_alpha, 'label'] = 1
                        elif details.iloc[index]['trust_score_binary'] == "low":
                            if label_regression == True: # set the scorea as a label for training a regression model
                                alpha_features_df.at[index_alpha, 'label'] = details.iloc[index]['trust_score']
                            else: # otherwise set it to the corresponding binary label
                                alpha_features_df.at[index_alpha, 'label'] = 0
                        else : # this is the case where the trust_score is not labeled correctly so we set a own threshold
                            if details.iloc[index]['trust_score'] >= 4.8:
                                if label_regression == True: # set the scorea as a label for training a regression model
                                    alpha_features_df.at[index_alpha, 'label'] = details.iloc[index]['trust_score']
                                else: # otherwise set it to the corresponding binary label
                                    alpha_features_df.at[index_alpha, 'label'] = 1
                            else:
                                alpha_features_df.at[index_alpha, 'label'] = 0
                    if flag_all_phases == True:
                        if current_phase == 1 or current_phase == 2 or current_phase == 5:
                            if details.iloc[index]['trust_score_binary'] == "high":
                                if label_regression == True: # set the scorea as a label for training a regression model
                                    alpha_features_df.at[index_alpha, 'label'] = details.iloc[index]['trust_score']
                                else: # otherwise set it to the corresponding binary label
                                    alpha_features_df.at[index_alpha, 'label'] = 1
                            elif details.iloc[index]['trust_score_binary'] == "low":
                                if label_regression == True: # set the scorea as a label for training a regression model
```

```
                                alpha_features_df.at[index_alpha, 'label'] = details.iloc[index]['trust_score']
                            else: # otherwise set it to the corresponding binary label
                                alpha_features_df.at[index_alpha, 'label'] = 0
                        else :
                            if details.iloc[index]['trust_score'] >= 4.8:
                                if label_regression == True: # set the scorea as a label        for training a regression model
                                    alpha_features_df.at[index_alpha, 'label'] = details.iloc[index]['trust_score']
                                else: # otherwise set it to the corresponding binary label
                                    alpha_features_df.at[index_alpha, 'label'] = 1
                            else:
                                if label_regression == True: # set the scorea as a label for training a regression model
                                    alpha_features_df.at[index_alpha, 'label'] = details.iloc[index]['trust_score']
                                else: # otherwise set it to the corresponding binary label
                                    alpha_features_df.at[index_alpha, 'label'] = 0

            sum_time += 60
        except Exception as e:
            print(f"Error processing row {index}: {e}")

# Save the processed EEG data to a new CSV file
if flag_all_phases == True:
    if label_regression == False:
        if not os.path.exists(f'data/labelingAll/AllPhases_labeled_Alpha_{file_path.replace(data_path, '')}.csv'):
            output_file_alpha_features =(f'data/labelingAll/AllPhases_labeled_Alpha_{file_path.replace(data_path, '')}.csv')
            alpha_features_df.to_csv(output_file_alpha_features)
    elif label_regression == True:
        if not os.path.exists(f'data/regr_labelingAll/AllPhases_labeled_Alpha_regression_{file_path.replace(data_path, '')}.csv'):
            output_file_alpha_features = (f'data/regr_labelingAll/AllPhases_labeled_Alpha_regression_{file_path.replace(data_path, '')}.csv')
            alpha_features_df.to_csv(output_file_alpha_features)
elif flag_all_phases == False:
    if label_regression == False:
        if not os.path.exists(f'data/labeling3and4/labeled_Alpha_{file_path.replace(data_path, '')}.csv'):
            output_file_alpha_features = (f'data/labeling3and4/labeled_Alpha_{file_path.replace(data_path, '')}.csv')
            alpha_features_df.to_csv(output_file_alpha_features)
    elif label_regression == True:
        if not os.path.exists(f'data/regr_labeling3and4/labeled_Alpha_regression_{file_path.replace(data_path, '')}.csv'):
            output_file_alpha_features = (f'data/regr_labeling3and4/labeled_Alpha_regression_{file_path.replace(data_path, '')}.csv')
            alpha_features_df.to_csv(output_file_alpha_features)
```

A.3. Code Extract Window Features

```
def extract_window_features(data, fs, band, n_channels):
    features = []
    # Iterate through windows
    for start in range(0, len(data), fs):
        end = start + fs
        if end > len(data):
            break  # Ignore the last window if it's incomplete
        # Slice the window
        window = data[start:end]
        # Compute Alpha band power
        alpha_power = [compute_alpha_psd_window(window[channel], fs, band) for channel in range(n_channels)]
        # Compute statistical features on Alpha power across the 14 channels
        alpha_mean = np.mean(alpha_power)
        alpha_peak = np.max(alpha_power)
        alpha_median = np.median(alpha_power)
        alpha_std = np.std(alpha_power)
        alpha_kurtosis = kurtosis(alpha_power)
        # Store the features for this window
        features.append([alpha_mean, alpha_peak, alpha_median, alpha_std, alpha_kurtosis])
    return np.array(features)
```

A.4. Code Calculation Power Spectral Density

```
def compute_alpha_psd_window(data, fs, band):
    # Welch's method to compute PSD
    freqs, psd = welch(data, fs=fs, nperseg=fs)
    # Find indices of the band
    band_idx = np.logical_and(freqs >= band[0], freqs <= band[1])
    # Return the average power in the alpha band
    return np.sum(psd[band_idx])
```

A.5. Label Counts per Participant

Participant_ID	label	count
1.	1.0	422
10	1.0	422
11	1.0	422
12	1.0	422
13	1.0	344
14	1.0	422
15	1.0	422
16	1.0	301
17	1.0	396
18	1.0	422
19	1.0	422
2.	1.0	302
20	1.0	302
21	1.0	302
3.	0.0	362
4.	0.0	362
5.	1.0	422
6.	0.0	301
6.	1.0	121
7.	0.0	422
8.	1.0	362
9.	1.0	606

Table A.1.: Distribution of high and low trust labels per participant for the labeling of phase 3 and 4.

A.6. Confusion Matrix EEGNet

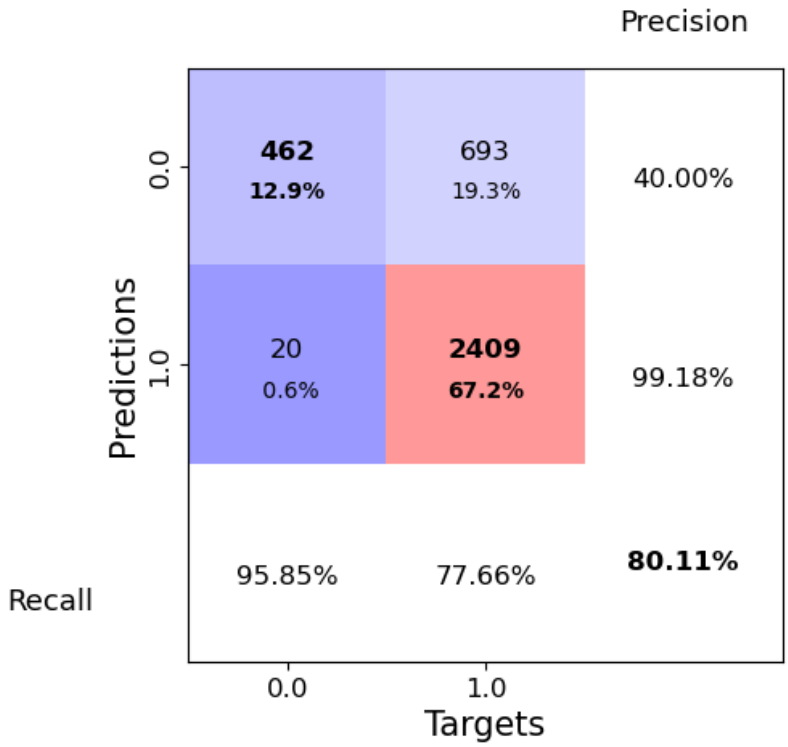


Figure A.1.: Confusion Matrix EEGNet Training 200 Epochs

Confusion Matrix of the labeled feature vectors classified as either high or low trust. As model the EEGNetv4 was trained for 200 epochs.

A.7. Training EEGNet Classification

A.7.1. First Training

ID	1	
Date	26.11.2024, 21:10	
Model Definition	n_outputs=n_classes(=2), n_chans=n_channels(=14), n_times=sampling_rate(=128), final_conv_length="auto"	
Classifier Definition	model, criterion=CrossEntropyLoss, default optimizer = optim.SGD, lr=0.001, warm_start = default false, train_split=None	
Training Epochs		
Epoch	**Train Loss**	**Duration (s)**
1	0.4451	46.5935
2	0.4296	45.6062
3	0.4227	44.6479
4	0.4252	44.6692
5	0.4220	44.9603
6	0.4194	44.8162
7	0.4190	45.5331
8	0.4214	48.8435
9	0.4143	47.5905
10	0.4173	47.6255
11	0.4147	55.4800
12	0.4121	50.4059
13	0.4114	58.1556
14	0.4161	58.6702
15	0.4107	73.5779
16	0.4070	56.5948
17	0.4111	54.4595
18	0.4102	54.7741
19	0.4106	56.9252
20	0.4082	48.9231
21	0.4108	52.7796
22	0.4096	55.5295
23	0.4081	57.1387
24	0.4107	51.4151
25	0.4077	54.1964
26	0.4074	49.4532

27	0.4068	47.8224
28	0.4088	45.0395
29	0.4091	46.6384
30	0.4061	55.3993
31	0.4076	50.5312
32	0.4073	45.7391
33	0.4055	44.7265
34	0.4055	44.1956
35	0.4036	43.7981
36	0.4063	44.4256
37	0.4054	44.4894
38	0.4040	43.7003
39	0.4030	43.7844
40	0.4038	44.4693
41	0.4043	44.6169
42	0.4023	44.2214
43	0.4038	44.2417
44	0.4052	44.1596
45	0.4024	43.8326
46	0.4034	44.0051
47	0.4047	44.1303
48	0.4031	44.0878
49	0.4044	43.9575
50	0.4016	44.4446
51	0.4032	44.1829
52	0.4027	44.1096
53	0.4026	44.0343
54	0.4018	44.0197
55	0.4003	44.1274
56	0.4024	44.1459
57	0.4034	44.7253
58	0.4033	44.1715
59	0.4018	44.0859
60	0.4003	43.8714
61	0.4042	43.6244
62	0.4033	43.9167
63	0.4003	44.4929
64	0.4005	43.6588
65	0.4004	43.9899
66	0.4010	43.6858
67	0.4006	44.5116
68	0.3999	43.7750
69	0.4006	44.0705

70	0.4005	43.6604
71	0.4010	44.1514
72	0.3994	44.4322
73	0.4013	44.9681
74	0.3985	43.8567
75	0.4009	43.7412
76	0.4007	44.5173
77	0.4004	43.7657
78	0.4009	44.8190
79	0.4005	44.7134
80	0.3992	44.9435
81	0.3993	43.9682
82	0.4019	44.0472
83	0.3996	43.9912
84	0.3981	44.4593
85	0.3993	44.2538
86	0.3980	43.8666
87	0.3979	43.3679
88	0.3986	43.6078
89	0.4007	44.0145
90	0.3999	44.1023
91	0.3995	43.6931
92	0.4000	43.4881
93	0.3999	44.1085
94	0.3999	43.6103
95	0.4006	43.7603
96	0.3988	43.9974
97	0.3978	44.4140
98	0.3990	44.3708
99	0.3978	44.3158
100	0.3986	44.0588
Accuracy	**test failure**	
balanced Accuracy	**test failure**	

Table A.2.: First training results of EEGNet for Classification

A.7.2. Second Training

ID	2
Date	27.11.2024, 8:30

Model Definition	n_outputs=n_classes(=2), n_chans=n_channels(=14), n_times=sampling_rate(=128), final_conv_length="auto"
Classifier Definition	model, criterion = CrossEntropyLoss, default optimizer = optim.SGD, lr = 0.001, warm_start = default false, train_split = None

Training Epochs

Epoch	Train Loss	Duration (s)
1	0.4333	49.0780
2	0.4097	48.7454
3	0.4086	50.1579
4	0.4085	59.3479
5	0.4067	54.9763
6	0.4072	52.0280
7	0.4026	53.1266
8	0.4039	54.1735
9	0.4039	50.4954
10	0.4026	50.8632
11	0.4023	47.8230
12	0.4032	50.0769
13	0.4014	49.2444
14	0.4009	48.9826
15	0.4010	52.7950
16	0.3988	56.7532
17	0.3980	55.8431
18	0.3997	57.6167
19	0.4005	65.9659
20	0.3976	54.0880
21	0.3978	54.3221
22	0.3978	53.2870
23	0.3988	53.0759
24	0.3966	51.7625
25	0.3976	52.0893
26	0.3955	52.4874
27	0.3975	47.6661
28	0.3938	49.0316
29	0.3972	47.3388
30	0.3968	47.3707
31	0.3964	47.3729

32	0.3974	47.9049
33	0.3952	45.3927
34	0.3958	44.1298
35	0.3946	47.9527
36	0.3942	45.5144
37	0.3928	45.9739
38	0.3928	46.2420
39	0.3946	46.4131
40	0.3919	45.7158
41	0.3908	45.7051
42	0.3940	45.7205
43	0.3922	45.1028
44	0.3901	45.3356
45	0.3912	45.5946
46	0.3920	45.7280
47	0.3920	45.2399
48	0.3911	45.1781
49	0.3899	45.0076
50	0.3898	46.1482
51	0.3889	49.8327
52	0.3901	49.5468
53	0.3894	48.6769
54	0.3895	48.7764
55	0.3877	48.7220
56	0.3871	49.5508
57	0.3885	46.8132
58	0.3882	46.1110
59	0.3865	46.0929
60	0.3880	45.7723
61	0.3860	47.5518
62	0.3841	51.3061
63	0.3858	51.7891
64	0.3870	52.6891
65	0.3840	48.4115
66	0.3867	46.6141
67	0.3846	46.8178
68	0.3850	49.0911
69	0.3853	49.4227
70	0.3845	50.4727
71	0.3861	50.3136
72	0.3847	48.3528
73	0.3849	45.2110
74	0.3842	44.5788

75	0.3833	47.7538
76	0.3853	51.0704
77	0.3843	49.3300
78	0.3827	52.4029
79	0.3830	52.0022
80	0.3831	46.8790
81	0.3837	47.5218
82	0.3832	47.8283
83	0.3824	49.5793
84	0.3826	57.8761
85	0.3844	47.1582
86	0.3818	45.9214
87	0.3822	48.6107
88	0.3814	50.8433
89	0.3825	45.7865
90	0.3793	48.7605
91	0.3817	47.3653
92	0.3824	48.7194
93	0.3794	49.7542
94	0.3800	49.9537
95	0.3816	49.5283
96	0.3805	49.8707
97	0.3816	51.8964
98	0.3823	49.1636
99	0.3818	46.5855
100	0.3817	45.9836
Accuracy	**86.94%**	
balanced Accuracy	**49.98%**	

Table A.3.: Second training results of EEGNet for Classification

A.7.3. Third Training

ID	3
Date	27.11.2024, 20:30
Model Definition	n_outputs=n_classes(=2), n_chans=n_channels(=14), n_times=sampling_rate(=128), final_conv_length="auto"

Classifier Definition	model, criterion=CrossEntropyLoss(weight=class_weights), default optimizer = optim.Adam, lr=0.001, batch_size=16, callbacks=callbacks, warm_start = default false, train_split=None			
Training Epochs				
Epoch	**Train Bal Acc**	**Train Loss**	**LR**	**Duration (s)**
1	0.5445	0.6877	0.0010	172.88
2	0.5869	0.6579	0.0009	181.99
3	0.6073	0.6436	0.0008	176.91
4	0.6065	0.6345	0.0005	173.98
5	0.6180	0.6375	0.0003	189.80
6	0.6178	0.6302	0.0001	178.60
7	0.6242	0.6303	0.0000	196.29
8	0.6239	0.6299	0.0001	197.34
9	0.6199	0.6296	0.0003	219.05
10	0.6188	0.6327	0.0005	208.37
11	0.6236	0.6261	0.0008	223.78
12	0.6260	0.6306	0.0009	210.67
13	0.6235	0.6251	0.0010	206.59
14	0.6427	0.6155	0.0009	212.75
15	0.6420	0.6129	0.0008	214.92
16	0.6559	0.6001	0.0005	227.42
17	0.6549	0.5968	0.0003	221.42
18	0.6522	0.5974	0.0001	220.75
19	0.6722	0.5896	0.0000	224.88
20	0.6716	0.5881	0.0001	207.02
21	0.6615	0.5953	0.0003	186.51
22	0.6639	0.5934	0.0005	188.37
23	0.6482	0.6039	0.0008	188.94
24	0.6497	0.5997	0.0009	3476.82
25	0.6593	0.5982	0.0010	205.19
26	0.6556	0.5973	0.0009	212.85
27	0.6547	0.5994	0.0008	220.71
28	0.6533	0.6021	0.0005	213.49
29	0.6598	0.5958	0.0003	215.56
30	0.6631	0.5951	0.0001	221.37
31	0.6715	0.5894	0.0000	231.45
32	0.6740	0.5889	0.0001	225.12
33	0.6763	0.5853	0.0003	230.78

34	0.6695	0.5906	0.0005	223.10
35	0.6757	0.5864	0.0008	229.22
36	0.6795	0.5785	0.0009	206.8535
37	0.6727	0.5796	0.0010	215.1617
38	0.6752	0.5772	0.0009	205.0652
39	0.6683	0.5773	0.0008	204.1466
40	0.6843	0.5718	0.0005	207.6666
41	0.6856	0.5669	0.0003	209.7018
42	0.6850	0.5602	0.0001	209.2332
43	0.6947	0.5530	0.0000	204.0090
44	0.6887	0.5611	0.0001	207.6596
45	0.6805	0.5640	0.0003	216.3195
46	0.6814	0.5639	0.0005	206.0514
47	0.6814	0.5689	0.0008	221.1731
48	0.6734	0.5724	0.0009	1069.2002
49	0.6890	0.5670	0.0010	1321.0946
50	0.6858	0.5697	0.0009	541.7838
51	0.6877	0.5568	0.0008	274.7334
52	0.6885	0.5626	0.0005	276.1541
53	0.6969	0.5491	0.0003	274.9639
54	0.6969	0.5458	0.0001	275.1854
55	0.7161	0.5351	0.0000	276.0556
56	0.7073	0.5367	0.0001	275.2748
57	0.7021	0.5519	0.0002	275.6932
58	0.7069	0.5509	0.0005	275.9602
59	0.7008	0.5498	0.0007	276.5452
60	0.6959	0.5522	0.0009	277.4040
61	0.7033	0.5494	0.0010	277.4738
62	0.6943	0.5573	0.0009	276.3584
63	0.6978	0.5486	0.0007	3119.4050
64	0.7016	0.5432	0.0005	750.7040
65	0.7114	0.5315	0.0002	1409.0754
66	0.7171	0.5262	0.0001	3098.8918
67	0.7169	0.5275	0.0000	2169.2174
68	0.7242	0.5303	0.0001	3110.8379
69	0.7124	0.5344	0.0003	234.8687
70	0.7073	0.5367	0.0005	552.8289
71	0.7092	0.5420	0.0008	2024.0592
72	0.7069	0.5405	0.0009	1775.6442
73	0.7054	0.5461	0.0010	1951.7870
74	0.7076	0.5404	0.0009	2023.6631
75	0.7143	0.5298	0.0008	1996.6283
76	0.7103	0.5281	0.0005	307.8688

77	0.7222	0.5182	0.0003	227.5595
78	0.7264	0.5188	0.0001	259.4791
79	0.7315	0.5135	0.0000	262.3955
80	0.7329	0.5140	0.0001	259.4302
81	0.7251	0.5196	0.0003	215.6974
82	0.7168	0.5229	0.0005	204.5370
83	0.7209	0.5310	0.0008	194.0692
84	0.7184	0.5308	0.0009	180.8413
85	0.7161	0.5312	0.0010	178.3031
86	0.7126	0.5410	0.0009	176.1423
87	0.7244	0.5253	0.0008	169.4450
88	0.7325	0.5192	0.0005	180.7435
89	0.7269	0.5142	0.0003	183.6866
90	0.7280	0.5157	0.0001	171.4747
91	0.7281	0.5054	0.0000	184.7220
92	0.7331	0.5079	0.0001	188.9478
93	0.7285	0.5184	0.0002	183.6521
94	0.7168	0.5165	0.0005	195.8214
95	0.7296	0.5129	0.0008	189.7627
96	0.7198	0.5226	0.0009	187.5050
97	0.7247	0.5283	0.0010	186.4643
98	0.7244	0.5264	0.0009	196.2164
99	0.7274	0.5235	0.0008	171.2919
100	0.7286	0.5123	0.0005	186.4620
101	0.7398	0.5078	0.0003	191.3543
102	0.7456	0.5000	0.0001	183.6076
103	0.7335	0.5099	0.0000	184.4555
104	0.7437	0.4964	0.0001	223.7835
105	0.7406	0.5006	0.0003	194.1666
106	0.7386	0.5039	0.0005	177.6554
107	0.7328	0.5131	0.0008	184.3795
108	0.7283	0.5111	0.0009	192.3937
109	0.7198	0.5252	0.0010	191.0542
110	0.7241	0.5201	0.0009	177.9224
111	0.7325	0.5155	0.0008	169.1213
112	0.7367	0.5043	0.0005	178.4413
113	0.7385	0.4993	0.0003	179.0867
114	0.7540	0.4986	0.0001	180.8570
115	0.7450	0.4956	0.0000	174.7618
116	0.7425	0.4981	0.0001	188.1958
117	0.7496	0.4994	0.0002	193.5494
118	0.7336	0.5097	0.0005	179.9302
119	0.7357	0.5038	0.0007	176.1915

120	0.7343	0.5097	0.0009	177.2198
121	0.7347	0.5170	0.0010	191.8987
122	0.7249	0.5166	0.0009	187.2312
123	0.7392	0.5065	0.0007	196.1636
124	0.7390	0.5026	0.0005	182.4759
125	0.7465	0.4981	0.0002	177.0069
126	0.7543	0.4846	0.0001	187.8869
127	0.7620	0.4832	0.0000	185.4104
128	0.7532	0.4873	0.0001	185.6866
129	0.7519	0.4862	0.0002	195.8693
130	0.7467	0.5004	0.0005	188.8569
131	0.7400	0.5038	0.0007	198.4198
132	0.7367	0.5038	0.0009	192.2901
133	0.7405	0.5014	0.0010	189.1152
134	0.7397	0.5078	0.0009	191.0535
135	0.7474	0.4969	0.0007	192.3888
136	0.7441	0.4926	0.0005	188.1908
137	0.7592	0.4781	0.0002	175.9198
138	0.7547	0.4810	0.0001	200.7405
139	0.7620	0.4766	0.0000	182.7721
140	0.7648	0.4804	0.0001	196.1903
141	0.7580	0.4829	0.0003	183.8936
142	0.7553	0.4817	0.0005	189.3177
143	0.7541	0.4898	0.0008	180.2261
144	0.7571	0.4840	0.0009	187.8880
145	0.7510	0.4865	0.0010	188.5554
146	0.7594	0.4728	0.0009	193.2598
147	0.7802	0.4487	0.0008	192.4138
148	0.7864	0.4402	0.0005	197.2609
149	0.7950	0.4195	0.0003	185.7089
150	0.8039	0.4125	0.0001	183.2769
151	0.8063	0.4123	0.0000	196.8654
152	0.8058	0.4129	0.0001	186.5226
153	0.8040	0.4138	0.0003	200.5708
154	0.7906	0.4257	0.0005	188.6320
155	0.7733	0.4498	0.0008	196.0203
156	0.7838	0.4431	0.0009	195.0388
157	0.7837	0.4417	0.0010	191.4496
158	0.7962	0.4308	0.0009	196.9720
159	0.7991	0.4239	0.0008	196.7952
160	0.7988	0.4109	0.0005	197.1041
161	0.8094	0.4011	0.0003	192.5399
162	0.8186	0.3935	0.0001	198.0158

163	0.8214	0.3915	0.0000	195.8594
164	0.8183	0.3938	0.0001	191.7052
165	0.8185	0.3993	0.0002	195.0230
166	0.8040	0.4023	0.0005	190.0244
167	0.7941	0.4222	0.0008	195.4310
168	0.7915	0.4267	0.0009	192.2605
169	0.7883	0.4241	0.0010	185.3332
170	0.7958	0.4237	0.0009	172.0494
171	0.8072	0.4153	0.0008	171.3118
172	0.8099	0.4029	0.0005	171.8860
173	0.8195	0.3837	0.0003	167.4542
174	0.8257	0.3851	0.0001	170.7568
175	0.8225	0.3811	0.0000	179.1034
176	0.8191	0.3843	0.0001	175.0586
177	0.8132	0.3957	0.0003	170.0416
178	0.8140	0.4021	0.0005	170.0005
179	0.8076	0.4031	0.0008	167.9900
180	0.7982	0.4223	0.0009	195.0081
181	0.7986	0.4214	0.0010	182.7346
182	0.8089	0.3981	0.0009	185.3227
183	0.8106	0.4051	0.0008	187.4161
184	0.8205	0.3886	0.0005	212.2136
185	0.8274	0.3736	0.0003	191.9770
186	0.8349	0.3718	0.0001	192.4283
187	0.8321	0.3762	0.0000	178.7631
188	0.8281	0.3692	0.0001	179.6160
189	0.8331	0.3708	0.0002	3075.1059
190	0.8268	0.3803	0.0005	180.7224
191	0.8162	0.3959	0.0007	174.6000
192	0.8076	0.4088	0.0009	174.8601
193	0.8125	0.4073	0.0010	181.9408
194	0.8156	0.4004	0.0009	187.1974
195	0.8201	0.3944	0.0007	190.4524
196	0.8233	0.3799	0.0005	176.8258
197	0.8305	0.3699	0.0002	180.9267
198	0.8350	0.3625	0.0001	178.3842
199	0.8368	0.3570	0.0000	183.5044
200	0.8414	0.3624	0.0001	180.9882
Accuracy	80.11%			
balanced Accuracy	86.76%			

Table A.4.: Third training results of EEGNet for Classification

A.7.4. Forth Training

ID	4
Date	02.12.2024, 13:45
Model Definition	n_outputs=n_classes(=2), n_chans=n_channels(=14), n_times=sampling_rate(=128), final_conv_length="auto"
Classifier Definition	model, criterion=CrossEntropyLoss(weight=class_weights), default optimizer = optim.Adam, lr=0.001, batch_size=16, callbacks=callbacks, warm_start = default false, train_split=predefined_split(val_dataset)

Training Epochs

Epoch	Train Bal Acc	Train Loss	Val Acc	Val Loss	LR	dur (s)
1	0.5242	0.7014	0.8571	0.6524	0.0010	149.30
2	0.5852	0.6589	0.4967	0.6449	0.0009	144.57
3	0.6069	0.6449	0.8220	0.6259	0.0008	143.40
4	0.6197	0.6412	0.6010	0.6174	0.0005	143.54
5	0.6095	0.6346	0.7366	0.6115	0.0003	158.64
6	0.6116	0.6297	0.6445	0.6121	0.0001	145.58
7	0.6292	0.6325	0.6367	0.6131	0.0000	143.20
8	0.6187	0.6384	0.6680	0.6131	0.0001	141.63
9	0.6237	0.6329	0.6507	0.6129	0.0003	147.54
10	0.6314	0.6261	0.4241	0.6661	0.0005	155.59
11	0.6253	0.6375	0.8309	0.6134	0.0008	159.72
12	0.6181	0.6335	0.8739	0.7700	0.0009	146.36
13	0.6246	0.6328	0.8415	0.6060	0.0010	152.64
14	0.6249	0.6271	0.6339	0.6027	0.0009	155.42
15	0.6293	0.6205	0.6641	0.6026	0.0008	151.82
16	0.6493	0.6104	0.7321	0.5945	0.0005	154.76
17	0.6538	0.6060	0.7165	0.5870	0.0003	152.79
18	0.6554	0.6030	0.6858	0.5866	0.0001	142.40
19	0.6579	0.6031	0.6797	0.5842	0.0000	148.91
20	0.6564	0.5988	0.6350	0.5905	0.0001	144.17
21	0.6545	0.6037	0.6970	0.5883	0.0003	145.38
22	0.6627	0.6019	0.8683	0.6446	0.0005	147.94
23	0.6450	0.6091	0.8527	0.5901	0.0008	142.05
24	0.6577	0.6016	0.8739	0.8654	0.0009	147.49
25	0.6514	0.6102	0.6735	0.5747	0.0010	138.79
26	0.6569	0.6015	0.4883	0.6145	0.0009	150.09

27	0.6583	0.5993	0.7729	0.5729	0.0008	149.29
28	0.6698	0.5927	0.5871	0.5901	0.0005	149.03
29	0.6802	0.5828	0.5045	0.6107	0.0003	148.75
30	0.6683	0.5851	0.7260	0.5630	0.0001	148.77
31	0.6740	0.5794	0.6903	0.5665	0.0000	144.52
32	0.6716	0.5817	0.6763	0.5656	0.0001	147.33
33	0.6788	0.5742	0.8387	0.5758	0.0003	154.59
34	0.6744	0.5876	0.5089	0.6187	0.0005	149.45
35	0.6583	0.5888	0.6574	0.5782	0.0008	168.13
36	0.6549	0.5968	0.8711	0.6545	0.0009	143.82
37	0.6685	0.5894	0.8025	0.5755	0.0010	147.48
38	0.6603	0.5906	0.5106	0.6175	0.0009	153.92
39	0.6688	0.5894	0.6144	0.5772	0.0008	154.48
40	0.6744	0.5730	0.8778	0.6013	0.0005	143.86
41	0.6803	0.5696	0.7991	0.5590	0.0003	140.23
42	0.6774	0.5673	0.7081	0.5468	0.0001	154.38
43	0.6739	0.5602	0.7065	0.5462	0.0000	154.30
44	0.6877	0.5640	0.7333	0.5448	0.0001	149.85
45	0.6848	0.5685	0.7885	0.5572	0.0003	151.22
46	0.6857	0.5728	0.8739	0.6033	0.0005	150.24
47	0.6815	0.5774	0.8276	0.5593	0.0008	156.87
48	0.6858	0.5712	0.7176	0.5506	0.0009	149.28
49	0.6753	0.5757	0.8728	0.8394	0.0010	153.48
50	0.6886	0.5687	0.8761	0.5798	0.0009	145.91
51	0.6801	0.5747	0.4308	0.6703	0.0008	152.40
52	0.6836	0.5696	0.7779	0.5390	0.0005	156.68
53	0.6921	0.5553	0.8571	0.5434	0.0003	160.17
54	0.6987	0.5541	0.6970	0.5295	0.0001	150.53
55	0.7023	0.5522	0.7143	0.5291	0.0000	150.73
56	0.6999	0.5536	0.7215	0.5289	0.0001	156.61
57	0.6955	0.5522	0.6892	0.5296	0.0002	149.57
58	0.6996	0.5473	0.8677	0.5409	0.0005	161.68
59	0.6867	0.5636	0.6836	0.5718	0.0007	152.26
60	0.6897	0.5667	0.5084	0.5920	0.0009	155.46
61	0.6838	0.5683	0.8510	0.5757	0.0010	153.36
62	0.6952	0.5551	0.8387	0.5439	0.0009	154.85
63	0.6907	0.5638	0.6892	0.5484	0.0007	160.11
64	0.7040	0.5450	0.8744	0.5405	0.0005	152.52
65	0.7051	0.5428	0.7065	0.5175	0.0002	150.44
66	0.7062	0.5378	0.7288	0.5043	0.0001	158.71
67	0.7118	0.5335	0.7199	0.5154	0.0000	213.70
68	0.7206	0.5333	0.7266	0.5155	0.0001	222.31
69	0.7006	0.5380	0.7494	0.5161	0.0003	224.04

70	0.7005	0.5440	0.6607	0.5386	0.0005	224.98
71	0.6897	0.5486	0.6044	0.5553	0.0008	225.31
72	0.7055	0.5564	0.8633	0.6053	0.0009	224.40
73	0.6940	0.5534	0.5926	0.5583	0.0010	223.95
74	0.6974	0.5582	0.6094	0.5475	0.0009	225.02
75	0.7082	0.5386	0.5106	0.5904	0.0008	225.15
76	0.7062	0.5418	0.7221	0.5207	0.0005	211.71
77	0.7144	0.5359	0.8209	0.5169	0.0003	164.30
78	0.7154	0.5282	0.7489	0.5072	0.0001	149.60
79	0.7089	0.5340	0.7478	0.5103	0.0000	151.81
80	0.7100	0.5307	0.7031	0.5117	0.0001	153.77
81	0.7174	0.5250	0.7455	0.5146	0.0003	148.46
82	0.7147	0.5355	0.7757	0.5131	0.0005	157.77
83	0.7078	0.5403	0.5569	0.5716	0.0008	159.55
84	0.7038	0.5380	0.5664	0.5656	0.0009	151.62
85	0.7010	0.5436	0.5050	0.5903	0.0010	150.16
86	0.6987	0.5437	0.8739	0.5551	0.0009	146.93
87	0.7128	0.5419	0.6607	0.5218	0.0008	146.95
88	0.7157	0.5257	0.6769	0.5283	0.0005	152.11
89	0.7206	0.5251	0.7277	0.5093	0.0003	140.38
90	0.7317	0.5211	0.7093	0.5001	0.0001	157.31
91	0.7269	0.5230	0.7193	0.5030	0.0000	150.56
92	0.7273	0.5156	0.7338	0.5041	0.0001	153.16
93	0.7175	0.5291	0.7310	0.5001	0.0002	153.35
94	0.7094	0.5432	0.8158	0.5198	0.0005	154.44
95	0.7059	0.5403	0.8270	0.5128	0.0008	155.41
96	0.7136	0.5444	0.8689	0.5245	0.0009	153.79
97	0.7054	0.5404	0.7243	0.5116	0.0010	149.34
98	0.7056	0.5394	0.7087	0.5057	0.0009	148.66
99	0.7247	0.5249	0.8354	0.5008	0.0008	161.89
100	0.7116	0.5323	0.7344	0.5011	0.0005	157.82
101	0.7264	0.5194	0.6200	0.5254	0.0003	160.70
102	0.7307	0.5156	0.7561	0.4926	0.0001	157.61
103	0.7320	0.5211	0.7422	0.4925	0.0000	155.57
104	0.7309	0.5143	0.7154	0.5048	0.0001	166.10
105	0.7305	0.5165	0.7048	0.5066	0.0003	151.01
106	0.7192	0.5228	0.8789	0.5383	0.0005	149.19
107	0.7274	0.5191	0.6283	0.5259	0.0008	147.60
108	0.7130	0.5356	0.6077	0.5570	0.0009	153.81
109	0.7126	0.5322	0.8198	0.5106	0.0010	150.78
110	0.7235	0.5314	0.4688	0.6509	0.0009	148.03
111	0.7137	0.5297	0.6434	0.5110	0.0008	149.87
112	0.7294	0.5190	0.8594	0.4896	0.0005	149.66

113	0.7248	0.5230	0.7506	0.4972	0.0003	157.14
114	0.7242	0.5150	0.7411	0.4928	0.0001	152.38
115	0.7297	0.5105	0.7679	0.4875	0.0000	157.73
116	0.7309	0.5063	0.7550	0.4862	0.0001	151.45
117	0.7251	0.5163	0.7104	0.4918	0.0002	149.49
118	0.7244	0.5211	0.8700	0.5286	0.0005	163.23
119	0.7260	0.5239	0.8717	0.5254	0.0007	154.14
120	0.7258	0.5220	0.5580	0.5645	0.0009	152.96
121	0.7178	0.5286	0.6730	0.4965	0.0010	151.63
122	0.7177	0.5269	0.8504	0.5031	0.0009	157.88
123	0.7208	0.5165	0.6401	0.5284	0.0007	156.22
124	0.7285	0.5097	0.7098	0.4869	0.0005	155.61
125	0.7343	0.5109	0.7121	0.4865	0.0002	156.19
126	0.7327	0.5025	0.7522	0.4903	0.0001	148.16
127	0.7363	0.5065	0.7500	0.4780	0.0000	158.41
128	0.7290	0.5124	0.7578	0.4807	0.0001	138.55
129	0.7391	0.5039	0.6975	0.4884	0.0002	160.36
130	0.7317	0.5145	0.8270	0.4902	0.0005	155.74
131	0.7230	0.5149	0.6641	0.5096	0.0007	149.81
132	0.7173	0.5265	0.8750	0.5223	0.0009	156.55
133	0.7258	0.5205	0.7026	0.5055	0.0010	160.11
134	0.7179	0.5269	0.8806	0.5391	0.0009	155.71
135	0.7226	0.5229	0.7896	0.4877	0.0007	154.53
136	0.7334	0.5083	0.8242	0.4927	0.0005	156.10
137	0.7371	0.5007	0.7182	0.4807	0.0002	143.15
138	0.7402	0.5046	0.7539	0.4771	0.0001	177.39
139	0.7406	0.4968	0.7478	0.4688	0.0000	158.29
140	0.7390	0.5003	0.7522	0.4742	0.0001	154.87
141	0.7326	0.5057	0.6786	0.4866	0.0003	151.26
142	0.7302	0.5097	0.7734	0.4818	0.0005	155.28
143	0.7321	0.5107	0.6869	0.4992	0.0008	159.34
144	0.7266	0.5203	0.7634	0.4886	0.0009	147.34
145	0.7287	0.5113	0.8756	0.6175	0.0010	152.84
146	0.7288	0.5211	0.7031	0.4904	0.0009	156.59
147	0.7281	0.5127	0.6032	0.5283	0.0008	157.22
148	0.7354	0.5023	0.5815	0.5263	0.0005	154.76
149	0.7429	0.4978	0.7478	0.4747	0.0003	156.85
150	0.7423	0.4997	0.7628	0.4755	0.0001	155.52
151	0.7480	0.4943	0.7433	0.4739	0.0000	154.66
152	0.7382	0.4990	0.7444	0.4762	0.0001	164.94
153	0.7326	0.5037	0.7545	0.4823	0.0003	155.46
154	0.7324	0.5040	0.7573	0.4829	0.0005	149.26
155	0.7409	0.5082	0.6875	0.4874	0.0008	155.21

156	0.7269	0.5134	0.7679	0.4784	0.0009	151.44
157	0.7330	0.5077	0.7433	0.4913	0.0010	155.85
158	0.7321	0.5170	0.8092	0.4865	0.0009	157.71
159	0.7418	0.5047	0.7779	0.4835	0.0008	151.85
160	0.7293	0.5114	0.6970	0.4804	0.0005	147.82
161	0.7370	0.4948	0.7422	0.4649	0.0003	150.17
162	0.7387	0.5055	0.7455	0.4707	0.0001	153.41
163	0.7455	0.4929	0.7366	0.4693	0.0000	150.86
164	0.7361	0.4974	0.7768	0.4686	0.0001	157.24
165	0.7310	0.5015	0.8371	0.4744	0.0002	145.34
166	0.7353	0.5003	0.5938	0.5293	0.0005	150.71
167	0.7312	0.5170	0.8756	0.5814	0.0008	157.28
168	0.7362	0.5034	0.7472	0.4789	0.0009	154.75
169	0.7333	0.5073	0.6708	0.4886	0.0010	154.80
170	0.7384	0.5012	0.7321	0.4810	0.0009	152.39
171	0.7407	0.5062	0.7891	0.4746	0.0008	150.23
172	0.7409	0.4946	0.6607	0.4931	0.0005	156.26
173	0.7501	0.4798	0.7388	0.4568	0.0003	157.05
174	0.7486	0.4938	0.7785	0.4562	0.0001	149.56
175	0.7519	0.4874	0.7517	0.4664	0.0000	163.33
176	0.7522	0.4898	0.7595	0.4627	0.0001	150.08
177	0.7431	0.4951	0.7310	0.4710	0.0003	162.38
178	0.7416	0.4919	0.6680	0.4879	0.0005	154.11
179	0.7449	0.5068	0.6194	0.5123	0.0008	157.10
180	0.7359	0.5022	0.6021	0.5141	0.0009	149.99
181	0.7353	0.5146	0.7009	0.4823	0.0010	157.56
182	0.7269	0.5011	0.6138	0.5205	0.0009	148.19
183	0.7385	0.5044	0.8666	0.4941	0.0008	157.47
184	0.7439	0.5016	0.8694	0.5049	0.0005	153.18
185	0.7493	0.4834	0.7271	0.4722	0.0003	145.72
186	0.7495	0.4805	0.7506	0.4674	0.0001	152.26
187	0.7429	0.4878	0.7712	0.4642	0.0000	153.40
188	0.7568	0.4898	0.7701	0.4722	0.0001	155.95
189	0.7519	0.4892	0.6708	0.4909	0.0002	153.20
190	0.7345	0.4964	0.7321	0.4681	0.0005	148.49
191	0.7381	0.5059	0.7656	0.4846	0.0007	154.28
192	0.7407	0.5059	0.6741	0.4908	0.0009	153.65
193	0.7332	0.5020	0.8683	0.5776	0.0010	144.90
194	0.7422	0.4954	0.7835	0.4763	0.0009	186.21
195	0.7501	0.4964	0.6239	0.5125	0.0007	153.09
196	0.7515	0.4875	0.8811	0.5299	0.0005	148.65
197	0.7455	0.4894	0.7065	0.4756	0.0002	147.56
198	0.7566	0.4825	0.7673	0.4614	0.0001	152.36

199	0.7572	0.4754	0.7467	0.4671	0.0000	156.72
200	0.7620	0.4693	0.7695	0.4591	0.0001	154.99
Accuracy	76.84%					
balanced Accuracy	79.17%					

Table A.5.: Forth training results of EEGNet for Classification

A.7.5. Fifth Training

ID	5					
Date	15.12.2024, 14:15					
Model Definition	n_outputs=n_classes(=2), n_chans=n_channels(=14), n_times=sampling_rate(=128), final_conv_length="auto"					
Classifier Definition	model, criterion=CrossEntropyLoss(weight=class_weights), default optimizer = optim.Adam, lr=0.001, batch_size=16, callbacks=callbacks, warm_start = default false, train_split=predefined_split(val_dataset)					
Training Epochs						
Epoch	Train Bal Acc	Train Loss	Val Acc	Val Loss	LR	dur (s)
1	0.5237	0.7043	0.8644	1.1427	0.0010	191.42
2	0.5766	0.6643	0.6989	0.6410	0.0009	191.08
3	0.6044	0.6484	0.7458	0.6391	0.0008	233.51
4	0.6098	0.6434	0.7955	0.6345	0.0005	267.10
5	0.6176	0.6327	0.6861	0.6216	0.0003	197.03
6	0.6328	0.6316	0.6532	0.6216	0.0001	181.30
7	0.6321	0.6264	0.6543	0.6199	0.0000	172.81
8	0.6279	0.6260	0.6602	0.6190	0.0001	175.06
9	0.6270	0.6294	0.7310	0.6155	0.0003	181.70
10	0.6319	0.6265	0.4916	0.6297	0.0005	179.60
11	0.6319	0.6216	0.8214	0.6559	0.0008	179.43
12	0.6340	0.6219	0.6850	0.6009	0.0009	185.68
13	0.6402	0.6158	0.8616	0.7236	0.0010	173.91
14	0.6411	0.6111	0.5061	0.6176	0.0009	180.55
15	0.6586	0.6018	0.8597	0.6362	0.0008	174.97
16	0.6531	0.6000	0.8089	0.5883	0.0005	178.93
17	0.6670	0.5909	0.6406	0.5786	0.0003	178.46
18	0.6590	0.5933	0.6777	0.5723	0.0001	183.42
19	0.6607	0.5810	0.7009	0.5696	0.0000	177.80

20	0.6701	0.5831	0.6783	0.5754	0.0001	191.28
21	0.6691	0.5866	0.6705	0.5722	0.0003	181.48
22	0.6665	0.5883	0.7533	0.5794	0.0005	194.08
23	0.6661	0.5899	0.8597	0.7365	0.0008	181.17
24	0.6657	0.5920	0.5047	0.5998	0.0009	186.36
25	0.6706	0.5892	0.8479	0.6335	0.0010	237.53
26	0.6587	0.5943	0.8641	0.6224	0.0009	293.21
27	0.6672	0.5802	0.7475	0.5630	0.0008	286.58
28	0.6806	0.5761	0.7302	0.5637	0.0005	285.92
29	0.6795	0.5673	0.6769	0.5511	0.0003	283.23
30	0.6840	0.5680	0.7015	0.5515	0.0001	278.28
31	0.6984	0.5617	0.6953	0.5503	0.0000	296.97
32	0.6824	0.5647	0.7134	0.5501	0.0001	234.36
33	0.6807	0.5688	0.7369	0.5462	0.0003	187.35
34	0.6859	0.5670	0.6180	0.5620	0.0005	254.87
35	0.6788	0.5744	0.7015	0.5539	0.0008	295.59
36	0.6684	0.5771	0.7486	0.5538	0.0009	295.24
37	0.6698	0.5804	0.7160	0.5691	0.0010	295.29
38	0.6876	0.5677	0.7746	0.5603	0.0009	228.07
39	0.6901	0.5678	0.8666	0.5908	0.0008	187.44
40	0.6844	0.5667	0.6655	0.5550	0.0005	183.73
41	0.6873	0.5561	0.7037	0.5391	0.0003	186.30
42	0.6886	0.5564	0.7051	0.5380	0.0001	180.21
43	0.6969	0.5514	0.6959	0.5357	0.0000	184.61
44	0.6921	0.5509	0.7017	0.5387	0.0001	183.70
45	0.6917	0.5512	0.7676	0.5436	0.0003	178.40
46	0.6859	0.5612	0.8382	0.5494	0.0005	178.83
47	0.6869	0.5655	0.7511	0.5396	0.0008	180.78
48	0.6920	0.5638	0.7224	0.5466	0.0009	192.52
49	0.6809	0.5697	0.8390	0.7042	0.0010	187.11
50	0.6900	0.5569	0.8276	0.5976	0.0009	186.96
51	0.7001	0.5550	0.7268	0.5353	0.0008	174.42
52	0.6925	0.5531	0.8092	0.5454	0.0005	245.89
53	0.7024	0.5418	0.6476	0.5326	0.0003	262.14
54	0.7030	0.5444	0.6897	0.5292	0.0001	212.63
55	0.7022	0.5439	0.7073	0.5263	0.0000	310.3
56	0.7154	0.5363	0.7210	0.5256	0.0001	242.22
57	0.7086	0.5460	0.7628	0.5278	0.0002	310.20
58	0.7083	0.5491	0.7684	0.5369	0.0005	389.18
59	0.6970	0.5517	0.6694	0.5319	0.0007	385.86
60	0.6874	0.5666	0.7469	0.5362	0.0009	406.48
61	0.7029	0.5493	0.6406	0.5668	0.0010	422.69
62	0.6999	0.5533	0.6119	0.5451	0.0009	429.58

63	0.6976	0.5495	0.8544	0.5606	0.0007	426.33
64	0.7035	0.5396	0.8075	0.5395	0.0005	421.65
65	0.7106	0.5411	0.7227	0.5249	0.0002	428.62
66	0.7152	0.5301	0.6939	0.5259	0.0001	422.31
67	0.7095	0.5363	0.7171	0.5178	0.0000	431.28
68	0.7180	0.5316	0.7238	0.5184	0.0001	430.28
69	0.7143	0.5336	0.7215	0.5181	0.0003	302.34
70	0.7053	0.5437	0.5935	0.5432	0.0005	290.10
71	0.7133	0.5458	0.8580	0.6680	0.0008	290.47
72	0.7056	0.5406	0.5935	0.5418	0.0009	291.81
73	0.7094	0.5439	0.8605	0.6152	0.0010	291.51
74	0.6986	0.5460	0.5929	0.5516	0.0009	294.57
75	0.7077	0.5384	0.5047	0.5934	0.0008	294.17
76	0.7066	0.5359	0.6964	0.5185	0.0005	292.52
77	0.7158	0.5277	0.6172	0.5246	0.0003	291.32
78	0.7261	0.5185	0.7277	0.5012	0.0001	292.70
79	0.7191	0.5271	0.7062	0.5095	0.0000	293.79
80	0.7131	0.5285	0.7146	0.5072	0.0001	293.21
81	0.7216	0.5232	0.8136	0.5095	0.0003	287.23
82	0.7146	0.5332	0.7006	0.5096	0.0005	230.11
83	0.7023	0.5393	0.8717	0.5497	0.0008	296.79
84	0.7037	0.5400	0.8583	0.5871	0.0009	295.85
85	0.7052	0.5372	0.8348	0.5319	0.0010	299.57
86	0.7190	0.5318	0.6161	0.5302	0.0009	295.01
87	0.7166	0.5341	0.7062	0.5122	0.0008	294.59
88	0.7115	0.5317	0.7999	0.5048	0.0005	297.11
89	0.7243	0.5218	0.7570	0.4977	0.0003	294.62
90	0.7255	0.5093	0.7645	0.4950	0.0001	292.90
91	0.7236	0.5108	0.7419	0.4979	0.0000	292.72
92	0.7205	0.5167	0.7360	0.5000	0.0001	291.91
93	0.7266	0.5162	0.7282	0.5028	0.0002	291.07
94	0.7137	0.5227	0.8189	0.5055	0.0005	289.49
95	0.7153	0.5322	0.5854	0.5458	0.0008	223.63
96	0.7113	0.5353	0.8680	0.5871	0.0009	287.77
97	0.7146	0.5295	0.5633	0.5479	0.0010	288.90
98	0.7198	0.5270	0.7026	0.5133	0.0009	288.40
99	0.7230	0.5149	0.8016	0.5144	0.0008	292.87
100	0.7238	0.5161	0.8465	0.5268	0.0005	294.22
101	0.7424	0.5088	0.7360	0.4918	0.0003	293.79
102	0.7271	0.5171	0.7709	0.4949	0.0001	293.58
103	0.7275	0.5159	0.7592	0.4951	0.0000	292.52
104	0.7291	0.5118	0.7637	0.4911	0.0001	294.48
105	0.7299	0.5124	0.8474	0.5188	0.0003	293.17

106	0.7252	0.5185	0.8544	0.5214	0.0005	291.91
107	0.7218	0.5238	0.8633	0.5215	0.0008	297.91
108	0.7137	0.5267	0.8708	0.5752	0.0009	274.72
109	0.7136	0.5307	0.8440	0.5085	0.0010	243.74
110	0.7229	0.5160	0.6590	0.5187	0.0009	296.57
111	0.7295	0.5117	0.8650	0.5193	0.0008	292.29
112	0.7175	0.5237	0.6747	0.5031	0.0005	294.47
113	0.7294	0.5107	0.8549	0.5138	0.0003	294.57
114	0.7294	0.5125	0.7617	0.4903	0.0001	293.58
115	0.7373	0.5044	0.7573	0.4895	0.0000	295.90
116	0.7446	0.5017	0.7520	0.4868	0.0001	292.62
117	0.7288	0.5052	0.7812	0.4947	0.0002	292.15
118	0.7314	0.5191	0.7079	0.4957	0.0005	290.18
119	0.7320	0.5158	0.7148	0.5036	0.0007	290.61
120	0.7175	0.5220	0.8619	0.6646	0.0009	289.42
121	0.7211	0.5288	0.8619	0.5521	0.0010	288.31
122	0.7232	0.5185	0.8125	0.5229	0.0009	206.10
123	0.7300	0.5157	0.7977	0.4960	0.0007	286.79
124	0.7223	0.5176	0.7190	0.5120	0.0005	295.07
125	0.7282	0.5086	0.7771	0.4966	0.0002	293.42
126	0.7364	0.5045	0.7818	0.4864	0.0001	293.61
127	0.7272	0.5087	0.7547	0.4905	0.0000	292.09
128	0.7370	0.5001	0.7556	0.4819	0.0001	290.57
129	0.7406	0.5013	0.7059	0.4805	0.0002	292.18
130	0.7314	0.5033	0.8309	0.4927	0.0005	291.34
131	0.7259	0.5165	0.6306	0.5125	0.0007	248.04
132	0.7269	0.5146	0.6030	0.5242	0.0009	191.59
133	0.7264	0.5201	0.7609	0.5044	0.0010	200.16
134	0.7282	0.5207	0.6549	0.5200	0.0009	197.72
135	0.7253	0.5150	0.7204	0.4905	0.0007	186.54
136	0.7314	0.5099	0.6987	0.4909	0.0005	189.69
137	0.7290	0.5060	0.7740	0.4883	0.0002	216.54
138	0.7477	0.4927	0.7924	0.4829	0.0001	183.88
139	0.7462	0.4956	0.7988	0.4809	0.0000	179.63
140	0.7358	0.4963	0.7924	0.4831	0.0001	183.34
141	0.7439	0.4982	0.8460	0.4956	0.0003	193.57
142	0.7362	0.5004	0.8493	0.5045	0.0005	195.01
143	0.7268	0.5136	0.8591	0.5091	0.0008	187.29
144	0.7305	0.5082	0.7899	0.4925	0.0009	192.88
145	0.7259	0.5128	0.8669	0.8603	0.0010	184.72
146	0.7295	0.5157	0.8739	0.5736	0.0009	188.54
147	0.7327	0.5079	0.6515	0.5030	0.0008	173.93
148	0.7364	0.5032	0.7603	0.4815	0.0005	185.03

149	0.7487	0.4964	0.8527	0.4877	0.0003	177.77
150	0.7417	0.4908	0.8033	0.4770	0.0001	175.93
151	0.7444	0.4857	0.8111	0.4699	0.0000	170.46
152	0.7401	0.4932	0.8092	0.4805	0.0001	172.81
153	0.7462	0.4895	0.7229	0.4781	0.0003	179.89
154	0.7352	0.4992	0.6766	0.4982	0.0005	180.3
155	0.7315	0.5020	0.6133	0.5123	0.0008	175.74
156	0.7374	0.5039	0.7397	0.4939	0.0009	175.92
157	0.7330	0.5137	0.6323	0.5006	0.0010	173.50
158	0.7265	0.5063	0.7891	0.5308	0.0009	175.08
159	0.7413	0.4979	0.7868	0.4851	0.0008	180.42
160	0.7370	0.4999	0.7143	0.5098	0.0005	170.33
161	0.7457	0.4889	0.8292	0.4762	0.0003	176.77
162	0.7493	0.4856	0.7907	0.4647	0.0001	175.51
163	0.7503	0.4814	0.7919	0.4660	0.0000	184.27
164	0.7537	0.4814	0.7849	0.4658	0.0001	185.27
165	0.7388	0.4915	0.7087	0.4779	0.0002	183.54
166	0.7481	0.4877	0.8669	0.4952	0.0005	178.74
167	0.7371	0.5005	0.8683	0.4992	0.0008	177.72
168	0.7394	0.4983	0.6515	0.4965	0.0009	182.25
169	0.7370	0.5026	0.8739	0.5343	0.0010	175.87
170	0.7374	0.5031	0.8298	0.4905	0.0009	181.50
171	0.7482	0.4905	0.8650	0.5170	0.0008	171.29
172	0.7369	0.4936	0.7347	0.4743	0.0005	179.11
173	0.7487	0.4814	0.7857	0.4620	0.0003	177.08
174	0.7581	0.4756	0.7916	0.4567	0.0001	183.01
175	0.7572	0.4749	0.7997	0.4559	0.0000	183.41
176	0.7512	0.4793	0.8069	0.4573	0.0001	188.63
177	0.7505	0.4837	0.8036	0.4602	0.0003	194.51
178	0.7507	0.4867	0.8189	0.4756	0.0005	199.43
179	0.7357	0.4957	0.8789	0.5826	0.0008	262.04
180	0.7409	0.4969	0.7405	0.4887	0.0009	208.22
181	0.7436	0.5038	0.8669	0.6896	0.0010	252.75
182	0.7454	0.4914	0.8689	0.5470	0.0009	226.18
183	0.7385	0.4968	0.8563	0.4833	0.0008	201.26
184	0.7509	0.4849	0.7059	0.4627	0.0005	229.42
185	0.7530	0.4772	0.7866	0.4509	0.0003	189.22
186	0.7621	0.4688	0.7988	0.4557	0.0001	178.88
187	0.7595	0.4717	0.8064	0.4526	0.0000	189.08
188	0.7535	0.4782	0.7843	0.4495	0.0001	235.50
189	0.7523	0.4792	0.7210	0.4591	0.0002	179.83
190	0.7550	0.4808	0.7467	0.4623	0.0005	190.29
191	0.7412	0.4868	0.8733	0.4946	0.0007	183.15

192	0.7498	0.4903	0.8719	0.6489	0.0009	218.11
193	0.7405	0.4957	0.6214	0.5104	0.0010	294.65
194	0.7427	0.4907	0.5109	0.5869	0.0009	293.48
195	0.7474	0.4934	0.8340	0.4935	0.0007	294.11
196	0.7550	0.4838	0.8089	0.4600	0.0005	296.95
197	0.7640	0.4694	0.7734	0.4478	0.0002	294.49
198	0.7618	0.4611	0.8094	0.4469	0.0001	294.82
199	0.7718	0.4622	0.8064	0.4455	0.0000	291.90
200	0.7581	0.4686	0.8061	0.4467	0.0001	292.11
Accuracy	82.14%					
balanced Accuracy	82.01%					

Table A.6.: Fifth training results of EEGNet for Classification

A.8. Training EEGNet Regression

A.8.1. First Training

Table A.7.: Log 1st Training EEGNet Regression

ID	1							
Date	02.12.24							
Model Definition	EEGNetv4(n_outputs=n_classes, n_chans=n_channels, n_times=sampling_rate, final_conv_length=auto)							
Classifier Definition	criterion=torch.nn.MSELoss optimizer=torch.optim.AdamW, optimizer__lr=0.001, train_split=predefined_split(val_dataset), batch_size=16, callbacks=callbacks							
Training Epochs								
Epoch	CoD	MAE	MSE	RMSE	tr_loss	val_loss	lr	dur
no metrics tracked during training								
Mean Absolute Error	0.88							
Mean Squared Error	1.88							
Root Mean Squared Error	1.37							
Coefficient of Determination	0.16							

Table A.7.: First training results of EEGNet for Regression

A.8.2. Second Training

ID	2							
Date	12.12.24							
Model Definition	EEGNetv4(n_outputs=n_classes, n_chans=n_channels, n_times=sampling_rate, final_conv_length=auto)							
Classifier Definition	criterion=torch.nn.MSELoss, optimizer=torch.optim.AdamW, optimizer__lr=0.001, train_split=predefined_split(val_dataset), batch_size=16, callbacks=callbacks							
Training Epochs								
Epoch	CoD	MAE	MSE	RMSE	tr_loss	val_loss	lr	dur
1	-0.3065	1.2267	2.9075,	1.7051	2.9075	2.5152	0.0010	180.4731
2	-0.1133	1.1266	2.4774,	1.5740	2.4774	2.3706	0.0009	181.5930
3	-0.0760	1.1072	2.3944,	1.5474	2.3944	2.3598	0.0008	175.9523
4	-0.0634	1.0929	2.3665,	1.5384	2.3665	2.3436	0.0005	179.5408
5	-0.0484	1.0852	2.3331,	1.5275	2.3331	2.3416	0.0003	190.3486
6	-0.0506	1.0882	2.3379,	1.5290	2.3379	2.3444	0.0001	169.1318
7	-0.0401	1.0758	2.3147,	1.5214	2.3147	2.3479	0.0000	177.2515
8	-0.0354	1.0778	2.3042,	1.5180	2.3042	2.3496	0.0001	175.8583
9	-0.0446	1.0832	2.3246,	1.5247	2.3246	2.3585	0.0003	178.0349
10	-0.0406	1.0813	2.3158,	1.5218	2.3158	2.5026	0.0005	185.1414
11	-0.0534	1.0823	2.3442,	1.5311	2.3442	2.5039	0.0008	172.6546
12	-0.0451	1.0868	2.3257,	1.5250	2.3257	2.4232	0.0009	173.3346
13	-0.0397	1.0761	2.3138,	1.5211	2.3138	2.4456	0.0010	190.1833
14	-0.0291	1.0711	2.2902,	1.5133	2.2902	2.3243	0.0009	192.5543
15	-0.0221	1.0665	2.2746,	1.5082	2.2746	2.3140	0.0008	188.4589
16	-0.0150	1.0593	2.2588,	1.5029	2.2588	2.3339	0.0005	173.5264
17	-0.0162	1.0637	2.2614,	1.5038	2.2614	2.4228	0.0003	196.8364
18	-0.0157	1.0669	2.2603,	1.5034	2.2603	2.3573	0.0001	193.4037
19	-0.0036	1.0569	2.2334,	1.4945	2.2334	2.3304	0.0000	3014.878
20	-0.0076	1.0590	2.2422,	1.4974	2.2422	2.3330	0.0001	565.9108
21	-0.0099	1.0634	2.2475,	1.4992	2.2475	2.3320	0.0003	172.5316
22	-0.0106	1.0641	2.2490,	1.4997	2.2490	2.3650	0.0005	185.7479
23	-0.0149	1.0622	2.2585,	1.5028	2.2585	2.3278	0.0008	183.0562
24	-0.0201	1.0701	2.2701,	1.5067	2.2701	2.3290	0.0009	184.4872
25	-0.0105	1.0630	2.2488,	1.4996	2.2488	2.3226	0.0010	185.7131
26	-0.0140	1.0649	2.2565,	1.5022	2.2565	2.3951	0.0009	185.1379
27	0.0103	1.0468	2.2024,	1.4840	2.2024	2.4431	0.0008	188.6029
28	-0.0023	1.0539	2.2306,	1.4935	2.2306	2.4282	0.0005	183.5427

29	0.0049	1.0532	2.2145,	1.4881	2.2145	2.3516	0.0003	181.3973
30	0.0052	1.0546	2.2137,	1.4879	2.2137	2.3361	0.0001	195.3919
31	0.0166	1.0521	2.1884,	1.4793	2.1884	2.3383	0.0000	181.3748
32	0.0101	1.0476	2.2028,	1.4842	2.2028	2.3437	0.0001	186.8553
33	0.0020	1.0529	2.2209,	1.4903	2.2209	2.3253	0.0003	169.3164
34	0.0009	1.0544	2.2233,	1.4911	2.2233	2.5503	0.0005	185.5510
35	0.0090	1.0516	2.2054,	1.4851	2.2054	2.3253	0.0008	188.6878
36	0.0073	1.0511	2.2091,	1.4863	2.2091	2.3607	0.0009	182.7593
37	0.0056	1.0518	2.2130,	1.4876	2.2130	2.5790	0.0010	191.3659
38	0.0147	1.0417	2.1926,	1.4807	2.1926	2.3677	0.0009	183.3520
39	0.0165	1.0445	2.1887,	1.4794	2.1887	2.3398	0.0008	181.8616
40	0.0167	1.0417	2.1882,	1.4793	2.1882	2.3483	0.0005	189.8190
41	0.0172	1.0388	2.1870,	1.4789	2.1870	2.4091	0.0003	190.9337
42	0.0268	1.0429	2.1657,	1.4716	2.1657	2.3409	0.0001	188.8018
43	0.0258	1.0341	2.1680,	1.4724	2.1680	2.3435	0.0000	191.5886
44	0.0310	1.0365	2.1563,	1.4684	2.1563	2.3397	0.0001	182.5502
45	0.0258	1.0415	2.1680,	1.4724	2.1680	2.3933	0.0003	185.9353
46	0.0290	1.0332	2.1609,	1.4700	2.1609	2.3023	0.0005	191.4714
47	0.0197	1.0425	2.1815,	1.4770	2.1815	2.5123	0.0008	191.9902
48	0.0210	1.0409	2.1786,	1.4760	2.1786	2.4278	0.0009	215.8513
49	0.0361	1.0309	2.1450,	1.4646	2.1450	2.7001	0.0010	281.0018
50	0.0202	1.0416	2.1804,	1.4766	2.1804	2.7441	0.0009	278.3598
51	0.0272	1.0358	2.1648,	1.4713	2.1648	2.3417	0.0008	177.5707
52	0.0460	1.0280	2.1230,	1.4571	2.1230	2.5465	0.0005	186.0012
53	0.0428	1.0314	2.1301,	1.4595	2.1301	2.3769	0.0003	250.4033
54	0.0476	1.0248	2.1194,	1.4558	2.1194	2.3662	0.0001	282.1488
55	0.0506	1.0267	2.1127,	1.4535	2.1127	2.3743	0.0000	180.3407
56	0.0526	1.0249	2.1083,	1.4520	2.1083	2.3919	0.0001	183.3175
57	0.0439	1.0255	2.1278,	1.4587	2.1278	2.4200	0.0002	184.9455
58	0.0453	1.0298	2.1245,	1.4576	2.1245	2.4127	0.0005	187.3616
59	0.0427	1.0244	2.1303,	1.4595	2.1303	2.7484	0.0007	191.1046
60	0.0392	1.0318	2.1381,	1.4622	2.1381	2.3492	0.0009	198.6264
61	0.0348	1.0327	2.1480,	1.4656	2.1480	2.5480	0.0010	177.8561
62	0.0432	1.0236	2.1292,	1.4592	2.1292	2.5142	0.0009	188.3796
63	0.0507	1.0270	2.1125,	1.4534	2.1125	2.3899	0.0007	190.8353
64	0.0606	1.0137	2.0905,	1.4458	2.0905	2.8167	0.0005	189.6917
65	0.0602	1.0171	2.0914,	1.4462	2.0914	2.3693	0.0002	192.0965
66	0.0729	1.0146	2.0632,	1.4364	2.0632	2.3660	0.0001	185.6044
67	0.0810	0.9953	2.0452,	1.4301	2.0452	2.3801	0.0000	179.2987
68	0.0766	1.0077	2.0550,	1.4335	2.0550	2.3861	0.0001	187.1959
69	0.0741	1.0118	2.0604,	1.4354	2.0604	2.3948	0.0003	189.7286
70	0.0608	1.0196	2.0900,	1.4457	2.0900	2.4293	0.0005	184.1997
71	0.0561	1.0211	2.1005,	1.4493	2.1005	2.5953	0.0008	181.2321

72	0.0614	1.0191	2.0887,	1.4452	2.0887	2.3852	0.0009	196.0336
73	0.0655	1.0149	2.0795,	1.4421	2.0795	2.3585	0.0010	191.5259
74	0.0720	1.0095	2.0650,	1.4370	2.0650	2.4156	0.0009	184.8705
75	0.0701	1.0107	2.0694,	1.4385	2.0694	2.3986	0.0008	191.0805
76	0.0807	1.0080	2.0458,	1.4303	2.0458	2.4186	0.0005	180.2187
77	0.0861	1.0035	2.0337,	1.4261	2.0337	2.4134	0.0003	187.8186
78	0.0866	1.0016	2.0327,	1.4257	2.0327	2.3951	0.0001	188.3614
79	0.0830	1.0133	2.0408,	1.4286	2.0408	2.4142	0.0000	176.3062
80	0.1010	1.0011	2.0006,	1.4144	2.0006	2.4083	0.0001	180.6168
81	0.0923	1.0010	2.0201,	1.4213	2.0201	2.4114	0.0003	183.4519
82	0.0974	0.9948	2.0087,	1.4173	2.0087	2.9806	0.0005	185.3650
83	0.0873	1.0058	2.0311,	1.4252	2.0311	2.8870	0.0008	189.2629
84	0.0751	1.0044	2.0582,	1.4346	2.0582	2.4945	0.0009	181.1211
85	0.0763	1.0103	2.0557,	1.4338	2.0557	2.4853	0.0010	188.9127
86	0.0882	1.0023	2.0291,	1.4245	2.0291	2.8389	0.0009	186.8603
87	0.0921	0.9984	2.0203,	1.4214	2.0203	2.4468	0.0008	187.9237
88	0.0948	0.9976	2.0145,	1.4193	2.0145	2.4383	0.0005	183.6183
89	0.1045	0.9891	1.9928,	1.4117	1.9928	2.4880	0.0003	181.9618
90	0.1101	0.9880	1.9803,	1.4072	1.9803	2.4190	0.0001	192.9191
91	0.1048	1.0006	1.9922,	1.4115	1.9922	2.4735	0.0000	191.7121
92	0.1067	0.9920	1.9879,	1.4099	1.9879	2.4560	0.0001	186.7844
93	0.1032	0.9923	1.9957,	1.4127	1.9957	2.4605	0.0002	182.5388
94	0.1067	0.9919	1.9880,	1.4100	1.9880	2.4825	0.0005	176.2206
95	0.0863	1.0036	2.0333,	1.4259	2.0333	2.9536	0.0008	188.4377
96	0.0871	1.0006	2.0316,	1.4253	2.0316	2.5300	0.0009	187.8739
97	0.0931	0.9992	2.0183,	1.4207	2.0183	2.4791	0.0010	186.4815
98	0.0936	0.9987	2.0171,	1.4203	2.0171	2.5290	0.0009	185.3493
99	0.0903	1.0007	2.0245,	1.4229	2.0245	2.4918	0.0008	193.3528
100	0.1009	0.9902	2.0009,	1.4145	2.0009	2.4730	0.0005	185.0081
101	0.1121	0.9877	1.9760,	1.4057	1.9760	2.4490	0.0003	183.5515
102	0.1216	0.9817	1.9548,	1.3981	1.9548	2.4932	0.0001	185.6768
103	0.1185	0.9885	1.9617,	1.4006	1.9617	2.4776	0.0000	184.9208
104	0.1256	0.9826	1.9459,	1.3950	1.9459	2.4728	0.0001	185.0173
105	0.1146	0.9830	1.9703,	1.4037	1.9703	2.4455	0.0003	179.0798
106	0.1134	0.9871	1.9729,	1.4046	1.9729	2.5392	0.0005	186.5005
107	0.1002	0.9920	2.0024,	1.4151	2.0024	2.6689	0.0008	189.7882
108	0.1040	0.9934	1.9939,	1.4120	1.9939	3.0638	0.0009	191.3723
109	0.1078	0.9881	1.9854,	1.4090	1.9854	2.8750	0.0010	184.0308
110	0.1079	0.9915	1.9853,	1.4090	1.9853	2.9160	0.0009	176.3959
111	0.1085	0.9863	1.9839,	1.4085	1.9839	2.6784	0.0008	190.6130
112	0.1232	0.9829	1.9512,	1.3968	1.9512	2.5593	0.0005	193.2684
113	0.1296	0.9733	1.9370,	1.3918	1.9370	2.4792	0.0003	185.4465
114	0.1347	0.9722	1.9256,	1.3877	1.9256	2.5074	0.0001	186.3062

115	0.1350	0.9760	1.9249,	1.3874	1.9249	2.5423	0.0000	183.1854
116	0.1365	0.9710	1.9216,	1.3862	1.9216	2.5500	0.0001	185.4017
117	0.1339	0.9727	1.9275,	1.3883	1.9275	2.5534	0.0002	181.5184
118	0.1309	0.9802	1.9340,	1.3907	1.9340	2.5159	0.0005	184.6062
119	0.1087	0.9917	1.9835,	1.4084	1.9835	2.6486	0.0007	175.1568
120	0.1167	0.9829	1.9657,	1.4021	1.9657	2.6620	0.0009	176.1714
121	0.1146	0.9834	1.9703,	1.4037	1.9703	2.9144	0.0010	191.0285
122	0.1216	0.9820	1.9549,	1.3982	1.9549	2.4230	0.0009	184.4030
123	0.1353	0.9740	1.9243,	1.3872	1.9243	2.6755	0.0007	220.8276
124	0.1355	0.9714	1.9239,	1.3870	1.9239	2.5283	0.0005	183.8989
125	0.1561	0.9656	1.8779,	1.3704	1.8779	2.5463	0.0002	180.3817
126	0.1631	0.9561	1.8624,	1.3647	1.8624	2.6685	0.0001	191.1005
127	0.1711	0.9572	1.8446,	1.3582	1.8446	2.6572	0.0000	183.0756
128	0.1695	0.9606	1.8481,	1.3594	1.8481	2.6250	0.0001	189.6790
129	0.1645	0.9591	1.8593,	1.3636	1.8593	2.6357	0.0002	187.1670
130	0.1488	0.9680	1.8943,	1.3763	1.8943	2.9202	0.0005	188.8528
131	0.1471	0.9715	1.8981,	1.3777	1.8981	2.5512	0.0007	186.6861
132	0.1522	0.9683	1.8868,	1.3736	1.8868	3.4029	0.0009	200.5303
133	0.1451	0.9672	1.9024,	1.3793	1.9024	9.9667	0.0010	188.4549
134	0.1334	0.9763	1.9285,	1.3887	1.9285	2.7689	0.0009	180.1840
135	0.1570	0.9637	1.8761,	1.3697	1.8761	2.4341	0.0007	186.4091
136	0.1717	0.9540	1.8433,	1.3577	1.8433	2.6370	0.0005	186.8676
137	0.1868	0.9464	1.8096,	1.3452	1.8096	2.7825	0.0002	222.7636
138	0.1849	0.9481	1.8139,	1.3468	1.8139	2.6870	0.0001	210.5311
139	0.1869	0.9382	1.8094,	1.3451	1.8094	2.6829	0.0000	205.5185
140	0.2067	0.9378	1.7654,	1.3287	1.7654	2.7080	0.0001	186.7164
141	0.1857	0.9509	1.8121,	1.3461	1.8121	3.0649	0.0003	184.1302
142	0.1725	0.9558	1.8415,	1.3570	1.8415	2.6085	0.0005	178.1886
143	0.1597	0.9651	1.8700,	1.3675	1.8700	2.5899	0.0008	185.7325
144	0.1491	0.9607	1.8936,	1.3761	1.8936	3.0805	0.0009	184.2642
145	0.1650	0.9594	1.8583,	1.3632	1.8583	3.1691	0.0010	176.3563
146	0.1514	0.9621	1.8885,	1.3742	1.8885	2.5470	0.0009	187.4277
147	0.1781	0.9519	1.8290,	1.3524	1.8290	2.6772	0.0008	184.6100
148	0.1802	0.9524	1.8244,	1.3507	1.8244	2.6388	0.0005	185.5853
149	0.1931	0.9396	1.7956,	1.3400	1.7956	3.0329	0.0003	179.5117
150	0.2076	0.9345	1.7633,	1.3279	1.7633	2.7916	0.0001	189.9267
151	0.2140	0.9323	1.7490,	1.3225	1.7490	2.7602	0.0000	188.2508
152	0.1968	0.9383	1.7874,	1.3370	1.7874	2.7369	0.0001	185.9297
153	0.1903	0.9453	1.8019,	1.3423	1.8019	3.0403	0.0003	187.5169
154	0.1882	0.9429	1.8065,	1.3441	1.8065	2.5716	0.0005	195.0633
155	0.1798	0.9501	1.8254,	1.3511	1.8254	3.0162	0.0008	188.6988
156	0.1730	0.9525	1.8403,	1.3566	1.8403	6.3245	0.0009	182.5203
157	0.1685	0.9540	1.8504,	1.3603	1.8504	2.6982	0.0010	191.0422

158	0.1726	0.9565	1.8414,	1.3570	1.8414	2.7174	0.0009	188.4224
159	0.1854	0.9442	1.8128,	1.3464	1.8128	3.5219	0.0008	183.7302
160	0.1960	0.9413	1.7892,	1.3376	1.7892	2.6797	0.0005	185.5948
161	0.2183	0.9241	1.7395,	1.3189	1.7395	2.6937	0.0003	186.3357
162	0.2145	0.9263	1.7481,	1.3222	1.7481	2.7624	0.0001	188.7681
163	0.2160	0.9250	1.7448,	1.3209	1.7448	2.6833	0.0000	188.7614
164	0.2111	0.9318	1.7556,	1.3250	1.7556	2.6382	0.0001	186.7067
165	0.2134	0.9294	1.7505,	1.3230	1.7505	2.9107	0.0002	189.9326
166	0.1958	0.9390	1.7896,	1.3377	1.7896	3.2125	0.0005	176.2777
167	0.1987	0.9369	1.7833,	1.3354	1.7833	2.5822	0.0008	180.0195
168	0.1869	0.9401	1.8096,	1.3452	1.8096	2.7573	0.0009	177.9334
169	0.1811	0.9461	1.8224,	1.3500	1.8224	3.0616	0.0010	182.3084
170	0.1948	0.9371	1.7919,	1.3386	1.7919	3.2843	0.0009	202.3247
171	0.1973	0.9345	1.7864,	1.3366	1.7864	3.3682	0.0008	179.6510
172	0.2119	0.9267	1.7538,	1.3243	1.7538	2.7012	0.0005	186.0048
173	0.2195	0.9227	1.7369,	1.3179	1.7369	2.6766	0.0003	191.2262
174	0.2299	0.9102	1.7139,	1.3091	1.7139	2.8524	0.0001	188.6558
175	0.2315	0.9239	1.7103,	1.3078	1.7103	2.7642	0.0000	182.5781
176	0.2251	0.9196	1.7245,	1.3132	1.7245	2.6515	0.0001	184.9428
177	0.2279	0.9180	1.7181,	1.3108	1.7181	2.6587	0.0003	190.1312
178	0.2234	0.9216	1.7282,	1.3146	1.7282	3.0326	0.0005	183.4043
179	0.1946	0.9363	1.7923,	1.3388	1.7923	2.7696	0.0008	194.1768
180	0.1900	0.9391	1.8024,	1.3426	1.8024	2.6509	0.0009	182.3768
181	0.1919	0.9349	1.7982,	1.3410	1.7982	2.5708	0.0010	181.9923
182	0.2036	0.9313	1.7723,	1.3313	1.7723	2.6296	0.0009	179.6951
183	0.2092	0.9238	1.7597,	1.3265	1.7597	3.6637	0.0008	190.7421
184	0.2282	0.9096	1.7176,	1.3106	1.7176	3.5212	0.0005	188.9844
185	0.2475	0.9054	1.6745,	1.2940	1.6745	2.9046	0.0003	195.7215
186	0.2491	0.9038	1.6710,	1.2927	1.6710	2.7340	0.0001	185.1401
187	0.2412	0.9144	1.6887,	1.2995	1.6887	2.8404	0.0000	184.0991
188	0.2571	0.8967	1.6533,	1.2858	1.6533	2.8975	0.0001	178.6797
189	0.2431	0.9095	1.6845,	1.2979	1.6845	2.6612	0.0002	191.6454
190	0.2320	0.9128	1.7092,	1.3074	1.7092	2.8037	0.0005	192.1172
191	0.2284	0.9189	1.7171,	1.3104	1.7171	3.0459	0.0007	180.8179
192	0.2085	0.9224	1.7613,	1.3272	1.7613	4.1145	0.0009	187.4894
193	0.2103	0.9224	1.7573,	1.3256	1.7573	2.4602	0.0010	181.3175
194	0.1969	0.9300	1.7871,	1.3368	1.7871	3.6912	0.0009	186.9780
195	0.2302	0.9158	1.7131,	1.3089	1.7131	2.6905	0.0007	187.3632
196	0.2357	0.9113	1.7009,	1.3042	1.7009	2.6880	0.0005	188.3884
197	0.2759	0.8865	1.6113,	1.2694	1.6113	3.0895	0.0002	193.0199
198	0.2631	0.8979	1.6399,	1.2806	1.6399	2.8201	0.0001	188.0415
199	0.2595	0.9001	1.6479,	1.2837	1.6479	2.7565	0.0000	184.9857
200	0.2672	0.8919	1.6308,	1.2770	1.6308	2.8833	0.0001	190.2855

Mean Absolute Error	0.83
Mean Squared Error	1.41
Root Mean Squared Error	1.19
Coefficient of Determination	0.35

Table A.8.: Second training results of EEGNet for Regression

A.8.3. Third Training

ID	3
Date	13.12.24
Model Definition	EEGNetv4(n_outputs=n_classes, n_chans=n_channels, n_times=sampling_rate, final_conv_length=auto)
Classifier Definition	criterion=torch.nn.MSELoss, optimizer=torch.optim.AdamW, optimizer__lr=0.01, train_split=predefined_split(val_dataset), batch_size=16, callbacks=callbacks

Training Epochs

Epoch	CoD	MAE	MSE	RMSE	tr_loss	val_loss	lr	dur
1	-0.1947	1.1439	2.6383	1.6243	2.6383	2.2295	0.0100	186.5581
2	-0.0847	1.0861	2.3956	1.5478	2.3956	2.2705	0.0093	178.7862
3	-0.0433	1.0657	2.3040	1.5179	2.3040	2.2129	0.0075	188.4882
4	-0.0116	1.0497	2.2341	1.4947	2.2341	2.5856	0.0050	202.8414
5	0.0167	1.0303	2.1715	1.4736	2.1715	2.0854	0.0025	189.9924
6	0.0315	1.0231	2.1388	1.4625	2.1388	2.1099	0.0007	176.9901
7	0.0348	1.0076	2.1316	1.4600	2.1316	2.0833	0.0000	176.1727
8	0.0379	1.0155	2.1248	1.4577	2.1248	2.1596	0.0007	174.4219
9	0.0279	1.0261	2.1467	1.4652	2.1467	2.1304	0.0025	175.5118
10	0.0217	1.0326	2.1606	1.4699	2.1606	2.6657	0.0050	171.1494
11	0.0153	1.0317	2.1746	1.4747	2.1746	2.1274	0.0075	169.2681
12	0.0267	1.0274	2.1494	1.4661	2.1494	2.2498	0.0093	184.2153
13	0.0157	1.0316	2.1738	1.4744	2.1738	2.0713	0.0100	177.8762
14	0.0314	1.0171	2.1391	1.4626	2.1391	3.1309	0.0093	179.8419
15	0.0435	1.0105	2.1124	1.4534	2.1124	2.4018	0.0075	180.9100
16	0.0533	0.9993	2.0908	1.4460	2.0908	2.1113	0.0050	186.2146
17	0.0703	0.9909	2.0533	1.4329	2.0533	2.2563	0.0025	180.4676
18	0.0725	0.9899	2.0484	1.4312	2.0484	2.1123	0.0007	174.3764

19	0.0720	0.9953	2.0495	1.4316	2.0495	2.0180	0.0000	172.6479
20	0.0764	0.9866	2.0396	1.4282	2.0396	2.2751	0.0007	173.4147
21	0.0770	0.9861	2.0384	1.4277	2.0384	2.0457	0.0025	186.4301
22	0.0537	0.9981	2.0900	1.4457	2.0900	2.1089	0.0050	180.7528
23	0.0577	1.0009	2.0810	1.4426	2.0810	2.2074	0.0075	177.8824
24	0.0492	1.0017	2.0999	1.4491	2.0999	2.7280	0.0093	178.1784
25	0.0504	1.0035	2.0970	1.4481	2.0970	2.7134	0.0100	188.6083
26	0.0449	1.0056	2.1094	1.4524	2.1094	2.0887	0.0093	187.0639
27	0.0472	1.0011	2.1042	1.4506	2.1042	2.1761	0.0075	193.0836
28	0.0694	0.9895	2.0551	1.4336	2.0551	2.1548	0.0050	188.2934
29	0.0828	0.9762	2.0257	1.4233	2.0257	2.7588	0.0025	189.2169
30	0.0885	0.9763	2.0131	1.4188	2.0131	2.0305	0.0007	193.2273
31	0.0924	0.9747	2.0044	1.4158	2.0044	2.0071	0.0000	174.5969
32	0.0920	0.9712	2.0053	1.4161	2.0053	2.0215	0.0007	179.5456
33	0.0876	0.9734	2.0151	1.4195	2.0151	2.3377	0.0025	175.6578
34	0.0740	0.9836	2.0451	1.4301	2.0451	2.0236	0.0050	184.9650
35	0.0659	0.9915	2.0628	1.4363	2.0628	2.0667	0.0075	200.2281
36	0.0520	1.0007	2.0936	1.4469	2.0936	2.0978	0.0093	214.7464
37	0.0635	0.9915	2.0683	1.4381	2.0683	2.1784	0.0100	193.2908
38	0.0554	0.9962	2.0861	1.4443	2.0861	2.2701	0.0093	172.5223
39	0.0551	0.9949	2.0868	1.4446	2.0868	2.0599	0.0075	181.7656
40	0.0776	0.9815	2.0372	1.4273	2.0372	2.0040	0.0050	172.7885
41	0.0857	0.9801	2.0193	1.4210	2.0193	2.2095	0.0025	185.8085
42	0.0929	0.9710	2.0033	1.4154	2.0033	1.9791	0.0007	182.2199
43	0.1059	0.9729	1.9746	1.4052	1.9746	1.9815	0.0000	186.4814
44	0.0980	0.9725	1.9919	1.4114	1.9919	1.9961	0.0007	177.0164
45	0.0932	0.9742	2.0025	1.4151	2.0025	2.0419	0.0025	180.2396
46	0.0861	0.9785	2.0182	1.4206	2.0182	2.0049	0.0050	181.0697
47	0.0673	0.9922	2.0599	1.4352	2.0599	2.1438	0.0075	173.7155
48	0.0679	0.9940	2.0585	1.4347	2.0585	2.9567	0.0093	184.0783
49	0.0558	0.9993	2.0853	1.4441	2.0853	2.4520	0.0100	180.7218
50	0.0624	0.9943	2.0706	1.4390	2.0706	2.3311	0.0093	177.2182
51	0.0787	0.9858	2.0346	1.4264	2.0346	2.7323	0.0075	172.7062
52	0.0776	0.9806	2.0371	1.4273	2.0371	2.4017	0.0050	171.9983
53	0.0938	0.9739	2.0012	1.4146	2.0012	2.4145	0.0025	177.0867
54	0.1032	0.9728	1.9805	1.4073	1.9805	2.1084	0.0007	182.6961
55	0.1063	0.9603	1.9738	1.4049	1.9738	1.9916	0.0000	180.2524
56	0.0956	0.9694	1.9973	1.4133	1.9973	1.9770	0.0007	182.9227
57	0.0938	0.9762	2.0012	1.4146	2.0012	2.2412	0.0025	182.1521
58	0.0843	0.9819	2.0223	1.4221	2.0223	2.4062	0.0050	173.0292
59	0.0736	0.9872	2.0460	1.4304	2.0460	3.9607	0.0075	181.3131
60	0.0663	0.9922	2.0621	1.4360	2.0621	2.0536	0.0093	178.4160
61	0.0670	0.9921	2.0605	1.4354	2.0605	2.1842	0.0100	187.1996

62	0.0690	0.9961	2.0561	1.4339	2.0561	2.1338	0.0093	180.1748
63	0.0729	0.9877	2.0475	1.4309	2.0475	2.0152	0.0075	180.3915
64	0.0757	0.9847	2.0412	1.4287	2.0412	2.5183	0.0050	181.4903
65	0.0878	0.9763	2.0144	1.4193	2.0144	2.0791	0.0025	177.4390
66	0.0994	0.9673	1.9889	1.4103	1.9889	2.0456	0.0007	185.8566
67	0.0979	0.9708	1.9922	1.4115	1.9922	1.9870	0.0000	175.2869
68	0.1049	0.9645	1.9769	1.4060	1.9769	1.9833	0.0007	185.7710
69	0.0949	0.9724	1.9989	1.4138	1.9989	2.1055	0.0025	184.9305
70	0.0947	0.9754	1.9993	1.4140	1.9993	2.1883	0.0050	185.9355
71	0.0760	0.9829	2.0405	1.4285	2.0405	2.2063	0.0075	176.5294
72	0.0693	0.9914	2.0554	1.4337	2.0554	3.2356	0.0093	173.1423
73	0.0688	0.9915	2.0566	1.4341	2.0566	2.1887	0.0100	185.3991
74	0.0647	0.9960	2.0656	1.4372	2.0656	2.4247	0.0093	183.4840
75	0.0750	0.9878	2.0429	1.4293	2.0429	2.1481	0.0075	173.4609
76	0.0818	0.9829	2.0277	1.4240	2.0277	2.0430	0.0050	178.7114
77	0.0940	0.9741	2.0009	1.4145	2.0009	2.0530	0.0025	173.4691
78	0.1050	0.9721	1.9765	1.4059	1.9765	2.0002	0.0007	179.5680
79	0.1023	0.9676	1.9826	1.4081	1.9826	1.9839	0.0000	184.2147
80	0.1075	0.9689	1.9710	1.4039	1.9710	1.9717	0.0007	182.9588
81	0.1019	0.9730	1.9835	1.4084	1.9835	1.9693	0.0025	187.4929
82	0.0920	0.9780	2.0054	1.4161	2.0054	2.1816	0.0050	195.2118
83	0.0844	0.9847	2.0221	1.4220	2.0221	2.0471	0.0075	197.0166
84	0.0721	0.9892	2.0492	1.4315	2.0492	2.9624	0.0093	183.1075
85	0.0682	0.9924	2.0579	1.4345	2.0579	2.4256	0.0100	177.2793
86	0.0606	0.9942	2.0746	1.4403	2.0746	2.0926	0.0093	181.2345
87	0.0754	0.9856	2.0419	1.4290	2.0419	2.0795	0.0075	177.1984
88	0.0878	0.9765	2.0146	1.4194	2.0146	2.0488	0.0050	177.7437
89	0.1034	0.9713	1.9802	1.4072	1.9802	2.6796	0.0025	177.9221
90	0.1016	0.9693	1.9841	1.4086	1.9841	2.0752	0.0007	181.6250
91	0.1163	0.9704	1.9516	1.3970	1.9516	1.9656	0.0000	176.8586
92	0.1066	0.9674	1.9731	1.4047	1.9731	1.9694	0.0007	183.0141
93	0.1112	0.9693	1.9630	1.4011	1.9630	3.2439	0.0025	181.5563
94	0.0921	0.9780	2.0050	1.4160	2.0050	2.3978	0.0050	179.0432
95	0.0798	0.9843	2.0322	1.4256	2.0322	2.4201	0.0075	180.6949
96	0.0682	0.9947	2.0578	1.4345	2.0578	2.3391	0.0093	179.2406
97	0.0707	0.9909	2.0524	1.4326	2.0524	2.1833	0.0100	176.9022
98	0.0722	0.9862	2.0490	1.4314	2.0490	2.3993	0.0093	175.0607
99	0.0736	0.9912	2.0460	1.4304	2.0460	2.1577	0.0075	176.1608
100	0.0891	0.9754	2.0116	1.4183	2.0116	2.7366	0.0050	187.0808
101	0.0923	0.9742	2.0046	1.4158	2.0046	1.9794	0.0025	174.8121
102	0.1191	0.9640	1.9454	1.3948	1.9454	1.9653	0.0007	169.1208
103	0.1169	0.9552	1.9503	1.3965	1.9503	1.9364	0.0000	173.2376
104	0.1113	0.9637	1.9627	1.4010	1.9627	1.9413	0.0007	187.4290

105	0.1131	0.9670	1.9588	1.3996	1.9588	2.5569	0.0025	182.7484
106	0.0928	0.9745	2.0035	1.4154	2.0035	2.4549	0.0050	179.4856
107	0.0823	0.9804	2.0266	1.4236	2.0266	2.4165	0.0075	175.2209
108	0.0737	0.9877	2.0457	1.4303	2.0457	2.3417	0.0093	173.2826
109	0.0724	0.9892	2.0486	1.4313	2.0486	2.0431	0.0100	186.1363
110	0.0694	0.9900	2.0553	1.4336	2.0553	2.2226	0.0093	183.8218
111	0.0733	0.9871	2.0467	1.4306	2.0467	2.1542	0.0075	185.4605
112	0.0979	0.9707	1.9922	1.4114	1.9922	2.0242	0.0050	179.0582
113	0.1053	0.9699	1.9758	1.4056	1.9758	1.9705	0.0025	184.5308
114	0.1212	0.9585	1.9407	1.3931	1.9407	1.9264	0.0007	191.2914
115	0.1143	0.9754	1.9560	1.3986	1.9560	1.9165	0.0000	187.8398
116	0.1222	0.9611	1.9387	1.3924	1.9387	1.9227	0.0007	184.4729
117	0.1211	0.9591	1.9410	1.3932	1.9410	2.2315	0.0025	180.6709
118	0.1068	0.9683	1.9726	1.4045	1.9726	1.9509	0.0050	183.4890
119	0.0969	0.9764	1.9944	1.4122	1.9944	2.0020	0.0075	187.0373
120	0.0592	0.9954	2.0777	1.4414	2.0777	3.2405	0.0093	187.7598
121	0.0579	0.9952	2.0805	1.4424	2.0805	2.0891	0.0100	187.4999
122	0.0581	0.9953	2.0801	1.4422	2.0801	2.0846	0.0093	191.1131
123	0.0562	0.9938	2.0844	1.4438	2.0844	2.3363	0.0075	179.4186
124	0.0779	0.9845	2.0363	1.4270	2.0363	2.4559	0.0050	196.6884
125	0.0950	0.9718	1.9988	1.4138	1.9988	1.9787	0.0025	188.3830
126	0.1065	0.9675	1.9732	1.4047	1.9732	1.9644	0.0007	194.5681
127	0.1052	0.9654	1.9760	1.4057	1.9760	1.9774	0.0000	192.2691
128	0.1080	0.9660	1.9700	1.4036	1.9700	1.9436	0.0007	185.7464
129	0.1049	0.9616	1.9767	1.4060	1.9767	2.7494	0.0025	183.8791
130	0.0962	0.9752	1.9960	1.4128	1.9960	2.0489	0.0050	183.5348
131	0.0815	0.9806	2.0284	1.4242	2.0284	3.5810	0.0075	184.3372
132	0.0697	0.9886	2.0546	1.4334	2.0546	2.0397	0.0093	184.1190
133	0.0710	0.9878	2.0516	1.4323	2.0516	3.1168	0.0100	187.0166
134	0.0621	0.9909	2.0714	1.4392	2.0714	2.4085	0.0093	187.5611
135	0.0809	0.9821	2.0299	1.4247	2.0299	2.5667	0.0075	176.5539
136	0.0892	0.9755	2.0114	1.4183	2.0114	2.2466	0.0050	181.6901
137	0.1077	0.9645	1.9706	1.4038	1.9706	2.0661	0.0025	185.1020
138	0.1252	0.9496	1.9320	1.3900	1.9320	2.1661	0.0007	187.5011
139	0.1266	0.9661	1.9288	1.3888	1.9288	1.9205	0.0000	191.8022
140	0.1223	0.9582	1.9383	1.3922	1.9383	1.9566	0.0007	184.6303
141	0.1124	0.9637	1.9602	1.4001	1.9602	1.9726	0.0025	182.8405
142	0.0991	0.9692	1.9896	1.4105	1.9896	1.9816	0.0050	187.1610
143	0.0893	0.9782	2.0112	1.4182	2.0112	2.4317	0.0075	182.9361
144	0.0723	0.9908	2.0488	1.4314	2.0488	2.4750	0.0093	185.8564
145	0.0737	0.9868	2.0457	1.4303	2.0457	3.0584	0.0100	193.6263
146	0.0727	0.9844	2.0479	1.4310	2.0479	2.0643	0.0093	191.9968
147	0.0770	0.9836	2.0383	1.4277	2.0383	2.1688	0.0075	188.2000

148	0.0986	0.9731	1.9908	1.4110	1.9908	2.1292	0.0050	187.8469
149	0.1115	0.9632	1.9622	1.4008	1.9622	1.9519	0.0025	184.9457
150	0.1209	0.9586	1.9415	1.3934	1.9415	1.9467	0.0007	183.8620
151	0.1237	0.9502	1.9352	1.3911	1.9352	1.9212	0.0000	185.9154
152	0.1355	0.9501	1.9093	1.3818	1.9093	1.9333	0.0007	183.6249
153	0.1106	0.9635	1.9641	1.4015	1.9641	1.9748	0.0025	181.0372
154	0.0992	0.9706	1.9893	1.4104	1.9893	2.1476	0.0050	184.3629
155	0.0925	0.9766	2.0041	1.4157	2.0041	2.0840	0.0075	182.5931
156	0.0746	0.9857	2.0437	1.4296	2.0437	2.3439	0.0093	184.8304
157	0.0755	0.9882	2.0417	1.4289	2.0417	2.1097	0.0100	181.6902
158	0.0795	0.9857	2.0330	1.4258	2.0330	2.5392	0.0093	188.9532
159	0.0816	0.9811	2.0283	1.4242	2.0283	2.8520	0.0075	179.8711
160	0.0911	0.9754	2.0073	1.4168	2.0073	2.5336	0.0050	183.0871
161	0.1194	0.9611	1.9448	1.3946	1.9448	1.9376	0.0025	182.5252
162	0.1282	0.9587	1.9254	1.3876	1.9254	1.9916	0.0007	188.1945
163	0.1385	0.9472	1.9027	1.3794	1.9027	1.9300	0.0000	177.7561
164	0.1207	0.9615	1.9418	1.3935	1.9418	2.2180	0.0007	180.4875
165	0.1176	0.9642	1.9488	1.3960	1.9488	2.0292	0.0025	182.2517
166	0.1051	0.9693	1.9763	1.4058	1.9763	2.5142	0.0050	194.4656
167	0.0929	0.9771	2.0033	1.4154	2.0033	2.0704	0.0075	176.1750
168	0.0791	0.9828	2.0337	1.4261	2.0337	2.4587	0.0093	177.0597
169	0.0779	0.9800	2.0365	1.4271	2.0365	2.0360	0.0100	176.5038
170	0.0826	0.9783	2.0260	1.4234	2.0260	2.1086	0.0093	178.0955
171	0.0928	0.9764	2.0035	1.4155	2.0035	2.0856	0.0075	179.8905
172	0.1129	0.9655	1.9591	1.3997	1.9591	2.0651	0.0050	178.8091
173	0.1235	0.9632	1.9357	1.3913	1.9357	2.1160	0.0025	176.4173
174	0.1333	0.9547	1.9140	1.3835	1.9140	1.9076	0.0007	176.7937
175	0.1343	0.9727	1.9119	1.3827	1.9119	1.8921	0.0000	184.5487
176	0.1382	0.9473	1.9032	1.3796	1.9032	1.9142	0.0007	166.9520
177	0.1312	0.9543	1.9188	1.3852	1.9188	1.9489	0.0025	181.7176
178	0.1143	0.9674	1.9561	1.3986	1.9561	1.9673	0.0050	304.9427
179	0.0983	0.9714	1.9914	1.4112	1.9914	2.1828	0.0075	182.3963
180	0.0859	0.9822	2.0188	1.4209	2.0188	2.1372	0.0093	183.8516
181	0.0868	0.9797	2.0167	1.4201	2.0167	2.2748	0.0100	187.1697
182	0.0882	0.9799	2.0136	1.4190	2.0136	2.1888	0.0093	182.9803
183	0.0974	0.9734	1.9933	1.4118	1.9933	2.2108	0.0075	186.5799
184	0.1120	0.9662	1.9612	1.4004	1.9612	1.9929	0.0050	192.3399
185	0.1254	0.9590	1.9315	1.3898	1.9315	2.0383	0.0025	185.8739
186	0.1423	0.9499	1.8942	1.3763	1.8942	1.8946	0.0007	190.2512
187	0.1512	0.9536	1.8746	1.3692	1.8746	1.8677	0.0000	183.1572
188	0.1430	0.9530	1.8926	1.3757	1.8926	1.8653	0.0007	192.3623
189	0.1331	0.9536	1.9144	1.3836	1.9144	2.1707	0.0025	191.4093
190	0.1229	0.9604	1.9371	1.3918	1.9371	2.4032	0.0050	186.8925

191	0.1023	0.9721	1.9826	1.4080	1.9826	1.9808	0.0075	191.6983
192	0.0960	0.9778	1.9963	1.4129	1.9963	2.1456	0.0093	181.6166
193	0.0817	0.9837	2.0280	1.4241	2.0280	2.4208	0.0100	188.4556
194	0.0811	0.9841	2.0294	1.4246	2.0294	2.2242	0.0093	198.3581
195	0.0923	0.9786	2.0046	1.4158	2.0046	3.0146	0.0075	185.7925
196	0.1133	0.9669	1.9582	1.3994	1.9582	3.0449	0.0050	184.4025
197	0.1297	0.9573	1.9220	1.3864	1.9220	1.9662	0.0025	180.3765
198	0.1402	0.9567	1.8988	1.3780	1.8988	1.8686	0.0007	189.3391
199	0.1403	0.9593	1.8985	1.3779	1.8985	1.8836	0.0000	181.9914
200	0.1482	0.9489	1.8811	1.3715	1.8811	1.8899	0.0007	179.2618
Mean Absolute Error	0.89							
Mean Squared Error	1.84							
Root Mean Squared Error	1.36							
Coefficient of Determination	0.15							

Table A.9.: Third training results of EEGNet for Regression

A.8.4. Forth Training

ID	4							
Date	14.12.24							
Model Definition	EEGNetv4(n_outputs=n_classes, n_chans=n_channels, n_times=sampling_rate, final_conv_length=auto)							
Classifier Definition	criterion=torch.nn.HuberLoss, optimizer=torch.optim.AdamW, optimizer__lr=0.001, train_split=predefined_split(val_dataset), batch_size=16, callbacks=callbacks							
Training Epochs								
Epoch	CoD	MAE	MSE	RMSE	tr_loss	val_loss	lr	dur
1	-0.3498	1.1841	3.0305	1.7408	0.8017	0.7030	0.0010	187.9177
2	-0.1469	1.0489	2.5750	1.6047	0.6842	0.5942	0.0009	181.2878
3	-0.1218	1.0265	2.5186	1.5870	0.6659	0.9272	0.0008	184.0841
4	-0.1228	1.0228	2.5208	1.5877	0.6643	0.5861	0.0005	176.3475
5	-0.1211	1.0128	2.5170	1.5865	0.6583	0.6178	0.0003	180.8175
6	-0.1027	1.0113	2.4758	1.5735	0.6547	0.5878	0.0001	186.9976
7	-0.1090	1.0118	2.4899	1.5779	0.6550	0.5858	0.0000	189.2606
8	-0.1122	1.0085	2.4970	1.5802	0.6541	0.5854	0.0001	224.8274

9	-0.1072	1.0097	2.4858	1.5767	0.6537	0.6077	0.0003	193.8915
10	-0.1056	1.0068	2.4821	1.5755	0.6524	0.6632	0.0005	176.7063
11	-0.1093	1.0044	2.4905	1.5781	0.6524	0.6358	0.0008	185.5902
12	-0.1076	1.0050	2.4868	1.5769	0.6520	0.6193	0.0009	189.0188
13	-0.0919	1.0002	2.4514	1.5657	0.6467	0.5850	0.0010	183.7406
14	-0.0827	0.9993	2.4308	1.5591	0.6439	0.6703	0.0009	190.3106
15	-0.0839	0.9909	2.4334	1.5599	0.6403	0.8727	0.0008	188.4155
16	-0.0716	0.9826	2.4059	1.5511	0.6333	0.5848	0.0005	186.1252
17	-0.0731	0.9861	2.4092	1.5522	0.6350	0.5820	0.0003	189.4943
18	-0.0679	0.9774	2.3976	1.5484	0.6300	0.5814	0.0001	189.2428
19	-0.0823	0.9815	2.4300	1.5588	0.6330	0.5814	0.0000	186.2622
20	-0.0697	0.9812	2.4015	1.5497	0.6323	0.5810	0.0001	186.7699
21	-0.0755	0.9861	2.4147	1.5539	0.6355	0.6075	0.0003	186.5668
22	-0.0752	0.9866	2.4139	1.5537	0.6355	0.5874	0.0005	190.0452
23	-0.0704	0.9777	2.4031	1.5502	0.6303	0.5819	0.0008	193.0485
24	-0.0650	0.9780	2.3909	1.5463	0.6292	0.5918	0.0009	203.8972
25	-0.0643	0.9808	2.3895	1.5458	0.6313	0.5858	0.0010	188.3294
26	-0.0644	0.9786	2.3896	1.5458	0.6291	0.6373	0.0009	189.0065
27	-0.0618	0.9734	2.3839	1.5440	0.6259	0.5814	0.0008	190.2507
28	-0.0518	0.9634	2.3615	1.5367	0.6190	0.5836	0.0005	194.7095
29	-0.0642	0.9703	2.3892	1.5457	0.6242	0.5806	0.0003	191.8256
30	-0.0461	0.9628	2.3487	1.5325	0.6177	0.5797	0.0001	187.7594
31	-0.0462	0.9581	2.3489	1.5326	0.6152	0.5798	0.0000	196.4950
32	-0.0612	0.9664	2.3826	1.5436	0.6221	0.5817	0.0001	192.5143
33	-0.0464	0.9618	2.3494	1.5328	0.6182	0.5789	0.0003	184.7918
34	-0.0524	0.9647	2.3627	1.5371	0.6195	0.6074	0.0005	193.9980
35	-0.0469	0.9644	2.3503	1.5331	0.6190	0.5913	0.0008	186.1309
36	-0.0488	0.9633	2.3548	1.5345	0.6187	0.5818	0.0009	190.2480
37	-0.0564	0.9631	2.3717	1.5400	0.6202	0.5963	0.0010	192.2249
38	-0.0557	0.9662	2.3701	1.5395	0.6213	0.5805	0.0009	176.4708
39	-0.0486	0.9586	2.3542	1.5343	0.6161	0.6019	0.0008	200.8534
40	-0.0407	0.9536	2.3365	1.5286	0.6119	0.5786	0.0005	191.8141
41	-0.0414	0.9507	2.3381	1.5291	0.6107	0.5849	0.0003	194.3440
42	-0.0357	0.9507	2.3252	1.5249	0.6096	0.5771	0.0001	188.5797
43	-0.0319	0.9453	2.3166	1.5221	0.6060	0.5774	0.0000	189.0305
44	-0.0385	0.9501	2.3314	1.5269	0.6100	0.5826	0.0001	189.4256
45	-0.0421	0.9482	2.3397	1.5296	0.6087	0.5766	0.0003	190.0534
46	-0.0392	0.9503	2.3330	1.5274	0.6095	0.5832	0.0005	204.3330
47	-0.0398	0.9514	2.3345	1.5279	0.6106	0.5825	0.0008	194.9726
48	-0.0411	0.9542	2.3374	1.5289	0.6119	0.8018	0.0009	192.9073
49	-0.0488	0.9543	2.3546	1.5345	0.6147	0.6650	0.0010	182.8947
50	-0.0334	0.9452	2.3202	1.5232	0.6074	0.6060	0.0009	187.4191
51	-0.0365	0.9501	2.3270	1.5255	0.6088	0.5805	0.0008	185.3124

52	-0.0340	0.9424	2.3214	1.5236	0.6052	0.5793	0.0005	195.0592
53	-0.0240	0.9355	2.2991	1.5163	0.6005	0.5815	0.0003	193.5252
54	-0.0267	0.9395	2.3050	1.5182	0.6026	0.5772	0.0001	192.3636
55	-0.0206	0.9361	2.2914	1.5137	0.5990	0.5784	0.0000	191.6893
56	-0.0219	0.9354	2.2944	1.5147	0.5997	0.5796	0.0001	186.4883
57	-0.0299	0.9386	2.3122	1.5206	0.6022	0.5938	0.0002	184.1985
58	-0.0285	0.9398	2.3090	1.5196	0.6028	0.5768	0.0005	187.5834
59	-0.0263	0.9386	2.3041	1.5179	0.6018	0.6018	0.0007	187.2096
60	-0.0357	0.9386	2.3254	1.5249	0.6027	0.6002	0.0009	189.4817
61	-0.0261	0.9393	2.3037	1.5178	0.6030	0.5988	0.0010	195.5894
62	-0.0182	0.9293	2.2861	1.5120	0.5958	0.5931	0.0009	196.8328
63	-0.0306	0.9353	2.3137	1.5211	0.6011	0.6203	0.0007	186.5415
64	-0.0252	0.9303	2.3016	1.5171	0.5970	0.5770	0.0005	184.5958
65	-0.0155	0.9265	2.2798	1.5099	0.5940	0.5854	0.0002	188.1104
66	-0.0140	0.9270	2.2766	1.5088	0.5923	0.5751	0.0001	192.0792
67	-0.0322	0.9216	2.3175	1.5223	0.5935	0.5755	0.0000	187.0533
68	-0.0139	0.9244	2.2763	1.5087	0.5915	0.5758	0.0001	177.7266
69	-0.0126	0.9228	2.2734	1.5078	0.5900	0.5911	0.0003	191.1974
70	-0.0154	0.9292	2.2796	1.5098	0.5941	0.5833	0.0005	179.6114
71	-0.0147	0.9275	2.2782	1.5094	0.5947	0.5898	0.0008	203.9528
72	-0.0142	0.9251	2.2769	1.5089	0.5922	0.5955	0.0009	189.6234
73	-0.0142	0.9263	2.2771	1.5090	0.5922	0.5811	0.0010	190.1540
74	-0.0110	0.9245	2.2698	1.5066	0.5911	0.5801	0.0009	194.8339
75	-0.0166	0.9255	2.2823	1.5107	0.5925	0.5754	0.0008	182.9987
76	-0.0039	0.9165	2.2540	1.5013	0.5855	0.6157	0.0005	180.9931
77	0.0040	0.9134	2.2360	1.4953	0.5822	0.5742	0.0003	187.4900
78	0.0011	0.9134	2.2427	1.4976	0.5837	0.5759	0.0001	184.5070
79	0.0074	0.9141	2.2286	1.4929	0.5830	0.5741	0.0000	186.2165
80	-0.0024	0.9133	2.2505	1.5002	0.5845	0.5731	0.0001	186.3858
81	-0.0017	0.9134	2.2489	1.4996	0.5838	0.5993	0.0003	184.7841
82	-0.0052	0.9183	2.2568	1.5023	0.5871	0.5815	0.0005	199.5206
83	-0.0048	0.9190	2.2560	1.5020	0.5882	0.5862	0.0008	191.1193
84	-0.0099	0.9231	2.2674	1.5058	0.5900	0.5743	0.0009	195.3166
85	-0.0110	0.9167	2.2698	1.5066	0.5878	0.6115	0.0010	187.9013
86	-0.0051	0.9182	2.2566	1.5022	0.5873	0.5812	0.0009	181.2491
87	-0.0032	0.9151	2.2522	1.5007	0.5848	0.5698	0.0008	184.1682
88	0.0025	0.9125	2.2396	1.4965	0.5822	0.5717	0.0005	187.6174
89	0.0097	0.9084	2.2234	1.4911	0.5792	0.5820	0.0003	191.4298
90	0.0131	0.9062	2.2156	1.4885	0.5777	0.5694	0.0001	211.4548
91	0.0059	0.9039	2.2319	1.4939	0.5773	0.5696	0.0000	188.3544
92	0.0086	0.9072	2.2259	1.4919	0.5789	0.5690	0.0001	184.3635
93	0.0114	0.9074	2.2196	1.4898	0.5781	0.5730	0.0002	188.3313
94	0.0114	0.9071	2.2195	1.4898	0.5787	0.5746	0.0005	190.8528

95	0.0056	0.9109	2.2324	1.4941	0.5811	0.5777	0.0008	188.4130
96	-0.0047	0.9158	2.2557	1.5019	0.5847	0.6558	0.0009	193.3539
97	0.0054	0.9110	2.2329	1.4943	0.5821	0.5757	0.0010	177.5285
98	0.0105	0.9095	2.2216	1.4905	0.5797	0.5759	0.0009	191.5631
99	0.0205	0.9062	2.1991	1.4829	0.5750	0.5937	0.0008	191.6838
100	0.0170	0.9050	2.2069	1.4856	0.5764	0.5639	0.0005	186.6238
101	0.0287	0.9012	2.1806	1.4767	0.5719	0.5601	0.0003	188.6743
102	0.0306	0.8952	2.1763	1.4752	0.5682	0.5605	0.0001	197.1948
103	0.0343	0.8938	2.1681	1.4724	0.5668	0.5603	0.0000	195.0347
104	0.0296	0.8962	2.1785	1.4760	0.5687	0.5603	0.0001	180.8192
105	0.0270	0.9008	2.1844	1.4780	0.5714	0.5584	0.0003	181.7684
106	0.0277	0.8979	2.1828	1.4774	0.5696	0.5744	0.0005	187.4188
107	0.0269	0.8991	2.1848	1.4781	0.5713	0.5654	0.0008	182.2106
108	0.0262	0.9019	2.1864	1.4786	0.5724	0.5800	0.0009	191.4389
109	0.0311	0.8991	2.1753	1.4749	0.5702	0.5901	0.0010	196.2558
110	0.0285	0.9001	2.1811	1.4768	0.5721	0.5786	0.0009	191.0572
111	0.0317	0.8970	2.1740	1.4744	0.5690	0.5961	0.0008	185.8133
112	0.0381	0.8911	2.1596	1.4696	0.5651	0.5492	0.0005	189.9630
113	0.0442	0.8837	2.1458	1.4649	0.5597	0.5550	0.0003	191.0307
114	0.0565	0.8852	2.1183	1.4554	0.5576	0.5485	0.0001	187.2875
115	0.0527	0.8813	2.1267	1.4583	0.5563	0.5484	0.0000	185.2089
116	0.0565	0.8821	2.1182	1.4554	0.5565	0.5525	0.0001	186.7559
117	0.0587	0.8838	2.1134	1.4537	0.5573	0.5655	0.0002	186.5934
118	0.0475	0.8861	2.1385	1.4624	0.5605	0.5493	0.0005	187.0162
119	0.0457	0.8893	2.1425	1.4637	0.5628	0.5490	0.0007	186.3050
120	0.0432	0.8931	2.1480	1.4656	0.5648	0.5740	0.0009	185.2980
121	0.0444	0.8925	2.1455	1.4647	0.5646	0.5513	0.0010	188.3141
122	0.0474	0.8898	2.1387	1.4624	0.5624	0.5526	0.0009	188.3067
123	0.0557	0.8863	2.1200	1.4560	0.5592	0.5480	0.0007	190.4722
124	0.0552	0.8819	2.1211	1.4564	0.5566	0.5509	0.0005	184.1732
125	0.0686	0.8769	2.0911	1.4461	0.5516	0.5501	0.0002	190.1379
126	0.0840	0.8704	2.0566	1.4341	0.5467	0.5396	0.0001	185.1583
127	0.0691	0.8724	2.0899	1.4457	0.5486	0.5396	0.0000	193.1543
128	0.0781	0.8732	2.0698	1.4387	0.5477	0.5405	0.0001	199.9866
129	0.0764	0.8743	2.0735	1.4400	0.5489	0.5448	0.0002	182.0414
130	0.0772	0.8768	2.0719	1.4394	0.5502	0.5761	0.0005	181.2524
131	0.0677	0.8788	2.0932	1.4468	0.5528	0.5657	0.0007	185.5206
132	0.0563	0.8851	2.1187	1.4556	0.5584	0.5699	0.0009	179.8091
133	0.0488	0.8905	2.1355	1.4613	0.5626	0.5744	0.0010	182.0383
134	0.0535	0.8847	2.1250	1.4577	0.5586	0.6813	0.0009	191.3310
135	0.0657	0.8803	2.0976	1.4483	0.5535	0.5407	0.0007	188.8991
136	0.0686	0.8772	2.0911	1.4461	0.5521	0.5522	0.0005	193.7789
137	0.0861	0.8705	2.0519	1.4324	0.5449	0.5367	0.0002	187.3301

138	0.0945	0.8654	2.0330	1.4258	0.5417	0.5349	0.0001	184.8024
139	0.1033	0.8608	2.0132	1.4189	0.5373	0.5330	0.0000	195.2261
140	0.0939	0.8677	2.0343	1.4263	0.5426	0.5351	0.0001	187.7545
141	0.0904	0.8677	2.0422	1.4290	0.5437	0.5384	0.0003	196.2041
142	0.0819	0.8761	2.0611	1.4357	0.5493	0.5589	0.0005	184.4552
143	0.0670	0.8822	2.0946	1.4473	0.5557	0.5389	0.0008	192.8860
144	0.0707	0.8853	2.0864	1.4444	0.5553	0.5509	0.0009	191.2397
145	0.0636	0.8826	2.1023	1.4499	0.5560	0.5405	0.0010	185.6324
146	0.0695	0.8824	2.0891	1.4454	0.5547	0.5815	0.0009	185.0767
147	0.0714	0.8783	2.0847	1.4439	0.5519	0.5399	0.0008	196.6503
148	0.0834	0.8713	2.0578	1.4345	0.5463	0.5387	0.0005	195.0629
149	0.0899	0.8707	2.0432	1.4294	0.5451	0.5385	0.0003	240.0038
150	0.1076	0.8608	2.0036	1.4155	0.5363	0.5294	0.0001	198.5873
151	0.1082	0.8623	2.0022	1.4150	0.5369	0.5263	0.0000	190.3601
152	0.1098	0.8605	1.9985	1.4137	0.5362	0.5284	0.0001	188.2519
153	0.1063	0.8649	2.0064	1.4165	0.5385	0.5315	0.0003	195.3985
154	0.0850	0.8729	2.0543	1.4333	0.5475	0.5598	0.0005	183.5568
155	0.0758	0.8792	2.0748	1.4404	0.5521	0.5399	0.0008	197.9733
156	0.0678	0.8812	2.0928	1.4467	0.5535	0.5705	0.0009	191.4203
157	0.0701	0.8807	2.0878	1.4449	0.5538	0.5501	0.0010	185.3297
158	0.0744	0.8777	2.0781	1.4416	0.5515	0.5436	0.0009	181.0478
159	0.0837	0.8708	2.0573	1.4343	0.5460	0.5507	0.0008	189.5151
160	0.0938	0.8674	2.0345	1.4264	0.5437	0.5392	0.0005	191.7602
161	0.1117	0.8593	1.9942	1.4122	0.5357	0.5254	0.0003	184.4407
162	0.1197	0.8569	1.9764	1.4058	0.5319	0.5239	0.0001	181.7902
163	0.1189	0.8527	1.9782	1.4065	0.5294	0.5219	0.0000	190.7141
164	0.1216	0.8517	1.9720	1.4043	0.5289	0.5224	0.0001	178.7995
165	0.1132	0.8581	1.9909	1.4110	0.5342	0.5284	0.0002	193.9408
166	0.1059	0.8640	2.0074	1.4168	0.5399	0.5270	0.0005	201.8978
167	0.0974	0.8704	2.0265	1.4235	0.5439	0.5811	0.0008	180.8566
168	0.0887	0.8741	2.0459	1.4303	0.5473	0.5342	0.0009	189.8165
169	0.0809	0.8716	2.0635	1.4365	0.5473	0.5523	0.0010	184.1478
170	0.0808	0.8743	2.0637	1.4366	0.5481	0.5369	0.0009	190.3453
171	0.0949	0.8672	2.0321	1.4255	0.5421	0.5273	0.0008	188.1608
172	0.0980	0.8656	2.0250	1.4230	0.5411	0.5247	0.0005	190.0704
173	0.1176	0.8578	1.9810	1.4075	0.5331	0.5211	0.0003	186.4963
174	0.1320	0.8490	1.9487	1.3960	0.5257	0.5232	0.0001	185.1851
175	0.1330	0.8490	1.9464	1.3951	0.5256	0.5158	0.0000	192.1226
176	0.1385	0.8486	1.9342	1.3908	0.5241	0.5107	0.0001	187.2969
177	0.1331	0.8536	1.9463	1.3951	0.5265	0.5293	0.0003	182.9602
178	0.1144	0.8596	1.9883	1.4101	0.5351	0.5902	0.0005	184.4501
179	0.1057	0.8707	2.0077	1.4169	0.5431	0.5801	0.0008	191.3567
180	0.1017	0.8687	2.0168	1.4201	0.5421	0.6455	0.0009	188.1606

181	0.0855	0.8733	2.0532	1.4329	0.5473	0.5420	0.0010	188.34
182	0.0992	0.8682	2.0223	1.4221	0.5420	0.5414	0.0009	182.93
183	0.1029	0.8646	2.0142	1.4192	0.5393	0.5483	0.0008	189.34
184	0.1180	0.8600	1.9802	1.4072	0.5340	0.5235	0.0005	191.69
185	0.1357	0.8522	1.9404	1.3930	0.5252	0.5120	0.0003	191.52
186	0.1490	0.8457	1.9106	1.3823	0.5202	0.5091	0.0001	190.06
187	0.1519	0.8444	1.9040	1.3799	0.5196	0.5070	0.0000	189.26
188	0.1486	0.8481	1.9115	1.3826	0.5214	0.5081	0.0001	191.99
189	0.1386	0.8535	1.9340	1.3907	0.5259	0.5073	0.0002	193.05
190	0.1273	0.8572	1.9594	1.3998	0.5304	0.5340	0.0005	187.16
191	0.1108	0.8641	1.9962	1.4129	0.5368	0.5684	0.0007	180.84
192	0.1152	0.8661	1.9865	1.4094	0.5387	0.5225	0.0009	188.67
193	0.1034	0.8663	2.0131	1.4188	0.5412	0.5826	0.0010	195.11
194	0.0981	0.8696	2.0249	1.4230	0.5425	0.5457	0.0009	196.35
195	0.1146	0.8616	1.9879	1.4099	0.5355	0.5963	0.0007	193.95
196	0.1254	0.8566	1.9636	1.4013	0.5302	0.5213	0.0005	191.62
197	0.1446	0.8444	1.9204	1.3858	0.5201	0.5040	0.0002	195.77
198	0.1549	0.8419	1.8974	1.3775	0.5156	0.4992	0.0001	190.14
199	0.1526	0.8402	1.9025	1.3793	0.5167	0.4985	0.0000	176.50
200	0.1584	0.8424	1.8895	1.3746	0.5168	0.4991	0.0001	187.06
Mean Absolute Error	0.83							
Mean Squared Error	1.91							
Root Mean Squared Error	1.38							
Coefficient of Determination	0.16							

Table A.10.: Forth training results of EEGNet for Regression

A.8.5. Fifth Training

ID	5
Date	15.12.2024
Model Definition	EEGNetv4(n_outputs=n_classes, n_chans=n_channels, n_times=sampling_rate, final_conv_length=auto)
Classifier Definition	criterion=torch.nn.L1Loss, optimizer=torch.optim.AdamW, optimizer__lr=0.001, train_split=predefined_split(val_dataset), batch_size=16, callbacks=callbacks

Training Epochs								
Epoch	CoD	MAE	MSE	RMSE	tr_loss	val_loss	lr	dur
1	-0.4453	1.2271	3.2640	1.8066	1.2271	0.8517	0.0010	265.28
2	-0.1725	1.0587	2.6477	1.6272	1.0587	1.0412	0.0009	216.51
3	-0.1402	1.0340	2.5750	1.6047	1.0340	0.8775	0.0008	204.70
4	-0.1288	1.0202	2.5491	1.5966	1.0202	0.8493	0.0005	265.15
5	-0.1097	1.0114	2.5060	1.5830	1.0114	0.9154	0.0003	281.34
6	-0.1176	1.0177	2.5239	1.5887	1.0177	0.8772	0.0001	284.06
7	-0.1240	1.0143	2.5382	1.5932	1.0143	0.8665	0.0000	283.13
8	-0.1135	1.0140	2.5147	1.5858	1.0140	0.8562	0.0001	283.20
9	-0.1202	1.0079	2.5298	1.5905	1.0079	0.8888	0.0003	278.37
10	-0.1141	1.0105	2.5159	1.5862	1.0105	0.9024	0.0005	282.67
11	-0.1202	1.0166	2.5298	1.5905	1.0166	0.8684	0.0008	279.49
12	-0.1085	1.0073	2.5033	1.5822	1.0073	0.8631	0.0009	278.21
13	-0.1029	0.9995	2.4907	1.5782	0.9995	0.9139	0.0010	240.00
14	-0.1066	0.9948	2.4989	1.5808	0.9948	0.8665	0.0009	184.67
15	-0.0955	0.9847	2.4740	1.5729	0.9847	0.8912	0.0008	176.13
16	-0.1035	0.9915	2.4920	1.5786	0.9915	0.8649	0.0005	173.43
17	-0.0861	0.9856	2.4528	1.5661	0.9856	0.8621	0.0003	190.71
18	-0.0782	0.9751	2.4349	1.5604	0.9751	0.8757	0.0001	190.38
19	-0.0805	0.9778	2.4400	1.5620	0.9778	0.8711	0.0000	185.61
20	-0.0881	0.9789	2.4572	1.5675	0.9789	0.8635	0.0001	183.33
21	-0.0908	0.9785	2.4633	1.5695	0.9785	0.8843	0.0003	183.73
22	-0.0942	0.9830	2.4710	1.5720	0.9830	0.8862	0.0005	188.81
23	-0.0939	0.9782	2.4702	1.5717	0.9782	0.8519	0.0008	184.78
24	-0.0978	0.9842	2.4792	1.5745	0.9842	0.8616	0.0009	188.98
25	-0.0821	0.9748	2.4438	1.5633	0.9748	1.0633	0.0010	180.98
26	-0.0977	0.9747	2.4788	1.5744	0.9747	0.8613	0.0009	184.07
27	-0.0883	0.9726	2.4577	1.5677	0.9726	0.8568	0.0008	180.67
28	-0.0839	0.9674	2.4478	1.5645	0.9674	0.9255	0.0005	188.43
29	-0.0856	0.9626	2.4516	1.5658	0.9626	0.8685	0.0003	177.04
30	-0.0706	0.9590	2.4177	1.5549	0.9590	0.8566	0.0001	187.73
31	-0.0802	0.9603	2.4393	1.5618	0.9603	0.8633	0.0000	255.08
32	-0.0806	0.9635	2.4404	1.5622	0.9635	0.8791	0.0001	279.41
33	-0.0848	0.9607	2.4498	1.5652	0.9607	0.8556	0.0003	280.15
34	-0.0819	0.9603	2.4431	1.5630	0.9603	0.9584	0.0005	277.12
35	-0.0838	0.9664	2.4475	1.5645	0.9664	0.8692	0.0008	278.16
36	-0.0854	0.9641	2.4510	1.5656	0.9641	0.9457	0.0009	275.98
37	-0.0838	0.9582	2.4474	1.5644	0.9582	0.9084	0.0010	277.55
38	-0.0850	0.9607	2.4502	1.5653	0.9607	0.9091	0.0009	278.13
39	-0.0816	0.9568	2.4425	1.5629	0.9568	0.8644	0.0008	278.04
40	-0.0749	0.9479	2.4273	1.5580	0.9479	0.9339	0.0005	277.30
41	-0.0780	0.9510	2.4344	1.5602	0.9510	0.8737	0.0003	275.05

42	-0.0759	0.9480	2.4297	1.5588	0.9480	0.8607	0.0001	276.61
43	-0.0712	0.9445	2.4191	1.5553	0.9445	0.8556	0.0000	273.38
44	-0.0775	0.9453	2.4333	1.5599	0.9453	0.8698	0.0001	222.54
45	-0.0681	0.9401	2.4122	1.5531	0.9401	0.8436	0.0003	258.16
46	-0.0760	0.9484	2.4299	1.5588	0.9484	0.8821	0.0005	277.75
47	-0.0749	0.9447	2.4274	1.5580	0.9447	0.9347	0.0008	283.06
48	-0.0739	0.9459	2.4252	1.5573	0.9459	0.9008	0.0009	277.04
49	-0.0818	0.9445	2.4430	1.5630	0.9445	0.8699	0.0010	278.05
50	-0.0756	0.9431	2.4289	1.5585	0.9431	0.8655	0.0009	279.88
51	-0.0674	0.9344	2.4106	1.5526	0.9344	0.9392	0.0008	279.50
52	-0.0723	0.9347	2.4216	1.5561	0.9347	0.9245	0.0005	278.54
53	-0.0675	0.9323	2.4107	1.5526	0.9323	0.8373	0.0003	277.30
54	-0.0615	0.9234	2.3972	1.5483	0.9234	0.8820	0.0001	278.37
55	-0.0570	0.9274	2.3870	1.5450	0.9274	0.8553	0.0000	278.06
56	-0.0586	0.9280	2.3906	1.5462	0.9280	0.8523	0.0001	283.92
57	-0.0632	0.9275	2.4009	1.5495	0.9275	0.8458	0.0002	282.53
58	-0.0621	0.9282	2.3986	1.5487	0.9282	0.9492	0.0005	212.36
59	-0.0689	0.9305	2.4139	1.5537	0.9305	0.8443	0.0007	279.71
60	-0.0691	0.9323	2.4143	1.5538	0.9323	0.8404	0.0009	283.69
61	-0.0658	0.9279	2.4070	1.5514	0.9279	0.8766	0.0010	283.20
62	-0.0623	0.9203	2.3990	1.5489	0.9203	0.8959	0.0009	281.62
63	-0.0685	0.9246	2.4129	1.5533	0.9246	0.8450	0.0007	284.44
64	-0.0542	0.9182	2.3806	1.5429	0.9182	0.8393	0.0005	280.63
65	-0.0569	0.9165	2.3868	1.5449	0.9165	0.8762	0.0002	276.79
66	-0.0533	0.9130	2.3787	1.5423	0.9130	0.8564	0.0001	276.48
67	-0.0498	0.9123	2.3707	1.5397	0.9123	0.8490	0.0000	276.98
68	-0.0582	0.9171	2.3898	1.5459	0.9171	0.8672	0.0001	276.65
69	-0.0522	0.9114	2.3761	1.5415	0.9114	0.8937	0.0003	276.94
70	-0.0519	0.9133	2.3754	1.5412	0.9133	0.9259	0.0005	277.98
71	-0.0516	0.9127	2.3748	1.5410	0.9127	0.9869	0.0008	276.63
72	-0.0579	0.9154	2.3891	1.5457	0.9154	0.8622	0.0009	233.69
73	-0.0537	0.9126	2.3795	1.5426	0.9126	1.0416	0.0010	280.62
74	-0.0512	0.9089	2.3738	1.5407	0.9089	0.9168	0.0009	277.37
75	-0.0527	0.9074	2.3773	1.5418	0.9074	0.8897	0.0008	277.68
76	-0.0466	0.9015	2.3634	1.5373	0.9015	0.8548	0.0005	281.77
77	-0.0417	0.8988	2.3524	1.5337	0.8988	0.8792	0.0003	278.33
78	-0.0397	0.8982	2.3479	1.5323	0.8982	0.8443	0.0001	276.60
79	-0.0447	0.8984	2.3592	1.5360	0.8984	0.8388	0.0000	277.52
80	-0.0466	0.8999	2.3636	1.5374	0.8999	0.8507	0.0001	273.45
81	-0.0402	0.9000	2.3490	1.5326	0.9000	0.8811	0.0003	281.18
82	-0.0477	0.9009	2.3659	1.5382	0.9009	0.8418	0.0005	282.99
83	-0.0461	0.9017	2.3623	1.5370	0.9017	0.8611	0.0008	283.24
84	-0.0480	0.8981	2.3666	1.5384	0.8981	0.8449	0.0009	282.22

85	-0.0438	0.8992	2.3571	1.5353	0.8992	0.8632	0.0010	282.42
86	-0.0478	0.8957	2.3661	1.5382	0.8957	0.8516	0.0009	282.06
87	-0.0399	0.8927	2.3483	1.5324	0.8927	0.8565	0.0008	284.14
88	-0.0396	0.8924	2.3476	1.5322	0.8924	0.8343	0.0005	283.90
89	-0.0365	0.8883	2.3407	1.5299	0.8883	0.8297	0.0003	279.30
90	-0.0388	0.8875	2.3458	1.5316	0.8875	0.8307	0.0001	280.41
91	-0.0357	0.8834	2.3389	1.5293	0.8834	0.8361	0.0000	279.01
92	-0.0329	0.8841	2.3325	1.5272	0.8841	0.8328	0.0001	278.65
93	-0.0382	0.8882	2.3446	1.5312	0.8882	0.8892	0.0002	224.67
94	-0.0379	0.8891	2.3439	1.5310	0.8891	0.8288	0.0005	271.76
95	-0.0412	0.8907	2.3514	1.5334	0.8907	0.8373	0.0008	277.85
96	-0.0422	0.8900	2.3536	1.5341	0.8900	0.8390	0.0009	278.62
97	-0.0352	0.8880	2.3378	1.5290	0.8880	0.9593	0.0010	249.77
98	-0.0386	0.8847	2.3454	1.5315	0.8847	0.8705	0.0009	195.60
99	-0.0328	0.8833	2.3323	1.5272	0.8833	0.8771	0.0008	190.68
100	-0.0321	0.8805	2.3307	1.5267	0.8805	0.8406	0.0005	192.33
101	-0.0300	0.8770	2.3260	1.5251	0.8770	0.8286	0.0003	186.77
102	-0.0276	0.8757	2.3206	1.5233	0.8757	0.8388	0.0001	196.01
103	-0.0186	0.8760	2.3003	1.5167	0.8760	0.8363	0.0000	191.43
104	-0.0232	0.8746	2.3107	1.5201	0.8746	0.8448	0.0001	187.04
105	-0.0266	0.8753	2.3184	1.5226	0.8753	0.8625	0.0003	188.88
106	-0.0270	0.8768	2.3191	1.5229	0.8768	0.8681	0.0005	201.75
107	-0.0273	0.8774	2.3198	1.5231	0.8774	0.8250	0.0008	183.07
108	-0.0355	0.8787	2.3385	1.5292	0.8787	0.8797	0.0009	188.02
109	-0.0356	0.8804	2.3388	1.5293	0.8804	0.8552	0.0010	193.72
110	-0.0328	0.8754	2.3323	1.5272	0.8754	0.9336	0.0009	186.40
111	-0.0289	0.8730	2.3235	1.5243	0.8730	0.8827	0.0008	190.99
112	-0.0258	0.8684	2.3166	1.5221	0.8684	0.8899	0.0005	187.57
113	-0.0276	0.8680	2.3206	1.5234	0.8680	0.8305	0.0003	195.56
114	-0.0265	0.8655	2.3181	1.5225	0.8655	0.8256	0.0001	194.06
115	-0.0219	0.8640	2.3078	1.5191	0.8640	0.8303	0.0000	185.39
116	-0.0267	0.8648	2.3186	1.5227	0.8648	0.8229	0.0001	182.23
117	-0.0226	0.8678	2.3094	1.5197	0.8678	0.8363	0.0002	189.18
118	-0.0301	0.8699	2.3263	1.5252	0.8699	0.8284	0.0005	179.84
119	-0.0312	0.8685	2.3288	1.5260	0.8685	0.8229	0.0007	181.71
120	-0.0290	0.8675	2.3237	1.5244	0.8675	0.8314	0.0009	187.55
121	-0.0361	0.8705	2.3399	1.5297	0.8705	0.8322	0.0010	197.38
122	-0.0344	0.8686	2.3360	1.5284	0.8686	0.8871	0.0009	187.87
123	-0.0330	0.8641	2.3328	1.5273	0.8641	0.8289	0.0007	194.93
124	-0.0304	0.8638	2.3269	1.5254	0.8638	0.8310	0.0005	189.02
125	-0.0213	0.8577	2.3063	1.5187	0.8577	0.8212	0.0002	182.05
126	-0.0190	0.8587	2.3011	1.5169	0.8587	0.8231	0.0001	184.34
127	-0.0218	0.8576	2.3076	1.5191	0.8576	0.8226	0.0000	189.55

128	-0.0192	0.8569	2.3017	1.5171	0.8569	0.8210	0.0001	178.73
129	-0.0227	0.8588	2.3095	1.5197	0.8588	0.8633	0.0002	185.22
130	-0.0187	0.8590	2.3006	1.5168	0.8590	0.8869	0.0005	185.19
131	-0.0212	0.8585	2.3062	1.5186	0.8585	0.9210	0.0007	256.22
132	-0.0322	0.8623	2.3310	1.5268	0.8623	0.9049	0.0009	285.40
133	-0.0280	0.8625	2.3215	1.5237	0.8625	0.8652	0.0010	286.95
134	-0.0310	0.8611	2.3282	1.5259	0.8611	0.8223	0.0009	286.54
135	-0.0252	0.8580	2.3151	1.5215	0.8580	0.9392	0.0007	285.98
136	-0.0267	0.8563	2.3186	1.5227	0.8563	0.8248	0.0005	287.59
137	-0.0201	0.8538	2.3036	1.5177	0.8538	0.8493	0.0002	288.70
138	-0.0144	0.8486	2.2908	1.5135	0.8486	0.8237	0.0001	289.99
139	-0.0260	0.8519	2.3169	1.5221	0.8519	0.8197	0.0000	292.33
140	-0.0180	0.8510	2.2990	1.5163	0.8510	0.8273	0.0001	288.12
141	-0.0203	0.8524	2.3041	1.5179	0.8524	0.8166	0.0003	289.43
142	-0.0208	0.8528	2.3052	1.5183	0.8528	0.8511	0.0005	290.76
143	-0.0237	0.8536	2.3118	1.5205	0.8536	0.8231	0.0008	289.63
144	-0.0256	0.8543	2.3160	1.5218	0.8543	0.8319	0.0009	209.73
145	-0.0313	0.8550	2.3289	1.5261	0.8550	0.8477	0.0010	253.93
146	-0.0214	0.8530	2.3065	1.5187	0.8530	0.8193	0.0009	294.93
147	-0.0218	0.8511	2.3076	1.5191	0.8511	0.8520	0.0008	279.99
148	-0.0295	0.8517	2.3249	1.5248	0.8517	0.8207	0.0005	181.00
149	-0.0147	0.8445	2.2915	1.5138	0.8445	0.8166	0.0003	185.31
150	-0.0111	0.8435	2.2832	1.5110	0.8435	0.8227	0.0001	179.39
151	-0.0082	0.8424	2.2768	1.5089	0.8424	0.8195	0.0000	178.32
152	-0.0073	0.8425	2.2747	1.5082	0.8425	0.8171	0.0001	187.63
153	-0.0120	0.8442	2.2854	1.5117	0.8442	0.8199	0.0003	178.27
154	-0.0183	0.8476	2.2996	1.5165	0.8476	0.8529	0.0005	186.96
155	-0.0235	0.8499	2.3113	1.5203	0.8499	0.8424	0.0008	186.39
156	-0.0197	0.8474	2.3028	1.5175	0.8474	0.8936	0.0009	190.88
157	-0.0281	0.8504	2.3216	1.5237	0.8504	0.8531	0.0010	188.48
158	-0.0266	0.8491	2.3182	1.5226	0.8491	0.8195	0.0009	184.04
159	-0.0257	0.8461	2.3164	1.5220	0.8461	0.8178	0.0008	181.06
160	-0.0301	0.8466	2.3261	1.5252	0.8466	0.8476	0.0005	179.36
161	-0.0190	0.8430	2.3011	1.5169	0.8430	0.8143	0.0003	197.20
162	-0.0181	0.8414	2.2991	1.5163	0.8414	0.8248	0.0001	190.17
163	-0.0099	0.8383	2.2805	1.5101	0.8383	0.8177	0.0000	182.85
164	-0.0081	0.8377	2.2766	1.5088	0.8377	0.8164	0.0001	176.07
165	-0.0149	0.8415	2.2919	1.5139	0.8415	0.8149	0.0002	184.36
166	-0.0194	0.8421	2.3020	1.5172	0.8421	0.8128	0.0005	184.61
167	-0.0241	0.8440	2.3126	1.5207	0.8440	0.8784	0.0008	185.62
168	-0.0236	0.8467	2.3117	1.5204	0.8467	0.8238	0.0009	177.24
169	-0.0296	0.8462	2.3251	1.5248	0.8462	0.8152	0.0010	191.80
170	-0.0311	0.8468	2.3284	1.5259	0.8468	0.8515	0.0009	183.67

171	-0.0261	0.8424	2.3172	1.5222	0.8424	0.8833	0.0008	197.97
172	-0.0185	0.8391	2.3000	1.5166	0.8391	0.8158	0.0005	183.79
173	-0.0130	0.8370	2.2876	1.5125	0.8370	0.8271	0.0003	186.83
174	-0.0129	0.8354	2.2873	1.5124	0.8354	0.8141	0.0001	196.50
175	-0.0120	0.8339	2.2853	1.5117	0.8339	0.8154	0.0000	190.98
176	-0.0098	0.8344	2.2803	1.5101	0.8344	0.8132	0.0001	177.92
177	-0.0163	0.8357	2.2951	1.5150	0.8357	0.8112	0.0003	190.83
178	-0.0181	0.8385	2.2992	1.5163	0.8385	0.8172	0.0005	198.20
179	-0.0206	0.8400	2.3047	1.5181	0.8400	0.8856	0.0008	3310.31
180	-0.0185	0.8416	2.3000	1.5166	0.8416	0.8179	0.0009	193.92
181	-0.0264	0.8420	2.3178	1.5224	0.8420	0.8171	0.0010	185.02
182	-0.0261	0.8394	2.3172	1.5222	0.8394	0.8168	0.0009	194.52
183	-0.0234	0.8374	2.3110	1.5202	0.8374	0.8320	0.0008	271.13
184	-0.0197	0.8364	2.3028	1.5175	0.8364	0.8285	0.0005	285.48
185	-0.0132	0.8337	2.2881	1.5127	0.8337	0.8114	0.0003	285.41
186	-0.0030	0.8294	2.2649	1.5050	0.8294	0.8112	0.0001	284.08
187	-0.0039	0.8313	2.2670	1.5057	0.8313	0.8114	0.0000	284.40
188	-0.0006	0.8289	2.2596	1.5032	0.8289	0.8159	0.0001	279.98
189	-0.0068	0.8328	2.2736	1.5079	0.8328	0.8109	0.0002	279.11
190	-0.0111	0.8349	2.2833	1.5111	0.8349	0.8149	0.0005	281.08
191	-0.0179	0.8374	2.2986	1.5161	0.8374	0.8389	0.0007	283.50
192	-0.0184	0.8379	2.2998	1.5165	0.8379	0.8196	0.0009	278.01
193	-0.0202	0.8386	2.3038	1.5178	0.8386	0.8200	0.0010	281.11
194	-0.0293	0.8385	2.3244	1.5246	0.8385	0.8333	0.0009	279.62
195	-0.0278	0.8370	2.3210	1.5235	0.8370	0.8122	0.0007	275.42
196	-0.0181	0.8334	2.2990	1.5163	0.8334	0.8375	0.0005	193.86
197	-0.0052	0.8294	2.2700	1.5066	0.8294	0.8312	0.0002	269.57
198	-0.0003	0.8269	2.2590	1.5030	0.8269	0.8125	0.0001	282.19
199	0.0054	0.8254	2.2461	1.4987	0.8254	0.8138	0.0000	284.74
200	0.0017	0.8254	2.2544	1.5015	0.8254	0.8108	0.0001	279.58
Mean Absolute Error	0.81							
Mean Squared Error	2.27							
Root Mean Squared Error	1.51							
Coefficient of Determination	-0.02							

Table A.11.: Fifth training results of EEGNet for Regression

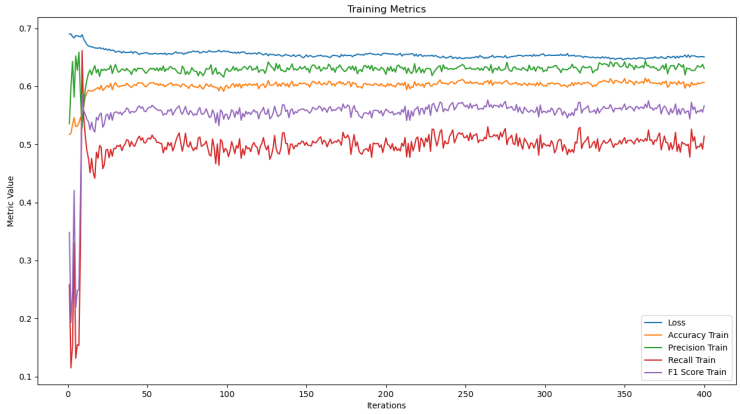
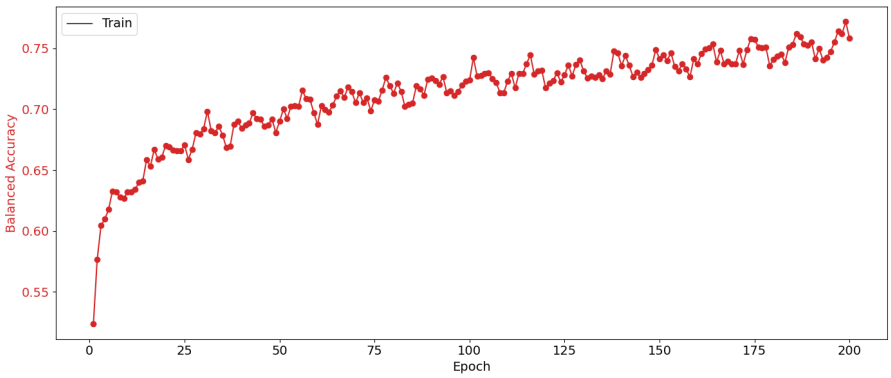


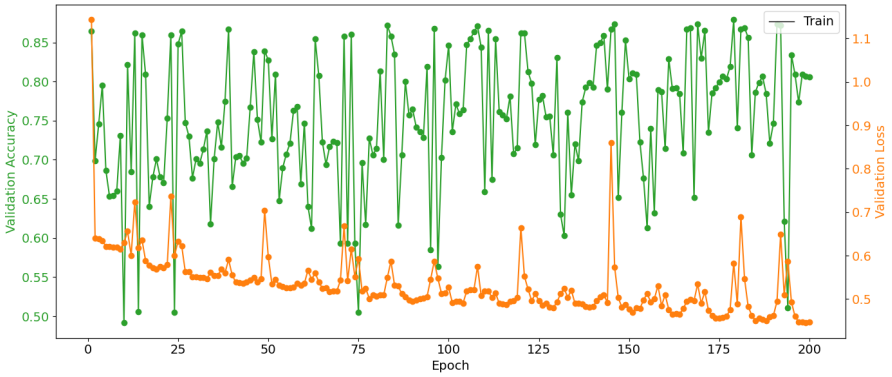
Figure A.2.: EEGNet classification 400 epochs training

For the classification task a training run over 400 epochs was executed to compare the convergence patterns of the EEGNetv4 when it comes to a longer training horizon. Here the metrics are shown in their corresponding color. The figure suggests that the model does not improve a lot in terms of loss reduction or accuracy improvement after epoch 60. Furthermore the fluctuations in the metrics become visible suggesting that a longer horizon of training for the base configuration does not yield better results.



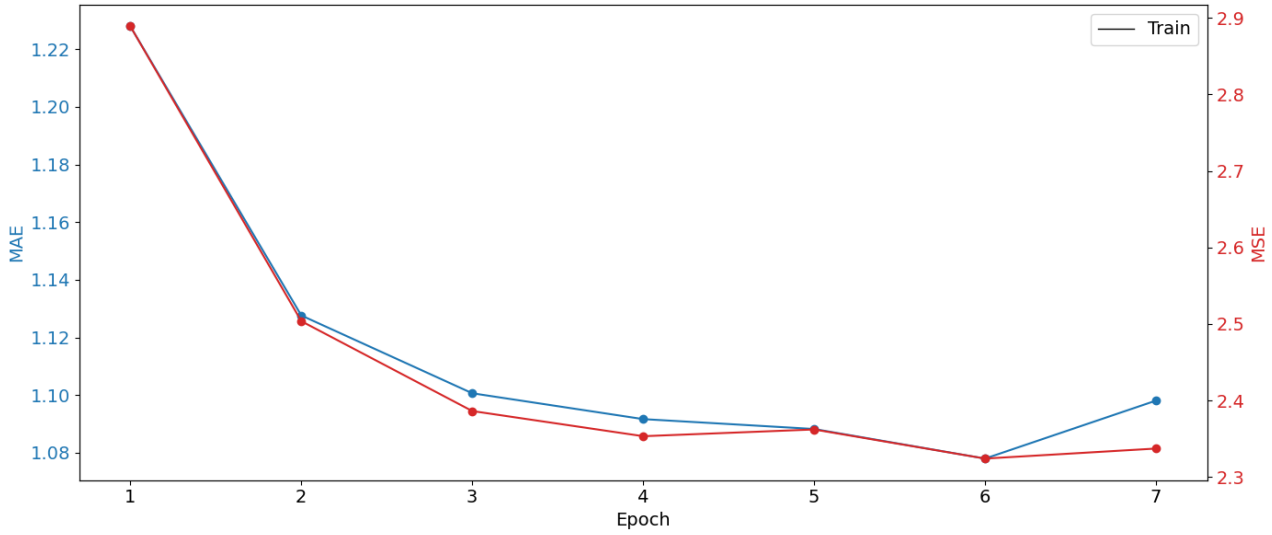
The figure shows the balanced accuracy over the course of a 200 epochs training run from the EEGNet.

Figure A.3.: balanced accuracy EEGNet classification 200 epochs training

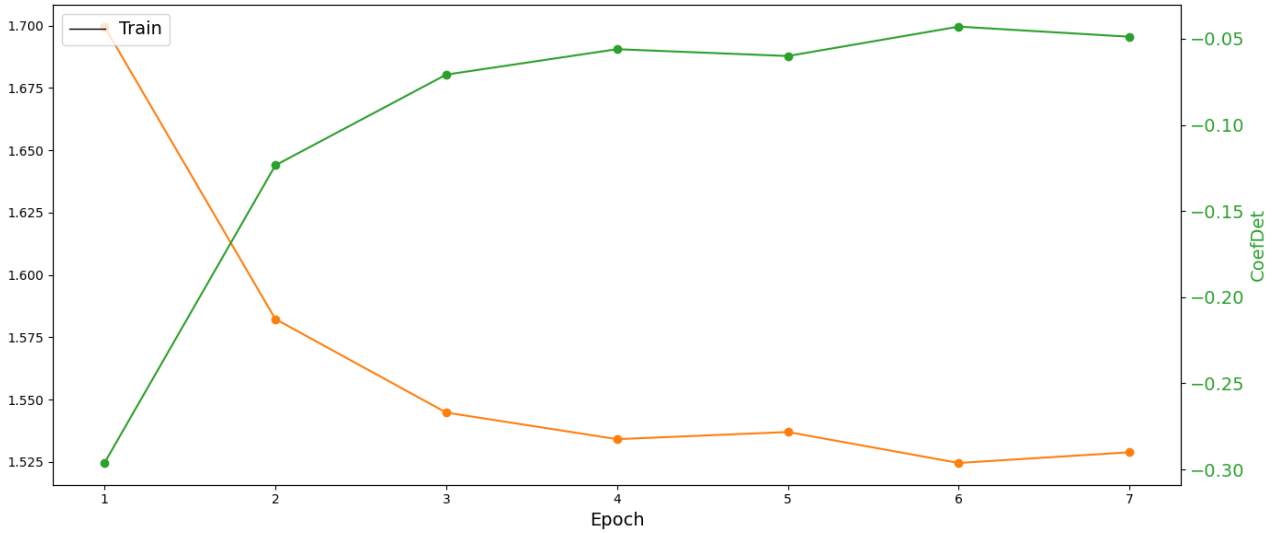


The figure shows the validation loss and accuracy over the course of a 200 epochs training run from the EEGNet.

Figure A.4.: Validation metrics EEGNet classification 200 epochs training

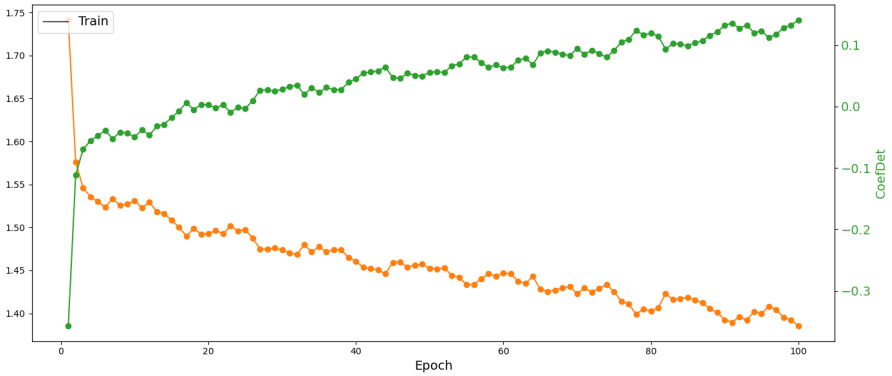


Evolution of MSE and MAE over the course of a 7 epoch training run for the EEGNet regressor.



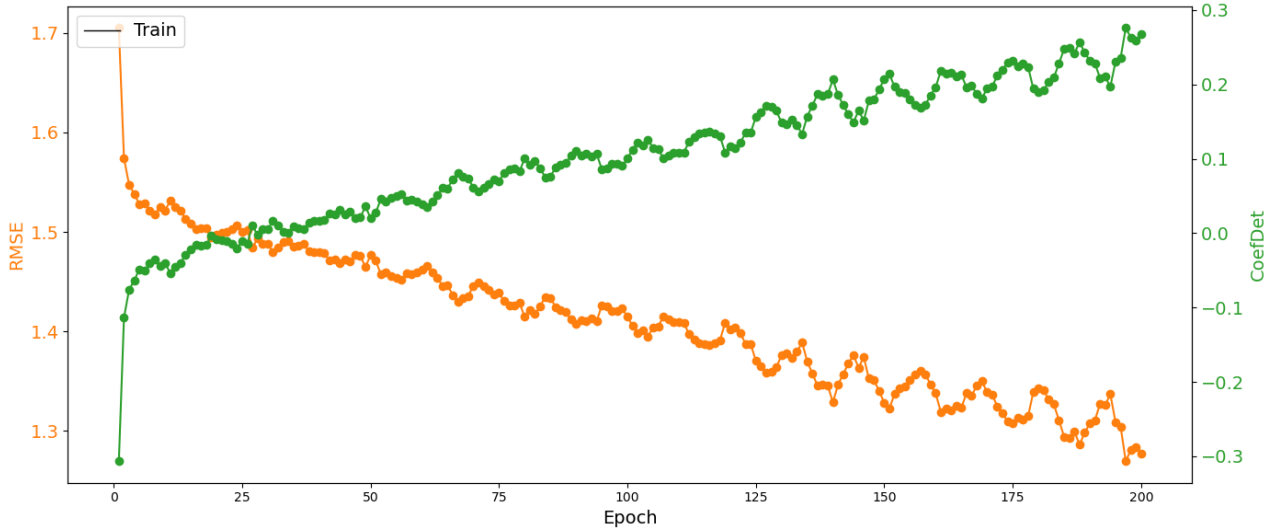
Evolution of Coeffiecient of Determination and RMSE over the course of a 7 epoch training run for the EEGNet regressor.

Figure A.5.: EEGNet Regression 7 epoch training metrics

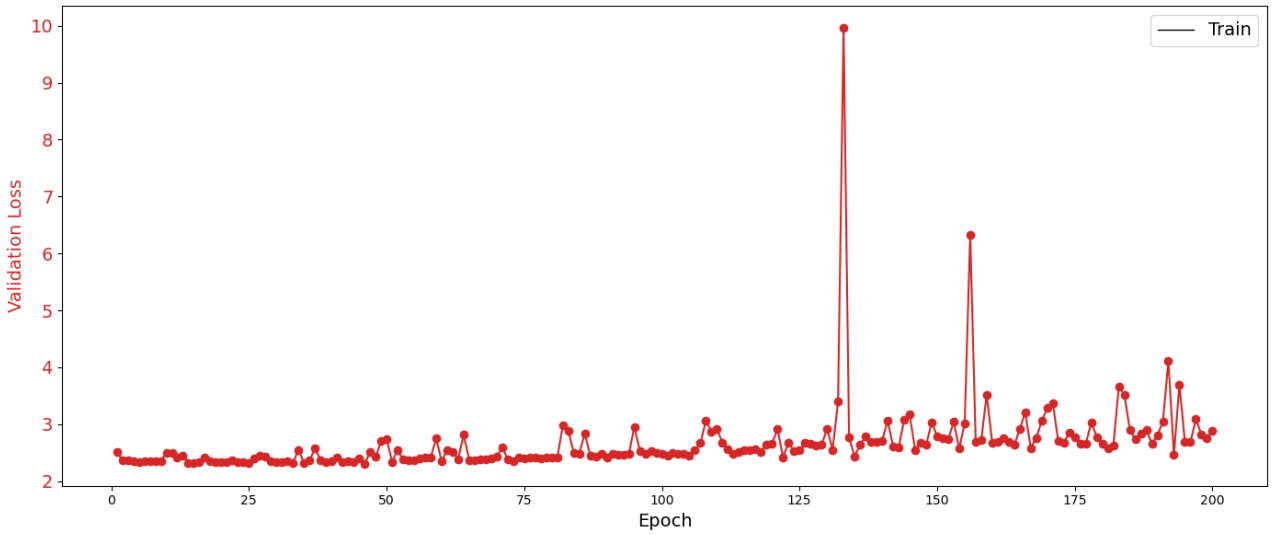


The figure shows the progression of the metrics over the course of a 100 epoch training run for the EEGNet regressor. It includes the evolution of the coefficient of determination and the RMSE.

Figure A.6.: EEGNet Regression 100 epochs metrics

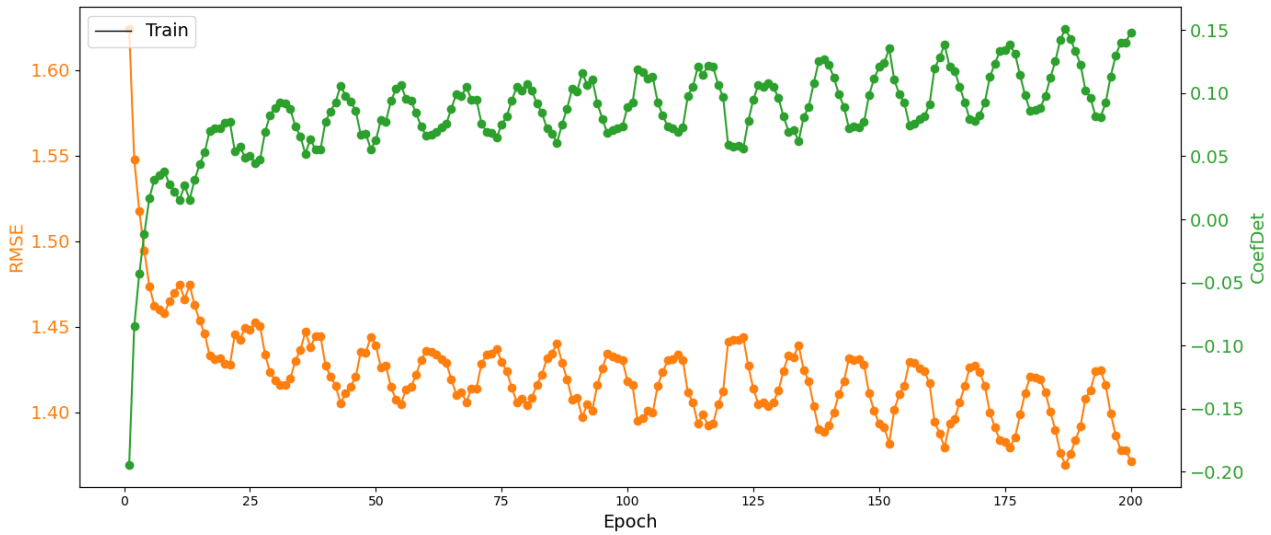


The graph shows the evolution of the metrics RMSE and the coefficient of determination over the course of a 200 epoch training.

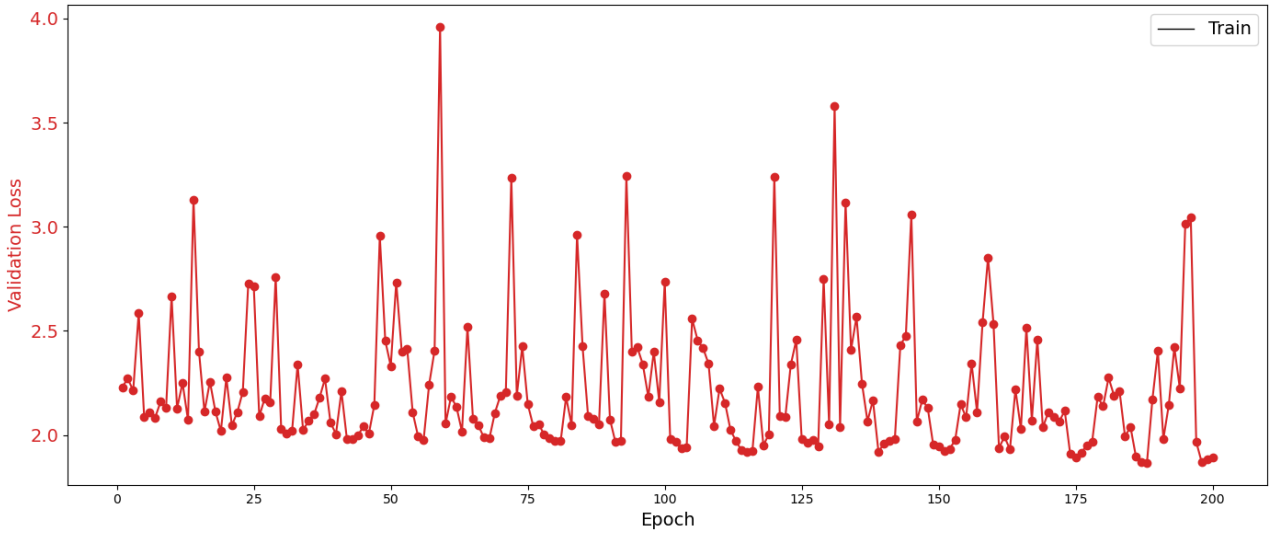


The graph shows the evolution of the validation loss over the course of a 200 epoch training.

Figure A.7.: Metrics 2nd training run EEGNet regression

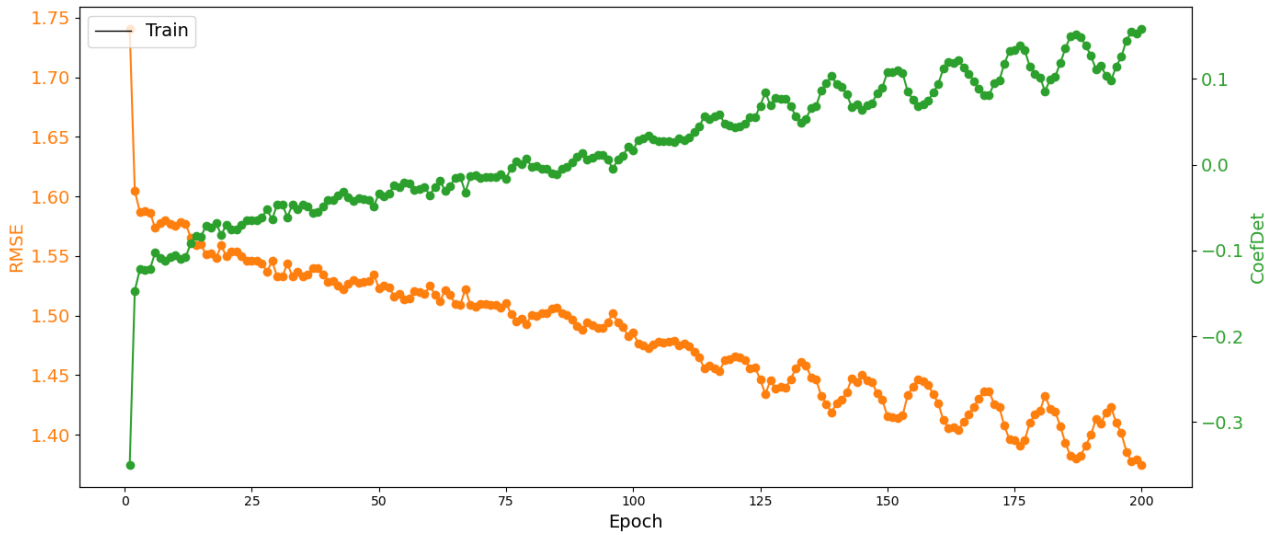


The graph shows the evolution of the metrics RMSE and the coefficient of determination over the course of a 200 epoch training. Hereby the LR was adjusted to 0.01 instead of 0.001.

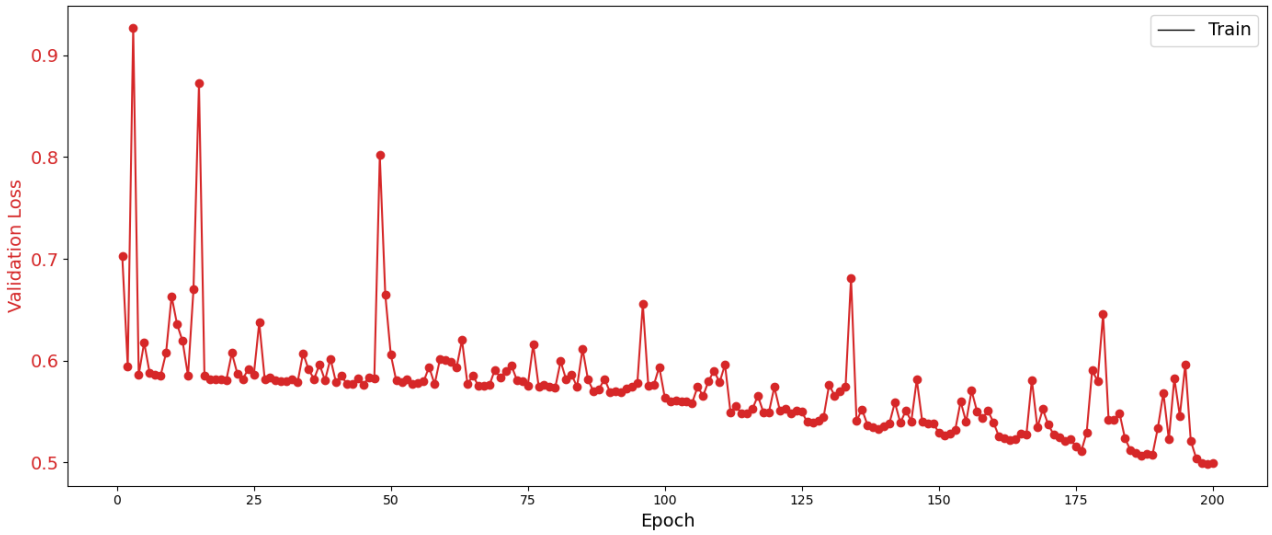


The graph shows the evolution of the validation loss over the course of a 200 epoch training with an adjusted LR 0.01.

Figure A.8.: Metrics 3rd training run EEGNet regression

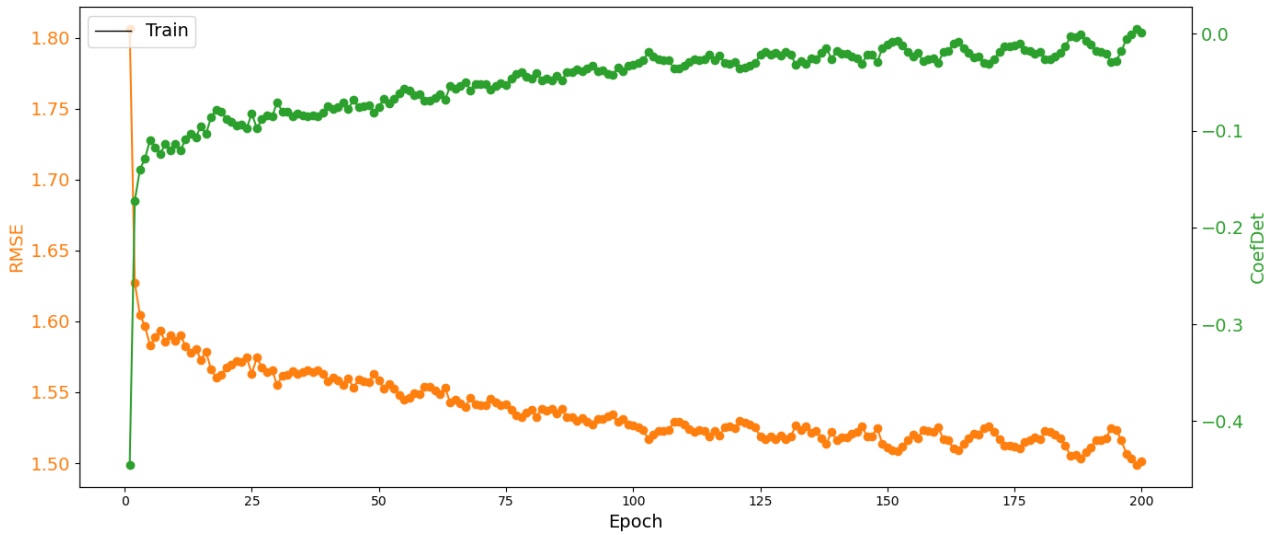


The graph shows the evolution of the metrics RMSE and the coefficient of determination over the course of a 200 epoch training. Hereby the Huber Loss was used instead of the previously incorporated MSE Loss.

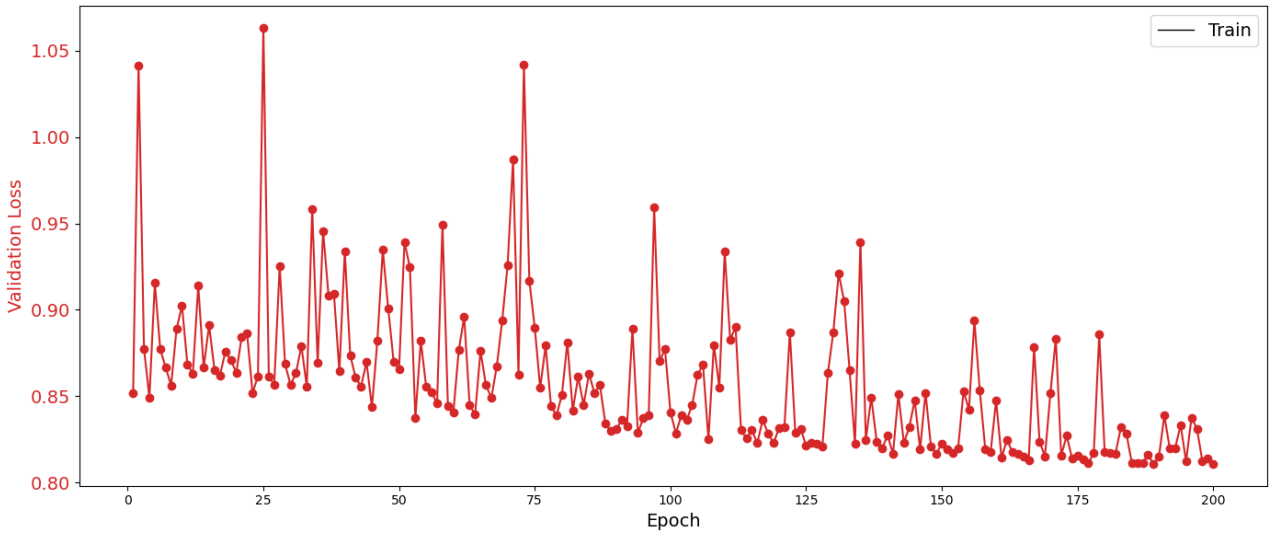


The graph shows the evolution of the validation loss over the course of a 200 epoch training with the loss function being changed to the Huber Loss.

Figure A.9.: Metrics 4th training run EEGNet regression



The graph shows the evolution of the metrics RMSE and the coefficient of determination over the course of a 200 epoch training. Hereby the L1 Loss was used instead of the previously incorporated MSE Loss



The graph shows the evolution of the validation loss over the course of a 200 epoch training with the loss function being changed to the L1 Loss.

Figure A.10.: Metrics 5th training run EEGNet regression