POLITECNICO DI TORINO

Master's Degree in Biomedical Engineering

Master's Degree Thesis

Transfer Learning strategies for time robust neural decoding in a Brain-machine interface

Supervisors

Dott. Paolo VIVIANI

Prof. Valentina AGOSTINI

Prof. Marco GHISLIERI

Candidate

Myriam LUBRANO

July 2023

Summary

Recent and continuous advancements in neuroengineering and Machine Learning demonstrates the huge potential of Brain-machine interface in the field of neuroprosthetics. This rapidly evolving technology aims to provide innovative solutions to people affected by disabilities, in order to restore motor, sensory, and cognitive functions. This is the goal of B-Cratos project, whose purpose is the development of a closed-loop neural interface for controlling a robotic hand prosthesis also capable of providing sensory feedback to the patient. Neural decoding is possible thanks to Deep Learning models trained using high-performance computing resources on datasets acquired from the German Primate Center (Deutsches Primatenzentrum, DPZ), that can classify signals recorded via implanted microelectrode arrays. DPZ researchers recorded the neural activity of two macaque monkeys trained to perform a grasping task with a series of objects of different shapes and sizes. These signals were pre-processed and used in previous works for the development of a classifier. For this purpose, a Bidirectional Recurrent Neural Network was trained to successfully identify the objects grasped by the monkeys, simulating a real-time decoding.

In this work, different *transfer learning* strategies were implemented in order to exploit the knowledge acquired by the pre-trained classifier in a model that can be used on new recording sessions, subsequent to the first one. The possibility to effectively transfer the information learned from a pre-trained model would represent a significant advantage in the use of BMIs, considering the high variability of neural signals and the need to recalibrate the device to maintain high performance over time. Before implementing the transfer learning, the two datasets used were appropriately reduced, in order to present a common dimensionality: the different dimensionality was due to the application of offline spike-sorting algorithms independently on the two sessions. Feature extraction was performed through different models and their performances were evaluated in terms of final accuracy achieved after the fine-tuning. Furthermore great importance was attributed to a *convergence analysis*: this analysis was conducted to evaluate the classifier's ability to quickly learn the neural patterns relevant for object classification.

Among all the models developed for the reduction and consequent classification, the

Partial Reduction model showed the best results, consisting in a single dense layer, with a latent space of the same dimension of the data from the first training session, preceding the layers responsible for classification whose weights can be setting as trainable or non-trainable parameters. This network showed to outperform a classifier trained from scratch in the classification task involving 37 objects with different shapes and sizes, both for the accuracy achieved and for the convergence speed. Specifically, the variant with frozen layers reported an accuracy of 45% and reached the 30% accuracy threshold in only 9 training epochs, compared to the reference that achieved a 37% final accuracy and required 45 epochs in order to get to the threshold. Despite the limitations presented by the datatset, this work showed that an appropriate weights initialisation can contribute to better and faster re-training, providing promising results for the implementation of transfer learning strategies in neural decoding models.

Table of Contents

Acronyms

AI

Artificial intelligence

AIP

Anterior intraparietal cortex

BMI

Brain-machine interface

BRNN

Bidirectional recurrent neural network

CAE

Convolutional autoencoder

CEC

Constant errror carrousel

CNN

Convolutional neural network

CNS

Central nervous system

DL

Deep learning

DNN

Deep neural network

ECoG

Electrocorticography

EEG

Electroencephalography

FAT-IBC

Fat intra-body communication

FMA

Floating microelectode array

FN

False negative

DPZ

Deutsches Primatenzentrum

GPU

Graphics processing unit

HPC

High-performance computing

LDA

Linear discriminant analysis

LM

Language model

LSTM

Long short-term memory

M1

Primary motor cortex

MAE

Mean absolute error

MEG

Magnetoencephalography

ML

Machine learning

MSE

Mean squared error

NHP

Non-human primate

NN

Neural network

NLP

Natural language processing

PA

Action potential

PCA

Principal component analysis

PMv

Ventral premotor cortex

PNS

Peripheral nervous system

ReLU

Rectified linear unit

RNN

Recurrent neural network

SCI

Spinal cord injury

SGD	Stochastic Gradient Descent
TL	Transfer learning
TN	True negative
TP	True positive

Chapter 1

Introduction

There are several pathologies that lead to the loss of motor control and these can have a significant impact on the quality of life for patients and their families. Limited or absent motor functions can be caused by disorders affecting the nervous system, such as cerebral palsy (CP) and neuromuscular diseases, as well as amputations or spinal cord injuries (SCI). These conditions can impair patients' ability to move, communicate, and participate in daily activities, making it difficult to perform basic tasks and maintain independence.
Innovative technologies such as Brain-Machine Interfaces (BMIs) have been developed in order to improve the quality of life for patients with disabilities.

BMIs are devices that allow to translate brain activity into commands that can be used to control external devices, such as prosthetic limbs or rehabilitation orthoses. This type of technology represents a promising solution for rehabilitation, aiming to restore lost functionalities and provide autonomy to the patient.
One of the main goals of neural interfaces used in the biomedical field is to provide functional support systems for people with disabilities, for example for controlling *neuroprosthetics*. In this field, neuroprosthetics aim to restore the functions associated with voluntary movements, which can be compromised due to various neuromotor pathologies, such as tetraplegia and amyotrophic lateral sclerosis (ALS). In these subjects, the communication pathway between the motor cortex and muscles is interrupted, so solutions like myoelectric prosthetics are not applicable, however, the brain signal which constitutes the source of the missing motor output can still be directly recorded.

Deep Learning (DL) has recently found wide use in the development of BMIs, due to its ability to identify complex patterns within the data and for the possibility to implement transfer learning. DL models also enhance performances in classification tasks and allow to automate the feature extraction process [1].

1.1 Basics of neuroanatomy and neurophysiology

1.1.1 Central and peripheral nervous system

The nervous system is the set of organs and structures that allow signals to be transmitted between different parts of the body and to coordinate their actions and functions [2]. It is divided into the Central Nervous System (CNS) and the Peripheral Nervous System (PNS): the CNS is mainly made up of the brain and spinal cord, which integrate the information coming from the other organs and from the external environment and process appropriate reactions, while the PNS mainly consists of nerves, which extend from the CNS and branch throughout the body. Nerves are responsible for transmitting nerve signals between the brain and the rest of the body, allowing to control muscle movements and receive sensory information. *Efferent neurons* refers to the nerve connections that carry information relating to movement or other functions from the CNS to the periphery, whereas *afferent neurons* carry sensory signals from periphery to the CNS.

1.1.2 Brain anatomy

From an anatomical standpoint, brain is divided into two hemispheres and each of them has four lobes:

- Frontal lobe, primarily responsible for the control of voluntary movements, for personality and planning abilities;
- Parietal lobe, that contains areas dedicated to receiving somatosensory input and proprioception;

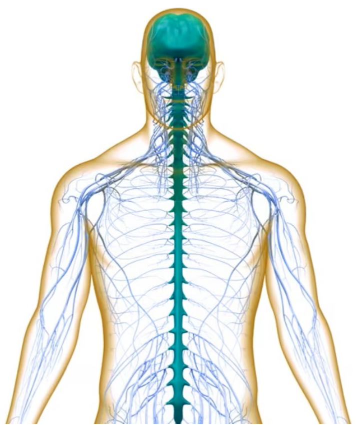


Figure 1.1: Human nervous system. [3]

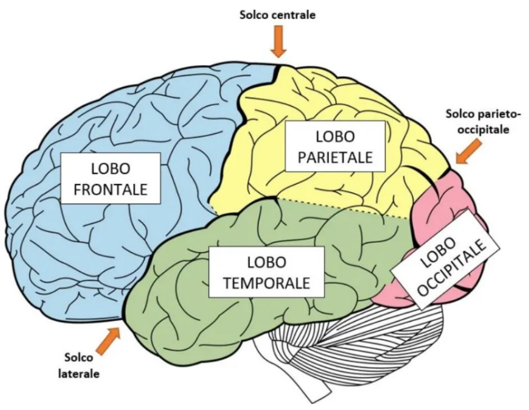


Figure 1.2: Division of the cerebral cortex into four lobes by sulci. [4]

- Temporal lobe, that plays a key role in perception and interpretation of auditory stimuli and language comprehension;
- Occipital lobe, that hosts the primary and secondary visual cortex, making it responsible for the processing of visual signals.

The outer surface of the brain is the cerebral cortex, which presents numerous grooves, identifying hundreds of sulci and gyri. These considerably increase the surface of the cortex itself and help to identify the different aforementioned lobes: the frontal lobe is immediately anterior to the central sulcus, while the parietal lobe is located behind it. The temporal lobes lie on the lower side of the brain and are separated from the parietal and frontal lobes by the lateral sulcus. Lastly, the occipital lobe, at the back of the brain, is isolated from the others by the parieto-occipital and temporal sulci.

In particular, the following areas will be of particular interest for the rest of this work, as they will be the target of the BMI implant:

- The primary Motor cortex (M1) is located in the posterior part of the frontal lobe, anterior to the central sulcus (green in Figure 1.3). From a functional point of view, it is directly involved in the planning, control, and execution of voluntary movements of the body [5] and, in particular, of fingers;
- The Anterior Intraparietal Cortex (AIP) is a region of the posterior parietal cortex (in red in Figure 1.3). Beyond motor neurons, it has a prevalence of visual neurons, which make it fundamental in the processing of visual and spatial information relating to objects;
- The ventral Premotor Cortex (PMv) is located on the inferior lateral surface of the frontal lobe, anterior to area M1 (in blue in Figure 1.3). Specifically,

area F5 is connected to the previous AIP as it receives from it the visual information it uses in coding the specific grip and manipulation of an object. This information is then sent to the M1 cortex for execution [6].

1.1.3 Neurons and synapses

Neurons are the basic nerve cells of the nervous system and are responsible for transmitting nerve signals between cells. Neurons are specialised to receive, process, and transmit information in the form of electrical and chemical impulses [8].
They are composed of a cell body, called soma, and two types of cytoplasmic extensions: dendrites propagate the afferent signals to the soma, while axons carry the neuron's signals towards other cells.
Functionally, neurons can be divided into three types:

1. Sensory or afferent neurons;
2. Motor or efferent neurons;
3. Interneurons, which connect the first two types and that can also communicate with each other.

Communication between two neurons takes place at *synapses*, also known as synaptic junctions. These represent the site where nerve impulses pass from a pre-synaptic to a post-synaptic cell. The transmission of a nervous impulse can occur electrically or chemically:

- In chemical synapses, the transmission of signals is mediated by neurotransmitters because there is a discontinuity between the two cells. This extracellular space between the pre-synaptic and post-synaptic membranes is called synaptic cleft. Neurotransmitters can be both excitatory and inhibitory;

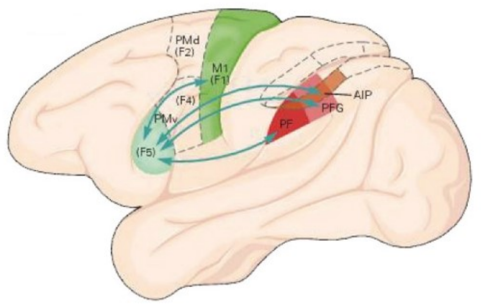


Figure 1.3: Activation pathway during grasping, involving M1, AIP, and F5 areas of the cerebral cortex. [7]

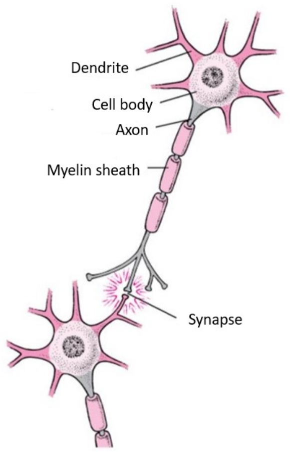


Figure 1.4: Neuron structure and synaptic junction between two nerve cells. [9]

- In electrical synapses, there is a direct current passage from one cell to another, thanks to the proximity or cytoplasmic continuity between the pre-synaptic and post-synaptic membranes. This type of transmission is faster.

1.1.4 Action potential

The Action Potential (AP), or neural spike, is a brief electrical impulse that occurs in nerve cells, generated by a rapid change in the membrane potential.
During the resting state, neurons exhibit a separation of electric charges across the cell membrane, consisting of an excess of positive charges on the outer surface. This charge difference is due to the presence of ion channels in the membrane, which regulate the passage of sodium and potassium, and is responsible for a resting potential approximately equal to -70 mV. In these conditions both sodium and potassium channels (voltage-gated channels) are closed and the concentration of K^+ ions is higher inside the cell, while the concentration of Na^+ ions is higher outside.
If the neuron is stimulated above the threshold potential of excitation, the membrane potential undergoes a variation that occurs in three phases: depolarisation, followed by repolarisation and a short period of hyperpolarisation:

- Depolarisation is caused by the opening of sodium channels, which allow the entry of Na^+ ions into the cell. As positively charged ions enter, the membrane potential reverses its polarity. During this change in polarity the membrane reaches a potential approximately equal to +40 mV;
- The repolarisation phase is caused by the closure of sodium channels and the opening of potassium channels. As a results, the membrane becomes less

permeable to sodium and this allows the Na^+ ions to come out, until the negative membrane potential is restored;

- Hyperpolarisation is the phase in which some potassium channels remain open, causing an increased permeability to potassium. This permeability results in a further reduction of the membrane potential, which reaches -90 mV. At this voltage, also potassium channels close and the membrane potential returns to its resting value.

These three events occur within a few milliseconds and represent a refractory period. By definition, the refractory period is the period of time right after the first AP during which a cell is incapable of generating a new one. There are two types of refractory periods, the absolute refractory one, which corresponds to the depolarisation and repolarisation of the membrane, and the relative refractory one, which corresponds to the hyperpolarisation: while during the first period the generation of a new PA is impossibile regardless of the intensity of the stimulus, during the second one a new depolarisation is possible only with a higher level of excitation. Consequently, neural spikes are isolated events that can't be summed and that can occur at a maximum frequency.

The action potential originates at the axon hillock and propagates along the axon without attenuation. The amplitude of the action potential is independent of the intensity of the stimulus, i.e., if the stimulus is high enough to exceed the threshold, the PA is generated with constant amplitude and shape. Furthermore, the shape of the PA is characteristic of each neuron.

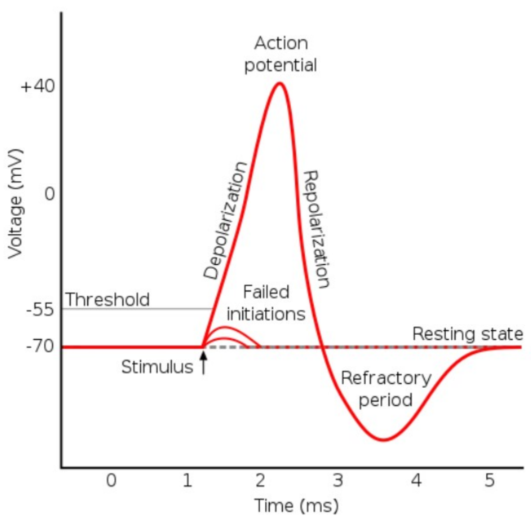


Figure 1.5: Trend of the membrane potential during the generation of the action potential. [10]

1.2 Brain-Machine Interfaces

The term Brain-Machine Interface (BMI) refers to a system that enables direct communication between the brain and an external device, bypassing the ordinary peripheral channels and directly exploiting the recording of brain signals. A BMI is therefore a system that measures the activity of the CNS and converts it into artificial outputs that replace, restore, enhance, or integrate the natural outputs of the CNS, allowing an individual to interact with the external environment [12].
One of the potential applications could be the restoration of a lost function due to injuries or diseases: for example, a person who has lost control of his limbs could use a BMI to drive a motorised wheelchair or move a robotic limb. Other applications involve the improvement of natural functions (such as enhancing attention in people driving a vehicle [13] or establishing the activation of an orthopedic device [14]) or the integration of the same (as in the use of an exoskeleton for safety at workplace [15]).

A BMI records the user's brain signals, extracts from them specific features, and translates those features into artificial outputs that act in the external world or on the body itself, therefore it has three basic components: a *signal acquisition* block, a *signal processing* block, and the *output device* (Figure 1.6). Signal acquisition techniques depend on the signal monitored and on the invasiveness of the electrodes used for the purpose: the type of acquisition allows to classify the BMI according to the scheme shown in the following subsection (1.2.1).
The signal processing block involves the features extraction and the decoding of the recorded signals, through a translation algorithm. Lastly, the output is represented by the external device of the BMI, which constitutes the target of the decoding but also a feedback that can be exploited by the brain or by the BMI itself [16].

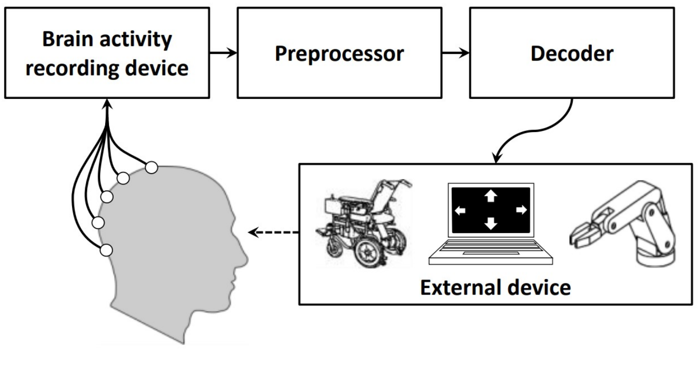


Figure 1.6: Schematic representation of the components of a BMI for controlling external devices. [11]

1.2.1 BMI classification

During the signal acquisition stage, two types of brain activity can be monitored: electrophysiological or hemodynamic brain activities.
Most BMIs exploit electrophysiological signals and, in this case, brain activity can be measured non-invasively, by electroencephalography (EEG), or invasively by electrocorticography (ECoG), with electrodes placed above or below the dura mater or directly into the brain via intracranial electrodes. This difference allows us to distinguish the BMIs into two classes (Figure 1.7):

- **Non-invasive.** They use sensors placed on the scalp in ordier to measure brain's electrical potentials (EEG) or magnetic field (MEG). Most recent BMIs obtain relevant information from brain activity through electroencephalography, which is widely used due to its high temporal resolution, relative low cost, and minimal risks for users. However, EEG-based BMIs rely on surface electrodes that lie on the scalp, thus, the quality of these signals is influenced by interposed layers as well as by background noise. An example is shown in Figure 1.8;

- **Invasive.** Due to the limitations of non-invasive acquisition, invasive recording methods such as electrocorticography (ECoG) or intracortical electrode recording have been introduced in an attempt to improve the quality of brain signals monitored by BMIs.
  Despite the need for high-resolution and low-interference signals, invasive BMIs require implantation of microelectrode arrays into the skull through surgery: this implies significant risks for the user, including the risk associated with surgery itself as well as issues related to infections or long-term

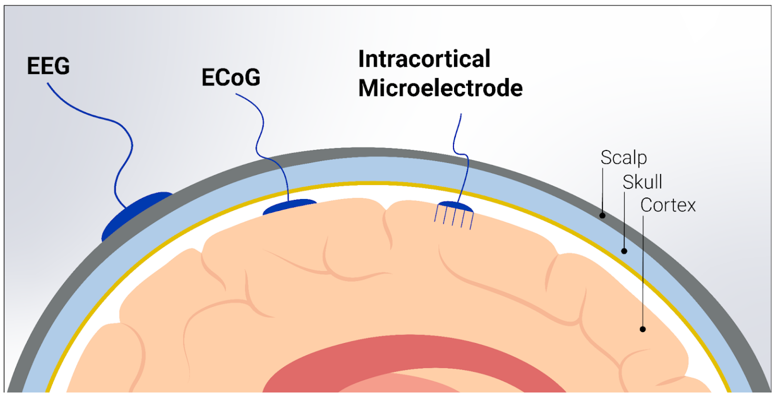


Figure 1.7: Comparison between neural signal acquisition techniques: EEG for non-invasive modality, ECoG for partially invasive modality, and intracortical microelectrodes implanted in the cortex for invasive modality. [17]

bio-compatibility. All these disadvantages severely limit their use outside of the experimental context.

A further classification of BMIs is based on the presence or absence of somatosensory feedback, which allows them to be distinguished into mono-directional and bi-directional.
In traditional mono-directional BMIs, the external device receives commands directly from signals deriving from the brain activity, without providing the user with a feedback on the action performed (except for visual feedback).
On the other hand, bi-directional BMIs combine this communication channel with a feedback loop that also allows information to flow from the external device back to the brain. These are the so-called *somatosensory neuroprostheses*. Sensations related to touch or proprioception constitute an important feedback, which can be sent to the peripheral nerves or directly to the somatosensory cortex [18]. It's important to highlight that somatosensory feedback provides not only sensorimotor but also cognitive benefits to users, as demonstrated by Klaes et al. [19] and by Preatoni et al. [20].

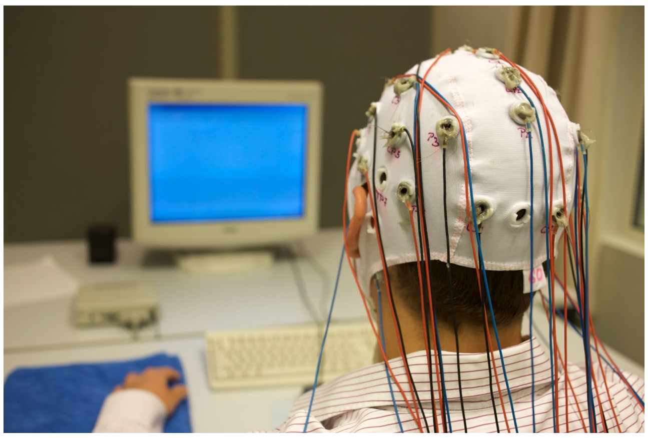


Figure 1.8: Example of BCI with non-invasive acquisition system. [17]

1.3 B-Cratos project

This master thesis was carried out at the LINKS Foundation, a research institute born from the collaboration between Compagnia di San Paolo and Politecnico di Torino. LINKS operates in applied research, digital technology and innovation, with active projects at national and international level.
This work is part of B-Cratos (Wireless Brain-Connect inteRfAce TO machineS), a European project funded by the EU Horizon 2020 research and innovation program

under grant agreement 965044, that aims to develop a wireless, bidirectional and battery-free BMI to restore hand functionality and touch to subjects with paralysis or amputation [21].
The main expected features and innovations are the following:

- **Design a proof-of-concept high-channel, high-speed, wireless brain implant capable of two-way communication without battery**
  The system features a battery-less, fully-implantable brain interface capable of sensing high-resolution neural signals and precisely stimulating cortical targets. The Utah Array is connected to a small biocompatible, hermetically-sealed implant containing custom electronics to detect, amplify, and digitise neuronal activity and deliver electrical stimulation. A wearable external module designed by NTNU researchers uses a novel wireless transmission technique to enable two-way (Figure 1.9);

- **Develop a general-purpose, high-speed communications platform technology through the fat tissue**
  Fat intra-body communication (FAT-IBC) is based on microwave propagation confined to the subdermal body fat with minimal interference from external electronic devices;

- **HPC based AI computing**
  Machine learning (ML) and deep learning (DL) algorithms for arm control and sensor stimulation will be implemented for pattern recognition and classification. Details are provided in Appendix C;

- **Artificial skin**
  The sensory system will employ a novel combination of tribo-electric nanogenerators (TENGs) and graphene-based hydrogels to provide a time-dependent force map in digital format;

- **Biomechatronic prosthetic upper**
  B-CRATOS will use the 5-axis Mia robotic arm from Prensilia s.r.l., an SME spin-off of Scuola Superiore Sant'Anna, which evolved (lighter, stronger, improved speed and force) from the IH2 Azzurra and which is used in research institutes worldwide (Figure 1.10).

B-Cratos's BMI can be classified as invasive due to its use of intracortical microelectrode arrays (Blackrock Microsystems Utah Array), which are connected to a biocompatible implant containing the electronics capable of processing the neural activity and providing stimulation. Wireless communication is ensured by an external wearable module, responsible for power supply and high-speed data transfer, in a two-way direction. The AI module has the task of translating and transmitting signals between the brain and the prosthesis in both ways: decoding neural signals taken from the implanted electrode arrays and, vice versa, encoding input signals from the hand through the sensorised skin.

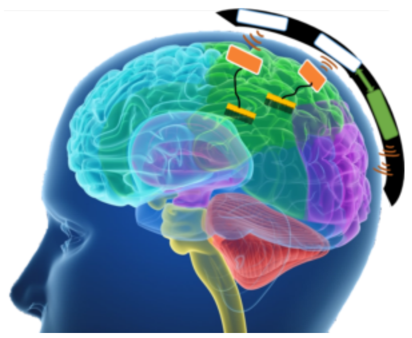


Figure 1.9: Schematic representation of the wireless communication between the implanted electrodes and the external module. [21]

1.3.1 Related works and intent of this thesis

Regarding the AI module task managed by LINKS, the purpose of training the Deep Learning models is to decode the neural activity from the motor cortex and translate it into meaningful commands in order to control the robotic prosthesis. In two previous thesis works [22, 23], starting from the neural signals acquired in a grasping activity session performed by a Non-Human Primate (NHP), a classifier was developed and trained to successfully predict the type of grip performed by the subject, simulating a real-time decoding.

The present thesis work aims to investigate our capability to transfer the information acquired by a classifier, pre-trained on a dataset acquired in a first specific session, on a new dataset acquired in a different day. This investigation is necessary because of the extreme variability of brain signals which, due to neuroplasticity[1], inflammatory processes or variations in the acquisition system, may not be sufficiently similar to the model training data, resulting in a reduction of the classifier performance through time.

[1]Neuroplasticity refers to the ability of the nervous system to reorganise its structure in response to internal or external stimuli. Numerous rehabilitation protocols aim to promote neuroplasticity through physical or cognitive exercises or through electrical and magnetic stimulation techniques.

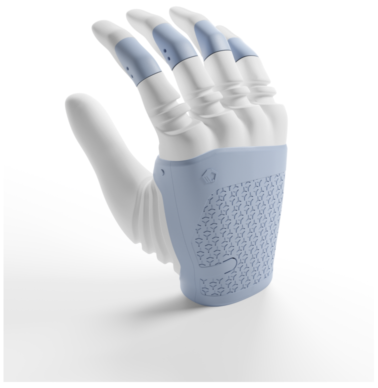


Figure 1.10: Mia Hand: robotic hand prosthesis developed by Prensilia. It allows 5 different types of grip, covering 80% of daily grips. [24]

Indeed, despite the numerous advances in the field of BMIs, most of the applications are still limited to experimental settings, because of the large amount of data required for the initial training and the frequency of re-training, making them hard to apply to everyday life: let us consider a user who intends to use a neuroprosthesis, but is forced to often recalibrate his BMI daily for two hours, possibly requiring assistance from a clinician, due to the variability of brain signals. This would take away time from actual usage to collect sufficient data for the neural network training. A more robust model or an appropriate transfer learning strategy would allow a drastic reduction of preparation and training time, leaving more time for utilisation. The intra-subject variability problem anticipates the more complex problem of inter-subject variability, whose overcoming would make it possible to reuse the previous knowledge acquired from one individual, adapting the BMI to other users.

The organisation of this thesis can be outlined as following: Chapter 2 briefly describes the concepts necessary for understanding the work; in Chapter 3 the dataset used in the project is presented, including the experimental set-up and the pre-processing techniques applied; Chapter 4 describes the methods and approaches adopted; in Chapter 5 the research results are presented and analysed. Finally, in the Chapter 6 a summary of the achieved results in the context of the work's objective is provided, along with the conclusions reached and any recommendations for future researches.

Chapter 2

Prior knowledge

In this chapter basic notions are provided for understanding the work and the techniques used and described in Chapter 4. Furthermore, the transfer learning concept is introduced in the context of Machine Learning.

2.1 Artificial Neural Networks

In Deep Learning field, an *Artificial neural network* (abbreviated in ANN or simply in NN for *Neural Network*) is a computational model composed of artificial *neurons*, loosely inspired by the simplification of a biological neural network [26]. Just as a biological one is made up of interconnected biological neurons, an NN is a mathematical computing model based on interconnected artificial neurons, which represent the nodes of the network itself.

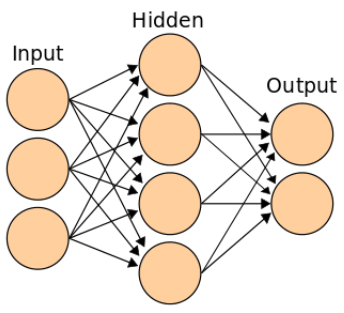


Figure 2.1: Architecture of the simplest ANN, consisting of an input layer with three neurons, a single hidden layer with four neurons and an output layer with two neurons. All layers are "fully connected", i.e., composed by neurons connected to all the neurons of the next layer. [25]

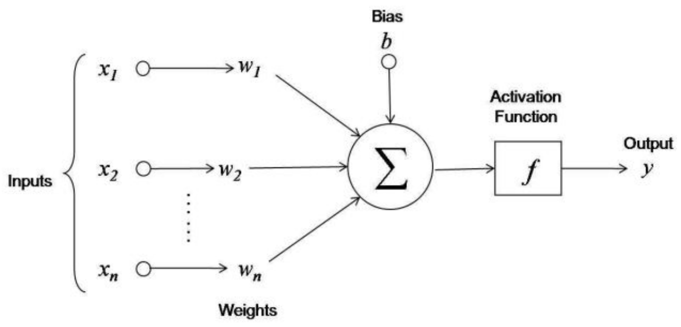


Figure 2.2: Structure of an artificial neuron. [27]

The basic structure of an NN consists of three layers: an input layer, a hidden layer, and an output layer, as shown in Figure 2.1. Signals are presented to the input layer nodes, each of which is connected with the next layer nodes. Every node processes the received signals through an activation function, which determines the output of the neuron itself, and transmits the result to the subsequent nodes (Figure 2.2). A neuron's input is therefore a weighted combination of the output from the previous layer neurons. This basic structure can be suitably complicated by varying the number of hidden layers and the number of nodes present on each of them.
The activation function aims to limit the neuron's output within a desired range of values and to introduce non-linearity into the model. Without non-linearity, the network would be unable to learn the more complex relationships in the data. Some of the most commonly used non-linear activation functions are:

- Sigmoid
  It restricts values in the range [0,1] and it is commonly used in the output layer for binary classification problems. The function is the following:

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1} \tag{2.1}$$

- Softmax
  It is a generalisation of the logistic function, that maps a K-dimensional vector x in a K-dimensional vector σ, whose values are in the range (0,1) and whose sum is equal to 1. It's mostly used in the output layer of multi-class classifiers. The function is the following:

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_{k=1}^{K}(e^{x_k})}, \text{per j = 1,...,K.} \tag{2.2}$$

- ReLU
  Rectified Linear Unit activation function (ReLU) is a fast and efficient one as it turns all the negative values to zero. It is generally chosen for hidden layers. Its equation is:
$$S(x) = x^+ = max(0, x) \tag{2.3}$$

During the training phase, the neural network modifies its randomly initialised weights based on the information it receives, in order to learn how to perform the required task. The most common learning algorithm for training an NN is the so-called *Backpropagation* [28], typically used in supervised learning, but whose application can also be extended to unsupervised networks. This algorithm compares, using a so-called *loss function*, the network's output with the target value and calculates the distance between the two. The resulting vector represents the gradient of the loss function and it is used to drive the training of the network. Thanks to the backpropagation algorithm this gradient is propagated backwards to recursively update network's weights, gradually converging the output values to the target ones. The overall training process is typically a flavour of the classical Stochastic Gradient Descent (SGD) [29].

The following sections provide a brief description of Recurrent Neural Networks and, specifically, of Long Short-Term Memory (LSTM) networks, as this is the model used to implement the classifier in the present work.

2.1.1 Recurrent Neural Network

A Recurrent Neural Network (RNN) is a type of artificial neural network that has a cyclic architecture, with neurons connected in a way that a node's output is the input of the same node, but in a subsequent time iteration (Figure 2.3). Therefore,

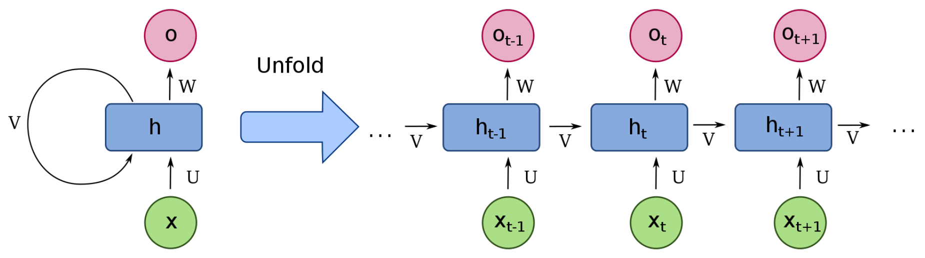


Figure 2.3: Diagram depicting the cyclic structure of an RNN: on the left the folded structure, on the right the unfolded structure, expanded through the procedure known as *backpropagation through time.* [30]

each neuron has the same weights in every time step, but the values of these weights are updated during the training of the network. The feedback transmits the output of a node back to its input forming a cycle and it allows the network to develop a short-term memory of information from previous time steps and use it to make decisions in the future. Indeed, these networks are widely used to process data sequences in applications such as natural language processing, speech recognition, machine translation, and other applications involving sequential data flows.

RNNs can also be bidirectional (BRNN): while traditional RNNs can recognise correlations between elements of a time sequence only in one direction (e.g., past), BRNNs can search for correlations in both temporal directions, in order to improve prediction.
Thus theoretically, traditional (or *vanilla*) RNNs can track long-term dependencies in input sequences. However, they present one computational limitation: when training a vanilla RNN using backpropagation, long-term gradients that are back-propagated can "vanish" (i.e., exponentially approach zero, stopping the network from further training) or "explode" (i.e. they can exponentially approach infinity), due to the accumulation of small/large gradients [31]. In order to mitigate these vanishing/exploding gradient effects more complex neurons, such as LSTM neurons, can be chosen.

2.1.2 LSTM networks

Long Short-Term Memory (LSTM) networks are a class of RNN networks capable of learning long-term and short-term dependencies [32]. The LSTM neuron has gates that control the information flowing into and out of the neuron itself, with the purpose of storing what is considered relevant for the task's resolution and must therefore be kept and what, on the contrary, it can be forgotten. The information is then filtered by the gates and the network learns the dependencies, both short-term and long-term, of the input sequences. LSTM networks can also solve the vanishing gradient problem, through a unit inside the cell called *Constant Error Carrousel* (CEC) [33], although the issue of exploding gradient problem still remains.

2.2 Transfer learning

Although Deep Learning models allow to solve extremely complex problems, the development of such models, especially in certain fields, requires the deployment of vast computing resources and time, as well as a huge amount of training data. For instance, as highlighted by Bender et al. [35], the training of large-scale language models (LMs) for *natural language processing* (NLP) entails the need for an assessment of environmental and economic impacts, focusing on the energy efficiency of the developed models.

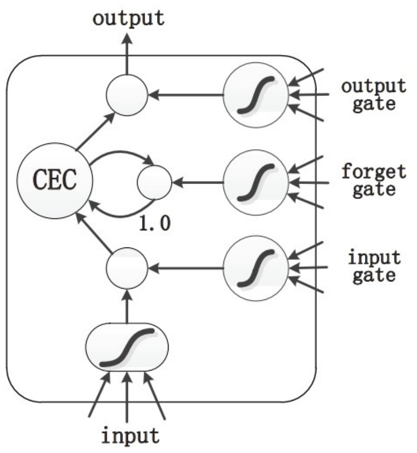


Figure 2.4: LSTM cell. It consists of an input gate, a forget gate, a Constant Error Carrousel (CEC), which stores information based on the value of the forget gate, and an output gate. [34]

This problem can be partially solved through *transfer learning*. In Deep Learning, the term transfer learning (TL) refers to a strategy employed to exploit a pre-trained neural network model on a large-scale dataset, in order to perform a new task, similar to or related to the one for which it was created. The goal is therefore to reuse knowledge from a model already trained or use it as a starting point for customisation and adaptation to new purposes. Thanks to this approach, it is possible to significantly optimise the already available resources, compared to creating and instructing a new model whenever there is a new requirement to satisfy.

This strategy is generally adopted in a supervised learning context and is widely used in image classification: for instance, a model designed to recognise a set of visual characteristics with the aim of discriminating between dogs and cats, can be further trained to discriminat two other categories, such as insects and fishes, but more quickly and effectively in terms of required amount of data and computational cost. The intuition behind is that if a model is trained on a sufficiently large and general dataset, it will effectively serve as a generic model for visual features and this allows to leverage the learned general feature maps without having to train a new model from scratch. However, to utilise transfer learning it is necessary to select a suitable pre-trained model as a starting point. Depending on the specific task, it may be appropriate making some changes to the model, such as fine-tuning the weights or freezing certain layers, in order to adapt it to the new specific task.

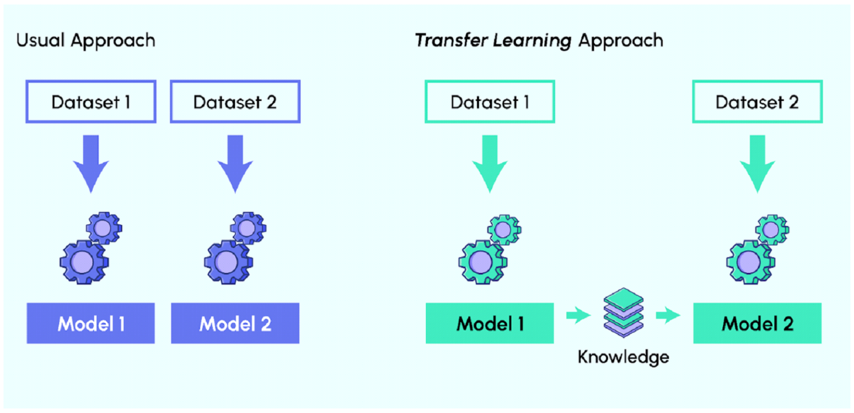


Figure 2.5: Comparison between traditional learning approach and transfer learning approach, which allows to transfer the acquired knowledge from Model 1 to Model 2, usable on a new dataset. [36]

Origins of transfer learning and today's applications

Despite the concept of transfer learning dates back to the 1970s (Bozinovski and Fulgosi, 1976) [37] and its modern definition was introduced in the 1990s, it is only with the advent of contemporary Machine Learning and the access to vast amounts of data that TL has become a popular and widely used strategy. Today, it is successfully applied in several areas, thanks to the increased computing power and the availability of libraries and DL models, which make it potentially accessible to everyone. Transfer learning algorithms are now frequently used, for example, in the following fields [38]:

- Computer vision, for object detection or image classification;
- Natural Language processing, for information extraction from text or machine translation;
- Speech recognition and automatic transcription;
- Medicine, in radiomics or diagnostic support.

2.2.1 Approaches

The two main strategies that can be applied to reuse a pre-trained model in order to transfer previous knowledge are as follows:

1. Feature extractor
   The pre-trained neural network is leveraged to effectively extract the most significant features from new input data by reusing its generally useful layers.

It's then sufficient adding downstream of that model a new classifier, which is trained from scratch specifically for the new task. An example can be found in computer vision, where the convolutional layer base of the pre-trained network is retained in order to extract features, while fully-connected layers of the classifier downstream of the model are replaced and trained from scratch;

2. Fine-tuning
   This involves customising a pre-trained network by keeping some layers fixed and selectively modifying the weights of other layers through a new training on a small amount of data. Freezing some layers allows to reduce training time and processing power [39], but this represents a complex technique.

These strategies can be implemented when it's not possible directly applying the pre-trained model on new data, due to their insufficient similarity to original dataset. The choice of the most suitable technique always depends on the specific application and the quality and quantity of available data.

Benefits

As anticipated, transfer learning offers many advantages over training a neural network from scratch, including:

- Saving time and computational resources: if the new task is similar to the original one, the pre-trained model can provide a good starting point, with a

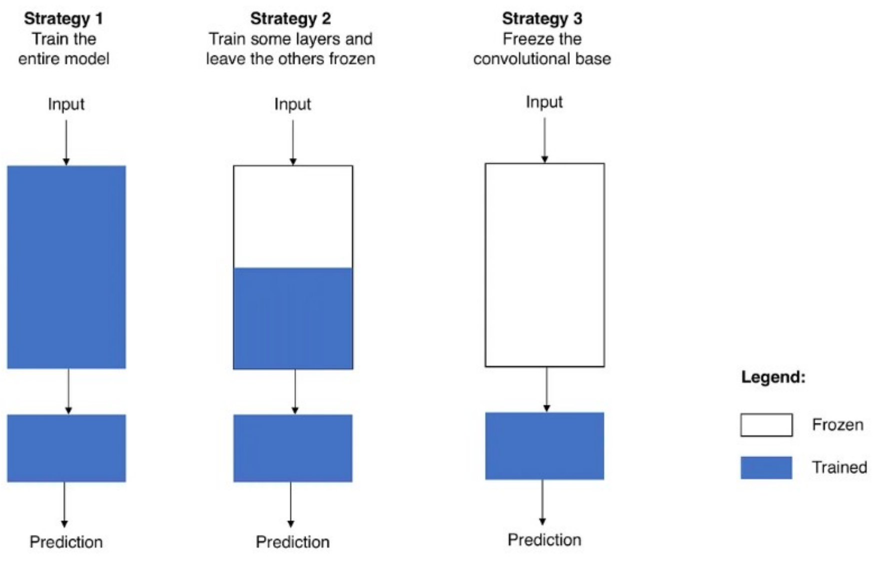


Figure 2.6: Fine-tuning strategies of a pre-trained model. [40]

number of already optimised parameters, reducing the time and computational resources required to train one from scratch;

- Access to high-quality models: it's possible to leverage other researchers's work and use high-quality models already trained on large amounts of labeled data;
- Performance improvement: transfer learning can help enhancing the performance of a model, especially when the new task is different but similar to the original task;
- Increased generalisation and reduction of overfitting: transfer learning can improve the model's ability to generalise, reducing the risk of overfitting.

As a result, transfer learning represents a huge opportunity to optimise deep learning processes, saving both time and energy.

Challenges and limitations

Despite the many advantages, transfer learning also presents some limitations and challenges, such as the choice of the starting model: if this model is too specific, it may not adapt well to the new task and the same would happen if, vice versa, it's too generic. Therefore, selecting the right starting model requires some experience and knowledge of the new task's features. Another challenge is the potential need to adapt the starting model for the new purpose. For example, it may be necessary to remove some layers of the initial model or add new ones. This operation requires a certain expertise in the manipulation of neural networks, too.
The main limitation of transfer learning lies in the fact that starting models are often trained on very large datasets, which may make it difficult using them in situations where only a small amount of labeled data is available. Finally, transfer learning may not be appropriate if the new task is significantly different from the original model's one: for instance, a model trained to recognise images of animals may not be effective in plant recognition.

2.3 Dimensionality reduction

In the Machine Learning field, many interesting problems exhibit a high dimensionality and this leads to several challenges. When dealing with high-dimensional datasets, it becomes difficult to identify patterns among the data under analysis and the problem known as *curse of dimensionality* indicates that the amount of samples needed to describe a dataset grows exponentially with the input dimensionality (i.e., the number of input features) [41]: the larger the volume and complexity of data, the more time and memory the algorithm takes to process them. Furthermore, in common experience we are accustomed to visualising data in two or three dimensions, while increasing dimensionality makes visualisation complex and hinders effective analysis. To overcome this problem, several researchers have devised techniques to reduce the size of datasets, making them more easily representable and, in some cases, computationally more manageable.

The reduction of dimensionality represents one of the data pre-processing operations used in the context of machine learning and it is exploited in order to eliminate from the dataset redundant (highly correlated), less or not important information for solving the problem. Thus this operation allows to synthesise the available information into a size suitable for the specific task and to represent the data in a lower and more interpretable dimensionality (e.g., to visualise the data in a 3D or 2D plot). It's important to highlight that, while dimensionality reduction allows to compress the volume of data, reducing computational time and memory necessary for the storage, on the other hand this can degrade the information and the predictive performance of the learning algorithm [42].

2.3.1 Feature selection and feature extraction

Dimensionality reduction methods are commonly divided into feature selection and feature extraction (Figura 2.7):

- Feature selection techniques enable to identify, from the overall set of starting features, a smaller subset to be used in the development of the model;
- Feature extraction transforms data from a high-dimensional space to smaller

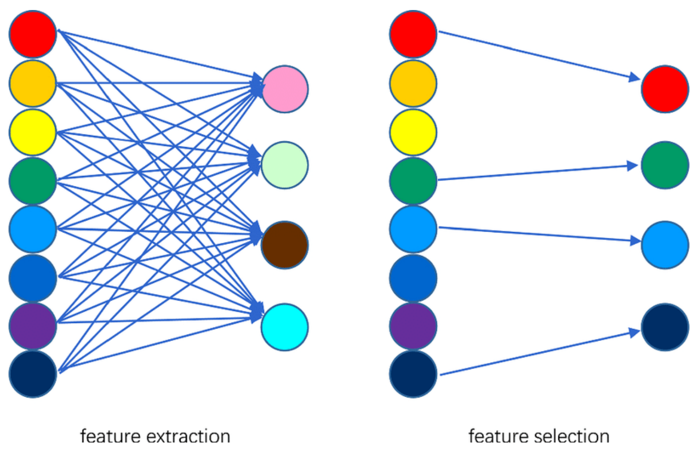


Figure 2.7: Comparison between two dimensionality reduction techniques: on the left the feature extraction, on the right the feature selection. [43]

space. This transformation can be linear, such as in Principal Component Analysis (PCA) or Linear Discriminant Analysis (LDA), but there are also non-linear techniques for performing dimensional reduction.

So, in conclusion, feature selection consists of simply selecting given features from the original features set without changing them, while the second one is about deriving information from these in order to create a new features subset.

2.3.2 Linear techniques

The main linear feature extraction techniques are Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA).
PCA is the most widely used feature extraction algorithm and is an unsupervised approach. It aims to identify the directions of highest variability and project the data into the new coordinate system defined by these orthogonal axes. The new axes (i.e., principal components) constitute a linear combination of the original features and represent the maximum variance directions of the entire dataset.
LDA, unlike PCA, is a supervised algorithm mainly used in clustering and classification problems. Its goal is to search for a linear combination of features that best enables to discriminate data into two or more classes by projecting the data in a way that maximises inter-class variance and minimises intra-class variance.
Both strategies only investigate linear combinations of the original data, so they are both unable to capture non-linear relationships among the features, and assume that the data has a Gaussian distribution, which is an often unverified condition. Differences between the two techniques can instead be visualised with a graphical example, such as the one in Figure 2.8.

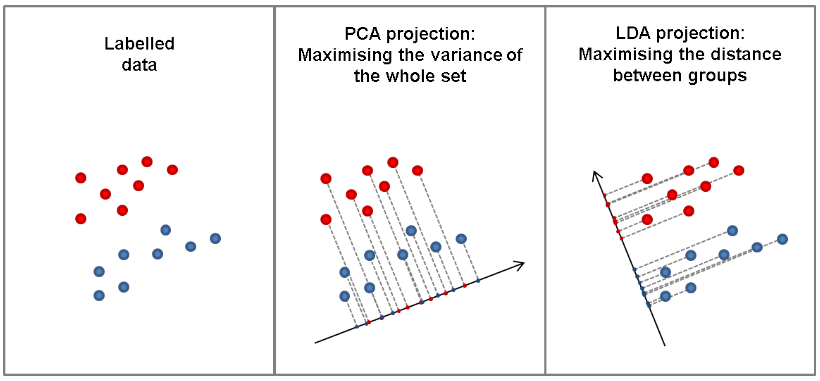

Labelled data
PCA projection: Maximising the variance of the whole set
LDA projection: Maximising the distance between groups

Figure 2.8: Comparison between the two linear feature extraction techniques: on the left the data whose we want to identify a significant dimensionality, in the center PCA reduction, on the right LDA reduction. [44]

2.3.3 Non-linear techniques: the autoencoder

Autoencoders are artificial neural networks based on unsupervised learning, whose main objective is to compress the input data and reconstruct them as output starting from the compressed information. They can be used to perform non-linear feature extraction and are typically employed to reduce data's dimensionality when the relationship among features is described by a non-linear function. The architecture of an autoencoder is often symmetric and it always consists of three parts, as shown in Figure 2.9:

- The **encoder** synthesises the input data information in a latent space of lower dimension z;
- The **bottleneck** (or code layer) represents the vector of compressed features. It forces the network to learn a compressed representation of meaningful attributes of the original data;
- The **decoder** attempts to recreate the original data using the features identified in the central code layer.

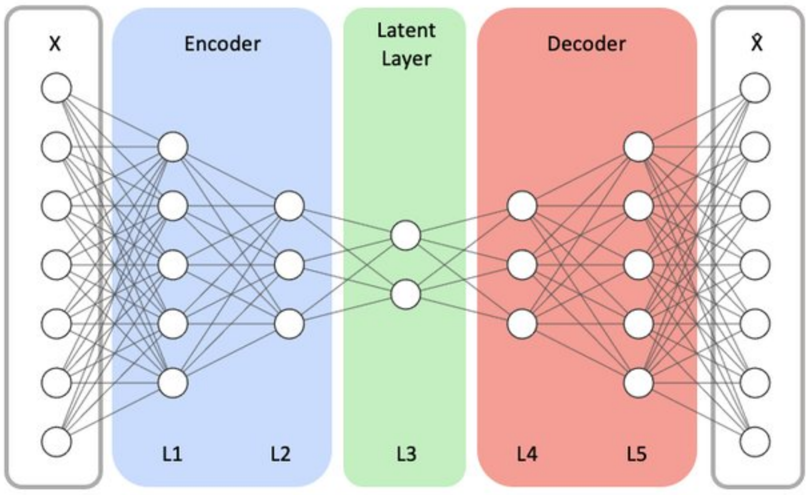


Figure 2.9: Example of an autoencoder architecture, with the three blocks (encoder, latent layer and decoder) in evidence. The vectors x and $\hat{x}$ respectively represent the input and the reconstruction of the same provided as output. [45]

The output of an autoencoder is therefore its prediction of the input, with a certain reconstruction error. In this way, during the training, the code layer learns to extract the most representative and generalisable information from the network's

input vector and synthesise it into a smaller number of neurons. In the context of feature extraction, the vector of interest is then extracted at the bottleneck.
Both the encoder and the decoder are feedforward neural networks, with input and output layers having the same number of neurons, determined by the size of the input data. The bottleneck is instead a single layer of desired size z, smaller than the input size.
A number of autoencoder variants have been developed, such as convolutional autoencoders (CAE) or variational autoencoders (VAE), each optimised for a specific task. In general, these networks are mainly used for anomaly detection, data denoising or image compression [46].

Hyperparameters

For the development of an autoencoder model, it is necessary to define the following hyperparameters:

- Number of layers, which determines the depth of the network and depends on the complexity of the problem:
- Sizes of the intermediate layers, which is the number of nodes in each layer of the encoder and decoder;
- Size of the code layer, that is the number of nodes at the bottleneck, on which the low-dimensional representation depends;
- Activation function for each layer, such as those described in the Section 2.1;
- Loss function, the function to minimise during the training. In the case of an AE it is based on the reconstruction error calculation, i.e. the difference between the input and the obtained output. An example of loss function, often used when inputs are real numbers, is the "Mean Squared Error" (MSE), which evaluates the average of the squared errors between the input x and its reconstruction $\hat{x}$. The MSE formula is:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (\hat{x}_i - x_i) \tag{2.4}$$

- Optimization method, which aims to minimise the loss function. The most commonly used algorithm is ADAM (Adaptive Moment Estimation): this is an advanced extension of the Stochastic Gradient Descent algorithm (SGD) that adapts learning rates individually for each weight of the model [47]. This algorithm allows to achieve a faster and more stable convergence during the model training;

- Regularization techniques, which improve generalisation of the model by penalising its complexity.

The choice of hyperparameters depends on the data and the purpose of the network and can impact the model's performance. The set of optimal hyperparameters can be chosen manually, but it's also possible to perform a tuning thanks to optimisation tools, in order to find the best combination of hyperparameters.

Chapter 3

Dataset

The dataset used in this work was created and provided by the German Primate Centre, one of the partners of the B-Cratos project.
Two macaque monkeys (*Macaca mulatta*) participated in the study: subject Z, a female, and subject M, a male, were trained to grasp a series of objects of equal weight, but different in shape and size. During the task execution the neural activity was recorded using Floating Microelectrode Arrays (FMAs[1]) permanently implanted in the AIP, F5 and M1 cortical areas. Simultaneously the kinematics of arm and hand were acquired thanks to an instrumented glove with electromagnetic tracking on the contralateral limb of the observed hemisphere (Figure 3.2) [49].
Two datasets were used in this research, both corresponding to subject M: these represent two different and successive recording sessions and are referred to as MRec40 and MRec41, respectively.

3.1 Experimental setup

The discussed experiment is described in the article by Schaffelhofer et al. [49]. During the behavioral training and experiments, the monkey sat in a *customised primate chair* with its head fixed. The graspable objects were presented to the animal at a distance of 25 cm at chest level thanks to a rotating table. Overall, in each session 48 different objects including rings, cubes, spheres, cylinders, and bars (Figure 3.1b), were presented and grabbed by the animal grabbed in a pseudo-random order until all objects were successfully grabbed at least 10 times. Additionally, power and precision grips were performed on a graspable handle

[1]Each FMA consisted of 32 monopolar platinum-iridium electrodes (impedance equal to 300-600 kΩ at 1 kHz) with different lengths in the range [1.5 mm, 7.1 mm] to adapt them to the morphology of the sulci [48].

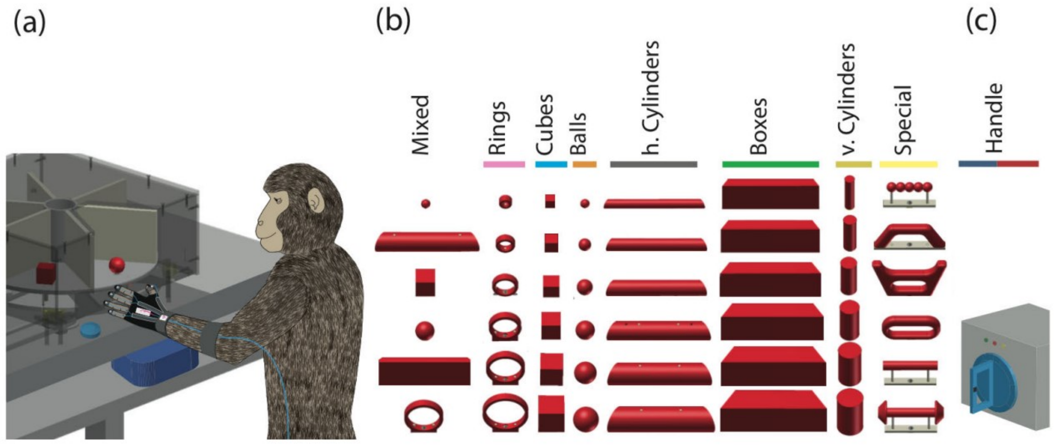


Figure 3.1: Monkey executing the grasping task during the experiment. a) Monkey grasping objects from a turntable; b) Wide range of objects presented to the subject during the experiment; c) Handle for power and precision grips. [50]

equipped with two touch sensors to detect animal's thumb and forefinger contact during precision grips and a pulling force sensor for power grips (Figure 3.1c).

3.1.1 Behavioral paradigm

Monkeys were trained to grasp and hold objects according to a precise behavioral paradigm, which can be described by the following summary steps:

- *Fixation epoch* (500-800 ms)
  While in complete darkness, the animal started a trial by pressing a button near its chest. Then, he had to fixate a red LED light while keeping its hand on the button for a variable time;

- *Cue epoch* (700 ms)
  A spotlight illuminated the graspable object;

- *Planning epoch* (600-1000 ms)
  With the spotlight was off, the animal had to wait until the LED blinked;

- *Movement epoch*
  The blinking of the LED indicated the animal to grab and lift the object;

- *Hold epoch*
  This phase required maintaining the grip for 0.5 seconds;

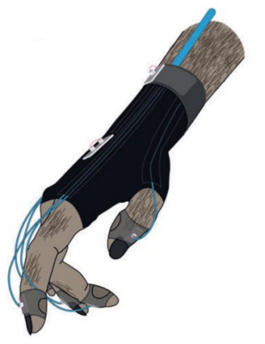


Figure 3.2: Instrumented fingerless glove for small primates equipped with seven electro-magnetic sensors located on each finger, on hand's dorsum and wrist. It allowed monitoring the animal's hand and arm kinematics in 27 DOF, at a sampling rate of 100 Hz. [51]

- *Reward epoch*
  The subject received a reward (a small amount of juice) for each correctly grasped item, while unsuccessfully completed trials were aborted without providing a reward.

Each phase, including the movement epoch, was performed in the dark, except for the cue epoch [48], as shown in Figure 3.3.

After the behavioral training, the animal was implanted with six FMAs, after the identification of anatomical landmarks via 3D anatomical MRI scan of skull and brain. Specifically, two FMAs were inserted in each area AIP, F5 and M1, consisting of 32 monopolar electrodes. Neural activity was recorded from a total of 192 electrodes and sampled at 24 kHz with a resolution of 16 bit.

3.2 Pre-processing

The original signals were filtered with an offline bandpass filter (0.3 - 7kHz) [6] and two *spike sorting* algorithms were applied, one automatic and one manual, both offline and independently on each session. Spike sorting enabled to detect and separate spikes corrisponding to different neurons, identifying the activation patterns of specific neurons and reassigning each spike to a different channel, starting from the 192 physical acquisition channels. For this purpose, the spike sorting algorithm perform an extraction of relevant features and a clustering.
Considering the two datasets under examination, these algorithms recognised the following number of neurons, which we will refer to as *channels*. The number of channels in each dataset is shown in the Table 3.1. However, after the application of

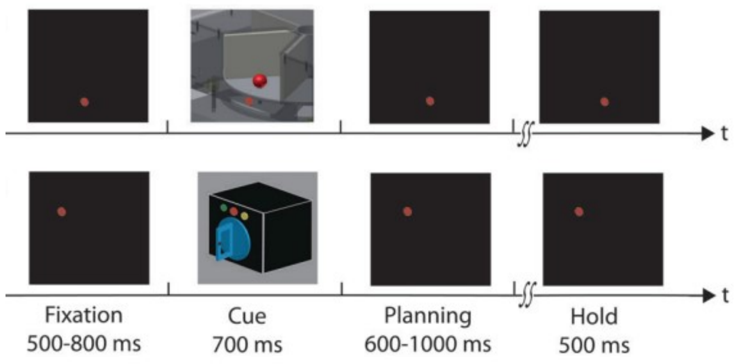


Figure 3.3: Main epochs of the execution of each trial, performed in darkness, except for the *cue epoch*, when the objects were illuminated: at the top, phases relating to the grasp of the objects, at the bottom, those relating to the grasp of the handle. [49]

this algorithm there is no information about the relationships between the channels extracted in the two sessions.
The dataset is composed of trials, each one corresponding to the grasp of a specific object, identified by an integer code, performed with an interval of about 1 second between them. A trial thus consists of a multichannel spike-train (Figure 3.4, top), which represents the activations (spikes) of each neuron over time. The two datasets present a different number of trials and a different number of channels (i.e., neurons), determined by the spike-sorting procedure applied to the sessions independently. The dimensions of the datasets are shown in the Table 3.1.

Dataset	n. trials	n. channels
MRec40	745	552
MRec41	757	568

Table 3.1: Table showing the number of trials and channels for both datasets belonging to subject M.

3.2.1 Pre-processing applied within this work

Datasets were provided by the partner already filtered and pre-processed with the spike-sorting algorithms described above and were further reworked by the authors of previous theses. The techniques applied are described below:

1. *Time-binning*
   The temporal axis was discretised into time bins of 20 ms. Every bin contains, for each channel, the number of spikes that occurred within the corresponding time interval (Figure 3.4, bottom plot).

2. *Sequence creation and labeling*
   In each trial the signal was decomposed by a sliding window of 24 time-bins, which slides over the signal one bin at a time. This windowing allows to present to the classifier a signal that evolves over time in order to simulate a real-time decoding of neural activity.
   The windows are 2D matrices of size *channels* x *window length* and, throughout the duration of the trial, only the windows with the last bin contained in the "Hold" phase were kept. Moreover, each sequence is assigned the *obj_id* code which allows to keep track of the grasped object in the original trial.

Figure 3.5 shows the output of the pre-processing of each trial. Finally, since the kinematic information wasn't made available, it was not considered in this study.

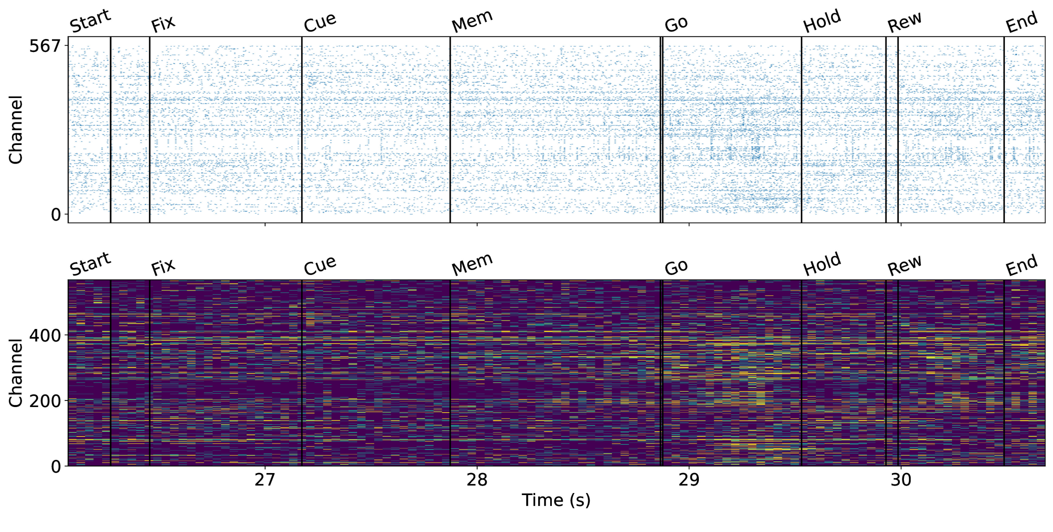


Figure 3.4: Example of a trial signal for subject M, session MRec41. In the top plot, the multi-channel spike-train; in the bottom plot, the intensity map of the same signal after time-binning. The color of each bin represents the number of spikes counted in the interval of discretisation, with intensity increasing from purple to yellow.

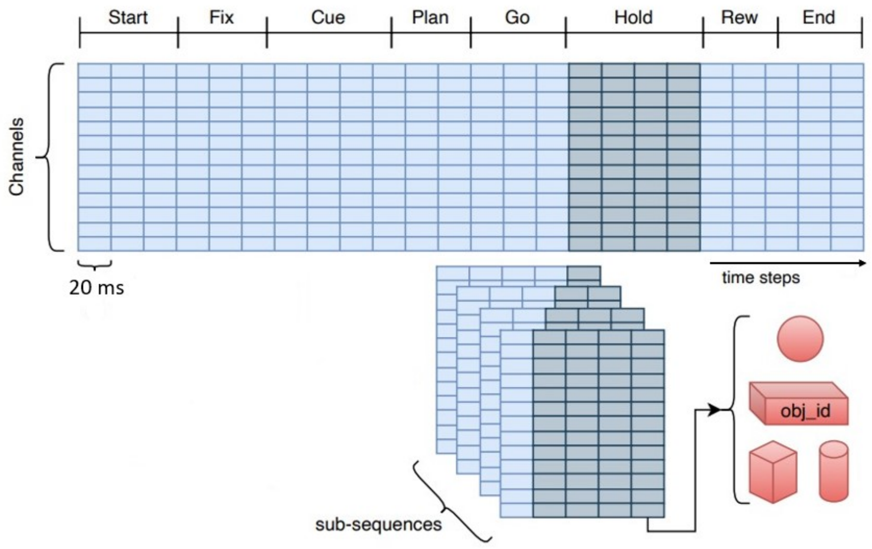


Figure 3.5: Example of dataset after time-binning and sequence creation to simulate on online decoding by decomposing the signal into overlapped windows of 24 bins. Each sequence is associated with the *obj_id* relating to the belonging trial.

Chapter 4

Methodologies

The purpose of the present work is to investigate the possibility to successfully transfer information learned from a model trained on the data of a first session, to a model usable for a subsequently acquired session, exploiting the concept of transfer learning.
The main challenge arises from the different number of features of the two datasets: as mentioned in Section 3.2 and reported in Table 3.1, the two datasets have a different number of channels, respectively equal to 552 for MRec40 and 568 for MRec41, due to the application of spike-sorting algorithms in a dedicated way for each session. This represents an issue when the classifier is trained on the first session dataset and has to be reused on the data from the second session. For this reason, several deep learning models have been developed and tested to perform dimensionality reduction and bring the two datasets to the same number of channels. These models are described in Section 4.2. The structure and hyperparameters of the classifier were obtained from a previous work [22] through a heuristic search in the space of architectures and hyperparameters. In particular, a bidirectional

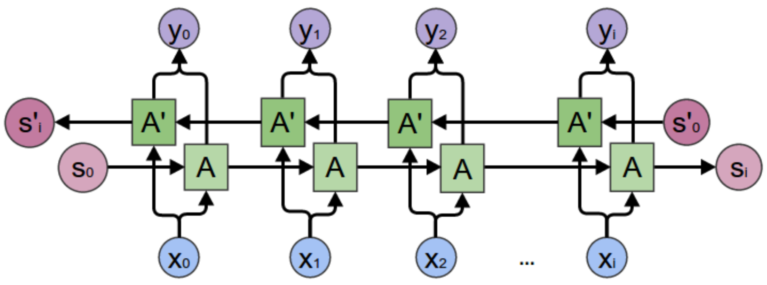


Figure 4.1: Architecture of a BRNN network: it includes a forward line A that transmits information in one direction and a corresponding backward line A' that transmits information in the opposite direction. [52]

LSTM (Figure 4.1) was used, whose parameters can be found in Appendix A.

Previous strategies

In a previous work [23], also conducted in the context of B-Cratos project, a linear dimensionality reduction approach was attempted by separately applying PCA and LDA to extract a fixed number of features and bring the two sessions to have a common number of channels. Since linear approaches yielded promising but not entirely satisfactory results and considering that the correlation between the channels of the two sessions is unknown, it was decided to investigate non-linear approaches. As we can see in Figure 4.2, a non-linear approach can perform a better reduction when data exhibit complex non-linear structures or when in a classification task classes are non-linearly separable.

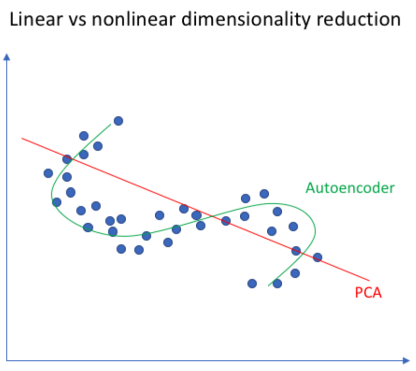


Figure 4.2: Example of a linear dimensionality reduction, performed through PCA, and a non-linear reduction, by an autoencoder. The second one can allow to capture more complex patterns and dependencies, providing a more faithful representation. [53]

4.1 Definition of the Machine Learning problem

In order to explore the feasibility of transfer learning in this context, we identified two specific machine learning problems, referred to as *simple task* and *complex task*. The distinction is based on how the available objects in the datasets, shown in Figure 3.1, are considered. Besides, out of the total represented objects in figure 3.1, only the first six columns were used, while the so-called *special objects* and the handle were excluded from the datasets due to the lack of samples.

- *Simple task*
  The objective is to classify the objects organised into 7 classes, according to their similarity in shape, size and type of grip performed by the monkey, as shown in Figure 4.3. This division into classes was developed in collaboration with the partner responsible for conducting the original experiment. This first and simpler classification was defined in order to evaluate and compare the performance of different neural networks, with the aim of selecting the most suitable model for the purpose.

- *Complex task*
  The final goal involves the classification of the 37 objects considered individually, which is a major challenge even on a single session due to the high similarity among adjacent objects in the dataset. We refer to 37 objects because, despite the 42 represented, the first column includes five objects that are already present in the other columns.

Below are shown the occurrences for each object in the complex task (Figure 4.5) and for the simple task (separated by class, as shown in Figure 4.4). In both cases, the number of trials for each grasped object is the same one in the two recording sessions.

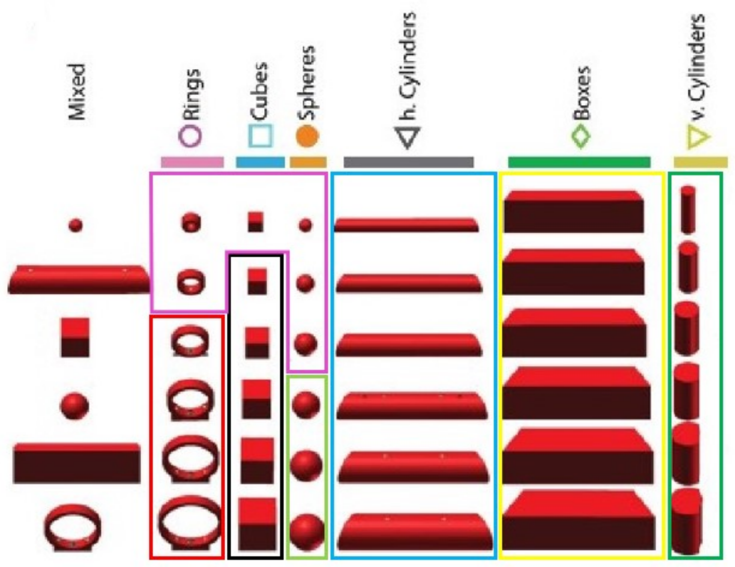


Figure 4.3: Objects from the datasets used during the grasping task. The seven classes identified for the simple task are represented in different colors. The first column, called Mixed, contains five objects already present in other columns, which have been remapped with the identification code of the objects in the specific columns.

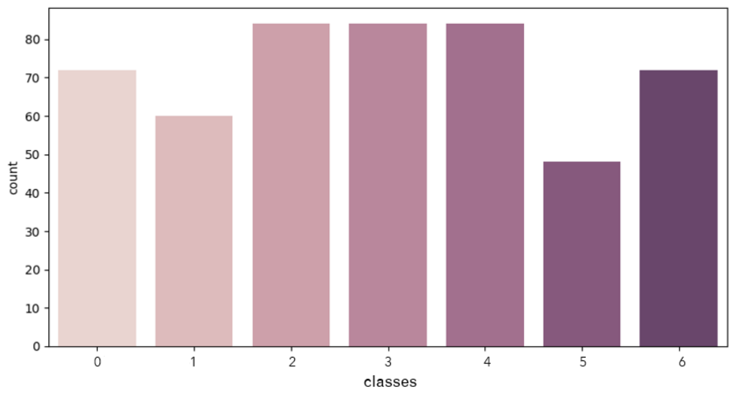


Figure 4.4: Simple task. Distribution of occurrences of the target objects grouped into the seven classes for the two sessions: classes are identified by numbers between 0 and 6.

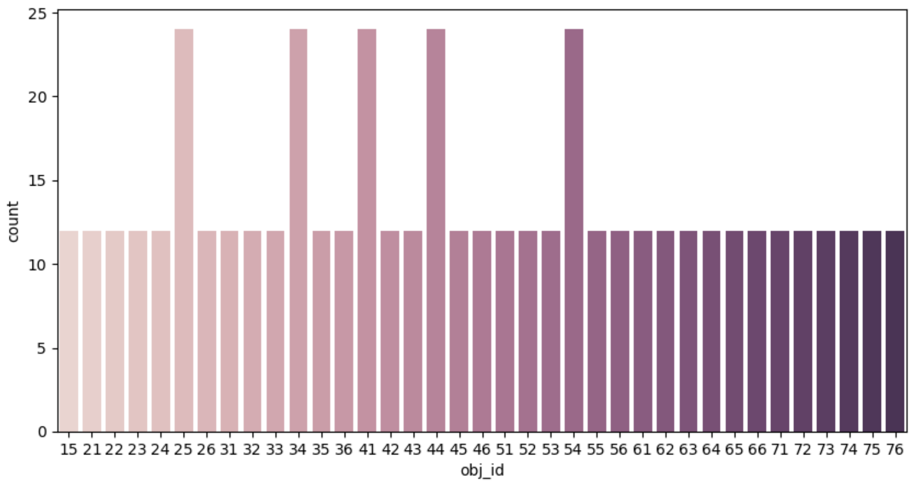


Figure 4.5: Complex task. Occurrence of target objects in the two session datasets: each object is identified by a two-digit *obj_id*. The most numerous objects correspond to the duplicates found in the first column of the dataset and remapped with the specific identification code.

4.2 Developed models

Considering that the different dimensionality of the two datasets represents a problem for the implementation of transfer learning, several models have been developed in order to homogenise the number of channels. All the models were developed starting from the structure of the autoencoder described in Section 2.3.3. These models are presented in the following subsections and summarised in Table 4.5 at the end of this section, while the results obtained from dimensionality reduction and fine-tuning are shown and discussed in Chapter 5. All the models were developed using Keras[1].

4.2.1 Simple autoencoder

The initial idea was to perform a feature extraction on both datasets, using an autoencoder with a latent space dimension z. The two datasets were reduced separately by applying the autoencoders at different times.
In order to establish the most suitable z value for the purpose, three simple autoencoders (Simple AE) were developed, with code layer dimensions shown in Table 4.1.
Except for the number of hidden layers and the dimension of the latent space, the structure of the three models and hyperparameters set are the same (see Listing 4.1 and Table 4.2).

Simple AE	**n. hidden layers**	**Latent space dimension z**
n. 1	2	128 channels
n. 2	1	256 channels
n. 3	1	350 channels

Table 4.1: Table reporting the number of hidden layers and the dimension of the latent space, i.e. the number of neurons in the code layer, for each Simple AE model.

```
input_size = X_train_scaled.shape[1]
interm_size = 256
code_size = 128
output_size = input_size

# Layers definition
input_layer = keras.Input(shape=(input_size,))
drop_layer = layers.Dropout(.2)(input_layer)
encod_layer1 = layers.Dense(interm_size, activation="relu")(
    drop_layer)
code = layers.Dense(code_size, activation="relu")(encod_layer1)
decod_layer1 = layers.Dense(interm_size, activation="relu")(code)
```

[1]Keras is an open-source library for Deep Learning written in Python. It allows to build Deep neural networks (DNNs) with a simple language quickly [54].

```
output_layer = layers.Dense(output_size, activation="sigmoid")(
    decod_layer1)

model_autoencoder = keras.Model(input_layer, output_layer)
model_autoencoder.summary()
```

Listing 4.1: Code example for the creation of a 128 channels *Simple Autoencoder* with Keras.

4.2.2 Convolutional autoencoder

A Convolutional Autoencoder (Convolutional AE or CAE) is a type of autoencoder that presents convolutional layers within its architecture. Specifically, the one-dimensional convolutional autoencoder (1D) is a network that uses one-dimensional convolution to learn main features from a 1D sequence of data, such as a time series. The network still consists of an encoder and a decoder, but characterised by the presence of 1D convolutional layers (Figure 4.6).
A *one-dimensional convolutional layer* is composed of a set of convolutional filters, or kernels, that are applied to the input and slide along a direction of the input sequence's lenght, whether spatial or temporal. Each filter has a fixed size and returns a so-called feature map. Each layer typically applies more filters, providing more feature maps. The convolutional operation is followed by a non-linear transformation through an activation function, usually the Relu function [55].
Each convolutional layer is followed by a pooling layer, whose function is to reduce the size of the feature maps and keep only the most significant information, without adding further weights to the model: the convolutional layer then identifies the features in the input data, while the pooling layer is responsible for selecting the most relevant ones [56]. Since the structure of an autoencoder is symmetric, in the decoder component the opposite operation to pooling is carried out by the deconvolution or upsampling layer, which restores the reduced dimensions to the

	Hyperparameters - Simple AE and CAE
Activation functions	*ReLU* in hidden layers, *sigmoid* in output layer
Loss function	Mean squared error (MSE)
Optimizer	Adam
Learning rate	0.0002
Dropout	Dropout layer
Scaler	MinMaxScaler

Table 4.2: Hyperparameters set for the three Simple AE variants.

original ones.
Only one CAE model was designed, with a convolutional layer after the input and one preceding the output. Dimensionality reduction was set to 350 channels and the number of convolutional filters was set equal to 10. The hyperparameters defined for the development of the CAE model are similar to those used for the Simple AE (Table 4.2). The Keras model summary is shown in Figure 4.7.

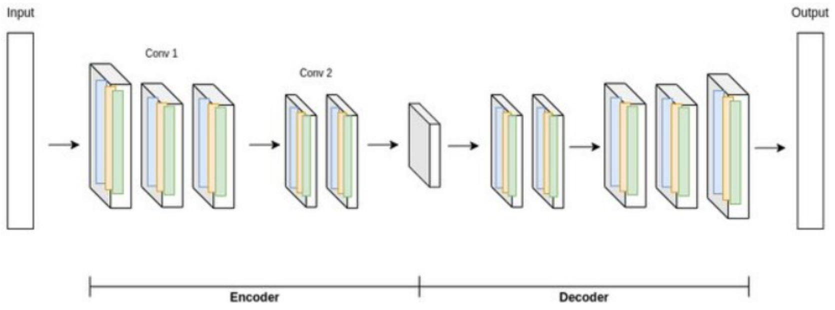
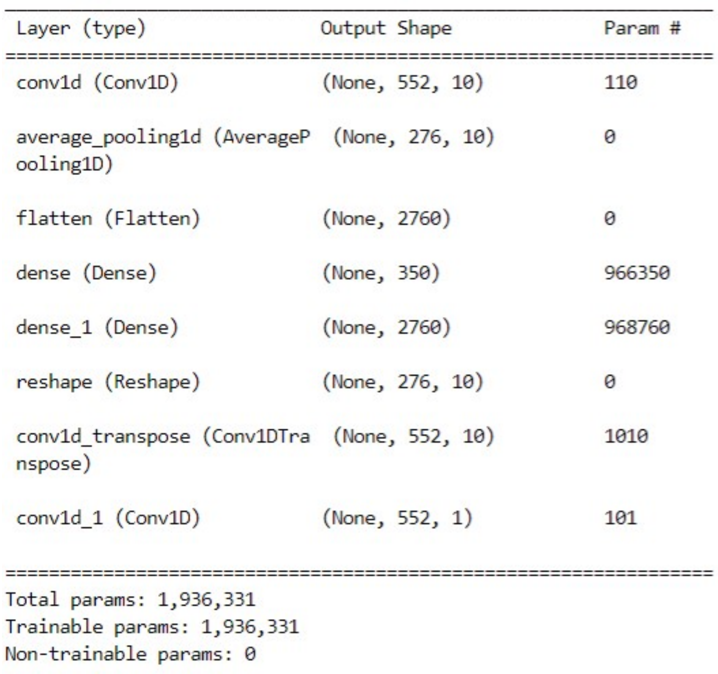


Figure 4.6: Scheme of a simplified 1D CAE architecture. [57]



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 552, 10)           110

 average_pooling1d (AverageP  (None, 276, 10)          0
 ooling1D)

 flatten (Flatten)           (None, 2760)              0

 dense (Dense)               (None, 350)               966350

 dense_1 (Dense)             (None, 2760)              968760

 reshape (Reshape)           (None, 276, 10)           0

 conv1d_transpose (Conv1DTra  (None, 552, 10)          1010
 nspose)

 conv1d_1 (Conv1D)           (None, 552, 1)            101

=================================================================
Total params: 1,936,331
Trainable params: 1,936,331
Non-trainable params: 0
```

Figure 4.7: Summary of the CAE model realised with Keras.

4.2.3 Biased autoencoder

As mentioned in the previous chapter, the training of a simple autoencoder is unsupervised, hence the autoencoder could learn to synthesise in the latent space unuseful information for the classification task under consideration: the network is trained to identify the most significant features for the input reconstruction, but no information is provided about the class or object corresponding to the signal itself. Therefore it was thought to create a *biased autoencoder*. This name refers to a network consisting of the encoder and the code layer typical of an autoencoder, in which the decoder component is replaced directly by the classifier itself. In this way the training of the encoder is driven by the final task and the features that best distinguish the objects are identified in the code layer [58]. Two variants of Biased Autoencoders were realised, both with a code layer size corresponding to 350 channels, but differing only for the update of the weights in the classifier layers during fine-tuning on the dataset MRec41: in the first variant, the LSTM classifier weights are free to be updated, while in the second variant these weights are frozen, that is, they were defined *non-trainable* parameters and therefore not modified during fine-tuning (Listing 4.2).

```
input_size = X_train.shape[2]
code_size = 350

# Encoder
model_encoder = Sequential()
model_encoder.add(Input(shape=(24, input_size)))
model_encoder.add(Dense(code_size, activation="relu"))

# Model definition
model_classifier_load.trainable = True/False
model = Sequential()
model.add(Input(shape=(24, input_size)))
model.add(model_encoder)
model.add(model_classifier_load)
model.summary()
```

Listing 4.2: Code example for the creation of a Biased Autoencoder: setting trainable to "False" freezes the layer's weights of the classifier.

The Table 4.3 describes the hyperparameters chosen for the model and used for both variants. In the output layer, representing the output of the classifier, the activation function *softmax* was used: this function is in fact commonly placed at the output of multi-class classifiers. Finally, validation accuracy was monitored for the definition of an early stopping [2] and for learning rate reduction through the callback `ReduceLROnPlateau` by Keras [60].

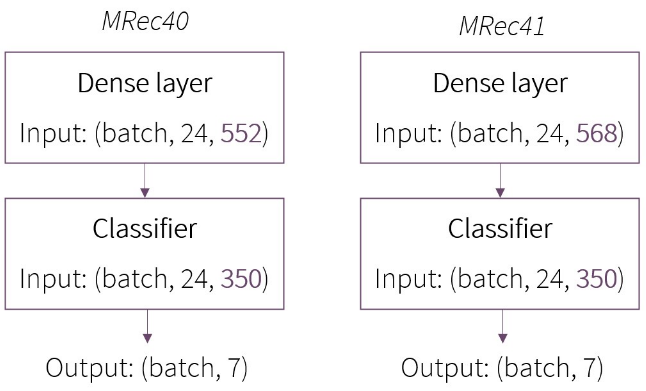


Figure 4.8: Block diagram of the Biased Autoencoder model applied on the two datasets. These networks reduce their dimensionality through the encoder block consisting of a dense layer and prepare them for classification.

	Hyperparameters - Biased AE
Activation functions	*ReLU* in hidden layers, *softmax* in output layer
Loss function	Categorical crossentropy
Optimizer	Adam
Learning rate	0.001 and *ReduceLROnPlateau* Keras callback
Dropout	0.6
Regularization	L2 and Early stopping
Scaler	StandardScaler

Table 4.3: Hyperparameters set for the two variants of Biased AE.

4.2.4 Partial reduction

The so-called *Partial reduction* model re-proposes the structure of the biased autoencoder, but differs from the models described above because it's applied only to the MRec41. The idea behind is to not reduce the dimensionality of both

[2]Early stopping is a strategy applied in order to prevent overfitting in an NN. It involves training interruption when the monitored metric shows no further improvement [59].

datasets to a common size, but to reduce only MRec41, to bring it to the original size of MRec40. In this way, the performance deriving from the initial training on MRec40 (the not reduced dataset) could be superior and allow the fine-tuning to reach greater accuracy, reducing the overall complexity.
In this case, the classifier is initially trained directly on the dataset related to the first session, maintaining 552 channels, then to perform the fine-tuning on the second session's dataset a dense layer was added to the classifier, to bring the 568 channels of MRec41 to be synthesised in 552 channels (Figure 4.9).
Also in this case, as in the previous one, two variants of the same Partial Reduction model were developed, one with the classifier weights free to be updated and one with the classifier layers frozen, while the hyperparameters differ only for the dropout value (Table 4.4). In Table 4.5, the models presented in the section are summarised, together with their dimensions and variants.

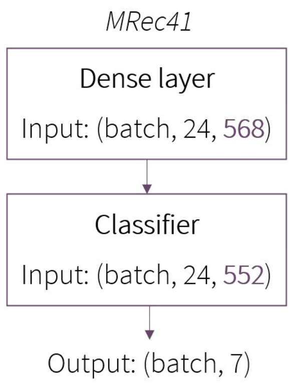


Figure 4.9: Block diagram of the Partial reduction model applied on MRec41. The dense layer reduces its dimensionality to that of MRec40, accepted by the classifier.

	Hyperparameters - Partial reduction
Activation functions	*ReLU* in hidden layers, *softmax* in output layer
Loss function	Categorical crossentropy
Optimizer	Adam
Learning rate	0.001 and *ReduceLROnPlateau* Keras callback
Dropout	0.8
Regularization	L2 and Early stopping
Scaler	StandardScaler

Table 4.4: Hyperparameters set for the two variants of Partial reduction.

Models	Latent space dimension z	Variants
Simple AE	128 channels	
Simple AE	256 channels	
Simple AE	350 channels	
Convolutional AE	350 channels	
Biased AE	350 channels	Free weights
Biased AE	350 channels	Frozen weights
Partial reduction	350 channels	Free weights
Partial reduction	350 channels	Frozen weights

Table 4.5: Table summarising the models described in the section, with their variants.

4.3 Training strategies

After the dimensionality reduction, for each model a pre-training and a fine-tuning of the pre-trained network were conducted, in order to adapt it for classifying data belonging to the new session. The steps involved in the process applied for the first three models are shown in Figure 4.10a, while Figure 4.10b illustrates the steps for the implementation of the Partial reduction model.

Pre-training is performed on 80% of MRec40 dataset, which includes both the training set and the validation set (training set: 64% of the total amount; validation set: 16% of the total amount), while fine-tuning is carried out on the remaining 20% of MRec41 dataset, still intended as sum of training set and validation set (training set: 10% of the total amount; validation set: 10% of the total amount). This dataset split is shown in Figure 4.11. This split was achieved using the `train_test_split` function from Scikit Learn, by imposing a stratification based on each trial's labels (intended as obj_id in both tasks), ensuring a balanced subdivision for training and testing sets. The split was performed directly on the trials before the extraction of the sliding windows, in order to prevent the model from memorising adjacent and partially overlapped sequences of each trial.

4.4 Evaluation metrics for multi-class classification

Evaluation metrics are used in Machine Learning to estimate the performance of an algorithm. Choosing the most appropriate metric is fundamental to assess its effectiveness and to improve its performance. In this specific work, the results

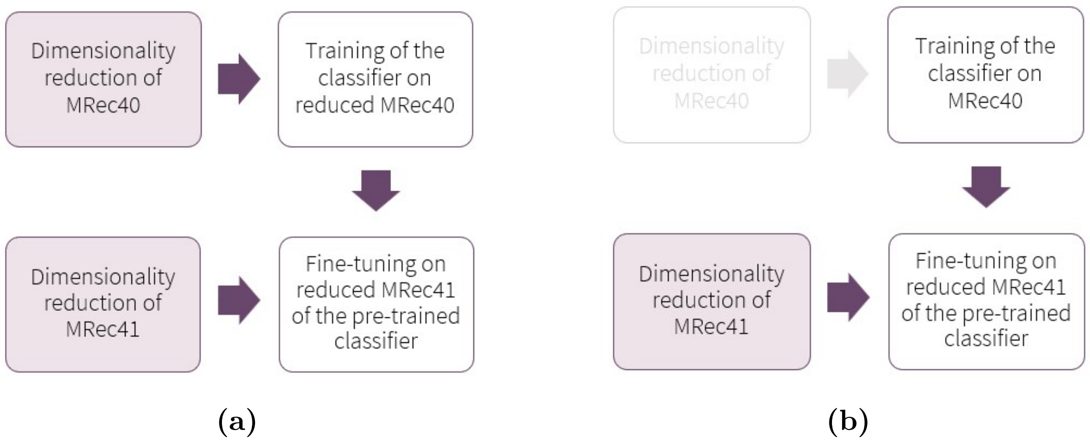

Dimensionality reduction of MRec40
Training of the classifier on reduced MRec40
Dimensionality reduction of MRec41
Fine-tuning on reduced MRec41 of the pre-trained classifier
(a)
Dimensionality reduction of MRec40
Training of the classifier on MRec40
Dimensionality reduction of MRec41
Fine-tuning on reduced MRec41 of the pre-trained classifier
(b)

Figure 4.10: Step summarising the processes: a) Dimensionality reduction for both dataset and training and fine-tuning for the classifier, valid for Simple AE, CAE and for Biased AE. b) Dimensionality reduction for dataset MRec41 and training and fine-tuning for the classifier, valid for the Partial Reduction model.

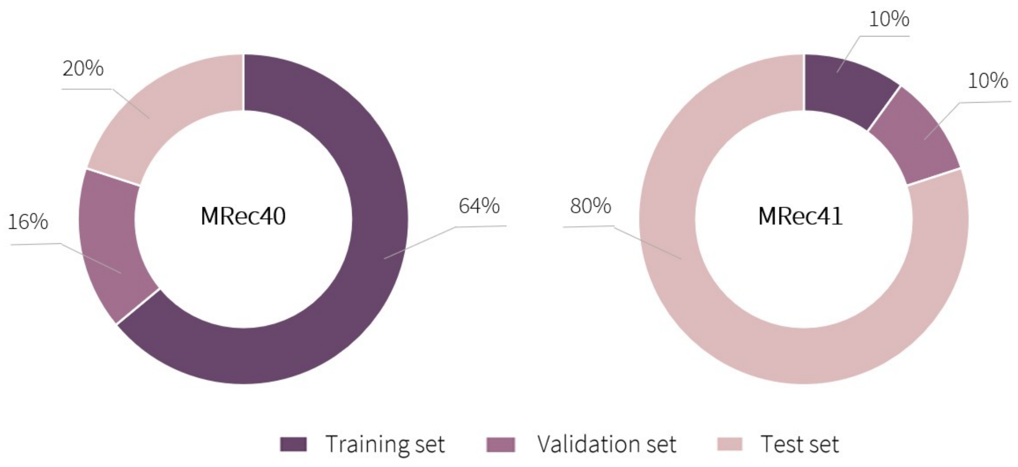


Figure 4.11: Pie charts showing the proportions of training set, validation set, and test set compared to the total dataset: on the left the percentages for pre-training on MRec40, on the right the percentages for fine-tuning on MRec41.

obtained by each model were observed and compared in terms of accuracy and convergence rate.
Accuracy is one of the most common metrics in the field of multi-class classification, especially in the case of balanced classes. It indicates the percentage of correct preditions out of the total elements (samples) classified by the network, therefore

it represents the probability that the model's prediction is correct [61].

$$Accuracy = \frac{Correctly\ predicted\ elements}{Total\ predicted\ elements}$$

A graphical representation of accuracy is provided by the *confusion matrix*. The confusion matrix is an NxN cross table, with N corresponding to the number of classes or outputs in the classification task [62]. This table provides additional information on the amount of correct and incorrect predictions for each class, as its rows display the real values and the columns stand for predicted values: on the main diagonal are shown the correctly classified elements, while outside this diagonal the incorrectly classified elements are shown. The following accuracy numbers always refer to the final value reached by the network on the test set.

Other metrics characteristic of multi-class classification borrow their definition from the metrics for binary classification, where the elements correctly classified as positive are defined True Positives (TP), True Negatives (TN) are the correctly classified negative elements, while False Positives (FP) and False Negatives (FN) represents the incorrectly predicted positive and negative elements, respectively. Starting from these definitions it's possible to convert a multi-class problem into a binary problem for each class [63] and calculate the following metrics:

- Precision indicates the fraction of predicted positives that are actually positive. It has to be maximised if a minimisation of the number of false positives is desired.
$$Precision = \frac{TP}{TP + FP}$$
- Recall indicates the fraction of positive elements correctly classified. It has to be maximised if a minimisation of the number of false negatives is desired.
$$Recall = \frac{TP}{TP + FN}$$
- F1-score indicates the harmonic mean of the precision and recall, representing both of them in a single metric in the range between [0,1]. It allows to minimise incorrect predictions.
$$F1 - score = \frac{2 * precision * recall}{precision\ +\ recall}$$

These metrics are calculated for each class and then averaged through one of the following techniques: *macro average*, simple arithmetic mean of the calculated metrics, suitable in case of balanced datasets, and *weighted average*, that takes into

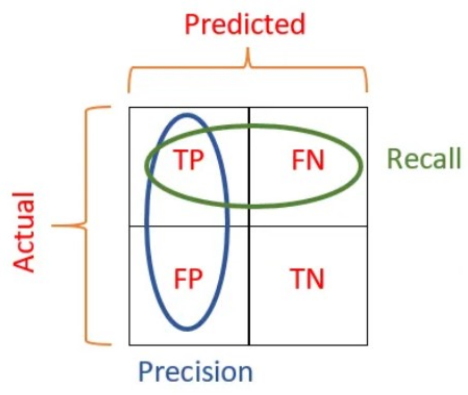


Figure 4.12: Confusion matrix for binary classification, which highlights the elements used to calculate precision (blue) and recall (green) [63].

account the number of examples in each class and assign them a weight. The latter is used when the dataset is unbalanced.

In addition to accuracy, a *convergence analysis* was performed for some models. The convergence rate is here expressed by the number of epochs that the model needs, during the training phase, for reaching a certain threshold of accuracy, calculated on the validation set. The fewer the epochs needed to converge, the faster the model's training speed. This metric is considered of remarkable importance in the current research, as a faster training enables to save valuable time during re-training for operational scenarios: it's possible to benefit from the acceleration of this re-training phase both during the experiments themselves, avoiding long waitings and distractions for the NHP while the DL model updates, and during the final use of a BMI. When calculated, the speed is defined by the number of epochs required to achieve a 90% accuracy on validation set for the simple task and a 30% accuracy on validation set for the complex task. The number of epochs reported in the following tables is calculated as an average over 10 repetitions, considering only the repetitions that reached the defined accuracy threshold: the number of epochs, over 10 repetitions, which didn't reach the threshold was also considered in the final evaluation of the results.

For the Simple AE and Convolutional AE models, we also evaluated the reconstruction errors of the output. In order to perform the dimensionality reduction, each autoencoder was trained in an unsupervised manner to reproduce the vector provided in input as output. The reconstruction error on the test set was calculated using the Mean Absolute Error (MAE) committed by each channel:

$$MAE(y, \hat{y}) = \frac{\sum_{i=1}^{n_{samples}} |y_i - \hat{y}_i|}{n_{samples}},$$

where y indicates the actual value and $\hat{y}$ represents the corresponding predicted value.
For each model, the average MAE committed by all channels and reported in Section 5 is compared to the range of values of the test set before normalisation, which is 8 for MRec40 and 12 for MRec41, in order to provide a percentage indication of the reconstruction error. Once the autoencoder was trained, the reduced dataset was extracted from the code layer and saved, with the original information compressed in the desired size, to be provided as input to the classifier.

```python
# Definition of the encoder
model_encoder = keras.Model(input_layer, code_layer)

# Feature extraction from code layer
X_train_reduced = model_encoder.predict(X_train_scaled)
X_val_reduced = model_encoder.predict(X_val_scaled)
X_test_reduced = model_encoder.predict(X_test_scaled)
```

Listing 4.3: Python code for features extraction from code layer.

Chapter 5

Results

This chapter presents the main results, obtained from the training of the classifier on dataset MRec40 and its fine-tuning on dataset MRec41, for each model described in the previous chapter. The results are divided according to the task, *simple* or *complex*, as defined by the definitions given in 4.1. Beside the fine-tuning approach, also a direct application of the pre-trained model has been tested, with poor results reported in Appendix B.

5.1 Reference

In order to carry out a meaningful comparison and an appropriate evaluation of the performances reached by every developed fine-tuning model, below it's provided what was assumed as a reference and its performances are described. In the present work, the reference is represented by the proposed classifier trained from scratch directly on 20% of dataset MRec41, without pre-training the model on the data from the first session MRec40. The weights are randomly initialised and the model has no prior knowledge of the task. The accuracy values and the rapidity necessary for its reaching are considered as the performances to exceed in order to speak of real success of transfer learning. Below are shown the results obtained by the reference, divided by task and summarised in Table 5.1.

Simple task

The reference achieved a final accuracy of 93%, with the 90% accuracy threshold reached in 21 epochs[1].

[1]The number of epochs is calculated as an average over 10 training repetitions, considering only the repetitions that reached the accuracy threshold defined for the task.

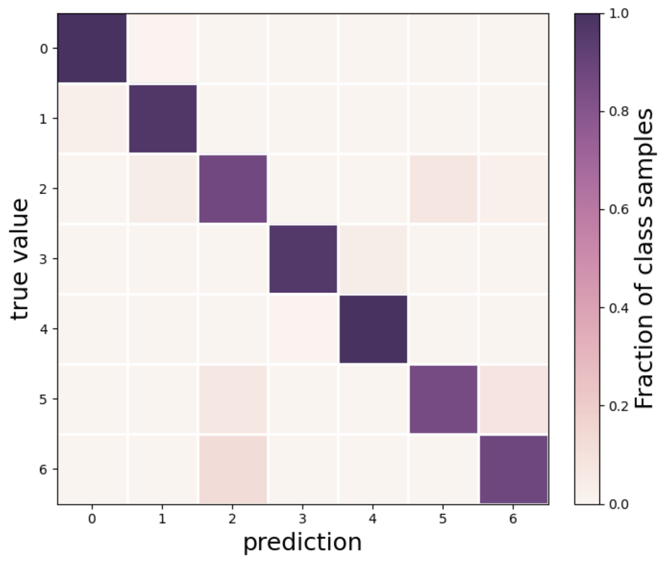


Figure 5.1: Simple task: confusion matrix of the classification results for the test set, using the reference model.

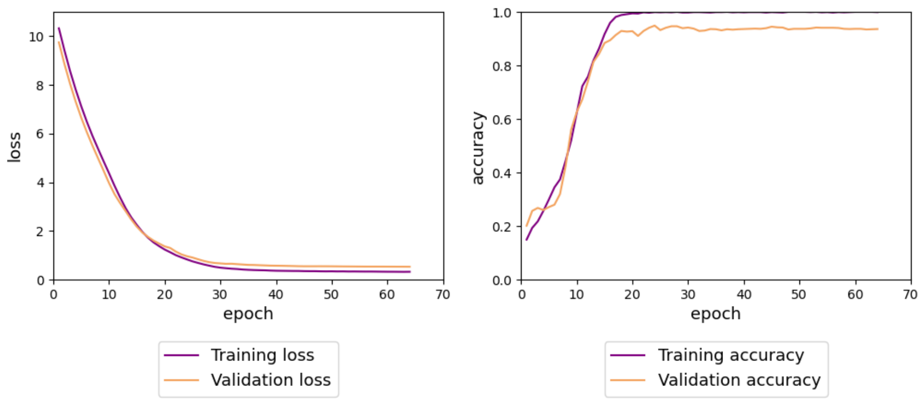


Figure 5.2: Simple task: history of the training from scratch, using the reference model. Trend of the loss function (left) and accuracy (right) in the epochs, both for training set and validation set.

Complex task

The final accuracy achieved by the reference is equal to 37%, with the 30% accuracy threshold reached in 45 epochs[1].

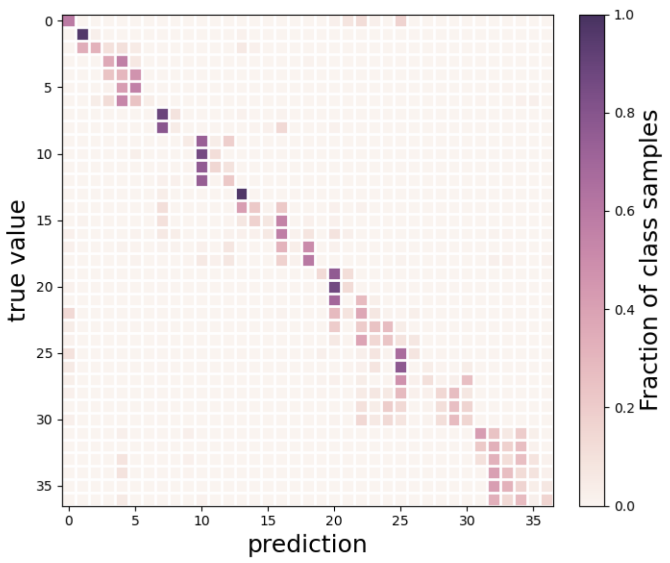


Figure 5.3: Complex task: confusion matrix of the classification results for the test set, using the reference model.

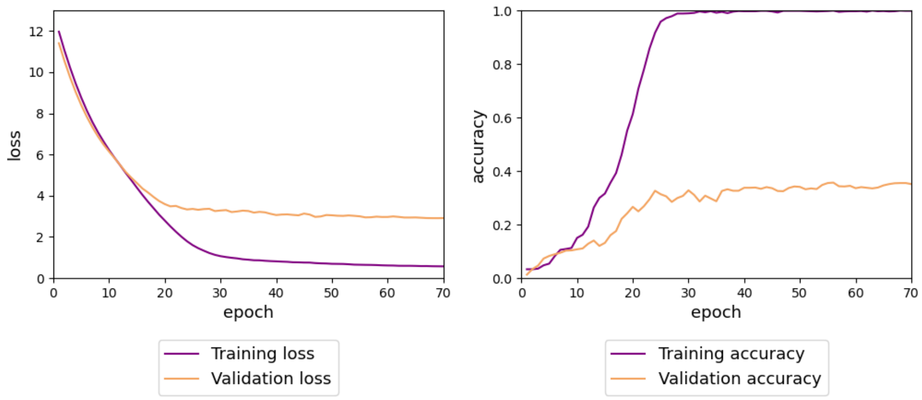


Figure 5.4: Complex task: history of the training from scratch, using the reference model. Trend of the loss function (left) and accuracy (right) in the epochs, both for training set and validation set.

Task	**Final accuracy**	**N. epochs to threshold**
Simple task	93%	21
Complex task	37%	45

Table 5.1: Performance of the reference (i.e., the model trained from scratch on a small percentage of dataset MRec41) for both tasks. The threshold value is an accuracy on the validation set of 90% and 30%, respectively.

As we can see in the confusion matrix related to the complex task (Figure 5.3), most of the incorrectly predicted elements lie very close to the main diagonal. Since in the complex task adjacent objects in the dataset are similar to each other and are also adjacent in the confusion matrix, it's possible to conclude that most of the samples are misclassified by the model mainly for their dimension and not for their shape. In addition, in Figure 5.4 we can notice a significant overfit, which can be attributed to the limited information available during the training, due to the use of a small portion of the dataset MRec41. The observed results for the complex task are consistent with previous works [23][64].

5.2 Simple task: selecting the model

This section discusses the performance of all the models in the simple task of classification, that is the task involving the objects grouped in 7 classes. The results were evaluated in terms of final accuracy achieved on test set, while the convergence rate was quantified for only those models that demonstrated to reach a higher accuracy value than the others and comparable with the reference trained from scratch. The application of this first simple task allows therefore to identify, among all the developed networks, the so-called *best model* to employ in the resolution of the complex task.

Simple autoencoder

We evaluated the training of the three Simple AE models through the MAE analysis. Figure 5.2 shows the histograms of MAE values of the different channels for the 350-channels Simple AE. The distribution of reconstruction errors on the test set presents similar trend and range values in the three models, but the only difference is the mean on 552 channels: the calculation of average percentage MAE across channels shows that in all three analysed models this error is below 39% for the first session and 35% for the second session, decreasing as the size of the latent space z increases (Table 5.2).

Simple AE	**Average % MAE across channels**	
	MRec40	MRec41
n. 1 - 128 channels	0.39%	0.35%
n. 2 - 256 channels	0.35%	0.35%
n. 3 - 350 channels	0.33%	0.33%

Table 5.2: Average percentages MAE across MRec40 and MRec41 channels for the three Simple AE models.

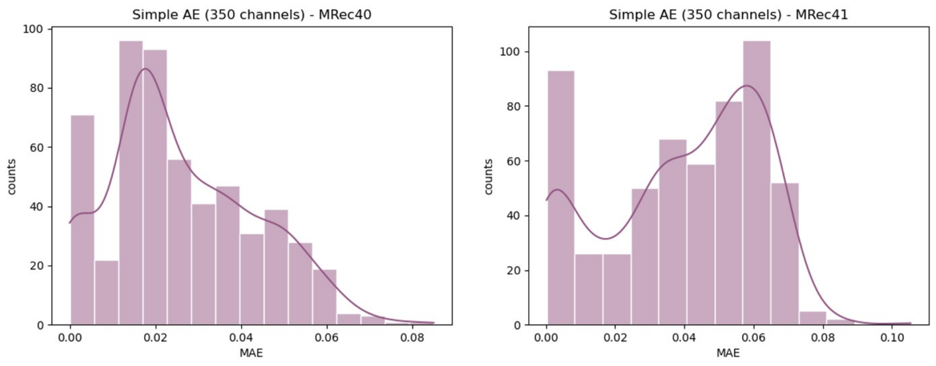


Figure 5.5: Reconstruction errors distribution on MRec40 test set (left) and MRec41 test set (right) for Simple AE n.3.

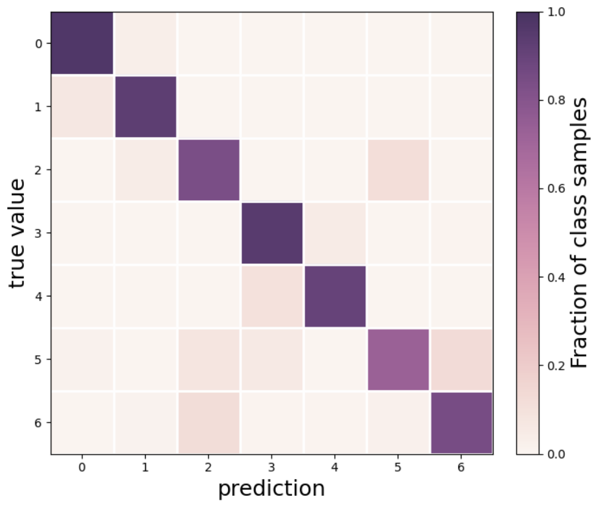


Figure 5.6: Simple task: confusion matrix of the classification results for the test set, after fine-tuning through Simple AE n.3 (350 channels).

Table 5.3 lastly reports the final accuracy values achieved by the classification following the dimensionality reduction of both datasets using the three Simple AE models. As well as in the reconstruction error, also in the performance of the classifier we can see an improvement as the number of extracted channels in the autoencoder-based feature extraction phase increases, although the final accuracy value obtained from the fine-tuning remains below the reference.

Simple AE	**Final accuracy**	
	MRec40	MRec41
n. 1 - 128 channels	87%	72%
n. 2 - 256 channels	94%	88%
n. 3 - 350 channels	96%	89%

Table 5.3: Performance in terms of accuracy of the three Simple AE models in the simple task, both for pre-training on MRec40 and for fine-tuning on MRec41.

Convolutional autoencoder

The training of the CAE was also evaluated using the Mean Absolute Error committed by channels, reported in Figure 5.4 for both datasets. It's evident that the distribution of the reconstruction error across the channels is similar to that of Simple AE models, except for the range of values: Convolutional AE training shows significantly lower average percentage MAE values compared to the previous models (Table 5.4). In addition to the classification performance shown in Table 5.5, Figure 5.8 reports the confusion matrix obtained from fine-tuning. Despite the better reconstruction of the test set achieved by the CAE, the classifier provides a lower accuracy compared to the ones observed with the Simple AE, for the same number of extracted channels (namely for the same final dimensionality).

Convolutional AE	**Average % MAE across channels**	
	MRec40	MRec41
CAE - 128 channels	0.08%	0.23%

Table 5.4: Average percentages MAE across MRec40 and MRec41 channels for CAE model.

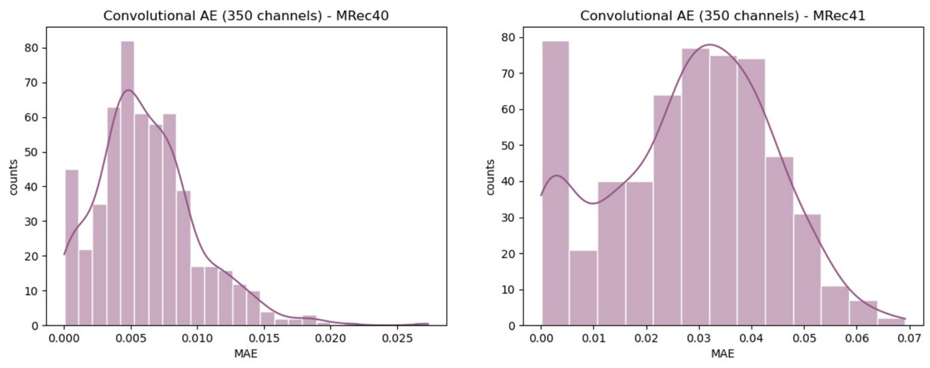


Figure 5.7: Reconstruction errors distribution on MRec40 test set (left) and MRec41 test set (right) for Convolutional AE.

Convolutional AE	**Final accuracy**	
	MRec40	MRec41
CAE - 350 channels	95%	85%

Table 5.5: Performance in terms of accuracy of the CAE model in the simple task, both for pre-training on MRec40 and for fine-tuning on MRec41.

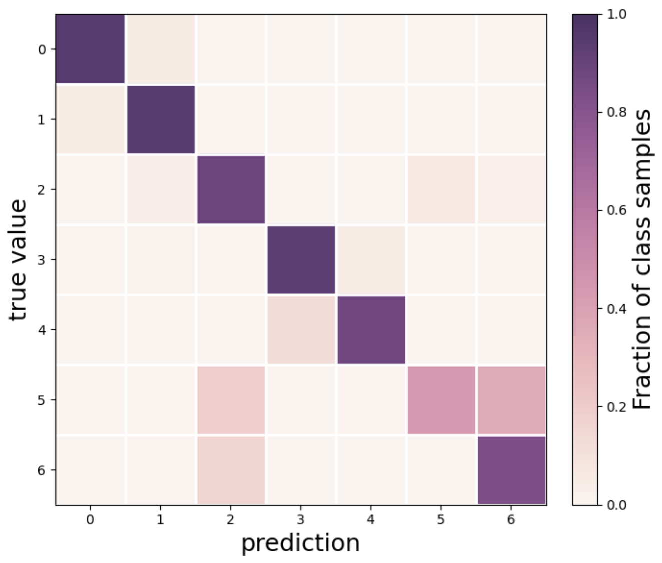


Figure 5.8: Simple task: confusion matrix of the classification results for the test set, after fine-tuning through the 350-channels CAE.

Biased autoencoder

The results obtained using the Biased Autoencoder model show an improvement in the classification of the first session dataset, with a 98% accuracy on the test set. As regards the fine-tuning performance, the first variant of this model (the one with weights free to be updated) reports a slightly lower accuracy compared to the one obtained using the previously discussed autoencoders, while the variant with frozen layers reaches a slightly higher final accuracy than the previous models. In both cases, the developed Biased AE model didn't allow to exceed the performance of the reference classifier.

Biased AE	**Final accuracy**	
	MRec40	MRec41
n.1 - Free layers	98%	87%
n.2 - Frozen layers	98%	91%

Table 5.6: Performance in terms of accuracy of the Biased AE model in the simple task, both for pre-training on MRec40 and for fine-tuning on MRec41.

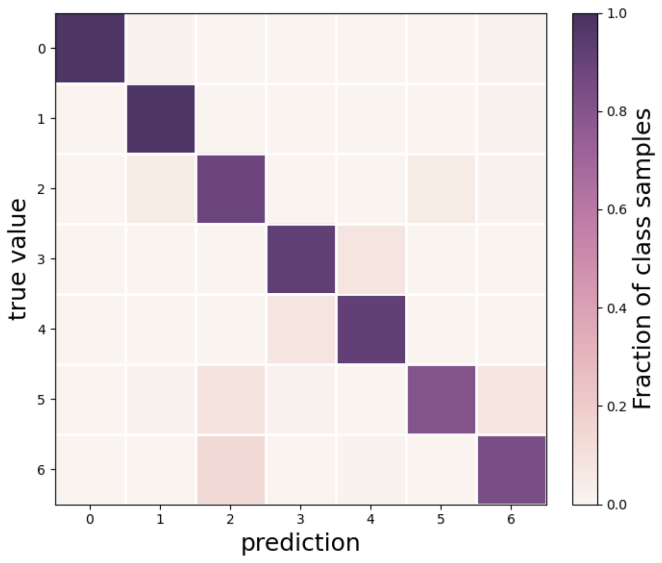


Figure 5.9: Simple task: confusion matrix of the classification results for the test set, after fine-tuning through the Biased AE model n.2 (frozen layers).

Partial reduction

As observed from the values reported in Table 5.7, the Partial reduction model is the one that shows the best results in terms of final accuracy, both on MRec40 dataset and after fine-tuning on MRec41 dataset. For this reason this model was selected as *best model* for the simple task was implemented, with its two variants, also for the complex task. In addition to the confusion matrix (Figure 5.10), Figure 5.11 shows the training history of the model on the second session data and this allows to graphically visualise the trend of loss function and accuracy for training set and validation set over the epochs. By observing these curves it's evident how the network exhibits a high convergence rate: 90% of accuracy is achieved in 8 epochs by the variant with free weights and in 14.5 epochs by the variant with frozen weights[1].

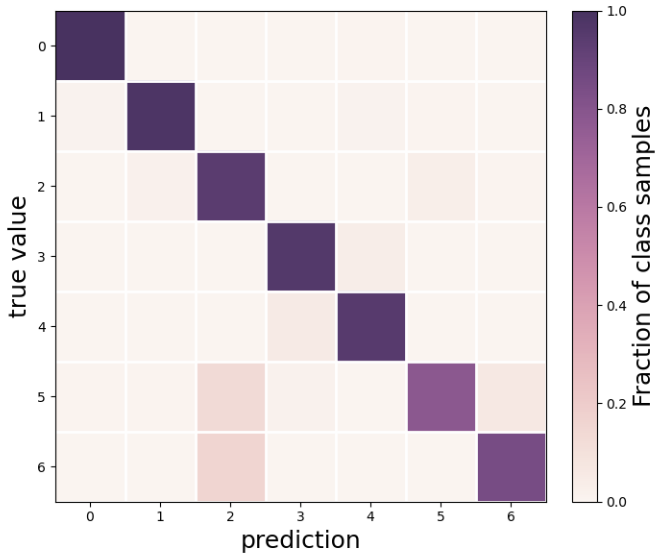


Figure 5.10: Simple task: confusion matrix of the classification results for the test set, after fine-tuning through the Partial reduction model n.1 (free layers).

[1]The number of epochs is calculated as an average over 10 repetitions of training, considering only the repetitions that reached an accuracy of 90%.

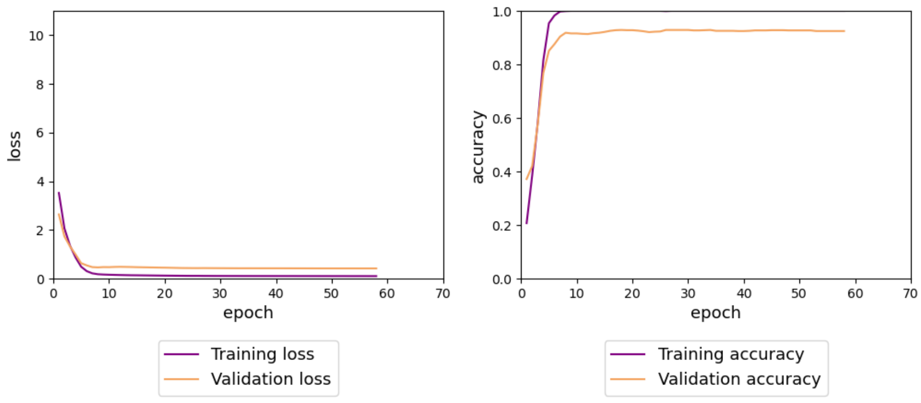


Figure 5.11: Simple task: history of the training for the Partial reduction model n.1 (free layers). It's possible to compare this chart with the one of the reference in Figure 5.2.

Partial reduction model	**Final accuracy** MRec40	MRec41
n.1 - Free layers	99%	93%
n.2 - Frozen layers	99%	92%

Table 5.7: Performance in terms of accuracy of the Partial reduction model in the simple task, both for pre-training on MRec40 and for fine-tuning on MRec41.

5.3 Complex task: final results

This section addresses the complex task, which represents the ultimate goal of the research, and shows the performance of the neural network selected as *best model* of the simple task, namely the Partial reduction model.

Partial reduction

Table 5.8 shows the final accuracy values reached by the two *best model* variants on the second session data (MRec41) and their respective convergence rates[2]. Both variants have proven to outperform the reference classifier, in terms of final accuracy and convergence speed. In particular, especially positive results are observed for

[2]The number of epochs is calculated as an average over 10 repetitions of training, considering only the repetitions that reached an accuracy of 30%.

variant n.1, that is the Partial reduction model with classifier weights free to be updated. It achieves a 30% accuracy in just 9 epochs (compared to the 45 epochs of the reference) and a final accuracy of 45% (compared to 37% of the reference). A further observation allows stating that, while the reference showed to achieve the accuracy threshold only in 8 repetitions over 10, the *best model* variants reached the defined value in all 10 repetitions. These numbers can vary due to the stochastity of training process, anyway this shows that fine-tuning provided consistent results. In the charts in Figure 5.13 the curves are interrupted around the twentieth epoch due to the use of an early stopping imposed on the accuracy of the validation set.

Partial reduction model	**Final accuracy**	**N. epochs to threshold**
n.1 - Free layers	45%	9
n.2 - Frozen layers	39%	12

Table 5.8: Performance of the Partial reduction model in the complex task.

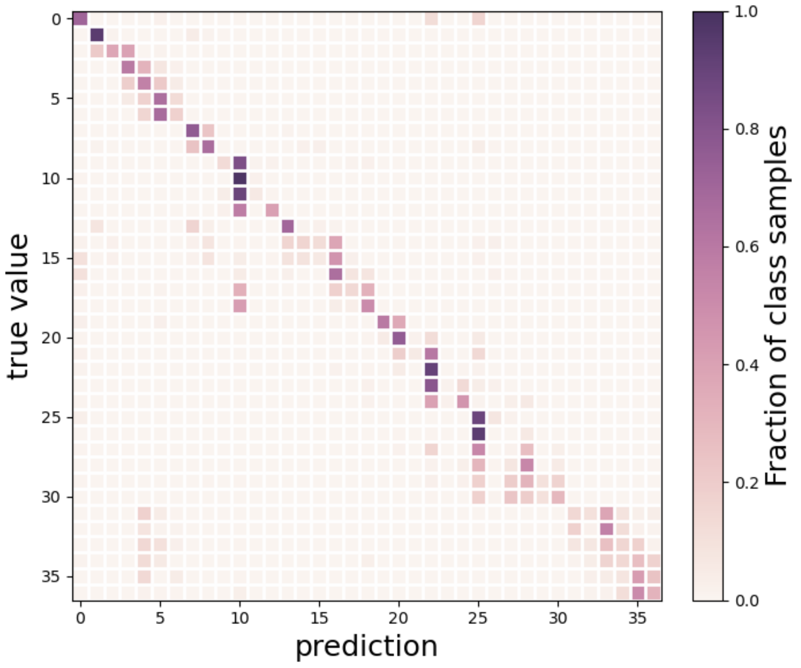


Figure 5.12: Complex task: confusion matrix of the classification results for the test set, after fine-tuning through the Partial reduction model n.1 (free layers).

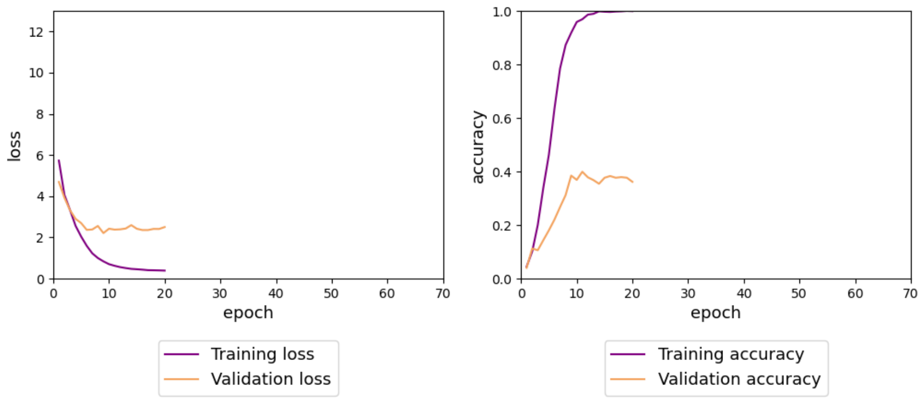


Figure 5.13: Complex task: history of the training for the Partial reduction model n.1 (free layers). It's possible to compare this chart with the one of the reference in Figure 5.4.

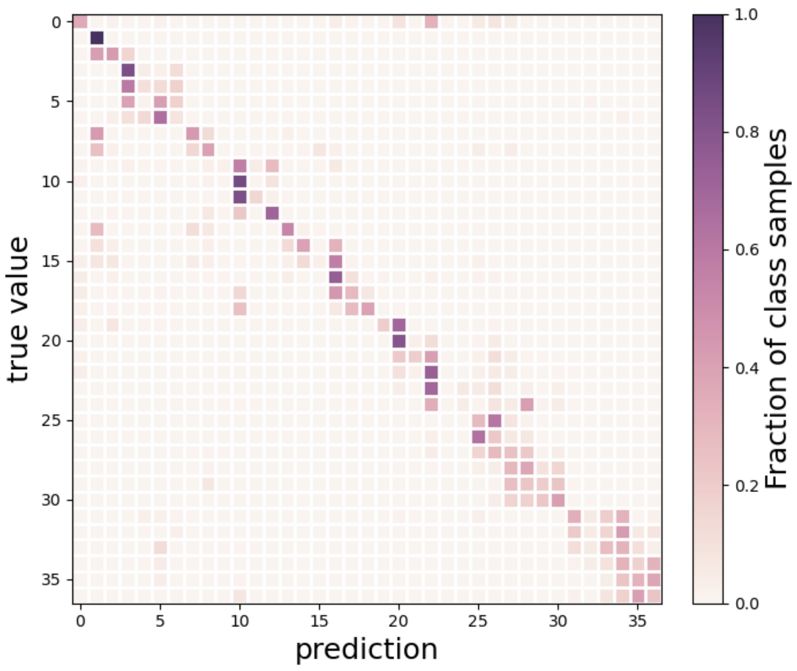


Figure 5.14: Complex task: confusion matrix of the classification results for the test set, after fine-tuning through the Partial reduction model n.2 (frozen layers).

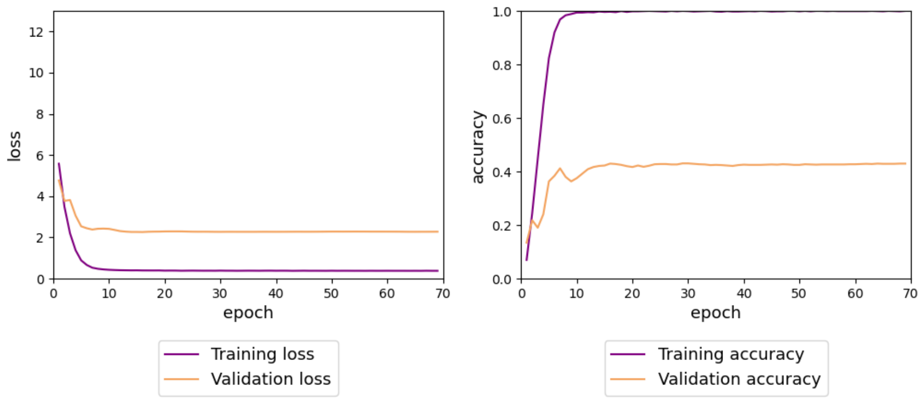


Figure 5.15: Complex task: history of the training for the Partial reduction model n.2 (frozen layers). It's possible to compare this chart with the one of the reference in Figure 5.4.

Finally, the reports of the two *best model* variants are presented, containing the main classification metrics for a more detailed analysis (Tables 5.9 and 5.10).

	Precision	**Recall**	**F1-score**
Accuracy			0.45
Macro avg	0.47	0.41	0.38
Weighted avg	0.48	0.45	0.40

Table 5.9: Partial reduction model's report in the complex task for the variant n.1 (free layers).

	Precision	**Recall**	**F1-score**
Accuracy			0.39
Macro avg	0.43	0.36	0.34
Weighted avg	0.44	0.39	0.36

Table 5.10: Partial reduction model's report in the complex task for the variant n.2 (frozen layers).

From the comparison with the classification report obtained by the reference classifier (Table 5.11), it emerges that the *best model* exceeds its performance also in terms of precision, recall and F1-score.

	Precision	**Recall**	**F1-score**
Accuracy			0.37
Macro avg	0.39	0.33	0.30
Weighted avg	0.41	0.37	0.33

Table 5.11: Reference's report in the complex task.

Chapter 6

Conclusion

The present work was interested in investigating the possibility of reusing information learned by a trained classifier also in subsequent sessions, in relation to the same task. The effectiveness of this operation would represent an important contribution within the field of BMI research. Indeed, transfer learning has proven to be extremely useful for neural decoding tasks related to BMIs, as the one B-Cratos project aims to realise: this strategy would help to deal with the variability of brain signals used in neural decoding models, which constantly evolve because of neuroplasticity, reducing time and resources required to maintain high performance in a long-term usage perspective.

For this purpose, several neural networks were developed and compared in order to perform an appropriate feature extraction that would allow to apply the same classifier to recording sessions of different dimensionality. This preliminary procedure was necessary because of the different number of channels (i.e., active neurons) identified by the spike-sorting algorithms applied separately on the two datasets. Specifically, starting from the structure of a classic autoencoder, the feature extractor's architecture was then modified, exploring *supervised* reduction. The choice of the most suitable network was made by comparing the performance of each of the developed models in the context of the so-called *simple task*: this choice was guided only by the evaluation of the final accuracy, as was previously done in the work of Gesmundo [23]. At first, a worsening relative to feature extraction and classification was observed, as the selected dimensions for the latent space decreases, showing that a higher number of channels can better represent the information useful for decoding. Finally, between all the developed models, the Partial reduction model was identified as the *best model* in the task in question, reporting the best results and exceeding the performance of the reference. This model showed to outperform the classifier trained from scratch also in the application of the *complex task*: specifically, the best result was obtained by setting the weights of the classifier layers as training parameters (variant n.1), achieving a 45% accuracy.

This research also focused on a convergence analysis that showed that the fine-tuning of the pre-trained classifier allows to reach the final performance significantly faster than the training from scratch. This is of considerable importance as, while accuracy values may be limited by the amount of information contained in 20% of MRec41 dataset, a rapid convergence would distinctly reduce the time needed to re-training the network, with advantages both in terms of research and greater usability by the user. Indeed, as previously discussed, users can benefit from BMIs, especially in order to restore lost functionality, but the extreme variability of neural signals results in the need of frequent data acquisition and re-training of the device, leading to a great effort for patients and clinicians. Even in the context of the experiments themselves, a significant reduction of the time needed to recalibrate the decoding model could improve the NHPs behavior management, reducing their anxiety or irritability associated with participating in the experiment. An important observation that could be made is that, despite the positive results obtained from the conducted fine-tuning, missing information about the correspondence between channels of the two datasets prevents from stating that a transfer of knowledge between the neural decoding models occurred: it's in fact possible that the higher performance resulting from the fine-tuning of the pre-trained classifier are due only to the better initialisation of the weights of the model. This more appropriate initialisation allows to reach the results more quickly. This can also justify the poor performance observed after the direct application of the pre-trained classifier on new data, since an explicit mapping between channel is missed.

This work provided promising results regarding the implementation of transfer learning strategies in neural decoding models, however, several limitations emerged by using the mentioned dataset, which contains a reduced number of recording sessions (three sessions in total, of which only the first two used in the present work), without any indication about the time distance between them. Moreover, access to only spike-sorted signals doesn't allow to have datasets with the same number of physical channels and makes it necessary, as seen, a feature extraction carried out without a known correspondence between channels of the different sessions, limiting the transfer of knowledge.

Potential future research may consider using datasets containing this information about time distribution and correspondence between channels. Moreover, while this study focused on a multi-class classification task, B-Cratos project's goal is the implementation of a model for real-time continuous control of robotic prostheses, so future works should explore the application of transfer learning to this type of task. Lastly, once the robustness of the model to intra-subject neural signal variations is verified, it would be interesting to attempt an inter-subject transfer of knowledge. This further step represents a major challenge in the field of BMIs, due to the high customisation required in the adaptation of models to the neural characteristics of different subjects.

Appendix A

Structure and hyperparameters of the classifier

```
# Model definition
model = Sequential()
model.add(Bidirectional(LSTM(40, return_sequences=True, dropout=0.8,
    kernel_regularizer='l2', recurrent_regularizer='l2'), input_shape
    =(X_train.shape[1], X_train.shape[2])))
model.add(Bidirectional(LSTM(units=40, return_sequences=False,
    dropout=0.8, kernel_regularizer='l2', recurrent_regularizer='l2'))
    )
model.add(Dense(y_train.shape[1], activation="softmax"))
model.summary()

# Complilation
opt = tf.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss="categorical_crossentropy", metrics
    =["accuracy"])
callback = keras.callbacks.ReduceLROnPlateau(monitor="val_accuracy",
    factor=0.5, patience=5, min_lr=0.00005)
callback2 = keras.callbacks.EarlyStopping(monitor="val_accuracy",
    patience=40, restore_best_weights=True)

# Training
history = model.fit(X_train, y_train, epochs=70, batch_size=256,
    validation_data=(X_val, y_val), callbacks=[callback,callback2],
    use_multiprocessing=True)
```

Listing A.1: Code example for the creation of the *BRNN*.

Appendix B

Further results

B.1 Direct application of the pre-trained model

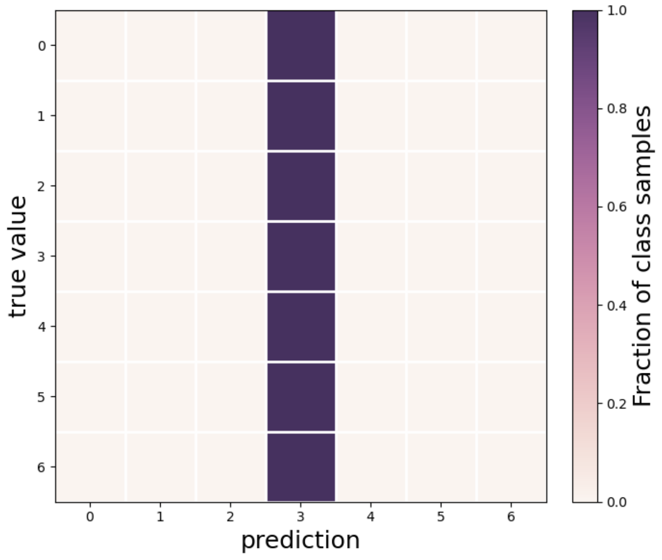


Figure B.1: Simple task: confusion matrix of the direct application of the pre-trained classifier on the MRec41 test set, after its dimensionality reduction through the 128-channels Simple AE.

B.2 Confusion matrices of developed models

Below are the confusion matrices resulting from the fine-tuning on MRec41 for the models described in Chapter 4.

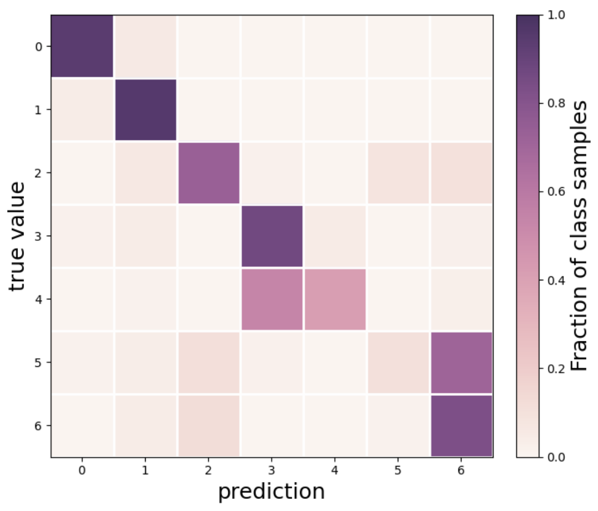


Figure B.2: Simple task: confusion matrix of the classification results for the test set, after fine-tuning through the Simple AE n.1 (128 channels).

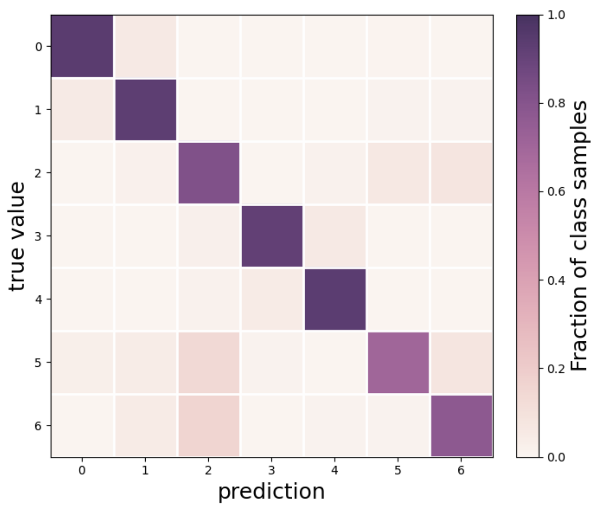


Figure B.3: Simple task: confusion matrix of the classification results for the test set, after fine-tuning through the Biased AE model n.1 (free layers).

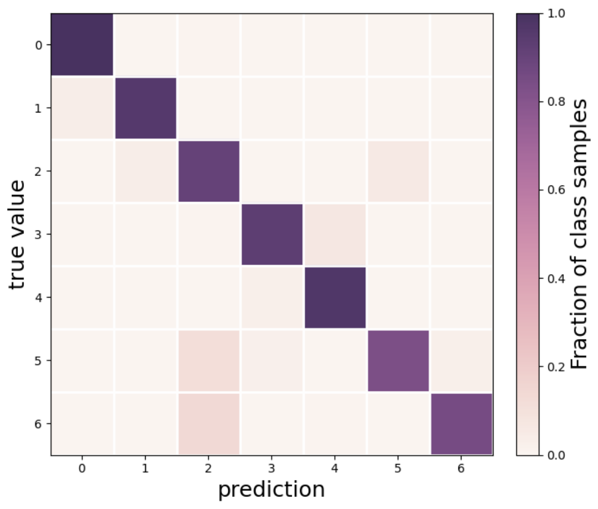


Figure B.4: Simple task: confusion matrix of the classification results for the test set, after fine-tuning through the Partial reduction model n.2 (frozen layers).

Appendix C

HPC based model training

High-performance computing (HPC) refers to the ability to process large amounts of data and perform complex high-speed calculations, leveraging computer clusters and parallel computing. HPC is today widely used in several fields of research and data analysis and allows the implementation of artificial intelligence algorithms and the high-speed training of Deep Learning models. The high-performance computing infrastructure used for this work is managed by Cineca, an inter-university consortium and major Italian computing center, which hosts Marconi100 supercomputer. Marconi100 consists of a cluster of 980 computational nodes equipped with GPUs that accelerate the training of deep neural networks, providing a computational capacity of approximately 32 PFlops.

	Marconi100 technical specifications
Nodes	980
Processors	2x16 cores IBM POWER9 AC922 at 3.1 GHz
Accelerators	4 x NVIDIA Volta V100 GPUs, Nvlink 2.0, 16GB
Cores	32 cores/node
RAM	256 GB/node
Peak Performance	~32 PFlop/s

Table C.1: Technical specifications from [65].

List of Figures

List of Tables

Bibliography

[1] Wonjun Ko, Eunjin Jeon, Seungwoo Jeong, Jaeun Phyo, and Heung-Il Suk. «A survey on deep learning-based short/zero-calibration approaches for EEG-based brain–computer interfaces». In: *Frontiers in Human Neuroscience* 15 (2021), p. 643386 (cit. on p. 1).

[2] https://www.humanitas.it/enciclopedia/anatomia/sistema-nervoso/ (cit. on p. 2).

[3] https://www.my-personaltrainer.it/fisiologia/sistema-nervoso.html (cit. on p. 2).

[4] https://brainintraining.it/corteccia-cerebrale/ (cit. on p. 3).

[5] https://it.wikipedia.org/wiki/Corteccia_motoria (cit. on p. 3).

[6] Veera Katharina Menz, Stefan Schaffelhofer, and Hansjörg Scherberger. «Representation of continuous hand and arm movements in macaque areas M1, F5, and AIP: a comparative decoding study». In: *Journal of neural engineering* 12.5 (2015), p. 056016 (cit. on pp. 4, 28).

[7] Giulia Malfatti. «New evidence of functional interactions within the hand motor system». PhD thesis. University of Trento, 2019 (cit. on p. 4).

[8] https://it.wikipedia.org/wiki/Neurone (cit. on p. 4).

[9] Luigia Canonico. «THE ROLE OF CONNEXINS IN THE PATHOGENESIS OF CHARCOT-MARIE-TOOTH DISEASE». PhD thesis. Università degli studi di Napoli (cit. on p. 5).

[10] https://teachmephysiology.com/nervous-system/synapses/action-potential/ (cit. on p. 6).

[11] Anderson Mora-Cortes, Nikolay V Manyakov, Nikolay Chumerin, and Marc M Van Hulle. «Language model applications to spelling with brain-computer interfaces». In: *Sensors* 14.4 (2014), pp. 5967–5993 (cit. on p. 7).

[12] Valentina Agostini. *Neuroengineering* (cit. on p. 7).

[13] Seonghun Park, Min-Su Kim, Hyerin Nam, and Chang-Hwan Im. «Development of an In-Car Environment Control System Using an SSVEP-based BCI with Visual Stimuli Presented on a Head-Up Display». In: *2022 10th International Winter Conference on Brain-Computer Interface (BCI)*. 2022, pp. 1–2. DOI: `10.1109/BCI53720.2022.9734982` (cit. on p. 7).

[14] Corentin Piozin, Gabriela Herrera Altamira, Catherine Simon, Brice Lavrard, Jean-Yves Audran, Florian Waszak, and Selim Eskiizmirliler. «Motion prediction for the sensorimotor control of hand prostheses with a brain-machine interface using EEG». In: *2022 10th International Winter Conference on Brain-Computer Interface (BCI)*. 2022, pp. 1–8. DOI: `10.1109/BCI53720.2022.9734823` (cit. on p. 7).

[15] https://germanbionic.com/en/solutions/exoskeletons/apogee/ (cit. on p. 7).

[16] Marcello Sicbaldi. «Brain-Computer Interface per riabilitazione motoria e cognitiva». PhD thesis. Alma Mater Studiorum - Università di Bologna, 2019 (cit. on p. 7).

[17] https://www.paradromics.com/blog-post/enabling-connection-ii-bci-for-assistive-communication (cit. on pp. 8, 9).

[18] Ankur Gupta, Nikolaos Vardalakis, and Fabien B Wagner. «Neuroprosthetics: from sensorimotor to cognitive disorders». In: *Communications Biology* 6.1 (2023), p. 14 (cit. on p. 9).

[19] Christian Klaes, Ying Shi, Spencer Kellis, Juri Minxha, Boris Revechkis, and Richard A Andersen. «A cognitive neuroprosthetic that uses cortical stimulation for somatosensory feedback». In: *Journal of neural engineering* 11.5 (2014), p. 056024 (cit. on p. 9).

[20] Greta Preatoni, Giacomo Valle, Francesco M Petrini, and Stanisa Raspopovic. «Lightening the perceived prosthesis weight with neural embodiment promoted by sensory feedback». In: *Current Biology* 31.5 (2021), pp. 1065–1071 (cit. on p. 9).

[21] https://www.b-cratos.eu/ (cit. on pp. 10, 11).

[22] Elios Ghinato. «Robust Classification of a Neuromuscular Signal for Real-Time Control of a Prosthetic Hand». PhD thesis. Torino, Italy: Polytechnic of Turin, Dec. 2022 (cit. on pp. 11, 32).

[23] Ilaria Gesmundo. «Real-Time Classification of Neural Signal from Motor Cortex through Multiple Recording Sessions». PhD thesis. Torino, Italy: Polytechnic of Turin, Mar. 2023 (cit. on pp. 11, 33, 50, 61).

[24] https://www.prensilia.com/mia-hand-ricerca/ (cit. on p. 12).

[25] https://bsj.berkeley.edu/how-artificial-neural-networks-work-from-the-math-up/ (cit. on p. 13).

[26] https://it.wikipedia.org/wiki/Rete_neurale_artificiale (cit. on p. 13).

[27] https://www.intelligenzaartificialeitalia.net/post/spiegazione-della-pi%C3%B9-semplice-rete-neurale-per-principianti-con-codice-implementazione-python (cit. on p. 14).

[28] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. «Learning representations by back-propagating errors». In: *nature* 323.6088 (1986), pp. 533–536 (cit. on p. 15).

[29] Herbert Robbins and Sutton Monro. «A stochastic approximation method». In: *The annals of mathematical statistics* (1951), pp. 400–407 (cit. on p. 15).

[30] https://it.m.wikipedia.org/wiki/Rete_neurale_ricorrente (cit. on p. 15).

[31] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. «On the difficulty of training recurrent neural networks». In: *International conference on machine learning*. Pmlr. 2013, pp. 1310–1318 (cit. on p. 16).

[32] Sepp Hochreiter and Jürgen Schmidhuber. «Long short-term memory». In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cit. on p. 16).

[33] Felix Gers. «Long short-term memory in recurrent neural networks». PhD thesis. Verlag nicht ermittelbar, 2001 (cit. on p. 16).

[34] Xin Wang, Yuanchao Liu, Chengjie Sun, Baoxun Wang, and Xiaolong Wang. «Predicting Polarities of Tweets by Composing Word Embeddings with Long Short-Term Memory». In: Jan. 2015, pp. 1343–1353. DOI: 10.3115/v1/P15-1130 (cit. on p. 17).

[35] Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. «On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?» In: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. FAccT '21. Virtual Event, Canada: Association for Computing Machinery, 2021, pp. 610–623. ISBN: 9781450383097. DOI: 10.1145/3442188.3445922. URL: https://doi.org/10.1145/3442188.3445922 (cit. on p. 16).

[36] https://data-science-blog.com/blog/2022/04/11/how-to-choose-the-best-pre-trained-model-for-your-convolutional-neural-network/ (cit. on p. 18).

[37] Stevo Bozinovski. «Reminder of the first paper on transfer learning in neural networks, 1976». In: *Informatica* 44.3 (2020) (cit. on p. 18).

[38] https://www.bnova.it/intelligenza-artificiale/transfer-learning-cose-i-modelli-e-quando-utilizzarlo/ (cit. on p. 18).

[39] https://www.ai4business.it/intelligenza-artificiale/transfer-learning-cose-come-funziona-e-applicazioni/ (cit. on p. 19).

[40] https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751 (cit. on p. 19).

[41] Lei Chen. «Curse of Dimensionality». In: *Encyclopedia of Database Systems*. Ed. by LING LIU and M. TAMER ÖZSU. Boston, MA: Springer US, 2009, pp. 545–546. ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_133. URL: https://doi.org/10.1007/978-0-387-39940-9_133 (cit. on p. 20).

[42] https://www.andreaminini.com/ai/machine-learning/riduzione-dimensionalita-dati (cit. on p. 21).

[43] Ye Ding, Kui Zhou, and Weihong Bi. «Feature selection based on hybridization of genetic algorithm and competitive swarm optimizer». In: *Soft Computing* 24 (2020), pp. 11663–11672 (cit. on p. 21).

[44] https://nirpyresearch.com/classification-nir-spectra-linear-discriminant-analysis-python/ (cit. on p. 22).

[45] Youngrok Song, Sangwon Hyun, and Yun-Gyung Cheong. «Analysis of autoencoders for network intrusion detection». In: *Sensors* 21.13 (2021), p. 4294 (cit. on p. 23).

[46] https://atcold.github.io/pytorch-Deep-Learning/it/week07/07-3/ (cit. on p. 24).

[47] https://atcold.github.io/pytorch-Deep-Learning/it/week05/05-2/ (cit. on p. 24).

[48] S Schaffelhofer and H Scherberger. «From vision to action: a comparative population study of hand grasping areas AIP, F5, and M1». In: *Bernstein Conference 2014*. 2014 (cit. on pp. 26, 28).

[49] 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. 26, 29).

[50] S Schaffelhofer, M Sartori, H Scherberger, and D Farina. «Musculoskeletal representation of a large repertoire of hand grasping actions in primates». In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 23.2 (2014), pp. 210–220 (cit. on p. 27).

[51] Stefan Schaffelhofer and Hansjörg Scherberger. «A new method of accurate hand-and arm-tracking for small primates». In: *Journal of neural engineering* 9.2 (2012), p. 026025 (cit. on p. 28).

[52] https://www.easy-tensorflow.com/tf-tutorials/recurrent-neural-networks/bidirectional-rnn-for-classification (cit. on p. 32).

[53] https://www.jeremyjordan.me/autoencoders/ (cit. on p. 33).

[54] https://it.wikipedia.org/wiki/Keras (cit. on p. 36).

[55] Xingchen Liu, Qicai Zhou, Jiong Zhao, Hehong Shen, and Xiaolei Xiong. «Fault diagnosis of rotating machinery under noisy environment conditions based on a 1-D convolutional autoencoder and 1-D convolutional neural network». In: *Sensors* 19.4 (2019), p. 972 (cit. on p. 37).

[56] Jannick Kuester, Wolfgang Gross, and Wolfgang Middelmann. «1D-convolutional autoencoder based hyperspectral data compression». In: *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 43 (2021), pp. 15–21 (cit. on p. 37).

[57] Diana Marcela Martinez Ricardo, German Efrain Castañeda Jimenez, Janito Vaqueiro Ferreira, Euripedes Guilherme de Oliveira Nobrega, Eduardo Rodrigues de Lima, and Larissa M de Almeida. «Evaluation of Machine Learning Methods for Monitoring the Health of Guyed Towers». In: *Sensors* 22.1 (2022), p. 213 (cit. on p. 38).

[58] Lemuel Puglisi. «Autoencoder per la riduzione della dimensionalità di dataset molecolari e conseguente predizione di dati clinici». PhD thesis. Università degli studi di Catania (cit. on p. 39).

[59] https://keras.io/api/callbacks/early_stopping/ / (cit. on p. 40).

[60] https://keras.io/api/callbacks/reduce_lr_on_plateau/ (cit. on p. 40).

[61] Margherita Grandini, Enrico Bagli, and Giorgio Visani. «Metrics for multi-class classification: an overview». In: *arXiv preprint arXiv:2008.05756* (2020) (cit. on p. 44).

[62] https://www.analyticsvidhya.com/blog/2021/06/confusion-matrix-for-multi-class-classification/ (cit. on p. 44).

[63] https://towardsdatascience.com/comprehensive-guide-on-multiclass-classification-metrics-af94cfb83fbd (cit. on pp. 44, 45).

[64] Federico Fabiani. «Brain-machine interface for bionic prosthetic arm actuation». PhD thesis. Torino, Italy: Polytechnic of Turin, Oct. 2021 (cit. on p. 50).

[65] https://www.hpc.cineca.it/hardware/marconi100 (cit. on p. 67).