UNIVERSITÀ DEGLI STUDI DI PADOVA

Department of Information Engineering

Master's Degree in Control Systems Engineering

CONVOLUTIONAL NEURAL NETWORKS FOR HAND GESTURE RECOGNITION WITH OFF-THE-SHELF RADAR SENSOR

Supervisor
Prof. Alberto Testolin

Co-supervisor
Prof. Jean Vanderdonckt
Université Catholique de Louvain

Master Candidate
Stefano Chioccarello

Student ID
2011656

Academic Year
2021-2022

UCLouvain

École polytechnique de Louvain

A special thanks to Professor Jean Vanderdonckt for the supervision during my staying in Louvain-La-Neuve (BE), and to Ph.D. Arthur Sluÿters for the help given with the development of the experimental part of this thesis.

Abstract

The presence of technology has become predominant in our lives, and the interest in the discipline of Human Computer Interaction (HCI) has increased enormously in the past few years. The necessity of actively interacting with devices brought the attention to the goal of easing their use and the contact with them; however, buttons, touch-screens, and other physical links can often be a bottleneck in the ease and fluency of their use. The challenge is thus to control such systems without the necessity of a physical contact, for example, by performing hand gestures in the air: movements are captured by sensors, and a specific action is triggered by each gesture. On the other hand, machine learning and artificial intelligence models are taking over any field for their versatility and high accuracy compared to more classical signal processing pipelines. This gave rise to the idea of exploiting machine (deep) learning also in the HCI field, especially for hand gesture recognition. The goal of this research work was to explore the use of advanced deep learning techniques to develop a system that can recognize and classify gestures starting from a custom set of movements acquired by means of the Walabot, an off-the-shelf Ultra Wide Band radar sensor.

The experimental part of this project was carried out at the Université Catholique de Louvain (UCL) in Belgium, in the context of the research training activity and the Erasmus experience, under the co-supervision of prof. Jean Vanderdonckt and Ph.D. student Arthur Sluÿters in Louvain-la-Neuve, and prof. Alberto Testolin, supervisor of this thesis. This manuscript provides an overview of the entire process, starting from data acquisition through the development of the model and to the results achieved.

Contents

Listing of figures

Listing of tables

1 INTRODUCTION

1.1 Context of the problem

It is undeniable that in recent years the presence of technology has had a huge impact on our daily lives. No matter what or where, most applications require a mutual interaction between humans and electronic devices. That is the reason why, in the last decade, the interest and studies on the subject of Human-Computer Interaction (HCI) have taken over the scientific community [1]. Many ways of interacting with machines exist, starting from **physical links**, such as buttons, keyboards, touch screen sensors, etc. However, sometimes this kind of instrument can **represent a bottleneck** for the correct functioning of the device, and do not allow an effective and user-friendly interface [1].

On the other hand, other kinds of interaction exist, such as human gestures and movements, which can be seen as a more natural way to interface with a machine. A gesture can be considered as any movement of any body part, such as the arms, hands, and face, to convey information without the necessity of speaking or touching any surface. Among the different parts of the body, hand gestures are widely used to build interactive applications [2, 3]. Hand gestures are an important part of non-verbal communication in our daily life, and we extensively use them for communication purposes such as pointing towards an object, conveying information about shape and space. Using **hand movements as input** instead of a keyboard and mouse can help people communicate with computers in a **more intuitive and easier way** [4]. Hand gestures used in this type of system should be simple and intuitive. Among the set of gestures that were taken in consideration for this project, we find, for example: "Open/close hand"; "Swipe up/down/left/right"; "Extend one/two/three/four fingers; "Draw a circle"; "Draw a letter", etc. These simple movements recall, for example, some finger actions that a person performs when using a smartphone or a touchscreen in general. The idea is to replicate these movements in the air without the need for a physical interaction. The use of hand movements as input to a human-computer interface can be extremely powerful; however, it also presents a set of problems and variables that can strongly influence the overall performance of the system. In fact, it must be considered that **all body parts** involved in those movements **are different for each user**, **distances from the sensor can vary** and **surrounding environment may change**. Therefore, it is necessary to build a strong and robust system to efficiently recognize hand gestures, regardless of the conditions and different variables that play a role in the task. The operation of hand gesture recognition can be considered as a pipeline-structured process in which information is conveyed through different steps that aim to capture the gesture, process it, and ultimately correctly classify the human movement. It goes without saying that each step plays an essential role in the overall quality of the system, therefore, each element must work correctly, from the acquisition system to the classification module.

Among the different approaches that could be taken for the classification of the gesture, the idea of applying machine learning models seems quite promising. This is because the great variability in hand movements can be neutralized by the strength that machine learning has demonstrated over the last few years in the countless applications where it is implemented [5, 6, 7]. More classical approaches are more sensitive to the variations of a same gesture between different users, and so they could be more likely to fail in recognizing gestures. The subfield of deep learning offers a variety of state-of-the-art models and optimization techniques that seem propitious for the level of complexity of this task [8, 9]. The purpose of this thesis is therefore to **explore the use of deep learning techniques** to develop a framework capable of **recognizing a set of hand gestures** acquired **through a radar sensor**. The choice of radar sensing as the acquisition system adds an additional variable to the problem. Usual systems such as cameras allow for more information on the gesture, and hence higher performances would be expected. On the other hand, **the complexity and difficult interpretation of radar signals** makes this task even more **challenging for gesture recognition**, but interesting results have already been achieved in this field with this technique [10].

1.2 Project statement

1.2.1 Statement

As anticipated before, the goal of this work is to try out deep learning-based approaches to tackle the task of recognizing hand movements captured with an off-the-shelf radar sensor.

- **Deep learning-based approaches**. Among all the different machine learning strategies and branches, a supervised deep learning model has been chosen for this project because of the enormous potential of this approach. It goes without saying that, especially during the last years, deep learning has taken control over a countless number of applications, for it can be applied in almost every field. In fact, no matter what kind of data is to be processed, the essential requirement for developing a good model is to have a huge amount

of them. The size of the dataset must be as large as possible, because this allows the model to adjust itself in order to fit the data as best as possible [11, 12]. It is also necessary to specify that the task in question is a classification task, which means that the goal is to train a machine learning model that is capable of correctly predicting the label of the input. For this reason, a supervised learning approach, which allows the model to train knowing the correct class of each input data, is preferred over an unsupervised one.
Among the numerous architectures of supervised deep learning, the choice fell on Convolutional Neural Networks (CNNs) [13]. The main reason for this is that CNNs are particularly efficient with multidimensional signals and data, and, considering that this approach involves the use of images, it was pretty natural to adopt this technique.

- **Radar sensor**. In this project, the use of a radar sensor was chosen over other sensors, such as cameras or LIDARs, for many reasons. There are various advantages of using a radar sensor:

  – It does not require particular light conditions: the radar sensor's transmission of radio waves is not affected by visibility, lighting, and noisy effects presents in cameras. Therefore, the radar performance is consistent with all environmental conditions.

  – Radar data are less computationally heavy than images from a camera; hence, a less powerful framework would be necessary to process data in a real-time structure.

  – It can pass through walls and objects, which a camera would not be able to.

  – It respects the privacy of the user, since radar data do not represent reality through pixels, as is done with images.

  On the other hand, there are some drawbacks in the use of radar sensors for the proposed task, among which we find:

- – There is less information and data available with respect to a camera.
- – Radar images are not optical images and are of very low resolution, since they are limited by the carrier frequency, size of the radar aperture, Doppler and clutter.
- – It is highly influenced by movements around the target person and it is more difficult to remove the background and outer elements with respect to an image, for which many image processing algorithms are available and working well.
- – If not set properly, might be less accurate than a camera acquisition system, since it is more difficult to recalibrate the radar sensor than a camera.

Radar sensing for gesture recognition has been practised in recent years [14] and represents an innovative acquisition system with respect to classical frameworks, which allows one to explore different approaches for the recognition task proposed in this thesis.

- **Off-the-shelf**. The "off-the-shelf" term is related to the fact that the radar sensor used, i.e. the Walabot [1], is a product that comes "ready-made" to the user, and it is ready to work with a minimum level of setup. This means that there are only reduced calibration settings to adjust the device from time to time, but overall, the hardware is already available to use from the beginning. It comes without saying that, if a custom device built for the specific purpose of gesture recognition were used, it could be that the performances of the recognition system would change significantly, but then, in a future perspective, it would be harder for it to become marketable and too expensive in case of high scale production. From this intuition, the choice of using

[1]https://www.walabot.com/

an off-the-shelf radar, which comes already with a custom SDK, is supported and updated, and with a rather large online community, is more appealing than building the entire acquisition system from scratch.

- **Hand gestures recognition**. The task proposed in this manuscript is a rather widely discussed problem in the human-computer interaction branch. The realization of a hand gesture recognition system opens the path to a natural, innovative, and modern way of non-verbal communication, and the field of application is very broad. Let us just think of the domotics environment to control objects in the house (opening/closing windows, turning on/off household electrical appliances, etc.), or the automotive industry (controlling infotainment systems while driving without touching any screen or button), or even the medical field to control machines and tools at distance without physically interacting with the patients, etc. The applications in which a hand gesture recognition system can be employed are various. For example, in the very recent work of Şiean et ai. [15] a TV gesture-based control was addressed by means of radar sensing and hand gestures, in particular with the Walabot device. Following similar ideas, the aim of this thesis is to explore possible approaches to solve the gesture recognition task. Let us precise that the gestures involved in this project are a limited number, and some important assumptions have to be made.
  First of all, the system is built to recognize gestures performed only by the right hand of the user. It is very complicated to extend the task to both hands, and it is a problem that may be tackled in a future development of this system. Furthermore, all gestures and samples collected during the data acquisition process were performed under controlled conditions. In particular, the user surrounding environment and, more importantly, the distance between the user/hand and the sensor were two key elements to keep in mind. These restrictions are due to the resolution of the radar, which works best when the gesture is performed at a certain distance, and in order to receive a cleaner

reflection signal, the fewer objects surround the user, the better it is. Second, the set of gestures selected for this classification problem is, again, restricted to a limited number of 20 gestures, which, of course, could then be extended in a future work scenario. However, the main and most naturally movements, such as swiping up/down, closing/opening the hand, indicating numbers with fingers, etc. are taken in consideration here. Therefore, at least the most intuitive gestures were included in this dataset. The complete list of hand movements is presented later in the thesis.

1.2.2 Research questions

Many variables come into play in this project, as can be seen in the project statement. The main focus of this thesis is on hand gestures captured by means of a radar sensor and recognized with a machine learning approach. Several steps detailed in each chapter of this thesis analyze accurately each of those elements, finally answering the following research questions that drove this project from the beginning:

- **RQ1**. *Is a radar sensor able to accurately capture hand and arm movements?*
- **RQ2**. *Is deep learning suitable for this classification task?*
- **RQ3**. *Is the framework able to generalize well and recognize gestures performed by new users that were not included in the training set?*

The use of a radar sensor to deal with this recognition problem is quite recent in the scientific community of HCI [16, 10]. For this reason, the expected results were unpredictable from the beginning, and any scenario would have been acceptable both in case of success and failure. In the following chapters, all these questions find an appropriate answer.

1.2.3 Working hypotheses

We assume the following working hypotheses for this thesis:

- **Hypothesis 1.** *Gestures set definition*: The gesture set was built according to the performance of the radar sensor. A device which is acquiring data in the same way as the Walabot can be taken into account, but not other devices.

- **Hypothesis 2.** *Context of use*: The device must be used in the same environment as when samples were collected and tested. It must also be used in the same position in front of the user and at the same distance.

- **Hypothesis 3.** *Offline recognition system*: The PC performance does not influence the execution time and recognition rate of the system, the process is not optimization-oriented.

- **Hypothesis 4.** *Variability in the gestures*: The data sets were collected by different users. Therefore, a sample may vary according to the speed and the user's understanding of the gesture.

- **Hypothesis 5.** *Dataset*: We suppose that inside the dataset, there should be enough different templates to ensure variation depending on each user. Therefore, every user can be considered as the target audience for the testing part.

1.2.4 Research method

Let us give a brief overview of the main steps taken during the entire duration of the experimental part of this project, which are detailed in the following chapters.

1. Data collection [Chapter 3]: sample collection process with different users and participants to gather data for model training;

2. Data processing [Chapter 4]: signal processing section regarding the preparation of the data for the machine learning model;

3. Model development [Chapter 5]: definition and implementation of the deep learning architecture;

4. Model training and validation [Chapter 6]: training process and validation of the model to obtain relevant results in terms of performances, classification accuracy, and other evaluation metrics.

The experiment can be considered to have a pipeline structure in which the data, starting from scratch, are processed and evaluated through different steps to finally recognize the performed movement. The steps anticipated in this subsection can be observed in Figure 1.1.

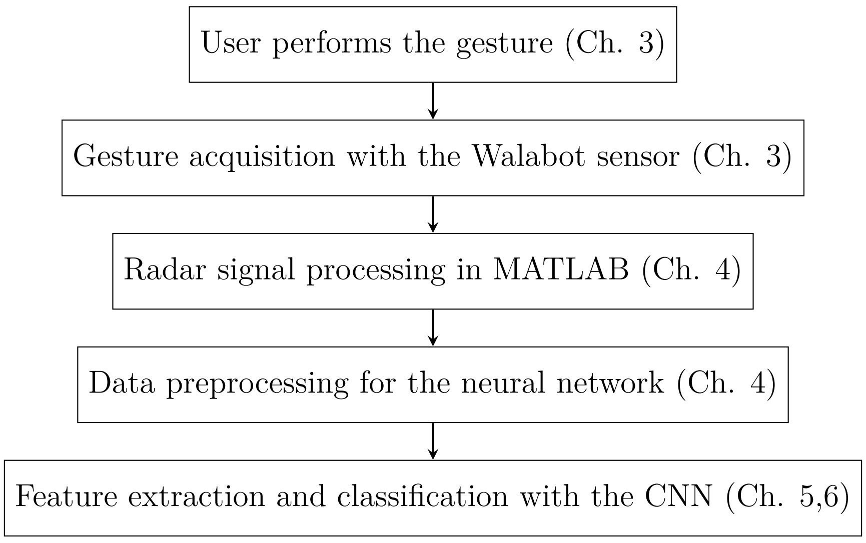

User performs the gesture (Ch. 3)
Gesture acquisition with the Walabot sensor (Ch. 3)
Radar signal processing in MATLAB (Ch. 4)
Data preprocessing for the neural network (Ch. 4)
Feature extraction and classification with the CNN (Ch. 5,6)

Figure 1.1: Main steps of the process from the data acquisition to the classification of the gesture

1.3 Overview of the thesis

To conclude the introductory part, here is a brief summary of the content of the following chapters.
In Chapter 2 the literature review done previously for the actual project is described, with references to related projects that used very different or very similar approaches to this, so that an overview of the current state is given. In addition, some details of the implementation regarding the device used and tools and software can be found at its end.
In Chapter 3 the whole data acquisition procedure is outlined, together with details about the dataset and the setup for the recording of the gesture samples.
In the following chapter, namely Chapter 4, all details regarding data processing can be found. Since the signal comes from the antennas in raw format, all the steps necessary to elaborate it to obtain significant data are specified there.
After that, Chapter 5 explains the model exploited for this task, together with the initial ideas and the different tests that led to its final version. An overview of the methodologies used is given as well.
Chapter 6, instead, regards the whole training and validation part of the aforementioned model. The results of the entire classification process are presented.
To conclude, we find Chapter 7 which summarizes the contributions, gives a critique analysis of the model with advantages and disadvantages of the choices taken, and their possible improvements. Finally, a brief overview of future work and what could be done with more time and resources is given in order to conclude the manuscript.

2 RELATED WORK

2.1 Targeted literature review

In this section, a brief review of relatable projects is performed to give an idea of the current literature about this topic, and what could be done to improve already available hand gesture recognition systems. The biggest differences in the numerous publications in this field can be found in the acquisition system, which determines the kind of data that is available, and in the recognition part, that can be tackled with different approaches, some of which are signal based, while others are feature and machine learning based. The analysis performed on the literature was tackled by differentiating the radar-based approaches from the rest.

2.1.1 Cameras and other sensors

The common hand gesture signal acquisition approaches today are cameras [17], infra-red sensors [18], and ultrasonic sensors [19]. On the other hand, radar sensors are recently and increasingly emerging due to their high recognition performance even in poor lighting conditions and complex background. In addition, low-cost commercial radar sensors are becoming widely available, and are capable of acquiring the peculiarities of hand movements which can yield high classification accuracy at lower processing costs [20].

One of the most common gesture acquisition approaches is based on image recognition and tracking by camera sensors [21, 22, 23]. With this kind of method, the target object (hand, fingers, eye, etc.) is captured and encoded into representative features using image / video processing [24]. The subsequent part, that is the recognition system, can be tackled by means of different techniques. One possible choice is that of using template matching approaches [25, 26]. In this case the idea is to use one or more models of the gesture as the principal template, and then compute the distance between the template and the analyzed gesture. returning the type of template with the shortest distance.

More recently, instead, the development of newer techniques and abundance of data brought the machine learning world into play with very interesting results. In this scenario, starting from the processing of the acquired gesture, the resulting feature parameters are fed into a classification system machine learning entity, which is trained to identify the unique set of signatures generated by each gesture and to classify it accordingly [27, 28]. Usual computer vision and image processing algorithms can be used for extracting significant features for each gesture [29, 30], however the trend is moving more and more towards the use of convolutional neural networks, due to the 2-D nature of the signal [31, 32, 33]. It is undeniable that the use of machine learning in this field, and CNNs in particular, enhanced greatly the recognition performances, as reported in Table 1 of [34], where it is shown how many approaches reach high accuracy in classifying gestures with such a technique, for example with [35, 36, 37].

However, for these methods to properly work, the classifier usually requires high quality images, which can be challenging especially when image acquisition takes place in a noisy background environment or poor lighting conditions.

2.1.2 Radar sensors

On the other hand, when using radar sensors for gesture sensing, the adverse effects of ambient environment noises such as lighting conditions, dust, complex backgrounds can be minimized. Moreover, the signal processing cost is lower than image/video based methods. Radar sensors use electromagnetic waves in certain spectrum bands, depending on the bandwidth of the antennas, which are unaffected by the ambient light. Furthermore, a radar is capable of instantaneously capturing the rich Doppler information associated with the movements of hands and fingers, thanks to their reflection. In the case of camera-images the task of velocity estimation requires a significant processing effort, with sometimes very low accuracy results [38]. Low-cost Doppler radar sensors are becoming widely available, thanks to the recent advancements in RF microelectronic technology. Doppler radars detect micro-Doppler signatures caused by electromagnetic signals reflected from moving non-rigid objects, or body parts, such as hands, fingers, wrists, ankles etc.
Based on the nature of the transmitted signals, radar technologies can be classified into two categories [39]:

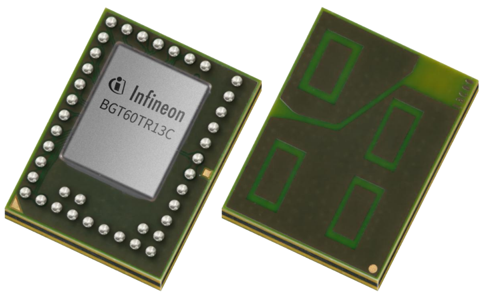
- Pulsed Radar;
- Continuous-Wave (CW) Radar.

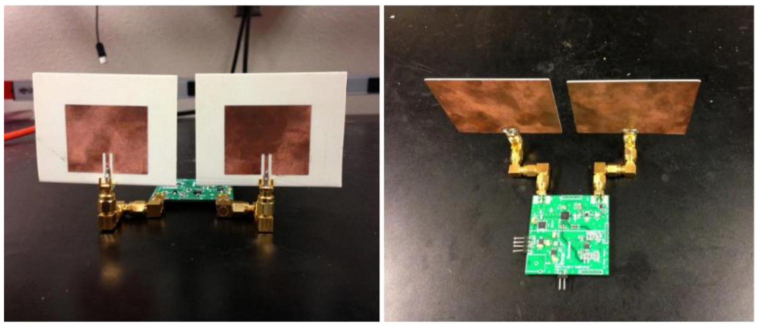
CW radars can then be further be classified as Frequency-Modulated CW (FMCW) radars and Single-Frequency CW (SFCW) radars, based on their capability of transmitting and receiving signals on a single-tone frequency level or at varying frequencies. In both cases, the important thing is that the Doppler phenomenon can be exploited [40]. The Walabot [41], for example, can be classified as a FMCW radar, for it works on a wide range of frequencies and it is not limited to a single value.

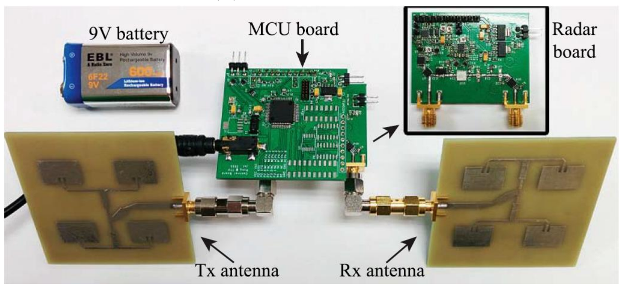
Many different radars have been used in many projects, each of them for a particular task and set of conditions. In Table 2.1 some radars frequently used in radar sending and HCI are reported, together with some features such as the bandwidth, the type of antenna, the power usage, the price etc.

Vendor	**Device**	**Band**	**Interface**	**Raw Data**	**Antenna**	**Power**	**Price (eval. Board)**	**Applications**	**Other**
Vayyar	Walabot 3D Sensor [41]	6.3-8 GHz (EU) 3-10 GH (USA)	USB	Yes	18 Antenna Array	5V	$250	Breath monitoring, object tracking, in-wall pipe detection	-
	Vayyar Care Device Walabot Home [42]	57-64 GHz	Voice Calling, Touchscreen Interface	No	24Tx/22Rc	Around (5V, 0.75 A)	$250	Fall detection at home	Requires Alexa. A complete product enclosure.
Novelda (Xethru Chip)	X4M02 [43]	7.29 or 8.748 GHz	UART, SPI, 12C	No	SISO	Around 600 mW	$425	Radar Development Kit	-
	X4M200 [44]	6.0 - 8.5 GHz	USB (UART)					Respiration	
	X4M300 [45]							Presence and Occupancy	
Innosent	IPM-170 / IPM-365 [46] / IPM165 [47]	24GHz	SPI	No	SISO	Around (5V, 0.3A)	-	Door openers, security applications and industrial applications	-
	INS-333X						$42	Velocity, direction, distance of moving objects	-
Infineon	BGT60LTR11AIP [48]	60GHz	USB (UART)	No	1Tx/3Rx	-	$175	Motion sensor	Demo board and IC are on sale
	BGT24LTR11 [49]	24GHz			4Tx/1Rx		$218	Movement detection, proximity & presence sensing	-
Inras	RadarBook2 [50]	10/24/77GHz	ETH	Yes	8Tx/16Rx	-	-	Automotive, Movement detection	-
TI	IWR6843ISK [51]	60GHz	USB(UART), ETH	Yes	3Tx/4Rx (up to 12x16)	around (5V, 0.4A)	$150	Presence, occupancy, motion, vital signs, target range/ Doppler/ angle information	In connection with DCA1000EVM raw radar data can be collected; Various antenna configs, incl AoP.
Silicon Radar	EVALKIT SiRad Easy r4 [52]	120GHz	USB (UART)	No	SISO	around (3.3V, 0.38A)	-	Presence, occupancy, motion, vital signs	Also supports a 24GHz FE MIMO.
iMotion	iMotion / iMotion 2/ iMotion 3 [53]	2.4GHz	LabVIEW script	No	Patch Antenna Pair	Around 5V	-	Vital Signs Detection, Through Wall Detection	-

Table 2.1: Radars frequently used in radar sensing and HCI


(a) Infineon ™XENSIV 60 GHz radar chip [1]


(b) iMotion Radar [2]



(c) iMotion3 Radar [3]

Figure 2.1: Examples of radars used in gesture recognition tasks

As stated before, Doppler radar is being increasingly studied for gesture recognition due to its high sensitivity to small movements and excellent ability to distinguish non-stationary objects from a stationary background.
In order to recognize and classify signatures of hand gestures, numerous techniques

[1]Source: https://www.infineon.com/cms/en/product/sensor/radar-sensors/radar-sensors-for-iot/60ghz-radar/bgt60tr13c/
[2]Source: https://sites.google.com/site/imotionradar/imotion
[3]Source: https://sites.google.com/site/imotionradar/imotion3

have been applied such as machine learning, principal component analysis, and differentiate / cross-multiply algorithms [38, 54, 55, 56]. Conventional supervised machine learning extracts and classifies gestures using predefined characteristic parameters that are the features [57, 58]. However, the optimal features in many cases are unknown. Therefore the performance of the classifier varies significantly depending on the selected ones. On the other hand, deep learning algorithms, which use multiple layers of filters, such as convolutional neural networks, do not require predefined features. Instead, the network self-learns them from an input signal during the training process [59]. Training and classification using CNNs is a promising approach in gesture recognition problems as it eliminates the need for predetermining the set of features.

In Table 2.2, taken from [10], are reported some relevant results obtained in the last decade in the context of HGR by means of a Doppler radar sensor. In most of these works, instead of trying to extract rich features directly from the radar recorded hand gesture signals, the input de-noised data are represented in a suitable format, and then a deep-learning-based classifier is used. It can also be observed how, over the years, for the same kind of sensor data, the trend is shifting from feature-based classification with classical machine-learning methods to deep-learning based classification.

Study and Year	**Data, Representation and Data Dimensions**	**Algorithmic Details**	**Frequency**	**No. of Gestures**	**Distance Between Hand & Sensor**	**Participants and Total Samples Per Gesture**	**Number of Radars**
Kim et al. [58] (2009)	TimeFrequency (3D; radar signal was passed through a STFT)	SVM	2.4 GHz	7 (including activities)	28 m	12, Not specified	1
Zheng et al. [60] (2013)	TimeRange (1-D; hand motion vector)	Differentiate and Cross-Multiply	N/A	Not applicable	01 m (tracking)	Not applicable (tracked hand)	2 and 3
Wan et al. [61] (2014)	TimeAmplitude (1D)	*kNN* $(k = 3)$	2.4 GHz	3	Up to 2 m	1, 20	1
Fan et al. [62] (2016)	Positioning (2D; motion imaging)	Arcsine Algorithm, 2D motion imaging algorithm	5.8 GHz	2	00.2 m	Did not train algorithm	1 (with multiple antennas)
Gao et al. [63] (2016)	TimeAmplitude (1D; A barcode was made based on zero-crossing rate)	time-domain zero-crossing	2.4 GHz	8	1.5 m, 0.76 m	Measured for 60 s to generate the barcode	1
Zhang et al. [64] (2016)	TimeDoppler frequency (2D)	SVM	9.8 GHz	4	0.3 m	1, 50	1
Huang et al. [65] (2017)	TimeAmplitude (1D)	RangeDoppler map (RDM)	5.1, 5.8, 6.5 GHz	2	0.2 m	Not applicable (hand-tracking)	1
Li. et al. [66] (2018)	Time-Doppler (2D)	NN Classifier (with Modified Hausdorff Distance)	25 GHz	4	0.3 m	3, 60	1
Sakamoto et al. [67] (2017)	Image made with the In-Phase and Quadrature signal trajectory (2D)	CNN	2.4 GHz	6	1.2 m	1, 29	1
Sakamoto et al. [68] (2018)	Image made with the In-Phase and Quadrature signal trajectory (2D)	CNN	2.4 GHz	6	1.2 m	1, 29	1
Amin et al. [69] (2019)	TimeDoppler frequency (3D RGB image)	*kNN with* $k = 1$	25 GHz	15	0.2 m	4, 5	1
Skaria et al. [70] (2019)	TimeDoppler (2D image)	DCNN	24 GHz	14	0.10.3 m	1, 250	1
Klinefelter and Nanzer [71] (2019)	TimeFrequency (2D; frequency analysis)	Angular velocity of hand motions	16.9 GHz	5	0.2 m	Not applicable	1
Miller et al. [72] (2020)	TimeAmplitude (1D)	*kNN with* $k = 10$	25 GHz	5	Less than 0.5 m	5, Continuous data	1
Yu et al. [73] (2020)	TimeDoppler (3D RGB image)	DCNN	24 GHz	6	0.3 m	4,	1
Wang et al. [74] (2020)	TimeDoppler (2D)	Hidden Gauss Markov Model	25 GHz	4	0.3 m	5, 20	1

Table 2.2: Summary of HGR algorithms for SFCW (Doppler) radar.

Apart from the aforementioned machine learning approaches applied to radar based gesture recognition, in the literature it is possible to find also examples that do not exploit this kind of data-driven processes. Let us focus in particular on the work [75] carried out by Arthur Sluÿters, together with and Sébastien Lambot and Jean Vanderdonckt, that is the co-supervisor of this thesis. In this research in particular, a novel data processing pipeline for hand gesture recognition was developed, in which an advanced full-wave electromagnetic modelling of the radar signal was created. This process was performed in order to classify the gesture without the necessity of collecting a big amount of data, which is instead a requirement for the correct behaviour of a machine learning model. In this case, however, a particular set of physical features was extracted in advance by exploiting the knowledge of the radar antenna model and the Green's functions for the normalization of the signal. Such framework differs from the purpose of this thesis because of the nature of the classifier, that is not entirely machine learning based. Nevertheless, it is essential to mention it for the reason that some data pre-processing steps (Chapter 4) are shared.

2.1.3 Walabot

When it came down to choosing the sensor to be used for the experiment, many were the reasons to choose the Walabot [41] radar sensor over other existing devices. First of all, it had to be considered the type of gestures that wanted to be recognized. In fact, depending on the kind of movement that the user has to perform it is necessary to look for the most suitable sensor, keeping in consideration also the distance at which it is desired to recognize such gestures. For instance, the Google Soli [76, 77] radar is a millimetre-wave radar that is widely used in HCI and gesture recognition. However, its architecture makes it suitable for miniature gesture sensing rather than medium/long range movements. For this reason, even if it is highly supported and studied in the scientific literature, and many promising results were achieved with it [78, 79, 80], it was still decided to exclude it being out

of scope of this project.
The choice was hence highly influenced by the type of gestures we wanted to focus on, that are movements performed at a medium range $(60 - 70cm)$ from the radar. Among the different off-the-shelf devices present on the market for this kind of application, the final decision fell onto the Walabot radar for various reasons. First of all, compared to other devices, it is cheap and supported by the developers and a rather wide user community. Moreover, many configurations are possible for its use, making it versatile and handy for many uses. It is portable and does not require any complex setup. Finally, it comes with a C++ API that is directly usable for retrieving raw data coming directly from the antennas, which can be processed in a second moment.
From a scientific point of view, the Walabot has already been used for different tasks, such as object material recognition [81], activity recognition [82, 83] and, more recently as mentioned above, also for hand gesture recognition [75, 84]. However, apart from the last one, the other implementations regarded static recognition, while in this case the idea is to apply such techniques to solve a dynamic task that is hand gesture recognition.

2.1.4 Gesture dataset

Together with the whole acquisition and processing of the data, it is also important to give details about what is encoded actually in the data in question. In order to train a machine learning model, and a deep learning model in particular, the main requirement is the abundance of data. This is essential for the network to be able to learn the features of each gesture that will be then used for the classification part. As the literature suggests, there exist already many datasets regarding hand gestures, but those are mainly vision and camera based. Among those it is possible to find the Cambridge Hand Gesture Database (released in 2009) containing nine hundred sequences of images for nine different hand gesture classes [85], MSRGesture3D (released in 2012) [86], and EgoGesture [87]. Moreover, the RGBD-HuDaAct [88]

dataset provides a human activity recognition dataset acquired with a video camera and a depth sensor.
However, regarding radar based gesture datasets, there was none until 2021, when the UWB-gestures Dataset [4] was released. The dataset contains a total of 9,600 samples gathered from eight different human volunteers. The will of the authors is to eliminate the need to employ UWB radar hardware to train and test the algorithm. Additionally, the idea is to provide a competitive environment for the research community to compare the accuracy of different hand gesture recognition (HGR) algorithms, enabling the provision of reproducible research results in the field of HGR through UWB radars.
Despite the availability of this dataset, it was still decided to not use it in favour of a custom one created on our own. The main reasons for this were two. First, the acquisition setup of the UWB-gestures dataset was different from the one that we had at disposal, since we used another device and it was difficult to reproduce the setup proposed in the aforementioned paper. Second, using another set of gestures such as that would have posed some limits to the kind of movements that we wanted to recognize and investigate, since it was desired to have an higher number of gestures and many of them were very different from the ones proposed in [4].
The sum of all these factors brought to the creation of a custom dataset, that is actually an extension of the one used in the solution proposed by [75]. The whole dataset creation process and acquisition of the data in reported in Chapter 3.

2.2 Walabot device architecture

Let us now provide some technical information on this device. First of all, there exist different versions available on the market: Starter, Creator, Developer (Table 2.3. The one used here is the Walabot Developer [89, 41]. It consists of a compact off-the-shelf ultra-wideband (UWB) frequency-modulated continuous-wave (FMCW) radar. Its dimensions are $144mm \times 85mm \times 18mm$ ($5.67in. \times 3.35in. \times 0.71in.$).

Capability \ Model	Walabot Starter	Walabot Creator	Walabot Developer
Physical			
Number of antennas	3	15	18
Board size	72 mm * 48 mm	72 mm * 140mm	72 mm * 140mm
External powering option	No	Yes	Yes
Software API capabilities			
Basic API functions	Yes	Yes	Yes
2D acquisition	Yes	Yes	Yes
3D acquisition	No	Yes	Yes
Multiport recorder (raw data)	No	No	Yes
Software application capabilities			
Breathing detection	Yes	Yes	Yes
Object detection	Yes	Yes	Yes
Short range imaging	No	No	Yes

Table 2.3: Feature summary of Starter, Creator and Developer models [4]

This device can be connected to a smartphone, tablet, or computer via a single USB cable. It exists in two versions: a US/FCC version operating in the 3.3-10 GHz frequency range and an EU/CE version operating over the narrower 6.3-8 GHz rangew, which we used in this work. The Walabot Developer consists of an array of 18 antennas, including four used as transmitters (depicted in red in Fig. 2.2), the rest being only receivers (depicted in green in Figure 2.2).

Depending on the configuration, it senses motion data with up to 40 pairs of antennas. The Walabot SDK provides two different profiles for distant scanning, which define the set of antenna pairs (right part of Fig. 1), the number of fast-time samples per frame, and the frame rate:

[4]Source: https://site.walabot.com/docs/walabot-tech-brief-416

	Bandwidth	Price			Average Transmit Power
		Starter	Creator	Developer	
American Version	$3.3 - 10GHz$	\$ 74.95	\$ 99.95	\$ 599.95	$< 41\frac{dBm}{MHz}$
European Version	$6.3 - 8GHz$	Not Available	$\sim$ 180 €	$\sim$ 300 €	$< 41\frac{dBm}{MHz}$

Table 2.4: Comparison between American and European version of the Walabot

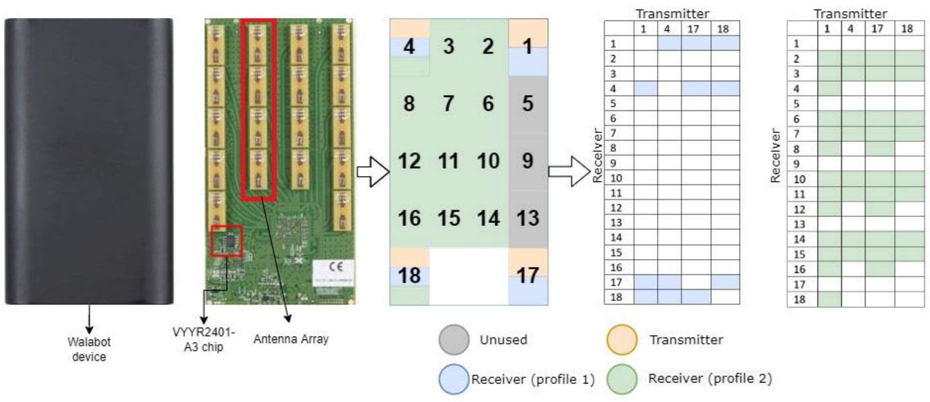


Figure 2.2: Walabot Developer: device, antenna array, antenna IDs, and antenna pairs IDs used in profiles 1 and 2.

- Profile 1 (PROF_SENSOR): 40 antenna pairs, 8192 fast-time samples/frame, approximately 20 frames/s
- Profile 2 (PROF_SENSOR_NARROW): 12 antenna pairs, 4096 fast-time samples/frame, approximately 41 frames/s

The second profile is the one employed in this application.
Basically, the Walabot is a programmable 3D sensor that detects objects using radio frequency technology that breaks through known barriers. It uses an antenna array to illuminate the area in front of it, and sense the returning signals. The signals are produced and recorded by the *VYYR2401 A3 System-on-Chip* integrated circuit. The data is communicated to a host device using a USB interface, which is implemented using the *Cypress* controller.
In Figure 2.3 it is possible to see: on the left the backside of the board, with the VYYR2401 chip, the USB controller and the micro-USB connectors; on the right the antennas array.
The Walabot senses the environment by transmitting, receiving and recording signals from multiple antennas. The broadband recordings from multiple transmit-receive antenna pairs are analyzed to reconstruct a three dimensional image of the surrounding space. The analysis of sequences of images allows to detect changes in the environment. Walabot is capable of short-range imaging into dielectric environments, while is not suited for long-range ($\geq 1m$) detection. In Figure 2.4 a sketch of the functioning of the Walabot is shown. The host device may be a computer, a smartphone or any other that can transfer data via USB. The acquisition board is then sending and receiving radar waves in order to sense the environment. On the other side, instead, we have a high level block diagram of the Walabot depicted in Figure 2.5 where the acquisition board that exchanges data via USB is detailed. The raw signal from the antenna array gets transmitted to the integrated circuit block, identified with the "VYYR2401" block, which in turn exchanges information with the micro-controller that is communicating with the host.

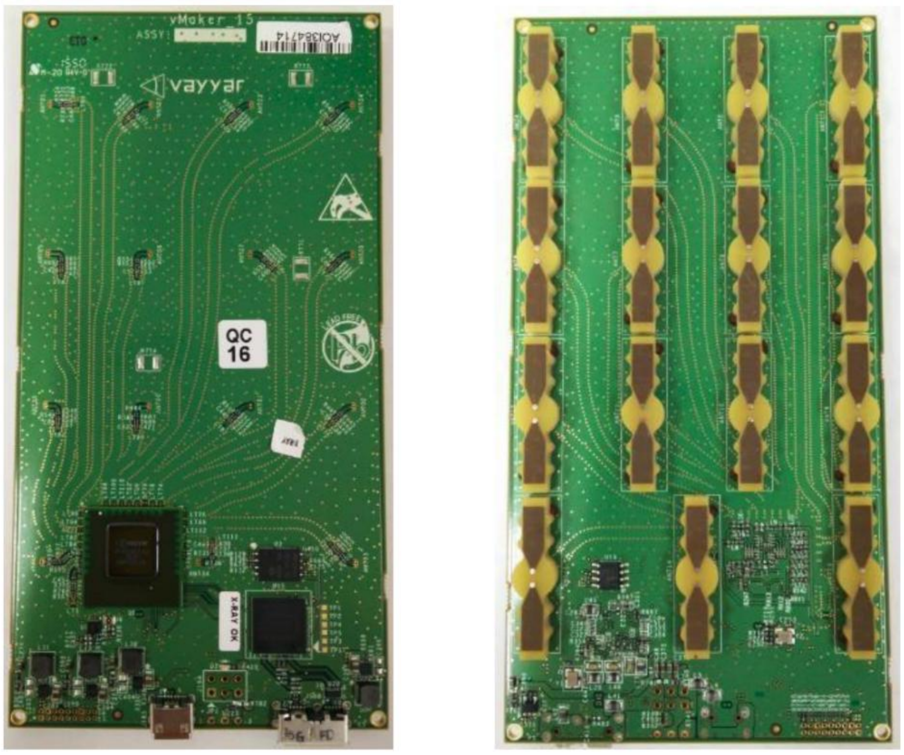


Figure 2.3: Integrated circuit and antennas of the Walabot

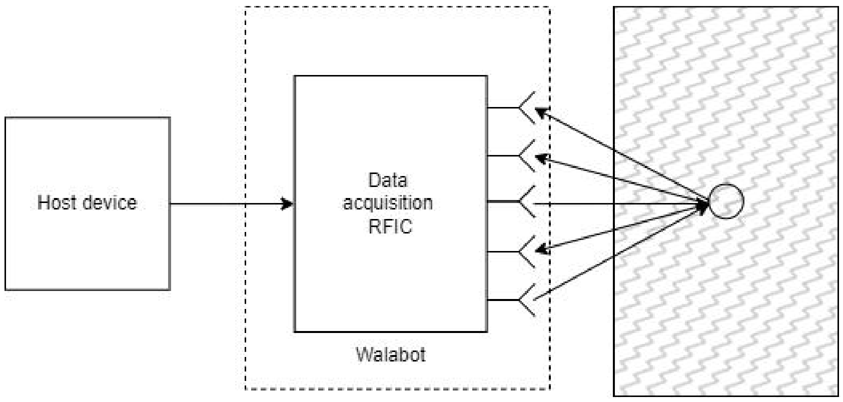


Figure 2.4: Basic functioning of the Walabot

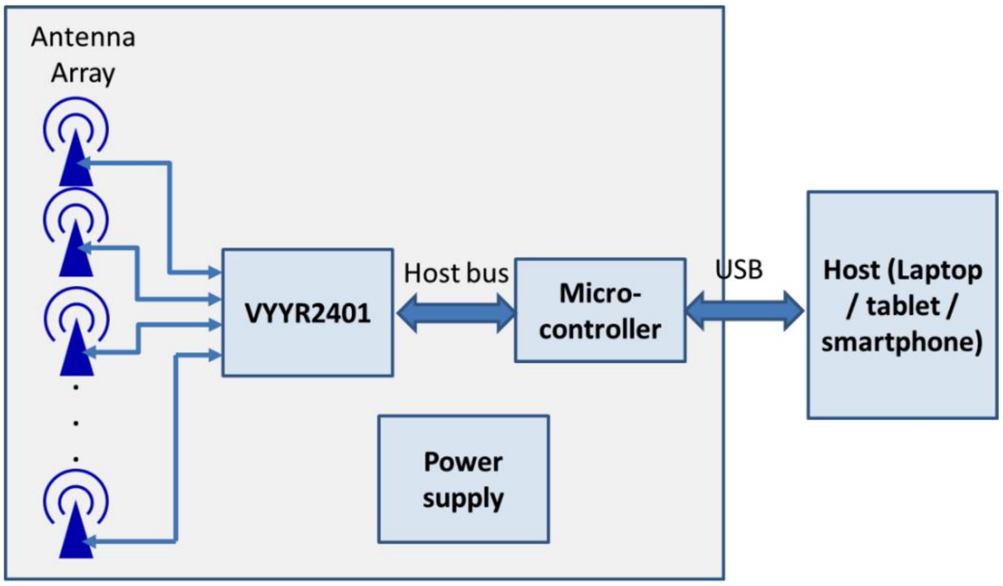


Figure 2.5: High level block diagram of the Walabot acquisition board

2.3 Tools and software

Let us give an overview of the tools, programming languages and software used for the development this project. The signal coming from the radar sensor was used in its raw format, taking it directly from the antennas. The Walabot radar sensor comes with a custom SDK [5] that provides access to this kind of data, and by means of a C++ script it was possible to record and acquire the gestures almost automatically. Regarding the signal processing part for the radar sensor output, the MATLAB [6] software (ver. *2021b*) was employed to process the signal and obtain the input data for the convolutional network. For the machine learning model, the Python [7] programming language (ver. *3.8*) was chosen for its wide range of machine

[5]https://site.walabot.com/getting-started
[6]https://it.mathworks.com/products/matlab.html
[7]https://www.python.org/

learning oriented libraries, data manipulation easiness, versatility and readability, and very rich graphic options. Regarding the specific libraries used, let us briefly report them here:

- *Pytorch* [8]: main framework for the deep learning part. It was used for its versatility and easiness of use, the possibility of parallel computing, the many implementations with optimizers and regularizers. It was chosen over other libraries (Keras, Tensorflow etc.) because it represented a good compromise both in terms of usability, stability and performances;
- *scikit-learn, pandas*: machine learning oriented libraries, used to perform model validation and obtaining important metrics about it;
- *os, shutil, pickle, json*: libraries necessary to manage filenames, data, dictionaries etc.;
- *numpy, random*: to work with arrays, data structures, generate random numbers etc.;
- *matplotlib*: to generate graphics and plots;
- *datetime, tqdm, itertools*: other utility libraries.

[8] https://pytorch.org/

3 DATA ACQUISITION

In this chapter, every detail regarding the gestures and data used later for the recognition part is presented. First of all, an overview on the gesture set chosen for this project is given. Second, the acquisition setup and process is explained.

3.1 Gestures definition

As anticipated in Section 2.1.4, despite the availability of a few online datasets it was still decided to create a new collection of gesture samples. This was done mainly because it gave us the possibility to choose our own set of hand gestures; moreover, we could create our own recording set up, which could then be easily reproduced for future work. Let us now provide some information about the choice of the gestures.
First of all, the choice of using hands as a way of communication finds its reasons in the human nature. Human gestures are a nonverbal medium of interaction mode and can provide the easiest and most intuitive ways to interact with computers.

Compared to other parts of the body, the human hand, which has been considered a natural means of human-to-human interaction, has been used widely for gesturing and can be the best suitable for communication between the human and the computer [90]. There are several typical applications of hand gesture recognition such as virtual game controller [91], sign language recognition [92], directional indication through pointing, [92], robot control [92], lie detection[92], etc. The increasing interest in this field has led researchers to do a large number of research which has been endured in a number of surveys given in [90]. These surveys are directly or indirectly related with hand gesture recognition. Studies such as [93] have shown how people tend to perform more naturally some gestures with respect to others, depending on the context. Moreover, it has to be said that a radar sensor could be potentially used to detect any kind of gesture, also coming form other limbs such as foot gestures, gait, etc. However, hand gestures are still the most intuitive and, with the flexibility of arms and wrists, plus the eventual use of fingers, many more gestures can be performed with respect to other parts of the body. After all these considerations, it was then decided to focus on hands for the gesture performance. Common movements and intuitive hand motions were the target for this task. It does not make much sense, in fact, to ask a user to perform very complicated and long gestures; instead, it is preferable to opt for natural, easy and rapid movements. Nevertheless, the goal was to train a model able to recognize as many gestures as possible, hence it was decided to fix the final number of gestures composing the dataset at 20. This gives the user many options and allows to associate to the movements many actions, but at the same time those are intuitive and hence easy to remember.

The final set of movements is represented in Figure 3.1. The lighter gray draw represents the start of the movement, while the black is the final part of it.

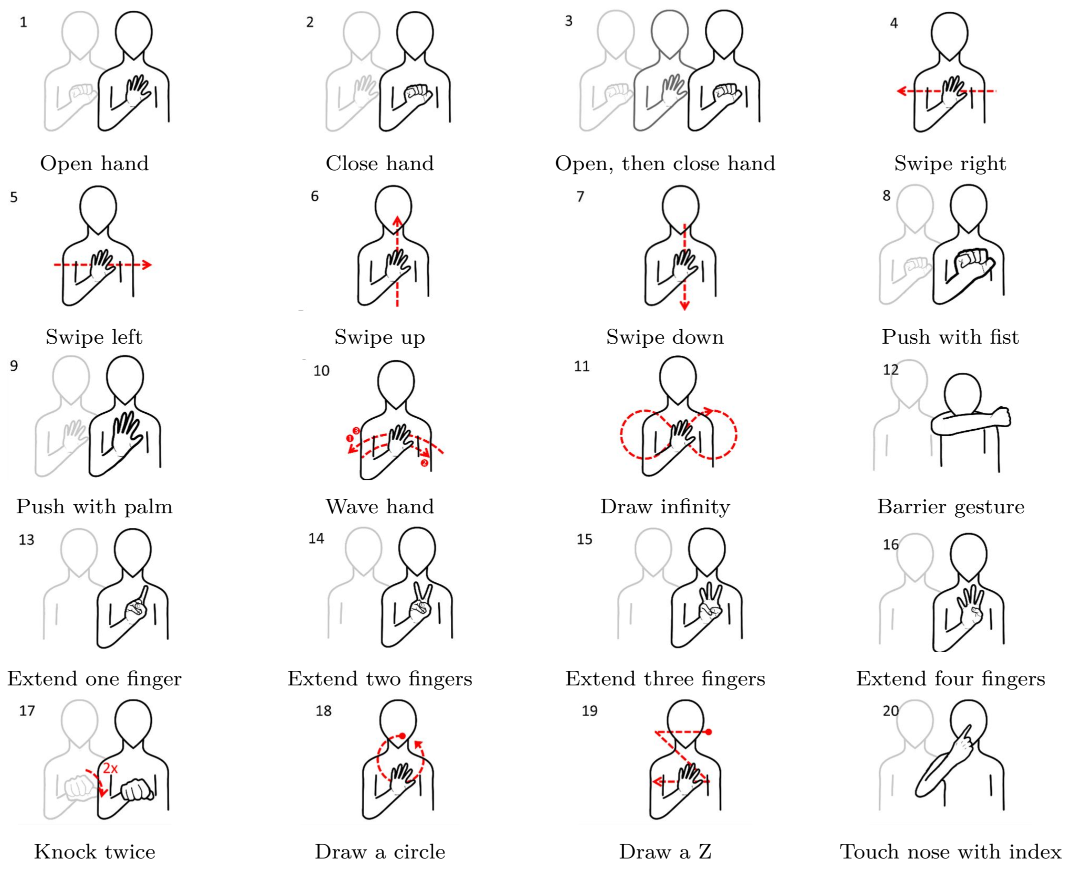


Figure 3.1: Gestures set

Following the [93] classification criteria, it is possible to divide our gestures into different categories based on how they are performed and interpreted by the users:

- **Pointing**. Pointing is used to indicate objects and directions, which does not necessarily involve a stretched index finger. It may also be performed with multiple fingers, the thumb, a flat palm, etc.

- **Semaphoric**. Semaphoric gestures are postures and movements of the hands that are used to convey specific meanings. Most of the time, gestures and meaning are completely unrelated and strictly learned. Therefore, we consider that semaphorics are the gestures that are more dependent on the background and experience of the actor. *Static* semaphorics are identified by a specific hand posture. Examples would be a thumbs up, which means ’okay’, or a flat palm facing the actor, which means ’stop’. *Dynamic* semaphorics convey information through their temporal aspects. A circular hand motion that means ’rotate’ is of this type, as well as a repeatedly flicking or waving of the hand sideward, which means ’no’.

- **Pantomimic**. Pantomimic gestures are used to demonstrate a specific task to be performed or imitated, which mainly involves motion and particular hand postures. They are usually performed by an actor without any objects actually being present, such as filling an imaginary glass with water by tilting an imaginary bucket. They often consist of multiple low-level gestures, e.g., grabbing an object, moving it, and releasing it again.

Pointing	8, 9 ,13, 14, 15, 16
Semaphoric	1, 2, 3, 4 , 5, 6, 7, 10, 11 , 18, 19
Pantomimic	12, 17, 20

Table 3.1: following [93] classification criteria

As said before, the driving criterion in the choice of the movements was to have something easy to perform and also to remember. Ideally, in an application that exploits a hand gesture recognition system, we would like to associate to each gesture a certain action. The motions listed here could be easily coupled with actions e.g., in a domotics context, in which each movement is associated to the opening or closing of parts of the house such as windows, doors, or the power on/off of some domestic appliances, etc. or, again, in some automotive applications in the infotainment system to control some features without the necessity of touching the screen, etc.

3.2 Recording setup

Regarding the arrangement for the recording of the gestures, the goal was to have the "cleanest" environment possible in terms of possible disturbances for the radar waves. This was necessary in order to obtain a signal as free as possible from objects and obstacles in the neighborhood, so that only the movement of the hand would be detected and recorded. The gestures were captured in two different rooms for organizational reasons. A small part of the dataset (8 users) was recorded in the first one, while the rest (14 users) was finished in the second and final room. It is true that, when acquiring this kind of data, it is not a good behaviour to change the environment in the middle of the experiment, because this may affect and alter the results of the recorded signal. However, in the second setup it was possible to closely reproduce the conditions of the first setup in terms of spaces, distances, and surrounding objects. Moreover, a signal filtering to process the raw data allowed to remove most of the noise and reflection effects; therefore, this change of environment did not significantly influence the final result. Figure 3.2 shows the first acquisition setup for the samples of the dataset. The user is placed in front of the Walabot radar at a distance of about 70cm, which is in turn elevated a bit with respect to the table, so that there are fewer reflections coming from the surface. In Figure 3.3, instead, is displayed the second setup, in which we tried to replicate the first set-up as faithfully as possible the first one.

Regarding the technical aspects of the setup, here are the most significant elements:

- **Laptop**. Asus Zenbook 13 UX331:
  - Intel ©Core i7-8550U Processor 1.8 GHz (8M cache, up to 4.0 GHz, 4 cores);
  - Intel ©UHD Graphics 620;
  - 13.3-inch, FHD (1920 x 1080);

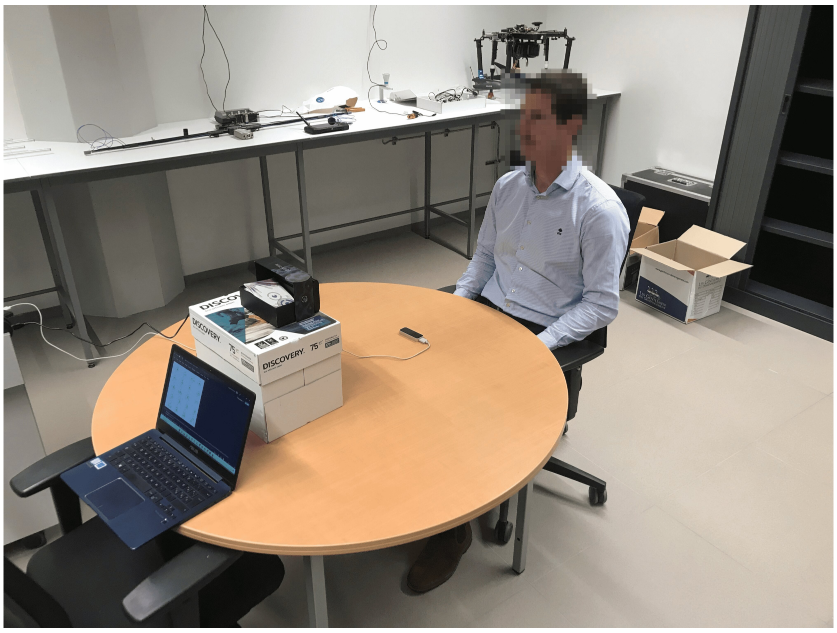


Figure 3.2: First recording setup of the Walabot dataset

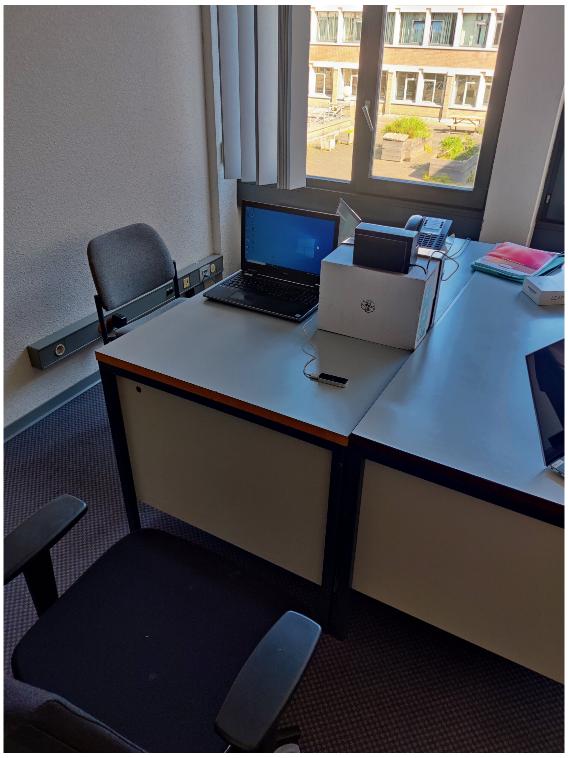


Figure 3.3: Second recording setup of the Walabot dataset

  - 8GB LPDDR3 RAM;
  - OS: Windows 11 Home version.

- **Walabot**. Walabot Developer Kit EU version:
  - Frequency range: $6.3GHz - 8GHz$;
  - Board size: $72mm \times 140mm$;
  - VYYR2401 A3 System-on-Chip for signal production and recording;
  - Cypress FX3 controller for USB communication and data pre-processing;
  - Micro-USB 2.0 for high-rate data communication;
  - Single supply voltage 4.5-5.5v input for non-USB power applications.

- **Software**. Console application for recording gestures in $C{+}{+}$ semi-automatically. The user is given the start-stop signal from the recorder, which is activating/deactivating the recording at each sample.

3.3 Recording and dataset structure

3.3.1 Recording procedure

Let us describe the recording phase and how the gestures were acquired. We tried to keep the procedure as consistent as possible through the whole set of participants, even if it was considered acceptable the case in which some of the samples were acquired in with slight differences, in order to allow for some variance in the dataset and, hopefully, make the model more robust.

In practice, this means that some degrees of freedom were given to the user when performing the gesture. Some centimeters of difference in the distance from the radar were tolerated, different ways of performing the gestures were accepted, as well as different speed in doing the movement etc. All these small differences were permitted, with the idea that it could have helped the model to improve in terms of robustness and flexibility over the different users.
Moreover, it was also very interesting to see how different people were interpreting the gestures in performing them. Some guidelines were given at the beginning of the experiment, for example the request of keeping the hand as parallel as possible to the Walabot device that was in front of the user, in order to get as much reflection as possible, or following the recording procedure that will be detailed later, in terms of start and end of the recording. Apart from these small details, complete freedom was left to the user in the act of the hand movement, so that it could be as natural as possible.
The biggest differences that could be observed in the different users were in term of:

- speed: some participants were performing the gesture very slowly, others were very fast in completing the movement;
- broadness: some people very moving the hand very widely, while others tended to keep the movement very short and close to the radar.

Having these differences can be denoted as a positive thing, because it improved the variance in the dataset. On the other side, it is also important to give some regulations and guidelines about the performance of the gesture, in order to not increase it too much and obtain too sparse data that are not useful anymore.
Regarding the acquisition of one sample, the procedure that we pursued was the following:

1. The participant starts with their hands on their laps (Figure 3.4a);

2. The recording is started;
3. The participant performs the gesture (Figure 3.4b, 3.4c);
4. The participant puts their hands back on their laps (Figure 3.4d);
5. The recording is stopped.

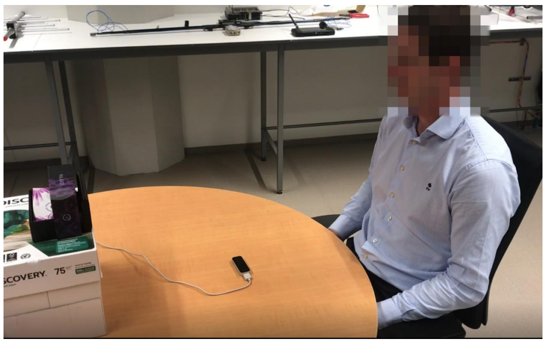
This procedure was repeated multiple times for each participant until the target number of samples for every gesture was reached. The total amount of time needed to complete the recording for one person was about 25/30 minutes.

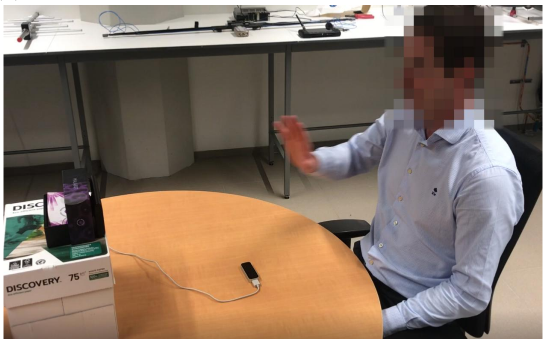
3.3.2 Dataset structure

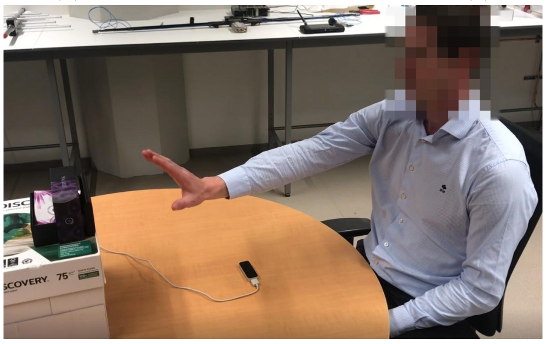
In this subsection is outlined the structure of the gesture collection done during the whole duration of the experiment. The total number of participants that was reached was of twenty-two. The initial dataset at the beginning of the experiment was of 5 users, but over time it was possible to increase such number to a total of 22 people. Of course, the ideal would be to have as many participants as possible to enlarge the dataset and train a more robust model, however the resources at disposal for this time allowed to reach this number of participants. Anyway, as a future improvement, the dataset shall be expanded to train bigger and more complex models, so that higher confidence thresholds in the recognition accuracy can be reached.

For each gesture it was asked to the user to perform the movement ten times, which for twenty gestures made a total of 200 samples.

The profile used for the recording was the "*PROF_SENSOR_NARROW*" which, as mentioned in Section 2.2, employs a total of twelve antenna pairs for the signal acquisition. By means of the signal processing phase explained in Chapter 4, it was possible to extract a measurement from each antenna pair, hence this brought to twelve significant datapoints for each gesture sample. To recap, the final number


(a) The participant starts with their hands on their laps


(b) The participant performs the gesture (1)


(c) The participant performs the gesture (2)

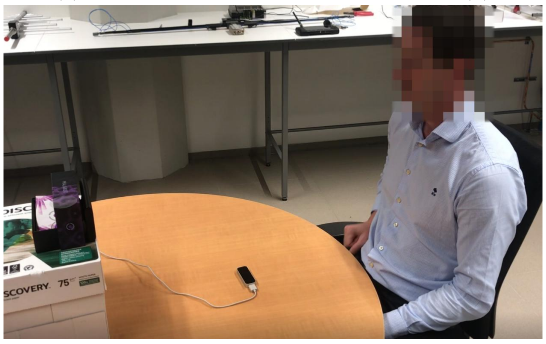

(d) The participant puts their hands back on their laps

Figure 3.4: Recording of one sample of Gesture 9 (Push with palm)

of elements of the final version of the Walabot dataset is give by:

$$\#\text{data} = 22\ \text{users} \times 20\ \text{gestures} \times 10\ \text{samples} \times 12\ \text{antennas} = 52'800$$

This is the final number of data samples that, after the filtering and pre-processing phase, can be used for the training and validation of the model. The details about the splitting for the different tasks are given in the appropriate chapter (Chap. 5).

4 DATA PROCESSING

Subsequently to the dataset acquisition by means of the recording process explained in Chapter 3, the signal processing part served to filter and prepare the data for the training of the model. The procedure to do so is explained in this chapter.
In order to determine the processing operation, it was necessary to understand the type and the structure of the data at disposal, and outline the type of data that was needed for the deep learning model.
As mentioned in Section 2.1.2, and as it can be observed by Table 2.2, there have been some precedent works that exploited images of Doppler responses as inputs to a convolutional neural network [67, 70]. The main drawback of $2D$ image based methods is that they lack of the third dimension that adds further information to the signature of each gesture. On the other hand, using $3D$ images, such as in [69, 73], and hence processing them using $3D$ convolutional layers that take into account also temporal correlations, allows to obtain richer information, but the model complexity and the need of high computational power for the training increases significantly [94].

From this considerations, the strategy chosen in the end was that of using a set of 2D images representing the radar signal extracted from the signal processing part as input to a Convolutional Neural Network for the feature extraction and classification. It was then clear that the objective was to transform the raw signal coming directly from the antennas into a significant image that could be given as input to the CNN.

4.1 Radar processing pipeline

As anticipated in Section 2.1.2, part of this project relied on the research carried out by Vanderdonckt, Slüythers and Lambot [75], which used the same Walabot device for collecting data, and aimed to obtain input images in a similar way to what was needed for this activity. In their research, the authors developed a radar data processing pipeline for hand gesture recognition in order to extract meaningful physical features such as hand-radar distance and the hand permittivity. This differs from our use case, since the idea is to give to the network an image from which it can self-learn relevant features associated to a particular gesture. Therefore, the idea was to use only part of the pipeline, since it is designed to be flexible to use depending on constraints imposed by the target usage context.

The main challenges brought by a radar signal based hand gesture recognition problem are essentially:

- Sensitivity to noise created by occlusions and surrounding objects' reflections;
- Complexity and size of radar data vs. other more interpretable data obtained through other imaging systems such as traditional cameras or infrared cameras etc.

The pipeline designed in [75] aims to counter-effect these problems in order to obtain significant data points for the learning of the model. It mainly relies on the

Fast Fourier Transform (FFT) and its inverse (IFFT) to bring the signal into the frequency domain and elaborate it to remove the noise and the bad reflection effects that could alter the final result. The radar data processing pipeline is composed of different stages that can be visualized in Figure 4.1.

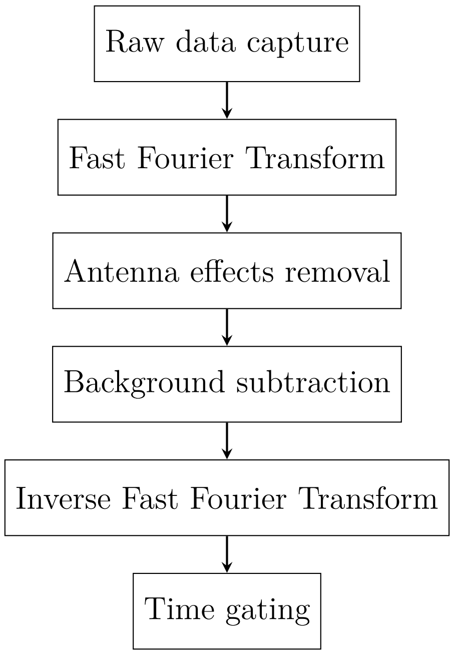


Figure 4.1: Radar data processing pipeline

4.2 Pipeline stages

The main steps that compose the signal processing and filtering shown in Figure 4.1 are outlined in the following:

- **Raw data capture.** Raw data is acquired from each radar antenna. For the case of the Walabot, data are provided in the time domain. For each registered sample, an output file is created. In such file, data are organized so that each line represents one measurement. The first column represents the elapsed time between the first and the current measurement, and the second column represents the amplitude of the measured signal. One frame consists of:

$$12 \text{ antenna pairs} \times 1024 \; \frac{\text{measurements}}{\text{antenna pairs}} = 12'288 \text{ lines} \tag{4.1}$$

As such, dividing the number of lines by 1024 gives the number of frames per gesture. The file is hence organized as follows:

Frame 1	Antenna pair 1 (1024 measurements)
	...
	Antenna pair 12 (1024 measurements)
...	...
Frame n	Antenna pair 1 (1024 measurements)
	...
	Antenna pair 12 (1024 measurements)

Table 4.1: Structure of a raw data output from the Walabot

Therefore, the first 1024 lines represent the signal measured by the first antenna in the first frame, the next 1024 lines represent the signal measured by the second antenna in the first frame, etc. In Table 4.2 is reported an example of such file containing the measurements (Gesture 1, sample 1, user 5). It is clear that the number of frames n is variable and depends on the length of the gesture. The measurement provided by the Walabot for each time instant is a double precision real number representing the voltage evaluated by the internal circuit normalized in the interval [-1, 1].

- **Fast Fourier Transform.** The radar signal is transformed from the time domain to the frequency domain through the FFT algorithm mentioned before. This operation is required for the next stage, which must be performed in the frequency domain.

Line number	**Frame**	**Antenna pair**	**Elapsed time (s)**	**Amplitude (V)**
1	Frame 1	Antenna pair 1	0.00000	0.015496
2			1.95313e-11	0.022563
3			3.90625e-11	0.017846
...			...	...
1024			1.99805e-08	0.002829
1025		Antenna pair 2	0.00000	0.084012
...			...	...
2048			1.99805e-08	0.000923
...		...	...	...
...		Antenna pair 12	...	...
12288			1.99805e-08	0.000108
12289	Frame 2	Antenna pair 1	...	...
...	...	...	...	...
...	Frame n	...	...	...

Table 4.2: Example of raw signal file structure

FFT algorithm

The FFT is an algorithm that computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IDFT). Fourier analysis converts a signal from its original domain, which in this case is time, to a representation in the frequency domain and vice versa. The DFT is obtained by decomposing a sequence of values into components of different frequencies [95]. Computing the transform directly from the definition is often too slow in the practical case. Instead, an FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors, reducing significantly the computational cost and time necessary.

- **Removal of radar source and antenna effects.** Through the modelization of the antenna and the equation given in [96, 97] the radar source and antenna effects (e.g., internal reflections and transmissions) and antenna-target interactions; this is it done by applying such equation the raw frequency domain signal.

Radar equation

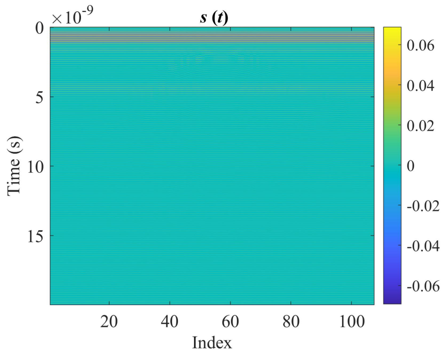
The Walabot radar data were processed using the radar equation of Lambot et al. [96, 97] assuming far-field conditions. First, the time-domain data were transformed into the frequency domain in which the radar equation applies. Then, after determination of the radar-antenna characteristic functions through a calibration procedure for each antenna pair, the Greens functions were calculated from the measured data, thereby filtering out radar-antenna effects including the source, antenna internal reflections and transmissions and antenna-target interactions. The Greens function represents the backscattered electric field given a unit electric source and, hence, is a quantity that is normalized and independent of the radar system.

- **Removal of the background scene.** Using the superposition principle, the first frame of a gesture is subtracted from the radar signal to remove the remaining reflections from static reflectors, such as walls, furniture, or other objects. This stage is required to ensure accurate feature extraction in the later stages, as reflections from other (static) objects could be confused with the users hand.

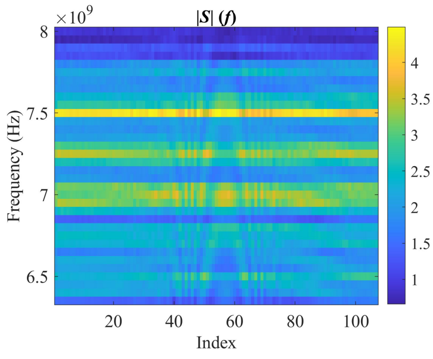
- **Inverse Fast Fourier Transform.** The filtered radar signal is transformed from the frequency to the time domain.

- **Time gating.** The time-domain data is truncated to only keep the portion of the signal that is relevant for gesture recognition. Here, only the signal received within a given time window is kept. This removes useless information, such as objects that are too far away from the radar, thus improving accuracy and reducing the processing time of the next stages.

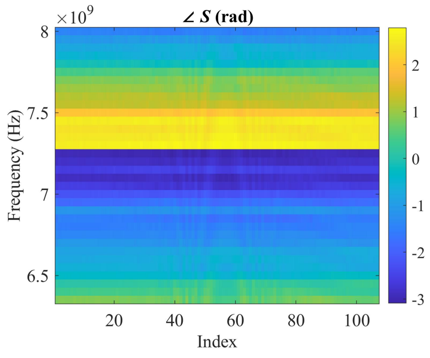
In Figure 4.2 are shown the images that represent the signal in the different stages that compose the pipeline, in order to demonstrate the effects of each part on the signal, starting from the raw data captured by the radar to the final image that will be used as input for the CNN.



(a) Raw signal in time domain

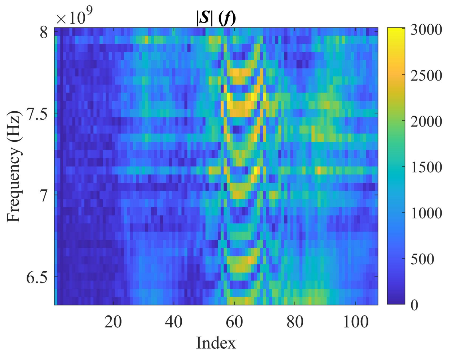


(b) Raw signal in frequency domain (Amplitude)

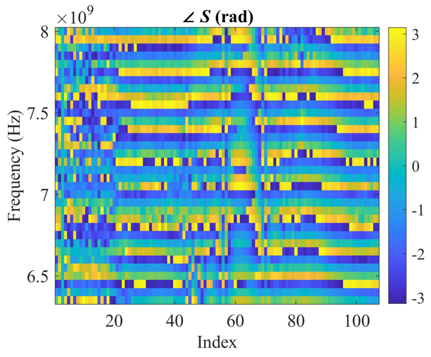


(c) Raw signal in frequency domain (Phase)

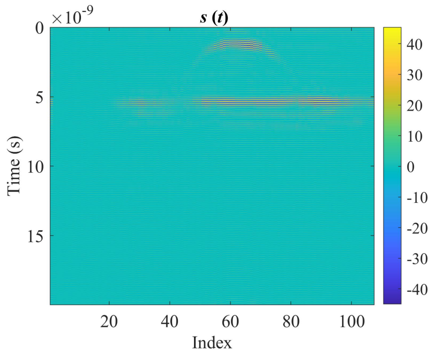
Figure 4.2: Visualization of the signal transformation from raw to processed form



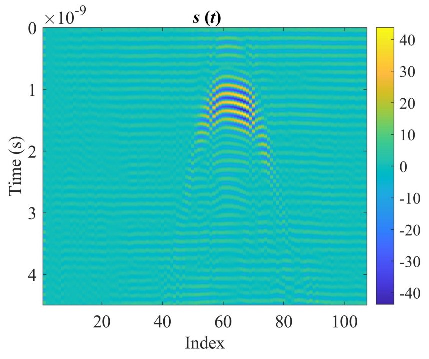
(a) Filtering in frequency domain - Background subtraction and antenna internal effect removal (Amplitude)



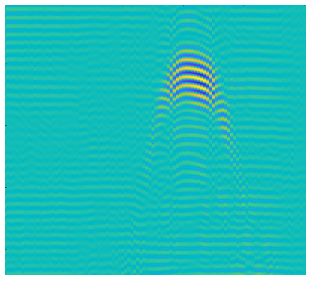
(b) Filtering in frequency domain - Background subtraction and antenna internal effect removal (Phase)



(c) Filtered signal transformed back in time domain



(d) Time gating of the time domain signal



(e) Final image used as input for the CNN

Visualization of the signal transformation from raw to processed form

As it can be observed from Figure 4.2, the pipeline involves many steps, yet all very important to obtain a reasonable data point for the model. If Subfigure 4.2a and Subfigure 4.2d are confronted, it is clear how the filtering in the frequency domain impacted the signal to achieve a significant image from which the model is going to extract relevant features for the classification task.
Nevertheless, it must be said that the whole signal process is quite computationally demanding. No scientific trials and experiments about the resource requirements were carried out, but the processing of the dataset took a long time to be done ($\sim$one week for the whole dataset). This suggests that, in order to transform the whole recognition system into an online system, it would require to optimize more the processing part and a decent hardware capable of performing the filtering operations in almost real time.

5
DEEP LEARNING MODEL

This chapter is related to the actual deep learning architecture that was developed to handle the classification task proposed in this thesis. In this section, a brief theoretical overview will be given of the main features and characteristics of classification models, together with an analysis of the most used deep learning architectures for such purposes. Subsequently will be outlined the development of the final version of the model that was used for the recognition task, together with its structure and parameters (layers, number of neurons per layer, etc.).

5.1 Overview on deep learning tasks and models

5.1.1 Supervised learning and classification task

Deep learning is a subfield of machine learning. The tasks that can be tackled with the different techniques available at the state-of-the-art depend on a broad number of variables such as the quantity of available data, the computational power at disposal, etc. Most machine learning algorithms can be branched into supervised and unsupervised learning tasks. Roughly speaking, these two categories can be described as follows.

- **Unsupervised learning algorithms** observe a dataset with several features and then detect important properties of this dataset's structure. For tasks like synthesis or denoising in the context of deep learning, we typically wish to learn the complete probability distribution that produced the dataset, whether directly as in density estimation or implicitly. Other unsupervised learning algorithms play different roles, such as clustering, which divides the dataset into groups of related cases.

- **Supervised learning algorithms** work with datasets containing features, but each example additionally has a label or target attached to it. For instance, a dataset on different animals will contain a certain number of images of such animals, and each image comes with its associated label.

In the classification task, the computer program is asked to specify which of k categories, denoted also as labels, some input x belongs to. The learning algorithm is asked to produce a function $f : \mathbb{R}^n \longrightarrow \{1, \ldots, k\}$. When $y = f(x)$, the model assigns an input described by the vector $\boldsymbol{x}$ to a category identified by the numeric code y.
Classification problems are usually tackled in a supervised setting. As expected, in this scenario, the dataset contains data points associated with an appropriate label. This allows the learning algorithm to learn the correct features for each class by having at its disposal the ground truth and eventually, after the training phase, to be able to classify unlabeled samples with the correct class of belonging.

5.1.2 Supervised deep learning

In the field of deep learning, there are many state-of-the-art models to deal with classification tasks, depending on the nature of the data. The main differences between these architectures lie in the type and number of layers used, and the choice of using one over another is mainly dictated by the type of data contained in the dataset. Let us briefly report the main architectures that are typically used for this task.

Fully connected neural networks

Fully connected neural networks (FCNNs) are a type of artificial neural network where the architecture is such that all nodes, or neurons, in one layer are connected to neurons in the next layer. The number of layers, together with the number of hidden units per layer, that compose the network are hyperparameters to be chosen and tuned in the development of the model. The major advantage of FCNNs is that they are 'structure agnostic', that is, no special assumptions need to be made about the input.

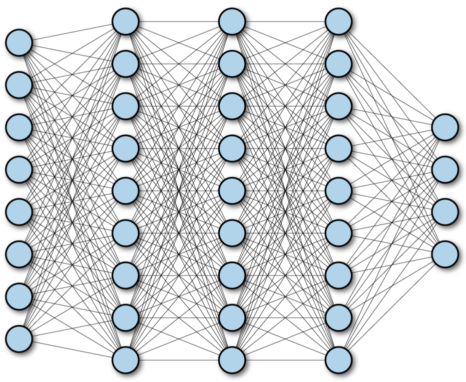


Figure 5.1: Example of Fully Connected Neural Network [1]

[1]Source: https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html

While being structure agnostic makes FCNNs very broadly applicable, such networks do tend to have weaker performance than special-purpose networks tuned to the structure of a problem space. The major limits of FCNNs are shown when dealing with image recognition and classification. Their fully connected structure increases a lot the number of learnable parameters, which may highly impact the computational expense and may be prone to overfitting. Nevertheless, they may show good performances in regression problems when the dataset is only limited to low dimensional datapoints.

Convolutional neural networks

Convolutional Neural Networks (CNNs/ConvNets) are another class of architectures usually employed for classification tasks. A CNN is usually well suited with images or in general for processing data that has a grid-like topology. There are typically three main types of layers, which are:

- *Convolutional layer*: This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field. The kernel is spatially smaller than an image can be deeper (not only e.g., 3 channels for RGB images). During the forward pass, the kernel slides across the height and width of the image-producing the image representation of that receptive region. This produces a two-dimensional representation of the image known as an activation map that gives the response of the kernel at each spatial position of the image. With convolutions, the size of the output is usually smaller than the input (e.g. 5x5 input convoluted with 3x3 kernel = 4x4 output). If $W \times H$ is the dimension of the input image, and K the kernel window size, then the general formula is:

$$(W \times H) * (K \times K) = (W - K + 1) \times (H - K + 1) \tag{5.1}$$

Note that here the kernel is considered to be squared, although it is possible to use also rectangular kernels with size $K_1 \times K_2$ If it is wanted to maintain the same size, it is possible to use some *padding*. Padding is a process of adding zeros to the input matrix symmetrically. With such a parameter, then the formula becomes:

$$(W \times H) * (K \times K) = (W + 2P - K + 1) \times (H + 2P - K + 1) \tag{5.2}$$

Another parameter is the sliding size of the kernel, defined as *stride*. The stride denotes how many steps we are moving in each step in convolution, and by default it is one. If both stride S and padding P are considered in the convolution, then the size of output volume can be determined by the following formula:

$$\text{Output size} = \left(\left\lfloor \frac{W - K + 2P}{S} \right\rfloor + 1 \right) \times \left(\left\lfloor \frac{H - K + 2P}{S} \right\rfloor + 1 \right) \tag{5.3}$$

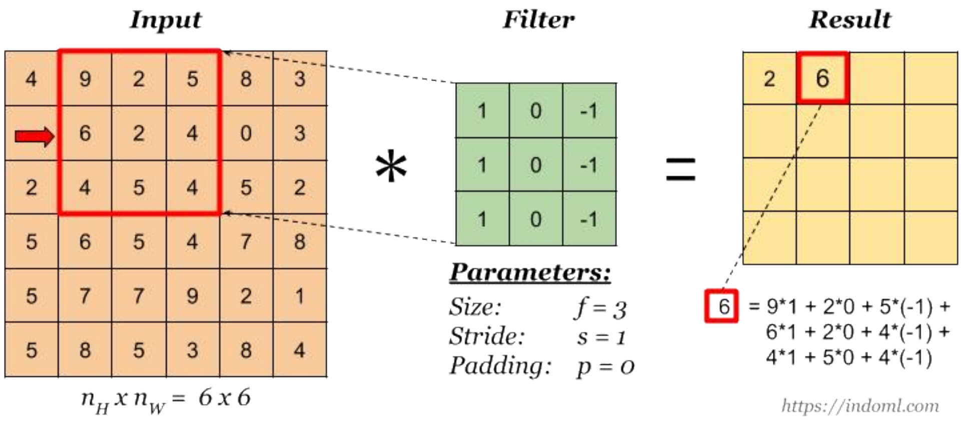


Figure 5.2: Convolution operation example with a 3×3 kernel [2]

- *Pooling layer*: the pooling layer is responsible for reducing the spatial size of

[2]Source: https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html

the convolved feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training the model. Usually *Max Pooling* is used, so the maximum value of the portion of the image covered by the kernel is returned. In the Pooling layer, usually only the kernel size and stride hyperparameters can vary. The computation for the output size is given by:

$$\text{Output size} = \left(\left\lfloor \frac{W-K}{S} \right\rfloor + 1\right) \times \left(\left\lfloor \frac{H-K}{S} \right\rfloor + 1\right) \tag{5.4}$$

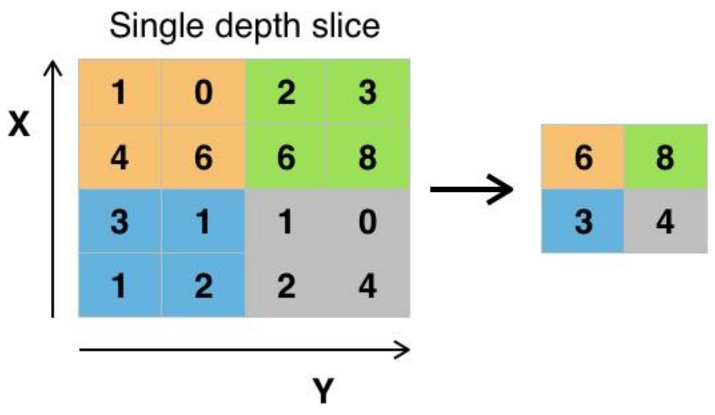


Figure 5.3: Max pooling with a 2×2 filter and stride = 2 [3]

- *Fully-connected layer*: This layer performs the task of classification based on the features extracted through the previous layers and their different filters. While convolutional and pooling layers tend to use ReLU functions, FC layers usually leverage a Softmax activation function to classify inputs appropriately, producing a probability from 0 to 1.

[3]Source: https://en.wikipedia.org/wiki/Convolutional_neural_network

In Figure 5.4 it is displayed an example of the structure of a CNN, where all the relevant layers are reported. Much more complex architectures nowadays are employed, with tens or hundreds of layers such as in [98].

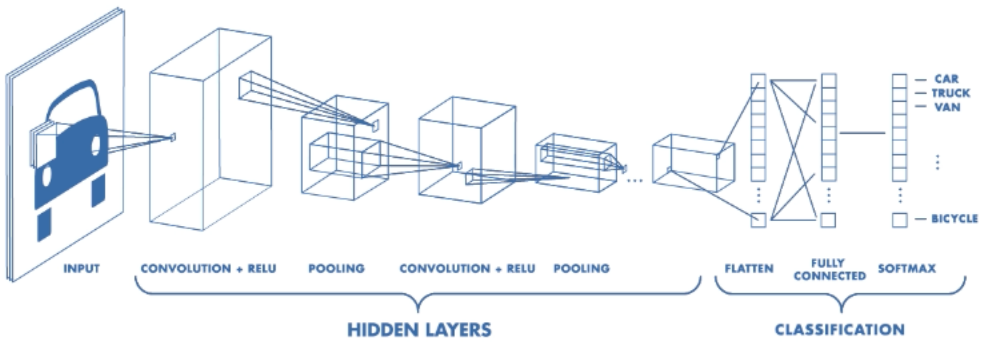


Figure 5.4: Convolutional Neural Network structure [4]

Other architectures

The targeted literature review performed in Ch. 2 highlighted the most frequent deep learning algorithms for radar-based hand gesture recognition. Among those, we find FCNNs and CNNs, explained previously hereby. On top of that, there are most certainly other deep learning architectures that have been tried for such task. For example, in [99] an hybrid solution composed by a CNN followed by a LSTM was used. Some other approaches were tempted with transfer learning models, such as in [100] where a team of researchers from Sweden used *ResNet-50* to classify hand gestures, or in [101], in which a classifier for hand gesture recognition and classification was presented, based on *GoogLeNet* architectures and *Inception* modules. Another approach was tempted in [102] using more classical machine learning algorithms such as *kNN*.

[4]Source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Despite the abundance of other approaches, it was still decided to go with a custom CNN from the principle. This finds its reasons in the fact that the dataset used for this project was characteristic and intended for this specific task. Therefore, all resources and efforts were made to develop a model tailored to the available data.

5.2 Overview on the training and validation of a deep learning model

The training phase of a machine learning model usually follows a precise sequence of actions regardless of the architecture of the model itself. What differentiate the most the process is the choice of hyperparameters and various techniques used in the training, which can highly affect the performances of the final model.
In the following, an overview of the main ingredients that compose a training of a deep neural network is given.

5.2.1 Train, validation and test sets

At the heart of a training process is the dataset, which has to be properly handled in order for the model to be able to extract useful information from it. Usually, the entire dataset is split into subsets for different purposes, namely:

- *Training set*: The sample of data used to fit the model. It is the actual dataset that we use to train the model (weights and biases in the case of a Neural Network). The model sees and learns from this data.

- *Validation set*: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The validation set is used to evaluate a given model, hence the model occasionally *sees* this data, but never does it *learn* from this.

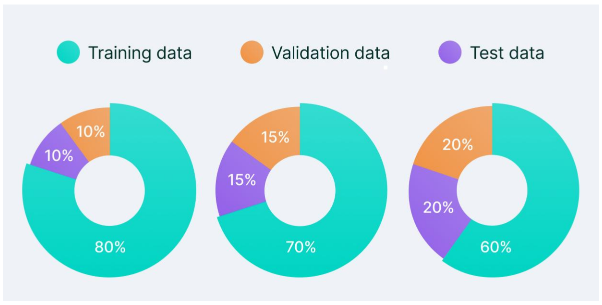


Figure 5.5: Some typical *train/validation/test* splits of the dataset [5]

- *Test set*: The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset. It is only used once a model is completely trained, using the above sets. It has to be carefully curated, meaning that it contains a good amount of samples from each class so that the model is put under test in all circumstances that it could encounter in a real world scenario. It is a good way of estimating the real performances of the final version of the model.

An important matter regards the splitting percentages of the total dataset. Some studies show how the changing the size of the train, validation and test subsets impacts the performances of the model [103, 104, 105], however there is not an absolute rule that is appropriate for each dataset. Over the years it has been defined as a rule of thumb a combination of splitting such as 80%/10%/10%, or 70%/15%/15% etc., as it is shown in Figure 5.5.

Some deep learning architectures have actually been trained with very different splittings, giving higher enphasis on the training part, e.g., 90%/3%/7% such as in [98]. However, in these cases, this is feasible for the quantity of data and computational power at disposal which is enormously high ($\sim$ 1.2 millions for [98]), therefore even the 2% or 3% of data means thousand and thousand of data. In conclusion, this situation was not applicable to this case. For this reason, it was preferred to stick to more usual data splittings. Anyhow, the thing to keep in mind while

[5]Source: https://www.v7labs.com/blog/train-validation-test-set

choosing the splitting is that: with less training data, the parameter estimates have greater variance; with less testing data, the performance statistic will have greater variance. The splitting should be chosen in order to balance this trade-off such that neither variance is too high.

5.2.2 Training algorithm

The training of a deep network consists in learning the parameters of each layer composing the network such that a given loss function is minimized. For this goal, it is very common for feed-forward neural networks to use the *Stochastic Gradient Descent* (SGD) algorithm. It is an iterative method for optimizing an objective function with suitable smoothness properties. This version of the algorithm is a stochastic approximation of the more exact *Gradient Descent* (GD) method, for it replaces the actual gradient (calculated from the entire data set) by an estimate thereof (calculated from a randomly selected subset of the data). This allows to reduce the computational complexity of the GD method and still achieve a pretty high accuracy in reaching the minimization of the loss function. In the following an example of pseudo code of this algorithm is presented.

Algorithm 5.2.1 The general gradient descent algorithm; different choices of the learning rate γ and the estimation technique for $\nabla\mathcal{L}(w)$ may lead to different implementations.

```
Input: initial weights $w^{(0)}$, number of iterations $T$
Output: final weights $w^{(T)}$
1.  for $t = 0$ to $T - 1$
2.      estimate $\nabla\mathcal{L}(w^{(t)})$
3.      compute $\Delta w^{(t)} = -\nabla\mathcal{L}(w^{(t)})$
4.      select learning rate $\gamma$
5.      $w^{(t+1)} := w^{(t)} + \gamma\Delta w^{(t)}$
6.  return $w^{(T)}$
```

The SGD algorithm by itself is often not enough in the case of deep networks, due to their non-linear nature. In fact, the main problem lies in the computation

of the gradient to perform the update of the weights.
For this reason, a second algorithm is used together with the SGD, that is the *back-propagation* algorithm. It is used for calculating the gradient of a loss function with respect to the variables of a model. The back-propagation algorithm, often simply called *backprop*, allows the information from the cost function to then flow backwards through the network, in order to compute the gradient. The algorithm involves the recursive application of the chain rule from calculus (different from the chain rule from probability) that is used to calculate the derivative of a sub-function given the derivative of the parent function for which the derivative is known.

5.2.3 Loss functions

A loss function is a function that compares the target and predicted output values; it measures how well the neural network models the training data. When training, the aim of training is to find the most suitable set of parameters that minimizes this loss between the predicted and target outputs. More formally, we have that the output of a neural network can be defined as:

$$\hat{y} = \sigma(w^T X + b) \tag{5.5}$$

where $\hat{y}$ is the predicted output, σ is the activation function, w and b are the learnable weights and biases and X is the input. Given that, the goal is to find the best set of (w^T, b) such that the average loss J is minimized:

$$J(w^T, b) = \frac{1}{m} \sum_{i=1}^{m} \mathfrak{L}(\hat{y}^{(i)}, y^{(i)}) \tag{5.6}$$

where m is the total number of samples. Many loss functions exist, and depending on the task some are more suitable than others. Let us briefly recap the most common ones.

Mean Squared Error (MSE)

The MSE loss function finds the average of the squared differences between the target and the predicted outputs.

$$\mathfrak{L}(\hat{y}, y) = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2 \tag{5.7}$$

It is well suited for calculating loss since the difference is squared, meaning that it does not matter whether the predicted value is above or below the target value; moreover, values with a large error are highly penalized. It is a convex and differentiable function with a clearly defined global minimum. It is usually used in **regression** problems.

Binary cross-entropy/Log loss

This loss function is used in binary classification problems, in which the model takes in an input and has to classify it into one of two categories (e.g., 0 or 1). The formulation takes into account the probability that the given input fits each category.

$$\mathfrak{L}(\hat{y}, y) = -\frac{1}{m}\sum_{i=1}^{m} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log(1 - \hat{y}_i) \tag{5.8}$$

Classification with neural networks is solved by comparing the actual value (0 or 1) with the probability that the input aligns with that category, e.g., $\hat{y}_i = p_i =$ probability that the category is 1, and $(1 - \hat{y}_i) = (1 - p_i) =$ probability that the category is 0.

Categorical cross-entropy

This loss function is the more general case for multiclass classification models, and it follows a process similar to binary cross-entropy, where the loss is computed between each pair of classes, and then everything is summed up all together:

$$\mathfrak{L}(\hat{y}, y) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{C} y_{ij} \cdot \log(\hat{y}_{ij}) \tag{5.9}$$

where C is the number of classes of the problem. Binary cross-entropy loss is a special case of this, in which $C = 2$.

5.2.4 Activation functions

In the context of artificial neural networks, the *activation function* of a node (neuron) defines the output of that node, given an input or a set of inputs. It is basically the element that decides whether a neuron should be activated or not. Many activation functions exist and are employed in certain parts of the neural network depending on the role of such activation. The main activation functions commonly employed in modern deep architectures are reported in the following.

Linear/Identity

The *linear* activation function, also known as the "identity" function, is used to obtain an output that is proportional to the input. The function does not do anything to the weighted sum of the input but rather simply returns the value it was given.

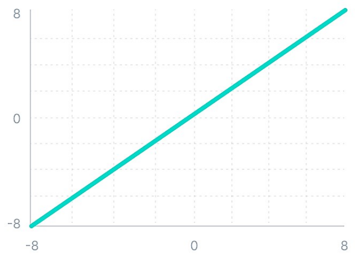


Figure 5.6: Linear Activation Function

It is usually employed in the input layer or in the output layer of a regression problem, as in such scenario, we are interested in numerical values without any transformation. By being a linear function, it is not possible to use backpropagation, as the derivative of is a constant and has no relation to the input. Moreover, the use of an identity function in the hidden layers would correspond to collapsing all the layers into one, thus losing the meaning of a deep network.

Sigmoid

The *sigmoid*, or logistic activation function, takes as input any real value and outputs values in the range $[0, 1]$. Mathematically, it corresponds to:

$$f(x) = \frac{1}{1 + e^{z-x}} \tag{5.10}$$

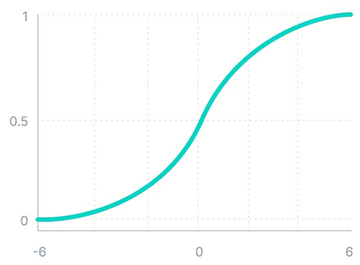


Figure 5.7: Sigmoid Activation Function

It is commonly used for models in which the output prediction should be a probability (between 0 and 1). The function is differentiable and provides a smooth gradient. The major limitation is that it suffers from the *vanishing gradient* problem as the gradient becomes very small.

Tanh

Tanh function is very similar to the sigmoid, with the difference that the output is now between -1 and 1. Mathematically we have the following:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5.11}$$

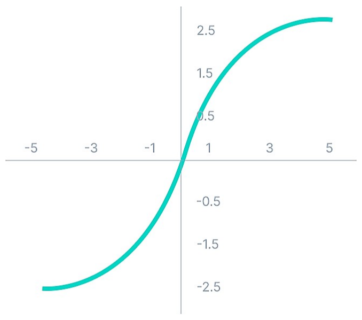


Figure 5.8: Tanh Activation Function

Tanh function is zero centered; this allows us to map the output values as strongly negative. strongly positive or neutral. The zero centering also allows us to have outputs around zero, which makes learning for the next layer much easier. The vanishing problem is also present here, together with a steeper gradient with respect to the sigmoid.

ReLU and Leaky ReLU

The Rectified Linear Unit (ReLU) activation function is widely used in modern deep learning architectures for its computational efficiency and its help in the convergence of the gradient. Although it seems to be a linear function, it is actually not; therefore, it is possible to use backpropagation. The idea is to have a function that does not activate all neurons at the same time, making it more efficient than others. Neurons will be activated only if the output of the linear transformation is greater than 0:

$$f(x) = \max(0, x) \tag{5.12}$$

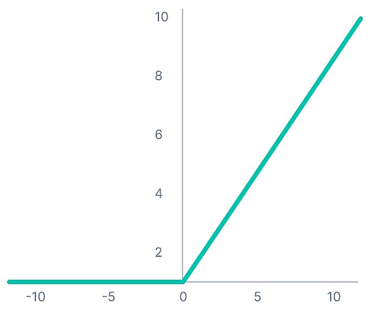


Figure 5.9: ReLU activation function

The main problem in this context is the so-called *Dying ReLU*, which consists of having neurons that during the backpropagation process are not updated for the left side of the graph that has zero gradient. Therefore, all negative input values become zero, which decreases the model's ability to fit or train from data correctly. To prevent this phenomenon, the *Leaky RelU* function can be exploited. It is an improved version that has a small positive slope in the negative area.

$$f(x) = \max(0.1 \cdot x, x) \tag{5.13}$$

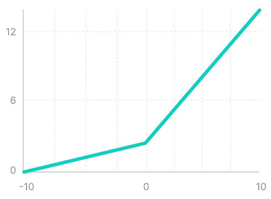


Figure 5.10: Leaky ReLU activation function

The main drawback is that having a small gradient value for negative values makes the learning of model parameters more time-consuming.

Softmax

The *softmax* activation function is based in the sigmoid function that works to calculate the probability values. The softmax can be described as a combination of multiple sigmoids, in which the relative probabilities for each class/output are computed. It is most commonly used as an activation function for the last layer of the neural network in the case of multiclass classification.

$$softmax(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{5.14}$$

5.2.5 Optimization algorithms

Many times, in the training of a deep network, the gradient descent algorithm by itself is not enough to reach convergence, and it may happen that it remains stuck in sub-optima points and/or the model suffers from overfitting. To help it to reach its convergence towards the minimum of the loss function, some optimization techniques can be employed to enhance the training time. Moreover, it is of common use to exploit some state-of-the-art algorithms that help the model avoid the overfitting problem. Let us briefly give an overview of the ones used for this project.

Data augmentation

Data augmentation is a process of artificially increasing the amount of data by generating new data points from existing data. This includes adding minor alterations to the data or using machine learning models to generate new data points in the latent space of the original data to amplify the dataset. Regarding image datasets, the main transformations that are applied to the starting data points are the following.

- *Geometric transformations*: randomly flip, crop, rotate, or translate;
- *Color space transformations*: change RGB color channels, intensify any color;
- *Kernel filters*: sharpen or blur an image;
- *Random Erasing*: delete a part of the initial image;
- *Mixing images*: mix images with others.

Adam

The *Adam* optimization algorithm is a very efficient method of adaptively changing the learning rate hyperparameter in the SGD algorithm. Having a constant learning rate is not very effective in training a network, and Adam provides a very efficient solution for finding suitable learning rates depending on the connection weight between neurons. It is based on two other extensions of SGD that are:

- *Adaptive Gradient* (AdaGrad) that maintains a per-parameter learning rate that improves performance in problems with sparse gradient.
- *Root Mean Square Propagation* (RMS) that also keep per-parameter learning rates, adapted based on the average of recent magnitude of the gradients for the weight (e.g., how quickly it is changing)-

Adam takes the two techniques and puts them together. In fact, instead of adapting the parameter learning rates based on the average first moment (mean) as in RMSProp, Adam also makes use of the second moments average of the gradient. Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient. More formally, an estimate of such quantities is given by:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned} \tag{5.15}$$

However, such estimates are biased; therefore, an unbiased corrected definition is employed.

$$\begin{aligned} \hat{m}_t &= \tfrac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \tfrac{v_t}{1-\beta_2^t} \end{aligned} \tag{5.16}$$

From these equations, the weight update rule is finally obtained:

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} \tag{5.17}$$

where η is the learning rate and the others are hyperparameters that are usually fixed at the values suggested by the researchers that defined this optimization algorithm, hence: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$. A good starting value for η is 0.001.

Momentum

Momentum is an extension to the gradient descent optimization algorithm, and it is designed to accelerate the optimization process, e.g., to decrease the number of function evaluations required to reach the optima, or to improve the capability of the optimization algorithm, possibly reaching a better final result. Momentum is used to overcome the problem in SGD, for which it tends to be slowed down by the randomness given by the stochasticity of the process, which sometimes can make the gradient move in a random direction, even uphill. The name momentum derives from a physical analogy in which the negative gradient is a force that moves a particle through parameter space, according to Newtons laws of motion. Momentum involves adding an additional hyperparameter that controls the amount of history (momentum) to include in the update equation, i.e. the step to a new point in the search space. The hyperparameter value is defined in the range 0.0 to 1.0 and often has a value close to 1.0, such as 0.8, 0.9, or 0.99. The closer to 1 is the value, the larger will be the history taken into account.

Weight decay

Weight decay is a regularization technique that helps prevent the network from overfitting the training data, as well as avoid the exploding gradient problem. It works by adding a penalty term to the cost function, which has the effect of shrinking the weights during backpropagation. Usually, the L2 penalty is applied as a regularization term. The squared sum of the weights is added to the cost function together with a hyperparameter λ, so that during backpropagation the computation of the gradient takes into account such an extra term, and consequently the update of the weights will be affected. The penalized cost can be formulated as the usual loss function plus the penalty term:

$$C(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathfrak{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2n} \sum_{i=1}^{m} w_i^2 \tag{5.18}$$

Higher weights would tend to keep neurons active and would use extra resources to fit the training data as accurately as possible. As a consequence, they would be more likely to pick up more "noise". Therefore, the shrinking of the weights helps prevent overfitting because penalizing bigger terms helps keep the activations to a normal level.

Batch normalization

Batch normalization is an optimization algorithm that helps in deep network training and makes it faster and more stable. It consists of normalizing the activation vectors from hidden layers using the first and second statistical moments (mean and variance) of the current batch. This normalization step is applied right before the non-linear function. In popular deep learning frameworks, this operation is implemented as an actual layer to be inserted in the architecture of the model.

Dropout

Another technique specifically designed to prevent overfitting is *dropout.* It consists of randomly removing input and hidden units during the processing of each pattern. The idea is that dropout regularizes each hidden unit to decrease the chances of learning statistical noise, thus avoiding the propagation of such noise in the rest of the network. It has a very efficient implementation because it is just needed to multiply the layer by a binary "mask" that contains some zeros with a fixed drop probability p that is usually a hyperparameter to be tuned.

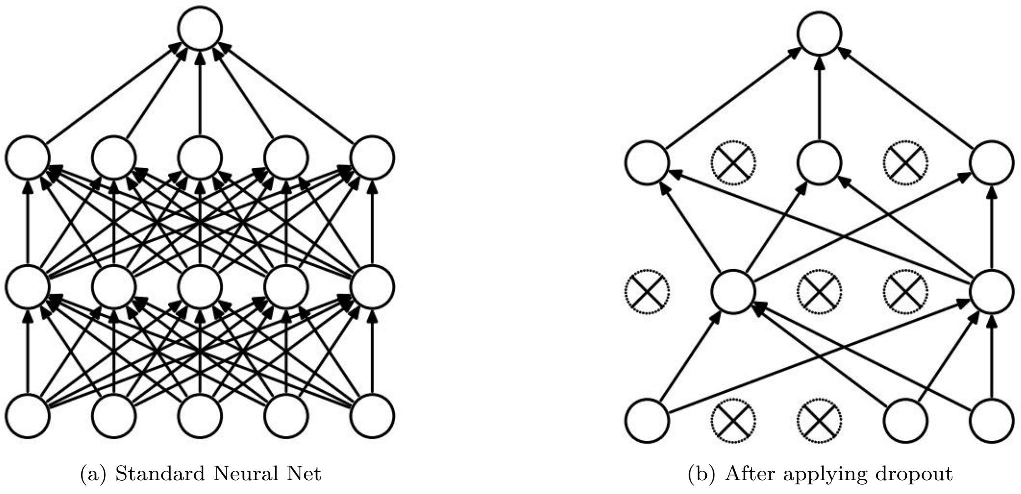


Figure 5.11: Example of application of dropout to a FCNN [6]

Early stopping

Early stopping is an optimization technique used to reduce overfitting without compromising the accuracy of the model. The main idea behind early stopping is to stop training before a model starts to overfit. The stopping of the training phase can be set up in many ways, for example, by checking on the weight update becoming significantly smaller; hence the network is not learning anymore. Another way of early stopping the training is by checking on the train loss and the validation loss: if for a repeated number of training epochs, the validation loss is higher than the train loss in the previous epoch, then the training is stopped. The number of epochs for this to happen is a hyperparameter to be tuned, usually referred to as the *patience* parameter.

[6]Source: https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

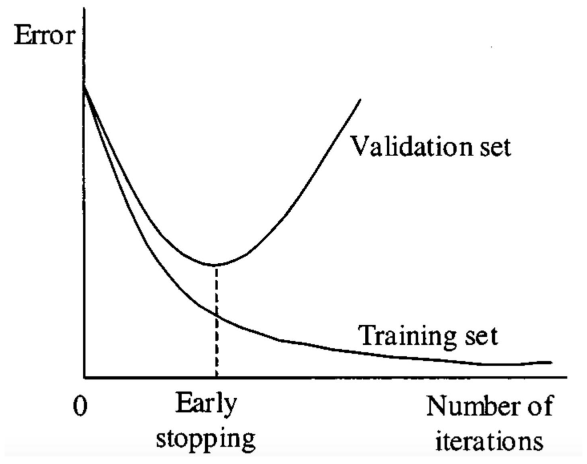


Figure 5.12: Early stopping

5.2.6 K-Fold cross validation

Cross-validation methods are techniques to assess the ability of a model to generalize to an independent data set. Cross-validation is a resampling method that uses different portions of the data to test and train a model on different iterations. Among the family of cross-validation techniques, *k-fold cross-validation* can be used to perform some model analysis and extract relevant information from the results. In k-fold cross-validation, the initial sample is randomly divided into k subsamples of equal size. One subsample of the total of k subsamples is kept as validation data for model testing, and the remaining $k-1$ subsamples are used as training data. Thereafter, the cross-validation procedure is carried out k times, using the validation data from each of the k sub-samples a single time. To create a single estimate, the k findings can then be averaged. The fact that all observations are used for both training and validation, and that each observation is used for validation exactly once, distinguishes this approach from repeated random subsampling. Although a 10-fold cross-validation is frequently used, the parameter k is not always fixed and depends on the application.

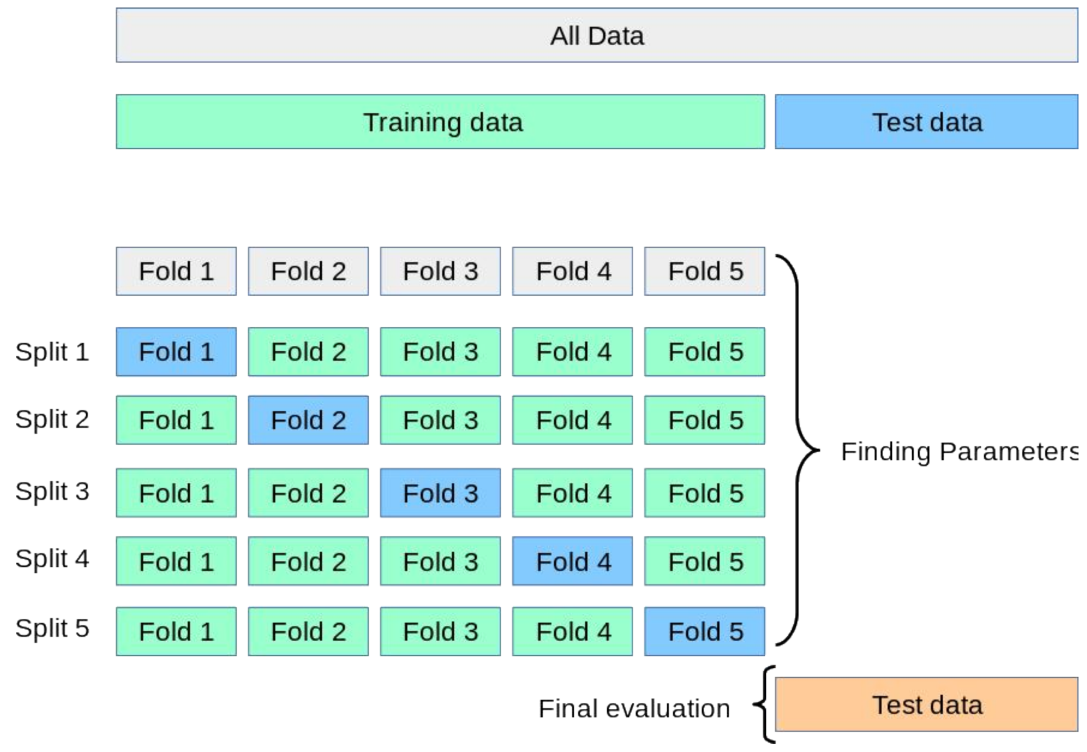


Figure 5.13: Example of data set splitting for k-fold cross-validation with $k = 5$ [7]

5.2.7 Evaluation metrics

In the assessment of the models, the following metrics were used.

Confusion matrix and statistics

In a classification scenario, it is useful to refer to samples with the following subdivision:

	Predicted Positive (PP)	**Predicted Negative (PN)**
Actual Positive (P)	True Positive (TP)	False Positive (FP)
Actual Negative (N)	False Negative (FN)	True Negative (TN)

Table 5.1: 2x2 confusion matrix

[7]Source: https://scikit-learn.org/stable/modules/cross_validation.html

This is referred to as a *confusion matrix*, in particular in a binary classification framework. It can be also extended to a multi-label classification problem, as it was done in this case.
From the confusion matrix, some performance statistics can be retrieved:

- *Model accuracy*: number of correctly classified samples divided by the total number of samples. The closer it is to 1, the more accurate is the model.

$$\frac{TP + TN}{TP + TN + FP + FN} \tag{5.19}$$

- *Recall*: Percentage of correctly classified instances of a class.

$$\frac{TP}{TP + FN} \tag{5.20}$$

- *Precision*: Fraction of positive instances correctly labelled.

$$\frac{TP}{TP + FP} \tag{5.21}$$

ROC curve

ROC (Receiving Operation Characteristic) curves typically feature true positive rate on the Y axis, and false positive rate on the X axis. This means that the top left corner of the plot is the ideal point - a false positive rate of zero, and a true positive rate of one. The larger is the area under the curve (AUC) the better. The steepness of ROC curves is also important, since it is ideal to maximize the true positive rate while minimizing the false positive rate. ROC curves are typically used in binary classification to study the output of a classifier. They can also be extended to multi-label classification, for which it is necessary to binarize the output.

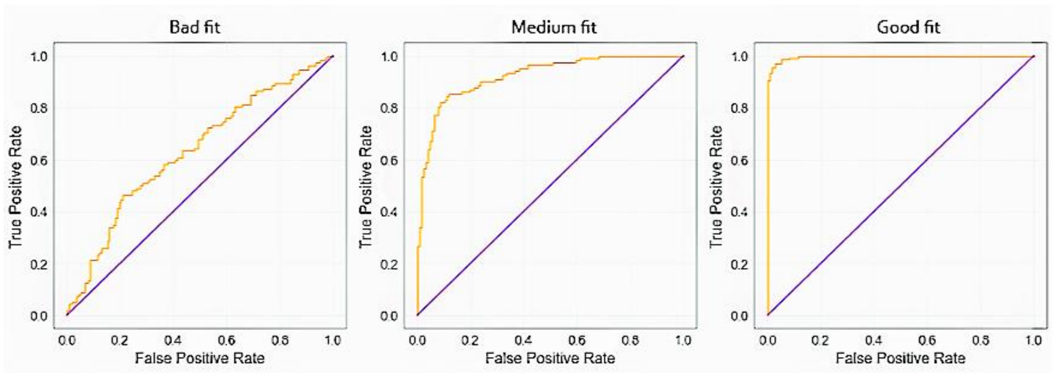


Figure 5.14: ROC curves comparison

5.3 Development of the model

Let us now continue the analysis of the steps done that led to the final version of the classification model. The above mentioned structures, CNNs in particular, were the main focus of this research, As anticipated in the project statement in Chapter 1, the choice of using a CNN as deep learning architecture was mainly dictated by the nature of the data at disposal. As it was outlined in Chapter 4, the output of the processing pipeline is an image, which is hence well suited to be fed into a convolutional network for the reasons listed before.
Therefore, the development of the model started from the beginning with this idea in mind, and other architectures were excluded from the principle. Many trials regarding the architecture and its structure have been done over the time of the experiment, especially due to the growth of the dataset. In fact, it is essential to mention that, at the beginning of the experiment, only a small dataset with 5 participants was available. This allowed to initialize the project and to start testing some networks, with the idea in mind that the data collection should have been expanded as soon as possible. As mentioned in Chapter 3, the final number of participants was of 22, which by the end of the experiment was enough to extract some relevant results. Of course, deep learning requires as many data as possible, hence and higher number of participants could have probably improved even more

the obtained results.
It is also worth to mention that, even if the data increased over time, there were no extreme changes in the architecture, and the modifications regarded mainly the set of hyperparameters, which tuning process is reported in Chapter 6.
For the sake of this report, a total of three models are outlined in the following paragraphs. Among the whole of the different trials that were tested for the duration of the experiment, these were the most significant ones that led to the final version.

5.3.1 Model 1

As anticipated before, the architecture chosen from the beginning is a CNN. The main differences between the various models lie on the structure, the number of hidden layers and units etc. Let us start with the very first significant model that will be addressed as *Model 1*. This network was employed at the beginning when the dataset was not complete yet, but contained only 5 participants. The structure of the network follows a very standard CNN modelling style, with different convolutional layers followed by pooling layers, and a final set of fully connected layers for the classification part. Model 1 is composed by:

- Input layer
- Convolutional layer 1 (Conv2D + BatchNorm + ReLU + MaxPool)
- Convolutional layer 2 (Conv2D + BatchNorm + ReLU + MaxPool)
- Convolutional layer 3 (Conv2D + BatchNorm + ReLU + MaxPool + Dropout)
- Flattening layer
- Classifier (FC1 + ReLU + Dropout + FC2 + ReLU + Output)

In Figure 5.15 a sketch of the model is represented, while in Table 5.2 all the relevant parameters regarding each layer composing the network is reported. From this representation, the batch normalization and the dropout layers are missing. This is to keep the report consistent, namely this chapter contains all the relevant information about the model structure, while Chapter 6 regards everything related to the training phase and the choices made regarding the hyperparameters contained in the model, dropout and BatchNorm parameters included.

Note how the output layer contains 20 units, namely one for each of the 20 classes of the dataset presented in Chapter 3

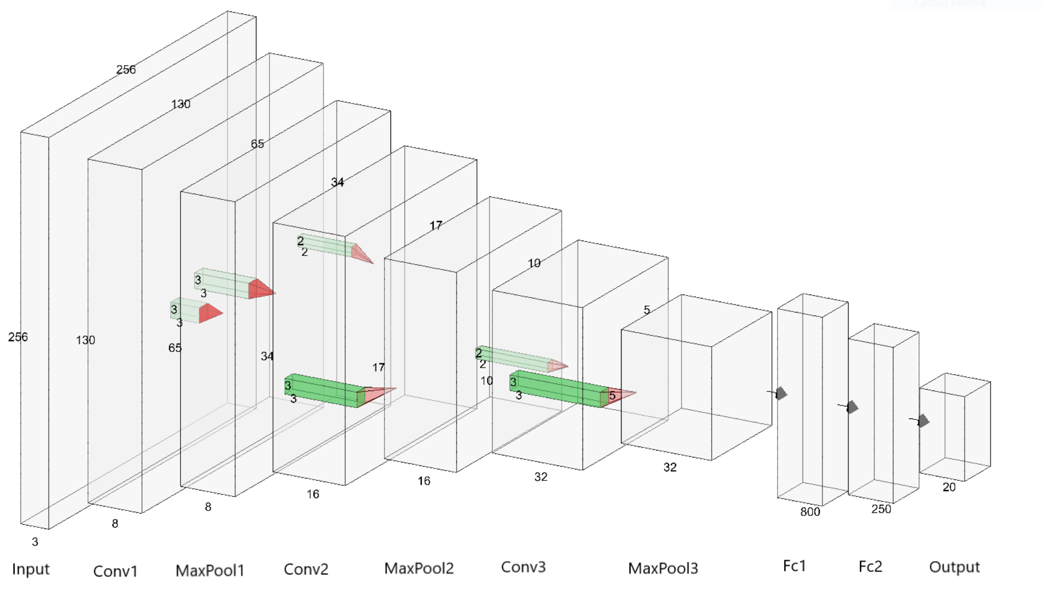


Figure 5.15: Visual representation of Model 1 structure

Layer	Input size (channels, height, width)	Output size (channels, height, width)	Kernel size (height, width)	Padding	Stride	Total number of parameters
Input	(3, 256, 256)					
Conv1	(3, 256, 256)	(8, 130, 130)	(3, 3)	3	2	135’200
MaxPool1	(8, 130, 130)	(8, 65, 65)	(2, 2)	0	2	
Conv2	(8, 65, 65)	(16, 34, 34)	(3, 3)	3	2	18’496
MaxPool2	(16, 34, 34)	(16, 17, 17)	(2, 2)	0	2	
Conv3	(16, 17, 17)	(32, 10, 10)	(3, 3)	3	2	3’200
MaxPool3	(32, 10, 10)	(32, 5, 5)	(2, 2)	0	2	800
Flatten	(32, 5, 5)	(1, 800)				
Fc1	(1, 800)	(1, 800)				800
Fc2	(1, 800)	(1, 250)				250
Output	(1, 250)	(1, 20)				20

Table 5.2: Model 1 layers parameters

5.3.2 Model 2

The development of the second network relevant for this manuscript is reported here and will be identified as *Model 2*. It was tried after the deployment of Model 1 to determine what would be the impact of removing one convolutional layer. At the time of this architecture, the dataset was still not complete yet, therefore it was in the interest to experiment less complex networks. The model is much similar to the previous one, with the difference that the third layer was removed from the architecture. Similarly to before, Figure 5.16 and Table 5.3 provide the main characteristics of this network.

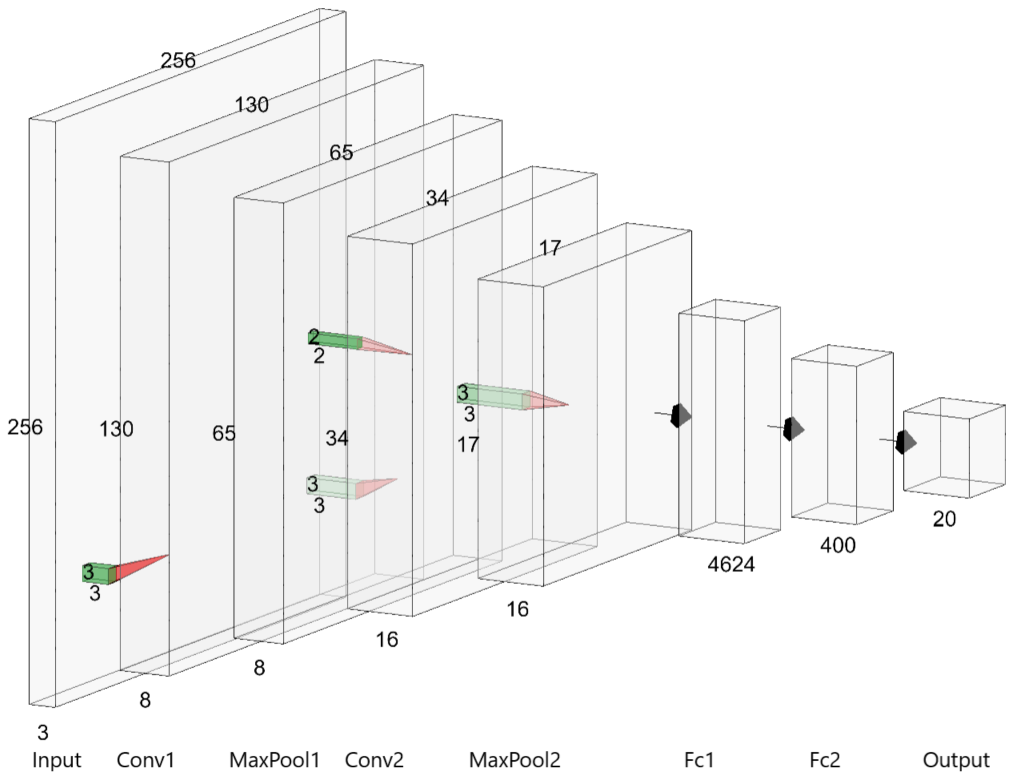


Figure 5.16: Visual representation of Model 2 structure

Layer	Input size (channels, height, width)	Output size (channels, height, width)	Kernel size (height, width)	Padding	Stride	Total number of parameters
Input	(3, 256, 256)					
Conv1	(3, 256, 256)	(8, 130, 130)	(3, 3)	3	2	135’200
MaxPool1	(8, 130, 130)	(8, 65, 65)	(2, 2)	0	2	
Conv2	(8, 65, 65)	(16, 34, 34)	(3, 3)	3	2	18’496
MaxPool2	(16, 34, 34)	(16, 17, 17)	(2, 2)	0	2	
Flatten	(16, 17, 17)	(1, 4’624)				
Fc1	(1, 4’624)	(1, 400)				800
Fc2	(1, 400)	(1, 100)				250
Output	(1, 100)	(1, 20)				20

Table 5.3: Model 2 layers parameters

5.3.3 Model 3 - Final model

This paragraph regards *Model 3*, namely what can be considered as the final version that was used for the classification task, for it was the one that gave the most interesting results. As one can observe from Figure 5.4 and Table 5.4, this network is not much different from the previous ones, however it has the particularity of having more convolutional layers and less max pooling layers. Moreover, a different set of padding parameters was employed in this case with respect to the others. The choice of removing some pooling layers was due to the fact that, otherwise, the image would have shrunk too fast without giving the chance to the network to learn useful features in its deeper layers. Therefore, it was necessary to reduce both pooling layers and padding parameters in order to not make the image too small.
This network is the result of a process of different trials with similar architectures, which however shared the fact that they were born once the full dataset (22 participants) was available. The driving criterion for the different choices was the fact that, with the dataset being larger, more data were to be processed, hence it would have taken more resources and hidden units to learn properly the features of each gesture.
The main differences between these networks, which in the end led to Model 3, regarded mainly the number of neurons in the fully connected layers, and the choice of using max pooling after each convolution or not. In the end, the evaluation of the results brought to the final architecture displayed hereafter.

In general, regarding the choice of some layer parameters such as stride, kernel size, number of channels etc. in the whole set of models, decisions about their values were highly influenced by the limited amount of computational resources. In an ideal scenario, an automatic hyperparameter tuning process should be performed, for example by means of random search or using optimization tools specifically designed for deep learning. However, it was not feasible for this case due to some restrictions given by the computational power at disposal. In conclusion, the final

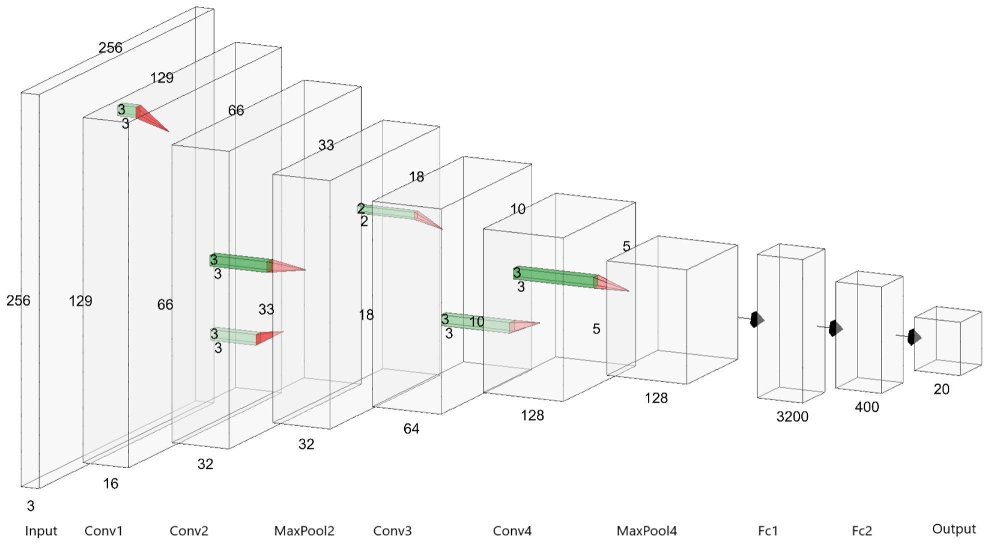


Figure 5.17: Visual representation of Model 3 structure

Layer	**Input size (channels, height, width)**	**Output size (channels, height, width)**	**Kernel size (height, width)**	**Padding**	**Stride**	**Total number of parameters**
Input	(3, 256, 256)					
Conv1	(3, 256, 256)	(16, 129, 129)	(3, 3)	2	2	266'256
Conv2	(16, 129, 129)	(32, 66, 66)	(3, 3)	2	2	139'392
Maxpool2	(32, 66, 66)	(32, 33, 33)	(2, 2)	0	2	
Conv3	(32, 33, 33)	(64, 18, 18)	(3, 3)	2	2	20'736
Conv4	(64, 18, 18)	(128,10,10)	(3, 3)	2	2	12'800
MaxPool4	(128, 10, 10)	(128, 5, 5)	(2, 2)	0	2	3'200
Flatten	(128, 5, 5)	(1, 3200)				
Fc1	(1, 3200)	(1, 500)				500
Fc2	(1, 500)	(1, 100)				100
Output	(1, 100)	(1, 20)				20

Table 5.4: Model 3 layers parameters

set of parameters was a result of a manual trial and error process with some of the most common values adopted in similar architectures.

6 RESULTS

In this chapter are presented the methodologies followed for the training and the validation of the models introduced in Chapter 5. Successively, the results of the most noticeable training sessions are exhibited through different metrics and plots, in order to also make a comparison between the various models and their characteristics.

6.1 Training and validation

Let us here describe the most important aspects of the training phase. The idea is to give an overview of the main steps that, over the little experiment time, led to the development of the final version of the model. The first trials that will be reported acting as example to describe the thought process behind the different choices taken to improve the performances of the classifier, therefore only the training part is contained. Successively, more focus will be given to the final model, which was the most promising one, hence together with the training phase it is valuable to report the main results regarding the validation and test of the model.

6.1.1 Model training

The description of the most relevant training sessions for the models outlined in Chapter 5 are hereby reported. The main differences that can be found here are about the hyperparameters choice. As mentioned before, the main limitation here was the computational power and resources at disposal, therefore many things that would be performed on an ideal deep learning application scenario could not be done here, such as running some search algorithm for the hyperparameters selection. Therefore, also here the final set of hyperparameters was the result of a hit-and-miss experimental process.
Moreover, regarding the training of Model 1 and Model 2, it is not worth it to dwell too much on the details of the training, since it was a very first setup of the classification task in which not all the resources were available and the dataset was not complete yet. The more important training phase for Model 3, instead, is valuable for the scope of this thesis and more focus will be given on that.
The training was performed on a cloud computing platform (Google Cloud Platform [1]) with a running virtual machine instance characterized by the following elements: 8 N1-standard virtual CPUs; 30 GBs RAM; 1 NVIDIA Tesla T4 GPU. The whole implementation was carried out in the *Pytorch* deep learning framework, which provides a set of objects and classes specifically designed for deep learning algorithms. Many optimization techniques mentioned in the previous section are already implemented in Pytorch, and they require only the choice of the hyperparameters, which are the focus on the next subsections.

[1]https://cloud.google.com/

Training A

Let us refer with *Training A* to the training phase performed for Model 1 and 2. There is no distinction between the two because the training hyperparameters were kept the same, and the difference was given only by the architecture. The training was performed in the early stages of the development, hence not all the optimization and regularization techniques outlined before were implemented in this case. Let us also recall that, in this context, the dataset size was still very small.
A first trial was done by previously performing an augmentation on the data, namely by adding random vertical and horizontal flipping, and randomly rotating some samples. In Figure 6.1 are shown examples of transformations applied to a dataset batch.

The training was then performed with the following parameters:

Optimizer	**Learning Rate**	**Momentum**	**Dropout prob. 1**	**Dropout prob. 2**	**Epochs**	**Batch size**
SGD	0.0001	0.7	0.0661	0.0968	10	16

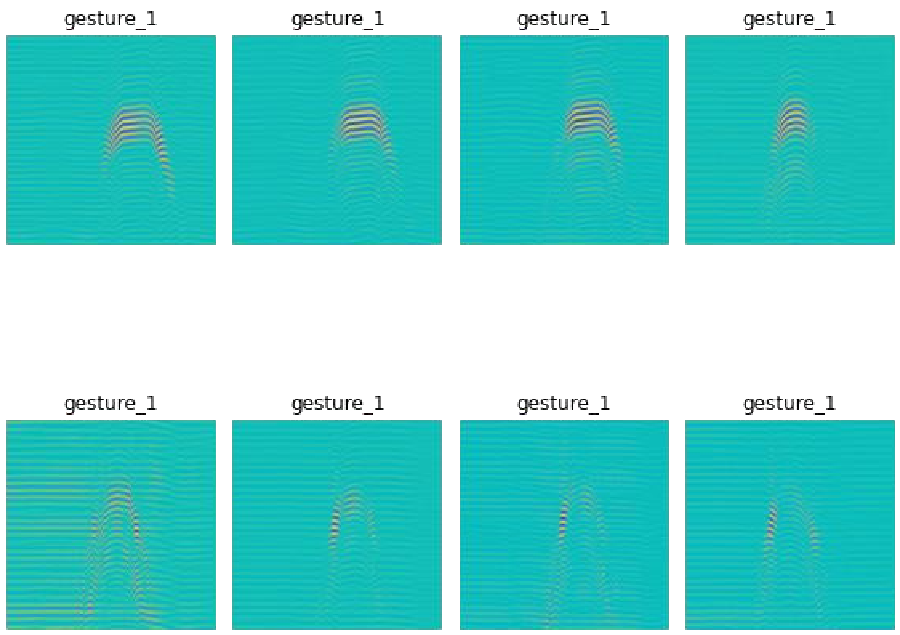
Table 6.1: Hyperparameters for Training A

Two different trainings were actually tried with this set of parameters: in the first one, the different transformations mentioned before were actuated, while in the second this modification got removed.

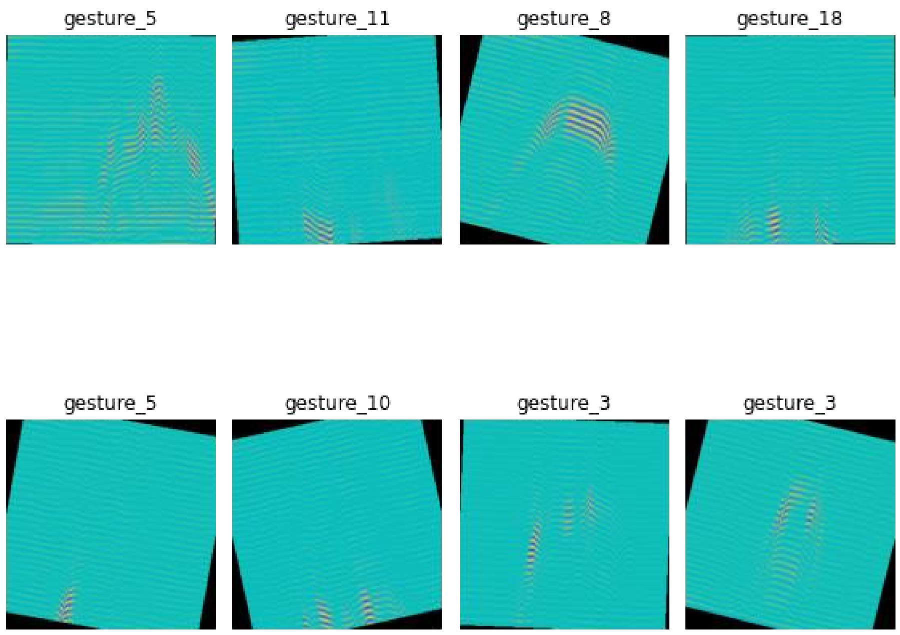
Model 2 followed the same criteria, with the idea of comparing the performance with one less convolutional layer. Therefore, the same training parameters were used as in Table 6.1. The results are presented later in this chapter.

Training B

The more important Model 3 was trained on the integral dataset, hence being the true test for the correct functioning of the classification system. As a result of a trial-and-error process carried out through the experiment, the following choices were taken with respect to the training of this model:



(a) Batch without transformations



(b) Batch with transformations

Figure 6.1: Example of augmentations on the training set

- No data augmentation was performed. The initial idea of applying geometric transformations to the images to augment the dataset was left out after observing a decrease of performances when used.

- Use Adam optimizer. A very typical approach to improve SGD convergence is to introduce an optimizer, and Adam usually performs very well in many contexts.

- Apply weight decay, dropout and batch normalization. Regularization techniques help prevent overfitting and are likely to improve overall performances of the model.

- Use early stopping to stop the training before overfitting.

The final set of hyperparameters that gave the most interesting results are reported in Table 6.2

Optimizer	**Learning Rate**	**Weight decay**	**Dropout prob. 1**	**Dropout prob. 2**	**Epochs**	**Patience**	**Batch size**
Adam	0.002	0.0001	0.2002	0.1945	43	4	16

Table 6.2: Hyperparameters for Training B

6.1.2 Model validation

To assess the model and test effectively its ability to generalize on unseen data from the dataset, two different strategies were applied.
A first, more classical approach was to perform the k-fold cross-validation technique exhibited in the Subsection 5.2.6 with $k = 5$, both for computational power reasons and for consistency with the 80%/20% splitting ratio used until now. For each fold, the training was repeated on a different subset composed of 80% of the total dataset, and the remaining was used to assess classification performances. This validation process was used mainly to see the model capability of generalizing on the whole dataset, and to not have very high mismatch between the data distribution and the

classification accuracies.
Another kind of validation, instead, was carried out more "manually" in the end, after a relevant consideration. The ultimate goal for a classification model of this kind would be of being able to recognize hand gestures performed by anyone, regardless of the person. This means that the algorithm should be strong enough to generalize on unseen participants and still be able to classify correctly the movement. If the model learns features from the training set taken from the whole set, this means that it will already see at least some data regarding gestures performed by all users composing the dataset. A more sensible test would be that of training the model on a dataset composed of say 80% of the users, and then test the model on the remaining 20%. In this case, it would be much clearer if the model is able distinguish the gestures regardless of the person doing it, or if the features learned on each gesture are also influenced by the user's way of performing the movement. In this scenario, two trainings were performed: one excluding the totality of the test set's participants from the training; one including a small percentage of the test set into the training set.

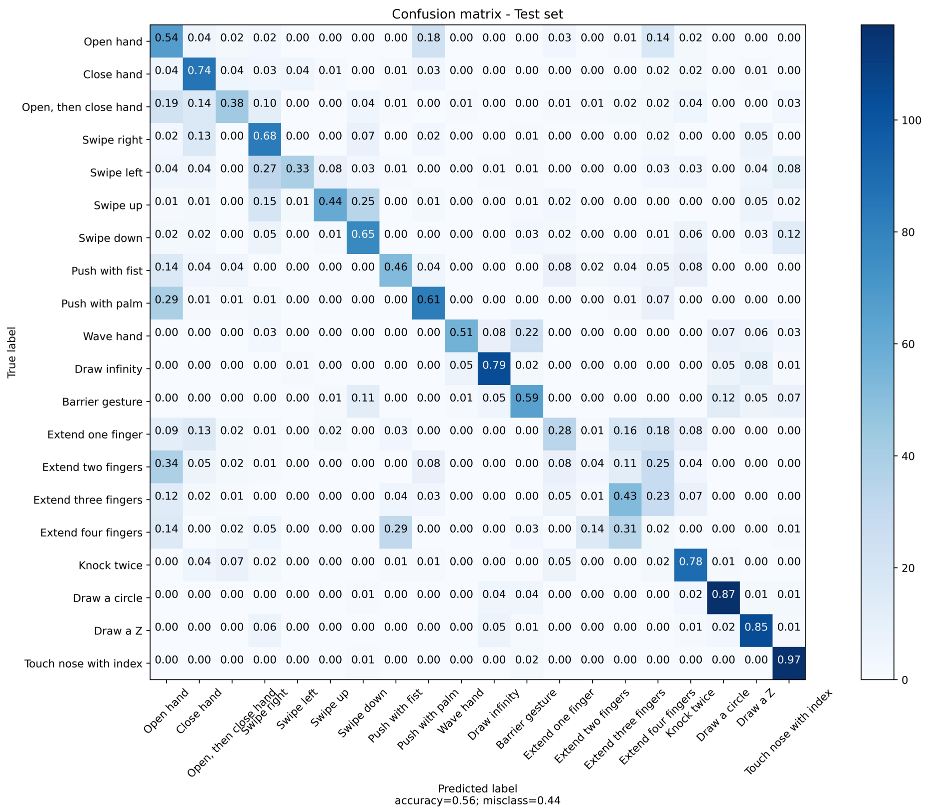
6.2 Results

6.2.1 Training A

Let us report here the results regarding the training of Model 1 and Model 2 which, again, were performed on the initial 5 participants dataset.
In Table 6.3 are reported the metrics regarding the first training performed over the dataset with images affected by the geometric transformations mentioned above, while in Figure 6.2 the relative confusion matrix is presented.

Classification Accuracy	**Precision**	**Recall**
0.562	0.568	0.567

Table 6.3: Results of Model 1 with dataset with transformations



	Open hand	Close hand	Open, then close hand	Swipe right	Swipe left	Swipe up	Swipe down	Push with fist	Push with palm	Wave hand	Draw infinity	Barrier gesture	Extend one finger	Extend two fingers	Extend three fingers	Extend four fingers	Knock twice	Draw a circle	Draw a Z	Touch nose with index
Open hand	0.54	0.04	0.02	0.02	0.00	0.00	0.00	0.00	0.18	0.00	0.00	0.00	0.03	0.00	0.01	0.14	0.02	0.00	0.00	0.00
Close hand	0.04	0.74	0.04	0.03	0.04	0.01	0.00	0.01	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.02	0.00	0.01	0.00
Open, then close hand	0.19	0.14	0.38	0.10	0.00	0.00	0.04	0.01	0.00	0.01	0.00	0.00	0.01	0.01	0.02	0.02	0.04	0.00	0.00	0.03
Swipe right	0.02	0.13	0.00	0.68	0.00	0.00	0.07	0.00	0.02	0.00	0.00	0.01	0.00	0.00	0.00	0.02	0.00	0.00	0.05	0.00
Swipe left	0.04	0.04	0.00	0.27	0.33	0.08	0.03	0.01	0.00	0.00	0.01	0.01	0.00	0.00	0.00	0.03	0.03	0.00	0.04	0.08
Swipe up	0.01	0.01	0.00	0.15	0.01	0.44	0.25	0.00	0.01	0.00	0.00	0.01	0.02	0.00	0.00	0.00	0.00	0.00	0.05	0.02
Swipe down	0.02	0.02	0.00	0.05	0.00	0.01	0.65	0.00	0.00	0.00	0.00	0.03	0.02	0.00	0.00	0.01	0.06	0.00	0.03	0.12
Push with fist	0.14	0.04	0.04	0.00	0.00	0.00	0.00	0.46	0.04	0.00	0.00	0.00	0.08	0.02	0.04	0.05	0.08	0.00	0.00	0.00
Push with palm	0.29	0.01	0.01	0.01	0.00	0.00	0.00	0.00	0.61	0.00	0.00	0.00	0.00	0.00	0.01	0.07	0.00	0.00	0.00	0.00
Wave hand	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.51	0.08	0.22	0.00	0.00	0.00	0.00	0.00	0.07	0.06	0.03
Draw infinity	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.05	0.79	0.02	0.00	0.00	0.00	0.00	0.00	0.05	0.08	0.01
Barrier gesture	0.00	0.00	0.00	0.00	0.00	0.01	0.11	0.00	0.00	0.01	0.05	0.59	0.00	0.00	0.00	0.00	0.00	0.12	0.05	0.07
Extend one finger	0.09	0.13	0.02	0.01	0.00	0.02	0.00	0.03	0.00	0.00	0.00	0.00	0.28	0.01	0.16	0.18	0.08	0.00	0.00	0.00
Extend two fingers	0.34	0.05	0.02	0.01	0.00	0.00	0.00	0.00	0.08	0.00	0.00	0.00	0.08	0.04	0.11	0.25	0.04	0.00	0.00	0.00
Extend three fingers	0.12	0.02	0.01	0.00	0.00	0.00	0.00	0.04	0.03	0.00	0.00	0.00	0.05	0.01	0.43	0.23	0.07	0.00	0.00	0.00
Extend four fingers	0.14	0.00	0.02	0.05	0.00	0.00	0.00	0.29	0.00	0.00	0.00	0.03	0.00	0.14	0.31	0.02	0.00	0.00	0.00	0.01
Knock twice	0.00	0.04	0.07	0.02	0.00	0.00	0.00	0.01	0.01	0.00	0.00	0.00	0.05	0.00	0.00	0.02	0.78	0.01	0.00	0.00
Draw a circle	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.04	0.04	0.00	0.00	0.00	0.00	0.02	0.87	0.01	0.01
Draw a Z	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.00	0.00	0.00	0.05	0.01	0.00	0.00	0.00	0.00	0.01	0.02	0.85	0.01
Touch nose with index	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.97

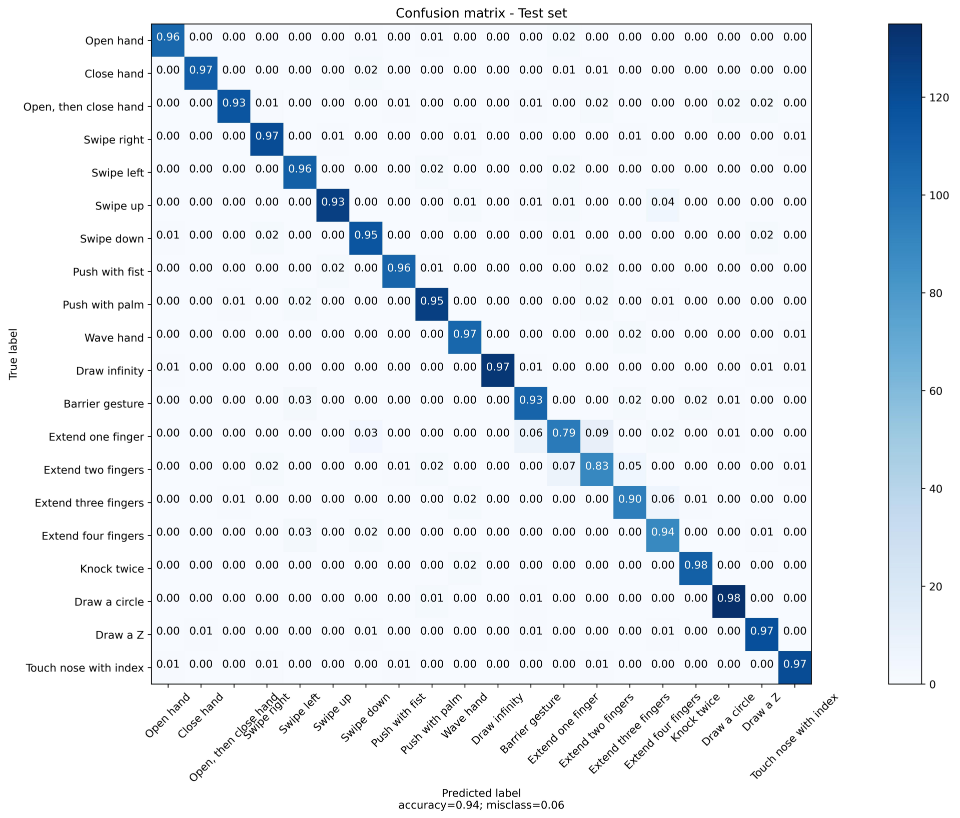
Figure 6.2: Model 1 Confusion matrix with transformed dataset

As it can be observed, the classification accuracy is quite low in this context, therefore the idea of removing the transformations was applied in the next trial. Table 6.4 and Figure 6.3 confirm that this solution was quite effective, since the accuracy improved drastically.

Classification Accuracy	**Precision**	**Recall**
0.9496	0.9592	0.9603

Table 6.4: Results of Model 1 with dataset without transformations

It is clear that the first implementation with the transforms was not very effective. This actually makes sense because the gesture is related to the direction in which it is performed. Rotating and flipping the image corresponds to performing a different gesture. After this first analysis, the data augmentation was then left aside, and



	Open hand	Close hand	Open, then close hand	Swipe right	Swipe left	Swipe up	Swipe down	Push with fist	Push with palm	Wave hand	Draw infinity	Barrier gesture	Extend one finger	Extend two fingers	Extend three fingers	Extend four fingers	Knock twice	Draw a circle	Draw a Z	Touch nose with index
Open hand	0.96	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.01	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Close hand	0.00	0.97	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.00
Open, then close hand	0.00	0.00	0.93	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.02	0.00	0.00	0.00	0.02	0.02	0.00
Swipe right	0.00	0.00	0.00	0.97	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.01
Swipe left	0.00	0.00	0.00	0.00	0.96	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Swipe up	0.00	0.00	0.00	0.00	0.00	0.93	0.00	0.00	0.00	0.01	0.00	0.01	0.01	0.00	0.00	0.04	0.00	0.00	0.00	0.00
Swipe down	0.01	0.00	0.00	0.02	0.00	0.00	0.95	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.02	0.00
Push with fist	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.96	0.01	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00
Push with palm	0.00	0.00	0.01	0.00	0.02	0.00	0.00	0.00	0.95	0.00	0.00	0.00	0.00	0.02	0.00	0.01	0.00	0.00	0.00	0.00
Wave hand	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.97	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.01
Draw infinity	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.97	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01
Barrier gesture	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.93	0.00	0.00	0.02	0.00	0.02	0.01	0.00	0.00
Extend one finger	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.00	0.00	0.06	0.79	0.09	0.00	0.02	0.00	0.01	0.00	0.00
Extend two fingers	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.01	0.02	0.00	0.00	0.00	0.07	0.83	0.05	0.00	0.00	0.00	0.00	0.01
Extend three fingers	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.90	0.06	0.01	0.00	0.00	0.00
Extend four fingers	0.00	0.00	0.00	0.00	0.03	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.94	0.00	0.00	0.01	0.00
Knock twice	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.00	0.00	0.00
Draw a circle	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.98	0.00	0.00
Draw a Z	0.00	0.01	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.97	0.00
Touch nose with index	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.97

Figure 6.3: Model 1 Confusion matrix with normal dataset

the dataset was kept as it was. Moreover, it was also clear how it was not worth it to fix a priori number of training epochs, but rather let the training run and use the training and validation loss values to eventually stop the process.

The classification accuracy increased drastically after dropping the transformations. However, a lot of confusion can still be observed between the classes 13, 14, 15, 16. This finds an explanation in the nature of such gestures, since they refer to the movement of extending one, two, three, etc. fingers. It is clear how the model has trouble in differentiating between these very similar gestures. This may also be due to the signal that is not completely noiseless, therefore it is easier to confuse two fingers with three, etc. Less confusion is made between very diverse gestures such as, for example, drawing a circle and pushing the palm. In the training with the transformations applied to the dataset this is even more visible, however, it is a

feature that we find often in the many results shown also in the following.
Model 2 was then tested after the training with parameters reported in the previous section, and the results are reported in Figure 6.4 and Table 6.5.

Classification Accuracy	**Precision**	**Recall**
0.946	0.9416	0.9427

Table 6.5: Results of Model 2

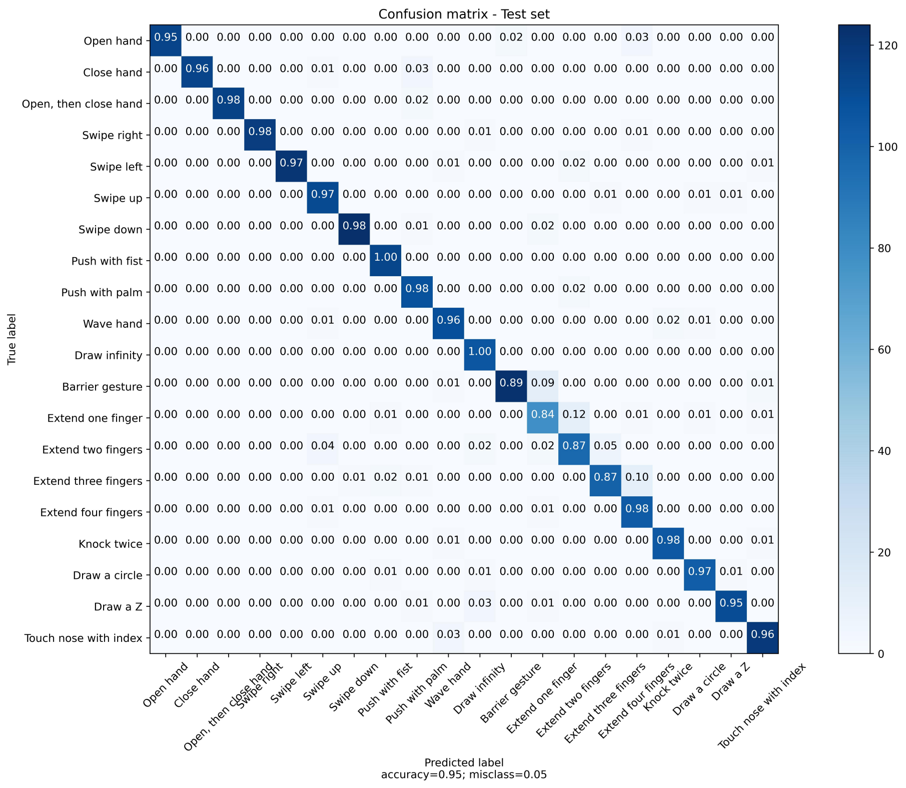


Figure 6.4: Model 2 Confusion matrix

It is evident how the modification of the architecture did not impact at all the performances of the classifier. However, it has to be considered that this phase was carried out in the early stages, with the incomplete dataset, therefore it was not exactly clear if such good results were actually due to an overfitting of the data. Only with the full size dataset and a validation of the model this could have been cleared out.

6.2.2 Training B

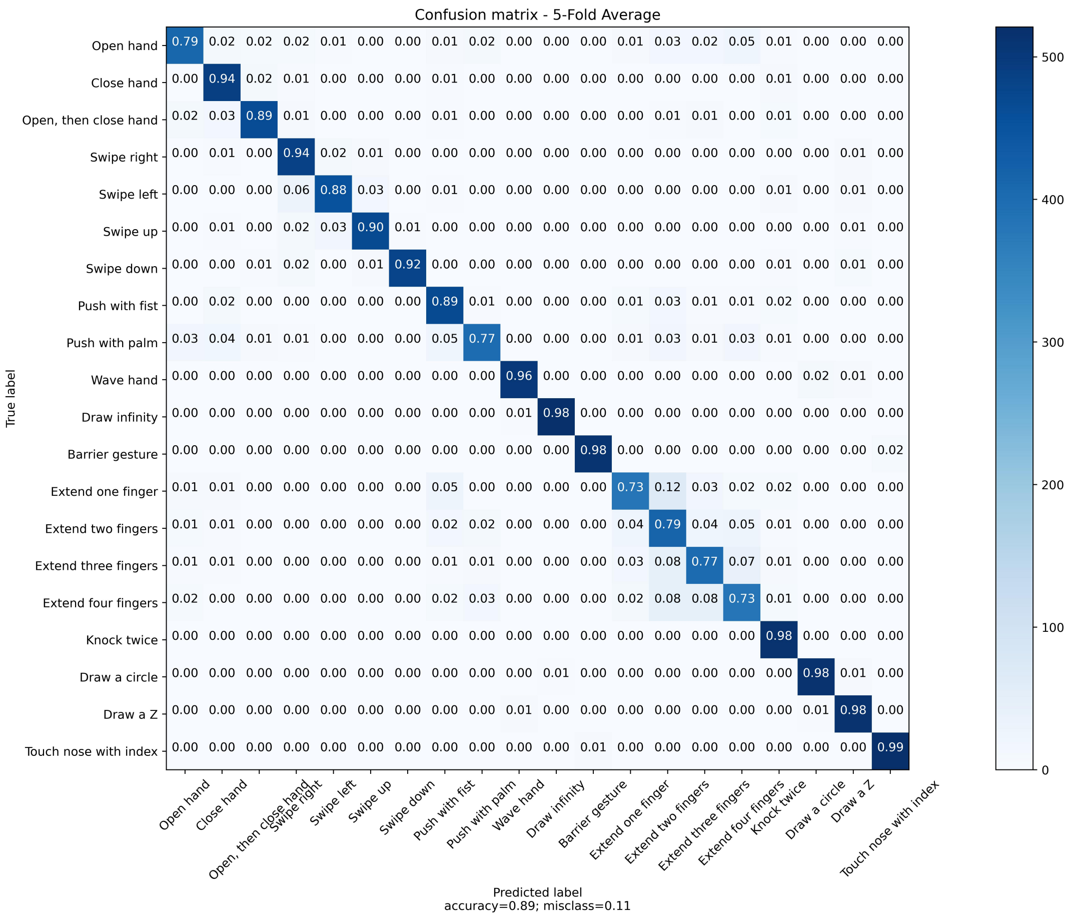
The more interesting results refer to Model 3 and its training, for it was a more complex model and the dataset was more expanded. As anticipated in the methods section, regarding this architecture there are more results to exhibit.
The model train-validation was performed first with the cross-validation technique mentioned before. For each fold, the entire dataset was divided into 80% for the training and 20% for the test with a random sampler, so that the data points ending up in the two sets would change each time. For each fold, the training was repeated from scratch so that the network would not be biased from the other folds. In Table 6.6 are reported the results for each fold. Figure 6.5, instead, represents the confusion matrix containing the average values for each fold. Another trial may be performed with a different value of k, for example $k = 10$. This could potentially increase the classification accuracy, because the model would be exposed to 90% of the total dataset for the training, hence there would be more samples available for the feature learning part, however it may be less realistic in terms of a real-world application performance.

	Classification Accuracy	**Precision**	**Recall**
Fold 1	0.863	0.8691	0.8635
Fold 2	0.8661	0.8758	0.8662
Fold 3	0.8368	0.8502	0.8368
Fold 4	0.9222	0.9251	0.9221
Fold 5	0.9186	0.9203	0.9184

Table 6.6: Results of Model 3 5-fold cross-validation

Some observations arise from these results:

- The average accuracy is good overall ($\sim$ 89% on average);
- The results are quite similar over the different folds, meaning that: the dataset is well balanced; the model is not influenced by the data that it is trying to classify;



True label \ Predicted label	Open hand	Close hand	Open, then close hand	Swipe right	Swipe left	Swipe up	Swipe down	Push with fist	Push with palm	Wave hand	Draw infinity	Barrier gesture	Extend one finger	Extend two fingers	Extend three fingers	Extend four fingers	Knock twice	Draw a circle	Draw a Z	Touch nose with index
Open hand	0.79	0.02	0.02	0.02	0.01	0.00	0.00	0.01	0.02	0.00	0.00	0.00	0.01	0.03	0.02	0.05	0.01	0.00	0.00	0.00
Close hand	0.00	0.94	0.02	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00
Open, then close hand	0.02	0.03	0.89	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.01	0.00	0.00	0.00
Swipe right	0.00	0.01	0.00	0.94	0.02	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00
Swipe left	0.00	0.00	0.00	0.06	0.88	0.03	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.01	0.00
Swipe up	0.00	0.01	0.00	0.02	0.03	0.90	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00
Swipe down	0.00	0.00	0.01	0.02	0.00	0.01	0.92	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.01	0.00
Push with fist	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.89	0.01	0.00	0.00	0.00	0.01	0.03	0.01	0.01	0.02	0.00	0.00	0.00
Push with palm	0.03	0.04	0.01	0.01	0.00	0.00	0.00	0.05	0.77	0.00	0.00	0.00	0.01	0.03	0.01	0.03	0.01	0.00	0.00	0.00
Wave hand	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.96	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.01	0.00
Draw infinity	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Barrier gesture	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02
Extend one finger	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.00	0.73	0.12	0.03	0.02	0.02	0.00	0.00	0.00
Extend two fingers	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.02	0.02	0.00	0.00	0.00	0.04	0.79	0.04	0.05	0.01	0.00	0.00	0.00
Extend three fingers	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.00	0.00	0.03	0.08	0.77	0.07	0.01	0.00	0.00	0.00
Extend four fingers	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.03	0.00	0.00	0.00	0.02	0.08	0.08	0.73	0.01	0.00	0.00	0.00
Knock twice	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.00	0.00	0.00
Draw a circle	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.01	0.00
Draw a Z	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.98	0.00
Touch nose with index	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99

Figure 6.5: 5-fold CV results for Model 3

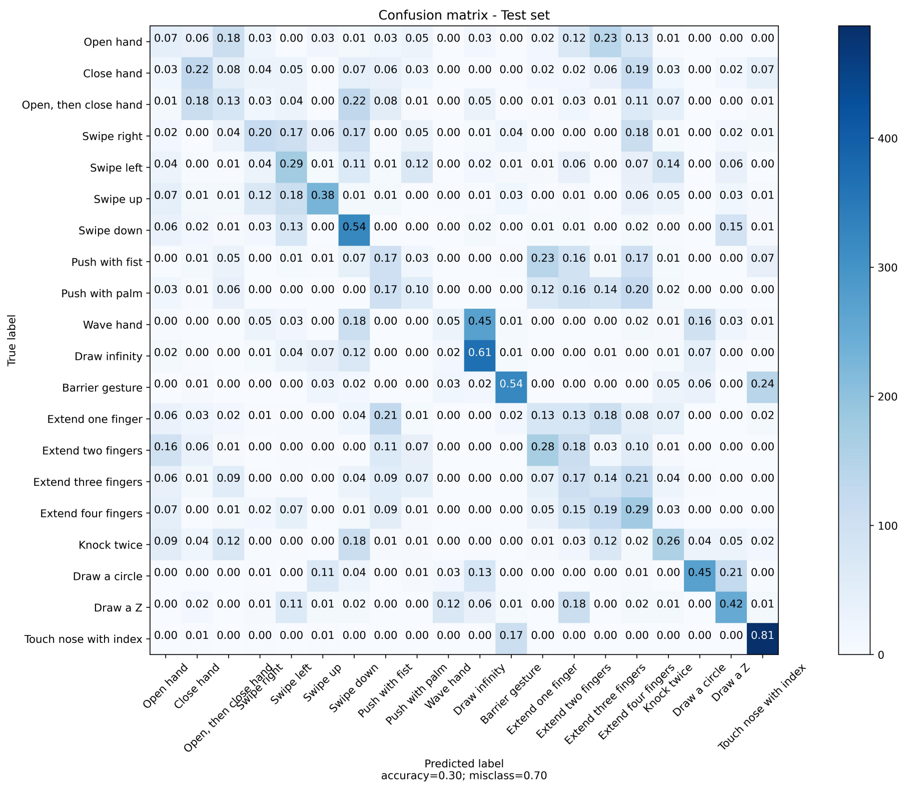
- The lower classification results are consistent through the different models, given since they happen in the same classes (13-16). This finds an explanation in the nature of such gestures, in fact those are very similar between them (extend one, two, three fingers etc.) hence it is more likely that they get confused between each other.

One final piece of results comes from the second attempt for the model validation which, as it was presented earlier in the chapter, consists in manually dividing training and test set in order to exclude totally or partially a set of participants, in order to truly understand the classification capabilities of the model.
In this scenario, two main results are relevant. The first one derives from the training done excluding totally the test set's participants, hence the model never saw the test set data points before the testing phase. In this case, the results were quite poor as it can be observed (Table 6.7, Figure 6.6):

Classification Accuracy	**Precision**	**Recall**
0.2992	0.3029	0.2991

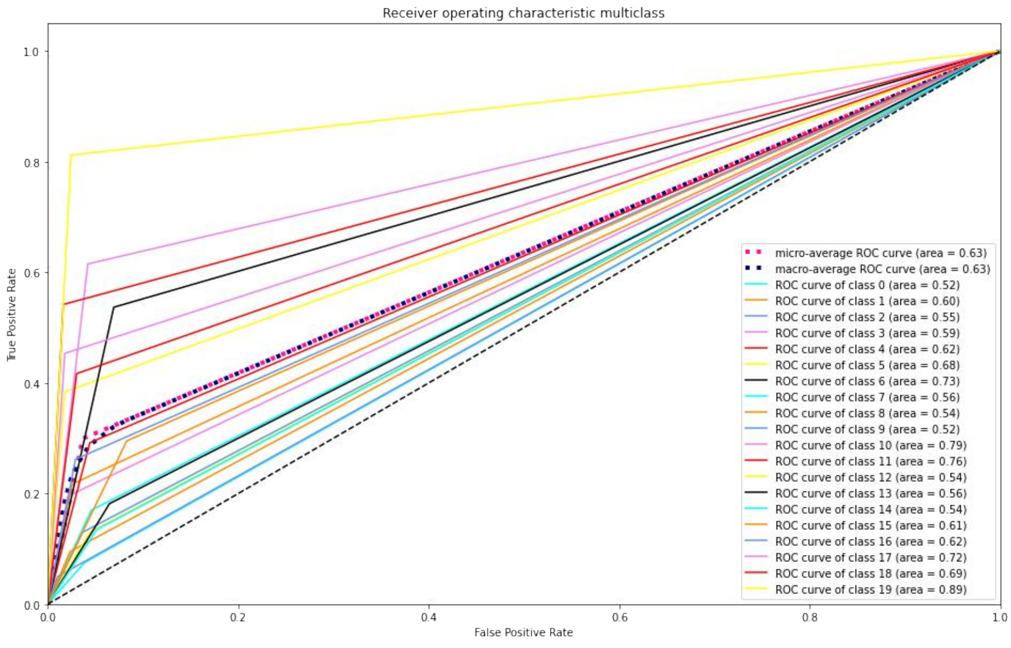
Table 6.7: Results of Model 3 training with total splitting

A weak 30% of accuracy was achieved in this scenario, making it unlikely to be sufficient for any purpose. Such a poor accuracy could find an explanation in the many variables that may make it difficult for the model to correctly identify the type of gesture. Differences in hand size, gesture articulation, hand-radar distance etc. are all elements that differentiate how people perform the movements, hence making the task harder.
However, after this outcome an idea came up. Starting from the fact that many devices in the HCI field (AI speakers, cameras, etc.) require an initial calibration to work properly with the user, the fact of adding part of the test set to the training set could be actually considered as a sort of calibration. In this way, the model learns features regarding gestures, but it is also a bit more used to the way that test users compute such gestures, a trade-off worth trying. By adding around 10%



	Open hand	Close hand	Open, then close hand	Swipe right	Swipe left	Swipe up	Swipe down	Push with fist	Push with palm	Wave hand	Draw infinity	Barrier gesture	Extend one finger	Extend two fingers	Extend three fingers	Extend four fingers	Knock twice	Draw a circle	Draw a Z	Touch nose with index
Open hand	0.07	0.06	0.18	0.03	0.00	0.03	0.01	0.03	0.05	0.00	0.03	0.00	0.02	0.12	0.23	0.13	0.01	0.00	0.00	0.00
Close hand	0.03	0.22	0.08	0.04	0.05	0.00	0.07	0.06	0.03	0.00	0.00	0.00	0.02	0.02	0.06	0.19	0.03	0.00	0.02	0.07
Open, then close hand	0.01	0.18	0.13	0.03	0.04	0.00	0.22	0.08	0.01	0.00	0.05	0.00	0.01	0.03	0.01	0.11	0.07	0.00	0.00	0.01
Swipe right	0.02	0.00	0.04	0.20	0.17	0.06	0.17	0.00	0.05	0.00	0.01	0.04	0.00	0.00	0.00	0.18	0.01	0.00	0.02	0.01
Swipe left	0.04	0.00	0.01	0.04	0.29	0.01	0.11	0.01	0.12	0.00	0.02	0.01	0.01	0.06	0.00	0.07	0.14	0.00	0.06	0.00
Swipe up	0.07	0.01	0.01	0.12	0.18	0.38	0.01	0.01	0.00	0.00	0.01	0.03	0.00	0.01	0.00	0.06	0.05	0.00	0.03	0.01
Swipe down	0.06	0.02	0.01	0.03	0.13	0.00	0.54	0.00	0.00	0.00	0.02	0.00	0.01	0.01	0.00	0.02	0.00	0.00	0.15	0.01
Push with fist	0.00	0.01	0.05	0.00	0.01	0.01	0.07	0.17	0.03	0.00	0.00	0.00	0.23	0.16	0.01	0.17	0.01	0.00	0.00	0.07
Push with palm	0.03	0.01	0.06	0.00	0.00	0.00	0.00	0.17	0.10	0.00	0.00	0.00	0.12	0.16	0.14	0.20	0.02	0.00	0.00	0.00
Wave hand	0.00	0.00	0.00	0.05	0.03	0.00	0.18	0.00	0.00	0.05	0.45	0.01	0.00	0.00	0.00	0.02	0.01	0.16	0.03	0.01
Draw infinity	0.02	0.00	0.00	0.01	0.04	0.07	0.12	0.00	0.00	0.02	0.61	0.01	0.00	0.00	0.01	0.00	0.01	0.07	0.00	0.00
Barrier gesture	0.00	0.01	0.00	0.00	0.00	0.03	0.02	0.00	0.00	0.03	0.02	0.54	0.00	0.00	0.00	0.00	0.05	0.06	0.00	0.24
Extend one finger	0.06	0.03	0.02	0.01	0.00	0.00	0.04	0.21	0.01	0.00	0.00	0.02	0.13	0.13	0.18	0.08	0.07	0.00	0.00	0.02
Extend two fingers	0.16	0.06	0.01	0.00	0.00	0.00	0.00	0.11	0.07	0.00	0.00	0.00	0.28	0.18	0.03	0.10	0.01	0.00	0.00	0.00
Extend three fingers	0.06	0.01	0.09	0.00	0.00	0.00	0.04	0.09	0.07	0.00	0.00	0.00	0.07	0.17	0.14	0.21	0.04	0.00	0.00	0.00
Extend four fingers	0.07	0.00	0.01	0.02	0.07	0.00	0.01	0.09	0.01	0.00	0.00	0.00	0.05	0.15	0.19	0.29	0.03	0.00	0.00	0.00
Knock twice	0.09	0.04	0.12	0.00	0.00	0.00	0.18	0.01	0.01	0.00	0.00	0.00	0.01	0.03	0.12	0.02	0.26	0.04	0.05	0.02
Draw a circle	0.00	0.00	0.00	0.01	0.00	0.11	0.04	0.00	0.01	0.03	0.13	0.00	0.00	0.00	0.00	0.01	0.00	0.45	0.21	0.00
Draw a Z	0.00	0.02	0.00	0.01	0.11	0.01	0.02	0.00	0.00	0.12	0.06	0.01	0.00	0.18	0.00	0.02	0.01	0.00	0.42	0.01
Touch nose with index	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.17	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.81

(a) Confusion matrix



(b) ROC multiclass

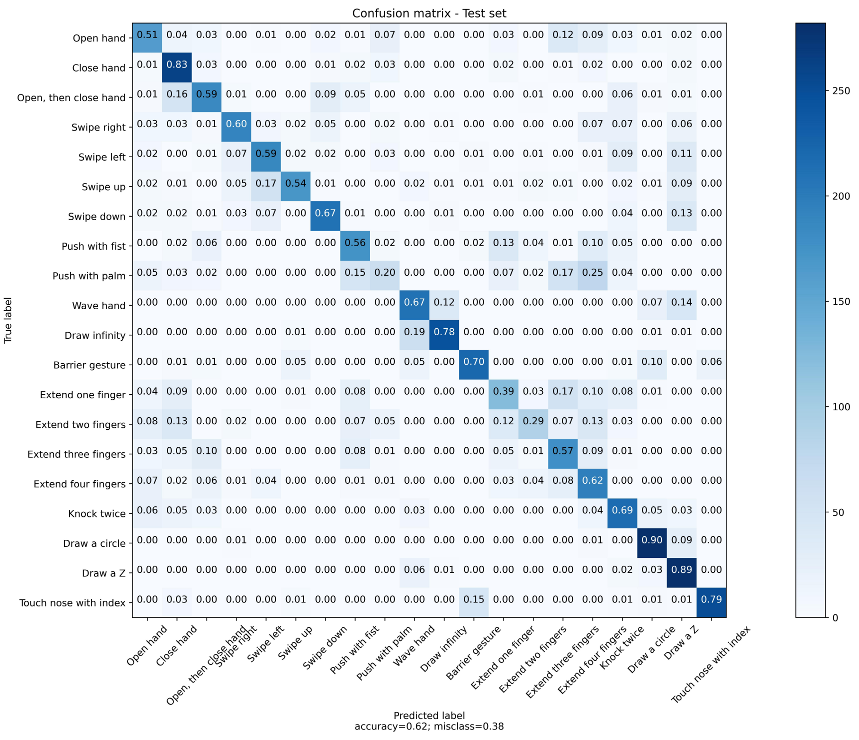
Figure 6.6: Results on Model 3 with total splitting

of the test dataset to the training part, the classification performances increased significantly (Table 6.8, Figure 6.7).
By doing this test the model was assessed quite deeply, in fact the scenarios in which the classifier has to recognize a gesture from a person that has never "seen" before are the most challenging ones, yet the closest to reality.

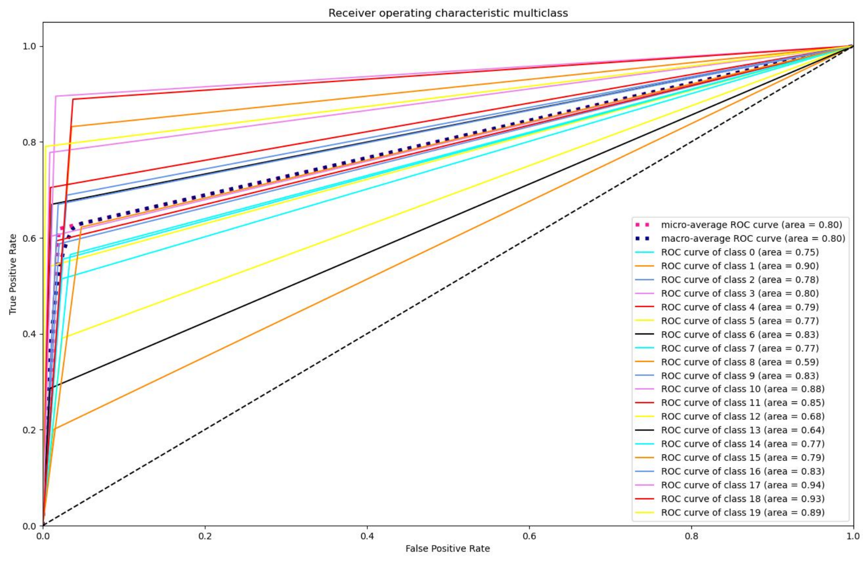
Classification Accuracy	**Precision**	**Recall**
0.6187	0.6325	0.6187

Table 6.8: Results of Model 3 training with partial splitting

In conclusion to the results exposed in this chapter, it is worth mentioning that different models responded in different ways to the changes brought each time to hyperparameters, dataset, etc. which means that they were actually learning some peculiar features of the gestures. The main gestures that suffered the most in terms of classification accuracy were the ones very similar between them, such as extending fingers, pushing fist/closing hand, pushing palm/opening hand etc. Most likely due to the low signal resolution, these gestures resulted in being the most confused ones across the different models.



(a) Confusion matrix



(b) ROC multiclass

Figure 6.7: Results on Model 3 with partial splitting

7 CONCLUSION

In this thesis the main goal was to explore the use of deep learning techniques to recognize hand gestures captured by a radar sensor.

First, a general overview of the problem was given, together with the motivations of the main elements of this thesis that led to do the choices outlined in this manuscript. Then, a targeted literature review was conducted in order to highlight the most relatable projects in the scientific community regarding hand gestures recognition, both in the case of radar acquisition systems and not, with machine learning-based classification techniques and more classical approaches. An analysis of the equipment at disposal was provided too.
The dataset formation for this project was then outlined, starting from the basics of the acquisition of the data and arriving to the post-processed signal ready for the model.
The main models developed during the experiment were exhibited, together with the choices taken regarding the structures and the parameters. These were followed

up by the overview on training and validation phase, where all methods and results accomplished were reported.
In the previous chapters, research questions posed in Subsection 1.2.2 found an answer. From the data acquisition and processing, the capability of a radar sensor in acquiring hand movements was tested. The signal required some processing to clear it from noise and interference, but in general it could be said that the Walabot was successful in capturing hand movements (RQ1).
Methods and results exposed in Ch. 5, 6 showed how deep learning has great potential in hand gesture recognition. The classification accuracy is not extremely high, but there are many things that can be improved. Deep learning is definetely suitable for this classification task (RQ2), provided that the working conditions are sufficient for the correct implementation of the model.
Finally, a specific test was performed on the model to answer to RQ3. In the validation of Model 3, results showed how there is potential for the network to generalize on new, unseen users. However, as the framework is right now, the results are not quite acceptable hence more work should be done of this.
Some limits were encountered in the process, due to the lack of resources. Many things could be done to improve and extend the work done for this thesis.
First, the dataset should be expanded as much as possible. A total of 100/150 participants would already be a significant load of information for a deep learning model to be quite strong in the classification.
Second, an amount of data such like this would require a more complex model, hence more computational power to perform the training. Moreover, linked to this, the access to more power would allow to run some hyperparameters tuning processes, since the manual search can't be efficient from a certain point on.
However, the results show that deep learning applied to hand gesture recognition tasks can be promising if tackled from the right way, and there is much more to explore in this context.

With more time and resources, many more approaches could be tried. As stated before, the increment of the dataset size would already help much in the strength of the model. Moreover, other deep architectures could be tried, in order to exploit the raw data coming from the sensor in a different way with respect to images. Another aspect that could be further analyzed would be that of optimizing the acquisition and the signal processing in order to make the classification in a real-time setting: for the moment, in fact, the time needed for the transformation of the signal plus the inference on the network is way above the threshold for an application to be considered on-line. Working on these aspects could already sensibly increase the performances of the system, possibly leading to a real application that was not feasible due to the limitations of this work.

References

[1] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, and T. Carey, *Human-computer interaction.* Addison-Wesley Longman Ltd., 1994.

[2] T. Vuletic, A. Duffy, L. Hay, C. McTeague, G. Campbell, and M. Grealy, "Systematic literature review of hand gestures used in human computer interaction interfaces," *International Journal of Human-Computer Studies*, vol. 129, pp. 74–94, 2019.

[3] S. Villarreal-Narvaez, J. Vanderdonckt, R.-D. Vatavu, and J. O. Wobbrock, "A systematic review of gesture elicitation studies: What can we learn from 216 studies?" in *Proceedings of the 2020 ACM Designing Interactive Systems Conference*, 2020, pp. 855–872.

[4] S. Ahmed, D. Wang, J. Park, and S. H. Cho, "Uwb-gestures, a public dataset of dynamic hand gestures acquired using impulse radar sensors," *Scientific Data*, vol. 8, no. 1, p. 102, Apr 2021. [Online]. Available: https://doi.org/10.1038/s41597-021-00876-0

[5] K. Shailaja, B. Seetharamulu, and M. Jabbar, "Machine learning in healthcare: A review," in *2018 Second international conference on electronics, communication and aerospace technology (ICECA).* IEEE, 2018, pp. 910–914.

[6] M. F. Dixon, I. Halperin, and P. Bilokon, *Machine learning in Finance.* Springer, 2020, vol. 1406.

[7] K. G. Liakos, P. Busato, D. Moshou, S. Pearson, and D. Bochtis, "Machine learning in agriculture: A review," *Sensors*, vol. 18, no. 8, p. 2674, 2018.

[8] M. K. Chowdary, T. N. Nguyen, and D. J. Hemanth, “Deep learning-based facial emotion recognition for human–computer interaction applications,” *Neural Computing and Applications*, pp. 1–18, 2021.

[9] S. M. S. A. Abdullah, S. Y. A. Ameen, M. A. Sadeeq, and S. Zeebaree, “Multimodal emotion recognition using deep learning,” *Journal of Applied Science and Technology Trends*, vol. 2, no. 02, pp. 52–58, 2021.

[10] S. Ahmed, K. D. Kallu, S. Ahmed, and S. H. Cho, “Hand gestures recognition using radar sensors for human-computer-interaction: A review,” *Remote Sensing*, vol. 13, no. 3, 2021. [Online]. Available: https://www.mdpi.com/2072-4292/13/3/527

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* MIT press, 2016.

[12] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, “Deep learning applications and challenges in big data analytics,” *Journal of big data*, vol. 2, no. 1, pp. 1–21, 2015.

[13] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[14] C. Gu, J. Wang, and J. Lien, “Motion sensing using radar: Gesture interaction and beyond,” *IEEE Microwave Magazine*, vol. 20, no. 8, pp. 44–57, 2019.

[15] A.-I. Siean, C. Pamparău, and R.-D. Vatavu, “Scenario-based exploration of integrating radar sensing into everyday objects for free-hand television control,” in *ACM International Conference on Interactive Media Experiences*, 2022, pp. 357–362.

[16] H.-S. Yeo and A. Quigley, “Radar sensing in human-computer interaction,” *interactions*, vol. 25, pp. 70–73, 12 2017.

[17] B. Ionescu, V. Suse, C. Gadea, B. Solomon, D. Ionescu, S. Islam, and M. Cordea, “Using a nir camera for car gesture control,” *IEEE Latin America Transactions*, vol. 12, no. 3, pp. 520–523, 2014.

[18] F. Erden and A. E. Cx0327;etin, “Hand gesture based remote control system using infrared sensors and a camera,” *IEEE Transactions on Consumer Electronics*, vol. 60, no. 4, pp. 675–680, 2014.

[19] Y. Sang, L. Shi, and Y. Liu, “Micro hand gesture recognition system using ultrasonic active sensing,” *IEEE Access*, vol. 6, pp. 49 339–49 347, 2018.

[20] A. Al-Hourani, R. Evans, P. Farrell, B. Moran, M. Martorella, K. Sithamparanathan, S. Skafidas, and U. Parampalli, *Millimeter-wave Integrated Radar Systems and Techniques*, 12 2017.

[21] Z. Ren, J. Meng, and J. Yuan, “Depth camera based hand gesture recognition and its applications in human-computer-interaction,” in *2011 8th International Conference on Information, Communications Signal Processing*, 2011, pp. 1–5.

[22] Z. Li and R. Jarvis, “Real time hand gesture recognition using a range camera,” in *Australasian Conference on Robotics and Automation*, 2009, pp. 21–27.

[23] A. Licsár and T. Szirányi, “Hand gesture recognition in camera-projector system,” in *International Workshop on Computer Vision in Human-Computer Interaction.* Springer, 2004, pp. 83–93.

[24] P. Parvathy, K. Subramaniam, G. K. D. Prasanna Venkatesan, P. Karthikaikumar, J. Varghese, and T. Jayasankar, “Retracted article: Development of hand gesture recognition system using machine learning,” *Journal of Ambient Intelligence and Humanized Comput-*

ing, vol. 12, no. 6, pp. 6793–6800, Jun 2021. [Online]. Available: https://doi.org/10.1007/s12652-020-02314-2

[25] L. Yun, Z. Lifeng, and Z. Shujun, "A hand gesture recognition method based on multi-feature fusion and template matching," *Procedia Engineering*, vol. 29, pp. 1678–1684, 2012.

[26] A. Corradini and H.-M. Gross, "Camera-based gesture recognition for robot control," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, vol. 4. IEEE, 2000, pp. 133–138.

[27] P. Trigueiros, F. Ribeiro, and L. P. Reis, "A comparison of machine learning algorithms applied to hand gesture recognition," in *7th Iberian Conference on Information Systems and Technologies (CISTI 2012)*, 2012, pp. 1–6.

[28] J. Shukla and A. Dwivedi, "A method for hand gesture recognition," in *2014 Fourth International Conference on Communication Systems and Network Technologies*, 2014, pp. 919–923.

[29] S. P. More and A. Sattar, "Hand gesture recognition system using image processing," in *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, 2016, pp. 671–675.

[30] A. Sharma, A. Mittal, S. Singh, and V. Awatramani, "Hand gesture recognition using image processing and feature extraction techniques," *Procedia Computer Science*, vol. 173, pp. 181–190, 2020, international Conference on Smart Sustainable Intelligent Computing and Applications under ICITETM2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S187705092031526X

[31] H.-I. Lin, M.-H. Hsu, and W.-K. Chen, "Human hand gesture recognition using a convolution neural network," in *2014 IEEE International Conference on Automation Science and Engineering (CASE)*, 2014, pp. 1038–1043.

[32] P. Molchanov, S. Gupta, K. Kim, and K. Pulli, "Multi-sensor system for driver's hand-gesture recognition," in *2015 11th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*, vol. 1, 2015, pp. 1–8.

[33] U. Allard, C. 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. corr 2018," *arXiv preprint arXiv:1801.07756*.

[34] S. Bhushan, M. Alshehri, I. Keshta, A. K. Chakraverti, J. Rajpurohit, and A. Abugabah, "An experimental analysis of various machine learning algorithms for hand gesture recognition," *Electronics*, vol. 11, no. 6, 2022. [Online]. Available: https://www.mdpi.com/2079-9292/11/6/968

[35] W. Zhang, J. Wang, and F. Lan, "Dynamic hand gesture recognition based on short-term sampling neural networks," *IEEE/CAA Journal of Automatica Sinica*, vol. 8, no. 1, pp. 110–120, 2021.

[36] Y. Zhao and L. Wang, "The application of convolution neural networks in sign language recognition," 11 2018, pp. 269–272.

[37] S. Tam, M. Boukadoum, A. Campeau-Lecours, and B. Gosselin, "A fully embedded adaptive real-time hand gesture classifier leveraging hd-semg and deep learning," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 14, no. 2, pp. 232–243, 2020.

[38] O. H. Y. Lam, R. Kulke, M. Hagelen, and G. Möllenbeck, “Classification of moving targets using mirco-doppler radar,” in *2016 17th International Radar Symposium (IRS)*, 2016, pp. 1–6.

[39] C. Li, Z. Peng, T.-Y. Huang, T. Fan, F.-K. Wang, T.-S. Horng, J.-M. Muñoz-Ferreras, R. Gómez-García, L. Ran, and J. Lin, “A review on recent progress of portable short-range noncontact microwave radar systems,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 65, no. 5, pp. 1692–1706, 2017.

[40] A. Pramudita, L. Lukas, and E. Ewer, “Time and frequency domain feature extraction method of doppler radar for hand gesture based human to machine interface,” *Progress In Electromagnetics Research C*, vol. 98, pp. 83–96, 01 2020.

[41] V. Imaging, “Walabot - technical brief,” 04 2016.

[42] “Vayyar care,” https://vayyar.com/care/b2b/overview/, accessed: 2022-08-07.

[43] “X4m02 datasheet,” http://laonuri.techyneeti.com/wp-content/uploads/2019/02/X4M02_DATASHEET.pdf, accessed: 2022-08-07.

[44] “X4m200 datasheet,” http://laonuri.techyneeti.com/wp-content/uploads/2019/02/X4M200_DATASHEET.pdf, accessed: 2022-08-07.

[45] “X4m300 datasheet,” http://laonuri.techyneeti.com/wp-content/uploads/2019/02/X4M02_DATASHEET.pdf, accessed: 2022-08-07.

[46] “Ipm 170 - ipm 365,” https://www.innosent.de/radarsensoren/ipm-series-innosent/ipm-170-ipm-365/, accessed: 2022-08-07.

[47] “Ipm 165,” http://nic.vajn.icu/PDF/sensors/radar/CDM324-doppler/InnoSenT/InnoSenT_application_note_III_11_06.pdf, accessed: 2022-08-07.

[48] "Infineon bgt60ltr11aip," https : / / www . infineon . com / dgdl / Infineon-BGT60LTR11AIP _ ProductBrief-ProductBrief-v01 _ 00-EN . pdf ? fileId=5546d4627564a75001756a38b2e74bc9, accessed: 2022-08-07.

[49] "Infineon bgt60ltr11," https : / / www . infineon . com / cms / en / product / evaluation-boards/demo-distance2gol/, accessed: 2022-08-07.

[50] "Radarbook 2," https://inras.at/en/radarbook2/, accessed: 2022-08-07.

[51] "Texas instrument radar," https://www.ti.com/tool/IWR6843ISK, accessed: 2022-08-07.

[52] "Evalkit sirad easy r4," https://siliconradar.com/evalkits/, accessed: 2022-08-07.

[53] "imotion 1, 2, 3," https://sites.google.com/site/imotionradar/imotion, accessed: 2022-08-07.

[54] L. Zhang, J. Xiong, H. Zhao, H. Hong, X. Zhu, and C. Li, "Sleep stages classification by cw doppler radar using bagged trees algorithm," in *2017 IEEE Radar Conference (RadarConf)*, 2017, pp. 0788–0791.

[55] G. Li, S. Zhang, F. Fioranelli, and H. Griffiths, "Effect of sparse-aware time-frequency analysis on dynamic hand gesture classification with radar micro-doppler signatures," *IET Radar, Sonar Navigation*, vol. 12, 04 2018.

[56] Z. Zhang, Z. Tian, and M. Zhou, "Latern: Dynamic continuous hand gesture recognition using fmcw radar sensor," *IEEE Sensors Journal*, vol. 18, no. 8, pp. 3278–3289, 2018.

[57] D. Miao, H. Zhao, H. Hong, X. Zhu, and C. Li, "Doppler radar-based human breathing patterns classification using support vector machine," in *2017 IEEE Radar Conference (RadarConf)*, 2017, pp. 0456–0459.

[58] Y. Kim and H. Ling, “Human activity classification based on micro-doppler signatures using a support vector machine,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 47, no. 5, pp. 1328–1337, 2009.

[59] Y. Kim and B. Toomajian, “Hand gesture recognition using micro-doppler signatures with convolutional neural network,” *IEEE Access*, vol. 4, pp. 7125–7130, 2016.

[60] C. Zheng, T. Hu, S. Qiao, Y. Sun, J. Huangfu, and L. Ran, “Doppler bio-signal detection based time-domain hand gesture recognition,” in *2013 IEEE MTT-S International Microwave Workshop Series on RF and Wireless Technologies for Biomedical and Healthcare Applications (IMWS-BIO)*, 2013, pp. 3–3.

[61] Q. Wan, Y. Li, C. Li, and R. Pal, “Gesture recognition for smart home applications using portable radar sensors,” in *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2014, pp. 6414–6417.

[62] T. Fan, C. Ma, Z. Gu, Q. Lv, J. Chen, D. Ye, J. Huangfu, Y. Sun, C. Li, and L. Ran, “Wireless hand gesture recognition based on continuous-wave doppler radar sensors,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 64, no. 11, pp. 4012–4020, 2016.

[63] X. Gao, J. Xu, A. Rahman, E. Yavari, A. Lee, V. Lubecke, and O. Boric-Lubecke, “Barcode based hand gesture classification using ac coupled quadrature doppler radar,” in *2016 IEEE MTT-S International Microwave Symposium (IMS)*, 2016, pp. 1–4.

[64] S. Zhang, G. Li, M. Ritchie, F. Fioranelli, and H. Griffiths, “Dynamic hand gesture classification based on radar micro-doppler signatures,” in *2016 CIE International Conference on Radar (RADAR)*, 2016, pp. 1–4.

[65] S.-T. Huang and C.-H. Tseng, “Hand-gesture sensing doppler radar with metamaterial-based leaky-wave antennas,” in *2017 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility (ICMIM)*, 2017, pp. 49–52.

[66] G. Li, R. Zhang, M. Ritchie, and H. Griffiths, “Sparsity-driven micro-doppler feature extraction for dynamic hand gesture recognition,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 54, no. 2, pp. 655–665, 2018.

[67] T. Sakamoto, X. Gao, E. Yavari, A. Rahman, O. Boric-Lubecke, and V. M. Lubecke, “Radar-based hand gesture recognition using i-q echo plot and convolutional neural network,” in *2017 IEEE Conference on Antenna Measurements Applications (CAMA)*, 2017, pp. 393–395.

[68] ——, “Hand gesture recognition using a radar echo i–q plot and a convolutional neural network,” *IEEE sensors letters*, vol. 2, no. 3, pp. 1–4, 2018.

[69] M. G. Amin, Z. Zeng, and T. Shan, “Hand gesture recognition based on radar micro-doppler signature envelopes,” in *2019 IEEE Radar Conference (RadarConf)*, 2019, pp. 1–6.

[70] S. Skaria, A. Al-Hourani, M. Lech, and R. Evans, “Hand-gesture recognition using two-antenna doppler radar with deep convolutional neural networks,” *IEEE Sensors Journal*, vol. PP, 01 2019.

[71] E. Klinefelter and J. A. Nanzer, “Interferometric radar for spatially-persistent gesture recognition in human-computer interaction,” in *2019 IEEE Radar Conference (RadarConf)*, 2019, pp. 1–5.

[72] E. Miller, Z. Li, H. Mentis, A. Park, T. Zhu, and N. Banerjee, “Radsense: Enabling one hand and no hands interaction for sterile manipulation of medical images using doppler radar,” *Smart Health*, vol. 15, p. 100089,

2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352648319300534

[73] M. Yu, N. Kim, Y. Jung, and S. Lee, "A frame detection method for real-time hand gesture recognition systems using cw-radar," *Sensors*, vol. 20, no. 8, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/8/2321

[74] Z. Wang, G. Li, and L. Yang, "Dynamic hand gesture recognition based on micro-doppler radar signatures using hidden gaussmarkov models," *IEEE Geoscience and Remote Sensing Letters*, vol. 18, no. 2, pp. 291–295, 2021.

[75] A. Sluÿters, S. Lambot, and J. Vanderdonckt, "Hand gesture recognition for an off-the-shelf radar by electromagnetic modeling and inversion," in *27th International Conference on Intelligent User Interfaces*, ser. IUI '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 506522. [Online]. Available: https://doi.org/10.1145/3490099.3511107

[76] "Google soli," https://atap.google.com/soli/, accessed: 2022-08-01.

[77] J. Lien, N. Gillian, M. E. Karagozler, P. Amihood, C. Schwesig, E. Olson, H. Raja, and I. Poupyrev, "Soli: Ubiquitous gesture sensing with millimeter wave radar," *ACM Trans. Graph.*, vol. 35, no. 4, jul 2016. [Online]. Available: https://doi.org/10.1145/2897824.2925953

[78] R. Min, X. Wang, J. Zou, J. Gao, L. Wang, and Z. Cao, "Early gesture recognition with reliable accuracy based on high-resolution iot radar sensors," *IEEE Internet of Things Journal*, vol. 8, no. 20, pp. 15 396–15 406, 2021.

[79] S. Wang, J. Song, J. Lien, I. Poupyrev, and O. Hilliges, "Interacting with soli: Exploring fine-grained dynamic gesture recognition in the radio-frequency spectrum," in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, ser. UIST '16. New York, NY,

USA: Association for Computing Machinery, 2016, p. 851860. [Online]. Available: https://doi.org/10.1145/2984511.2984565

[80] H.-S. Yeo, G. Flamich, P. Schrempf, D. Harris-Birtill, and A. Quigley, "Radarcat: Radar categorization for input interaction," in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, ser. UIST '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 833841. [Online]. Available: https://doi.org/10.1145/2984511.2984515

[81] G. Agresti and S. Milani, "Material identification using rf sensors and convolutional neural networks," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 3662–3666.

[82] D. Avrahami, M. Patel, Y. Yamaura, and S. Kratz, "Below the surface: Unobtrusive activity recognition for work surfaces using rf-radar sensing," in *23rd International Conference on Intelligent User Interfaces*, ser. IUI '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 439451. [Online]. Available: https://doi.org/10.1145/3172944.3172962

[83] S. Zhu, J. Xu, H. Guo, Q. Liu, S. Wu, and H. Wang, "Indoor human activity recognition based on ambient radar with signal processing and machine learning," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[84] B. Zhang, L. Zhang, M. Wu, and Y. Wang, "Dynamic gesture recognition based on rf sensor and ae-lstm neural network," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[85] T.-K. Kim and R. Cipolla, "Canonical correlation analysis of video volume tensors for action categorization and detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 8, pp. 1415–1428, 2008.

[86] Y. Kong, B. Satarboroujeni, and Y. Fu, "Learning hierarchical 3d kernel descriptors for rgb-d action recognition," *Computer Vision and Image Understanding*, vol. 144, pp. 14–23, 2016.

[87] Y. Zhang, C. Cao, J. Cheng, and H. Lu, "Egogesture: A new dataset and benchmark for egocentric hand gesture recognition," *IEEE Transactions on Multimedia*, vol. 20, no. 5, pp. 1038–1050, 2018.

[88] J. Duan, J. Wan, S. Zhou, X. Guo, and S. Z. Li, "A unified framework for multi-modal isolated gesture recognition," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 14, no. 1s, pp. 1–16, 2018.

[89] "Walabot," https://site.walabot.com/makers, accessed: 2022-08-01.

[90] S. S. Rautaray and A. Agrawal, "Vision based hand gesture recognition for human computer interaction: a survey," *Artificial intelligence review*, vol. 43, no. 1, pp. 1–54, 2015.

[91] X. Zhang, X. Chen, W.-h. Wang, J.-h. Yang, V. Lantz, and K.-q. Wang, "Hand gesture recognition and virtual game control based on 3d accelerometer and emg sensors," in *Proceedings of the 14th international conference on Intelligent user interfaces*, 2009, pp. 401–406.

[92] S. Mitra and T. Acharya, "Gesture recognition: A survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 3, pp. 311–324, 2007.

[93] R. Aigner, D. Wigdor, H. Benko, M. Haller, D. Lindbauer, A. Ion, S. Zhao, and J. Koh, "Understanding mid-air hand gestures: A study of human preferences in usage of gesture types for hci," *Microsoft Research TechReport MSR-TR-2012-111*, vol. 2, p. 30, 2012.

[94] A. Stamoulakatos, J. Cardona, C. Michie, I. Andonovic, P. Lazaridis, X. Bellekens, R. Atkinson, M. Hossain, and C. Tachtatzis, “A comparison of the performance of 2d and 3d convolutional neural networks for subsea survey video classification,” 08 2021.

[95] M. Heideman, D. Johnson, and C. Burrus, “Gauss and the history of the fast fourier transform,” *IEEE ASSP Magazine*, vol. 1, no. 4, pp. 14–21, 1984.

[96] S. Lambot and F. André, “Full-wave modeling of near-field radar data for planar layered media reconstruction,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 52, no. 5, pp. 2295–2303, 2014.

[97] S. Lambot, E. Slob, I. van den Bosch, B. Stockbroeckx, and M. Vanclooster, “Modeling of ground-penetrating radar for accurate characterization of subsurface electric properties,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 42, no. 11, pp. 2555–2568, 2004.

[98] C. "Szegedy, W. "Liu, Y. "Jia, P. "Sermanet, S. "Reed, D. "Anguelov, D. "Erhan, V. "Vanhoucke, and A. "Rabinovich, “Going deeper with convolutions,” 2015.

[99] N. T. Attygalle, L. A. Leiva, M. Kljun, C. Sandor, A. Plopski, H. Kato, and K. Čopič Pucihar, “No interface, no problem: gesture recognition on physical objects using radar sensing,” *Sensors*, vol. 21, no. 17, p. 5771, 2021.

[100] L. O. Fhager, S. Heunisch, H. Dahlberg, A. Evertsson, and L.-E. Wernersson, “Pulsed millimeter wave radar for hand gesture sensing and classification,” *IEEE Sensors Letters*, vol. 3, no. 12, pp. 1–4, 2019.

[101] S. Ahmed and S. H. Cho, “Hand gesture recognition using an ir-uwb radar with an inception module-based classifier,” *Sensors*, vol. 20, no. 2, p. 564, 2020.

[102] E. Miller, Z. Li, H. Mentis, A. Park, T. Zhu, and N. Banerjee, “Radsense: Enabling one hand and no hands interaction for sterile manipulation of medical images using doppler radar,” *Smart Health*, vol. 15, p. 100089, 2020.

[103] N. Saunshi, A. Gupta, and W. Hu, “A representation learning perspective on the importance of train-validation splitting in meta-learning,” in *International Conference on Machine Learning*. PMLR, 2021, pp. 9333–9343.

[104] A. Rácz, D. Bajusz, and K. Héberger, “Effect of dataset size and train/test split ratios in qsar/qspr multiclass classification,” *Molecules*, vol. 26, no. 4, p. 1111, 2021.

[105] Z. Reitermanova *et al.*, “Data splitting,” in *WDS*, vol. 10. Matfyzpress Prague, 2010, pp. 31–36.