University of Padova

Department of Information Engineering

Master Thesis in ICT for Internet and Multimedia Engineering

Human Pose and Skeleton Reconstruction with Deep Neural Networks from MmWave Radar Point Clouds

Supervisor
Prof. Michele Rossi
University of Padova

Master Candidate
Luca Michelon

Academic Year

2021-2022

Abstract

The foreseen adoption of millimeter-wave (mmWave) communication systems in our everyday life has increased the research interest on using this technology for *sensing* applications. Previous work has shown that the reflections of the signals transmitted by a mmWave device can be used to localize objects or people in the environment and analyze their movement. Higher carrier frequencies, with their large bandwidth availability are key to obtain better spatial resolution, effectively allowing the usage of dedicated mmWave radar devices for the fine-grained analysis of human gaits. This is of particular interest in bio-mechanical motion tracking systems used in clinical and rehabilitation contexts, which are typically based on expensive and impractical marker-based devices. In this work we design and implement a deep-learning framework for mmWave radar-based motion tracking of human gait, which can reconstruct the position of a set of key points on the human body from the raw radar signal during motion. The proposed system extracts point clouds representing the moving subject, making use of signal processing and tracking techniques to remove noise from static objects and interference. Then, we adapt a state-of-the-art Neural Network (NN) for point clouds, Pointnet++, to our sparse and noisy radar data and propose a modification of the architecture to exploit the temporal correlation of radar point clouds. The results obtained on our own dataset show promising results despite the technical limitations of the motion tracking system used as a ground truth to train the proposed NN. Our system can reconstruct the position of 30 markers placed on the subject's body with an average accuracy of 17 cm, paving the road for future work on mmWave markerless motion tracking based on a larger and more diverse set of measurements.

Sommario

L'introduzione su larga scala di dispositivi di comunicazione a onde millimetriche (mmWave) ha portato a un rinnovato interesse per le applicazioni che sfruttano questa tecnologia per attività di monitoraggio dell'ambiente circostante (*sensing*). Diversi studi hanno dimostrato che le riflessioni dei segnali trasmessi da questi dispositivi possono essere usate per localizzare oggetti o persone e analizzarne i movimenti. La maggiore disponibilità di banda, grazie all'utilizzo di frequenze più elevate, ha permesso la realizzazione di dispositivi radar a onde millimetriche in grado di analizzare nel dettaglio il movimento delle articolazioni delle persone. Ciò può risultare particolarmente utile in contesti clinici e riabilitativi, dove attualmente vengono utilizzati sistemi di tracciamento con marker, molto più costosi e di scarsa praticità. In questo progetto è stato sviluppato e implementato un framework di *deep learning* per individuare e tracciare alcuni punti specifici delle articolazioni, a partire dai dati grezzi ottenuti da radar a onde millimetriche su persone in movimento. Il sistema proposto è in grado di estrarre una nuvola di punti corrispondente al soggetto in movimento, utilizzando tecniche di elaborazione del segnale e tracciamento per rimuovere il rumore introdotto da oggetti statici e interferenze. Quindi, una versione modificata della rete neurale Pointnet++, attualmente fra le reti stato dell'arte per nuvole di punti, è stata usata sui dati grezzi prodotti dal radar, che presentano una bassa densità ed elevata presenza di rumore. La rete neurale proposta, inoltre, è in grado di sfruttare anche l'evoluzione temporale delle nuvole di punti. I risultati ottenuti su un dataset reale, raccolto come parte di questo lavoro, sono promettenti, nonostante le limitazioni dell'equipaggiamento utilizzato per il tracciamento dei marker, necessari per l'allenamento della rete neurale. Il modello di deep learning proposto è in grado di ricostruire la posizione di 30 marker con un'accuratezza di 17cm, dimostrando le potenzialità di un sistema di tracciamento senza marker, possibilmente basato su misurazioni più complete e diversificate.

Contents

Listing of figures

Listing of tables

Listing of acronyms

ADC Analog to Digital Converter.

ANN Artificial Neural Network.

AoA Angle of Arrival.

BGD Batch Gradient Descent.

BN Batch Normalization.

BQ Ball Query.

CD Chamfer Distance.

CDM Code Division Multiplexing.

CFAR Constant False Alarm Rate.

CNN Convolutional Neural Network.

DBSCAN Density-Based Spatial Clustering of Applications with Noise.

DFT Discrete Fourier Transform.

EM Electromagnetic.

EMI Electromagnetic Interference.

FC Fully Connected.

FDM Frequency Division Multiplexing.

FFNN Feedforward Neural Network.

FMCW Frequency Modulated Continuous Wave.

FOV Field of View.

FP Feature Propagation.

FPS	Farthest Point Sampling.
GAN	Generative Adversarial Network.
GD	Gradient Descent.
GRU	Gated Recurrent Units.
GUI	Graphical User Interface.
HD	Hausdorff Distance.
IF	Intermediate Frequency.
KF	Kalman Filter.
LIDAR	Light Detection And Ranging.
LP	Low Pass.
LSTM	Long-Short Term Memory.
MIMO	Multiple Input Multiple Output.
MLP	Multi Layer Perceptron.
MRG	Multi-Resolution Grouping.
MSE	Mean Squared Error.
MSG	Multi-Scale Grouping.
MTI	Moving Target Indication.
NN	Neural Network.
NN-JPDA	Nearest-Neighbors Joint Probabilistic Data Association.
NTP	Network Time Protocol.
PPD	Per-Point Distance.
PSNR	Peak-Signal-to-Noise Ratio.
RADAR	Radio Detection And Ranging.

ReLU	Rectified Linear Unit.
RNN	Recurrent Neural Network.
RTT	Round Trip Time.
RX	Receive.
SA	Set Abstraction.
SGD	Stochastic Gradient Descent.
SIMO	Single Input Multiple Output.
SPPD	Selective Per-Point Distance.
SSIM	Structural Similar Index Measure.
TDM	Time Division Multiplexing.
TX	Transmit.

1 Introduction

Many times in history technologies developed for a specific task have been reused, adapted and transformed to apply them to new problems, often yielding unexpectedly good results. What is considered a source of errors or a disturbance for a certain goal can sometimes be exploited in a new environment enabling unforeseen applications. In wireless communications, multiple copies of the transmitted signal arrive at the receiver delayed, shifted in frequency and attenuated due to the *multipath* effect. This phenomenon is caused by the objects in the environment that reflect the transmitted signal in multiple directions, including towards the receiver. This, in general, results in a disturbance from the communication standpoint, as some processing is required at the receiver side to recognize the received signals. However, using some appropriate processing, useful information on the location, movement and nature of the reflecting objects can be recovered from the multipath effect. This is the underlying principle of Radar devices, which can be even exploited to created an accurate map of the environment.

The recent advancement in telecommunications towards transmissions with higher frequency to increase the capacity of a transmission also allowed to reach much more precise localization of the objects. This is due to the large available bandwidth used, e.g., in mmWave systems, such as 5G-NR. The localization accuracy can reach a few centimeters with a few GigaHertz of bandwidth (see Ch. 2). This increase in resolution allowed this technology to be used in many more applications, effectively giving birth in the last few years to the field of *radio sensing*. Some examples include the automotive industry, where radars can be used to recognize objects and pedestrians and their activity for both assisted and self driving cars, the security sector for intrusion detection or monitoring of crowds, human computer interaction and the healthcare field to recognize mobility problems in their initial phases, or to follow physical rehabilitations. In this thesis we focus on the latter application, exploring the possibility of using mmWave radar sensing to replace existing marker-based motion tracking systems. These are typically expensive and require rooms equipped with multiple fixed cameras. Moreover they operate by tracking the position of markers placed on the body of the patient, usually directly on the skin, which may prove challenging with special patients (e.g. children) and in general may be uncomfortable. Different technologies can be used to extract this spatial information, including cameras, Light Detection And Ranging (LIDAR) and motion tracking setups. The choice of a radar system, however, has many advantages with respect to the aforementioned alternatives: it is relatively cheap, it does not require specific environments and calibration procedures and in fact could be considered as a portable device, it is not affected by lighting conditions, no device needs to be worn by the subject and no personal data is collected. At the same time, developing a radar-based motion tracking system is extremely challenging: *(i)* the reflected signals are heavily influenced by noise and unwanted reflections by external sources which can degrade the results, *(ii)* the accuracy of the spatial information on the reflectors is not on par with LIDAR-based devices and *(iii)* the required accuracy on the localization of the body parts is very high, i.e., in the order of a few millimeters.

This thesis has been developed as the initial part of a collaboration between the *Signet* and the *BioMovLab* groups at the University of Padua. One of the main goals is the reconstruction of the spatial position of some key points of the human body, in particular related to the movements of lower limbs during gait. This set of points is defined by the *IOR-GATE protocol* [1] and is tracked with a marker-based motion tracking system. Contrary to the majority of the current research on this topic that tries to identify only the joints, here the final objective is to reach a very fine precision, as some of the markers are only a few centimeters apart from each other. The current work does not achieve the desired results yet, also due to a lack of data, as there are no currently publicly available datasets that use the same protocol. Moreover, because of the technical limitations of our motion capture system and the time constraints, our own dataset was not sufficiently large and diversified. Indeed the majority of the research in this field is still based on dataset either computed synthetically from 3D meshes or with more advanced hardware that generate much more dense and uniform point clouds that would not perform as well in a real scenario. Despite these limitations, the present work showed the great potential of mmWave radars for motion tracking when paired with powerful deep learning models.

1.1 Radar and Marker Data Processing and Reconstruction

The collection and initial processing of the data requires configuring the radar with the appropriate parameters to obtain the resolution necessary to track a person. After the acquisition, the resulting point clouds are processed to identify the set of points belonging to the subject and remove the noise introduced by other objects and sources of interference. This is achieved by using both signal processing techniques and tracking algorithms. Once most of the noise has also been removed, the point clouds are matched in time with the respective markers from the motion capture system to obtain a synchronized set of data. The resulting dataset is then used to train a Neural Network model with the goal of reconstructing the position of the markers from the corresponding point cloud. We propose an adaptation of a state-of-the-art architecture for point clouds, *Pointnet++* [2], to sparse and noisy data and a modification to exploit the temporal correlation of the radar point clouds. The contributions of this thesis can be summarized as follows:

1. the development of an algorithm to match two time sequences of point clouds with variable numbers of points. The algorithm is also able to exploit possible differences in time intervals (i.e. the frequency at which the data is captured);

2. the definition of a complete pipeline to enable end-to-end training of deep learning models from raw mmWave radar data. This includes also the manipulation in time and space of the point clouds to apply data augmentation;

3. the adaptation of an advanced neural network, Pointnet++, to perform better with sparse and noisy point clouds and its modification with the addition of a Recurrent Neural Network (RNN) to exploit the temporal correlation

of point clouds in a time sequence. Moreover, the network is adapted so that its objective is general and customizable beyond pure classification nor semantic segmentation (see Sec. 3.2 and Sec. 4.3.2), allowing the reconstruction of arbitrary spatial points. Finally a *semi-supervised* training approach is introduced to speed up convergence.

The structure of the thesis is as follows. In Ch. 2 an introduction to Frequency Modulated Continuous Wave (FMCW) radar devices is presented, along with the related theoretical background. The principles of Multiple Input Multiple Output (MIMO) radars are also summarized. Ch. 3 provides and overview of neural networks: the main concepts are introduced, with a focus on the main elements of interest for this project, including the Convolutional Neural Network (CNN) and the Recurrent Neural Network. In Ch. 4 the processing pipeline of the data and the structure of the proposed neural network are described in details. A description of the algorithms used and the new ones developed is also present. Ch. 5 presents the results obtained by the processing pipeline step-by-step, along with the final evaluation of the Neural Network. In Ch. 6 conclusions are drawn and possible future advancements are briefly introduced.

1.2 State of the Art

For a complete and recent review of the current research in human joints estimation with point clouds, [3] provides details related to the different possible methodologies that can be exploited to tackle this task, including templates, features and machine learning approaches. In [4] instead the authors focus mainly on the advancements in neural networks for point clouds datasets. In this section, we will not go into the details of the different approaches already present in the literature, but only give a short overview of some interesting and advanced researches. In [5] the authors use a CNN to perform skeleton estimation with a mmWave FMCW radar by converting the 3D spatial coordinates of the point clouds to 2D heatmaps. This technique however is limited to a specific input resolution and mapping between coordinates and pixels, and does not fully exploit the positions of the points, as the mapping reduces the precision. Moreover the time dependance of subsequent point clouds is not exploited. In [6] the time correlation of the sequences is used with specially crafted CNNs that are applied on the *planar* *decomposition* of the 4D input data (3 dimensions for space and 1 for time) in two 3D elements. The authors use a FMCW mmWave radar, but they obtain the position of the target joints with a camera system synchronized with a Network Time Protocol (NTP). Related to the use of Pointnet++, and its adaptation to time sequences, [7] introduces the concept of *PointRNN* which consists of a RNN that includes the feature extraction technique of Pointnet++. This is certainly an interesting architecture, but it is tailored for point cloud future prediction rather than key point localization.

2

Overview of Mmwave Radar Devices

This chapter includes an overview of Radio Detection And Ranging (RADAR) devices and their main characteristics. The analysis will focus on a subset of features, the most important ones for the scope of this thesis, and in particular the aspects that have a real impact for an application of human sensing and detection. In Sec. 2.1 the main hardware components of the radar are briefly introduced, while in Sec. 2.2 one specific type of radar, the FMCW radar is introduced. In Sec. 2.3 formulas for both range and velocity estimation and resolution will be derived and in Sec. 2.4 the concept of MIMO radars will be presented. An analysis on the estimation and range of the angle of arrival is also included to present all the necessary components to obtain the spatial $3D$ position of an object as seen by the radar.

2.1 Introduction to Radar Technology

Radars are electronic devices capable of emitting Electromagnetic (EM) signals which can be either reflected or scattered by one or multiple targets, depending on their surface, respectively smooth or rough. The difference between the transmitted and the received signals is used to compute properties of interest of the targets, including their position and velocity.

A radar usually consists of five main modules:

- a **transmitter** that generates the EM waves;
- an **antenna** that introduces the EM waves into the propagation medium, usually the atmosphere;
- a **receiver** that performs mixing on the reflected EM signals and includes an Analog to Digital Converter (ADC);
- a **circulator** or switch that connects both the transmitter and the receiver to the antenna and provides mutual isolation;
- a **signal processor** that further processes the digital signal to collect the measurements and, possibly, perform the detection of the targets.

The received signals are subject to four different forms of interference: electronic *noise*, Electromagnetic Interference (EMI) from other sources, both unintentional and intentional (*jamming*) and reflections from objects different from the intended target [8].

Radars can be classified according to the positioning of their Transmit (TX) and Receive (RX) antennas, either on the same device or separated, and to the modulation of the signal they transmit. Regarding the modulation, the most widely used radar type for sensing applications is the FMCW radar.

2.2 FWCM Radars

FMCW radars transmit signals called *chirps*. A chirp is a trigonometric function, usually a sinusoid, whose frequency increases linearly with time. A chirp is characterized by a start frequency f_c, bandwidth B and duration T_c. The slope of the chirp S is computed as:

$$S = \frac{B}{T_c}. \tag{2.1}$$

The frequency at any given time can thus be computed as:

$$f(t) = f_c + \frac{B}{T_c}t = f_c + St. \tag{2.2}$$

The transmission of a sequence of N equally spaced chirps in time is called *frame*. Each frame is then associated to a single static capture of the environment

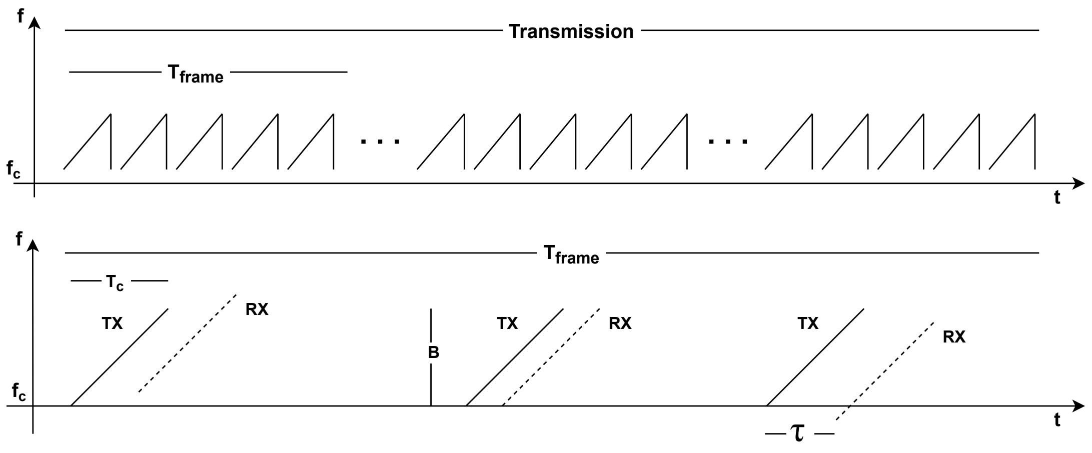

Figure 2.1: Visualization of a transmission with chirps.

at time t. To capture the evolution in time of the environment, multiple frames can be repeated with time T_{frame} and thus form the sawtooth pattern in Fig. 2.1 in the time-frequency domain.

The received signals from the target objects are mixed with the original ones with the formula [9]

$$x_{if} = x_{TX}x_{RX} = \left(\frac{A_{TX} \cdot A_{RX}}{2}\right)\cos[2\pi(f_{TX} - f_{RX})t + (\phi_{TX} - \phi_{RX})], \qquad (2.3)$$

$$\text{with } \ x_{TX} = A_{TX}\cos(2\pi f_{TX}t + \phi_{TX})$$

$$x_{RX} = A_{RX}\cos(2\pi f_{RX}t + \phi_{RX})$$

to obtain the Intermediate Frequency (IF) signals, also known as *beat* signals. The mixer, as depicted in Fig 2.2, includes a Low Pass (LP) filter so equation 2.3 does not include the summation term in the trigonometric formula for the product of cosines

$$\cos(\alpha)\cos(\beta) = (\cos(\alpha+\beta) + \cos(\alpha-\beta))/2. \qquad (2.4)$$

2.3 Information Content in Radar Signals

Different quantities of interest can be obtained from FMCW radars: from the frequency of IF signals we can obtain the radial distance of an object, while from

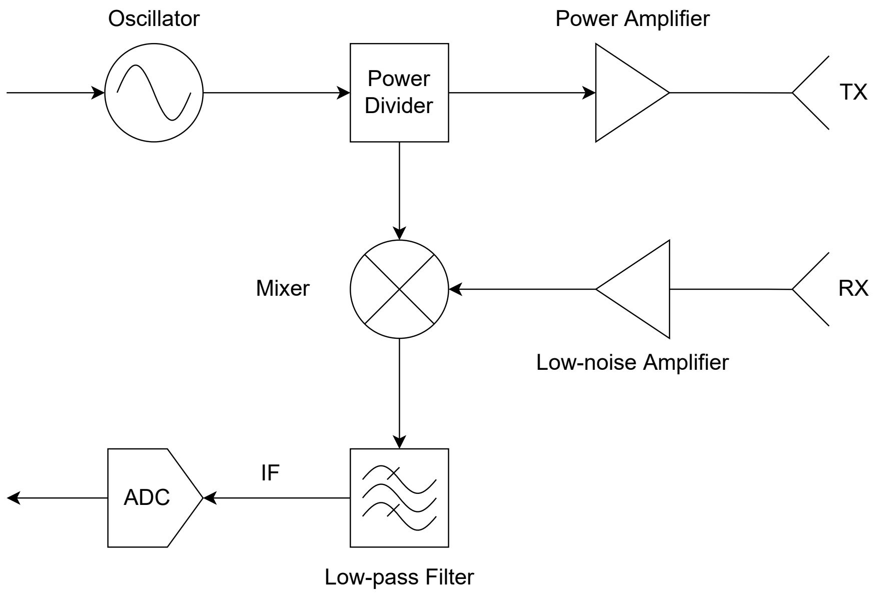

Figure 2.2: Block diagram of a mixer. Adapted from [10].

the phase we can derive its *Doppler velocity*.

2.3.1 Range Estimation and Resolution

The time it takes a transmitted signal to reach a target object at distance r and be reflected back is [11]

$$\tau = \frac{2r}{c} = \Delta f \frac{T_c}{B}, \tag{2.5}$$

which is the Round Trip Time (RTT) for a given distance r, assuming that the velocity of propagation, c, is the speed of light. It follows that

$$r = \frac{\tau}{2}c = \Delta f \frac{T_c}{2B}c. \tag{2.6}$$

Given the formula for an IF signal as in Eq. 2.3 we can apply a 2D Discrete Fourier Transform (DFT) on the IF signal in time domain and obtain the same result as in 2.5 that the frequency of a IF signal is [12]

$$\Delta f = S\frac{2r}{c} = S\tau. \tag{2.7}$$

In particular we can see that the frequency depends only on the distance and does not allow to distinguish objects that occupy spatially distinct positions but at the same radial distance from the radar. In fact, this DFT generates a peak for each value of r where a target is detected and is thus called *range-DFT*.

The maximum distance that can be correctly identified by the radar, r_{max}, determines a maximum IF frequency of (Eq. 2.7)

$$\Delta f_{max} = \frac{S2r_{max}}{c} \tag{2.8}$$

and therefore if the ADC on the radar has a fixed sampling rate F_s, for the Nyquist-Shannon theorem [13], to avoid *aliasing* it is necessary that $F_s > \Delta f_{max}$ and thus

$$r_{max} = \frac{F_s c}{2S}. \tag{2.9}$$

On the other hand, the minimum distance that allows a radar to distinguish two near objects as independent is called range resolution and it is a parameter to optimize, according to the type of sensing application involved. For the properties of the DFT, the minimum distance for two objects to appear as independent peaks in the range-DFT is for them to have a difference in frequency $\Delta f > 1/T_c$. Substituting Eq. 2.7 we obtain [11]

$$S\frac{2\Delta r}{c} > \frac{1}{T_c} \tag{2.10}$$

$$\Delta r > \frac{c}{2ST_c} \tag{2.11}$$

$$\Delta r > \frac{c}{2B} \tag{2.12}$$

which implies that the range resolution depends only on the bandwidth. This is an important result as it decouples the dependency of the range resolution to the carrier frequency and allows to reach very high accuracy as long as a sufficient bandwidth is available. As a direct consequence the designer of a FMCW radar is free to choose the carrier frequency of choice depending on other constraints, for example the bandwidth availability or desired accuracy on the Doppler velocity. Currently, a lot of communication devices operate with frequencies below 6 GHz, so a mmWave radar that operates at more than 60 GHz can exploit more bandwidth availability.

2.3.2 Velocity Estimation and Resolution

To distinguish two objects at the same radial distance we can not rely only on the frequency of the IF signals, but we can analyze their phase to obtain their Doppler velocities. In fact, a small displacement of an object will result in a evident change in the phase of its IF signal, due to the small-scale Doppler effect induced by the movement of the object. This takes the name of *μ-Doppler* effect.

For an object at distance r the phase is defined as

$$\phi = \frac{4\pi r}{\lambda} \tag{2.13}$$

and in particular the phase difference between two subsequent chirps is

$$\Delta\phi = \frac{4\pi \Delta r}{\lambda} = \frac{4\pi v T_c}{\lambda} = \omega \tag{2.14}$$

from which we can compute

$$v = \frac{\lambda \omega}{4\pi T_c}. \tag{2.15}$$

Eq. 2.15 gives the Doppler velocity provided that we know that the two IF signals we are considering come from the same object. In the general case where we receive multiple reflected signals, to be able to distinguish the different reflectors, we need to transmit a repetition of chirps equally spaced, i.e. a frame. We then apply a range-DFT to each one of the groups of IF signals associated to a chirp and obtain a matrix that contains the peaks in range of the reflected signals. By applying a 2D DFT, called *doppler-DFT*, on the matrix of their phases it is then possible to distinguish the different targets at the same range and their velocities as seen in Fig. 2.3.

To obtain an unambiguous value for the velocity, i.e., to distinguish when the target is moving towards or away from the radar, we must assume that

$$|\omega| < \pi \tag{2.16}$$

and from Eq. 2.15 we obtain the maximum measurable velocity by a FMCW radar

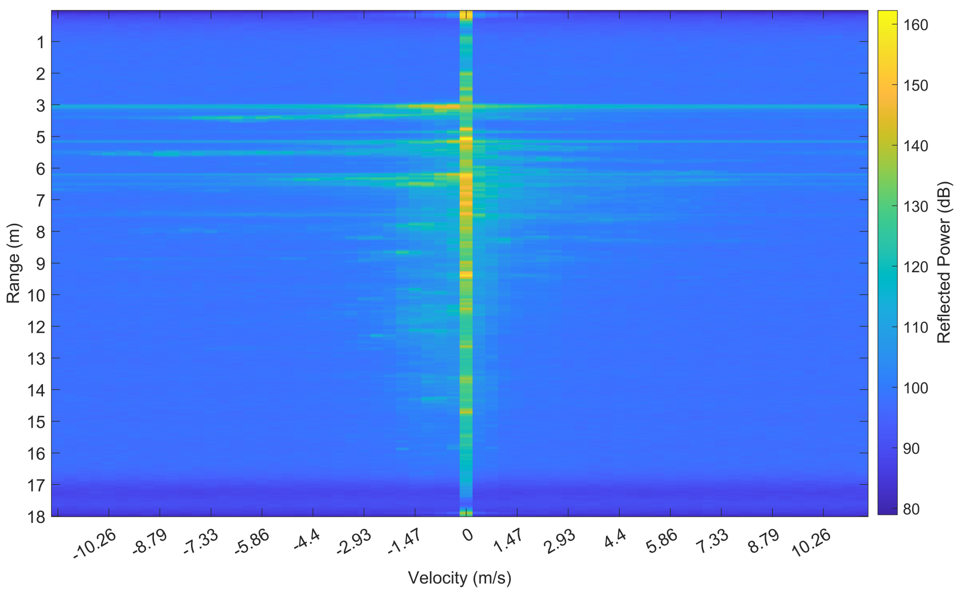

Figure 2.3: 2D Doppler-DFT of IF signals.

[12]

$$v_{max} = \frac{\lambda}{4T_c}. \tag{2.17}$$

For the properties of the Fourier transform a DFT is able to separate two frequencies ω_1 and ω_2 if

$$|\omega_1 - \omega_2| > \frac{2\pi}{N} \tag{2.18}$$

where N is the number of bins used for the DFT. From Eq. 2.18 and Eq. 2.14 we obtain the velocity resolution

$$v_{res} = \frac{\lambda}{2T_f} = \frac{\lambda}{2NT_c} \tag{2.19}$$

where T_f is the length of a frame, and N the number of chirps in a frame.

2.4 MIMO

A FMCW radar with a single antenna to transmit and receive is able to determine only the distance and the velocity of a target object. To compute its spatial location it is necessary to know also the Angle of Arrival (AoA) of the received

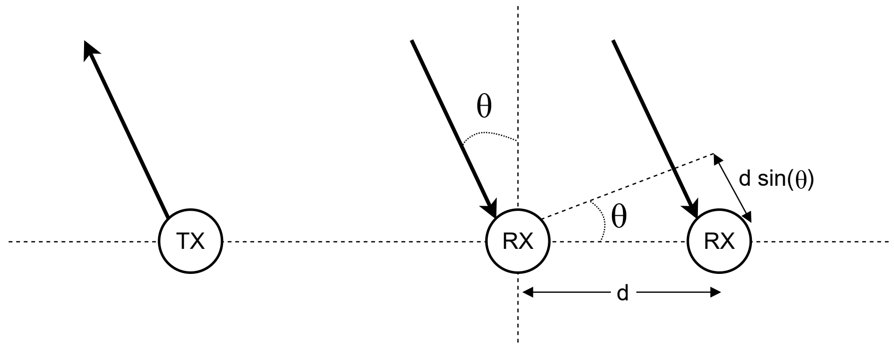

Figure 2.4: AoA estimation in a SIMO radar.

signals, both in elevation and azimuth. This can be achieved by using multiple antennas spaced apart on the radar.

Two main configurations can be set up for a multi antenna configuration:

- **Single Input Multiple Output (SIMO)** refers to a radar with a single TX antenna and multiple RX antennas. The angle resolution depends directly on the number of RX antennas: a higher number of antennas results in a better resolution. Each RX antenna must have however its own independent hardware processing chain and this poses some limits in terms of efficiency and scalability.

- **Multiple Input Multiple Output (MIMO)** refers to a radar with multiple TX antenna and multiple RX antennas. The main advantage of this configuration is that, given N_{TX} TX antennas and N_{RX} RX antennas, its angle resolution is equivalent to the angle resolution of a SIMO radar with $N_{TX} \times N_{RX}$ antennas, but requires less hardware.

2.4.1 Angle of Arrival Estimation

In this section we will analyze a SIMO radar with two RX antennas, but these considerations can be easily applied also to any number of RX antennas.

If the TX antenna transmits a signal, it will be received by both the RX antennas, but the distance that the RX signal will have to travel will depend on the location of the antennas. Fig 2.4 shows that the additional length that the signal needs to travel to reach the second RX antenna is, thanks to the *far-field* *approximation*,

$$\Delta r = d \sin(\theta) \tag{2.20}$$

where d is the distance between the two RX antennas and θ is the angle of arrival. From Eq. 2.20 and 2.14 we can easily obtain θ

$$\theta = \sin^{-1}\left(\frac{\Delta r}{d}\right) = \sin^{-1}\left(\frac{\omega\lambda}{2\pi d}\right) \tag{2.21}$$

where we can note that it differs by a factor 2 from Eq. 2.14 because here only the RX signal has to travel the additional Δr distance. In this case, from Eq. 2.16 we obtain the maximum angle that the radar can estimate, called Field of View (FOV)

$$\theta_{max} = \pm \sin^{-1}\left(\frac{\lambda}{2d}\right) = \pm 90 \tag{2.22}$$

which is obtained when the distance between the RX antennas is $d = \lambda/2$.

Similarly to the doppler-DFT we can apply a DFT on the 2D peaks of the doppler-DFT to distinguish objects with equal range and velocity, but different Angle of Arrival as seen by the two RX antennas. This procedure is called *angle-DFT*.

To compute the angular resolution we look at two object with AoA θ and $\theta + \Delta\theta$, the phase difference from Eq. 2.14 and 2.20 is

$$\begin{aligned}\Delta\omega = \frac{2\pi\Delta d}{\lambda} &= \frac{2\pi d}{\lambda}\left(\sin\left(\theta + \Delta\theta\right) - \sin\left(\theta\right)\right) \\ &\approx \frac{2\pi d}{\lambda}\cos\left(\theta\right)\Delta\theta\end{aligned} \tag{2.23}$$

as $\frac{\delta}{\delta\theta}\sin\left(\theta\right) = \cos\left(\theta\right)$. For the properties of the Fourier transform we have

$$\begin{aligned}\Delta\theta &> \frac{2\pi}{N} \\ \frac{2\pi d}{\lambda}\cos\left(\theta\right)\Delta\theta &> \frac{2\pi}{N} \\ \Delta\theta &> \frac{\lambda}{Nd\cos\left(\theta\right)}\end{aligned} \tag{2.24}$$

which implies that the angle resolution depends on the Angle of Arrival, and in

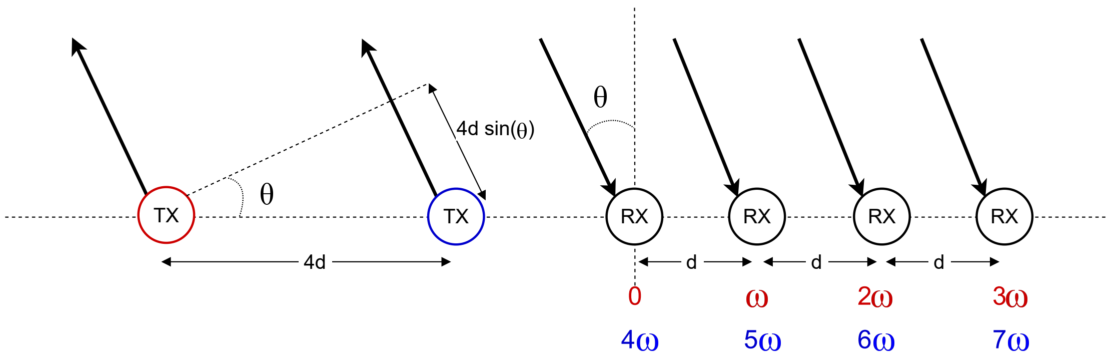

Figure 2.5: MIMO configuration in a radar.

particular is at its maximum with $\theta = 0$ and thus

$$\begin{aligned}\theta_{res} &= \frac{\lambda}{Nd} \qquad (2.25)\\ &= \frac{2}{N} \quad \text{with} \quad d = \frac{\lambda}{2}\end{aligned}$$

where N is the number of RX antennas.

2.4.2 Principles of MIMO Radars

From the previous section we obtained that the resolution on the AoA depends on the number of RX antennas, but adding more antennas has important drawbacks, as discussed earlier.

We now assume that we have a MIMO radar with 2 TX antennas and 4 RX antennas and that the RX antennas are spaced apart by a distance d such that the difference in phase at each RX antenna is a multiple of ω, as depicted in Fig. 2.5. If the second TX antenna is at a distance $4d$ from the first antenna, the transmission signal will have to travel an additional $4d\sin(\theta)$ according to Eq. 2.20 which results in a total phase difference at the receiver antennas of 4ω more than with respect to the one seen from the first TX antenna. This is the same result that we would have obtained if we had a total of 8 RX antennas in a SIMO radar, but only a single TX antenna was added.

In general, we say that a MIMO radar with N_{TX} TX antennas and N_{RX} RX antennas properly positioned, generates a virtual array of $N_{TX} \times N_{RX}$ virtual antennas. By applying the same concept also to the elevation we can obtain a 2D

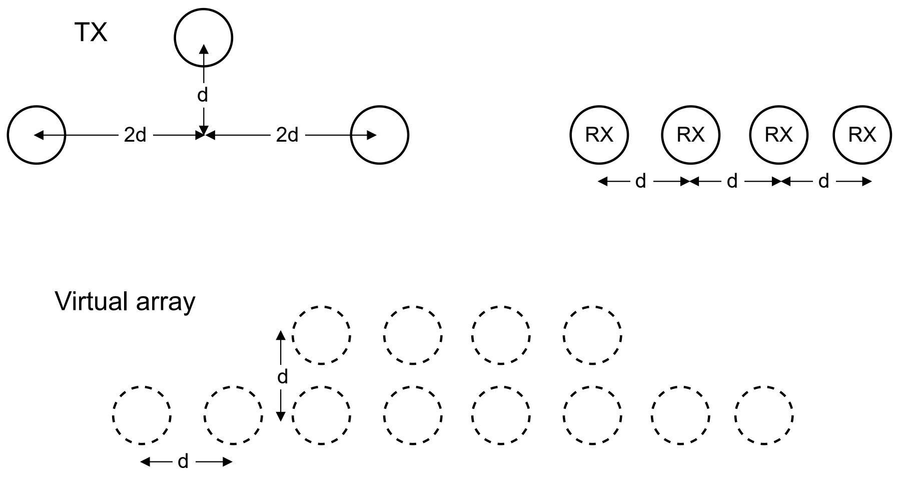

Figure 2.6: Array of virtual antennas in a MIMO radar.

virtual array as in Fig. 2.6.

2.4.3 Multiplexing

In MIMO radars the same RX antennas need to process the signals from different TX antennas, and to do so they need to be able to distinguish the reflected signals according to the TX antenna that sent them. There are multiple ways to achieve this, and they usually belong to the category of *multiplexing* techniques, which are widely adopted also in telecommunication technologies, such as in cellular networks. The property that allows signals to be distinguished is their orthogonality and it can be achieved in three different ways:

- **Time Division Multiplexing (TDM)** achieves orthogonality in the time domain: each transmitted frame is divided in blocks, and each block corresponds to a chirp transmitted in sequence, one at a time, by each TX antenna. This technique, however, does not take full advantage of all the transmission capabilities of the device, as only a single TX antenna is active at any given time.

- **Frequency Division Multiplexing (FDM)** achieves orthogonality in the frequency domain by transmitting the same chirp with all TX antennas that operate at different sub-bands so that they do not overlap. Contrary to TDM, here we transmit with all the TX antennas at the same time, but the available bandwidth is split among them.

- **Code Division Multiplexing (CDM)** achieves orthogonality thanks to the encoding of the signals themselves that make them orthogonal to each other. This allows to transmit at the same time with all TX antennas and occupy the full bandwidth, but requires some additional processing at the receiver.

3 Introduction to Neural Networks

This chapter provides an introduction to Artificial Neural Network (ANN), sometimes referred to as just Neural Network (NN), both as a general architecture and more in details for the components that have been used in this project. Sec. 3.1 will present the main concepts of Neural Networks and in Sec. 3.2 different learning paradigms will be introduced with a focus on their principal use cases. Sec. 3.3 will introduce the loss functions typically used in ANN and in particular those used in this project, while in Sec. 3.4 and Sec. 3.5 a short introduction to the optimization of the training procedure and to some well known problems and solutions linked to it are available. Finally Sec. 3.6 and Sec. 3.7 present two architectures of NN, the convolutional and recurrent NN respectively.

3.1 Neural Networks

In the last decade Artificial Neural Networks have seen a sharp increase in adoption and development in many different sectors ranging from healthcare [14] to agriculture [15], to surveillance [16] thanks to their flexibility and easiness of implementation.

Historically we may consider as the first example of ANN the *perceptron*. The perceptron was initially developed as a binary classifier that could be taught to approximate a binary function. The teaching procedure involves providing both the input and the output real values of the function to the perceptron, which would internally try to predict the output and adjust its knowledge if the result was wrong. By repeating this step with different input values the perceptron learns an approximation of the target function. Assuming that the target function is the *sign* function, given as input a set of real values $\mathbf{x} \in R^n$, and the corresponding set of signs $\mathbf{y} \in \{0, 1\}$, the objective of the perceptron is to find a vector of weights $\mathbf{w}$ and a constant term, called *bias*, b such that for every tuple $(x_i,\, y_i)$, it holds that $\text{sign}(w_i x_i + b) = y_i$ or formally

$$\text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b) = \mathbf{y} \tag{3.1}$$

The algorithm works with an iterative procedure that updates the vector of weights and the bias whenever an input element is misclassified. It can be proved that if the input set is linearly separable the perceptron converges to a perfect binary classifier [17]. Neural Networks are an extension of this algorithm as in principle they are able to approximate any generic function, not only binary functions.

We can define Neural Networks at a fundamental level as weighted graphs $G = \{N, E\}$ where each node, called *neuron*, is connected to the other nodes with weighted edges. Each neuron receives in input a value $\mathbf{x}$, weighted according to the input edges E_{in}, and with a bias b added. The output value of the neuron is obtained by applying a non-linear function $\sigma(\cdot)$, called *activation function*, to the input value.

$$\mathbf{y} = \sigma\left(\sum_{e \in E_{in}} w_e \mathbf{x} + b\right) \tag{3.2}$$

It should be noted that in the literature the bias may be collapsed in the vector of weights: in this case it usually correspond to the value w_0 which is always associated to an input value $x_0 = 1$. This notation will be used from here on. Formally a Neural Network is thus determined by the triplet (V, E, σ), which is called the *architecture* of the network. Contrary to the initial implementation of the perceptron, the activation function in Neural Networks is not the *sign* function, nor is it usually linear; if a linear function were used, for the linearity of the function itself, the whole network would behave as a single perceptron, thus eliminating the flexibility that non linearity introduces in the architecture according to the *Universal approximation theorem* [18].

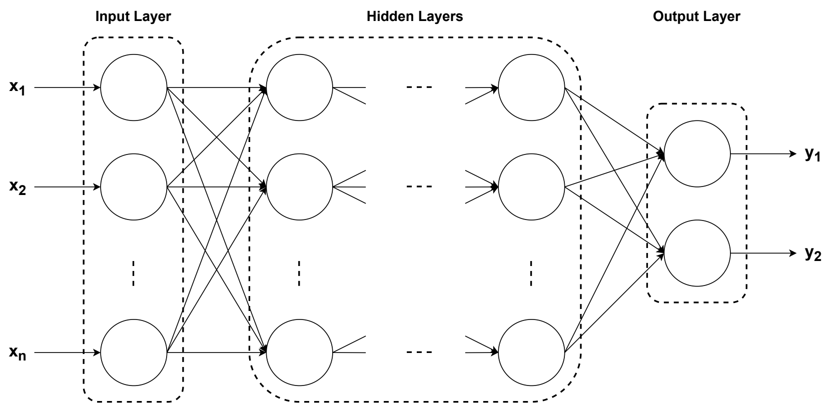

Figure 3.1: Graph structure of a feedforward Neural Network.

The activation functions that are most commonly used are the *sigmoid* $\sigma(z) = 1/\left(1 + e^{-z}\right)$, the *hyperbolic tangent* $\sigma(z) = \tanh(z)$ and the *Rectified Linear Unit (ReLU)* $\sigma(z) = \max(0, z)$.

The simplest form of ANN is called *Feedforward Neural Network (FFNN)* and its architecture is a directed acyclic graph with possibly different choices for the activation function. To visualize the structure of a FFNN as in Fig. 3.1 we say that the neurons are organized in *layers*, which are nonempty disjoint subsets of nodes N_t such that every edge in E connects some node from layer N_{t-1} to N_t and $\bigcup_{t=1}^{T} N_t = N$. If all the nodes of layer N_t are connected to all the nodes of layer N_{t+1} $\forall t \in T$ the network is said to be *fully connected.* The output of each node j at layer $t+1$ can thus be computed as [17]

$$y_{t+1,j} = \sigma\left(\sum_{i:(n_{t,i},n_{t+1,j})\in E} \mathbf{w}\left((n_{t,i}, n_{t+1,j})\right) y_{t,i} + b_t\right) \tag{3.3}$$

which is the activation function applied to the weighted sum of the outputs of the neurons at layer t.

The layer N_0 is called *input* layer, the layer N_T is called *output* layer, and all layers $N_1, ..., N_{t-1}$ are called *hidden* layers. When a network has more that 5 or 6 layers they are usually called *deep networks.*

3.2 Supervised, Unsupervised, Reinforcement Learning

As in the case of the perceptron, a Neural Network learns by comparing its prediction with the expected result and by updating the values of the weights and bias whenever its prediction is wrong. In Sec. 3.4 the training procedure will be presented more in details, while in Sec. 3.3 the main functions used for the comparison will be provided, here instead we want to focus on the different goals of the networks and on the source of the expected results.

According to the learning task, ANN can be divided in these main categories:

- In **Supervised learning** the input data is always provided in tuples (x, y) where each input value is accompanied by its expected result usually referred to as *label* [19]. The collection of labels is called *ground truth*. The network is then trained to match the expected results as closely as possible. The final performance of the network is usually tested with some data that was not provided during the training procedure, to avoid the situation where the network learns a direct mapping of the input data to its expected result rather than a generic mapping. This problem is called *overfitting*.

  In many cases the labels are manually generated by actual people, for example for the task of classification of images or sounds, and this historically lead to very bad performance of the first networks [20]. A breakthrough in this field was the introduction of the *ImageNet* database [21] with its collection of 3.2 million labelled images, which allowed networks to be trained with a huge amount of data and thus obtain better results.

  Networks trained with supervised learning usually have one of the following two objectives:

  - **Classification** or *pattern recognition* is the task of correctly mapping each input data to one or more predefined labels, called *classes*. If there are only two classes we call the network a *binary classifier*. If an input value must be mapped to a single class it is called *one-class classifier* [22], otherwise it is called *multiclass classifier*. The results can either be *one-hot encoded*, which is a 0-1 result for each class, or a probability density for each label.
  - **Regression** or *function approximation* is the task of learning the generating function of a set of input data, or a function that approximates it. In this case the labels are the results of the target function applied to the input values.

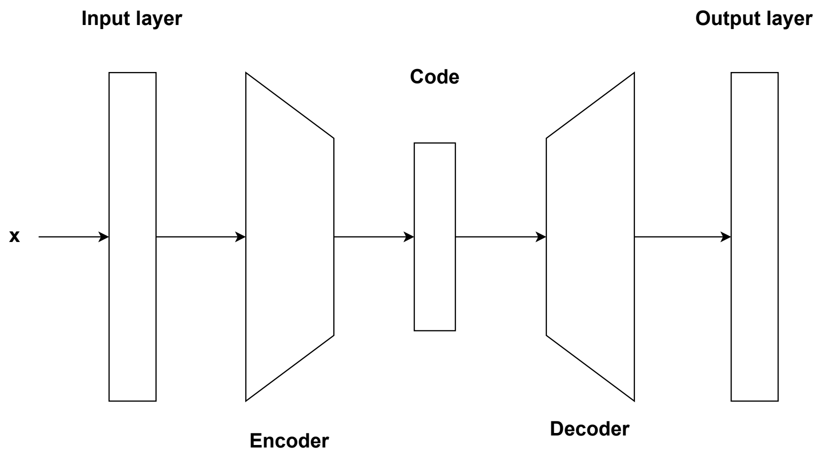

Figure 3.2: Graph structure of an autoencoder.

- In **Unsupervised learning** the input data does not include labels, but a cost function is provided as a function of the input data and the predicted result. For instance, if we want to compress an image, the cost function may be a score of how similar the compressed image is to the original data, for example with *Peak-Signal-to-Noise Ratio (PSNR)* or *Structural Similar Index Measure (SSIM)* [23].

  As already anticipated, unsupervised learning is mainly used in estimation problems and its main goal include clustering, dimensionality reduction (as in compression or denoising), pattern recognition (as in anomaly detection). One such architecture, the *autoencoder* [24], has gained popularity especially in image compression and upsampling [25] and has been implemented also in the video game industry [26, 27]. As seen in Fig. 3.2 it consists of two main components: an *encoder* and a *decoder*. The encoder performs a dimensionality reduction on the input data and gives in output a vector called *code* that will be the input to the decoder. The decoder then performs an upscale of the code to reconstruct the original data. The versatility of this architecture originates from the possibility of using the standalone encoder for compression, the standalone decoder for upsampling or the full network for denoising.

- **Semi-supervised learning** is a technique that allows to train a network with a part of labelled input data and a part of data without label. It has the advantage of combining the effective learning of precise labelled data combined with the abundance and variability of unlabelled data [28].

- **Reinforcement learning** is a paradigm in which a Neural Network, called

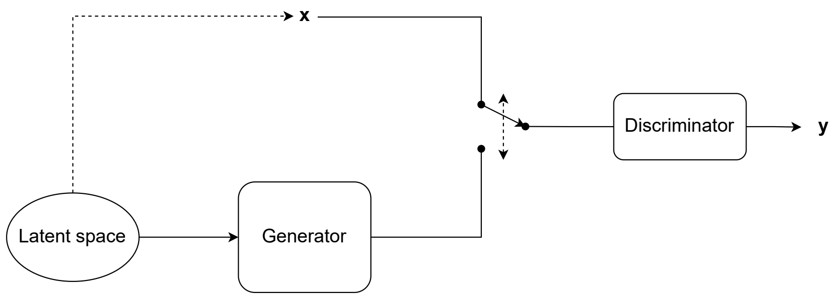

Figure 3.3: Graph structure of a GAN.

agent, tries to maximize a *reward*. The agent interacts with an *environment* through a set of *actions* that depend on the *state* of the agent and that may lead, depending on the state, to an immediate reward, but also influence future rewards. The agent starts with no prior knowledge of the environment and its task is to obtain the maximum cumulative reward by balancing the exploration of new actions and the exploitation of the current knowledge. This type of ANN resembles the core logic of many games and has been proved to be able to play and beat some of them, including against human beings [29]. On the other hand this type of network is a key component also for autonomous systems, both for self driving cars and industrial processes [30].

To conclude this section we briefly introduce one final architecture that was initially developed to help training unsupervised networks, but was later extended also to supervised learning [31], semi-supervised learning [32] and reinforcement learning [33]: the *Generative Adversarial Network (GAN)* [34]. A GAN is composed of two different networks that compete in a game where the gain of one network is the loss of the other, which is a *zero-sum game*. As depicted in Fig. 3.3 one network, the *generator*, creates new input data from a latent space from which also the real input data can be generated, while the other network, the *discriminator*, is given as input either a real input data or a fake data generated by the generator and needs to distinguish between real and fake. The generator has no knowledge of the input dataset and its weights are updated whenever the discriminator correctly identifies the data, while the discriminator is updated whenever it fails.

This network is particularly important, because it showed impressive results in the generation of synthetic portraits and reinforced the concerns related to the importance of privacy for visual images, one of the key aspects of the work behind this thesis [35].

3.3 Loss Function

We have seen in the previous sections that an ANN learns by comparing its prediction with predefined values (labels) and updating their weights accordingly. In Sec. 3.4 we will see how it is done, while in this section we define how to compute the error of the prediction from the label.

The function used to evaluate a candidate is called *objective function* and we may want to either maximize or minimize it. Usually we want to compute the error of the prediction, so we want to minimize it and we call it *loss function* or simply *loss*.

In general the most used losses are:

- **Cross-entropy** is commonly used for classification, whenever the output of the network is a probability distribution and it is usually referred to as *negative log-likelihood*. For multiclass classification it is called *Categorical cross-entropy* and is computed as

$$L = \sum_{j=1}^{M} y_j \log\left(\hat{y}_j\right) \tag{3.4}$$

where y_j is the expected outcome, $\hat{y}_j$ is the predicted output and M is the number of classes. The cost function computed on the complete output is thus

$$C = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} \log\left(\hat{y}_{ij}\right) \tag{3.5}$$

with N the number of data samples.

- **Mean Squared Error (MSE)** is usually used for regression tasks and its loss function, as the name implies is:

$$L = \left(y_i - \hat{y}_i\right)^2 \tag{3.6}$$

with the cost function defined as

$$C = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \hat{y}_i\right)^2 \tag{3.7}$$

These loss functions, however, are not well suited for point clouds. In this case, we need a loss that can capture how similar two set of points are in an euclidean space; three possibilities have been explored:

- **Chamfer Distance (CD)** between two point sets is the average minimum distance of all the points in a set from the other set [36].

  Definition 1. *(Chamfer Distance) The Chamfer Distance between two point sets S_1 and S_2 is defined as:*

  $$d_{CD}(S_1, S_2) = \frac{1}{|S_1|} \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2 + \frac{1}{|S_2|} \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|_2 \tag{3.8}$$

- **Hausdorff Distance (HD)** between two point sets measures the maximum distance between any point in one set and its nearest point in the other set.

  Definition 2. *(Hausdorff Distance) The Hausdorff Distance between two point sets S_1 and S_2 is defined as:*

  $$d_{HD}(S_1, S_2) = \max\left(h(S_1, S_2), h(S_2, S_1)\right) \tag{3.9}$$

  $$\textit{with } h(S_1, S_2) = \max_{x \in S_1} \min_{y \in S_2} \|x - y\|_2$$

- **Per-Point Distance (PPD)** is a measure of the distance between two ordered set of points, computed as the mean of the distance of each pair of ordered point.

  Definition 3. *(Per-Point Distance) Given two sets S_1 and S_2 with $|S_1| = |S_2| = N$ and a distance function d, the Per-Point Distance between the sets S_1 and S_2 is:*

  $$d_{PPD}(S_1, S_2) = \frac{1}{N} \sum_{i=1}^{N} d(S_1^{(i)}, S_2^{(i)}) \tag{3.10}$$

  where $S_1^{(i)}$ is the $i - th$ point in set S_1.

  Some examples of distance functions are:

  - **Squared euclidean distance** The Squared Euclidean Distance between two points x and y, $x, y \in R^n$ is:

    $$d(x, y) = \sum_{i=1}^{n} (x_i - y_i)^2 \tag{3.11}$$

- **Cosine Similarity** The Cosine Distance or Cosine Similarity between two points x and y, $x, y \in R^n$ is:

$$d(x, y) = \cos(\theta) = \frac{x \cdot y}{|x||y|} \tag{3.12}$$

  where θ is the angle between the vectors x and y.

- **Manhattan distance** The Manhattan Distance between two points x and y, $x, y \in R^n$ is:

$$d(x, y) = \sum_{i=1}^{n} |(x_i - y_i)| \tag{3.13}$$

Both Chamfer and Hausdorff Distances can be used for unordered set of points and are well suited for unsupervised learning approaches, however they can not be used to reliably learn ordered set of points by definition. The PPD loss instead can be used as a simple and fast way to compute different type of distance metrics between two ordered set of points with equal cardinality.

3.4 Training and Optimization

From this section we will consider a NN trained with a supervised learning approach. Most of the concepts however are valid also for the other learning paradigms. In theory, Neural Networks are *universal approximators*, which means that even a FFNN with a single hidden layer can approximate any function [37], under certain assumptions. In practice, however, the size of the network would grow exponentially with the complexity of the function and be therefore impossible to implement.

We have thus established that the architecture of a network is of fundamental importance to approximate our target function. At the same time we have introduced in Sec. 3.1 the first approach to the learning paradigm with a single layer and the *sign* activation function, while in Sec. 3.3 we have seen that the correctness of the predictions can be computed and quantified only at the output layer.

We now link everything together and analyze how we can propagate the loss computed at the output layer to all the hidden layer of a NN, in a general approach independent from its architecture and activation functions, and how to update the weights to minimize the loss.

3.4.1 Gradient Descent

In this section we assume that we want to minimize a differentiable convex objective function $f(\mathbf{w})$ that depends on the weights of our networks $\mathbf{w}$ (bias included). The analysis can be extended also to non-differentiable functions by using subgradients [17].

Definition 4. *(Gradient) The gradient of a function $f : R^d \to R$ is*

$$\nabla f(\mathbf{w}) = \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial f(\mathbf{w})}{\partial w_d} \right) \tag{3.14}$$

In particular we observe that in a stationary point, corresponding to a minimum, either local or global, the gradient is a vector of zeros, while if it is computed in any other points we obtain a vector that points in the direction of steepest ascent, which is the largest increase of f in the region around the point. Intuitively, to minimize the objective function, we want to move in the direction opposite to the gradient. We can then build an iterative procedure that takes the name of *Gradient Descent (GD)* or *Batch Gradient Descent (BGD)* as follows:

1. initialize the vector of weights $\mathbf{w}^{(0)}$ with zeros, a random distribution or with an initial guess if available

2. compute the gradient of the function with weights $\mathbf{w}^{(t)}$ from its first order Taylor approximation

$$f(\mathbf{u}) \approx f(\mathbf{w}) + \langle \mathbf{u} - \mathbf{w}, \nabla f(\mathbf{w}) \rangle \tag{3.15}$$

   where (t) is the current time step

3. update the weights to move in the direction opposite to the gradient

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla f(\mathbf{w}^{(t)}) \tag{3.16}$$

   where η, called *learning rate*, controls the length of the step and thus influences the convergence speed and the accuracy

4. repeat step 2 and 3 until a stopping condition is met, for instance: $\nabla f(\mathbf{w}^{(t+1)}) = 0$ which means the gradient is zero, a set number of iterations has been reached, or the improvement on the solution is too small

where $\mathbf{w}^{(t)}$ is the vector of weights at time step t.

GD has two main disadvantages: it updates the weights only after all the input data has been processed, requiring a possibly large amount of time and memory, and it converges to the first minimum that it finds, depending on the starting weights $\mathbf{w}^{(0)}$, which in the case of non-convex functions may not be the global minimum. To solve both issues, variations of GD have been developed. The two main variations are:

- **Stochastic Gradient Descent (SGD)** performs the update of the weights for every input data, one at a time, sampled uniformly. This reduces both the time and memory footprint, but the point-wise updates are only a noisy estimate of the true gradient and may locally deviate from it (but they still add up to the true gradient). This non linear behavior on the other hand could help in avoiding local minima and can also be stabilized with the use of other techniques such as *momentum*. Momentum is used to accelerate the descent of SGD in its main direction by dampening the occasional oscillations. This is done by adding a fraction γ of the update vector from the previous time step to the current update vector [38].

$$\mathbf{u}_t = \gamma \mathbf{u}_{t-1} + \eta \nabla f(\mathbf{w}^{(t)}) \tag{3.17}$$
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \mathbf{u}_t$$

  where $\mathbf{u}$ is the update vector.

- **Mini-batch Gradient Descent** is a solution in the middle between GD and SGD. It uses mini batches of input data of size k, a parameter chosen at the beginning, and performs the update of the weights after every mini batch. This results in a reduced time and memory usage with respect to GD and more stable updates with respect to SGD, effectively balancing the two solutions.

Besides Gradient Descent, other techniques are briefly presented [39]:

- **Adagrad** [40] uses different learning rates for each parameter and adapts them at each time step, by increasing the update vector for parameters that are less frequent.

$$\mathbf{w}_i^{(t+1)} = \mathbf{w}_i^{(t)} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \nabla f(\mathbf{w}_i^{(t)}) \tag{3.18}$$

where i is the index of a parameter, t is the time step, ϵ is a smoothing factor that avoids divisions by zero and $G_{t,ii}$ is a diagonal matrix where each diagonal element is the sum of the squares of the gradients in $\mathbf{w}_i$ up to time step t.

- **Adadelta** [41] is an extension of Adagrad that reduces its strong decrease in learning rate and memory footprint necessary to store all the past gradients. To do this it considers only a window of size n of the previous gradients and does not store them directly, but keeps only a decaying average of their squares.

- **Adam** or *Adaptive Moment Estimation* [42] keeps both a decaying average of the squared gradients v_t and a decaying average of the gradients themselves m_t, similarly to momentum. The update vector is then computed as

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{3.19}$$

$$\text{with } \hat{m}_t = \frac{m_t}{1 - \beta_1^t} \text{ and } \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

with β_1 and β_2 parameters.

3.4.2 Forward Pass and Backpropagation

The last element that defines the training and learning procedure of a Neural Network is the actual computation of the gradient and the propagation of the update vectors from the output layer to the first hidden layer, according to the contribution of the neurons in the prediction. This process consists of two steps: *forward pass* and *backpropagation*.

Forward Pass

Forward pass is the step that follows the natural path of the network: at each step t, for a given input data d_t each neuron connected to the input layer, $n_{0,j}$

computes its activation according to its weight and activation function and the result is passed to the following neurons that are connected to the input layer. Each neuron $n_{l,j}$, $l = 1, \ldots, T$ of the following layer then sums all the activations from its parent neurons with Eq. 3.3 and computes its activation and so on until the last hidden layer is reached. The activations of the neurons in the output layer will be the prediction of the network $\hat{y}$, but the activations of all the nodes are needed for the backpropagation step.

Backpropagation

Once the error on the prediction has been computed with the loss function, the weights of the neurons involved in the prediction are updated according to their contribution. From Eq. 3.3 we know that the final activation is a function of all the previous activations, and for the chain rule of the derivative we have that, given $f\left(g(x)\right)$, its derivative is $\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx}$ which implies that

$$\frac{\partial L(\mathbf{w})}{\partial w_{l,j}} = \sum_{k} \frac{\partial L(\mathbf{w})}{\partial a_k} \frac{\partial a_k}{\partial w_{l,j}} \tag{3.20}$$

where a_k is the activation at each node k. We can then compute the update vector for each weight by starting from the last layer, where each activation is a function of the activations of the neurons that are connected to it, and going back recursively until we reach the first hidden layer. This procedure that in a single sweep propagates the errors from the end to the beginning of the network is thus called *backpropagation* [43].

3.5 Overfitting

The training of a Neural Network usually is not a fast and simple procedure. When all the samples in the input dataset have been used once for training, we say that an *epoch* has elapsed. Usually with Neural Networks a few hundreds of epochs are necessary to reach convergence. This means that in practice the network will see the same data multiple times and may learn an approximation of the data itself rather than of the generating function. This problem is commonly referred to as *overfitting* [44].

There are multiple techniques that can be used to mitigate this problem, and the main ones can be divided in two categories: *structural stabilization* and *regularization* [45].

Structural stabilization consists in changing the architecture of the network itself, and in particular the number of weights involved in the prediction process. A network should be scaled according to the problem it needs to solve, if it is simple, in the sense that it can be modelled with a small number of parameters, a big network would probably overfit the training data. There is no universal rule to estimate the number of parameters needed for a given problem, but a good approach consists in performing the same training with scaled versions of the same network to understand its behavior. The reduction in size can be achieved either by eliminating some weights (*pruning*) or by sharing the same weights among multiple layers [46].

With regularization, instead, we usually indicate the addition of constraints on the weights of a network, usually by limiting their values. In fact small parameters usually lead to a more generalized model, while large weights may cause big changes in activation functions and generate big variations in output even for small changes in input. This may lead to problem especially in the presence of noise in the input data. Some techniques that can be used are [47]: *weight decay* [48] which applies a penalty on the value of the weights, usually by using their L^2 norm; *weight elimination* [49] that favours strong connections by discarding smaller activations; *dropout* which randomly removes some nodes during training [50].

Besides the optimization of the network itself, to avoid overfitting, it is possible to act also on the input dataset to generalize the data. With *data augmentation* we refer to the technique that allows to extend an input dataset by adding carefully crafted variations of the original data. To make an example, in the case of images of numbers, we can rotate the images, flip them, rescale the number in the image, stretch or warp it or add noise. This will increase the data available for training and increase the capability of the network in generalizing the problem.

3.6 Convolutional Neural Networks

A *Convolutional Neural Network (CNN)* [51] is a specialized type of FFNN that is particularly good in finding relations in 2D structures, both in a temporal (e.g. time series) and spatial (e.g. images) sense. They saw a huge boost in popularity after its adoption in *AlexNet*, a Neural Network that achieved state-of-the-art performance in image classification [52]. The two main problems of fully-connected FFNN that CNN solve are:

- the huge amount of weights used to connect each neuron from layer l to all neurons in layer $l + 1$
- the lack of spatial dimension due to the flattening of the input data to a 1D array.

At the core of a CNN there is the *convolution*, a mathematical operation that substitutes the matrix multiplication of Eq. 3.3 and that is applied on the 2D representation of the input data I with a filter, called *kernel*, K that is a matrix of weights. The result of the convolution is then passed through an activation function to a single neuron in the following layer. Each neuron in the following layer is thus connected only to a subset of neurons in the previous layer, which corresponds to the region where the kernel was applied, called *receptive field*. While the size of the kernel, and thus of the receptive field, is usually small, if multiple convolutional layers are used in sequence, the nodes in the deeper layers will still have a large receptive field with respect to the input layer.

Differently from the convolution of signal processing, the convolution in a CNN is defined as [43]:

$$S\left(m,n\right) = \left(K * I\right)\left(m,n\right) = \sum_{i}\sum_{j} I\left(m-i,n-j\right)K\left(i,j\right) \tag{3.21}$$

but is in practice implemented as the *cross-correlation*

$$S'\left(m,n\right) = \left(K \otimes I\right)\left(m,n\right) = \sum_{i}\sum_{j} I\left(m+i,n+j\right)K\left(i,j\right) \tag{3.22}$$

which differs from Eq. 3.21 for the orientation of the kernel. However this is not a problem if we consider that all kernels will be flipped in the same way and the

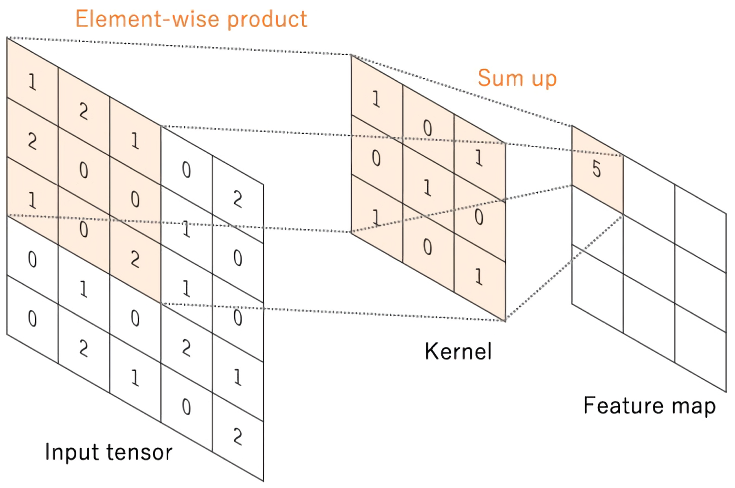

Figure 3.4: Convolution with a kernel in a CNN. Adapted from [53]

network will simply learn *flipped* features.

As shown in Fig. 3.4 the kernel slides on the input data according to a parameter, the *stride*, that controls the length of the shift at each step. To preserve the original dimension or to make input data compatible with the size of the kernel and the stride, *padding* can be applied. The result of the sweep of a single kernel on the input data generates a *feature map*. Multiple kernels then generate multiple feature maps. If the input data is composed of multiple channels, for example the RGB channels of an image, the result of the convolution on each channel is summed and the generated feature map for each kernel will be again a 2D matrix. Whenever the output of a CNN is used as input of another CNN, the feature maps of the input are treated as channels.

Convolutional Neural Networks aim at extracting features that are invariant to translation, deformation and scaling. For this reason they apply the following techniques:

- the weights are shared in the same feature map, which means that to generate a feature map the same kernel is shifted across the input data. This allows the network to be invariant to shifts as the same feature is checked across the whole input.
- convolutions are usually followed by *pooling* operations [54], which take as input a feature map and return a scaled version of it. The scaling is performed by shifting on each feature map a window of configurable size that performs

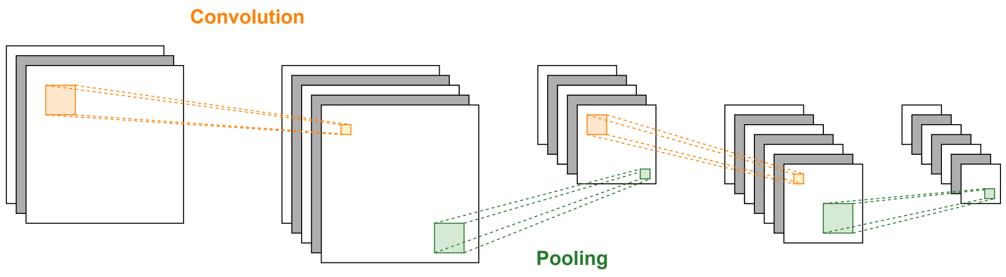

Figure 3.5: Graph representation of a CNN

a unidirectional mapping of the data, usually by taking the maximum or the average of the data in the window. The length of the shift, called stride, and the size of the windows are parameters to tune. This allows each layer to be invariant to small scale deformation of the input data.

- multiple convolutional and pooling layers are concatenated one after the other, so that small kernels can be used to extract small features at the first layers and large features at the last layers, thanks to the reduction in scale of the pooling, as shown in Fig. 3.5. This guarantees a good invariance to translations, deformations and scaling of the input data.

3.7 Recurrent Neural Networks

A *Recurrent Neural Network (RNN)* is another type of advanced ANN developed to address the lack of Neural Networks capable of exploiting events correlated in time, or in other words, time series. This section will not go into the details of the implementation or of the learning approach of RNN, but will just be a general introduction. For a more in depth analysis, [55] provides all the mathematical background.

The first written reports on *recurrent* networks date back to the *40s* [56] while in the *80s* the first mathematical analysis appears [57, 58], but it is only with the advent of discrete GPUs and their huge leap in performance from the mid *2010s* that it became possible to practically explore this new type of architecture. The input of a RNN is no more a single vector, but a time series of vectors, and its events are passed to the network in a sequential order. To learn from past inputs the RNN keeps an internal *state* or *context* that depends on the previous inputs and a matrix of weights that is shared for all input vectors.

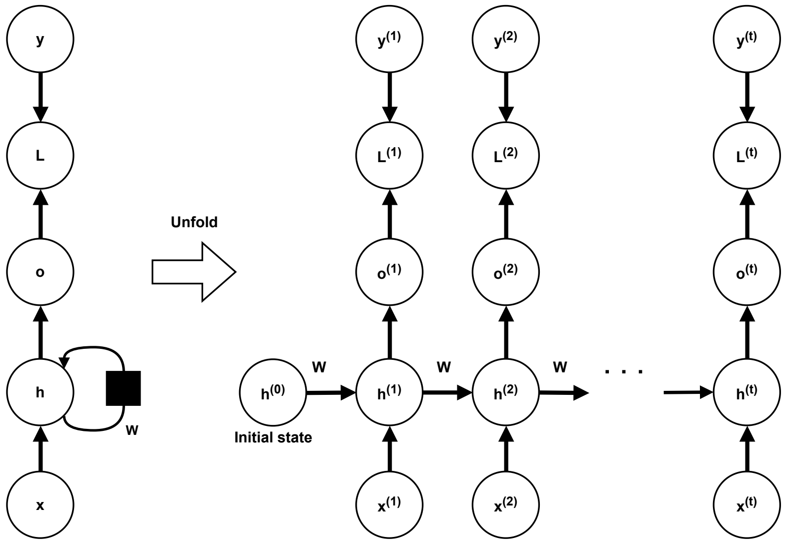

Figure 3.6: Graph representation of a RNN where **x** is the input, **h** is the state, **W** is the vector of weights, **o** is the output, **L** is the loss and **y** is the ground truth.

The term *recurrent* implies that the network can be represented in a recursive way, as depicted in Fig. 3.6, but at the same time it can also be easily unfolded in a directed acyclic graph. This implies that backpropagation can be used also in RNN and in particular it takes the name of *backpropagation through time (BPTT)*.

The main problem of standard RNN is their weak long-term temporal dependence, which is due to the *vanishing* of the gradient. With this term we refer to the reduction in the value of the gradient that we observe when we compute it at the end of a time sequence and we propagate it to the first vectors of the sequence, or to the first nodes in the unfolder graph.

Long-Short Term Memory (LSTM) networks [59] were developed to solve this issue. They are built on top of the standard RNN architecture, but they introduce the concept of *gated units*. Gated units are cells that manipulate the state at each time step by choosing the dependance on the new input and on the previous state as seen if Fig. 3.7. LSTM cells include three gates:

- the **input gate** controls what information from the new input data to save;
- the **forget gate** controls what information from the previous state to discard;
- the **output gate** filters the current state to compute the output.

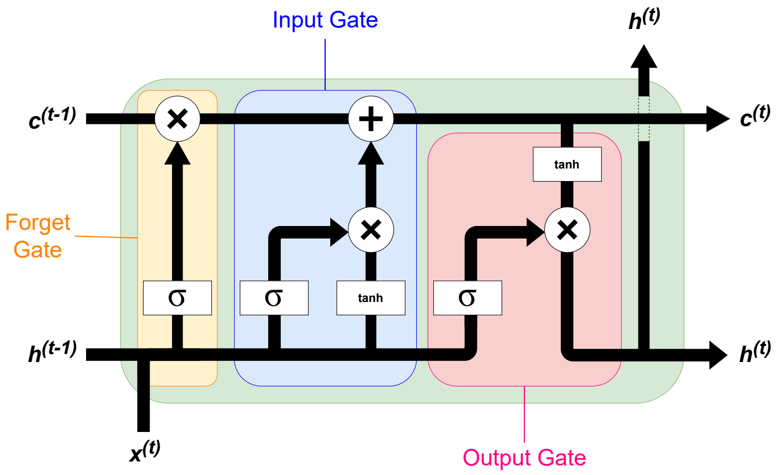

Figure 3.7: LSTM cell representation.

It is then possible to control the reduction in gradient during backpropagation by acting on the gates.

Gated Recurrent Units (GRU) [60] are a variation of LSTM that simplify the structure of the cells by removing the output gate and by using only two gates, called *update gate* and *reset gate*. This in turn reduces the overall number of parameters required for the network.

Some variation of the GRU include *Minimal Gated Units* (MGU) [61] that use a single gate and *Content Adaptive Recurrent Units* (CARU) [62] that use a content-adaptive gate instead of the update gate.

4 Proposed Solution

This chapter describes the processing framework from the raw data captured by the radar to the final output of the proposed neural network, including all the necessary algorithms implemented for the intermediate steps. The processing involves two main parts:

- the **preprocessing** of the data from the acquisition system to obtain a dataset of labelled *point clouds*. A point cloud is the set of points that represents the objects detected by the radar at a time t, also called *radar frame*, and each label contains the spatial position of the markers at time t;
- the **prediction** of the coordinates of each marker using a novel neural network architecture based on the popular PointNet++ [2].

In Sec. 4.1 the extraction of the point clouds and the ground truth is discussed in a step-by-step approach. Sec. 4.2 introduces the algorithms that have been developed and used. Finally in Sec. 4.3 further processing of the data, specifically related to its use in NN, is discussed followed by the description of the main components of the Pointnet++ architecture and of the proposed models.

4.1 Raw Data Processing

The preprocessing includes the capture of the data and its manipulation. The acquisition system is made up of a mmWave radar and a motion capture system or, in general, a device that can provide the position of the markers. A possible alternative could be a system made up of multiple cameras. In the following, we denote by *sequence* a set of ordered frames corresponding to a single data acquisition.

The pipeline of the data preprocessing steps is as follows:

1. raw data is captured concurrently by the radar and the motion capture system in a synchronous fashion;

2. the raw data is then processed in parallel:

   (a) the 3D spatial coordinates of the *valid*[*] points are extracted together with their velocity to generate point clouds;

   (b) the data from the motion capture setup is processed to generate the spatial coordinates of the markers;

3. clustering and tracking algorithms are applied on the sequences of point clouds from the radar to identify, in each frame, the subset of points corresponding to the target person;

4. for each sequence the point clouds and the markers' coordinates are compared to find the spatial shift that determines the best overlap. This procedure is needed to find the best alignment between the ground truth markers and the measured radar point cloud whenever the two capture systems can not be synchronized precisely, for instance with a NTP;

5. each sequence is divided in subsequences of equal length with overlapping for the training process. The choice of a fixed length for each subsequence is mandated by the development framework used, but allows also to restrict the dependance in time only to a limited amount of frames, both to reduce complexity and generalize the model.

[*]The definition of *valid* points is provided in Sec. 4.1.1

4.1.1 Point Cloud Extraction

As described in Ch. 2 with a FMCW radar in MIMO configuration it is possible to estimate the 3D spatial coordinates of any object that reflects the signals. In particular the raw data from the radar usually includes the distance, the velocity, the angle of arrival in both azimuth and elevation and the strength of the reflections.

With this information it is possible not only to localize the source of every reflected signal, but also to keep only the *relevant* information, i.e., the reflections from the main reflectors in the environment. This is achieved by sparsifying the signal which implies keeping only the main reflecting points through the *Constant False Alarm Rate (CFAR)* algorithm [8]. This applies a dynamic threshold on the power spectrum of the output signal, thus only retaining points that have a significantly higher reflected power with respect to their local background. Instead, to discard the reflections from static objects, the *clutter*, a high pass filter, called *Moving Target Indication (MTI)* [8], is exploited to remove the reflections with Doppler frequency close to zero. This procedure allows us to keep only the points that are probably reflected by the subject, as human beings are characterized by continuous changes in Doppler frequency [63].

4.1.2 Person Detection and Tracking

The point clouds returned by the initial radar processing will still include noise from multiple sources, including interference and moving objects. To identify only the points reflected by the target person, the typical approach is to *(i)* perform a frame by frame detection of clusters of points, based on their density, and *(ii)* to *sequentially* track these clusters using Kalman Filtering (KF) techniques [64]. The method applied here is taken from [65] and is briefly described in the following.

For the density-based detection approach, *Density-Based Spatial Clustering of Applications with Noise (DBSCAN)* [66] is the algorithm of choice as it performs well as long as the subjects to track are sufficiently separated, or, as in our case, a single subject is present [67]. It requires two parameters as input, a radius ϵ, and an integer, m_{pts}, that represent the number of points contained in the sphere of radius ϵ centered on a point for it to be considered *dense* and thus part of a cluster. For simplicity the clustering is performed on the 2D space $(x - y)$, by ignoring the z axis, the elevation.

Given a set of clusters for each frame, their centroid is computed as the *observation* of their position. At each time step k, which corresponds to a frame of the sequence, each cluster is identified by its *state* $\mathbf{s}_k = [x_k, y_k, \dot{x}_k, \dot{y}_k]^T$ which contains the position and the velocity of its points. We then assume that the state follows a constant-velocity evolution $\mathbf{s}_k = \mathbf{A}\mathbf{s}_{k-1}$ where $\mathbf{A}$ is the transition matrix. Finally we use a KF tracker to sequentially estimate the state of a cluster $\hat{\mathbf{s}}_k$ by updating the predictions based on the constant-velocity model with the positions from the observations. The clusters, and thus the observations, are associated with the corresponding tracks in subsequent frames with a Nearest-Neighbors Joint Probabilistic Data Association (NN-JPDA) algorithm [68].

4.1.3 Motion Capture Setup

The collection of the data for the ground truth requires a setup that can precisely identify the position of some specific parts of the human body. For this project the *IOR-GATE* protocol [1] was used with the *Vicon* [†] motion capture system which was proven to be very precise and stable [69]. Before starting the acquisition process, the software used, *Nexus*[‡], needs to be configured with a model (referred to also as protocol) that represents the relative positions of the markers with respect to each other. When a capture acquisition is performed the software tries to fit the model on the markers it sees and provides a label for each marker that is able to see and that is compatible with the model. If a marker is not seen or it does not fit in the model it is discarded. Due to the challenging positioning of some markers and on the suboptimal number of cameras available, some markers could not be tracked in all frames, with some of them appearing in as low as 20% of the frames. To tackle this issue and still be able to perform a supervised training, some adaptations have been performed, including a variation on the computation of the loss as described in Sec. 4.3.4.

[†]https://www.vicon.com
[‡]https://www.vicon.com/software/nexus/

4.2 Algorithms

4.2.1 Point Clouds and Markers Matching

The radar and the motion capture setup are separated systems and thus have no automatic way to match on a frame by frame basis the point clouds with the corresponding marker positions. For the training to be effective, however, it is necessary to have a dataset that for each frame of the radar associates the coordinate of the corresponding markers.

One simple and effective way to tackle this problem is to overlap the two sets, shift one of the two and find the most optimal shift according to some distance measure. Alg. 4.1 shows the pseudocode that computes the best shift given a sequence of frames of point clouds $S_r = [r_0, r_1, \ldots, r_n]$ and of marker positions $S_m = [m_0, m_1, \ldots, m_n]$ and a distance function d.

Algorithm 4.1 Point clouds and marker matching for a single sequence

```
function COMPUTE_SHIFT($S_r$, $S_m$)
    $n \leftarrow \text{size}(S_r)$
    $z \leftarrow \text{size}(S_m)$
    $shifts \leftarrow [-n, -n+1, \ldots, n]$
    $distances \leftarrow [\,]$
    for $shift$ in $shifts$
        $d_{shift} \leftarrow 0$
        for $i$ in $[0, \ldots, n)$
            if $i + shift$ in $[0, \ldots, z)$
                $r_i \leftarrow S_r\,[i]$
                $m_i \leftarrow S_m\,[i + shift]$
                $d_i \leftarrow \text{d}\,(r_i, m_i)$
            else
                $d_i \leftarrow penalty$
            $d_{shift} \leftarrow d_{shift} + d_i$
        $distances.\text{insert}\,(d_{shift})$
    $index \leftarrow \text{argmin}\,(distances)$
    return $shifts\,[index]$
```

However this algorithm does not take into account that the radar may capture data with a different frame rate with respect to the motion system, nor that we may want to add a penalty that is dynamically computed according to the number of overlapping frames, rather than adding a constant term. For the first problem, we may consider sampling from the longer sequence at the lower rate of the two, but this would discard a lot of information, in particular the frames that may be near one of the sampled values but that carry much more information (for instance frames that contain more markers positions). To solve the second problem, instead, we can also keep track of the number of overlapping frames and scale the final distance accordingly.

This leads to the revision in Alg. 4.2 where we assume that the motion capture system has a higher frame rate and its set is $S_m = [m_0, m_1, \ldots, m_z]$ with $z \geq n$.

This algorithm, however, in this form has a possible weak point: if the sequence includes a repetitive action, there could be multiple good shifts and, depending on the distance function used and on the points captured in the frames, the penalty associated to a lower overlapping may not be enough to find the real shift (as in Fig. 5.5). If timestamps are available for both the radar and the motion captures it is possible to further refine the results by leveraging the temporal correlation. If instead, as in our case, the timestamps are available only for the radar while the motion capture system returns only the time at which it began the capture, but not of any specific frame, it is still possible to obtain an approximate time difference. This time delta can be computed between the timestamp of the first radar frame and the corresponding shifted marker frame (or vice versa). For a given acquisition setup this time difference will be similar for different sequences and can thus be used to find possible outliers in the computed shifts. Sec. 5.3.3 will provide an example for this case.

Moreover heuristic considerations can be applied to speed up the computation, for example by considering a reduced range for the possible shifts, for instance if the radar captures always start after the motion captures, we can discard the negative values. If instead we know that the captured action is not repetitive and thus should show a prominent minima, we can avoid testing shifts one by one and perform an optimized search.

One final consideration regarding this algorithm is related to the distance function used. For this project the *query ball* algorithm (presented in Sec. 4.2.3) was used by considering as centroids the markers. If we assume that the markers are spread evenly on the surface of the body, the query ball algorithm reaches its minimum when the point cloud is *surrounded* by markers, however in practice the

Algorithm 4.2 Complete point clouds and marker matching for a single sequence

```
function LOOK_AROUND($set$, $index_{start}$, $index_{end}$)
    $subset \leftarrow [\,]$
    for $s$ in $set\,[index_{start}, \ldots, index_{end}]$
        $subset$.insert (size ($s$))                              ▷ Use size as the information content
    $interval \leftarrow (index_{end} - index_{start})\,/2$
    $penalty \leftarrow [1, \ldots, interval)$                    ▷ Example of linear penalty
    $penalty \leftarrow$ concatenate (flip ($penalty$) , [0] , $penalty$)
    $subset \leftarrow subset - penalty$
    $index \leftarrow$ argmax ($subset$)
    return $set\,[index_{start} + index]$
function COMPUTE_SHIFT($S_r$, $S_m$, $fps_{marker}$, $fps_{radar}$, d)
    $n \leftarrow$ size($S_r$)
    $z \leftarrow$ size($S_m$)
    $shifts \leftarrow [-z, -z+1, \ldots, -1, 0, 1, \ldots, z]$
    $distances \leftarrow [\,]$
    $fps_{diff} \leftarrow$ floor $((fps_{marker}/fps_{radar})/2)$   ▷ Interval between two sampled values
    for $shift$ in $shifts$
        $d_{shift} \leftarrow 0$
        $n_{non\_overlaps} \leftarrow 0$
        for $i$ in $[0, \ldots, n)$
            if $i + shift$ in $[0, \ldots, z)$
                $r_i \leftarrow S_r\,[i]$
                $z_i \leftarrow$ floor $(i * fps_{marker}/fps_{radar})$
                $m_i \leftarrow$ look_around $(S_m, z_i + shift - fps_{diff}, z_i + shift + fps_{diff})$
                $d_i \leftarrow$ d $(r_i, m_i)$
            else
                $n_{non\_overlaps} \leftarrow n_{non\_overlaps} + 1$
            $d_{shift} \leftarrow d_{shift} + d_i$
        $n_{non\_overlaps} \leftarrow n_{non\_overlaps}/n$           ▷ Compute non-overlapping as percentage
        $d_{shift} \leftarrow d_{shift} * (1 - n_{non\_overlaps})$      ▷ Example of distance scaling
        $distances$.insert $(d_{shift})$
    $index \leftarrow$ argmin ($distances$)
    return $shifts\,[index]$
```

radar receives most of the reflections from the surface in front of it, which could lead to a small displacement in time, or in other words in a systematic error. That however does not influence the neural network, as it would learn a systematic displacement of the positions of all the points, while their relative position would still be correct. If instead the majority of the markers are in the front or the back of the person, this could lead to a worse error if the captures are performed with different sides facing the radar: the two systematic errors will be opposite to each other and thus generate a double error.

4.2.2 Farthest Point Sampling

Farthest Point Sampling (FPS) is an iterative algorithm used to extract a subset C of elements from a set of points S where a distance function d exists such that the sampled points are as far as possible among each other, according to d, so that C should be a good approximation for S. It works as follow:

1. Choose the first point from S and add it to the set of chosen points C
2. Compute the distance from each point in S from its nearest point in C
3. Choose the point in S with the maximum distance and add it to C
4. Repeat step 2 and 3 until you reach the desired number of points in C

The choice of the first point to add to C determines the final output of the algorithm. To obtain the best result it should be executed for any possible choice of the first point and the best set C should be kept. However, to reduce the computational time, it can also be chosen at random from a uniform distribution as shown in Alg. 4.3. In this case the solution is approximate.

4.2.3 Ball Query

Ball Query (BQ), also known as *query ball point*, is a simple algorithm that, given a set of points S, a set of centroids C and a radius r, finds all the points in S that are distant at most r from any point in C as described in Alg. 4.4.

As described in Sec. 4.3.1, some points may be repeated in the same frame, and this could influence the result of the ball query algorithm, which could group the same point multiple times. Although on average points are repeated once or twice,

Algorithm 4.3 Farthest Point Sampling Algorithm

```
function FARTHEST_POINT_SAMPLING($S$, $C_{size}$, d)
    $C \leftarrow [\,]$
    $c_0 \leftarrow \text{random}(S)$
    $C.\text{insert}(c_0)$
    while $\text{size}(C) < C_{size}$
        $dists \leftarrow [\,]$
        for $s$ in $S$
            $d_{min} \leftarrow +\infty$
            for $c$ in $C$
                $d_c \leftarrow \text{d}(c, s)$
                if $d_c < d_{min}$
                    $d_{min} \leftarrow d_c$
            $dists.\text{insert}(d_{min})$
        $index \leftarrow \text{argmax}(dists)$
        $C.\text{insert}(S[index])$
    return $C$
```

as seen in Fig 4.3, we still apply a unique constraint on the source set of points to remove the duplicates and keep the structure as similar as possible to the original.

Algorithm 4.4 Revised Ball Query Algorithm

```
function QUERY_BALL_POINT($S$, $C$, $r$, d)
    $S \leftarrow \text{unique}(S)$
    $dists \leftarrow \text{d}(S, C)$                          ▷ Distance matrix $S \times C$
    $dists \leftarrow \min(dists, \text{axis} = 1)$            ▷ Min distance for each $s$ in $S$
    $mask \leftarrow dists < r$
    return $S[mask]$
```

4.3 Neural Network Architecture

The architectures proposed in this section are characterized by a common underline architecture that is in fact a revision of Pointnet++ [2]. Pointnet++ was chosen as the base model because it currently performs among the best in the extraction of features from point clouds and it is easy to modify and adapt to different purposes, for instance to sparse point clouds [4], such as the ones produced by mmWave radars. We define three different models: a baseline model with a variation of the original Pointnet++ to be more suited to our dataset and two models that take advantage also of the temporal dependance of the frames in a sequence by exploiting a RNN. Given the very specific positioning of the markers we need to predict, rather than a generic skeleton, we rely on supervised learning, but at the same time test the efficacy of a semi-supervised approach where the same network tries to learn both the markers positions and to reconstruct the original point cloud.

4.3.1 Dataset Preprocessing

Neural networks often perform best when they are trained on input datasets that are normalized to a reference interval, usually $[0, 1]$ [45]. Although in theory preserving the original dimension would simply scale the weights involved, in practice this poses other problems. In fact Gradient Descent optimizes all weights with the same pace, which implies that if one dimension, for instance the range, has higher absolute values than the azimuth and elevation, the optimization will focus mainly on the range. This consideration is particularly important when we deal with the absolute position of the points. If we consider that in our setup the subject walks towards and away from the radar, we realize that the elevation is almost constant, while the range and azimuth may change considerably inside a sequence. If we were to apply a global normalization, where we compute the maximum and minimum per-sequence, we would deform excessively the shape, while a local one (per-frame) would retain the correct proportions, provided that the normalization is performed independently on the three axis. If the three axis were to be normalized together, the result would be, again, deformed, as the absolute positions of the points may not be comparable in the three axis. This is relative to a specific action, in our case walking, but the reasoning stands also for other activities, e.g. jumping. In fact, what we observed as best performing for the learning process between the two, was not a global rescale, but a local one.

However the normalization to a fixed interval on a per-frame basis destroys the absolute dimensions of the point clouds, for instance if two subsequent point clouds are one two times wider in azimuth with respect to the other, they would both be normalized to the same interval and the difference in size would be lost. This problem can be tackled with a different approach: instead of a fixed normalization, we can shift the reference system of the point clouds from an absolute value (the position of the radar that captured the data) to a relative one, to reduce the absolute values associated to the points, while still preserving the real dimensions of the point clouds.

For this purpose we tested multiple approaches. The simplest solution would be to center each frame on the centroid (the point computed with coordinates equal to the mean of all the points of the point cloud) given by its point cloud and shift the markers by the same translation matrix. Other solutions include computing the centroid on the first or last frame of a sequence and use it for the whole sequence, to maintain the spatial movement that is lost if each frame is centered with its own centroid. The last solution that has been tested includes computing the centroid and thus the translation matrix at each frame, but use the translation from the previous frame instead of the current one for the actual shift of the point clouds. This allows maintaining the spatial movement, but only on a local level.

Another preprocessing that is applied to the input data is directly related to the learning framework, that requires that the shape of the input data be constant, which means that the sequences in input must be of a predefined length. Considering that the frame rate of our radar is 10 frames per second, we build our dataset in sequences of 30 consecutive frames, which results in 3 seconds of data acquisitions, as done in [6]. This provides a good balance between enough information to follow the movement of the subject while maintaining the computational complexity in the NN reasonable. To expand the total amount of data available for training we allow two subsequent sequences to overlap by a fixed value of 25 frames. To have a consistent input shape we also assume that each frame is made up of 256 points, which is close to the maximum number reached in the whole dataset, and in case there are not enough points we repeat some of them, sampled uniformly at random.

4.3.2 Pointnet++ Architecture

Pointnet++, also referred to as *pointnet2*, [2] is a hierarchical neural network architecture developed as an extension of the older Pointnet network [70]. Its goal is to learn a set of features that identify a point cloud by exploiting the spatial encoding of its points and aggregate them at different resolutions. Contrary to Pointnet, the process of feature extraction is performed at different scales to capture the details of both fine and large-scale structures. In principle, this could be done with a CNN, i.e., a sequence of convolutional and pooling layers. However, it would have some practical drawbacks, like discarding some of the spatial information about the input data, e.g., the density, and not being invariant to the ordering of the input points.

The processing performed by Pointnet++ can be described by the following steps:

1. partition the set of input points in regions determined by a distance metric;
2. compute local features in each region for a small neighborhood of points;
3. subsample the input set of points to obtain higher level features;
4. repeat from step 1 until you reach a sufficient abstraction.

For this procedure to be effective, three key elements must be defined: how to partition the input set, how to compute the features given an input set and how to subsample it.

For the partitioning, the original paper uses the BQ algorithm described in Sec. 4.2.3. This algorithm, given a set of centroids and a radius, returns all the points in the input set that are contained in a sphere centered in each centroid with the given radius. Moreover the authors limit the maximum number of points to associate to each centroid. For the choice of the centroids, they use the FPS algorithm described in Sec. 4.2.2 that guarantees a better coverage with respect to random sampling. The feature learner of choice at each scale is Pointnet, which provides robust results for unordered set of points, as shown in the original paper. Finally, to subsample the input sets and reduce the scale of the learning problem, the same centroids and associated spheres from the partitioning step are used.

The output of this approach is a set of features that can be used for classification. However, these are not sufficient if the task is, e.g., semantic segmentation (the problem of separately labeling each single point), or in general involves the reconstruction of the original set or a scaled version of it, as in our case. When dealing with these more challenging problems, the point features are used in layers called Feature Propagation (FP), that try to reconstruct the original set of points

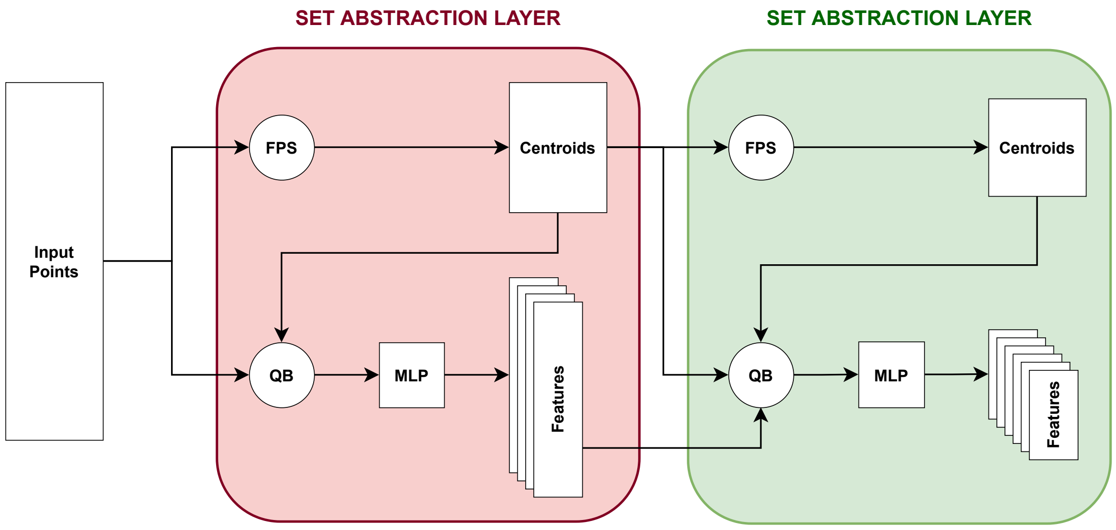

Figure 4.1: Visual representation of a Set Abstraction layer.

and associate some feature to each one, in an architecture that we could associate to an autoencoder.

Set Abstraction Layer

The *Set Abstraction (SA)* layer is used to extract the features of the input point set at a given scale. Given a set of points S with optional features f, a number of centroids N, and a radius r, the SA layer computes the set of N centroids, C, with FPS then applies the BQ algorithm to partition S and f and passes each partition to a *Multi Layer Perceptron (MLP)* layer. A MLP is implemented as a convolutional layer with a kernel of unitary size, followed by an optional normalization performed on each batch, called *Batch Normalization (BN)*. These features can be further used as input to other MLP layers with a different number of filters (kernels) and the final result is obtained after a max pooling. The feature extractor corresponds to the original Pointnet. Fig. 4.1 shows the architecture for this layer.

Grouping Strategy

One of the main contribution of Pointnet++ is its hierarchical architecture that allows the network to adapt the learning resolution depending on the density of the points. Point clouds, especially if obtained with mmWave radars, contain spaces where the concentration of points is high and others where it is low, which means that the point clouds have usually different densities in different areas. This implies that we may want to extract features at a low scale for dense areas and at a large scale in sparse areas, to capture both the details and the bigger structures. To do so the authors propose two different approaches:

1. in **Multi-Scale Grouping (MSG)** the different scales are obtained by grouping points in increasing spheres around each centroid obtained with FPS; the inscribed points are then the input of different pointnet feature extractors scaled accordingly to the input area and the final features are concatenated together. In other words, the input set S and number of centroids N are used as input to multiple SA layers with different radius $r = [r_0, \ldots, r_s]$ and the final features are concatenated. This process can be expensive, especially when the number of centroids is high. To limit the computational burden, subsampling may be applied to reduce the overall number of points and centroids between groups of SA layers;
2. in **Multi-Resolution Grouping (MRG)** the scales are obtained with a subsampling based on the centroids after each SA layer. Given an input set S, number of centroids N and radius $r = [r_0, \ldots, r_s]$, the first SA layer receives as input (S, N, r_0) and returns the features f_0, then the centroids are considered the new input set $S' = C$ and a new number of centroids to compute N' are defined for S'. The second SA layer then computes features with the new input (S', N', r_1, f_0) by concatenating the summarized features from the previous layer and the result of a new Pointnet applied on the current input.

Feature Propagation Layer

The *Feature Propagation* layer allows propagating features from the subsampled layer to the original input set. It works with a hierarchical propagation strategy by interpolating feature values based on the inverse distance weighted average from k nearest neighbors according to [2]:

$$f^{(j)}(x) = \frac{\sum_{i=1}^{k} w_i(x) f_i^{(j)}}{\sum_{i=1}^{k} w_i(x)} \tag{4.1}$$

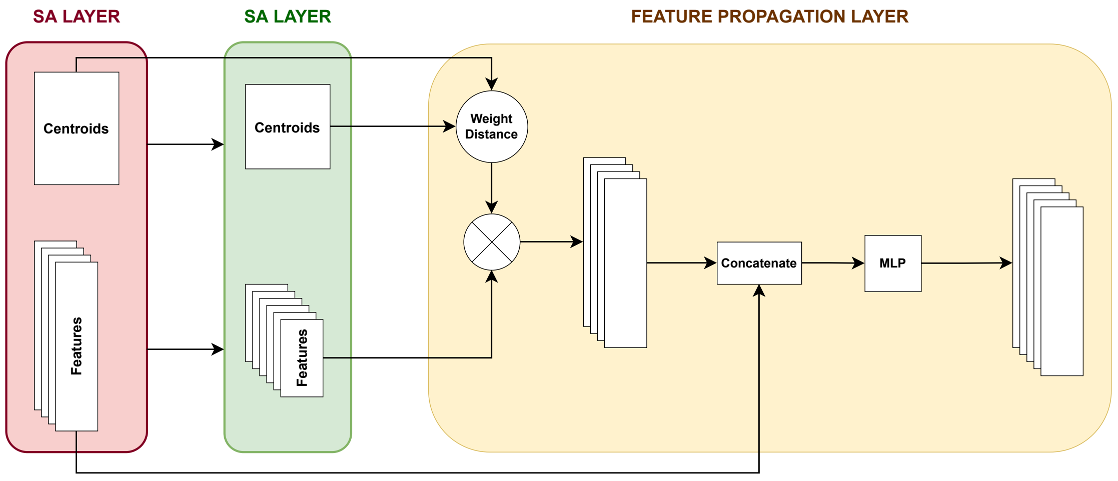

Figure 4.2: Visual representation of a Feature Propagation layer.

$$\text{with } w_i(x) = \frac{1}{d\left(x, x_i\right)^p},\ j = 1, \ldots, C.$$

Fig. 4.2 shows the connection between SA and FP layers.

4.3.3 Architecture Revisions

Pointnet++ proved to reach state-of-the-art results in multiple datasets, including MNIST [71] and ModelNet40 [72]. In both cases, however, the goal was either classification or semantic segmentation and it did not involve the construction of new points. Moreover the aforementioned datasets are made up of very dense point clouds, with several thousands sampled points per image, while in our case the radar outputs always less that 300 valid points per frame and on average 95 points with a *90-percentile* of just 170 as seen also in Fig. 4.3. Finally, those datasets were constructed by uniformly sampling 2D or 3D models, and variable densities were used only during training, while our data represents real subjects in a real environment with all the limitations that this setup implies, including missing frames, noise, and much more variable densities of the point clouds. On the other hand, each frame of our dataset is part of a sequence and as such can exploit the temporal correlation with respect to the other frames of its sequence.

All this implies that the original Pointnet++ can not be applied directly for our use case, but needs both a rescaling to match our lower resolution and a change in architecture to exploit the temporal information in a sequence.

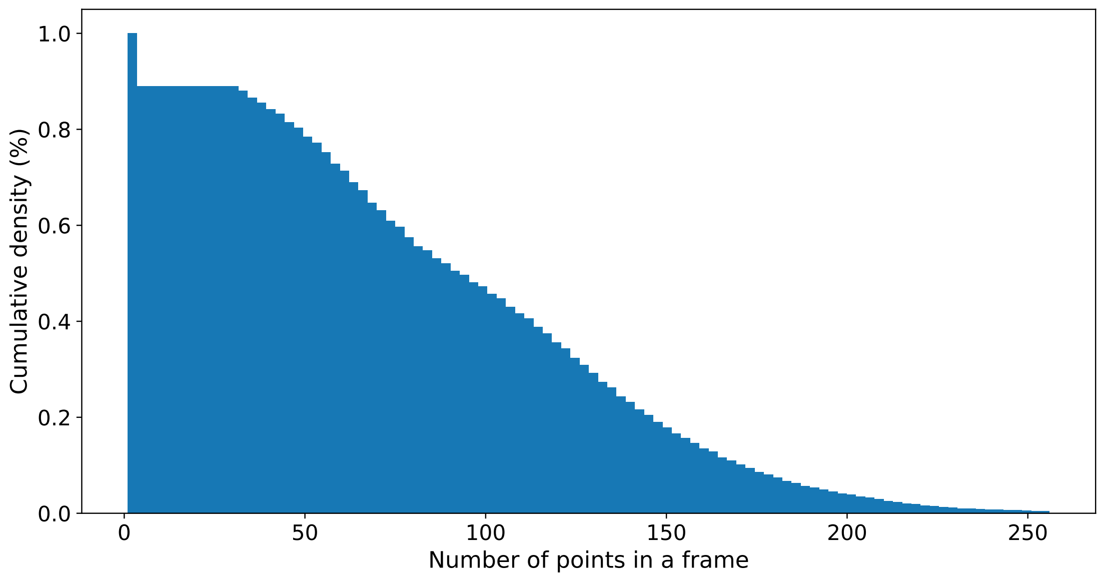

Figure 4.3: Cumulative histogram of the number of points in a frame

Baseline Model

The first aspects to consider while modifying the network architecture are the objective and the data in input to the network. Indeed, the choice of the number of centroids to extract in each SA layer and the radius of the spheres around them is a challenging task that requires the knowledge of the hardware used for the capture, the subject to capture and the conditions of the environment. This implies that the parameters used are not universal, but should be tuned for different settings.

In our case the point clouds are very sparse and the number of centroids used in the original Pointnet++ were not compatible with our data, in fact in the first SA layer the authors use 1024 centroids which is four times the total number of points in our most complete frames. Moreover the number itself of SA layers was too high and in fact we found that the best configuration for us included three and not four SA layers. For the choice of the radius we need to consider if normalization is applied, as it changes the range and thus the distance of the points. In this case our values are similar to the original ones as the coordinates are defined in a similar interval.

These considerations lead to a baseline model that we use as reference to evaluate the performance of the recurrent models. It includes three SA layers with

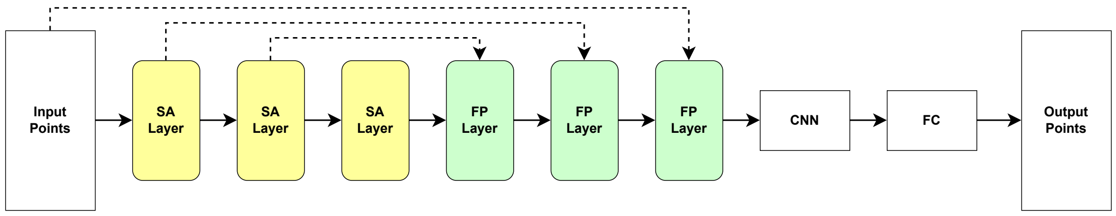

Figure 4.4: Visual representation of the baseline neural network architecture.

MRG grouping and the same number of FP layers to rescale the features. We then append a convolutional layer followed by maxpooling and a final Fully Connected (FC) layer to obtain the positions of the markers. The FP layers were included after a study on their efficacy and in all cases they provided a higher accuracy as long as the network was not too big. Indeed, the most important element to keep into consideration is the scale of the network. Given that we have a limited amount of data, it's very difficult to avoid overfitting and in particular we need to keep the number of weights to a low value. For this reason we also included the final convolution and maxpooling and considered adding another FC layer with a low number of neurons, to reduce the complexity, but our tests showed it degraded the performance. A visual representation of the baseline model is depicted in Fig. 4.4.

Recurrent Models

The design of the recurrent models is also an interesting study. The first element to focus on is the placement of the recurrent layers: we may consider using them before the SA layers, between SA and FP layers, or after the FP layers. The first possibility was discarded as RNN do not perform well with unordered data. The other two were both evaluated, including the option of removing the FP layers and directly feeding the output of the RNN to the final layers. The best numerical results were obtained by placing a single RNN layer between the SA and FP layers. Different types of RNN have been tested, including GRU and LSTM, in different combinations, for example with multiple subsequent layers and bidirectional. Our tests show that the simplest architecture, with a single GRU, was the one that performed best, possibly because of the limited amount of data available for training.

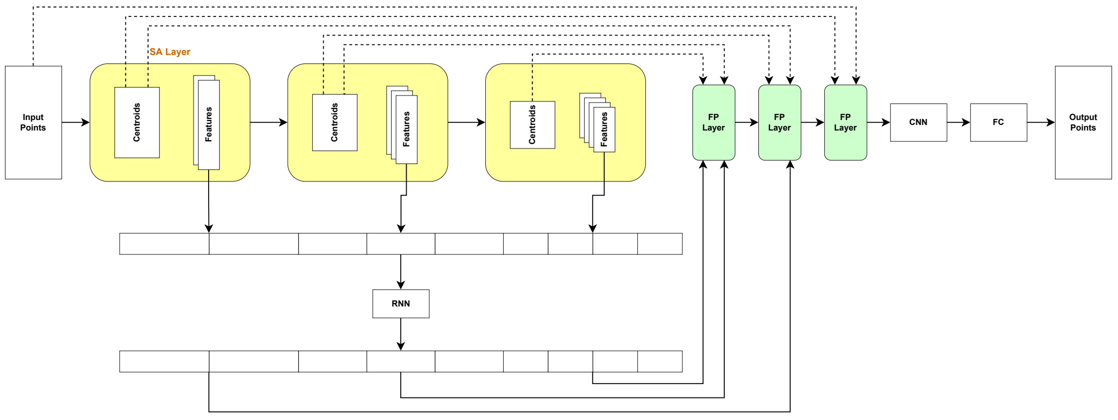

Figure 4.5: Visual representation of the ConcatGRU neural network architecture.

The last part of the design is related to the way in which the features from the SA layers are passed to the RNN. A RNN in fact requires a time sequence of 1D vectors, while each SA layer returns a list of variable length of features. We tested two possible alternatives: either concatenating the features from all the SA layers in a single vector, or using as input of the RNN the features from the SA layers, one layer at a time. In the second case we tried with RNN with and without shared weights. The different ways of encoding the features also implies a difference in the size of the RNN, as it needs to output the same number of features it receives in input for the subsequent FP layers to work properly. Adding FC layers could also be considered an alternative to restore the original size of the SA features, but from our results it did not perform as well. The tests we performed show that both ways of encoding provide similar results. In particular we observe that if we concatenate all the features, it performs best when we limit the number of filters in the SA layers to downscale the size of the RNN, while when we use the features from the SA layers one layer at a time it gives better results when the weights are shared. We define as *ConcatGRU* the model with concatenated features and *SequentialGRU* the other. The visual representations of these recurrent models are shown in Fig. 4.5 and Fig. 4.6.

To reduce the possibility of overfitting, as described in Sec. 3.5, we applied regularization to the RNN and dropout throughout the network, in particular, before, inside and after the RNN and after the FP layers. We decided not to apply pruning of points as our frames are already sufficiently sparse and with

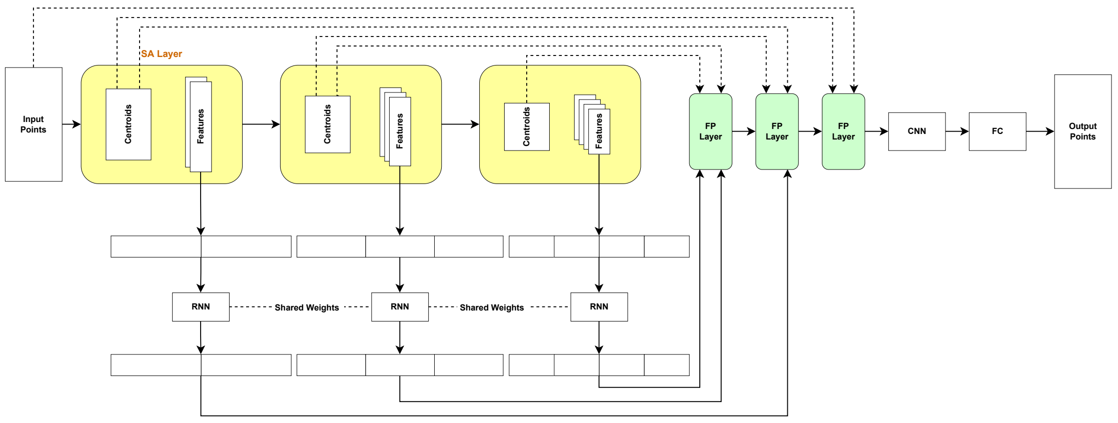

Figure 4.6: Visual representation of the SequentialGRU neural network architecture.

a strong variability between consecutive frames. Moreover the radar sometimes captures empty frames, further reducing the actual number of complete frames.

4.3.4 Selective Loss

The choice of the loss, as explained in Sec. 3.3 is particularly important, especially given the non-ideal condition of our setup. The goal of our network is to compute the position of a fixed number of markers for each frame and in particular we require that each output point corresponds to a specific marker. This implies that while the input set is unordered, as we have no control over how many points are obtained for each part of the body, the output points must be ordered, to reconstruct the correct positioning of the markers.

For this to work it is necessary that the loss be computed on a point by point basis between each pair of predicted points and markers. This is in fact the definition of the *per-point distance*. In our case the motion capture system could not determine the position of every marker for every frame. As a consequence only the visible markers could be used for the computation of the loss. Given that the learning framework requires a fixed number of markers for each frame, the simplest solution was for us to add an additional dimension that represents if a marker was recognized or not and set all those not available to 0 on a per-frame basis. This leads to a revision of Per-Point Distance which takes the name of *Selective Per-Point Distance (SPPD)* whose formula is Eq. 4.2 where M is the set of markers, P is the set of predicted points, $M^{(i,0)}$ is the position of the i-th marker and $M^{(i,1)}$ is 1 if the marker is visible and 0 otherwise.

$$d_{SPPD}(M, P) = \frac{1}{\sum_{j=1}^{N} M^{(j,1)}} \sum_{i=1}^{N} d(M^{(i,0)}, P^{(i)}) M^{(i,1)} \tag{4.2}$$

This loss is in practice the mean distance between the visible markers and their predictions. The choice of the mean rather than the sum is to avoid the increase in loss if more markers are present.

For the training procedure a semi-supervised approach has also been tested by connecting the results of the FP layers also to a decoder that reconstructs the original point clouds. For this test the loss used was the Chamfer Distance for this output and the usual SPPD for the prediction of the markers, weighted respectively by a factor of 5 and 1.

4.3.5 Data Augmentation

Data augmentation is a set of techniques that is used to perform transformations on the input data to reduce the possibility of overfitting and possibly avoid local minima in Gradient Descent. Contrary to regularization, it is not a collection of general techniques, but it is tailored to the type of data in input. Moreover, contrary to dataset preprocessing, it is applied to extend the original input data, not to change its representation.

For our tests we applied four different types of randomized transformations:

- independent **shift** of each point in any combination of the three axis x, y, z, in a predefined interval;
- global **permutation** of the axis x, y, z on a per-frame basis;
- **shuffle** of the points in a frame;
- per-frame **rotation** of the point cloud in any combination of the three axis x, y, z.

It should be noted that Pointnet++ is invariant to shuffles of the point clouds, so this type of transformation was used only to verify this property.

5 Results

In this chapter we describe the results of the whole pipeline of data processing, from the setup of the equipment to the final predictions. In Sec. 5.1 the equipment used for the acquisition of the dataset is presented, highlighting its main characteristics. Sec. 5.2 briefly describes the software enhancements developed for the radar application and the developed framework for managing the whole pipeline from the processing of the data to the training of the models. In Sec. 5.3 the results of the data processing is described step-by-step. In Sec. 5.4 the details of the proposed models and learning parameters are explained, while in Sec. 5.5 the final results are presented and discussed.

5.1 Equipment Setup

5.1.1 Data Acquisition Setup

The acquisition of the dataset that has been used for this project was performed in the bio-mechanics research laboratory of the University of Padua, in the Department of Information Engineering. The laboratory is equipped with a motion capture system from Vicon, paired with the *Nexus* software. The system includes six cameras placed on the two long walls with the radar placed on one of the shorter walls, on a table at about 80 cm of height. Fig. 5.1 shows the arrange-

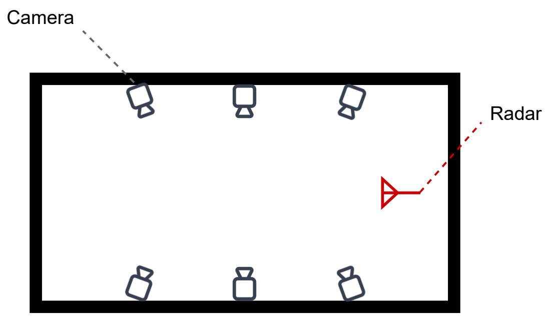

Figure 5.1: Planimetry of the laboratory.

ment of the hardware. Due to the positioning of the cameras of the Vicon system, some markers were not simple to track, in particular those in the inner side of the knees and feet, as seen in Fig. 5.2. One of the parameters to consider during the setup of the equipment is the FOV of the radar both in azimuth and elevation. This in practice limits the minimum distance from the radar that the test subject must maintain, as the limited elevation FOV can prevent a complete view of the subject's body in the radar reflected signal, as shown in Fig. 5.3.

We acquired a total of 100 sequences of 10 seconds. However, the actual usable sequences in which at least one marker is visible amount to around 8 minutes of acquisitions due to the aforementioned Vicon and radar FOV limitations.

5.1.2 Radar Parameters

The radar used for our project is the combination of the *MMWCAS-RF-EVM*[*] and *MMWCAS-DSP-EVM*[†] boards from Texas Instruments, which use four *AWR2243P*[‡] radar chips. Each chip includes 4 RX and 3 TX antennas for a total of 16 RX and 12 TX antennas. The radar operates at frequencies in the range from 76Ghz to 81GHz and can be used in different configurations, including SIMO and MIMO. For our tests we used all antennas with a MIMO setup that creates a total of 86 non overlapping virtual antennas. Tab. 5.1 shows the main TX parameters of the radar that can be used to compute the metrics that we defined in Ch. 2. Some of

[*]https://www.ti.com/tool/MMWCAS-RF-EVM
[†]https://www.ti.com/tool/MMWCAS-DSP-EVM
[‡]https://www.ti.com/product/AWR2243

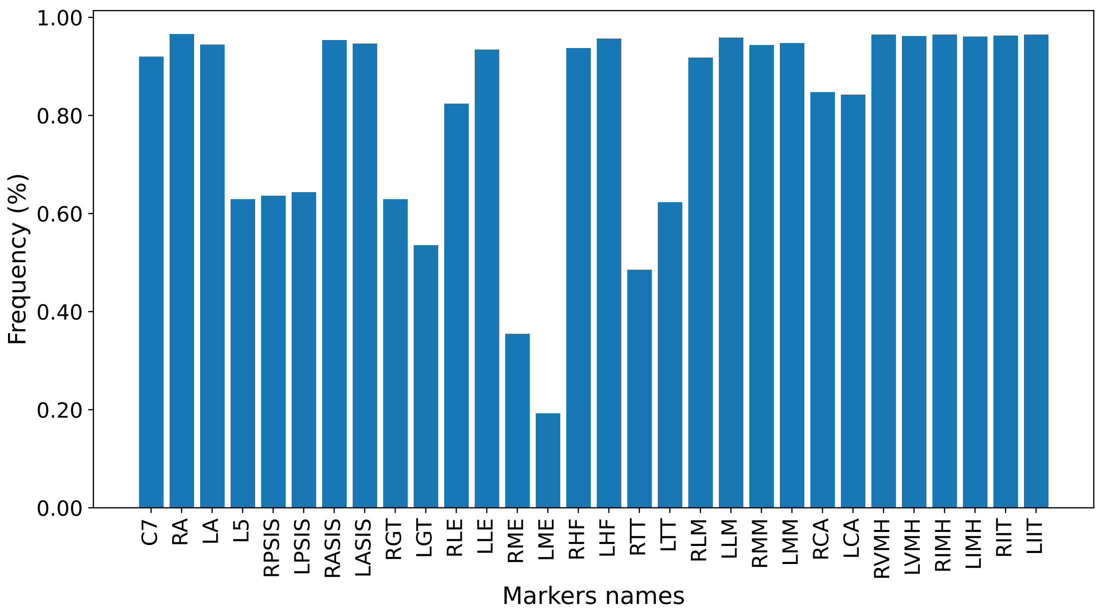

Figure 5.2: Frequency of the markers in the dataset for the IOR-GATE protocol. Refer to [1] for the positioning of the labels.

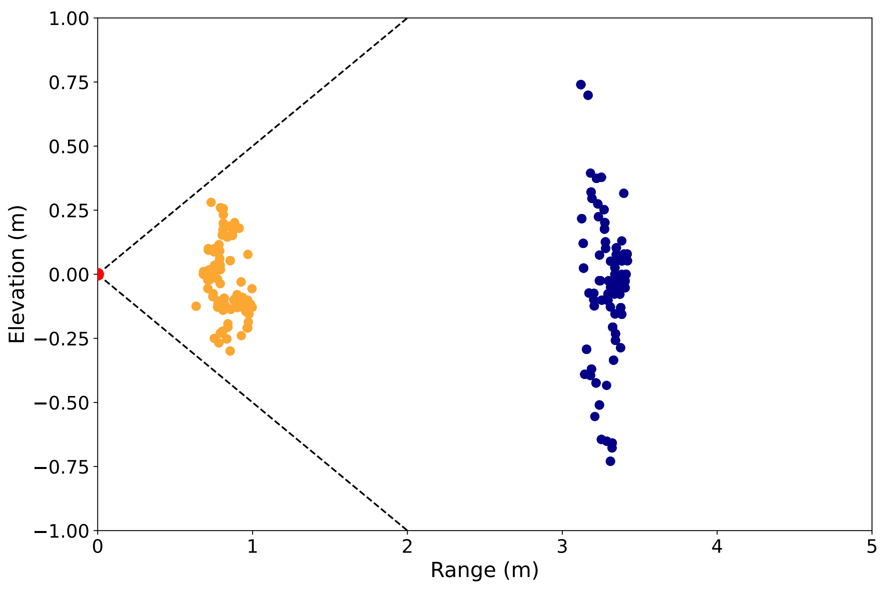

Figure 5.3: Difference between a point cloud near (orange) and far (blue) from the radar (red). The black dotted lines represent the FOV.

Table 5.1: Radar parameters

Parameter	**Value**
FOV Azimuth	$\pm\pi/3$
FOV Elevation	$\pm\pi/6$
Chirp duration (T_c)	81 μs
Chirps per frame (N_c)	64
Samples per chirp (N_s)	512
Sampling frequency (F_s)	8 MHz
Starting frequency	76 GHz
Bandwidth	4.16 GHz

them are computed hereafter: from Eq. 2.9 and Eq. 2.1 the maximum range is

$$r_{max} = \frac{F_s c}{2S} = 18.46 \text{ m}$$

with a resolution in range from Eq. 2.12 of

$$\Delta r = \frac{c}{2B} = 0.036 \text{ m}. \tag{5.1}$$

For the velocity, if we consider as carrier frequency the frequency at half chirp, $f_c \simeq 78$GHz, we can compute the maximum velocity from Eq. 2.17, remembering that $\lambda = c/f$, as

$$v_{max} = \frac{\lambda}{4T_c} = \frac{c}{4f_cT_c} = 11.72 \text{ m/s}.$$

The resolution on the velocity is instead from Eq. 2.19

$$v_{res} = \frac{\lambda}{2T_f} = \frac{\lambda}{2N_cT_c} = 0.36 \text{ m/s}.$$

5.2 Software Development

5.2.1 Radar Acquisition Software Development

As described in Sec. 4.2.1, our capture setup involves two independent systems that can not communicate with each other and thus can not synchronize precisely the beginning of a capture. Moreover, the software provided by *Texas Instruments* does not save the timestamp for each frame, nor does the software for the Vicon system, which simply saves one timestamp at the beginning of the capture. While we don't have control on the software for the motion capture system, we can execute a limited set of *lua*[§] commands on the radar interface, *mmWave Studio*[¶]. This allowed us to define both a Graphical User Interface (GUI) to easily configure the chirp parameters of the radar and start each capture sequence, and to collect the timestamp whenever a new capture is started. The set of lua instructions available does not allow to realize a GUI, so we developed a client that constantly polls a server written in *Python* which displays and is controlled by the aforementioned GUI. It should be noted, however, that the collected timestamps are not precise, as we lack control of the actual functions that interact with the Linux subsystem in the radar, and can only collect the time at which those functions are called, but not when the radar actually starts the first transmission. A finer control could be achieved only by writing a new application in C++ that directly interacts with the radar.

5.2.2 Framework Architecture

Besides the aforementioned development of the software for the radar, the original code provided by the authors[‖] has also been rewritten completely to be compatible with the latest versions of the *Tensorflow*[**] framework. The functions written in C++ have been converted to python to make it more portable and allow the training procedure to be performed also on CPU and servers with no control over the installed libraries. Besides this, a complete framework has been written from the ground up to dynamically generate new architectures by providing a list of the desired layers as a simple command line parameter or in a JSON file. This framework can be used also to control any type of data augmentation described in Sec. 4.3.5 and allows the use of wildcards to dynamically infer the shapes of the custom layers without the need to provide them manually. Full control of all the parameters of the SA and FP layers is also guaranteed, together with a limited flexibility over the main architecture: the block of SA and FP layers is fixed but multiple types of layers can be added in the middle or after.

[§]https://www.lua.org/

[¶]https://www.ti.com/tool/MMWAVE-STUDIO

[‖]https://github.com/charlesq34/pointnet2

[**]https://www.tensorflow.org/

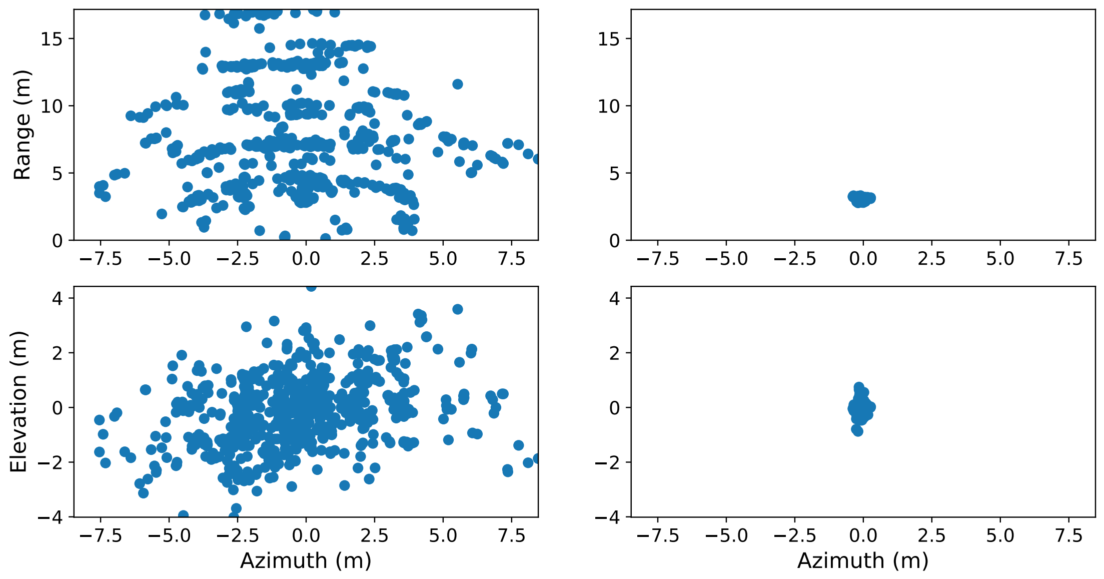

Figure 5.4: Radar point cloud before (left) and after (right) the execution of the clustering and tracking algorithms.

5.3 Data Processing

5.3.1 Clutter Removal

The removal of the clutter, the reflections with Doppler velocity close to zero, is particularly important, as it can be clearly seen in Fig. 2.3 that represents the result of the Doppler-DFT. We removed all the reflecting points with a velocity between ±0.05m/s, as they possess very low Doppler velocity and thus are probably not generated by the intended subject. The resolution of the DFT corresponds to the number of chirps per frame, $N_c = 64$, for the Doppler-DFT and to the number of samples taken for each chirp during the analog to digital conversion, $N_s = 512$, for the range-DFT.

5.3.2 Person Detection

Although the clutter removal process discards the reflections from static objects, other sources of undesired reflecting points still remain. Fig. 5.4 shows that many reflections are present with a velocity greater than the clutter removal threshold we used. From the figure the noise seems overhelming with respect to the points corresponding to the subject, but in practice this type of noise usually appears in a single frame and is different in the subsequent ones, while the points of the subjects are close in adjacent frames, and thus relatively easy to track in time. As a consequence the clustering algorithm with tracking described in Sec. 4.1.2 is able to remove most of the noise as shown in the figure.

5.3.3 Point Clouds and Markers Matching

As introduced in Sec. 5.2.1 the matching between point clouds and markers can't be based only on the timestamps. However in the case of a repetitive action, where by repetitive we mean an action that is repeated in time with similar values for the three axis $x\,, y\, z$, the matching algorithm introduced in Sec. 4.2.1 will still yield multiple minima, as seen in Fig. 5.5. In this case the lowest error corresponds to the most probable shift, as the algorithm considers the total overlapping portions of the two sequences, the point clouds and the markers positions, but the timestamps can still help in determining possible outliers. In fact, we can compute the time difference between the timestamp of the point cloud at the target shift and compare it with the timestamp returned by the motion capture system when it started the capture. In a set of acquisitions performed with the same hardware and in sequence, we can assume that these time differences should match and, in particular, if the repetitions are sufficiently separated in time, it is possible to find outliers. This time comparison may not work if the acquisitions are collected in different days or with different types of hardware as the computers may have a drift in their local time. While Fig. 5.5 shows the error associated to a person swinging his arms in place, Fig. 5.6 shows the error computed for a person walking. Contrary to the previous image, here we have a more pronounced minima and the comparison with the time stamps may not be necessary. Fig. 5.7 shows the superposition of the point cloud for the radar and the markers which we proved to be sufficient for our purposes.

5.4 Networks Architectures

The main parameters of the SA and FP layers are shared among all three models, and in particular Tab. 5.2 summarizes the parameters used for the SA layers, while for the FP layers we use the filters defined in Tab. 5.3. Batch Normalization is applied in the MLP of the SA layers, but it is not included in the FP layers. The

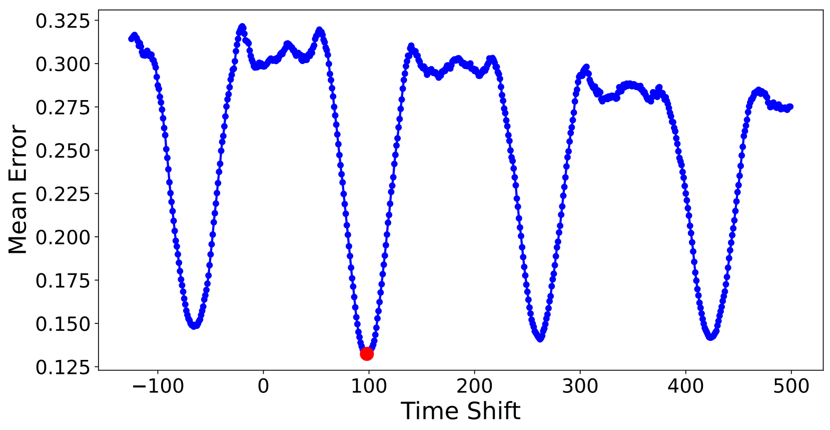

Figure 5.5: Error between markers and point clouds computed for each shift of the point cloud for an action with a repeting pattern.

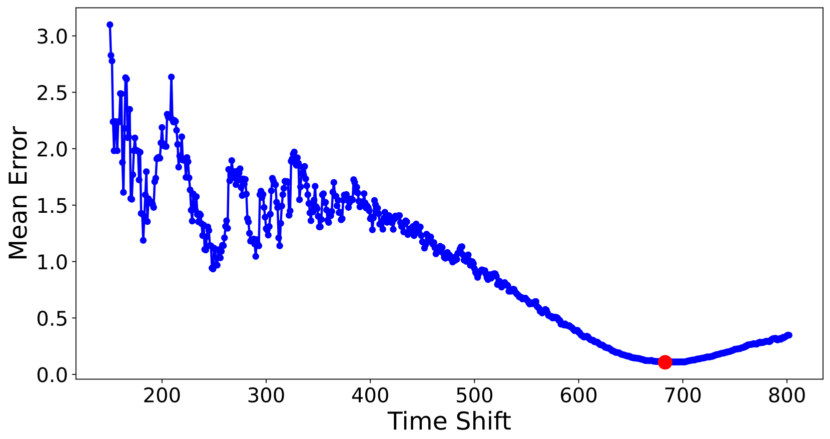

Figure 5.6: Error between markers and point clouds computed for each shift of the point cloud for an action with a unique pattern.

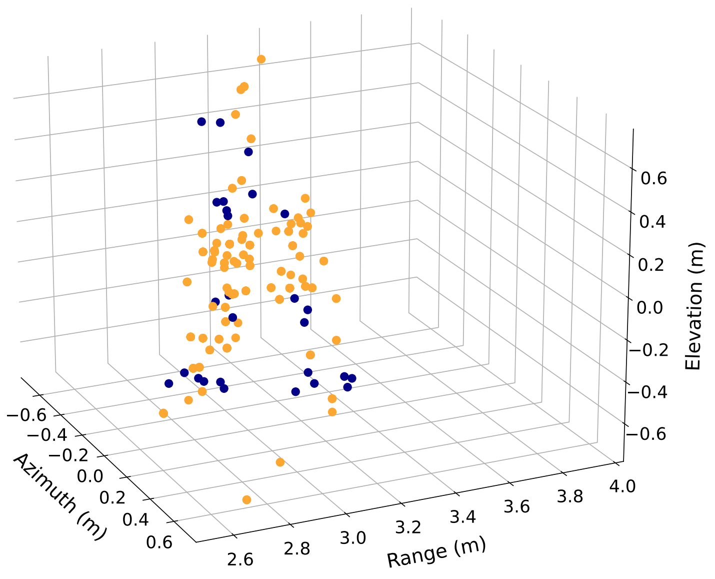

Figure 5.7: Superposition of matched point cloud (orange) and markers (blue)

Table 5.2: Models SA layers parameters

SA layer 1	
Centroids	32
Radius	0.15 m
Points in sphere	8
MLP filters	$[4, 4, 8]$
SA layer 2	
Centroids	16
Radius	0.25 m
Points in sphere	8
MLP filters	$[8, 8, 16]$
SA layer 3	
Centroids	8
Radius	0.4 m
Points in sphere	8
MLP filters	$[16, 16, 32]$

Table 5.3: Models FP layers parameters

Parameter	**Value**
MLP filters FP 1	$[32, 32]$
MLP filters FP 2	$[32, 16]$
MLP filters FP 3	$[16, 16, 16]$

final layers, after the FP layers, include a CNN with 32 filters and kernel size $[1, 5]$ followed by max pooling and a final FC layer with 30×3 units, to compute the 3D coordinates of the predicted marker points. Dropout is applied with probability 0.2.

For ConcatGRU we divide the number of MLP filters in the SA layers in half and we use a GRU layer with 384 units, while for the SequentialGRU model we use a GRU layer with 256 units. While it would be advisable to reduce the number of units in the GRU layers, these should be in the same number as the components of the output vectors returned by the SA layers for the FP layers to work. The use of FC layers to reduce the two dimensions did not provide satisfactory results. In all recurrent models we apply recurrent dropout with probability 0.2 and regularization both to the kernel and the recursion operation with the L_2 norm with parameter 0.01.

5.4.1 Learning Parameters and Dataset

The semi supervised approach introduced in Sec. 4.3.4 improved the speed of convergence, but the computational burden introduced, in the end negated such improvement in time. Considering that the training has been performed on CPU and not on GPU, the results may change if a graphic card were to be used, so we don't discard completely this modification of the architecture for future studies. Regarding the improvement in accuracy, we did not observe any benefit.

We performed the training of our models with the *Adadelta* optimizer, with an initial learning rate of 0.01. Contrary to what is usually done with NN we do not decrease the learning rate during the training procedure, but instead we restore the optimizer to its original state and learning rate to speed up convergence after 300 epochs for two times, and then we let the optimizer adapt the learning rate for the remaining 900 epochs. In total our training requires 1500 epochs to reach convergence.

With the considerations reported in Sec. 5.1.1 we obtain in total 1116 sequences of 30 frames with around 80% of overlapping frames. We divide the dataset in 1000 sequences for training and 116 for test. The test sequences are chosen uniformly from the original set of acquisitions and correspond to complete captures, which implies that they do not share frames with sequences from the training set.

5.5 Networks Results

As described in Sec. 4.3.2, SA layers compute the features also on the set of features associated to its input point cloud. While in Fig. 4.1 we showed that the first SA layer receives in input only the 3D coordinates, it is in fact possible to associate also to the input point cloud some features. This can be for example the relative power of the reflected signal or the Doppler velocity. The latter is particularly interesting as it shows a strong correlation between adjacent points, as seen in Fig. 5.8. In our tests, however, it did not improve the results and was thus not included for the final evaluation.

In the following considerations we define with x the azimuth, with y the depth or range and with z the elevation. The values reported are in absolute form, which

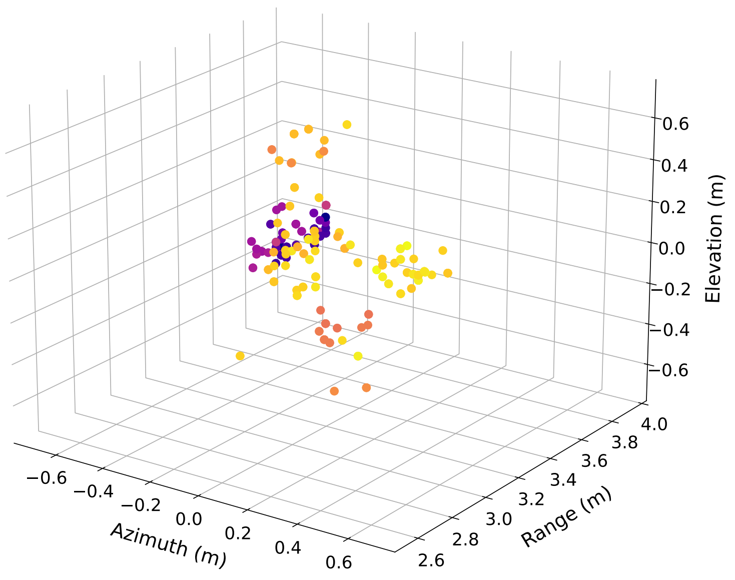

Figure 5.8: Velocity in a radar acquisition

means that they are not normalized, as we have observed that normalization results in worse performance (computed by taking into account the change in scale due to the normalization). Instead our choice of centering each frame on its own centroid provided the best results, even when compared to centering on the centroid of the previous frame. This implied that the preservation of the transition between two consecutive frames does not help with the prediction in this case. This result can be explained if we consider that, contrary to synthetic datasets, our data includes noise. A direct normalization in the $[0, 1]$ interval would in fact result in very different final values between adjacent frames if there is even a single noisy point distant from the main set of points corresponding to the subject. Moreover this type of normalization would make the frames where the subject is too close to the radar, or some parts of the cloud are missing for instance due to the clustering algorithm, simply wrong as they would be considered as complete point clouds and strecthed accordingly, while instead they are only partial. A simple change of reference system to the centroid of each point cloud, instead, preserves the original shape across the same sequence and even between sequences, while avoiding that the network learns an absolute reference system.

Table 5.4: Absolute mean distance (in meters) of the proposed models from the ground truth.

Model	**x**	**y**	**z**
Baseline	0.11	0.21	0.17
ConcatGRU	0.08	0.13	0.09
SequentialGRU	0.08	0.14	0.10

Tab. 5.4 shows that the addition of the RNN provides better results. It also appears that concatenating the features from the SA layers is better than passing a SA layer at a time, even if we consider that we are using a GRU layer with more units and thus less performant for our case. In fact, a further reduction in the number of units would require the removal of a SA layer, while an increase always results in worse performance.

From Tab. 5.4 we also notice that there is a prevalent error in the y axis, the depth for all models. There may be multiple reasons to explain this, but two are probably the most relevant:

- when the subject is too close to the radar, besides the FOV limitations described in Sec. 5.1.1, we also observe that the noise is much stronger and

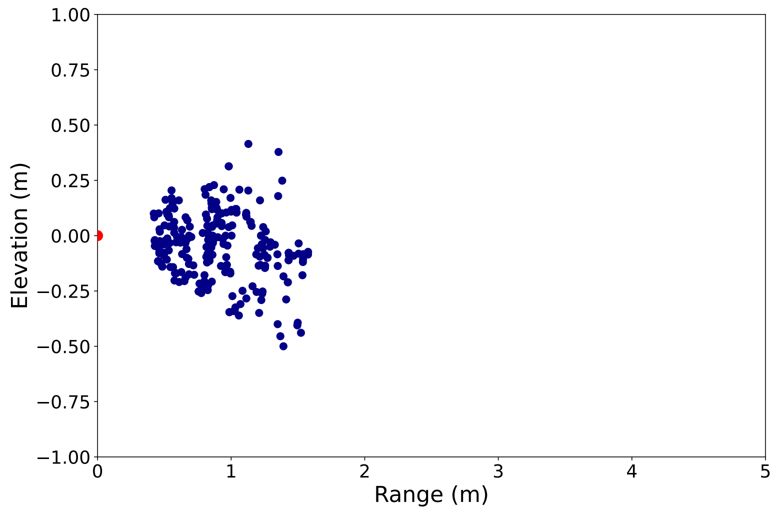

Figure 5.9: Radar acquisition of a subject too close to the radar (red)

more difficult to remove. In particular, our current clustering algorithm is not able to remove it completely. The algorithm to match markers and radar frames still works properly as it considers the whole sequence, but the addition of the noise results in point clouds that are distorted primarily in the y and x axis, as shown in Fig. 5.9. Moreover, the computation of the centroid for the translation of the reference system will also consider the noise as part of the point cloud and thus create a bias mainly in the y axis, while the x axis balances itself;

- our dataset is made up of acquisitions of a subject walking towards and away from the radar in a single direction for a given acquisition. This implies that when the subject changes direction, the position of the markers with respect to the point cloud is reversed both in the x and y axis, further increasing the difficulty of the task, especially with such a small dataset.

Moreover in Tab. 5.5 we can see that the bias is comparable in the different groups of markers, for instance the last 12 markers are all on the feet. Indeed in Fig. 5.10 it appears that the predicted markers assume the correct structure but are simply translated. If we look at Fig. 5.11 the results appear even more convincing. However if we consider a sequence where the subject is in the opposite direction as in Fig. 5.12, we realize that the network predicts the wrong direction and is also not able to recognize and fit the position when the feet are separated. This could be again due to a lack of data, which we plan to deal with in future

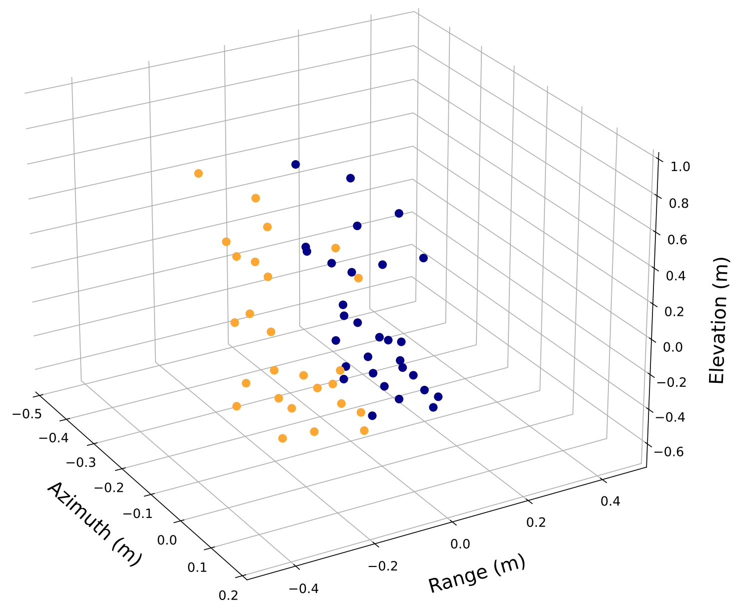

Figure 5.10: 3D visualization of predicted markers (blue) and ground truth (orange) with SequentialGRU. The points are centered on the source point cloud centroid.

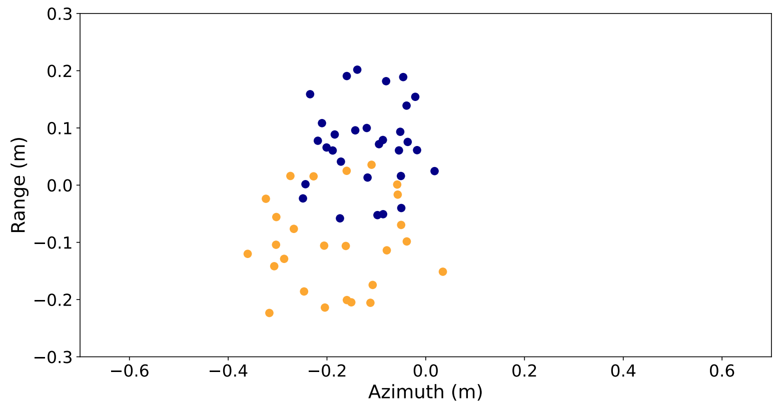

Figure 5.11: 2D visualization of predicted markers (blue) and ground truth (orange) with SequentialGRU. The points are centered on the source point cloud centroid.

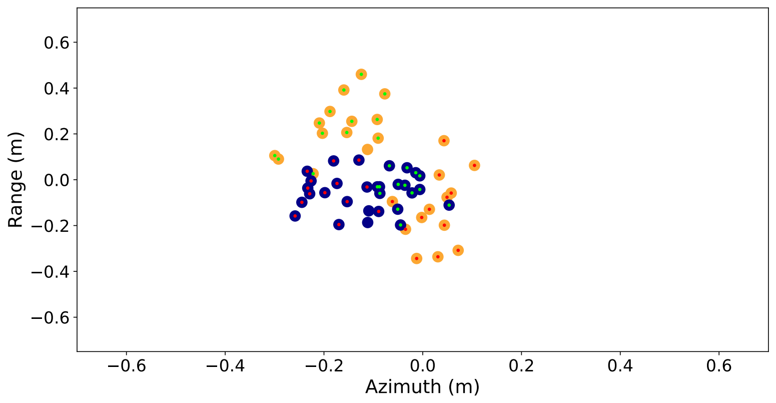

Figure 5.12: 2D visualization of predicted markers (blue) and ground truth (orange) with SequentialGRU with markers on the left (red) and right (green) side highligthed. The points are centered on the source point cloud centroid.

measurement campaigns.

These results show that absolute values like the ones showed in Tab. 5.4 are not sufficiently informative of the accuracy of a model. In particular, this is amplified when the input data is not sufficient in terms of diversification and quantity, which could lead to overfitting.

Table 5.5: Mean distance of ConcatGRU from the ground truth for each marker in meters.

Marker label	x	y	z
C7	0.05344	0.12524	0.08356
LA	0.12136	0.10959	0.07847
RA	0.11081	0.12590	0.08638
L5	0.05355	0.10816	0.06401
RPSIS	0.05833	0.11102	0.06182
LPSIS	0.06827	0.11485	0.06574
RASIS	0.09579	0.10547	0.06941
LASIS	0.09837	0.08712	0.07183
RGT	0.14165	0.09517	0.06663
LGT	0.10096	0.09697	0.08053
RLE	0.10102	0.10882	0.07002
LLE	0.09705	0.11567	0.08316
RME	0.05826	0.11246	0.07320
LME	0.03821	0.12335	0.08367
RHF	0.09902	0.11282	0.09128
LHF	0.09797	0.12057	0.08779
RTT	0.08986	0.13612	0.09867
LTT	0.07036	0.12375	0.11152
RLM	0.08274	0.14580	0.12514
LLM	0.08357	0.14563	0.11948
RMM	0.05930	0.14159	0.11665
LMM	0.06134	0.14485	0.11244
RCA	0.06682	0.15594	0.11476
LCA	0.06862	0.16612	0.12361
RVMH	0.09442	0.15031	0.11491
LVMH	0.09840	0.14838	0.11867
RIMH	0.06341	0.16080	0.12507
LIMH	0.06636	0.16310	0.11216
RIIT	0.07977	0.16437	0.11279
LIIT	0.08024	0.15665	0.11215

6 Conclusion

In this work we have explored the field of radio sensing for motion tracking applications. In particular, we explored the possibility of using radars to substitute the need of markers in specific situations where privacy or mobility may play a major role. We have proposed a pipeline for the acquisition and processing of the data captured by a FMCW radar assisted by a motion capture system. The generated dataset has then been used to train different models of Neural Networks to collect preliminary leads on the efficacy of different architectures.

Our results show that the presence of noise poses a significant problem to the training of a neural network even when trained with multiple data augmentation types. Our current clustering technique in particular was not able to distinguish the noise introduced when the subject was too close to the radar, a situation that should however be avoided during data acquisition for this specific type of objective. At the same time, however, it performed properly at correct distances and it can also be tuned to possibly work better at lower ranges. Our novel algorithm to match the sequences from the radar and the motion capture system, instead, proved to work reliably even in the presence of noise. This allows to more easily set up test environments to capture new data without the necessity of precise synchronization between the radar and the motion capture system.

We have also developed a complete framework for the processing of the data in Tensorflow which includes loading, repartitioning, enhancing and training. We observed that among the transformations of the data that we defined, the subtraction of the centroid of each point cloud as a form of normalization gave the best results. This may be related to the presence of noise and incomplete frames in our dataset, but can still give some insight for future works. Given the reduced amount of data available our proposed models do not perform that well and in particular are easily subject to overfitting. The data augmentation techniques we tested did not improve the situation, which may be interpreted that with RNN a certain amount and diversity of data is still a prerequisite that can not be overlooked. Our results, however, still show that RNN can in fact exploit the temporal correlation in the sequences of data from a radar and improve the prediction, suggesting that this is probably the right way to tackle the problem of skeleton estimation.

6.1 Future Works

This project was a preliminary work toward the design of contactless radar-based motion tracking systems. Here we propose some insights into future developments:

- the high quantity of frames where markers are not present due to the difficulty in tracking some specific markers with the motion capture system could be lowered by interpolating frames. By assuming that the change in position of a marker in two adjacent frames is limited, we can exploit the fact that the motion capture system has a higher frame rate to create hybrid frames that include all the markers in the adjacent frames with their position interpolated;

- the impact of noise and undesired multipath reflections in a real mmWave radar dataset can not be overlooked. Pointnet++ was originally designed with synthetic or LIDAR-like data in mind. For this reason, it employs techniques that do not perform as well with noisy data. For instance it uses FPS that in the presence of noise would almost always put some centroid outside of the set of points corresponding to the subject. In this case, in fact, even random sampling would probably perform better. However we suggest investigating some more advanced algorithm that does not simply rely on the distance, but also on the density of the points in the area. For instance FPS could be simply reused with a distance function that is computed as the distance from the nearest point in the set of centroids, divided by the number of points in a sphere of a given radius around it. This in practice makes use of the BQ algorithm presented in Sec. 4.2.3;

- attention layers have shown great performance in datasets with noisy data, where noise may not only be the addition of unwanted points, but also the lack of parts of the point cloud due to the clustering algorithm, occlusion or other factors. We plan to consider them in a new model architecture;
- our tests showed signs of overfitting with models with higher complexity, for instance with Bidirectional RNN, but these architectures still hold the potential to greatly improve results. We plan on reconsidering them after the collection of a larger dataset;
- given the symmetrical nature of the human body it may be possible to define a loss that does not discriminate between left and right, at least for an initial pre-training of the weights.

References

[1] A. Leardini, Z. Sawacha, G. Paolini, S. Ingrosso, R. Nativo, and M. G. Benedetti, "A new anatomically based protocol for gait analysis in children," Gait & Posture, vol. 26, no. 4, pp. 560–571, 2007.

[2] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," in Proceedings of the 31st International Conference on Neural Information Processing Systems, ser. NIPS'17. Curran Associates Inc., 2017, pp. 5105–5114.

[3] T. Xu, D. An, Y. Jia, and Y. Yue, "A Review: Point Cloud-Based 3D Human Joints Estimation," Sensors, vol. 21, p. 1684, 03 2021.

[4] E. Camuffo, D. Mari, and S. Milani, "Recent advancements in learning algorithms for point clouds: An updated overview," Sensors, vol. 22, no. 4, 2022.

[5] A. Sengupta, F. Jin, R. Zhang, and S. Cao, "mm-Pose: Real-Time Human Skeletal Posture Estimation using mmWave Radars and CNNs," IEEE Sensors Journal, vol. PP, pp. 1–1, 05 2020.

[6] M. Zhao, Y. Tian, H. Zhao, M. A. Alsheikh, T. Li, R. Hristov, Z. Kabelac, D. Katabi, and A. Torralba, "RF-Based 3D Skeletons," in Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, ser. SIGCOMM '18. Association for Computing Machinery, 2018, pp. 267–281.

[7] H. Fan and Y. Yang, "Pointrnn: Point recurrent neural network for moving point cloud processing," ArXiv, vol. abs/1910.08287, 2019.

[8] M. Richards, J. Scheer, J. Scheer, and W. Holm, Principles of Modern Radar: Basic Principles, Volume 1, ser. Electromagnetics and Radar. Institution of Engineering and Technology, 2010.

[9] A. M. Niknejad, "Introduction to Mixers," 2005. [Online]. Available: http://rfic.eecs.berkeley.edu/~niknejad/ee142_fa05lects/pdf/lect15.pdf

[10] B. van Berlo, A. Elkelany, T. Ozcelebi, and N. Meratnia, "Millimeter Wave Sensing: A Review of Application Pipelines and Building Blocks," IEEE Sensors Journal, vol. 21, no. 9, pp. 10 332–10 368, 2021.

[11] W. Wiesbeck, "Radar Systems Engineering," 2009. [Online]. Available: https://www.ihe.kit.edu/download/RSE_script___2009.pdf

[12] S. Rao, "Introduction to mmwave Sensing: FMCW Radars," 2017. [Online]. Available: https://training.ti.com/intro-mmwave-sensing-fmcw-radars-module-1-range-estimation

[13] R. Oshana, DSP Software Development Techniques for Embedded and Real-Time Systems (Embedded Technology). Newnes, 2005.

[14] N. Shahid, T. Rappon, and W. Berta, "Applications of artificial neural networks in health care organizational decision-making: A scoping review," PLOS ONE, vol. 14, p. e0212356, 02 2019.

[15] A. Kamilaris and F. X. Prenafeta-Boldú, "Deep learning in agriculture: A survey," Computers and Electronics in Agriculture, vol. 147, pp. 70–90, 2018.

[16] A. Amato, V. Di Lecce, and V. Piuri, "Neural network based video surveillance system," in CIHSPS 2005. Proceedings of the 2005 IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety, 2005, pp. 85–89.

[17] S. Shalev-Shwartz and S. Ben-David, Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, 2014.

[18] G. V. Cybenko, "Approximation by superpositions of a sigmoidal function," Mathematics of Control, Signals and Systems, vol. 2, pp. 303–314, 1989.

[19] A. Singh, N. Thakur, and A. Sharma, "A review of supervised machine learning algorithms," in 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), 2016, pp. 1310–1315.

[20] M. Z. Alom, T. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. Nasrin, M. Hasan, B. Essen, A. Awwal, and V. Asari, "A State-of-the-Art Survey on Deep Learning Theory and Architectures," Electronics, vol. 8, p. 292, 03 2019.

[21] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248–255.

[22] M. M. Moya and D. R. Hush, "Network constraints and multi-objective optimization for one-class classification," Neural Networks, vol. 9, no. 3, pp. 463–474, 1996.

[23] A. Horé and D. Ziou, "Image Quality Metrics: PSNR vs. SSIM," in 2010 20th International Conference on Pattern Recognition, 2010, pp. 2366–2369.

[24] P. Baldi, "Autoencoders, Unsupervised Learning, and Deep Architectures," in Proceedings of ICML Workshop on Unsupervised and Transfer Learning, ser. Proceedings of Machine Learning Research, vol. 27. PMLR, 02 Jul 2012, pp. 37–49.

[25] A. J. Watson, "Deep Learning Techniques for Super-Resolution in Video Games," ArXiv, vol. abs/2012.09810, 2020.

[26] AMD. (2020) AMD FidelityFX. [Online]. Available: https://www.amd.com/en/technologies/radeon-software-fidelityfx

[27] NVIDIA. (2020, 03) NVIDIA DLSS 2.0: A Big Leap In AI Rendering. [Online]. Available: https://www.nvidia.com/en-gb/geforce/news/nvidia-dlss-2-0-a-big-leap-in-ai-rendering/

[28] O. Chapelle, B. Schölkopf, and A. Zien, Semi-Supervised Learning. The MIT Press, 2006.

[29] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go

with deep neural networks and tree search," Nature, vol. 529, pp. 484–489, 01 2016.

[30] B. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. Sallab, S. Yogamani, and P. Perez, "Deep Reinforcement Learning for Autonomous Driving: A Survey," IEEE Transactions on Intelligent Transportation Systems, vol. PP, pp. 1–18, 02 2021.

[31] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-Image Translation with Conditional Adversarial Networks," in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 5967–5976.

[32] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved Techniques for Training GANs," in Proceedings of the 30th International Conference on Neural Information Processing Systems, ser. NIPS'16. Curran Associates Inc., 2016, pp. 2234–2242.

[33] J. Ho and S. Ermon, "Generative Adversarial Imitation Learning," in Advances in Neural Information Processing Systems, vol. 29. Curran Associates, Inc., 2016.

[34] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," in Advances in Neural Information Processing Systems, vol. 27. Curran Associates, Inc., 2014.

[35] T. Karras, S. Laine, and T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 43, no. 12, pp. 4217–4228, dec 2021.

[36] T. Wu, L. Pan, J. Zhang, T. WANG, Z. Liu, and D. Lin, "Density-aware Chamfer Distance as a Comprehensive Metric for Point Cloud Completion," in In Advances in Neural Information Processing Systems (NeurIPS), 2021.

[37] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," Neural Networks, vol. 2, no. 5, pp. 359–366, 1989.

[38] N. Qian, "On the momentum term in gradient descent learning algorithms," Neural Networks, vol. 12, no. 1, pp. 145–151, 1999.

[39] S. Ruder, "An overview of gradient descent optimization algorithms," ArXiv, vol. abs/1609.04747, 2016.

[40] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," J. Mach. Learn. Res., vol. 12, pp. 2121–2159, jul 2011.

[41] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," ArXiv, vol. abs/1212.5701, 2012.

[42] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," International Conference on Learning Representations, 12 2014.

[43] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. The MIT Press, 2016.

[44] R. Reed and R. J. Marks, Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks. The MIT Press, 02 1999.

[45] C. M. Bishop, Neural Networks for Pattern Recognition. Oxford University Press, Inc., 1995.

[46] Y. Lecun, J. Denker, and S. Solla, "Optimal Brain Damage," in Advances in Neural Information Processing Systems, vol. 2. Morgan-Kaufmann, 01 1989, pp. 598–605.

[47] J. Kukacka, V. Golkov, and D. Cremers, "Regularization for Deep Learning: A Taxonomy," ArXiv, vol. abs/1710.10686, 2017.

[48] A. Krogh and J. A. Hertz, "A Simple Weight Decay Can Improve Generalization," in Proceedings of the 4th International Conference on Neural Information Processing Systems, ser. NIPS'91. Morgan Kaufmann Publishers Inc., 1991, pp. 950–957.

[49] A. Weigend, D. Rumelhart, and B. Huberman, "Generalization by Weight-Elimination with Application to Forecasting," in Advances in Neural

Information Processing Systems, vol. 3. Morgan-Kaufmann, 01 1990, pp. 875–882.

[50] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," Journal of Machine Learning Research, vol. 15, no. 56, pp. 1929–1958, 2014.

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

[52] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Neural Information Processing Systems, vol. 25, 01 2012.

[53] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," Insights into Imaging, vol. 9, pp. 611 – 629, 2018.

[54] Y.-L. Boureau, J. Ponce, and Y. Lecun, "A Theoretical Analysis of Feature Pooling in Visual Recognition," in ICML 2010 - Proceedings, 27th International Conference on Machine Learning, 11 2010, pp. 111–118.

[55] A. Sherstinsky, "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network," Physica D: Nonlinear Phenomena, vol. 404, p. 132306, 03 2020.

[56] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," Bulletin of Mathematical Biophysics, vol. 5, pp. 115–133, 1943.

[57] D. E. Rumelhart and J. L. McClelland, Learning Internal Representations by Error Propagation. The MIT Press, 1987, pp. 318–362.

[58] M. I. Jordan, "Serial Order: A Parallel Distributed Processing Approach," Advances in psychology, vol. 121, pp. 471–495, 1997.

[59] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory," Neural computation, vol. 9, no. 8, pp. 1735–1780, 11 1997.

[60] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, "On the Properties of Neural Machine Translation: Encoder–Decoder Approaches," in Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation. Association for Computational Linguistics, Oct. 2014, pp. 103–111.

[61] G.-B. Zhou, J. Wu, C.-L. Zhang, and Z.-H. Zhou, "Minimal Gated Unit for Recurrent Neural Networks," International Journal of Automation and Computing, vol. 13, pp. 226–234, 03 2016.

[62] K.-H. Chan, W. Ke, and S.-K. Im, "CARU: A Content-Adaptive Recurrent Unit for the Transition of Hidden State in NLP," in Neural Information Processing: 27th International Conference, ICONIP 2020. Springer-Verlag, 11 2020, pp. 693–703.

[63] V. Chen, F. Li, S.-S. Ho, and H. Wechsler, "Micro-doppler effect in radar: phenomenon, model, and simulation study," IEEE Transactions on Aerospace and Electronic Systems, vol. 42, no. 1, pp. 2–21, 2006.

[64] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," Journal of Basic Engineering, vol. 82, no. 1, pp. 35–45, 03 1960.

[65] M. Canil, J. Pegoraro, and M. Rossi, "milliTRACE-IR: Contact Tracing and Temperature Screening via mm-Wave and Infrared Sensing," IEEE Journal of Selected Topics in Signal Processing, pp. 1–1, 2021.

[66] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, ser. KDD'96. AAAI Press, 1996, pp. 226–231.

[67] Z. Meng, S. Fu, S. Yan, H. Liang, A. Zhou, S. Zhu, H. Ma, J. Liu, and N. Yang, "Gait Recognition for Co-Existing Multiple People Using Millimeter Wave Sensing," in AAAI Conference on Artificial Intelligence. AAAI Press, 02 2020.

[68] T. Wagner, R. Feger, and A. Stelzer, "Radar Signal Processing for Jointly Estimating Tracks and Micro-Doppler Signatures," IEEE Access, vol. 5, pp. 1220–1238, 2017.

[69] P. Merriaux, Y. Dupuis, R. Boutteau, P. Vasseur, and X. Savatier, "A Study of Vicon System Positioning Performance," Sensors, vol. 17, no. 7, 2017.

[70] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 77–85.

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

[72] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 06 2015, pp. 1912–1920.