Bachelor Thesis

Technische Hochschule Ulm

Informatik

Deep Learning based Motor-Imagery Brain-Computer Interface

Author:

Philipp Kessler

Matriculation number

[redacted]

May 11, 2021

1. Advisor:

[redacted]

2. Advisor:

[redacted]

Declaration of Authorship

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Abstract

EEG-based technologies allow to record brain activity, which leads to the idea of a Brain Computer Interface. To control a machine by simply thinking what you want the system to do sounds like science-fiction but might become reality soon enough. In this thesis the possibility of such a BCI is validated. Since classifying brain activity of a human is not a trivial task, Machine Learning is heavily used in the proposed system. For that purpose, this thesis elaborates on a framework to test Artificial Neural Networks, more specifically the chosen EEGNet [11] model, for the use-case of a BCI. It is able to execute subject global Training and Testing to validate a given model with a known dataset and evaluate its usage for a BCI. As a basis, the Physionet Motorimagery Dataset [16] is used for Training and Testing. The framework additionally allows to investigate Transfer Learning in the context of a Live Simulation mode, which means that a pretrained model can be further trained using subject specific data. Afterwards this network can be utilized to classify on the used subject's live incoming data in realtime. To be able to fully test and validate the capabilities of the model and the used data, the framework is highly configurable and easy to extend with new models and datasets.

Contents

List of Abbreviations

ANN Artificial Neural Networks

BCI Brain Computer Interface

CNN Convolutional Neural Network

CUDA Compute Unified Device Architecture

EDF+ European Data Format plus

EEG Electroencephalogram

ELU Exponential Linear Unit

SDK Software Development Kit

SoC System on a Chip

List of Figures

List of Tables

1. Introduction

1.1. Motivation

In modern times, most machines such as robots, heavy machinery, televisions, video games are always controlled with a type of remote. May it be hardware terminals with buttons, switches, joysticks, hand-held controllers or touchscreens. These sorts of input devices all need an additional user interface, they need to be intuitive, easy to control for anybody, but for the user to know how exactly to translate his desired actions to the controlling device, this can become frustrating the more complicated and bigger the amounts of controls are.

This is where the so called Electroencephalogram(EEG) comes in. It provides a way to record human brain-activity with easy to apply electrodes on the subject's scalp. If it were possible to connect this device to a machine like a computer and then classify what the subject is thinking, the problem of translating the users desired action to the actual execution of that task on the machine could work without any additional interface. The user can just think whatever he wants the machine to do and the command will be recognized by the machine directly. This direct interface is called a Brain Computer Interface(BCI). This may sound very futuristic because classifying brain activity is neither trivial nor yet well researched. But since the rise of Machine Learning, more and more very complex problems can be tackled by these new approaches without the developer having to understand the entire nature of the data at hand.
So this thesis tries to develop a concept based on Machine Learning algorithms to find out whether or not it is possible to build a reliable and cost-efficient BCI. This thesis is not about developing a new Artificial Neural Networks, but rather trying to wrap an existing model, called *EEGNet* in a testing environment to validate its feasibility and usability for a BCI.

1.2. Task

The proposed task to be solved in this work, can be split into different questions. The overall goal of this work is to propose and implement a first approach to a BCI. As a basis, the pre-existing Convolutional Neural Network called *EEGNet* is used. It has to be validated if this model is well suited for the needs of a BCI, fed with EEG data. For that purpose, a framework has to be built in order to test the chosen model with a highly configurable testing environment, providing a way to easily configure all important parameters. As a data source, the Physionet Motorimagery Dataset (see 2.4) should be used, which also makes it possible to compare the shown results with existing works based on the EEGNet. The implementation should provide ways of effortlessly extending it with other datasets or models.

2. Fundamentals

2.1. Machine Learning

With modern technologies increasingly more abstract problems can be solved by machines. Usually such a problem is defined by a certain input of data, algorithms to compute this input and the desired result as an output. This is a good approach for many trivial problems and scenarios, but with modern technologies more complex problems arise that can not be solved by this traditional approach. This is where Machine Learning comes into play.
Instead of defining algorithms that lead to a certain result with a given input, so called Artificial Neural Networks(ANN) are defined. These networks are supposed to mimic the functionality of a human brain by defining a digital version of the neural network in order to learn and abstract from the given data.

2.1.1. Structure of Neural Networks

The smallest unit inside of an ANN is called a **Perceptron**. It is a mathematical representation of a *Neuron*, which the human brain consists of. A Perceptron takes input values, applies so called **Weights** and **Biases** and then adds the results in a Summation function which gets fed into a mathematical *Activation Function*. The outputs are fed into other Perceptrons' inputs.
Multiple Perceptrons make up a *Neural Network Layer* of the ANN. The layers are interconnected and build the inner or hidden structure of the network. Now for the network to actually be able to learn something from its inputs, it uses **Backpropagation**. For that, at first some inputs are fed into the network, they get forwarded through all the layers with their respective Perceptrons and in the

end we get a final output of the last layer. This output represents the result of the network for the given input. Now the network has to know how good its results are, so for that we calculate a **Loss**. To do that, we need a desired result to compare the given result to. There are different mathematical functions available to determine the actual Loss value. Most commonly the *Mean-Squared Error* is used.

The Loss is now fed back into the network and the Weights and Biases of the Perceptrons inside the hidden layers are updated accordingly, to minimize the Loss the next time an input is forwarded. The entire goal of feeding pre-selected data into the model is to find the global minimum of the Loss function. To achieve this, the **Gradient descent** optimization is applied. This is also where the so called *Learning Rate* comes in. It is a value that controls how large a step to take during minimization of the Loss is. The greater its value the bigger the step to find a better point in the Loss function. [35]

Fully Connected Layer

A so called Fully Connected layer contains neurons with **every neuron being connected to every single neuron of the next layer** (see Fig. 2.1). This kind of layer is used in nearly every ANN. As you can imagine, the higher the neuron count the higher the computational cost of such a layer becomes. Usually for classification models, these layers are placed towards the end of the network which output the various resulting class predictions.

Convolutional layer

The so called *convolution* layers are the basis for Convolutional Neural Network(CNN). They are mostly used in connection with multi-dimensional data, where specfic patterns are supposed to be detected and learned from. A convolutional layer performs a **convolution**. This is a linear operation which entails the inputs and a set of weights, which are called **kernels** or **filters**. The **scalar product** of patches of the inputs and the filters are calculated. The result is a single value for each of the input patches, which is called **feature map** (see Fig. 2.2). It is supposed to represent a sort of feature detection, finding certain important patterns in the data, which then get fed through an activation function to be connected to the next layers. [24]

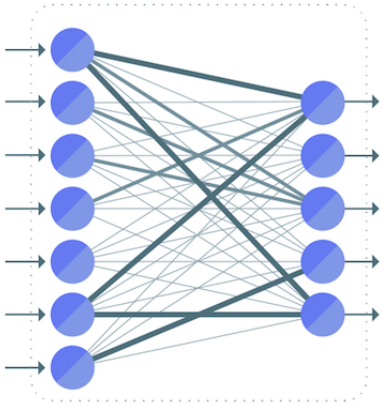

Figure 2.1.: [23, *"Fully Connected Layer"*]

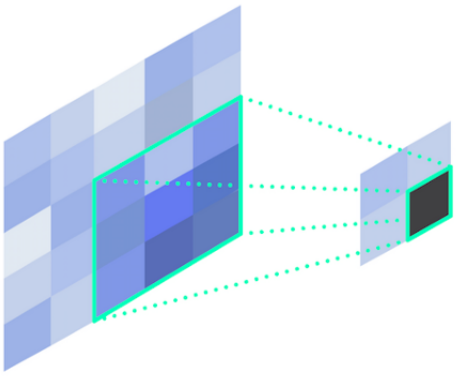

Figure 2.2.: [23, *"Convolutional Layer"*]

2.1.2. Supervised Learning

For typical classification tasks like image classification or also the here wanted classification of EEG data, *Supervised Learning* is the preferred Learning procedure. In this context, supervised means that during Training, the model gets **feedback on how good it classifies** on the given dataset. Therefore we need **labeled data**, which means the dataset does not only contain the raw data but also labels which specify the associated classes for the individual samples. Using such labeled data, the model can infer on an input from the known dataset and compare its outputs to the desired results in the labeled data. The difference between the actual labels and the predicted labels build the loss, which gets fed back into the model as described in the above Section (2.1.1). So metaphorically the model gets a supervisor who watches the Training and tells the model how good its predictions are.

2.1.3. Performance Metrics

The following contains a short summary of what important metrics for Machine Learning models exist and which are used in this work. For more detailed information on these and other metrics, you can refer to Chapter 5.6 in the book *"The Hundred-Page Machine Learning Book"* [3].

Accuracy

The most used metric for how good the Training process worked, is the so called accuracy. "Accuracy is given by the number of correctly classified examples divided by the total number of classified examples" [3, chap. 5.6.3]. It therefore describes how good the trained model is at classifying given samples. For classification tasks like this, the most important part is that the system reliably predicts the correct classes.

Confusion Matrix

The confusion matrix gives an overview of the performance on classifying on given samples of different classes. It is a table that contains the number of correctly and falsely classified samples for every single class.
"The confusion matrix for multiclass classification has as many rows and columns as there are different classes" [.] The individual cells of every row (actual class) and every column (predicted class) are either ones of the following:

- True Positive (**TP**): It refers to the number of predictions where the classifier correctly predicts the positive class as positive.
- True Negative (**TN**): It refers to the number of predictions where the classifier correctly predicts the negative class as negative.
- False Positive (**FP**): It refers to the number of predictions where the classifier incorrectly predicts the negative class as positive.
- False Negative (**FN**): It refers to the number of predictions where the classifier incorrectly predicts the positive class as negative.

(Citation of towardsdatascience [28])

Precision/Recall

Other popular metrics are *precision* and *recall*.
Precision is the ratio of correct positive predictions to the overall number of positive predictions:

$$precision = \frac{TP}{TP + FN}$$

Recall is the ratio of correct positive predictions to the overall number of positive samples in the dataset:

$$recall = \frac{TP}{TP + FN}$$

[3, chap 5.6.2, p.15]

2.1.4. Convolutional Neural Network EEGNet

For the classification task at hand, there are already several CNN models from other scientific works. Since the system is supposed to be run on a low power embedded system and used for a BCI, the model called *EEGNet* was chosen. This model is very simple in its inner structure and can therefore be run on low-spec hardware systems without any problems. Its core idea is to combine multiple convolutional layers to extract features and patterns specific to the nature of EEG data. A basic overview of the layers of the model can be seen in Figure 2.3.

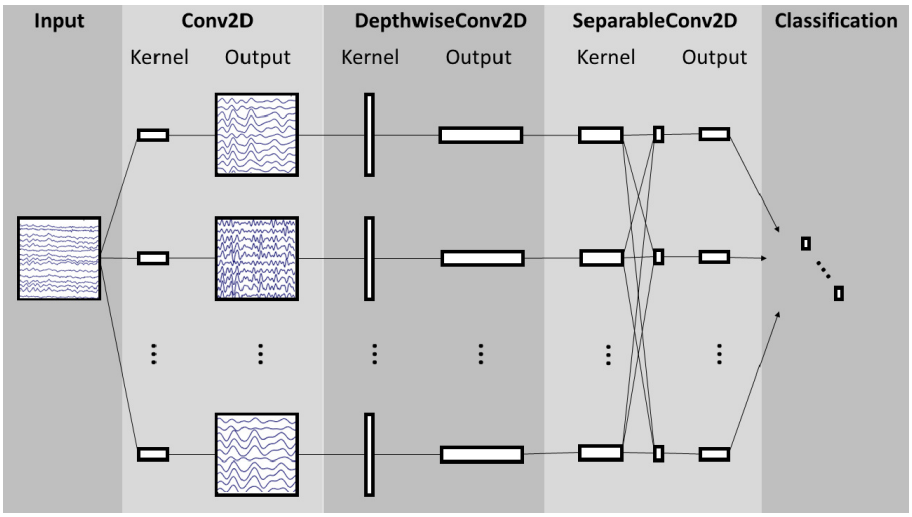

Figure 2.3.: EEGNet Structure [11, Fig.1]

"The network starts with a **temporal convolution** (second column) to learn frequency filters, then uses a **depthwise convolution** (middle column), connected to each feature map individually, to learn frequency-specific spatial filters. The **separable convolution** (fourth column) is a combination of a depthwise convolution, which learns a temporal summary for each feature map individually, followed by a pointwise convolution, which learns how to optimally mix the feature maps together" [11, Fig.1 caption].
For a more in-depth look at how the EEGNet model works, refer to the original proposal in *"EEGNet: a compact convolutional neural network for EEG-based brain-computer interfaces"* [11].

2.2. Electroencephalogram (EEG)

An EEG is a passive, non-invasive way of recording **brain activity**. It entails detecting electrical activity of a human brain with the use of small electrodes that are attached to the subject's scalp. Human brain cells emit electrical pulses that correspond to certain actions or even just imaginations. Every electrode provides a recording of electrical activity of a certain part of the subject's brain which can be interpreted for different uses.

The recordings contain electrical voltage measurements for each electrode per channel. Usually the values are denoted in micro voltage(µV). An example EEG can be seen in Figure 2.4.

In medicine its main use case is diagnosing cognitive disorders like epilepsy or seizure disorders in general but might also be useful for diagnosing strokes or even brain tumors [36]. Another field of use is the concept of a **Brain Computer Interface**(**BCI**). It proposes a system where the user can send commands to the computer system via brain activity, especially **sensorimotor** commands from the Motor/Sensory Cortex, shown in Figure 2.5. [22]

10-20/10-10 System

To be able to precisely define the **locations of the electrodes** on a subject's scalp, the so called *10-20 System* was introduced in 1999 and is the international standard for EEG electrode placement [10]. The *10* and *20* in the name denote the distances between two neighboring electrodes in percent of the total front to back or right to left distance of the scalp.
A variation of this system called *10-10 System* is used in the Physionet Dataset (see 2.4), that is used in this work. There the right-left distance of the electrodes is only 10 percent instead of 20. This standard also provides a labeling method for the positions based on the general areas of the brain as well as how far left or right the electrode is placed. This schematic is visualized in Figure 2.6.

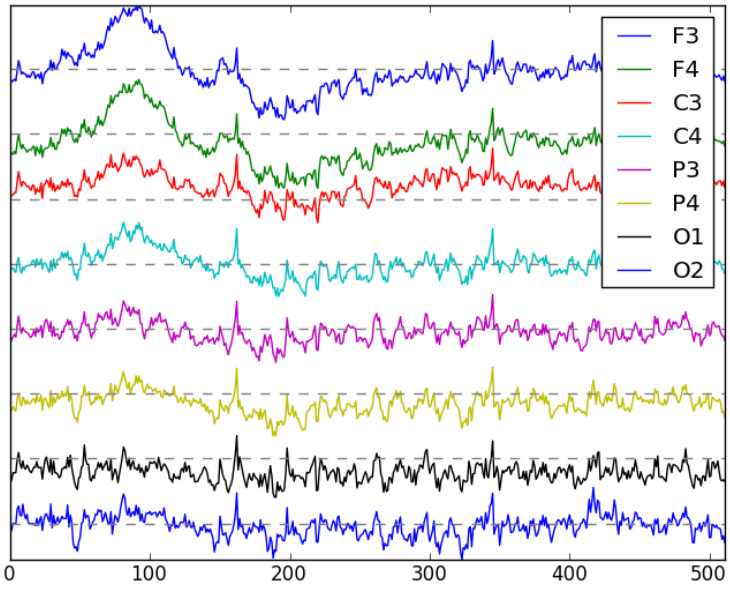

Figure 2.4.: EEG example recording with 8 EEG Channels [37]

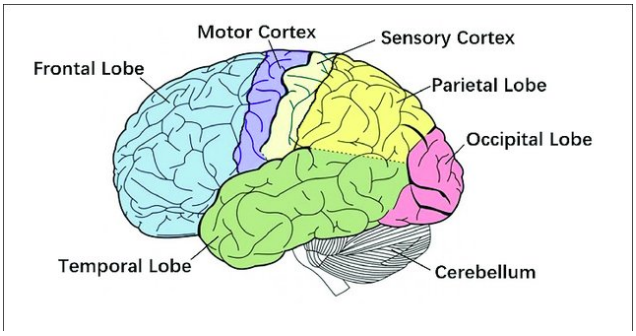

Figure 2.5.: "Primary brain regions" [12, Fig.2]

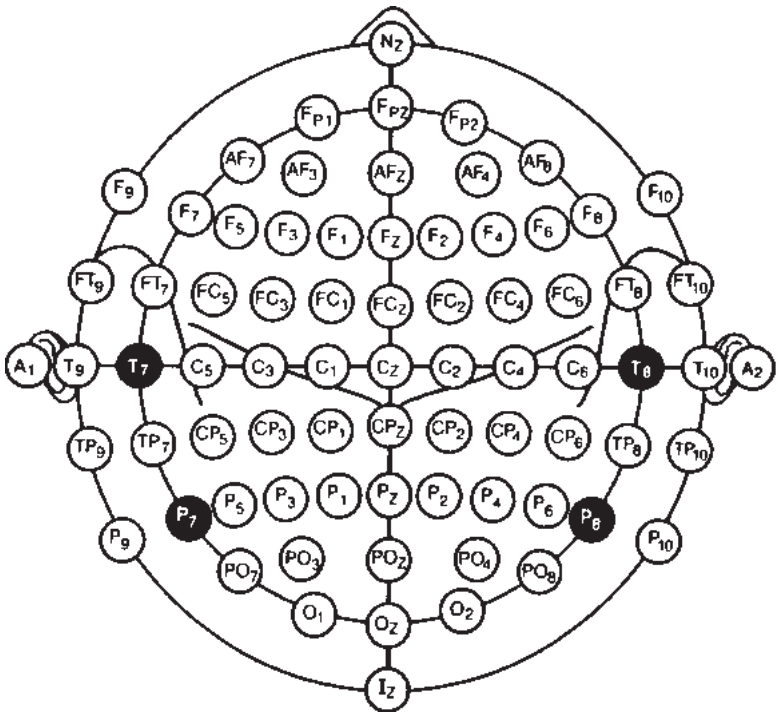

Figure 2.6.: International 10-20 System for EEG Electrode placement [10, Fig.7]

2.3. NVIDIA Jetson Nano Platform

As an **Embedded Platform** the NVIDIA Jetson Nano is used. It is based on low power SoC developed by NVIDIA [30] designed specifically for Machine Learning applications like classification tasks. It only draws about 5-10 Watts at maximum utilization. The integrated 128-core Maxwell **CUDA** [29] Cores can be used to optimize performance by utilizing parallel execution which is specifically useful in Machine Learning. As an operating system, *Ubuntu 18.04 LTS* is installed on an external SD-Card.

Table 2.1.: NVIDIA Jetson Nano Technical Specifications [30]

CPU	Quad-core ARM A57 @ 1.43 GHz
GPU	128-core Maxwell
Memory	4 GB 64-bit LPDDR4
Storage	External SanDisk Extreme Pro 64GB

2.4. PhysioNet Motor Imagery Dataset

The dataset that is used in this thesis originates from *PhysioNet* [5] and provides recordings from an experiment at the *New York State Department of Health* [16].

It includes EEG recordings using 64 electrodes placed according to the 10-10 system (see 2.2) on **109 different human subjects** (see Figure 2.7). Each subject had to perform 4 different Tasks of **motor imagery actions**. Every **Task** includes 2 motions like for example opening/closing left or right fist, as well as recordings of the subject just resting (*Rest*). For each Task the subject completed 3 two-minute **Runs** with the motor Tasks actually executed as well as 3 Runs with the same Tasks but only imagined executions.

For the proposed system it is important to be able to determine prediction accuracies for a given amount of motorimagery event types that should be distinguishable. Each of these event types is called a *class*, with the number of available classes denoted by n. Therefore, each task of the Physionet Dataset contains 3 classes, 2 motorimagery ones and the rest class.
A single occurrence of a class event is called **Trial**. Every motorimagery Trial is followed by a Rest Trial. A more detailed view of the structure of the Physionet Motorimagery Dataset can be seen in Figure 2.8. The there mentioned signals `T0`, `T1` and `T2` are so called control channels which have been synchronously recorded with the EEG data and which indicate the points in time, when certain actions should be executed by a subject. E.g. `T0` indicates the time slices where a subject has been instructed to rest. A classification with n classes is named **n-class classification**. In this work, a closer look at the 2, 3 and 4-class classification performance of the system is taken, with the classes chosen as follows.
All rest trials are omitted from the actual runs and instead are simply picked from the so called Baseline run. Every subject in the dataset had to complete this run, which contains a 1 minute recording with nothing but Rest Trials. These and only these Trials are used for the following classifications as Rest Trials.
Task 2 of the motorimagery dataset is used, which includes 2 classes, imagining opening/closing the left(L) or the right(R) fist. For this task, runs 4, 8 and 12 are available and in total provide 6 minutes of EEG recordings per subject. Therefore every Task includes 3 different event classes, the Rest class and 2 motorimagery classes. The 2

class classification uses the same Task and Runs as the 3 class classification but omits the Rest class events and therefore only classifies the remaining 2 motorimagery classes. To get another class for *n=4*, the open/close both-feet(*D*) class of Task 4 is also added to the above described 3 classes. All remaining Trials of Task 4 are omitted.

A sampling rate of 160Hz was used, which ensures that both the mu and the beta frequency bands with 8-12Hz and 18-25Hz are properly recorded. According to *"A wavelet-based time-frequency analysis approach [...]"* [13], these frequencies are of most importance when trying to classify sensorimotor events.

The dataset is supplied in the European Data Format plus (EDF+) format which contains the samples from all 64 used EEG channels and additional metadata to define which samples belong to which Trial as well as the labels for the corresponding events.

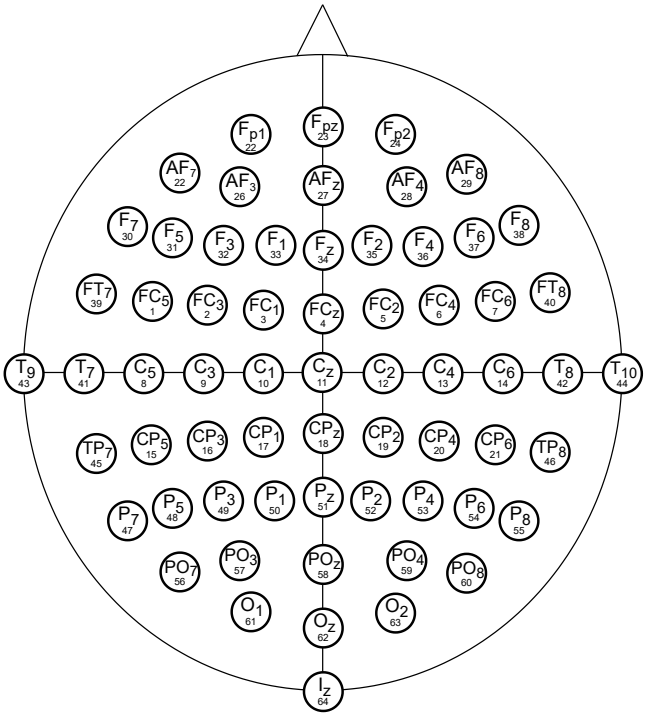

Figure 2.7.: Available EEG Channels of Physionet Dataset (10-10-system)

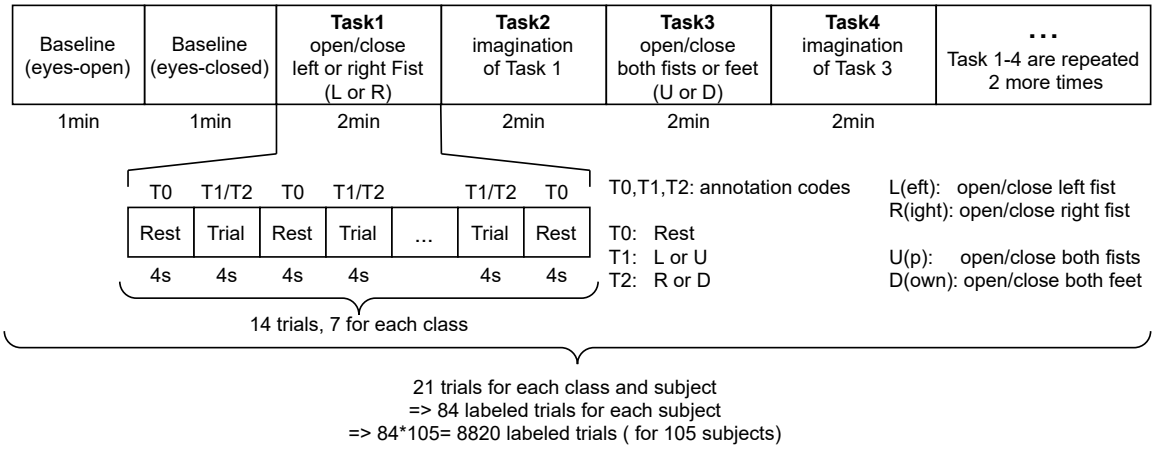

Figure 2.8.: Physionet Motorimagery Dataset Structure

3. ML-BCI1.0: Framework to investigate Machine-Learning based BCI Systems

3.1. Requirements

Since the proposed system is supposed to be a fully usable BCI application, among others in a scenario where the EEG data is measured by actual hardware that has to be acquired as well as a specific hardware platform on which the system is run, there are several important requirements the system has to comply with. On one hand there are typical Machine Learning phenomenons that can occur, which have to be mitigated. On the other hand, analyzing EEG data also leads to some questions like if or how the data has to be preprocessed or also where the electrodes are supposed to be placed as well as if the data can be abstracted from to be subject independent. Table 3.1 gives a quick overview of the present requirements.

Implementation

In the initial task one of the requirements was to use the **PyTorch** Framework [34] which is written for the Python language. It offers the ability to define Neural Network models and implementations for the Training and Testing Procedure of Supervised Learning. Python also includes easy to use methods to split the available dataset into Training and Test sets as well as Batch loading. Since many neural network models, like the **EEGNet**, are originally implemented **in Tensorflow** [1]/Keras [25], all used models have to be translated into PyTorch.

High Configurability

As described in 2.1, there are a lot of hyperparameters to tweak and optimize the Training process as well as the hidden layers in a CNN. Therefore it is paramount that the implementation provides ways to easily test different parameter combinations and determine metrics to compare different settings. The resulting software is supposed to be usable as a **testing framework** for the EEGNet model, while staying variable enough to also add and test other models or other datasets.

Global Accuracy

"Accuracy is a useful metric when errors in predicting all classes are equally important" [3, chap. 5.6.3]. Since this is exactly the case for the here described application, accuracy has been chosen as the **most paramount metric** to validate the proposal. It is important that the system delivers a high global classification accuracy. Here, global means that the accuracy performance is not subject specific but about the same for each individual subject.
To have a measurement of how good the system performance is in terms of accuracy, the Physionet Dataset is used to validate the proposed approach, since it provides EEG data of a large number of different subjects. For each subject an accuracy can be determined by comparing the events' predictions of the system with the labels of the prepared dataset. The absolute minimum in terms of accuracy for a system like this are the **chance levels**. These are defined for n = amount of classes as:

$$cl(n) = \frac{1}{n}$$

which represents an accuracy of the system just plain guessing which class the present Trial belongs to. If the system is below or near this accuracy the Training Process did not work as desired.

Performance Benchmarking

For the system to work with live EEG measurements, the hardware platform has to provide high enough technical specifications to ensure **usability in real time**. Since

there is a large amount of EEG data provided every second, the system has to have enough memory and computing power to load, preprocess and classify the amount of generated data. On top of that, it has to provide enough memory to store all the necessary weights and biases of the used CNN model. Therefore it is necessary to also implement a performance benchmarking to validate if the system is fast enough for live usage.

Comparison to existing work

To ensure that the implementation works correctly and delivers **reasonable results**, it is necessary to compare the shown results with **existing works** that also use the EEGNet. For this purpose, much of the Machine Learning processes and parameters used in this work are aligned with the paper *"An Accurate EEGNet-based Motor-Imagery Brain-Computer Interface for Low-Power Edge Computing"* [21], which also proposes a system to use the EEGNet for a motorimagery BCI.

Subject specific Transfer Learning

To get a high classification accuracy for a completely new subject, it is often a good idea to use **Transfer Learning** to specialize the pretrained model to possible special features of the new subject's brain activity. Therefore, it has to be possible to load a model pretrained with the subjects from the chosen dataset and then execute Transfer Learning with some new Test data of the introduced subject. This will often lead to better accuracy on this individual subject without much effort.

Live Usage

The system also has to support predictions in a real time mode where the EEG data is fed into the system in a certain continuous **sliding time window** and not only static analysis of past recordings. This differs from the usual Supervised Learning Training process and has to be implemented separately. This mode is further explained in Section 3.3.8

EEG Channel Selection

Another important aspect is the financial cost of acquiring the necessary hardware as well as maintenance costs. Since EEG measuring Hardware, like the *OpenBCI Starter Kit* [32], cost hundreds, even up to thousands of Euros, it is important to take a closer look at which EEG channels are actually important for the classification tasks at hand. This thesis solely focuses on the classification of motorimagery events, so the **amount of EEG channels and their placement** on the subject's brain is impacting the performance as well as the initial financial requirements. As mentioned in Section 2.2, sensorimotor actions are emitted from the so called motor and sensory cortexes which are located on the top center of the brain and extend to both sides (see 2.5). Therefore, in theory, only the EEG channels placed in these regions should be sufficient to classify sensorimotor events of a human brain. Existing works, such as *"Multi-objective genetic algorithm as channel selection for P300 and motor imagery dataset"* [8] show that a reduction in the amount of used channels does not necessarily lead to a high loss in accuracy for a BCI-like classification scenario.

Table 3.1.: Requirements

No.	Requirement
1	Implementation in Python with PyTorch
2	High Configurability
3	High global accuracy
4	Performance Benchmarking
5	Comparison to existing work
6	Subject-specific transfer learning
7	Mode for realtime live usage
8	Minimizing number of needed EEG channels

3.2. Potential Problems

Quality of the Dataset

To achieve the highest possible accuracy on the trained model, it is paramount that the dataset used for the Supervised Learning Process complies to certain properties.

Most importantly since EEG measurements can be very individual for each subject the dataset has to include many different subjects. Otherwise it is not possible for the CNN to learn data specific features or patterns.
Furthermore, there has to be an equal amount of Trials per motorimagery event type to ensure equally distributed accuracies across the different classes, otherwise the CNN could develop a bias towards certain classes with more Trials and therefore impact the average accuracy across all classes.

Over-/Underfitting

A common issue with Supervised Learning is Over-/Underfitting.
On one hand, **Overfitting** is the phenomenon that can occur when the Training process becomes too focused on **minimizing the loss on the Training data**. Usually the goal of Supervised Learning is to train a model with existing data and then get a low loss on completely new data which includes the same patterns or notions. But to optimize the loss on the Training data the model might learn to memorize very specific features of the Training data instead of developing an abstract understanding of the structure of the given data type. This could potentially lead to a high discrepancy between the achieved loss on the Training compared to the Testing data.
On the other hand, **Underfitting** is the exact opposite of Overfitting. This occurs when the model is **not able to learn** the specific characteristics and variability of the Training data [2]. Therefore, there has to be a balance between Training on a known dataset and the resulting loss on new and unknown data.

3.3. Concept and Design

Since the specific behaviors and patterns of EEG data in relation to imaginated or actually executed sensorimotor tasks has not yet been fully researched and understood, it is difficult to build a solution for the given problems with a conventional algorithm-based system. Classification Tasks like the described one can often be implemented more intuitively using ANN. Using **Supervised Learning**, the developer does not

necessarily have to know how to destructure and analyze the given data himself, but can train a model with given data to get an abstract understanding of the scenario at hand. Since EEG data is recorded over time and contains spatial information since the measurements come from different regions of the brain, most Machine Learning approaches for classifying electrical brain activity are based on a Convolutional Neural Network (CNN).
These are designed to find and learn from patterns in e.g. time or spatial convolutions. For this work, the CNN called *EEGNet* is used, which focuses on **classifying EEG data** inputs while keeping its inner structure simple and small. [see 2.1.4]

The model's parameters have to be configured to suit the Physionet Dataset properties, because some of the hidden layers depend highly on the used sample rate as well as the amount of EEG channels of the data (64). Using Supervised Learning, the CNN is trained on the dataset-specific motorimagery events (classes) like left/right fist clenching. Since EEG measurements are highly individual for each subject it is important to select a dataset with many different subjects to get a good global accuracy.

Due to the simple structure of the EEGNet it is very suitable to be used on low power Embedded Systems like the Jetson Nano that is used in this work. The Jetson Nano Board provides additional computing power as well as an on board GPU, which still leaves room for more complicated CNN models.

3.3.1. PyTorch Implementation with Python

As Requirement 1 demands, the chosen approach is implemented using the **PyTorch** Framework [34]. The implementation of the EEGNet model was originally authored by *Tibor Schneider* in his paper *Q-EEGNet: an Energy-Efficient 8-bit Quantized Parallel EEGNet Implementation [...]* [17], which implements the EEGNet model version 8.2 from the original proposition in *EEGNet: a compact convolutional neural network [...]* [11].
The given model allows to be trained for n-class classifications, which means it can be configured for multiple amounts of EEG event types. In this paper, the performance of the EEGNet is determined for **2-, 3- and 4-class classification**. A general overview of the implemented features can be seen in Table 3.2.

Table 3.2.: Implementation Features

Feature	**Description**
Command Line Interface	Interface to start and configure all available modes with all important parameters
Global Training	Training on Physionet Dataset with 5-Fold Cross Validation
Inferencing Benchmarking	Performance benchmarking for Inferencing on EEG Trials
Subject-specific Training	Transfer Learning for pre-trained model with new subject data
Live Simulation Mode	Simulating realtime usage with incoming data in a sliding time window
Visualizations	Plots and statistics to evaluate results of all modes

3.3.2. Global Training Process on Physionet Dataset

Data Loading

Before being able to use the system to classify new EEG data, the model has to be trained with Supervised Learning as described in Section 2.1. At first, the actual data used for the Training process has to be loaded in memory. **Loading the Physionet Dataset** is quite simple since the recordings are stored in individual files for each subject and run. Each file contains the EEG measurements for each EEG Channel as well as labels for the individual motorimagery events that occur. The data also contains timestamps for each sample in the dataset. These timestamps can be used to divide the run into the individual Trials, by using the metadata of the provided labels, which contain starting time of the event, duration and the according event type/label. For that purpose, the *MNE* Library [6] is used in the implementation of this work, which provides ready to use functions to get the so called *epoched* EEG data, which is represented by a numpy array [7].

The **dimensions** of this data array are **Subjects, Trials, EEG Channels, Timepoints**. To be able to feed the data into the EEGNet, it is divided into batches of Trials, without the subjects dimension. Another dimension also has to be inserted, since the first convolutional Layer of the CNN expects an input shape of (Trials, 1, EEG Channels, Timepoints) [see 2.1.4]. The amount of Trials corresponds to the used batch size and can be variable. The final, fully connected layer of the EEGNet returns an array of its prediction of what class has been fed into the model. The class with the highest prediction value is the predicted class of the input. Since this last layer of the EEGNet depends on the amount of classes to be distinguished, it is not possible to use e.g. a model that has been configured and trained with 2 classes, with data that contains 3 or more classes, since this means a change in the structure of the model and therefore different shapes of the trained weights and parameters of the network.

Because the amount of Trials of **subjects 88, 92, 100, 104** differ from the rest, they are **excluded** from Training which leads to a total number of 105 subjects (see [9, chap. 2]). Since the Physionet Dataset does not offer the same amount of Trials per subject as well as the **same amount of Trials per class** for every subject, it

is important to ensure an equal amount of Trials per subject and an equal amount of Trials per class manually in the implementation. For each subject 21 Trials per class per subject are extracted which leads to a total number of 4410 Trials for *n=2*, 6615 for *n=3* and 8820 for *n=4* in the dataset.

5-Fold Cross Validation

To address the problem of potential Over-/Underfitting as described in Section 3.2, the Training Process is implemented using **5-Fold** Cross Validation, with **100 epochs** per Fold. This means the dataset is split into subsets of subjects (21 subjects per subset, which results in 5 subsets) and for each Fold, 4 subsets are used for Training and 1 for the Testing of the trained Model. In every Fold, the Training and Test set contain different subjects (see Figure 3.1).

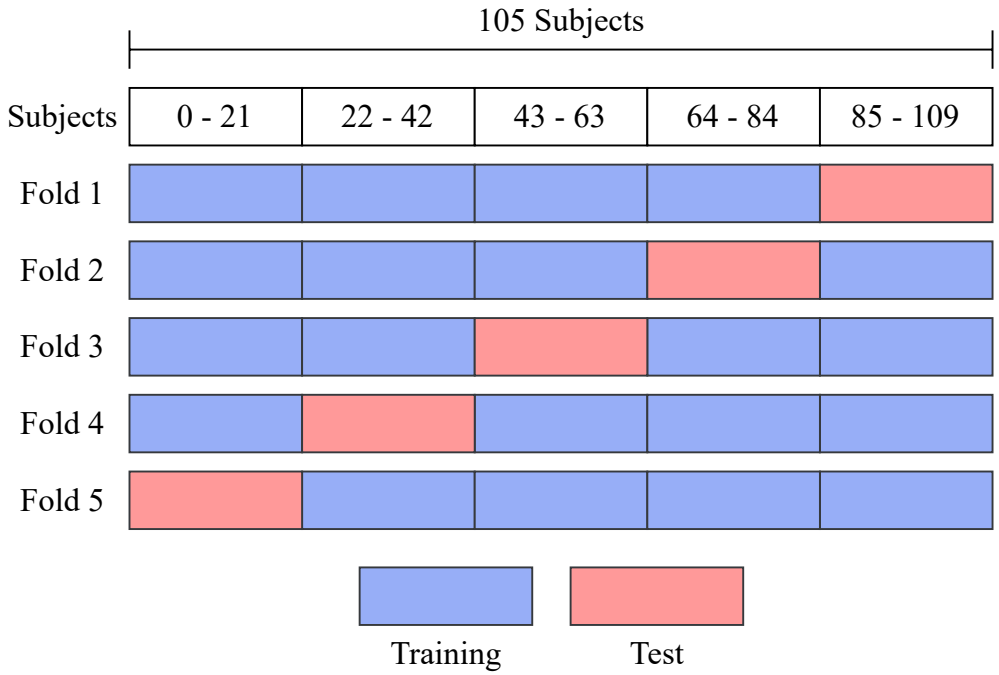

Figure 3.1.: Physionet Dataset Folds for 5-Fold Cross Validation with 21 Subjects per subset

This ensures that the given results are valid globally across the entire dataset and do not contain a subject-specific bias. In every Fold, the Training set is fed into the model to infer on it, calculate its loss according to the predicted outputs and perform **backpropagation** to update the weights and biases of the hidden layers.

Since requirement 5 demands comparability of the results for the n-class classification accuracies, the Training process was adjusted according to the configuration used in *"An Accurate EEGNet-based Motor-Imagery Brain-Computer Interface for Low-Power Edge Computing"* [21]. You can find the used **parameter settings** in Table 3.3. The *F1* value is the amount of spectral filters in the EEGNet model, the *Pool Size* controls the pooling size of the first Average Pooling in the model as well as the number of features fed to the final layer. For more information on these parameters refer to the EEGNet architecture in the original proposal [11, Table 2]. As for the loss function the Exponential Linear Unit(ELU) is used with an Adam Optimizer. The *Learning Rate* starts at 0.01 and gets divided by 10 starting with epoch 20 and gets again divided by 10 at epoch 50.

Table 3.3.: Training parameters for comparison with referenced work [21]

Parameter	**Value**
Epochs	100
Splits	5
n_classes	[2, 3, 4]
Learning Rate	Start= 0.01, milestones=[20, 50], gamma= 0.1
Batch Size	16
Trial Interval	[0;3]s
F1	8
Pool Size	4
Dropout	0.4
Loss Function	ELU
Optimizer	Adam

After the 100 epochs of Training are completed, the trained model is **validated using the Fold's Test set**. It is inferred on in batches, just like in the Training process. For each inference the indexes of the class with the highest output values are used as the prediction for a Trial. These predictions are compared to the actual labels (numbers starting from 0) provided by the dataset and therefore an accuracy

on the Test set can be determined. Furthermore, after each epoch of Training the loss on both the Training set as well as the Test set is calculated and stored for further analysis, as can be seen in Chapter 4. This process is repeated for all 5 Folds, so that an average accuracy of the EEGNet model can be determined by **averaging the accuracies** of each of the folds. Since every Fold's Test set contains different 21 subjects, in total all subjects have been validated on after the 5 Folds, therefore the **average accuracy of all Folds** can be declared as a global accuracy across all subjects. A visualization of the Training Process can be found in Figure 3.2. For the Fold with the highest accuracy on the Test set (called **Best Fold**), the trained weights and biases of the model are saved to a file to be able to load the pretrained model for e.g. benchmarking or inferencing.

To further analyze and improve the accuracy on the given dataset, the possibility of **Trials Slicing** is also implemented. Instead of learning from the full single Trials of time length `TMAX-TMIN`, the Trials can be **split into smaller Trials**. Every Trial is split into k equally long parts, where the time length is:

$$l(k) = \frac{TMAX - TMIN}{k}$$

This leads to a bigger sample size during Training and might improve the accuracy.

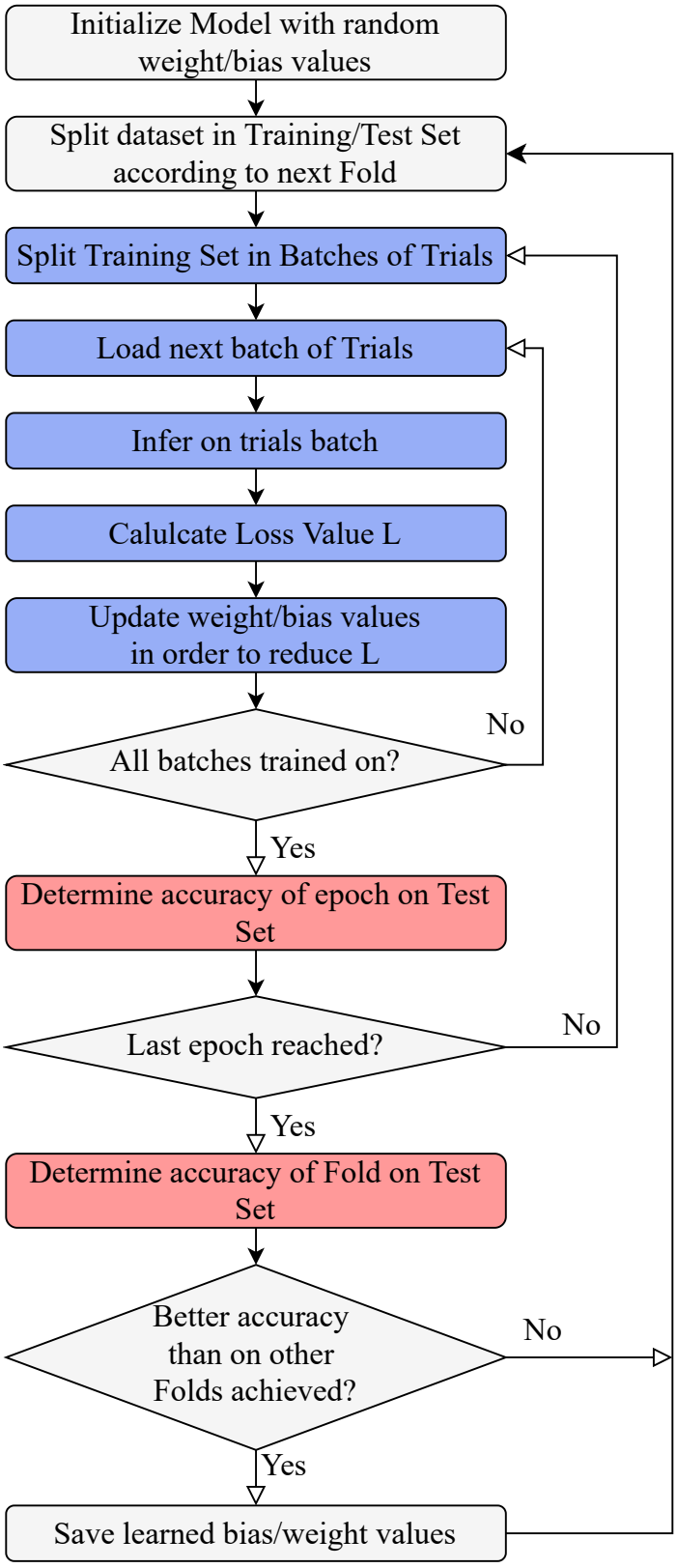

Figure 3.2.: 5-Fold Cross Validation Training Process

3.3.3. Benchmarking of Inferencing Time

To measure the actual performance of the described system, multiple properties are important. Since the Batch Size per Inference can vary it is vital to look at the **Batch Latency**. This latency states the time that is needed to copy the input data onto the GPU, infer on that data and return it from the GPU into memory again. To ensure the GPU is up and running, a quick warm-up before the actual benchmarking is executed. To get the GPU out of any possible sleep or idle modes, some random inputs are copied to the GPU and inferred on. It should be noted that for the implementation of measuring the time to infer one batch it is necessary to omit the time to actually load the batch data into CPU memory since this is system dependent. For actually using the system with live EEG measurements to for example control a machine it is important to look at the **inference time per Trial** or the **Trials per Second(*tps*)** to ensure the system is responsive in real time:

$$tps = \frac{1}{BatchLatency} * BatchSize$$

To fully utilize the chosen hardware platform, different batch size and optimizations like TensorRT or the usage of the Jetson Nano CUDA cores have been tested, as described in Section 3.3.6 and Section 3.3.4.

3.3.4. CUDA Acceleration on Jetson Nano

The here used NVIDIA Jetson Nano board is a high performance low power system which also comes with a built-in NVIDIA GPU that supports CUDA. According to existing works [14][18], the execution of Machine Learning algorithms can highly profit from **parallel execution on GPUs**. To take advantage of the Jetson Nano's CUDA enabled GPU, all the data as well as the CNN model parameters have to be transferred to the GPU memory and can then be utilized on the 128 CUDA cores of the Jetson's GPU. Another advantage is that the Batch Size can be increased until the GPU is fully utilized while executing the Inference, which further improves the Trials per Seconds. The effect on the performance can be seen in Chapter 4.

3.3.5. Selection of significant EEG Channels for Motor Imagery Events

Since one of the initial requirements is to **minimize the amount of EEG Channels** used to classify sensorimotor events, different EEG Channel configurations were tested. As stated in Section 2.2, there is a **specific area of the brain** that controls sensorimotor actions which are the relevant types of action in this scenario. To validate the possibility of using only electrodes in said areas, different combinations of 16 electrodes in the area of the motor and sensory cortex were selected from the Physionet Dataset.
Since it is not always clear from which specific parts of the dataset the neural network model learns the most from, there is also a channel selection with no electrodes in the described sensorimotor cortexes (`chs_16_bs`). If the model truly learns the most from the electrodes that are in the corresponding areas, this **"random" selection of channels** should only have a very low accuracy in comparison to the other selections. Figures 3.3 - 3.6 show the different channel selections that have been tested. The `16` and `chs_16_1` selections only contain channels inside of the sensory or the motor cortex while the `chs_16_bs` selection does not contain a single channel inside these regions. Finally, the `chs_16_openbci` selection is supposed to test a possible channel configuration for the *OpenBCI Starter Kit* (16 Channels) [32]. This is done to get an estimate of how good the system could perform with actual EEG recording hardware that differs from the one used in the Physionet Dataset.

To implement this channel selection it does not take much effort. The model has a parameter for the amount of channels and the remaining channels in the dataset are simply omitted from the data loading. This reduces the amount of parameters in the model as well as highly reducing the amount of data that has to be loaded and inferred on.

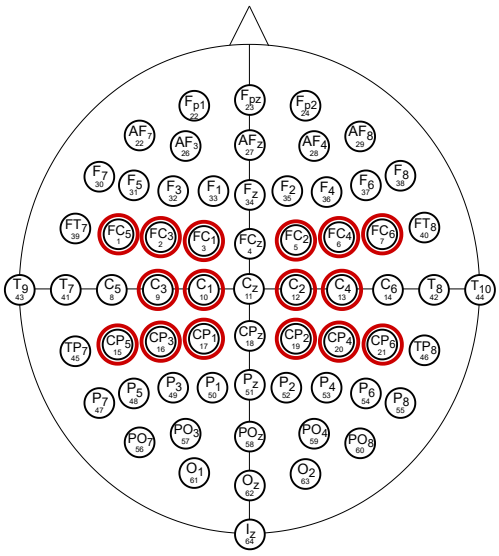

Figure 3.3.: `chs_16`
Channels Selection

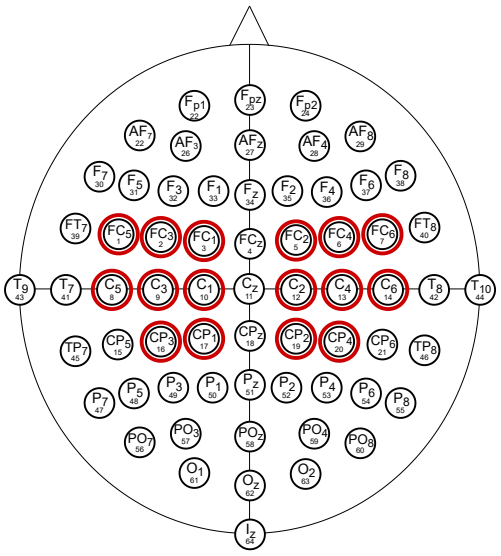

Figure 3.4.: `chs_16_1`
Channels Selection

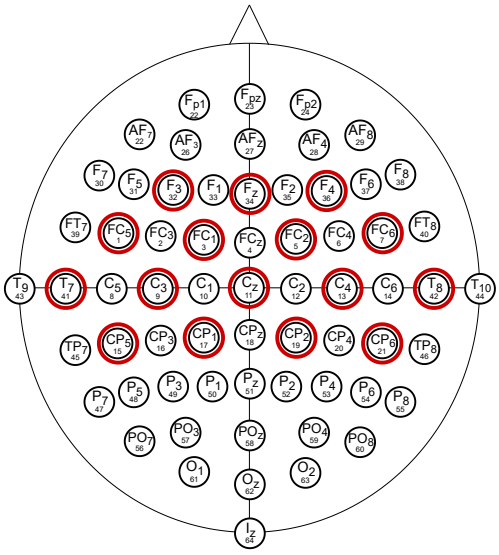

Figure 3.5.: `chs_16_openbci`
Channels Selection

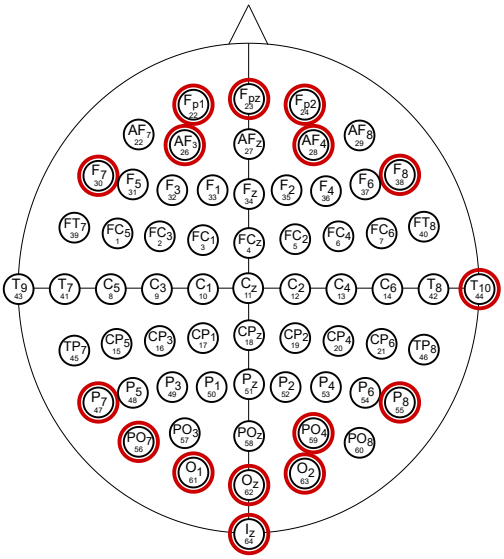

Figure 3.6.: `chs_16_bs`
Channels Selection

3.3.6. Optimization of trained EEGNet with TensorRT

To further **optimize the benchmark performance** while inferencing on a neural network, there are several SDKs to achieve better results. One of the most popular ones is *TensorRT*, developed and maintained by NVIDIA [31]. It is based on CUDA, which enables **highly parallel executions** of Machine Learning algorithms on multi-core GPUs.
To achieve a better and faster inferencing rate, TensorRT provides a multitude of different optimizations as stated in the documentation:

1. **Weight Activation Precision Calibration** Maximizes throughput by quantizing models to INT8 while preserving accuracy
2. **Layer Tensor Fusion** Optimizes use of GPU memory and bandwidth by fusing nodes in a kernel
3. **Kernel Auto-Tuning** Selects best data layers and algorithms based on target GPU platform
4. **Dynamic Tensor Memory** Minimizes memory footprint and re-uses memory for tensors efficiently
5. **Multi-Stream Execution** Scalable design to process multiple input streams in parallel
6. **Time Fusion** Optimizes recurrent neural networks over time steps with dynamically generated kernels

(Citation of NVIDIA's TensorRT Documentation [31])

Another interesting optimization is the possibility to select either **floating point(fp)** 16 or fp32. This option allows to set the precision that the inner parameters of the model and inputs are converted to. By using fp16 the memory usage can be halved, which should also lead to higher performance. The possible downside to fp16 is, that it has to be validated if there is a loss in accuracy due to the lower precision. The influence of fp16 is addressed in Section 4.2.
Since TensorRT is tightly connected to Tensorflow, another Python library called *torch2trt* [27] has to be used in order to use TensorRT with a PyTorch model. It

allows to easily convert an existing and trained model into a TensorRT optimized network, which can then be used just like any other PyTorch model.

3.3.7. Subject Specific Training

Requirement 6 states that a pre-trained model should also be able to be further trained with a new subject. This is called **Transfer Learning**. The idea is that you put the most effort into the global Training of the CNN like described above and then to further **increase the accuracy for the new subject**, you execute a small Training process before actually using the model for the subject. Transfer Learning requires much less data than the global Training, since the model should already be trained quite well and be able to abstract from subject specific patterns.
In this case, for the 2- and 3-class classification we have 3 Runs available from Task 2 for every subject. They amount to 42 Trials for 2-class and 63 Trials for 3-class classification for the single subject. 4-class classification is not of much use when doing subject specific Training, since it also uses another Task and only extracts 1 more class and omits the remaining Trials of the extra Runs. The execution of the subject specific Training is exactly the same as in the Global Training Process 3.3.2, except that we do not cross-validate with different Folds. The **Trials from Runs 4 and 8** of Task 2 are used **for Training**, while Trials from **Run 12** are used **for Testing** (see Figure 3.7). After the Transfer Learning, the model is more subject specific to the newly introduced subject, which should lead to a better accuracy.

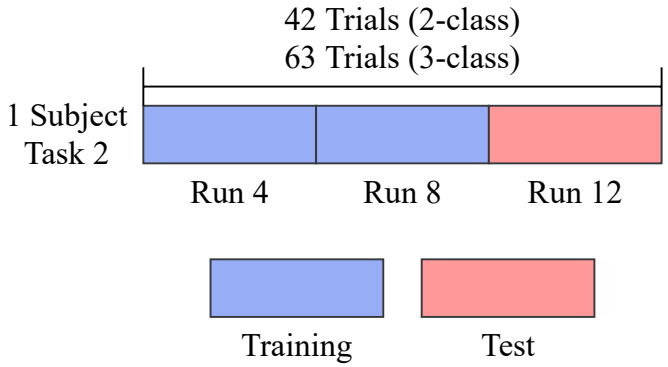

Figure 3.7.: 2- and 3-class Classification Subject Specific Dataset of 1 Subject

3.3.8. Live Simulation of Subject Run

As Requirement 7 states, it is desired to implement a mode that can be used in a real time manner, meaning **classifying live EEG data** provided by actual hardware connected to an actual subject.
This classification process differs from the above described Training process. Instead of having fully labeled data the data is inferred and predicted on in real time. Furthermore, instead of having a single set time frame per Trial like in Training, the live simulation works with a **sliding time window**. It constantly moves forward with every new timepoint and so the inference's input are always the samples from the last 3 seconds (`TMIN=0`, `TMAX=3`) of the recording. A visualization of this sliding window is shown in Figure 3.8.

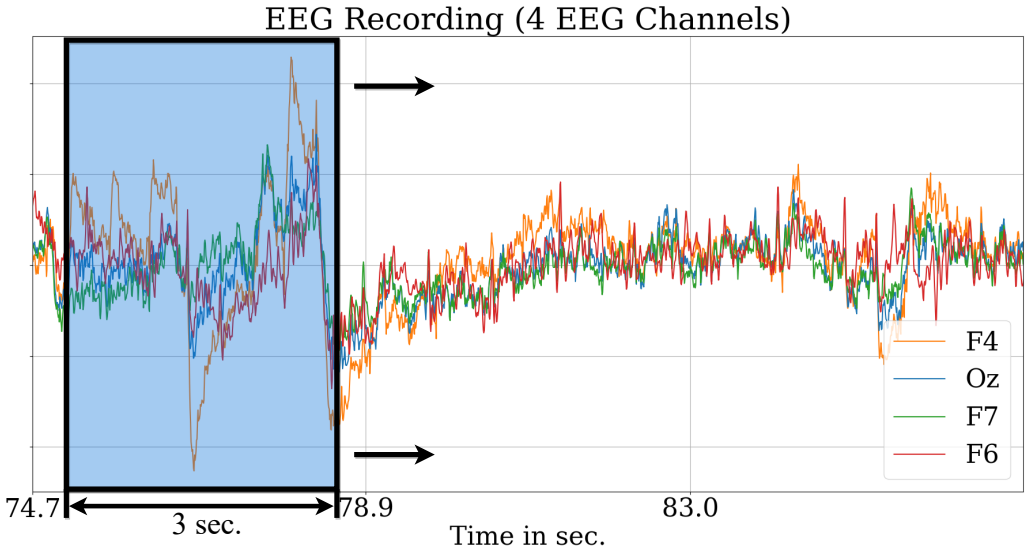

Figure 3.8.: EEG Recording (4 EEG Channels) with 3 sec. Sliding Window

3.4. Implementation

The project's implementation is split into different packages with different scripts. A basic overview of each of the packages(folders) and their containing scripts is shown in Table 3.4. Since Requirement 2 (see 3.1) demands that the Training process is highly configurable concerning hyperparameters and setup of the Supervised Learning, the implementation provides a number of **global configuration variables**. All the hard coded parameters are stated in Table 3.5.
The main script to run either the Training, Benchmarking or Live-Simulation modes is the `main.py` script. Since the system should be highly configurable (Requirement 2), it is executed via a **Command Line Interface** that offers a lot of arguments to further set important parameters. All of the available arguments of the `main.py` script are listed in Table 3.6.

The implementation was developed using the IDE *PyCharm* by *JetBrains* [33]. Python 3.6.9 [20] was used with the following packages installed:

- mne 0.22.0
- numpy 1.17.0
- scipy 1.5.4
- torch 1.7.0
- tqdm 4.56.0
- matplotlib 3.3.3
- pandas 1.1.5
- scikit_learn 0.24.2

Furthermore, the *NVIDIA CUDA Toolkit 10.2* [26] as well as *TensorRT 7.1.3* [31] was installed on the Jetson Nano to enable GPU usage as well as more optimizations.

Table 3.4.: Main Files and Packages(Folders)

File / Folder	**Description**
`config.py`	Global Default Values
`main.py`	Entrypoint to start Training, Benchmarking, Live-Simulation
`machine_learning/`	Folder for all Scripts related to Machine Learning
`modes.py`	Main methods for all modes like Training, Benchmarking, Live-Simulation
`inference_training.py`	Dataloader Loops for training, benchmarking, testing
`eegnet.py`	EEGNet PyTorch Implementation
`data/`	Folder for all Scripts related to Data Loading/Preprocessing
`physionet_dataset.py`	Configuration Values for Physionet Dataset Usage
`data_loading.py`	Loading and preprocessing EEG Data with MNE Library
`data/datasets/`	Folder for all available Datasets (until now only Physionet Dataset)
`results/`	Configs and Result Files, Plots for all Modes
`requirements.txt`	List of all used Python packages

Table 3.5.: Training Parameters

Parameter	Description	File
EPOCHS	Amount of Epochs during Training	config.py
SPLITS	Amount of Splits for Cross Validation	config.py
N_CLASSES	List of n-class classifications to be executed	config.py
LR	Learning Rate Parameters (Start, Gamma)	config.py
BATCH_SIZE	Amount Trials per Batch	config.py
global_config	Frequency Filters (High/Low/Notch)	config.py
PHYSIONET	Time Window for each Trial, Trials Slices, Samplerate	physionet_dataset.py
REST_TRIALS_FROM_BASELINE_RUN	Whether or not the rest Trials are taken from the baseline run or the actual runs	physionet_dataset.py
TRIALS_PER_SUBJECT_RUN	Amount of Trials per Run of a Subject	physionet_dataset.py
excluded_subjects	Subjects to be excluded from the used Dataset	physionet_dataset.py

Table 3.6.: main.py Command Line Arguments

Argument	Description
n_classes	List of nclass classifications to run (2/3/4Class possible)
name	Name for executed Run (stores results in ./results/{benchmark/training}/{name})
tag	Optional Tag for results files (e.g. for different batch sizes)
device	Device to use, either "gpu" or "cpu"
bs	Trial Batch Size (default:16)
model	Relative Folder path of trained model(in ./results/.../training/ folder), used for benchmark or train_ss or live_sim
subject	Subject used for live_sim or train_ss
trials_slices	Will slice every Trial into n x Trials (default: 1)
tmin	Start Time of every Trial Epoch (default: 0)
tmax	End Time of every Trial Epoch (default: 3)
train	Runs Training with Cross Validation with Physionet Dataset
epochs	Number of epochs for Training (default:100)
ch_names	List of EEG Channels to use (see config.py MNE_CHANNELS for all available Channels)
ch_motorimg	Use and set amount of predefined Motor Imagery Channels for Training (either 5,8,8_2,8_3,8_4,8_5,8_6, 12,14,14_2,14_3,14_4,16,16_2,16_openbci,16_bs,18,21 channels)
all_trials	Use all available Trials per class for Training (if True, Rest class ('0') has more Trials than other classes)
early_stop	If present, will determine the model with the lowest loss on the validation set
excluded	List of Subjects that are excluded during Training (default excluded Subjects:[88, 92, 100, 104])
train_ss	Runs Subject specific Training on pre-trained model
benchmark	Runs Benchmarking with Physionet Dataset with specified trained model
subjects_cs	Chunk size for preloading subjects in memory (only for benchmark, default:10, lower for less memory usage)
trt	Use TensorRT to optimize trained EEGNet
fp16	Use fp16 for TensorRT optimization
iters	Number of benchmark iterations over the Dataset in a loop (default:1)
all	If present, will only loop benchmarking over entire Physionet Dataset, with loading Subjects chunks in between Inferences (default: False)
live_sim	Simulate live usage of a subject with n_class classification on 1 single run

4. Results

In this Chapter the achieved results for all the above described modes are shown.

4.1. Global Training results of 5-Fold Cross Validation

The following section showcases statistics of the results for the Global Training. A default configuration (*defaults*) was defined to act as a reference for any further parameter testing. Training parameters such as Trials Slicing, Trial Time Window and Channel Selection have been further analyzed and validated in reference to the *defaults* configuration's performance.

4.1.1. Defaults

For the *defaults* Training configuration the following parameter values were used:

Table 4.1.: Training Parameters (see Figure 3.5)

Parameter	Value
EPOCHS	100
SPLITS	5
N_CLASSES	[2, 3, 4]
LR	Start= 0.01, milestones=[20, 50], gamma= 0.1
BATCH_SIZE	16
global_config	No Filters
PHYSIONET	TMIN=0, TMAX=3, SAMPLERATE=160
REST_TRIALS_FROM_BASELINE_RUN	True
TRIALS_PER_SUBJECT_RUN	21
excluded_subjects	[88, 92, 100, 104]

The results are visualized by different plots and tables:

- **Training Losses**: The Losses on the Training Set for each epoch of every Fold
- **Train-Test Losses of best Fold**: The Losses on the Training and Test Set of the Fold with the highest accuracy on the Test Set
- **Cross Validation**: Accuracies on the Test Sets for every Fold with the average of all Folds highlighted with dotted red line
- **Class Accuracies**: Accuracies of the n individual classes on the Test Set of the Fold with the highest accuracy with the average of all classes highlighted with dotted red line
- **Confusion Matrix of best Fold**: Confusion Matrix (see 2.1.3) containing all Trials predictions for all n classes of the Fold with the highest accuracy on the Test Set with the precision and recalls for every class

For the exact results metrics values as well as the Overfittings refer to Table A.1.

2-Class Training

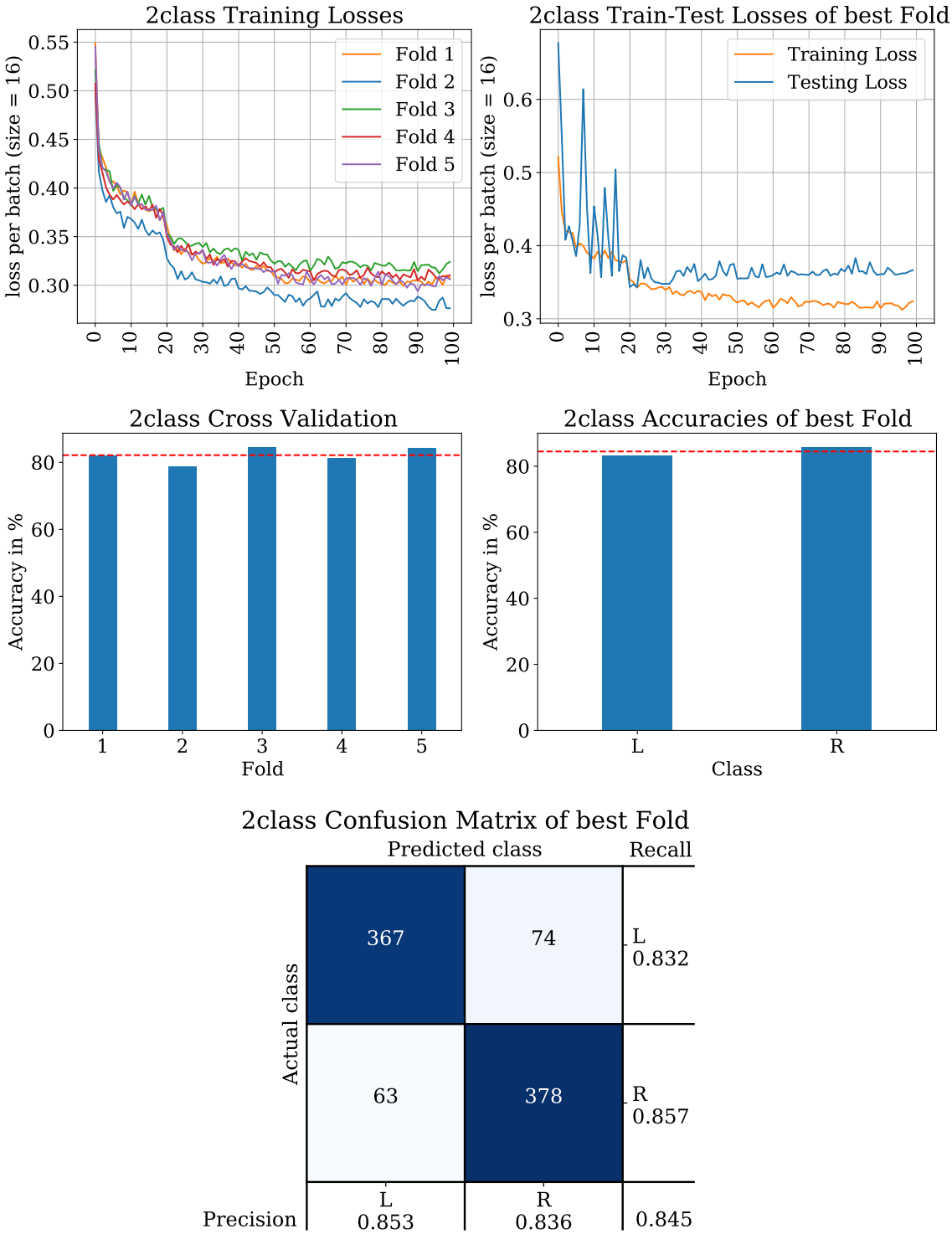

Figure 4.1.: 2class Training Results

3-Class Training

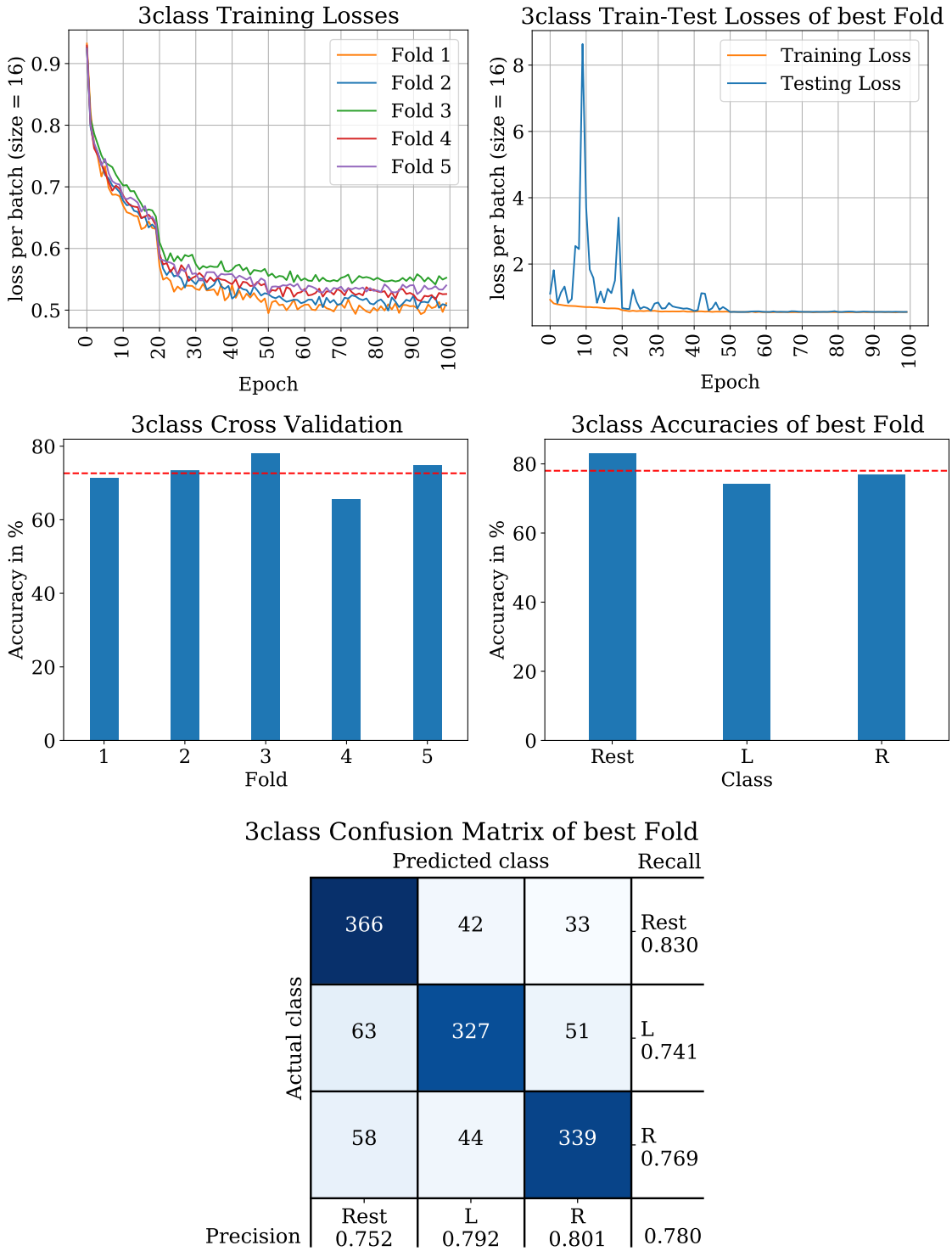

Figure 4.2.: 3class Training Results

4-Class Training

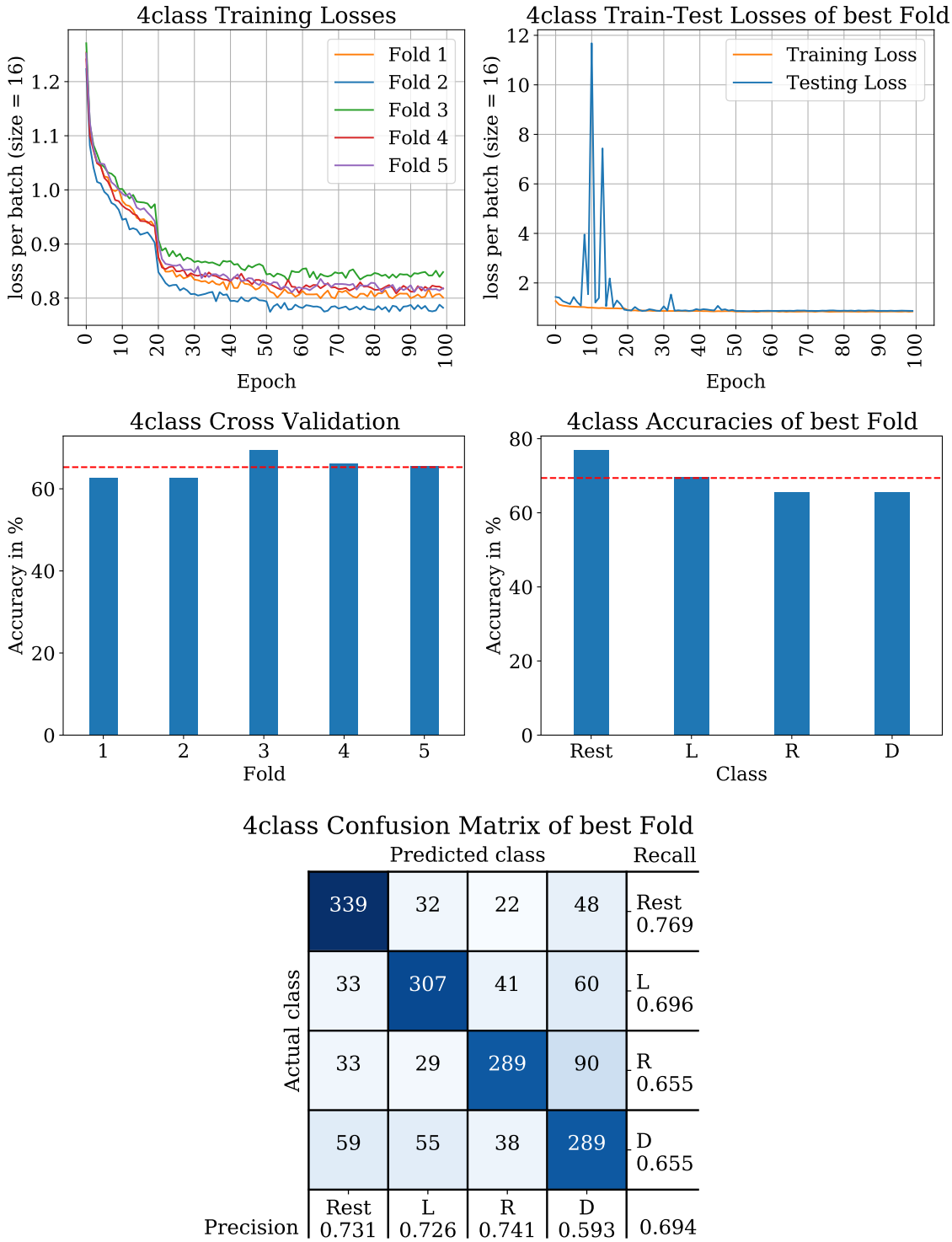

Figure 4.3.: 4class Training Results

The Figures 4.1 - 4.3 visualize the achieved Training results for the *defaults* configuration. For all n-class classifications the **Training loss consistently decreases** and converges to a minimum value over the 100 epochs. In the earlier epochs ($<$ 20), the Testing loss of the best Fold has very high spikes. This is very common to happen since in the first few epochs the model is not well trained yet and its predictions on not-yet seen inputs can be very random and therefore its loss can go up significantly for some isolated epochs. But after these initial spikes we can see that the Testing loss also converges to a similar level as the Training loss. The accuracies of the individual Cross Validation Folds vary by a few percent for every n-class. This is to be expected and since there is no huge differences in accuracy, the values seem reasonable. For the accuracies of the individual classes we can see that for *n=2* both classes have nearly exactly the same accuracy. For *n=3,4* there is a bit more variation, the biggest difference is between the classes *Rest* and *R* for *n=4* with a difference of ~15%. This specific case can probably be explained due to the fact that the *Rest* Trials in theory should be much more similar to each other in their patterns because they in general have lower brain activity. The sensorimotor classes *L*, *R* and *D* on the other hand fluctuate a bit more.

Since the results should be validated by looking at other works (Requirement 5), the here shown results are compared to the achieved accuracies in the paper "*An Accurate EEGNet-based Motor-Imagery [...]*" [21] in Table 4.2. The accuracies of all n-class classifications in this work are very similar to the existing results. Also, for all classifications the achieved accuracies are **well above the chance levels** (see 3.1) and therefore they can be acknowledged as high global accuracies (Requirement 3). Looking at the average **Overfitting** (see Table A.1), it stays around -5% to -8% consistently for all n-classes. Yet this means a clear Overfitting is present, which could lead to slightly worse performance with new EEG data.

Table 4.2.: Defaults Training Accuracies

Accuracies on Test Set		
Classification	**EEGNet Edge-computing [21]**	**This work**
`2class`	82.43	82.01
`3class`	75.07	73.92
`4class`	65.07	65.57

4.1.2. EEG Channel Selection

As described in Requirement 8, the biggest effect on reducing the cost of the necessary EEG hardware is minimizing the channels. The following Figure 4.4 shows the achieved results with the 4 16-channels selections introduced in Section 3.3.5. All shown accuracies are the averages over all 5 Cross Validation Folds.

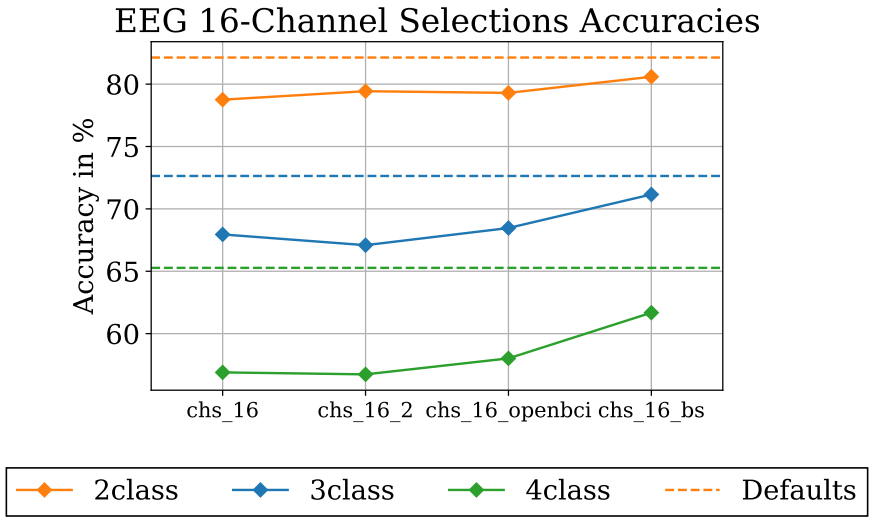

Figure 4.4.: 2,3,4-class Classification Accuracies for 16-Channel Selections and *defaults* configuration Accuracies (dotted lines)

Reducing the amount of used channels from 64 to the desired 16 **decreases** the **accuracies** for all classes. Also there is an increase in the loss of accuracy from *n=2* up to *n=4* (difference of the achieved accuracies and the *defaults* [dotted lines]). For *n=2* the accuracy loss is about 2-4%, for *n=3* 1-5% and for *n=4* 4-9%.
It is interesting to see that out of the 4 chosen selections, the `chs_16_bs` performs the best, with the least accuracy loss across all n-classes classifications. This is the only selection where none of the electrodes are located in either the **sensory or motor cortex**, which were to believed the best regions to be used for sensorimotor Tasks classifications. This leads to the question how this phenomenon comes to pass, which is further discussed in Chapter 5. If we look at the Overfitting values for the 16-channel configs (see Table A.1), reducing the amount of channels also reduces the Overfitting by ~2-3% for all n-classes.

4.1.3. Trials Slicing

In the following the impact of Trials Slicing on the n-class accuracies are described. All Trials where split into k slices, which leads to k times the amount of Trials for the Training. All shown accuracies are the averages over all Cross Validation Folds. As a Time interval `TMIN=0` and `TMAX=4` was used, which means the entire duration of every Trial is loaded.

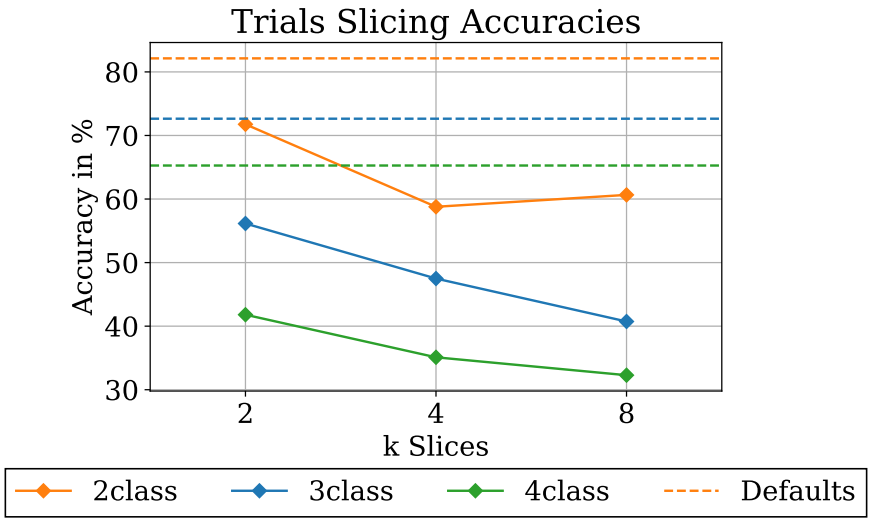

Figure 4.5.: 2,3,4-class Classification Accuracies for Trials Slicing and *defaults* Accuracies (dotted lines)

Figure 4.5 shows that the more slices (higher k) are used, the bigger is the **loss in accuracy**. Even only using k=*2*, meaning twice the amount of Trials with half the length each, leads to a loss in accuracy of about 11% for n=*2*, 16% for n=*3* and 24% for n=*4* and it gets even worse for k=*4* and k=*8*. For k=*8* the accuracies are merely 10% above the chance levels. This could hint to the fact that the model was not able to abstract the patterns of the different classes from only **small parts of the Trials**. Another, yet minor side effect, is that with higher k, the time needed for Training increases as well, since smaller but many more Trials are being trained on.

4.1.4. Trial Time Window

A very import parameter affecting the amount of data being used is the Trial Time Window. It is defined by the interval of `[TMIN;TMAX]` and sets the starting and ending time delta of every Trial relative to the starting timepoint of the Trial. In the Physionet Dataset every Trial is about 4 seconds long. But as can be seen in the *defaults* configuration, not the whole 4 seconds have to be used in order to reliably classify the Trial. So in the following Figure 4.6 you can see the achieved results for other Trial Time Windows. All shown accuracies are the averages over all 5 Cross Validation Folds.

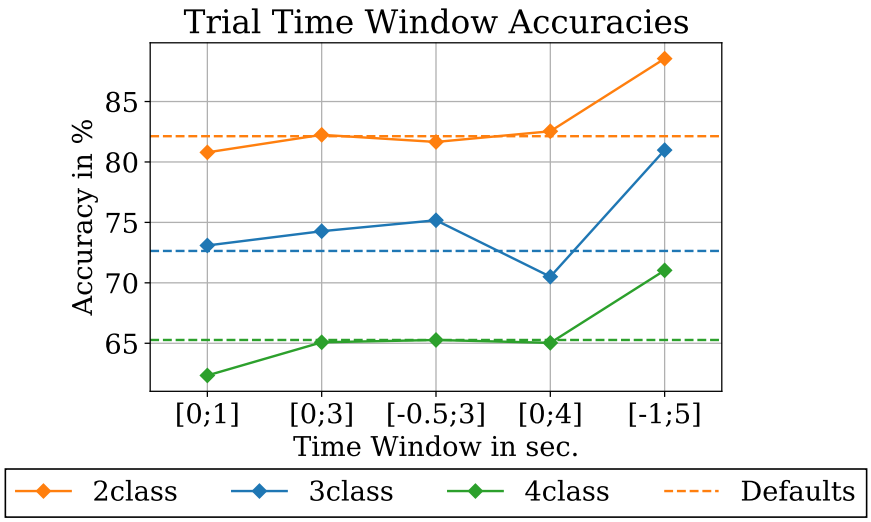

Figure 4.6.: 2,3,4-class Classification Accuracies different for Trial Time Windows and *defaults* Accuracies (dotted lines)

We can see that for `TMIN=0` and `TMAX=3/4`, the accuracies are very similar, while a small loss in accuracy of a few percent can be seen for the window `[0;1]`. This shows that the model still performs well even with much **smaller Time Windows**. Even only with one single second of a 4 second Trial it is still close to the *defaults* accuracies. An interesting observation is that the Time Window of `[0;1]` is exactly the same as the first slice of using Trials Slicing with *k=4* with `TMIN=0` and `TMAX=4` as described in the section above. Yet the accuracy for this Time Window is much higher than with *k=4* with 4 1-second Slices. For *k=4* it achieved ~59%, ~47% and ~35% for *n=2,3,4* whereas with *[0;1]* it achieved ~81%, ~73%, ~62%. This hints to

the phenomenon that the model can learn very well from the 1st second of a 4 second Trial but can not learn as much from the other 1-second slices of that Trial.
There seems to be a small anomaly for the 3class classification with `[0;4]`, where the accuracy drops below the *defaults* one. A big improvement can be seen with the interval `[-1;5]`, the accuracies rise significantly, for *n=2* by ~6%, for *n=3* by ~8% and for *n=4* by ~6%. But it has to be noted that negative values for `TMIN` mean it also uses a fraction of the Trial before the currently used Trial. This might hint to the fact that it could be beneficial to use a Trials **overlapping Time Windows** instead of static Training of the single Trials.

4.2. Inferencing Benchmarking

The following plots showcase the achieved performance with the EEGNet model. Inferencing was tested with *n=2,3,4* on the pre-trained EEGNet model with the above given trained *defaults* model. For each benchmark the n-class Trials data for 10 subjects of the Physionet Dataset are loaded and inferred on 10 times each. The showcased results are the averages of these 10 inferencing iterations.
To test the impact of using either the CPU or GPU, optimizing with **TensorRT with fp16/fp32** (see Section 3.3.6) and using **different batch sizes**, 4 different configurations have been defined, as shown in Table 4.3. It has to be noted that for `c_cpu` using the CPU as the device, only a batch size of 8 was used, since the Jetson Nano's CPU can not handle sizes bigger than 15.

Table 4.3.: Benchmarking Configurations

Parameter	`c_cpu`	`c_gpu`	`c_gpu_trt_fp32`	`c_gpu_trt_fp16`
`Device`	CPU	GPU	GPU	GPU
`Batch Sizes(BS)`	[8]	[8,16,32]	[8,16,32]	[8,16,32]
`TensorRT(fp16/fp32)`	No(fp32)	No(fp32)	Yes(fp32)	Yes(fp16)

2class Benchmarking

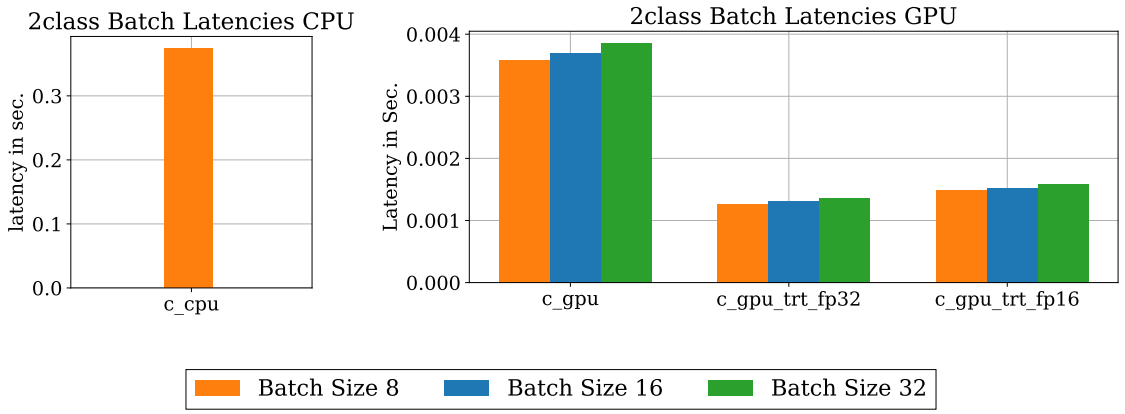

Figure 4.7.: 2class Batch Latencies

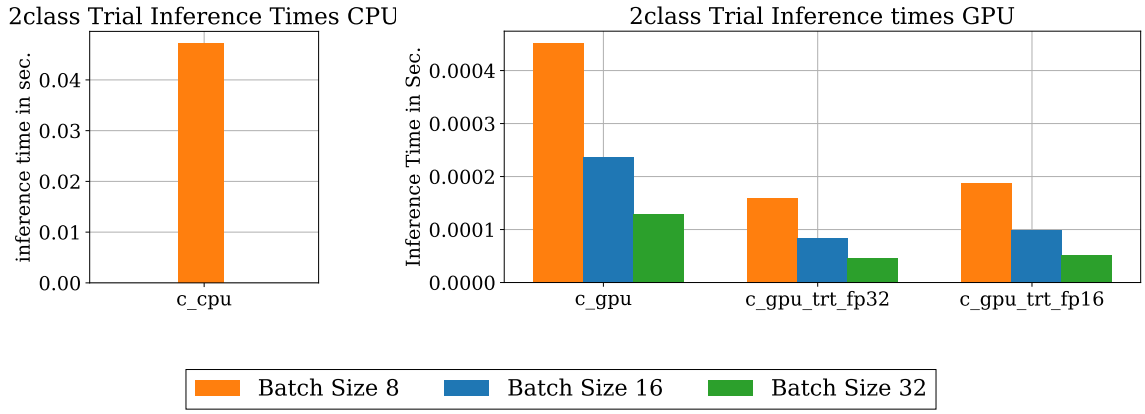

Figure 4.8.: 2class Trial Inference Times

3class Benchmarking

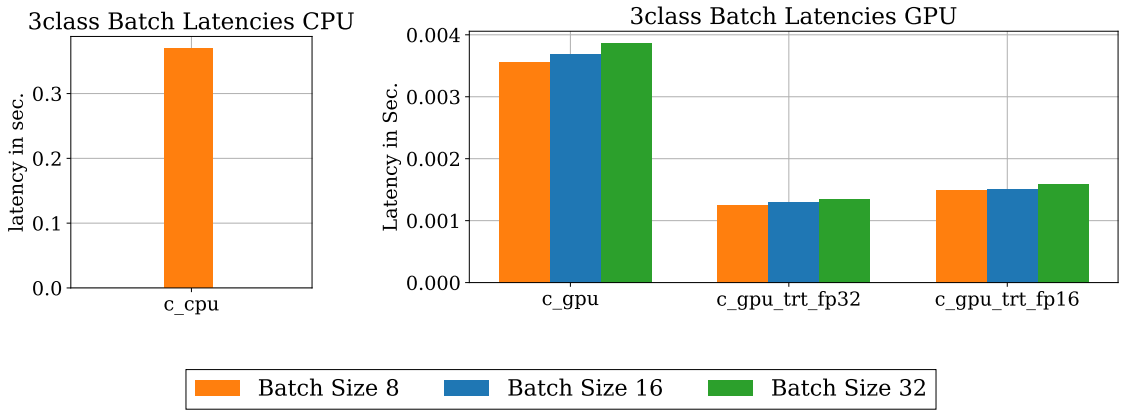

Figure 4.9.: 3class Batch Latencies

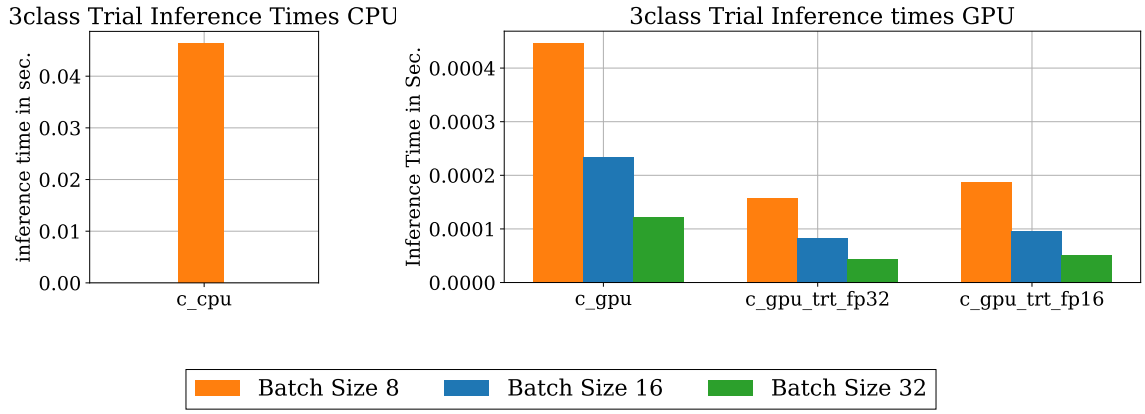

Figure 4.10.: 3class Trial Inference Times

4class Benchmarking

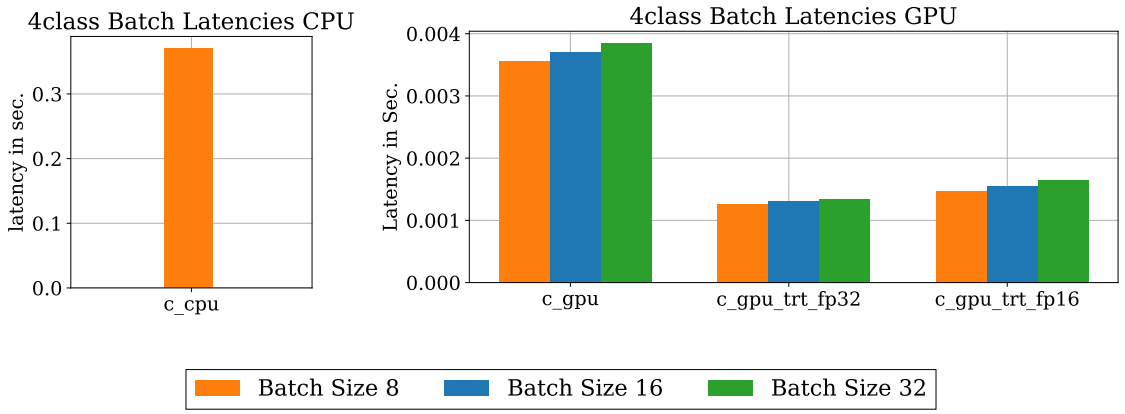

Figure 4.11.: 4class Batch Latencies

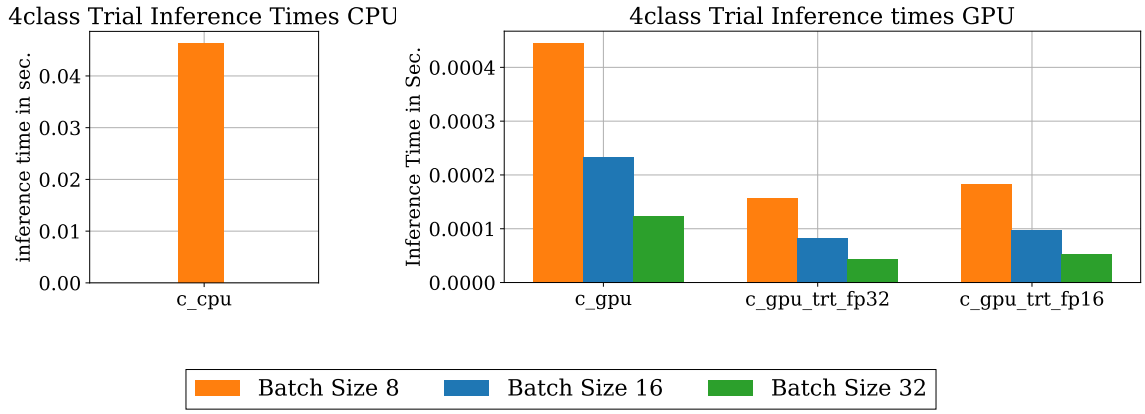

Figure 4.12.: 4class Trial Inference Times

Configuration	2class	3class	4class
`c_gpu`	90.71%	86.19%	79.05%
`c_gpu_trt_fp32`	90.71%	86.19%	79.05%
`c_gpu_trt_fp16`	90.71%	86.03%	78.45%

Figure 4.13.: TensorRT floating point accuracies

The Figures 4.7-4.12 visualize the speed performance on inferencing. For the exact result values, refer to Table A.2. As described in Section 3.3.3, the data is inferred on in batches. To fully utilize the hardware capabilities of the Jetson Nano and its GPU, **batch sizes** of 8, 16 and 32 were tested. Higher batch sizes like 64 are not possible, since the Jetson then reaches its bottleneck in memory.

You can immediately see that **using the multicore GPU** instead of the plain CPU is definitely recommended. You can not use higher batch sizes than 15 with the Jetson Nanos CPU and if we look at the performance, the **CPU** performs about **100 times worse** than any of the configurations utilizing the GPU (see Figures 4.7- 4.12). This clearly shows that the here proposed model highly benefits from the parallel execution of Machine Learning algorithms on multi-core GPUs, especially from the here used CUDA GPU.
As you can see in the **Batch Latencies GPU** plots, for `c_gpu-c_gpu_trt_fp16`, the latencies for batch sizes 8, 16 and 32 do not significantly rise with higher batch sizes. This suggests that for lower batch sizes like 8 the Jetson Nano's GPU is not fully utilized and can handle larger batches. If the batch latency stays about the same for bigger batches that also leads to **more Trial inferences per second**. This is showcased in the Trial Inference Times plots, where a steady decrease in time per Trial with higher batch size can be seen.

Another major improvement in terms of speed can be achieved with **TensorRT**, as `c_gpu_trt_fp32` and `c_gpu_trt_fp32` show. In the appended Table A.2, you can see that the Trials per second are nearly **3 times higher** with the TensorRT optimized models than without. This is a massive improvement which leads to **over 20,000 Trials** being inferred on **per second**.

It has to be addressed that there does not seem to be a benefit in terms of performance if TensorRT is used with **fp16 versus fp32**. Looking at `c_gpu_trt_fp32` and `c_gpu_trt_fp16`, fp16 actually performs slightly worse than fp32 in batch latency and Trials inference time which is why it is not recommended to use fp16 for this model. Since using only 16 bits of memory also means less precision, it might lead to a loss in accuracy as well, if the data values are more precise than that. But as shown in Table 4.13, this negative effect is only marginally present, the accuracies of fp16 are identical to fp32 for *n=2* and only less than one percent lower for *n=3,4*.

The achieved Trials per second are more than sufficient to be able to handle live incoming EEG data and classify it in real time (Requirement 4). Looking at the Trials per second it also becomes clear that the performance does not depend on the amount of classes (n) to be distinguished. For n=*2,3,4* the Trials per second are about the same, which means the model should easily be **able to handle even more different classes** in terms of performance.

4.3. Subject-specific Training

The subject-specific Training is tested by training a model with the global Training process, but excluding a subject, here subject 42 was excluded. This trained network has never seen this subject before, which simulates the scenario where **a new subject** wants to use the proposed system. In order to get higher classification accuracies, this pre-trained model is further trained with the runs of subject 42, as described in Section 3.3.7. Because of the very small Test Set (1 Run, 14/21 Trials), the accuracies are not very significant or comparable. This **subject-specific** trained **model** is solely used as an optimized model for the following Live Simulation mode, of which the results are shown in Section 4.4.

4.4. Live Simulation Results

Figure 4.14 shows the finished execution of a Live Simulation Mode Run of subject 42 with Run 12. The colored areas highlight the individual Trials and their labels. The colored lines show the prediction values for all 3 classes at every single timepoint in the Run. The vertical **dotted lines** highlight the **timepoints** for which the **Training process has been trained on**. This means during Training the data from the start of a Trial up until the dotted line in that Trial were used as the Training samples. As you can see, at nearly all of these timepoints the class with the highest prediction is the correct class of that Trial.

Before and after these specific timepoints the predictions are not as good. This leads to the conclusion that the model was able to get an abstract understanding of how the different EEG events (classes) are structured but only for the trained on timepoints and not the entire recording. This means the Training process should be further adjusted for a more variable time window on the EEG data which should lead to **more consistent predictions** across the Trials. Possible approaches to this are further described in Chapter 5.

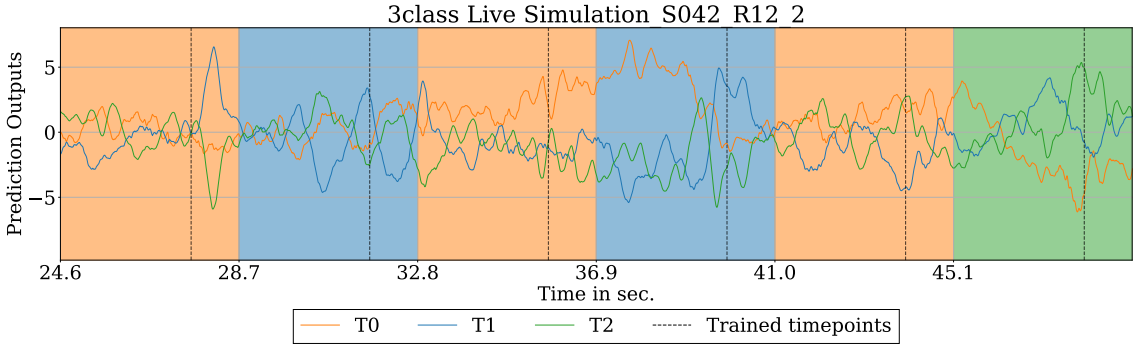

Figure 4.14.: 3class Live Simulation Run of Subject 42 (Run 12), T=[24.6;49.2]s

5. Evaluation

5.1. Discussion

Since this work is supposed to be usable as a basis for developing a full grown BCI, there are several different aspects and problems that have been revealed in the above shown results 4.

The overall goal of implementing a highly configurable testing framework for a BCI using Supervised Learning has been achieved. All the important parameters are fully customizable and can be combined however necessary. The implementation also provides different modes, from the global Training, over subject-specific Transfer Learning, Benchmarking on Inference and a Live Simulation mode. In the following the open questions from the results chapter are discussed.

5.1.1. Training Parameters

As showcased in Table 4.2, the achieved results with the same configuration as in the referenced work come very close to the existing results. This helps validating the overall correctness of the system (Requirement 4). Choosing different settings for the available parameters revealed some issues on how good and reliable the proposed model works. In the following the problems concerning the individual parameter settings are further explained.

EEG Channel Selection

One of the biggest question marks is how to best select and minimize the used channels. In section 4.1.2 we can see that for all the 16-channel selections the accuracy is lower than with the default 64 channels. Even more confusing is the fact that the `16_bs` has the highest accuracies of all 16-channel selections, without having a single electrode in the proposed regions. This might be explained by some **overall error** in the recordings affecting all channels and adding possible noise. Another explanation could be that the model is **not really able to abstract** and extract these sensorimotor specific patterns in the data. To evaluate this, it is recommended to **test different CNN models** with the here used dataset. If the same phenomenom occurs with other models, it is highly likely that the Physionet Dataset has serious flaws in its recordings. Another good idea is to get a more sophisticated understanding of how to best select the channels. This could be further tested with brute-force channel selections or by more sophisticated approaches like using genetic algorithms, as done in e.g. *"Multi-objective genetic algorithm as channel selection [...]"* [8].

Trials Slicing

Another interesting parameter is the Trials Slicing. In theory, a CNN should benefit from a larger sampling size since it means **a lot more samples to be trained on**, but then each individual sample does not represent a whole EEG Trial anymore. But for the final usage of the system as a realtime BCI, the system should be able to **handle these smaller time windows** as well. The Training process is very static and only trains with 1 sample per 4 second Trial, but as described in section 3.3.8, the live usage looks very different and needs to be more flexible with the constantly new incoming data. In Figure 4.5, you can see that the **loss in accuracy increases** the more slices are used. They are not only marginally lower accuracies, they drop about 10% or more for every n-class classification. Of course the Trials Slicing is directly dependent on the Trial Time Window settings `TMIN` and `TMAX` and the achieved results of these two parameter tests correlate. A smaller Trial Time Window leads to smaller accuracy as shown in Figure 4.6, exactly like with more slices. As already mentioned in Section 5.1.1, this could also be explained by some overall error in the recordings or an structural issue in the EEGNet model.

Trial Time Window

The other important parameters are `TMIN` and `TMAX`, which set the used time window for all Trials. In Figure 4.6 you an see that the longer the interval `[TMIN;TMAX]` is, the higher the accuracy is. This makes sense since it means the model gets a bigger portion of the entire Trial instead of just a small time fragment. But for the live BCI usage of the model this is an important aspect. The smaller the interval is, the faster the response of the system is in realtime.
As described in Section 3.3.8, a sliding time window is used to capture the live incoming data. If this time window is 2 seconds long, it takes the system 2 seconds before it is able to classify what the subject thought in these last 2 seconds. So to make the proposed system more realtime friendly, further optimizations when using smaller time windows have to be implemented and tested. The predictions in the Live Simulation mode have to become more reliable and consistent.

5.2. Outlook

This last section gives an overview of recommended next steps to build a realtime capable BCI using CNNs. There are different aspects of the implementations that can be substituted. At first it is a good idea to further validate the functionality of the here used EEGNet by introducing other models used for classifying EEG data. Many other convolutional model architectures, some also based on the EEGNet, have been proposed, such as the EEGNet Fusion [15] or the CNN model of *"An End-to-end Deep Learning Approach to MI-EEG Signal Classification for BCIs"* [4]. This could clear up the question whether the problems addressed in Section 5.1 are caused by a flaw in the EEGNet model or if it is a problem with the Physionet Dataset.

The implementation also allows to add other datasets to train and test on. A commonly used Dataset for BCI applications validation is the *"BCI Competition IV Dataset 2A"* [19], which is also used in the original proposal paper of the EEGNet [11]. With other datasets the channel selections could also be validated further. The partly confusing results of the 16-channels selections results may also be caused by a bad configuration of the Physionet Dataset recordings. Other datasets use other EEG measuring hardware and setups, with different electrode placements, and could therefore shed some light on the issues at hand.

Talking about Trials Slicing, it is a good idea to introduce overlapping Trials Slicing. This could optimize the Training process to be more similar to the Live Simulation mode with a constantly moving Trial Time Window. This is a critical requirement in order to make the proposed system actually usable in real time scenarios.

Apart from further extending the software, it is also a good idea to get actual EEG recording devices and test the implementation with it. As the EEG Channel Selection results (see Section 4.1.2) already contain a configuration based on the *OpenBCI Starter Kit* [32] (`chs_16_openbci`), this hardware kit might be a good place to start, in order to come closer to an actual ready to use Brain Computer Interface.

6. Bibliography

Books und Journals

[1] M. Abadi et al. “Tensorflow: A system for large-scale machine learning”. In: *12th Symposium on Operating Systems Design and Implementation*. 2016, pp. 265–283 (cit. on p. 15).

[2] H. Allamy and R. Z. Khan. “Methods to Avoid Over-Fitting and Under-Fitting in Supervised Machine Learning (Comparative Study)”. In: Jan. 2014, pp. 163–172 (cit. on p. 19).

[3] A. Burkov. *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019. url: https://books.google.de/books?id=0jbxwQEACAAJ (cit. on pp. 6, 7, 16).

[4] H. Dose et al. “An end-to-end deep learning approach to MI-EEG signal classification for BCIs”. In: *Expert Systems with Applications* 114 (Aug. 2018) (cit. on p. 56).

[5] A. L. Goldberger et al. “PhysioBank, PhysioToolkit, and PhysioNet”. In: *Circulation* 101.23 (June 2000) (cit. on p. 12).

[6] A. Gramfort et al. “MEG and EEG Data Analysis with MNE-Python”. In: *Frontiers in Neuroscience* 7.267 (2013), pp. 1–13 (cit. on p. 22).

[7] C. R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. url: https://doi.org/10.1038/s41586-020-2649-2 (cit. on p. 22).

[8] C.-Y. Kee, S. Ponnambalam, and C.-K. Loo. "Multi-objective genetic algorithm as channel selection method for P300 and motor imagery data set". In: *Neurocomputing* 161 (2015), pp. 120–131. URL: https://www.sciencedirect.com/science/article/pii/S0925231215002295 (cit. on pp. 18, 54).

[9] Y. Kim et al. "Motor Imagery Classification Using Mu and Beta Rhythms of EEG with Strong Uncorrelating Transform Based Complex Common Spatial Patterns". In: *Computational Intelligence and Neuroscience* 2016 (Jan. 2016) (cit. on p. 22).

[10] G. Klem et al. "The ten-twenty electrode system of the International Federation. The International Federation of Clinical Neurophysiology." In: *Electroencephalography and clinical neurophysiology. Supplement* 52 (1999) (cit. on pp. 9, 11).

[11] V. J. Lawhern et al. "EEGNet: a compact convolutional neural network for EEG-based brain–computer interfaces". In: *Journal of Neural Engineering* 15.5 (July 2018), p. 056013. URL: http://dx.doi.org/10.1088/1741-2552/aace8c (cit. on pp. E, 8, 21, 24, 56).

[12] J. Liu, Y. Sheng, and H. Liu. "Corticomuscular Coherence and Its Applications: A Review, Figure 2". In: *Frontiers in Human Neuroscience* 13 (Mar. 2019) (cit. on p. 10).

[13] L. Qin and B. He. "A wavelet-based time–frequency analysis approach for classification of motor imagery for brain–computer interface applications". In: *Journal of Neural Engineering* 2.4 (Aug. 2005), pp. 65–72. URL: https://doi.org/10.1088/1741-2560/2/4/001 (cit. on p. 13).

[14] R. Raina, A. Madhavan, and A. Y. Ng. "Large-Scale Deep Unsupervised Learning Using Graphics Processors". In: ICML '09. Montreal, Quebec, Canada: Association for Computing Machinery, 2009, pp. 873–880. URL: https://doi.org/10.1145/1553374.1553486 (cit. on p. 27).

[15] K. Roots, Y. Muhammad, and N. Muhammad. "Fusion Convolutional Neural Network for Cross-Subject EEG Motor Imagery Classification". In: *Computers* 9.3 (2020). URL: https://www.mdpi.com/2073-431X/9/3/72 (cit. on p. 56).

[16] G. Schalk et al. "BCI2000: a general-purpose brain-computer interface (BCI) system". In: *IEEE Trans Biomed Eng* 51.6 (June 2004), pp. 1034–1043 (cit. on pp. E, 12).

[17] T. Schneider et al. "Q-EEGNet: an Energy-Efficient 8-bit Quantized Parallel EEGNet Implementation for Edge Motor-Imagery Brain–Machine Interfaces". In: *arXiv preprint arXiv:2004.11690*. 2020 (cit. on p. 21).

[18] D. Steinkraus, I. Buck, and P. Y. Simard. "Using GPUs for machine learning algorithms". In: *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*. 2005, 1115–1120 Vol. 2 (cit. on p. 27).

[19] M. Tangermann et al. "Review of the BCI Competition IV". In: *Frontiers in Neuroscience* 6 (2012), p. 55. URL: https://www.frontiersin.org/article/10.3389/fnins.2012.00055 (cit. on p. 56).

[20] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009 (cit. on p. 33).

[21] X. Wang et al. *An Accurate EEGNet-based Motor-Imagery Brain-Computer Interface for Low-Power Edge Computing*. 2020. arXiv: 2004.00077 [eess.SP] (cit. on pp. 17, 24, 42).

[22] J. R. Wolpaw, D. J. McFarland, and T. M. Vaughan. "Brain-computer interface research at the Wadsworth Center". In: *IEEE Transactions on Rehabilitation Engineering* 8.2 (2000), pp. 222–226 (cit. on p. 9).

Internet Sources

[23] M. I. (PerceptiLabs. *Four Common Types of Neural Network Layers*. URL: https://towardsdatascience.com/four-common-types-of-neural-network-layers-c0d3bb2a966c (visited on 04/20/2021) (cit. on p. 5).

[24] J. Brownlee. *How Do Convolutional Layers Work in Deep Learning Neural Networks?* Apr. 2020. URL: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/ (visited on 04/20/2021) (cit. on p. 4).

[25] F. Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras (visited on 04/07/2021) (cit. on p. 15).

[26] *CUDA Toolkit*. URL: https://developer.nvidia.com/cuda-toolkit (visited on 04/21/2021) (cit. on p. 33).

[27] *Github NVIDIA-AI-IOT/torch2trt*. 2021. URL: https://github.com/NVIDIA-AI-IOT/torch2trt (visited on 04/12/2021) (cit. on p. 30).

[28] J. Mohajon. *Confusion Matrix for Your Multi-Class Machine Learning Model*. May 2020. URL: https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826 (visited on 04/19/2021) (cit. on p. 7).

[29] NVIDIA, ed. *CUDA Zone*. URL: https://developer.nvidia.com/cuda-zone (visited on 03/13/2021) (cit. on p. 11).

[30] NVIDIA, ed. *Jetson Nano Developer Kit*. URL: https://developer.nvidia.com/embedded/jetson-nano-developer-kit (visited on 03/13/2021) (cit. on p. 11).

[31] *NVIDIA TensorRT*. 2021. URL: https://developer.nvidia.com/tensorrt (visited on 04/12/2021) (cit. on pp. 30, 33).

[32] *OpenBCI - DIY Neurotechnologist's Starter Kit*. 2021. URL: https://shop.openbci.com/collections/frontpage/products/d-i-y-neurotechnologists-starter-kit?variant=13043169493064 (visited on 04/01/2021) (cit. on pp. 18, 28, 56).

[33] *PyCharm*. URL: https://www.jetbrains.com/pycharm/ (visited on 04/21/2021) (cit. on p. 33).

[34] *PyTorch*. 2020. URL: https://pytorch.org/ (visited on 03/22/2021) (cit. on pp. 15, 21).

[35] ReginaOfTech. *Neural Networks: Basics*. Nov. 2019. URL: https://towardsdatascience.com/neural-networks-basics-29cc093b82be (visited on 05/02/2021) (cit. on p. 4).

[36] M. C. Staff. *EEG (electroencephalogram)*. 2020. URL: https://www.mayoclinic.org/tests-procedures/eeg/about/pac-20393875 (visited on 03/18/2021) (cit. on p. 9).

[37] C. A. (S. University. *Getting Started with EEG Data*. 2015. URL: https://www.cs.colostate.edu/eeg/data/json/doc/tutorial/_build/html/getting_started.html (visited on 04/07/2021) (cit. on p. 10).

A. Appendix

Abbreviations

- **Acc**: Average Accuracy of all 5-Folds (rounded to 2 decimal places)
- **OF**: Average Overfitting of all 5-Folds in percent (rounded to 2 decimal places)
- **Config**: Configuration name (see Figures in chapter 4)
- **BS**: Batch Size
- **BL**: Batch Latency in milliseconds (rounded to 3 decimal places)
- **TPS**: Trials per second (rounded to full number)

Table A.1.: Training Results

	2class		3class		4class	
Config	**Acc**	**OF**	**Acc**	**OF**	**Acc**	**OF**
`defaults`	82.13	-5.53	72.63	-7.70	65.27	-7.23
Channel Selections						
`chs_16`	78.75	-3.15	67.95	-4.39	56.89	-4.13
`chs_16_2`	79.43	-2.59	67.08	-4.27	56.73	-4.29
`chs_16_openbci`	79.29	-3.15	68.46	-5.22	58.01	-4.65
`chs_16_bs`	80.58	-2.95	71.15	-4.20	61.67	-4.45
Trials Slicing						
`2 slices`	71.76	-5.02	56.15	-8.42	46.86	-6.93
`4 slices`	58.78	-3.17	47.49	-5.71	35.10	-5.42
`8 slices`	60.65	-2.57	40.73	-5.27	32.29	-4.87
Trial Time Window						
`[0;1]`	80.79	-4.45	73.09	-7.29	62.33	-4.98
`[0;3]`	82.24	-5.73	74.27	-7.82	65.07	-7.21
`[-0.5;3]`	81.65	-6.42	75.17	-7.95	65.27	-7.39
`[0;4]`	82.53	-6.27	70.50	-8.51	65.03	-8.13
`[-1;5]`	88.54	-5.43	80.98	-5.43	71.03	-8.32

Table A.2.: *defaults* Benchmarking Results

		2class		3class		4class	
Config	**BS**	**BL**	**TPS**	**BL**	**TPS**	**BL**	**TPS**
c_cpu	8	374.070	21	369.657	21	370.391	21
c_gpu	8	3.579	2213	3.560	2239	3.560	2247
c_gpu	16	3.689	4216	3.685	4273	3.697	4286
c_gpu	32	3.854	7782	3.864	8151	3.847	8085
c_gpu_trt_fp32	8	1.262	6277	1.254	6356	1.262	6277
c_gpu_trt_fp32	16	1.310	11869	1.297	12141	1.310	11869
c_gpu_trt_fp32	32	1.368	21924	1.355	23237	1.368	21924
c_gpu_trt_fp16	8	1.490	5316	1.492	5344	1.468	5448
c_gpu_trt_fp16	16	1.528	10179	1.512	10410	1.547	10241
c_gpu_trt_fp16	32	1.581	18984	1.587	19840	1.647	18881

