POLITECNICO DI TORINO

Master of Science Degree in
Mechatronic Engineering

MASTER'S THESIS

In collaboration with
California State University Los Angeles

Deep learning and data augmentation techniques for indoor environment characterization via UWB technology

---

Supervisors
Prof. Chiaberge Marcello
Prof. Mondin Marina

Candidate
MORIN Mattia

ACADEMIC YEAR 2021-2022

Abstract

Ultra-wideband is a radio-based communication technology for short-range use and fast and stable transmission of data. Its main characteristics are extremely large bandwidth, very low signal power density, robustness against fading, and low cost. These features make ultra-wideband suitable for indoor localization applications. Referring to the positioning accuracy, it is around five and twenty-five times better than adopting Bluetooth Low Energy (BLE) beacons or Wi-Fi, respectively. Despite these promising results, ultra-wideband localization accuracy robustly degrades when moving to non-ideal conditions, including Non-Line-Of-Sight and the presence of dynamic environmental factors. This work aims at characterizing an indoor environment via channel impulse responses retrieved by ultra-wideband technology, opening the doors to future research in terms of sensor fusion techniques for improving indoor localization accuracy and indoor channel characterization.

As a first step of this work, a sufficiently large dataset is collected in a typical indoor scenario, including static, dynamic, Line-Of-Sight and Non-Line-Of-Sight conditions. In this regard, data augmentation techniques are exploited to enlarge and enrich the dataset. Then, a deep neural network in the context of deep learning is chosen to classify a certain number of positions from the channel impulse responses. In parallel, a class activation map algorithm is considered to provide model explainability, a crucial missing aspect in many deep learning projects.

Results show channel impulse responses carry enough information about the multipath delays in an indoor environment, at least in the static case. When a realistic dynamic environment is considered, indoor environment characterization becomes harder. As well known, a very large dataset is needed when dealing with deep learning models. In this regard, data augmentation comes in hand, leading to better results in terms of network generalization. The last chapter draws the conclusion of the presented work, including suggestions for future research and objectives to be pursued.

Contents

List of Figures

List of Tables

Acronyms

AI
Artificial Intelligence
CAM
Class Activation Map
CIR
Channel Impulse Response
CNN
Convolutional Neural Network
DL
Deep Learning
DNN
Deep Neural Network
EIRP
Effective Isotropic Radiated Power
IR
Impulse Radio
GPS
Global Positioning System
ML
Machine Learning
MLP
Multi Layer Perceptrons
MTS
Multivariate Time Series
NLOS
Non Line Of Sight
PRF
Pulse Repetition Frequency
PSD
Power Spectral Density

ReLU
Rectified Linear Unit
SNR
Signal-to-Noise Ratio
TSC
Time Series Classification
TWR
Two-way Ranging
UWB
Ultrawide band
XAI
Explainable Artificial Intelligence

Chapter 1

Introduction

1.1 Thesis objective

The objective of the proposed work is to validate the idea of characterization of an indoor environment via Channel Impulse Responses based on ultrawide-band technology. Therefore, an overview of the UWB and CIR is presented first. Then, a dataset collection and analysis is carried out, together with a data augmentation stage. Finally, the exploitation of a Deep Learning model is shown to prove the thesis, together with limitations of the presented approach.

1.2 Thesis organization

The thesis is structured in five main chapters, here briefly summarized.

1. **Ultrawide-band** The first chapter describes the ultrawide-band technology, together with its history and regulation. Moreover, an overview about channel modelling in an indoor environment is presented and discussed in detail.

2. **Artificial Intelligence** This section briefly describes what AI actually is and the difference between Machine Learning and Deep Learning. Additionally, it presents the most common classifiers for time-series data. Finally, it introduces the concept of Explainable Artificial Intelligence.

3. **Experimental Set** In this chapter the software and hardware resources are presented. Together with that, the measurement setup for data collection is illustrated.

4. **Dataset Collection and Analyses** Any ML or DL model needs data. This section reports the collected datasets and relative inspections. Furthermore, data augmentation techniques are presented and discussed in detail.

5. **Classification** In this chapter the whole pipeline for environment characterization via CIR is presented. In addition, the data pre-processing stage, the adopted DNN model and learning strategies are discussed as well. This section also reports the obtained results in terms of metrics and confusion matrices. Furthermore, it shows the outcome of the Grad-CAM algorithm on a test dataset.

Chapter 2

Ultrawide-band

2.1 History and regulation

UWB systems were born at the end of XIX century when Guglielmo Marconi, obsessed by the idea of a wireless connected-world, built the first radio communication apparatus. Beside that, the first UWB communication system started in London few years later, linking two post offices at a distance greater than one mile [1].

The interest in UWB was then renewed after the Second World War, when subnanosecond instruments became to be available. Today, thanks to the research activities carried out in the last decades in the fields of analog and digital electronics, commercial UWB systems are accessible worldwide.

In 1998, the Federal Communication Commission (FCC) recognized the importance of the UWB technology and a substantial change occurred in February 2002 when the FCC issued a ruling pointing out that UWB could be used for data communications as well as radar and safety applications [2].
Consequently of the cited regulation, the UWB technology has been authorized for the commercial uses with different applications, operating frequency bands and transmitted power spectral densities.

2.2 Definition

In accordance with the definition given by FCC, a UWB transmission is *any signal which shows a fractional bandwidth* (B_f) *larger than 0.20, or which occupies a bandwidth greater than 500 MHz*, i.e.,

$$B_f \geq 0.2 \quad or \quad BW > 500\ MHz \tag{2.1}$$

The fractional bandwidth is defined as follows:

$$B_f = \frac{BW}{f_c} = \frac{(f_H - f_L)}{(f_H + f_L)/2}, \tag{2.2}$$

where f_H and f_L are the highest and the lowest transmitted frequencies at the -10 dB emission point, respectively, BW is the signal bandwidth and f_c is the center frequency.

As shown in Fig. 2.1 the conventional radio transmission systems (i.e. narrowband and wideband systems) have small fractional bandwidths if compared to the UWB signals.

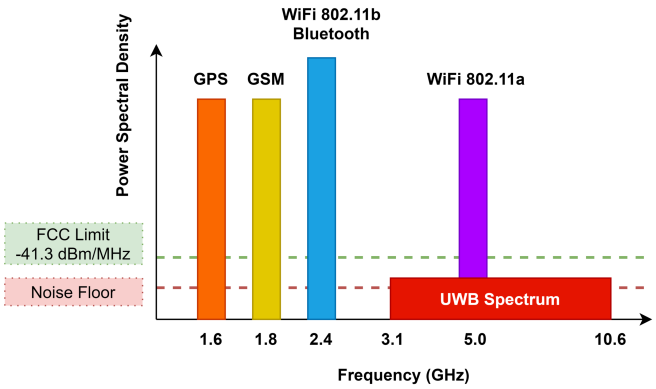


Figure 2.1: PSD comparison - UWB and other narrowband systems

The band the FCC allocated to UWB communications is 7.5 GHz between 3.1 and 10.6 GHz, which is perhaps the largest allocation of bandwidth to any commercial terrestrial system.

If the entire band is utilized, the maximum power available to a transmitter is approximately 0.5 mW. This is just a minute fraction if compared to what is available for the 2.4 GHz and 5.8 GHz ISM (industrial, scientific and medical) bands used by WLAN standards such as IEEE 802.11 a/b/g. The power limitation relegates UWB to indoor, short-range communications for high data rates, or very low data rates for longer link distances.

Because UWB systems operate with such a very large bandwidth, they share the frequency spectrum with the others existing communication systems. To avoid interference, different regions of the spectrum have different allowed power spectral densities. These are regulated by the FCC which has assigned the EIRP allowed for each frequency band.

Figure 2.2 shows the FCC and ECC (Electronic Communications Committee) masks for an indoor UWB device, respectively valid for the US and the European Union.

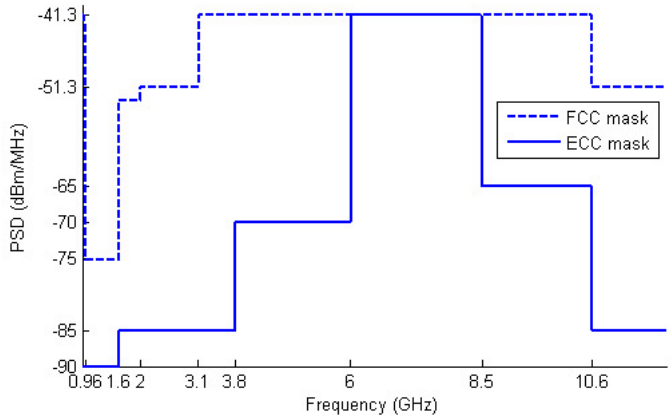


Figure 2.2: UWB spectral masks - Indoor environment

2.3 Characteristics

Unlike other communication systems, the UWB transmitter produces a very short time-domain pulse (typically in the order of sub-nanoseconds) which is able to propagate without the need for an additional RF (radio frequency) mixing stage. It makes the UWB a potentially low complexity and low cost system.

In addition, thanks to the low energy density of the transmitted signal, it appears that the low-power, noise-like UWB transmissions do not cause significant interference to existing radio systems.

Because of the large bandwidth of the transmitted signal, very high multipath resolution is achieved. This aspect, together with the discontinuous transmission makes the UWB resistant to multipath propagation and jamming.

Finally, the very narrow time-domain pulses make the UWB able to offer very precise timing, much better than GPS and other radio systems, also motivating its use for indoor localization.

2.4 Basics

2.4.1 Transmission

A type of UWB communications system able to transmits such narrow-band UWB pulses is called *impulse radio*. In a IR UWB system, UWB pulses are transmistted in a discontinuous way.
On the basis of the employed UWB pulse shape, a certain banwdwidth is occupied by the signal.

This type of transmission does not require the use of additional carrier modulation as the pulse will propagate well in the radio channel. The technique is therefore a basebad signaling technique and this radio concept is known as impulse radio [2].

The UWB pulse waveform can be any function as long as it satisfies the spectral mask regulatory requirements shown in Fig. 2.2.

An example of UWB pulse shape is illustrated in Fig. 2.3.

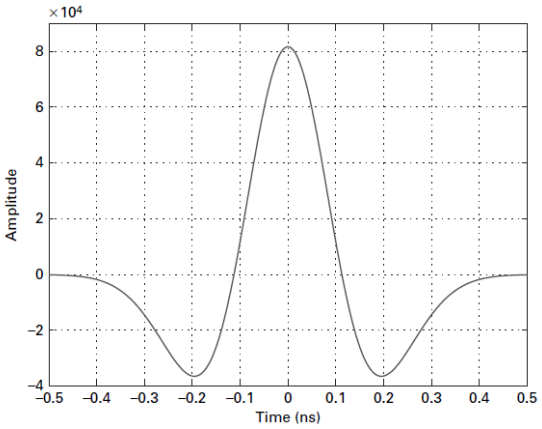


Figure 2.3: UWB pulse shape

The typical signal used for UWB transmission is the Gaussian pulse, defined by:

$$p(t) = \frac{A}{\sqrt{2\pi\sigma^2}} e^{(-\frac{t^2}{2\sigma^2})} \tag{2.3}$$

where, in (2.3), A denotes the amplitude and σ denotes the spread of the Gaussian pulse.

In Fig. 2.4 a typical shape of a Gaussian pulse is shown.

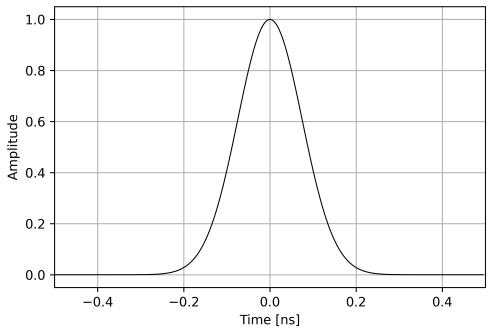


Figure 2.4: UWB Gaussian pulse shape

To be compliant with the FCC mask, usually higher derivatives of the Gaussian shape are exploited for the UWB transmission. The n-th derivative of a Gaussian pulse can be computed in a recursive way from the following equation:

$$p^{(n)}(t) = -\frac{n-1}{\sigma^2} p^{(n-2)}(t) - \frac{t}{\sigma^2} p^{(n-1)}(t) \tag{2.4}$$

The PSD shape of different derivatives of the Gaussian pulse is depicted in Fig. 2.5.

According to this figure, to comply with the FCC indoor emission limits, at least the *4-th* derivative should be transmitted.

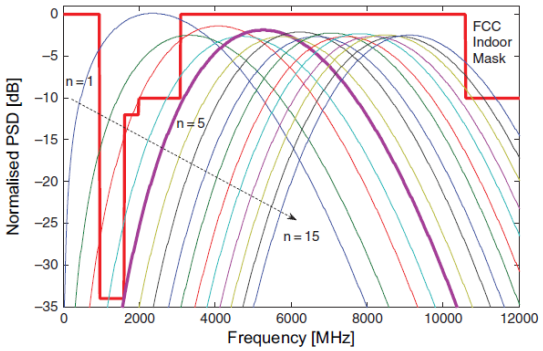


Figure 2.5: PSD of different order derivatives of the Gaussian pulse

2.4.2 Modulation

Modulation is required to carry information together with the signal.
There are a few modulation schemes that may be used with UWB systems, below concisely described.

Pulse amplitude modulation (PAM)

In this modulation scheme, the transmitted binary pulse amplitude-modulated signal $s_{tr}(t)$ can be written as:

$$s_{tr}(t) = d_k w_{tr}(t) \tag{2.5}$$

where $w_{tr}(t)$ is the UWB pulse waveform, k represents the transmitted bit ("0" or "1") and

$$d_k = \begin{cases} -1 & \text{if k} = 0 \\ +1 & \text{if k} = 1 \end{cases} \tag{2.6}$$

is used for the antipodal representation of the transmitted bit k.

On-Off Keying (OOK)

The OOK scheme is based on the PAM one with the only difference that no signal is transmitted in the case of bit "0", i.e.,

$$d_k = \begin{cases} 0 & \text{if k} = 0 \\ 1 & \text{if k} = 1 \end{cases} \tag{2.7}$$

Pulse Position Modulation (PPM)

With Pulse Position Modulation, the bit to be transmitted influences the position of the UWB pulse, instead of the amplitude. It means that bit "0" is represented by a pulse at the initial time instant 0, while bit "1" is shifted in time by an amount δ from 0. In mathematical terms:

$$s_{tr}(t) = w_{tr}(t - \delta d_k) \tag{2.8}$$

where d_k is defined as in (2.7).

Pulse Shape Modulation (PSM)

Among all modulation schemes, PPM and PAM are the most common in IR UWB systems. PSM is an alternative and more complicated scheme where different pulse shapes are used to represent information bits.

Fig. 2.6 depicts all modulation schemes graphically [3]:

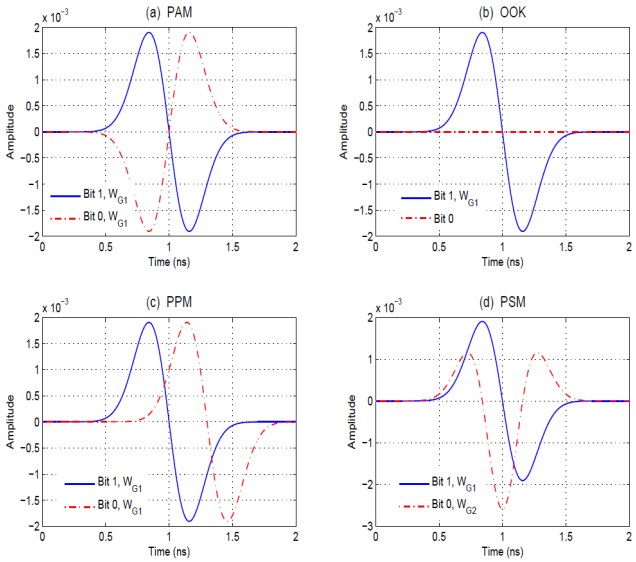


Figure 2.6: UWB modulation schemes

2.5 Wireless channel

The medium between the transmitting antenna and the receiving antenna is generally termed as *channel* [4].

In wireless transmission, the characteristics of the signal change and evolve as long as it travels through the channel. It is due to several phenomena, including reflection, refraction, the transmitter-receiver relative motion and signal attenuation.
In addition, wireless channels are more susceptible than cabled channels to noise and interference.

Before explaining the adopted UWB channel model for wireless communication, an overview about indoor channel modelling is presented.

2.5.1 Indoor environments

Since this work focuses on an indoor environment scenario, a brief description of channel modelling in this regard is presented.

The indoor environment is a dynamic one with distinctive characteristics. The presence of walls makes the indoor coverage constrained and diffraction plays an important role, especially in NLOS.
Additionally, even in case of stationary transmitter and receiver, the movements of people make the indoor channel to be time-variant [5]. In addition to that, the indoor propagation analysis also strictly depends on the building's geometry and indoor channels are perhaps very sensitive to the location of the antenna.
All these aspects make the indoor channel characterization an hard task to tackle.

2.5.2 Channel modelling

There are two main approaches to model wireless indoor channel propagation, known as stochastic and deterministic.
In stochastic modelling (as the name suggests), the model is obtained from data. First data are collected from real measurements, then via statistical analysis channel coefficients are extracted to turn the data into parametric equations.
On the contrary, deterministic modelling is based on laws of physics.

While deterministic models are built for a specific environment, stochastic models adapt to environments which share similar characteristics to the one used to

construct the model. Since our work does not want to be site-specific, the focus is placed on stochastic models.

There are many stochastic models for indoor wireless channels. Among them, the Saleh-Valenzuela (SV) model [6] is a popular model used to describe the behaviour of multipath in indoor environments and it has been adopted in the IEEE 802.15.3a and IEEE 802.15.4a.
The indoor channel based on the SV model can be described by:

$$h(t) = \sum_{l=0}^{\infty} \sum_{k=0}^{\infty} \beta_{kl} e^{j\phi_{kl}} \delta(t - T_l - \tau_{kl}) \tag{2.9}$$

where β_{kl} is the multipath gain, ϕ is the phase associated with the lth cluster and kth ray, l is the number of clusters, k is the number of arrival rays within the lth cluster, T_l is the arrival time of the lth cluster and τ_{kl} is the arrival time of the kth ray within the lth cluster.
The SV model was originally developed for wideband systems, but it is valid for UWB systems as well.

2.5.3 UWB Channel Impulse Response model

In UWB communication, denoting with w_{tr} the transmitted pulse (forgetting for a while about modulation), the received signal *r(t)* after the channel can be written as:

$$r(t) = Aw_{rec}(t - \tau) + n(t) \tag{2.10}$$

where τ is the propagation delay between transmitter and receiver and *n(t)* is the additive noise. Among other things, note that due to the UWB channel nature, the transmit waveform changes from w_{tr} to w_{rec}. It is then delayed by τ and attenuated by A.

The UWB wireless channel can be entirely described by its impulse response function *h(t)*, expresses as follows:

$$h(t) = \sum_{n=0}^{N} a_n \delta(t - \tau_n) e^{j\theta_n} \tag{2.11}$$

where a_n , τ_n , θ_n , N are the parameters of the nth path and they represent amplitude,delay,phase and number of multipath components, respectively.
When a UWB signal is a baseband one (e.g. in case of IR UWB signals), the phase θ in (2.11) can be omitted and the equation becomes easily interpretable.
Note that the equation in (2.11) is a simplification of the more complex SV model discussed in Section 2.5.2.

This equation is crucial for the thesis objective since it explains the multipath distribution in an indoor environment, via the channel impulse response.

2.5.4 Channel Impulse Response

In order to accurately illustrate the Channel Impulse Response, a review of the transmit and receive signal models is presented first.
Bandpass signals are frequently employed to model transmitted and received signals in communication systems. Since the transmitter circuitry is built so as to produce real sinusoids only, these bandpass signals are real.

A bandpass signal *s(t)* at a carrier frequency f_c can be expressed in the following form:

$$s(t) = s_I(t)cos(2\pi f_c t) - s_Q(t)sin(2\pi f_c t) \tag{2.12}$$

Conventionally, $s_I(t)$ and $s_Q(t)$ are called the *in-phase* and *quadrature* components of *s(t)*, respectively. Defining the complex signal $u(t) = s_I(t) + js_Q(t)$, it implies $s_I(t) = Re\{u(t)\}$ and $s_Q(t) = Im\{u(t)\}$. On the basis of this definition,

$$s(t) = Re\left\{u(t)\right\} cos(2\pi f_c t) - Im\left\{u(t)\right\} sin(2\pi f_c t) = Re\left\{u(t)e^{j2\pi f_c t}\right\} \tag{2.13}$$

As a natural consequence, the received signal will have a similar form,

$$r(t) = Re\left\{v(t)e^{j2\pi f_c t}\right\} + n(t) \tag{2.14}$$

The received signal in (2.14) is made up of two components: the first corresponds to the transmitted signal after propagation through the channel, the second models the noise added by the channel.

In a typical indoor environment, a transmitted wireless signal, modelled as in (2.13), runs into multiple objects in the surroundings, producing reflected and diffracted copies of the original transmitted signal. These additional copies are known as *multipath signal components* (MPCs) which can be attenuated in power, delayed in time and shifted in phase with respect to the LOS signal path at the receiver side.
If the transmitter (TX), receiver (RX) and reflectors are all stationary (i.e., immobile), then the characteristics of the multiple received paths are fixed. Conversely and rationally, if the transmitter or the receiver are moving, the MPCs vary over time.

Due to multipath, a single pulse trasmitted over a multipath channel appears, at the receiver side, as a pulse train. Each received pulse corresponds to either the LOS component, or a distinct multipath component associated with a distinct reflection or cluster of reflections. Furthermore, the multipath channel is time-varying by its nature [7]. This characteristic arises from the fact either the transmitter or the receiver is unfixed, or the environment is a dynamic one, i.e., it is evolving over time because people and/or objects are moving over time. As a result, the relative location of the reflectors in the transmission path (elements which give rise to multipaths) will change over time as well.

Considering the transmitted signal as modelled in (2.13) and neglecting the noise induced by the channel, the signal at the RX side is the sum of the LOS path and all resolvable multipath components. It can be expressed by:

$$r(t) = Re\left\{\sum_{n=0}^{N(t)} \alpha_n(t)u(t-\tau_n(t))e^{j(2\pi f_c(t-\tau_n(t))+\phi_{Dn}}\right\} \tag{2.15}$$

where $n=0$ corresponds to the LOS signal path, α_n, τ_n, D_n are the amplitude, delay, Doppler phase shift of the *nth* multipath component and f_c is the carrier frequency.
It is essential to draw attention to the fact the *nth* resolvable multipath component may correspond to the multipath associated with:

1. a single reflector
2. multiple reflectors grouped together which generate MPCs with similar delays

The two cases are depicted in Fig. 2.7.

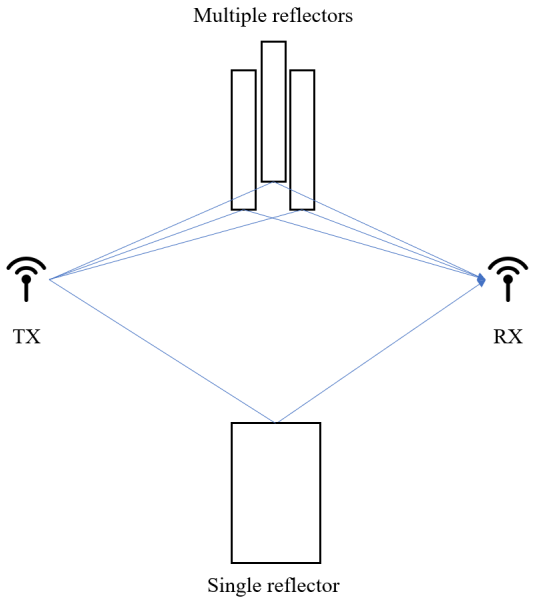


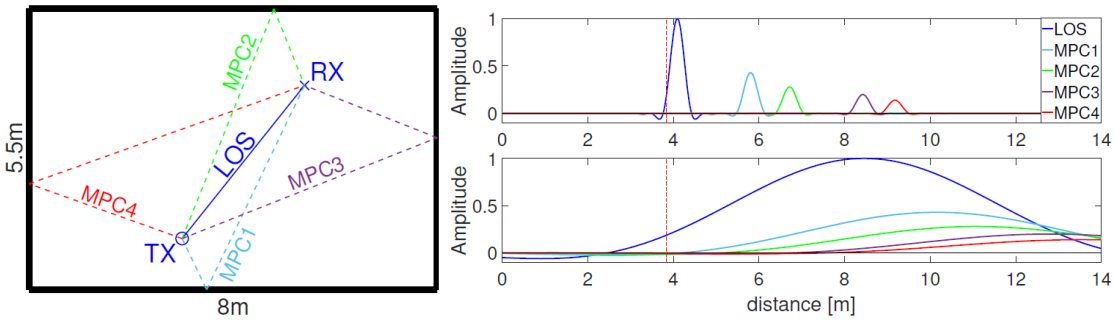
Figure 2.7: Multipath components induced by single or multi reflectors

In the former case, everything is simple and intuitive. Suppose to consider the case where the *nth* multipath component is the outcome of multiple reflectors. By definition, two MPCs with delay τ_1 and τ_2 are said to be *resolvable* if:

$$|\tau_1 - \tau_2| \gg B_u^{-1} \tag{2.16}$$

where B_u refers to the signal bandwidth.
Multipath components which do not satisfy the resolvability criteria expressed in (2.16) cannot be separated at the receiver side and we call them *nonresolvable* multipath components (NMPCs). These NMPCs are grouped into a single multipath with delay $\tau \approx \tau_1 \approx \tau_2$ and amplitude and phase equivalent to the summation of the different components.
It leads to a significant conclusion. With reference to a static environment with immobile TX and RX, τ_1 and τ_2 are fixed in value. In a narrowband system, the bandwidth is relatively small. As a consequence, the second term in the resolvability equation is high, leading to the generation of NMPCs. On the contrary, wideband channels tend to have resolvable multipath components.

On the basis of this conclusion, the employment of the ultrawide-band technology (being its wireless channel a ultrawide-band one) allows to retrieve the multipath components (MPCs) from the use of the characteristic tiny pulse, as shown in Fig. 2.8, [8].



(a) Rectangular floor plan (b) Multipath reflections

Figure 2.8: (a) floor plan representing LOS and MPCs; (b) multipath reflections with $BW = 900\ MHz$ (top) and $BW = 50\ MHz$ (bottom)

In detail, Fig. 2.8a depicts an example of a rectangular floor plan where a TX sends pulses to a RX. The solid line shows the LOS signal, whereas the dashed lines represent the first-order MPCs (i.e., the pulse that is reflected by a single object only).
The theoretically received signals for $BW = 900\ MHz$ (reasonable value for a UWB signal) and $BW = 50\ MHz$ (narrowband system) are illustrated in Fig. 2.8b.

Fig. 2.8b motivates the adoption of the UWB technology for indoor environment characterization (the objective of this work). In fact, the multipath reflections at 50 MHz are highly overlapping and it is not possible to extract multipath components, according to (2.16). On the contrary, UWB systems are resistant to multipath fading, making the received multipaths well separated.

When the bandwidth is sufficiently high, the received pulse (as well as the transmitted one) is extremely tiny and Fig. 2.8b resembles the Channel Impulse Response, carrying the information about the multipath propagation consisting of reflections from walls, objects and moving people, typical elements of an indoor environment.
The CIR can be modelled as in (2.11) and will be exploited as input in a classification algorithm, so as to provide a location-based indoor environment characterization.

2.6 Ranging

Having investigated the UWB signal and channel modelling, this section focuses on the main ranging methods to compute the distance between two (or more) UWB modules.
The distance estimation can be either computed directly from the signal travelling between the two nodes (i.e., *direct positioning*), or by a two-step procedure. In the latter, at first one or more parameters are extracted from the signals, then these parameters are exploited to estimate the distance (see Fig. 2.9).
In the following, two-step approaches are considered only.

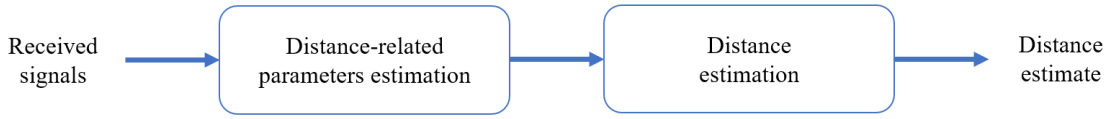


Figure 2.9: Two-step ranging approach

The most common two-step approaches to compute the distance between two UWB modules are:

- **Received signal strength (RSS)**, which computes the distance on the basis of the received amplitude (strength) of the signal. The main assumption behind this approach is that a relation between the distance and power loss has to be known;
- **Angle of Arrival (AOA)**, based on the direction (angle) of the incoming signal;
- **Time of Arrival (TOA)**, which provides the distance between two nodes by estimating the time of flight of the signal travelling from the transmitter to the receiver;
- **Time Difference of Arrival (TDOA)**, where the difference between the arrival times of two signals travelling from the target node to two reference nodes is estimated. It requires to have at least three UWB modules.

The ToA-based ranging method is among the most popular ones and the SS-TWR ranging techniques implements it.

2.6.1 Single-sided Two-way Ranging

In a SS-TWR scheme the two devices exchange one message only and it involves the measurement of a round-trip delay of such message. It is depicted in Fig. 2.10, from [9].

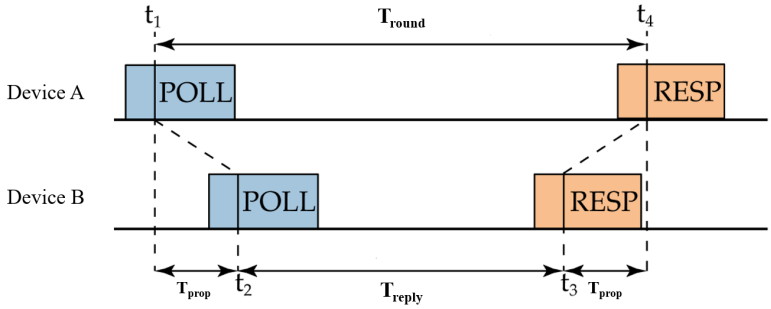


Figure 2.10: SS-TWR scheme

The device A (initiator) initiates the exchange and device B (responder) responds to complete the exchange. More precisely, the initiator transmits a POLL packet and stores the corresponding TX timestamp t_1. Then the receiver replies back with a RESPONSE packet after a certain delay T_{reply}. On the basis of the RX timestamp t_4, the initiator can calculate the round-trip delay as $T_{round} = t_4 - t_1$. Since the RESPONSE packet includes also the RX timestamp t_2 of the received POLL and the TX timestamp t_3 of the RESPONSE , the device A is able to compute the actual response delay of the responder as $T_{reply} = t_3 - t_2$.

The Time of Flight (T_{prop} in Fig. 2.9, with reference to the fact it is the propagation time of the signal) can be computed as:

$$T_{prop} = \frac{T_{round} - T_{reply}}{2} = \frac{(t_4 - t_1) - (t_3 - t_2)}{2} \tag{2.17}$$

And from (2.17) the distance between the two devices can be easily calculated as:

$$d = T_{prop} \cdot c \tag{2.18}$$

where c is the speed of light in the air.

2.6.2 Double-sided Two-way Ranging

SS-TWR is the simplest but not the best approach to compute the Time of Flight between two UWB nodes. In fact, the error in the computation in the propagation time increases as the clock offset of the two devices increases. A preferred technique to lower this error is DS-TWR. In this case, the inaccuracy in the ToF

estimate is constant with reference to the clock offset.

The DS-TWR is an extension of the SS-TWR in which two round-trip measurements are performed and combined to result in a reduced-error ToF, even for quite long response delays and high clock offsets [10].
Considering Device A as initiator and Device B as responder, Fig. 2.11 shows the operation of this scheme:

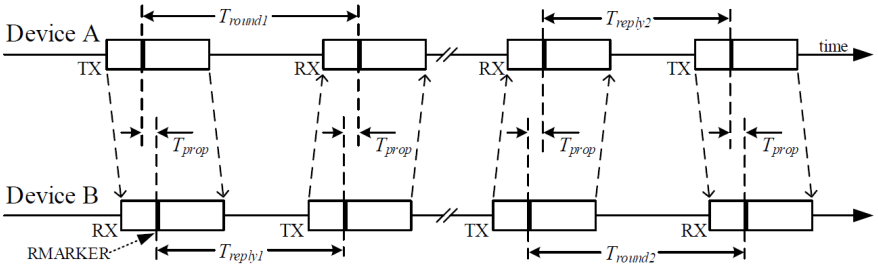


Figure 2.11: DS-TWR scheme

The DS-TWR can be further simplified into a three-message scheme, simply using the first reply of the responder as the initiator for the second round-trip measurement, as shown in Fig. 2.12:

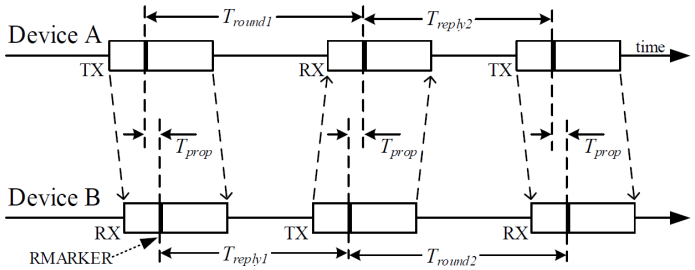


Figure 2.12: Three messages DS-TWR scheme

With reference to the three-message scheme, the Time of Flight can be estimated as:

$$T_{prop} = \frac{T_{round1} \cdot T_{round2} - T_{reply1} \cdot T_{reply2}}{T_{round1} + T_{round2} + T_{reply1} + T_{reply2}} \tag{2.19}$$

The equation in (2.19), together with the three-messages DS-TWR ranging scheme are implemented in the EVK1000 Evaluation Kit [11] employed in this work.

Chapter 3

Artificial Intelligence

Artificial Intelligence (AI) is a field of computer science devoted to solve problems by taking inspiration from the human intelligence.

Computers and software have been programmed to execute tasks by applying a certain number of pre-defined rules, sometimes in a relatively straightforward way, other times in a more challenging one.
Until 2012, AI was limited to advanced technological companies, government institutions and research agencies. Since then, thanks to the wide availability of GPUs (Graphics Processing Units) and to the almost infinite storage of data of every type, it has broken the boundaries becoming available into real-world business everyday solutions [12].

The subfields of Artificial Intelligence are Machine Learning and Deep Learning, as depicted in Fig. 3.1.

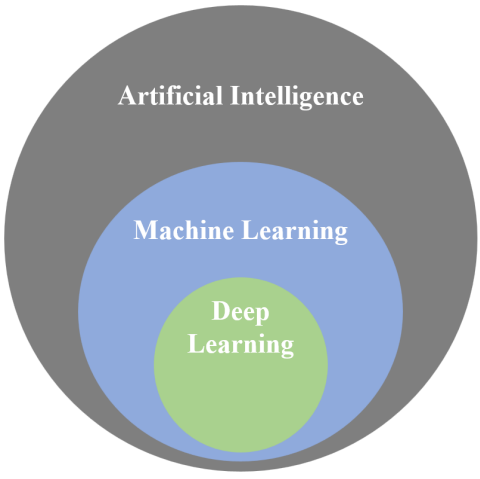


Figure 3.1: Artificial Intelligence and its subfields

3.1 Machine Learning

As already pointed out, until recently most computer programs have been coded as a rigid set of rules. However, there are problems for which this is not possible. Imagine to write a software able to predict the weather for the next days or to classify images according to a specific set of classes. The former may depend on a pattern that evolves over time, the latter involves relationships which are really complicated. In problems like these, even the greatest software engineers might struggle in coding solutions. This is where Machine Learning comes in hand.

Machine Learning (ML) can be conceived as the ability of making computers able to learn and perform tasks without being explicitly programmed. It turns the deterministic approach characteristic of the standard programming, into a statistical, data-based one.
Every machine learning problem needs:

1. A *model* to handle and transform the data
2. The *data* from which the model can learn
3. An *objective function* to quantify how much the model is doing well
4. A *learning algorithm* to adjust the model's parameters to improve the objective function

This is all but simple and it poses several challenges into ML problems. The most common ones are briefly discussed in the next section.

3.1.1 Challenges

Although Machine Learning is widely used in almost every industry and field, there exist a lot of challenges in the development of a ML application that can not be ignored.

Lack of training data

Consider to make a newborn learn how to say "mum". All you have to do, it is just to say him "mum" over and over again. If you do it enough, the child can absorb that word.

This is exactly what is needed for many Machine Learning models to properly work as well. ML models needs a lot of data (even thousands for a simple task and millions for harder ones). On the other hand, if the data availability lacks during

training, even the best ML model can fail.
This is even more true for Deep Learning models, as it will be pointed out later.

Poor data quality

Even in case the data availability is enough, if the training data contains a lot of outliers, errors and noise, it makes the ML model infeasible in detecting a correct pattern during training. As a consequence, the model will perform badly.
This explains why Data Scientists spend a relevant percentage of their time in data cleaning.

Irrelevant features

Having lots of data and pre-processing them correctly is not enough. It is fundamental to have the *right* data as well. Learning will be surely unsuccessful if the selected features are not predictive of the target quantity of interest [13].
Once again, no matter how much the adopted ML model is valid, if the model is fed with garbage data, the model output will be garbage as well. This is the reason why feature engineering (i.e., the process of selecting and transforming the most relevant features from raw data to make an effective ML model) is a key ability of any Machine Learning engineer.
Fig. 3.2 shows the commonplace *"Garbage in, Garbage out"* in this regard.

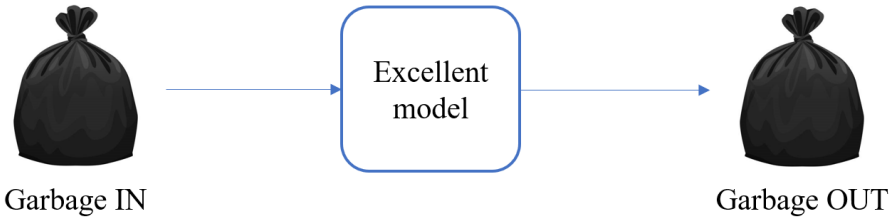


Figure 3.2: "Garbage in, garbage out" clichè

Nonrepresentative training data

The training data must be an accurate representation of the new cases so as to obtain a model which generalizes and performs successfully. If the model is trained by using a nonrepresentative training set, it will never be accurate in predictions, causing it to be biased against one class or a group of classes (in case of classification).

Overfitting and underfitting

There are many ways to introduce the concept of overfitting. Consider the training performances as the results of the football matches of a certain team preparing for the Champions League final. Even if these results are promising, it does not guarantee the team is going to win the tournament. The team may either have played versus low-level teams during training, or may have some injured players for the final match.

It is easy to draw the conclusion that a model is *overfitting* the training data when it performs well on training set but fails to generalize to new, unseen data.

Underfitting is the oppostite of overfitting, and it typically happens when the model is too simple to learn something from the data. Fig. 3.3 depicts the different cases.

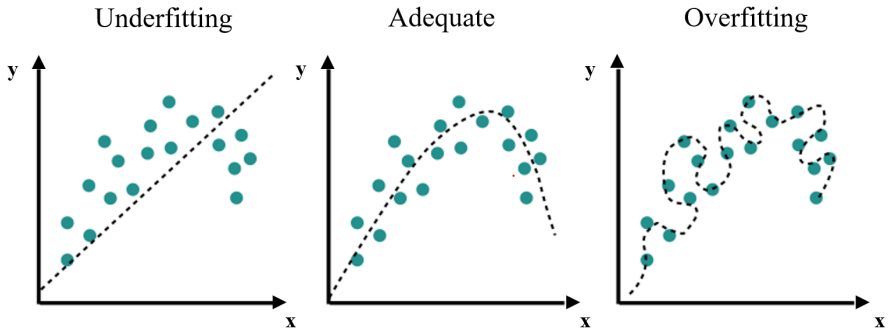


Figure 3.3: Overfitting, underfitting problems

3.2 Deep Learning

Deep Learning is a complex evolution of Machine Learning. If ML is about computers able to perform tasks without being explicitly programmed, DL goes one step further. In fact, in ML algorithms computers still think and act like machines, whereas DL takes the challenge of analyzing data and solving problems drawing conclusions in a similar way to how a human thinks.
To achieve this, DL uses a layered structure of algorithms called Artificial Neural Network (ANN). The design of ANNs is inspired by the biological neural network of the human brain, explaining the reason why we refer to DL as a subset of ML used to solve problems in a human-like way. A simple example of ANN is shown in Fig. 3.4.

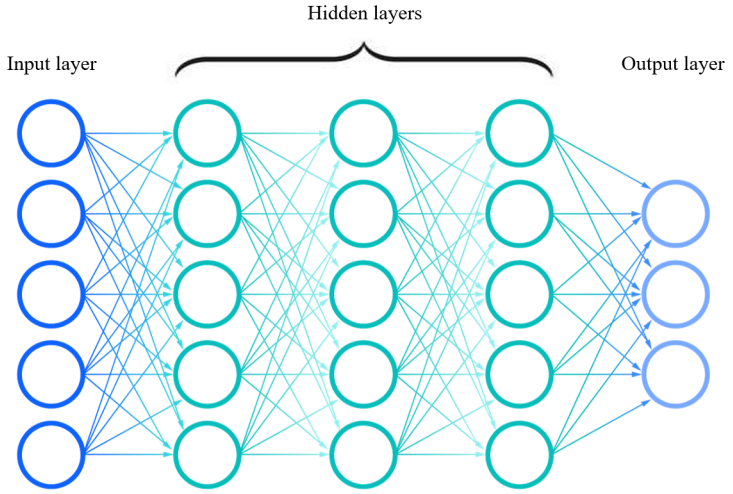


Figure 3.4: A simple Artificial Neural Network

In between the input and output layers, there are the hidden layers, so called because their values cannot be observed in the training set. In other words, the hidden layers are trained weights used by the network to perform its job. The more hidden layers constitute an ANN, the deeper the network (from which the name Deep Learning). Any ANN with at least two hidden layers is defined as Deep Neural Network (DNN).

Why Deep Learning? What motivates its adoption in a large field of contexts like automated driving, object detection, NLP and many more? Firstly, in a traditional ML algorithm, the responsible engineer would have to select and extract the relevant features (see Section 3.1.1). On the contrary, the ANN, because of its nature, performs *automatic features engineering*, requiring less human intervention. Secondly, DL can work with images, video and unstructured data in a way ML cannot simply do.
What about the drawbacks of Deep Learning? Above all, it requires a *huge amount of data* and considerable *computer power*. The former reveals the need of large training datasets, the latter the necessity of high-performance GPUs to speed up the model training. In addition, the deep non-linear transformations performed by an ANN make the model interpretability extremely abstract. To overcome this issue, Explainable Artificial Intelligence is rising interest in research in the last years, helping humans to understand the decisions made by an AI model. XAI will not be described here, as it is discussed in one of the next sections.

3.3 Deep learning for time series classification

Any classification problem, using data that is registered taking into account some notion of ordering, can be cast as a time series classification (TSC) problem [14]. Time series are present in many real-world scenarios, ranging from quantitative finance (with stocks) to medicine (in human activity recognition tasks) and cybersecurity (to spot patterns via darknets, for example). Before diving into the deep learning models for the TSC task, a set of definitions is presented first.

Definition 1

A univariate time series $X = [x_1, x_2, ..., x_T]$ is an ordered set of real values. The length of X is equal to the number of real values T.

Definition 2

A multivariate time series (MTS) with M dimensions, $X = [x^1, x^2, ..., x^M]$ is made of M different univariate time series with $X^i \in \mathbb{R}^T$.

Definition 3

A dataset $D = \{(X_1, Y_1), (X_2, Y_2), ..., (X_N, Y_N)\}$ is a collection of pairs (X_i, Y_i) where X_i could either be a univariate or multivariate time series with Y_i as its corresponding one-hot label vector. For a dataset containing K classes, the one-hot label vector Y_i is a vector of length K where each element $j \in [1, K]$ is equal to *1* if the class of X_i is j and *0* otherwise.

The magnitude of the channel impulse response can be seen as a univariate time series and every location in an indoor environment (in accordance with a selected accuracy) represents one of the K classes, referring to Definition 3.
Training a classifier on a dataset D is the objective of TSC, which aims to map the input space to a probability distribution over the values of the class variable (labels).

A strong baseline in TSC has been proved to be a nearest neighbor (NN) classifier coupled with the Dynamic Time Warping (DTW) distance function [15]. Recently, the interest moved to the development of ensemble methods, like the COTE [16], an ensemble of 35 classifiers. An extension of the COTE is the HIVE-COTE which has been proven to outperform the COVE by leveraging a new hierarchical structure with probabilistic voting [17]. HIVE-COTE has been considered the start-of-the-art algorithm for time series classification [15] in the year 2016, when evaluated on the UCR/UEA archive, one of the most popular time series dataset.

Even if HIVE-COTE reaches very high accuracy, it suffers from two main drawbacks. At first, it requires to train and optimize 37 classifiers, which may be really time consuming and even infeasible in certain circumstances. In addition, the NN is one of the key ingredient of the HIVE-COVE. By its nature, at a test time, the NN must first scan the training set before making a prediction. It causes the HIVE-COVE to require high classification time, making its deployment in real-time applications very limited.
All these reasons, together with the wide availability of GPU resources and the supreme feature of ANNs of automatic feature engineering, motivate the adoption of deep learning models for time series classification.

The deep learning framework for a generic M-dimensional TSC is depicted in Fig. 3.5.

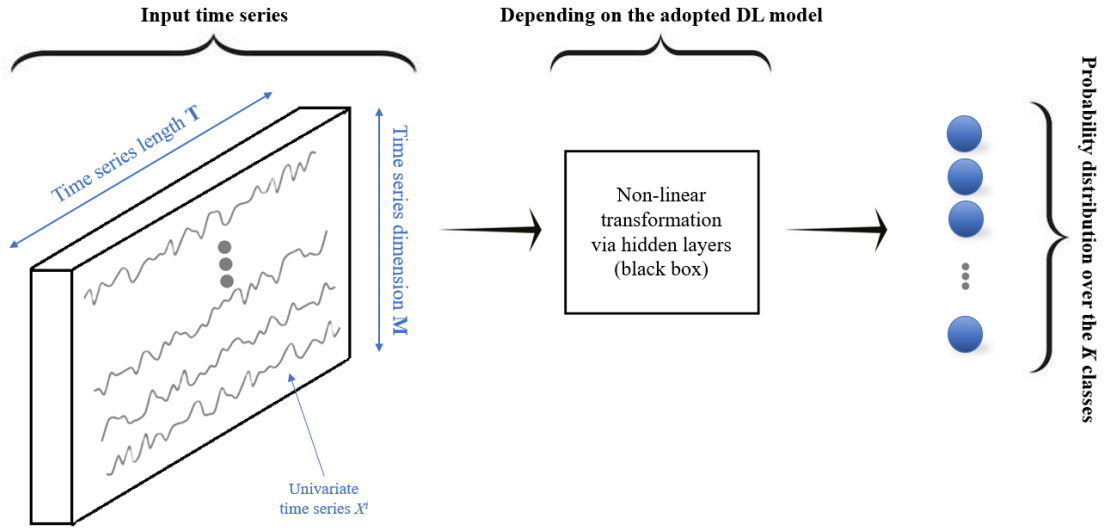


Figure 3.5: Generic deep learning framework for time series classification

3.3.1 DNN architecures for TSC

As described in Section 3.1, any ML (or DL, being a ML subset) problem needs a model. Deep learning approaches for TSC can be divided into two categories: the *generative* and *discriminative* models [18]. The former generally shows an unsupervised training step preceding the learning stage of the classifier, so as to find a reliable representation of a TS pre-training (e.g., encoders). The latter refers to a DL model which is a classifier able to directly learn the mapping between the raw input of a TS and the output probability distribution over the K classes. In

this work, the most common discriminative models for TSC are presented and discussed in detail.

Multi Layer Perceptrons

The MLP architecture represents the oldest and uncomplicated architecture among all deep learning models. Its structure is more or less equivalent to the ANN depicted in Fig. 3.4, except for an additional bias term not represented here. It is perhaps widely known as fully connected network (FCN). The network name is not random. In fact, we say it is fully connected because each neuron in a generic layer l_i is connected to every neuron in the previous layer l_{i-1}. The connections between layers are modelled as weights and they represent the way for a neural network to learn from data.

The non-linear transformation in an MLP, according to Fig. 3.5, can be written as:

$$A_{l_i} = f(w_{l_i} \cdot X + b_{l_i}) \tag{3.1}$$

where X refers to the input time series, w_{l_i} being the weights of the connections linking layers $[l_{i-1}, l_i]$, b_{l_i} the bias term, f the adopted non-linear function, A_{l_i} the output of neurons of layer l_i.
The number of hidden layers in an MLP is a choice, as well as the number of neurons in each layer, which represents a hyperparameter for the network.
If the number of hidden layers can be selected, the final layer of an MLP for TSC is commonly a softmax layer. It is an FC layer with softmax function[1] as activation function and a number of neurons equal to the number of classes K to be classified. It guarantees the sum of probabilities over the K classes to be equal to 1.
The output of the softmax function is computed as follows:

$$Prob_j(X) = \frac{e^{A_{j-1} \cdot w_j + b_j}}{\sum_{k=1}^{K} e^{A_{k-1} \cdot w_k + b_k}} \tag{3.2}$$

where $Prob_j(X)$ represents the probability of input X (having class Y) to belong to class j over the K classes. The weights w_j and the bias b_j are connected to the activation in layer l_{j-1}.

[1]The softmax function converts a vector of K real numbers into a probability distribution of K possible outcomes.

The main drawback of an MLP is that it does not show any invariance. It means that the network does not preserve the temporal information of the time series, processing the TS elements independently. This may cause to fail the classification when the time series appearance varies in some way (e.g. delayed in time). The MLP network represents in any case a strong baseline for TSC and deep learning in general, which has lead to the development of more complex and effective models.

Convolutional Neural Network

As any ANN, a CNN consists of an input layer, a certain number of hidden layers and an output layer. In a MLP, any neuron in a certain hidden layer is connected to all neurons in the previous one. On the contrary, in a CNN the hidden layer is generally a convolutional layer, the core building block of any CNN. Differently from MLPs, where weights are trained and modelled as interconnections between neurons, in a CNN there is a *filter* (or kernel) which slides over the input performing convolutions. The result of a convolution is a feature map. A graphical representation of the convolution operation over an image is shown in Fig. 3.6, taken from [19]. The optimal values for the filters (weights of the network) are computed via network training, as for any other DL model.

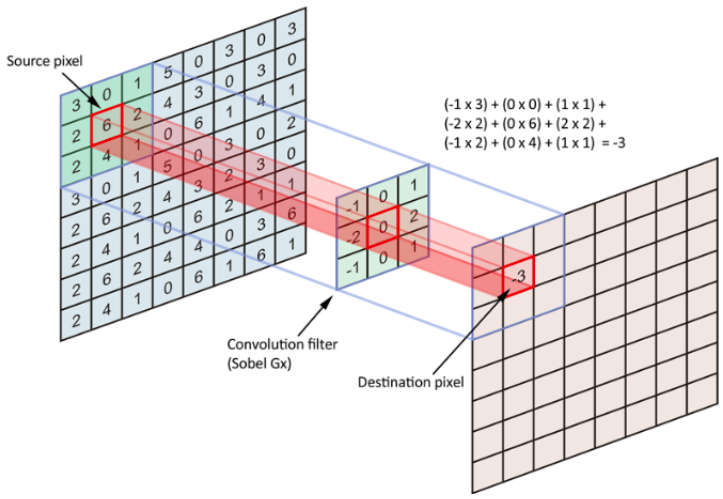
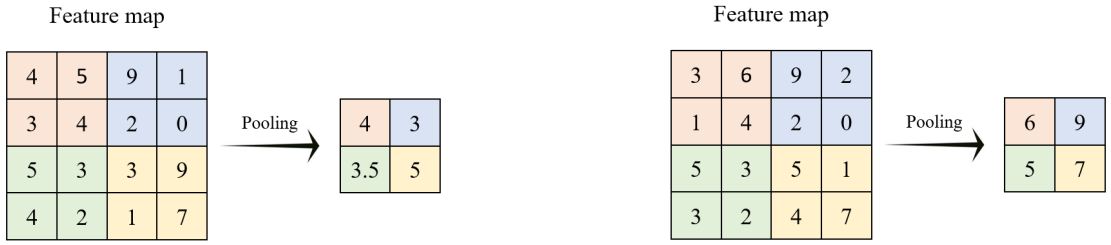


Figure 3.6: Generic convolution operation on an image

Differently from images, the filter in a convolution operation over a univariate time series slides along only one dimension (time $\rightarrow$ Conv1D) instead of two dimensions (width, height $\rightarrow$ Conv2D). Denoting with X the input time series, the output of the convolution on X is another time series X' which has undergone filtering. As a consequence, applying n convolutions (filters) over a time series

X results in a MTS X' whose dimension is equal to the number of applied filters n.

A CNN is not made of convolutional layers only. In between the input and output layers, other operations generally occur. The output feature maps are potentially sensitive to the location of the features in the input. To reduce this issue, a *pooling* operation may be employed, so as to down sample the feature maps, making them more robust to changes in the location in the input, promoting the invariance. Pooling layers down sample the feature maps by condensing the features in blocks. Two common methods are average pooling (Fig. 3.7a) and max pooling (Fig. 3.7b) which compress the average existence of a feature and the most activated one, respectively.



(a) Average pooling (b) Max pooling

Figure 3.7: Different pooling operations

Average and max pooling are local pooling techniques. Global pooling operations exist too. Making use of a global pooling operation on a time series results in a down sampling to a single real value. Usually a global aggregation is adopted to reduce drastically the number of parameters in a model thus decreasing the risk of overfitting, while enabling the use of CAM to explain the model's decision [20]. A CNN that does not contain any local pooling and with a GAP layer instead of a traditional FC layer before the output softmax layer is traditionally referred as Fully Convolutional Neural Network (FCN), firstly proposed in [21]. Since a FCN does not contain any local pooling, the input time series length remains unchanged throughout all the convolutions, under the assumption of same padding in the network configuration.

In the middle of convolutional layers, batch normalization[2] layers are typically included in DL architectures. For time series data, the batch normalization operation is performed over each channel preventing the internal covariate shift[3] across

[2]Batch normalization is a procedure employed to train ANNs in a faster and more robust way through normalization of layers' inputs by re-centering and re-scaling.

[3]The Internal Covariate Shift is defined as the change in the distribution of network activations

one mini-batch training of time series [22].

An example of FCN is depicted in Fig. 3.8. It is constituted of an input

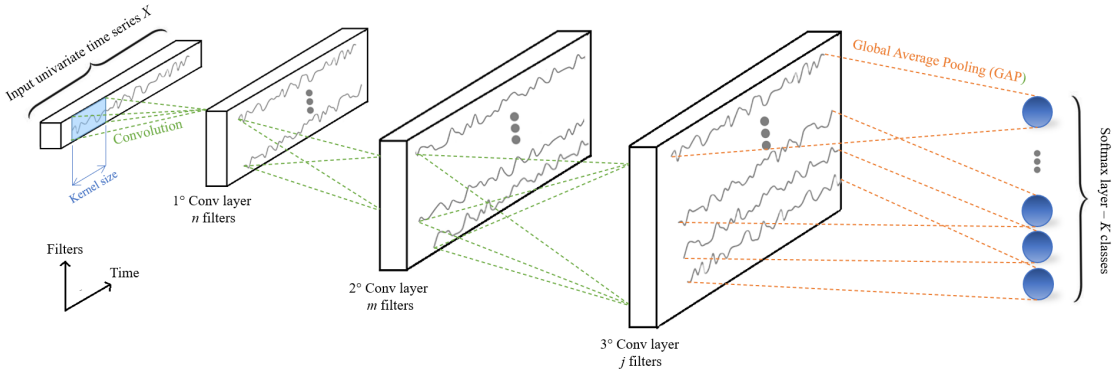


Figure 3.8: Fully Convolutional Neural Network for Time Series Classification

layer (accommodating a univariate time series), three convolutional layers (with a progressively increasing filters number $n < m < j$), a Global Average Pooling layer and a final softmax layer with K neurons (classes).

Residual Neural Network

A Residual Neural Network (ResNet) can be seen as an extension of a standard CNN. A standard CNN (or ANN in general) "simply" tries to find the direct mapping between input and output via weight layers and applying non-linear activation functions (commonly a ReLU). On the contrary, the ResNet implements the *skip connection* mechanism, or identity mapping. It is used to jump over some layers, adding the original input to the output of the weight layer. Fig. 3.9 shows the comparison between any ordinary ANN implementation (right) and the ResNet building block (left):

due to change in network parameters during training. In an ANN, the output of a layer feeds the next one. This is from input to output. When the parameters of a layer change, the distribution of inputs to subsequent layers does the same. These shifts in input distributions may be problematic during the network training, even more if the number of layers is large. Batch normalization helps to mitigate this effect.

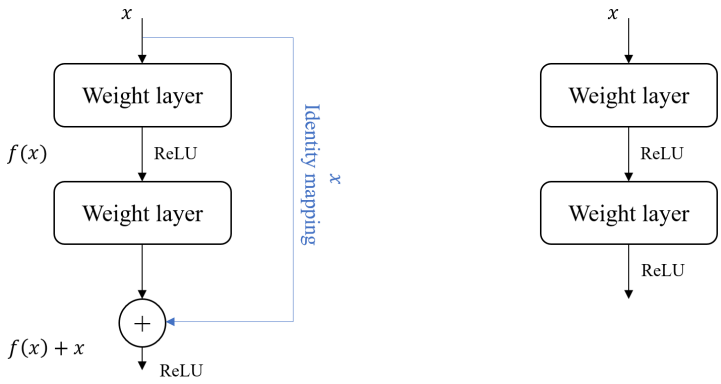


Figure 3.9: Comparison between the ResNet building block (left) and standard ANN implementation (right)

ResNets help preventing the vanishing gradient problem[4], making the training of the neural network much easier.

With reference to Fig. 3.8, the residual building block can be added as follows:

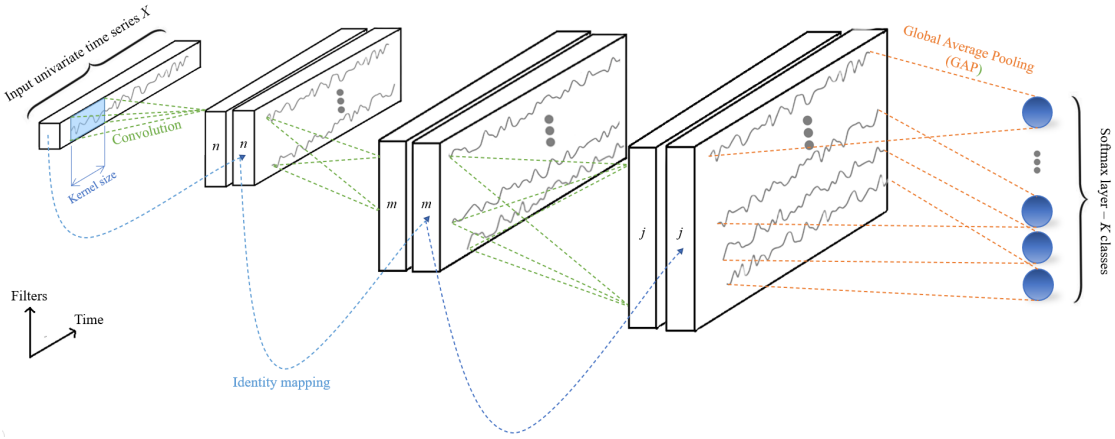


Figure 3.10: ResNet architecture for TSC

[4]In ML, the vanishing gradient problem is faced when training ANNs with a gradient-based learning technique and backpropagation. In these methods, the ANN's weights are updated proportionally to the partial derivative of the error function with respect to the current weight, via the chain rule. Sometimes the gradient vanishes just a little, causing the weights to change in a tiny way.

3.4 Time series data augmentation techniques

The top-notch performance of deep neural networks extremely relies on large training dataset to avoid overfitting issues. However, in many real-world cases, data may be limited either because can not be collected (e.g. in the medical field) or because their collection is really time consuming (as in the task treated in this work). As a consequence, data augmentation is of primary importance to enrich the size and quality of the training dataset so as to make the DNN successfully working.

The basic idea of data augmentation is to generate synthetic dataset covering unexplored input space while maintaining correct labels [23]. Data augmentation techniques are task dependant (some methods may apply for classification but not for regression) and input type dependant (time series are intuitively different from image data, since they exhibit the temporal dependency property).

Time series data augmentation methods can be divided into *basic* and *advanced* approaches. The former include time series manipulation in time and frequency domains. The latter are more complicated and typically learning-based ones, taking into account the data augmentation methods should be capable of mimicking the features of the real data while generating synthetic ones. In the following, some of these methods are presented and detailed for the classification task.

3.4.1 Basic approaches

Window slicing

A first basic method is known as *window slicing*. As the name suggests, it consists in extracting slices from the raw time series and performing classification with the extracted slice. The synthetic generated dataset is composed of N sliced time series where each slice is extracted from a random raw time series and labelled with the same class of the original one. The size of the slice is the only parameter of this technique. The classifier is then learned on the augmented dataset.

Window warping

A second basic technique is named *window warping*. It works in time domain and manipulate the time series directly. It consists in warping a randomly selected slice by speeding it up (up-sampling) or down (down-sampling). Of course, it generates synthetic time series of different lengths, according to the chosen sampling. To deal with that, a cropping window is then used on the artificial time series to have

all the same length. The working principle is shown in Fig. 3.11, with reference to a UWB channel impulse response.

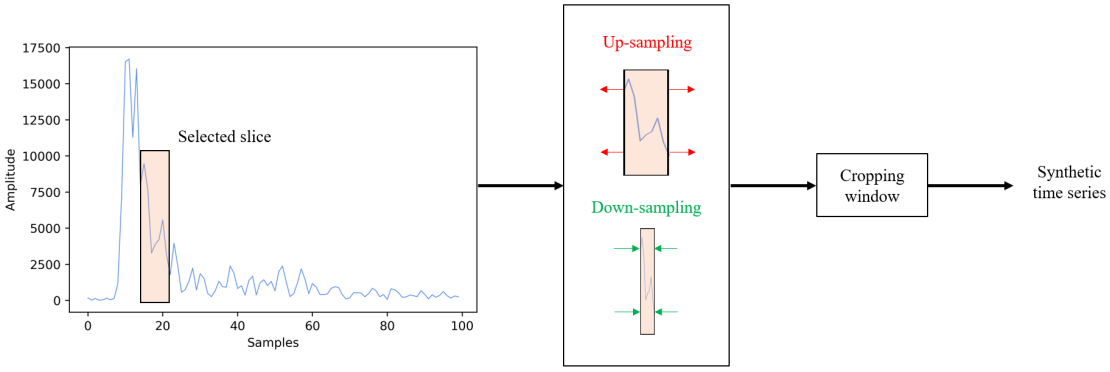


Figure 3.11: Window warping data augmentation method on an example CIR

Gaussian noise

Even if one typically refers to the addition of Gaussian noise on top of a raw time series as a way to improve network generalization, injecting noise to the input of a neural network can also be seen as a form of data augmentation [24]. The addition of Gaussian noise to the inputs of a neural network is traditionally referred as *jittering*, after the use of the term in signal processing to refer to the uncorrelated random noise in electrical circuits.

The amount of added noise (i.e. the spread of standard deviation, considering zero mean value) is a configurable parameter. Too little noise has no effect, whereas too much noise makes the input time series exceedingly different from the original raw one. For this reason, the amount of noise has to be carefully chosen so as to avoid to make the mapping function too challenging to learn on one hand, and to make it effective on the other.

Linear combination

Another possible data augmentation technique is performing linear combinations between samples. In particular, for each pair of neighboring samples, a new sample is generated which is the linear combination of them (according to a chosen multiplication factor, hyperparameter of the method). This technique is exploited in [25], where the data augmentation is performed in the learned embedding space. This technique is proven to be particular useful for time series classification, which is perhaps the same task to be carried out in this work.

3.4.2 Advanced approaches

Advanced approaches can be further subdivided in: decomposition methods, statistical generative models and learning methods. Even if they have not been adopted in this work, the most recent and effective advanced method for time series data augmentation is presented for completeness.

TimeGAN

Deep generative models (DGMs) have recently risen a lot of interest in the research community, being able to generate near-realistic high-dimensional data, especially for images and sequences (text, audio). They can be extended to time series as well. Among DGMs, the generative adversarial networks (GANs) are the most popular AI technique to produce synthetic samples, increasing the training set preserving the real characteristics of the data. In this regard, recently the authors in [26] proposed Time-series Generative Adversarial Network (TimeGAN), a new framework for generating realistic time-series data. TimeGAN consists of four network components: an embedding function, a recovery function, a sequence generator and a sequence discriminator. The key feature of TimeGAN is that the autoencoding components (first two) are trained jointly with the adversarial components (latter two), such that TimeGAN simultaneously learns to encode features, generate representations and iterate across time, preserving the temporal dynamics of the time series. It uses both supervised and unsupervised losses. Fig. 3.12 shows the TimeGAN block diagram and training scheme, according to the nomenclature defined in [26].

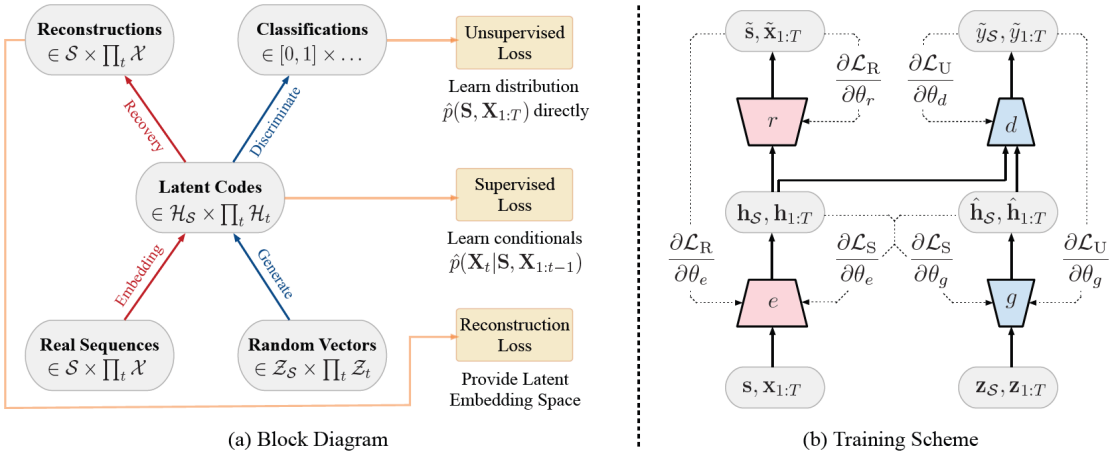


Figure 3.12: (a) Block diagram of network components and objective functions; (b) Training scheme: solid lines indicate forward propagation, dashed lines indicate backpropagation of gradients

3.5 Explainable Artificial Intelligence

Because of their highly non-linear structure, one of the main drawbacks of deep learning models is that they are usually black-box models. It means that, even if they are successful, it is unclear what information the model uses to make its predictions. *Explainable Artificial Intelligence* (XAI) is the subset of AI which attemps to provide interpretability to these black-box structures. The main goals of XAI are:

- *system verification*, since in many applications one cannot trust a black-box model;
- *learning from the system*, because today's AI agents may observe patterns undetectable by humans;
- *regulation*: who has the responsibility when the AI makes a wrong decision?
- *system improvement*, since to enhance the model performances, it is crucial to make it more interpretable

Among all XAI objectives, this work includes a XAI algorithm for system improvement.

3.5.1 Class Activation Mapping

The *class activation map* (CAM) algorithm was firstly introduced in [27] to identify the image regions which contributed most to a certain classification. It can be basically extended to TSC as well, since the only assumptions are to deal with a CNN adopting a GAP layer preceeding a softmax output layer. If the hypotheses are satisfied, the weights of the output softmax layer can be projected back onto the convolutional feature maps to identify the importance of certain features over others (e.g., most significant image or time series regions for a certain classification). This is what it is meant by class activation mapping. Fig. 3.13 shows the CAM for a network which last convolutional layer is made of 3 filters.

As described in Section 3.3.1 and represented in Fig. 3.13, the GAP layer averages the whole feature map of the last convolutional layer into a single value. Then, a weighted sum of the GAP output is used to generate the final output via the softmax layer.
Considering the TSC problem, let $b(t)$ be the activation of the last convolutional layer. It is a MTS with N variables, where N is the number of filters of the last convolutional layer. As a consequence, $b_n(t)$ is the univariate time series obtained by applying the *nth* filter, where $n \in [1, N]$. The result of GAP on $b_n(t)$ is

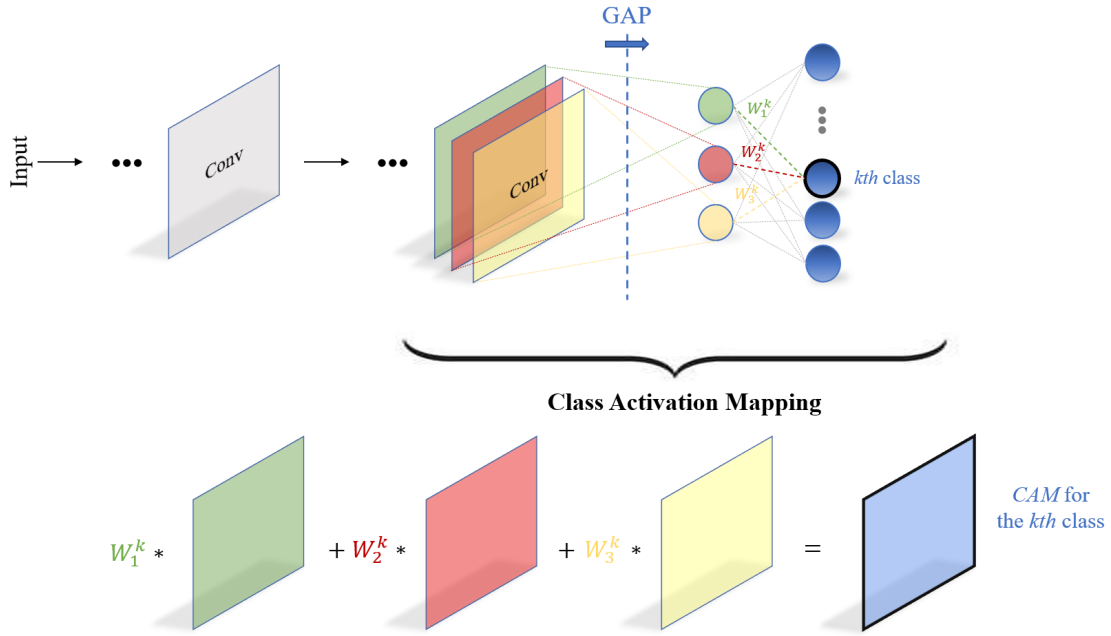


Figure 3.13: Class Activation Mapping: graphical-based representation of the algorithm

$B_n = \sum_t b_n(t)$. So, for a given class k, the input to the softmax is $S_k = \sum_n w_n^k B_n$. The weight w_n^k simply denotes the relevance of B_n for class k. By merging all together:

$$S_k = \sum_n w_n^k \sum_t b_n(t) = \sum_t \sum_n w_n^k b_n(t) \tag{3.3}$$

Finally, the class activation map (CAM) explaining the region of the TS used most to perform the classification for the class k is given by:

$$CAM_k = \sum_n w_n^k b_n(t) \tag{3.4}$$

The result of the CAM for a TSC task is a univariate time series where each value (at every time stamp t) is equal to the weighted sum of the N data points at t, with the weights being learned by the neural network [14].

Chapter 4

Experimental set

The objective of this work is to validate the idea of indoor environment characterization via CIR obtained adopting the UWB technology, under the hypothesis that every environment is characterized by its own electromagnetic firm. To do so, at first a set of data is needed to perform analyses. Furthermore, other data are required for network training and testing. To accomplish this task, a set of UWB sensors has been selected and some software has been used to collect and store such data. In the following sections, the chosen hardware is shown, together with its characteristics and issues. The software used to collect the data is presented as well. Finally, the measurement scenario is shown, as well as the UWB boards configuration.

4.1 Hardware and Software

4.1.1 Decawave EVK1000 kit

Among all the UWB devices available in the market, it has been selected the EVK1000 evaluation kit from Decawave. As demostrated in [28], this system shows the best performances when compared to other commercially available UWB systems like the ones of Ubisense and BeSpoon. Each EVK1000 kit contains two EVB1000 boards (see. Fig. 4.1) equipped by a DW1000 CMOS radio tranceiver IC compliant with the IEEE 802.15.4 UWB standard, a STM32F105 ARM Cortex M3 μprocessor, a USB interface, an LCD display, an antenna and a set of jumpers and switches for board configuration. The system reaches an accuracy of $\pm$10 cm using TWR TOF measurements [10]. It spans 6 RF bands from 3.5 GHz to 6.5 GHz and supports data rates of 110 kbps, 850 kbps and 6.8 Mbps.

Using the dip switches it is possible to choose between two channels (2 and 5) among the 6 available ones. They have two different centre frequencies (3.99 GHz

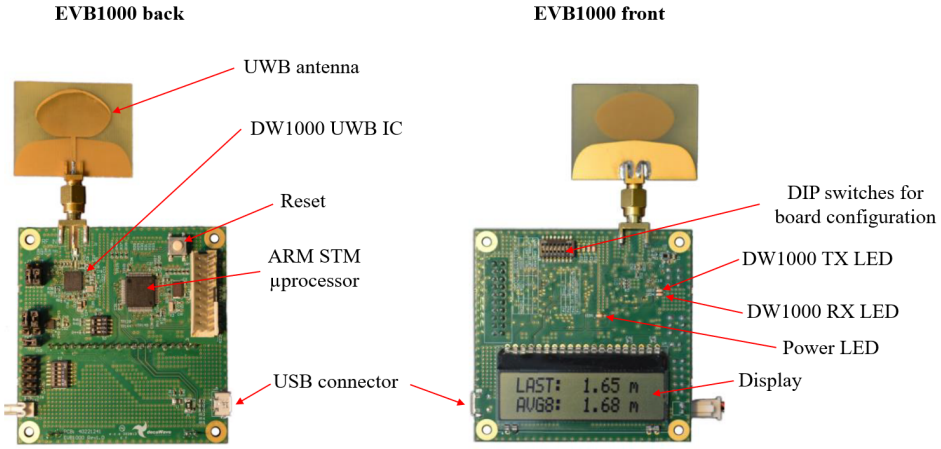


Figure 4.1: EVB1000 board: back and front views

and 6.49 GHz, respectively) and the same bandwidth (499.2 MHz). The default board configuration is for channel 2. More details about channel configuration are reported in Table 4.1.

Channel Number	**Centre Frequency (MHz)**	**Bandwidth (MHz)**	**Band (MHz)**
1	3494.4	499.2	3244.8 - 3744
2	3993.6	499.2	3774 - 4243.2
3	4492.8	499.2	4243.2 - 4742.4
4	3993.6	1331.2*	3328 - 4659.2
5	6489.6	499.2	6240 - 6739.2
7	6489.6	1081.6*	5980.3 - 6998.9
*The DW1000 has a maximum receive BW of 900 MHz			

Table 4.1: UWB IEEE 802.15.4 UWB channels suported by the DW1000

4.1.2 Mileseey X6 laser distance meter

In order to collect the Channel Impulse Responses for a certain number of locations in an indoor environment, a laser distance meter is needed. In particular, the Mileseey X6 one has been selected. Fig. 4.2 shows the instrument and Table 4.2 reports the technical specifications.

Figure 4.2: Mileseey X6 laser distance meter

Measuring accuracy	±2 mm
Measuring range	0.16m - 70m
Measuring speed	< 0.5 seconds
Measuring units	m/ft/in
Laser class	II
Laser type	635 nm < 1mW

Table 4.2: Mileseey X6 laser distance meter technical specifications

4.1.3 DecaRanging

The *DecaRanging* PC application offers an alternative to the embedded DecaRanging application preinstalled on the on-board ARM microntroller of each EVB1000 board. The DecaRanging PC application drives the DW1000 UWB IC allowing to perform TWR with another DW1000 module, defining how to exchange messages between the pair of devices, calculate the ToF, display the resultant distance and much more. The ranging method used by the DecaRanging software is the three messages DS-TWR explained in Section 2.6.2. Fig. 4.3 shows the software GUI.

When the application is run, it is set in listener mode. First of all, the pairing between the two boards is needed, so as to set one board as the tag, and the other one as the anchor. Once the pairing is done, the ranging algorithm can be started. At the top of the main window, some statistics are reported. They include the count of successfully sent and received frames (TX, RX), the instant ToF, the average distance over 8 or long-term measurements.
The configuration button allows to select the channel (among the 6 available ones), the preamble length, the data rate, the PRF and the timing the two DW1000 modules use to exchange messages during the TWR measurement.

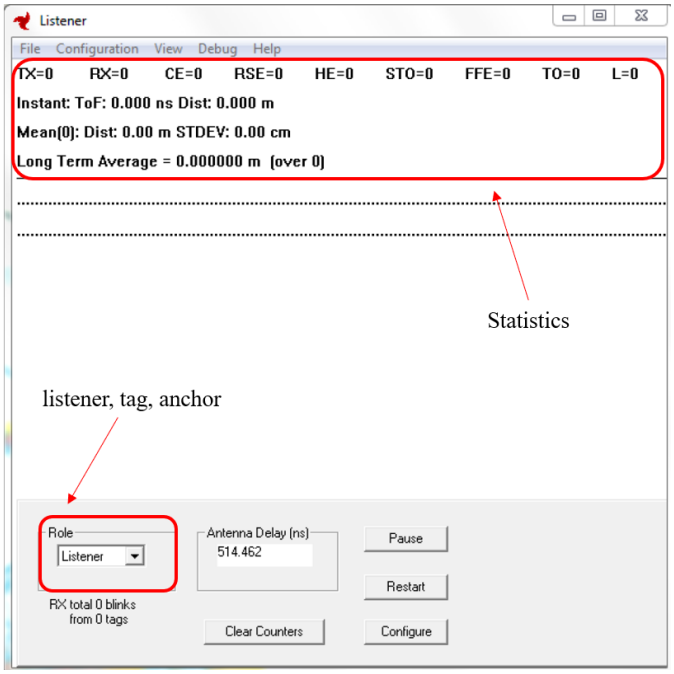


Figure 4.3: DecaRanging PC application

Similarly, the view button gives the possibility to enable the view of the Channel Impulse Response (detailed in Section 2.5.4). The DW1000 measures the CIR upon RX packets with a clock frequency of $f_s = 998.4\ MHz$ corresponding to a sampling period of $T_s = 1/f_s = 1.0016\ ns$ and it stores it in an internal buffer. The time span of the CIR is the duration of the preamble symbol, corresponding to 1016 samples for a 64MHz PRF or 992 for a 16MHz PRF. Each sample is a complex number whose real and imaginary parts are 16-bit signed integers. An example of extracted CIR from the board for a 64MHz PRF is depicted in Fig. 4.4.

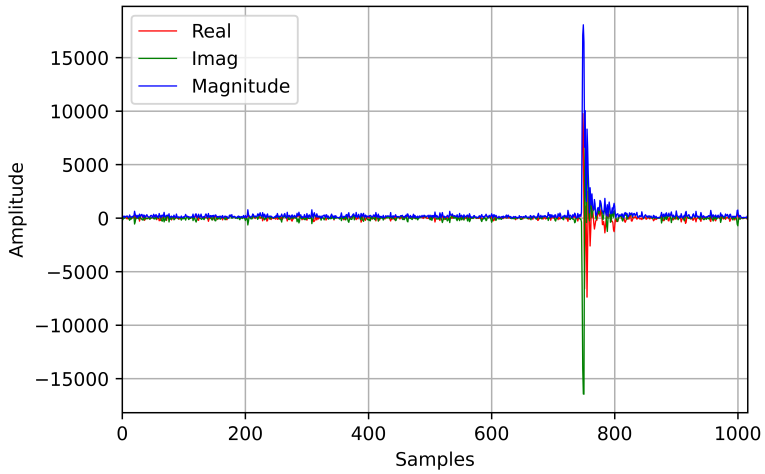


Figure 4.4: Channel Impulse Response view - 64MHz PRF

The debug button allows to automatically record the measurements in a *.log* file. The way the *.log* file is formatted is reported in Appendix A.

4.2 Board issues and calibration

There are two fundamental sources of error in a DW1000 based TWR scheme. The first one is related to clock drift in the two nodes. In a typical TWR scheme, the error is proportional to T_{reply} (see Section 2.6.1). It can be reduced adopting the three messages DS-TWR scheme (SDS-TWR), as it is in the DecaRanging software. In this case, the error in the ranging becomes proportional to $\Delta_{reply} = T_{reply2} - T_{reply1}$, making it much smaller (refer to Section 2.6.2).
The second source of error depends on the incident signal level at a node. Ideally there should be no relationship between the timestamp of a received signal and the Received Signal Level (detailed in Appendix A). In practice, one can observe a bias which varies with the Received Signal Level (RSL). This behaviour is illustrated in Fig. 4.5 for channels 1,2,3,5 having 500 MHz bandwidth.

Since the effect of range bias is actually dependant on the RSL level at the pins of the chip, this error is affected by the antenna gain. For this reason, a precise antenna delay calibration is needed. Table 4.3 below lists the calibration distances used for the different channels and for a PRF of 64 MHz.

To collect measurements, channel 5 has been selected with a PRF of 64 MHz. The obtained antenna delay value, positioning the two boards 5.0m apart, has been 516.025 ns. According to Fig. 4.3, it can be configured in the DecaRanging application before starting to take measurements.

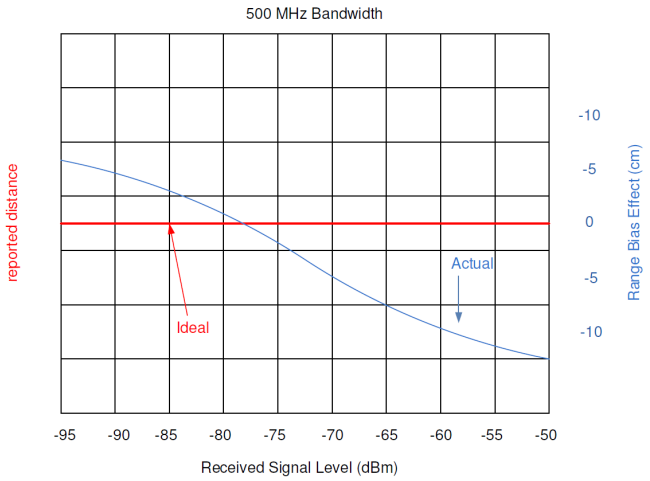


Figure 4.5: Effect of range bias on the reported distance

Channel Number	f_c **(MHz)**	**Bandwidth (MHz)**	**Calibration distance (m)**
2	3993.6	499.2	12.9
3	4492.8	499.2	7.2
4	3993.6	900	8.7
5	6489.6	499.2	5.0
7	6489.6	900	5.3

Table 4.3: Calibration distance for DW1000 channels and PRF of 64 MHz

The DecaRanging souce code includes a range bias adjustment to compensate for this effect.

4.3 Measurement campaign

Having clarified the hardware and software adopted to collect the measurements, the measurement campaign carried out to collect the dataset is now presented. At first, the boards configuration is detailed. Next, the measurement scenario is shown. Finally, the data extraction is presented. After that, everything will be ready for data analysis and network training and testing, detailed in the next chapters.

4.3.1 Board configuration

Before starting to collect the measurements, the two EVB1000 boards have to be configured. It is possible to configure the board via either the dip switches or the DecaRanging PC application. The former allows to a lower number of possibilities. Depending on the configuration, it is possible to maximise the range of communication, to allow for more energy through the channel, to increase the data rate and much more. For the thesis objective, it is important to retrieve the CIR, ToF and SnR in a robust way. In principle, it can be with any configuration. For this reason, the boards have been configured with the default values, despite the channel (allowing for a higher centre frequency) and the antenna delay. Table 4.4 reports the details about the configuration of each EVB1000 board.

Channel number	5
Centre frequency	6498.6 MHz
Bandwidth	499.2 MHz
Antenna delay	516.025 ns
PRF	64 MHz
Data Rate	110 kbps
Preamble length	1024
Tag blink period	1000 ms
Tag poll period	1000 ms
Tag response delay	200 ms
Anchor response delay	150 ms
SFD	Non standard

Table 4.4: EVB1000 configuration settings

In order to collect the measurements, two EVB1000 boards are used. One is power supplied and set to be the tag, the other is connected to the laptop via a USB cable and set as anchor, working with the *MoDecaRanging* PC application detailed in Appendix A. This is needed to store the measurements.

4.3.2 Measurement scenario

The dataset have been collected in the hall of the Department of Electrical and Control Engineering of the engineering building of the California State University, Los Angeles. This room has been chosen because it includes some of the typical features of an indoor environment (see Section 2.5), including static elements (walls

and objects) and dynamic ones (people walking around interacting with doors and elevators). Fig. 4.6a shows the environment and Fig. 4.6b shows the boards configurations.

(a) Measurement environment (b) Boards setup in the measurement scenario

Figure 4.6: Measurement environment (a), EVK1000 setup in the measurement scenario (b)

In a typical scenario where UWB is exploited for indoor positioning localization, the tag moves and the anchor does not. In the measurement scenario considered in this work it is the opposite, because the CIR reconstruction happens at the anchor side. In the measurement setup, the tag is fixed on the glass cabinet while the anchor is moved all around the room. Nevertheless, it does not matter since the interest is just to collect enough measurements in a robust way, so as to validate the hypothesis of characterization of an indoor environment via channel impulse responses.

In order to cover the whole area, a grid has to be defined. It has been chosen a 80x100cm grid where every point has been numbered (1-to-41) from the lower-right corner, referring to the floor plan in Fig. 4.7. It has been found to be a trade-off between the density of points and the time needed to collect the data.

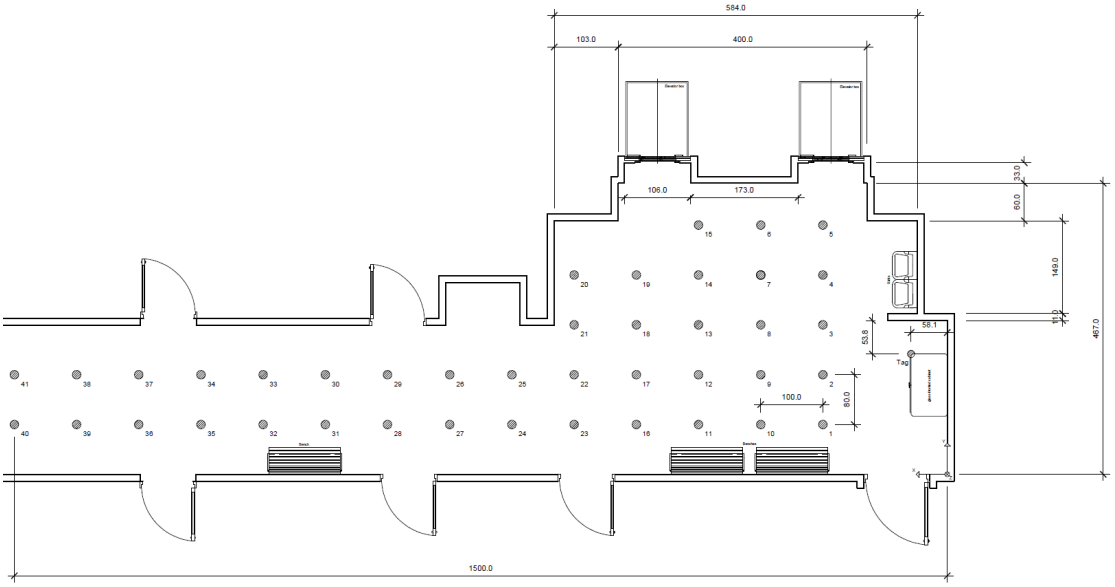


Figure 4.7: Floor plan grid - 80x100cm

4.3.3 Data extraction

The MoDecaRanging PC application produces a *.log* file in a specific format, according to Appendix A. Collecting thousands of measurements for every point, and collecting them for many points (according to the grid defined in Fig. 4.7), an automatic extraction of the useful data is needed. To do so, a Python script reads the .log file, saves the valuable data into a dictionary structure and a pickle[1] file is generated, so as to have the dataset ready for later analyses.
For each point, defining n as the number of successfully recorded frames with no errors, the dictionary is structured as in Fig. 4.8.

CIR imag (1016,n)	CIR real (1016,n)	ToF (n/2)	Distance (n/2)	FP_AMPL1 (n)	FP_AMPL2 (n)	FP_AMPL3 (n)	RSL (n)	SNR (n)	FPhw (n)	...

Figure 4.8: Extracted data - structure of each point samples

Since the SDS-TWR ranging method is implemented, there are half ToF measurements with respect to the reconstructed channel impulse responses. Every CIR is composed of 1016 samples with a sampling period of $T_s = 1.0016\ ns$, according to Section 4.1.3 and 4.3.1.

[1]The pickle module implements binary protocols for serializing (pickling) and de-serializing (unpickling) a Python object structure.

Chapter 5

Data collection and analysis

Having clarified the measurement scenario and the data structure of the samples, the next step is about analysing the collected data. At first, a description about the collected dataset is presented. Then, a channel impulse response analysis is carried out to show the differences between one point and another in the environment, also varying with the introduced dynamics. Finally, a the results obtained from a Principal Component Analysis are shown.

5.1 Collected dataset

The measurements have been recorded according to the grid defined in Fig. 4.7. In particular, measurements from four different days compose four different dataset. Of course, it is possible to merge all these ones in a single one but it has been preferred to leave them separated for certain reasons that will be detailed later. In the following, it is possible to refer to such dataset as:

1. *Ideal*, the first collected dataset, composed of 41 points, $n \approx 800$;
2. *Real*, the second dataset, made of 13 points, $n \approx 2800$;
3. *Realv2*, the third one, constituted of 13 points, $n \approx 1000$;
4. *Realv3*, the last one, composed of 13 points as well, $n \approx 1000$.

where n is the number of successfully recorded frames with no errors for each point, as in 4.3.3. For instance, the total number of samples for the *Ideal* dataset is around $41 \times 800 = 32800$.

The *Ideal* dataset covers the whole grid in Fig. 4.7 and it has been recorded in static conditions only, i.e., no dynamics is introduced in the environment. Differently, the *Real, Realv2, Realv3* dataset have been collected in both static and

dynamic conditions, i.e., the recorded samples include most of the typical indoor environment features (reflections from walls and objects and people-environment interactions).

The *Ideal* dataset has been used to carry out a first analysis only, in terms of distributions and power delay profiles. The indoor characterization of the environment in static conditions is definitely easier. Each CIR is more or less the same, since the environment does not change and there is no external factor influencing it.

Of course, the CIR changes as the environment changes. The channel impulse responses for different environments are intuitively different, being different the environment itself. At the same time, the presence of dynamics makes the CIR dissimilar one to each other, even considering the same location. To collect measurements in a more robust and realistic way, the dynamics introduced by people is needed. This reason motivates the collection of the other dataset *Real, Realv2, Realv3*.
In particular, the *Real* dataset will be used for the analyses in the next sections. The samples, for each point, have been collected in the following way:

- The first 800 samples are in static conditions (same as in the *Ideal* dataset);
- The next 800 samples have been recorded in a dynamic condition, with people walking around the room;
- The following 400 samples refer again to a static condition, this time having introduced new static elements (e.g. person sit on a bench, employee waiting for the elevator, student refilling the bottle at the sink, opened door);
- The next 400 samples have been taken rotating the antenna of the anchor (clockwise for the first 200, anti-clockwise for the others);
- The last 400 samples include the dynamics introduced by people in the environment and around 100 samples refer to NLOS conditions where a human body was immobile in between the theoretically LOS of the two boards.

The *Realv2* dataset has been collected some days after the *Real* one and it includes both static and dynamic conditions, this time in a random way. Finally, the *Realv3* is even more random, being collected after the fall semester began. It means many more people were introducing dynamics interacting with the environment, making the collected samples realistic of an indoor scenario.

The thirteen locations of the *Real, Realv2, Realv3* dataset are a subset of the 41 locations of the *Ideal* one, according to Fig. 5.1.

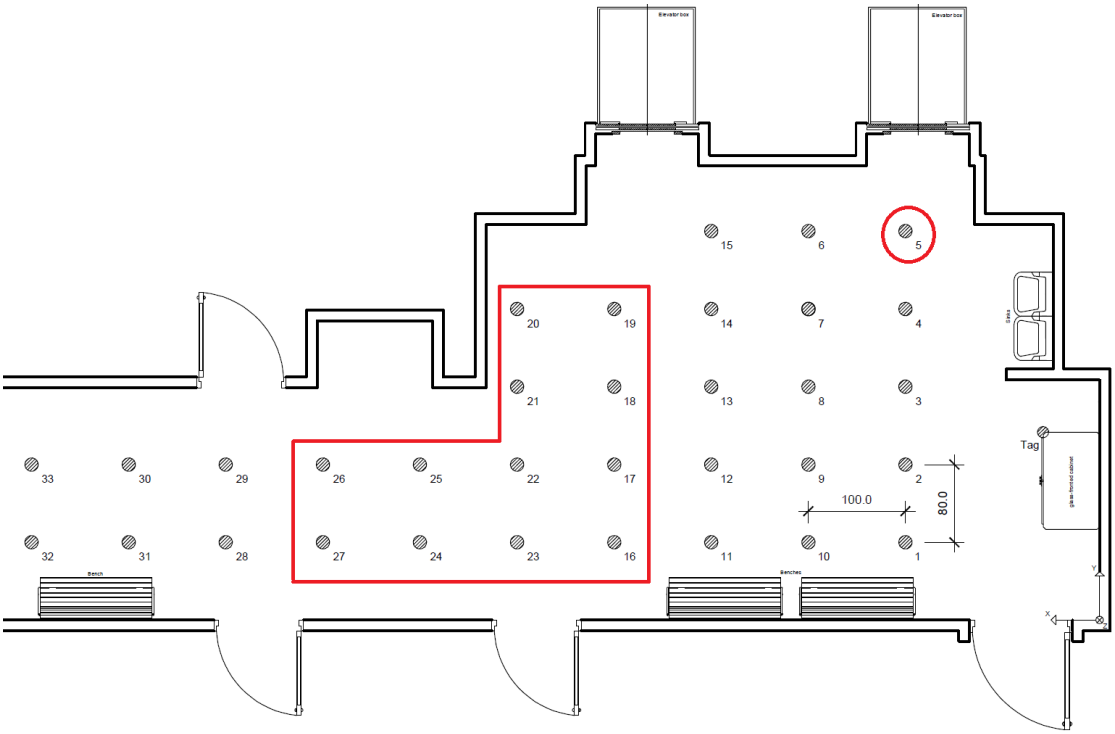


Figure 5.1: Highlight on a grid portion - dataset *Real, Realv2, Realv3*

Collecting measurements is particularly time consuming. This is the reason behind the choice to reduce the analysis from the 41 points to 13 only. According to Fig. 5.1, the analysis include part of the hall and the corridor (points 16-29) and point 5, which is partially in NLOS because of the presence of a wall obstructing the LOS between the tag and the anchor. All other points are in LOS.

Fig. 5.2 shows the number of samples n for each of the 13 points (classes) for the *Real, Realv2, Realv3* dataset. Note that every class has more or less the same number of samples. It is because particularly attention has been placed to avoid any class-imbalance in the dataset collection.

In the following chapter, *Real* and *Realv2* dataset are merged together, data augmented and used for network training. The entire *Realv3* dataset is used as a test set for the network, so as to feed the DNN by totally unseen data. The *Ideal* dataset will be no longer considered, since it refers to static conditions only.

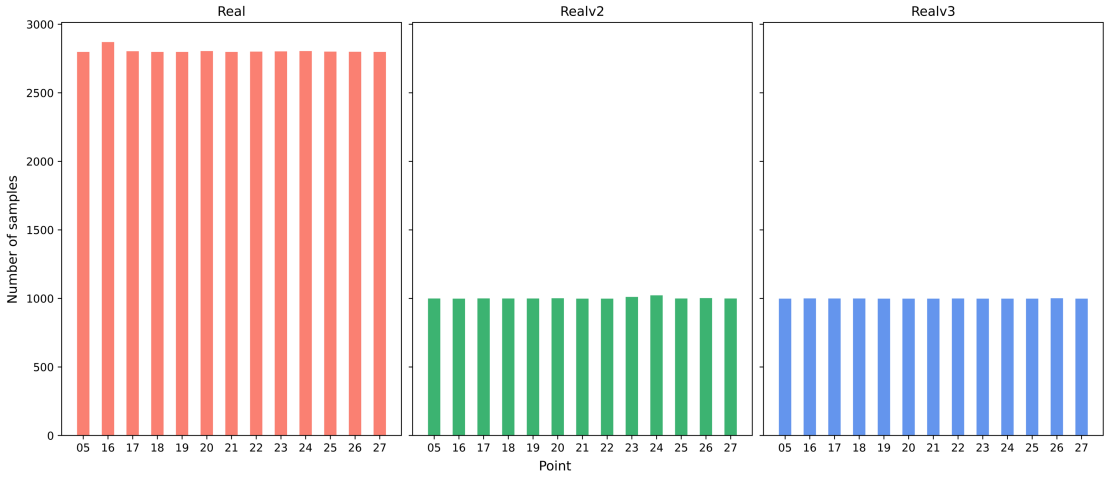


Figure 5.2: Number of samples per point - dataset *Real, Realv2, Realv3* (from left to right)

5.2 Channel impulse response analysis

As already highlighted many times, the channel impulse response is the key element of this work, since it is used as input for a DNN in order to characterize an indoor environment. For this reason, an analysis about the minimum number of CIR samples carrying the information of the indoor environment electromagnetic firm has been performed. Furthermore, the differences of CIRs among different points and the influence of the dynamic factors on the channel impulse responses has been investigated.

The CIR is stored in an internal buffer of the DW1000 and published in the *.log* file in the form of (real,imag) samples via the MoDecaRanging PC application. Then it is extracted and saved via a Python script as detailed in 4.3.3. The real and imaginary components of the CIR simply corresponds to the in-phase and quadrature parts of the signal (refer to 2.5.4). The proposed method uses the amplitude of the CIR, computed as:

$$|CIR| = \sqrt{CIR_{Real}^2 + CIR_{Imag}^2} \tag{5.1}$$

5.2.1 Samples selection

According to 4.1.3 and 4.3.1, the length of an entire CIR is 1016 samples. Looking at Fig. 4.4, it is possible to intuitively conclude that just a portion of the extracted CIR matters. In fact, all samples before Fphw[1] and after a certain index are noise floor. The number of bins holding most of the information available about the propagation characteristics in the environment is equal to 152, starting from the first path index [29]. It is validated in Fig. 5.3 where neither point 5 (the "worst" one in terms of multipath propagation) exceeds the threshold considered by the LDE algorithm[2] after the 152^{th} bin.

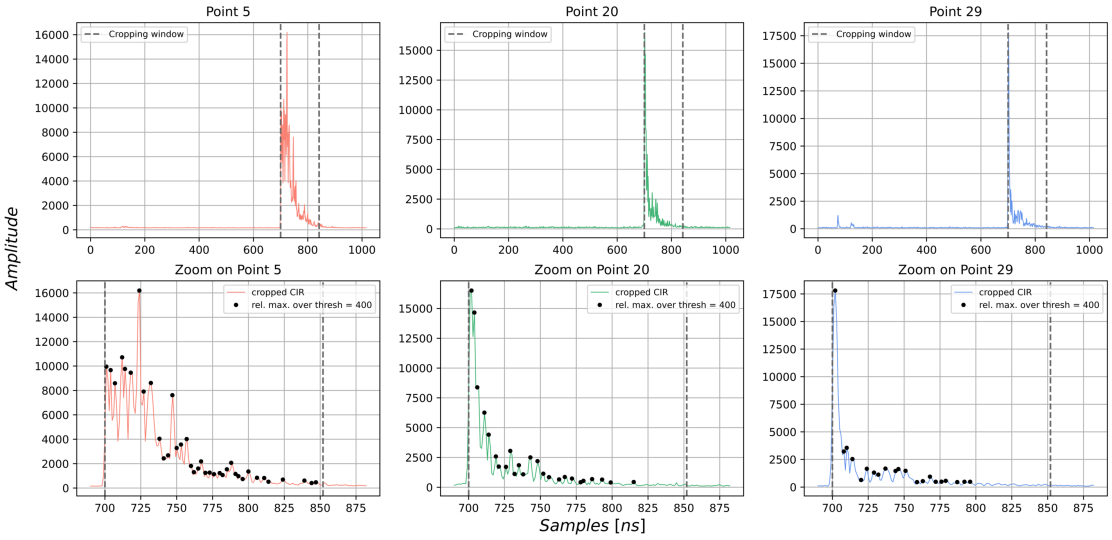


Figure 5.3: Channel impulse response samples number selection

From now on, the channel impulse response is cut according to a cropping window long 152 samples, starting from the first path index.

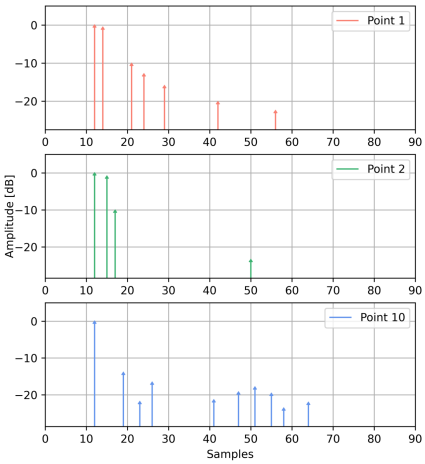
5.2.2 Point-to-point analysis

As detailed in 2.5.4 the channel impulse response retained from UWB technology carry the information about the multipath distribution of an indoor environment.

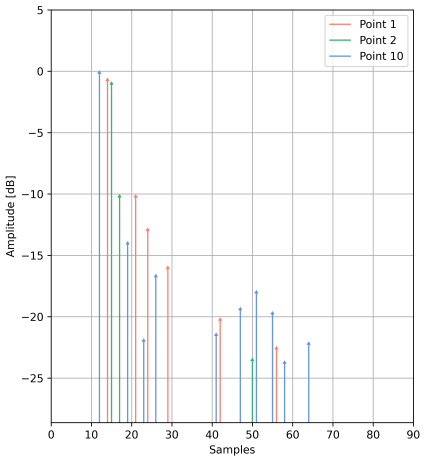
[1]First path index. It corresponds to the first time index when the CIR amplitude exceed the energy threshold. This threshold is calculated based on an estimate of the noise made during the LDE algorithm's analysis of the accumulator data.

[2]LDE is a proprietary Decawave algorithm. It reads the pulse data at the RX side and estimates the first path, looking at the output accumulated over a number of UWB pulses, trying to set the Fphw around 745. Details on the operation of the LDE algorithm are protected by IP and so are not publicly available.

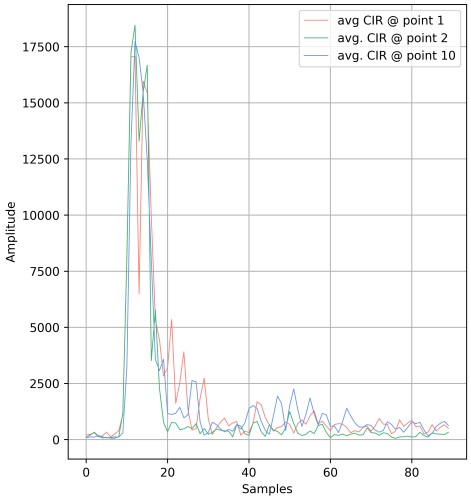
In other words, the reflections from walls and objects appear as peaks in the CIR with different amplitudes and delays. In this regard, consider Fig. 5.4 where the average CIR (together with the PDPs[3]) are compared for points 1,2,10 of the *Ideal* dataset.



(a) Normalized PDPs comparison



(b) Over-imposed normalized PDPs



(c) Average CIR

Figure 5.4: Channel impulse response and power delay profiles comparison for points 1,2,10 - Ideal dataset

As expected, point 2 shows the lowest number of multipath, being in the "best"

[3]The power delay profile (PDP) gives the signal power received through a multipath channel as a function of time delay. For small scale channel modelling, it can be computed taking the spatial average of the CIR, i.e. $|CIR(t)|^2$ over a local area [30].

LOS conditions. On the contrary, points 1 and 10 exhibit an higher number of multipath components, probably because of the reflections generated by the bench and the walls at their right side (see Fig. 4.7). Despite this analysis might be extended to all points of the dataset, here only points 1,2,10 have been considered to lighten the discussion. Fig. 5.5 reports the average channel impulse response for each point of the *Real* dataset and Fig. 5.6 the corresponding power delay profiles.

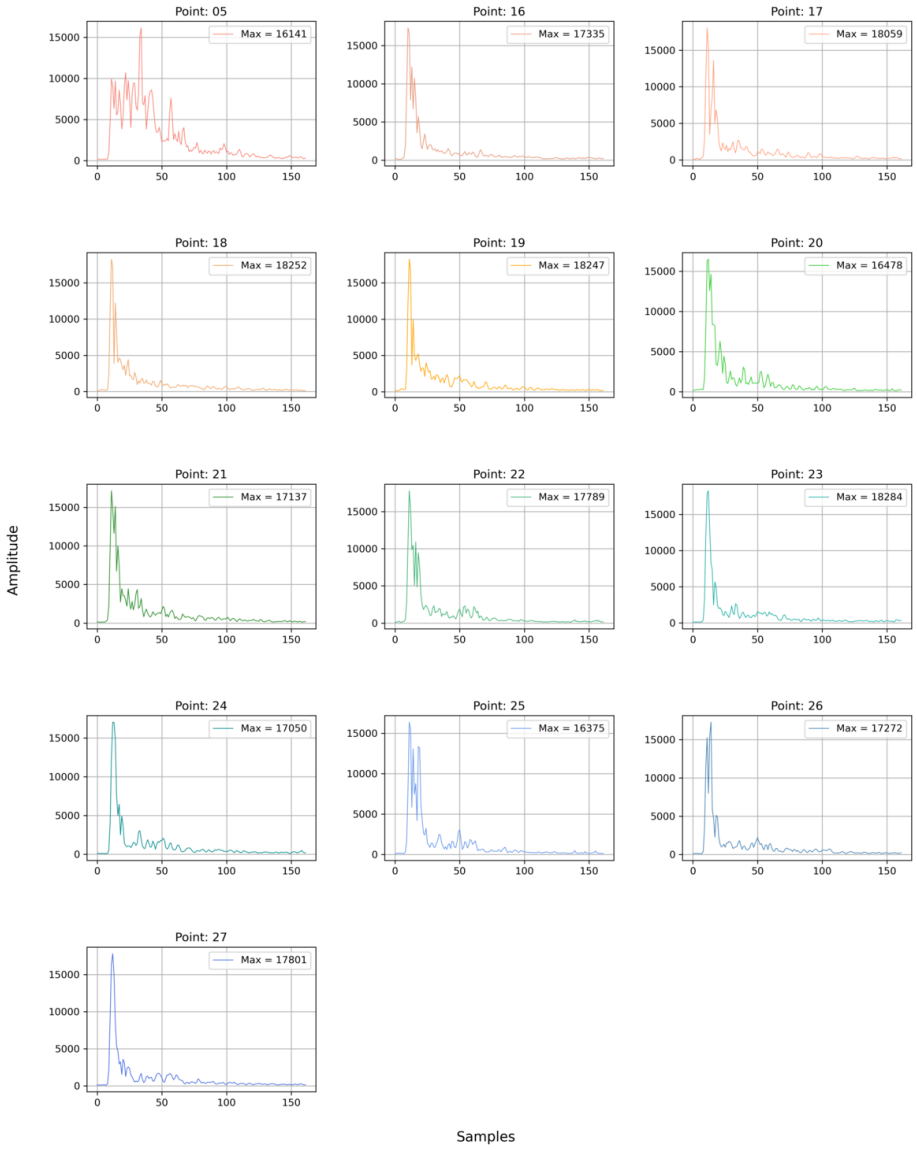


Figure 5.5: Average channel impulse response - Real dataset

It is important to remember that the *Real* dataset has been collected as detailed in 5.1 so as to include in the sample collection more or less the same number of static, dynamic and NLOS channel impulse responses for every point.

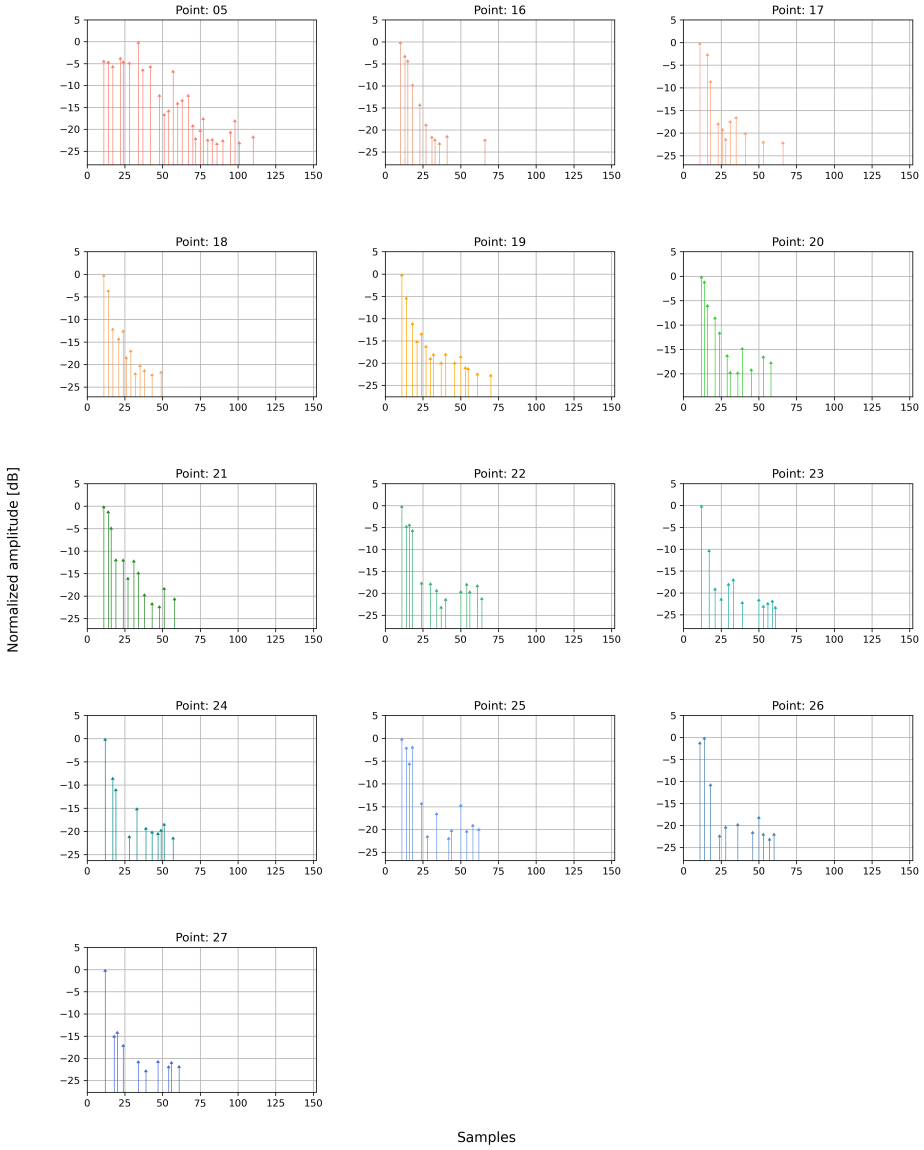


Figure 5.6: Power Delay Profiles computed on the average CIR - Real dataset

5.2.3 Environmental factors influence

This section aims to analyse the change in the channel impulse response amplitudes due to the introduction of either dynamic elements (e.g. people walking around the room), or static ones (e.g. person sit on a bench) in the test environment. To do so, a reference point is considered, taken 5m far from the tag in direct line of sight (shown in Fig. 5.7). Then, a certain number of samples (from 200 to 400) has been collected for each case, so as to get a consistent average result. The static case is compared with all others, since the goal was to analyse the influence of the environmental factors on the easiest case (i.e., the static one, where the multipath components are generated by fixed objects and walls only). Fig. 5.8 shows the obtained outcome.

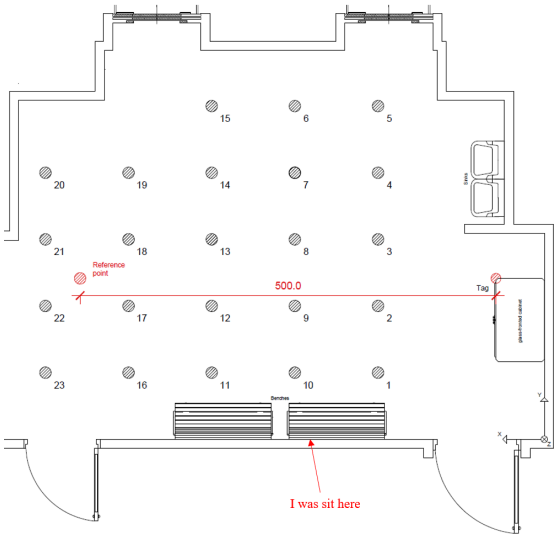
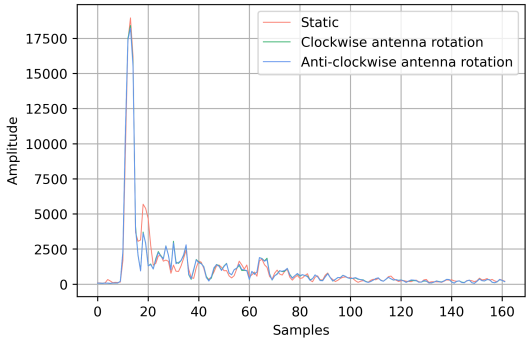


Figure 5.7: Reference point location in the test environment

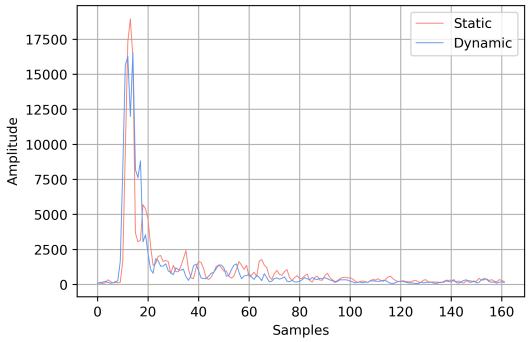
Fig. 5.8a shows the change in the average CIR by rotating the antenna either clockwise or counter-clockwise. As expected, the retrieved CIR does not change with the antenna rotation, since the UWB antenna of each EVB1000 board is an omnidirectional one. Curiously, the CIR amplitude changes a bit, even if the multipath distribution (in terms of delays) remains more or less similar.

Fig. 5.8b compares the static CIR with the dynamic one, when people walk around the room, open the doors, take the elevator and so on. As one can notice, in this case the CIR changes a lot both in terms of amplitude and multipath delays, raising the question: *"will the AI model be still able to make the CIR → location classification correctly, even in the dynamic case?"*.

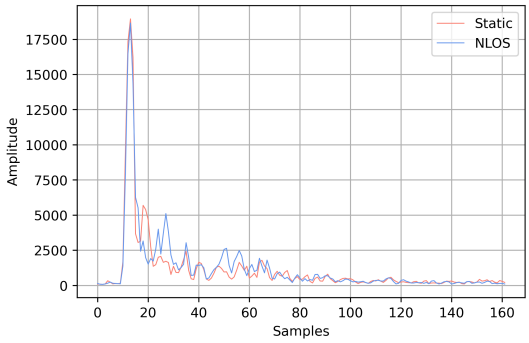
Fig. 5.8c depicts what happens to the average CIR when a person is immobile in NLOS between the two EVB1000 boards (around 2.5m from both). Differently



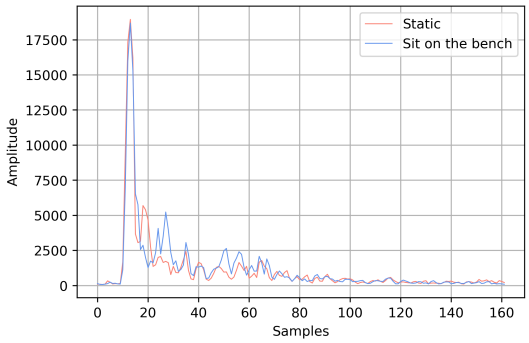
(a) Static vs rotated antenna



(b) Static vs dynamic



(c) Static vs NLOS



(d) Static vs mod. static environment

Figure 5.8: Environmental factors influence on the channel impulse response

from the dynamic case, the first path remains the same, both in terms of amplitude and delay. On the contrary, the subsequent multipath components change, especially in amplitude. The same conclusions can be drawn for the case when a person is sit on a bench in the test environment, referring to Fig. 5.7 and 5.8d.

Unexpectedly, the channel impulse response maximum amplitude (obtained for the first path) shows almost the same amplitude in all cases. The same can be noticed in Fig. 5.5 where the average CIR is computed for all points of the *Real* dataset. In principle, the amplitude of the first path should decrease with the distance, as well as in NLOS case. It leads to the conclusion the Decawave LDE proprietary algorithm performs some sort of internal normalization in the CIR reconstruction. This aspect will be taken into account later, during the data preprocessing stage.

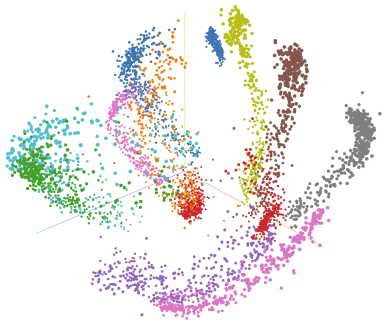
5.2.4 Principal Component Analysis

Principal Component Analysis (PCA) is one of the most popular unsupervised machine learning techniques for exploratory data analysis. It is useful to analyse dataset containing a high number of dimensions per observation (say n). It is a technique for dimensionality reduction, since it transforms the data from a high-dimensional space n into a low-dimensional space $m < n$. It does so by linearly transforming the data into a new coordinate system where most of the variation (i.e. the information in the data) is described with fewer dimensions than the initial data, increasing data interpretability.

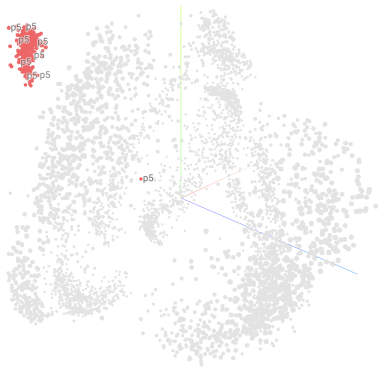
In this work the cropped channel impulse response made of 152 features (considering each time-stamp as a feature) is projected down into a 3D space. In this regard, the Tensorboard Projector by Tensorflow is used, after generating embeddings from the CIR samples. Despite many studies use PCA to transform the data into a 2D space, it has been chosen to keep one dimension more to preserve more information, since $152 \rightarrow 3$ is already a strong dimensionality reduction. Although PCA is an unsupervised technique, the transformed samples in the 3D space have been labelled with the purpose of visualizing the distribution of the clusters of each class.

The obtained results from PCA are shown in Fig. 5.10 and 5.9. In the former case, channel impulse responses from dynamic and NLOS conditions are considered only (people interacting with the environment and in between the two UWB modules). In the latter case, static CIR samples are selected (LOS conditions, reflections given by walls and objects only). In particular, Fig. 5.10a shows the projected CIRs in the 3D space for all points (i.e., for all 13 classes considered in the dataset). Similarly, Fig. 5.10b, 5.10c, 5.10d show the same highlighting the cluster distribution for classes (points) 5,24,27 respectively. The same structure is adopted in Fig. 5.9. The clusters in the static case are much more compact than the ones in the dynamic and NLOS case.

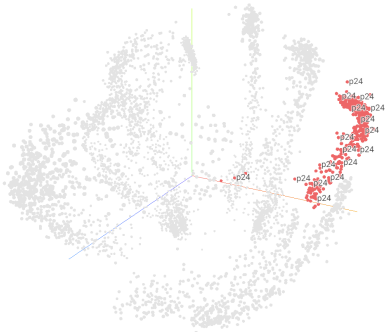
For instance, consider the class denoted by point 5 in the environment. The corresponding projected samples in the 3D space are highlighted in red and represented Fig. 5.9b (static CIRs) and 5.10b (dynamic and NLOS CIRs). The static samples are densely concentrated, with one sample only far from the others (representing most probably an outlier). On the contrary, Fig. 5.10b reveals that dynamic and NLOS samples are sparser, occupying an higher volume in the 3D space. This represents a plausible result for two reasons. First, if reflections are produced by walls and objects only, it is reasonable to get channel impulse responses similar one to each other. Second, interactions with the environment produce different reflections and attenuation, leading to changes in the channel impulse response. This is an accordance with what presented in Section 5.2.3.



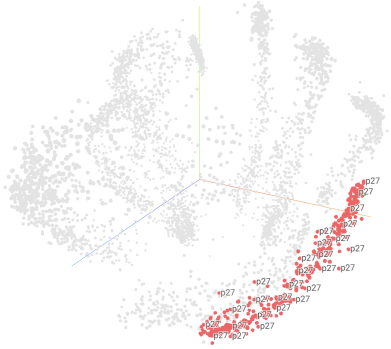
(a) PCA of static CIR samples for all classes



(b) PCA of static CIR samples - Highlight on point 5



(c) PCA of static CIR samples - Highlight on point 24



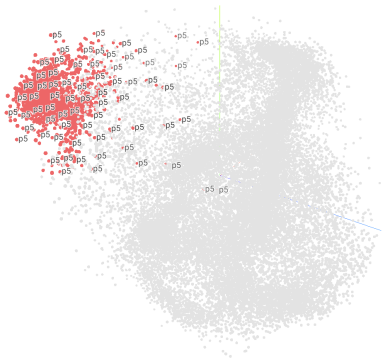
(d) PCA of static CIR samples - Highlight on point 27

Figure 5.9: Principal component analysis considering static CIR samples

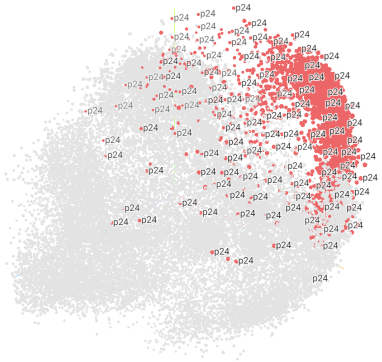
Similar reasons can be drawn for points 24 and 27, represented in Fig. 5.9c, 5.9d, 5.10c and 5.10d. Special attention has to be paid on the dynamic and NLOS samples. It it straightforward to notice that clusters strongly overlap in this case, making the channel impulse response of one point (e.g. point 27) possibly much more similar to the CIR of another (e.g. point 24), if compared with the static case.

It is crucial to observe that the fact the projected samples occupy the space in a sparse way with overlapping clusters has two consequences. On one hand, it means the dataset has been collected in a robust way, with channel impulse responses different enough to cope with the real-world scenario. On the other hand, the fact the CIRs may differ a lot even for the same point could make the learning process of the AI model much more difficult, raising the risk the DNN is not able to classify the location correctly anymore. The adopted DNN for the

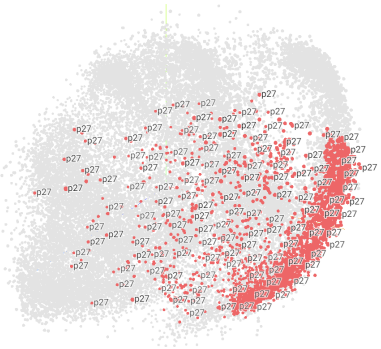
(a) PCA of dynamic and NLOS CIR samples for all classes



(b) PCA of dynamic and NLOS CIR samples - Highlight on point 5



(c) PCA of dynamic and NLOS CIR samples - Highlight on point 24



(d) PCA of dynamic and NLOS CIR samples - Highlight on point 27

Figure 5.10: Principal component analysis considering dynamic and NLOS CIR samples

classification task, as well as the learning stategy and the results in this regard are reported and detailed in the next chapter.

Chapter 6

Classification

This chapter focuses on the development of the ML pipeline to characterize the indoor environment of test through the channel impulse responses collected in the dataset detailed in the previous section. In particular, the problem can be cast as a classification task where a DNN attemps to extract features from the input pre-processed CIR to classify it correctly, according to a specific number of classes (corresponding to the locations in the environment).

As first step, data augmentation is carried out to enlarge and enrich the dataset. Then, a set of data preprocessing stages have been considered to properly train the deep neural network while improving generalization. Next, the adopted DNN model is shown and detailed, together with the learning strategy. The obtained results in terms of accuracy and confusion matrices are presented as well. Finally, the CAM algorithm is exploited to show which are the most discriminative samples of the input CIR in order to characterize one location rather than another. The complete pipeline is schematically described in Fig. 6.1.

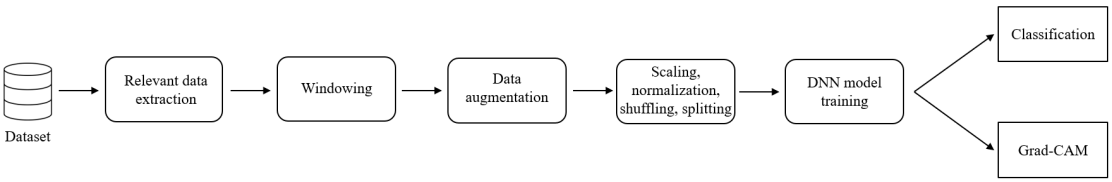


Figure 6.1: Complete machine learning pipeline of the presented work

The chosen development environment for the project is Google Colaboratory, a Jupyter notebook environment that runs entirely in the cloud. It allows to write and execute arbitrary Python code and is especially well suited to machine learning and data analysis. Despite it is free of charge to use, its free version has some limitations. Firstly, the Colab resources are not guaranteed and not

unlimited, fluctuating with the users' usage of the platform. Secondly, it is limited in RAM and GPU and it does not support the background execution. For these reasons, Google Colab Pro+ has been adopted. The Pro+ subscription makes available Nvidia P100, T4 or V100 GPUs, 2x vCPU and up to 52GB of RAM. The adoption of Google Colab Pro+ as a development environment speeds up the training process, allowing a background execution of the code up to 24 hours. It requires no setup to use and supports famous libraries like NumPy, Pandas and Keras, a simple and powerful API for Tensorflow, enabling fast experimentation and development of deep neural network models.

6.1 Preprocessing

Data preprocessing includes the steps to follow to transform and encode data so that they can be easily digested by the AI model. It is also important to improve the overall data quality, since quality predictions must be based on quality data. In addition, the easier the features to be interpreted, the better the result. In the following, the data preprocessing stages carried out in the presented work are presented and discussed in detail.

6.1.1 Data augmentation

In the presented work only basic data augmentation approaches have been exploited. In particular, linear combination has been used to produce synthetic channel impulse responses. Differently from [25], where the data augmentation is performed in the latent space, here it is carried out in the input space directly. Consider a generic location i and one of its adjacent locations j (horizontally, vertically, diagonally). Let *m,l* be the m-th and l-th random CIR for location i and j respectively. The synthetic channel impulse response $syntCIR_{i,m,l}$ can be computed as:

$$\begin{cases} syntCIR_{i,m,l} = \alpha \cdot CIR_{i,m} + \beta \cdot CIR_{j,l} \\ \alpha > 0.5 \\ \beta = 1 - \alpha \end{cases} \tag{6.1}$$

where $\alpha > 0.5$ is set to consider the synthetic generated CIR belonging to location i (i.e. it is labelled according to the i-th class), $i \in [16, 27]$. Note that the channel impulse responses of point 5 are not augmented since it is an isolated point, i.e. there is not any adjacent point in the considered dataset. It does not represent a drawback, since point 5 shows a fairly different multipath distribution from all

others, making its characterization an easy task for the DNN. An example of artificial CIR generated from two randomly picked CIRs for $i = 16$ and $j = 17$ is depicted in Fig. 6.2. Notice that the synthetic CIR is much more similar to the CIR of point i, being $\alpha = 0.8$ and $\beta = 1 - \alpha = 0.2$.

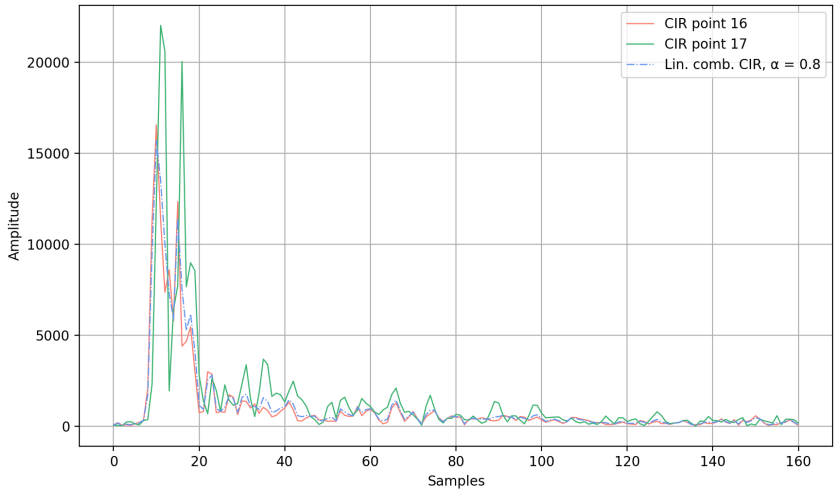


Figure 6.2: Synthetic CIR via linear combination of adjacent points - $\alpha = 0.8$

As detailed in 5.1, the data augmentation stage has been performed on the dataset obtained by merging together *Real* and *Realv2* collection of measurements. The augmented dataset is much larger than the original one, recalling the need of large dataset when training deep neural networks to improve generalization. Fig. 6.3 shows the number of samples per class for both the original (the concatenation of the *Real* and *Realv2* dataset) and the augmented dataset, considering the portion of data used for network training only. Notice some locations (e.g. point 24,25) are characterized by an higher number of samples with respect some others (e.g. point 26,27) in the augmented dataset. This occurs because some points in the test environment have a larger number of adjacent locations than others. Even for those points, the number of samples is at least 10 times larger if compared to the original dataset.

6.1.2 Scaling

The final goal after all preprocessing stages is to feed the DNN by the channel impulse responses, checking its capability to classify them correctly, according to the pre-defined locations in the environment depicted in Fig. 5.1. As already detailed in 2.5.4, the multipath distribution carries the information about the reflections. At the same time, locations characterized by similar reflections at different distances should show nonidentical amplitudes. Intuitively, the amplitude of the CIR

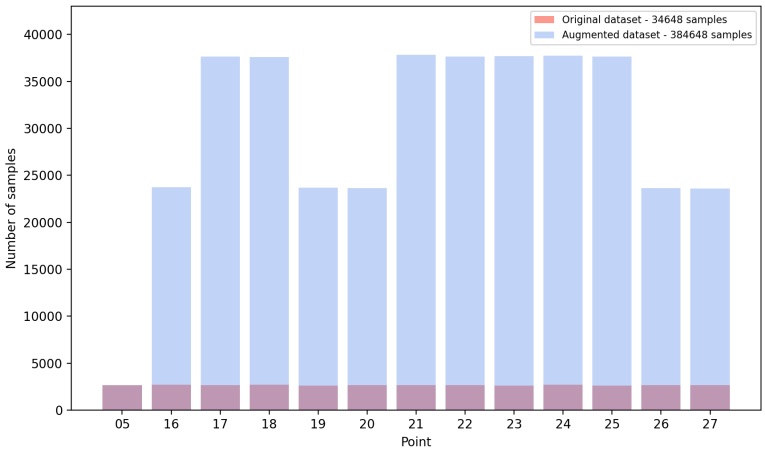


Figure 6.3: Number of samples per point used for training - Original vs Augmented dataset

(especially at the first path) should decrease with the distance, since the CIR amplitude is related to the energy detected at the receiver side. The channel impulse responses retrieved from the EVB1000 boards are somehow internally scaled by the proprietary LDE algorithm of Decawave, as already described in 5.2.3. It is sufficient to notice that the maximum amplitude of Point 27 is higher than the one of Point 24, despite it is farther from the tag. To deal with this issue, the Signal-to-Noise-Ratio is used to scale the CIR amplitude accordingly. Note that, the information of the Signal-to-Noise-Ratio comes together the reconstructed channel impulse response. As a consequence, the SNR is always avaiable to scale the CIR accordingly. Recalling the EVB1000 board implements a TWR method to compute the ToF, it is important to highlight also that the maximum packet duration of a DW1000 module is in the order of a few milliseconds [31].

The Signal-to-Noise-Ratio[1] (SNR) is related to the CIR amplitude as in Eq. 6.2, where A_i represents the amplitude of a generic CIR and P_m is the average noise power level.

[1]The Signal-to-Noise-Ratio is the ratio between the desired information (or the power of a signal) and the power of the background noise. It is often expressed in dB and for UWB communications is likely below 0 dB since the transmitted power is very low, according to the FCC mask.

$$SNR_i \cong \frac{A_i^2}{P_m} \quad \longrightarrow \quad A_i \propto \sqrt{SNR_i} \tag{6.2}$$

The defining equation for the SNR in dB is given by:

$$SNR_{dB} = 10 \cdot log_{10}\left(\frac{P_{signal}}{P_{noise}}\right) \tag{6.3}$$

Consider two locations producing similar ToF, either because at the same anchor-tag distance or due to an incorrect ToF computation in NLOS conditions. Scaling the CIR via the SNR value may help the differentiation, being the two cited points in different conditions. This motivates the idea of scaling the CIR amplitude the SNR.

Since the SNR value extracted from the *.log* file is expressed in [dBm][2], and recalling the relation in 6.2, the linearly scaled CIR amplitude $\overline{A_{i,j}}$ is computed as:

$$\overline{A_{i,j}} = \sqrt{10^{\frac{SNR_{i,dB}}{10}}} \cdot A_{i,j} \tag{6.4}$$

where the scaling coefficient is the linear SNR transformed from its dB representation.

6.1.3 Normalization

Normalization is a data preparation techniques that is frequently used in Machine Learning. It typically improves the performance and training stability of the model. This is necessary when dealing with multiple features ranging in different intervals, so as to transform all of them into a linear scale, for instance in the range [0,1]. Popular normalization techniques are Min-Max Scaling and Standardization. In the presented work, each CIR sample represents a feature and all channel impulse responses more or less range in the same amplitude intervals. For this reason, the dataset is normalized so as to preserve the information in the CIR amplitude. Mathematically, we have the following:

$$k_{scale} = \max_k \max_i \overline{A_{i,k}} \tag{6.5}$$

[2]dBm is defined as the dB using a reference P_{noise} of 1 mW

where, $i \in [1, n_{CIRs\ in\ D_k}]$, $j \in [1, K]$ and $\overline{A_{i,k}}$ is the scaled amplitude of the i-th CIR of class k, computed as in Eq. 6.4. D_k represents the set of training samples for class k and K is the total number of classes (equal to 13). Then, k_{scale} is used as normalization coefficient for all remaining samples of both the validation and test dataset of the augmented dataset, and for the *Realv3* dataset as well.

6.1.4 Shuffling

The collected and extracted dataset is originally sorted by their class, since a *.log* file is produced for every location in the environment and the relevant data are automatically extracted via a Python script scrolling one *.log* file at a time. Here, shuffling is needed to make sure the training, validation and test sets are representative of the overall distribution of the data. If the dataset is split before shuffling, the model is trained for some classes only and tested on others, completely failing the process. By rearranging the data, it is ensured that each data point influences the model in an "independent" manner and is not influenced by the values that came before it. Shuffling the data also helps in reducing the variance and overfitting issues. It is implemented by randomly permutating the data samples in the dataset.

6.1.5 Splitting

The splitting procedure is used to train and then estimate the performance of any machine learning algorithm. In order to build a solid model there is a specific protocol of splitting the data into three sets: one for training, one for validation and one for final evaluation. In particular,

- *Training* dataset is the set of examples used for learning, that is to fit the weights of the model;
- *Validation* dataset, composed of data samples used to provide an evaluation of a model fit on the training dataset, while tuning model hyperparameters;
- *Test* dataset, used only to evaluate the performance of the final model on new, unseen data.

Note that the training dataset is only one used for training the model. During each epoch, the model is trained on samples taken from the training set. Instead, the model is validated on each sample of the validation set. Although there is no optimal split percentage, most practitioners in Deep Learning suggest to take at least the 60% of the dataset for training. In the presented work, 70% of the dataset is used for training, 20% for validation and 10% for the final evaluation.

The split refers to the augmented dataset, detailed in 6.1.1. Furthermore, the model is tested on the *Realv3* dataset, representing totally unseen data from the network perspective and useful to test the network generalization performance in a real robust way.

6.2 Adopted model

The choice of a deep learning model is not an easy task almost in every application. It has to be a trade-off among many aspects, including network complexity, generalization performance, training time, scalability of the model and many more. For example, a very complex model may work efficiently when run in cloud but completely unfeasible if one wants to bring the ML model capabilities locally to an edge device (e.g. an embedded platform). In section 3.3.1 the most common DNN models for the TSC task are reported and detailed. Actually, many others may be considered, including Echo State Networks and Long-Short Term Memory networks. In this work, a Fully Convolutional Neural Network has been adopted, taking inspiration from the one proposed in [14]. In particular, it is inspired from

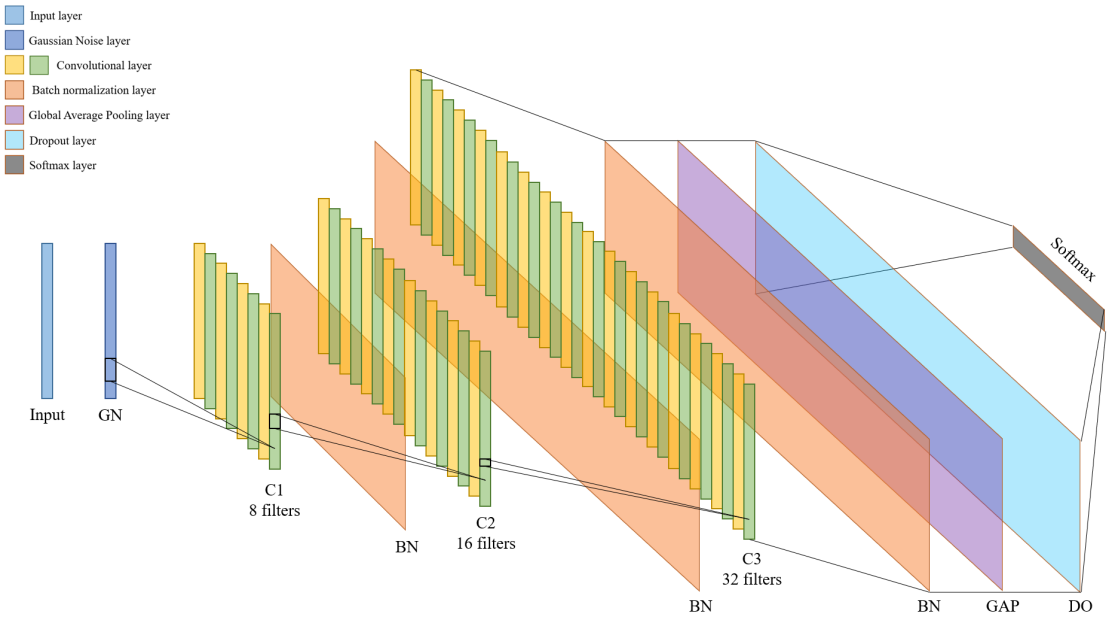


Figure 6.4: Adopted FCN model for the presented TSC task

the one depicted in Fig. 3.8 with the addition of three different layer types: Gaussian noise layer, Batch Normalization layer and the Dropout layer, detailed in the following subsections. The input layer has a (152,1) shape in accordance with the windowed channel impulse response detailed in 5.2.1. The output layer is a softmax

Layer (type)	**Output shape**	**Param #**
Input	(152,1)	0
Conv 1D	(152,8)	72
Batch normalization	(152,8)	32
Activation (ReLU)	(152,8)	0
Conv 1D	(152,16)	656
Batch normalization	(152,16)	64
Activation (ReLU)	(152,16)	0
Conv 1D	(152,32)	1568
Batch normalization	(152,32)	128
Activation (ReLU)	(152,32)	0
Global Average Pooling	(,32)	0
Dropout	(,32)	0
Dense (softmax)	(,13)	429
Total # of parameters: 2949		
Trainable parameters: 2837		
Non-trainable parameters: 112		

Table 6.1: Network layers description and parameters

layer with $K = 13$ classes, since 13 is the number of locations considered in the acquired dataset. The intermediate layers are mainly a sequence of convolutional and batch normalization layers and the length of the CIR remains unchanged over the convolutions because same padding is applied. The final adopted DNN model for the presented work is shown in Fig. 6.4. The hyperparameters configuration is instead reported in Table 6.3. Finally, the description of each layer in terms of output shape and number of parameters is detailed in Table 6.1.

6.2.1 Gaussian noise

The addition of a Gaussian noise on top of the input can be seen as a form of data augmentation, as detailed in 3.4.1. This is useful to mitigate overfitting and is a natural choice as corruption process for real valued inputs. The Keras API employed to build the DNN in the Tensorflow framework allows to apply an additive zero-centered Gaussian noise via the inbuilt GaussianNoise class [32]. Since it acts as a regularization layer, it is only active at training time. The only required argument is the standard deviation (stddev) of the noise distribution. It

represents the amount of added noise which has to be chosen so as to sufficiently alter the input data avoiding to make it exceedingly different from the original one. Different tests have been carried out and, at the end, it has been found the best value equal to 0.0035.

6.2.2 Batch normalization

Batch normalization is employed to train deep neural networks in a more robust way, helping to reduce the problem of internal covariance shift, as discussed in 3.3.1. It does so by standardizing the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning phase and reducing the required number of training epochs. The Keras Batch normalization layer [33] applies a transformation that takes the output of the previous layer and maps it into a mean output close to 0 and a standard deviation close to 1.

6.2.3 Dropout

Deep neural networks with large number of parameters are very powerful but they strongly suffer from overfitting. In this regard, dropout represents an efficient regularization technique to prevent overfitting. The key idea is to randomly drop units (along with their connections) from the neural network during training. In particular, the term *dropout* refers to dropping out the nodes (input and hidden layer) in a neural network, thus creating a new network architecture out of the parent network according to a chosen probability p [34]. The effect of dropout is shown in Fig. 6.5. It can be easily implemented via the Dropout class in the Keras API, recalling the dropout layer only applies during training such that no values are dropped during inference [35].

6.3 Learning strategy

This section analyses the learning strategy adopted to properly train the adopted model for the presented work, together with the hyperparameters selection.

6.3.1 Optimizers

Every deep learning model consists of an input layer, a certain number of hidden layers and an output layer. The way the network tries to generalize the data is by using an algorithm that maps the examples of inputs (training dataset) to the outputs. This is done by adopting an optimization algorithm which attempts to find the best model's parameters that minimize the loss when mapping inputs to

(a) Standard Fully Connected Network

(b) New architecture after dropout

Figure 6.5: Effect of dropout regularization on a FCN

outputs. It is possible to generalize the concept saying that an *optimizer* is an algorithm used to try to minimize an error function or to maximize the efficiency of the considered problem. Since the model's parameters cannot be computed analytically in a deep learning model, an optimization algorithm is needed. Many optimization algorithms have been proposed over the years; the most common and effective ones are reported in the following.

Gradient Descent

The Gradient Descent (GD) is a first-order optimization algorithm which attempts to find the model's weights by minimizing a loss function. It reduces the loss function in an iterative way by moving in the opposite direction to the steepest ascent (i.e. the opposite direction to the gradient) and stops in a local minima. It is straightforward to understand and easy to implement but it is not feasible for large dataset since it uses the data of the entire training set to calculate the gradient, requiring large amount of memory and slowing down the training process. The recursive model weights update can be expressed as:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\partial(loss)}{\partial(\theta_t)} \tag{6.6}$$

where η is the learning rate parameter of the GD algorithm and controls how much the weights can change on each update. Fig. 6.6 shows the Gradient Descent algorithm in a graphical way, highlighting the effect of the learning rate choice.

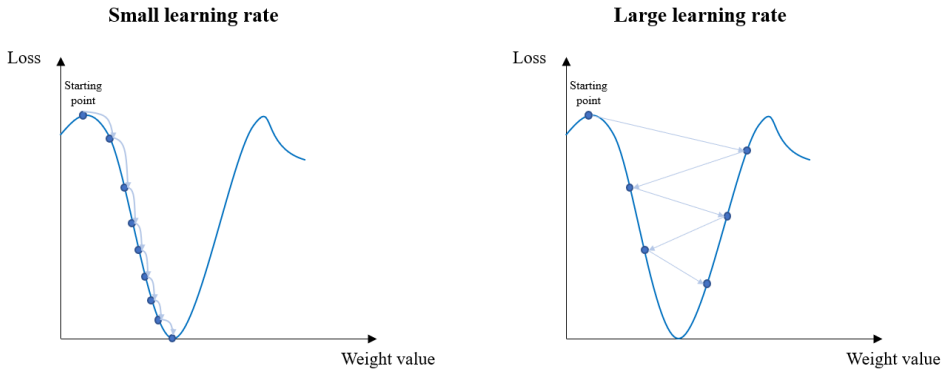


Figure 6.6: Gradient Descent algorithm with small learning rate (left) and large learning rate (right)

Stochastic Gradient Descent

Gradient Descent may be slow when run on very large dataset, since one iteration of GD requires the computation of the gradient on the whole dataset. To overcome this issue, Stochastic Gradient Descent (SGD) has been proposed. This variation takes a random selected batches of data to compute the gradient and update the model's weights (from which the name "stochastic"). Because the algorithm does not use the whole dataset but only a batches of it at each iteration, the model weights update may be noisy. To tackle this problem, typically SGD uses a higher number of iterations to reach the local minima of the loss function if compared to GD. Regardless this aspect, if the dataset is large, SGD should be preferred over the GD optimization.

AdaGrad

A limitation of both GD and SGD algorithms is that they use the same learning rate for each input variable during the optimization phase. This may represent a problem when the objective function has a large dimension and different curvatures, requiring different step sizes to properly reach the local minima. Adaptive Gradients, or AdaGrad for short, is an extension of the Gradient Descent algorithm allowing for a different learning rate in each dimension tuned automatically by the algorithm on the basis of the computed gradient at each step of the optimization. The parameters with the largest partial derivative of the loss have a correspondingly rapid decrease in their learning rate, while parameters with small partial derivatives have a relatively small decrease in their learning rate [24]. The update of the model's parameters can be written as:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \times g_t \tag{6.7}$$

where G_t is a matrix containing the sum of the squares of the gradients with respect to all parameters θ up to time step t along its diagonal, ϵ is a smoothing term that avoids division by zero, g_t is used to denote the gradient at time step t where $g_{t,i} = \frac{\partial(loss)}{\partial(\theta_{t,i})}$ and $\times$ denotes the cross-product operator.

One of the main Adagrad's benefit is that it eliminates the need to tune the learning rate. At the same time, since the accumulated sum of gradients keeps growing during training, its main weakness is that the learning rate may become infinitesimally small, causing the dead neuron problem (i.e. the algorithm is no longer acquiring knowledge).

RMS-Prop

The Root Mean Square Propagation (RMS-Prop) is a special version of Adagrad in which the learning rate is not taken as the cumulative sum of squared gradients, while the exponential average of gradients is considered instead. This attempts to solve the Adagrad's vanishing learning rate problem. In mathematical terms the update rule looks like the following:

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1 - \beta) g_t^2 \tag{6.8}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \times g_t \tag{6.9}$$

where $E[g]$ is the moving average of squared gradients and β is the moving average parameter. Typical default values are 0.9 and $1e-3$ for β and η respectively.

Adam

Adaptive Moment Estimation (Adam) is another algorithm designed to take the benefits of both AdaGrad and RMS-Prop optimization methods. The Adam algorithm computes individual adaptive learning rates for different parameters from estimates of first and second order moments of the gradients [36]. In addition to the computation of the exponential average of gradients, the notion of momentum[3] is taken into account. Firtly, the decaying averages of past and past squared gradients is computed, as follows:

[3]Formally, the n-th moment of a random variable is defined as the expected value of that variable to the power of n, i.e. $m_n = E[X^n]$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{6.10}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{6.11}$$

where, β_1 and β_2 are the exponential decay rates for the first and second moment estimates respectively. Since m_t (the first moment - the mean) and v_t (the second moment - the uncentered variance) are initialized as vectors of 0's, they are biased towards zero. To tackle this issue, the bias-corrected first and second moments are estimated as:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{6.12}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{6.13}$$

then, the first and second order bias-corrected moments $\hat{m}_t$ and $\hat{v}_t$ are used to update the model weights as in the AdaGrad and RMS-Prop optimization algorithms:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{6.14}$$

The Adam optimizer is easy to implement, computationally efficient and has little memory requirements. In addition, empirical results demonstrate that Adam works well in practice even using large models and dataset, proving it can efficiently solve practical deep learning problems. For all cited reasons, the Adam optimizer has been the choice for the presented work in the training phase of the adopted DNN model.

6.3.2 Loss functions

The loss function is a method to quantify how well the AI model is performing. In other words, it is a way to measure how good the model is predicting the output with respect to the expected outcome. The loss function is the function minimized by the optimizer and a low value means the model is providing good results. During the training phase, it is evaluated on both the training and validation dataset.

Loss functions can be grouped into two main categories referring to the types of problem one may encounter in the real-world: classification and regression. Since the presented work is about a TSC task, the regression problem is taken away from the discussion. In classification problems, the task is to identify which category (class) an observation (input) belongs to. In the following, the most common loss functions for the classification task are detailed.

Hinge Loss

Primarily developed for the Support Vector Machine (SVM) classifier, the Hinge Loss function is computed as:

$$L = max(0, 1 - y \cdot f(x)) \tag{6.15}$$

where y is the actual outcome and $f(x)$ is the output of the classifier.

Binary Cross-Entropy

Also called logarithmic loss or log loss, this is the most common loss function used in classification problems. Considering a classification model whose predicted output is a probability value between 0 and 1, it measures the performance of the model on how far it is from the actual expected value. For a binary classification problem:

$$L = -\frac{1}{m}\sum_{i=1}^{m}(y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)) \tag{6.16}$$

where, y_i is the actual class and $\hat{y}_i$ is the predicted class for the i-th training sample and m is the number of training samples. Fig. 6.7 depicts the log loss function for a classification task where the "true" label is identified by value 1.

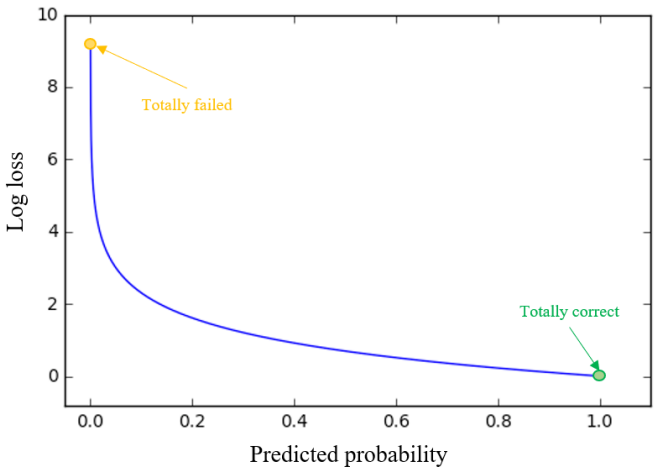


Figure 6.7: Log loss function

Categorical Cross-Entropy

It is an extension of the Binary Cross-Entropy for multi-class classification, when a softmax layer is employed as output layer of the DNN. It is the loss function adopted in the presented work to evaluate the model performance and for a TSC task it can be computed as:

$$L(X) = -\sum_{k=1}^{K} Y_k log\hat{Y}_k \tag{6.17}$$

with L denoting the loss when predicting the class for the input time series X, K the total number of classes, Y_k and $\hat{Y}_k$ representing the actual and predicted probability (output of the softmax layer) for the k-th class.

6.3.3 Hyperparameters

Tuning the hyperparameters for deep neural networks is not an easy task as the best combination may vary a lot depending on the specific domain of application and because of the high number of parameters to configure. Following the taxonomy proposed in [37], the key hyperparameters related to the design of a Convolutional Neural Network are reported in Table 6.2. Some of those have been considered in the hyperparameters tuning for the adopted model. A large number of tests have been carried out with a grid search tuning in order to find the best trade-off between model complexity and performance. In particular, for the adopted model shown in Section 6.2, the range of the hyperparameters in the grid search approach and the adopted ones are reported in Table 6.3.

6.4 Results

After having investigated the dataset collection, data distribution, data preprocessing, network decision and training, this section aims to show the obtained results for the presented work. In this regard, various metrics can be adopted to evaluate the performance of an AI model. Among them, there is the quite popular *accuracy*. It generally describes how the model performs across all classes and it is useful when all classes are of equal importance. Since every location in the environment has the same relevance, it has been adopted as metric to evaluate the network performances in the current research. Formally, it is computed as:

$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{6.18}$$

Hyperparameters	**Symbol**
Number of convolutional layers	N_c
Number of kernels of each convolutional layers	$kN_{i,i\in N_c}$
Kernel size in each convolutional layer	$kS_{i,i\in N_c}$
Activation function in each convolutional layer	$aF_{i,i\in N_c}$
Pooling size (if any) after each convolutional layer	$pS_{i,i\in N_c}$
Number of dense layers	N_d
Connectivity pattern of each dense layer	$P_{i,i\in N_d}$
Number of neurons of each dense layer	$nN_{i,i\in N_d}$
Weight regularization in each dense layer	$R_{i,i\in N_d}$
Dropout	$R_{dropout}$
Batch size	S_{batch}
Learning rule	L_{rule}
Learning rate	L_{rate}

Table 6.2: Key hyperparameters in the CNN design

Despite the accuracy, which is useful to evaluate the fraction of predictions the model got right, other evaluation methods can be considered. Among them, the confusion matrices are surely a useful way to get insight about how the network is doing the classification. In particular, a confusion matrix is a KxK matrix used for evaluating the performance of a classification model, where K is the number of target classes. The matrix compares the instances in an actual class (along the rows) with those predicted by the model (along the columns), or viceversa. This gives a graphical view of how well the classification model is performing and "where" the model is failing most. In the following, the confusion matrices and the accuracy are evaluated on both the test dataset split from the augmented one (equal to the 10% of the total dataset cardinality) and on the entire *Realv3* dataset, representing totally unseen data from the network perspective.

Finally, the results obtained from the implementation of the Class Activation Map algorithm are shown and detailed, highlighting which samples of the channel impulse responses are used most for characterizing the different locations in the environment.

6.4.1 Confusion matrices

In this section the obtained results in terms of confusion matrices and accuracy are presented and discussed in detail. In this regard, it is important to recall that the network has been tested on two different set of data. A first test has been carried

Hyperparameter range	**Adopted value**
$N_c \in (1, 2, 3)$	3
$kN_{i,i\in N_c} \in (2, 4, 8, 16, 32)$	$(8, 16, 32)$
$kS_{i,i\in N_c} \in (2, 3, 4, 5, 6, 8, 10)$	$(8, 5, 3)$
$aF_{i,i\in N_c}$	ReLU
N_d	1
$nN_{i,i\in N_d}$	$K = 13$ classes
$R_{i,i\in N_d} - L2 \in (5e-2, 5e-3, 5e-4)$	$5e-3$
$R_{dropout} \in (0.1, 0.2, 0.3, 0.4, 0.5)$	0.3
$S_{batch} \in (2, 4, 8, 16, 32, 64, 128)$	4
$L_{rate} \in (1e-3, 4e-3, 5e-3, 1e-4, 5e-4, 1e-5, 5e-5)$	$4e-3$

Table 6.3: Adopted hyperparameters in the FCN design

out on the test portion of the augmented dataset. As in any deep learning project, the dataset has been split in training, validation and test portions where the test portion has been chosen equal to the 10% of the overall dataset cardinality, as detailed in Section 6.1.5. A second test has been performed on the *Realv3* dataset. As discussed in Section 5.1, this dataset is composed of around 13000 samples including both static and dynamic environmental factors. More importantly, it really represents totally new and unseen data from the network perspective. In fact, the extracted test set from the augmented dataset (which is just the 10% of the overall dataset cardinality after shuffling) may contain some similar channel impulse responses which are already in the training and validation sets, leading to false better results. This is even more true in the case of the presented work, where a deep learning location-based indoor environment characterization via UWB channel impulse response is analysed for the first time. Figures 6.8 and 6.9 show the confusion matrices for both test datasets and Table 6.4 reports the obtained accuracies. Some important considerations can be drawn on the basis of the obtained results. Naturally, one hopes to get confusion matrices which are strong on the main diagonal, meaning the network fails the classification just in a few cases. This is the case of Fig. 6.8, when the network is evaluated on the test set extracted from the augmented dataset. In this case, results are promising. The network classifies correctly the locations in the environment on the basis of the input CIR at least 8 times of 10 (check the accuracy reported in Table 6.4). When the model fails the classification, it does so by misleading close points. It is is a reasonable result, since close locations in the environment are expected to have similar channel impulse responses. On the contrary, the network shows difficulties to generalize when tested on the *Realv3* dataset. In fact, the obtained confusion

Adopted model	**Test dataset**	**Cardinality**	**Accuracy**
FCN	10% of the aug. dataset	54950	82.73%
FCN	Realv3	11554	52.56%

Table 6.4: Adopted FCN model obtained accuracies

matrix in Fig. 6.9 is much more sparse if compared to the one in Fig. 6.8. At the same time, the accuracy drops down to 52.56% as reported in Table 6.4.

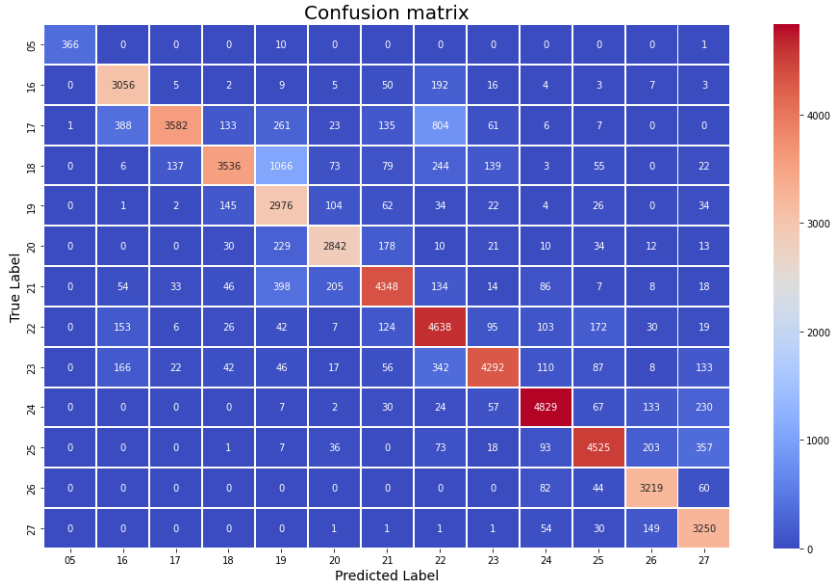


Figure 6.8: Confusion matrix - Test set: extracted 10% test set for the augmented dataset

This result raises several considerations, including the overfitting of the augmented dataset by the adopted model and the fact the collected dataset may be not completely representative of the indoor test scenario. These considerations, together with possible solutions, are discussed in detail in the next and final chapter. Notice that in both cases point 05 is almost always correctly classified, being its multipath distribution much more different from all others, as it can be clearly noticed in Fig. 5.5.

It is also relevant to heavily investigate the meaning of the obtained confusion matrices so as to understand if the network misclassifies a point with one of its neighbors most times or not. Table 6.5 provides some results in this regard. As one can clearly notice, the network misclassifies a point with one of its neighbors around 8 times of 10 and 6 times of 10 for the augmented and *Realv3* dataset respectively. This is a meaningful result, highlighting the fact that when the network fails, it

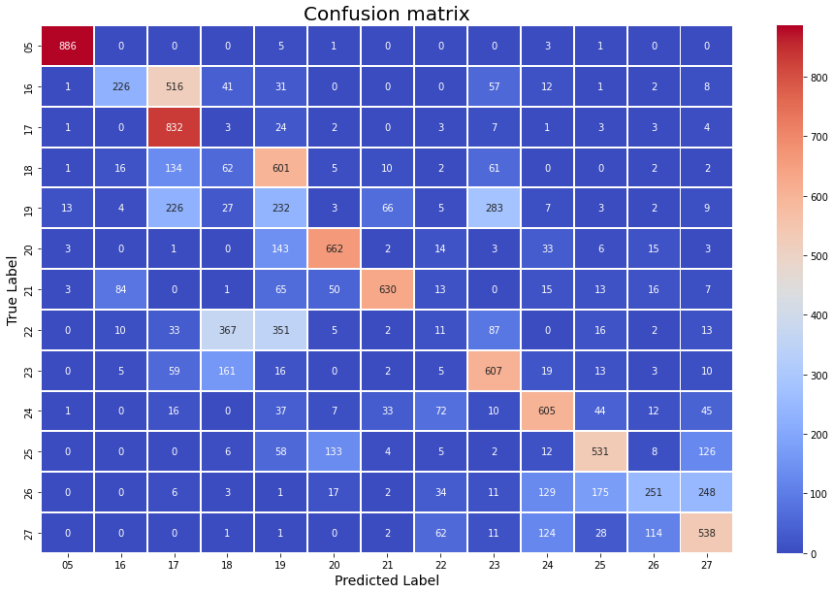


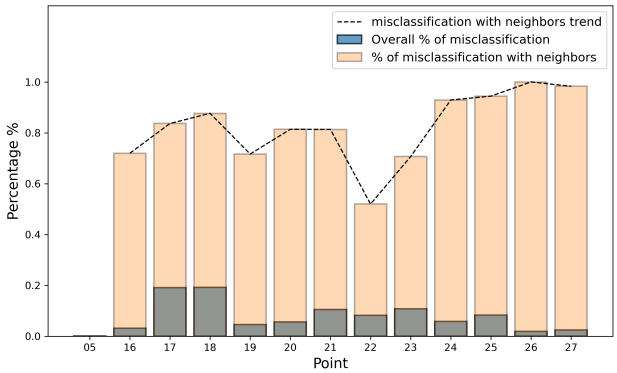
Figure 6.9: Confusion matrix - Test set: *Realv3* dataset

Test dataset	**Dim**	**# of misclassifications**	**% of misclassifications with neighbors**
10% of the aug. dataset	54950	9490	81.16%
Realv3	11554	5481	63.16%

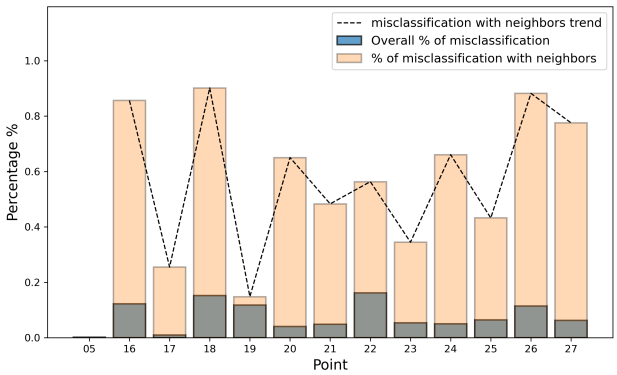
Table 6.5: Misclassified locations with neighbors results

mostly does so without confusing the CIR of a location with the one of a completely different location. As an example, referring to Fig. 6.9, point 16 is almost always misclassified with point 17. Similarly for point 18 with locations 17 and 19. This is not the case for the location numbered as 22 which is confused most of the times with location 19. An in-depth analysis regarding the misclassified points is shown in Fig. 6.10. It reports both the overall percentage of misclassification (i.e. how many times a specific location has been misclassified in percentage with respect to the total number of misclassifications) and the percentage of misclassifications with neighbors, describing how often the network confuses the CIR of a location with the CIR of a neighbor one. Firstly, notice how much often some locations are misclassified than others (as an example, consider points 17-18 with respect to 26-27 in the augmented dataset). As a second insight, it is also straightforward to note that there is not a clear relation between the misclassifications on the augmented dataset and the ones on the *Realv3* dataset. In fact, points which

are mostly classified uncorrectly by the network in the augmented dataset are not the same in the *Realv3* dataset (refer to points 16-17 for example). Finally, the network shows a more constant behaviour when misclassifying points with neighbor locations on the augmented dataset. In fact, the dashed line represented in Fig. 6.10a is much more smoothed if compared to the one in Fig. 6.10b which shows a jagged behaviour. The highlighted behaviour is in accordance with the percentage of misclassifications with neighbors reported in Table 6.5. These results point out a clear difference in terms of network performances on the two datasets: the model fails the classification less than 20% of cases on the augmented dataset and when it fails, it does so in most cases (more than 80%) with a neighbor location. On the other hand, the network misclassifies the *Realv3* dataset around 47% of times and in such cases only around 6 times over 10 is done by misleading the location with a neighbor one.



(a) Augmented dataset



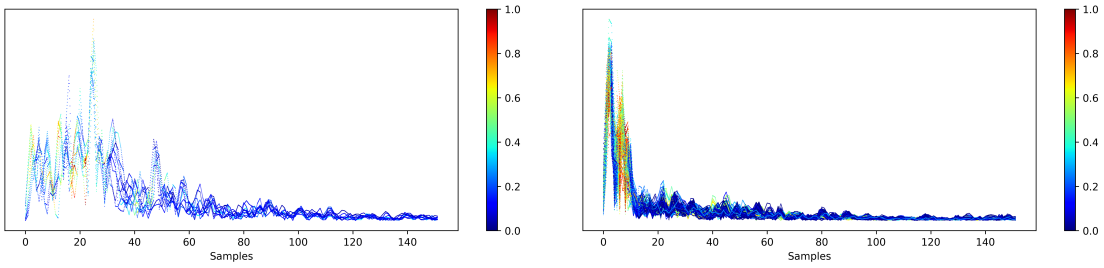
(b) *Realv3* dataset

Figure 6.10: Misclassification analysis on the two test dataset

6.4.2 Grad-CAM

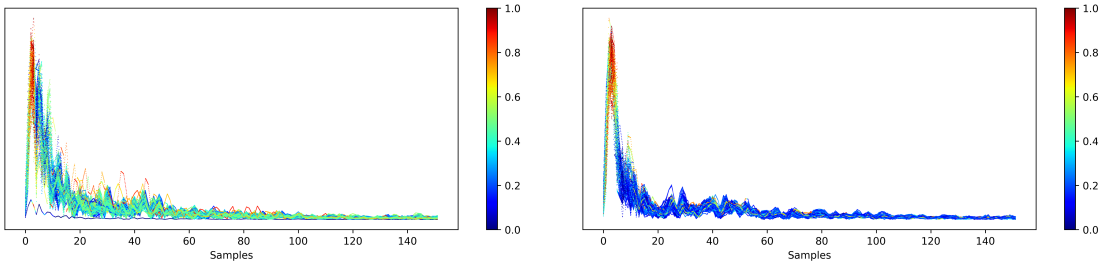
Which are the most crucial samples of the channel impulse response in characterizing a specific location in the test indoor environment? This section aims to provide an answer to the posed question, opening the black-box model of the deep learning model via the implementation of the Class Activation Map algorithm described in 3.5.1. In particular, the Grad-CAM algorithm has been considered so as to show a coloured representation of the input CIR according to a chosen colormap.

To produce the results of the Grad-CAM algorithm, a subset of both the test set extracted from the augmented dataset and the *Realv3* dataset has been considered. In particular, the algorithm has the aim to show the subsequences of the channel impulse response that contributed most to a certain classification. For this reason, the output of the CAM is produced only in case the predicted label matches the actual one. Recalling Section 3.5.1, employing the CAM is only possible for networks with a GAP layer preceding the softmax output classifier. Referring to Fig. 6.4, it is clear the adopted model shows a dropout layer in the middle of those. This aspect does not represent an issue, since the dropout layer is used during the training phase only.



(a) Grad-CAM - Point 05

(b) Grad-CAM - Point 17

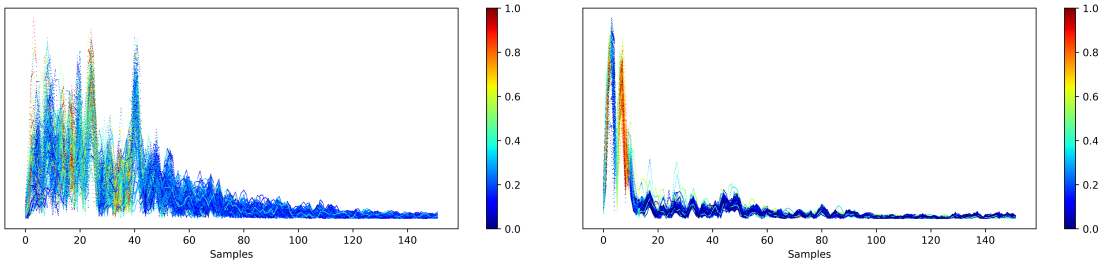


(c) Grad-CAM - Point 20

(d) Grad-CAM - Point 27

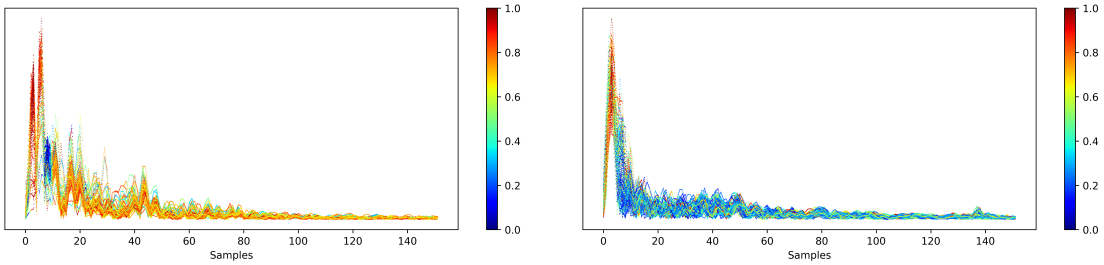
Figure 6.11: Grad-Class Activation Map algorithm results - Test set: 10% extracted from the augmented dataset

Fig 6.11 and 6.12 depict the obtained outcome of the algorithm on a subset of



(a) Grad-CAM - Point 05

(b) Grad-CAM - Point 17



(c) Grad-CAM - Point 20

(d) Grad-CAM - Point 27

Figure 6.12: Grad-Class Activation Map algorithm results - Test set: *Realv3* dataset

locations (classes) of the augmented and *Realv3* dataset, respectively. Red regions represent the most discriminative CIR samples used for the classification, while blue regions refer to the less discriminative ones (according to the colormap). It is interesting to highlight that just a few samples are mostly used to perform the classification and they are close to the first path (refer to points 17,20,27). It means the information retrieved by the next multipath components is not widely exploited by the network to distinguish one location from another. At the same time, as expected the model uses more or less the same regions for both the augmented and the *Realv3* dataset for the classification. If points 17 and 27 show a similar trend, this is not the case for points 5 and 20. In fact, an higher number of peaks (multipaths) is used by the network to classify location 5. This is probably related to the presence of a larger number of MPCs with greater amplitude for the specific channel impulse response. On the contrary, particular attention has to be posed on the results obtained for point 20, shown in Fig. 6.11c and 6.12c. Differently from all other cases, the classification for such points exploit the entire channel impulse response. In addition, despite the fact the most discriminative regions are close to the first path for both the augmented and the *Realv3* dataset, it is clear the network strongly relies on most of the multipath components to perform the classification. This aspect is even more evident when the Grad-CAM algorithm

is run on the CIRs belonging to the *Realv3* dataset where, apart from the third multipath, all others are coloured in orange.

The Class Activation Map algorithm has clearly posed the basis to get insights about the decisions taken by the DNN model during the classification. It is clear that in most cases the network relies on the first multipath components and "neglects" all others. This aspect may represent an issue, since most of the MPCs should characterize every specific location in an indoor environment. Nevertheless, this is an open research problem which may be addressed in future researches. The next and final chapter proposes some suggestions and draws the conclusion of the presented work.

Chapter 7

Conclusions

The main objective of the presented work has been inspired by the question *"Does an indoor environment have its own electromagnetic firm? In other words, could a specific location within an indoor environment be somehow identified by its own propagation characteristics with respect to a reference point?"*. To tackle the problem of characterizing every location in an indoor environment, the channel impulse responses (CIRs) retrieved from an ultrawide-band transponder has been used, exploiting the potentiality of the current deep learning algorithms. The decision of adopting UWB modules for the presented research relies on two main aspects. First, the ultrawide-band technology is already widely adopted in indoor positioning applications (which is the working scenario selected for this work). Second, the channel impulse responses carry the information about the multipath components (MPCs) of an indoor enviroment (i.e., the reflections generated by walls, objects and people). The large bandwidth of UWB schemes allows resolving a large number of MPCs and generates a better characterization of the multipath environment with respect to narrowband transmission schemes.

In order to train the network in a robust way, particular attention has been posed in the dataset collection. In this regard, the chosen indoor environment included most of the typical elements of an indoor scenario including both static elements (walls, doors, and many others) and dynamic ones (the presence of people interacting with it). A balanced dataset with over $49'000$ samples have been collected. Then, a data augmentation technique in the input space has been performed to augment it by around 7 times, trying to tackle the problem of the dataset cardinality when training deep learning models. A fully connected convolutional neural network (FCN) has been adopted to prove the validity of the presented idea. The obtained results show that a high classification accuracy is obtained when the model is fed by the test set extracted from the augmented dataset. In fact, an accuracy of 82% is obtained. In addition, when the network misclassifies a location with another, it does so with neighbor locations in at least the 80% of

cases, which represents both a reasonable and promising result. To have a more restrictive benchmark, another test has been carried out testing the network on a new set of measurements collected much time after the previous one and totally unseen during the network training. Results show the accuracy drops to around 53% and also in the misclassification cases, the model fails the classification with neighbors in lower instances, around the 63% of times. The main cause of error may be attributed to the consistent presence of dynamic factors introduced in the environment by the people interacting with it. In fact, reflections and NLOS conditions generated by the human body cause a relevant change of the channel impulse response, even considering the same location in the environment.

Despite the obtained results may pose the basis for the validity of the model, some critical aspects of the presented work have to be reported and discussed in detail. First, regardless a lot of attention has been posed in the dataset collection, it surely presents a weakness in terms of completeness and it does not include the whole statistics of the indoor test scenario. This is evident by the fact the accuracy drops down when evaluating the network on a new, completely unseen set of measurements. This suggests even more time and consideration has to be taken in the dataset collection and analysis. Second, another weakness of the presented research is that too short time has been spent in the optimization of the deep learning model, both in terms of network design and hyperparameters and learning. The results obtained from the Grad-CAM algorithm show that in most cases a few samples are used to perform the classification of a location, given the input channel impulse response. Designing a more complex and accurate network may result in the extraction of more hidden and deep features, helping the differentiation of a specific indoor location from another.

From the analysis performed up to this point the author believes that the proposed localization technique is viable only when the complete statistics of the environment is represented in the collected dataset. The obtained result can be attributed to the extremely large statistical variability of the considered scenario, which may vary enormously from one day to the next. The rich statistics of the problem may however require an extremely long data collection phase, which may take this characterization method very costly in absence of other sensory information that may facilitate the localization. The use of additional sensory information, together with transfer learning techniques and mixed-inputs deep neural networks can be tackled in follow-up researches.

Appendix A

MoDecaRanging PC application

The debug section of the DecaRanging PC application has been used to collect the dataset. In particular, the source code of the DecaRanging PC application has been modified and a new version has been built in Visual Studio so as to store additional values in the *.log* file produced by the software during the TWR measurements. One can refer to it as *MoDecaRanging* PC application.
The EVK1000 uses STMicrolectronics STM32 ARM cortex M3 μcontroller and employs the STM32 USB driver from STMicrolectronics. It allows the DecaRanging PC application to communicate to the DW1000 on the EVB1000 board over the Virtual COM port (over USB) interface via "USB-to-SPI". In this way, there is the possibility to read and write data from/to the DW1000 IC accessing the registers.

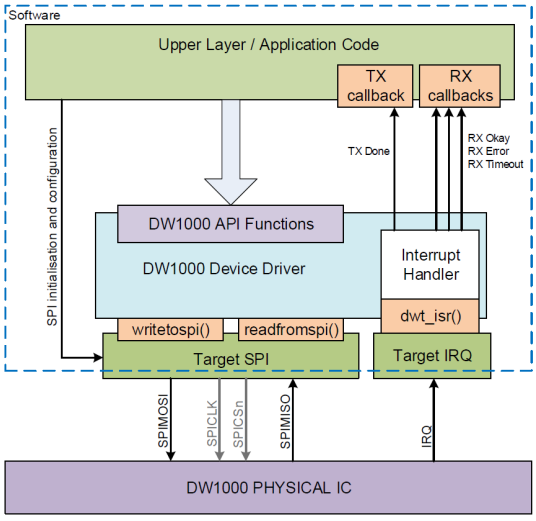


Figure A.1: General software framework of DW1000 device driver

With reference to Fig. A.1, the DW1000 device driver controls the DW1000 IC through its SPI interface. More precisely, the DW1000 device driver abstracts the target SPI device by calling it through generic functions *writetospi()* and *readfromspi()*. The control of the DW1000 IC through the DW1000 device driver software is achieved via a set of API functions, provided by Decawave (details can be found in [38], [39]). In particular, the uint16 $dwt\_read16bitoffsetreg$(int regFileID, int regOffset) is used to read a 16-bit DW1000 register that is part of a sub-addressed block. As an example, the SNR can be obtained from the receive signal power, computed as:

$$RXLevel = 10 \times log_{10}\left(\frac{C \times 2^{17}}{N^2}\right) - A\ [dBm]$$

and,

$$SNR = RXLevel + delta$$

where A and $delta$ are constants, N is the preamble accumulation count and C is the Channel Impulse Response Power (CIR_PWR), stored in the register file 0x12.

REG:12:00 – RX_FQUAL – Rx Frame Quality Information (Octets 0 to 3, 2x16-bit values)																												
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FP_AMPL2																STD_NOISE															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

REG:12:04 – RX_FQUAL – Rx Frame Quality Information (Octets 4 to 7, 2x16-bit values)																												
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CIR_PWR																PP_AMPL3															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure A.2: Register file 0x12 - Rx Frame Quality Information

Referring to Fig. A.2, it is possible to retrieve C as dwt_read16bitoffsetreg(0x12, 0x6), since every octet spans 8 bit.

Having clarified how to extract additional data from the registers of the DW1000 IC, some of those values have been included in the *.log* file generated via the *MoDecaRanging* PC application, which heading structure is depicted in Fig. A.3. The data included in the file are then extracted by a Python script to perform data analysis and network training and testing.

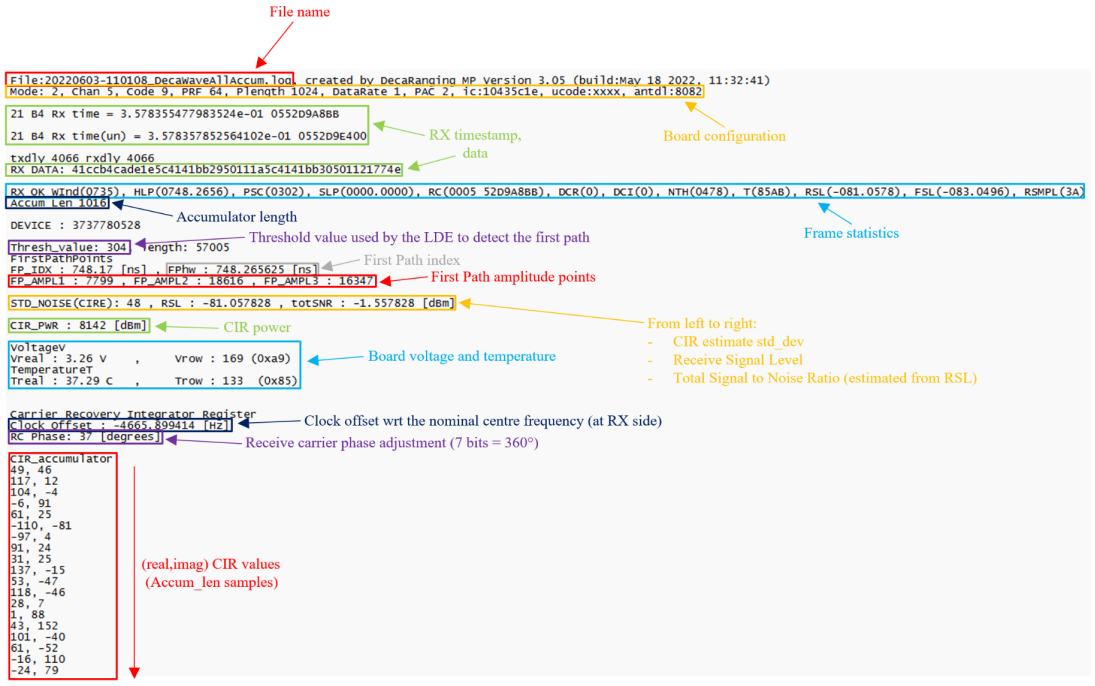


Figure A.3: LOG file heading structure

Considering that FP_AMPL1, FP_AMPL2, FP_AMPL3 are the amplitude of the 1^{st}, 2^{nd} and 3^{rd} point after ceiling Fphw respectively, a sub-portion of the Channel Impulse Response extracted from the *.log* file is shown in Fig. A.4.

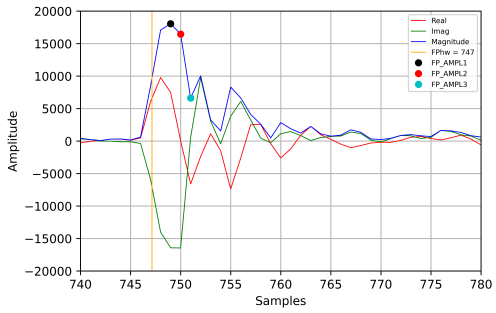


Figure A.4: CIR first path amplitude points - zoom

Bibliography

[1] H. Nikookar and R. Prasad, *Introduction to ultra wideband for wireless communications*. Springer Science & Business Media, 2008.

[2] M.-G. Di Benedetto, *UWB communication systems: a comprehensive overview*. Hindawi Publishing Corporation, 2006.

[3] S. M.-S. Sadough, "A tutorial on ultra wideband modulation and detection schemes," *Shahid Beheshti Univ. Fac. Electr. Comput. Eng, Tehran, IR Iran, no. April*, pp. 1–22, 2009.

[4] D. Tse and P. Viswanath, *Fundamentals of Wireless Communication*. USA: Cambridge University Press, 2005.

[5] H. Obeidat, A. Alabdullah, E. Elkhazmi, W. Suhaib, O. Obeidat, M. Alkhambashi, M. Mosleh, N. Ali, Y. Dama, Z. Abidin, *et al.*, "Indoor environment propagation review," *Computer Science Review*, vol. 37, p. 100272, 2020.

[6] A. A. Saleh and R. Valenzuela, "A statistical model for indoor multipath propagation," *IEEE Journal on selected areas in communications*, vol. 5, no. 2, pp. 128–137, 1987.

[7] A. Goldsmith, *Wireless Communications*. Cambridge University Press, 2005.

[8] B. Großwindhager, C. A. Boano, M. Rath, and K. Römer, "Concurrent ranging with ultra-wideband radios: From experimental evidence to a practical solution," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1460–1467, 2018.

[9] P. Corbalán and G. P. Picco, "Ultra-wideband concurrent ranging," *ACM Trans. Sen. Netw.*, vol. 16, sep 2020.

[10] Qorvo, "Dw1000 user manual." `https://www.qorvo.com/`, 2017.

[11] "Evk1000 evaluation kit." `https://www.qorvo.com/products/p/EVK1000`.

[12] P. Ongsulee, "Artificial intelligence, machine learning and deep learning," in *2017 15th international conference on ICT and knowledge engineering (ICT&KE)*, pp. 1–6, IEEE, 2017.

[13] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into deep learning," *arXiv preprint arXiv:2106.11342*, 2021.

[14] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data mining and knowledge discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[15] A. J. Bagnall, A. Bostrom, J. Large, and J. Lines, "The great time series classification bake off: An experimental evaluation of recently proposed algorithms. extended version," *CoRR*, vol. abs/1602.01711, 2016.

[16] A. Bagnall, J. Lines, J. Hills, and A. Bostrom, "Time-series classification with cote: The collective of transformation-based ensembles," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 9, pp. 2522–2535, 2015.

[17] J. Lines, S. Taylor, and A. Bagnall, "Hive-cote: The hierarchical vote collective of transformation-based ensembles for time series classification," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 1041–1046, 2016.

[18] M. Längkvist, L. Karlsson, and A. Loutfi, "A review of unsupervised feature learning and deep learning for time-series modeling," *Pattern Recognition Letters*, vol. 42, pp. 11–24, 2014.

[19] Towards Data Science, "Applied deep learning - part 4: Convolutional neural networks." `https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2`.

[20] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2921–2929, 2016.

[21] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *2017 International joint conference on neural networks (IJCNN)*, pp. 1578–1585, IEEE, 2017.

[22] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[23] Q. Wen, L. Sun, F. Yang, X. Song, J. Gao, X. Wang, and H. Xu, "Time series data augmentation for deep learning: A survey," *arXiv preprint arXiv:2002.12478*, 2020.

[24] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[25] T. DeVries and G. W. Taylor, "Dataset augmentation in feature space," *arXiv preprint arXiv:1702.05538*, 2017.

[26] J. Yoon, D. Jarrett, and M. Van der Schaar, "Time-series generative adversarial networks," *Advances in neural information processing systems*, vol. 32, 2019.

[27] B. Zhou, A. Khosla, À. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," *CoRR*, vol. abs/1512.04150, 2015.

[28] A. R. Jiménez Ruiz and F. Seco Granja, "Comparing ubisense, bespoon, and decawave uwb location systems: Indoor performance analysis," *IEEE Transactions on Instrumentation and Measurement*, vol. 66, no. 8, pp. 2106–2117, 2017.

[29] K. Bregar and M. Mohorčič, "Improving indoor localization using convolutional neural networks on computationally restricted devices," *IEEE Access*, vol. 6, pp. 17429–17441, 2018.

[30] T. S. Rappaport *et al.*, *Wireless communications: principles and practice*, vol. 2. prentice hall PTR New Jersey, 1996.

[31] "Maximum permitted speed of dw1000 nodes for correct operation." `https://www.decawave.com/wp-content/uploads/2018/10/TB001_Max_DW1000_Speed.pdf`.

[32] Keras, "Gaussian noise layer." `https://keras.io/api/layers/regularization_layers/gaussian_noise/`.

[33] Keras, "Batch normalization layer." `https://keras.io/api/layers/normalization_layers/batch_normalization/`.

[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, p. 1929–1958, jan 2014.

[35] Keras, "Dropout layer." `https://keras.io/api/layers/regularization_layers/dropout/`.

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

[37] W. Zhu, W.-C. Yeh, J. Chen, D. Chen, A. Li, and Y. Lin, "Evolutionary convolutional neural networks using abc," pp. 156–162, 02 2019.

[38] "Dw1000 software api guide." `https://usermanual.wiki/Pdf/DW1000SoftwareAPIGuiderev2p4.1120642274/html`.

[39] "Decaranging pc source code guide." `https://forum.qorvo.com/uploads/short-url/fPJbrN3ctkaWvPsatdg33N6qxLA.pdf`.