



**UNIVERSITÀ
DEGLI STUDI
DI PADOVA**



**DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE**

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

**CORSO DI LAUREA MAGISTRALE IN
BIOINGEGNERIA**

**“DEVELOPMENT AND COMPARISON OF DATAGLOVE AND SEMG
SIGNAL-BASED ALGORITHMS FOR THE IMPROVEMENT OF A HAND
GESTURES RECOGNITION SYSTEM”**

Relatore: Prof. Giada Giorgi

Laureando/a: Sarah Franceschin

ANNO ACCADEMICO 2021 – 2022

Data di laurea 03/10/2022

Abstract (EN)

Hand gesture recognition is a topic widely discussed in literature, where several techniques are analyzed both in terms of input signal types and algorithms. The main bottleneck of the field is the generalization ability of the classifier, which becomes harder as the number of gestures to classify increases. This project has two purposes: first, it aims to develop a reliable and high-generalizability classifier, evaluating the difference in performances when using Dataglove and sEMG signals; finally, it makes considerations regarding the difficulties and advantages of developing a sEMG signal-based hand gesture recognition system, with the objective of providing indications for its improvement. To design the algorithms, data coming from a public available dataset were considered; the information were referred to 40 healthy subjects (not amputees), and for each of the 17 gestures considered, 6 repetitions were done. Finally, both conventional machine learning and deep learning approaches were used, comparing their efficiency. The results showed better performances for dataglove-based classifier, highlighting the signal informative power, while the sEMG could not provide high generalization. Interestingly, the latter signal gives better performances if it's analyzed with classical machine learning approaches which allowed, performing feature selection, to underline both the most significant features and the most informative channels. This study confirmed the intrinsic difficulties in using the sEMG signal, but it could provide hints for the improvement of sEMG signal-based hand gesture recognition systems, by reduction of computational cost and electrodes position optimization.

Abstract (IT)

Il riconoscimento dei gesti della mano è un argomento ampiamente discusso in letteratura, dove diverse tecniche sono analizzate in termini di segnale di ingresso e di algoritmi impiegati. La principale difficoltà in questo campo è l'abilità di generalizzazione del classificatore, che si complica all'aumentare del numero di gesti da classificare. Questo lavoro ha due obiettivi: in primo luogo si vuole sviluppare un classificatore affidabile a dalle alte capacità di generalizzazione, valutando la differenza delle prestazioni tra l'utilizzo di un segnale sEMG e di un guanto di dati; infine, fa considerazioni riguardanti le difficoltà e i vantaggi nello sviluppo di un sistema di riconoscimento dei gesti della mano basato interamente su segnale sEMG, con l'obiettivo di fornire indicazioni per un suo miglioramento. Per sviluppare gli algoritmi si sono considerati i dati provenienti da un database pubblico; questi sono riferiti a 40 soggetti sani mentre eseguono 6 ripetizioni di 17 movimenti differenti. Infine, sono stati testati sia gli approcci classici di machine learning, sia quelli più recenti di deep learning, confrontandone le performances. I risultati hanno dimostrato migliori prestazioni per i classificatori basati sul guanto di dati, evidenziandone il potere informativo, mentre l'sEMG non è riuscito a raggiungere un'elevata generalizzazione. Quest'ultimo ha fornito migliori prestazioni quando analizzato con approcci classici, che hanno permesso, grazie alla selezione delle features, di sottolineare sia le features più significative, sia i canali più informativi. Questo studio ha quindi confermato le difficoltà intrinseche nell'utilizzo di segnali sEMG, ma può fornire indicazioni per il miglioramento di sistemi di riconoscimento di gesti della mano basati su segnali sEMG, riducendone la complessità computazionale e ottimizzandone la posizione degli elettrodi.

Contents

1	Introduction	1
1.1	Applications	1
1.2	Sensing Modalities	2
1.3	Algorithms	3
1.4	Hand gesture recognition using data gloves	5
1.4.1	Classical Machine Learning	7
1.4.2	Deep Learning	8
1.5	Hand gesture recognition using sEMG	9
1.5.1	Classical Machine Learning	10
1.5.2	Deep Learning	16
1.6	Databases	17
1.7	Objectives	18
2	Methods	19
2.1	NinaPro DB	19
2.1.1	Kinematic Data	20
2.1.2	Electromyography Data	20
2.1.3	Signal Processing	21
2.1.4	Training set, Validation Set, Test Set	22
2.2	Data glove	22
2.2.1	CNN - Convolutional Neural Network	23
2.2.2	LSTM - Long Short Term Memory	25
2.2.3	ResNet - Residual Network	26
2.2.4	Conventional classifiers	29
2.3	sEMG	31
2.3.1	CNN	33

2.3.2	Random Forest	36
2.4	Data glove and sEMG: the fusion approach	38
3	Results	41
3.1	Data glove	42
3.1.1	CNN	43
3.1.2	LSTM	47
3.1.3	ResNet	50
3.1.4	Classical machine learning approaches	53
3.2	sEMG	62
3.2.1	CNN	62
3.2.2	RF	73
3.3	The Bayesian Fusion	86
4	Discussion	89
4.1	Data glove	90
4.2	sEMG	93
4.3	The Bayesian Fusion	97
4.4	Conclusion and Future Improvements	97
5	References	100
A	Deep Learning approaches	103
A.1	Convolutional Neural Networks	105
A.1.1	CNN variants: the Residual Network (ResNet)	107
A.2	Recurrent Neural Networks	108
A.2.1	RNN variants: the LSTM	110
B	Conventional Machine Learning approaches	112
B.1	KNN	112
B.2	Decision Tree	113
B.3	Support Vector Machine	114

Chapter 1

Introduction

Hands are a fundamental tool in every day human-life, because they allow to perform a variety of essential tasks, from simple grasping objects, passing through manual works, since communication. All these tasks involve a series of hand movements, called gestures. Hand gesture recognition and classification was one of the firsts tasks considered in the scientific literature. The evolution of materials, novel sensing techniques, miniaturization of embedded systems and machine learning algorithms have allowed to gain important progresses in this field, improving both the simplicity of application and its accuracy.

1.1 Applications

Hand gesture recognition has a wide-scale application; for sure, the most important could be the following:

- **Communication.** Not all people are able to use the oral communication, due to health conditions (for example muteness) or pathologies that compromise that ability. This is the reason why sign languages were born. However, the portion of the population that is able to use sign languages is limited, leading to difficulties in communicating and to the necessity of the presence of a trained person that acts as an intermediate. Moreover, sign languages are different in relation to the country where they are used. Thus, it is possible to understand the impact that this application had in studying an automated hand gesture classification: each gesture (also dynamic) could be associated to a word or even to a sentence that a translator could display during the communication.
- **Rehabilitation.** This field could be actually divided into two big branches: the unassisted training and the assisted training. Both of them analyze movements (of the hands in this

case) to provide a biofeedback to the user and so to improve its mobility. In particular, for the unassisted training, patients usually have moderate to high function; for them, specific wearable sensors, that analyze the movements, classify it and give feedbacks regarding the correctness of its execution, could be designed. On the other hand, assisted training involves patients with low to moderate mobility and it requires the presence of a robot that assists the movement itself, by the application of corrective forces. The system works by detecting a signal that comes from the patient, processing it and then applying the corrective force, specifically designed for the necessary gesture correction, both in terms of amplitude and direction.

- **Prosthetics Control.** Researchers have demonstrated the opportunity for amputees to regain their independence by the design of a mechanical and controllable prosthetic hand. This is achievable thanks to the amount of signals (human and environmental) that can be analyzed and associated to a specific hand movement (gesture), thus used to govern the prosthetic. Because of its strong social impact, this can be considered as one of the most powerful applications of hand gesture recognition.

1.2 Sensing Modalities

From an hardware point of view, there are several ways to detect the hand movements in virtue of their intrinsic nature. When there is the willingness of moving an hand, the nervous system delivers an electrical impulse to the muscles of interest. The signal is particularly addressed to the Motor Units (MU), which are in this way activated; after that, they respond with an action potential that causes the muscle fibers contraction. Thus, hand movements consist of electrical signals, that can be measured either with superficial electrodes (resulting in the sEMG signal) or with needle electrodes (more invasive and typically used for clinical purposes). Secondly, when moving a hand, joint angles and position of the fingers or wrist in the space are changing too; thus it is possible to sense the different gestures through mechanical sensors that are able to retrieve a signal resulting from the hand/finger/wrist position (or their movement) in the space. Moreover, acoustic/vibratory, optical or visual sensing methods could be used.

Each sensing modality is characterized by different levels of *Sensitivity*, *Wearability* and *Maturity*:

- **Sensitivity:** Sensing modalities like sEMG, NIRS (optical sensing) or FMG (mechanical sensing) measure directly the muscular contraction, capturing respectively the electrical

signal, the changing in blood flow and the muscular movement. Thus, they have the highest sensitivity and resolution. Regarding IMUs sensors, their sensitivity strictly depends on the device set up.

- **Wearability:** In this case, optical sensing methods usually have the smallest size and can easily be integrated into commercial devices; electrical sensing methods though, can be made into wristband types, which is of course a comfortable configuration too. On the contrary, IMUs or strain sensors (which can be made using resistive/conductive material sensible to strain) are usually made with gloves, a more bulky set up. In the last position for what concerns the wearability of the sensing system, we can find solutions based on ultrasound sensors.
- **Maturity:** sEMG, IMUs, strain sensors are widely used in research and there is an high availability of commercial devices. Other modalities like NIRS, are commonly used for other type of task, so the amount of the available literature is limited.

1.3 Algorithms

Typically, hand gesture recognition algorithms can perform either a classification of hand poses and/or trajectories, or regression of continuous parameters. There are two big families in which these algorithms divide: the conventional machine learning techniques and the deep learning techniques. In literature, there are several references for both approaches and many configurations were tried leading to different results with several levels of reliability.

Briefly, machine learning approaches usually involve the necessity of extracting hand-crafted features from the signal used as input, while instead deep learning methods offer an end-to-end approach. In this case, the models automatically analyze the signal and look for intrinsic characteristics, which may not be identified manually by the user, and try to automatically understand the relationship between them and the output of the task (classification in this case). In both cases, the classifier must be reliable and have to generalize well.

The generalization is the ability of a model to provide high performances not just with seen data (consisting of information used for the training phase) but with unseen data too. In the biomedical domain, this is a fundamental and crucial characteristic, because, in general, it is required a model that, once trained, it results reliable for similar future purposes (e.g. new subjects and future periods of time). Unfortunately, generalization is usually hard to address, because biological data typically is characterized by a high level of variability, both inter-subject

and intra-subject. Usually, in literature it happens that the model demonstrates high accuracy on classifying training data, while its ability to recognize hand gestures on new, unseen (test) data is drastically lower. Furthermore, addressing high generalization becomes harder as the number of gestures to recognize increases. Indeed, having more gestures to distinguish means increasing the difficulty of the task, which in statistical terms is referred as decreasing the chance level.

There are several ways to increase the generalization ability of the model, which is directly related to the problem of *overfitting*. In particular, a model is overfitting if it fits too much well the data that were given as input, learning the correlations that just those data have with the outputs, without understanding the correct underlying relationships (i.e. physiological relationship) that is found repeated over all the similar kind of data. In machine learning, there is a strong problem that has to be taken into account whenever a model is trained, which is the *bias - variance tradeoff*. The bias is the difference between the average prediction of the model and the correct value. A model with high bias pays very little attention to the training data and oversimplifies the model. The variance is the variability of model prediction for a given data point or a value which provides information about the spread of the data. A model with high variance pays a lot of attention to training data and does not generalize on unseen data. For this reason, developing a model with high generalization requires having low variance, but if the variance decreases, the bias increases.

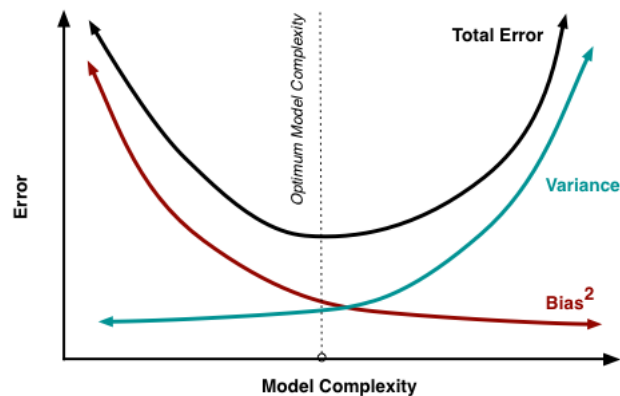


Figure 1.1: *Bias - Variance Trade-Off.* The *Total Error* is given by the sum between the squared bias, the variance and the Irreducible Error. Finding the trade-off, means reducing the Total Error.

Both sufficient generalization ability and limited bias can be achieved when the optimal bias-variance trade-off is reached. There are several approaches that can be used:

- **Considering a large training set** in order to achieve an appreciable heterogeneity. Unfortunately, it is still difficult to have large training set for this kind of task, due to the expensiveness in terms of time; however, there is now a growing number of publicly available datasets with consistent number of subjects considered. Furthermore, the treated dataset

should be divided into a training set and test set, which increases the difficulty of having sufficiently-large training set. Usually, the training set consists of $\frac{4}{5}$ of the overall data, considering either $\frac{4}{5}$ of the movement repetitions of all the subjects or $\frac{4}{5}$ of the subjects themselves.

- **Data augmentation.** This approach is strictly related to the previous point. Data augmentation consists in a set of techniques that aims to generate a larger and more heterogeneous dataset, from the manipulation of the available data. In particular, similar informations are artificially generated.
- **Transfer Learning.** When there are critical poor data conditions, transfer learning approaches can be used: these techniques allow to reuse pre-existing model that were trained for similar tasks and adapting their structures for the new problem taken into account. This could also drastically reduce the computational time required for the training phase.
- **Features Selection.** Proper of the traditional machine learning approaches, features selection techniques are used to reduce the probability of overfitting, the complexity of the model, the computational workload and to increase the model comprehensibility.
- **Dropout,** used in neural networks, to add a random component that reduces the probability of overfitting. It consists of random de-activation of the network neurons, with a specified rate.

1.4 Hand gesture recognition using data gloves

As seen before, there are many approaches to perform hand gesture classification and one of them is measuring hand position with data gloves. There are three types of data gloves that can be distinguished for the type of sensors:

- **Fiber-optic sensors:** they can associate light absorption to a specific strain. They usually provide very accurate measurements because of the strong correlation between the light attenuation and the fiber contortion. The main disadvantage is their necessity of having a light source, which increases the size and weight of the glove, thus its bulkiness.



Figure 1.2: Fiber-optic sensor data glove. [1]

- **Conductive strain sensors:** they are based on the relationship between the strain exerted on them and their electrical resistance (or capacitance) change in correspondence to a joint angle movement. They are formed from conductive nanomaterials embedded on flexible textile polymers. Generally these gloves provide accurate measures and they are lightweight.



Figure 1.3: Resistive sensor data glove. [1]

- **Inertial sensors:** they track the position and orientation over the space of the hand joints, thanks to the use of gyroscopes and accelerometers. They are typically used to record dynamic gestures, but they are the least flexible and tend to make the glove too much bulky.



Figure 1.4: Inertial sensor data glove. [1]

All the previous systems have lower computational cost than others like, as example, vision data systems, which instead typically require high computational resources to be processed accurately before and when using it. Moreover, data gloves allow a continuous recording of the hand gesture and they can be constructed with cheap off-the-shelf components such as bend sensors and a textile glove, which acts as a support structure. Finally, they can reduce significantly the subjectivity of the task, with higher results if gestures are performed with specific protocols. Hand positions, in fact, have specific joint-angles, which can be influenced slightly by the width of the hand and so by the position of the capacitive sensors around it.

The production of datagloves started in 1977 by researchers in MIT (Massachusetts Institute of Technology), and it evolved over time, till now, when the principal datagloves available include the “Cyberglove”, which uses piezoresistive sensors, “5DT Data Glove”, with fiber-optic sensors and “Didjiglove”, which has capacitive sensors.

1.4.1 Classical Machine Learning

Several techniques can be used to perform hand gesture recognition with data glove, exploiting algorithms either from classical machine learning approaches or new deep learning ones. For what concerns the conventional machine learning classifiers, the most common are the K-Nearest Neighbors, the Support Vector Machine (SVM), the Decision Tree, the Artificial Neural Network (ANN) and the Probabilistic Neural Network (PNN). All of them use a specific model to understand the relationship between data and their classes. These techniques were the first studied in the history, requiring the extraction and evaluation of hand crafted features from the signal.

Interesting results were obtained from the work of *Ibarguren et al.* in 2010, which considered a multiclassifier for the classification of 18 gestures of the ASL alphabets using a 5DT Data Glove. In particular, the gestures were classified using a combination of decision tree and k-nn algorithms. First, it was built the decision tree using a clustering method based on Euclidean distance. Then, a 1-nn classifier was used to classify letters at the lowest level nodes, leading to a final 94.61% accuracy over seen data.

In the study conducted by *Heumer et al.* in 2007, a Cyberglove was used to implement grasp recognition using 28 different classical machine learning algorithms and comparing their performances over six classification scenarios. The whole classifiers were grouped into 5 categories:

- **Rule sets:** they use a set of logical rules. The maximum accuracy obtained was of 78.13% and the minimum was 30.88%.
- **Trees:** they consist of pyramid of binary decisions. The maximum and minimum accuracy obtained were 83.44% and 31.06% respectively.
- **Function approximators:** they try to derive a function that relates the input data with output data. The maximum and minimum accuracy obtained were 86.8% and 81.41% respectively.

- **Lazy learners:** they delay the classification of an input data until a request is received, then the class is determined looking at the one of the nearest sample, considering a specific distance metric. They had the highest maximum accuracy of 87.61%.
- **Probabilistic methods:** they produce probability models of each class and then they determine the probability of an input data to belong to them. The maximum accuracy obtained was of 75.31% and the minimum was 61.02%.

1.4.2 Deep Learning

Recently, deep learning approaches were evaluated too. These approaches are interesting because they typically outperform classical machine learning ones, without the necessity neither of signal processing nor extracting hand-crafted features. However, these approaches have the main bottleneck of the lack of understandability, which is the reason why they are usually called black boxes: without specific approaches, it is difficult to understand why a specific result is obtained and in general which physiological information are used to, in this case, classify a certain gesture. The most common deep learning algorithms used are Convolutional Neural Networks (CNN) and Recurrent Neural Network (RNN), also with its variant of Long Short Term Memory (LSTM). An interesting study was the one of *Ayodele et al., 2021*, where they implemented a simple CNN for grasp classification, using a piezoresistive textile data glove knitted from a conductive yarn and an elastomeric yarn. Using windows of 600x5x1 (600 from the sampling of 30 s of signals, 5 from the data glove channels), their CNN consisting of two convolutional blocks (each of them with convolutional, RELU, max pooling and dropout layers) and one fully connected block, outperformed the compared classical machine learning algorithms as k-nn, Gaussian SVM and Decision Tree, with an average accuracy of 88.27%. This study demonstrates the advantages of deep learning techniques compared to the conventional machine learning ones: despite the simplicity of the network, it outperforms the most common conventional classification algorithms in the same scenario.

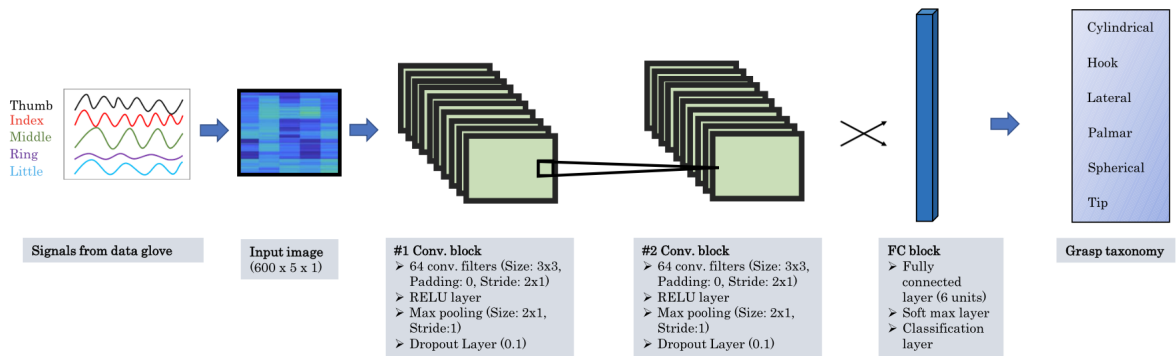


Figure 1.5: Ayodele et al., 2021's CNN architecture. [3]

1.5 Hand gesture recognition using sEMG

The sEMG signal is a non-invasive measurement of the electrical signal coming from muscles when they are activated to perform a specific movement. Briefly, a muscle is composed by a series of Motor Units (MU), which deliver contractile action potentials along the muscle fiber when they are excited by the nervous signal. During the muscle contraction, what happens is that more than one MU could activate, thus the measured electrical signal is formed by the superposition of more than one Motor Unit Action Potentials and this is why it is common to call the electromyography signal "Superposed Motor Unit Action Potential". The sEMG is a non-invasive measure because it is performed with electrodes placed on the skin, on the contrary of the invasive EMG, which uses needle electrodes that penetrate inside the tissue. The number of superficial electrodes influences the type of measure: low-density sEMG when sparse electrodes are used, high-density sEMG when dozens of closely spaced electrodes are used.

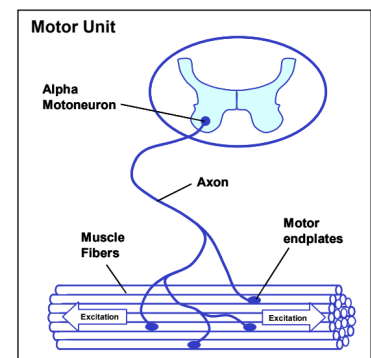


Figure 1.6: Motor Unit. [5]

Since the sEMG is a neuromuscular measurement and contains abundant and fundamental information about muscle contraction, it is widely used also for clinical purposes, trying to understand possible anomalies on the functioning of the muscles. However, one of the main difficulties in analyzing properly the sEMG, is overcoming the extreme variability of this signal, which is not just inter-subject (between different subjects), but also intra-subject (for the same subjects, between different days or within the same day). This property further complicates the task of gesture classification based on the EMG signal, independently from the approach used. A lot of researchers have tried and

still try to implement algorithms that succeed in achieving sufficient accuracy and that can be applied in every-day life, exploring both conventional machine learning and deep learning field. Indeed, in the literature it is possible to find a wide-variety of studies.

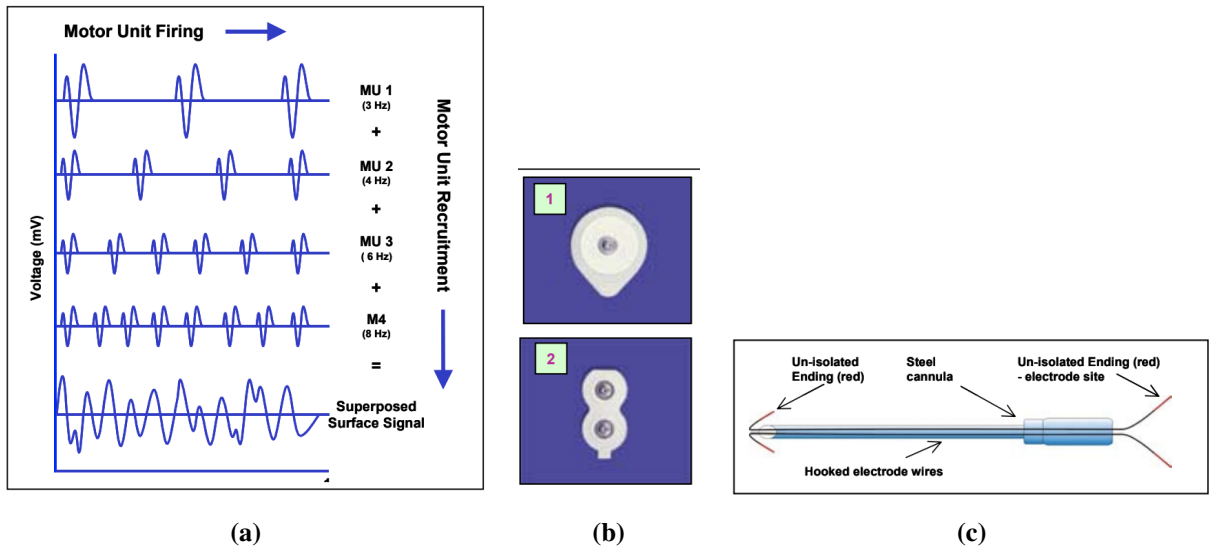


Figure 1.7: (a) Schematic representation of the action potential unit superposition due to the multiple recruitment of motor units. (b) Example of superficial electrodes (NORAXON INC. USA). (c) Schematics of a fine wire electrode: two fine wires with un-isolated endings are located with a steel cannula. System MEDELEC. [5]

1.5.1 Classical Machine Learning

As mentioned before, the classical machine learning approaches share the presence of a significant signal processing phase, followed by the extraction of hand-crafted features; these are then given as input to the mathematical model during the training procedure, such it could adapt its parameters in order to find the correct relationship and dependence between the input data and the output classes, thanks to the minimization of a well-established loss function.

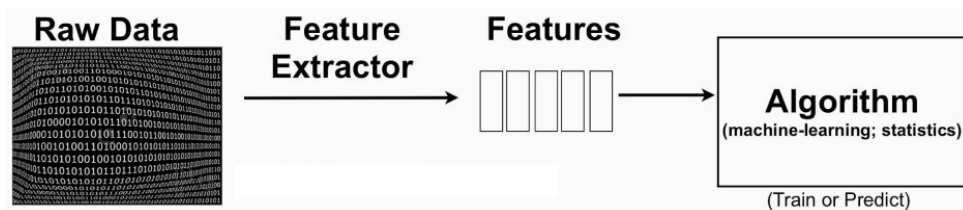


Figure 1.8: Schematics of the conventional machine learning approaches.

Usually, once the raw sEMG signal is acquired, it is filtered to remove noise components, using either a digital high-pass/band-pass filter or applying the wavelet transformation (WT), reconstructing the signal without the noise information. Then, a signal segmentation is performed using time-windows, such that within the windows are present all the encoded information that

will be used as samples to train, validate and test the models. Here, the time-window length is a crucial parameter to be set: larger windows mean more temporal information, but at the same time cause a delay between the event to be detected and the related network output; on the other hand, smaller windows are more adequate for real time applications but consider a limited amount of information. In literature, typical window length values are within the range 30 ms - 300 ms. Another choice that has to be made is choosing between overlapping windows and adjacent sliding windows. At this point, feature extraction techniques are applied, mapping the EMG into a feature set. These techniques extract features in different domains, such as time, frequency, time-frequency, space and fractal.

- **Time domain features.** These are usually quick and easy to compute, such that their main advantage is being low computational demanding than the others. Their disadvantages are mostly related to their assumption of the signal being stationary, which is not true for the sEMG, specially when dynamic gestures are performed. Here are reported some of the most used:

- *Integrated EMG . IEMG.* It is related to the firing point and can be expressed as

$$IEMG = \sum_{i=1}^N |x_i| \quad (1.1)$$

where $|x_i|$ represents the EMG signal in a segment i and N denotes length of the EMG signal.

- *Mean Absolute Value - MAV.* It is similar to the IEMG (Eq. 1.1) and it works better for the prosthetic limb control. It is defined as

$$MAV = \frac{1}{N} \sum_{i=1}^N |x_i|. \quad (1.2)$$

Two manipulations of it could also be used, as:

- * *MAV1*, which assigns discrete weights for improving the robustness of the MAV.

It is expressed as

$$MAV1 = \frac{1}{N} \sum_{i=1}^N w_i |x_i|. \quad (1.3)$$

$$w_i = \begin{cases} 1 & \text{if } 0.25N \leq i \leq 0.75N \\ 0.5 & \text{otherwise} \end{cases}$$

* *MAV2*, which assigns instead continuous weights. It is expressed as

$$MAV2 = \frac{1}{N} \sum_{i=1}^N w_i |x_i|. \quad (1.4)$$

$$w_i = \begin{cases} 1 & \text{if } 0.25N \leq i \leq 0.75N \\ 4 \frac{i}{N} & \text{else if } i < 0.25N \\ 4 \frac{(i-N)}{N} & \text{otherwise} \end{cases}$$

– *Variance of EMG - VAR*. It is a power index and because the mean value of EMG signal is close to zero, the variance is defined as:

$$VAR = \frac{1}{N-1} \sum_{i=1}^N x_i^2 \quad (1.5)$$

– *3rd, 4th and 5th temporal moments - TM3, TM4, TM5*.

$$TM3 = \left| \frac{1}{N} \sum_{i=1}^N x_i^3 \right| \quad (1.6)$$

$$TM4 = \frac{1}{N} \sum_{i=1}^N x_i^4 \quad (1.7)$$

$$TM5 = \left| \frac{1}{N} \sum_{i=1}^N x_i^5 \right| \quad (1.8)$$

– *Root Mean Square*. It is modeled as amplitude modulated Gaussian random process which relates to constant force and non-fatiguing contraction. Its definition is:

$$RMS = \sqrt{\frac{1}{N} \sum_{i=1}^N x_i^2} \quad (1.9)$$

– *Log Detector - LOG*. It provides an estimate of the muscle contraction force. It is defined as:

$$LOG = e^{\frac{1}{N} \sum_{i=1}^N \log x_i} \quad (1.10)$$

– *Average amplitude change - AAC*. It is a measure of the EMG complexity. It is expressed as:

$$AAC = \frac{1}{N} \sum_{i=1}^{N-1} |x_{i+1} - x_i| \quad (1.11)$$

- *Difference absolute standard deviation value - DASDV*. It is similar to RMS feature (eq. 1.9) because it represents a standard deviation value of the wavelength.

$$DASDV = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N-1} (x_{i+1} - x_i)^2} \quad (1.12)$$

- *Zero Crossing - ZC*. It allows to have a frequency information in the time domain. Its definition is:

$$ZC = \sum_{i=1}^{N-1} [\text{sign}(x_i \times x_{i+1}) \cap |x_i - x_{i+1}| \geq \text{threshold}] \quad (1.13)$$

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x \geq \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

- *Myopulse percentage rate- MYOP*. It is an average value of myopulse output which is defined as one when absolute value of the EMG signal exceeds a pre-defined threshold value; it is calculated as:

$$MYOP = \frac{1}{N} \sum_{i=1}^N [f(x_i)] \quad (1.14)$$

$$f(x) = \begin{cases} 1 & \text{if } x \geq \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

- *Willison amplitude - WAMP*. It is another measure of frequency information. Its definition is:

$$WAMP = \sum_{i=1}^{N-1} [f(|x_i - x_{i+1}|)] \quad (1.15)$$

$$f(x) = \begin{cases} 1 & \text{if } x \geq \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

- *Slope sign change - SSC*. It is another method for measuring frequency information. It is defined as:

$$SSC = \sum_{i=2}^{N-1} [f[(x_i - x_{i-1}) \times (x_i - x_{i+1})]] \quad (1.16)$$

$$f(x) = \begin{cases} 1 & \text{if } x \geq \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

The suitable value for the ZC, MYOP, WAMP and SSC features is normally chosen between 50 μV and 100 mV and it is dependent on setting of gain value of the instrument and on level of background noise.

- **Frequency domain features.** Also called spectral domain features, they are mostly used to study fatigue of the muscle and MUs recruitment. They are evaluated thanks to the calculation of the power spectral density (PSD). Here follows the most common:

- *Mean Frequency - MNF.* It represents the average frequency of the signal, calculated as:

$$MNF = \frac{\sum_{j=1}^M f_j P_j}{\sum_{j=1}^M P_j} \quad (1.17)$$

where f_j is a frequency of the spectrum at frequency bin j , P_j its correspondent EMG power spectrum and M the length of the frequency bin.

- *Median Frequency - MDF.* It is a frequency at which the spectrum is divided into two regions with equal amplitude.

$$\sum_{j=1}^{MDF} P_j = \sum_{j=MDF}^M P_j = \frac{1}{2} \sum_{j=1}^M P_j \quad (1.18)$$

where MDF stands for Median Frequency.

- *Peak Frequency - PKF.* It is defined as the frequency at which the maximum power occurs, and it is given by:

$$PKF = \max(P_j), \quad j = 1, \dots, M \quad (1.19)$$

- *Mean Power - MNP.* It is an average power of the EMG spectrum, calculated as:

$$MNP = \frac{\sum_{j=1}^M P_j}{M} \quad (1.20)$$

- *Total Power - TTP.* Also referred to as Zero spectral moment, it is evaluated as:

$$TTP = \sum_{j=1}^M P_j \quad (1.21)$$

- *1st, 2nd, 3rd Spectral Moments - SM1, SM2, SM3:*

$$SM1 = \sum_{j=1}^M P_j f_j \quad (1.22)$$

$$SM2 = \sum_{j=1}^M P_j f_j^2 \quad (1.23)$$

$$SM3 = \sum_{j=1}^M P_j F_j^3 \quad (1.24)$$

- *Frequency Ratio - FR*. It is proposed to distinguish between contraction and relaxation of the muscle, defined as:

$$FR = \frac{\sum_{LLC}^{ULC} P_j}{\sum_{LHC}^{UHC} P_j} \quad (1.25)$$

where *ULC* and *LLC* are the upper and lower-cutoff frequency of the low frequency band and *UHC* and *LHC* are the upper and lower-cutoff frequency of the high frequency band, respectively. It is necessary to set a threshold which divides between low frequencies and high frequencies, and there are two ways to do that:

- (a) Through experiments.
- (b) By using a value of MNF feature.

- *Variance of central frequency - VCF*. It can be defined using *SM0* (is equal to *TTP*), *SM1*, *SM2* features (eq. 1.21, eq- 1.22, 1.23):

$$VCF = \frac{SM2}{SM0} - \left(\frac{SM1}{SM0}\right)^2 \quad (1.26)$$

- **Time-Frequency domain features.** These features are obtained from the application either of the discrete wavelet transform (DWT) or the continuous wavelet transform (CWT). Either directly their coefficients can be used, their average alternatively (Mean Of the Absolute Coefficients), or their standard deviation (Standard Deviation Of the wavelet Coefficients).
- **Space-Domain Features.** They try to evaluate the relationship between the channels or the individuality of them. An example is the Scaled Mean Absolute Value (SMAV), obtained scaling the MAV (eq. 1.2) of each window by the Mean MAV (MMAV) between all the channels.
- **Fractal-Domain Features.** Computed using techniques as the Higuchi fractal decomposition.

After having computed the features, the next step is to choose and train the classifier. In literature, a wide variety of classifiers are explored, but the most used are Support Vector Machine (SVM) and Linear Discriminant Analysis (LDA) [12].

1.5.2 Deep Learning

Deep learning techniques involve the use of deep neural networks, which differ from simply neural networks, because of the higher structure complexity and the higher number of layers. The architecture of a deep neural network is made of three levels of layers:

- **Input layer:** it accepts data either as raw signal, processed signal or hand-crafted features;
- **Hidden layers:** they can be of different type, as fully connected, convolutional or recurrent, in relation to how data are processed;
- **Output layer:** it ends the task, providing the predictions and evaluating their error.

Each layer is characterized by an activation function that finally transforms data just before they enter the next layer. There are several activation functions, different for the modality of use and the type of output they give; moreover, each layer is characterized by a set of weights, which are the parameters that the network has to learn. During the training procedure, weights are randomly initialized and then updated through the back propagation: a specific loss function is evaluated and its loss value (representing the error of the classification) is back propagated to update weights. The back propagation can happen in several ways, according to different optimizer (Stochastic Gradient Descent, Momentum optimizer, ADAM optimizer. . .).

The most common networks used for hand gesture recognition are Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) and whenever there is the willingness to perform hand gesture recognition using deep learning approaches, it must be taken into account the work of Atzori et. Al, 2016. The objective of their study was to develop a robust classifier using a simple deep neural network, with the final purpose of move and control a prosthetic hand. The results obtained showed that their architecture outperformed the classical classification methods. They in particular developed a CNN made of four blocks, each of them having a convolutional layer followed by a non-linear activation function; the input of the network consisted of pre-processed (subsampling and low pass filtered) signal windows of 150 ms, with all the channels considered. They compared its classification performances with RF, SVM, k-NN, and LDA evaluating features as the Histogram, the Waveform Length, the RMS and the Discrete Wavelet Transform. The models were tested among different datasets (totally, 40 gestures and 9 force patterns were considered) and for each of them, the average classification accuracy of the convolutional network was comparable or slightly greater than the average results obtained from classical classification methods.

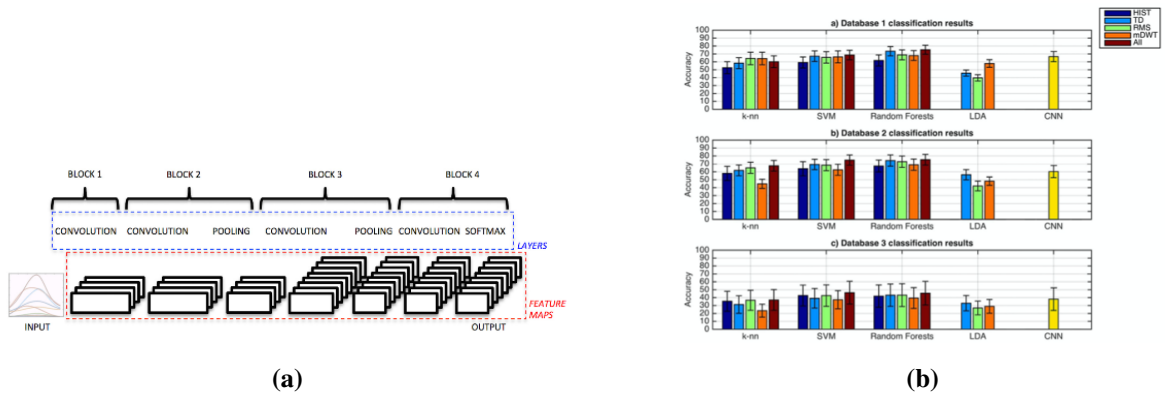


Figure 1.9: (a) CNN Architecture of the work of Atzori et. Al, 2016. (b) Results comparison for each dataset. [7]

This study actually confirmed the power of new deep learning approaches, demonstrating how it is possible to obtain comparable results compared to the conventional approaches using very simple architecture.

1.6 Databases

The variety of studies that is possible to find in the literature is due to the recent launch of several publicly available databases, which contain different kind of data recorded during the execution of gestures from different subjects. The most famous are:

- **NinaPro.** It includes over 300 data acquisitions divided in 10 different databases, with electromyography, kinematic, inertial, eye tracking, visual, clinical and neurocognitive data. The subjects considered are both amputees and healthy.
- **CapgMyo.** It is a benchmark database of HD-sEMG recordings of hand gestures performed by 23 healthy participants. To collect HD-sEMG data, they used a non-invasive wearable device consisted of 8 acquisition modules, each of them having a matrix-type (8x2) electrode array. The gestures performed are a subset of the one of the NinaPro DB.
- **UCI database.** UCI is a machine learning repository where different kind of databases can be found.
- **GRABMyo.** It consists of forearm and wrist sEMG data collected from 43 healthy participants while performing 16 hand and finger gestures.

1.7 Objectives

In this work, data glove and sEMG will be analyzed independently to perform hand gesture classification. The idea is to compare the two different signals in terms of ease of use and presence of significant information directly related to the gesture. The data exploited came from a publicly available dataset and both classical machine learning and deep learning techniques were evaluated. Moreover, it was analyzed the potentiality of considering together the two signals for the classification, to more precisely understand if that could integrate the loss of information that there could be when just one of them is used. Thus, the purposes of this thesis are two. First, it aims at developing a robust classifier to perform hand gesture recognition, comparing the differences in using data glove and sEMG signal. Then, it tries to give clues and hints for the implementation of a hand gesture recognition system based entirely on sEMG signals. The work is structured by explaining the methods used, then showing the results and finally presenting the discussion.

Chapter 2

Methods

The considered data and the implemented classifiers are described in this section. For the development of the code, both MATLAB and Python environment were used. The first was used for the analysis of data:

- Subdivision in training set, validation set, test set;
- Signal windowing;
- Feature Extraction;

For the implementation of the classifiers, Python was used instead. In particular, it was used the machine learning library *TensorFlow*, which has an ecosystem of tools that allow to easily build and deploy ML powered applications.

2.1 NinaPro DB

The data used for this work come from the publicly available dataset *NinaPro*, in particular from the database number 2. The dataset refers to a set of 40 healthy subjects, both female and male, who performed 6 repetitions of gestures coming from three different exercises:

During the acquisition, the subjects were asked to repeat the movements with the right hand. Each movement repetition lasted 5 seconds and was followed by 3 seconds of rest. Electromyography, kinematic and inertial data were acquired, but for this study just the first two kinds were considered.

1. Basic movements of the fingers and of the wrist.
2. Grasping and functional movements.

3. Force patterns.



Figure 2.1: Photos of the hand movements. The first 17 gestures were considered in this study. [2, 14]

2.1.1 Kinematic Data

Hand kinematics was measured using a 22-sensor *CyberGlove II* dataglove (CyberGlove Systems LLC, www.cyberglovesystems.com). The data glove uses proprietary resistive bend-sensing technology to transform hand and finger motions into real-time digital joint-angle data. The device returns 22 8-bit values proportional to these angles for an average resolution of less than one degree depending on the size of the subject's hand.

The data provided were organized in a matrix form (MATLAB format), having the 22 channels of the data glove as columns and the time points of the acquisition as rows. The raw data are declared to be proportional to the angles at the joints in the CyberGlove manual.

2.1.2 Electromyography Data

The muscular activity was recorded using 12 *Trigno Wireless* electrodes (Delsys, Inc, ww.delsys.com). The sEMG signals were sampled at a rate of 2 kHz. Eight electrodes were equally spaced around the forearm at the height of the radio-humeral joint; two electrodes on the main activity spots of the flexor digitorum superficialis and of the extensor digitorum superficialis, and the last two electrodes were also placed on the main activity spots of the biceps brachii and of the triceps brachii. The main activity spots were identified by palpation.

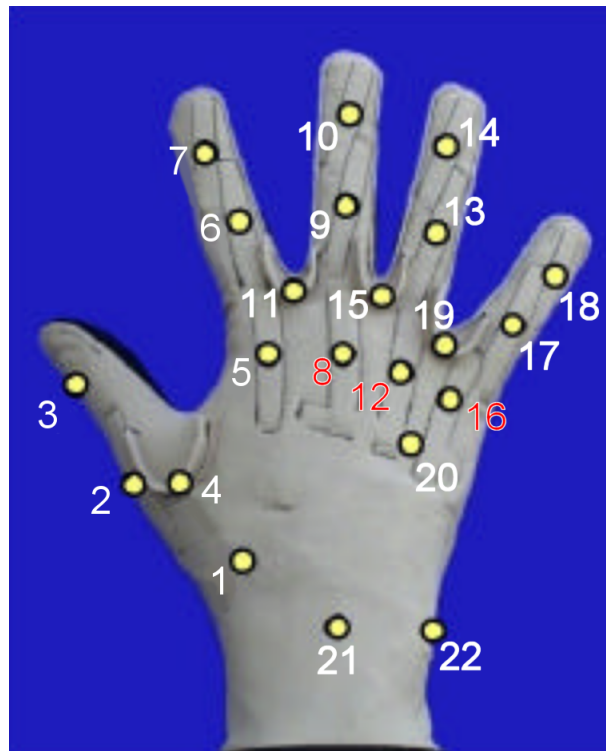


Figure 2.2: Schematics of the CyberGlove II. The marked points corresponds to the channel position over the hand. [2, 14]

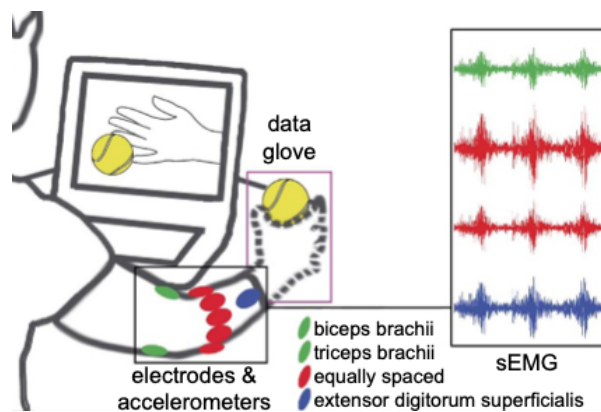


Figure 2.3: Schematics of the Trigno Wireless disposition on the forearm and representation of the overall acquisition setup. [2, 14]

The data provided were organized in a matrix form (MATLAB format), having the 12 electrodes as columns and the time points of the acquisition as rows.

2.1.3 Signal Processing

The publicly available data are not raw but processed through three steps:

1. **Filtering:** Since the Delsys electrodes are not shielded against power line interferences, the sEMG signals were cleaned from 50 Hz (and harmonics) power-line interference using a Hampel filter.
2. **Synchronization:** The data streams were synchronized by up-sampling them to the high-

est sampling frequency (2 kHz) using linear interpolation.

3. **Relabelling:** Since the movements performed by the subjects may not perfectly match with the video stimuli proposed, the erroneous movement labels have been corrected by applying a generalized likelihood ratio algorithm offline.

2.1.4 Training set, Validation Set, Test Set

Accordingly to what is usually done in the literature, the entire dataset was divided into three subsets, representing the training set, the validation set and the test set.

The training set represents the dataset that the model has to use during the first phase of training: for each sample is known its label, such that at the end the model has learned the intrinsic relationships binding the input data and their labels. During this phase, a specific loss function is defined with respect to the type of classifier used, which is directly related to how much classification errors it does while classifying training data; the objective of this training phase is adapting the model parameters to minimize that loss.

The test set is a smaller subset which is used during the final validation of the model that has been trained. It consists of unseen data and it mimics real circumstances where the model has to work well with data that were not used during the training phase. Good performances of the classifier over this subset could mean that the model is generalizing well.

The validation set is the smallest subset which is used during the training phase too, but it acts as a test set useful for the tuning of hyperparameters. By evaluating the performances of the model over the validation set, it is possible to adjust its structure to achieve higher results. After having decided the best model, this should be trained with the union of the validation and training set, in order to let the model learn from the biggest amount of data possible.

Since the dataset consisted of 6 repetitions for each of the 17 gestures for each of the 40 subjects, 3/6 repetitions were considered as training set, 2/6 repetitions as test set and 1/6 repetitions as validation set. This was done accordingly to the gold standard and allowed to obtain the usual and recommended proportion between training set (70%) and test set (30%).

2.2 Data glove

Using the data glove, several classifiers were implemented, exploring both the classical machine learning approaches and the newest techniques from the deep learning field. Input data were organized in windows of 150 ms, according to the suggested lengths from the literature.

Because of the sampling rate at 2 kHz, each window had a dimension of 300x22. Raw data, coming from the CyberGlove II, was directly employed without any processing in order to test the potential ability of deep learning approaches to extract autonomously useful information for the classification. The neural networks that were tested are CNN, LSTM and ResNet. Due to their high computational requirements, their parameters were empirically determined, so that each network could have the best possible performances.

2.2.1 CNN - Convolutional Neural Network

The architecture tested has three convolutional blocks as illustrated in Fig. 2.4, each of them with an increasing number of convolutional filters (32, 64, 128) and with dimension of 3x3, 3x3 and 2x2 respectively.

After the application of the convolutional filter, a non-linear activation function is applied (ReLU). Then, to decrease the probability of overfitting, Batch Normalization is done, followed by a Max Pooling layer (2x1) and a Dropout (rate of 0.3). The final block is a Fully Connected Block, which first of all flattens data into an array of dimension 17x1, then it applies the Softmax Function to perform the classification task.

The training phase consisted of 20 epochs, with a decreasing learning rate approach (through the use of the validation set). The loss function was the categorical-cross entropy, typically involved in non-binary classification problems. The ADAM optimizer was the one considered.

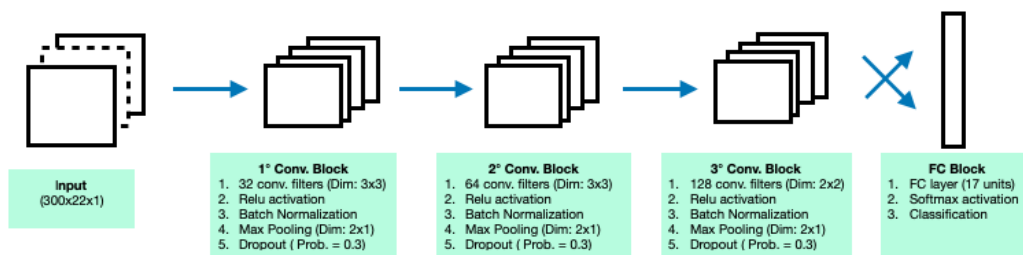


Figure 2.4: Data Glove CNN architecture

The CNN is one of the most common network in this field. Even if it was born for addressing imaging problems, it can be adapted to hand gesture recognition tasks by giving as input "images" of signal. These images correspond, in this case, to a 2D representation of the signal, having the information coming from all the channels in the considered observation window. Thanks to the application of convolutional filters, data of different channels and time points are simultaneously analyzed and combined, trying to extract features coming from their relationship. In terms of Python code, the network was created with the following lines.

```

1 model = Sequential()
2 model.add(Conv2D(32, (3, 3), input_shape=(300, 22, 1)))
3 model.add(Activation('relu'))
4 model.add(BatchNormalization(axis=1))
5 model.add(MaxPooling2D((2, 1)))
6 model.add(Dropout(0.3))
7 model.add(Conv2D(64, (3, 3)))
8 model.add(Activation('relu'))
9 model.add(BatchNormalization(axis=1))
10 model.add(MaxPooling2D((2, 1)))
11 model.add(Dropout(0.4))
12 model.add(Conv2D(128, (2, 2)))
13 model.add(Activation('relu'))
14 model.add(BatchNormalization(axis=1))
15 model.add(MaxPooling2D((2, 1)))
16 model.add(Flatten())
17 model.add(Dense(17, activation='softmax'))

```

Listing 2.1: CNN glove code

Sequential, *Conv2D*, *Activation*, *Batch Normalization*, *MaxPooling*, *Dropout* and *Flatten* are functions coming from the *Keras* framework of the *TensorFlow* library and they automatically allow to create sequentially specific layers for the networks. In details:

- *Conv2D* implements a convolutional layer. The first input parameter corresponds to the number of convolutional filters; then, their dimensions have to be specified. For the first layer of the network, it is always required to give the shape of the input data. It was necessary to re-adapt the signal window (300x22) in 3D matrix with dimensions 300x22x1.
- *Activation* applies the activation function. In this case, ReLU was chosen.
- *BatchNormalization* implements the batch normalization. The dimension along which it operates should be stated using the keyword *axis*. In this case, 1 was that dimension, meaning that each feature (channels could be thought as features) is normalized having a zero mean and a standard deviation equal to 1.
- *MaxPooling2D* reduces the dimensionality of the input matrix by sliding a mask over it. This operation consists in taking the maximum value of the input within the superposition with the mask. Thanks to the ability of autonomously extracting features from the inputs,

the data outputting each network layer can be called feature maps. Usually, the pixels with higher values are also the most important for the task. Max pooling allows to look for the most important features along the matrix, downscaling it and removing for possible redundancy. The only parameter required is the mask dimensionality.

- *Flatten* shrinks the dimensions of the 2D array into a 1D array.
- *Dense* implements a fully connected layer; the first input parameter corresponds to the number of units, then the second parameter specifies the type of activation. For the last layer of the network, it applies the *Softmax* activation, which gives as output the prediction probability of each class. In this case, the total number of gestures analyzed were 17, which had to coincide with the units of the final fully connected layer.

2.2.2 LSTM - Long Short Term Memory

LSTM is a particular implementation of a RNN, that allows to overcome problems like the vanishing or exploding gradients. Moreover, this kind of RNN is capable of learning long-term dependencies. It usually requires a consistent amount of time to be trained, so its architecture should be very simple. Input data pass initially through an LSTM layer consisting of 350 units, then they undergo Batch Normalization. At the end, two fully connected layers allow to flatten data into a final array of dimensionality 17x1. The number of epochs was still set to 20; *RmsProp* optimizer was selected and it was used the decreasing learning rate strategy.



Figure 2.5: Data Glove LSTM architecture

Also the LSTM network accepts 2D data as input, but differently from the CNN, it analyses individually each channel, trying to look and learn for temporal dependencies, which could result as relevant peculiarities of a specific gesture. Below, it is possible to find the code used.

```

1 model = Sequential()
2 model.add(LSTM(350, return_sequences=True, input_shape = (300,22)))
3 model.add(BatchNormalization(axis=1))
4 model.add(Dense(100))
5 model.add(Activation('relu'))

```

```

6 model.add(Flatten())
7 model.add(Dense(17, activation = 'softmax'))

```

Listing 2.2: LSTM glove code

LSTM is a function that creates a LSTM layer. It considers in input: the number of units (350 in this case), a boolean variable to select whether to return the last output in the output sequence or the full sequence and, finally, the size of input data.

2.2.3 ResNet - Residual Network

Finally, the ResNet architecture was tested. It is a sort of CNN, with the presence of residual blocks: they are characterized by the fact that they receive both the output and input of the previous block as input. The final architecture proposed in this work is a variant of the known ResNet18 and it consists of one convolutional block, having 64 3x3 convolutional filters, Batch Normalization layer and MaxPooling layer, followed by three ResNet blocks and ending with one Fully Connected layer. The first two ResNet blocks have 64 convolutional filters 3x3, Batch Normalization, 64 convolutional filters 3x3 and the Add layer. The latest two are identical but the convolutional filters are 128. The final FC layer has 17 units, equivalent to the final number of gestures to be classified.

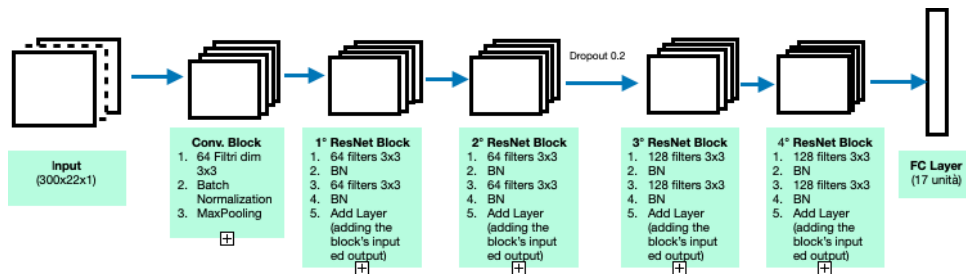


Figure 2.6: Data Glove ResNet architecture

For some kinds of tasks, these networks have significantly improved the performances of the classical CNN. For this reason, there was the willingness of evaluating their performance also for hand gesture classification tasks based on data glove. For the implementation of the network, it was first of all necessary to define two classes:

- *ResNetBlock*, which implements a standard resnet block with its sequence of layers characterized by a Convolutional Layer, a Batch Normalization layer, a ReLU activation, another Convolutional Layer, a Batch Normalization layer, an Additive Layer and a final ReLU activation.

```

1 import tensorflow as tf
2 class ResnetBlock(Model):
3     """
4     A standard resnet block.
5     """
6
7     def __init__(self, channels: int, down_sample=False):
8         """
9         channels: same as number of convolution kernels
10        """
11        super().__init__()
12
13        self.__channels = channels
14        self.__down_sample = down_sample
15        self.__strides = [2, 1] if down_sample else [1, 1]
16
17        KERNEL_SIZE = (3, 3)
18
19        self.conv_1 = Conv2D(self.__channels, strides=self.__strides
20        [0], kernel_size=KERNEL_SIZE, padding="same", kernel_initializer=
21        INIT_SCHEME)
22        self.bn_1 = BatchNormalization()
23        self.conv_2 = Conv2D(self.__channels, strides=self.__strides
24        [1], kernel_size=KERNEL_SIZE, padding="same", kernel_initializer=
25        INIT_SCHEME)
26        self.bn_2 = BatchNormalization()
27        self.merge = Add()
28
29        if self.__down_sample:
30            self.res_conv = Conv2D(self.__channels, strides=2,
31            kernel_size=(1, 1), kernel_initializer=INIT_SCHEME, padding="same")
32            self.res_bn = BatchNormalization()
33
34        def call(self, inputs):
35            res = inputs
36
37            x = self.conv_1(inputs)
38            x = self.bn_1(x)
39            x = tf.nn.relu(x)
40            x = self.conv_2(x)

```

```

36     x = self.bn_2(x)
37
38     if self.__down_sample:
39         res = self.res_conv(res)
40         res = self.res_bn(res)
41
42     # if not perform down sample, then add a shortcut directly
43     x = self.merge([x, res])
44     out = tf.nn.relu(x)
45     return out

```

Listing 2.3: ResNet Block code

- *ResNet18*, which implements a modified version of the well-known ResNet18 (Fig. 15).

```

1 class ResNet18(Model):
2
3     def __init__(self, num_classes, **kwargs):
4         """
5             num_classes: number of classes in specific classification
6             task.
7         """
8         super().__init__(**kwargs)
9         self.conv_1 = Conv2D(64, (3, 3), strides=1, padding="same",
10            kernel_initializer="he_normal")
11         self.init_bn = BatchNormalization()
12         self.pool_2 = MaxPool2D(pool_size=(3, 3), strides=2, padding="
13            same")
14         self.res_1_1 = ResnetBlock(64)
15         self.res_1_2 = ResnetBlock(64)
16         self.dp = Dropout(0.2)
17         self.res_2_1 = ResnetBlock(128, down_sample=True)
18         self.res_2_2 = ResnetBlock(128)
19         self.avg_pool = GlobalAveragePooling2D()
20         self.flat = Flatten()
21         self.fc = Dense(num_classes, activation="softmax")
22
23     def call(self, inputs):
24         out = self.conv_1(inputs)
25         out = self.init_bn(out)
26         out = tf.nn.relu(out)
27         out = self.pool_2(out)

```

```

25     for res_block in [self.res_1_1, self.res_1_2, self.dp, self.
    res_2_1, self.res_2_2]:
26         out = res_block(out)
27         out = self.avg_pool(out)
28         out = self.flat(out)
29         out = self.fc(out)
30     return out

```

Listing 2.4: ResNet18 code

These classes were then used to build the classifier, as reported below:

```

1 model = ResNet18(17)
2 model.build(input_shape = (None, 300, 22, 1))

```

Listing 2.5: classifier architecture code. In line {1} the class ResNet18 was called giving as input the number of final classes (the gestures) to be classified. In line {2} instead the model was built giving as input the shape of input data.

2.2.4 Conventional classifiers

The networks previously described were compared with three classifiers belonging to the conventional machine learning approaches. As described before, these approaches require hand-crafted features. To keep the models as simple as possible, the Root Mean Square features were extracted from each of the 22 channels, resulting in 22 features for each time window and thus shrinking the input data in a 1D array (1x22). The new data were normalized and then given as input to three classifiers: K-Nearest Neighbors, Decision Tree and Support Vector Machine.

- **k-Nearest Neighbors:** it is the simplest classifier in the machine learning field and it does not use an explicit model. The only two parameters for this classifier are: the number of neighbors and the type of geometrical distance, which can be a Manhattan or an Euclidean norm. To decide the optimal value for these two parameters, a hyperparameter tuning was done, using a 5-fold Cross Validation (CV) procedure and iterating for the four possibilities of the neighbors number (1,2,3,4) and for the two kind of distances. To implement the K-NN classifier, a function *KNeighborsClassifier* is available from the *Sci-kit Learn*. An example of code is the following.

```

1 from sklearn.neighbors import NearestNeighbors, KNeighborsClassifier
2 neigh = KNeighborsClassifier(n_neighbors = 1, algorithm='auto', p = 1)

```

Listing 2.6: k-NN glove code

n_neighbors is the number of neighbors to be considered, *algorithm* is the type of algorithm used to compute the nearest neighbors ('auto' option will decide autonomously the best one) and *p* is an integer number that stands for the distance that shall be considered (1 Manhattan distance, 2 Euclidean distance).

- **Decision Tree:** this classifier is widely used in clinical practice because it reflects the process of human decision making. Indeed, it makes splits to divide the overall set in subsets that contain similar samples, each associated to the same label. The two parameters that can be tuned in this case are the maximum depth of the tree (the more is the depth, the more is the risk of overfitting) and the splitting criterion, that evaluates the node impurity. Different approaches can be considered, more precisely the gini index or the entropy criterion. The gini index uses the following formula:

$$G = \sum_k P(k)[1 - P(k)] \quad (2.1)$$

where *k* represents the classes and $P(k)$ is the ratio at which the *k*th class is present in the node. The Entropy criterion instead uses the formula below:

$$E = \sum_k P(k)[\log P(k)] \quad (2.2)$$

It is clear how in both cases, if in the node there is complete purity, the index is exactly 0. Following the same approach of the k-nn classifier, the tuning was done. Code-implementing speaking, from *Sci-kit learn* is available a function called *DecisionTreeClassifier*, which can be implemented as in the following code:

```
1 from sklearn.tree import DecisionTreeClassifier
2 clf = DecisionTreeClassifier(random_state=0, criterion = 'gini',
   max_depth = 30)
```

Listing 2.7: Decision Tree glove code

where *random_state* is a variable for repeatability.

- **Support Vector Machine:** it is the most complex classifier and it works trying to find a hyperplane that divides the samples belonging to different classes. The Support Vector Machine (SVM) is by default a binary classifier but it can be adapted to perform multi-class classification using a one-vs all approach: for each class, it creates two sets, one having the samples of the considered class, the other having the samples belonging to the remaining classes. The parameters that influence the performances of the model are

three. First, the kernel, which is the function that is exploited to build the hyperplane: it can be linear, polynomial, sigmoidal or a radial basis function. Secondly, a regularization parameter C defines how strongly the model fits the data and it influences the probability of overfitting. Finally, another parameter γ comes when the radial basis function is used. Then, if the most suitable kernel results as the polynomial, its degree should be tuned, knowing that the higher the degree, the higher the risk of overfitting because of the increased complexity of the model. For the code implementation, the function *SVC* from *Sci-kit learn* framework was considered:

```
1 from sklearn.svm import SVC
2 from sklearn.pipeline import make_pipeline
3 svm = make_pipeline(SVC(C=0.1, kernel='rbf', gamma=0.1))
```

Listing 2.8: SVM glove code

Since SVM may be computationally heavy, thus requiring a lot of time it was not used the same approach for the parameters tuning. More precisely, the classifier, for each possible combination of parameters, was trained over the same training set and tested over the same validation set.

2.3 sEMG

In this section, an approach for performing a hand gesture recognition by using only sEMG signals was considered.

The database used in this work provided noise-filtered sEMG data. The sEMG data processing involved:

1. Considering non overlapping windows of 150 ms. This choice was made for the sake of comparability with the performances of the glove networks;
2. Subdivision in training, validation and test set; this operation was made identically to the one of data glove;
3. Amplitude normalization of the training, validation and test set, respectively to the peak of the whole training set. The amplitude normalization is a technique commonly used in literature and it aims to reduce the high-variability that characterizes the sEMG signal;
4. Evaluation and extraction for each window of dimension 300x12 (the sampling rate was still 2 kHz) of the following hand-crafted features:

- (a) ZC (eq. 1.13);
- (b) SSC (eq. 1.16);
- (c) RMS (eq. 1.9);
- (d) IEMG (eq. 1.1);
- (e) MM2 (equivalent to the variance, eq. 1.5);
- (f) MM3 (equivalent to the TM3, eq. 1.6);
- (g) MM4 (equivalent to the TM4, eq. 1.7);
- (h) MM5 (equivalent to the TM5, eq. 1.8);
- (i) LOG (eq. 1.10);
- (j) DASDV (eq. 1.12);;
- (k) MAV1 (eq. 1.3);
- (l) MAV2 (eq. 1.4);
- (m) MYOP (eq. 1.14);
- (n) AAC (eq. 1.11);
- (o) WAMP (eq. 1.15);
- (p) MNF (eq. 1.17);
- (q) PKF (eq. 1.19);
- (r) MNP (eq. 1.20);
- (s) TTP (eq. 1.21);
- (t) SM1 (eq. 1.22);
- (u) SM2 (eq. 1.23);
- (v) SM3 (eq. 1.24);
- (w) FR (eq. 1.25);
- (x) VCF (eq. 1.26);

Each feature was extracted separately for each channel. The data was then organized in matrix of dimensions 23x12, the number of electrodes as columns and the number of features as rows.

Again, both deep learning approaches and conventional machine learning approaches were tested.

2.3.1 CNN

The implementation of a CNN is justified because, with data organized in this way, the network could possibly learn some interesting deep features, due to the relationships between channels and hand-crafted features, connected with the gestures. Two variants were tested:

1. Using data organized as 1D array, having size (1x276). This was possible thanks to the methods *reshape* of the *numpy* library; more in details, it works by concatenating the rows along the column dimension.

In this configuration the architecture consisted of three blocks. In the first block, 32 filters of dimensions 24x1 are applied, then neurons are activated with a non linear activation function, the ReLU, which is typically used in convolutional neural networks. Finally, to reduce the risk of overfitting, a Dropout layer is inserted, with a rate of 0.2. The second convolutional block is identical, but at the beginning it has 64 filters of dimensions 48x1. The last instead has just 128 filters (96x1) and the ReLU activation. Then, the network shrinks the data in a 1D array thanks to a flatten layer and the final classification is made from the final Dense layer, consisting of 17 units and using a softmax activation function.

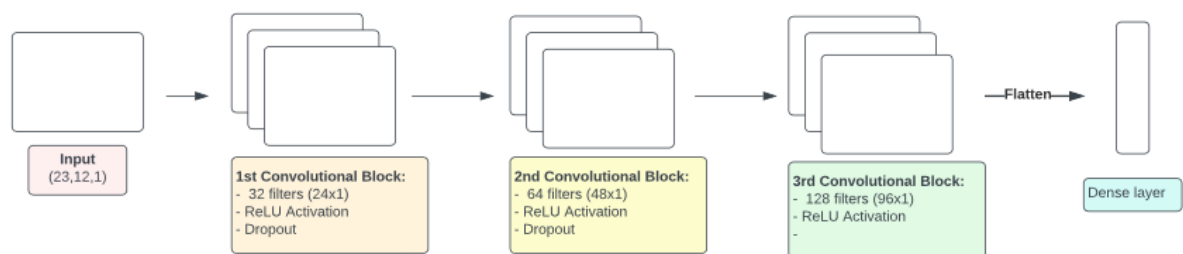


Figure 2.7: CNN architecture with sEMG, 1st configuration

```
1 model = Sequential()
2 model.add(Conv2D(32, (24, 1), input_shape=(276, 1, 1)))
3 model.add(Activation('relu'))
4 model.add(Dropout(0.2))
5 model.add(Conv2D(64, (48, 1)))
6 model.add(Activation('relu'))
7 model.add(Dropout(0.2))
8 model.add(Conv2D(128, (96, 1)))
9 model.add(Activation('relu'))
10 model.add(Flatten())
```

```
11 model.add(Dense(17, activation='softmax'))
```

Listing 2.9: CNN sEMG code (1st configuration)

In order to reduce the computational complexity, it was built a variation of this network to deal just with time-domain features. This led to input of size (168,1,1) and the architecture slightly changed. In particular the dimensions of the filters were: (12x1, 24x1, 48x1).

- Using data organized as 2D matrix 23x12. In this second configuration, the network had 4 convolutional blocks with the same structure: first the convolution with filters is made, then the ReLU activation function is applied and finally there is the Dropout layer. The filters applied are in sequence: 32 filters 2x2, 64 filters 2x2, 128 filters 3x3, 256 filters 4x4. The 4th convolutional block has not the Dropout, but it is connected with a Dense layer of 17 units.

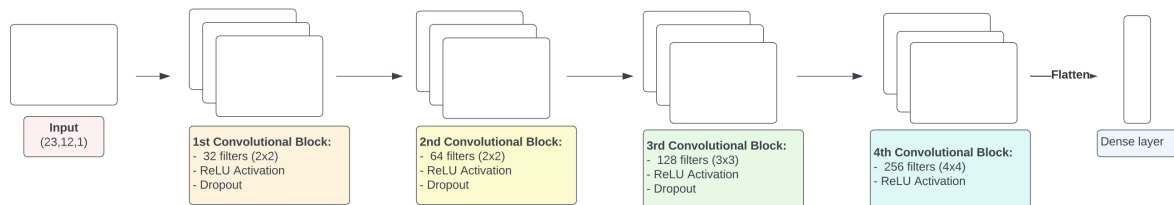


Figure 2.8: CNN architecture with sEMG, 2nd configuration

```
1 model = Sequential()
2 model.add(Conv2D(32, (2, 2), input_shape=(23,12,1)))
3 model.add(Activation('relu'))
4 model.add(Dropout(0.2))
5 model.add(Conv2D(64, (2,2)))
6 model.add(Activation('relu'))
7 model.add(Dropout(0.2))
8 model.add(Conv2D(128, (3,3)))
9 model.add(Activation('relu'))
10 model.add(Dropout(0.3))
11 model.add(Conv2D(256, (4,4)))
12 model.add(Activation('relu'))
13 model.add(Flatten())
14 model.add(Dense(17, activation='softmax'))
```

Listing 2.10: CNN sEMG code (2nd configuration)

In a second moment, accordingly to the previous configuration, only time-domain features were considered. This required some modifications to the previous network: 2x2 filters

were employed for all the convolutional blocks but the 2nd, in which a 3x3 dimension was chosen instead.

In order to have a more reliable estimates of the network performances, a Cross Validation (CV) procedure was done; such process consisted in dividing the overall training set in k folds and has k steps. At each step, one fold was selected as validation set; the remaining k-1 folds formed the training sets. After the training of the model, the accuracy over the training set and validation set was determined and saved, such that at the end, a more robust estimate of the model accuracy was obtained by averaging the training and validation accuracy of each stage. This procedure was possible for sEMG data thanks to the limited computational requirements, which instead were too high for glove data, thus preventing the CV execution.

The *Scikit Learn* library provides a machine learning tool to implement the Cross Validation, called *KFold*: specifying the number of folds and the willingness of shuffling or not, it splits the dataset into k consecutive folds. At each step of the procedure, one of them is used as validation set and the remaining k-1 as training sets. Once the estimates of the performances are all established, the network is trained again using the whole training set (including the validation one).

```
1 from sklearn.model_selection import KFold
2 acc_per_fold = []
3 loss_per_fold = []
4
5 # Define the K-fold Cross Validator
6 kfold = KFold(n_splits=5, shuffle=True)
7
8 # K-fold Cross Validation model evaluation
9 fold_no = 1
10 for train, test in kfold.split(inputs, targets):
11
12     # Define the model architecture
13     model = Sequential()
14     model.add(Conv2D(32, (23, 1), input_shape=(276, 1, 1)))
15     model.add(Activation('relu'))
16     model.add(Dropout(0.2))
17     model.add(Conv2D(64, (46, 1)))
18     model.add(Activation('relu'))
19     model.add(Dropout(0.2))
20     model.add(Conv2D(128, (92, 1)))
21     model.add(Activation('relu'))
```

```

22     model.add(Flatten())
23     model.add(Dense(17, activation='softmax'))
24
25     # Compile the model
26     model.compile(optimizer='adam',
27                  loss='categorical_crossentropy',
28                  metrics=['accuracy'])
29
30
31
32     # Generate a print
33     print('
-----
')
34     print(f'Training for fold {fold_no} ...')
35     history = model.fit(inputs[train], targets[train],
36                        batch_size=36,
37                        epochs=30,
38                        verbose="auto")
39
40     # Generate generalization metrics
41     scores = model.evaluate(inputs[test], targets[test], verbose=0)
42     print(f'Score for fold {fold_no}: {model.metrics_names[0]} of {scores
43           [0]}; {model.metrics_names[1]} of {scores[1]*100}%')
44     acc_per_fold.append(scores[1] * 100)
45     loss_per_fold.append(scores[0])
46
47     # Increase fold number
48     fold_no = fold_no + 1

```

Listing 2.11: code for the CV implementation for the specific example of the first CNN configuration; To use properly the Sci-kit learn tool KFold it is necessary to build a for loop that iterates through the indices that defines at each step the training and validation set.

2.3.2 Random Forest

In order to make a comparison with the performances of classical machine learning approaches, a Random Forest (RF) classifier was built. This choice is justified because of the limited computational requirements and so computational time to train the network, which were instead too high to implement efficiently KNN or SVM. Moreover, since Random Forest works by implementing a consistent number of decision trees with an intrinsic randomness and then av-

eraging their predictions, its results can be considered robust without the necessity of making a Cross Validation procedure. Furthermore, RF comes with the possibility of evaluating the importance of a feature in making the classification. The so-called feature importance is related to the number of times the feature is considered to make the best split possible at each step. This information could be used to select the features. To be trained, the classifier requires the specification of few hyperparameters, like:

- **Number of estimators:** this is the actual number of decision trees that are built;
- **Number of features:** the number of features to be considered when looking for the best split;
- **Minimum number of samples per leaf:** this is the minimum number of samples required to be at a leaf node. It can be considered as a stopping criterion.
- **Criterion:** the criterion that has to be considered when the impurity of a node is evaluated.

In this work, a number of 300 estimators was used, in order to guarantee a robustness of the network, and the number of features were not specified, thus using the default setting that defines the square root of the total number of the calculate hand-crafted features as the number of features parameter. Then, it was adopted a specific strategy to evaluate the optimal minimum number of samples per leaf and the criterion, similarly to the one of the Support Vector Machine used for the analysis of data glove (see 2.2.4). In particular, using the training set and validation set defined in the section 2.1.4, for each combination of the two parameters, it was trained a Random Forest classifier using the first subset; then, to select the best combination, the performances over the validation set were considered.

```
1 from sklearn.ensemble import RandomForestClassifier
2 rf = RandomForestClassifier(n_estimators = 100, min_samples_leaf = 2,
    verbose = 1, criterion = 'gini' )
```

Listing 2.12: RF sEMG code

Once the best hyperparameters were chosen, the feature selection procedure followed, in order to understand which were the most significative features that the model has looked up to perform the classification. Subsequently, it was taken advantage of this information to shrink the number of features that the model has finally to consider, leading to a diminished computational cost and so a reduced computational time. At the end, the classifier was trained over the complete training set (including the validation set) and evaluated on the test set.

For the development of the RF classifier, the function *RandomForestClassifier* was imported from the *sklearn* framework. If not specified, the number of features considered at each step is by default equal to the square root of the total number of features.

2.4 Data glove and sEMG: the fusion approach

The fusion is a technique widely used in literature for hand gesture classification: it consists in giving in input to the classifier the combination of two or more kinds of signals, so that it could have a more complete information to analyze. This technique has the intrinsic potentiality to increase the generalization ability of the classifier, thus its classification performances. As an example, in the study of Ceolini, et. Al, 2020 the group worked on optimizing a framework that integrated two complementary systems: electromyography (EMG) signals from the muscles and visual information. In the work of Colli-Alfaro et. Al, 2019, instead, a user-independent gesture classification method based on a sensor fusion technique using surface electromyography (EMG) and an inertial measurement unit (IMU) was presented.

These studies have suggested to investigate the performances achievable by combining data glove and sEMG data in this work. The choice was justified because data glove have the advantage of being less user-dependent due to their lower variability: they give joint-angle measurements, which are quite reproducible within a set of different subjects, as long as the sensors are displaced correctly over the hand. On the other hand, sEMG signal contains neuromuscular information that could be strongly related to the gesture.

As reference for the fusion development, the work of Tortora et. Al, 2020 was considered. There, a fusion between EEG and EMG signals was done to decode gait activity. More precisely, the approach consisted in designing two completely separate and independent networks, one working with EMG only, the other with EEG. Both of them had to perform classification, independently one from the other; then, only at the end, the two classifications were merged using a Bayesian Belief Fusion approach.

The Bayesian Belief Fusion is a method that looks at the uncertainty of the classifiers in determining the most reliable one for a given gesture to be classified. More precisely, for each class (gesture) a specific parameter called *Belief* is determined. This parameter can be interpreted as a measure of the assignment correctness. It is evaluated from priori and conditional probability coming from the K classifiers involved:

$$Bel(c_i) = P(c_i) \frac{\prod_{k=1}^K P(c_i | e_k = c_j)}{\prod_{k=1}^K P(c_i)} \quad (2.3)$$

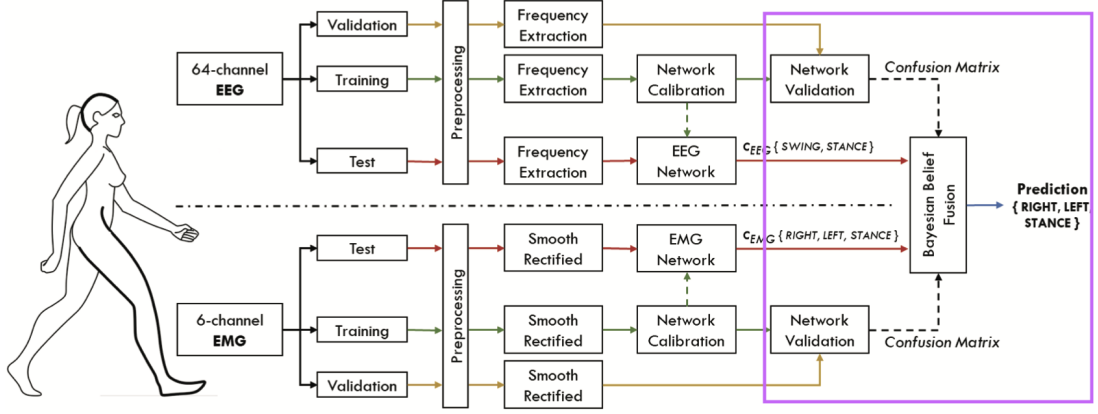


Figure 2.9: Schematic representation of the hybrid human-machine interface proposed by Tortora et. Al. In purple, it is highlighted the bayesian fusion. [19]

where $P(c_i)$ is the priori probability of class i and $P(c_i|e_k = c_j)$ is the probability that the true class is i when the $k - th$ classifier outputs class j . The conditioned probability is properly of the $k - th$ classifier; it reflects its reliability and it is evaluated from its confusion matrix. Given the confusion matrix

$$CM = \begin{bmatrix} n_{11}^k & \dots & n_{1j}^k & \dots & n_{1M}^k \\ \dots & \dots & \dots & \dots & \dots \\ n_{i1}^k & \dots & n_{ij}^k & \dots & n_{iM}^k \\ \dots & \dots & \dots & \dots & \dots \\ n_{M1}^k & \dots & n_{Mj}^k & \dots & n_{MM}^k \end{bmatrix} \quad (2.4)$$

where n_{ij}^k represents the number of samples belonging to the $i - th$ class that were predicted as the $j - th$ class by the $k - th$ classifier, the condition probability $P(c_i|e_k = c_j)$ is evaluated as:

$$P(c_i|e_k = c_j) = \frac{n_{ij}^k}{\sum_{i=1}^M n_{ij}^k}, i = 1, \dots, M; j = 1, \dots, M \quad (2.5)$$

Accordingly to the work of Tortora et. Al, 2020, the same prior probability for all the classes was used.

In this thesis the Bayesian Belief Fusion was implemented by considering the best classifier for glove data and the best classifier for sEMG data. After the training using the training set alone, the confusion matrices were evaluated on the validation set. The bayesian technique was finally applied for the predictions over the test set: for each predicted gesture, the method evaluates the belief for each class and outputs the class with the maximum belief. Along with the new (in some sense corrected) predictions, the performances of the method were evaluated and compared with that obtained without fusion.

Technically speaking, the bayesian fusion method required the use of a pre-defined function called *confusion_matrix* from the Scikit-learn, which accepts in input the ground truth target

values and the classifier estimate targets and it provides in output the confusion matrix. Using this function, the confusion matrices of the two best classifiers obtained with sEMG and data glove were evaluated on the validation set. The condition probabilities were then extracted from the confusion matrices and for each sample of the test set the belief vector was determined. The final prediction corresponds to the class that had the maximum belief value.

```

1 cond_prob1 = np.zeros(conf_matr1.shape)
2 cond_prob2 = np.zeros(conf_matr2.shape)
3 bel = np.zeros(17)
4 n_gest = 17
5 pred_correct = np.zeros(N)
6
7 for i in range(0,n_gest):
8     for g in range(0,n_gest):
9         cond_prob1[i,g] = conf_matr1[i,g]/sum(conf_matr1[:,g])
10
11
12 for i in range(0, n_gest):
13     for g in range(0, n_gest):
14         cond_prob2[i,g] = conf_matr2[i,g]/sum(conf_matr2[:,g])
15
16 bel = np.zeros(17)
17
18 for i in range(0, N):
19     p1 = pred_test1[i]
20     p2 = int(pred_test2[i])-1
21     for j in range(0, n_gest):
22         bel[j] = 1/17*cond_prob1[j, p1]*cond_prob2[j, p2]
23     index = np.where(bel == np.amax(bel))
24
25     pred_correct[i] = index[0][0]
26     bel = np.zeros(17)
27
28
29 print(classification_report(y_test_new,pred_correct))

```

Listing 2.13: Bayesian fusion implementation. "cond_prob1" and "cond_prob2" are the conditioned probability of the two independent classifiers evaluated from their confusion matrices; "pred_test1" and "pred_test2" are their respective predictions over the test set. "bel" is the belief vector. It is possible to see how the corrected prediction coming from the fusion corresponds to the class with the maximum belief value.

Chapter 3

Results

In this chapter, the results will be presented: first, data glove based classifiers are analyzed, then classifiers for sEMG data. At the purpose, the *Sci-kit Learn* framework provides the function called *classification_report*, which allows to investigate the most common classification metrics as *precision*, *recall*, *f1-score*, *accuracy* and does also a *macro average* and *weighted average* of the first three. Given the predicted - truth values grid:

		Truth	
		Positive	Negative
Predicted	Positive	TP	FP
	Negative	FN	TN

- **Precision** is the fraction of correct classifications divided by the total number of classifications. It can be called positive predicted value and it is defined as

$$Precision(PPV) = \frac{TP}{TP + FP} \quad (3.1)$$

A model having high precision means that it succeeds in predicting a sample as positive.

- **Recall**, also known as **sensitivity**, is the rate of correct positive classifications among the total number of positive outputs. It is evaluated by:

$$Recall(TPR) = \frac{TP}{TP + FN} \quad (3.2)$$

A model with low recall is not able to find all the positive samples in the dataset.

- **F1-score**: it is a combination of precision and recall and takes into account the eventual data imbalance. Indeed, the ideal configuration is achieving both high precision and

recall. For this reason, it is defined as an harmonic mean between the two previous metrics:

$$f1 - score = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (3.3)$$

An ideal model should have high f1-score, meaning that both Precision and Recall are high.

- **Accuracy.** This is the simplest metric that can be taken into account, it refers to the correct number of classifications among all the others, but without taking into consideration the possible data imbalance. It is computed by:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \quad (3.4)$$

Finally, the fusion outcome is presented.

3.1 Data glove

The following classifiers were developed and tested:

1. **CNN**, with three Convolutional blocks; the first block had 32 filters 3x3, a ReLU activation, a Batch Normalization and a MaxPooling 2x1 layer. The second was identical but with 64 filters. The third had 128 filters 2x2. Finally, the network ended with a Dense layer made of 17 units.
2. **LSTM**, with an LSTM layer of 350 units, a Batch Normalization layer, a Dense layer with 100 units and finally with a Dense layer of 17 units.
3. **ResNet**, with a Convolutional Layer that applied 64 filters 3x3, a Batch Normalization layer, a Max Pooling 3x3, two Resnet Blocks of 64 units, a Dropout, two other Resnet Blocks with 128 units, a Global Average Pooling and finally a Dense layer with 17 units.
4. **KNN**.
5. **Decision Tree**.
6. **SVM**.

3.1.1 CNN

In order to test this classifier, the number of epochs was set to 20. With this choice, the classifier took more than 9 minutes to complete the training, at the end of which it reached an accuracy over the training set of almost 88% and 82% over the validation set. It is interesting to analyze the trends of these two values, which can be seen in Fig. 3.1. The accuracy trends suggest that the network is not affected by overfitting: at the final epoch, the training and validation accuracy values are quite close to each other. This statement can be confirmed by observing the loss trends in Fig. 3.2: the loss is decreasing as the number of epochs increases, meaning that the network is learning the relationship between training data and their labels, and again the distance between the two final values is not indicating the presence of the overfitting problems. Furthermore, as expected, the final training loss is lower than the validation one. The network was then evaluated over the test set.

Performance values were evaluated for the considered network and reported in Tab. 3.1; since the problem consisted in classifying among 17 different gestures but the metrics refer to a binary problem, they were calculated using a one-vs-all approach: for each gesture, all the corresponding samples were considered as positive samples, while all the others were considered as negative samples.

By analyzing the classification metrics, it is clear how the network had some difficulties in classifying some gestures. The confusion matrix can be therefore evaluated in order to understand which are the gestures that the network misclassifies and, more interestingly, which are the classes of gestures that are causing the misunderstanding.

The same architecture was finally tested only on a subset of gestures, from number 1 to number 8. In this case, the network performances increased thanks to the diminished chance level. Indeed, the number of possible classes was decreased. As mentioned before, the classifier had the same identical structure of the previous one, except for the final dense layer, that for this task had only 8 units, one unit for each gesture. Obviously, this reduced classifier has less parameters, going from more than 1 million to slightly less than 700 thousands. At the end of the training phase, the final training and validation accuracy were respectively 96% and 93%. The network was evaluated over the test set: the classification metrics are reported in Tab. 3.2 and the confusion matrix in Fig. 3.4.

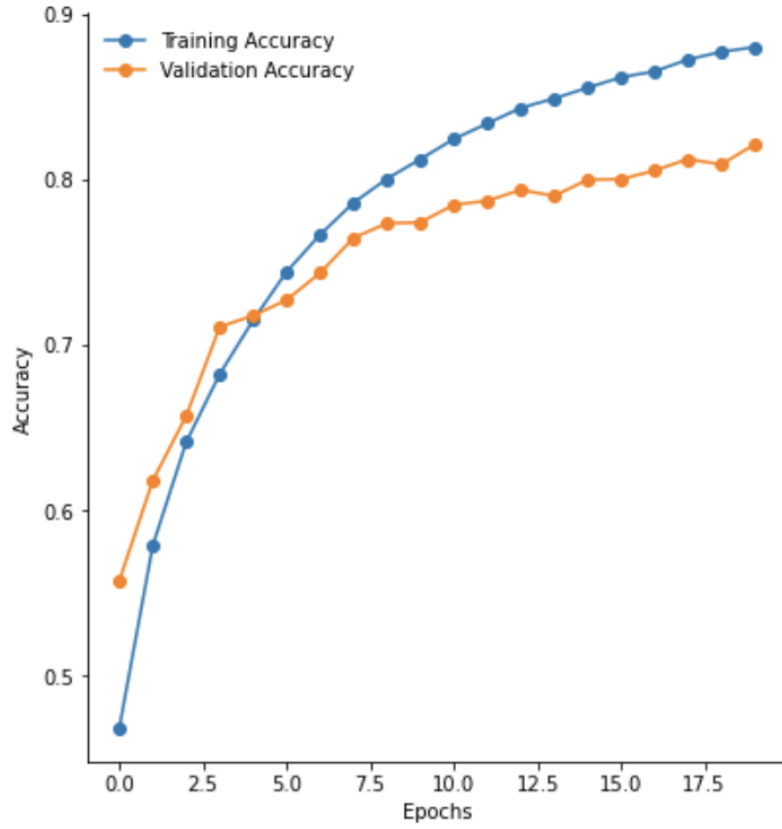


Figure 3.1: Accuracy trends of the validation and training set during the training epochs of the network.

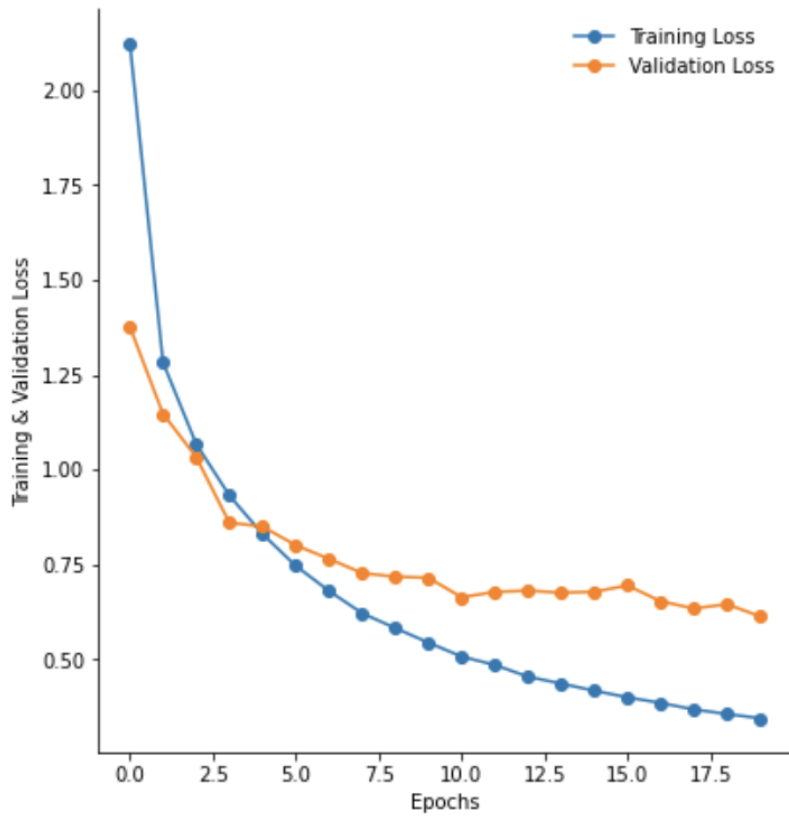


Figure 3.2: Loss trends of the validation and training set during the training epochs of the network.

		precision	recall	f1-score
Gesture	1	0.94	0.93	0.94
	2	0.91	0.84	0.87
	3	0.86	0.94	0.90
	4	0.82	0.81	0.81
	5	0.83	0.85	0.84
	6	0.90	0.84	0.87
	7	0.89	0.94	0.92
	8	0.86	0.62	0.72
	9	0.70	0.67	0.69
	10	0.72	0.79	0.76
	11	0.72	0.66	0.69
	12	0.64	0.79	0.70
	13	0.83	0.65	0.73
	14	0.73	0.81	0.77
	15	0.85	0.91	0.88
	16	0.69	0.76	0.72
	17	0.78	0.72	0.75
accuracy				0.80
macro average		0.80	0.80	0.80
weighted average		0.81	0.80	0.80

Table 3.1: Classification metrics for the CNN using data glove.

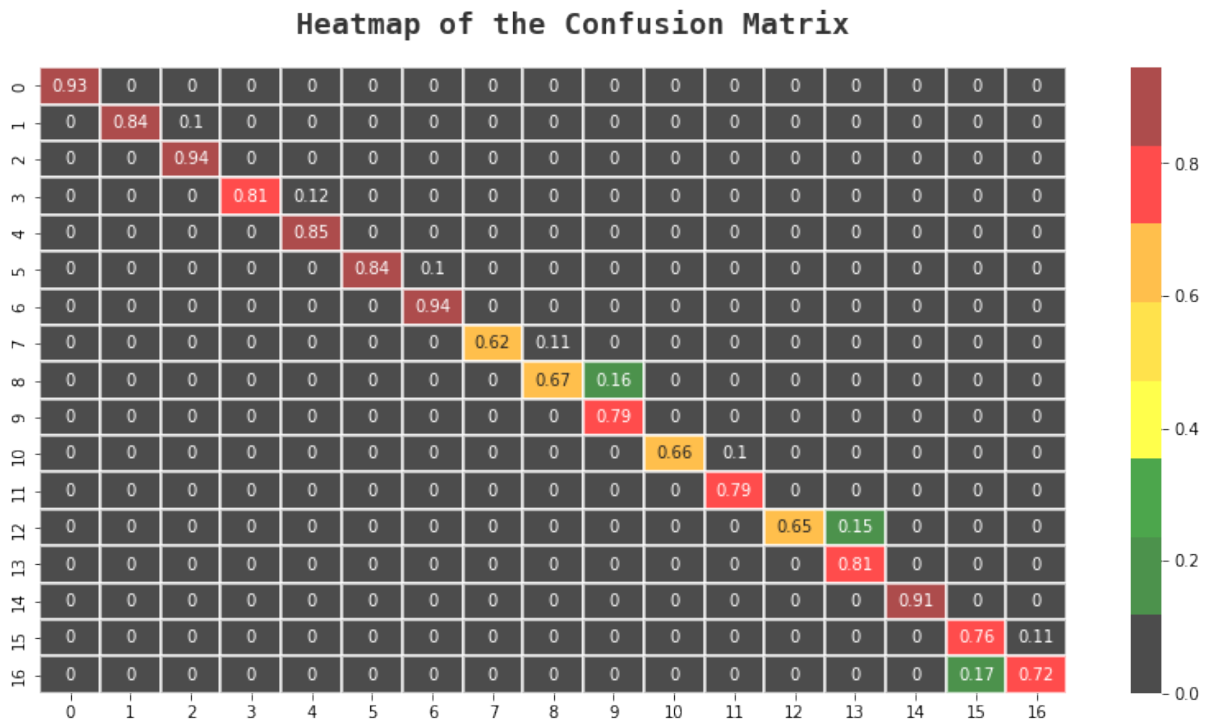


Figure 3.3: Confusion matrix of the CNN. Note that a threshold is applied: all the values lower than 0.1 are set as 0.

		precision	recall	f1-score
Gesture	1	0.95	0.95	0.94
	2	0.93	0.89	0.91
	3	0.91	0.92	0.92
	4	0.86	0.79	0.82
	5	0.83	0.92	0.87
	6	0.87	0.89	0.88
	7	0.92	0.93	0.92
	8	0.91	0.88	0.89
	accuracy			0.90
	macro average	0.90	0.90	0.90
	weighted average	0.90	0.90	0.90

Table 3.2: Classification metrics over the test set for the reduced set of gestures.

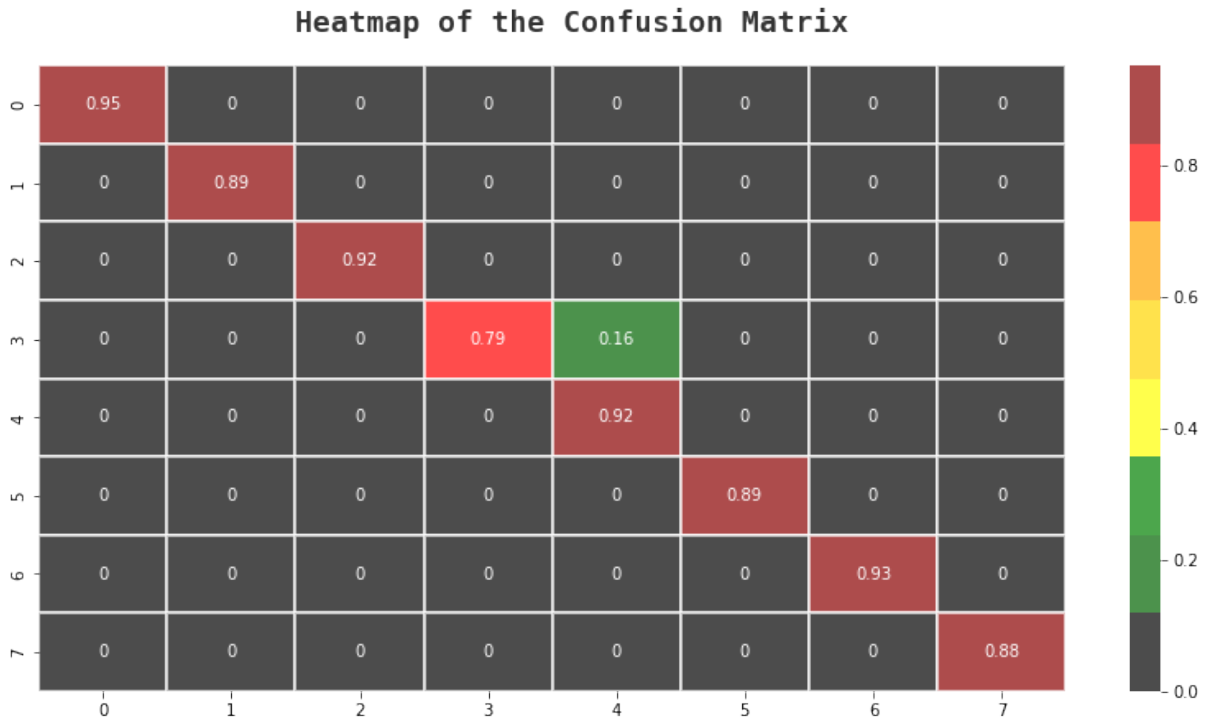


Figure 3.4: Confusion matrix for the reduced gesture set configuration with the CNN.

3.1.2 LSTM

The LSTM network had more than one million parameters and it took more than 20 minutes to be trained. As for the CNN, the LSTM is not affected by overfitting because of the unsubstantial difference between the training and validation accuracy, which reached 91% and 85% respectively at the end of the training. This can be easily viewed from the accuracy and loss trends over the training epochs, reported in Fig. 3.5.

Using the test set, the performances of the LSTM were evaluated by computing the classification metrics described before (Tab. 3.3) and the confusion matrix (Fig. 3.6).

The reduced set of gestures was evaluated also with the LSTM network. As for the CNN, the architecture was kept identical and it was modified only the final layer, which had in this case 8 units. The number of parameters shrunk to almost 800 thousands and the number of epochs was so reduced to 15. The final training accuracy was 92%, while the validation accuracy was 89%. The network was then evaluated over the test set, and it produced the classification metrics reported in the table Tab. 3.4. The confusion matrix is displayed in Fig. 3.7.

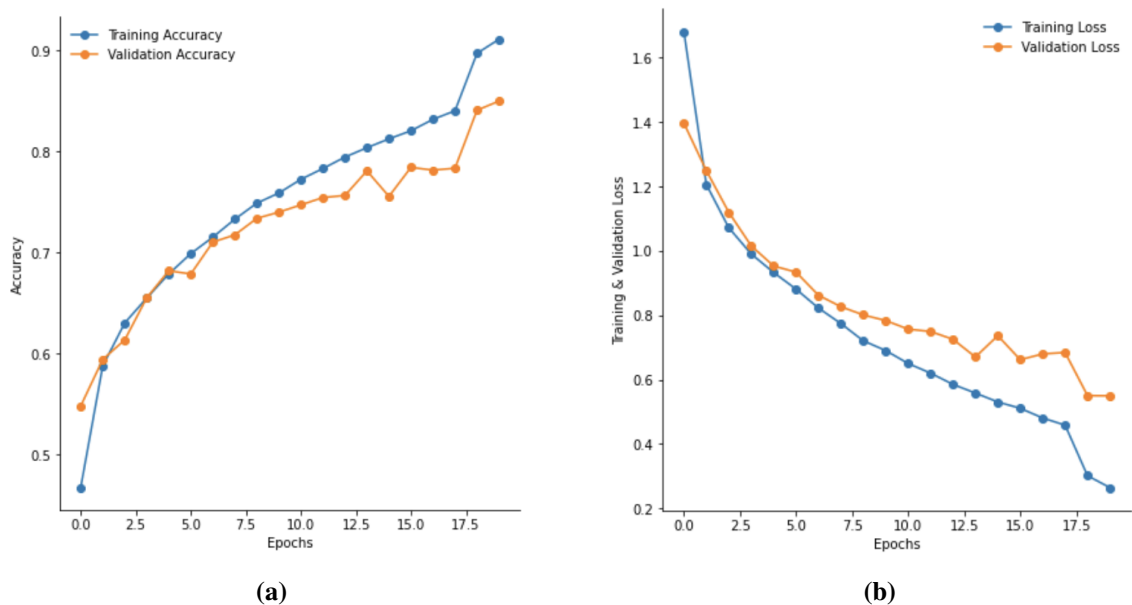


Figure 3.5: (a) Accuracy trends of the LSTM with data glove. (b) Loss trends of the LSTM with data glove.

		precision	recall	f1-score
Gesture	1	0.95	0.96	0.96
	2	0.93	0.89	0.91
	3	0.91	0.94	0.93
	4	0.85	0.80	0.83
	5	0.83	0.91	0.87
	6	0.93	0.87	0.90
	7	0.93	0.94	0.93
	8	0.81	0.79	0.80
	9	0.73	0.75	0.74
	10	0.77	0.79	0.78
	11	0.71	0.70	0.70
	12	0.77	0.73	0.75
	13	0.77	0.74	0.76
	14	0.81	0.81	0.81
	15	0.89	0.93	0.91
	16	0.70	0.75	0.73
	17	0.76	0.73	0.74
	accuracy			0.83
	macro average	0.83	0.83	0.83
	weighted average	0.83	0.83	0.83

Table 3.3: Classification metrics for the LSTM using data glove.

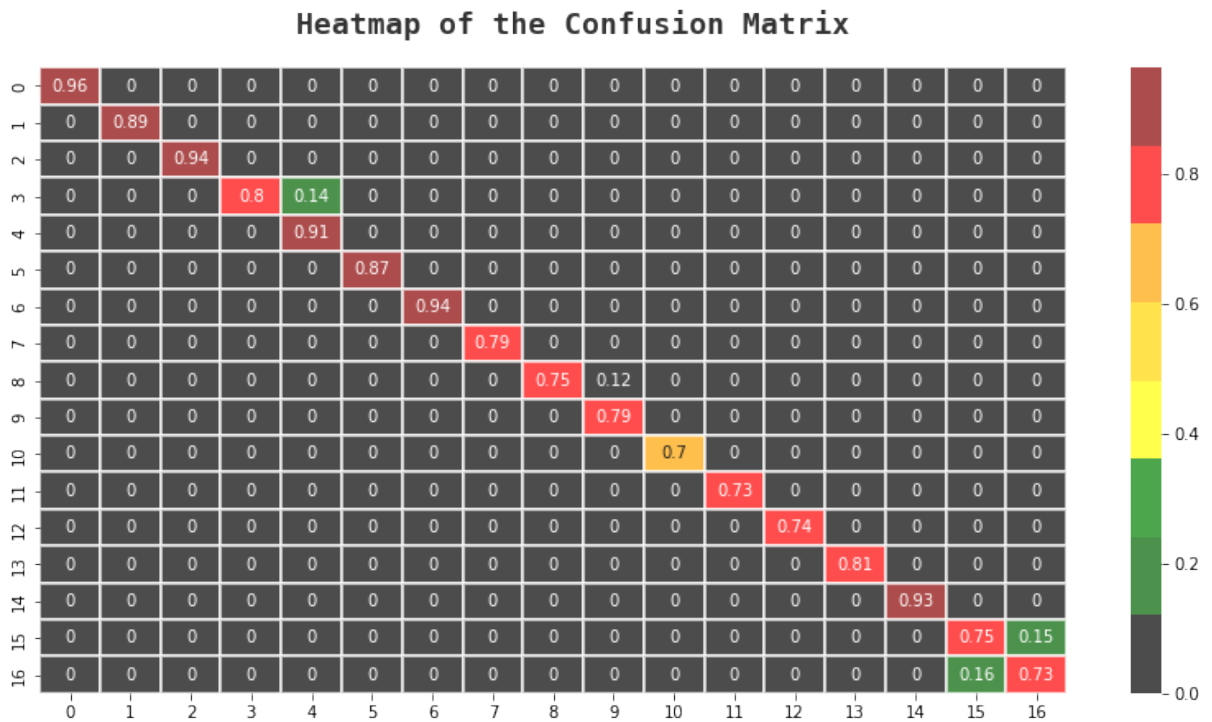


Figure 3.6: Confusion matrix of the LSTM over the test set. Note that a 0.1 threshold is applied: all the values below it were set to 0.

		precision	recall	f1-score
Gesture	1	0.93	0.94	0.93
	2	0.88	0.92	0.90
	3	0.93	0.91	0.92
	4	0.85	0.79	0.82
	5	0.88	0.82	0.85
	6	0.81	0.82	0.81
	7	0.86	0.93	0.89
	8	0.91	0.88	0.89
	accuracy			0.88
	macro average	0.88	0.88	0.88
	weighted average	0.88	0.88	0.88

Table 3.4: Caption

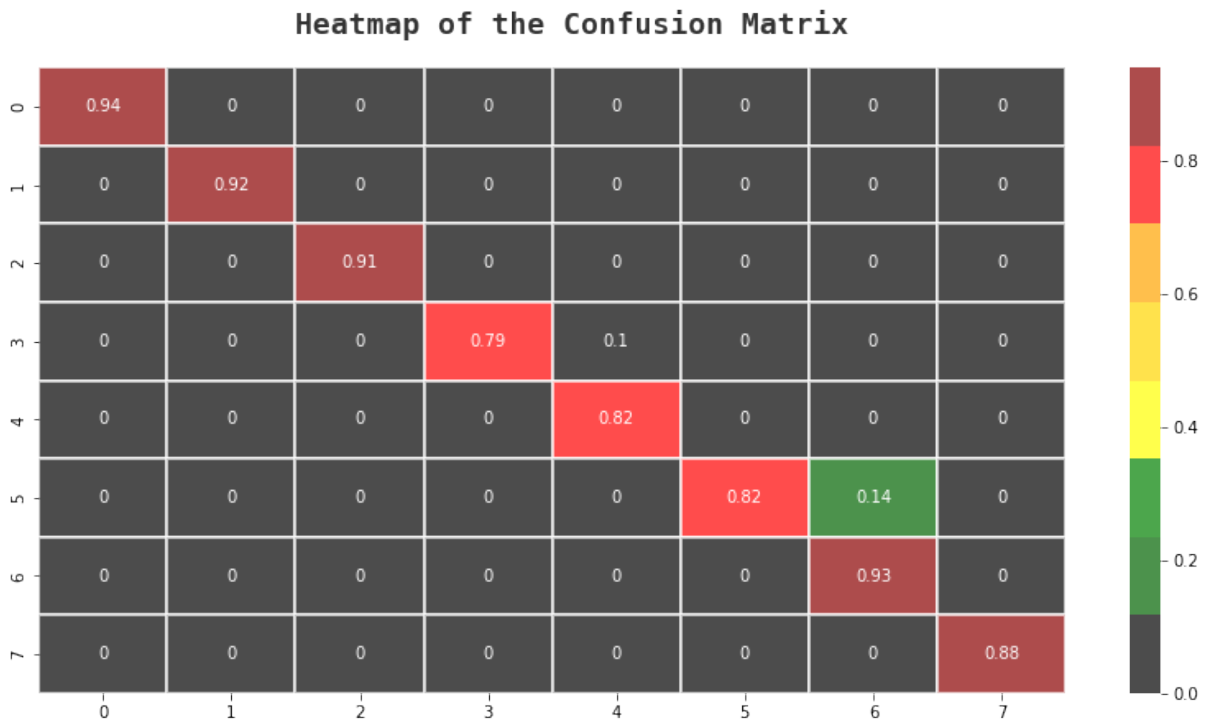


Figure 3.7: Confusion Matrix evaluated over the test set.

3.1.3 ResNet

The ResNet is a modified version of the ResNet18. It had almost 600 thousands parameters, thus a substantially lower number than the previous two network configurations. The number of epochs was reduced to 15 and it took around 12 minutes to complete the training, at the end of which the accuracy were 98% and 91% for the training and validation set respectively. These results suggest how this classifier is not overfitting data too, as the accuracy and loss trends are confirming in Fig. 3.8.

The performances over the test set were evaluated through the computation of the same classification metrics. The results are shown in the Tab. 3.5. Even ResNet has some problems in well-identifying some gestures: the confusion matrix helps in clarifying the type of misclassifications (Fig. 3.9).

Accordingly to what was done for the previous two networks, the ResNet was evaluated with the reduced gestures set. The only layer that was modified was again the final dense layer, which had 8 units. The number of parameters did not significantly change and the number of epochs was kept equal to 15. The training procedure lasted slightly more than 6 minutes and it allowed to reach a training accuracy of 99% and a validation accuracy of 96%. The network performances were evaluated over the test set, and the classification metrics reported in table Tab. 3.6 were obtained. The confusion matrix highlights which gestures the network is confusing (Fig. 3.10).

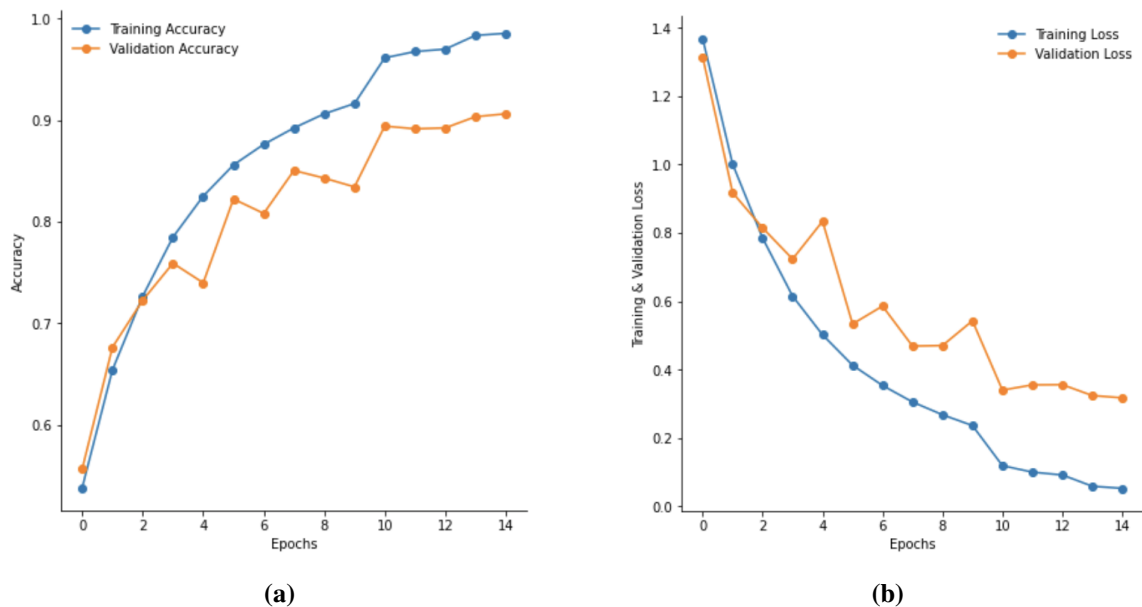


Figure 3.8: (a) Training and validation accuracy trends of the ResNet. (b) Training and validation loss trends of the ResNet

		precision	recall	f1-score
Gesture	1	0.98	0.96	0.97
	2	0.94	0.93	0.93
	3	0.94	0.90	0.94
	4	0.88	0.75	0.85
	5	0.84	0.89	0.87
	6	0.92	0.85	0.91
	7	0.93	0.91	0.93
	8	0.87	0.82	0.84
	9	0.79	0.64	0.79
	10	0.82	0.84	0.83
	11	0.81	0.72	0.79
	12	0.75	0.78	0.80
	13	0.82	0.85	0.82
	14	0.84	0.83	0.85
	15	0.93	0.94	0.93
	16	0.84	0.69	0.80
	17	0.79	0.82	0.84
	accuracy			0.87
	macro average	0.86	0.86	0.86
	weighted average	0.87	0.87	0.87

Table 3.5: Classification metrics for the ResNet using data glove.

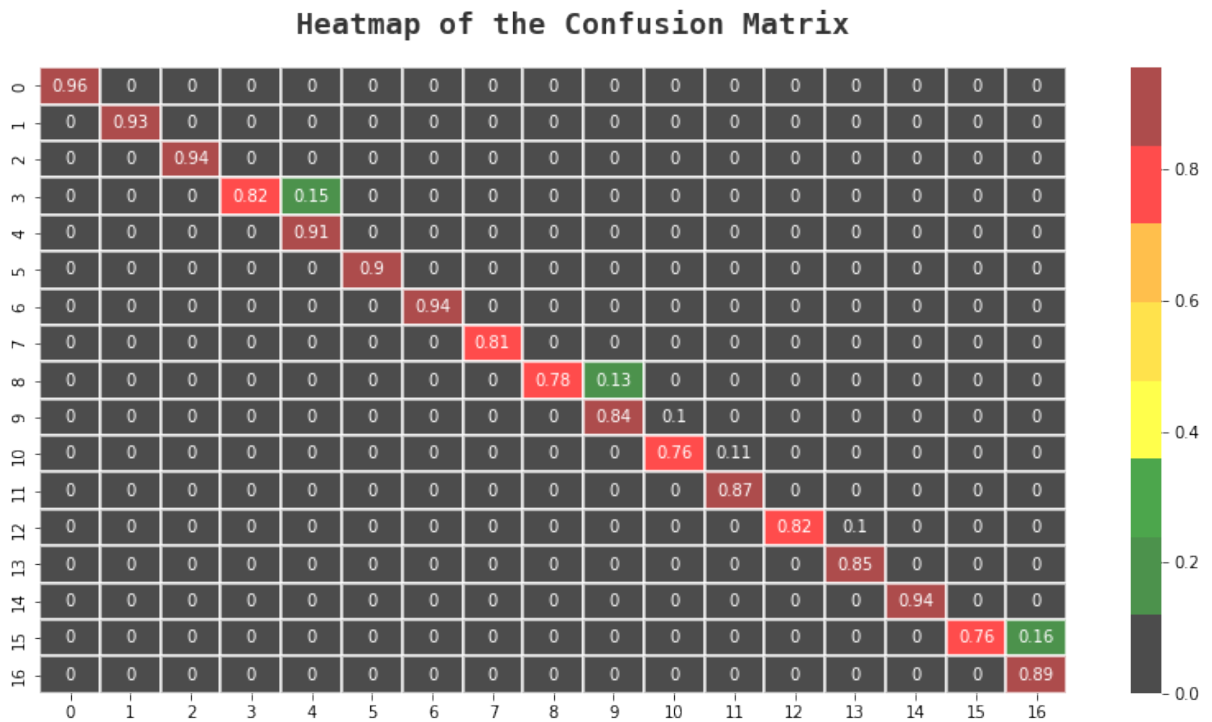


Figure 3.9: Confusion matrix of the ResNet. Note that a 0.1 threshold is applied: all the values below that number are set to 0.

		precision	recall	f1-score
Gesture	1	0.98	0.96	0.97
	2	0.93	0.92	0.93
	3	0.93	0.94	0.93
	4	0.86	0.83	0.84
	5	0.84	0.91	0.87
	6	0.92	0.90	0.91
	7	0.94	0.93	0.94
	8	0.92	0.93	0.92
	accuracy			0.92
	macro average	0.91	0.92	0.91
	weighted average	0.92	0.92	0.92

Table 3.6: Classification metrics for the ResNet using data glove and the reduced gestures set.

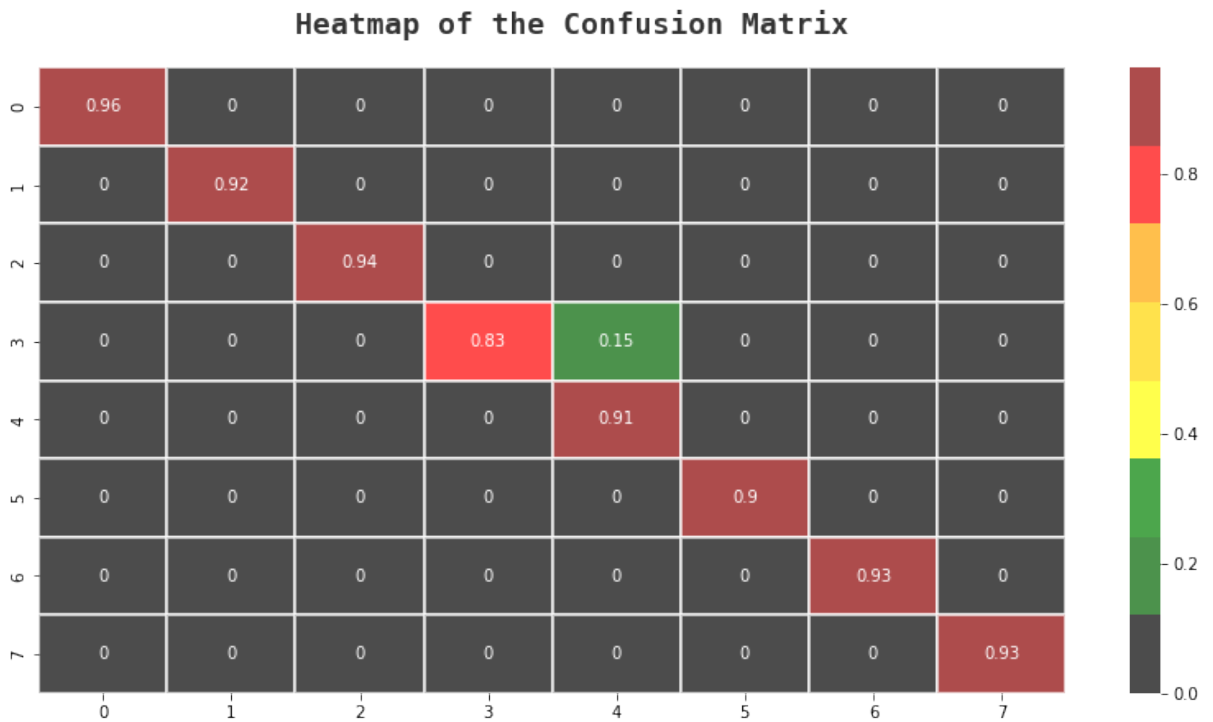


Figure 3.10: Caption

3.1.4 Classical machine learning approaches

The previous networks were compared with three conventional classifiers: K Nearest Neighbors (KNN), Decision Tree (DT) and Support Vector Machine (SVM). This investigation was done in order to highlight the differences between deep learning and classical machine learning approaches along with their potentiality and their ability to interpret and execute correctly the task by the analysis of input data. A first difference refers to the kind of data provided in input to conventional machine learning models. In the case of DL classifiers, row data were directly provided in input as matrices having size: [samples x channels]. Classical ML approaches, instead, take in input features, previously extracted from the raw data. For example, the Root Mean Square (RMS) feature was extracted from each channel for the considered time window. Therefore after this computation, input data were shrunk along the row dimension, being now a vector with number of elements corresponding to the number of channels. The total number of RMS features were 22: one for each channel.

In the following, the results obtained using these approaches are illustrated.

KNN

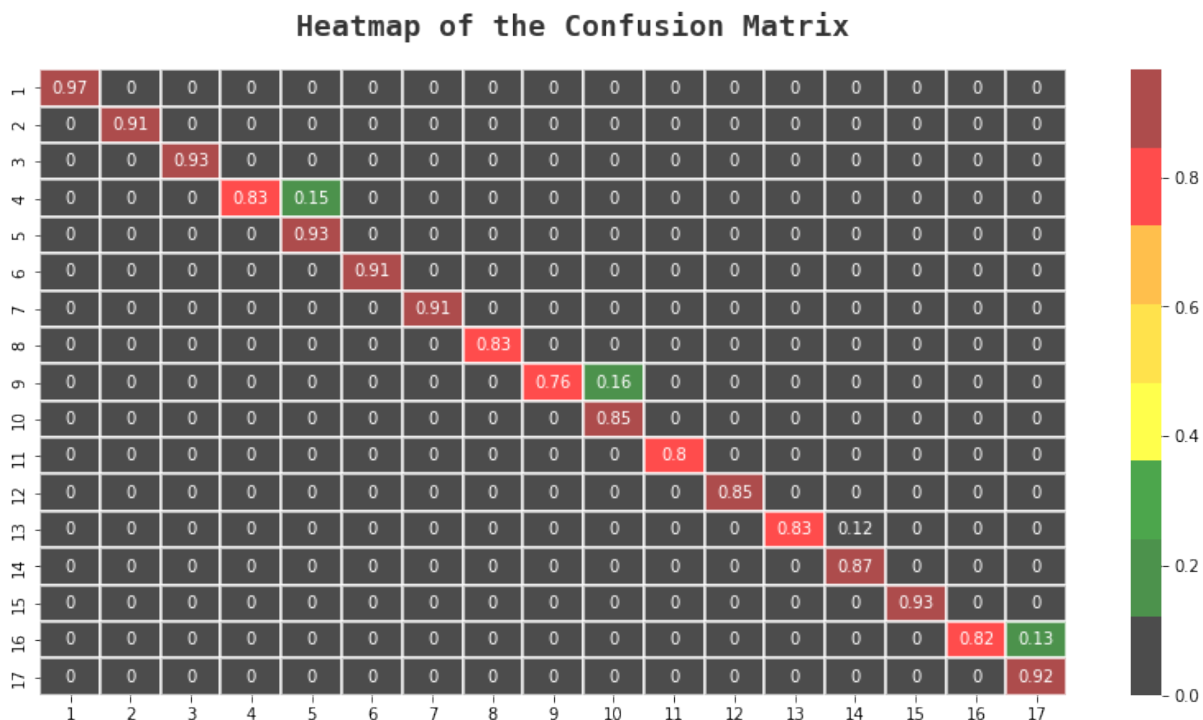
First of all, with the CV tuning procedure, a neighbors value of 1 and the Manhattan distance resulted as the best tuned hyperparameters (see Tab. 3.7). Then, the classifier was trained using the training set, over which it reached an accuracy of 100%. The classifier was evaluated over the test set, on which it was 87% accurate. To be compliant with the deep learning analysis, the same classification metrics were computed and reported in the Tab. 3.8.

	k = 1	k = 2	k = 3	k = 4
Manhattan dist.	0.962	0.952	0.949	0.944
Euclidean dist.	0.959	0.946	0.944	0.937

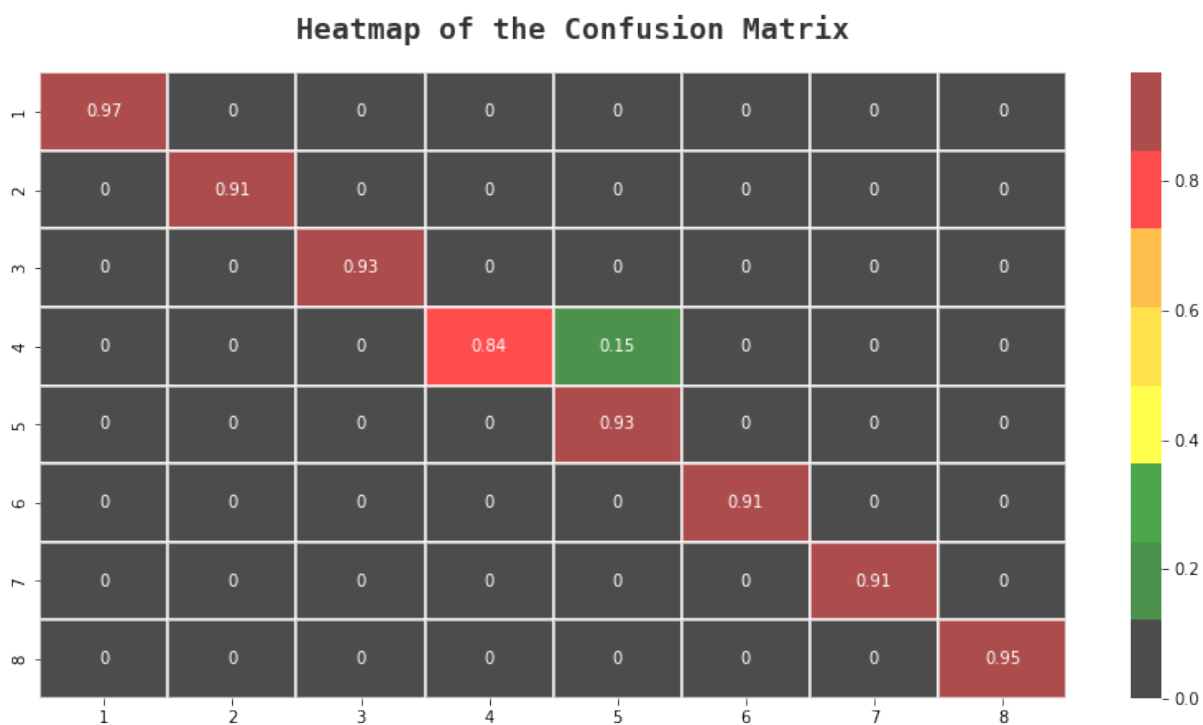
Table 3.7: Average accuracy on the validation set after the CV procedure for each combination of distance and neighbors.

The performances of the classifier were finally assessed by the computation of the confusion matrix over the test set. The confusion matrix has the true classes as rows and the predicted classes as columns (Fig. 3.11a).

As previously done for deep learning approaches, the performances of the model were tested also on a reduced gesture set, where only the first eight gestures were considered. In this case, once trained on the training reduced gesture set, the model exhibited an accuracy of 92% on the test set. The corresponding confusion matrix is reported in Fig. 3.11b.



(a)



(b)

Figure 3.11: (a) Confusion matrix of the KNN classifier evaluated on the entire test set, using data glove. (b) Confusion matrix of the KNN classifier trained and tested on the reduced gestures set, using data glove.

		precision	recall	f1-score
Gesture	1	0.98	0.97	0.97
	2	0.95	0.91	0.93
	3	0.92	0.93	0.92
	4	0.88	0.83	0.86
	5	0.85	0.93	0.89
	6	0.93	0.91	0.92
	7	0.93	0.91	0.92
	8	0.86	0.83	0.84
	9	0.83	0.76	0.79
	10	0.79	0.85	0.82
	11	0.81	0.80	0.81
	12	0.81	0.85	0.83
	13	0.80	0.83	0.81
	14	0.83	0.87	0.85
	15	0.93	0.93	0.93
	16	0.86	0.82	0.84
	17	0.84	0.92	0.88
	accuracy			0.87
	macro average	0.87	0.87	0.87
	weighted average	0.88	0.87	0.87

Table 3.8: Classification metrics for the knn with data glove.

Decision Tree

A CV-based hyperparameters tuning was performed, investigating the best combination between the maximum depth of the tree and the splitting criterion. As Tab. 3.9 shows, the procedure suggested that there is no substantial difference in adopting a maximum depth of 30 or of 40. A maximum depth of 30 was chosen, which allowed to improve the computational cost. Furthermore, the entropy criterion was selected as splitting criterion since it allowed to reach slightly better performances.

	10	20	30	40
gini index	0.530	0.791	0.828	0.827
entropy	0.572	0.829	0.834	0.835

Table 3.9: Average accuracy on the validation set after the CV procedure for each combination of maximum depth (columns) and splitting criterion (rows).

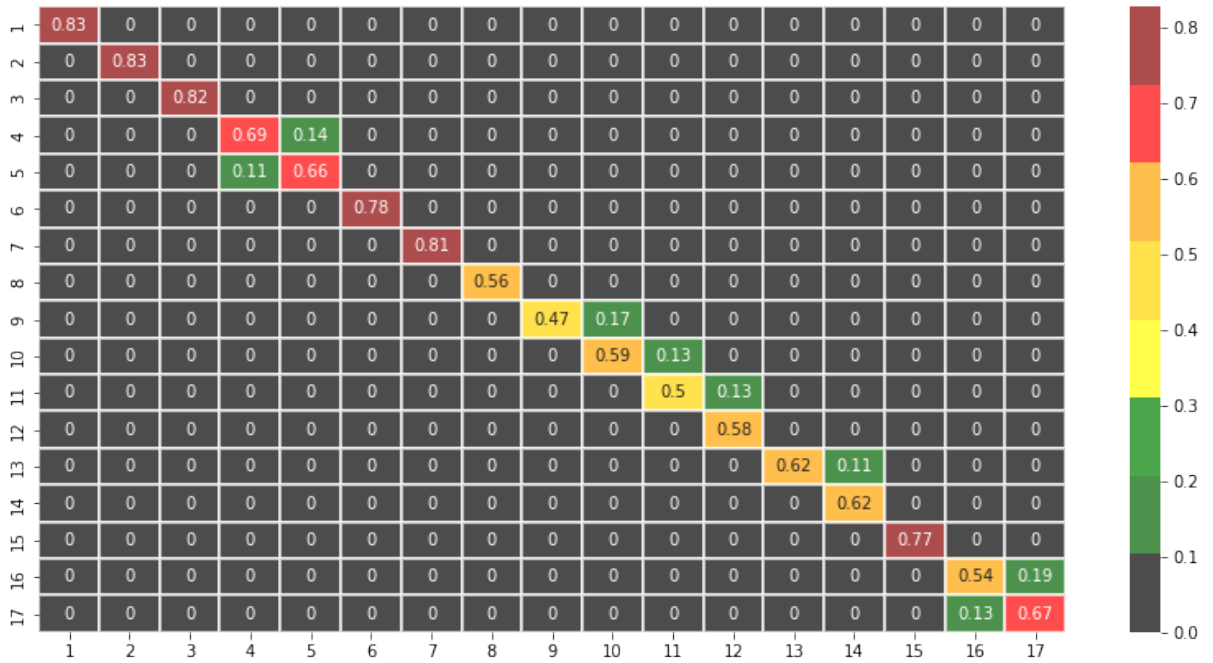
The model was built by choosing these settings and then it was trained over the entire training set, thus obtaining 100% of accuracy over it. The accuracy over the test set was of the 67%. Classification metrics were finally computed: precision, recall and f1-score were evaluated for each gesture (Tab. 3.10). The confusion matrix, evaluated on the test set, is reported in (Fig. 3.12a).

The same model was evaluated on the diminished gestures set; the percentage of accuracy is expected to be higher because of the simplified problem. Indeed, it reached an accuracy on the test set of 81%. The resulting confusion matrix is reported in (Fig. 3.12b).

		precision	recall	f1-score
Gesture	1	0.88	0.83	0.85
	2	0.85	0.83	0.84
	3	0.81	0.82	0.82
	4	0.66	0.69	0.68
	5	0.67	0.66	0.67
	6	0.75	0.78	0.77
	7	0.86	0.81	0.83
	8	0.57	0.56	0.56
	9	0.53	0.47	0.50
	10	0.60	0.59	0.59
	11	0.52	0.50	0.51
	12	0.51	0.58	0.54
	13	0.60	0.62	0.61
	14	0.63	0.62	0.62
	15	0.75	0.77	0.72
	16	0.54	0.54	0.54
	17	0.59	0.67	0.63
	accuracy			0.67
	macro avg	0.66	0.67	0.67
	weighted avg	0.68	0.67	0.67

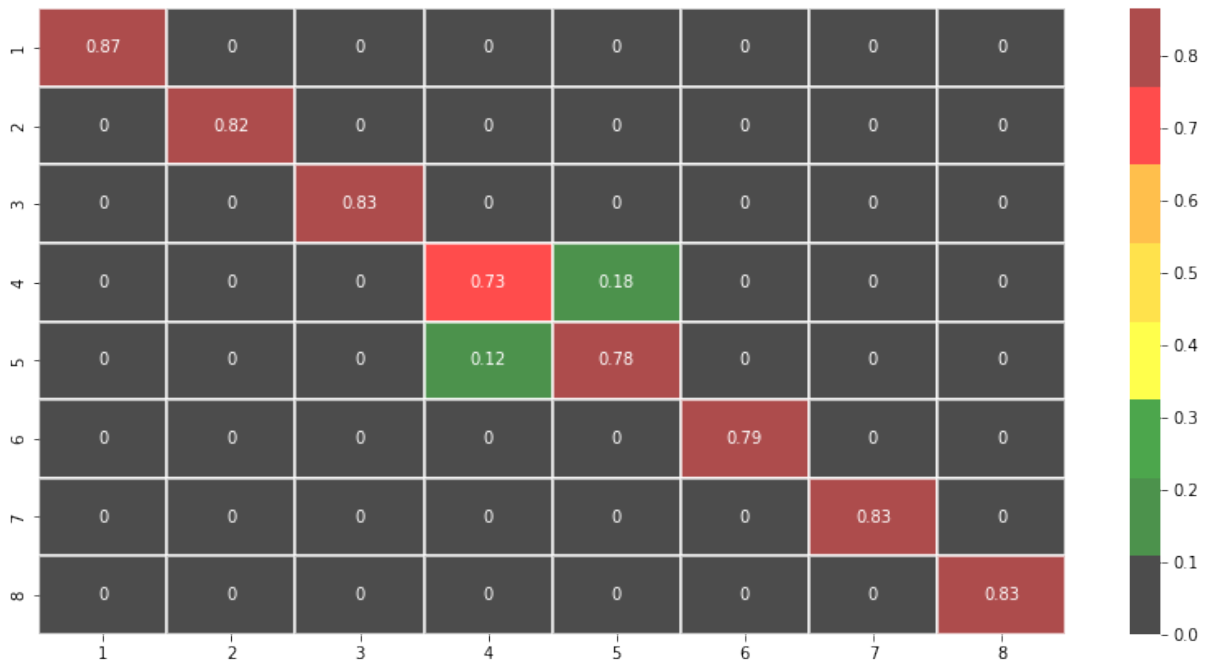
Table 3.10: Classification metrics of the decision tree with data glove.

Heatmap of the Confusion Matrix



(a)

Heatmap of the Confusion Matrix



(b)

Figure 3.12: (a) Confusion matrix of the DT classifier evaluated on the entire test set, using data glove. (b) Confusion matrix of the DT classifier trained and tested on the reduced gestures set, using data glove.

Support Vector Machine

For this classifier, the tuned hyperparameters were the type of kernel (linear, polynomial, radial basis function or sigmoidal) and the regularization parameter C . Moreover, during the radial basis function kernel evaluation, the gamma parameter was tuned too. The procedure was slightly different this time, because of the computational complexity of the model training that needed a considerable amount of time. For this reason, the CV procedure was avoided, and for each hyperparameters combination, the model was trained on the same training set and tested on the same validation set (referring to the section 2.1.4). Results suggested to use a radial basis function, with the regularization parameter equal to 10 and a gamma value of 0.1 (see Tab. 3.11).

Thereafter, the model based on these optimal parameters were trained over the complete training set, as done for the previous configurations, and tested over the test set. With a 98% of accuracy over the training set, it achieved an accuracy of 87% over the test set. For each gesture, the classification metrics were finally evaluated (Tab. 3.12). The confusion matrix is reported in Fig. 3.13a.

Even for this classifier the reduced gesture set was considered. The model accuracy on the test set went up to 93%, and its confusion matrix is displayed in Fig. 3.13b.

<i>linear</i>	C = 0.01	0.529
	C = 0.1	0.539
	C = 1	0.539
	C = 10	0.540
<i>polynomial</i>	C = 0.01	0.426
	C = 0.1	0.577
	C = 1	0.741
	C = 10	0.829
<i>sigmoid</i>	C = 0.01	0.392
	C = 0.1	0.281
	C = 1	0.242
	C = 10	0.236

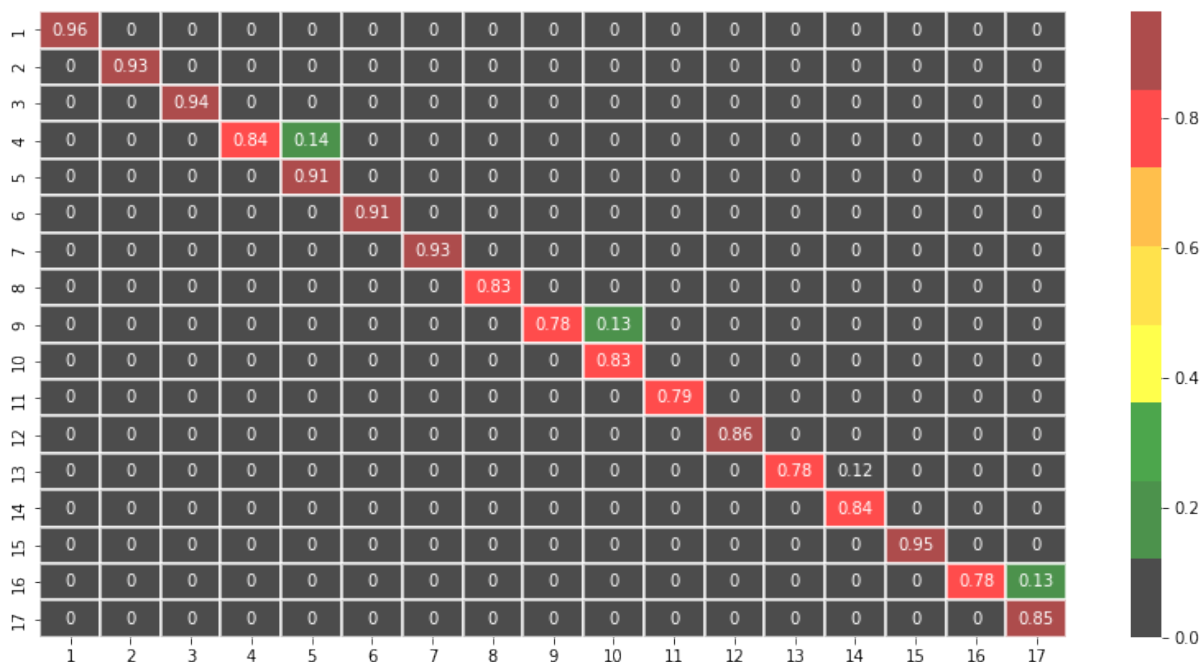
		$\gamma = 0.001$	0.01	0.1	1
<i>rbf</i>	C = 0.01	0.163	0.440	0.516	0.078
	C = 0.1	0.454	0.531	0.698	0.530
	C = 1	0.515	0.609	0.842	0.838
	C = 10	0.555	0.723	0.890	0.844

Table 3.11: Average accuracy on the validation set after the CV procedure for each combination of regularization parameter C (rows) and γ (columns).

		precision	recall	f1-score
Gesture	1	0.98	0.96	0.97
	2	0.93	0.93	0.93
	3	0.93	0.94	0.94
	4	0.83	0.84	0.84
	5	0.84	0.91	0.87
	6	0.93	0.91	0.92
	7	0.95	0.93	0.94
	8	0.88	0.83	0.85
	9	0.81	0.78	0.80
	10	0.80	0.83	0.82
	11	0.81	0.79	0.80
	12	0.79	0.86	0.82
	13	0.82	0.78	0.80
	14	0.82	0.84	0.83
	15	0.94	0.95	0.95
	16	0.84	0.78	0.81
	17	0.84	0.85	0.84
	accuracy			0.87
	macro avg	0.87	0.87	0.87
	weighted avg	0.87	0.87	0.87

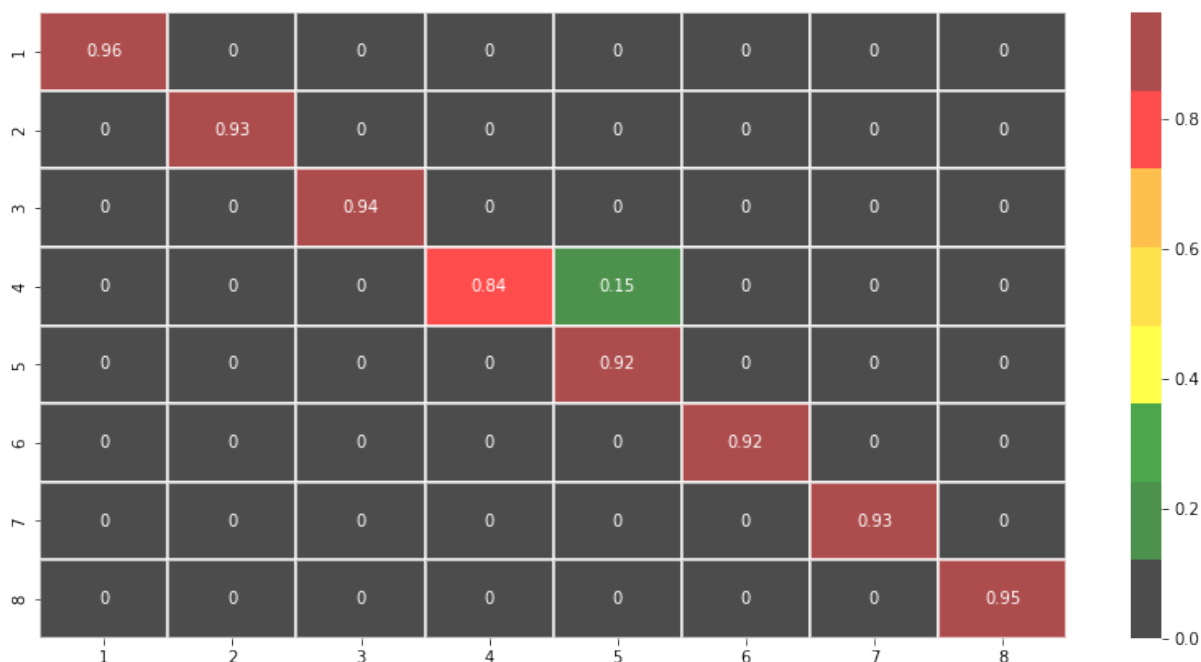
Table 3.12: Classification metrics of the svm with data glove.

Heatmap of the Confusion Matrix



(a)

Heatmap of the Confusion Matrix



(b)

Figure 3.13: (a) Confusion matrix of the SVM classifier evaluated on the entire test set, using data glove. (b) Confusion matrix of the SVM classifier trained and tested on the reduced gestures set, using data glove.

3.2 sEMG

Hand gesture classification based entirely on sEMG was performed exploring both machine learning approaches and deep learning approaches. As described in details in the methods chapter, the analyzed models were the following:

- **Convolutional Neural Network** with two different configurations. In the first one, data were organized in an array having elements corresponding to the whole features extracted (one for each channel). Two architectures were explored. The first network architecture had three convolutional blocks, with an increasing number of convolutional filters (32, 64, 128) and increasing dimensionality (23x1), (46x1), (92x1). This structure worked with all the extracted features, belonging to the time and frequency domains. When only time domain features were considered, instead, the architecture was identical in all its aspects but the dimension of filters, which reduced to (13x1), (26x1) and (52x1).

In the second configuration, data were structured as 2D matrices. The architecture of the network consisted of four convolutional blocks with a growing number of convolutional filters (32, 64, 128, 256). When both time and frequency domains were used, the filters had respectively the dimensions of (2x2), (2x2), (3x3) and (4x4); on the other hand, when just time domain features were examined, they were, in order, (2x2), (3x3), (2x2) and (2x2).

- **Random Forest**, using 300 estimators and working with data organized as an array.

3.2.1 CNN

For each of the previously mentioned configurations, the robustness of the network results was assessed by performing a Cross Validation procedure. All the networks demonstrated to be quite robust in relation to a slight change of the training set (Tab. 3.13a). Then, each model was retrained using the entire training set and evaluated over the test set, with the computation of classification metrics as *precision*, *recall* and *f1-score* and *accuracy*. Finally, the same network architectures were tested on a reduced gestures set, as previously described for the data glove. Indeed, the CV was done to assess the average accuracies of the network under this pruned condition (Tab. 3.13b); both the classification metrics and the confusion matrix were evaluated on the test set. Here below, the obtained results are presented.

- **Time and frequency features - 1D**. For what concerns the complete gestures set, with 30 epochs, it reached a final training accuracy of 75% spending less than 7 minutes.

Accuracy	All features - 1D	Time features - 1D	All features 2D	Time features - 2D
Training	0.82	0.75	0.67	0.68
	0.72	0.78	0.65	0.65
	0.82	0.76	0.68	0.66
	0.80	0.79	0.70	0.64
	0.75	0.77	0.69	0.64
Average	0.78	0.77	0.68	0.65
Test	0.75	0.73	0.62	0.65
	0.69	0.74	0.61	0.65
	0.70	0.73	0.61	0.65
	0.70	0.74	0.62	0.65
	0.68	0.73	0.63	0.65
Average	0.69	0.73	0.62	0.65

(a)

Accuracy	All features - 1D	Time features - 1D	All features 2D	Time features - 2D
Training	0.88	0.85	0.74	0.72
	0.85	0.78	0.76	0.72
	0.87	0.83	0.76	0.74
	0.86	0.83	0.72	0.72
	0.85	0.84	0.77	0.73
Average	0.86	0.83	0.75	0.73
Test	0.75	0.81	0.68	0.73
	0.75	0.80	0.70	0.72
	0.76	0.80	0.70	0.74
	0.76	0.80	0.70	0.73
	0.74	0.81	0.70	0.73
Average	0.75	0.80	0.70	0.73

(b)

Table 3.13: k-th fold accuracy and average accuracy of the Cross Validation procedure for the k-th training set and k-th test set, (a) entire gesture set and (b) reduced gesture set.

The classification metrics can be analyzed in Tab. 3.14. The confusion matrix was also evaluated and reported in Fig. 3.14.

Regarding the reduced gestures set, it took less than 4 minutes to complete the training with an accuracy of 86%. The performances over the test set are summarized in Tab. 3.15 and the confusion matrix is showed in Fig. 3.15 .

- **Only time features - 1D.** Treating the complete gesture set, the network trained in less than 5 minutes, and after 30 epochs it reached an accuracy of 79%. Classification metrics and confusion matrix were evaluated in Tab. 3.16 and Fig. 3.16, respectively.

On the other hand, when analyzing the selected first 8 gestures, the network completed the training after less than 3 minutes, with an accuracy of 84%. Classification metrics and confusion matrix are reported in Tab. 3.17 and Fig. 3.17.

- **Time and frequency features - 2D.** In this case, when dealing with the complete gesture set, the network took 6 minutes to complete the training, at the end of which it was 68% accurate over the training set. When analyzing the restricted gesture set instead, the learning procedure spent slightly more than 3 minutes and it ended with a 73% of accuracy.

In both cases, the performances of the networks were evaluated over the test set by the computation of the classification metrics and the confusion matrix (Tab. 3.18, Fig. 3.18, Tab. 3.19, Fig. 3.19).

- **Only time features - 2D.** This configuration spent less than 5 minutes to complete the training through 30 epochs and with 67% training accuracy, when analysing the complete gesture set. The training time diminished when the analysis focused only on the first 8 gestures; the accuracy augmented, because in less than 3 minutes it reached an accuracy of 73%.

The classification metrics and the confusion matrix were then evaluated over the test set (Tab. 3.20, Fig. 3.20, Tab. 3.21, Fig. 3.21).

		precision	recall	f1-score
Gesture	1	0.78	0.77	0.77
	2	0.68	0.71	0.70
	3	0.68	0.67	0.68
	4	0.64	0.56	0.60
	5	0.57	0.56	0.57
	6	0.66	0.69	0.68
	7	0.71	0.65	0.68
	8	0.54	0.59	0.56
	9	0.67	0.64	0.66
	10	0.67	0.65	0.66
	11	0.51	0.55	0.53
	12	0.54	0.52	0.53
	13	0.63	0.64	0.63
	14	0.64	0.65	0.65
	15	0.69	0.73	0.71
	16	0.57	0.54	0.56
	17	0.59	0.68	0.63
	accuracy			0.64
	macro avg	0.63	0.64	0.63
	weighted avg	0.64	0.64	0.64

Table 3.14: Classification metrics for the CNN working with all the features, using the 1D configuration with the complete gestures set.

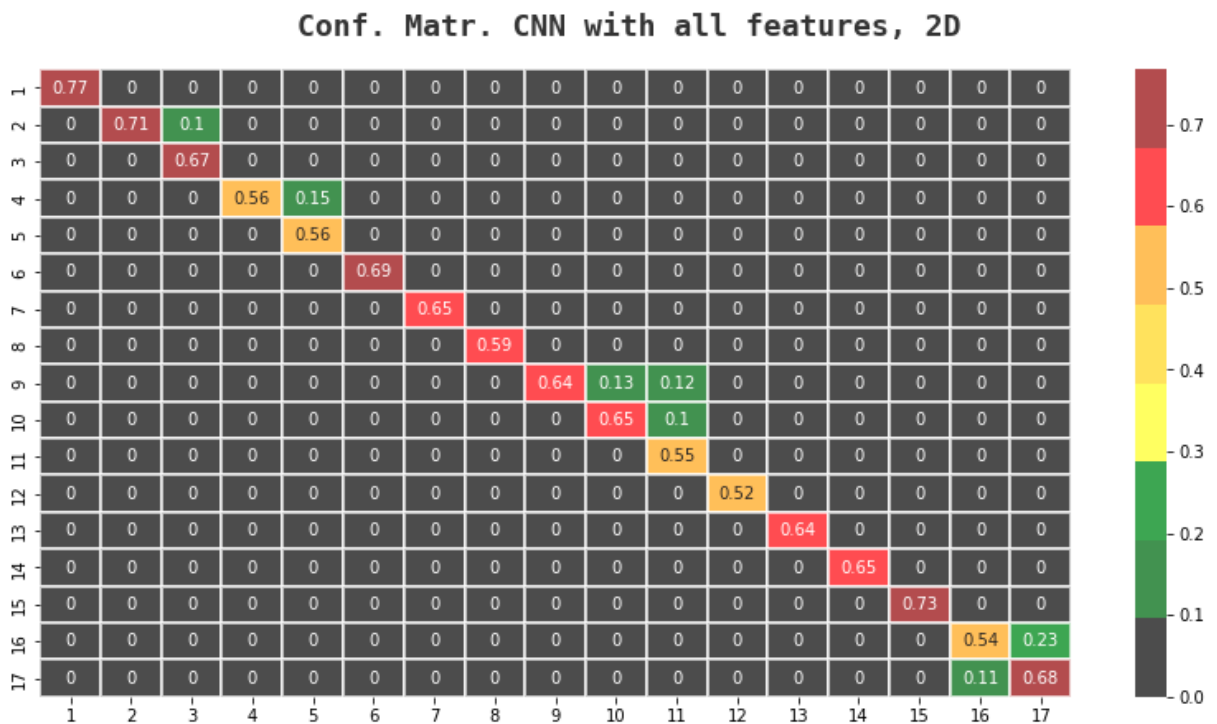


Figure 3.14: Confusion matrix for the CNN working with all the features, using the 1D configuration with the complete gestures set.

		precision	recall	f1-score
Gesture	1	0.83	0.78	0.81
	2	0.71	0.70	0.70
	3	0.71	0.70	0.70
	4	0.70	0.60	0.64
	5	0.59	0.66	0.62
	6	0.70	0.72	0.71
	7	0.71	0.75	0.73
	8	0.65	0.69	0.67
	accuracy			0.70
	macro avg	0.70	0.70	0.70
	weighted avg	0.71	0.70	0.70

Table 3.15: Classification metrics for the CNN working with all the features, using the 1D configuration with the reduced gestures set.

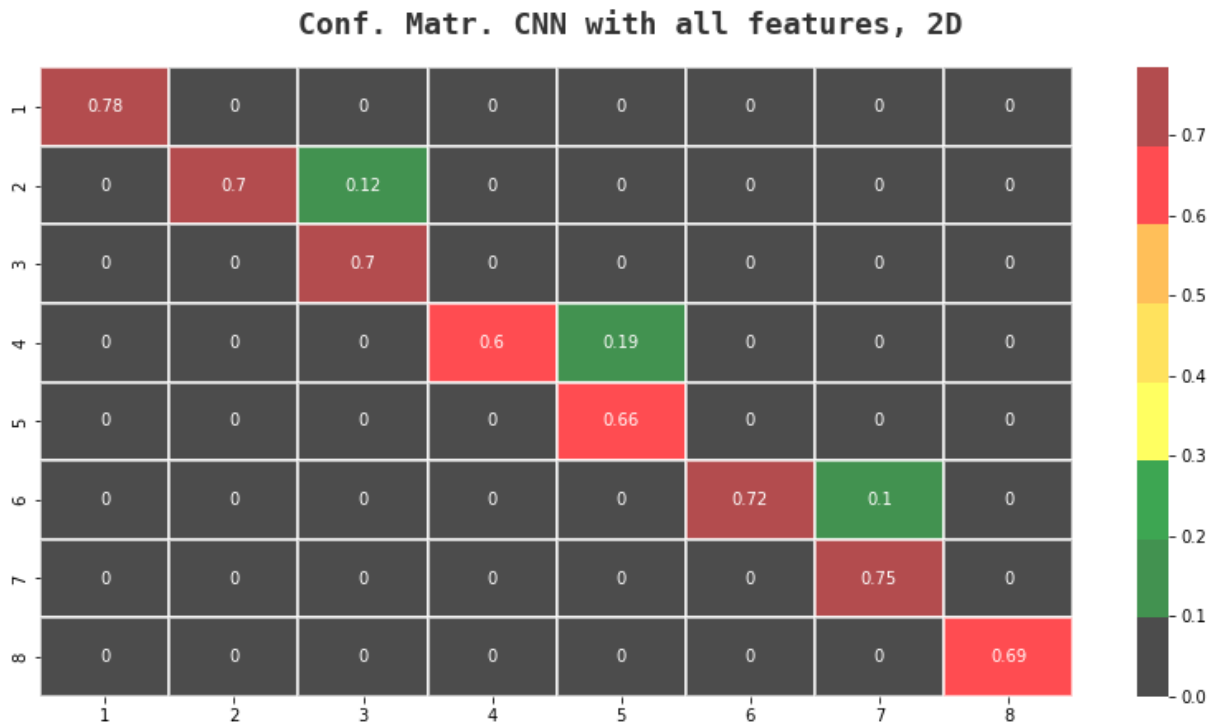


Figure 3.15: Confusion matrix for the CNN working with all the features, using the 1D configuration with the reduced gestures set.

		precision	recall	f1-score
Gesture	1	0.82	0.79	0.80
	2	0.69	0.75	0.72
	3	0.75	0.70	0.72
	4	0.65	0.62	0.63
	5	0.57	0.62	0.59
	6	0.77	0.68	0.72
	7	0.68	0.71	0.69
	8	0.57	0.61	0.59
	9	0.68	0.67	0.67
	10	0.71	0.64	0.67
	11	0.58	0.56	0.57
	12	0.54	0.55	0.55
	13	0.65	0.69	0.67
	14	0.68	0.67	0.68
	15	0.70	0.77	0.73
	16	0.64	0.62	0.63
	17	0.66	0.65	0.66
	accuracy			0.67
	macro avg	0.67	0.66	0.67
	weighted avg	0.67	0.67	0.67

Table 3.16: Classification metrics for the CNN working with only the time features, using the 1D configuration with the complete gestures set.

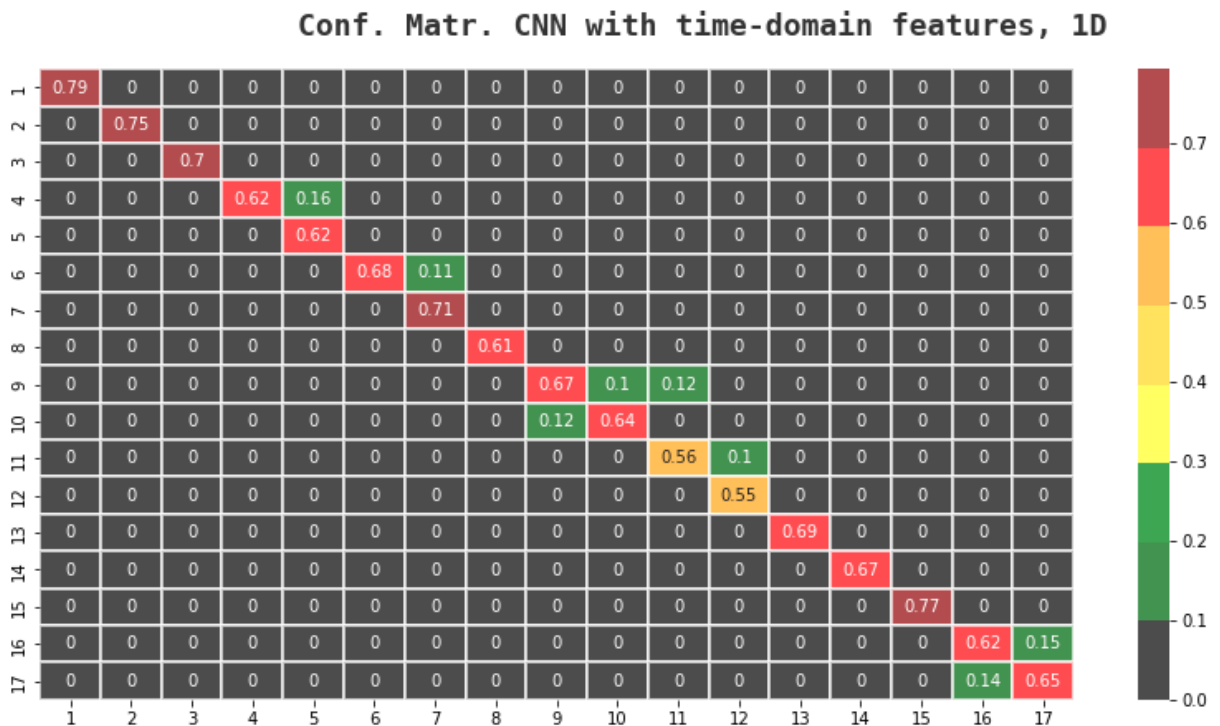


Figure 3.16: Confusion matrix for the CNN working with only the time features, using the 1D configuration with the complete gestures set.

		precision	recall	f1-score
Gesture	1	0.82	0.85	0.84
	2	0.77	0.70	0.74
	3	0.71	0.74	0.73
	4	0.74	0.61	0.67
	5	0.60	0.73	0.66
	6	0.74	0.76	0.75
	7	0.75	0.74	0.75
	8	0.71	0.69	0.70
	accuracy			0.73
	macro avg	0.73	0.73	0.73
	weighted avg	0.74	0.73	0.73

Table 3.17: Classification metrics for the CNN working with only the time features, using the 1D configuration with the reduced gestures set.

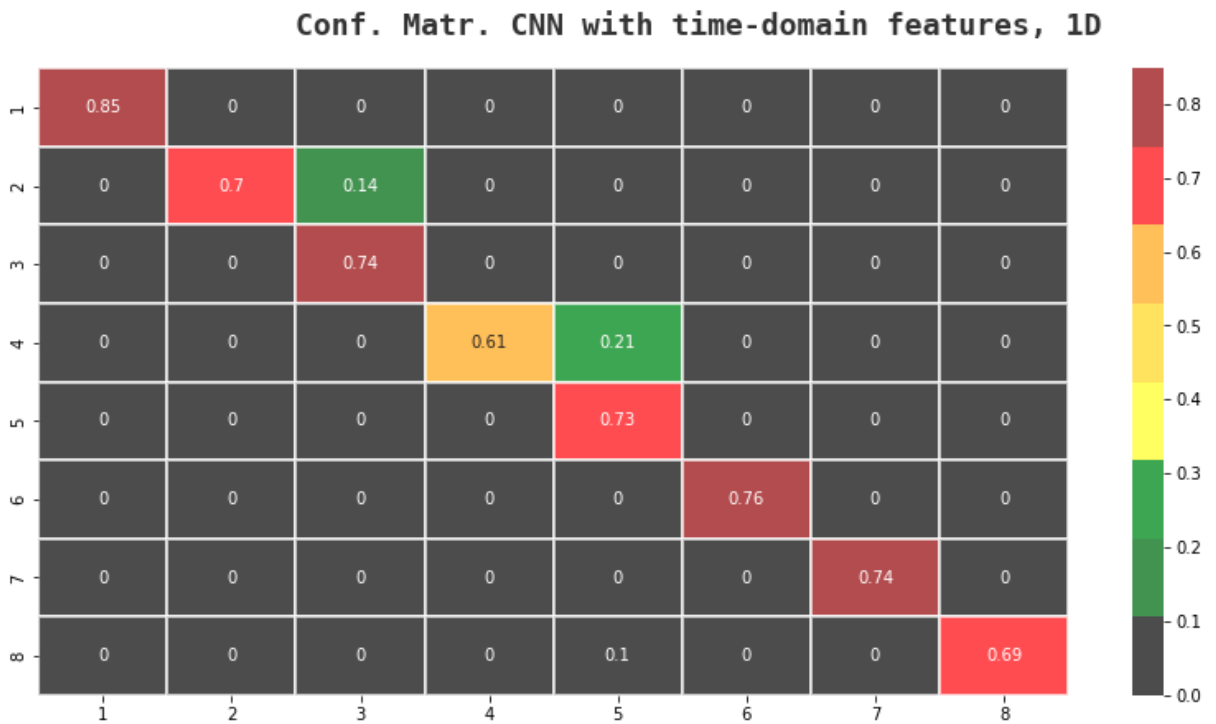


Figure 3.17: Confusion matrix for the CNN working with only the time features, using the 1D configuration with the reduced gestures set.

		precision	recall	f1-score
Gesture	1	0.71	0.75	0.73
	2	0.61	0.65	0.63
	3	0.65	0.67	0.66
	4	0.56	0.54	0.55
	5	0.50	0.52	0.51
	6	0.66	0.62	0.64
	7	0.68	0.64	0.66
	8	0.53	0.52	0.53
	9	0.69	0.54	0.60
	10	0.61	0.62	0.61
	11	0.49	0.49	0.49
	12	0.47	0.46	0.47
	13	0.59	0.60	0.59
	14	0.61	0.62	0.61
	15	0.65	0.72	0.69
	16	0.53	0.52	0.52
	17	0.58	0.63	0.60
	accuracy			0.60
	macro avg	0.59	0.59	0.59
	weighted avg	0.60	0.60	0.60

Table 3.18: Classification metrics for the CNN working with all the features, using the 2D configuration with the complete gestures set and.

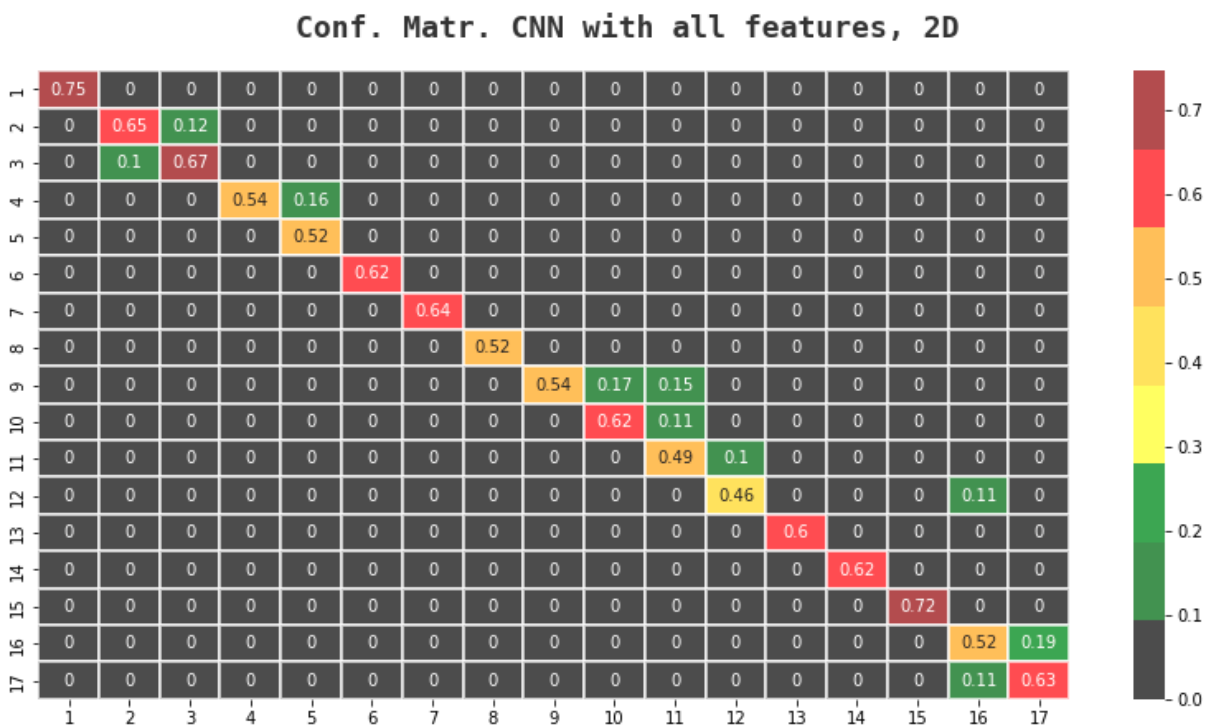


Figure 3.18: Confusion matrix for the CNN working with all the features, using the 2D configuration with the complete gestures set.

		precision	recall	f1-score
Gesture	1	0.76	0.79	0.77
	2	0.66	0.67	0.67
	3	0.70	0.66	0.68
	4	0.65	0.53	0.58
	5	0.55	0.62	0.59
	6	0.66	0.69	0.68
	7	0.69	0.68	0.69
	8	0.64	0.64	0.64
	accuracy			0.67
	macro avg	0.66	0.66	0.66
	weighted avg	0.67	0.67	0.67

Table 3.19: Classification metrics for the CNN working with all the features, using the 2D configuration with the reduced gestures set.

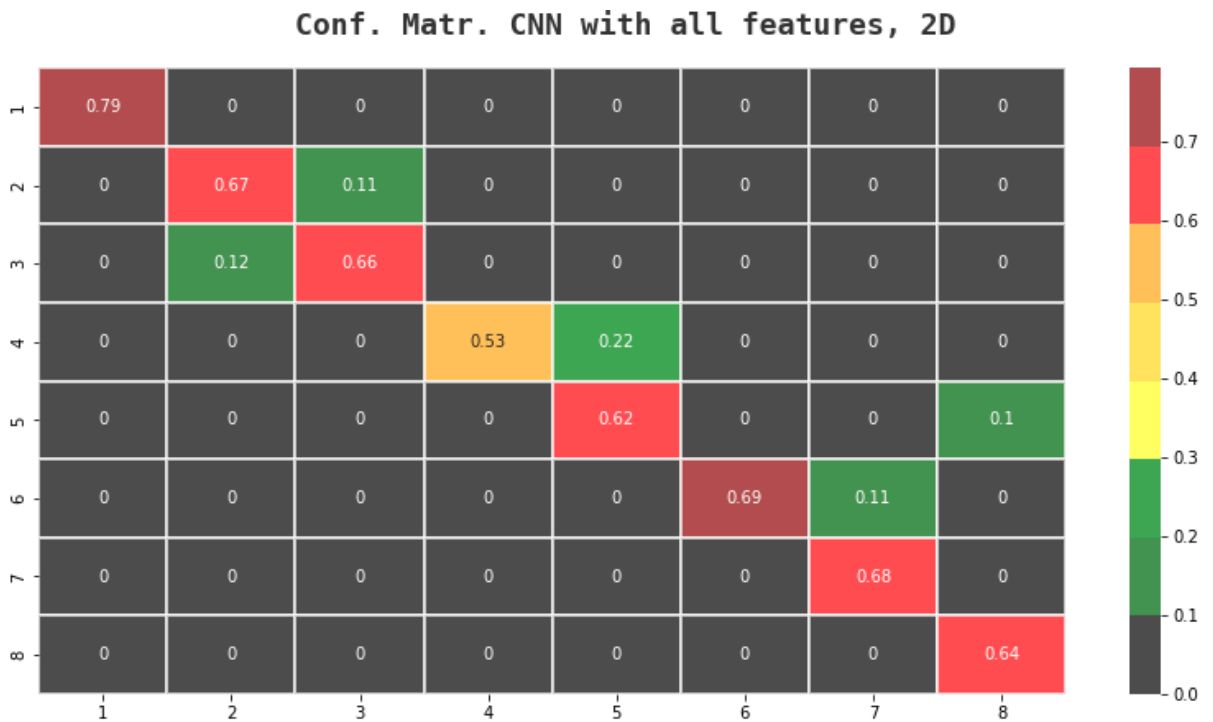


Figure 3.19: Confusion matrix for the CNN working with all the features, using the 2D configuration with the reduced gestures set and.

		precision	recall	f1-score
Gesture	1	0.71	0.78	0.74
	2	0.65	0.64	0.65
	3	0.65	0.65	0.65
	4	0.57	0.52	0.54
	5	0.49	0.59	0.54
	6	0.70	0.62	0.66
	7	0.62	0.67	0.64
	8	0.53	0.54	0.53
	9	0.65	0.62	0.64
	10	0.64	0.58	0.61
	11	0.51	0.48	0.49
	12	0.49	0.50	0.49
	13	0.69	0.60	0.64
	14	0.59	0.64	0.62
	15	0.68	0.70	0.69
	16	0.60	0.49	0.54
	17	0.56	0.66	0.61
	accuracy			0.61
	macro avg	0.61	0.60	0.60
	weighted avg	0.61	0.61	0.61

Table 3.20: Classification metrics for the CNN working with only time features, using the 2D configuration with the complete gestures set.

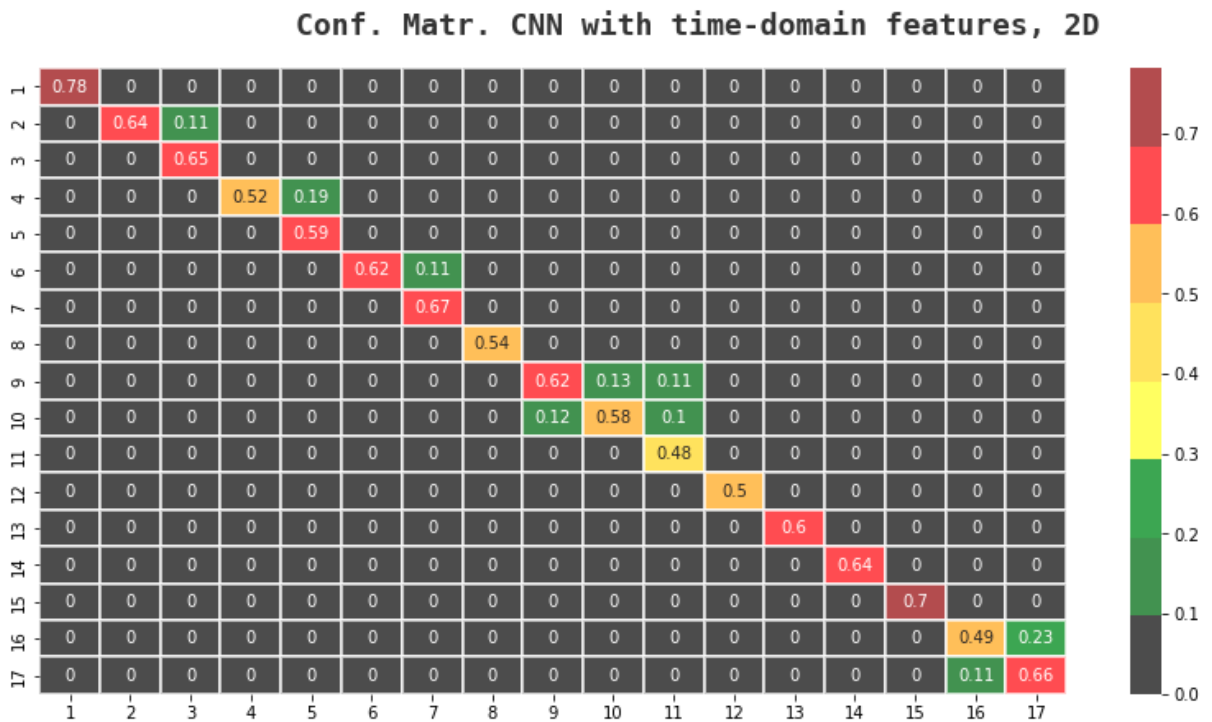


Figure 3.20: Confusion matrix for the CNN working with only time features, using the 2D configuration with the complete gestures set.

		precision	recall	f1-score
Gesture	1	0.80	0.79	0.80
	2	0.68	0.67	0.67
	3	0.68	0.71	0.69
	4	0.73	0.54	0.62
	5	0.58	0.67	0.63
	6	0.72	0.72	0.72
	7	0.67	0.72	0.70
	8	0.68	0.69	0.68
	accuracy			0.69
	macro avg	0.69	0.69	0.69
	weighted avg	0.70	0.69	0.69

Table 3.21: Classification metrics for the CNN working with only time features, using the 2D configuration with the reduced gestures set.

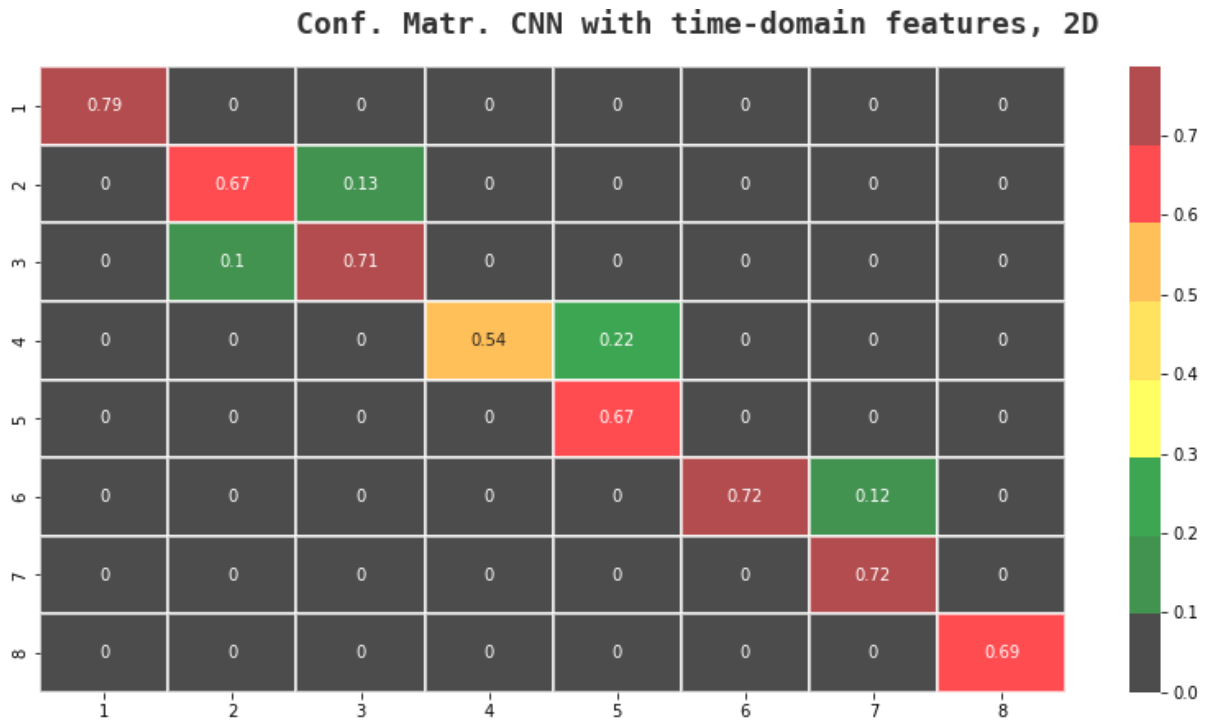


Figure 3.21: Confusion matrix for the CNN working with only time features, using the 2D configuration with the reduced gestures set.

3.2.2 RF

The hyperparameters tuning led to the use of the entropy criterion and to consider 2 minimum samples per leaf, as the results in Tab. 3.22 suggest.

min. samples x leaf	gini crit	entropy crit
2	0.68	0.69
8	0.64	0.65
12	0.62	0.63

Table 3.22: Validation accuracy for each combination of splitting criterion (columns) and minimum sample per leaf (rows).

Working with all the features, the accuracy of the model over the validation set was of 69%. The function *RandomForestClassifier* from the *Sci-Kit learn* tool, comes with an attribute called *feature_importances* that allowed to evaluate the homonym property; in Fig. 3.22 the hand-crafted features are sorted based on their relative importance (the graph was divided in 4 pieces to allow a clearer visualization).

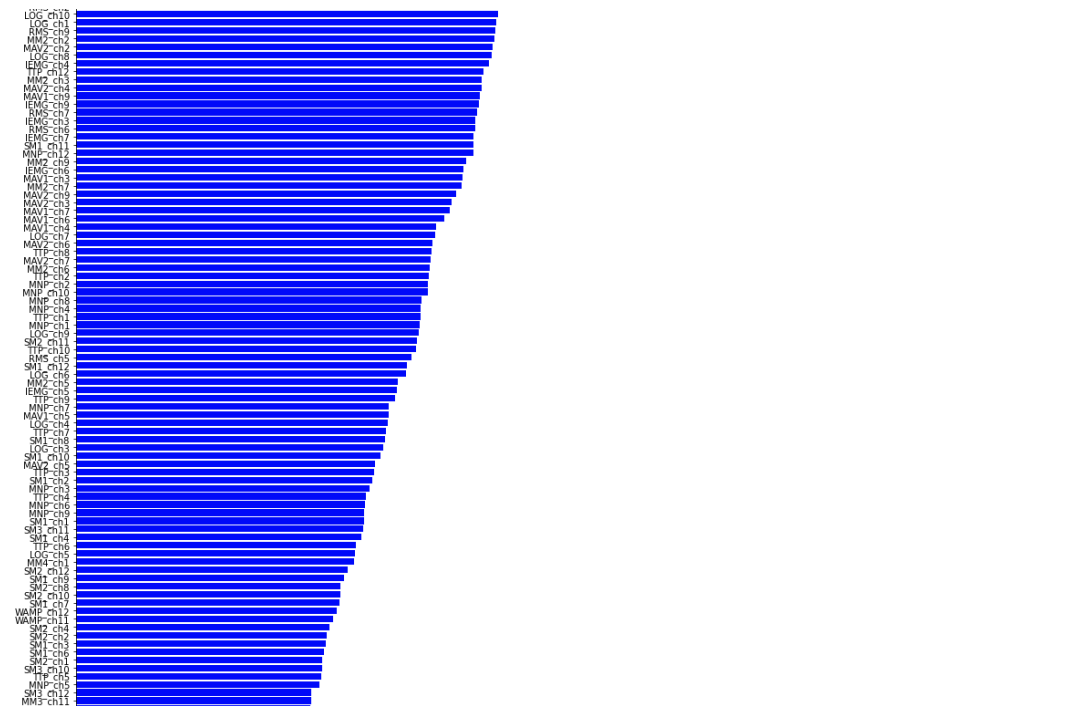
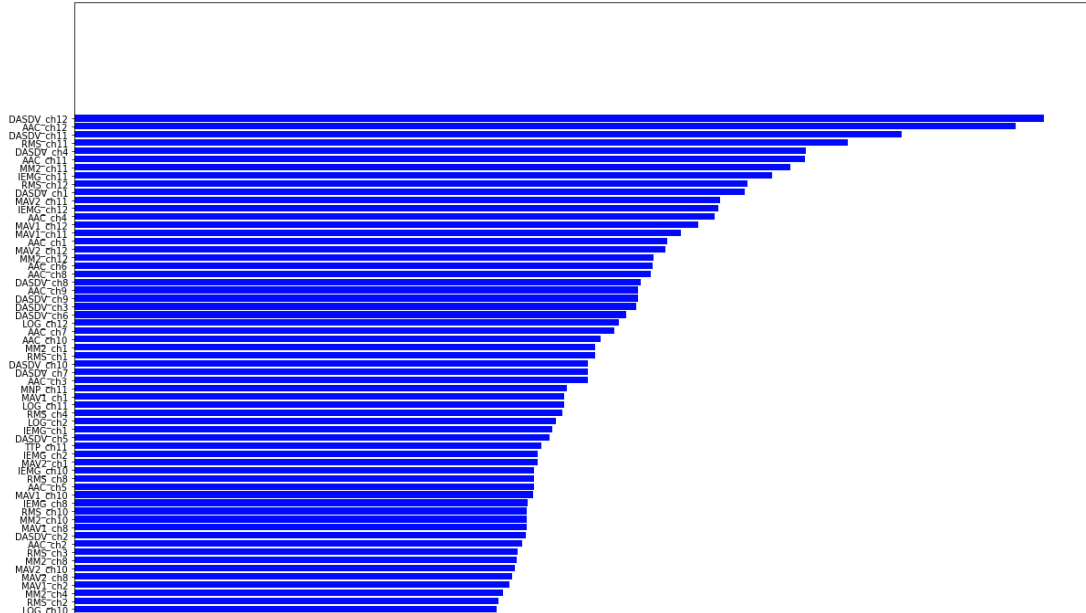
During the feature selection procedure, only features with a relative importance higher than 50-60% were considered. This led to choose up to 7 hand-crafted features (for each channel, resulting in 84 features): RMS (eq. 1.9), IEMG (eq. 1.1), DASDV (eq. 1.12), MAV1 (eq. 1.3), MAV2 (eq. 1.4), MM2 (eq. 1.5), AAC (eq. 1.11). Then, the RF was re-trained considering only the selected features and the performances of the new classifier were evaluated on the validation set. The validation accuracy increased up to 71%. The selected features importance on the new RF is displayed in the Fig. 3.24.

The final RF model was then trained by using only the selected features and by merging the training and validation set. A final accuracy of 100% was obtained over the training set and of 71% on the test set. Moreover, performances were evaluated by the computation of the classification metrics and the confusion matrix, using the test set (Tab. 3.23 and Fig. 3.23).

Accordingly to all the previous analysis, the performances of the same RF model were assessed over a reduced gestures set. In this case, the feature selection procedure was repeated for each subset of gestures (the first from number 1 to number 8, the second all the others): the idea was to understand if distinct properties characterized the two different kinds of gestures, so that they could be better identified by using different features.

Fig. 3.26 represents the relative features importance when working with the first gestures subset. The features that appeared to be fundamental are: RMS (eq. 1.9), IEMG (eq. 1.1), the MM2 (eq. 1.5), DASDV (eq. 1.12), MAV1 (eq. 1.3), and AAC (eq. 1.11) (Fig. 3.25).

Feature Importances



		precision	recall	f1-score
Gesture	1	0.79	0.87	0.83
	2	0.75	0.78	0.76
	3	0.75	0.73	0.74
	4	0.67	0.65	0.66
	5	0.63	0.67	0.65
	6	0.77	0.69	0.73
	7	0.71	0.81	0.75
	8	0.66	0.64	0.65
	9	0.72	0.71	0.72
	10	0.70	0.73	0.72
	11	0.61	0.58	0.59
	12	0.61	0.59	0.60
	13	0.72	0.67	0.70
	14	0.76	0.65	0.70
	15	0.74	0.83	0.78
	16	0.67	0.63	0.65
	17	0.73	0.67	0.70
	accuracy			0.71
	macro avg	0.71	0.70	0.70
	weighted avg	0.71	0.71	0.71

Table 3.23: Classification metrics of the final RF

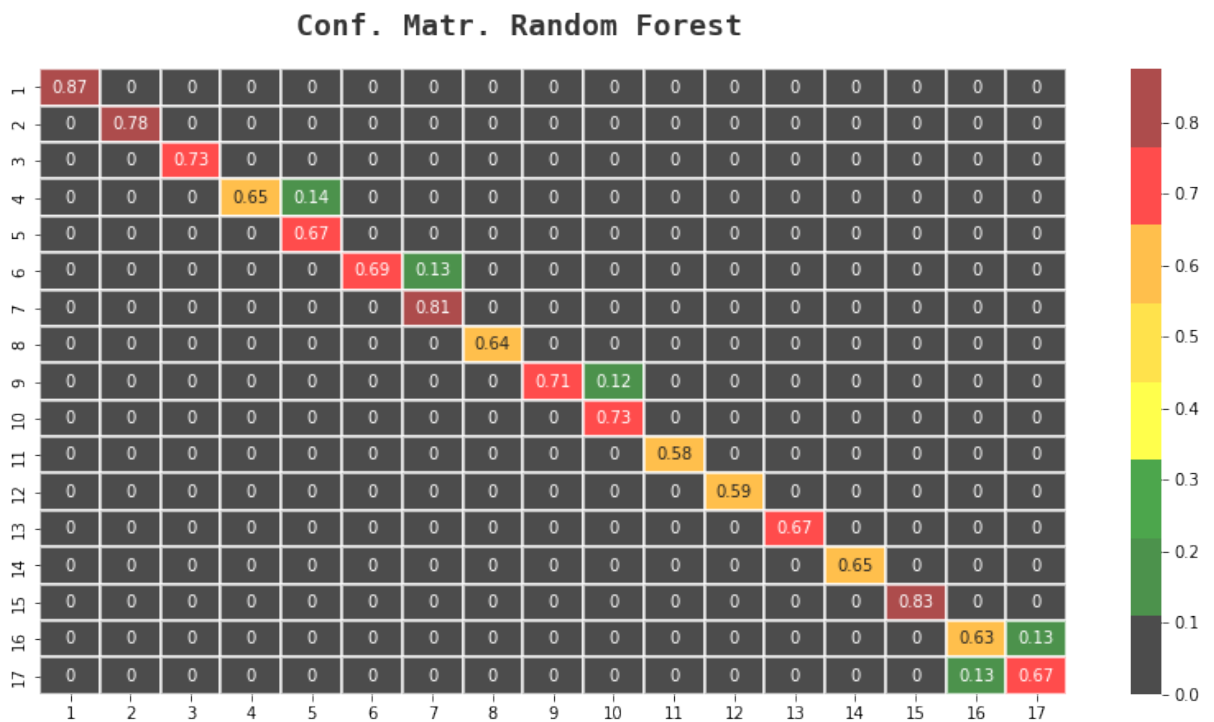


Figure 3.23: Confusion Matrix of the final RF model

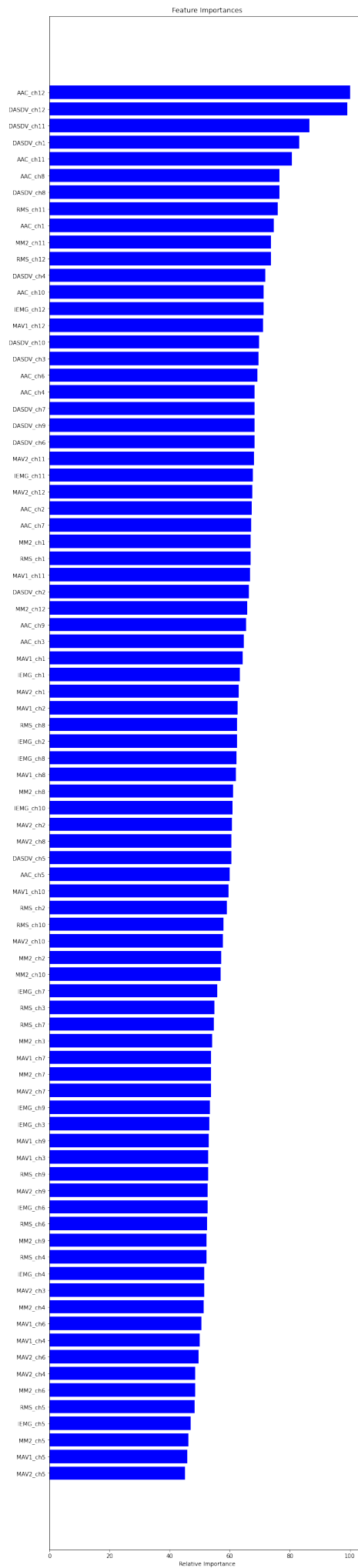
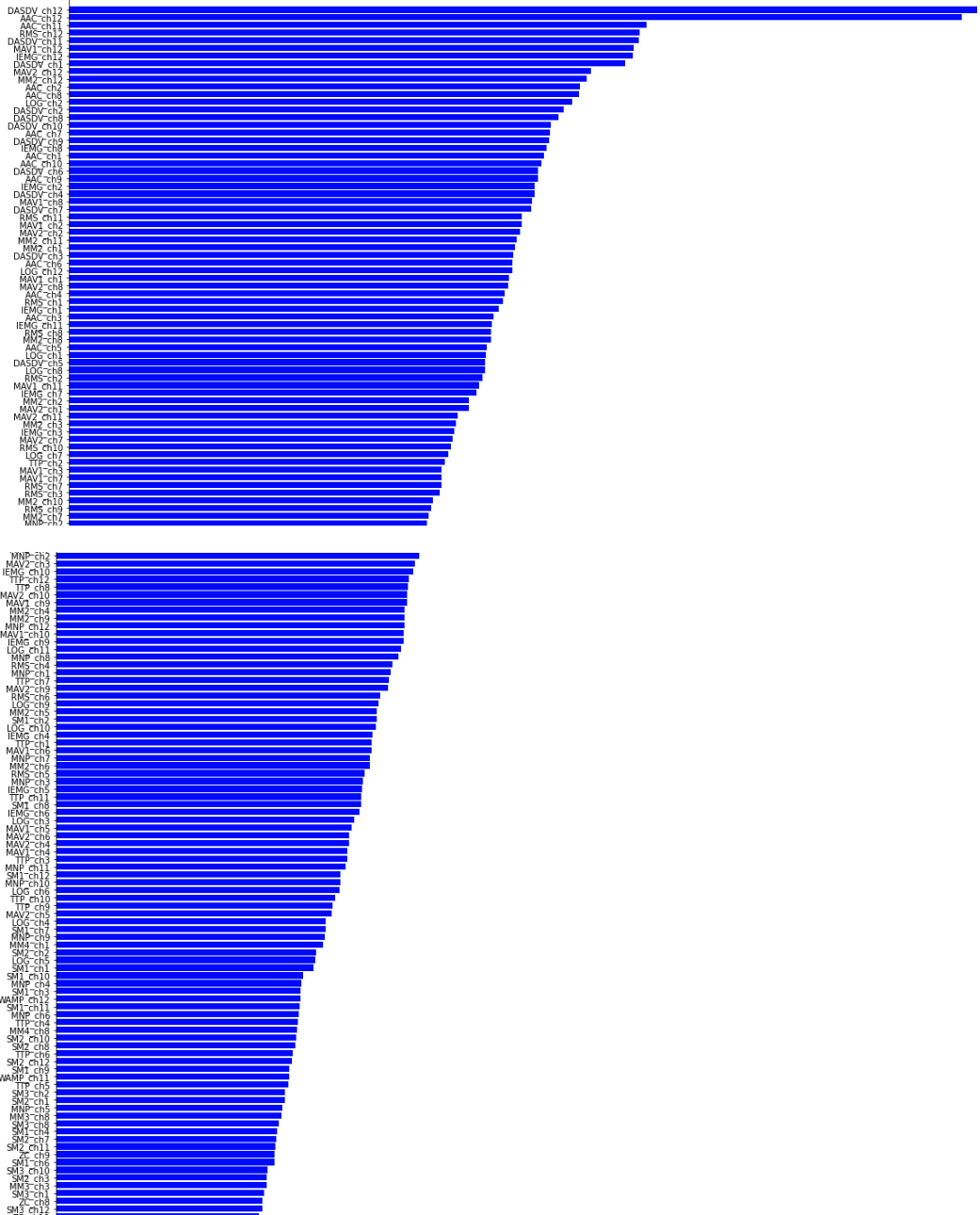


Figure 3.24: Feature importance for the final RF model

Selecting these 6 features for each channel, the total number of features shrunk from 276 to 72, leading to a reduced computational cost; the model was trained again and the validation accuracy increased from 75% to 78% (in Fig. 3.27a it is possible to see the new features importance). Finally, it was trained considering training and validation set merged. The test accuracy was of the 77%. The performances were therefore evaluated on the test set through the computation of the *precision*, *recall* and *f1-score* and of the confusion matrix (Tab. 3.24a, Fig. 3.28a).

Working with the second subset of gestures, the most important features were: RMS (eq. 1.9), IEMG (eq. 1.1), MM2 (eq. 1.5), DASDV (eq. 1.12), TTP (eq. 1.21) and MNP (eq. 1.20) (Fig. 3.26). Using these features (7x12), the new validation accuracy augmented from 68% to 71% and the new features importance was evaluated (Fig. 3.27b). The model was then trained merging the validation and training set: the performances were assessed over the test set using the classification metrics and the confusion matrix, which can be seen respectively in Tab. 3.24b and Fig. 3.28b. The final test accuracy was of 70%.

Feature Importances



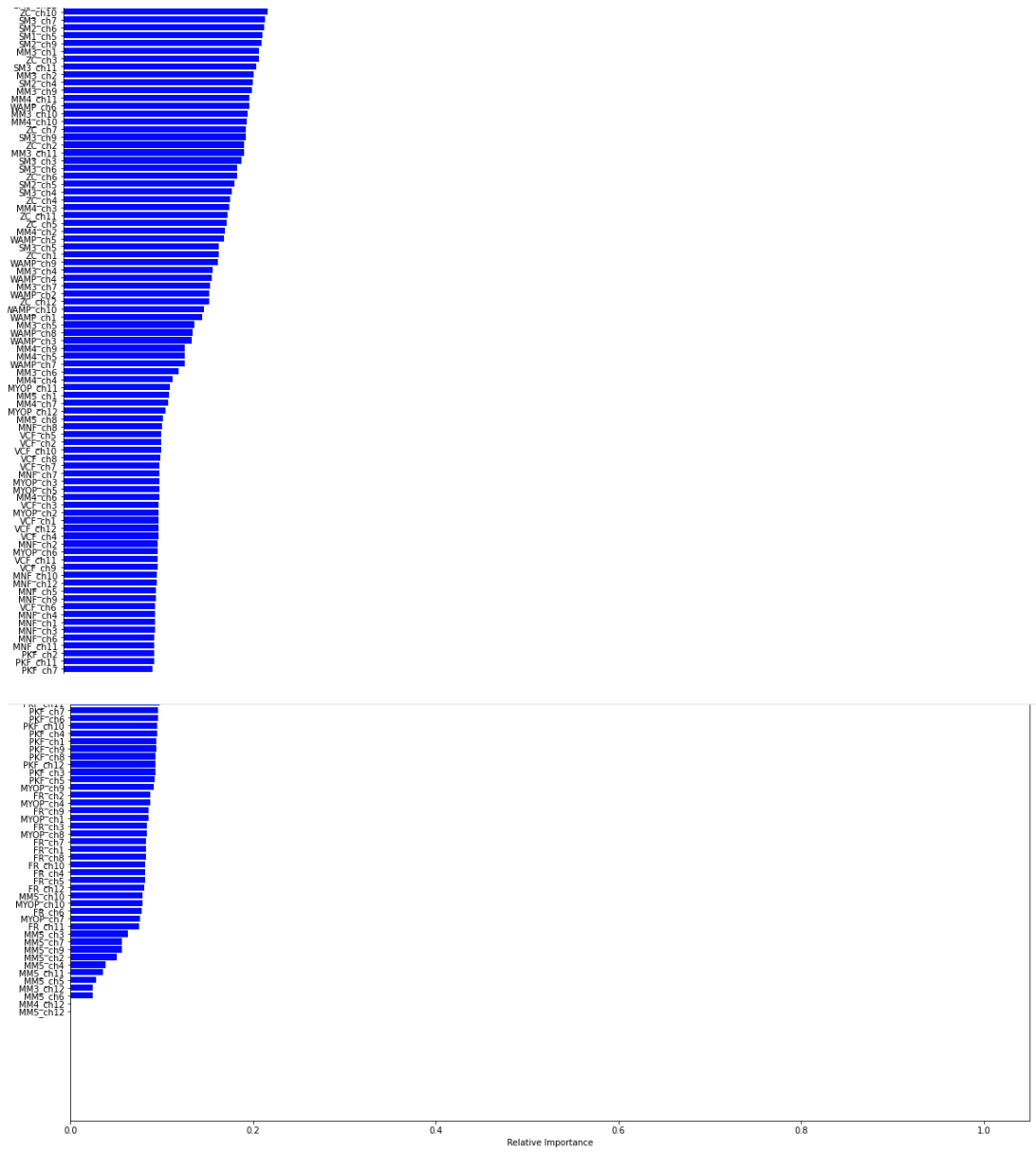
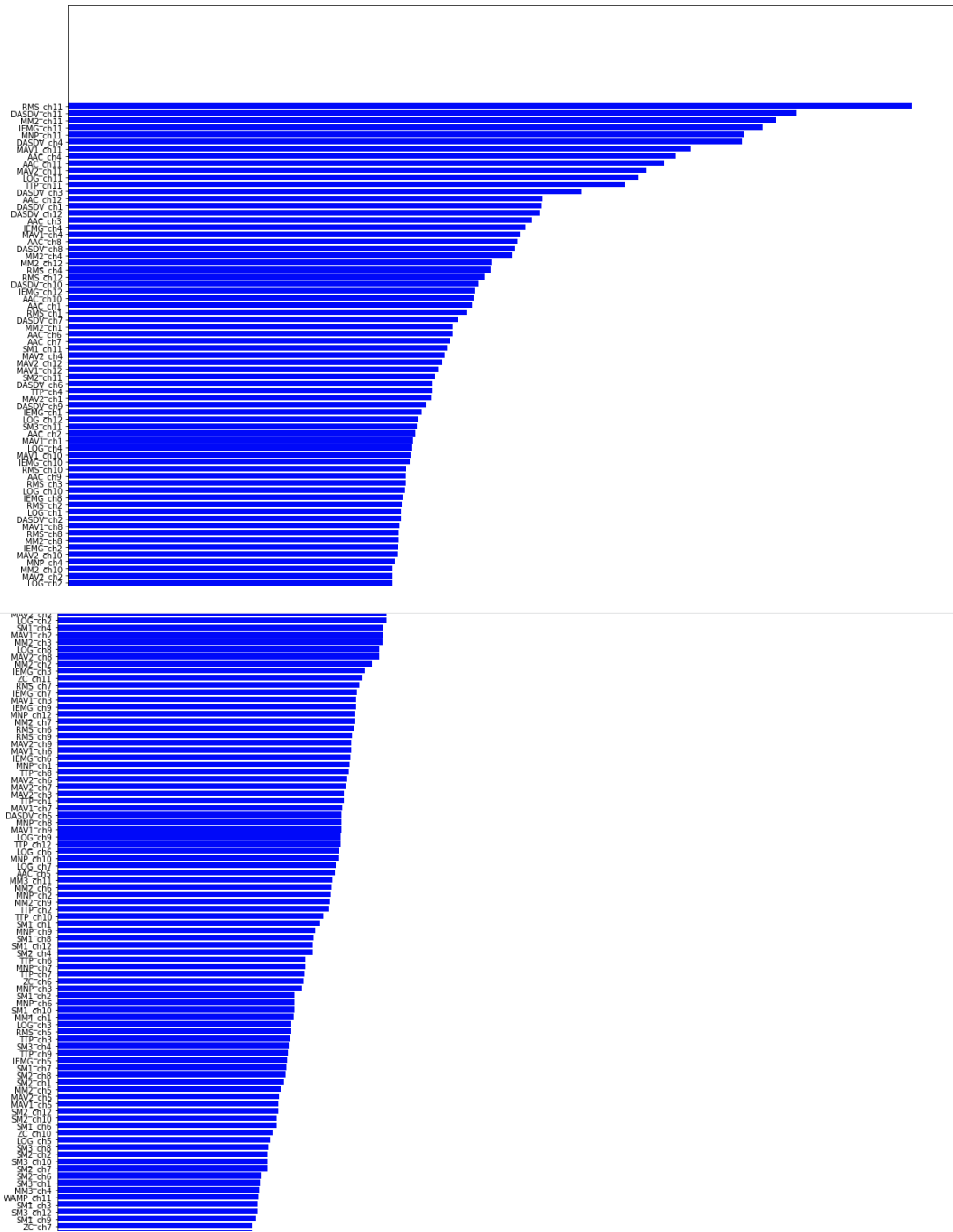


Figure 3.25: Relative importance features for the first gestures subset



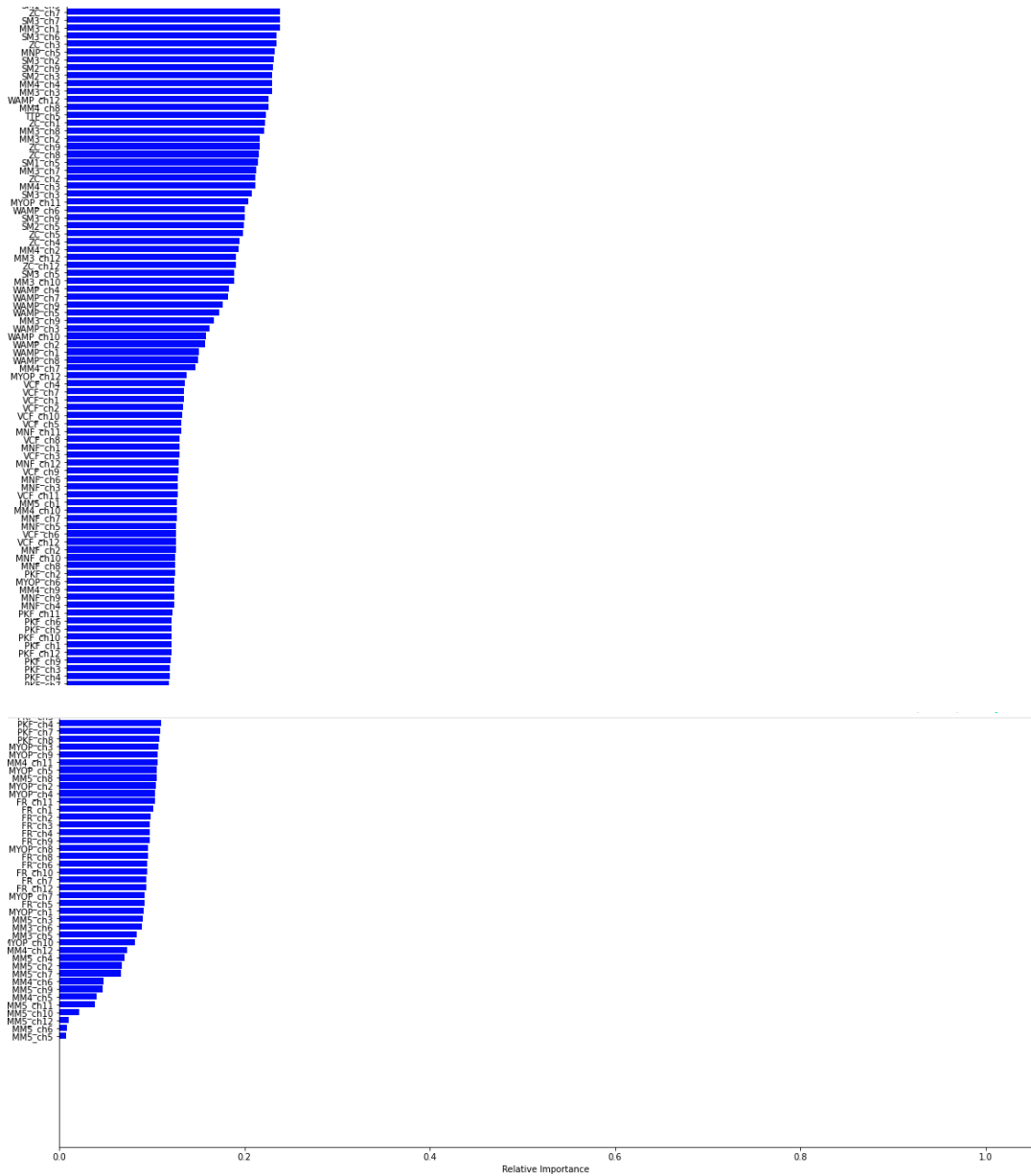


Figure 3.26: Relative importance features for the second gestures subset

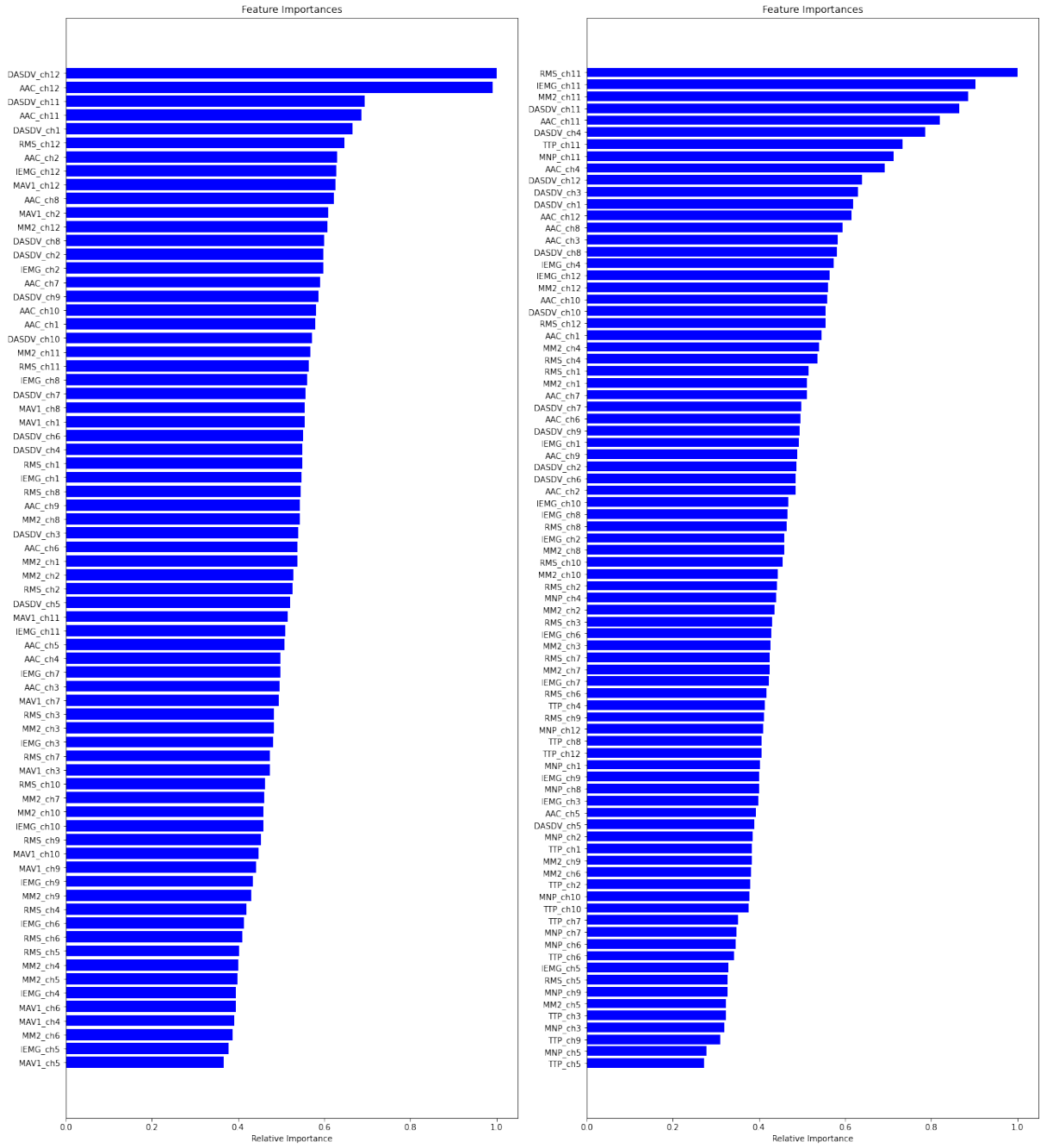


Figure 3.27: Final features importance for the (a) first and (b) second gestures subsets.

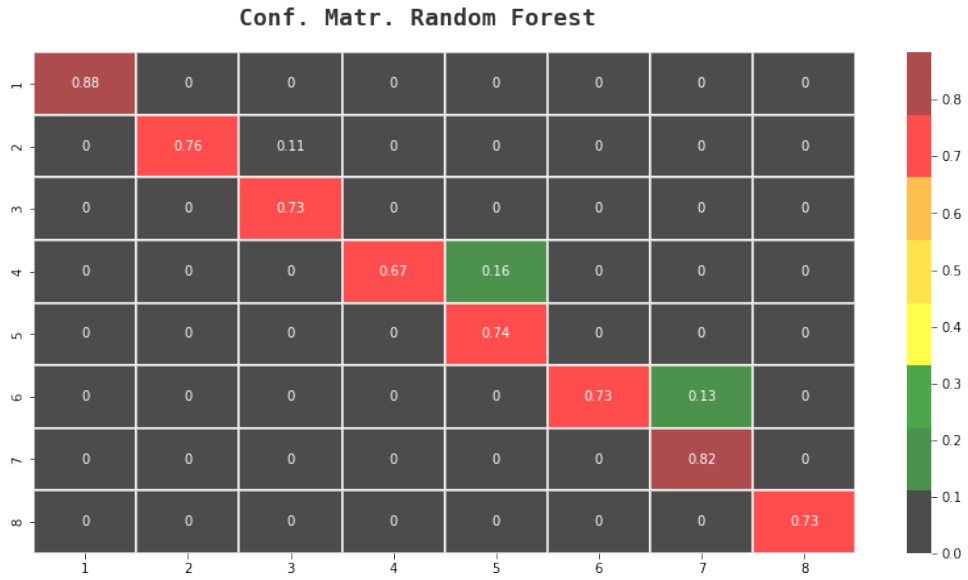
		precision	recall	f1-score
Gesture	1	0.83	0.89	0.86
	2	0.79	0.77	0.78
	3	0.77	0.74	0.75
	4	0.74	0.68	0.71
	5	0.68	0.73	0.71
	6	0.81	0.73	0.77
	7	0.75	0.82	0.78
	8	0.75	0.72	0.73
accuracy				0.77
macro avg		0.76	0.76	0.76
weighted avg		0.77	0.77	0.77

(a)

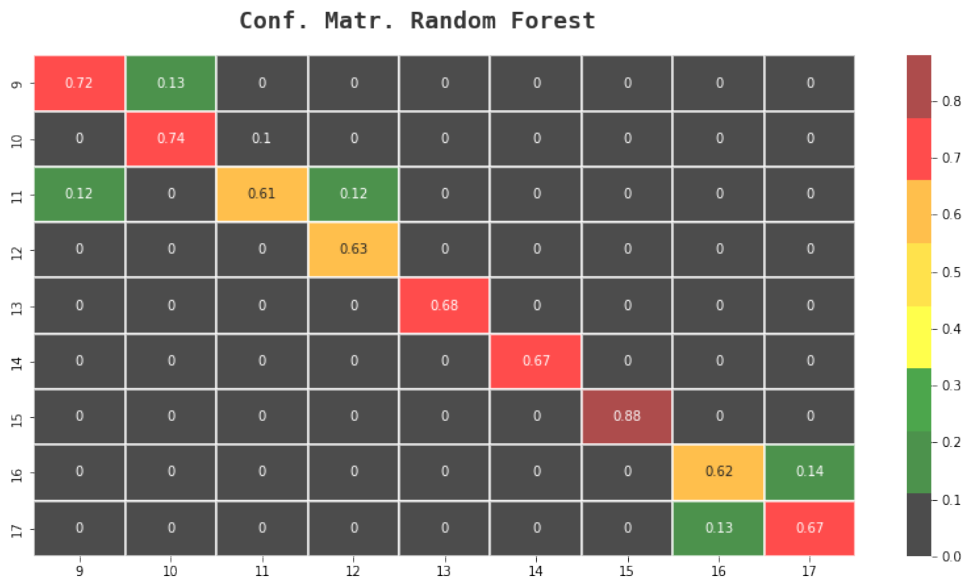
		precision	recall	f1-score
Gesture	9	0.74	0.73	0.74
	10	0.72	0.75	0.73
	11	0.63	0.62	0.63
	12	0.62	0.65	0.63
	13	0.72	0.69	0.71
	14	0.74	0.68	0.70
	15	0.75	0.88	0.81
	16	0.69	0.63	0.66
	17	0.73	0.68	0.71
accuracy				0.70
macro avg		0.70	0.70	0.70
weighted avg		0.70	0.70	0.70

(b)

Table 3.24: Classification metrics for the final RF with the first (a) and second (b) gestures subset.



(a)



(b)

Figure 3.28: Confusion matrices for the final RF with the first (a) and second (b) gestures subset.

3.3 The Bayesian Fusion

As explained in the chapter 2, the bayesian fusion was employed considering the best models obtained for each of the studied signals. For what concerns the sEMG signal, there are no doubts that the conventional machine learning approach with the RF resulted the best one. On the other hand, working with glove, there were three models that outed to work similarly: the ResNet, the K-NN and the SVM. From the analysis of the confusion matrices and classification matrix of these classifiers, it was decided to discard the ResNet from this kind of analysis. Indeed, it had an average of 86% for each of *precision*, *recall* and *f1-score* versus the 87% of the others two. Thereafter, since no substantial difference was found between the K-NN and SVM, both of them were exploited to develop the fusion.

As described before, the fusion approach uses the model validation confusion matrices. This techniques was implemented following two steps:

1. Evaluation of the confusion matrix over the validation set for both the models.
2. Networks training on the validation and training set merged and performances evaluation over the test set.

To highlight the performances of the fusion approach, The results were represented by comparing the average accuracy on the test set between the single signal models and the fusion one. In Tab. ?? a comparison between the test accuracies when using the KNN is presented: KNN (glove), RF (sEMG), KNN and RF fused (glove and sEMG).

On the other hand, in Tab. 3.25 it is possible to see a comparison between the test accuracies when using the SVM: SVM (glove), RF (sEMG), SVM and RF fused (glove and sEMG).

The results underline how the fusion approach does not allow to increment the overall accuracy in relation to when only glove data are used. Since in Tab. 3.25 only the average accuracy is displayed, it could be interesting to analyze the changes in the confusion matrices of the models due to the fusion approach, thus revealing how the gestures are newly classified.

Glove	sEMG	Glove & sEMG
KNN	RF	KNN + RF
87%	71%	87%

(a)

Glove	sEMG	Glove & sEMG
SVM	RF	SVM + RF
87%	71%	87%

(b)

Table 3.25: Fusion results: (a) first configuration and (b) second configuration

1	0.09	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	0	0.14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	0	0	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0.19	0.01	0	0	0	0	0	0	0	0	0	0	0	
5	0	0	0	0	0.24	0	0	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0.21	(-) 0.12	0	0	0	0	0	0	0	0	0	
7	0	0	0	0	0	0	0.1	0	0	0	0	0	0	0	0	0	
8	0	0	0	0	0	0	0	0.25	0	0	0	0	0	0	0	0	
9	0	0	0	0	0	0	0	0	0.2	0.04	0	0	0	0	0	0	
10	0	0	0	0	0	0	0	0	0	0.10	0	0	0	0	0	0	
11	0	0	0	0	0	0	0	0	0	0	0.28	0	0	0	0	0	
12	0	0	0	0	0	0	0	0	0	0	0	0.18	0.12	0	0	0	
13	0	0	0	0	0	0	0	0	0	0	0	0	0.16	0.12	0	0	
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0.22	0	0	
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.09	0	
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.19	0.01
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	(-) 0.13	0.24
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

(a)

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	(-) 0.01	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	(-) 0.01	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0.05	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	(-) 0.07	0.12	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	(-) 0.01	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	(-) 0.01
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

(b)

Figure 3.29: Differences between the confusion matrices of the fusion approach (KNN and RF) and (a) RF (sEMG only) (b) KNN (glove only). In red are displayed worsening changes from the single signal approach to the fused signals one, in green are figured performances improvements.

1	0.08	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	0	0.16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	0	0	0.21	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0.19	0	0	0	0	0	0	0	0	0	0	0	0	
5	0	0	0	0	0.21	0	0	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0.22	(-) 0.12	0	0	0	0	0	0	0	0	0	
7	0	0	0	0	0	0	0.12	0	0	0	0	0	0	0	0	0	
8	0	0	0	0	0	0	0	0.19	0	0	0	0	0	0	0	0	
9	0	0	0	0	0	0	0	0	0.07	0.01	0	0	0	0	0	0	
10	0	0	0	0	0	0	0	0	0	0.09	0	0	0	0	0	0	
11	0	0	0	0	0	0	0	0	0	0	0.21	0	0	0	0	0	
12	0	0	0	0	0	0	0	0	0	0	0	0.22	0.11	0	0	0	
13	0	0	0	0	0	0	0	0	0	0	0	0	0.13	0	0	0	
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0.15	0	0	
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.11	0	
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.16	0.01
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	(-) 0.13	0.18
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

(a)

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0.01	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	(-) 0.01	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0.01	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0.01	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	(-) 0.01	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	(-) 0.04	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0.02	(-) 0.01	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	(-) 0.04	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

(b)

Figure 3.30: Differences between the confusion matrices of the fusion approach (SVM and RF) and (a) RF (sEMG only) (b) SVM (glove only). In red are displayed worsening changes from the single signal approach to the fused signals one, in green are figured performances improvements.

Chapter 4

Discussion

Briefly, in this work the potentiality of data glove and sEMG signals were tested to address hand gesture classification tasks, exploring the ability of both classical machine learning and deep learning approaches. The results obtained among all the investigated configurations are summarized in Tab. 4.1, when working with data glove, and in Tab. 4.2 when dealing with the superficial electromyography.

	Network	Training Accuracy	Test Accuracy
Glove	CNN	91%	80%
	LSTM	92%	83%
	RESNET	98%	87%
	KNN	100%	87%
	DT	100%	67%
	SVM	98%	87%

Table 4.1: Summarized glove results






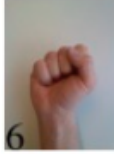
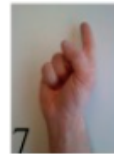

	Network	Training Accuracy	Test Accuracy
sEMG	CNN - 1st configuration	75%	64%
	CNN - 2nd configuration	79%	67%
	CNN - 3rd configuration	68%	60%
	CNN - 4th configuration	67%	61%
	RF (final)	100%	71%

Table 4.2: Summarized sEMG results










4.1 Data glove

Results suggest how the glove is more informative than the sEMG signal. DL approaches had success in addressing the task, without the necessity to process the signal. In fact, raw signal was directly given as input to the network just as it was measured, with the only requirement of splitting the acquisition in observation windows of 150 ms.

Another interesting result is that, using deep learning approaches, the glove seems to have a good generalization ability. This can be retrieved looking at the differences between the training accuracy and the test accuracy, which are not drastically high. This means that the networks are learning the true underlying relationship between the samples and their labels, and that these dependencies repeat also in unseen data. This was an expected result because the glove is characterized by a stronger ability in overcoming the problem of the high variability in physiological signals; in fact, it measures joint angles which are quite reproducible among different subjects, or at least their proportions.

1st exercise							
1	2	3	4	5	6	7	8
							

(a)

2nd exercise				
9	10	11	12	13
				
14	15	16	17	
				

(b)

Table 4.3: Gestures map. [2, 14]

Among the tested DL networks, the ResNet resulted the best one because it reached an

optimal compromise between the time spent to learn the data-labels dependencies (8 minutes less than the LSTM) and also because of the final accuracy on the test set. It is interesting also to analyze the classification metrics and the confusion matrix in order to understand which are the gestures that have more probability to be misclassified than others.

Looking at the confusion matrix of the ResNet (Fig. 4.1), it is possible to identify three groups of hand movements that are most of all confounded (using as reference the Tab. 4.3):

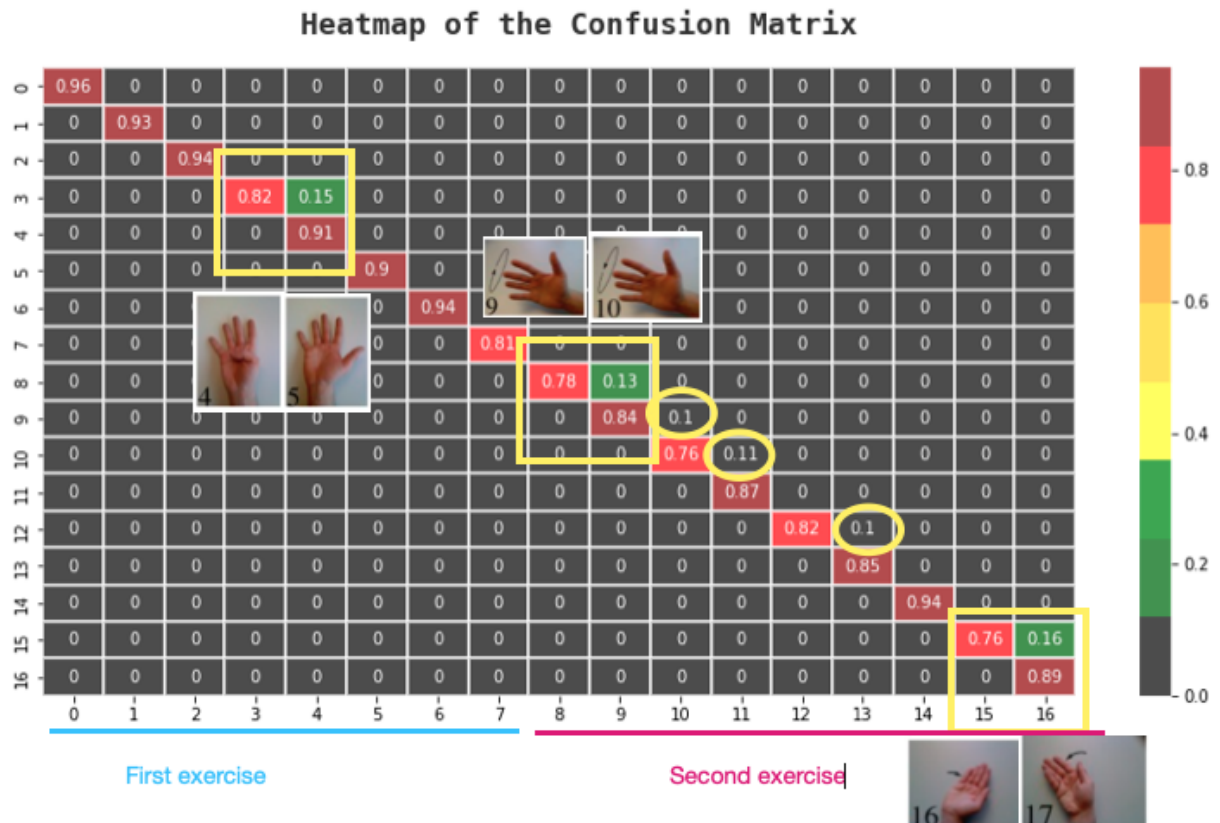


Figure 4.1: ResNet Confusion Matrix. The rows and columns gestures labels are decreased of 1 unit. The yellow squares are highlighting the major confounding issues of the network. The yellow circles instead highlight the minor problems.

- **Gestures 4 and 5:** The gesture number 4 is confounded with the number 5. The difference between them is the thumb position; indeed, in the gesture number 4 the thumb is flexed, while in the other it is extended;
- **Gesture 9 and 10:** these two movements differ very little one from the other, because they involve two opposites direction of the wrist rotation.
- **Gestures 16 and 17:** even in this case, the glove may have some difficulty in distinguishing correctly these two gestures, because their changing is related only to the direction of the movement.

- **Second subset of gestures:** it seems like the network, in its overall behaviour, has more difficulty in distinguishing the second subset of the gestures. The lower performances could be due to the fact that it contains completely different kind of movements with respect to the first subset; indeed, they represent wrist movement gestures.

Conventional machine learning techniques, as KNN and SVM, worked well too when dealing with data glove. It is interesting to notice how their performances highlight the ease of use of such signal. Indeed, it was extracted only one feature per channel (22), the RMS, in order to make these classifiers work. The difference between the two type of accuracies were not so high, meaning that it was achieved a substantial generalization ability. The only model that should be discarded in such sense is the Decision Tree, which came up with a pretty strong overfitting.

As described in the results chapter, there are no substantial differences between the results obtained using KNN and SVM. From the analysis of their confusion matrices (Fig. 3.11, 3.13), similar considerations to the previous can be made, even though the performances of this conventional classifiers moderately increased for the second subset. Indeed, compared to the ResNet performances, these classifiers have a little confounding component just between the gestures number 12 and 13. The other, instead, had more significant components of misclassifications (9-10, 10-11, 12-13).

In both cases, with conventional classifiers and with deep learning approaches, the glove data resulted to be a signal that easily allows to build robust and reliable classifiers, without any particular effort. On the other hand, using data glove generally implies higher expenses, not just in terms of money but also in terms of computational cost, than when dealing with superficial electromiography. Here, it can be underlined an important difference between machine learning and deep learning approaches: the seconds typically requires more time to be trained, but once the network parameters are determined, the time needed to perform predictions is drastically lower.

Final considerations can be made regarding the outcomes obtained when the reduced gestures set was tested (considering the first exercise only). From the results previously obtained, it appeared that the first subset of gestures is easier to be classified. The expected results were thus an increasing on the overall performances when the reduced subset was considered. Indeed, it improved the performances reaching the 92% of test accuracy. Despite this increment, the network had still some problem in distinguishing correctly between the gestures number 4 and 5 (Fig. 4.2). Moreover, it appears that when the second exercise set is discarded, the

network isolates better the gesture number 8.

Conventional machine learning gave the same overall outcomes. The performances of SVM in particular increased slightly more than the others, because it reached an average accuracy of 93%.

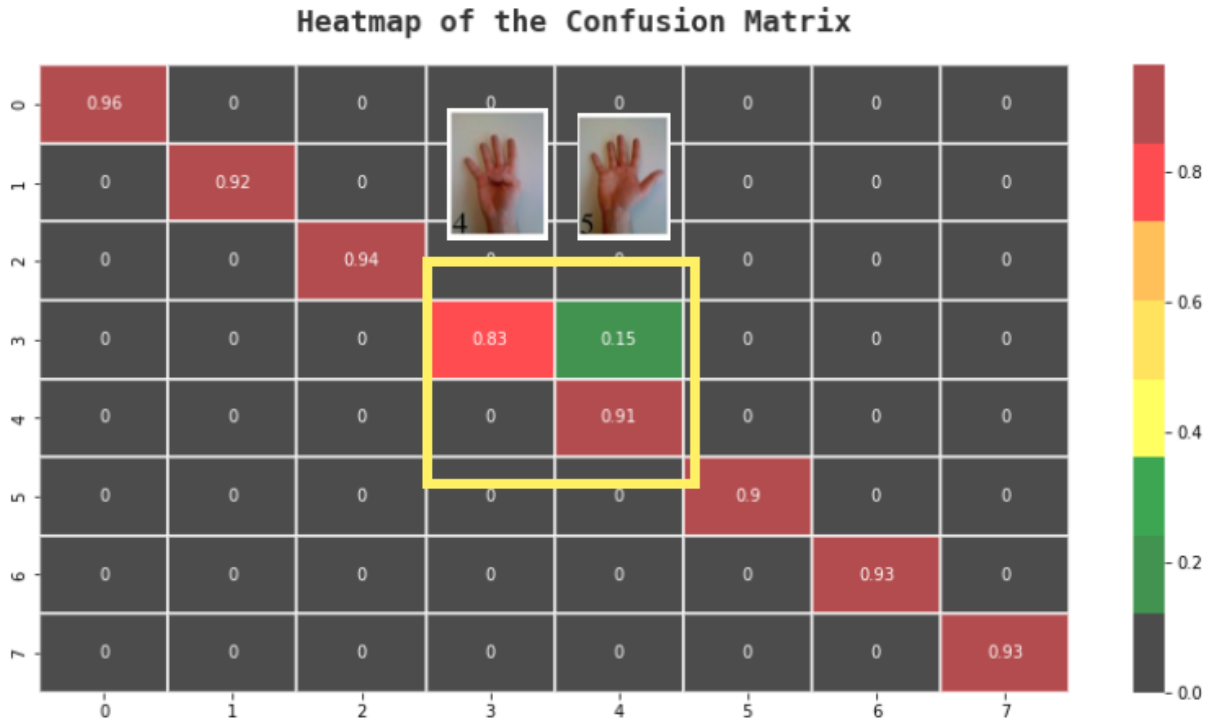


Figure 4.2: Confusion Matrix of the ResNet when dealing with the first exercise hand positions only. The yellow square shows the major difficulty of the network.

4.2 sEMG

From the results obtained by the handling of the sEMG, it is instead clear its weakness in terms of signal explainability and repeatability for the same hand movement among different subjects, or within the same. Indeed, the performances obtained in this case are not really high and the classifiers generally have a significant difficulty in distinguishing correctly between the different kind of gestures.

With respect to the abilities of data glove, the sEMG needed to be processed accurately to become informative. In particular, DL techniques were built in order to look for relationships between the features and the 12 measurement channels. This ability seemed to be restricted due to the percentage of accuracy they reached, which got worse with the 2D configuration. Indeed, input data organized as vectors allowed the networks to increase the performances. Moreover, the performances slightly increased when only time domain features were used, suggesting

how probably there is some redundancy among the whole extracted features. Since it is well-known how working with too much features may be inappropriate and could alter (worsening) the performances of the network, a RF approach was employed in order to easily evaluate the features importance. This was useful to select a restricted number of features with which the model could work, testing the eventual advantage of casting down the redundancy. This is possible thanks to how the Random Forest works: it builds a specified number of Decision Trees (300 in this case) and for each of them it considers randomly a target number of features. The Decision Tree uses a top-down greedy approach, where at each step it builds the best possible split by evaluating a specific feature and comparing it upon a defined threshold. The feature importance is derived by looking at how many times a feature is taken into account (among all the estimators) to perform splits and how strongly it decreases the node impurity after the split itself. The feature importance evaluation is a robust and reliable process, because it is based on a consistent number of estimators.

When the complete gesture set was evaluated, the most important features (with a relative importance higher than 60%) were RMS, IEMG, DASDV, MAV1, MAV2 and AAC. Interestingly, when the complete set is considered, the time domain features completely dominate upon all the others. The accuracy increased from 69% to 71%, underlying how selecting features allowed to shrink the redundancy and so to improve the performances. From the analysis of the Random Forest classification metrics and confusion matrix Fig. 4.3, it is possible to understand which are the gestures that most trouble the model:

- **Gestures 4 e 5:** these gestures were confounded with data glove too, underlining how these are probably very similar, both from joint angles and neuromuscular points of view.
- **Gestures 6 and 7:** this issue was not present when dealing with data glove, meaning that probably the muscular signal here is pretty similar for the two gestures. The difference between these two gestures is in the extensions of the index finger on the gesture number 7.
- **Gestures 9 and 10:** The same problem characterized the network that worked with data glove;
- **Gestures 16 ad 17:** the issue was present even working with data glove, but in this case it seems emphasized.

For the classifier, it was harder to correctly distinguish the gestures of the second exercise, because of the lower levels of accuracy that each class has with respect to the values of the first

exercise gestures.

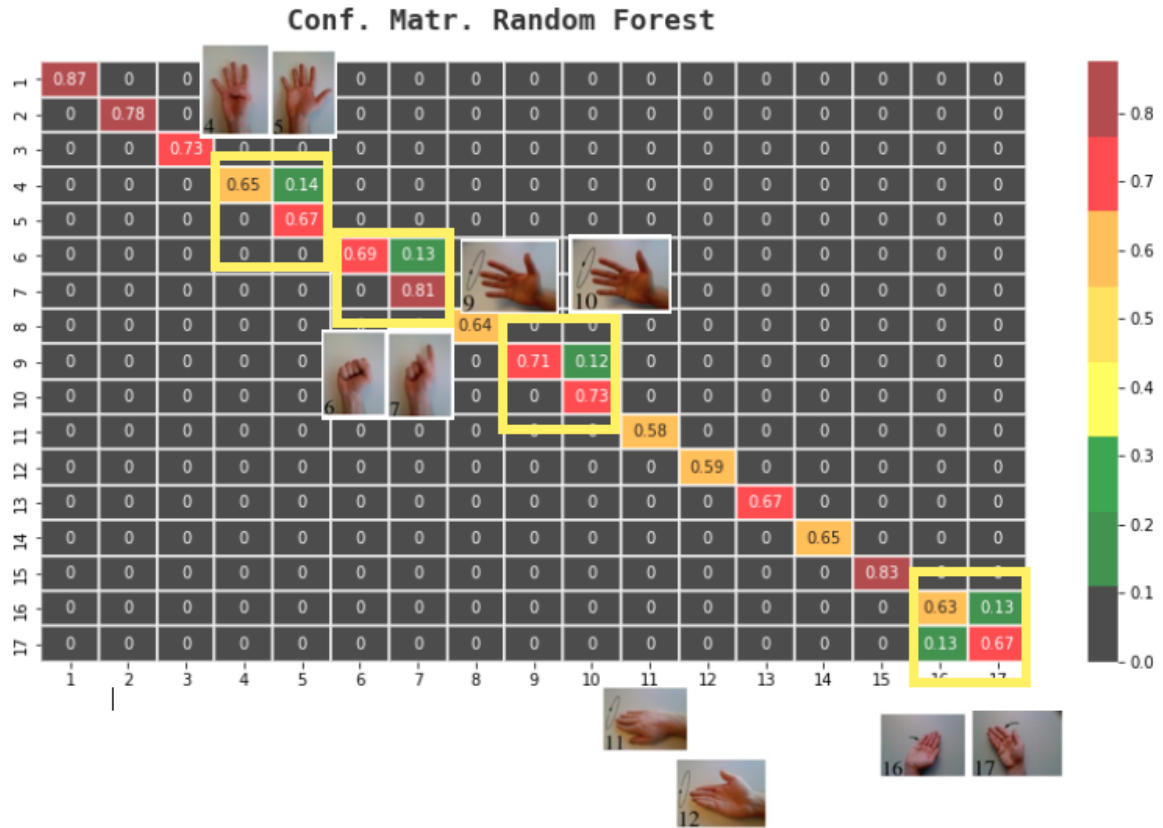


Figure 4.3: Random Forest confusion matrix. The yellow squares identify the major issues of the network.

At this point, it was decided to analyze separately the two subsets of gestures, investigating which are the features that are mostly related to each of them. As discussed in the methods chapter, the most important for the first subset resulted: RMS, IEMG, MM2, DASDV, MAV1 and the AAC; interestingly, these were all time-domain features. The final RF built with the selected features gave as result the confusion matrix at Fig. 3.28a. The overall accuracy is in this case of the 77%, which is greater than the all-gestures configuration. The augmented performance is due both to the restricted range of classes and to how much a class is correctly identified by the selected features. Even though the general performances increased, the model has still some problems in distinguishing correctly between the couples of gestures 4-5 and 6-7, exactly how it was when the entire gesture set was considered.

Another interesting consideration came up when the second subset of gestures was analyzed. The most important features were this time: RMS, IEMG, MM2, DASDV, TTP and MNP. Differently, this time there are two frequency domain features among the most relevant according to the results of the Random Forest. This outcome can be interpreted by thinking at these kind of features as related to dynamic gestures and thus to the dynamic movements. Indeed, as described in the introduction, frequency domain features are related to the muscle

fatigue, which can be thought to be associated to the dynamic movement of the hand. When the RF classifier was trained with the selected features to classify the second subset of gestures, it did not reach a satisfying accuracy value. It was almost identical, even less than when all the gestures were considered: this remarks how the second subset of gestures is more difficult to be correctly classified. Analyzing the confusion matrix (reported in Fig. 4.4) it is possible to see how the model is confounding a group of gestures (9-12).



Figure 4.4: Confusion matrix for the RF working with the second subset of gestures.

The hand-crafted features extracted were referred to each of the 12 channels used by the scientists that have collected data for the NinaPro DB. When the features importance tool was used, the RF looked at each feature of each channel separately. Indeed, looking at Fig. 3.22, 3.24, 3.26, each channel contributes in a different manner; notably, in each of the previous figure the same pattern can be identified. It seems like the channels 8, 10, 11, 12 are on the top of the ranking features list. Probably this result is due to the fact that these electrodes are the best positioned among the others, in the sense that, for the gestures analyzed, they optimally capture the signal. In Atzori et. Al it is explained how the electrodes are positioned: the number 11 and 12 were placed on the main activity spots of the biceps brachii and of the triceps brachii, the 9 and 10 instead on the main activity spots of the flexor digitorum superficialis and of the extensor digitorum superficialis. The number 8 was with the remaining around the forearm, at the height of the radio-humeral joint, but unfortunately the precise position is not known.

4.3 The Bayesian Fusion

The last element of the work was building and applying the Bayesian Fusion approach, trying to exploit the best performances obtained from the two signals. The obtained results were not satisfying; the approach worked from the sEMG point of view because it actually allowed to increase its performances. On the other hand, it seemed like the fusion between the two models behaved exactly like the glove model alone did. This was an expected result: due to the low performances obtained by analysing the sEMG with RF and to the way in which the Bayesian Fusion works, it looked like the fused model decided to rely mostly in the glove model predictions. These considerations can be easily retrieved by looking at Fig. 3.29 and 3.30. In these figures, the differences between the confusion matrices of the Fusion and of the two single-signal classifiers used are represented. The significant changes (>0.1) are only present in the confusion matrix evaluated by the subtraction from the one of the Bayesian Fusion and the one of the Random Forest (sEMG only). The other, did not have significant changes, neither positive nor negative. This also confirm the strong similarity between the two classifiers considered to make the Bayesian Fusion.

4.4 Conclusion and Future Improvements

In conclusion, this work underlines the ease of use of the data glove in the development of a reliable and robust hand gestures classifier. Despite the high computational cost, it does not require particular processing to extract relevant information aimed at the correct interpretation of the hand movement. Firstly, deep learning approaches as the ResNet clearly succeed in autonomously extract features of the signal; secondly, machine learning approaches were able to provide accurate results by the computation of one feature per channel.

On the contrary, the results obtained confirmed the sEMG low ease of use. To obtain marginal satisfying results, it was necessary to extract and compute hand-crafted features. Deep Learning approaches could not outperform the conventional machine learning techniques, even though the first were not affected by overfitting. Despite the higher test performances of the conventional techniques, they were not able to generalize well because of the high difference between the training accuracy and the validation accuracy. The analysis of the most important features, allowed to do interesting considerations regarding not only the kind of features that better represent the two subsets of gestures (referring to two different kind of exercises), but even the channels that mostly contributed to the classifications. These information could

eventually be used as starting points for the development of an improved sEMG hand gesture recognition system.

Having obtained such low performances when dealing with only the sEMG, the bayesian fusion approach was not expected to work and increase the overall performances. Indeed, they did not significantly improve or change in relation to the one achieved using data glove only. The fusion actually enhanced the behaviour of the sEMG classifier, but there would be no advantage in using the technique under this conditions; indeed, since it would be sufficient to use only data glove, it uselessly increases the cost of the system.

The future improvements that could be suggested at this point are mainly focused in augmenting the sEMG performances:

- **Selecting differently the features:** the intrinsic RF functionality of evaluating the feature importance allowed to select the ones that contribute more to determining the trees splits. Having obtained an increased accuracy once reduced the features means that some redundancy was discarded. The image of the features importance, highlights how there were some channels that were distinctly more significative than others. In this work, a subset of features were selected, without caring to the channels different impact; in future works, the channels relevance can be considered. In this way it could be analyzed if there is redundancy even among the channels; in fact, the information could be repetitive not just in the way it is extracted but also in how it is measured.
- **Evaluating new hand-crafted features:** in this work, only time domain and frequency domain features were computed and used. As discussed in the introduction, there are more features that can be eventually used, exploring, as examples, the time-frequency domain or the fractal domain.
- **Different Pre-processing:** sEMG signal windowing was performed in order to correctly compare the performances with the classifiers that used data glove; in particular, non-overlapping windows were considered. Introducing overlaps between adjacent windows may allow to evaluate the hand-crafted features with a sort of continuity that can be more representative of the gestures execution. This new data organization may permit to explore new deep learning networks also without extracting hand crafted features.

Moreover, for features like WAMP, MYOP, ZC, and SSC, a specific threshold was selected ($100 \mu V$). Other thresholds could be tried.

- **Different Fusion type:** once the performances of the sEMG are increased, the Fusion

approach could be tested again to see if the idea worked like it did in Tortora et. Al. Then, more fusion approaches could be evaluated, like the Fuzzy template. This technique works by creating a template template for each class from the validation set, then weighting the classifiers predictions based on the similarity with the class template.

Chapter 5

References

1. AYODELE, Emmanuel, et al. A review of deep learning approaches in glove-based gesture classification. *Machine Learning, Big Data, and IoT for Medical Informatics*, 2021, 143-164.
2. ATZORI, Manfredo, et al. Electromyography data for non-invasive naturally-controlled robotic hand prostheses. *Scientific data*, 2014, 1.1: 1-13.
3. AYODELE, Emmanuel, et al. Grasp classification with weft knit data glove using a convolutional neural network. *IEEE Sensors Journal*, 2021, 21.9: 10824-10833.
4. XXU, Deyou. A neural network approach for hand gesture recognition in virtual reality driving training system of SPG. In: *18th International Conference on Pattern Recognition (ICPR'06)*. IEEE, 2006. p. 519-522.
5. KONRAD, Peter. The abc of emg. A practical introduction to kinesiological electromyography, 2005, 1.2005: 30-5.
6. JIANG, Shuo, et al. Emerging wearable interfaces and algorithms for hand gesture recognition: A survey. *IEEE Reviews in Biomedical Engineering*, 2021, 15: 85-102.
7. ATZORI, Manfredo; COGNOLATO, Matteo; MÜLLER, Henning. Deep learning with convolutional neural networks applied to electromyography data: A resource for the classification of movements for prosthetic hands. *Frontiers in neurorobotics*, 2016, 10: 9.
8. IBARGUREN, Aitor; MAURTUA, Iñaki; SIERRA, Basilio. Layered architecture for real time sign recognition: Hand gesture and movement. *Engineering Applications of Artificial Intelligence*, 2010, 23.7: 1216-1228.

9. HEUMER, Guido, et al. Grasp recognition with uncalibrated data gloves-a comparison of classification methods. In: 2007 IEEE Virtual Reality Conference. IEEE, 2007. p. 19-26.
10. TAVAKOLI, Mahmoud; BENUSSI, Carlo; LOURENCO, Joao Luis. Single channel surface EMG control of advanced prosthetic hands: A simple, low cost and efficient approach. *Expert Systems with Applications*, 2017, 79: 322-332.
11. PHINYOMARK, Angkoon; PHUKPATTARANONT, Pornchai; LIMSAKUL, Chusak. Feature reduction and selection for EMG signal classification. *Expert systems with applications*, 2012, 39.8: 7420-7431.
12. JARAMILLO-YÁNEZ, Andrés; BENALCÁZAR, Marco E.; MENA-MALDONADO, Elisa. Real-time hand gesture recognition using surface electromyography and machine learning: a systematic literature review. *Sensors*, 2020, 20.9: 2467.
13. BUONGIORNO, Domenico, et al. Deep learning for processing electromyographic signals: A taxonomy-based survey. *Neurocomputing*, 2021, 452: 549-565.
14. ATZORI, Manfredo, et al. Characterization of a benchmark database for myoelectric movement classification. *IEEE transactions on neural systems and rehabilitation engineering*, 2014, 23.1: 73-83.
15. GENG, Weidong, et al. Gesture recognition by instantaneous surface EMG images. *Scientific reports*, 2016, 6.1: 1-8
16. PRADHAN, Ashirbad; HE, Jiayuan; JIANG, Ning. Open Access Dataset for Electromyography based Multi-code Biometric Authentication. *arXiv preprint arXiv:2201.01051*, 2022.
17. CEOLINI, Enea, et al. Hand-gesture recognition based on EMG and event-based camera sensor fusion: A benchmark in neuromorphic computing. *Frontiers in Neuroscience*, 2020, 14: 637.
18. COLLI-ALFARO, J. Guillermo; IBRAHIM, Anas; TREJOS, Ana Luisa. Design of user-independent hand gesture recognition using multilayer perceptron networks and sensor fusion techniques. In: 2019 IEEE 16th International Conference on Rehabilitation Robotics (ICORR). IEEE, 2019. p. 1103-1108.

19. TORTORA, Stefano, et al. Hybrid human-machine interface for gait decoding through Bayesian fusion of EEG and EMG classifiers. *Frontiers in Neurorobotics*, 2020, 14: 582728.

Appendix A

Deep Learning approaches

DL approaches are very recent techniques that offer an end-to-end approach in developing predictors or classifiers. Since they do not require the evaluation and extraction of hand-crafted features, they typically suffer from a the lack of understandibility. However, there is a growing interest into analyzing and using these approaches thanks to their ability to outperform, in some cases, the classical machine learning techniques.

Deep neural networks work by autonomously looking for interesting dependencies or features within the data during the training phase. This procedure consists in learning the weights that characterize and define the layers (independently from their type), thus determining the activation maps (or feature maps) of each layer from input data. The weights are trained thanks to the back propagation, which is the main concept behind the neural networks training. The training procedure of a network is subdivided in epochs and it starts by randomly assigning weights to each layer. Within an epoch, the training data are subdivided in data batches. For each batch, the classification/regression is made and then it is compared with the true label/value through the use of a specific loss function. There are several kinds of loss function, to be used differently in relation to the kind of task to perform and the most common for classification are:

- **Binary Cross Entropy.** It is used if the network has to solve a binary problem:

$$L = -\frac{1}{2} \sum_{i=1}^N y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i). \quad (\text{A.1})$$

y_i is the label referred to the i^{th} class, and can be 1 or 0. \hat{y}_i is the probability outed by the classifier for the i^{th} class.

- **Categorical Cross Entropy.** It is employed for multi-class classification problem:

$$L = - \sum_{i=1}^N y_i * \log(p(\hat{y}_i)). \quad (\text{A.2})$$

y_i is the ground truth and \hat{y}_i is the score for the i^{th} class outed by the network.

The evaluation of the loss is a measure to understand how much the network is making errors. To upgrade the weights, what is exploited in particular is the evaluation of the gradient of the loss, which is the vector of partial derivatives with respect to all coordinates of the weights:

$$\Delta_W L(W) = \left[\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_2}, \dots, \frac{\partial L}{\partial W_m} \right]^T \quad (\text{A.3})$$

If $\Delta_W L(W) = 0$, it means that a local optimum minimum is reached, which is the situation wanted. In order to reach the optimal minimum, it is necessary to move in direction $-\Delta_W L(W)$. In particular, each time a batch of data is predicted, the weights are updated following this formula:

$$W^{t+1} = W^t - \alpha \Delta_W L(W) \quad (\text{A.4})$$

where α is called learning rate and it determines how strongly should be the movement in direction $-\Delta_W L(W)$. When $\Delta_W L(W)$ is evaluated over a batch of data, it is called Stochastic Gradient Descent ($g^{(t)}$).

There are several ways of updating weights:

- **With momentum.** The objective is to make the trajectory of the weights update more stable, preventing losing time in oscillations. It uses the momentum, which can be interpreted as a force proportional to the velocity of the object but that acts in the opposite direction.

$$W^{t+1} = W^{(t)} + p^{(t+1)} \quad (\text{A.5})$$

$$p^{(t+1)} = \mu p^{(t)} - \alpha g^{(t)} \quad (\text{A.6})$$

where: $\mu \in [0, 1]$ is the momentum hyperparameter and $p^{(t)}$ is the momentum.

- **RMS propagation (RMSprop):** it scales gradients dividing by a moving average Root Mean Squared gradient ($s^{(t)}$), aiming at slowing down learning in directions where the gradient is higher and speeding up the process where the gradient is lower.

$$W^{(t+1)} = w^{(t)} - \alpha \frac{g^{(t)}}{\sqrt{s^{(t)}}} \quad (\text{A.7})$$

$$s^{(t)} = \beta s^{(t-1)} + (1 - \beta)(g^{(t)})^2 \quad (\text{A.8})$$

where $\beta \in [0, 1]$ is a moving-average decay factor.

- **Adaptive Gradient - ADAGRAD:** it is similar to RMSprop, but it uses the cumulative sum of squared gradients ($c^{(t)}$).

$$W^{(t+1)} = w^{(t)} - \alpha \frac{g^{(t)}}{\sqrt{c^{(t)}}} \quad (\text{A.9})$$

$$c^{(t)} = \sum_{j=1}^t (g^{(j)})^2 \quad (\text{A.10})$$

- **Momentum and RMSprop - ADAM:** it combines momentum and RMSprop moving averages.

$$W^{(t+1)} = w^{(t)} - \alpha \frac{p^{(t)}}{\sqrt{s^{(t)}}} \quad (\text{A.11})$$

$$p^{(t+1)} = \beta_1 p^{(t)} - (1 - \beta_1) g^{(t)} \quad (\text{A.12})$$

$$s^{(t)} = \beta_2 s^{(t-1)} + (1 - \beta_2) (g^{(t)})^2 \quad (\text{A.13})$$

A.1 Convolutional Neural Networks

The field in which CNNs were born is image analysis, which started from a simple classification task and evolved till reaching computer vision. Convolutional Neural Networks have a structure composed by several layers among which the most frequent are convolutional layers: they allow to perform convolution (2D or 3D) between the image received as input and a determined number of filters, which have specific dimensions in terms of width and height (it could have eventually depth dimension, for 3D convolution). Each convolutional layer is characterized also by the application of a non-linear activation function at the filters' output, leading to the so-called activation maps: it can be chosen the Rectified Linear Unit (ReLU), the Exponential Linear Unit (ELU), the Leaky ReLU or the Parameterized ReLU (PReLU). Each activation function has its own characteristics: Before the activation maps enter the subsequent convolutional block, they typically undergo a pooling layer, which is useful to reduce their dimensions by keeping just the most useful information. Several alternatives are presents, differing in how data are processed within a defined kernel (1D, 2D, 3D) that slides over the whole image:

- **Max Pooling:** it works by taking the maximum pixel value on an area defined by the kernel.
- **Average Pooling:** it computes the average of the values within the area of the superposed kernel.

Name	Function	PRO	CONS
ReLU	$f(x) = \max(0, x)$	<ul style="list-style-type: none"> No saturation for $x > 0$ 	<ul style="list-style-type: none"> No activation for $x < 0$ Not zero-centered
ELU	$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$	<ul style="list-style-type: none"> No saturation for $x > 0$ No de-activation for $x < 0$ 	<ul style="list-style-type: none"> Not zero-centered
Leaky ReLU	$f(x) = \max(0.01x, x)$		
PReLU	$f(x) = \max(\alpha x, x)$		

Table A.1: Non linear activation functions comparison.

Usually, at the end of the network there are fully connected layers which summarize the activation maps obtained by the final convolutional layer in a 1D vector, which is then converted into a probabilistic distribution. This means that the final FC layer must have the same number of units as the classes: in each unit, when a sample is given as input to the overall network, there will be the probability of that sample to belong at the class corresponding to the unit position. The formation of the final probabilities is possible thanks to the *softmax* activation function:

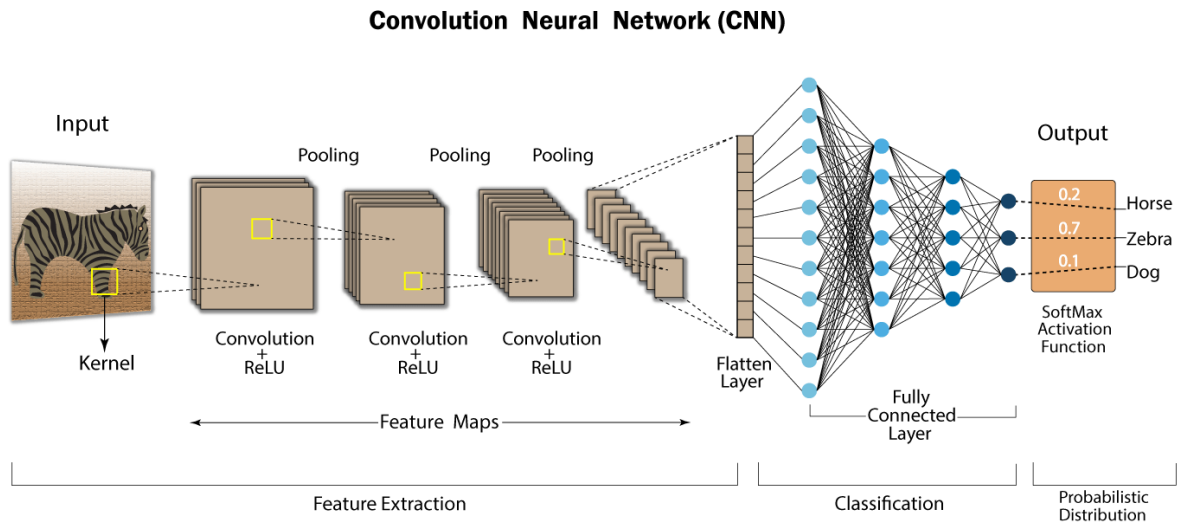


Figure A.1: Example of an usual CNN architecture. The Kernel is the filter (or more than one) that is convoluted with the image at each step.

$$f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^K e^{x_j}} \quad \text{for } i = 1, \dots, K \text{ and } \mathbf{x} = (x_1, \dots, x_k) \in \mathbb{R}^k \quad (\text{A.14})$$

where \mathbf{x} is the vector representing the final FC layer and K the number of its units (thus the number of classes). It applies the standard exponential function to each element x_i of the input vector \mathbf{x} and normalizes these values by dividing by the sum of all these exponentials; this

normalization ensures that the sum of the components of the output vector $\sigma(\mathbf{x})$ is 1, thus allowing to interpret the final values as probabilities. The final prediction is made by seeing the class that has the highest probability value.

A.1.1 CNN variants: the Residual Network (ResNet)

Through the years, some variants of the classical CNN were born, introducing modifications that improved, year by year, the network performances over a worldwide images dataset: the ImageNet. In Fig. A.2 the most relevant are represented along with their error rate. In 2015

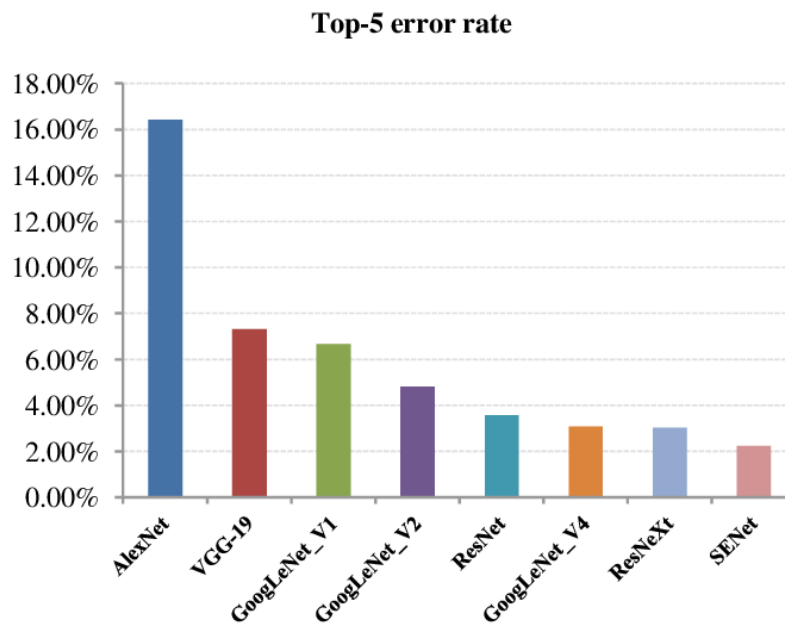


Figure A.2: Errors rate of the CNN variants introduced during the years.

the so-called ResNet was introduced leading to a consistent improvement in the research. On the contrary of the previous configurations that were seeing a reducing in performances after increasing more than a certain number the number of layers, the Resnet keep improving adding more layer. Its architecture sees the presence of residual connections: the input and of one layer could be used to compute the output of some layer after (Fig. A.3). This characteristic allows in some sense to filter weights in the activation phase according to relevance. There exist several variants of the ResNet: ResNet18, ResNet34, ResNet50, ResNet101, ResNet152; they all change in the total number of layers and in filters' shape, as represented, in order from left to right, in Fig. A.4.

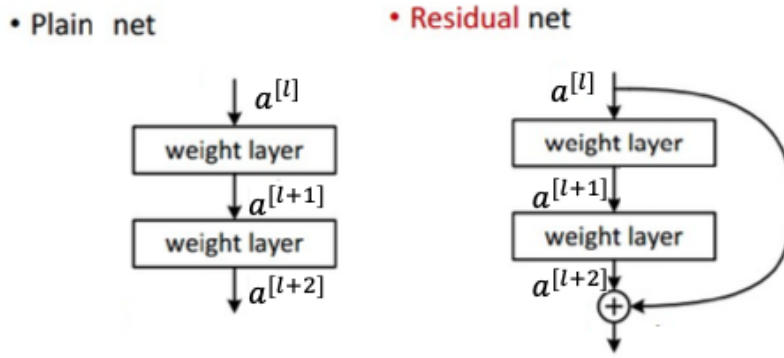


Figure A.3: Plain net VS Residual net

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				

Figure A.4: Comparison between the different type of networks. In order from left to right the architectures of the following are described: ResNet18, ResNet34, ResNet50, ResNet101, ResNet152

A.2 Recurrent Neural Networks

RNNs are the other big branch of deep networks that differs completely from CNN, because they works with sequence data. Their layers have loops inside them, which allow the information to persist. One of the field in which Recurrent Neural Networks (RNN) is used is the text processing: their ability is to look at meaningful connections and relationship between data over time. To better understand the concept of ‘loops’, it is possible to imagine to unroll them: this will result in a sequence of states, each of them characterized by an input vector an hidden state vector (which summarize the state of the neuron) and an output vector; each state is associated to a specific time point and what is crucial to see is the relationship between them: the output of a state, is not influenced only by its corresponding input vector, but also from the precious vector state.

These kind of networks offer a lot of flexibility because they can accept as input and give as output different combination between single vector or multiple vectors, in relation to the task that has to be addressed. The Fig. A.6 explains well the different configurations.

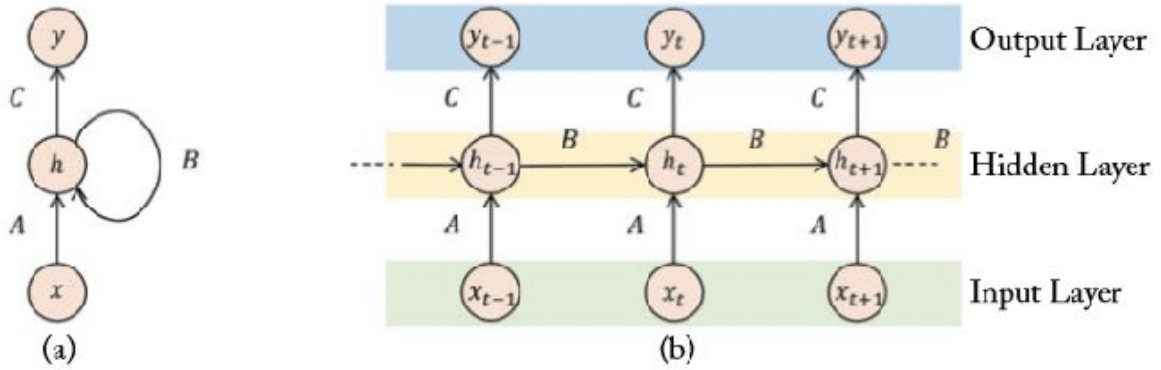


Figure A.5: Schematics and interpretation of a RNN. (a): the "rolled" configuration. (b): the "unrolled" version.

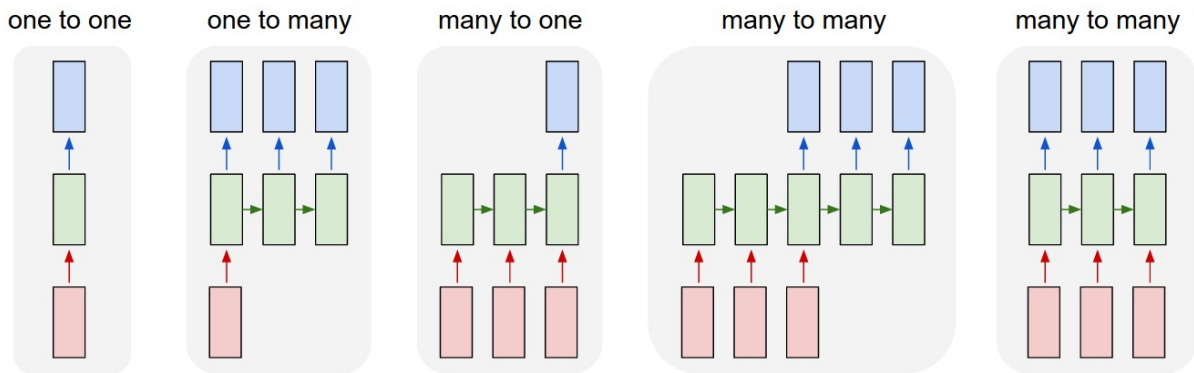


Figure A.6: Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN state. From left to right: (1) Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). (2) Sequence output (e.g. image captioning takes an image and outputs a sentence of words). (3) Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). (4) Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). (5) Synced sequence input and output (e.g. video classification where each frame of the label has to be labeled). Notice that in every case there are no pre-specified constraints on the lengths of sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

Taking the most simple configuration also called Vanilla RNN (one-to-one), it is interesting to analyze how it works. As mentioned before, RNNs are characterized by a state, which is not more than a function defined as:

$$h_t = f_w(h_{t-1}, x_t) \quad (\text{A.15})$$

where h_{t-1} is the previous state and x_t is the current input. The function is recalled as f_w because it is defined by some weights (e.g. $h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$), which will change during the network training. The state h_t is needed to evaluate the output y_t :

$$y_t = W_{hy}h_t \quad (\text{A.16})$$

where W_{hy} is the weight matrix that relates the state h_t and the input x_t with the output y_t . The training phase consists in initializing at the beginning the weight matrices randomly, then

updating their values aiming at reducing the loss with the true labels; the process is possible thanks to the back propagation of the error.

A.2.1 RNN variants: the LSTM

One of the main problems of RNNs is their improper ability to 'remember' big amounts of data; this usually brings to exploding or vanishing gradients (the back propagated error), thus to unsuccessful results. Moreover, they lack in learning long-dependencies. To overcome this problem, the LSTM (Long Short Term Memory) variant has been introduced; if in a standard RNN the repeating module has a very simple structure, in LSTM it includes four modules: the cell state and three gates, responsible for protecting and controlling the cell state. The gates are made of sigmoid function and a pointwise multiplication operation and the regulate the way in which information pass through the cell states.

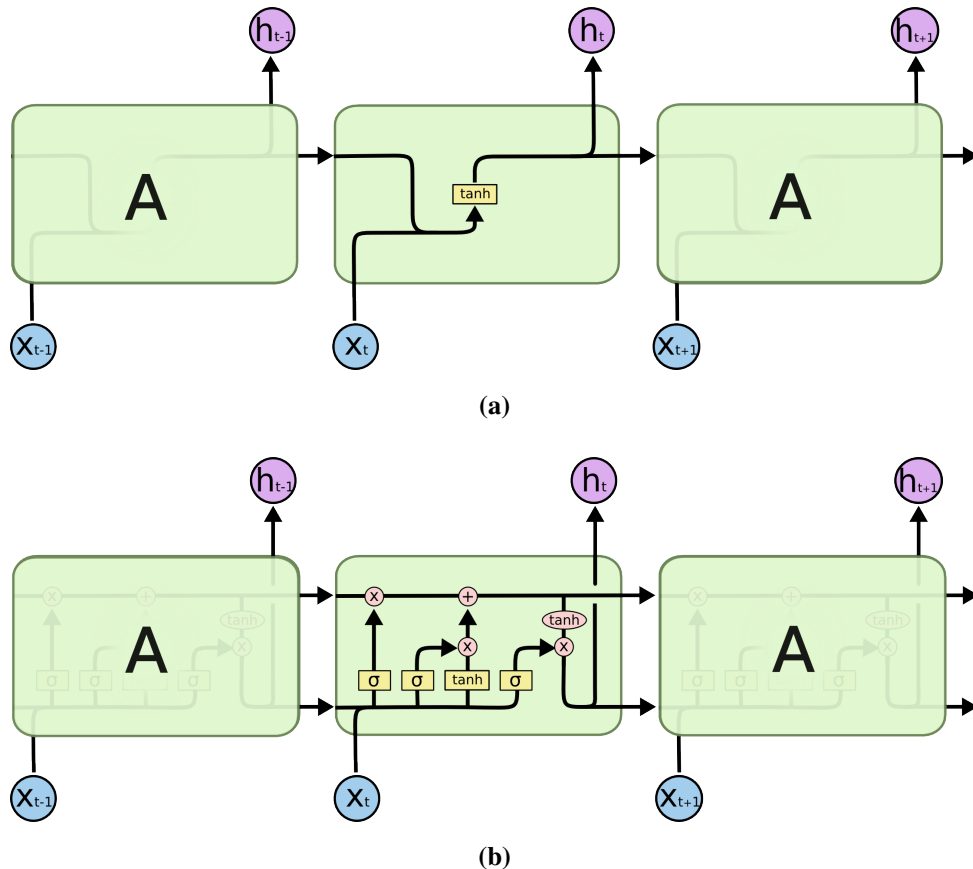


Figure A.7: (a) Repeating module of a simple RNN. (b) Repeating module of the LSTM.

The gates are three and their functions are described in Tab. A.2.

Gate	Function	Functionality
Forget gate	$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f)$	It decides what information have to be thrown away from the cell state.
Medium gate	(a) $i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i)$ (b) $\check{C}_t = \tanh(W_C * [h_{t-1}, x_t] + b_C)$	With two layers, it decides new things to remember. The (a) is the input gate layer, which, with a sigmoid function, decides the values that have to be updated; the (b) is the Tanh layer, that creates a vector of new candidate values that could be added to the cell state. At this point, the new cell state is evaluated with $C_t = f_t * C_{t-1} + i_t * \check{C}_t$.
Output gate	(c) $o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o)$ (d) $h_t = o_t * \tanh(C_t)$	It decides which parts of the cell state will be included in the output (c). It finally update the hidden state h_t (d).

Table A.2: Explanation of the three controlling gates.

Appendix B

Conventional Machine Learning approaches

Conventional machine learning approaches are all characterized by the necessity of processing the signal and of evaluating hand-crafted features. Thus, given a number N of extracted features, each samples is defined in a N -dimensionality feature space. Here are described the underlying functioning of the types of classifier used in this work.

B.1 KNN

K-Nearest Neighbors is one of the most simple classifier of the classical machine learning approaches. It does not use a specific model, but it works with the concept of similarity between samples. Given a number K , it looks at the classes of the K nearest neighbors for each analyzed sample, and vote the most frequent one. Despite the absence of a model, two parameters can influence the classifier performances:

- **K**: it is the number of neighbors to consider. Typically, lower values tend to increase the probability of overfitting, while higher values increase the bias.
- **Distance**. Considering the K nearest neighbors implies looking at the samples that are minimizing the distance with the one originally considered. The distance between two samples $x, y \in \mathbb{R}^N$ can be evaluated using different metrics:

– *Euclidean distance*:

$$d_e(x, y) = \sqrt{\sum_{i=1}^N (x_i - y_i)^2} \quad (\text{B.1})$$

– Manhattan distance:

$$d_m(x, y) = \sum_{i=1}^N |x_i - y_i| \quad (\text{B.2})$$

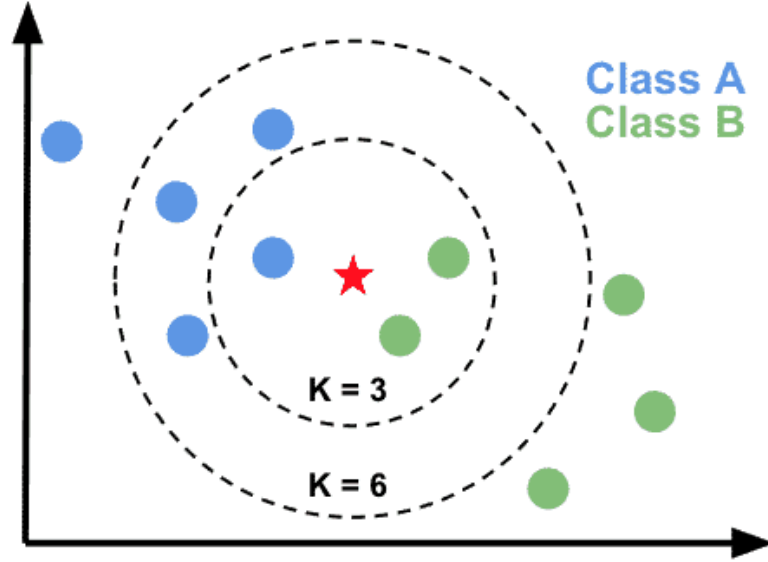


Figure B.1: KNN Schematics in a 2D feature space. The red star represents the analyzed sample.

B.2 Decision Tree

Decision Trees (DT) are classifiers typically appreciated in the biomedical domain because they well-represent the human decision making. Starting from a set containing the whole data and called root node, they perform subsequent splits with the objective of creating each time smaller subsets with lower impurity. The final subsets are called leaf nodes.

When a split has to be made, the classifier take a feature and compare its value to a defined threshold. The procedure is repeated for each feature and several thresholds are evaluated, until the optimal splitting criterion is reached. More precisely, the best split is reached when the change Δ in impurity is maximized:

$$\Delta = i(t)_P - [p_L i(t)_L + p_R i(t)_R] \quad (\text{B.3})$$

where $(i(t)_P)$ is the parent impurity and $i(t)_L, i(t)_R$ are the children impurity, with p_L and p_R as their splitting percentages. The impurity represents the similarity of the samples within a node, and it can be computed using different measures, like the *Training Error*, the *Gini index* and the *Cross Entropy*.

When a DT classifier is trained, the stopping criterion must be defined. It can be based on:

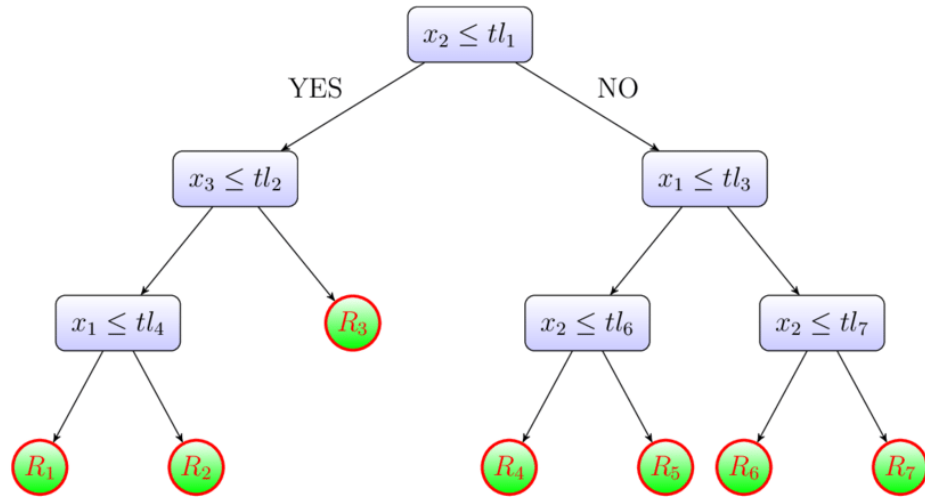


Figure B.2: Schematics of Decision Tree. tl_1, tl_2, tl_3, \dots are the thresholds. x_1, x_2, x_3, \dots are the features. R_1, R_2, R_3, \dots are the final leaf nodes.

- **Minimum samples per leaf:** minimum samples required in the future child node to perform the split.
- **Maximum depth.**
- **Minimum samples per split:** minimum samples required in a node to perform a split.

One of the problem of the decision tree is that it easily overfits data. This is why a common procedure is to perform pruning: exploiting the Cross Validation procedure, it evaluates the performances of the classifier with decreasing depth. Typically, a pruned tree has an higher predictive power, even if they are less accurate over the training set.

B.3 Support Vector Machine

Support Vector Machine (SVM) is a classifier that looks for an hyperlane that divides the samples in different regions of the space in relation of their class. It is originally defined for a binary classification problem.

The idea of SVM is to optimally separate the data with an hyperplane by maximizing the margine, which is the distance between the hyperplane itself and the clouds of data (of the different classes). This hyperplane is defined by a set of weights $\mathbf{w} \in \mathbb{R}^m$ and a bias parameter b . Given (\mathbf{w}, b) , the predictions are made with the formula:

$$y_i = \text{sign}(\mathbf{w}^T \mathbf{x}_i + b) \tag{B.4}$$

When data are separable, the following conditions can be satisfied:

$$y_i * (\mathbf{w}^T \mathbf{x}_i + b) > 0 \quad \forall i \quad (\text{B.5})$$

If not, the model training consists on minimizing the Hinge Loss, defined as:

$$L^{hinge}(\mathbf{w}, b) = \frac{1}{n} \sum \max(0, 1 - y_i * (\mathbf{w}^T \mathbf{x}_i + b)) \quad (\text{B.6})$$

To increase the generalization ability, a penalty on $\|\mathbf{w}\|$ is added; in this way, the overall cost function is:

$$L^{SVM} = L^{hinge}(\mathbf{w}, b) + \lambda \|\mathbf{w}\| \quad (\text{B.7})$$

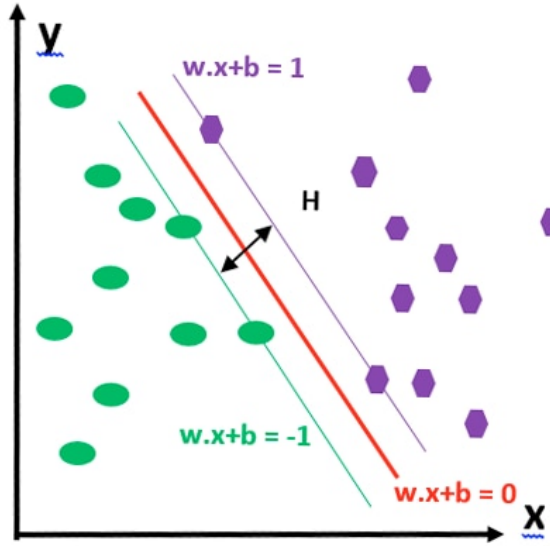


Figure B.3: Schematics of svm algorithm

When data are not linearly separable, an embedding function ψ is considered, such that:

$$\psi : \chi \rightarrow F \quad (\text{B.8})$$

χ is the domain set, and $F = \mathbb{R}^d$ is the feature space. Considering the embedding function ψ , the loss function become:

$$\frac{1}{n} \sum \max(0, 1 - y_i * (\mathbf{w}^T \psi(\mathbf{x}_i))) + \lambda \|\mathbf{w}\| \quad (\text{B.9})$$

Since working with the embedding function could be too much complex, it is preferred to define a kernel K , such that:

$$\mathbf{w}^T \psi(\mathbf{x}_i) = \sum \alpha_i K(\mathbf{x}_i, \mathbf{x}) \quad (\text{B.10})$$

There are several kernels:

- **Polynomial kernel:**

$$K(\mathbf{x}, \mathbf{x}') = (1 + \langle \mathbf{x}, \mathbf{x}' \rangle)^k \quad (\text{B.11})$$

K is the degree of the polynomial.

- **Gaussian kernel (or Radial Basis Function):**

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma}\right) \quad (\text{B.12})$$

- **Sigmoidal kernel:**

$$K(\mathbf{x}, \mathbf{x}') = \tanh(\mathbf{x}^T \mathbf{x}' + a) \quad (\text{B.13})$$

In Fig. B.4 it is possible to see the differences in using the kernels previously presented.

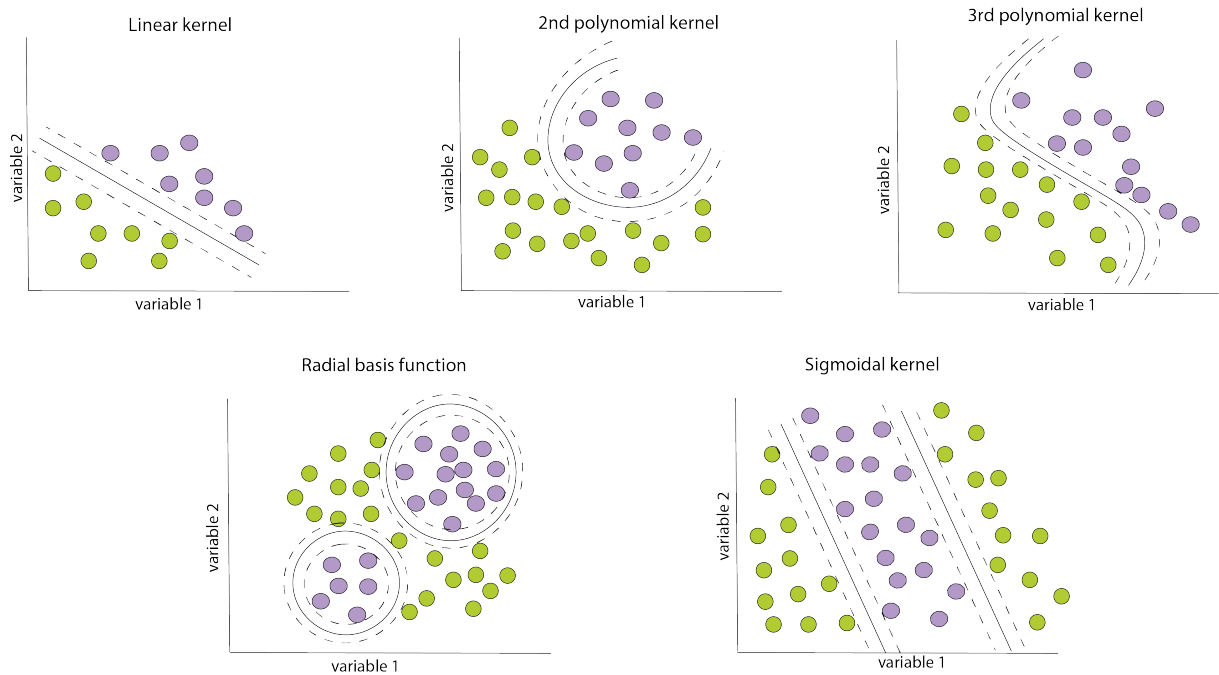


Figure B.4: Schematics of the different kernel effects over the creation of the hyperplane. The two colors represent the two labels of the samples.