POLITECNICO DI TORINO

MASTER's Degree in BIOMEDICAL ENGINEERING

MASTER's Degree Thesis

EEGNet for Real-time EEG-Based Stress Analysis

Supervisors

Prof. Francesco Paolo ANDRIULLI

Prof. Sadasivan PUTHUSSERYPADY

Dott. Matteo SAIBENE

Candidate

Gioele TIRABOSCHI

JULY 2025

Abstract

Chronic stress is an increasingly critical issue for public health, but the current methods to detect it are often subjective, slow, or inadequate for a real-time application. This project proposes a system for an automatic analysis of mental stress based on electroencephalographic (EEG) signals, leveraging a Convolutional Neural Network (CNN) from the EEGNet architecture [1]. The model was optimised to achieve high performance in terms of both accuracy and processing speed, to enable real-time integration in wearable and portable devices.

The model has been trained on the public SAM40 [2] dataset, and its hyperparameters were fine-tuned using K-fold cross-validation. To evaluate generalisation capabilities, the model was also tested on newly collected EEG data specifically acquired for this project across five sessions. Results show a classification accuracy of 92.73%±2.08% when all sessions were included in the training set, and 67.65%±6.76% when only the first four sessions were used for training and the last session for testing. This indicates that including data from the target subject's session is essential to improve the model's performance.

The inference speed of the complete pipeline, including data loading, preprocessing, preparation, and classification, was also evaluated. For each 2-second-long segment, the system required 252.7 ms ± 45.5 ms, corresponding to an Information Transfer Rate (ITR) of 16.62 bits/min. This latency is compatible with real-time applications. However, more than 70% of the processing time is currently consumed by the preprocessing step, which includes Independent Component Analysis (ICA) for artefact removal.

Future work should aim to optimise the preprocessing pipeline to reduce computational load without compromising artefact removal quality. Additionally, although the current implementation operates offline on pre-recorded data, transitioning to an online, real-time system represents a key next step.

Beyond stress detection, the proposed model has the potential to be adapted to classify other cognitive states, such as fatigue, distraction, or cognitive overload. When integrated into neurofeedback systems, it could enable real-time interventions for self-regulation and burnout prevention, paving the way for intelligent, adaptive tools to address mental well-being.

ACKNOWLEDGMENTS

I want to extend my sincere thanks to Sada and Matteo for their unwavering support throughout this thesis, and to Professor Andriulli for accepting being my supervisor.

Table of Contents

List of Figures

List of Tables

Acronyms

ADC	Analog-Digital Converter.
AI	Artificial Intelligence.
ApEn	Approximate Entropy.
AR	Auto Regressive.
BCI	Brain-Computer Interfaces.
BLSTM	Bidirectional Long Short-Term Memory.
BPF	Band Pass Filter.
CAR	Common Average Reference.
CNN	Convolutional Neural Network.
DL	Deep Learning.
DNN	Deep Neural Network.
DT	Decision Trees.
DWT	Discrete Wavelet Transform.
ECG	Electrocardiogram.
EEG	Electroencephalogram.
ELU	Exponential Linear Unit.
EMG	Electromyography.
FFT	Fast Fourier Transform.
fMRI	functional Magnetic Resonance Imaging.
FT	Fourier Transform.
GSR	Galvanic Skin Response.
HPA	Hypothalamic-Pituitary-Adrenal.
HRV	Heart Rate Variability.
IFFT	Inverse Fast Fourier Transform.
ICA	Independent Component Analysis.

ITR	Information Transfer Rate.
LDA	Linear Discriminant Analysis.
LPF	Low Pass Filter.
LR	Logistic Regression.
LSL	Lab Streaming Layer.
LSTM	Long Short-Term Memory.
MI	Motor Imagery.
ML	Machine Learning.
MLP	Multi Layer Perceptron.
NN	Neural Network.
NIRS	Near Infrared Spectroscopy.
PCA	Principal Component Analysis.
PFC	Prefrontal Cortex.
ReLU	Rectified Linear Unit.
RNN	Recursive Neural Network.
SCWT	Stroop Colour Word Test.
SDCAN	Symmetrical Deep Convolutional Adversarial Network.
SGD	Stochastic Gradient Descent.
SMO	Sequential Minimal Optimization.
SNR	Signal-to-Noise Ratio.
SSVEP	Steady-State Visually Evoked Potentials.
STFT	Short-Time Fourier Transform.
SVM	Support Vector Machine.
VGG	Visual Geometry Group.
VR	Virtual Reality.
WHO	World Health Organization.

Chapter 1

Introduction

1.1 Motivation and Public Health Relevance

Chronic stress represents an increasing problem for public health. However, the available methods for detecting it are still mostly subjective, slow, or not suitable for real-time applications. It is then necessary to develop an objective, fast, and non-invasive system that can detect stress when it manifests itself.

Stress is a determinant factor in many physical and mental pathologies, such as cardiovascular diseases, anxiety, and depression [3, 4]. Since the COVID-19 pandemic began, the rise of remote work has led to a significant increase in stress-related illnesses. Furthermore, the increasing concern about irreversible climate change, the coming of new wars, and world instability did nothing more than generally enhance preoccupation and stress.

According to the World Health Organisation (WHO), stress-related disorders are responsible for more than 1 trillion dollars per year in productivity loss [5]. This figure underscores the immense economic burden of chronic stress, highlighting not only the direct costs of healthcare and treatment but also the indirect losses derived from reduced output, absenteeism, and impaired cognitive function in the workforce. This colossal financial drain on the global economy emphasises the urgent need for effective stress detection and management strategies, as investments in this area could yield substantial returns in both public health and economic prosperity.

The possibility to detect stress early and in real-time would allow timely interventions, improving quality of life. A real-time stress monitor could alert a driver to take a break, or notify a student to pause before reaching a burnout, or it could be integrated in a workspace environment to adapt task difficulty based on cognitive load.

1.2 EEG and BCI as Tools for Stress Detection

It is known that stress affects brain signals by modulating its activity in specific frequency bands. By studying these variations, it's possible to get insights to classify mental states as stressed or relaxed [6]. Recent evolution in Brain-Computer

Interfaces (BCI), systems that can connect the human brain to external devices, and Electroencephalography (EEG), a non-invasive brain signal acquisition technique with a high temporal resolution, have made it possible to directly monitor brain activity for these goals.

1.3 Limitations of Current Literature

Literature proposed numerous approaches based on Machine Learning and Deep Learning algorithms to classify stress from EEG signals. These models are very effective on the EEG data, often achieving accuracies higher than 90% (see Sec. 3.3.4). However, the vast majority of the studies do not focus on real-time analysis. Traditional pipelines rely on extensive preprocessing, including manual Independent Component Analysis (ICA) and filtering, which are difficult to automate and are usually more precise than the automatised algorithms. Furthermore, studies often work with computationally heavy models, usually difficult to apply in real-time. Then, the models are usually tested on test sets from the same dataset used for the training. In this way, they consist of recordings of the exact same type and acquisition conditions as the training data, limiting the generalisation to new subjects. In the end, the articles analysed use private datasets, making it difficult to replicate the studies.

1.4 Project Objectives and Methodology

In this project, an EEG-based stress classifier based on Convolutional Neural Networks (CNN) will be developed and evaluated, exploiting the EEGNet architecture [1], which has never been used in stress analysis before. The main goal is to optimise the model in terms of velocity and accuracy, to get a reliable net with inference times compatible with real-time usage. This way, the algorithm might be applied in everyday wearable or mobile devices to detect and intervene in high-level stress states. The model is first trained on a public dataset, the SAM40 dataset [2], and then it is tested on new EEG data, acquired specifically to evaluate the capacity of the model to work on a completely different dataset.

The classifier achieved great performance both during the training and during the testing, except in the experiments involving the testing on subjects not present in the training set. The results suggest that, to achieve high accuracy, the classifier still requires data from the tested subject in the training set, indicating limited generalisation capability.

Nevertheless, the results show that the use of CNN in stress analysis is a promising path for real-life contexts, such as wearable devices or an adaptive system for mental health. In particular, the EEGNet model, even though it had not been used before in stress analysis, was found to be particularly fit for the task. Its convolutions use different kernel sizes, and this allows it to find patterns both in time, if a one-dimensional kernel is applied, and space, by using bidimensional kernels.

Beyond stress detection, the methodology developed in this project opens the door to broader applications in cognitive monitoring. Through slight adaptations, it might be extended to classify other mental states, such as fatigue, distraction, or cognitive overload. Additionally, by integrating the model in a neurofeedback framework, it might allow real-time intervention related to self-regulation, focusing improvement, or burnout prevention. This leads to an intelligent future system that will be able to dynamically respond to cognitive and emotive users' needs.

1.5 Thesis Structure

The thesis includes the following chapters:

Chapter 2 presents theoretical bases, including an overview of the human brain, EEG characteristics, stress and its impact, and principles of BCI.

Chapter 3 describes the state-of-the-art in EEG stress detection, including stress-inducing techniques and classification methods.

Chapter 4 illustrates the materials and methods used: computational setup, dataset, preprocessing techniques, model architecture, new data collection, and evaluation techniques.

Chapter 5 reports the experimental results, including the fine-tuning of the EEGNet's parameters, the model's final accuracies, and the evaluation on the new dataset.

Chapter 6 critically discusses the results and the limits of the work, concluding with ethical considerations related to the study.

Chapter 7 provides a conclusion of the project, suggesting possible future developments.

Chapter 2

Background

This chapter lays out the theoretical background, essential to better understand the study proposed in the paper. It will start by detailing the *human brain*, describing its structure and functions. Then, the discussion will move to the *EEG signal*, a powerful non-invasive method for recording brain activity, enabling the analysis of cognitive and emotive states, such as stress response. A central focus will be on *stress*, defining its nature, how it can be classified, and its effects on human health. The chapter will conclude with an introduction to *BCIs*, systems that enable direct communication between the brain and external devices.

2.1 Brain and EEG signal

The human brain is an extremely complex organ, responsible for the regulation of the cognitive, emotional, and physiological processes. It consists of approximately 86 billion neurons [7] that communicate through electrical and chemical signals, forming complex networks that are the foundation of all cerebral functions. Among the various techniques available for studying brain activity, EEG has emerged as one of the most widely used methods, due to its non-invasive nature, high temporal resolution, and ability to capture neural dynamics in real-time.

This section gives an overview of the structure and functions of the brain, the principles of EEG signal acquisition, and its applications in stress analysis.

Structure and Function

As shown in Fig. 2.1, the human brain is anatomically divided into distinct regions, each contributing to different cognitive and physiological functions. The *frontal lobe*, placed in the anterior part of the cortex, is mainly associated with executive functions, to decisional process, and voluntary movement control. Next to it, the *parietal lobe* plays a crucial role in the processing of sensory input and spatial orientation. The *temporal lobe*, positioned laterally, is essential for auditory tasks, such as language comprehension and memory. The *occipital lobe*, positioned at the posterior end of the brain, is focused on the visual processing [8]. In particular, the *prefrontal cortex*

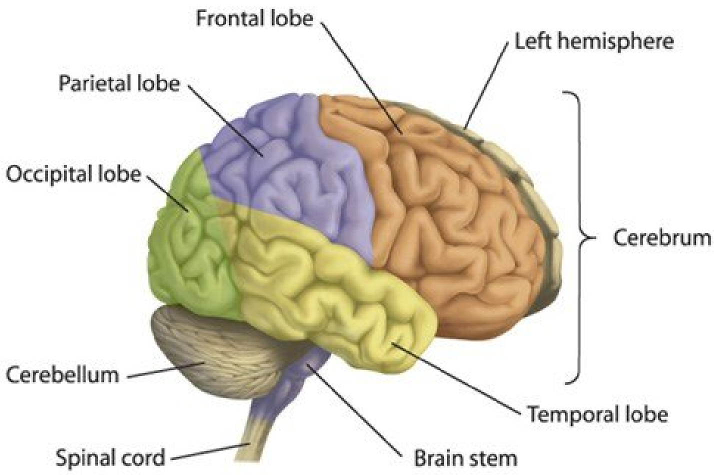

Figure 2.1: Human brain scheme, image taken from here.

in the frontal lobe is associated with EEG activity linked to stress, reflecting its role in stress responses [9].

Below the cerebral cortex, the *cerebellum*, found inferior to the occipital lobe, is fundamental for motor coordination, equilibrium, and fine regulation of voluntary movements. The *brain stem*, comprising structures such as the midbrain, pons, and medulla oblongata, connects the brain and spinal cord, regulating the vital autonomic functions, like breathing, and heart rate [10].

Deeper structures such as the *limbic system*, which includes the amygdala and hippocampus, are crucial for emotional processing and memory formation. In the same way, the *hypothalamus*, a central regulator of the endocrine system, is responsible for homeostatic processes, including the stress response through the hypothalamic-pituitary-adrenal (HPA) axis. These subcortical structures interact extensively with the cortical regions to regulate cognition, emotion, and physiological stability [11].

Neuronal activity in these regions generates electrical potentials that can be measured using EEG. When neurons communicate, they produce synchronised electrical discharges known as brainwaves, which vary in frequency and amplitude depending on the brain's state. These brainwaves are categorised into several bands, each associated with different mental states and functions:

- *Delta waves* (0.5–4 Hz), which are predominant during deep sleep.
- *Theta waves* (4–8 Hz), associated with drowsiness, meditation, and memory consolidation.
- *Alpha waves* (8–12 Hz), which are present during relaxed wakefulness and closed-eye states.
- *Beta waves* (12–30 Hz), linked to active thinking, focus, and stress.
- *Gamma waves* (30–100 Hz), which are involved in higher cognitive processes and information integration.

Understanding these brainwave patterns is essential for interpreting EEG signals and identifying neural correlates of stress. Among these, *beta, gamma* and *alpha* waves are particularly relevant for stress detection. Increased beta activity, especially in the frontal and central regions, has been associated with a higher cognitive load and stress responses. For instance, studies have shown that elevated beta power in areas like Fz (mid-frontal) and F3 (left frontal) is indicative of mental effort and stress during demanding cognitive tasks [6]. This increase in beta activity reflects enhanced cortical arousal and information processing related to the perceived stressor. In the same way, increased gamma activity can point to hyper-excitation, or excessive cognitive effort, which might lead to stress states. On the contrary, stress has also been linked to a reduction in alpha power, because these waves are usually associated with relaxation. Therefore, the analysis of the balance between these bands can provide useful insights regarding stress levels and their neural underpinnings [12].

EEG Signal Acquisition

EEG is a non-invasive technique that measures electrical activity on the scalp using electrodes positioned in specific locations following standardised systems such as the 10-20 or the 10-10 international systems (see Fig.2.2). This system ensures a coherent positioning of the electrodes across different studies, thereby allowing for reproducible results. The electrodes detect voltage fluctuations generated by the summation of post-synaptic potentials in large groups of neurons. These signals are then amplified, filtered, and digitised for analysis.

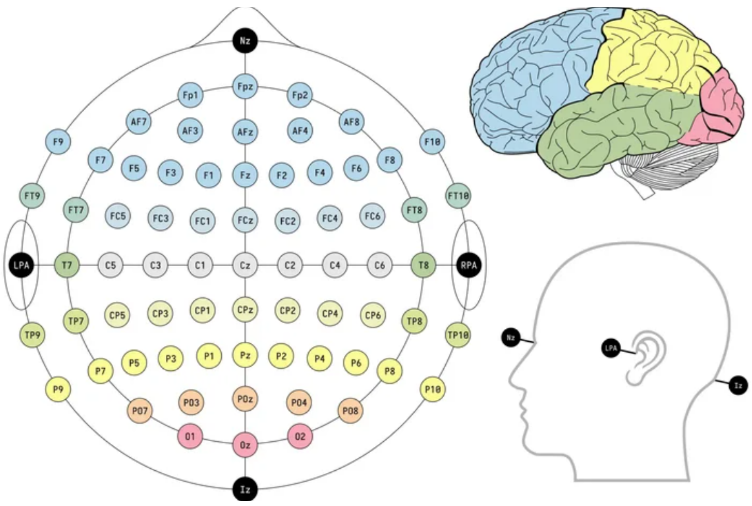

Figure 2.2: EEG electrodes positioning with the 10-10 system; image taken from here.

One of the main advantages of EEG is its high temporal resolution, enabling the capture of neural activity in milliseconds. This makes EEG particularly suitable for studying dynamic processes like stress responses, which could occur rapidly and vary over time. However, EEG also has some limitations, such as a low spatial resolution due to the blurring effect of the skull and scalp, as well as susceptibility to artefacts from muscular activity, eye movements and environmental noise. Advanced signal

Figure 2.3: Three examples of state-of-the-art EEG devices. From left to right: g.tec g.GAMMAsys with dry electrodes for high-density, quick setup; Brain Products BrainCap featuring active electrodes for high signal quality; the portable Emotiv EPOC X, ideal for everyday applications.

processing techniques, such as ICA and other Machine Learning (ML) algorithms, are often employed to mitigate these challenges and extract meaningful information from EEG data.

State-of-the-Art EEG Devices

State-of-the-art EEG devices represent a technological frontier in brainwave acquisition. These modern devices can have a really high channel density, allowing for a high spatial resolution by better capturing neural activity. A key innovation is the integration of dry electrodes, which completely remove the need for conductive gel. This notably reduces preparation time, increases the subject's comfort, and makes the EEG more accessible for application outside of clinical spaces.

Many cutting-edge devices are now *portable and wireless*, facilitating research and the monitoring of EEG signals in every natural environment. This has opened up new possibilities for studying cognition, sleep, stress, and performance in a real-world context.

Examples of these EEG caps and systems include systems like the *g.tec g.GAMMAsys* with its high channel count and dry electrodes options for quick setup (on the left in Fig.2.3), the *BrainCap* series from *Brain Products* (in the middle, in Fig. 2.3), known for its high-density active electrodes, providing exceptional signal quality and spatial resolution, and the increasingly popular wearable solutions such as those from *Emotiv*; for instance, the *Emotiv EPOCH X* (on the right in Fig.2.3) uses fewer electrodes, mainly in frontal and temporal areas, and is very practical for its portability and its potential in all-day-life applications.

Furthermore, electronic advancement led to a higher Signal-to-Noise Ratio (SNR) and higher signal quality. These improvements are making EEG a more and more versatile and powerful tool not only for clinical research, but also for real-life applications such as BCIs, neurofeedback and well-being monitoring.

EEG Applications in Stress Analysis

As previously stated, EEG has been widely used in stress research due to its ability to detect real-time changes in cerebral activity related to stress. Studies have demonstrated that stress alters the power and connectivity of specific brainwave bands, particularly in the frontal and temporal regions. For example, increased beta activity in the prefrontal cortex has been linked to heightened stress levels, while lower alpha activity is often associated with impaired relaxation, and cognitive overload [12].

Recent advancements in EEG technology, combined with ML approaches like CNNs, have further enhanced the capacity to detect and classify stress states. For example, researchers have developed models which use EEG data to differentiate between stress and non-stress conditions with high accuracy. These models are based on characteristics such as spectral power (the strength of brain waves at different frequencies), coherence (the degree of synchronized activity between different brain regions), and asymmetry ratios (differences in brain activity between the left and right hemispheres), which provide insights about neural mechanisms underlying stress [13].

EEG-based BCI systems have shown promising capacity in real-time monitoring and intervention. For example, neurofeedback techniques use EEG signals to provide users with real-time feedback about their cerebral activity [14], enabling them to learn self-regulation strategies to manage stress. Such applications highlight the potential of EEG as a tool for personalised stress interventions.

Challenges and Future Directions

Despite its many advantages, EEG-based stress analysis has still to face many challenges. One major obstacle is the inter-individual variability of the EEG patterns, which could make it difficult to generalise results between populations. Furthermore, while advanced signal processing techniques (such as ICA or Principal Component Analysis) are employed to counteract common EEG limitations like artefact contamination and inherent low spatial resolution, these methods are often computationally expensive and do not always achieve complete artefact removal.

Future research should focus on addressing these challenges, developing faster and more robust algorithms for artefact removal, improving the EEG spatial resolution through advanced techniques of source localisation, and integrating EEG with other modalities, such as the functional Near-InfraRed Spectroscopy (fNIRS), or physiological sensors (e.g. heart rate variability). Additionally, longitudinal studies are needed to better understand how stress-related EEG models evolve with time and in response to interventions.

2.2 Stress

In recent years, stress has emerged as a significant public health issue, particularly in light of the numerous societal disruptions, including the COVID-19 pandemic. The rise of remote work in daily life, the reduction of human interactions, and the growing uncertainty about global safety (wars, climate change, among other factors) contributed to an increase in stress levels among the global population. As people spend more time in isolation, often working at desks for many hours per day, chronic stress has emerged as a major challenge for public health [15, 16]. This section provides an overview of stress, its physiological and psychological mechanisms, its effects on health and the importance of early detection and intervention.

Definition and Mechanisms of Stress

Stress is scientifically defined as the natural response of the body to challenges or demands, also called *stressors* [17]. This response can be manifested at a physical, emotional or psychological level, and is essential to help the subjects adapt to changing environments. The physiological stress response is primarily mediated by the hypothalamic-pituitary-adrenal (HPA) axis, which triggers the release of stress hormones such as cortisol, adrenaline, and noradrenaline [18]. These hormones prepare the body for a rapid reaction, enhancing the heart rate, the blood pressure, and activating energy reserves. This reaction is known as *fight-or-flight*, and is a survival mechanism that individuals use to effectively respond to perceived threats.

Types of Stress: Eustress and Distress

Stress is not intrinsically harmful, and can be classified into two different types: *eustress* and *distress* [18, 19, 20]. *Eustress* refers to a positive form of stress that improves cognitive performance, concentration, and motivation. It usually manifests during short-term challenges that lead to personal growth, higher productivity, and resilience. For instance, eustress could appear during an exam at university, increasing how focused students are, helping them answer the questions correctly, or in athletic competitions, sharpening the performance, or even in a life-threatening situation in an unknown environment, triggering alertness. On the contrary, *distress* describes the negative effects of excessive and prolonged stress, which might drastically reduce cognitive functions, alter sleeping patterns, and contribute to long-term health complications. Understanding the distinction between these types of stress is crucial for developing effective strategies to address this issue.

Effects of Chronic Stress on Health

When the stress response is activated too frequently or is prolonged, it might lead to significant health consequences [21]. Chronic stress is strongly linked to the development of mental health disorders, such as anxiety and depression, as well as physical conditions including cardiovascular disease, immune system dysfunction, and gastrointestinal issues [3, 4]. Prolonged exposure to stress hormones, especially cortisol, might have a harmful impact on various bodily systems, like cancer and other chronic illnesses. Moreover, chronic stress could worsen pre-existing mental health conditions and reduce the overall well-being of individuals.

Prevalence and Societal Impact of Stress-Related Disorders

Mental disorders, like anxiety and conditions related to stress, are some of the most expensive pathologies worldwide for governments [22]. The WHO estimates that depression and anxiety problems cost more than one trillion dollars to the global economy, in productivity loss [5]. In the USA alone, even before the pandemic, mental health cost 193.2 billion dollars per year [23].

Health costs have rapidly increased because of stress conditions, with estimates suggesting that a substantial proportion of medical consultations are influenced or exacerbated by stress-related disturbances [24]. In 2021, a study in the UK revealed that anxiety and depression were the main causes of sick days in the country, representing 50% of the job-related sick days.

Epidemiological studies showed that a significant percentage of the population declares facing anxiety or stress problems, positioning them among the most prevalent chronic disorders globally. Typically, anxiety disorders start manifesting early on, under 15 years of age [4], highlighting how important it is to approach this issue at an early stage and suppress their progression, through early intervention and prevention strategies. Even though the diffusion of these problems is increasing, only a small percentage of the affected subjects receive adequate help. This underscores the immediate need for innovative approaches to detect and manage stress.

2.2.1 Neurobiology of Stress

The physiological response to stress is not solely mediated by the HPA axis, but it deeply involves other important brain regions. The *amygdala*, a fundamental component of the limbic system, plays a crucial role in the processing of fear and emotive responses, rapidly evaluating the potential threats and starting a stress response. The prefrontal cortex, responsible for executive functions such as the decision-making process, usually modulates the activity of the amygdala. However, under chronic stress, the capacity of the prefrontal cortex to control these functions can be compromised, leading to a higher emotive reactivity. Also, the *hippocampus*, vital for memory formation, is highly influenced by stress. A prolonged exposure to stress hormones, particularly cortisol, can lead to a reduction in the volume of the hippocampus, influencing memory and emotional resilience. Chronic stress can also induce structural changes in the prefrontal cortex, contributing to the misregulation of the stress response and to a higher susceptibility to mood disturbances.

The Role of Neuroscience in Stress Analysis

The increasing development of neuroscience and of biomedical research opened new avenues for understanding and addressing stress. In particular, EEG stands out as an ideal tool for stress detection, thanks to its direct and real-time access to brain activity. Stress deeply influences the always-changing mental states, leading to dynamic variations in the brainwaves, which EEG can capture because of its high temporal resolution. On the contrary, other acquisition techniques such as functional Magnetic Resonance Imaging (fMRI) are expensive, non-portable, or with low temporal resolution. In a similar way, Galvanic Skin Response (GSR) and Heart Rate Variability (HRV) provide indirect measures of the activation of the sympathetic nervous system, while EEG provides direct information about neural activity, cognitive and emotional states related to stress. This direct and real-time access to brain activity makes EEG particularly efficient in analysing the cerebral dynamics underlying stress responses, allowing for fast interventions to mitigate them.

EEG and BCI technologies, combined with ML techniques such as CNN, offer promising tools for early detection and intervention [25, 26, 27]. By analysing neural activity patterns associated with stress, researchers can get more information about its fundamental mechanisms, develop strategies to mitigate, and sometimes prevent, its negative effects. This progress shows great potential to improve stress management, especially chronic stress, promoting general well-being in an increasingly stressful world.

Conclusion

Stress is a complex phenomenon that plays a pivotal role in adaptation and survival. However, when it becomes chronic or excessive, it could lead to grave consequences for both mental and physical health. With the increasing prevalence of stress-related conditions, it's crucial to develop solutions for early detection and treatment. Thanks to advancements in neuroscience and technology, researchers are exploring innovative solutions to address stress and its impact on health, contributing to improving the quality of life for individuals worldwide.

2.3 BCI

BCI technology is a revolutionary advancement in the neuroscience field and in biomedical engineering. BCIs are systems which enable direct communication between the brain and external devices, bypassing traditional pathways like muscles and nerves. By translating brain activity into actionable commands, BCIs have the potential to revolutionise various domains, such as health care, assistive technology, and stress management. This section provides an overview of the principles of BCI, its applications in stress analysis, and the challenges associated with its implementation.

Principles of BCI

A BCI system is typically built with five interconnected components, each playing a critical role in the system's functionality. These components are: signal acquisition, signal preprocessing, feature extraction, classification, and application interface. Together, they form a cyclic system that translates brain activity into commands or feedback. A scheme of this system is presented in Fig. 2.4.

1. *Signal Acquisition*: the first component in a BCI system is the signal acquisition. This requires neuroimaging techniques to catch the brain activity. The most used modalities are EEG, fMRI, and Near InfraRed Spectroscopy (NIRS). Between these, EEG is the most used, as stated before, due to its non-invasive nature, portability, high temporal resolution, and low cost. The EEG electrodes, positioned on the scalp, detect electrical potential generated by neural activity, giving real-time data about cerebral dynamics.

2. *Signal pre-processing*: once brain activity has been acquired, it undergoes pre-processing to remove noise and artefacts which could interfere with an accurate analysis. This step is crucial to guarantee the quality of the data. Techniques such as filtering, artefact removal, and signal normalisation are usually applied. Preprocessing prepares raw signals for further analysis, improving their clarity and reliability.

3. *Feature Extraction*: the next step is feature extraction, where patterns or significant characteristics in the signals are identified. These characteristics might include spectral power (for example, alpha, beta or gamma power), Event-Related Potentials (ERP), or connectivity measures (e.g. coherence between different brain regions). The choice of the characteristics depends on the specific application of the BCI: stress analysis will require different features than Motor Imagery (MI) or Steady State Evoked Potentials (SSVEP). Effective feature extraction is essential to properly capture the neural correlates of the user's intentions or mental states.

4. *Classification*: after the feature extraction, the signals are sent to a classification module, where ML algorithms interpret the data and translate it into actionable commands. Common algorithms include Support Vector Machines (SVM), CNNs, and Linear Discriminant Analysis (LDA). These algorithms classify the brain signals into predefined classes, such as different mental states (e.g. stress or relaxation), or types of movements (e.g. left arm up, left arm down). The accuracy and the efficiency of the classification process are essential to studying the performance of the BCI system.

5. *Application Interface*: the final component is the application interface, which works as a bridge, connecting the BCI system to the user through feedback. This interface changes depending on the application. For example, in a neurofeedback system, the interface could give auditory or visual feedback to help users regulate

their brain activity. In a motor control application, the interface could translate the brain signals into commands for a robotic arm or a virtual keyboard. The interface is what makes the BCI system practical and easy to use, enabling a real-world implementation.

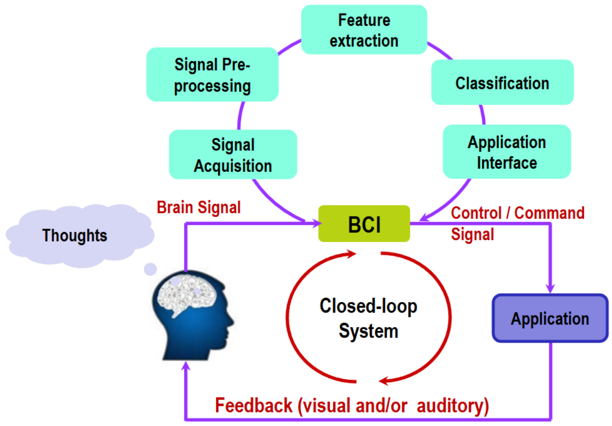

Figure 2.4: Scheme of the interconnected components in a BCI system; image from [28].

Together, these five components form a closed-loop system, where brain activity is always monitored, processed, and translated into actions or feedback. This loop enables a real-time interaction between the user and the BCI system, enabling applications like stress management, assistive technology and cognitive improvement. The integration of these components is what makes BCIs powerful tools to understand and exploit the brain's potential.

Applications of BCI in Stress Analysis

BCI technology has demonstrated great potential in the field of stress detection. By using brain activity in real time, BCIs could provide personalised insights about stress levels in a subject, making it easier to intervene in time. For instance, neurofeedback-based BCIs could use EEG signals to monitor stress-related patterns in the brainwaves, such as an increase in the beta activity in the prefrontal cortex, providing users with real-time feedback. This feedback might help people to learn self-regulation techniques, such as deep breathing, to modulate their stress response.

Another BCI application in the field of stress is the development of adaptive systems, which respond to the mental state of the user. For example, a working environment connected to a BCI system could regulate the illumination, the temperature or the task difficulty depending on the stress levels of the user, promoting a more comfortable and productive environment. At the same time, the BCI-based Virtual Reality (VR) environments have already been used for studying stress reduction, allowing the users to immerse themselves in relaxing landscapes while receiving real-time feedback about their brain activity [25].

Ethical Considerations

The development and the implementation of BCIs raise many ethical considerations. The main one is privacy. BCIs collect highly sensitive brain data, making it essential to have robust security measures to prevent unauthorised access or improper use. Another challenge is the signal interpretation, as current BCIs might not always reflect with high accuracy the intention or the mental state of the subject, leading to wrong interpretations. Furthermore, there is a risk of bias in ML classifiers, where algorithms trained on non-representative datasets might lead to the creation of systems that don't work properly with a larger number of users. Addressing these ethical considerations through careful design and continuous dialogue is fundamental for the responsible development of BCIs.

Chapter 3

State of the art

The first BCI-based systems emerged in the late 1980s and early 1990s. Initial developments focused on applications such as P300-based spellers [29], which used flickering lights in a 6×6 matrix containing alphabetical letters and other symbols. By analysing the P300 signal, which is a positive deflection in the EEG occurring around 300 milliseconds after the user perceives a relevant stimulus, researchers were able to identify the letter the user was looking at. In the last two decades, the field of BCI in biomedical engineering has grown significantly, including applications in neurorehabilitation [30, 31, 32, 33, 34], emotion recognition [35, 36], brain-to-brain communication in humans [37, 38, 39], smart home control [40, 41], and even music composition [42].

Regarding stress analysis, research has been conducted for multiple years to classify stress states into binary (stress vs. no stress [12, 13, 25, 27, 43]) or multiclass categories (e.g., low, moderate, and high stress [44, 45]). These advancements aim to improve real-time stress detection and intervention strategies based on EEG and other physiological signals.

3.1 Methods for inducing stress

To study stress responses, many experimental techniques are used to induce stress in the subjects under controlled conditions. These stressors can broadly be categorised into physiological stressors, which directly challenge the body's homeostasis (e.g., a cold pressor test where a hand is submerged in ice water), and psychological stressors, which involve cognitive or emotional demands (e.g., public speaking, cognitive load tasks). The most commonly used methods are the psychological stressors, presented in the following list. Visual examples of these stress-inducing tasks will be shown in Sec. 4.2.2, in Fig. 4.2.

- *Stroop Test*: this task requires the participant to name the colour of a written word, as fast as possible. The cognitive interference arises from the incongruence between the colour of the word and the word itself, which is the name of a colour (e.g. the word "red" could be written with blue ink). This cognitive conflict increases the cognitive load, thereby raising the stress level, especially when the time available to answer is limited [25, 27, 43, 46].

- *Arithmetic Tasks*: the participants have to solve arithmetical operations, which involve additions, subtractions, multiplications, or divisions of numbers with more than two digits. By increasing the difficulty of the operations, the probability of making mistakes raises, as well as the stress level of the subject. Time limitations make stress increase even more [25, 43, 44, 46].

- *Mirror-Image Matching Task*: in this test, participants have to determine as fast as possible if two images, one next to the other, are mirror-symmetrical. The time pressure contributes to stress induction [12, 13, 27, 43, 47].

- *Stressful Driving Simulation*: in this task, participants are placed in a VR environment, where they have to drive in challenging driving scenarios. These include high-traffic urban areas, complicated crossroads, or highway driving at high speeds. The scenarios are designed to induce stress through time pressure, unexpected obstacles, and the need for rapid decision-making [48].

- *Music and Video Stimuli*: specific auditory and visual stimuli can be used to evoke stress-related emotions. For example, fast-paced and discordant music, or distressing videos might activate stress-related brain activities, while relaxing music or peaceful landscapes can induce relaxation [49, 50].

- *Performance Feedback and Time Pressure*: to further improve stress induction, it is possible to visualise real-time feedback, comparing the participant's performance against a (fictitiously) inflated average score to create a sense of underperformance. Another way to induce stress is to create a noisy and disturbing environment, which would make it difficult for the subject to focus. Additionally, showing a timer with a countdown enhances the time-pressure, especially when little time remains, increasing the participant's stress levels [51].

These methods, often combined, create a controlled environment that induces stress, enabling researchers to effectively study physiological and neurological responses to stress.

It is crucial to note that, for ethical reasons, stress induction in research settings must always be mild and reversible. Participants' well-being cannot be overlooked, and studies are designed to ensure that induced stress does not cause lasting harm or discomfort.

3.2 Classification methods

To classify the signals recorded after stress-inducing tasks, two main supervised approaches are used: Machine Learning (ML) and Deep Learning (DL).

Machine Learning

ML is a subset of Artificial Intelligence (AI) that focuses on the development of algorithms capable of learning from data and improving their performance over time without being explicitly programmed. In the context of EEG-based stress analysis, ML is commonly used to create predictive models that can distinguish between different stress conditions. The accuracy of the ML model strongly depends on the quality of the data and, most of all, on the extracted features. One of the main advantages of ML methods is their capacity to get good results even with relatively small datasets, if they are well-balanced and without noise, and the feature extraction step is optimised.

The most commonly used features in the literature can be categorized into time-domain, frequency-domain, connectivity, and non-linear features. In Tab. 3.1, a description of the most widely adopted EEG features for stress analysis is presented.

Feature Type	**Feature Name**	**Description** [References]
Time-Domain	Mean	Average amplitude of the EEG signal. [52]
	Standard Deviation	Measures the variability of the signal. [52]
	Skewness	Describes the asymmetry of the amplitude distribution. [52]
	Kurtosis	Measures the peakedness of the distribution, useful for detecting transient activity. [52]
	Phase Lag	Quantifies the delay between signals from different brain regions, indicating synchronisation. [50]
	AR Coefficients	Models EEG signals as a linear combination of past values to capture temporal dependencies. [52]
Frequency-Domain	Delta Power (0.5-4 Hz)	Associated with deep sleep and unconscious states. [13, 25, 50]
	Theta Power (4-8 Hz)	Related to relaxation, drowsiness, and cognitive workload. [13, 25, 43, 50]
	Alpha Power (8-12 Hz)	Associated with a relaxed but alert state. [13, 25, 43, 50]
	Beta Power (12-30 Hz)	Related to active thinking and anxiety. [13, 25, 43, 50]
	Gamma Power (>30 Hz)	Linked to high-level cognitive processing and focused attention. [13, 25, 50]
	Power Ratios	Ratios like Theta/Alpha and Beta/Alpha give insights into cognitive and emotional states. [12, 13, 43, 50]
	Wavelet Transform (WT)	Captures transient patterns in EEG signals with time-frequency analysis. [12]
Connectivity	Coherence	Measures synchrony between EEG signals at specific frequencies. [13, 50]
	Asymmetry	Difference in power between different electrodes (e.g. left or right hemisphere), linked to emotional states. [13, 50]
Non-linear	Approximate Entropy (ApEn)	Quantifies signal complexity, where higher values indicate increased irregularity. [52]
	Hurst Exponent	Measures long-term memory or persistence in time series, useful for analysing fractal structure in EEG signals. [52]

Table 3.1: Overview of EEG Features Used for Stress Classification in ML algorithms.

In literature on EEG stress analysis, several ML classifiers are commonly used, each with distinct mechanisms:

1. *Support Vector Machines* (SVM): SVM is a supervised learning algorithm that aims to find the hyperplane that best separates the different classes in the feature space. It works by maximising the margin between the closest points of each class, called support vectors. The articles that used SVM are [13, 25, 43].

2. *Sequential Minimal Optimisation* (SMO): SMO is an efficient algorithm used to train the SVM, particularly with large datasets. The training process in the SVM implies the solution of a complex quadratic programming problem. SMO simplifies it by dividing the problem into subproblems, which can be solved analytically, avoiding the computationally expensive numerical optimisation. This makes SMO particularly useful for SVM training, especially on a large scale. The article which used SMO is [50].

3. *Stochastic Gradient Descent* (SGD): SGD is an optimisation algorithm used for training models, especially with large datasets. Instead of calculating the gradient on the whole dataset, it calculates the gradient for a random batch of data, making the process way faster. This method is often used in both linear and non-linear classification problems in EEG stress analysis. The article which used SGD is [50].

4. *Logistic Regression* (LR): LR is a supervised algorithm used for classification, particularly for binary problems. It predicts the probability that an observation belongs to a specific class. To do so, it applies a sigmoid function to a linear combination of the input variables. The sigmoid function maps the output to the range (0, 1), interpreted as a probability. Depending on whether this probability is higher or lower than a certain threshold, the observation is assigned to a class or the other. The articles which used LR are [13, 50].

5. *Decision Trees* (DT): DT is a model that divides the data into subsets, depending on the value of a feature. The splitting continues recursively, creating branches that represent decisions. The final decision is made at the leaves of the tree. DT is simple to interpret and can handle both categorical and continuous data, making it suitable for stress classification in EEG analysis. The article which used DT is [13].

Deep Learning

DL is a branch of ML that relies on the use of artificial Deep Neural Networks (DNN), which are constituted of many layers of neurons. These models are particularly well-suited for analysing complex data like EEG, given their capacity to learn hierarchical representations. This is useful to learn progressively more abstract representations and automatically extract the most relevant features from raw data, without any explicit human intervention. Different from traditional ML methods, which need a manual feature selection, DL can identify the most significant features directly from data, improving in this way the efficiency and accuracy of the model. One of the most used DL techniques is the CNNs, particularly efficient in the time-series signals analysis like EEG, due to their ability to detect complex spatial and temporal features.

The main difference between ML and DL is in their complexity and in the way in which they learn from data. While traditional ML methods are based on explicit mathematical techniques and require a manual feature extraction, DL is based on much more complex and deep models, which learn automatically from raw data, eliminating the need for human intervention in the feature design. This allows DL to face harder tasks, but it usually requires larger datasets and computational resources.

To address the need for extensive data, techniques such as *data augmentation* are used to artificially increase the number of samples in the dataset by modifying existing ones (e.g., rotating images, adding noise). Additionally, *transfer learning* is a powerful countermeasure where a pre-trained model on a large, generic dataset is fine-tuned for a specific task, significantly reducing the amount of data and computational power required for training. *GPU* computing is used to accelerate the training of these complex Neural Networks (NN).

While large datasets and computational resources are key challenges, others exist, such as interpretability, meaning that the results coming from DL are usually a *black box*, potentially leading to overfitting if not handled correctly, and the need for careful hyperparameter tuning.

DL models have been widely used in EEG-based stress analysis because of their capacity to automatically extract significant features from complex brain signals. The following architectures have been explored in the reviewed studies:

- *Multi-Layer Perceptron* (MLP): MLP is a fully connected feedforward NN, with multiple hidden layers, in which each neuron applies a weighted sum followed by a non-linear activation function. MLPs are efficient for classification tasks, but lack specialised mechanisms to capture temporal or spatial dependencies in EEG signals. The articles which used MLP in stress analysis are [25, 50].

- *Deep Neural Network* (DNN): DNN is a general term for NN with multiple hidden layers that allow a hierarchical learning of the features. They can extract complex features from EEG data, but, to avoid overfitting, they need very large datasets. The articles which used DNNs in stress analysis are [25, 53].

- *Convolutional Neural Network* (CNN): CNNs are DL models designed to extract spatial features through convolutional layers. CNNs are frequently used in EEG-based stress analysis, processing time-frequency representations or raw EEG signals to detect relevant patterns for the stress classification. The articles which used CNNs (alone or in combination) in stress analysis are [13, 27, 44, 47, 49, 53, 51].

- *VGGish-CNN*: A variation of CNN based on the VGG (Visual Geometry Group) architecture, originally developed for image classification. In EEG-based stress analysis, VGGish-CNN is used to analyse spectrogram-like EEG representation, exploiting the deep hierarchical extraction of the features for a higher accuracy. The paper that used VGGish-CNN is [46].
- *Long Short-Term Memory* (LSTM): LSTM is a variant of a Recurrent Neural Network (RNN), designed to model long-range temporal dependencies. LSTMs are particularly adapted for EEG-based stress classification, because they can capture the evolution in time for the brain activity, improving the classification performances. Articles that used LSTM (alone or in combination) in stress analysis are [12, 44, 47, 49, 51, 52].
- *Bidirectional LSTM* (BLSTM): BLSTM is an extension of the LSTM, which elaborates the EEG sequences in both forward and backwards directions. In this way, the model can learn features both from past and future timesteps. This bidirectional processing improves the feature extraction from the EEG signal, making BLSTM highly effective in stress classification. The articles which used BLSTM (alone or in combination) in stress analysis are [47, 52].
- *Symmetrical Deep Convolutional Adversarial Network* (SDCAN): SDCAN is a new framework of DL which integrates the feature extraction based on CNN with adversarial learning. SDCAN uses adversarial inference to automatically capture invariant and discriminative features from the raw EEG signals, improving accuracy and generalisation between subjects. This approach aims to improve the robustness of the model, particularly in cross-subject stress classification scenarios. The article which used SDCAN in stress analysis is [45].

These DL architectures offer multiple advantages: CNNs excel in the extraction of spatial patterns. LSTM and BLSTM capture temporal dependencies, and adversarial networks improve the robustness and the generalisation in EEG-based stress classification.

Comparison of Machine Learning and Deep Learning

Both ML and DL approaches offer unique strengths and weaknesses when applied to EEG-based stress analysis. The choice between them often depends on the specific dataset characteristics, available computational resources, and desired model interpretability.

Tab. 3.2 provides a concise overview of the fundamental differences and trade-offs between traditional ML and DL.

Unsupervised and Semi-Supervised Methods

These approaches are useful when the labelled data is scarce, and they look for hidden patterns in unlabelled data. Some techniques are clustering (grouping similar

FEATURE	ML	DL
Feature Extraction	Manual, domain-specific	Automatic, learned from data
Data Requirements	Works with smaller datasets	Requires large datasets
Computational Resources	Less demanding (CPUs)	Demanding (GPUs needed)
Complexity & Model Depth	Simpler, shallower models	Complex, multi-layered networks
Interpretability	Generally more interpretable	Often a "black box"
Performance on Complex Tasks	May struggle with raw data	Excels with complex raw data
Overfitting Risk	Lower with small datasets	Higher with small datasets

Table 3.2: Comparison of ML and DL for EEG-based Stress Classification.

patterns in the same cluster) and dimensionality reduction (to simplify complex data). *Semi-supervised* methods combine a small quantity of labelled data with many unlabelled data to improve the training, inferring labels or refining decision boundaries.

Even though non-supervised and semi-supervised methods are theoretically well-suited for EEG analysis because of the scarce amount of labelled data, they are rarely applied to stress detection. This is mainly because of the difficulty in validating the extracted patterns, which are usually abstract and cannot be directly interpreted and verified, of the inherently subjectivity of stress, which does not have clear and objective labels, and of the proven effectiveness of the supervised methods, which have always achieved high performance in the field of stress classification.

3.3 Literature review

Tables 3.4 and 3.5 give an overview of published research papers in EEG stress analysis, summarising the information displayed below.

3.3.1 Electrodes Chosen

EEG-driven stress analysis is based mainly on signals from the frontal and frontopolar regions, which have a crucial role in the emotive regulation and cognitive control [54]. The literature highlights the importance of frontal midline theta activity (4–8 Hz), particularly in Fz, which increases under cognitive load and stress. Additionally, frontal alpha asymmetry (8–13 Hz), measured between the left and right prefrontal regions, is a well-established biomarker, with reduced alpha power in the left hemisphere indicating higher stress levels. Increased beta activity (13–30 Hz) in frontal regions is also associated with heightened anxiety and arousal [43].

Beyond frontal electrodes, some studies include temporal, parietal and occipital channels, to capture wider neural stress-related responses. In any case, the choice of the electrodes varies with the methodology and with the study goals. The minimal configurations often prioritise a smaller set of frontal electrodes for practical applications [50], like wearable devices, while more complete studies use higher density configurations, up to 64 channels [44], to improve spatial resolution. Despite these variations, the predominance of the frontal and frontopolar channels in the literature confirms their critical role in the EEG-based stress analysis, balancing the physiological relevance with the practical feasibility.

3.3.2 Public Datasets

The vast majority of the datasets containing raw data (not *features* from the data) used to train the models are private and collected for the specific research. Only the SEED, DEAP and SAM40 datasets are publicly available. Table 3.3 presents the number of subjects, channels, labels (classes), and total duration per subject for these three datasets.

Dataset Name	Subjects, Channels, Duration	Labels
DEAP	32, 32, 40 minutes	Multiclass (Arousal, Valence)
SEED	20, 62, 60 minutes	Multiclass (Positive, Neutral, Negative emotion)
SAM40	40, 32, 6 minutes	Multiclass (see Par. 4.2.2)

Table 3.3: Overview of Key Public Open-Access EEG Stress Datasets.

Between these three datasets, it is important to highlight that the SAM40 dataset is the only one which has a clear distinction between the stress and the relaxation classes. On the contrary, DEAP and SEED present more heterogeneous classes that, even if they are related to emotive or cognitive states, don't always offer a clear distinction between the stress and relaxation classes, making these datasets less suited for research focused on these two states.

3.3.3 Preprocessing Steps

The preprocessing techniques applied in the EEG-based stress analysis are focused mainly on artefact removal and signal standardisation. The most commonly used methods are the Band Pass Filtering (BPF), with frequency ranges varying across the studies (from 0-4 Hz as lower cut-off frequency, to 30-60 Hz as higher cut-off frequency), to eliminate low-frequency drifts and high-frequency noise. Some studies also implement the baseline correction and the Common Average Referencing (CAR) to reduce the variability between channels. The notch filtering (at 50 or 60 Hz, depending on the country) is usually applied to suppress the power line interference. The z-normalisation is often used to have comparable recordings.

Among more complex techniques, the ICA is widely used to remove artefacts, in particular eye movement, while the Principal Component Analysis (PCA) appears less frequently. Some studies apply the Discrete Wavelet Transform (DWT) to decompose the signals in different frequency bands, but this technique is less common than the others.

Overall, the preprocessing choices in the literature reflect a balance between computational efficiency and the need for a robust artefact removal, ensuring that the EEG extracted features remain meaningful for the stress classification.

3.3.4 Accuracy Performances

The performance of classifiers used in EEG-based stress analysis varies across studies. Below, we report the results obtained by different ML and DL models.

Machine Learning Classifiers

- *Support Vector Machine (SVM):* Achieved accuracy rates of 88%, 96%, 94.64%, and 95% in binary classification tasks, and 75% in three-class classification, demonstrating its potential for reliable stress state classification.
- *Sequential Minimal Optimization (SMO):* Reached an accuracy of 97.53%, highlighting its efficiency and effectiveness in handling large datasets.
- *Stochastic Gradient Descent (SGD):* Achieved 96.30%, showcasing the potential of gradient-based optimization methods.
- *Logistic Regression (LR):* Produced accuracies of 98.77% and 84%. While its performance is exceptional in the first case, the drop in the second suggests high sensitivity to feature selection and data quality.
- *Decision Trees:* Achieved a modest accuracy of 84%, indicating that it may not be well-suited for the complexity of EEG-based stress classification.

Deep Learning Classifiers

- *Feedforward Networks:* MLP achieved 92.59% and 94.64% accuracy, while DNN achieved lower results (86.62%), showing that deeper architectures do not always guarantee better classification unless properly optimised.
- *Convolutional Neural Networks (CNNs):* Standard CNNs demonstrated a wide range of accuracy values: 64.2%, 96%, and 97.61% in binary classification, and 90.46% in a three-class setting. The VGGish-CNN model outperformed all, achieving 99.25% accuracy.
- *Long Short-Term Memory Networks (LSTMs):* Performance varied significantly across studies, with accuracy values of 86% and 70.67%, indicating that sequential modelling alone may not always be optimal for EEG-based stress detection.
- *Symmetrical Deep Convolutional Adversarial Network (SDCANs):* Reached only 60.52% (four-class) and 48.17% (five-class) accuracy.
- *Hybrid Architectures:* Many studies developed hybrid architectures, leading to some of the best results. BLSTM-LSTM reached 82.57% to classify stress and 86.33% to classify relaxation. CNN-LSTM, which integrates convolutional feature extraction with temporal modelling, reached 96.70% and 97.8%

accuracy in binary classification, maintaining above 91% for a four-class problem. CNN-BLSTM achieved 99.2%, nearly matching VGGish-CNN, suggesting that combining CNN feature extraction with bidirectional recurrence is highly effective for stress classification.

3.3.5 Processing Time

The vast majority of the articles in the literature reviewed do not explicitly report the duration of the classification algorithm, since the main goal is usually to maximise the accuracy, even by reducing the temporal efficiency of the algorithm. For instance, in the article [46], the classification process alone required 184 seconds, reaching an accuracy of 99.25%. This result is reached through a complex NN, based on the VGGish model, that extracts 4096 features from each segment, which are then elaborated by a CNN. However, it is not specified the time requested for each of the 25-millisecond-long segments for the classification. Instead, in the article [49], an LSTM model is used, achieving an accuracy of 97.8%. Also, in this case, the complexity of the architecture leads to a processing time of 12.5 seconds for each input. Added to this, a significant preprocessing time, due to complex techniques such as the azimuthal projection. The most balanced approach between accuracy and computational efficiency is presented in [13]. In this article, each segment is elaborated in 327.1 ms, of which only 7.1 ms are used for feature extraction and classification. The other 320 milliseconds are used for preprocessing. These results highlight a critical point in the field of EEG-based stress analysis: temporal efficiency, often overlooked to achieve higher accuracies, represents a fundamental aspect for real-time applications, where each millisecond can make a difference.

3.3.6 Limitations of Current Research

Despite the promising results reported in the literature, multiple limitations do not allow a diffused application and the generalisability of these EEG-based systems:

- *Limited Real-Time Implementations*: while the research demonstrates a really high accuracy in offline analysis, almost no article aims to have a fast classification algorithm. The vast majority of the presented models are really efficient, but also very computationally expensive, making the real-time implementation an insurmountable challenge. In addition, many articles are not completely automatic, having manual techniques such as ICA for artefact removal.

- *Scarcity of Public Datasets*: as mentioned before, the vast majority of the datasets used are not publicly available. This implies that it is difficult to replicate and validate the proposed models, or to compare different models on the same dataset. This means that the accuracies reported in the articles might not be directly comparable or representative of real-world performance.

- *Testing Datasets*: all the models' performances have been evaluated on the same dataset used in the training. Other datasets with different characteristics

(SNR, number of channels, and others) might not perform as good as the original dataset. The lack of an external validation on an independent dataset represents a significant limit for the model's generalisability.

In conclusion, even if the progress in EEG stress analysis is notable, to be able to pass from a research environment to a real application, it is essential to face the challenges related to real-time implementation and robustness of the models on unseen data. Overcoming these limitations will open the way to more reliable and widely usable stress monitoring systems.

Paper Name	Number of Subjects, Channels	Measurements Duration	Protocol	Preprocessing	Classifier used	Accuracy	Key findings
EEG based Stress Level Identification [43]	10,14 (AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4)	8 min 20 s	30 s (Stroop o Math, randomly) 20 s Rest 10 times	BPF (0.16-43) Hz Baseline Correction	SVM	88% Stroop 96% Math 75% *(3)	Rest: P_α is higher Stress: P_β is higher Math $\rightarrow$ higher stress
Human stress classification using EEG signals in response to music tracks [50]	27, 4 (AF7,AF8,TP9,TP10)	6 min	3 min baseline 3 x 1 min songs	Notch filter (45-64) Hz	SMO SGD LR MLP	97.53% 96.30% 98.77% 92.59%	English music reduces stress more than Urdu music
EEG-based mental workload estimation using deep BLSTM-LSTM network and evolutionary algorithm [52]	48, 14 (AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4)	-	Simultaneous Capacity-based multitasking activity No task	PBF (4-32) Hz	BLSTM + LSTM	82.57% (S) 86.33% (R)	The proposed model classifies better than RF and SVM
Detection of Mental Stress through EEG Signal in VR Environment [25]	28, 32 (Fp1, Fp2, F7, F3, Fz, F4, F8, FC5, FC1, FC2, FC6, T7, C3, Cz, C4, T8, CP5, CP1, CP2, CP6, P7, P3, Pz, P4, P8, PO3, POz, PO4, O1, Oz, O2, AF7)	18 min	2 min (Stroop) 4 min (Relax) 3 times	BPF (0.2-45) Hz ICA (Picard method) PCA	SVM MLP DNN	94.64% 94.64% 86.62%	Best results using all brain waves
Performance Evaluation of EEG-based Mental Stress Assessment Approaches for Wearable Devices [13]	22, 19 (Fp1, Fp2, F3, F4, F7, F8,C3, C4, T3, T4, T5, T6, P3,P4, O1, O2, Fz, Cz, and Pz)	-	Math tasks Rest	BPF (4-30) Hz	SVM DT LR CNN	95% 84% 84% 96%	CNN is better in accuracy and computational time
Cognitive Stress Recognition during Mathematical Task and EEG changes following Audio-Visual Stimuli for Relaxation [12]	- , 8	-	Math tasks Funny videos	Parks-McClellan optimal FIR filter (2-40) Hz to remove artifacts	LSTM	86%	A happy state increases α Power
Mobile EEG-Based Workers' Stress Recognition by Applying Deep Neural Network [53]	10, 14	-	Low stress (simple tasks) High stress (risky tasks)	BPF (0.5-60) Hz Notch filter ICA	CNN FCDNN	64.20% 86.62%	The best NN has 2 hidden layers with: - 83 neurons - 23 neurons

Table 3.4: Literature Review (1/2); * indicates multiclass classification (number of class in brackets).

Paper Name	Number of Subjects, Channels	Measurements Duration	Protocol	Preprocessing	Classifier used	Accuracy	Key findings
Symmetric Convolutional and Adversarial Neural Network Enables Improved Mental Stress Classification From EEG [45]	21, 32 (Fp1, Fp2, F7, F3, Fz, F4, F8, FC5, FC1, FC2, FC6, T7, C3, Cz, C4, T8, CP5, CP1, CP2, CP6, P7, P3, Pz, P4, P8, PO3, POz, PO4, O1, Oz, O2, AF7)	25 min	Rest 5 min no Stress 10 min Pre Stress 5 min Stress Period 5 min Math	BPF (0.5-40) Hz CAR filter ICA	SDCAN	60.52% *(4) 48.17% *(5)	It is possible to classify stress in multiple stages
A novel technique for stress detection from EEG signal using hybrid deep learning model [47]	39, 19 (Fp1, Fp2,F3, F4, F7, F8, Fz,C3, C4, Cz,P3, P4, Pz,T3,T4, T5, T6,O1, O2)	4 min 4 s	62 s Subtraction tasks 182 s Rest	Discrete WT	CNN-LSTM CNN-BLSTM	96.70% 99.2%	CNN: automatic FS BLSTM: perfect for info-extraction → CNN-BLSTM has high accuracy
EEG-based stress identification and classification using deep Learning [51]	14, 8 (Fp1, Fp2, Fpz, Fz, F3, F7, F8, and F4)	40 min	15 min Untimed Math Test 10 min Relax 15 min Timed Math Test	LPF (0-31) Hz ICA	LSTM CNN	70.67% *(3) 90.46% *(3)	Time-limitation increases stress levels
StressNet: Hybrid model of LSTM and CNN for stress detection from electroencephalogram signal (EEG) [49]	SEED: 20, 62 DEAP: 32, 32	SEED: 50 min DEAP: 40 min	SEED: 2-4 min movie + 30 s eval DEAP: 40 x 1 min music videos	Azimuthal Projection Technique	LSDM + 2D CNN	97.8%	Method valid on different datasets
StressDetect: A Deep Learning Approach for Mental Stress Detection Using Time-Frequency Representation of EEG Signals [27]	40, 32 (Fp1, Fp2, F7, F3, Fz, F4, F8, FC5, FC1, FC2, FC6, T7, C3, Cz, C4, T8, CP5, CP1, CP2, CP6, P7, P3, Pz, P4, P8, PO3, POz, PO4, O1, Oz, O2, AF7)	6 min	[25 s Relax (25 s Stress + 5 s Rest) 3 times] 3 times	Transformation of EEG signal in a TF 2D image	TF + CNN	97.61%	TF+CNN performs better than previous NNs, especially using random dataset
StreXNet: A Novel End-to-End Deep-Learning-Based Improved Multilevel Mental Stress Classification From EEG Sensors [44]	18, 64	8 min 40 s	60 s Eyes Closed 60 s Eyes Open (60 s MAT + 40 s Rest) 4 times	BPF (0.1-45) Hz Artifact Subspace Reconstruction ICA z-normalization	CNN + LSTM + Extreme Gradient Boosting + Squeeze Excitation	91.60% (2) 91.81% *(4)	Effective and Robust model for multi-class classification
Stress detection based EEG under varying cognitive tasks using convolution neural network [46]	40, 32 (Fp1, Fp2, F7, F3, Fz, F4, F8, FC5, FC1, FC2, FC6, T7, C3, Cz, C4, T8, CP5, CP1, CP2, CP6, P7, P3, Pz, P4, P8, PO3, POz, PO4, O1, Oz, O2, AF7)	6 min	[25 s Relax (25 s Stress + 5 s Rest) 3 times] 3 times	Normalization 1-channel reduction Segmentation Conv. Audio Signal Mal-scaled FB	VGGish-CNN	99.25%	High performances reached

Table 3.5: Literature Review (2/2); * indicates multiclass classification (number of class in brackets).

Chapter 4

Materials and Methods

This chapter provides a detailed description of the materials and methods used in this research. It begins by describing the dataset used in the training, validation and testing of the proposed classifier, including the tasks designed to induce stress in the subjects. Subsequently, it will explain in detail the preprocessing techniques used, as well as the data augmentation strategy and the final steps to prepare the data for the classifier. The chapter presents then the architecture of the chosen net, going into detail about its structure and logic. In the end, it will treat the characteristics of the new data collected, describing in detail the devices and software used in the experimental protocol defined and the interfaces used to induce stress.

4.1 Computational Setup

All data analyses and model training were conducted on a computer equipped with a *Windows 11 Education N (version 24H2)* operating system, running on an *Intel Core i5-7600K CPU @ 3.80 GHz* with a *64-bit architecture*. The system was configured with *32 GB of RAM*, ensuring sufficient memory for handling large EEG datasets and DL computations.

For GPU-accelerated computations, the system utilised an *NVIDIA GeForce GTX 1080*, which provided substantial processing power for training CNNs and fast data classification. The torch.cuda.is_available() function confirmed GPU usage during program execution.

The DL framework used was `PyTorch`, running on *Python 3.10.16* within a *virtual environment* managed by *Anaconda*. This setup ensured flexibility in package management and reproducibility of the computational experiments.

The combination of high-speed SSD storage, a dedicated GPU, and a multi-core processor allowed efficient execution of DL models for the EEG-based stress classification.

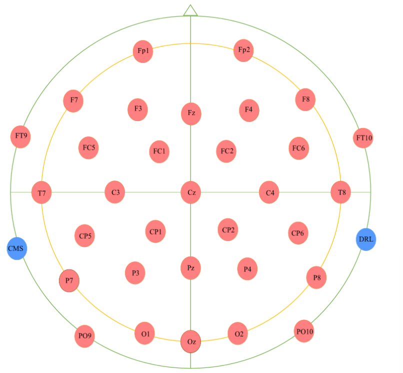

Figure 4.1: Placement of the 32 electrodes for the SAM40 data acquisition, following the 10-10 international system; image from [2].

4.2 Dataset

The SAM40 Dataset [2] has been used for this study. This EEG dataset, publicly available, contains recordings from 40 healthy subjects (14 females, 26 males) aged 18-25 years (mean age: 21.5 years). The dataset has been specially designed to monitor the short-term stress, induced through simple tasks. Below are the key characteristics of the dataset.

4.2.1 Data Acquisition

The EEG signals have been recorded using an Emotiv Epoc Flex gel-based system with 32 channels (Emotiv Inc., San Francisco, USA) with CMS/DRL references. Data has been sampled at 128 Hz, with a dynamic range of ±4.12 mV and a resolution of 0.51 µV/bit. Electrodes have been positioned according to the 10-10 international system, covering the frontal, central, parietal, temporal and occipital regions. Visual stimuli have been shown on a 24-inch monitor, placed 70 cm from the participants.

The channels considered for recording the brain activity are presented in Fig. 4.1, and were: C_z, F_z, Fp_1, F_7, F_3, FC_1, C_3, FC_5, FT_9, T_7, CP_5, CP_1, P_3, P_7, PO_9, O_1, P_z, O_z, O_2, PO_{10}, P_8, P_4, CP_2, CP_6, T_8, FT_{10}, FC_6, C_4, FC_2, F_4, F_8, Fp_2.

4.2.2 Experimental Tasks

Each subject performed three trials, each composed of the following sequence of four cognitive tasks, which correspond to the four classes with which the dataset has been labelled:

- *Initial Relaxation:* subjects started each trial with a 25-second relaxation period, during which they were asked to stay still, listening to calming instrumental music.

- *Stroop Colour-Word Test (SCWT):* to induce cognitive interference and stress, the subjects performed a Stroop test (on the left in Fig. 4.2). During this task, they were asked to identify and verbally report the colour of the ink of colour words, which could have been congruent (e.g. "GREEN" printed in green ink) or incongruent (e.g. "GREEN" printed in red ink). The task included 11 different words visualised on the screen for 25 seconds. The task required a fast cognitive elaboration and response selection, contributing to enhancing the cognitive load.

- *Arithmetic Task:* after the SCWT, subjects had to perform 25 seconds of arithmetical calculation (central image in Fig. 4.2). They were presented with a series of mathematical equations, and they had to quickly determine if the equations were correct. Each subject had to complete six arithmetical operations in the 25 seconds. This task was designed to engage working memory and problem-solving ability, increasing cognitive stress levels.

- *Mirror Image Recognition:* in the final tasks, the subjects were asked to evaluate the symmetry of two mirror-reflected images (on the right in Fig. 4.2). A set of eight images was shown sequentially, and the subject had to answer as fast as possible, determining if the image was specular or not. Participants gave their answers using predefined gestures, ensuring minimal movement artefact in the EEG recordings. The task lasted 25 seconds and was designed to further increase the cognitive workload by demanding visual attention and spatial reasoning.

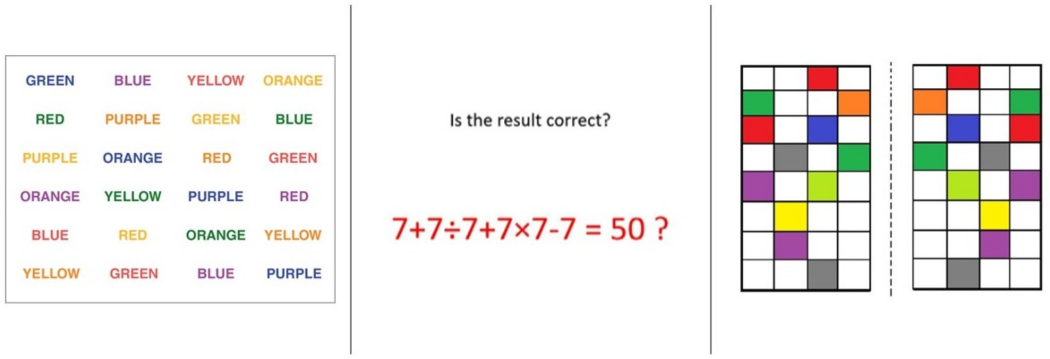

Figure 4.2: Examples of stress-inducing tasks: Stroop Test (left), Arithmetic Task (middle), Mirror-Image Matching Task (right); images from [2].

During the experiment, a structured temporal sequence was implemented to regulate the transition between the different tasks. A five-second relaxation period was inserted between the stressful tasks, enabling the subjects to momentarily recover from the cognitive load of the previous activity. Afterwards, a 10-second time interval was taken before each task to explain and clarify the objective of the following task.

After completing each trial, the subjects were asked to self-evaluate their perceived stress levels for each task, on a 1 to 10 scale, with 1 representing the lowest stress level and 10 the highest. This subjective stress evaluation provided an additional measure of the cognitive load beyond EEG recordings. In any case, since this self-evaluation was a *subjective* measure, it was not used in this study as a criterion to label the data, and it was not further used in the analysis.

To enhance the stress induction in the subject, an experimenter was present to monitor and record the subject's answers, giving real-time feedback on the performance accuracy. This part of the protocol was intended to create an evaluative pressure environment, further contributing to stress induction.

The experimental design ensured that EEG signals captured during the tasks accurately reflected stress-induced neural activity. The SAM40 dataset thus offers a comprehensive resource for studying stress-related EEG patterns under controlled cognitive challenges.

4.3 Signal Preprocessing

Signal preprocessing is a crucial step in the classification task because it removes noise and artefacts from the raw signals. This ensures that the classifier elaborates only the relevant data, improving its performance. The following preprocessing techniques were applied:

1. Z-normalisation:

Z-normalisation standardises a signal by subtracting its mean value and then dividing by its standard deviation. This process yields a signal with zero mean and unit standard deviation, and is mathematically expressed as:

$$z = \frac{x - \mu}{\sigma}, \tag{4.1}$$

where x represents the original signal, μ is the mean, and σ is the standard deviation.

Normalisation is an important step for NNs because it ensures that all the inputs have comparable scales, preventing a single characteristic from dominating the learning process. This improves the stability and the efficiency of the training, and it accelerates the convergence by avoiding issues related to vastly different magnitudes and reducing the risk of vanishing or exploding gradients, particularly in deep architectures. Moreover, z-normalisation is expected to help mitigate the influence of intra-subject variability, ensuring that the model learns features related to the condition (stress or no-stress, in this case) rather than subject-related features. In the work, z-normalisation was applied channel-wise, so that every channel presented a mean of zero and a unit variance.

2. Band-Pass Filtering:

In EEG stress analysis, the frequency band of interest is typically between 4 and 30 Hz (see Sec. 2.1). To focus only on this bandwidth and suppress lower and higher undesired frequencies, a BPF has been applied. A fourth-order Butterworth filter was used to attenuate frequencies lower than 1 Hz and higher than 40 Hz, and its module is shown in the left panel in Fig. 4.3. The lower cut at 1 Hz helps remove the slow drifts and the baseline fluctuations, while the higher cut at 40 Hz helps suppress the high-frequency noise, including muscular and environmental artefacts. Although the EEG information relevant to stress analysis lies in the 4-30 Hz range, the cut-off frequencies have been chosen to ensure a minimal distortion in the bandwidth of interest, preserving signal integrity for further analysis.

3. Notch Filtering:

A notch filter is used to remove a very narrow frequency bandwidth, such as power line interference. Since the data has been collected in India, where the mains frequency is 50 Hz, the notch filter has been set at this frequency. The `iirnotch` function from the `scipy` library was used, with a quality factor of 30, defined as the ratio between the central frequency and the 3 dB bandwidth (on the right in Fig. 4.3). Even if the BPF had already been applied, also filtering the 50 Hz interference, the notch filter was additionally used to ensure a more efficient suppression of power line interference. This decision was made because the magnitude response of the previous BPF, at 50 Hz, was slightly below 10^{-1}, meaning that residual power line interference could still be considered present in the signal. Applying the notch filter ensured a more effective suppression of this noise component without significantly affecting nearby frequency bands.

4. Independent Component Analysis (ICA):

ICA is a computational method which separates a multivariate signal into additive and statistically independent components. It realises this separation by finding a linear transformation which statistically decorrelates the data, maximising the non-Gaussianity of the final components. The assumption on which it works is that the original sources of the mixed signals are statistically independent and non-Gaussian. This technique is particularly useful and efficient for EEG signals, which often contain various artefacts such as ECG, EMG and eye movements. ICA is usually performed after filtering, so that big sources of noise, such as the 50 Hz power line, the EMG or ECG interference, have already been removed from the signals.

The method works by estimating a linear transformation that maximises the statistical independence of the components. Mathematically, given a set of observed mixed signals $x(t) = [x_1(t), x_2(t), \ldots, x_N(t)]^T$, where N is the number of observed channels (e.g., EEG electrodes) and t represents time, ICA models these observations as a linear mixture of M unknown independent source signals s(t):

$$s(t) = [s_1(t), s_2(t), \ldots, s_M(t)]^T$$

$$x(t) = As(t) \tag{4.2}$$

where A is an unknown $N \times M$ mixing matrix. The goal of ICA is to find an unmixing matrix W (which is ideally the inverse of A, i.e., $W = A^{-1}$), such that the estimated source signals $\hat{s}(t)$ are obtained by:

$$\hat{s}(t) = Wx(t) \tag{4.3}$$

The rows of $\hat{s}(t)$ represent the estimated independent components. The process of finding W involves maximising the statistical independence of the components, often by maximising their non-Gaussianity, as a sum of independent non-Gaussian variables tends to be more Gaussian (Central Limit Theorem).

However, it's important to note that no ICA method perfectly removed artefacts from all the datasets on which it has been applied. The approach detailed below has been chosen for its efficacy, but many other techniques exist. For instance, the combination of the WT with ICA offers another way to remove artefacts, although it is computationally expensive, which is why it wasn't applied in this work. Alternatively, a visual inspection and manual removal of the artefactual components allows an almost perfect artefact suppression. This option has been discarded because, in the research, an automatic algorithm was needed.

Many methods have been tested on the dataset, and the specific method employed in the study is described below.

ICA was performed using the `FastICA` function from the `sklearn` library with the following parameters: `n_components=32`, `max_iter=200`, `whiten=unit-variance`. This ensures that all the resulting independent components have zero mean and unit standard deviation. The `FastICA` algorithm was chosen due to its prevalent use and strong performance reported in the literature for EEG artefact removal [55].

To effectively remove the artefactual components, the kurtosis value was calculated for each component. The kurtosis is a statistical measure that quantifies the tailedness of a distribution, indicating whether the distribution is thin or broad [56].

$$\text{Kurtosis}(X) = E\left[\left(\frac{X-\mu}{\sigma}\right)^4\right]$$

This formula calculates the fourth standardised moment of a variable X. Here, $E[\cdot]$ represents the expected value, μ is the mean of the distribution, and σ is its standard deviation. The expression $\left(\frac{X-\mu}{\sigma}\right)$ standardises the variable, transforming it to a scale-independent form, which allows for the comparison of the shape of different distributions regardless of their original scale or location. By raising this standardised variable to the fourth power, the formula gives more weight to extreme deviations from the mean, thereby highlighting the presence of outliers or heavy tails.

High kurtosis indicates a distribution with a narrow peak, while low kurtosis indicates a broader peak. EEG artefacts, especially ocular movements, usually present higher kurtosis values compared to brain-originated signals, due to their transient, high-amplitude characteristics [57]. In the algorithm, a component was automatically marked for elimination if it satisfied both of the following conditions:

- Its kurtosis value was greater than 12. This threshold was determined through a visual inspection of the eye movement artefactual components and their respective kurtosis value in the SAM40 dataset, but it is important to notice that, even if this threshold was effective for the dataset, it might need adjustment if used for other datasets.

- The estimated number of blinks in the component was lower than two per second. This criterion is based on the physiological assumption that a subject in a normal situation blinks less than twice a second. To accurately calculate the blinks and mitigate the high-frequency noise impact, which could lead to a higher detection of peaks, one close to the other, each component was first smoothed using a Low-Pass Filter (LPF) with a cut-off frequency of 15 Hz. After this smoothing, the function `find_peaks` from `scipy` was applied, looking for peaks whose amplitude was higher than half of the maximum value of the component studied. This refinement ensured the counting of the significant peaks alone, which probably corresponded to eye blinks.

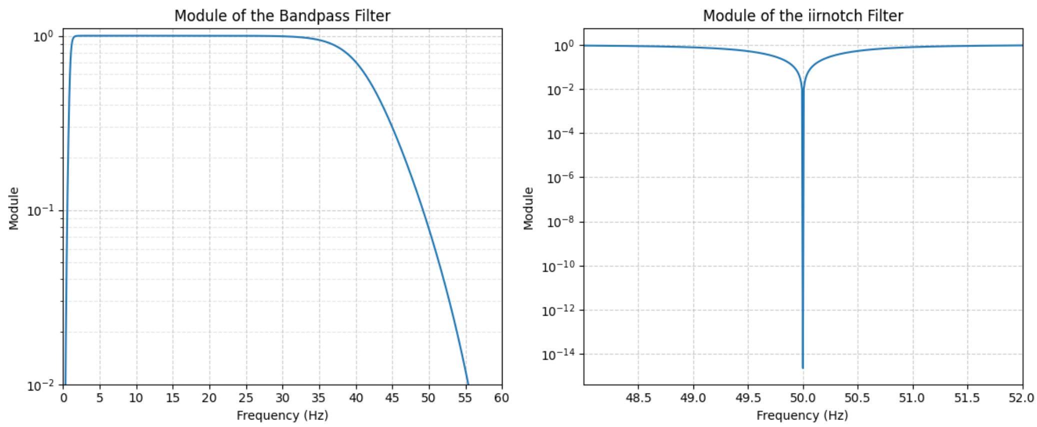

Figure 4.3: Module of the Butterworth BPF (Left) and Notch Filter (Right) used in the analysis.

Fig. 4.4 illustrates the progressive refinement of the EEG signal for subject 5, channel F_7, through the preprocessing steps: raw data, normalisation and filtering, and ICA. In the top panel, the raw EEG trace shows multiple contaminations. Low-frequency drifts are noticeable (for instance, at about 20 seconds), as well as numerous sharp, high-amplitude transients, probably due to ocular artefacts such as eye blinking. In addition, the signal contains high-frequency noise, potentially coming from muscular activity (EMG) or other sources.

The middle panel shows the EEG signal after the normalisation and filtering. These steps effectively attenuate the slow drifts and suppress part of the high-frequency noise. However, some artefacts, in particular the transient peaks associated with eye movement, remain accentuated. This confirms the fact that basic filtering is not sufficient to eliminate non-stationary artefacts which are not strictly frequency-localised.

The bottom panel shows the EEG signal after the application of ICA. Here, the artefacts that were dominant in the first two panels, especially the sharp peaks, are

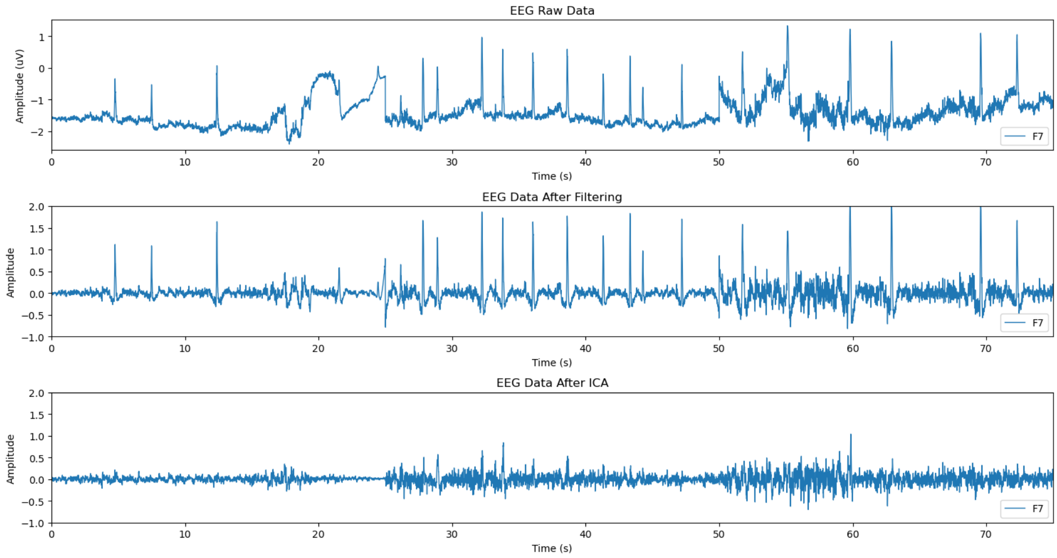

Figure 4.4: Signal comparison for subject 5, channel F_7. Top panel: raw signal. Middle panel: after filtering. Bottom panel: after ICA.

significantly reduced. ICA isolates and removes successfully the components related to non-neural sources, exploiting their statistical independence from genuine brain activity. As a result, the signal appears cleaner and more physiologically plausible, revealing subtle fluctuations that are likely to reflect cortical processes.

Overall, this comparison highlights the complementary roles of filtering and ICA in EEG preprocessing. While filtering aims to remove noise from specific bandwidths, ICA excels in identifying and removing artefactual non-neural components. Together, they improve the signal-to-noise ratio (SNR) and ensure the preservation of meaningful electrophysiological schemes, which is crucial for a reliable classification.

4.4 Data Augmentation

In a preliminary phase of the model development, a data augmentation strategy was applied to compensate for the limited data available. The chosen technique is called *Linear Surrogating*. This approach generates new signals, keeping the original amplitude spectrum while introducing random phase information. The idea is to create new signals which have the same fundamental spectral properties of the original signals, but with a different temporal structure, potentially useful to help the generalisation of the NN.

The procedure is as follows:

1. *Compute the FFT* of the original signal to convert it into the frequency domain.

2. *Retain the magnitude* spectrum of the signal.

3. Generate a new set of *phase values randomly*, typically from a uniform distribution.

4. *Combine the original magnitude with the newly generated phase values* to form a modified frequency-domain representation.

5. Apply the *inverse FFT (IFFT)* to transform the modified signal back into the time domain.

The algorithm was applied only on the training and validation sets, without modifying the test set, guaranteeing in this way its independence, and better simulating a real application. By applying this procedure, the number of signals in the training and validation sets was doubled. The new signals kept the same characteristics in amplitude as the original ones, but with different phase properties.

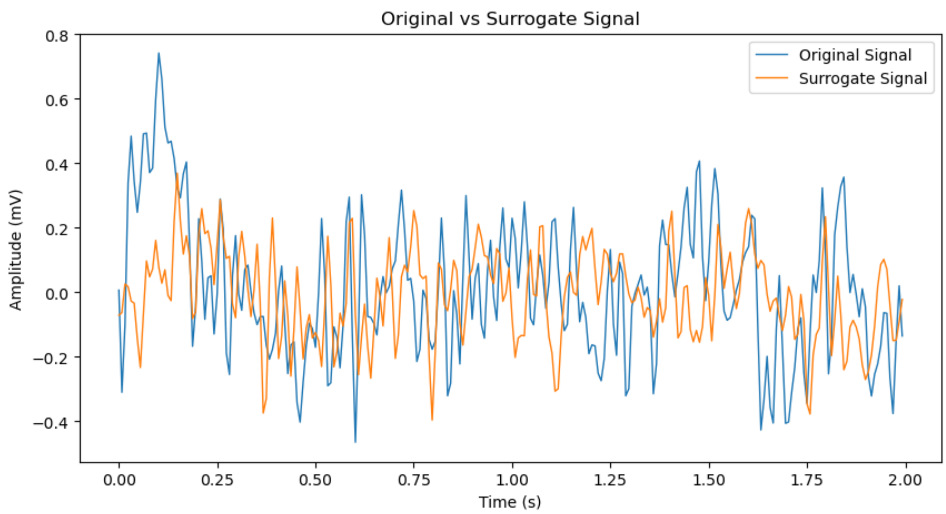

Figure 4.5: Original signal (blue) and its linear surrogate (orange).

Fig. 4.5 illustrates an example of the original signal (blue) and its linear surrogate (orange). At first, both signals show comparable amplitude envelopes, confirming that the magnitude spectrum is preserved during the augmentation process. However, the surrogate signal shows clear, distinct temporal characteristics due to the randomised phase components.

The experimental results highlighted a drop in the accuracy on the test set, while the training and validation sets kept performing well. For this reason, it was decided not to use this technique, but only to use the non-augmented data for the training of the model.

The observed phenomenon, called *overfitting*, can be explained by the fact that, even by introducing some variability in the signal phase, the magnitude of the signal is not changed. Since the classifier used is designed to capture the spectral characteristics of the signal, especially the ones related to the power in certain frequency bands, the surrogates are excessively similar to the original ones, in the domain in which the model is learning. In this way, it recognises very well the patterns in the augmented training, but struggles to generalise to new signals, such as the ones present in the test set.

4.5 Data Preparation

The input to the NN consists of segments of the signal in the time domain. Fig. 4.6 and the following paragraph both describe the steps required to go from the original 75-second-long signals to the 2-second-long subsegments used as input for the classifier.

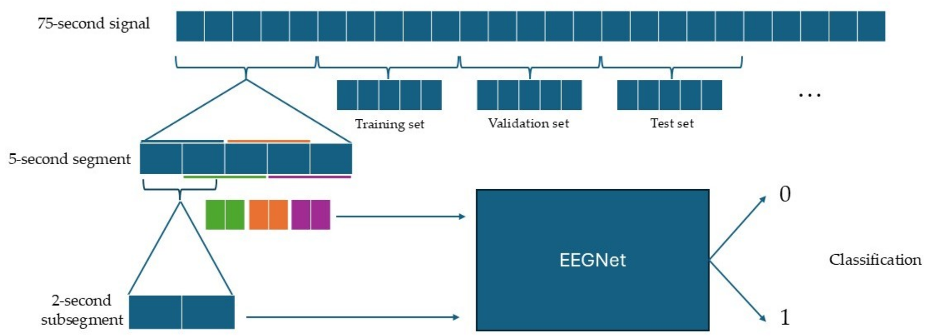

Figure 4.6: Data preparation steps from the 75-second signal to the 2-second subsignals used as input for the classifier.

The recording of each subject, of a duration of 75 seconds, was initially divided into 15 subsignals without overlap, each with a duration of 5 seconds. These subsignals were then split into a construction (training and validation) set and a test set, ensuring that both contained segments from all the subjects, but no signal was repeated, not even a single overlapped part, in the three sets. Then, these 5-second subsignals were further divided into smaller 2-second segments of 2 seconds (256 timeframes). To increase the number of samples in the construction set, an overlap of 50 % was applied during the segmentation. No overlap was applied in the test set to maintain independent samples. In the end, the 2-second-long segments were converted into PyTorch tensors and loaded into `DataLoader` objects with a batch size of 32 for efficient training.

The algorithm steps, from preprocessing to classification, are presented in Fig. 4.7.

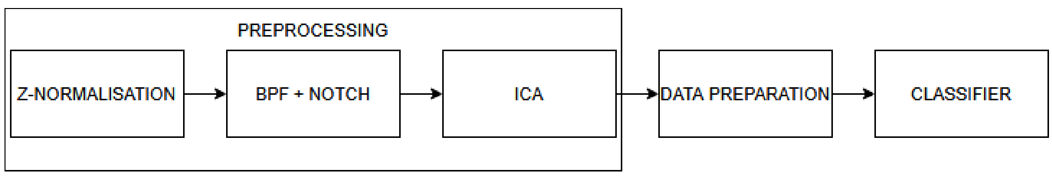

Figure 4.7: Flowchart with the steps of the algorithm from the preprocessing to the classifier.

Considering that the goal of this work was the binary classification of EEG data into *stress* or *relaxation* states, and to prevent the problem of unbalanced classes, the analysis was conducted exclusively using the Stroop Test as stress data. For a more comprehensive classification, it could have been possible to consider all four original classes (Relax, Stroop, Arithmetic, Symmetric). However, this in-depth analysis goes beyond the main goal of the thesis, which is focused on basic stress detection.

4.6 Architecture Used

Recent studies demonstrated that CNNs are extremely efficient in EEG-based classification tasks, due to their capacity to extract complex temporal and spatial features present in EEG signals [27, 44, 49, 51]. In this work, a variant of *EEGNet* [1] has been adapted for the classification task, for fast stress detection. EEGNet is a CNN structured as a sequence of three main convolutional blocks, followed by a fully connected classification layer. Each block is designed to extract and progressively refine the features, granting at the same time stable training dynamics and preventing overfitting.

Layer 1: Temporal Convolution and Basic Regularisation. The first layer starts with a *Conv2D* with a kernel dimension of (1, 64), which targets temporal filtering by convolving the segment only in the temporal dimension. This design allows the model to capture fundamental temporal patterns before any spatial elaboration. After this, a *Batch Normalisation* layer follows, which stabilises the learning process by normalising the activation distributions. This not only accelerates convergence, but also the impact of the internal covariate shifts, indirectly contributing to regularisation.

To introduce nonlinearity, an *ELU* (Exponential Linear Unit) activation function is applied. ELU is defined as:

$$\mathrm{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases} \quad ; \quad \mathrm{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \tag{4.4}$$

with $\alpha > 0$ (in this work, $\alpha = 1$). Unlike the classical ReLU function, which sets to zero all the negative values, and might cause the *dying neuron* problem (meaning that some neurons get stuck, with zero gradients), the ELU function allows small negative outputs with smooth gradients. This reduces the bias shift, supporting a more stable and efficient training. The comparison between the two activation functions is shown in Fig. 4.8.

Finally, a *dropout* layer with a rate of 25% is used to randomly deactivate neurons during training, encouraging the network to learn more redundant and robust features.

Layer 2: Spatial Filtering and Dimensionality Reduction. The second block begins with a *ZeroPadding* operation, which pads the input with 16 pixels on the left, 17 on the right, and 1 on the bottom. This padding ensures that the output dimensions will be compatible with the next convolutional operation. A *Conv2D layer* with a kernel (2, 32) then performs a spatial filtering. This level is designed to extract spatial features by scanning both channels and temporal segments. The

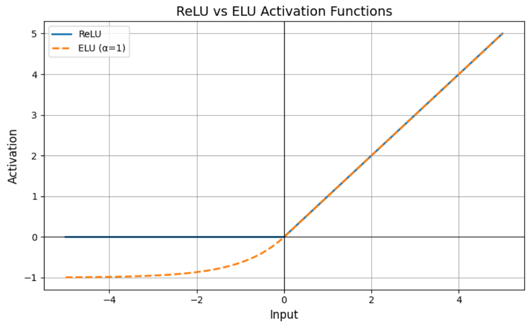

Figure 4.8: Comparison of ReLU (blue) and ELU (orange) activation functions. ELU smoothly continues into negative values, while ReLU is strictly zero for negative inputs.

extracted features are normalised with a *Batch Normalisation*, and activated with the *ELU* function. Then a new *Dropout* layer is applied to further prevent overfitting. In the end, an Average Polling is performed with a kernel dimension of (2, 4), reducing the spatial dimensions. This pooling not only reduces the computational load for the following layers, but it also emphasises the most salient features, by aggregating local averages, which can be particularly useful when little variations in EEG data are indicative of stress.

Layer 3: Depthwise Separable Convolutions and Enhanced Feature Extraction. The third convolutional block introduces another *ZeroPadding* with the configuration (2, 1, 4, 3) to preserve the spatial dimensions needed for further convolutions. A *depthwise convolution* is then applied. This is a convolution with the number of groups equal to the number of channel inputs, meaning that each input channel is convolved with its own filter, and so there are no connections between different input channels in this layer. This operation works as a spatial filter applied independently to each feature map, allowing the net to learn channel-specific patterns with a low computational complexity. After the depthwise convolution, a *pointwise convolution* (1×1 convolution) is applied. This merges the information across channels by linearly combining the outputs from the depthwise filters. This 2-phase strategy (depthwise followed by pointwise convolution) is called *depthwise separable convolution* and is efficient in reducing the number of parameters while still capturing rich features [58, 59, 60]. As in previous layers, *Batch Normalisation*, *ELU activation*, and *dropout* are applied to maintain numerical stability, incorporate nonlinearity, and prevent overfitting. An additional *average pooling* operation with a kernel size of $(2, 4)$ further extracts the features and compresses the final feature map dimensions before classification.

Fully Connected Layer. After the convolutional blocks, each sample has a $4 \times 8 \times 12$ feature map. These multidimensional feature maps are then flattened into a 384-value unidimensional vector through a fully connected layer. This maps the extracted features to two output neurons corresponding to the *stress* and *non-*

stress classes. Finally, a *softmax* activation function converts the logits into class probabilities, thus enabling probabilistic interpretation of the results.

Parameter Summary:
For clarity and reproducibility, detailed layer parameters (such as specific padding values and kernel sizes) are summarised in Tab. 4.1.

Block	Layer	Out #filters	Out shape (C, H, W)
1	Input	—	(1, 32, 256)
	Conv2D (1 $\rightarrow$ 16, kern=(1, 64))	16	(16, 32, 193)
	BatchNorm	16	(16, 32, 193)
	ELU	16	(16, 32, 193)
	Dropout (p=0.25)	16	(16, 32, 193)
2	ZeroPad2d (l=16, r=17, t=0, b=1)	—	(16, 33, 226)
	DepthConv (16 $\rightarrow$ 4, kern=(2, 32))	4	(4, 32, 195)
	BatchNorm	4	(4, 32, 195)
	ELU	4	(4, 32, 195)
	Dropout (p=0.25)	4	(4, 32, 195)
	AvgPool2d (kern=(2, 4))	4	(4, 16, 48)
3	ZeroPad2d (l=2, r=1, t=4, b=3)	—	(4, 23, 51)
	DepthwiseConv (4 $\rightarrow$ 4, kern=(8, 4), groups=4)	4	(4, 16, 48)
	PointwiseConv (4 $\rightarrow$ 4, kern=1)	4	(4, 16, 48)
	BatchNorm	4	(4, 16, 48)
	ELU	4	(4, 16, 48)
	Dropout (p=0.25)	4	(4, 16, 48)
	AvgPool2d (kern=(2, 4))	4	(4, 8, 12)
	Flatten	—	(384)
Classifier	Linear (384 $\rightarrow$ 2)	2	(2)
	Softmax (dim=1)	2	(2)

Table 4.1: EEGNet architecture Parameters Summary.

4.7 Training Hyperparameters

Table 4.2 summarises the key training hyperparameters and their values.

Parameter	Value
Batch size	32
Number of epochs	101
Early stopping patience	20 epochs
Loss function	Cross Entropy Loss
Optimizer	Adam ($\eta = 0.005$, $(\beta_1, \beta_2) = (0.9, 0.999)$, weight_decay $= 10^{-4}$)
Scheduler	CosineAnnealingLR

Table 4.2: Training hyperparameters and their values.

The classifier was trained for a maximum of 101 iterations with a *batch size* of 32. Each epoch includes one full forward and backwards pass through all training examples, and the model's generalisation was monitored through the validation set.

To prevent overfitting and reduce unnecessary computation, *early stopping* was applied: training automatically concluded if the validation accuracy did not improve for 20 consecutive epochs, and the model weights corresponding to the highest validation accuracy achieved at that epoch were saved.

The loss function used is the *Cross Entropy Loss*, defined for binary logits $\mathbf{z} \in \mathbb{R}^2$ and true label $y \in \{0, 1\}$ as:

$$\mathcal{L}_{\mathrm{CE}}(\mathbf{z}, y) = -\log\left(\frac{\exp(z_y)}{\sum_{j=1}^{2} \exp(z_j)}\right), \tag{4.5}$$

This loss function penalises the misclassification, encouraging confident probabilistic outputs, making it ideal for softmax-based classifiers.

Weight updates were performed using the *Adam* optimiser, selected for its robust, adaptive learning rate properties and its wide usage in the literature, with hyperparameters tuned to further reduce overfitting and control update variance:

- Learning rate: $\eta = 0.005$; this controls the step size used in updating the model parameters during gradient descent. A smaller η means slower but more stable convergence.
- Weight decay (L2): $\lambda = 1 \times 10^{-4}$; this term is added to the loss function to penalise large weights. It helps reduce overfitting and improve the model's generalisation ability by discouraging overly complex solutions.
- Momentum parameters: $\beta_1 = 0.9,\ \beta_2 = 0.999$; these are exponential decay rates used in optimisers like Adam. β_1 controls the decay rate of the moving average of the first moment (mean of gradients), and β_2 controls the decay rate of the second moment (uncentered variance of gradients), helping to stabilise and accelerate convergence.

To improve convergence and to have an automatic learning rate tuning, a *cosine annealing scheduler* was employed:

$$\eta_t = \tfrac{1}{2}\,\eta_0\left(1 + \cos(\tfrac{t\pi}{T_{\max}})\right), \tag{4.6}$$

where t is the current epoch and $T_{\max} = 101$. This schedule smoothly decays the learning rate from η_0 to zero, encouraging gradual exploration of the minimum value of the loss function, without abrupt drops.

4.8 Data Collection

To evaluate the classifier and its ability to generalise across different individuals, a new dataset was collected. To avoid any potential ethical concerns, all data were acquired from a single subject, the author. From now on, this individual will be referred to in general terms as the "subject of study".

4.8.0.1 EEG cap and Electrodes

To collect data, an OpenBCI device was used. This EEG cap had 16 channels (while the SAM40 dataset used 32), positioned according to the 10-20 system. The electrodes present on the cap were dry, and they included: Fp_1, Fp_2, F_3, F_4, F_z, C_3, C_4, C_z, P_3, P_4, P_z, O_1, O_2, T_3, T_5, T_6. The reference electrode was T_4. In Fig. 4.9 it is shown a scheme of the electrode positions for the EEG cap used. In green, the 16 differential channels, while in red is the reference electrode.

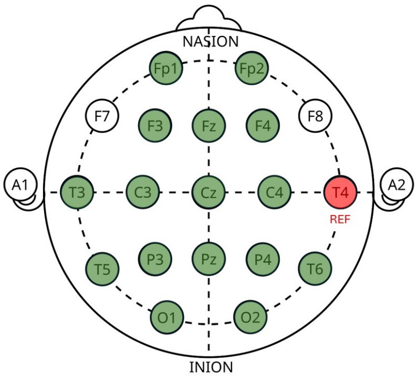

Figure 4.9: Electrodes positioning in the EEG cap used for the data acquisition. In green, the 16 recording electrodes, in red the reference electrode, in white the other electrodes from the 10-20 international system, which have not been used in this data collection.

4.8.0.2 Bioamplifier

The data acquisition system is shown in Fig. 4.10 and consists of a 32-bit Cyton OpenBCI board, coupled with an OpenBCI Daisy module. This combined configuration allows for the acquisition of 16 differential channels of EEG signals. At the core of the amplification system, there are two *Texas Instruments ADS1299* Analog-Digital Converters (ADC): one for the Cyton board, and one for the Daisy module, which guarantee a 24-bit resolution for each channel. Each channel supports both active and passive electrodes. The Cyton board operates with a 3.3V digital operating voltage and a ±2.5V analog operating voltage, accepting an input voltage range of 3.3-12V. Similarly, the Daisy module has a 3.3V digital operating voltage and a ±2.5V analog operating voltage. The sampling frequency is 125 Hz (while the SAM40 dataset was sampled at 128 Hz), and the wireless communication with the computer is achieved through an OpenBCI USB dongle, utilising RDFuino radio modules, or Bluetooth Low Energy for mobile devices.

4.8.0.3 Software used

For the data acquisition, two main software programs were used:

1. *OpenBCI GUI*: this user interface is presented in Fig. 4.11, and it visualises in real-time the signals from the 16 channels, calculating for each of them the

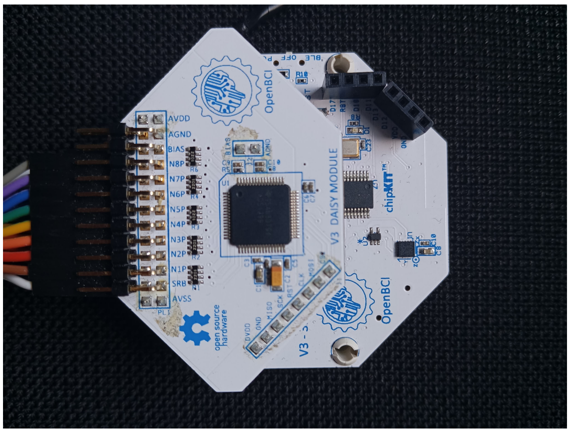

Figure 4.10: OpenBCI Cyton board coupled with the Daisy module, which together form the bioamplifier system used for EEG data acquisition.

percentage of "railing", which indicates the proportion of the signal that has exceeded the maximum or minimum measurable voltage range of the acquisition system. A value of railing inferior to 75% typically indicates a good channel recording. In Fig. 4.11, channel 15 (T_6, in the 10-20 system) exemplifies a railed signal: with a railing percentage of 81.94%, it is labelled as *Near railed* and highlighted in orange to alert the user. This indicates that a large portion of the signal is saturating the input range, and the electrode may need to be repositioned to restore signal quality. The interface also shows the FFT of each channel, allowing for a visual spectral analysis. In the plot, the signals are already filtered, as can be seen by the minimal amplitude at 50 Hz. A significant interference at 25 Hz is visible, probably due to the 50 Hz interference. In the end, the User Interface allows for a connection through an LSL (Lab Streaming Layer) Stream to external software for data recording. Although it was possible to transmit 3 streams at the same time, only the raw data was sent to the LabRecorder in the data acquisition. This means that all the interference that was absent in the FFT plot in the GUI was still present in the acquired data, and will still need a data preprocessing strategy before the classification.

2. *LabRecorder*: this software is responsible for receiving the data transmitted via LSL, and saving it in the `.xdf` format in the directory specified by the user. LabRecorder facilitates the organisation of the data acquired, allowing for the automatic creation of a hierarchy of subfolders based on parameters such as subject, session, type of measurement (e.g., EEG), and the progressive number of execution.

4.8.0.4 Protocol Definition

The protocol used for the data is a replica of the one used to collect the data for the SAM40 dataset, which consists of four sequential tasks: 25 seconds of relaxation, 25 seconds of SCWT, 25 seconds of Symmetrical Images, and 25 seconds of Arithmetical

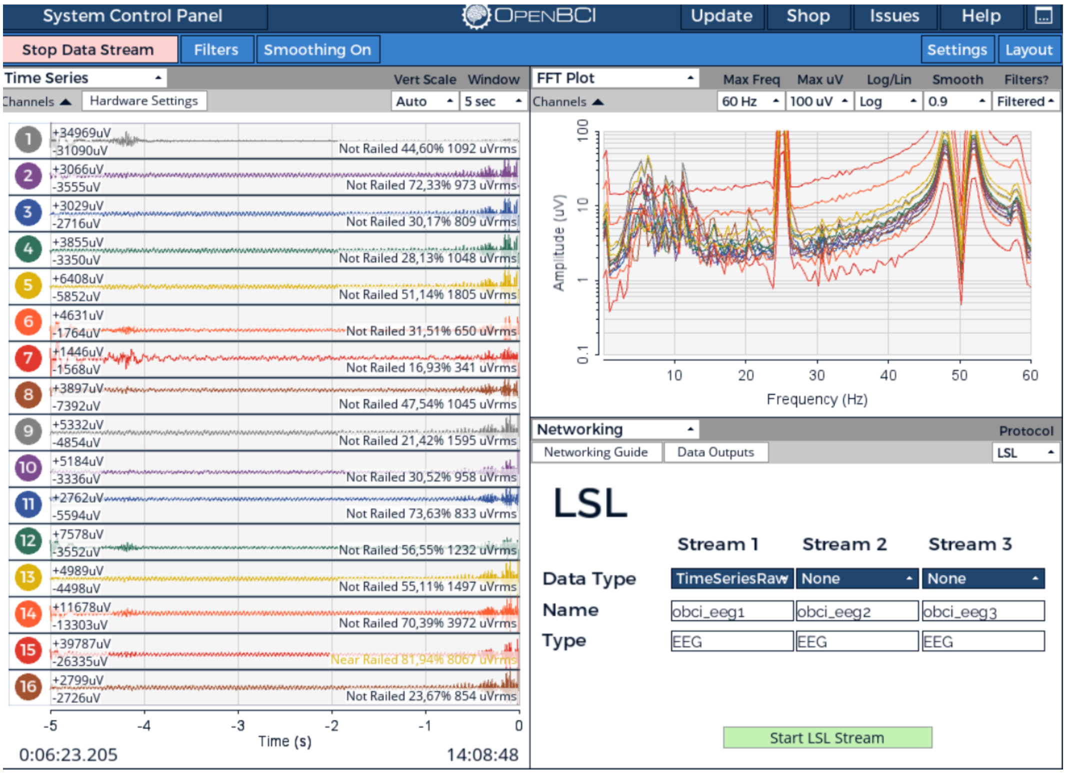

Figure 4.11: OpenBCI GUI, showing the time-signals from the 16 channels (left), the FFT of the 16 channels (top-right), and the networking block (bottom-right), which allows sending the data to the LabRecorder.

Tasks. These tasks have already been explained in Sec. 4.2.2. The subject of study repeated this protocol across five different sessions, resulting in a total of 50 recordings. Fig. 4.12 shows the subject during an arithmetical task. The EEG cap is connected through the cables to the bioamplifier, which is on the desk. The amplifier streams the data through Bluetooth to the USB dongle connected to the laptop, which will collect and save it.

The tasks were explained to the subject, and subsequently, he was presented with the screen showing the interfaces. These were demonstrated using an automated Python script. An example of the 4 tasks is shown in Fig. 4.13.

4.8.0.5 New Data Processing

The new dataset was preprocessed, trained and tested in the same way as it was done for the SAM40 dataset. The data was z-normalised, filtered with a BPF (cut-off frequencies of 1 and 40 Hz) and a notch, and then ICA was performed. After this, the data was divided into 2-second-long subsegments, as explained before (see Sec. 4.5), and tested on the EEGNet.

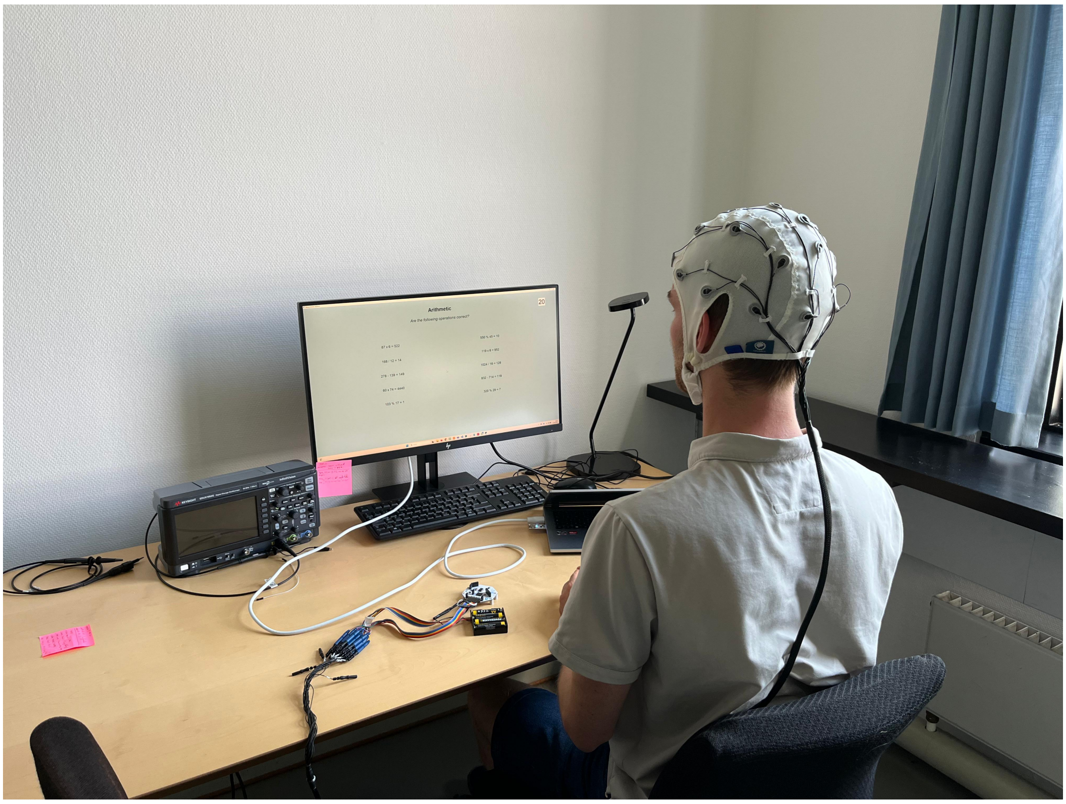

Figure 4.12: Experimental Setup. The subject is positioned in front of the screen with the interfaces. The EEG cap is connected via cables to the amplifier, which is connected via Bluetooth to the USB dongle in the laptop.

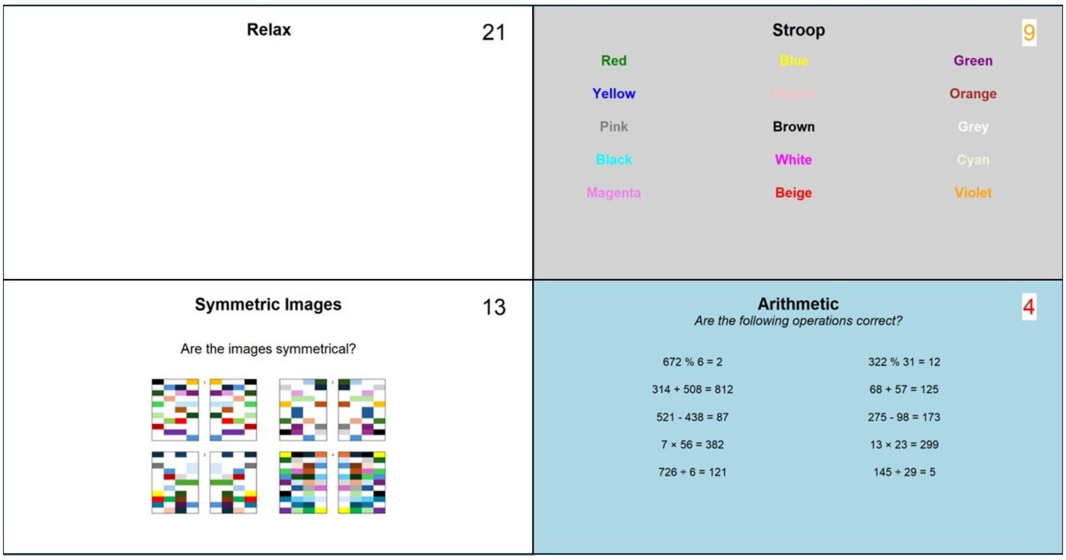

Figure 4.13: Example of the four tasks in the protocol. In the top-right corner of each interface, the countdown before the next interface appears.

4.9 Evaluation Methods

For a comprehensive and rigorous evaluation of the proposed classification algorithm, two primary criteria were employed: the classification accuracy and processing speed. These methods allowed to quantify both the predictive efficiency of the model and its computational efficiency, providing a complete summary of the performance.

4.9.1 Classification Accuracy

Accuracy is one of the most common and intuitive metrics for evaluating the performance of a classification model. This represents the proportion of correct predictions carried out by the model compared to the total number of predictions. In other words, it measures how often the algorithm correctly "guesses" the state (stress or relaxation) in which an EEG segment belongs.

Formally, the accuracy is calculated as:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}} \tag{4.7}$$

A high Accuracy indicates that the model can reliably distinguish between the different classes, which is fundamental for practical applications where the correctness of the predictions is critical.

4.9.2 Algorithm Velocity

Beyond the accuracy, the *processing speed* is a crucial parameter, especially in contexts which require real-time (or almost real-time) answers, such as BCI systems. To quantify this metric, the time taken by each 2-second-long segment to complete the entire inference process will be measured. This includes the following phases:

- *Segment loading*: time taken to recover the data segment from the memory or the acquisition stream.
- *Preprocessing*: time taken to perform normalisation, filtering and ICA.
- *Classification*: time taken by the EEGNet to predict the class of the segment studied.

After this, to provide a velocity measure comparable to other BCI studies, the Information Transfer Rate (ITR) was calculated. ITR is a standard metric, mainly used for BCI spellers, which quantifies the quantity of useful information which a system is able to transfer in a certain time interval. It takes into account not only classification accuracy but also the algorithm speed, offering an estimate of how quickly the system processes information. A high value of ITR indicates a more efficient system in translating the intention or the cognitive state into an action or an estimate.

The general formula for the ITR [61] is:

$$\text{ITR (bit/trial)} = \log_2(N) + P \cdot \log_2(P) + (1-P) \cdot \log_2\left(\frac{1-P}{N-1}\right) \tag{4.8}$$

$$\text{ITR(bit/min)} = B \cdot \text{ITR (bit/trial)} \tag{4.9}$$

where:

- N is the number of possible classes (in this work, 2).
- P is the mean classification accuracy, expressed as a probability between 0 and 1.
- B is the number of classifications per minute, calculated as $60/T$, where T is the processing time for each classification (in seconds).

The combination of accuracy and ITR offered a robust evaluation of the performance of the classification algorithm, allowing the comparison of the results with other pre-existing methodologies, highlighting both precision and efficiency.

4.10 Acknowledgements

For this research, the author used an AI assistant to identify and correct errors in the code. The main contribution was debugging and refining existing code segments to improve robustness and accuracy. This ensured the reliability of the computational results presented.

Chapter 5

Results

This chapter describes the techniques used for model optimisation, presents the typical loss and accuracy curves of EEGNet training, and then examines its generalisation through two different K-fold cross-validations on subjects and using a different dataset. Ultimately, it focuses on the velocity of the classification algorithm.

5.1 Hyperparameter Optimisation via Fine-tuning and K-fold Cross-Validation of EEGNet

To achieve optimal performance with the EEGNet model, a systematic process of hyperparameter optimisation was conducted. This involved fine-tuning some key model parameters and rigorously evaluating the resulting configurations through K-fold cross-validation to ensure robust performance estimates.

5.1.1 Impact of Batch Size on Testing Accuracy

In the initial phase of the hyperparameter exploration, the influence of the *batch size* on the *testing accuracy* of the EEGNet model was studied. The *batch size* represents the number of training samples that the model processes in a single iteration. This is a crucial hyperparameter that influences both the training dynamics and the generalisation ability of the NN, as it determines the gradient estimate used for weight updates.

Smaller batch sizes will provide a noisier gradient estimate, which could help the model avoid the sharp minima and potentially generalise better. However, this leads to slower training and requires more frequent weight updates. On the contrary, larger batch sizes offer a more fluid and stable gradient estimate, potentially leading to a more rapid convergence, but also risking a worse generalisation and a convergence to sharper minima.

The following discrete values were examined: $[16, 32, 64, 128]$. These values were chosen to cover a range of common batch sizes used in articles about DL.

The model was trained and tested ten times, and the average testing accuracies $\pm$ standard deviation obtained for each batch size are summarised in Tab. 5.1.

Batch Size	16	32	64	128
Avg. Test Accuracy (%)	89.46 ± 1.37	91.04 ± 1.59	90.42 ± 0.58	86.59 ± 2.85

Table 5.1: Average $\pm$ standard deviation testing accuracy of EEGNet across different batch sizes.

Results reveal that the batch size has a big effect on the model performance. In particular, the average test accuracies achieved with batch sizes of 16 and 128 were inferior by more than one percentage point compared to the accuracies achieved with batch sizes of 32 and 64. In addition, the standard deviation of the accuracies provides information about the consistency of the performances across the different folds. The relatively lower standard deviation observed with a batch size of 64 indicates a more stable and consistent performance across different runs. Based on these initial results, the following in-depth analysis about the other hyperparameters was conducted, focusing only on the batch sizes of 32 and 64, parameters which had shown superior and more stable performance in this initial evaluation.

5.1.2 K-Fold Cross Validation

To obtain an unbiased estimate of the generalisation performance of the EEGNet and to guard against overfitting to any particular data split, the K-fold cross-validation technique was applied. In this procedure, the available dataset is partitioned into K folds of the same dimension. For each of the K iterations, a different fold is kept aside as a validation set, while the remaining $K-1$ folds are used for the training. The model is trained from scratch on the training folds and then is evaluated on the fold kept aside. After cycling through all the K folds, the average of the validation accuracies is calculated. This approach ensures that each sample is used both for the training and for the validation exactly once, providing a robust evaluation of the model's performance on the whole dataset and reducing the variance due to random train-validation splitting.

In the experiments, $K = 5$ was used. This is a very common value in literature since it balances the big training data needed by a DL net and a big enough test set to have reliable results. In this way, the net is trained with 80% of the data, and 20% is used for testing. For each fold, a grid search was performed over the following hyperparameter space:

- Batch size: $\{32,\ 64\}$
- Learning rate (η): $\{7.5 \times 10^{-4},\ 1 \times 10^{-3},\ 2.5 \times 10^{-3},\ 5 \times 10^{-3}\}$
- Scheduler usage: $\{$`True`, `False`$\}$
- Weight decay: $\{0,\ 1 \times 10^{-4},\ 1 \times 10^{-3}\}$
- Adam β coefficients (β_1, β_2): $\{(0.9, 0.999),\ (0.8, 0.98)\}$

A total of $2 \times 4 \times 2 \times 3 \times 2 = 96$ hyperparameter combinations were evaluated on each of the 5 folds, yielding 480 training runs.

Table 5.2 lists the top 8 configurations ranked by their mean validation accuracy $\pm$ standard deviation over the 5 folds. These are the only configurations which reached a mean accuracy greater than 90%. For the complete 96 combinations results, see Appendix A.1.

Rank	Batch size	LR	Scheduler use?	Weight decay	β_1, β_2	μ (Acc.%)
1	64	5×10^{-3}	True	1×10^{-3}	(0.9,0.999)	90.79 ± 1.91
2	32	2.5×10^{-3}	True	1×10^{-4}	(0.8,0.98)	90.60 ± 1.30
3	32	5×10^{-3}	True	1×10^{-4}	(0.8,0.98)	90.56 ± 1.67
4	64	5×10^{-3}	True	0	(0.9,0.999)	90.52 ± 1.79
5	32	5×10^{-3}	True	0	(0.8,0.98)	90.25 ± 1.45
6	64	2.5×10^{-3}	False	0	(0.9,0.999)	90.23 ± 1.49
7	32	2.5×10^{-3}	False	0	(0.8,0.98)	90.15 ± 1.88
8	32	5×10^{-3}	True	1×10^{-3}	(0.9,0.999)	90.06 ± 1.91

Table 5.2: Top 8 hyperparameter configurations from 5-fold cross validation, ordered by mean validation accuracy μ.

Analysing the results from the table, it can be seen that the top-performing configuration (Rank 1) utilised a batch size of 64, a learning rate of 5×10^{-3}, employed a scheduler, a weight decay of 1×10^{-3}, and Adam β coefficients of $(0.9, 0.999)$, achieving a mean accuracy of $90.79\% \pm 1.91\%$.

Interestingly, a learning rate of 5×10^{-3} appears more frequently in the best configurations. This suggests its strong effectiveness for this model and dataset. The second most frequent learning rate in the best eight configurations is 2.5×10^{-3}. These two learning rates are also the two highest values in the list of the learning rates investigated in the K-fold cross-validation. The other possible values, 1×10^{-3} and 7.5×10^{-4}, do not appear in the top eight configurations. This indicates that the EEGNet, at least with the dataset analysed, needs a more aggressive learning rate to effectively capture relevant features.

In addition, including a learning rate scheduler seems to be a significant factor in achieving high performance. This is clear from the fact that the best five configurations all include a learning rate scheduler.

In the best eight configurations, both batch sizes 32 and 64 are utilised. Although there are slightly more configurations with a batch size of 32 in the top-8, the best configuration utilises a batch size of 64, suggesting that both the values can be used to achieve excellent results, depending on the other hyperparameter settings. In the same way, both the weight decays and the tuples of Adam coefficients β, demonstrated robust performance through different hyperparameter settings.

In light of these findings, it can be concluded that the learning rate and the use of a scheduler are probably the most influential parameters to achieve a high accuracy with an EEGNet. While batch size was crucial in initial optimisation steps to narrow down to the effective range of 32 and 64, within this pre-optimised range, its specific value, along with weight decay and the β coefficients, played a comparatively less critical role in determining the final performance.

5.1.3 Segment Length

The signal length is a crucial parameter that determines the duration of the segments in which each signal is split before being fed into the EEGNet for training or classification.

All the preceding experiments have been conducted with a signal length of two seconds. To identify the most suitable signal length, another K-fold cross-validation was implemented. The other hyperparameters utilised are the same of the best configuration in the preceding cross-validation: batch size of 64, learning rate of 5×10^{-3}, weight decay of 1×10^{-3}, and $\beta = (0.9, 0.999)$, with scheduler.

The following discrete values have been examined: $[1, 2, 3, 4, 5]$ seconds. These values were chosen so that the segments were neither too short, which could lack sufficient temporal information to accurately capture physiological changes related to stress, nor too long, which could lose temporal and non-stationary information within the segment.

Segment Length (s)	1	2	3	4	5
Accuracy % ($\mu \pm \sigma$)	88.37 ± 3.15	90.36 ± 3.20	88.25 ± 3.32	83.83 ± 2.76	78.50 ± 4.25

Table 5.3: Accuracy % by Segment Length (s).

From the results presented in Tab. 5.3, it becomes evident that increasing the segment length over a certain limit does not lead to an improvement in the classification accuracy. Longer segments (4 and 5 seconds) seem to negatively influence the performance, with the 5-second segments showing a noticeable drop in the accuracy, reaching $78.50\% \pm 4.25\%$.

On the contrary, signal segmentation into shorter durations, in particular 2 seconds, produced higher average accuracy values, achieving $90.36\% \pm 3.20\%$. Segment lengths of 1 and 3 seconds both resulted in slightly lower accuracies ($88.37\% \pm 3.15\%$ and $88.25\% \pm 3.32\%$, respectively).

This trend suggests that shorter segments are generally more effective in capturing relevant temporal dynamics related to stress. Longer segments might introduce additional variability, not recognised by the EEGNet, thereby reducing the model's performance.

Overall, the best performance is achieved with a segment length of two seconds, offering an optimal trade-off between capturing sufficient temporal context and avoiding the dilution of stress-related features across long segments. For these reasons, the final segment length used in the model is two seconds.

5.2 Training and Validation Performance Analysis

After optimising the EEGNet parameters, it is possible to analyse the trend of the training and validation curves for the loss function and accuracy as a function of the number of epochs. This can provide insights into the stability and quality of the EEGNet, as well as its capacity for generalising. Fig. 5.1 illustrates these trends for the model trained with the optimised parameters.

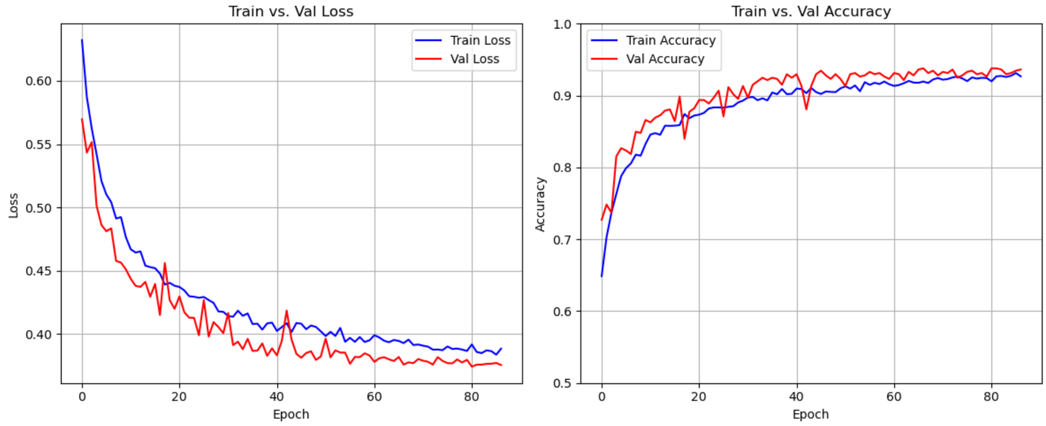

Figure 5.1: Training and Validation Loss (left) and Accuracy (right) as a Function of Training Epoch.

5.2.1 Analysis of Loss Curves

The left panel in Fig. 5.1 shows the trend of the loss function of training and validation as a function of the training epochs. Both curves exhibit a characteristic decreasing trend, indicating that the model is efficiently learning to minimise the difference between its predictions and the real labels. The steeper initial decline in the two curves suggests a rapid acquisition of the most salient features during the first epochs. As training progresses, the reduction of the loss function becomes less evident, eventually stabilising in the final epochs. This asymptotic behaviour implies that the model has converged, and further training yields diminishing returns in terms of reducing the overall error.

Interestingly, the validation loss curve closely follows the training loss function, without a significant divergence. This suggests that the model is generalising well to unseen data, and the net is not overfitting the training set with these hyperparameter settings. A substantial increase in the validation loss, despite a continued decrease in the training loss, would have pointed to overfitting, a phenomenon that seems not to be present with this particular training configuration.

Even if the training loss curve is almost monotonically decreasing through all the epochs, the validation loss presents occasional peaks. These indicate that not all the weight modifications performed during the net training to reduce the training loss are equally efficient on data outside the training set. However, this behaviour does not seem to pose a significant problem, since, after a few epochs, the validation loss stabilises again, resuming its monotonic, decreasing trend it was following before. In particular, in the last epochs, the validation loss appears to be constant and without any significant fluctuations, suggesting that the modifications in the net weights are minimal and lead to negligible variations.

5.2.2 Analysis of Accuracy Curves

The right panel in Fig. 5.1 shows the training and validation accuracy as a function of the training epochs. Consistent with the trend of the loss curves, both accuracy curves show a general increasing trend, indicating an improvement in the model capacity of classifying correctly the EEG signals in time. The rapid initial increase of the accuracy corresponds to the fast diminution observed in the loss curves, further highlighting the efficient learning of discriminatory features in the initial training phases.

The near-monotonic increase in both training and validation accuracy highlights the stability of the learning process. In particular, the validation accuracy stabilises and slightly fluctuates in the final epochs, suggesting that the model reached a point of maximum generalisation performance. While the training accuracy still shows a marginal increase, the stabilisation of the validation accuracy indicates that further training might not lead to significant improvements in the model's capacity to classify new, unseen data, but it might reduce it. To avoid this, the patience of 20 epochs without any improvement in the validation accuracy stopped the learning at around 85 epochs. The proximity of the training and validation accuracy also supports the idea of good generalisation for this specific hyperparameter configuration.

As a parallel to what has been observed for the loss function, where the validation loss presented some peaks, which indicated that not all the training improvements are translated into benefits for unseen data, in this plot, a similar behaviour is presented. While the training accuracy curve is monotone and does not show evident valleys, the validation accuracy curve shows valleys which temporally correspond to the epochs in which the peaks in the validation loss were present. This parallelism reinforces the hypothesis that those peaks were the result of less effective modifications of the net weights during the training. Similarly to the loss function, after these fluctuations, the validation accuracy tends to stabilise again, continuing its monotone trend, reaching a plateau of performance in the last epochs.

5.3 Evaluation on 10% of the Dataset

Following the K-fold cross-validation experiments and the EEGNet training and validation loss and accuracy curves analysis, the net has been evaluated on a dedicated held-out test set. This evaluation has used the hyperparameter configuration which has been determined as optimal in the previous experiments: 2-second long segments, learning rate of 5×10^{-3}, weight decay of 1×10^{-3}, and Adam parameters $\beta = (0.9, 0.999)$, with learning rate scheduler but without early stopping.

In previous analyses, the training of the net always included a validation set. This is fundamental to detect when the net is learning features which are too specific to the training set, not relevant for external data, and so to prevent overfitting. Training a net without a validation set has the advantage that more data is available for the training, allowing the net to acquire more relevant and generalised features. A downside of this is that there is no way to understand when the net is not correctly generalising, meaning that there is no reliable variable which can be used to decide when to stop the training. Tab. 5.2 did not definitively identify the superior batch size between 32 and 64, and so both values were considered in this analysis.

The dataset was split into a training set of 90% and a test set of 10%, with 2-second-long segments from all subjects in both sets. The net was trained for a fixed duration of 100 epochs on the training set. To get a more stable and reliable estimate of the model's performance on the test set, the entire training process was run ten times for both batch sizes, 32 and 64.

The resulting test accuracies are presented in Tab. 5.4.

Acc.%, BS=32	91.25	94.17	92.92	88.33	90.83	91.67	93.75	91.25	89.17	87.92
Acc.%, BS=64	88.75	90.00	90.42	86.25	90.83	94.58	88.75	90.00	92.08	91.25

Table 5.4: Test Accuracies on 10% Held-Out Dataset for Batch Sizes 32 and 64. Respective Mean and Standard Deviation are: 91.13% ± 2.04%, and 90.29% ± 2.10%

The average test accuracy for batch size 32 (91.13%) is slightly higher than for batch size 64 (90.29%). Furthermore, the standard deviation for batch size 32 (2.04%) is lower compared to the one from batch size 64 (2.10%). This indicates that the results for batch size 32 are slightly more consistent in the ten repetitions, suggesting that this batch size might provide slightly better and more reliable generalisation performance on the held-out test set. For these reasons, the final network uses a batch size of 32.

Tab. 5.5 shows the six optimised parameters and their final value.

Batch Size	Learning Rate	Scheduler	Weight Decay	Betas	Segment Length
32	$5 \cdot 10^{-3}$	True	$1 \cdot 10^{-3}$	$(0.9, 0.999)$	2 seconds

Table 5.5: Optimised parameters and their value.

5.4 K-Fold Cross-Validation on Subjects

To further evaluate the model's ability to generalise to unseen data, a K-fold cross-validation at the subject level was performed. This approach guaranteed that the training and test sets contained different individuals, providing a more robust estimate of the model's performance with new subjects. Two different values of K were explored.

5.4.1 K=5

A cross-validation with $K = 5$ is a very common choice, dividing the dataset into five folds of the same dimensions. In each iteration, four folds (80% of the subjects) are used for the model training, while the last fold (20% of the subjects) is used as a test set. With a total of 40 subjects in the dataset, 32 subjects were used for the model training, and 8 for the testing for each fold.

The test accuracies obtained for each of the five folds are presented in Tab. 5.6.

Fold	1	2	3	4	5	Mean ± std.
Accuracy	70.95%	71.79%	52.20%	71.96%	61.82%	65.74% ± 7.76%

Table 5.6: Test Accuracies for K=5 Cross-Validation.

The overall mean test accuracy across the five folds gives an estimate of the expected performance of the model on new and unseen subjects.

Table 5.6 shows a mean test accuracy of 65.74%, with a standard deviation of 7.76%. In particular, the third fold achieved a lower, almost random accuracy (52.20%), while the other folds varied between 61.82% and 71.96%. The variability in the accuracy between the folds suggests that the capacity of the model to learn features that generalise to unseen subjects might be limited. To further investigate the capacity of generalisation of the model, another K-fold cross-validation has been performed, this time using K=40.

5.4.2 K=40

In this k-folding, the net is trained using 39 subjects, and is tested using one subject. This simulates the application of a trained net on a new subject, that is, a potential application of the net in the real world.

The test accuracies obtained for each of the folds are presented in Fig. 5.2.

Fig. 5.2 illustrates the net's performance on the single subjects, highlighting the huge variability in the test accuracies. These accuracies vary between a minimum of 4.05% for subject 21 to a perfect 100% for subject 32, highlighting the substantial inconsistency of the net's capacity to classify the data from different subjects. The figure also indicates the comprehensive accuracy of the $K = 40$ cross-validation, which is $67.03\% \pm 24.04\%$.

This average is similar to the 65.74% achieved in the $K = 5$ cross validation, suggesting a limit in the net's capacity to generalise on unseen individuals. The high standard deviation of 24.04% confirms this, indicating a significant dispersion in the predictive capability across the subjects. In other words, the model classifies perfectly some subjects, while it struggles with others. This inconsistency raises concerns about the practical applicability of the model to new populations.

This observation reinforces the hypothesis that the net, in this current configuration or with the current available training data, might not be able to learn features which are robust enough and invariant to the inter-subject variability. This

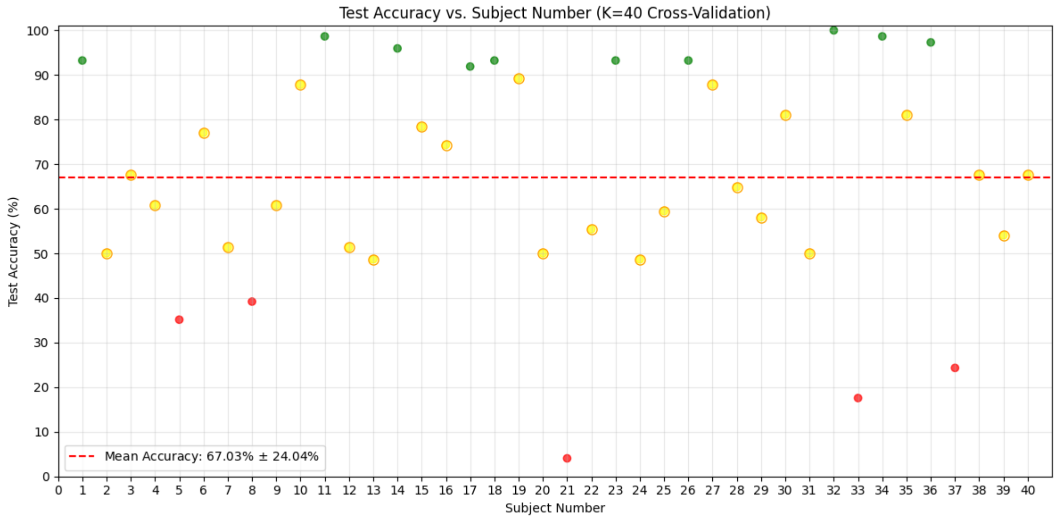

Figure 5.2: Test accuracies for each subject in the K=40 cross-validation, illustrating the variability of the model's performance across individual subjects. In yellow, the subjects who performed in the range mean ± standard deviation. In red, the subjects with lower values. In green, the subjects with higher values.

suggests that the model might involuntarily learn patterns more subject-specific than generalisable underlying characteristics relevant to the task.

5.5 EEGNet Applied on New Data

The preceding analyses were conducted utilising the SAM40 dataset, which comprises EEG recordings from 32 channels sampled at 128 Hz. However, the data acquisition device employed for the present study is limited to 16 EEG channels, sampled at 125 Hz.

Due to these two substantial differences, the author chose to retrain the EEGNet using the new dataset. The same hyperparameters as the ones optimised in Sec. 5.1 were used, except for the learning rate, which was reduced to 4×10^{-5}.

This modification was necessary because with the previous learning rate (5×10^{-3}), the model training was not effective. It is possible that the optimal learning rate does not depend only on the specific hyperparameters of the net, but also on the intrinsic properties of the dataset used. Data with different characteristics (for instance, number of channels, sampling frequency, SNR) might require a lower learning rate to allow the model to converge with more stability without losing the minima of the loss function.

The division of training and test sets was tested in two different ways. In the first case, the training and test sets both contained data from all five sessions. In the second case, the test set was the data from the last session of recordings. In both cases, the inputs for the EEGNet were 2-second-long segments, as was in the trials based on the SAM40 dataset.

5.5.1 Testing on the Data from all the Sessions

In this trial, the data from the five sessions was split into 2-second-long non-overlapping segments and then divided into training, validation and test sets with an 80%, 10%, 10% split, chosen to maximise the data available for training.

In the preprocessing steps, which were the same used for the SAM40 dataset, two different values of kurtosis were tested for the ICA to optimise the threshold of the components to be eliminated. Tab. 5.7 shows the percentage accuracy values achieved in ten different runs, with two different values for the kurtosis.

kurt=8	90,72	89,69	89,18	90,72	92,78	88,66	91,75	92,78	88,14	89,69
kurt=12	94,33	94,33	90,72	89,18	90,21	93,30	91,24	94,95	93,81	95,36

Table 5.7: Accuracy % for ten runs varying the kurtosis values. Kurtosis threshold of 8 led to an average accuracy of $90.41\% \pm 1.55\%$, while a kurtosis threshold of 12, to an average accuracy of $92.73\% \pm 2.08\%$

As it was expected, the testing accuracies are extremely high. This is mainly because of two reasons:

- The net has been previously optimised (see Sec. 5.1). The optimisation of the EEGNet ensured that the architecture and the parameters were suited for the classification task. This means that the net was already able to efficiently learn the complex patterns present in the data, achieving in this way higher performance.

- Using the data from a single subject drastically reduces the inter-subject variability. NNs tend to perform better on data coming from the same source on which they have been trained, because they can learn and exploit characteristics which are subject-specific. This is a significant factor which contributes to the almost perfect accuracy, but could limit the generalisability of the model to new subjects.

From Tab. 5.7, it is also evident that the best threshold between 8 and 12 for the kurtosis is the second one. The corresponding average and standard deviation for the two thresholds are: $90.41\% \pm 1.55\%$ and $92.73\% \pm 2.08\%$. This result is intuitively sensible for the following reasons:

- *Lower threshold - 8*: a lower threshold in the ICA implies that a higher number of components will be classified as artefactual and eliminated. There is a significant risk that useful components, containing information about the brain patterns which characterise stress and relaxation states, but presenting a high value of kurtosis, will be removed. This will impoverish the data, thereby reducing the model's capacity to correctly distinguish between the two states.

- *Higher threshold - 12*: a higher threshold is more selective and tends to remove almost exclusively the artefactual components. This allows for better preservation of the most relevant brain components, achieving a higher final accuracy.

By analysing these results, the default value for the kurtosis threshold function was set to 12.

5.5.2 Testing on the Data from the Fifth Session

In this trial, the data from the first four sessions was used to train the EEGNet, while the data from the last session was used to test the model. After 40 cycles of training and testing, the net produced a mean accuracy of 67.65% $\pm$ 6.76%. The 40 accuracy values are presented in Fig. 5.3.

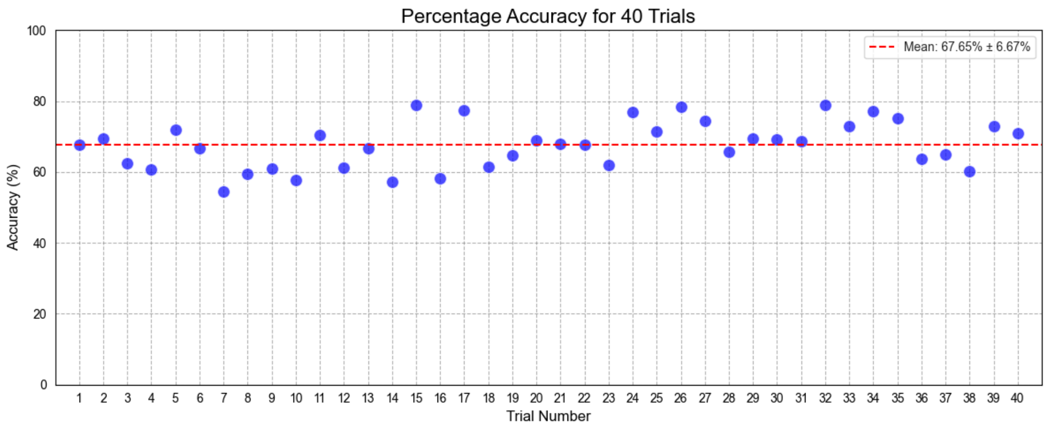

Figure 5.3: Percentage accuracies of the model over 40 trials, testing on the data from the fifth session.

This value is considerably low, especially if compared to the 92.73% obtained in the previous trial, where the training and the test sets included data from all sessions. This drastic reduction in the performance is probably due to the incredibly high variability of the EEG signal. It has been previously explained that the EEG signal is subject-specific, meaning that it varies a lot across different subjects (see Sec. 2.1). However, it seems that a similar phenomenon is happening on the same subject across different sessions. Even if the brain is the same, and the external stimuli are identical, the EEG signal can vary and modify its behaviour, its dominant frequencies, or the patterns, with time. This intra-subject variability across sessions makes the classification extremely difficult, because the model trained on the first four sessions might not efficiently generalise the patterns, even if they are from the same subject, of the fifth session. The model, having learnt the specific features from the first sessions, struggles to adapt to the slight but significant differences that emerged in the last recording, independently of the good preprocessing techniques.

Other factors that might have contributed to the accuracy reduction in this configuration include:

- Contact between electrodes and scalp: a suboptimal contact of the electrodes in some sessions can introduce noise or signal distortion, making the classification task harder for the EEGNet.

- Hydration of the subject: the hydration of the subject can influence the skin impedance and, consequently, the quality of the EEG signal.
- Room condition: variations in the environmental conditions in the room (for instance, temperature or humidity) can influence the quality of the recorded data and their coherence between sessions.

These elements, especially if combined, together with the natural variability of the EEG signal with time, even in the same subject, explain the drop in the accuracy when the model is tested on a session never seen during the training.

5.6 Velocity of the Algorithm

The processing time is a crucial factor for real-time applications, especially in contexts where an immediate answer is required. To evaluate this aspect of the algorithm, the total time needed for the loading, preprocessing and classification of a 2-second-long segment was measured. The measure was repeated ten times for more reliable results.

The total time recorded was 252.7 ms $\pm$ 45.5 ms. Analysing the phases of the algorithm, these are the mean timings:

- Data Loading: 40.2 ms $\pm$ 7.6 ms
- Preprocessing and Data Preparation: 177.5 ms $\pm$ 47.2 ms
- Classification: 35.0 ms $\pm$ 5.6 ms

As can be noted, the preprocessing phase is the most computationally expensive component, taking more than 70% of the total time. This was expected, since this step includes ICA. This is fundamental for the artefact removal and for the improvement of the signal quality, but is known to be computationally complex, and this justifies the greater contribution to the total processing time. Inference in the EEGNet, on the contrary, is extremely fast, demonstrating the efficiency of the model in the classification phase.

5.6.1 ITR

Based on the processing times measured and on the accuracy achieved, the ITR was calculated, utilising the formulas presented in Equations 4.8 and 4.9. For this calculation, the following parameters were used:

- N = 2 classes.
- P = 0.9273 is the accuracy, calculated from the testing on all the sessions.
- B = 60 / 2.2527, resulting in 26.6 classifications per minute.

Applying the values to the formula, the ITR value obtained is **16.62 bits/min**.

5.6.2 Literature Comparison

The only article which showed potential for real-time applications was [13]. Here, the preprocessing time was 320 milliseconds and the classification time 7.1 milliseconds for each segment. The results proposed in this work are five times slower for the classification (35.0 milliseconds), but took half of the time for the preprocessing (178 ms instead of 320 ms). This trade-off between preprocessing and classification time highlights a key aspect of system design for real-time EEG-based applications: optimising the pipeline as a whole is more impactful than focusing on a single stage. In this sense, the proposed method achieves a reasonable balance between computational load and classification accuracy, and paves the way for future improvements aimed at meeting the stringent constraints of fully real-time systems.

The research for scientific papers studying EEG-based stress analysis with ITR as a measure for the algorithm velocity produced no direct reference in the field. However, it is possible to compare the ITR value with studies from other BCI domains. In literature, ITR values are extremely variable, reflecting the diversity in methodology, applications, and EEG signal characteristics.

For instance, spellers based on Steady-State Visually Evoked Potentials (SSVEP) can reach significantly higher ITR values, such as the 105 bits/min achieved in [62]. On the other hand, Motor Imagery (MI) tasks, which often present higher complexity in decoding the intentions of the subject, achieve lower ITR values, such as the 5.99 bits/min reached in [63].

The ITR of 16.62 bits/min, even if it is not between the best ITR values in literature, is in a reasonable range and shows a discrete capacity of information transfer for an EEG-based stress analysis system. It is important to note that stress analysis is an intrinsically more complex task compared to spellers, and this may justify the lower ITR value.

5.6.3 Potential Improvement

Further preprocessing optimisation might lead to an increase in the ITR, making the algorithm more reactive. However, it is important to see the trade-off between velocity and accuracy of the system. A more aggressive or simplified preprocessing, aiming to reduce the elaboration time, would potentially lead to a decrease in the accuracy. This would compromise the model's capacity to correctly distinguish stress and relaxation states, due to the loss of useful data or components.

To conclude, every future velocity optimisation should be balanced with an attentive evaluation of the impact on the accuracy.

Chapter 6

Discussion

Despite the promising performance of the classifier, particularly in the cases in which training, validation and test sets contained segments from the same subjects (as in the analysis with the SAM40 dataset), or from the same recording session (as in the recorded dataset), many limitations reduce its practical applicability.

6.1 Classifier Performance and Generalisation Limits

The best results achieved with the EEGNet are slightly lower than those reported in the literature (tables 3.4 and 3.5). However, many articles do not treat explicitly the problem of the model generalisation on subjects not present in the test set. Those models might show a significant drop in performance if tested on new data from different subjects.

The accuracy drop observed in this work might also be due to the limited dimension and the low variability of the dataset. Increasing the dataset dimension, in terms of the number of subjects, sessions, and trials, might improve the classifier's capability to learn features independent of the subjects, more correlated to stress and relaxation states.

Possible new data must maintain coherence in the acquisition parameters, such as sampling frequency, number and type of channels, and electrode positioning. This standardisation would facilitate the learning, improving the model's capability to find hidden patterns in the EEG signals.

6.2 Device Limitations

The EEG cap used for the data acquisition presents many limitations concerning the realisation of a real-time stress detection system which could be used in daily life:

- *Absence of real-time acquisition*: the device does not support the real-time EEG data stream. The Graphical User Interface (GUI) allows exclusively a qualitative visualisation of the signals to verify the presence of particularly active bands, or railed electrodes. To save the data, it is necessary to transmit it through LSL to an external software (Lab Recorder), making a live analysis impossible.

- *Discomfort in a prolonged use*: the rigid electrodes directly press on the scalp, causing discomfort to the subject even after less than one hour of usage. This makes it inadequate for prolonged sessions.
- *Scarce fitness for daily usage*: the device is bulky and visible, resulting in incompatibility with working or social environments. Its appearance might generate embarrassment in the subject who wears that.
- *Movement sensibility*: some electrodes tend to become railed even in the presence of little cap movements. Even if it is possible to manually reposition them, in the absence of the GUI, the problem might be unobserved, compromising the quality of the acquired data.

Between the devices presented in Fig. 2.3, the *Emotiv EPOC X* represents a better alternative for future applications. This system presents:

- *Higher comfort*: the soft electrodes better adapt to the scalp shape, without causing discomfort, even for prolonged sessions.
- *Lower impedance*: the electrodes can be wetted with electrolytical solutions, improving the conductivity and the SNR.
- *Frontal focalisation*: a higher number of electrodes is positioned in the frontal lobe, an area which is particularly important for stress analysis [54].
- *Discrete design*: the device is less visually invasive, and can be more easily hidden, making it suitable for an usage in daily contexts.

These characteristics make the Emotiv EPOC X, or other similar Emotiv EEG devices, promising devices for future developments of real-time wearable stress detection systems.

6.3 Ethical considerations

The proposed algorithm has the potential to improve the mental health and well-being of individuals through an early detection of stress conditions and the activation of personalised interventions. However, an improper use might generate grave ethical problems.

One possible misuse scenario involves excessive monitoring in professional environments to optimise employee performance. In such a context, stress metrics could be used not to improve well-being, but to enforce productivity targets, detect moments of reduced concentration, or even penalise employees perceived as “not stressed enough” to be working at full capacity. This risk of over-monitoring is not unique to stress detection systems but is common to many emerging technologies involving biometric data.

Even in domestic individual usage, misuse of the stress analyser can be harmful. Continuous self-monitoring of mental state might generate an anxiety or over-alert state, leading the person to excessively worry about their stress levels, even in the absence of real signs of danger.

It is important that the development and usage of these technologies are followed by clear ethical guidelines, guaranteeing transparency, informed consent, privacy and responsible use of data. The EEG-based stress detection systems should only be employed in contexts that bring concrete and voluntary benefits to the subject, respecting their rights and dignity.

Chapter 7

Conclusion

The purpose of the study was to build a fast, automatic, and accurate EEG-based classifier for stress analysis. To do so, a preprocessing was needed, followed by data preparation to feed the data into the model.

7.1 Summary

The first step consisted of building a robust preprocessing algorithm. It started with a z-normalisation to normalise the data so that every channel had a zero mean and a unit variance. Then, the literature suggested that the most important frequency band to be studied was between 4 and 30 Hz, and so a BPF was applied. The cut-off frequencies were 1 and 40 Hz, to minimise the signal distortion in the useful bandwidth, but also properly remove low-frequency drifts and high-frequency noise (EMG signal and power line interference). To further suppress the power line interference, a notch filter was applied at 50 Hz. A powerful and automatic ICA algorithm based on the `FastICA` function was used. This helped to remove visually evident artefacts such as ocular movements, which were not filtered through the previous filtering steps.

After this, the data was divided into 2-second-long subsegments and split into training, validation, and test sets. The EEGNet [1] model was slightly modified, and subsequently it was trained and tested. To maximise the performance of the model, many K-fold cross-validation experiments were run. These experiments varied the batch size, learning rate, scheduler usage, weight decay, Adam β coefficients, and segment length, and were used to perform the fine-tuning of the EEGNet.

Testing the model achieved an accuracy of 91.13% $\pm$ 2.04% with the net trained and tested with segments from all the subjects (but not repeated in the different sets). However, if the model was trained on 39 subjects and tested on the last subject, the EEGNet achieved only an accuracy of 67.03% $\pm$ 24.04%. This showed a low generalisation ability for the model on new subjects.

To verify that the model was able to perform well on different datasets, new EEG data was collected from a single subject. Using this, the EEGNet was trained and tested. The test accuracy was over 92%, indicating that the model was efficiently classifying the data into the two classes. However, when the net was trained on data from the first recorded sessions and tested on the last recorded session, the accuracy dropped to 68%. Knowing the stochastic behaviour of the EEG signal between subjects, it was hoped that on the same subject, it would maintain the same characteristics. This was revealed to be partially true, especially on data from the same session, and not on data from different sessions.

In the end, to address the need for a fast algorithm, the velocity of the entire process, from data loading to classification, was calculated, resulting in 252.7 milliseconds. This can be considered to be fast enough for a real-time application. The ITR value was also calculated and resulted to be 16.62 bits/min. Many comparisons can be done with ITR values of SSVEP spellers, which can achieve ITR values higher than 100 bits/min [62], and MI classifiers, with ITR values that can be lower than 10 [63]. Unfortunately, it was not compared to other stress-classifiers' ITR values, as in those analyses, the classification time is usually not calculated, since they often aim to achieve high accuracy values, regardless of the time required by the process.

7.2 Limitations

Despite the promising results, this study is subject to several limitations that require consideration for future research. Firstly, the generalisation capabilities of the developed model are inherently tied to the characteristics of the datasets used for training, which were primarily collected in controlled laboratory environments. Real-world stress manifestations can be far more varied and complex. The scope of stress-inducing stimuli was confined to SCWT, Arithmetical, and Symmetrical Images tasks. Expanding to a broader range of ecological stressors would enhance the model's robustness. In addition, while the system demonstrated almost real-time performance, the computational demands, particularly during the preprocessing phase, could pose challenges for deployment on highly resource-constrained edge devices, necessitating further optimisation. Future work should therefore aim to address these limitations by validating the system in naturalistic settings.

7.3 Future Research

Future research should focus on four main topics:

- *Applying the model in an online real-time application*: to make this research impactful, the EEGNet has to be tested in an online real-time application. This implies the creation of an algorithm that takes real-time data from a subject, preprocesses it, and classifies it. Due to the low performance of data from different sessions, the idea would be to record the EEG signal at the beginning of each session in both a stressed and a relaxed condition. Then, train the model using this data and a dataset with the same characteristics in terms of sampling frequency, electrodes number and positioning as the device used.

In the end, perform real-time classification of the new EEG signals using the EEGNet model, with the weights from the previous training. This is expected to achieve performance higher than the 68% achieved in this research, since data from the same session would be present in the training set.

- *Finding a faster artefact removal algorithm*: the proposed algorithm is extremely fast, but the data preprocessing is particularly time-consuming, especially due to ICA. Finding and using a different algorithm to automatically remove artefacts, especially ocular movements, would significantly help reduce the time required by the pipeline.

- *Analyse the impact of the segment length*: experimental results showed that, in the SAM40 dataset, a 2-second segment provided an effective compromise between classification performance and temporal resolution, achieving higher accuracies than longer segments. However, this duration might not be optimal for other datasets, especially if the stress dynamics develop over longer periods. Furthermore, on short segments, algorithms such as ICA might fail to converge, degrading the quality of the artefact removal. A valuable direction for future work would therefore be to investigate the performance of the algorithm on longer segments, assessing the trade-off between preprocessing stability and classification accuracy.

- *Exploring alternative data augmentation strategies*: the *linear surrogating* technique led to overfitting, and was therefore not implemented in the research. Other data augmentation methods could be investigated to increase the robustness and the generalisability of the model.

In conclusion, this work demonstrated the feasibility of an automatic system for EEG-based stress analysis, using a light and fast architecture, fit for a real-time application. Despite the highlighted limitations, the results achieved represent a concrete step towards the development of wearable and customizable solutions for mental state monitoring. Potentially, this approach is not limited only to stress detection, but could be applied in mental health, fatigue detection, and cognitive performance. Further developments in terms of generalisation, preprocessing optimisation, and real-environment validation would offer efficient tools for the prevention and management of stress in daily life.

Appendix A

K-Fold Cross-Validation Table

This chapter presents the table with all 96 hyperparameter combinations, coming from the experiments presented in Section 5.1.2.

Rank	Batch size	LR	Scheduler use?	Weight decay	β_1, β_2	μ (Acc.%)
1	64	5×10^{-3}	True	1×10^{-3}	(0.9,0.999)	90.79 ± 1.91
2	32	2.5×10^{-3}	True	1×10^{-4}	(0.8,0.98)	90.60 ± 1.30
3	32	5×10^{-3}	True	1×10^{-4}	(0.8,0.98)	90.56 ± 1.67
4	64	5×10^{-3}	True	0	(0.9,0.999)	90.52 ± 1.79
5	32	5×10^{-3}	True	0	(0.8,0.98)	90.25 ± 1.45
6	64	2.5×10^{-3}	False	0	(0.9,0.999)	90.23 ± 1.49
7	32	2.5×10^{-3}	False	0	(0.8,0.98)	90.15 ± 1.88
8	32	5×10^{-3}	True	1×10^{-3}	(0.9,0.999)	90.06 ± 1.91
9	32	2.5×10^{-3}	False	1×10^{-4}	(0.9,0.999)	89.92 ± 1.93
10	64	5×10^{-3}	False	1×10^{-4}	(0.8,0.98)	89.83 ± 2.02
11	32	5×10^{-3}	True	1×10^{-3}	(0.8,0.98)	89.83 ± 1.61
12	32	2.5×10^{-3}	True	1×10^{-3}	(0.9,0.999)	89.73 ± 1.97
13	32	5×10^{-3}	True	0	(0.9,0.999)	89.67 ± 1.54
14	32	2.5×10^{-3}	False	0	(0.9,0.999)	89.63 ± 2.42
15	32	1×10^{-3}	False	0	(0.8,0.98)	89.60 ± 1.59
16	64	2.5×10^{-3}	True	1×10^{-3}	(0.9,0.999)	89.60 ± 1.59
17	32	5×10^{-3}	True	1×10^{-4}	(0.9,0.999)	89.50 ± 1.48
18	64	2.5×10^{-3}	False	1×10^{-4}	(0.8,0.98)	89.50 ± 1.74
19	64	5×10^{-3}	True	1×10^{-3}	(0.8,0.98)	89.42 ± 1.64
20	64	5×10^{-3}	False	0	(0.9,0.999)	89.38 ± 1.88
21	32	1×10^{-3}	True	0	(0.8,0.98)	89.35 ± 1.56
22	32	5×10^{-3}	True	1×10^{-4}	(0.9,0.999)	89.31 ± 1.42
23	64	2.5×10^{-3}	True	1×10^{-4}	(0.8,0.98)	89.17 ± 1.73
24	32	2.5×10^{-3}	True	0	(0.8,0.98)	89.12 ± 1.88
25	32	7.5×10^{-4}	True	0	(0.8,0.98)	88.08 ± 1.87
26	64	2.5×10^{-3}	False	1×10^{-3}	(0.9,0.999)	88.02 ± 1.79
27	32	7.5×10^{-4}	True	0	(0.9,0.999)	88.94 ± 1.87
28	64	5×10^{-3}	False	0	(0.8,0.98)	88.92 ± 1.81
29	32	2.5×10^{-3}	False	1×10^{-3}	(0.9,0.999)	88.92 ± 1.21
30	64	1×10^{-3}	True	1×10^{-4}	(0.8,0.98)	88.87 ± 2.32
31	32	1×10^{-3}	False	0	(0.9,0.999)	88.85 ± 1.68
32	32	1×10^{-3}	True	0	(0.9,0.999)	88.73 ± 1.51
33	64	5×10^{-3}	False	1×10^{-4}	(0.8,0.98)	88.71 ± 1.93
34	32	2.5×10^{-3}	True	1×10^{-4}	(0.8,0.98)	88.60 ± 1.75
35	64	5×10^{-3}	True	1×10^{-3}	(0.9,0.999)	88.60 ± 1.54
36	64	1×10^{-3}	False	0	(0.9,0.999)	88.58 ± 2.96
37	64	7.5×10^{-4}	True	1×10^{-4}	(0.9,0.999)	88.56 ± 1.95
38	64	7.5×10^{-4}	False	1×10^{-3}	(0.9,0.999)	88.54 ± 1.92
39	64	2.5×10^{-3}	False	1×10^{-4}	(0.9,0.999)	88.50 ± 2.25
40	32	5×10^{-3}	False	1×10^{-3}	(0.9,0.999)	88.48 ± 1.87

Table A.1: Hyperparameter configurations ranked 1-40 from 5-fold Cross-Validation, ordered by mean validation accuracy μ.

Rank	Batch size	LR	Scheduler use?	Weight decay	β_1, β_2	μ (Acc.%)
41	64	7.5×10^{-4}	False	0	(0.9,0.999)	88.46 ± 1.68
42	32	2.5×10^{-3}	True	1×10^{-4}	(0.9,0.999)	88.44 ± 1.99
43	32	5×10^{-3}	False	0	(0.8,0.98)	88.37 ± 1.77
44	32	7.5×10^{-4}	False	1×10^{-4}	(0.8,0.98)	88.35 ± 1.94
45	64	2.5×10^{-3}	True	0	(0.8,0.98)	88.33 ± 1.83
46	64	2.5×10^{-3}	True	1×10^{-4}	(0.8,0.98)	88.33 ± 1.77
47	32	1×10^{-3}	False	1×10^{-4}	(0.9,0.999)	88.31 ± 1.80
48	32	2.5×10^{-3}	True	1×10^{-3}	(0.8,0.98)	88.27 ± 1.79
49	64	2.5×10^{-3}	False	0	(0.8,0.98)	88.27 ± 3.46
50	32	7.5×10^{-4}	True	0	(0.8,0.98)	88.25 ± 1.21
51	64	2.5×10^{-3}	True	0	(0.9,0.999)	88.23 ± 2.60
52	64	5×10^{-3}	False	0	(0.8,0.98)	88.23 ± 3.00
53	32	2.5×10^{-3}	False	1×10^{-3}	(0.8,0.98)	88.21 ± 1.69
54	32	7.5×10^{-4}	False	0	(0.9,0.999)	88.19 ± 1.85
55	64	5×10^{-3}	False	1×10^{-4}	(0.8,0.98)	88.17 ± 2.78
56	32	5×10^{-3}	False	1×10^{-3}	(0.9,0.999)	88.15 ± 1.90
57	32	2.5×10^{-3}	True	1×10^{-4}	(0.9,0.999)	88.15 ± 1.85
58	64	1×10^{-3}	False	1×10^{-4}	(0.8,0.98)	88.13 ± 3.28
59	32	1×10^{-3}	False	0	(0.9,0.999)	88.10 ± 1.85
60	32	1×10^{-3}	True	0	(0.9,0.999)	88.00 ± 1.69
61	64	5×10^{-3}	False	1×10^{-4}	(0.9,0.999)	88.00 ± 1.77
62	32	2.5×10^{-3}	True	1×10^{-4}	(0.8,0.98)	87.94 ± 1.75
63	64	5×10^{-3}	True	1×10^{-3}	(0.8,0.98)	87.94 ± 1.54
64	64	1×10^{-3}	Ture	0	(0.8,0.98)	87.87 ± 2.52
65	64	7.5×10^{-4}	False	1×10^{-4}	(0.8,0.98)	87.85 ± 1.97
66	64	7.5×10^{-4}	False	1×10^{-3}	(0.9,0.999)	87.81 ± 2.01
67	64	2.5×10^{-3}	False	1×10^{-4}	(0.9,0.999)	87.81 ± 3.09
68	32	5×10^{-3}	False	1×10^{-3}	(0.8,0.98)	87.75 ± 1.94
69	64	7.5×10^{-4}	False	0	(0.9,0.999)	87.73 ± 1.95
70	32	2.5×10^{-3}	True	1×10^{-4}	(0.9,0.999)	87.71 ± 2.13
71	32	5×10^{-3}	False	0	(0.9,0.999)	87.71 ± 2.29
72	32	7.5×10^{-4}	False	1×10^{-4}	(0.9,0.999)	87.69 ± 2.16
73	64	2.5×10^{-3}	True	0	(0.8,0.98)	87.60 ± 2.01
74	64	2.5×10^{-3}	True	1×10^{-4}	(0.8,0.98)	87.54 ± 1.91
75	32	1×10^{-3}	False	1×10^{-4}	(0.9,0.999)	87.27 ± 2.03
76	32	2.5×10^{-3}	True	1×10^{-3}	(0.8,0.98)	87.15 ± 1.82
77	64	1×10^{-3}	False	0	(0.9,0.999)	87.02 ± 2.58
78	64	7.5×10^{-4}	True	0	(0.8,0.98)	86.96 ± 2.32
79	64	1×10^{-3}	True	1×10^{-4}	(0.9,0.999)	86.92 ± 2.42
80	32	1×10^{-3}	True	1×10^{-3}	(0.9,0.999)	86.92 ± 2.12
81	32	7.5×10^{-4}	False	1×10^{-4}	(0.8,0.98)	86.75 ± 1.98
82	64	1×10^{-3}	True	0	(0.8,0.98)	86.63 ± 2.32
83	64	7.5×10^{-4}	False	0	(0.8,0.98)	86.62 ± 2.17
84	32	7.5×10^{-4}	True	0	(0.9,0.999)	86.17 ± 2.21
85	64	1×10^{-3}	False	1×10^{-4}	(0.9,0.999)	86.04 ± 2.49
86	32	5×10^{-3}	False	1×10^{-4}	(0.8,0.98)	85.96 ± 2.34
87	64	7.5×10^{-4}	True	1×10^{-4}	(0.8,0.98)	85.79 ± 2.47
88	64	7.5×10^{-4}	True	1×10^{-3}	(0.9,0.999)	85.67 ± 2.38
89	32	1×10^{-3}	False	1×10^{-4}	(0.8,0.98)	85.60 ± 2.45
90	64	1×10^{-3}	True	0	(0.8,0.98)	85.58 ± 2.50
91	32	7.5×10^{-4}	False	0	(0.9,0.999)	85.42 ± 2.51
92	64	7.5×10^{-4}	False	1×10^{-4}	(0.8,0.98)	84.69 ± 1.96
93	64	7.5×10^{-4}	False	1×10^{-3}	(0.8,0.98)	84.54 ± 2.22
94	32	1×10^{-3}	True	0	(0.9,0.999)	84.46 ± 2.54
95	64	7.5×10^{-4}	True	0	(0.8,0.98)	84.35 ± 1.94
96	32	7.5×10^{-4}	True	1×10^{-3}	(0.8,0.98)	84.17 ± 2.27

Table A.2: Hyperparameter configurations ranked 41–96 from 5-fold cross validation, ordered by mean validation accuracy μ.

Appendix B

Codes

This chapter presents all the codes used for processing and classifying the data.

Preprocessing

```
# Applies z-normalization, BPF, notch filtering to raw EEG data
# %%
import os
import glob
import time
import torch
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.signal import butter, filtfilt, iirnotch
from sklearn.model_selection import train_test_split

# %%
# set to false to not plot the signals
plot_data = True

# put False to avoid the Linear Surrogating Technique for data augmentation #
use_surrogate = False

# Data loading
path = "D:\\Student_Projects\\Thesis Gioele\\00_04_RAW EEG STRESS DATASET"
relax_files = glob.glob(os.path.join(path, "Relax", "*.csv"))
stroop_files = glob.glob(os.path.join(path, "Stroop", "*.csv"))
arithmetic_files = glob.glob(os.path.join(path, "Arithmetic", "*.csv"))
mirror_files = glob.glob(os.path.join(path, "Mirror_image", "*.csv"))

# loading CSV files into a list of DataFrames
def load_csv_files(file_list):
    data_list = []
    for file in file_list:
        df = pd.read_csv(file)
        df = df.drop(df.columns[0], axis=1) # Remove the first column (index
    of timeframes)
        data_list.append(df.values)
    return data_list

```

```
relax_data = load_csv_files(relax_files)
stroop_data = load_csv_files(stroop_files)
arithmetic_data = load_csv_files(arithmetic_files)
mirror_data = load_csv_files(mirror_files)

fs = 128

start_filt = time.time()

# z-normalisation
def z_normalization(data_list):
    for i in range(len(data_list)):
        data_list[i] = (data_list[i] - data_list[i].mean()) / np.std(data_list
    [i]) # subtract avg, divide by std
    return data_list

relax_data_avg = z_normalization(relax_data.copy())
stroop_data_avg = z_normalization(stroop_data.copy())
arithmetic_data_avg = z_normalization(arithmetic_data.copy())
mirror_data_avg = z_normalization(mirror_data.copy())

# %%  Filtering
def bandpass_filter(data, lowcut=1, highcut=40, fs=128, order=4):
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = butter(order, [low, high], btype='band')
    return filtfilt(b, a, data, axis=0)

def notch_filter(data, fs=128, freq=50, quality_factor=30):
    b, a = iirnotch(freq, quality_factor, fs)
    return filtfilt(b, a, data, axis=0)

def preprocess_data(data_list): # passband + notch filter
    return [notch_filter(bandpass_filter(df, lowcut=1, highcut=40, fs=fs), fs=
    fs) for df in data_list]

relax_data_filt = np.stack(preprocess_data(relax_data_avg), axis=0)
stroop_data_filt = np.stack(preprocess_data(stroop_data_avg), axis=0)
arithmetic_data_filt = np.stack(preprocess_data(arithmetic_data_avg), axis=0)
mirror_data_filt = np.stack(preprocess_data(mirror_data_avg), axis=0)

end_filt = time.time()
print("Data filtered in ", end_filt-start_filt, "seconds")

# %% Data Augmentation using Linear Surrogating Technique
if use_surrogate:
    def linear_surrogate(signal_tensor):
        fft_signal = torch.fft.fft(signal_tensor)
        magnitude = torch.abs(fft_signal)          # Module
        phase = torch.angle(fft_signal)              # Original phase
        random_phase = torch.rand_like(phase) * 2 * torch.pi   # New random
    phase
        fft_surrogate = magnitude * torch.exp(1j * random_phase)   # New FFT
    with the random phase
        surrogate_signal = torch.fft.ifft(fft_surrogate).real      # IFFT (take
     the real part)
        return surrogate_signal

    def augment_linear_surrogating(data_list):
        augmented_data = []
```

```python
        for subject_array in data_list:
            surrogate_array = np.zeros_like(subject_array)

            for ch in range(subject_array.shape[1]):
                signal_tensor = torch.tensor(subject_array[:, ch].copy(), dtype=torch.float32)
                surrogate_signal = linear_surrogate(signal_tensor)
                surrogate_array[:, ch] = surrogate_signal.numpy()
            augmented_data.append(surrogate_array)
        return augmented_data

    # Splitting into training and testing sets, then applying data augmentation
    def split_and_augment(data):
        train_data, test_data = train_test_split(data, test_size=0.15, random_state=42)
        train_augmented = augment_linear_surrogating(train_data)
        train_doubled = np.concatenate((np.array(train_data), np.array(train_augmented)), axis=0)
        return train_doubled, test_data

    relax_train_doubled, relax_test = split_and_augment(relax_data_filt)
    stroop_train_doubled, stroop_test = split_and_augment(stroop_data_filt)
    arithmetic_train_doubled, arithmetic_test = split_and_augment(arithmetic_data_filt)
    mirror_train_doubled, mirror_test = split_and_augment(mirror_data_filt)

    relax_data_filt = relax_train_doubled
    stroop_data_filt = stroop_train_doubled
    arithmetic_data_filt = arithmetic_train_doubled
    mirror_data_filt = mirror_train_doubled

    relax_all = np.concatenate((relax_data_filt, relax_test), axis=0)
    stroop_all = np.concatenate((stroop_data_filt, stroop_test), axis=0)
    arithmetic_all = np.concatenate((arithmetic_data_filt, arithmetic_test), axis=0)
    mirror_all = np.concatenate((mirror_data_filt, mirror_test), axis=0)

else: # no data augmentation  (full dataset)
    relax_all = relax_data_filt.copy()
    stroop_all = stroop_data_filt.copy()
    arithmetic_all = arithmetic_data_filt.copy()
    mirror_all = mirror_data_filt.copy()

    # train - test splitting
    relax_data_filt, relax_test = train_test_split(relax_data_filt, test_size=0.15, random_state=42)
    stroop_data_filt, stroop_test = train_test_split(stroop_data_filt, test_size=0.15, random_state=42)
    arithmetic_data_filt, arithmetic_test = train_test_split(arithmetic_data_filt, test_size=0.15, random_state=42)
    mirror_data_filt, mirror_test = train_test_split(mirror_data_filt, test_size=0.15, random_state=42)
```

ICA

```
# Apply ICA to  filtered EEG Data
# %%
import time
import torch
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import kurtosis
from sklearn.decomposition import FastICA
from scipy.signal import butter, filtfilt, find_peaks

# %%
# set to True if you want the plot
plot_signal = True

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# DATA LOADING
folder_path = "D:\\Student_Projects\\Thesis Gioele\\Codes\\Data\\00\\00_"
end_path = "_zNorm_full_filtered_aug.npy"

# Load the filtered data (full dataset)
relax_data = np.load(folder_path + "relax" + end_path, allow_pickle=True)
stroop_data = np.load(folder_path + "stroop" + end_path, allow_pickle=True)
arithmetic_data = np.load(folder_path + "arithmetic" + end_path, allow_pickle=
    True)
mirror_data = np.load(folder_path + "mirror" + end_path, allow_pickle=True)

fs = 128

# %%
start_ica = time.time()

def clean_signal_ica(raw_data: np.ndarray,
                     fs: float,   # Hz
                     segment_duration: float = 2.0,  # s
                     peak_rate_max: float = 2,  # Hz = 2 blinks per second
                     kurtosis_thresh: float = 12.0,
                     ica_n_components: int = None) -> np.ndarray:
    """
    Perform ICA-based cleanup of multi-channel signal.

    Input shape must be (timeframes x channels). If the input shape is (
    channels x timeframes),
    it is automatically transposed.
    """
    transposed = False
    if raw_data.shape[0] < raw_data.shape[1]:
        raw_data = raw_data.T
        transposed = True

    n_samples, n_channels = raw_data.shape

    if ica_n_components is None:
        ica_n_components = n_channels

    samples_per_segment = int(segment_duration * fs)
```

```
    n_segments = int(np.ceil(n_samples / samples_per_segment))
    cleaned = np.zeros_like(raw_data)

    for seg_idx in range(n_segments):
        start = seg_idx * samples_per_segment
        end = min(start + samples_per_segment, n_samples)
        segment = raw_data[start:end, :].T  # shape (channels, segment_length)

        # ICA decomposition
        ica = FastICA(n_components=ica_n_components, random_state=0)
        sources = ica.fit_transform(segment.T).T  # (n_components, n_time)

        # Check how many iterations were used
        if ica.n_iter_ >= 200: # 200 is the default max_iter in FastICA. If it reaches this threshold, it means it did not converge
            print(f"Segment {seg_idx}: ICA did not converge (n_iter_ = {ica.n_iter_}). Skipping artifact removal.")
            cleaned[start:end, :] = raw_data[start:end, :]
            continue

        else: # ICA converged, proceed with the artifact removal
            # collect dropped components and their criteria
            drops = []  # list of (component_idx, criterion)

            for ic in range(sources.shape[0]):
                src = sources[ic]

                # Mid-band peak-rate
                nyq = 0.5 * fs
                b, a = butter(4, 15 / nyq, btype='low')
                filt = filtfilt(b, a, src) # filtering the component to remove multiple peaks
                threshold = 0.5 * np.max(np.abs(filt))
                peaks, _ = find_peaks(np.abs(filt), height=threshold)
                rate = len(peaks) / (filt.size / fs)

                # Kurtosis artifact
                k = abs(kurtosis(src))
                if k > kurtosis_thresh and rate < peak_rate_max:
                    drops.append((ic, 'Kurtosis'))

            # Zero-out dropped components and reconstruct
            if drops:
                drop_indices = [ic for ic, _ in drops]
                sources[drop_indices] = 0
            recon = ica.inverse_transform(sources.T).T  # shape (channels, time)
            cleaned[start:end, :] = recon.T  # shape (time, channels)

    if transposed:
        cleaned = cleaned.T

    return cleaned

relax_data_ica = [clean_signal_ica(subject_data, fs=fs) for subject_data in relax_data]
stroop_data_ica = [clean_signal_ica(subject_data, fs=fs) for subject_data in stroop_data]
arithmetic_data_ica = [clean_signal_ica(subject_data, fs=fs) for subject_data in arithmetic_data]
```

```
mirror_data_ica = [clean_signal_ica(subject_data, fs=fs) for subject_data in
    mirror_data]

end_ica = time.time()
print("ICA total time (full dataset): ", end_ica - start_ica)
```

Training and Testing the EEGNet on the SAM40 Dataset

```python
# %%
import time
import torch
import numpy as np
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
from torchmetrics import Accuracy
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import train_test_split

# %% LOADING DATA
plot_results = False

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

folder_path = "D:\\Student_Projects\\Thesis Gioele\\Codes\\Data\\01\\01_"
end_path = "_ica_v0.npy"

relax_data = np.load(folder_path + "relax" + end_path, allow_pickle=True)
stroop_data = np.load(folder_path + "stroop" + end_path, allow_pickle=True)

channels_32 = True
if channels_32: num_channels = relax_data[0].shape[1]
else: # to train the model for testing on the OpenBCI data, which has 16 channels
    num_channels = 16
    idx = [i - 1 for i in [3, 32, 5, 30, 2, 7, 28, 1, 13, 22, 17, 16, 19, 14, 25, 10]] # channels similar to the channels in the OpenBCI device
    relax_data = [sub[:, idx] for sub in relax_data]
    stroop_data = [sub[:, idx] for sub in stroop_data]

# %% data segmentation and division in train, val, test set.
# Divide each trial into 15 segments of 5 seconds each (6400 samples). Then split these 5-second blocks into train, val, test sets.
relax_data_divided = []
stroop_data_divided = []
num_segments = 15
len_segment = len(relax_data[0]) // num_segments # 5 seconds = 6400 samples

for sub in relax_data: # for each sub-> 9600x32
    for i in range(num_segments):
        relax_data_divided.append(sub[i*len_segment:(i+1)*len_segment, :] )

for sub in stroop_data:
    for i in range(num_segments):
        stroop_data_divided.append(sub[i*len_segment:(i+1)*len_segment, :])

tot_divided = np.stack(relax_data_divided + stroop_data_divided, axis=0)
all_labels  = np.array([0]*len(relax_data_divided) + [1]*len(stroop_data_divided))

# divide in construction and test set
trials_train_val, trials_test, labels_train_val, labels_test = train_test_split(
    tot_divided, all_labels,
```

```
    test_size=0.15, random_state=42,
    stratify=all_labels
)

fs = 128  # sampling frequency
batch_size = 32
print("Batch size:", batch_size)
num_class = 2

def segment_data_trials(data, trial_labels, segment_length, step):
    segments, labels = [], []
    for trial, lab in zip(data, trial_labels):
        for start in range(0, trial.shape[0] - segment_length + 1, step):
            seg = trial[start:start+segment_length, :].T  # (channels, time)
            segments.append(seg)
            labels.append(lab)
    return np.stack(segments), np.array(labels)


factor = 2
segment_length = factor * fs
print("Segment length:", factor, "seconds")
step_train_val = int(0.5 * segment_length) # overlap = 1 - step_%
step_test      = int(1 * segment_length) # putting overlap in the test will
    only generate more samples for the test set, but not more information
print("Train Overlap:", 1 - (step_train_val / segment_length))

segments_tv, labs_tv = segment_data_trials(trials_train_val, labels_train_val,
     segment_length, step_train_val)
segments_test, labs_test = segment_data_trials(trials_test,       labels_test,
     segment_length, step_test)

# 3) Shuffle-and-split train+val segments into train and val sets
seg_train, seg_val, lab_train, lab_val = train_test_split(
    segments_tv, labs_tv,
    test_size=0.15, random_state=42,
    stratify=labs_tv
)

# 4) Convert to torch tensors and create DataLoaders;  add channel dimension
    with unsqueeze(1)
signal_train = torch.tensor(seg_train, dtype=torch.float32).unsqueeze(1)
signal_val   = torch.tensor(seg_val,   dtype=torch.float32).unsqueeze(1)
signal_test  = torch.tensor(segments_test, dtype=torch.float32).unsqueeze(1)

label_train = torch.tensor(lab_train, dtype=torch.long)
label_val   = torch.tensor(lab_val,   dtype=torch.long)
label_test  = torch.tensor(labs_test, dtype=torch.long)

train_loader = DataLoader(TensorDataset(signal_train, label_train), batch_size
    =batch_size, shuffle=True)
val_loader   = DataLoader(TensorDataset(signal_val,   label_val),   batch_size
    =batch_size, shuffle=False)
test_loader  = DataLoader(TensorDataset(signal_test,  label_test),  batch_size
    =batch_size, shuffle=False)

# EEGNet Model inspired from https://github.com/aliasvishnu/EEGNet/blob/master
    /EEGNet-PyTorch.ipynb

class EEGNet(nn.Module):
    def __init__(self, num_channels, segment_length, num_class=2):
```

```
        super().__init__()

        self.num_channels = num_channels
        self.segment_length = segment_length

        # Layer 1
        self.conv2d = nn.Conv2d(1, 16, kernel_size=(1, 64), padding=0)
        self.bn1    = nn.BatchNorm2d(16)
        self.elu    = nn.ELU()
        self.drop   = nn.Dropout(0.25)

        # Layer 2
        self.pad1       = nn.ZeroPad2d((16, 17, 0, 1))
        self.depth_conv = nn.Conv2d(16, 4, kernel_size=(2, 32))
        self.bn2        = nn.BatchNorm2d(4)
        self.pool1      = nn.AvgPool2d((2, 4))

        # Layer 3
        self.pad2       = nn.ZeroPad2d((2, 1, 4, 3))
        self.sep_dep    = nn.Conv2d(4, 4, kernel_size=(8, 4), groups=4)
        self.sep_point  = nn.Conv2d(4, 4, kernel_size=1)
        self.bn3        = nn.BatchNorm2d(4)
        self.pool2      = nn.AvgPool2d((2, 4))

        # use a dummy pass to infer flattened feature size
        with torch.no_grad():
            dummy = torch.zeros(1, 1, num_channels, segment_length)
            x = self._forward_features(dummy)
            n_features = x.shape[1]

        # final classifier
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(n_features, num_class),
            nn.Softmax(dim=1)
        )

    def _forward_features(self, x):
        x = self.conv2d(x)
        x = self.bn1(x);  x = self.elu(x);  x = self.drop(x)

        x = self.pad1(x)
        x = self.depth_conv(x)
        x = self.bn2(x);  x = self.elu(x);  x = self.drop(x)
        x = self.pool1(x)

        x = self.pad2(x)
        x = self.sep_dep(x)
        x = self.sep_point(x)
        x = self.bn3(x);  x = self.elu(x);  x = self.drop(x)
        x = self.pool2(x)

        x = torch.flatten(x, 1)
        return x

    def forward(self, x):
        x = self._forward_features(x)
        x = self.classifier(x)
        return x
```

```
model = EEGNet(num_channels=num_channels, segment_length=segment_length).to(
    device)

# Functions for train and val
class AverageMeter(object):
    """Computes and stores the average and current value"""
    def __init__(self):
        self.reset()

    def reset(self):
        self.val = 0
        self.avg = 0
        self.sum = 0
        self.count = 0

    def update(self, val, n=1):
        self.val = val
        self.sum += val * n
        self.count += n
        self.avg = self.sum / self.count

def train_one_epoch(model, train_loader, loss_fn, optimizer):
    model.train()
    loss_train = AverageMeter()
    acc_train = Accuracy(task="multiclass", num_classes= num_class).to(device)

    for i, (inputs, targets) in enumerate(train_loader):
        inputs = inputs.to(device)
        targets = targets.to(device)

        outputs = model(inputs)
        loss = loss_fn(outputs, targets)

        loss.backward()
        nn.utils.clip_grad_norm_(model.parameters(), 1)
        optimizer.step()
        optimizer.zero_grad()

        loss_train.update(loss.item())
        acc_train(outputs, targets.int())

    return model, loss_train.avg, acc_train.compute().item()

def validate(model, val_loader, loss_fn):
    model.eval()
    loss_val = AverageMeter()
    acc_val = Accuracy(task="multiclass", num_classes=num_class).to(device)

    with torch.no_grad():
        for inputs, targets in val_loader:
            inputs = inputs.to(device)
            targets = targets.to(device)

            outputs = model(inputs)
            loss = loss_fn(outputs, targets)

            loss_val.update(loss.item())
            acc_val(outputs, targets.int())

    return loss_val.avg, acc_val.compute().item()

```

```
loss_train = []
acc_train = []
loss_val = []
acc_val = []

num_epochs = 101
tot_epochs = num_epochs
loss_fn= nn.CrossEntropyLoss().to(device)
optimizer = optim.Adam(model.parameters(), lr=0.005, weight_decay=1e-3, betas
    =(0.9,0.999))
use_scheduler = True
if use_scheduler: scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(
    optimizer, T_max=num_epochs)

best_val_acc = 0.0
if channels_32: model_path = "D:\\Student_Projects\\Thesis Gioele\\Codes//Data
    //04//04_model_2cl_EEGNet.pt"
else: model_path = "D:\\Student_Projects\\Thesis Gioele\\Codes//Data//04//04
    _model_2cl_EEGNet_16.pt" # 16 channels

patience = 20  # Max epochs without val_acc improvement before early stopping
counter = 0  # Counts the epochs without any improvement

start_train = time.time()
for epoch in range(num_epochs):
    model, loss_train_ep, acc_train_ep = train_one_epoch(model, train_loader,
    loss_fn, optimizer)

    loss_train.append(loss_train_ep)
    acc_train.append(acc_train_ep)

    loss_val_ep, acc_val_ep = validate(model, val_loader, loss_fn)
    loss_val.append(loss_val_ep)
    acc_val.append(acc_val_ep)

    if acc_val_ep > best_val_acc:
        best_val_acc = acc_val_ep
        torch.save(model.state_dict(), model_path)
        counter = 0
    else:
        counter += 1

    if counter >= patience:
        print(f"Early stopping ({patience} epochs) triggered after {epoch+1}
    epochs.")
        tot_epochs = epoch + 1
        break

    if (epoch % 10 == 5) or (epoch % 10 == 0):
        print(f'epoch {epoch}:')
        print(f' Loss = {loss_train_ep:.4}, Tr Accuracy = {acc_train_ep*100:.2
    f}%, Val Accuracy = {acc_val_ep*100:.2f}% (best_val_acc = {best_val_acc
    *100:.4f}%)\n')

    if use_scheduler: scheduler.step()
end_train = time.time()
print("Train completed in {:.2f} seconds".format(end_train - start_train))

start_test = time.time()
model.load_state_dict(torch.load(model_path, weights_only=False))
test_loss, test_acc = validate(model, test_loader, loss_fn)
```

```
end_test = time.time()
print(f"Test set - Loss: {test_loss:.4f}, Accuracy: {test_acc*100:.2f}%")
print("Test completed in {:.2f} seconds".format(end_test - start_test))

# %% Plot Accuracy and Loss
if plot_results:
    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.plot(range(tot_epochs), loss_train, 'b-', label='Train Loss')
    plt.plot(range(tot_epochs), loss_val, 'r-', label='Val Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Train vs. Val Loss')
    plt.legend()
    plt.grid(True)

    plt.subplot(1, 2, 2)
    plt.plot(range(tot_epochs), acc_train, 'b-', label='Train Accuracy')
    plt.plot(range(tot_epochs), acc_val, 'r-', label='Val Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title('Train vs. Val Accuracy')
    plt.ylim(0.5,1)
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()
```

K-Fold Cross-Validation for the 96 Hyperparameter Combinations

```
# k-fold on: batch size, learning rate, weight decay, betas, use_scheduler

import torch
import numpy as np
import pandas as pd
from torch import nn, optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import KFold
from torchmetrics import Accuracy

# ----- Parameters -----
k_folds = 5
batch_sizes = [32, 64]
learning_rates = [7.5e-4, 1e-3, 2.5e-3, 5e-3]
use_scheduler_opts = [False, True]
weight_decays = [0.0, 1e-4, 1e-3]
betas_opts = [(0.9, 0.999), (0.8, 0.98)]
num_epochs = 90
fs = 128  # sampling frequency
segment_length = 2 * fs
step_train = int(0.5 * segment_length)
step_test = segment_length
num_class = 2

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# ----- Data Loading -----
folder_path = "D:/Student_Projects/Thesis Gioele/Codes/Data/00/00_"
end_path = "_full_filtered.npy"
relax = np.load(folder_path + "relax" + end_path, allow_pickle=True)
stroop = np.load(folder_path + "stroop" + end_path, allow_pickle=True)

# ----- Big segmentation -----
num_segments = 15
big_chunks, big_labels = [], []
for sub in relax:
    for i in range(num_segments):
        big_chunks.append(sub[i*(len(sub)//num_segments):(i+1)*(len(sub)//
    num_segments), :])
        big_labels.append(0)
for sub in stroop:
    for i in range(num_segments):
        big_chunks.append(sub[i*(len(sub)//num_segments):(i+1)*(len(sub)//
    num_segments), :])
        big_labels.append(1)
big_chunks = np.stack(big_chunks)
big_labels = np.array(big_labels)

def segment_windows(data, labels, seg_len, step):
    segs, labs = [], []
    for trial, lab in zip(data, labels):
        for start in range(0, trial.shape[0] - seg_len + 1, step):
            segs.append(trial[start:start+seg_len, :].T)
            labs.append(lab)
```

```
    return np.stack(segs), np.array(labs)

all_segs, all_labs = segment_windows(big_chunks, big_labels, segment_length,
    step_train)

# Precompute chunk index mapping
chunk_idxs = []
idx = 0
for _ in range(len(big_chunks)):
    count = (big_chunks.shape[1] - segment_length) // step_train + 1
    chunk_idxs.append(list(range(idx, idx+count)))
    idx += count

# ----- Model Definition -----
class EEGNet(nn.Module):
    def __init__(self, num_class=2):
        super(EEGNet, self).__init__()

        # Layer 1
        self.conv2d = nn.Conv2d(1, 16, kernel_size=(1, 64), padding=0)
        self.Batch_normalization_1 = nn.BatchNorm2d(16)
        self.Elu = nn.ELU()
        self.Dropout = nn.Dropout(0.25)

        # Layer 2
        self.padding1 = nn.ZeroPad2d((16, 17, 0, 1))
        self.Depthwise_conv2D = nn.Conv2d(16, 4, kernel_size=(2, 32))
        self.Batch_normalization_2 = nn.BatchNorm2d(4)
        self.Average_pooling2D_1 = nn.AvgPool2d(kernel_size=(2, 4))

        # Layer 3
        self.padding2 = nn.ZeroPad2d((2, 1, 4, 3))
        self.Separable_conv2D_depth = nn.Conv2d(4, 4, kernel_size=(8, 4),
    padding=0, groups=4)
        self.Separable_conv2D_point = nn.Conv2d(4, 4, kernel_size=(1, 1))
        self.Batch_normalization_3 = nn.BatchNorm2d(4)
        self.Average_pooling2D_2 = nn.AvgPool2d(kernel_size=(2, 4))

        # Layer 4
        self.Flatten = nn.Flatten()
        self.Dense = nn.Linear(384, num_class) # 384 for 2 seconds segment
        self.Softmax = nn.Softmax(dim=1)

    def forward(self, x):
        # Layer 1
        x = self.conv2d(x)
        x = self.Batch_normalization_1(x)
        x = self.Elu(x)
        x = self.Dropout(x)

        # Layer 2
        x = self.padding1(x)
        x = self.Depthwise_conv2D(x)
        x = self.Batch_normalization_2(x)
        x = self.Elu(x)
        x = self.Dropout(x)
        x = self.Average_pooling2D_1(x)

        # Layer 3
        x = self.padding2(x)
        x = self.Separable_conv2D_depth(x)
```

```
        x = self.Separable_conv2D_point(x)
        x = self.Batch_normalization_3(x)
        x = self.Elu(x)
        x = self.Dropout(x)
        x = self.Average_pooling2D_2(x)

        # Layer 4
        x = self.Flatten(x)
        x = self.Dense(x)
        x = self.Softmax(x)
        return x

# ----- Experiment Loop -----
results = []
kf = KFold(n_splits=k_folds, shuffle=True)

for bs in batch_sizes:
    for lr in learning_rates:
        for wd in weight_decays:
            for betas in betas_opts:
                for use_scheduler in use_scheduler_opts:
                    fold_accs = []
                    print(f"Config: BS={bs}, LR={lr}, WD={wd}, Betas={betas}, Scheduler={use_scheduler}")
                    for fold, (train_chunks, test_chunks) in enumerate(kf.split(big_chunks), 1):
                        # build indices
                        train_idx = [i for fc in train_chunks for i in chunk_idxs[fc]]
                        test_idx  = [i for fc in test_chunks  for i in chunk_idxs[fc]]
                        X_train = torch.tensor(all_segs[train_idx], dtype=torch.float32).unsqueeze(1).to(device)
                        y_train = torch.tensor(all_labs[train_idx], dtype=torch.long).to(device)
                        X_test  = torch.tensor(all_segs[test_idx], dtype=torch.float32).unsqueeze(1).to(device)
                        y_test  = torch.tensor(all_labs[test_idx], dtype=torch.long).to(device)

                        train_loader = DataLoader(TensorDataset(X_train, y_train), batch_size=bs, shuffle=True)
                        test_loader  = DataLoader(TensorDataset(X_test, y_test),  batch_size=bs, shuffle=False)

                        model = EEGNet(num_class).to(device)
                        optimizer = optim.Adam(model.parameters(), lr=lr, betas=betas, weight_decay=wd)
                        if use_scheduler:
                            scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=num_epochs)
                        loss_fn = nn.CrossEntropyLoss()

                        # train
                        for epoch in range(1, num_epochs+1):
                            model.train()
                            for inputs, labs in train_loader:
                                outputs = model(inputs)
                                loss = loss_fn(outputs, labs)
                                optimizer.zero_grad(); loss.backward(); optimizer.step()
```

```
                            if use_scheduler:
                                scheduler.step()

                        # eval
                        model.eval()
                        acc_metric = Accuracy(task="multiclass", num_classes=num_class).to(device)
                        with torch.no_grad():
                            for inputs, labs in test_loader:
                                preds = model(inputs)
                                acc_metric(preds, labs)
                        acc = acc_metric.compute().item()
                        print(f"  Fold {fold} accuracy: {acc*100:.2f}%")
                        fold_accs.append(acc)

                    avg_acc = np.mean(fold_accs)
                    results.append({
                        'batch_size': bs,
                        'lr': lr,
                        'weight_decay': wd,
                        'betas': betas,
                        'use_scheduler': use_scheduler,
                        'fold_accuracies': fold_accs,
                        'avg_test_acc': avg_acc
                    })
                    print(f"-> Avg acc: {avg_acc*100:.2f}%\n")

# ----- Save Results -----
df = pd.DataFrame(results)
df = df.sort_values('avg_test_acc', ascending=False).reset_index(drop=True)
print("\n=== BEST CONFIG ===")
print(df.iloc[0])
df.to_csv("kfold_results.csv", index=False)
print("\nResults saved to kfold_results.csv")
```

K-Fold Cross-Validation for the Segment Length

```python
# k-fold on the segment_length
import torch
import numpy as np
import pandas as pd
import torch.nn as nn
import torch.optim as optim
from torchmetrics import Accuracy
from sklearn.model_selection import KFold
from torch.utils.data import DataLoader, TensorDataset

# Configuration
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
fs = 128  # sampling frequency
batch_size = 64
num_class = 2
num_folds = 5
segment_factors = [1, 2, 3, 4, 5] # signal segment lengths in seconds
num_epochs = 80

# Data paths
folder_path = "D:/Student_Projects/Thesis Gioele/Codes/Data/00/00_"
end_path = "_zNorm_full_filtered.npy"
relax_data = np.load(folder_path + "relax" + end_path, allow_pickle=True)
stroop_data = np.load(folder_path + "stroop" + end_path, allow_pickle=True)

# Prepare trials by splitting each subject into folds
relax_trials = [seg for sub in relax_data for seg in np.array_split(sub, num_folds)]
stroop_trials = [seg for sub in stroop_data for seg in np.array_split(sub, num_folds)]
all_trials = np.array(relax_trials + stroop_trials)
all_labels = np.array([0]*len(relax_trials) + [1]*len(stroop_trials))

kf = KFold(n_splits=num_folds, shuffle=True) # divide into 5 folds

# EEGNet definition
class EEGNet(nn.Module):
    def __init__(self, num_channels, segment_length, num_class=2):
        super().__init__()

        self.num_channels = num_channels
        self.segment_length = segment_length

        # Layer 1
        self.conv2d = nn.Conv2d(1, 16, kernel_size=(1, 64), padding=0)
        self.bn1    = nn.BatchNorm2d(16)
        self.elu    = nn.ELU()
        self.drop   = nn.Dropout(0.25)

        # Layer 2
        self.pad1       = nn.ZeroPad2d((16, 17, 0, 1))
        self.depth_conv = nn.Conv2d(16, 4, kernel_size=(2, 32))
        self.bn2        = nn.BatchNorm2d(4)
        self.pool1      = nn.AvgPool2d((2, 4))

        # Layer 3
```

```
        self.pad2       = nn.ZeroPad2d((2, 1, 4, 3))
        self.sep_dep    = nn.Conv2d(4, 4, kernel_size=(8, 4), groups=4)
        self.sep_point  = nn.Conv2d(4, 4, kernel_size=1)
        self.bn3        = nn.BatchNorm2d(4)
        self.pool2      = nn.AvgPool2d((2, 4))

        # use a dummy pass to infer flattened feature size
        with torch.no_grad():
            dummy = torch.zeros(1, 1, num_channels, segment_length)
            x = self._forward_features(dummy)
            n_features = x.shape[1]

        # final classifier
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(n_features, num_class),
            nn.Softmax(dim=1)
        )

    def _forward_features(self, x):
        x = self.conv2d(x)
        x = self.bn1(x);  x = self.elu(x);  x = self.drop(x)

        x = self.pad1(x)
        x = self.depth_conv(x)
        x = self.bn2(x);  x = self.elu(x);  x = self.drop(x)
        x = self.pool1(x)

        x = self.pad2(x)
        x = self.sep_dep(x)
        x = self.sep_point(x)
        x = self.bn3(x);  x = self.elu(x);  x = self.drop(x)
        x = self.pool2(x)

        x = torch.flatten(x, 1)
        return x

    def forward(self, x):
        x = self._forward_features(x)
        x = self.classifier(x)
        return x


# segment trials into windows
def segment_trials(trials, labels, segment_length, step):
    segs, labs = [], []
    for trial, lab in zip(trials, labels):
        for start in range(0, trial.shape[0] - segment_length + 1, step):
            segs.append(trial[start:start+segment_length, :].T)
            labs.append(lab)
    return np.stack(segs), np.array(labs)

# Main k-fold over segment lengths
results = []
for factor in segment_factors: # factor is the segment length in seconds
    seg_len = factor * fs
    step_train = int(0.5 * seg_len)
    step_test = seg_len
    fold_idx = 0
    for train_idx, test_idx in kf.split(all_trials):
        fold_idx += 1
```

```
        X_train_trials = all_trials[train_idx]
        y_train_trials = all_labels[train_idx]
        X_test_trials = all_trials[test_idx]
        y_test_trials = all_labels[test_idx]

        X_train, y_train = segment_trials(X_train_trials, y_train_trials, seg_len, step_train)
        X_test, y_test = segment_trials(X_test_trials, y_test_trials, seg_len, step_test)

        train_ds = TensorDataset(torch.tensor(X_train, dtype=torch.float32).unsqueeze(1), torch.tensor(y_train, dtype=torch.long))
        test_ds = TensorDataset(torch.tensor(X_test, dtype=torch.float32).unsqueeze(1), torch.tensor(y_test, dtype=torch.long))
        train_loader = DataLoader(train_ds, batch_size=batch_size, shuffle=True)
        test_loader = DataLoader(test_ds, batch_size=batch_size, shuffle=False)

        model = EEGNet(num_channels=all_trials.shape[2], segment_length=seg_len).to(device)
        optimizer = optim.Adam(model.parameters(), lr=0.005, weight_decay=1e-4, betas=(0.9,0.999))
        scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=num_epochs)
        loss_fn = nn.CrossEntropyLoss().to(device)

        # Training loop
        for epoch in range(1, num_epochs+1):
            model.train()
            for inputs, targets in train_loader:
                inputs, targets = inputs.to(device), targets.to(device)
                outputs = model(inputs)
                loss = loss_fn(outputs, targets)
                loss.backward()
                nn.utils.clip_grad_norm_(model.parameters(), 1)
                optimizer.step(); optimizer.zero_grad()
            scheduler.step()

        # Evaluation
        model.eval()
        acc_metric = Accuracy(task="multiclass", num_classes=num_class).to(device)
        with torch.no_grad():
            for inputs, targets in test_loader:
                inputs, targets = inputs.to(device), targets.to(device)
                outputs = model(inputs)
                acc_metric(outputs, targets)
        test_acc = acc_metric.compute().item()
        results.append({"factor_s": factor, "fold": fold_idx, "accuracy": test_acc})
        print("Factor:", factor, "Fold:", fold_idx, "Accuracy:", test_acc)

# Save CSV
import pandas as pd
out_df = pd.DataFrame(results)
out_path = "eegnet_kfold_results.csv"
out_df.to_csv(out_path, index=False)
print(f"Results saved to {out_path}")
```

Training and Testing the EEGNet on the Collected Data

```
import torch
import numpy as np
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
from scipy.stats import kurtosis
from sklearn.decomposition import FastICA
from scipy.signal import butter, filtfilt, iirnotch, resample, find_peaks
from torch.utils.data import ConcatDataset, DataLoader, TensorDataset,
    random_split

# File paths
relax_path_1  = "D:\Student_Projects\Thesis Gioele\Raw_data\sub-Gio\ses-S002\\
    npy\\all_relax.npy"
stroop_path_1 = "D:\Student_Projects\Thesis Gioele\Raw_data\sub-Gio\ses-S003\\
    npy_stress\\all_stroop.npy"
relax_path_2  = "D:\Student_Projects\Thesis Gioele\Raw_data\sub-Gio\ses-S004\\
    npy\\all_relax.npy"
stroop_path_2 = "D:\Student_Projects\Thesis Gioele\Raw_data\sub-Gio\ses-S005\\
    npy_stress\\all_stroop.npy"
dir_r = "D:\Student_Projects\Thesis Gioele\Raw_data\sub-Gio\ses-S007\\
    all_relax.npy"
dir_s = "D:\Student_Projects\Thesis Gioele\Raw_data\sub-Gio\ses-S007\\
    all_stress.npy"
dir_r8 = "D:\Student_Projects\Thesis Gioele\Raw_data\sub-Gio\ses-S008\\
    all_relax.npy"
dir_s8 = "D:\Student_Projects\Thesis Gioele\Raw_data\sub-Gio\ses-S008\\
    all_stroop.npy"
dir_r9 = "D:\Student_Projects\Thesis Gioele\Raw_data\sub-Gio\ses-S009\\
    all_relax.npy"
dir_s9 = "D:\Student_Projects\Thesis Gioele\Raw_data\sub-Gio\ses-S009\\
    all_stroop.npy"

# Load arrays
stroop_data_1 = np.load(stroop_path_1)
relax_data_1  = np.load(relax_path_1)
stroop_data_2 = np.load(stroop_path_2)
relax_data_2  = np.load(relax_path_2)
stroop_data_3 = np.load(dir_s)
relax_data_3  = np.load(dir_r)
stroop_data_4 = np.load(dir_s8)
relax_data_4  = np.load(dir_r8)
stroop_data_5 = np.load(dir_s9)
relax_data_5  = np.load(dir_r9)

# Sampling rate (Hz)
fs_old = 125 # OpenBCI fs
fs_new = 125 # If needed
fs = fs_new

# number of samples for testing
test_samples = 50 * fs_old

# TESTING DATA
stroop_test_raw = stroop_data_5
relax_test_raw  = relax_data_5
```

```python

# 2) Concatenate rest per train+val
stroop_rest = np.concatenate((stroop_data_1, stroop_data_2, stroop_data_3, stroop_data_4), axis=0)
relax_rest  = np.concatenate((relax_data_1, relax_data_2, relax_data_3, relax_data_4), axis=0)

# Preprocessing function (ICA, filtering, resample)
def clean_signal_ica(raw_data: np.ndarray,
                     fs: float,  # Hz
                     segment_duration: float = 2.0,  # s
                     peak_rate_max: float = 2,  # Hz = 2 blinks per second
                     kurtosis_thresh: float = 12.0,
                     ica_n_components: int = None) -> np.ndarray:
    """
    Perform ICA-based cleanup of multi-channel signal.

    Input shape must be (timeframes x channels). If the input shape is (channels x timeframes),
    it is automatically transposed.
    """
    transposed = False
    if raw_data.shape[0] < raw_data.shape[1]:
        raw_data = raw_data.T
        transposed = True

    n_samples, n_channels = raw_data.shape

    if ica_n_components is None:
        ica_n_components = n_channels

    samples_per_segment = int(segment_duration * fs)
    n_segments = int(np.ceil(n_samples / samples_per_segment))
    cleaned = np.zeros_like(raw_data)

    for seg_idx in range(n_segments):
        start = seg_idx * samples_per_segment
        end = min(start + samples_per_segment, n_samples)
        segment = raw_data[start:end, :].T  # shape (channels, segment_length)

        # ICA decomposition
        ica = FastICA(n_components=ica_n_components, random_state=0)
        sources = ica.fit_transform(segment.T).T  # (n_components, n_time)

        # Check how many iterations were used
        if ica.n_iter_ >= 200: # 200 is the default max_iter in FastICA. If it reaches this threshold, it means it did not converge
            print(f"Segment {seg_idx}: ICA did not converge (n_iter_ = {ica.n_iter_}). Skipping artifact removal.")
            cleaned[start:end, :] = raw_data[start:end, :]
            continue

        else: # ICA converged, proceed with the artifact removal
            # collect dropped components and their criteria
            drops = []  # list of (component_idx, criterion)

            for ic in range(sources.shape[0]):
                src = sources[ic]

                # Mid-band peak-rate
                nyq = 0.5 * fs
```

```
                b, a = butter(4, 15 / nyq, btype='low')
                filt = filtfilt(b, a, src) # filtering the component to remove multiple peaks
                threshold = 0.5 * np.max(np.abs(filt))
                peaks, _ = find_peaks(np.abs(filt), height=threshold)
                rate = len(peaks) / (filt.size / fs)

                # Kurtosis artifact
                k = abs(kurtosis(src))
                if k > kurtosis_thresh and rate < peak_rate_max:
                    drops.append((ic, 'Kurtosis'))

            # Zero-out dropped components and reconstruct
            if drops:
                drop_indices = [ic for ic, _ in drops]
                sources[drop_indices] = 0
            recon = ica.inverse_transform(sources.T).T  # shape (channels, time)
            cleaned[start:end, :] = recon.T  # shape (time, channels)

    if transposed:
        cleaned = cleaned.T

    return cleaned

def preprocess_signal(data, fs_old=125, fs_new=125,
                      bp_low=1., bp_high=40., bp_order=4,
                      notch_freq=50., notch_q=30.):
    n_samps, n_ch = data.shape

    # Z-normalization
    data_norm = (data - data.mean(axis=0)) / data.std(axis=0)
    # Band & notch filters
    nyq = 0.5 * fs_old
    low, high = bp_low/nyq, bp_high/nyq
    b_bp, a_bp = butter(bp_order, [low, high], btype='band')
    b_notch, a_notch = iirnotch(notch_freq, notch_q, fs_old)
    data_filt = np.zeros_like(data_norm)
    for ch in range(n_ch):
        tmp = filtfilt(b_bp, a_bp, data_norm[:, ch])
        data_filt[:, ch] = filtfilt(b_notch, a_notch, tmp)

    data_ica = clean_signal_ica(data_filt, fs=fs_old)

    # Resample
    if fs_old != fs_new:
        n_new = int(round(n_samps * fs_new / fs_old))
        return resample(data_ica, n_new, axis=0)
    return data_ica

# Funzione che estrae sliding windows di 2 s e applica preprocess su ciascuna
def sliding_preproc_subblocks(data_raw: np.ndarray,
                              sub_secs: float,
                              fs: int,
                              overlap_frac: float,
                              label: int) -> TensorDataset:
    sub_size = int(sub_secs * fs)
    step     = int(sub_size * (1.0 - overlap_frac))
    assert step > 0, "overlap_frac deve essere < 1.0"
    subs = []
    for start in range(0, len(data_raw) - sub_size + 1, step):
```

```
        seg = data_raw[start:start + sub_size, :]               # [sub_size, n_ch]
        seg_p = preprocess_signal(seg, fs_old=fs, fs_new=fs)    # preprocess su 2 s
        subs.append(seg_p.astype(np.float32))
    if not subs:
        raise ValueError("Nessuna finestra generata: controlla dimensioni e overlap")
    sub_tensor = torch.from_numpy(np.stack(subs))               # [N_sub, sub_size, n_ch]
    labels     = torch.full((len(subs),), label, dtype=torch.long)
    return TensorDataset(sub_tensor, labels)

# Parametri finestre
sub_secs       = 2.0
overlap_train  = 0.5   # 50% overlap per train/val
overlap_test   = 0.75  # 75% overlap per test

# Creo i dataset direttamente da raw + preprocess a 2 s
relax_train_ds  = sliding_preproc_subblocks(relax_rest,    sub_secs, fs, overlap_train, label=0)
stroop_train_ds = sliding_preproc_subblocks(stroop_rest,   sub_secs, fs, overlap_train, label=1)
relax_test_ds   = sliding_preproc_subblocks(relax_test_raw, sub_secs, fs, overlap_test,  label=0)
stroop_test_ds  = sliding_preproc_subblocks(stroop_test_raw,sub_secs, fs, overlap_test,  label=1)

# Split train vs val
total_relax = len(relax_train_ds)
val_len_r   = int(0.1 * total_relax)
train_len_r = total_relax - val_len_r
relax_train, relax_val = random_split(relax_train_ds, [train_len_r, val_len_r],
                                      generator=torch.Generator().manual_seed(42))

total_stroop = len(stroop_train_ds)
val_len_s    = int(0.1 * total_stroop)
train_len_s  = total_stroop - val_len_s
stroop_train, stroop_val = random_split(stroop_train_ds, [train_len_s, val_len_s],
                                        generator=torch.Generator().manual_seed(42))

# Concat e DataLoader
train_ds = ConcatDataset([relax_train, stroop_train])
val_ds   = ConcatDataset([relax_val,   stroop_val])
test_ds  = ConcatDataset([relax_test_ds, stroop_test_ds])

batch_size = 32
train_loader = DataLoader(train_ds,  batch_size=batch_size, shuffle=True, drop_last=True)
val_loader   = DataLoader(val_ds,    batch_size=batch_size, shuffle=True, drop_last=False)
test_loader  = DataLoader(test_ds,   batch_size=batch_size, shuffle=True, drop_last=False)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# %%
```

```python
class EEGNet(nn.Module):
    def __init__(self, num_channels, segment_length, num_class=2):
        super().__init__()

        # keep basic hyper-params
        self.num_channels = num_channels
        self.segment_length = segment_length

        # Layer 1
        self.conv2d = nn.Conv2d(1, 16, kernel_size=(1, 64), padding=0)
        self.bn1    = nn.BatchNorm2d(16)
        self.elu    = nn.ELU()
        self.drop   = nn.Dropout(0.25)

        # Layer 2
        self.pad1       = nn.ZeroPad2d((16, 17, 0, 1))
        self.depth_conv = nn.Conv2d(16, 4, kernel_size=(2, 32))
        self.bn2        = nn.BatchNorm2d(4)
        self.pool1      = nn.AvgPool2d((2, 4))

        # Layer 3
        self.pad2       = nn.ZeroPad2d((2, 1, 4, 3))
        self.sep_dep    = nn.Conv2d(4, 4, kernel_size=(8, 4), groups=4)
        self.sep_point  = nn.Conv2d(4, 4, kernel_size=1)
        self.bn3        = nn.BatchNorm2d(4)
        self.pool2      = nn.AvgPool2d((2, 4))

        # use a dummy pass to infer flattened feature size
        with torch.no_grad():
            dummy = torch.zeros(1, 1, num_channels, segment_length)
            x = self._forward_features(dummy)
            n_features = x.shape[1]

        # final classifier
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(n_features, num_class),
            nn.Softmax(dim=1)
        )

    def _forward_features(self, x):
        x = self.conv2d(x)
        x = self.bn1(x);  x = self.elu(x);  x = self.drop(x)

        x = self.pad1(x)
        x = self.depth_conv(x)
        x = self.bn2(x);  x = self.elu(x);  x = self.drop(x)
        x = self.pool1(x)

        x = self.pad2(x)
        x = self.sep_dep(x)
        x = self.sep_point(x)
        x = self.bn3(x);  x = self.elu(x);  x = self.drop(x)
        x = self.pool2(x)

        x = torch.flatten(x, 1)
        return x

    def forward(self, x):
        x = self._forward_features(x)
        x = self.classifier(x)
```

```
        return x

# Model instantiation
tot_epochs = 40
num_channels = relax_data_1.shape[1]
segment_length = int(sub_secs * fs)
model = EEGNet(num_channels=num_channels, segment_length=segment_length).to(
    device)
loss_fn = nn.CrossEntropyLoss().to(device)
LR = 0.00004

print("LR:", LR)
optimizer = optim.Adam(model.parameters(), lr=LR, weight_decay=1e-4, betas
    =(0.9, 0.999))
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=
    tot_epochs)

# 4) Training / validation loop
train_losses = []
train_accs = []
val_losses = []
val_accs = []

for epoch in range(1, tot_epochs+1):
    #           TRAINING
    model.train()
    train_loss = 0.0
    train_acc  = 0.0
    for X, y in train_loader:
        # X: [B, T, C] -> reorder for Conv2d: [B, 1, C, T]
        X = X.permute(0, 2, 1).unsqueeze(1).to(device)
        y = y.to(device)

        optimizer.zero_grad()
        logits = model(X)
        loss = loss_fn(logits, y)
        loss.backward()
        optimizer.step()

        # accumulate
        train_loss += loss.item() * X.size(0)
        train_acc  += (logits.argmax(dim=1) == y).sum().item()

    train_loss /= len(train_loader.dataset)
    train_acc  /= len(train_loader.dataset)
    train_losses.append(train_loss)
    train_accs.append(train_acc)

    #           VALIDATION
    model.eval()
    val_loss = 0.0
    val_acc  = 0.0
    with torch.no_grad():
        for X, y in val_loader:
            X = X.permute(0, 2, 1).unsqueeze(1).to(device)
            y = y.to(device)

            logits = model(X)
            loss = loss_fn(logits, y)

            val_loss += loss.item() * X.size(0)
```

```
            val_acc   += (logits.argmax(dim=1) == y).sum().item()

    val_loss /= len(val_loader.dataset)
    val_acc  /= len(val_loader.dataset)
    val_losses.append(val_loss)
    val_accs.append(val_acc)


    #            SCHEDULER STEP
    if scheduler:
        scheduler.step()

    #            LOG
    print(f"Epoch {epoch:03d} | "
          f"Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.4f} | "
          f"Val Loss:   {val_loss:.4f}, Val Acc:   {val_acc:.4f}")

# 5) After training, run test set
model.eval()
test_acc = 0.0
with torch.no_grad():
    for X, y in test_loader:
        X = X.permute(0, 2, 1).unsqueeze(1).to(device)
        y = y.to(device)
        logits = model(X)
        test_acc += (logits.argmax(dim=1) == y).sum().item()
test_acc /= len(test_loader.dataset)
print(f"\nTest Accuracy: {test_acc:.4f}")
```

ITR Calculation

```python
import math

accuracy = 0.9273
num_classes = 2
signal_duration = 2.0  # seconds
processing_time = 0.2527 #0.355  # seconds

trial_time = signal_duration + processing_time  # seconds
trial_time_minutes = trial_time / 60.0

# Calculate the ITR (Information Transfer Rate)
# Formula: ITR = log2(N) + P * log2(P) + (1-P) * log2((1-P)/(N-1)) bits/trial
# Where N is the number of classes and P is the accuracy


bits_per_trial = math.log2(num_classes) + accuracy * math.log2(accuracy) + (1
    - accuracy) * math.log2((1 - accuracy) / (num_classes - 1))
itr_bpm = bits_per_trial / trial_time_minutes

print(f"The ITR value is: {itr_bpm:.2f} bits/minute")
print(f"The ITR value is: {bits_per_trial:.2f} bits/trial")
```

Bibliography

[1] Vernon J. Lawhern, Amelia J. Solon, Nicholas R. Waytowich, Stephen M. Gordon, Chou P. Hung, and Brent J. Lance. “EEGNet: A Compact Convolutional Network for EEG-based Brain-Computer Interfaces”. In: *Journal of Neural Engineering* 15.5 (Oct. 2018). arXiv:1611.08024 [cs], p. 056013. ISSN: 1741-2560, 1741-2552. DOI: 10.1088/1741-2552/aace8c. URL: http://arxiv.org/abs/1611.08024 (visited on 04/14/2025) (cit. on pp. I, 2, 38, 64).

[2] Rajdeep Ghosh et al. “SAM 40: Dataset of 40 Subject EEG Recordings to Monitor the Induced-Stress while performing Stroop Color-Word Test, Arithmetic Task, and Mirror Image Recognition Task”. In: *Data in Brief* 40 (Feb. 2022), p. 107772. DOI: 10.1016/j.dib.2021.107772 (cit. on pp. I, 2, 29, 30).

[3] Sheldon Cohen, Denise Janicki-Deverts, and Gregory E. Miller. “Psychological stress and disease”. eng. In: *JAMA* 298.14 (Oct. 2007), pp. 1685–1687. ISSN: 1538-3598. DOI: 10.1001/jama.298.14.1685 (cit. on pp. 1, 10).

[4] Ronald C Kessler. “THE ECONOMIC BURDEN OF ANXIETY AND STRESS DISORDERS”. en. In: () (cit. on pp. 1, 10).

[5] *Guidelines on mental health at work*. en. URL: https://www.who.int/publications/i/item/9789240053052 (visited on 06/04/2025) (cit. on pp. 1, 10).

[6] Farzad Saffari, Kian Norouzi, Luis E. Bruni, Sahar Zarei, and Thomas Z. Ramsøy. “Impact of varying levels of mental stress on phase information of EEG Signals: A study on the Frontal, Central, and parietal regions”. en. In: *Biomedical Signal Processing and Control* 86 (Sept. 2023), p. 105236. ISSN: 17468094. DOI: 10.1016/j.bspc.2023.105236. URL: https://linkinghub.elsevier.com/retrieve/pii/S1746809423006699 (visited on 06/04/2025) (cit. on pp. 1, 6).

[7] Frederico A. C. Azevedo, Ludmila R. B. Carvalho, Lea T. Grinberg, José Marcelo Farfel, Renata E. L. Ferretti, Renata E. P. Leite, Wilson Jacob Filho, Roberto Lent, and Suzana Herculano-Houzel. “Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain”. eng. In: *The Journal of Comparative Neurology* 513.5 (Apr. 2009), pp. 532–541. ISSN: 1096-9861. DOI: 10.1002/cne.21974 (cit. on p. 4).

[8] M. M. Mesulam. “From sensation to cognition”. eng. In: *Brain: A Journal of Neurology* 121 (Pt 6) (June 1998), pp. 1013–1052. ISSN: 0006-8950. DOI: 10.1093/brain/121.6.1013 (cit. on p. 4).

[9] Matti Gärtner, Lea Rohde-Liebenau, Simone Grimm, and Malek Bajbouj. “Working memory-related frontal theta activity is decreased under acute stress”. In: *Psychoneuroendocrinology* 43 (May 2014), pp. 105–113. ISSN: 0306-4530. DOI: 10.1016/j.psyneuen.2014.02.009. URL: https://www.sciencedirect.com/science/article/pii/S0306453014000602 (visited on 03/19/2025) (cit. on p. 5).

[10] John Edward Hall, Arthur C. Guyton, and Diego Tronca. *Guyton e Hall fisiologia medica*. ita. 14. ed. / a cura di Pietro Baldelli [e altri]. OCLC: 1274124889. Milano: Edra, 2021. ISBN: 978-88-214-5541-4 (cit. on p. 5).

[11] J. E. LeDoux. “Emotion circuits in the brain”. eng. In: *Annual Review of Neuroscience* 23 (2000), pp. 155–184. ISSN: 0147-006X. DOI: 10.1146/annurev.neuro.23.1.155 (cit. on p. 5).

[12] Rashmi CR and Shantala C P. “Cognitive Stress Recognition During Mathematical Task and EEG Changes Following Audio-Visual Stimuli for Relaxation”. In: *2023 International Conference on Sustainable Communication Networks and Application (ICSCNA)*. Nov. 2023, pp. 612–617. DOI: 10.1109/ICSCNA58489.2023.10370715. URL: https://ieeexplore.ieee.org/document/10370715/ (visited on 02/04/2025) (cit. on pp. 6, 8, 15–17, 20, 26).

[13] Ubaid M. Al-Saggaf, Syed Faraz Naqvi, Muhammad Moinuddin, Sulhi Ali Alfakeh, and Syed Saad Azhar Ali. “Performance Evaluation of EEG Based Mental Stress Assessment Approaches for Wearable Devices”. English. In: *Frontiers in Neurorobotics* 15 (Feb. 2022). Publisher: Frontiers. ISSN: 1662-5218. DOI: 10.3389/fnbot.2021.819448. URL: https://www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2021.819448/full (visited on 02/14/2025) (cit. on pp. 8, 15–19, 24, 26, 60).

[14] Girijesh Prasad, Pawel Herman, Damien Coyle, Suzanne McDonough, and Jacqueline Crosbie. “Applying a brain-computer interface to support motor imagery practice in people with stroke for upper limb recovery: a feasibility study”. eng. In: *Journal of Neuroengineering and Rehabilitation* 7 (Dec. 2010), p. 60. ISSN: 1743-0003. DOI: 10.1186/1743-0003-7-60 (cit. on p. 8).

[15] Jiaqi Xiong et al. “Impact of COVID-19 pandemic on mental health in the general population: A systematic review”. In: *Journal of Affective Disorders* 277 (Dec. 2020), pp. 55–64. ISSN: 0165-0327. DOI: 10.1016/j.jad.2020.08.001. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7413844/ (visited on 03/19/2025) (cit. on p. 9).

[16] Nina Vindegaard and Michael Eriksen Benros. “COVID-19 pandemic and mental health consequences: Systematic review of the current evidence”. eng.

In: *Brain, Behavior, and Immunity* 89 (Oct. 2020), pp. 531–542. ISSN: 1090-2139. DOI: 10.1016/j.bbi.2020.05.048 (cit. on p. 9).

[17] Hans Selye. "Stress and the General Adaptation Syndrome". In: *British Medical Journal* 1.4667 (June 1950), pp. 1383–1392. ISSN: 0007-1447. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2038162/ (visited on 03/19/2025) (cit. on p. 9).

[18] *Physiology and Neurobiology of Stress and Adaptation: Central Role of the Brain*. en. DOI: 10.1152/physrev.00041.2006. URL: https://journals.physiology.org/doi/epdf/10.1152/physrev.00041.2006 (visited on 02/14/2025) (cit. on p. 9).

[19] *Hans Selye - Stress Without Distress-Lippincott Williams & Wilkins (1974) PDF | PDF*. it. URL: https://www.scribd.com/document/442108406/Hans-Selye-Stress-Without-Distress-Lippincott-Williams-Wilkins-1974-pdf (visited on 02/14/2025) (cit. on p. 9).

[20] Susan Folkman. "Stress: Appraisal and Coping". en. In: *Encyclopedia of Behavioral Medicine*. Ed. by Marc D. Gellman and J. Rick Turner. New York, NY: Springer, 2013, pp. 1913–1915. ISBN: 978-1-4419-1005-9. DOI: 10.1007/978-1-4419-1005-9_215. URL: https://doi.org/10.1007/978-1-4419-1005-9_215 (visited on 02/14/2025) (cit. on p. 9).

[21] B. S. McEwen. "Stress, adaptation, and disease. Allostasis and allostatic load". eng. In: *Annals of the New York Academy of Sciences* 840 (May 1998), pp. 33–44. ISSN: 0077-8923. DOI: 10.1111/j.1749-6632.1998.tb09546.x (cit. on p. 9).

[22] *Depression and Other Common Mental Disorders*. en. URL: https://www.who.int/publications/i/item/depression-global-health-estimates (visited on 02/14/2025) (cit. on p. 10).

[23] *Mental Disorders Cost Society Billions in Unearned Income*. EN. Sept. 2015. URL: https://www.nih.gov/news-events/news-releases/mental-disorders-cost-society-billions-unearned-income (visited on 06/04/2025) (cit. on p. 10).

[24] Lori L. Davis, Jeff Schein, Martin Cloutier, Patrick Gagnon-Sanschagrin, Jessica Maitland, Annette Urganus, Annie Guerin, Patrick Lefebvre, and Christy R. Houle. "The Economic Burden of Posttraumatic Stress Disorder in the United States From a Societal Perspective". In: *The Journal of Clinical Psychiatry* 83.3 (Apr. 2022). ISSN: 1555-2101. DOI: 10.4088/JCP.21m14116. URL: https://www.psychiatrist.com/jcp/economic-burden-posttraumatic-stress-disorder-united-states-societal-perspective (visited on 06/23/2025) (cit. on p. 10).

[25] *Detection of Mental Stress through EEG Signal in Virtual Reality Environment*. URL: https://www.mdpi.com/2079-9292/10/22/2840 (visited on 02/03/2025) (cit. on pp. 11, 13, 15–19, 26).

[26] Aniana Cruz, Gabriel Pires, Ana C. Lopes, and Urbano J. Nunes. "Detection of Stressful Situations Using GSR While Driving a BCI-controlled Wheelchair". In: *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. ISSN: 1558-4615. July 2019, pp. 1651–1656. DOI: `10.1109/EMBC.2019.8857748`. URL: `https://ieeexplore.ieee.org/abstract/document/8857748` (visited on 01/31/2025) (cit. on p. 11).

[27] Kannadasan K, Lilly Pushparani M P, Jerina Stanis J, Haresh M V, Ambati Rami Reddy, and B. Shameedha Begum. "StressDetect: A Deep Learning Approach for Mental Stress Detection Using Time-Frequency Representation of EEG Signals". In: *2024 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES)*. Sept. 2024, pp. 1–6. DOI: `10.1109/SPICES62143.2024.10779753`. URL: `https://ieeexplore.ieee.org/document/10779753/?arnumber=10779753&tag=1` (visited on 02/12/2025) (cit. on pp. 11, 15, 16, 19, 27, 38).

[28] Sadasivan Puthusserypady. *22053_Lecture_1_Introduction*. English. 2025 (cit. on p. 13).

[29] L. A. Farwell and E. Donchin. "Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials". In: *Electroencephalography and Clinical Neurophysiology* 70.6 (Dec. 1988), pp. 510–523. ISSN: 0013-4694. DOI: `10.1016/0013-4694(88)90149-6`. URL: `https://www.sciencedirect.com/science/article/pii/0013469488901496` (visited on 02/17/2025) (cit. on p. 15).

[30] Jiahua Xu, Tung-Lung Liu, Zhenglei Wu, Zheng Wu, Yang Li, and Andreas Nurnberger. "Neurorehabilitation System in Virtual Reality with Low-Cost BCI Devices". In: *2020 IEEE International Conference on Human-Machine Systems (ICHMS)*. Rome, Italy: IEEE, Sept. 2020, pp. 1–3. ISBN: 978-1-7281-5871-6. DOI: `10.1109/ICHMS49158.2020.9209560`. URL: `https://ieeexplore.ieee.org/document/9209560/` (visited on 02/17/2025) (cit. on p. 15).

[31] *BCI controlled FES system for complete neurorehabilitation of post-stroke patients - DTU Findit*. URL: `https://findit.dtu.dk/en/catalog/5d2b0df4d9001d018a233410` (visited on 02/17/2025) (cit. on p. 15).

[32] Sergi Bermúdez I Badia, Hani Samaha, Andrés García Morgade, and Paul F.M.J. Verschure. "Exploring the synergies of a hybrid BCI - VR neurorehabilitation system". In: *2011 International Conference on Virtual Rehabilitation*. Zurich, Switzerland: IEEE, June 2011, pp. 1–8. ISBN: 978-1-61284-475-6 978-1-61284-474-9. DOI: `10.1109/ICVR.2011.5971813`. URL: `https://ieeexplore.ieee.org/document/5971813/` (visited on 02/17/2025) (cit. on p. 15).

[33] A. Miladinovic, M. Ajcevic, P. Busan, J. Jarmolowska, G. Silveri, M. Deodato, S. Mezzarobba, P. P. Battaglini, and A. Accardo. "Evaluation of Motor Imagery-Based BCI methods in neurorehabilitation of Parkinson's Disease patients". In: *2020 42nd Annual International Conference of the IEEE Engineering in*

Medicine & Biology Society (EMBC). Montreal, QC, Canada: IEEE, July 2020, pp. 3058–3061. ISBN: 978-1-7281-1990-8. DOI: 10.1109/EMBC44109.2020.9176651. URL: https://ieeexplore.ieee.org/document/9176651/ (visited on 02/17/2025) (cit. on p. 15).

[34] Jiahua Xu, Tung-Lung Liu, Zhenglei Wu, Zheng Wu, Yang Li, and Andreas Nürnberger. "Neurorehabilitation System in Virtual Reality with Low-Cost BCI Devices". In: *2020 IEEE International Conference on Human-Machine Systems (ICHMS)*. Sept. 2020, pp. 1–3. DOI: 10.1109/ICHMS49158.2020.9209560. URL: https://ieeexplore.ieee.org/document/9209560/?arnumber=9209560 (visited on 02/17/2025) (cit. on p. 15).

[35] Edgar P. Torres, Edgar A. Torres, Myriam Hernández-Álvarez, and Sang Guun Yoo. "EEG-Based BCI Emotion Recognition: A Survey". en. In: *Sensors* 20.18 (Jan. 2020). Number: 18 Publisher: Multidisciplinary Digital Publishing Institute, p. 5083. ISSN: 1424-8220. DOI: 10.3390/s20185083. URL: https://www.mdpi.com/1424-8220/20/18/5083 (visited on 02/17/2025) (cit. on p. 15).

[36] Haiyun Huang, Qiuyou Xie, Jiahui Pan, Yanbin He, Zhenfu Wen, Ronghao Yu, and Yuanqing Li. "An EEG-Based Brain Computer Interface for Emotion Recognition and Its Application in Patients with Disorder of Consciousness". In: *IEEE Transactions on Affective Computing* 12.4 (Oct. 2021). Conference Name: IEEE Transactions on Affective Computing, pp. 832–842. ISSN: 1949-3045. DOI: 10.1109/TAFFC.2019.2901456. URL: https://ieeexplore.ieee.org/document/8651389/?arnumber=8651389 (visited on 02/17/2025) (cit. on p. 15).

[37] Linxing Jiang, Andrea Stocco, Darby Losey, Justin Abernethy, Chantel Prat, and Rajesh Rao. "BrainNet: A Multi-Person Brain-to-Brain Interface for Direct Collaboration Between Brains". In: *Scientific Reports* 9 (Apr. 2019). DOI: 10.1038/s41598-019-41895-7 (cit. on p. 15).

[38] Rajesh P. N. Rao, Andrea Stocco, Matthew Bryan, Devapratim Sarma, Tiffany M. Youngquist, Joseph Wu, and Chantel S. Prat. "A Direct Brain-to-Brain Interface in Humans". en. In: *PLOS ONE* 9.11 (Nov. 2014). Publisher: Public Library of Science, e111332. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0111332. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0111332 (visited on 02/17/2025) (cit. on p. 15).

[39] Seung-Schik Yoo, Hyungmin Kim, Emmanuel Filandrianos, Seyed Javid Taghados, and Shinsuk Park. "Non-invasive brain-to-brain interface (BBI): establishing functional links between two brains". eng. In: *PloS One* 8.4 (2013), e60410. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0060410 (cit. on p. 15).

[40] Günter Edlinger, Clemens Holzner, and Christoph Guger. "A Hybrid Brain-Computer Interface for Smart Home Control". en. In: *Human-Computer Interaction. Interaction Techniques and Environments*. Ed. by Julie A. Jacko.

Berlin, Heidelberg: Springer, 2011, pp. 417–426. ISBN: 978-3-642-21605-3. DOI: 10.1007/978-3-642-21605-3_46 (cit. on p. 15).

[41] Xiaoke Chai, Zhimin Zhang, Kai Guan, Yangting Lu, Guitong Liu, Tengyu Zhang, and Haijun Niu. "A hybrid BCI-controlled smart home system combining SSVEP and EMG for individuals with paralysis". In: *Biomedical Signal Processing and Control* 56 (Feb. 2020), p. 101687. ISSN: 1746-8094. DOI: 10.1016/j.bspc.2019.101687. URL: https://www.sciencedirect.com/science/article/pii/S174680941930268X (visited on 02/17/2025) (cit. on p. 15).

[42] Andreas Pinegger, Hannah Hiebel, Selina C. Wriessnegger, and Gernot R. Müller-Putz. "Composing only by thought: Novel application of the P300 brain-computer interface". en. In: *PLOS ONE* 12.9 (2017). Publisher: Public Library of Science, e0181584. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0181584. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0181584 (visited on 02/17/2025) (cit. on p. 15).

[43] Guo Jun and K. G. Smitha. "EEG based stress level identification". In: *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (Oct. 2016). Conference Name: 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC) ISBN: 9781509018970 Place: Budapest, Hungary Publisher: IEEE, pp. 003270–003274. DOI: 10.1109/SMC.2016.7844738. URL: http://ieeexplore.ieee.org/document/7844738/ (visited on 02/11/2025) (cit. on pp. 15–18, 21, 26).

[44] Amit Kumar, Barath J. K., Shanmukh P., and Deepak Joshi. "StreXNet: A Novel End-to-End Deep-Learning-Based Improved Multilevel Mental Stress Classification From EEG Sensors". In: *IEEE Sensors Journal* 25.2 (Jan. 2025). Conference Name: IEEE Sensors Journal, pp. 3538–3551. ISSN: 1558-1748. DOI: 10.1109/JSEN.2024.3506984. URL: https://ieeexplore.ieee.org/document/10780949/?arnumber=10780949&tag=1 (visited on 02/13/2025) (cit. on pp. 15, 16, 19–21, 27, 38).

[45] Ruiqi Fu et al. "Symmetric Convolutional and Adversarial Neural Network Enables Improved Mental Stress Classification From EEG". eng. In: *IEEE transactions on neural systems and rehabilitation engineering: a publication of the IEEE Engineering in Medicine and Biology Society* 30 (2022), pp. 1384–1400. ISSN: 1558-0210. DOI: 10.1109/TNSRE.2022.3174821 (cit. on pp. 15, 20, 27).

[46] Heba M. Afify, Kamel K. Mohammed, and Aboul Ella Hassanien. "Stress detection based EEG under varying cognitive tasks using convolution neural network". en. In: *Neural Computing and Applications* (Jan. 2025). ISSN: 1433-3058. DOI: 10.1007/s00521-024-10737-7. URL: https://doi.org/10.1007/s00521-024-10737-7 (visited on 02/12/2025) (cit. on pp. 16, 20, 24, 27).

[47] Lokesh Malviya and Sandip Mal. "A novel technique for stress detection from EEG signal using hybrid deep learning model". en. In: *Neural Computing and Applications* 34.22 (Nov. 2022), pp. 19819–19830. ISSN: 0941-0643, 1433-3058. DOI: 10.1007/s00521-022-07540-7. URL: https://link.springer.com/10.1007/s00521-022-07540-7 (visited on 02/12/2025) (cit. on pp. 16, 19, 20, 27).

[48] Chin-Teng Lin, I.-Fang Chung, Li-Wei Ko, Yu-Chieh Chen, Sheng-Fu Liang, and Jeng-Ren Duann. "EEG-based assessment of driver cognitive responses in a dynamic virtual-reality driving environment". eng. In: *IEEE transactions on bio-medical engineering* 54.7 (July 2007), pp. 1349–1352. ISSN: 0018-9294. DOI: 10.1109/TBME.2007.891164 (cit. on p. 16).

[49] Swaymprabha Alias Megha Mane and Arundhati Shinde. "StressNet: Hybrid model of LSTM and CNN for stress detection from electroencephalogram signal (EEG)". In: *Results in Control and Optimization* 11 (June 2023), p. 100231. ISSN: 2666-7207. DOI: 10.1016/j.rico.2023.100231. URL: https://www.sciencedirect.com/science/article/pii/S2666720723000334 (visited on 02/17/2025) (cit. on pp. 16, 19, 20, 24, 27, 38).

[50] Anum Asif, Muhammad Majid, and Syed Muhammad Anwar. "Human stress classification using EEG signals in response to music tracks". In: *Computers in Biology and Medicine* 107 (Apr. 2019), pp. 182–196. ISSN: 0010-4825. DOI: 10.1016/j.compbiomed.2019.02.015. URL: https://www.sciencedirect.com/science/article/pii/S0010482519300629 (visited on 02/12/2025) (cit. on pp. 16–19, 21, 26).

[51] Muhammad Adeel Hafeez and Sadia Shakil. "EEG-based stress identification and classification using deep learning". en. In: *Multimedia Tools and Applications* 83.14 (Oct. 2023), pp. 42703–42719. ISSN: 1573-7721. DOI: 10.1007/s11042-023-17111-0. URL: https://link.springer.com/10.1007/s11042-023-17111-0 (visited on 02/13/2025) (cit. on pp. 16, 19, 20, 27, 38).

[52] Debashis Das Chakladar, Shubhashis Dey, Partha Roy, and Debi Dogra. "EEG-based mental workload estimation using deep BLSTM-LSTM network and evolutionary algorithm". In: *Biomedical Signal Processing and Control* 60 (July 2020), p. 101989. DOI: 10.1016/j.bspc.2020.101989 (cit. on pp. 17, 20, 26).

[53] Houtan Jebelli, Mohammad Mahdi Khalili, and SangHyun Lee. "Mobile EEG-Based Workers' Stress Recognition by Applying Deep Neural Network". en. In: *Advances in Informatics and Computing in Civil and Construction Engineering*. Ed. by Ivan Mutis and Timo Hartmann. Cham: Springer International Publishing, 2019, pp. 173–180. ISBN: 978-3-030-00220-6. DOI: 10.1007/978-3-030-00220-6_21 (cit. on pp. 19, 26).

[54] Bruce S. McEwen and Peter J. Gianaros. "Central role of the brain in stress and adaptation: links to socioeconomic status, health, and disease". eng. In: *Annals of the New York Academy of Sciences* 1186 (Feb. 2010), pp. 190–222.

ISSN: 1749-6632. DOI: 10.1111/j.1749-6632.2009.05331.x (cit. on pp. 21, 62).

[55] Houtan Jebelli, Sungjoo Hwang, and SangHyun Lee. "An EEG Signal Processing Framework to Obtain High-Quality Brain Waves from an Off-the-Shelf Wearable EEG Device". In: *Journal of Computing in Civil Engineering* 32 (Oct. 2017). DOI: 10.1061/(ASCE)CP.1943-5487.0000719 (cit. on p. 33).

[56] Anders Kallner. "Formulas". In: *Laboratory Statistics (Second Edition)*. Ed. by Anders Kallner. Elsevier, Jan. 2018, pp. 1–140. ISBN: 978-0-12-814348-3. DOI: 10.1016/B978-0-12-814348-3.00001-0. URL: https://www.sciencedirect.com/science/article/pii/B9780128143483000010 (visited on 05/29/2025) (cit. on p. 33).

[57] M. A. Klados, C. Bratsas, C. Frantzidis, C. L. Papadelis, and P. D. Bamidis. "A Kurtosis-Based Automatic System Using Naïve Bayesian Classifier to Identify ICA Components Contaminated by EOG or ECG Artifacts". en. In: *XII Mediterranean Conference on Medical and Biological Engineering and Computing 2010*. Ed. by Panagiotis D. Bamidis and Nicolas Pallikarakis. Berlin, Heidelberg: Springer, 2010, pp. 49–52. ISBN: 978-3-642-13039-7. DOI: 10.1007/978-3-642-13039-7_13 (cit. on p. 33).

[58] Harsh Srivastava and Kishor Sarawadekar. "A Depthwise Separable Convolution Architecture for CNN Accelerator". In: *2020 IEEE Applied Signal Processing Conference (ASPCON)*. Oct. 2020, pp. 1–5. DOI: 10.1109/ASPCON49795.2020.9276672. URL: https://ieeexplore.ieee.org/abstract/document/9276672 (visited on 04/28/2025) (cit. on p. 39).

[59] Francois Chollet. "Xception: Deep Learning With Depthwise Separable Convolutions". In: 2017, pp. 1251–1258. URL: https://openaccess.thecvf.com/content_cvpr_2017/html/Chollet_Xception_Deep_Learning_CVPR_2017_paper.html (visited on 04/28/2025) (cit. on p. 39).

[60] Junxiu Liu, Mingxing Li, Yuling Luo, Su Yang, Wei Li, and Yifei Bi. "Alzheimer's disease detection using depthwise separable convolutional neural networks". In: *Computer Methods and Programs in Biomedicine* 203 (May 2021), p. 106032. ISSN: 0169-2607. DOI: 10.1016/j.cmpb.2021.106032. URL: https://www.sciencedirect.com/science/article/pii/S0169260721001073 (visited on 04/28/2025) (cit. on p. 39).

[61] J. R. Wolpaw et al. "Brain-computer interface technology: a review of the first international meeting". eng. In: *IEEE transactions on rehabilitation engineering: a publication of the IEEE Engineering in Medicine and Biology Society* 8.2 (June 2000), pp. 164–173. ISSN: 1063-6528. DOI: 10.1109/tre.2000.847807 (cit. on p. 47).

[62] *(PDF) A high-ITR SSVEP-based BCI speller*. en. URL: https://www.researchgate.net/publication/269998531_A_high-ITR_SSVEP-based_BCI_speller#fullTextFileContent (visited on 06/04/2025) (cit. on pp. 60, 65).

[63] Xiaotong Lyu, Peng Ding, Siyu Li, Yuyang Dong, Lei Su, Lei Zhao, Anmin Gong, and Yunfa Fu. "Human factors engineering of BCI: an evaluation for satisfaction of BCI based on motor imagery". en. In: *Cognitive Neurodynamics* 17.1 (Feb. 2023), pp. 105–118. ISSN: 1871-4080, 1871-4099. DOI: 10.1007/s11571-022-09808-z. URL: https://link.springer.com/10.1007/s11571-022-09808-z (visited on 06/04/2025) (cit. on pp. 60, 65).

Dedications

Thanks to my mum and my dad for supporting me in these 5 years of my life. Thanks to Sabrina and Davide for inspiring me to go and finally see the world. Thanks to zia ansia for the tagliata and the canteen food, either in Turin or in DTU. Thanks to all my relatives, those who are still here, and those who helped me from Heaven. Thanks to Silve for the deep talks and the bike trips together, always a source of peace in my life. Thanks to Flu, Bob, Fede, Flucio for having the highest number of nicknames in the world, and for bringing a touch of Cuneo to Copenhagen. Thanks to Jack for the walks in Turin, always calming the spirits. Thanks to Santo for the psycology walks to understand the purpose of life. Thanks to Bocca, Ribe, Mattia, and Ricky for sharing moments in via Fratelli Carando 16 that will be unforgettable. Thanks to Johnny and Sara for all the Biomedical Engineering lessons followed side-by-side. Thanks to all the people from the Salesian Oratory, especially to don Thierry, don Flaviano, don Alby, and don Eric, for guiding me through life.

Thanks to Gaia and all my Erasmus peers from PoliTO for sharing a nice year here in Denmark. Thanks to Enrico, Marta and Martina for the mental support in the lab.

Thanks to all CAYAC and the *one and only dorm*, especially to Fr. Daniel for the laughs and serious moments, Maria, Sara and Juan Josè Maria for all the nights spent at the guitar, and praying rosaries, and doing holy hours; these moments will always be stuck in my mind. Thanks to Beni for the best music in the dorm, Cons for being my tata, Darcy for being my favourite dane, Elias for *Un dos tres* and *Monopoli* nights, Elis for not making me the worst cook in the dorm, Esther for the peace you brought in the dorm, Francisca for having the best boyfriend, Maca for the paella, Marta for the spanish lessons without wanting, Mix for the lunches together and the breathtaking hugs, Pepe for being finally an Italian who can cook, Teresinha for teching me that you can be in the dorm even if you're not in the dorm, Tommy for the crazy ideas that I don't know how to say no; without your mental support, I would have quit my thesis in February. Thanks to Pedro for being the most silent roommate ever, and thanks to Luisa for 3 months of crazy puzzle, I have to come back at least to do another one with you (this time, at least 4k). Thanks to you, who are reading these dedications. You have or you will be changing my life in good, for sure. Thanks! Thanks to God for all the months You've given to me and for all the people You've made me meet. I'm *ETERNALLY GRATEFUL*. Thanks to everyone whom I forgot. I love you all!