Deep Learning Approaches to Decoding Sensorimotor Rhythms for Accurate Cursor Control in Brain-Computer Interfaces Using a Large-Scale Dataset

Thesis in partial fulfillment of the requirements for the degree Master of Science (M.Sc.)

in the course of studies
Elektortechnik

submitted by
Ali Alouane

Matriculation number
397293

Under the scientific supervision of
Dr. rer. nat. Daniel Miklody

Under the scientific guidance of
Prof. Dr. Benjamin Blankertz

Berlin, January 2024

Technische Universität Berlin
Fakultät IV — Elektrotechnik und Informatik
Institut für Softwaretechnik und Theoretische Informatik
Fachgebiet Neurotechnologie

Acknowledgments

I would like to extend my deepest gratitude to my supervisors, Dr. Daniel Miklody and Dr. Oleksandr Zlatov , for their invaluable guidance, encouragement, and support throughout the course of this research. Their expertise and insight have been fundamental to the success of this work.

I am deeply thankful to Professor Benjamin Blankertz for his guidance and support throughout my studies. His mentorship has been crucial in my academic development.

Furthermore, I would like to express my heartfelt thanks to my family: my parents and my two sisters. This thesis is dedicated to them for their unwavering love, support, and belief in me. They have always been there for me, providing encouragement and strength during the challenging times of this academic endeavour. Their presence in my life has been a pillar of stability and a constant source of happiness.

Abstract

Deutsch

Motorimaginationsbasierte Gehirn-Computer-Schnittstellen (MI-BCIs) repräsentieren ein bedeutendes Paradigma im Bereich der Neurotechnologie, insbesondere für Menschen mit Lähmungen. Durch das mentale Üben von Bewegungen erleichtern MI-BCIs die Kontrolle über Prothesen oder Rollstühle. Die Konstruktion eines Klassifikators für MI-BCI-Anwendungen beinhaltet eine komplexe Verarbeitung und Merkmalsextraktion aus Gehirnsignalen, um zwischen verschiedenen mentalen Zuständen zu unterscheiden. BCI ist von Natur aus multidisziplinär und erfordert Expertise nicht nur in Neurowissenschaft, maschinellem Lernen und Elektrotechnik, sondern auch in der Softwareentwicklung. Die Komplexität der Integration dieser Disziplinen erfordert eine umfassende Toolbox, die Standardmethoden umfasst und regelmäßig mit neuen Datensätzen und Algorithmen aktualisiert wird. Eine solche Toolbox ermöglicht es Wissenschaftlern und Ingenieuren, Algorithmen effizient und schnell zu prototypisieren und zu entwickeln. Angesichts der zunehmenden Bedeutung von Deep Learning in zahlreichen Bereichen führt diese Arbeit eine Deep-Learning-Schnittstelle in eine bestehende BCI-Toolbox ein. Diese Schnittstelle vereinfacht die Entwicklung und das Training von neuronalen Netzwerken und folgt einem stärker automatisierten und standardisierten Machine-Learning-Lebenszyklus. Unter Verwendung des Sensorimotor Rhythm (SMR)-Datensatzes, dem größten offenen BCI-Datensatz, integriert diese Arbeit ihn als Standardressource in die BBCPy-Toolbox. Der Datensatz wird als Benchmark zur Bewertung der Leistung von Deep-Learning-Modellen bei der Klassifizierung von Motorimaginations-BCIs verwendet.

Schlüsselwörter: *BCI; EEG; Deep neural Network, BBCPy*

English

Motor imagery-based brain-computer interfaces (MI-BCIs) represent a significant paradigm within the field of neurotechnology, particularly for individuals with paralysis. By mentally rehearsing motor movements, MI-BCIs facilitate control over prosthetic limbs or wheelchairs. The construction of a classifier for MI-BCI applications involves complex processing and feature extraction from brain signals to distinguish between various mental states. BCI is inherently multidisciplinary, demanding expertise not only in neuroscience, machine learning, and electrical engineering but also in software development. The complexity of integrating these disciplines necessitates a comprehensive toolbox that encompasses standard methods and is regularly updated with new datasets and algorithms. Such a toolbox enables scientists and engineers to rapidly prototype and develop algorithms efficiently. Given the increasing prominence of deep learning in numerous fields, this thesis introduces a deep learning interface incorporated into an existing BCI toolbox. This interface streamlines the development and training of neural networks, adhering to a more automated and standardized machine learning lifecycle. Utilizing the Sensorimotor Rhythm (SMR) dataset, the largest open BCI dataset available, this thesis integrates it into the BBCPy toolbox as a standard resource. The dataset is employed as a benchmark for evaluating the performance of deep learning models in the Motor imagery BCI classification task.

Keywords: *BCI; EEG; Deep neural Network, BBCPy*

Contents

List of Figures

List of Tables

List of Abbreviations

Terms

ANN	**A**rtificial **N**eural **N**etworks
BCIs	**B**rain **C**omputer **I**nterfaces
CNNs	**C**onvolutional **N**eural **N**etworks
CSP	**C**ommon **S**patial **P**attern
CSP	**C**ommon **S**patial **P**attern
EEG	**E**lectro**E**ncephalo**G**raphy
GEVD	**G**eneralized **E**igenvalue **D**ecomposition
LSTM	**L**ong **S**hort-**T**erm **M**emory
MBAT	**M**ind-**B**ody **A**wareness **T**raining
MI	**M**otor **I**magery
RNN	**R**ecurrent **N**eural **N**etwork
SPD	**S**ymmetric **P**ositive **D**efinite
TPE	**T**ree-structured **P**arzen **E**stimator

1 Introduction

1.1 Motivation

***B**rain **C**omputer **I**nterfaces* (BCIs) are systems designed to decode brain signals into executable commands. Non-invasive BCIs utilize external electrodes or sensors, such as ***E**lectro**E**ncephalo**G**raphy* (EEG), mounted on the scalp without the need for surgical intervention. EEG is widely used and has become more attractive for commercial applications due to its cost-effectiveness. However, EEG suffers from high variability and non-stationary behavior in brain patterns, which drastically decreases the signal-to-noise ratio. Numerous algorithms and paradigms have been developed around motor sensor activity to enhance EEG decoding capabilities, yet this often comes at a cost: the raw signal must undergo excessive and complicated processing pipelines to extract salient features. With the recent breakthroughs of deep neural networks in various fields and their substantial capabilities to solve complex problems, the neurotechnology community has started to experiment with deep neural networks. These networks have the potential to shorten the traditional processing pipeline; however, deep learning is still a new approach that requires a significant amount of data compared to classical methods and also demands software skills to develop and maintain the machine learning lifecycle to maximize the full capabilities of deep learning.

1.2 Objectives and contributions

This work is two-fold. Firstly, it aims to investigate the Sensorimotor dataset (Stieger et al., 2021b), which is considered the largest dataset for motor imagery experiments—350 times larger than the benchmark BCI Competition datasets 2a

and 2b (**Gwon**). This dataset provides an excellent starting point for experiments with deep learning models, as such models require substantial amounts of data. It also allows for a more specific investigation into benchmark deep learning architectures like EEGNet (Lawhern et al., 2018) and other models for offline classification tasks. The second pillar of this thesis is to contribute to the BBCPy toolbox, developed by the neurotechnology group at TU Berlin, and to integrate the studied dataset as a standard datatype. This toolbox is considered a successor to the Wyrm toolbox (Venthur, Blankertz, 2014), and with BBCPy, we aim to develop a universal toolbox for BCI applications. It supports both offline and online classification and facilitates easy integration with other toolboxes such as MNE (Zubarev et al., 2022) and MOABB (Jayaram, Barachant, 2018), as well as other public BCI datasets. Therefore, this thesis endeavors to develop a deep learning interface within BBCPy toolbox that supports the PyTorch framework, delivering an easy and rapid design for experimentation and fast prototyping with deep learning models.

1.3 Outline of Work

This thesis is organized as follows:

- Chapter 2 provides a brief introduction to the basic concepts and theories behind EEG, BCI, and machine learning, with a particular emphasis on deep learning.
- Chapter 3 offers an intensive description of the dataset used in this study, including experimental design and data characteristics.
- Chapter 4 introduces the BBCPy toolbox and details the integration of deep learning capabilities.
- Chapter 5 describes the methodologies and procedures applied to construct an offline BCI classifier using an inter-subject approach.
- Chapter 6 presents the results and discusses the findings of the research.

- Finally, Chapter 7 concludes the thesis, summarizing the work and highlighting the limitations and areas for future research.

2 Theoretical Background

This chapter provides a comprehensive overview of the fundamental concepts and technologies underpinning brain-computer interfaces (BCIs), with a focus on electroencephalography (EEG), classical BCI algorithms used for MI-BCI classification, and deep learning techniques. The initial section offers a concise introduction to EEG, elucidating its role and significance in the context of BCIs. Subsequently, we illuminate the principal concepts and operational principles of BCIs, laying the groundwork for understanding their functionality and applications. Following this, classical approaches for decoding brain signals are explored, offering insights into traditional methods. The chapter closes with an in-depth examination of deep learning, including an overview of various network architectures and their components.

2.1 EEG

Historical Background. EEG is a non-invasive imaging technique that measures the electrical activity of the brain through electrodes placed on the scalp. This method was pioneered by Hans Berger, a German psychiatrist, who in 1929 used Einthoven's string galvanometer to measure and document human brain activity. Berger discovered different frequency ranges within these oscillations, naming them as the α wave (8–12 Hz), observed in resting states, the β wave (12–30 Hz), associated with attentive states, the γ wave (>30 Hz to 100Hz), and the δ wave (below 4 Hz) (Berger, 1929).

Measurement Mechanisms. EEG records bioelectrical potentials on the scalp, which originate at the micro-level from ions moving along neurons, causing momentary polarization. This leads to electrical charging of billions of interconnected

neurons, resulting in a nerve impulse or spike. At the macro-level, due to its poor spatial resolution and potential weakness, EEG cannot record individual neuron oscillations. However, the sum of synchronous firings of a large number of neurons with similar spatial orientation can be detected as macroscopic oscillations in the form of field potentials 2.1.

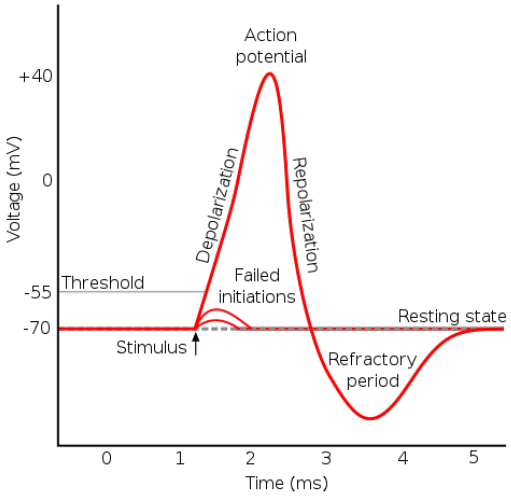


Fig. 2.1: Action potential. (Wikipedia, 2023).

Hardware. EEG systems employ caps with electrodes for placement on the scalp. These electrodes are primarily of two types: dry electrodes, utilizing spring-loaded pins for contact, and wet electrodes, which require conductive gel to minimize impedance and enhance signal integrity. Active electrodes are also utilized, which amplify EEG signals and mitigate noise and artifacts, in contrast to passive electrodes that simply relay raw brain voltage fluctuations. EEG caps vary in the number of electrodes and are arranged according to international norms. A prevalent setup is the 64-electrode configuration adhering to the 10-20 system, illustrated on the right panel of Figure 2.2. In this system, the '10' and '20' numerals indicate the percentage of the total front-to-back or right-to-left inter-electrode distance across the skull.

Electrode designations generally correspond to the brain region they overlay, as shown on the left side of Figure 2.2. These are labeled from the front to the back of the brain as follows: **Fp** for the pre-frontal or frontal pole, **F** for the frontal, **C** for the central line of the brain, **T** for the temporal, **P** for the parietal, and **O**

for the occipital regions. Electrodes straddling two regions bear composite labels reflecting this boundary, arranged from front to back. Notably, **M** and **A** denote mastoids or earlobe electrodes, which typically act as reference points due to their reduced sensitivity to artefacts, thus potentially enhancing the Signal-to-Noise Ratio (SNR).

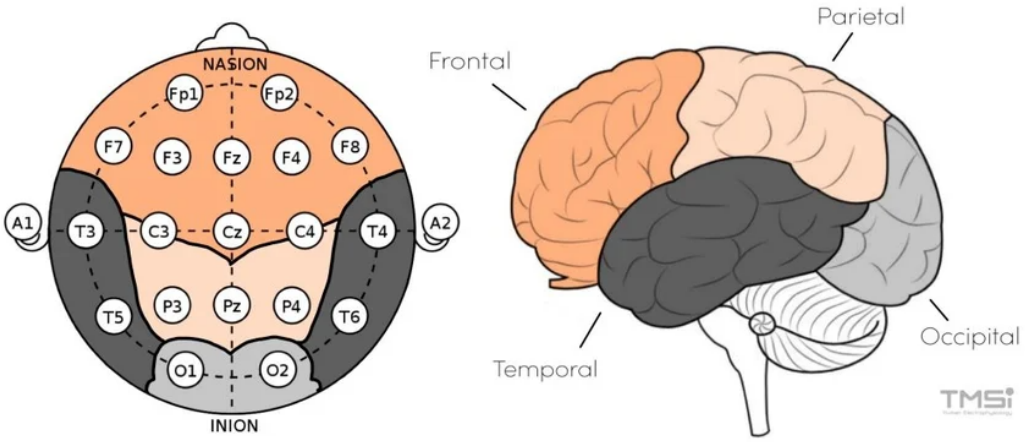


Fig. 2.2: The electrode layout of the 10-20 system (left) and corresponding brain regions (right). (TMSi, 2023).

Challenges. EEG is extensively researched and widely used due to its non-invasive nature, lightweight equipment offering portability (compared to MEG), and lower cost. Unlike fMRI, it does not require special confinement. Despite its advantages, EEG presents several drawbacks, including the time-consuming setup for electrode placement and calibration. It is susceptible to various challenges such as artefacts from muscle movement and eye blinking, biological background activity, drift currents, electromagnetic induction (e.g., 50 Hz power line noise), interface noise, and electrode impedance issues, particularly with dry electrodes.

Typically, the recorded EEG signal has a low amplitude, around 50-100 1 μV, while the tension of a single neuron's membrane ranges from -70 to 40 5 mV. EEG primarily measures electrical fields originating from the cerebral cortex's pyramidal cell dendrites, see (a) in 2.3. Most pyramidal cells fire asynchronously, leading to irregular polarization and cancellation of most potentials. However, synchronous polarization of a small group can strengthen the potential and produce a visible

EEG signal, as illustrated in Figure 2.3 (Hebb Rule: "What fires together, wires together" Morris, 1999).

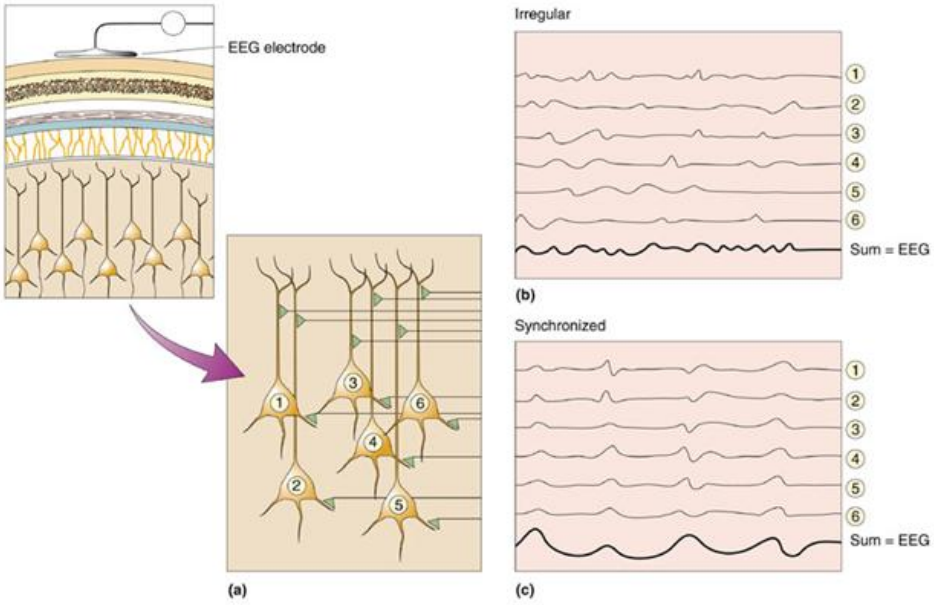


Fig. 2.3: Neocortical pyramidal cells and EEG Signal Composition. Panel (a) represents the layered structure of the cerebral cortex and the positioning of pyramidal cells. Panel (b) illustrates the effect of irregular, asynchronous firing of neurons, which typically yields a negligible cumulative EEG signal. Panel (c) contrasts this by displaying synchronized neuronal firing, where the temporal alignment of action potentials from multiple neurons produces a strong EEG signal that can be detected by surface electrodes. (*Neurons and Neural Networks* n.d.).

EEG's poor spatial resolution is attributed to the complex cortical surface with folds and grooves, leading to multi-directional propagation of potentials, known as "paradoxical lateralization." Furthermore, the "conduction volume" effect, resulting from different medium conductivities between the EEG electrode and the cerebral cortex, weakens the signal (see Figure 2.4 for brain structure).

Data Acquisition Techniques. To mitigate challenges and enhance the quality of EEG data acquisition, a series of systematic steps are employed. Initial noise reduction is achieved through an analog pre-filtering process, which includes using a common mode rejection ratio (CMRR) to eliminate ubiquitous noise sources such as power line frequencies. This is crucial, as such noises can contaminate the EEG signals, leading to inaccurate readings (**empty citation**). Subsequent am-

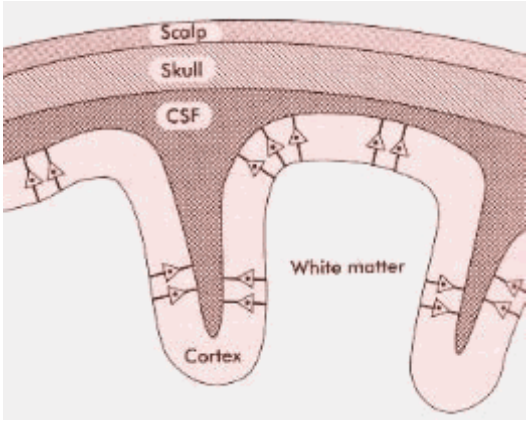


Fig. 2.4: Cross-Sectional View of Human Brain Layers. This diagram shows the layered architecture of the head, illustrating the scalp, skull, cerebrospinal fluid (CSF), gray matter (cortex), and white matter. It also highlights the complex topography of the brain, including the cortical folds and grooves. (Aguiar et al., 2000).

plification of the EEG's inherently weak signals is accomplished using a cascade amplifier block. This block is equipped with a differential gain of 10^4, effectively bolstering the signal strength without amplifying the noise at the same rate. This differential gain is critical; it ensures that the signal of interest is enhanced relative to the noise, thus improving the Signal-to-Noise Ratio (SNR).

The final stage involves the quantization of the analog signal into a digital format through an analog-to-digital converter (ADC). This conversion is vital for digital signal processing and storage. The sampling rate for this conversion typically ranges between 256 and 512 Hz, tailored to the specific requirements of the Brain-Computer Interface (BCI) application at hand. A low-pass filter is also implemented as part of the anti-aliasing protocol to comply with Nyquist's sampling theorem. This theorem is fundamental to signal processing, stipulating that the sampling rate must be at least twice the maximum frequency present in the signal to accurately reconstruct the original analog signal from its digital counterpart. By adhering to this principle, the integrity of the EEG data is preserved, ensuring that the digital representation remains true to the neural activity it reflects.

2.2 Brain-Computer Interface

Definition of BCIs. Brain-computer interfaces (BCIs) are innovative systems that convert brain activity into output for various applications, such as communication (e.g., P300 speller, brain-to-text via handwriting), medical uses (e.g., prosthetic control, stroke or epilepsy monitoring), wellness (e.g., sleep control, stress reduction), and entertainment (e.g., virtual reality, gaming) (Rashid et al., 2020).

History of BCI. The field of BCI, although still evolving, is not new. It was first introduced in 1973 by Jacques J. Vidal, who conducted the inaugural invasive BCI experiment, training monkeys to control specific brain regions using conditioning methods (Vidal, 1973). The first human non-invasive BCIs emerged in the late 1980s with Farwell and Donchin's P300 speller paradigm. Voluntary control of slow cortical potentials (SCP) was documented by Elbert et al. in 1980, leading to the development of SCP-based BCIs.

BCI Paradigms. BCI paradigms are designed to elicit specific voluntary behaviours, creating distinguishable brain activity that serves as features for BCI applications. Examples include:

- P300 paradigm: Based on the P300 event-related potential (ERP) pattern.
- Steady-State Visually Evoked Potentials (SSVEP) paradigm: Utilizes repetitive visual stimuli (RVS) to stimulate EEG oscillation in the occipital lobe.
- SCP paradigm: Characterized by slow shifts in cortical brain activity.
- Motor Imagery (MI) paradigm: Involves sensorimotor rhythms (SMRs) related to motor tasks.

Motor System. Understanding the Motor Imagery (MI) based Brain-Computer Interface (BCI) requires a fundamental grasp of the neural pathways involved in motor movement. Voluntary motor control involves both cortical and subcortical brain areas. The process typically originates in the primary motor cortex (M1), with signals transmitted to peripheral muscles through the central nervous system (CNS), which includes the brainstem and spinal cord. Moreover, the effective

execution of precise and coordinated tasks necessitates the collaboration of various sensory and associative cortical areas.

Motor Cortex. The motor cortex encompasses several key areas, as depicted in Figure 2.5: the primary motor cortex (M1), the premotor cortex (PMA), the posterior parietal cortex, and the supplementary motor area (SMA). Located in the precentral gyrus of the frontal lobe, M1 is responsible for initiating voluntary movements. The PMA, situated anterior to M1, is involved in planning movements and providing sensory guidance. The SMA, positioned anteriorly to M1, plays a role in postural stabilization, coordination of bilateral movements, and sequencing of complex actions (*Motor cortex* 2024). The primary motor cortex is organized somatotopically, meaning different muscle groups are controlled by distinct regions of M1 (*Motor cortex* 2024). For instance, the representation areas for the feet and legs are located in the medial wall of the precentral gyrus, while those for the hands and face are on the lateral side 2.6.

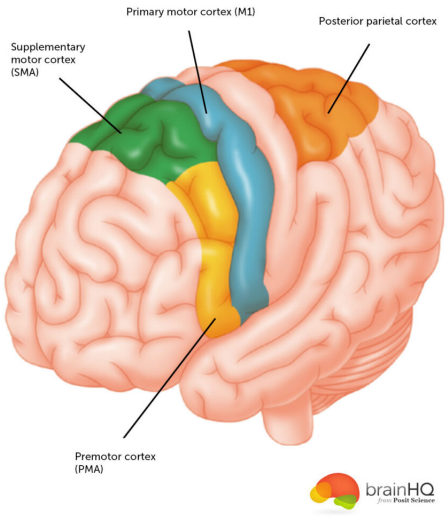


Fig. 2.5: Motor system structure on the brain .

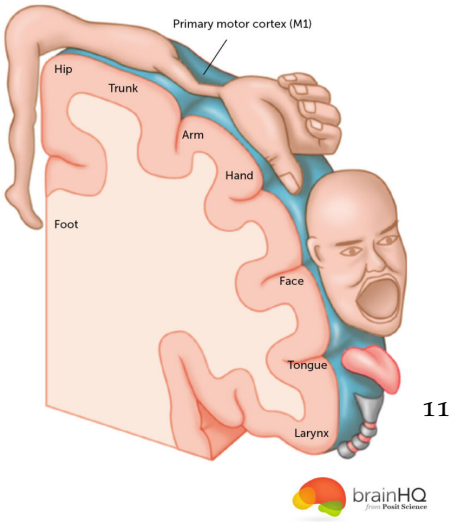


Fig. 2.6: Somatotopic arrangement.

2.2.1 Motor Imagery Driven BCI

Overview. Motor imagery-driven BCIs capitalize on the neural mechanisms underlying the imagination of motor actions without actual movement. This paradigm taps into the neural substrates of motor planning and execution, particularly engaging the sensorimotor cortex. By imagining specific movements (e.g., hand clenching), individuals generate distinct patterns in the sensorimotor rhythms (SMRs) that can be detected and decoded by BCIs. This approach is particularly beneficial for individuals with motor impairments, offering a means of communication and control without physical movement.

2.2.2 Sensorimotor Rhythm

BCIs have evolved by incorporating a variety of experimental and training paradigms. These paradigms are designed to extract certain voluntary behaviors from users by guiding them through specific mental states. In EEG-based BCIs, a variety of paradigms exist, each with different applications and research objectives. The most common EEG paradigms include:

- P300 paradigms, which rely on the P300 event-related potential (ERP) pattern.
- SSVEP paradigms, based on the flickering frequency that stimulates the eye, observable in EEG oscillations from the occipital lobe.
- MI paradigms, which focus on specific frequency bands relevant to motor imagination and execution.

This thesis focuses on the latter paradigm, a Motor Imagery ***M**otor **I**magery* (MI) refers to the mental simulation of muscle activation, involving the imagination and execution of limb movement. It stimulates, among other areas, the brain regions responsible for the sensory and motor cortex (Scherer, Vidaurre, 2018).

The discriminant information in Motor Imagery (MI) is primarily observed in the sensorimotor rhythm (SMR) range of 8 to 13 Hz. These SMR modulations

are detected in the primary motor cortex area, with their specific locations corresponding to the brain's somatotopic arrangement, as illustrated in Figure 2.6. For example, activities related to hand or finger movements elicit responses in the primary motor cortex, known as M1. This region, situated in the frontal lobe of the brain, is crucial for planning and executing movements, as detailed in Figure 2.5 (Yuan, He, 2014).

2.3 Feature extraction methods in processing stage

2.3.1 Covariance Matrix

For multivariate time-series data, covariance matrices are particularly useful due to their ability to generate features that capture the spectral variability of the signal. In EEG-based BCI applications, covariance matrices construct an $N_c \times N_c$ matrix (where N_c is the number of channels or electrodes). These matrices summarize the spatial distribution of signal and noise power across the channel space, represented by the diagonal entries, and the spatial correlations between all EEG channels, indicated by the non-diagonal entries, as depicted in Figure 2.7.

The covariance matrix of the signal $\mathbf{X} \in \mathbb{R}^{N_c \times N_s}$ can be estimated using the equation

$$\mathbf{C} = \frac{\mathbf{X}\mathbf{X}^T}{N_s - 1} \in \mathbb{R}^{N_c \times N_c} \tag{2.1}$$

where N_c represents the number of EEG channels, N_s is the number of time samples. It is important to note that there exist multiple approaches to estimate a covariance matrix (Beltrachini et al., 2010).

2.3.2 Common Spatial Patterns

Common Spatial Patterns (CSP) is a spatial filtering tool that applies decomposition technique on two populations of multivariate signals into a set of signal patterns.

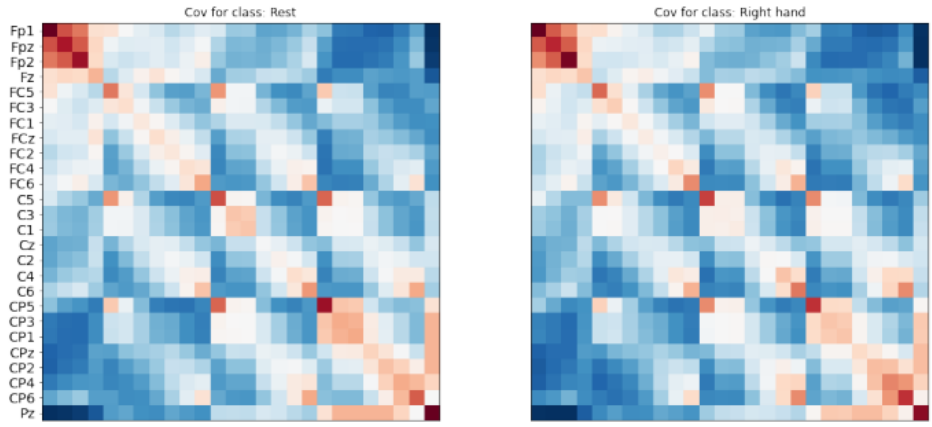


Fig. 2.7: Covariance matrices for two MI states (rest and right hand).

In the context of EEG Motor-Imagery Brain-Computer Interface (MI-BCI) systems, the CSP yields discriminative features between two mental states that can later be classified using a supervised approach of labeled EEG signals (Blankertz et al., 2008). We denote $\mathbf{X} \in \mathbb{R}^{N_C \times N_S}$ as a single trial of EEG signal, where N_C represents the number of EEG channels and N_S the number of time samples.

Binary scenario. For the scenario of two different MI classes, the following steps describe the CSP algorithm:

1) Band-pass filtering: select a band rhythm by using band-pass filter on the EEG signal.

2) Covariance matrix estimation: the estimated normalized covariance matrix Σ of each trial signal X is expressed as follows:

$$\Sigma = \frac{XX^T}{tr(XX^T)} \in \mathbb{R}^{N_C \times N_C} \tag{2.2}$$

3) Trial-wise averaging: average the covariance matrix of all trials of each class (c = {+, -}).$|C|$ denotes the size of each class set $(c \in \{+, -\})$

$$\Sigma^{(c)} = \frac{1}{|C|} \sum_{i \in C} X_i X_i^T \quad (c \in \{+, -\}), \tag{2.3}$$

4) Optimization problem: the diagonalization expression of the covariance matrices, is expressed as follows:

$$W^T\Sigma^{(+)}W = \Lambda^{(+)}, \quad W^T\Sigma^{(-)}W = \Lambda^{(-)} \tag{2.4}$$

where **W** consisting of the generalized eigenvectors $w_j(j = 1,...,C)$ and Λ is the diagonal matrix that verifies the identity $\Lambda^{(+)} + \Lambda^{(-)} = I$. Thus, to obtain the optimal spatial filter W, we need to solve the ***G**eneralized **E**igenvalue **D**ecomposition* (GEVD) problem written in the equation 2.5, which can be reformulated as a maximization problem in equation 2.6 and 2.7. $J(w)$ is known as the Rayleigh quotient.

$$\Sigma^{(+)}W = (\Sigma^{(-)} + \Sigma^{(+)})W\Lambda \tag{2.5}$$

$$w^* = \underset{w}{\operatorname{argmax}} J(w) \tag{2.6}$$

$$J(w) = \frac{w^T\Sigma^{(+)}w}{w^T\Sigma^{(-)}w} \tag{2.7}$$

The obtained generalized eigenvectors w_j are then sorted by magnitude and m pair are selected, where the large positive ones correspond to the class $(+)$ and the large negative ones correspond to the class $(-)$. They form the spatial filter $\mathbf{W} \in \mathbb{R}^{N_C \times 2m}$ by stack them horizontally. Finally, the transformed signal can be computed as follows in equation 2.8

$$X_{csp} = W^T X \in \mathbb{R}^{2m \times N_S} \tag{2.8}$$

5) CSP features: The CSP features are then normalized and transformed into log-variance space, as denoted in equation 5.2 as vector f with dimension $N_F = 2m$ components.

$$f = \log\left(\frac{Var(X_{csp})}{\|Var(X_{csp})\|}\right) \in \mathbb{R}^{N_F} \tag{2.9}$$

Multi-class scenario. To extend the CSP framework to the multi-class problem, two strategies are commonly adopted. The first strategy considers the CSP multi-class as a binary problem and adopts the one-versus-rest scheme that

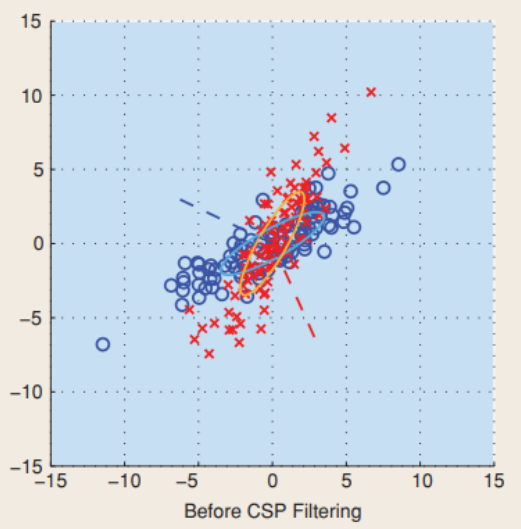


Fig. 2.8: Before CSP filtering, blue and red ellipsoids represent the class-conditional covariance matrices, with dashed lines indicating the directions of the two CSP projections..

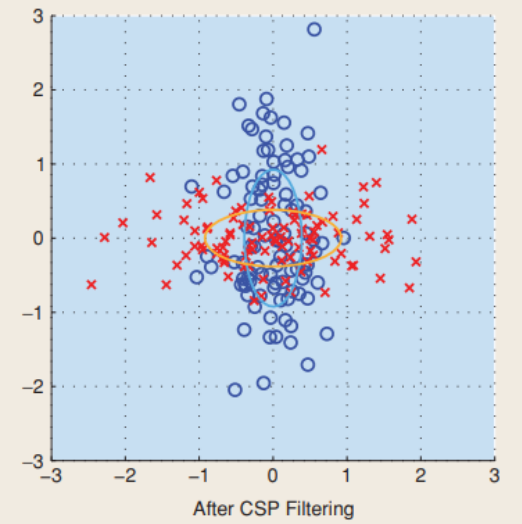


Fig. 2.9: After CSP filtering, the variance along the horizontal axis is maximized for the red class and minimized for the blue class, whereas along the vertical axis, the situation is reversed, demonstrating the CSP's efficacy in enhancing class separability..

Fig. 2.10: Effect of CSP Filtering on Signal Distributions (Blankertz et al., 2008)..

leads to $N_F = 2mN_L$ features or the all combination scheme that generates $N_F = mN_L(N_L - 1)$ features, where N_F denotes the number of filters, and N_L the number of classes (Blankertz et al., 2008). The second strategy involves applying the joint approximate diagonalization (JAD) to find a transformation matrix of the spatial filter $\mathbf{W} \in \mathbb{R}^{N_C \times N_C}$. JAD aims to diagonalize the covariance matrices Σ for N_M different classes of EEG signal $\mathbf{X} \in \mathbb{R}^{N_C \times N_S}$, as illustrated in the equation 2.10

$$\mathbf{W}^T \mathbf{\Sigma}_i \mathbf{W} = \mathbf{D}_i, \quad i = 1, \ldots, N_M \tag{2.10}$$

,where $\mathbf{D} \in \mathbb{R}^{N_C \times N_C}$ is the corresponding diagonal matrices. The motivation for using JAD for multi-class CSP is rooted in the notion that CSP for two classes can be conceptualized as diagonalizing two covariance matrices (Grosse-Wentrup, Buss, 2008).

2.3.3 Riemann Geometry

Riemannian Geometry studies surfaces in curved spaces, generalizing notions and concepts from the classical theory of curves and surfaces in Euclidean space. The motivation for using Riemannian geometry in spatial filters, which are expressed as spatial covariance matrices, stems from the fact that these matrices are ***S**ymmetric **P**ositive **D**efinite* (SPD). A covariance matrix is inherently symmetric and, given sufficient data for its estimation, will be positive definite. Typically, the covariance matrix is estimated in Euclidean space, which is prone to approximation errors. Therefore, applying Riemannian geometry, which minimizes the error in covariance estimation, is advantageous. Generally, there are two variations of Riemannian approaches: direct use of the SPD manifold and projection onto the tangent space (Xu et al., 2020).

In Riemannian geometry, a geodesic represents the shortest path between two points on a manifold $\mathcal{M}$, analogous to a straight line in Euclidean space. The length of this geodesic is the Riemannian distance between the points. This distance can be expressed using the following equation 2.11:

$$d_R(\mathbf{C}_1, \mathbf{C}_2) = \| \log(\mathbf{C}_1^{-\frac{1}{2}} \mathbf{C}_2 \mathbf{C}_1^{-\frac{1}{2}}) \|_F, \tag{2.11}$$

where $\mathbf{C}_1$ and $\mathbf{C}_2$ are covariance matrices and F denotes the Frobenius norm (Congedo et al., 2014).

The concept of a Riemannian center of mass, or Fréchet mean, is used to define a central point among a set of points on $\mathcal{M}$, which minimizes the sum of squared distances to all points in the set. It is defined as:

$$\bar{\mathbf{C}} = \arg\min_{\mathbf{C}} \left(\frac{1}{I} \sum_{i=1}^{I} d_R^2(\mathbf{C}, \mathbf{C}_i) \right), \tag{2.12}$$

Here, I denotes the number of trials of the selected class, and the center of mass represents the average distance between trials of that class within the tangent space.

2.3.4 Tangent Space

To visualize this, consider Figure 2.11, which depicts a Riemannian manifold $\mathcal{M}$, a point Ω on $\mathcal{M}$, and the tangent space $T_\Omega\mathcal{M}$ at Ω. The tangent vector V begins at Ω and corresponds to a point Φ on $\mathcal{M}$ via the unique geodesic that passes through Ω and Φ.

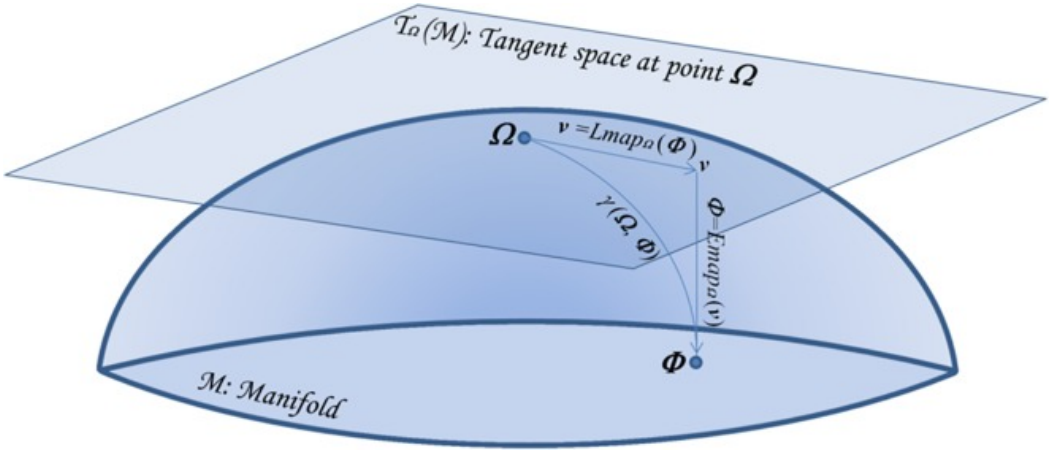


Fig. 2.11: Riemann manifold.

The geodesic curve $\gamma(\Omega \to \Phi)$ is calculated as follows, where β serves as the arc length parameter:

$$\gamma(\Omega \to \Phi) = \Omega^2(\Omega^{-\frac{1}{2}}\Phi)^{\beta\Omega^2}, \quad \beta \in [0,1], \tag{2.13}$$

When $\beta = 0$, we are at Ω; when $\beta = 1$, we arrive at Φ; and when $\beta = 0.5$, the point represents the geometric mean of the two points (Congedo et al., 2014).

The mapping functions known as the exponential and logarithmic maps facilitate the transition between points on $\mathcal{M}$ and vectors in the tangent space $T_\Omega\mathcal{M}$. The exponential map is denoted by $\Phi = \mathrm{Exp}_\Omega(V)$ and is defined as:

$$\Phi = \mathrm{Exp}_\Omega(V) = \Omega \exp\left(\Omega^{-\frac{1}{2}} V \Omega^{-\frac{1}{2}}\right)^{\frac{1}{2}}, \tag{2.14}$$

Conversely, the logarithmic map, denoted by Lmap_Ω, inverts this process:

$$V = \text{Lmap}_\Omega(\Phi) = \Omega \log\left(\Omega^{-\frac{1}{2}}\Phi\Omega^{-\frac{1}{2}}\right)^{\frac{1}{2}}, \tag{2.15}$$

which projects the geodesic $\gamma(\Omega \to \Phi)$ back onto a tangent vector V in $T_\Omega\mathcal{M}$, as expressed in Equation 2.15.

2.4 Classification stage

In supervised learning, the objective of classification tasks is to determine the category to which a given set of inputs belongs, selected from a predefined set of classes. This process necessitates prior knowledge, which is acquired through a training phase. During this phase, the model learns to assign classes based on patterns discerned from the training data.

A variety of classification methods have gained prominence in the BCI applications. In this section, we will introduce the Support Vector Machine (SVM) classifier, a powerful and widely used model for BCI classification tasks.

2.4.1 Support-Vector Machine

The Support Vector Machine (SVM) is one of the most classical classification algorithms used for supervised learning. It solves the classification problem by introducing the concept of margin, as Figure 2.12 demonstrates, which is defined as the smallest distance between the decision boundary and the samples (Bishop, 2006).

In the binary case, for a multi-dimensional feature vector $\mathbf{x} \in \mathbb{R}^D$, the SVM classifier can be expressed as in Equation 2.16:

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b \tag{2.16}$$

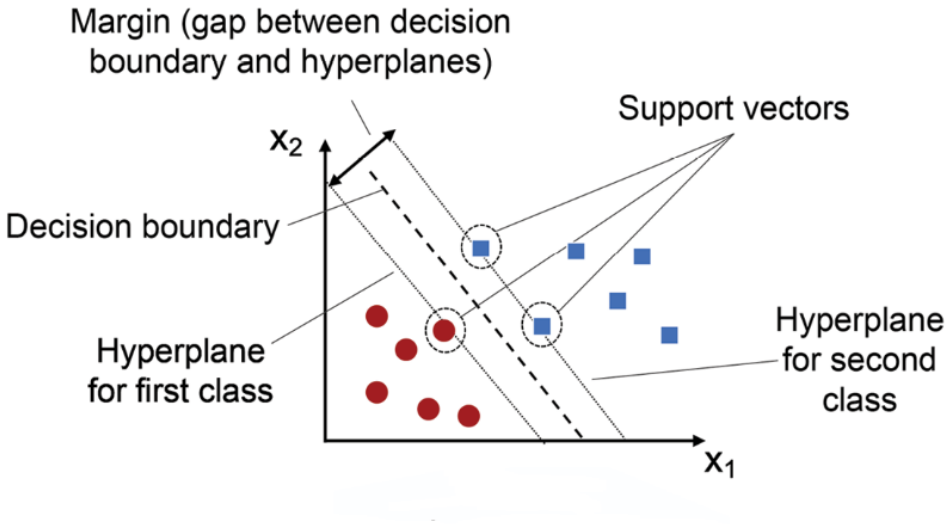


Fig. 2.12: SVM margins for binary classification.

where $\mathbf{w} \in \mathbb{R}^D$ is the weight vector, and b is the bias scalar. The function f aims to map the input feature vector $\mathbf{x}$ into a binary outcome, either $+1$ or -1, depending on the function sign.

The hyperplane that separates between the two classes is then obtained by maximizing the margin of the hyperplane. The margin is given by the perpendicular distance to the nearest datapoint $\mathbf{x}_i$ and can be expressed in 2.17

$$\frac{|\mathbf{w}^\top \mathbf{x}_i + b|}{||\mathbf{w}||}, \tag{2.17}$$

where y_i represents the data labels. In order to obtain the maximum distance, the optimization equation in ref needs to be optimized with respect to $\mathbf{W}$ and b

$$argmax_{\mathbf{w},b} \left\{ \frac{1}{||\mathbf{w}||} \min_k [y_k \cdot (\mathbf{w}^\top \mathbf{x}_k + b)] \right\} \tag{2.18}$$

Such a quadratic optimization problem is solved using the Lagrange multiplier theorem (Bishop, 2006). This leads to the dual formulation, which aims to find the optimal values α_i^* such that :

$$\max_{\alpha \in C} \left\{ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T \mathbf{x}_j \right\} \quad \text{s.t.} \quad \alpha_i \geq 0 \text{ and } \sum_{i=1}^{N} \alpha_i y_i = 0 \tag{2.19}$$

where N is the total number of training examples, C is a penalty (regularization) parameter and α_i is the i-th Lagrangian multiplier. C can be seen as a trade-off parameter between misclassification rate and margin maximization, a lower value allows for more classification errors but a smaller margin and vice versa.

Finally, the hyperplane equation is expressed as

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i^* y_i \mathbf{x}_i^T \mathbf{x} + b \tag{2.20}$$

2.5 Artificial Neural Networks

Artificial Neural Networks (ANN) have gained significant attention recently, partly due to advancements in hardware, especially computational power. Furthermore, improvements and developments in algorithms have accelerated the training and evaluation of ANNs. Inspired by the biological brain with its network of billions of neurons, the cornerstone of ANNs is the artificial neuron, as depicted in Figure 2.13. ANNs can be conceptualized as directed graphs, where nodes are artificial neurons, and directed edges are synaptic connections. Each artificial neuron receives inputs from its neighboring neurons via weighted connections. These inputs are aggregated and then transformed by a non-linear activation function, commonly known as a transfer function, before being transmitted to the next neuron. Throughout the training phase, the weights are iteratively updated until the learning algorithm achieves convergence.

Artificial neurons are typically arranged in layers. These layers are then organized in various configurations and quantities to form a neural network. When a network consists of only one hidden layer between the input and output layers, it is referred to as a shallow network. In contrast, a deep neural network is constructed with multiple hidden layers, which can drastically increase the number

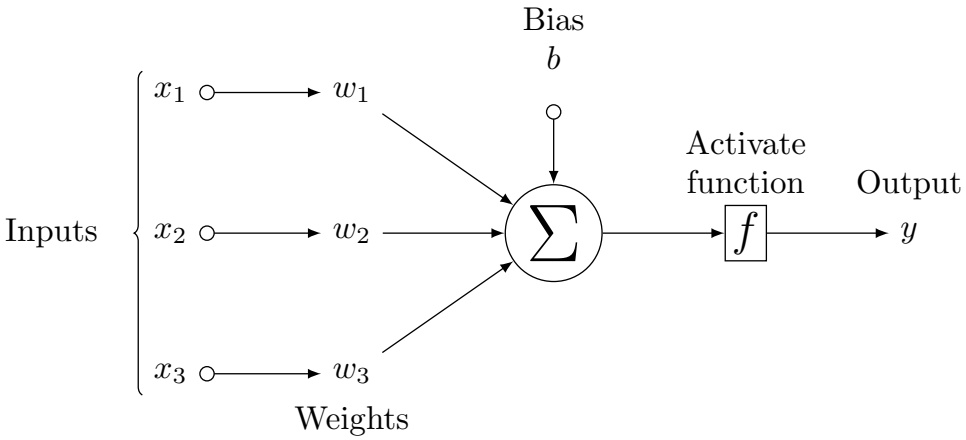


Fig. 2.13: Artificial neuron.

of neurons. There are multiple types of artificial neural networks, each with different structures and designs, hence intended for different tasks. The following subsections will illustrate the most important neural network in the deep learning domain, including their characteristics and components.

2.5.1 Feedforward Neural Networks

Feedforward neural networks represent the most fundamental and straightforward architecture among ANN. These networks consist of a series of layers, each containing a specific number of neurons. Information in these networks flows unidirectionally, starting from the input layer and culminating at the output layer. In scenarios where each neuron in a layer is interconnected with all neurons in adjacent layers, the network is classified as a fully-connected or dense network. This network structure can be effectively illustrated as shown in figure 2.14.

The ability of neural networks to handle nonlinearity is predominantly attributed to the activation functions. These functions are critical for enabling the network to learn complex patterns. There is a range of activation functions commonly employed in neural networks, such as the sigmoid, softmax, ReLU (Rectified Linear Unit), and tanh (hyperbolic tangent). The choice of activation function typically

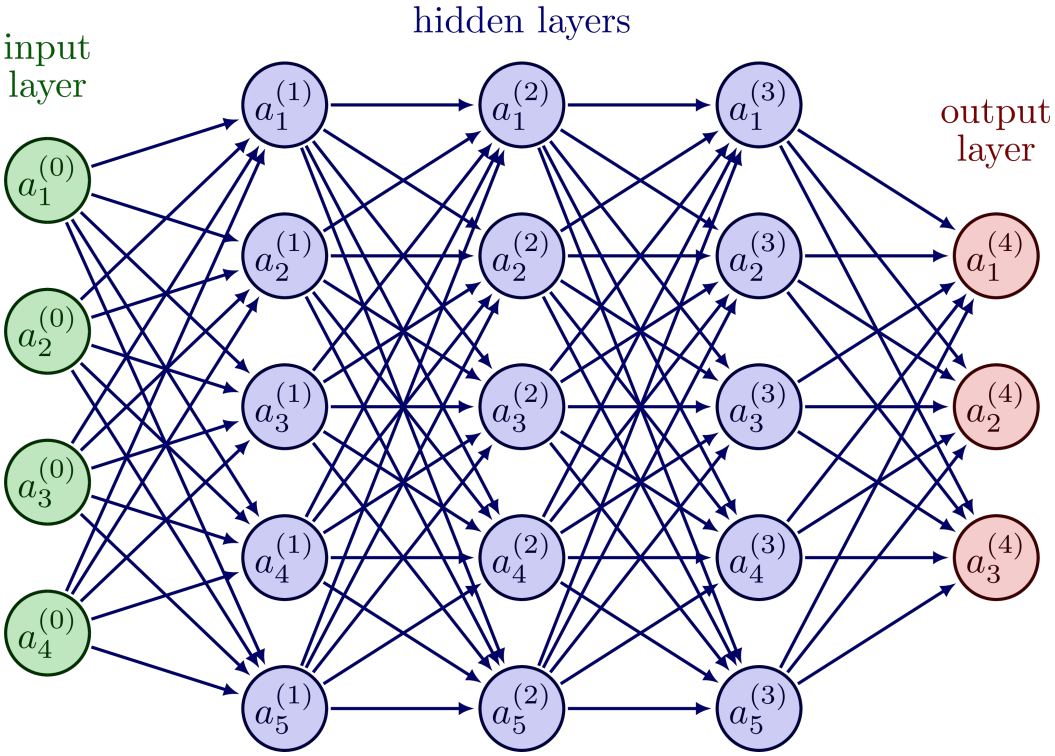


Fig. 2.14: Feedforward and fully-connected network..

depends on the specific requirements of the learning task and the characteristics of each layer. Illustrations of these activation functions and their typical applications are presented in figure 2.15.

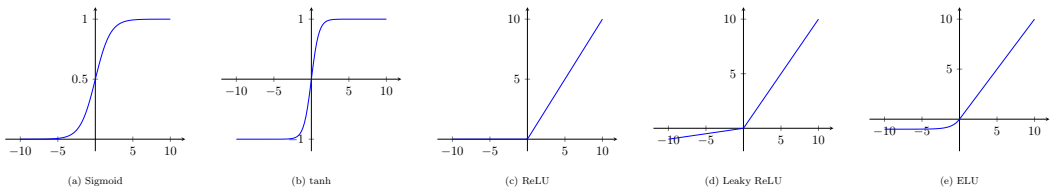


Fig. 2.15: Collection of activation functions.

2.5.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are inspired by the visual cortex of animals and were first implemented in the LeNet network by LeCun et al. (LeCun et

al., 1989). Generally, CNNs comprise two distinct types of layers: convolutional layers and pooling layers, which are typically arranged in alternating sequences (as shown in figure 2.16). CNNs have gained significant prominence in the field of computer vision due to their ability to learn hierarchical feature representations from images. This includes understanding the spatial relationships between patterns, a key aspect of visual data processing.

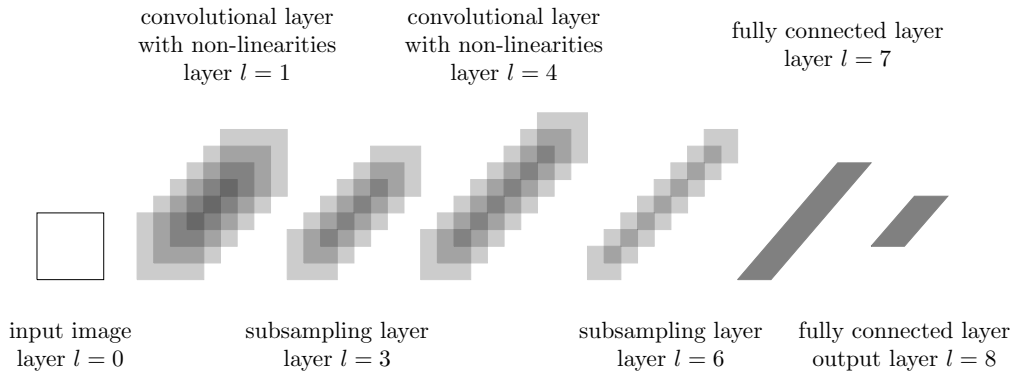


Fig. 2.16: Convolution Neural Nets (CNNs).

A fundamental component of CNNs is the receptive field, which introduces local connectivity and spatial arrangement to the network. The receptive field allows CNNs to focus on small segments of the input data at a time, capturing local features before integrating them into a global understanding. This section will provide a more detailed exploration of the main layers that constitute CNNs, highlighting how they work together to process and learn from visual data effectively.

The convolution layer (Goodfellow et al., 2016), as the cornerstone of CNNs, plays a pivotal role in extracting salient features from input data. It operates through a set of trainable parameters, commonly referred to as receptive fields or spatial filters, which are updated during the learning process. Each filter, typically of dimensions $K = [F \times F]$, can capture the full depth of the input volume, particularly in the case of 3D inputs. For instance, figure 2.17 demonstrates a typical filter with a size of 3×3. During the forward pass, this filter convolves along the input data I of shape 7×7, computing the dot product between the filter coefficients and the input window, subsequently generating an activation map.

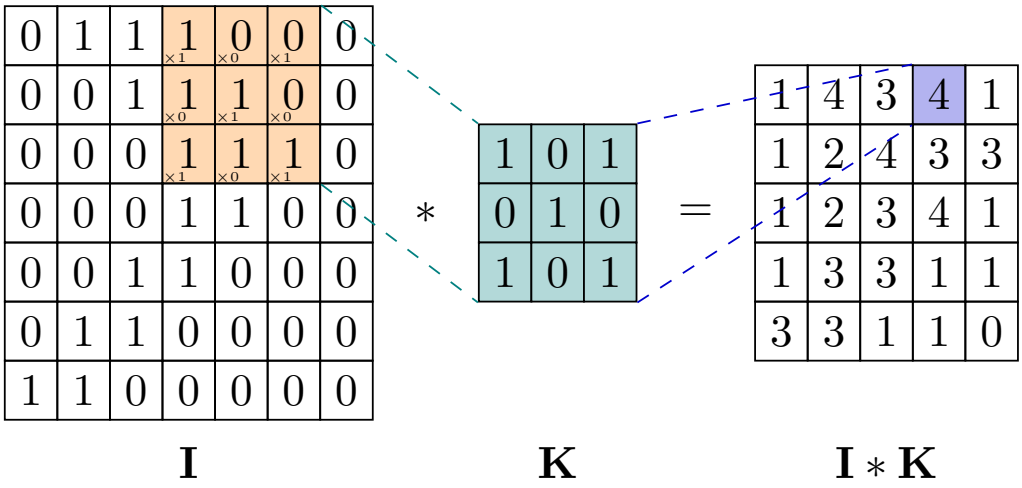


Fig. 2.17: Convolution operation.

Throughout training, the convolution layer learns to produce activation maps that encapsulate features, such as edges or colors. These feature maps are heavily influenced by the shape of the receptive field, which imposes a specific spatial arrangement on the input volume. These spatial arrangements are considered hyperparameters, including the depth K, which corresponds to the number of filters used. These filters observe the same region but with different sets of coefficients. The stride S translates to the sliding effect of the filter; a stride of 1 means the filter moves one unit at a time, while a stride of 2 indicates a two-unit jump. Zero-padding P is another important hyperparameter, designed to address the limited space available for the filter to operate on. By padding the input volume with zeros around the borders, it controls the spatial size of the output post-convolution. Lastly, the filter size K defines the size of the receptive field.

The equations in 2.21 summarize the relationship between these parameters. When a convolution layer operates on an input volume with size $W_1 \times H_1 \times D_1$, then the obtained output volume has the size $W_2 \times H_2 \times D_2$ and is calculated as follows:

$$W_2 = \frac{W_1 - F + 2P}{S} + 1 H_2 = \frac{H_1 - F + 2P}{S} + 1 D_2 = K \tag{2.21}$$

The Pooling Layer (Goodfellow et al., 2016) is commonly used to reduce the spatial size of the representation, which implies a significant reduction in the number of trainable parameters. This can lead to improved model training and avoidance of overfitting. There are various types of pooling layers, but the most utilized and promising is the max pooling. The figure 2.18 illustrates a max pooling operation with a 2×2 filter and a stride of 2, which downsamples every depth slice in the input by 2 along the width and height. Similar to the convolution layer, the dimensions of the output size $W_2 \times H_2 \times D_2$ are obtained by:

$$W_2 = \frac{(W_1 - F)}{S} + 1 H_2 = \frac{(H_1 - F)}{S} + 1 D_2 = D_1 \tag{2.22}$$

where F is the filter size and S is the stride of the pooling layer, and $W_1 \times H_1 \times D_1$ is the input volume shape.

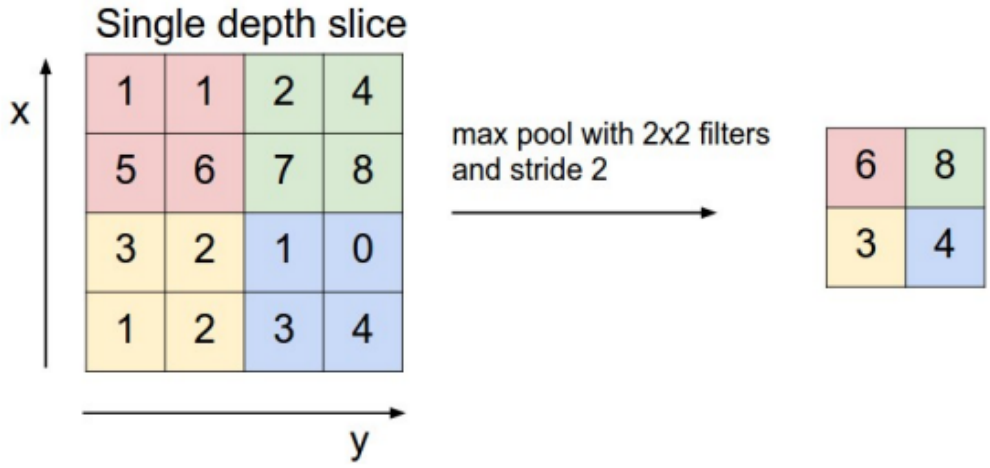


Fig. 2.18: Max pooling (*CS231n Convolutional Neural Networks for Visual Recognition* 2024).

2.5.3 Recurrent Neural Network

An important neural network that addresses the problem of information loss across time and the shortcomings of standard feedforward neural networks in utilizing prior knowledge from previous inputs is the ***R**ecurrent **N**eural **N**etwork* (RNN). RNNs offer a mechanism to retain prior information in a memory cell and

enable the retrieval of this information over time through loops (*Understanding LSTM Networks – colah's blog* 2024).

The fundamental unit of the RNN architecture is a cell, denoted by A, that has access to the input x_t, produces an output h_t, and has a connection that allows the passage of information forward to the neighboring cell. By concatenating multiple cells and linking them together, a chain-like structure is formed, which is known as the recurrent neural network. This structure is depicted in figure 2.19.

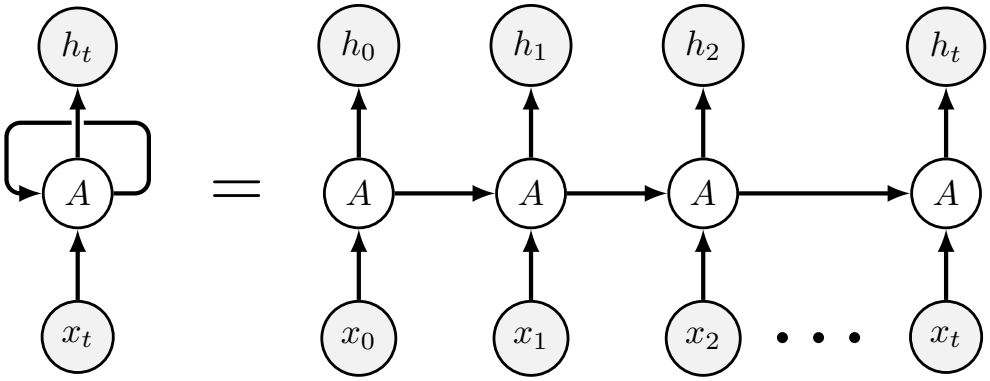


Fig. 2.19: Recurrent neural network.

There are variations of the RNNs, in which one tackles the problem of "long-term dependencies" due to the vanishing gradient issue. This is caused by a complex and large data dependency and therefore requires a long sequential chain. ***L**ong **S**hort-**T**erm **M**emory* (LSTM) is capable of overcoming this problem by introducing a more complex structure inside the cell A as depicted in the figure 2.20 (Goodfellow et al., 2016). To describe how LSTM operates, first, let us introduce the three tunable gates that build the LSTM cell:

- Forget gate: $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
- Input gate: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
- Output gate: $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$

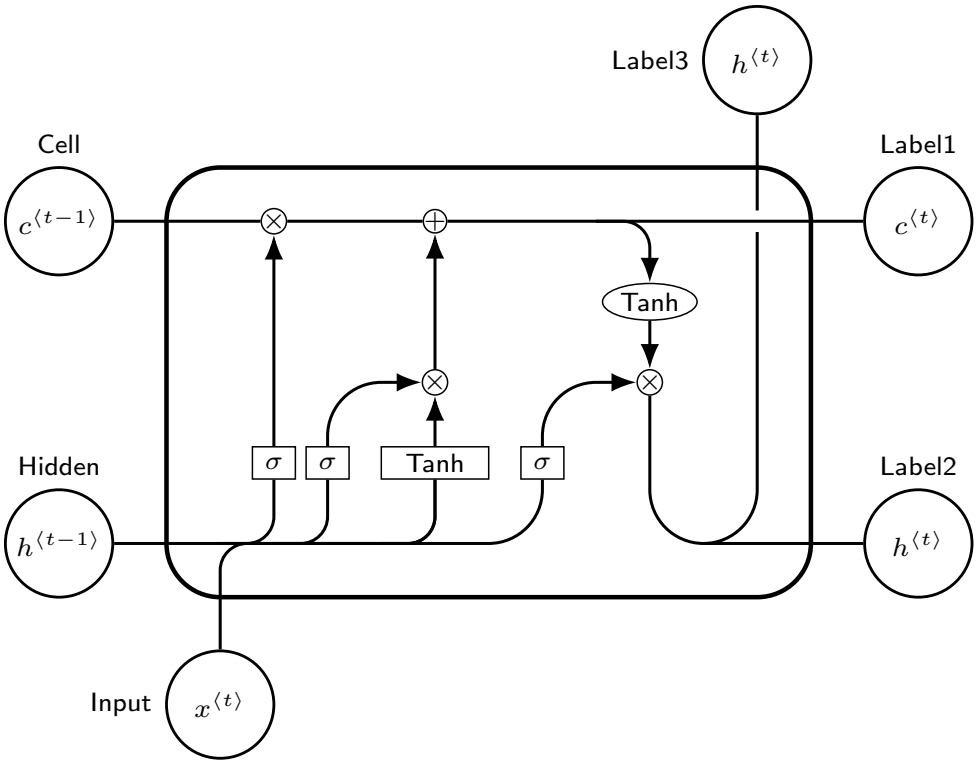


Fig. 2.20: LSTM cell.

let's now examine how these components dynamically interact within the LSTM workflow. This sequential process, encompassing the forget, input, and output gates, forms the core operational cycle that enables LSTMs to effectively capture temporal dependencies in data.

- **First step:** in the forget gate layer, decide what information we're going to discard from the cell state:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

- **Second step:** decide what new information is going to be stored in the cell state. This step includes an input gate layer that applies a sigmoid function:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

And a *Tanh layer* that creates a vector of new candidate values that could be added to the state:

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Finally, both layers are combined together to create an update of the new state:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- **Final step:** the output will be calculated based on the new state by applying a sigmoid layer:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

Subsequently, a hidden state is also computed based on the new state by applying the tanh function and multiplying it by the output of the sigmoid gate:

$$h_t = o_t * \tanh(C_t)$$

3 Dataset

This chapter elucidates the dataset collected from a BCI experiment focusing on sensorimotor rhythms, encompassing the experimental design, hardware configuration, and data acquisition techniques. It explains the procedural steps of the experiment, the structure and integrity of the data. It concludes by introducing the Percent Valid Correct (PVC) metric as the key measure of participant performance.

3.1 Dataset description

Overview. The dataset used in this study is based on 2-class MI-BCI paradigm, specifically focusing on the imagery of left and right-hand movements. It was collected by researchers from the University of Minnesota and Carnegie Mellon University. The primary objective of this research was to examine the influence of ***M**ind-**B**ody **A**wareness **T**raining* (MBAT), such as yoga or meditation, on the proficiency of participants in controlling a continuous sensorimotor rhythm (SMR) based live-feedback interface system (Stieger et al., 2021a). The study was conducted with two distinct groups of participants. The experiment investigated whether mindfulness-based stress reduction (MBSR) classes could enhance subjects' ability to control SMR-based BCIs, compared to those who did not receive such intervention. Subjects with less than 90% accuracy in 1D control during the initial session were considered for the study, to primarily focus on performance improvement. Post the initial session, subjects were divided into two groups: one attending an 8-week MBSR class and the other, a control group, waiting for the same duration before participating in 6 to 10 follow-up sessions. These sessions were designed to monitor changes in their BCI control proficiency (Stieger et al., 2021b).

The dataset is particularly valuable for deep learning applications, which typically require extensive data for effective training. It is recognized as one of the largest MI-BCI datasets ever recorded, with over 600 hours of data and more than 269,099 trials (Sannelli et al., 2019). This size is approximately 350 times larger than the combined data of BCI competition datasets 2a and 2b, and offers ten times more data than the next largest public MI dataset (Gwon et al., 2023). Consequently, it provides a significant benchmark for validating and developing new deep learning algorithms in the MI-BCI domain.

3.1.1 Experiment design

Hardware Configuration. The experimental setup utilized a high-density EEG cap equipped with 64 channels, adhering to the international 10-10 system (see Figure 3.1). The precise positioning of the cap was ensured with a margin of ±0.25 cm between the nasion (Nz), inion (Iz), and the left/right preauricular points (LPA and RPA) relative to the cap's Cz electrode, as illustrated in Figure 3.1. EEG data acquisition was conducted using SynAmps RT amplifiers and Neuroscan acquisition software (Compumedics Neuroscan, VA), depicted in Figure 3.2. The SynAmps RT amplifiers are designed with a feature to monitor the impedance of the electrodes, maintaining it within a range below 5 kΩ. Electrodes with impedance exceeding this threshold were identified and labeled as noisy, ensuring the quality of the recorded data. Additionally, the position of each electrode was precisely recorded using a FASTRAK digitizer from Polhemus.

Data Acquisition. The raw EEG data was digitized at a sampling rate of 1000 Hz. It underwent a bandpass filtering process ranging from 0.1 to 200 Hz, supplemented by a notch filter at 60 Hz to eliminate powerline interference. This filtering process is further detailed in Chapter 2, Section 3 **??**. Subsequently, the processed data was stored in the MATLAB-compatible .mat file format, facilitating offline analysis and research applications.

Participants. The dataset encompasses EEG data from 62 healthy adult subjects, split into two groups: 33 MBSR subjects (Age = 42 ± 15, Female = 26) and 29 control subjects (Age = 36 ± 13, Female = 23). The number of sessions per participant varied between 7 and 11, with each session comprising 450 trials. This

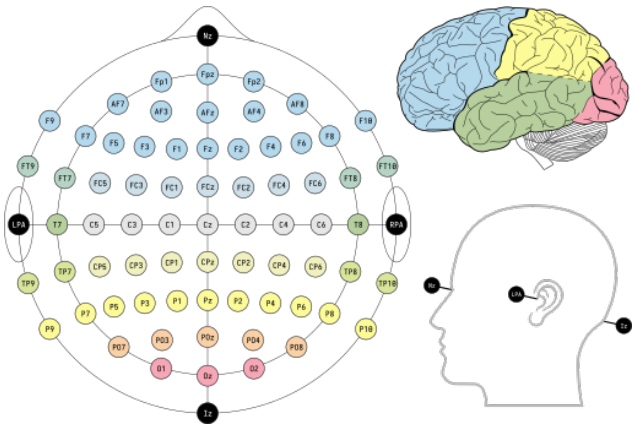


Fig. 3.1: International 10-10 system electrode placement for EEG monitoring, showcasing the standard positions across the scalp.(Wikimedia Commons, 2023).

Fig. 3.2: SynAmps RT amplifier and a high-density EEG cap. (Compumedics Neuroscan, 2024).

amounted to an average of 9.9 hours of EEG data per participant. The experiment spanned over four months, with an approximate commitment of 6 hours per week from each participant.

3.1.2 Experimental procedure

The experimental procedure was designed in alignment with the MI-BCI system, employing the BCI2000 platform for real-time cursor control (Schalk et al., 2004). Participants were instructed to navigate a virtual cursor from the center of the screen towards one of four targets, each corresponding to a distinct motor imagery task as depicted in Figure 3.3. Participants were positioned approximately 65 cm from a monitor screen. Prior to initiating the BCI tasks, specific instructions were communicated for each task. For example, participants were directed to imagine opening and closing their left or right hand to effectuate horizontal cursor movement, to envision the movement of both hands to move the cursor upwards, and to engage in a state of relaxation or mental clarity to move the cursor downwards (Stieger et al., 2021b).

The trials were categorized into 3 tasks based on the direction of cursor movement: left/right (LR) only, up/down (UD) only, or a combination of both in a 2D

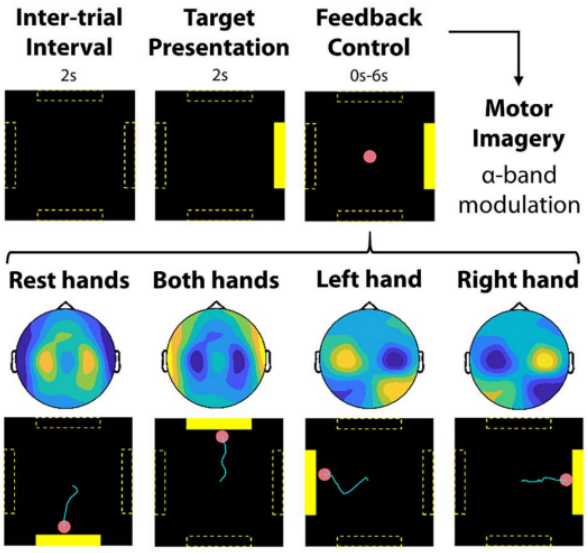


Fig. 3.3: Sequence of a BCI task trial: A 2-second inter-trial interval precedes the target presentation, guiding the participant where to direct the cursor. In the subsequent feedback control period, lasting up to 6 seconds, cursor movement is driven by alpha power changes in the bilateral motor cortices. Lateralized motor imagery leads to alpha power reduction (blue) in the contralateral cortex and slight increases (yellow) in the ipsilateral cortex, while bilateral imagery causes a general desynchronization compared to rest, governing both lateral and vertical cursor movements.(Stieger et al., 2021a).

movement (2D). Each experimental block consisted of three runs, with each run including 25 trials within a specific task category. Participants were allowed a rest period of 5–10 minutes following the completion of each block. After the rest break, participants were required to repeat the task sequence, thereby completing the second block. In total, each session comprised two blocks of three runs for each task category. This resulted in a total of 225 trials per task and an aggregate of 450 trials for the entire session, as depicted in Figure 3.4.

BCI recordings. The structure of each trial across all three tasks follows the same format, as illustrated in Figure 3.5. Each trial begins with a 2-second **inter-trial interval** during which the screen remains blank. This is followed by the appearance of a target, represented by a yellow bar, on one of the four designated locations on the screen for a duration of 2 seconds. This **visual cue** indicates to the participant where the cursor should be directed. Subsequently, a cursor, typically represented

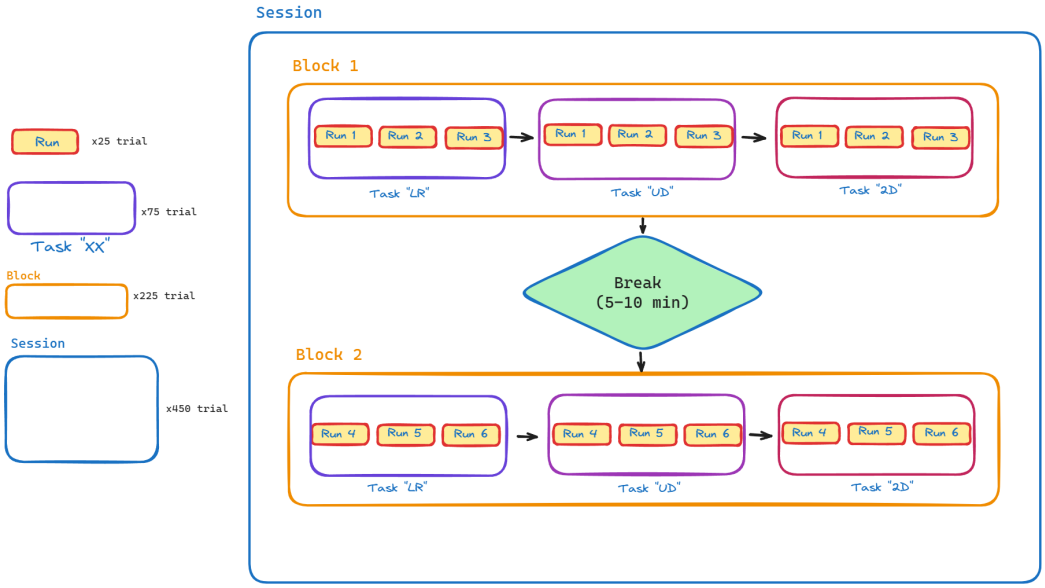


Fig. 3.4: Block design of BCI Experiment Sessions. The diagram outlines the session structure for the BCI experiment. Each session is divided into two blocks, with each block containing three runs for the tasks 'LR', 'UD', and '2D', amounting to 25 trials per run. A rest period of 5-10 minutes is provided between the two blocks, leading to a total of 450 trials per session..

as a pink ball, appears at the center of the screen, signaling the commencement of the **feedback control** period. Participants have a maximum of 6 seconds to move the cursor towards the target using the BCI2000 system (Schalk et al., 2004). This system facilitates online modulation of the , utilizing changes in the event-related desynchronization (ERD) power spectrum within the alpha frequency band, predominantly recorded over the sensorimotor areas (Stieger et al., 2021a,b). The cursor's position is updated at intervals of 40 ms, providing real-time feedback to the participant.

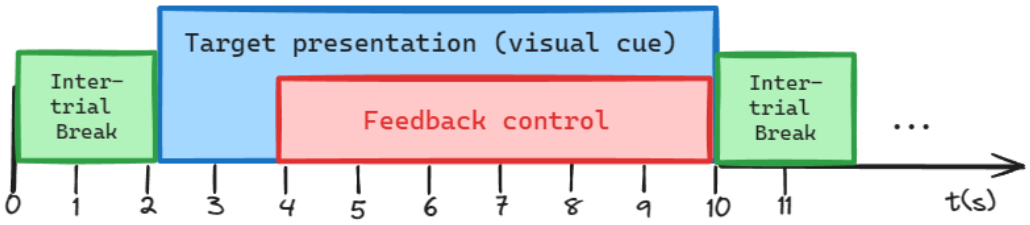


Fig. 3.5: Timeline of a Single BCI Trial..

A trial is considered successful, labeled as a "hit" (equivalent to a value of "1"), when the cursor makes contact with the correct target. Conversely, a "miss" (equivalent to a "0") is recorded if the cursor touches any other non-targeted rectangle. If the participant fails to select any target within the assigned 6 seconds, the trial is classified as a "timeout."

3.2 Data description

Data structure. The raw EEG data, available for download from the figshare repository (*Human EEG Dataset for Brain-Computer Interface and Meditation* 2021), comprises 598 files in the `.mat` format. Each file corresponds to a single session (X) for a participant (Y), labeled as *SX_Session_Y.mat* and is structured in a consistent field; the **BCI** structure contains EEG recordings as described in Table A.1. This structure encapsulates raw signals from 450 trials, including time vectors, since each trial varies in length. The trial feedback control extends up to a maximum of 6.04s during. Utilizing the BCI2000 system, cursor coordinates, both `positionx` and `positiony`, are consistently recorded, reflecting the cursor position in real-time.

Additionally, the **TrialData** structure provides a comprehensive description of trial details, as outlined in Table A.2, which includes key outcomes and labels for the training data. The fields within this structure—`Tasknumber`, `Targetnumber`, `Triallength`, `Resultind`, `Result`, `Forcedresult`, and `artifact`—serve critical roles in the preprocessing phase. For example, `Tasknumber` categorizes the type of BCI task, while `Targetnumber` specifies the target presented to participants, with identifiers such as 1 for right, 2 for left, 3 for up, and 4 for down. The `Triallength` indicates the duration of feedback control in seconds, and `Resultind` pinpoints the time index of the feedback control´s end. `Result` denotes the trial's outcome, with 1 representing a correct target hit, 0 for a miss, and NaN for timeouts. In timeout scenarios, `Forcedresult` conjectures the most probable target based on the cursor's final position. The presence of artifacts within a trial is flagged by the `artifact` field, crucial for data preprocessing.

Furthermore, the **metadata** structure, table A.3, contains demographic details (e.g. gender, MBSR participation), which facilitates data segmentation for analysis—such as assessing the impact of Mindfulness-Based Attention Training (MBAT) on BCI proficiency—by categorizing participants into two groups: those who have completed mindfulness training and the control group (Stieger et al., 2021a).

Lastly, the electrode information of the EEG cap is stored in the **chaninfo** structure as depicted in Table A.4. This includes the 62 channel names found in the `label` subfield. The electrode positions (X, Y, Z) are either stored according to the projection of the electrode using the nasion and preauricular points of the participant's head if the subfield `positionsrecorded` is set to 1, or they are generated using the standard 10-10 international system of the Neuroscan Quik-Cap, as described in Section 3.1.1. The subfield `noisechan` contains information about the noisy channels that are labeled during the BCI session, which will be utilized later in the preprocessing phase to clean the EEG data from artifacts.

3.3 Data quality

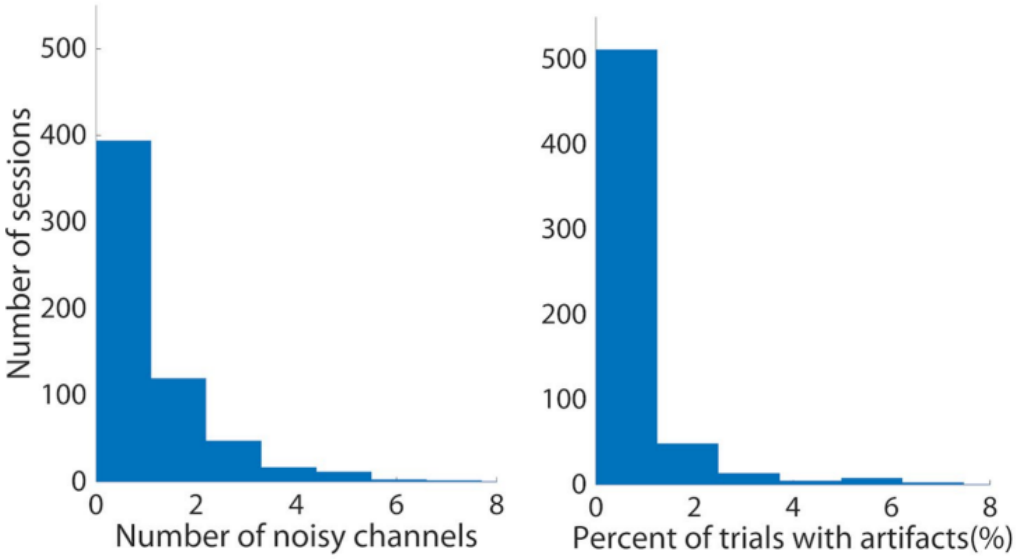
Technical validation. To ensure the data quality and accuracy of the EEG signal, and during the BCI experiment, the author of the paper applied an automatic artifact detection method, which is summarized as follows (Stieger et al., 2021b):

A bandpass filter was applied to the EEG data between 8 Hz and 30 Hz to track the sensorimotor rhythm and verify:

- If the variance (across time) of the bandpass filtered data is above a z-score threshold of 5 (z-score between electrodes), then these electrodes were labeled as artifact channels for a given session and can be found in the `BCI.chaninfo.noisechan` field. Figure 3.6a illustrates that only 3% of sessions contained more than 4 artifact channels.

- If the bandpass filtered data from any remaining electrode exceeded a threshold of ±100 µV throughout a trial, this trial was labeled as containing an artifact and can be found in `BCI.TrialData.artifact`. Figure 3.6b indicates

that only 3% of sessions had more than 5% of trials labeled as containing artifacts.



(a) shows the histogram of the percentage of trials with artifacts per session..

(b) corresponds to the histogram of the number of noisy channels per session..

Fig. 3.6: Histograms of data records containing noisy data (Stieger et al., 2021b)..

Percent Valid Correct. To measure the session performance, a metric is introduced that can quantify the success rate of the participant for each trial. It is named as the percent valid correct (PVC) metric and is written as follows in Equation 3.1 (Stieger et al., 2021b):

$$\text{PVC} = \frac{\text{hits}}{\text{hits} + \text{misses}} \tag{3.1}$$

The participants are categorized according to the PVC metric for a specific task if their median performance across all sessions crosses predefined threshold levels. To evaluate the performance of an offline SMR-BCI classification model, participants are divided into three distinct levels based on their BCI performance, and models are trained on these distinct subsets to observe how they behave with different participant performances. Group A consists of subjects with the best performance, where the median PVC score is above the upper threshold. Group B includes participants with moderate performance, where the median PVC is bounded between the upper and lower thresholds. Finally, Group C comprises

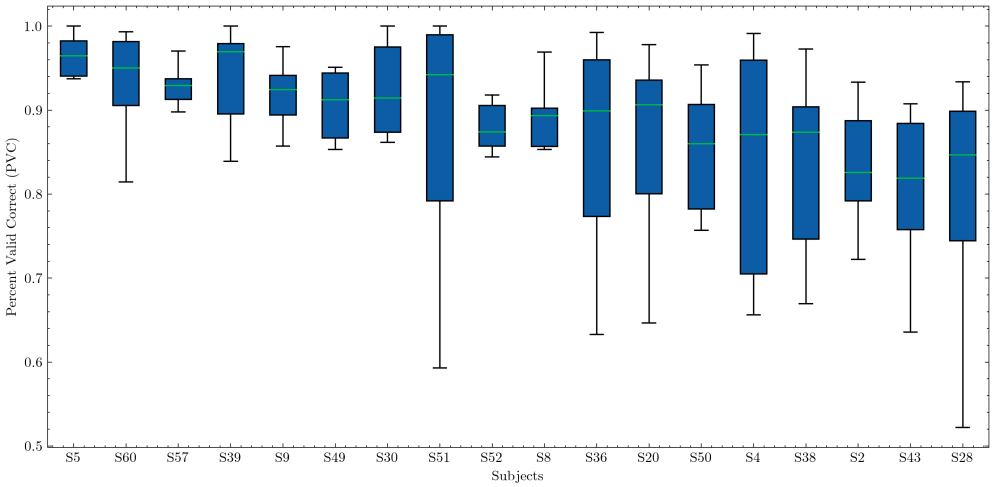
participants with poor performance, whose the median PVC falls below the lower threshold.

For the task group 1D, an 80% as the upper threshold and 70% for the lower threshold is used. This includes tasks LR and UD, and participants are divided into three groups as the graph **??** demonstrates: Group A comprises subjects that achieve more than 80% PVC, as shown in Figure 3.7a; Group B includes subjects whose performance lies between 80% and 70%, Figure 3.7b; and Group C includes those with PVC below 70%, as illustrated in Figure 3.7c.

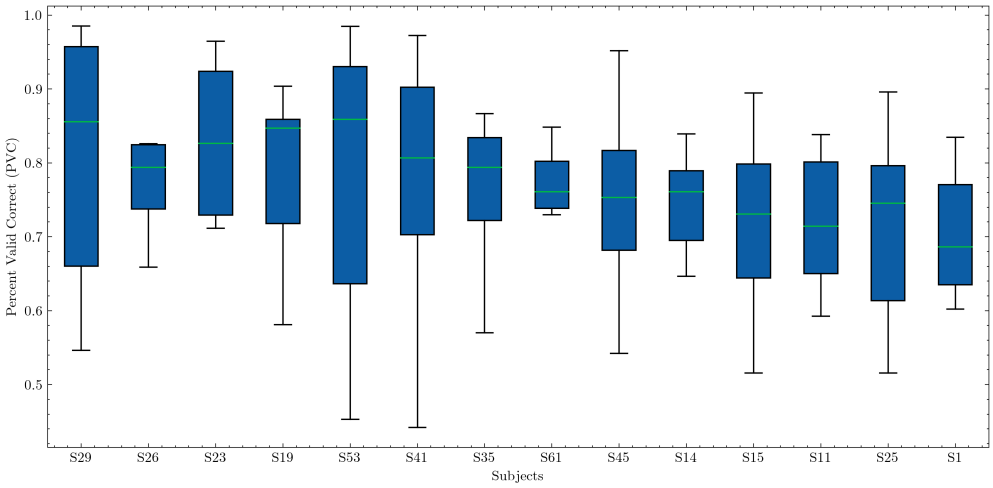
Similarly, for the task 2D, a threshold interval of 50% to 40% PVC is defined. Consequently, three groups are formed: Group A consists of participants with a PVC above 50%, shown in Figure 3.8a; Group B includes those whose performance ranges between 50% and 40%, presented in Figure 3.8b; and Group C encompasses participants with PVC below 40%, as depicted in Figure 3.8c.

The distribution of PVC scores for each participant across sessions is visualized using individual boxplots as displayed in Figure 3.7 for the left-right task and Figure 3.8 for the two-dimensional task. A single boxplot summarizes the central tendency and dispersion of the PVC scores for one participant. Specifically, the boxplot's box represents the interquartile range (IQR), which is the range between the first quartile (Q1, 25th percentile) and the third quartile (Q3, 75th percentile), encompassing the middle 50% of the data. Within each box, a horizontal line, depicted in green, indicates the median PVC metric — the point at which half the scores are above and half are below. This median line is crucial as it determines the participant's central tendency of performance, serving as a reliable indicator of their typical session efficacy. It is this median value that is used to categorize participants into different performance groups.

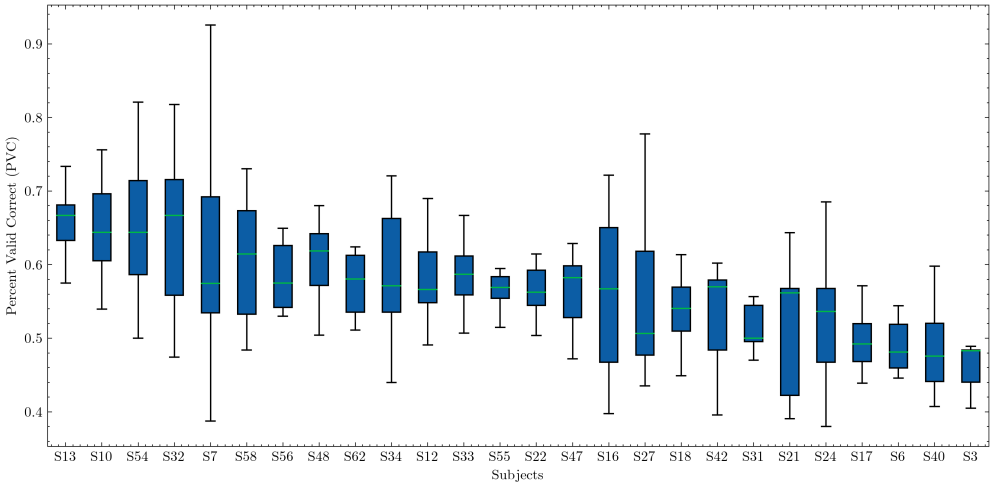
Fig. 3.7: Task Left-Right (LR) Performance Across Categories.



(a) Group C1 - PVC above 80%.

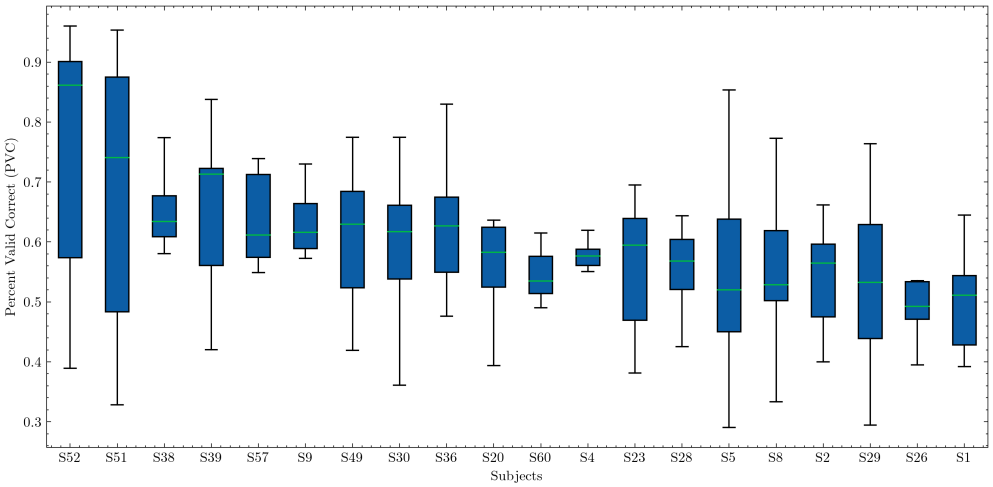


(b) Group C2 - PVC between 70% and 80%.

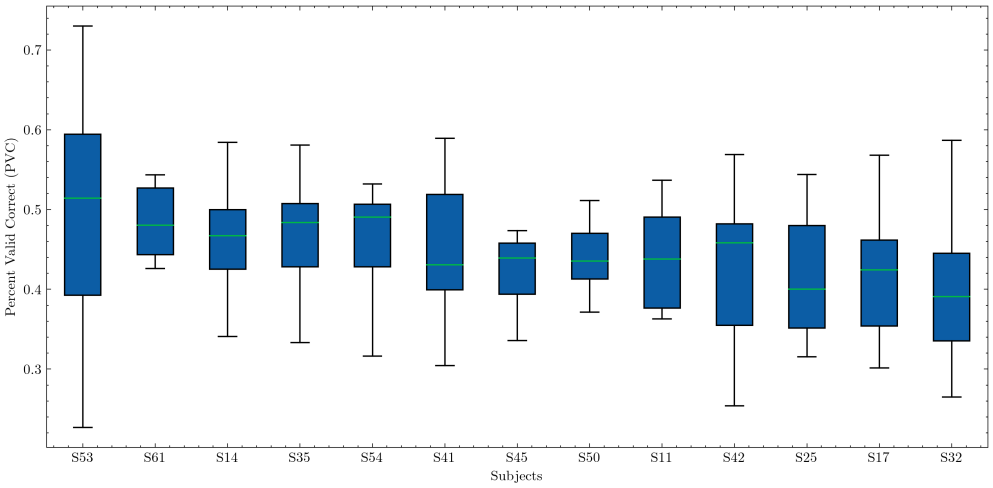


(c) Group C3 - PVC below 70%.

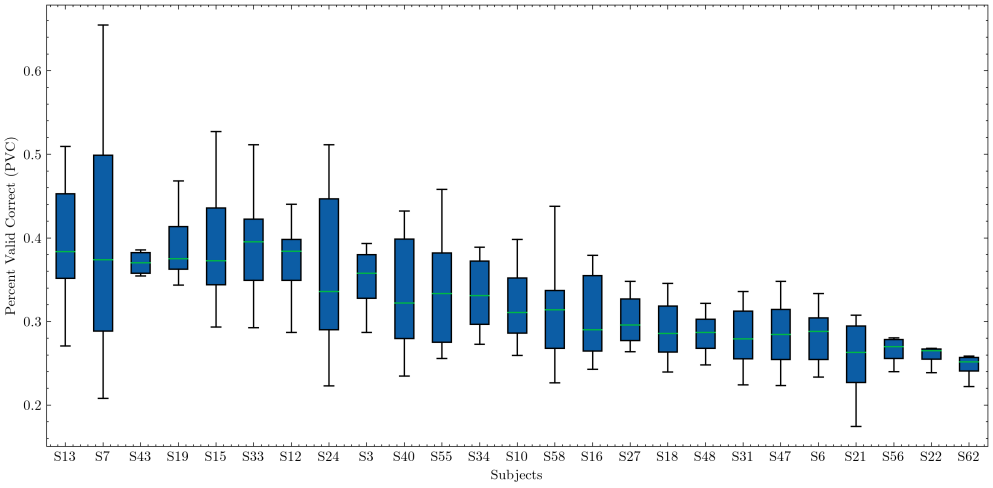
Fig. 3.8: Task two-dimensional (2D) Performance Across Categories.



(a) Group C1 - PVC above 50%.



(b) Group C2 - PVC between 50% and 40%.



(c) Group C3 - PVC below 40%.

4 Toolbox: Berlin Brain-Computer Pynterface

The development of offline Brain-Computer Interface (BCI) systems for decoding brain signals has become a critical endeavor in the field of neurotechnology. These systems necessitate a robust and reproducible machine-learning workflow, which encompasses comprehensive signal-processing methods for filtering and preprocessing, as well as advanced machine-learning algorithms for classification. The emergence of deep neural networks in BCI applications has furthered the complexity of developing new models. Coupled with the increased accessibility and affordability of BCI hardware, the volume of data available for BCI systems has significantly expanded.

Constructing an effective BCI system is a challenging task that demands expertise in both machine learning and neuroimaging disciplines. The creation of a Toolbox that aligns with the foundational machine learning workflow is essential. Such a toolbox would enable researchers and developers to train, evaluate, and interpret new models in a time-efficient and reproducible manner. Incorporating a Deep Learning component within this toolbox is particularly crucial. It allows for leveraging the capabilities of modern neural networks, which are adept at managing the high-dimensional data typical of BCI applications.

This Toolbox would make the whole process easier, from starting to work with the data to understanding the model's results. It would make building BCI systems less complicated and bring together knowledge from different areas. This would help create new ideas and make it easier to use BCI research in real-world situations to help people.

4.1 BBCPy toolbox

The **Berlin Brain-Computer Pynterface (BBCPy)** project, developed by the neurotechnology group at TU Berlin, is engineered as a versatile system for BCIs applications suitable for offline analysis and online experiments. As the successor to the **Wyrm** toolbox (Venthur, Blankertz, 2014), also from the same group, its purpose is to normalize different types of neurophysiological data (e.g., EEG, ECoG, fMRI, or NIRS) into a consistent and unified format—**NumPy** arrays. These arrays ensure compatibility with widely used libraries such as **scikit-learn**. **NumPy** library (Harris et al., 2020) is chosen for its broad mathematical capabilities, compatibility with different hardware and computing platforms, and especially because of its optimized C code that provides speed during computation. This is beneficial for handling large datasets efficiently. Moreover, **NumPy** arrays are well-known and user-friendly among Python users, which makes brain data easier and faster to work with.

Moreover, BBCPy encompasses a suite of standard processing techniques, including filtering, artifact rejection, and spatial filtering, among others. Its structure is inspired by the **scikit-learn** (Buitinck et al., 2013) pipeline, permitting the chaining of multiple processing steps in a fixed order. This setup not only streamlines the overall workflow but also improves the readability of the code. It further prevents data leakage, ensuring that each data sample is uniformly processed through all stages.

4.1.1 Architectural overview

The architecture of the BBCPy toolbox is a modular composite, illustrated in figure 4.1 as a UML diagram, which navigates the complexity of data processing and model development. It is organized into several packages, each targeting distinct stages of the data pipeline.

- **datatypes**: A package for transforming raw brain data into specific data types suitable for various applications, based on the `numpy.ndarray` class.
  - `arrays.py`: Manages data array classes.

  - `eeg.py`: Defines classes for EEG data formats.
  - `SMR_eeg.py`: Contains classes for EEG datasets tailored to sensorimotor rhythms.
  - `utils.py`: Includes utility functions for data manipulation.
- **functions**: This package includes all necessary steps for analyzing and decoding brain signals.
  - `artifactreject.py`: Module dedicated to artifact rejection.
  - `base.py`: Orchestrates the transformation steps within the pipeline.
  - `classifiers.py`: Aggregates classes for classification algorithms.
  - `helpers.py`: Provides auxiliary functions for metric computation.
  - `normalizations.py`: Features functions for data normalization.
  - `sourcespace.py`: Locates sources of brain activity.
  - `spatial.py`: Implements spatial filters like CSP and MBCSP.
  - `spectral.py`: Manages power spectral transformations.
  - `statistics.py`: Calculates statistical metrics like covariance matrices, Z-score, and variance.
  - `structural.py`: Transforms data into an epoch-based structure.
  - `temporal.py`: Applies temporal filtering to the data.
- **load**: A package to load various data formats, currently supporting `.mat` files.
  - `eeg.py`: Loads raw EEG data.
  - `SMR_eeg.py`: Specialized for loading SMR EEG data.
- **pipeline**: Defines the workflow that strings all processing steps together.
  - `core.py`: The central component of the pipeline.
  - `model_selection.py`: Manages cross-validation.

  - `pipeline.py`: Adapts the `scikit-learn.pipeline.Pipeline` for integration within the toolbox.

- **visual**: Encompasses visualization tools including head model visualization and statistical analysis plots.
  - `scalp.py`
  - `statistics.py`

Each module within the toolbox is designed with a specific role in the processing pipeline, ensuring a cohesive and streamlined workflow from the initial data input to the final model analysis.

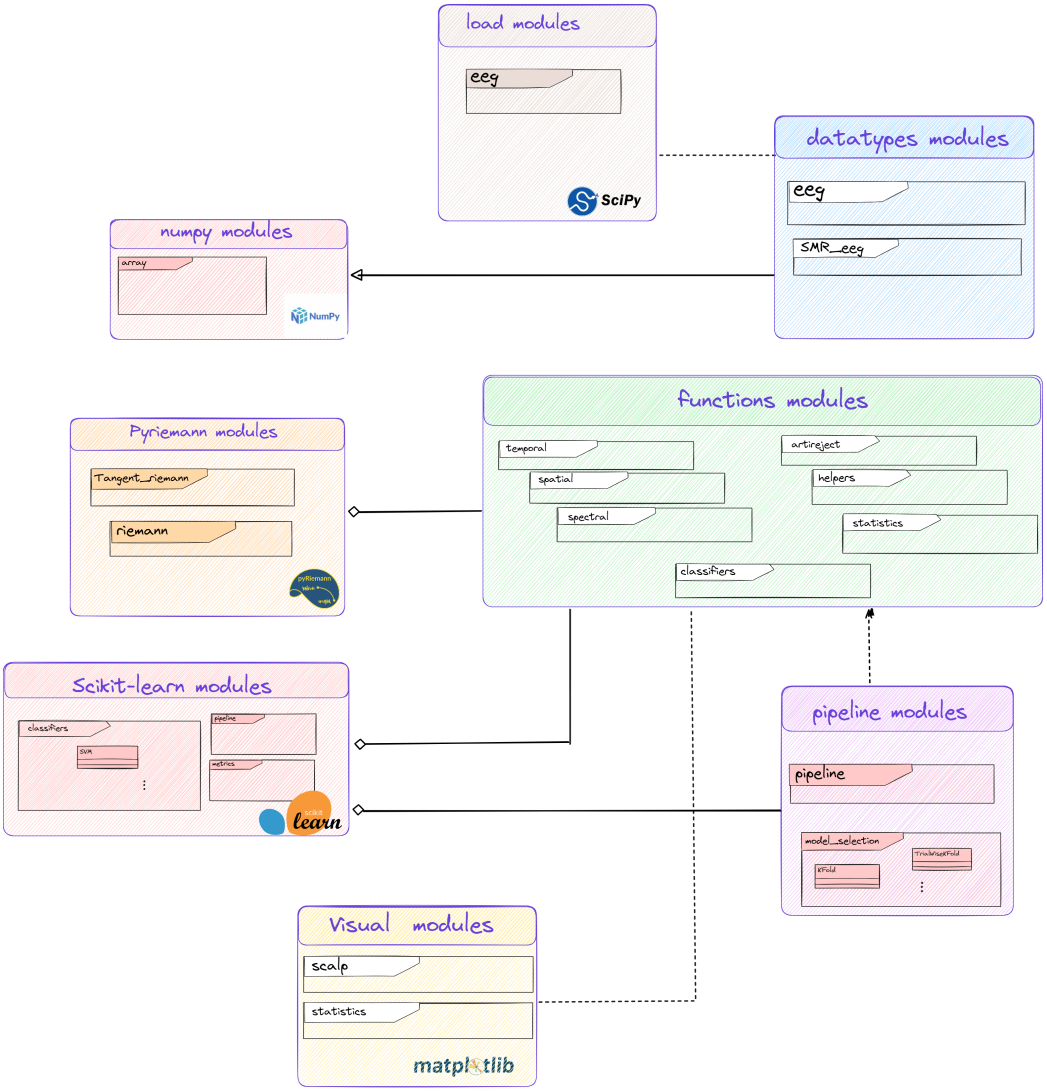


Fig. 4.1: BBCPy toolbox components.

4.2 BBCPy_DNN

The BBCPy_DNN is an integral component for the BBCPy toolbox designed to facilitate the development and exploration of Deep Neural Networks (DNNs) within the **PyTorch** framework (Paszke et al., 2019), a dynamic open-source machine learning library renowned for its deep neural network capabilities. It is distinguished by its dynamic computation graph, offering a versatile interface conducive to model training, which enhances its suitability for rapid prototyping and scalability. Its compatibility with GPU and distributed training makes it highly scalable on large clusters. The active development and adoption by the scientific community attest to its cutting-edge status in deep learning technology.

To enhance usability, particularly for newcomers, the BBCPy_DNN interface provides an intuitive interface, reducing complexity and accelerating adoption. By leveraging **PyTorch Lightning** (Falcon, The PyTorch Lightning team, 2019), it streamlines the training process, offering a balance between user-friendliness and high-level functionality. This aligns with the goal of providing researchers and engineers with the flexibility needed for efficient deep learning (DL) research without compromising performance. By integrating with PyTorch Lightning, the tool enriches the ML workflow, including experiment tracking and hyperparameter optimization.

4.2.1 Design principles and features

BBCPy_DNN supports a lifecycle that seeks to cultivate the best-performing models, delivering superior results through continuous improvement. This iterative process encompasses four stages:

- **Data Extraction and Exploration**: The first step in a machine learning project is data extraction and exploration. This phase is essential for understanding the dataset's structure and identifying the underlying patterns and trends. It involves statistical methods and data manipulation techniques such as reshaping, filtering, and sampling to prepare the data for subsequent modelling stages. This step lays the foundation for model development by

highlighting significant features and potential predictors for the desired outcomes.

- **Model Development**: Developing a model begins with defining the appropriate input shape and architecture. This stage involves feature engineering and the careful selection of model parameters. It is crucial to ensure that the data fits the model correctly, which often requires transforming raw data into a format suitable for feature space representation. A significant challenge during this phase is the curse of dimensionality, which can lead to overfitting and impact model stability. The core functionality of model development is to evaluate the model from different perspectives and to determine the best trade-off between underfitting and overfitting, leveraging hyperparameter tuning and performance metrics.

- **Tracking and Monitoring**: The final step in the ML lifecycle involves continuously monitoring the model's performance during the training and evaluation phases. This step ensures that any signs of saturation in learning or unexpected behavior are promptly identified and addressed. It is a critical process that helps in refining the model before deployment.

- **Deployment**: If the ML project is intended for a real-time application, such as an online BCI system, the deployment phase is vital. This stage focuses on integrating the model into the production environment to deliver real-time predictions. The deployment step is not merely about launching a model; it is about ensuring that the model operates reliably and efficiently in a live setting.

Each phase of the ML lifecycle is iterative and interconnected, requiring constant validation and refinement to ensure the model's accuracy and robustness. The **BBCPy_DNN** interface is crafted following an open-source, state-of-the-art ML template (Lukas, 2024), enables swift experimentation via the **Hydra** framework (Yadan, 2019).

Hydra serves as a dynamic manager for configuration files, providing a control mechanism that streamlines the ML workflow. This methodology presents numerous advantages: it bestows developers with a more adaptable and non-monotonous approach. This flexibility is evident in tasks such as optimizing the

model, adjusting hyperparameters, permitting multiple entities to concurrently train and refine various models, and managing the ML pipeline's workflow without modifying the foundational source code.

The organizational structure, depicted in Figure 4.2, illustrates the comprehensive configuration system that orchestrates the complete ML pipeline. This system encompasses all components, including data loading, preprocessing, model architecture definition, hyperparameter tuning, and the preservation of artifacts. The **Hydra** framework employs a user-friendly YAML syntax, which can be effortlessly overridden either from the main configuration or directly via the command line, thus offering augmented flexibility and efficiency. A fragment of the configuration file is presented in Figure 4.3.

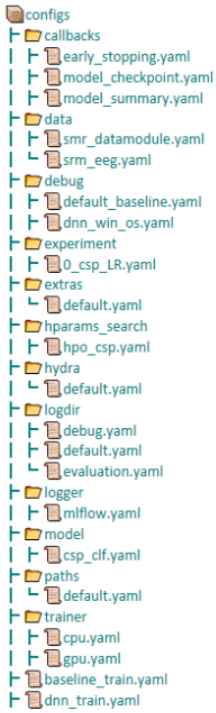


Fig. 4.2: Configuration structure..

```
# to execute this experiment run:
# python train.py experiment=example

defaults:
  - override /data: srm_eeg.yaml
  - override /model: csp_clf.yaml
  - override /trainer: sklearn.yaml
  - override /logger: mltflow.yaml
  - override /paths: default.yaml
# all parameters below will be merged with parameters from default
    configurations set above
# this allows you to overwrite only specified parameters
tags: ["baseline", "alpha_band", "Within-Subject", "csp", "LR"]

seed: 12345

data:
  task_name: "LR"
  trial_type: "forced"
  ival: 2s:10s:2ms
  chans: [ "*" ]
  bands: [ 8, 20 ]
```

Fig. 4.3: Configuration parameters for the experiment..

4.2.2 Architectural overview

The **BBCPy_DNN** interface, structured following the ML lifecycle, consists of four principal components showcased in the UML diagram in Figure 4.4. These components, orchestrated by configuration files, allow for minimal modifications to the core code, thus streamlining and simplifying the toolbox's operation. Crucially, they enable the initiation of multiple concurrent experiments while maintaining an accurate log of parameter records.

The interface's essential components are summarized in the subsequent steps:

Data Preparation. This step is devised to prime and preprocess raw brain signals optimally, contingent on the machine learning algorithm selected. Initially, raw data ingestion is facilitated by the *load module*, followed by data format standardization (either EEG or SRM_EEG) utilizing *datatype modules* from the foundational **BBCPy** toolbox. Subsequently, data transformations like filtering or artifact removal are performed with a suite of methods in the function modules. **PyTorch** contributes two data primitives: `torch.utils.data.DataLoader` and

`torch.utils.data.Dataset`, which are utilities for data loading and batch creation essential for model training. These classes, redefined in the *dataloader module*, enable efficient and parallel data loading, thus expediting the process, particularly for expansive datasets.

Model Training. This phase encompasses model design, hyperparameter setting, metric definition, and the execution of training loops. The **BBCPy_DNN** interface adopts the **BBCPy** toolbox's pipeline design, incorporating metric tracking to monitor model performance. It supports a wide array of cutting-edge models from the **PyTorch** library and provides all necessary components and methods for creating new architectures. This includes selecting appropriate loss functions and optimization algorithms. To enhance model performance, tracking, logging, and history points are integrated with suitable metrics. Model design and implementation are carried out within the *models modules*.

Hyperparameter Tuning. Optimizing models, particularly complex ones requiring deep networks, is a challenging and iterative task. As the complexity and hyperparameter count rise, so does the impact on learning outcomes and training velocity. Traditional manual hyperparameter searches are exhaustive and resource-intensive. Modern open-source libraries like **Optuna** (Akiba et al., 2019) automate this process using sophisticated optimization algorithms. **Optuna**'s core design principles include a dynamic define-by-run API, efficient sampling and pruning strategies, and a straightforward setup conducive to both lightweight and large-scale distributed computing environments.

Model Monitoring. A key feature of the **BBCPy_DNN** interface is the continuous monitoring of the model throughout training and hyperparameter optimization. Utilizing the **MLflow-Dashboard**, the framework allows for real-time tracking of model performance, providing an intuitive visualization platform. During hyperparameter tuning, the **Optuna-Dashboard** is employed to observe model performance across hyperparameter configurations. This dashboard furnishes interactive, live-updating visualizations of optimization trials, enabling detailed analysis of hyperparameter impact on model efficacy and informing the strategic selection of the optimal hyperparameter set.

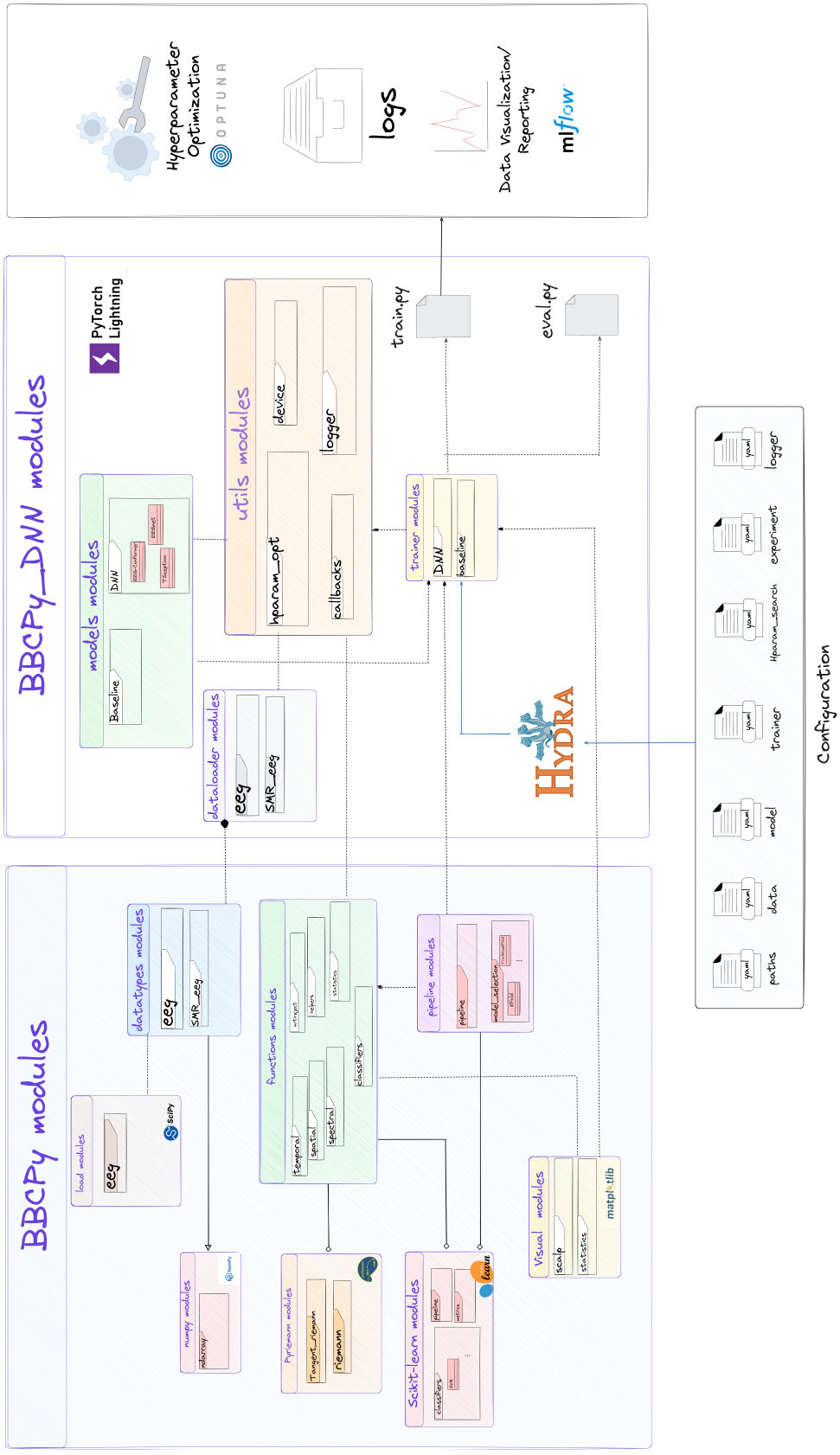


Fig. 4.4: BBCPy_DNN design.

5 Methodology

This chapter outlines the methodology employed for training both baseline and deep learning models using the dataset introduced earlier. The aim is to construct an offline MI-BCI classifier. The chapter begins by detailing the preprocessing stage, which is crucial for ensuring the quality and reliability of the data fed into the models. Following this, we delve into the cross-validation strategy, specially tailored to accommodate the block design of the experiment. We then provide a detailed description of the baseline and deep learning models, highlighting their architectural differences and specificities. Finally, the chapter concludes with an in-depth discussion of the training and testing procedures.

5.1 Pre-processing Stage

Before model training, data typically undergo a pre-processing phase to ensure quality for reliable results. This phase aims to process bad trials, which often suffer from artifacts or noisy channels by applying processing techniques and includes the following steps:

- The utilized dataset (Stieger et al., 2021b) compiles all trials during a session, labeling them based on participant performance. Labels include *'1'* for a correct target hit, *'0'* for a miss, and *'NaN'* for timeouts. The dataset also provides an additional label in the `forcedtrials` subfield to indicate whether the participant completely failed to hit the target, which in this case is labeled **'0'**, or if they were likely near the target, then it is considered as successful and marked as *'1'*. This approach aims to increase the dataset size and also to introduce a level of uncertainty.

- To ensure data quality, which includes identifying noisy channels as imperative, the dataset has already tagged noisy channels for each trial in the `chaninfo` structure. To maintain data shape consistency and avoid discarding data, we apply *Laplacian referencing* to noisy channels.

- Epoching involves selecting the trial length. In this thesis, we exclude inter-trial break segments and include all intervals of target presentation and control feedback. The duration of control feedback varies, with a maximum of 6 seconds because the participant could finish early. The longest trial length is thus considered for consistency. Furthermore, due to high-resolution data and a 1000 Hz sampling rate, downsampling to 500 Hz is necessary to avoid computation overload without information loss because typically the important brain signals are within 1 Hz to 100 Hz.

- For the data's spectral content, a bandpass filter is applied to remove noises and select the Sensorimotor rhythm region, typically between 8-13 Hz for α waves and 13-30 Hz for α+β waves (see Section 2.2.2). Specifically, a 5th order Butterworth band-pass filter with frequencies $f_{p1} = 8$ and $f_{p2} = 20$ Hz is applied to the EEG data, offering a flat frequency response within the passband and attenuation towards zero in stop-bands.

- Normalization or standardization of the EEG data before model training is crucial due to EEG's non-stationary behavior and high variability among participants. This process helps in mitigating these variations, ensuring consistent and comparable results across subjects and sessions. It also enhances training speed and stability, particularly for neural network-based models, where the learning algorithm converges much faster. To facilitate fair comparisons between different algorithms, standardizing pre-processing steps is essential.

5.2 Cross-validation

An important aspect of building a robust and reliable offline classifier for BCI systems lies in the well-studied selection of cross-validation procedures, which must consider the block design of the BCI experiment. Due to the autocorrelation

of brain activity, there is a lack of independence between single trials, potentially violating the requirement of independence between training and test sets typically assumed in standard cross-validation. This situation can lead to overfitting results because of information leakage, ultimately producing an ineffective classifier. Additionally, cross-validation serves as a strategy to address data imbalance, a common challenge in MI-BCI systems for multiclass classification. It's common for certain classes to be underrepresented in a session, influenced by a participant's performance and preferences. Implementing cross-validation helps ensure an equitable distribution of classes across the folds in both training and validation sets. This equitable distribution is crucial for the development of a balanced and effective classification model.

In this thesis, we have carefully designed a cross-validation approach that takes into account the block design of the BCI experiment, as described in section 3.1.2. Typically, a participant performs three different tasks in one session, divided into two major blocks with a 5-10 minute break in between.

Each block requires the participant to complete three runs of each task, with each run comprising 25 trials. Consequently, the cross-validation is structured into 5 folds, each corresponding to a single run. In each iteration of this process, one fold is designated as the validation set, while the remaining parts constitute the training set. This procedure is repeated five times, ensuring that each part is used as the validation set exactly once. The remaining run is reserved for a final evaluation of the models, serving as the ultimate test set. The detailed design of this cross-validation block is illustrated in figure 5.1.

5.3 Baseline approaches

This section introduces three "classical" methods that will serve as baselines for benchmarking the dataset in use. The first method, Common Spatial Patterns (CSP), has been a standard in the field of Motor Imagery (MI) EEG classification for several years and is crucial for assessing the dataset's difficulty. The second method involves a Riemannian covariance approach based on multiscale spectral features, which offers a significant performance enhancement over the CSP

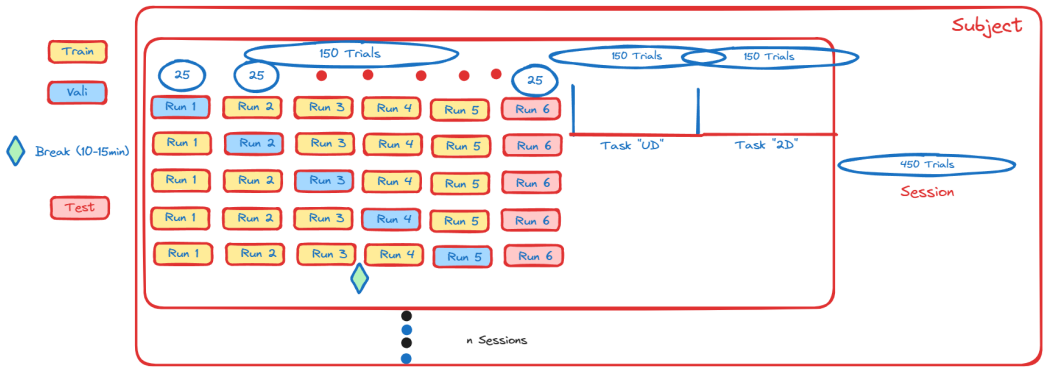


Fig. 5.1: Strategy A: Cross-Validation Configuration according to the design block..

method. Finally, we present an advanced version of the Riemannian Covariance method, which incorporates the Tangent space transformation, known as the Tangent Riemann method.

5.3.1 Common Spatial Pattern

The Common Spatial Pattern (CSP) is the most commonly used algorithm in BCI applications and it belongs to the spatial filter family. It extracts a series of spatial filters by decomposing the multidimensional data into a set of uncorrelated single patterns. For an offline BCI classifier, the CSP is used as a supervised decomposition technique that is optimized to create the most discriminant features between two mental states that can be easily differentiated using a simpler classifier (see Section 2.3.2).

The CSP classifier consists mainly of 4 blocks, as depicted in Figure 5.2. After data preprocessing, the EEG signal has the shape (#trials, #channels, #timepoints).

Stage 1: Band-Pass Filtering - The goal of this stage is to emphasize the motor rhythm frequency band, where the salient information is located. Therefore, a bandpass with a range from 8 to 20 Hz, which is a compromise between the alpha and beta bands, is used. Then the data is selected and concatenated class-wise.

Each segment is a trial and denoted for the left trial as $\mathbf{X}_k^L$ and for the right trial as $\mathbf{X}_k^R$, with both trial vectors presented below:

$$\mathbf{X}_L = [\mathbf{X}_1^L, \ldots, \mathbf{X}_k^L]; \mathbf{X}_R = [\mathbf{X}_1^R, \ldots, \mathbf{X}_k^R] \tag{5.1}$$

Stage 2: Spatial Filtering - The goal of this stage is to compute the filter matrix W, by first calculating the covariance matrices of both classes (Left and Right), as follows:

$$\Sigma_L = \frac{1}{T_L - 1}\mathbf{X}_L\mathbf{X}_L^T, \quad \Sigma_R = \frac{1}{T_R - 1}\mathbf{X}_R\mathbf{X}_R^T \tag{5.2}$$

According to the optimization equation 2.5 in Section 2.3.2, the filter matrix W is obtained that holds eigenvectors for both Σ_L and Σ_R.

Stage 3: Feature Selection - The objective of this stage is to identify the most discriminative CSP filters that satisfy two key conditions: the variance $\text{var}(w_i^T\mathbf{X}_1)$ should be maximized, implying that d_i is large; conversely, $\text{var}(w_i^T\mathbf{X}_2)$ should be minimized, indicating that $1 - d_i$ is small, where d_i represents the eigenvalues. The critical criterion is to select the eigenvector w_i from matrix W that corresponds to the larger eigenvalue d_i in relation to Σ_1. Subsequent to the selection process, the CSP features for each trial are normalized and transformed into a log-variance space, as detailed in Equation 2.8.

Stage 4: Classification - Finally, for the classification stage, there is a variety of algorithms to choose from. In our case, we apply the support vector machine classifier, introduced in section 2.4, on the CSP feature space.

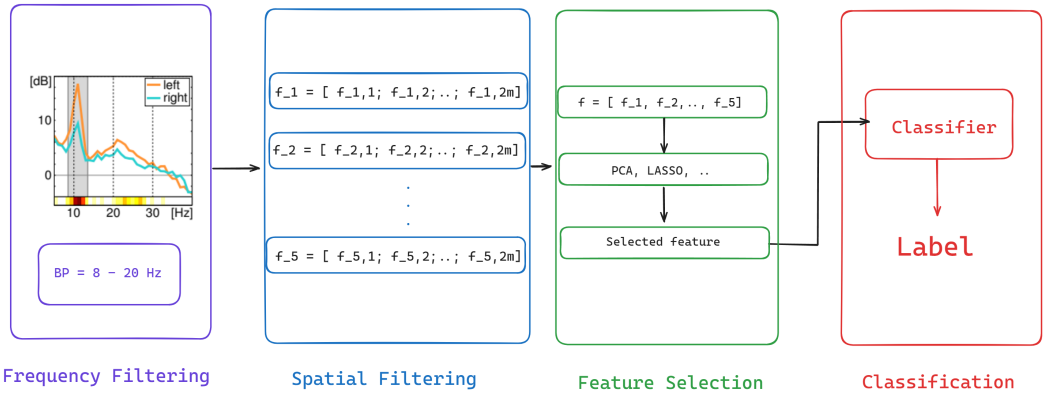


Fig. 5.2: CSP pipeline.

5.3.2 Riemannian covariance

The second baseline approach applies the Riemannian geometry introduced in Section 2.3.3. This approach uses covariance matrix estimation and obviates the need for a feature selection stage. The advantage of Riemannian geometry is that the manipulation of covariance matrices is more natural in the Riemannian space because of the properties of geodesic distance. The geodesic distance is the natural metric in the space of covariance matrices, making calculations like averaging covariance matrices and computing their relative distance more intuitive.

The Riemannian approach consists of three stages, as described in the figure 5.3:

Stage 1: Band-Pass Filtering, the same stage that undergoes the CSP approach 5.3.1, where a bandpass filter is applied in the range of 8-20 Hz.

Stage 2: Feature Selection, it starts with computing the covariance matrices for each trial vector $\mathbf{X}_L$ and $\mathbf{X}_R$ for Left and Right class (see section 2.3.1). Then a Riemannian transformation is applied using the logarithmic mapping function 2.14 of the computed SPD covariance matrix. Finally, the Riemannian mean or also known as the Fréchet mean is used to calculate the distance between trials 2.12).

Stage 3: Classification; SVM classifier with a Riemannian-based kernel **riemann_kernel** is applied to the feature space in the Riemannian manifold.

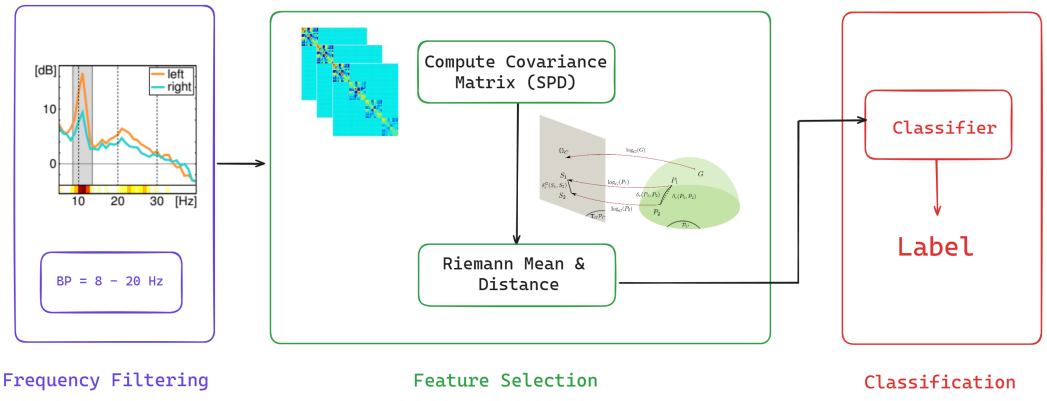


Fig. 5.3: Riemann piepline.

5.3.3 Tangent Riemann covariance

Another variant that improves the previous Riemannian approach is to project the covariance matrices of the EEG signals into a local space that has the Euclidean properties, which can easily apply operations such as averaging and calculating distances.

The diagram in Figure 5.4 demonstrates the variation of the tangent local tangent space at some point Ω using Equation 5.3

$$S_i = \mathbf{C}^{-\frac{1}{2}} \ln(\mathbf{C}^{-\frac{1}{2}} \mathbf{C}_i \mathbf{C}^{-\frac{1}{2}}) \mathbf{C}^{\frac{1}{2}} \tag{5.3}$$

Then S_i can be vectorized to turn it into an orthonormal Euclidean $\mathbb{R}^n$ space. This transformation also preserves the distances between the computed center of mass C (identity on the tangent space) and each covariance matrix C_i. Finally, a linear classifier is applied, in this case, a standard SVM.

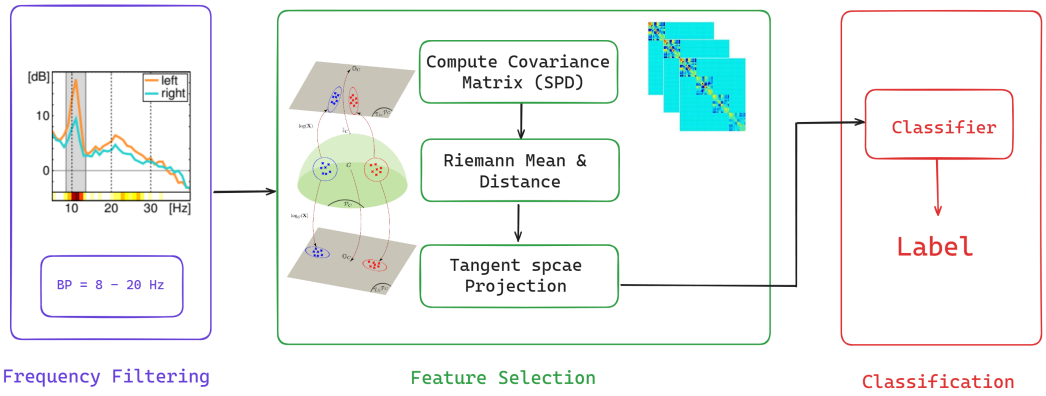


Fig. 5.4: Tangent Riemann pipeline.

5.4 Deep Learning approaches

Deep learning (DL) has recently garnered considerable attention for machine learning tasks due to its capability to extract features directly from raw input data with minimal preprocessing and cleaning. The success of DL is attributed

to improvements in computational power and the increasing availability of data, which plays a crucial role in the enhancement and learning of DL algorithms. For BCI systems, DL offers an opportunity to simplify processing pipelines by enabling automatic end-to-end learning of preprocessing, feature extraction, and classification blocks.

This section introduces a set of deep learning models that have been applied to the continuous sensorimotor dataset for offline classification. The main goal is to evaluate the performance of these models for inter-subject classification on the largest motor imagery (MI) dataset available to date, which can later be utilized as benchmark dataset to devolve and design new deep learning models for both online and offline BCI applications.

5.4.1 EEGNet

EEGNet (Lawhern et al., 2018) is a deep neural network model that primarily utilizes multiple convolutional networks (see Section 2.5.2) with different variations to create a general-purpose architecture for EEG-based classification tasks. The key components of EEGNet are the usage of depthwise separable convolutions, which combine a depthwise convolution—applying a single filter to each input channel separately, creating a feature map from each channel alone—and a pointwise convolution, which combines the outputs of the depthwise convolution. This allows information exchange between channels by applying a 1×1 filter.

The advantage of this approach is the significant reduction in the number of trainable parameters compared to conventional CNNs, and the ability to learn features from spatial and temporal space. The EEGNet architecture, depicted in Figure 5.5 and summarized in Table 5.1, consists of the following blocks:

- **Block 1:** Formed using two types of convolutions – temporal and spatial. Initially, the input with shape $C \times T$ (where C is the number of channels and T is the number of time points) is fed to a Conv2D layer with filter size F_1, followed by batch normalization. Subsequently, a DepthwiseConv2D is utilized to learn spatial filters, then batch normalization is applied again. A spatial convolution is applied using the DepthwiseConv2D to learn spatial

filters, drastically reducing the number of trainable parameters, especially when applying AveragePooling2D. This block, with the help of temporal and spatial filters, aims to extract spatial features at a set of frequencies with the help of the depthwise operation. Additionally, Dropout, a regularization technique, is applied to avoid overfitting.

- **Block 2:** This mainly consists of separable convolutions, which include a depthwise convolution followed by a pointwise convolution layer with parameter F_2. This also reduces the number of trainable parameters and aims to combine the feature maps extracted from Block 1 into the best output combination. Batch normalization is applied along with Average Pooling and Dropout. The final feature maps are then flattened using a Flatten operation.
- **Block 3:** The classification layer applies a softmax classification layer with N units, corresponding to the number of classes.

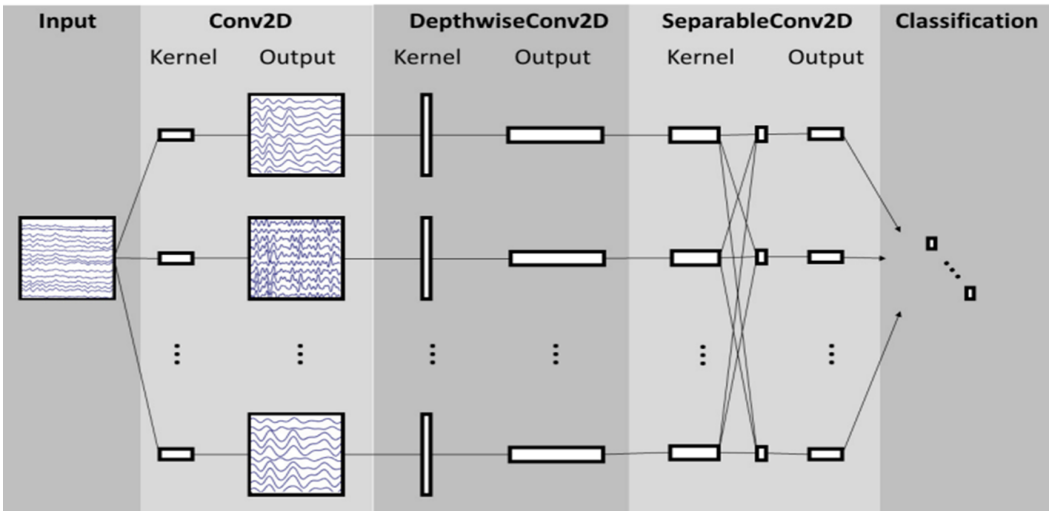


Fig. 5.5: EEGNet structure (Lawhern et al., 2018).

Module	Layer	Filters	Size	Output	Activation
1	Input	—	—	$C \times T$	—
	Reshape	—	—	$1 \times C \times T$	—
	Conv2D	F_1	$(1, 64)$	$F_1 \times C \times T$	Linear
	BatchNorm	—	—	$F_1 \times C \times T$	—
	DepthwiseConv2D	$D \times F_1$	$(C, 1)$	$(D \times F_1) \times 1 \times T$	Linear
	BatchNorm	—	—	$(D \times F_1) \times 1 \times T$	—
	Activation	—	—	$(D \times F_1) \times 1 \times T$	ELU
	AveragePool2D	—	$(1, 4)$	$(D \times F_1) \times 1 \times \frac{T}{4}$	—
	Dropout	—	$p = 0.5$	$(D \times F_1) \times 1 \times \frac{T}{4}$	—
2	SeparableConv2D	F_2	$(1, 16)$	$F_2 \times 1 \times \frac{T}{4}$	Linear
	BatchNorm	—	—	$F_2 \times 1 \times \frac{T}{4}$	—
	Activation	—	—	$F_2 \times 1 \times \frac{T}{4}$	ELU
	AveragePool2D	—	$(1, 8)$	$F_2 \times 1 \times \frac{T}{32}$	—
	Dropout	—	$p = 0.5$	$F_2 \times 1 \times \frac{T}{32}$	—
	Flatten	—	—	$F_2 \times \frac{T}{32}$	—
3	Classifier	—	—	N	Softmax

Tab. 5.1: Architecture of the EEGNet..

5.4.2 TSception

The TSception network is a combination of temporal and spatial convolution designed to capture the temporal dynamics and spatial asymmetry in EEG data. It was originally applied for emotion recognition, as studies have shown that emotional information correlates with brain activity across multiple frequency bands (Ding et al., 2020). In this thesis, we adapt the TSception architecture for MI EEG data. Prior to training, it is crucial to transform the EEG data into a set of channel configurations, reducing the number of channels to 52. This transformation is essential for the later stages, hence it is performed before the training phase.

The TSception consists of three main blocks, as depicted in Figure 5.6:

- **Temporal Learner:** The input data, with the shape $(\text{channels} \times 1 \times \text{time points})$, is fed into the temporal learner. Since the TSception aims to scan multiple frequency bands, the ratio coefficients $\alpha_k \in \mathbb{R}$, where k is the level of the

temporal learner, define the number of frequency levels K that the temporal learner can scan. For example, if $K = 3$ and $\alpha = 0.5$, then for $k = 1$ to 3, the ratio coefficients would be $[0.5, 0.25, 0.125]$, allowing the capture of frequencies at 4 Hz and above, and 8 Hz and above. After applying K operations with respective T kernels, batch normalization is applied.

- **Spatial Learner:** This block has multi-scale 1D convolutional kernels and applies three different kernels to each EEG channel. These spatial filters correspond to the channel arrangement configuration and are named the global kernel, hemisphere kernel, and local kernel, as shown in the electrode map in Figure 5.7, which divides the 32 electrodes into three groups corresponding to the kernel categories.

  Finally, The obtained spatial features are then flattened.

- **Classifier:** a classification layer that applies a softmax function with N units corresponding to the number of classes T.

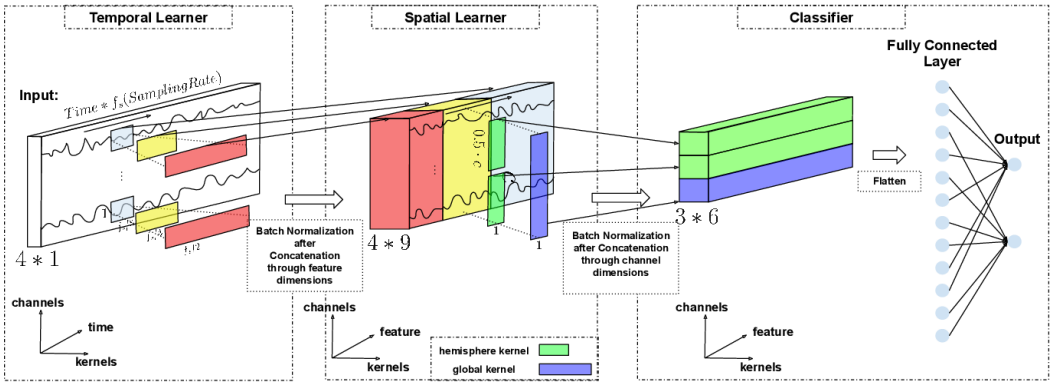


Fig. 5.6: TSception structure (Ding et al., 2020).

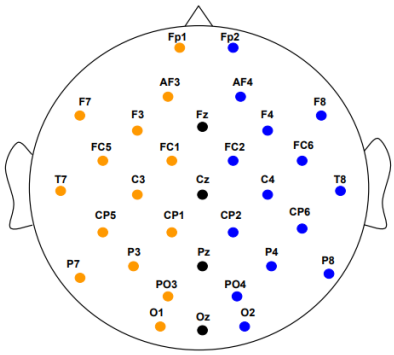


Fig. 5.7: The location map of 32 channels cap (Ding et al., 2020).

5.5 Training Procedure

This section defines the training schemes adopted for the baseline and deep learning approaches. This study focuses on inter-subject classification, where the classifier is trained on trial data from the corresponding subject and does not have access to data from other subjects. The dataset used (Stieger et al., 2021b) encompasses a total of four classes: imagining moving the left hand (Left), imagining moving the right hand (Right), imagining moving both hands (Up), and a relaxed state (Down). We aim to train two distinct classifiers to differentiate between MI states in two stages. The first stage is constructing a binary classifier that differentiates between the Left and Right classes, denoted as task LR. The second stage involves all classes to build a multiclass classifier with four classes, denoted as task 2D.

Furthermore, we adopt a cross-validation technique that aligns with the design block of the BCI system. This technique is applied for hyperparameter search and validation, as discussed in Section 5.2. It is also important to note that the distribution of trials for each class varies because we only considered successful trials. Additionally, we included "forced" trials—those labeled as timeouts but classified during the BCI online classification during data acquisition based on the location of the cursor on the screen, which increases the amount of data available.

5.5.1 Baseline Models

The training procedure for baseline models in this study is methodically designed to ensure optimal model performance. This process is composed of two main steps:

- 1. Pilot Run: Initially, each model is configured with a standard setup and trained using data from three sessions per subject. This preliminary stage, referred to as the pilot run, is crucial for assessing the model's initial behavior and determining the necessity for extensive hyperparameter optimization. If the model demonstrates promising performance during this phase, training is recommenced with the inclusion of remaining sessions. This approach adheres to the cross-validation strategy from *Strategy A* 5.1, encompassing five folds and iterating across all subject sessions.

- 2. Hyperparameter Optimization (HPO): In cases where the model's performance in the pilot run is either random or inferior compared to benchmark results (Gwon et al., 2023), as per the selected metrics, a hyperparameter optimization is conducted. Baseline models typically require tuning of a limited number of parameters, often no more than three. Therefore, grid search algorithm (*Grid Search - an overview | ScienceDirect Topics* 2024) is employed for identifying the optimal combination of parameters that maximize performance metrics for the given task, whether binary or multiclass.

An important aspect of all baseline models is the application of a Support Vector Machine (SVM) classifier in the classification stage. However, a challenge arises due to the slight variation in the number of trials for all classes for each subject, resulting in an unbalanced dataset. Standard SVM models tend to underperform with skewed class distributions. To address this, SVM variants have been developed that include a penalty parameter C, which is adjusted during training to favor the smaller class. This variant effectively calculates the class distribution at each iteration and updates the C coefficient accordingly. The scikit-learn SVM implementation offers an option to overcome the unbalanced data problem, enabling the adjustment of the classifier to better handle such disparities in class distribution.

5.5.2 Deep Learning Models

The training of deep learning models in this study involves a distinct three-step process:

1. Pilot Run: The initial phase involves training the neural networks on a small sample of trials. This step is primarily focused on evaluating the impact of various model hyperparameters, which differ from one architecture to another. During this phase, models are trained on a random selection of subjects for a limited number of epochs while varying model parameters. Throughout the training process, the Adam optimization algorithm, a member of the stochastic gradient descent family, is utilized for its efficiency and quick convergence toward the global minimum. Nevertheless, certain parameters, such as the learning rate and momentum, still require fine-tuning (Kingma, Ba, 2017).

2. Hyperparameter Optimization (HPO): Unlike the baseline models, deep learning approaches demand substantial computational resources and a longer time, even for a single run. To manage the increasing number of hyperparameters without overburdening the computational cluster, a novel approach is adopted. This approach deviates from the traditional cross-session training by tuning the models on the subject with the highest average performance across all sessions, as measured by the percent valid correct (PVC) metric 3.3. This metric assesses the success rate of participants in each trial. The goal is to ensure that deep neural networks can effectively capture salient motor imagery features from EEG data of a well-performing subject. This strategy aims to reduce computational load and time, establishing a starting point for longer training. For both tasks, the subject with the highest PVC metric across all sessions is selected for intensive hyperparameter search using the ***T**ree-structured **P**arzen **E**stimator* (TPE) algorithm (Watanabe, 2023) within the Optuna framework. TPE is a Bayesian optimization method that accelerates hyperparameter search by modeling the hyperparameter space distribution, creating two subsets of distributions - one "bad" and the other "good." During evaluation, TPE favors parameters likely under the "good" distribution.

3. Long Run: In the final stage, after establishing the set of parameters, models are trained separately for each subject over extended epochs (100 epochs). The

training process is assisted by MLflow, a monitoring tool that visualizes model performance using well-defined metrics. Artifacts and model checkpoints are regularly saved for subsequent analysis. Additionally, as a regularization technique, an early stopping mechanism is employed. This technique is commonly used to prevent model overfitting and getting stuck in local minima by setting a monitoring validation metric as a threshold condition and fixing the maximum number of iterations.

5.6 Testing Procedure

The final phase in evaluating the performance of the models involves testing them on a set of data that remains unseen during the training process. This approach, similar to the one used for the validation set, ensures that the test data is completely new to the models, thereby guaranteeing a fair and unbiased comparison of their performance.

The test set comprises trials from the last runs of each session. This selection strategy results in the test set containing approximately 16% of the total trials for each subject. It is crucial to maintain consistency in data handling across different stages of model development. Therefore, the pre-processing steps used for the test set mirror those applied during the training phase.

6 Results and Discussion

The chapter will illustrate the results obtained by performing two types of classifications. The first type, denoted as Task LR, is a binary classification that aims to decode left and right-hand movements from the EEG data used. The second type, denoted as Task 2D, considers all mental states, including left, right, both hand movements, and rest, resulting in four distinct classes to be classified. Throughout the study, we focus exclusively on Within-Subject Classification, where models are only trained on individual subject trials. This is also considered cross-session training, where the participant performs the BCI task over 8 weeks across 7 to 11 sessions.

Additionally, subjects are categorically divided according to their PVC metric 3.3, which indicates the participant's proficiency. We have divided the subjects into three categories: C1, C2, and C3, based on the PVC metric intervals. For more details on this categorization, see Section 3.3.

6.1 Task LR

In this section, we will illustrate the results of both baseline and deep learning models for Task LR, which corresponds to binary classification. Consequently, binary accuracy is utilized to compare performances. All models receive EEG data with the shape (#trials, channels, timepoints), where the number of channels is consistently 62, except for the TSception model, which includes only 52 channels and 8000 data points corresponding to the feedback interval of the BCI experiment. All models undergo preprocessing steps, from artifact rejection to bandpass filtering in the interval [8 Hz, 20 Hz] (including alpha and a portion of the beta bands).

Before discussing the models results, it is important to note the subject categorization for the LR task, which is defined as follows:

- Category C1: Greater than 80% accuracy, denoted as best subject performance.
- Category C2: Between 80% and 70% accuracy, denoted as average performance.
- Category C3: Lower than 70% accuracy, denoted as BCI literacy.

6.1.1 Baseline Models

CSP: For the CSP model, which performed poorly in the pilot training, the important parameter is the number of discriminative spatial filters for each class, denoted as "n_components". Therefore, we conducted a hyperparameter search to find the optimal number of spatial filters that deliver the best performance. A grid search was applied using cross-validation from strategy A on a set of filter numbers $n\_components = \{2, 4, 6, 8, 10, 12, 14, 16\}$.

Applying this approach to all subjects, we determined that the best number of filters is 4 for each class, implying a total of 8 spatial filters for the LR task.

Riemann Models: There are no particular hyperparameters to tune for the Riemann and tangent variant models, and they are used in their standard form. Both models employ the PyRiemann framework (Barachant et al., 2023) for the Riemannian geometry algorithms.

Using the unseen data split, the test set, the test accuracies of the baseline models – CSP, Riemann, and Tangent Riemann models – are presented in Figure 6.1. Each model's performance is depicted as a violin plot combined with a scatter plot, where the violin plot illustrates the distribution of accuracies, and individual subject data points are overlaid as colored dots. The CSP model shows a wide spread of accuracies, with some subjects achieving high performance. The Riemann model demonstrates a more concentrated distribution, indicating consistent performance across subjects. Conversely, the Tangent Riemann model exhibits a broad distribution with a lower median accuracy.

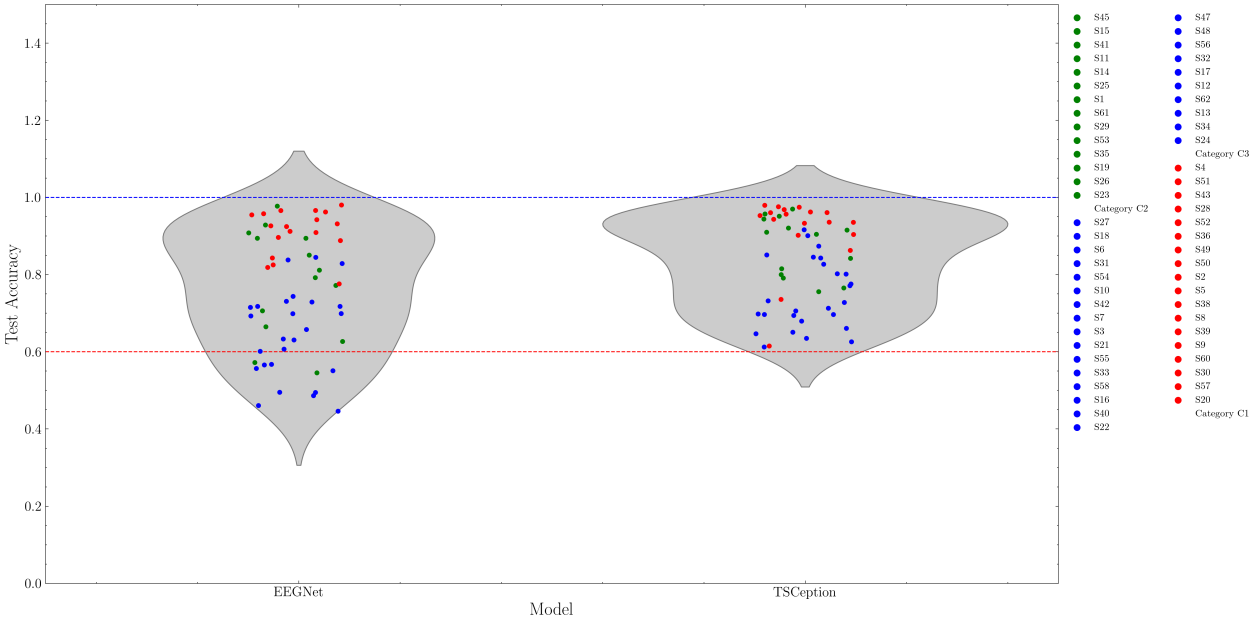


Fig. 6.1: Distribution of Test Accuracies of Baseline Models for the LR Task.

The variability of test accuracies suggests differences in model robustness and subject-specific factors that affect BCI performance. This is observed by the high scores of most C1 category subjects and the poor outcomes of CSP models and Tangent for the C3 category subjects. Overall, CSP delivers the poorest performance in comparison to the Riemann family models.

6.1.2 Deep Learning Models

The table **??** presents the best parameters used to train all subjects using the EEGNet model. According to the model structure outlined in Table **??**, these parameters are then utilized to train models for other subjects.

Similarly, the same approach is applied for the TSception model, where the hyperparameter optimization is conducted based on the set of hyperparameters listed in Table **??**. This approach ensures that each model is optimally configured for its specific task and subject dataset.

Tab. 6.1: Hyperparameter Configurations for EEGNet and TSception Models (LR task).

Tab. 6.2: EEGNet Model.

Hyperparameter	Value
Dropout rate	0.5
First kernel size (kernel_1)	512
Second kernel size (kernel_2)	256
Number of filters (F_1)	6
Number of filters (F_2)	16
Number of spatial filters (D)	8

Tab. 6.3: TSception Model.

Hyperparameter	Value
Sampling Rate	500
Number of Temporal Filters (num_T)	15
Number of Spatial Filters (num_S)	20
Hidden Channels	32
Dropout Rate	0.5

Figure 6.2 presents the distribution of test accuracies for the EEGNet and TSception models on the LR task. The distribution suggests that the TSception model tends to yield higher median test accuracies compared to EEGNet, as reflected by the narrower distribution towards the upper end of the accuracy range. However, both models exhibit a broad spectrum of performances, with some subjects falling below the designated threshold of 0.6, highlighted by the red dotted line. This threshold is considered to be the minimum benchmark accuracy for offline BCI classifiers. Consistently, we observe the same pattern of lower accuracy for the Category C3 group, which indicates the struggles of some participants with the BCI task, making it challenging to build a decoding model.

6.2 Task 2D

Task 2D presents a multiclass problem, which is inherently more complex than the binary one. The models must differentiate between four classes, and due to the imbalance in class distribution, we introduce a weighted coefficient to both approaches. This coefficient is usually addressed within the classification block of each model.

Similarly to Task LR, subjects are categorized into three groups for this task, but with different intervals:

- Category C1: Greater than 50% accuracy, denoted as best subject performance.

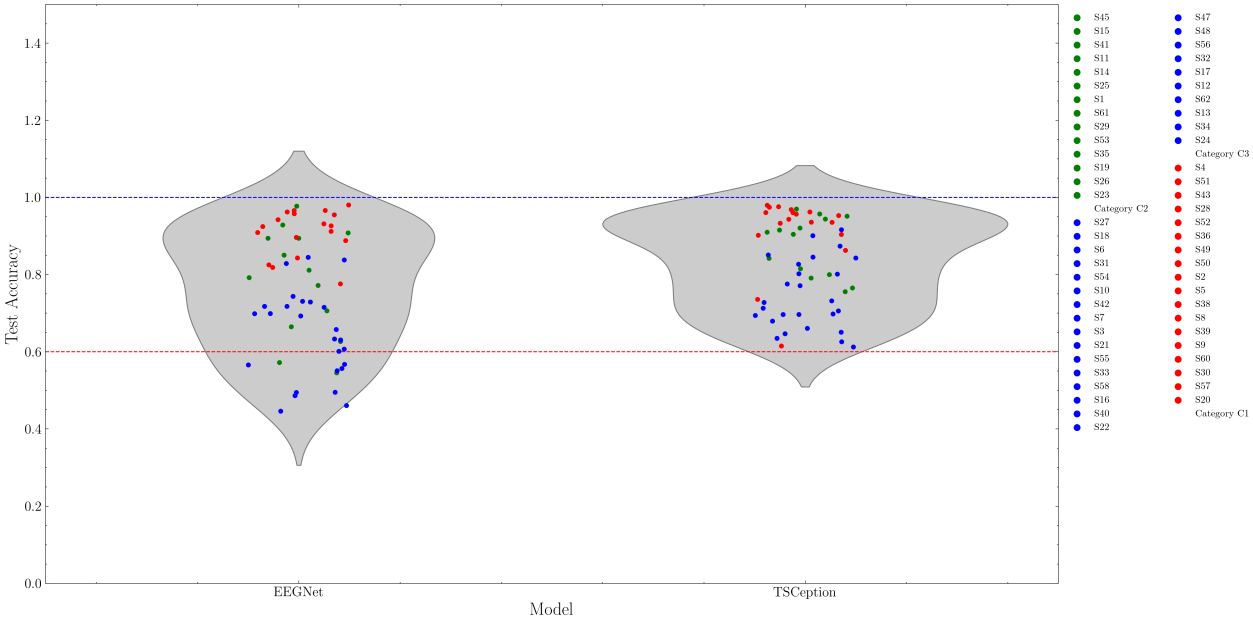


Fig. 6.2: Distribution of Test Accuracies of DL Models for the LR Task.

- Category C2: Between 50% and 40% accuracy, denoted as average performance.
- Category C3: Lower than 40% accuracy, denoted as BCI literacy.

6.2.1 Baseline Models

Traditionally, CSP addresses binary classification. However, variations exist, such as the approaches that apply joint approximate diagonalization (JAD), as discussed in Section 2.3.2. CSP also includes hyperparameter tuning for the optimal number of spatial filters "n_components", which, for this task, remains 4 for each class. For the Riemannian models, there are no significant changes other than requiring more time than the previous task. In all models, we introduce a weighting factor to the classification block, in this case, the SVM, to prevent the model from favoring the larger class.

Figure 6.3 illustrates the distribution of test accuracies for the baseline models for 2D task. The chance level for a correct random guess in this task is 0.25, as

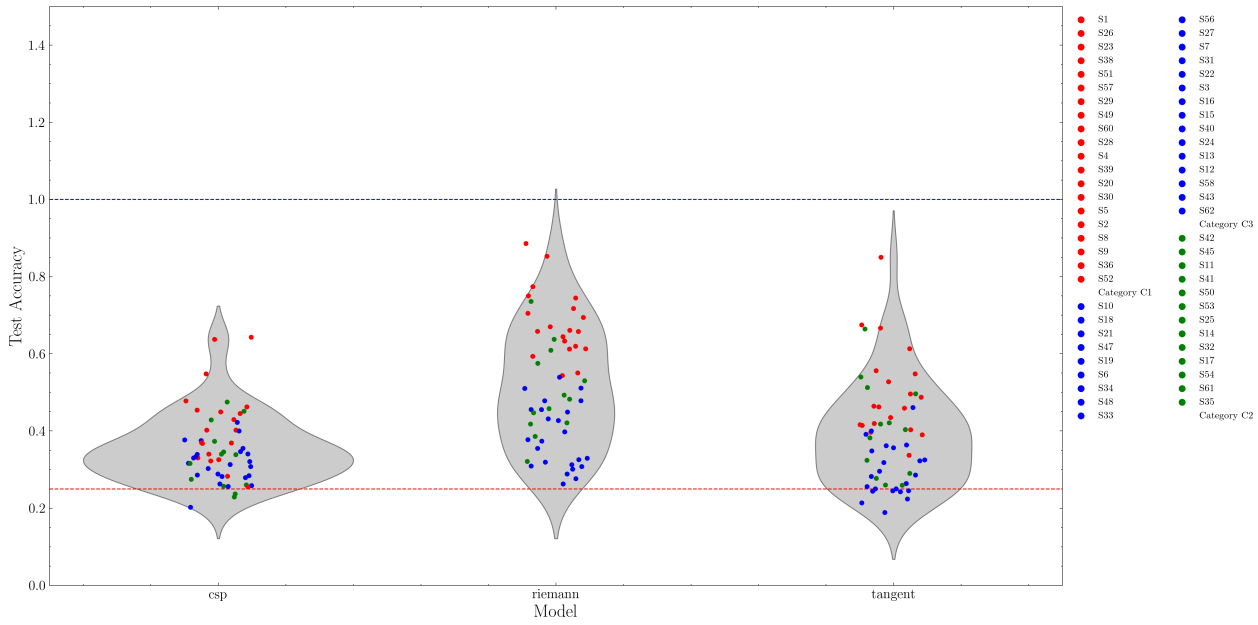


Fig. 6.3: Distribution of Test Accuracies of Baseline Models for the 2D Task.

indicated by the red dotted line. It is evident from the plot that while a significant number of subjects performed around or below the chance level in the CSP model, the Riemann and Tangent Riemann models generally yielded higher median test accuracies, suggesting a more effective decoding of the 2D task by these models. However, variability in subject performance is still present, as indicated by the spread of the data points across the accuracy spectrum.

6.2.2 Deep Learning Models

The overall setup for the deep learning models is similar to the binary classification, except for the loss function. In multiclass classification, we typically choose a Categorical Cross-Entropy Loss (**deeplearning_book**). For EEGNet, the procedure involves an initial hyperparameter search followed by extended training. The best hyperparameters are illustrated in Table 6.5. Similarly, the best parameters for TSception are presented in Table 6.6.

Figure 6.4 illustrates the test accuracy distribution for the EEGNet and TSception models on the Task 2D. The EEGNet model displays a broader distribution,

Tab. 6.4: Hyperparameter Configurations for EEGNet and TSception Models (2D task).

Tab. 6.5: EEGNet Model.

Hyperparameter	Value
Dropout rate	0.5
First kernel size (kernel_1)	512
Second kernel size (kernel_2)	256
Number of filters (F_1)	8
Number of filters (F_2)	32
Number of spatial filters (D)	2

Tab. 6.6: TSception Model.

Hyperparameter	Value
Sampling Rate	1000
Number of Temporal Filters (num_T)	10
Number of Spatial Filters (num_S)	15
Hidden Channels	128
Dropout Rate	0.5

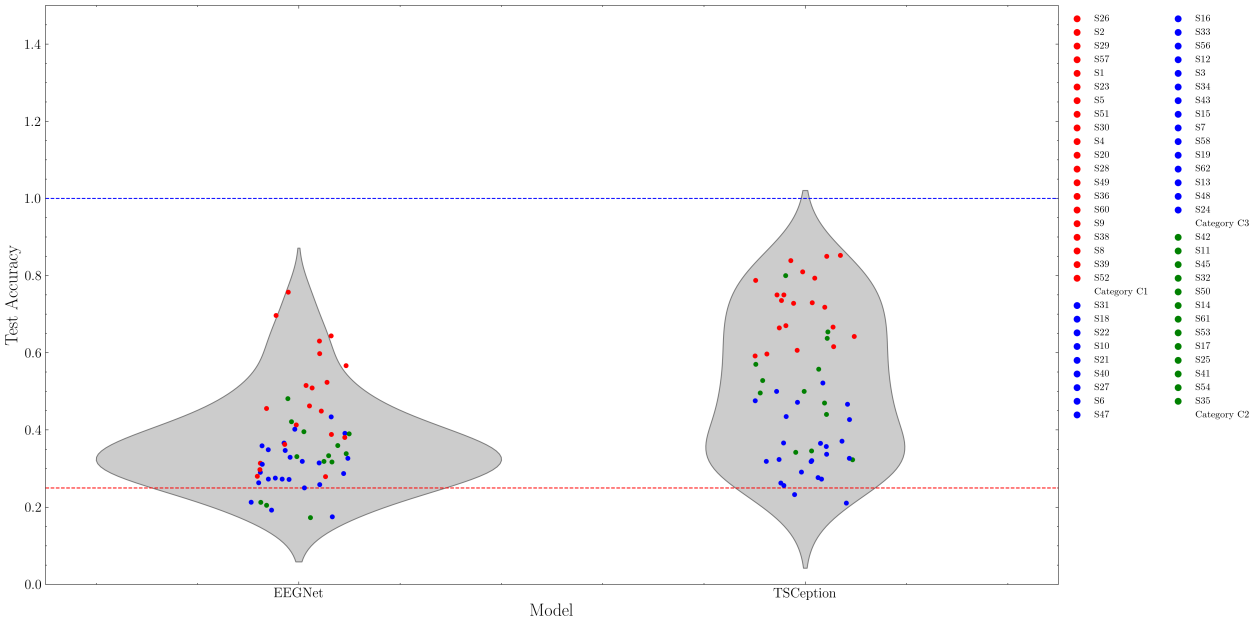


Fig. 6.4: Distribution of Test Accuracies of DL Models for the 2D Task.

indicating a wider variance in subject accuracy, while the TSception model's distribution suggests a slightly higher central tendency. This indicates that TSception's ability to scan multiple frequency bands in the initial temporal block 5.4.2 has proven effective in extracting salient features in both the temporal and spatial domains. However, this does not rule out the possibility of a suboptimal choice of hyperparameters for EEGNet, and a cross-subject approach involving more than one subject could potentially improve the model's performance.

Notably, some subjects' accuracies fall below the chance level, and these are primarily from the group with BCI literacy, which confirms the impact of BCI illiteracy on BCI classifier performance.

6.3 Models comparison

Figure 6.5 showcases the performance of all models by subject category for both test (left) and validation (right) accuracies in the LR task. The subject categories are distinguished by color, with C1, C2, and C3 each assigned a specific hue as indicated in the legend.

A consistent trend is observed across both the test and validation sets, with no significant changes in model performance between the two. This suggests that the models' ability to generalize from training to unseen data is stable. Notably, the performance is influenced by BCI inefficiency (Acqualagna et al., 2016), as demonstrated by the poor performance of the classifier for the C3 category with subpar BCI performance.

Importantly, the Riemann model almost equally performs to the TSception model across both datasets. Despite the advanced architecture of TSception, the Riemann model, which is a baseline method, shows equal or superior performance. This indicates that for the specific dynamics of the LR task in BCI, the simpler Riemann model is as effective as the more complex TSception model.

Finally, figure 6.6 illustrates the performance of various models on the more complex 2D task. The task's complexity is reflected in the lower overall accuracy levels when compared to the LR task. It is evident that deep neural network models like EEGNet and TSception have a superior performance over the baseline models, which is attributable to their enhanced capacity for managing the complexity of multiclass problem.

However, the EEGNet model exhibits an underperformance compared to TSception. This disparity underscores the challenges even sophisticated models face with complex MI-BCI tasks. Despite the advanced architectures, the deep learning models do not completely overcome the challenges posed by BCI illiteracy, which

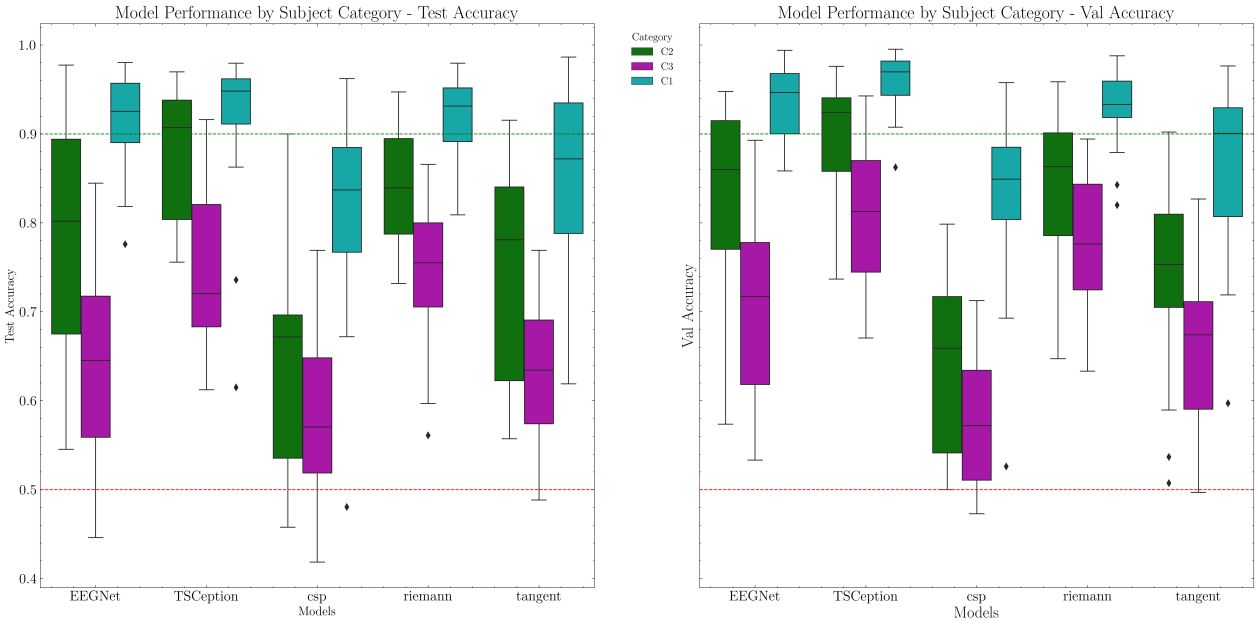


Fig. 6.5: Models performances by subject category for the LR task.

affects both the baseline and neural network. This phenomenon is characterized by significant variability in BCI performance task across subjects, highlighting the need for further research into model robustness and inter-subject training .

Tab. 6.7: Comparison of Model Performance on LR and 2D Tasks.

	LR Task		2D Task	
Model	val/acc	test/acc	val/acc	test/acc
EEGNet	0.80 ±0.14	0.76 ±0.16	0.43 ±0.11	0.37 ±0.13
TSCeption	**0.88** ±0.09	**0.83** ±0.12	**0.60** ±0.14	**0.52** ±0.19
csp	0.67 ±0.14	0.67 ±0.15	0.35 ±0.09	0.35 ±0.09
riemann	0.84 ±0.10	0.82 ±0.10	0.54 ±0.15	0.52 ±0.16
tangent	0.74 ±0.14	0.73 ±0.13	0.39 ±0.12	0.39 ±0.14

To conclude, we draw attention to the significant findings presented in Table **??** One notable observation is that the Riemannian models outperform the CSP models in both the LR and 2D tasks, which is unexpected given that CSP is generally considered the superior model among classical methods. Furthermore, the TSception model surpasses all other models, including the benchmark model

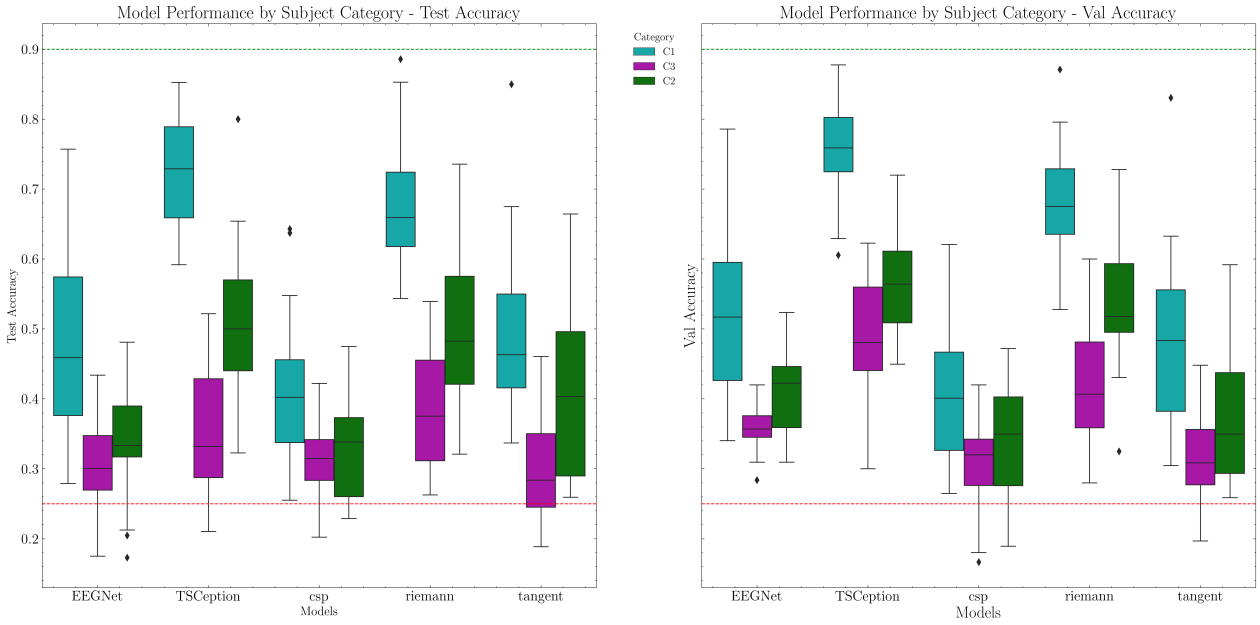


Fig. 6.6: Models performances by subject category for the 2D task.

EEGNet, in both binary and multiclass tasks. This superior performance of TSception may be attributed to its unique architecture, which effectively combines spatial and temporal features. The results suggest that with additional training and fine-tuning, there is potential to further improve the performance of the models, particularly TSception. This could be due to its ability to capture the complex underlying patterns within the EEG data, which are essential for accurate classification in BCI tasks.

7 Conclusion and Outlook

This study has demonstrated that the dataset under investigation is indeed suitable for deep learning model training in MI-BCI applications. The dataset's expansive size allows for substantial model training, while its structure facilitates cross-session studies. Additionally, the dataset's inclusion of various proficiency levels among subjects has enabled a thorough examination of BCI inefficiency, a crucial aspect often overlooked in similar research. Baseline models were also trained to establish benchmarks, and the results were convincingly within acceptable ranges.

Significant strides were made into the BBCPy framework by building an interface that integrates deep learning approaches that obey the Machine Lifecycle and offers scalability for distributed runs on a cluster. This toolbox is engineered to expedite the development process for researchers and engineers, particularly those without an extensive software background. The integration aims to accommodate all MI-BCI datasets as standardized datatypes, with smooth interoperability with other toolboxes such as MNE (Zubarev et al., 2022) and MOABB (Jayaram, Barachant, 2018). This toolbox provides user-friendly methods to develop and test new methods for constructing BCI classifiers, which are pertinent for both online and offline applications.

The deep learning approaches generally outperformed the baseline models, particularly in the multiclass classification tasks. The dataset proved to be an excellent choice for developing and prototyping new network architectures. It also served as a fertile ground for further exploration into inter-subject classification and transfer learning across subjects.

The implications of these findings are important. The investigated dataset is highly recommended for further MI-BCI research, especially in deep learning.

Additionally, the BBCPy toolbox is not only an open platform for enhancement and contributions from the neuroscience technology community but also for collaboration and further integration of new algorithms and datatypes.

However, the limitations of this research should be acknowledged. The study focused solely on the classification problem using an inter-subject approach and revealed gaps in the training of deep learning models, particularly in the depth and methods required for effective hyperparameter optimization. Also, the BCI inefficiency observed among the subjects leads to heterogeneous behaviors that can affect the models' fair evaluation. Future work should aim to address these limitations and explore the full potential of methods that consider a cross-subject approach.

Appendix A

Data Structure

Tab. A.1: BCI structure. (Stieger et al., 2021b).

Subfield Name	Primary Data	Secondary Data	High-Level Description
data	1 x 450 cell	nChannels x nTime matrix	EEG data from each trial of the session
time	1 x 450 cell	1 x nTime vector	vector of the trial time (in ms) relative to target presentation
positionx	1 x 450 cell	1 x nTime vector	X position of cursor during feedback
positiony	1 x 450 cell	1 x nTime vector	Y position of cursor during feedback
SRATE	1 x 1 scalar	N/A	Sampling rate of EEG recording

Tab. A.2: TrialData structure. (Stieger et al., 2021b).

Subfield Name	High-Level Description	Values
tasknumber	Identification number for task type	1 = 'LR' 2 = 'UD' 3 = '2D'
runnumber	The run to which a trial belongs	1–18
trialnumber	The trial number of a given session	1–450
targetnumber	Identification number for target presented	1 = right 2 = left 3 = up 4 = down
triallength	The length of the feedback control period in s	0.04s–6.04s
targethitnumber	Identification number for target selected by BCI control	1 = right 2 = left 3 = up 4 = down NaN = no target selected; timeout
resultind	Time index for the end of the feedback control portion of the trial	Length(trial) – 1000
result	Outcome of trial: success or failure?	1 = correct target selected 0 = incorrect target selected NaN = no target selected; timeout
forcedresult	Outcome of trial with forced target selection for timeout trials: success or failure?	1 = correct target selected or cursor closest to correct target 0 = incorrect target selected or cursor closest to incorrect target
artifact	Does the trial contain an artifact?	1 = trial contains an artifact identified by technical validation 0 = trial does not contain an artifact identified by technical validation

Tab. A.3: metadata structure. (Stieger et al., 2021b).

Subfield Name	**High-Level Description**	**Values**
MBSRsubject	Did the participant attend the MBSR intervention?	1 = yes 0 = no
meditationpractice	Hours of at-home meditation practiced outside of the MBSR intervention	Hours = MBSR group NaN = control group
handedness	The handedness of the participant	'L' = left handed 'R' = right handed NaN = data not available
instrument	Did the participant play a musical instrument?	'Y' = yes 'N' = no 'U' = used to NaN = data not available
athlete	Did the participant consider themselves to be an athlete?	'Y' = yes 'N' = no 'U' = used to NaN = data not available
handsport	Did the participant play a hand based sport?	'Y' = yes 'N' = no 'U' = used to NaN = data not available
hobby	Did the participant have a hobby that required fine motor movements of the hands?	'Y' = yes 'N' = no 'U' = used to NaN = data not available
gender	Gender of the participant	'M' = male 'F' = female NaN = data not available
age	Age of the participant in years	18–63
date	Date of BCI training session	yeardaymonthday
day	Day of the week of the BCI training session	1–7 starting with 1 = Monday
time	Hour of the start of the BCI training session	7–19

Tab. A.4: chaninfo structure. (Stieger et al., 2021b).

Subfield Name	Data type	High-level Description
positionsrecorded	bool	1 = subject specific electrode positions were recorded 0 = subject specific electrode positions were not recorded
labels	1 x 62 cell	Electrode names of the EEG channels that form rows of the trial data matrix
noisechan	nNoisyChannels x 1 vector	A vector of the channels labeled as noisy from automatic artifact detection. Empty if no channels were identified as noisy.
electrodes	nChannels x 4 struct (Columns: Label, X, Y, Z)	3D positions of electrodes recorded individually for each session. If this data is not available, generic positions from the 10–10 system will be included.
fiducials	3 x 4 struct (Columns: Label, X, Y, Z)	If positionsrecorded = 1 provides the location of the nasion and left/right preauricular points, otherwise empty.
shape	nPoints x 3 struct (Columns: X, Y, Z)	If positionsrecorded = 1 provides the location of the face shape information, otherwise empty.

List of References

Acqualagna, L., L. Botrel, C. Vidaurre, A. Kübler, B. Blankertz (2016): Large-Scale Assessment of a Fully Automatic Co-Adaptive Motor Imagery-Based Brain Computer Interface. en. *PLOS ONE* 11 (2). Ed. by D. Yao, e0148886. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0148886. URL: https://dx.plos.org/10.1371/journal.pone.0148886 (last access 12/05/2023) (cit. on p. 78).

Aguiar, P., A. David, S. Paulo, A. Rosa (2000): EEGSolver - Brain Activity and Genetic Algorithms., 80–84. DOI: 10.1145/335603.335702 (cit. on p. 9).

Akiba, T., S. Sano, T. Yanase, T. Ohta, M. Koyama (2019): *Optuna: A Next-generation Hyperparameter Optimization Framework*. arXiv:1907.10902 [cs, stat]. DOI: 10.48550/arXiv.1907.10902. URL: http://arxiv.org/abs/1907.10902 (last access 01/26/2024) (cit. on p. 52).

Barachant, A., Q. Barthélemy, J.-R. King, A. Gramfort, S. Chevallier, P. L. C. Rodrigues, E. Olivetti, V. Goncharenko, G. W. v. Berg, G. Reguig, A. Lebeurrier, E. Bjäreholt, M. S. Yamamoto, P. Clisson, M.-C. Corsi (2023): *pyRiemann/pyRiemann: v0.5*. DOI: 10.5281/zenodo.8059038. URL: https://doi.org/10.5281/zenodo.8059038 (cit. on p. 72).

Beltrachini, L., N. von Ellenrieder, C. H. Muravchik (2010): Shrinkage approach for EEG covariance matrix estimation. eng. *Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual International Conference* 2010, 1654–1657. ISSN: 2375-7477. DOI: 10.1109/IEMBS.2010.5626668 (cit. on p. 13).

Berger, H. (1929): Über das Elektrenkephalogramm des Menschen. de. *Archiv für Psychiatrie und Nervenkrankheiten* 87 (1), 527–570. ISSN: 1433-8491. DOI: 10.1007/BF01797193. URL: https://doi.org/10.1007/BF01797193 (last access 12/26/2023) (cit. on p. 5).

Bishop, C. (2006): *Pattern Recognition and Machine Learning*. Springer. ISBN: 978-0-387-31073-2 (cit. on pp. 19, 20).

Blankertz, B., M. Tangermann, F. Popescu, M. Krauledat, S. Fazli, M. Danóczy, G. Curio, K.-R. Müller (2008): The Berlin Brain-Computer Interface. Vol. 5050, 79–101. ISBN: 978-3-540-68858-7. DOI: 10.1007/978-3-540-68860-0_4 (cit. on pp. 14, 16).

Buitinck, L., G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. Vanderplas, A. Joly, B. Holt, G. Varoquaux (2013): *API design for machine learning software: experiences from the scikit-learn project*. arXiv:1309.0238 [cs]. DOI: 10.48550/arXiv.1309.0238. URL: http://arxiv.org/abs/1309.0238 (last access 01/25/2024) (cit. on p. 44).

Compumedics Neuroscan (2024): *SynAmps RT 64-channel Amplifier*. https://compumedicsneuroscan.com/product/synamps-rt-64-channel-eeg-erp-ep-amplifier/. Accessed: 2024-01-13 (cit. on p. 33).

Congedo, M., B. Afsari, A. Barachant, M. Moakher (2014): Approximate joint diagonalization and geometric mean of symmetric positive definite matrices. eng. *PloS One* 10 (4), e0121423. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0121423 (cit. on pp. 17, 18).

CS231n Convolutional Neural Networks for Visual Recognition (2024). URL: https://cs231n.github.io/convolutional-networks/ (last access 01/29/2024) (cit. on p. 26).

Ding, Y., N. Robinson, Q. Zeng, D. Chen, A. A. Phyo Wai, T.-S. Lee, C. Guan (2020): TSception:A Deep Learning Framework for Emotion Detection Using EEG. *2020 International Joint Conference on Neural Networks (IJCNN)*, 1–7. DOI: 10.1109/IJCNN48605.2020.9206750 (cit. on pp. 64–66).

Falcon, W., The PyTorch Lightning team (2019): *PyTorch Lightning*. Version Number: 1.4. DOI: 10.5281/zenodo.3828935. URL: https://www.pytorchlightning.ai (cit. on p. 48).

Goodfellow, I., Y. Bengio, A. Courville (2016): *Deep Learning*. MIT Press (cit. on pp. 24, 26, 27).

Grid Search - an overview | ScienceDirect Topics (2024). URL: https://www.sciencedirect.com/topics/mathematics/grid-search (last access 01/30/2024) (cit. on p. 67).

Grosse-Wentrup, M., M. Buss (2008): Multiclass Common Spatial Patterns and Information Theoretic Feature Extraction. *IEEE Transactions on Biomedical Engineering* 55 (8). Conference Name: IEEE Transactions on Biomedical Engineering, 1991–2000. ISSN: 1558-2531. DOI: 10.1109/TBME.2008.921154. URL: https://ieeexplore.ieee.org/document/4473042 (last access 11/05/2023) (cit. on p. 16).

Gwon, D., K. Won, M. Song, C. S. Nam, S. C. Jun, M. Ahn (2023): Review of public motor imagery and execution datasets in brain-computer interfaces. *Frontiers in Human Neuroscience* 17. ISSN: 1662-5161. URL: https://www.frontiersin.org/articles/10.3389/fnhum.2023.1134869 (last access 01/13/2024) (cit. on pp. 32, 67).

Harris, C. R., K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T. E. Oliphant (2020): Array programming with NumPy. en. *Nature* 585 (7825). Number: 7825 Publisher: Nature Publishing Group, 357–362. ISSN: 1476-4687. DOI: 10.1038/s41586-020-2649-2. URL: https://www.nature.com/articles/s41586-020-2649-2 (last access 01/25/2024) (cit. on p. 44).

Human EEG Dataset for Brain-Computer Interface and Meditation (2021). en. DOI: 10.6084/m9.figshare.13123148.v1. URL: https://figshare.com/articles/dataset/Human_EEG_Dataset_for_Brain-Computer_Interface_and_Meditation/13123148/1 (last access 01/16/2024) (cit. on p. 36).

Jayaram, V., A. Barachant (2018): MOABB: trustworthy algorithm benchmarking for BCIs. en. *Journal of Neural Engineering* 15 (6). Publisher: IOP Publishing, 066011. ISSN: 1741-2552. DOI: 10.1088/1741-2552/aadeao. URL: https://dx.doi.org/10.1088/1741-2552/aadeao (last access 01/25/2024) (cit. on pp. 2, 81).

Kingma, D. P., J. Ba (2017): *Adam: A Method for Stochastic Optimization*. arXiv:1412.6980 [cs]. DOI: 10.48550/arXiv.1412.6980. URL: http://arxiv.org/abs/1412.6980 (last access 01/30/2024) (cit. on p. 68).

Lawhern, V. J., A. J. Solon, N. R. Waytowich, S. M. Gordon, C. P. Hung, B. J. Lance (2018): EEGNet: a compact convolutional neural network for EEG-based brain-computer interfaces. eng. *Journal of Neural Engineering* 15 (5), 056013. ISSN: 1741-2552. DOI: 10.1088/1741-2552/aace8c (cit. on pp. 2, 62, 63).

LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel (1989): Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation* 1 (4). Conference Name: Neural Computation, 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541. URL: https://ieeexplore.ieee.org/document/6795724 (last access 01/29/2024) (cit. on p. 23).

Lukas (2024): *ashleve/lightning-hydra-template*. original-date: 2020-11-04T14:30:09Z. URL: https://github.com/ashleve/lightning-hydra-template (last access 01/20/2024) (cit. on p. 49).

Morris, R. G. (1999): D.O. Hebb: The Organization of Behavior, Wiley: New York; 1949. eng. *Brain Research Bulletin* 50 (5-6), 437. ISSN: 0361-9230. DOI: 10.1016/s0361-9230(99)00182-3 (cit. on p. 8).

Motor cortex (2024). en. URL: https://www.kenhub.com/en/library/anatomy/motor-cortex (last access 01/01/2024) (cit. on p. 11).

Neurons and Neural Networks (n.d.). https://tpc2.abe.msstate.edu/abe4323/class18_neurons8/neurons8.html. Accessed on December 27, 2023 (cit. on p. 8).

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala

(2019): *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. arXiv:1912.01703 [cs, stat]. DOI: 10.48550/arXiv.1912.01703. URL: http://arxiv.org/abs/1912.01703 (last access 01/26/2024) (cit. on p. 48).

Rashid, M., N. Sulaiman, A. P. P. Abdul Majeed, R. M. Musa, A. F. Ab. Nasir, B. S. Bari, S. Khatun (2020): Current Status, Challenges, and Possible Solutions of EEG-Based Brain-Computer Interface: A Comprehensive Review. *Frontiers in Neurorobotics* 14. ISSN: 1662-5218. URL: https://www.frontiersin.org/articles/10.3389/fnbot.2020.00025 (last access 12/27/2023) (cit. on p. 10).

Sannelli, C., C. Vidaurre, K.-R. Müller, B. Blankertz (2019): A large scale screening study with a SMR-based BCI: Categorization of BCI users and differences in their SMR activity. en. *PLOS ONE* 14 (1). Publisher: Public Library of Science, e0207351. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0207351. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0207351 (last access 11/20/2023) (cit. on p. 32).

Schalk, G., D. McFarland, T. Hinterberger, N. Birbaumer, J. Wolpaw (2004): BCI2000: a general-purpose brain-computer interface (BCI) system. *IEEE Transactions on Biomedical Engineering* 51 (6). Conference Name: IEEE Transactions on Biomedical Engineering, 1034–1043. ISSN: 1558-2531. DOI: 10.1109/TBME.2004.827072. URL: https://ieeexplore.ieee.org/document/1300799 (last access 01/16/2024) (cit. on pp. 33, 35).

Scherer, R., C. Vidaurre (2018): Chapter 8 - Motor imagery based brain–computer interfaces. In: *Smart Wheelchairs and Brain-Computer Interfaces*. Ed. by P. Diez. Academic Press, 171–195. ISBN: 978-0-12-812892-3. DOI: 10.1016/B978-0-12-812892-3.00008-X. URL: https://www.sciencedirect.com/science/article/pii/B978012812892300008X (last access 12/20/2023) (cit. on p. 12).

Stieger, J. R., S. Engel, H. Jiang, C. C. Cline, M. J. Kreitzer, B. He (2021a): Mindfulness Improves Brain–Computer Interface Performance by Increasing Control Over Neural Activity in the Alpha Band. *Cerebral Cortex* 31 (1), 426–438. ISSN: 1047-3211. DOI: 10.1093/cercor/bhaa234. URL: https://doi.org/10.1093/cercor/bhaa234 (last access 11/23/2023) (cit. on pp. 31, 34, 35, 37).

Stieger, J. R., S. A. Engel, B. He (2021b): Continuous sensorimotor rhythm based brain computer interface learning in a large population. en. *Scientific Data*

8 (1). Number: 1 Publisher: Nature Publishing Group, 98. ISSN: 2052-4463. DOI: 10.1038/s41597-021-00883-1. URL: https://www.nature.com/articles/s41597-021-00883-1 (last access 01/13/2024) (cit. on pp. 1, 31, 33, 35, 37, 38, 55, 66, 83–86).

TMSi (2023): *The 10-20 System for EEG*. URL: https://info.tmsi.com/blog/the-10-20-system-for-eeg (last access 12/25/2023) (cit. on p. 7).

Understanding LSTM Networks – colah's blog (2024). URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (last access 01/29/2024) (cit. on p. 27).

Venthur, B., B. Blankertz (2014): *Wyrm, A Pythonic Toolbox for Brain-Computer Interfacing*. arXiv:1412.6378 [cs]. URL: http://arxiv.org/abs/1412.6378 (last access 01/25/2024) (cit. on pp. 2, 44).

Vidal, J. J. (1973): Toward Direct Brain-Computer Communication. *Annual Review of Biophysics and Bioengineering* 2 (1), 157–180. DOI: 10.1146/annurev.bb.02.060173.001105. URL: https://doi.org/10.1146/annurev.bb.02.060173.001105 (last access 12/05/2023) (cit. on p. 10).

Watanabe, S. (2023): *Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance*. arXiv:2304.11127 [cs]. DOI: 10.48550/arXiv.2304.11127. URL: http://arxiv.org/abs/2304.11127 (last access 01/30/2024) (cit. on p. 68).

Wikimedia Commons (2023): *EEG 10-10 system with additional information*. URL: https://commons.wikimedia.org/wiki/File:EEG_10-10_system_with_additional_information.svg (last access 12/25/2023) (cit. on p. 33).

Wikipedia (2023): *Action potential — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/Action_potential (last access 12/25/2023) (cit. on p. 6).

Xu, J., M. Grosse-Wentrup, V. Jayaram (2020): Tangent space spatial filters for interpretable and efficient Riemannian classification. eng. *Journal of Neural Engineering* 17 (2), 026043. ISSN: 1741-2552. DOI: 10.1088/1741-2552/ab839e (cit. on p. 17).

Yadan, O. (2019): *Hydra - A framework for elegantly configuring complex applications*. URL: https://github.com/facebookresearch/hydra (cit. on p. 49).

Yuan, H., B. He (2014): Brain-Computer Interfaces Using Sensorimotor Rhythms: Current State and Future Perspectives. *IEEE transactions on bio-medical engineering* 61 (5), 1425–1435. ISSN: 0018-9294. DOI: 10.1109/TBME.2014.2312397. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4082720/ (last access 11/20/2023) (cit. on p. 13).

Zubarev, I., G. Vranou, L. Parkkonen (2022): MNEflow: Neural networks for EEG/MEG decoding and interpretation. *SoftwareX* 17, 100951. ISSN: 2352-7110. DOI: 10.1016/j.softx.2021.100951. URL: https://www.sciencedirect.com/science/article/pii/S2352711021001795 (last access 01/21/2024) (cit. on pp. 2, 81).

Index