University of Padova

Department of Information Engineering

Master Thesis in ICT for Internet and Multimedia Engineering

Person Identification from Millimeter-Wave Radar μ-Doppler Signature

Supervisor
Prof. Michele Rossi
University of Padova

Master Candidate
Jacopo Pegoraro

Academic Year
2018-2019

Abstract

Radio frequency indoor sensing technologies are becoming a widely studied topic in recent years due to their large applicability in security and monitoring systems. In this work the problem of person identification from the reflected signal of a radar device is addressed, exploiting the Doppler effect caused by small motions that happen as the subject moves. This phenomenon, called micro-Doppler signature, is specific of each person depending on the particular reflective properties of the individual and can be used as a discriminator of the person identity among a group. To extract and process the complex features of the signal that carry the micro-Doppler information a novel processing algorithm is proposed, comprising adaptive denoising, density based clustering and classification. The classifier structure makes use of the recent developments in deep learning models to perform automatic feature extraction. The evaluation of the proposed approach is carried out on a publicly available dataset of indoor mm-wave radar measurements, resulting in an improvement over the state of the art in both accuracy of the classification and adaptability of the designed processing pipeline. An accuracy ranging from 84 to 90% was reached among 5 subjects, depending on the level of information available on the noise distribution. The clustering phase, on the other hand, allowed to separate contributions from multiple targets present in the environment enabling multi-person identification, a not yet explored application in the literature. Multi-person identification is demonstrated on own measured data from a mm-wave device and 2 subjects, showing the versatility of the approach.

Sommario

Le tecnologie di *indoor sensing* a radiofrequenza stanno guadagnando crescente interesse negli ultimi anni per la loro applicabilità ai sistemi di sicurezza e di monitoraggio. In questa tesi viene affrontato il problema dell'identificazione di persone dal segnale reflesso di un dispositivo radar, sfruttando l'effetto Doppler causato dai piccoli movimenti che avvengono durante la camminata. Questo fenomeno, chiamato firma micro-Doppler, è specifico di ogni persona: dipende dalle particolari proprietà riflettive dell'individuo e può essere utilizzato per distinguere l'identità del soggetto all'interno di un gruppo. Per estrarre e processare le complesse caratteristiche del segnale, contenenti le informazioni riguardo all'effetto micro-Doppler, viene proposto un nuovo algoritmo che comprende le fasi di rimozione del rumore in maniera adattiva, di clustering e infine di classificazione. Il classificatore utilizza i recenti modelli proposti nel campo del deep learning per effettuare un'estrazione automatica delle caratteristiche del segnale. L'approccio proposto è stato valutato su un dataset pubblicamente disponibile ottenuto con un radar a onde millimetriche, ottenendo dei risultati che migliorano lo stato dell'arte sia come precisione nella classificazione che nell'adattabilità degli algoritmi. L'accuratezza ottenuta varia dall'84 al 90% tra cinque soggetti, in base a quanta informazione è disponibile sulla distribuzione del rumore. La fase di clustering, inoltre, permette di separare il contributo di diversi soggetti presenti contemporaneamente nell'ambiente rendendo così possibile l'identificazione di persone multiple, argomento ancora non trattato in letteratura. La dimostrazione di questo fatto è stata condotta su dei dati misurati con un dispositivo radar a onde millimetriche su due soggetti, mostrando la versatilità dell'approccio proposto.

Contents

Listing of figures

Listing of tables

Listing of acronyms

AE	Auto-Encoder
ANN	Artificial Neural Network
AoA	Angle of Arrival
CNN	Convolutional Neural Network
CW	Continuous Wave
DAE	Denoising Auto-encoder
DAG	Directed Acyclic Graph
DBSCAN	Density-Based Spatial Clustering for Applications with Noise
DCAE	Denoising Convolutional Auto-Encoder
DFT	Discrete Fourier Transform
DHMM	Deep Hidden Markov Model
ELM	Extreme Learning Machine
ELU	Exponential Linear Unit
ERM	Empirical Risk Minimization
FFNN	Feedforward Neural Network
FFT	Fast Fourier Transform
FMCW	Frequency Modulated Continuous Wave
GD	Gradient Descent
GMM	Gaussian Mixture Model
GRU	Gated Recurrent Unit
HDBSCAN	Hierarchical Density-Based Spatial Clustering for Applications with Noise

HMM	Hidden Markov Model
IM	Inception Modules
KDE	Kernel Density Estimation
LFMCW	Linearly Frequency Modulated Continuous Wave
MIMO	Multiple Input Multiple Output
MRD	Maximum Reachability Distance
MSE	Mean Squared Error
MST	Minimum Spanning Tree
μ**-D**	micro-Doppler
PCA	Principal Component Analysis
PDF	Probability Density Function
PRF	Pulse Repetition Frequency
PRI	Pulse Repetition Interval
PSD	Power Spectral Density
RCN	Reservoir Computing Network
R-D	Range-Doppler
ReLU	Rectified Linear Unit
RF	Radio Frequency
RNN	Recurrent Neural Network
RX	Receiver
SGD	Stochastic Gradient Descent
SOM	Self Organizing Map
SP	Salt and Pepper
SPRT	Sequential Probability Ratio Test

STFT	Short-Time Fourier Transform
SVM	Support Vector Machine
TX	Transmitter

1 Introduction

In the field of wireless communications signals transmitted from a source are received in multiple reflected and attenuated copies at the receiver due to the multipath effect. This aspect, usually considered as negative from a communication standpoint, can be exploited in another perspective to gain information about the location and the type of obstacles in the environment from the features of the backscattered signal, as recent research works have shown [4]. The most prominent development in this sense regards the use of radar sensors, a consolidated technology that exploits reflected radio waves to infer the position and velocity of certain targets present in the environment. Recently, however, the technological development of antenna arrays and the extensive utilization of *mm-wave* transceivers, i.e. wireless transmitters and receivers using frequency bands above 10 GHz, has increased the interest in applications exploiting the information available from reflected radio waves, giving birth in the lat few years to the field of *radio sensing*. Research on this topic aims at utilizing fine-grained information on the wireless signal such as phase changes and the so called *micro-Doppler* effect to infer a precise and detailed description of particular aspects of the environment, especially the presence and behavior of humans. Applications of this concepts range from obstacles and pedestrian detection in the automotive field to the monitoring of indoor environments for health care or security purposes. With respect to widely used devices such as wearable sensors, LIDARs* and cameras, radio waves have several benefits: they are relatively low-cost and energy efficient, they do not require good light conditions to work effectively and do not suffer from performance degradation if dust is present in the environment, finally, no device needs to be worn or carried by a human subject. The above advantages come at the cost of processing complexity. Due to the many hardware non-idealitities and to the difficulty of precisely model multipath reflections, which are subject to a countless number of disturbances and unpredictable variations, extracting useful features from the received signal is a difficult task. These problems can however be tackled by using an empirical, measurement-based approach, with the aid of modern *deep* learning algorithms that allow automatic feature extraction from data and that have shown their effectiveness in a large number of fields including signal processing and time-series analysis.

1.1 Person Identification from micro-Doppler Signatures

In this work we focus on person identification from the features of the reflected radar signal by the person itself. In particular, the micro-Doppler (μ-D) effect caused by gait is used to identify a person among a group from their individual specific features. The physical phenomenon of μ-D effect refers to the frequency modulation of the reflected signal caused by small motions of the reflector such as rotations and vibrations [22]. If the target is a walking human, its specific way of walking, i.e. the gait, causes a peculiar μ-D signature in the reflected waveform that under appropriate processing can be used to identify the subject distinguishing it from others. This application of radio sensing has recently gained increasing attention both from the industry and the research community for the huge impact it could have on security and surveillance systems, allowing identification of intruders even in bad visibility conditions. We restrict the discussion to the case of indoor environments and mm-wave radar devices for two reasons: first, indoor environments are more challenging for being more affected by secondary and unwanted reflections due to walls and static objects that cause disturbance to the identification process, second, mm-wave radars enable higher precision and resolution properties, especially when dealing with μ-D signals as shown in Section 2.5.

Laser Imaging Detection and Ranging is a technique that allows a precise mapping of objects and surfaces in an environment using laser impulses.

Here, we analyze the problem of identification from the μ-D signature of gait starting from the raw data obtained by the radar to the final classification of the subject. We propose a signal processing workflow that, using a different approach from those found in the literature, allows the adaptive denoising, clustering and final classification of the μ-D signals. The contributions of this study can be summarized as follows:

1. The development of a time-adaptive denoising algorithm that operates on the radar range-Doppler maps (see Section 2.3) based on a non-parametric statistical estimation technique called Kernel Density Estimation (KDE).
2. The application of density-based clustering algorithms (HDBSCAN) to isolate and manage the reflections from multiple targets, allowing a more accurate and powerful denoising and the possibility of multi-target identification.
3. The implementation of an elaborate classifier architecture based on deep Convolutional Neural Networks (CNN) and Inception Modules (IM) that carries out the task of automatic feature extraction and classification in the considered complex setting.
4. The experimental evaluation of the proposed algorithms on a mm-wave radar dataset and, partially, on independently measured data.

The evaluation of the proposed algorithms will be done on a public dataset of μ-D data from 5 different subjects, obtaining an accuracy as high as 90% in classifying the identity of the specific person among the group of 5, outperforming the classifiers from the literature. We stress, however, that the most important contribution of this work is in the novel approach to the pre-processing of the data: an adaptive denoising phase and clustering enable future developments in the direction of multiple-subject identification, a not yet studied setup.

The thesis is structured as follows. In Chapter 2 a basic overview of radar devices is presented, focusing on the mathematical model of the transmitted and received signals in the specific case of Frequency Modulated Continuous Wave (FMCW) radars. The second part is dedicated to the mathematical description and derivation of the range-Doppler maps and the μ-D spectrograms, the main signals discussed in the thesis. In Chapter 3 the main algorithms used to process the data are thoroughly reviewed, from the density estimation technique used for denoising, KDE, to density based clustering and deep learning. A large part of this chapter focuses on the CNN description as one of the fundamental tools in recent deep learning applications, along with the training algorithms and optimization techniques used. Chapter 4 presents a detailed discussion of the proposed solution, proceeding step by step through the processing phases and design choices. In Chapter 5 an evaluation of the algorithms is carried out, presenting their performance and effectiveness under different metrics. Chapter 6 presents the conclusions drawn from the study and a brief discussion on possible future developments of this work.

1.2 State of the Art

The problem of human identification from radar sensors is a recent topic in the literature that is rapidly gaining momentum. Many works have been published targeting the classification of the subject identity from the μ-D signature of gait using radio signals [5, 3, 6, 7, 8, 9, 10]. Other studies focus on human activity recognition or vital signs detection [11, 12, 13, 14, 15, 16, 17]. The general setup is the recognition of different human movements and activities from the features of the backscattered signal for security or smart-home applications [11, 12, 13, 14], or the tracking of the respiration rate and heartbeat as these cause a detectable movement of the subject's chest [15, 16, 17].

In [11], Principal Component Analysis (PCA) is employed to extract features from μ-D spectrograms and a CNN is used as a classifier for a crowd-counting application. A similar aim is pursued in [12], but here the authors exploit the feature-extraction capabilities of deep learning models and only use a deep CNN. The model is able to learn automatically what features to keep from the data and at the same time perform the classification.

A more complex deep learning architecture is used in [13] to classify μ-D signatures into 12 different classes corresponding to activities such as running, walking, crawling, falling and others. A deep CNN whose weights are initialized by means of an auto-encoder is adopted, using filter concatenation techniques to extract small scale and wide scale information from the input concurrently.

In outdoor environments radars are also used for automotive applications, as in [14] for vulnerable road users classification. The authors developed a CNN classifier to distinguish if detected obstacles in the radar range are other cars, cyclists or pedestrians, guaranteeing an accuracy of 94 % and 87% on two different test sets.

Mm-wave radars provide interesting sensing capabilities also in the health-care field. In [15] a low power FMCW radar at 77 GHz is used to successfully measure the respiration rate and the heartbeat of a person with bare chest in front of the device. Despite the un-realistic setup, the study proves that the small movement of the chest due to the heart beating and respiration is detectable and can be monitored without the person needing to wear any sensor or additional device. [16] and [17] also target the respiration rate estimation, the first using a classical signal processing approach and an ad-hoc radar device, the second using a 1D deep CNN. In [17], in addition, a classification between 4 different respiration patters was performed with considerable accuracy.

The main focus of this thesis is however *gait recognition* and person identification. For this reason a more detailed description of the works on this topic is provided in the following.

In [5] the authors employ for the first time a classifier based on the deep CNN AlexNet[†] to identify a person from the μ-D signature of gait, reaching an accuracy of about 97% with 4 subjects. Differently from our setup, their experiment takes places in an outdoor environment, where correlated noisy reflections from static objects are typically weaker: walls in indoor environments are significantly close to the target of interest in most scenarios and they cause the noise level to increase making the extraction of the useful signal features harder.

Chen *et al.* [8] instead utilized a multi-static radar with three nodes and a pre-trained deep CNN for image recognition in order to classify an armed/non-armed person or identify a person between two subjects in an indoor environment, mostly for security applications. In [6] a complex novel CNN architecture is presented, based on the technique of *adaptive weight learning*, for the identification of a person identity using the μ-D signature of six different movements including walking and running. Running turned out to be the most discriminative action, providing an identification accuracy of 95.21% among 15 subjects.

The above studies focused on simplified experimental scenarios, where the person had to be constrained walking on a straight line in radial direction from the device. In [7] instead a treadmill positioned at different distances from the radar is used,

[†]Krizhevsky Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

and a *ResNet50*[‡] neural network is trained for the classification of 22 subjects. This approach can be useful to simplify the classification by making gait features more evident, but is not realistic and lacks the generality needed for a practical application.

Vandersmissen *et al.* [3] instead used a dataset where 5 subjects are free to walk randomly in two different rooms to train a CNN classifier, attempting to move towards a more robust learning. However, some insight on the data statistical distribution is needed to lower the noise level by thresholding, otherwise the classification performance of the network drops significantly. This may limit the applicability of this approach in situations where the noise level changes from time to time, or where an analysis of the noise distribution cannot be conveniently carried out. The same authors also propose two improvements over this baseline. In [9] a Reservoir Computing Network (RCN) is employed. This kind of model is based on Extreme Learning Machines (ELM) [18] and is particularly suited to problems where training speed is an important factor as they are typically solvable in closed form and only require few seconds/minutes to be trained. The accuracy results are obtained on *raw* data without noise removal but are still inferior to the baseline in [3]. In [10] a generative model based on Deep Hidden Markov Models (DHMM) is utilized to generate hidden states corresponding to the gait features of the target. On top of it, a RNN classifier provides the labels for the classification task receiving as input the generated states. At the cost of requiring a further log-scaling in the pre-processing steps, in addition to noise thresholding, this network provides a significant improvement in terms of accuracy with respect to [3]. The problem of low generality of this kind of processing, however, remains because the denoising is still tailed to the specific dataset and no separation of multiple targets is possible.

In this thesis we enhance the accuracy of the identification and aim at generalizing even more the classifier capabilities. On one hand we do not make use of any dataset-specific parameter or prior knowledge on the noise distribution and we propose a signal processing framework that can work adaptively on any μ-D spectrogram data; on the other hand our algorithm can be easily extended to the case where multiple subjects are walking at the same time in a room, selecting automatically the reflection from each individual and carrying out the identification independently.

[‡]He Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

2 Overview of the Radar Technology

This chapter provides a general overview of the working principles of radar devices. The topic is too wide for an accurate review of the related literature, so the focus will be on the particular technology used in the experimental work of this thesis, that is the *Linear Frequency Modulated Continuous Wave* (LFMCW) radar, treated in Section 2.2. However, an attempt to put this description into perspective is made in Section 2.1, providing the basics of radar target detection and a classification of existing technologies.

2.1 Introduction to Radar Devices

A radar is an electronic system that transmits radio-frequency (RF) electromagnetic waves in the environment, usually towards a preferred direction, and receives the reflected signal from the objects located in the radiation area [1]. The received wave can be processed to obtain interesting properties of the targets, such as the distance from the device (range), the velocity component in radial direction from the radar and the position in space.

The main blocks of a generic radar architecture are shown in Figure 2.1.

1. **Transmitter**: generates the waveform to be transmitted.
2. **Antenna**: propagates the waveform into the transmission medium (i.e. the atmosphere).

3. **Switch**: ensures that the antenna is used only by the transmitter or by the receiver at a given time.

4. **Receiver**: amplifies the received signal and performs the mixing and demodulation depending on the type of radar device.

5. **Signal Processor**: operates on the output of the receiver to analyze the data, performing the target detection.

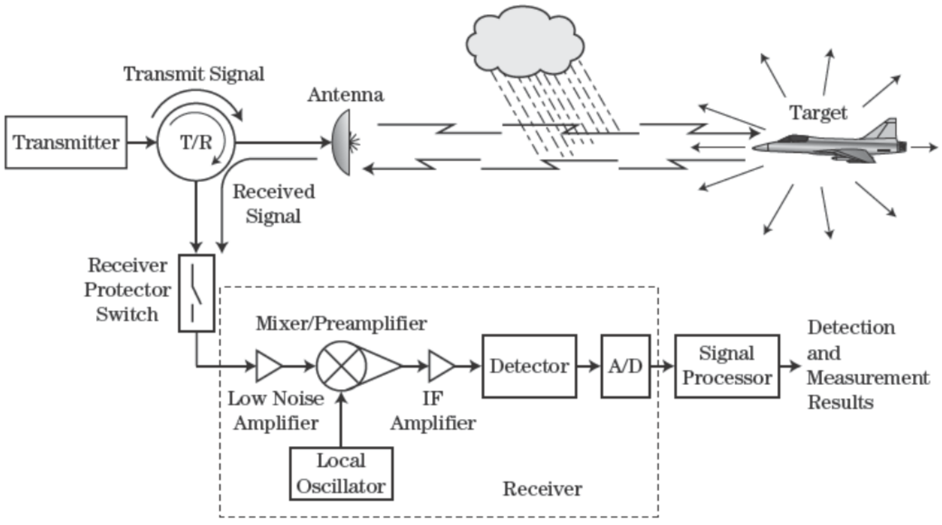


Figure 2.1: General block diagram of a radar device, figure from [1].

There are many factors that may compromise the detection of the target: *thermal noise* in the receiver circuits, unwanted reflections, called *clutter*, from other objects different from the target (i.e. buildings or trees in an outdoor environment, walls in an indoor environment) or interference from other RF sources.

The most basic operation that can be performed by a radar is the target *range estimation*, that is the measurement of the distance of the target from the device. This can be done by, upon receiving the reflected signal from the target, computing the delay of this reception with respect to the original waveform transmission, called τ. Then the expected range of the target can be computed as:

$$R = \frac{c\tau}{2}, \tag{2.1}$$

where the factor 2 accounts for the fact that the distance R is traveled two times by the signal and $c \approx 3 \times 10^8\ m/s$ is the speed of light. To precisely compute τ some kind of *timing mark* is needed on the signal, like the waveform being time-limited, though not all kind of radar devices provide this, as we will see in the following.

Another quantity of interest is the velocity of the target. This can be estimated exploiting the physical phenomenon called *Doppler effect*: when a moving object comes in contact with an electro-magnetic wave with frequency f, the reflected wave undergoes a frequency shift $f + f_d$ where f_d is the Doppler frequency:

$$f_d = 2 \cdot \frac{v}{c} \cdot f = \frac{2v}{\lambda}, \tag{2.2}$$

where $\lambda = c/f$ is the wavelength and v is the relative velocity of the target with respect to the radar. The factor 2 is due to the fact that the Doppler effect affects both the waveform emitted from the transmitter and encounters the target and the reflected copy from the target itself that becomes a secondary source.

2.1.1 Basic Radar Configurations and Waveforms

Radar devices are classified according to their antenna configuration in two main categories:

- **Mono-static**: the receiver and transmitter antennas are the same device, or are very close to each other (i.e. on the same structure).
- **Bi/multi-static**: the transmitter and receiver antennas are enough separated in space to consider them as different devices. In the case of muli-static radars, multiple receiver antennas are displaced in the environment to capture target reflections from multiple angles.

It is important to notice that what distinguishes the two categories is the distance between the TX/RX antennas and not the fact that they are the same antenna or not.

Regarding instead the waveform type, we have the *pulsed* and the *continuous wave* radar:

- **Continuous wave** (CW): the transmitted waveform is continuous in time, therefore the device is always transmitting when in operation. This kind of radar is highly sensitive to transmitter/receiver interference because they need to work simultaneously. For this reason in CW radars usually the bi-static configuration is preferred and the applications are usually limited to close-range [1]. Plain CW transmission can not be used for range estimation because the signal lacks the needed timing mark to compute the delay of the reflection. The velocity of the target is instead easily estimated by computing the frequency shift on the received waveform.

- **Pulsed**: the waveform is limited in time to an interval T called *pulse width*. Pulses are sent each *pulse repetition interval* (PRI), denoted in the following by T_{REP}, and the idle time between the end of the transmission and the successive one is used by the receiver to detect the reflected signal. During transmission the receiver is completely turned off to avoid interference. To perform the detection, after transmitting a pulse the received signal must be sampled with a sampling period long at most T, otherwise the returning waveform may be missed.

  Target ranging can be performed by computing the delay of the returning pulse during the detection, while velocity estimation is usually performed on a *pulse-to-pulse* basis, sampling the Doppler shift at the pulse repetition frequency (PRF) $f_{REP} = 1/T_{REP}$.

The radar device considered in this thesis belongs to the category of LFMCW radars, that exploit linear frequency modulation of the transmitted waveform and a CW approach, as described in the following section.

2.2 LFMCW Radar Model

The LFMCW radar combines the benefits of continuous wave transmission, that is radial velocity estimation, with the capability to estimate the range of the target. This is performed by means of transmitting a sequence of *chirp* signals of duration T every T_{REP}, where the frequency of transmission is linearly increased from a base value f_0 to a maximum f_1, as depicted in Figure 2.2. We denote the quantity $B = f_1 - f_0$ as the *bandwidth* of the chirp pulse. Therefore, the expression of the instantaneous frequency of the transmitted signal is [19]

$$f^{TX}(t) = f_0 + \frac{B}{T}t = f_0 + Kt \qquad \text{for } 0 \leq t \leq T, \tag{2.3}$$

where we defined the *chirp constant* $K = B/T$, that is the slope of the linear relation between time and frequency of the signal.

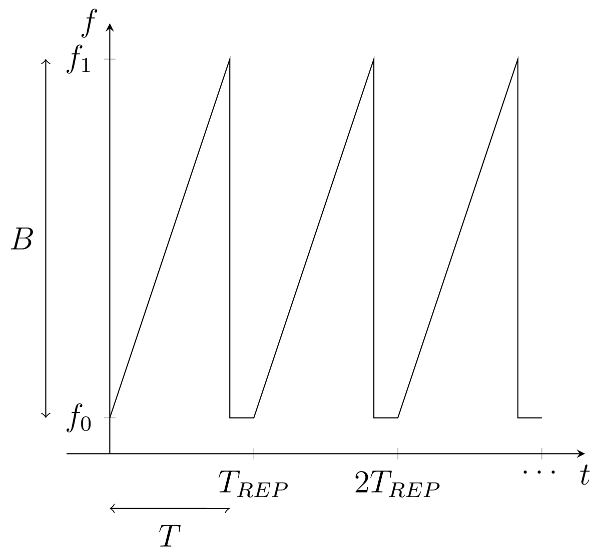


Figure 2.2: Visual representation of the chirp signal parameters.

By definition, the phase of the transmitted signal $\phi^{TX}(t)$ is related to the instantaneous frequency by the following relation

$$\frac{1}{2\pi}\frac{d\phi^{TX}(t)}{dt} = f^{TX}(t), \tag{2.4}$$

so we can derive it as [19]

$$\begin{aligned}\phi^{TX}(t) &= 2\pi\int_0^t f^{TX}(t')dt' = &(2.5)\\ &= 2\pi\left[f_0t' + 0.5Kt'^2\right]_0^t = &(2.6)\\ &= 2\pi\left(f_0t + 0.5Kt^2\right). &(2.7)\end{aligned}$$

If we assume to transmit a complex exponential signal, its expression will then be:

$$s(t) = e^{j2\pi(f_0+0.5Kt)t} \qquad \text{for } 0 \le t \le T \tag{2.8}$$

according to the above argument. In practice radar devices transmit real sinusoidal signals, but the overall discussion is substantially the same.

2.3 Range and Velocity Estimation

The common assumption done with radar devices is that the signal will encounter the target at some point in space, generating a backscattered copy that can be detected at the receiver antenna. For now we assume that the receiver has a single antenna element, while the extension to the Multiple Input Multiple Output (MIMO) case will be done in the next section.

The reflection will be equal to the transmitted waveform with a delay τ depending on the distance of the target from the radar R and its relative radial velocity v. In addition, we need to add a multiplicative complex constant α to account for the reflective properties of the material of the target and a noise term $w(t)$, due to the thermal noise receiver electric circuitry, that we assume to be zero-mean, Gaussian and circularly symmetric. The final expression for the received signal is therefore [20]

$$r(t) = \alpha s(t - \tau) + w(t), \tag{2.9}$$

where τ can be expressed as:

$$\tau = \frac{2(R + vt)}{c}. \tag{2.10}$$

With c we denoted the speed of light $c = 3 \cdot 10^8 m/s$ and the factor 2 accounts for the round trip distance.

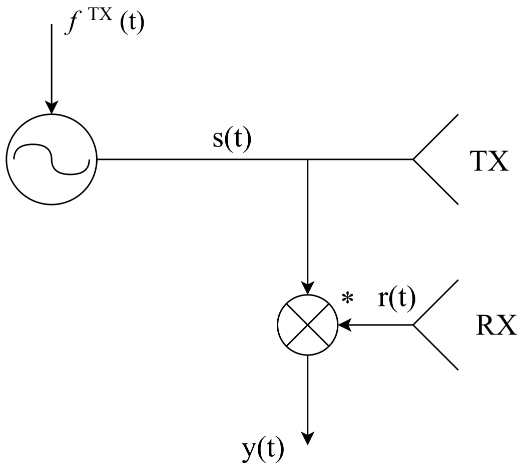


Figure 2.3: FMCW radar simplified block diagram.

Typical receiver implementations for an FMCW radar compute the conjugate product of the transmitted and received signal through a mixer, obtaining the so called *beat signal* (see Figure 2.3)

$$\begin{aligned} y(t) &= s(t)r^*(t) = & (2.11)\\ &= \alpha e^{j2\pi(f_0 t+0.5Kt^2-f_0(t-\tau)-0.5K(t-\tau)^2)} + w'(t) = & (2.12)\\ &= \alpha e^{j2\pi(f_0\tau+Kt\tau-0.5K\tau^2)} + w'(t) \approx & (2.13)\\ &\approx \alpha e^{j2\pi(f_0\tau+Kt\tau)} + w'(t), & (2.14)\end{aligned}$$

where the last approximation is motivated by the fact that usually $\tau/T \ll 1$ so the second order term at the exponent can be neglected [19], and $w'(t)$ is a rotated version of $w(t)$ with the same statistics. The beat signal is the starting point for the estimation of the range and velocity of the target. If we substitute Equation (2.10) into Equation (2.11), we obtain

$$y(t) \approx \alpha e^{j2\pi\left(\frac{2f_0R}{c}+\frac{2f_0v}{c}t+\frac{2KR}{c}t+\frac{2Kv}{c}t^2\right)} + w'(t). \qquad (2.15)$$

The second order term at the exponent is called *range-Doppler coupling* and can be neglected as before. Grouping together the terms that depend on t we get

$$y(t) \approx \alpha e^{j2\pi\left[\frac{2f_0R}{c}+\left(\frac{2f_0v}{c}+\frac{2KR}{c}\right)t\right]} + w'(t), \qquad (2.16)$$

where it is evident how the frequency shift on the received signal has a component due to the Doppler effect $2f_0v/c = 2v/\lambda = f_d$, with λ being the wavelength, and one due to the distance of the target from the radar that we call *beat frequency* $f_b = 2KR/c$.

At the output of the mixer we sample $y(t)$ with sampling frequency $f_s = 1/T_s$. This operation is called *fast time sampling* and is performed on each transmitted chirp pulse. The result can be written as:

$$y(n) \approx \alpha e^{j2\pi\left[\frac{2f_0R}{c}+(f_d+f_b)nT_s\right]} + w'(n) \qquad \text{for } n = 1, \dots, N. \qquad (2.17)$$

Assuming now that a total of P chirps are sent, we can additionally perform a *slow time sampling* by taking a single sample for each pulse, every T_{REP}. Combining the new signal obtained from this operation with the one from Equation (2.17) we obtain a 2-D output for which the expression is derived as follows. Considering slow time sampling, the range at which the target is found in each sampling instant pT_{REP} is given by $R + vpT_{REP}$ with $p = 1, \dots, P$. Therefore, accounting for this modified range, the output of the mixer becomes [20]:

$$\begin{aligned} y(n,p) &\approx \alpha e^{j2\pi\left[\frac{2f_0(R+vpT_{REP})}{c}+\left(\frac{2f_0 v}{c}+\frac{2K(R+vpT_{REP})}{c}\right)nT_s\right]} + w'(n,p) && (2.18) \\ &\approx \alpha e^{j2\pi\left[\frac{2f_0 R}{c}+f_d pT_{REP}+(f_d+f_b)nT_s\right]} + w'(n,p). && (2.19) \end{aligned}$$

The above expression points out that the 2-D output of the mixer is an exponential signal with frequency components $f_d + f_b$ in the fast time domain and f_d in the slow time domain. These frequencies can be extracted by performing first an FFT in the fast time domain, obtaining a delta function centered in $(f_d + f_b)NT_s$ and another FFT in the slow time domain obtaining a delta function centered in $f_d PT_{REP}$, where N and P are the number of samples taken along the fast and slow time domain respectively*. By estimating the location of the two peaks we get estimates of the Doppler frequency f_d and, by subtraction, of the beat frequency f_b, that can be related to the range and radial velocity of the target respectively as

$$\hat{R} = \frac{f_b c}{2K}, \tag{2.20}$$

$$\hat{v} = \frac{f_d c}{2f_0}, \tag{2.21}$$

the last from Equation (2.2). The locations of the peaks could be modified by the presence of noise, therefore leading to a wrong estimate of the target position in the range-Doppler dimension.

2.3.1 Resolution

The resolution of a radar is the precision with which it is able to estimate range and velocity of the target. In the LFMCW case, resolution depends on the bandwidth and on the PRF of the transmission.

Regarding range resolution, we call ΔR the minimum distinguishable range of a LFMCW radar. From Equation (2.20), we see that the range precision depends on

*Note that the number of samples in slow time is equal to the number of transmitted pulses the precision of the estimate of the beat frequency, Δf_b, as

$$\Delta R = \frac{\Delta f_b c}{2K}. \tag{2.22}$$

Due to the Discrete Fourier Transform (DFT) properties, the smallest detectable frequency is equal to the inverse of the chirp duration $1/T$. If we substitute this result into Equation (2.22) we get

$$\Delta R = \frac{c}{2KT} = \frac{c}{2B}. \tag{2.23}$$

The above equation states the important fact that the resolution of a LFMCW radar does not depend on any parameter other than the bandwidth of the chirp pulse. The range precision is therefore not affected by the slope of the linear sweep, by the pulse duration or the PRF.

Velocity resolution instead depends on the Doppler frequency resolution Δf_d, and therefore on T_{REP} as follows

$$\Delta v = \frac{\Delta f_d c}{2f_0}, \tag{2.24}$$

and the precision in the estimate of f_d depends on T_{REP} and L as $\Delta f_d = 1/PT_{REP}$, again because of the properties of the DFT. Equation (2.24) becomes

$$\Delta v = \frac{c}{2f_0 PT_{REP}}. \tag{2.25}$$

The radar parameters also set a maximum range and an interval of velocities beyond which the device cannot get information about the target. The maximum detectable range can be computed from the fast time sampling period T_s as

$$\hat{R}^{\max} = \frac{c}{2T_s K} \tag{2.26}$$

The velocity measurement instead is limited by the sampling theorem: the maximum detectable Doppler frequency is $f_d^{\max} = |f_{REP}/2|$ where we recall that $f_{REP} = 1/T_{REP}$ is the pulse repetition frequency. For this reason the radar device can measure velocities in the interval $[-cf_{REP}/4f_0, cf_{REP}/4f_0]$.

2.4 Angle of Arrival (AoA) Estimation

The previous discussion focuses on the calculation of range and velocity information from a target. However, with a single antenna element that provides omnidirectional transmission, nothing can be inferred about the angular position of the object. With multiple antennas (at least at the receiver) instead, angular information can be extracted from the received signal: if we denote the three dimensional position of a target in spherical coordinates as the triple (R, θ, ϕ), where θ and ϕ are respectively the azimuthal and elevation angle, depending on whether a planar or a linear antenna array is available the full triple or the couple (R, θ) can be estimated respectively. With no loss of generality we focus for now on the case of a linear array, so the estimation of the range R, velocity v and azimuthal angle θ is possible.

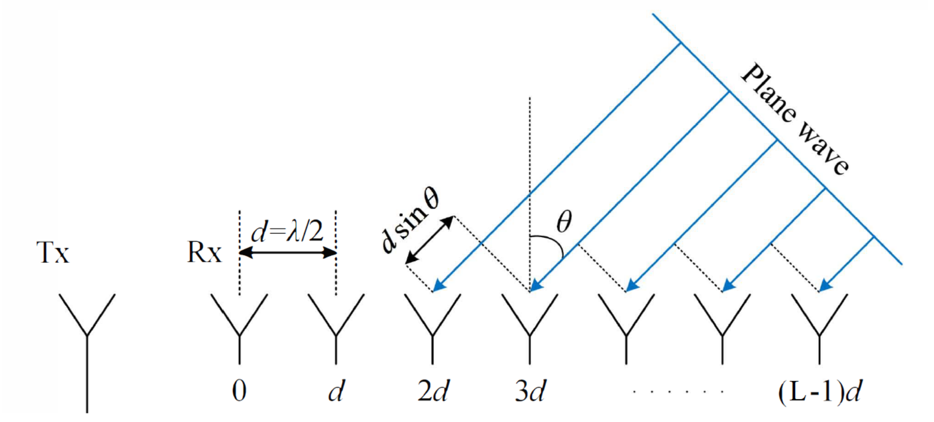


Figure 2.4: Scheme of the transmitter and receiver antenna array (Figure adapted from [2]).

Assume to have a linear antenna array of L elements spaced apart of a distance d from each other[†], where index $l = 0, \dots, L-1$ is used to specify the element number. The different position of each element translates into a different distance from the target and therefore into a different delay. The additional distance that the wave has to travel to reach antenna element l with respect to the reference distance of element 0 is computed as $ld \sin\theta$ thanks to the *far-field approximation*, where θ is the angular position of the target in the azimuth plane. In the most general case, a total of Q targets are present in the radar illumination range, identified

[†]Usually $d \approx \frac{\lambda}{2}$

by indices q and with positions (R_q, v_q, θ_q) in the range-Doppler-azimuth space for $q = 0, \dots, Q-1$. The delay of the reflection from target q at the l-th antenna element is given by

$$\tau_{lq} = \frac{2(R_q + v_q t) + ld \sin\theta_q}{c}. \tag{2.27}$$

Therefore, if we consider the spatial diversity introduced by the multiple antennas at the receiver, we obtain the output of the mixer in the case of linear antenna arrays and multiple targets as the three dimensional signal given by the sum of the reflections from each target at different angles [20]

$$y(n,p,l) \approx \sum_{q=0}^{Q-1} \alpha_q e^{j2\pi\left[\frac{2f_0 R_q}{c} + f_{d_q} p T_{REP} + \frac{f_0 ld \sin\theta_q}{c} + \left(f_{d_q} + f_{b_q}\right) n T_s\right]} + w'(n,p,l), \tag{2.28}$$

where each sample received at a given antenna, denoted by index l, has a different additional delay term according to Equation (2.27). Performing a DFT along the three dimensions n, p, l, we can identify for each target the locations of the three peaks in $(f_{d_q} + f_{b_q}) N T_s$, $f_{d_q} P T_{REP}$ and $f_0 L d \sin(\theta_q)/c$. As in Equation (2.20) and Equation (2.21), we can estimate the range and the velocity of the target q, but now we can also get information regarding its angular position on the azimuthal plane by deriving θ_q from the peak in the l dimension.

2.5 MICRO-DOPPLER

Real radar targets typically present vibration or different moving parts, therefore their overall motion is more complex than just translation. As an example, a person moving has arms and legs rotating around the joints and the head or the chest performing small movements as well. The small-scale vibrations or rotations also cause a Doppler shift that depends on time and is usually represented as a frequency modulation on the reflected signal that presents unique features based on the specific target considered [21].

In [22] is presented an extensive study of this effect, called μ-Doppler *signature* of the target, of which here we give a brief introduction referring to the paper for further details. Assuming that the vibration of the target has the form of a periodic oscillation, it causes a variation of the reflected signal phase expressed as:

$$\phi(t) = \frac{4\pi D_v}{\lambda} \sin(\omega_v t), \tag{2.29}$$

where D_v is the maximum vibration amplitude and λ is the wavelength. Computing the time-dependent Doppler frequency shift as the derivative of the phase we get:

$$f_D = \frac{1}{2\pi}\frac{d\phi(t)}{dt} = \frac{2D_v\omega_v}{\lambda} \cos(\omega_v t), \tag{2.30}$$

therefore the maximum value assumed by the shift is $f_D = 2D_v\omega_v/\lambda$. Equation (2.30) is only valid under the unrealistic assumption that the vibrating motion is a pure oscillation, however it gives an important insight on the fact that for very small λ, even low values of D_v, ω_v give significant frequency shifts. The use of mm-wave radar transmitters is therefore motivated by the higher sensitivity to the μ-D effect.

The extraction of the μ-D signature from the received signal can be performed by computing an STFT on the slow-time sampled waveform to estimate the power spectral density (PSD) along the Doppler dimension, as done in [7]. An alternative is to compute the R-D map first and then integrate along the range dimension [3]. The second option is computationally more expensive but is preferred here for two reasons:

1. The R-D map can be used to locate the person in the range-Doppler space and remove possible noisy peaks with suitable algorithms.

2. From the R-D map a clustering step can be performed to separate different subjects present at the same time in the environment, while the μ-D spectrogram does not allow this distinction lacking the range information.

3 Algorithms

This chapter covers the algorithms and methods used in this thesis from theoretical standpoint. At first, Section 3.1 describes *Kernel Density Estimation* for background removal [23], in Section 3.2 *Density-based Clustering* algorithms are presented [24, 25]. Section 3.3 focuses on *Convolutional Neural Networks* [26] and *Auto-encoders* [27, 28], the deep learning algorithms used for feature extraction and classification.

3.1 Kernel Density Estimation

Kernel density estimation (KDE) is a widely known *non-parametric* approach to the probability density function (PDF) estimation problem [29].

Assume that we have a set of observation points $\boldsymbol{x}_i \in \mathcal{X}, i = 1, \dots, N$ of dimensionality D, drawn from an unknown PDF $p(\boldsymbol{x})$. With this approach, no assumption on the underlying distribution is made, and the target PDF is empirically estimated from the set of N available data points $\boldsymbol{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$ as

$$\hat{p}(\boldsymbol{x}) = \frac{1}{N} \sum_{i=1}^{N} K_\sigma(\boldsymbol{x} - \boldsymbol{x}_i), \tag{3.1}$$

where K_σ is a *kernel function* and σ a parameter called *bandwidth* of the kernel.

A kernel function must satisfy the following properties:

1. $K_\sigma(\boldsymbol{x}) = \frac{1}{\sigma^D} K(\frac{\boldsymbol{x}}{\sigma})$,
2. $K(\boldsymbol{x}) \geq 0$,
3. $\int_{-\infty}^{+\infty} K(\boldsymbol{x}) d\boldsymbol{x} = 1$.

This kind of estimate converges to any distribution for a large N [23]. The simplest choice for the kernel function is derived from the so called *Parzen window*, that represents an hypercube of side σ centered on the origin

$$k(\boldsymbol{x}) = \begin{cases} 1, & |x_j| < \frac{1}{2}, \quad j = 1, \dots, D \\ 0, & \text{otherwise} \end{cases} . \tag{3.2}$$

The above expression implies that the number of observation points that fall into the region delimited by the hypercube of side σ centered in the data point $\boldsymbol{x}_i$ is computed as $\sum_{i=1}^{N} k((\boldsymbol{x} - \boldsymbol{x}_i)/\sigma)$, therefore to satisfy property 3 the Parzen window needs to be normalized with the factor $1/\sigma^D$ obtaining a valid kernel function $K_\sigma(\boldsymbol{x}) = \left(1/\sigma^D\right) k_\sigma(\boldsymbol{x}/\sigma)$ as for property 1.

The resulting estimate of the PDF as in Equation (3.1) is equivalent to an histogram and presents discontinuities due to the definition of $k(\boldsymbol{x})$. To avoid this fact, that leads to the undesirable property of non-differentiability, smoother kernel functions are often used. The most common example is the Gaussian kernel, in which case σ represents the standard deviation and Equation (3.1) becomes:

$$\hat{p}(\boldsymbol{x}) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{||\boldsymbol{x} - \boldsymbol{x}_i||^2}{2\sigma^2}\right). \tag{3.3}$$

The value of the parameter σ is critical to obtain a good estimate, similarly to the problem of choosing a good binning interval for histograms. It must be carefully selected taking into account that a too small value can lead to overfitting on the points in the dataset, while a large one often can not describe the data with sufficient accuracy.

The problem with practical implementations of KDE is computational speed, especially in cases where the number of input data points is large because the complexity of the algorithm is linear in the number of observations [29].

3.2 Density Based Clustering

Density-based clustering can be viewed as an alternative approach to the clustering problem other than the so called *distance-based* algorithms. The central idea is that what identifies a cluster of objects in a dataset is that it has greater density than the outer points, where the meaning of density needs to be precisely defined. Among other density-based techniques, the Density Based Spatial Clustering for Applications with Noise (DBSCAN) algorithm stands out for implementation simplicity and versatility [24]. The algorithm requires two parameters, described in the following, whose selection is not always easy as their correlation can be hard to predict. To solve this problem, a hierarchical modification of DBSCAN, called HDBSCAN, was proposed in [25]. This version works with only one parameter, the minimum number of data points that can form a cluster, whose selection is significantly simpler. The discussion on DBSCAN will serve as an introduction to HDBSCAN, that is the method used in the proposed processing pipeline (see Chapter 4).

3.2.1 DBSCAN

The DBSCAN algorithm in its original version is able to identify clusters based on their density, classifying points with density below a certain threshold as *noise*. Moreover, the number of clusters does not need to be known in advance as in distance-based approaches like *k-means* [30]. We start by describing the notation used in the following and then proceed giving some useful definitions from the original paper [24].

The problem is to cluster a set of N data points called X. Each point belonging to the set will be identified by x_i, for $i = 1, \dots, N$. We also assume that some kind of distance function is defined between the points in X, e.g. the Euclidean distance, denoted by $d(\cdot, \cdot)$. This allows us to formalize the intuitive idea of density, as the property of having at least a certain number of points in a neighborhood specified in terms of the distance d. The algorithm takes as input two parameters, called n_{mpts} and ϵ, representing respectively the minimum number of points needed in the neighborhood and the distance defining it. The following definition expresses how the distance metric relates to the neighborhood:

Definition 1 *(ϵ-neighborhood) The ϵ-neighborhood of a point x_1, denoted by $N_\epsilon(x_1)$*

is the set:

$$N_\epsilon(x_1) = \{x_2 \in X | d(x_1, x_2) \leq \epsilon\} \tag{3.4}$$

If we consider a generic cluster, there will be points that lie inside the cluster, called *core points* and points that are in its border, the *border points.* We define the core points as the points that have at least n_{mpts} points inside their ϵ-neighborhood. Simply requiring that to belong to a cluster a point should have a certain density level, for example at least n_{mpts} points inside its ϵ-neighborhood, would exclude border points, that still belong conceptually to the cluster but have lower density. For this reason the notion of *direct density-reachability* is introduced:

Definition 2 *(Directly density-reachable) A point x_1 is directly density-reachable from another point x_2 with respect to the parameters ϵ and n_{mpts} if:*

1. $x_1 \in N_\epsilon(x_2)$
2. $|N_\epsilon(x_2)| \geq n_{mpts}$

This notion is not symmetric for couples including a core point and a border point. From a core point we can always directly reach a border point according to the above definition, while the opposite does not hold.

Definition 3 *(Density-reachable) A point x_n is density-reachable from another point x_1 with respect to the parameters ϵ and n_{mpts} if there exists a chain of points $x_1, x_2, \ldots, x_n$ such that x_{i+1} is directly reachable from x_i.*

The notion of density-reachability allows to extend the model with a measure of connection between a generic core point and a border point even if they are not in the neighborhood of one another. However, for a point to be density-reachable from another one we must have that all previous points in the chain satisfy the core point condition. Therefore we also introduce the notion of *density-connectedness*, that links together two border points by exploiting the fact that they must be both density-reachable from the same core point.

Definition 4 *(Density-connected) A point x_1 is density-connected to another point x_2 with respect to the parameters ϵ and n_{mpts} if there exists a point x_3 such that both x_1 and x_2 are density reachable from x_3 with respect to the parameters ϵ and n_{mpts}.*

We are now ready to define the notion of *cluster* based on density.

Definition 5 *(Cluster) A cluster C on the set of data points X with respect to ϵ and n_{mpts} is a non-empty subset of X that satisfies the following conditions:*

1. *(Maximality condition) $\forall x_1, x_2$, if $x_1 \in C$ and x_2 is density-reachable from x_1 with respect to ϵ and n_{mpts}, then $x_2 \in C$.*
2. *(Connectivity condition) $\forall x_1, x_2 \in C$, x_1 is density-connected to x_2 with respect to ϵ and n_{mpts}.*

The first condition states that all points that are density-reachable from a point belonging to the cluster must also belong to the cluster. This fact is in agreement with the idea that the cluster can end with points that do not satisfy the core point condition, given that they are density-reachable from points that satisfy it. The second condition instead is needed to guarantee that the cluster is connected, i.e. there is continuity in terms of density.

In addition to defining the clusters, we also intend to classify some of the points in the dataset as noisy points: they do not satisfy the core point condition and they are not density-reachable from any core point, therefore they cannot be considered to belong to any cluster.

Definition 6 *(Noise) Let $C_1, C_2, \ldots, C_k$ be the clusters of the dataset X with respect to the parameters ϵ_i and n_{mpts_i} for $i = 1, \ldots, k$. Then the noise W in X is defined as the set of points that do not belong to any of the clusters:*

$$W \triangleq \{x \in X | \forall i : x \notin C_i\} \tag{3.5}$$

The DBSCAN algorithm is based on the following fact: the clusters can be obtained from dataset X by firstly verifying the core point condition for all $x \in X$, then, starting from a randomly chosen point among the ones that satisfy the property, we can add to the cluster of point x_1 all points that are density-reachable with respect to ϵ and n_{mpts} from it. This approach raises an important point: a cluster is uniquely identified by any of its core points and contains all and only the elements that are density-reachable from it. The following lemmas formalize these concepts.

Lemma 1 *Let $x \in X$ be a data point that satisfies the core point condition, i.e. $N_\epsilon(x) \geq n_{mpts}$. Then the set O of all the points in X that are density-reachable with respect to ϵ and n_{mpts} from x is a cluster with respect to ϵ and n_{mpts}.*

Lemma 2 *Let C be a cluster with respect to ϵ and n_{mpts}. Let $x \in C$ be any data point that satisfies the core point condition, i.e. $N_\epsilon(x) \geq n_{mpts}$. Then it holds that C coincides with the set O of the points in X that are density-reachable from x with respect to ϵ and n_{mpts}.*

3.2.2 HDBSCAN

Hierarchical DBSCAN (HDBSCAN) is an extension of DBSCAN that improves it adding a hierarchical structure to the obtained clusters [25]. This fact has an important implication in removing the need for the parameter ϵ in the algorithm, greatly simplifying the choice of the parameters and enhancing its robustness. In [25], the algorithm is presented by relating it to a slightly modified version of DBSCAN called DBSCAN*, that eliminates the notion of border point and only defines clusters based on core points. The following definitions are the DBSCAN* versions of the ones presented in the previous section. We keep the notation identical for the sake of clarity.

Definition 7 *(Core points) A data point x_1 is called a core point with respect to ϵ and n_{mpts} if its ϵ-neighborhood contains at least n_{mpts} points, i.e. $N_\epsilon(x_1) \geq n_{mpts}$. Any other point for which the core point condition is not satisfied is considered to be noise.*

Core points are defined as in DBSCAN, while border points are eliminated from the model. Points that are not core points are labeled a noise.

Definition 8 *(ϵ-reachability) Two data points x_1 and x_2 are ϵ-reachable with respect to ϵ and n_{mpts} if $x_1 \in N_\epsilon(x_2)$ and $x_2 \in N_\epsilon(x_1)$.*

The new definition of reachability is now transitive.

Definition 9 *(Density-connected) Two data points x_1 and x_2 are density-connected with respect to ϵ and n_{mpts} if they are directly or transitively ϵ-reachable.*

Where by *transitively* ϵ-reachable we mean that, similarly to Definition 3, the point can be reached via a chain where each point is directly reachable from the previous point. In this case however the relation is always symmetric as it was for core points in DBSCAN. The definition of cluster is adapted to include only core points.

Definition 10 *(Cluster) A cluster with respect to ϵ and n_{mpts} is a non-empty maximal subset C of X such that*

$$\forall x_1, x_2 \in C : x_1 \text{ and } x_2 \text{ are density connected.} \tag{3.6}$$

The key idea behind HDBSCAN is to perform a hierarchical clustering on a transformed version of the data points, using a transformation that separates the important data from the noise. Typically, hierarchical clustering requires the creation of a complete graph G_d where each edge is weighted with the value of the distance between the two nodes connected to it. From this graph, a hierarchical partitioning of the dataset is usually obtained by computing the *minimum spanning tree* (MST) and iteratively drop the edges with the lowest weight to construct a *dendrogram* that represents the hierarchy between clusters. This is the approach, for example, of the common *Single Linkage Algorithm* [31].

HDBSCAN follows the same steps but on a different graph, called $G_{n_{mpts}}$, that is based on the *mutual reachability distance* between nodes instead of the normal distance. The definitions below clarify this new concept:

Definition 11 *(Core distance) The core distance of a point $x_1 \in X$ with respect to n_{mpts} is the distance of x_1 from its n_{mpts}-nearest neighbor, denoted by $d_{core}(x_1)$.*

The core distance will generally be small for core points, as their density is high, while it will assume much larger values for noise points.

Definition 12 *(ϵ-core point) A point $x_1 \in X$ is called an ϵ-core point for every value of ϵ that is greater than or equal to the core distance of x_1 with respect to n_{mpts}.*

Definition 13 *(Mutual reachability distance) The mutual reachability distance (MRD) between two points x_1 and x_2 is defined as:*

$$d_{mreach}(x_1, x_2) \triangleq \max\{d_{core}(x_1), d_{core}(x_2), d(x_1, x_2)\}. \tag{3.7}$$

Definition 14 *(Mutual reachability graph) The mutual reachability graph over dataset X is a complete graph $G_{n_{mpts}}$ in which the points in X are the vertices and the weight of each edge is the mutual reachability distance with respect to n_{mpts} between the points connected to it.*

Given that the core distance for noise points is large, for such points it will be the dominant term in Equation (3.7). For this reason, in $G_{n_{mpts}}$ noise points are even more separated from core points than they are in G_d, while core points remain untouched because of the small d_{core} that makes $d(x_1, x_2)$ dominant inside the max operator. This reasoning makes evident how the mutual reachability graph is much more suited than the distance graph for clustering purposes.

For the sake of clarity we report a simple example of the previously described steps. Consider the dataset X of bi-dimensional points as in Figure 3.1, obtained by generating two clusters of bi-dimensional noisy points from two moon-shaped distributions.

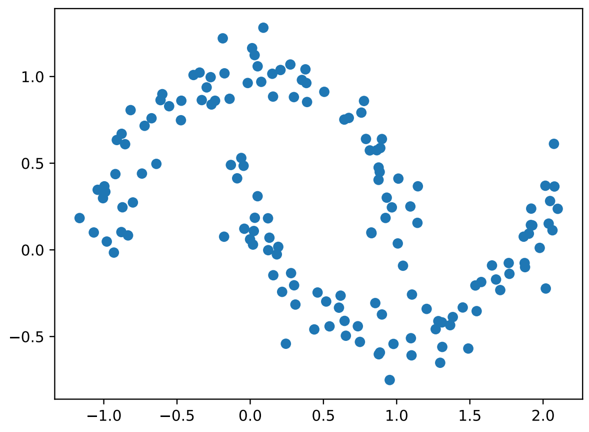


Figure 3.1: Example dataset.

For each point in X the mutual reachability distance is computed, and the MST of the graph $G_{n_{mpts}}$ obtained is depicted in Figure 3.2.

Rearranging the tree into a dendrogram, we obtain the scheme in Figure 3.3. From this structure, every possible output partitioning of the data given by DBSCAN* can be obtained by cutting the dendrogram at a certain level determined the dis-

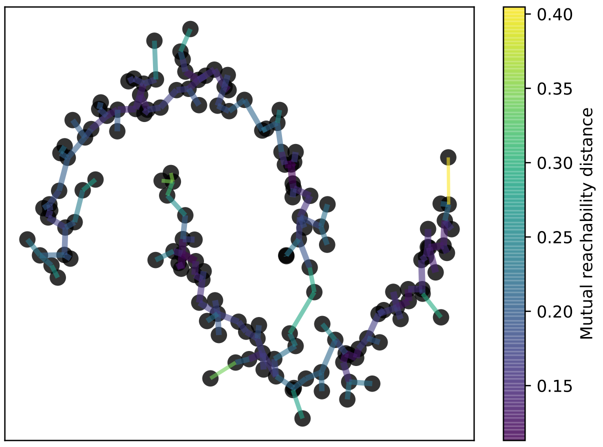


Figure 3.2: Mutual reachability graph MST obtained from the example dataset.

tance ϵ (i.e. removing all the edges with greater value of the distance). For notation simplicity, instead of ϵ we will refer to the parameter $\lambda = 1/\epsilon$.

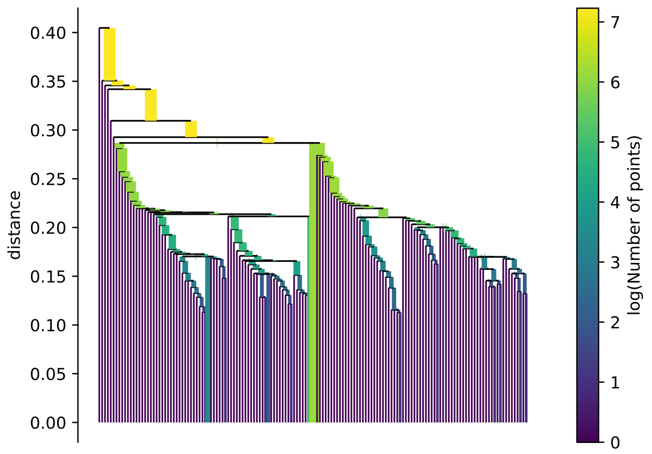


Figure 3.3: Dendrogram obtained from the MST of the example dataset.

So far HDBSCAN only performs single linkage on the transformed graph $G_{n_{mpts}}$.

However, inspecting Figure 3.3 one can easily realize how the obtained dendrogram contains many splits that are not significant to extract a meaningful hierarchy. Indeed, many of the separations that happen by incrementing λ are just single or few points that are excluded from persistent clusters: these events should not be considered as meaningful splits. For this reason, after computing the MST of $G_{n_{mpts}}$, the algorithm simplifies the dendrogram obtained with the following rule:

1. Progressively increase the value of λ (decrease ϵ).
2. At each removal, focus on the resulting components after the split and label as noise every component that contains less than n_{mpts} points.
3. If all components resulting from the split are noise we say that the cluster has disappeared.
4. If a single component is not noise then we interpret the split as the cluster shrinking due to the reduced ϵ and we keep its previous label.
5. If two or more components are not noise then we have a valid split and we assign new labels to each new cluster.

The above procedure is summarized in the following algorithm:

Algorithm 3.1 Condensed tree

```
1: Assign the same label to all $x \in X$ for the root cluster.
2: Sort edges in decreasing order.
3: for all edges in the MST
4:     Set the dendrogram scale to the weight of the edge.
5:     Remove the edge.
6:     After the removal: process each cluster that contained the just removed
       edge(s) and label it with: the noise label if it contains less than $n_{m_{pts}}$ points,
       the same label as the before-splitting cluster if it's the unique non-noise com-
       ponent, new labels if more than one components are non-noise.
7: end for
```

Figure 3.4 shows the resulting tree, after the above procedure. Each cluster is represented by a rectangle whose width represents the number of points it contains. The shrinking of a component corresponds to a shrink of the associated rectangle. This structure is much simpler than before and can be utilized to perform hierarchical clustering over all the possible outputs of DBSCAN*.

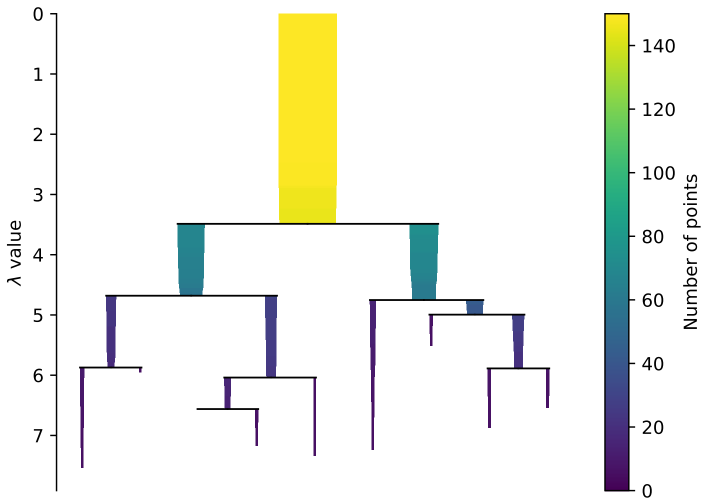


Figure 3.4: Condensed dendrogram of the example dataset.

The problem now is to design a method to extract a flat clustering partition from such a structure, and to do so in a meaningful manner. Intuitively we want to select those clusters that are more robust to variations of λ, because they do not split easily into smaller clusters and contain less noisy points. To do this we introduce the notion of cluster *stability*. We define the quantity $\lambda_{max}(x_j, C_i) = 1/\epsilon_{min}(x_j, C_i)$ as the value of λ for which point x_j is excluded from cluster C_i (either because it is labeled as noise or because of a cluster split), and $\lambda_{min}(C_i) = 1/\epsilon_{max}(C_i)$ the value of λ for which the cluster C_i disappears.

Definition 15 *(Stability) The stability of a cluster in the condensed tree of the HDBSCAN algorithm is:*

$$S(C_i) = \sum_{x_j \in C_i} (\lambda_{max}(x_j, C_i) - \lambda_{min}(C_i)) = \sum_{x_j \in C_i} \left(\frac{1}{\epsilon_{min}(x_j, C_i)} - \frac{1}{\epsilon_{max}(C_i)} \right) \quad (3.8)$$

Therefore, the best clustering partition based on the condensed tree can be obtained by maximizing the overall stability of the selected clusters, i.e. the sum of each stability for clusters in the final partition. In addition, no cluster that is a descendant of an already selected cluster can be chosen, because we search for a flat partition of the data points. To this end the following optimization problem can be solved [25]:

$$\boxed{\begin{array}{ll} \max\limits_{\delta_2,\dots,\delta_K} & \sum\limits_{i=2}^{K} \delta_i S(C_i) \\ \text{s. t.} & \delta_i \in \{0,1\}, \text{ for } i = 2,\dots,K \\ & \sum\limits_{j \in \boldsymbol{I}_h} \delta_j = 1, \ \forall h \in \boldsymbol{L}, \end{array}} \tag{3.9}$$

where:

$$\begin{aligned} \boldsymbol{L} &\triangleq \{h | C_h \text{ is a leaf cluster}\} \\ \boldsymbol{I}_h &\triangleq \{j | j \neq 1 \text{ and } C_j \text{ is ascendant of } C_h\} \end{aligned}$$

$$\delta_i = \begin{cases} 1, & \text{if } C_i \text{ belongs to the chosen partition.} \\ 0, & \text{otherwise.} \end{cases}$$

To solve the problem we can follow a recursive approach. The initial step is to compute the stability of the leaf clusters in the tree. From this information, we can proceed by processing the parent nodes, keeping track of the best so far encountered stability in each subtree. At each cluster division, a decision between keeping the cluster (node) C_i or the best clusters in its subtrees has to be made, therefore we introduce the notion of total cluster stability for the node C_i recursively as follows:

$$\hat{S}(C_i) = \begin{cases} S(C_i), & \text{if } C_i \text{ is a leaf node} \\ \max\left\{S(C_i), \hat{S}(C_{i_1}), \hat{S}(C_{i_2}), \dots, \hat{S}(C_{i_M})\right\}, & \text{if } C_i \text{ is an internal node} \end{cases} \tag{3.10}$$

where $C_{i_1}, C_{i_2}, \dots, C_{i_M}$ are the children of node C_i. After having processed the whole tree the selection of the clusters with the highest stability is straightforward. Algorithm 3.2 provides a summary of the solution steps.

In Figure 3.5 is represented the solution in the case of the example dataset presented above, both on the tree structure and on the points themselves. We can see how HDBSCAN correctly identifies the two clusters and labels outliers as noise. The complete HDBSCAN procedure is stated in algorithm 3.3, it makes use of algorithms 3.1 and 3.2.

Algorithm 3.2 Problem 3.9 solver

```
1: Set $\delta_2 = \delta_3 = \cdots = \delta_K = 1$ and $\hat{S}(C_i) = S(C_i)$ for leaf clusters.
2: Traverse the tree starting from leaf clusters:
3: if $S(C_i) < \sum_{m=1}^{M} \hat{S}(C_m)$:
4:     set $\hat{S}(C_i) = \sum_{m=1}^{M} \hat{S}(C_m)$ and $\delta_i = 0$
5: else
6:     set $\hat{S}(C_i) = S(C_i)$ and $\delta_j = 0$ for $C_j$ in $C_i$ subtrees.
7: end if
```

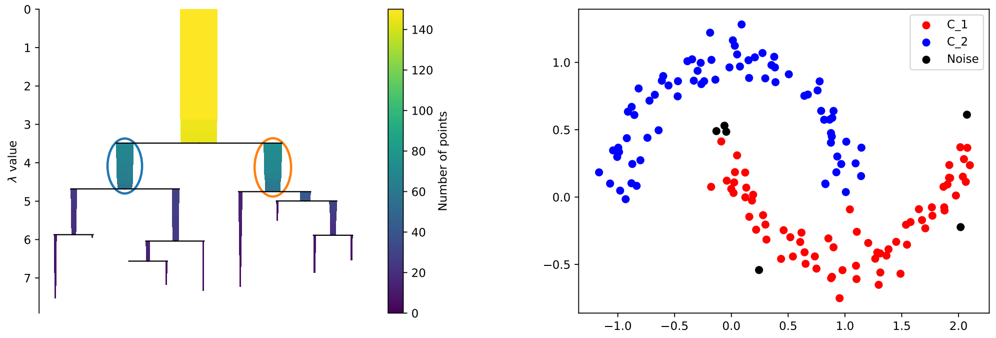


Figure 3.5: Solution of problem (3.9) for the example dataset on the condensed tree (left) and on the points scatter plot (right).

Algorithm 3.3 HDBSCAN

```
1: Compute the core distance w.r.t. $n_{m_{pts}}$ of all $x \in X$.
2: Compute the MRD for all couples $x_1, x_2 \in X$.
3: Build the graph $G_{n_{m_{pts}}}$ based on the MRD and its minimum spanning tree.
4: Obtain the condensed tree structure with algorithm 3.1.
5: Compute the stabilities $S(C_i)$ of each node cluster $C_i$.
6: Solve problem 3.9 recursively with algorithm 3.2.
7: Output the resulting partition.
```

3.3 Overview of Convolutional Neural Networks

Artificial Neural Networks (ANN) have become in recent years the most successful approach to many classification and regression problems. The aim of this section is to introduce the basics of what a Neural Network is and focus mostly on *Convolutional Neural Networks* (CNN), a particular type of ANN that has been used in this work. For a deeper understanding of the basics we refer to [26, 29].

3.3.1 Artificial Neural Networks

An ANN can be represented as weighted graph $G = \{N, E\}$ whose nodes are called *neurons* in analogy with the human brain. Each neuron processes its input x, that it receives through the incoming connections, by applying a non-linear function called *activation function* $f(\cdot)$ to it. Before entering the neuron, the input is weighted by the weights of the incoming edges E^{in} and a constant term, the *bias* b, is added. The final input-output relation of each neuron is:

$$y = f\left(\sum_{e \in E^{in}} w_e x + b\right) \tag{3.11}$$

The structure of each neuron is therefore equal to the *perceptron* [32], a fundamental algorithm in machine learning for linear binary classification, to which a non-linearity is applied. Possible choices for the activation function can be the *sigmoid* function, defined as $\sigma(x) = 1/\left(1 + e^{-x}\right)$ or the more common *Rectified Linear Unit* (ReLU), defined as $ReLU(x) = \max(x, 0)$ [26].

The goal of an ANN is to learn some function f that depends on the input to the network, $\boldsymbol{x} \in \mathbb{R}^d$ and on the weights and biases, that are the network parameters $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{b}\}$. If we think of a classification problem, f can be a mapping from the input space to the discrete set of possible classes $\mathcal{C}$. The simplest architecture for an ANN is the *Feedforward Neural Network* (FFNN), whose neurons are organized in a *Directed Acyclic Graph* (DAG), that processes the input without any structural feedbacks. In addition to the DAG properties, the structure of a FFNN is such that different *layers* of neurons can be identified, and each layer processes only the output of the previous layer without cross-connections, as shown in Figure 3.6. Moreover, each neuron at a certain layer receives as input the output of every neuron at the

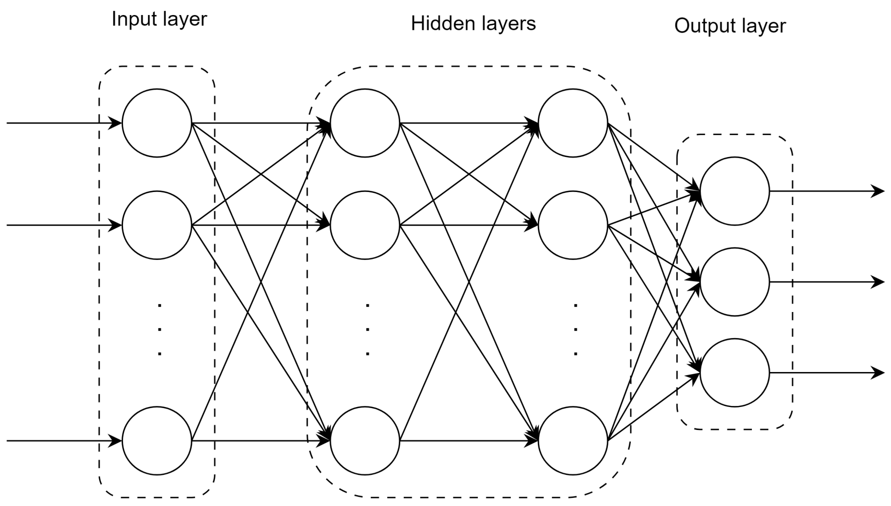


Figure 3.6: Graph structure of a feedforward neural network.

previous layer and passes its output to every neuron of the next layer, i.e. the network is *fully-connected*. The input-output relation of a neuron can be rewritten in the case of a FFNN due to this particular structure:

$$\boldsymbol{y} = f\left(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}\right) \tag{3.12}$$

Layers between the input and the output layer are called *hidden layers*, and when their number is large (i.e. usually more than 5-6), the network is called *deep*.

Usually the output of the network, in the classification case, is a vector with a number of components equal to $|\mathcal{C}|$ constrained between 0 and 1 that express the probability that the input belongs to the class identified by the index of the component. Learning is performed by tuning the parameters in such a way that the output of the network approximates the one of the function f, with the objective of minimizing a specific loss function $\boldsymbol{\mathcal{L}}$. The loss function usually provides a measure of the similarity between the output of the network $\hat{\boldsymbol{y}}$ and the correct output given by the label $\boldsymbol{y}$, averaged over all the available N input samples. Further details on possible choices for the loss function and the training algorithms will be given in Section 3.3.2 for Convolutional Neural Networks.

It has been proved that even with a single hidden layer a neural network can learn to approximate any function, although the cost is in the number of neurons that becomes exponential in the input dimension [33]. The difficulties in the training procedures of deep neural networks, however, have prevented for many years their extensive utilization. The correct configuration of parameters to reach a global minimum of the loss function is usually hard to find, because it requires the solution of a non-convex optimization problem. Iterative methods such as the *Gradient Descent* algorithm and its relatives (see Section 3.3.3) are used, but there is no guarantee that the resulting minimum is global and not local. More details on the implementation difficulties in deep learning is provided in the next section.

3.3.2 Convolutional Neural Networks

Convolutional Neural Networks are a particular kind of FFNN that specializes on learning *spatial* features of the input [26]. A fully-connected FFNN has the following issues:

- Every neuron has connections with every neuron in the previous layer and every neuron in the next layer, therefore the number of weights can be huge in large networks, making the training intractable.
- The fully-connected property does not preserve the spatial features of the input because each input component is fed to every neuron of the following layer. Therefore, the information about *where* a certain component is located in the input vector or matrix is lost.

The CNN solves these problems and has been proved to perform extremely well on input types where spatial structure is important and the number of components is high, like images.

A CNN usually deals with bi-dimensional input data, that can be represented as a three-dimensional structure called *tensor* whose dimensions account for the input data height and width and eventual multiple *channels* (e.g. RGB channels in an image); therefore layers in a CNN are three-dimensional tensors of nodes instead of uni-dimensional arrays. The key difference with respect to the FFNN is the operation performed by each node: the matrix multiplication between the weights and the input is replaced by the convolution with a filter, or *kernel*, K of smaller dimension, that is passed over the whole input I. Each neuron on a layer l is therefore connected only to a region of layer $l-1$, of the same dimension of the kernel, as shown in Figure 3.7, that is called *receptive field* of the neuron. This

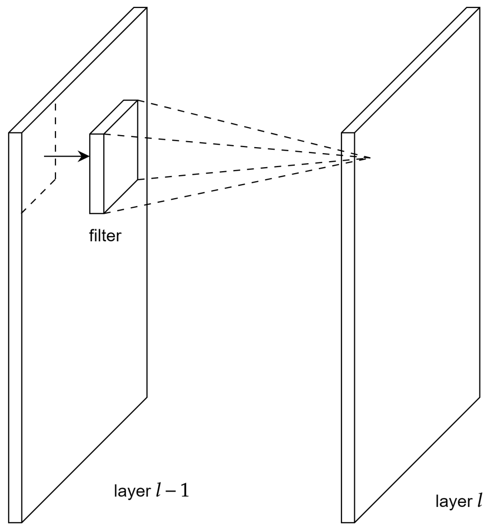


Figure 3.7: Graphical representation of the relation between the filter, the receptive field and the feature map of the next layer.

fact reduces the number of connections drastically with respect to a fully connected architecture, making the number of parameters to train significantly lower. The convolution operation is defined differently from the classical convolution between signals, and is actually a discrete cross-correlation [26]

$$(I \otimes K)(m,n) = \sum_{i=0}^{k_1-1} \sum_{j=0}^{k_2-1} I(i+m, j+n)K(i,j), \tag{3.13}$$

where $k_1 \times k_2$ is the dimension of the kernel.

The result of applying the cross correlation between I and K is a tensor of dimensions that depend on how the filter is slid across the input; the *stride* parameter regulates of how many positions we shift the kernel after each correlation step. If the input dimension does not match with the selected kernel dimension and stride, *padding* can be applied before the transformation. The following equation computes the height/width of the output of the correlation O given the parameters kernel dimension k, stride S, padding P and input dimension D

$$O = \frac{D - K + 2P}{S} + 1. \tag{3.14}$$

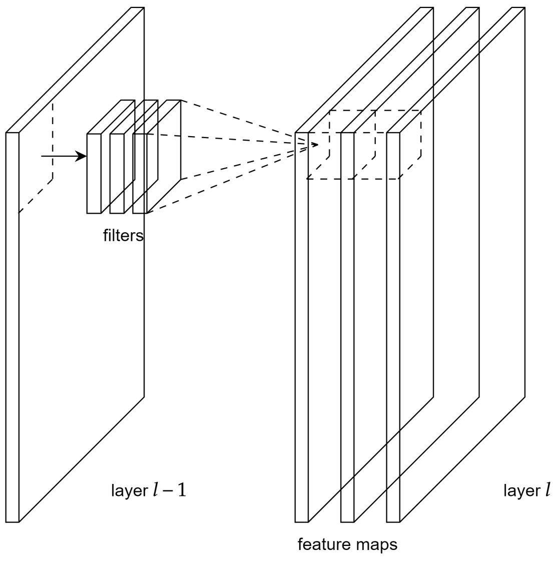


Figure 3.8: Graphical representation of several feature maps at layer l.

Intuitively, we aim to make the network learn meaningful filters K, such that each filter extracts a certain feature in the input. For this reason, the parameters of a filter should be the same across the whole input because the presence of a certain shape does not depend on *where* that shape is located, that is the result should be invariant to translations. This is achieved by *parameter sharing*: at a certain layer, a filter keeps the same weights in the connections with the whole next layer, instead of having a different weight for every neuron in the previous layer. An additional effect of this technique is to further reduce the number of parameters to tune during the training phase. The result of the cross-correlation between a tensor and a kernel is called *feature map*, and at each layer multiple feature maps can be computed with different kernels and therefore different parameters, as shown in Figure 3.8.

A common choice for the activation function of a convolutional layer is the ReLU, because of its simple implementation and good practical results. Recently, however, often the *Exponential Linear Unit* is preferred [34]

$$\mathrm{ELU}(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{if } x \leq 0 \end{cases}, \tag{3.15}$$

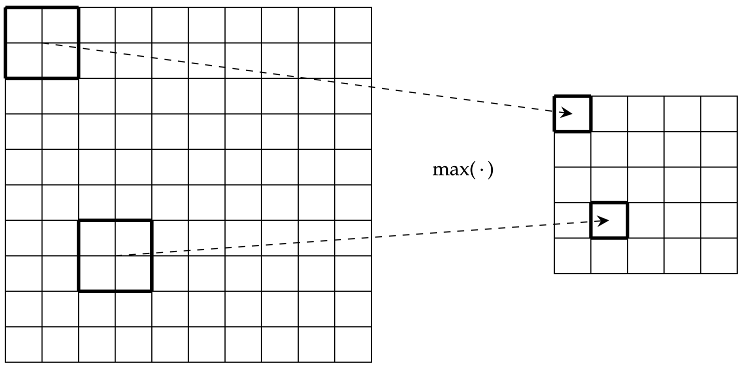


Figure 3.9: Graphical representation of a 2×2 max pooling operation.

because is solves some issues with the ReLU that arise during training (e.g. the vanishing gradient problem, see Section 3.3.3).

As we described, the convolutional layers of a CNN can be seen as extractors of particular shapes or features from the input data. To further improve the robustness of this extraction, and additionally reduce the dimensionality of the hidden layer representations in the feature maps, after each convolution and non-linearity, usually a *pooling layer* is applied. Pooling refers to the following operation: we select a window on the output of the non linearity and retain only one value from this window, that can be the average or, more commonly, the maximum. These two practices are called respectively average pooling and max-pooling. The pooling layer has been proven to be fundamental to improve the learning in CNN, and is based on the fact that losing the information about *where* a feature is located is not important if the relative position with respect to other features is kept intact. Figure 3.9 shows how pooling reduces the dimensionality of the feature maps before the next convolutional layer.

The last layers of a CNN are typically fully connected, as they are used to perform the actual classification after the convolutions and pooling have extracted the most meaningful features from the data. For this reason, after the last pooling operation is applied the hidden tensor representing the feature maps is flattened into a uni-dimensional vector and passed to a fully connected layer. The last layer, as in standard FFNN, contains as many neurons as the number of classes in the classification problem, that we call M, and each neuron represents a class. The activation of this last layer is usually the *softmax* function, defined for each neuron m as follows

$$\text{softmax}(x_m) = \frac{e^{x_m}}{\sum_{j=1}^{M} e^{x_j}}. \tag{3.16}$$

The denominator is a sum over all the last layer neurons, that ensures that the output is between 0 and 1. This output value of neuron m can be interpreted as a probability: it's the probability that the input data belongs to class m. Under this setting, the labels with the correct classification can be provided with a *one-of-K representation* [29], that is each integer label from 1 to M is replaced by an M-dimensional binary vector in which only the component corresponding to the label value is set to 1 and the rest to 0. Ideally, a perfect network should provide probability 1 for the correct class and 0 for the others, therefore matching perfectly the label in one-of-K representation. Measuring the error made by the network can be done using the *cross-entropy loss*

$$\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{n=1}^{N} \sum_{j=1}^{M} y_{nj} \ln(\hat{y}_{nj}), \tag{3.17}$$

where with $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ we represent the correct label and the network output respectively. The loss is accumulated over each input sample for $n = 1, \ldots, N$ and over each neuron in the output layer for $j = 1, \ldots, M$. To train the network, adaptation of the parameters is needed in order to approximate the target labeling function. However, a direct approach is not possible due to the indirect dependencies between the weights in the hidden layers and the loss function that by definition can only be computed on the output of the last layer. The following section describes the training algorithms used in deep learning to solve this issues and perform learning in an efficient way.

3.3.3 Training Algorithm

Optimization in the deep learning field is almost always non-convex [26]. For this reason, numerical methods are necessary to provide a solution to the problem of finding the best network parameters $\boldsymbol{\theta} = \{\boldsymbol{W}, \boldsymbol{b}\}$. The most widely used algorithm is *Gradient Descent* (GD) along with its variants [35]. GD allows to find a stationary point of a differentiable function $f(\boldsymbol{\theta})$, with no guarantees to converge to a global minimum, using a first order Taylor approximation of the function. In a stationary point the gradient of the function with respect to the parameters is a vector of zeros, while in a non-stationary point the gradient is a vector that points in the direction of *steepest ascent* of the function. The steps of the algorithm can be summarized as follows:

1. Start from an initial selection for the parameters $\boldsymbol{\theta}^{(0)}$.
2. Compute a linear approximation of the function using Taylor expansion

$$g(\boldsymbol{\theta}) = f(\boldsymbol{\theta}^{(i)}) - \nabla f(\boldsymbol{\theta}^{(i)})^T(\boldsymbol{\theta} - \boldsymbol{\theta}^{(i)}), \tag{3.18}$$

   where $\nabla f(\boldsymbol{\theta}^{(i)})$ is the gradient vector of f computed in $\boldsymbol{\theta}^{(i)}$.

3. Adjust the parameters in order to go *downhill* in the function f, towards a stationary point where $\nabla f(\boldsymbol{\theta}) = \mathbf{0}$; the downhill direction is given by the opposite of the gradient vector: $-\nabla f(\boldsymbol{\theta}^{(i)})$.
4. Iterate points 2 and 3 until a convergence condition is met. Examples could be that a certain number of iterations have been performed, or that the norm of the gradient vector is small enough.

It can be proven that the correct adjustment on the parameters in order to perform a step towards a local minimum or a saddle point is given by [35]

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \eta\nabla f(\boldsymbol{\theta}^{(i)}), \tag{3.19}$$

where η is an hyperparameter, called *learning rate*, that controls the size of the step performed, influencing the convergence speed and the accuracy of the final result.

To train a neural network, the function that we need to minimize is the loss function $\mathcal{L}$, that not inly depends on the parameters but also on the input $\boldsymbol{x}$. Indeed, we can express the total loss function as the sum of the loss computed on each data point

$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{n=1}^{N} \mathcal{L}(\boldsymbol{y}_n, \hat{\boldsymbol{y}}_n, \boldsymbol{x}_n). \tag{3.20}$$

The global loss computed on all data points and the local loss for a single point will be distinguished in the following by their arguments: we will always make explicit the dependence on $\boldsymbol{x}_n$ for the local loss. Typically we have a sequence of input points belonging to a dataset, so the update in Equation (3.19) can be computed summing over all the N data points

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \eta \sum_{n=1}^{N} \nabla \mathcal{L}(\boldsymbol{\theta}^{(i)}, \boldsymbol{x}_n). \tag{3.21}$$

However, for large datasets, this approach is too costly from the computational point of view, as it requires loading into memory the whole sequence of points and computing the gradient for each of them. For this reason in practical application plain GD as we described it is almost never used. What is actually implemented is a variant of GD called *Stochastic Gradient Descent* (SGD), that uses the following rule for updating $\boldsymbol{\theta}$

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \eta \nabla \mathcal{L}(\boldsymbol{\theta}^{(i)}, \boldsymbol{x}_n). \tag{3.22}$$

The above equation states that for the update in SGD we use only one point from the dataset, therefore only one point needs to be loaded into memory at each step. SGD uses multiple updates as the one in Equation (3.22) instead of a single update using the whole dataset. This approach leads to the same update as the standard GD *on average* if we pick the vector $\boldsymbol{x}_n$ uniformly at random each time [33], but the partial updates can also lead to a completely different update path than standard GD.

Compromises between standard GD and SGD are also used, and are the most popular choice in practical implementations. In these cases instead of one single point, a so called *minibatch*, i.e. a small random selection of points from the dataset, is loaded and used for the update. The update rule in this case is

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \eta \sum_{\boldsymbol{x}_n \in \mathcal{B}} \nabla \mathcal{L}(\boldsymbol{\theta}^{(i)}, \boldsymbol{x}_n), \tag{3.23}$$

where we denote the minibatch with $\mathcal{B}$. Typical sizes for the minibatch are powers of 2 like $32, 64, \ldots, 256$ to fully exploit parallelization properties of the computing hardware (in particular of GPUs). The utilization of a small number of points provides a trade-off between memory usage or computational speed and stability in the convergence of the algorithm.

To perform learning in neural networks we need to apply Equation (3.23) iteratively, so the knowledge of the gradient of the loss function with respect to every network parameter is required. The efficient computation of the gradient is the most difficult part in the training phase of an ANN, and is done in two phases: *forward pass* and *backpropagation* described in the following sections. Forward pass consists in feeding the network with an input sample and propagating it forward towards the output, performing all the convolutions and operations needed at each layer. When the sample reaches the output layer the network output $\hat{\boldsymbol{y}}$ is computed and the value of the partial loss function $\mathcal{L}(\boldsymbol{\theta}^{(i)}, \boldsymbol{x}_n)$ is obtained. From the forward pass we also retain the intermediate values of the outputs of the activation functions and their inputs for every neuron in the network. Using this information, with the backpropagation the gradient of the loss with respect to each parameter is computed proceeding backwards from the output layer to the input. Alternating forward pass and backpropagation for every input sample allows the learning procedure with SGD to be completed. We say that an *epoch* is finished when all the data samples used for training have been used once to update the parameters. Typically deep networks are trained for several epochs before the loss function converges.

Forward pass

The forward pass requires an input sample $\boldsymbol{x}$ and a current choice for the parameters of the CNN, $\boldsymbol{\theta}$. As we are discussing CNNs, the input is assumed to be bi-dimensional, and so are the kernels for each feature map and each layer (we describe the simpler case of single-channel input, but the case is similar for RGB images or other data with more channels). A CNN is assumed to be structured as a sequence of convolutional and pooling layers, followed by some final fully connected layers. We introduce the following notation:

- $l = 1, 2, \ldots, L$ represents the layer index.
- The input feature map to a layer has shape $H \times W$ with respective indices i, j.
- The kernel $\boldsymbol{w}$ for a certain layer has shape $k_1 \times k_2$ with indices m, n.
- $w^l_{m,n}$ represents the weight connecting neurons from layer $l-1$ to layer l.
- b^l is the bias for layer l.
- $f(\cdot)$ is the activation function.

- $y^l_{i,j}$ is the output at layer l:

$$y^l_{i,j} = f(a^l_{i,j}) \tag{3.24}$$

- $a^l_{i,j}$ is the activation at layer l, that is the result of the cross-correlation between the receptive field in the output of layer $l-1$ and the kernel of layer l.

- $\boldsymbol{\mathcal{L}}(\boldsymbol{w}, \boldsymbol{b})$ is the loss function that depends on the collection of the network parameters.

A generic convolutional layer l applies a certain number of filters to the input tensor. Each filter is passed through the whole tensor computing the cross-correlation with the receptive field of each neuron in layer $l+1$. The activation of a neuron is computed as

$$a^l_{i,j} = \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} w^l_{m,n} y^{l-1}_{i+m,j+n} + b^l, \tag{3.25}$$

and applying the activation function the output of a neuron is

$$y^l_{i,j} = f(a^l_{i,j}) = f\left(\sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} w^l_{m,n} y^{l-1}_{i+m,j+n} + b^l\right), \tag{3.26}$$

After the convolution and non-linearity are computed, pooling is usually applied, selecting for each window defined by the pooling width and height the maximum or the average value depending on the pooling type.

Equation (3.26) can be used for every convolutional layer in sequence to propagate the input inside the network. In the last layers however the situation changes, because of the full connectivity. After the last convolution or pooling is applied, the output is flattened into an uni-dimensional vector, that is passed through the feedforward layers using Equation (3.12). The last layer applies softmax activation (Equation (3.16)) to obtain the network output or *prediction* $\hat{\boldsymbol{y}}$, that is used to compute the loss $\boldsymbol{\mathcal{L}}(\hat{\boldsymbol{y}}, \boldsymbol{y}, \boldsymbol{x})$, concluding the forward propagation.

Backpropagation

Backpropagation is the standard algorithm used to train every neural network, so it can be also applied to the CNN. It is an iterative algorithm that allows to compute the gradient of the cost function with respect to each network parameter in order to perform the weights and biases update as in Equation (3.22). The algorithm has to be executed after a forward pass through the network, because it requires the knowledge of the activations and the outputs of every neuron, and uses the chain rule of the derivative to compute the gradient of the loss starting from the last layer and going backwards propagating the gradients at each layer in a recursive manner (hence the name backpropagation)*. In the following we provide the detailed mathematical description of the algorithm in the case of CNNs, using the same notation as in the forward pass description.

The aim of the backpropagation algorithm is to compute the gradient of the cost function with respect to every parameter of the network, that is

$$\frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{b})}{\partial w^l_{m,n}}, \qquad \forall \quad l, m, n \tag{3.27}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{b})}{\partial b^l}, \qquad \forall \quad l. \tag{3.28}$$

Knowing this quantities means knowing how much each single parameter influences the total loss, allowing the computation of the upgrade step in the SGD iteration. For notation simplicity we omit the dependence of the loss from the currently processed input sample as it would lead to confusion between the indices. Anyway, all the processing done in the forward and backward procedures has to be done for *each* input sample so specifying it would be redundant.

We first focus on the weights $\boldsymbol{w}$. The following decomposition holds, from the chain rule of the derivative

$$\frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{b})}{\partial w^l_{m,n}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{b})}{\partial a^l_{i,j}} \frac{\partial a^l_{i,j}}{\partial w^l_{m,n}} \tag{3.29}$$

$$= \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta^l_{i,j} \frac{\partial a^l_{i,j}}{\partial w^l_{m,n}}, \tag{3.30}$$

where we defined $\delta^l_{i,j} = \frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{b})}{\partial a^l_{i,j}}$ as the derivative of the loss with respect to the activation at layer l. Now, using Equation (3.25), we can compute the term $\frac{\partial a^l_{i,j}}{\partial w^l_{m,n}}$

*The chain rule of the derivative allows to compute the derivative of a composite function $f(g(x))$ as $\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx}$. In the case of backpropagation, this method can be used to unfold the dependencies of the gradient of the loss from the activations and the outputs in the hidden layers.

as simply

$$\frac{\partial a^l_{i,j}}{\partial w^l_{m,n}} = y^{l-1}_{i+m,j+n}, \tag{3.31}$$

so Equation (3.29) becomes

$$\frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{b})}{\partial w^l_{m,n}} = \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta^l_{i,j} y^{l-1}_{i+m,j+n} = \boldsymbol{\delta}^l \otimes \boldsymbol{y}^{l-1}_{m,n}, \tag{3.32}$$

that is the cross-correlation between the output of layer $l-1$ and the derivative of the loss with respect to the activation at layer l. At this point we are left with computing $\delta^l_{i,j}$, with the chain rule of the derivative we can decompose it exploiting its relation with the activation of the next layer

$$\begin{aligned} \delta^l_{i,j} = \frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{b})}{\partial a^l_{i,j}} &= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{b})}{\partial a^{l+1}_{i-m,j-n}} \frac{\partial a^{l+1}_{i-m,j-n}}{\partial a^l_{i,j}} && (3.33) \\ &= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \delta^{l+1}_{i-m,j-n} \frac{\partial a^{l+1}_{i-m,j-n}}{\partial a^l_{i,j}}, && (3.34) \end{aligned}$$

if we can express the last term in the above equation as a function of known elements then we have completed a recursive formulation for $\boldsymbol{\delta}^l$, that could be computed knowing $\boldsymbol{\delta}^{l+1}$. We have that

$$a^{l+1}_{i-m,j-n} = \sum_{m'=0}^{k_1-1} \sum_{n'=0}^{k_2-1} w^{l+1}_{m',n'} y^l_{i-m+m',j-n+n'} + b^{l+1}, \tag{3.35}$$

therefore

$$\begin{aligned} \frac{\partial a^{l+1}_{i-m,j-n}}{\partial a^l_{i,j}} &= \frac{\partial}{\partial a^l_{i,j}} \left(\sum_{m'=0}^{k_1-1} \sum_{n'=0}^{k_2-1} w^{l+1}_{m',n'} y^l_{i-m+m',j-n+n'} + b^{l+1} \right) && (3.36) \\ &= \frac{\partial}{\partial a^l_{i,j}} \left(\sum_{m'=0}^{k_1-1} \sum_{n'=0}^{k_2-1} w^{l+1}_{m',n'} f\left(a^l_{i-m+m',j-n+n'} \right) \right). && (3.37) \end{aligned}$$

If we consider the last expression, it's easy to notice that whenever $m' \neq m, n' \neq n$, the summation becomes a constant with respect to the derivation variable $a^l_{i,j}$.

Therefore the derivative is non-zero only for $m' = m$ and $n' = n$

$$\frac{\partial a^{l+1}_{i-m,j-n}}{\partial a^{l}_{i,j}} = w^{l+1}_{m,n} \frac{\partial f(a^{l}_{i,j})}{\partial a^{l}_{i,j}} = w^{l+1}_{m,n} f'(a^{l}_{i,j}). \tag{3.38}$$

Using this result into Equation (3.33) we get

$$\begin{aligned} \delta^{l}_{i,j} &= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \delta^{l+1}_{i-m,j-n} w^{l+1}_{m,n} f'(a^{l}_{i,j}) && (3.39) \\ &= f'(a^{l}_{i,j}) \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \delta^{l+1}_{i-m,j-n} w^{l+1}_{m,n}. && (3.40) \end{aligned}$$

The above expression allows to compute the values of $\boldsymbol{\delta}^{l}$ in a recursive fashion, using $\boldsymbol{\delta}^{l+1}$. For layer L, however, we do not have a following layer, so the derivative of the loss with respect to the activation is just

$$\delta^{L}_{i,j} = \frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{b})}{\partial y^{L}_{i,j}} f'(a^{L}_{i,j}), \tag{3.41}$$

that can be directly computed knowing the activation at layer L and the output of the network, values that are obtained in the forward pass. Therefore, after performing a forward pass, all values needed are available to compute recursively the values of $\boldsymbol{\delta}$ at all layers from L to the input one. Once all $\boldsymbol{\delta}$ are known, Equation (3.32) can be used locally at each layer to compute the gradient of the loss with respect to that layer's weights, and consequently perform the GD update as

$$w^{l}_{m,n} = w^{l}_{m,n} - \eta \frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{b})}{\partial w^{l}_{m,n}}. \tag{3.42}$$

To update the biases we follow similar steps, starting from the gradient of the loss with respect to the biases

$$\begin{aligned}\frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{b})}{\partial b^l} &= \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{b})}{\partial a_{i,j}^l} \frac{\partial a_{i,j}^l}{\partial b^l} && (3.43)\\ &= \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta_{i,j}^l \frac{\partial a_{i,j}^l}{\partial b^l} && (3.44)\\ &= \sum_{i=0}^{H-k_1} \sum_{j=0}^{W-k_2} \delta_{i,j}^l, && (3.45)\end{aligned}$$

where the last passage is motivated looking at the expression of the activation in Equation (3.25), that has a linear dependence with coefficient 1 from the bias, therefore the derivative it's equal to 1.

The SGD update for the biases is then

$$b^l = b^l - \eta \frac{\partial \mathcal{L}(\boldsymbol{w}, \boldsymbol{b})}{\partial b^l}. \tag{3.46}$$

3.3.4 Advanced Optimization Techniques

In most practical cases, optimization of an ANN parameters with plain SGD may perform poorly. Many more advanced techniques have been developed, for a detailed discussion we refer to [26]. Here we will focus on giving a brief description of the most widely used approaches in order to improve SGD during the learning phase. A simple idea that however has a great impact on the results is to make the learning rate η adaptive during the epochs, starting with an higher value that is lowered as the training proceeds. The intuition behind this procedure is that a faster convergence is possible if an higher learning rate is used, but the risk is to *overshoot* the solution, so an initial high learning rate that becomes smaller as the epochs pass is a good trade-off between the two cases. In this case the learning rate is considered as a sequence of learning rates η_t where the index t refers to the epoch number. A sufficient condition for SGD to converge is that [26]

$$\begin{cases} \sum_{k=1}^{+\infty} \eta_k \to \infty \\ \sum_{k=1}^{+\infty} \eta_k^2 < \infty \end{cases}. \tag{3.47}$$

Linear or exponential decay rates are widely used.

Another technique is the one of SGD with *momentum*. The update rule of SGD is modified as follows, introducing a new quantity $\boldsymbol{v}$ that can be regarded as the *velocity* of the update process during the epochs [26]

$$\begin{array}{l} \boldsymbol{v}^{(i+1)} \leftarrow \alpha \boldsymbol{v}^{(i)} - \eta \nabla \mathcal{L}(\boldsymbol{\theta}^{(i)}, \boldsymbol{x}_n) \\ \boldsymbol{\theta}^{(i+1)} \leftarrow \boldsymbol{\theta}^{(i)} + \boldsymbol{v}^{(i+1)} \end{array} . \tag{3.48}$$

The network parameters are updated by the quantity $\boldsymbol{v}$, that is a weighted sum (controlled by the hyperparameter α) between the past value of $\boldsymbol{v}$ and the standard update of SGD based on the gradient of the loss. This causes the update process to gain some *momentum*, in analogy with physics, in the direction where most of the updates point. The effect is an increased stability of the optimization procedure, that is less likely to oscillate randomly as in standard SGD.

Several other methods of increasing complexity have been studied, combining momentum with adaptive learning rate, and trying to introduce information about a second-order approximation of the cost function, we mention RMSprop [36], Adadelta [37] and Adam [38]. In the particular case of Adadelta, the optimizer used in this work, no learning rate needs to be provided as it is adaptively chosen by the algorithm itself for each parameter, based on the intensity of previous updates.

3.3.5 The Problem of Overfitting

A severe problem in deep learning is the one of *overfitting*. When training a neural network, or a machine learning algorithm in general, what we would ideally do is to minimize the so called *true error* of the model, i.e. the expected error under the input data distribution

$$\mathcal{L}^{true} = E_{p(X)} \left[\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}, \boldsymbol{x}) \right] \tag{3.49}$$

where $p(X)$ represents the true input data true distribution and $\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}, \boldsymbol{x})$ is the loss function depending on the true label $\boldsymbol{y}$, the algorithm prediction $\hat{\boldsymbol{y}}$ and the input sample $\boldsymbol{x} \sim p(X)$. In general $p(X)$ is however unknown, so we can not compute the expectation in Equation (3.49) and we need to instead minimize the *training* error, i.e. the empirical average of the loss function over the training samples, using the so called *Empirical Risk Minimization* (ERM) paradigm. The relation between the true and the training error has been widely investigated, and we refer to [33] for a detailed discussion. The main risk of the ERM approach is that the algorithm specializes on the training set, obtaining low values of the training error, but is not able to generalize because it has not learned the features of the true input distribution. Intuitively, the problem of overfitting can be though in this way: minimizing the empirical error instead of the true one is most of the times the best we can do, having at our disposal only the training samples to perform the learning phase, however if too many parameters are available to our model, it may end up memorizing the desired output result for each sample in the training set, performing poorly when faced with unseen test data. This problem is especially severe with deep learning models that easily have hundreds of thousands or millions of tunable parameters and therefore more ”memorization” capabilities.

Here we present three widely used methods to counter or avoid overfitting when training neural networks, namely *regularization*, *dropout* and *data augmentation.*

1. **Regularization**: This approach is the most studied one and presents several other benefits other than reducing overfitting, like making the loss function more convex thereby simplifying the learning optimization [33]. It consists in modifying the loss function by adding a term that penalizes models that are *too complex*, i.e. are not enough general. A common type of regularization is the L_2 regularization, where the additional term in the loss is $\lambda||\boldsymbol{\theta}||^2$, where $\boldsymbol{\theta}$ represents the sum of all the network parameters and λ is a tunable parameter that controls the strenght of the regularization with respect to the empirical error minimization. It has been proven that reducing the norm squared of the network weights has an effect in keeping under control the complexity of the model [26, 33]. The complete loss function when L_2 regularization is used is therefore in the form

$$\mathcal{L}^{reg}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(\boldsymbol{y}_n, \hat{\boldsymbol{y}}_n, \boldsymbol{x}_n) + \lambda||\boldsymbol{\theta}||^2. \tag{3.50}$$

   From the above equation one can see that minimizing a regularized loss function leads to different approximation capabilities on the training set, in general lowering the accuracy of the model because the minimization is carried out also on a term that does not depend on the training data. However this comes with better generalization capabilities on test samples and better modeling of the input distribution that is our final aim.

2. **Dropout**: While regularization is applied on a large number of machine

learning models and comes from theoretical motivations, dropout is a specific technique tailored for neural networks driven by empirical considerations. The idea is very simple, yet incredibly effective: when training a network, at each batch iteration we randomly ignore (*drop*) some neurons with a probability p_{drop}, not updating their weights nor considering their output for the forward and backward passes. This procedure causes the model to better generalize because it makes the training more random, and each neuron can not specialize on certain input samples due to the possibility of being dropped and therefore prevented from influencing the output. Using dropout basically allows to train, with no additional computational cost, multiple models at once, each obtained by removing a set of neurons from the original model. This implies, citing [26], that dropout *regularizes each hidden unit to be not merely a good feature, but a feature that is good in many different contexts.*

3. **Data augmentation**: The key problem in overfitting is that the minimization of the training error may not correspond to a minimization of the true error. Therefore a good method to reduce the difference between the two is enlarging the training set, making it more difficult for the algorithm to specialize on each sample and at the same time making the training data more representative of the distribution $p(X)$. In practice however training data is often scarce, so techniques to artificially enlarge it are used: adding noise, rescaling, cropping etc. are all valid and widely utilized methods to generate more data from the available set.

3.4 Auto-Encoders

For the training of deep neural networks, where the optimization problem is in most of the cases non-convex, the random initialization of the parameters is often not enough to ensure good results or even the training convergence [28]. For this reason, methods to provide an intelligent way to initialize weights and biases before the backpropagation training have been widely studied. Several works have shown that *unsupervised pre-training* of single or multiple layers can improve significantly the network learning by producing an high-level representation of the data encoded in the network parameters, that can be afterwards trained in the usual supervised manner to be *fine-tuned* on the particular dataset [39], [28]. Moreover, as we will show in the following, the problem of learning an higher level representation of the input is strictly related to the one of data compression and dimensionality reduction.

[27] showed that a good way to carry out the pre-training is using a neural network structure called *autoencoder* (AE): this particular neural network is made of two parts, the *encoder*, that learns a lower dimensional representation of the data, and the *decoder* that learns how to reconstruct the original input from the compressed version. Focusing on a single layer structure for simplicity, we can mathematically define an AE as follows.

We call encoder, $f_{\theta_e}(\boldsymbol{x})$, the deterministic mapping that taken as input a vector $\boldsymbol{x}$ of dimension d transforms it into its compressed representation $\boldsymbol{y}$ of dimension $l < d$, parameterized on the collection of parameters θ_e. In the neural network framework, this function can be represented by an affine function specified by the network's weights and biases $\theta_e = \{\boldsymbol{W}_e, \boldsymbol{b}_e\}$, followed by a non-linearity $a(\cdot)$

$$f_{\theta_e}(\boldsymbol{x}) = \boldsymbol{y} = a\left(\boldsymbol{W}_e\boldsymbol{x} + \boldsymbol{b}_e\right),$$

where $\boldsymbol{W}_e$ is a $l \times d$ matrix and $\boldsymbol{b}_e$ a vector of l elements [28]. Notice that the above formulation is equivalent to the one of a single layer neural network.

The decoder can be defined instead as the parametrized function

$$g_{\theta_d}(\boldsymbol{y}) = \hat{\boldsymbol{x}} = a\left(\boldsymbol{W}_d\boldsymbol{y} + \boldsymbol{b}_d\right),$$

where $\boldsymbol{W}_d$ is a $d \times l$ matrix and $\boldsymbol{b}_d$ a d-dimensional vector [28].

The output of the decoder $\hat{\boldsymbol{x}}$ can be seen as an attempt to reconstruct the input from its compressed version, therefore the loss function of the network should intuitively compare this reconstruction to $\boldsymbol{x}$, measuring their similarity. However, what is really important about the way the AE learns is that it actually not only maximizes the similarity between the input and the reconstructed output, but also the *mutual information* between the random variable generating the input and the one generating its lower dimensional representation $\boldsymbol{y}$.

Let's assume $\boldsymbol{x}$ and $\boldsymbol{y}$ are realizations of the corresponding random variables X and Y, generalizing the previous discussion to stochastic mappings of the type $q(Y|X;\theta)$. The mutual information between X and Y can be written as $I(X,Y) = H(X) - H(X|Y)$, where $H(X)$ is the Shannon entropy of the random variable X. Denoting with $E_{p(\cdot)}[\cdot]$ the expectation under probability distribution $p(\cdot)$, entropy and mutual information are defined as

$$H(X) = E_{p(X)}\left[-\log(p(X))\right], \tag{3.51}$$

$$I(X,Y) = E_{p(X,Y)}\left[\log\left(\frac{p(X,Y)}{p(X)p(Y)}\right)\right]. \tag{3.52}$$

If we want Y to retain as much information as possible regarding X then we should select the θ that makes $q(Y|X;\theta)$ such that $I(X,Y)$ is maximal. Using the previously stated decomposition for the mutual information, and noticing that the term $H(X)$ does not depend on θ as the distribution of the input is unknown and non modifiable, the problem of maximizing $I(X,Y)$ is equivalent to

$$\begin{aligned}\arg\max_{\theta} I(X,Y) &= \arg\max_{\theta} H(X) - H(X|Y)\\ &= \arg\max_{\theta} -H(X|Y)\\ &= \arg\max_{\theta} E_{q(X,Y;\theta)}\left[\log q(X|Y;\theta)\right].\end{aligned}$$

In addition, the following inequality holds for any distribution $p(X|Y)$: $E_{q(X,Y)}\left[\log p(X|Y)\right] \leq E_{q(X,Y)}\left[\log q(X|Y)\right]$ [28], so the new optimization problem over the parametrized distribution $p(X|Y;\theta')$

$$\arg\max_{\theta,\theta'} E_{q(X,Y;\theta)}\left[\log p(X|Y;\theta')\right] \tag{3.53}$$

is a bound maximization problem on the mutual information between X and Y. This general discussion holds in particular in the case of deterministic mappings p, q, as we required in the AE description. Therefore, setting $q(Y|X;\theta) = \delta(Y - f_\theta(X))$ we fall back to the previously described situation in which $Y = f_\theta(X)$ and the bound maximization becomes

$$\arg\max_{\theta,\theta'} E_{q(X)}\left[\log p(X|Y;\theta')\right]. \tag{3.54}$$

In practice, the exact distribution $q(X)$ is unknown, but given the available training set we can approximate the expectation in Equation (3.54) with an empirical average over the training samples. Finally, we set the AE loss function to be proportional to the negative log-likelihood of the data, given the output of the network

$$\mathcal{L}(\boldsymbol{x}, \hat{\boldsymbol{x}}) \propto -\log p(\boldsymbol{x}|\hat{\boldsymbol{x}}), \tag{3.55}$$

that under Gaussian assumption for $p(\boldsymbol{x}|\hat{\boldsymbol{x}})$ becomes that squared-distance loss, and under Bernoullian assumption and $\boldsymbol{x} \in [0,1]^d$ is the cross-entropy loss[†].

The AE training with SGD minimizes the empirical average of the loss function with respect to the encoder and decoder parameters θ_e, θ_d

$$\begin{aligned} \arg\min_{\theta_e,\theta_d} E_{\tilde{q}(X)}\left[\mathcal{L}(X,\hat{X})\right] &= \arg\max_{\theta_e,\theta_d} E_{\tilde{q}(X)}\left[\log p(X|g_{\theta_d}(f_{\theta_e}(X)))\right] && (3.56) \\ &= \arg\max_{\theta_e,\theta_d} E_{\tilde{q}(X)}\left[\log p(X|Y = f_{\theta_e}(X); \theta_d)\right], && (3.57) \end{aligned}$$

where $E_{\tilde{q}(X)}$ represents the empirical average over the available data samples and the first passage is due to Equation (3.55). Equation (3.56) is therefore equivalent to equation Equation (3.54), only replacing the expectation with its empirical version and setting $\theta = \theta_e$ and $\theta' = \theta_d$.

The previous discussion proves that training an AE implies learning an higher level representation of the input data, generating a lower dimensional representation. This is ensured by the fact that minimizing the loss function of the AE we indirectly maximize a lower bound on the mutual information between the input and its compressed version. This fact is easily generalized to deeper structures with many layers for the encoder and decoder parts. If we think of our original problem of initializing a deep network's weights, the AE provides a conceptually simple but powerful solution: if we set the encoder to be equal to the network we want to train, then the unsupervised pre-training can be carried out by attaching a decoder network at the output of it, creating an AE that can be trained using only the original data and no labels. After the training has converged to a low value of the loss function, the decoder part can be removed and the final fine-tuning process can be done in a supervised fashion. The fact that the lower dimensional representation Y retains most of the information present in X ensures that the weights are initialized in a meaningful manner, as shown from practical demonstrations [28, 39].

The risk with the above method for the parameters initialization, however, lies

[†]For this last case it is typically necessary to use a *sigmoid* activation function or another non-linearity that ensures the bound on the components of $\boldsymbol{x}$ is respected.

in the simple fact that to minimize the loss function, the AE is pushed towards copying the input exactly, which would be useless as it would mean that the network parameters have not learned the high level features of the input but just memorized its value [28]. The fact that we required the compressed representation $\boldsymbol{y}$ to be *undercomplete*, i.e. $d > l$, partially prevents this fact, as in most of the cases $\boldsymbol{y}$ is a *lossy* compression and it is not possible to exactly reconstruct the input from it because some information is lost. One of the most recent and successful techniques in avoiding the AE learning the identity function, however, is the so called *denoising* autoencoder, described in the following section.

3.4.1 Denoising Auto-Encoders

The principle behind a denoising autoencoder (DAE) [28] is to provide as input to the network a corrupted version of the data, that we will call $\tilde{\boldsymbol{x}}$, obtained by artificially applying noise to $\boldsymbol{x}$. The loss at the output of the decoder is however measured between the reconstructed version of the input $\hat{\boldsymbol{x}}$ and the *clean* data $\boldsymbol{x}$. In this way we force the AE not to copy exactly the input, but rather to learn removing the noise we applied to it. Intuitively, by learning to remove noise the network parameters have to focus on important features of the input distribution, that help in replicating the original data and therefore in lowering the loss.

We call the stochastic function that applies noise $e_{\mathcal{D}}(\cdot)$ according to the noise distribution $\mathcal{D}$, the equation describing the DAE are the following

$$\tilde{\boldsymbol{x}} = e_{\mathcal{D}}(\boldsymbol{x}), \tag{3.58}$$

$$\boldsymbol{y} = f_{\theta_e}(\tilde{\boldsymbol{x}}) = a\left(\boldsymbol{W}_e\tilde{\boldsymbol{x}} + \boldsymbol{b}_e\right), \tag{3.59}$$

$$\hat{\boldsymbol{x}} = g_{\theta_d}(\boldsymbol{y}) = a\left(\boldsymbol{W}_d\boldsymbol{y} + \boldsymbol{b}_d\right), \tag{3.60}$$

and the loss function, computed between $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$ can be for example the squared loss $\mathcal{L}_2$ or the cross-entropy loss $\mathcal{L}_H$

$$\mathcal{L}_2(\boldsymbol{x}, \hat{\boldsymbol{x}}) = k||\boldsymbol{x} - \tilde{\boldsymbol{x}}||^2, \tag{3.61}$$

$$\mathcal{L}_H(\boldsymbol{x}, \hat{\boldsymbol{x}}) = -\sum_{i=1}^{d}\left[x_i \log \hat{x}_i + (1 - x_i)\log(1 - \hat{x}_i)\right], \tag{3.62}$$

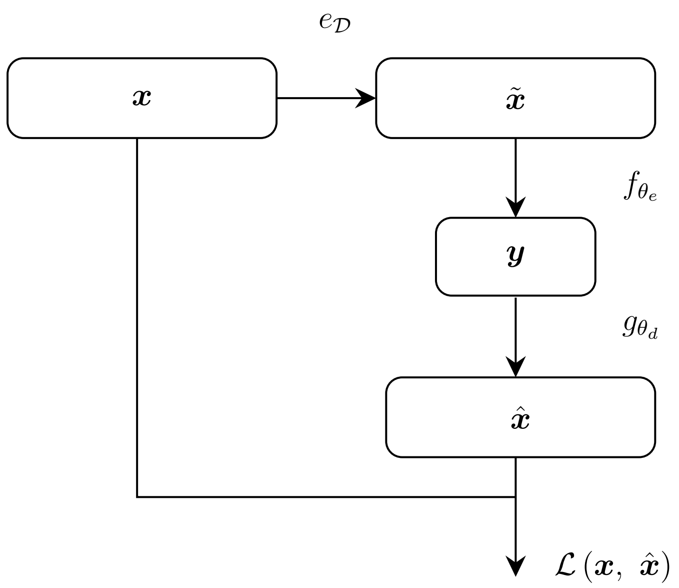


Figure 3.10: Block diagram of the denoising autoencoder architecture. The loss function is computed using the clean data and the reconstructed version from the noisy input.

where k is a constant. The block diagram in Figure 3.4.1 represents the DAE structure more clearly.

The key point that makes the DAE successful is that denoising is only performed as a training procedure to learn high level features of the input: the intermediate representation can be interpreted as a coordinate system for the data points onto a lower dimensional *manifold*, and the DAE learns to map noisy input points back onto the manifold.

Several different types of noise $\mathcal{D}$ have been investigated in the literature; here we will focus on the most used ones:

- *Gaussian noise*: in this case $e_{\mathcal{D}}(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{w}$ where $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$ and σ^2 is the noise variance per component.

- *Salt and pepper noise*: in this case with probability p_{SP} each component of the input is set to its maximum or minimum possible value with probability 0.5 each. As an example, in case the input components are limited to the interval $[0, 1]$, like in the output of a sigmoid activation function, we randomly set to 0 or 1 a percentage equal to p_{SP} of components, on average.

4 Proposed Solution

This chapter describes in detail the proposed signal processing workflow. The overall process is shown in Figure 4.1. The extraction of the gait features from the μ-D spectrogram can be analytically very difficult, and the results are heavily influenced by the environment and hardware non-idealities. Here we propose a framework that exploits CNNs for an automatic feature extraction, after a denoising and successive clustering phase. The raw data obtained at the output of the radar mixer, see Equation (2.18), undergoes the following steps:

1. R-D map computation by double FFT and static objects contribution removal, Section 4.1.
2. Denoising, using KDE to extract the foreground component of the signal, Section 4.2.
3. Clustering with the HDBSCAN algorithm to separate residual noisy peaks from the subject reflection and eventually select the different subject contributions, Section 4.3
4. Classification with CNN on the μ-D spectrogram obtained via integration, Section 4.5.

The human μ-D signature is complex as it contains the superposition of many components due to the different parts of the body moving. Moreover, the high number of noisy reflections in an indoor environment and the hardware imperfections

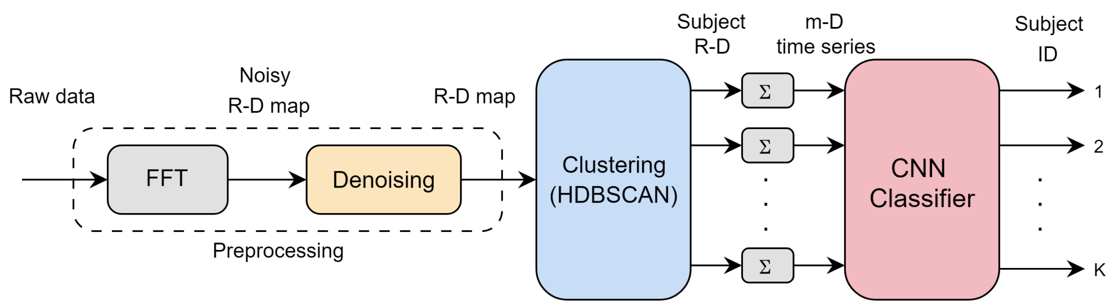


Figure 4.1: Signal processing workflow.

can cause non negligible disturbances to the useful signal which are hard to remove by filtering. For the above reasons, an analytical extraction of the discriminative features that are then fed to a classifier, such as a *Support Vector Machine* (SVM), has been proven to perform worse than automatic feature extraction via deep learning [3, 40]. We follow the latter approach by training a deep CNN for the task of human identification from gait features, introducing a novel workflow for noise removal and clustering of the subject reflections. The computation of the R-D map is needed to localize the subject in the range-Doppler dimension and pre-process the data using this information to improve the classification accuracy. KDE is used to automatically and adaptively provide an estimate of the noise probability density function (PDF) so that the points that belong to the R-D image *background* can be removed at the beginning of the process. This is achieved with a probabilistic threshold, without any knowledge of the reflected power value in presence or absence of a target.

Density-based clustering with HDBSCAN allows to group and filter out from the R-D map any residual noisy peaks that resemble valid targets contributions, caused by random reflections from the environment. In addition, in the case of multiple subjects moving concurrently, clustering is essential to separate the various contributions in order to still be able to perform the identification.

The CNN classifier is the main block of the processing pipeline, that carries out the identification from the μ-D spectrogram. Portions of time-frequency μ-D signal are fed as greyscale images to the CNN that outputs the classification among the various possible subjects. The benefits of unsupervised pre-training are exploited in order to improve the training convergence time and the accuracy [39].

4.1 Range-Doppler Map Extraction

The first block in the pipeline receives as input the radar raw data, i.e. the matrix containing the bi-dimensional signal represented by Equation (2.18). The device outputs one of these signals at every time frame and, according to the discussion in Section 2.3, an FFT for each dimension is computed to obtain the R-D map. To lower the border contributions we applied an additional Hanning window h_0, defined below

$$h_0(n) = \begin{cases} \frac{1}{2}\left[1 - \cos\left(\frac{2\pi}{N-1}n\right)\right], & 0 \leq n \leq N-1 \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

The FFT is performed at first along the fast-time/range dimension and secondly along the slow-time/Doppler dimension. The dimension of the R-D map is dictated by the resolution of the radar device with the transmitted pulse duration and repetition interval. In the range-Doppler space, all static objects are mapped onto the vertical line in correspondence of the 0 velocity value: these reflections are not useful to our scope, so we remove them by cutting the central columns from the R-D map.

4.2 Denoising with KDE

In the computer vision field, extracting the *foreground* component of an image is a core problem, for which many solutions have been designed.

The statistical approach is based on computing an estimate of the background distribution, that in our case is the radar clutter, and then extract the foreground points using a probability threshold or other more structured methods. In [23], the non-parametric method of KDE is used for the first time to this end (see Section 3.1).

To compute the PDF of the power of noisy samples in the background, we apply KDE for each radar frame, refining the estimate with new samples each time. Starting from an initial PDF, as a new frame is available the foreground points are extracted as all the points that have a probability lower than a threshold p^* to be noise. The remaining noisy points are used to refine the PDF estimate using KDE, this new PDF is used on the next frame and so on, iteratively. The procedure is summarized below.

Algorithm 4.1 KDE denoising

```
1: Start from an initial estimate of the noise PDF, $p^{(t)}(x)$.
2: At time frame $t$ receive the corresponding R-D map.
3: Using $p^{(t)}(x)$, extract all the points belonging to the foreground as the points
   with $p^{(t)}(x) < p^*$.
4: Update the PDF estimate for the next timestep $p^{(t+1)}(x)$ with a random selec-
   tion of the background points, using Equation (3.3).
5: Repeat from step 2 indefinitely.
```

Two practical issues had to be solved: the initialization of the PDF in the first step can be done by applying KDE on an empty R-D map where only noise is present; the computational cost of KDE is usually high, so only a random selection of noisy points is used at each step.

The above approach assumes that each noisy pixel is a good representation of a global behavior of the noise, and that it's not necessary to compute an estimate of the PDF *per pixel* as done in [23]. This assumption holds in the case of radar clutter as we will see in Chapter 5, but it's not applicable for foreground detection in the computer vision field.

4.3 Clustering

To effectively perform clustering on each filtered R-D map and identify the subject reflection, we map each point (pixel) of the image to a vector of coordinates $\boldsymbol{x}_i = (r_i, v_i)^T$ where the components are respectively the range and the velocity corresponding to the pixel. Only the points that are marked as belonging to the foreground by the KDE at the previous phase are considered here. HDBSCAN can be executed on the dataset of bi-dimensional points obtained, separating each contribution based on the density: some points of low density are classified as noise and discarded, while a partition of the remaining contributions is provided that we denote with $C_0, C_1, \dots, C_{K-1}$ with K the number of clusters identified. We assume the list of clusters to be ordered in decreasing number of points.

In the case of multiple subjects, in number of N_{sub}, we can assume that the first N_{sub} largest clusters among the K identified correspond to the reflections caused by the target movement. The next step is then to generate N_{sub} different R-D maps where in each only one subject reflection is kept and the others, along with noisy peaks, are removed. Each of these images is then treated as a separate frame, passed to the classifier after integration along the range axis to get the μ-D spectrogram.

For the case of a single target, instead, we can increase the level of precision in the clustering step. We consider the largest cluster in terms of number of points, called C_0, to be the useful reflection of the subject containing most of the information needed about the μ-Doppler signature. In addition, the nearest cluster to C_0 is considered to be part of the person too, because it could be the reflection from an arm or leg that extend outside the main cluster. This second cluster is called C_1, and is only considered is its distance from C_0 is not larger than 1 m (otherwise it hardly represents a part of the subject's body). The distance among clusters considered in this work is the distance between the peak power values of each cluster. Once C_0 and C_1 are identified, only the points belonging to one of the two are retained in the R-D map, canceling all the other contributions. The effect of this operation is to remove from the environment all the peaks that are caused by unwanted reflections or fluctuations, that make the subsequent classification harder and less robust to changes in the environment.

The clustering phase allows to maintain the same classifier architecture for both the cases of single target and multiple targets, operating on the input data in the R-D space. Separation in the μ-D spectrogram would be impossible due to the lack of range information. In addition, we stress that the multiple subjects case includes the single subject one: keeping only the main reflection surely leads to a loss of information, but is a conservative approach in case an a priori decision of which of the two cases we are in can not be made.

4.4 μ-Doppler Spectrogram

The μ-D spectrogram computation is straightforward once the cleaned R-D map is obtained. Figure 4.2 depicts the procedure: each R-D image is transformed into a uni-dimensional vector with as many components as the available Doppler channels by summing all contributions of the different range bins. The resulting vectors are aligned in time to obtain the spectrogram of the μ-D signature.

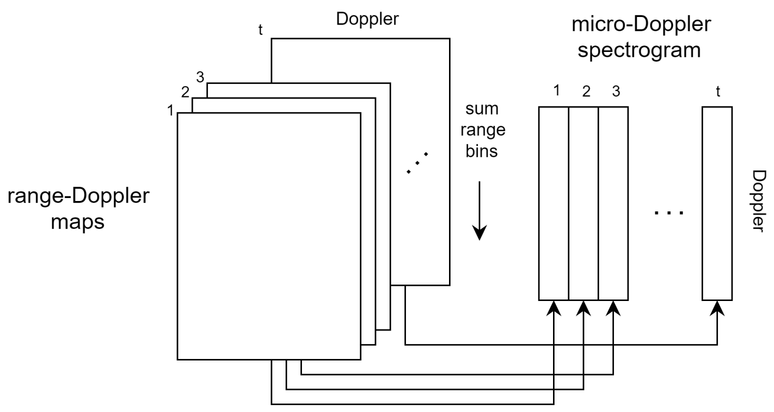


Figure 4.2: Summation over all the range bins to compute the μ-D spectrogram from the time sequence of R-D maps.

4.5 Proposed CNN Classifier Architecture

The proposed classifier architecture is based on *Inception Modules*, a recent CNN structure developed for complex feature extraction in the case of image recognition [41]. The idea behind this approach is to extract features at different scales, using at every layer of the CNN different kernel sizes and concatenate the resulting feature maps. In our case, 1×1, 3×3 and 9×9 filters are applied at each layer, following the idea to extract both small and wide scale characteristics of the μ-D signature that has been proven to be effective in [13]. An efficient implementation of this module is shown in Figure 4.3 on the left. Computationally expensive 3×3 and 9×9 convolutions are preceded by 1×1 convolutions that reduce the dimensionality of the input by lowering the number of feature maps. In this way the number of parameters is kept under control and can be prevented from becoming too large [41]. An additional branch to perform a pooling operation is usually added. The input to the network is a sequence of 45 frames of μ-D vectors, for about 3 seconds of exposition time for the subject. We selected this value to enable direct comparison with [3] on the IDRad dataset (see Section 5.2). Figure 4.3 shows the proposed classifier structure on the right. The number of Doppler channels selected is 208 (see Chapter 5 for the evaluation setup detailed description), so the input image has dimension 208×45. Four inception modules are stacked to extract

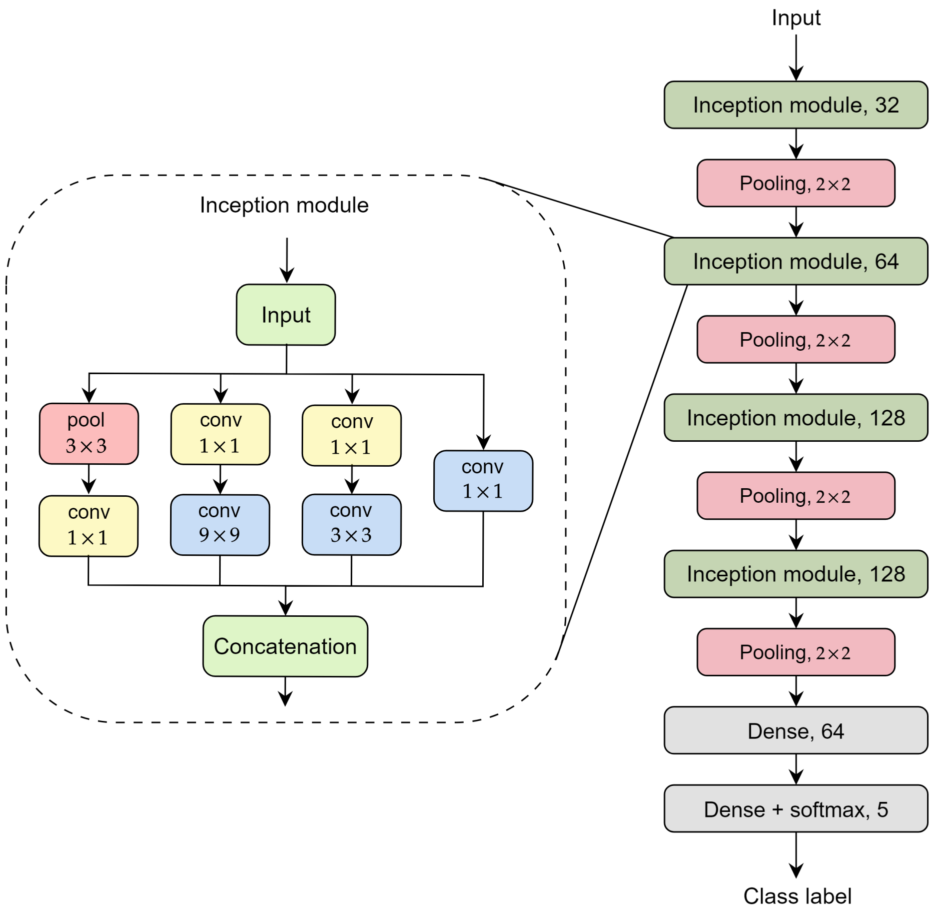


Figure 4.3: Proposed CNN classifier and detail of the dimensionality reduction inception module.

the useful features from the spectrogram, with number of output channels equal to $32, 64, 128, 128$ respectively, each followed by a 2×2 max-pooling operation to reduce the dimensionality. The convolutional layers inside each inception module have all the same number of feature maps, that is $8, 16, 32, 32$ respectively for layers from 1 to 4. This includes also the 1×1 convolutions for dimensionality reduction, so the number of channels is reduced at the input of each layer in order to keep the number of parameters acceptable. Before the output layer, we used a fully-connected layer with 64 units, on which we apply *dropout* to reduce the overfitting on the training data. The output layer applies a *softmax* function and all the nodes use the ELU activation function. The complete classifier CNN of Figure 4.3 has

560,181 parameters.

4.5.1 Pre-Training via Convolutional Denoising Auto-Encoders

In our case, input data is bi-dimensional, so for both encoder and decoder we need to use CNNs, building a *Denoising Convolutional Auto-encoder* (DCAE): we take the proposed classifier network up to the last convolutional layer as the encoder, then the unsupervised pre-training can be carried out by connecting a decoder network at the output of it, creating an auto-encoder structure that can be trained using only the original data. After the training has converged to a low value of the loss function, the decoder part can be removed and the final fine-tuning process can be done in a supervised fashion adding a pooling operation and the last two fully-connected layers. Figure 4.4 shows the complete architecture used for pre-training and classification. The decoder network is implemented as four convolutional layers with $32, 32, 16$ and 1 3×3 filters respectively. The noisy input we used for the training of the auto-encoder was obtained by applying salt and pepper noise to the spectrogram portions. To effectively train the DCAE, given that the input images have values in $[0, 1]$, we use the binary cross-entropy loss function of Equation (3.62) averaged over all pixels of the image.

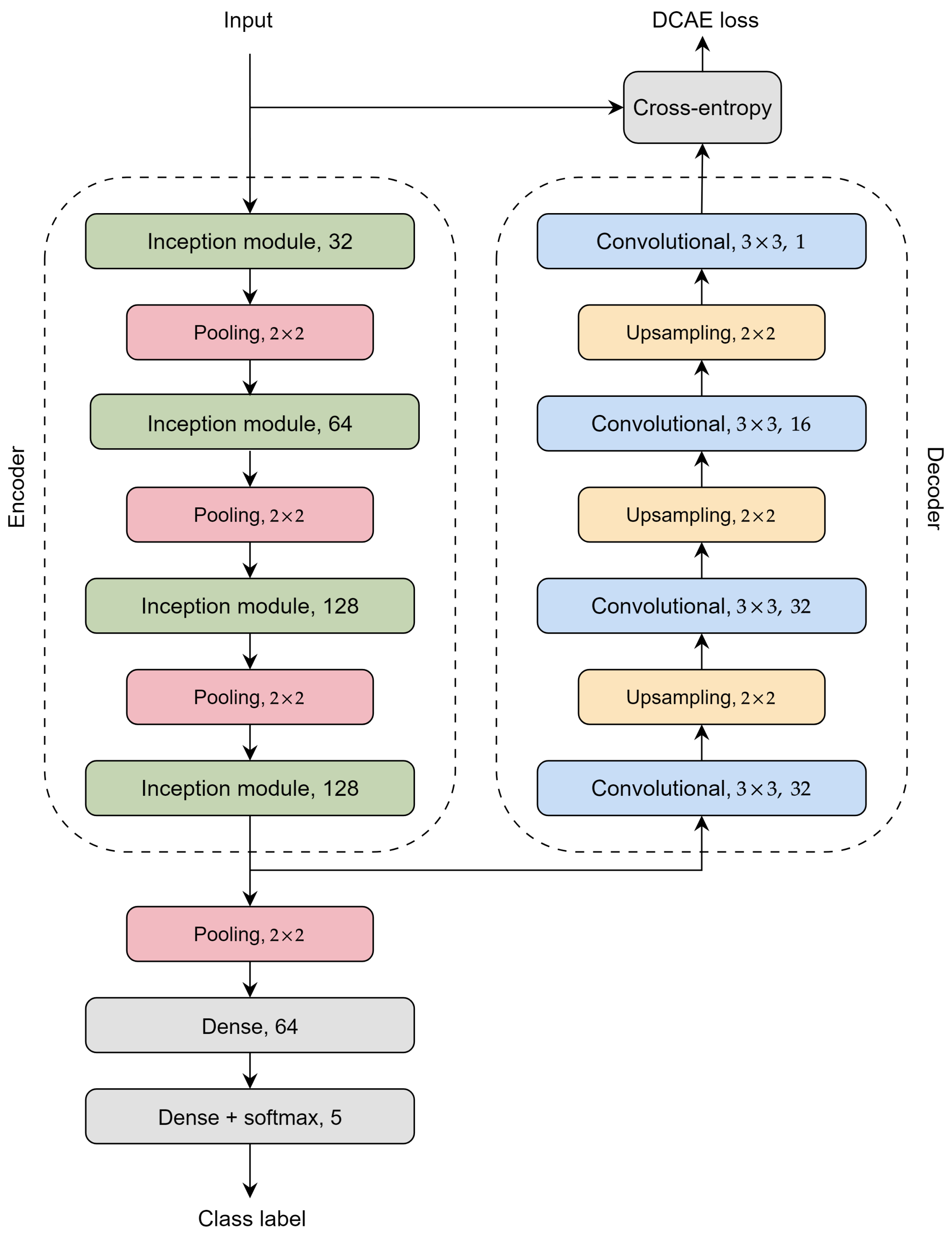


Figure 4.4: Proposed architecture for pre-training via DCAE.

5 Experimental Results

In this chapter a detailed summary of the results obtained with the proposed processing architecture is presented. The evaluation is carried out on the IDRad dataset with respect to the accuracy of the classification provided by our method. Some final considerations on the effectiveness of the proposed pre-processing and clustering are taken on original data measured with an INRAS RadarLog[*] sensor working at 77 GHz. The metrics considered for the analysis are described in Section 5.1, while a description of the dataset is given in Section 5.2. In Section 5.3, Section 5.3.1 and Section 5.4 the results of the phases of pre-processing, denoising and clustering of the R-D maps are presented. Section 5.5 focuses on the *data augmentation* techniques used to enlarge the training dataset for the classifier and on the pre-training phase with the DCAE. In Section 5.6 the performances of the proposed method are evaluated on the test set of the IDRad dataset[†]; comparisons with other classifiers, both independently implemented and taken from the literature are provided. All the implementations are done in Python, using the Numpy, Tensorflow and Keras libraries. Finally, in Section 5.7 the applicability of this method on multi-person data is investigated.

[*]http://www.inras.at/en/products/new-radarlog.html

[†]https://www.imec-int.com/en/IDRad.

5.1 Classification Metrics

In a classification problem the classifier produces labels for each input sample, that have to be compared to the ground-truth labels in order to evaluate its performance on the task. We define the following concepts:

1. True positives (TP): are the samples from class i whose label is correctly predicted as i by the classifier.
2. False positives (FP): are the samples of a class $j \neq i$ that are wrongly classified as belonging to i.
3. True negatives (TN): refer to the samples of a class $j \neq i$ that are correctly not assigned to class i.
4. False negatives (FN): refer to the samples of class i that are instead assigned to class $j \neq i$.

In a classification problem the most intuitive and popular metric is the *accuracy* of the classifier, i.e. the fraction of correctly classified data samples for class i over the total available

$$\text{accuracy}_i = \frac{|\text{TP}|_i + |\text{TN}|_i}{|\text{TP}|_i + |\text{TN}|_i + |\text{FP}|_i + |\text{FN}|_i} \tag{5.1}$$

To this end we use the IDRad test set as a common evaluation ground for the proposed classifier and the possible alternative methods. Given that 5 classes are present in the dataset (see Section 5.2), the average of the accuracy per class $i \in \{1, 2, \ldots, 5\}$ is used. In the case of an unbalanced dataset, where the number of testing samples from each class is quite different, some alternative metrics can be considered, such a *precision*, *recall* and *F1-score*. In our case however the dataset is almost perfectly balanced as it contains the same number of samples for each class, with small differences, so the accuracy is considered to be a good metric for the evaluation.

The *confusion matrix* of the classification labels will be provided, that is a matrix whose rows represent the true classes and whose columns refer to the predicted classes. Each row is then normalized and the result can be expressed in percentage (%). In row i and column j we will then find the percentage of samples from class i that has been assigned to class j by the classifier. High values on the diagonal mean a good classification, giving more insight than just the average accuracy between all the classes.

5.2 IDRad Dataset

The IDRad dataset was made available by researchers from the University of Ghent, that proposed and published a classifier that we use as benchmark in the following [3]. The dataset is also used in some later works that improve over the baseline and provide further analysis [9, 10].

Measurements are taken with 5 different subjects, all male, with weight ranging from 60 kg to 99 kg and height ranging from 178 to 185 cm. Three sets with training, validation and test data are provided to allow a deterministic comparison on the test set, with respectively 100, 25 and 25 minutes of measurements. In addition, training data is recorded in different time moments to reduce the influence of non-discriminating factors such as the clothes wore by the subjects. The radar configuration parameters are listed in Table 5.1, where the ranging and resolution capabilities are computed as in Section 2.3.1.

Measurement parameters	
Center frequency f_c	77 GHz
Chirp bandwidth (B)	1.5 GHz
Chirp duration (T)	256 μs
Sampling frequency	2 GHz
Range resolution (ΔR)	10 cm
Velocity resolution (Δv)	2 cm/s

Table 5.1: Summary of the radar parameters in the IDRad dataset.

The classification is made more challenging by the fact that the measurements are taken in two different rooms for training set and validation/test sets, with respective dimensions equal to 9×6 m and 8×5 m. This fact causes the radar clutter to change significantly in the two cases, making the classifier struggle in non overfitting the training data.

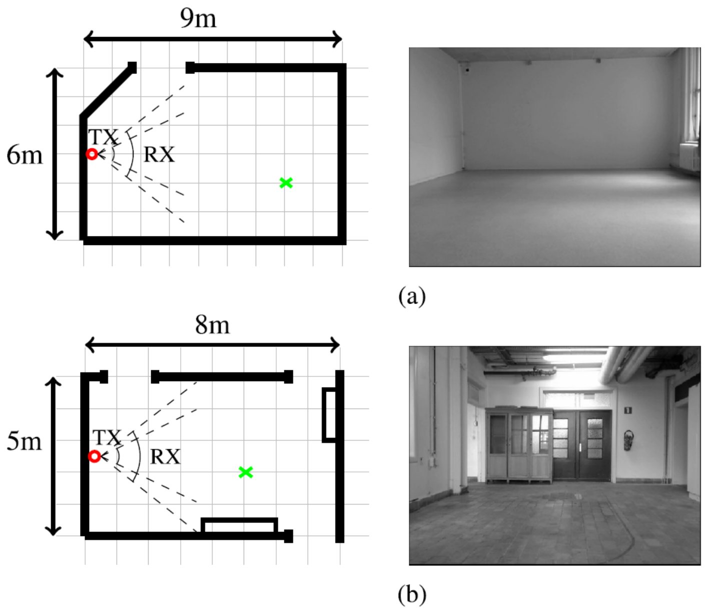


Figure 5.1: Details of the two rooms used to record training and validation/test data. Figure from [3].

The subject is let walk freely in the room in all recordings, simulating a real situation. The dataset provides raw R-D maps and pre-computed μ-D spectrograms obtained from the raw data by applying a threshold at -45 dB to eliminate noise. This approach is motivated by a statistical analysis of the power distribution of the R-D points in the case of an empty room and a room with a person walking. The authors decided to consider as noise all points below the aforementioned threshold, as shown in Figure 5.2. Thresholding works well in the IDRad dataset environment and improves significantly the accuracy of the classifier with respect to the unthresholded case [3].

μ-D spectrograms are obtained by summing the thresholded R-D maps over all range bins and cut into sequences of 45 frames, corresponding roughly to 3 seconds of measurement. The resulting matrices can be interpreted as grayscale images and are fed to the classifier.

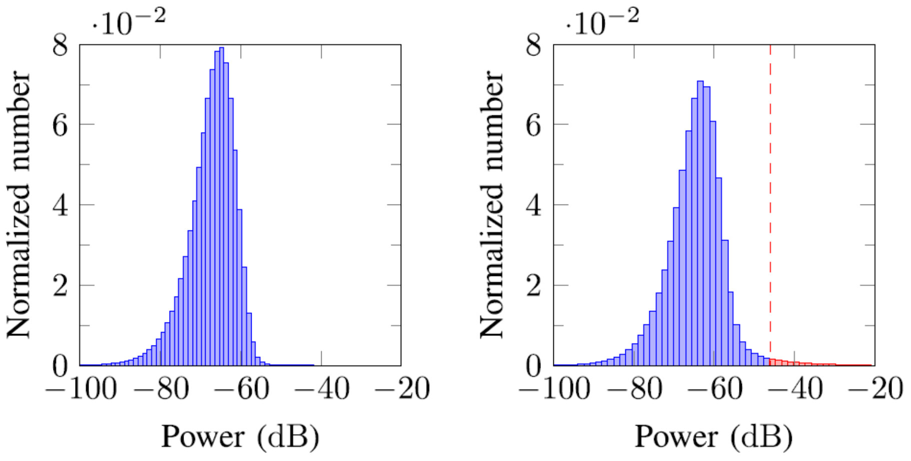


Figure 5.2: Histogram of the power distribution of 1000 R-D maps in an empty room (left) or in a room where a person is walking (right). It can be seen that all points below -45 dB can be considered to be noise. Figure from [3].

In the following, the approach of estimating the noise level adaptively from the data rather than using an offline approach will require the usage of the raw R-D maps rather than the μ-D spectrograms.

5.3 Range-Doppler Pre-Processing

The raw data of the IDRad dataset is passed through a bi-dimensional FFT as described in Section 2.3 to extract the R-D image. The number of channels in the Doppler domain is 256, while the number of range bins is 380, so the overall image dimension is 380×256. The result appears very noisy and presents unwanted peaks due to random reflections of the signal, as shown in Figure 5.3. The pre-processing step is aimed at removing background noise and to discard useless information. This phase is divided into two steps:

1. Removal of the reflections from static objects: as shown in Figure 5.3, static objects are located in the R-D image as an high power peak along the vertical line at 0 m/s velocity. As in indoor environments walls are static and always present, we can remove their unwanted contribution by eliminating the 4 central columns of each R-D map.

2. Denoising by means of KDE as explained in Section 4.2, or by thresholding at -45 dB as done in [3].

The dimension of the image after step 1 is 380 × 252 because 4 columns are removed, as shown in Figure 5.3.

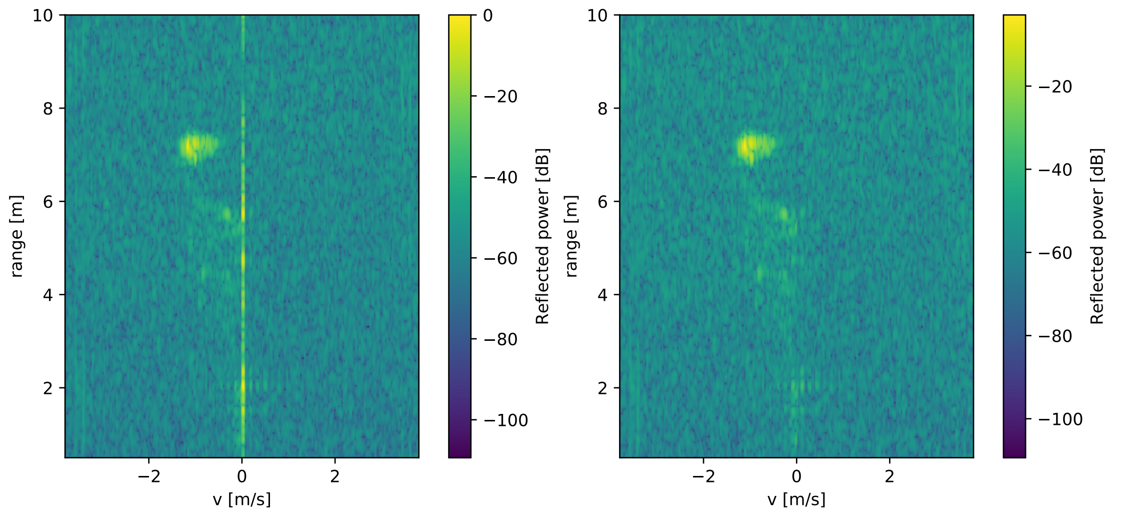


Figure 5.3: Example of raw R-D map (left) and the same map after the removal of static reflections (right).

5.3.1 Noise pdf Estimation

The procedure to estimate and remove the noise from the background of an R-D image is fundamental in the proposed pipeline to allow efficient and meaningful clustering in the following step, in addition to significantly reduce the complexity of the classification as shown in [3, 10]. Here we visually show and compare the results of using the proposed KDE estimation method with the static threshold applied in the cited works.

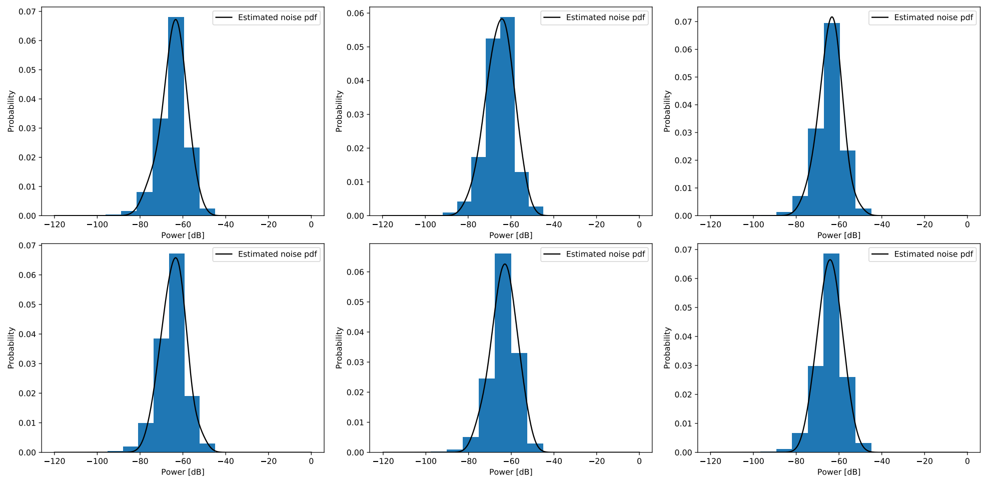


Figure 5.4: Noise pdf obtained from KDE in 6 subsequent range-Doppler frames.

Figure 5.4 presents an example of how the adaptive estimation is able to follow small changes in the noise distribution. The black curve is the PDF estimated from a random selection of N points classified as noise based on the previous timestep PDF. In particular, at timestep t a point x is classified as noise if it has a probability $p^{(t-1)}(x)$ greater than p^* to be noise according to the pdf at time $t-1$, as stated in algorithm 4.1.

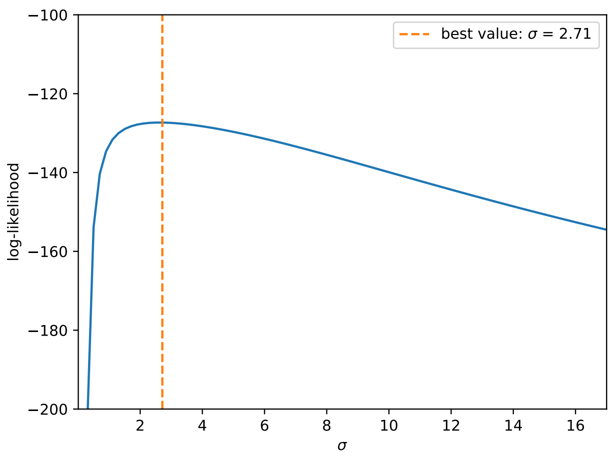


Figure 5.5: Log-likelihood obtained from 5-fold cross-validation on the selected points for KDE, as a function of the bandwidth σ, for the current model.

The reference value represented by the histogram is instead the distribution of noise according to the threshold at -45 dB using *all the image noisy points*: we can see than the adaptive estimation matches correctly the underlying noise PDF in all six frames. This fact is interesting for two reasons:

1. We are not using all the noisy pixels available at each frame but only N randomly selected. This greatly enhances the computational speed of the algorithm but in general could provide a bad estimate of the real PDF. The fact that a small selection of points is enough to approximate the PDF means that the noise has poor correlation in the range-Doppler space.

2. We are not setting a fixed threshold on the received power, that could change if a different room is used for tests or even in the same room during time. This allows more flexibility in the denoising procedure increasing the generality of the approach to other datasets than IDRad. The chosen threshold on the probability of belonging to the background is general and motivated by the desired false alarm probability rather than from empirical considerations as the fixed threshold is.

Adaptivity however comes with the drawback of being subject to an unlucky choice of the N samples or to brief modifications of the noise PDF that have an effect on subsequent frames. Figure 5.6 shows the result of denoising with KDE in a critical case where the noise distribution is not estimated precisely and many noisy points from the background are retained in the final image (the sparse light-blue dots on the background). However, even in these cases, we will show that the combination of KDE with clustering can mitigate the problem and lead to results almost identical to the ones of the threshold method. The parameters to chose in this pre-processing steps are the probabilistic threshold p^*, the number of samples N and the bandwidth of the Gaussian KDE kernel function σ. The latter was chosen using *5-fold cross-validation* considering the *log-likelihood* of the available data under several possible choices, obtaining a best value of $\sigma = 2.71$ dB (see Figure 5.5), while the procedure showed an high robustness to any value of N over 50-100 points. The probabilistic threshold is instead set to $p^* = 5 \times 10^{-3}$, but any value in the same order of magnitude has shown to yield the same performance.

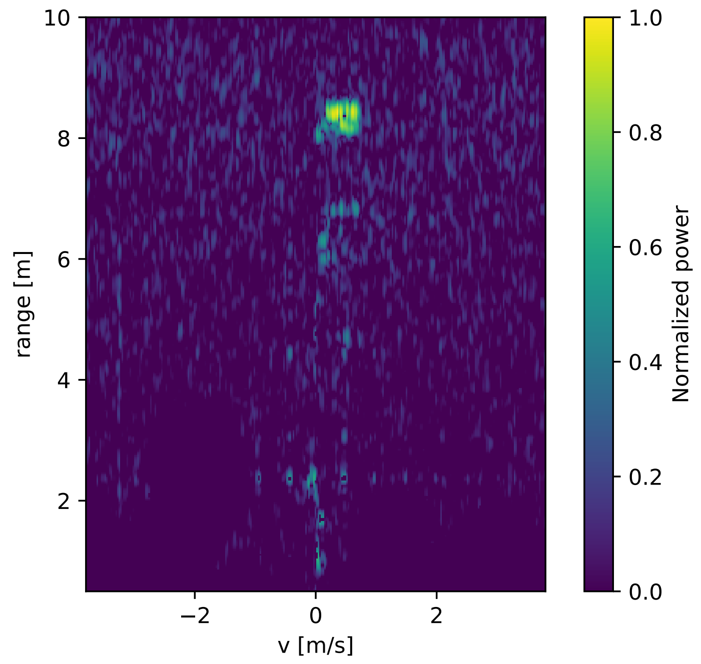


Figure 5.6: Example results of the denoising procedure with KDE.

5.4 Clustering Results

Clustering is performed in the R-D space to remove the additional noisy power peaks whose intensity is unusually strong with respect to the noise distribution. In indoor environments this kind of unwanted reflections are frequent and need to be filtered out from the image. The set of range-Doppler coordinates of the foreground points is therefore fed to the HDBSCAN algorithm in order to identify a partition of such points. Partitioning on the based of density is crucial for the purpose of removing this kind of residual noise: points from the background that have not been removed by KDE are sparse, so HDBSCAN easily identifies them as noise (i.e. they are not part of any selected cluster), moreover, high power peaks are densely grouped in specific positions so they are also easily separable from the main cluster most of the times. The only parameter that needs to be set is the minimum number of points that can make a cluster, n_{mpts}. Assuming that the biggest cluster is made by the reflection from the subject, a trade-off has to be considered between excluding as many small noisy peaks as possible and not missing the main cluster when it is small due to disturbances in the received signal.

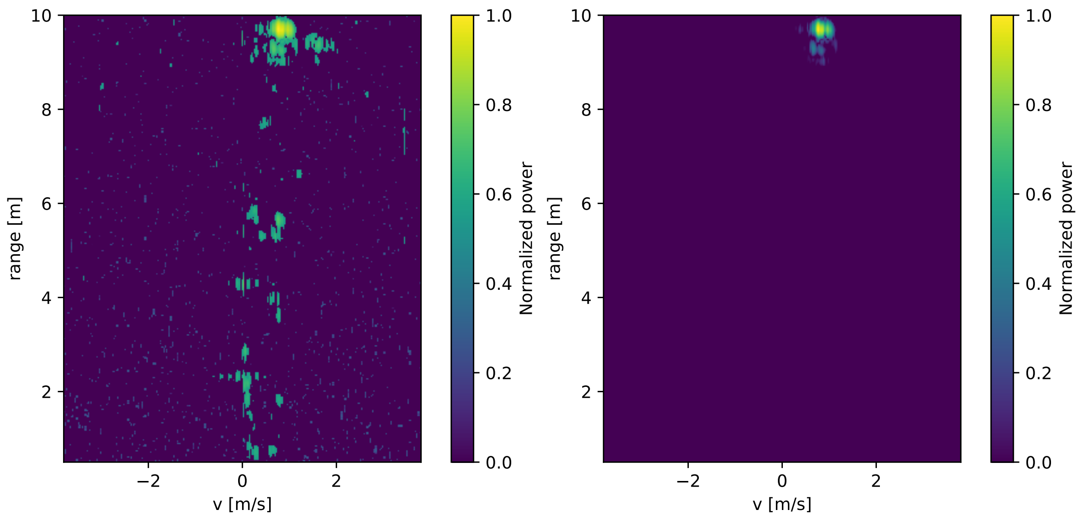


Figure 5.7: Denoised R-D map (left) and the same map after clustering and cluster selection is applied (right).

$n_{mpts} = 25$ was chosen as a good value, considering that the main reflection has on average from 300 to 1000 points, while the unwanted peaks carry much smaller values around 10-50. Figure 5.7 and Figure 5.8 show the effect of changing the value for this parameter on the clusters detected by HDBSCAN. The first image represents the denoised R-D map, to which clustering is applied, and the resulting output after the removal of unwanted clusters is performed. The following images show the output of HDBSCAN with $n_{mpts} = 10, 25, 50, 100, 200$ respectively, where different colors correspond to different clusters: we can see how increasing the value of the parameter leads to detecting fewer and bigger clusters, in some cases even when the resulting partition is not very meaningful from a visual point of view.

Figure 5.9 shows the result of applying the denoising and clustering steps one after the other. The final result is clean and contains only the interesting reflection from the subject, without any additional peaks or clutter. This is achieved by combining the clutter removal capabilities of foreground estimation with density based clustering that covers the weakness of KDE in removing sparse peaks.

After clustering is performed, summation over the columns of each R-D map gives the μ-D vector for each frame. The additional Doppler channels from 1 to 24 and from 228 to 252 were removed as non-informative: they showed not to contain any signal component in all of the images. The resulting number of Doppler channels

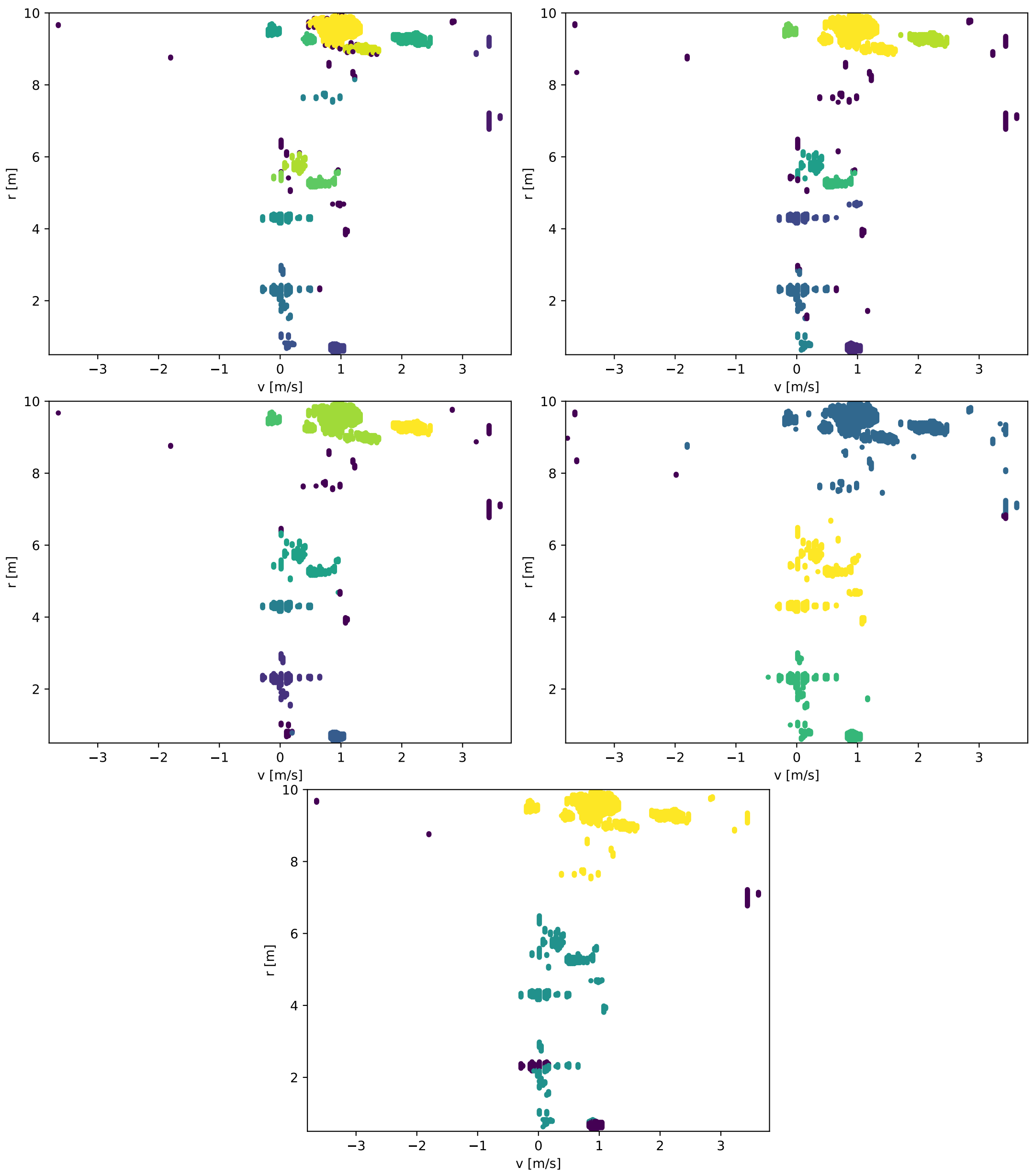


Figure 5.8: Effect of varying the parameter n_{mpts} on the HDBSCAN clustering results.

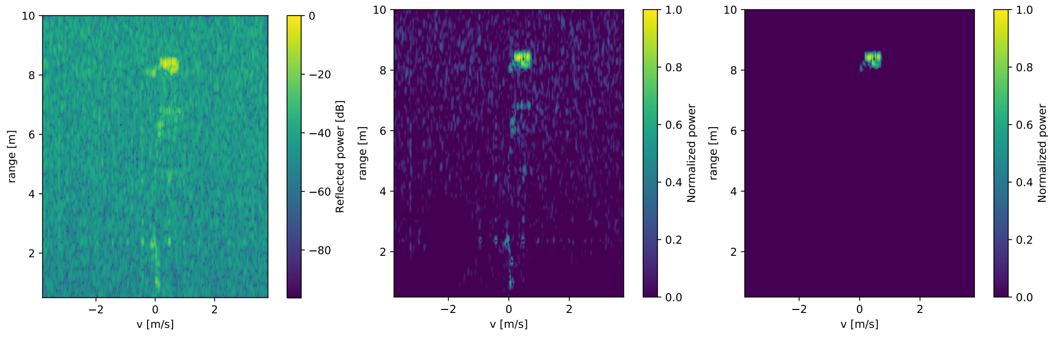


Figure 5.9: Results of the pre-processing steps: raw data, effect of denoising, effect of clustering.

considered from here on is 204. Figure 5.10 shows examples of the spectrograms for each user obtained by stacking the μ-D vectors along time. The distinction between the different users is very difficult or impossible to do for a human being, as the features that characterize the subject are complex and hidden in the signal. At this step the data to be fed to the classifier is cut into windows of 45 samples, corresponding to about 3 seconds of measurement, resulting in input images of shape 204×45.

5.5 Data Augmentation and Training

The scarceness of training data is a severe problem in deep learning. Complex neural network architectures with hundreds of thousands of tunable parameters need large amounts of training samples to converge to good results. The method of pre-training via unsupervised learning is used here to simplify the initialization of the network parameters. To this end two different training phases are needed:

1. Unsupervised pre-training of the inception modules with the DCAE described in Section 4.5.

2. Training of the fully connected layer and fine-tuning of the inception modules on the labeled data.

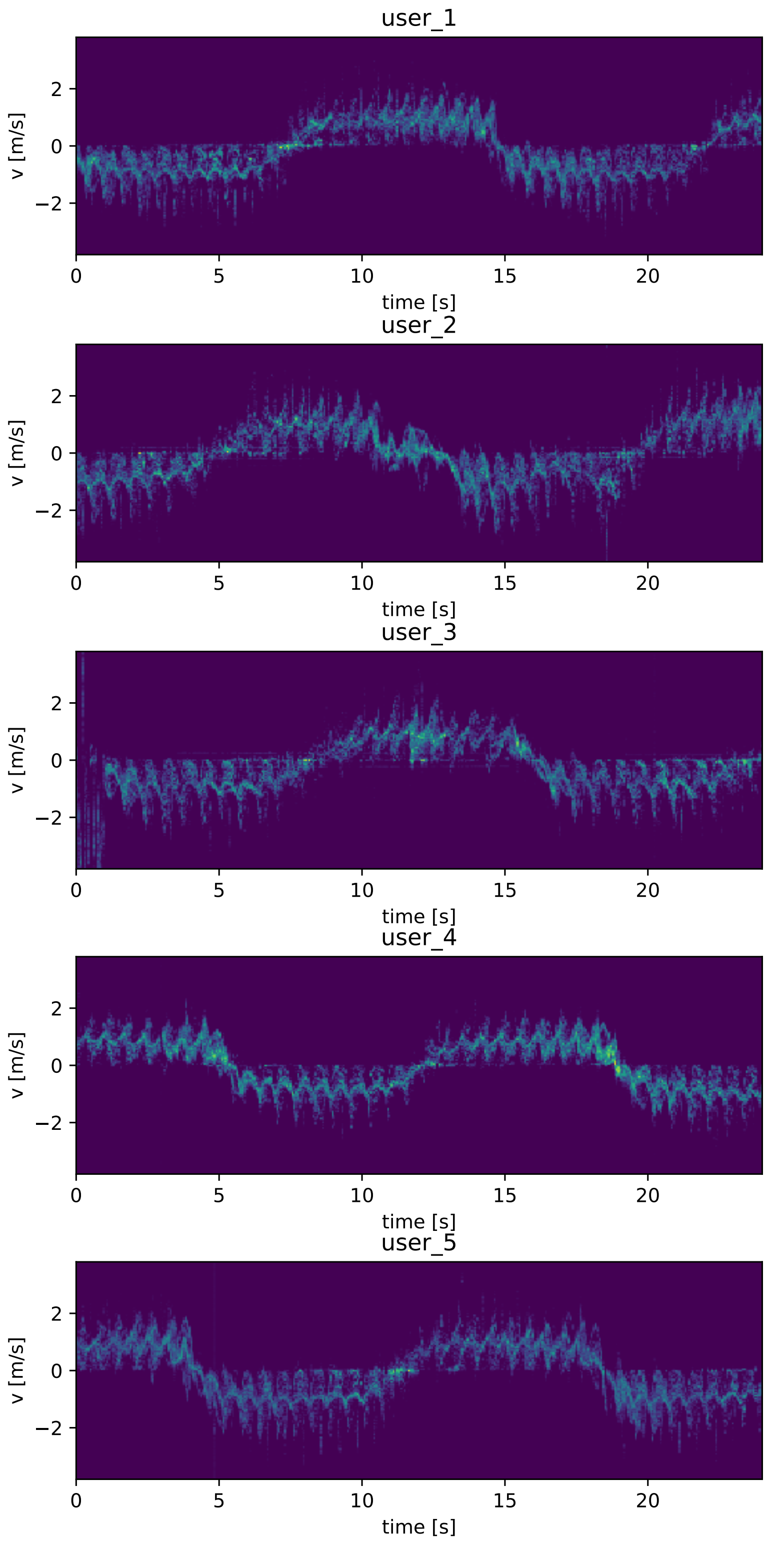


Figure 5.10: Examples of the μ-D signature of each user after the clustering step, data is taken from the test set.

IM classifier final parameters	
Optimizer	Adadelta
Loss function	Cross-entropy
Regularization param. λ	3×10^{-3}
Dropout	0.5
Conv. filter sizes	1×1, 3×3, 9×9
Number of Inc. Modules	4
Pooling size	2×2

Table 5.2: Summary of the parameters used in the fine-tuning of the classifier.

DCAE final parameters	
Optimizer	Adadelta
Loss function	Binary cross-entropy
IM conv. filter sizes	1×1, 3×3, 9×9
Encoder number of Inc. Modules	4
Pooling size	2×2
Decoder number of layers	4
Decoder filter sizes	3×3
Decoder feature maps	8, 16, 32, 32

Table 5.3: Summary of the parameters used in the pre-training of the classifier via DCAE.

5.5.1 Unsupervised Pre-Training

For the first step, a noisy version of the training set data has to be generated, in order to provide it as input to the denoising auto-encoder, while the ground-truth labels are the original training data. Salt and pepper noise is used, setting a probability of $p_{SP} = 0.2$ of corrupting a pixel that will be set to 0 or to 1 with equal probability. Figure 5.12 shows the effect of this procedure on a clean training image. The unsupervised learning aimed at the reconstruction of the original data involves the architecture of Figure 4.4, with the parameters listed in the following tables.

The choice of the loss-function was dictated by considerations on the quality of the training convergence and on the final result. We performed training under the two alternatives of MSE and binary cross-entropy, described in Section 3.4.1, discovering that MSE leads to an uncertain outcome depending on the initial configuration of the weights. In most cases, training under MSE minimization led to the optimizer getting stuck after 1-2 epochs without any further decrease of the loss. Only in a few cases the training ended with a good convergence of the loss, meaning that the combination of the Adadelta optimizer with MSE in this case is not enough robust to the initial choice of the weights. On the other hand, binary cross-entropy led to much better training results, converging always to a minimum under any starting point for the weights. For this reason in the final pre-training of the classifier we trained the DCAE with the the binary cross-entropy loss. The behavior of the training and validation loss in this case is shown in Figure 5.11: overfitting is not an issue as proved by the fact that the validation loss continues decreasing along with the training loss, and a good convergence is achieved after 200-250 epochs.

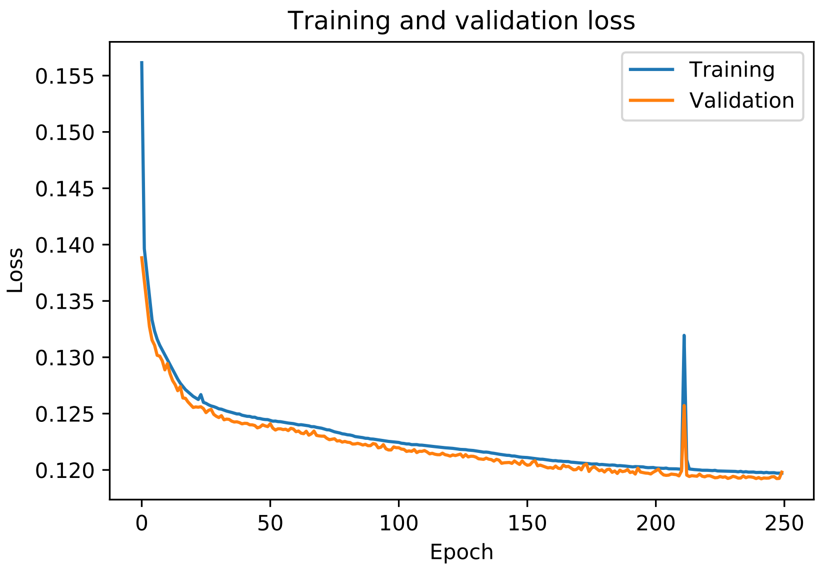


Figure 5.11: Training and validation loss curve during training of the DCAE.

For our scope, however, training speed is preferred over a low value of the loss function, as the auto-encoder only serves as an initializer for the weights of the classifier. For this reason we experimented by training the classifier with a different number of epochs in the pre-training phase and discovered that after epoch 20 the performance remained approximately the same, showing that the full training of the DCAE is not needed to perform in order to obtain a good initialization of the classifier parameters. In the final training phase we chose 30 pre-training epochs for the above reason.

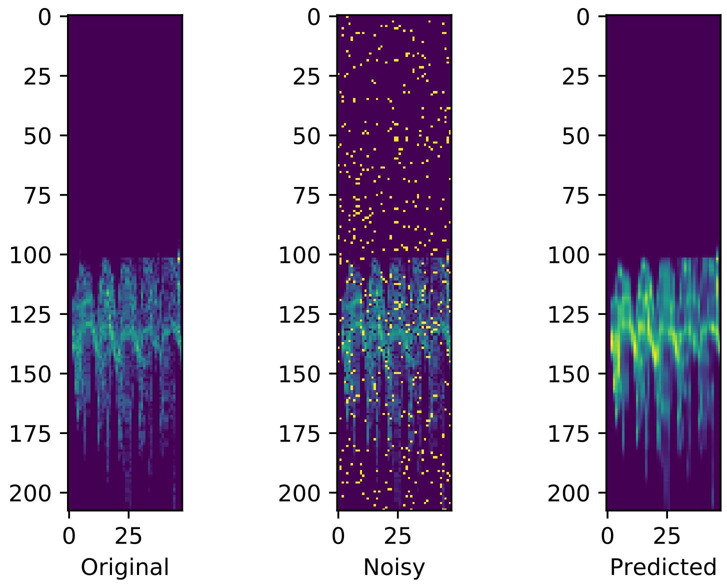


Figure 5.12: Results of the trained DCAE on the test set. The network is able to reconstruct an image (on the right) similar to the original one (on the left) from a corrupted version (in the middle).

Figure 5.12 shows the results of the DCAE training. We feed a test image that was not used for training to the network, after adding salt and pepper noise, and the output appears very similar to the original image. In addition the result is also much smoother and regular than the original version, that carries some residual noise from the pre-processing phase, meaning that the network can reconstruct the *shape* of the waveform and has learned important features of the input to do so. The shape of the input images to the auto-encoder is 208×48 after applying zero-padding to match the network required dimensions.

5.5.2 Fine Tuning

After unsupervised pre-training, the weights of the convolutional layers of the classifier are initialized in a meaningful manner, so fine tuning on the labeled data for classification can be performed. As described in Section 4.5, two fully connected layers are applied to the output of the last inception module. To reduce the overfitting problem, we apply dropout with probability 0.5 to the first of such layers, and an overall L_2 regularization with parameter $\lambda = 3 \times 10^{-3}$. Due to the non-homogeneity of the dataset, overfitting was a severe problem in the network training. For this reason also data augmentation had to be applied in order to reduce it. We used the addition of Gaussian noise to each pixel of the μ-D signal and random sampling along the time dimension:

1. Random sampling in time is applied by selecting randomly a set of images from the long spectrogram sequence for each user obtained after clustering and summation along the range dimension. This allows to pick an arbitrary number of images from a finite spectrogram, however extracting too many samples from the same sequence may end up in repeating the same data.

2. Gaussian noise is added creating a noisy version of each training image, hence doubling the dataset. The chosen value for the artificial noise standard deviation is $\sigma = 0.05$, acting on normalized images with values in $[0, 1]$.

The training data after augmentation is made of 20000 204×45 samples, 4000 per user.

Training was carried out for 50 epochs (see Figure 5.13), noticing two important facts: convergence is rapid, thanks to the pre-training procedure, but overfitting still appears to be present even after the above mentioned technique were applied, as testified by the gap between the training and validation losses. However, comparing our work to [3, 9] it seems to be less intense as our network retains better general information about the input even without seeing data gathered from the validation/test room.

The cause of overfitting is confirmed to be the fact that training and validation sets are taken in different rooms, as training on the union of the two sets caused the network not to overfit anymore as shown in the training/validation loss plot of Figure 5.13 on the right. The hyperparameters such as the regularization rate λ, the dropout probability, the choice of the optimizer and the filter sizes were chosen with a simple grid search approach of which we do not report the full results for conciseness. Grid search consists in selecting a uni-dimensional grid of values for each parameter, then training a separate network for each combination of the possible values on a multi-dimensional grid obtained by combining the grids of each parameter. It is a computationally demanding procedure when the number of hyperparameters is high, however it was affordable in our case due to the fast convergence of the network training.

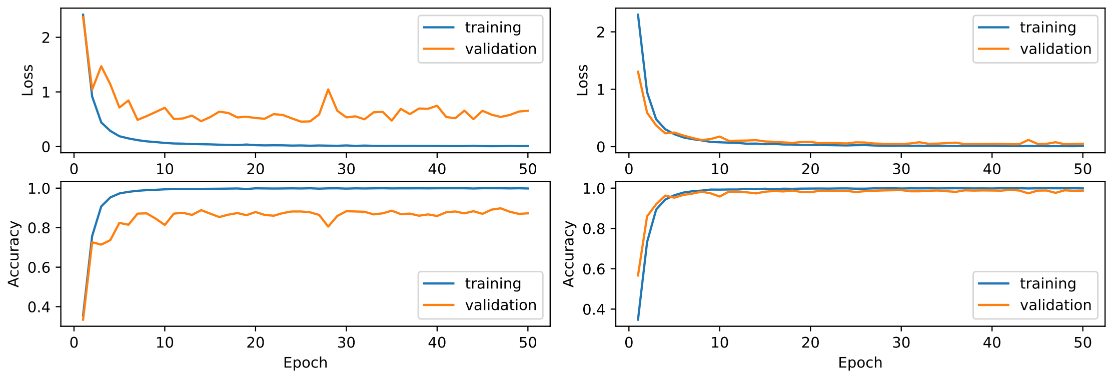


Figure 5.13: Loss and accuracy behavior in the standard case (left) and when the training and validation sets are mixed before training (right).

The final selection of the best parameter values is done by comparing the results of grid search on the validation set. The number of inception modules and layers of the decoder in the DCAE were instead selected separately choosing the value that led to the best performance on validation, without considering the relation with other parameters.

5.6 Performance Analysis

The results of the proposed architecture on the classification task are evaluated on the IDRad test set. We implemented several other models as a comparison, the models from the literature are indicated with the corresponding paper citation:

1. A baseline simple CNN to evaluate the effect of the clustering phase on accuracy. The network is made of 4 convolutional layers separated by 2×2 max pooling. The filter size is 3×3 for all layers and we have $8, 16, 32, 64$ feature maps at each layer respectively. The final layers are fully connected and dropout with probability 0.5 is applied on the first one.

2. A Recurrent Neural Network (RNN) based on bidirectional Gated Recurrent Units (GRU) layers used as comparison. We do not discuss the theory of

GRU layers or RNNs here and refer to the original paper [43]. This model is denoted by GRU in the following.

The structure of the network is a two GRU layer architecture with 128 units each. Both layers are *bi-directional*, meaning that the input is processed both in the forward time direction and in the backward direction. The images of 45 frames with the μ-D spectrograms are split into the single frames vector and fed to the RNN sequentially as a time series. Figure 5.14 shows the just described structure as a block diagram.

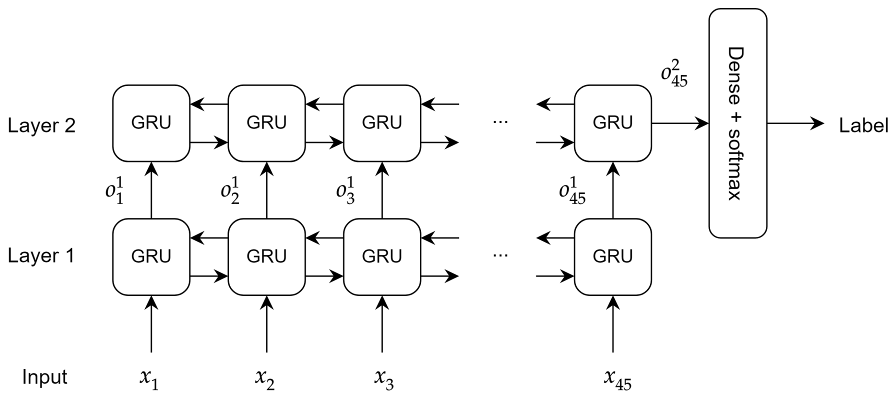


Figure 5.14: RNN structure used as comparison. The diagram shows the time-unfolded architecture, where input vectors from 1 to 45 are sequentially processed in a bi-directional fashion.

3. The base version of the IM-based classifier presented in Section 4.5, without pre-training with the auto-encoder, called IM.

4. The final pre-trained IM model, denoted by pre-IM.

In Table 5.4 and Table 5.5 we present the results, in terms of accuracy on the validation and test sets of the IDRad dataset, of the above models, compared with the results from the literature. In the first table we compare the performance of our models when the denoising phase is performed with the thresholding at -45 dB on the R-D map, so no KDE is applied, to allow a direct comparison of the combination of clustering and the proposed classifier with [3, 10]. The pre-trained IM models outperforms all the other ones, including the combination of the generative model and RNN from [10] thanks to its ability to extract very complex features from the data. The idea of considering different filter sizes to extract information at different

	Accuracy on IDRad [%]	
Classifier	Validation	Test
RCN [9] (raw data)	73.54	75.65
CNN	73.02	70.46
GRU	73.15	75.83
pre-IM	83.43	84.30

Table 5.5: Accuracy obtained on IDRad validation and test set when KDE is used for denoising.

scales has proven to be of key importance, bringing a large improvement over the baseline CNN that makes use of 3×3 filters only. The result of the CNN is also important to highlight that the clustering phase has an impact on the accuracy in that it removes noisy contributions from the image that can mislead the classifier. This is shown by the improvement of our CNN model with respect to the one in [3].

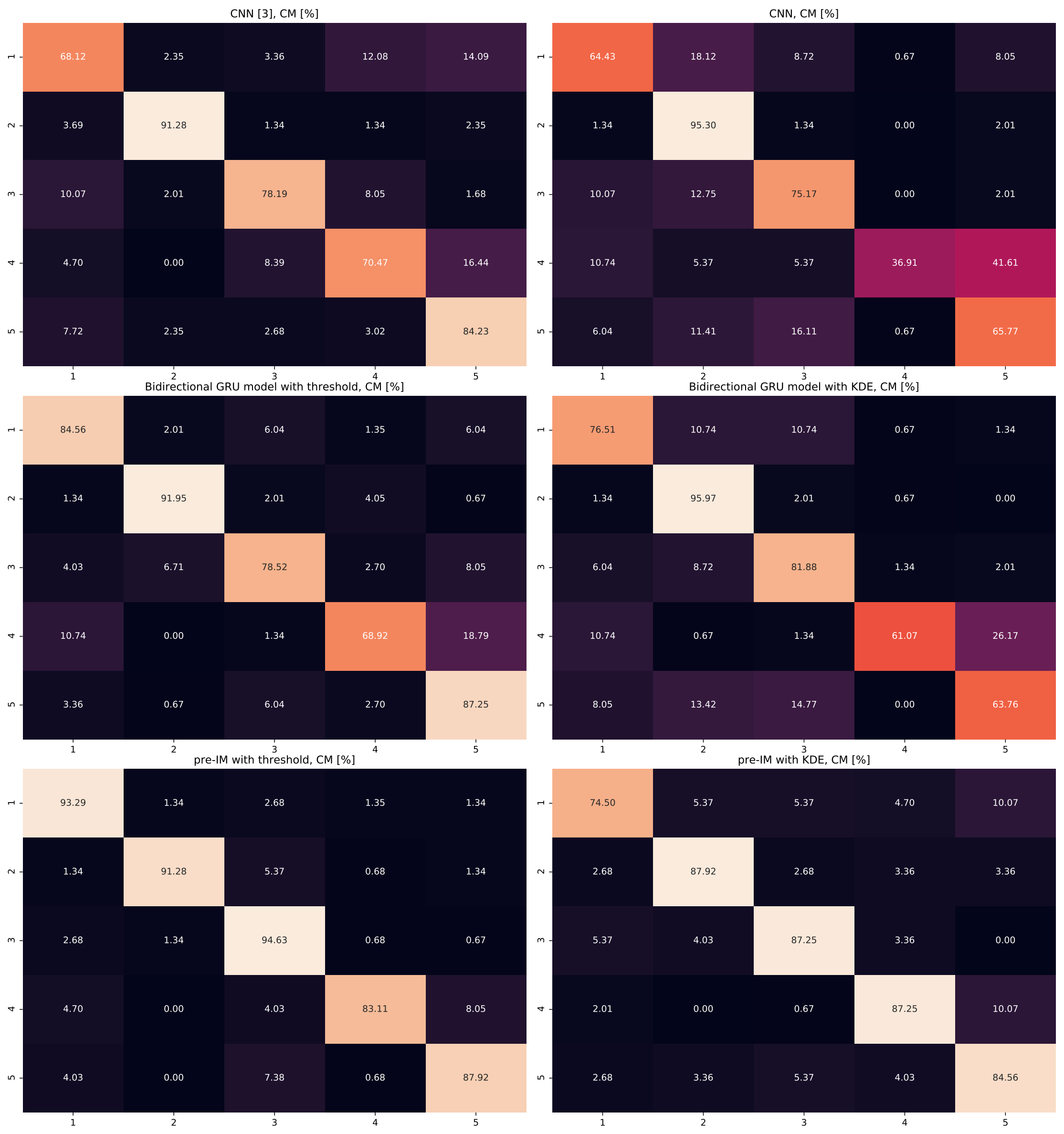
	Accuracy on IDRad [%]	
Classifier	Validation	Test
CNN [3]	75.30	78.46
RNN [10]	88.08	89.56
CNN	77.37	82.26
GRU	79.46	82.25
IM	90.06	88.44
pre-IM	87.65	90.05

Table 5.4: Accuracy values obtained on IDRad validation and test sets by algorithms from the literature and the ones proposed here when thresholding is used as denoising.

The GRU model instead does not lead to good results, especially compared to the other RNN-based approach in [10]. This can be due to the fact that the GRU layers are implemented using fully-connected layers. This means that the spatial structure of the single μ-D vector is not considered as a characteristic of the input, only the temporal relation is accounted for. This issue is solved when a generative model that provides hidden states for the gait process as input to the classifier is used, as spatial information may not be important is such states. In Table 5.5 instead the denoising with KDE is applied to all the implemented models instead of the threshold. The comparison is made directly with those models in the literature that operate on the *raw* data [9], as our denoising procedure does not require any setup or pre-processing. The overall results appear to be worse than in the thresholded case, but still significant considering that no information on the background noise has to be provided to the algorithm and that the estimation is adaptive. The RCN implementation in [9] can handle the raw data but its overall accuracy is worse than the proposed model. However, in terms of training speed the RCN is significantly faster as it does not require an iterative backpropagation algorithm but only a matrix inversion.

In order to gain better insight on how the classifier operates on the 5 subjects, confusion matrices are computed for each model (Figure 5.15). Again, KDE has an evident negative impact on the accuracy. In practical applications, however, it may still be preferable due to its adaptability, depending on how much information about the noise distribution is available or on how much the noise level is supposed to change during time.

From an analysis of the confusion matrices one can notice the following problems of the model proposed in [3]: class 1 has a low accuracy, and many samples form this class are wrongly assigned to classes 4 and 5, class 4 is quite often confused with class 5. Both these problems are solved by our pre-IM model, that presents a much more robust accuracy over all classes in case thresholding is used as denoising. With KDE instead class 1 remains the hardest to classify correctly, as 10% of its samples are labeled as class 5. The GRU models instead have interesting capabilities but still are not able to provide the performances obtained by inception modules and especially in the case with KDE give insufficient accuracy values for classes 4 and 5.



CNN [3], CM [%]

	1	2	3	4	5
1	68.12	2.35	3.36	12.08	14.09
2	3.69	91.28	1.34	1.34	2.35
3	10.07	2.01	78.19	8.05	1.68
4	4.70	0.00	8.39	70.47	16.44
5	7.72	2.35	2.68	3.02	84.23

CNN, CM [%]

	1	2	3	4	5
1	64.43	18.12	8.72	0.67	8.05
2	1.34	95.30	1.34	0.00	2.01
3	10.07	12.75	75.17	0.00	2.01
4	10.74	5.37	5.37	36.91	41.61
5	6.04	11.41	16.11	0.67	65.77

Bidirectional GRU model with threshold, CM [%]

	1	2	3	4	5
1	84.56	2.01	6.04	1.35	6.04
2	1.34	91.95	2.01	4.05	0.67
3	4.03	6.71	78.52	2.70	8.05
4	10.74	0.00	1.34	68.92	18.79
5	3.36	0.67	6.04	2.70	87.25

Bidirectional GRU model with KDE, CM [%]

	1	2	3	4	5
1	76.51	10.74	10.74	0.67	1.34
2	1.34	95.97	2.01	0.67	0.00
3	6.04	8.72	81.88	1.34	2.01
4	10.74	0.67	1.34	61.07	26.17
5	8.05	13.42	14.77	0.00	63.76

pre-IM with threshold, CM [%]

	1	2	3	4	5
1	93.29	1.34	2.68	1.35	1.34
2	1.34	91.28	5.37	0.68	1.34
3	2.68	1.34	94.63	0.68	0.67
4	4.70	0.00	4.03	83.11	8.05
5	4.03	0.00	7.38	0.68	87.92

pre-IM with KDE, CM [%]

	1	2	3	4	5
1	74.50	5.37	5.37	4.70	10.07
2	2.68	87.92	2.68	3.36	3.36
3	5.37	4.03	87.25	3.36	0.00
4	2.01	0.00	0.67	87.25	10.07
5	2.68	3.36	5.37	4.03	84.56

Figure 5.15: Confusion matrices for different classifiers: the rows correspond to the true labels, while the columns are the predicted labels.

5.7 Considerations on Multi-Person Identification

The clustering phase of the proposed processing pipeline allows not only to separate high-power noisy reflections from the contribution of the subject, but also to handle the case where multiple subjects are walking in the room at the same time.

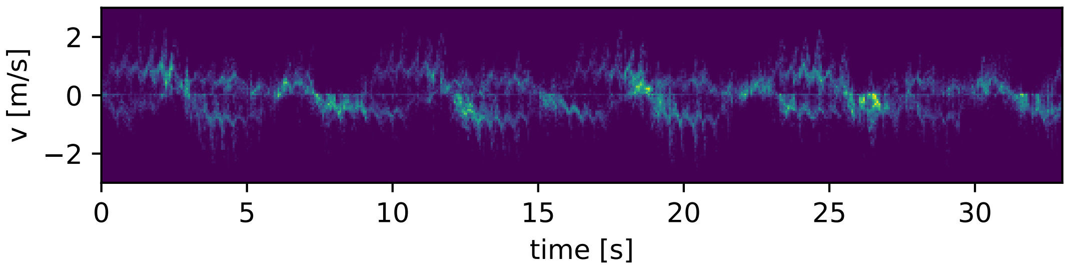


Figure 5.16: μ-D signature obtained from a scenario where 2 subjects are present in the room at the same time.

Measurement parameters	
Center frequency (f_c)	77 GHz
Chirp bandwidth (B)	2 GHz
Chirp duration (T)	320 μs
Sampling frequency	2 GHz
PRI (T_{REP})	500 μs
Chirps per sequence	256
Sequence rep. interval	130 ms

Table 5.6: Summary of the radar parameters used for the multi-subject measurements.

Figure 5.16 shows the μ-D spectrum of the measurement when 2 targets are present, obtained from measurements done with an INRAS RadarLog device working at center frequency 77 GHz. In the simulation, $K = 2$ subjects are let walking randomly in a room, which is not completely empty from furniture, increasing the complexity of the task. The radar parameters used are listed in Table 5.6. Clearly, to distinguish the different subjects contribution in the μ-D space is difficult, therefore we need to perform the separation in the R-D map. In this case, following the description of Section 4.3, we need to select the K biggest clusters from the R-D map and interpret them as the reflections from K different subjects. The single R-D image then can be split into K different images and integrated along the rage dimension to obtain μ-D signatures for each subject. At this point the μ-D sequences can be fed to the classifier as in the single-target case with no modifications. To prove that such a separation in the R-D space is possible we tested it with the HDBSCAN algorithm on the new measurements.

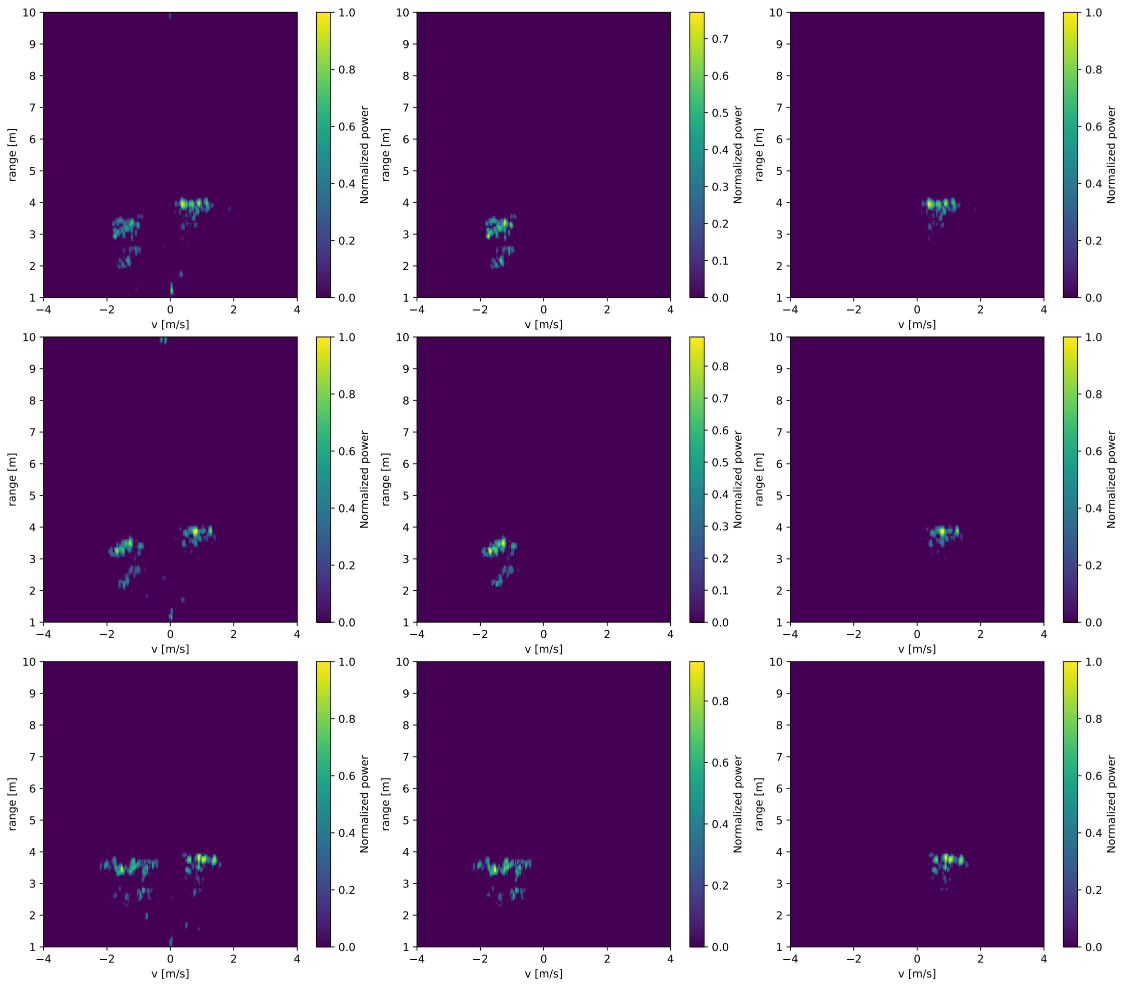


Figure 5.17: Clustering with HDBSCAN in the case of $K = 2$ subjects. The original R-D map (left) shows the two contributions of the subjects, while the separated clusters (center and right) contain only the information about the specific target.

As we focus on a proof-of-concept that the clustering phase can separate multiple targets we do not apply KDE denoising here, instead a threshold at -45 dB is chosen. HDBSCAN is applied with $n_{mpts} = 25$, selecting the two biggest obtained clusters, discarding the rest. The results for a few different frames are shown in Figure 5.17: each row represents a different time frame (we chose three consecutive frames) and the first row contains the measured R-D map with the two reflections from the two targets. The second and third rows respectively show the result of selecting separately the two biggest clusters after applying HDBSCAN. The algorithm is clearly able to extract the two contributions of the subjects allowing the creation of two separate streams of μ-D data. The obtained sequences are specific of each subject and can be used for classification, enabling the not yet explored multi-person identification, as shown in Figure 5.18; the signature of Figure 5.16 is correctly split into the contributions of each subject by operating on the R-D map before integration along the range dimension.

To manage the correct labeling of the different clusters during time, i.e. assigning the same label to a specific cluster in different time frames, some level of *tracking* of the subject has to be performed: this task is left as a future work.

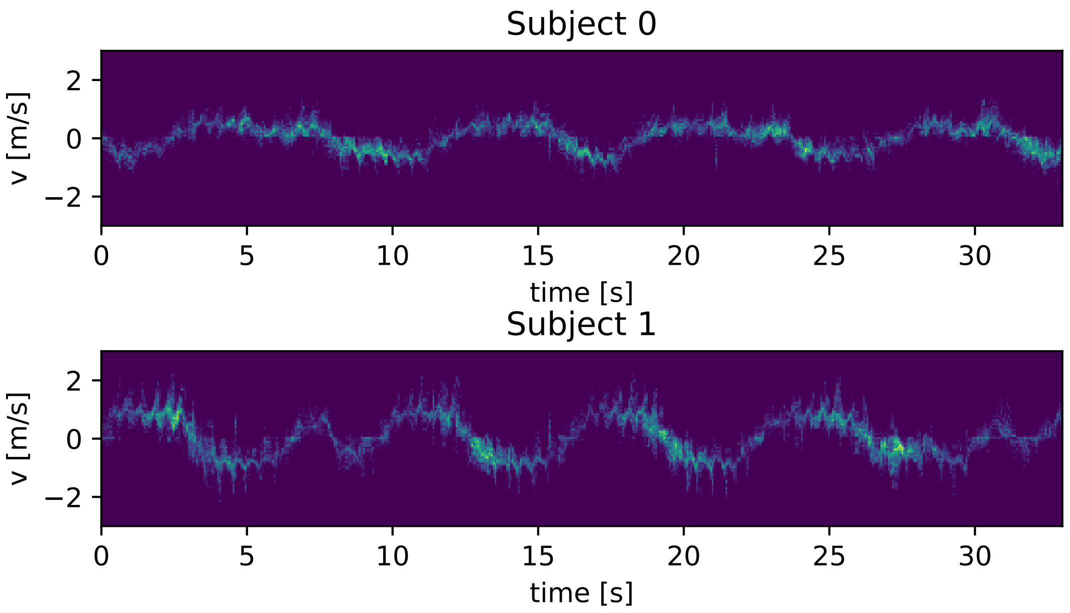


Figure 5.18: Results of the separation in the R-D space. The obtained μ-D signatures belong to the two different subjects.

6 Conclusions and Future Works

In this work we tackled the problem of human subject identification from the micro-Doppler signature of gait, measured with a mm-wave radar sensor. We proposed a novel signal processing framework in order to extract from the radar range-Doppler raw data the useful reflection from the target, separating it from the noise contribution and from unwanted reflections. This was achieved with a combination of a de-noising and a clustering steps that allowed an effective separation and management of the different contributions in the range-Doppler space, bringing an improvement over the pre-processing methods used in the literature and paving the way for the not yet studied multi-person identification problem. In addition, the proposed processing is adaptive with respect to the background noise level and variations in time, allowing a possible online implementation of the system with little preliminar setup. To perform the final identification, we designed a classifier based on the inception modules architecture for Convolutional Neural Networks, that allows feature extraction at different scales, improving the state of the art results on the publicly available IDRad dataset. We presented an analysis of the results and the effects of every processing phase, motivating the trade-offs and the choice of the parameters in each case. The obtained results are promising for the following reasons:

1. The pre-processing and clustering in the R-D space allows a fast and easy management of the different received reflections. This is of key importance when multiple targets or noisy peaks from secondary reflections are present

in the environment, and no other attempt to isolate the single contributions in the R-D map has been studied in the literature, preventing multi-subject identification.

2. Adaptivity may be fundamental in real scenarios where a system of this kind can not be re-programmed at each variation of the background noise level or when moving to different rooms.
3. The proposed classifier demonstrates that CNNs and the IM architecture are very effective in this case thanks to their ability to extract complex features from bi-dimensional input data, outperforming the other methods with an accuracy of 90% and 84% on the IDRad test set when thresholding and adaptive denoising are used respectively.

6.1 Future Improvements

We investigated several improvements of this work, in order to extend the identification to multiple targets and to improve the adaptive denosing phase. Here we briefly describe some possible developments:

1. Improve the KDE denoising.
2. Perform multi-subject identification, including subject tracking in the processing pipeline.
3. Optimize the system to enable online identification.

6.2 Denoising Phase Improvement

The KDE of the background noise PDF has shown to allow adaptive denoising at the cost of reducing the accuracy of the classifier. Further investigation of this fact needs to be made, in order to asses if the trade-off between accuracy and adaptivity can bring some benefits in real situations. For this reason more data, not belonging to the same IDRad dataset used in this work, is needed. The application of the proposed workflow to measurements taken in different rooms and various conditions can verify the effectiveness of the algorithms used and their generality. In addition, in the computer vision field several algorithms have been designed other than KDE to perform foreground detection, such as Gaussian Mixture Models (GMM), Self Organizing Maps (SOM) and deep CNNs [44, 45], all of which can be applied and tested as an alternative to KDE in our problem.

6.3 Multi-target Identification and Tracking

As described in Section 5.7, the proposed clustering in the R-D space allows separation of different targets contributions and the consequent independent classification of each of them. In order to be able to keep labeling correctly the different clusters among subsequent frames, however, some level of tracking of the subject has to be performed. Tracking is a widely studied topic in the radar research community, but few works focus on tracking in the range-Doppler space while the more useful (in general) tracking in the Cartesian space (or polar coordinates) is preferred due to its immediate physical meaning.

Two possible approaches are therefore proposed:

1. Tracking in the R-D frames is viable, using for example Kalman filtering [46] or novel machine learning techniques such as RNNs.
2. The information from multiple channels of the MIMO radar can be exploited, allowing the computation of the angular position of the targets as shown in Section 2.4. Tracking can then be performed in this range-angle dimension and then projected into the R-D map by selecting the Doppler channels corresponding to the range and angle bins occupied by the detected target.

6.4 Online Identification

Online processing would be the final step in the development of the human identification system from μ-D signatures. To this end the pipeline has to be optimized in order to handle the latency requirements of real-time processing. Moreover, this phase may require moving beyond the proposed approach of classifying images of the same size based on a uniform time duration of 3 s: the aim would be an adaptive system that can operate with time windows of different lengths and provide more confident classifications by continuing the measurements. Therefore a possible improvement could be the transition from a time window classification approach to a more meaningful *walking cycle*-based classification, where a walking cycle is the full sequence of movements that compose a step of the walk, after which the cycle is repeated. As shown in [47], the statistical features of different walking cycles can be considered independent, and the typical duration of a human walking cycle is of about 1 second. This kind of approach can be handled with sequential hypothesis-testing algorithms derived by Wald's theory of Sequential Probability Ratio Test (SPRT), in which the classifier keeps accumulating data until the decision it can make satisfies some pre-defined level of confidence [48], achieving the level of adaptivity and versatility expected from a real monitoring and security system.

References

[1] M. A. Richards, J. Scheer, W. A. Holm, and W. L. Melvin, *Principles of modern radar.* Citeseer, 2010.

[2] G. O. Manokhin, Z. T. Erdyneev, A. A. Geltser, and E. A. Monastyrev, “MUSIC-based algorithm for range-azimuth FMCW radar data processing without estimating number of targets,” in *2015 IEEE 15th Mediterranean Microwave Symposium (MMS).* IEEE, 2015, pp. 1–4.

[3] B. Vandersmissen, N. Knudde, A. Jalalvand, I. Couckuyt, A. Bourdoux, W. De Neve, and T. Dhaene, “Indoor person identification using a low-power FMCW radar,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 7, pp. 3941–3952, Jul 2018.

[4] B. Wang, Q. Xu, C. Chen, F. Zhang, and K. R. Liu, “The promise of radio analytics: a future paradigm of wireless positioning, tracking, and sensing,” *IEEE Signal Processing Magazine*, vol. 35, no. 3, pp. 59–80, 2018.

[5] P. Cao, W. Xia, M. Ye, J. Zhang, and J. Zhou, “Radar-ID: human identification based on radar micro-Doppler signatures using deep convolutional neural networks,” *IET Radar, Sonar & Navigation*, vol. 12, no. 7, pp. 729–734, Jul 2018.

[6] Y. Yang, C. Hou, Y. Lang, G. Yue, Y. He, and W. Xiang, “Person Identification Using Micro-Doppler Signatures of Human Motions and UWB Radar,” *IEEE Microwave and Wireless Components Letters*, vol. 29, no. 5, pp. 366–368, May 2019.

[7] S. Abdulatif, F. Aziz, K. Armanious, B. Kleiner, B. Yang, and U. Schneider, “Person identification and body mass index: A deep learning-based study on micro-Dopplers,” in *IEEE Radar Conference (RadarConf)*, Boston, Massachusetts USA, Apr 2019.

[8] Z. Chen, G. Li, F. Fioranelli, and H. Griffiths, “Personnel recognition and gait classification based on multistatic micro-Doppler signatures using deep convolutional neural networks,” *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 5, pp. 669–673, May 2018.

[9] A. Jalalvand, B. Vandersmissen, W. De Neve, and E. Mannens, “Radar signal processing for human identification by means of reservoir computing networks,” in *IEEE Radar Conference (RadarConf)*, Boston, Massachusetts USA, Apr 2019.

[10] V. Polfliet, N. Knudde, B. Vandersmissen, I. Couckuyt, and T. Dhaene, “Structured inference networks using high-dimensional sensors for surveillance purposes,” in *International Conference on Engineering Applications of Neural Networks (EANN)*, Crete, Greece, May 2018.

[11] F. Luo, S. Poslad, and E. Bodanese, “Human Activity Detection and Coarse Localization Outdoors Using Micro-Doppler Signatures,” *IEEE Sensors Journal*, pp. 1–1, May 2019.

[12] R. Trommel, R. Harmanny, L. Cifola, and J. Driessen, “Multi-target human gait classification using deep convolutional neural networks on micro-Doppler spectrograms,” in *European Radar Conference (EuRAD)*, London, United Kingdom, Oct 2016.

[13] M. S. Seyfioğlu, A. M. Özbayoğlu, and S. Z. Gürbüz, “Deep convolutional autoencoder for radar-based classification of similar aided and unaided human activities,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 54, no. 4, pp. 1709–1723, Feb 2018.

[14] R. Pérez, F. Schubert, R. Rasshofer, and E. Biebl, “Single-Frame Vulnerable Road Users Classification with a 77 GHz FMCW Radar Sensor and a Convolutional Neural Network,” in *19th International Radar Symposium (IRS)*, Bonn, Germany, Jun 2018.

[15] F. Weishaupt, I. Walterscheid, O. Biallawons, and J. Klare, “Vital Sign Localization and Measurement Using an LFMCW MIMO Radar,” in *19th International Radar Symposium (IRS)*, Bonn, Germany, Jun 2018.

[16] C.-H. Hsieh, Y.-F. Chiu, Y.-H. Shen, T.-S. Chu, and Y.-H. Huang, "A UWB radar signal processing platform for real-time human respiratory feature extraction based on four-segment linear waveform model," *IEEE transactions on biomedical circuits and systems*, vol. 10, no. 1, pp. 219–230, Feb 2015.

[17] S.-H. Kim and G.-T. Han, "1D CNN Based Human Respiration Pattern Recognition using Ultra Wideband Radar," in *International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, Okinawa, Japan, Feb 2019.

[18] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 2, pp. 513–529, Apr 2011.

[19] V. Winkler, "Range Doppler detection for automotive FMCW radars," in *European Radar Conference (EuRAD)*, Munich, Germany, Oct 2007.

[20] S. M. Patole, M. Torlak, D. Wang, and M. Ali, "Automotive radars: A review of signal processing techniques," *IEEE Signal Processing Magazine*, vol. 34, no. 2, pp. 22–35, Mar 2017.

[21] V. C. Chen, "Analysis of radar micro-Doppler with time-frequency transform," in *Proceedings of the Tenth IEEE Workshop on Statistical Signal and Array Processing*, Pocono Manor, Pennsylvania, USA, Aug 2000.

[22] V. C. Chen, F. Li, S.-S. Ho, and H. Wechsler, "Micro-Doppler effect in radar: phenomenon, model, and simulation study," *IEEE Transactions on Aerospace and electronic systems*, vol. 42, no. 1, pp. 2–21, Aug 2006.

[23] A. Elgammal, R. Duraiswami, D. Harwood, and L. S. Davis, "Background and foreground modeling using nonparametric kernel density estimation for visual surveillance," *Proceedings of the IEEE*, vol. 90, no. 7, pp. 1151–1163, Jul 2002.

[24] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *2nd Inter-*

national Conference on Knowledge Discovery and Data Mining, Portland, Oregon, USA, Aug 1996.

[25] R. J. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Pacific-Asia conference on knowledge discovery and data mining.* Gold Coast, QLD, Australia: Springer, Apr 2013.

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

[27] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul 2006.

[28] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of machine learning research*, vol. 11, pp. 3371–3408, Dec 2010.

[29] C. M. Bishop, "Pattern recognition and machine learning," 2007.

[30] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Oakland, CA, USA, Jun 1967.

[31] J. C. Gower and G. J. Ross, "Minimum spanning trees and single linkage cluster analysis," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 18, no. 1, pp. 54–64, 1969.

[32] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[33] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms.* Cambridge university press, 2014.

[34] D. A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," in *International Conference on Learning Representations (ICLR)*, San Juan, Puerto Rico, May 2016.

[35] J. Watt, R. Borhani, and A. K. Katsaggelos, *Machine learning refined: foundations, algorithms, and applications.* Cambridge University Press, 2016.

[36] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[37] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[39] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, vol. 11, pp. 625–660, Feb 2010.

[40] D. Tahmoush and J. Silvious, "Radar micro-Doppler for long range front-view gait recognition," in *IEEE 3rd International Conference on Biometrics: Theory, Applications, and Systems*, Washington DC, Columbia, USA, Sep 2009.

[41] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, Boston, Massachussets, USA, Jun 2015.

[42] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, Jun 2014.

[43] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1724–1734.

[44] M. Babaee, D. T. Dinh, and G. Rigoll, "A deep convolutional neural network for video sequence background subtraction," *Pattern Recognition*, vol. 76, pp. 635–649, 2018.

[45] G. Wang, J. Xu, and M. Fang, "A comparison of background subtraction algorithms evaluated with BMC dataset," in *2016 12th World Congress on Intelligent Control and Automation (WCICA)*. IEEE, 2016, pp. 2608–2613.

[46] D. Fränken and S. Woischneck, "Range/Doppler tracking with the Kalman filter and its relatives—A comparative study," in *2017 Sensor Data Fusion: Trends, Solutions, Applications (SDF)*. IEEE, 2017, pp. 1–6.

[47] M. Gadaleta and M. Rossi, "IDNet: Smartphone-based gait recognition with convolutional neural networks," *Pattern Recognition*, vol. 74, pp. 25–37, 2018.

[48] A. Tartakovsky, I. Nikiforov, and M. Basseville, *Sequential analysis: Hypothesis testing and changepoint detection.* Chapman and Hall/CRC, 2014.

Acknowledgments