Alma Mater Studiorum · Università di Bologna

Scuola di Scienze
Dipartimento di Fisica e Astronomia
Corso di Laurea Magistrale in Fisica

sEMG-based Hand Gesture Recognition with Deep Learning

Relatore:
Prof. Daniel Remondini

Presentata da:
Marcello Zanghieri

Correlatori:
Dott. Simone Benatti
Dott. Francesco Conti
Dott. Manuele Rusci

Anno Accademico 2017/2018

Abstract

Hand gesture recognition based on surface electromyographic (sEMG) signals is a promising approach for the development of Human-Machine Interfaces (HMIs) with a natural control, such as intuitive robot interfaces or poly-articulated prostheses. However, real-world applications are limited by reliability problems due to motion artifacts, postural and temporal variability, and sensor re-positioning.

This master thesis is the first application of deep learning on the Unibo-INAIL dataset, the first public sEMG dataset exploring the variability between subjects, sessions and arm postures, by collecting data over 8 sessions of each of 7 able-bodied subjects executing 6 hand gestures in 4 arm postures. In the most recent studies, the variability is addressed with training strategies based on training set composition, which improve inter-posture and inter-day generalization of classical (i.e. non-deep) machine learning classifiers, among which the RBF-kernel SVM yields the highest accuracy.

The deep architecture realized in this work is a 1d-CNN implemented in Pytorch, inspired by a 2d-CNN reported to perform well on other public benchmark databases. On this 1d-CNN, various training strategies based on training set composition were implemented and tested.

Multi-session training proves to yield higher inter-session validation accuracies than single-session training. Two-posture training proves to be the best postural training (proving the benefit of training on more than one posture), and yields 81.2% inter-posture test accuracy. Five-day training proves to be the best multi-day training, and yields 75.9% inter-day test accuracy. All results are close to the baseline. Moreover, the results of multi-day trainings highlight the phenomenon of user adaptation, indicating that training should also prioritize recent data.

Though not better than the baseline, the achieved classification accuracies rightfully place the 1d-CNN among the candidates for further research.

Sommario

Il riconoscimento di movimenti della mano basato su segnali elettromiografici di superficie (sEMG) è un approccio promettente per lo sviluppo di interfacce uomo-macchina (HMI) a controllo naturale, quali interfacce robot intuitive o protesi poliarticolate. Tuttavia, le applicazioni *real-world* sono limitate da problemi di affidabilità dovuti ad artefatti di movimento, variabilità posturale e temporale, e riposizionamento dei sensori.

Questa tesi magistrale è la prima applicazione dell'apprendimento profondo (*deep learning*) al dataset Unibo-INAIL, il primo dataset sEMG pubblico che esplora la variabilità tra soggetti, sessioni e posture del braccio, raccogliendo dati da 8 sessioni per ciascuno di 7 soggetti integri che eseguono 6 movimenti della mano in 4 posture del braccio. Negli studi più recenti, la variabilità è affrontata con strategie di addestramento basate sulla composizione del *training set*, che migliorano la generalizzazione inter-postura e inter-giorno di classificatori basati sull'apprendimento automatico (*machine learning*) classico (ossia non profondo), fra i quali la RBF-kernel SVM dà l'accuratezza più alta.

L'architettura profonda realizzata in questo lavoro è una 1d-CNN implementata in PyTorch, ispirata a una 2d-CNN con buone prestazioni dimostrate su altri dataset pubblici di riferimento. Su questa 1d-CNN, si implementano e testano varie strategie di addestramento basate sulla composizione del *training set*.

L'addestramento multi-sessione si dimostra produrre accuratezze di validazione intersessione più alte rispetto all'addestramento mono-sessione. L'addestramento bi-postura si dimostra la miglior strategia posturale (dimostrando il beneficio di addestrare su più di una postura), e produce un'accuratezza di test inter-postura dell'81.2%. L'addestramento su cinque giorni si dimostra la miglior strategia multi-giorno, e produce un'accuratezza di test inter-giorno del 75.9%. Tutti i risultati sono vicini alla *baseline*. Inoltre, i risultati degli addestramenti multi-giorno evidenziano il fenomeno dell'adattamento dell'utente, indicando che l'addestramento deve privilegiare dati recenti.

Benché non migliori della *baseline*, le accuratezze ottenute pongono a buon diritto la 1d-CNN tra i modelli candidati per la ricerca futura.

Contents

Chapter 1

Introduction

Hand gesture recognition based on electromyographic (EMG) signals is an innovative approach for the development of human-computer interaction, the vast field whose aim is to implement human-computer interfaces (HMIs) and intuitive interaction devices. The research in this field is motivated by the need for intelligent devices able to extract information from data coming from sensors, operating in real time and under significant power, size and cost constraints. The wide range of applications of HMIs with EMG-based intuitive control includes (but is not limited to) robot interaction and industrial robot control, game or mobile interfaces, interactions for virtual environments, sign language recognition, rehabilitation, and control of poly-articulated prostheses [1, 2, 3, 4, 5].

The electromyographic (EMG) signal is the biopotential generated by the ionic flow through the membrane of the muscular fibers during contraction, and is a major index of the muscular activity. EMG data can be acquired either with invasive or non-invasive measuring instruments. Invasive methods employ wire or needle electrodes, which penetrate the skin to reach the muscle of interest. On the contrary, surface electromyography (sEMG) is a non-invasive technique that uses surface electrodes applied on the surface of the skin [6]. In the HMI field, building gesture recognition on the analysis of sEMG signals is one of the most promising approaches, since non-invasiveness is an essential requirement for many types of HMIs.

An open challenge in HMI design is the development of solutions based on a robust recognition approach. On the one hand, the implementation of devices showing high recognition capabilities in controlled environments raised industrial interest and led to the availability of commercial solutions based on the EMG-based interaction paradigm. On the other hand, in many real-world scenarios the adoption of EMG-based HMIs is still limited by reliability problems such as motion artifacts, postural and temporal variability, and issues caused by sensors re-positioning at each use.

Nowadays, research efforts focus on solving issues related to postural variability effects and long-term reliability. Such research efforts are being boosted by two factors. (1) The release of public EMG databses suitable for the analysis of variability, whose publication eases the creation of benchmarks for the research community; the Unibo-INAIL dataset studied in this master thesis was realized for this very purpose. (2) The increasing recourse to the deep learning, whose deep hierarchical approach, together with the entirely data-driven feature extraction, promises to speed up the search for effective representations able to empower the recognition models.

1.1 EMG-based HMIs and generalization issues

The Electromyogram (EMG) is the biopotential signal resulting from muscular activity, and is a major index thereof. It can be sensed by means of non-invasive surface electrodes, giving rise to the surface EMG (sEMG) signal. The processing of sEMG signals is a promising approach for the implementation of non-invasive EMG-based Human-Machine Interfaces.

However, the current state-of-the-art has to cope with challenging issues. The sEMG signal is severely affected by many factors, such as differences between subjects, fatigue, user adaptation and the variability introduced from the re-positioning of electrodes at each data collection session. These issues limit long-term use and reliability of the devices relying on EMG analysis.

In the machine learning framework, these variability factors can be modeled with the concept of data sources, i.e. data subsets coming from different distributions (the concept of multi-source data is defined in Subsection 4.1.3). Identifying multiple sources makes machine learning on EMG data a challenging task, where the ambition is to implement classifiers capable of good inter-source generalization, e.g. in inter-posture, inter-session or inter-subject scenarios. Up to now, the classification accuracy with Leave-One-Subject-Out cross-validation (LOSOCV) is still much lower than that attained with Within-Subject cross-validation (WSCV) [8].

1.2 Deep Learning revolution

At present, an increasingly important role in sEMG-based gesture recognition (and in human-computer interaction in general) is being played by deep learning. Existing deep learning architectures are mainly based on two kinds of architecture: the Convolutional Neural Network (CNN), able to capture spatial information of the signal, and the Recurrent Neural Network, which allows to exploit the sequential nature of the data. This master thesis is exclusively focused on the CNN architecture, because CNNs are (1) easier to train, and (2) more suitable for deployment in embedded hardware, hence making them more attractive for this work. On the other hand, it must be noted that RNNs are typically more accurate on time-domain data, so the choice between the two models is not clear cut.

In the framework of sEMG-based gesture recognition based on classical machine learning, the pipeline typically consists of data acquisition, data preprocessing, feature extraction, feature selection, model definition and inference [6, 9]. The reason behind the new deep learning perspective for sEMG-based gesture recognition is that it mitigates the strong need for feature extraction, feature selection and parameter tuning, which strongly rely on specific domain knowledge. The advantage of a deep architecture is its ability to incorporate feature learning: a consistent part of the traditional pipeline can be entrusted to the training of the algorithm, which has enough capacity to learn effective feature representations on its own. This mitigates the reliance on rigid combinations of preprocessing steps and precise sets of hopefully discriminative features.

The deep learning approach has already started to speed up the research in this field. In particular, the CNN-based sEMG gesture recognition has been studied by Atzori et al. in [10], achieving comparable performance with traditional methods on the Non-Invasive Advanced Prosthetics (NinaPro) database. Another CNN architecture with adaptive feature learning improving inter-subject generalization has been proposed by Park and Lee in [11]. Geng et al. [12, 13] presented a new CNN architecture for instantaneous sEMG images on three sEMG benchmark datasets. Du et al. [14] designed a semi-supervised deep CNN which also exploits the auxiliary information of a data glove: classification performance is improved by training the auxiliary task of regression of glove signal.

1.3 Purpose of this master thesis

The purpose of this master thesis is to apply deep learning for the first time to the Unibo-INAIL dataset, the first public dataset of surface electromyographic signals (sEMG) exploring the impact of combined postural and temporal variabilities on myoelectric hand gesture recognition [15]. In particular, the deep learning architecture used is a one-dimensional Convolutional Neural Network (1d-CNN), which performs convolutions over the time dimension.

This is a novel approach since the application of deep learning yields new knowledge about the performance of machine learning on this dataset, which was previously analysed only with algorithms belonging to classical (i.e. non-deep) machine learning, applied only to instantaneous signal values. The CNN architecture used in this work is an adaptation of the 2d-CNN module of the hybrid CNN-RNN architecture proposed by Hu et al. in [9]. The evaluation of performance is made in such a way to directly compare deep learning and classical machine learning on each of the intra- and inter-session scenarios available as baseline for the various training and validation sets compositions.

At the dataset's state-of-the-art, the variability is addressed with training strategies that improve inter-posture and inter-session generalization of classical (i.e. non-deep) machine learning classifiers, among which the RBF-kernel SVM yields the highest accuracy. The deep model implemented and tested is a CNN architecture reported to perform well on other public benchmark databases. On this CNN, the state-of-the-art training strategies are implemented and tested.

1.4 Thesis structure

The remainder of this master thesis is structured as follows:

- Chapter 2 introduces the fundamentals of surface electromyography (from muscular activation potential to signal collection at the electrodes), compares the classical machine learning framework and the deep learning framework of sEMG-based hand gesture recognition, and summarizes the state of the art regarding the classification robustness against the signal variability factors (both in general and on the Unibo-INAIL dataset);

- Chapter 3 introduces the fundamentals of deep learning, of convolutional neural networks and of the architectural features used in this master thesis;
- Chapter 4 illustrates the materials and methods of this work: the Unibo-INAIL dataset, the adopted preprocessing and pipeline, the CNN architecture used (together with all the training settings), and the training strategies implemented and tested;
- Chapter 5 describes the scripts developed and how they exploit the main packages of the open source deep learning platform PyTorch;
- Chapter 6 presents the results obtained for the various training strategies;
- Chapter 7 exposes the conclusions and outlines the future work.

Chapter 2

Surface Electromyography and sEMG-based gesture recognition

Surface Electromyography is the field that studies the sensing, elaboration and uses of the surface-electromyographic (sEMG) signal, i.e. the electrical signal generated by contractions of skeletal muscles and sensed by non-invasive surface electrodes. For the development of Human-Machine Interfaces (HMIs), building gesture recognition on the analysis of sEMG signals is one of the most promising approaches, since for many applications non-invasiveness is an essential requirement.

This chapter is structured as follows:

- Section 2.1 introduces the fundamental concepts of surface electromyography, from the muscular activation potentials to the signal collection at the electrodes;
- Section 2.2 is devoted to sEMG-based gesture recognition: it provides an overview of the techniques and results of both the classical machine learning framework and the deep learning framework, and summarizes the state of the art regarding the search for classifiers that are robust against the signal variability factors, both in the general field and on the Unibo-INAIL dataset.

2.1 Surface Electromyography

Electromyography [16, 17] is the discipline that studies the detection, analysis, and applications of the electromyographic (EMG) signal, the electrical signal generated by muscles in correspondence with contractions. In particular, surface electromyography (sEMG) is a non-invasive technique for measuring and analysing the EMG signal of skeletal muscles by means of surface electrodes.

The electromyographic (EMG) signal is the bio-electric potential that arises from the current generated by the ionic flow through the membrane of the muscular fibers. It is therefore a major index of the muscular activity. During a muscular contraction, the depolarization of the tissue cell membrane, caused by the flow of Na^+ and K^+ ions, propagates along the muscle fibers. The origin of the potential is the electrical stimulus that starts from the central nervous system, and passes through the motor neurons (motoneurons) innervating the muscular tissue, giving rise to the Action Potentials (APs).

EMG data can be acquired either with invasive or non invasive measuring instruments. Invasive methods employ wire or needle electrodes, which penetrate the skin to reach the muscle of interest. On the contrary, surface electromyography (sEMG) is a non-invasive technique that uses surface electrodes that operate on the surface of the skin [6]. For many of human-machine interfaces, non-invasiveness is an indispensable requirement. In the sEMG setup, APs can be detected by means of an instrumentation amplifier with the positive and negative terminals connected to two metal plates positioned on the skin surface: the sEMG signal results from by the superposition of all the detected APs underlying the amplifier [15].

The EMG signal amplitude depends on the size and distance of the muscles underlying the electrodes, and typically ranges from 10 µV to 10 mV. The EMG signal bandwidth stays between 2 kHz. The noise sources affecting the EMG signal are many, the most important being motion artifacts, floating ground noise and crosstalk. A major source of interference is Power Line Intereference (PLI) [18], which is due to the capacitive coupling between the body and the surrounding electrical devices and power grid, and is common to most biomedical signals. Despite having nominal main frequency 50 Hz in Europe and 60 Hz in USA, PLI is non-stationary and can present variations in frequency (up to ± 2 Hz) and variations in amplitude (depending on instrumentation and environment), both mainly originating from the AC power system. Though beyond the scope of this master thesis, the development of filters and removal algorithms to reject PLI effectively is an active field of research.

2.1.1 Muscular activation potentials

Motor units are the basic functional units of a muscle, and muscle fibers are innervated motor units. When activated, motor units generate a Motor Unit Action Potential (MUAP). MUAPs can be generated in sequence by repeated activations, giving rise to MUAP Trains (MUAPTs). MUAPTs are possible as long as the muscle is able to generate force.

A convenient elementary model of MUAPs can be built with the tools of linear response theory [19]. In the linear response model, represented in Figure , a MUAPT can be fully described with two mathematical entities: the sequence of instants of the n pulses of the train $\{t_i\}_{i=1,\cdots,n}$, and the MUAP linear response function $h(t)$, accounting for the shape of the MUAP and also called the MUAP waveform. Interpulse intervals are defined as $\{\Delta_i = t_{i+1} - t_i\}_{i=1,\cdots,n-1}$, i.e. the time intervals between consecutive MUAPs. Each impulse is a Dirac delta impulse $\delta(t)$ (using dimensionless quantities), and the impulse train $\bar{\delta}(t)$ is given by their sum:

$$\bar{\delta}(t) = \sum_{i=1}^{n} \delta(t - t_i). \tag{2.1}$$

The expression $u(t)$ that describes the MUAPT as a function of time is obtained applying the kernel h on the impulse train $\bar{\delta}(t)$:

$$u = h * \bar{\delta} \tag{2.2}$$

which more explicitly is

$$u(t) = (h * \bar{\delta})(t) = \int_{-\infty}^{+\infty} \bar{\delta}(t - t')h(t')\mathrm{d}t' = \tag{2.3}$$

$$= \sum_{i=1}^{n} h(t - t_i) \tag{2.4}$$

where $*$ denotes convolution.

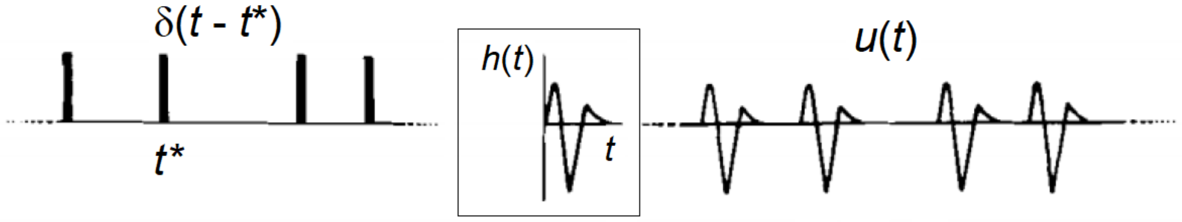


Figure 2.1: Linear response model of the Motor Unit Action Potential Train (MUAPT). Image adapted from [19].

More precisely, MUAP values depend also on the force F generated by the muscle: $u = u(t; F)$. The value of the EMG signal $m(t; F)$ sensed at time t depends on the generated force F as well, and is given by the sum of all the MUAPTs $u_{\text{unit}}(t)$ generated by the probed motor units:

$$m(t; F) = \sum_{\text{unit} \in U} u_{\text{unit}}(t; F) \tag{2.5}$$

where U is the set of the probed motor units. A comprehensive scheme following the potential from the AP to the recorded EMG signal is shown in Figure 2.2.

In both invasive and surface electromyography, the observed MUAP waveform depends on the relative position between the electrode and the active muscle fibers. This relative position may not be constant over time. The MUAP waveform and the EMG signal are also affected by any significant biochemical change in the muscle tissue: this is the case of muscle fatigue, which is a known source of variability for the EMG signal.

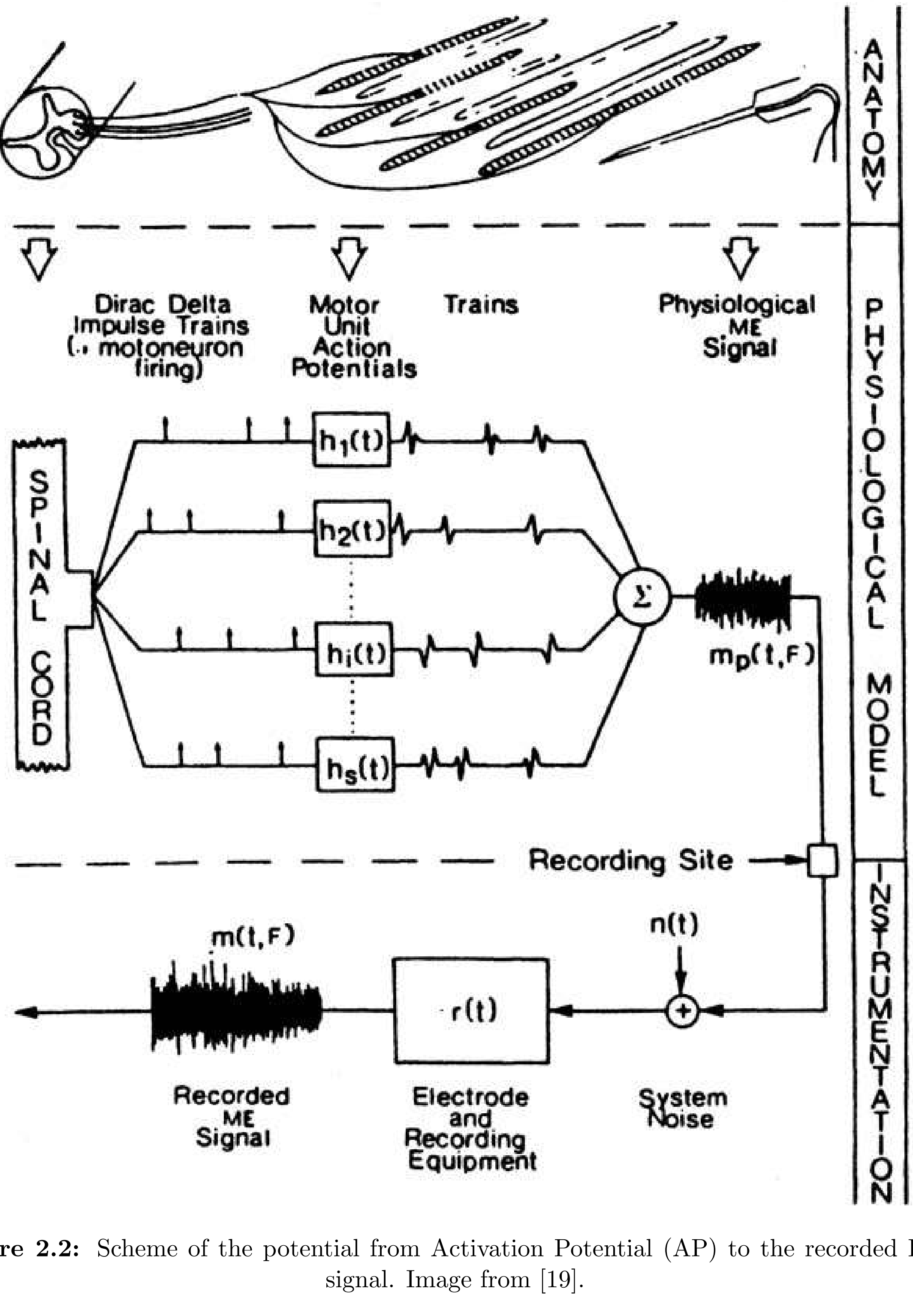


Figure 2.2: Scheme of the potential from Activation Potential (AP) to the recorded EMG signal. Image from [19].

2.2 sEMG-based gesture recognition

sEMG-based gesture recognition is a promising technique for the development of Human-Machine Interfaces (HMIs). On the one hand, sEMG has the virtue of being non-invasive, which is essential for many HMIs. On the other hand, entrusting recognition, i.e. signal classification, to automated learning is a successful method for circumventing the complexity of the task by removing the need of a complete physiological understanding of the underlying motor functions.

Automated learning has produced advances with a rich variety of approaches. Whereas the central goal of gesture recognition is signal classification, many other machine learning techniques have been successfully exploited as auxiliary tasks to assist classification at either training or evaluation time, increasing classification accuracy. Examples of auxiliary tasks (detailed in the following subsections) are force estimation by means of regression, and semi-supervised and unsupervised methods for calibration.

However, classical machine learning still requires field-specific knowledge, such as tested and established feature extracting procedures to maximize discriminative power, recourse to the suitable signal domain (time domain, frequency domain, or time-frequency domain), and empirical expertise about the portability of preprocessing and extraction procedures across datasets acquired with different experimental setups.

This need for field-specific knowledge can be overcome by migrating to deep learning algorithms, capable of autonomous feature learning, i.e. to learn good feature representations autonomously, provided sufficient data. The success of deep-learning models has even driven the ambition of reversing the perspective: once trained a satisfactory deep classifier, model explainability can be leveraged to shed light a posteriori on the feature representations, or even on the physiological understanding, lacked a priori. Steps toward demonstrating the feasibility of this approach have already been moved in the related field of electroencephalogram (EEG)-based gesture recognition for Brain-Computer Interfaces (BCIs): for instance, Nurse et al. in [20, 21], have shown correspondences between channel-time domain filters learned by the first layer of CNNs and known activation patterns of brain regions.

2.2.1 Classical Machine Learning approach

Classical Machine Learning (classical ML), also referred to as traditional or conventional machine learning, is the broad class of non-deep algorithms which includes k-Nearest Neighbors (kNN), Support Vector Machine (SVM), Linear Discriminant Analysis (LDA), and Random Forests (RF) (all with the related kernel methods), and Multi-Layer Perceptron (MLP) with one hidden layer. In the framework of sEMG-based gesture recognition based on classical ML, the pipeline typically consists of data acquisition, data preprocessing, feature extraction, feature selection, model definition, and inference [9, 6].

However, a major disadvantage of the classical ML pipeline is the strong reliance on domain-specific knowledge, needed for feature extraction, feature selection and parameter tuning. The reason why deep learning is appealing for sEMG-based gesture recognition is that it loosens these requirements, replacing feature extraction with feature learning incorporated in the algorithm training.

The general problem of finding good discriminative hand-crafted features is so hard that the quest for better ways to capture the temporal and frequency information of the signal has characterized decades of research. A clear overview of the typical sEMG features identified and used can be found in [6], and can be summarized by domain as follows:

- time domain: Root Mean Square (RMS), variance, Mean Absolute Value (MAV), Zero Crossings (ZC), Slope Sign Changes (SSC), waveform length, histogram;
- frequency domain: Short Time Fourier Transform (STFT), cepstral coefficients;
- time-frequency domain: Marginal Discrete Wavelet Transform (MDWT).

Two significant studies on sEMG classification with traditional ML techniques were made by Hugdings in [22] and Englehart and Hudgins [23], who studied the classification problem of 4 hand gestures and obtained a classification accuracy higher than 90% by working with 200 ms segments of 4 channel sEMG signals, extracting 5 time-domain features and feeding them to MLP and LDA classifiers, robustifying the assignment by applying a majority vote window to the predictions. Instead, Castellini et al. [53] worked on 3 types of grasp motions and achieved a 97.1% classification accuracy using the RMS value from 7 electrodes as the input to an SVM classifier.

The first successes in the classification of a large number of hand gestures were obtained by Kuzborskij et al. [25], using any of the proposed features in both time- and frequency-domain, fed to a SVM classifier with Radial Basis Function (RBF) kernel, and reaching a 70-80% accuracy on the 52 hand gestures of the 8-channel database Non-Invasive Advanced Prosthetics (NinaPro) presented by Atzori et al. in [26]. These results where improved by Atzori et al. [27], by considering linear combination of features and using a RF classifier resulting in an average accuracy of 75.32%; and further improved by Gijsberts et al. in [28], by evaluating different kernel classifiers jointly on EMG and acceleration signals, increasing classification accuracy by 5%.

2.2.2 Deep Learning Revolution

In recent years, sEMG-based gesture recognition has seen a progressive shift from traditional machine learning to deep learning. Deep Learning (DL) is the class of machine learning algorithms that differ from classical ML approaches in that feature extraction is part of the model definition, therefore obviating the need for hand-crafted features.

Existing deep learning architectures are mainly based on two kinds of architecture: the Convolutional Neural Network (CNN), successfully deployed for image classification due to its ability to capture spatial (but also temporal) information of the signal, and Recurrent Neural Network, which allows to exploit the sequential nature of the data and has had successes in speech recognition. The work of this master thesis is exclusively focused on the CNN architecture. Although deep algorithms themselves are not new, they are the most computationally demanding, so that deep classifiers gained attention only relatively recently, thanks to increased availability of data and powerful improvements in computing hardware [29].

The advantage of deep architectures is their ability to incorporate feature learning: a consistent part of the traditional pipeline can be entrusted to the training of the algorithm, which has enough capacity to learn good feature representations on its own. This mitigates the reliance on rigid combinations of preprocessing steps and precise sets of hopefully discriminative features.

The first end-to-end DL architecture was proposed by Park and Lee [11], who built a CNN-based model for the classification of six common hand movements resulting in a better classification accuracy compared to SVM. Atzori et al. [10] proposed a simple CNN architecture based on 5 blocks of convolutional and pooling layers to classify the 52 hand gestures from the NinaPro database [26], reaching classification accuracy comparable to those obtained with classical methods, though not higher than the best performance achieved on the same problem using a RF classifier.

Geng et al. [12] and Wei et al. [30] improved the results across various datasets by adding batch normalization and dropout (layer types explained in Subsections 3.3.1 and 3.3.3, respectively) to the model architecture, and by using a high-density electrode array, thus benefiting from the setup of High Density sEMG (HD-sEMG). Building the analysis on instantaneous EMG images [12] achieves a 89.3% accuracy on a set of 8 movements, going up to 99.0% when using majority voting over 40 ms signal windows. In [30] it is shown that for some movements a significant role is played by a small, group of muscles, and therefore a multi-stream CNN architecture is used that divides the inputs into smaller images, to be separately processed by convolutional layers before being merged with fully connected layers, reporting a increase in accuracy by 7.2% (from 77.8% to 85%).

Successful approaches also involve multi-modality data (also called sensor fusion): Du et al. [14] exploit the auxiliary information of a data glove to add semi-supervised auxiliary tasks (regression on glove signals) to a CNN, reporting improved classification accuracy.

2.2.3 State-of-the-art on the variability factors

Currently, the state-of-the-art of sEMG-based gesture recognition is coping with challenging issues. The sEMG signal is severely affected by many variability factors such as differences between subjects, fatigue, user adaptation and the variability introduced from the re-positioning the electrodes at each session of data collection. These issues limit long-term use and temporal reliability of the devices relying on sEMG analysis.

In the framework of machine learning, these factors of variability can be modeled with the concept of data sources, i.e. data subsets coming from different distributions (the concept of multi-source data is defined in Subsection 4.1.3). The Multi-source nature of the data makes machine learning on EMG data a challenging problem, where the ambition is to improve the inter-source generalization capability of classifiers. Examples of this are the inter-posture, inter-session and inter-subject scenarios. Up to know, for instance, the performance with Leave-One-Subject-Out cross-validation (LOSOCV) is still much lower than that reached with Within-Subject cross-validation (WSCV) on the main public benchmark databases [8].

Some studies have already dealt with inter-subject variability, resorting to recalibration techniques or model adaptation methods often moving from the technique of batch normalization (explained in Subsection 3.3.1). The network proposed in [31] takes as input downsampled spectrograms (i.e. time-frequency representations) of sEMG segments, and improvement is achieved by updating the network weights using the predictions of previous sessions corrected by majority voting. In [8] it is assumed that, while the weights of each layer of the network learn information useful for gesture discrimination, the mean and variance of the batch normalization layers store information related to discrimination between sessions/subjects. Moreover, a variation of batch normalization called Adaptive Batch Normalization (AdaBN) is used in [32]: only the normalization statistics are updated for each subject using a few unlabeled data, improving performance with respect to a model without adaptation.

In [33] transfer learning techniques are used to exploit inter-subject data learned by a pre-trained source network. In this architecture, for each subject a new network is instantiated with weighted connections to the source network, and predictions for a new subject are based both on previously learned information and subject-specific data. Doing so achieves an accuracy of 98.3% on 7 movements.

2.2.4 State-of-the-art on the Unibo-INAIL dataset

The state-of-the-art of sEMG-based hand gesture recognition on the Unibo-INAIL dataset is exposed by Milosevic et al. in [15]. The dataset itself represents the state of the art with regard to databases able to explore inter-subject, temporal, and postural variability, and is extensively described in Section 4.1.

Knowledge about the performance of classifiers on the dataset is limited to classical machine learning classifiers, namely Support Vector Machine (SVM), Neural Network (NN) with only one hidden layer, Random Forest (RF) and Linear Discriminant Analysis (LDA). However, these classifiers have been used to explore a large number of data partitions and training strategies, trying to optimize the generalization capability on new arm postures and new days.

The evaluated algorithms have similar recognition performance, higher than 90% (precise values depending on data partition and training strategy). The Radial Basis Function (RBF)-kernel SVM is at present the one achieving the highest accuracy, both intra-session and inter-session.

With regard to inter-session generalization, the baseline classification accuracy was higher than 90% for intra-posture and inter-day analysis, suffering a degradation of up to 20% when testing on data from different postures or days. This is consistent with the inter-session accuracy decline shown for other datasets (e.g. the 27% morning-to-afternoon decline reported for the NinaPro Database 6 by Palermo et al. [34]). The work showed that this inter-posture and inter-day accuracy decline is mitigated by training with combinations of data from multiple sessions. Moreover, results on temporal variability show a progressive user adaptation trend and indicate that (re)training strategies should prioritize the availability of recent data.

However, despite the variety of strategies and the number of results, the state of the art has two limitations. The first limitation is that, as said above, the state of the art on this dataset is limited to classical machine learning algorithms. The second limitation concerns data preprocessing: all the algorithms are applied on singled out, instantaneous signal values (i.e. 4-dimensional data points whose only dimension is channel number), so that the sequential nature of the EMG signal is not exploited at all. This master thesis addresses both these limitations.

Chapter 3

Deep Learning and Convolutional Neural Networks

Deep learning is the sub-field of machine learning which deals with deep neural networks, i.e. neural networks having more than one hidden layer. Deep learning is currently the fundamental approach for the development of artificial intelligence. One of the most important deep neural network architectures is the convolutional neural network, which is the model used in this work.

This chapter is structured as follows:

- Section 3.1 introduces neural networks and deep learning, contextualizing them with respect to machine learning and artificial intelligence;
- Section 3.2 provides the fundamentals of deep networks training, and illustrates some specific techniques and settings used in this work;
- Section 3.3 introduces convolutional neural networks, explaining the essential layer types (i.e. convolutional and fully connected) and also introducing the other particular layers used in this work.

3.1 Neural Networks and Deep Learning

Deep learning (DL) is the branch of Machine Learning (ML) which is nowadays the base of the research and applications of Artificial Intelligence (AI) [35], and in the last decade has produced vast advances in many fields such as speech recognition [36] and image recognition [37], cancer detection [38], self-driving vehicles [39], and playing complex games [40]. In some applications, DL models already exceed human performance.

The superior performance of DL models resides in the ability to incorporate automated data-driven feature extraction selection, i.e. the ability to extract high-level features from raw data by using statistical learning on large datasets. This approach produces effective representations of the inputs space, based on powerful discriminative features, in a different way from the earlier approaches based on non-deep ML, which are built on hand-crafted features or rules designed by expert researchers.

It is important to contextualize clearly the relationship DL has with AI and ML, as illustrated in Figure 3.1. With respect to the broad domain of AI, ML can be considered a vast sub-domain. Inside ML, is the area referred to as brain-inspired computation, whose interest is the developing of programs or algorithms whose basic functionality is inspired by natural brains and aims to emulate some aspects of how we currently understand the brain works [41]. Brain-inspired computing is divided into to main branches: spiking computing and DL. Spiking computing, which is beyond the scope of this work, takes inspiration from the fact that communication between neurons happens via spike-like pulses, with information not simply coded in spike's amplitude, but also in the timing of the spikes. In contrast, the branch of brain-inspired computing relevant here is Neural Networks (NNs), which are a well-known ML algorithm.

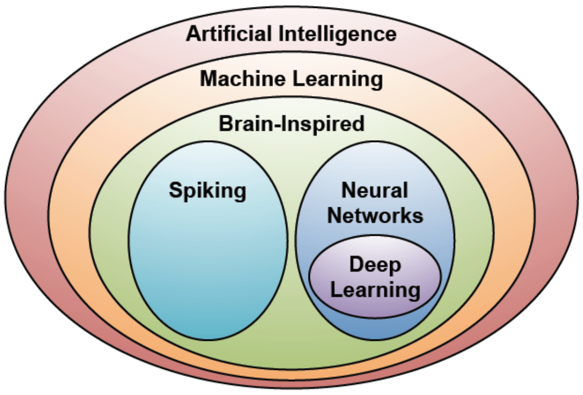


Figure 3.1: Deep Learning contextualized with respect to Artificial Intelligence and Machine Learning. Image from [41].

Neural networks are inspired by the fact that the signal processing performed by a neuron can be modeled as a weighted sum of the input activations, followed by a non-linear function with threshold and bias, which generates the neuron's output signal [42] as illustrated in Figure 3.2. By analogy, neural networks are built assembling units (also referred to as neurons) which apply a non-linear function to the weighted sum of the input values they receive.

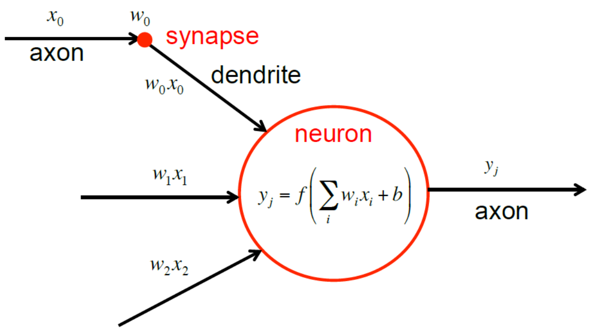


Figure 3.2: Model of the computations performed by a brain neuron: x_i are the input activations, w_i are the weights of the weighted sum, $f(\cdot)$ is the non-linear function, and b is the bias term. Image from [42].

Neural Networks, like the computations they execute, are structured as layers of units, and an example scheme is shown in Figure 3.3. The neurons belonging to the input layer receive the raw input values, and propagate them to the neurons of the middle layer, called the hidden layer. The weighted sums computed by one or more hidden layers ultimately reach the output layer, whose units compute the final output of the network. Taking the brain-inspired terminology further, the neuron outputs are referred to as activations, and the weights are sometimes called synapses. In addition to the network structure, Figure 3.3 also shows the computations made at each layer, which follow the formula

$$y_j = f\left(\sum_{i=1}^{3} W_{ij} x_i + b\right) = \tag{3.1}$$

$$= f\left(\mathbf{W}\mathbf{x} + b_j\right), \tag{3.2}$$

where $\mathbf{x}$ are the input activations, $\mathbf{W}$ are the weights, b_j are the bias terms, y_j are the output activations, and $f(\cdot)$ is the non-linear activation function.

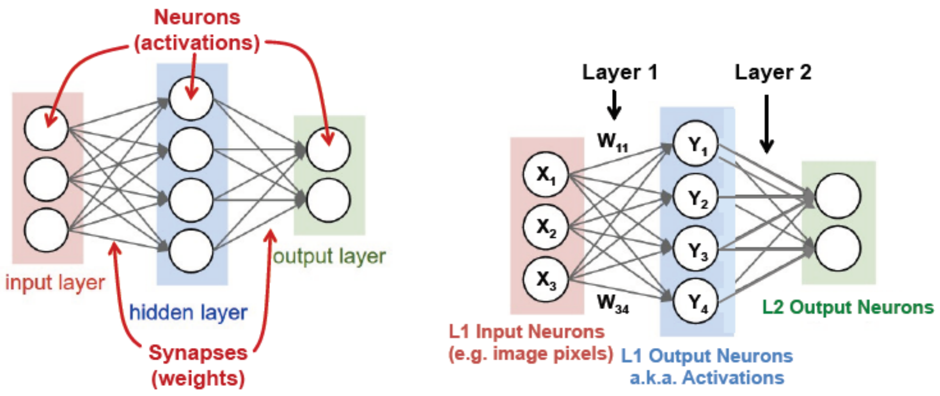


Figure 3.3: Scheme of a simple neural network's structure and terminology. Left: neurons and synapses. Right: weighted sum computation for each layer (bias term b omitted for simplicity). Image from [41].

Deep learning is the field of Neural Networks which focuses on Deep Neural Networks (DNNs), which are networks having more than one hidden layer, or, equivalently, more than three layers in total (typically, from five to more than a thousand [41]). DNNs have the ability to learn high-level features with more complexity and abstraction than shallow (i.e. non-deep) neural networks. For instance, in image processing, pixels of an image are fed to the first layer of a DNN, whose outputs can be interpreted as representing the presence of low-level features in the image, such as lines and edges. Subsequent layers progressively combine these features, eventually yielding a measure associated to the presence of higher level features (e.g. edges are combined into shapes, then into sets of shapes). As final output, the network produces an estimate of the probability that the highest-level features comprise a particular object or scene. This paradigm is referred to as deep feature hierarchy, and is what gives the DNNs the ability to obtain superior performance.

3.2 Training deep networks: gradient descent and back-propagation

In a DNN, like in machine learning algorithms in general, there is a basic program that does not change while learning to a desired task. For DNNs, the basic program is the structure of the functions implemented by the layers, and learning consists in determining the value of the network's weights and biases, through an optimization called training. Once trained, the network can execute its task by computing the output using the optimized weights and biases. Running the trained networks to evaluate inputs is referred to as inference.

The training scenario relevant in this work is the supervised training for a classification task. In classification tasks, trained DNNs receive input data (e.g. the pixels of an image) and return vector of scores, one for each class. The highest-score class is the one the network estimates as most probable. The main goal of DNN training is to optimize the weights and the bias values so as to maximize the score of the correct class and minimize the scores of the incorrect classes. In supervised learning, the correct class of the data the network is trained on is known. The dissimilarity between the ideal correct scores and the scores computed by the DNN (based on its current weights and biases) is called the loss L, which is the objective function of the training, i.e. function to minimize.

A widely used algorithm for weight optimization is a hill-climbing iterative optimization procedure called gradient descent. In gradient descent, at each iteration t the weights w_{ij} is updated by subtracting a multiple of the gradient of L with respect to the weights. Element-wise:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \alpha \frac{\partial L}{\partial w_{ij}} \tag{3.3}$$

where the multiplication factor α is called learning rate. Iterating reduces L.

In contrast with weights and biases, which are referred to as model parameters since they are the arguments of the objective function to minimize, the learning rate α is not involved in differentiation and hence is a training hyper-parameter, i.e. a constant that regulates the optimization without being affected by training. As the hyper-parameters describing the structure or training of any machine learning model, α can be tuned to the optimal value through cross-validation (which can be computationally expensive) or through a shorter preliminary analysis (as done in this work), with a further speed-up commonly yielded by following heuristics instead of grid search. The optimal α is typically small (some orders of magnitude below unity), but the its order of magnitude is strongly dependent on data, task, and model architecture. Hence the search must cover different orders of magnitude.

Finer techniques for tuning the learning rate are scheduling and recourse to per-parameter learning rates. (1) With scheduling, α is reduced over the iterations, typically with a stepwise or exponential decay, in order to adapt the tuning to different moments of the gradient descent. (2) On the other hand, introducing per-parameter learning rates allows to perform a customized tuning on different parameters or groups of parameters. Both techniques come at the cost of an increased number of combination to explore.

An efficient procedure for computing the partial derivatives of the loss is back-propagation, which derives from the chain rule of calculus and operates by passing values backwards through the network to compute how the loss is affected by each weight. Computation using back-propagation, illustrated in Figure 3.4, requires some steps used also for inference. To back-propagate through each layer, one has to: (1) compute the gradient of L relative to the weights from the layer inputs (i.e., the forward activations) and the gradient of L relative to the layer outputs; (2) compute the gradient of L relative to the layer inputs from the layer weights and the gradients of L relative to the layer outputs. It is worth to note that back-propagation requires intermediate activations to be preserved for the backward computation, so that training has increased storage requirements compared to inference.

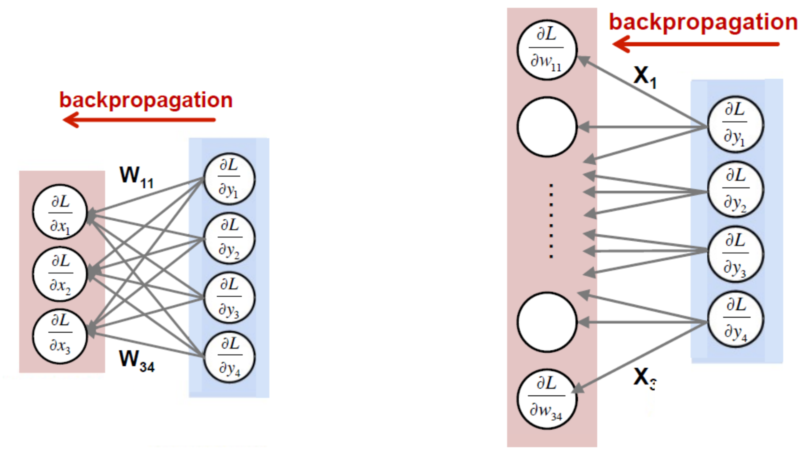


Figure 3.4: Example of back-propagation through a neural network: (1) (left) computation of the gradient of the loss relative to the layer inputs; (2) (right) computation of the gradient of the loss relative to the weights. Image from [41].

Another popular training method, orthogonal to the techniques exposed so on, is fine-tuning. In fine tuning, the weights of a trained network are available, and are used as a starting point for the iterative optimization. This practice results in faster training than using random initialization. Moreover, the scenario in which the weights are adjusted for a new dataset is the sub-field of machine learning defined transfer learning.

More specific training settings and techniques used in this work are explained in the next subsections.

3.2.1 Cross-entropy loss

When training neural networks, a convenient choice for the objective function (adopted also in this work) is the cross-entropy function. Originating in the field of probability theory and information theory, the main virtue of cross-entropy is that it leverages the soft assignments produced by the network, interpreting them as probabilities. Given a multiclass classification task on C classes, the loss for the observation x_i is computed by summing the losses due to each soft assignment $\hat{y}_{ic}$ with respect to the true label y_{ic} (i.e. the one-hot encoding):

$$\text{loss}(x_i) = -\sum_{c=1}^{C} y_{ic} \log \hat{y}_{ic}, \tag{3.4}$$

with arbitrary choice for the base; e.g., taking the logarithm in base 2 yields a result in bits, whereas choosing base e yields a result in nats. The overall loss L is then computed by averaging over all the items to classify. When working with an unbalanced dataset, it can be useful to weight the elements by class when taking the average.

3.2.2 Stochastic gradient descent with mini-batches

Stochastic Gradient Descent (SGD) is a variation of gradient descent which helps avoiding local minima during training. This technique does so introducing randomness in the optimization process by randomly partitioning the training set, referred to as the batch, into B equal-sized subsets called mini-batches. Then, gradient descent is performed using a single mini-batch per iteration, cycling over all the mini-batches. The sequence of B iterations that processes all the mini-batches exactly once is referred to as an epoch. Thus, during an epoch, all the data contribute to the optimization to the same amount. At the end of each epoch, the split into mini-batches is re-randomized, in order to prevent systematic bias generated by a particular order of comparison of the data.

It is important to note that the avoidance of local minima provided by SGD comes at the cost of introducing a new training hyper-parameter, the batch size b (or, equivalently, the number of batches B).

Since some sources reserve the name SGD for the extreme case $b = 1$, in the remainder of this work the term *SGD with mini-batches* will be used to avoid confusion.

3.2.3 L$_2$ regularization

The technique of L$_2$ regularization takes its name from the L$_2$-norm and is a method to counter overfitting. It consists in adding to the loss a multiple of the squared L$_2$-norm of the vector of all the parameters, and using the new formula L_{reg} as objective function. In formulas, for a deep neural network:

$$L_{\text{reg}} = L_0 + L_{\text{L}_2} = \tag{3.5}$$

$$= L_0 + \lambda_{\text{L}_2} \sum_{l=2}^{N_{\text{layers}}} \sum_{i=1}^{n_{l-1}} \sum_{j=1}^{n_l} |w_{ij}^{(l)}|^2, \tag{3.6}$$

where L_0 is the non-regularized loss, L_{L_2} is the regularization term, N_{layers} is the number of layers, n_l is the number of neurons in layer l, and $w_{ij}^{(l)}$ is the weight connecting neuron i of layer $l-1$ to neuron j of layer l. The factor λ_{L_2} is the training hyper-parameter regulating the amount of regularization enforced. Too low λ_{L_2} has no effect, and too high λ_{L_2} incurs underfitting. The optimal value can be determined via preliminary analysis or cross-validation, covering different orders of magnitude.

3.3 Convolutional Neural Networks

Deep networks have a vast variety of architectures and sizes, continuously evolving to increase performance. Convolutional neural networks are a successful class of architectures, which can be introduced only after surveying the strategies used to progressively reduce the storage and computation required by layers: sparsity, structured sparsity, weight sharing, and convolution.

Deep networks can be entirely composed of fully-connected (FC) layers, shown in

Figure 3.5, and in this case they are defined Multi-Layer Perceptrons (MLP). In a FC layer, all outputs units are connected to all inputs units, so that all output activations are computed with a weighted sum of all input activations. Although the FC configuration requires significant computation and storage, in many situations it is possible to zero some weights (thus removing the relative connections) without impacting performance. The resulting layer, also shown in Figure 3.5, is called sparsely connected layer.

Opposed to generic sparsity, structured sparsity is the configuration in which each output is only a function of a fixed-size window of inputs. Even further efficiency is gained when the computation of every output employs the same set of weights. This configuration is known as weight sharing, and strongly reduces the storage requirements for weights. A particular case of weight sharing arises when the computation is structured as a convolution, as shown in Figure 3.6: the weighted sum for each output activation is computed using only a narrow neighborhood of input activations (by zeroing the weights outside the neighborhood), and every output shares the same set of weights (i.e., the filter is space invariant). This gives rise to convolutional (CONV) layers, which are the characteristic building block of convolutional neural networks.

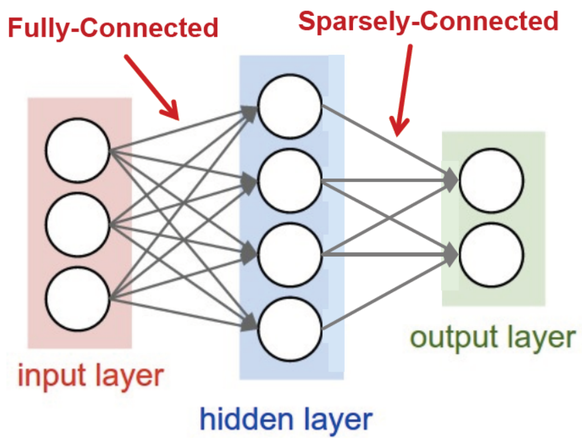


Figure 3.5: Fully conncted layer versus sparsely connecetd layer. Image from [41].

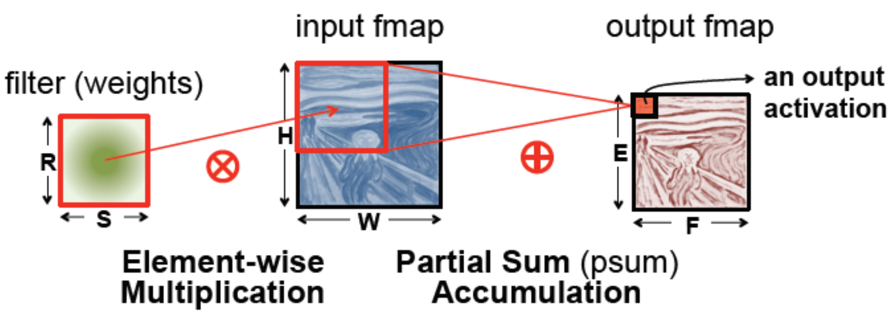


Figure 3.6: 2d convolution in traditional image processing. Image from [41].

Convolutional Neural Networks (CNNs) are a successful deep network architecture, composed of multiple CONV layers, as shown in Figure 3.7. In CNNs, each layer generates a progressively higher-level representation of the input data, referred to as feature map (fmap), extracting the essential information for the network's task. Modern CNNs have attained superior performance by implementing a very deep hierarchy of layers. CNN are widely used in a variety of applications including image understanding [37], speech recognition [43], robotics [44] and game play [40]. In this work, CNNs are applied to the task of classifying time windows of a 4-channel signal.

CNNs fall into the category of feed-forward networks. In a feed-forward network, all computations are executed as a sequence of operations taking place from one layer to the next one. A feed-forward network has therefore no memory, and the output for a given input is always identical irrespective of the history of the inputs fed previously.

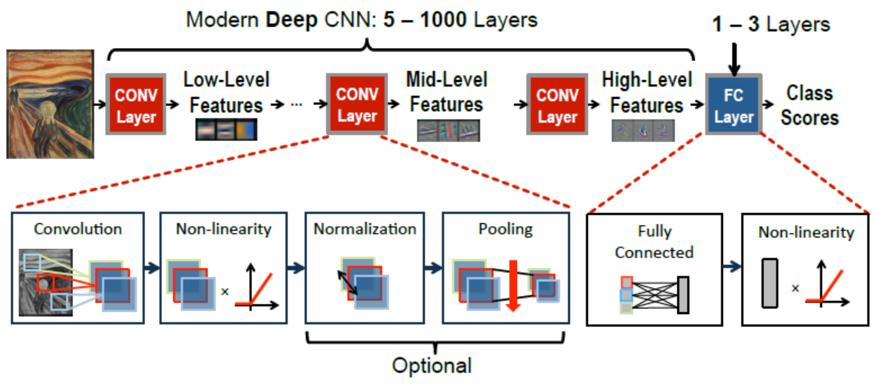


Figure 3.7: A convolutional neural network. Image from [41].

Each CONV layer is mainly constituted by high-dimensional convolutions, as shown in Figure 3.8. In this computation, the input activations of a layer have the structure of a set of 2d input feature maps (ifmaps), each referred to as a channel. Each channel is convolved with a distinct 2d filter from the stack of filters, one for each channel. The stack of 2d filters being a 3d structure, it is sometimes collectively called a 3d filter. The results of the convolutions at each point are summed across channels, and a 1d bias is optionally [45] added to the results. The result of this computation are the output activations constituting one channel of the output feature map (ofmap). Additional output channels can be created by applying additional 3d filters on the same input.

With the notation for shape parameters defined in Table 3.1, the computation executed by a CONV layer is described by the formula

$$\mathbf{O}[z][u][x][y] = \mathbf{B}[u] + \sum_{k=0}^{C-1}\sum_{i=0}^{S-1}\sum_{j=0}^{R-1} \mathbf{I}[z][k][Ux+i][Uy+j] * \mathbf{W}[u][k][i][j], \tag{3.7}$$

with

$$0 \leq z < N, \quad 0 \leq u < M, \quad 0 \leq x < F, \quad 0 \leq y < E, \tag{3.8}$$

$$E = \frac{H-R+U}{U}, \quad F = \frac{W-S+U}{U}, \tag{3.9}$$

where **O**, **I**, **W** and **B** are the matrices of ofmaps, ifmaps, filters and biases, respectively, U is a fixed stride size, and $*$ denotes discrete convolution. A graphical representation of this computation is shown in Figure 4.2 (where biases are omitted for simplicity).

To align the CNN terminology with the general DNN terminology, is it worth remarking that

- filters are composed of weights (corresponding to synapses in nature);
- input and output feature maps (ifmaps, ofmaps) are composed of activations of inputs and output neurons.

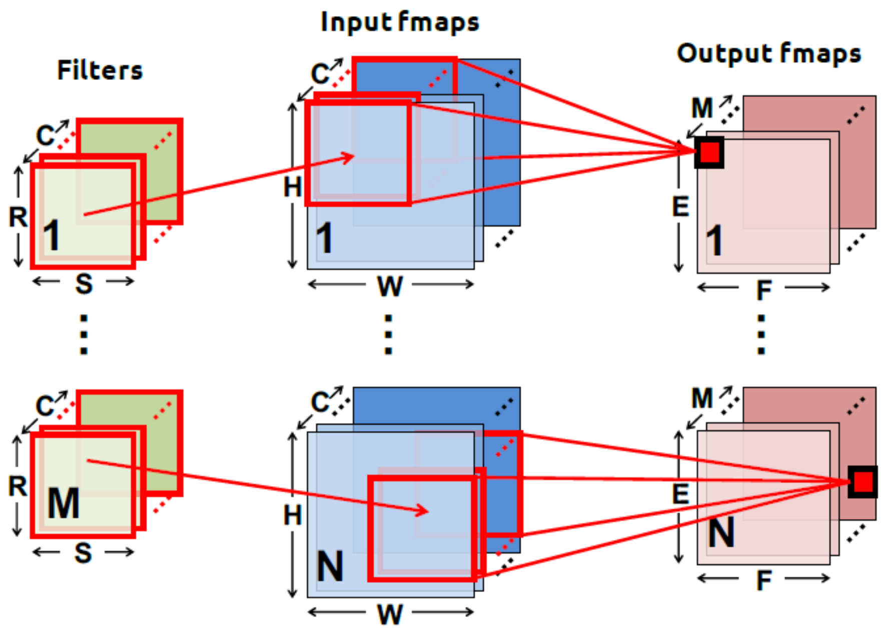


Figure 3.8: High-dimensional convolutions in CNNs. Image from [41].

Shape parameter	Meaning
N	batch-size of 3d maps
M	number of 3d filters / number of batch channels
C	number of ifmap/filter channels
H/W	ifmap plane heigth/width
R/S	filter plane heigth/width (= H or W in FC)
E/F	ofmap plane heigth/width (= 1 in FC)

Table 3.1: Shape parameters of a CONV/FC layer.

In the CNN used in this work, the CONV layers are CONV-1d layers, i.e. layers performing a 1-dimensional convolution. Though less common, CONV-1d layers follow the same principles as CONV-2d layers, but work with inputs having 1 dimension (plus channel number). The choice to use CONV-1d layers was made in order to act on the time dimension of the signal, while using the 4 sEMG channels as CNN input channels. In addition to CONV layers and FC layers, the CNN implemented in this work contains other elements, namely the rectified linear unit, batch normalization and dropout, whose mechanisms are explained in the following two subsections. These three kinds of intervention on activations are sometimes conceptualized as layers.

3.3.1 Batch-normalization

Controlling the input distribution across layers can speed up training and improve accuracy. Accordingly, the distribution of a layer's input activations (described by its mean μ and standard deviation σ) can be standardized to zero mean and unit standard deviation. Batch Normalization (BN) [46] is the technique in which the standardized activations are further scaled and shifted, undergoing the transformation

$$y = \frac{x - \mu}{\sqrt{\sigma^2 + \varepsilon}}\gamma + \beta, \tag{3.10}$$

where the parameters γ and β are learned from training, and ε is a small constant used to avoid numerical problems. Batch normalization is mostly applied between the CONV or FC layer and the non-linear activation function, and is usually turned off after training.

3.3.2 ReLU activation function

Non-linear activation functions are typically applied after each CONV or FC layer. The most common functions used to introduce non-linearity into a DNN are shown in Figure 3.9. Historically, the sigmoid and the hyperbolic tangent are the most conventional, while the Rectified Linear Unit (ReLU) has become common in the last years due to its simplicity and its ability to make training faster [47]. The leaky ReLU, parametric ReLU, and exponential LU are variations of the ReLU explored for increased accuracy. ReLU is mostly applied after the CONV or FC layer or after batch normalization (if present).

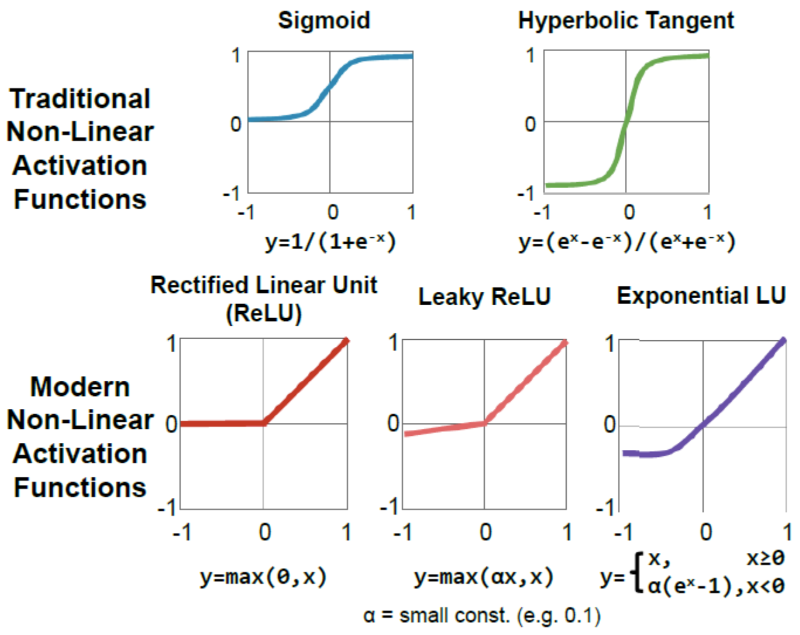


Figure 3.9: The most common non-linear activation functions. Image from [48].

3.3.3 Dropout

Dropout is a technique to improve accuracy by reducing overfitting [49, 50]. It works by randomly dropping units (and their connections) from the network during training. This prevents the phenomenon of units co-adaptation, forcing each neuron to learn a feature helpful for computing the correct output. In particular, the random dropout of the units of the interested layer during training is regulated by the drop probability p, for which a common value is $p = 0.5$. On every forward pass, each unit is zeroed out independently and randomly, drawing from the Bernoulli distribution parameterized by p. Moreover, the outputs are multiplied by $\frac{1}{1-p}$. Dropout is active only during training: during inference, the dropout layer is disabled (or, equivalently, the drop probability p is set to 0). Dropout is mostly applied immediately before the CONV or FC layer.

Chapter 4

Materials and Methods

This chapter explains all the materials and methods used in this master thesis. It is structured as follows:

- Section 4.1 exhaustively describes the Unibo-INAIL dataset;
- Section 4.2 details the implemented machine learning pipeline, preprocessing, CNN architecture and training settings;
- Section 4.3 defines the concept, essential in this work, of training strategy based on training set composition, and illustrates the training strategies used.

4.1 Unibo-INAIL dataset

The work of this master thesis is entirely focused on the Unibo-INAIL dataset. The Unibo-INAIL dataset is a surface electromyography (sEMG) dataset realized to explore the impact of arm posture and temporal variability (either alone or combined) on sEMG-based hand gesture recognition. The dataset was presented by Milosevic et al. in [15], and was built on the preliminary analysis carried out by Benatti et al. in [51]. This master thesis evaluates the performance of Convolutional Neural Networks (CNNs) trained on the dataset with all the training strategies described in [15], where the performance of classical machine learning algorithms is reported.

The data were acquired from 7 able-bodied (i.e. non-amputee) subjects performing 6 discrete hand gestures in 4 arm postures, repeated for 8 days, thus probing a total of 224 different data sources, each identified by three discrete indexes (subject, day and arm posture) and containing 6 classes, i.e. five hand gestures plus the rest position. This acquisition protocol, in addition to investigating the signal patterns associated to the 6 classes, allows to characterize the following sources of variability affecting signal and patterns:

- inter-posture variability: for each subject individually, keeping sensors on, different arm postures and wrist orientations cause variability in signal and patterns, due to differences in muscle activity and muscle position (since sensors are only fixed with respect to the skin);
- inter-day variability: for each subject individually, temporal variability of signal and patterns are mainly due to two factors:
  - sensor placement: from day to day, the sensors are removed and repositioned, causing differences in signal and patterns, due to the change of the relative position of the electrodes with respect to the muscles;
  - user adaptation: user adaptation (explained in Subsection 4.1.4) is the transient observed when the inter-day differences in gesture execution decrease over time, due to the fact that new users adapt to the repetitive exercise during the first days;
- inter-subject variability: signal and patterns are influenced by the anatomical variability between subjects (even if all able-bodied).

For the research on sEMG-based hand gesture recognition for reliable Human-Machine Interfaces (HMIs), the Unibo-INAIL dataset is a valuable ground for two reasons. The first reason is that the acquisition setup is based on commercial sensors, chosen so as to make the setup is easily repeatable and thus suitable for integrated HMI controllers [52]. The second reason is that the Unibo-INAIL dataset is the first public sEMG dataset to date to include both arm-postural and session variability, providing a realistic scenario for evaluating classification algorithms and training approaches.

4.1.1 Unibo-INAIL collaboration and motivation for the dataset

The Unibo-INAIL dataset is the results of a research project funded by the Istituto Nazionale per l'Assicurazione contro gli Infortuni sul Lavoro (INAIL), in which the University of Bologna (Unibo) was designated for assessing the feasibility of real-time control of poly-articulated hand prostheses by means of pattern recognition algorithms implemented on a microcontroller. The project was inspired by a previous study by Castellini et al. [53] on intuitive prosthesis control, which demonstrated that SVMs can recognize different muscle activation patterns with high precision. In particular, the SVMs classify gestures up to a precision of 95% and approximate the forces with an error of as little as 7% of the signal range, sample-by-sample at 25 Hz.

The first part of the project aimed to assess how accurate the recognition can be on diverse data produced in different conditions (i.e. intra-session scenario), and to verify whether the system was stable on different sessions (i.e. inter-session generalization). Although in prosthetics sensor (re)positioning is less relevant, since sensors are fixed to the prosthesis and thus much less mobile, this variability factor was included, planning future developments.

The second branch of the project involved the validation of the algorithms in real-time control scenario. The controller was implemented on a microcontroller, in a system in which the real-time, fresh data were acquired with the same embedded setup used for the first part of the project, in order to reproduce the previous system. The algorithm implementation was also made identical by using the open source machine learning library LIBSVM, which is implemented for both Matlab and C [54].

This master thesis continues the first part of the project, aiming to expand the results obtained for classical machine learning algorithms (SVM, shallow NN, RF and LDA) applied on instantaneous 4-channel signal values. This work extends the analysis to deep CNNs applied on 150 ms time windows of the 4-channel signal.

4.1.2 Outline of acquisition setup and experimental protocol

The acquisition setup, shown in Figure 4.1, was designed to be reliable and repeatable, with characteristics typical of prosthetic applications. However, since all the data are collected from able-bodied (i.e. non-amputee) subjects, the dataset is useful for any HMI application.

The acquisition setup is based on the Ottobock 13E200 pre-amplified single-ended sEMG electrode (Figure 4.2a), which is a commercial sensor. It amplifies and integrates the raw EMG signal to reach an output span of $0 - 3.3$V. The sensors have bandwidth spanning $90 - 450$Hz and integrate an analog notch filter to remove the noise due to Power Line Interference (PLI), i.e. the capacitive coupling between the subject and the surrounding electrical devices and power grid (detailed in Section 2.1). The output analog signals were acquired with a custom embedded board based on a microcontroller equipped with an internal 16-bit ADC. The digitalized signals were streamed via Bluetooth to a laptop, for storage and off-line data analysis.

The subjects involved were able-bodied (i.e. non amputee) males, 29.5 ± 12.2 years. During the acquisition the subjects worn an elastic armband with 4 Ottobock sensors placed on the forearm muscles involved in the selected movements (i.e. *extensor carpi ulnaris*, *extensor communis digitorum*, *flexor carpi radialis* and *flexor carpi ulnaris*) as shown in Figure 4.2b. Sensors were placed on the proximal third at 30 mm respectively on the left and on the right side of two axial lines ideally traced on the forearm. Each acquisition consisted in 10 repetitions of each hand gesture, with 3 second contractions interleaved by 3 seconds of muscular relaxation to be later labeled as rest gesture. After each acquisition, gesture segmentation was performed with a combination of manual inspection and an adaptive threshold to separate contractions from rest.

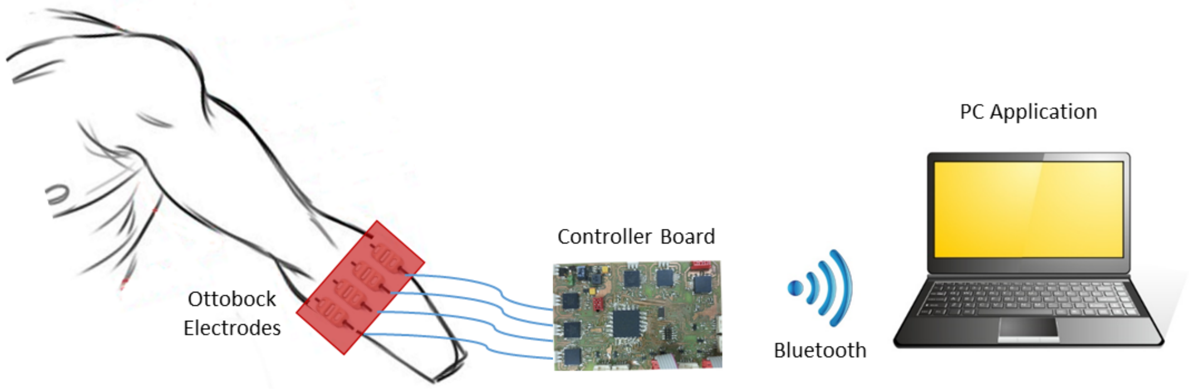
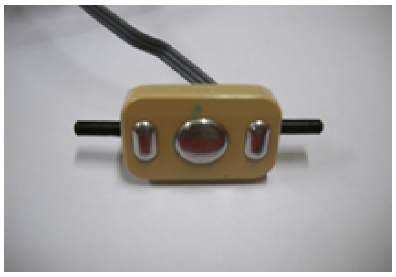
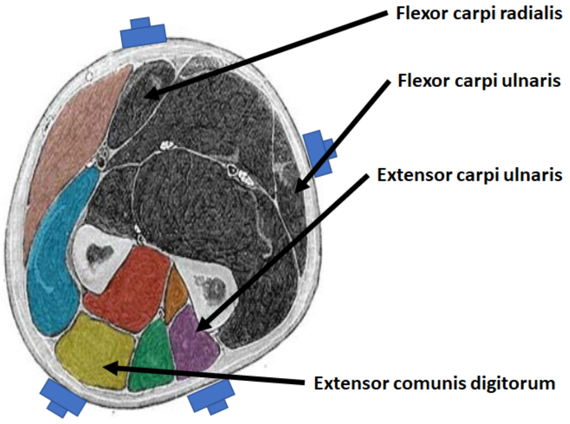


Figure 4.1: Acquisition setup of the Unibo-INAIL dataset. Image from [15].



(a) Ottobock sensor used to produce the Unibo-INAIL dataset. Image from [51].



(b) Forearm muscle cross-section and sensors placement. Image from [15].

Figure 4.2

4.1.3 Multi-source data structure

The fundamental property of the Unibo-INAIL dataset is that it contains 224 different data sources, since the data were collected for all the combinations obtained from 7 seven subjects, 8 days and 4 arm postures. Each subject-day-arm posture combination contains all hand gestures (i.e. the classes), each one repeated 10 times. The dataset can thus be regarded as a collection of 224 autonomous sub-datasets, for which the pattern to learn in order to assign signals to hand gestures is subjected to inter-subject, inter-day and inter-posture variability.

In machine learning, data having this structure are defined multi-source data. For the Unibo-INAIL dataset, each combination identified by a 3-ple subject-day-arm posture combination is a source:

$$\text{source} = (u, d, p) \quad \text{with} \begin{cases} u & = 1, \cdots, 7 \\ d & = 1, \cdots, 8 \\ p & = 1, \cdots, 4 \end{cases} \tag{4.1}$$

totalling 224 sources. Each source can be regarded as a smaller, complete dataset containing 10 repetitions of all the 6 gestures.

NOTE. In this work, the term *multi-source data* is used according to the meaning it has in statistical learning [7], where it refers to data-subsets having different but similar distributions. The intended meaning is not data coming from different types of sensors or modalities.

The subject-index u refers to the 7 subjects involved. Each one underwent data collection over 8 days: this is the day-index d of the data sources. Arm posture p is the third index, and the four collected arm postures are:

P1. proximal: the only one with the arm not fully extended, and the most common in EMG-based hand gesture recognition literature;

P2. distal;

P3. distal with the palm oriented down: the different wrist orientation aims to introduce additional difference compared to P2 and P4;

P4. distal with the arm lifted up by $45°$.

These arm postures are displayed in Figure 4.3.

To constitute the classes of the dataset, five common hand gestures used in daily life were chosen: power grip, two-fingers pinch grip, three-fingers pinch grip, pointing index and open hand. Rest position, recorded when muscles were relaxed between two subsequent movement repetitions, was also included as a class, totalling 6 classes:

G_0: rest position: including this class means addressing also the task of gesture detection, in addition to gesture recognition;

G_1: power grip;

G_2: two-fingers pinch grip;

G_3: three-fingers pinch grip;

G_4: pointing index;

G_5: open hand.

The five hand gestures $G_1, \cdots, G_5$ are shown in Figure 4.2, together with examples of 4-channel sEMG signal patterns of the gestures and the rest position G_0.
NOTE. It is of crucial importance, when handling the Unibo-INAIL dataset, not to mistake *arm posture* and *hand gesture*, since they are partitions acting at two different levels: the arm posture is the position of the arm in which all hand gestures (plus rest

position) were executed. Each combination of subject, day and arm posture contains 10 repetitions of all the five hand gestures plus rest, and can thus be regarded as a complete sub-dataset containing all the 6 classes.

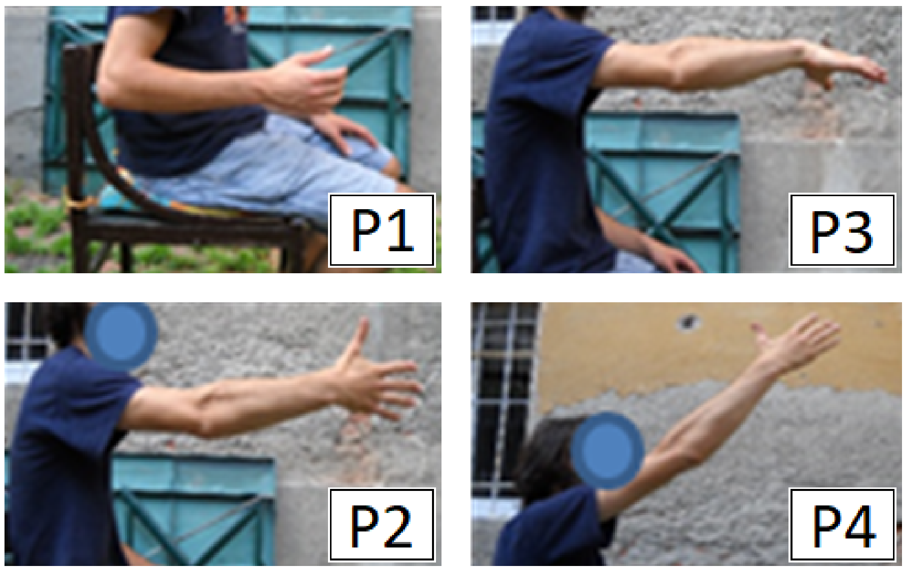


Figure 4.3: Arm postures of the Unibo-INAIL datase. P1: proximal; P2: distal; P3: distal with the palm oriented down; P4: distal with the arm lifted up by 45°. Image from [51].

4.1.4 User adaptation

User adaptation is the source of temporal variability which consists in a transient observed in the first days of many benchmarks datasets for sEMG-based hand gesture recognition [55, 56, 57]. The phenomenon consists in the fact that the inter-day differences in gesture execution decrease over time, due to the tendency of users to adapt to the repetitive exercise during the first days.

In literature, these inter-day differences are detected and measured by analysing how classification accuracy deteriorates when passing from intra-day validation to inter-day validation. With this method, classical machine leaning algorithms have already been able to highlight user adaptation also on the Unibo-INAIL dataset [15], which means that user the adaptation transient is not masked by the temporal variability caused by day-to-day sensor repositioning, which affects the signal on both the earlier and later days.

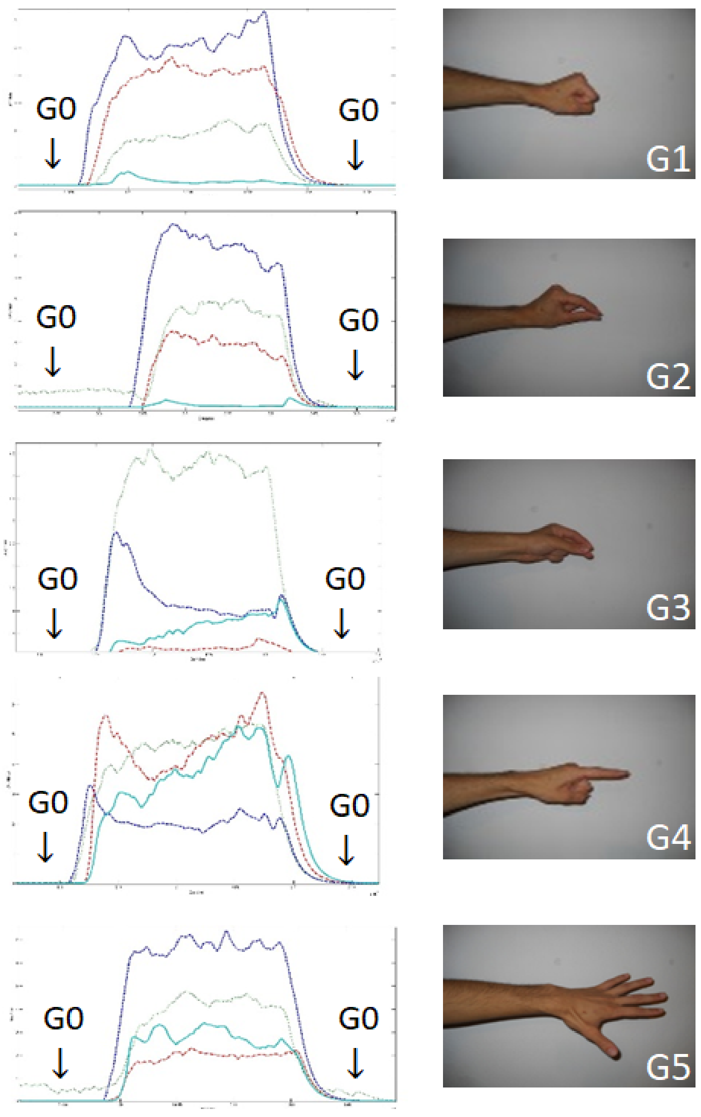


Figure 4.4: Amplified sEMG signal patterns of the hand gestures constituting the classes of the Unibo-INAIL dataset. G_0: rest position; G_1: power grip; G_2: two-fingers pinch grip; G_3: three-fingers pinch grip; G_4: pointing index; G_5: open hand. Image adapted from [51].

4.2 Pipeline and CNN architecture

4.2.1 Preprocessing: windowing

Data preprocessing consisted in an overlapping windowing scheme: the electromyogram signals were decomposed into segments of duration 150 ms, with a 75% overlap between consecutive ones. Since the dataset was acquired with 4 electrodes and sampling rate 500 Hz, for approximately 15 min/session, this windowing produced a sample size of the order of $M \sim 25.000$ windows/session, each with dimensions 75 samples $\times$ 4 channels. Each window was given the label of the hand gesture of its central sample.

The choice of duration 150 ms and overlap 75% was made with a preliminary analysis exposed in Subsection 6.2.1, showing that duration 150 ms yields a higher classification accuracy than 50 ms and 100 ms.

Window duration and overlap are parameters strictly related to the engineering problem met in HMI development: real-time classification robustness, especially during transients. Longer windows allow to reduce the impact of transients, but the window length must be shorter than 300ms [22] to satisfy real-time usage constraints. 150 ms is a favourable but realistic compromise. With regard to overlap, 75% is a good compromise to produce an adequate sample size ($M \sim 25.000$ windows/session) without exceeding in redundancy.

4.2.2 Three-way data partition

Each of the 224 sessions (i.e. combinations of subject, day and arm posture) of the dataset was subjected to a three-way data partition.

Random 10% holdout. First, a random 10% of the signal windows were held out as test set. Reproduction of identical holdout at each execution was ensured by setting NumPy's pseudorandom seed to a fixed value. No stratification with respect to class was enforced. This test set was used in the very last step of the pipeline, to compute on new data:

- the inter-posture test accuracy of the best postural training strategy;
- the inter-day test accuracy of the best multi-day training strategy.

2-fold partition with gesture integrity. After holdout, a 2-fold partition was applied on the remaining data (again, separately for each data session), to create two sets acting in turn as training set and validation set, in a 2-fold cross-validation scheme. The two folds were created starting from a 10-fold linear split, then putting the odd intervals into Fold 1 and the even intervals into Fold 2. This partition was chosen because it is the one yielding the best classification accuracy in [15], where it is called Training 50%D (D meaning decimal). The motivation of this scheme is shuffling the 10 gesture repetitions while approximately preserving the integrity of each repetition.

4.2.3 CNN architecture implemented

The architecture of the CNN implemented is an adaptation of the CNN module of the attention-based hybrid CNN-RNN architecture proposed by Hu et al. in [9] for sEMG-based gesture recognition on five public benchmark databases (not including the Unibo-INAIL dataset). More in detail, the hybrid architecture is a very large sequential model which stacks multiple parallel 2d-CNNs (identically trained), an LSTM, and an attention module. This model was chosen as a starting point due to its good performance (better than the state-of-the-art at publication) and to its modular structure, which is inspiring for exploring variations.

In this work, the 2d-CNN module from [9] was taken and converted to a 1d-CNN. Conversion from 2d to 1d was needed to act on the time windows produced in the preprocessing step (Section 4.2.1), having dimensions 75 samples $\times$ 4 channels. For a CNN, this format corresponds to 1d images 75×1 possessing 4 channels (or "colors").

The resulting CNN architecture is shown in Figure 4.5 and has 9 layers, listed in Table 4.1. The first two layers are 1d-convolutional layers with 64 kernels of size 3. They are followed by two locally-connected layers with 64 kernels of size 1, employed to extract features of the sEMG signal that are temporally circumscribed (i.e. "local" in time). For all these layers, batch normalization is applied to reduce internal covariate shift. The fifth, sixth and seventh layers are all fully-connected layers with batch normalization. Moreover, dropout with probability $p = 0.5$ is applied to the first two fully-connected layers to provide regularization. The fully connected layers are followed by a 6-way fully-connected layer (6 being the number of classes, i.e. the five hand gestures plus the rest position) and a softmax classifier. Except for the latter, all layers have ReLU activation function.

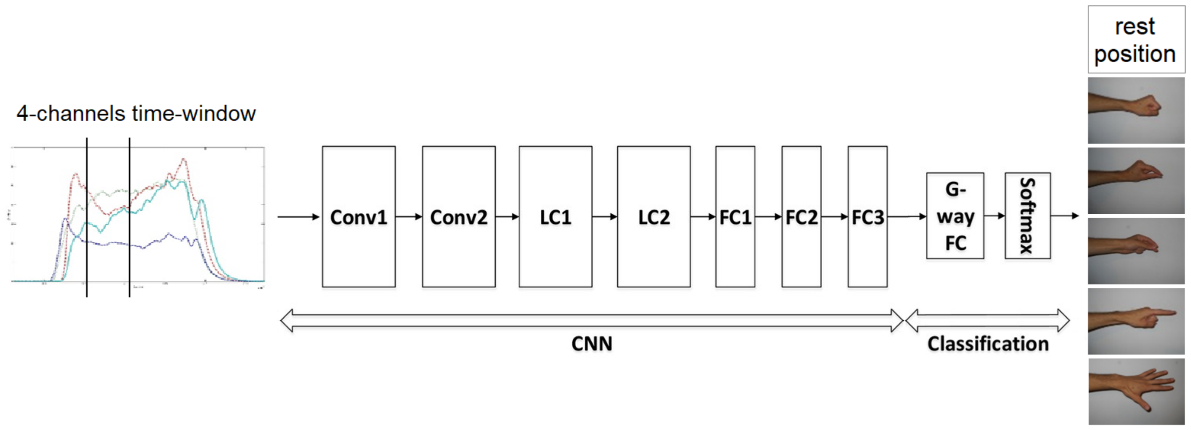


Figure 4.5: Diagram of the 1d-CNN implemented. Image adapted from [9] and [52].

Layer	Name	Details
1	Conv1	Convolutional 1d, 64 kernels, kernel size 3
2	Conv2	Convolutional 1d, 64 kernels, kernel size 3
3	LC1	Locally connected, 64 kernels
4	LC2	Locally connected, 64 kernels
5	FC1	Fully connected, 512 units
6	FC2	Fully connected, 512 units
7	FC3	Fully connected, 128 units
8	6-Way FC	Fully connected, 6 units
9	SoftMax	Softmax function

Table 4.1: Layers of the 1d-CNN implemented.

4.2.4 Training settings

The optimal parameters regulating CNN training were found via preliminary analyses, observing the network's behaviour when trained with various settings. In particular, the optimal learning rate, number of epochs and scheduling were chosen through the preliminary analysis exposed in Subsection 6.2.2. The optimal settings, used for all the trainings in the pipeline, are the following:

- random initialization of weights and biases: PyTorch default;
- loss function: cross-entropy loss, implemented by the PyTorch command `criterion = torch.nn.CrossEntropyLoss()`, which stacks log-softmax (`nn.LogSoftmax()`) and negative log-likelihood loss (`nn.NLLLoss()`);
- optimization algorithm: Stochastic Gradient Descent (SGD) with the number of mini-batches kept fixed at $B = 50$ for all trainings (thus with mini-batch size varying proportionally to training set size); mini-batches are re-randomized at each epoch;
- learning rate: `lr` = 0.001;
- early stopping: 20 epochs;
- scheduling: learning rate is divided by 10 after epoch 19;
- regularization: L_2 regularization setting PyTorch parameter `weight_decay = 0.1`, which corresponds to $\lambda_{L_2} = 0.05$.

4.3 Training strategies

The very aim of this work is to assess the ability of the implemented CNN model to generalize to data coming from arm postures or days not seen in training. In particular, the focus is on evaluating whether training on more postures or more days benefits the CNN's generalization ability. This is done by implementing the same training strategies studied by Milosevic et al. in [15], i.e. single-session, two-posture, two-day, and five-day training. The positive results of Milosevic et al. (limited to classical machine learning classifiers applied to instantaneous 4-channel signal values) play the role of baseline.

4.3.1 Single-session training strategy

The simplest training strategy implemented is the single-session training strategy: for each session, i.e. a combination of user, day, and arm posture, 2 CNNs are trained (one for each fold), using only data taken from that session. Classification accuracy is then evaluated via intra-session validation and inter-session validation:

- in intra-session validation, the trained CNN is evaluated on the data of the fold not used for training;

- in inter-session validation, the trained CNN is evaluated the data of a different session, to measure the CNN's ability to generalize; for the single-session training strategy, the inter-session validations computed are the inter-posture validation (all combinations), and inter-day validations on days D2 to D8 after training on day D1.

4.3.2 Two-posture training strategy

The training strategy developed to address postural variability is two-posture training: for each subject, day, and arm posture pair, 2 CNNs are trained (one for each fold), using only data taken from those two sessions (i.e. that subject, that day and those two postures). The posture pair considered are P1+P2, P1+P3, and P1+P4, where the choice of always including P1 is motivated by the fact that P1 is the only posture with the arm not fully extended, thus the most dissimilar from the other ones. Classification accuracy is then measured via intra-postures validation and inter-posture validation:

- in intra-postures validation, the trained CNN is evaluated on the fold not used for training (the plural intra-posture*s* is because each fold now contains data from two arm postures);
- in inter-posture validation, the model is evaluated on the data of a different posture, to measure the model's ability to generalize between postures.

4.3.3 Multi-day training strategies

The training strategies proposed to address temporal variability is multi-day training: for each subject, day combination, and arm posture, 2 CNNs are trained (one for each fold), using only data from that subject, those days and that posture. In particular, multi-day trainings are two-day trainings, which use D1+D2, D1+D5, and D4+D5, and five-day training, which use D1 to D5. These combinations were chosen in order to repeat the setup which produced the baseline results. Classification accuracy is then measured via intra-day validation and inter-days validation:

- in intra-days validation, the trained CNN is evaluated on the fold not used for training (the plural intra-posture*s* is because each fold now contains data from two or five days);

- in inter-day validation, the model is evaluated on the data of days D6 to D8 (absent in all multi-day training combinations), to measure the model's ability to generalize to different days.

Chapter 5

Implementation

The pipeline and the CNN described in the previous chapter were implemented in Python scripts whose heart, dealing with CNN definition, training and evaluation, relies on the open source library PyTorch.

This chapter is structured as follows:

- Section 5.1 illustrates in general the scripts developed for the different training strategies;
- Section 5.2 explains more in detail the main PyTorch packages and how they were used.

5.1 Scripts developed

Although the Unibo-INAIL dataset is publicly available with a series of Matlab scripts to help analyses, these were not exploited. Only one was partially used: the script `script_training_single_sessions.m`, devoted to training all the classical machine learning tested in [15] with single-session data, corresponding to the first training strategy (i.e. data from one subject, one day and one arm posture; details in Section 4.3). This script was translated into Python and the following advances were implemented:

- data partition (detailed in Section 4.2) was rewritten from scratch as the function `load_preprocess_and_split(_)`, to make it more compact, to add holdout and to make the K-fold partition easily costumizable by simply setting a parameter `K`;

- windowing preprocessing (see Section 4.2.1) was added as the function `windowing(_)` (called inside the previous one, immediately before doing data partition), since the previous implementations only addressed the classifications of 4-channel instantaneous signal values;
- in the core block, devoted to instantiation, training and validation, the four classical algorithms were replaced by the CNN, implemented with the library PyTorch.

The structure of `for` loops cycling on subjects, days and arm postures was approximately preserved.

This revised script was in turn used as a template to implement the two-posture, two-day and five-day training strategies (see Section 4.3), each one in a separate script, totalling four scripts:

- `CNN_on_UniboINAIL.py`, for single-session trainings;
- `biposture.py`, for two-posture trainings;
- `biday.py`, for two-day trainings;
- `fiveday.py`, for five-day trainings.

Execution times on a single GPU were approximately 8 h for `CNN_on_UniboINAIL.py` and `biposture.py`, and approximately 5 h for `biday.py` and `fiveday.py`.

5.2 Usage of PyTorch platform

The CNN instantiation, training and validation were implemented using the Python library PyTorch, an open source deep learning platform for Python, based on Torch, that is widely used in deep learning applications. This work has taken advantage of both PyTorch's high-level features: tensor computation on variables of class `torch.Tensor` (partially analogous to NumPy arrays, but more powerful), and the possibility of GPU computation.

In this master thesis, the CNNs were implemented in compliance with the standard structure of PyTorch scripts for CNNs, and the main PyTorch packages exploited are `torch.nn`, for neural networks instantiation and usage, `torch.autograd`, for automatic differentiation, `torch.optim`, for optimization, and `torch.cuda` for GPU computation. The following subsections explain these packages and the way they were used.

5.2.1 `torch.nn` package

`torch.nn` is the package that helps defining the complex neural networks for which operations on `torch.Tensor`s with raw `autograd` alone are too low-level. In particular, `torch.nn.Module` is the base class for all neural network modules, and user-defined models must subclass this class. If desired, modules (submodules) can be nested inside other Modules, creating a tree structure.

While reporting the code written to define the CNN architecture would be excessive, it is worth to show how instantiating the defined architecture is straightforward. It is done via two commands:

```
net = Net()
criterion = torch.nn.CrossEntropyLoss()
```

The first line instantiates the CNN `net`. The second line instantiates the loss function, termed criterion in PyTorch terminology. The class `torch.nn.CrossEntropy` implements a cross-entropy loss function which stacks a log-softmax operation (`torch.nn.LogSoftmax()`) and negative log-likelihood loss (`torch.nn.NLLLoss()`).

5.2.2 `torch.autograd` package and `torch.Tensor` class

`torch.autograd` is the package that implements automatic differentiation: gradients with respect to the parameters are automatically calculated directly at the forward pass (by recording the executed operations into computational graphs and re-executing them backward), thus reducing both development and execution time. The classes and functions provided implement automatic differentiation of arbitrary scalar-valued functions, with the only requirement that the variables with respect to which gradients are computed be `torch.Tensor` objects.

`torch.Tensor` is the base class of the `autograd` package. `torch.Tensor`s are multi-dimensional arrays (similar to NumPy array), which support automatic differentiation. Every `torch.Tensor` variable possesses a `requires_grad` flag, which allows to enable or disable gradient computation with respect to that variable. So, typically, data are `torch.Tensor`s with `requires_grad=False`, and model parameters are `torch.Tensor`s with `requires_grad=True`. Disabling gradients is also useful for freezing parts of a model, e.g. when fine-tuning other model parts.

The operations performed on tensors having `requires_grad` set to `True` are tracked. After finishing computations, gradients can be computed automatically by calling `backward()` on the result, i.e. the variable whose value was computed with the function to differentiate. The gradient with respect each `torch.Tensor` is then accumulated in its `.grad` attribute[1].

Formally, `autograd` is a reverse automatic differentiation system. As manipulations on data are executed, `autograd` produces a graph recording the operations that originate the result. This yields a directed acyclic graph having input tensors as leaves and output tensors as roots. Gradients are computed automatically by chain rule, by tracing the computational graph from roots to leaves (thus executing back-propagation).

Inside `autograd`, computational graphs are represented as graphs of `torch.Function` objects. During the forward pass, `autograd` simultaneously creates the graph representing the function that computes the gradient. Then, in the backward pass the graph is evaluated to compute the gradients. A valuable property is that the graph is re-built from scratch at every iteration, which allows to use arbitrary Python control flow statements, that can originate different graph structures at every iteration. This is the define-by-run framework, in which there is no need to encode all possible paths before launching the training, also described with the sentence "What you run is what you differentiate".

The `torch.autograd` mechanics were exploited to compute the gradient of the training loss, computed as a cross-entropy. After initializing the model and the cross-entropy objective function by `net = Net()` and `criterion = torch.nn.CrossEntropyLoss()` (as explained in the previous subsection), the values of the loss and loss gradient were computed at each iteration as follows:

```
loss = criterion(net(XTrain[batch_idxs]), YTrain[batch_idxs])
loss.backward()
```

[1]Formally, differentiation computes gradients *of* functions *with respect to* arguments (here, `torch.Tensor` parameters). Sometimes, in machine learning and deep learning, the abuse of language is made of speaking of derivatives *of* the parameters. The justification is that differentiation almost always acts on the training objective function. However, in `autograd`, speaking of gradients *of* the tensors is correct in the sense that gradient values are stored in each tensor's `.grad` attribute.

5.2.3 `torch.optim` package

The package `torch.optim` implements the most common iterative optimization algorithms. It is used instantiating and handling an `Optimizer` object (or more than one), which keeps track of the current state of optimization and, when its `step()` method is called, updates the parameters based on the gradients previously computed.

The optimization algorithm used in this work is Stochastic Gradient Descent (SGD) with mini-batches (re-randomized at each epoch), implemented via the command

```
optimizer = torch.optim.SGD(net.parameters(), lr=0.001, weight_decay=0.1)
```

where `SGD(_)` is the `torch.optim` function which instantiates SGD optimizers (without requiring to declare mini-batch size), `net.parameters()` are intuitively the parameters of the CNN model `net`, `lr` is the learning rate, and `weight_decay` corresponds to $2 \cdot \lambda_{\mathrm{L}_2}$ determining the amount of L_2 regularization.

For each mini-batch, optimization steps are executed by calling the `step()` method of the optimizer, immediately after computing the gradients on the mini-batch via `backward()`:

```
loss = criterion(net(XTrain), YTrain)
loss.backward()
optimizer.step()
```

This performs one update of the parameters, based on the gradients.

To implement scheduling, `torch.optim.lr_scheduler` was used, which provides methods for non-dynamic scheduling, i.e. learning rate adjustment based solely on epoch number, without adaptive validation measurements. The scheduling chosen after preliminary analysis (results in Subsection 6.2.2), i.e. division of the learning rate by 10 at epoch 19, was implemented via the command:

```
scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                             step_size=19, gamma=0.1)
```

where `optimizer` is the optimizer (in this case, a `torch.optim.SGD` object) whose learning rate is being scheduled, and `step_size` and `gamma` are the period and the multiplicative factor of the learning rate decay, respectively. Then, scheduling is applied simply by calling `scheduler.step()` at each epoch.

5.2.4 `torch.CUDA` package

In addition to supporting automatic differentiation, a further advantage of `torch.Tensor`s over NumPy arrays is that they can be cast to a GPU to improve computational performance.

This can be made using the package `torch.cuda`. It keeps track of the currently selected GPU, and all allocated CUDA tensors are created by default on that device. A `torch.cuda.device` context manager allows to change the selected device. However, once a `torch.Tensor` is allocated, operations can be performed on it irrespective of the selected device, and the results are automatically placed on the same device as the `torch.Tensor`. A `torch.Tensor`'s device can be accessed via the `Tensor.device` property.

The GPU computation is enabled by adding

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
```

at the beginning of the script, to set `device` to `'cuda:0'` if a GPU device is available. A `torch.device` is an object representing the device on which a `torch.Tensor` is or will be allocated, and is constructed so as to contain the device type (`'cpu'` or `'cuda'`) and an optional device ordinal for the device type.

The changes to cast model and data (and thus computations) to the GPU are few and straightforward, and are made via the command `.to(device)`. The CNN instantiation `net = Net()` becomes

```
net = Net().to(device)
```

and the evaluation and loss computation `loss = criterion(net(XTrain), YTrain)` becomes

```
loss = criterion(net(XTrain[batch_idxs, :, :].to(device)),
                 YTrain[batch_idxs].to(device))
```

Chapter 6

Results

6.1 Accuracy distributions and reported accuracies

Due to the highly multi-source nature of the Unibo-INAIL dataset, the CNNs trained with each training strategy (single-session, two-postures, two- and five-day) return a distribution of classification accuracies, over the 7 subjects, 8 days and 4 arm postures. This allows to investigate the variability of performance over the dataset. On this basis, in this Chapter all accuracies are reported with:

- average accuracy μ (also called mean for simplicity), computed over the sessions of interest for each case;
- accuracy standard deviation σ, which quantifies the performance variability over the data sources;
- standard error on the mean accuracy $\mathrm{SE} = \sigma/\sqrt{N}$, used to estimate the uncertainty on the average accuracy.

In particular, the SE does not describe the broadness of the accuracy distribution, but is an estimate of the fluctuations affecting the average accuracy. The 224 sessions (times 2 folds) of the Unibo-INAIL dataset allow to divide by a large $\sqrt{N}$ (varying according to the training strategy studied). Thus the 224 sessions are not only a computational burden, but also allow to report average accuracies with a SE of the order of $\sim 0.1\%$.

In this chapter, the reported accuracies are compared with the baseline achieved by the best classical machine learning classifier reported in [15]. However, for the baseline accuracies the values of σ or SE are not reported, not allowing for a complete statistical comparison.

6.2 Preliminary analyses

6.2.1 Optimal length of time windows

The first preliminary analysis was made to optimize the window length to be used in preprocessing, consisting in an overlapping windowing scheme as detailed in Subsection 4.2.1. Since window length must be shorter than 300ms to satisfy real-time usage constraints, the values explored were 50 ms, 100 ms and 150 ms.

In this analysis, the performance of interest is the accuracy obtained on 2-fold cross-validation on single-session data (i.e. single user, single day, single arm posture), after training on data from the same session (i.e. using the two folds alternately for training and intra-session validation). The results are reported in Table 6.1. With a $(94.4 \pm 0.2)\%$ accuracy, duration 150 ms proved to be the best one and was adopted the preprocessing.

Since the dataset was acquired with 4 electrodes and sampling rate 500 Hz, for approximately 15 min/session, this windowing produced a sample size of the order of $M \sim 25.000$ windows/session, each with dimensions 75 samples $\times$ 4 channels. Moreover, with regard to overlap, 75% was chosen as a good compromise to produce an adequate sample size ($M \sim 25.000$ windows/session) without exceeding in redundancy.

Window length	**Intra-session val. accuracy**	
	$\mu \pm$ SE	σ
50 ms	$(93.6 \pm 0.2)\%$	3.7%
100 ms	$(94.0 \pm 0.2)\%$	3.6%
150 ms	$(94.4 \pm 0.2)\%$	3.5%

Table 6.1: Intra-session validation accuracies yielded by the time-windows durations explored.

6.2.2 Learning curves

The second preliminary analysis was performed to optimize the learning rate and the scheduling thereof. This exploration was carried on with the same single-session scheme used to optimize the window length described in Subsection 6.2.1.

The optimal learning rate was searched by looking at single intra-session validation learning curves, like the one shown in Figure 6.1 (obtained for subject 1, day 1, arm posture 1, training on fold 1 and validation on fold 2). Using SGD with the training set split into 50 mini-batches (re-randomized at each epoch), the optimal learning rate was found to be `lr` $= 0.001$, which yielded the best compromise between a reasonably fast convergence and a sufficiently low final loss. This optimal value was adopted for all the trainings.

Average learning curves were used to diagnose overfitting, as shown in Figure 6.2: the validation loss reaches a minimum between epochs 15 and 20, than increases again, indicating overfitting. A simple scheduling tactic, consisting in dividing the learning rate by 10 at epoch 20, allowed to improve the minimum of the validation loss, as shown in Figure 6.3. Since this scheduling does not remove the overfitting trend, but only makes it slower, the final choice was to exploit the sudden minimum: the final scheduling chosen consists in dividing the learning rate by 10 at epoch 19, then applying early stopping at epoch 20. This scheduling was used in all the trainings.
NOTE. After this preliminary analysis, validation cross-entropy losses were abandoned in favour of classification accuracies. This was motivated by the ease of interpretation and informal comparison with baseline accuracies.

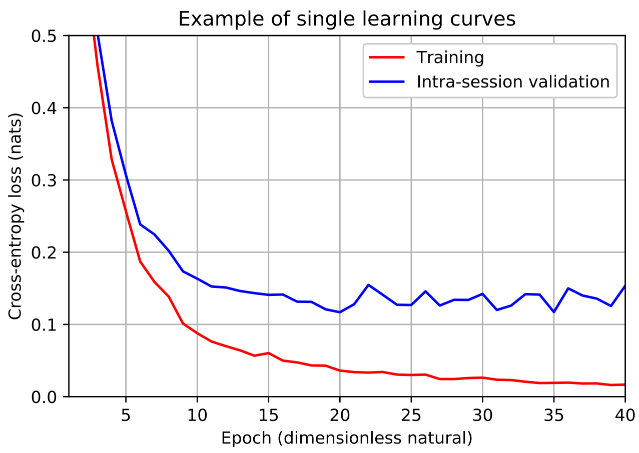


Figure 6.1: Single learning curves, obtained for subject 1, day 1, arm posture 1, training on fold 1 with learning rate `lr` $= 0.001$, validation on fold 2.

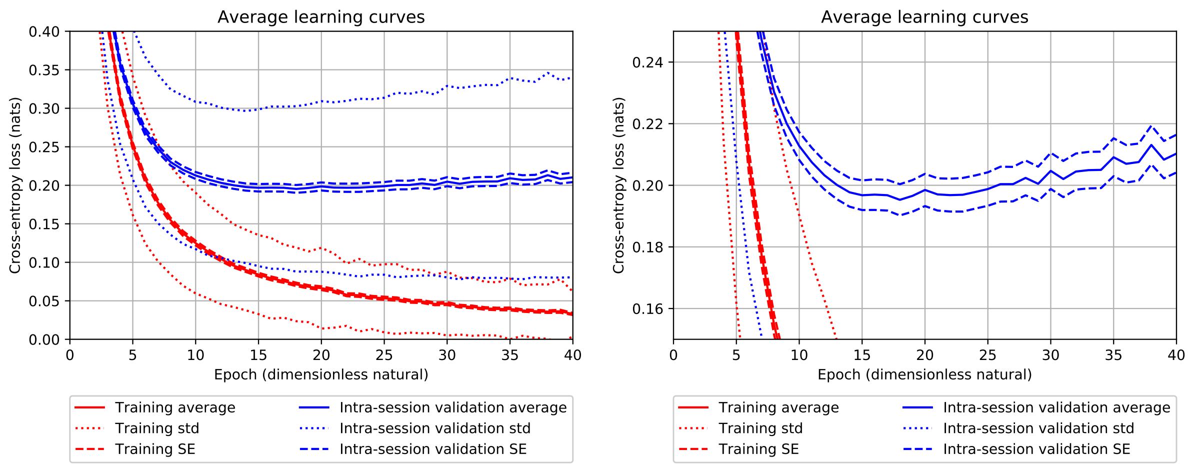


Figure 6.2: Average learning curves obtained with learning rate `lr` $= 0.001$, no scheduling. Averages, σ's and SEs taken on all subjects, all days, all arm postures, both folds. The left and right panels show the same learning curves, visualized with y range $[0, 0.4]$, and $[0.15, 0.25]$, respectively.

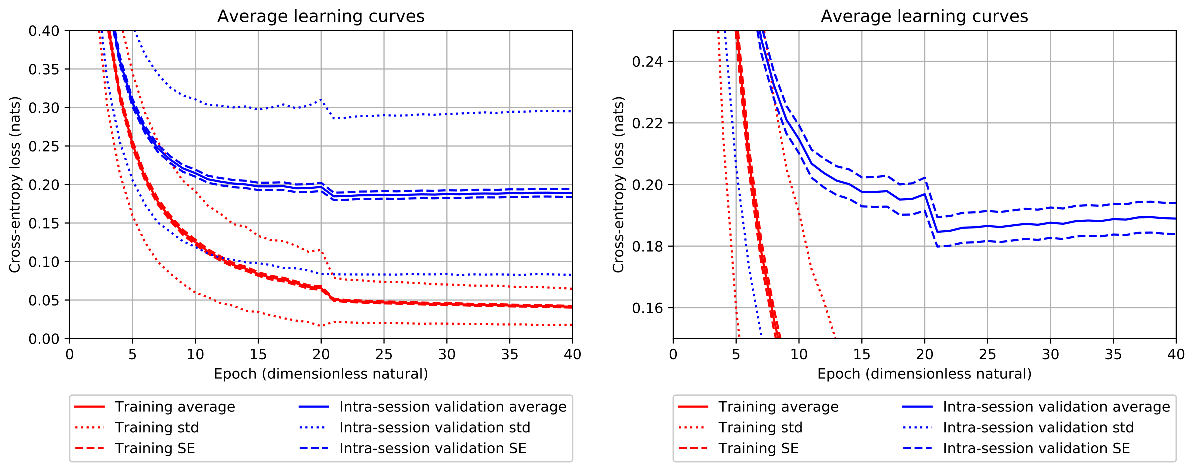


Figure 6.3: Average learning curves obtained with learning rate `lr` $= 0.001$, divided by 10 at epoch 20. Averages, σ's and SEs taken on all subjects, all days, all arm postures, both folds.

6.3 Single-session training strategy

The basic training strategy explored is the single-session training strategy: for each session, i.e. a combination of user, day, and arm posture, 2 CNNs are trained (one for each fold), using only data from that session. Performance is then measured via intra-session validation and inter-session validation:

- in intra-session validation, the trained CNN is evaluated on the fold not used for training;
- in inter-session validation, the model is evaluated on the data of a different session, to assess the model's ability to generalize; for the single-session training strategy, the inter-session validations computed are the inter-posture validation (all combinations), and inter-day validations on days D2 to D8 after training on day D1.

With regard to intra-session validation, the overall accuracy is $(\mu \pm \text{SE}) = (94.5 \pm 0.2)\%$, with $\sigma = 3.5\%$. This performance is very similar to the baseline value of 94% achieved with a RBF-SVM, which is not consistent within the SE, but a complete statistical comparison is not possible because the baseline σ is not available.

Looking at accuracy distributions computed by subject, by day and by arm posture, some interesting findings emerge. Accuracy distributions by subject are reported in Figure 6.4 and in Table 6.2; accuracy distributions by day are reported in Figure 6.5 and in Table 6.3; and accuracy distributions by arm posture are reported in Figure 6.6 and in Table 6.4.

The accuracy distributions (visualized as $\mu \pm \sigma$) of the 8 days and of the 4 arm postures always overlap within the standard deviations. The situation is different for the accuracy distributions for the 8 subjects: the means present larger variations, and Subject 3 has a mean so lower than the others that its accuracy distribution is not consistent within the standard deviation with the distributions of Subjects 4, 5, and 7. Subject 3's accuracy distribution is also the one presenting the highest standard deviation. Operatively, this indicates that, on average, within Subject 3's sessions hand gesture recognition is a harder task. It is worth to remark that the inter-session setup does not authorize to attribute lower accuracy to larger inter-day or inter-postural variability.

With regard to inter-session validation, Figure 6.7 and Table 6.7 show the inter-posture validation accuracies and compare them with the intra-posture case; all performances are reported by training posture. The overall inter-posture validation accuracy is 80.6%, corresponding to a 13.9% accuracy drop compared to the intra-posture (i.e. intra-session) scenario. This accuracy drop quantifies the amount of overfitting the single-session training produces with respect to the task of generalizing to different postures. The overall inter-posture accuracy is similar to the baseline value of 79%, but, again, a statistical comparison is not possible because the baseline σ or SE are not available.

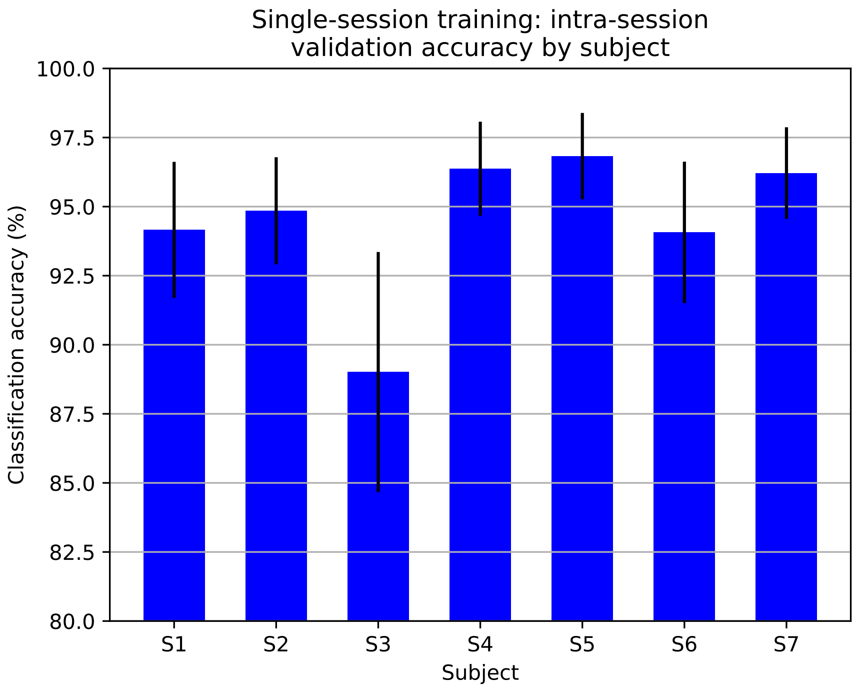


Figure 6.4: Single-session training: intra-session validation accuracies by subject. Error bars stand for $\pm\sigma$

Subject	**Intra-session val. accuracy**	
	$\mu \pm \mathrm{SE}$	σ
1	$(94.2 \pm 0.3)\%$	2.5%
2	$(94.8 \pm 0.2)\%$	1.9%
3	$(89.0 \pm 0.5)\%$	4.3%
4	$(96.4 \pm 0.2)\%$	1.7%
5	$(96.8 \pm 0.2)\%$	1.6%
6	$(94.1 \pm 0.3)\%$	2.6%
7	$(96.2 \pm 0.2)\%$	1.7%

Table 6.2: Single-session training: intra-session validation accuracies by subject.

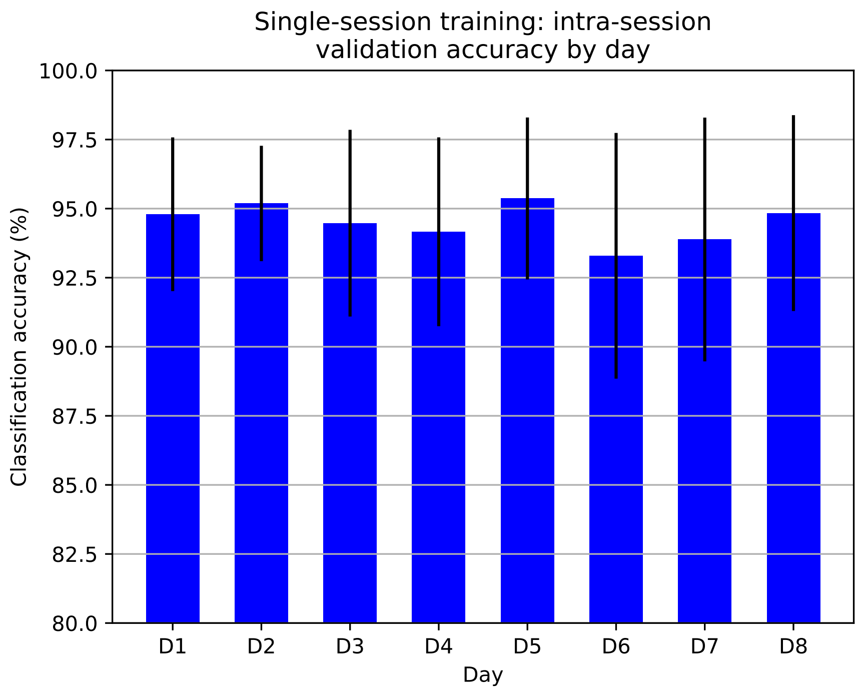


Figure 6.5: Single-session training: intra-session validation accuracies by day. Error bars stand for $\pm\sigma$

Day	**Intra-session val. accuracy**	
	$\mu \pm$ SE	σ
1	$(94.8 \pm 0.4)\%$	2.8%
2	$(95.2 \pm 0.3)\%$	2.1%
3	$(94.5 \pm 0.4)\%$	3.4%
4	$(94.2 \pm 0.4)\%$	3.4%
5	$(95.4 \pm 0.4)\%$	2.9%
6	$(93.3 \pm 0.6)\%$	4.4%
7	$(93.9 \pm 0.6)\%$	4.4%
8	$(94.8 \pm 0.4)\%$	3.5%

Table 6.3: Single-session training: intra-session validation accuracies by day.

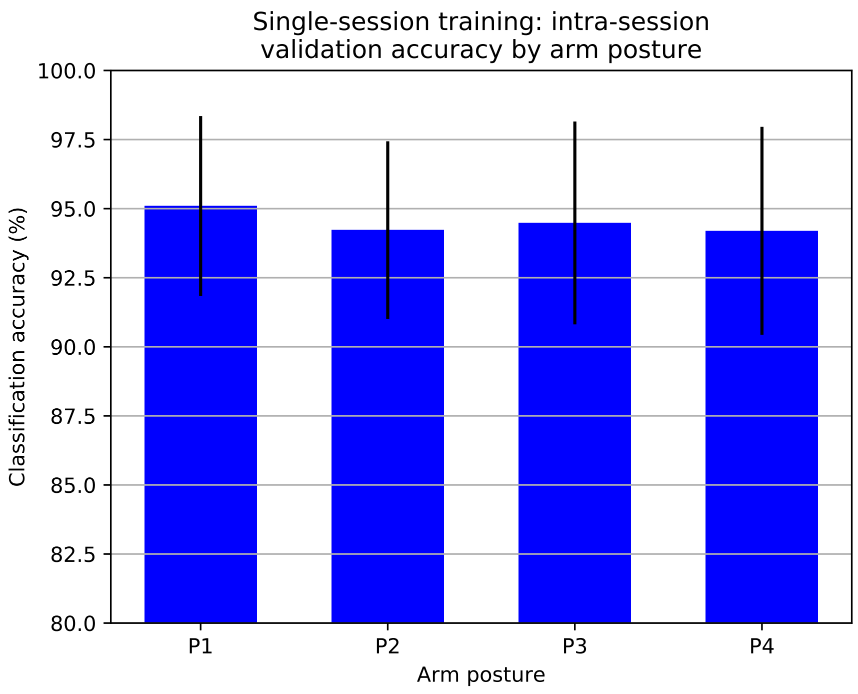


Figure 6.6: Single-session training: intra-session validation accuracies by arm posture. Error bars stand for $\pm\sigma$

Arm posture	**Intra-session val. accuracy**	
	$\mu \pm \mathrm{SE}$	σ
1	$(95.1 \pm 0.3)\%$	3.3%
2	$(94.2 \pm 0.3)\%$	3.2%
3	$(94.5 \pm 0.3)\%$	3.7%
4	$(94.2 \pm 0.4)\%$	3.8%

Table 6.4: Single-session training: intra-session validation accuracies by arm posture.

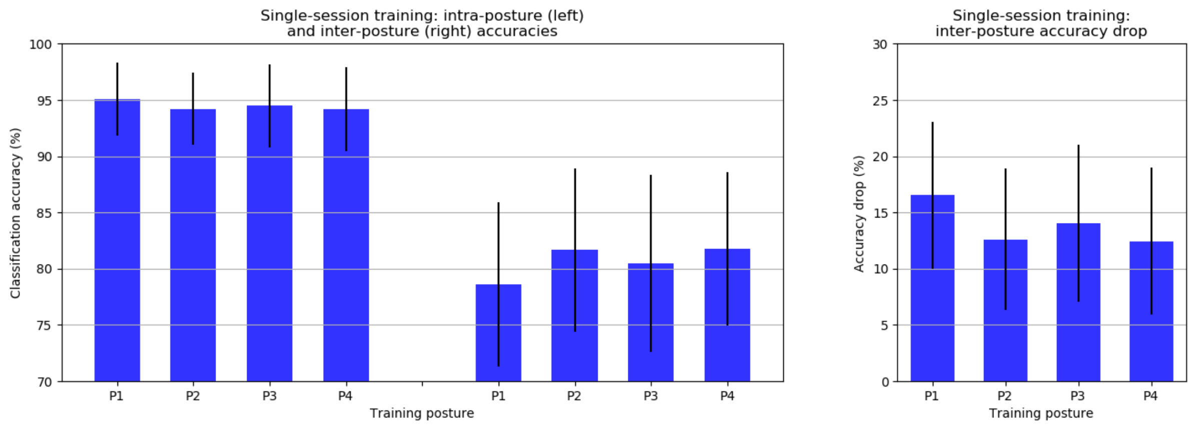


Figure 6.7: Single-session training: intra-posture and inter-posture validation accuracies, and accuracy drops, by training posture. Error bars stand for $\pm\sigma$

Train posture	**Intra-posture accuracy**		**Inter-posture accuracy**		**Accuracy drop**	
	$\mu \pm$ SE	σ	$\mu \pm$ SE	σ	$\mu \pm$ SE	σ
1	$(95.1 \pm 0.3)\%$	3.3%	$(78.6 \pm 0.4)\%$	7.3%	$(16.5 \pm 0.4)\%$	6.5%
2	$(94.2 \pm 0.3)\%$	3.2%	$(81.6 \pm 0.4)\%$	7.3%	$(12.6 \pm 0.3)\%$	6.3%
3	$(94.5 \pm 0.3)\%$	3.7%	$(80.5 \pm 0.5)\%$	8.9%	$(14.0 \pm 0.4)\%$	7.0%
4	$(94.2 \pm 0.4)\%$	3.8%	$(81.8 \pm 0.4)\%$	6.8%	$(12.4 \pm 0.4)\%$	6.6%

Table 6.5: Single-session training: intra-posture and inter-posture validation accuracies.

6.4 Two-posture training strategy

The training strategy implemented to address postural variability is two-posture training: for each subject, day, and arm posture pair, 2 CNNs are trained (one for each fold), using only data from those two sessions (i.e. that subject, that day and those two postures). Posture pair considered are P1+P2, P1+P3, and P1+P4, where the choice of always including P1 is motivated by the fact that P1 is the only posture with the arm not fully extended, thus the most dissimilar from the other ones. Performance is then measured via intra-postures validation and inter-posture validation:

- in intra-postures validation, the trained CNN is evaluated on the fold not used for training;
- in inter-posture validation, the model is evaluated on the data of a different posture, to measure the model's ability to generalize between postures.

All the results, divided by training posture pair, are shown in Figure 6.8 and in Table 6.6.

With regard to intra-postures validation, the overall accuracy is $(\mu \pm \text{SE}) = (94.3 \pm 0.2)\%$, with $\sigma = 3.5\%$, which differs only by 0.2% from the intra-posture validation accuracy obtained with single-session (thus single-posture) training (which is $(\mu \pm \text{SE}) = (94.5 \pm 0.2)\%$). This difference is comparable to the SEs of the two compared averages, thus not statistically significant. Moreover, the intra-postures validation accuracy yielded by two-posture training is higher than the corresponding baseline value of 90% (standard deviation or SE not available). This indicates that in intra-posture validation the 1d-CNN performs better than the RBF-SVM.

The interpretation of these results is that the 1d-CNN, thanks to its higher capacity compared to SVM, is able to learn the hand gesture patterns coming from two arm postures with the same accuracy as it learns the patterns from a single posture. However, this success is marginal since the real aim is to improve inter-posture accuracy.

With regard to inter-posture validation, the overall inter-posture validation accuracy is 82.0%, corresponding to a 12.3% accuracy drop compared to the intra-postures case. The corresponding baseline value is 83%, which is higher but does not allow for a statistical comparison since it is available without standard deviation or SE. This result is better by 1.4% compared to the inter-posture validation accuracies achieved with single-session training. The accuracy drop quantifies the amount of overfitting produced by training on P1+Pi, with respect to the task of generalizing to non-training postures. This amount of overfitting is slightly smaller compared to one given by the single-session (this single-posture) training, which was 13.9%.

Thus, two-posture training improves the inter-posture generalization of the CNN. The fact that the intra-posture performance is not impacted, means that an amount of overfitting is removed without adding significant bias. This can be attributed to the 1d-CNN's capacity, which enables it to learn more diverse distributions (i.e. patterns from two postures instead that one) while preserving classification accuracy.

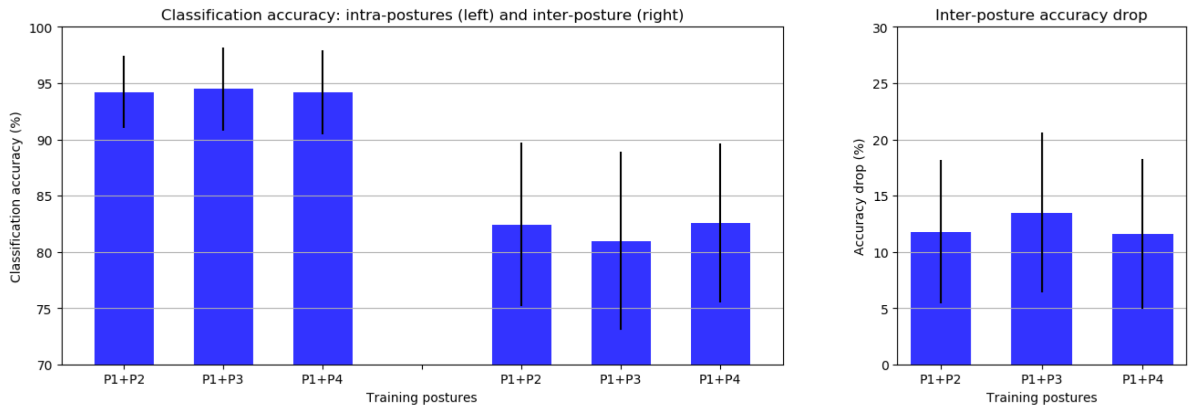


Figure 6.8: Two-posture training: intra-postures and inter-posture validation accuracies, and accuracy drops, by training posture pair. Error bars stand for $\pm\sigma$

Train postures	**Intra-postures accuracy**		**Inter-posture accuracy**		**Accuracy drop**	
	$\mu \pm$ SE	σ	$\mu \pm$ SE	σ	$\mu \pm$ SE	σ
P1+P2	(94.2 ± 0.2)%	3.2%	(82.4 ± 0.4)%	7.3%	(11.8 ± 0.4)%	6.4%
P1+P3	(94.5 ± 0.2)%	3.7%	(81.0 ± 0.5)%	7.9%	(13.5 ± 0.5)%	7.1%
P1+P4	(94.2 ± 0.3)%	3.8%	(82.6 ± 0.5)%	7.0%	(11.6 ± 0.4)%	6.7%

Table 6.6: Two-posture training: intra-postures and inter-posture validation accuracies.

6.5 Multi-day training strategy

The training strategy proposed to address temporal variability is multi-day training: for each subject, day combination, and arm posture, 2 CNNs are trained (one for each fold), using only data from that subject, those days and that posture. In particular, multi-day trainings are two-day trainings, which use D1+D2, D1+D5, and D4+D5, and five-day training, which use D1 to D5 (these combinations were chosen in order to repeat the setup which produced the baseline results). Performance is then assessed via intra-days validation and inter-day validation:

- in intra-days validation, the trained CNN is evaluated on the fold not used for training;
- in inter-day validation, the model is evaluated on the data of days D6 to D8 (absent in all multi-day training combinations), to measure the model's ability to generalize to different days.

All the results, divided by training posture pair, are shown in Figures 6.9 and 6.10 and in Table 6.7, which also include a comparison with single-day (thus single-session) training on D1 alone.

With regard to inter-day validation of the training strategy based on D1 alone, the $(66.9 \pm 1.1)\%$ accuracy means a 27.9% drop in accuracy compared to the intra-day validation, which yields $(94.8 \pm 0.3)\%$ accuracy. This proves that the inter-day variability is a larger effect than inter postural variability, whose impact was quantified as a 13.9% accuracy drop (inter-posture validation of the single-session training strategy).

Improvement compared to single-day training is evident for all the multi-day training strategies implemented. The D1-to-D5 training strategy yields the best inter-day validation accuracy, $(\mu \pm \text{SE}) = (76.4 \pm 1.2)\%$ with $\sigma = 9.0\%$. The five-day training strategy proves to be the best one also in the baseline results based on classical machine learning algorithms. The corresponding baseline value is 77%, which is consistent with the CNN result within the SE, allowing to say that the performance is statistically equivalent.

The second best strategy is D4+D5 strategy, achieving $(\mu \pm \text{SE}) = (74.4 \pm 1.2)\%$ with $\sigma = 9.2\%$. The statistical difference between the two was checked via a paired samples Wilcoxon test, which is the non-parametric equivalent of the paired samples t-test, chosen for its robustness with respect to sample distributions. The test yielded

$p_{\text{Wilcoxon}} = 4.0 \cdot 10^{-4}$, which means that the five-day training strategy is statistically significantly better (in validation) then the other ones.

Moreover, a trend can be noted which can be identified as user adaptation (explained in Subsection 4.1.4): the training strategies based on later days yield higher classification accuracy. This can be interpreted as the fact that inter-day differences in gesture execution decrease over time, as a consequence of the tendency of users to adapt to the repetitive exercise. This indicates again that training strategies prioritizing the recent data yield better performances.

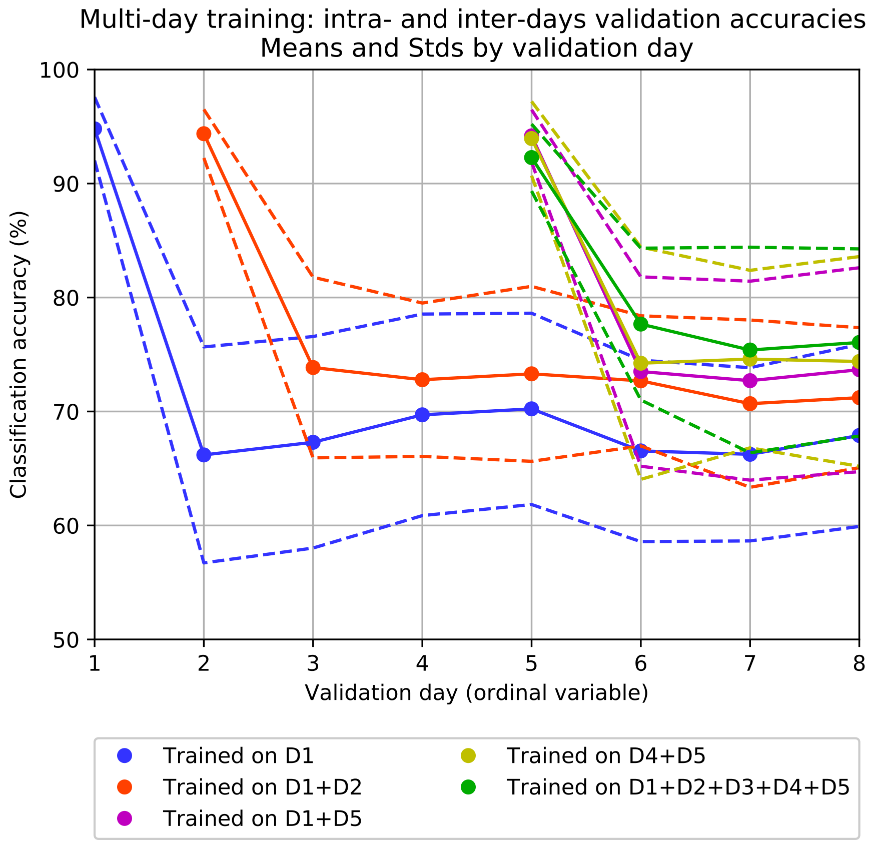


Figure 6.9: Multi-day training strategies compared to training on D1 alone: intra-day(s) accuracies and inter-day accuracies. Dots indicate average accuracies, and dashed lines indicate standard deviations.

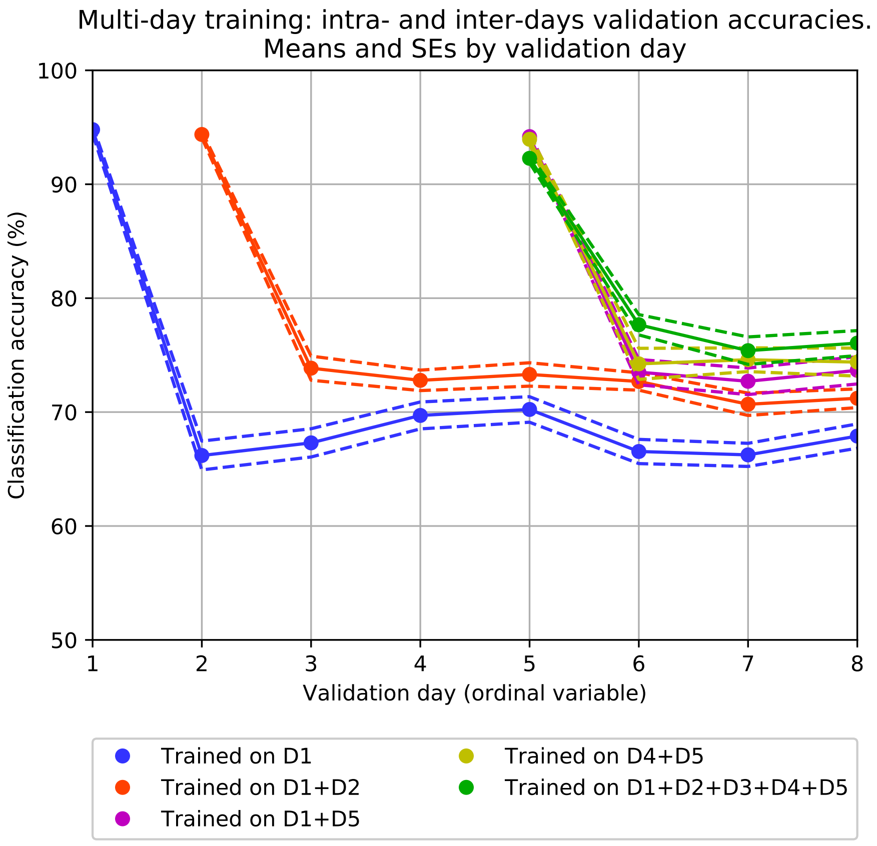


Figure 6.10: Multi-day training strategies compared to training on D1 alone: intra-day(s) accuracies and inter-day accuracies. Dots indicate average accuracies, and dashed lines indicate SEs.

Train days	**Intra-days val. accuracy**		**Inter-day val. accuracy on D6 to D8**	
	$\mu \pm$ SE	σ	$\mu \pm$ SE	σ
D1	$(94.8 \pm 0.3)\%$	2.3%	$(66.9 \pm 1.1)\%$	8.0%
D1+D2	$(94.7 \pm 0.3)\%$	2.1%	$(71.5 \pm 1.0)\%$	7.3%
D1+D5	$(94.2 \pm 0.3)\%$	2.3%	$(73.3 \pm 1.2)\%$	8.9%
D4+D5	$(93.9 \pm 0.4)\%$	3.3%	$(74.4 \pm 1.2)\%$	9.2%
D1 to D5	$(92.3 \pm 0.4)\%$	2.9%	$(76.4 \pm 1.2)\%$	9.0%

Table 6.7: Multi-day training strategies compared to training on D1 alone: intra-day(s) accuracies and inter-day accuracies.

6.6 Training strategies selection and test

On the basis of the inter-session validation results, two-posture training was selected as the best strategy for postural generalization, and five-day training was selected as the best strategy for temporal generalization. This indicates that training strategies should prioritize data from more than one posture and from recent days. This outcome is the same as in the baseline results obtained with classical machine learning algorithms. After retraining, i.e. repeating the CNNs training using both folds, these two strategies were tested on the 10% of data previously held out as test set (holdout details in Subsection 4.2.2).

The two-posture training strategy yielded inter-posture test accuracy $(\mu \pm \mathrm{SE}) = (81.2 \pm 0.4)\%$ with $\sigma = 7.3\%$,. The five-day training strategy yielded inter-day (on D6 to D8) test accuracy $(\mu \pm \mathrm{SE}) = (75.9 \pm 0.7)\%$ with $\sigma = 8.6\%$

Test baseline values are not available, since the test step is not present in the pipeline which produced the baseline results. As reference, it is possible to consider the corresponding validation accuracies, which are 83% for the inter-posture validation of the two-posture training and 77% for the inter-day validation of the five-day training (both provided without σ or SE). These results are similar to the test accuracies, but the limit of this comparison is that the baseline values, being the validation accuracies of the winner model (i.e. the RBF-SVM), are upward biased. However, the available values are sufficient to conclude that the final CNN performance is comparable with the baseline.

Chapter 7

Conclusions and future work

This master thesis is the first application of deep learning on the Unibo-INAIL dataset, the first public sEMG dataset exploring the variability between subjects, sessions and arm postures, by collecting data over 8 sessions of each of 7 able-bodied subjects executing 6 hand gestures in 4 arm postures. With the open-source deep learning platform PyTorch, it was possible to implement and test a 1d-CNN architecture trained with most recent strategies based on training set composition.

The single-session training strategy achieves 94.5% intra-session validation accuracy, but deteriorates to 80.6% in inter-posture validation and to 66.9% (for day 1) in inter-day validation. This proves that inter-day variability has a larger impact than inter-posture variability. A possible reason is that, on each day, the data of the 4 arm postures were collected without repositioning the sensors.

Multi-posture and multi-day training strategies yield higher inter-session validation accuracies. Two-posture training proves to be the best postural training strategy, indicating that the training strategies should prioritize data from more than one posture, and yields 81.2% inter-posture test accuracy. Five-day training proves to be the best multi-day training strategy, and yields 75.9% inter-day test accuracy. All the results are close to the baseline, provided by the accuracy of a RBF-SVM.

Moreover, the results of multi-day trainings allow to highlight user adaptation, the phenomenon which causes the inter-day differences in gesture execution to decrease over time, due to the tendency of users to adapt to the repetitive exercise during the first days. The detection of user adaptation indicates that the training strategies should also prioritize recent data.

Though not better than the baseline, the achieved classification accuracies rightfully place the 1d-CNN among the candidates for further research.

Future work will continue this research line investigating whether the fact that the deep 1d-CNN does not outperform the baseline is preprocessing-dependent or is due to an accuracy limit inherent to the Unibo-INAIL dataset. The question will be addressed using deep learning models relying on different data pre-processing, the first candidate being time-frequency domain analysed with 2d-CNNs.

Ringraziamenti

Al termine di questo lavoro, desidero rivolgere dei ringraziamenti.

Ringrazio il Prof. Daniel Remondini per la proficua supervisione durante il lavoro di tesi, e per l'incoraggiamento che mi ha fornito. Ringrazio il Dott. Simone Benatti, il Dott. Francesco Conti e il Dott. Manuele Rusci per la guida costante nel corso del lavoro di ricerca, e per il prezioso supporto che mi hanno dato. Ringrazio queste persone anche per i commenti, sempre dettagliati e tempestivi, elargiti nel corso della redazione dell'elaborato di tesi. Ringrazio infine il Prof. Luca Benini per avermi permesso di svolgere il lavoro di tesi magistrale presso il laboratorio MICREL (Unibo) da lui diretto.

Bibliography

[1] M. Cheok, Z. Omar, and M. Jaward, A review of hand gesture and sign language recognition techniques. *International Journal of Machine Learning and Cybernetics*, 2017.

[2] D. Farina, N. Jiang, H. Rehbaum, A. Holobar, B. Graimann, H. Dietl, and O.C. Aszmann, The extraction of neural information from the surface emg for the control of upper-limb prostheses: emerging avenues and challenges. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 22(4):797–809, 2014.

[3] R. Meattini, S. Benatti, U. Scarcia, D. De Gregorio, L. Benini, and C. Melchiorri, An sEMG-based human-robot interface for robotic hands using machine learning and synergies. In *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 2018.

[4] T. S. Saponas, D. S. Tan, D. Morris, J. Turner, and J. A. Landay, Making muscle-computer interfaces more practical. In *SIGCHI Conference on Human Factors in Computing Systems*, pages 851–854, 2010.

[5] M. Hakonen, H. Piitulainen, and A. Visala, Current state of digital signal processing in myoelectric interfaces and related applications. *Biomedical Signal Processing and Control*, 18:334–359, 2015.

[6] P. Tsinganos, B. Cornelis, J. Cornelis, B. Jansen, and A. Skodras, Deep Learning in EMG-based Gesture Recognition. In *5th International Conference of Physiological Computing Systems (PhyCS)*, 2018.

[7] K. Crammer, M. Kearns, J. Wortman, Learning from Multiple Sources. *Journal of Machine Learning Research* 9:1757-1774, 2008.

[8] Y. Du, W. Jin, W. Wei, Y. Hu, and W. Geng, Surface EMG-Based Inter-Session Gesture Recognition Enhanced by Deep Domain Adaptation. *Sensors*, 17(3) 2017.

[9] Y. Hu, Y. Wong, W. Wei, Y. Du, M. Kankanhalli, and W. Geng, A novel attention-based hybrid CNN-RNN architecture for sEMG-based gesture recognition. PLoS ONE 13(10), 2018.

[10] M. Atzori, M. Cognolato, and H. Müller, Deep learning with convolutional neural networks applied to electromyography data: a resource for the classification of movements for prosthetic hands. *Frontiers in neurorobotics*, 10:9, 2016.

[11] K.H. Park and S.W. Lee, Movement intention decoding based on deep learning for multiuser myoelectric interfaces. In *International Winter Conference on Brain-Computer Interface*, pages 1–2, 2016.

[12] W. Geng, Y. Du, W. Jin, W. Wei, Y. Hu, and J. Li, Gesture recognition by instantaneous surface EMG images. *Scientific Reports*, 6:36571, 2016.

[13] Y. Du, W. Jin, W. Wei, Y. Hu, and W. Geng, Surface EMG-based inter-session gesture recognition enhanced by deep domain adaptation. Sensors, 17(3), 2017.

[14] Y. Du, Y. Wong, W. Jin, W. Wei, Y. Hu, M. Kankanhalli, et al., Semi-supervised Learning for Surface EMGbased Gesture Recognition. In *International Joint Conference on Artificial Intelligence*, pages 1624–1630, 2017.

[15] B. Milosevic, E. Farella, and S. Benatti, Exploring Arm Posture and Temporal Variability in Myoelectric Hand Gesture Recognition. In *Proceedings of the IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics, IEEE Computer Society*, pages 1032-1037, 2018.

[16] C. De Luca, *Electromyography. Encyclopedia of Medical Devices and Instrumentation*, pages 98-109, 2006.

[17] L.G. Tassinary, J.T. Caccioppo, and E. Vanman, *The Skelemotor System: Surface Electromyography*, pages 267-299.

[18] M. Tomasini, S. Benatti, B. Milosevic, E. Farella, and L. Benini, Power line interference removal for high-quality continuous biosignal monitoring with low-power wearable devices. *IEEE Sensors Journal*, 16(10):3887–3895, 2016.

[19] R.M. Rangayyan, *Biomedical Signal Analysis* (IEEE Press series in biomedical engineering), 2015. Wiley.

[20] E.S. Nurse, P.J. Karoly, D.B. Grayden, and D.R. Freestone, A generalizable brain-computer interface (bci) using machine learning for feature discovery. PLoS ONE, 10(6):e0131328, 2015.

[21] E. Nurse, B.S. Mashford, A.J. Yepes, I. Kiral-Kornek, S. Harrer, and D.R. Freestone, Decoding EEG and LFP signals using deep learning: heading TrueNorth. In *Proceedings of the ACM International Conference on Computing Frontiers*, pages 259-266, 2016.

[22] B. Hudgins, P. Parker, and R. Scott, A new strategy for multifunction myoelectric control. *IEEE Transactions on Biomedical Engineering*, 40(1):82–94, 1993.

[23] K. Englehart and B. Hudgins, B., A robust, real-time control scheme for multifunction myoelectric control. *IEEE Transactions on Biomedical Engineering*, 50(7):848–854, 2003.

[24] C. Castellini, A. Fiorilla, and G. Sandini, Multisubject/daily-life activity EMG-based control of mechanical hands. *Journal of neuroengineering and rehabilitation*, 6:41, 2009.

[25] I. Kuzborskij, A. Gijsberts, and B. Caputo, On the challenge of classifying 52 hand movements from surface electromyography. In *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 4931–4937, 2012. IEEE.

[26] M. Atzori, A. Gijsberts, I. Kuzborskij, S. Elsig, A. Hager, O. Deriaz, C. Castellini, H. Müller, and B. Caputo, Characterization of a benchmark database for myoelectric movement classification. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 23(1):73–83, 2015.

[27] M. Atzori, A. Gijsberts, C. Castellini, B. Caputo, A. Hager, S. Elsig, G. Giatsidis, F. Bassetto, and H. Müller, Electromyography data for non-invasive naturally-controlled robotic hand prostheses. *Scientific Data*, 1(140053), 2014.

[28] A. Gijsberts, M. Atzori, C. Castellini, H. Müller, and B. Caputo, Movement error rate for evaluation of Machine Learning methods for sEMG-based hand movement classification. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 22(4):735–744, 2014.

[29] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, 2016. MIT Press, Cambridge, MA.

[30] W. Wei, Y. Wong, Y. Du, Y. Hu, M. Kankanhalli, and W. Geng, A multi-stream Convolutional Neural Network for sEMG-based gesture recognition in muscle-computer interface. In *Pattern Recognition Letters*, 2017.

[31] X. Zhai, B. Jelfs, R. Chan, and C. Tin, Selfrecalibrating surface EMG pattern recognition for neuroprosthesis control based on Convolutional Neural Network. *Frontiers in Neuroscience*, 11:379–390, 2017.

[32] Y. Li, N. Wang, J. Shi, J. Liu, J., and X. Hou, Revisiting Batch Normalization for practical domain adaptation. ArXiv e-prints, 2016.

[33] U. Côté-Allard, C.L. Fall, A. Drouin, A. Campeau-Lecours, C. Gosselin, K. Glette, F. Laviolette, and B. Gosselin, Deep Learning for electromyographic hand gesture signal classification by leveraging transfer learning. arXiv:1801.07756, 2018.

[34] F. Palermo, M. Cognolato, A. Gijsberts, H. Müller, B. Caputo, and M. Atzori, Repeatability of grasp recognition for robotic hand prosthesis control based on sEMG data. In *Rehabilitation Robotics (ICORR)*, pages. 1154–1159, 2017.

[35] Y. LeCun, Y. Bengio, and G. Hinton, Deep learning. *Nature*, 521(7553):436–444, 2015.

[36] L. Deng, J. Li, J.T. Huang, K. Yao, D. Yu, F. Seide, M. Seltzer, G. Zweig, X. He, J. Williams et al., Recent advances in deep learning for speech research at Microsoft. In *ICASSP*, 2013.

[37] A. Krizhevsky, I. Sutskever, and G.E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, 2012.

[38] A. Esteva, B. Kuprel, R.A. Novoa, J. Ko, S.M. Swetter, H.M. Blau, and S. Thrun, Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, 2017.

[39] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, Deepdriving: Learning affordance for direct perception in autonomous driving. In *ICCV*, 2015.

[40] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[41] V. Sze, Y.H. Chen, T.J. Yang, and J. S. Emer, Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proceedings of the IEEE*, 105(12):2295-2329, 2017.

[42] F.F. Li, A. Karpathy, and J. Johnson, Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition, 2018. `http://cs231n.stanford.edu/`. (Accessed on 28/02/2019).

[43] T. N. Sainath, A.R. Mohamed, B. Kingsbury, and B. Ramabhadran, Deep convolutional neural networks for LVCSR. In *ICASSP*, 2013.

[44] S. Levine, C. Finn, T. Darrell, and P. Abbeel, End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[45] K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition. In *CVPR*, 2016.

[46] S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[47] V. Nair and G. E. Hinton, Rectified Linear Units Improve Restricted Boltzmann Machines. In *ICML*, 2010.

[48] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, Caffe: Convolutional architecture for fast feature embedding. In *ACM International Conference on Multimedia*, 2014.

[49] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580, 2012

[50] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, Dropout: A simple way to prevent Neural Networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[51] S. Benatti, E. Farella, E. Gruppioni, and L. Benini, Analysis of robust implementation of an emg pattern recognition based control. In *BIOSIGNALS*, pages 45–54, 2014.

[52] S. Benatti, E. Farella, E. Gruppioni, and L. Benini, A prosthetic hand body area controller based on efficient pattern recognition control strategies. *Sensors*, 2017.

[53] C. Castellini, E. Gruppioni, A. Davalli, G. Sandini, Fine detection of grasp force and posture by amputees via surface electromyography, *Journal of Physiology Paris*, 103:255–262, 2009. Elsevier.

[54] LIBSVM – A Library for Support Vector Machines. `https://www.csie.ntu.edu.tw/~cjlin/libsvm/`. (Accessed on 26/02/2019).

[55] P. Kaufmann, K. Englehart, and M. Platzner, Fluctuating emg signals: Investigating long-term effects of pattern matching algorithms. In *EMBC*, 2010.

[56] S. Amsuss, L. Paredes, R. Zihlschlacht, N. Rudigkeit, B. Graimann, M.J. Herrmann, and D. Farina, Long term stability of surface emg pattern classification for prosthetic control. In *EMBC*, pages 3622–3625, 2013.

[57] J. He, D. Zhang, N. Jiang, X. Sheng, D. Farina, and X. Zhu, User adaptation in long-term, open-loop myoelectric training: implications for emg pattern recognition in prosthesis control. *Journal of neural engineering*, 12(4):046005, 2015.