ETH ZÜRICH

MASTER OF SCIENCE IN ROBOTICS, SYSTEMS AND CONTROL

Leveraging Deep Learning for Real-time EEG Classification at Cybathlon's BCI-race

Author:
Frédéric DEBRAINE

Supervisors:
Prof. Dr. Nicole WENDEROTH
Ernest MIHELJ

Master Thesis

in the

Neural Control of Movement Lab
Department of Health Sciences and Technology

October 1, 2019

Contents

Introduction

From the motivations to use BCI. With the recent advances in technology and the advent of new fields such as 3D bio-printing, exo-skeleton and smart prosthetics, we are getting closer to being able to heal people from any traumatic injuries, diseases or congenital conditions, allowing them to recover a fully functional human body. For the past few decades, brain computer interface (BCI) technology has triggered an increasing amount of interest in both the scientific and public community as it conveys the hope to assist people with motor disabilities. Electroencephalography (EEG) data - the superficial brain signals - can be collected using electrodes placed along the scalp. The main objective of BCI is to analyze these data and decode it into a set of pre-defined actions that could be fed to the next generation of connected devices such as wheelchairs or exoskeletons, enabling an increase in autonomy and mobility for disabled people. But the motivation for such a technology has also been cultivated by a general interest in human computer interface (HCI) applications such as sleep quality analysis, brain to speech translation, monitoring of stroke or epilepsy and various other medical purposes. The main advantages of EEG-based BCIs compared to other methods, such as MRI or implants, are that they can rely on cheap and non invasive hardware, creating safe and accessible tools for patients and researchers. However major advances in BCI technology will not be possible without overcoming the numerous obstacles it unveils.

Current challenges. BCIs rely on a set of different technical fields making it necessary to have large panels of experts or broad knowledge in neuroscience (functioning of the brain), electrical engineering (electrodes & signal acquisition/processing) and machine learning (feature extraction and model training) available. Furthermore, the few publicly available datasets and the absence of real consensus about the procedure to follow for EEG data acquisition - as each lab uses specific hardware/software tools to acquire their dataset - constitute another challenge. Additionally, the inherent low signal to noise ratio makes it difficult to extract useful information from the EEG data and often requires a researcher to come up with handcrafted preprocessing steps that are specific to the acquisition protocol. Finally, the high variability of the signal among inter-session and inter-subject recordings probably remains one of the biggest challenges to overcome before being able to develop general models that would work reliably on anyone at anytime. Therefore research teams from all over the world are organizing international BCI competitions, such as Cybathlon 2016 and 2020, to promote BCI technology to the public and foster advances in this field.

Potential approaches, solutions & limits. Many original solutions arose from these competitions during the last decade. Among which a new learning paradigm, called deep learning, has shown its superiority to traditional machine learning approaches

especially in the field of computer vision. By leveraging powerful training algorithms with an artificial neural network composed of several layers of nodes interconnected by adaptive weights, we are now able to alleviate the feature extraction phase by partially automatizing it. While traditional methods rely on heavy preprocessing steps and hand-crafted features, deep learning indeed reduces this dependency on preprocessing steps while boosting state-of-the-art results. The main disadvantage of using neural networks is, nevertheless, the loss in interpretability of the model as it is now a black box whose weights are being optimized based on a objective function, which tells us nothing about the extracted features and their relationship with our current knowledge of the brain. Moreover, a huge amount of data is generally required to correctly train such models, making the data collection phase even more important and time-consuming than before.

Contributions. This master thesis aspires to propose a fully functional BCI composed of the following elements:

- An offline pipeline for the training and development phase that will take care of formatting, loading, preprocessing the data, extract features using one of the multiple proposed methods, train and validate the model and optimize the full pipeline.
- An online pipeline in which we will benchmark our trained models with relevant metrics regarding the real-time Cybathlon competition.
- Visualization tools for offline/online analysis comprising signal and embedding representation, model inference and metrics to provide better interpretability and easier debugging.

Structure. The remainder of this master thesis is structured as follows:

- Chapter 1 provides a non-exhaustive background introduction to EEG-based BCI.
- Chapter 2 presents a short introduction to the field of deep learning.
- Chapter 3 offers a description of the Cybathlon BCI 2020 competition.
- Chapter 4 tackles the data collection procedure for both publicly available and our own "Competition" dataset.
- Chapter 5 gives a large overview of the existing feature extraction techniques for EEG classification.
- Chapter 6 introduces our methodology for designing our BCI and our results.

List of Abbreviations

ANN	**A**rtificial **N**eural **N**etwork
BCI	**B**rain **C**omputer **I**nterface
CNN	**C**onvolutional **N**eural **N**etwork
CSP	**C**ommon **S**patial **P**attern
EEG	**E**lectro**E**ncephalo**G**raphy
EOG	**E**lectro**O**culo**G**raphy
ERP	**E**vent **R**elated **P**otential
ERD/S	**E**vent **R**elated **D**esynchronization/**S**ynchronization
FBCSP	**F**ilter **B**ank **C**ommon **S**patial **P**attern
GAN	**G**enerative **A**dversarial **N**etwork
GPU	**G**raphics **P**rocessing **U**nit
HCI	**H**uman **C**omputer **I**nterface
LDA	**L**inear **D**iscriminant **A**nalysis
LSTM	**L**ong **S**hort-**T**erm **M**emory
M1	Primary Motor Cortex
MI	**M**otor **I**magery
MIBIF	**M**utual **I**nformation based **B**est **I**ndividual **F**eature
RBF	**R**adial **B**asis **F**unction
RG	**R**iemannian **G**eometry
RNN	**R**ecurrent **N**eural **N**etwork
SPD	**S**ymmetric **P**ositive **D**efinite
SVM	**S**upport **V**ector **M**achine
STFT	**S**hort-**T**ime **F**ourier **T**ransform

1 EEG 101

This chapter introduces electroencephalography (EEG) in its historical context and gives a short overview of the underlying biological principles responsible for brainwaves. It then focuses on motor imagery (MI)-EEG-based BCI and defines potential regions of interest, in terms of brain areas and frequency spectrum, before describing the EEG data acquisition procedure. Finally, a few typical application examples of this technology are discussed.

Some EEG history. Electroencephalography is an imaging technique used to record the electrical activity of the brain. First cerebral activity recordings were performed by Richard Caton in 1875 while using a galvanometer in order to capture electrical impulses from the surfaces of living animal brains. His work lead to the development of the electroencephalogram by Hans Berger in 1924, who introduced the first human EEG recording. An example of EEG signal recording is given in Figure 1.1. For the past few decades, EEG has been used extensively in neuroscience research to study brain activity patterns and diagnose neurological disorders such as epilepsy or Alzheimer's disease. More recently, this technology has also been utilized in a brain computer interface context in order to establish new means of communication between humans and machines.

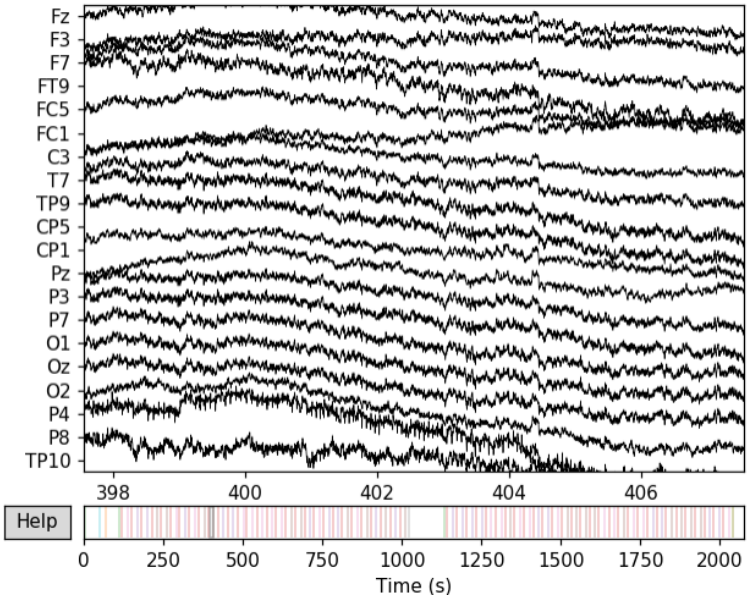
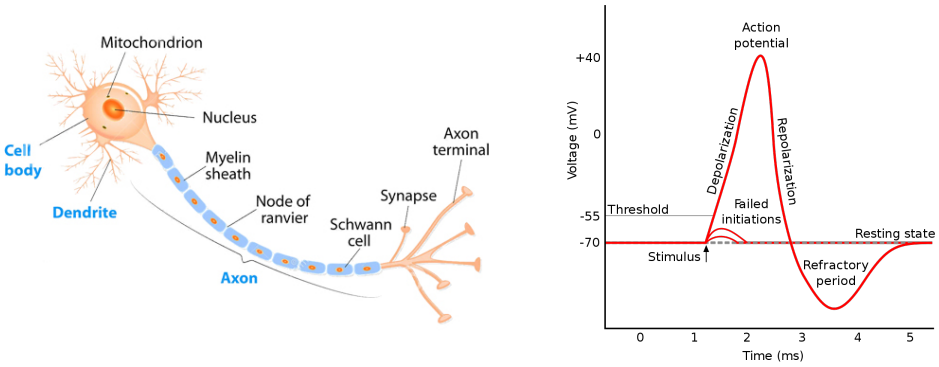


Figure 1.1: Elecroencephalograph. Multi-channel EEG signal visualized using the MNE Python library[1].

[1] https://martinos.org/mne/stable/index.html



(a) Typical structure of a neuron. [1] **(b)** Action potential signal. [2]

Figure 1.2: Neuronal signal. The electrical activity of the brain stems from ion transfers happening on a neuronal scale and leading to action potentials. These signals propagate from the neuron body cell towards its synapses through the axon.

Biological source of the EEG signal. The electrical activity of the brain is maintained by its billions of interconnected neurons (Fig. 1.2a). On a micro-scale, neurons are electrically charged by pumping ions in and out their membrane causing them to be momentarily polarized. The depolarization of the neuron's membrane causes the depolarization of adjacent locations leading to a nerve impulse or spike - also called action potential - that propagates along the axons to the synapses and then to other neurons (Fig. 1.2b). At a macro level, the electrical potential generated by an individual neuron is too weak to be measured by an EEG electrode. It is rather the summation of synchronous nerve impulses of thousands of neurons having similar spatial orientations that can be detected on the scalp as brain waves by EEG (Fig. 1.3).

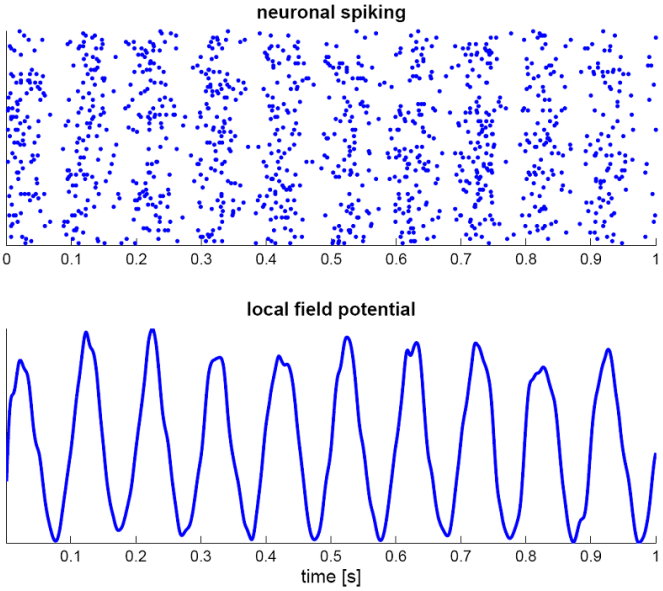
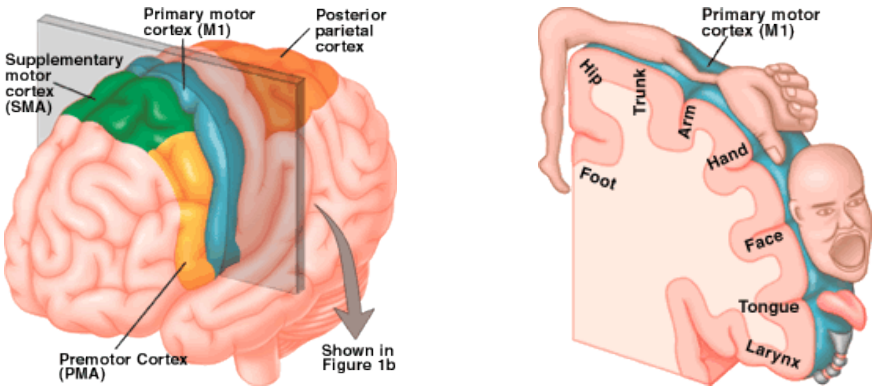


Figure 1.3: Neural oscillations. Simulation of macroscopic oscillations (10Hz) from synchronized patterns of action potentials. The upper panel represent individual neuron spikes as dots throughout time (horizontal axis) and the lower panel reflects the resulting activity of all action potentials being summed up. [3]

MI-EEG-based BCI. A brain computer interface can be conceptualized as means of communication between a human and a machine based purely on brain signals. This is achieved using either invasive methods that rely on neural implants or non-invasive methods which collect brain signals without the need of surgical interventions. EEG-based BCIs are often considered as a safe alternative to invasive BCI technologies and also carry additional advantages compared to other non-invasive methods such as magnetoencephalography (MEG) and functional magnetic resonance imaging (fMRI). Despite their high susceptibility to noise and low spatial accuracy, EEG-based BCIs are among the most studied types of brain computer interfaces thanks to their high temporal resolution, low setup cost and ease of use. Motor imagery is one of the main BCI paradigms, along with visual evoked potentials (VEP), to be used for neuroimaging. While former involves the imagination of limb movements to activate the sensorimotor cortex, latter relies on the specific electric potential appearing in the signal after a visual stimulus is presented to the subject. In our case, we will focus solely on motor imagery as we expect the subject to exploit the same areas of the brain that are commonly utilized while performing motor tasks. Furthermore, this strategy is believed to be more in adequation with the Cybathlon BCI challenge as the pilot should actively try to control the avatar instead of passively relying on the decoding model to interpret his visual input.

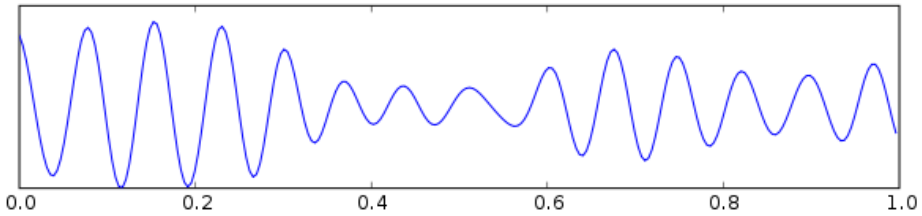
Brain areas responsible for motor function. The regions of the brain involved in motor function are numerous (Fig. 1.4a). However, the primary motor cortex, or M1, is often considered to be the principal brain area responsible for it. This region is located in the frontal lobe of the brain and generates neural impulses that control the execution of movement. Whereas secondary motor areas, such as the supplementary motor cortex (SMA) and Posterior parietal cortex (PPC), are involved in the planning of complex movements and the transformation of visual information into motor commands respectively. Each part of the body is represented in the primary motor cortex and assigned to a specific location area (Fig. 1.4b). The amount of brain matter allocated exhibits the degree of control M1 has over that limb. Body parts that need to handle complex movements, such as the hands and fingers, require more cortical space, and are consequently represented in larger areas of the primary motor cortex.



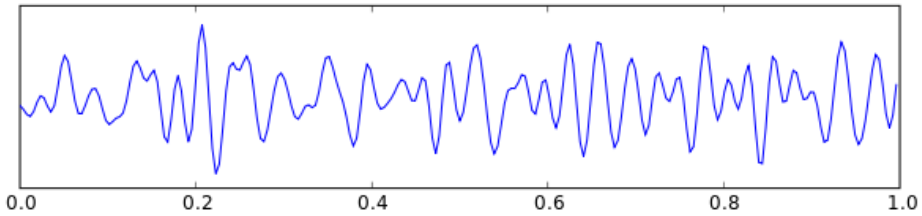
(a) Brain structure of the motor system. **(b)** Body map encoded in the M1 section.

Figure 1.4: Brain topography. Representation of the motor areas in the brain. [4]

Frequency bands in neural oscillations. Scalp EEG activity displays oscillations at a variety of frequencies ranging approximately from 0 to 100 Hz. These oscillations stem from the synchronized activations of neurons associated to different mental states and body movements. In the context of motor imagery, the region of interest in the frequency spectrum lies in the range from 8 to 13 Hz, that comprises mu waves, as well as in the range from 14 to 30 Hz, which corresponds to beta waves. Mu rhythms reflects the synchronous firing of motor neurons in rest state and are generally attenuated during active movements (Fig. 1.5). Similarly, beta waves are associated with muscle contractions and are suppressed prior to and during movement changes (Fig. 1.6). These frequency bands are therefore a suitable region of interest to look for in MI-EEG signals as they are closely linked to motor behavior.



(a) Mu waves [8-13] Hz



(b) Beta waves [14-30] Hz

Figure 1.5: Brain waves. Principal frequency bands associated to motor imagery. [5]

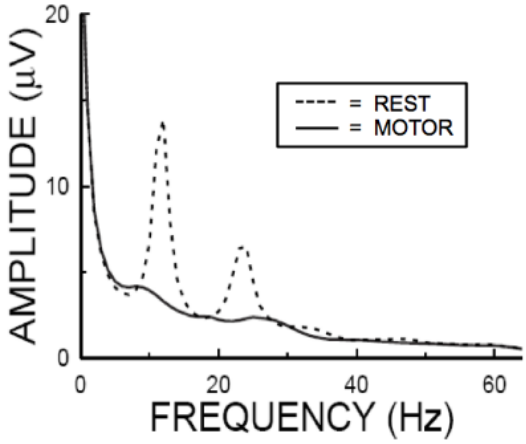
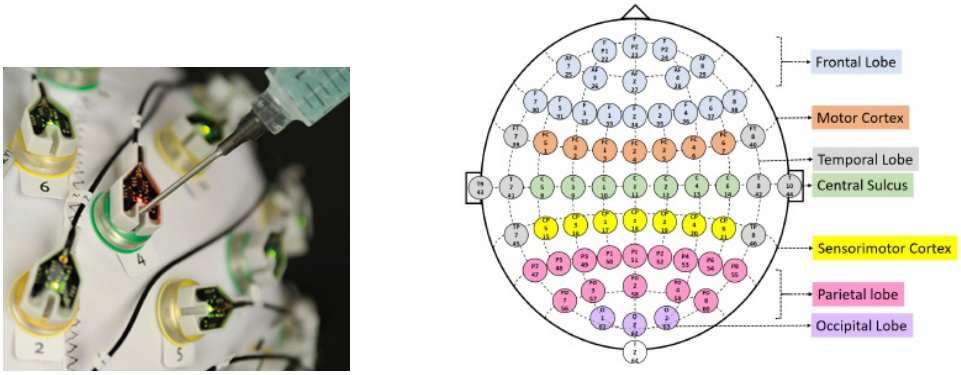


Figure 1.6: Frequency spectrograms. At rest, we expect high amplitude mu and beta waves (dashed). During a motor action, those same brain oscillations are expected to be attenuated (plain). [6]

EEG data acquisition. EEG data can be recorded using electrodes placed along the scalp. There exist different categories of electrodes. While wet electrodes involve the use of conductive gel to reduce the impedance between the electrodes and the scalp (Fig. 1.7a), dry electrodes generally rely on spring loaded pins to ensure mechanical contact with the skin. Active electrodes, in opposition to passive electrodes, embed a differential amplifier that boost the voltage signal w.r.t. the reference electrode. Despite being more expansive and cumbersome to wear, active wet electrodes are usually preferred in order to get a cleaner EEG signal. The number of electrodes can vary from just a few to 128 and are usually attached to a skullcap and placed on the scalp using the 10-20 international system, which allows for reproducible positioning of the electrodes and specifies a standardized channel naming convention (Fig. 1.7b). During a MI-EEG recording, the subject is asked to imagine a specific motor task without performing the movement. The beginning and the end of each MI phase are often delimited by visual or acoustic cues. A trial is a chunk of recording comprised of a focus period, a MI phase and a break. EEG recordings are generally collected in sessions of multiple consecutive trials in order to reduce the time spent on preparing and re-placing the electrodes on the scalp. The captured signal is then digitized using sampling frequencies ranging from 256 to 512 Hz before being pre-filtered to remove movement artifacts and ambient electromagnetic noise.



(a) Conductive gel being applied to a cap of active electrodes. [7]

(b) 64 electrodes positioning using the international 10-20 system. [8]

Figure 1.7: Electrodes system. Type of electrodes and positioning convention used during the EEG data acquisition.

BCI applications. The possible domains of application for BCI technology are countless and can even be extended outside clinical environments. Sleep analysis (Fig. 1.8), for instance, has greatly benefited from the advances in BCI from the past few years and one could easily imagine such technology in a consumer device that would help fight insomnia troubles and could potentially improve the life quality of millions of people. BCI also has a considerable potential for revolutionizing the way humans communicate with machines for applications such as exoskeleton control for disabled people, workers but also the military.

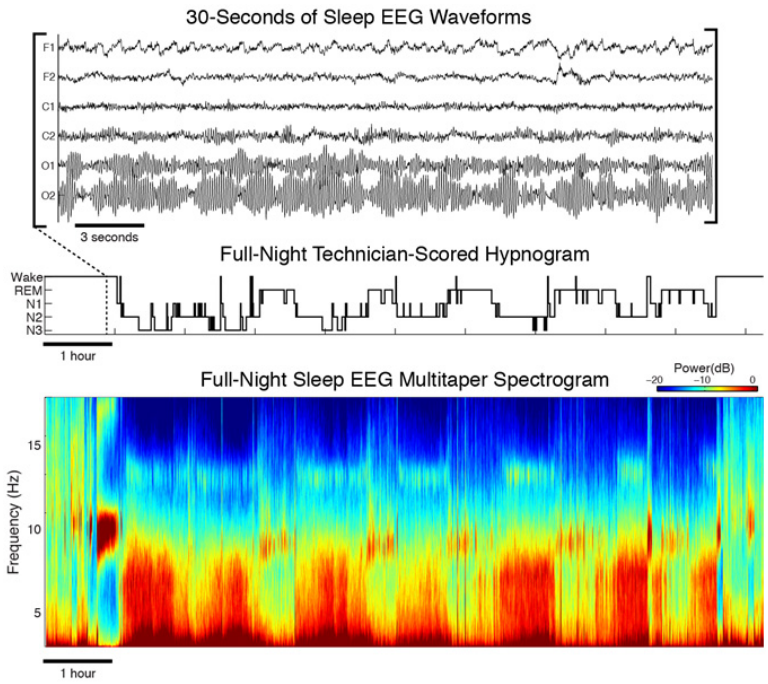


Figure 1.8: EEG use-case for sleep analysis. Top graph represents 30s of raw EEG recording used to create a hypnogram (middle graph) that categorizes the sleep into stages. Bottom graph provides a visualization of the frequency power of EEG oscillations over time (blue is low and red is high). [9]

2 Deep learning 101

This chapter gives a compact and general overview of the deep learning field. We present here the general architecture of artificial neural networks, their training procedure and the framework we will use throughout our project.

Some deep learning history. Deep learning is a sub-field of machine learning based on artificial neural networks (ANNs) that aims to automatically extract features in a hierarchical manner. The first working ANN was developed by Alexey Ivakhnenko in 1965 using a multilayer perceptron [10], a simplified model of biological neurons. However, because of the lack of computational resources, the field of deep learning did not become popular before the 90s, when Yann LeCun trained a deep neural network to recognize handwritten ZIP codes [11]. With the advent of more powerful machines, GPUs in particular, artificial neural networks are now becoming a standard in AI industry as they are able to produce results comparable, or even sometimes superior, to human experts. Despite being massively present in research for the past decade in fields such as computer vision and natural language processing, deep learning is only in its infancy when it comes to BCI technology.

Basic ANN architecture. Artificial neural networks are inspired by the biological processes and distributed communication happening in our brain. Basic ANNs are composed of multiple layers of aggregated nodes that transmit information from one layer to another through node interconnections, while applying non-linear operations on the signal. Each interconnection's strength is represented by a weight that can either increase or decrease during the training process of the neural network (Fig. 2.1). On one hand, the input layer is specifically designed to "fit" the dimension of the input data. On the other hand, the output layer typically contains as many nodes as labels[1] and returns a score value for each class that can be later interpreted as a probability via specific activation functions (see equations [2.1] and [2.2]). In the hidden layers, a variety of activation functions [12] can be used to extract the non-linear features present in the signal.

For a binary classification problem, a sigmoid is often used to convert the score value z of the single output neuron into a probability

$$\sigma(z) = \frac{1}{1 + e^{-z}} \in [0, 1] \tag{2.1}$$

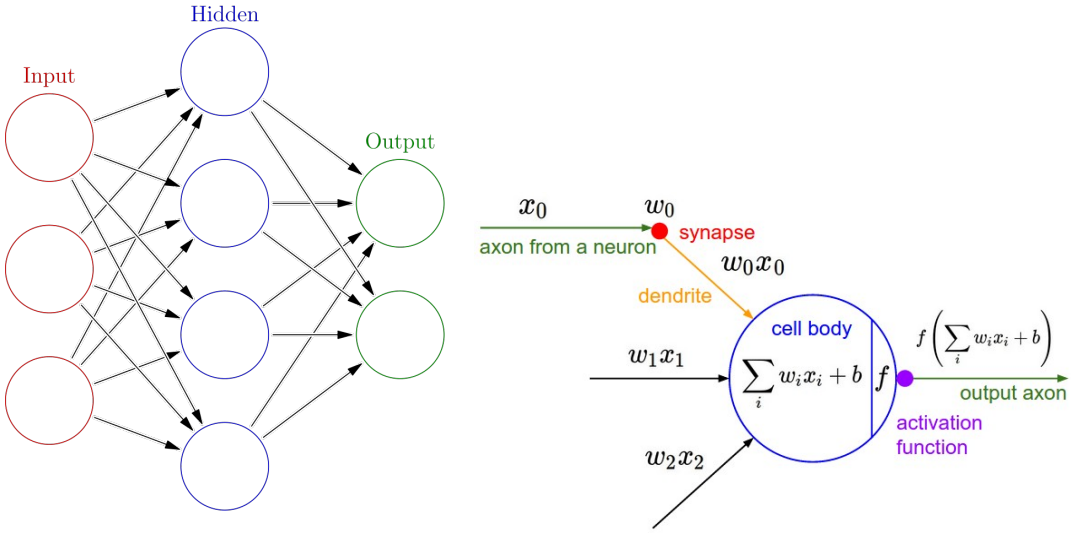
This value represents the predicted probability that the input example belongs to class 1.

[1] In the case of binary problems, one can choose to use either one or two output neurons. In this chapter, we always consider the former.

For a classification problem with $N_{classes} > 2$ categories, one should employ the softmax activation function

$$\sigma(z_c) = \frac{e^{z_c}}{\sum_{i=1}^{N_{classes}} e^{z_i}} \in [0,1] \tag{2.2}$$

where z_c is the score value returned by the output neuron associated to class c. Similarly, this value represents the predicted probability that the input example belongs to class c.



(a) Simple neural network structure. [13] **(b)** Artificial neuron model. [14]

Figure 2.1: Basic ANN architecture. Typical neural networks (a) are composed of nodes (circles) aggregated in layers that can be categorized as input, hidden or output layers. The number of hidden layers defines the depth of the architecture. Consecutive layers are connected by weights (arrows) that represent how the signal will be amplified or attenuated before being added to other signals arriving to the next neuron (b).

Training ANNs. The training procedure of a neural network is an iterative process that consists of feeding the training data to the model and computing the errors between the network's output and the expected output (feedforward propagation) as well as the weight adjustments (backpropagation). One iteration of this cycle is called an epoch and leads the network to progressively learn to fit its input data. The procedure of training a network involves the following notions that are important to define:

- ***Feedforward propagation.*** This first step consists of presenting the training data to the input layer and computing the activations of each neuron, layer by layer, until the output of the network is reached (see equation [2.3]). In practice, a set of input training examples, called batches, are considered in order to accelerate the training through parallel computation and have less noisy weight adjustments. The batches should be disjoint subsets of equal size and cover the full training dataset when considered all together.

The feedforward operation applied at a layer l, fully connected to layer $(l-1)$, can be expressed as the vectorized operation

$$\mathbf{z}^{(l)} = f(\mathbf{W}^{(l)}\mathbf{z}^{(l-1)} + \mathbf{b}^{(l)}) \tag{2.3}$$

where f designates a non-linear activation function, the vector $\mathbf{z}^{(l)} \in \mathbb{R}^{n_l}$ represents the activations of each of the n_l neurons in the layer l and the matrix $\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ holds the interconnection weights between the neurons of layers $l-1$ and l, and $\mathbf{b}^{(l)}$ the bias vector at layer l.

- ***Cost function.*** The cost function (or loss function) is a metric that aims at quantifying the error between the expected output (ground truth) of a neural network and its current feedforward output. In our context of supervised classification problem, we will only consider the following popular cost functions:
  - Binary cross-entropy is used when we are dealing with only two categories or classes and consists of the following scalar value

$$J(\boldsymbol{\theta}) = -\frac{1}{N_{batch}} \sum_{i=1}^{N_{batch}} \left[y_i \log\left(z_i^{(L)}\right) + (1-y_i) \log\left(1 - z_i^{(L)}\right) \right] \tag{2.4}$$

    where $\boldsymbol{\theta}$ represent the set of all trainable weights in the neural network, N_{batch} is the batch size, $y_i \in \{0,1\}$ and $z_i^{(L)} \in [0,1]$ are respectively the ground truth and predicted output (probability of label 1) for the input batch example i.

    The first term in the sum penalizes the network when an example i has an expected value $y_i = 1$ but predicted value $z_i^{(L)}$ close to 0 and vice-versa for the second term.

  - Categorical cross-entropy is used as soon as there are more than two categories to classify and is computed as follows

$$J(\boldsymbol{\theta}) = -\frac{1}{N_{batch}} \sum_{i=1}^{N_{batch}} \sum_{c=1}^{N_{classes}} I\left[y_i = c\right] \log\left[z_{c,i}^{(L)}\right] \tag{2.5}$$

    where $I\left[y_i = c\right]$ is the indicator function that equals to 1 when the example's expected label is c and 0 otherwise and $z_{c,i}^{(L)}$ is the activation of the neuron associated to class c in the last later L for the input batch example i.

- ***Backpropagation.*** This step consists of a recursive algorithm whose goal is to compute the weights updates in order to improve the model. This can be formulated as the optimization problem

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta}) \tag{2.6}$$

  We start by calculating the gradient of the cost function with respect to each weight of the last layer L and then propagate this gradient through the earlier layers with the help of the chain rule.

The gradient of the cost function with respect to the weight connecting the layers $l-1$ and l can be computed as a scalar-by-matrix partial derivative [15]

$$\begin{aligned}\Delta \mathbf{W}^{(l)} &= \frac{\partial J}{\partial \mathbf{W}^{(l)}} = \boldsymbol{\delta}^{(l)} \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{W}^{(l)}} \\ &= \sum_{j=1}^{n_l} \delta_j^{(l)} \frac{\partial z_j^{(l)}}{\partial \mathbf{z}_j^{(l-1)}} \\ \boldsymbol{\delta}^{(l)} &= \frac{\partial J}{\partial \mathbf{z}^{(l)}} = \boldsymbol{\delta}^{(l+1)} \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{z}^{(l)}}\end{aligned} \tag{2.7}$$

where J is the scalar cost function defined in equations [2.4] or [2.5], $\mathbf{W}^{(l)}$ is the matrix of weights connecting layers $(l-1)$ and l, $\boldsymbol{\delta}^{(l)}$ is the row vector corresponding to the backpropagated error signal at layer l and $\mathbf{z}^{(l)}$ represents the activation column vector of layer l defined in equation [2.3].

- ***Learning rule.*** This is the algorithm which defines how the parameters of the neural network, or weights, are being updated through each epoch. Many different algorithms [16] have been designed to tackle this non-convex optimization problem and to deal with the numerous local minima of the cost function. A simple batch gradient descent leads to the following update rule

$$\mathbf{W}^{(l)}(t+1) = \mathbf{W}^{(l)}(t) - \eta \Delta \mathbf{W}^{(l)} \qquad \forall l \in [1; L] \tag{2.8}$$

where t is the current epoch, η is the learning rate parameter and is the weights update computed in [2.7].

Validation. This step is an essential component of any machine learning and deep learning approach as it allows to tune the model's hyperparameters such as the number of hidden layers or the type of activation functions. In order to evaluate if the model is correctly learning, we regularly compute the loss (or any other metric) associated to a validation set, while training. This validation set should be disjoint to the training set to provide an unbiased validation metric estimate. A popular technique, called early stopping, consists in using this metric to also automatically decide when to stop the training. In practice, the training process is stopped when the validation metric has not improved for a fixed amount of epochs, called patience.

Overfitting and regularization techniques. A very common issue encountered in deep learning is called overfitting. This happens when the neural network is also learning to fit the noise inherent to the training data and is not able to generalize correctly on new data. This problem can be detected by monitoring the validation loss for specific patterns (Fig. 2.2). Regularization techniques are here to help prevent overfitting and can take several forms

- ***Early stopping.*** The training of the network is stopped as soon as the validation loss does not decrease anymore or is increasing, which is often a sign that the model starts to overfit the training data.
- ***Dropout.*** Random node activations are dropped (zeroed-out) during training in order to force the network to learn robust and general features.
- ***Weight constraints.*** We can incorporate a regularization term in the cost function to penalize for large weight values. Popular weight constraints include

max-norm and L2-norm and usually lead to a more stable model that is less sensitive to statistical fluctuations present in the input data.

- ***Batch normalization.*** This technique normalizes the layers activations in order to keep the input distribution of each consecutive layer similar. In practice, adding batch normalization after each layer, greatly improves training time, reduces overfitting and sometimes boosts accuracy.

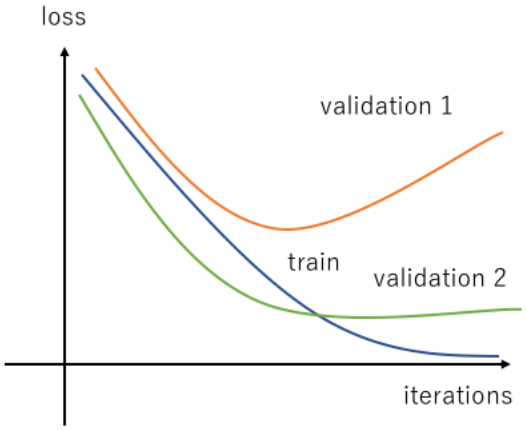


Figure 2.2: Validation loss monitoring. During training both the train and validation losses should be monitored. While the first is (almost) always expected to converge towards zero, the second typically either draws a hyperbole (orange curve) or stop converging after some epochs (green curve). The case of the orange curve displays signs of overfitting and should be avoided. [17]

Software. Deep learning frameworks are very useful to rapidly design and train neural networks, without having to deal with heavy matrix calculations. In our project we use Keras, a high level API, that leverages Tensorflow as backend. Keras allows for easy and rapid prototyping via the use of sequential models (the architecture is defined layer by layer) and many fully configurable modules (neural layers, optimizers, activation functions, regularization schemes, etc...).

3 Cybathlon BCI competition

This chapter gives a short overview of the Cybathlon[1] competition and introduces the 2020 BCI challenge, its rules and the motivation for such an event. Also, a few words about the former 2016 BCI competition are mentioned.

Figure 3.1: Cybathlon logo. [18]

General competition and goals. Cybathlon is a two-day event organized by ETH that is going to be held in May 2020. This contest consists of challenges in 6 disciplines in which people with physical disabilities compete against each other using state-of-the-art technical assistance systems. The challenges should demonstrate tasks that are relevant to the everyday life with the goal of connecting the public to the advances in research while promoting these new technologies.

BCI competition. The BCI race is one of the 6 disciplines that gathers pilots with quadriplegia to control avatars in a computer game using their brain waves (Fig. 3.2). During the competition, pilots are connected to the game via a cap of electrodes recording their brain activity while sitting in front of their individual screen, showing their avatar (Fig. 3.3) as well as the positions of the competing avatars. Each team is expected to come up with a full BCI and is allowed to use MI-EEG to detect brain signals as well as other methods such as near infrared spectroscopy (NIRS). The decoding of the pilot's brain signals is up to each team but should not rely on ocular movements.

Figure 3.2: BCI Pilot. Game player wearing a cap of electrodes during the BCI Cybathlon Competition 2016. [19]

[1] https://cybathlon.ethz.ch/

Game rules. The goal of this game environment is to test the reliability and precision of each team's BCI. The avatars receive orders from the decoding model, among a set of possible actions, at specific times of the game. If the BCI outputs the wrong action, the avatar is slowed down as a penalty. The winning team is the one that crosses the finish line first.

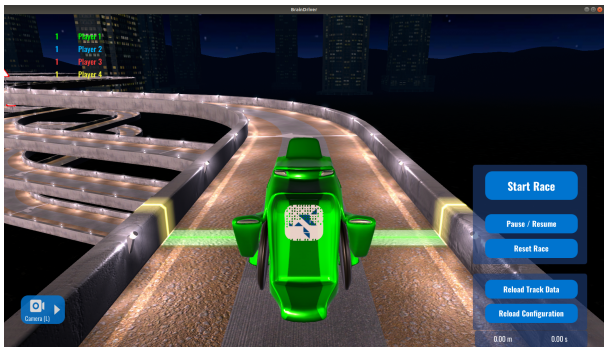


Figure 3.3: Game preview. Each pilot needs to control their designated avatar (here in green) using motor imagery brain signals only.

Past edition in 2016. The last edition of the Cybathlon competition, that took place in October 2016 at ETH Zurich, also included a BCI discipline. This competition gathered 11 international teams and took the form of a race similar to the one introduced earlier (Fig. 3.4), in which avatars are asked to perform a set of actions triggered by the brain of the pilot. Many of the teams, including the winner team, Brain Tweakers, relied on traditional statistical and machine learning approaches [20, 21, 22]. Despite the impressive results obtained during this competition, we expect the upcoming Cybathlon edition to reflect the exciting advances in BCI technology and the rise of new approaches such as deep learning.

Figure 3.4: Brain Runners game. Preview of the game used during the BCI race of Cybathlon competition 2016. [23]

Motivation for BCI. Despite being a relatively new field, BCI technologies are expected to have a great impact in the near future. BCI is a motivating field not only because it is a quite challenging and open problem, but also because it will drive the future of how humans interact with machines.

4 Data collection

This chapter presents the publicly available datasets employed for benchmarking as well as our recorded pilot dataset used to train our decoding models for the competition. Also, information about the hardware involved, the MI paradigms and the recording procedure are given.

4.1 Publicly available datasets

The objective behind leveraging publicly available datasets is two-fold. First, it will allow us to benchmark our models and compare them with some state-of-the-art approaches discussed in Chapter 6. Second, these datasets are known to be clean and would thus be easier to work with, in a first stage, in order to develop our pipeline while the pilot's Competition dataset is being recorded. These datasets could also be employed to assess the benefit of a inter-subject transfer learning strategy when leveraging a deep learning model.

BCIC IV 2a dataset. The BCI Competition IV 2a, or Graz dataset A, is a publicly available dataset comprising the EEG data from 9 subjects. The cue-based paradigm consists of four different motor imagery movement tasks, namely left hand, right hand, both feet and tongue. The dataset was recorded over two sessions on different days and each session contains 288 trials per subject, i.e. 72 trials per class. As stated in the dataset description[1], the first session should be used for training the decoding model, whereas the second session should be exclusively considered for testing it, and thus should remain unseen by the model until evaluation. During the recording, the subject stays in a sitting position in front of a screen. Each trial complies to the same following timing procedure (Fig. 4.1):

- At t=0s, a fixation cross appears on the black screen with a short acoustic tone.
- At t=2s, a cue appears on the screen for 1.25s. It takes the form of an arrow pointing either to the left, right, up or down and is assigned to a specific motor imagery task.
- The subject is asked to perform the movement imagination as the cue disappears, until the fixation cross disappears from the screen (from t=3s until t=6s).
- A short break of a few seconds follows until the next trial starts.

The signals were recorded using twenty-two dry passive electrodes placed over the scalp according to the international 10-20 system. They were sampled at 250 Hz and band-pass filtered between 0.5 Hz and 100 Hz.

[1] http://www.bbci.de/competition/iv/desc_2a.pdf

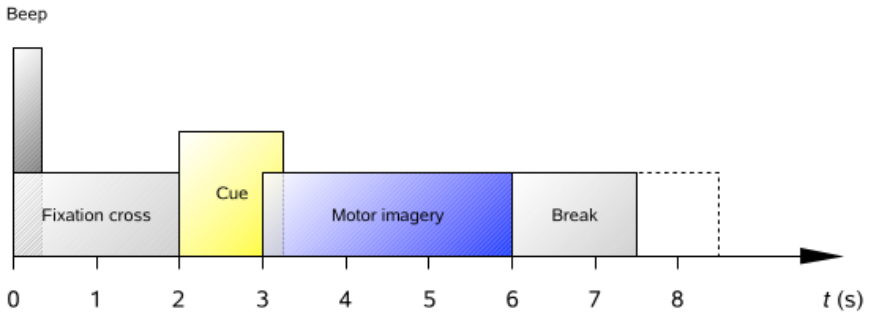


Figure 4.1: Recording procedure. Timing procedure for each trial in the BCIC IV 2a dataset.

BCIC IV 2b dataset. The BCI Competition IV 2b, or Graz dataset B, is another publicly available dataset comprising the EEG data from 9 healthy subjects. Contrary to the previous dataset, this one considers only two motor imagery movement tasks: left hand and right hand. The full dataset was recorded over five sessions, with 400 trials per subject for each session, i.e. 200 trials per class per session. As stated in the dataset description[2], the first three sessions should be used as training dataset while the two remaining should only be considered for testing. The recording procedure of the last three sessions slightly differs from the two previous ones as they incorporate feedback information during the motor imagery phase. Each trial of the first two sessions (no feedback) complies to the following timing procedure (Fig. 4.2):

- At t=0s, a fixation cross appears on the black screen for 3s with a short acoustic tone at 2s.
- At t=3s, a cue appears on the screen for 1.25s. It takes the form of an arrow pointing either to the left, right and is assigned to a specific motor imagery task.
- The subject is asked to perform the movement imagination over a period of 3s (from t=4s until t=7s).
- A short break of a 1 to 2s follows until the next trial starts.

The three following sessions (with feedback) have the following timing procedure:

- At t=0s, a grey smiley appears on the black screen for 3s with a short acoustic tone at 2s.
- At t=3s, a cue appears on the screen for 4.5s. It takes the form of a colored smiley moving either to the right or the left depending on a classifier output. As the smiley moves to the correct direction its color changes to green, whereas if it moves to the false direction it changes to red.
- The subject is asked to keep the smiley on the correct side as long as possible.
- A short break of a few seconds follows until the next trial starts.

The signals were recorded using three dry passive electrodes (C3, Cz and C4) placed over the scalp according to the international 10-20 system. They were sampled at 250 Hz and band-pass filtered between 0.5 Hz and 100 Hz.

[2] http://www.bbci.de/competition/iv/desc_2b.pdf

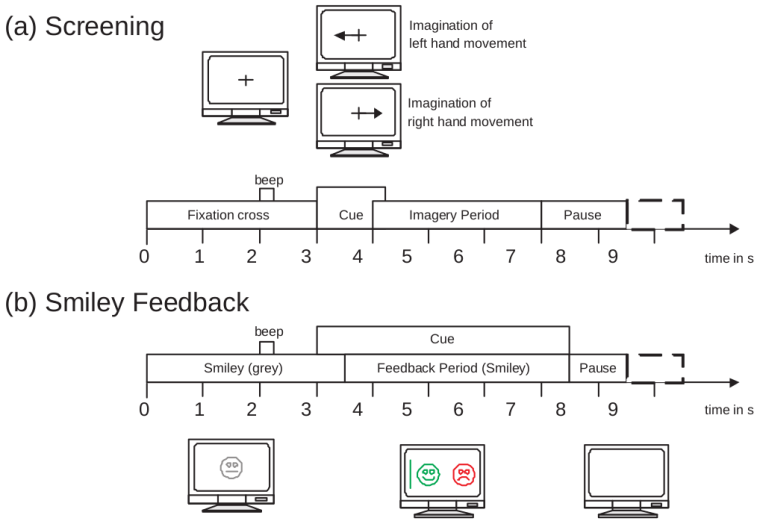


Figure 4.2: Recording procedure. Timing procedure in the BCIC IV 2b dataset for each trial of the first two sessions (a) and the last three sessions (b).

4.2 Our Competition dataset

Our Competition dataset comprises the EEG data from our pilot subject who will participate to the Cybathlon BCI competition. The cue-based paradigm consists of three motor imagery tasks and a resting state. We use the following mapping for the avatar actions (Fig. 4.3):

- Closing the left hand: the avatar should turn to the left.
- Closing the right hand: the avatar should turn to the right.
- Closing both hands simultaneously: the avatar should turn its headlights on.
- Resting: the avatar should not perform any action.

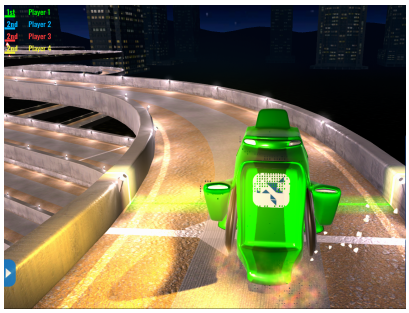
The data is recorded over multiple sessions on different days and each session contains a variable amount of trials for each motor imagery class (between 20 and 80 trials). Several recording strategies were applied over different sessions in order to find the most suitable, both in terms of convenience for the subject and decoding performance. All sessions follow a timing procedure comparable to the BCIC IV 2a dataset but might differ in terms of number of trials per class, the amount of rest between two consecutive trials and the record of the same or random MI tasks in consecutive trials. The signals are recorded using sixty-one wet active electrodes placed over the scalp according to the international 10-20 system (Fig. 4.4) and are under-sampled at 250 Hz with a notch filter at 50 Hz.

In the near future, we aim to include feedback into our recording procedure in order to gather more realistic data. The subject will be sitting in front of a screen displaying the Cybathlon game interface in self-playing mode. Each game track is composed of consecutive sub-tracks in which the avatar is asked to perform a specific action. This action can be performed as soon as the avatar passes a visual cue, that comprises left turns, right turns or headlight warning panels. Two consecutive visual cues are

always separated by a short amount of straight road during which no action should be performed by the avatar.

(a) Idle.

(b) Headlights on.



(c) Left turn.

(d) Right turn.

Figure 4.3: Avatar's action space. The pilot should perform the associated motor imagery task such that the avatar triggers the action after it crosses the white line cue.

Figure 4.4: Recording cap. 64 Brain Products actiCAP active electrodes.

5 Feature extraction and classification

Feature extraction consists of finding relevant and non-redundant information in the signal data in order to facilitate the learning and detection of a desired task. This step is typically followed by a classifier whose role is to categorize the extracted features into classes. In this chapter, we explore the different stages of feature extraction as well as two widely used classifiers. We will first present several commonly used methods before exploring state-of-the-art approaches relying on machine learning and deep learning techniques.

Stages in feature extraction. Feature extraction can occur at different stages of a BCI pipeline. The first stage is pre-processing and comprises the application of mathematical operations on the data in order to reduce its intrinsic noise and thus improve the signal to noise ratio. Examples of such operations are given in Section 5.1. The second stage is usually implemented in the trained model and often aims at finding a suitable representation of EEG signals as well as reducing their dimensionality thanks to optimization methods. Commonly used paradigms are discussed in Section 5.2.

5.1 Feature extraction methods in the pre-processing stage

There are many different possible pre-processing steps that can be applied on EEG data. This section presents a non-exhaustive list of operations along with with their description, role and impacts on the signal.

5.1.1 Common pre-processing steps

Re-referencing. Re-referencing expresses the voltage at each EEG channel w.r.t. a new reference, the signal of the reference being subtracted from each EEG channel. The choice of the reference depends on the task performed and should not be located near an area where we expect to extract some information. It should ideally contain little signal but the same noise distribution as the other electrodes. A popular re-referencing procedure is Common Average Reference (CAR) and is based on the principle that the activity of the whole head at every moment sums up to zero. This allows to distribute the "responsibility" over all electrodes that should be uniformly placed on the scalp, rather than assigning it to an electrode at a specific location which could result in a biased signal. The following equation describes the mathematical operation applied on the original EEG data x of channel index i to

obtain the re-referenced EEG data x^*:

$$x_i^*(t) = x_i(t) - \frac{1}{N_c}\sum_{j=1}^{N_c} x_j(t) \tag{5.1}$$

where N_c is the number of EEG channels.

Temporal filtering. The goal of temporal filtering is to remove components of the signal that are not within a given frequency domain. This step allows the extraction of mu and beta rhythms that are typical for motor imagery as well as the attenuation of power line noise, muscle movements and electrodes drift. More information about digital signal processing can be found in [24].

Six-sigma clipping. This operation aims at rectifying outliers in the EEG data such as high voltage spikes that are due to unintentional movements. This is an important step when using Z-score normalization [5.3] as these outliers tend to bias the temporal mean and standard deviation, and would lead to very small normalized data values.

$$x_i^*(t) = \begin{cases} -6 * \sigma_i & \text{if } x_i(t) \leq -6 * \sigma_i \\ 6 * \sigma_i & \text{if } x_i(t) \geq 6 * \sigma_i \\ x_i(t) & \text{otherwise} \end{cases} \tag{5.2}$$

Z-score normalization. Standardization, or Z-score normalization, consists of subtracting the temporal mean to the signal and dividing the result by the temporal standard deviation of the original signal. This operation is done for all EEG channels and aims at bringing the data on the same scale, thus reducing the complexity of the problem for the model. This pre-processing step is particularly useful when using neural networks as none of the neurons are wasted on learning this normalization and can be used more efficiently to extract relevant features.

$$x_i^*(t) = \frac{x_i(t) - \mu_i}{\sigma_i} \tag{5.3}$$

5.1.2 Limitations

However, pre-processing of EEG data comes with its own specific challenges. First, finding the optimal set of operations to perform on the signal requires a time-consuming trial and error approach. Second, one should keep in mind that such operations are not harmless as they tend to distort the EEG temporal structure. Filtering, for instance, adds a phase delay to the pre-processed signal. Therefore, the number of pre-processing steps should be kept at a strict minimum. Finally, the lack of data preparation standards in the scientific community constitutes a dilemma for researchers who wish to share or to combine recordings from multiple datasets for large scale analysis.

5.2 Feature extraction paradigms in the processing stage

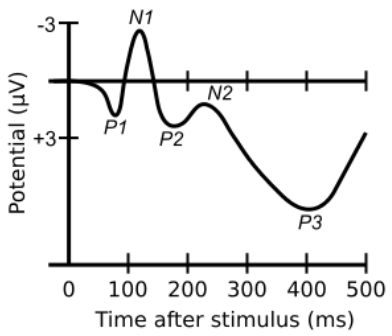
This section gives an overview of some of the most common feature extraction paradigms used in the field of MI-EEG classification. These tools often aim at highlighting robust patterns that appear in the EEG signal during the motor imagery phase.

Evoked versus induced potentials. Sensory stimulation is known to trigger two types of changes in the electrical activity of the cortex:

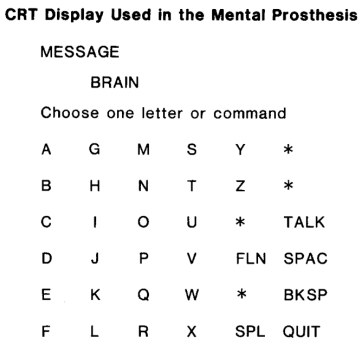
- time- and phase-locked responses (evoked potentials) that can be extracted using linear operations on the signal such as averaging over trials.
- time-locked but not phase-locked responses (induced potentials) that can be extracted using nonlinear methods such as spectral analysis.

5.2.1 Event-related potentials

Characterization. Evoked potentials [25] or event related potentials (ERPs) consist of small amplitude voltage fluctuations in the EEG signal and result from specific sensory, cognitive, or motor event stimuli. These fluctuations are however not directly observable due to the low signal to noise ratio but should be common to each trial. It is thus necessary to conduct many trials and average the results together in order to only keep the relevant waveform, called the ERP, while the embedded random brain activity is averaged out. ERP waveforms comprise a series of positive and negative voltage deflections, called components, each referred by a letter (N or P) indicating the polarity (negative or positive) and a number indicating the latency in milliseconds w.r.t. the stimulus (Fig. 5.1a).



(a) Post-stimulus ERP signal. [26]



(b) P300 speller. [27]

Figure 5.1: Event-related potentials. ERP waveform (a) obtained after averaging the EEG data over multiple trials and usecase of P300 component for BCI application (b).

Usecase. Some of these components, such as P300[1] components, are used extensively in neuroscience research because of their consistency among subjects regardless of the type of stimulus presented. One example of BCI based on P300 responses involves the use of a grid of characters displayed on a computer screen, with the subject focusing attention on the letter he wants to communicate [27] (Fig. 5.1b). This detection is achieved by repeatedly flashing rows and columns of the matrix grid. As the elements containing the chosen character are flashed, a P300 can be detected by the computer.

Limitations. Despite ERP component analysis being a reliable and powerful tool, it requires consistent, time-locked and phase-locked events which makes it harder to exploit for motor imagery where the stimulus is internal, often non consistent and difficult to precisely detect in time.

[1] Sometimes called P3.

5.2.2 Event-related synchronization and desynchronization

Characterization. Event-related synchronization (ERS) and desynchronization (ERD) represent the change in signal power occurring in a given band relative to a reference interval [28]. By convention, a power decrease corresponds to an ERD and a power increase to an ERS. These fluctuations comprise a different type of neural response that occurs after a stimulus, simultaneously to the ERP. The main differences being that the ERD/ERS is not phase-locked (but still time-locked) to the event and is highly frequency band-specific, allowing the use of time-frequency analysis while ERP can only rely on time-domain analysis.

Procedure. A typical procedure to compute the ERD/ERS includes the following steps (Fig. 5.2a):

- 1. Band-pass filter each trial in a frequency interval of interest.
- 2. Obtain the power signal by applying a squaring operation.
- 3. Average the power signal over all trials.
- 4. Apply temporal smoothing to the averaged power signal (e.g. running mean).
- 5. Compute the percentage values for ERD/ERS. The following equation describes the operation, where activity corresponds to the values of the smoothed averaged power signal during analysis and baseline is the mean signal value during rest

$$ERD/ERS(t) = \frac{activity(t) - baseline}{baseline} \times 100(\%) \tag{5.4}$$

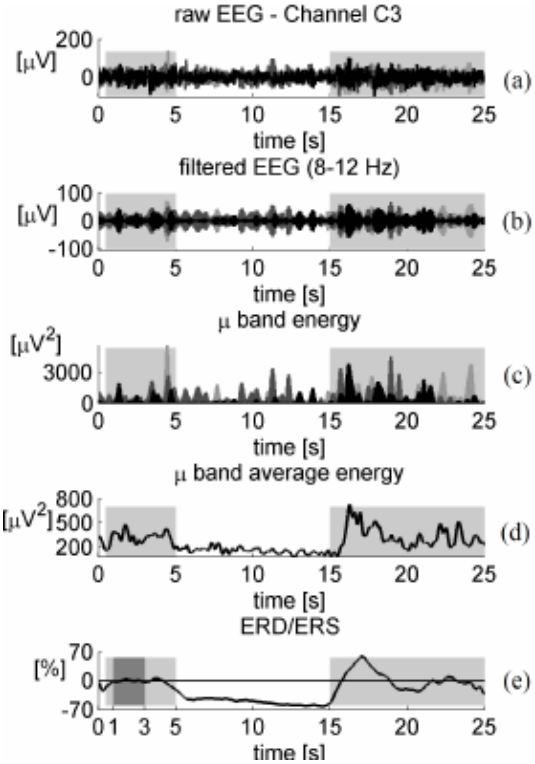
Usecase. Focusing on mu and beta bands, ERD and ERS are correlated to activated and deactivated cortical areas respectively during motor imagery. They can be used to study the dynamics of cortical activation patterns and visualized as temporal and spatial activity maps (Fig. 5.2b).

Limitations. Despite being an interesting visualizable and non-phase locked feature for time-frequency analysis, ERD/ERS remains sensitive to the choice of the baseline and might not be adequate for online BCI. Furthermore, the differentiation between the different motor imagery tasks is visually unpractical and requires the use of additional tools.

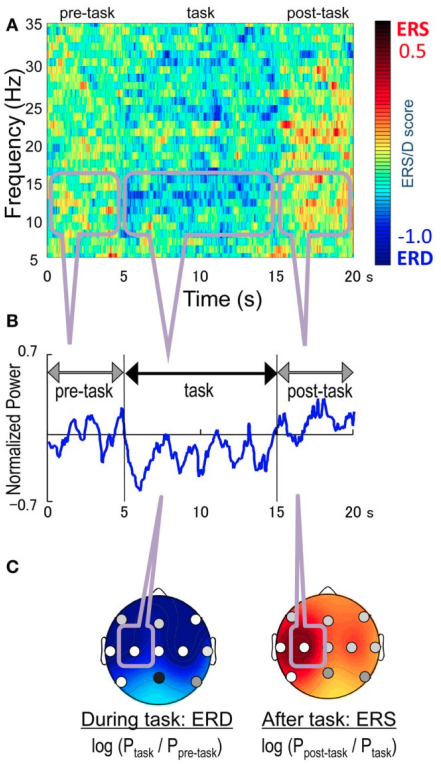
5.2.3 Covariance matrix.

Characterization. In the field of multivariate time-series classification, covariance matrices have been shown to be interesting features as they are able to capture the spectral variability of the signal.

Intuition. Intuitively, the covariance matrix generalizes the notion of variance to multiple dimensions. It is represented by a $N_c \times N_c$ matrix (N_c being the number of channels or electrodes) and summarizes the spatial distribution of the signal and noise power in the channel space (diagonal entries) as well as the spatial correlations between all EEG channels (non-diagonal entries).



(a) ERD/ERS computation procedure. [28]



(b) ERD/ERS as time-frequency (top) and spatial (bottom) score maps. [29]

Figure 5.2: Visualization of ERD/ERS features. Steps for computing ERD/ERS features from a raw EEG **(a)** and example of extracted features during right finger-tapping **(b)** represented as normalized power spectrum obtained by FFT, average power of EEG oscillation and topographical patterns.

Given the signal $\mathbf{X} \in \mathbb{R}^{N_c \times N_s}$, the covariance matrix of the signal can be estimated as

$$\mathbf{C} = \frac{\mathbf{X}\mathbf{X}^T}{N_s - 1} \in \mathbb{R}^{N_c \times N_c} \tag{5.5}$$

where N_c represents the number of EEG channels and N_s the number of time samples. There are, however, different possible estimators to estimate a covariance matrix [30].

A statistical analysis approach. In statistical analysis, the EEG recording of a given motor imagery task can be seen as a sample from a specific multivariate distribution. The goal of MI-EEG classification is thus to determine in which class distribution the query signal lies. In practice, MI-EEG recordings are classified using a minimum distance classifier according to their corresponding MI covariance matrices. The underlying training problem can thus be seen as a covariance matrix estimation in order to approximate the actual covariance matrix of each MI class. Typical procedures consist of averaging over trials the covariance matrices of the class-related signals using Euclidean or Riemannian approaches [31, 32].

Common covariance matrix operations. The notions of averaging and computing the distance between covariance matrices are often encountered in BCI [33, 34, 35,

36]. While the first operation (see equation [5.6]) is mostly used for estimating the covariance matrix associated to a particular MI task during feature extraction (training phase of a model), the second operation (see equation [5.7]) defines metrics for comparing how 'close' two covariance matrices are, which is often needed for a classifier to categorize the query EEG data (inference phase).

The arithmetic mean of a set $\{\mathbf{C}_i\}_{i=1}^{n}$ of n covariance matrices is

$$\mathfrak{U}(\mathbf{C}_1, \mathbf{C}_2, ..., \mathbf{C}_n) = \frac{1}{n}\sum_{i=1}^{n}\mathbf{C}_i \tag{5.6}$$

The Euclidean distance between two matrices $\mathbf{C}_1$ and $\mathbf{C}_2$ is

$$\delta_E(\mathbf{C}_1, \mathbf{C}_2) = \|\mathbf{C}_1 - \mathbf{C}_2\|_F \tag{5.7}$$

As a reminder, the Frobenius norm of a matrix $\mathbf{C} \in \mathbb{R}^{n\times m}$ is given by

$$\|\mathbf{C}\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}|c_{ij}|^2} = \sqrt{tr\left(\mathbf{C}\mathbf{C}^H\right)}$$

where $\mathbf{C}^H$ is the conjugate transpose of $\mathbf{C}$.

Advantages. Using covariance matrices as an embedding space is convenient for a least two reasons. First, this representation space of the brain signal allows for dimensionality reduction (by a factor of $\frac{N_s}{N_c}$) which is a critical aspect for real-time BCI. Second, it can leverage powerful mathematical tools from linear algebra, statistics or even differential geometry to help classify robust MI-EEG data (see Section 5.4).

Drawbacks. Covariance matrix operators are however very sensitive to noise that is inherent to EEG data collected using non-invasive setups. Therefore, such computations are often preceded by a filtering operation that requires additional tuning and might remove useful information from the EEG recording.

5.3 Common spatial pattern

Principle. The common spatial pattern (CSP) algorithm is a popular feature extraction technique for BCI systems. It is usually applied as a spatial filtering method that linearly combines information from the different electrodes, and yields features that are optimal for discriminating the different MI tasks [33]. The core idea of CSP is to diagonalize the average covariance matrices (over trials) of each class-related EEG signals.

Steps of the algorithm in a binary case. In a situation where two different MI classes need to be discriminated against each other, the CSP algorithm consists of the following steps:

1. ***Signal band-pass filtering.*** In a first step, the input signal has to be cleaned from its noisy components using a band-pass filter.

2. ***Covariance matrix estimation.*** The spatial covariance of each trial signal $\mathbf{X} \in \mathbb{R}^{N_c \times N_s}$ is estimated by

$$\mathbf{C} = \frac{\mathbf{X}\mathbf{X}^T}{tr(\mathbf{X}\mathbf{X}^T)} \tag{5.8}$$

This is a normalized version of equation [5.5] as the denominator here helps eliminating the magnitude variations in the EEG between sessions and subjects.

3. ***Trial-wise averaging.*** We then compute the arithmetic average covariance matrix over all trials of each class $j \in \{0, 1\}$

$$\overline{\mathbf{C}}_j = \frac{1}{N_j} \sum_{k=1}^{N_j} \mathbf{C}_j^{(k)} \tag{5.9}$$

where $\mathbf{C}_j^{(k)}$ corresponds to the covariance matrix of the k^{th} trial where class j occurred and N_j is the total number of trials of class j.

4. ***Solving Rayleigh quotient.*** The CSP can hence be formulated as a maximization problem whose goal is to find a set of optimal spatial filters $w^* \in \mathbb{R}^{N_c}$

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmax}}\, J(\boldsymbol{w}) \tag{5.10}$$

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^T \overline{\mathbf{C}}_1 \boldsymbol{w}}{\boldsymbol{w}^T \overline{\mathbf{C}}_2 \boldsymbol{w}} \tag{5.11}$$

This Rayleigh quotient is maximized for $\boldsymbol{w}$ by formulating the generalized eigenvalue decomposition problem (GEVD). We find the matrix of eigenvectors $\mathbf{U} \in \mathbb{R}^{N_c \times N_c}$ and the matrix of eigenvalues ordered by decreasing order $\mathbf{\Sigma} \in \mathbb{R}^{N_c \times N_c}$

$$\overline{\mathbf{C}}_1 \mathbf{U} = \overline{\mathbf{C}}_2 \mathbf{U} \mathbf{\Sigma} \tag{5.12}$$

For each MI label, we can extract m pairs[2] of spatial filter vectors $\boldsymbol{w}^*$ corresponding to the eigenvectors with the largest and smallest eigenvalues in magnitude (first and last columns of $\mathbf{U}$ respectively). These vectors are stacked horizontally in a matrix $\mathbf{W} \in \mathbb{R}^{N_c \times 2m}$, allowing to compute the CSP transform of a trial signal as follows

$$\mathbf{X}_{CSP} = \mathbf{W}^T \mathbf{X} \in \mathbb{R}^{2m \times N_s} \tag{5.13}$$

5. ***Computing CSP features.*** Finally, the CSP features are computed by normalization and log-variance calculation of the CSP-transformed signal, yielding a feature vector of $N_f = 2m$ components in a binary case

$$\mathbf{f} = \log\left(\frac{Var(\mathbf{X}_{CSP})}{\|Var(\mathbf{X}_{CSP})\|}\right) \in \mathbb{R}^{N_f} \tag{5.14}$$

where Var is the variance operated on the time axis and $\|.\|$ represents the Euclidean norm.

[2] In practice we often choose $m = 2$ or $m = 4$.

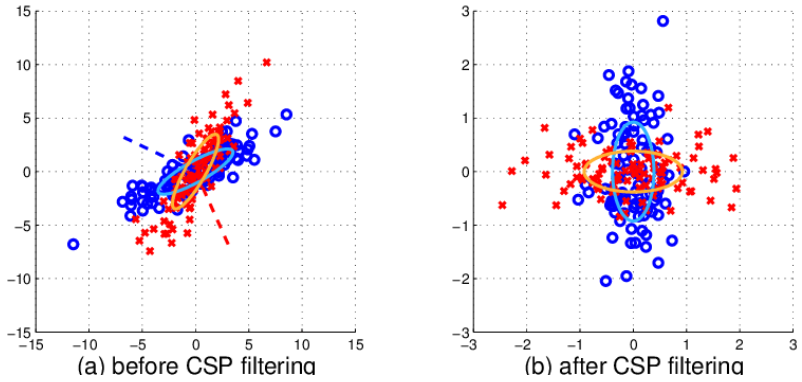


Figure 5.3: Effects of CSP transform on the signal. A 2D toy example consisting of two sets of samples marked by red crosses and blue circles are drawn from two Gaussian distributions. The distribution of samples from each set are shown before and after applying CSP. Ellipses represent the associated estimated covariances while dashed lines show the direction of CSP projections. [37]

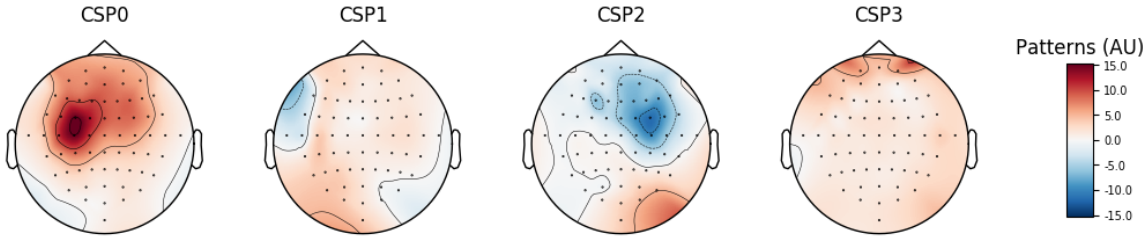


Figure 5.4: Visualization of CSP patterns. The patterns of 4 learned CSP filters are shown on a topographic map. These patterns explain how the data was generated from the neural sources. [38]

Multi-class case. The CSP algorithm can be adapted for a N_L-class problem by using either

- a one-versus-rest approach leading to $N_F = 2mN_L$ features. If we consider $N_L = 4$ classes and $m = 2$, we apply CSP on the combination of classes $\{1 \text{ vs } 2 \cup 3 \cup 4\}$, $\{2 \text{ vs } 1 \cup 3 \cup 4\}$, $\{3 \text{ vs } 1 \cup 2 \cup 4\}$, $\{4 \text{ vs } 1 \cup 2 \cup 3\}$ and obtain $N_F = 16$ feature vectors.
- a pair-wise approach leading to $N_F = mN_L(N_L - 1)$ features. Using the same example as before, CSP is applied on $\{1 \text{ vs } 2\}$, $\{1 \text{ vs } 3\}$, $\{1 \text{ vs } 4\}$, $\{2 \text{ vs } 3\}$, $\{2 \text{ vs } 4\}$, $\{3 \text{ vs } 4\}$ to obtain $N_F = 24$ feature vectors.

Intuition & interpretation. This algorithm has been shown to be highly successful in calculating spatial filters for revealing ERD/ERS [28]. Those spatial filters correspond indeed to the transformation to a latent space (alternative signal representation) where the estimated covariances of each MI class can be represented as orthogonal ellipses (Fig. 5.3). It is also possible to visualize the learned CSP filters as scalp topography maps (Fig. 5.4).

Limitations. Despite being a very popular approach for MI-EEG classification, the CSP algorithm carries multiple drawbacks. First, the quality of the learned spatial filters greatly depends on the covariance matrix estimator and the signal to noise ratio. Second, the choice of the frequency bands used during the filtering pre-processing step should be tuned accordingly to each subject and recording session.

5.4 Riemannian Geometry

Riemannian geometry (RG) is a concept originating from the domain of differential geometry that connects to the mathematical analysis of curves and surfaces. It generalizes Euclidean geometry to spaces that are smoothly curved and behave locally as Euclidean spaces.

Motivation. The concept of Riemannian geometry is particularly useful when it comes to studying covariance matrices. These matrices lie in the space of symmetric positive-definite (SPD) matrices which can be defined as a Riemannian manifold. However, most classical methods relying on covariance matrices treat them as if they were naturally lying in the Euclidean space, leading to approximation errors when using covariance matrix estimators. Therefore, RG offers tools that can be leveraged to manipulate covariance matrices in their native space, paving the way for better estimators. In practice, Riemannian approaches operate either on the projection of the data in a tangent space or directly on the SPD manifold.

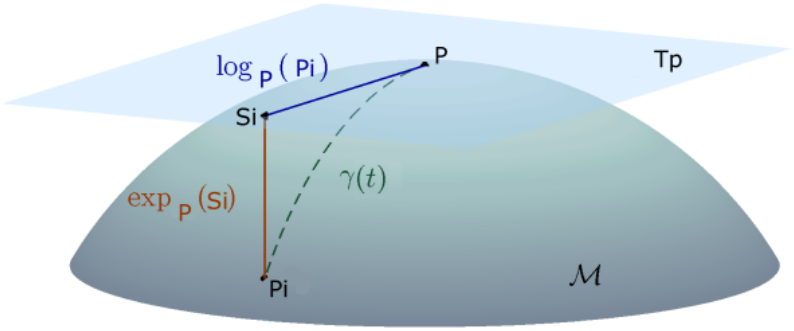


Figure 5.5: Tangent space in Riemannian manifold. In Riemannian geometry, a smoothly curved space (green hyperboloid) can be locally approximated by a flat Euclidean space (blue plane). A point $\mathbf{P}_i$ lying in the Riemannian manifold $\mathcal{M}$ can be projected in the tangent space Tp associated to the reference point P by using the logarithmic map equation [5.15]. The resulting point S_i can then be processed using conventional Euclidean metrics. Such metrics can also be adapted in a Riemannian sense if we want to operate on points in the Riemannian manifold without projecting them. Riemannian metrics often rely on the shortest geodesic paths $\gamma(t)$ joining the two points. [39]

Projection to the tangent space. A convenient method to keep using Euclidean space metrics is to project the covariance matrix on a local tangent space (Fig. 5.5), which forms an Euclidean space. We denote by $\mathcal{M}$ the set of all symmetric positive-definite matrices. The logarithmic map is used to project the matrix $\mathbf{P}_i \in \mathcal{M}$ from

the SPD space to the tangent space at a given reference point $\mathbf{P} \in \mathcal{M}$

$$\mathbf{S}_i = \text{Log}_P(\mathbf{P}_i) = \mathbf{P}^{1/2}\text{logm}\left(\mathbf{P}^{-1/2}\mathbf{P}_i\mathbf{P}^{-1/2}\right)\mathbf{P}^{1/2} \tag{5.15}$$

where logm() denotes the matrix logarithm function. The logarithm of a diagonalizable matrix $\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^{-1}$ is defined as $\text{logm}(\mathbf{A}) = \mathbf{V}\mathbf{D}'\mathbf{V}^{-1}$ where $\mathbf{D}'$ is a diagonal matrix of elements $d'_{i,i} = log(d_{i,i})$ [36].

Such a projection conserves the inner structure of the covariance matrix while making it possible to apply standard operations for covariance matrix as in equations [5.6] and [5.7]. The inverse operation is also possible using the exponential map to project a vector $\mathbf{S}$ lying in the tangent space at a given reference point $\mathbf{P}$ back to the SPD space

$$\mathbf{P}_i = \text{Exp}_P(\mathbf{S}_i) = \mathbf{P}^{1/2}\text{expm}\left(\mathbf{P}^{-1/2}\mathbf{S}_i\mathbf{P}^{-1/2}\right)\mathbf{P}^{1/2} \tag{5.16}$$

where expm() denotes the matrix exponential function. This operation is obtained the same way as for the logarithm of a matrix.

Choice of the reference point. P (or P_{ref}) is a free parameter where the tangent plane is computed. A common choice is to take the arithmetic or geometric mean of the complete set of covariance matrices (from the trials of the training set). Because we are working with a smooth space, a slight variation of the reference point should not affect too much the calculations in the tangent space.

Riemannian metrics. The direct use of metrics on SPD matrices in a Riemannian sense is also possible, yet less commonly used. The Riemannian distance between two SDP matrices $\mathbf{C}_1$ and $\mathbf{C}_2$ can be interpreted as the shortest path along their geodesic [39] and is defined as

$$\delta_R(\mathbf{C}_1, \mathbf{C}_2) = \left\|\text{logm}\left(\mathbf{C}_1^{-1}\mathbf{C}_2\right)\right\|_F = \left[\sum_{i=1}^{E} log^2\lambda_i\right] \tag{5.17}$$

where $\{\lambda_i\}_{i=1}^{E}$ are the real eigenvalues of $\mathbf{C}_1^{-1}\mathbf{C}_2$

The geometric mean of n SPD matrices can be computed using iterative algorithms, as there is no closed-form solution [39], and is defined as

$$\mathfrak{B}(\mathbf{C}_1, \mathbf{C}_2, ..., \mathbf{C}_n) = \underset{\mathbf{C}}{\text{argmin}} \sum_{i=1}^{n} \delta_R(\mathbf{C}, \mathbf{C}_i)^2 \tag{5.18}$$

Advantages. Riemannian geometry makes it possible to build simpler and more accurate BCI approaches, involving fewer stages than traditional methods and thus less parameter tuning. Moreover, the logarithmic nature of Riemannian operations has a beneficial impact regarding the robustness, especially when dealing with noisy input signals. Finally, RG can be applied successfully to the different BCI paradigms (MI, ERPs, etc...) and is invariant to linear transformations of the input signal, making it a powerful and robust tool for BCI, with good generalization capabilities.

Limitations. Despite relying on better covariance matrix estimates, Riemannian geometry is exposed to higher risks of numerical instabilities. Equation [5.17] indicates indeed that the computation of Riemannian distance between two SPD matrices involves adding squared logarithms of real eigenvalues. Such operations become numerically unstable and ill-conditioned, as these eigenvalues tend towards zero

when the number of electrodes increases or the temporal window size for estimating $\mathbf{C}_1^{-1}\mathbf{C}_2$ decreases. Furthermore, RG approaches are often lacking benchmarking in the domain of BCI and comparison to more traditional methods.

5.5 Neural networks

Artificial neural networks (ANN) compose an essential part of modern machine learning approaches gathered under the field of deep learning (see Chapter 2). These brain-inspired computing systems have gained popularity in the past few decades as they proved to be excellent tools for finding patterns that would be too complex or numerous for humans to extract.

Motivations. One of the major incentives for using neural networks is their ability to automatically "learn" relevant features without any prior knowledge about the task (outside the input dataset itself). This property can notably alleviate the feature extraction phase when dealing with designing a decoding model for BCI. Classic machine learning approaches require indeed the design of hand-crafted discriminative features, and thus, in-depth knowledge of the specific domain. Therefore, deep learning paves the way for new approaches to complex problems by relying on automatic feature extraction.

Architectures. Neural networks come in various types of architectures, among which two popular: convolutional neural networks (CNN) and recurrent neural networks (RNN). The former is based on the use of convolutional layers where each neuron has access to its local neighborhood information (either temporally or spatially) from neurons of the previous layer. The latter is able to exhibit the temporal dynamic behavior of the input data by relying on memory cells that can process the information sequentially while retaining their internal state. The next two paragraphs take a deeper look into CNN and RNN architectures in the context of feature extraction for MI-EEG classification.

Convolutional neural networks. A typical CNN architecture is composed of a succession of convolutional and pooling layers before ending with one or multiple fully connected layers (Fig. 5.6). Such an architecture is able to extract features hierarchically and classify the input signal in one forward pass. During training, the first few layers tend to learn low-level features from the MI-EEG data such as simple signal processing operations, whereas the deeper layers are able to grasp higher-level features such as specific rhythms patterns that could correspond to mu or beta frequency bands components (Fig. 5.7).

Three types of layers can be encountered in a CNN:

- **Convolutional layers.** A convolutional layer consists of a set of learnable kernels, also called filters, convolved with the input volume across its height and width axis (corresponding to the spatial and temporal dimensions for the EEG data). This operation is performed by sliding the kernel over the input volume and computing the dot product for each position (Fig. 5.8). The resulting volume is usually passed through a non-linearity, called activation function, and leads to a set of stacked 2D activity maps (one for each kernel) that should produce a higher response where relevant features are detected (Fig. 5.7). The convolution operation enforces local connectivity between nodes of adjacent layers: each node being connected to a partial region of the input volume.

Figure 5.6: CNN architecture. VGG-16 is a popular architecture used in computer vision composed of convolutional (conv), pooling (pool) and fully connected (fc) layers. The use of two or three successive convolutional layers (here of kernel size 3 by 3) followed by a pooling layer is typical in CNN architectures. Notice also the increasing number of kernels in "conv" layers which corresponds to an increasing amount of extracted features as we go deeper in the network. [40]

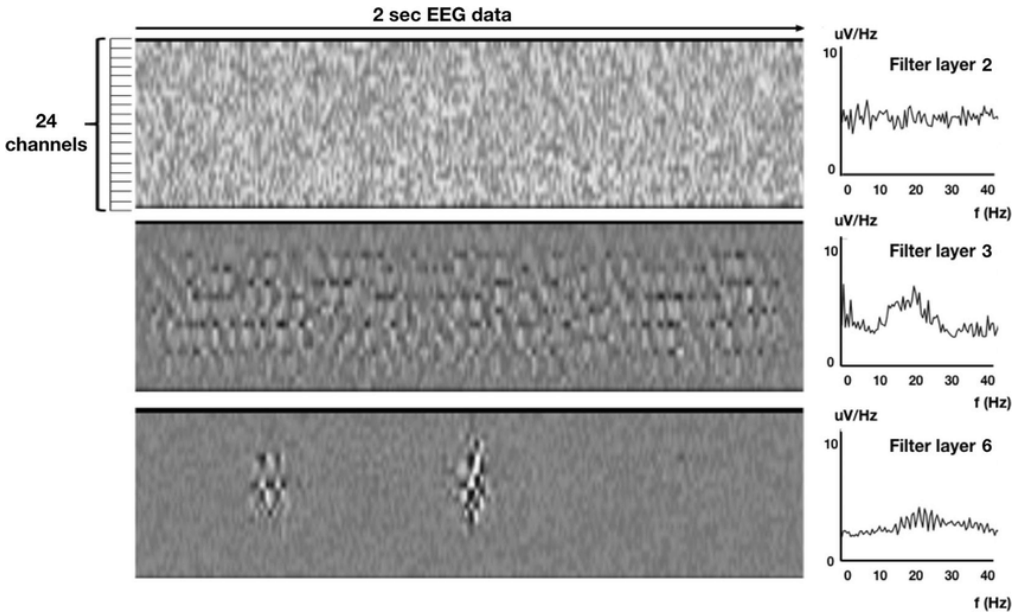


Figure 5.7: Visualization of CNN activations. The activations of the trained CNN can display the position of relevant features in time and space. Power spectral density analysis (right graphs) tends to show that the network is progressively learning how to filter the signal in order to extract specific features in the beta frequency band. [41]

These connections can be local in space (between channels) or in time (between samples) and characterize the receptive field of a node, i.e. the amount of information it can access from the input. Convolutional layers also tend to use a weight sharing scheme to control the number of parameters and rely on the assumption that if a patch feature (or kernel operation) is relevant to compute at some time or space position, it should also be useful to compute at other positions. Thus, the weights of a kernel are the same when performing the convolution operation at a given depth.

The hyperparameters of a convolutional layer include

– N_k, the number of learnable kernels, which define the depth of the output volume.

- K, the kernel size, that determines the receptive field of each node.
- S, the stride, which is the sliding increment used during convolution and is useful to control the amount of overlapping between the receptive fields.
- P the padding, which consists of bordering the input volume with zeros and is useful to control the size of the output volume.

The spatio-temporal size $C_{out} \times T_{out}$ of the output volume can be computed as a function of the input volume size $C_{in} \times T_{in}$ as

$$C_{out} = \frac{C_{in} - K - 2P}{S} \tag{5.19}$$

$$T_{out} = \frac{T_{in} - K - 2P}{S} \tag{5.20}$$

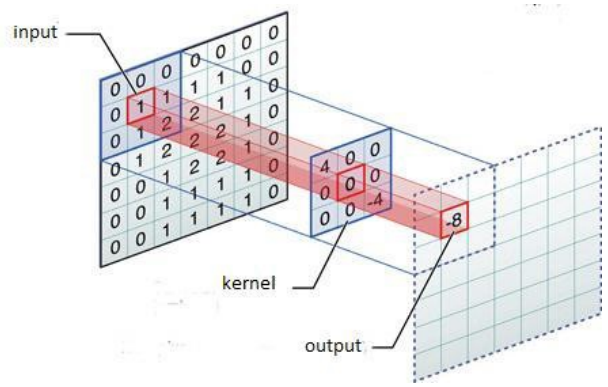


Figure 5.8: Convolution operation. Convolution using a kernel of size 3 by 3 on a single depth slice. The output value (in red) is obtained by performing the dot product between the kernel and overlapping input elements. The complete output slice is computed by repeating this process while sliding the same kernel over different positions of the input. In practice, one convolutional layer exploit multiple learnable kernels and outputs a 3D tensor where the depth corresponds to the different kernels. [42]

- **Pooling layers.** Pooling is another type of layer that is often found between successive convolutional layers in CNN architectures. It has a down-sampling effect by performing a non-linear function on the input volume while leaving the depth dimension intact. The most common operation used for pooling layers is max-pooling. It consists of partitioning the input volume into a set of non-overlapping squares and outputting the maximum value for each sub-region (Fig. 5.9). The main goal of pooling layers is to progressively reduce the spatial and temporal size of the signal representations as we go deeper in the network. This enables the control of the number of parameters, computation time and memory footprint. Pooling layers also provide some translation invariance to the network. The hyperparameters of a pooling layer include the kernel size and the stride, similar to the one encountered in a convolutional layer.
- **Dense layers.** Fully connected - or dense - layers connect every neuron from one layer to every neuron of the next layer. They can be seen as the classifier

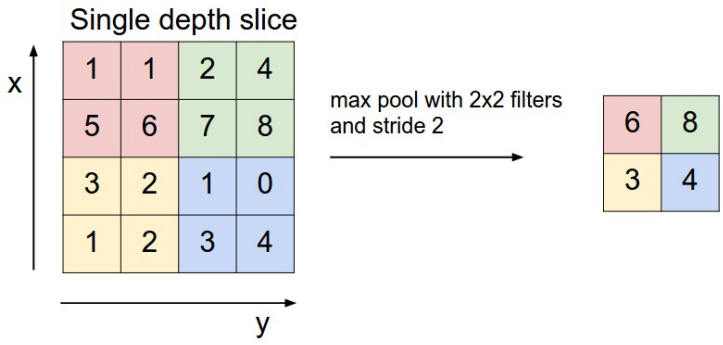


Figure 5.9: Pooling operation. Max-pooling operation of kernel size 2 by 2 and stride 2 for a single depth slice of input size 4 by 4. Only one forth of the activations are kept, meaning that 75% of the activations are discarded, which leads to a representation of output size 2 by 2. [14]

stage of a CNN as they learn non-linear combinations of all the high-level features extracted by the convolutional layers in order to make the final decision. The last layer of a CNN classifier is often a fully connected layer comprising the same number of nodes as classes and uses a softmax activation in order to output interpretable class-probability vectors (see equation [2.2]).

Recurrent neural networks. A typical RNN architecture consists of a memory cell processing the information sequentially in time in a loop. Such a network can be "unfolded" in multiple copies of the same cell, each looking at a different slice of the input sequence and passing information to the successor cell (Fig. 5.10).

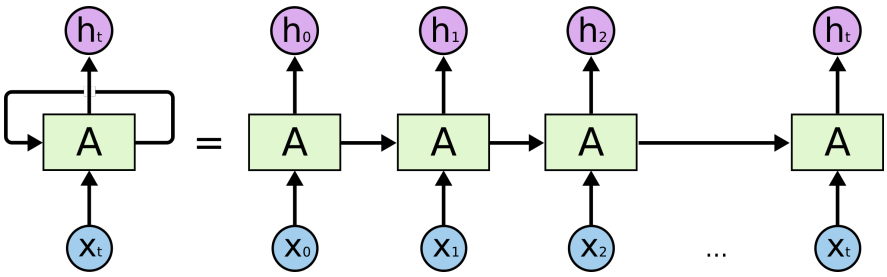


Figure 5.10: RNN architecture. A memory cell A processes sequentially some input data x_t and outputs a value h_t while a loop allows information to be passed from one step to the next. The unfolded architecture is composed of copies of the same cell, each looking at a time slice of the sequential input and sharing its state with the next cell. [43]

One of the appeals of recurrent neural networks is the idea that they are able to connect information appearing at different steps of the sequential process. The gap between dependant features can sometimes become very large but should, in theory, be handled by RNNs. However, in practice, these neural networks are not able to learn such “long-term dependencies" due to vanishing gradients [44].

Specific memory cells such as long short-term memory (LSTM) cells have been designed to palliate this weakness of standard RNNs [45] and are now widely used to treat a large variety of problems. These cells are composed of multiple regulated structures, called gates, that have the ability to filter information (Fig. 5.11). Each

LSTM cell is composed of three tunable gates that apply sigmoid and pointwise multiplication operations on the previous hidden state h_{t-1} and current input x_t

- **Forget gate:** $f_t = \sigma(\mathbf{W}_f[h_{t-1}, x_t] + b_f)$
- **Input gate:** $i_t = \sigma(\mathbf{W}_i \cdot [h_{t-1}, x_t] + b_i)$
- **Output gate:** $o_t = \sigma(\mathbf{W}_o \cdot [h_{t-1}, x_t] + b_o)$

where **W** and **b** are learnable parameters and σ represents the sigmoid function.

Finally, the updated cell state C_t and hidden state h_t (output of LSTM cell) can be computed as

- **Hidden state:** $h_t = o_t * tanh(C_t)$
- **Cell state:** $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$

where $\tilde{C}_t = tanh(\mathbf{W}_c \cdot [h_{t-1}, x_t] + b_c)$

In this paper, we do not explore the use of RNN architectures for BCI classification [46, 47, 48] as they seem to be lagging behind CNN architectures in terms of performance and often require a training procedure that is difficult to optimize.

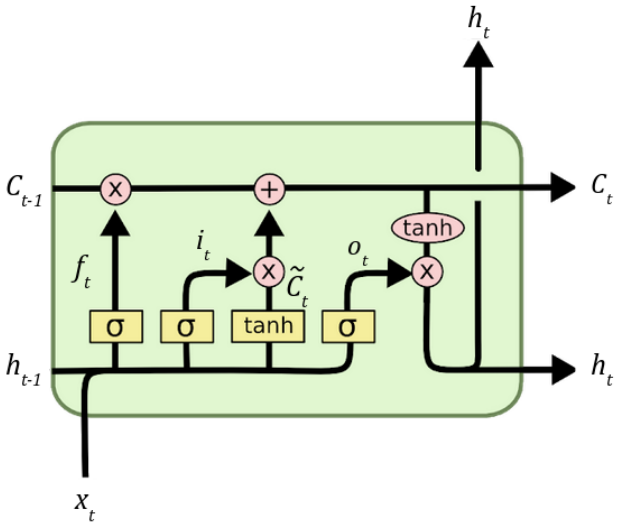


Figure 5.11: LSTM cell. The structure of this cell is composed of tunable gates that select which part of the input information $\mathbf{x}_t$ to keep, while considering the cell state $\mathbf{C}_{t-1}$ and hidden state $\mathbf{h}_{t-1}$ of the previous cell. [43]

Advantages. The motivation for using neural networks is not only limited to their ability to perform automatic and hierarchical feature discovery. The iterative nature of their training process makes neural networks easily scalable to large datasets. This also renders the use of transfer learning strategies possible, in which a model can be pre-trained on EEG data of other sessions and/or subjects and then fine-tuned on data from a new subject. Furthermore, ANNs are well suited for end-to-end learning, relying on the joint optimization of both feature extraction and classifier stages, which often leads to a performance boost compared to traditional machine learning approaches. Finally, the fast inference of ANN models makes them well suited for real-time BCI applications.

Limitations. Despite the general hype surrounding deep learning, neural networks are far from perfect. First, they often require a large amount of training data and may take longer to train than simpler models. Second, deep neural networks can be notoriously difficult to interpret as we often lack the explanations of why some features are extracted. Last but not least, deep learning models currently lack the robustness of traditional models as they may output false predictions with high confidence. It is indeed possible to design input samples, called adversarial examples, to trick the model into misclassifying them just by adding targeted noise [49]. This poses, of course, serious ethical and safety concerns especially if such a technology would end up being used for medical or industrial purposes.

5.6 Classification stage

This section introduces two common algorithms used in the classification stage of many traditional BCI pipelines.

5.6.1 Support-vector machine.

Principle. Support-vector machine (SVM) is a popular machine learning algorithm that can be used for supervised classification problems. It usually receives multi-dimensional features $\mathbf{x} \in \mathbb{R}^D$ as input, that are linearly separable, but can also deal with non-linearly separable data using the kernel trick. In a binary case, the linear SVM classifier f can be expressed as

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b \tag{5.21}$$

where $\mathbf{w} \in \mathbb{R}^D$ is the weight vector and b the bias scalar. By convention, the input is categorized as 'negative' label when the classifier outputs a negative value, and as 'positive' label otherwise.

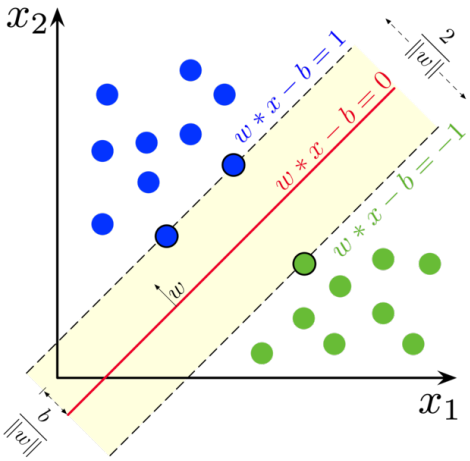


Figure 5.12: SVM margins in simple binary case. The hyperplane (in red) and the margins (in dashed line) split the data space in two categories. Data points falling above the hyperplane are considered as blue labels while the others are green labels. Data points falling on the margin (black circle) are called support vectors and define by themselves the hyperplane equation. [50]

Algorithm. The SVM approach consists of finding the maximum-margin hyperplane that differentiates the data points of each class (Fig. 5.12). The margin associated to a data point $\mathbf{x_i}$ of label y_i can be computed as $\frac{y_i(\mathbf{w}^T\mathbf{x}_i+b)}{\|\mathbf{w}\|}$. For a classifier to be correctly trained, the numerator should always be greater or equal to one. The goal is thus to solve the optimization problem

$$\underset{w,b}{\text{argmax}} \frac{1}{\|\mathbf{w}\|} \min_k \left[y_k * (\mathbf{w}^T\mathbf{x}_k + b) \right] \tag{5.22}$$

This can also be formulated in a more convenient form as

$$\underset{w,b}{\text{argmin}} \frac{\|\mathbf{w}\|^2}{2} \quad \text{s.t.} \quad \forall k,\ y_k(\mathbf{w}^T\mathbf{x}_k + b) \geq 1 \tag{5.23}$$

Such a quadratic optimization problem is solved using the Lagrange multiplier theorem [51]. This leads to the dual formulation which aims to find the optimal values α_i^* such that

$$\max_{\alpha \leq C} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \quad \text{s.t.} \quad \alpha_k \geq 0 \text{ and } \sum_{i=1}^{N} \alpha_i y_i = 0 \tag{5.24}$$

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

$$f(x) = \sum_{i=1}^{N} \alpha_i^* y_i \mathbf{x}^T \mathbf{x}_i + b \tag{5.25}$$

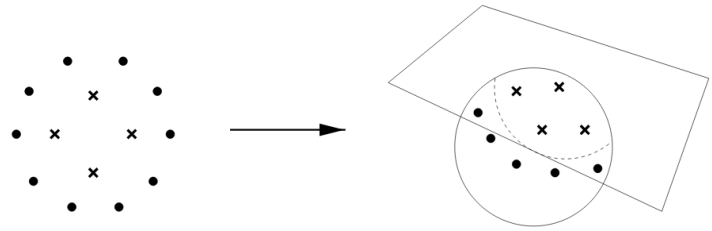


Figure 5.13: SVM for non-linear case. The kernel trick consists in implicitly mapping the input data into a higher dimensional space where it is linearly separable. In this simple binary case, we map 2D data points into a 3D space where a hyperplane is able to categorize them successfully. [52]

Non-linear case. SVMs are also capable of handling non-linearly separable input data using what is called the kernel trick. This consists of implicitly mapping the input data to a high-dimensional feature space and constructing the hyperplane in that space (Fig. 5.13). We are now searching for the hyperplane f defined as

$$f(x) = \mathbf{w}^T \phi(\mathbf{x}) + b \tag{5.26}$$

where ϕ is a non-linear transformation. The kernel trick allows for the use of such

mapping without explicitly defining it. Instead, we only need to define a kernel function K which satisfies $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$. Finally, the solution remains very similar to the linear case

$$f(x) = \sum_{i=1}^{N} \alpha_i^* y_i K(\mathbf{x}, \mathbf{x}_i) + b \tag{5.27}$$

A widely used kernel function is the radial basis function (RBF) kernel defined as

$$K(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|^2}{2\sigma^2}\right) \tag{5.28}$$

where σ is a free parameter.

Advantages. SVMs are quite effective in high-dimensional spaces even in the case where the input dimension is greater than the number of input examples. They also have the advantage of being memory efficient, since the hyperplane can be computed using only the training points lying on the margins (support vectors). Finally, SVMs are versatile, as the kernel trick makes it possible to tackle non-linear classification problems using pre-defined kernel functions

Disadvantages. The main drawback when using SVMs is the lack of probability estimates when categorizing a query input. Furthermore, SVMs can be quite sensitive to over-fitting and require careful tuning of the penalty parameter.

5.6.2 Linear discriminant analysis

Principle. Linear discriminant analysis (LDA) is another popular algorithm that is often used as a dimensionality reduction technique or linear classifier in the context of BCI classification. The goal of this algorithm is to find the best linear combination of the input data by projecting it onto a lower-dimensional space with good class-separability (Fig. 5.14).

Algorithm. Given a two-class problem, we make the assumption that the probability density function of both classes is normally distributed with mean and covariance parameters $(\mathbf{m}_1, \mathbf{S}_1)$ and $(\mathbf{m}_2, \mathbf{S}_2)$. The probability of a input vector $x \in \mathbb{R}^d$ to belong to class k is expressed as

$$p(\mathbf{x}|y = k) = \frac{1}{(2\pi)^{d/2}|\mathbf{S}_k|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{m_k})^T \mathbf{S}_k^{-1}(\mathbf{x} - \mathbf{m_k})\right) \tag{5.29}$$

In order to categorize $\mathbf{x}$, the classifier needs to look at the sign of $\log\left(\frac{p(y=1|\mathbf{x})}{p(y=2|\mathbf{x})}\right)$. Using the homoscedasticity assumption ($\mathbf{S}_1 = \mathbf{S}_2 = \mathbf{S}$) and the Bayes rule $p(y = k|\mathbf{x}) = \frac{p(\mathbf{x}|y=k)p(y=k)}{p(\mathbf{x})}$, the discrimination rule becomes

$$y = \begin{cases} 1 & \text{if } z \geq \hat{z} \\ 2 & \text{if otherwise} \end{cases} \tag{5.30}$$

$$\text{where } z = (\mathbf{m}_1 - \mathbf{m}_2)^T \mathbf{S}^{-1}\mathbf{x} \quad \text{and} \quad \hat{z} = \frac{1}{2}\left(\mathbf{m}_1^T \mathbf{S}^{-1}\mathbf{m}_1 - \mathbf{m}_2^T \mathbf{S}^{-1}\mathbf{m}_2\right)$$

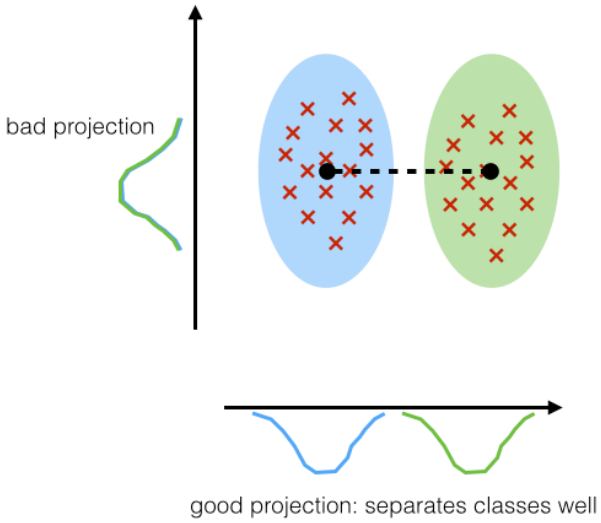


Figure 5.14: LDA principle. Given a set of input data points (red crosses), linear discriminant analysis aims at finding the best projection vector such that the projected distribution associated to each class (blue and green curves) can be easily separated. [53]

Multi-class case. In a general c-class problem, the previous algorithm can still be applied using a "one-versus-rest" or "pair-wise" approach.

Dimensionality reduction. LDA can also be used for dimensional reduction before later classification. We define the input data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ containing n vectors of dimension d and the associated label vector $\mathbf{y} \in \mathbb{R}^n$. D_i designates the set of n_i input vectors of label i. The procedure works as follows

- 1. Compute the class-mean vector associated to each label i

$$\mathbf{m}_i = \frac{1}{n_i} \sum_{\mathbf{x} \in D_i} \mathbf{x} \tag{5.31}$$

- 2. Compute the within-class [5.32] and in-between-class [5.33] scatter matrices

$$\begin{aligned} \mathbf{S}_W &= \sum_{i=1}^{2} \mathbf{S}_i \\ \text{where } \mathbf{S}_i &= \sum_{\mathbf{x} \in D_i} (\mathbf{x} - \mathbf{m}_i)(\mathbf{x} - \mathbf{m}_i)^T \end{aligned} \tag{5.32}$$

$$\mathbf{S}_B = \sum_{i=1}^{2} n_i (\mathbf{m}_i - \mathbf{m})(\mathbf{m}_i - \mathbf{m})^T \tag{5.33}$$

  where m is the mean of all n input vectors.

- 3. Compute the eigenvectors and eigenvalues of the matrix $\mathbf{S}_W^{-1}\mathbf{S}_B$.
- 4. Select the k eigenvectors associated to the k largest eigenvalues and stack them into an eigenvector matrix $\mathbf{W} \in \mathbb{R}^{d \times k}$.
- 5. Project the input data onto the new subspace

$$\mathbf{Z} = \mathbf{X}\mathbf{W} \in \mathbb{R}^{n \times k} \tag{5.34}$$

Advantages. LDA enables a compression of the input data while maintaining the class-discriminatory information. This is very useful to speed up learning algorithms and reduce the storage space required during classification.

Disadvantages. However, this algorithm assumes strong assumptions regarding the input data that rarely hold, especially for EEG signals. The algorithm expects indeed the data to be normally distributed, with identical covariance matrices for every class and statistically independent features.

6 Methodology

This chapter introduces the methodology adopted throughout our experiments. It first gives an overview of our BCI pipeline and project contributions, before describing the different models to be compared. Finally, we present our training and testing procedures along with our benchmarking results.

6.1 Pipeline overview and contributions

The main goal of this thesis is to provide a full BCI pipeline that could be employed to design, train and benchmark the decoding model for the Cybathlon competition. This pipeline can be split into complementary functional blocks (Fig. 6.1):

- *"Data formatting"* deals with the diverse MI-EEG dataset formats and proposes functions for importing continuous EEG data from different popular formats (such as .npz, .gdf, .vhdr and .mat), splitting the continuous labelled signals into trials and saving them into standardized EEG data $\mathbf{X} \in \mathbb{R}^{n_{trials} \times n_{channels} \times n_{samples}}$ and label arrays $\mathbf{y} \in \mathbb{R}^{n_{trials}}$ in a .npz format.
- *"Data pre-processing"* proposes functions to load and split formatted EEG data as train, validation and test sets while applying selected pre-processing operations on the signal such as the ones defined in Section 5.1.
- The *"feature extraction"* block can be seen as a bank of both traditional and modern feature extractions approaches (see Sections 5.2, 6.3 and 6.4) that can be plugged into our pipeline.
- *"Model training"* gathers tools and strategies to train and tune our models (see Section 6.5).
- *"Online inference"* consists of a visual interface to observe the pre-processed query signal and the model inference. This interface can also play the game in real-time to simulate real conditions of the competition.
- *"Visualization tools"* gathers functions to display training and validation metrics during offline training as well as different signal representations in time, frequency and time-frequency domains.

This block fashion approach allows for easy integration and upgradability of the pipeline, all functions being written in Python 3 and attached to a complete documentation through comments and example codes.

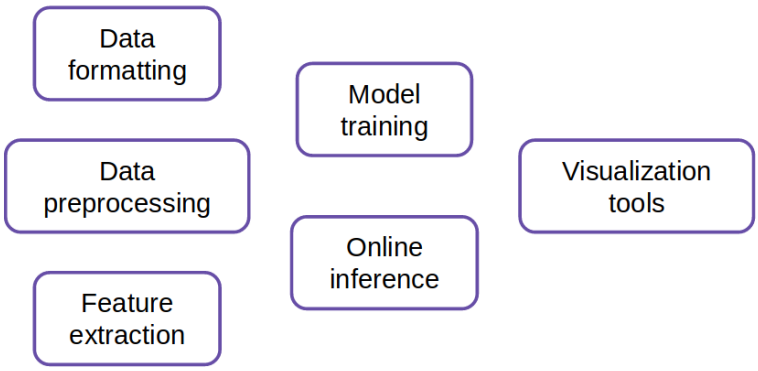


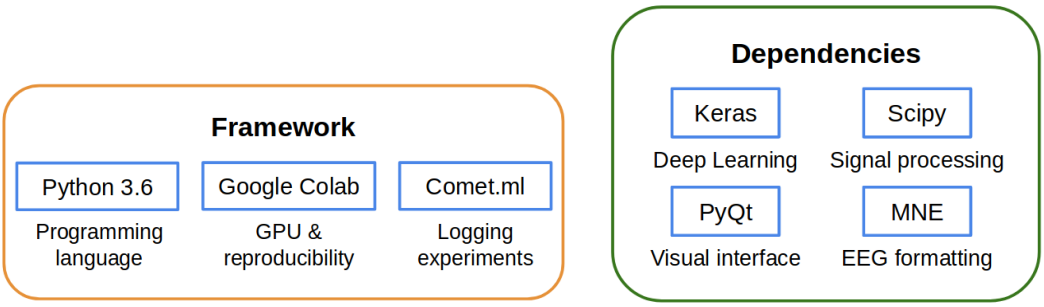
Figure 6.1: Pipeline blocks. Each block contains a set of functions that can be plugged together to test different BCI pipelines.

6.2 Framework

Regarding the additional software tools, we use Google Colab[1] and Comet.ml[2] in order to have reproducible and easy-to-run experiments (Fig. 6.2a). The former is a cloud-based Jupyter notebook that allows for code sharing and free access to GPU whereas the latter is a cloud platform that keeps track of the different experiments along with their metrics, hyperparameters and visual plots. All experiments and benchmarking regarding the baseline models were conducted on a Intel(R) Xeon(R) CPU @ 2.30GHz with 12GB RAM, whereas neural networks were trained using a Tesla K80 with 12GB of RAM.

We aim at employing a minimal amount of dependencies[3] to run the code (Fig. 6.2b):

- MNE for formatting the EEG signal.
- Scipy for signal processing functions in the back-end.
- Keras as a deep learning framework.
- PyQt for the online inference visual interface.



(a) Software tools. **(b)** Dependencies.

Figure 6.2: Framework. Our pipeline is based on a simple framework of software tools and easy to install dependencies.

[1] https://colab.research.google.com/
[2] https://www.comet.ml/
[3] We also rely on native Anaconda packages such as numpy and matplotlib.

6.3 Baseline approaches

This section introduces two "traditional" methods that we will use as baselines during our benchmarking. The first one, filter bank CSP (FBCSP) [34], has been adopted as a reference in the field of MI-EEG classification for the past few years and will be useful to determine how difficult/clean our pilot dataset is for a popular benchmark approach. The second one is a Riemannian covariance method based on multi-scale spectral features [35] and offers an interesting performance boost over the first approach on the BCIC IV datasets. Despite not being a popular method in motor imagery benchmarks at the moment, we expect this baseline to give us a complementary analysis regarding the performance of state-of-the-art models on different datasets.

6.3.1 Filter bank common spatial pattern

Filter bank common spatial pattern (FBCSP) is an approach that enhances the popular CSP method introduced in Section 5.3, whose effectiveness greatly depends on the subject-specific frequency bands. FBCSP helps palliate this issue by selecting autonomously discriminative features in each frequency domain.

Principle. This method builds on top of the CSP algorithm by adding a filter bank before the spatial filtering. This filter bank contains multiple band-pass filters that receive the input signal in parallel. Each resulting filtered signal is then fed to a specific CSP block (Fig. 6.3). After the spatial filtering stage, a feature selection algorithm keeps the most discriminative CSP features which are then classified as a motor imagery label.

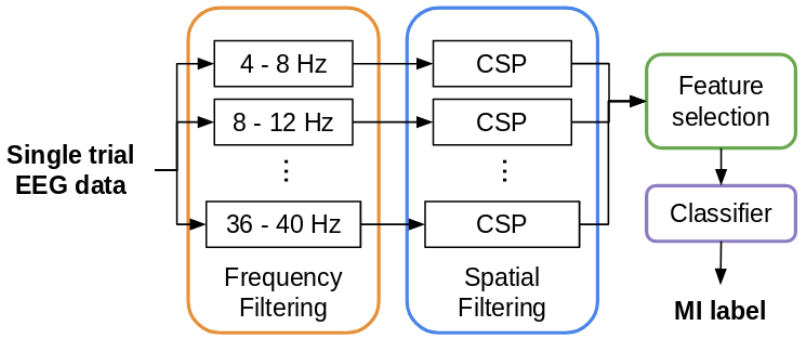


Figure 6.3: Architecture of the FBCSP algorithm. A first stage filters the signal using different frequency bands in parallel. Each of the frequency-filtered signals are given to a CSP block. Finally the most discriminative CSP feature vectors are selected and fed to a classifier.

Filter bank. The initial configuration used for the filter bank relies on nine causal and zero-phase Chebyshev Type II band-pass filters ranging from 4 to 40 Hz with bandwidths of 4 Hz. Each filter receives the same EEG data and outputs the filtered signal to one CSP block per frequency band.

CSP block. The same CSP algorithm as described in Section 5.3 is used here.

Feature selection. FBCSP employs a mutual information-based feature selection that selects a subset of k CSP features to maximize classification accuracy. The mutual information based best individual feature (MIBIF) algorithm is often used and consists of the following steps:

1. Consider a set of d feature vectors $F = \{\mathbf{f}_1, \mathbf{f}_2, ... \mathbf{f}_d\}$ and initialize the set of selected features $S = \varnothing$.
2. Compute the mutual information $I(\mathbf{f}_i\,; y)$ (see equation [6.1]) between each feature $\mathbf{f}_i \in F$ (continuous random variable) and the class label y (discrete random variable).
3. Select the best k [4] features as follows:
   - Select the feature $\mathbf{f}_j \in F$ that maximizes $I(\mathbf{f}_j\,; y)$
   - Update $F = F \setminus \{\mathbf{f}_j\}$ and $S = S \cup \{\mathbf{f}_j\}$
   - Repeat until $|S| = k$

The mutual information between the continuous random variable X and the discrete random variable Ω is expressed as

$$I(\mathbf{X}; \Omega) = H(\Omega) - H(\Omega|\mathbf{X}) \tag{6.1}$$

where the entropy of Ω is

$$H(\Omega) = -\sum_{\omega=1}^{|\Omega|} p(\omega) \log_2[p(\omega)] \tag{6.2}$$

and the conditional entropy between Ω and $\mathbf{X}$ is

$$H(\Omega|\mathbf{X}) = -\int_{x \in \mathbf{X}} \sum_{\omega=1}^{|\Omega|} p(\omega|x) \log_2[p(\omega|x)] \mathrm{d}x \tag{6.3}$$

Classifier. The selected CSP features (and their associated pair) are fed to a SVM that outputs the inferred motor imagery task (see Section 5.6.1).

Advantages. The FBCSP algorithm has the main advantage of requiring a minimal amount of tuning while being adaptive to the subject-specific frequency bands.

Limitations. However, the computation of mutual information can be tricky as the feature vectors lie in a continuous space requiring the use of density estimation techniques such as Parzen windows. Additionally, FBCSP is sensitive to outliers as it involves estimations of covariance matrices from noisy EEG measurements.

[4] A value of k=4 is often used.

6.3.2 Multi-scale Riemannian covariance

The second baseline approach also relies on the filter bank idea proposed in Section 6.3.1 but combines it with a Riemannian covariance approach [35]. The use of Riemannian geometry, introduced in Section 5.4, allows for better covariance matrix estimates and eliminates the need for a feature selection block.

Principles. Just like FBCSP, this model is composed of multiple stages:

- *Multi-scale frequency filtering*: processes the input signal in parallel through multiple band-pass filters with various bandwidths.
- *Riemannian feature extraction*: computes the covariance matrix for each filtered signal using a Riemannian approach.
- *Classification*: feeds the concatenated feature vectors in a classifier to output the corresponding motor imagery task label.

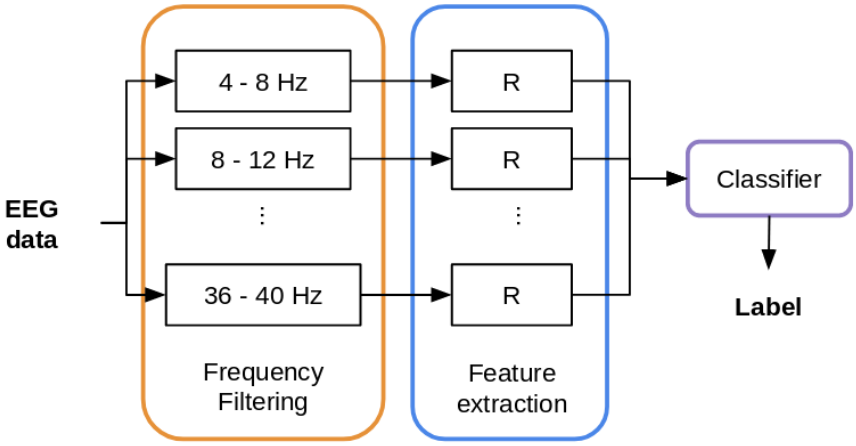


Figure 6.4: Riemannian covariance model architecture. A filter bank process the input EEG data using multiple frequency intervals in parallel. Each filtered signal is then passed to a specific Riemannian block (detailed in Figure 6.5) to extract covariance features using Riemannian geometry which are vectorized, concatenated and fed to a classifier.

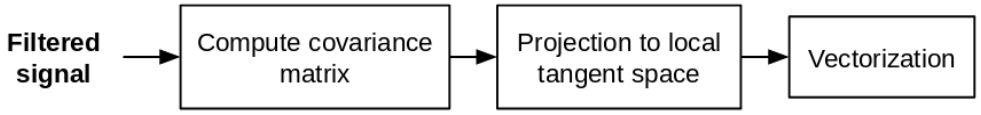


Figure 6.5: Riemannian block. First, the covariance matrix of the filtered signal is computed as in [5.5]. This matrix is then projected to a local tangent space of the Riemannian manifold using as reference point the average covariance matrices of the filtered signals (trial-wise). Finally, the resulting matrix is vectorized as in [6.4].

Filter bank. The configuration of this filter bank stands out from the one in FBCSP and employs 43 second-order Butterworth filters ranging from 4 to 40 Hz with various bandwidths from 2 to 32 Hz (Fig. 6.4). Each filter receives the same EEG data and outputs the filtered signal to one Riemannian block (R) per frequency band.

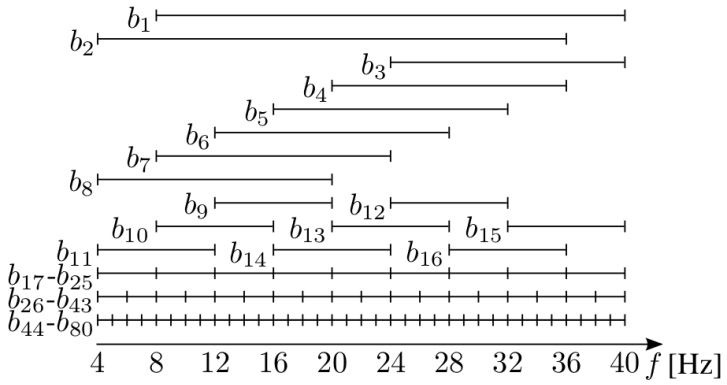


Figure 6.6: Filter bank multi-scale bands. The current Riemannian approach relies on 43 pass-band filters on the frequency spectrum [4-40]Hz with bandwidths varying from 2Hz ($b_{26} - b_{43}$) to 32Hz ($b_1 - b_2$). [35]

Riemannian block. The main idea behind Riemannian covariance feature extraction is to project the signal's covariance matrix into a local tangent space that has properties of an Euclidean space. This allows for the use of traditional Euclidean metrics such as averaging [5.6] covariance matrices or computing their relative distance [5.7]. When projecting to the local tangent space, a different reference matrix is used in each Riemannian block. This matrix is computed as the mean covariance matrix of all the training data after being processed by the corresponding frequency filtering block. The resulting matrix is vectorized using equation [6.4] leading to $(N_c + 1)N_c/2$ features per block with N_c being the number of channels.

A symmetric matrix can be vectorized as follows

$$\overrightarrow{\mathbf{C}} := vect(\mathbf{C}) = \left[c_{1,1}; \sqrt{2}c_{1,2}; ...; c_{N_c,N_c}\right] \in \mathbb{R}^{(N_c+1)N_c/2} \tag{6.4}$$

where c_{ij} represents the elements at row i and column j of the matrix $\mathbf{C}$. The off-diagonal elements should be scaled by $\sqrt{2}$ to preserve the norm $\|\mathbf{C}\|_F = \|vect(\mathbf{C})\|_2$

Classifier. The feature vectors are then fed to a simple linear SVM with l_2-regularization (see Section 5.6.1).

Advantages. This approach enables an interesting performance boost compared to FBCSP regarding the decoding accuracy and robustness.

Limitations. Despite reaching relatively good results, this model has several limitations. First, this approach tends to output a number of features that is several orders of magnitude higher than FBCSP. The quantity of features produced depends on the number of EEG channels and the filter bank configuration and can lead to computationally expansive operations if directly fed to a classifier. The computational complexity of Riemannian metrics are indeed estimated to grow cubically with the number of electrodes. This could make the inference too slow for a real-time EEG classification competition.

6.4 Deep learning approach

This section introduces a deep learning model that could be used as contender to the baselines for MI-EEG classification. Our main goal is to showcase the potential benefits and limitations of using neural networks in the context of both offline and online BCI competitions.

6.4.1 Shallow ConvNet

Our deep learning model is based on the Shallow ConvNet proposed by [54]. This architecture, inspired by the FBCSP approach, is tailored to extract and classify band power features from raw EEG data.

Interpretable architecture. The shallow ConvNet has a compact architecture composed of two convolutional layers, one mean pooling layer, two non-linear operations and a dense layer (Fig. 6.7). The transformations performed by each of these layers are akin to the one appearing in the FBCSP approach. The first two convolutional layers perform indeed a temporal and spatial filtering on the data respectively and are analogous to the band-pass frequency and CSP spatial filtering steps in FBCSP. While the first layer learns how to optimally combine information locally in time, the second layer learns it across the channel axis. The next three operations, namely a square non-linearity, a temporal mean pooling layer and a logarithmic non-linearity, are comparable to the log-variance computation happening at the end of the CSP blocks in FBCSP. The mean pooling operation allows the network to summarize the band powers temporal variations within a trial. Finally a softmax activation outputs the probability associated to each MI task label. The full architecture used for the BCI IV 2a dataset is shown in Table 6.1.

Advantages. Shallow ConvNet displays a clear advantage over the two baseline models as it embeds all the computation steps in a single network, enabling joint optimization. Its simple architecture, based on the popular FBCSP model, allows to keep a relatively good interpretability of the extracted features compared to complex neural network architectures. Furthermore, its fast inference is well suited for online brain-signal decoding, especially when used along a cropping strategy (see 6.7.1).

Potential limitations. Besides the drawbacks of neural networks, mentioned in Section 5.5, this model does not exploit the full potential of the deep learning field as its architecture is based on a non deep learning method. More complex architectures, relying on the use of consecutive CNN and RNN, might be more adequate for capturing both spatial and temporal information. Such approaches are, however, missing in the current BCI literature and often suffer from a lack of feature's interpretability.

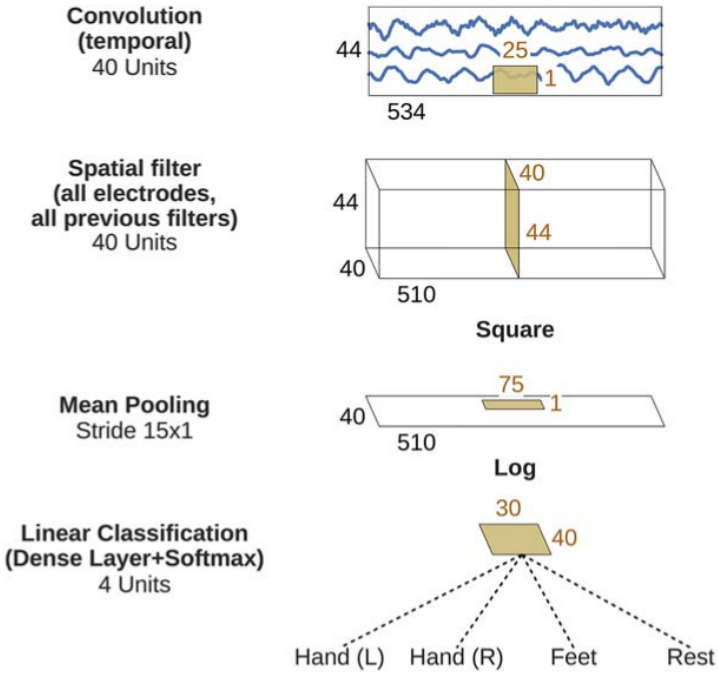


Figure 6.7: Shallow ConvNet architecture. The 2D EEG input is processed by two convolutional layers, in the temporal domain first, and then over the spatial domain. We expect these two layers to increase the SNR similarly to the frequency and spatial filtering steps in FBCSP (see Section 6.3.1). The data is then passed to a square function, a mean pooling layer and a logarithmic function that should extract log-variance power band features. Finally, a dense layer and softmax activation allow the network to classify the initial EEG trial and output a probability for each MI label.

Layer	Input	Operation	Output	Parameters
1	$E \times T$	$40 \times$ Conv2D (1×25)	$40 \times E \times 476$	1 040
2	$40 \times E \times 476$	$40 \times$ Conv2D $(E \times 1)$	$40 \times 1 \times 476$	35 240
	$40 \times 1 \times 476$	BatchNorm	$40 \times 1 \times 476$	-
	$40 \times 1 \times 476$	ELU	$40 \times 1 \times 476$	-
3	$40 \times 1 \times 476$	Square	$40 \times 1 \times 476$	-
	$40 \times 1 \times 476$	MeanPool2D (1×75) with stride (1×15)	$40 \times 1 \times 27$	-
	$40 \times 1 \times 27$	Log	$40 \times 1 \times 27$	-
4	$40 \times 1 \times 27$	Dropout (0.5)	$40 \times 1 \times 27$	-
	$40 \times 1 \times 27$	Flatten	1 080	-
	1080	Dense + Softmax	K	4 324
Total				40 764

Table 6.1: Detailed architecture of Shallow ConvNet. E is the number of channels, T is the number of timestamps and K is the number of labels. Input and output sizes are shown here for $E = 22$ electrodes, $T = 500$ (2-second temporal window at 250 Hz) and $K = 4$ classes.

6.5 Training procedures

In this section, we introduce the training schemes applied for our baseline and deep learning models. Throughout our experiments, we adopt the common practice in which we consider one model per subject. Moreover, we always make sure that the training set is balanced, i.e. has the same amount of trials for each MI task, in order to avoid any performance bias.

6.5.1 Baseline models

The offline training scheme employed for the baseline models can be divided into two iterative steps:

1. *Pilot-specific training.* The baseline model is fitted to the train set of a given subject of interest.
2. *Hyperparameters optimization.* Evaluation metrics (see Section 6.6.1) are then computed on a validation set consisting of unseen data of the same subject. This allows to estimate how well the model would perform on new data. The models hyperparameters can thus be optimized by repeating both steps and saving the model with the best metrics.

In practice, one should use a k-fold cross validation approach that consists of splitting the dataset of a given subject into k subsets of equally distributed labels. $k-1$ of these subsets are utilized as training set to fit the model while the remaining subset is used as a validation set to compute evaluation metrics. This process is repeated k times with a different validation set for each iteration.

6.5.2 Deep learning models

The offline training scheme we employ for the deep learning models can be divided into three steps:

1. *Inter-subject pre-training.* The first step comprises pre-training the model on the full training dataset of all available subjects. This allows the network to exploit the high amount of data to initialize its weights. The model consequently learns general features that characterize a EEG signal as well as the first low level features that will distinguish the different MI classes. The potential benefits of such an approach were already mentioned in [54]. During the pre-training, we regularly test the model performance on a subset of the initial training dataset, called the validation set, using a split ratio of 20%. This set is not seen by the network as it should be disjoint from the actual train set. We stop the pre-training once there is no more improvement in validation metrics (decrease/increase in validation loss/accuracy respectively) for at least a given number of epochs (called patience). We then save the model corresponding to the best validation metric.
2. *Pilot-specific training.* In a second step, we load the previously pre-trained model and continue its training using only the training dataset of the pilot of interest. Just as before, we monitor the validation metrics to know when to stop the training.

3. *Hyperparameters optimization.* We repeat the last two steps and keep the model with the best validation metric in order to tune the parameters. A non-exhaustive list of such parameters are the temporal window location and size, the learning rate, batch size or patience as well as the set of pre-processing steps to apply on the signal.

6.6 Benchmarking the publicly available datasets

This section introduces the evaluation procedure employed to assess each model on the BCI Competition IV datasets (see Chapter 4). We start by presenting our evaluation metrics and how they will be computed during the testing procedure. We then compare each model in terms of classification performance, in order to estimate how well they would do in an offline BCI competition. During this benchmarking, we optimize the extracted temporal window on all trials and use the same one for all subjects during training and testing. Moreover, we do not take into account the inference delay and only care about the decoding performance metrics as it is often the case for offline competitions.

6.6.1 Metrics

We use two performance metrics in order to benchmark our different models in an offline fashion: accuracy and Cohen's kappa coefficient. The former allows to have an interpretable performance metric of our model and is widely used in the field of MI-EEG classification. The latter is a more robust metric as it takes into account the distribution of the true labels but is also more difficult to interpret. This metric is, however, useful when it comes to unbalanced datasets and is sometimes used as main metric during BCI competitions.

Accuracy. The accuracy metric is defined as

$$accuracy = \frac{1}{N_{trials}} \sum_{t=1}^{N_{trials}} I\left[y^t_{pred} = y^t_{true}\right] \tag{6.5}$$

where y^t_{pred} is the predicted label and y^t_{true} is the ground truth for trial t.

Kappa coefficient. Cohen's kappa coefficient is defined as

$$\begin{gathered} \kappa = \frac{p_o - p_e}{1 - p_e} \\ p_o = accuracy \\ p_e = \frac{1}{N^2} \sum_{k=1}^{N_{labels}} \left[\sum_{t=1}^{N_{trials}} \left(I\left[y^t_{pred} = k\right] \right) \sum_{t=1}^{N_{trials}} \left(I\left[y^t_{true} = k\right] \right) \right] \end{gathered} \tag{6.6}$$

where p_o is the relative observed agreement between predicted and true labels (identical to the accuracy metric) and p_e is the hypothetical probability of chance agreement.

A value of $\kappa = 0$ would mean that the classifier is performing as good as random while a value of $\kappa = 1$ would mean that the classifier is in perfect agreement with the ground truth labels.

6.6.2 Testing procedure

Once a given model has been trained, we can evaluate it on the corresponding subject test set. Just like the validation set, the test set consists of a set of trials that has never been seen by the model during training in order to provide an unbiased evaluation metric. We perform the same pre-processing operations and use the same temporal windows as during training. The pre-processed test trials are then fed to the trained model which outputs the predicted MI label. Finally, the evaluation metrics introduced in Section 6.6.1 are computed.

6.6.3 Results - BCIC IV 2a

FBCSP. The first baseline model, introduced in Section 6.3.1, uses the raw temporal window $[2.5 - 4.5]$s[5] of each input trial. Our filtering blocks configuration consists of 9 Butterworth band-pass filters of order 2 ranging from 4 to 40 Hz with a constant bandwidth of 4 Hz. We use the CSP setting of $m = 2$ pairs and feed all features (no feature selection) to a SVM with a RBF kernel and a penalty parameter $C = 20$. Performance metrics are shown in Figures 6.8 and 6.9.

Riemannian. Our second baseline model (see Section 6.3.2), receives as input the raw temporal window $[2.5 - 6]$s. Its filtering block configuration consists of 43 Butterworth band-pass filters of order 2 ranging from 4 to 40 Hz using bandwidths scaling from 2 Hz to 32 Hz. We employ a linear SVM as classifier with a penalty parameters $C = 0.1$. Performance metrics are shown in Figures 6.10 and 6.11.

Shallow ConvNet. Our model of interest, introduced in Section 6.4.1, operates on a pre-processed temporal window $[1.75 - 4]$s of each input trial. The pre-processing operations (see Section 5.1) consist of re-referencing, clipping, standardizing and band-pass filtering the frequency band (0-38)Hz using a Butterworth filter of order 3. During training, we used a batch size of 64 trials and a validation ratio of 0.2. We employed the Adam optimizer with the default parameters of Keras and early stopping with a patience parameter set to 50 epochs. Performance metrics are given in Figures 6.12 and 6.13.

Discussion. As expected, the ConvNet model allows for an interesting increase in performance over both baseline models. However, the training time required to fit a model is more consequent for the neural network than for the FBCSP but remains in the same order of magnitude as the Riemannian approach. Some subjects seem to be more difficult to fit in than others, namely subjects 2, 4, 5 and 6. This variability of performance among the subjects is particularly visible for the baseline models. The ConvNet, on the other hand, is capable of handling most subjects with smaller variations in performance, except for subject 2. Regarding the misclassified trials, a quick analysis of the confusion matrices in Figures 6.9, 6.11 and 6.13 reveals that FBCSP tends to either over-predict the "Left" label (subjects 2, 5 and 6) or to mistake "Foot" with "Tongue" trials (subjects 1, 4 and 9) and "Left" with "Right" trials (subjects 2, 4 and 6). This lack of robustness can be a consequence of overfitting the data partially. The Riemannian approach and the Shallow ConvNet do not seem to suffer from this issue. Pre-training the ConvNet took 190s and allowed for a nice performance boost while reducing the average training time (faster loss convergence).

Model	Mean accuracy	Mean kappa	Train time per trial [ms]	Test time per trial [ms]
FBCSP	0.67 ± 0.16	0.57 ± 0.21	**9**	4
Riemannian	0.75 ± 0.14	0.67 ± 0.19	60	35
ConvNet	0.75 ± 0.12	0.66 ± 0.16	230	**0.4**
ConvNet (pre-trained)	$\mathbf{0.80 \pm 0.08}$	$\mathbf{0.73 \pm 0.11}$	140	**0.4**

Table 6.2: Performance summary. Average metrics (w.r.t. subjects) and timings[6]. for each model on the BCIC IV 2a dataset.

[5]The motor imagery phase lasts from t=3s to t=6s.

[6]These numbers benefit from a speed up due to the vectorization of stacked trials.

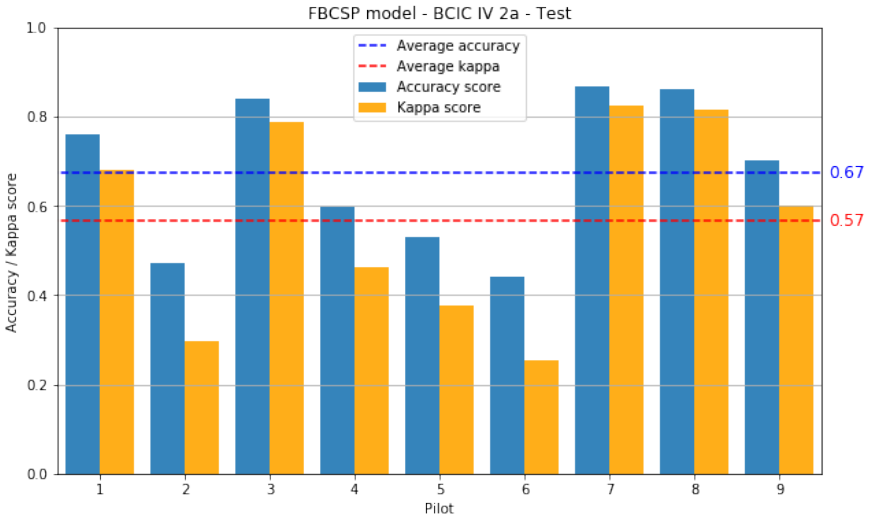


Figure 6.8: Summary bar chart for FBCSP model. Accuracy and kappa scores for all 9 pilots of the BCIC IV 2a dataset (4 classes). Each score was computed using the subject-specific trained model and the corresponding test set.

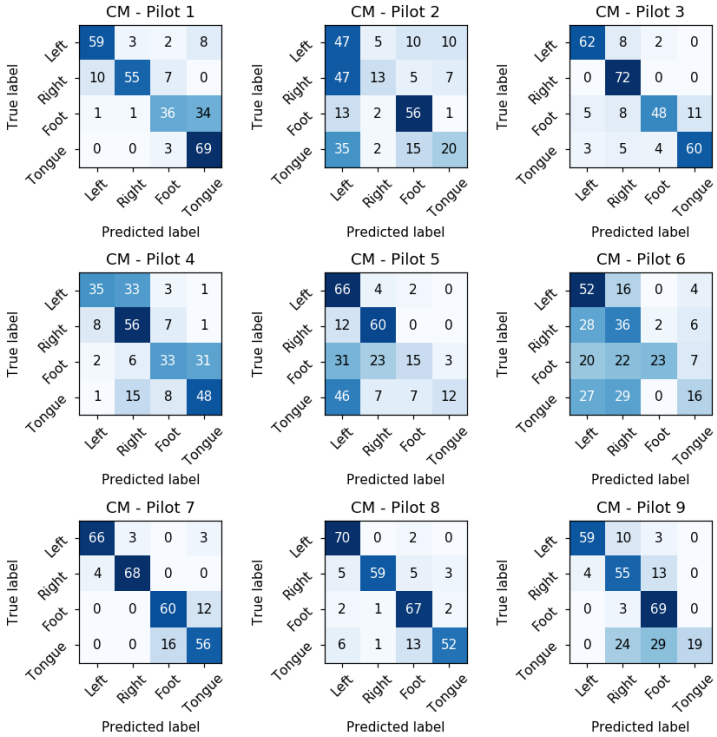


Figure 6.9: Detailed results for FBCSP. Confusion matrices associated to results introduced in Figure 6.8. This allows to detect, for each pilot, which motor imagery tasks are misclassified by the model.

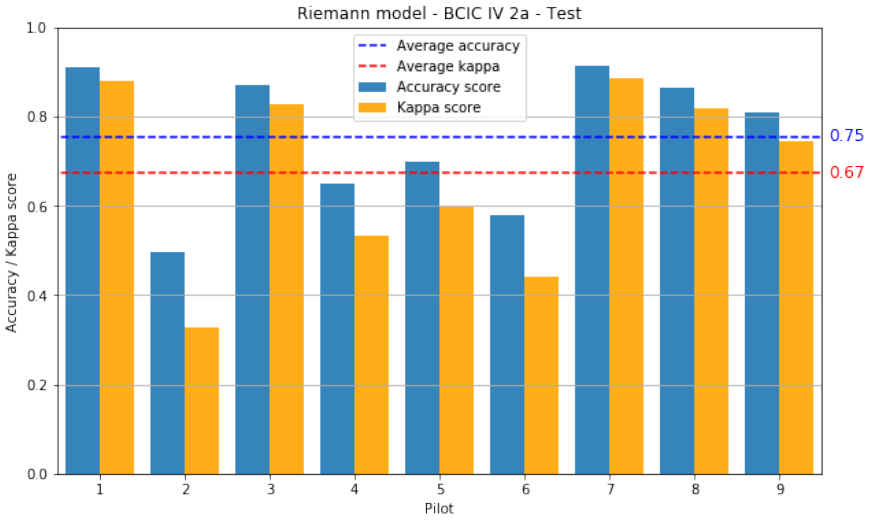


Figure 6.10: Summary bar chart for Riemannian model. Accuracy and kappa scores for all 9 pilots of the BCIC IV 2a dataset. Each score was computed using the subject-specific trained model and the corresponding test set.

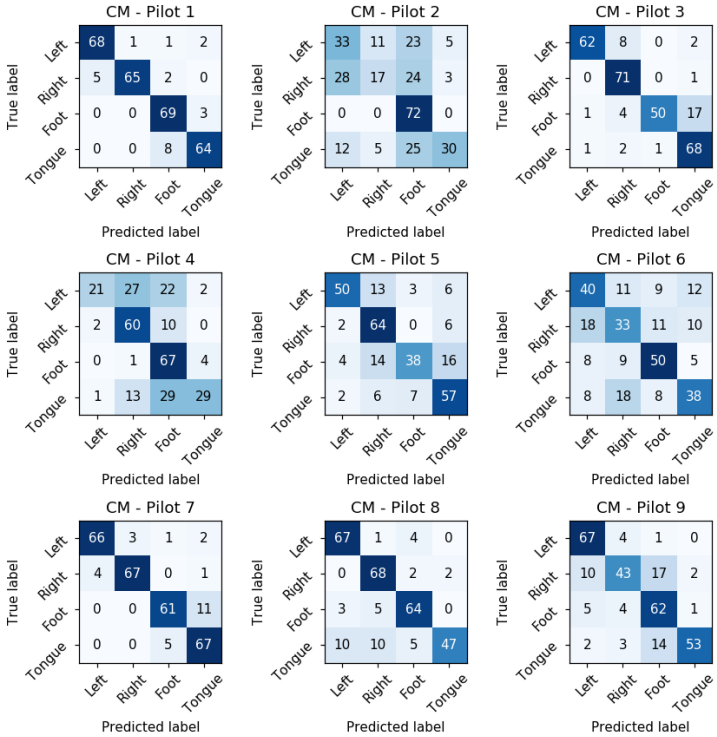


Figure 6.11: Detailed results for Riemannian model. Confusion matrices associated to results introduced in Figure 6.10.

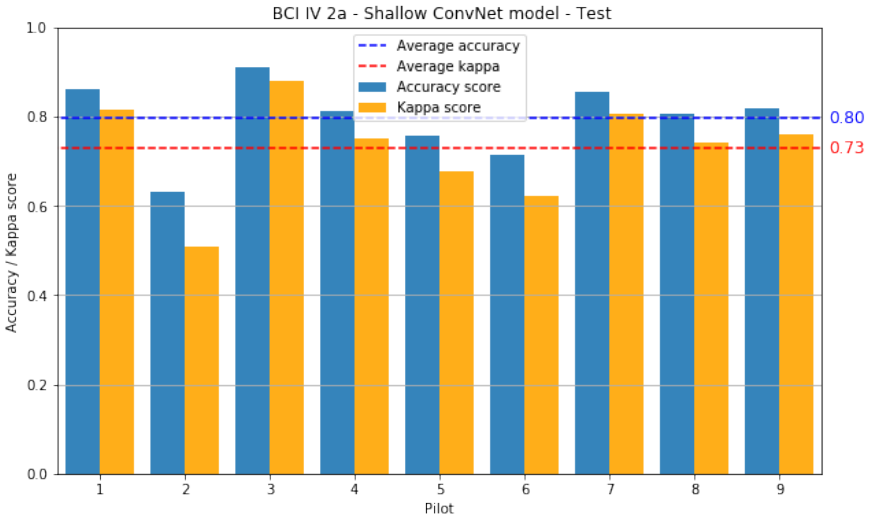


Figure 6.12: Summary bar chart for Shallow ConvNet model. Accuracy and kappa scores for all 9 pilots of the BCIC IV 2a dataset. Each score was computed using the subject-specific trained model and the corresponding test set. We consider the pre-trained version of the ConvNet.

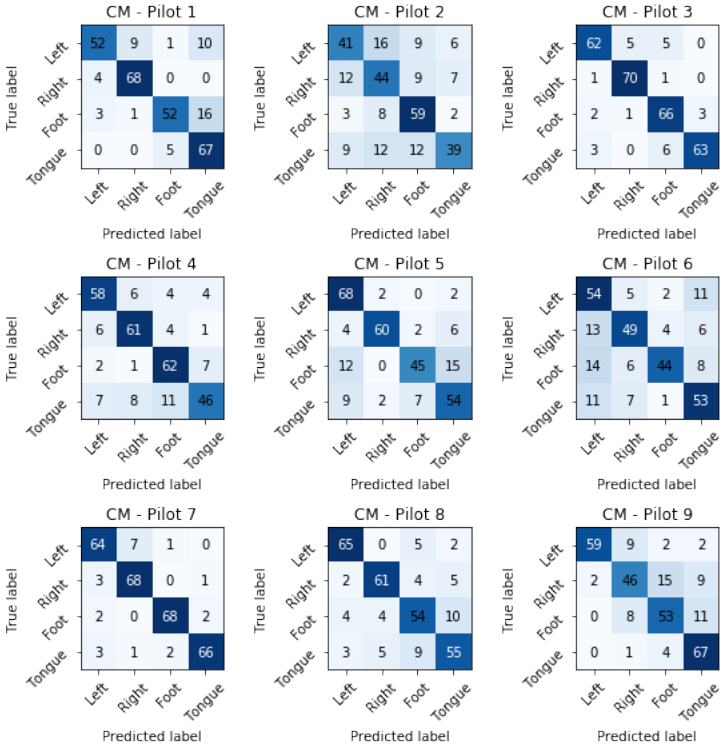


Figure 6.13: Detailed results for ConvNet. Confusion matrices associated to results introduced in Figure 6.12.

6.6.4 Results - BCIC IV 2b

The particularity of the BCIC IV 2b dataset, despite proposing two motor imagery tasks, is that each subject dataset builds upon five recording sessions while only two of them are necessary for the BCIC IV 2a dataset. Therefore, this constitutes a good dataset for assessing the ability of a model to generalize over multiple sessions.

FBCSP. We operate on the raw temporal window $[3.5 - 5.5]$s[7] of each input trial. The rest of the pipeline is identical to the one used for the BCIC IV 2a dataset, except the CSP setting which is here set to $m = 1$ pair. Performance metrics on the BCIC IV 2b dataset are shown in Figures 6.14 and 6.15.

Riemannian. We feed the raw temporal window $[3.5 - 7]$s of each input trial to the model. The rest of the pipeline is identical to the one employed for the BCIC IV 2a dataset, except the penalty parameter which is here set to $C = 0.01$. Performance metrics are given in Figures 6.16 and 6.17.

Shallow ConvNet. We use a pre-processed temporal window $[2.75 - 5]$s of each input trial. The pre-processing operations consist of clipping and standardizing. Other pre-processing operations did not lead to any performance improvement. The rest of the pipeline is identical to the one employed for the BCIC IV 2a dataset. Performance metrics are shown in Figures 6.18 and 6.19.

Discussion. Again, the ConvNet model shows its superior decoding capabilities over the two baseline models. Looking at the detailed results, subjects 2 and 3 seem to be particularly challenging for the baselines, while the ConvNet is able to achieve decent performance. In terms of timings, FBCSP is the fastest model to train while the Shallow ConvNet is the fastest to perform inference. Pre-training the neural network on all subjects took approximately 90s and enabled a small boost in decoding performance while reducing the training time of the model. Regarding, the optimized temporal window values, it seems that the ConvNet might also benefit from learning the temporal information induced by the visual cue preceding the MI phase.

Model	Mean accuracy	Mean kappa	Train time per trial [ms]	Test time per trial [ms]
FBCSP	0.77 ± 0.13	0.55 ± 0.26	**1**	0.3
Riemannian	0.79 ± 0.15	0.58 ± 0.31	10	6
ConvNet	0.83 ± 0.09	0.66 ± 0.19	75	**0.1**
ConvNet (pre-trained)	$\mathbf{0.85 \pm 0.08}$	$\mathbf{0.70 \pm 0.17}$	25	**0.1**

Table 6.3: Performance summary. Average metrics (w.r.t. subjects) and timings for each model on the BCIC IV 2b dataset.

[7] The motor imagery phase lasts from t=4s to t=7s.

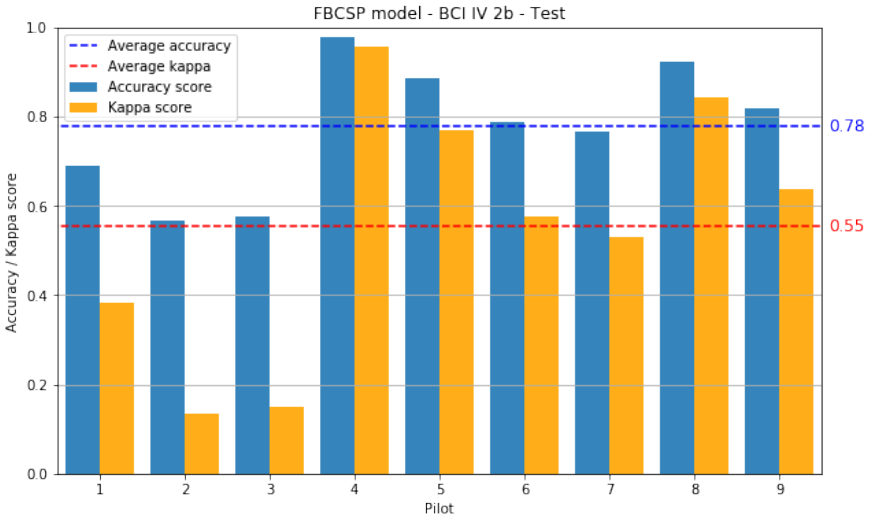


Figure 6.14: Summary bar chart for FBCSP model. Accuracy and kappa scores for all 9 pilots of the BCIC IV 2b dataset. Each score was computed using the subject-specific trained model and the corresponding test set.

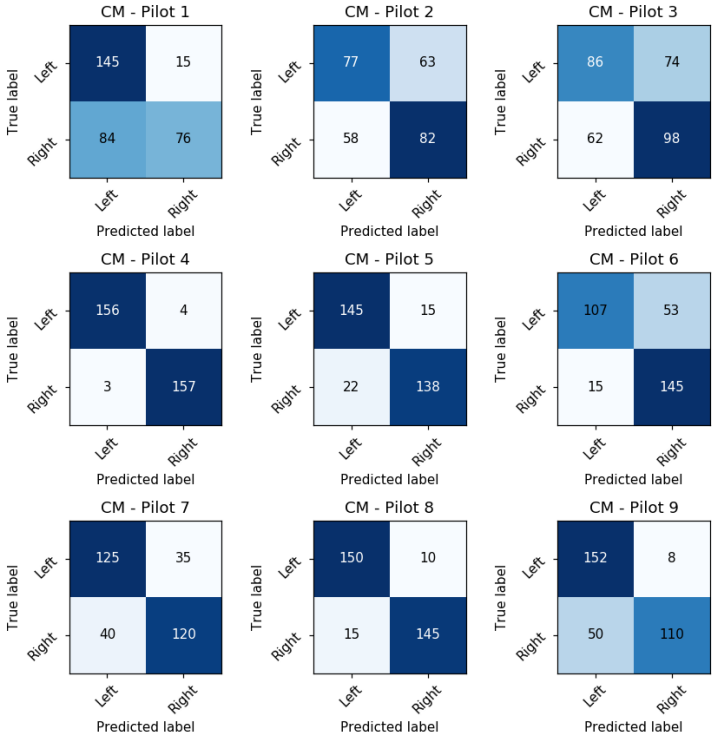


Figure 6.15: Detailed results for FBCSP. Confusion matrices associated to results introduced in Figure 6.14. This allows to detect, for each pilot, which motor imagery tasks are misclassified by the model.

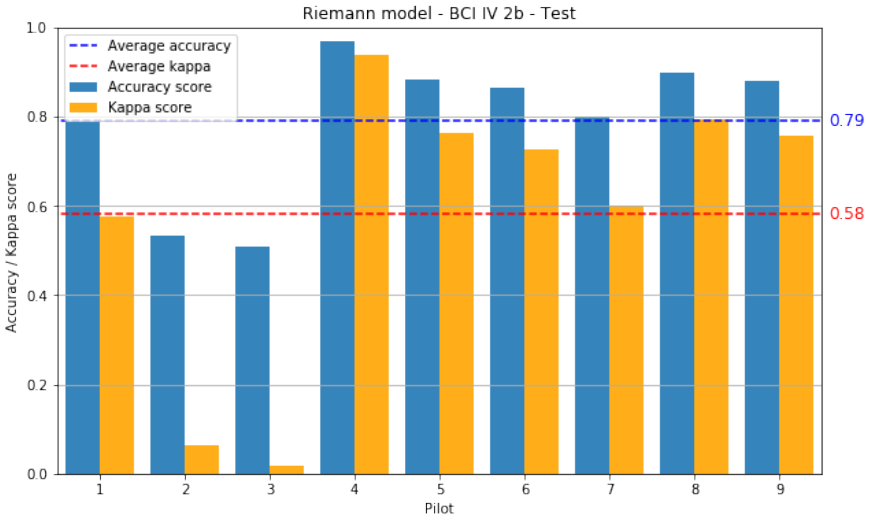


Figure 6.16: Summary bar chart for Riemannian model. Accuracy and kappa scores for all 9 pilots of the BCIC IV 2b dataset. Each score was computed using the subject-specific trained model and the corresponding test set.

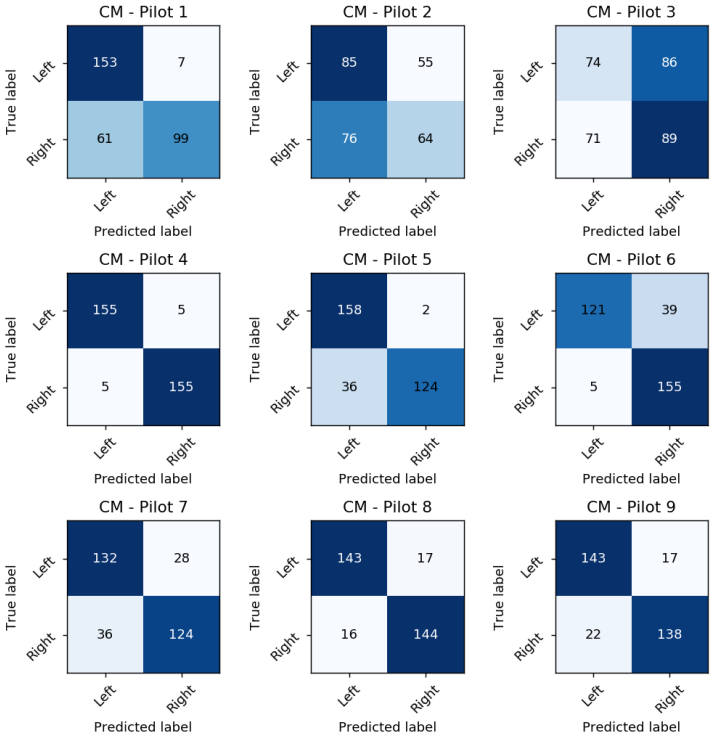


Figure 6.17: Detailed results for Riemannian model. Confusion matrices associated to results introduced in Figure 6.16.

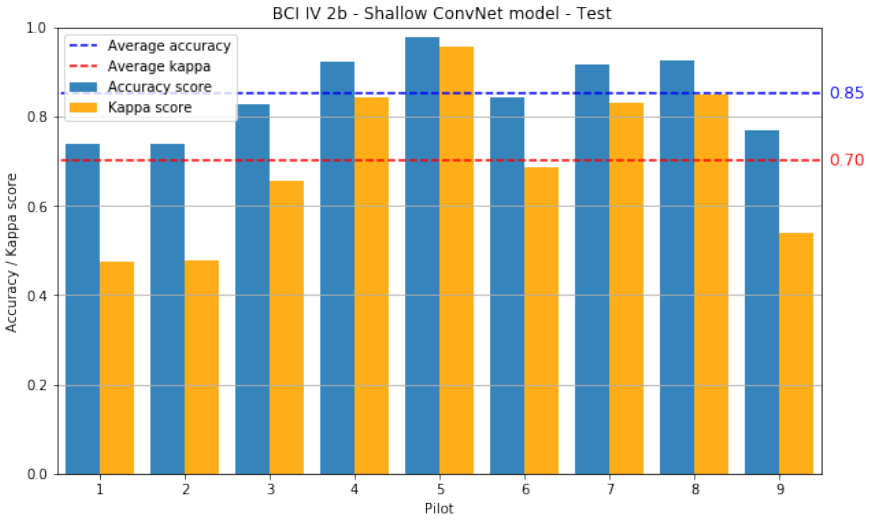


Figure 6.18: Summary bar chart for Shallow ConvNet model. Accuracy and kappa scores for all 9 pilots of the BCIC IV 2b dataset. Each score was computed using the subject-specific trained model and the corresponding test set. We consider the pre-trained version of the ConvNet.

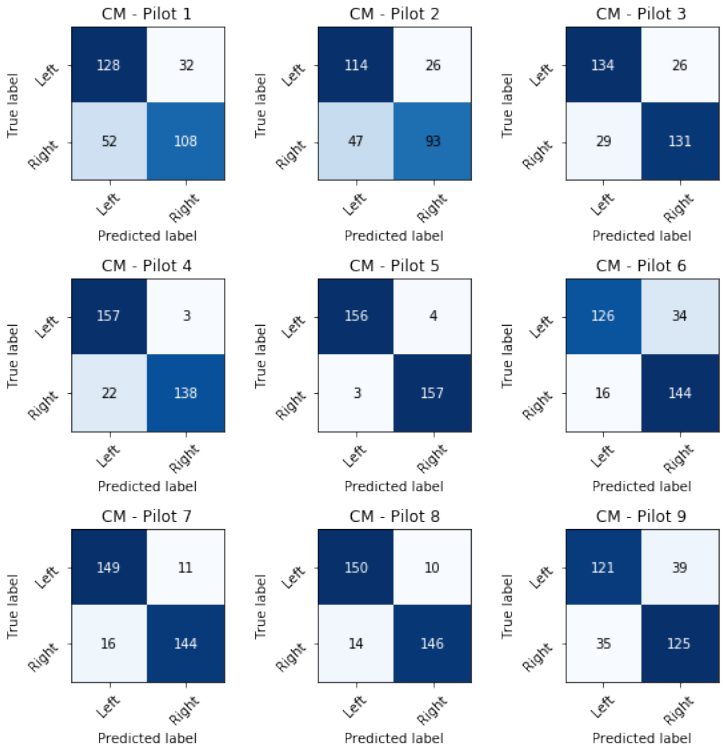


Figure 6.19: Detailed results for ConvNet. Confusion matrices associated to results introduced in Figure 6.18.

6.7 Benchmarking our Competition dataset

Similarly to Section 6.6, this section introduces the evaluation procedure employed to assess each model on our Competition dataset, presented in Chapter 4. Considering the real-time nature of the Cybathlon BCI competition, we employ a slightly different strategy to train and evaluate the models, bringing a complementary analysis for the different approaches. Furthermore, due to the lack of test sessions like in BCIC datasets, we used a stratified 5-fold cross-validation in order to get a proper estimate of the performance metrics. We will mainly focus on session 1 of the Competition dataset by providing detailed metrics for each model. A similar analysis can be run on the remaining sessions, for which we give a concise performance summary.

6.7.1 Cropping strategy

A cropping procedure can be utilized during the training and evaluation to augment the number of trials and force the network to exploit all available information in the EEG signal. This procedure consists of cutting each trial into overlapping time-windows during the motor imagery phase. The interest is two-fold: augmenting the size of our dataset and reducing the temporal window length, leading to faster inference. Furthermore, using small crops also unlocks the possibility of improving the inference robustness via predictions averaging. This approach was mentioned in [54], where an increase in offline decoding performance for the neural network models was reported. We believe that it could also be beneficial in the context of an online BCI competition regarding the trade-off between decoding delay and decoding accuracy. The minimum time needed to decode an input brain signal should roughly correspond to the duration of the crop added to the model inference time.

6.7.2 Online metrics

We rely on the same performance metrics as the one introduced in 6.6.1 as well as considering their "online" version. These online metrics are more adequate to get a realistic grasp of how each model would perform in an real-time BCI competition such as Cybathlon. In order to respect the principle of causality (only accessing the past data), we slide a buffer looking at past values over the full trials and compute the performance metrics by looking at the model predictions every 100ms. This is also particularly useful for visualizing the decoding delay induced by both the buffer input size and the model inference time (Fig. 6.20).

6.7.3 Visual interface

We also provide a visual interface that is able to interact with the Cybathlon game environment. This interface loads a trained model along with a pre-recorded dataset and outputs the inferred actions to the game avatar in a simulated online fashion. It also allows to visualize the current time-window of EEG data that the model receives as input which might be helpful to detect signal artifacts as well as the weaknesses of the model. Furthermore, it is worth mentioning that such a visual interface could theoretically be integrated with the data collection procedure by connecting it directly to the electrodes while the subject is playing the game (Fig. 6.21).

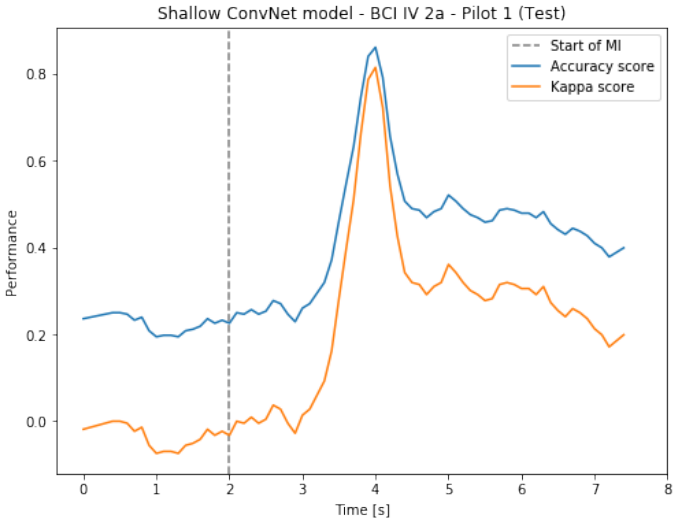


Figure 6.20: Online performance. At each times step, the average accuracy and kappa scores are computed over all test trials in a sliding buffer window looking at the past data. In this example, we used a Shallow ConvNet model trained on subject 1 of BCIC IV 2a dataset.

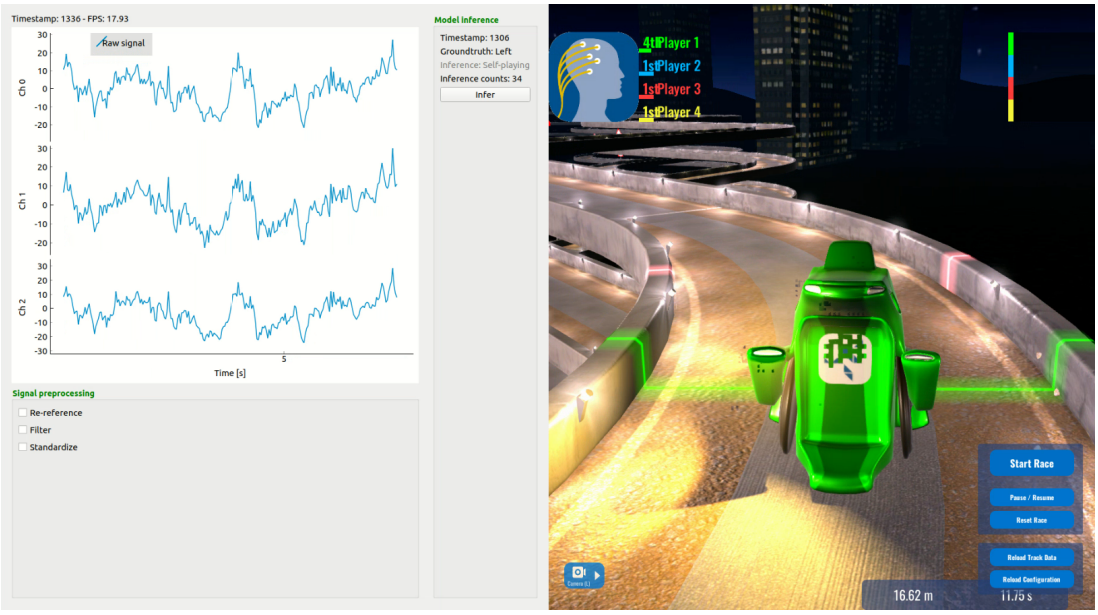


Figure 6.21: Visualization interface. Our interface (left) can send commands to the avatar (right) using either the log files of the game (self-playing mode) or the trained model inferences (feed-back mode). The pre-recorded EEG signal can be displayed in real-time and fed as input to the model.

6.7.4 Results on the Competition dataset - Session 1

FBCSP. We operate on the raw temporal window $[2.0 - 5.5]$s[8] of each input trial, which corresponds to the full motor imagery phase. Our filtering blocks configuration contains 9 Butterworth band-pass filters of order 2 ranging from 4 to 40 Hz with a constant bandwidth of 4 Hz. We use the CSP setting of $m = 2$ pairs and feed all features (no feature selection) to a SVM with a RBF kernel and a penalty parameter $C = 10$. Performance metrics are shown in Figures 6.22, 6.23 and 6.24.

Riemannian. For our second baseline model, we use the standardized temporal window $[2.0 - 5.5]$s of each input trial. Our filtering blocks configuration consist of 43 Butterworth band-pass filters of order 2 ranging from 4 to 40 Hz using different bandwidths from 2 Hz to 32 Hz. For the classifier, we employ a linear SVM with a penalty parameter $C = 1$. Performance metrics are shown in Figures 6.25, 6.26 and 6.27.

Shallow ConvNet. Our ConvNet model uses a pre-processed temporal window $[2. - 5.5]$s of each input trial from which we extract 14 overlapping crops of 0.5s in length (corresponding to a 50% overlapping ratio). The pre-processing operations consist of re-referencing, clipping and standardizing. During training, we used a batch size of 32 and a validation ratio of 0.1. We employed the Adam optimizer with the default parameters of Keras and early stopping with a patience parameter set to 50 epochs. Performance metrics are given in Figures 6.28, 6.29 and 6.30.

Discussion. Surprisingly, FBCSP is clearly beating the two other models, which contrasts our results obtained for the BCIC IV 2 datasets. The poor performance of the Shallow ConvNet is most likely due to the small training data available and the lack of a data cleaning procedure. The Riemannian approach, which is expected to be more robust than FBCSP, is probably hindered by the high number of channels (see Limitations paragraph in Section 5.4). Cropping did not bring about any improvement, both in terms of online and offline performance, when employed with the baseline models, but it had a significant and beneficial impact on the neural network. The use of a pre-training strategy has not been considered within our Competition dataset as the trials were collected on a single subject with specific MI tasks and electrode positioning (not compatible with the BCIC IV datasets). Regarding the online performance of all models, we noticed relatively high values before the MI phase. This is due to the specific recording protocol of session 1, in which each MI task was recorded in consecutive trials of the same run. Therefore, we strongly believe that this protocol puts the subject into a specific brain state, during the whole run, which is captured by the models.

Model	Mean accuracy	Mean kappa	Train time per trial [ms]	Test time per trial [ms]
FBCSP	$\mathbf{0.78 \pm 0.07}$	$\mathbf{0.71 \pm 0.10}$	**30**	12
Riemannian	0.75 ± 0.14	0.67 ± 0.19	220	125
ConvNet	0.70 ± 0.14	0.60 ± 0.19	80	**3**

Table 6.4: Performance summary. Average metrics (w.r.t. folds) and timings for each model on our Competition dataset (session 1). All four MI labels are considered here.

[8]The motor imagery phase lasts from t=2s to t=5.5s.

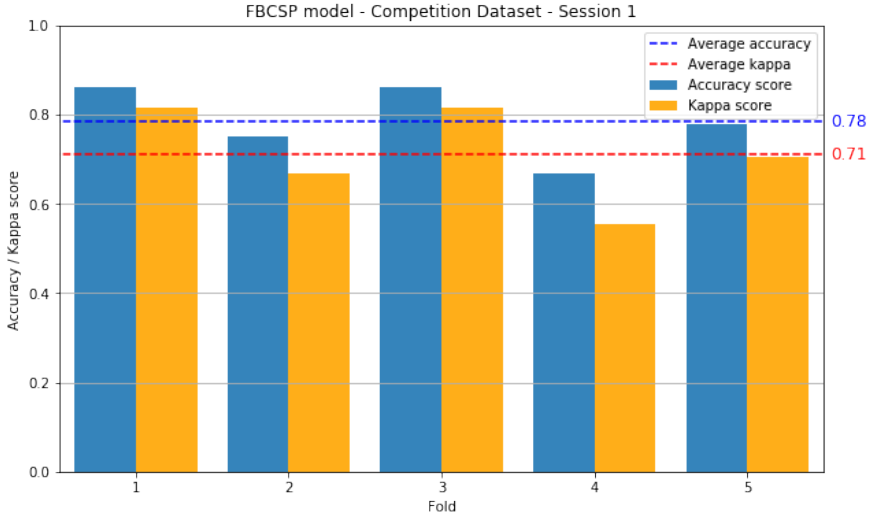


Figure 6.22: Bar chart - FBCSP. Accuracy and kappa scores for each test subset of the 5-fold cross validation procedure.

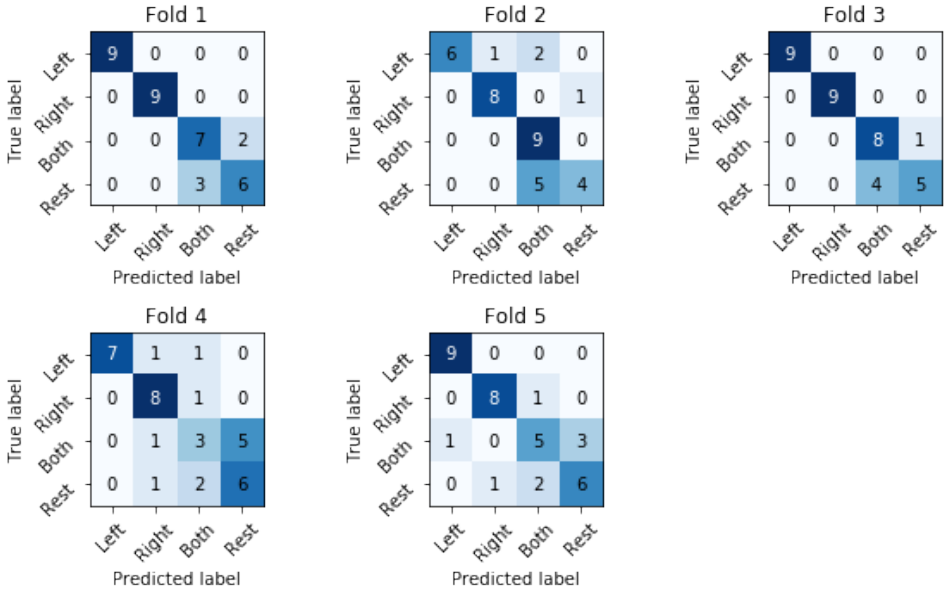


Figure 6.23: Detailed results for FBCSP. Confusion matrices associated to results introduced in Figure 6.22.

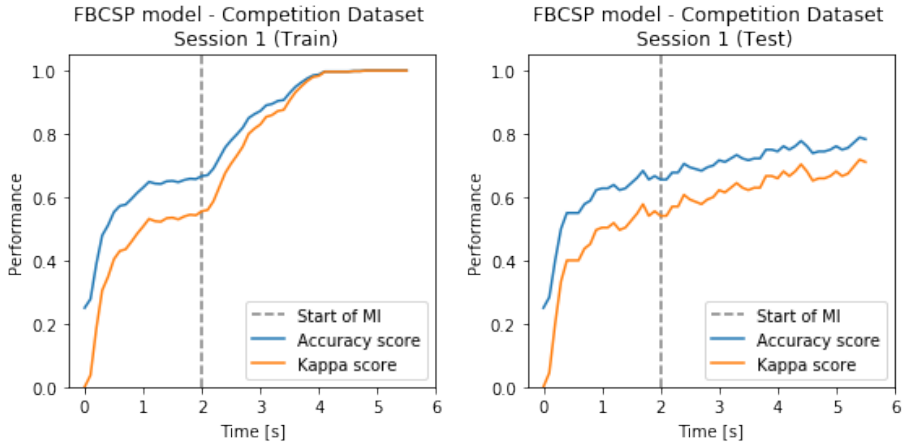


Figure 6.24: Online performance - FBCSP. The model is evaluated every 100ms on the full trial by considering the past 3.5s of EEG data.

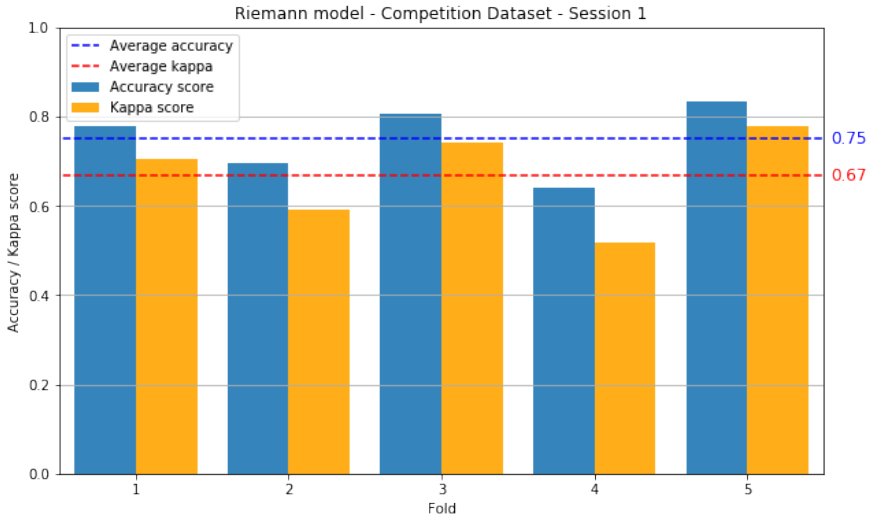


Figure 6.25: Bar chart - Riemannian model. Accuracy and kappa scores for each test subset of the 5-fold cross validation procedure.

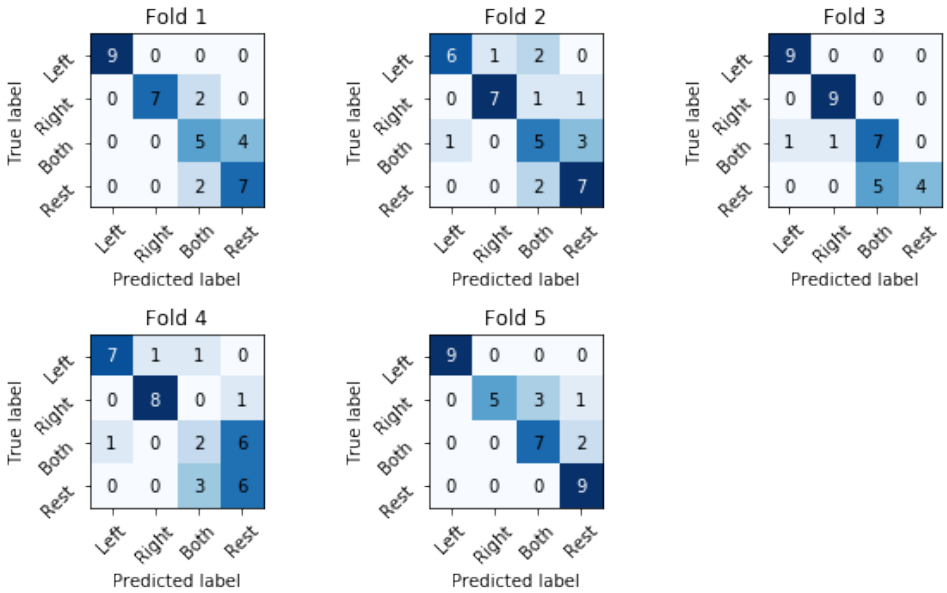


Figure 6.26: Detailed results for Riemannian model. Confusion matrices associated to results introduced in Figure 6.25.

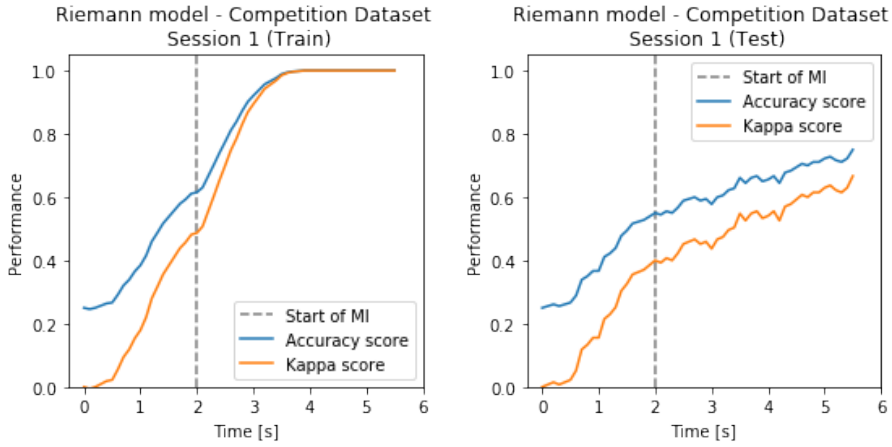


Figure 6.27: Online performance - Riemannian model. The model is evaluated every 100ms by considering the past 3.5s of EEG data.

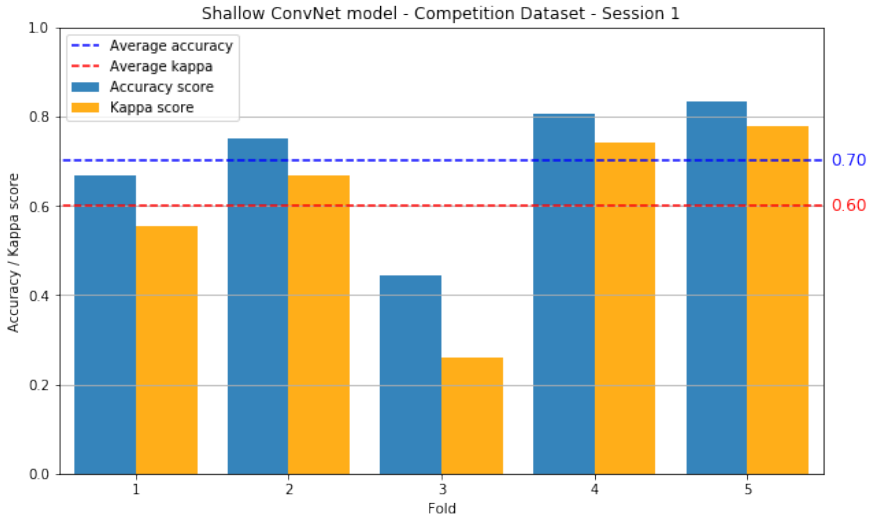


Figure 6.28: Bar chart - Shallow ConvNet. Accuracy and kappa scores for each test subset of the 5-fold cross validation procedure.

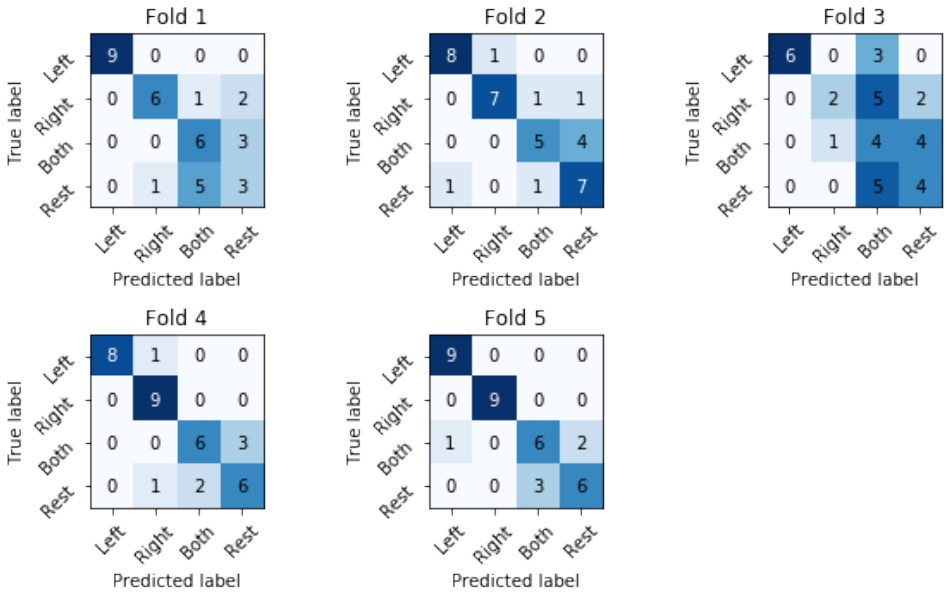


Figure 6.29: Detailed results for ConvNet. Confusion matrices associated to results introduced in Figure 6.28.

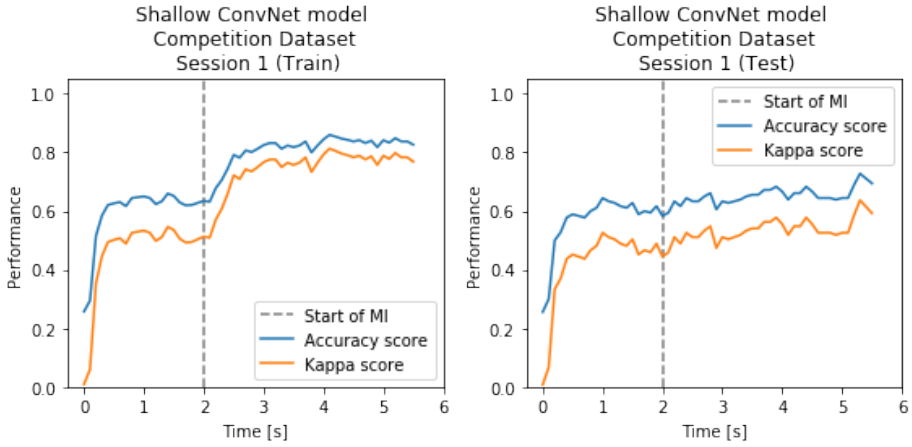


Figure 6.30: Online performance - ConvNet. The model is evaluated every 100ms on the full trial by considering the past 0.5s of EEG data.

6.7.5 Results on the Competition dataset - Remaining sessions

Here, we present a condensed summary of our complementary results, obtained in the remaining sessions of the Competition dataset. The recording procedure is the same as the one introduced in Section 4.2. However, in contrast with session 1, we only assess two classes: action and rest state. The action class is composed of the three dynamic motor imagery tasks, namely closing left, right and both hands.

FBCSP. We operate on the raw temporal window $[2.5 - 6]$s[9] and use the exact same model settings as for session 1.

Riemannian. As an input we feed the temporal window $[2.5 - 6]$s, standardized for session 6 and raw for the others. The SVM penalty parameter is set to $C = 1$ for sessions 3 and 4 and $C = 10^{-2}$ for sessions 5,6 and 8.

Shallow ConvNet. This model uses a pre-processed temporal window $[2.5 - 6]$s of each input trial from which we extract 14 overlapping crops of length 0.5s. The pre-processing operations consist of re-referencing, clipping and standardizing. All other parameters remain identical to the analysis of session 1.

Discussion. As stated in Table 6.5, both Riemannian and ConvNet models reach top performance over all sessions, with a small advantage for the ConvNet. This might be the case thanks to the larger amount of trials per label available in the training set as the trials corresponding to each of the three dynamic MI tasks were concatenated together. We also compare online performance on session 3 in Figures 6.31, 6.32 and 6.33. This time, all models are trained using the same cropping strategy as for the ConvNet (14 overlapping crops of length 0.5s from each trial input). Globally, we observe better performance for the Shallow ConvNet model, which also seems less prone to overfitting the training data (left side). On the test set (right side), the ConvNet is capable of reaching 80% decoding accuracy 1.5s after the start of the MI phase while FBCSP never reaches this accuracy and the Riemannian model takes about 2s. In an online context, however, the Riemannian approach would not be an adequate solution considering its high inference time: 3.3s compared to only 9ms for the ConvNet in average.

Model	Session 3	Session 4	Session 5	Session 6	Session 8
FBCSP	0.78 ± 0.10 0.56 ± 0.20	0.85 ± 0.10 0.70 ± 0.20	0.77 ± 0.04 0.53 ± 0.09	0.78 ± 0.03 0.57 ± 0.06	0.77 ± 0.04 0.53 ± 0.08
Riemannian	0.88 ± 0.04 0.75 ± 0.07	**0.90 ± 0.04** **0.80 ± 0.09**	0.74 ± 0.02 0.48 ± 0.03	0.86 ± 0.06 0.72 ± 0.13	**0.87 ± 0.06** **0.73 ± 0.12**
ConvNet	**0.90 ± 0.03** **0.79 ± 0.07**	0.88 ± 0.06 0.77 ± 0.11	**0.79 ± 0.06** **0.58 ± 0.12**	**0.86 ± 0.04** **0.72 ± 0.09**	0.84 ± 0.07 0.68 ± 0.14

Table 6.5: Performance summary. Average metrics on all sessions[10] of our Competition dataset. Binary labels (action versus rest) are considered here to compute both accuracy and kappa score for each model.

[9] The motor imagery phase lasts from t=2s to t=6s.

[10] Sessions 2 and 7 were not taken into account due to data formatting issues.

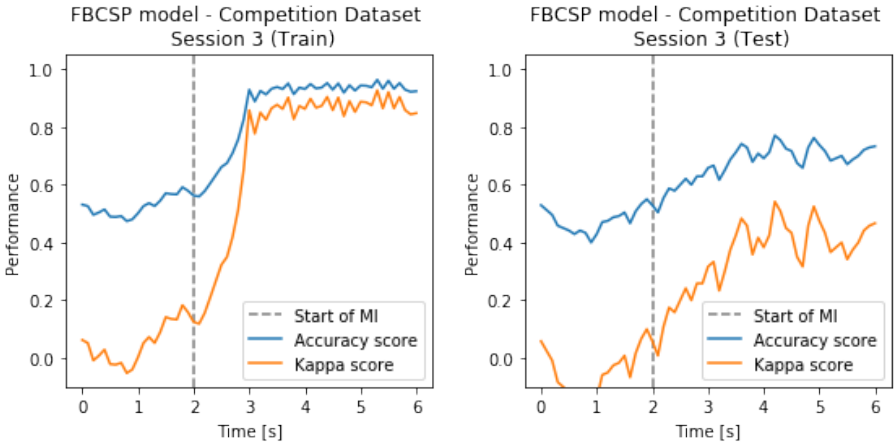


Figure 6.31: Online performance - FBCSP. The model (trained with cropping like ConvNet) is evaluated every 100ms on the full trial by considering the past 0.5s of EEG data. The average inference time is 80ms.

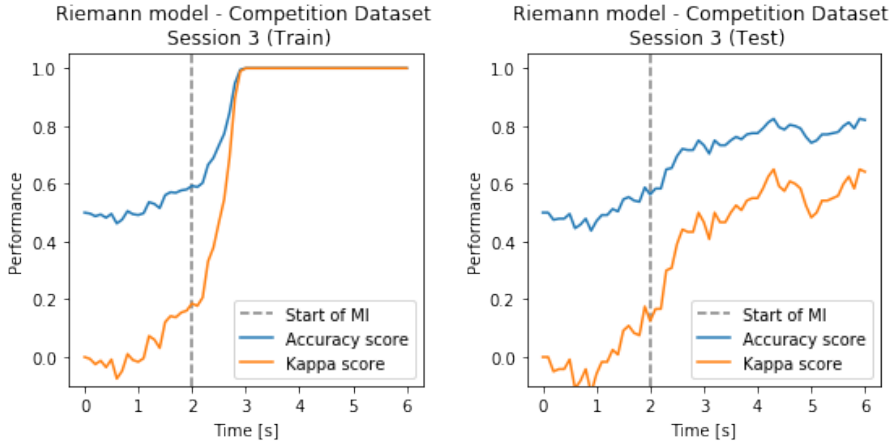


Figure 6.32: Online performance - Riemannian model. The model (trained with cropping like ConvNet) is evaluated every 100ms on the full trial by considering the past 0.5s of EEG data. The average inference time is 3.3s.

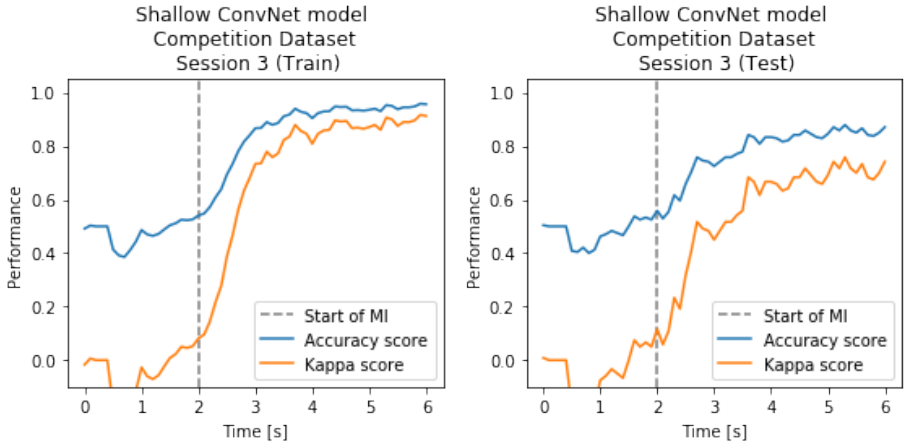


Figure 6.33: Online performance - ConvNet. The model is evaluated every 100ms on the full trial by considering the past 0.5s of EEG data. The average inference time is 9ms.

6.8 Room for improvement

Our analysis shows that, despite not beating the baseline models by a large margin, ConvNets can be considered as a great contender in BCI competitions. We believe that deep learning approaches would lead to even more impressive results if the following improvements were taken into consideration:

- ***Larger datasets.*** Current EEG datasets are relatively small in number of trials for deep learning methods to show their full potential. One solution would be to record more data, collected in several sessions under the same conditions. This is, however, not always possible, requires a lot of time for both the subject and the preparators as well as specific hardware. Another solution would be to use augmentation techniques (similar to our cropping strategy) to artificially augment the number of trials. Generative adversarial networks (GANs) are a type of architecture often used in Computer Vision to generate artificial data and might be a relevant approach to try in the context of MI-EEG classification [55].
- ***Data cleaning.*** Another enhancement opportunity in our pipeline would be to add a data cleaning block in order to detect artifacts, remove bad trials and select relevant channels [56]. This cleaning process is sometimes done by experts in openly available datasets like BCI Competition IV but that is not the case for our Competition dataset. Such a procedure would certainly allow the models to learn more relevant features without being impacted by outlier signals.
- ***More complex models.*** As mentioned earlier, the Shallow ConvNet model has a very basic architecture, similar to FBCSP, and might not showcase the full potential of deep learning. Many new architectures have been recently published and are, for instance, based on integrating spatio-temporal EEG information into a CNN-RNN combination [55] or feeding a CNN with short-time Fourier transform (STFT) spectrograms [57]. Designing a model that is able to learn from both temporal and electrode spatial information might be relevant in the near future. Finally, allowing the network to receive any input size, both in terms of number of electrodes and temporal window size, would enable transfer learning strategies between different datasets.

Conclusion

On a high level, this work described each element of our full BCI pipeline, from the EEG data formatting, pre-processing and feature extraction to the model training and evaluation as well as signal and metrics visualization. In the future, new or existing blocks could easily be integrated to assess other decoding strategies or benchmark different datasets.

More specifically, this work explored the use of neural networks in the context of motor imagery classification from EEG data. We leveraged a simple architecture, as proof of concept, to show the potential and limits of deep learning for both offline and online BCI competitions. Two baselines showcasing the current state-of-the-art decoding MI-EEG models were employed as comparison. Throughout our experiments, we showed that a simple CNN model can successfully extract relevant information from non-stationary time series in a jointly optimized process. One of its main advantages is its capacity for fast inference, which is a key component in online BCI competitions. Also, in some cases, the CNN model allowed for a small boost in decoding performance compared to baselines and paved the way for new training strategies such as trial cropping and transfer learning over multiple subjects. However, we also encountered several limitations regarding the use of deep learning in the field of BCI classification. First and foremost, the lack of large MI-EEG datasets with a standardized recording procedure makes it difficult for neural networks to show their full potential. Furthermore, the loss of interpretability of the extracted features hampers the design of new architectures that would be more successful and more suited for EEG data.

All in all, we believe that the use of deep learning for BCI is still in its infancy but is definitely worth considering for the Cybathlon competition as the Shallow ConvNet often proved to be more capable of decoding motor imagery tasks from EEG data than traditional methods.

Bibliography

[11] Y. LeCun et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* (1989).

[12] C. Nwankpa et al. “Activation Functions: Comparison of trends in Practice and Research for Deep Learning”. In: *CoRR* (2018).

[16] S. Ruder. “An overview of gradient descent optimization algorithms”. In: *CoRR* (2016).

[20] S. Perdikis et al. “The Cybathlon BCI race: Successful longitudinal mutual learning with two tetraplegic users”. In: *PLOS Biology* (2018).

[21] K. Statthaler et al. “Cybathlon experiences of the Graz BCI racing team Mirage91 in the brain-computer interface discipline”. In: *Journal of NeuroEngineering and Rehabilitation* (2017).

[22] D. Novak et al. “Benchmarking Brain-Computer Interfaces Outside the Laboratory: The Cybathlon 2016”. In: *Frontiers in Neuroscience* (2018).

[25] S. Luck. “An Introduction to the Event-Related Potential Technique”. In: *The MIT Press* (2005).

[27] L.A. Farwell and E. Donchin. “Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials”. In: *Electroencephalography and Clinical Neurophysiology* (1988).

[28] G. Pfurtscheller and F.H. Lopes da Silva. “Event-related EEG/MEG synchronization and desynchronization: basic principles”. In: *Clinical Neurophysiology* (1999).

[29] T. Tamura et al. “Audio-Vocal Monitoring System Revealed by Mu-Rhythm Activity”. In: *Frontiers in psychology* (2012).

[30] L. Beltrachini, N. von Ellenrieder, and C. H. Muravchik. “Shrinkage Approach for Spatiotemporal EEG Covariance Matrix Estimation”. In: *IEEE Transactions on Signal Processing* (2013).

[31] F. Yger, F. Lotte, and M. Sugiyama. “Averaging Covariance Matrices for EEG Signal Classification based on the CSP: an Empirical Study”. In: *EUSIPCO* (2015).

[32] A. Barachant et al. “Classification of covariance matrices using a Riemannian-based kernel for BCI applications”. In: *Neurocomputing* (2013).

[33] Z.J. Koles, M.S. Lazar, and S.Z. Zhou. “Spatial patterns underlying population differences in the background EEG”. In: *Brain Topography* (1990).

[34] K.K. Ang et al. “Filter Bank Common Spatial Pattern Algorithm on BCI Competition IV Datasets 2a and 2b”. In: *Frontiers in Neuroscience* (2012).

[35] M. Hersche et al. “Fast and Accurate Multiclass Inference for MI-BCIs Using Large Multiscale Temporal and Spectral Features”. In: (2018).

[36] A. Barachant et al. “Classification of covariance matrices using a Riemannian-based kernel for BCI applications”. In: *Neurocomputing* (2013).

[37] B. Blankertz et al. “Optimizing Spatial filters for Robust EEG Single-Trial Analysis”. In: *Signal Processing Magazine, IEEE* (2008).

[39] A. Barachant et al. “Riemannian Geometry Applied to BCI Classification”. In: *LVA/ICA* (2010).

[41] M. van Putten, S. Olbrich, and M. Arns. “Predicting sex from brain rhythms with deep learning”. In: *Scientific Reports* (2018).

[44] R. Pascanu, T. Mikolov, and Y. Bengio. “Understanding the exploding gradient problem”. In: *CoRR* (2012).

[45] S. Hochreiter and J. Schmidhuber. “Long Short-term Memory”. In: *Neural computation* (1997).

[46] M. Li et al. “Combined long short-term memory based network employing wavelet coefficients for MI-EEG recognition”. In: (2016).

[47] S. Kumar, A. Sharma, and T. Tsunoda. “Brain wave classification using long short-term memory network based OPTICAL predictor”. In: *Scientific Reports* (2019).

[48] Z. Tayeb et al. “Validating Deep Neural Networks for Online Decoding of Motor Imagery Movements from EEG Signals”. In: *Sensors* (2019).

[49] I.J. Goodfellow, J. Shlens, and C. Szegedy. “Explaining and Harnessing Adversarial Examples”. In: *arXiv e-prints* (2014).

[51] B.T. Smith. “Lagrange Multipliers Tutorial in the Contextof Support Vector Machines”. In: (2004).

[54] R. Schirrmeister et al. “Deep learning with convolutional neural networks for decoding and visualization of EEG pathology”. In: (2017).

[55] D. Zhang et al. “EEG-based Intention Recognition from Spatio-Temporal Representations via Cascade and Parallel Convolutional Recurrent Neural Networks”. In: (2017).

[56] Md.K. Islam, A. Rastegarnia, and Z. Yang. “Methodsfor Artifact Detection and Removal from Scalp EEG: A Review”. In: *Clinical Neurophysiology* (2016).

[57] Z. Tayeb et al. “Validating Deep Neural Networks for Online Decoding of Motor Imagery Movements from EEG Signals”. In: *Sensors* (2019).

Online resources

[1] Brain Key. *Neuron structure*. 2013. URL: http://brain-key.com/the-science/our-brain/.

[2] Wikipedia. *Action potential*. 2007. URL: https://en.wikipedia.org/wiki/Action_potential (visited on 08/01/2019).

[3] Wikipedia. *Neural oscillations*. 2010. URL: https://en.wikipedia.org/wiki/Neural_oscillation (visited on 08/01/2019).

[4] Brain Connection. *The anatomy of movement*. 2013. URL: https://brainconnection.brainhq.com/2013/03/05/the-anatomy-of-movement/ (visited on 08/01/2019).

[5] Wikipedia. *Wave patterns*. 2005. URL: https://en.wikipedia.org/wiki/Electroencephalography#Wave_patterns (visited on 08/01/2019).

[7] Brain Products. *actiCAP*. 2017. URL: https://pressrelease.brainproducts.com/active-electrodes-walkthrough/ (visited on 08/02/2019).

[8] V. Asanza. *IEEE ITCM*. 2017. URL: https://www.academia.edu/35623849/EEG_Signal_Clustering_for_Motor_and_Imaginary_Motor_Tasks_on_Hands_and_Feet?auto=download (visited on 08/01/2019).

[9] Massachusetts General Hospital. *Advanced EEG analysis reveals the complex beauty of the sleeping brain*. 2017. URL: https://www.massgeneral.org/News/pressrelease.aspx?id=2055 (visited on 08/01/2019).

[13] Wikipedia. *Artificial neural network*. 2013. URL: https://en.wikipedia.org/wiki/Artificial_neural_network (visited on 08/07/2019).

[14] A. Karpathy Stanford. *CNN for Visual Recognition*. 2019. URL: http://cs231n.github.io/convolutional-networks/ (visited on 08/07/2019).

[15] Wikipedia. *Matrix calculus*. 2019. URL: https://en.wikipedia.org/wiki/Matrix_calculus (visited on 08/20/2019).

[17] K. Kurita. *Loss curves*. 2018. URL: http://mlexplained.com/2018/04/24/overfitting-isnt-simple-overfitting-re-explained-with-priors-biases-and-no-free-lunch/ (visited on 08/14/2019).

[18] ETH Zürich. *Cybathlon logo*. 2019. URL: https://cybathlon.ethz.ch/ (visited on 08/01/2019).

[19] Nicola Pitaro ETH Zürich. *Team Athena-Minerva GER*. 2016. URL: https://cybathlon.ethz.ch/races-and-disciplines/bci-race.html (visited on 08/01/2019).

[23] IEEE Spectrum. *Video Game Racing with Your Brain*. 2016. URL: https://www.youtube.com/watch?v=4FN-pyQpXO8 (visited on 08/31/2019).

[24] S.W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. 1997. URL: http://www.dspguide.com/pdfbook.htm (visited on 08/31/2019).

[26] Wikipedia. *actiCAP*. 2008. URL: https://en.wikipedia.org/wiki/Event-related_potential (visited on 08/05/2019).

[38] M. Billinger MNE. *Decoding CSP EEG*. 2015. URL: https://www.martinos.org/mne/stable/auto_examples/decoding/plot_decoding_csp_eeg.html?highlight=csp\%20plot_pattern (visited on 08/06/2019).

[40] G. Wong Github. *VGG-16 net*. 2019. URL: https://github.com/gabrielwong159/tf (visited on 08/07/2019).

[42] T. Bluche. *Convolutional Neural Network*. 2017. URL: http://technodocbox.com/3D_Graphics/70716176-Deep-neural-networks-applications-in-handwriting-recognition.html (visited on 08/07/2019).

[43] C. Olah. *Understanding LSTMs*. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 08/09/2019).

[50] Wikipedia. *SVM margin*. 2018. URL: https://en.wikipedia.org/wiki/Support-vector_machine (visited on 08/27/2019).

[52] J. Weston. *Support Vector Machine Tutorial*. 2006. URL: http://www.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf (visited on 08/27/2019).

[53] S. Raschka. *Linear Discriminant Analysis*. 2014. URL: https://sebastianraschka.com/Articles/2014_python_lda.html#principal-component-analysis-vs-linear-discriminant-analysis (visited on 08/27/2019).