# POLYTECHNIC OF TURIN

Master's Degree in ICT for Smart Societies



Master's Degree Thesis

# Brain-computer interface for bionic prosthetic arm actuation

Relator and Supervisor

Prof. Monica VISINTIN

Prof. Guido PAGANA

External Supervisors

Prof. Robin AUGUSTINE

Prof. Mauricio PEREZ

Candidate

Federico FABIANI

October 2021

# Summary

Recent progress in neurobiology goes hand to hand with the development of artificial intelligence. The latter allows decoding the complex patterns underlying the brain activity recorded by new generation implanted electrodes. This combination represents the turning point in the development of neuroprosthetic. In the past decades, we witnessed advancements in this field like paralytics patients driving wheelchairs by visual stimulation, monkeys controlling robotic arms, or patients moving a cursor with their minds to write on a virtual keyboard. However, no project ever tried to be as ambitious as some are being now, such as *Neuralink* or *B-Cratos*, that aim at pushing the concept of the closed-loop neural interface to a new limit. The challenge they state for the next decade is to exploit intra-cranial EEG (with their low signal to noise ratio) and advanced neural networks to reach *fine* controlling of external devices and to stimulate sensory *feedback* response in the patient brain.

In this paper, I am investigating the possibility of controlling a prototype prosthetic arm to replicate the movement performed by a macaque monkey during a grasping trial composed of several phases (wait, cue, planning, moving, holding). Every grasping trial had as a target one out of thirty-six objects, organised in six shapes and six sizes. Researchers of *German Primate Centre* directed the experiment, during which they recorded the brain activity of the animal using implanted microelectrodes arrays. Next, they applied offline data processing known as *spike sorting*, clustering raw electrical activity to identify individual neurons activations. The first step of my work consisted in comparing several network archetypes (*Feed-Forward Networks*, *Convolutional Networks* and *Recurrent Networks*) on the task of classifying the target object from a part of the neural activity. For doing so, I automatically optimised every network with the support of the *Hyperopt* library: this is a tool that supports optimisation in Python, drawing pseudo-random parameters configurations from a defined search space while accounting also previous results. Such an optimisation history also allowed me to investigate which parameters were most correlated to good classification results. Eventually, I assessed that the recurrent ones were those performing the better and, in particular, the idea to have a convolutional layer extracting spatial

features and one successive *LSTM* or *GRU* layer to weigh temporal dependencies was successful.

Secondly, based on this optimised network, I wrote the brain-computer interface software that I used in a proof of concept of low-delay decoding. In the demonstration, I simulated a real-time neural recording, buffering the last ten measurements from which the BCI decoded the current state of the experiment and the object shape and size. Then, a proper actuation signal was composed and sent to the arm via radio transmission.

The overall results are promising, showing that the different objects and trial phases shape the neural activity in such a unique way that makes it possible to classify them, with the size being the trickiest to decode. The respective testing accuracy for state, shape and size decoding tasks are around 90%, 92% and 25%. However, overfitting was a constant problem in the development of the BCI. Therefore, building a deeper and more complex network and train it on more examples might be a necessary future step.

# Table of Contents

# List of Figures

# Acronyms

**AI**
artificial intelligence

**ML**
machine learning

**NN**
neural network

**DL**
deep learning

**BCI**
brain computer interface

**EEG**
electroencephalography

**EMG**
electromyography

**MI**
motor imagery

**iEEG**
intracranial EEG

**MEMS**
microelectromechanical systems

**FFT**

fast fourier transformation

**CWT**

continuous wavelet transformation

**FMA**

floating microelectrode arrays

**PCA**

principal components analysis

**FC**

fully connected

**DNN**

dense neural network

**CNN**

convolutional neural network

**RNN**

recurrent neural network

**GRU**

gated recurrent unit

**LSTM**

long short-term memory

**TPE**

tree of parzen estimators

# Chapter 1

# Introduction

The research fields of *Artificial Intelligence* (AI) and *neuroscience* shared a mutually beneficial relationship since their origin. In the early stages of the evolution of *Machine Learning* (ML), neurobiology inspired to build *Neural Networks* (NN) that mimic brain structure under two aspects [1]. Firstly, similarly to the human brain, the NN consist of interconnected neurons that receive inputs, get activated and send information to other neurons. Secondly, as humans learn new things creating and strengthening the connection between neurons, in a NN, each connection is associated with a weight that will be tuned during the so-called training phase to reflect the strength of that specific interaction.

The flip side of the relationship, sees nowadays AI boosting the advancement in neuroscience [2]. The ML main strength lies in the ability to recognise patterns buried under the data and this is especially crucial when it comes to analysing the information obtained from human minds. Actually, despite the brain recordings are complex, described by many dimensions and requiring a lot of features extraction, with the advancement of ML, neuroscientists are hacking them and discovering how billions of brain neurons work together.

In recent years, ML found its evolution in the *Deep Learning* (DL) (i.e. ML based on *deep*-NNs) that has replaced the traditional approaches in most of the fields. Among all the differences, the most important one is the ability of deep NN to extract features from raw (or lightly pre-processed) high dimension data, while, on the contrary, ML algorithms rely on handcrafted features. This transformation improved the research in many fields, particularly in computer vision and natural language processing. Now the same evolution is found in the task of decoding the brain activity (with high variability and non-stationary noise) into a meaningful signal. DL models with a high number of parameters seem promising to learn from raw *Electroencephalogram* (EEG) electrical measurements and to extract better information to improve performance and robustness.

This possibility was initially an intuition of the professor *Jacques Vidal*, from

*University of California, Los Angeles*, that coined the term *Brain Computer Interface* (BCI, or sometimes *Brain Machine Interface*-BMI) and stated the "BCI challenge": control of external objects or devices using neural signals [3]. BCIs technology is strictly related then to *neuroprosthetics*, subject with which it shares aims, such as restoring sight, hearing, movement, ability to communicate and even cognitive function [4]. The first application was from the same author in 1977 when a cursor was moved through *Visual Evoked Potentials* (i.e. induction of higher brain activity projecting specific images, light or colours) [5]. Since then, we witnessed huge advancements in both hardware (electrodes) and software (neural networks), having some recent examples of robotic arms actuated via EEG [6] or *Electromyography* (EMG) (i.e. the electrical activity of peripheral nervous system recorded on the limb) [7, 8]. Now, the next step seems to be found in the integration of invasive techniques [9], with projects that aim to be milestones in neuroprosthetics such as *Neuralink* [10] and *B-Cratos* [11].

Challenges for these ambitious projects include:

- Designing a fully implantable recording device not rejected by the organism and not (or slowly) degrading in quality of time
- Further developing real-time computational algorithms
- Introducing a bi-directional communication, for stimulating the brain with sensory feedback from the actuators beside the motor signal translated and sent from the brain to the actuators
- Designing and building prostheses that can be controlled directly by brain-decoded signals

By reaching these milestones, future BCIs will be able to drive and control revolutionary prostheses that feel and act like the human arm.

## 1.1 Background

In this section, I will analyse the background in which BCI technologies are developed, starting from their definition and origin.

### 1.1.1 Brain-computer interface

Neurophysiological signals are complex, as they result from the composition of a multitude of parameters. Hence, even well-trained specialists spend a considerable amount of time analysing them to perform diagnoses. BCI is the technology that comes in aid to automate such a process and possibly to deliver much more accurate results. This kind of system relies on NNs and exploits their ability to recognise

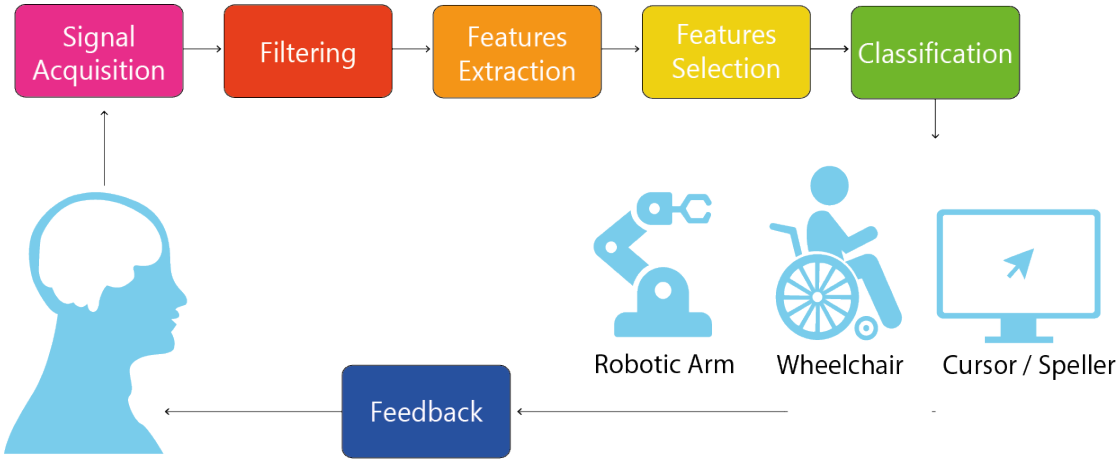



**Figure 1:** BCI flowchart

patterns to extract intent from the brain activity of a subject (humans or animals) and translate it into messages appropriate for machines [12].

Nowadays, BCIs are intensively investigated by researchers because of all the promising results obtained in the past decade [13, 14]. Among the others, some of the most remarkable studies got a monkey controlling robotic arms to feed itself [15], a patient handling a wheelchair with their thoughts [16] and moving a cursor on a virtual keyboard to type about eight words per minute [17]. However, applications are not only those in which the intent is translated into a command for an external device, for instance, since some other regards: automatic analysis of drowsiness and sleeping patterns [18], short-term epilepsy prognosis [19] or monitoring of attention deficit hyperactivity disorder [20], mental workload [21], emotional involvement while entertained [22] and many more. BCI based applications are virtually unlimited.

Overall, all the BCI systems have in common a structure composed of six phases: (**i**) collecting data, (**ii**) *optional*: preprocessing the raw data to remove as much background noise as possible, (**iii**) extracting specific features from the clean version of data depending on the kind of application, (**iv**) *optional*: selecting the most salient features from the extracted ones, (**v**) classifying the obtained features to decode an intent and use it depending on the application, and (**vi**) eventually providing feedback to the user (both physically or visually) [23]. Figure 1 displays the decoding flow chart of a BCI system. Points (**ii**) and (**iv**) are optional since manual features manipulation application depends on technology used and researchers preferences.

### 1.1.2 Neuroprosthetics

Neuroprosthetics refers to a variety of artificial devices or systems that can be used to enhance compromised motor, sensory, cognitive, visual, auditory, and communicative abilities. Those deficits might arise from spinal cord injuries, motor diseases or other pathologies that disconnect the nervous system from the brain.[24].

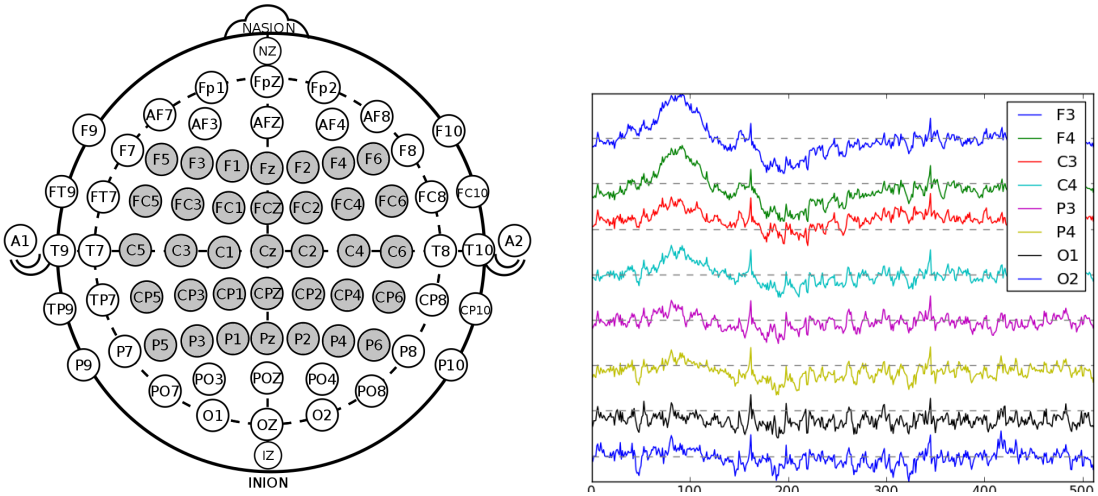
Because of its characteristics, BCI represents the enabling technology for a new generation of neuroprosthetics, since paralysed patients still manifest brain activity related to the intention of acting, even if the signal never reaches the peripheral nerves. Therefore, for these subjects, myoelectric prostheses are not applicable, because they depend on activated nerves in limbs, but the brain activity could be directly accessed and interfaced with assisting devices [25].

In most cases, such technologies are based on *Motor Imagery* (MI) neural data. With the MI technique, the imagination of acting causes a variation of neuron activation in the motor cortex area which is afterwards translated into electrical signals. This signal then can be intercepted and measured with electrodes, both externally on the scalp or internally, if they are implanted inside the cortex. MI proved to apply to a different part of the body [26] and this rise aspiration to design a reliable decoder for several actions.

### 1.1.3 MI EEG recordings

Electroencephalography is a monitoring technique where electrodes are placed on the scalp surface to capture physiological activities of the brain underneath [27]. The conducting electrodes are placed on the scalp according to the well-known *10–20 international placement system* (reported in Figure 2a) and each one of them records a one-dimensional vector of raw EEG data. An example of the recordings from multiple electrodes is shown in Figure 2b. The electrical signals are the product of the postsynaptic potentials in cortical neurons, reaching the scalp surface through volume conduction across multiple brain tissue. Hence, EEG signals are prone to be weak, non-stationary, and characterised by a low signal-to-noise ratio. Moreover, they are often affected by *artifacts* (i.e. signals recorded by EEG but not generated by the brain) such as the power cable of the recording device and electrode displacement. Often the undesired background noise is addressed through hand-engineered features selection via traditional signal processing techniques.

Besides the drawbacks intrinsic to EEG, even more, issues arise when recording MI-based signals. Firstly, signals related to different parts of the body like the eyes, head, neck, or any other muscle, will be mixed with the imagined action. Secondly, MI EEG signals are highly variable due to physiological and psychological features for each person at each time, meaning that recordings of the same task may differ from subject to subject and from session to session for a particular subject.




**(a)** **(b)**

**Figure 2:** International 10-20 system for placement of MI EEG electrodes (a) and example of EEG record (b)

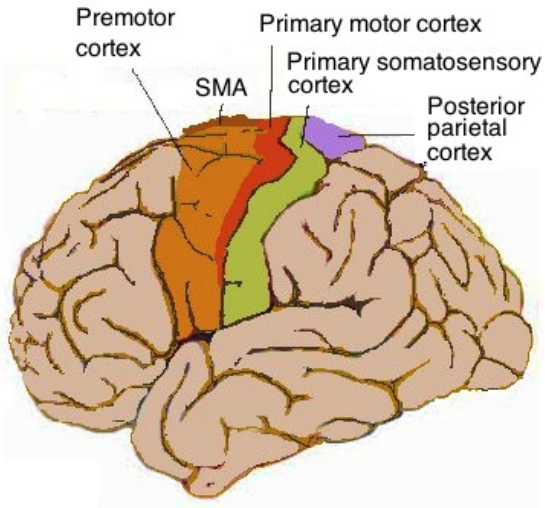



**Figure 3:** Human motor cortex [28]

On the other hand, MI EEG brings some advantages over other neurological acquisition techniques, is characterised by: (**1**) low cost of the hardware, often composed by a wearable helmet and a variable number of electrodes, (**2**) portability, and (**3**) no side effects thanks to its non-invasive approach.

It is worth mentioning some of the technical aspects of MI EEG recording. For what concern the placement of the sensors, the electrodes used for MI recording are chosen among the grey ones in the Figure 2a [29]. This is due to the central area

of the brain cortex being recognised as the motor cortex, as pictured in Figure 3. It is in this area where the brain activity related to the planning of a movement and its execution is generated and propagated from [28].

Once the electrical activity is recorded, the amplitude and frequency values in EEG signals are used for discriminating various physiological activities. The amplitude is normally fluctuating in micro-volts and it is the main feature used for classification, while the frequency range can be split into several bands, of which the most popular for MI analysis are the *mu* ($9 - 13\,\text{Hz}$), the *alpha* ($8 - 12\,\text{Hz}$), the *beta* ($13 - 30\,\text{Hz}$), and the *gamma* ($> 30\,\text{Hz}$) frequency bands [30].

Due to their drawbacks, in particular being dependent on the subject state of mind, the following key characteristics are to be annotated to any MI EEG datasets: (**i**) number and type of MI tasks, (**ii**) number of EEG channels, (**iii**) number of participated volunteers, (**iv**) number of sessions accomplished with each volunteer, (**v**) number of trials within a session, (**vi**) length of a trial, and (**vii**) the period between two successive sessions.

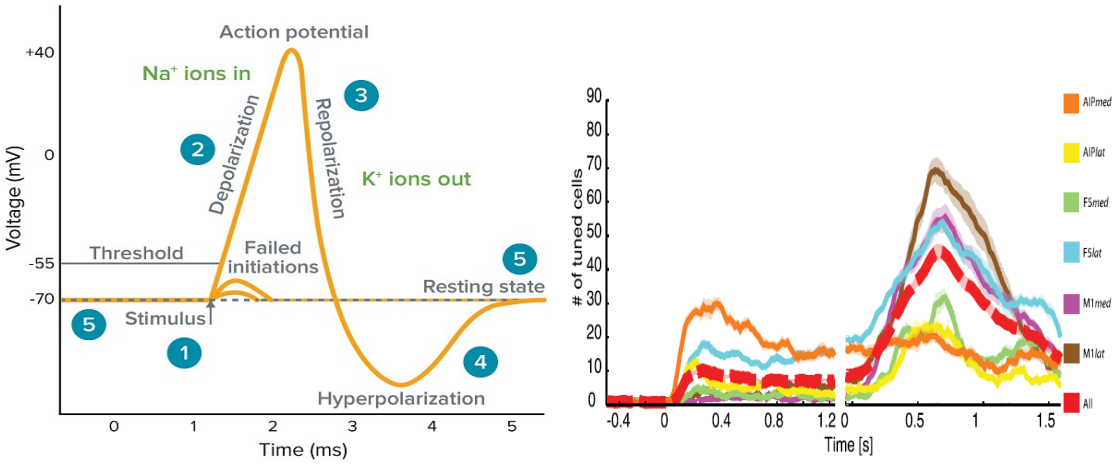
### 1.1.4 Intracranial EEG recordings

*Intracranial EEG* (iEEG) is an invasive recording technique, that requires electrodes to be implanted inside the brain, at depth depending on the application. This monitoring technique is a good answer to the limits of EEG systems: recording the electrical activity directly below the surface of the scalp allows to obtain a higher signal to noise ratio (and a higher quality signal in general) since no shading was performed from the skull, skin and hairs. Moreover, the recorded signal is less likely to be affected by artifacts. Recent developments in neural interfaces show that it is possible to have fine control of a robotic prosthetic by interfacing directly with the motor cortex of the human brain [31].

However, this advantage in quality does not come for free. The first constraint regards the development of suitable hardware, that should be long lasting once implanted while at present time, not constant results were obtained. Secondly, the effects of damage from implantation are still inconclusive and immune responses remain a problem for long-term use [31]. Some recent researchers tried bio-active molecules and bio-compatible materials, to prevent immune responses, but still, more controlled study is needed before intracortical systems become widespread. Overall, given the uncertain performance and the risks related to human testing, this technology must still prove to be way better than less invasive solutions.

The study of neuronal activity still lack a systematic theory, and mostly the science is proceeding by independent attempt. In general, the intracortical systems can be distinguished in *bundle of wire systems* or *Microelectromechanical Systems* (MEMS). MEMS usually, refer to microscopic smart devices, build with conductive and semiconductive materials, that combine electronic, biologic, chemical or

mechanical functions in a highly constrained place [32]. Both technology show promise, but MEMS have been popularised for not only enabling the study of the single neuronal unit but also to scale to the large population [33]. In particular, more complex intracortical systems are built on top of the Michigan probe and Utah microarray.




**(a)** **(b)**

**Figure 4:** Neuron action potential diagram (a) and example of collection of spikes (b)

Such implanted systems, if placed sufficiently close to a neuron can detect and record the potential changes that happen during its activation. This is possible because during that event, the usually constant voltage difference across the plasma membrane of around $-70\,\text{mV}$, called *membrane potentials*, temporary and rapidly changes to $\sim 30\,\text{mV}$. Synapses stimulation brings the neuronal cell to *depolarise*. Once this depolarisation overcomes a neuron-specific threshold, an *action potential* is triggered, during which, the membrane potential rapidly increases and then suddenly drops back down in a spike-like waveform (see Figure 4a). This waveform is typically fairly consistent within a cell throughout activation potentials. This sudden variation then triggers synapses in the next connected neuron in a process that will be propagated through the whole nervous system. Moreover, it will results in a measurable extracellular current due to the ion flow through the sodium and potassium channels of the cell, that is what an electrode usually measures [31].

However, measuring the activity of individual neurons accurately can be difficult due to large amounts of background noise and the difficulty in distinguishing the action potentials of one neuron from the other neighbour. The task of detecting and classifying single action potentials is commonly referred to as *spike sorting* and many algorithms and tools have been developed over the years [34]. In general, the key concept is found in the previously mentioned cell-specific neural spike waveform. Exploiting this assumption, and given the measurements of many electrodes in a

close area, it is possible to recognise unique *spikes*, and then clustering them to associate each wave shape to unique neurons, even when spikes overlap. As a result of such processing, is obtained a collection of spikes for each sensed neuron.

## 1.2 Related Work

In this section, I will discuss the recent progress in the BCI research field. The first point of the discussion will address the non-invasive techniques, for which a huge number of works is available, and I will make use of two systematic reviews to try to put an order in such heterogeneous literature. The first one provides an overview of non-invasive BCI, and in particular, proposes a way to classify works. The second one focuses only on the MI applications and draw conclusions. Secondly, I will discuss the few pieces of literature available for invasive BCI and once more, I will use a deep networks comparison to help to understand the current most promising approaches.

### 1.2.1 Non-invasive BCIs

The first high-level discrimination for BCIs is whether their make use of non-invasive (EEG based) or invasive (iEEG based) recording methods [23].

Literature provides a multitude of non-invasive examples, being EEG a largely tested and accessible technique. Google scholar counts more than 2000 works only in 2020 when searching for "non-invasive BCI". Despite this huge availability, it is not easy to delineate a clear state-of-the-art: every possible technology sooner or later was tested on the neural decoding task.

To help navigating such a heterogeneous literature I am supporting this discussion with the review performed in 2019 by researchers from "*Noninvasive Brain–Machine Interface System Laboratory*", *University of Houston* [14]. In their systematic review they propose a categorisation of BCIs based on: (**i**) Task details, (**ii**) Artifact removal strategy, (**iii**) Frequency included in the analysis and input formulation, (**iv**) Deep learning strategy, and (**v**) Highest achieved accuracy. Some relevant examples are reported in Table 1.

**Table 1:** Some exemplar recent works related to EEG based BCI

| Title and Authors | Summary |
|---|---|
| *A novel MI-EEG imaging with the location information of electrodes* [35] M.A. Li, J.F. Han, and L.J. Duan | They extracted mu and beta frequency bands with FFT. The experiment is divided into short sections and for each of them, a 2D RGB image is generated, mapping the average power sensed by one electrode to its location, and interpolating. Classification is performed with transfer learning based on VGG-16 pre-trained parameters, and The decoding task scored a 10-fold cross-validation accuracy of 92.13% (4 classes) and 96.82% (2 classes). |
| *Multi-class classification of motor imagery EEG signals using image-based deep recurrent convolutional neural network* [36] W. Fadel, C. Kollod, M. Wahdow, Y. Ibrahim, and I. Ulbert | Energy distribution was mapped into 2D images using azimuthal equidistant projection (AEP) of electrodes. They filtered mu, beta and delta bands. The decoder was a composition of CNN (inherited from VGG) and an LSTM layer. The accuracy on the 5 classes Physionet dataset MI EEG was ~70% |
| *Application of continuous wavelet transform and CNN in decoding motor imagery brain-computer interface* [37] H.K. Lee and Y.-S. Choi | They used the continuous wavelet transform (CWT) to construct 2D images for training a CNN model. The filters of the convolutional layer are long as the number of channels, to reduce spatial dimension to 1 |
| *EEG Classification with Transformer Based Models* [38] J. Sun, J. Xie, and H. Zhou | Tried to extract spatial or temporal features applying transformers on one dimension after CNN extracted features on the other one. On the Physionet dataset, they obtained 68% accuracy on 4 classes |
| *An end-to-end deep learning approach to MI-EEG signal classification forBCIs* [39] H. Dose, J.S. Møller, H.K. Iversen, and S. Puthusserypady | CNN applied to raw EEG signals. It consists of a temporal and spatial convolution layer for feature extraction and a fully connected layer for classification. The classifier reaches 80.10%, 69.72%, and 59.71% mean accuracy using Physionet dataset with two, three, and four classes, respectively, validated with 5-fold cross-validation |

Continued on next page

Table 1 – continued from previous page

| Title and Authors | Summary |
|---|---|
| *A CNN-LSTM deep Learning classifier for motor imagery EEG detection using a low-invasive and low-Cost BCI head-band* [40] F.M. G. Moreno, M. B. Edo, M.J. R. Fórtiz, and J. L. Garrido | They used a composition of CNN and LSTM to classify 2 classes EEG for 4 subjects with an accuracy of 86%. The best performances were obtained with a 3-sec long window and with manual filtering of alpha, beta, theta, delta and gamma waves. Without the preprocessing, results drastically decreased |
| *A deep learning scheme for motor imagery classification based on restricted Boltzmann machines* [41] N. Lu, T. Li, X. Ren, and H. Miao | EEG recordings is processed in frequency domain using FFT and WPD separately for comparison purposes. Then deep belief network, composed of restricted Boltzmann machines is trained using the frequency data. The use of FFT led to a better performance than that of WPD and the frequency preprocessing proved to positively affect results |
| *A novel deep learning approach for classification of EEG motor imagery signals* [42] Y. R. Tabar and U. Halici | They propose a deep network composed of two other stacked: a CNN to extract the features and SAE for classification. The training was performed on 2D images, obtained mapping the frequency bands power on the location of the electrodes. The frequency filtering was performed with the short-time Fourier transform (STFT) |

Summarising, we can conclude that in this field, similarly to many others, the recent focus is on deep NN based applications rather than on traditional ML ones. Plenty of studies have shown the capability of deep NNs in analysing EEG MI signals, and to cope with the so said "curse of dimensionality" problem. This is arising from the non-stationarity of signals, multi-channel recording paradigm, channel correlation, and the existence of noise and artefacts.

Other common findings include: most of the study performed a reduction of frequency range under analysis between 10 and 40Hz; a huge majority ignored the artefacts while only some manually removed it; most of the researches manually calculated features, some directly work on raw measurements and only a few transformed signals into images.

The strategy used to calculate features are many and mostly repeated in only one work. Some of the most common are: FFT: fast Fourier transform, MAD: mean absolute difference, PSD: power spectral density, STFT: short-time Fourier transformation, SVD: singular value decomposition, SWD: swarm decomposition

From all the above-mentioned studies, only a few regards motor imagery. The

work of the researchers from *University of Mosul* [26] represents at the moment the only deep literature review on the use of deep NNs for classification of MI EEG data. They focused on three aspects: (**1**) Identify which deep neural network architecture is best suited for the classification process, (**2**) which structure of input data has a more positive impact on deep learning and (**3**) what frequency range must be considered during the analysis. Their findings can be summarised as follows:

- Most of the projects filtered frequencies between 5 and 40Hz
- Popular regularisation for the model was dropout and batch normalisation
- Most used datasets are BCI Competition IV 2a and 2b (4 and 2 classes)
- As input, raw time series, images and calculated features, was proposed the same way
- The features were calculated in many unique approaches, with few repetitions of BCSP-Filter Bank Common Spatial- and CSP
- The images were generated in many unique ways, with few repetitions of CWT.
- The most trendy network, used in the 3/4 of works was CNN, sometimes stacked with LSTM, SAE or VAE.
- The CNN was mainly activated with ReLU 45% and ELU 19%
- The majority of optimisation algorithms was Adam 47% or SGD 26%

Overall Deep CNN architecture with the ReLU activation function is found to be the most effective architecture for the classification task of MI EEG. DL has the capability of exploiting the whole input data for training networks, hence, raw EEG data could be reasonably used in the training process. Moreover, it is observed that the classification of MI records that are related to a single limb, e.g. tasks of different fingers, are more challenging than the classification of tasks that are related to different limbs. This is because of the high similarity of the signals propagated from the same area. It is worth mentioning that a large number of hidden layers does not always lead to better performance; in contrast, it may cause the network to have an overfitting problem, as well as increase the computational complexity.

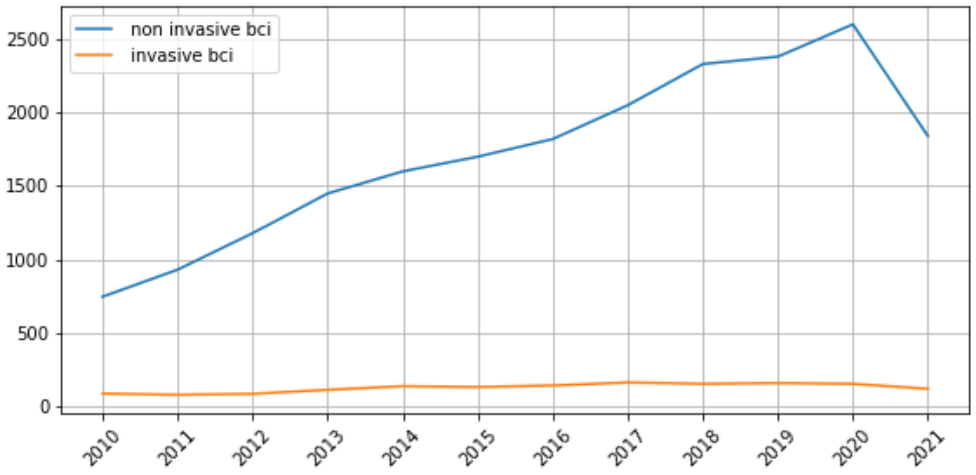



**Figure 5:** Number of results for a query on google scholar about invasive and non-invasive BCI over years

### 1.2.2 Invasive BCIs

Invasive BCI approaches are based on recordings of single or multiple neurons, acquired in close proximity of the brain cells. The run to develop brain decoders based on this technique started in 1980 when *Edward Schmidt* raised the possibility that voluntary motor commands could be extracted from raw cortical neural activity and used to control a prosthetic device designed to restore motor functions in severely paralysed patients [43]. Those experiments proved that monkeys, encouraged by a feedback reward, managed to learn how to control their cortical neurons voluntarily. Despite being investigated for such a long time, this kind of application is still not diffuse, and the few experiments are mainly performed on animals. That is due to the complexity and the uncertain outcome of the delicate brain surgery that is necessary to prepare a subject to the experiments [23].

Moreover, the research in this field is still focused more on the biology behaviour analysis [44], on the hardware development or to assess the long term stability and safety of invasive application on humans. To provide an idea of the shortage of research in science about deep NN applied to iEEG, in Figure 5 are reported the different number of results when searching for "non+invasive+bci" and "-non+invasive+bci" over the last decade. Among those few words, it is hard to find researches targeting the same dataset or the same task, making it hard to state a clear state of the art. However, a good support is provided by researchers from *university of Chicago*, *Pennsylvania* and *Columbia* [45]. In 2020 they performed a deep comparison of different deep NNs and ML models applied on three distinct datasets. Each dataset was composed of a collection of spikes collected from neurons

located in the somatosensory cortex, hippocampus and motor cortex. The latter is of interest for this discussion, being the area where the MI brain activity is originated. In the task for decoding from the motor cortex, monkeys moved a manipulandum (a level like object, with multiple degrees of freedom) that controlled a cursor on a screen [46]. The dataset was composed of the recording from 164 neurons, and the 21 minutes recording was time binned in steps of 50ms during which the number of activation was counted.

They proved that on the task of regressing the cursor position and velocity, deep NNs outperforms traditional ML algorithms, such as Kalman Filter, Naive Bayes, Wiener Cascade, Extreme Gradient Boost and Support Vector Machine. In particular, LSTM was the network that performed the best among the simple ones, as its performance was slightly exceeded by an ensemble model. The ensemble was a complex model, that combined the outcome of all the other classifiers, and presented it to a further fully connected network for obtaining the final output. Another relevant finding is that the neural networks exceeded expectations in a context with a limited amount of data. The explanation the researchers propose for these good results is found in the size of the networks. On average, the tested deep NNs had a number of parameters in the order of $10^5$, while common networks for image classification such as the famous VGG16 can have up to $10^8$ parameters [47]. Thus, the relatively smaller size of their networks (counting hundreds of hidden units) may have allowed for excellent prediction with limited data [48]. Moreover, the fact that the tasks under analysis had a low-dimensional structure, and therefore the neural data were also likely low dimensional, could have contributed to increasing the decoding accuracy [49].

## 1.3 Intent of this project

Thank the ongoing collaboration between Microwave for Medical Engineering Group, Uppsala University and German Primate Center in the context of the B-Cratos project [11] I had access to an exclusive dataset. It is composed of the neural activity recording from the motor cortex of a monkey while performing a grasping task, used in the context of a previous article [9].

The purpose of my work is to develop a full working demonstration, starting from neural data analysis, network building and training, and real-time decoding to actuate a prototype 3D printed prosthetic arm.

At the same time, I want to contribute to compensate for the lack of literature about deep NN applied to neuron spikes dataset, offering a deep comparison of different networks and hyperparameters configurations, on this dataset. Since only a few pieces of literature regarding invasive BCI claimed the superiority of deep NN, this is what be the focus of the research, and some approaches that were

well-performing on non-invasive BCI will be included in the analysis.

This manuscript will be organised as follows: in chapter 2 I will present an in-depth explanation of the dataset, as well as the experiment set up to collect it, and the preprocessing performed on it. In chapter 3 I will present a deep comparison of many deep NNs tested on a smaller portion of the dataset, with a particular focus on the hyper-parameters role in the classification accuracy. Lastly in chapter 4 starting from taking the network that performed the best in the offline analysis, I will present the procedure that brought to the development of the brain-computer interface for online the demonstration on a prototype prosthetic arm reported in chapter 5.

# Chapter 2

# Dataset

## 2.1 Experiment description

The dataset used for this discussion was generated by researchers of *German Primate Centre* in 2015, and firstly described in their article "*Decoding a Wide Range of Hand Configurations from Macaque Motor, Premotor, and Parietal Cortices*a" [9]. In their experiment, two macaque monkeys, a male (called subject M) and a female (subject Z) were trained to perform a sequence of steps to grasp one object presented to them, while both hand kinematic and brain activity were monitored and recorded.

At the beginning of each session of trials the set up displayed in Figure 6a took place: the monkey was sitting in a primate chair, with its head fixed and its hand laying on a support, wearing a custom kinematic glove necessary to record the real time position of the hand, fingers and wrist. A rotating turntable was placed in front of the animal 25 cm far from its chest. It was used to quickly change the object presented between trials. Eight different turn tables were available, and the set up allowed to switch among them in less than one minute. In Figure 6b is represented the content of every turntable: six of them had objects with the same shape but increasing sizes; one had mixed object with mid size and the last had special uniquely shaped objects. This variety was designed to acquire a huge variation of grips type. The main shapes are rings, cubes, spheres, horizontal cylinders, boxes and vertical cylinders. Alternatively, the monkey could have been presented with an handle and be required to perform a strength or precision grasp on it (Figure 6d).

The experiment session lasted around 90 minutes and it counted many trials, as described in Figure 6c. While in complete darkness, the animal could initiate a trial by pressing a button near its chest. Then, it had to fixate a red LED light for a variable time (*fixation epoch*), with duration between 500-800 ms (mean 650

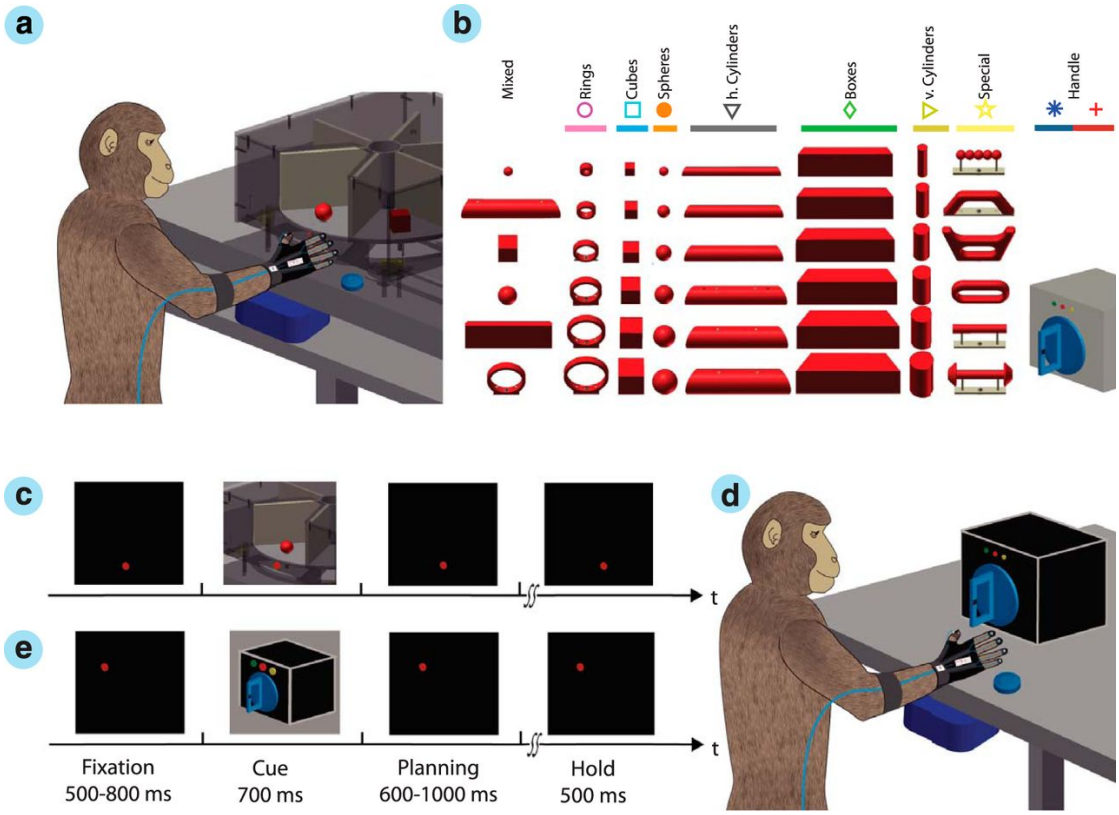



**Figure 6:** Standard experimental task set up (a) and special precision/strength one (d); list of 50 target objects (b); sequence of standard experiment phases (c) and special precision/strength ones (e) [9]

ms). Then a spotlight illuminated the graspable object for 700 ms (*cue epoch*). When the light faded, the animal had to wait another random delay of 600-1000 ms, until the fixation LED blinked (*planning epoch*) and only after it could execute the movement and lift the object (*movement epoch*). After holding it for 500 ms (*hold epoch*) the monkey received some juice as reward. In case of a special task, the handle was placed instead of the turning table in front of the animal, and it would recognise it during the cue epoch (Figure 6e). Error trials were immediately aborted without providing reward. Later in this section, the list of epochs will be longer, including many sub-phases as well as the substitution and the operation of the turntable. A comprehensive description for all of them is provided in Appendix A.

For the whole duration of the trial, all the behavioural and task-relevant parameters were controlled and recorded, by sensors and cameras. In particular the hand kinematic was monitored using a custom developed instrumented glove for small primates. This kinematic tracking device is based on an electromagnetic tracking system (WAVE) and consists of seven sensors coils that are placed on all fingertips,

the back of the hand and at the lower fore harm just proximal to the wrist. The electromagnetic sensors were tracked also when partially occluded, because they did not depend on line of sight to a camera, but only on the presence of inductive metal (reason why all the instruments avoided such material as much as possible). From this configuration was possible to reconstruct offline the whole kinematic description of the arm and hand, including 18 joints and 27 degrees of freedom.

Concerning the brain monitoring, the recording were obtained with six surgery implanted *Floating Microelectrode Arrays* (FMAs) producted by *MicroProbes for Life Science* (Figure 7a). Specifically, two FMAs were inserted in areas AIP, F5, and M1 as represented in picture Figure 7b, taken during the surgery of the female monkey. Schematic placement of all the FMAs are represented in Figure 7c for female (Z) monkey and Figure 7d for male one. Each FMA consisted of 32 non-moveable monopolar platinum–iridium electrodes. From the implanted electrode arrays, the researchers recorded spiking activity (single units and multiunits) simultaneously from a total population of 192 electrodes in AIP, F5, and M1. Neural activity was sampled at a rate of 24 kHz with a resolution of 16 bit and stored to disk together with behavioral data and hand and arm kinematics using a *RZ2 Biosignal Processor*.

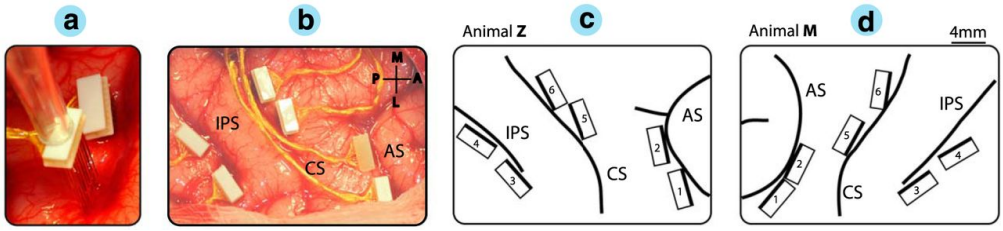



**Figure 7:** Implantation of 32 individual electrodes (a) and their placement in the bank of the sulcus (b); schematics of FMA placemets of animal Z (c) and animal M (d) [9]

The raw measurements were processed offline with two spike sorting algorithms. First processing is performed with WaveClus [50] for automatic sorting and secondly with OfflineSorter (Plexon) for subsequent manual resorting. Thanks to this procedure the classification of neurons was automatized and as well as an additional evaluation of cluster quality with respect to signal stability and interspike interval histograms.

## 2.2 Classification dataset

From the authors of the work explained in the previous section, I received the pre-sorted recordings of 759 trials (a whole session) of the male monkey, each one labelled with the target object to grasp. Starting from it I generated the

classification dataset used for this project. Data were provided in Neo format: a data format specifically designed for representing electrophysiology recordings [51]. Such format of data is easily explorable with *Electro Physiology Analysis Tool* (Elephant) python module.

Of all the 759 trials, 131 of were aborted for errors of the animal, leaving 628 full measurements, associated to the 50 possible objects (Figure 6b). As previously mentioned, the recordings were already pre-processed with spike sorting algorithms, that well-isolated 552 firing neurons. With this data format, every neural unit activity is represented by a *SpikeTrains* element: a collection of timestamps when that neuron fired.

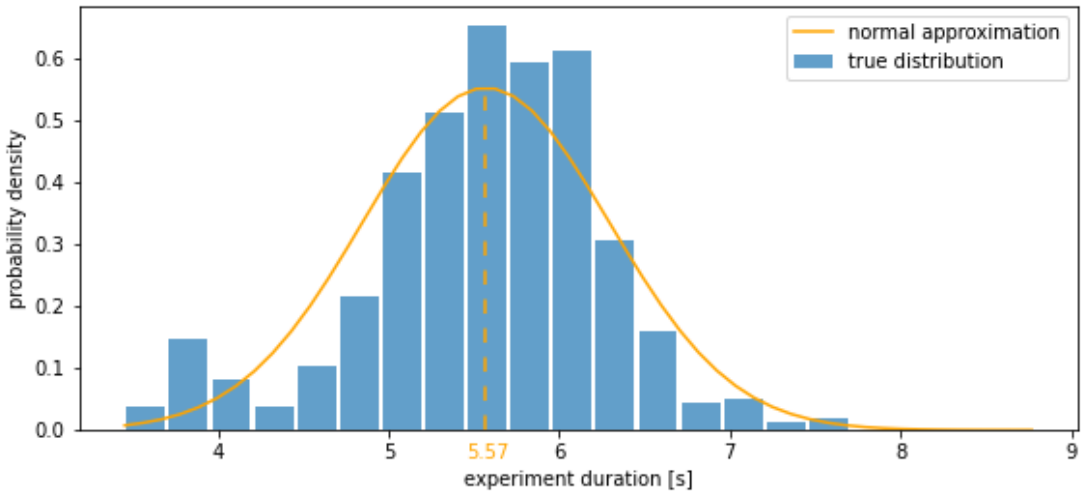



**Figure 8:** Normalised distribution of duration of trials in the dataset

Since this kind of data will be used to feed deep NN, being it convolutional or recurrent, it is necessary to reshape such a format into another one more NN friendly. In this discussion I propose to reformat every trial into a matrix fashion where neural units are the rows, while columns are obtained dividing the total trial time in small time units (time bins). Elements of the matrices will be the count of how many times each neuron shoot during a specific time bin.

However, despite the number of neurons is constant for all the recordings from this animal, to determine the number of time bins, was a design decision that would affect the rest of the work. The problem arise for the the length of the trials being variable but same dimension windows must be presented as input to a NN. In order to get an equal dimension on time axis it was possible: (**1**) to fix a binning time, and discarding some parts of the longest trials or (**2**) to use a variable binning time.

Figure 8 displays the probability distribution function of the duration of the trials in the dataset. It is worth highlight how it resembles a normal distribution

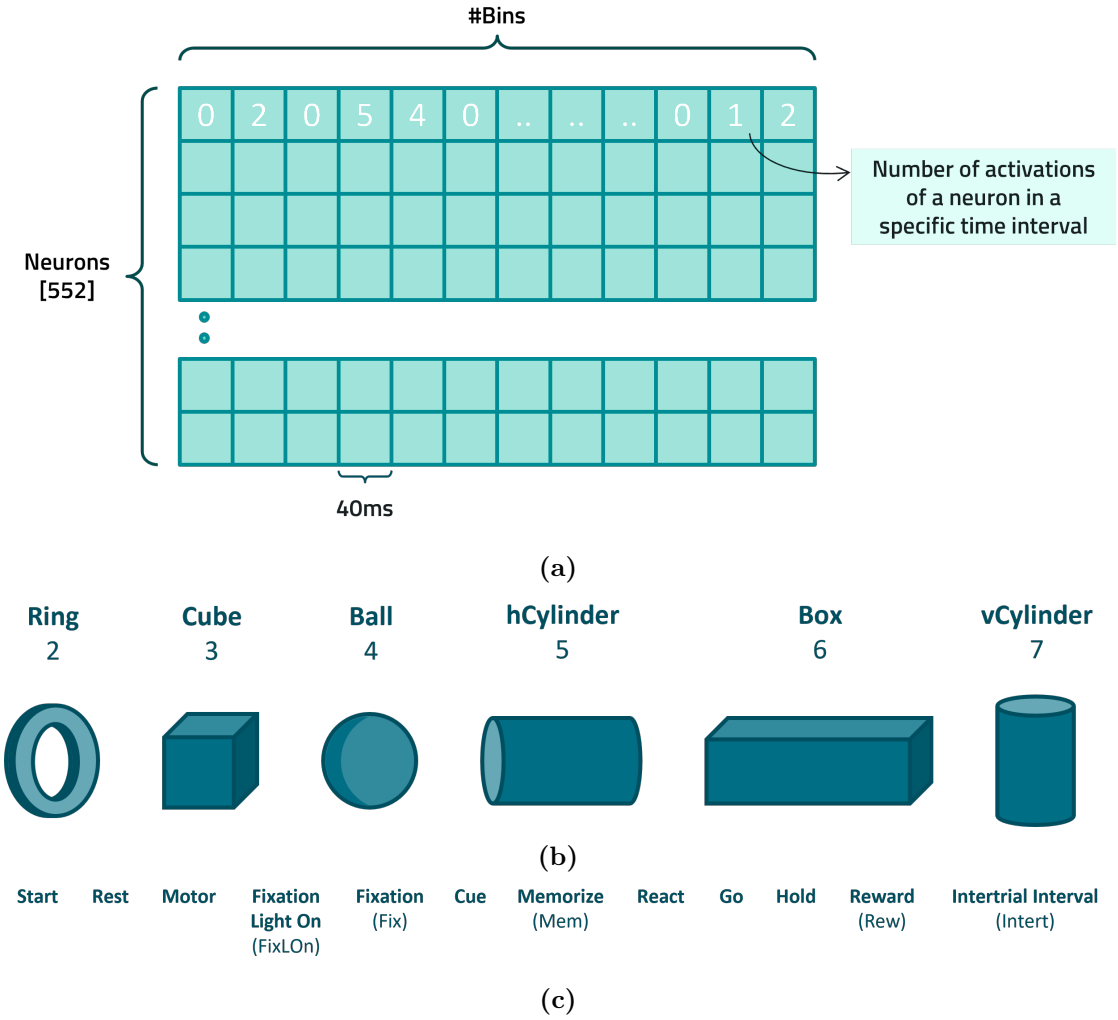



**Figure 9:** Example of a trial described as a matrix (a), associated to one of the 36 target object (b) and to a list of trial epochs (c)

with $\mu 5.57s$ and $\sigma 0.72$. This variance, and the long tails of the distribution are due to a stacking of some random variables. Firstly the fixation time of the experiment is normally distributed between 500-800ms, secondly the monkey is allowed to grasp the object after a random period between 600-1000ms and lastly, the time to reach the object is totally depending on the animal action, being therefore unpredictable. Taking into account this distribution, the idea of cutting the windows to match the shortest seemed to be too much of a waste of information, especially given the long tails, while the second option on the contrary, would allow to use all the data, if the time bin dimension will be taken from a normal distribution. So the idea is to set a desired duration for each time bin, and calculate the number

of bin that such a duration would generate in the average-length trial: this will produce same dimension windows, where each time bin would be described by a normal distribution with the desired mean and same standard deviation as the whole dataset one.

The duration of a bin should be long enough to reduce the dimension on the time axis, but still small for not losing temporal information. Researchers from German Primate Centre suggested from that 40 ms proved to be a good trade-off in their experience. This is coherent with the related works discussed in chapter 1 where time bin for the motor cortex data was 50 ms.

I evaluated the time binning transformation starting from the spikes lists, exploiting the Elephant python module. Being the average duration of a whole trial $5.57s$, that led to 139 time bins. An example of a window obtained with such a transformation is displayed in Figure 9a. Each trial is described as a matrix, having the individual neurons as rows, and the count of their activation every $\sim 40ms$ as elements. To each window is assigned as label the target object of that trial. For this dataset, only the most common objects were accounted, having only few examples for each special shape. A complete distribution of all the labels in the complete session is reported in Appendix A. The object was picked the from the 36 possible ones reported in Figure 9b, and each of them is uniquely described as $Shape\_Number$ and $Size\_Number$ (an integer between 1-smallest- and 6-biggest-). Lastly every time bin is also associated to its concurrent trial epoch (Figure 9c), leading to an array with 139 elements used as second label for each trial window.

## 2.3 Dataset Analysis

After discussing the procedures that brought to the creation of the dataset, in this section I will discuss its consistency before proceeding to any application of machine learning on it.

Firstly, I was concerned that the number of trials in the dataset might be too few and possibly too similar among each other. Such a situation would likely cause the ML model to learn the data and not how to extract features (the so said *overfitting* problem). The most common solution to the small number of examples is data augmentation: algorithms that randomly modify real examples to generate synthetic ones. However, those are specifically designed for one kind of data and are not likely to fit different ones: for example, it is well known the approach of tilting, cropping, masking images, but this is not something applicable to time series. The pattern of neurons activations is highly subject dependent, and so far it is unclear how to generate realistic synthetic ones. For this reason, I avoided trying this path. However, I could investigate the inner variance of the dataset.

Once more, there is no clear suggestion about how to assess the diversity of binned neural recordings. I decided to consider the total number of activations per time step and to check how much it differs among the trials of the dataset, using the standard deviation as the metric. The result is displayed in Figure 10. The blue line reports the average number of total activations for all the trials in the dataset over the experiment duration. The yellow area is the standard deviation in the number of activations, evaluated per each time step. We can observe that the variance is constantly high enough, with respect to the mean value, to assume the trials are different among each others and not repetitive.

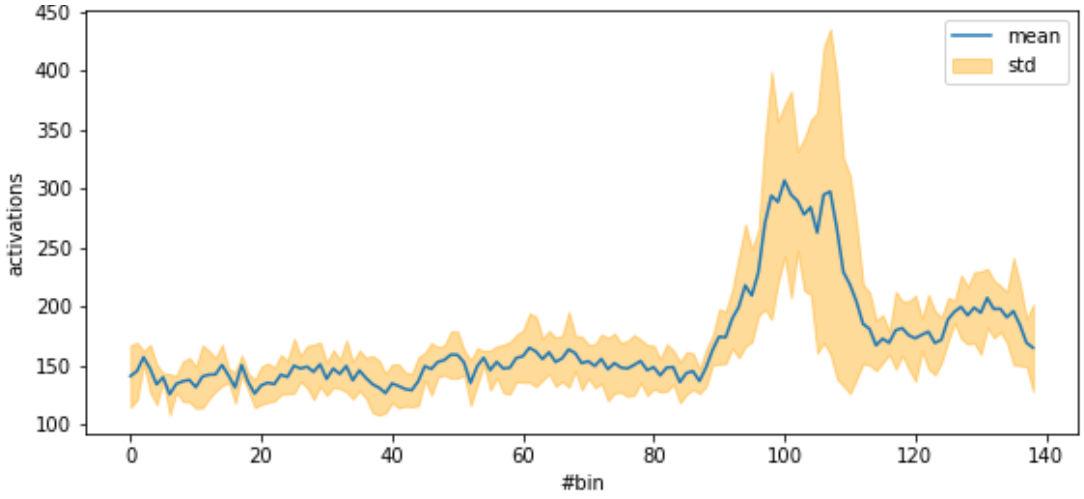



**Figure 10:** Average number and variation of activations among all the trials

Second analysis regards the distribution of target objects in the recorded trials. As stated in the previous section, I reduced the number of objects to avoid the special shapes that appeared only once and could confuse the model, keeping only the standard six shapes (2-ring, 3-cube, 4-ball, 5-horizontal cylinder, 6-box and 7-vertical cylinder) with their six increasing sizes. In Figure 11 I plot the number of time each object was presented to the monkey during the session composing the database. Recall that each object is encoded as a two integers label, representing "Shape" and "Size". All of the objects were presented to the monkey 12 times in the intercourse of the session. Although the number of examples per class is really small, I am assured that the dataset is not biased toward any particular classes.

Lastly, I am reporting in Figure 12 the analysis of the distribution of epochs of the experiments. The distribution of this label is clearly unbalanced: the number of times each epoch is presented in the dataset vary a lot. That is due to the different phases of the experiment having different duration, especially few of them are extremely short. This might be a problem when it comes to training a


**Figure 11:** Distribution of target objects (encoded as Shape-Size) in the dataset

neural network, that might privilege the classification of the most common classes, therefore some balancing procedure will be required before proceeding with the regression training, such as removing the least important epochs and/or capping the number of examples of the most represented ones.

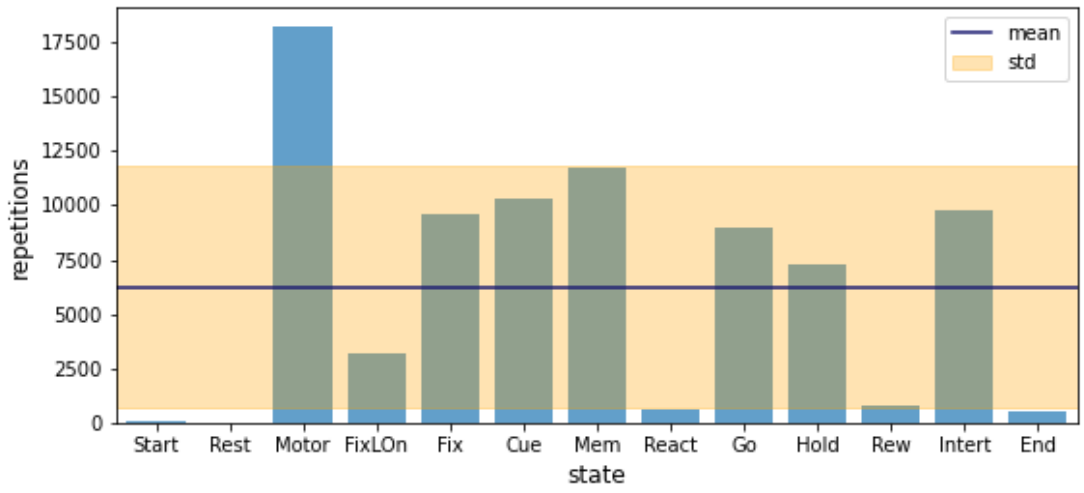



**Figure 12:** Distribution of trial states in the dataset

# Chapter 3

# Offline decoding

In this section, I am about to discuss the classification results obtained with on *offline* analysis of the dataset. With "offline", we usually refer to analysis performed when the complete recording is already available and can be decoded at once, without any constraints about the delay or resource usage. A decoder based on a NN trained offline will be hardly implementable in a real-time situation, but it will be rather used only for research purposes. To perform such an analysis I selected some architectures to try, I investigated how their parameters affected results and then compared the final results obtained by each of them, after being optimised. A particular focus will be put on the network that outperformed the others and that will be taken as starting point for the *online* decoding in chapter 4.

## 3.1 Methodologies

In chapter 1 I reviewed the related work about non-invasive BCI. In that context, besides the shortage of references, I highlighted as important limitations of the literature, that so far in around one-third of the works, features were manually extracted, following researchers own knowledge of the data nature. However, this approach that used to be popular, raises two objections. The first one introduces an extra preprocessing step that, depending on the time it requires, can introduce delays in the classification if it is short or do not apply to real-time applications if it is long. Secondly, most of the features engineering algorithms, such as *principal component analysis* (PCA), or selecting few frequencies bands, as in the case of FFT processing, lead to a reduction of the information available. For instance, PCA drops features that seem not necessary to reconstruct the final output, but still, those deleted ones might have been correlated to temporary or spatially connected measurements, in a deeper pattern that a statistical approach like PCA can not detect. This problem is even more crucial with FFT, CWT, and similar

approaches found in literature, where most of the data is discarded because so far they have not proved to be necessary. But if such an approach was crucial to reduce noise for statistical models, nowadays should be not necessary, and possibly counterproductive when working with deep NNs, especially if the dataset is large. For these reasons, and also considering that this field has not been deeply explored yet, it is preferable to assess how much deep NN can extract proper features, and eventually compare in future this raw results with others obtained on manipulated data.

To decide which networks to involve in this offline analysis, I summarised the major requirements they should answer to be suited for the real-time demonstration that is the final purpose of this project:

- deep NN based, for it's proving to be good at extracting features from raw data
- Quick network, for granting short delay in real-time application
- Lightweight to be implemented on small microprocessors, perhaps wearable and possibly implanted

The last two points can be also summarised as the necessity to keep the network simple. This is also what researchers who compared the NNs applied on iEEG claimed to be the reason behind their good performances despite the small dimension of the datasets: probably due to the low number of parameters they were less likely to *overfit*.

Overfitting represents indeed, the major problem to cope with, being the collection of data relatively small, highly related to the subject and biased by the experimental optimal conditions and routine: a too complex model might learn the sequence of the experiment steps instead of proper features, or be badly affected by the noise of brain activity not related to the task. Besides keeping the network simple, regularisation techniques, such as L1, L2 or dropout will be implemented for the same reasons.

Lastly, since science literature has not found yet the best models to work with data of this nature, networks popular for other tasks will be considered, such as *convolutional neural network* (CNN) that is especially effective on image detection, or *recurrent neural network* (RNN), that is suited for natural language processing. However, it is worth recalling that there is not any network that can outperform all the others in every task. This is called "*no free lunch theorem*" [52] and it a basic theorem of ML. This is because deep NNs are the best approximators of non-linear correlations, and they do so by making intrinsic assumptions about the nature of the relationship they are emulating. These assumptions are what make one network more suitable for a task than the others. For this reason, it is a good

practice to test basic and different models on this not deeply explored yet task, instead of focusing on the most trendy networks.

### 3.1.1 Networks

To address all the requirements listed so far, I decided to only use only deep NNs, composed of a single hidden layer, responsible to extract features from raw data in a peculiar way. It is regularised by a dropout step and followed then by a final *fully connected* (FC) layer for the classification task. The single layers that will characterise the network are taken from the most popular ones but are not mixed to reduce complexity and to make it easier to compare the performance that different families of architectures can achieve on this dataset. The final FC will fix the number of outputs to the number of possible classes. This layer is activated by the *Softmax* function that will force all the outputs to sum up to one. This way, each output can be read as the probability of the input belonging to that specific class. All the following networks can be found in the GitHub repository under "/utils/decoders.py". The code provides support to perform the same comparison on a different dataset.

#### Dense Neural Network

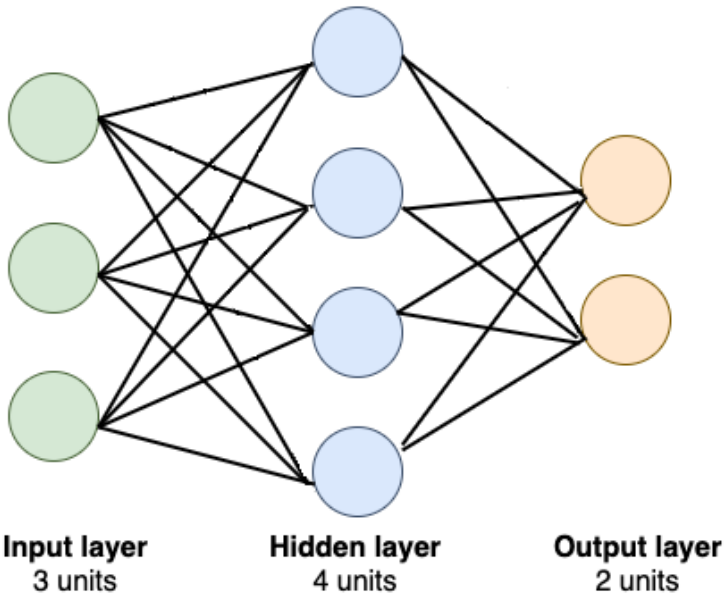



**Figure 13:** An example of Feed-forward Neural Network with one hidden layer

In *Dense Neural Network* (DNN) all the hidden layers are FCs as displayed in Figure 13. For this reason, this NN is also said *Feed Forward*. With such a fashion

every unit broadcasts its output to all the ones in the following layer. The output is obtained as linear transformation of the inputs activated by non some linear function (e.g sigmoid, hyperbolic tangent, rectified linear unit, etc..)[53]. This model requires data to be shaped in a long mono-dimensional array, so that every entry can be treated as unique features in the input layer. Then, in the hidden layer, an (usually) high number of neurons linearly combine all of the inputs before activating them with the 'ReLU' function, as attempt to generate new and possibly more significant features. A dropout step is introduced for regularisation purpose, to randomly cut a fixed percentage of connections. Some of the neurons in the hidden layer will be forbidden to broadcast their output, reducing the chance of overfit. Lastly a FC is add for classification.

```
DNN(window, channels, outputs, neurons, dropout)
"""
    window, channels: are the input data dimensions
    outputs:  is the number of classes
    neurons: is the number of neurons in the fully connected layer
    dropout: is the fraction of dropout regularisation
"""
# Initialisation
model = Sequential()
# Flattening the input to fit the feedforward network
model.add(Reshape(target_shape=(channels * window,),
  input_shape=(channels, window)))
# Add hidden layer
model.add(Dense(units[i], activation='relu'))
# Add dropout
model.add(Dropout(dropout))
# Add output layer
model.add(Dense(outputs, activation='softmax'))
```

### Convolutional Neural Network

In CNN the input needs to be shaped matrix fashion so that several filters (i.e. smaller matrices) can be used to perform *convolution* over it. With this term, we refer to the sliding of a filter over the input and multiplying it for the area underneath as displayed in Figure 14. Usually, the matrix product is also activated by a non-linear function. Differently from the example, the input can be also three dimensional (as usually, pictures are) and so will be the filters. This process generates a new matrix that reflects how much the filter reacted to a different portion of the input. Output dimensions depend on the filter size, the sliding step and the number of filters. Actually, for every filter, a new layer will be added to the final output, which will then be three-dimensional [53]. In the model used for

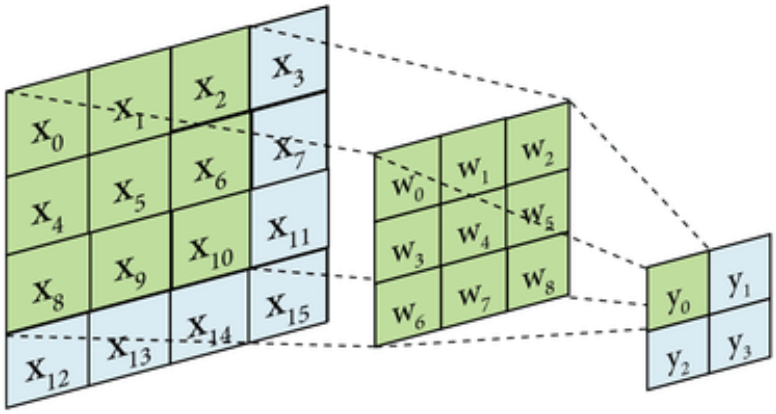


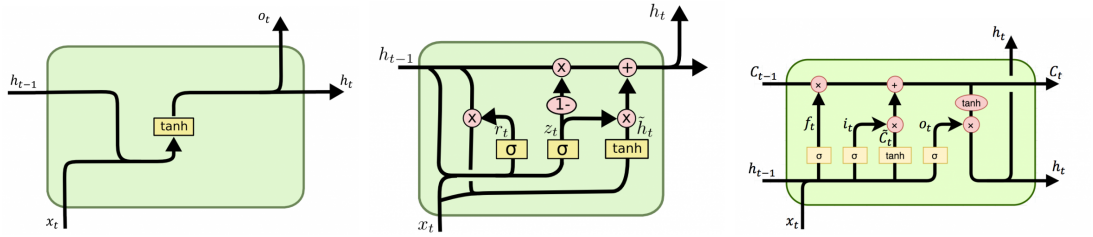

**Figure 14:** Example of feature extracted by convolution (Samrat Sahoo - *medium.com*)

this work, the input is provided in its window shape, having as dimensions the channel number and the time step. The output of the convolution is activated by the 'ReLU' function and regularised by dropout. Then a pooling layer is added, that reduces the dimension of the output, substituting to every pair of cells, the maximum one. This step has deeply proved to be useful in CNNs, as a processing step before flattening the results and apply a final FC for classification.

```
CNN(window, channels, outputs, filters, size, dropout, pool_size)
"""
    window, channels: are the input data dimensions
    outputs:  is the number of classes
    filters: is the number of filters in the convolutional layer
    size1, size2: dimensions of each one of the filters
    dropout: is the fraction of dropout regularisation
    pool_size: size of the kernel used for MaxPooling
"""
# Initialisation
model = Sequential()
# Reshaping the input adding a dimension to fit the convolutional
   layer
model.add(Reshape(target_shape=(channels, window, 1),
   input_shape=(channels, window)))
# Add convolutional layer
model.add(Conv2D(filters, (size1, size2), activation='relu'))
# Add dropout
model.add(Dropout(dropout))
# Add pooling layer
model.add(MaxPool2D(pool_size))
# Add output layer
model.add(Flatten())
```

```
22 model.add(Dense(outputs, activation='softmax'))
```

### Simple Recurrent Neural Network




**(a)** **(b)** **(c)**

**Figure 15:** A comparison among simple RNN (a), GRU (b) and LSTM (c) (*dprogrammer.org*)

The peculiarity of RNNs is that they introduce recursion in the flow of information. That is obtained having special units which store a hidden state so that the output they send to the following layer is the activated linear transformation of the input *and* the previous state. Such a network can be seen as the recursion over the same unit or more often, the sequence of several units with unique states. Such a network is suitable to find temporal dependencies, but they require input to be provided in temporal sequence fashion [53]. The first and most simple implementation of such a network is based on the SimpleRNN unit described in Figure 15a. The outputs generated by a layer composed of many units are activated by *hyperbolic tangent* function, regularised by dropout and then fed to FC for classification.

```
1  SimpleRNN(window, channels, outputs, units, dropout)
2  """
3      window, channels: are the input data dimensions
4      outputs: is the number of classes
5      units: is the number of units in the recurrent layer
6      dropout: is the fraction of dropout regularisation
7  """
8  # Initialisation
9  model = Sequential()
10 # Swapping channels and window axis to fit recursive layer
11 model.add(Permute((2, 1), input_shape=(channels, window)))
12 # Add recurrent layer
13 model.add(SimpleRNN(units, activation='tanh'))
14 # Add dropout
15 model.add(Dropout(dropout))
16 # Add output layer
```

```
model.add(Dense(outputs, activation='softmax'))
```

### Gated Recurrent Unit

*Gated Recurrent Unit* (GRU) is a variation of the basic RNN. It exploits gated units represented in Figure 15b to regulate the information that can flow through various parts of the network. In practice, these gated units allow for better learning of long-term dependencies [53].

```
GRU(window, channels, outputs, units, dropout)
"""
    window, channels: are the input data dimensions
    outputs: is the number of classes
    units: is the number of units in the recurrent layer
    dropout: is the fraction of dropout regularisation
"""
# Initialisation
model = Sequential()
# Swapping channels and window axis to fit recursive layer
model.add(Permute((2, 1), input_shape=(channels, window)))
# Add recurrent layer
model.add(GRU(units, activation='tanh'))
# Add dropout
model.add(Dropout(dropout))
# Add output layer
model.add(Dense(outputs, activation='softmax'))
```

### Long Short-Term Memory

Like GRU, *Long Short-Term Memory* (LSTM) extends the concept of RNN, adding extra gates beside the hidden state (Figure 15c), that allows the network to evaluate when some information can be forgotten and perform better learning of long term dependencies. However, this comes with a higher number of parameters [53].

```
LSTM(window, channels, outputs, units, dropout)
"""
    window, channels: are the input data dimensions
    outputs: is the number of classes
    units: is the number of units in the recurrent layer
    dropout: is the fraction of dropout regularisation
"""
# Initialisation
model = Sequential()
```

```
# Swapping channels and window axis to fit recursive layer
model.add(Permute((2, 1), input_shape=(channels, window)))
# Add recurrent layer
model.add(LSTM(units, activation='tanh'))
# Add dropout
model.add(Dropout(dropout))
# Add output layer
model.add(Dense(outputs, activation='softmax'))
```

### Ensemble: CNN and LSTM

The last network tested is the composition of two different layers, to separate in two phases the extraction of spatial and temporal features. Similar approaches were investigated in works about raw EEG data [39, 38, 40], and I found it interesting to propose a custom one for this analysis. For each time step, the recording from the channels is processed by a mono-dimensional convolutional layer to reduce dimensions while extracting spatial features on the channel axis. Secondly, a recurrent layer learns temporal dependencies. After both layers, the output is regularised by a peculiar dropout fraction.

```
Ensemble(window, channels, outputs, filters, size, dropout,
    pool_size, units, neurons)
"""
    window, channels: are the input data dimensions
    outputs: is the number of classes
    filters: is the number of filters in the convolutional layer
    size: size of each one of the filters
    dropout: is the fraction of dropout regularisation
    pool_size: is the size of kernels in the pooling layer
    units: is the number of units in the recurrent layer
    neurons: is the number of neurons in the fully connected layer
"""
# Initialisation
model = Sequential()
# Reshaping the input swapping axis and adding a dimension to fit
    the convolutional and the recursive layers
model.add(Reshape((window, channels, 1), input_shape=(channels,
    window)))
# Add convolutional layer to extract spatial features from each
    time step
model.add(TimeDistributed(Conv1D(filters, size, activation='elu')
# Add dropout
model.add(Dropout(dropout))
# Add pooling layer
model.add(TimeDistributed(AvgPool1D(pool_size)))
```

```
# Add recursive layer to extract temporal features
model.add(TimeDistributed(Flatten()))
model.add(LSTM(units, activation='tanh'))
model.add(Dropout(recurrent_dropout))
# Add fully connected layer
model.add(Flatten())
model.add(Dense(neurons))
# Add output layer
model.add(Dense(outputs, activation='softmax'))
```

### 3.1.2 Hyperparameters tuning: hyperopt

The task of tuning the network searching for its best configuration is called *Hyperparameters optimisation* (or tuning). Hyperparameters are all the values that can tweak in a network, that are not directly affected by the training phase but influence the outcome (common examples are batch size, learning rate, number of epochs, etc..). This task is known to be extremely long and greedy of resources since it necessarily requires a lot of training sessions, during which a configuration must be used for the training process and then tested on the validation set. The complexity is related to the number of parameters to tune, their *search space* (i.e. their range and density of variation) and the optimisation algorithm used. Hyperparameters optimisation is represented in equation form as:

$$x^* = \underset{x \in X}{\operatorname{argmin}}\, f(x) \tag{3.1}$$

where $f(x)$ is the metric score on the validation set to minimise, $X$ is the set with all the elements of the search space (all the possible combinations of parameters) and $x*$ is the configuration that minimises the metric score.

The most basic optimisation algorithm is said *Grid Search* and consists in dividing the search space into several equally distant values, and methodically try all their possible combinations. Grid search is guaranteed to find the best configuration among all those it was given, but it might take forever. Moreover, there is no assurance that the best combination with the given values corresponds with the optimal one, and every time you make the grid denser or expand the borders the computation time grows as well.

Different philosophy is found in searching algorithms based on randomicity that usually shows the best results when time is limited: they randomly extract configurations to test from the search space and repeat until a stopping condition is met, like a certain number of iterations. Unfortunately, the common implementation of random search completely ignores information on the trials already computed, and each new sample is drawn from the same initial distribution. Both random search and grid search waste a significant amount of time on evaluating bad

hyperparameters. Fortunately, there is a way to account for them, correcting the random sampling introducing some kind of weights that could penalise an area close to a particular value that performed poorly. In other words, we want to get more points from the regions with a high probability of yielding good results and get fewer points from elsewhere. That's exactly what *Hyperopt* module helps to deploy [54]. This approach is said *bayesian* as it reshapes the problem from the form of probability of the hyperparameters into the probability of the score given hyperparameters:

$$P(score|hyperparameters) \tag{3.2}$$

From this assumption begins the so said *Sequential Model-Based Optimization* (SMBO), that is a formalization of Bayesian optimization: trials are executed iteratively keeping track of a hystory (configuration-score), each time sampling hyperparameters using Bayesian rule and updating the probability model [55]. In particular, the standard algorithm embed by Hyperopt to select next configuration to test is said *Tree of Parzen Estimators* (TPE) [56].

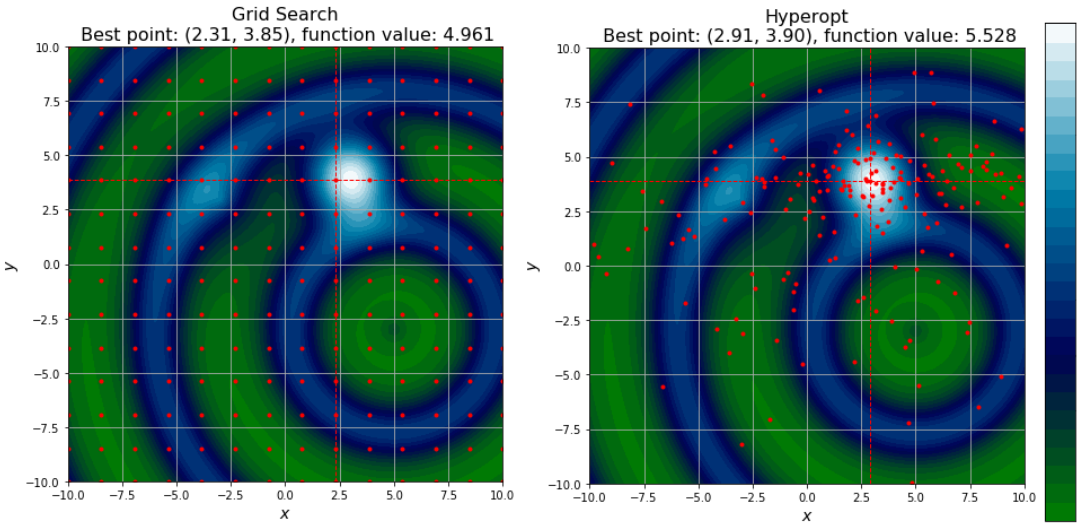



**Figure 16:** Grid search and hyperopt optimised random search sampling comparison (Alexey Serov -Keggle) [57]

Figure 16 displays the different strategies that grid search and hyperopt follow to draw a configuration from the search space [57]. In this simple example the parameters are only two, represented by the two axes of the plots while the colour represents the accuracy score obtained with that configuration: the brightest the colour, the higher the accuracy. We can observe the advantage introduced by hyperopt optimised random search, for which the selection of the parameters converges toward an area of relative minimum of the search space.

### 3.1.3 Experiment description

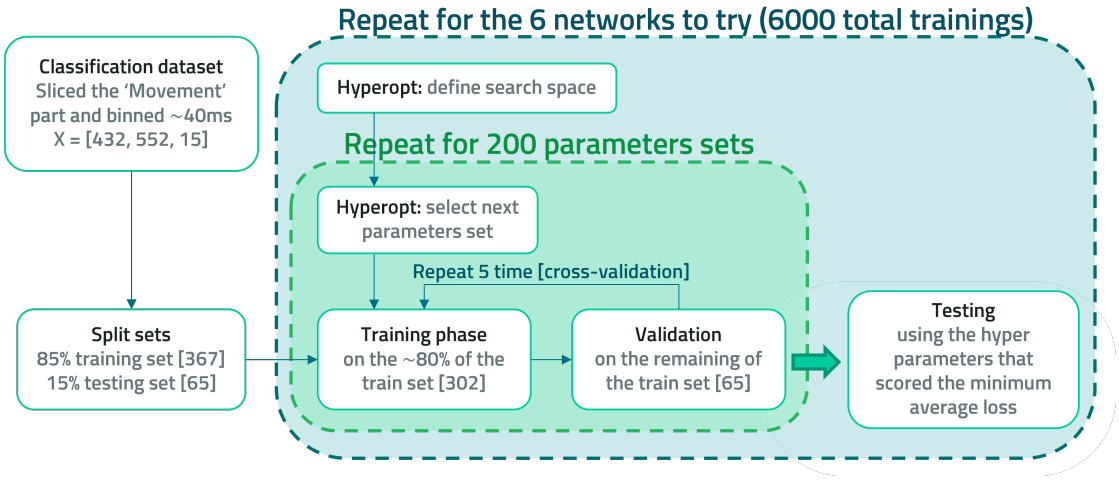



**Figure 17:** Experiment description to find the best network for offline classification

The flow chart in Figure 17 summarises the experiment set up to compare networks and parameters configurations based on the performance they achieved on the given dataset. From the collection of trials windows shaped as described in chapter 2, a smaller dataset is generated, slicing only the brain activity simultaneous to the arm and hand movement. This choice purpose is to account for the computational constraints for this project, speeding up the experiment, that will count thousands of training phases. To operate a prosthetic arm, the 'holding' phase cannot be used for classification, since by the time the command is to hold the object, the decoder must already have decoded the hand configuration to replicate. Therefore, the recordings anticipating that moment are considered to be the most informative for the task. Recalling that the desired average binning time is 40ms, the 'movement' windows were 15 bins long.

After being shuffled, the 85% of the dataset was split to compose the training and validation sets while the remaining 15% was used as testing set. Secondly, for five times the larger split was re-shuffled and from it two smaller portions were generated, counting around 80% and 20% entry each. With this procedure, I generated five training sets, five validation sets and one testing set of samples not present in any of the previous. The meaning of the five training sets is to repeat every training session five times, and every time to be able to validate results on some data unseen in that training phase. Repeating this for five times, all the data are expected to be used at least once for training and validation due to the 80-20 proportion. I preferred this approach with multiple random repetitions over the standard *K-Fold cross validation* for pursuing higher robustness.

For each network under test, a searching space was defined and an exhaustive description of the search spaces for every network is reported in Appendix B.

Summarising, the experimental process start selecting a network, Hyperopt then chose a feasible configuration of hyperparameters to build the model with, and then the network is trained on each one of the training sets and every time validated on the respective validation set. After five training phases, the mean validation accuracy score and the mean training and validation losses were stored as performance related to that specific configuration. This was repeated with 200 configurations elected by the hyperopt module, leading to a total of 1000 training sessions per network. The only exception is the ensemble network, for which 400 configurations were tested due to its higher number of parameters. Its 2000 training sessions plus the ones for the other five networks under analysis led to a total of 7000 training phases.

It is important to highlight how the width of search spaces and the number of variables were again limited to account for the limited hardware capabilities available. There is a concrete chance that the best parameters were not found by this research, due to the not so high density of parameters. However, the purpose of this work was not to find the best possible network or parameters, but that of exploring how different network layers and specific parameters were correlated to the decoder ability to interpret this kind of brain activity record.

## 3.2 Results

### 3.2.1 Hyperparameters tuning history

In Figure 16 the convergence of the search is easy to picture in a two-dimensional plot, due to the small number of parameters. The plots listed later in this section are meant to provide the same information when the search space counts much more dimensions. To understand them it is necessary to recall how the hyperopt optimisation algorithm works: the higher the performance due to a parameter value, the more its neighbour area was searched. Since the search space, I gave to the algorithm was discrete and not continuous, searching the close area often meant to repeat the same value. Therefore, the distribution of the 200 searches per network, will be flat if no value of that parameter increased the objective function result, while an area or a specific value should have been repeated many times, otherwise. The parameters that all the networks shared are batch size, dropout fraction and number of epochs; besides, some specific ones were investigated. The objective of each search was to find the configuration that minimises the average cross-entropy loss function on the validation sets.

### Dense Neural Network

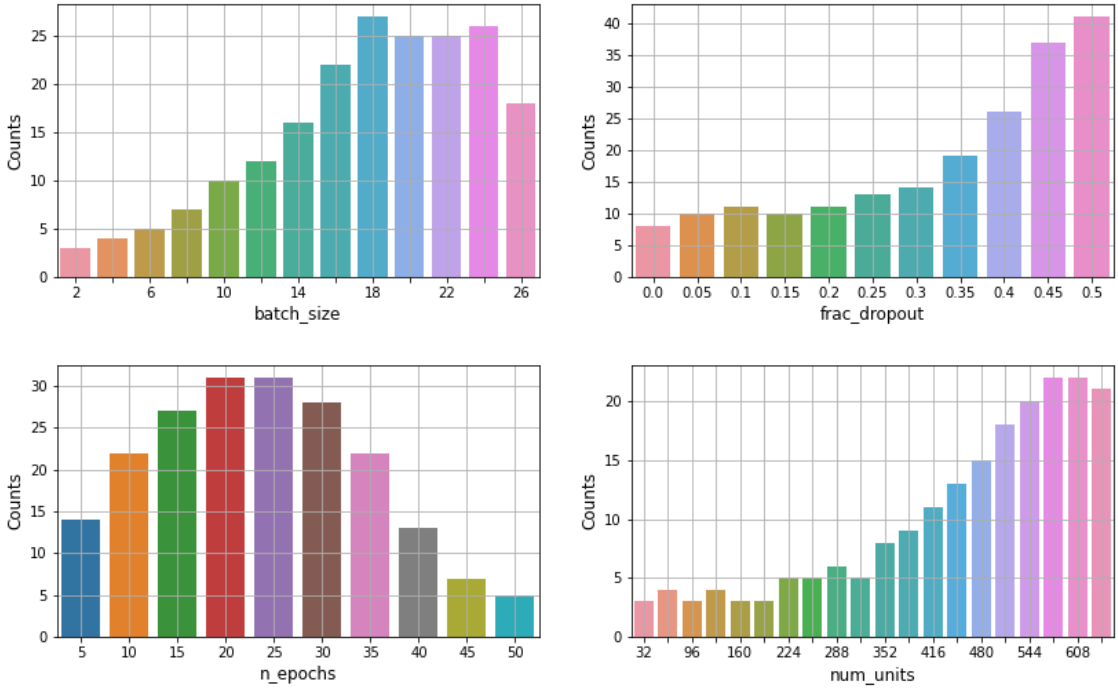



**Figure 18:** DNN hyperopt results

In Figure 18 we can observe the searching history for the four parameters that I tuned for DNN. They are the three common ones and the number of units in the hidden feed forward layer. Starting the analysis from the latter, it seems that the more the units the better the results, since the distribution constantly rise until it meets its maximum at around 600 elements.

This decision makes the network highly prone to overfitting, in fact the number of units is known to be the main issue in allowing the network to learn directly the data and not the pattern underneath. On the other hand a lot of units are necessary for this network, being that one the only step where features are extracted. The other parameters were tuned accordingly to compensate for this risk.

A larger batch size force the model to *generalise* (i.e. to find weight that describe more examples at the same time) and actually its distribution shows a peak at 18 and remains stable for values up to 24, while it linearly decreases for smaller batch dimensions. It is similar the distribution of the number of epochs, where the maximum is found for 20 or 25, but in general, there are not huge disadvantages also for values close to them. However, it is highly discouraged to use a really big number of epochs, once more to prevent the model to learn to carefully data from training set. Last parameter is the dropout fraction, whose distribution is

almost flat until 0.3 and then start rising with the maximum possible values 0.5 preferred over all the others. It is worth recalling the while batch size and number of epochs are indirect way to address overfitting, the dropout represent the only active regularisation applied in those network, and a strong was found necessary.

### Convolutional Neural Network

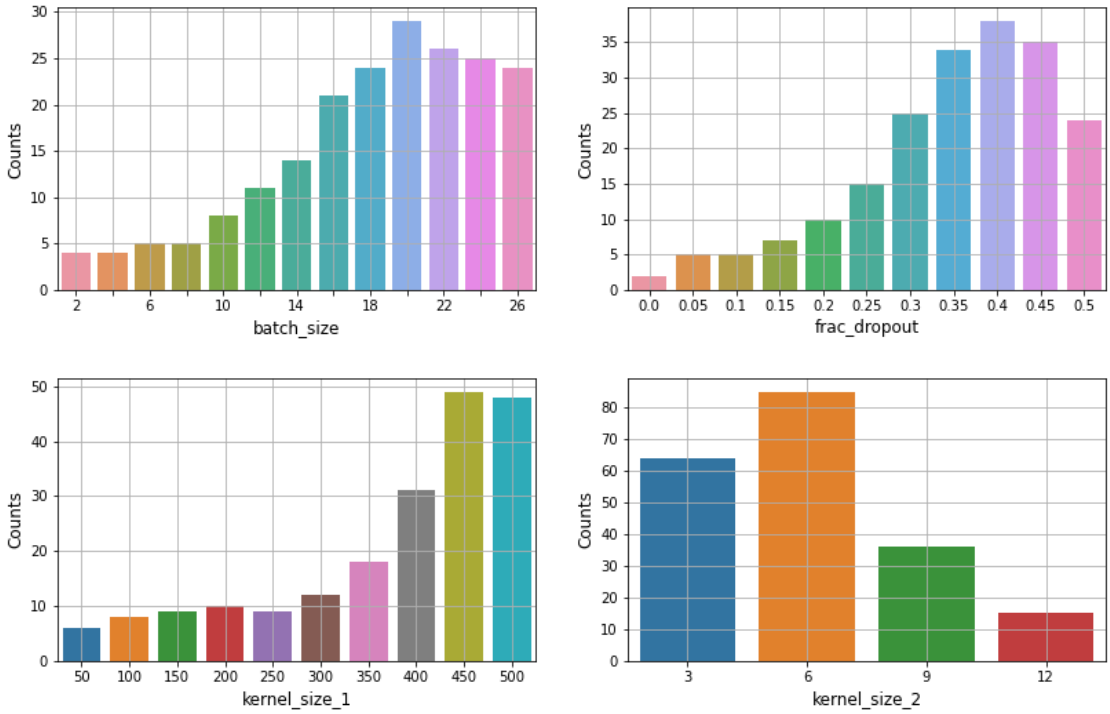



**Figure 19:** CNN hyperopt results part 1

For the CNN model, the peculiar parameters I investigated are the number of filters (or kernels) to apply in the convolutional step, their two dimensions on channel dimension (kernel_size_1) and on the time bin one (kernel_size_2). Moreover whether or not to apply a max-pooling layer with filter dimension 2 was let as a choice to the algorithm. The results displayed in Figure 19 point out that the choice of some parameters such as kernels sizes and the application of a pooling layer were crucial since we observe that some values led to results much better than the others. The favourite filter size was above 450 on the first dimension and 6 on the second one. It is worth recalling that the number of channels, i.e. the row dimension of the input window was 552, therefore the first convolution greatly reduced the dimension on this axis, acting itself as a regulator. The number of filters is relatively small, with a peak between 128 and 160 with results that linearly

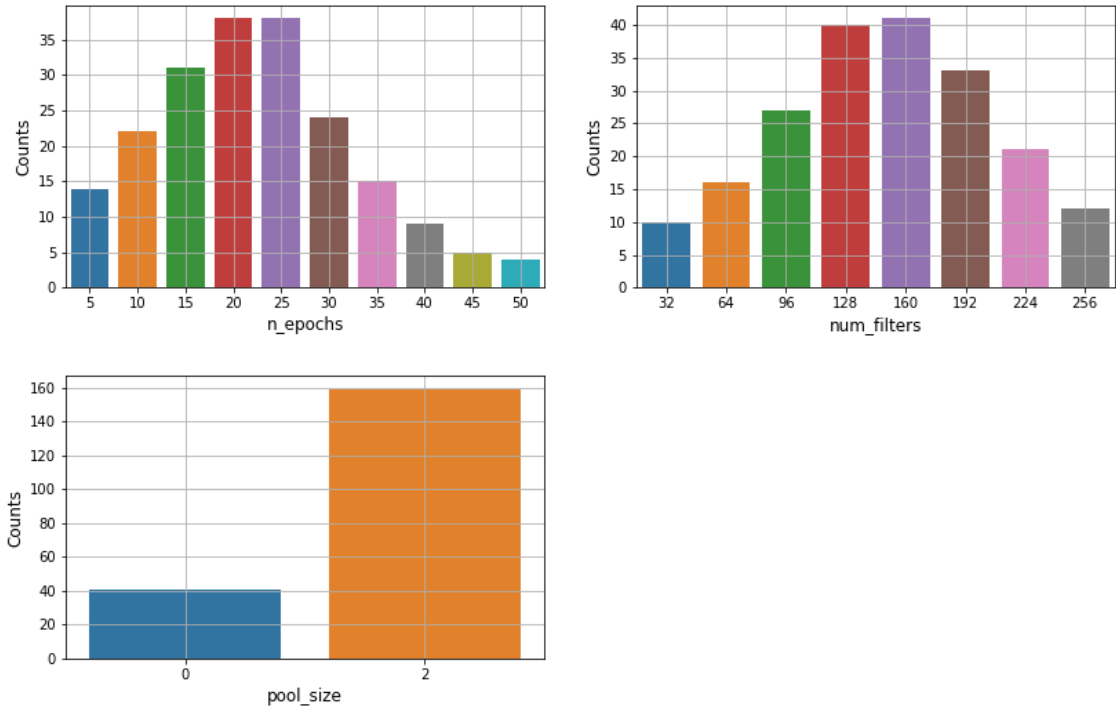



**Figure 19:** CNN hyperopt results part 2

decrease with the distance from those values. Interesting is to find the extremely beneficial effect of introducing a max-pooling layer, that shrinks the dimension even more. For what concern common parameters we find the optimal batch size to count at least 20 samples and the number of epochs to peak at $20 - 25$ iterations, but in general preferred to be small. Despite the regulatory effect intrinsic in the big batch size and the short training phase and the relatively small number of parameters associated with the convolution layer, still, a high fraction of dropout was necessary, at least 0.4. This means that among all the outputs found by the convolution, most of them could and should be dropped in order to improve accuracy.

### Simple Recurrent Neural Network

The only network-specific parameter for all the recurrent layers is the number of units. For what concerns the network composed of a simple RNN layer, we observe in Figure 20 quite a predictable outcome. Firstly, the results improved exponentially with the number of units of the recursive layer, getting stable around the maximum value of $240 - 252$, which was the most popular. Secondly, middle length training phases were required, with possibly 30-40 epochs, to allow the

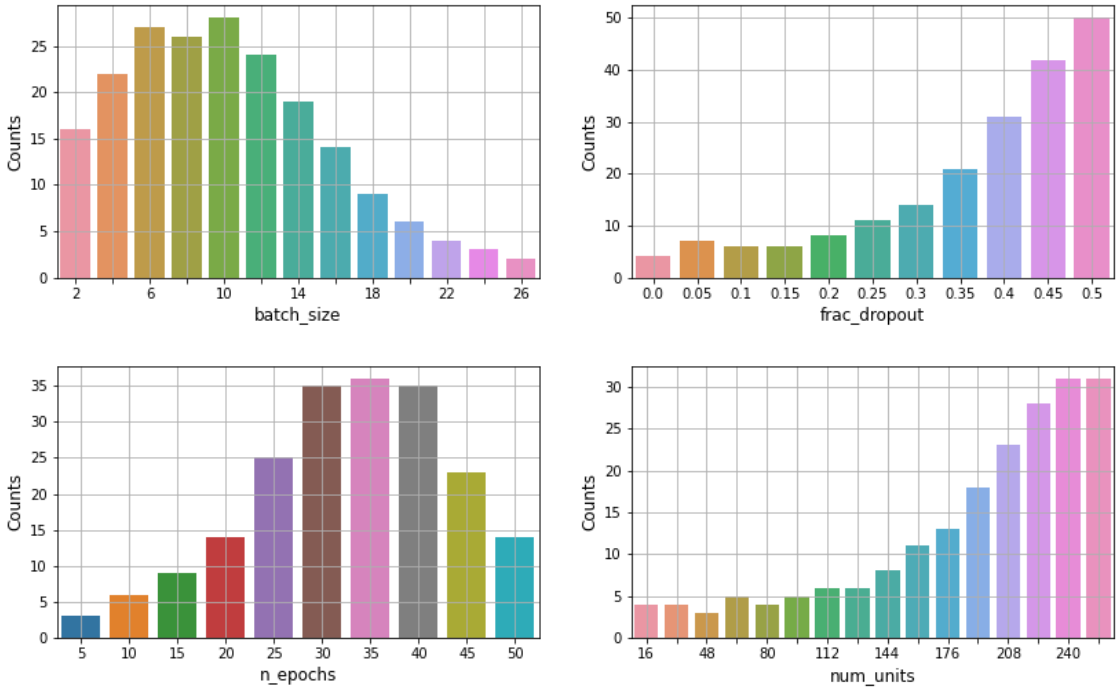



**Figure 20:** RNN hyperopt results

units to learn the temporal patterns. Similarly to what happened with DNN, the maximum fraction of dropout was necessary to balance the high number of units, but despite that, a small batch size between 6 and 10 was sufficient. This is curious since batch size forces the model to generalise to more samples, but that was not found by the algorithm to be proportional to the loss reduction. Probably the high dropout is meant not to avoid overfitting, but to get rid of useless features found by the recurrent step.

### Gated Recurrent Unit

In Figure 21 are displayed the results of GRU tuning. Those are interesting especially if compared to the RNN ones. Even if the same trend is found for the number of gated units, which is directly correlated to the performance, the way the network assures regularisation is the opposite. The dropout fraction seems to be not important for this network, being the flattest distribution among all, with a preference for small values of $0.05 - 0.15$. On the other hand, was crucial the batch size, which peaked for 24 but in general, was proportional to the performance, and the best number of the epoch was 24. Those results suggest that the network extracted meaningful features that should not be discarded by dropout.

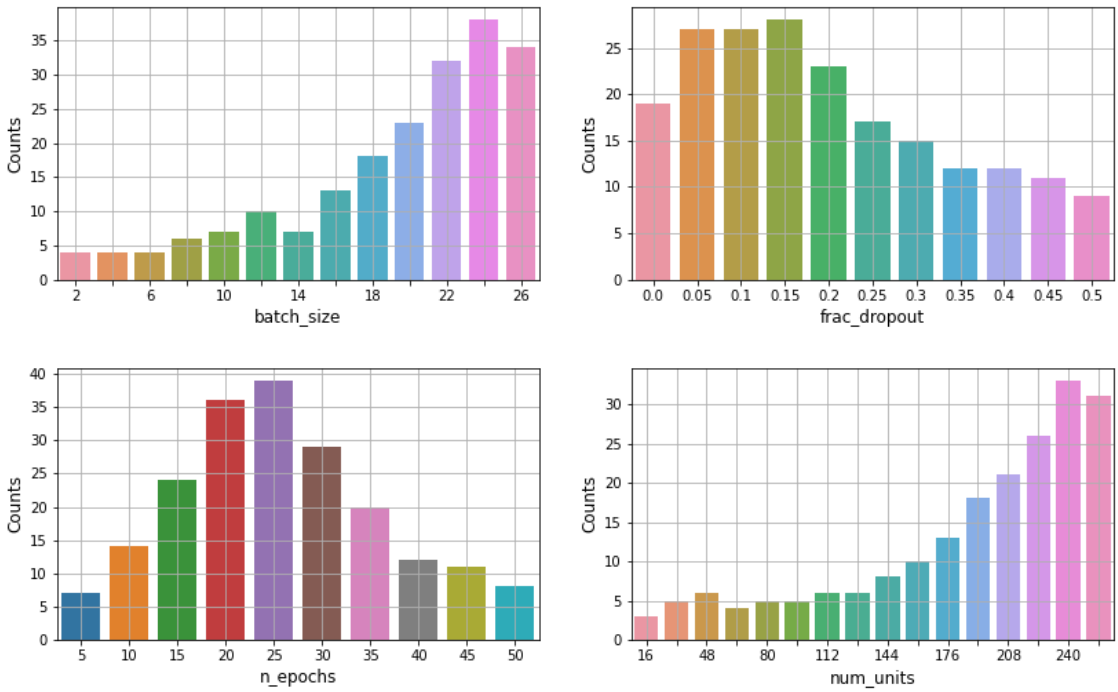



**Figure 21:** GRU hyperopt results

### Long Short-Term Memory

Figure 22 plots show how LSTM tuning history is similar to the GRU one and again opposite to RNN. It is worth mentioning that slightly less dropout was necessary since the best fraction is 0.05 and an even bigger batch size was preferred, with the peak at 26. In general, we can claim that recursion based networks share the need for a high number of units, but the introduction of some sort of gates effectively allows the extraction of temporary features in a reduced learning time.

### Ensemble: CNN and LSTM

The ensemble network is the one with the highest number of parameters to optimise: besides the common ones, like batch size and the number of epochs, I included the CNN specific number of filters and filter size (single value being the filters mono-dimensional), the number of recursive units, one dropout fraction for each feature extracting layer and the number of units in the FC one. Since applying a pooling layer already proved in CNN analysis to be crucial, I substituted that choice with the test of two different pooling rules, '*Max value*' and '*Average value*'. Then I let the algorithm compare whether to include GRU or LSTM layer. Lastly, I took the chance to investigate if '*elu*' activation function, used in a similar work

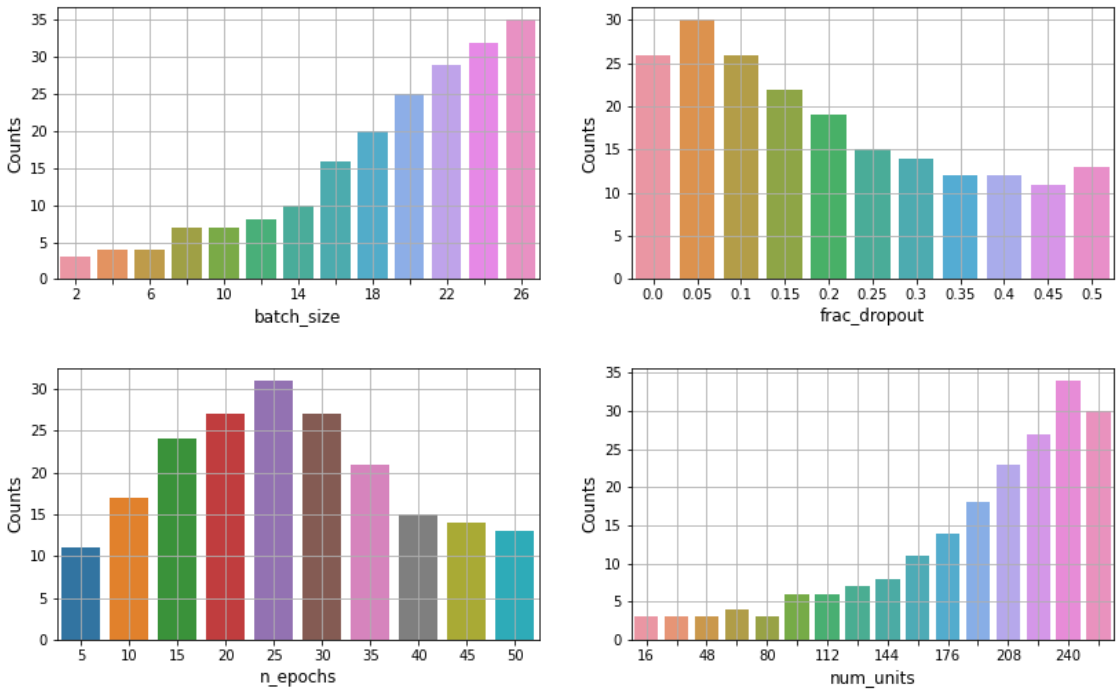



**Figure 22:** LSTM hyperopt results

[39], could be better than 'relu' to introduce non-linearity after the convolution. Due to the high number of combinations, only for this model, I doubled the number of configurations tested from 200 to 400.

The results are reported in Figure 23. The complexity of the network required long training phases with 40 iterations and a quite big batch size with $18 - 20$ elements. The convolutional layer performance increased proportionally to the number of filters, which best size was $208 - 224$ activated by 'elu' function. One characteristic of this network was that the stride of the filter was set equal to its dimension, therefore filters with this size can be applied only twice on the channels dimension, counting 512 rows. The max-pooling layer then was greatly preferred to shrink even more the output, that however, seems to be often meaningful, since only 0.2 dropout fraction was sufficient to generalise after this step. For what concern the extraction of temporal features, the 'LSTM' layer was preferred most of the time, but differently from its own results, here only 32 units (or even less) were sufficient. Probably because the really small arrays they are processing at this step. However, even those few units require a high dropout fraction of 0.45. Lastly, as happened for DNN, the number of units highly increased results, showing a peak in the distribution at value 432.

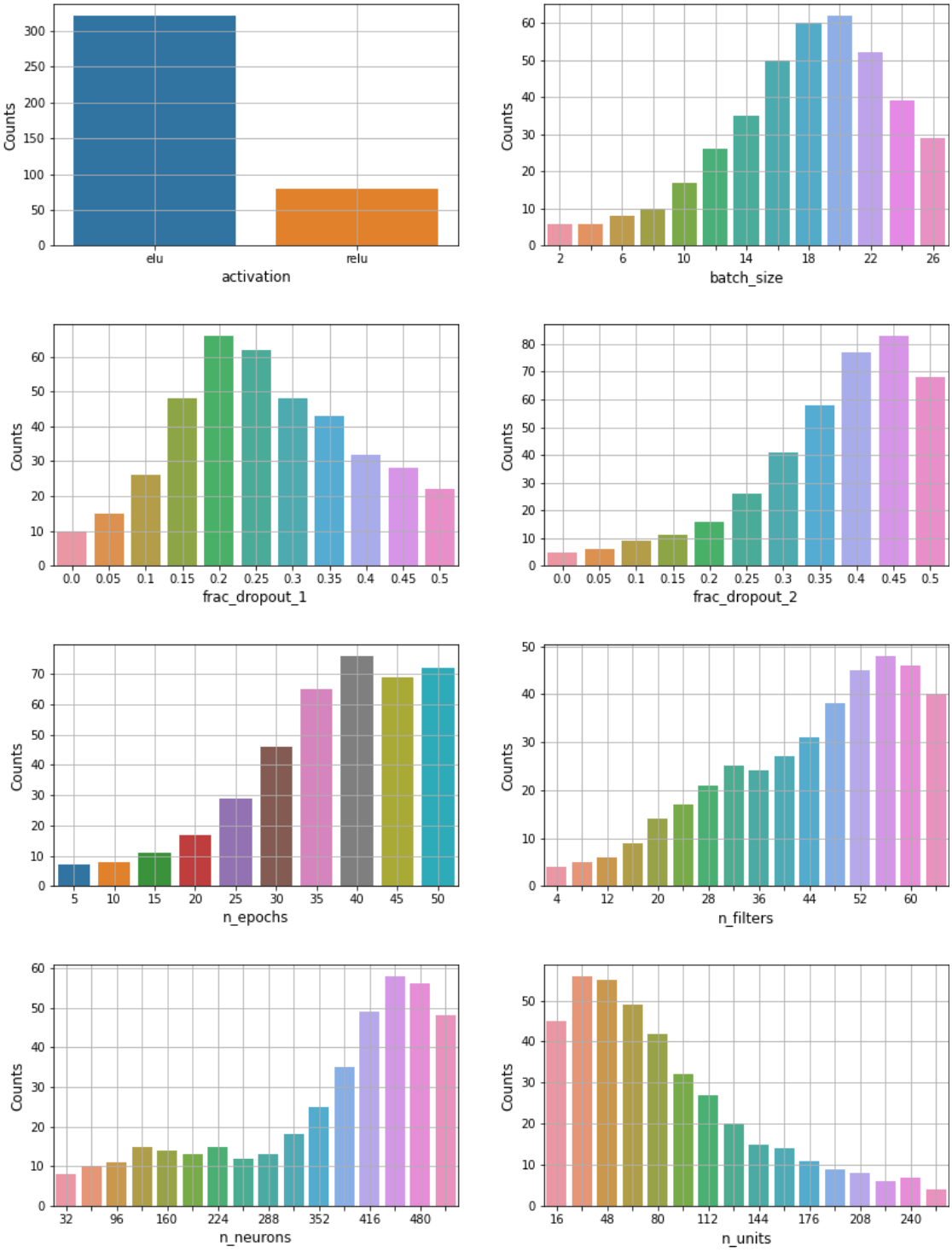



**Figure 23:** Ensemble hyperopt results part 1

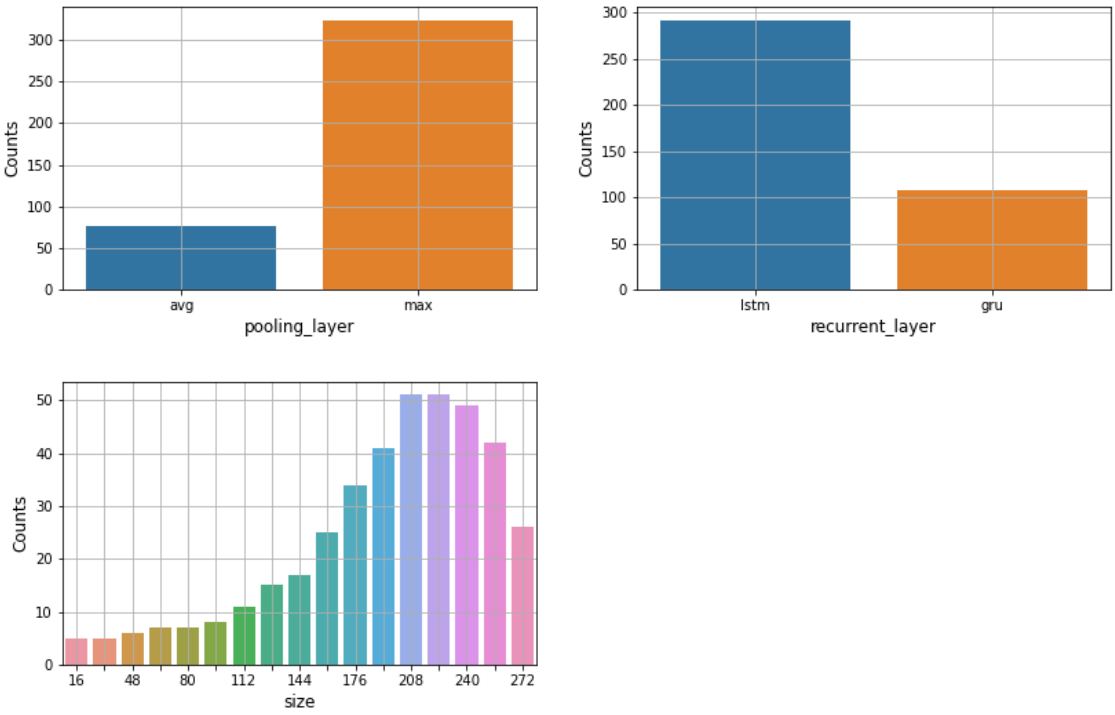



**Figure 23:** Ensemble hyperopt results part 2

### 3.2.2 Tuned networks comparison

As a result of the hyperparameter tuning, I identified the configuration that optimised each one of the networks. An optimised network is the one that scored the average lowest cross-entropy on the five validation sets. The next step was to assess how much each of them could generalise to unseen data. It is for this purpose that a testing set was prepared, composed of the samples not provided during the tuning phase. Every network was trained with its best configuration five more times on the training sets, and every time tested on the testing set. The metrics for each network were stored and displayed in Figure 24. In particular, we observe the cross-entropy loss and accuracy, in terms of mean value and standard error. Being inversely proportional the rest of the discussion will address only the accuracy score, which is more intuitive to understand. However, the same considerations are specular for the loss score.

Firstly we observe that no visible overfitting affected the decoders the accuracy on the validation and testing set are strictly similar, proving the decoders ability to generalise to unseen data. Variance inaccuracy represents a concern only for DNN and CNN. In particular, the latter is the most problematic model, whose results vary significantly between extremely different scores of 0 and 35% accuracy. Such

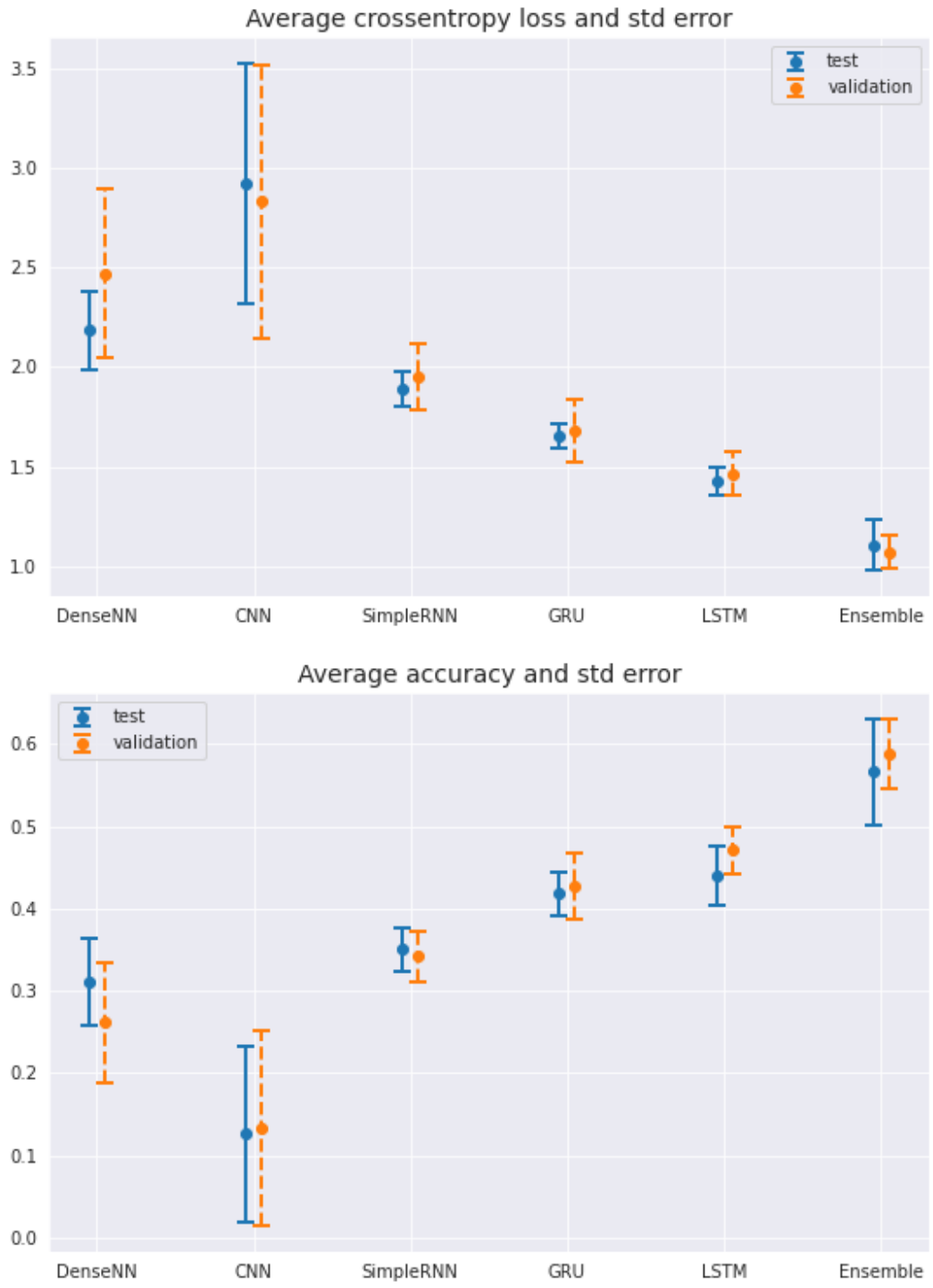



**Figure 24:** Network with tuned parameters comparison on test set

behaviour might suggest that the model faced unknown conditions that forbade it to converge, probably getting stacked at the beginning in a local minimum. However, that remains the network that performed most poorly, followed by DNN. This is not a surprise, being the dense one the most naive decoder. Performance significantly rose after embedding temporal information in the decoding process. In particular, GRU and LSTM proved to be good successors of vanilla RNN, and their gates allowed to increase accuracy by few percentage points. Despite not being successful alone, introducing a convolutional layer on top of an LSTM one was a winning decision, since we observe the superiority of the proposed ensemble over all the other networks. Its average accuracy on the test set is over 55% and the standard error is only a little bigger than the LSTM one, reason why we can consider it reliable.

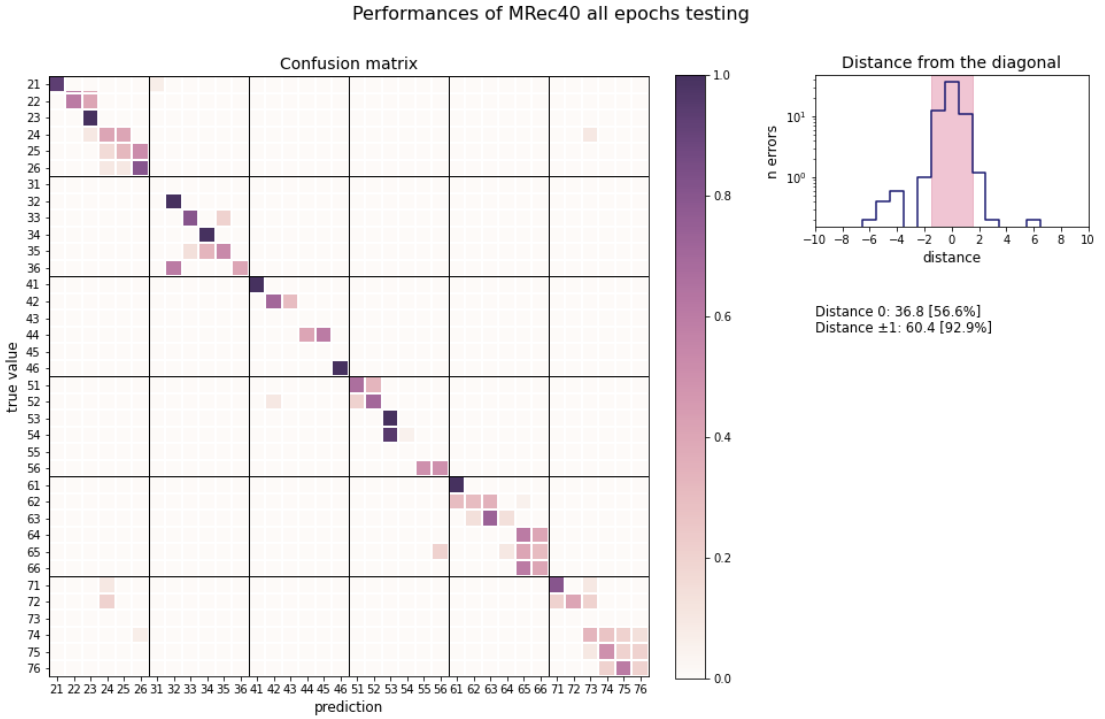



**Figure 25:** Ensemble confusion matrix

In Figure 25 I reported the average confusion matrix obtained by the ensemble. That picture is obtained averaging the five confusion matrices resulted from the five times ensemble model predicted the test set after a training session. This allows us to explore in-depth what did the accuracy score mean. On the vertical axis we find the true label of a test sample, and on the horizontal one, its predicted classes. If an element was correctly classified it is placed on the diagonal. The colour of the cell represents the percentage of elements of that $y$ class (sum of all the elements

in a row) classified with a certain $x$ label. The sum of elements on the diagonal is 36.8 (decimal due to the average function among five matrices) corresponding to 56.6% of the testing set. However, grouping classes by shapes makes it clear that the intra-shape classification is almost perfect. In fact, it is worth recalling that classes are not independent: most of the objects share the same shape and differ only by size one to the others. In these terms, missing the correct class, for one of its neighbours is not a huge mistake from the network. I translated this one-step-mistake accuracy as the distance from the matrix diagonal. For instance, a prediction located next to the diagonal was classified with the class (the size) just before or after the correct one. As expected, the number of samples accepted introducing flexibility is 60.4 that correspond to 92.9% of the whole set. Almost double as the previous result.

The last analysis I propose in this discussion was meant to assess how much those tuned network performances can generalise to a different input, in particular, to a different number of time steps. This would allow getting a rough understanding about how much the network features extraction can be applied in real-time, when the amount of information is being collected, or how much they rely on already having access to the complete window.

In Figure 26 I am reporting the average and standard deviation performance that the networks scored on the test set. Once more the accuracy is specular to the loss, and every consideration on the former is to be intended for both. This time the training phase was repeated as before, but the testing phase was repeated 15 times, every step decreasing the length of the windows provided as input. The missing time steps were substituted by left zero padding in order to keep the windows the same size as the ones used for training. CNN was the network more affected by the reduction of the dimension, while the recurrent based ones were more robust, being them suitable to work with different length input. Once more the ensemble proved to be the most reliable, performing better than the others until four steps long window, being then taken over from LSTM even smaller ones.

For the performance shown so far, I decided to develop my BCI based on the ensemble network, and the parameters found in the optimisation phase.

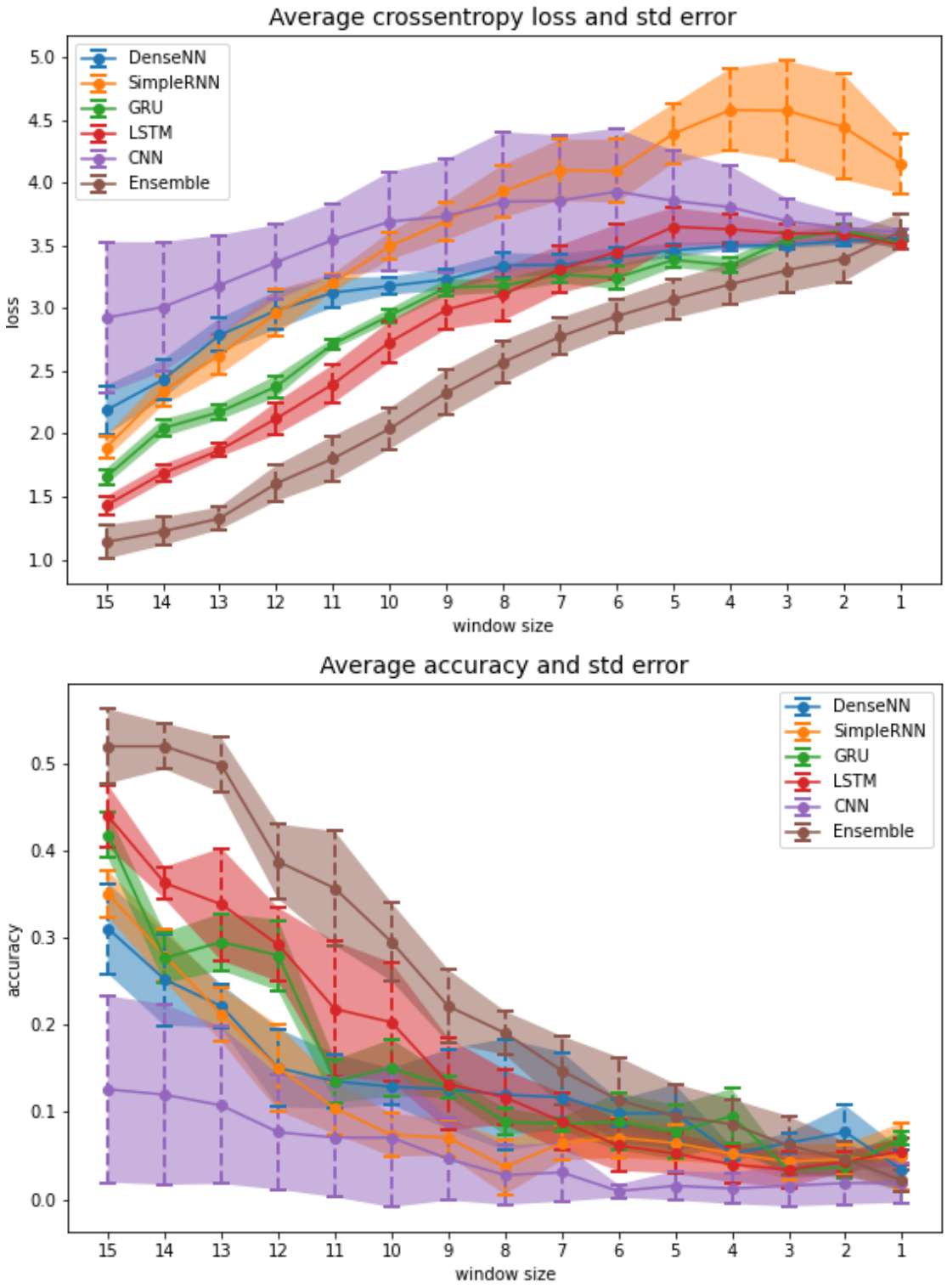



**Figure 26:** Networks performance changing input window size

# Chapter 4

# Online decoding

In this chapter I am reporting the pipeline I followed to develop the decoder to use in the online application. The difference from the task analysed in chapter 3 is that this decoder must work in a context where time is a constraint and the available information will be accumulated over time, but always limited by memory. The final purpose is to realise a demonstration where a prosthetic arm will replicate the hand and arm movement of a monkey, exploiting the brain activity of the animal recorded during the experiment. The difference is more important than it could seem, and actually will require reshaping the dataset and rethink the training process.

In the previous chapter I tested and compared a set of simple deep NNs which parameters were optimised to extract data from a spike binning brain activity. The networks chosen answered the needing of being (**1**) quick, (**2**) lightweight and (**3**) low resource consuming. Among all, the one that scored the highest results in the classification of the 'moving' epoch task, and that was also robust to a variation of the input window is the ensemble, composed of CNN stacked over LSTM. For those reasons, the ensemble will be used as the core for the BCI.

## 4.1 Methodologies

The first novelty introduced in this discussion is due to an online decoder having different characteristics than an offline one: the computational time, which was not a constraint in the latter, becomes one. Secondly, the amount of information is not constant: no samples are available at the beginning but they will accumulate over time. It is necessary then to decide what will be the buffer size: how much of the past recording will be kept in memory and be available for decoding purposes. In this proof of concept, constraints are only hypothetical, since no real hardware is available, so a small buffer that stored the last 10 measurements was simulated.

Note that a *Numpy* array of type *float32* with shape $(552, 10)$ weight on memory 22Kb. This seemed a reasonable number of recordings, short enough to be stored on a small memory, but long almost as the whole 'moving' phase, that in average lasted 15 steps. To set up a simulation based on this assumption it is mandatory to reshape the dataset to be composed of windows shaped as the buffer that the network can learn from during the training phase. The last introduction of this section is the needing for decoding the state of the trial besides the target object. This is necessary to possibly replicate the monkey task, that the BCI can predict when the movement must start and finish.

### 4.1.1 Sliding window to generate dataset

The first step consisted in generating the new dataset necessary to train the online decoder. It is worth recalling that the set of recordings used as starting point counted 628 trials, of which 432 were kept because they were associated with non-special objects (the ones selected at the beginning, see Figure 11; 6 shapes x 4 sizes x 12 samples each = 432). Each trial was described by a matrix of 552 channels and 139 time bins lasting around 40ms. Since the classifier model will be required to analyse an input of only 10 time steps, it was necessary to cut smaller windows from the whole trial. This reshaping of the trials was performed through the *sliding window* approach described in Figure 27: to simulate a buffer that progressively stored past recordings, a copying window with the desired dimensions is placed on top of the first column of the trial window (one column represents a collection of the 552 neurons activity in the past 40ms); then that column and the 9 before it are copied and stored separately. Afterwards, the window is slid by one step and the procedure is repeated until the last possible position. For the first 9 time steps for which there are not enough past recordings to fill the copying window, the missing information was replaced by *0 paddings*. Such a procedure results in the generation of 139 smaller windows with shape $(552, 10)$ from the whole trial matrix. However it would be a great mistake to consider this transformation an easy way to drastically increase the dataset dimension: two consecutive windows shares 90% of the information, and although the new column could change the meaning, it is important to assume the two to be in general highly correlated. NN learning from correlated examples is likely to overfit around them instead of learning features. To prevent overfitting problems to distort the results, a preventive division in training, validation and testing sets was performed *before* the sliding window transformation: the trials were grouped according to their target object, and from each group, a 70-15-15 division was applied. Therefore, three sets were finally obtained: training set with 228 trials, validation and test with 72 each (40032, 10008, 10008 respectively after reshaping).

During the slicing procedure, every window was also associated with three

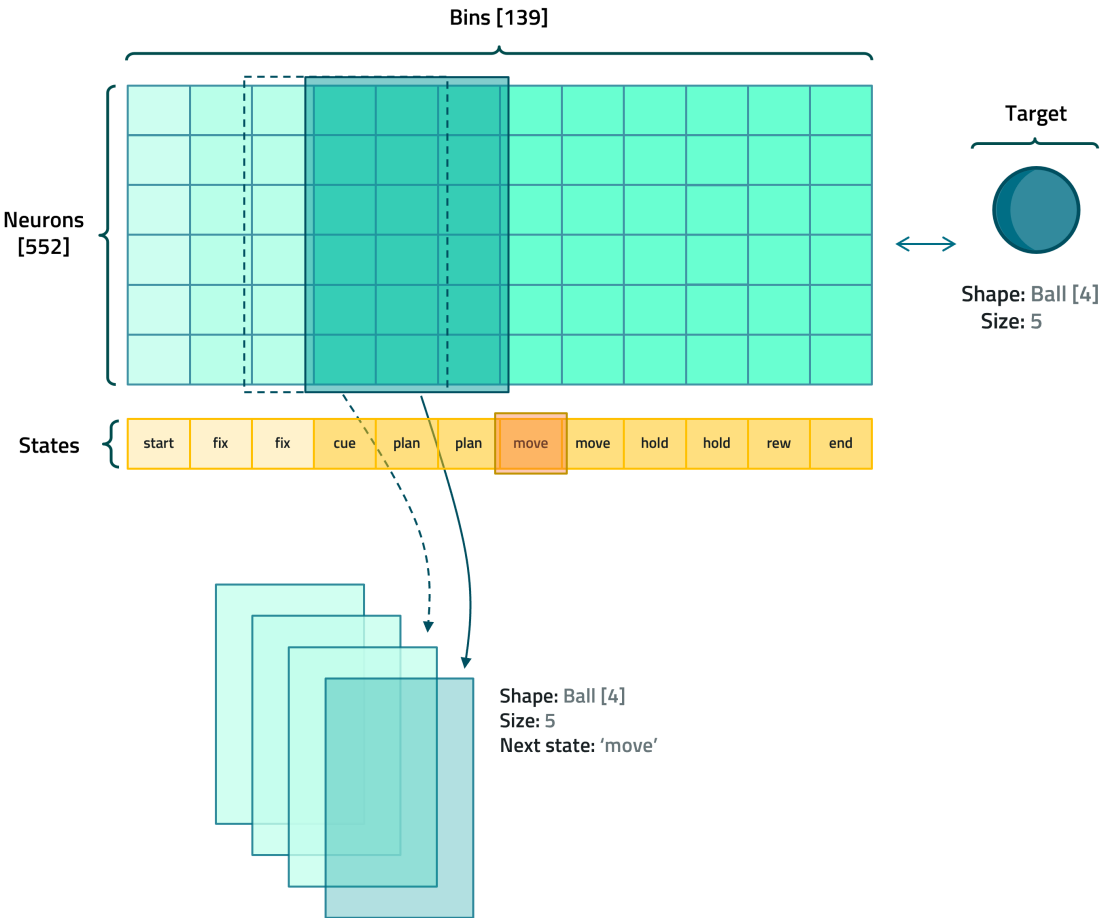



**Figure 27:** Sliding window transformation to simulate progressive buffered recordings

distinct labels: (**i**) object shape, (**ii**) object size, (**iii**) next trial state. It is worth explaining the meaning of the next trial state. In order to replicate the real experiment, the decoder should learn when to approach ('movement' phase), to lift ('holding' phase), and to lay ('ending' phase) the target object. Therefore, I associated to each window, while slicing, the trial epoch of the next time bin, or 'end' if the experiment was over. The decision of splitting object attributes shape and size in two different labels is due to make it easier for the machine to predict a single feature at a time, to have more examples per class, and possibly increase the performance from those reported in chapter 3. However, this choice will require distributing the decoding task among three individual networks, each one of them specifically trained to predict one label.

### 4.1.2 Dataset analysis

Before proceeding with the training phase of the models it is good practice to assess the goodness of the training datasets (similarly to what was done in chapter 2), in particular in this section I am discussing their balance, in terms of classes representation.

#### States training set

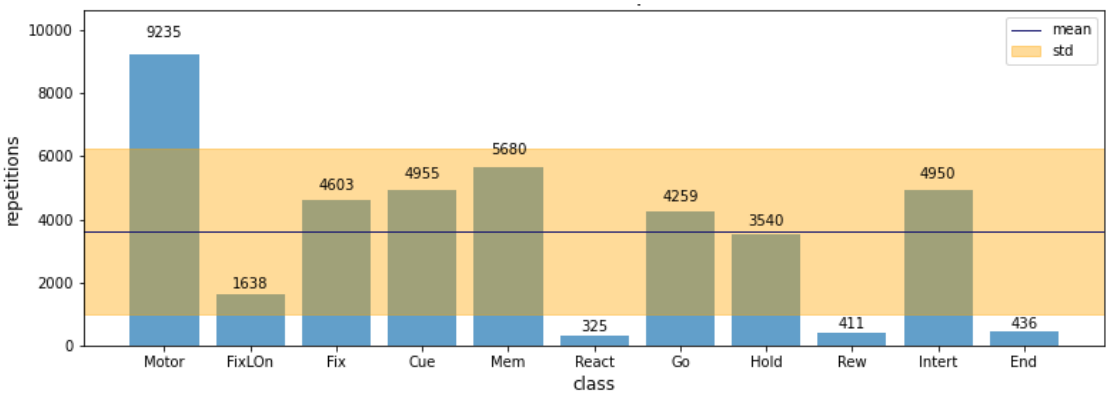



**Figure 28:** Distribution of states in the examples in the new dataset

The first analysis regards the distribution of the state, the label introduced specially for this task. In Figure 28 we can observe that the number of examples relative to the experiment epochs is highly irregular. However, this is not a surprise as it reflects what we already observed in the original dataset (see Figure 12). It is worth mentioning that some small classes disappeared because they were the first state of the trial which is lost during the sliding window transformation.

Although the state's distribution is coherent with the reality since some epochs actually lasted longer than others, it is not desirable to have the decoder knowing it. Way better is to force the BCI to learn actual features to discriminate states instead of learning the timing. It is not uniquely determined when a dataset can be claimed to be balanced, but my guess was that 72% of standard deviation in representation was not acceptable, and a 10% was the minimum I required.

The first transformation proposed, is to group the states in macro classes, as researchers of the original paper of this dataset did [9]. So I grouped classes in the following macro labels: *Waiting* (Start + Rest + Motor + FixLOn + Fix), *Cue*, *Planning* (Memorise), *Moviment* (React + Go), *Hold* (Hold + Reward), *End* (Intert + End).

As expected, grouping labels brought to the smother distribution displayed in Figure 29. The variance got reduced to 60% proving that the idea was winning,

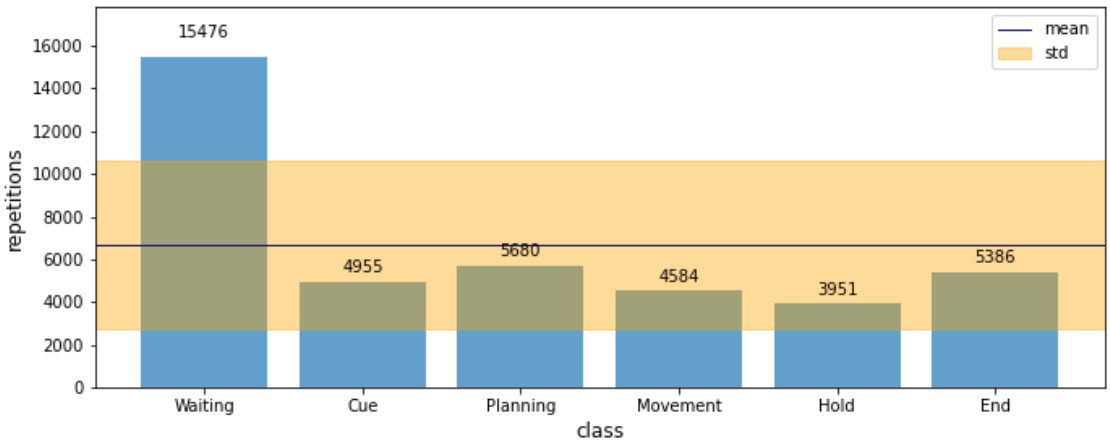



**Figure 29:** Distribution of grouped states in the new dataset

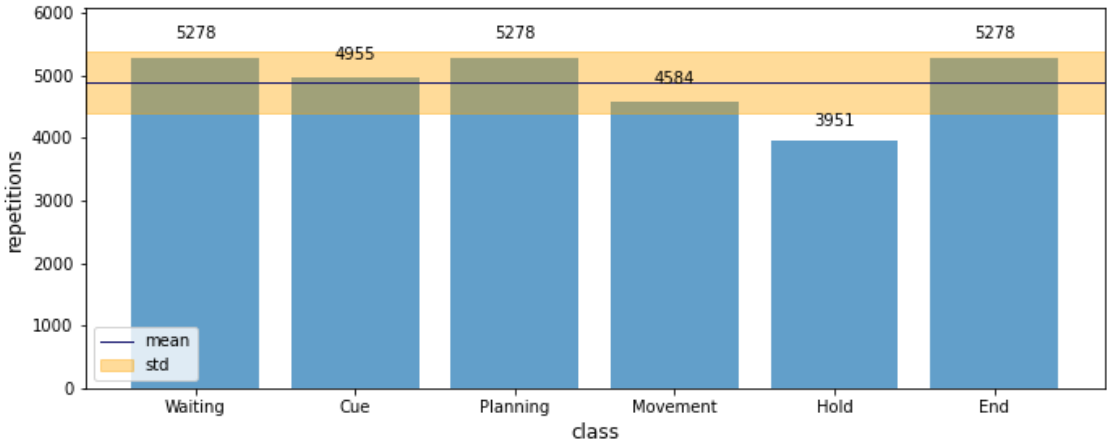



**Figure 30:** Distribution of states grouped and capped in the new dataset

but still, the difference between the main classes and the others is huge. However, aside from the 'Waiting' class, the variation of the number of examples among the others seems almost acceptable. Therefore, the last transformation I applied was capping the number of elements in the major classes. To do so, I evaluated the minimum number of elements to guarantee a variance of 10% and used that as the superior limit for those classes having more examples, randomly removing the exceeding examples. The final result is reported in Figure 30.

### Shapes training set

Before proceeding to the analysis of shapes and sizes distribution in the dataset, it is worth recalling that not all the samples are useful. Even if all the small windows are associated with an object, that was the target of the original whole trial, there are many of them for which it is not possible that the animal knows the object yet. For example, a 'waiting' window can be associated with a 'ball' of size '4' but this is meta-information that the decoder should not exploit. Actually, there is no chance that the brain activity of the monkey in the 'waiting' phase can already show anything characteristic of the object, and including them in the decoding training would only badly affect results. For this reason, all the windows associated with an epoch prior to the 'Cue' phase are excluded from the shape and size dataset. Besides those, also all the windows associated with an epoch subsequent to 'Movement' were discarded, since by the time the simulation reaches the holding phase, the object must have already been decoded in order to replicate the grasp. The examples that survived this filtering step are 21178. After this two specifications, we can check the distributions of objects shapes in Figure 31 and sizes in Figure 32. Variance in the shapes examples is 3%, with a light abundance of boxes. However, dropping some elements from that group seemed a waste of data, and on the contrary, the difference is not that huge to think it will negatively bias the decoder learning phase.

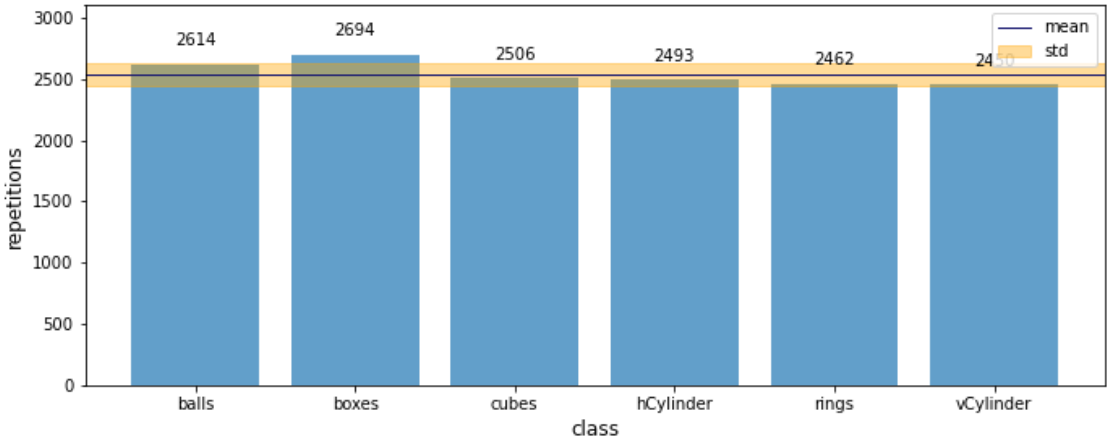



**Figure 31:** Shapes distribution in the classification dataset for online decoding

### Sizes training set

Optimal is the situation for the representation of the size, that it is clearly balanced and did not require any further transformation.

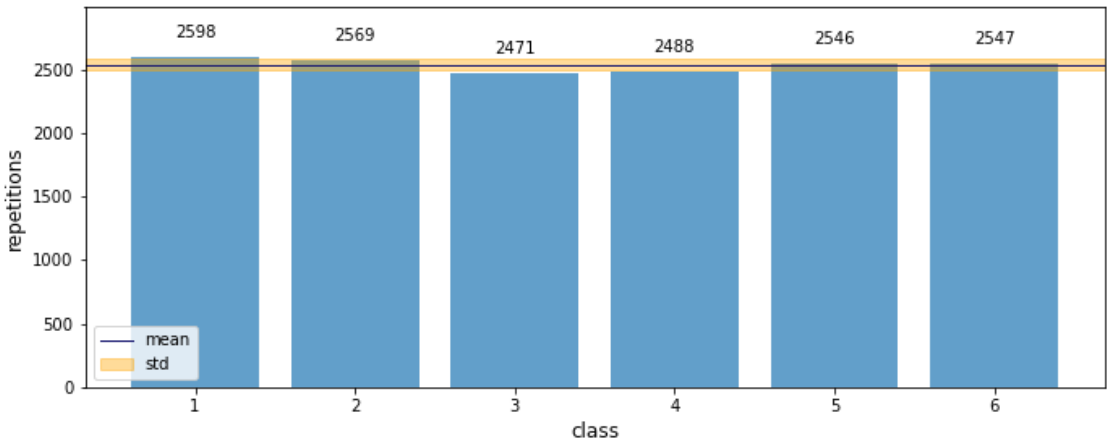



**Figure 32:** Sizes distribution in the classification dataset for online decoding

## 4.2 Results

To obtain the three working decoders, I built three ensemble models. Each one of them was initialised with the optimal parameters found in chapter 3. The main difference is the number of epochs that were not followed, since it was fit around a different classification task. Instead, I used a fixed maximum number of 100 epochs and added an *early stopping* condition. Early stopping is a *callback* function provided by *keras* that allow monitoring after every epoch one metric on the validation set, and to introduce a condition to interrupt the training and restore the weight that performed the best on the validation set. In this case, the training is stopped if the validation loss stops decreasing for more than 10 consecutive epochs (minimum sensed improvement is 0.01). After training those networks on the respective training sets, and keeping the weights that maximised performance on their validation set I tested them on the previously unseen data of the testing set. For each model, I am displaying its performance as accuracy confusion matrix and distance from the diagonal.

### 4.2.1 State decoder

No feasibility tests were previously performed to regress the experiment epoch. Therefore it was a surprise to find the extremely good results achieved by this network, displayed in Figure 33. The accuracy score for the exact macro-state is 89.1% that rises up to 99.5% if one class error is tolerated. The network successfully managed to learn proper weights to extract features to distinguish the phase of the experimental trial and will be used to decide whether there is the will to start moving the prosthetic limb.

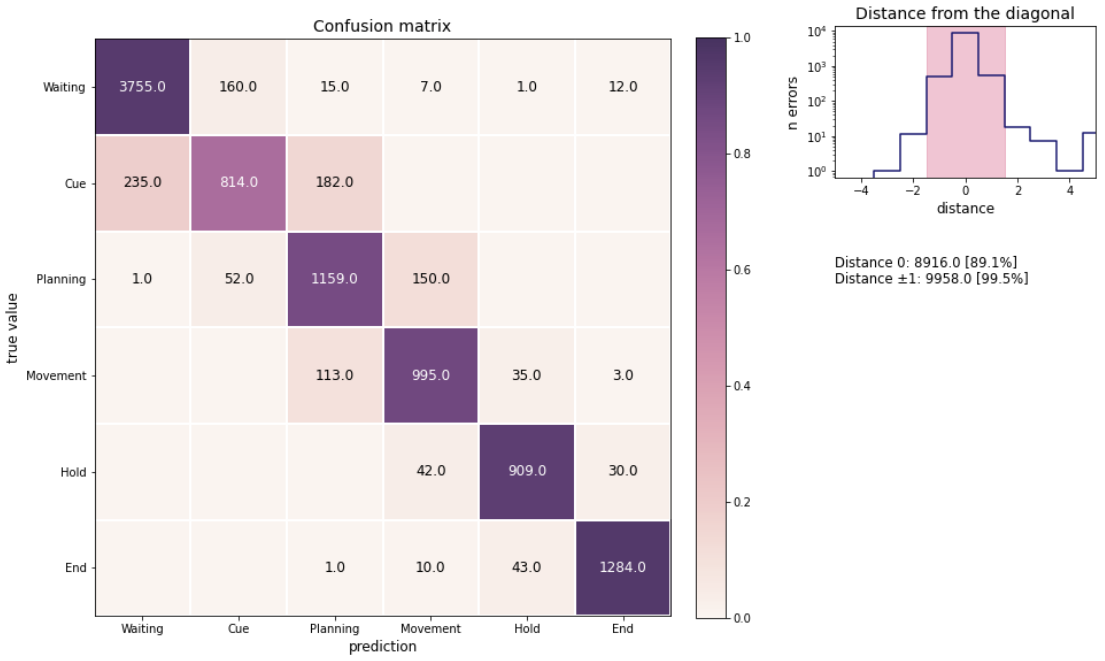



**Figure 33:** Testing phase of state decoder

### 4.2.2 Shape decoder

Figure 34 prove that shape decoding was the most successful among all the tasks. With 92.2% of accuracy, it can be definitely claimed that the object physics induced variation in the grasp positioning evident enough to be uniquely recognised. In this case, the distance from the diagonal is not meaningful, being the classes different and not correlated.

### 4.2.3 Size decoder

Lastly, I am analysing the performance of the size decoder represented in Figure 35. Those are the poorest ones among all with the network correctly classifying only the 25.4% of the testing samples. However, the accuracy increases to 62.2% if allowing one class tolerance. Looking at results in confusion matrix, it is interesting to observe that the decoder mainly learnt to distinguish two main classes: small and big. It is worth recalling, that the network was trained to distinguish the size of every object at once as I assumed this decision would make the network to be less rigid and pushed it to learn features related to the closure of the hand and fingers. Another possibility could have been to train an individual decoder for each shape.

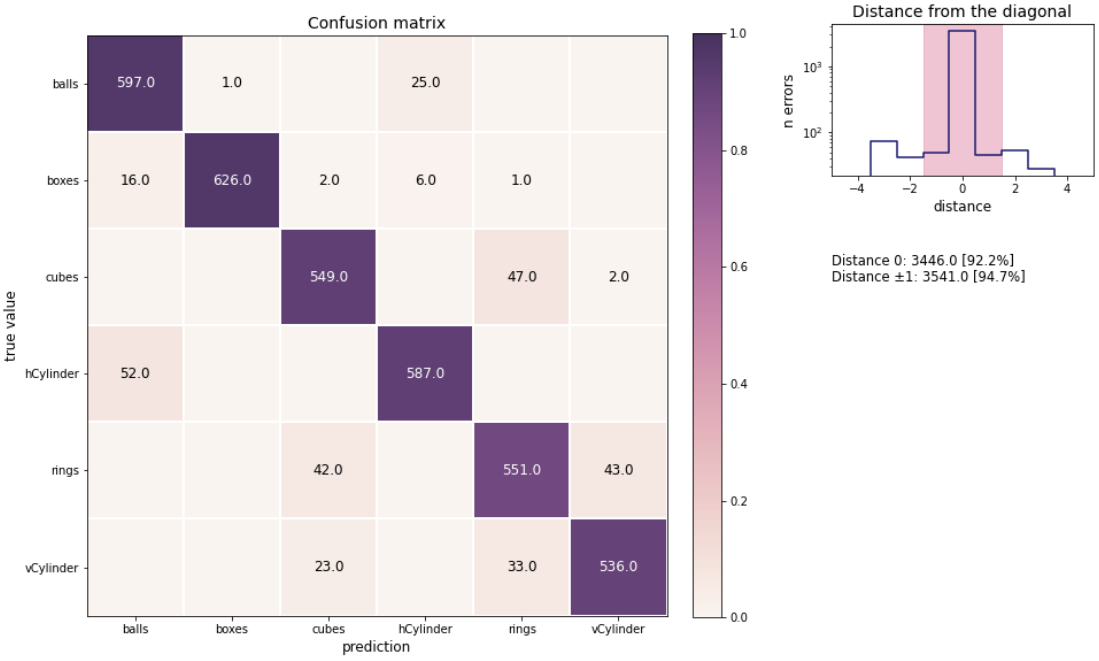



**Figure 34:** Testing phase of shape decoder

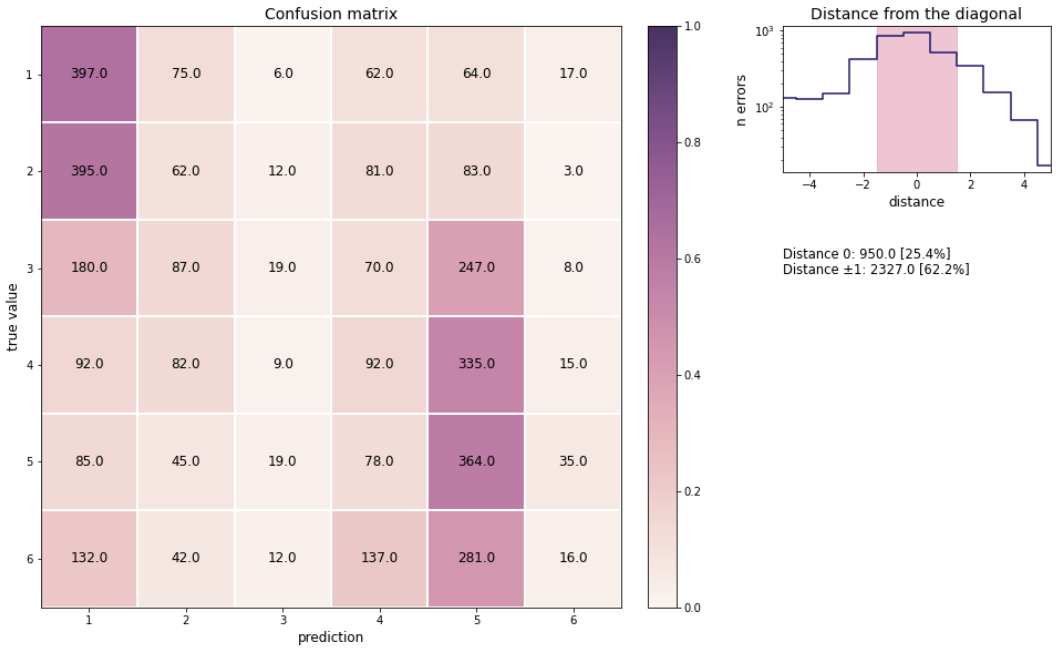



**Figure 35:** Testing phase of size decoder

# Chapter 5

# Demonstration

## 5.1 The bionic arm prototype

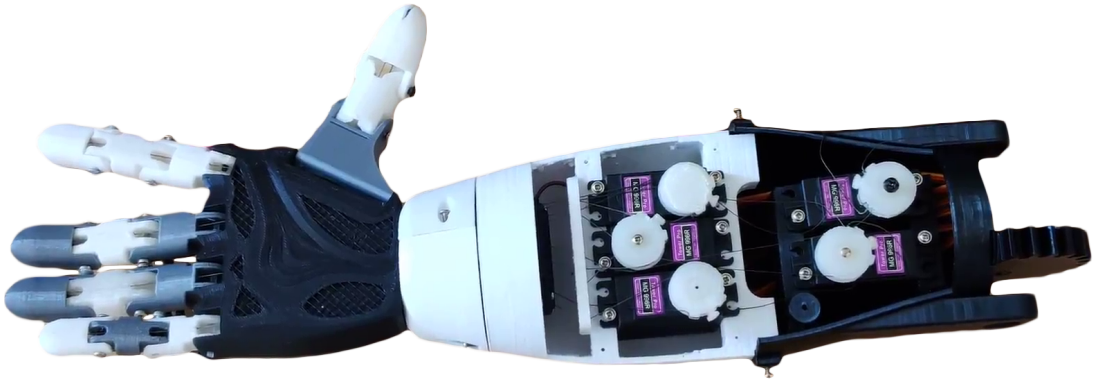


**Figure 36:** Bionic arm picture

The decoders trained in chapter 4 are tested in demonstration where the controlled arm is the one pictured in Figure 36. This prototype has been designed, printed and assembled by *Arvin Selvand*, a former master student collaborating with MMG, whose work is still under publication. This device counts six degrees of freedom: one for each finger and one for the wrist. Fingers are actuated by a motor whose torque pull a wire causing the contraction of the finger, while the wrist is directly connected to a step-motor responsible for its rotation. The arm is controlled by an externally plugged Arduino system, which receives and unpacks the command signal payload -composed by an array of six bytes- and activates motors according to it. The communication happens via radio antennas *Idas Probe* following the IEEE 802.15.4 protocol. *Johan Engstrand* is the PhD student who assessed the packet loss affecting this communication.

## 5.2 Software

To replicate the real time reading of brain activity the demonstration proceed in a loop, set up as following:

```
1: procedure SIMULATION(trial_binned_window)
2:     ▷ trial_binned_window is the matrix with shape (n_channels=553, n_bins=139)
3:     i ← 0                                          ▷ Index of the first iteration set to 0
4:     ▷ Execution
5:     while i ≤ n_bins do
6:         ▷ Slice the so far seen brain activity from the whole matrix
7:         window ← trial_binned_window[:, max(0, i − 9) : i + 1]
8:         ▷ Using the state decoder regress the next instant epoch
9:         next_epoch ← state_decoder(window)
10:        ▷ Check if it is possible to classify object
11:        if next_epoch in [Cue, Planning, Movement] then
12:            obj_shape ← shape_decoder(window)
13:            obj_size ← size_decoder(window)
14:        end if
15:        ▷ Generate and send payload
16:        payload ← action_dict(next_epoch, obj_shape, obj_size)
17:        i ← i + 1                                              ▷ Update index
18:    end while
19: end procedure
```

The trial used is selected from the pool of experiments used to generate the testing set of the online decoding.

The payload is composed of 8 bytes, each one of them corresponding to a value [0-255] to actuate one of the motors. The first five values control the closure of the fingers, the sixth the rotation of the wrist, the seventh the extension of the elbow and eighth the rotation of the shoulder on the sagittal plane (one more DOF for the shoulder rotation on the frontal plane could be included but it would imply harder kinematics).

All the values are just simulated since the arm used to test the software was only prototypical, with 6 DOF, therefore is no chance to approach, lift and lay the object, and no fine granularity in fingers closure. As a proof of concept, I associated the approaching phase to the closure of one finger, the holding as two fingers, lifting as all fingers, and then backward for laying and retreat.The software can however easily accept new kinematics, finely tuned to describe the movement, in future development.

# Chapter 6

# Conclusion

In this thesis, I investigated the ability of deep neural networks to extract features and decode lightly-preprocessed recordings from the brain, collected with intracortical implanted arrays. I compared many networks archetypes finding that recurrent ones were those performing the best. In particular, a combination of a convolutional step to reduce the channels dimension and then an LSTM was successful in classifying the target object shape in an online application with around 92% of accuracy. Two additional models, with the same architecture, were trained to decode the object size and the trial state (i.e. the intent of extending, lifting, laying or retreating the arm) and scored, respectively, 25% and 89% accuracy.

Given these results, I can claim that the execution of a grasp and the target-specific hand configuration uniquely shape the neural activity. Consequently, it is possible to decode the will to perform a movement and some information about the target object. The main concern regards the classification of the object size, which performed poorly. The idea of grouping the items by size had the intent to push the decoder extracting information about the closure of fingers or the tension of tends. However, this had not happened; it is not easy to understand whether it is a problem of the quantity of data or if, perhaps, that is not information expressed in the brain area under analysis. In conclusion, results suggest that deep neural networks represent a promising tool for decoding intracortical neural recordings. However, some adjustments must be included, especially regarding the data collection.

## 6.1 Concerns about the quality of dataset

The first problem that I want to address regards the quality of the dataset. It is well known that the performance of a neural network is highly affected by the number of examples available to learn from and, if those are few, it will be harder

for the network to learn the pattern underneath them. Since the beginning of this work, 432 trials seemed to be not much, and the overfitting problem was addressed in multiple ways. For example, the classes were grouped in order to have more representation for each one, the networks were kept simple (since the more complex they are, the more likely to memorise the data), and dropout regularisation was largely used. This problem applies also to the classification dataset obtained after the sliding window. Despite this transformation drastically increased the number of samples, it is questionable to consider it a proper data augmentation. In fact, the smaller windows mostly overlap, without really introducing diversity that might force the network to generalise. Despite the overall good performance obtained by shape and state decoders, their training history shows that the models only learnt for a couple of epochs and then overfitted. Moreover, the size decoder learnt to distinguish only between two-dimensions. The question about whether this is all the information that can be extracted from brain data or if more can be obtained through different software or more data, will remain unanswered in this discussion.

## 6.2 Concerns about the real world application

The second doubt regards the possibility to replicate the monkey movement in real life. Being the demonstration mostly theoretical, the capability of the buffer was only hypothetical; it is not guaranteed that it could be really possible to work with a ten-step long window. Perhaps, a smaller one would be necessary, probably affecting results. Moreover, the simulation took few milliseconds (around 37ms) when decoding the state only, but it took around 100ms when decoding all three labels. It is hard to state whether these results are limited by the available hardware or if it is necessary to change the software. Probably, a real-time application would require a single network that classifies at once the state, shape and size.

## 6.3 Future work

Given the concerns related to dataset limitation and real-time application, there is a future step I would propose. A deeper and more complex network might be able to find deeper patterns in the data that were missed so far, possibly increasing the accuracy for all the labels. However, increasing the complexity of the network would lead to overfitting results, so this could be allowed only by having more recordings from the same animal. Another advantage of training a deeper network is that possibly a single one might be sufficient to classify state, shape and size simultaneously, highly reducing the decoding time.

# Appendix A

# Dataset

The Figure 37a was attached to the dataset I received and describe all the sub phases of the trials. All of them was finely monitored with the aid of a multitude of sensors and cameras that kept track of the hand position of the animal. It is worth mentioning that between two consecutive trials there was a waiting time during which the platform rotated to present a new object. After all the six object on that platform were presented in random order to the monkey, an operator manually substituted the turning table. In total, there were eight turning tables, which are those displayed as columns in Figure 37b. Therefore, in most of the cases, consecutive trials presented the same shape of object with different size.

| States | |
|---|---|
| 0 | Trial Start |
| 1 | Handrest |
| 2 | Optional Reward |
| 3 | Motor |
| 4 | Fixation |
| 5 | Cue |
| 6 | Memory |
| 7 | Reaction |
| 8 | Grasping Go |
| 9 | Hold |
| 10 | Reward |
| 11 | Intertrial Interval |
| 12 | Wait after Error |
| 13 | Trial End |
| 14 | Pause |
| 15 | Eye Calibration Task |
| 16 | Move motor manually |
| 17 | Fixation Light On |

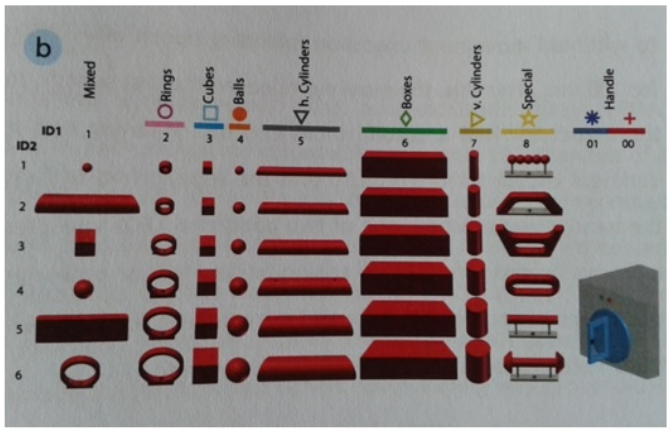



**Figure 37:** Complete list of trial states and target objects

# Appendix B

# Hyperopt set-up

Hyperopt required to set up a search space from which sample the parameters to test. A brief explanation of the main functions is following: hp.quniform define values between a min and a max with a step; hp.choice define values among those in a list.

**DNN**

- 'num_neurons': hp.quniform('num_neurons', 32, 640, 32)
- 'frac_dropout': hp.quniform('frac_dropout', 0., 0.5, 0.05)
- 'n_epochs': hp.quniform('n_epochs', 5, 50, 5)
- 'batch_size': hp.quniform('batch_size', 2, 26, 2)

**SimpleRNN**

- 'num_units': hp.quniform('num_units', 16, 256, 16)
- 'frac_dropout': hp.quniform('frac_dropout', 0., 0.5, 0.05)
- 'n_epochs': hp.quniform('n_epochs', 5,50,5)
- 'batch_size': hp.quniform('batch_size', 2, 26, 2)

**GRU**

- 'num_units': hp.quniform('num_units', 16, 256, 16)
- 'frac_dropout': hp.quniform('frac_dropout', 0., 0.5, 0.05)
- 'n_epochs': hp.quniform('n_epochs', 5, 50, 5)
- 'batch_size': hp.quniform('batch_size', 2, 26, 2)

**LSTM**

- 'num_units': hp.quniform('num_units', 16, 256, 16)
- 'frac_dropout': hp.quniform('frac_dropout', 0., 0.5, 0.05)
- 'n_epochs': hp.quniform('n_epochs', 5, 50, 5)
- 'batch_size': hp.quniform('batch_size', 2, 26, 2)

**CNN**

- 'num_filters': hp.quniform('num_filters', 32, 256, 32)
- 'kernel_size_1': hp.quniform('kernel_size_1', 50, channels-50, 50)
- 'kernel_size_2': hp.quniform('kernel_size_2', 3, window-3, 3)
- 'pool_size': hp.choice('pool_size', [2, 0])
- 'frac_dropout': hp.quniform('frac_dropout', 0., 0.5, 0.05)
- 'n_epochs': hp.quniform('n_epochs', 5, 50, 5)
- 'batch_size': hp.quniform('batch_size', 2, 26, 2)

**Ensemble**

- 'num_filters': hp.quniform('num_filters', 4, 64, 4)
- 'size': hp.quniform('size', 16, ceil(channels/2), 16)
- 'activation': hp.choice('activation', ['relu', 'elu'])
- 'frac_dropout_1': hp.quniform('frac_dropout_1', 0., 0.5, 0.05)
- 'pooling_layer': hp.choice('pooling_layer', ['max', 'avg'])
- 'recurrent_layer': hp.choice('recurrent_layer', ['gru', 'lstm'])
- 'n_units': hp.quniform('n_units', 16, 256, 16)
- 'frac_dropout_2': hp.quniform('frac_dropout_2', 0., 0.5, 0.05)
- 'n_neurons': hp.quniform('n_neurons', 32, 512, 32)
- 'n_epochs': hp.quniform('n_epochs', 5, 50, 5)
- 'batch_size': hp.quniform('batch_size', 2, 26, 2)

# Bibliography


[1] Sun-Chong Wang. «Artificial neural network». In: *Interdisciplinary computing in java programming*. Springer, 2003, pp. 81–100 (cit. on p. 1).

[2] Suryakiran Navath et al. «How Brain-Computer Interfaces Are Leading To Advances In Neuroscience». In: *Journal of Neuroscience & Psychology* 1.1 (2021), pp. 1–2 (cit. on p. 1).

[3] Jacques J Vidal. «Toward direct brain-computer communication». In: *Annual review of Biophysics and Bioengineering* 2.1 (1973), pp. 157–180 (cit. on p. 2).

[4] Max O Krucoff, Shervin Rahimpour, Marc W Slutzky, V Reggie Edgerton, and Dennis A Turner. «Enhancing nervous system recovery through neurobiologics, neural interface training, and neurorehabilitation». In: *Frontiers in neuroscience* 10 (2016), p. 584 (cit. on p. 2).

[5] Jacques J Vidal. «Real-time detection of brain events in EEG». In: *Proceedings of the IEEE* 65.5 (1977), pp. 633–641 (cit. on p. 2).

[6] Jianjun Meng, Shuying Zhang, Angeliki Bekyo, Jaron Olsoe, Bryan Baxter, and Bin He. «Noninvasive electroencephalogram based control of a robotic arm for reach and grasp tasks». In: *Scientific Reports* 6.1 (2016), pp. 1–15 (cit. on p. 2).

[7] Yamik Mangukiya, Brij Purohit, and Kiran George. «Electromyography (EMG) sensor controlled assistive orthotic robotic arm for forearm movement». In: *2017 IEEE Sensors Applications Symposium (SAS)*. IEEE. 2017, pp. 1–4 (cit. on p. 2).

[8] Philip P Vu et al. «A regenerative peripheral nerve interface allows real-time control of an artificial hand in upper limb amputees». In: *Science translational medicine* 12.533 (2020) (cit. on p. 2).

[9] Stefan Schaffelhofer, Andres Agudelo-Toro, and Hansjörg Scherberger. «Decoding a wide range of hand configurations from macaque motor, premotor, and parietal cortices». In: *Journal of Neuroscience* 35.3 (2015), pp. 1068–1081 (cit. on pp. 2, 13, 15–17, 50).

[10] Neuralink. URL: https://neuralink.com (visited on 07/18/2021) (cit. on p. 2).

[11] B-Cratos project. URL: https://www.b-cratos.eu (visited on 07/18/2021) (cit. on pp. 2, 13).

[12] Jingles (Hong Jing). *Deep Learning in Brain-Computer Interface*. Towardsdatascience. 2020. URL: https://towardsdatascience.com/deep-learning-in-brain-computer-interface-f650d00268d0 (visited on 07/18/2021) (cit. on p. 3).

[13] Pietro Aricò, Nicolina Sciaraffa, and Fabio Babiloni. *Brain–computer interfaces: Toward a daily life employment*. 2020 (cit. on p. 3).

[14] Alexander Craik, Yongtian He, and Jose L Contreras-Vidal. «Deep learning for electroencephalogram (EEG) classification tasks: a review». In: *Journal of neural engineering* 16.3 (2019), p. 031001 (cit. on pp. 3, 8).

[15] Meel Velliste, Sagi Perel, M Chance Spalding, Andrew S Whitford, and Andrew B Schwartz. «Cortical control of a prosthetic arm for self-feeding». In: *Nature* 453.7198 (2008), pp. 1098–1101 (cit. on p. 3).

[16] Tom Carlson, Guillaume Monnard, and José del R Millán. «Vision-based shared control for a BCI wheelchair». In: *International Journal of Bioelectromagnetism* 13.ARTICLE (2011), pp. 20–21 (cit. on p. 3).

[17] Chethan Pandarinath, Paul Nuyujukian, Christine H Blabe, Brittany L Sorice, Jad Saab, Francis R Willett, Leigh R Hochberg, Krishna V Shenoy, and Jaimie M Henderson. «High performance communication by people with paralysis using an intracortical brain-computer interface». In: *Elife* 6 (2017), e18554 (cit. on p. 3).

[18] Neha Borulkar, Pravin Pandey, Chintan Davda, and Joyce Chettiar. «Drowsiness detection and monitoring the sleeping pattern using brainwaves technology and IoT». In: *2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC) I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC), 2018 2nd International Conference on*. IEEE. 2018, pp. 703–706 (cit. on p. 3).

[19] Alexandros T Tzallas, Nikolaos Giannakeas, Kostantinos N Zoulis, Markos G Tsipouras, Euripidis Glavas, Katerina D Tzimourta, Loukas G Astrakas, and Spyridon Konitsiotis. «EEG classification and short-term epilepsy prognosis using brain computer interface software». In: *2017 IEEE 30th International Symposium on Computer-Based Medical Systems (CBMS)*. IEEE. 2017, pp. 349–353 (cit. on p. 3).

[20] Choon Guan Lim, Tih Shih Lee, Cuntai Guan, Daniel Shuen Sheng Fung, Yudong Zhao, Stephanie Sze Wei Teng, Haihong Zhang, and K Ranga Rama Krishnan. «A brain-computer interface based attention training program for treating attention deficit hyperactivity disorder». In: *PloS one* 7.10 (2012), e46692 (cit. on p. 3).

[21] Debashis Das Chakladar, Shubhashis Dey, Partha Pratim Roy, and Debi Prosad Dogra. «EEG-based mental workload estimation using deep BLSTM-LSTM network and evolutionary algorithm». In: *Biomedical Signal Processing and Control* 60 (2020), p. 101989 (cit. on p. 3).

[22] Raffaella Folgieri and Roberto Zampolini. «BCI promises in emotional involvement in music and games». In: *Computers in Entertainment (CIE)* 12.1 (2015), pp. 1–10 (cit. on p. 3).

[23] Mikhail A Lebedev and Miguel AL Nicolelis. «Brain–machine interfaces: past, present and future». In: *TRENDS in Neurosciences* 29.9 (2006), pp. 536–546 (cit. on pp. 3, 8, 12).

[24] Sheital Bavishi, Joseph Rosenthal, and Marcia Bockbrader. «Chapter 17 - Neuroprosthetics». In: *Rehabilitation After Traumatic Brain Injury*. Ed. by Blessen C. Eapen and David X. Cifu. Elsevier, 2019, pp. 241–253. ISBN: 978-0-323-54456-6. DOI: https://doi.org/10.1016/B978-0-323-54456-6.00017-7. URL: https://www.sciencedirect.com/science/article/pii/B9780323544566000177 (cit. on p. 4).

[25] Nicholas G Hatsopoulos and John P Donoghue. «The science of neural interface systems». In: *Annual review of neuroscience* 32 (2009), pp. 249–266 (cit. on p. 4).

[26] Ali Al-Saegh, Shefa A Dawwd, and Jassim M Abdul-Jabbar. «Deep learning for motor imagery EEG-based classification: A review». In: *Biomedical Signal Processing and Control* 63 (2021), p. 102172 (cit. on pp. 4, 11).

[27] Wikipedia. URL: https://en.wikipedia.org/wiki/Electroencephalography (visited on 07/20/2021) (cit. on p. 4).

[28] Wikipedia. URL: https://en.wikipedia.org/wiki/Motor_cortex (visited on 08/22/2021) (cit. on pp. 5, 6).

[29] O-Yeon Kwon, Min-Ho Lee, Cuntai Guan, and Seong-Whan Lee. «Subject-independent brain–computer interfaces based on deep convolutional neural networks». In: *IEEE transactions on neural networks and learning systems* 31.10 (2019), pp. 3839–3852 (cit. on p. 5).

[30] Donald L Schomer and Fernando Lopes Da Silva. *Niedermeyer's electroencephalography: basic principles, clinical applications, and related fields*. Lippincott Williams & Wilkins, 2012 (cit. on p. 6).

[31] Peter Henderson. *Implanted intracortical electrodes as chronic neural interfaces to the central nervous system*. Tech. rep. PeerJ PrePrints, 2015 (cit. on pp. 6, 7).

[32] Wikipedia. URL: https://it.wikipedia.org/wiki/MEMS (visited on 08/22/2021) (cit. on p. 7).

[33] John P Seymour, Fan Wu, Kensall D Wise, and Euisik Yoon. «State-of-the-art MEMS and microsystem tools for brain research». In: *Microsystems & Nanoengineering* 3.1 (2017), pp. 1–16 (cit. on p. 7).

[34] Michael S Lewicki. «A review of methods for spike sorting: the detection and classification of neural action potentials». In: *Network: Computation in Neural Systems* 9.4 (1998), R53 (cit. on p. 7).

[35] Ming-Ai Li, Jian-Fu Han, and Li-Juan Duan. «A novel MI-EEG imaging with the location information of electrodes». In: *IEEE Access* 8 (2019), pp. 3197–3211 (cit. on p. 9).

[36] Ward Fadel, Csaba Kollod, Moutz Wahdow, Yahya Ibrahim, and Istvan Ulbert. «Multi-class classification of motor imagery EEG signals using image-based deep recurrent convolutional neural network». In: *2020 8th International Winter Conference on Brain-Computer Interface (BCI)*. IEEE. 2020, pp. 1–4 (cit. on p. 9).

[37] Hyeon Kyu Lee and Young-Seok Choi. «Application of continuous wavelet transform and convolutional neural network in decoding motor imagery brain-computer interface». In: *Entropy* 21.12 (2019), p. 1199 (cit. on p. 9).

[38] Jiayao Sun, Jin Xie, and Huihui Zhou. «EEG Classification with Transformer-Based Models». In: *2021 IEEE 3rd Global Conference on Life Sciences and Technologies (LifeTech)*. IEEE. 2021, pp. 92–93 (cit. on pp. 9, 30).

[39] Hauke Dose, Jakob S Møller, Helle K Iversen, and Sadasivan Puthusserypady. «An end-to-end deep learning approach to MI-EEG signal classification for BCIs». In: *Expert Systems with Applications* 114 (2018), pp. 532–542 (cit. on pp. 9, 30, 40).

[40] Francisco M Garcia-Moreno, Maria Bermudez-Edo, María José Rodríguez-Fórtiz, and José Luis Garrido. «A CNN-LSTM deep Learning classifier for motor imagery EEG detection using a low-invasive and low-Cost BCI headband». In: *2020 16th International Conference on Intelligent Environments (IE)*. IEEE. 2020, pp. 84–91 (cit. on pp. 10, 30).

[41] Na Lu, Tengfei Li, Xiaodong Ren, and Hongyu Miao. «A deep learning scheme for motor imagery classification based on restricted Boltzmann machines». In: *IEEE transactions on neural systems and rehabilitation engineering* 25.6 (2016), pp. 566–576 (cit. on p. 10).

[42] Yousef Rezaei Tabar and Ugur Halici. «A novel deep learning approach for classification of EEG motor imagery signals». In: *Journal of neural engineering* 14.1 (2016), p. 016003 (cit. on p. 10).

[43] Edward M Schmidt. «Single neuron recording from motor cortex as a possible source of signals for control of external devices». In: *Annals of biomedical engineering* 8.4 (1980), pp. 339–349 (cit. on p. 12).

[44] Aneesha K Suresh, James M Goodman, Elizaveta V Okorokova, Matthew Kaufman, Nicholas G Hatsopoulos, and Sliman J Bensmaia. «Neural population dynamics in motor cortex are different for reach and grasp». In: *Elife* 9 (2020), e58848 (cit. on p. 12).

[45] Joshua I Glaser, Ari S Benjamin, Raeed H Chowdhury, Matthew G Perich, Lee E Miller, and Konrad P Kording. «Machine learning for neural decoding». In: *Eneuro* 7.4 (2020) (cit. on p. 12).

[46] Joshua I Glaser, Matthew G Perich, Pavan Ramkumar, Lee E Miller, and Konrad P Kording. «Population coding of conditional probability distributions in dorsal premotor cortex». In: *Nature communications* 9.1 (2018), pp. 1–14 (cit. on p. 13).

[47] Karen Simonyan and Andrew Zisserman. «Very deep convolutional networks for large-scale image recognition». In: *arXiv preprint arXiv:1409.1556* (2014) (cit. on p. 13).

[48] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. «Understanding deep learning (still) requires rethinking generalization». In: *Communications of the ACM* 64.3 (2021), pp. 107–115 (cit. on p. 13).

[49] Peiran Gao, Eric Trautmann, Byron Yu, Gopal Santhanam, Stephen Ryu, Krishna Shenoy, and Surya Ganguli. «A theory of multineuronal dimensionality, dynamics and measurement». In: *BioRxiv* (2017), p. 214262 (cit. on p. 13).

[50] R Quian Quiroga, Zoltan Nadasdy, and Yoram Ben-Shaul. «Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering». In: *Neural computation* 16.8 (2004), pp. 1661–1687 (cit. on p. 17).

[51] Neuralensemble. URL: https://neuralensemble.org/neo/ (visited on 07/25/2021) (cit. on p. 18).

[52] David H Wolpert and William G Macready. «No free lunch theorems for optimization». In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82 (cit. on p. 24).

[53] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016 (cit. on pp. 26, 28, 29).

[54] James Bergstra, Daniel Yamins, and David Cox. «Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures». In: *International conference on machine learning*. PMLR. 2013, pp. 115–123 (cit. on p. 32).

[55] Will Koehrsen. *A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning*. towardsdatascience. 2018. URL: https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f (visited on 08/13/2021) (cit. on p. 32).

[56] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. «Algorithms for hyper-parameter optimization». In: *Advances in neural information processing systems* 24 (2011) (cit. on p. 32).

[57] Alexey Serov. *Tutorial on hyperopt*. Kaggle. 2019. URL: https://www.kaggle.com/fanvacoolt/tutorial-on-hyperopt (visited on 08/13/2021) (cit. on p. 32).