UNIVERSITÀ
DI PAVIA

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRICAL, COMPUTER AND BIOMEDICAL ENGINEERING

MASTER'S THESIS IN COMPUTER ENGINEERING

EDGE AI APPROACHES FOR POSTURAL ASSESSMENT IN HUMAN ACTIVITY RECOGNITION

A.Y. 2025/2026

Candidate:
DAVIDE DE VITTORIO

Supervisor:
PROF. ELISA MARENZI

Co-supervisors:
PROF. GIOVANNI DANESE

ABSTRACT

Continuous monitoring of human posture is essential to protect independence and detect motor decline in frail individuals. However, existing technologies such as cameras and wearable sensors raise issues related to user comfort and privacy.
Conversely, millimeter-wave (mmWave) radar sensors represent a promising alternative, enabling accurate monitoring while preserving privacy and eliminating the need for users to wear or recharge devices. This thesis proposes an innovative approach to postural assessment leveraging mmWave radar technology and the implementation of optimized Artificial Intelligence (AI) models. The objective is to detect human postures to ensure continuous, non-invasive monitoring of vulnerable individuals.

The radar sensor employed in this study is based on a Texas Instruments (TI) chip, while the embedded hardware platform was assembled in collaboration with AZCOM Technology S.r.l. Using this device an experimental campaign was conducted to build a dataset involving 27 participants. Based on the collected data, with a focus on point cloud distribution, various data augmentation and preprocessing techniques were implemented to enhance data quality and the informational content of the point clouds. These data were subsequently fed into different AI models and the robustness of the architectures was validated using Explainable AI (XAI) techniques.

Given that computationally intensive models may prove demanding for resource-constrained embedded systems, a Neural Architecture Search (NAS) was implemented. Through Bayesian optimization, this process identified the optimal balance between classification accuracy, computational complexity, and latency. The best-performing networks were then quantized to facilitate deployment on hardware with limited resources.

In addition, an evaluation of diverse development boards from STMicroelectronics and experimental tests with the best performing models was conducted to identify the best solution in terms of processing speed and energy efficiency. This work enabled to find a system that is optimized from both a software and hardware perspective.

This thesis demonstrates the potential of integrating mmWave radar sensors with AI models to develop highly accurate, transparent, and efficient Human Activity Recognition (HAR) systems.

The thesis is structured as follows. Chapter 1 reviews the state of the art in postural recognition. Chapter 2 presents the sensor used, its operating principle and the hardware solution. Chapter 3 introduces the theoretical fundamentals of AI techniques applied to HAR. Chapter 4 outlines research methodology, data collection, preprocessing techniques and implemented network architectures. Chapter 5 presents the performance evaluation of the adopted models optimized for real-time edge computing and provides a comparative analysis of various boards to identify the most suitable solutions for embedded systems. Finally, chapter 6 draws conclusions and describes ideas for future developments.

TABLE OF CONTENTS

CHAPTER 1 HUMAN POSTURE RECOGNITION 1

CHAPTER 2 RADAR TECHNOLOGY 10

2.1 Working Principles 12

2.1.1 Millimeter Waves Technology 12

2.1.2 Linear Frequency Modulated Signal 12

2.1.3 Multiple Target Detection 16

2.2 MIMO Radar 17

2.2.1 Estimate of Target's Position 18

2.2.2 Virtual Array 21

2.3 Hardware Solutions 22

2.3.1 Sensors 22

2.3.2 Boards 23

2.3.3 Chosen hardware: AZ-SENS-RAD-3T4R-60Ghz 25

CHAPTER 3 ARTIFICIAL INTELLIGENCE FOR POSTURAL ASSESSMENT 28

3.1 Artificial Intelligence Approaches 28

3.1.1 Theoretical Foundations of Recurrent Neural Networks 28

3.1.2 Theoretical Foundations of Convolutional Neural Networks 33

3.1.3 Automation of Network Design: Neural Architecture Search 36

3.1.4 Explainable AI 37

CHAPTER 4 RESEARCH METHODOLOGY 39

4.1 Data Collection 39

4.1.1 Posture Analysis 41

4.1.2 Critical Issues 44

4.1.3 External Dataset 46

4.2 Pre-processing techniques 48

4.2.1 Confidence Ellipsoid 48

4.2.2 Double Ellipsoid Model 51

4.2.3 Data Augmentation 52

4.3 Neural Networks Implementation 55

4.3.1 PointConv 55
4.3.2 PointConv LSTM 58
4.3.3 NAS Optimization 59
4.4 Training and Evaluation Metrics 65
4.4.1 Training 65
4.4.2 Evaluation metrics 66
4.5 Models Adaptation for Hardware Execution 69
4.5.1 Quantization 69
4.5.2 Real Time 72
CHAPTER 5 EXPERIMENTAL EVALUATIONS 77
5.1 Model analysis 77
5.1.1 Networks Performances 78
5.1.2 Post Training Quantization Performances 89
5.1.3 Device Performance 91
5.1.4 Discussion 94
CHAPTER 6 CONCLUSIONS AND FUTURE WORKS 98
LIST OF ABBREVIATIONS 103
GLOSSARY 105
BIBLIOGRAPHY 107

CHAPTER 1
HUMAN POSTURE RECOGNITION

In recent decades society has changed significantly. Continuous advancements in medicine have reduced mortality rates, thus increasing life expectancy. These medical results have given rise to a phenomenon that is complex to manage: the exponential increase in the number of "frail individuals".

In the medical field, the term "frailty" does not simply refer to aging but it describes a condition in which an individual has a reduced physiological reserve, leading to a loss in resilience (homeostasis) [1]. Consequently, she/he is more vulnerable than usual and even minor adverse events can trigger a disproportionate and rapid clinical decline [2]. This condition can be defined as unstable disability. This category is constantly growing and it includes a variety of people: patients undergoing post-traumatic rehabilitation, subjects with neurodegenerative or chronic conditions and those with permanent mobility issues, all of whom could be still able to live independently if properly monitored.

Currently, several technologies exist for this purpose, but they face limitations related to privacy or the high level of user interaction required, which make their use difficult and not very widespread.

The growing number of patients and the technological limitations that prevent continuous home monitoring are placing significant pressure on the healthcare systems of even the most developed nations. Prolonged stays in hospital or inpatient facilities are unsustainable, neither from the economic point of view of national healthcare systems nor from the psychological point of view of patients and their families [3]. This creates an urgent need to transition monitoring to domestic settings, thereby freeing up resources for those who need constant clinical assistance.

The possibility for a frail person to maintain a dignified independence at home is strictly related to the capability of guaranteeing continuous monitoring of daily activities. Being able to assess the posture of a frail individual allows for the diagnosis of problems such as muscle weakness, excessive sedentary habits, anomalous behaviors, or other issues age-related issues.

When monitoring a frail individual, not all movements have the same clinical weight. For instance, the ability to perform basic Activities of Daily Living (ADL), such as getting out of bed, standing up, sitting on a chair or walking, is a primary metric for evaluating a patient's functional independence. Specifically, the sit to stand and stand up from lying transitions are among the most mechanically complex activities. An increase in the time required to complete these movements or a decrease in their fluidity are clear indicators of a declining motor function. Furthermore, monitoring static posture offers insights into patient's general well-being; for example, an increase in daytime lying or sitting may signal physical health issues, but it could also reflect underlying depressive states or cognitive decline [4].

The physical vulnerability of frail individuals is closely linked to psychological consequences, most notably the "Fear of Falling" (FoF). Persistent feelings of instability often trigger anxiety, leading individuals to autonomously restrict their daily activities to essential minimum. This sedentary lifestyle exacerbates cardiovascular issues and muscle atrophy, exponentially increasing the actual risk of falls [5]. To break this vicious cycle, it is essential to restore the patient's confidence. Continuous home monitoring serves this purpose, as it eases anxiety and encourages an active lifestyle by providing the reassurance that any critical event will be promptly detected.

The literature has addressed these problems through two main categories of sensors:

- Wearable devices: smartwatches, smart rings and smart clothes are currently the most widely used solutions for fall detection and vital sign tracking. While studies demonstrate their high accuracy [6], [7] and cost-effectiveness, their reliance on the human factor remains a significant limitation. Frail individuals, especially those who experience cognitive decline or dementia, often forget to wear these devices or intentionally do not use them due to discomfort or due to the reluctance to show visible signs of their vulnerability and loss of independence. Furthermore, wearables require regular battery maintenance and usually need to be removed during activities such as showering, which is one of the situations with the highest risk of falls. Consequently, a monitoring system that is contingent upon the patient's active cooperation cannot ensure reliable, continuous oversight.

- Vision-Based systems: another widely used monitoring system involves the use of environmental sensors such as RGB and infrared cameras. These systems provide constant monitoring without requiring the person to wear anything, and studies in the literature demonstrate the effectiveness of applying AI models to these devices [8], [9]. The main problem here is the lack of perceived privacy. Many individuals strongly oppose the installation of cameras in sensitive areas such as bedrooms and bathrooms, viewing it as a serious violation of privacy. This psychological barrier persists even with advanced privacy-protecting cameras that perform immediate skeletal joint extraction locally without storing raw video streams; users often still feel as though they are being watched and harbor a deep distrust of the technology. Furthermore, although the use of infrared cameras can mitigate the severe drop in performance that standard RGB sensors experience in low-light conditions [10], they are still optical lenses and therefore fail to resolve the underlying issue of the perception of an invasion of privacy.

The limitations of both these technologies have shifted research focus toward non-intrusive alternatives that protect privacy. Thus, Frequency Modulated Continuous

Wave (FMCW) radars have emerged as a leading solution [11]. These sensors operate effectively regardless of lighting conditions and, most importantly, ensure privacy by mapping the environment through radio waves, representing detected subjects as an anonymous, sparse point clouds. However, this approach presents a technical challenge to face. The raw data generated by radar sensors is multidimensional, noisy and computationally intensive. Transmitting continuous point cloud streams to remote cloud servers for AI processing raises critical issues regarding network latency, bandwidth consumption and data security. To enable resilient real-time monitoring, AI must be moved directly where the sensor itself is located, a paradigm known as Edge AI [12]. Implementing deep learning models on microcontrollers with severely limited memory and power resources represents one of the most pressing challenges in modern engineering.

The use of FMCW radar for postural recognition has generated a vast body of literature. Over the years, researchers have focused on how to process the large amount of raw data reflected by the human body and feed it into neural networks. The current state of the art is divided into three types of data representation: Time Frequency maps, Range-Doppler maps and Point Clouds.

The first research branch explored involves transforming the radar signal into a two-dimensional image, also known as Time Frequency map or Micro-Doppler spectrogram, which shows the temporal evolution of the Doppler associated with the various reflections generated by the detected person.
A study conducted by Kim Y. and Ling H. [13] used a radar sensor to extract Micro-Doppler signatures from seven different human activities. Through the use of Support Vector Machines (SVM), they demonstrated that similar dynamic activities, such as walking and running, could be accurately distinguished based on the signatures of micro-movements in the limbs. In this approach, extraction was performed manually by the authors, whereas Jokanovic et al. [14] proposed the use of Deep Autoencoders (DAE) applied directly to radar spectrograms for fall detection. Their study demonstrated that deep neural networks could autonomously learn the most

important features from the image. The risk of this approach, however, is that the model requires higher computational and memory costs, which could be incompatible with embedded solutions.

The limitations highlighted by Luo F. et al. [15] concern the spectrogram approach for practical AAL applications regarding two main aspects. On the one hand, there is an efficiency limitation: generating and processing high-resolution images (e.g., 256 x 256 pixels) require heavy neural networks and millions of floating-point operations (FLOPs), making it incompatible with the memory and battery constraints of Edge AI devices. On the other hand, there is an informational limitation: the Micro-Doppler spectrogram focuses analysis exclusively on the time-frequency domain, losing information regarding the subject's spatial position. This "spatial blindness" impedes the accurate distinction between a fall from other daily movements that exhibit similar Doppler signatures but different spatial trajectories.

To address the loss of spatial information associated with the use of Micro-Doppler maps, a parallel line of research has explored the use of multidimensional maps, also known as Range-Doppler maps. These are two-dimensional matrices generated by processing the radar signal, where one axis represents the target's distance from the sensor (range) and the other is its radial velocity. Hsu et al. [16] developed a fall detection system utilizing these maps to preserve target distance information during the action. The true innovation of this study lies in mitigating the limitations of heavy 3D-CNN architectures, which are often too computationally intensive for local processing; instead, they introduced the use of Bidirectional Long Short-Term Memory (Bi-LSTM) recurrent networks. By feeding the network with sequences of Range-Doppler maps, the model analyzes the temporal dynamics of a fall in both directions (past and future). As demonstrated by their experiments, this approach achieves high accuracy in distinguishing a real fall from everyday actions with similar velocity profiles, such as sitting down quickly, drastically reducing the false positive rate thanks to the integration of distance data.

A more advanced approach is proposed by Khalid et al. [17], who introduced a Multi-View CNN-LSTM architecture for human activity recognition. This system decomposes the 3D radar data cube (time-range-Doppler) into three distinct two-dimensional projections: Time-Range, Time-Doppler (the classic Micro-Doppler spectrogram), and Range-Doppler. This multi-view strategy enables the simultaneous capture of the target's temporal dynamics, velocity, and spatial position. The architecture employs CNNs for spatial feature extraction, followed by LSTM layers that share weights across the different projections to jointly learn temporal correlations.

Because of the memory overhead, the academic community has begun to explore a different approach discarding image-based methodologies, which are computationally intensive to implement, in favor of sparse spatial representations known as point clouds. This transition provides significant advantages for systems with limited computational resources. While Range-Doppler maps require the processing of matrices composed of thousands of pixels, a point cloud is composed only by those reflections that exceed thresholds defined by specialized algorithms such as the Constant False Alarm Rate (CFAR). In this way, an individual can be accurately represented by a compact tensor consisting of a few dozen or a few hundred points per frame, reducing the memory footprint of the networks required to process this data by several orders of magnitude.
However, unlike map-based approaches, point clouds do not possess a fixed data size. This variability precludes the direct application of traditional Convolutional Neural Networks (CNNs). An initial attempt was made in this direction. Singh et al. [18], voxelized the 3D space, that is, divided it into cubes to create a dense matrix, with a value assigned to each cube representing the number of points contained within it. This solution clearly remains computationally inefficient in this case as well, due to the continued use of images.

To overcome the inefficiency of voxelization, researchers tried to process the point cloud directly. A breakthrough was achieved by Charles et al. [19] with the

introduction of PointNet and its derivative models, which avoid 3D grid conversion by processing point cloud to extract feature vector. Specifically, these networks make use of Multi-Layer Perceptrons (MLPs) to extract spatial features, followed by aggregation functions such as max pooling to generate a global feature vector. These networks are also invariant to permutations, meaning that regardless of the order in which the points are arranged, the output remains unchanged.

Following the work by Charles et al., several studies have successfully adapted PointNet architecture to process sparse point clouds. A particularly significant contribution in this area is the study made by Dang et al [20]. In this work, researchers collected point clouds generated by human movements and then used the PointNet++ architecture, which demonstrated an extraordinary ability to extract spatial features from the points in the clouds, achieving an accuracy of 94% in walking recognition and 93% in fall detection.

However, because human movement is dynamic, evaluating a single static frame is often insufficient to distinguish complex activities, such as transitioning from sitting to standing or differentiating an accidental fall from lying down voluntarily. To address this, the academic community has begun integrating spatial extractors with sequential layers to capture temporal evolution. A typical architectural pattern involves using lightweight variants of PointNet as spatial feature extractor for individual frames and then feeding these vectors into Recurrent Neural Networks (RNNs). The crucial role of these temporal layers has been highlighted, for instance, in the work conducted by De Vittorio et al. [21]. Here, it has been emphasized that integrating LSTM layers allows the network to keep long-term dependencies between temporal data, improving the system's ability to interpret complex posture variation and predict falls based on previous temporal instants.

To summarize, the analysis of the current state of the art reveals a clear evolution within the field of AAL. Traditional technologies such as wearable sensors and vision-based systems, while effective, have limitations regarding user comfort and privacy. Consequently, FMCW radars appear to be a promising and non-invasive

alternative. Concerning radar signal processing, the academic community has demonstrated that image-based representations such as Micro-Doppler maps and Range-Doppler maps are computationally more demanding than representations using 3D point clouds. These findings are particularly significant for Edge AI applications, where minimizing memory footprint and latency is critical.

The challenge of finding an optimal balance between classification accuracy and computational complexity remains open. This master's thesis aims precisely to find a solution to this challenge. A novel spatial representation has been studied and implemented to capture complex postural transitions. Data augmentation techniques have been employed to address the issue of point cloud data sparsity, and Explainable AI techniques have been applied to validate the performance of these models and the effectiveness of the features used. To ensure optimal deployment on embedded devices, NAS was implemented to identify the most efficient neural networks. After that, quantization was applied to these networks to exploit two main purposes: first, to reduce and optimize resource utilization in embedded system; second, to facilitate the evaluation of AI approaches across diverse hardware configurations, thereby identifying optimal deployment strategies. The following chapter will provide a theoretical foundation for the technologies used, with a particular focus on the operation of MIMO FMCW radar sensor and the generation of point clouds.

CHAPTER 2
RADAR TECHNOLOGY

Radar, an acronym for Radio Detection and Ranging, is a technology used to perform detection in various fields. It relies on the use of electromagnetic waves to detect a series of parameters from targets within its field of view. Radar emits electromagnetic waves through a transmitting antenna, with frequencies ranging from 3 MHz up to 300 GHz, which travel through space; when they encounter an obstacle, part of the energy is reflected. Its amount depends on various factors, such as the surface and size of the object. The reflected signal is captured by a receiving antenna and processed to extract the relevant information.

There are several radar architectures and signal modulation schemes, each suited for specific applications based on their operational principles:

- **Continuous Wave (CW):** basic CW radars transmit a continuous sinusoidal signal at a constant and unmodulated frequency. They represent a highly effective tool for measuring the radial velocity of moving targets through the Doppler effect. However, since the transmitted signal lacks timing or frequency variation, standard CW radars cannot directly estimate the target range;

- **Frequency Modulated Continuous Wave (FMCW):** these radars overcome the limitations of standard CW systems by continuously varying the frequency of the transmitted signal over time, typically using a linear modulation known as a chirp. The received echo is mixed with the transmitted signal to generate an Intermediate Frequency (IF) signal, whose frequency is proportional to the target distance. By processing the IF signal across multiple

chirps, the radar can simultaneously estimate both range and radial velocity. Due to their high accuracy, compactness, and low power consumption, FMCW radars are widely adopted in indoor human monitoring applications;

- **Frequency Shift Keying (FSK):** such radars alternate the transmitted signal between two or more discrete frequencies. By analyzing the phase difference of the received echoes corresponding to the different transmitted frequencies, the system can estimate both target range and velocity. Compared to FMCW radars, FSK systems generally require lower computational complexity but provide also a lower range resolution [22].

In addition to waveform modulation techniques, radar systems can also be classified according to their antenna configuration:

- **Single Input Multiple Output (SIMO):** these radars employ a single transmitting antenna and multiple receiving antennas. The spatial separation among the receivers causes phase differences in the received echoes, enabling the estimation of the angle of arrival and therefore the spatial localization of the target, typically in a 2D Range-Azimuth representation;

- **Multiple Input Multiple Output (MIMO):** such radars use multiple transmitting and multiple receiving antennas. By transmitting time-separated signals, they create a virtual antenna array whose size is equal to the product of the number of transmitters and receivers. This significantly improves angular resolution and enables more accurate 2D or 3D target localization allowing the generation of dense spatial point clouds [23].

The sensor employed in this thesis is a FMCW MIMO system. Its fundamental operating principles will be discussed in detail in the following pages.

2.1 **Working Principles**

2.1.1 ***Millimeter Waves Technology***

Millimeter waves are electromagnetic waves with wavelengths (λ) ranging from one to ten millimeters. This short wavelength offers significant technical advantages. First, mmWave technology allows for a significant reduction in hardware size, and since components progressively diminish their dimension proportionally to the wavelength, it is possible to integrate complex multi-antenna systems directly into a single compact, low-power integrated circuit. Second, operating at the millimeter scale ensures greater sensitivity to detecting micro-movements. In fact, in radar, the measurement of small displacements is based on the phase shift $\Delta\phi$ of the reflected signal. This phase shift is related to the physical displacement Δx by the equation

$$\Delta\phi = \frac{4\pi\Delta x}{\lambda} \qquad (1)$$

A typical value for the wavelength λ, for example when operating at 80 GHz, is 3.75 mm. Therefore, millimetric movement causes a macroscopic and clearly detectable phase shift [24].

2.1.2 ***Linear Frequency Modulated Signal***

Looking at the technical details, the signal emitted by an FMCW radar sensor consists of a linear frequency-modulated (LFM) pulse, commonly referred to as a chirp. In a chirp signal the frequency is not constant but increases linearly over time:

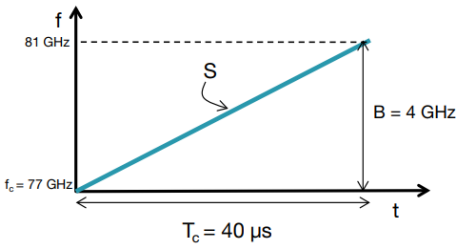

Figure 2.1. Example of Chirp signal [24]

The signal is characterized by three parameters:

- initial frequency f_c,
- bandwidth B,
- chirp duration T_c.

The frequency variation index is defined as the slope (S), calculated as $S = \frac{B}{T_c}$ and represented graphically in Figure 2.1. The steeper the slope, the greater the rate of frequency variation.

As illustrated in the block diagram in Figure 2.2, at the architectural level, several components are required to generate, transmit, and receive this waveform:

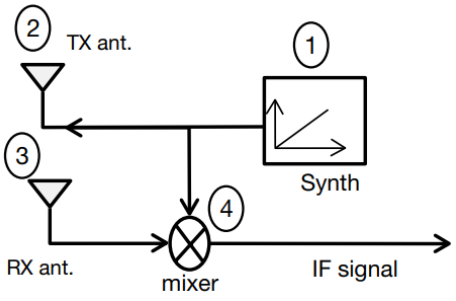

Figure 2.2. Radar block diagram [24]

The heart of the system is the synthesizer (synth), which is responsible for physically generating the chirp. This wave is then radiated into the surrounding space via the transmitting antenna (TX ant). When the electromagnetic wave encounters an object,

part of it is reflected toward the sensor. The echo, which is a time-delayed version of the transmitted pulse, is captured by the receiving antenna (RX ant) and directed to the mixer, an electronic component capable of combining the reflected and received signals through multiplication.

Assuming that, at a given instant, the two signals can be approximated as sinusoidal functions:

$$x_1 = \sin(\omega_1 t + \phi_1) \quad (2)$$

$$x_2 = \sin(\omega_2 t + \phi_2) \quad (3)$$

The mixer combines these inputs to produce an output that remains sinusoidal, but with a phase equal to the difference between those of the two input signals:

$$x_{out} = \sin[(\omega_1 - \omega_2)t + (\phi_1 - \phi_2)] \quad (4)$$

Figure 2.3 shows the transmitted and the received chirp providing a clearer understanding of the situation:

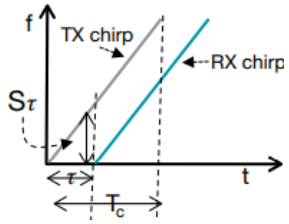

Figure 2.3. Time difference between the transmitted and received chirp signals [24]

The received signal is very similar to the transmitted one, with the difference that it is delayed in time by a value of:

$$\tau = \frac{2d}{c} \quad (5)$$

Where d is the distance of the object from the sensor and c is the speed of light.

By subtracting them, a signal with a constant frequency is obtained at the mixer's output, known as the IF signal. This persists for the duration of the overlap between the transmitted and received chirps:

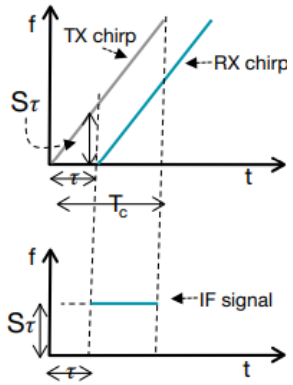

Figure 2.4. IF signal [24]

Considering an ideal scenario involving a single static target, the time-domain function describing this static signal will be a pure sinusoidal wave whose phase is equal to the phase difference between the transmitted and received chirp signals at the moment in which they begin to overlap. This phase can be mathematically described as:

$$\phi_0 = 2\pi f_c \tau \qquad (6)$$

Since the time delay τ is directly proportional closely related to the distance d between the sensor and the detected object, by analyzing the phase and frequency of this sinusoidal wave it is possible to calculate the distance to the target [24].

2.1.3 *Multiple Target Detection*

When it is necessary to detect multiple targets at different distances, it must be considered that each target generates an echo that reaches the receiving antenna with a different delay τ, which is proportional to its distance from the sensor. The mixer must therefore process multiple signals simultaneously. As a result, the IF signal will no longer be a single sinusoidal wave, but a superposition of different frequency components.

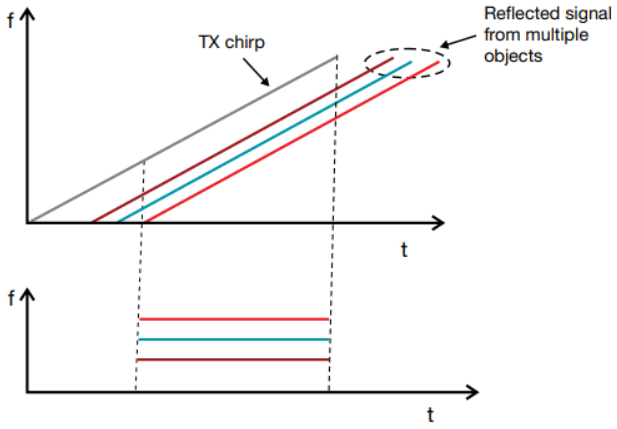

Figure 2.5. IF signals with multiple targets [24]

Although each of these components maintains a constant frequency, they must be separated to extract their spatial information. This is achieved by processing the IF signal using the Fast Fourier Transform (FFT). The result of this operation is a frequency spectrum in which the signal's energy is concentrated in discrete, separate peaks, and each peak indicates the presence of an object at a specific distance.

One effective method to improve range resolution, beyond increasing the number of antennas, is to extend the observation period (i.e. the duration of the chirp signal). When using a short observation window, two closely spaced targets produce sinusoids with negligible frequency differences. After applying the FFT, these peaks

merge into a single broad peak, leading the system to detect only one target (instead of two):

Figure 2.6. Detection of two nearby objects using a short observation period [24]

To overcome this limitation and successfully identify the two targets, it is helpful to extend the observation window, which results in the objects moving further away from the sensor. In this way, the phase difference between the two sinusoidal waves will become more pronounced, and consequently, using the Fourier transform, it will be possible to clearly identify the two frequency peaks [24].

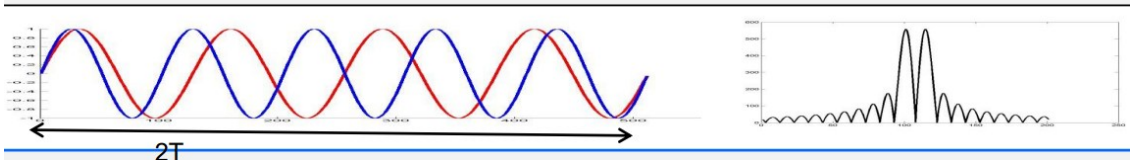

Figure 2.7. Detection of two nearby objects using a longer observation period [24]

2.2 MIMO Radar

SIMO radars employ a single transmitter and multiple receiving antennas. As the number of receiving antennas increases, so does the resolution angle; however, each additional receiver requires a dedicated signal processing chain, which significantly increases hardware cost and power consumption.

To address this, MIMO (Multiple-Input Multiple-Output) radar technology was developed. By utilizing multiple antennas for both reception (N_{RX}) and transmission (N_{TX}), it is possible to create a virtual antenna array achieving the same resolution angle as would be obtained with $N_{TX} * N_{RX}$ receiving antennas in a SIMO radar, with

the advantage that in this case it does only require N_{RX} processing chains. This concept will be explained in more detail in the following paragraphs.

2.2.1 *Estimate of Target's Position*

To determine the target's spatial position, it is not enough to measure the radial distance, but it's also necessary to estimate the angle of arrival. For simplicity, it is possible to consider a system consisting of a single transmit antenna (T_X) and two receive antennas R_{X1}, R_{X2} separated by a distance d. The signal emitted by T_X and reflected by the target, which is located at an angle θ relative to the sensor, will not reach the two receivers at the same time; instead, the electromagnetic wave will first reach antenna R_{X1} at the origin and then R_{X2}.

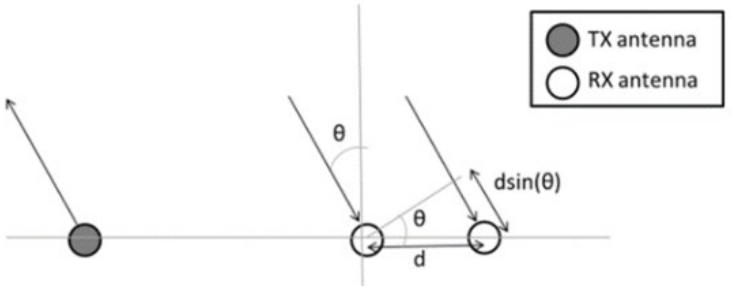

Figure 2.8. SIMO Radar with one transmitting and two receiving antennas [25]

This geometric difference results in a phase difference ω. In fact, the signal must travel an additional distance d, which results in a phase shift that can be mathematically expressed as:

$$\omega = \frac{2\pi}{\lambda} dsin(\theta) \qquad (7)$$

By reversing equation 7, it is possible to estimate the target's angle of arrival:

$$\theta = \sin^{-1}(\frac{\omega\lambda}{2\pi d}) \quad (8)$$

The phase difference between two signals is mathematically confined to the interval $[-\pi; +\pi]$. By substituting these boundary values into the previous equation, it is possible to derive the expression for the radar's field of view (FOV), which is the maximum angular range within which the sensor can detect targets:

$$\theta = \pm\sin^{-1}(\frac{\lambda}{2d}) \quad (9)$$

To maximize the FOV, the distance between the antennas is typically set during the design phase equal to half the wavelength ($d = \lambda/2$). In this configuration, the argument of the arcsine function becomes equal to ± 1, ensuring an optimal theoretical FOV of $\pm 90°$, which allows the radar to capture everything that moves in front of it.

A two-antenna radar therefore allows for the estimation of the angle of arrival, but using only two receiving elements is not sufficient to guarantee angular resolution that can be considered precise. By increasing the number of receiving antennas, for example by switching to an array of four antennas spaced d apart from one another, the reflected signal will have a phase shift that varies linearly along the array.

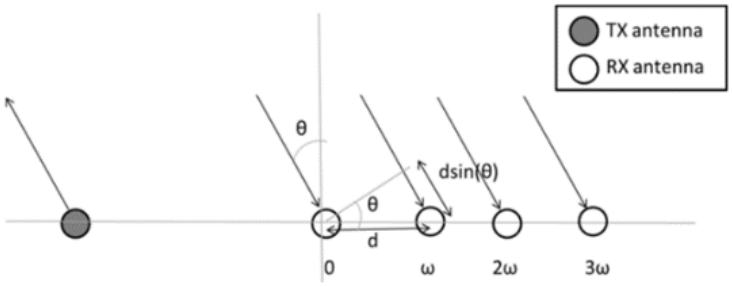

Figure 2.9. SIMO Radar with one transmitting and four receiving antennas [25]

Considering the first antenna as the zero-phase reference, the subsequent receivers will receive signals with a phase shift equal to $\omega, 2\omega, 3\omega$. By sampling the signal through the N antennas and processing it using a Fast Fourier Transform (FFT), it is possible to estimate the angle θ more accurately. The radar's ability to distinguish between two objects that are close together depends heavily on the number of antennas. In Figure 2.10 this concept is clarified:

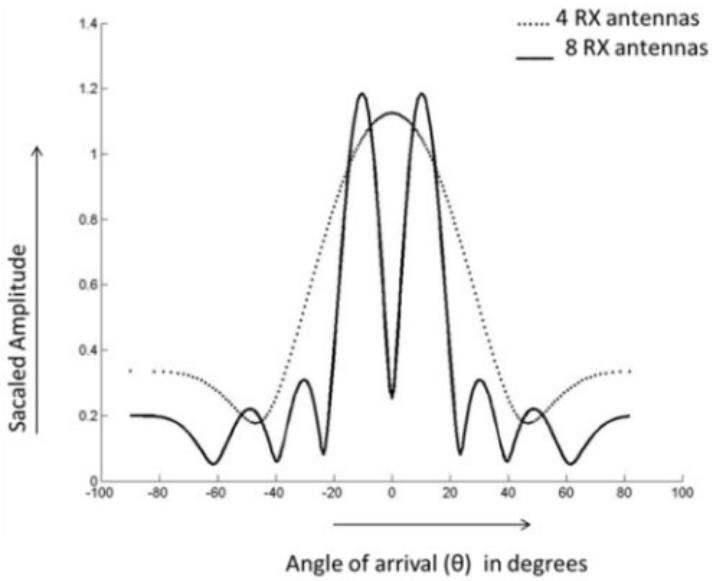

Figure 2.10. Ability to distinguish between two nearby targets with four and eight receiving antennas [25]

An analysis of the spectrum generated by two targets located at $\theta = -10°$ and $\theta = +10°$ reveals a clear difference in performance between four antennas and eight antennas array. In the case of four receiving antennas, the presence of two targets is not distinguishable, and they are merged into a single central peak, causing the systems to erroneously detect a single object located directly in front of the sensor at $\theta = 0°$. Conversely the use of eight antennas significantly narrows the width of the main peak. This configuration ensures the detection of the two different targets as separate entities in space [25].

2.2.2 *Virtual Array*

In MIMO radars, angular resolution is scaled by increasing both the number of transmitting and receiving antennas. A MIMO configuration with two transmitters and four receivers produces the same angular resolution as a SIMO radar with eight receiving antennas.

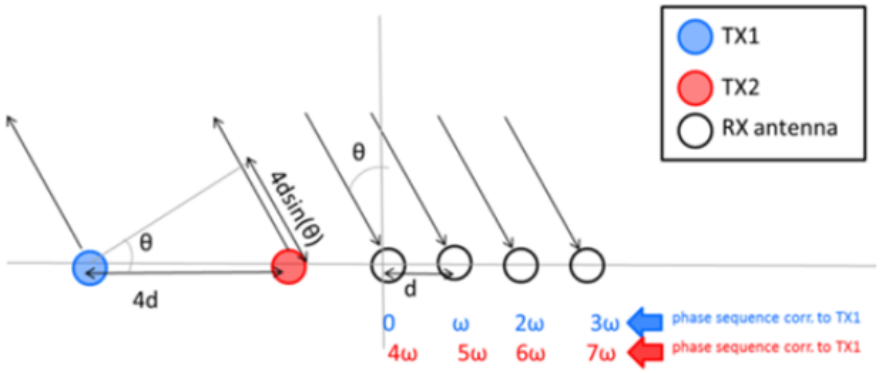

Figure 2.11. MIMO Radar with two transmitting and four receiving antennas [25]

The signal emitted by antenna TX1 will be received by the four antennas with a phase shift of 0 for the first, ω for the second, 2ω for the third, and 3ω for the fourth. Since antenna TX2 is positioned at a distance of $4d$ from antenna TX1, the signal emitted by antenna TX2 will travel a path $4dsin(\theta)$ longer than the signal emitted by TX1, and consequently the signal received by each antenna will be received with an additional phase shift of 4ω. The phase sequence of the signal emitted by TX2 as perceived by the four antennas will be $(4\omega, 5\omega, 6\omega, 7\omega)$, and by concatenating the two sequences, the following array is obtained: $(0, \omega, 2\omega, 3\omega, 4\omega, 5\omega, 6\omega, 7\omega)$, which is the same sequence that would have been obtained with eight receiving antennas and one transmitting antenna.

It can therefore be stated that, in a MIMO radar, the combination of two transmit antennas and four receive antennas generates a so-called "virtual array" equivalent to eight receive antennas in a SIMO radar. Generalizing this principle, a MIMO system

equipped with N_{TX} transmit elements and N_{RX} receive elements is capable of generating a virtual array with dimensions equal to $N_{TX} * N_{RX}$ antennas. This approach allows for increased angular resolution while maintaining a compact hardware footprint [25].

2.3 **Hardware Solutions**

The selection of the most appropriate hardware is a foundational step toward developing an efficient and reliable solution.

2.3.1 ***Sensors***

Different options have been evaluated before the final device was chosen. Table 2.1 shows the alternative evaluated sensors and their principal specifications.

Sensor	Frequency (GHz)	Tx Antennas	Rx Antennas
IWR1443 [26]	76-81	3	4
IWR1843 [27]	77-81	3	4
Vayyar IMAGEVK-74 [28]	62-69	20	20
BGT60TR13C [29]	58-63.5	1	3

Table 2.1. Alternative sensors evaluated

Among the options from Texas Instruments, both the IWR1443 and IWR1843 were considered. IWR1443 integrates a hardware accelerator instead of a programmable Digital Signal Processor (DSP), reducing power consumption, but this limits implementation flexibility on the chip itself. Furthermore, IWR1843 and IWR1443 sensors operate in the 77–81 GHz band and are therefore subject to the regulatory limitations typical of the automotive market.

Another alternative is the Vayyar IMAGEVK-74. With its array of 20 transmitting antennas and 20 receiving antennas, it produces significantly denser point clouds than TI sensors. However, the large amount of data generated requires significant computing power, which increases costs, device size, and energy consumption, making it less suitable for an embedded solution.

Finally, another option is miniaturized radar sensors such as the Infineon BGT60TR13C, which are designed for low power consumption and are therefore ideal for embedded solutions. However, these sensors are optimized for short-range, ultra-high-resolution detection, such as millimeter-scale gestures, rather than posture monitoring.

2.3.2 *Boards*

A radar sensor alone often lacks the computational capacity to handle complex neural networks without interrupting or slowing down critical data acquisition tasks. To mitigate this risk, an external processing board is required to manage AI-based inference.

Several development boards have been evaluated. Specifically, the research involved testing the Microprocessor Units (MPUs) supplied by STMicroelectronics, using the ST Edge AI Developer Cloud platform for virtual benchmarking. This platform accelerates development and reduces costs by eliminating the need for physical hardware. Here it is possible to upload, run, and test AI models to optimize their performance and evaluate the boards' power consumption before actually deploying them on physical devices.

Table 2.2 shows the STMicroelectronics evaluated boards and their principal specifications.

Board	CPU	Frequency (MHz)	RAM
STM32MP257F-EV1 [30]	Dual-Core Arm Cortex-A35	1500	4 GB
STM32MP157F-DK2 [31]	Dual-Core Arm Cortex-A7	800	512 MB
STM32MP135F-DK [32]	Single-Core Arm Cortex-A7	1000	512 MB

Table 2.2. STMicroelectronics evaluated boards

The first development board evaluated is the STM32MP257F-EV1. It features a 64-bit architecture with a dual-core Arm Cortex-A35 CPU, paired with a Cortex-M33. In addition, it also features a dedicated Neural Processing Unit (NPU), a hardware accelerator designed to perform matrix multiplications and other complex operations natively. This allows neural networks workloads to be executed with extremely low latency without straining the main CPU.

The second board considered is the STM32MP157F-DK2. It combines a dual-core Arm Cortex-A7 processor with an Arm Cortex-M4 coprocessor, allowing for task separation: the A7 cores can handle the high-level operating system and data routing, while the Cortex-M4 manages sensor interfacing. Overall, it offers considerable computing power but still relies on the CPU for neural network inference since it lacks a dedicated NPU.

Finally, the last board evaluated is the STM32MP135F-DK. This device is a basic, cost-effective MPU built with a single Arm Cortex-A7 core. It does not have a dedicated NPU.

The focus on these specific devices stems from a limitation regarding the Microcontroller Units (MCUs) provided by STMicroelectronics. As highlighted in Section 5.1.4, although MCUs are efficient components, they lack native support for

the complex management of recurrent layers. This technical limitation made it necessary to conduct testing only on the MPUs provided via the cloud.

2.3.3 *Chosen hardware: AZ-SENS-RAD-3T4R-60Ghz*

Based on these considerations, the final device that was chosen to be adopted in this thesis work is the AZ-SENS-RAD-3T4R-60Ghz. It is composed of two main elements.

The first one is the AZ-RPI-RAD sensor board based on the mmWave TI SOC IWRL6843 [33] radar. It resolves the limitations of the previously analyzed alternatives.

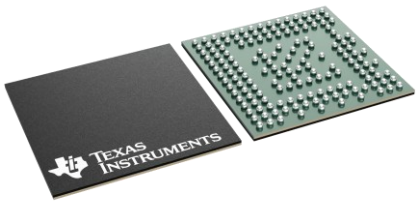

Figure 2.12. Sensor TI SOC IWRL6843

The sensor operates in the 60–64 GHz frequency band and this complies with indoor application regulations. By leveraging a MIMO configuration with three transmitting and four receiving antennas, it creates a 12-antenna virtual array. This allows for a detailed three-dimensional analysis of the environment, guaranteeing strong spatial resolution over an azimuth and elevation Field of View (FoV) of 120°, with a continuous detection rate of 10 frames per second. Furthermore, the hardware includes a DSP for highly efficient signal processing.

The second component is the Raspberry Pi 4 [34]. While the STMicroelectronics MPUs represent the most suitable solutions for the final, low-power edge deployment, the physical setup selected for the experimental development and data

collection phase of this thesis relies on the Raspberry Pi 4. In fact, it is an ideal prototyping tool that is easily accessible during the early stages of development, even though it is energy inefficient.

Figure 2.13. Raspberry Pi 4

It is equipped with a 64-bit quad-core processor and it guarantees hardware integration with the radar, providing the robust computing power and networking interfaces required for rapid prototyping, real-time data streaming and preliminary algorithm validation.

CHAPTER 3
ARTIFICIAL INTELLIGENCE FOR POSTURAL ASSESSMENT

3.1 **Artificial Intelligence Approaches**

3.1.1 ***Theoretical Foundations of Recurrent Neural Networks***

For accurate and precise postural analysis using radar sensors, the temporal dimension is essential. In fact, a single frame provides a static description of posture, but this is not sufficient to distinguish between classes that exhibit strong temporal dependence. It is therefore necessary to use models equipped with internal memory capable of retaining the sequence history.

Unlike traditional feedforward networks, RNNs incorporate a feedback mechanism that allows past information, known as the hidden state, to be propagated across the time steps of the sequence.
At each time step t, the RNN takes an input vector, x_t, and updates its hidden state, h_t, using equation 10:

$$h_t = \sigma(W_x x_t + U_h h_{t-1} + b_h) \text{ [35]} \qquad (10)$$

where:

- W_x is the weight matrix between the input and hidden layer,
- U_h is the weight matrix for the recurrent connection,
- b_h is the bias vector,

- σ is the activation function, that typically is the hyperbolic tangent function (tanh).

The output at each time step, t, is given by equation 11:

$$y_t = \sigma_y(W_y h_t + b_y) \qquad (11)$$

where:

- W_y is the weight matrix between the hidden and output layers,
- b_y is the bias vector,
- σ is the activation function for the output layer.

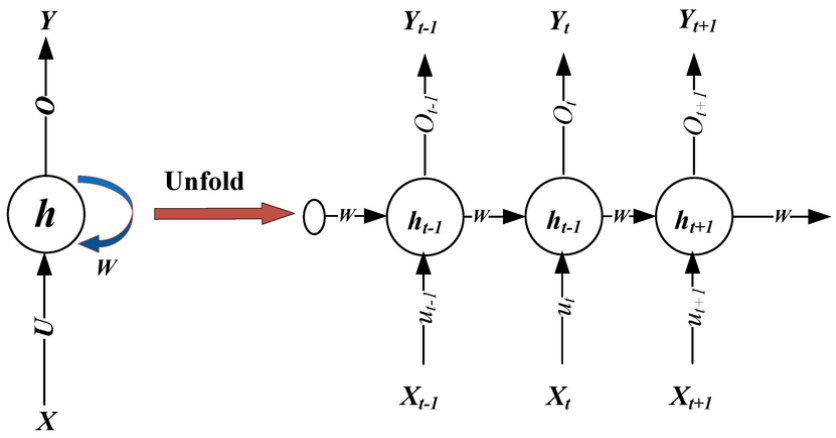

Figure 3.1. Workflow graph of RNN [36]

Long Short-Term Memory (LSTM)

Basic RNNs suffer from limited long-term memory capacity. Since the hidden state is updated at each step, new information tends to overwrite past information causing the loss of initial data. This phenomenon, linked to the vanishing gradient problem, prevents the model from learning relationships between distant elements in the sequence and compromises its effectiveness in processing complex data structures.

LSTM is an evolution of the basic RNN model designed to address the problem of vanishing or exploding gradients. LSTM introduces a new mechanism to carry information through the computation, that is, a cell state vector c_t that encodes the memory of past computations. The gradient can thus propagate backward without significant attenuation. The information in the module is managed by three gates:

1. Forget Gate (f_t): the first gate selects how much of the previous memory c_{t-1} should be retained and how much of the memory can be forgotten:

$$f_t = \sigma(\mathrm{W_f} x_t + U_f h_{t-1} + b_f) \tag{12}$$

2. Input Gate (i_t): the second gate selects which new information can be stored in memory, that is, which can become part of the candidate cell state $\widetilde{c_t}$:

$$i_t = \sigma(\mathrm{W_i} x_t + U_i h_{t-1} + b_i) \tag{13}$$

$$\widetilde{c_t} = \tanh(\mathrm{W_c} x_t + U_c h_{t-1} + b_c) \tag{14}$$

3. Output Gate (o_t): the last gate specifies which portion of the cell's state should be displayed and which should be hidden:

$$o_t = \sigma(\mathrm{W_o} x_t + U_o h_{t-1} + b_o) \tag{15}$$

The actual update of the state of cell c_t and of the actual hidden state h_t is obtained by combining the previous contributions using Hadamard products:

$$c_t = c_{t-1} \odot f_t + \widetilde{c_t} \odot i_t \tag{16}$$

$$h_t = o_t \odot \tanh(c_t) \tag{17}$$

Thanks to this combination of filters operating in parallel, the LSTM cell is able to preserve information related to postural kinematics over a large number of frames [37].

Gate Recurrent Unit (GRU)

GRU is a simplified variant of the LSTM cell, designed to reduce computational complexity while retaining the ability to handle long-term dependencies. The GRU combines the cell state and the hidden state into a single vector h_t, reducing the number of gates to two:

- Reset Gate (r_t): it determines what portion of the previous state should be taken into account when calculating the new candidate:

$$r_t = \sigma(\mathrm{W_r} x_t + U_r h_{t-1} + b_r) \tag{18}$$

- Update Gate (z_t): it defines the amounts of both existing and new information to be retained:

$$z_t = \sigma(\mathrm{W_z} x_t + U_z h_{t-1} + b_z) \tag{19}$$

The reset gate is used to modulate the previous state in the computation of the candidate's new state:

$$\widetilde{h_t} = tanh(\mathrm{W_h} x_h + U_h(r_t \odot h_{t-1}) + b_h) \tag{20}$$

Finally, the update gate is used to mix the previous and the candidate states:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \widetilde{h_t} \tag{21}$$

Thanks to this structural simplification, it is possible to speed up the computation phases and reduce the number of network parameters compared to a standard LSTM [38].

Projected LSTM

In embedded applications it is necessary to minimize memory usage and the number of mathematical operations as much as possible. In classic LSTM layers, the Wx operation uses a weight matrix W that grows enormously as the input and the number of neurons increase. This requires a large amount of memory, a problem that the LSTM Projected layer avoids by replacing the classic operation with a seemingly more complex multiplication, WQQ^Tx. In practice, instead of using a large matrix W, the calculation is split: first, the input is compressed by passing through a projection matrix Q, and then it is processed by the weight matrix $W' = WQ$. Storing the matrices Q and W' in memory takes up less space than the large initial matrix W. It is therefore possible to streamline the model without changing the size of its output, allowing the subsequent layers of the network to continue functioning without requiring modifications [39].

Bidirectional LSTM (Bi-LSTM)

The sequential networks described so far process data exclusively in a forward direction over time; therefore, the current state h_t depends only on past events. However, classification accuracy can be improved by also considering the future context. A Bi-LSTM network allows for simultaneous processing via two processing chains of both a forward and a backward sequence. For each time step t, the hidden state vectors produced by one chain are concatenated with those resulting from the other, providing a symmetric and comprehensive view [40].

3.1.2 *Theoretical Foundations of Convolutional Neural Networks*

Convolutional Neural Networks (CNNs) are well-suited for extracting spatial features. The success of these architectures is based on two fundamental properties: local connectivity, which eliminates the need to connect every neuron in one layer to every neuron in the next layer, and weight sharing. In fact, unlike fully connected layers, where the large number of parameters leads to a high risk of overfitting, convolutional layers apply compact linear filters called kernels. These filters scan the entire input applying the same transformation to each local region, ensuring efficient feature extraction and reduced memory usage. The mathematical operation of two-dimensional discrete convolution between an input x and a kernel w of size $k \times k$ is defined analytically as:

$$z_{u,v} = \sum_{t=0}^{k-1} \sum_{s=0}^{k-1} w_{s,t} \; x_{u+s,v+t} \tag{22}$$

The result is then processed by a nonlinear activation function to enable the learning of complex representations [41].

The discrete convolution operator requires the input to have a fixed size, such as the pixel matrix of an image. However, the data generated by the radar sensor and processed in this thesis consist of point clouds, that is, sparse and disordered sets of points. Applying traditional CNNs to this type of data therefore presents challenges [42]. In fact, the point cloud does not have a grid structure; the distance between points, and more importantly, the number of points, varies continuously from frame to frame. Furthermore, the point cloud is stored as an $N \times D$ matrix, where D represents the features associated with the points and N the number of points, and the order in which these points are stored is arbitrary. CNNs are not directly applicable because they are not invariant to permutations and depend strictly on the ordering and local proximity of the data.

To overcome these limitations, some approaches in the literature focused on a technique known as voxelization, which involves dividing three-dimensional space into a grid to perform three-dimensional convolutions. This approach, however, has some drawbacks; for example, it introduces a quantization error regarding spatial resolution and results in significant computational waste, since the body of the person being monitored occupies only a small portion of the room's volume, and consequently the grid will be sparse, forcing the CNN to perform calculations on regions lacking information [43]. This method is therefore problematic for embedded solutions.

Given these challenges, architectures have been developed drawing inspiration from traditional CNNs. The most well-known architecture is PointNet [44].
To implement the weight-sharing mechanism without having to use a grid, PointNet introduces shared multilayer perceptrons (shared MLPs). Applying a shared MLP block to a set of points is equivalent to performing a convolution with a 1x1 kernel. This operation ensures that every point in the cloud is processed by the same mathematical function that projects the information into a higher-dimensional space. While this "1x1 filter" ensures that the model remains invariant to data permutations, it also sacrifices the principle of locality typical of CNNs, since each point is processed in isolation. The only time the architecture has a global view of the point cloud's shape is at the end of processing, when Max Pooling is applied, which allows for the extraction of macroscopic information from the point cloud.
To address the issue of local blindness in PointNet, Point Convolution (PointConv) was introduced, an architecture that reintroduces the concept of convolution [45].
Here, discrete convolution is replaced with a continuous convolution operation. The convolution of a feature function F with the continuous weight function W, evaluated at x, is defined as follows:

$$Conv(W,F)_x = \iiint W(\delta)F(x+\delta)d\delta \qquad (23)$$

Where δ represents the three-dimensional displacement relative to the centroid. However, since the point cloud does not have a continuous volume, the expression in 23 is approximated using a sum calculated over nearby points.

$$Conv(x) = \sum_{x_i \in N(x)} W(\Delta x_i) F(x_i) \tag{24}$$

The network, by using the K-Nearest-Neighbor (KNN) algorithm, selects a subset of reference points called centroids and then selects K points adjacent to them, creating small local sets of points. Within these local sets, the network calculates the relative distances between the points and uses a continuous function to combine their information, replicating the operation of a traditional CNN filter.

Finally, to prevent more reflective areas of the body from obscuring the details of less reflective ones, the architecture employs a density compensation mechanism, whereby a scaling factor $S(x_i)$ inversely proportional to the local density is added to the formula expressed in 24.

$$Conv(x) = \sum_{x_i \in N(x)} S(x_i) W(\Delta x_i) F(x_i) \tag{25}$$

In this way, the network is able to rebalance the importance of each node and capture the geometric relationships between adjacent nodes and local structures.

The centroids calculated in the first step become the new input points for the next layer, and with each step, the number of centroids decreases, but the information associated with them increases. When very few points remain in the point cloud, the network applies a max pooling operation, resulting in a one-dimensional output vector. This vector is fed into an MLP, which produces its output.

3.1.3 *Automation of Network Design: Neural Architecture Search*

Manually designing a neural network is a complex process and for this reason diverse approaches have been envisioned; more specifically, NAS was developed as a solution to this implementation challenge, becoming the primary tool for automatically designing neural networks that are effective in achieving the desired objective. NAS does not merely optimize weights or fine-tune an existing model by adjusting a few hyperparameters; rather, it automates the architecture design process using an algorithm that searches for the optimal structure for the task [46], [47].

NAS is essentially based on three main elements:

1- Search Space: it defines the set of components that the algorithm might use and find useful within the final architecture. Within the search space, it is also possible to define the problem to be solved to narrow such space and reduce the computational complexity of the search.
The Search Space may therefore contain the various layers or elementary cells that could be part of the final architecture, and it is also possible to specify different hyperparameters for each of these individual components so that the best possible combination can be identified. In addition, NAS is also capable of indicating the number of cells or layers, as well as the connection scheme.

2- Search Strategy: this is the methodology used to navigate the search space in order to find the best network. The search strategy must balance exploration and refinement of the network. In other words, it must explore as many alternatives as possible within the search space without wasting too much time, while simultaneously refining architectures that already show promising results, avoiding convergence toward hasty and suboptimal solutions. There are several types of search strategies with various levels of complexities, ranging from Random Search to Reinforcement Learning, where a controller

generates architectures that are scored based on their accuracy, or algorithms that base their strategy on gradient descent [47].

3- Performance Estimation Strategy: To fully and comprehensively evaluate the effectiveness of a neural network, it must be trained on the training dataset and then validated on the test set. However, this approach is so time-consuming and resource-intensive that it is impractical to apply, since the process would need to be repeated on multiple networks, each of which could take hours or even days to complete. The actual strategy, therefore, involves avoiding training the network until it converges. Instead, early stopping techniques are used, along with models capable of predicting the final accuracy by observing only the first few epochs of the training phase.

3.1.4 *Explainable AI*

AI architectures present the problem that they operate as black boxes; in other words, neural networks often use millions of parameters, making it impossible to understand the exact logical process that leads to a particular decision. To clarify the internal mechanisms of these models, XAI techniques are employed to identify the key features driving the network's predictions [48], [49].

Permutation Feature Importance (PFI) [50] technique was used in this thesis. This algorithm is applied post-training and is used to quantify the importance of each feature. More specifically, a single feature is altered at a time by randomly shuffling its values within the test set, while keeping everything else unchanged. The network's response is then evaluated, and if this shuffling causes a drastic drop in the model's overall accuracy, it is concluded that this feature played an essential role in the final decision.

CHAPTER 4
RESEARCH METHODOLOGY

4.1 **Data Collection**

To train and validate artificial intelligence algorithms for posture recognition, a large and diverse dataset is required. The problem encountered with mmWave radar is that publicly available datasets are rather limited: some of them contain actions that are unsuitable for the purpose of this thesis, which is the detection of simple daily activities, while others are not publicly available, or they were collected using radar configurations that differ significantly from those adopted in this thesis (in terms of hardware). For this reason, a significant portion of the work carried out in this thesis was dedicated to conducting an experimental campaign. The objective was to build a proprietary dataset based on the point clouds generated by the radar during the performance of various daily activities.

Data collection took place in a room located within the laboratories of the University of Pavia. The room in question is rectangular and spacious; however, to simulate the physical dimensions of a real-life home environment, the area of interest within which monitoring was conducted was reduced by setting a bounding box approximately 3.5 meters wide and 5 meters long. The radar was installed on a wall at a height of approximately 2.35 meters above the floor. After several tests, this configuration proved to be the most suitable for achieving good coverage of the room and better monitoring of the subject's movements. In fact, this type of overhead positioning allows for good observation of variations along the body's z-axis.

To obtain a dataset that was as diverse as possible, the experimental study involved a total of 27 participants with varying physical characteristics in terms of height and

body type. As for their age, however, the dataset is not very diverse in that regard; in fact, the average age of the subjects is around 22 ± 3. This represents a weakness in the collected dataset, because the way an adult of this age moves is certainly different from that of an elderly person, who is the primary target group for which the sensor is intended. On the other hand, physical diversity is a strength of the dataset. In fact, characteristics such as height, weight, and chest circumference significantly alter the point clouds, which exhibit very different densities and spatial distributions [51]. Furthermore, despite the age-related limitations, constructing a dataset with healthy adults provides crucial advantages. It facilitates a much more efficient and safer data collection and updating process, completely bypassing the severe ethical, logistical, and safety constraints associated with involving frail or elderly individuals in extensive physical trials. This approach allows also for the creation of a highly versatile dataset that can be readily utilized by other research groups. It serves as a valuable resource for investigating a wide variety of daily postures within the broader field of HAR, as well as providing a foundational benchmark for fall prevention applications.

Each participant performed a series of daily life activities, following a defined protocol. To prevent the various neural networks from simply learning the sequence of actions rather than the actual characteristics of the movements, the activities were performed in random order during each session [52]. In each acquisition, the participants performed all actions only once, and each participant completed between 5 and 10 acquisitions. This allowed for the collection of diverse performances, such as different speeds depending on the participant. As a result, the dataset is less rigid and more robust for the training phase. The collected data were manually labelled in a subsequent phase by visualizing the point cloud evolution and the frames corresponding to the various postures, to obtain the ground truth necessary for the supervised training of the models.

4.1.1 *Posture Analysis*

The dataset has been organized into six main categories, selected to represent some of the most relevant postures for monitoring frail or elderly individuals:

Class	Action
1. Stand	Walk or stand still
2. Sit	Sit on a chair
3. Grasp	Pick up an object
4. Floor Sit	Sit on the ground
5. Lie Down	Lie Down
6. Transient	Postural Transition

Table 4.1. Postures included in the dataset and their descriptions

Stand: this class represents the subject's movement within the area monitored by the radar. The point cloud appears elongated, closely following the silhouette, and the centroid is located at a high elevation relative to the floor. Doppler variations are observed due to the movement of the limbs while walking.

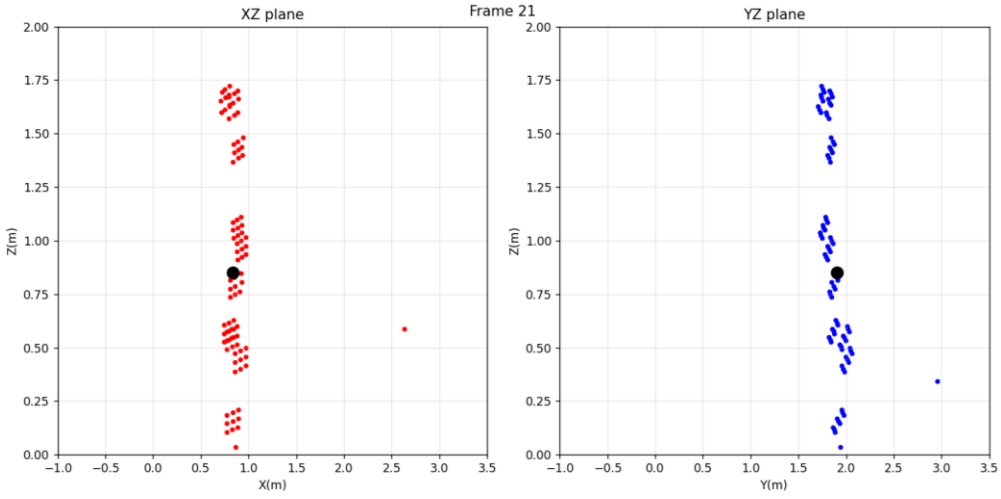

Figure 4.1. Point cloud for the stand posture

Sit: it refers to a static posture in which the subject is seated on a chair. The point cloud tends to stabilize at a height midway between the subject and the floor. Compared to the standing position, the distribution of points is more compact and less extensive. Due to prolonged immobility, the number of points gradually tends to decrease.

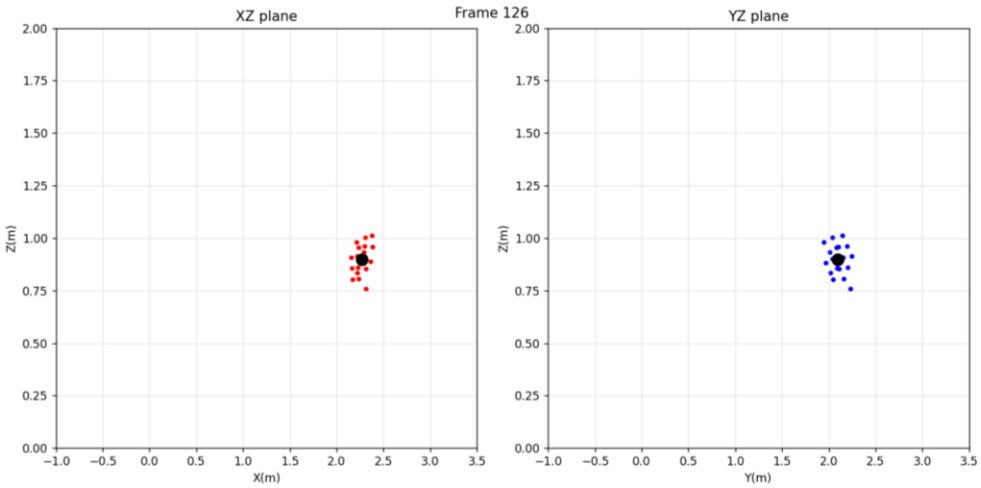

Figure 4.2. Point cloud for the sit posture

Grasp: it represents the motion of the body required to pick up an object from the ground. During this action, the subject quickly bends forward toward the floor and then returns to the starting position. This behavior results in significant changes in the geometry of the point cloud and in the height of the body's center of mass.

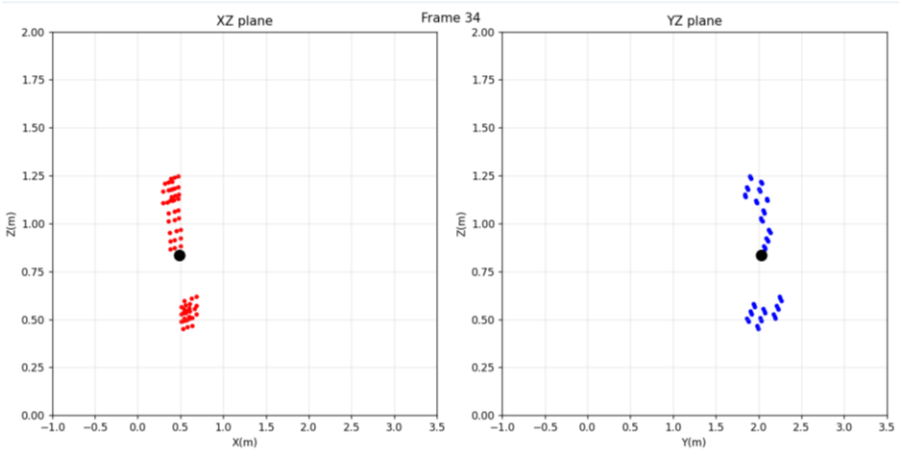

Figure 4.3. Point cloud for the grasp posture

Floor Sit: this class describes a posture in which the subject is sitting directly on the floor. From a geometric perspective, the point cloud is located at a low elevation relative to the vertical z-axis, and since the subjects were sitting with their sides facing the radar, their bodies tend to form a sort of "L" shape in the XZ plane.

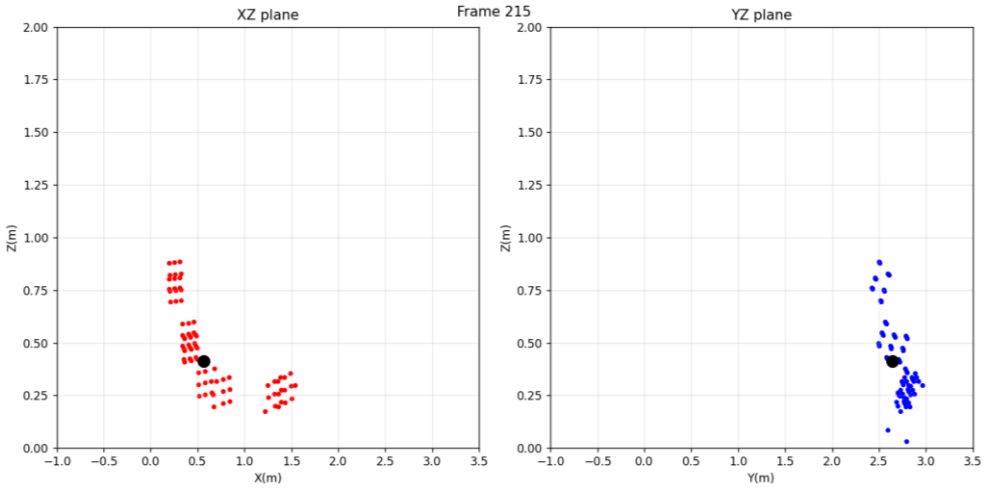

Figure 4.4. Point cloud for the floor sit posture

Lie Down: it represents a subject lying flat on the floor. This is a static posture characterized by minimal vertical extension and a greater distribution of points across the horizontal plane, although as the person maintains this posture, the points tend to cluster together and decrease in number. In this configuration, the body's center of mass is among the lowest in the dataset.

Transient: this class includes all those intermediate situations in which the subject is transitioning from one posture to another, for example, from a seated to a standing position or vice versa. During these transitions, the body assumes configurations that do not fully belong to either the initial or final posture. From the point cloud perspective, transient postures are characterized by gradual changes in the spatial distribution of points and the height of the body's centroid. Numerous points are present due to the movement, which causes multiple reflections. The inclusion of this class improves the classifier's robustness, reducing potential classification errors during posture transitions.

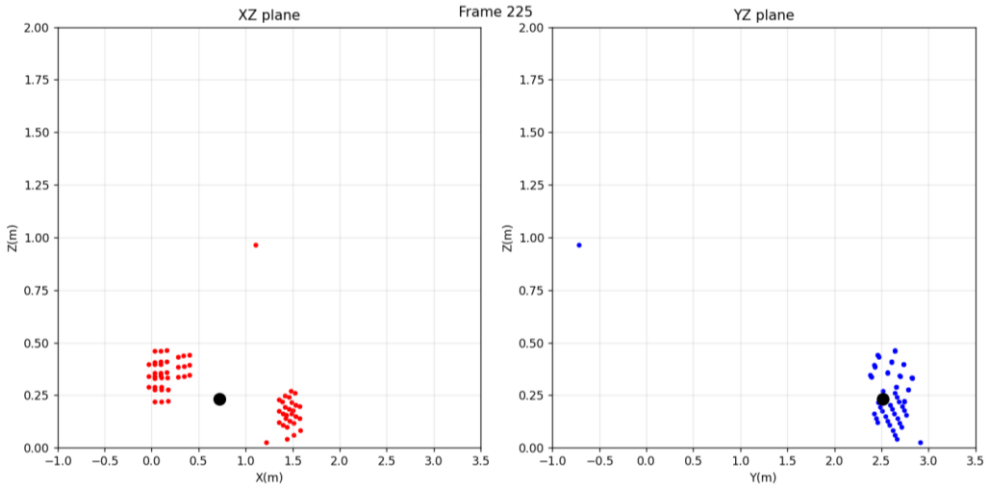

Figure 4.5. Point cloud for the lie down posture

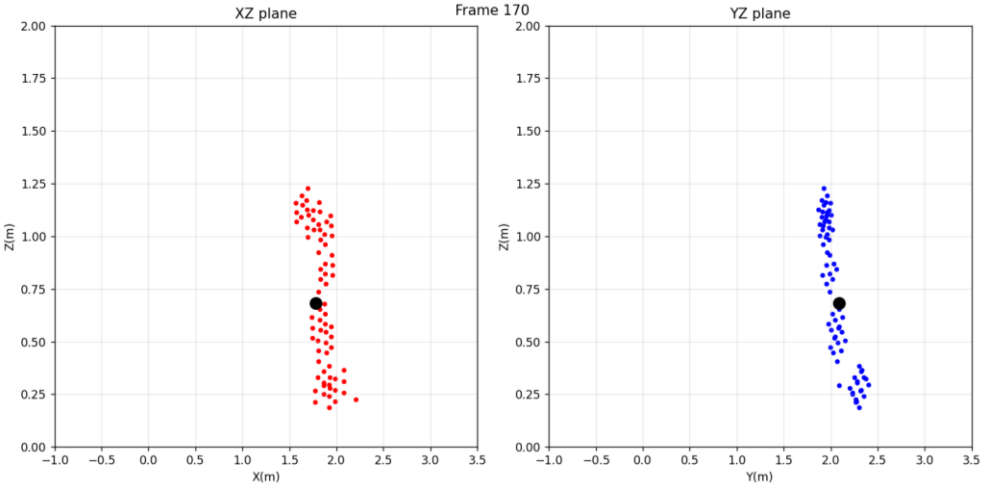

Figure 4.6. Point cloud during a transient from sit to stand

4.1.2 *Critical Issues*

During the experimental campaign, analysis of the acquired data revealed a clear improvement in the quality of the point cloud compared to what was observed in the bachelor's thesis. Firmware enhancements achieved cleaner, more stable and, overall, more reliable data. One of the most noticeable aspects was the significant reduction in the phenomenon of "spot splitting," that is, the splitting of the detected subject. In

the new acquisitions, this effect was much less pronounced, allowing the system to track the subject's outline continuously and in a more accurate way during movement. Even while performing activities that are more complex or characterized by rapid movements (such as grasping), the point cloud exhibited greater geometric consistency. In general, the subject's body is represented more faithfully to reality even when posture changes rapidly. This improvement in the quality of the source data is a crucial factor for all subsequent work. Having a point cloud with less noise and one that more accurately reflects the body's actual geometry means providing artificial intelligence models with more reliable information.

However, the challenge posed by the low point cloud density in stationary situations persists. Due to the inherent characteristics of FMCW radar sensors and the static noise removal algorithms required, stationary human targets are often discarded along with environmental reflections. This results in a low signal-to-noise ratio (SNR) for the subject's micro-movements, causing detection algorithms to discard most human reflections, leading to spatial representations characterized by few points in the point cloud [53].

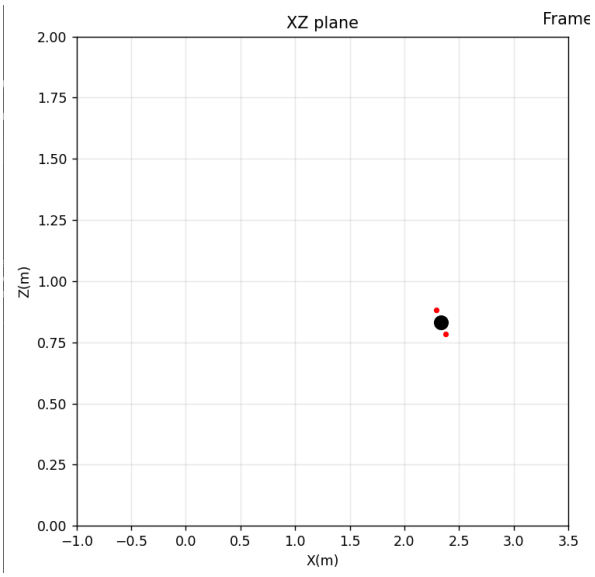

Figure 4.7. Point cloud during a prolonged sitting

4.1.3 *External Dataset*

Despite the difficulties in finding a dataset suitable for the thesis, the public MM-DCDR dataset [54] was identified. This dataset, released by Gao et al., is useful for evaluating the generalization capabilities of the implemented architectures. The dataset includes approximately 352,000 frames in which 11 subjects performed 8 different activities in various rooms, similar to those carried out in the dataset collected at the University of Pavia laboratory:

- standing,
- sitting on a chair,
- sitting on the floor,
- turning a chair,
- bowing,
- waving hands,
- squatting,
- lying down.

The data were collected using two different radar sensors; however, to remain consistent with the objective of the thesis, only the point clouds collected with the TI AWR1843 sensor were considered.
An important note regarding this dataset concerns the morphological differences between these data and those collected in the proprietary dataset. In fact, the point cloud in the proprietary dataset closely follows the silhouette of the human body, whereas in the MM-DCDR dataset, the clusters are scattered and chaotic structures are visible. The points, therefore, do not outline a recognizable shape but appear to be randomly distributed, making it impossible to visually identify the posture of the monitored subject.

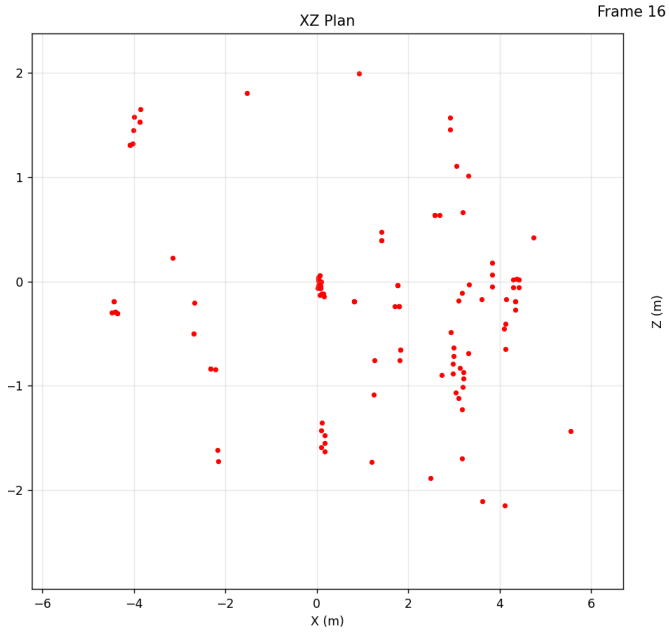

Figure 4.8. Point cloud for the stand posture in the MM-DCDR dataset

This is likely due to the different methodological choices made by the researchers, most notably the decision to mount the sensor directly in front of the subject being monitored, at a distance of 1 meter and at a height of 1 meter above the ground meter. Another reason is that no target-tracking or spatial clustering algorithm, such as the well-known DB-SCAN, was implemented. Consequently, the point cloud does not only cover the subject's body but also includes any false targets generated by signal reflections off the walls.

Clearly, the absence of a silhouette prevents the use of neural networks that rely on geometric features.

Another important methodological aspect of this dataset is how a 'frame' is defined. Here, a single frame consists of a compilation of point clouds collected over 60 actual sensor frames, with the resulting total number of points being standardized to 128.

4.2 **Pre-processing techniques**

4.2.1 ***Confidence Ellipsoid***

Before proceeding with the design and training of new Artificial Neural Networks (ANN), the initial phase of this work involved the application of the classification model developed in [21]. The goal is to test the developed architecture on the newly collected dataset, given the high variability introduced by the 27 participants involved. The architecture in question is a sequential hybrid model designed to minimize computational overhead. It consists of a fully connected feature extractor equipped with a ReLU nonlinear activation function [55], which takes a set of geometric parameters as input and projects them into a latent space. This output is subsequently processed by different types of Recurrent Neural Networks (RNNs), tasked with extracting temporal dependencies between sequential frames, and then converges into a final linear classifier. The key input feeding the entire network is not the raw point cloud, but rather the confidence ellipsoid.

To explain what a confidence ellipsoid is, it is helpful to start with the concept of a confidence interval. Given a cluster of points, the 95% confidence interval is the segment of space, centered on the cluster, that has a 95% probability of containing any point detected by the sensor [56].

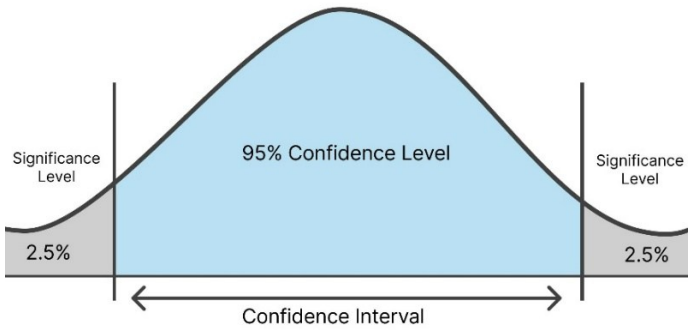

Figure 4.9. 95% Confidence interval [57]

The confidence ellipsoid is the extension of this concept into three dimensions. It consists of a volumetric boundary that models the spatial distribution of the cluster, enclosing 95% of the points within it.

By employing the confidence ellipsoid, a volumetric envelope is generated to approximate the sparse point cloud into a defined three-dimensional shape, facilitating its association with a specific posture. Furthermore, this ellipsoidal modeling enables the extraction of highly informative geometric features, such as the volume and eccentricity of the cluster.

Eccentricity is a parameter that indicates how much an elliptical orbit differs from a perfect circle. The eccentricity of an ellipsoid can be defined in relation to the 2D planes (XY, XZ, YZ) that intersect the ellipsoid. Each plane contains an ellipse, and the eccentricity can be calculated for each of these ellipses using the formula:

$$e = \sqrt{1 - \frac{b^2}{a^2}} \qquad (26)$$

where b and a are two of the three semi-axes of the ellipsoid, depending on which of the three eccentricities is being calculated. The more the two semi-axes are equal, the closer the formula's result is to zero, the more the ellipse tends to be circular.

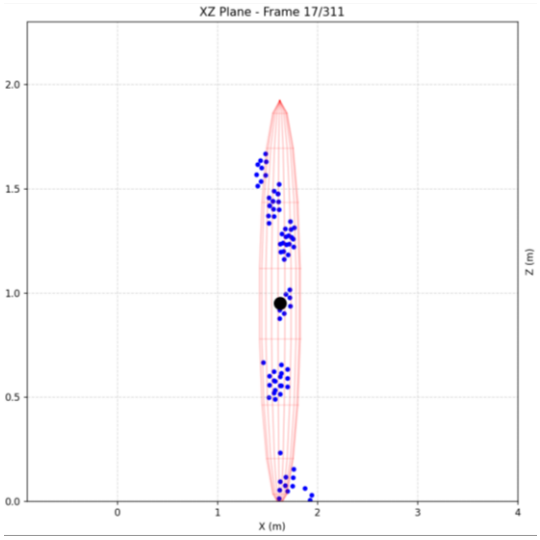

Figure 4.10. Confidence ellipsoid for a standing person

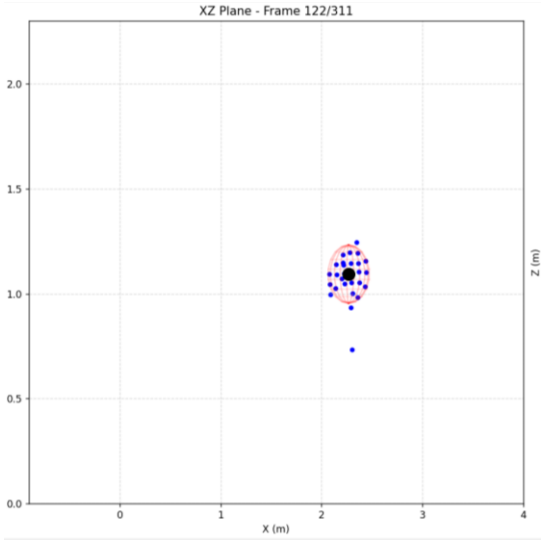

Figure 4.11. Confidence ellipsoid for a seated person

In Figure 4.10, it is possible to see how the ellipsoid is very elongated and, consequently, very high eccentricity values are expected along all three planes. In Figure 4.11, however, given the clustering of the points in the scatter plot that causes the ellipsoid to become almost a sphere, the eccentricities will be close to zero.

From a practical standpoint, the construction of this confidence ellipsoid begins with a filtering process. All points whose distance exceeds three standard deviations from the centroid are discarded and do not contribute to the formation of the ellipsoid. Furthermore, in this initial approach to constructing the confidence ellipsoid, the covariance matrix is not calculated; instead, only the standard deviation along the three axes is determined. Consequently, the ellipsoid cannot be tilted. This is done to give greater weight to the eccentricity information. In fact, since it cannot tilt, the ellipsoid will tend to resemble more closely to a sphere so as to encompass all points, hence resulting in a reduction in eccentricities.
The features extracted using this model and subsequently fed into the AI models are:

- mean of the z coordinate,
- eccentricity along xy plane,

- eccentricity along xz plane,
- eccentricity along yz plane,
- doppler signal,
- SNR,
- volume.

4.2.2 *Double Ellipsoid Model*

To improve postural analysis, a two-ellipsoid model was developed, with the upper ellipsoid representing the torso and the lower ellipsoid representing the lower limbs. In this approach, by allowing the ellipsoids to tilt, it is possible to track other important information, such as the tilt of the two halves of the body relative to a vertical axis, thereby providing more features related to the dynamics of movement to distinguish them more precisely.

To separate the points belonging to the upper body from those belonging to the lower body, a partitioning is performed; however, this is not based on statistical data such as the median, which could compromise the construction of the ellipsoids if there is a difference in density between the points reflecting the legs and those reflecting the torso. Instead, the equation (27) is used:

$$z_{cut} = z_{min} + \frac{z_{max} - z_{min}}{2} \qquad (27)$$

The maximum and minimum elevation points are averaged to determine the exact midpoint, defined as z_{cut}. All points with an elevation greater than z_{cut} will be used to create the upper ellipsoid, while all points with an elevation lower than z_{cut} will be used to create the lower ellipsoid.
At this point, for each of the two clusters, the centroid and the covariance matrix are calculated, from which two essential mathematical pieces of information can be

obtained: eigenvalues and eigenvectors. The principal eigenvector, associated with the largest eigenvalue, allows us to determine the radius, and thus the extent and direction of maximum variance and, consequently, the inclination of the ellipsoid [58], [59].

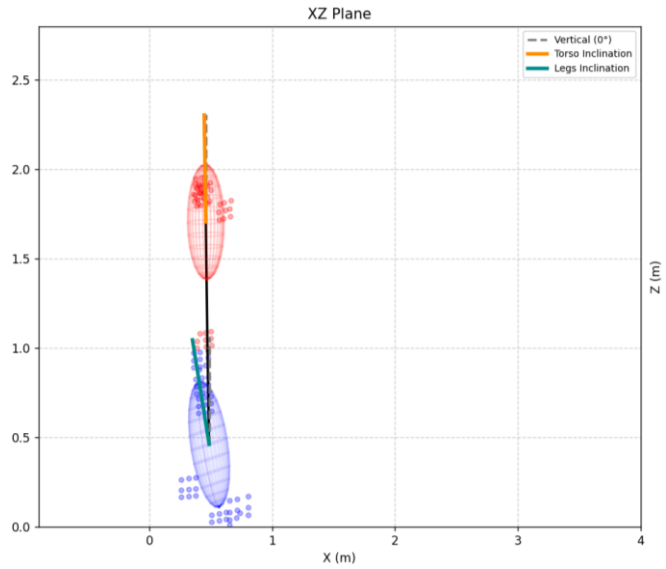

Figure 4.12. Double confidence ellipsoid for a standing person

The features extracted using this model and subsequently fed into the AI methods are the following:

- mean of the torso z coordinate and mean of the legs z coordinate,
- torso tilt and legs tilt with respect to the vertical axis,
- torso SNR and legs SNR,
- torso doppler and legs doppler,
- distance between centers.

4.2.3 *Data Augmentation*

Another challenge for radar sensors is data sparsity, which represents a problem regarding the variable dimensionality of the point cloud and, consequently, of the

ellipsoid generated. In fact, radar often returns frames with a high density of points, while at other times it returns frames that are very sparse in information [60], and so the shape of the point cloud and the confidence ellipsoid may be adversely affected. To address this issue and provide classification models with a more consistent volume of data, after careful consideration, it has been decided to standardize the number of points by setting a minimum threshold of points per frame. In cases where the number of points falls below this threshold, the missing points are generated artificially. This, however, is not a simple duplication of points or the addition of random noise, but rather a form of data augmentation that is also physically consistent. In practice, to generate these points artificially, two geometric transformations (translation and rotation of a few centimeters) are performed starting from the existing points, accompanied by a recalculation of the SNR to adapt it to the new points.

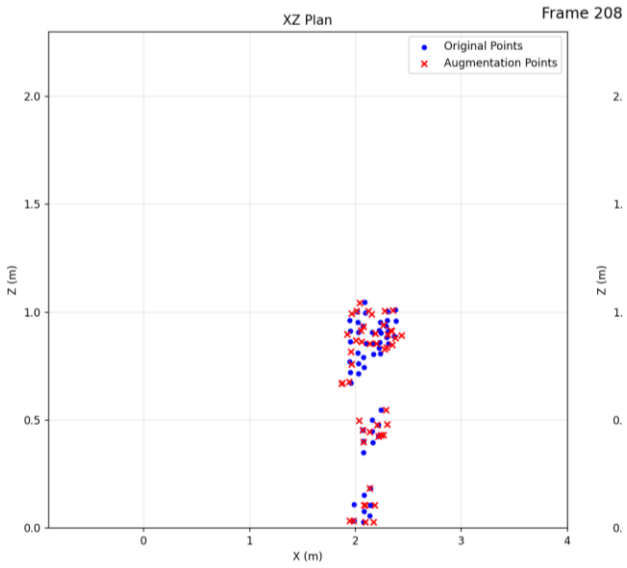

Figure 4.13. Data Augmentation for a seated subject

The first geometric transformation is a random translation of $\pm 3cm$, used to simulate a slight spatial movement of the target away or toward the viewer. From a geometric standpoint, to translate a point in three-dimensional space without altering its angular position, it is not sufficient to simply add an offset to a single coordinate, such as the

y coordinate but it's necessary to multiply all the coordinates of that vector by the same scalar:

$$d_{new} = k\, d_{old} \tag{28}$$

It should also be noted that the received signal strength is not constant across space but varies at a rate proportional to the inverse of the scale factor raised to the fourth power. To maintain the realism of the point cloud, it is therefore necessary to adjust the SNR, that is, to reduce it for points farther away than the original and increase it for points closer to the original.

$$SNR_{new} = \left(\frac{d_{old}}{d_{new}}\right)^4 SNR_{old} \tag{29}$$

In addition to changes in distance, points may undergo a small random angular rotation of $\pm 3°$. To rotate a point around the origin in three-dimensional space by an angle θ, the point's coordinates must be multiplied by a rotation matrix [61]:

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \end{bmatrix} \tag{30}$$

From which:

$$x_{new} = x_{old}\cos(\theta) - y_{old}\sin(\theta) \tag{31}$$

$$y_{new} = x_{old}\sin(\theta) + y_{old}\cos(\theta) \tag{32}$$

In this case as well, to ensure a translation that accurately reflects reality, the physics associated with the radar sensor must be considered. The gain (G) of an antenna, in fact, decreases significantly as the theta angle changes. It is maximum at the center of the beam and decreases as the beam rotates. Consequently, the SNR of the point is recalculated as reported in 33:

$$SNR_{new} = \left(\frac{G_{new}}{G_{old}}\right) SNR_{old} \quad (33)$$

By applying these formulas, it is possible to enhance frames with few data points by adding extra data points, while always adhering to the physics of the radar sensor's electromagnetic waves [62].

4.3 **Neural Networks Implementation**

Following the theoretical foundations outlined in Chapter 3, this chapter details the practical implementation of the neural architectures developed. A comprehensive analysis of the performance and results achieved by these models will be subsequently discussed in Chapter 5

4.3.1 ***PointConv***

PointConv extends the core principles of a traditional CNN, specifically local feature aggregation and weight sharing, to sparse and unordered point clouds. As previously mentioned, the primary issue encountered when working with networks that utilize the concept of convolution is that the tensors must have a fixed size. Point clouds, however, vary from frame to frame. To resolve this inconsistency, a thresholding mechanism is employed:

- Number of points > 64 (128 for the MM-DCDR dataset): the excess points are discarded by performing a simple random selection.
- Number of points < 64 (128 for the MM-DCDR dataset): to reach the quota of 64 points, some are duplicated. Duplication does not consist of a simple overlay, but rather the data augmentation technique explained in Section 4.2.3 is applied.

The threshold of 64 was determined by analyzing the average point density per frame in the proprietary dataset (Table 4.2). The resulting average was rounded to the nearest power of 2 to optimize tensor operations.

Posture	Average Points
Stand	86
Sit	24
Grasp	51
Floor Sit	68
Lie down	37

Table 4.2. Average points per posture in the proprietary dataset

From these values, the average was calculated and rounded to the nearest power of 2 to facilitate tensor operations.

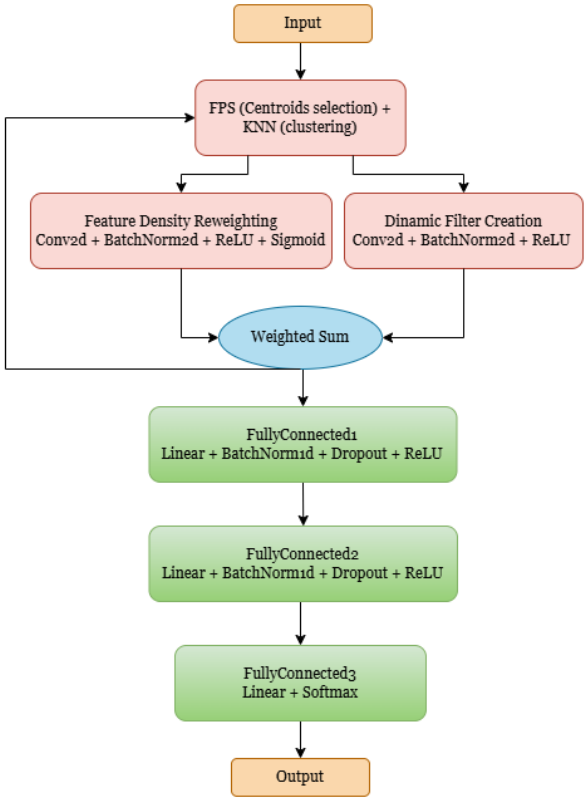

Figure 4.14. PointConv architecture

The points are then fed into the network. The first convolutional layer selects some key points, called centroids. This selection is performed using the Farthest Point Sampling (FPS) algorithm, which selects the farthest point from those already selected to ensure uniform coverage across the entire space. This ensures that the various centroids cover the entire body and not just a portion of it. For each of the centroids, the KNN algorithm is used to select the *n* points closest to them.

At this stage, the network dynamically generates the weight matrices used as convolutional filters. Through 1x1 convolutional layers, the spatial features are expanded, enabling a more complex and high-dimensional latent representation. In parallel, a density reweighting mechanism is applied to compensate for the non-uniform spatial distribution of the point cloud. Because a fixed number of nearest neighbors can occupy vastly different spatial volumes, a scaling factor inversely proportional to the local density is computed. This adjustment reduces the impact of highly clustered points, preventing them from mathematically dominating and obscuring sparser, yet structurally critical, regions. This calculation is performed using a multi-layer perceptron (MLP), whose output is passed through a sigmoid activation function to map the rebalancing factor into a continuous range between 0 and 1. Finally, the layer concludes by computing a weighted sum between the dynamically generated filter matrix and the cluster's input features, which have been proportionally scaled by their respective density factors.

This operation is repeated several times, allowing to end up with a single, information-rich point.

The information-rich super-sample is simply a vector of 1024 features, representing a summary of the frame. This vector is fed into a final MLP consisting of fully connected layers that progressively combine the information while reducing the vector's dimensionality (1024 → 512 → 256). The final fully connected layer generates an output vector as large as the number of postures, and after applying the Softmax function, each cell will contain the probability of each posture. The highest value indicates the posture predicted by the network.

4.3.2 *PointConv LSTM*

To achieve even more effective postural recognition, incorporating temporal information proves to be very useful. A hybrid architecture has therefore been implemented that combines the PointConv network with a recurrent LSTM layer. This solution allows for the analysis of a sequence of consecutive frames rather than a single frame at a time.

The architecture differs from the classic PointConv mainly in its final section. The final MLP, composed of fully connected layers, no longer produces an immediate probabilistic output. Processing stops at the output of the second fully connected layer, where a vector of 256 features is produced, representing a summary of what was seen in that frame. A long sequence of 60 frames (equivalent to 6 seconds of footage), each summarized in a vector of 256 features, is split into 12 packets of 5 frames each and sent to the LSTM layer (with 128 hidden units), which analyzes how the features vary over this long sequence of frames [63].

However, to ensure GPU processing without relying on long iterative cycles, temporal unfolding is performed first. That is, the various input frames are initially treated as independent samples, allowing them to be processed in parallel on the GPU. At the end of this phase, the resulting vectors are reordered to restore the correct chronological sequence and then fed into the LSTM layer.
Another noteworthy aspect of this architecture is the implementation of temporal padding. In fact, actions have variable durations, they are not always fixed, and to avoid issues during GPU processing (which expects inputs of fixed size), it is necessary to standardize them by adding zeros, without this causing confusion in the network. The model therefore implements a mechanism that ensures the classifier extracts information and draws its conclusions based solely on the actual frames, and not on the padding frames, which serve only to compensate for the correct matrix dimensions. Finally, the LSTM output is projected by the last fully connected layer into a vector of the same dimension as the number of classes, and here too, through

the Softmax function, the content of these cells is converted into probability. The class with the highest associated probability will be the network's prediction.

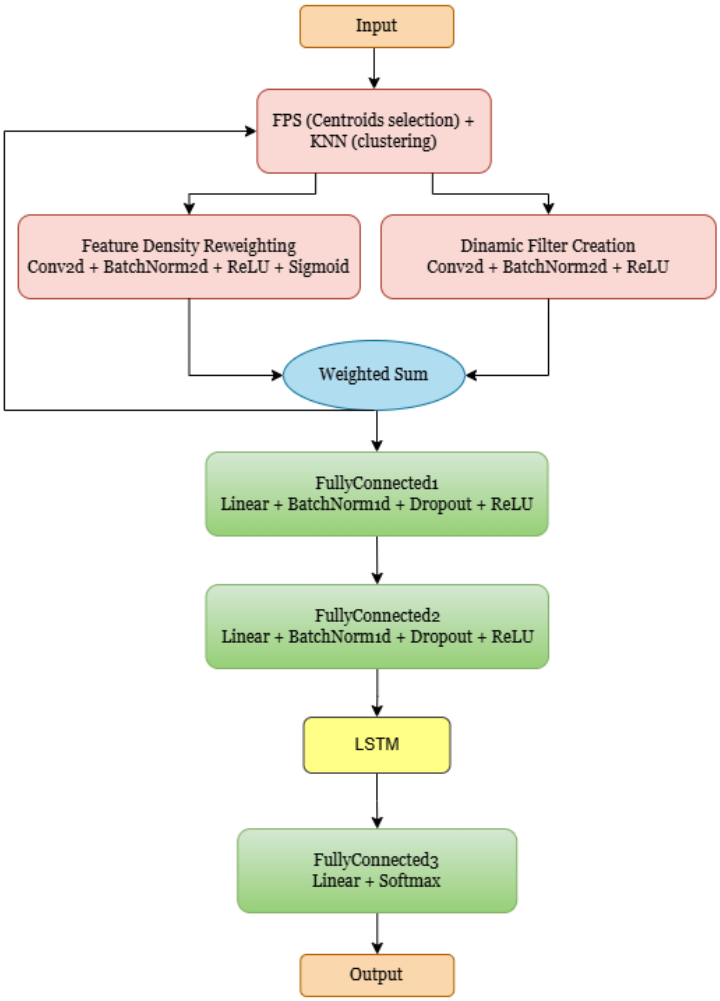

Figure 4.15. PointConv LSTM architecture

4.3.3 *NAS Optimization*

A network based on a single recurrent layer [21] was developed and even though it achieved reasonable accuracy, it left room for significant improvement. To identify the optimal evolution of this architecture the NAS approach was employed. Since the

model will perform real-time inference on an embedded architecture, the search algorithm was trained to optimize four conflicting metrics:

- accuracy,
- latency,
- memory footprint,
- computational cost.

The research framework has been divided into three modules: the first one for preprocessing, the central module dedicated to the recurrent architecture itself, and the third representing the pipeline that transmits the output from the recurrent layers to the generation of the network's actual output, that is, the prediction. In addition, the selection of certain hyperparameters is also planned.

The first out of the three modules in Figure 4.16 is the feature extractor. The algorithm could choose whether to pass the raw data directly to the recurrent layer, or to first have the data processed by one (Shallow MLP) or two linear layers (Deep MLP). In addition, it could also select other details here, such as the activation function and the number of neurons.
The second module represents the core choice of the recurrent architecture. Here, the algorithm could select the best option from various recurrent layers such as GRU, LSTM, Bi-LSTM, and Proj-LSTM. In this case as well, the algorithm could determine the number of these layers (one or two), the number of hidden neurons, and the dropout rate.
The third and final module is responsible for the final classification, and the algorithm could choose between a simple linear layer that took the recurrent output and transformed it into predictions (Shallow MLP), or an MLP (Deep MLP).

Furthermore, NAS was also used to identify the optimal values of two hyperparameters (Table 4.3).

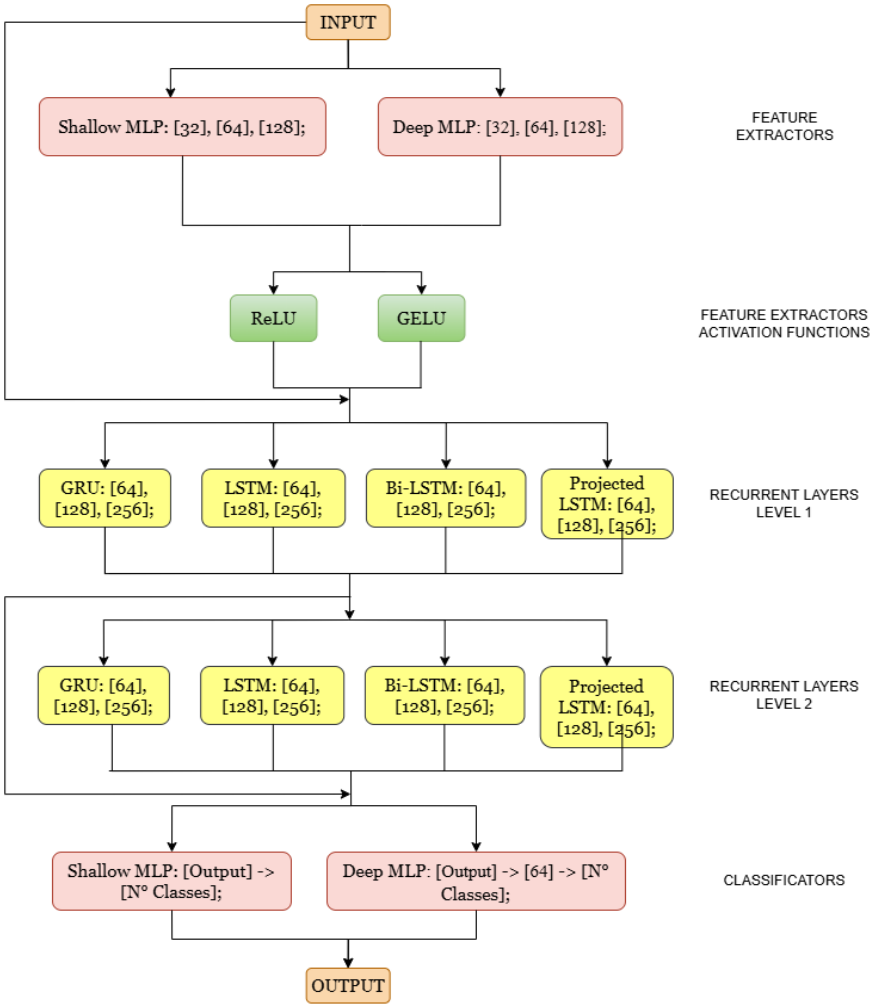

Figure 4.16. Search space used to identify the best architecture.

Architectural Choice	Search Space Options	Description
Dropout Rate	[0.1 – 0.5]	Percentage of neurons deactivated for generalization
Learning Rate	[1e-4 – 1e-2]	Step for optimizing weights during training

Table 4.3. Search Space for two hyperparameters

To identify the best solution, Bayesian optimization was adopted using the Tree-structured Parzen Estimator (TPE) algorithm provided by the Optuna framework [64]. The TPE models the search space probabilistically and learns from the results of past iterations. In the initial iterations, the system evaluates an architecture and

identifies a certain statistical dependency, which, for example, indicates that using heavy recurrent architecture, such as two Bi-LSTM layers, ensures good accuracy but worsens latency and memory usage. Consequently, based on this information, the algorithm narrows the search space, for example, by changing the type of layer or reducing the number of layers.

To determine whether a particular combination of layers and hyperparameters is effective in terms of accuracy, the entire training cycle is not completed; this is done precisely to avoid excessive delays in identifying the optimal architecture. Instead, training is intentionally terminated before its natural conclusion and, based on the accuracy achieved up to that point, an estimate is made of the model's true performance. Other network metrics, such as memory footprint and computational cost, are instead precalculated by providing a dummy input to the network. Finally, the process concludes with the selection of the best architecture. The most significant results obtained are represented in the following table:

Feature Extractor	**RNN Core (Neurons, layers)**	**Final Classifier**	**Acc (%)**	**FLOPs (M)**	**Mem. Footprint (MB)**	**Latency (ms)**
S_MLP (128, ReLU)	LSTM (64, 1)	S_MLP	93.56	0.6778	0.2144	0.5437
S_MLP (32, GELU)	P-LSTM (128, 1)	S_MLP	92.80	1.0267	0.1856	0.9634
None	P-LSTM (64, 1)	S_MLP	92.56	0.5472	0.1283	0.6771
None	LSTM (128, 1)	S_MLP	92.42	1.0883	0.3447	0.3462
S_MLP (128, GELU)	Bi-LSTM (64, 1)	D_MLP	94.70	1.2881	0.4353	0.6732
None	GRU (64, 1)	D_MLP	93.56	0.2656	0.0987	0.7532
S_MLP (128, GELU)	GRU (64, 2)	S_MLP	93.94	0.8329	0.2622	1.4090
S_MLP (32, GELU)	Bi-LSTM (128, 2)	D_MLP	95.83	6.8186	2.2107	1.5890

Table 4.4. NAS architecture results

The two neural networks highlighted in yellow in Table 4.4 were selected because they provide the optimal balance among the various metrics considered:

- Accuracy-Oriented Architecture: it maximizes performance in terms of classification accuracy.
- Latency/Memory-Oriented Architecture: a very lightweight one with a small number of parameters and, consequently, a short inference time.

Accuracy-Oriented Architecture

The Accuracy-Oriented network extracted via NAS is structured as follows: first, the data augmentation explained in 3.1.5 was applied. Then the feature extractor, which receives 60 frames as input divided into 12 batches of 5 frames each, consists of a single linear layer followed by a GELU activation function [65]. Next is the recurrent core, organized into two Bi-LSTM layers to enable the network to process the input sequences in both directions. The final part of the network features an MLP with two linear layers and a ReLU activation function, which generates the prediction as output.

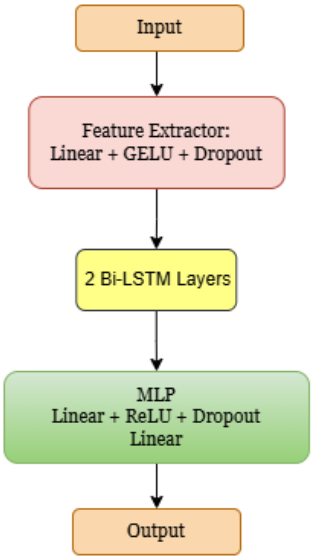

Figure 4.17. Accuracy-Oriented architecture

Latency/Memory-Oriented Architecture

The Latency/Memory optimized network extracted via NAS is structured as follows: no feature extractor is present, so the input (which is composed of 60 frames divided into 12 batches of 5 frames each) after undergoing data augmentation is passed directly into the recurrent layer. This consists of a single, non-bidirectional GRU layer with only 64 neurons. This is where most of the memory reduction is achieved, differentiating this approach from the previous network. This result was expected given that, by definition, the GRU is one of the least computationally expensive recurrent layers. The final part of the network, on the other hand, features an MLP that is very similar to that of the Accuracy-Oriented model, with the sole difference that dropout is not used here.

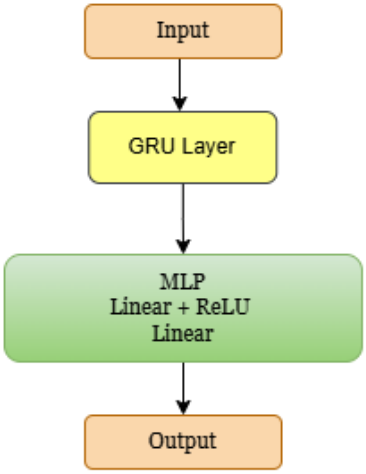

Figure 4.18. Latency/Memory-oriented architecture

4.4 **Training and Evaluation Metrics**

4.4.1 ***Training***

The various architectures implemented were trained. For each network with a recurrent core, the input has always been the same: 60-frame intervals, divided into 12 packets of 5 frames each. The other networks work frame by frame. To train the networks it is necessary to configure and define several hyperparameters and training solutions.

An epoch represents a complete training cycle that the network performs while processing the entire dataset. The number of training epochs is set to 30, since it has been observed that a lower number limits the network's potential. In fact, stopping too early resulted in lower accuracy values during inference on the test set. Conversely, higher values such as 50 or 100, would have caused the network to overfit, that is, to achieve an extremely high accuracy value on the training set (which it essentially memorized) while maintaining an accuracy value on the test set that fell far short of the one achieved during training.
As for the batch size, 32 was chosen. It has been demonstrated in the literature that the use of these small batches, called mini-batches, facilitates the identification of the optimal point more quickly and with less uncertainty.

A crucial aspect of the training methodology involved evaluating the models' generalization capabilities across varying volumes of training data. To achieve this, the dataset was systematically partitioned using progressively increasing training-to-testing ratios. The evaluation started with an initial 50-50 split, advancing through 60-40, 70-30, 80-20, up to a final 90-10 configuration. This progressive splitting strategy allowed for a comprehensive assessment of how data scarcity or abundance impacted the learning convergence and the overall stability of the networks.

Cross-entropy is used as loss function. This function is used to calculate the error made by the network. In practice, it takes as input the numerical values associated with each class output by the network and transforms them into probabilities using Softmax. It then checks whether the class with the highest probability matches the label of the sample being processed; if it does not, this means the network is making a mistake, and it is therefore forced to adjust its weights in a certain direction [41].

The optimization algorithm used to adjust the network weights was the Adaptive Moment Estimation (Adam) [66]. The unique feature of this algorithm is that, instead of using second-order information such as the Hessian matrix, it approximates this information through the simpler calculation of first-order information alone, namely the gradients. With this information, it is possible to adjust the weights with varying intensity. The variance is calculated, and if a parameter's gradient is changing too rapidly, it is limited by imposing small updates; conversely, if it is changing too little, it indicates that the algorithm is in a flat region, so larger steps are taken to exit this flat zone.

4.4.2 *Evaluation metrics*

There are several metrics used to evaluate the quality of a model. The main ones are listed and explained below. The metrics will be divided into two broad categories: performance metrics and computational efficiency metrics.

Performance Metrics

This first category of metrics is used to assess how well the network is able to correctly assign the right label to a given frame window. To obtain these metrics, the confusion matrix [67] must be constructed: this is a tabular representation that compares the ground truth labels with the model's predictions. For each class, it

categorizes the outcomes into correct predictions (True Positives, TP, and True Negatives, TN) and misclassifications (False Positives, FP, and False Negatives, FN)

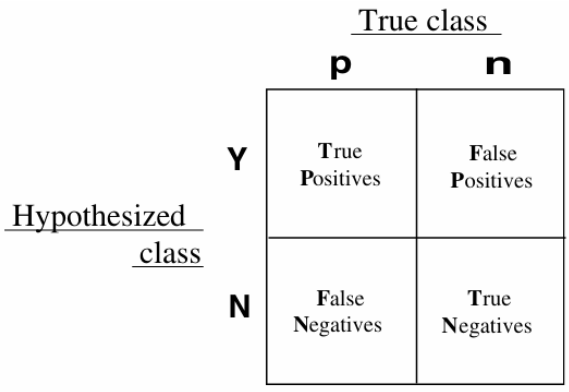

Figure 4.19. Confusion Matrix from [67]

Based on these four parameters, it is possible to derive the most relevant performance metrics.

Accuracy: it refers to the ratio of the number of correct predictions to the total number of samples analyzed:

$$Accuracy = \frac{\sum_i TP_i}{\sum_i (TP_i + TN_i + FP_i + FN_i)} \tag{34}$$

High accuracy rate reflects the overall reliability of the model across al classes.

Precision: it indicates the ratio of the number of correct predictions for a class to the total number of predictions made for that class:

$$Precision = \frac{TP}{TP + FP} \tag{35}$$

High precision helps reduce the number of false positives.

Recall: it indicates the ratio of the number of correct predictions for a class to the total number of samples in that class:

$$Recall = \frac{TP}{TP + FN} \tag{36}$$

A high recall rate helps reduce the number of false negatives.

F1-Score: a metric that serves as a single indicator for finding the right balance between precision and recall:

$$F1 = 2\frac{Precision \times Recall}{Precision + Recall} \tag{37}$$

This metric, calculated as a harmonic mean, is useful because it penalizes any imbalances in the data. If a model had very high recall but low precision, the F1 score would plummet. Consequently, a high F1 score ensures that the model performs well on both measures. It is therefore considered the gold standard for evaluating models on datasets, even those that are not perfectly balanced [68].

Efficiency Metrics

This second type of metric measures the model's computational footprint at the hardware level. In fact, running a neural network on an embedded device such as the one used in this thesis involves stringent constraints, making hardware efficiency just as important a factor as accuracy [68].

Memory Footprint: it indicates the network's impact in terms of computational resources and memory; specifically how much memory space (in bytes) the network requires to store its parameters and weights;

Latency: it indicates the time between the acquisition of the input data and the network's return of the prediction;

FLOPs: it indicates the number of floating-point operations a processor must perform to carry out an inference operation, that is, to classify new data using the weights and structures obtained through prior training. It is important to monitor this metric, as latency and power consumption are closely tied to the number of operations required [69].

4.5 **Models Adaptation for Hardware Execution**

4.5.1 ***Quantization***

The implemented neural networks operate in 32-bit floating-point (FP32). The use of floating-point notation ensures numerical stability and precision and is therefore used for training; however, it can prove problematic when deploying the network on embedded devices, where minimizing memory usage and power consumption is essential. To transition from training, which uses 32-bit floating-point numbers, to deployment on a device and uses the 8-bit integer (INT8) numerical format, quantization is necessary to ensure significant memory and energy savings. This process can potentially lead to a slight decrease in model accuracy [70], but there are advanced techniques that often allow the network to preserve the original performance levels.

Quantization allows a continuous range of values to be mapped to a discrete set. In PyTorch, which is the most widely used framework for developing and training neural networks, the post-training quantization (PTQ) process is divided into two techniques: static and dynamic [71].

It should be noted, however, that the architectures developed already had a low memory footprint in FP32, relative to the available memory on the Raspberry Pi. The application of quantization, therefore, was not dictated by a lack of space on the device, but by the opportunity to gain other advantages. Among these, it is worth noting that by reducing the size of the models, it is possible to store all weights directly within the cache, which would reduce latency caused by potential cache misses and the resulting RAM accesses. Furthermore, the transition from FP32 to qint8 would allow the ALU to perform calculations faster and with lower power consumption, without using floating-point registers [72]. This simplification of calculations is also of great help considering that the device must handle both signal acquisition and signal processing.

Static PTQ

In static quantization, both weights and activations are converted right from the start. To know how to correctly convert numbers from floating-point to integers, the system must first undergo a calibration phase, during which the maximum and minimum values of the activations for each layer are observed. This approach, however, presents a major problem. Suppose that during the calibration phase, activations fluctuate between -x and +x and within this range the 256 intervals required for 8-bit quantization are defined. If, after calibration, the fluctuations take on values that exceed this range, the system would map them to one of the two extreme values by performing the so-called “clipping”. This phenomenon would destroy potentially relevant information, especially in RNNs, where this type of error would be propagated and amplified at every time step [73]. For this reason, PyTorch prevents the application of static quantization to recurrent layers.

Dynamic PTQ

This approach instead is usually preferred, as quantization is applied post-training, only to the network weights, which at this point are fixed values. The activations, on

the other hand, remain in floating-point format, and when the product with the integer weights needs to be computed, their scaling factors are not fixed a priori but are calculated dynamically, adapting to the data actually flowing through the network at that moment.

For fully recurrent architectures, it has been used PyTorchFX, a tool that transforms models into logical graphs and then converts FP32 operations into their quantized versions. Subsequently, to ensure compatibility with the Raspberry Pi 4, the quantized model was converted into the industry-standard format known as Open Neural Network Exchange (ONNX) [74].

Quantizing the hybrid model, PointConv LSTM, proved to be a more complex challenge. The model performs complex operations, such as extracting the farthest points within the point cloud, and then applies KNN for dynamic clustering. PyTorchFX is unable to trace graphs in cases of complex control flows or dynamic indexing, effectively making it impossible to use for network quantization.
It was therefore decided to directly quantize the model within the ONNX Runtime ecosystem, implementing certain workarounds.

First, in the function responsible for identifying the farthest points within the point cloud, ‘distance’ tensor was taken, and a dynamic mask was calculated. Based on this mask, only certain values within the ‘distance’ tensor were replaced. This in-place tensor modification operation cannot be represented by ONNX as a deterministic node. The solution is to replace this instruction with the creation of a new vector (which overwrites the previous `distance`) instead of modifying the previous one.

A second issue arises due to the padding applied to frame sequences shorter than 60 frames. The key point is that the model must recognize that these padding frames should not be considered when taking the final decision. To achieve this, a mechanism was implemented that ensured the LSTM layer considered the frame sequence only up to the last valid frame.

```
last_valid_indices = seq_lengths - 1
final_features = lstm_out[batch_indices, last_valid_indices, :]
out = self.fc(final_features)
```

In other words, the `last_valid_indices` variable was created to keep track of the last valid index. This dynamic indexing technique is not easily handled by ONNX, and therefore generates a chaotic subgraph that significantly slows down inference. Hence, it is preferable to use the `gather` function, which is analogous to the previous instruction but is better supported by ONNX.

```
# Calculation of the index of the last frame [Batch, 1, 1]
last_indices = (seq_lengths - 1).view(-1, 1, 1)

# Expansion of the last index to indicate it to all the 256 features
# [Batch, 1, 256]
last_indices_expanded = last_indices.expand(-1, 1, lstm_out.size(-1))

# Gather function to extract those values
final_features = torch.gather(lstm_out, dim=1,
index=last_indices_expanded).squeeze(1)
```

Since it is a low-level operation, `gather` does not accept generic commands such as the colon but it requires an explicit indexing matrix (an unwrapped 3D tensor) that acts as a "coordinate map" for each individual feature to be extracted. The slightly increased complexity of the PyTorch code is rewarded by the drastically faster inference implementation on the Raspberry Pi.
Finally, ONNX uses the Operational Set (Opset) version 16, a library used to map PyTorch's native instructions into ONNX-compatible nodes.

4.5.2 ***Real Time***

To ensure the device operates in real time, an event-driven Python script was implemented to handle data acquisition, processing, and classification. The Message

Queuing Telemetry Transport (MQTT) protocol was used to enable communication between the sensor and the script [75], [76].

The sensor acts as the so-called Publisher, meaning it continuously transmits data packets to the central node, also known as the MQTT Broker, located on the Raspberry Pi. The inference script, on the other hand, acts as a Subscriber or Client, that is, the entity that reads the data. The MQTT protocol divides communication into channels called Topics. The MQTT client has been configured to listen for a specific Topic, the one dedicated to communicating point cloud information. All other traffic is therefore ignored. The client remains in passive listening mode on that Topic, waiting for a signal. When the Broker detects an incoming packet on the Topic of interest, the MQTT library wakes up the client by triggering a callback function *(on_message),* which receives the payload and initiates the processing chain.

```
def on_message_wrapper(msg):
        classificator.message_processing(msg.payload.decode())

    # Client initialization
    client = mqtt.Client()
    client.on_connect = on_connect

    # Callback function
    client.on_message = on_message_wrapper

    try:
      # Broker connection
      client.connect(MQTT_BROKER, MQTT_PORT, 60)
        client.loop_forever()
    except KeyboardInterrupt:
        client.disconnect()
        sys.exit(0)
```

Within the processing pipeline, the dataset undergoes an initial preprocessing phase. First, a spatial filter is applied to remove any outliers. Only the points that fall within the area of interest are retained. Data augmentation is then applied to ensure consistency in the number of points relative to the training dataset. Finally, the point cloud is decomposed into two ellipsoids to extract the nine features of interest used

to train the models. These features are normalized using constant values for mean and standard deviation calculated from the training set.

The processed frames are placed into a circular buffer (a First-In-First-Out (FIFO) queue) with a capacity of sixty frames. This value was chosen to maintain consistency with the conditions of the training phase, where the network was trained by observing time windows of sixty frames divided into twelve batches. The neural network is not activated until the buffer contains exactly sixty frames. From this point on, the system will adopt a sliding window. In this way, the buffer will always contain only the last six seconds of acquired data, and based on this, it will perform the inference phase continuously and in real time.

```
# Adding the normalized frame to the FIFO queue
self.frame_buffer.append(normalized_frame)

# Inference only when the time window is full of 60 frames
if len(self.frame_buffer) == MAX_FRAMES:

    raw_data = np.array(self.frame_buffer, dtype=np.float32)

    # Reshaping: [1, 12, 45]
    input_data = raw_data.reshape(1, NUM_WINDOWS, FRAMES_PER_WINDOW *
FEATURES_PER_FRAME)

    # Inference
    logits = self.session.run(None, {self.input_name:
input_data})[0][0]
```

The inference output will be in the form of scores (logits), which are converted into probability values using the Softmax operator. Despite the high performances of the implemented networks, errors may occur in real time due to sensor noise. Relying solely on the instantaneous classification performed every millisecond by the network would lead to flickering, resulting in a perceived lack of stability of the system.

To address this, a temporal smoothing mechanism to improve consistency of the final output has been implemented. The predictions generated by the network in each frame are stored in a second buffer with a capacity of fifteen steps. To filter out unreliable classifications, only those predictions that have achieved a confidence score greater than a threshold value of 0.60 are included in this buffer. The system then determines the final output by identifying the most frequent label within the buffer, provided it meets a minimum occurrence constraint: a specific posture must appear at least ten times out of the fifteen stored frames to be confirmed as the system's output.

Finally, it has been observed that when a subject remains completely still, such as when standing motionless or in a crouched position, the number of reflections, as previously mentioned, drops dramatically. This radar “blindness” deprives the network of the information needed to classify these situations, leading it to make the prediction closest to this scenario, namely, a seated person, since this posture is the most static among those used during training, and thus the one with the fewest features. To prevent this error in static conditions, at least the confusion between a stationary standing person and a stationary seated person, a threshold mechanism has been implemented by observing the center of gravity height and the number of reflections received, both averaged over a sliding window of ten frames. When a person remains stationary while standing, they generate few reflections, but those will have a higher average height compared to a seated person. Therefore, if the last confirmed posture was Stand, the number of reflections received falls below a preset threshold (fifty points), but the subject’s average height remains consistent with an upright posture, that is, above a certain threshold, then the network’s prediction is overruled by imposing a safety block. The “stand” posture is confirmed in the output, while the fifteen frames buffer is cleared of incorrect “sit” predictions to prevent them from affecting future predictions.

CHAPTER 5
EXPERIMENTAL EVALUATIONS

5.1 **Model analysis**

This chapter presents and analyzes the results obtained from the various architectures developed and tested during this thesis. The evaluation of network performance does not focus solely on classification accuracy; networks will also be assessed in terms of memory usage and computational latency. Essentially, five networks were evaluated:

- RNN: reference model implemented in [21]. The network is characterized by a single recurrent layer;
- PointConv: model that analyzes spatial configuration, focusing on the application of convolution to the point cloud. It requires no memory and no inter-frame processing;
- PointConv LSTM: an iterative evolution of the previous architecture. The information extracted by the simple PointConv is processed by an LSTM layer to capture temporal dynamics;
- Accuracy-Oriented Network: optimal architecture in terms of accuracy achieved through the NAS process;
- Latency/Memory-Oriented Network: architecture optimized for inference latency and memory occupation following the NAS process.

Regarding the datasets used to train and validate the various models, the following were used:

1- UniPV Dataset: proprietary dataset collected in the laboratories of the University, to which the preprocessing and data augmentation techniques mentioned in this thesis were applied;
2- Public MM-DCDR dataset [54].

5.1.1 *Networks Performances*

RNN

The first step involved testing a simple architecture, which consisted of a simple recurrent layer. The architecture was tested on the UniPV dataset, and the preprocessing technique applied involved constructing a single ellipsoid around the point cloud. This first step was necessary to validate the performance of this network implemented in my previous work on the newly collected data. In fact, thanks to the improvements made to the signal acquisition and preprocessing algorithm, the point cloud was more accurate and better followed the person's silhouette.

AI Model	**Accuracy (%)**				
	50-50	**60-40**	**70-30**	**80-20**	**90-10**
LSTM	91.46	90.78	90.05	89.62	89.67
Bi-LSTM	90.81	90.56	90.42	89.97	89.29
Projected LSTM	90.42	90.46	89.54	89.58	89.48
GRU	90.90	89.76	90.10	89.81	89.80

Table 5.1. RNN accuracy results without transient

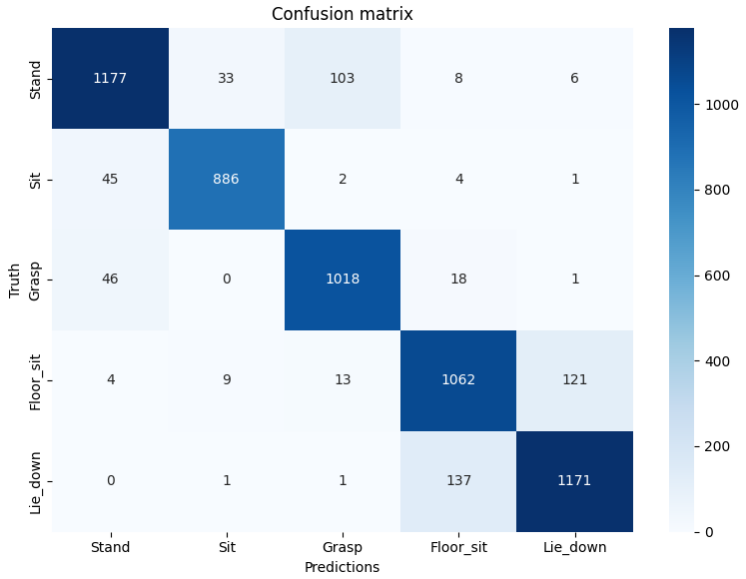

Figure 5.1. Split 80-20 Confusion Matrix

	LSTM	**Bi-LSTM**	**Projected-LSTM**	**GRU**
Precision (%)	90	91	90	90
Recall (%)	90	91	90	90
F1-Score (%)	90	91	90	90

Table 5.2. RNN average metrics

The accuracy results obtained by varying the splitting percentages of the dataset are shown in Table 5.1. The accuracy values range between 89% and 91%, which is promising. It is worth noting that these results were obtained by training the network using only frames corresponding to the five well defined postures (Stand, Sit, Grasp, Floor_Sit, Lie_Down). Transitional frames were intentionally excluded. It is reasonable to assume that including a sixth posture would certainly lead to a deterioration in performance. This was done initially to ensure complete continuity with my previous work, since transitional frames had never been considered in it.

Compared to the values obtained in [21], however, the network showed a slight declines, but this is due also to the different point cloud that the network must process.

From the confusion matrix shown in Figure 5.1, it can be observed that the network struggles to distinguish certain postures that it considers similar, such as Floor_sit and Lie_down, and Grasp and Stand.

PointConv

The second step involved evaluating the PointConv architecture. which is a purely spatial network. To validate the model's generalization ability, it was trained and tested on both benchmark datasets (Tables 5.3 and 5.4).

	90-10	**80-20**	**70-30**	**60-40**	**50-50**
Accuracy (%)	90.03	87.51	88.81	87.43	87.02
Precision (%)	90	88	89	88	88
Recall (%)	90	88	89	88	88
F1-Score (%)	90	88	89	88	87

Table 5.3. PointConv results on the UniPV Dataset without transient

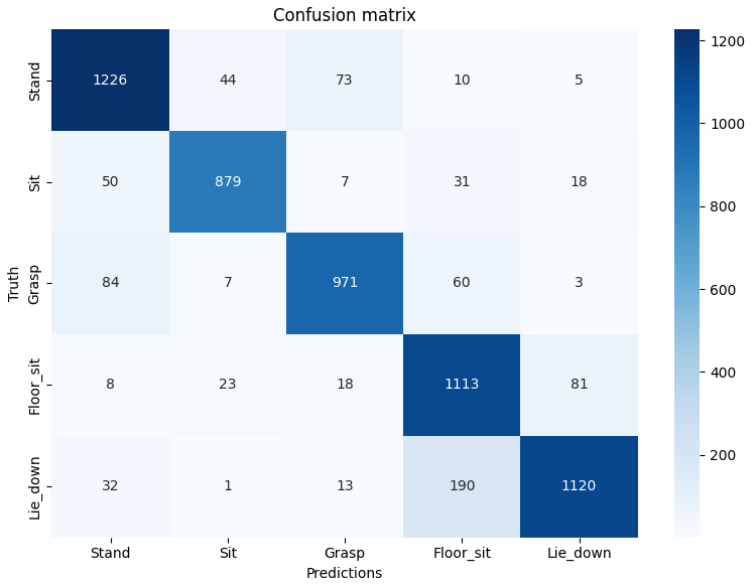

Figure 5.2. Split 80-20 Confusion Matrix, UniPV Dataset

	90-10	80-20	70-30	60-40	50-50
Accuracy (%)	95.19	94.49	93.31	92.42	90.80
Precision (%)	95	94	93	93	91
Recall (%)	95	95	93	93	91
F1-Score (%)	95	94	93	92	91

Table 5.4. PointConv results on MM-DCDR Dataset

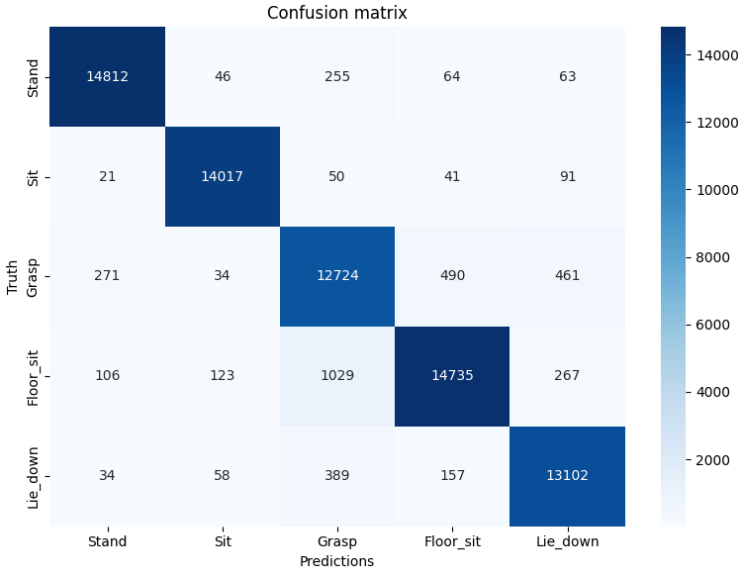

Figure 5.3. Split 80-20 Confusion Matrix, MM-DCDR Dataset

On the UniPV dataset (Table 5.3), the network achieves a peak accuracy of 90.03% in the 90-10 configuration. While the accuracy is decent, it does not reach excellent values, although it should be noted that these results were obtained by including transients and are therefore better than those of the simple RNN. In contrast, the results obtained on the MM-DCDR dataset (Table 5.4) show an improvement of 4–5 percentage points, reaching 95.19%. This difference stems from the fact that the public dataset features more standardized point clouds, acquired with a different setup, which allows the network to perform better. Furthermore, since this network works frame by frame, the absence of temporal information in the proprietary dataset, where the movement of the point cloud is more visible and significant, is another reason for the difference in performance.

PointConv LSTM

To overcome the lack of long-term memory and temporal information, the architecture was enhanced by adding a recurrent layer. Three different tests were conducted to validate this architecture.

An initial test involved training the network on the proprietary dataset while excluding frames containing transients. The excellent results shown in Table 5.5 demonstrate a clear improvement achieved by this approach.

	90-10	**80-20**	**70-30**	**60-40**	**50-50**
Accuracy (%)	99.45	99.34	98.24	98.11	93.42
Precision (%)	1	1	0.98	98	94
Recall (%)	99	99	98	98	93
F1-Score (%)	99	99	98	98	93

Table 5.5. PointConv LSTM results on the UniPV Dataset without transient

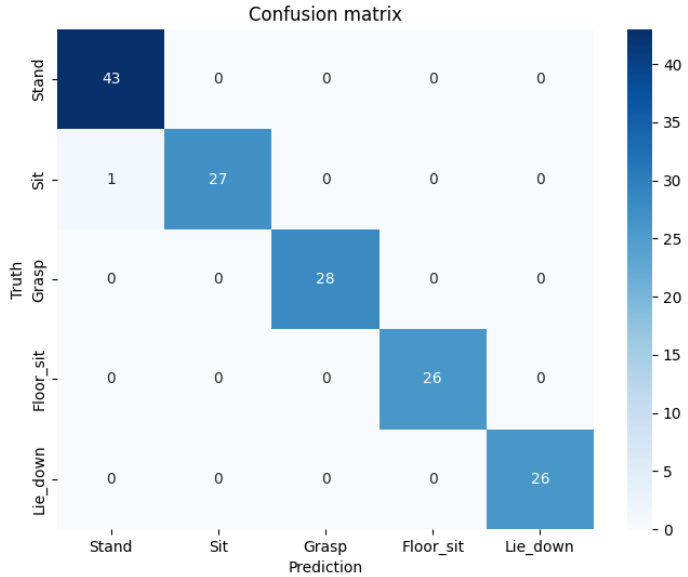

Figure 5.4. Split 80-20 Confusion Matrix, UniPV Dataset without transient

The combination of the geometric extractor with a recurrent layer enables the network to achieve results that border on perfection, confirming that, when applied to well-defined postures, the network is capable of excellent classification.

When transients are also included, the complexity of the problem increases and the accuracy decreases by about seven percentage points, as shown in Table 5.6.

	90-10	**80-20**	**70-30**	**60-40**	**50-50**
Accuracy (%)	93.59	92.89	91.32	91.28	92.59
Precision (%)	94	91	91	91	93
Recall (%)	93	93	91	91	94
F1-Score (%)	94	92	91	91	93

Table 5.6. PointConv LSTM results on the UniPV Dataset with transient

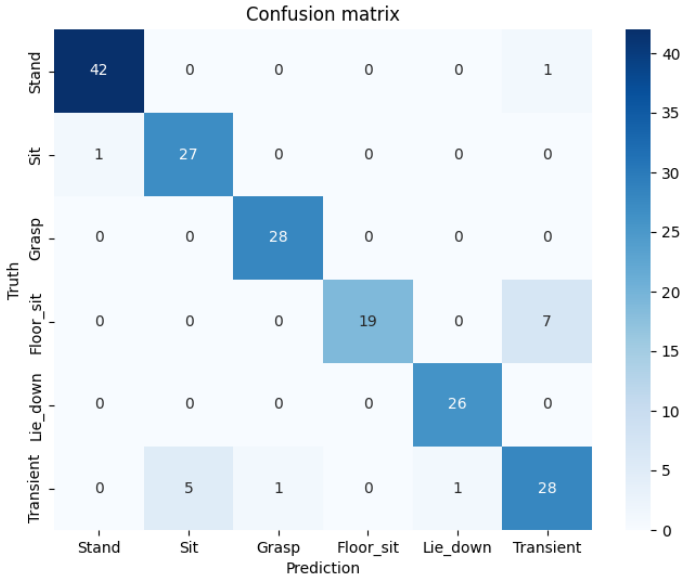

Figure 5.5. Split 80-20 Confusion Matrix, Proprietary Dataset with transient

As expected, introducing a posture that shares spatial features with the starting and ending postures introduces a degree of uncertainty but the results remain still good and are better compared with the simpler PointConv without transient.

Finally, the third and final step involved testing the model on the public dataset.

	90-10	80-20	70-30	60-40	50-50
Accuracy (%)	97.22	96.16	96.19	95.47	93.95
Precision (%)	97	96	96	95	94
Recall (%)	97	96	96	95	94
F1-Score (%)	97	96	96	95	94

Table 5.7. PointConv LSTM results on MM-DCDR dataset

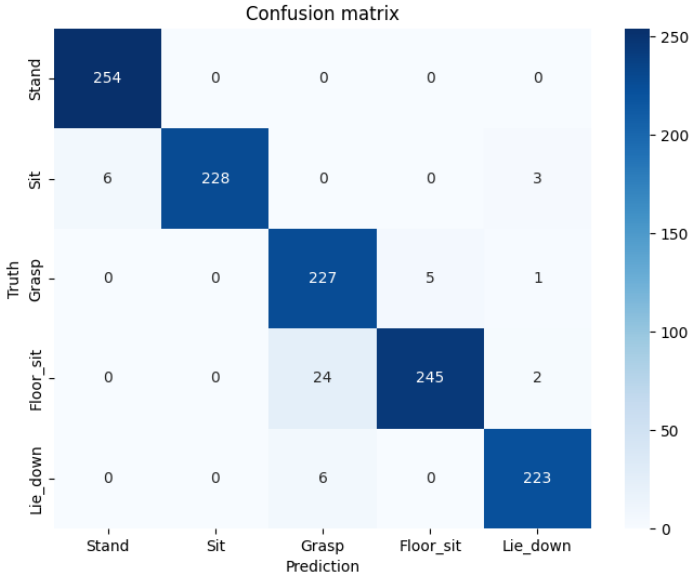

Figure 5.6. Split 80-20 Confusion Matrix, MM-DCDR Dataset

The results shown in Table 5.7 reveal a peak in the 90-10 split, reaching 97.22%, compared to the 95.19% achieved with PointConv alone, demonstrating once again the value of including the recurrent layer.

Accuracy and Latency/Memory Oriented Networks

Finally, the last networks evaluated are the two best architectures identified through the NAS process. The training and inference of these two networks were

conducted by implementing both the preprocessing model based on a single ellipsoid and the one based on two ellipsoids.

Before analyzing the networks' performance, it is necessary to discuss the benefits of switching from the single-ellipsoid model to the two-ellipsoid one. Through the analysis of Permutation Feature Importance, it was noted that the single ellipsoid model had a structural weakness. The network, in fact, depended predominantly on the height of the ellipsoid's center of mass. Any perturbation of this variable causes a drop in accuracy of approximately 40%. The remaining features play a marginal role, making the system fragile.

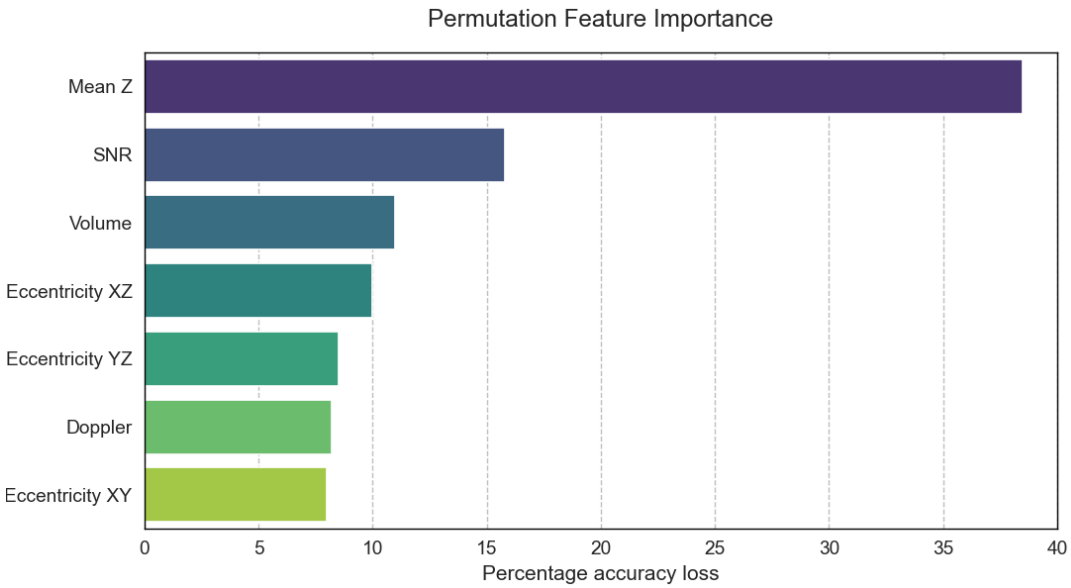

Figure 5.7. PFI in the case of single-ellipsoid model

The introduction of the two-ellipsoid technique resolved this imbalance. Dividing the point cloud into two ellipsoids and subsequently extracting different features forced the networks to distribute their attention across a greater number of features. The PFI analysis using the two-ellipsoid model showed that no single feature plays a dominant role over the others.

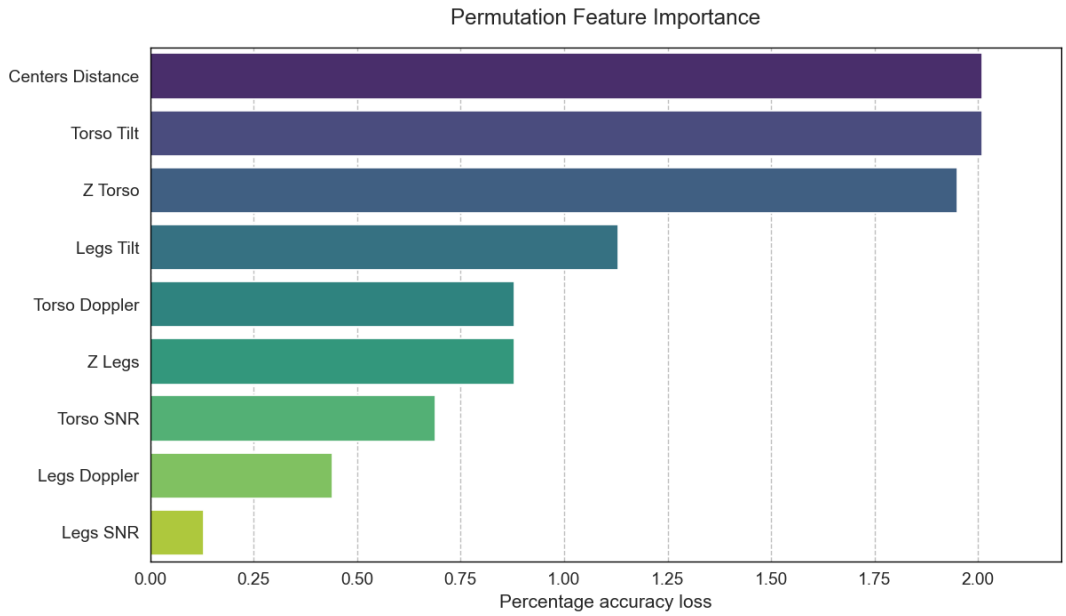

Figure 5.8. PFI in the case of two-ellipsoid model

The most influential feature is *centers distance*, followed by *torso tilt* and *z torso*, but altering any of these parameters results in a maximum loss of about 2%.

The models whose results will now be discussed, evaluated using the two-ellipsoid technique on the proprietary dataset, achieved an overall accuracy approximately 3 percentage points higher than the single-ellipsoid model.

The Accuracy-Oriented Network is the model that achieved the highest performance for this specific problem on the proprietary dataset.

	90-10	**80-20**	**70-30**	**60-40**	**50-50**
Accuracy (%)	96,5	96,14	94.35	93.90	93.34
Precision (%)	95	95	92	92	93
Recall (%)	95	93	92	92	93
F1-Score (%)	95	94	92	92	93

Table 5.8. Accuracy-Oriented Network results

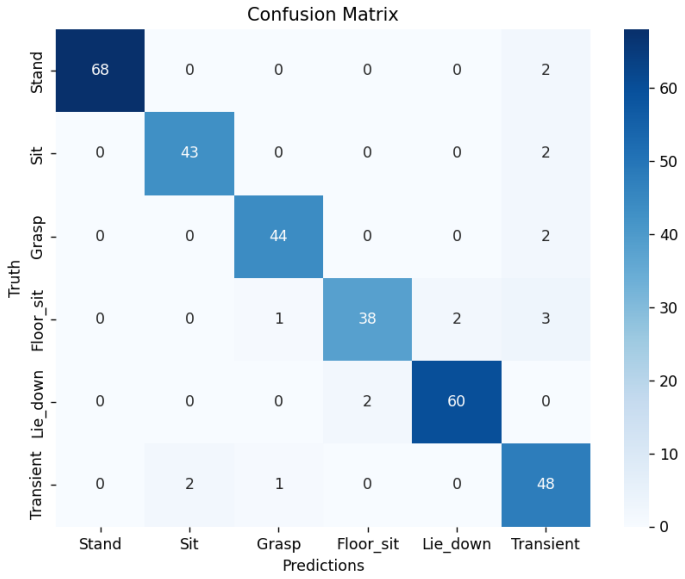

Figure 5.9. Split 80-20 Confusion Matrix

The results confirm the effectiveness of this architecture. Thanks to the highly informative input, the network is able to accurately map even transients, which have so far been the most difficult class to identify.

	90-10	**80-20**	**70-30**	**60-40**	**50-50**
Accuracy (%)	94.45	93.89	92.15	92.35	91.58
Precision (%)	93	93	92	92	90
Recall (%)	94	93	91	91	91
F1-Score (%)	94	93	91	91	90

Table 5.9. Latency/Memory-Oriented Network results

The Latency/Memory architecture delivered excellent performance, falling just a few percentage points short of the Accuracy-Oriented Network.

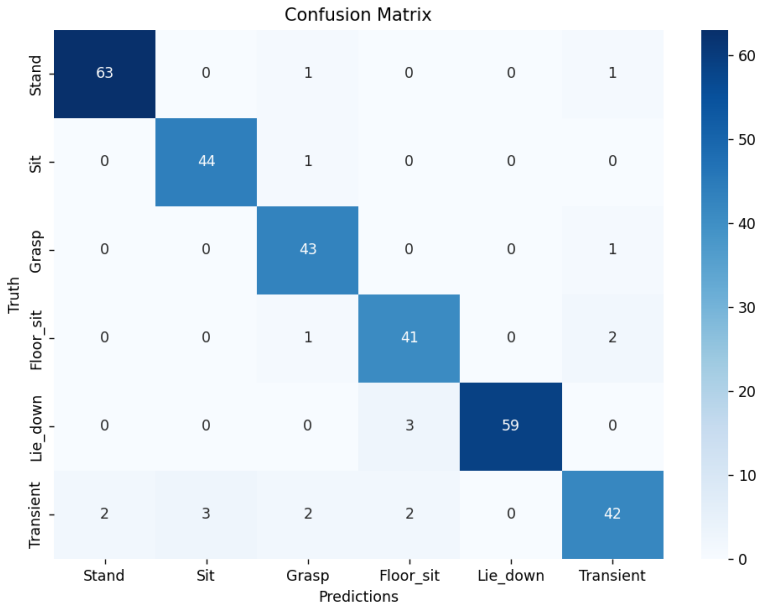

Figure 5.10. Split 80-20 Confusion Matrix

Computational Complexity

To assess the feasibility of deploying these networks on embedded systems, it is essential to analyze the computational requirements of the various architectures tested. As summarized in Table 5.10, the evaluation was conducted using two main metrics: memory footprint, that is, memory usage, and computational cost measured in Millions of Floating-Point Operations per Second (MFLOPs), which directly impacts inference latency and power consumption.

	Memory Footprint (MB)	**Computational Cost (MFLOPs)**
RNN	0.1135	0.0445
PointConv	19.047	6.4
PointConv LSTM	19.798	97.46
Accuracy-Oriented	2.2107	6.8186
Latency/Memory-Oriented	0.0987	0.2656

Table 5.10. Comparison between Networks Computational Complexity

A review of the collected data clearly shows that models based on spatial architectures require significant resources. The PointConv network requires over 19 MB of memory to run, as well as 6.4 MFLOPs. Adding the temporal layer to the PointConv LSTM leaves the memory footprint nearly unchanged but drastically increases the computational cost, reaching 97.46 MFLOPs. This is caused by the network's unrolling which must perform the LSTM's forward pass iteratively for each of the 12 input time windows.

Significantly lower values were obtained by testing the purely recursive architectures. RNN has negligible requirements but results in lower accuracy compared to the other architectures, as shown in Tables 5.1 and 5.2. The Accuracy-Oriented Network manages to reduce the memory footprint to 2.21 MB while maintaining a high MFLOPs value of 6.8, which is comparable to that achieved by PointConv.
The architecture that best optimizes both parameters without a drastic drop in accuracy is the Latency/Memory-Oriented Network which guarantees the lowest Memory Footprint and a value of MFLOPs comparable to the RNN. The use of the two-ellipsoid model was particularly useful for this architecture. In fact, the network has a small number of trainable parameters to keep latency and memory usage low, so it lacks the computational capacity needed to extract particularly complex features from the input. This characteristic of the network can be compensated by providing at the input a feature vector that is already rich in meaning.

5.1.2 *Post Training Quantization Performances*

The PTQ was applied to the most promising models validated in the previous chapter:

- Accuracy Oriented Network (AON)
- Latency/Memory Oriented Network (L/MON)
- PointConv LSTM (PCVLSTM)

This conversion significantly reduces the memory footprint and speeds up inference, but at the same time introduces quantization errors that can negatively affect the model's predictive quality. Table 5.11 shows the results of this process, which will be discussed below.

Networks	**Pre Quantization**		**Post Quantization**	
	Memory Footprint (MB)	**Accuracy (%)**	**Memory Footprint (MB)**	**Accuracy (%)**
AON	2.2	96.14	0.55	95.82
L/MON	0.9	93.89	0.027	91.64
PCVLSTM	20	92.89	5	82.34

Table 5.11. Quantization Results

The first architectures to be quantized were the two optimal networks identified using NAS. Quantizing the Accuracy-Oriented Network reduced the memory footprint from 2.2 MB to 0.55 MB. Despite this reduction, the impact on accuracy was negligible: it dropped from 96.14% to 95.82%.

The Latency/Memory-Oriented Network, which is already lightweight by nature, was quantized, and its memory footprint decreased from 0.9 MB to 0.027 MB. Its predictive capability was most affected by the quantization, dropping from 93.80% to 91.64%.

While these networks have demonstrated a certain degree of robustness even in the face of quantization, the same cannot be said for the PointConv LSTM network. In fact, the network underwent significant compression, shrinking from a 20 MB footprint to 5 MB. However, accuracy dropped dramatically, falling from approximately 93% to 82%, a loss of about 11 percentage points. The reasons for this collapse lie in the fact that the operation of this network relies heavily on two fundamental algorithms: FPS and KNN. FPS requires calculating the distance

between various points. These calculations are now performed with lower precision since the INT8 format is used. Consequently, floating-point values undergo drastic truncation. Working in this 8-bit compressed space, two points that are just a few centimeters apart will end up having the same quantized coordinates. Consequently, the distance calculation will return a zero value or, at the very least, a distorted one. The algorithm will fail to identify centroids uniformly within the point cloud, concentrating them in noisy areas and leaving entire parts of the body uncovered.
Once the centroids are calculated incorrectly, the network applies KNN to create clusters, to which convolutional filters are then applied. Here, too, cluster formation is based on calculating the distances between points. Due to 8-bit quantization error, these clusters are not formed correctly.

5.1.3 *Device Performance*

After validating the accuracy of the models and compressing the weights through quantization, the final step involved running them in real time on various devices to evaluate their inference speed.

Device	**CPU**	**Average Inference Time (ms)**
Raspberry Pi 4	Quad-core Cortex-A72 (1.5 GHz)	142.71
STM32MP257F-EV1	Dual-core Cortex-A35 (1.5 GHz)	275.95
STM32MP157F-DK2	Dual-core Cortex-A7 (800 MHz)	665.96
STM32MP135F-DK	Single-core Cortex-A7 (1.0 GHz)	885.36

Table 5.12. Average Inference Time for the PCVLSTM

To do this, the average inference time for the different architectures is calculated. The tests were conducted on various devices, including both the device available in the laboratory and the boards provided by STMicroelectronics.

The first test focused on the PointConv LSTM architecture. Despite quantization, the network still must perform several convolutions and subsequently handle the temporal unrolling of the recurrent cell. As a result, this places a significant workload on the ALU of embedded processors.

An analysis of Table 5.12 highlights the resource-intensive nature of this architecture. The best inference time was achieved on the Raspberry Pi 4. This is acceptable given the real-time requirements of the application, but the computational load can cause the device to overheat quickly, thus resulting in suboptimal power consumption if the device is battery-powered. Moving the model to STM processors reveals further limitations. The STM32MP257-EV1 manages to keep the time under 300 ms, while Cortex-A7-based boards exhibit significantly high latency, which is not optimal for a real-time solution. It is important to note that this is only the time required for the network to make the prediction; it does not account for the time needed to extract features, and one must also consider the additional delay caused by the buffering solution used to strengthen the result in order to make the final decision.

The situation changes when analyzing architectures implemented using NAS. Although the Accuracy Oriented Network is complex, it demonstrates remarkable responsiveness on all the hardware tested (Table 5.13).

Once again, the Raspberry Pi 4 proves to be the fastest device. Inference also yields good results on the STM boards.

Device	CPU	Average Inference Time (ms)
Raspberry Pi 4	Quad-core Cortex-A72 (1.5 GHz)	3.55
STM32MP257F-EV1	Dual-core Cortex-A35 (1.5 GHz)	4.12
STM32MP157F-DK2	Dual-core Cortex-A7 (800 MHz)	11.75
STM32MP135F-DK	Single-core Cortex-A7 (1.0 GHz)	13.37

Table 5.13. Average Inference Time for the Accuracy-Oriented Network

Finally, the performance of the Latency/Memory-Oriented Network was evaluated. The network is specifically optimized for latency, and the tests confirm its ultra-light nature (Table 5.14).

Device	CPU	Average Inference Time (ms)
STM32MP257F-EV1	Dual-core Cortex-A35 (1.5 GHz)	0.58
Raspberry Pi 4	Quad-core Cortex-A72 (1.5 GHz)	1.08
STM32MP135F-DK	Single-core Cortex-A7 (1.0 GHz)	1.61
STM32MP157F-DK2	Dual-core Cortex-A7 (800 MHz)	1.90

Table 5.14. Average Inference Time for the Latency/Memory-Oriented Network

This network achieves near-instantaneous inference speeds. On the STM32MP257-EV1 board's processor, inference times fall below one millisecond, outperforming the Raspberry Pi 4 for the first time. It is also worth noting that even on processors

that have so far proven to be significantly slower, inference times are still around 1–2 ms.

One final interesting comparison that can be made between these boards concerns energy consumption.

Device	Idle Power Consumption	Running Power Consumption
STM32MP257F-EV1 [77]	221 mW	582 mW
STM32MP135F-DK [78]	455 mW	653 mW
STM32MP157F-DK2 [79]	710 mW	790 mW
Raspberry Pi 4	2.7 W	6.4 W

Table 5.15. Boards Power Consumption

As can be seen from the data in Table 5.15, the power consumption analysis reveals a significant difference among the devices under consideration. The solutions offered by STMicroelectronics are specifically designed for embedded applications and, in fact, maintain low power consumption in the range of a few hundred milliwatts. In particular, the STM32MP257F-EV1 board stands out for its maximum efficiency, with only 221 mW at idle and, more importantly, just 582 mW at full load. In contrast, the Raspberry Pi 4, being a general-purpose board rather than one designed specifically for embedded solutions, exhibits significantly higher power consumption, ranging from 2.7 W at idle up to 6.4 W at full load.

5.1.4 *Discussion*

To optimize the AI models implemented for deployment, quantization was applied, allowing to switch from working with FP32 floating-point numbers to the INT8 integer format. Although static quantization is the standard in these contexts, empirical tests have shown that it cannot be used. In fact, for it to be successfully

applied, a calibration dataset is required on which to run the network to calculate and freeze the maximum and minimum activation ranges of each individual layer. The problem is that the activations generated by processing the point cloud have a wide and unpredictable dynamic range; consequently, forcing the activations within static calibration intervals would have led to approximation errors, known as clipping, which, in the case of recurrent networks, would progressively amplify, leading to a collapse in accuracy. This characteristic of the data clearly explains why PyTorch does not provide support for static quantization on recurrent layers, demonstrating that dynamic quantization is not an option, but a necessity for preserving the model's robustness.

Furthermore, the implementation testing phase using STMicroelectronics Edge AI Developer Cloud platform revealed some architectural limitations; in fact, it was not possible to implement recurrent networks MCUs. This is because MCUs are optimized for static architectures (such as classic CNNs) and currently lack the native support required to execute recurrent layers. This highlights the need to rely on advanced platforms such as MPUs or devices equipped with dedicated NPUs to fully support the complexity of processing unstructured point clouds and recurrent architectures.

Despite the challenges encountered in implementing on-device AI models, this thesis introduces several significant contributions in the fields of HAR and AAL.

First, an innovative geometric preprocessing method was developed that enriches the information content of the point cloud. In the literature, this representation of radar data is considered to be uninformative, prompting the scientific community to adopt solutions that are more complex to process. The approach proposed here, however, demonstrates that it is possible to extract robust spatial features while maintaining excellent computational efficiency.

More generally, the combination of mmWave radar technology with the evolution of AI approaches already present in literature has enabled high accuracy in posture detection. This result makes it possible to create a completely non-invasive solution that overcomes the psychological stigma, physical discomfort, and lack of privacy, whether real or perceived, associated with devices currently on the market. Consequently, the entire solution developed in this study demonstrates concrete and solid potential for future industrial and commercial applications.

CHAPTER 6
CONCLUSIONS AND FUTURE WORKS

This thesis project concerned the design and validation of a postural recognition system based on millimeter-wave radar technology exploiting and optimizing artificial intelligence models. The choice of radar technology was necessary to enable monitoring in home environments ensuring the privacy of the subjects.

The different algorithms evaluated have enabled the development of an excellent postural classifier suitable for embedded solutions. The system is based on an efficient pipeline that starts with data augmentation, followed by data preprocessing techniques, with the two-ellipsoid model emerging as the optimal approach. In fact, this model has eliminated the reliance on the point cloud's center of mass by decomposing it into two distinct volumes, which enabled the extraction of a feature vector that is richer in information and more balanced, thereby avoiding the issue of feature polarization, a deeply concerning problem when dealing with individuals whose height deviates from the average.

The use of NAS proved crucial in identifying several efficient architectures, such as the Accuracy-Oriented Network and the Latency/Memory-Oriented Network. These architectures were implemented and compared with well-known architectures in the literature that are widely used in this context, such as the Point Convolution and the Point Convolution LSTM. Furthermore, to address the constraints associated with edge computing, the networks underwent a post-training quantization process to compress the weights into the INT8 format. During this process, it became apparent that the quantized version of the previous mentioned architecture described in the literature are more computationally intensive than even the non-quantized version of the networks identified using NAS, and moreover, a significant decline in predictive

performance was observed during the quantization process. As for the networks identified using NAS, they continued to deliver excellent results even after this phase. After generating, training, and testing various architectures, the optimal network, capable of striking an ideal balance between computational cost and predictive accuracy, was found to be the Accuracy Oriented Network. It guarantees post-quantization accuracy of around 96% and low memory usage, making it suitable for implementation in an embedded solution. Tests conducted in a real-world environment, performed after deploying the model on the device, confirmed the architecture's reliability, demonstrating high robustness and excellent predictive capability.

To verify the performance of the hardware used, the entire quantization system was tested using the development boards provided by STMicroelectronics for this purpose through their ST Edge AI Developer Cloud service. The comparison between these boards and Raspberry Pi 4, the only hardware currently available, showed that the STM32MP257F-EV1 microprocessor is the best solution, particularly suited to the Edge AI context in which this thesis is situated. In fact, it demonstrated inference times comparable to, and in some cases even better than, those achieved on the Raspberry Pi 4. Combining this observation with the analysis of power consumption, where STM's microprocessors clearly consume three orders of magnitude less power, led to the conclusion that this microprocessor is the best among those tested.

The work carried out has therefore yielded very good results; nonetheless, there is still ample room for improvement and further development.
First of all, radar excels at detecting moving targets, but it struggles significantly when detecting subjects who remain stationary for extended periods of time (for example, standing still or crouching motionless to pick up an object). These positions cause a significant reduction in the echoes received and deemed valid by the antenna. The few remaining signals mostly originate from the most reflective part of the human body, the torso, and consequently the center of gravity is lowered. Thus, for example, in the case of a person standing still, even if robust data augmentation has

been applied, the entire slender silhouette is not reconstructed, and as a result, the network struggles to recognize the static upright posture, mistaking it for a seated person. The problem was resolved within the inference script using a threshold mechanism, as previously explained; however, to address this issue more robustly, there are two approaches that can also be carried out in parallel. The first strategy involves conducting an even more extensive campaign of prolonged data collection, in which subjects perform normal daily activities, including some that require maintaining a largely static posture for an extended period of time, in order to further enrich the dataset. The second most promising method involves the development of data augmentation techniques based on logic that stores the last detected posture. The goal is to design an intelligent filter capable of identifying a drop in the number of points and intervening by reconstructing the person's silhouette based on how it appeared before the drop occurred, using the history of the immediately preceding frames. The biggest challenge of this approach lies in implementing a control logic that prevents the algorithm from reconstructing the point cloud even when the drop in the number of points is caused by a person who has actually sat down and then stopped moving.

Further future developments involve extending postural recognition to include situations where multiple people are present in the same room. Currently, the model has been trained only to recognize a single subject. The two-ellipsoid model processes the entire point cloud and divides it into two volumes, thereby implicitly assuming that all processed points belong to a single person. A second person in the room would result in the construction of two ellipsoids in a completely incorrect manner. In the future, it will be important to introduce a point cloud clustering module at the beginning of the processing pipeline, which would allow multiple point clouds to be separated at the beginning, so that each person's ellipsoids could be constructed. As for the implemented neural network, it could process the different subjects in parallel by performing independent inferences and classifications to recognize the different postures assumed by the various subjects present in the room.

Finally, once the classification of daily activity postures has been established, it will be particularly useful to introduce a mechanism capable of detecting potential falls. This final step would allow the postural classifier to be developed into a full-fledged alarm system that can be used in home settings for the non-invasive monitoring of frail individuals.

List Of Abbreviations

ADL: Activities of Daily Living

ALU: Arithmetic Logic Unit

BLSTM: Bidirectional Long Short-Term Memory

CNN: Convolutional Neural Network

CPU: Central Processing Unit

DSP: Digital Signal Processor

FFT: Fast Fourier Transform

FMCW: Frequency-Modulated Continuous-Wave

FP32: 32-bit Floating Point

FPS: Farthest Point Sampling

GRU: Gated Recurrent Unit

HAR: Human Activity Recognition

INT8: 8-bit Integer

IoT: Internet of Things

KNN: K-Nearest Neighbors

MCU: Microcontroller Unit

mmWave: Millimeter Wave

MQTT: Message Queuing Telemetry Transport

NAS: Neural Architecture Search

ONNX: Open Neural Network Exchange

PFI: Permutation Feature Importance

PTQ: Post-Training Quantization

RNN: Recurrent Neural Network

XAI: Explainable AI

GLOSSARY

Broker: In the MQTT protocol, the central server receives messages from publishers and routes them to the appropriate subscribers.

Inference: The phase in which a trained neural network processes unseen live data to make a prediction or classification.

Iteration: A single update of a neural network's internal weights during training, performed after processing one mini-batch of data.

Logits: The raw, non-normalized numerical scores output by the final layer of a neural network before being mapped into a probability distribution via Softmax.

Memory Footprint: The amount of memory (RAM) that a machine learning model occupies while running on a target device.

Publisher / Subscriber: The complementary roles in the MQTT architecture; the publisher broadcasts data to a specific topic, while the subscriber passively listens for data on that topic.

Quantization: An optimization technique that compresses a neural network by reducing the mathematical precision of its weights and activations (e.g., from 32-bit floating-point to 8-bit integer).

Sliding Window: A processing technique used for sequential data analysis, involving a fixed-size temporal window that shifts continuously over the data stream to capture dynamic transitions.

Test Set: A subset of the available data, never seen by the model during the training phase, used exclusively to evaluate the network's final generalization accuracy.

Training Set: Set of data utilized to train the neural network, allowing the optimization algorithm to learn patterns and adjust weights.

BIBLIOGRAPHY

[1] L. P. Fried, A. A. Cohen, Q.-L. Xue, J. Walston, K. Bandeen-Roche, and R. Varadhan, "The physical frailty syndrome as a transition from homeostatic symphony to cacophony," *Nat. Aging*, vol. 1, no. 1, p. 36—46, Jan. 2021, doi: 10.1038/s43587-020-00017-z.

[2] A. Clegg, J. Young, S. Iliffe, M. O. Rikkert, and K. Rockwood, "Frailty in elderly people," *The Lancet*, vol. 381, no. 9868, pp. 752–762, Mar. 2013, doi: 10.1016/S0140-6736(12)62167-9.

[3] K. Christensen, G. Doblhammer, R. Rau, and J. W. Vaupel, "Ageing populations: the challenges ahead," *The Lancet*, vol. 374, no. 9696, pp. 1196–1208, Oct. 2009, doi: 10.1016/S0140-6736(09)61460-4.

[4] A. J. Cruz-Jentoft *et al.*, "Sarcopenia: European consensus on definition and diagnosis: Report of the European Working Group on Sarcopenia in Older People," *Age Ageing*, vol. 39, no. 4, pp. 412–423, Jul. 2010, doi: 10.1093/ageing/afq034.

[5] B. J. VELLAS, S. J. WAYNE, L. J. ROMERO, R. N. BAUMGARTNER, and P. J. GARRY, "Fear of falling and restriction of mobility in elderly fallers," *Age Ageing*, vol. 26, no. 3, pp. 189–193, May 1997, doi: 10.1093/ageing/26.3.189.

[6] E. Torti *et al.*, "Embedded Real-Time Fall Detection with Deep Learning on Wearable Devices," in *2018 21st Euromicro Conference on Digital System Design (DSD)*, 2018, pp. 405–412. doi: 10.1109/DSD.2018.00075.

[7] X. Fafoutis, L. Marchegiani, A. Elsts, J. Pope, R. Piechocki, and I. Craddock, "Extending the battery lifetime of wearable sensors with embedded machine

learning," in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, 2018, pp. 269–274. doi: 10.1109/WF-IoT.2018.8355116.

[8] H. H. Pham, H. Salmane, L. Khoudour, A. Crouzil, S. A. Velastin, and P. Zegers, "A Unified Deep Framework for Joint 3D Pose Estimation and Action Recognition from a Single RGB Camera," *Sensors*, vol. 20, no. 7, 2020, doi: 10.3390/s20071825.

[9] D. Mehta *et al.*, "VNect: real-time 3D human pose estimation with a single RGB camera," *ACM Trans. Graph.*, vol. 36, no. 4, Jul. 2017, doi: 10.1145/3072959.3073596.

[10] S. Nooruddin, Md. M. Islam, F. A. Sharna, H. Alhetari, and M. N. Kabir, "Sensor-based fall detection systems: a review," *J. Ambient Intell. Humaniz. Comput.*, vol. 13, no. 5, pp. 2735–2751, 2022, doi: 10.1007/s12652-021-03248-z.

[11] V. N. Dang, N. C. Hoang, M. T. Le, K. Nguyen, and Q. C. Nguyen, "Deep Learning-Based Human Activity Recognition With FMCW Radar: A Review," *IEEE Sens. J.*, vol. 26, no. 4, pp. 5302–5326, 2026, doi: 10.1109/JSEN.2026.3651578.

[12] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, 2016, doi: 10.1109/JIOT.2016.2579198.

[13] Y. Kim and H. Ling, "Human Activity Classification Based on Micro-Doppler Signatures Using a Support Vector Machine," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 47, no. 5, pp. 1328–1337, 2009, doi: 10.1109/TGRS.2009.2012849.

[14] B. Jokanovic, M. Amin, and F. Ahmad, "Radar fall motion detection using deep learning," in *2016 IEEE Radar Conference (RadarConf)*, 2016, pp. 1–6. doi: 10.1109/RADAR.2016.7485147.

[15] F. Luo, S. Khan, A. Li, Y. Huang, and K. Wu, “EdgeActNet: Edge Intelligence-Enabled Human Activity Recognition Using Radar Point Cloud,” *IEEE Trans. Mob. Comput.*, vol. 23, no. 5, pp. 5479–5493, 2024, doi: 10.1109/TMC.2023.3309938.

[16] W.-L. Hsu, J.-X. Liu, C.-C. Yang, and J.-S. Leu, “A Fall Detection System Based on FMCW Radar Range-Doppler Image and Bi-LSTM Deep Learning,” *IEEE Sens. J.*, vol. 23, no. 18, pp. 22031–22039, 2023, doi: 10.1109/JSEN.2023.3300994.

[17] H.-U.-R. Khalid, A. Gorji, A. Bourdoux, S. Pollin, and H. Sahli, “Multi-View CNN-LSTM Architecture for Radar-Based Human Activity Recognition,” *IEEE Access*, vol. 10, pp. 24509–24519, 2022, doi: 10.1109/ACCESS.2022.3150838.

[18] A. D. Singh, S. S. Sandha, L. Garcia, and M. Srivastava, “RadHAR: Human Activity Recognition from Point Clouds Generated through a Millimeter-wave Radar,” in *Proceedings of the 3rd ACM Workshop on Millimeter-Wave Networks and Sensing Systems*, in mmNets ’19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 51–56. doi: 10.1145/3349624.3356768.

[19] 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. doi: 10.1109/CVPR.2017.16.

[20] X. Dang, P. Jin, Z. Hao, W. Ke, H. Deng, and L. Wang, “Human Movement Recognition Based on 3D Point Cloud Spatiotemporal Information from Millimeter-Wave Radar,” *Sensors*, vol. 23, no. 23, 2023, doi: 10.3390/s23239430.

[21] D. De Vittorio, A. Barili, G. Danese, and E. Marenzi, “Artificial Intelligence for the Evaluation of Postures Using Radar Technology: A Case Study,” *Sensors*, vol. 24, no. 19, 2024, doi: 10.3390/s24196208.

[22] S. M. Patole, M. Torlak, D. Wang, and M. Ali, “Automotive radars: A review of signal processing techniques,” *IEEE Signal Process. Mag.*, vol. 34, no. 2, pp. 22–35, 2017, doi: 10.1109/MSP.2016.2628914.

[23] J. Li and P. Stoica, “MIMO Radar with Colocated Antennas,” *IEEE Signal Process. Mag.*, vol. 24, no. 5, pp. 106–114, 2007, doi: 10.1109/MSP.2007.904812.

[24] S. R. Cesar Iovescu, “The fundamentals of millimeter wave sensors,” *Texas Instruments*, pp. 1–8, May 2017, Accessed: May 12, 2026. [Online]. Available: https://www.mouser.lt/pdfdocs/mmwavewhitepaper.pdf

[25] Sandeep Rao, “MIMO Radar,” *Texas Instruments Application Report*, pp. 1–13, May 2017, Accessed: May 12, 2026. [Online]. Available: https://www.ti.com/lit/an/swra554a/swra554a.pdf?ts=1778499304481&ref_url=https%253A%252F%252Fdev.ti.com%252F

[26] Texas Instruments, “IWR1443 Single-Chip 76- to 81-GHz mmWave Sensor,” 2018, Accessed: Jul. 05, 2026. [Online]. Available: https://www.ti.com/lit/ds/symlink/iwr1443.pdf?ts=1783193032155

[27] Texas Instruments, “IWR1843 Single-Chip 76 to 81GHz FMCW mmWave Sensor,” 2024, Accessed: Jul. 05, 2026. [Online]. Available: https://www.ti.com/lit/ds/symlink/iwr1843.pdf?ts=1783256701065

[28] Vayyar Imaging, “Mini-Circuits and Vayyar bring 3D millimeter wave imaging and sensing to the global market”, Accessed: Jul. 05, 2026. [Online]. Available: https://en.sekorm.com/doc/3650701.html

[29] Infineon Technologies AG, “BGT60TR13C datasheet,” 2023. Accessed: Jul. 05, 2026. [Online]. Available:

https://www.infineon.com/assets/row/public/documents/24/49/infineon-bgt60tr13c-datasheet-en.pdf

[30] STMicroelectronics, “Evaluation board with STM32MP257F MPU”, Accessed: Jul. 05, 2026. [Online]. Available: https://www.st.com/en/evaluation-tools/stm32mp257f-ev1.html

[31] STMicroelectronics, “STM32MP157F-DK2”, Accessed: Jul. 05, 2026. [Online]. Available: https://www.st.com/en/evaluation-tools/stm32mp157f-dk2.html

[32] STMicroelectronics, “STM32MP135F-DK”, Accessed: Jul. 05, 2026. [Online]. Available: https://www.st.com/en/evaluation-tools/stm32mp135f-dk.html

[33] Texas Instruments, “IWR6843, IWR6443 Single-Chip 60 to 64GHz mmWave Sensor,” 2025, Accessed: Jul. 05, 2026. [Online]. Available: https://www.ti.com/product/IWR6843

[34] Raspberry Pi Ltd, “Raspberry Pi 4 Model B,” 2026, Accessed: Jul. 05, 2026. [Online]. Available: https://pip-assets.raspberrypi.com/categories/634-raspberry-pi-compute-module-4/documents/RP-008168-DS-4-cm4-datasheet.pdf

[35] I. D. Mienye, T. G. Swart, and G. Obaido, “Recurrent Neural Networks: A Comprehensive Review of Architectures, Variants, and Applications,” *Information*, vol. 15, no. 9, 2024, doi: 10.3390/info15090517.

[36] H. Lin and Q. Sun, “Crude Oil Prices Forecasting: An Approach of Using CEEMDAN-Based Multi-Layer Gated Recurrent Unit Networks,” *Energies (Basel).*, vol. 13, p. 1543, May 2020, doi: 10.3390/en13071543.

[37] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997, doi: 10.1162/neco.1997.9.8.1735.

[38] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,” Dec. 2014, doi: 10.48550/arXiv.1412.3555.

[39] Inc. The MathWorks, “lstmProjectedLayer,” MATLAB Documentation. Accessed: May 21, 2026. [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/lstmprojectedlayer.html

[40] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997, doi: 10.1109/78.650093.

[41] J. Heaton, “Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning: The MIT Press, 2016, 800 pp, ISBN: 0262035618,” *Genet. Program. Evolvable Mach.*, vol. 19, May 2016, doi: 10.1007/s10710-017-9314-z.

[42] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, “Deep Learning for 3D Point Clouds: A Survey,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PP, p. 1, May 2020, doi: 10.1109/TPAMI.2020.3005434.

[43] S. Gao, J. Zhang, L. Mei, S. Wang, and X. Wang, “Exploring Spatial-Temporal Representation via Star Graph for mmWave Radar-based Human Activity Recognition,” Dec. 2025, doi: 10.1109/TMC.2025.3634221.

[44] 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. doi: 10.1109/CVPR.2017.16.

[45] W. Wu, Z. Qi, and L. Fuxin, “ PointConv: Deep Convolutional Networks on 3D Point Clouds ,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* , Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2019, pp. 9613–9622. doi: 10.1109/CVPR.2019.00985.

[46] C. White *et al.*, “Neural Architecture Search: Insights from 1000 Papers,” *arXiv preprint arXiv:2301.08727*, Jan. 2023.

[47] T. Elsken, J. Metzen, and F. Hutter, “Neural Architecture Search: A Survey,” Jun. 2018. doi: 10.48550/arXiv.1808.05377.

[48] D. Gunning, M. Stefik, J. Choi, T. Miller, S. Stumpf, and G.-Z. Yang, “XAI—Explainable artificial intelligence,” *Sci. Robot.*, vol. 4, no. 37, p. eaay7120, 2019, doi: 10.1126/scirobotics.aay7120.

[49] A. Barredo Arrieta *et al.*, “Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI,” *Inf. Fusion*, vol. 58, no. C, pp. 82–115, Jun. 2020, doi: 10.1016/j.inffus.2019.12.012.

[50] N. Gillani and T. Arslan, “Clinically Interpretable XAI-Based FMCW Radar Fall Detection with Reduced False Alarms,” in *2025 6th International Conference on Bio-engineering for Smart Technologies (BioSMART)*, 2025, pp. 1–4. doi: 10.1109/BioSMART66413.2025.11046189.

[51] P. Zhao *et al.*, “mID: Tracking and Identifying People with Millimeter Wave Radar,” in *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2019, pp. 33–40. doi: 10.1109/DCOSS.2019.00028.

[52] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, “Deep learning for sensor-based activity recognition: A survey,” *Pattern Recognit. Lett.*, vol. 119, pp. 3–11, 2019, doi: https://doi.org/10.1016/j.patrec.2018.02.010.

[53] M. Jiang, S. Guo, H. Luo, Y. Yao, and G. Cui, “A Robust Target Tracking Method for Crowded Indoor Environments Using mmWave Radar,” *Remote Sens. (Basel).*, vol. 15, no. 9, 2023, doi: 10.3390/rs15092425.

[54] Y. Gao, R. Geng, D. Zhang, Y. Hu, H. Lin, and Y. Chen, “MM-DCDR: A Benchmark of Device Configuration and Data Representation for mmWave-Based Human Sensing,” in *2024 16th International Conference on Wireless*

Communications and Signal Processing (WCSP), 2024, pp. 801–806. doi: 10.1109/WCSP62071.2024.10827599.

[55] A. F. Agarap, "Deep Learning using Rectified Linear Units (ReLU)," Apr. 2026, doi: 10.48550/arXiv.1803.08375.

[56] R. A. Johnson and D. W. Wichern, *Applied Multivariate Statistical Analysis*, 6th ed. Pearson Prentice Hall, 2007.

[57] Ibtesam Ahmed, "Confidence Intervals and how to find them." Accessed: Jul. 02, 2026. [Online]. Available: https://medium.com/@ibtesamahmex/confidence-intervals-and-how-to-find-them-42fe4e0900c3

[58] M. Weinmann, B. Jutzi, S. Hinz, and C. Mallet, "Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 105, Jun. 2015, doi: 10.1016/j.isprsjprs.2015.01.016.

[59] S. Gottschalk, M. C. Lin, and D. Manocha, "OBBTree: a hierarchical structure for rapid interference detection," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, in SIGGRAPH '96. New York, NY, USA: Association for Computing Machinery, 1996, pp. 171–180. doi: 10.1145/237170.237244.

[60] M. Wu, H. Jiao, and J. Nan, "Sparse 3D Point Cloud Parallel Multi-Scale Feature Extraction and Dense Reconstruction with Multi-Headed Attentional Upsampling," *Electronics (Basel).*, vol. 11, no. 19, 2022, doi: 10.3390/electronics11193157.

[61] Gilbert Strang, *Linear Algebra and Its Applications*, Fourth. Brooks Cole, 2006.

[62] Z. Wang, D. Jiang, B. Sun, and Y. Wang, “A Data Augmentation Method for Human Activity Recognition Based on mmWave Radar Point Cloud,” *IEEE Sens. Lett.*, vol. 7, no. 5, pp. 1–4, 2023, doi: 10.1109/LSENS.2023.3270894.

[63] Y. Qiang, K. Zhou, and K. I.-K. Wang, “Privacy Preserving Dual Millimetre Wave Radar Based Human Activity Recognition,” in *2025 IEEE International Conference on High Performance Computing and Communications (HPCC)*, 2025, pp. 710–717. doi: 10.1109/HPCC67675.2025.00108.

[64] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A Next-generation Hyperparameter Optimization Framework,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, in KDD ’19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 2623–2631. doi: 10.1145/3292500.3330701.

[65] D. Hendrycks and K. Gimpel, “Gaussian Error Linear Units (GELUs),” Jun. 2023, doi: 10.48550/arXiv.1606.08415.

[66] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” Jan. 2017, doi: 10.48550/arXiv.1412.6980.

[67] T. Fawcett, “Introduction to ROC analysis,” *Pattern Recognit. Lett.*, vol. 27, pp. 861–874, Jun. 2006, doi: 10.1016/j.patrec.2005.10.010.

[68] M. Sokolova and G. Lapalme, “A systematic analysis of performance measures for classification tasks,” *Inf. Process. Manag.*, vol. 45, pp. 427–437, Jun. 2009, doi: 10.1016/j.ipm.2009.03.002.

[69] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, “Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing,” *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019, doi: 10.1109/JPROC.2019.2918951.

[70] S. Han, H. Mao, and W. J. Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding,” Feb. 2016, doi: 10.48550/arXiv.1510.00149.

[71] A. Paszke *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” Dec. 2019, doi: 10.48550/arXiv.1912.01703.

[72] B. Jacob *et al.*, “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713. doi: 10.1109/CVPR.2018.00286.

[73] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, “A Survey of Quantization Methods for Efficient Neural Network Inference,” Jun. 2021, doi: 10.48550/arXiv.2103.13630.

[74] J. , L. F. , Z. K. Bai, “ONNX: Open Neural Network Exchange. GitHub repository.” Accessed: Jun. 05, 2026. [Online]. Available: https://github.com/onnx/onnx

[75] U. Hunkeler, H. Truong, and A. Stanford-Clark, “MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks,” Jun. 2008, pp. 791–798. doi: 10.1109/COMSWA.2008.4554519.

[76] B. Mishra and A. Kertész, “The Use of MQTT in M2M and IoT Systems: A Survey,” *IEEE Access*, vol. 8, pp. 201071–201086, Jun. 2020, doi: 10.1109/ACCESS.2020.3035849.

[77] STMicroelectronics, “Guidelines for measuring the system power consumption on the STM32MP25x line MPUs”, Accessed: Jul. 05, 2026. [Online]. Available: https://www.st.com/resource/en/application_note/an5730-guidelines-for-measuring-the-system-power-consumption-on-the-stm32mp25x-line-mpus-stmicroelectronics.pdf

[78] STMicroelectronics, “STM32MP13x product lines system power consumption”, Accessed: Jul. 05, 2026. [Online]. Available: https://www.st.com/resource/en/application_note/an5787-stm32mp13x-product-line-system-power-consumption-stmicroelectronics.pdf

[79] STMicroelectronics, “STM32MP1 Series system power consumption”, Accessed: Jul. 05, 2026. [Online]. Available: https://www.st.com/resource/en/application_note/an5284-stm32mp1-series-system-power-consumption-stmicroelectronics.pdf