TÉCNICO LISBOA

Error perception classification in Brain-Computer interfaces using Convolutional Neural Networks

José Rafael Cabral Correia

Thesis to obtain the Master of Science Degree in

Biomedical Engineering

Supervisors: Prof. João Miguel Raposo Sanches
Prof. Luca Mainardi

Examination Committee

Chairperson: Prof. Patrícia Margarida Piedade Figueiredo
Supervisor: Prof. João Miguel Raposo Sanches
Member of the Committee: Prof. Athanasios Vourvopoulos

January 2021

Declaration

I declare that this document is an original work of my own authorship and that it fulfils all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Ad maiorem Dei gloriam

To my parents, sister and brother!
For everything!

Preface

The work presented in this thesis was performed at Department of Electronics, Information and Bioengineering of Politecnico di Milano (Milan, Italy), during the period February-July 2020, under the supervision of Prof. Luca Mainardi and PhD student Davide Marzorati, and within the frame of the Erasmus+ program. The thesis was co-supervised at Instituto Superior Técnico (Lisbon, Portugal) by Prof. João Sanches.

Acknowledgments

This work would not be complete without a proper acknowledgement to all the people who, directly or indirectly, contributed to it!

Firstly, many thanks to my supervisors, professors Luca Mainardi and João Sanches. It is due, certainly, for their advises and words of encouragement but above all for the opportunity of adding an international component to my thesis experience. Thanks also to the Ph.D. student and co-supervisor Davide Marzorati for sharing materials and insights in the beginning of the work.

A few days after arriving to Milan, the Covid-19 outbreak struck Italy, forcing a quarantine isolation. Thanks to all the people who, during this time, kept me company through the countless hours of video chat and specially to Madalena G., Madalena R., Margarida R., Tiago R., Salomé S., Diogo R. and Miguel M. for actually being there, even if only for a short moment.

Many more people have, indirectly, helped me to reach this point. They shaped my personality, my beliefs, my scientific curiosity and provided me the tools, motivation and guidance to pursue my dreams. Thanks, Jeenal M. and João A., for all the (excessively) long philosophical talks after classes at high-school! Thanks, Sofia C. and André M., for the friendship and patience along all these years! All the close friends from *AL*s and all my course colleagues, thanks for the fond memories. Thanks to Rafael M., Vicente G. and the rest of my *praxis* family, for helping me integrate in academic life during the first years. Thanks to all my Erasmus friends both from the Netherlands and from Italy: the whole *Funest* family, the Ducklings, Isabelle, Alessandro, Julie, Parham, Vinish, Sanchana, Sharan, Vishwajit, Ann-Kathrin, David, Chloe, Nicole, Oyvind, Eva, Nicolas, Nadine, Katerina, Oscar, Malte, Anne, Robert, and so many more! Thanks to the *V.I.P* and the *Maranatha*s! Thanks to the guys from *WePlay* that rocked alongside me! Thanks to Joe and Chloe, for having crossed the Atlantic to become family!

Thanks to so many other friends whose friendship backgrounds are so diverse that I would not have space to enumerate them all: Raquel B., Bruno D., Mariana G., Miguel R., Miguel S.R., Jorge S.R., João P.A., Ombline H., Rafael L-H., Molly S., Pe. Hugo S., Marisa P., Carlos C., Gonçalo M., Mette S., Francesco P., Aisulu Y., Ainel R., Gaukhar M., Sanzhar S., Johanna K. just to name a few.

And finally, to my dear family! No amount of words can describe the support and care they have given me! All I can do is try to give it all back on every new day that passes!

Abstract

Motivation: Capturing the error perception of a human interacting with a Brain-Computer interface is a key element in improving the performance of these systems and making the interaction more seamless. Convolutional neural network (CNN) have been recently applied for this task rendering the classification model free of feature-selection. Despite the advances in the last years, there is still room for improving the accuracy so that it can be used in real-life applications.
Objectives: The goals of the present work are to investigate, replicate and validate previous CNN models used for error perception classification; to propose new CNN models based on the advances in Machine Learning and lastly, to make all the code and models developed publicly available.
Methods: After a literature review, three recent CNN models for error-related potential (ErrP) classification are replicated. Then, the author evaluates different new models that result from investigating CNN models used for classification of ErrP and *P300* signals. The Monitoring Error-Related Potential dataset is used to train and test all the models.
Results: The best model from the literature achieves an accuracy, sensitivity, and specificity of 77.6%, 71.7%, and 83.1%, respectively. For the best proposed model, these metrics are of 80.4%, 75.9%, and 84.7%, respectively, which represents a statistically significant increase on the literature models ($p = 0.0004$). Furthermore, an EEG input with a shorter temporal size of $600ms$ is successfully applied instead of the typical one-second long input without significant loss of performance ($p = 0.647$). All models are made available online for easier future replication and peer review.
Conclusions: The new proposed model outperforms the state-of-the-art. The $600ms$ input allows faster processing times in real-time BCI applications without loss of performance.

Keywords

Brain-computer interface; Convolutional Neural Networks; Feedback error; Error-related Potentials

Resumo

Motivação: Obter a percepção de erro de um humano ao interagir com uma interface cérebro-computador é um ponto chave para melhorar o desempenho desses sistemas. Recentemente, redes neurais convolucionais (CNN) foram usadas com esse propósito, libertando o modelo da necessidade de escolher características específicas apartir dos dados. Apesar dos avanços dos últimos anos, há ainda espaço para melhorias.
Objetivos: Os objetivos deste trabalho são replicar e validar modelos anteriores de CNN usados para classificar percepção de erro; propor novos modelos CNN baseados em avanços de Aprendizagem Automática e ainda disponibilizar todo o código e modelos desenvolvidos publicamente.
Métodos: Depois de uma revisão bibliográfica, três modelos CNN de classificação de potenciais dependentes de erro (ErrP) foram escolhidos para replicação. Depois, o autor avalia diferentes novos modelos resultantes do estudo de modelos CNN que classificam sinais ErrP e P300. A base de dados "Monitoring Error-Related Potential" é usada para treinar e testar todos os modelos.
Resultados: O melhor modelo da literatura atinge valores de precisão, sensibilidade e especificidade de 77.6%, 71.7%, e 83.1%, respetivamente. Para o melhor modelo proposto, estas métricas são de 80.4%, 75.9% e 84.7%, respectivamente, o que representa um aumento estatisticamente significante sobre os modelos da literatura ($p = 0.0004$). Um input de EEG de mais curta duração ($600ms$) é também usado em vez do input mais comum de um segundo com sucesso e sem perda de performance.
Conclusões: O novo modelo proposto supera o desempenho do state-of-the-art. O input de $600ms$ permite tempos de processamente menores em aplicações de tempo-real sem perda de desempenho.

Palavras Chave

Interface cérebro-computador; Rede neural convolucional; Erro de *feedback*; Potenciais de erro

Sommario

Motivazione: Ottenere la percezione erronea di una persona che interage con una Brain-Computer Interface è un elemento chiave nel meglioramento della performance di questo tipo di sistema. A questo fine, sono state utilizzate le Convolutional-Neural Networks (CNN), che rendono il modello libero dalla necessità di selezionare specifiche caratteristiche dai dati. Nonostante i progressi negli ultimi anni, c'è ancora spazio per il miglioramento della precisione dei modelli.

Obiettivi: Gli obiettivi del presente lavoro sono replicare e convalidare i precedenti modelli CNN utilizzati per la classificazione della percezione degli errori; proporre nuovi modelli CNN basati sui progressi del Machine Learning e, infine, rendere disponibili pubblicamente tutto il codice e i modelli sviluppati.

Metodi: Dopo una revisione della letteratura, vengono replicati tre modelli CNN recenti per la classificazione del potenziale correlato all'errore (ErrP). Quindi, l'autore valuta diversi nuovi modelli che derivano dai modelli CNN utilizzati per la classificazione dei segnali ErrP e P300. Il dataset "Monitoring Error-Related Potential" viene utilizzato per addestrare e testare tutti i modelli.

Resultati: Il miglior modello della letteratura raggiunge un'accuratezza, sensibilità e specificità rispettivamente del 77.6%, 71.7% e 83.1%. Per il miglior modello proposto, queste metriche sono rispettivamente dell' 80.4%, 75.9% e 84.7%, il che rappresenta un aumento statisticamente significativo sui modelli della letteratura ($p = 0.0004$). Inoltre, un input EEG con una dimensione temporale inferiore di $600ms$ viene applicato con successo invece del tipico ingresso di un secondo, senza una significativa perdita di prestazioni. Tutti i modelli sono resi disponibili online per una più facile replica futura e revisione tra pari.

Conclusioni: Il nuovo modello proposto supera lo stato dell'arte. L'input di $600ms$ consente tempi di elaborazione più rapidi nelle applicazioni BCI in tempo reale senza perdita di prestazioni.

Parole Chiave

Brain-Computer Interface; Convolutional-Neural Networks; Errore di *feedback*; Potenziale di errore

Contents

List of Figures

List of Tables

Acronyms

ACC	Anterior cingulate cortex
ALS	Amyotrophic lateral sclerosis
ANN	Artificial Neural Network
BCI	Brain-Computer interface
BMI	Brain-Machine interface
BN	Batch normalization
CNN	Convolutional neural network
DL	Deep Learning
DNI	Direct Neural interface
ECoG	Electrocorticography
EEG	Electroencephalogram
ELU	Exponential linear unit
ERN	Error-related negativity
ERP	Event-related potential
ErrP	Error-related potential
FC	Fully-connected
fMRI	Functional magnetic resonance imaging
fNIRS	Functional near-infrared spectroscopy
FRN	Feedback-related negativity
GPU	Graphical Processing Unit
INR	Intracortical neuron recording
k-NN	K-nearest neighbours
LeakyReLU	Leaky Rectified Linear Unit
LDA	Linear discriminant analysis
MCC	Matthews correlation coefficient
MEG	Magnetoencephalography
ML	Machine Learning
MLP	Multilayer perceptron
MMI	Mind-Machine interface
nMCC	normalized Matthews correlation coefficient
Pe	Positive deflection
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
SNR	Signal-to-noise ratio
SVM	Support vector machine
TanH	Hyperbolic tangent
TPU	Tensor Processing Unit
XOR	Logic exclusive OR

1 Introduction

Contents

1.1 Motivation

The development of Brain-Computer interface (BCI) systems is a very active field of research and has grown considerably during the last decades [1, 2]. Usually, computer input requires the user to perform a muscular action controlled by the brain such as when using the mouse, keyboard, voice commands, or other methods. BCI systems define a way of interaction between a human and a computer that relies solely on brain activity, rendering the use of an intermediate actuation step by means of peripheral nerves and muscles unnecessary [3].

Nowadays many BCI systems are being designed and applied with a great focus on the medical field [1]. Some of these include applications on prevention [4], diagnosis [5–8] and rehabilitation [9–11]. Other non-medical applications also exist for education [12], marketing [13], security [14],or entertainment [15]. Despite some of these applications still being in a very early development stage, BCI systems are certainly going to change the way we interact with and are aware of our surrounding technology. Although a sci-fi future, where computers and machines are controlled by the mere will of the user, seems still a distant reality, it is very plausible that BCI systems will be at the base of the next technological revolution [16, 17], such as the advent of airplanes or the internet was at their time.

Different brain signals can be used to assess the subject's intention so that it is translated into a computer or machine command. One such signal is the error-related potential (ErrP) which has gained popularity among the BCI community as it provides insight as to when a user makes or perceives an error during the execution of a task [18] but still presents low accuracy rates when compared to its counterparts [19]. If high accuracy on error-related potential classification is obtained, BCI systems can better predict the real intent of the user by automatically detecting errors and correct for them, thus increasing the system's performance.

Many methods can be used to classify the occurrence of ErrPs in the brain. However, issues such as the processing of enormous amounts of data or lack of generalization present challenging problems. In the last few years, Deep Learning (DL) has become an emergent technology in tackling these problems. Deep Learning brings a lot of advantages when handling complex and large types of data and it is referred to as state-of-the-art in many fields such as image recognition [20], natural language [21], stock market [22], advertising [23] or healthcare [24]. It allows the scientific community to move away from feature engineered methods and provides end-to-end solutions which learn meaningful, high-level features on its own [25].

1.2 Objective and original contributions

The present work aims at the development of a Deep Learning based classifier for error perception detection in a BCI context using error-related potentials (ErrPs).

The goals of the present work are three-fold: (a) investigate and replicate previous CNN models used for ErrP classification; (b) develop new models based on advances in the field of Deep Learning that can provide improvements to the classification of ErrP and (c) provide open-source code for all the models reviewed in this work and the new proposed ones.

The main contributions of this work relate to the mentioned goals and are:

1. Revision, summary, and validation of previous literature;
2. Development of new models for error-related potential classification with the best model outperforming the state-of-the-art;
3. Release of open-source code for the previous literature models and the new proposed ones.
4. Paper submission to *ICASSP 2021* (International Conference on Acoustics, Speech, and Signal Processing).

1.3 Thesis outline

In chapter 2, the background regarding BCI systems is presented: what constitutes such a system, how does it measure brain activity, what different event-related signals are used to control it, and some of the features and Machine Learning models used to classify ErrPs.

Chapter 3 introduces the background knowledge for Deep Learning models. Starting from the general idea of an Artificial Neural Network (ANN), it evolves into the concept of a convolutional neural network (CNN), where its building blocks are presented, together with the training process. Finally, the state-of-the-art for classification of event-related potentials (ERPs) is reviewed.

The methodology, in chapter 4, starts by presenting the problem with a mathematical formal description. The subsequent sections address the methods taken to solve the problem: database selection, data pre-processing, CNN models, and performance measures. In the end, the coding framework used to develop the thesis work is also detailed.

Chapter 5 presents the experimental results obtained as well as a discussion on those results.

Finally, chapter 6 ends the work with conclusion notes and future work suggestions.

2

Brain-computer interface background

Contents

2.1 Definition

A Brain-Computer interface (BCI), also referenced in the literature as Mind-Machine interface (MMI) [26], Direct Neural interface (DNI) [27] or Brain-Machine interface (BMI) [28], is a system that directly converts brain activity into computer commands without making use of the normal peripheral output pathways such as nerves and muscles. It allows, for example, patients with severe motor disabilities such as amyotrophic lateral sclerosis (ALS) or spinal cord injuries to communicate with the external world [29].

Conventionally, BCIs are composed of five abstract layers as shown in Figure 2.1: acquisition, pre-processing, feature extraction, classification and application [30]. During acquisition, the electrical activity of the brain is registered and converted into digital information. Pre-processing removes noise from the signal, thus increasing its signal-to-noise ratio. Feature extraction finds relevant characteristics in the signal which are fed into the classifier that then associates the features with one of several possible application commands (classes). Finally, the application receives the intent of the user as a command, applies it, and displays some feedback back to the user.

Although this is the conventional scheme, the distinction between feature extraction and classification is only applicable for BCI systems that do not use Deep Learning methods to train their models. As explained ahead, in chapter 3, DL algorithms perform those two steps at the same time: non-engineered features are learned while the classifier is being trained.

In section 2.2, different methods for signal acquisition are presented and their advantages and disadvantages are highlighted.

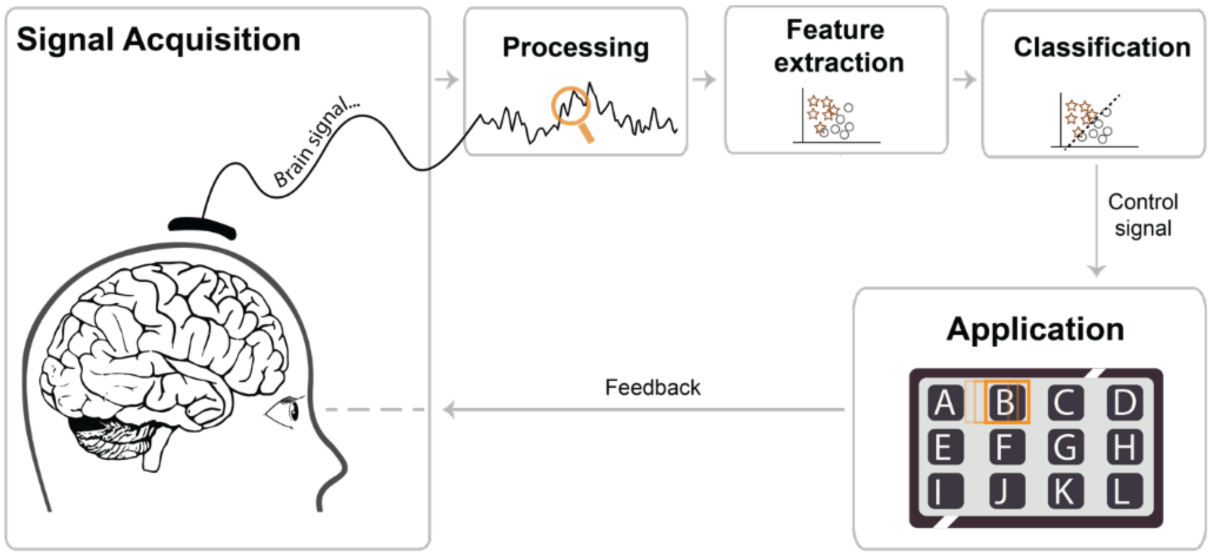


Figure 2.1: Components of a BCI system. Taken from [30] with permission.

2.2 Signal acquisition

Signal acquisition methods can be categorized depending on multiple parameters such as invasiveness level, temporal and spatial resolution, complexity, or price. Most commonly, these methods are referred to as invasive or non-invasive. Invasive techniques record neural activity through electrodes that are surgically implanted either inside or on top of the brain while non-invasive techniques do not require surgical intervention, thus reducing the risk to the patient [31].

Electroencephalogram (EEG) is, by far, the most used non-invasive acquisition method, while two common invasive techniques are electrocorticography (ECoG) and intracortical neuron recording (INR). All three methods are depicted in Figure 2.2. Although not discussed here, other used non-invasive techniques include functional magnetic resonance imaging (fMRI), magnetoencephalography (MEG) or functional near-infrared spectroscopy (fNIRS) [31].

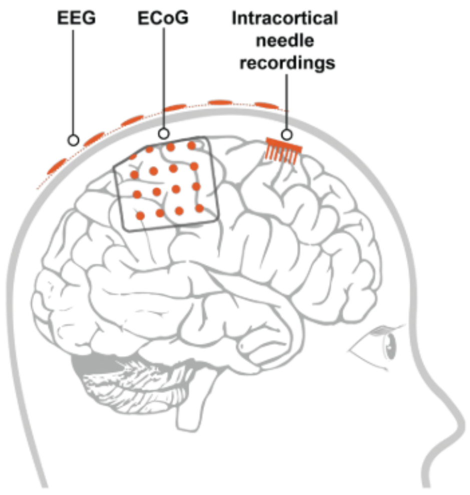


Figure 2.2: Common brain activity acquisition methods: INR, ECoG and EEG. Taken from [30] with permission.

Intracortical neuron recording

INR makes use of microelectrode arrays inserted in the cortex. Many different types of electrodes exist for INR, depending on the specific requirements needed (spatio-temporal resolution, expected lifetime of the device, or number of electrodes). Due to its high spatial resolution (0.05 to 0.5mm), these electrodes permit the recording of single-neuron activity [32]. Most studies have been applying this technology for BCIs with primates allowing for the control of robotics arms with various degrees of freedom [33, 34]. There have also been some BCI studies with tetraplegia patients [35,36]. In [33], the research group was able to make a monkey control a four degrees-of-motion prosthetic arm (three for end-point velocity and one extra for aperture velocity of the gripper fingers) to feed himself. For that, they used a mathematical model to derive the four-dimensional control from the multi-channel parallel activity measured using INR. Being able to read the activity of specific neurons is a major advantage of this method, as opposed to the average activity of a large group of neurons in other acquisition techniques. However, several drawbacks exist that makes INR not the preferred acquisition technique in practical applications. Firstly, it is very invasive, not only due to the required surgery, but also because the micro-electrodes have to penetrate the cortex. It also presents a challenge regarding the long-term stability and reliability of brain activity acquisition. The penetration of electrodes into the cortex leads to several acute and chronic complications: surface vasculature damage, injury of neurons, recruitment of activated microglia, or degeneration of neuronal processes [31]. All this reaction of the tissue around the inserted electrode changes the signal reliability throughout time which is an undesired characteristic.

Very recently, Neuralink [37], a company founded by Elon Musk, has developed an INR solution that addresses many of these issues [38]. The electrodes are flexible and very thin (4 to 6 μm) which increases the complacency with the tissue. Due to the use of a robot that inserts the electrodes (at a rate of 192 per minute) avoiding any surface vasculature, the surgical invasiveness and complications are greatly reduced. Their micro-electrode arrays have two orders of magnitude more electrodes than any other clinically approved technology. In their paper from 2019, they implanted 3072 electrodes in rats and, in late 2020, they presented *Link*, a device with 1024 channels successfully used in pigs and under an FDA approval period for human tests. With this kind of technology and the developments expected in the near future, the preferred paradigm may shift from the current EEG to INR-based BCI systems.

Electrocorticography

Electrocorticography (ECoG), which presents a less invasive technique than INR despite still requiring a surgical procedure, is a method that implants an electrode array subdurally over the cortex. Although the electrodes are not immersed in the tissue, ECoG is able to record neural activity because most cortical neurons are perpendicularly oriented in relation to the surface and due to the synchronization of local neural populations the signal builds up to a detectable amount.

Compared to non-invasive techniques which are separated from the cortex by the skull bone and the scalp, electrocorticography records signals with higher amplitude and signal-to-noise ratio, broader frequency bandwidth, and from a smaller neural population with a spatial resolution in the order of $1mm$ [31, 32, 39].

Subjects from studies using ECoG-based BCI systems managed to control one [40] and two [41, 42] dimensional computer cursors in real time.

Electroencephalogram

Electroencephalogram (EEG) is, by far, the most popular and used signal acquisition method for BCI systems [31, 39]. It measures the weak electrical potentials in the brain (5 to 100 μV [44]) using electrodes placed on the scalp. Its temporal resolution is very high (in the order of milliseconds), it is easy to set up, non-expensive, portable, and a standardized modality due to its use in various fields during the last century. It is possible to measure brain activity from many different places on the scalp simultaneously (commonly up to 256 electrodes [31]) and one of its most important features is that it is not invasive. All these advantages make EEG extremely practical in a laboratory setup. Two disadvantages of this technique are its lacks of a good spatial resolution (around 10mm [32]) and susceptibility to artifacts.

To increase standardization in research and practice, electrode placement layouts have been defined, where the position of each electrode on the scalp is defined in relation to specific anatomical regions on the head. One of the most used layouts is the International 10/20 system as shown in Figure 2.3. This system labels each electrode with letters and a number. The letters identify the lobe: pre-frontal (*Fp*), frontal (*F*), temporal (*T*), parietal (*P*), occipital (*O*), and although no central lobe exists it is also given a letter (*C*). The letter Z is added to the electrode's name to refer to its position on the midline sagittal plane of the skull (*FpZ, Fz, Cz, Oz*) and *A* (or *M*) refers to the mastoid process. The number refers to the location of the electrode within the specified lobe. Even-numbered electrodes belong to the right hemisphere, while odd-numbered electrodes are from the left hemisphere. Other systems that allow for higher spatial resolution exist where more electrodes are placed in between those of the 10/20 system [45].

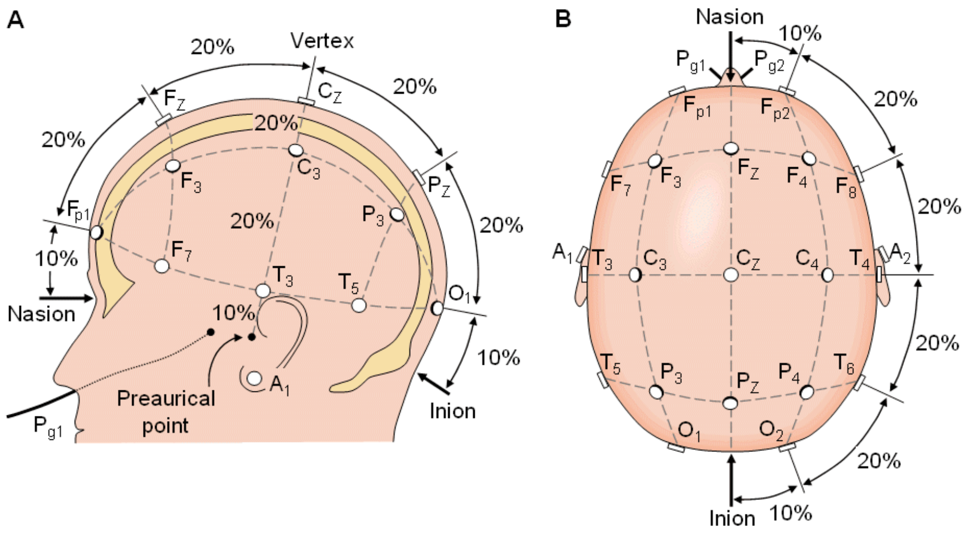


Figure 2.3: International 10/20 system. **(A)** Sagittal view with the skull bone in profile to elucidate the references for the system: nasion and inion. **(B)** Top view elucidating the distribution of electrodes over the scalp. The system gets its name from the relative angular distances between contiguous electrodes. Redrawn from [43].

2.3 Event-related potentials

Event-related potentials (ERPs) are electrical brain signals elicited by a stimulus that may have different origins: sensory (visual, auditory, tactile, etc), cognitive (attention, memory, perception, etc), or motor. ERPs can be registered using different signal acquisition methods such as the ones presented in the previous chapter although EEG is usually used in an investigation context. ERPs are composed of one or more wave components which are characterized by their amplitude, latency, and scalp distribution. Each component is commonly named in the literature according to its polarity (negative or positive) and latency (in milliseconds): *P50, N100, P100, N200, P300*, and so on. A very common ERP component is the *P300* which is elicited around the parietal cortex after a rare and task-related stimulus is presented in the middle of a random series of stimulus events [46]. It has been extensively studied and applied in the field of BCI [47–50]. One frequent experimental setup uses *P300* signals to select items displayed on a computer screen by simply asking the subject to focus the attention on the intended item. The various items are highlighted, one at a time, and when the intended item is highlighted a *P300* signal is elicited.

The present work focuses on Error-related potentials (ErrPs), yet another ERP component which have been much less studied than *P300* but hold promising advances in the field of BCI. Nowadays, ErrP has become an umbrella term in the more engineering-oriented research referring to the various sub-components that may be correlated to error handling in different paradigms. In the early '90s, the error produced by a subject in a speed choice task was correlated with a negative potential ERP in the fronto-central area at around $50 - 100ms$ after the error [51]. This potential is called error-related negativity (ERN) and is followed by another positive potential at the centro-parietal area called positive deflection (Pe).

A similar pattern occurs on a different paradigm where the user is presented with feedback about the previously chosen action. Called feedback-related negativity (FRN), this event-related potential happens in the medial-frontal area and between $200ms$ and $300ms$ after feedback presentation. Similar signals in other tasks such as observing other subjects committing errors, support the idea that ErrPs are elicited when the subject's expectation and the actual outcome do not match [52].

Various studies have tried to use the occurrence of such signals to detect the error perception of the subject when using a BCI [53–55]. The signal is elicited when the user, after being presented with a feedback stimulus, realizes a mistake was committed by the system when trying to identify the users' intention. From now on, the term "ErrP" will be used to refer to this particular feedback error potential. The ErrP presents three main features in its waveform: a positive peak at around $200ms$ after feedback, a large negative deflection between 200 and $250ms$ and another positive peak at about $320ms$ [52, 56]. Figure 2.4 shows an instance of an ErrP wave. This pattern is generated in the anterior cingulate cortex (ACC) [52] and has been reported to be more or less constant even when comparing trials months apart [57].

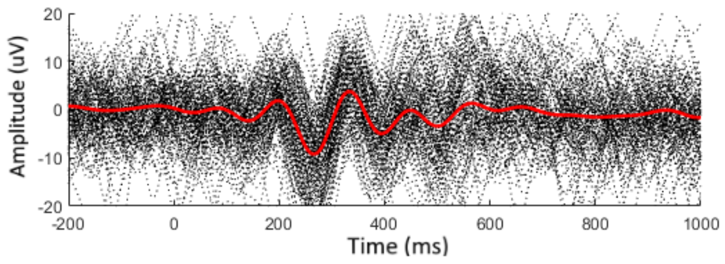


Figure 2.4: Instance of the ErrP waveform obtained in this work. Black dotted lines are single trials, and the red bold line is the grand average. The time axis values are in relation to the feedback presentation instant.

An advantageous feature of ErrP is that it is a physiological signal that occurs naturally during the interaction with Brain-Computer interfaces, contrary to other ERPs that require training or stimulation by the user such as those elicited with a motor imagery paradigm. However, this does not mean that the user's level of attention is not important. In fact, the user should be focused on the task, as the level of attention is described to modulate the error-related potential [52]. Such BCI systems in which the potential is generated naturally are called passive BCIs.

Another advantage of ErrP is that it can be successfully detected on a single-trial basis, as concluded by Chavarriaga *et al.* after reviewing over a decade of literature in ErrPs [52]. Usually, ERP signals are averaged using multiple trials [48] to improves accuracy as it dissipates the random electrical activity and evidences the response triggered by the event. Because ErrP can be classified with single-trials, it is appropriate for correcting the BCI system after feedback presentation. Figure 2.5 illustrates two ways in which error-related potentials can intervene in BCIs in order to address errors caused by the system. On one hand, as shown on the left, the system can use the perceived error to correct the erroneous action. The correction might be choosing the remaining option for binary decision models or the next most likely option in a rank-based system. On the other hand, as shown on the right, the error can be used internally by the model to improve itself and learn to perform better in future classifications. The two solutions are not mutually exclusive as the BCI can also both correct the action and improve its classifier.

2.4 Previous work on ErrP classification

Identifying the presence of error potentials in a BCI context depends on two fundamental steps that must be processed in real-time: feature selection from the EEG signal and classification with a machine learning model. In this section, some of these methods are presented (Deep Learning methods are only presented in the next chapter).

Most of the studies to date make use of temporal features rather than frequency ones, retrieving

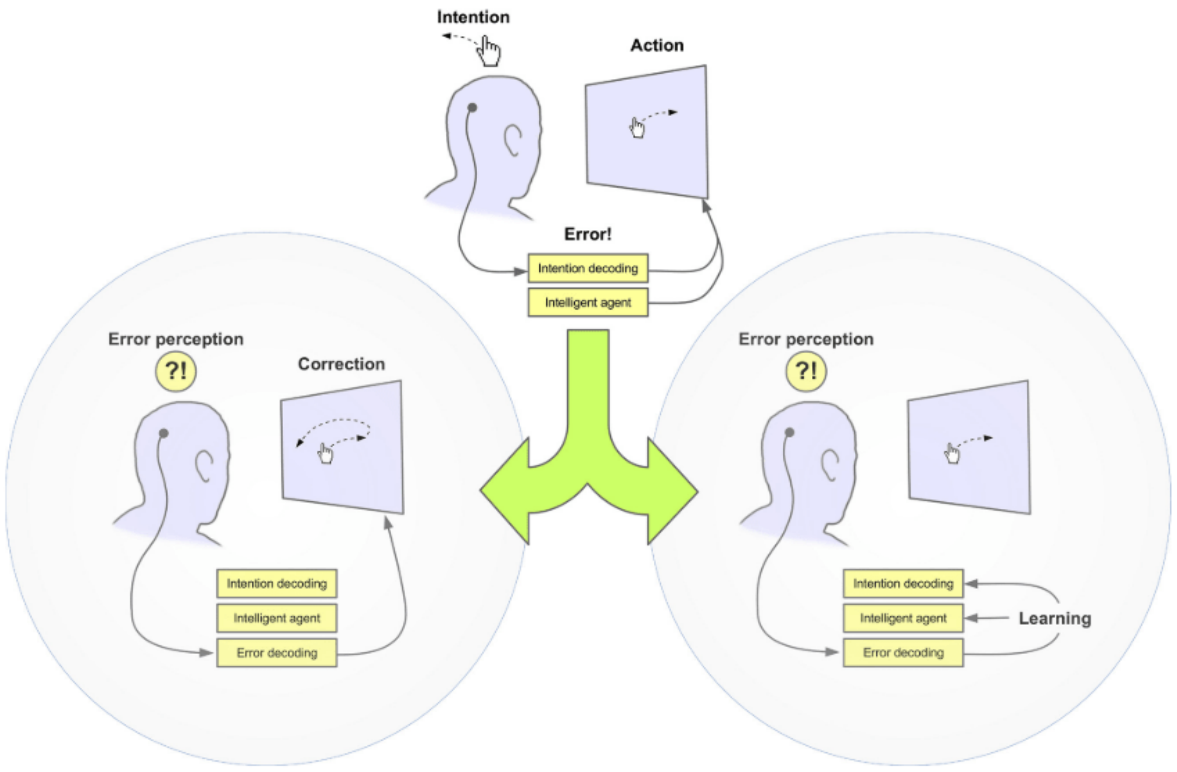


Figure 2.5: Use of error-related potentials to improve BCI systems. **Top:** BCI system mis-classifies the intent of the user and feedback is provided. **Left:** Error is perceived by the user and the single-trial detection of the error-related potential is used to correct the erroneous action. **Right:** Alternatively, the detected ErrP can be used to improve the system's classifier. Taken from [52].

information from the ErrP waveform. Some studies developed ways to automatically select the features by calculating the power of individual features with methods such as the Fisher score, t-statistics, or r^2 [53,58]. These statistical methods end up selecting similar features to those that use manual selection but provide a better way to select features based on the subject's variability. Other studies use different techniques to compute features such as spatio-temporal filters or SVD [58, 59].

Although less common, frequency features are also used and bring interesting results. Omedes *et al.* [60] tested the use of frequency features on classification problems and concluded that frequency features offer a higher generalization power between different tasks than temporal features. However, they also concluded that using both frequency and temporal features offers the best of both domains: higher performance (from the temporal features) and better generalization (from the frequency features). Another study [61] concluded that the error-related potential differences across these tasks are mainly caused by latency, hence supporting the idea that frequency features are less dependent on temporal variations between individual ErrP trials.

After obtaining the features, the proper model must be applied to correctly classify the EEG data. Various Machine Learning techniques have been proposed in different studies such as k-nearest neighbours (k-NN) [62], support vector machine (SVM) [55, 62, 63], Gaussian classifier [57, 63, 64] or linear discriminant analysis (LDA) [61]. Some of these studies try to compare the performance of one method over others. Spüler *et al.* compared LDA with SVM having reported SVM as the best classifier although no metrics were presented [55]. Wang *et al.* also compared LDA with SVM [65] but found no difference when testing the performance on the same subject. If, however, the testing was done on subjects excluded from the training set (which it should always be for a fair model performance analysis) LDA presented better sensitivity although the result was close to random. In another study [62], the authors compared SVM with k-NN and found no relevant difference in the performance of both classifiers.

Ranking the performance of these methods is not feasible as they do not share the same database, pre-processing, feature selection, or evaluation metrics. However, although a fair comparison is not possible, commonly reported accuracies range between 60% and 80%.

3

Deep Learning background

Contents

3.1 Overview

In this chapter, the relevant background knowledge on Deep Learning (DL) is exposed. This *buzz* expression in today's scientific community refers to a family of Machine Learning (ML) methods that emulate, to some degree, the structure and function of a biological brain.

Starting with Artificial Neural Networks (ANNs), the basic concepts of a neuron, weight, bias, activation function, and network are introduced. Then, convolutional neural networks (CNNs) are presented in section 3.3, where the unique way in which CNN architectures make use of weights in the network is detailed. All the relevant building blocks of CNNs are presented: convolutional, pooling and dropout layers, batch normalization, and fully connected layers. Finally, in section 3.5, state-of-the-art using CNNs for ERP classification is reviewed, which serves as a starting point for the present work.

3.2 Artificial Neural Networks

Artificial Neural Networks are network-like computational models that learn to perform tasks by using examples without being programmed in advance [66]. They were inspired by the neural networks present in biological brains. A neuron, as illustrated in Figure 3.1, is a cell that receives electrical inputs, processes them, and transmits outputs. Each dendritic input weights the signal differently and after the accumulated signal reaches some threshold, the neuron creates an action potential that travels from the cell body up to the axon terminals, where it passes the signal onto other neurons that repeat the process. Some neurons work in the network as excitatory cells while others as inhibitory ones. Depending on the connections and weights associated with each neuron, different logic operations are performed.

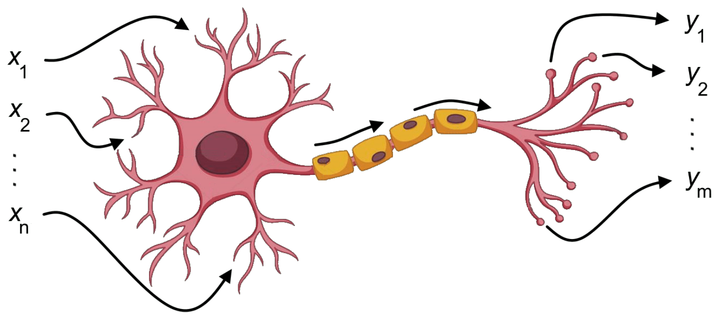


Figure 3.1: Biological neuron as inspiration for the processing unit of ANNs. The input, $\{x_i\}$, is processed and the outputs, $\{y_i\}$, are passed on to the next neuron.

This basic biology knowledge can be used to build Artificial Neural Networks by considering neurons modeled as non-linear functions. After receiving a set of N inputs, $\{x_i\}$, it applies a set of weights, $\{w_i\}$, to the inputs, sums them with a bias term, b, and applies a non-linear activation function, f. Equation 3.1 describes this operation.

The ANN's version of a single neuron that emulates this behavior is called a Perceptron, described in Figure 3.2. It is, as the neuron is for the brain, the building block for most Artificial Neural Networks [66]. The process of unidirectionally passing information from the inputs to the outputs is denominated *forward propagation* and a network that employs such a process is called a *feed-forward network*.

$$y = f\left(b + \sum_{i=1}^{N} w_i x_i\right) \tag{3.1}$$

The operations that add non-linearity to the model are called activation functions. There are many described functions in the literature and each brings its own advantages and disadvantages. Two very simple activation functions are the binary function, where the output is zero if the input is negative or one otherwise (Equation 3.2a), and the sign function which returns the sign of the input (Equation 3.2b). These functions, however, can only output two values that do not reflect the magnitude and change of the input. Therefore, other functions are more commonly used.

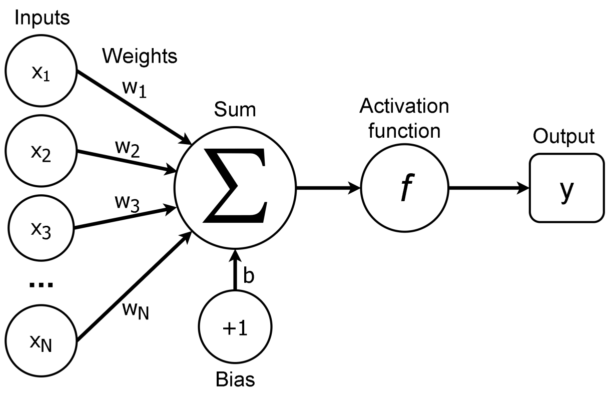


Figure 3.2: Mathematical model representation of the Perceptron (based on Equation 3.1).

One of the most popular activation functions nowadays is the Rectified Linear Unit (ReLU) described in Equation 3.2d. It is computationally efficient and converges faster than other common functions such as the sigmoid (Equation 3.2c) or the hyperbolic tangent (TanH). One disadvantage of the ReLU is that for a negative input, it returns zero which results in ”dead” neurons where the null derivatives computed during backpropagation (covered in section 3.4) prevent weights and biases from updating [67]. To address this behaviour, both the Leaky Rectified Linear Unit (LeakyReLU) proposed in 2013 and the exponential linear unit (ELU) proposed in 2015 can be used [67]. These two functions are expressed by Equations 3.2e (where α is typically 0.1) and 3.2f (where α is typically 1), respectively. Both functions can produce negative outputs, allowing updates to the weights and the biases in both directions. However, ELU is offers faster learning and better generalization then ReLU and LeakyReLU [67].

$$B(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{3.2a}$$

$$sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases} \tag{3.2b}$$

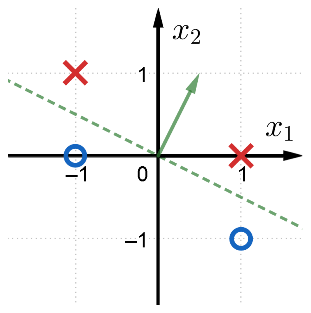
$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.2c}$$

$$ReLU(x) = max(0, x) \tag{3.2d}$$

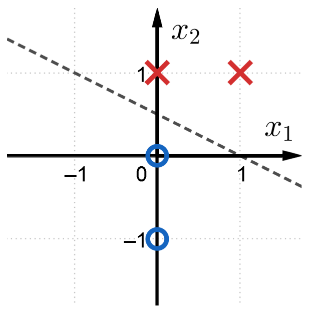
$$LeakyReLU(x, \alpha) = max(\alpha x, x) \tag{3.2e}$$

$$ELU(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases} \tag{3.2f}$$

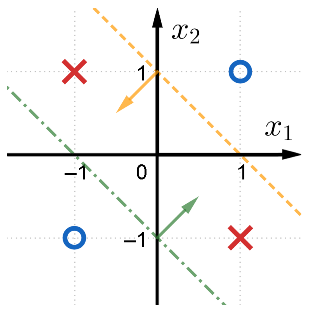
To give a simple example of how ANNs can be used on classification problems, consider a single perceptron with two inputs ($N = 2$) present in Figure 3.4(a). The inputs are the coordinates x_1 and x_2 from the graphs in Figure 3.3. Each marked point in those graphs is an data point that can be fed to the perceptron with an input of the form $\{x_1, x_2\}$. Consider the red crosses as with label 1 and the blue circles as with label 0. Notice, in Figure 3.4, that the perceptron schematics from Figure 3.2 were simplified, merging the summing operation with the activation function, and that now the activation function is defined as the binary function (Equation 3.2a), shown as a step function.



(a) Separation hyperplane passing through the origin.



(b) Separation hyperplane with offset from origin.



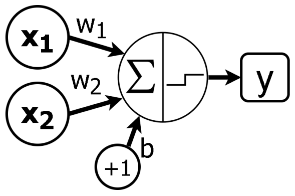
(c) Non-linearizable dataset.

Figure 3.3: Examples of classification problems and simple ANN solutions to solve them.

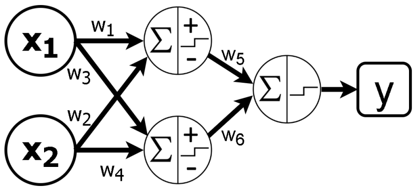
The problem is to know whether or not it is possible to obtain a set of weights, $\{w\}$, such that the value of the output, y, describes the label (0 or 1) of the inputted point. For both graphs 3.3(a) and 3.3(b) it is possible because the different labels are linearly separable by a hyperplane (the dotted line in this $2D$ case) and a perceptron can always classify linearly separable datasets.

To understand how, consider the green vector in 3.3(a) that is perpendicular to the defined hyperplane. To correctly get the label of any data point, first calculate the dot product between this vector and the vector that starts in the origin and extends to the point's coordinate. If this value is positive then the point belongs to class 1, otherwise, it belongs to class 0. This is exactly what the perceptron is doing: it calculates the dot product between the point's coordinates, (x_1, x_2), and the weights vector,

$\{w\}$, (element-wise multiplication and summation) and then applies the non-linear binary function which asks the question *"Is this value positive?"* and returns 1 if so or 0 otherwise. From this logic, it is clear that the weight values must be the components of the green vector. Hence, the solution for Figure 3.3(a) is $w_1 = 0.5$ and $w_2 = 1$ or any other vector perpendicular to a suitable hyperplane.



(a) Perceptron model to classify data points from Figures 3.3(a) and 3.3(b).



(b) Multi-layer network model to solve the non-linearizable problem from Figure 3.3(c)

Figure 3.4: Simple ANN solutions to solve the proposed examples.

However, the bias was not used ($b = 0$) and the reader might be wondering about the necessity of such a node. To understand its importance, consider the example in Figure 3.3(b) where the hyperplane does not intersect the origin. In this case, instead of asking *"Is the dot product positive?"* it should be asked *"Is the dot product greater than some non-zero constant?"*. To allow the activation function to pose that question it should be shifted and that is precisely what the bias does. It adds to the summation so that the dot product is always shifted by a constant amount before being fed to the activation function. To solve the classification problem from Figure 3.3(b) the weights vector is unchanged (as the hyperplane' steepness is the same) and b can be set to any value that offsets the hyperplane between the different classes: $b \in]-1, 0[$.

Looking now at the dataset in Figure 3.3(c) it is clear that the classes are not linearly separable. This is where the single perceptron can no longer classify the data points. Most real-life problems are also non linearly separable and hence, the simple idea of the perceptron is not enough.

The solution to this limitation is to build multilayer perceptrons (MLPs), networks made up of several perceptrons (from now on denominated *nodes*), mimicking the fact that biological neural networks are constituted by linked neurons and not isolated ones. MLPs are organized into three types of layers known as *input layers*, *hidden layers* and *output layers*. The input layer is the set of nodes that hold the input values. The output layer presents the output, which is not always a single value like before, but might also be a vector of values. In between these two layers, one or more hidden layers exist each of which might have as many nodes as necessary to solve the problem.

To solve the classification problem in Figure 3.3(c) (which is equivalent to the XOR problem) the network shown in Figure 3.4(b) is used (for simplicity, the representation of the bias weights are omitted). It adds one hidden layer with two nodes to the previous single-perceptron solution. Notice that different activation functions can be used within the same network: the two hidden nodes now perform the sign function (as shown by the $+/-$ signs) while the activation function for the output note remains the binary function. In this network, each node in the hidden layer can divide the Cartesian plane individually (two hyperplanes are represented). Then, a linear combination (with the weights w_5 and w_6) of those two hyperplanes provide the correct answer at the output. One set of weights that works in this case is $w_1 = 1$, $w_2 = 1$ and $b_1 = 1$ for the green dotted hyperplane, $w_3 = -1$, $w_4 = -1$ and $b_2 = 1$ for the yellow dashed hyperplane and $w_5 = 1$, $w_6 = 1$ and $b_3 = -1$ for the combination of both.

With this basic knowledge, solving harder classification problems is only a matter of adding enough hidden layers to the network. Although computational models process information much faster than animal brains because biological neurons use diffusion to transmit the signal, they are much more limited in the number of both neurons and connections a single neuron can perform, simulating the brain organization in a very gross manner. As a reference, the human brain has around 8.3×10^9 neurons and 6.7×10^{13} connections, almost 10000 synapses per neuron [68]. Nevertheless, from simple networks and only allowing for very simple mathematical operations, ANNs can solve complex, ill-defined, and highly non-linear problems [66].

3.3 Convolutional Neural Networks

In Machine Learning, both networks represented in Figure 3.4 are called Shallow Neural Networks where one or none hidden layers are used. This type of neural network contrasts with another type in which not one but multiple hidden layers are added and that is conveniently called Deep Neural Network from where the field ”Deep Learning” also draws its name.

The addition of more hidden layers is necessary to train models in harder problems and bigger data. It was shown by Cybenko and other independent mathematicians that, theoretically, a neural network can arbitrarily approximate any continuous function of n variables with a single fully-connected layer given that it has sufficient nodes [69]. However, developing architectures with multiple hidden layers is the most used practice. As problems become increasingly complex and the inputs become larger the number of trainable parameters in the network also increases. This increase is eventually limited by memory or processing capabilities, forcing a compromise between performance and the available resources.

Image classification is a classic example of such a complex problem, where a large number of images needs to be classified according to some criteria like the presence of an object or animal. The problem is complex because differentiating between a dog or a cat, for example, certainly requires more than a couple of hyperplanes in traditional ANNs which translates into a large number of hidden layers. On the other side, the input also poses a problem because images are very big data structures. They usually are composed of three channels to encode for color (RGB) and each channel is a 2D matrix of pixels. A colored image defined by a grid of 500×500 pixels would have $500 \times 500 \times 3 = 750000$ data points as inputs to the network. If each hidden layer would have the same number of nodes as the input, then a network with only 5 hidden layers and 2 output nodes would have $5 \times 750000^2 + 750000 \times 2 \approx 2.81 \cdot 10^{12}$ trainable parameters which, if stored as 4-byte integers, would occupy more than $11TB$ of memory.

Convolutional neural networks are a type of Deep Neural Network that solve the problem just mentioned above. Its architecture drastically decreases the number of weights in the network due to three main concepts that will be elaborated in a further section: sparse interaction, parameter sharing, and equivariant representations.

It must be noted that although CNNs were first designed for image classification and that this is still the main use of the architecture, it is also possible and useful to use it for time series classification which is the intended use in the present work. A recent study [70] tested 18 time series classification algorithms and none was based on a convolutional neural network model (not even a Deep Learning model). This comes to show that the scientific community lacks an understanding of the benefits that convolutional neural networks can bring to time series classification.

3.3.1 Convolutional layers

In this subsection, the concept of convolutional layers is introduced. Some of its characteristics are also explained, such as sparse interactions, parameter sharing and equivariant representation. Besides the kernel size, two other convolutional parameters are also presented: padding and stride.

Convolutional operation

The most important building block in convolutional neural networks is the convolutional layer. When discussing Artificial Neural Networks, Equation 3.1 only considered one output node and to calculate its value, y, an element-wise multiplication was applied between the input, $\{x_i\}_{i=1}^N$, and the weights, $\{w_i\}_{i=1}^N$. When considering more than one output node ($M > 1$), the values of those nodes are given by the following matrix multiplication:

$$\begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_M \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,N} \\ w_{2,1} & w_{2,2} & \dots & w_{2,N} \\ \dots & \dots & \dots & \dots \\ w_{M,1} & w_{M,2} & \dots & w_{M,N} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_N \end{bmatrix} \tag{3.3}$$

Using this traditional approach each hidden layer has $N \times M$ trainable parameters. As exposed above, depending on the input size and amount of hidden layers, the number of trainable parameters might be too high for the computational resources available. To decrease the number of parameters the idea of matrix multiplication can be replaced with that of convolution which is a mathematical operation performed between two tensors: the input tensor and a tensor called kernel. Considering the common case in CNNs where the input is an image represented by a 2D matrix then the convolution happens between the image, I, and the kernel matrix, K. The output of the operation, O, is given by Equation 3.4 (where $*$ denotes the convolution operation).

$$O_{i,j} = (I * K)_{i,j} = \sum_m \sum_n I_{i-m,j-n} K_{m,n} \tag{3.4}$$

The convolution calculates the output by swiping the kernel matrix over the input matrix and, at each new position, computing the element-wise multiplication between the kernel and the region of I covered by the kernel. Figure 3.5 shows a visual representation of Equation 3.4 and Figure 3.6 shows the application of three common kernels in image processing: an edge detector (3.6(b)), a Gaussian filter (3.6(c)) and an image sharpener (3.6(d)). The outputs are obtained by applying Equation 3.4 with the kernels given below each image. These filters have specific values that were "hard-coded" to obtain the desired output. However, during the training of CNNs the kernels start with a random set of values and evolve to improve on the problem goal. This topic is further discussed in section 3.4.

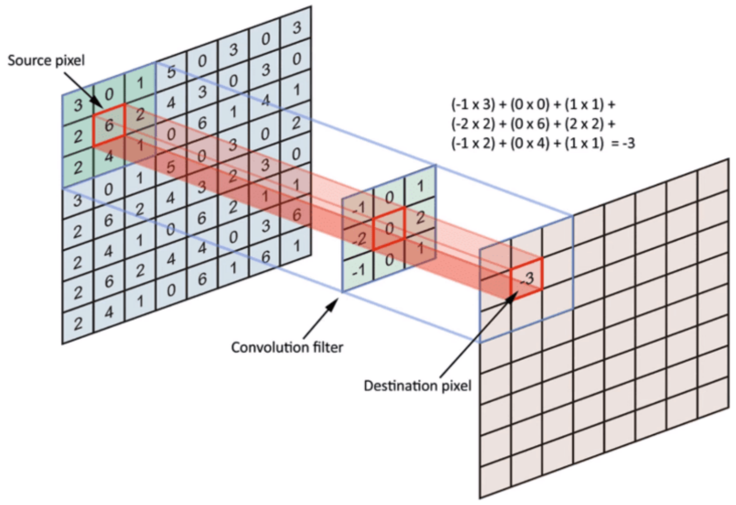
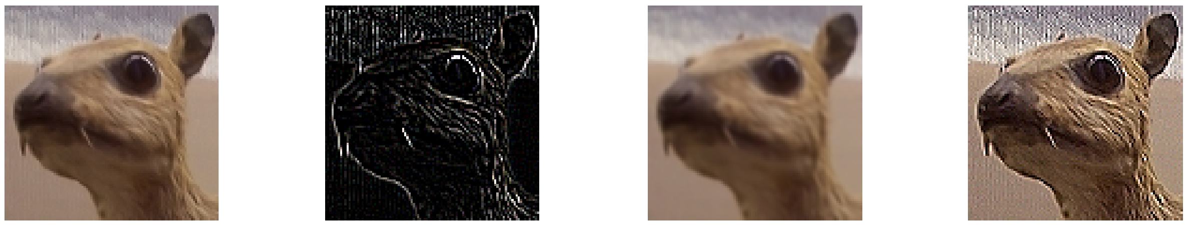

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
Destination pixel

Figure 3.5: Visual representation of a convolutional operation. After the first element-wise multiplication is done in the current position of the kernel, it slides to a new position and repeats the calculations to generate a new value on the output matrix.

In this approach, the trainable parameters are the values of the kernel matrix. Using Figure 3.5, a quick comparison can be done between the number of weights necessary in the traditional matrix multiplication and the convolution method. The kernel in the Figure is a 3×3 matrix and thus the convolution method only requires 9 weights. Both the input and the output are 8×8 matrix images and thus the traditional matrix multiplication requires $8^2 \times 8^2 = 4096$ weights (on a network without hidden layers). This considerable decrease in the number of parameters is the consequence of two important concepts in convolutional neural networks: sparse interaction and parameter sharing.



$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

(a) Original image **(b)** Edge detection **(c)** Gaussian blur **(d)** Sharpener

Figure 3.6: Feature extraction from convolution applying various kernels. Image 3.6(a) is the original image and the matrix below is the identity kernel which does not alter the input. Taken from [71]

Sparse interactions, parameter sharing, and equivariant representation

Sparse interactions refer to the decrease of connectivity between the input nodes of a layer with its output nodes. As seen above, in traditional ANNs the matrix multiplication used for the feed-forward operation forces an interaction between all input nodes and all output nodes: input node i connects with output node j through weight $w_{i,j}$. In CNNs, however, one output node only connects with the input nodes required by the kernel and since the kernel is usually much smaller than the input there are many more unconnected nodes than the connected ones.

Parameter sharing refers to the re-usability of the same weight set for the calculation of different output values. When performing a convolution, the kernel remains constant as it traverses the input and hence each output value is the weighted sum of a new set of inputs with the same set of weights given by the constant kernel itself.

Although an important feature, decreasing the number of weights in a network is not the only advantage of CNNs. Let us consider again the traditional Artificial Neural Network now trying to classify the image of a dog in a park. At the first hidden layer, all the pixels of the image will be summed together with different weight combinations for all the outputs in that layer. However, combining a pixel from the background (the park) with a pixel from the head of the dog does not make much sense as these pixels are unrelated and will not help to classify the image as a "dog". In convolutional neural networks the kernel forces the values of the output to be a weighted sum of contextually-related pixels in small regions of the input, such as the eyes, the tail, or the snout.

Furthermore, it is also desired that if in another image the dog moves, to the left for example, the classification remains the same ("dog"). Indeed, this is also a property of CNNs present in the convolution operation called equivariant representation [72]. A function is said to be equivariant if, when the input changes, the output changes in the same way. Specifically for time-series classification, this allows the model to identify a set of features from the signal independently of where they occur. Features such as noisy or smooth regions, spikes or valleys, etc, can be detected regardless of their position in the input signal. Note that the equivariance mentioned above only applies to translations, not rotations nor scaling.

Padding and stride

From Figure 3.5 it can be seen that the first output value calculated (-3) is not placed at the top-left most corner but in position $(1, 1)$ (considering a 0-indexed system of coordinates). After the kernel convolves with the entire input image, the output image will have an empty margin of size 1 all around. This happens because a kernel of width 3 can only occupy 6 different regions along the horizontal direction in an input image of width 8. The 2 pixels whose values are not computed are the borders and this effectively decreases the output size by 2 in each dimension (width and height) turning the 8×8 input into a 6×6 output. If this decrease is not intended, padding can be applied to the input image which adds extra pixels around the original input as shown in Figure 3.7. The number of added border layers is the padding size, p, where $p = 0$ means an un-padded matrix.

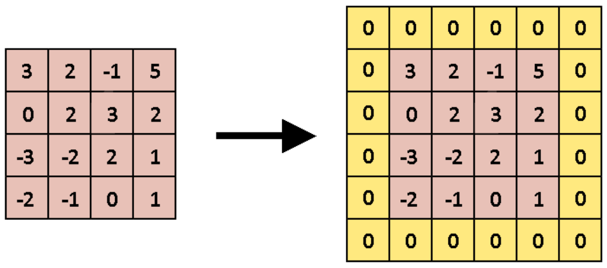


Figure 3.7: Zero-padding added to matrix. Added padding ($p = 1$) is shown in yellow. Besides zero-padding other padding techniques exist such as reflective padding.

Another common parameter is the stride, s, which indicates the step size of the kernel when convolving. Usually, $s = 1$ meaning that the kernel must evaluate convolutions at all possible positions since the step size is unitary. If $s = 2$, then only half of the positions will be convoluted as every other pixel is skipped. For one dimensional data, the size of the output after convolution is given by

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1, \tag{3.5}$$

where n_{in} is the number of input features, k is the kernel size and $\lfloor x \rfloor$ represents the floor rounding operation of x. For multi-dimensional data, Equation 3.5 can be applied to each individual dimension.

Maps

One kernel might already extract relevant information, but the network needs multiple features to accurately classify the input. For that reason, each convolution layer usually makes use of more than just one kernel matrix. Each output of a kernel is called a map or a feature (exactly because it described a new feature of the input). On the following convolution layer, the kernel convolves with all the maps in the input as if the maps were an extra dimension of the data.

3.3.2 Pooling layer

The pooling layer is a stage usually applied after the activation function of convolution layers. Its main purpose is to reduce the number of trainable parameters by applying a function that compresses the size of the input. Multiple functions can be used and each value of the output is obtained by applying the function at a local neighborhood of the input. A very commonly used function is the max function which returns the maximum value within the set of input values. Other functions can be used such as the min, the $mean$, or a weighted sum of the input based on the distance to the center.

The final size of the output is given by

$$n_{out} = \left\lfloor \frac{n_{in} - f + s}{s} \right\rfloor, \tag{3.6}$$

where f is the pooling region size and s is the stride. Again, the one dimensional formula presented can be applied in higher dimensions. Since in pooling layers the stride is usually equal to the pooling region size ($s = f$), the total output size is generally reduced by a factor equal to f. The minimum (and most common) pooling size is $f = 2$ which already halves the size of the output in the one-dimensional case (for higher dimensions, the reduction is of f^d, where d is the dimension size). Figure 3.8 depicts the application of a 2×2 max pooling layer with stride of 2 on a 2D matrix of size 4×4 which results in a 2×2 matrix ($2^2 = 4$ times smaller).

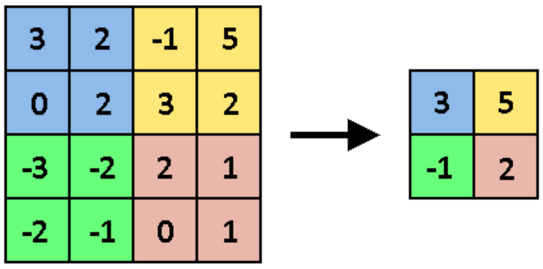


Figure 3.8: Pooling operation. Max pooling kernel of size 2×2 is applied to input 4×4 matrix. A stride of 2×2, the same size as the kernel, is applied preventing overlapping on the pooling operation. Each color on the input represents a region where the pooling was applied and the respective result (maximum value from that region) on the output.

Besides reducing the number of trainable parameters downstream in the network, pooling layers present another important property: local translation invariance. This is a similar property to equivariant representation but happens locally because the values in the output are sampled from a neighborhood in the input which might have been the center value, one of the extremes, or any other value in between. Local translation invariance is a useful property in the case where knowing the presence of a feature is more important than knowing the precise location of the feature [73]. For example, in the detection of event-related potentials (such as ErrPs or P300's), it is not expected that all waveforms have its onset exactly at the same time after stimulus presentation. Therefore it is useful to introduce some time invariance in this context.

3.3.3 Fully connected layer

A typical CNN model is composed of two parts: first, one or more modules made up of a convolution layer, the activation function, and a pooling layer, and then, one or more fully-connected (FC) layers. This last type of layer works just like a regular matrix multiplication in ANNs meaning that every input connects to every output. The purpose of this layer at the end is to linearly combine the learned features from previous layers. Before being fed into the FC layer, the data structure is re-arranged (flattened) from a tensor to a vector form whose size is equal to the product of the dimension sizes of the tensor.

The final output structure of a CNN model can vary but usually, for classification problems with C classes, it is composed of C nodes. In the present work, $C = 2$ as the two only possible labels are "ErrP is present" or "ErrP is not present" (binary classification problem). In this method, where the vector of labels is used as an output, the fully-connected layer learns the correlation of the high-level features learned with each of the individual label nodes. Indeed, many CNNs used for binary classification adopt two output nodes to follow the general case of any number, C, of classes. However, for the binary case, the output vector only requires one node which, after defining a threshold for its value, can represent two different labels.

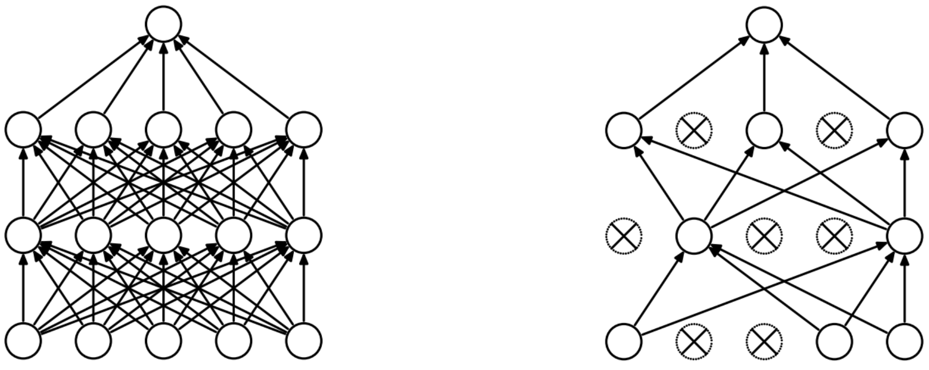
3.3.4 Dropout layers

When a complex network is trained with a small dataset it may lead to overfitting, i.e., instead of the weights being trained to generalize a problem, it highly adjusts to predict well samples on the training set. This makes the network perform badly on the test dataset.

To prevent this behavior dropout layers were developed in 2012 [74]. This technique consists of ignoring a certain percentage, p, of nodes when performing the forward propagation. On each forward propagation, a new random set of nodes is selected and their values are zeroed so that, although multiplied by the respective weight, they are not taken into account by the nodes in the next layer. Figure

3.9 shows a representation of this concept: in the network on the left all nodes are considered while in the network on the right about half of the nodes were zeroed and their contributions are not taken into consideration when forward feeding the network.

Intuitively, a dropout layer is equivalent to a mechanism that randomly changes the topology of a network (by defining the presence or absence of nodes) so that noise is introduced into the small dataset, thus preventing the network from ”memorizing” samples and being able to generalize the dataset.



(a) Network where dropout is not applied and all nodes contribute to the next layer. **(b)** Network where dropout is applied, preventing some nodes from contributing to the next layer.

Figure 3.9: Application of dropout to a Neural Network. Taken from [75]

It is important to note that dropout layers are only active during the training phase. When the network is used for testing new samples the dropout layer is inactivated $(p = 0)$ as all nodes are needed for the actual classification.

3.3.5 Batch normalization

Batch normalization (BN) is yet another type of layer that can be applied to a CNN model to improve the training process. It was first introduced in 2015 to mitigate a problem the authors called internal covariance shift in which different distributions in the data at each layer affect the learning speed of the network [76].

To understand this problem at each hidden layer, consider first the problem of inputting data into a neural network that contains different types of information in a variety of different ranges such as inputting the age and height of a person to predict something about that person. Age can generally range between 0 and 100 years while height can range between 0 and 2 meters. These two parameters have very different distributions with different means and variances and the model will firstly need to learn these distributions before starting to actually learn meaningful correlations between the variables. Therefore, normalizing both variables such that they range between 0 and 1 helps the model not having to train on each of the distributions.

Now, after this input is given to the model, the same problem can arise: the trained weights between the input and the first hidden layer may create a new distribution for that layer. For that reason, a new normalization may be needed. This normalization is applied to each mini-batch (a random subset of the dataset), B, hence the name ”batch normalization” and to each dimension of the input. The normalization is computed as

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\sigma_B^{(k)^2} + \epsilon}} * \gamma + \beta, \tag{3.7}$$

where $x_i^{(k)}$ is the k^{th} dimension of i^{th} element in the mini-batch, $\mu_B^{(k)}$ and $\sigma_B^{(k)^2}$ are, respectively, the mean and variance of the mini-batch B and ϵ is an arbitrarily small constant to avoid numerical instability. The values β and γ define the new mean and standard deviation for the distribution and are also trainable parameters.

One question that is still on debate is the best position to apply batch normalization. In the original paper [76], the authors used the BN layer just before applying the non-linear activation function and this is also the way many people in the DL community have been using this layer. Depending on the problem and chosen architecture, the best placement of this layer may change [77].

3.4 Training

In the previous sections, some of the parts that make up the convolutional neural network network were presented. After selecting the model's architecture, it must be trained to achieve the desired goal.

Training is the process of tuning the model such that its predicted output, $\hat{y}$, is the most similar to the expected output, y. Since the model is merely defined by the set of its weights and biases then training is, more concretely, the process of finding the best set of weights and biases for the network.

Convolutional neural networks do this in a supervised manner by using examples with outcomes known *a priori*. A dataset of N such examples (with inputs, x, and their expected output, y) is split into three different sets: the training set, the validation set, and the test set. The first is used to train the model and since a lot of data is needed to do this, the training set is usually the biggest (around 70% to 90% of the whole dataset). Each parse of the complete training set through the network during training is called an epoch. The validation set is used to assess the performance of the model during training by presenting data that the model never trained with. This is useful to avoid the problem of overfitting explained ahead. Finally, the test set is used to assess the performance of the model after training.

During the training process, the input is fed into the model, the output is calculated and compared with the expected output. From this comparison a loss value is generated, where a high loss means the calculated and expected outputs disagree and a low loss means the calculated output is close to the expected output. The goal is then to minimize the loss. Considering the loss as a function of the weights and biases that make up the network then, the problem of training the model becomes an optimization problem where the best set of weights and biases needs to be found to minimize an objective function. A method called backpropagation was developed in the 1970s to search for this minimum [78].

Backpropagation, as the name indicates, propagates the error at the output (the loss) in a backward fashion from the output nodes until the input nodes updating the values of the weights to better adjust to the expected output. It applies the chain rule to calculate the derivative of the loss function with respect to each weight on previous layers, $\nabla_w L(w)$. It must be noted that the loss function is not globally known but only on the calculated points. Hence, the only way to search for the minimum of the loss function is to rely on the derivative calculated by the backpropagation at each training step. In Machine Learning, the algorithms that adjust the weights of the model are called optimizers (introduced ahead).

In Figure 3.10 the loss of both the training and validation sets are plotted during training. The loss for the training set usually decreases monotonically (not considering the effect of noise) meaning that the weights of the network are being updated in such a way that the calculated output from the training examples becomes concordant with the expected outputs. The validation loss, however, follows a different pattern. It decreases initially but starts rising again at a certain point. This rise is known as overfitting and happens because the network is only being trained on the training set and not on the validation set. First, the model learns to generalize the solution (lowering both the training and validation losses) but then it highly adjusts (overfits) to the given training examples affecting the generalization to new unseen examples. Since the goal is to have the highest generalization possible, the final model should be the one at the iteration with the lowest validation loss (around epoch 180 in Figure 3.10).

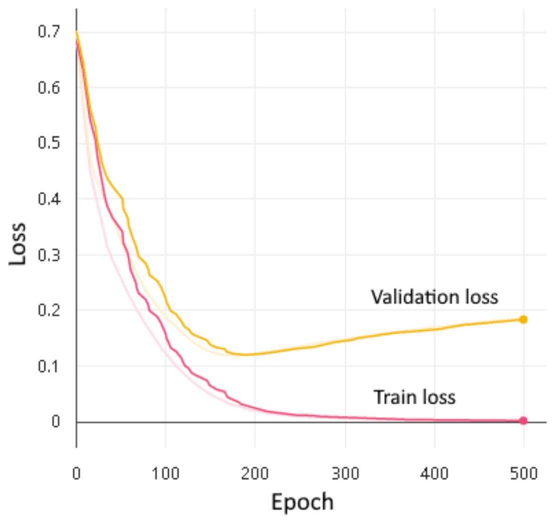


Figure 3.10: Loss for validation and test sets.

Optimizers

Consider the trivial one-dimensional optimization problem in which the value of x must be such that it minimizes the function $f(x) = x^2$. Considering that x is initialized as a random value different from the solution ($x = 0$), then x can be updated by traversing against the gradient by some amount proportional to the gradient: $x = x - \alpha \nabla_x f(x)$, where α is called the learning rate. For example, if initially $x = 1$ and $\alpha = 0.2$ then the gradient is equal to $\nabla_x f(x) = 2$ and x can be updated as $x = 1 - 0.2 \times 2 = 0.6$ which is closer to the minimum.

This method, called Gradient Descent, is a basic optimizer that can be used to train CNNs. Instead of just one variable, x, there are as many as the number of weights in a network and the function to minimize is the loss function, $L(w)$. Hence, in the Gradient Descent optimizer, the weights are updated in a very similar manner to the previous example [79]:

$$w_t = w_{t-1} - \alpha \nabla_w L(w_t), \tag{3.8}$$

where the subscripts t and $t-1$ refer to the current and previous training steps, respectively. One problem with Gradient Descent is that it only updates the weights after each epoch which leads to large updates since the final gradient is the accumulation of all the gradients for each example in the dataset. However, finer updates are desired to properly search for the minimum of the loss function.

To solve this issue, the weights can be updated after each new input instead of only at the end of each epoch. This optimizer is called Stochastic Gradient Descent (SGD) and uses the same update formula as the regular Gradient Descent. A problem of using SGD after each new data point is that this introduces noise into the training process because an atypical input (outlier) may correct the model in the wrong direction. To avoid this, instead of updating the weights after every single input, the training set can be split into mini-batches and the weights are only updated after traversing each mini-batch. This method is called Mini-batch Gradient Descent although it is often referred to as SGD accompanied by the size of the mini-batch. This size influences both the training time and the noise introduced.

A way to further increase speed and decrease the effect of noise is to add the concept of momentum. In a mini-batch, it is likely that most examples follow the same pattern and only eventually one example may follow a different pattern. Momentum keeps the direction of the update more or less stable even if some outlier example is presented. The update is performed as follows:

$$v_t = \beta v_{t-1} + (1-\beta) \nabla_w L(w_t) \tag{3.9a}$$

$$w_t = w_{t-1} - \alpha v_t, \tag{3.9b}$$

where β is the momentum constant (usually set to 0.9). An acceleration component can also be added to further control the update of the weights.

So far, with the optimizers presented above, all the weights are updated using the same learning rate.

This can also be changed by using adaptive learning rate optimizers. Adagrad is one such optimizer that achieves this by dividing the original learning rate by the sum of the squares of the gradients for each individual weight (indexed i) until the current iteration, t:

$$G_{t,i} = G_{t-1,i} + \nabla^2_{w_{t,i}} L(w_{t,i}) \qquad \text{(3.10a)} \qquad\qquad w_t = w_{t-1} - \frac{\alpha}{\sqrt{G_{t,i}}} \nabla_{w_{t,i}} L(w_{t,i}) \qquad \text{(3.10b)}$$

However, as $G_{t,i}$ becomes larger with each iteration, the modified learning rate parameters tends to zero slowing down the learning process. Other optimizers such as Adadelta tackle this particular issue by introducing a γ parameter in equation 3.10a reducing the effect of past iteration by an exponential factor.

Yet another very famous algorithm is the Adam optimizer which is also an adaptive learning rate optimizer. It is similar to Adadelta but adds on top the idea of momentum introduced before. Although it is one of the most used optimizers nowadays it is also said to have a worse generalization effect than SGD when testing on new data [80].

One other way to update the learning rate dynamically during training is to use a learning rate scheduler. It changes the learning rate without interfering with the update formulas following a defined rule. For example, one can define that after every 5 epochs the learning rate should decrease by a factor of 0.1 for a finer search of the optimum weights. Similarly, the learning rate can be modified based on other criteria such as the occurrence of plateaus in the loss function meaning that the training has stagnated.

The best optimizer choice is not always obvious. On one hand, adaptive optimization methods such as Adam provide faster convergence speeds by allowing the rates to change as the learning progresses, and also because of that, it requires less fine-tuning of the parameters. On the other hand, these methods are reported to have a worse generalization capability than SGD or to prevent from converging due to unstable and extreme learning rates [80]. That may be one of the reasons why still so many recent studies make use of SGD with a simple learning rate scheduler [79].

3.5 State-of-the-art

In the previous chapter (Brain-computer interface background) some ML methods used to classify error-related potentials were introduced. In the present section, previous work using convolutional neural networks for this same classification problem is reviewed.

Table 3.1 lists CNN models that are reviewed in this section. It includes models for *P300* classification because of the shared nature between ErrP and *P300*. Both signals are event-related potentials representing brain electrical activity. A model trained to classify *P300* and that performs well is a model that extracts the relevant features of an ERP signal and then combines them for an accurate classification. The model does not understand what originated the signal, the physiological pathways that gave

Table 3.1: Listing of CNN models used to classify both ErrP and P300 signals.

Year	Authors	Target ERP	Model name
2019	Bellary *et al.*	ErrP	ConvArch
2018	Torres *et al.*	ErrP	ConvNet
2018	Luo *et al.*	ErrP	CNN-L
2018	Shan *et al.*	P300	OCLNN
2017	Liu *et al.*	P300	BN3
2015	Manor *et al.*	P300	CNN-R
2011	Cecotti *et al.*	P300	CCNN

rise to it, or any other abstract concept about the signal. Hence, if a CNN model performs well for *P300* it should, in principle, also perform well for ErrPs.

A very important factor for the success of a CNN classifier is the architecture of its model [81] and that is where most of the previous work has focused on: building upon previous architectures, changing the number and type of layers and adding new advances in the field of DL. Most of the model name abbreviations presented in this section are taken from the paper of Shan *et al.* [25]. An exception is the model from Luo *et al.* [82] whose model name is not given and is here called *CNN-L*.

P300 models

The first CNN models built to classify P300 for BCI purposes used similar architectures. The input is a $C \times N$ tensor, where C is the number of EEG channels used and N is the number of time samples. It is fed into the CNN, whose architecture has three distinct stages: first, a spatial convolution along the dimension of the channels, then a convolution along time to learn temporal features, and finally fully-connecteds layers to correlate all the extracted features.

The first implementation of such an architecture was proposed by Cecotti and Gräser in 2011 [81]. Figure 3.11 is taken from the original paper and depicts the architectural organization of the *CCNN* model. In this case, the input has $C = 64$ electrode channels and $N = 78$ time samples (the signal length in seconds depends on the sampling rate used to acquire the dataset). The first convolutional layer uses 10 feature maps while the second uses 50. Then, two fully-connected layers follow, leading to an output with two nodes.

In 2015, Manor *et al.* proposed an improvement to the *CCNN* model by allowing the network to be deeper and wider (Figure 3.12). The model uses more feature maps with smaller kernel sizes and more neurons for the fully-connected layers (2048 and 4096). Furthermore, it also makes use of pooling and dropping layers to compensate for the larger architecture, thus avoiding overfitting.

Liu *et al.* improved on the *CCNN* model by adding BN layers [84]. The network is called *Batch Normalization Neural Network*, or *BN3* for short (architecture shown in Figure 3.13). One BN layer is applied directly to the input which was not normalized during pre-processing and another to the second

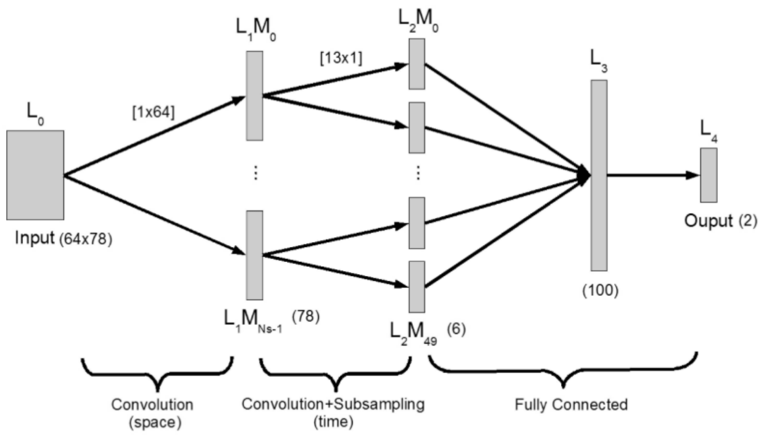


Figure 3.11: CCNN architecture. ”The number of neurons for each map is presented between brackets; the size of the convolution kernel is between hooks.” Taken from [81].

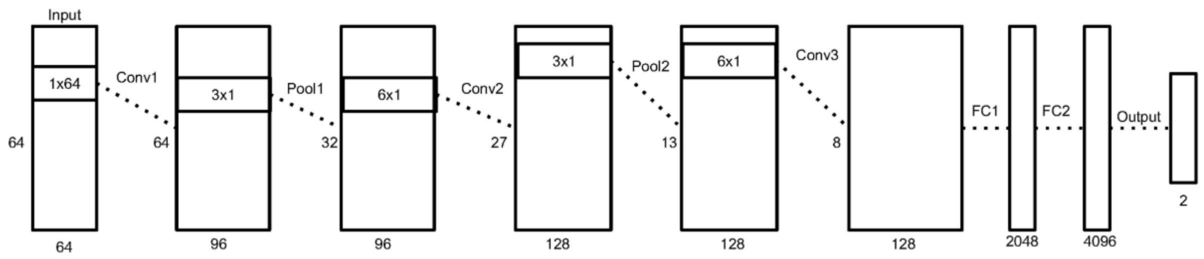


Figure 3.12: CNN-R architecture. ”Each box represents the data matrix going through the network. For the convolution and pooling layers, we note the number of time samples on the left side and the number of output filters at the bottom. The dimensions inside the box represent the filter size of the following convolution or pooling layer. Before the first fully connected layer, the 2D data matrix is reshaped into a 1D vector.” Taken from [83].

convolution layer. Dropout layers and three FC layers are also used to, respectively, avoid overfitting and increase generalization.

One year later, in 2018, Shan *et al.* identified a problem with the sequence of convolutions used in previous models (namely *CCNN*, *CNN-R* and *BN3*), observing that the first convolution was always performed on the spatial domain [25]. The authors argued that this prevented the CNN models from correctly learning temporal features because the spatial convolution transforms the original EEG data into abstract data. The temporal convolution that follows ends up taking as input this abstract temporal data instead of raw temporal signals. Furthermore, the authors say that as a consequence, the models require deeper and wider architectures to better apprehend temporal features, thus presenting higher complexity. The model they propose as a solution, called *OCLNN* (standing for ”one convolution layer neural network”), performs both temporal and spatial convolutions in the first layer (Figure 3.14). Because better temporal features are learned, only one FC layer is necessary to achieve better accuracy than in previous models, significantly reducing the architecture’s complexity.

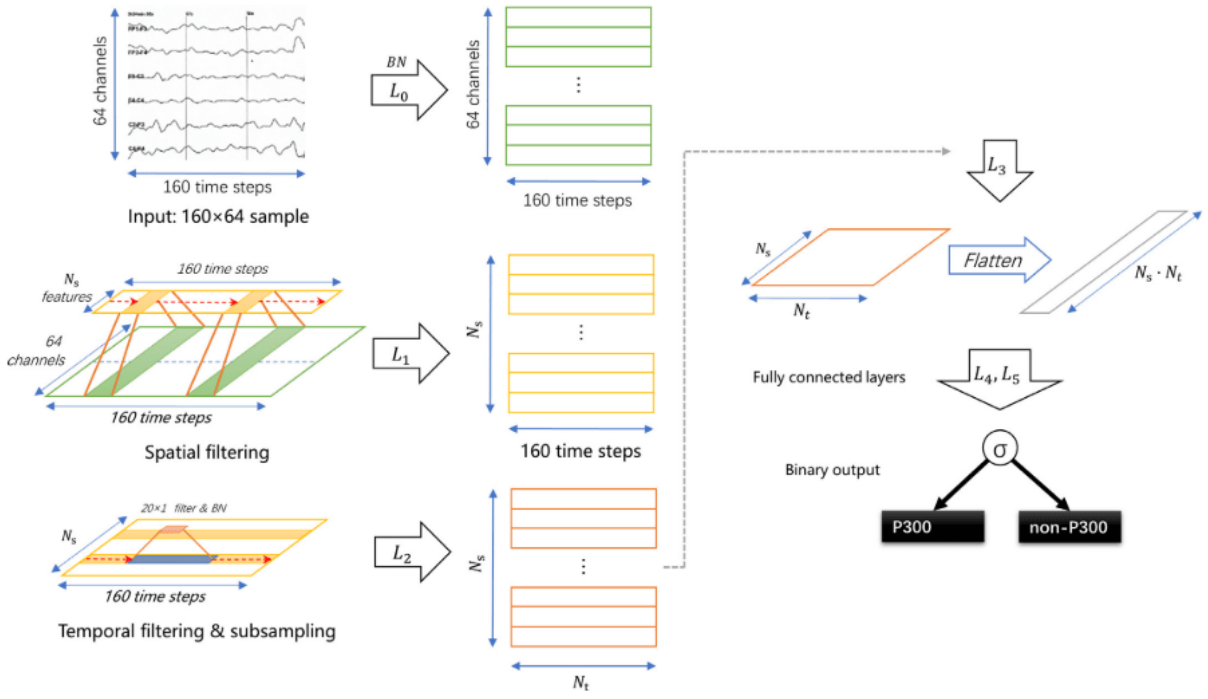


Figure 3.13: BN3 architecture. Taken from [84].

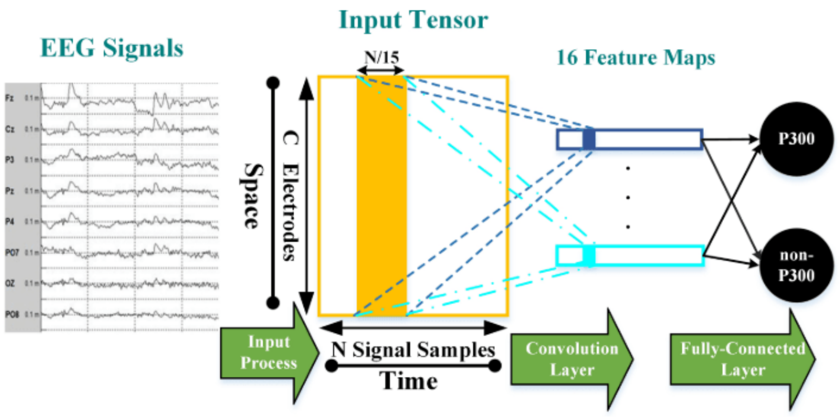


Figure 3.14: OCLNN architecture. Taken from [25].

ErrP models

So far, all the models presented are used for *P300* classification. In more recent years, CNN models for ErrP classification were also developed.

In 2018, Luo *et al.* [82] proposed a model which, contrarily to all the previous *P300* classifiers, applies temporal convolution in the first layer and only successively applies spatial convolution (Figure 3.15). After the convolutions, the model applies BN, average-pooling, and a 50% rate dropout before the FC layer that outputs two-class nodes. This particular ErrP classification model was used in the study to improve the efficiency of experiments that aimed at obtaining human intuitive preferences. By observing

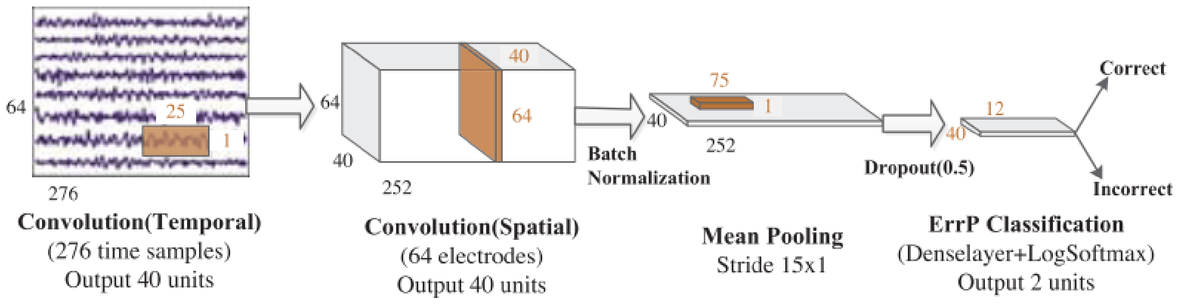


Figure 3.15: CNN-L architecture. Taken from [82].

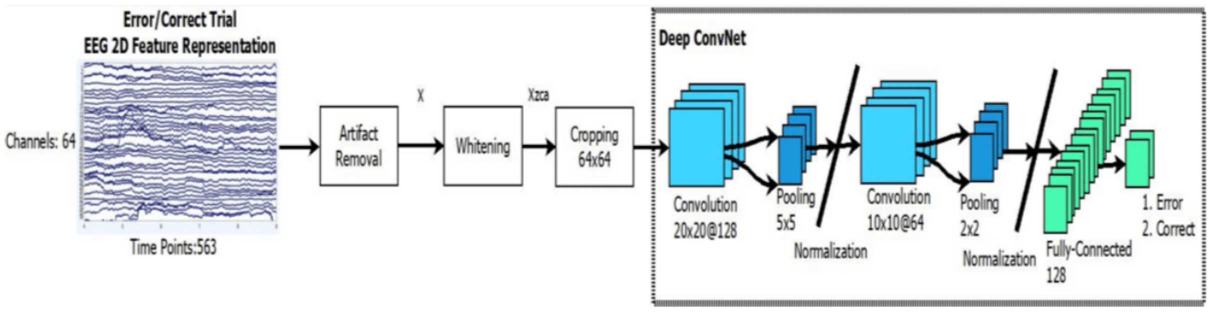


Figure 3.16: ConvNet architecture. Taken from [63].

the error perception caused by a random selection of the computer (contrary to the user preference) the authors were able to train the model to reach a 67% accuracy level.

In the same year, Torres *et al.* [63] proposed a model that used a similar idea to Shan *et al.*: convolve both spatial and temporal dimension at the same layer. Their model, called *ConvNet*, performs a mixed convolution with a max-pooling layer followed by another mixed convolution with max-pooling and a FC layer at the end with two nodes as output. Additionally, as part of the pre-processing before feeding the data into the *ConvNet* model, the EEG passes through three stages as seen in Figure 3.16: artifact removal, ZCA whitening, and cropping. During the last stage, the signal, which has a size of 64×563, is cropped to a size of 64×64 at a random point, decreasing the temporal dimension. The authors argue that this process decreases the likelihood of trapping the model at a local minimum during training.

In 2019, Bellary and Conrad [19] described a CNN model called *ConvArch* that only takes two electrodes as input after visual inspection of the topographical maps of the averaged ErrP. The largest variation was observed at the region of the anterior cingulate cortex, hence the choice of the *FCz* and *Cz* electrodes out of the 64 available channels. Again, the first convolution layer performs a mixed convolution operation with both the temporal and spatial dimensions. Then, a set of three modules follow, each being composed of a temporal convolution layer (with ReLU activation functions) and a max-pooling layer that halves the size of the matrix data. At the end, a single fully-connected layer is followed by a *softmax* activation function that outputs two nodes. The authors experimented with two versions of this architecture: one as just described (*ConvArch1*) and another with added batch normalization and dropout layers (*ConvArch2*). Unfortunately, the exact arrangement of these added layers inside the architecture of *ConvArch2* and the dropout rates are not mentioned in the paper and must be defined through an educated guess.

Just like with the Machine Learning models presented in the ”Brain-computer interface background” chapter, the comparison of results from all the models presented in this section is not obvious due to the different datasets, pre-processing, and training methods used. However, in general, these models present performances that range from 65% to 80%. A summary of the dataset pre-processing, model architecture, and training parameters from these studies is presented in Appendix A.

4

Methodology

Contents

4.1 Overview

The main goal of this thesis is to improve the accuracy of error-related potential classification using CNNs. To achieve this, the author applies a two-fold strategy: (a) build and test models from other studies in the literature that use CNNs to classify ErrP signals and (b) develop new models based on the knowledge gained from previous literature, both on ErrP and *P300* signals (section 3.5), to further improve the accuracy.

Studies that use CNNs to classify ErrPs achieved, with various levels of performance, the goal of the present thesis and thus are important to consider and reproduce. Studies that use CNNs to classify *P300* are also considered as relevant to test the transfer of their capabilities on classifying *P300* into classifying ErrPs due to the similar nature of these two ERPs as explained previously. Thus, their benefits and drawbacks will be considered when developing new models.

This chapter is organized as follows: in section 4.2, the problem formulation is stated mathematically. Then, on sections 4.3 and 4.4, the used dataset and its pre-processing are detailed. Section 4.5 describes the implementation of three ErrP models from recent studies and the architectural considerations took to develop a new model that can surpass the state-of-the-art. This section also describes the train settings and performance metrics used to evaluate the models.

Besides developing models for ErrPs classification, a secondary goal of this thesis is to promote the open-source share of code by researchers when publishing papers, thus facilitating the work of other researchers when developing new models or experimenting on published ones. Section 4.6 addresses this goal by exploring free tools used to open-source all the code and models used in this work. The author hopes that the suggested tools can be used by other researchers when publishing Deep Learning research and viewed as a way to make scientific results more easily accessible to all.

4.2 Problem formulation

This section presents a mathematically abstract formulation of the problem presented in this thesis. It concerns the main goal which is to classify, with the best accuracy possible, the EEG signal generated after a feedback is presented to a patient. This classification depends on the presence of a specific type of event-related potential called error-related potential (ErrP). If the ErrP waveform is present then the model should classify the given EEG segment as ”containing an error-related potential” or otherwise ”not containing an error-related potential”. The mathematical abstraction can be broadly divided into three parts: the input, the function, and the output.

The input refers to the EEG signal segment which is composed of one or more channels (let C be the number of channels), lasts for a certain period of time after the feedback presentation, t (in seconds), and is sampled at a rate f (in Hertz). It is represented as a matrix of size $C \times N$, where $N = t \times f$ is the number of time samples in the signal.

The output is the classification label attributed to the input. In this problem the classification is binary, hence, the output can be one of two labels: "the EEG has an ErrP present" or "the EEG has not an ErrP present". The label can be renamed to whatever is more appropriate such as "positive" and "negative" but in mathematical language, 0 and 1 are often used. Therefore, the classifier will label 0 to an input that has not an error-related potential present and 1 otherwise.

The function is the classifier itself that takes the EEG input and returns the classification output. In this work, this function is a CNN model. Since this model is explained in the "Deep Learning background" chapter, it will not be further discussed in this section.

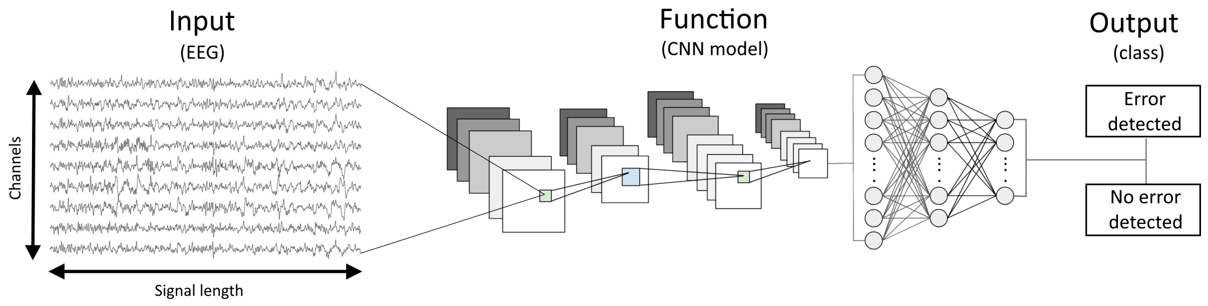


Figure 4.1: Problem definition summary

4.3 Dataset

The dataset used in this work was created by Chavarriaga and Millán in 2010 [64]. Its data and description files are publicly available at the BNCI Horizon 2020 project website under the name *Monitoring error-related potentials (013-2015)* [85].

This dataset was created to study error-related potentials (ErrPs). The signals are generated while the user is monitoring an external device upon which no control is possible. Each subject has to monitor a screen where a green square cursor and a target location are displayed (see Figure 4.2). The cursor can travel along a horizontal line made up of 20 evenly spread positions. On each trial, a target appears either at the left (as a blue square) or at the right (as a red square) of the cursor. Then, the cursor moves in the direction of the target taking approximately 2 seconds to reach it. Subjects are asked to monitor this motion while staring at the center of the screen. After reaching the target, the cursor stops, and a new target comes up no further than 3 positions away from the last target (if the new position falls outside the working window it is reset to the center). The subjects have no control over the cursor, but are informed that the goal of the cursor is to reach the target. To elicit ErrPs in the subject, the cursor has a 20% probability of moving away from the target, contrary to its determined goal.

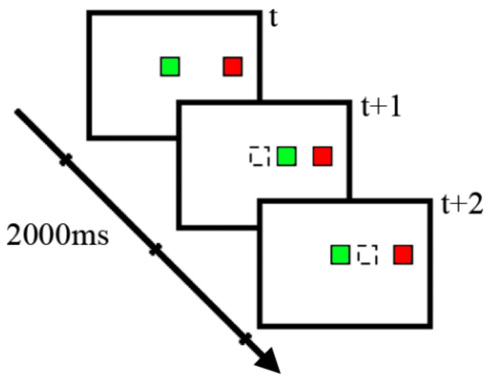


Figure 4.2: Experiment used for generating the dataset. The green square is the moving cursor, the red square is the target and the dotted square is the position of the cursor in the previous time step. Correct and erroneous behaviour at times $t + 1$ and $t + 2$, respectively. Taken from [64].

The dataset includes six subjects (mean age of 27.83 ± 2.23 years) who performed two sessions each, separated by several days (indicated in Table 4.1). Each session contains 10 blocks (3 minutes each) with approximately 50 trials per block. The raw dataset (prior to any balancing stage) contains 1030 target trials (ErrP) and 4085 non-target trials (without ErrP) as refered on Table 4.2.

Table 4.1: Number of days between the two sessions of the dataset.

Subject	1	2	3	4	5	6
Days between sessions	51	50	54	211	628	643

Table 4.2: Number of trials in the dataset per class and subject.

Trial type	**Per subject**						**Total**
	1	2	3	4	5	6	
Target	181	184	142	165	186	172	1030
Non-target	628	654	706	676	696	725	4085

EEG acquisition was performed using a Biosemi ActiveTwo system (Biosemi, Amsterdam, Netherlands) at 512Hz with 64 electrodes according to the extended 10/20 international system (extended version of the system in Figure 2.3).

Both the target position and the cursor's direction of motion are stored using hardware triggers encoded in a binary fashion: bits 0 and 1 for the target located at left or right, respectively, and bits 2 and 3 for the cursor moving towards the left or right, respectively. Hence, converting these into decimal representation: events 5 (0101) and 10 (1010) correspond to correct behavior which should not elicit an error-related potential in the subject, and events 6 (0110) and 9 (1001) correspond to incorrect behavior which should elicit error-related potentials.

4.4 Data pre-processing

After downloading the dataset and converting the trigger coding into proper binary labels (correct/incorrect behavior of the cursor) some pre-processing is needed before feeding the data into the models. Figure 4.3 depicts the pre-processing pipeline from the raw data to its feeding to the CNN model. Four stages are used and discussed below: filtering, channel selection, epoching, and balancing.

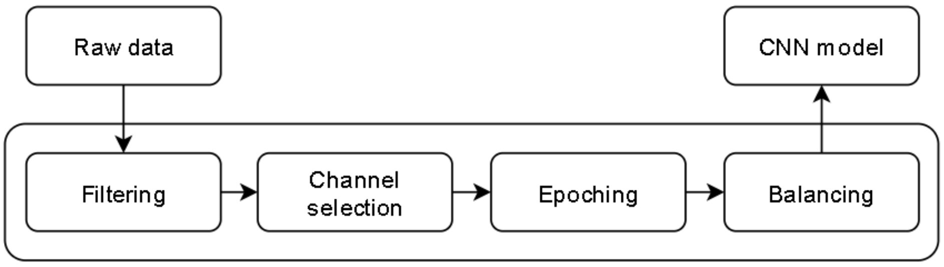


Figure 4.3: Pre-processing pipeline.

Filters

One of the most used tools for signal pre-processing is a filtering process. The goal is to remove unwanted noise such as slow signal drifts (low-frequency noise) or the periodic oscillations caused by the electric grid (high-frequency noise). Most of the studies collected from the literature use a band-pass filter with varying ranges. Figure 4.4 depicts the ranges used by these studies. The largest common range shared between all studies is the range from 1 to $10Hz$.

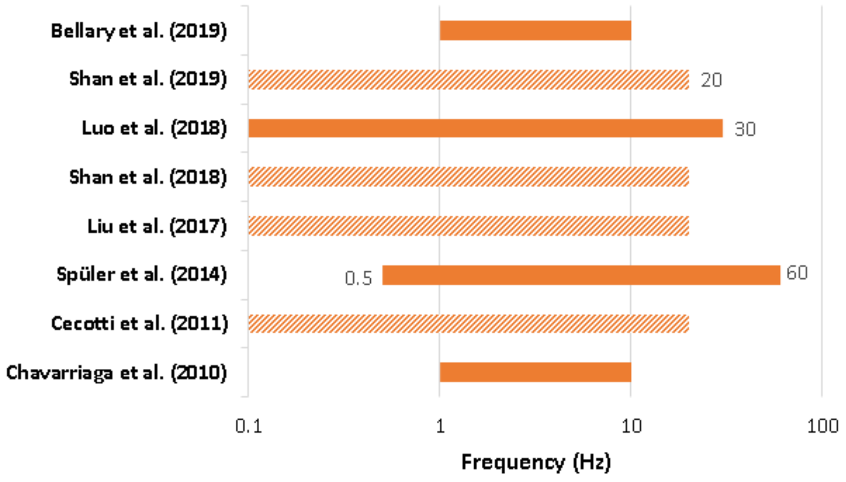


Figure 4.4: Band-pass filter ranges used to pre-process the data in various studies. Solid bars refer to studies that classified ErrP while stripped bars refer to studies that classified *P300* signals. Horizontal axis is in logarithmic scale.

The limits of a band-pass filter depend on the specific application to which it is meant and there is no precise way to select them. In this work, the range used to pre-process the data for all models is set to be from 1 to $10Hz$ as ErrP occur at low frequencies [86]. It has been shown that ErrPs elicit an increased response in *theta* activity [52], defined as the spectrum of brain activity between 3.5 and $7.5Hz$, and hence, the suggested filter range is appropriate.

Channels selection

When reproducing the models from previous papers, the channel selection method of the original paper is preserved as it affects the data fed to the model by allowing access to different areas of the brain and greatly changes the amount of data used. As an example, the *ConvArch* model from Bellary and Conrad uses 2 channels only, while *ConvNet* from Torres *et al.* uses all 64 channels.

For testing new models, all the available channels are always used, which, given the presented dataset, means using 64 channels. The reason for this is two-fold. Firstly, because preliminary tests using all 64 channels showed better performance than using the channel subset presented by Bellary and Conrad (*FCz* and *Cz*). No other subset was tested since this was the only channel subset encountered in the literature and because these are the channels in the anterior cingulate cortex area which is believed to be the brain region responsible for the generation of ErrPs, and thus a preferred subset over others. The second reason has to do with the trend of evolution of brain signal acquisition methods. As presented in section 2.2, among the multiple available options for brain activity acquisition, EEG is one of the preferred methods due to its low invasiveness and price. However, it has lower spatial resolution and poorer signal quality than other methods such as electrocorticography or intracortical neuron recording. As the invasiveness and complications of implanting these methods decrease due to medical and robotic advances, a shift will be observed regarding the preference of acquisition methods. The Link device by Neuralink is an example of these advances. As methods with far greater spatial resolution and thousands of electrodes become available, the question is not which channels to select, but rather how to merge the information from all the channels. This is because instead of having small sets of electrodes covering broad brain areas whose functions are known, devices such as the Link have hundreds or thousands of electrodes in the same local neural population. Hence, it is imperative that investigation focuses on the best way to make deep learning models select or combine the best channels possible rather than hand-picking specific electrodes which might lead to sub-optimal performance.

Although the database used presents low spatial resolution, the author intends to push the investigation in the direction just mentioned above.

Epoching

Epoching is the process of cropping a set of trials to a common temporal window. Figure 4.5 shows the epoching window range performed in different studies. When reproducing some of the models from previous studies the original epoch ranges are used. As it can be seen from Figure 4.5, most epoch windows have a temporal duration of about one second. The majority starts the epoch at the beginning of the feedback (for ErrP) or stimulation (for *P300*) but some authors choose to start before that event. Only a few studies use a shorter epoch size of around $600ms$ [81, 82, 84]. Regardless of the used size, none of the studies justifies the choice of the epoching used.

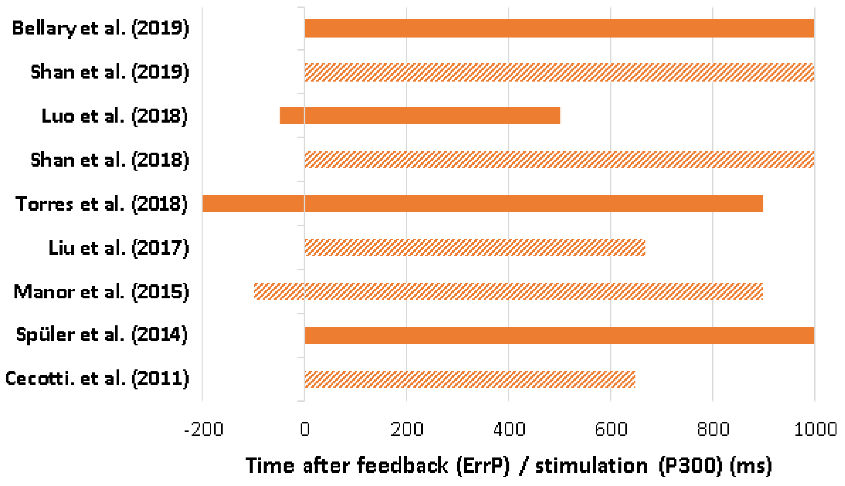


Figure 4.5: Epoch time windows used in various studies. Solid bars refer to studies that classified ErrP while stripped bars refer to studies that classified *P300* signals.

This raises a question of whether the epoching window size can be reduced or if that reduction would jeopardize the performance of the model. In some aspects, however, it seems that reducing the epoching size brings some benefits to both the model and the BCI application. Firstly, for a fixed sampling rate, a shorter epoch window means fewer time samples which translates to a smaller input size, thus reducing the complexity of the model. Similarly, if a higher sampling rate is desired for greater temporal detail of the signal, the same model complexity can be achieved by decreasing the epoch window size. Secondly, when considering online setups (both in real or experimental contexts) that depend on a feedback loop, an important factor to have in mind is the lag time between the feedback presentation and the corrective action. Using ErrPs to correct errors caused by the BCI system is such an example. The correction is only applied after a feedback loop which takes time to process: first, the feedback stimulus is presented and it is perceived by the user, then the EEG signal must be captured (the whole desired epoch window) and processed to obtain the final classification, and finally, corrective action must be applied in the case of a detected anomaly. By decreasing the epoch window size, the time during which the EEG needs to be acquired before being processed is also decreased, thus reducing the total lag time in a real-time application.

In the present work, the author hypothesizes that an epoch window of $600ms$ is a time window with sufficient temporal information about the ErrP to allow for current state-of-the-art accuracy, while also almost halving the acquisition time lag when comparing to most other models. This value is chosen so that the peak of activity of the ErrP, which is around $300ms$ after the feedback, is placed at the center of the proposed range, allowing the tail and damping of the signal to also be included in the window.

To confirm this hypothesis, an experiment is performed where a group of different ranges is used to epoch the input signal. The first range starts at feedback presentation and lasts $1000ms$. This is the most common range and is thus used as the control range. Then, several increasingly smaller ranges follow: from the range $[0, 600]ms$ until the range $[0, 200]ms$ in steps of $100ms$. After epoched, the data is fed to a CNN model, and the accuracies obtained are compared. If no significant differences in performance are observed between the control range and the $[0, 600]ms$ range, then the hypothesis is valid and the shorter window both preserves signal information and reduces lag time. The remaining ranges are tested to verify how much the epoch window can be reduced before any significant performance reduction is noticed. The results are shown in the ”Experimental results” chapter.

Balancing

Depending on the problem that is being considered, certain databases might present classes more biased than what would be desirable. For example, to build a model that could predict the gender of a person given the length of his/her hair, then a database with examples would be necessary. Since the world population has an equitable share of both men and women, a random sample would produce a balanced dataset such that both classes (man and woman) have approximately the same size. On the contrary, if the model is built to predict if a person is a left-handed writer or a right-handed writer based on some other characteristic, then a random sample from the population would create a database with much more right-hand writers as these occur more commonly. In this case, the database would be unbalanced. The problem with this is that the model might tend to learn to classify any given person as a right-hand writer just because that is the most probable outcome, rather than relying solely on the given characteristics.

Hence, balancing the classes becomes necessary. One way to achieve this is by decreasing the number of examples in the largest class by removing these from the database. This approach, however, is usually undesirable when training Machine Learning models as they require large amounts of data for training. Another approach is to increase the size of the smallest class which can be achieved in several ways. The simplest is to replicate multiple data points until the class reaches the desired size. Another is to apply data augmentation, a technique that slightly modifies a data point from the class, for example by adding some noise or cropping the signal, or by creating newly synthetic examples. Data augmentation is a popular technique, for example in the field of image recognition, where the input signal does not lose its meaning by rotating, zooming, or cropping the image or by changing its colors or adding white noise. The limits where the transformation deprives the input of its meaning, such as cropping the image without including the target or adding too much noise, are known as images are intuitive data for humans. However, with neurophysiological data, the signals are not intuitive and the limits to which a signal can be transformed without losing its essential information are unknown. This might be one of the reasons why no paper was found to apply this technique in EEG data.

The dataset presented in section 4.3 is clearly unbalanced, with the class where no error occurs being more common (80%) than the class where the cursor error is forced (20%). To balance the dataset, a random replication of the smallest class without any data augmentation is used. All the studies that mention the balancing step make use of this method as well [19, 83, 84, 87].

4.5 CNN models

In the present section, the implementation of the models and the training process is detailed. Sub-section 4.5.1 describes the way that previous CNN models for ErrP classification are replicated and how some missing information in the original papers is handled. Then, on sub-section 4.5.2 the author proposes a set of improvements based not only on ErrP models but also on previous *P300* models. Details for the training process are presented in sub-section 4.5.3 and the metrics used to assess the models' performance are described in sub-section 4.5.4.

4.5.1 Replication of literature models

In this work, the author replicates three recent CNN models found in the literature that attempt to classify ErrP signals: *ConvArch*, *ConvNet* and *CNN-L* from Bellary and Conrad (2019) [19], Torres *et al.* (2018) [63] and Luo *et al.* (2018) [82], respectively. The first two works use the same dataset as the one used in the present work, while Luo *et al.* use an original database. In the present work, all three models are trained and tested with the *Monitoring error-related potentials* dataset (section 4.3).

All models are replicated as close to the description given in the original papers as possible. It is important that the original architectures are preserved even if parts of the pre-processing need to be modified for a fair comparison of all the models. Although generally most of the relevant information is present in all the three papers, some details are missing. Because each author describes the methodology in a different way, including the pre-processing, the model architecture, or the training parameters, they do not always refer all the important information necessary for a full replication by independent research groups. A solution that addresses this issue is suggested in section 4.6.

An architectural representation is also provided in Figures 4.6, 4.7 and 4.8 for models *ConvArch*, *ConvNet* and *CNN-L*, respectively. Although the original architectural representations of *CNN-L* and *ConvNet* are provided in their original papers (see Figures 3.15 and 3.16), these new representations provide an homogeneous styling which facilitate the comparison and analysis. Each container represents a data matrix, with its dimensions written on top. The number in square brackets refers to the number of maps after a convolution output. The dimensions inside the smaller container (which is depicted as a box that slides inside the larger container) refer to the kernel size. Between two data matrices, a set of operations is performed, described in order (top to bottom) under the dotted link. A series of abbreviations are used due to space constraints: ”Conv.” is a convolution layer whose kernel characteristics are referred inside the preceding data container; ”BN” is a batch normalization layer; ”MP” and ”AP” are max/average pooling layers (the first number refers to the kernel size and the second, if mentioned, to the stride size); ”D” is a dropout layer (the number refers to the dropout rate); ”FC” is a FC layer; ”ReLU” and ”ELU” refer to activation functions.

ConvArch

As mentioned in the ”State-of-the-art” section, Bellary and Conrad proposed two different architectures. Their first model, *ConvArch1*, offered a typical CNN layout: a series of convolutions and max-pooling layers with a fully connected layer at the end. The second model, *ConvArch2*, builds on top of the first by adding batch normalization and dropout layers. In this work, only the second model is replicated as it outperformed the first [19]. It is, hereon, referred to as simply *ConvArch*.

However, two issues arise with the implementation of this model: the number and position of the BN and dropout layers are not known and the dropout rate is also not indicated. Therefore, these parameters are estimated based on literature recommendations: batch normalization is applied after the convolution linearity but before the non-linearity and the dropout layer is applied just before the FC layer with a 20% dropout rate. Justification for this can be found in the next section (Proposed models).

For the training, a batch size of 512 and an SGD optimizer is used with a learning rate of 10^{-3}, a

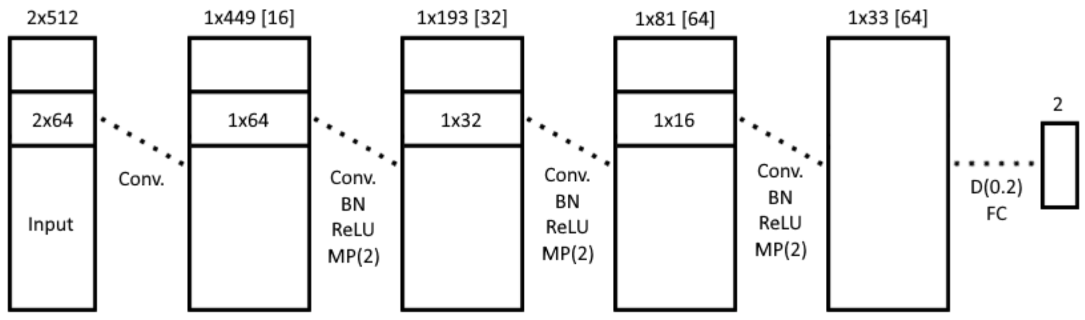


Figure 4.6: *ConvArch* architecture.

weight decay of 10^{-5} and a momentum of 0.9. The epoching interval is the same as in the original paper which corresponds to one second after the feedback presentation.

ConvNet

Contrarily to the *ConvArch* model where only two EEG channels are used, *ConvNet* makes use of all the 64 available channels. Another change to the input is that Torres *et al.* decided to crop the signal along the temporal dimension. The epoch window in the pre-processing starts $200ms$ before the feedback and extends until $900ms$ after it and just before being fed to the model, each data batch is cropped to a length of 64 temporal samples at a random point in time. Hence, the input size shown in Figure 4.7 is 64×64.

As later seen in the Experimental results chapter, this model approaches random performance, very likely due to the extreme cropping performed to the input before feeding the data to the model. Therefore, another version of *ConvNet* is tested without the cropping process to analyze the impact of the architecture itself.

An SGD optimizer is used for training with a learning rate of 10^{-4}, weight decay of 2×10^{-3}, no momentum, and a batch size of 120.

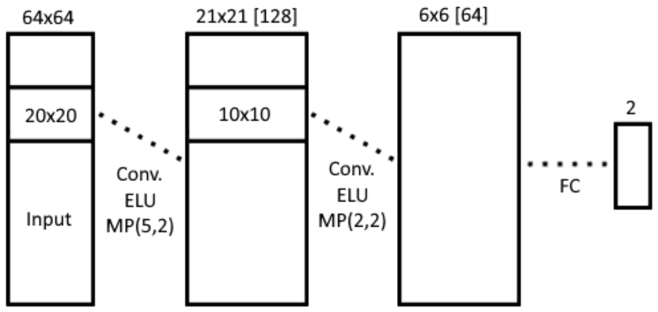


Figure 4.7: *ConvNet* architecture.

CNN-L

The *CNN-L* model is similar, in architecture, to the *ConvNet* model. Both use two convolution layers with ELU activation functions, pooling layers, and a single FC layer. The main differences are the use of a longer input (epoched from $-50ms$ until $550ms$ but without further cropping), convolutions that do not mix the temporal and spatial dimensions, average pooling layer instead of the common maximum function (max-pooling), and the use of a dropout layer.

For the training process, Luo *et al.* used the Adam optimizer but did not explicit any further parameters. Hence, the advised default parameters are used: learning rate of 10^{-3} and beta values of 0.9 and 0.999.

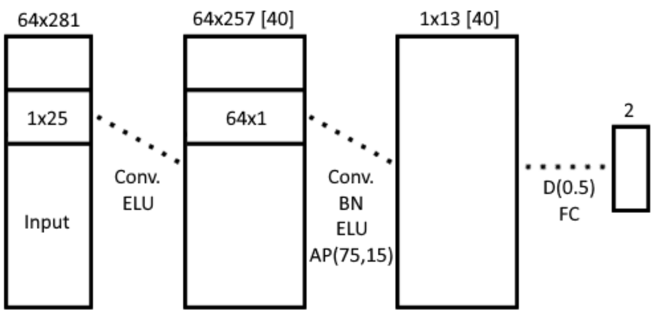


Figure 4.8: *CNN-L* architecture.

4.5.2 Proposed models

Because *P300* models have been studied for much longer, they can provide very useful information on how to implement better CNN classification models for ErrP. Table 4.3 summarizes many aspects of architectural design considered for further improvement of the state-of-the-art models which are all discussed below in this section.

At the end of the section, a generic model is presented in Figure 4.9 and the specification of 9 proposed models is defined in Table 4.5. They present slight changes between each other such as the presence of input normalization, the dropout rate or the number of fully connected layers.

Table 4.3: Summary of some architectural aspects from previous literature to be considered when designing new models. A dash (-) means that the aspect considered is not used or is not applicable to the model. A question mark (?) means that the considered aspect is used but its value is unknown.

	ErrP			**P300**			
	ConvArch (2019)	**ConvNet (2018)**	**CNN-L (2018)**	**OCLNN (2018)**	**BN3 (2017)**	**CNN-R (2015)**	**CCNN (2011)**
Input normalization	-	Pre-processing	-	Pre-processing	On model (BN layer)	Pre-processing	Pre-processing
Conv. order: Temporal (TC) vs Spatial (SC)	Mix	Mix	TC >SC	Mix	SC >TC	SC >TC	SC >TC
Activation function(s)	ReLU	ELU	ELU	ReLU	ReLU	ReLU	Sigmoid
Dropout rate	?	-	0.5	0.25	0.2	?	-
Number of fully connected layers	1	2	1	1	3	3	2
Number of output nodes	2	2	2	2	1	2	2

Input normalization

Starting with the input normalization issue, it can be seen on Table 4.3 that from the three models focusing on ErrP, only *ConvNet* applies some kind of normalization to the input by applying ZCA whitening in the pre-processing stage. On the contrary, all *P300* models use some form of input normalization. *CCNN*, *CNN-R* and *OCLNN* apply a mean and standard deviation correction as part of the pre-processing normalization, forcing the mean to be null and the standard deviation to be unitary. By changing the input distribution to a single and expected distribution, a model that uses normalization is able to learn much quicker because the model does not need to account for all the different distributions.

BN3 does not normalize during pre-processing but uses a batch normalization layer as the very first layer of the model. When using a BN layer, the model does not naively normalize the mean to 0 and the standard deviation to 1 but instead adds two learnable parameters (β and γ in Equation 3.7) which are the new mean and standard deviation. Following this approach, and to avoid the normalization during the pre-processing, a batch normalization layer is added as the first layer to all the new proposed model.

Convolution order

One other important aspect is the order in which the temporal and spatial convolution layers are placed inside the architecture. Shan et al. state that performing the spatial convolution before the temporal convolution prevents the CNN models from learning the temporal features well [25]. Because of that, they present a mixed convolution, where the kernel convolves both in the temporal and spatial dimensions simultaneously. All the ErrP models seem to already implement this strategy, either by applying a mixed convolution or by adding the spatial after the temporal as done in the *CNN-L* model. In this work, the temporal convolution is applied first, followed by the spatial convolution. This is done, instead of the mixed version, to decouple these two independent operations.

Batch normalization layer

The optimal placement of a batch normalization layer in a convolution layer is a topic of discussion among the Deep Learning community. Some argue that it should occur after both the linear and the non-linear functions while others say that sandwiching the BN between the two provides the best results. These divergences of opinion might be originated on different experimental performances that depend not only on the architecture but also on the problem, the dataset used or the training parameters.

In the original paper that introduces the concept of batch normalization, Ioffe and Szegedy apply it just before the non-linearity [76]. They compute the output of a complete convolution layer as $z = g(BN(Wu + b))$, where u is the input, W is the linear operation, b is the bias, g is the non-linear function (activation function) and BN is the batch normalization layer. The authors also argue that BN could also be applied to the input, u, but because this is probably the output of another non-linearity, its distribution is likely to change during training. On the contrary, $Wu + b$ is more likely to present a Gaussian-like distribution, making it the best input to the batch normalization layer [76]. Similarly, for this work, all the proposed models apply BN before the non-linearity.

Activation function

Concerning the activation function used for the convolution layers, ReLU is the most used function for *P300* models. Cecotti and Gräser probably did not use ReLUs in their model, *CCNN*, as this activation function was only developed in 2010 [67] and was not yet a well-known function. During the last years, it has dominated the field of Machine Learning, being the most used activation function in state-of-the-art solutions [67]. However, it introduces the dead neuron problem that affects the training process as explained in the "Artificial Neural Networks" section. Both the LeakyReLU and the ELU functions address this issue, although the latter provides better generalization and faster training. Therefore, and following the *ConvNet* and *CNN-L* models, the ELU function is used for this work.

Dropout rate

As discussed in the "Dropout layers" section, the dropout layer is an important element in studies with relatively small datasets, preventing the model from overfitting during training [88]. The only parameter concerning these layers is the dropout rate which controls the percentage of random nodes that will be zeroed at any given training step. If too many nodes are ignored, then the model might learn slowly or not learn at all. On the contrary, if the rate is too low, then overfitting might happen and the purpose of the layer is lost. In the paper where dropout layers were proposed, the authors used dropout rates of 50% [74]. A more recent study suggests that using lower dropout rates such as 20% is better [88]. In fact, the *P300* models that implemented this technique, chose a relatively low dropout rate (20% and 25%). To verify the effectiveness of this layer, three different dropout rates are tested: 0%, 20%, and 50%.

Furthermore, all the three *P300* models that use dropout layers (*OCNLNN*, *BN3* and *CNN-R*) apply it in the FC layers. Liu *et al.* concluded that using dropout in previous layers leads to worse classification. The reason behind this positioning is that if during training a relevant feature (calculated from the convolutions) is zeroed by the dropout layer, then the FC layer must adapt and use other features to improve the classification, avoiding, thus, complex and unnecessary co-adaption between FC neurons.

Fully connected layers

The number of FC layers used after the convolution layers can also be optimized. As the number of FC layers increases, the model has the ability to develop more complex relations between the features to infer the correct classification. However, as referred in section 3.3, a network can approximate any continuous function with a single FC layer given that it has sufficient nodes. Most of the ErrP models use only one FC layer, while the *P300* models tend to use up to three layers. To verify the effectiveness of extra layers in the architecture, three different models are tested: with 1, 2, and 3 FC layers.

Output nodes

In Machine Learning, the number of output nodes is usually set equal to the number of classes in the problem. When dealing with a binary problem, such as classifying the presence of a specific event-related potential in an EEG signal, the tendency is to use 2 classes. This is in fact what is observed from the collected studies, where almost all studies used 2 output nodes. The *BN3* model is the only to use one single output node whose value is fed to a sigmoid function which defines the probability of a certain class being the correct one. Given a threshold value (in the case of *BN3* this is 0.5), the most probable class is selected. Having a single output node for a binary case not only avoids redundancy, but it decreases by half the number of weights in the last FC layer. Therefore, in this work, only one output node is used.

Kernel size

To be able to properly compare and analyze the kernel sizes from previous models in the literature, it is necessary to convert the kernel size from the number of samples to the effective time it encompasses. The reason for this is that not all datasets used to train models have data sampled at the same frequency. As a consequence, two identical models trained with datasets sampled at different rates will not produce the same convolution operation. Table 4.4 summarizes both the sampling frequency of the datasets used to train various models and their kernel discrete sizes. With these two pieces of information, it is possible to calculate the temporal size of the kernel, present in the last column.

Considering that the minimum and maximum values used in the literature are $40ms$ and $125ms$, the

Table 4.4: Kernel sizes used in different models and correspondent temporal size

		Sampling frequency (Hz)	Kernel size (samples)	Kernel size (ms)
ErrP	ConvArch	512	64	125
	ConvNet	512	20	39
	CNN-L	500	25	50
P300	OCLNN	240	16	67
	BN3	240	20	83
	CNN-R	64	6	94
	CCNN	120	13	108

author decides to experiment with three different kernel sizes within this range: 20, 40, and 60 samples which correspond to temporal sizes of approximately $40ms$, $80ms$ and $120ms$, respectively.

Stride

The stride size of the convolution is also an interesting parameter to discuss as it allows a significant reduction in the size of the output. By controlling the overlap between two consecutive convolutions (when the kernel slides on the input) more or less redundant information is obtained. If the stride is unitary, then a long kernel only slides by one sample point obtaining the almost exact same information as before. This increases the complexity and output size unnecessarily. To decrease both these parameters and consequently the training time, the architectures proposed here use a stride size equal to the kernel size. This allows no input overlap when calculating two consecutive kernel convolutions. Because this stride choice largely reduces the data dimensionality, no other methods such as max-pooling are used for that effect. A study by Springenberg *et al.* re-assessed the state-of-the-art concerning object detection, where the common pipeline is using a sequence of convolutions with max-pooling layers followed by fully-connected layers. They found that the max-pooling layers can be replaced by a larger convolutional stride without loss of accuracy [89].

Conclusion

Finally, a summary of the architectural elements defined previously is given. Figure 4.9 depicts a generic model with the various possible changes to be tested.

A batch normalization layer is always added as a first layer to normalize the input. Then, a temporal convolution is performed with three different kernel sizes tested ($T = 20$, $T = 40$ and $T = 60$). The removal of the BN and ELU activation layers after the first convolution is also tested to see if it produces

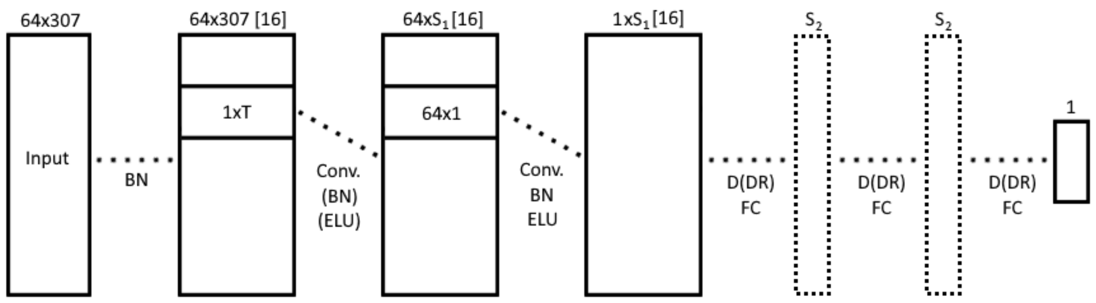


Figure 4.9: Generic architecture for the proposed models. Variables refer to the aspects of the model to be tested. T is the kernel size of the temporal convolution. S_1 and S_2 refer to dimensions of the data matrices which depend on T. Testing the use of a different number of FC layers is represented with the dotted data matrices at the end. Furthermore, both *ConvArch* and *BN3* do not use an activation function nor a BN layer in their first convolution. Hence, the inclusion of both these components is also tested to verify if any improvement takes place (this is represented in the architecture by the parenthesis).

higher accuracy. To test the necessity of multiple FC layers, models are also tested with 1, 2, and 3 of these layers. Finally, the author also experiments with various dropout rates, whose layers are always added just before the FC layers.

In Table 4.5, the various instances of the architecture in Figure 4.9 are detailed. Although many more experiments were performed, the table summarizes and organizes the models with relevance for the present work. The first model is defined with the simplest set of parameters: shortest kernel size, a single FC layer, no dropout added, and uses a complete convolution set for the first convolution (with BN layers and ELU activation functions). After testing one set of parameters, the best value is fixed and used for the remaining models. As an example consider the first three models which are defined with different kernel sizes. These models are compared to understand the kernel size that produces the best performance. After this value is determined, it is defined as the kernel size for all other models (4 to 9, in this case).

Finally, an extra model is added at the end without batch normalization layers to verify their effectiveness in both increasing the accuracy of the model and reducing the overfitting phenomenon during training.

Table 4.5: Summary of the considerations for the proposed models. After testing one set of parameters, the best value is fixed and used for the remaining models, replacing the dash (-) shown.

Model number	Kernel size (samples)	Number of FC layers	Uses BN and ELU on first conv. layer	Dropout rate	Uses batch normalization
1	20	1	Yes	0%	Yes
2	40	1	Yes	0%	Yes
3	60	1	Yes	0%	Yes
4	-	2	Yes	0%	Yes
5	-	3	Yes	0%	Yes
6	-	-	No	0%	Yes
7	-	-	-	20%	Yes
8	-	-	-	50%	Yes
9	-	-	-	-	No

4.5.3 Train details

In this section, the training parameters used to train the new proposed models are described. Table 4.6 compares these parameters with the values used in previous studies.

Table 4.6: Training parameters for ConvArch, ConvNet, CNN-L and the proposed models.

	ConvArch	**ConvNet**	**CNN-L**	**Proposed models**
Optimizer	SGD	SGD	Adam	SGD
Learning rate	10^{-3}	10^{-4}	10^{-3}	10^{-3}
Weight decay	10^{-5}	2×10^{-3}	0.0	10^{-5}
Momentum	0.9	0.0	0.9	0.9
Batch size	512	120	128	128

Optimizer

In the background ”Optimizers” section, it became obvious that many different optimizers can be used and that the choice for the best one is very empirical and depends on the datasets and type of problem. The present work does not intend to investigate the best optimizer to the presented problem. Hence, following the majority of the literature the author chooses to use the SGD optimizer (with momentum) to train the new proposed models. This is a well-established optimizer that does not suffer from generalization problems as other optimizers such as Adam. The remaining parameters also follow the most common practices in the Machine Learning community: a learning rate of 10^{-3}, a weight decay of 10^{-5} and a momentum of 0.9.

Because of the way processing units (such as CPUs or GPUs) load chunks of data into memory, the batch size is often chosen as a power of 2 to increase performance [90]. For the present work, a batch size of 128 is chosen.

Early stopping

The number of training epochs used in this work is not pre-established as in many other studies. The models train until the overfitting effect starts to be observed, *i.e.*, when the validation loss starts to increase. This can be implemented using an ”early stop with patience” which evaluates a target metric (the validation loss in this case) and stops the training when some condition is met such as if the target metric keeps increasing for some number of epochs (the ”patience” parameter). In this way, unnecessarily long training times are avoided, reducing it to the minimum time necessary.

Dataset split

Before splitting the dataset into training, validation, and test sets, it must be noted that the dataset is itself split in two sessions separated in time (see Table 4.1). Although it could be assumed, as done by Torres *et al.* [63], that these two sessions are sampled from the same distribution, making it admissible to merge the two as indistinguishable sub-sets from the same dataset, this is not the approach followed in this work. In fact, in the present work, the two sessions are assumed to be sampled from different distributions. This change between sessions might have been caused by an alteration of the subject's mental state, such as the mood or the attention level. Since the two sessions cannot be merged and because each contains approximately half of the total dataset, an unusual data split is performed: one entire session is used as the train set, while the remaining session is split into the validation and test sets. These two last sets must share the same distribution because the purpose of the validation set is to verify the performance of a hypothetical test set during training. Hence, drawing these two sets from different distributions would not be of interest.

During the development of a BCI system, a model is trained to control the BCI and only then, a certain amount of time later, is the model tested and used for its designed application. To emulate this sequence of events, the training set is assigned to the first session of the dataset, which occurs first in time, and the validation and test sets are assigned to the second session.

Because the reliability of the reported performance is more important than knowing when the training is overfitting, the test set should have a larger share of the second session than the validation set. Hence, the split of the second session is set as 70% and 30% for the test and validation sets, respectively.

To further increase the reliability of the performance metrics, the results are averaged on 5 independently trained models. On each new train, both the test and validation sets are randomly sampled from the second session.

4.5.4 Performance metrics

To evaluate the trained models on the test dataset a set of metrics must be defined. The simplest values that can be reported are the numbers of true positives (a target trial classified as such), true negatives (a non-target trial classified as such), false positives (a non-target classified as a target), and false negatives (a target classified as a non-target). These values are often abbreviated to TP, TN, FP, and FN, respectively.

From these four raw values, a variety of metrics can be calculated. By far, the most common is the accuracy which simply states the percentage of correctly classified trials over the total number of test trials (Equation 4.1a). In the literature, the term ”accuracy” is sometimes misused when the terms ”sensitivity” or ”specificity” are meant. Sensitivity, also called recall, hit rate, or true positive rate (TPR), refers to the percentage of correctly classified target trials over the total number of target trials (Equation 4.1c). Specificity, also called selectivity or true negative rate (TNR), refers to the percentage of correctly classified non-target trials over the total number of non-target trials (Equation 4.1d).

It must be made clear that, in the context of the present work, target trials are those where the error is forced (the green cursor behaves erroneously and goes away from the target) and where an ErrP signal is expected. On the contrary, a non-target trial is one where no error happens (the cursor goes toward the target) and no ErrP signal is expected. Hence, sensitivity refers to the ability of a model to correctly classify erroneous trials, while specificity refers to the ability of a model to correctly classify non-erroneous trials.

The harmonic mean between precision and sensitivity, called the F_1 score (Equation 4.1b), is yet another metric that is also commonly reported to assess the performance of DL classifiers [81,84]. This measure, however, has been greatly criticized [91] as it does not take into account the size of each class. In other words, the F_1 score (as well as the accuracy) can show overoptimistic results for a binary classifier when tested on unbalanced datasets. Chicco and Jurman incentive the spread use of an already existing metric that is not biased by unbalanced data: the Matthews correlation coefficient (*MCC*) [91]. This metric ranges in the interval $[-1, +1]$ where $MCC = +1$ means perfect classification, $MCC = -1$ means perfect mis-classification and $MCC = 0$ means the network is a random classifier. Since all the other previously mentioned metrics are in the range $[0, 1]$ it is better to also present a normalized version of the Matthews correlation coefficient, refered to as *nMCC* (given by Equation 4.1e), where now the random classifier threshold is 0.5.

To show the inadequacy of the F_1 score and the accuracy when dealing with unbalanced data, consider a dataset made up of 90 target trials and 10 non-target trials. Consider as well that the model classifies well the target trials but is bad at classifying non-target trials. In that case, one training session might generate a model that has the following performance: $TP = 85$, $TN = 2$, $FP = 8$ and $FN = 5$. In this case, the F_1 score and accuracy are 93% and 87%, respectively. Although both metrics are high, indicating a ”good” performance of the model, this is a highly biased conclusion that depends on the initial size of the classes. Only by also checking the sensitivity (94%) and specificity (20%) is it that it becomes clear that the model performs well for target trials and poorly for non-target trials. Instead of using all these metrics, *nMCC* can be used. Its value for the given example is 58% which is much more close to the random behavior since this metric penalized the poor performance of the model on the non-target trials. For a well-trained model whose performance is $TP = 86$, $TN = 7$, $FP = 3$ and $FN = 4$, the F_1 score and accuracy are still high (96% and 93%, respectively) but now the *nMCC* is 81% indicating the much better performance of the model.

Although in this thesis the datasets are balanced before training and testing, the *nMCC* is also presented when reporting the results for an easier comparison with previous and future studies that might not use balanced datasets.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{4.1a}$$

$$F_1 score = 2\frac{PPV \times TPR}{PPV+TPR} \tag{4.1b}$$

$$Sensitivity = \frac{TP}{TP+FN} \tag{4.1c}$$

$$Specificity = \frac{TN}{TN+FP} \tag{4.1d}$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \quad ; \quad nMCC = \frac{MCC+1}{2} \tag{4.1e}$$

4.6 Computational environment

One of the goals presented in the Introduction of this thesis is to deliver, as part of the final work product, all the code used to pre-process the dataset, and build, train and test the CNN models. The reason for this is that during the literature review, the author realized that the vast majority of the scientific papers never shares any of the code that generated the published results. Instead, the authors seek to describe the implementation details and although the main parameters are often shared, not all are, preventing a full replication of the study which is the problem mentioned in section 4.5.1. By sharing the code, the authors explicitly share all the necessary details to reproduce the experiments such as the pre-processing parameters, all the model architecture details, and all the training parameters. In addition, most of the papers also fail to mention the computational resources or the programming framework used, both of which also have an impact on the results.

The code must be open-source, well documented, and up-to-date with the most recent libraries in use for the field of Machine Learning so that it can be shared and easily used by others in the field. All the code was developed in *Google Colab* using *PyTorch Lightning* as the Machine Learning framework library and *Comet ML* as the visualization tool. Each of these components is shortly introduced in this section for the unfamiliar readers. To store and open-source the code, the GitHub repository is used.

Google Colab

Google Colaboratory [92] (or *Google Colab*, for short) is a free Google Research product that allows people to write and run Python code directly on the browser. It hosts a Jupyter notebook service without any installations required and is mostly used for ML and data analysis. All notebooks can be stored in Google Drive and can be shared or downloaded at any time.

One great advantage of *Google Colab* is that it provides computing resources such as GPUs or TPUs. The GPUs available in *Colab* include the NVIDIA models K80, P4, T4, P100 and V100. It is not possible to choose the processor and it may change from session to session. Thus, when training a model, the specific processor used to train it will be stored as part of its metadata.

PyTorch (Lightning)

PyTorch is a Python library used for Machine Learning applications. Although there are other Python libraries available for ML such as TensorFlow or Keras, PyTorch has been chosen for this work due to its advantages and increasing popularity.

Figure 4.10 clearly shows the fast increase of the PyTorch framework's popularity. The percentage of papers presented in top research conferences[1] that used PyTorch compared with TensorFlow jumped from 39% in 2018 to 66% in 2019 and is still increasing in 2020. Programming on a framework used by

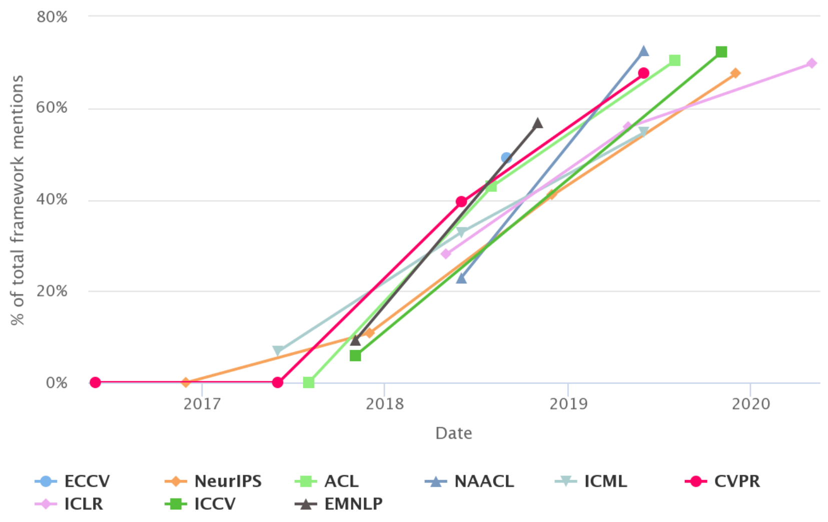


Figure 4.10: Popularity of PyTorch compared with TensorFlow. Percentage of papers published in top research conferences that used PyTorch compared with all the papers that used either PyTorch or TensorFlow. Taken from [93].

most of the scientific community is desirable because it allows for easier comparison between papers and provides a wider troubleshooting community for beginner researchers.

Table 4.7 classifies the three most common Machine Learning libraries according to various features which helps to illustrate why *PyTorch* has become so popular in the last few years. Despite it's complexity (which comes with its own advantages), *PyTorch* provides high performance for training (whereas in *Keras* the performance is slower), has great debugging capabilities (seldomly needed in *Keras* due to its simplicity and harder to perform in *TensorFlow*) and is able to handle large datasets.

Table 4.7: Comparative summary of the most common Machine Learning frameworks for Python

	PyTorch	**Keras**	**TensorFlow**
Speed	High performance	Slower performance	High performance
Architecture	Complex	Simple (readable and concise)	Not so easy to use
Debugging	Great debugging capabilities	Not commonly needed	Difficult to perform
Dataset	Handles large datasets	Used for small datasets	Handles large datasets

Some of *PyTorch*'s disadvantages such as the architectural complexity can be overcome by using a lightweight wrapper called *PyTorch Lightning* [94]. It is a Python library built on top of *PyTorch* designed to organize its code. *Lightning* divides all the necessary *PyTorch* code into three categories: research code, engineering code and non-essential research code.

[1]Conferences shown in Figure 4.10: ECCV (European Conference on Computer Vision); NeurIPS (Conference on Neural Information Processing Systems); ACL (Association for Computational Linguistics); NAACL (Conference of the North American chapter of the Association for Computational Linguistics); ICML (International Conference on Machine Learning); CVPR (Conference on Computer Vision and Pattern Recognition); ICLR (International Conference on Learning Representations); ICCV (International Conference on Computer Vision); EMNLP (Conference on Empirical Methods in Natural Language Processing).

This way of organizing code makes it extremely easy to read other people's code since every step of the process (train, validation and test steps, data load, optimizer and scheduler definition, etc) has to be written inside functions with pre-defined names. This modularity and consistency help in the uniformization of Machine Learning literature. Furthermore, all the hyper-parameters that are not explicitly defined in the model are automatically set according to the best present practices.

Comet ML

Comet [95] is a machine learning online platform used to track, compare, explain and reproduce machine learning experiments. On the website, it is stated that *Comet* was created to "push machine learning research and encourage reproducibility". It allows users to share datasets, parameters, code changes, and experimentation history. These features bring efficiency, transparency, and reproducibility to Machine Learning and, in particular, to scientific experiments. *Comet* provides deeper reporting and more features compared to other similar available platforms such as *Tensorboard*. Additionally, it allows users to view multiple experiments and manage all experiments from a single location. Figure 4.11 showcases some types of charts used to compare different experiments.

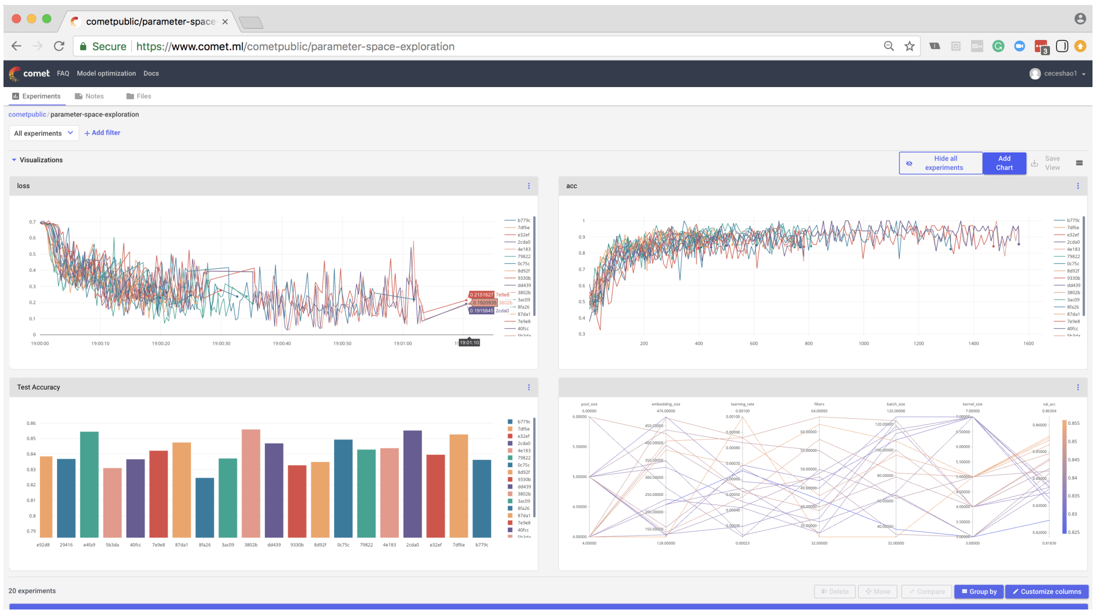

cometpublic/parameter-space
Secure https://www.comet.ml/cometpublic/parameter-space-exploration
comet FAQ Model optimization Docs
Experiments Notes Files
cometpublic / parameter-space-exploration
All experiments
+ Add filter
Visualizations
Hide all experiments
Add Chart
loss
acc
Test Accuracy
20 experiments
Group by
Customize columns

Figure 4.11: Example of a dashboard in *Comet ML* comparing results from multiple experiments.

5 Experimental results

Contents

This chapter presents the results of this thesis. Firstly, the computational results of the work are presented. On section 5.2, the obtained signals after pre-processing are presented and the problem of the epoch size is addressed. Next, on section 5.3, the performance of the reproduced literature models are presented and the best literature model is selected. Finally, section 5.4 presents the performance of the new proposed models and compares it with the best model from the literature.

5.1 Computational environment

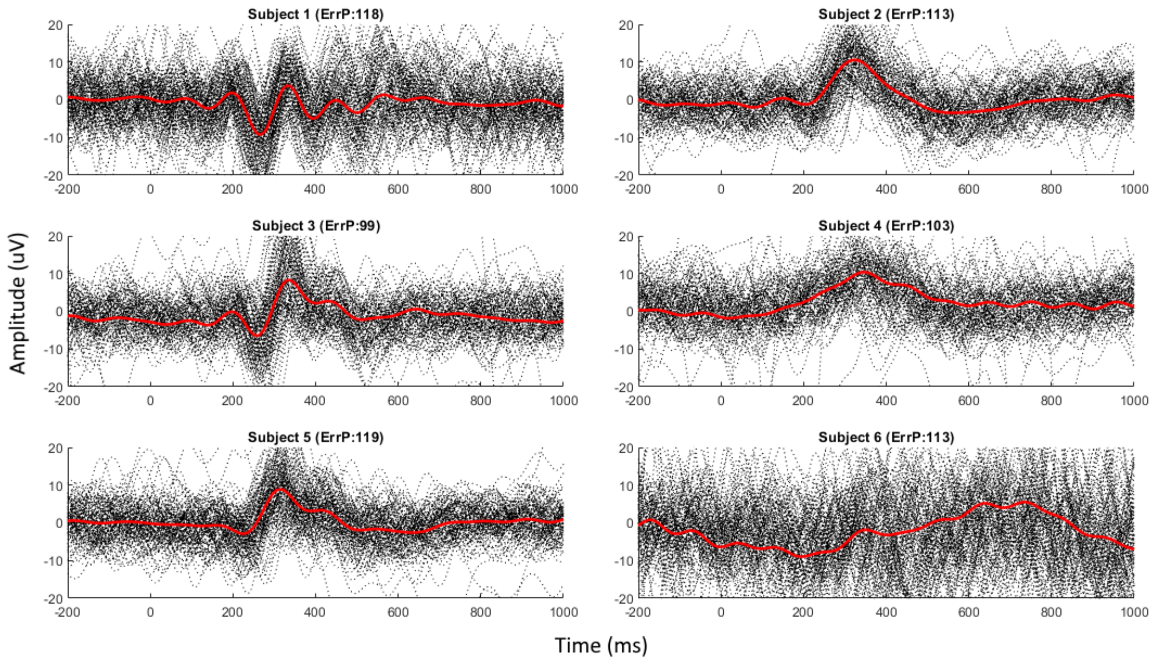
All the code developed for this work can be seen and downloaded from the author's GitHub repository [96] under an MIT license. This license allows private use, modifications, distribution, and even commercial use under the only condition that copyright and license notices are preserved. The front page gives a high-level overview of how the code is organized and explains how to replicate the results, consult them in Comet, or even how to use other datasets to train the models.

The processor used for training is a *Tesla K80* with 25GB of available RAM.

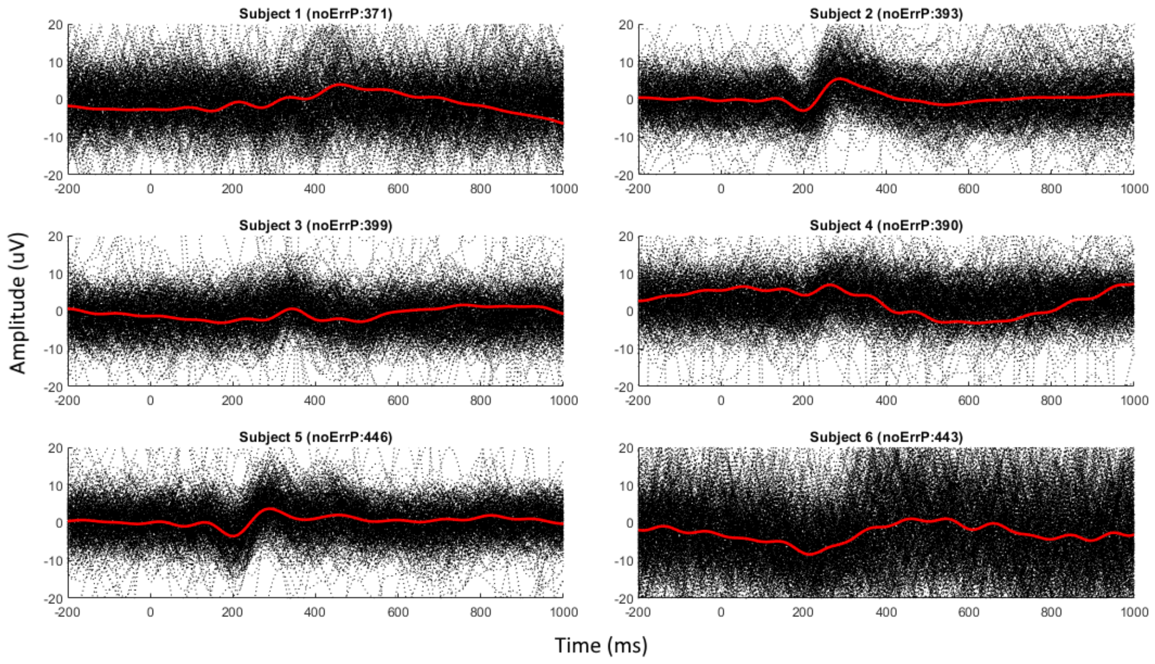
5.2 Pre-processing

Figures 5.1(a) and 5.1(b) show the result of the first pre-processing step, the band-pass filter from 1 to $10Hz$, applied to the erroneous and non-erroneous subsets, respectively. When observing the averaged ErrP signal for the 6 subjects present in the dataset, it is clear that not all have the same waveform. The signals from subjects 1 and 3 are the ones that best reflect the ErrP waveform described in the ”Event-related potentials” section with subject 1 having a larger negative peak. In subject 2, both the first positive peak and the negative peak that follows are not as noticeable as in subjects 1 and 3 but are also present. Subject 4 just presents a wider positive peak without any negative deflection between $200ms$ and $250ms$. Subject 5 presents a slight negative peak where the accentuated negative deflection is expected followed by a large positive peak. Subject 6 does not show any discernible ErrP feature expect for a positive rise between $300ms$ and $400ms$ on top of the low-frequency signal drift.

All these incongruities with the ErrP waveform described in the literature might have several causes. Firstly, the subject's attention level towards the experiment (in the case of the dataset used, concentrating on the moving cursor). The concentration level depends not only on the subject, with factors such as mood or tiredness, but also on the experimental setup such as with the presence of distracting elements in the environment. Another psychological variable of interest is motivation which can increase the amplitude of the elicited potential. [97]. Incentives such as informing the subject that his/her results are compared with those of other subjects, or monetary reward can positively increase the motivation of a subject and thus, amplify the potential. In the online description of the dataset, Chavarriaga and Millán



(a) Trials with forced error where ErrP is expected



(b) Trials where no error occurs and no ErrP is expected

Figure 5.1: Pre-processed dataset from erroneous trials. Trials from the *FCz* electrode, Session 1. Number of trials shown in parenthesis. Black dotted lines are single trials and thick red line is the average.

do not mention incentive strategies to improve motivation nor provide details about the surrounding environment other than ”Subjects seat in front of a computer screen [...]” [85]. One suggestion to increase the attention level is to make the user observe the experiment using a virtual reality headset forcing the field of view of the subject to the target of interest.

The majority of the average signals from the non-erroneous trials where no ErrP is expected (Figure 5.1(b)) stop presenting the described ErrP features, as would be expected. Subjects 2 and 5, however, still seem to present them to some extent.

When presenting their results, Bellary and Conrad show this same data for subject 1 (bottom plot from Figure 5.2). The red average trial line matches almost identically with the corresponding average obtained. As seen above, subject 1 presents the best approximation to ErrP described in the literature. Either because the subject was more concentrated or motivated, this might explain why the accuracy the authors present is higher for this subject. Since only the signals from subject 1 are plotted in their paper, this conclusion could not be drawn. Ahead, when presenting the results for the literature and proposed models, it is also verified that subjects whose ErrP signal most resembles that of the literature are more accurately predicted by the models.

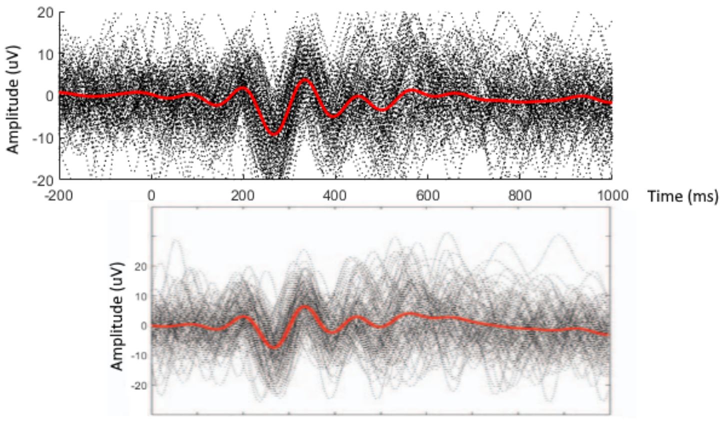


Figure 5.2: Comparison of pre-processed dataset in this work (top plot) with the results from Bellary and Conrad (bottom plot) [19]. These signals are the erroneous trials from electrode *FCz*, subject 1 and Session 1. The bottom graph is shorter to adjust with the time interval used (from 0 to $1000ms$).

Epoch size

As explained in the ”Data pre-processing” section, the hypothesis that an epoching window starting at the feedback and spanning until $600ms$ later is sufficient to capture the characteristic features of a ErrP signal is postulated. To test this, the accuracy of one model trained with differently epoched inputs is compared. For this test, the most simple version of the architecture presented in Figure 4.9 is used (model 1 from Table 4.5). The temporal convolution uses the smallest kernel size ($T = 20$) with BN and ELU activation function. Furthermore, only one FC layer is used with the minimum dropout rate (20%).

Table 5.1 summarizes the results which are averaged over 5 independent training runs. To compare each shorter epoch with the control epoch, t-tests are used with the accuracy as the comparison metric. Both the two first shorter ranges, $[0, 600]ms$ and $[0, 500]ms$, do not show accuracies statistically different from the control range, with *p-values* of 0.647, and 0.277, respectively. This means that the range can be decreased from the common $[0, 1000]ms$ range, to $[0, 600]ms$ or $[0, 500]ms$ without compromising the performance of the model. The remaining ranges, $[0, 400]ms$, $[0, 300]ms$, and $[0, 200]ms$, present a significant difference in accuracy when compared with the range $[0, 1000]ms$, with *p-values* of 0.012, 0.001, and 2.92×10^{-11}, respectively.

The $[0, 600]ms$ epoch window provides the advantage of shortening the lag time when acquiring the input signal in real-time applications by about half and is used by other studies [81, 84]. The average training time also decreases slightly as less data is processed at each train iteration. Despite the $[0, 500]ms$ range being shorter, the temporal difference of $100ms$ is not substantial and the decrease in sensitivity, although not statistically significant ($p = 0.109$), suggests that using the $[0, 600]ms$ range is a better approach. Hence, the hypothesis is experimentally verified and due to the advantages presented, the $[0, 600]ms$ epoching window is used for all the proposed models.

In Table 5.1, it is possible to observe an effect commented by Chavarriaga *et al.*, where previous studies often report higher accuracy for correct trials than erroneous trials [52]. These two accuracies correspond, in the case of the present work, to the specificity and sensitivity, respectively. Chavarriaga *et al.* suggested that this effect was due to an unbalanced dataset. However, this is not the case as the present dataset is balanced before being fed to the models and the difference is still observed across all epoch ranges.

Table 5.1: Accuracy of model 1 (Table 4.5) when training with different epoched temporal windows. Highest performance of each column indicated in bold.

Epoch time range (ms)	Overall			Subjects					
	Accuracy	Sensitivity	Specificity	1	2	3	4	5	6
[0,1000]	79.1% ±0.5%	75.2% ±1.0%	82.9% ±1.0%	85.2% ±2.3%	84.5% ±1.6%	**83.7%** ±0.6%	71.3% ±0.9%	**78.2%** ±1.4%	**71.6%** ±1.6%
[0,600]	**79.4%** ±0.7%	**76.4%** ±2.0%	82.2% ±1.7%	87.4% ±1.3%	85.6% ±0.8%	81.5% ±1.3%	**73.3%** ±1.8%	76.9% ±0.8%	71.2% ±1.7%
[0,500]	78.5% ±1.0%	73.5% ±2.4%	83.1% ±3.1%	87.9% ±1.6%	**85.8%** ±0.5%	80.6% ±2.4%	71.7% ±2.4%	77.4% ±1.1%	67.1% ±3.6%
[0,400]	77.9% ±0.6%	74.2% ±2.6%	81.3% ±1.5%	**88.7%** ±1.2%	84.0% ±0.9%	80.3% ±1.2%	70.7% ±1.6%	76.5% ±1.9%	66.2% ±1.9%
[0,300]	73.3% ±1.5%	62.7% ±3.5%	**83.4%** ±1.8%	79.3% ±3.5%	82.2% ±1.5%	81.2% ±0.8%	62.7% ±1.7%	70.8% ±1.7%	63.1% ±3.1%
[0,200]	59.6% ±0.6%	47.4% ±1.6%	71.1% ±1.1%	62.4% ±1.9%	64.2% ±1.0%	60.9% ±2.3%	58.6% ±0.9%	55.1% ±0.9%	56.3% ±0.8%

5.3 Literature models

Table 5.2: Performance of the original and replicated models from the ErrP literature. All the replicated results are averaged over 5 training runs with test and validation sets randomly sampled from the second session of the dataset. Highest replicated performance of each metric indicated in bold.

Author(s) [Year] Model	Results from	Accuracy	Sensitivity	Specificity	nMCC
Bellary *et al.* [2019] ConvArch	Original paper	86.1%	-	-	-
	Replicated	71.2% ±1.7%	56.1% ±6.7%	**85.4%** ±3.2%	71.9% ±1.2%
Torres *et al.* [2018] ConvNet	Original paper	-	77.5%	79.5%	-
	Replicated	51.7% ±0.8%	63.1% ±32.3%	40.9% ±30.5%	52.4% ±1.4%
	Replicated (no crop)	76.0% ±0.8%	67.1% ±3.8%	84.5% ±3.1%	76.3% ±0.8%
Luo *et al.* [2018] CNN-L	Original paper	67.5%	-	-	-
	Replicated	**77.6%** ±0.7%	**71.7%** ±2.8%	83.1% ±1.5%	**77.7%** ±0.6%

The first objective of this thesis is to reproduce previous CNN models used to classify ErrP. For that, three models are used: *ConvArch*, *ConvNet* and *CNN-L*, whose implementation details are defined in section 4.5.1.

Table 5.2 presents the performance results of the original and replicated models, where the accuracy, sensitivity, specificity, and nMCC metrics are detailed. Again, all the performance values are an average of 5 independently trained models. Next, follows a discussion about these results.

In their paper, Bellary and Conrad present an overall accuracy of 86.1%. After replication, the obtained accuracy for *ConvArch* is of 71.2%, which constitutes a decrease of 15%. The most likely reason for this discrepancy lies in the dataset split which defines the test set used to evaluate the performance of the model. As a balancing technique, Bellary and Conrad replicated samples from the smaller class, as also done in the present work. If this process is not done randomly, then over-representation of a particular subject may emerge in the dataset. For example, if all data samples are ordered by subject index and the balancing is done by replicating the first N samples, then subject number 1 will be over-represented, followed by subject 2 and so on, depending on the number of copied samples. Having no access to the dataset splits (train, test, and validation sets), it is not possible to analyze the distribution of subjects among each of the sets. However, the results from Bellary and Conrad suggest that subject number 1 is, in fact, over-represented in the test set. While the overall accuracy is 86.1%, the individual accuracies for subjects 1 to 6 are 87.6%, 80.1%, 79.4%, 78.5%, 70.9%, 78.8%, respectively.

With a homogeneous subject distribution, the overall accuracy would be approximately the average of the accuracies of all subjects which is 79.2%. Since subject 1 is the only subject with an accuracy higher than the reported overall accuracy, it follows that this is the subject that is over-represented in the test split. The problem with over-representing subject 1, in particular, is that it is the subject that always produces the best individual accuracy (as seen in Table 5.1, for example) and thus, it introduces a bias by over-estimating the performance of the model. When analyzing the pre-processed signal (Figure 5.1(a)) it was clear that the average erroneous signal from subject 1 was the closest to the waveform of ErrP as described in the literature. Probably due to the consistency of the single trials and their similarity to the ErrP waveform, this is the subject that consistently performs with the highest accuracy among all the subjects. In the present work, not only all the balancing and splitting operations are completely random, but all the sets used to train and test the individual models are stored for future reference and analysis.

Furthermore, it is unclear what the split percentages used by Bellary and Conrad are. In the experimental design, they state a split of $70\%, 20\%$ and 10% for the train, test, and validation sets, respectively, but later the table that presents the performances suggest that the models are trained with one session and tested with another session (the origin of the validation set is not clear). Since each session represents about half of the whole dataset, either the train or the test sets would also contain around half of the dataset each, which contradicts the initially stated split.

Torres *et al.* do not report an overall accuracy, but rather the sensitivity and specificity of their model in two different test sets (session 1 and session 2). The values indicated in Table 5.2 (77.5% and 79.5%) are the average of those two test cases for sensitivity and specificity.

The reproduced model, when considering the crop of the input, obtains an almost random accuracy of 51.7%. Both the obtained sensitivity (63.1%) and specificity (40.9%) present a standard deviation of around 30% and thus, it is clear that the model, when training, settles at a point where it more or less randomly classifies the majority of the samples as one of the two classes. An extreme example of this is a particular train instance where the model performed with a sensitivity and specificity of 0.1% and 99.9%, respectively.

The cropping of the input at the start of the architecture seems to be the problem and to test this hypothesis, another version of the same model was replicated, this time with no cropping of the input. This second model performs much better, with an accuracy of 76.0% and significant sensitivity and specificity (standard deviation of around 3%).

When Torres *et al.* suggest the cropping step, they argue that papers such as that of Schirrmeister *et al.* [98] used it to reduce the probability of identifying a false training local minimum. However, the cropped percentage used by Schirrmeister *et al.* is of 50% (from 1000 to 500 samples), while the cropped percentage used by Torres *et al.* is of 11% (from 563 to 64 samples). Such a small crop makes the model blind to the overall temporal process, making it hard to correlate and extract features from different temporal crops, thus preventing the model from learning and performing well.

The second *ConvNet* model tested not only lacks the cropping stage as it also does not implement the other pre-processing stages used in the original work such as the artifact removal or the *ZCA* whitening. Removing these pre-processing stages simplifies the pipeline while maintaining a reasonable accuracy (76.0%), and allows faster corrective responses when the user perceives and error made by the BCI.

While the studies for the two previous models (*ConvArch* and *ConvNet*) use the same dataset as the present work (*Monitoring error-related potentials*), the *CNN-L* model from Luo *et al.* uses its own original dataset which contains 12 subjects. Therefore, the performance of the original and replicated models is not expected to be necessarily the same. In fact, an increase of 10% is achieved for the accuracy with the replicated model (77.6%) when compared with the original paper (67.5%).

Furthermore, *CNN-L* presents the highest replicated accuracy of the three models. Besides other possible reasons, one factor that is singular to this model that may explain its accuracy is the sequential order of its convolutional operations. The majority of the models described in the ”State-of-the-art” section start with a spatial convolution followed by a temporal convolution or use a mixed convolution, where a *2D* kernel convolves both spatial and temporal dimensions simultaneously. *CNN-L*, on the other hand, places the temporal convolution before the spatial convolution. Other authors state that using spatial convolutions as the first layer prevents the model from adequately learning temporal features because the first spatial convolution generates abstract temporal maps which are fed to the temporal convolution instead of the initial raw temporal signals [25]. Additionally, computing each convolution separately instead of in a mixed convolution simplifies the complexity of the model by decreasing the number of trainable parameters. Considering that the temporal kernel is of size S_T and the spatial kernel of size S_S, then the number of parameters in a model with one convolution following the other is the sum of both sizes, $S_T + S_S$, while the number of parameters in a model with a mixed convolution is $S_T \times S_S$ (assuming the mixed *2D* kernel has the same temporal and spatial size as the individual *1D* kernels).

It can also be seen that, as commented before, the models often report higher accuracy for correct trials than erroneous trials. For these three models, the specificity is always higher than the sensitivity. The difference is of 29.3%, 17.4% and 11.4% for the *ConvArch*, *ConvNet* and *CNN-L* models, respectively. When discussing the proposed models in the next section, it is noted that these differences decrease by a significant amount.

Before moving on to the ”Proposed models” section, the best of the three literature models is selected for later comparison against the proposed models. The *ConvNet* model with input cropping is not considered as its accuracy is almost random. Based on the accuracy, a one-way *ANOVA* test confirms the significance of the difference between the three models ($p = 0.0012\%$). Applying a *post-hoc* t-test with Bonferroni correction ($5\%/3 = 1.67\%$) between *ConvArch* and *CNN-L* ($p = 0.0893\%$) and between *ConvNet* and *CNN-L* ($p = 1.5659\%$), reveals that the best model is, in fact, *CNN-L*. This is, therefore, the champion model from the literature that is used as a comparison with the proposed models in the next section.

Finally, it should be pointed out that all the disparities between the reported and reproduced performance metrics found on Table 5.2 are an indicator that the lack of shared code difficult the reproducibility and advancement of work done in the area of Machine Learning. This problem was addressed by the author in section 4.6.

5.4 Proposed models

In this final section, the performance of the proposed models is analyzed and discussed. Table 5.3 summarizes the results and provides the actual parameters used by each model (inside square brackets) which were still unknown in Table 4.5.

The first set of three models (model numbers 1, 2, and 3 in Table 5.3) tests the kernel size for the possible values of 20, 40, and 60 samples. A one-way *ANOVA* reveals that there is no significant difference between their accuracies ($p = 0.808$). To avoid over-complicating the model, when no statistical significance is found between a group of models, the simplest architecture is chosen. In the case of the kernel size, choosing a smaller kernel yields fewer parameters, which simplifies the model. Therefore, model 1 is chosen as champion, and models 4 to 9 implement a kernel size of 20 samples.

Before moving on to the analysis of the FC layers, a comparison between model 1 (the current champion) and *CNN-L* (the literature champion) is done to verify if there is already an improvement on the state-of-the-art. A t-test between these two models shows that there is no significant difference yet ($p = 0.297$).

To analyze the appropriate number of FC layers, three models are used: model 1, 4, and 5 which all contain the same kernel size but a different number of FC layers (1, 2, and 3, respectively). Once more, a one-way *ANOVA* test does not find any statistical difference between these three models ($p = 0.568$). Therefore, the simplest model must be chosen. In this case, similarly to the previous one, less FC layer also generate less trainable parameters and since extra layers do not increase the performance of the model, they are not necessary. Following the same reasoning as before, model 1 remains the current champion, and models 6 to 9 implement a single fully-connected layer in their architecture.

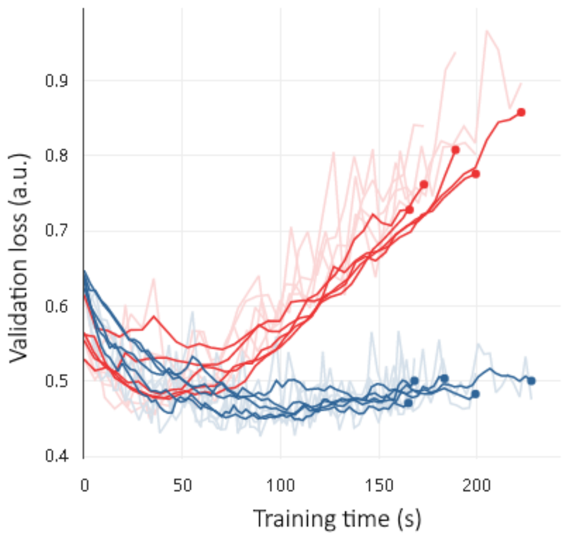
Concerning the presence of the batch normalization layer and ELU activation function in the first convolution layer, models 1 and 6 are compared, since they only differ in that regard. A t-test shows that there is no statistical difference between the two models ($p = 0.117$). The simplest model, in this case, is model 6 which performs fewer calculations not considering the statistical metrics to normalize the batch or the non-linear functions. This also simplifies the backpropagation calculations during training since fewer functions make part of the derivation chain. The non-linearity of the model is still guaranteed with the activation function of the second convolution.

Next, the dropout rates are tested with models 6, 7, and 8, corresponding to rates of 0%, 20%, and 50%, respectively. The *ANOVA* test for these three models shows that there is no significant difference between their accuracies ($p = 0.427$). Since a dropout layer does not present any trainable parameters or performs calculations, there is no particular way to choose the most simple of the three models. In this case, the choice is based on two factors. Firstly, the advantage of having this layer for controlling the overfitting phenomenon excludes model 6 as it does not make use of it. Secondly, following the considerations of previous studies [88] and the practices of other CNN models, the smaller dropout rate of 20% is chosen. Therefore, model 7 is the new current champion.

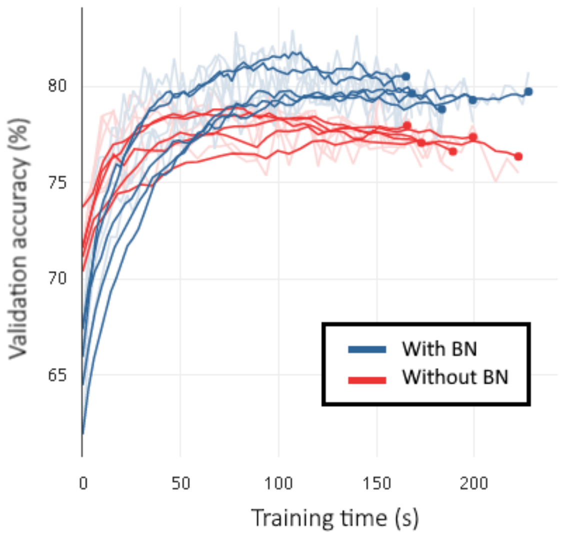
Finally, model 9 is trained to verify the effectiveness of the BN layers in reducing the overfitting phenomenon. The comparison is performed between models 7 and 9. A t-test shows that the accuracies

Table 5.3: Performance results for the proposed models. Values between square brackets are chosen for each model after being tested against the other parameter values and considered the best. Highest performance of each metric indicated in bold.

Model number	Model architecture					Performance metrics			
	Kernel size	FC layers	BN & ELU	Dropout rate	Use BN	Accuracy	Sensitivity	Specificity	nMCC
1	20	1	Yes	0%	Yes	78.4% ±1.4%	72.6% ±1.8%	83.9% ±1.8%	78.5% ±1.4%
2	40	1	Yes	0%	Yes	78.0% ±0.7%	71.7% ±1.3%	84.0% ±1.3%	78.1% ±0.7%
3	60	1	Yes	0%	Yes	78.0% ±1.0%	71.2% ±1.9%	84.5% ±1.0%	78.2% ±1.0%
4	[20]	2	Yes	0%	Yes	78.7% ±1.1%	73.0% ±3.6%	84.2% ±1.9%	78.8% ±1.0%
5	[20]	3	Yes	0%	Yes	77.7% ±1.4%	71.9% ±1.8%	83.4% ±1.5%	77.9% ±1.4%
6	[20]	[1]	No	0%	Yes	79.9% ±0.8%	75.3% ±1.5%	84.1% ±2.4%	79.9% ±0.9%
7	[20]	[1]	[No]	20%	Yes	**80.4%** ±0.7%	**75.9%** ±1.5%	**84.7%** ±1.5%	**80.5%** ±0.7%
8	[20]	[1]	[No]	50%	Yes	79.9% ±0.3%	77.4% ±0.5%	82.1% ±0.9%	79.8% ±0.4%
9	[20]	[1]	[No]	[20%]	No	76.3% ±0.7%	68.9% ±1.7%	83.3% ±2.3%	76.4% ±0.8%



(a) Validation loss



(b) Validation accuracy

Figure 5.3: Loss and accuracy plots for the validation set comparing the effect of models with and without batch normalization. The red group represents the models without batch normalization. Each line is an independent trained model.

are statistically different ($p = 0.00005$). This shows that adding batch normalization layers to the architecture, either at the output or before non-linearities, has an effective impact on the performance of the model. Furthermore, it does also have a positive influence in reducing the overfitting of a model during training. Figure 5.3 depicts the training evolution of several instances of model 7 (blue) and model 9 (red). From the validation loss graph (Figure 5.3(a)), it is clear that, after the initial reduction of the error, both models display very different behaviors: the loss of model 7 remains more or less stable after the initial decrease, while the loss of model 9 starts to increase again which means that the model is overfitting. Thus, adding the BN layers effectively prevents overfitting. Furthermore, observing the

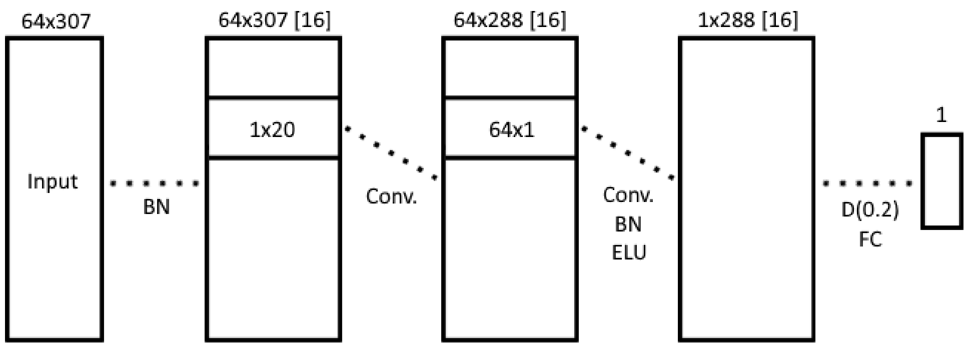


Figure 5.4: Architecture of the champion model (number 7). In this model, the input is normalized by a BN layer, the temporal kernel has a size of 20 samples, the first convolutions does not include a BN layer nor an activation function, the dropout rate is set to 20% and only one fully-connected layer is included.

validation accuracy plot (Figure 5.3(b)) it is also clear that, as the t-test showed, there is a significant and consistent improvement in accuracy when using batch normalization.

After considering all the proposed models and coming up with the best combination of features from a set of possible variations (model 7), it should be verified if this model performs better than the best model from the literature. Comparing model 7 against *CNN-L* with a t-test ($p = 0.0004$), it becomes clear that the new proposed model has significantly higher accuracy (80.4%) than the literature model (77.6%). The architecture of the champion model can be seen in Figure 5.4.

Notice that even though model 1 was not significantly better than *CNN-L* and that on each group of models the statistical analysis never concluded that there was a significant difference in performance, the accumulation of various architectural choices such as the kernel size, the number of FC layers or the dropout rate contributed to a final champion model that outperforms the state-of-the-art champion.

In the previous section, it was noted that the specificity is consistently higher than the sensitivity. For models *ConvArch*, *ConvNet* and *CNN-L*, the difference between these two metrics is of 29.3%, 17.4% and 11.4%, respectively. With the new proposed models, the specificity still remains higher than the sensitivity although the difference decreases (8.8% for model 7) evidencing a more homogeneous classification distribution.

6 Conclusion

Contents

6.1 Conclusion

The present thesis had three main goals which were all achieved: (a) investigate and replicate previous CNN models used for ErrP classification; (b) to develop new models based on advances in the field of Deep Learning that can provide improvements to the classification of ErrP and (c) provide open-source code for all the models reviewed in this work and the new proposed ones.

To achieve the first goal, the author made use of three models found in the literature which were introduced in the State-of-the-art section: *ConvArch*, *ConvNet* and *CNN-L*. After recreating the models and filling in the architectural and training details not mention in the papers, they were trained and the performance results were compared with the originally reported results. *CNN-L* was found to be the best performing model.

For the second goal, the author considered a group of CNN models from the literature used to classify both ErrP and *P300*. The reason why *P300* models are also considered was two-fold: firstly, both signals are event-related potentials and are, therefore, very similar in their nature, and secondly, many more studies have been conducted to classify *P300* signals with CNNs. These models are also introduced in the State-of-the-art section and from them, several architectural features were considered for new models. Multiple variations were tested and added incrementally. The best proposed model exceeds the literature model *CNN-L* with statistical significance and achieves an overall accuracy, a sensitivity, specificity, and nMCC of 80.4%, 75.9%, 84.7% and 80.5%, respectively, while for *CNN-L* these metrics are of 77.6%, 71.7%, 83.1% and 77.7%, respectively.

The third goal originated from the understanding that very few studies end up publicly sharing the code that generated the results. In fact, none of the studies from where the considered models are drawn shares the code for their construction and training. This lack of code was the source of many uncertainties about the specific implementation of each model, although the main aspects were described in the papers. In this work, all the replicated and proposed models were developed in *Python*, using the *PyTorch* library, one of the most popular Machine Learning library available and the most used in published papers. All these models, their training specificities, and all the code that pre-processed the dataset is publicly available at the author's GitHub repository [96].

Another important achievement was the reduction of the input's temporal size. Many models tend to use signals epoched with a length of around one second. In an online context, where the BCI system is relying on the real-time assessment of the feedback, waiting for a one-second input means introducing a one-second lag period in the processing pipeline. The author showed that it is possible to cut this period in about half without compromising the performance of the model.

One limitation of the work concerns the dataset. The reduced number of subjects prevented the research of some interesting questions due to statistical insignificance, such as if the number of days between the two sessions influences the performance when training and testing with different sessions.

6.2 Future work

The first suggestion for future work regards the limitation just mentioned about the small dataset. After prototyping and testing with available datasets, future studies should create new ones and make them publicly available. These datasets should contain a large number of subjects to introduce diversity and allow inter-subject variability to be studied but also be based on robust experimental setups that keep the subject motivated while performing the task, allow the attention levels to be high, and guarantee a good signal quality.

Another suggestion for future work is to use transfer learning to further improve the performance of models. It is possible to train a general model to recognize and identify high-level features from EEG signals and then apply this ML technique to transfer the gained knowledge to a more specific use such as building unique models for independent patients to optimize the individual accuracy. This is done by first training on a large dataset with many different subjects and, subsequently, freezing the first layers (trained on general EEG features) and fine-tuning the later layers on specific characteristics of individual subjects. Just like in the field of personalized medicine, this method could provide a more reliable BCI usage experience to patients and users.

Regarding the framework environment for future studies in the area of ML, the author advises the use of *PyTorch Lightning* due to its growing popularity among the scientific community which makes it easier to replicate models and compare results.

Bibliography

[1] S. N. Abdulkader, A. Atia, and M. S. M. Mostafa, "Brain computer interfacing: Applications and challenges," *Egyptian Informatics Journal*, vol. 16, no. 2, pp. 213–230, 2015.

[2] U. Chaudhary, N. Birbaumer, and A. Ramos-Murguialday, "Brain–computer interfaces for communication and rehabilitation," *Nature Reviews Neurology*, vol. 12, no. 9, pp. 513–525, sep 2016. [Online]. Available: http://www.nature.com/articles/nrneurol.2016.113

[3] A. Rezeika, M. Benda, P. Stawicki, F. Gembler, A. Saboor, and I. Volosyak, "Brain–computer interface spellers: A review," *Brain Sciences*, vol. 8, no. 4, 2018.

[4] C. T. Lin, S. F. Tsai, and L. W. Ko, "EEG-based learning system for online motion sickness level estimation in a dynamic vehicle environment," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 10, pp. 1689–1700, 2013.

[5] V. S. Selvam and S. Shenbagadevi, "Brain tumor detection using scalp eeg with modified Wavelet-ICA and multi layer feed forward neural network." *Conference proceedings : ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, vol. 2011, pp. 6104–6109, 2011.

[6] M. Poulos, T. Felekis, and A. Evangelou, "Is it possible to extract a fingerprint for early breast cancer via EEG analysis?" *Medical Hypotheses*, vol. 78, no. 6, pp. 711–716, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.mehy.2012.02.016

[7] S. F. Liang, F. Z. Shaw, C. P. Young, D. W. Chang, and Y. C. Liao, "A closed-loop brain computer interface for real-time seizure detection and control," *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC'10*, pp. 4950–4953, 2010.

[8] C. W. Fadzal, W. Mansor, and L. Y. Khuan, "Review of brain computer interface application in diagnosing dyslexia," *Proceedings - 2011 IEEE Control and System Graduate Research Colloquium, ICSGRC 2011*, pp. 124–128, 2011.

[9] N. Birbaumer and L. G. Cohen, “Brain-computer interfaces: Communication and restoration of movement in paralysis,” *Journal of Physiology*, vol. 579, no. 3, pp. 621–636, 2007.

[10] J. J. Daly and J. R. Wolpaw, “Brain-computer interfaces in neurological rehabilitation,” *The Lancet Neurology*, vol. 7, no. 11, pp. 1032–1043, 2008. [Online]. Available: http://dx.doi.org/10.1016/S1474-4422(08)70223-0

[11] S. W. Tung, C. Guan, K. K. Ang, K. S. Phua, C. Wang, L. Zhao, W. P. Teo, and E. Chew, “Motor imagery BCI for upper limb stroke rehabilitation: An evaluation of the EEG recordings using coherence analysis,” *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, pp. 261–264, 2013.

[12] K. A. Sorudeykin, “An Educative Brain-Computer Interface,” no. September, 2010. [Online]. Available: http://arxiv.org/abs/1003.2660

[13] G. Vecchiato, F. Babiloni, L. Astolfi, J. Toppi, P. Cherubino, J. Dai, W. Kong, and D. Wei, “Enhance of theta EEG spectral activity related to the memorization of commercial advertisings in Chinese and Italian subjects,” *Proceedings - 2011 4th International Conference on Biomedical Engineering and Informatics, BMEI 2011*, vol. 3, no. October, pp. 1491–1494, 2011.

[14] I. Nakanishi, S. Baba, K. Ozaki, and S. Li, “Using brain waves as transparent biometrics for on-demand driver authentication,” *International Journal of Biometrics*, vol. 5, no. 3-4, pp. 288–305, 2013.

[15] L. Bonnet, F. Lotte, and A. Lécuyer, “Two brains, one game: Design and evaluation of a multiuser bci video game based on motor imagery,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 5, no. 2, pp. 185–198, 2013.

[16] A. Ruys, *Alumina Ceramics: Biomedical and Clinical Applications*. Woodhead publishing, 2018.

[17] M. Londei, “BCI and Virtual Reality: gaming of the future — inBrain,” 2019. [Online]. Available: https://inbrain.tech/bci-virtual-reality/693/

[18] M. Spüler and C. Niethammer, “Error-related potentials during continuous feedback: Using EEG to detect errors of different type and severity,” *Frontiers in Human Neuroscience*, vol. 9, no. MAR, pp. 1–10, 2015.

[19] S. A. Swamy Bellary and J. M. Conrad, “Classification of Error Related Potentials using Convolutional Neural Networks,” in *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*. IEEE, jan 2019, pp. 245–249. [Online]. Available: https://ieeexplore.ieee.org/document/8776901/

[20] Z. Q. Zhao, P. Zheng, S. T. Xu, and X. Wu, “Object Detection with Deep Learning: A Review,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[21] S. Wu, K. Roberts, S. Datta, J. Du, Z. Ji, Y. Si, S. Soni, Q. Wang, Q. Wei, Y. Xiang, B. Zhao, and H. Xu, “Deep learning in clinical natural language processing: a methodical review,” *Journal of the American Medical Informatics Association : JAMIA*, vol. 27, no. 3, pp. 457–470, 2020.

[22] E. Chong, C. Han, and F. C. Park, “Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies,” *Expert Systems with Applications*, vol. 83, pp. 187–205, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2017.04.030

[23] N. Mohamed Ali, M. M. A. El Hamid, and A. Youssif, “Sentiment Analysis for Movies Reviews Dataset Using Deep Learning Models,” *International Journal of Data Mining & Knowledge Management Process*, vol. 09, no. 03, pp. 19–27, 2019.

[24] A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun, and J. Dean, “A guide to deep learning in healthcare,” *Nature Medicine*, vol. 25, no. 1, pp. 24–29, 2019. [Online]. Available: http://dx.doi.org/10.1038/s41591-018-0316-z

[25] H. Shan, Y. Liu, and T. Stefanov, “A Simple Convolutional Neural Network for Accurate P300 Detection and Character Spelling in Brain Computer Interface,” in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*. California: International Joint Conferences on Artificial Intelligence Organization, jul 2018, pp. 1604–1610. [Online]. Available: https://www.ijcai.org/proceedings/2018/222

[26] G. Pandarinathan, S. Mishra, A. M. Nedumaran, P. Padmanabhan, and B. Gulyás, “The potential of cognitive neuroimaging: A way forward to the mind-machine interface,” *Journal of Imaging*, vol. 4, no. 5, 2018.

[27] H. Gao, R. M. Walker, P. Nuyujukian, K. A. Makinwa, K. V. Shenoy, B. Murmann, and T. H. Meng, “HermesE: A 96-channel full data rate direct neural interface in 0.13 μm CMOS,” *IEEE Journal of Solid-State Circuits*, vol. 47, no. 4, pp. 1043–1055, 2012.

[28] E. López-Larraz, A. Sarasola-Sanz, N. Irastorza-Landa, N. Birbaumer, and A. Ramos-Murguialday, “Brain-machine interfaces for rehabilitation in stroke: A review,” *NeuroRehabilitation*, vol. 43, no. 1, pp. 77–97, 2018.

[29] H. Cecotti, B. Rivet, M. Congedo, C. Jutten, O. Bertrand, E. Maby, and J. Mattout, “A robust sensor-selection method for P300 brain – computer interfaces,” no. February, 2011.

[30] M. P. Branco, “Boosting BCI Technology,” Ph.D. dissertation, 2018.

[31] B. He, *Neural Engineering*, 2nd ed., B. He, Ed. Springer Science & Business Media, 2013.

[32] L. F. Nicolas-Alonso and J. Gomez-Gil, "Brain computer interfaces, a review," *Sensors*, vol. 12, no. 2, pp. 1211–1279, 2012.

[33] M. Velliste, S. Perel, M. C. Spalding, A. S. Whitford, and A. B. Schwartz, "Cortical control of a prosthetic arm for self-feeding," *Nature*, vol. 453, no. 7198, pp. 1098–1101, 2008.

[34] M. A. . L. . Nicolelis and J. K. . Chapin, "Controling Robots with the Mind," 2002.

[35] L. R. Hochberg, M. D. Serruya, G. M. Friehs, J. A. Mukand, M. Saleh, A. H. Caplan, A. Branner, D. Chen, R. D. Penn, and J. P. Donoghue, "Neuronal ensemble control of prosthetic devices by a human with tetraplegia," *Nature*, vol. 442, no. 7099, pp. 164–171, 2006.

[36] W. Truccolo, G. M. Friehs, J. P. Donoghue, and L. R. Hochberg, "Primary motor cortex tuning to intended movement kinematics in humans with tetraplegia," *Journal of Neuroscience*, vol. 28, no. 5, pp. 1163–1178, 2008.

[37] "Neuralink," 2019. [Online]. Available: https://www.neuralink.com/

[38] E. Musk, "An Integrated Brain-Machine Interface Platform With Thousands of Channels," *Journal of Medical Internet Research*, vol. 21, no. 10, p. e16194, oct 2019. [Online]. Available: http://www.jmir.org/2019/10/e16194/

[39] F. L. e. Chang S. Nam, Anton Nijholt, *Brain-Computer Interfaces Handbook: Technological and Theoretical Advances*, C. S. Nam, A. Nijholt, and F. Lotte, Eds. CRC Press, 2018, vol. 73, no. January.

[40] E. C. Leuthardt, G. Schalk, J. R. Wolpaw, J. G. Ojemann, and D. W. Moran, "A brain-computer interface using electrocorticographic signals in humans," *Journal of Neural Engineering*, vol. 1, no. 2, pp. 63–71, 2004.

[41] G. Schalk, J. Kubánek, K. J. Miller, N. R. Anderson, E. C. Leuthardt, J. G. Ojemann, D. Limbrick, D. Moran, L. A. Gerhardt, and J. R. Wolpaw, "Decoding two-dimensional movement trajectories using electrocorticographic signals in humans." *Journal of neural engineering*, vol. 4, no. 3, pp. 264–275, 2007.

[42] G. Schalk, K. J. Miller, N. R. Anderson, J. A. Wilson, M. D. Smyth, J. G. Ojemann, D. W. Moran, J. R. Wolpaw, and E. C. Leuthardt, "Two-dimensional movement control using electrocorticographic signals in humans," *Journal of Neural Engineering*, vol. 5, no. 1, pp. 75–84, 2008.

[43] J. Malmivuo and R. Plonsey, *Bioelectromagnetism: Principles and Applications of Bioelectric and Biomagnetic Fields*. Oxford University Press, 1995.

[44] D. S. Tan and A. Nijholt, *Brain-Computer Interfaces*, ser. Human-Computer Interaction Series, D. S. Tan and A. Nijholt, Eds. London: Springer London, feb 2010, vol. 44, no. 8. [Online]. Available: http://link.springer.com/10.1007/978-1-84996-272-8

[45] V. Jurcak, D. Tsuzuki, and I. Dan, "10/20, 10/10, and 10/5 systems revisited: Their validity as relative head-surface-based positioning systems," *NeuroImage*, vol. 34, no. 4, pp. 1600–1611, feb 2007. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1053811906009724

[46] T. Al-ani and D. Trad, "Intelligent and Biosensors." Intech, 2010, ch. 2.

[47] R. Dinteren, M. Arns, M. L. Jongsma, and R. P. Kessels, "P300 development across the lifespan: A systematic review and meta-analysis," *PLoS ONE*, vol. 9, no. 2, 2014.

[48] C. C. Duncan, R. J. Barry, J. F. Connolly, C. Fischer, P. T. Michie, R. Näätänen, J. Polich, I. Reinvang, and C. V. Petten, "Event-related potentials in clinical research : Guidelines for eliciting , recording , and quantifying mismatch negativity , P300 , and N400," *Clinical Neurophysiology*, vol. 120, no. 11, pp. 1883–1908, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.clinph.2009.07.045

[49] N. Sharma, "Single-trial P300 Classification using PCA with LDA, QDA and Neural Networks," pp. 1–17, 2017. [Online]. Available: http://arxiv.org/abs/1712.01977

[50] A. Rakotomamonjy and V. Guigue, "BCI Competition III: Dataset II- Ensemble of SVMs for BCI P300 Speller," vol. 55, no. 3, pp. 1147–1154, 2008.

[51] M. Falkenstein, J. Hoormann, S. Christ, and J. Hohnsbein, "ERP components on reaction errors and their functional significance: a tutorial," *Biological Psychology*, vol. 51, no. 2-3, pp. 87–107, jan 2000. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0301051199000319

[52] R. Chavarriaga, A. Sobolewski, and J. d. R. Millán, "Errare machinale est: the use of error-related potentials in brain-machine interfaces," *Frontiers in Neuroscience*, vol. 8, no. 8 JUL, pp. 1–13, jul 2014. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fnins.2014.00208/abstract

[53] B. Dal Seno, M. Matteucci, and L. Mainardi, "Online detection of P300 and error potentials in a BCI speller," *Computational Intelligence and Neuroscience*, vol. 2010, 2010.

[54] N. M. Schmidt, B. Blankertz, and M. S. Treder, "Online detection of error-related potentials boosts the performance of mental typewriters," *BMC Neuroscience*, vol. 13, no. 1, 2012.

[55] M. Spüler, M. Bensch, S. Kleih, W. Rosenstiel, M. Bogdan, and A. Kübler, "Online use of error-related potentials in healthy users and people with severe motor impairment increases performance of a P300-BCI," *Clinical Neurophysiology*, vol. 123, no. 7, pp. 1328–1337, 2012.

[56] S. Bhattacharyya, A. Konar, D. N. Tibarewala, and M. Hayashibe, “A Generic Transferable EEG Decoder for Online Detection of Error Potential in Target Selection,” vol. 11, no. May, pp. 1–13, 2017.

[57] P. W. Ferrez and J. Del R. Millán, “Error-related EEG potentials generated during simulated brain-computer interaction,” *IEEE Transactions on Biomedical Engineering*, vol. 55, no. 3, pp. 923–929, 2008.

[58] I. Iturrate, L. Montesano, and J. Minguez, “Task-dependent signal variations in EEG error-related potentials for brain–computer interfaces,” *Journal of Neural Engineering*, vol. 10, no. 2, p. 026024, apr 2013. [Online]. Available: https://iopscience.iop.org/article/10.1088/1741-2560/10/2/026024

[59] R. Phlypo, N. Jrad, S. Rousseau, and M. Congedo, “A non-orthogonal SVD-based decomposition for phase invariant error-related potential estimation,” in *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE, aug 2011, pp. 6963–6966. [Online]. Available: http://ieeexplore.ieee.org/document/6091760/

[60] J. Omedes, I. Iturrate, L. Montesano, and J. Minguez, “Using frequency-domain features for the generalization of EEG error-related potentials among different tasks,” in *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, jul 2013, pp. 5263–5266. [Online]. Available: http://ieeexplore.ieee.org/document/6610736/

[61] I. Iturrate, R. Chavarriaga, L. Montesano, J. Minguez, and J. Millán, “Latency correction of event-related potentials between different experimental protocols,” *Journal of Neural Engineering*, vol. 11, no. 3, p. 036005, jun 2014. [Online]. Available: https://iopscience.iop.org/article/10.1088/1741-2560/11/3/036005

[62] E. M. Ventouras, P. Asvestas, I. Karanasiou, and G. K. Matsopoulos, “Classification of Error-Related Negativity (ERN) and Positivity (Pe) potentials using kNN and Support Vector Machines,” *Computers in Biology and Medicine*, vol. 41, no. 2, pp. 98–109, 2011. [Online]. Available: http://dx.doi.org/10.1016/j.compbiomed.2010.12.004

[63] J. M. M. Torres, T. Clarkson, E. A. Stepanov, C. C. Luhmann, M. D. Lerner, and G. Riccardi, “Enhanced Error Decoding from Error-Related Potentials using Convolutional Neural Networks,” in *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, vol. 2018-July. IEEE, jul 2018, pp. 360–363. [Online]. Available: https://ieeexplore.ieee.org/document/8512183/

[64] R. Chavarriaga and J. del R. Millán, “Learning From EEG Error-Related Potentials in Noninvasive Brain-Computer Interfaces,” *IEEE Transactions on Neural Systems and*

Rehabilitation Engineering, vol. 18, no. 4, pp. 381–388, aug 2010. [Online]. Available: https://ieeexplore.ieee.org/document/5491194http://ieeexplore.ieee.org/document/5491194/

[65] S. Wang, C.-J. Lin, C. Wu, and W. A. Chaovalitwongse, "Early Detection of Numerical Typing Errors Using Data Mining Techniques," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 41, no. 6, pp. 1199–1212, nov 2011. [Online]. Available: http://ieeexplore.ieee.org/document/5752873/

[66] G. Daniel, *Principles Of Artificial Neural Networks (3rd Edition)*, ser. Advanced Series In Circuits And Systems. World Scientific Publishing Company, 2013. [Online]. Available: https://books.google.pt/books?id=Zz27CgAAQBAJ

[67] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of trends in Practice and Research for Deep Learning," pp. 1–20, nov 2018. [Online]. Available: http://arxiv.org/abs/1811.03378

[68] J. M. J. Murre and D. P. F. Sturdy, "The connectivity of the brain: multi-level quantitative analysis," *Biological Cybernetics*, vol. 73, no. 6, pp. 529–545, nov 1995. [Online]. Available: http://link.springer.com/10.1007/BF00199545

[69] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303–314, dec 1989. [Online]. Available: http://link.springer.com/10.1007/BF02551274

[70] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 31, no. 3, pp. 606–660, 2017.

[71] "Kernel (Image processing)," 2020. [Online]. Available: https://en.wikipedia.org/wiki/Kernel{_}(image{_}processing)

[72] I. Goodfellow, Y. Bengio, and A. Courville, "Convolutional Networks," in *Deep Learning*. MIT Press, 2016, ch. 6, pp. 326–366. [Online]. Available: http://www.deeplearningbook.org/contents/convnets.html

[73] ——, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[74] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," pp. 1–18, 2012. [Online]. Available: http://arxiv.org/abs/1207.0580

[75] N. Srivastava, G. Hinton, A. Sutskever, K. Ilya, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning Research*, vol. 15, 2014.

[76] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *32nd International Conference on Machine Learning, ICML 2015*, vol. 1, pp. 448–456, 2015.

[77] D. Mishkin, N. Sergievskiy, and J. Matas, “Systematic evaluation of convolution neural network advances on the Imagenet,” *Computer Vision and Image Understanding*, vol. 161, pp. 11–19, aug 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314217300814https://github.com/ducha-aiki/caffenet-benchmarkhttps://linkinghub.elsevier.com/retrieve/pii/S1077314217300814

[78] S. Linnainmaa, “The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors.” Master thesis, Univ. Helsinki, 1970.

[79] S. Ruder, “An overview of gradient descent optimization algorithms,” pp. 1–14, sep 2017. [Online]. Available: http://arxiv.org/abs/1609.04747

[80] L. Luo, Y. Xiong, Y. Liu, and X. Sun, “Adaptive Gradient Methods with Dynamic Bound of Learning Rate,” *The Dictionary of Genomics, Transcriptomics and Proteomics*, no. 2018, pp. 1–1, feb 2019. [Online]. Available: http://doi.wiley.com/10.1002/9783527678679.dg05534http://arxiv.org/abs/1902.09843

[81] H. Cecotti and A. Gräser, “Convolutional neural networks for P300 detection with application to brain-computer interfaces,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 3, pp. 433–445, 2011.

[82] T.-j. Luo, Y.-c. Fan, J.-t. Lv, and C.-l. Zhou, “Deep reinforcement learning from error-related potentials via an EEG-based brain-computer interface,” in *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, dec 2018, pp. 697–701. [Online]. Available: https://ieeexplore.ieee.org/document/8621183/

[83] R. Manor and A. B. Geva, “Convolutional Neural Network for Multi-Category Rapid Serial Visual Presentation BCI,” *Frontiers in Computational Neuroscience*, vol. 9, no. DEC, pp. 1–12, dec 2015. [Online]. Available: http://journal.frontiersin.org/Article/10.3389/fncom.2015.00146/abstract

[84] M. Liu, W. Wu, Z. Gu, Z. Yu, F. Qi, and Y. Li, “Deep learning based on Batch Normalization for P300 signal detection,” *Neurocomputing*, vol. 275, pp. 288–297, jan 2018. [Online]. Available: https://doi.org/10.1016/j.neucom.2017.08.039https://linkinghub.elsevier.com/retrieve/pii/S0925231217314601

[85] "Data sets - BNCI Horizon 2020." [Online]. Available: http://bnci-horizon-2020.eu/database/data-sets

[86] C. Wirth, P. M. Dockree, S. Harty, E. Lacey, and M. Arvaneh, "Towards error categorisation in BCI: Single-trial EEG classification between different errors," *Journal of Neural Engineering*, vol. 17, no. 1, 2020.

[87] M. Spüler, C. Niethammer, and W. Rosenstiel, "Classification of error-related potentials in EEG during continuous feedback," *6th International Brain-Computer Interface Conference*, pp. 8–11, 2014.

[88] S. Park and N. Kwak, "Analysis on the Dropout Effect in Convolutional Neural Networks," ser. Lecture Notes in Computer Science, S.-H. Lai, V. Lepetit, K. Nishino, and Y. Sato, Eds. Cham: Springer International Publishing, 2017, vol. 10112, pp. 189–204. [Online]. Available: http://link.springer.com/10.1007/978-3-319-54184-6http://link.springer.com/10.1007/978-3-319-54184-6{_}12

[89] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for Simplicity: The All Convolutional Net," *3rd International Conference on Learning Representations, ICLR 2015 - Workshop Track Proceedings*, pp. 1–14, dec 2014. [Online]. Available: http://arxiv.org/abs/1412.6806

[90] R. Ponnuru, A. K. Pookalangara, R. K. Nidamarty, and R. K. Jain, "CIFAR-10 Classification using Intel Optimization for TensorFlow," 2017.

[91] D. Chicco and G. Jurman, "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation," *BMC Genomics*, vol. 21, no. 1, pp. 1–13, 2020.

[92] "Welcome to Colaboratory - Colaboratory." [Online]. Available: https://colab.research.google.com

[93] H. He, "PyTorch vs TensorFlow," 2019. [Online]. Available: http://horace.io/pytorch-vs-tensorflow/https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/

[94] W. Falcon, "PyTorch Lightning," *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning Cited by*, vol. 3, 2019. [Online]. Available: https://github.com/PyTorchLightning/pytorch-lightning

[95] "Comet - Build better models faster." [Online]. Available: https://www.comet.ml/

[96] R. Correia, "Error perception classification in BCI using CNN," 2020. [Online]. Available: https://github.com/LeafarCoder/BCI-Feedback-Classification-using-CNN

[97] S. C. Kleih, F. Nijboer, S. Halder, and A. Kübler, "Motivation modulates the P300 amplitude during brain – computer interface use," *Clinical Neurophysiology*, vol. 121, no. 7, pp. 1023–1031, 2010. [Online]. Available: http://dx.doi.org/10.1016/j.clinph.2010.01.034

[98] R. T. Schirrmeister, J. T. Springenberg, L. D. J. Fiederer, M. Glasstetter, K. Eggensperger, M. Tangermann, F. Hutter, W. Burgard, and T. Ball, "Deep learning with convolutional neural networks for EEG decoding and visualization," *Human Brain Mapping*, vol. 38, no. 11, pp. 5391–5420, nov 2017. [Online]. Available: http://doi.wiley.com/10.1002/hbm.23730

A

Previous work summary

Table A.1: Dataset and pre-processing used in various studies from the literature

Authors [Year]	Model name	Dataset	Number of subjects	Number of target ERPs [target / no target (%)]	Fix class imbalance	Sampling frequency (Hz) [resampled to]	Filter [order; low-high(Hz)]	Normalization	Epoching (ms) [onset; end]
Cecotti et al. [2011]	CCNN	Dataset II (BCI Competition II)	2	11100 [17% / 83%]	-	240 [120]	Bandpass [8; 0.1-20]	Mean & std.dev for individual electrodes	[0; 650]
Manor et al. [2015]	CNN-R	Original + BCI Competition III (for benchmark)	15 + 2	- [10% / 90%]	Replicate the target class	256 [64]	Highpass [-; 0.1]	Mean & std.dev for individual electrodes	[-100; 900]
Liu et al. [2017]	BN3	Dataset II (BCI Competition III) Dataset IIb (BCI Competition II)	3	38730 [17% / 83%]	Replicate the target class (4 times)	240	Bandpass [8; 0.1-20]	-	[0; 667]
Shan et al. [2018]	OCLNN	Dataset II (BCI Competition III) Dataset IIb (BCI Competition II)	3	13290 [18% / 82%]	-	240	Bandpass [-; 0.1-20]	Mean & std.dev for individual pattern and electrodes	[0; 1000]
Torres et al. [2018]	ConvNet	Monitoring Error-Related Potential	6	1322 [20% / 80%]	-	512	-	ZCA whitening process	[-200; 900]
Luo et al. [2018]	CNN-L	Original dataset	12	9000 (target + no target)	-	50	Bandpass [-; 0.1-30]	-	[-50; 500]
Bellary et al. [2019]	ConvArch	Monitoring Error-Related Potential	6	1322 [20% / 80%]	Replicate the target class	512	Bandpass [-; 1-10]	-	[0; 1000]

Table A.2: Architecture of various CNN models from the literature

Authors [Year]	Model name	Architecture [1]	Input size	Output size	Activation function(s)	Loss function
Cecotti et al. [2011]	CCNN	C[10/1x64] >AF >C[50/13x1;13x1] > AF >FC[100] >FC[2]	64x78 [ch x t]	2	Sigmoid	MSE
Manor et al. [2015]	CNN-R	C[96/1x64] >AF >MP[3x1;2] >C[128/6x1] > AF >MP[3x1;2] >C[128/6x1] >AF >D[?] > FC[2048] >D[?] FC[4096] >SM[2]	64x64 [t x ch]	2	ReLU	Binary Cross Entropy
Liu et al. [2017]	BN3	BN >C[16/1x64] >C[1/20x1;20] >BN > AF[ReLU] >D[0.2] >FC[128] >AF[TanH] > D[0.2] >FC[128] >AF[TanH] >FC[1] >AF[Sigm]	160x64 [t x ch]	1	ReLU Tanh Sigmoid	-
Shan et al. [2018]	OCLNN	C[16/64x16;1x16] >AF >D[0.25] >SM[2]	64x240 [ch x t]	2	ReLU	Binary Cross Entropy
Torres et al. [2018]	ConvNet	C[128/20x20] >AF >MP[5x5;2x2] >C[64/10x10] > AF >MP[2x2;2x2] >FC[128] >SM[2]	64x563 [ch x t]	2	ELU	-
Luo et al. [2018]	CNN-L	C[40/1x25] >AF >C[40/64x1] > BN >AF >AP[75x1;15x1] > D[0.5] >FC[2]	64x276 [ch x t]	2	ELU	-
Bellary et al. [2019]	ConvArch	C[16/2x64] >C[32/1x64] >AF > MP[1x2] >C[32/1x32] >AF >MP[1x2] > C[64/1x16] >AF >MP[1x2] >SM[2]	2x512 [ch x t]	2	ReLU	-

[1] List of abbreviations used:
C[$m \setminus k; s$] is a convolutional layer with m output feature maps, kernel of size k and stride s.
AF is an activation function (when different are used in the same model, its name is indicated inside square brackets).
MP[$k; s$] is a max-pooling layer with kernel of size k and stride s.
AP[$k; s$] is an average-pooling layer with kernel of size k and stride s.
D[r] is a dropout layer with dropout rate r.
FC[n] is a fully connected layer with n output nodes.
BN is a batch normalization layer.

Table A.3: Train settings of various CNN models from the literature

Authors [Year]	**Model name**	**Optimizer**	**Learning rate**	**Momentum**	**Other optimizer parameters**	**Epochs**	**Batch size**	**Average train time (minutes)**	**Systems specs**
Cecotti et al. [2011]	CCNN	-	0.2	-	-	15	-	10	Intel Core 2 Duo T7500 CPU
Manor et al. [2015]	CNN-R	SGD	0.001	0.9	-	-	1	30	NVIDIA GTX 650 GPU
Liu et al. [2017]	BN3	Adam	0.00001	0.9	$\beta_2 = 0.999$ $\epsilon = 10^{-8}$	15	64	10	Intel i7 3770 CPU with Windows 10 Pro OS without GPU acceleration
Shan et al. [2018]	OCLNN	SGD	0.01	0.9	Weight decay = 0.0005	15	128	-	NVIDIA GeForce GTX 980 Ti GPU
Torres et al. [2018]	ConvNet	-	0.0001	-	Weight decay = 0.002	1200	120	-	-
Luo et al. [2018]	CNN-L	Adam	-	-	Early stopping applied	-	-	-	-
Bellary et al. [2019]	ConvArch	SGD	0.001	0.9	Weight decay = 0.00001	1000	512	-	-