eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Bachelor's Degree in Informatics Engineering

Computation Engineering

Bachelor Final Project

BCI system for motor imagery classification using convolutional neural networks

Author

Adrián Bécares Cantón

2021

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Bachelor's Degree in Informatics Engineering

Computation Engineering

Bachelor Final Project

BCI system for motor imagery classification using convolutional neural networks

Author

Adrián Bécares Cantón

Supervisor

Ibai Gurrutxaga Goikoetxea

Abstract

The following project aims to analyze the ability of Convolutional Neural Networks (CNNs) to discriminate raw Electroencephalographic (EEG) signals for Brain-computer interfaces (BCI), in order to develop a solid and reliable model that is capable of solving these medical and clinical applications. The project also aims to serve as foundations for future research projects of the UPV/EHU research group *Aldapa*, as well as being a starting framework to apply modern techniques such as Transfer Learning or Semi-supervised Learning.

To achieve this, this report collects and explains the mathematical and theoretical foundations of the architectures and models used for the development, based on the article of [Schirrmeister et al., 2017] and the large EEG database provided by [Kaya et al., 2018]. Following the model implementation, an experimentation is designed and tested, among with an Hyperparameter Optimization setup for the developed model. Finally, the results show that the performance of the model depends on the subject and EEG recording session. It also shows that some hyperparameters influence the model, for example the optimization algorithm, but other hyperparameters barely affect the performance of the implementation.

Contents

List of Figures

List of Tables

1. CHAPTER

Introduction

1.1 Project overview

Brain-Computer interface (BCI) systems are a central component for medical and neurological applications. These BCI systems aim to translate the neural activity in the brain into controllable signals for external devices, and these signals can be extracted using Electroencephalographic (EEG) motor imagery. EEGs are an electrophysiological method to record the electrical activity of the brain and save it digital data, which can be consequently used for extracting information.

Machine Learning techniques allow extracting that information from EEG recordings of brain activity in an end-to-end learning, that is, learning from the raw data. The interest in using these techniques, in particular the use of Deep Learning and Convolutional Neural Networks (CNN) is increasing, but a better understanding of how to design and train CNNs into EEG decoding and how to visualize the informative EEG features is still needed.

The aim of this project is to analyze the ability of these CNNs to discriminate raw EEG data in order to develop a solid and reliable model that is capable of solving modern EEG-based clinical applications. In order to achieve this, the development of the model goes through the optimization of its hyperparameters using modern techniques. An experimentation process is performed to test the performance achieved with the model architecture and its optimal hyperparameters, among with the collection and analysis of the results, that will be summarized in a report. A more detailed analysis of these objectives can be

found in the following section. This project also serves as a general base and reference for future works and research projects of the UPV/EHU research group *Aldapa*, so they can apply more advances techniques such as Transfer Learning or Semi-supervised Learning.

This project is structured as follows: First of all, the main theoretical and mathematical concepts concerning BCI and EEG recordings are explained. Secondly, the model's architecture mathematical basis are analyzed, including the usage of Machine Learning and Deep Learning techniques and the main Neural Network model used for the project: CNNs. Then, the model's decisions are chosen and justified. Next, the model concrete architecture is developed and each element of it is analyzed in detail. Following, the experimentation design is explained, among with other details such as the Hyperparameter Optimization setup and the used hardware and software specifications. Finally, a complete report of the results is attached and the conclusions that emerge from them are detailed.

2. CHAPTER

Project objectives report

The following chapter reports the project in a descriptive way, analyzing the project scope, describing the project objectives, list the working packages and diagrams, the scheduling of the project and the methodologies used for the completion of this work.

2.1 Description and objectives of the project

The general objectives of the project are to analyze the ability of neural networks to discriminate imaginary movements recorded as electroencephalographic (EEG) data for Brain Computer Interface (BCI) [Schirrmeister et al., 2017], given a large dataset of EEG motor imagery dataset [Kaya et al., 2018].

This objective is subdivided into several steps, which are detailed below:

1. **Documentation and research**: in order to complete this project, a lot of documentation and research has to be made to understand the theoretical and mathematical foundations of the models and techniques used, as well as comprehend the type and distribution of the required data. The main documents that is project is based on are EEG for Brain Computer Interfaces (BCI) Machine Learning models [Schirrmeister et al., 2017] and a large EEG database for BCI [Kaya et al., 2018], among with theoretical foundations in Machine Learning and Neural Networks, Deep Learning and Convolutional Neural Networks, Bayesian Optimization and Hyperparameter Tunning.

2. **Develop a complete and robust model**: the development of a model that is capable of compute and extract useful features from the raw data is one of the main objectives of a research project. Different research has to be made from the student in order to develop the programming skills necessary for the correct implementation of a model with the above-mentioned characteristics.

3. **Hyperparameter Optimization**: extract the optimal hyperparameter configurations for the different versions of the dataset, including data from different test subjects, diverse amounts of data subsets and distinct number of classes to discriminate. This optimization must be performed using novel optimization techniques, such as the previously mentioned Bayesian Optimization.

4. **Experimentation**: designing a complete and reliable experiment in order to analyze the behaviour of the model given such situations is another key-point of the developed project, as well as serve as a base for the future work performed by the research group *Aldapa*. This experimentation includes validating the performance for the different subjects, data subsets and hyperparameter configurations.

5. **Results and analysis**: exploring which hyperparameters influence more, discard those that have less weight in the final results, find patterns in the optimal hyperparameters according to the database, which data amount is required or which data subsets are easier to classify meant to facilitate the development of future work on this type of data. Establishing a reference for future work and development of more advanced techniques is the final purpose of this project.

6. **Summarize and final report**: a complete final report must be written in order to achieve the academic purposes of the project, as well as serve as a document available to future researchers.

2.2 Project tasks and scheduling

2.2.1 WBS diagram

Figure 2.1 shows the *Work Breakdown Structure* diagram of the project, which is subdivided in different working packages that will be explained in the following sections.

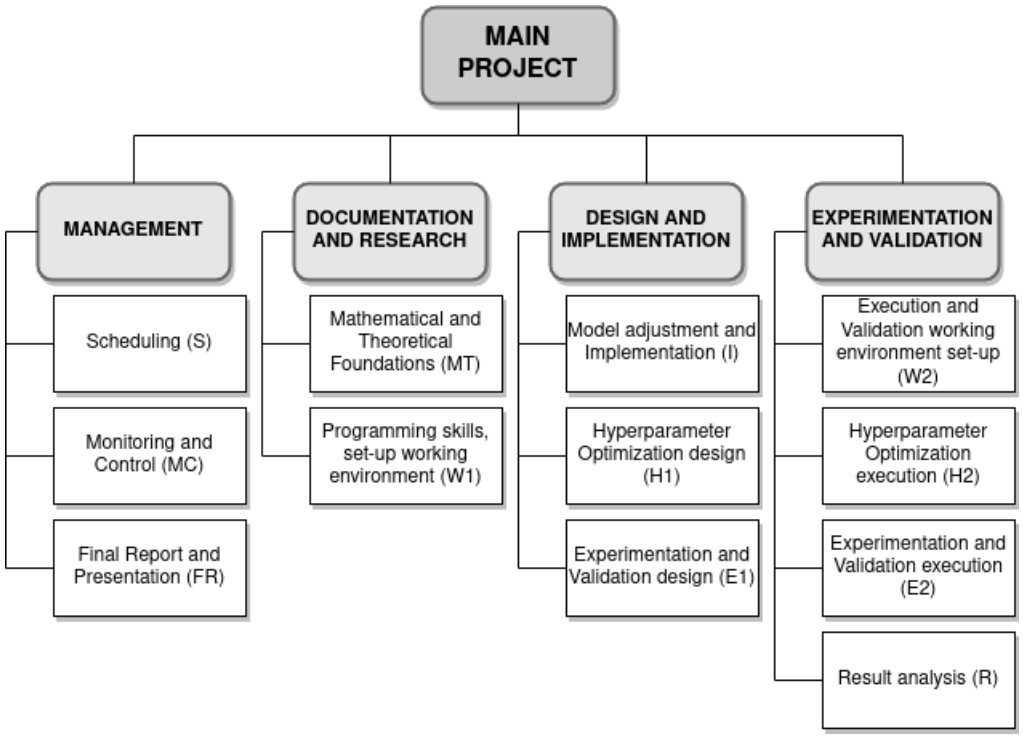

Figure 2.1: *Work Breakdown Structure* (WBS) diagram of the project. The different Work Packages of the project are indicated in brackets.

2.2.2 Work packages

Following the WBS diagram, the different packages are analyzed below.

MANAGEMENT

- **Scheduling (S)**: Project scheduling, including objectives definition, tasks definition and tool selection.
- **Monitoring and Control (MC)**: Established schedule monitoring, date monitoring and deliverables control. Regular meetings with the supervisor.
- **Final Report and Presentation (FR)**: Final report design, writing and error checking, presentation design and trials.

DOCUMENTATION AND RESEARCH

- **Mathematical and Theoretical Foundations (MT)**: Adquire mathematical and theoretical foundations in EEGs, BCI, Machine Learning, Deep Learning, CNNs and Bayesian Optimization.

- **Programming skills, set-up working environment (W1)**: Adquire the programming skills necessary to implement this project and set-up a comfortable working environment and version control repositories.

DESIGN AND IMPLEMENTATION

- **Model Adjustment and Implementation (I)**: Adapt the proposed network model to the large EEG database. Implement the model.
- **Hyperparameter Optimization design (H1)**: Select hyperparameters to optimize, select ranges and possible values of those hyperparameters.
- **Experimentation and Validation design (E1)**: Select subjects and data subsets to test. Design validation process.

EXPERIMENTATION AND VALIDATION

- **Execution and Validation working environment set-up (W2)**: Implement the experimentation and execution framework.
- **Hyperparameter Optimization execution (H2)**: Execute the optimization process and collect best configurations.
- **Experimentation and Validation execution (E2)**: Validate the results using models generated with the best configuration.
- **Result analysis (R)**: Analyze the results and plots of the validation and optimization steps.

2.2.3 Deliverables and milestones

The project has the following deliverables:

- **Implementation**: the working implementation of the project. It is mainly coded in Python and can be accessed in the following GitHub repository.
- **Project report**: the formal document that collects the work that has been performed.

Deliverable	*Milestone date*
Implementation	01/08/2021
Project report	05/09/2021
Project advocacy	13/09/2021 - 17/09/2021

Table 2.1: Deliverables and milestones of the project.

Working Package		**2021**					
		April	*May*	*June*	*July*	*August*	*September*
Management	S						
	MC						
	FR						
Documentation and Research	MT						
	W1						
Design and Implementation	I						
	H1						
	E1						
Experimentation and Validation	W2						
	H2						
	E2						
	R						

Figure 2.2: Gantt diagram of invested time for each working package.

- **Project advocacy**: the live presentation and verbal explanation of the performed work.

The table 2.1 shows the milestones of the described deliverables.

2.2.4 Scheduling and invested time

Table 2.2 shows the time invested in each working package. Figure 2.2 shows the *Gantt* diagram of the invested time in the project.

Notice that the time spent in the *H2* and *E2* packages refers to the activities of setting up the machines were the experimentations were performed and other tasks, and not to the total runtime of the experiments. The total runtime of the experiments can be found in chapter 5.

Work Packages	*Invested Time (hours)*
MANAGEMENT	
Scheduling (S)	10
Monitoring and Control (MC)	25
Final Report and Presentation (FR)	70
DOCUMENTATION AND RESEARCH	
Mathematical and Theoretical Foundations (MT)	15
Programming skills, set-up working environment (W1)	10
DESIGN AND IMPLEMENTATION	
Model adjustment and Implementation (I)	80
Hyperparameter Optimization design (H1)	15
Experimentation and Validation design (E1)	15
EXPERIMENTATION AND VALIDATION	
Execution and Validation working environment set-up (W2)	50
Hyperparameter Optimization execution (H2)	5
Experimentation and Validation execution (E2)	5
Result analysis (R)	15
Total:	315

Table 2.2: Distribution of time spent in each working package in hours.

2.3 Methodology

This project aims to serve as a general base and reference for future works and research projects of the UPV/EHU research group *Aldapa*. The student has received support from one researcher of the group, Ibai Gurrutxaga Goikoetxea, which is in turn the supervisor of the project.

The project has been developed locally using the Python language and stored and controlled using the GitHub repository platform. For the experimentation process the student was granted access to the servers of the home university to decrease the runtime due to the computational needs of the project.

2.3.1 Meetings

The student and the supervisor had regular meetings during the development of the project. There was not a formal scheduling for the meetings due to the personal and professional affairs of both individuals. As a results, the meetings were organised with verbal accor-

dance and attending to the needs of the project. In any case, the meetings were held with a periodicity of 1-2 weeks, with the exception of the August month, that only held one meeting. Most of the meetings were held telematically using the BBC tool, but some meetings were held in a traditional style in the general office of the supervisor.

2.3.2 Workplace

The student has worked mostly in his home due to the Covid-19 pandemic restrictions, among with other personal affairs. The student has also used some university facilities, e.g. the library.

2.3.3 Scheduling

Due to professional student affairs, a formal week scheduling was not created for the development of the project. However, the student mostly worked on afternoons during the first months and mornings during the august month, with a mean working time of 3-4 hours per working day.

2.3.4 Used tools, software and data types

Several tools have been used during the development of the project. Here are listed the most relevant ones:

- The Python general purpose language, among with the above-mentioned libraries are one of the main features that this project is based on.
- The Visual Studio Code IDE has served as the general workplace for the coding process. Other text editors, including the terminal ones (Kate and Nano) have been used in certain moments for a quick updates of the code.
- The data types used during the project include `.mat` files for the raw EEG data and `.json` and `.csv` for the processed data. The reason to use this type of data is the complementation with the Python Pandas [Pandas-Dev-Team, 2020] library.
- Overleaf LaTeX online editting tool has been used for the writing of this report.
- For communications with the supervisor, emails for asynchronous communication and BlackBoard Collaborate tool for synchronous communications have been used.

2.4 Risk analysis

Generally speaking, every project accounts for some risks. This section analyzes the possible risks that this project has and explains a possible prevention and method of operation in case a risk is detected during the development of the project.

2.4.1 Risks

1. **Covid-19 pandemic**: The global pandemic is the main source of uncertainty that this project brings with it. The extension of the ERASMUS+ mobility of the student and the the tightening of deadlines are the main problems that the pandemic may bring.

2. **Extension of the international mobility**: As pointed before, the ERASMUS+ mobility that the student had during the year, starting from September 2020 and finalizing in April 2021 is a source of possible problems during the development of the project.

3. **Novelty of the student**: The student is performing its first mid-size project with the extra difficulties pointed out before, among with the general difficulties a project of this size may have. These general difficulties include the lack of research practice, the implementation errors and the difficulties in drafting the project report in English.

2.4.2 Prevention

1. To tackle the problems arising from the pandemic, the student has scheduled the working period to fit the temporal requirements of the project.

2. The extension of the mobility derived problems can be prevented by starting the project before the returning of the student to its home university.

3. The student has a good communication with its supervisor, in order to solve possible academic problems that may arise during the development of the project.

3. CHAPTER

Theoretical foundations

3.1 Electroencephalographic motor imagery for Brain-computer interfaces

3.1.1 Background

Patients immobilized due to trauma or other medical conditions suffer from a significant deficit of motor and communication functions. Recent advances in neural science may improve the condition of such patients by allowing them to regain control of certain motor and communication abilities. **Brain-computer interfaces (BCI)** aim to translate neural activity in the brain into control signals for external devices, for example, robotic arms or legs. In order to do this, we need a tool to record this information and save it in a digital way, which is what Electroencephalographic motor imagery can achieve.

Electroencephalography (EEG) is an electrophysiological monitoring method to record electrical activity on the scalp that has been shown to represent the activity of the surface layer of the brain. Figure 3.1 shows an example of an EEG signal. It is typically non-invasive, with the electrodes placed along the scalp, but intracranial or invasive methods can also be used. [Schomer and Da Silva, 2012]

Studies aimed into invasive or intracranial BCI presents considerable potential for high degree of freedom control of assistive robots, but research into BCIs that do not need risky brain surgery is also of great importance, and EEG motor imagery for BCI presents

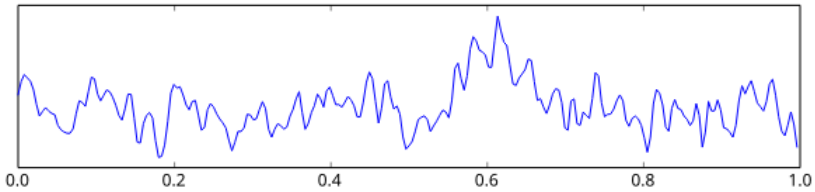

Figure 3.1: An example of 1 second of an EEG recording.

a particularly interesting direction. Important advances in BCI include neural control of robotic devices primarily in humans, but some animals such as monkeys or nonhuman primates can also benefit from it. The key advantages of EEG for BCI are the maturity of the technology, relative easy handling and low costs, as well as the robustness, portability and versatility of recent EEG devices.

3.1.2 A large EEG database for BCI

Development of more effective data processing and analysis methods for EEG BCI has been affected by a lack of large, uniform and accessible datasets. Some EEG for BCI datasets are available on the internet, but most are limited by short recording times, a small number of participants or a small number of BCI signals.

The database that has been chosen for this work is the one developed by [Kaya et al., 2018], in order to supply the above-mentioned lack of large and reachable source. The dataset contains 60 hours of EEG BCI recordings across 75 recording sessions of 13 participants, 60,000 mental imageries, and 4 BCI interaction paradigms, with multiple recording sessions and paradigms of the same individuals. In order to fit the size and purpose of this work, only one paradigm (CLA) has been selected to work with, but more paradigms are available such as HaLT, 5F, and others.

13 individuals between the ages of 20 and 35 participated in the study, all healthy volunteers from students studying in the engineering and science programs, identified only by their aliases. Only 4 subjects had been selected to train the model of this project: *Subject B, Subject C, Subject E* and *Subject F*.

The selected paradigm CLA (Classical) includes a popular EEG BCI interaction model based on three imageries of left and right-hand movements and one passive mental imagery, in which participants remained neutral and engaged in no motor imagery.

The data acquisition and processing procedures were as follows: First, action signals were

presented to participants indicating one of the mental imageries to be implemented (in this case, left and right hand). The imagery was implemented by participants once during the period that that action signal remained on. The EEG signal corresponding to the implemented imagery was recorded by EEG-1200 hardware and saved via Neurofax recording software. After the experiment, the acquired EEG data were saved and exported for further processing and analysis.

Further analysis of the implementation in this specific work will be presented in the following sections.

3.1.3 EEG for BCI using Convolutional Neural Networks

Machine-learning techniques allow extracting information from EEG recordings, and therefore play a crucial role in several important EEG-based research and application areas, such as BCI systems for clinical applications. However, there is still room for considerable improvement with respect to several important aspects of information extraction from the EEG, including its accuracy, interpretability, and usability for online applications. A recent and prominent example of machine learning usage for BCI-EEG tasks is the application of Deep Learning, in particular Convolutional Neural Networks (CNN). Classic solutions to this type of problems have been performed extracting features from the EEG signal "manually" i.e., based on expert judgements. In contrast, Deep Learning techniques allow extracting the most relevant features from (almost) raw signals.

CNNs are artificial neural networks that can learn local patterns in data by using convolutions as their key component, which are analyzed in section 3.2.2.

CNNs have been generally used in computer vision tasks, due to the similarities with nature when analyzing their environment (see section 3.2.2). Therefore, existing CNN architectures are not well suited for EEG analysis, and thus these models need to be adapted and the resulting decoding accuracies rigorously evaluated. For that purpose, a well-defined baseline is crucial, that is, a comparison against an implementation of a standard EEG decoding method validated on published results for that method. This comparison and model creation is introduced by the work of [Schirrmeister et al., 2017], where 3 different CNN models are evaluated, ranging from a "shallow" 2-layer CNN up to a 31-layer deep Residual network [He et al., 2015].

Based on the models presented by [Schirrmeister et al., 2017], this project aims in using

the "deep" type of network. The model decisions and other aspects of the implementation can be obtained in section 4.

3.2 Artificial Intelligence, Machine Learning, Deep Learning and Convolutional Neural Networks

3.2.1 Artificial Intelligence and Machine Learning

Artificial intelligence (AI) is intelligence exhibited by machines. Colloquially, this term is associated to machines that mimic the cognitive functions that humans link to skills that other human minds can achieve, such as learning and solving problems. In computer science, it refers to the study of "intelligent agents", which stands for any device that perceives its environment (in form of digital data) and takes actions to maximize its chance of success at some goal. [Russell and Norvig, 2002]

Machine Learning (ML) is the subfield of computer science that gives computers the ability to learn without being explicitly programmed. This subfield evolves from the study of pattern recognition and computational learning theory (other subfields of artificial intelligence). The main basis of this field is the study and development of algorithms that, given some data, can make predictions or decisions, thus building a model that can be further used. [Muñoz-Villamizar et al., 2020]

Machine Learning subdivides into two main fields [Davenport, 2018]: supervised learning and unsupervised learning, among with other fields, for example, semi-supervised learning or reinforcement learning:

- Supervised learning algorithms are used when the desired output is known. For example, we have a system composed by three variables (x_1, x_2, x_3), and an output variable y. The learning algorithm learns, given a dataset of combinations of the (x_1, x_2, x_3) variables and its correct $\hat{y}$ output, comparing its actual output with the correct one, and then modifying the model accordingly. Through methods like classification or regression, supervised learning uses patterns to predict the values of the label on additional unlabeled data.
- Unsupervised learning is used against data that has no historical labels. The system is not told the "right answer". The algorithm must figure out what is being shown.

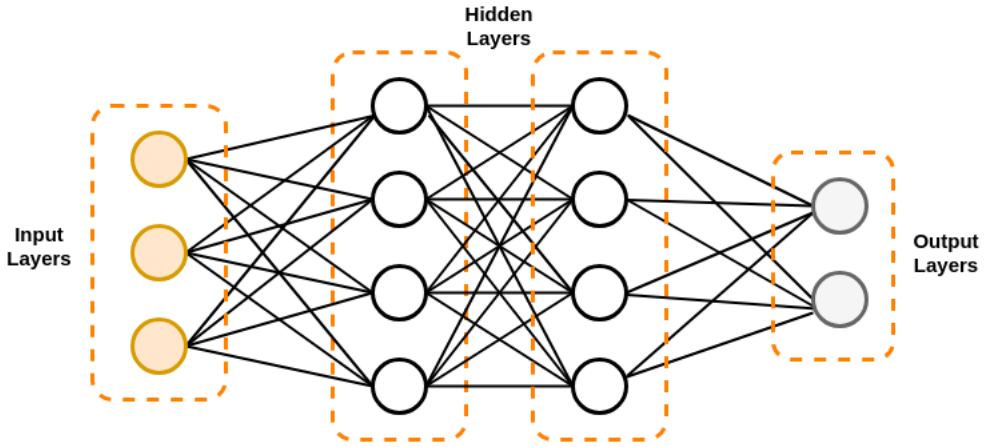

Figure 3.2: A generic neural network example, which includes some hidden layers between the input and output layers.

The goal is to explore the data and find some structure within. This structure needs to be further analyzed by an expert, that will help in the understanding of the obtained structure. A common example of unsupervised learning is clustering, where some samples need to be grouped and labeled by its similarities in one class, while being sufficiently different from samples of other classes.

3.2.2 Deep Learning and Convolutional Neural Networks

An **artificial neural network (NN)** is a type of algorithm inspired by the biological neural networks that constitute animal brains. These networks are collections of connected nodes that transmit real numbers between each others. This nodes receive a real number, perform some non-linear mathematical computations and output another real number, that is transmitted to the next connected neuron. Each node has typically both a weight that adjust the computations that are made and an activation function that is computed before the signal is transmitted to the next neuron. Neurons are also aggregated into layers. The signals travel from the first layer (input layer) to the last layer (output layer) potentially traversing multiple layers in between. [Ongsulee, 2017]

Deep Learning studies Machine Learning models (mostly artificial neural networks) that contain more than one hidden layer. These models use a cascade of many layers of non-linear transformations to extract features from them. Each successive layer uses the output from the previous one as input, and can be both supervised (generally in the output layer) and unsupervised (the rest of the layers). The more layers a deep model has, the higher

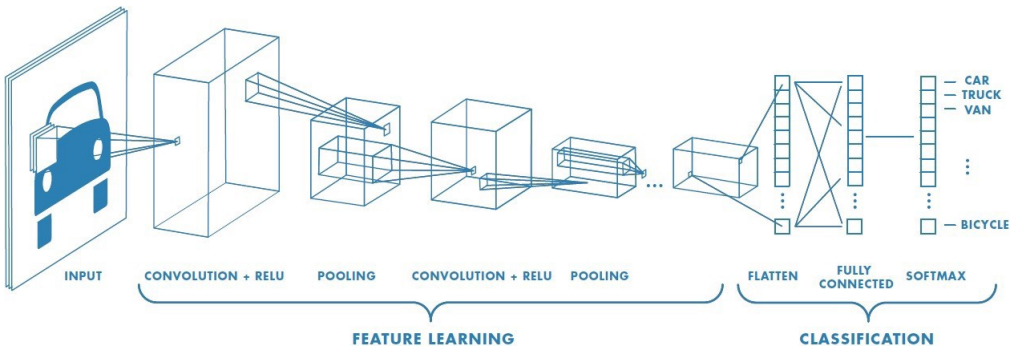

Figure 3.3: A generic diagram of a CNN, where we have an image of a car as input. The convolution and pooling layers are iteratively structured, until a flatten + fully connected layer is used, and finally using a softmax layer that classifies the image as car.

level of features are derived, which correspond with higher levels of abstraction, and thus creating a hierarchy of concepts. The most known deep learning architectures are **Deep Neural Networks (DNN)**, which have demonstrated an out-performing of other machine learning techniques on task such as computer vision, signal analysis, and other related topics such as medical imagery or natural language processing. [Deng and Yu, 2014]

Convolutional Neural Networks (CNN) are a type of feed-forward deep artificial neural network in which its connectivity pattern is inspired in the animal visual cortex, where each cortical neuron respond to stimuli in a restricted region of the space, known as receptive field. Then these regions overlap to cover the entire visual field. This stimuli can be mathematically approximated by a convolution operation. These type of networks are great to detect hierarchical structures, such as the simple shapes that form more complex structures in an image. [Fukushima et al., 1983]

A CNN consist of multiple layers of receptive fields, each one consisting on small neurons collections. The outputs of these collections are then tiled so that their input regions overlap, and thus obtaining a higher resolution representation of the original input. CNNs may include local or global pooling layers, which combine the output of these regions. They may also consist of combinations of convolutional and fully-connected layers, known for their prone to overfitting data. They are also based on a shared-weight architecture, which makes them shift invariant. Combining all these features we obtain a model that prevents overfitting and also makes them far less complex than other types of DNNs.

Figure 3.3 shows the building blocks of a CNN:

- The convolutional layer is the core building block of a CNN. This layer consists of a set of learnable filters or kernels that have a small receptive field, but extend

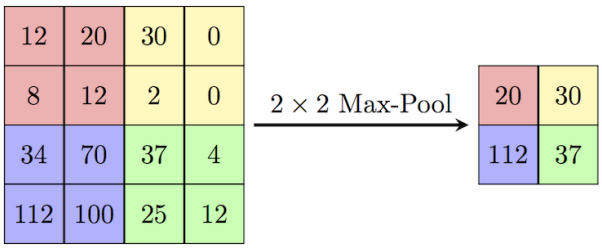

Figure 3.4: A 2×2 max-pooling example, where a 4×4 grid is subdivided in 4 grids and then a max operation is performed in each grid.

through the full depth of the input volume. Each filter is convolved across the width and height of the input volume, resulting in a 2-dimensional activation map of that filter. This means that the network learns filters that activate when they detect some specific type of feature at some spatial position in the input. [Géron, 2019]

- Another important layer of a CNN is the pooling, which is a form of down-sampling. The most common function that implements this layer is the max-pooling, as shown in Figure 3.4. It partitions the input image into a set of rectangles and, for each sub-section, outputs the maximum value.

- A ReLU function is the non-saturating activation function $f(x) = \max(0,x)$ which effectively removes the negative values from the activation map by setting them to zero. [Krizhevsky et al., 2017] Another similar function is the ELU function, which has negative values but smoothly traverses to the positive values of ReLU. [Trottier et al., 2017]

- After several combinations of convolutional + ReLU and max-pooling layers, the classifications is done via fully connected layers. These layers have each neuron output connected to every neuron input on the next layer, as general non-convolutional NNs.

- Finally, the softmax loss function is used to predict a single class of K mutually exclusive classes, which effectively selects the final output of the model.

3.3 Bayesian Optimization for Hyperparameter Tuning

Hyperparameters are a key piece of any Machine Learning (ML) model, as they directly control the behaviour of an algorithm and affect significantly to the performance of

these models. Usually, finding the best hyperparameters for a model requires professional knowledge and expert views, or even sometimes it has to depend on brute-force search. Therefore, if an efficient hyperparameter optimization algorithm is developed to optimize any kind of ML algorithm, it could greatly improve its performance.

In contrast to traditional ML models, Deep Neural Networks (DNNs) are known to be specially sensitive to the choice of its hyperparameters. These hyperparameters do not refer to the internal model parameters, such as a NN weights (those can be learned during the model training phase), but to the higher level parameters of a model. For example, in the case of a DNN, an important hyperparameter would be the number of layers it has, or its learning rate. [Cho et al., 2020]

The process of finding the optimal hyperparameters of a ML model is called **hyperparameter optimization or tuning**. [Wu et al., 2019] There are two main kinds of hyperparameter tuning methods: manual search and automatic search.

- Manual search tries to optimize these hyperparameters by hand. It depends on the intuition and previous knowledge of an expert user who can identify the vital hyperparameters that have a greater impact on the performance of the model. It is also extremely hard to apply by non-expert users. Besides, as the number of hyperparameters and its range grows, it becomes quite difficult to humans to manage and handle high dimensional data, thus it becomes easy to misinterpret the relationships or trends between hyperparameters.

- To overcome the drawbacks of manual hyperparameter tunning, automatic methods have been proposed. One type of these methods are exhaustive methods, for example, grid-search (training a model for each combination of hyperparameters possible and finding the optimal combination), but these methods suffer from the curse of dimensionality [Köppen, 2000], i.e., the efficiency of an algorithm decreases rapidly as the number of hyperparameters being tuned and its range of values increases. Another less expensive automatic method is random search, but, as shown in [Bergstra and Bengio, 2012], this method is unreliable for training some complex models.

Therefore, having a high precision and high efficiency algorithm for hyperparameter tuning has always been a problem that is not fully solved in Machine Learning.

3.3.1 Bayesian Optimization

The method proposed in [Wu et al., 2019] to develop and improve automatic hyperparameter optimization, and the method that is implemented in this work is called **Bayesian Optimization (BO)**.

Traditional optimization techniques like Newton method or gradient descent cannot be applied due to the non-differentiable nature of these optimizations (most of the times, the cost function of this kind of optimization problems are not representable or cannot be differentiated).

BO is a very effective tool for solving functions that are computationally expensive to find the extreme points. It can be applied for solving a cost function which does not have a closed-form expression. Treating this cost function as a kind of a black-box that outputs some values, we can combine prior information of it to obtain some posterior information using the Bayesian Theorem. [Swinburne, 2004]

Algorithm 1: Bayesian Optimization

begin

  for $t = 1, 2, \ldots$ **do**

    Find x_t by optimizing the acquisition function u over function f:

$$x_t = \arg\max_x u(x|D_{1:t-1}).$$

    Sample the objective function: $y_t = f(x_t)$.

    Augment the data $D_{1:t} = \{D_{1:t-1}, (x_t, y_t)\}$ and update the posterior of the function f with a Gaussian Process.

  end

The principal basis on Bayesian Optimization and the complete algorithm is shown in Algorithm 1:

1. The optimization goal is to find the maximum value at the sampling point x for an unknown function f:

$$x^* = \arg\max_{x \in A} f(x) \equiv \arg\max_x u(x|D)$$

   where u is an adquisition function 3.3.1, D represents the dataset of previous observations, i.e., the prior information, and A represents the search-space of x.

2. Then, for each iteration-step t, we compute new suboptimal values using the optimization goal of above:

$$x_t = \arg\max_x u(x|D_{1:t-1}).$$

3. Now, we sample our objective function or black-box function $y_t = f(x_t)$, in order to obtain a new value that we can use to expand our data and finally, update the posterior distribution of the function f with a Gaussian Process.

$$D_{1:t} = \{D_{1:t-1}, (x_t, y_t)\}$$

Figure 3.5 shows in an intuitive way how the algorithm performs, as we can see how successively the current sampling point approaches the optimum value of the black-box function.

A popular implementation of a Bayesian Optimization algorithm is **Tree-structured Parzen Estimator (TPE)**, [Ghanbari-Adivi and Mosleh, 2019] which is the model implemented for this project. More information can be found in chapter 4.

Finally, BO needs an **adquisition function** u [Wilson et al., 2018] to derive the maximun of the function f. Normally, we assume that a high value of the adquisition function corresponds to a large value of the black-box function f. This are some common adquisition functions that are largely used for BO problems:

- Expected Improvement (EI) calculates the expectation of the degree of improvement that a point can achieve when exploring the vicinity of the current optimum value, i.e., calculates how will a near point improve the current optimum value, and chooses that point based on this expectation degree.
- Upper Confidence Bound (GP-UCB) determines whether the next sampling point should make use of the current optimum (a point near our current best), or explore other zones of lower confidence.

3.3.2 Gaussian Process

Now that the general algorithm is clarified, the mathematical tool that this method used is explained as follows. A **Gaussian Process (GP)** [Williams and Rasmussen, 2006] is

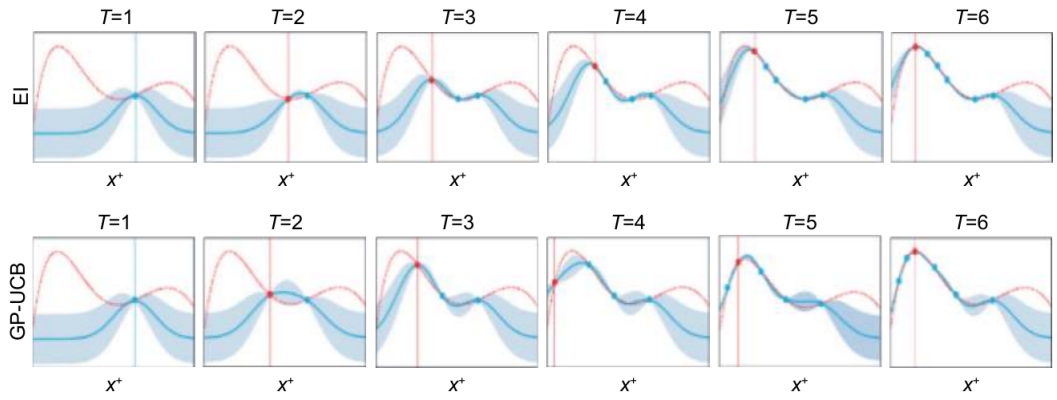

Figure 3.5: Bayesian Optimization workflow. Each rectangle show one iteration-step of the algorithm. The blue line represents our current knowledge of the black-box function, modelated via a Gaussian Process. The red line is the real black-box function shape, and the red vertical line and the red dot represents our current optimal sample. Each row represents how different adquisition functions 3.3.1 perform when optimizing a black-box function.

a kind of technique developed in the basis of Gaussian stochastic process and Bayesian learning theory.

A stochastic process is a *"type"* of probability distribution that, whereas a probability distribution describes random variables or vectors, a stochastic process governs the properties of functions. This means that, for any sub-collection of random variables in this stochastic process, it has a multivariate Gaussian distribution. Figure 3.6 shows how a Gaussian Process may look like.

As any other gaussian distribution, a Gaussian Process is defined by a mean function $m : x \to R$, normally assumed that $m(x) = 0$, and a covariance function $k : x \times x \to R$. A popular choice for the covariance function is the exponential square function:

$$k(x_i, x_j) = \exp(-\tfrac{1}{2} \left\| x_i - x_j \right\|^2)$$

Finally, we formally denote a Gaussian Process as:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x'))$$

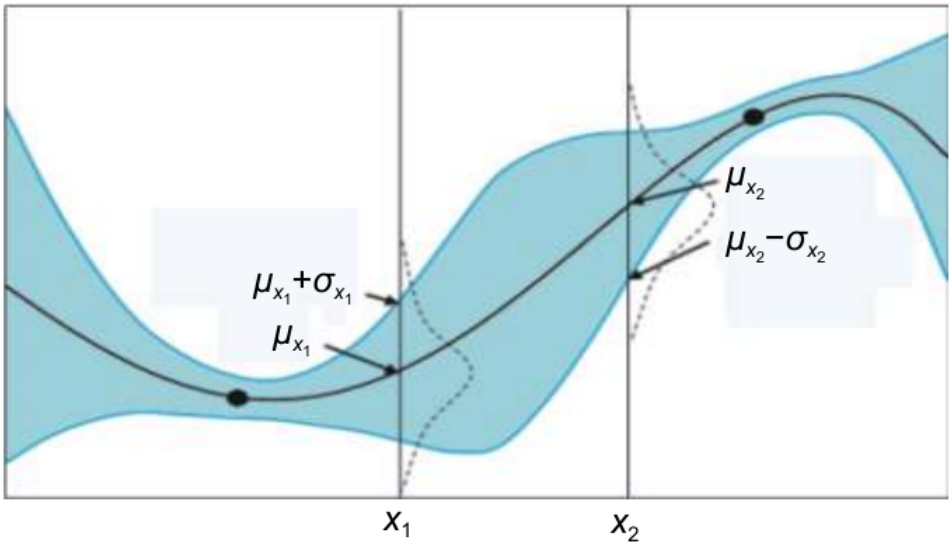

Figure 3.6: A one-dimensional Gaussian Process. Each time we sample a Gaussian Process it does not return a scalar value, but instead returns a gaussian probability distribution, denoted with its own mean μ and variance σ. The black dots indicate the observations of data points D we currently have in out BO problem. The black curve is the predicted mean of the objective function of a BO problem. The blue area represents the range of the standard deviation near the mean. As it can be seen, this area is small when is close to an observation, and large when it is away from it.

4. CHAPTER

Implementation of the model

This section aims to explain the process of adapting the proposed database to the final implemented work, as well as elaborate the design choices made for the implementation of the Neural Network and describe the model used for the experimentation.

4.1 Dataset analysis, extraction and preprocessing

The proposed database in the work [Kaya et al., 2018] is composed of 75 data files, each containing the complete data record of one BCI recording session plus one text description file. Each recording session contains 45 minutes of BCI imagery data, and consists of approximately 900 mental imagery symbols. Each session is performed for one participant and uses one interaction paradigm, identified via a systematic naming convention.

For example, the filename `CLASubjectB1510193StLRHand.mat` indicates the recording session of subject B that took place on October 15th, 2019, with paradigm CLA consisting of 3 states (3st), left- and right-hand movements with a passive state. The recording session mnemonic is Left-Right Hand (LRHand).

The only interaction paradigm used for this project is paradigm CLA, which consists of 3 mental imagery states: left-hand, right-hand and a passive state. Other interaction paradigms have been discarded due to the early nature of this work.

Participants focused a fixation point in the center of the eGUI (see Figure 4.1) screen.

Each visual action signal was shown for 1 second during which time the participant implemented the corresponding mental imagery once. The left- and right-hand motor imageries were implemented by imagining closing and opening the respective fist once. After that, the action stimulus disappeared and a random duration off-time of 1.5-2.5 seconds followed, concluding one trial. During this random duration, the participant remained passive and did not engage in any voluntary mental imagery until the beginning of the next trial. These trials were repeated about 300 times, resulting in a total segment duration of 15 minutes. Participants relaxed for 2-3 minutes, and continuing starting a new segment. Each recording session finally contains 3 of these segments, making a total of 45 minutes of BCI interaction time per recording session.

All data files are shared in `.mat` files and contain MATLAB-readable records of the raw EEG data and the recording session's interaction record. Each data file is represented as a MATLAB structure named "o", with 5 key fields (see Table 4.1).

The fields of the record "o" comprising the data are as follows. The main fields are "marker" and "data", which contain the recording session's interaction record and the EEG raw data, respectively. The fields "nS" and "sampFreq" contain the total number of EEG signal samples and the sampling rate expressed in Hz. The sampling rate used in this project is 200Hz, but a 1000Hz option is available for some sessions, referred in the data file by the identifier HFREQ (high frequency). In any case, this project only focuses on the 200Hz recordings.

The "data" field is a 2D MATLAB array of size $nS \times 22$, where each column is a time-series of voltage (in μV) measurements from a single EEG input lead. The ordering of this lead is described in a `description.txt` file and is the same for the entire database.

The "marker" field contains the recording session's interaction record. This record is a 1D MATLAB array of size $nS \times 1$, with integer values from 0 to 99. Each value encodes the state eGUI at the time mapping to the corresponding EEG data sample in the "data" array at the same index location. The marker codes from 1 to 6 to encode visual stimuli directing the participants to implement the given mental imageries in the order 1: "left hand", 2: "right hand", 3: "passive". The other codes have been omitted due to its non-appearance in the CLA paradigm. Codes greater than 10 indicate service periods, such as, 99: "initial relaxation", 91: "intersession breaks", 92: "experiment end". Code 0 means "blank" or nothing is displayed in the eGUI. The record codes are summarized in Table 4.2.

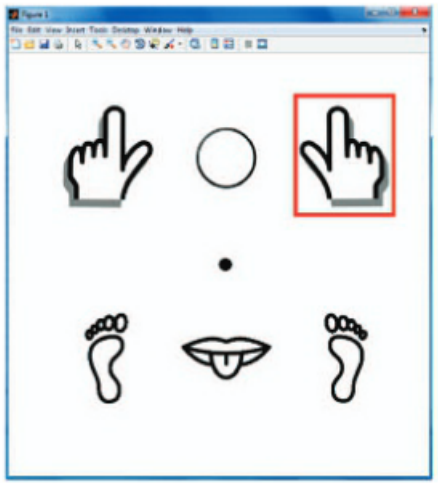

Figure 4.1: An example of the graphical user interfaces (eGUI) used for BCI interactions. For CLA (and other) interaction paradigms, the eGUI displayed six icons symbolizing left hand, right hand, left leg, right leg and tongue motor imageries together with a passive imagery indicated by a symbol. A fixation point in the center of the screen is also shown. The action to be implemented is represented as a red rectangle around the respective icon.

Data Record Keys	
id	A unique alphanumeric identifier of the record
nS	Number of EEG data samples
sampFreq	Sampling frequency of the EEG data
marker	The eGUI record codes
data	The raw EEG data

Table 4.1: Data Record Keys used in MATLAB object "o".

"marker" code	**Meaning**
1	Left hand
2	Right hand
3	Passive/neutral
91	Session break
92	Experiment end
99	Initial relaxation
0	Nothing

Table 4.2: Explanation of the numerical codes used in recording session interaction records.

4.1.1 EEG signal extraction and preprocessing

This data is then extracted and loaded to the model using Python. The data is loaded using the Python library SciPy [Virtanen et al., 2020]. Another key library is then used: MNE [Gramfort et al., 2013]. This package is meant to explore, visualize and analyze human neurophysiological data, for example, EEGs. Loading and processing the "o" object results in a MNE Raw object, containing the channel names, number of time points, the time points and additional information of the EEG signal.

Similar to what is done in [Schirrmeister et al., 2017], the sync channel is then dropped and the signals are converted from μV to V, while adding a band-pass filter between $0.1Hz$ and $40Hz$. Next, the signal is standarized for each trial, subtracting the mean of the full signal and dividing for the standard deviation.

Finally, the events of the signal are found and attached along with the active event id's (the "marker" codes) into a processed MNE object. In the creation of this object the time between events must be specified, in this case, 1 second (the duration of the mental imagery implementation). The final object will be processed by the neural network, at the time of performing cropped training (see subsection 4.2.1).

4.2 Model design choices and hyperparameters

The Neural Network model implemented for this project is based on the architecture proposed by [Schirrmeister et al., 2017], using both the proposed methods used in the cited work and some other methods that are also mentioned but not implemented in it.

4.2.1 Design choices

The general architecture of the NN model is based on a Convolutional Neural Network of the "deep" type mentioned in [Schirrmeister et al., 2017]. Several architectural choices were evaluated. The general idea of the model is to implement a first "special" convolutional block, whose aim is to handle EEG inputs, followed by some (or none) standard convolutional max-pooling blocks and adding some final dense softmax classification layers.

The first "special" convolutional is a split into two layers of a normal convolution layer, in order to better handle the large number of input channels (one per electrode, in contrast

to the common 3 input channels of RGB-images). In the first layer, a convolution over time is performed, and in the second layer, a spatial filtering with weights for all possible pairs of electrodes with filters of the preceding temporal convolution is performed. Note that as there is no activation function in between the two layers, they could in principle be combined into one layer. However, using two layers implicitly regularizes the overall convolution by forcing a separation of the linear transformation into a combination of a temporal convolution and a spatial filter.

Other features were also evaluated:

- The usage of batch normalization. This is meant to facilitate the optimization by keeping the inputs of the layers to a normal distribution during training, standardising intermediate outputs to zero mean and unit variance for a batch of training examples.
- Dropout randomly some sets of neurons, preventing the model to overfit the data and generalize better.
- Reduction of the number of neurons in the fully-connected layer blocks, and how strongly this reduction should be applied.
- The loss function optimization algorithm. Gradient descent based algorithms were chosen to evaluate, such as Stochastic Gradient Descent (SGD) or Adaptive Moment Estimation (Adam). [Kingma and Ba, 2017]
- Early-stopping techniques, allowing the model to stop training when certain metrics do not improve in successive iterations.
- Activation function of the neurons. Two activation functions are evaluated, the popular ReLU function or the novel ELU function, sensitive to the type of data that is used in this project.
- The usage of temporal filtering, aimed to remove or attenuate frequencies within the raw signal that are not of interest or do not influence the model at all.

Another key component of this model is the usage of **cropped training**. This strategy is focused on using crops, that is, sliding an input window withing the trial, which leads to many more training examples for the network that the trial-wise training strategy.

Before applying the cropped training, the data is filtered extracting only the relevant seconds of each session i.e., the 1.5-2.5 seconds of passive imagery is discarded and only the mental imagery implementation of 1 second is kept.

Two parameters control this cropping, called window (or crop) size and step size. The window size represents a portion of each trial. It can be seen as a moving window across the trial, and how much that window moves is represented by the step size parameter. The crop size range selected for the model is of 0.5 seconds (window size of 50) or 1 second (window size of 100), meaning that the trials are divided into a bunch of piece-trials, depending on the step size, or the full trial is preserved. The size between each crop (step size) is ranged from 10% to 100% of the window size. To determine the number of crops that will be made for one trial is computed using the following equation:

$$\text{Number of crops per trial} = \left\lfloor \frac{\text{trial size} - \text{window size}}{\text{window size} \times \text{step size}} + 1 \right\rfloor$$

Figure 4.2 shows the summarized process of the data extraction and cropped training, as well as a clarification of each partition of the data for better understanding.

Finally, a general outline of the model layers and blocks is described in figure 4.3. The model architecture is variable, meaning that some hyperparameters directly control the number of blocks, layers and features that the model has. A further explanation of the hyperparameters of the model can be found in the next subsection.

4.2.2 Hyperparameters of the model

Using the above-mentioned design choices, a total of 13 hyperparameters have been implemented in the model. These hyperparameters will then be optimized via hyperparameter optimization algorithms (see section 4.3). The selected hyperparameters are: *window_size* and *step_size*, related to cropped training; *kernel_size*, to control the size of the convolution kernel used in convolutional blocks; *pool_size*, to select the size of the max-pooling algorithm applied; *learning_rate*, an essential hyperparameter in every NN model; *temporal_filters*, to optimize which frequencies are less relevant to the model, *optimizer*, the optimization algorithm; *activation*, the activation function; *batch_normalization*, to select whether this technique is applied or not; *n_conv_layers*, to optimize the number of convolutional blocks used in the model; *n_fc_layers*, same as previous hyperparameter, but with fully-connected blocks; *n_neurons_2nd_layer*, to select the percentage of reduction in the successive fully-connected layers and, finally, *dropout_rate*, to drop certain sets of neurons to avoid overfitting. These hyperparameters are summarized in Table 4.3

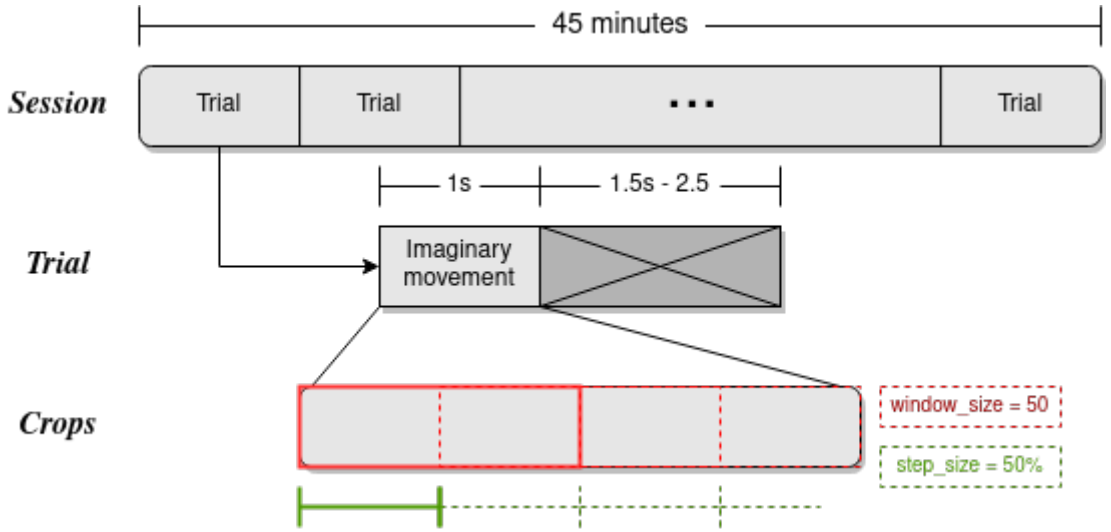

Figure 4.2: A diagram summarizing the processing of the data. First, the full session is subdivided into its trials. Each trial is then filtered, discarding the passive states (represented as gray crossed boxes) and collecting only the active implementation imageries. Finally, the cropped training is performed, where the window size is represented by the red dashes and the step size is represented by the green lines and dashes.

Hyperparameter	Description
window_size	The window crop size that will be used for the cropped training.
*step_size**	The step between the different crops. *This hyperparameter is not really a hyperparameter for the CNN, but used instead in the process of cropping the data described in this section.
kernel_size	Kernel size of the convolution.
pool_size	The max-pooling dimension (2×2 or 3×3).
learning_rate	Learning rate value of the model.
temporal_filters	The number of filters that are applied during the temporal convolution.
optimizer	Optimizer of the model.
activation	Activation function of the model.
batch_normalization	Decide to apply batch normalization or not to the different blocks of the model.
n_conv_layers	Number of additional convolutional blocks.
n_fc_layers	Number of Fully-Connected layer blocks.
n_neurons_2nd_layer	Reduction of neurons in the subsequent Fully-Connected layers (including the first one, if exists).
dropout_rate	Percentage of neurons to be dropped in the fully-connected layers.

Table 4.3: List of hyperparameters of the implemented model.

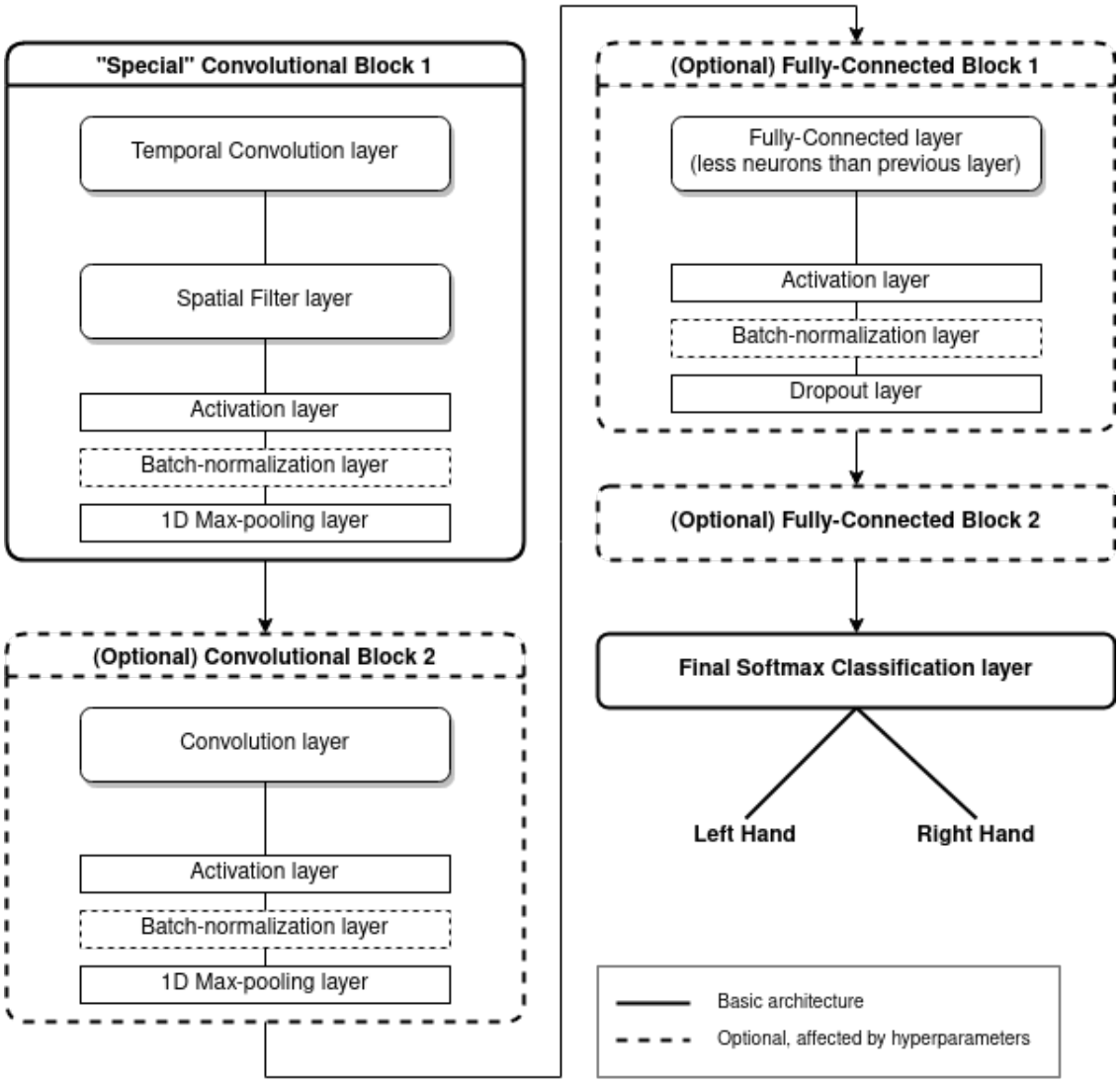

Figure 4.3: General outline of the architecture of the model. The architecture is composed of a "special" convolution block, which splits the convolution into two layers. Next, an optional second convolutional block is added, without the convolution split. Further, 1 or 2 optional FC layers are included, with the final softmax classification layer to classify between left or right hand. Optional layers and blocks depend on its corresponding hyperparameters, for example, the optional convolutional block will appear or not depending on *n_conv_layers* hyperparameter, and the number of FC layer blocks depends on *n_fc_layers* hyperparameter. More information about the hyperparameter ranges can be found in section 5.1.

4.3 Hyperparameter optimization framework setup

The Keras [Chollet et al., 2015] Python library was mainly used to implement the described model, among with other common libraries such as NumPy [Harris et al., 2020], the previously mentioned SciPy [Virtanen et al., 2020], the plotting library Matplotlib [Hunter, 2007] and the data analysis and manipulation tool Pandas [Pandas-Dev-Team, 2020].

To implement the hyperparameter optimization phase, a scalable and distributed execution framework is used: Ray-project [Moritz et al., 2017]. Attached to this framework, an experiment execution library for hyperparameter tunning is used: Tune [Liaw et al., 2018]. This framework and libraries allows to implement easily an optimization benchmark out of the box, including support for most common Deep Learning libraries (in this case, Keras) and state of the art optimization algorithms, for example, Bayesian Optimization.

The selected setup for the optimization of the hyperparameters mentioned in section 4.2.2 consists of a set of about 250 iterations, where each one is evaluated and the best hyperparameter configuration is chosen among all of them. The first set of hyperparameters is uniformly chosen between the bounds detailed in Table 4.3. Successive hyperparameter configurations are optimized using the Tree-based Parzen Estimation (TPE) algorithm, based on Bayesian Optimization and implemented using the Hyperopt library [Bergstra et al., 2015]. The metric used to estimate which configuration is considered the best one is the mean validation accuracy over a 10-fold cross-validation for each model generated with each set of hyperparameters. Each fold creates a model and fits it, outputting an accuracy in the process. This accuracy is validated using the 20% of each fold. When all 10 folds are completed, the mean accuracy of those folds is computed and the resulting value is the accuracy of the model. A further explanation on the experiments performed with the model can be found in section 5.

5. CHAPTER

Experimentation design

The following sections explains the design of the performed experiments, including the test subjects selected for the experimentation, the hyperparameter ranges whose performance is tested, the employed hardware specifications, the different dataset sizes used to test the performance of the model, the total collected data and the final survey style validation step.

5.1 Hyperparameter ranges and choices

As explained in the previous chapter, the hyperparameter ranges and choices were selected as follows:

- *window_size*: a choice between 50 and 100 was selected. This means that the model will perform the cropped training with half a second or a full second, respectively.
- *step_size*: this parameter ranges from 0.1 to 1 in intervals of 0.1, meaning that the difference between windows can vary between a 10% of the *window_size* to its complete size, in 10% intervals.
- *kernel_size*: a common choice among kernel sizes are odd numbers. Therefore, the set with the 4 first odd numbers (3, 5, 7, 9) excluding 1 was selected as the parameter range.

- *pool_size*: some popular choices for max-pooling sizes are 2×2 or 3×3. Therefore, it was decided to test these 2 strides, referring them as a choice of 2 or 3.
- *optimizer*: as previously mentioned, the two loss function optimization algorithms selected for this project are Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (Adam).
- *activation*: similar to the previous hyperparameter, the mentioned two activation functions were tested: ReLU and ELU functions.
- *learning_rate*: a logarithmic continuous range between 10^{-5} and 10^{-2} was selected as the possible values of the learning rate of the model.
- *temporal_filters*: the number of filters that are applied during convolution has a range between 10 and 50 filters, in intervals of 10.
- *batch_normalization*: the choice between using or not batch normalization is stated as this parameter range.
- *n_conv_layer*: the number of additional convolutional blocks is controlled by this hyperparameter.
- *n_fc_layers*: in the same way, the number of fully-connected blocks is controlled by this parameter, ranging from no FC blocks, 1 additional block or 2 additional blocks.
- *n_neurons_2nd_layer*: the percentage of neurons that consecutive FC blocks have is controlled by this hyperparameter, ranging from a 20% of the previous neurons up to a 75% of the previous number of neurons, in intervals of 5%.
- *dropout_rate*: a choice between 0 or 0.6 dropout rate is selected for this hyperparameter.

A summarized version of these intervals, choices and ranges can be found in table 5.1

5.2 Employed hardware specifications

The employed machine to perform the experimentation is a GNU/Linux distribution based OS, with an Intel(R) Xeon(R) Gold 5120 @2.20GHz CPU with 56 cores and 2 threads per

Hyperparameter	Range, Interval or Choices
window_size	{50, 100}
step_size	[0.1, 0.2, ..., 1]
kernel_size	{3, 5, 7, 9}
pool_size	{2, 3}
learning_rate	$[10^{-5}, 10^{-2}]$
temporal_filters	[10, 20, ..., 50]
optimizer	{Adam, SGD}
activation	{ReLU, ELU}
batch_normalization	{Yes, No}
n_conv_layers	{0, 1}
n_fc_layers	{0, 1, 2}
n_neurons_2nd_layer	[0.2, 0.25, ..., 0.75]
dropout_rate	{0, 0.6}

Table 5.1: List of hyperparameters with their ranges, intervals or choices. The square brackets stand for ranges and intervals, and the braces stand for sets of choices.

core and 62 GB of RAM. A total of 5 GPUs were used for the experimentation, 4 for the main experiments computation and 1 for the final validation step (see Section 5.4). The first 4 GPUs are NVIDIA(R) Tesla(R) P40, with 3840 CUDA cores and 24 GB GDDR5 memory size. The last GPU to perform the validation step is a NVIDIA(R) GeForce GTX TITAN Black, with 2880 CUDA cores and 6 GB GDDR5 memory size.

5.3 Selected test subjects, dataset samples and collected data

5.3.1 Selected subjects, data samples and cross-validation

From a total of 13 participants, 7 test subjects were selected for the CLA paradigm BCI interaction mental imagery collection. Not all of these test subjects had 3 complete mental imagery sessions. Therefore, the first 4 test subjects that had 3 BCI sessions completed were selected as the subjects used for this project: *Subject B, Subject C, Subject E* and *Subject F*.

For each subject, 2 of its sessions were used as training data for the model. The last session was used for validation purposes. This facilitates both the validation task (as the 3rd session is never used for training the model) and the separation of the data when tests with less available data are performed.

At first, 4 different data samples were raised for the experimentation: 100% of the available data (2 complete BCI interaction sessions); 75% of the available data (first BCI session complete and half of the second session); 50% of the available data (only the first session) and 25% of the available data (half of the first session). These samples were tested with all subjects in a first experimentation round. Having regard to the results of the first experimentation, two more data samples were made for *Subject C* and *Subject E*, which had interesting results. These samples had 15% of the available data (30% of the first session) and 10% (20% of the first session).

Finally, for each subject and data sample performed, a total of 10 identical models were evaluated using the cross-validation technique. This method splits the available data into 10 different random subsets, and performs a full model training and validation using each set, computing the mean validation accuracy of the 10 models when finalizing.

5.3.2 Total executions, collected data and final runtime

A total of 20 experimentation executions were performed. 4 executions for each subject (one per sample) and 2 extra executions for *Subjects C* and *E*. Each execution included between 150 and 300 iterations, depending on the data available, runtime performance and possible errors due to unsuitable model configurations (some hyperparameters values are incompatible between them).

For each iteration, the mean validation accuracy was collected among each hyperparameter value and the date stamp, which were saved in a data frame. In addition, the confusion matrices of the iterations were also saved for further analysis, but weren't primarily used in this work. Finally, the extraction of the best configuration per experiment was performed and created a final file containing all 20 execution's best hyperparameters.

The sum of time necessary to complete the experimentation was approximately 550 hours: about 100 hours for the 6 executions of *Subject C*, 100 hours for the 6 executions of *Subject E*, 200 hours for 4 executions of *Subject B* and 150 hours for 4 executions of *Subject F*. The difference of execution time between different subjects is further explained in Chapter 6. However, the real amount of time required to complete the experimentation was far less, due to the usage of paralelization and multiple GPU devices for computation (see Section 5.2).

5.4 Survey style validation

In contrast to the classical final validation step or, commonly named, test set, a different type of validation was performed to check the quality of the model. The standard way to perform a validation consists on taking a predicted values array and a true or real values array that is previously labeled according to a given training set. Then these two arrays are compared side by side, and the number of guesses can be obtained, and thus the accuracy. Other metrics such as the true positives, false positives, true negatives and false negatives can be obtained, arranging them in a confusion matrix, which also serves for the calculation of more complex metrics, for example, precision, recall, F1-score, etc. (in the case of a binary classification, like this model's one). [Fawcett, 2006]

For this project, a different type of validation was designed. The objective of this implementation is to classify trials, and not crops. Therefore, the scope of the validation is to take the predictions of every crop that is inside a trial, make a survey between all crops and assign the most voted class to the trial. A reason to justify this survey validation is that comparing the results of a prediction inside each trial is more reliable than traditional validation, as the full window size inside each segment is used for evaluation and not individual outputs, and thus implies a better generalization of the model. An example of how this validation is implemented and its procedure can be seen in Figure 5.1.

The model is validated using this method 10 times for each subject and data sample, using the best configuration of hyperparameters obtained in the previous experiments. These results are both saved as confusion matrices and validation accuracy values to facilitate the result analysis. The results of these experiments can be seen in Chapter 6.

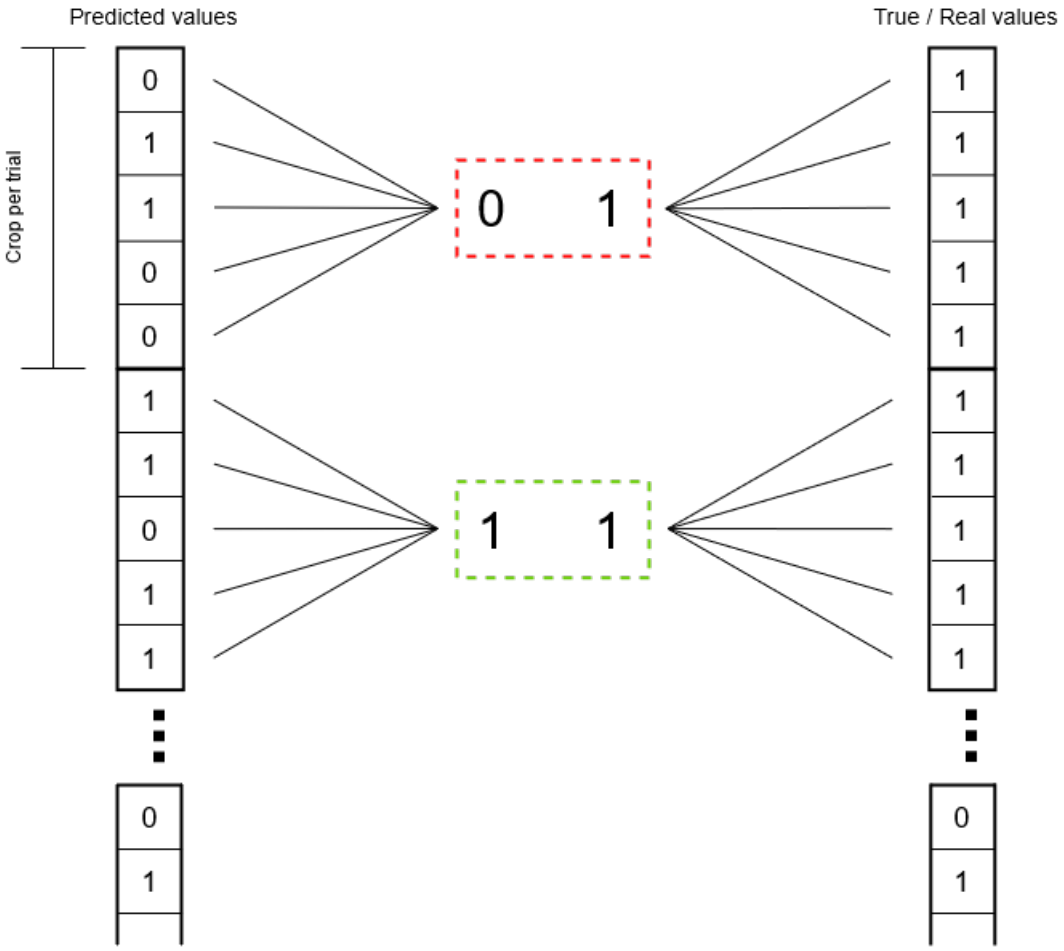

Figure 5.1: Survey style validation example. This new validation consists on splitting both the predicted values array and the true or real values array into K different arrays where $K =$ Number of crops per trial, which is previously mentioned in section 4.2.1. Then, for each new array, a survey is made among all the elements of this array, and the label with more presence is the new label of the array. Finally, the comparison between both sets of arrays is performed as usual, and the same metrics can be obtained.
In the example, a crop per trial of 5 elements is made, each element labeled as 0 or 1. In the first crop, the number of 0s is bigger than the number of 1s for the predicted values array. In the case of the true values array, the label will always be the same for each element in the crop, thus not requiring a survey process. The first crop will output an error, as the values are not the same in the predicted array and the real array. In contrast, the second prediction array has more 1s than 0s, and does not output an error. This process is repeated for each crop, and finally the resulting accuracy and confusion matrices are computed.

6. CHAPTER

Results of the experimentation

The following sections analyzes in detail the results of the experiments explained in the previous section, including the hyperparameter analysis and their influence in the model, as well as the validations tests results and the general performance of the different models, subjects and dataset samples tested.

6.1 Optimizer results, best hyperparameter configurations and influence analysis

6.1.1 Optimizer general performance

Figure 6.2 show the scatter-plot of the accuracy improvements over all iterations of the optimizer for subject F. Applying to every subject, the optimizer overall increases the accuracy of the models, and the best found hyperparameter configuration always obtains a high validation accuracy value. It must be noticed, however, that the validation performed during the optimization process is different that the final validation previously mentioned in section 5.4, but it is performed extracting the 20% of the available data and reserving it as validation subset. Figure 6.1 shows a diagram with the distribution of data for the subsamples, cross-validation, etc.

Table 6.1 shows the best configurations found among with its accuracy values. Some accuracy values decrease as the sample size increases. This is counter-intuitive, although

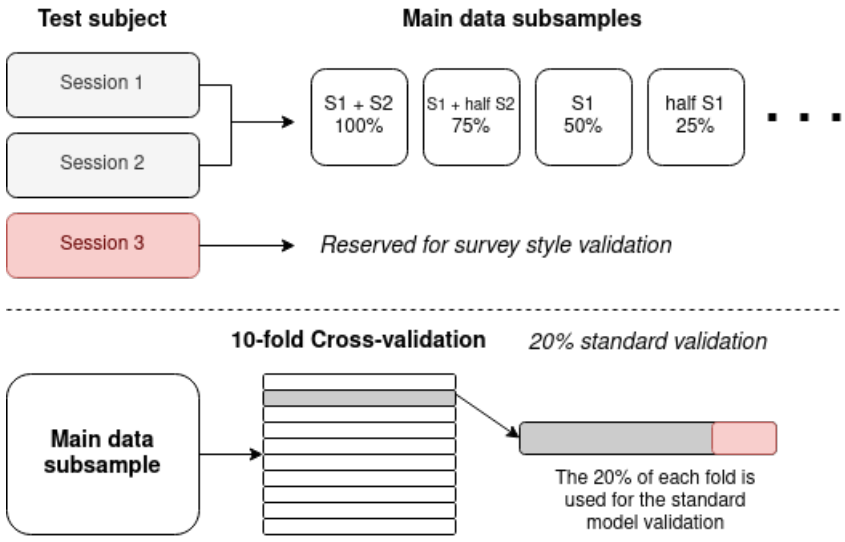

Figure 6.1: Distribution of the data for the different processes of the model.

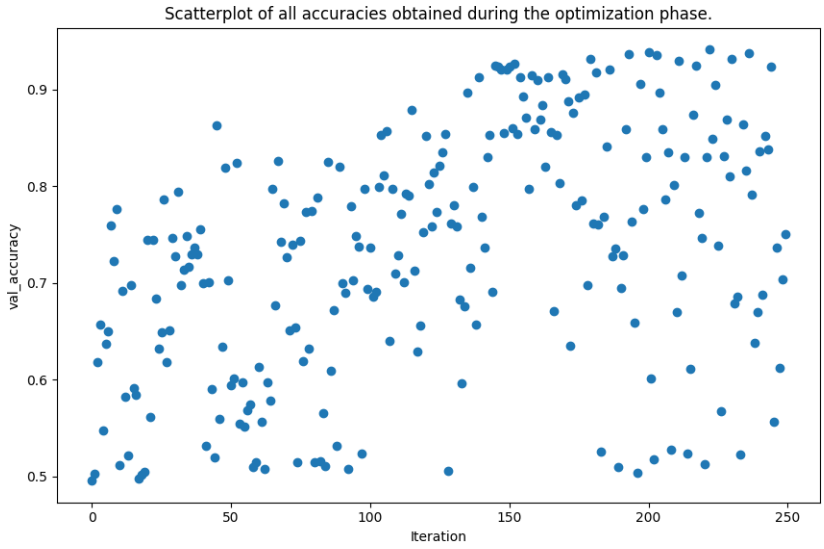

Figure 6.2: Scatter-plot of the 250 iterations of the optimizer for subject F and 100% of the data. The graph shows clearly an improvement over time, however, it can also be seen how the optimizer jumps over different configurations before deciding which one should be the next best one.

an explanation can be seen: the first 4 data samples (10%, 15%, 25%, 50%) are taken from the first session. If, for whatever reason, some subsets of this session turn out to be easier to classify, this may make these first results better.

6.1.2 Hyperparameter result analysis

In general, the hyperparameter experiments do not show many major conclusions, however, some information can be gathered from these experiments. Some hyperparameters clearly influence the performance of the models, independently from the subject and data sample. Other hyperparameters only influence in certain subjects or samples, or are more desirable in certain configurations while other configurations seem to be less affected by them. Finally, a bunch of hyperparameters do not affect at all the different subjects, samples or configurations, its influence is not clear or there are not enough samples to justify a certain value or choice.

Consecutively, an in depth analysis is listed below:

- The optimization algorithm has a clear influence in the model: it is better to use Adam over SGD (see 6.3). This confirms that the optimization algorithm is an important hyperparameter, and therefore it is convenient to test more optimization algorithms in future experiments performed for this model.
- Window size is in most cases better at a value of 100 than in a value of 50. Similar to the optimization algorithm, it may be interesting to test other values in future experiments, as it is one of the main hyperparameters that directly control the behaviour of the model. Figure 6.4 shows an example of the clear difference in the optimal parameter preference.
- The step size shows a descending size pattern, which clearly shows that lower values have better results than higher values. This contrast with the fact that *window_size* is clearly better with a value of 100 and therefore this parameter is only useful when *window_size* is 50, but the parameter is still optimized even when it's not useful at all.
- ReLU activation function seems to perform better than ELU function in most cases, especially for Subjects B and F.
- The learning rate varies between subjects. The optimal values seem to be between 10^{-4} and 10^{-2}, but the exact location depends on the subject and the data sample

and the best configurations tend to be near 10^{-4}, except for the bigger data samples in subject E (Figure 6.5).

- Temporal filters do not offer major clear results, but lower values (between 10 and 20) tend to be worse in many cases.
- The usage of additional convolutional blocks is dependant on the subject. For subject B, a predictable result is shown: it is better to use an additional block if more data is available. For subject F, always using an additional block is better than not using one. However, for subjects C and E, a clear patter does not appear at all.
- Batch normalization does not show a clear pattern. It seems that its use is not justified. However, most iterations show a preference in not using this technique at all, but in some other examples this is just the opposite. The reason of this result could be the above-mentioned optimization value stuck, but it is not clear at all.
- Similarly to batch normalization, the usage of dropout is also not justified (at least at a 0.6 value) as there is no clear pattern of improvement. Being usually a successful parameter, it could be worth as future work to test other smaller values of the parameter, for example, a value of 0.2.
- The kernel size is another hyperparameter that does not offer any clear successful value. It seems not to be a relevant hyperparameter in this experimentation (Figure 6.6).
- The number of additional FC layers do not show any clear influence in the performance of the model.
- It is difficult to distinguish a pattern or influence in the number of neurons for the additional FC layers. It must be taken into account that the number of possible values for this hyperparameter is relatively high. Therefore, we have less samples for each value and the reliability of the results is lower. In any case, it seems desirable to avoid extreme values, as some examples fail miserably.
- Finally, the max-pooling size does not follow any clear pattern.

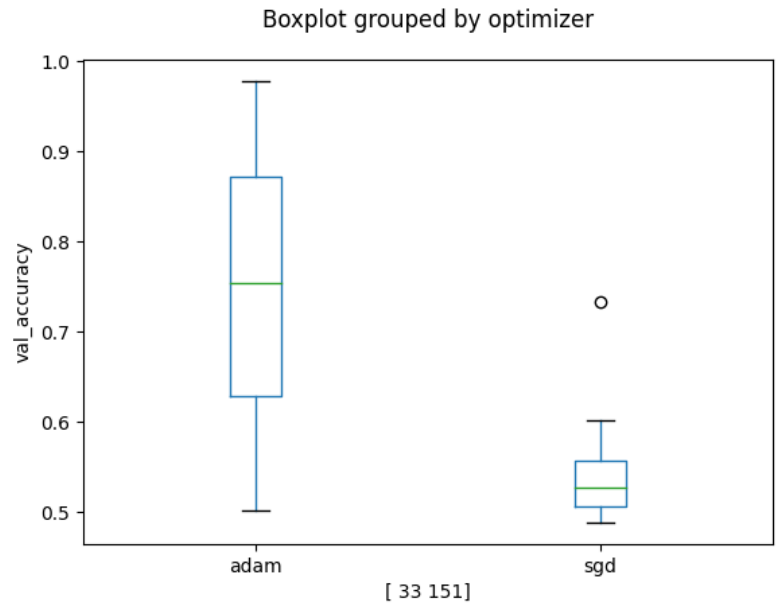

Figure 6.3: Example box-and-whisker plot for subject B and 25% of the total data for *optimizer* hyperparameter. The plot shows how strong the difference between both choices is. The numbers between brackets indicate how many samples of each hyperparameter value was selected during the optimization process. Interestingly, the most popular value for this example is the SGD algorithm, but Adam clearly outperforms it.

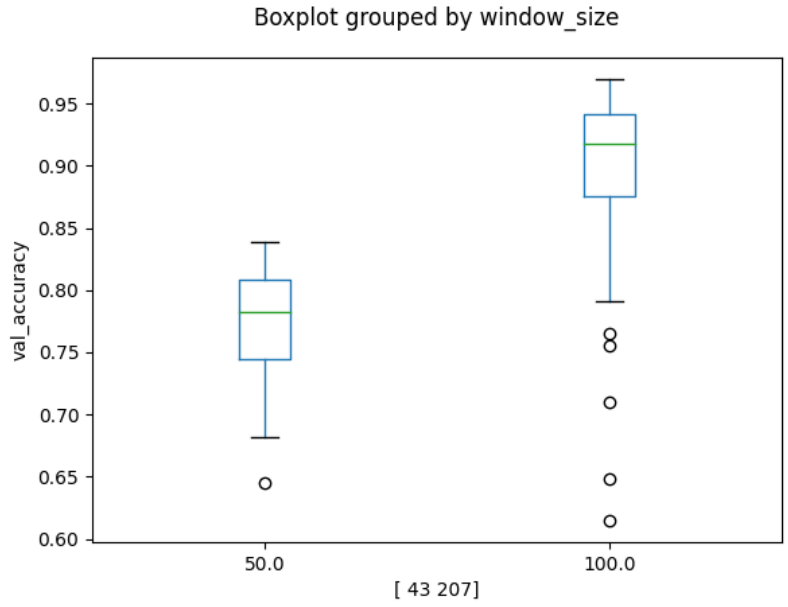

Figure 6.4: Boxplot for subject C and 75% of the total data for *window_size* hyperparameter. The plot clearly shows how the value of 100 is better than a value of 50.

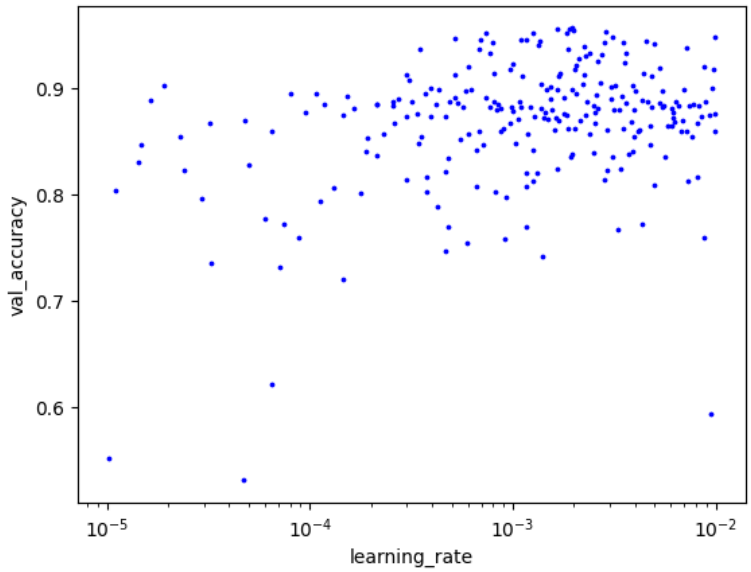

Figure 6.5: *learning_rate*'s logarithmic scale plot for subject E and 75% of the data. In contrast with the generalized 10^{-4} value for almost every subject and data sample, here the best values are around 10^{-3}.

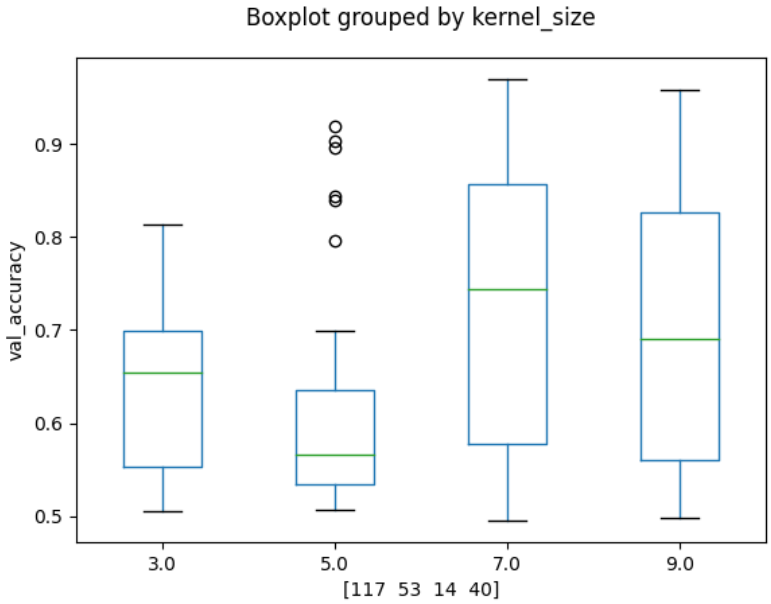

Figure 6.6: Boxplot of the *kernel_size* values for subject F and 25% of the data. Although the most popular values tend to be smaller ones, the plot doesn't show any visible pattern in the preference or the optimal value for the convolution's kernel. The greater values that are obtained with the values of 7 and 9 are not really reliable due to the absence of more samples.

It should also be taken into account that this analysis is not definitive, as it may happen that, by chance, some values of the hyperparameter we are analysing have been combined with less suitable values for other hyperparameters i.e. one hyperparameter choice has been used in combination with bad values of other hyperparameters values and that, as a result, it comes out worse than using another choice of the first hyperparameter. Other problems may have come from the fact that the optimization of certain hyperparameters could have got stuck in some suboptimal values, as some hyperparameters have more influence than others in this process. In any case, this chance effect should reduce the more iterations and samples we have.

Subject	Hyperparameters													Validation
	window_size	*step_size*	*kernel_size*	*pool_size*	*optimizer*	*activation*	*learning_rate*	*temporal_filters*	*batch_normalization*	*n_conv_layers*	*n_fc_layers*	*n_neurons_2nd_layer*	*dropout_layer*	**Accuracy**
B25	50.0	0.1	9.0	3.0	adam	relu	1.92×10^{-4}	50.0	False	0.0	1.0	0.75	0.0	0.977
B50	50.0	0.1	9.0	2.0	adam	elu	1.34×10^{-4}	40.0	False	0.0	2.0	0.6	0.0	0.98
B75	50.0	0.1	7.0	2.0	adam	relu	2.58×10^{-4}	50.0	False	1.0	1.0	0.45	0.0	0.951
B100	50.0	0.1	7.0	2.0	adam	relu	2.22×10^{-3}	40.0	True	1.0	1.0	0.4	0.6	0.907
C10	50.0	0.1	5.0	2.0	adam	relu	6.9×10^{-4}	50.0	False	1.0	2.0	0.45	0.6	0.99
C15	50.0	0.1	7.0	2.0	adam	relu	4.53×10^{-4}	50.0	False	1.0	1.0	0.65	0.6	0.985
C25	100.0	0.1	9.0	3.0	adam	relu	8×10^{-5}	40.0	False	1.0	1.0	0.35	0.6	0.979
C50	100.0	0.6	3.0	2.0	adam	relu	7.1×10^{-4}	50.0	False	1.0	2.0	0.25	0.0	0.966
C75	100.0	0.6	9.0	3.0	adam	relu	3×10^{-4}	40.0	True	0.0	2.0	0.65	0.6	0.967
C100	100.0	0.6	5.0	3.0	adam	elu	6.54×10^{-4}	40.0	False	0.0	2.0	0.25	0.0	0.969
E10	100.0	0.1	9.0	3.0	adam	relu	8.83×10^{-4}	20.0	False	1.0	1.0	0.35	0.6	0.997
E15	100.0	0.1	2.0	3.0	adam	relu	9.76×10^{-4}	40.0	False	1.0	0.0	0.7	0.0	0.992
E25	100.0	0.1	3.0	3.0	adam	relu	7.33×10^{-4}	50.0	False	1.0	2.0	0.5	0.6	0.974
E50	100.0	0.1	3.0	3.0	adam	relu	8.31×10^{-4}	40.0	False	1.0	1.0	0.55	0.6	0.968
E75	100.0	0.1	5.0	2.0	adam	relu	1.96×10^{-3}	20.0	False	1.0	0.0	0.5	0.0	0.956
E100	100.0	0.1	9.0	2.0	adam	relu	1.88×10^{-3}	50.0	False	1.0	0.0	0.3	0.6	0.962
F25	50.0	0.1	7.0	2.0	adam	relu	2.92×10^{-4}	50.0	False	1.0	0.0	0.55	0.0	0.969
F50	100.0	0.1	9.0	3.0	adam	relu	4.5×10^{-4}	50.0	False	1.0	2.0	0.75	0.6	0.95
F75	100.0	0.1	9.0	3.0	adam	relu	1.99×10^{-4}	50.0	False	1.0	1.0	0.7	0.0	0.941
F100	100.0	0.1	9.0	3.0	adam	relu	3.2×10^{-4}	50.0	False	1.0	2.0	0.7	0.6	0.941

Table 6.1: Complete table of all best found hyperparameter configurations for all test subjects and data samples with its corresponding validation accuracy. The subjects are organized by its name and data sample percentage, for example, first entry of the table is **B25**, which stands for subject B and 25% of the available data. It is also noticeable the decrease in performance when augmenting the data amount in the optimization process.

6.2 Validation tests results

Figure 6.7 shows the results of the mean validation accuracy obtained for each subject and data sample, using the survey validation explained in section 5.4.

In general, the C and E subjects have better results than the subjects B and F, with this first mentioned subjects achieving a mean validation accuracy greater than 0.8 for almost every data sample and the impressive results of subject E with the full data available for training.

In contrast, subjects B and F suffer from what is called **BCI Illiteracy**. A a slight improvement is noticed across the increase in data size, but the barrier of 0.6 validation accuracy value is never reached, noticing that subject B struggles more than subject F to reach a higher value. This BCI Illiteracy could be improved by making changes in the model, rather than accomplishing a better hyperparameter configuration. Another problem that BCI Illiteracy carries with it is the increase in the cost of computing, as it is mentioned in section 5.3. [Vidaurre and Blankertz, 2010]

Taking into account that the validation is made with the third session, and the training is performed with the first and second sessions, it is normal that the achieved results with this validation differ a lot when compared with the hyperparameter optimization values. Table 6.2 shows the accuracies obtained during the survey validation accuracy phase.

Subjects B and E suffer from the same “curse” that the accuracies of the configurations found during the optimization have when tested with the survey validation. It is noticeable how poorly subject E performs with 25% of the data (in contrast with the performance of 15%) and even worse with 50% of the data. In the case of subject B, the decrease is much slighter, but it is also quite remarkable. A possible explanation is that some subsets of this session may be really similar to the 3rd session, which would make these first results better.

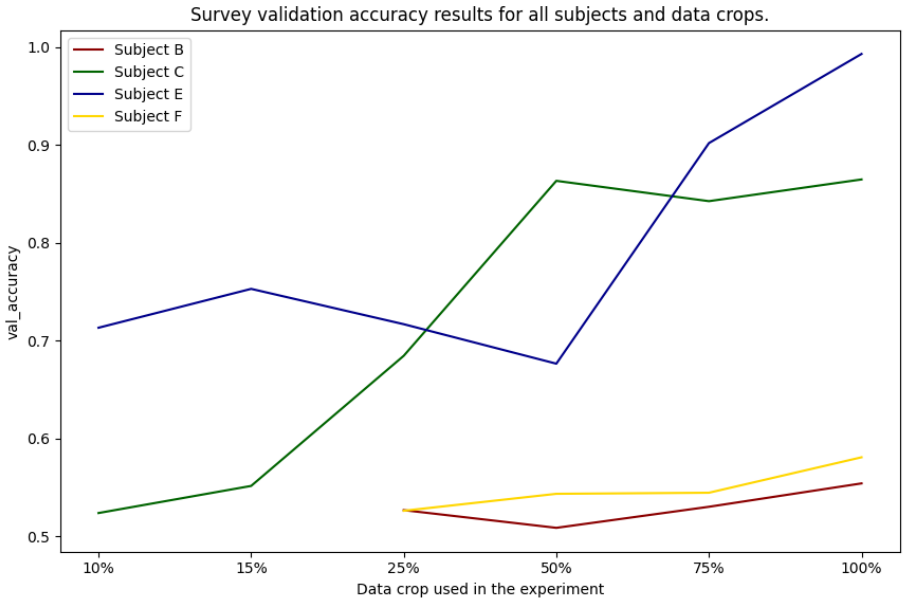

Figure 6.7: Plot of validation accuracies obtained via survey validation method. The values of subjects B and F are clearly worse than the results obtained by subjects C and E. The subject C value hill when training with more than the 50% is also visible. The decay in performance of subject E in the first data sample and its astounding results with 100% of the data can also be seen.

However, when introducing the second session, these values both improve or stay close to the best achieved value, except for subject C. The case with subject C is peculiar when compared with the other 3 subjects. It seems that with the data from the first session he reaches his maximum performance and it is not able to improve with more data. This may mean that sessions 1 and 3 are quite similar and, in contrast, session 2 seems to not adding any knowledge to the model in this case.

Subject	Survey validation accuracy
B25	0.526
B50	0.508
B75	0.53
B100	0.554
C10	0.523
C15	0.551
C25	0.684
C50	0.863
C75	0.842
C100	0.864
E10	0.713
E15	0.752
E25	0.716
E50	0.676
E75	0.902
E100	0.993
F25	0.526
F50	0.543
F75	0.544
F100	0.58

Table 6.2: Final survey validation accuracies. The decrease in performance compared with the results in the optimization process (see table 6.1) is notorious due to the usage of the third session as validation, as it has never been seen before during the model fitting step.

7. CHAPTER

Conclusions and future work

7.1 Conclusions

As has been shown, BCI interaction paradigms aim to translate neural activity into control signals for external devices. To record this information, we need EEG motor imagery, which translates these brain signals into usable raw data. Machine Learning techniques allow extracting information from EEG recordings of brain activity in order to perform an end-to-end learning from this raw data. A particular technique of Machine Learning is Deep Learning and Convolutional Neural Networks, which are well-fitted to computer-vision and image related AI problems, but it can be modified to perform an EEG problem classification and obtain satisfactory results. These results lay a basis in more and extended experiments, among with future research.

Summarizing the proposed objectives of this project, a functional model that is capable of discriminate imaginary movements recorded as EEGs for BCI has been developed. In addition, the hyperparameters that control the behaviour and performance of this model have been selected and optimized, using modern optimization techniques such as Bayesian Optimization. A complete and reliable experimentation has been developed in order to evaluate the behaviour of the model given the best found hyperparameter configurations and, finally, an analysis of the results of the experimentation has been performed and its conclusions have been annotated. To conclude, the theoretical foundations, model design, experimentations and its results have been collected and summarized in a final report.

The experimentation shows some results that are highly relevant for future researching and open new ways of experimenting with this type of models. For example, the results are clearly dependant on the subject we are working with; it seems that there are differences between sessions, and sessions that are more similar between them than others (experiments with more sessions may be a starting point for Transfer Learning research); the most relevant parameters (*optimizer* Adam, *window_size* of 100, ReLU *activation* function, ...) but also take into account other parameters that seem to not influence at all the model (*kernel_size* or *max_pooling*).

In conclusion, EEG for BCI interactions are a promising research field for development of primarily medical and neurological tools. In addition, CNNs are not only a novel, promising tool in the EEG decoding field, but combined with innovative Machine-Learning and optimization techniques, they may also open up new windows for EEG-related problems.

7.2 Future work

Many fields of this project may be extended in future works. The usage of more BCI interaction paradigms is one of the main topics a future research may start with, among with the inclusion of more test subjects and even more data subsets. The technique of Transfer Learning when working with different sessions per subject is also another important feature that this project does not cover.

Selecting the survey validation as the standard validation for all steps of the implementation may be another addition to the future experiments performed in this topic.

Curating the proposed hyperparameters, finding new ones and updating the search values of them are also possibilities to extend the experimentation in the proposed model and, finally, improving the model and include novel or advanced Machine-Learning related techniques, such as the previously mentioned Transfer Learning, or even Semi-Supervised Learning may lead the results of this project into a higher level.

All in all, the possibilities in continuing the research are uncountable and this project lay a foundation on them.

Bibliography

[Bergstra and Bengio, 2012] Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2).

[Bergstra et al., 2015] Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., and Cox, D. D. (2015). Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008.

[Cho et al., 2020] Cho, H., Kim, Y., Lee, E., Choi, D., Lee, Y., and Rhee, W. (2020). Basic enhancement strategies when using bayesian optimization for hyperparameter tuning of deep neural networks. *IEEE Access*, 8:52588–52608.

[Chollet et al., 2015] Chollet, F. et al. (2015). Keras. https://keras.io.

[Davenport, 2018] Davenport, T. (2018). Machine learning: What it is and why it matters. In *SAS*. SAS Institute Inc.

[Deng and Yu, 2014] Deng, L. and Yu, D. (2014). Deep learning: methods and applications. *Foundations and trends in signal processing*, 7(3–4):197–387.

[Fawcett, 2006] Fawcett, T. (2006). An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874. ROC Analysis in Pattern Recognition.

[Fukushima et al., 1983] Fukushima, K., Miyake, S., and Ito, T. (1983). Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE transactions on systems, man, and cybernetics*, (5):826–834.

[Géron, 2019] Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.

[Ghanbari-Adivi and Mosleh, 2019] Ghanbari-Adivi, F. and Mosleh, M. (2019). Text emotion detection in social networks using a novel ensemble classifier based on parzen tree estimator (tpe). *Neural Computing and Applications*, 31(12):8971–8983.

[Gramfort et al., 2013] Gramfort, A., Luessi, M., Larson, E., Engemann, D. A., Strohmeier, D., Brodbeck, C., Goj, R., Jas, M., Brooks, T., Parkkonen, L., and Hämäläinen, M. S. (2013). MEG and EEG data analysis with MNE-Python. *Frontiers in Neuroscience*, 7(267):1–13.

[Harris et al., 2020] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

[Hunter, 2007] Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

[Kaya et al., 2018] Kaya, M., Binli, M. K., Ozbay, E., Yanar, H., and Mishchenko, Y. (2018). A large electroencephalographic motor imagery dataset for electroencephalographic brain computer interfaces. *Scientific Data*, 5(1):180211.

[Kingma and Ba, 2017] Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

[Köppen, 2000] Köppen, M. (2000). The curse of dimensionality. In *5th Online World Conference on Soft Computing in Industrial Applications (WSC5)*, volume 1, pages 4–8.

[Krizhevsky et al., 2017] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.

[Liaw et al., 2018] Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., and Stoica, I. (2018). Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*.

[Moritz et al., 2017] Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Paul, W., Jordan, M. I., and Stoica, I. (2017). Ray: A distributed framework for emerging AI applications. *CoRR*, abs/1712.05889.

[Muñoz-Villamizar et al., 2020] Muñoz-Villamizar, A., Rafavy, C. Y., and Casey, J. (2020). Machine learning and optimization-based modeling for asset management: a case study. *International Journal of Productivity and Performance Management*.

[Ongsulee, 2017] Ongsulee, P. (2017). Artificial intelligence, machine learning and deep learning. In *2017 15th International Conference on ICT and Knowledge Engineering (ICT KE)*, pages 1–6.

[Pandas-Dev-Team, 2020] Pandas-Dev-Team (2020). pandas-dev/pandas: Pandas.

[Russell and Norvig, 2002] Russell, S. and Norvig, P. (2002). Artificial intelligence: a modern approach.

[Schirrmeister et al., 2017] Schirrmeister, R. T., Springenberg, J. T., Fiederer, L. D. J., Glasstetter, M., Eggensperger, K., Tangermann, M., Hutter, F., Burgard, W., and Ball, T. (2017). Deep learning with convolutional neural networks for eeg decoding and visualization. *Human Brain Mapping*, 38(11):5391–5420.

[Schomer and Da Silva, 2012] Schomer, D. L. and Da Silva, F. L. (2012). *Niedermeyer's electroencephalography: basic principles, clinical applications, and related fields*. Lippincott Williams & Wilkins.

[Swinburne, 2004] Swinburne, R. (2004). Bayes' theorem. *Revue Philosophique de la France Et de l*, 194(2).

[Trottier et al., 2017] Trottier, L., Giguere, P., and Chaib-Draa, B. (2017). Parametric exponential linear unit for deep convolutional neural networks. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 207–214. IEEE.

[Vidaurre and Blankertz, 2010] Vidaurre, C. and Blankertz, B. (2010). Towards a cure for bci illiteracy. *Brain topography*, 23(2):194–198.

[Virtanen et al., 2020] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E.,

Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

[Williams and Rasmussen, 2006] Williams, C. K. and Rasmussen, C. E. (2006). *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA.

[Wilson et al., 2018] Wilson, J. T., Hutter, F., and Deisenroth, M. P. (2018). Maximizing acquisition functions for bayesian optimization. *arXiv preprint arXiv:1805.10196*.

[Wu et al., 2019] Wu, J., Chen, X.-Y., Zhang, H., Xiong, L.-D., Lei, H., and Deng, S.-H. (2019). Hyperparameter optimization for machine learning models based on bayesian optimization. *Journal of Electronic Science and Technology*, 17(1):26–40.