POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

TinyML UWB-Radar based fall detection

Tesi di Laurea Magistrale in
Computer Science Engineering - Ingegneria Informatica

Amedeo Carrioli, 953778

Abstract: Fall detection is the process of identifying human falls; this is an increasingly important task in various fields, specifically in healthcare and elderly care, where falls happen frequently. Falls can lead to important injuries, which often remain unnoticed for long periods of time, hence the need for an automated fall detection method. This project presents an algorithm that solves the problem of fall detection using an Ultra-Wideband (UWB) radar and an advanced neural network model, which is deployed on an Internet of Things (IoT) device. The model makes inference directly on the device, an Arduino microcontroller with limited resources. The model's training is computed on several UWB-radar recordings of falls and other human activities, from which it learns to recognize typical patterns of human falls. UWB radar brings numerous benefits to the fall detection field compared to traditional technologies such as accelerometers, gyroscopes, and cameras. Indeed, it is a non-intrusive system that preserves the user's privacy (UWB radars do not capture or record visual images, individuals' faces, and physical appearances).
This thesis emphasizes the role of Tiny Machine Learning (TinyML), which enabled the optimization of the algorithm for devices with limited computational resources, leading to a compact and energy-efficient solution.
In summary, this research successfully integrates UWB radar technology, deep neural networks, and TinyML to develop an innovative, privacy-conscious fall detection solution. These technologies hold great potential to improve the life quality of elderly people and beyond, acting as a benchmark for future advancements in this critical health and safety field.

Advisor:
Prof. Manuel Roveri

Co-advisors:
Massimo Pavan

Academic year:
2022-2023

Key-words: Tiny Machine Learning, Ultra-Wideband Radar, Fall Detection, Internet of Things, Deep Neural Networks

1. Introduction

Tiny Machine Learning (TinyML) is a fast-emerging field that combines Machine Learning (ML) with compact, low-power devices, embedded systems, and Internet of Things (IoT) technology. As the trend towards edge computing grows, the demand for TinyML is rapidly increasing due to its ability to bring artificial intelligence to devices without needing continuous cloud connectivity. The revolutionary ability of TinyML is its ability to process data on the spot where it's generated, ensuring enhanced privacy and security.
The aim of this thesis is to develop an algorithm that, exploiting UWB radars, detects falls of people by making inference directly on a microcontroller.
Since the inference process is performed locally on the device, there's no need to send data to a central server and then wait for a response. This reduces the round-trip time substantially, ensuring real-time or near-real-time responses, and guaranteeing low latency, which are crucial for certain applications like health monitoring. Furthermore, by running machine learning models locally, TinyML eliminates the need for a continuous internet connection and can thus operate offline.
The targets of the proposed algorithm are IoT devices because they are more affordable than other devices such as cameras, and they can be precisely deployed in controlled environments.
UWB radars, on the other side, are gaining traction due to their versatility. They offer centimeter-level precision in determining the location between a transmitter and receiver (ranging); and they can detect movements and targets using a single device, by calculating the time-of-flight of radar pulses sent and bounced back from the object (radar). This thesis makes use of the latter functionality. UWB radars are becoming increasingly precise while their costs are decreasing. Given the advantages they offer, including privacy and low power consumption, coupled with their vast application potential, UWB radar technology is undeniably of great significance.

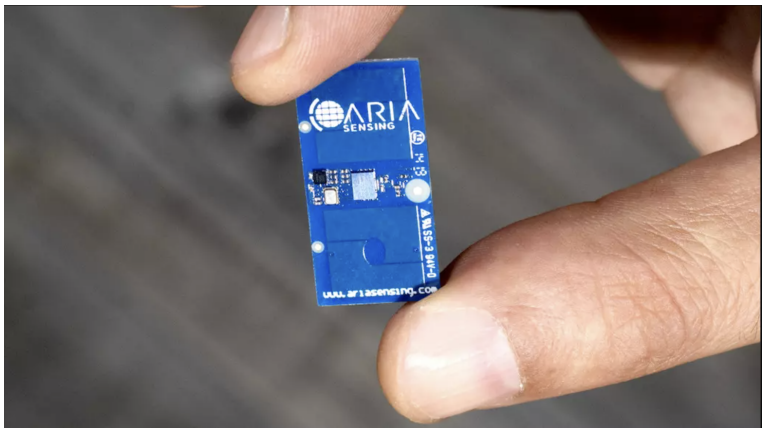


Figure 1: UWB radar module produced by Aria Sensing

The combination of TinyML and UWB radars can lead to the development of new, low-power and low-memory consumption, on-device, and privacy-preserving applications, exemplified by this project.
The fall-detection solution, hereby proposed, could be applied to multiple environments such as private houses, industrial environments, or even hotels or cruises; Nonetheless, the thesis has been developed for deployment in nursing homes, where falls happen frequently, often leading to significant injuries on delicate patients.
To develop this algorithm, a novel neural network architecture has been designed, which is especially efficient for the data collected. Indeed, none of the well-studied neural networks in literature were suited for this application given their size (too big for deployment on IoT devices, a crucial component of this proposed solution).
The results achieved are promising: 0.98 accuracy on the original model (no quantization applied) and 0.78 accuracy on the quantized model (which means the data types of the network are reduced from 32 to 8-bit) deployed on a microcontroller, and the memory footprint is only 44KB.

The model used for these achievements is trained on an ad-hoc collected dataset of radar recordings capturing the most common human activities, such as sitting on a chair, walking around, or even picking up something from the ground (an activity specifically chosen because is similar to a fall).
Not only so, but a new radar model has been used for the very first time for a problem of this nature - the whole algorithm is designed to exploit the self-collected data at its best.
The attention is focused on the dataset, which was collected in controlled environments (different rooms) with the help of different contributors; Then, a preprocessing sequence was applied to extract significant information from those data. The preprocessing step includes a series of operations and a decluttering method, whose aim is to remove static noise and highlight the relevant movements of subjects.
An important step was the deployment of the solution on an IoT device, an Arduino microcontroller with only 256KB of SRAM. The microcontroller used is a placeholder for the final UWB radar that will be used in real applications, for example in nursing homes. Bringing the solution on a very limited memory microcontroller shows that the proposed solution works on limited-resource devices, hence bringing it to the final radar is a feasible next step. This proves the model's adaptability and scalability, highlighting the way for its integration with sophisticated UWB radars.
Many experiments have been conducted to finetune not only the preprocessing part but also the network itself since a balance between network performance and its dimensions is essential for its deployment on microcontrollers.

1.1. Thesis structure

- **Chapter 2** presents the state of the art and some required background.
- **Chapter 3** contains a formal description of the problems that this thesis solves.
- **Chapter 4** describes the devices used for the development of this research.
- **Chapter 5** delineates the proposed solutions to the problem.
- **Chapter 6** introduces the dataset used, describing the data-collection process.
- **Chapter 7** reports the deployment of the solution on a small, memory-constrained device.
- **Chapter 8** focuses on the experiment and subsequent results reached by this work.
- **Chapter 9** draws the conclusions, presenting the future developments that are planned.

2. Background and state-of-the-art

This section gives an overview of topics related to this thesis.

2.1. Fall detection

At its core, fall detection refers to a mechanism that automatically identifies when an individual experiences a fall. It is often paired with alert systems to notify caregivers or medical professionals. This process becomes particularly crucial in settings like nursing homes, where residents face a higher risk of falling and might not have immediate access to assistance. Fall detection has grown increasingly important in various sectors, especially in healthcare and elderly care, as falls can potentially lead to excruciating injuries and can hence reduce the quality of life [34].
Historically, fall detection systems were based on wearable sensors or cameras [25]. Wearable-based systems, often in the form of pendants or wristbands, rely on accelerometers and gyroscopes to detect changes in motion and orientation. When abnormal patterns are detected that align with a fall, an alert is sent out. On the other hand, camera-based systems use visual data to analyze human posture and detect falls. Both these methods, while effective, come with challenges: wearables require individuals to consistently wear the device, let it be a wristband or a pendant, and camera systems pose privacy concerns.
Numerous studies sought to refine these methods and propose alternative solutions. A growing body of research has investigated the potential of neural networks to solve the fall detection problem. Neural networks, with their ability to discern patterns in large and complex datasets, offer a promising path for accurate and timely fall detection, especially when paired with new data-collection tools.
One emerging tool in this field is the Ultra-Wideband (UWB) radar. UWB radars, known for their high precision and low power consumption, have shown great potential for human activity recognition. Unlike cameras, UWB radars preserve privacy by not capturing identifiable visual data, and unlike wearables, they do not require user compliance. These radars send out signals that reflect off of objects, including humans. By analyzing the returned signals, it is possible to discern human activities, including falls. Preliminary research in this field indicates promising results [24], highlighting the potential of UWB radars, in conjunction with neural networks, to revolutionize fall detection, especially in high-risk environments like nursing homes. Although numerous studies propose algorithms that solve this problem, none of them combine TinyML with UWB radar technology, deploying the solution on a low-power and limited resources device, like the algorithm proposed in this thesis.

2.2. Human Activity Recognition

Human Activity Recognition (HAR) is the process of identifying and categorizing human motions or actions using data acquired from sensors or other sources. Human Activity Recognition stands at the forefront of advancements in areas such as health monitoring, surveillance, and smart environments. By recognizing and analyzing human motions and postures, HAR systems can provide detailed insights into individual behaviors, making them suitable for a multitude of applications.
Historically, HAR was implemented using camera-based systems [47]. These systems, equipped with advanced computer vision algorithms, were adept at processing visual data to identify and categorize various human activities. However, while effective, these systems were often criticized for potential invasions of privacy, especially in personal or sensitive environments. Parallelly, wearable sensors, embedded with accelerometers and gyroscopes, offered an alternative [32]. These devices could monitor the user's movements, interpreting them to recognize specific activities. Yet, they come with the need for consistent user wearability and battery maintenance.
With the ever-growing demand for non-intrusive and accurate HAR systems, researchers began exploring innovative techniques and tools, such as Ultra-Wideband (UWB) radars. Unlike traditional methods, UWB radars could capture human activities without any direct visual identification or the need for wearables.

Moreover, the application of UWB-based HAR is vast. In healthcare, it can aid in patient monitoring without the constraints of wearables or the privacy concerns of cameras. In smart homes, it can enhance automation by recognizing user activities and adjusting the environment accordingly.
In summary, as the field of Human Activity Recognition evolves, the integration of UWB radars, especially when paired with advanced neural networks, offers a promising path. This combination respects individual privacy while ensuring detailed and accurate activity recognition, paving the way for smarter, more intuitive systems across various sectors. Even if this thesis's main focus is fall detection, we will explain that the collected dataset used in this project is composed of different human activities, and the experiments on it have been extended to the HAR field.

2.3. TinyML

Tiny machine learning (TinyML) represents an exciting frontier in the field of artificial intelligence because it concerns the deployment of machine learning solutions on low-resource embedded devices. These kinds of devices have been considered incompatible with ML solutions because of their limited memory and computational power; Thanks to model compression methods, however, machine learning algorithms can be successfully deployed on such devices despite their limitations. This area of study is gaining traction due to its potential to revolutionize real-time analytics in various applications, one of which is fall detection.
Integrating machine learning solutions with fall detection systems poses unique challenges primarily due to the resource limitations of the embedded devices. These devices often lack the memory and processing power that typical machine learning models demand, and yet they need to run these models in real-time. Furthermore, fall detection, by its very nature, requires high accuracy and reliability to be effective.
Despite these challenges, innovative techniques have emerged that make it possible to deploy complex machine learning models on the aforementioned low-resource devices. Model compression, for instance, plays a crucial role by trimming down the size of the machine learning model without significant impact on its performance. For example, pruning of channels and layers of Convolutional Neural Networks (CNNs) has proven to be successful in reducing the memory and computational demand of the model [17]. Another approach is quantization, through which data is stored in a limitedly precise way, consequently reducing the memory required to store CNNs models, for example, weights and activation functions can be converted from 32-bit to 8-bit [7].
Edge processing signifies another significant leap in this direction. By executing machine learning models on the device itself, where data is generated, it eliminates the need to transmit data to the cloud for processing. This results in less latency and enhanced privacy, a key advantage in sensitive applications like fall detection.
Importantly, these approaches apply to model evaluation only, which is the testing of an already trained model; The training of the model itself is a much more complex topic since it requires memory to store intermediate activations, and it relies on precise derivative calculations.
The research is in the nascent stages, but the prospects are encouraging. With ongoing advancements, TinyML holds the potential to significantly improve the safety of those at risk of falls, opening new possibilities for deploying robust fall detection models on a wide array of devices, including UWB radars, smartphones, and wearables.

2.4. Ultra-Wideband Radars

Ultra-wideband (UWB) radars are a type of radar that uses radio waves with a wide bandwidth. This allows UWB radars to achieve high accuracy in distance and speed measurements, as well as high-resolution images of objects in the environment. UWB radars emit low-power radio waves, another characteristic that differentiates them from traditional radars, which makes them less likely to interfere with other devices and more suitable for use in crowded environments.
UWB radars emit short-pulses radio waves, typically in the order of a few nanoseconds, at an arbitrary frequency [44]. These pulses are reflected off of objects in the environment and the time it takes for the pulses to return to the radar is used to determine the radar-object distance. The wider the bandwidth of the UWB radar, the

more accurate the distance measurements can be [11].
In addition, UWB radars can be used to measure the speed of objects by comparing the time it takes for the pulses to travel to the object and back to the radar. This is done by using a technique called Time of Flight (ToF) [33].
What sets UWB radars apart from conventional radar systems is their precision. Not only can they detect subtle shifts in movement, but they have also showcased their prowess in specialized applications like fall detection. This becomes priceless for vulnerable groups such as the elderly, providing a safety net against potential accidents.
Delving into the technical aspects, UWB radars utilize a spectrum spanning 3.1 to 10.6 GHz, giving them an expansive frequency range to work with. Consequently, their wide bandwidth facilitates rapid data conveyance. Despite their promising features, UWB radars present some challenges. Indeed, they can be sensitive to environmental elements such as walls or bulky furniture, and their deployment might be costly compared to traditional sensors. The big tech industry, however, is betting on them. Companies like Apple have integrated UWB radar technology into flagship products like iPhones (ever since iPhone 11) and AirTags [2]. The potential applications of this emerging technology are vast: from pinpointing indoor locations, and actively tracking both stationary and mobile targets, to monitoring physiological parameters like breathing and heart rate.
With their ability to operate in both Line of Sight (LoS) and Non-Line of Sight (NLoS) conditions, UWB radars exhibit versatility in various settings. LoS refers to direct transmission pathways between the transmitter and receiver, and NLoS indicates paths where signals are obstructed or reflected by obstacles. The reason for this capability, especially in NLoS conditions, is that the pulses used by UWB can penetrate through certain obstacles such as walls, foliage, and other non-metallic barriers. This penetration ability allows UWB radars to detect and measure signals even when there's an obstruction between the radar and the target [12]. For instance, UWB radars are employed in search and rescue operations to locate survivors trapped under rubble or behind walls, capitalizing on their NLoS detection capabilities.
There are only a few scientific papers that leverage UWB technology in the TinyML domain. An example is the one by Pavan, which showcases a presence detection algorithm based on UWB radar, executing it on tiny devices such as Internet-of-Things units, setting new standards for this technology [31].
There is growing interest in the utilization of UWB radars for fall detection, with companies developing UWB-based systems and an expanding body of research on the topic. With ongoing exploration and advancements, UWB radars hold the potential to become a standard technology for fall detection.

2.5. CNN

Convolutional neural networks (CNNs) are a type of deep learning neural network commonly used for image classification and other computer vision tasks. CNNs are central in deep learning because their architecture excels at processing spatial hierarchies, making them especially qualified at tasks like image and speech recognition. By leveraging convolutional layers, they can detect intricate patterns in data with fewer parameters than traditional neural networks. CNNs are composed of a series of convolutional layers, each of which performs a convolution operation on its input data. The convolution operation extracts features from the input data, which are then used by subsequent layers to classify the data.
Convolution is a mathematical operation that takes two functions as input and produces a third function. In the context of CNNs, the first function is the input data, and the second function is a filter. The filter is a small matrix of weights that is applied to the input data to extract features. The convolutional layers of a CNN are arranged in a hierarchical fashion. The first convolutional layer extracts simple features, such as edges and corners. The subsequent convolutional layers combine these simple features to extract more complex features, such as shapes and objects.
CNNs have been shown to be very effective for several tasks, including image classification, object detection, and face recognition. They are now state-of-the-art for many computer vision tasks.
A study by Park et al., for example, used a CNN to make human Activity Recognition (HAR) of people using cameras. The CNN was able to achieve an accuracy of 99.55% [30].

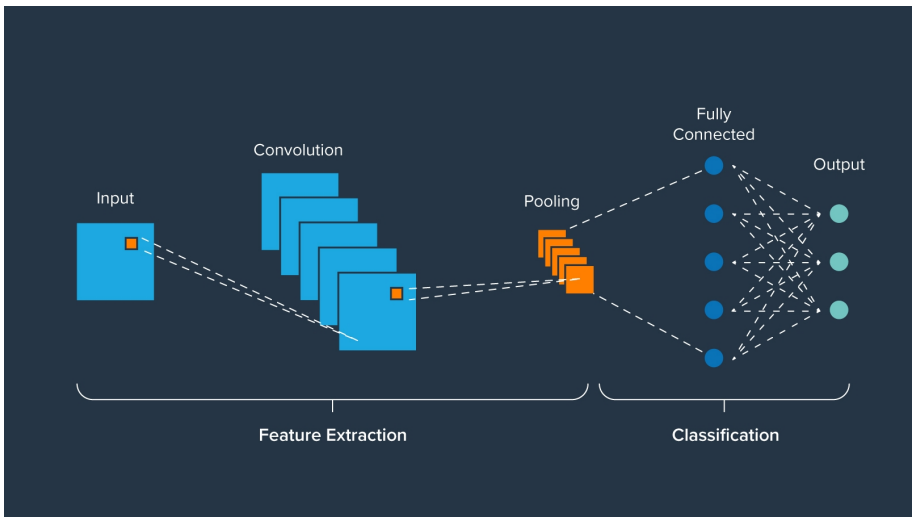


Figure 2: Convolution neural network

Binary classification is a type of supervised machine learning task where the goal is to predict a categorical outcome with two possible classes, such as "dog" or "cat", "yes" or "no". The input data for a binary classification problem is typically a set of features that describe the object being classified. The output data is a single binary value, such as 0 or 1.
Multi-classification, by contrast, is a type of supervised machine learning task where the goal is to predict a categorical outcome with more than two possible classes. The input data for a multi-classification problem is typically a set of features that describe the object being classified. The output data is a vector of values, one for each possible class.
The work of CNNs with UWB radars is still in its early stages, but it has the potential to be a powerful tool for a variety of applications. For example, CNNs could be used to classify the activities of people in a room, to detect falls in elderly people, or to track the movement of objects in a factory.
The importance of CNNs for the development of UWB radar data cannot be overstated. UWB radar systems generate high-dimensional datasets rich with information about the environment, including the detection of movements like human falls. Given the complexity of these datasets, it's often challenging to interpret them using traditional machine learning methods. CNNs, however, with their ability to recognize and learn intricate patterns in data, can be highly effective in analyzing UWB radar signals.
In essence, CNNs represent a promising tool for UWB radar data processing. Their potential, although explored to some extent, calls for more extensive investigations. These neural networks may indeed unlock new dimensions in the development of effective, real-time fall detection systems, significantly contributing to the safety and well-being of vulnerable individuals.

2.6. Tensorflow lite micro

TensorFlow Lite Micro (TFLM) is a specialized extension of TensorFlow crafted for running neural network models on tiny, resource-constrained devices. This framework is tailored to be deployed on embedded systems, specifically microcontrollers, and compact integrated circuits designed to govern a specific operation in an embedded system. A typical microcontroller includes a processor, memory, and input/output (I/O) peripherals on a single chip [13]. A noteworthy feature of TFLM is its ability to run without dependencies on standard libraries or dynamic memory allocation.
TFLM is fundamentally constructed around two pivotal components: the converter and the interpreter. The converter is the tool that transforms this model into the `.tflite` format, which can further be transformed into a hex string which, in turn, can be uploaded directly to the microcontroller. The interpreter, juxtaposed to the converter, resides on the microcontroller. Its primary function is to interpret and execute inferences based on the model definition handed over by the converter. A noteworthy point here is that TFLM is strictly an

inference engine; it does not support training capabilities on-device.
Given the nature of TinyML, memory optimization is essential. In the quest for conserving memory, TFLM incorporates techniques like pruning and quantization.

2.7. Quantization

Quantization, in the context of machine learning, is the process of reducing the number of bits that represent the weights, biases, and in some cases input and output of a neural network model. Instead of using 32-bit floating-point numbers to represent these values, quantization might use fewer bits, such as 8 bits (known as 8-bit quantization). This can be done by rounding or truncating the value to a smaller number of bits. By doing so, the model size shrinks significantly, which can lead to faster computation and decreased memory usage, making it highly suitable for resource-constrained environments.
To achieve accurate activation quantization, the framework necessitates representative input examples. These input samples play a pivotal role in gauging the range of activation values, ensuring the quantized model remains robust and accurate.
It has been shown that with quantization-aware training, networks can achieve comparable performance to their floating-point counterparts, even with significantly reduced precision [3]. The importance of quantization can't be overstated, especially for on-device machine learning, because it allows models to fit into tiny microcontrollers. Devices like smartphones, IoT gadgets, and particularly microcontrollers, which don't possess the computational firepower of cloud servers, greatly benefit from quantized models. They can run models locally, saving on both time and energy that would have been expended in communicating with larger servers.
TensorFlow Lite Micro heavily relies on quantization. The memory constraints of microcontrollers make running typical deep learning models impossible, but with quantization, especially 8-bit full quantization, TensorFlow Lite Micro can bring the power of machine learning to these tiny devices.
In summary, quantization, and in particular 8-bit full quantization, is revolutionizing the deployment of machine learning models. Machine learning is not merely concerning colossal servers crunching numbers; Thanks to techniques like quantization, machine learning exists right in our pockets, our homes, and our everyday gadgets. With advancements like these, the future of on-device intelligence looks not just promising but inevitable.

2.8. Decluttering

Decluttering is an essential procedure in signal processing, especially in the context of radar systems. It refers to the removal or reduction of unwanted interference or noise, ensuring that the primary signal is as clear and interpretable as possible. This technique is especially important for radar technologies that operate in densely populated spectral environments, where numerous signals can interfere with the target signal, making the signal of interest hard to delimit [42].
Ultra-wideband radars, known for their high resolution and wide bandwidth, present unique challenges for signal processing. The wide bandwidth provides UWB radars with exceptional detection capabilities, even in cluttered environments. However, this sensitivity also makes them more prone to environmental interference, emphasizing the need for advanced decluttering techniques.
Several established decluttering methods have been studied to suit different applications and environments. One of these is "moving average filter", a widely recognized method that computes the average of data points over a specified interval, thereby reducing short-term fluctuations and emphasizing longer-term trends, resulting in particularly clear signals with a constant mean [5]. Exponential smoothing, by contrast, employs a weighted average where more recent data points have a higher weightage, making it adaptive to rapid changes in the signal environment [20]. The Butterworth filter, another noteworthy technique, is characterized by its flat frequency response in the passband. It serves to minimize frequency distortions, making it particularly adaptable for radars that operate over a broad frequency range, such as UWB radars [10].
The next figure shows the spectrum of different decluttering techniques applied to a radar sample.

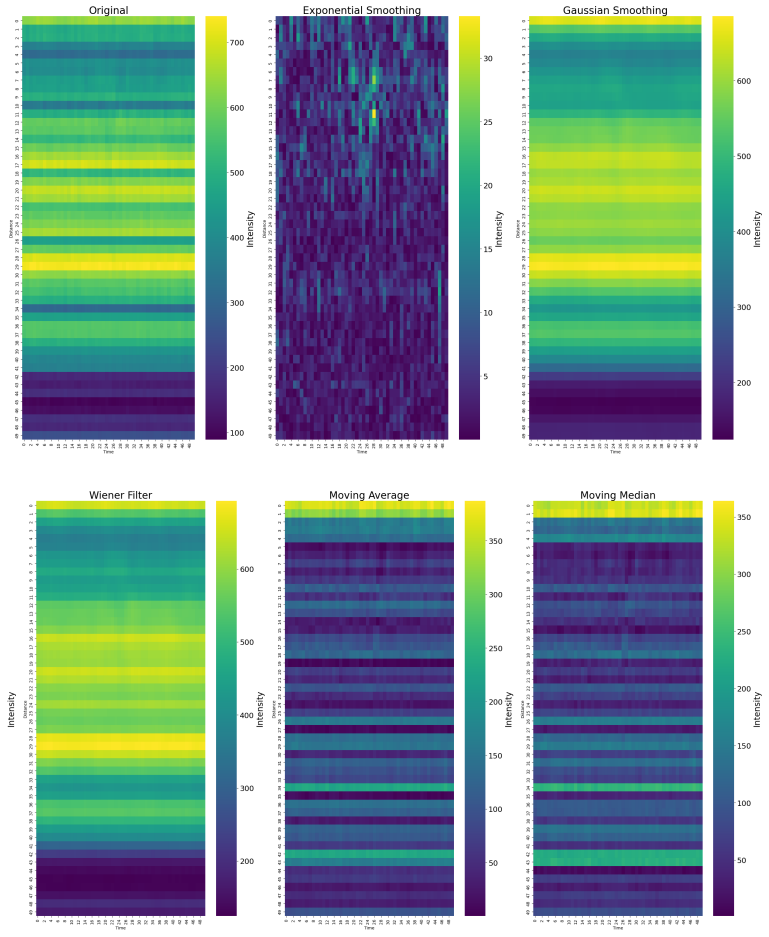


Figure 3: Spectrum of decluttering methods of a fall

2.9. Related work

Fall detection has been widely studied in the literature, Yu described a tripartite division of fall detection methods: wearable device-based, ambiance device, and vision-based [46]. Each category has its advantages and disadvantages. Wearable devices, including accelerometers and gyroscopes, are cheap and easy to set up and operate, but they have the main problem of being intrusive: the individual has to constantly wear them. Ambiance devices use pressure to detect a fall, such as vibration sensors on the floor or bed sensors under mattresses. They are cheap and non-intrusive yet often inaccurate - it's impossible to know if the pressure on the device is caused by the patient falling or by another object.
Vision-based methods, as the word suggests, leverage cameras, and have the advantages of being non-intrusive, the accuracy depends on the algorithm, but the installation process is complex, the devices used can be expensive and there is an intrinsic privacy problem.
An example of a fall detection solution with wearable sensors is the work conducted by Ozcan et al [29] in which wearable devices such as smartphones and tablets are equipped with cameras and accelerometers, utilizing data collected by both to detect a fall. This solution is advantageous in that the detection of a fall is not limited to a controlled environment, contrary to UWB radars. Moreover, the fall can be detected anywhere, as long as the subject carries the device. Other studies use accelerometers; Kulurkar et al. for example, developed an ad-hoc low-power device with embedded three-axis accelerometers and reached 95% of accuracy in detecting falls [23].
A solution to the problem using pressure devices has been conducted by Jeon et al [22] who employed a pressure-sensing system using a triboelectric nanogenerator (TENG) array, which exploits a nanostructure to convert ambient mechanical energy into electrical output. In the research study, 15 nano cells were collocated next to the bed on which the individuals fell. This system achieved a classification accuracy of 95.75% in identifying real falls.

Remarkable studies using cameras include a fall detection method using k-nearest neighbor and camera [16]. In particular, a "motion history image" of the movement is constructed from the images and then the fact that the shape of the body changes when a person falls is leveraged. Hence, the ratio between the minor and major axes of the body is used to decide if a fall occurs, along with a pre-determined threshold. Another worthy study uses dynamic images, which are an amalgamation of a number of sequential frames in a video. These images have the ability to simultaneously capture both the appearance and temporal evolution of information in a video by employing a rank learning method [14].

Another study uses a combination of neural network and radar data to make fall detection. The microwave radar operates at 24GHz and is attached at a height of 1.5m from the floor. The linear frequency modulation continuous waves (LFMCW) are transmitted and received by the radar. An optical camera (1920x1080 resolution, 60 fps) captures images of human motions frame by frame. The detection results of the radar and optimal camera are then fused to ensure low false alarms, which makes the fall detection system more efficient and robust. Furthermore, CNN is adopted for classification and recognition. Two kinds of CNNs based on Alex-Net and single shot multi-box detector Net (SSD), which uses bounding boxes, are employed to classify the features. This study has outstanding results in a controlled environment with an ad-hoc dataset, reaching 99% accuracy [48].

All the studies previously mentioned use different devices than the ones exploited for the proposed solution.

More recently, UWB radars have already been used to develop solutions to the problem of detecting falls. It's worth mentioning the work from M. Noori et al [26], who use UWB radar data collected through a robot to monitor and observe subjects from a specific distance (1.5-2.0m) and a special type of recurrent neural network, long short-term memory (LSTM). The results are exceptional, reaching an accuracy of 99.6%. Another interesting study using UWB technology [4] reaches 80% accuracy using the random forest method on an ad hoc dataset, collected with 10 people performing 15 different ADLs (activities of daily living), such as bathing or showering, dressing, getting in and out of bed or a chair, walking, using the toilet, and eating, in a 40 meters apartment.

Although all these solutions demonstrated that fall detection can be effectively automatized, none of them studied and developed a solution aimed at IoT (hence privacy-centric, cost and computationally efficient), deployed it on a microcontroller with limited resources, and used UWB radar data, as in the proposed algorithm.

3. Problem formulation

The primary objective of this thesis is to develop a neural network for fall detection using UWB radar data for IoT, and deploying it on a microcontroller with limited resources.
More formally, this problem can be reformulated as the design of a classifier able to map a radar recording into its label: let $s_t \in \mathbb{R}^{N \times M}$, with $M, N \in \mathbb{N}$, be the signal received by the receiving antenna of the UWB radar, being N the number of scans or pulses emitted by the UWB radar and M the number of spatial "bins", which is the number of "quantized" distances in the acquisition range, and $s_t \in S$, where S is the set of all radar acquisitions. Furthermore, $s_t[i, j]$ with $i \in \{1, ..., N\}$ and $j \in \{1, ..., M\}$ is the energy acquired by the receiving antenna at the i-th scan at the j-th bin. The problem aims to map s_t to its label y_t, where $t \in T$ is the set of labels, being $T = \{fall, non - fall\}$. In particular, the classifier has to occupy a memory M with $M <= A$ where A is the available memory on the microcontroller.

$$y_t = f(s_t) = \begin{cases} 0 & non\text{-}fall \\ 1 & fall \end{cases}$$

4. Devices used

For the development of the whole project three devices have been used: two different UWB radar models, each one used to collect a dataset, and an Arduino microcontroller, on which the proposed solution has been deployed. It's worth mentioning that the microcontroller is used as a placeholder for the UWB radar, to demonstrate that is possible to deploy the solution on a limited resources device, hence proving the feasibility of bringing the solution on the radar that will be used in real-world application as in nursing homes.

4.1. NXP Semiconductors UWB radar

This UWB radar used is produced by NXP Semiconductors [35], and has been used to collect the main dataset of this thesis, used to develop the proposed solution. It works on UWB bands from $6.24GHz$ to $8.24GHz$ and supports the detection and relative location of moving objects based on the changes in the reflected signal, measured by means of channel impulse response (CIR) estimates.
A sequence of modulated pulses is transmitted and the receiver is continuously listening to any reflections from objects in the surroundings for the duration of the frame. The length of the computed CIR estimate is a function of the time taken for the pulse to reflect back from the object. The magnitude of the received signal is a measure of the strength of the signal reflected by the object and depends on the reflected object's properties such as size, material, and angle of incidence. Moving objects cause a change in the phase if the reflected carrier due to the Doppler effect. Furthermore, it has an average power consumption of $185mW$. At a hardware level, this radar module presents one transmit channel and two receive channels. The transceiver is operated in a full-duplex mode, meaning that the receivers are active while the transmitter is sending the pulses.

4.2. Aria Sensing UWB radar

This device has been used to collect the dataset used for the first part of the development of the thesis; it is a UWB radar produced by Aria Sensing [36]. It's called $LT103OEM$ and is a high-precision, compact, and lightweight ultra-wide radar module developed for indoor applications, integrating high-end antennas, the signal processing unit, and the communication interface. It operates at frequencies between $7.3GHz$ and $8.5GHz$, the maximum detection range is 10 meters, and has a maximum power consumption of $150mW$, furthermore, the embedded antenna has an aperture of $\pm 60°$ on the azimuth and $\pm 60°$ of elevations, resulting in a cone of $120°$ [37]. The LT103OEM is a highly configurable UWB radar. This module combines a full UWB transceiver and an onboard microcontroller unit (MCU). The module is targeted for applications like presence detection,

position tracking, breath detection, and analysis. The operating principle of the system is based on the direct readout of the backscattered pulse. The transmitter emits pulses that travels into space and hit the targets that are in an active area of the radar. The targets reflect part of the incoming energy (echoes) backward to the radar module (as shown in image 4). The receiver converts the incoming signal to digital data, these data are provided to the MCU and processed according to the application.

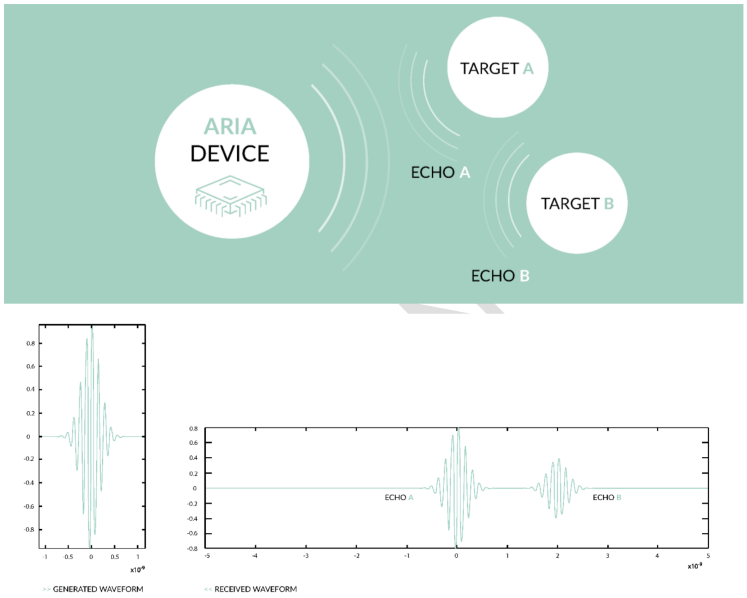


Figure 4: The basic principle and the waveforms. Generated pulse at the transmitter (left) and generated echoes from targets (right)

4.3. Arduino nano 33 BLE sense

The microcontroller used, on which the proposed solution has been uploaded, is the Arduino nano 33 BLE sense, it has an ARM Cortex M4 MCU running at 64MHz and only 256KB of SRAM [43].

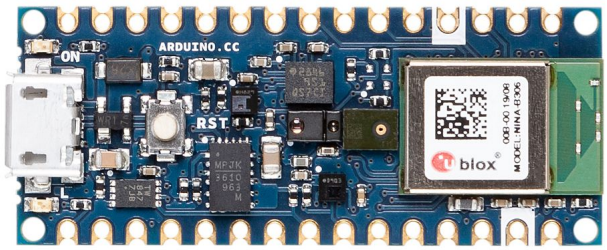


Figure 5: Arduino nano 33 BLE sense

4.4. Research direction

The problem of detecting falls has been previously studied in the literature, but in the specific considered case, it presents multiple challenges. At first, the developed solution needs to meet the requirements of the chosen microcontroller, such as the computational load and memory footprint. Then, there are challenges concerning the environments in which the neural network is trained since the UWB radar can become extremely attached to environments.
Moreover, the dataset has extreme importance in the development of these solutions, in particular, the bigger the dataset, the more the network can learn the characteristics of different types of falls, in different environments, of different subjects. This aspect doesn't concern only fall patterns, but also other human activities.
Another challenge on this problem is the regards the data preprocessing and the model architectures. The studies on these aspects can be further extended and optimized.
Therefore, the following research questions can be posed:

- Is it possible to exploit TinyML to analyze more complex UWB data at an IoT level?
- What is the best decluttering method?
- What is the best model architecture?
- How can the solution be extended not only to detect falls but any human activity and deploy it on an IoT device?
- What are possible pre-processing techniques that enable deep learning for this task?
- What are the environmental limitations?
- What is the best trade-off between memory required (possible device used) and model architecture?
- What is the best quantization process (at a training and evaluation level)?

5. Proposed solution

The proposed solution is an algorithm that takes the UWB radar data as input, preprocesses them, and subsequently gives them as input to a pre-trained neural network model which classifies them as "fall" or "non-fall". More formally, let $s_t \in \mathbb{R}^{N \times M}$ and $s_t \in S$, where S is the set of all radar acquisitions, with $M, N \in \mathbb{N}$, the signal received by the receiving antenna of the UWB radar, being N the number of scans or pulses emitted by the UWB radar and M the number of spatial "bins", which is the number of "quantized" distances in the acquisition range. This signal is the input to a preprocessing function Θ_p, and its output, $\Theta_p(s_t)$ is the input to the classifier Φ which is composed by a feature extraction ϕ_f block and a classification block ϕ_c which classifies the input to the output class y_t, with $t \in T$ being T the set of classes, furthermore $T = \{fall, non-fall\}$.
The constraints about Φ are imposed by the microcontroller for on-device implementation, in particular, the size of $\Phi + \Theta_p(S) <= M$, with M the memory available on the microcontroller.

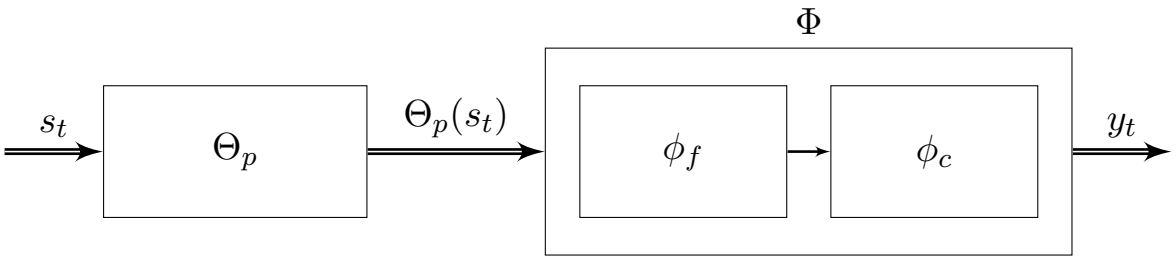


Figure 6: Proposed solution process

5.1. Preprocessing and decluttering

The proposed solution utilizes data collected with the NXP Semiconductors radar. The preprocessing stage for these data is particularly relevant because the significant signal received by the radar was subtle, and the noise received from the static environments was relevant. Each data sample S has initial dimensions of $256x128$. One could also view it as $128x128x2$ given that each pulse provides 256 values, comprising both amplitude and phase for every spatial bin. As part of the preprocessing, the norm of every value is determined, resulting in a 128x128 matrix, denoting time-space dimensions. Decluttering, a common technique in signal processing aimed to remove or reduce of unwanted interference, is then applied. Different decluttering techniques were employed and tested out, to find the best possible solution to the problem, but the proposed solution uses decluttered data with moving average filter [40].
Moving average filter calculates the average value of a predefined window of data points and then subtracts this average from the current data point. This technique emphasizes sudden changes in value which deviate significantly from their local average. This filter is applied over both dimensions, but the operation is inherently local, operating on a window defined by `window_size`. Thus, for every data point, it considers its neighboring points in both time and space dimensions before determining its new value. This ensures that any significant spatial-temporal event that deviates from its local mean gets highlighted. More formally:

$$M_{i,j}(R, w) = R_{i,j} - \frac{1}{w^2} \sum_{m=i-w}^{i-1} \sum_{n=j-w}^{j-1} R_{m,n}$$

where $M_{i,j}$ is the moving average filter function applied on the i^{th} time point at the j^{th} spatial bin of the radar matrix R, and w is the window size, set as 3. Moving average filters are widely recognized for their simplicity and effectiveness in removing short-term fluctuations.
Post-decluttering, the absolute value is applied to radar data, this is done because we want to highlight the movements of the individual and not its direction. In fact, before computing the absolute value, a subject

moving away from the radar would result in a diminishing intensity signal, instead, we want to highlight these patterns, since falls can happen also in the opposite direction with respect to the radar.
Then, the radar data is subject to normalization, which adjusts each radar data sample, ensuring that the values lie between 0 and 1. This is a foundational step when preparing data for neural network training. In the context of deep learning with UWB radar data, the importance of normalization becomes more pronounced due to the intricate and subtle patterns these data might contain. Neural networks rely on gradient-based optimization methods, such as gradient descent, to adjust their weights. When features across different input dimensions are of varying scales, the gradients can oscillate and diverge, leading to wrong and inefficient learning. Moreover, in deep networks with multiple layers, excessively large or small data values can result in very large or very small activations. As the data propagates through layers, this can lead to the phenomenon of exploding or vanishing gradients, making the network challenging to train or even causing training to fail. Additionally, normalization ensures a more symmetrical error landscape, enabling optimization algorithms to converge faster to the loss function's global minimum. For networks utilizing activation functions like sigmoid or tanh, normalization prevents the saturation of these functions. In the saturation regions, the derivatives are nearly zero, which can drastically slow down or even halt the learning.
Each data matrix is then cut to consider only the relevant part of the recording, in particular, the closest and furthest spatial bins didn't carry relevant information about the subject's movements, resulting in a [56, 107] shape, where 56 denotes the space dimension and 107 the time dimension.

5.1.1 Model architectures

It's worth remembering that one of the main goals was to develop a small, low parameters network and deploy it on microcontrollers with very limited memory. This was a main constraint on the possible architectures employable for this research because all of the well-studied neural networks in literature are too complex and their sizes are too big for deployment on IoT devices. Hence, the need to create a network from scratch specifically studied for this project.
Among the numerous architectures tested, the ones in which an inception module, inspired by InceptionNet but simplified, performs well [41]. An inception module is an element where multiple parallel branches with different convolutional and pooling operations are present, which are then concatenated. It allows the network to learn multi-scale features simultaneously at each layer.
Furthermore, different activation functions were implemented and tested. In particular, an interesting experiment that showed good results was the juxtaposition of `'tanh'` and `'relu'` activations within the same network.
Batch normalization, employed in multiple networks, also helped to develop the best network. The use of BatchNormalization brings several benefits to the training process. Normalizing activations can smooth the optimization landscape, leading to faster convergence during training. It also allows the use of higher learning rates, which could be problematic without normalization due to issues such as exploding or vanishing gradients. Furthermore, BatchNormalization can also have a mild regularizing effect, sometimes reducing the need for other regularization methods like dropout [21]. This process of finetuning the network, yet keeping it simple, brought to the creation of a few networks that had good performances.
The proposed solution uses Fall-Net, a neural network with 45.601 parameters with an "Inception Module" characterized by parallel branches of different convolutional operations. Within this module, there's a 1x1 convolution, a 3x3 convolution following a 1x1 convolution, a 5x5 convolution following a 1x1 convolution, and a 1x1 convolution following a max-pooling operation. These branches are then concatenated to form the module's output. The intention behind this parallel structure is to allow the model to learn different spatial hierarchies in the input data simultaneously. The 1x1 convolution in the Inception module reduces dimensionality while preserving spatial information, aiding to computational efficiency and allowing the model to learn cross-channel correlations. After the Inception module, the network flattens the output, passes it through a dense layer, includes a dropout for regularization, and finally outputs through a `sigmoid` activation function.

A network worth mentioning is Fall-Net-2, a traditional-style CNN with 15.297 parameters. It has an Input layer designed to accept 2D UWB radar data, represented as a $56x107$ 2D matrix. The architecture then has its convolutional phase with a Conv1D layer that utilizes 8 filters of size 3 and the `'tanh'` activation function, followed by another convolutional layer which leverages 16 filters of size 6, this time with a `'relu'` activation function, enhancing the model's ability to extract more intricate patterns based on preceding layer outputs.
To maintain the stability and accelerate the training, BatchNormalization layers have been introduced post each convolution. Then, MaxPooling1D layers have been added, which serve to down-sample the feature maps, compressing the spatial dimensions. After the convolutional operations, the data is transformed into a 1D format via the `Flatten()` layer, preparing for the fully connected phase. The architecture ends in the output layer, a single dense neuron with a `sigmoid` activation function. The next images show the architectures of Fall-Net.

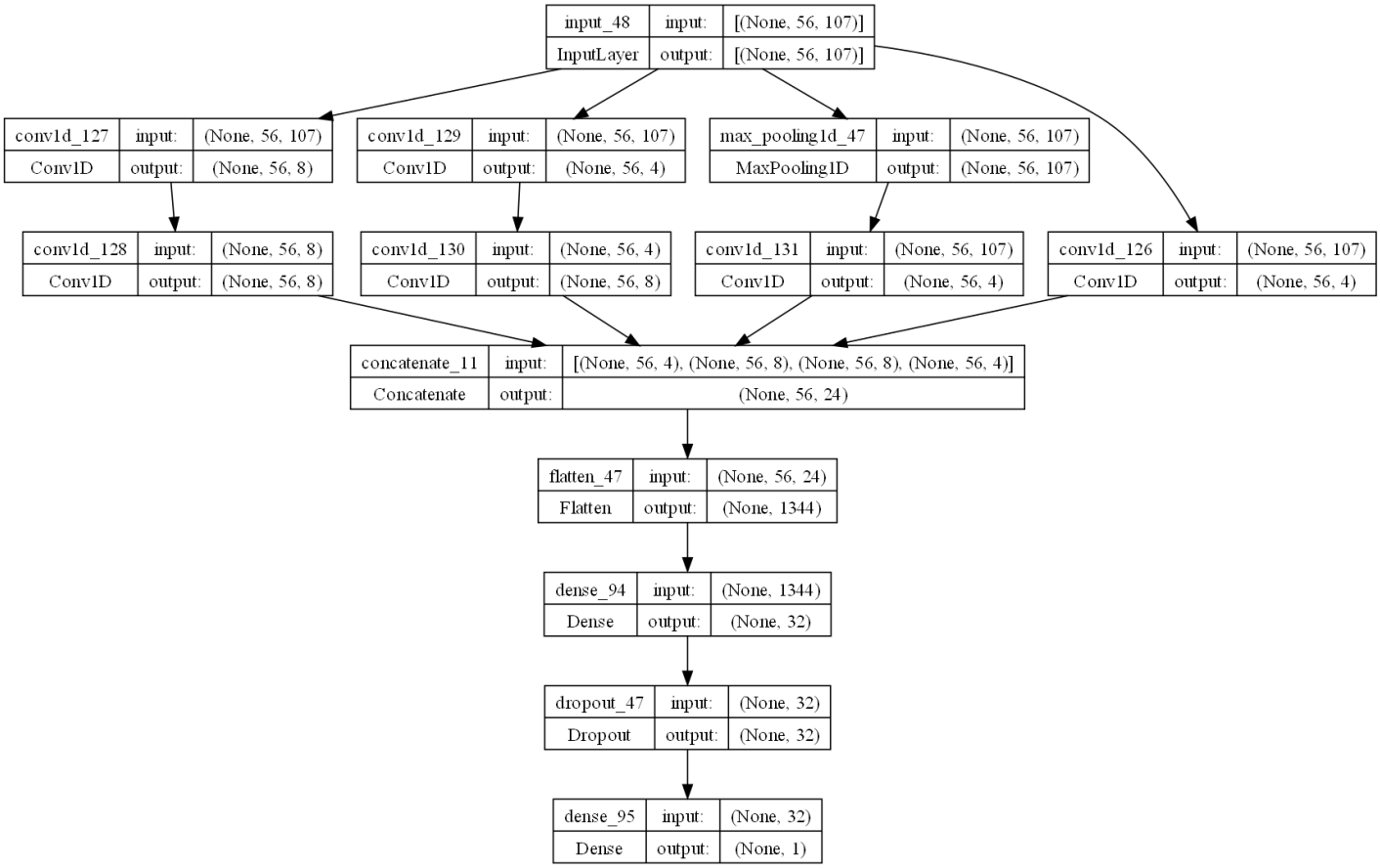


Figure 7: Structure of Fall-Net (proposed solution)

6. Dataset

For the development and assessment of a fall detection algorithm with UWB radar, two tailored datasets were utilized. The main one, collected with the NXP Semiconductor's radar, encapsulates recordings from various room settings, highlighting the algorithm's adaptability to diverse environments. Each entry consists of a matrix representation which is the received signal by the radar's antenna. The specific challenges introduced by certain room characteristics offer valuable nuances, ensuring the dataset's depth and applicability for this research. A secondary dataset, which will be presented, has been used exclusively to conduct experiments that helped with the development of the final proposed solution. In particular, this secondary dataset was used in the first part of the development of the thesis, and had various limitations, for example, only a few fall recordings were present and were collected in one environment only. The need for another ad-hoc, improved dataset was present, hence, the following one.

6.1. NXP Semiconductors radar dataset

The primary dataset used for this research was collected by me, with help from various people, which gave an important help, especially because the dataset became more heterogeneous, from the movement of the people falling and doing different activities to the dimensions and shapes of the bodies themselves. This dataset was instrumental in our study and consistently yielded optimal outcomes.
For the data collection process, the UWB radar from NXP Semiconductors was used, which operated at a frequency of 10Hz and each recording had a duration of 12.8 seconds. This gave us 128 distinct pulses per recording. The radar is able to capture both the real and imaginary components of a complex number. In the context of radar signals, these components provide insights into the amplitude and phase of each signal for each of the 128 spatial bins. To put it in perspective, each spatial bin roughly covers a distance of 5cm, allowing our radar to perceive up to a distance of 6.4 meters in a 120-degree conical sweep.
In terms of data representation, each of our recordings can be interpreted as a $128x128x2$ matrix, where $128x128$ denote the spatial bins (distance) and the number of radar pulses (time), and 2 represent the amplitude and the phase of each.
It's a general practice in radar signal processing to represent signals as complex numbers, as they effectively capture amplitude (magnitude) and phase (directional) information. The real component typically encapsulates the amplitude, while the imaginary component conveys the phase. The overall strength or "loudness" of the radar's return signal can be understood through the magnitude calculated by the formula $|c| = \sqrt{a^2 + b^2}$, which is what has been computed in this case.
To be clear, each spatial bin returned the real and imaginary components of an imaginary number, hence each imaginary number corresponds to a specific instant and distance of the recording.
Each recording, after the norm between each real and imaginary part of each complex number has been calculated, converges to a 128x128 matrix, demarcating time and space. In UWB radar data analysis, preprocessing often entails decluttering to remove undesirable static noise or interference. This not only refines the quality of data but also ensures precision in subsequent analysis. Once decluttered, the data is trimmed to emphasize the relevant spatial and time intervals.
Our dataset is diverse, consisting of four distinct room environments. Each room brings its unique architectural structure, varying in terms of size, layout, and building materials.
In particular, one room is narrow without the wall facing the radar, resulting in the radar signal returns more precise since the static component of the wall is not present.
Another environment presents glass and metal elements that could induce significant interference in radar signals.
Cumulatively, the dataset collected consists of 1656 recordings, divided into eight different classifications. A comprehensive breakdown of these categories is presented in the following image.
For the collection of this dataset, the radar was positioned in different positions in the room, pointing to different directions. Furthermore, the radar was never pointing in the vertical direction and was always positioned on

Figure 8: Radar positioning in one of the dataset environments

top of a tripod. The purpose of moving the radar multiple times was to make the network learn the interesting features of the radar signals, and not some useless ones such as the architecture of the room, becoming too attached to the environment.
As you can notice in the image, the activities in the datasets can be divided into 8 classes: `non-presence`, `sitting and moving around`, `standing and moving`, `chair still`, `standing still`, `fall`, `pick up something`, `laying on the ground`. These classes were chosen accurately to represent the most common activities of a person, with special consideration to the activities of elderly people in nursing homes.

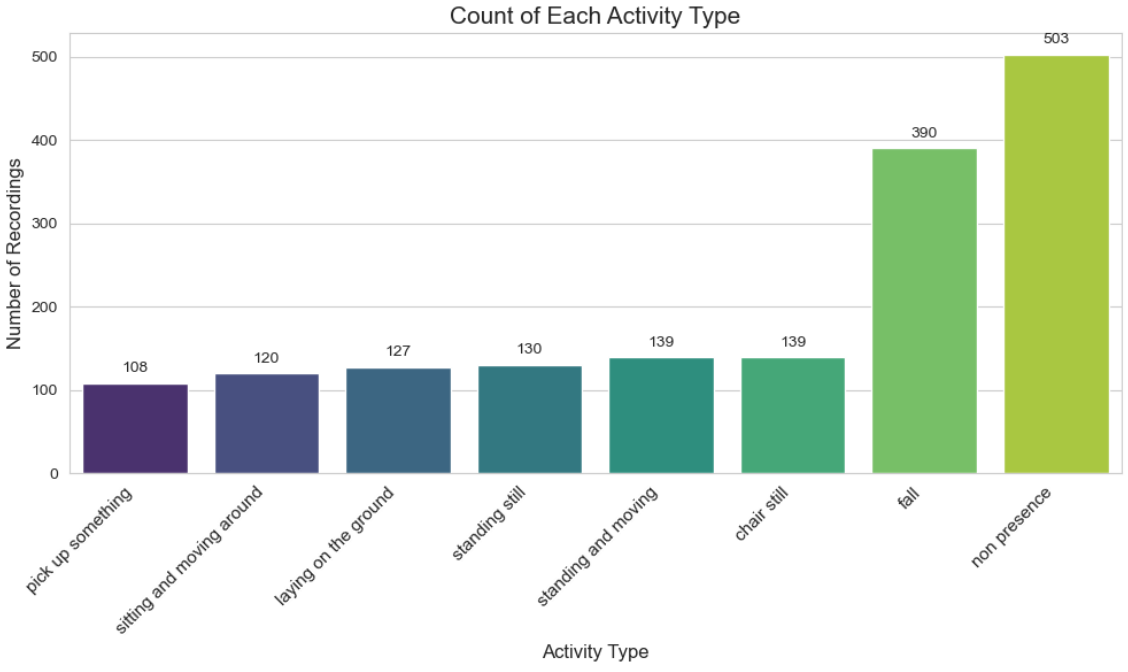


Figure 9: Class division of the dataset

The class labeled as "laying on the ground" was initially thought of as the subject sleeping on the bed, but since for some recordings were collected using yoga mats, which eventually evolved into a bed mattress positioned on the floor, the class was labeled so.
A class worth mentioning is "pick up something", it was chosen because the act of picking up something from the ground, which was in some cases a quick gesture towards the floor, has many commonalities with a fall. The recordings of this class consist of the subject walking around or standing at some point in the room and then crouching down to pick up an object. This class was of significant importance because the networks trained learned the difference between a subject picking up something and a fall, which otherwise could have led to false alarms.
Concerning the falls, the corresponding recordings were conducted falling in multiple directions: falling away from the radar, toward the radar, from the right to the left of the radar, and vice-versa. Furthermore, both falls from standing and falls from sitting on a chair were conducted. In this way, the solution proposed is able to detect the most heterogeneous types of falls.

6.1.1 Environments

As previously mentioned, the environments in which the dataset has been collected have been chosen to test our algorithm on different environments to prove it can generalize.
Even though all the recordings are from rooms, each room is unique. They vary in size, shape, and the materials they're made of. One room, for example, is quite narrow. It also doesn't have a wall that would be in front of the radar. This means the radar doesn't get back the usual signals from that direction, making a person's movement in the room seem more accentuated to the radar.
A second room, instead, introduces difficulties in terms of refraction, since its presence of materials such as metal and glass.
Metal is a highly conductive material, which means it strongly reflects electromagnetic waves, such as those emitted by UWB radars. When a UWB radar signal hits a metal object, it can result in a phenomenon known as "multipath propagation." This is where the signal reflects off the metal and travels in various paths before reaching the radar receiver. These multiple reflected signals can interfere with each other and with the direct line-of-sight signal, causing distortion and reducing the radar's ability to distinguish between true and false targets.
Glass, on the other hand, can sometimes be transparent to certain frequencies but reflective or even absorptive to others, depending on its composition and the frequency of the radar signal. This can lead to partial reflections or signal attenuation, hence introducing complexities to the radar signal interpretation. For UWB radars, which operate over a wide range of frequencies, this behavior can be particularly unpredictable.
This room, where a glass and metal table and a metal column are present, creates an environment where the radar signal is scattered, refracted, or absorbed in unpredictable ways. This complicates the task of accurately interpreting the signals and can reduce the radar's effectiveness in detecting the intended target, be it a moving person or any other object.

6.2. Aria Sensing radar dataset

On this dataset, the first experiments were conducted and it wasn't collected by me.
The recordings were 18, with different durations based on the activity conducted by the subject. Furthermore, most of the recordings have a duration of 5 minutes but present also a few registrations with 1, 10, and 15 minutes durations for different activities involved. The few ones where one or more falls are present have 1 minute length.
In this dataset, the activities conducted are `'none'`, `'come-and-go'`, `'fall'`, `'agitazione'`, `'attività-normale'`, where `'none'` represents the empty environment, `'come-and-go'` the subject entering and exiting the area of detection of the radar, `'fall'` the subject falling, even multiple times in one recording, `'agitazione'` the subject sat and moving, and `'attività-normale'` the subject stood up and moving, for example walking

around.
The radar had a sample frequency of 10Hz and had a sight of 106 spatial bins. Considering that the radar has a maximum sight of 10 meters, each spatial bin spans for approximately $9cm$.
The precision was limited since each recording was then dissected into small 20 seconds windows. The main problem with this dataset was that the whole length of each recording was classified with one label, hence, all its sub-windows were classified the same, resulting in some miss-labeled samples. Consider the 1 minute recordings of the falls; each fall has a very short duration, around few seconds, and knowing that only a few falls are present in each registration, there must be some 20 seconds windows without the presence of any fall, therefore miss labeled.
Instead, the labels that corresponded to continuous activities didn't influence the research much; for example, `'attività-normale'`, or `'none'` recordings were loyal to their labels since all the other classes are continuous movements.
Multiple problems were raised with the use of this dataset, especially for the inconsistency and lack of data, in fact, the whole dataset results in 210 independent, 20 seconds recordings, and only 12 represent falls.

7. On-device deployment

The solutions have been tested on the Arduino nano 33 BLE sense lite [43]. It has an ARM Cortex M4 MCU running at $64MHz$ clock and 256 KB or SRAM.
The process of deploying the trained models on the microcontroller requires first saving the models as `.tflite`, then converting it into a header file, `model.h`.
A `.tflite` model is a file format used by TensorFlow Lite, which is a lightweight version of TensorFlow designed for mobile and embedded devices. The `.tflite` model represents a streamlined version of a TensorFlow model that has been converted and optimized for size, performance, and platform compatibility.
Converting a `.tflite` model to a `.h` file means transforming the model from its regular serialized format into a format that can be directly included and compiled with a C or C++ program, especially on platforms where standard filesystems might not be available or practical to use, like microcontrollers.
The `.h` model is uploaded to the device using the Arduino IDE framework, in particular, the C++ code developed for it specifies a `setup()` function which establishes a serial connection to communicate with the device. The model is then loaded into memory, and a check is made to ensure the schema version of the model matches what TensorFlow Lite expects. Next, an interpreter is initialized; this is the component that runs the TensorFlow Lite model.
Then, a `loop()` function runs indefinitely, typical of Arduino sketches, and waits for data to become available over the serial connection. When data arrives, the neural network's input is populated, and the model is invoked to perform inference. After the inference is executed, the prediction (output of the model) is sent back over the serial connection.
Once the model is uploaded on the Arduino microcontroller, a Python script establishes a serial connection to it and starts the communication, sending the samples to predict and receiving back the prediction of each.
Both Fall-Net and Fall-Net-2 have been uploaded on the microcontroller. The time of execution for the prediction for each sample on device is 19 milliseconds for Fall-Net and 35 milliseconds for Fall-Net-2.

```
105   // Provide input to the neural network and invoke
106   uint8_t* input = interpreter->input(0)->data.uint8;
107   for (int k = 0; k < 56 * 107; k++) {
108     input[k] = data[k / 107][k % 107];
109   }
110
111   Serial.println("Invoking the model...");
112
113   if (interpreter->Invoke() != kTfLiteOk) {
114     error_reporter->Report("Invoke failed");
115     return;
116   }
117
118   uint8_t output_value = interpreter->output(0)->data.uint8[0];
119   Serial.print("Prediction: ");
120   Serial.println(output_value);
```

Figure 10: Snippet of Arduino sketch that invokes the model and makes prediction on device

7.1. Quantization

The models and the data used for this process were quantized. In particular, an 8-bit full quantization has been performed. The quantization process was conducted to reduce the model's size to make each model fit into the microcontroller [3]. Since Arduino only accepts full-quantization models, which means that input, output, activation functions and weights need to be quantized, a full 8-bit quantization was introduced. The initial structure of the networks and size of the data were reduced drastically, from 32 bits to 8 bits. In fact,

initially, the data were of type `float32`, and have been quantized to `UINT8`. This process inevitably reduced the performance of the networks but was a forced decision for the on-device deployment.
Fall-Net-2 reduced its memory occupation from 68KB to 12KB, while Fall-Net from 186 KB to 44 KB.
Next, is the Python function used to quantize the model. It's worth mentioning that also the dataset used for testing has been converted to `UINT8`.

```
def save_quantized_model(model, representative_data, path):
    # Convert the model to the TensorFlow Lite format with quantization
    converter = tf.lite.TFLiteConverter.from_keras_model(model)
    converter.optimizations = [tf.lite.Optimize.DEFAULT]
    # Use representative_dataset to calibrate quantization
    converter.representative_dataset = representative_data
    # Ensure that if any ops can't be quantized, the converter throws an error
    converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
    # Set the input and output tensors to uint8
    converter.inference_input_type = tf.uint8
    converter.inference_output_type = tf.uint8
    tflite_model = converter.convert()
    # Save the model to disk
    open(path, "wb").write(tflite_model)
```

Noteworthy of the reported code is the representative dataset which is a subset of samples used to better calibrate the quantization process. Furthermore, the following unix command is used to generate a C source file that contains the converted TensorFlow Lite model as a header file.

```
xxd -i model.tflite > model.h
```

8. Experiments and results

This section shows and explains the experiments conducted, describing multiple preprocessing techniques, model architectures and tasks that helped to develop the proposed solution.

8.1. NXP Semiconductors dataset

The next tables report the results of the proposed solution (row highlighted in green) alongside other interesting results obtained with Fall-Net and Fall-Net-2 with different decluttering techniques on the NXP Semiconductors dataset. In particular, the columns of the table, from left to right represent, the model name and decluttering technique, the accuracy of the original model, the quantized accuracy using the quantized model and making inference directly on the microcontroller, memory occupation of the model, execution time to invoke the model, and make inference on device.

	Accuracy	Quantized accuracy	Memory	Exec. time
Fall-Net + MA (Proposed solution)	0.98	0.78	44KB	19ms
Fall-Net + MM	0.93	0.73	44KB	19ms
Fall-Net-2 + Raw data	0.94	0.76	12KB	35ms
Fall-Net-2 + MA	0.98	0.72	12KB	35ms

Table 1: Fall detection results

Multiple experiments have been conducted using different decluttering methods and model architectures. It's interesting to analyze the results accomplished. The following decluttering techniques have been tested on the radar data, which are briefly explained:

- **Exponential smoothing** [9]: the key idea behind this technique is to give more importance to the recent measurements and gradually decrease the weightage as we move backward in time. The smoothing factor alpha determines the weight of the present data point versus the historical data. It is applied in a progressive manner; it begins with the initial measurements and proceeds through the matrix, adjusting each value based on its immediate predecessor and the smoothing factor. As a result, the data in each spatial bin evolves smoothly over time, where sudden jumps or drops are mitigated, and gradual trends become more pronounced. Exponential smoothing is commonly used in time series forecasting due to its ability to adapt to sudden changes without being overly sensitive to random fluctuations. Formally, the exponential smoothing decluttering function E is:

  $$E(x_t, x_{t-1}, \alpha) = x_t - (\alpha x_{t-1} + (1-\alpha)x_t)$$

  where x_t is the energy acquired by the receiving antenna at time t, x_{t-1} is the smoothed value from the previous step and α is the smoothing factor, set to 0.9. This function is applied row-wise, meaning that aims to smooth over space, considering each row (time) independently.

- **Moving median filter** [38]: similar to the moving average filter in operation, the moving median filter subtracts the median value of a window of data points from the current point, rather than the mean. The median, being a robust statistic, can be more resistant to outliers. Thus, this filter can retain sharp changes or transitions in the data that might otherwise be smoothed out by a mean-based approach. As with the moving average filter, the operation of the moving median filter in the code spans both time and

space dimensions. For the radar matrix R, the formula is:

$$M_{m,i,j}(R, w) = R_{i,j} - \text{median}\left\{R_{(i-w):i,(j-w):j}\right\}$$

where $R_{i,j}$ signifies the i^{th} time point at the j^{th} spatial bin and w is the window size. The window is of size 3, which means the median is computed using a 3x3 neighborhood around each matrix element.

- **Gaussian smoothing** [15]: when applied to images, can be viewed as blurring. In essence, each data point (or pixel, in image processing terms) is replaced by a weighted average of its neighbors, where the weights are determined by a Gaussian function, it is performed using the equation:

$$G(R) = R * g(\sigma)$$

where R is the radar matrix, $*$ denotes the convolution operation, and $g(\sigma)$ is a 2D Gaussian function with standard deviation σ.

- **Wiener filter** [8]: it aims to estimate the most probable "true" radar return at each time point within every spatial bin, based on the observed data and the statistical characteristics of the noise and the underlying signal. Its equation is:

$$W_{i,j}(R, w) = m_R + \frac{\sigma_R^2}{\sigma_R^2 + \sigma_N^2}(R_{i,j} - m_R)$$

where $R_{i,j}$ represents the value at the i^{th} time point at the j^{th} spatial bin of the radar matrix, m_R is the local mean of the radar matrix computed within a window w around the coordinate (i, j), σ_R^2 represents the local variance of the radar matrix within the same window w, σ_N^2 denotes the variance of the noise, which can be estimated from regions in the radar matrix known to only contain noise or based on prior knowledge, and w is the window size used to estimate the local statistics.

It's reasonable to agree that the filters that had the most consistency over the models and the tasks are exponential smoothing, moving average, and moving median. The next figure shows a comparison of the decluttering techniques reporting the accuracies on the test set of Fall-Net.

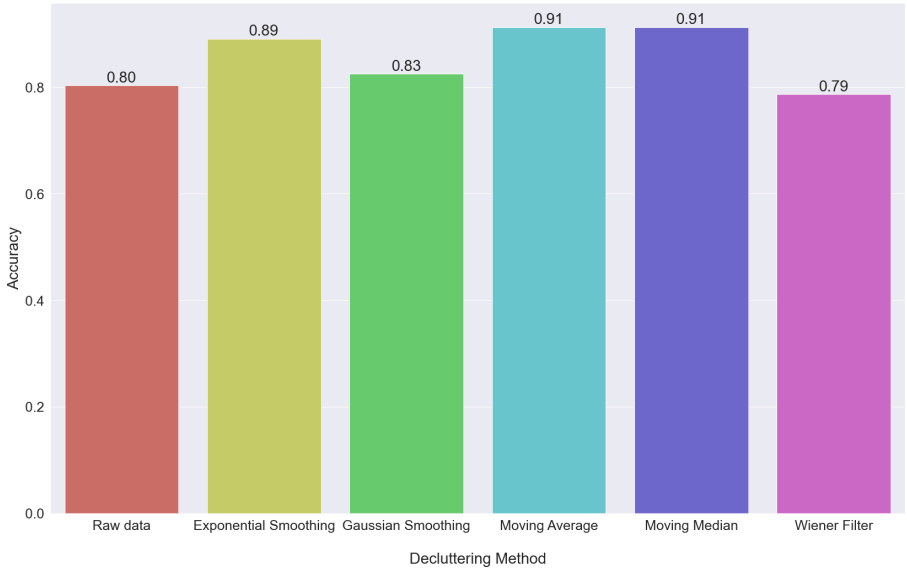


Figure 11: Comparison over decluttering methods

For Fall-Net and Fall-Net-2 trainings, Adam optimizer has been used with an initial learning rate of 0.001 and `binary_crossentropy` loss. To ensure effective training, `early stopping` is implemented, monitoring the validation loss and halting if no improvement is seen for 100 epochs. Alongside, a 'learning rate scheduler' adjusts the learning rate based on performance, reducing it by a factor of 0.2 if the validation loss stagnates for 10 epochs, with a floor set at 0.0001. The training runs for 5000 epochs using batches of 32 samples.

Although the primary goal of this research is fall detection, experiments have initially been done for presence detection, a binary classification task on the same dataset where all the classes in which an individual is present have been given a `presence` label. Also for this task, both Fall-Net and Fall-Net-2 have been deployed on device, and inference was made directly on it, to have a comparison between different tasks, model architectures, and decluttering techniques. The next table shows the results of this task. In particular, the best result is achieved with Fall-Net-2 using exponential smoothing decluttering technique, reaching 0.79 accuracy on the test set on device (highlighted in green).

	Accuracy	**Quantized accuracy**	**Memory**	**Exec. time**
Fall-Net-2 + EXP	0.86	0.79	12KB	35ms
Fall-Net-2 + MA	0.98	0.74	12KB	35ms
Fall-Net + MA	0.91	0.69	44KB	19ms
Fall-Net + MM	0.83	0.77	44KB	19ms

Table 2: Presence detection results

The confusion matrix below shows the results on the testing set of Fall-Net-2 using exponential smoothing on presence detection task.

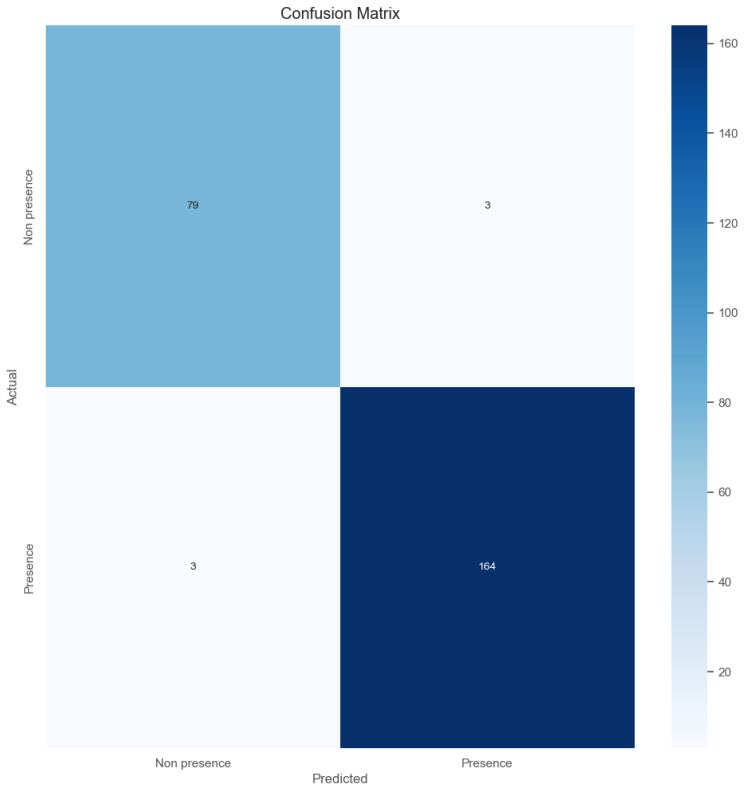


Figure 12: Presence detection confusion matrix of Fall-Net-2

Furthermore, the next table reports the results of different networks, which brought outstanding results, but since the sizes of these networks were too big, requiring more than 256KB on the microcontroller, they couldn't be deployed on device.

	Parameters	Accuracy	Deployable on device
CNN-Skip + Raw data	422.152	0.96	No
Inception-DenseNet + Raw data	10M	0.98	No

Table 3: Presence detection results on too complex networks to be deployed on device

In particular, CNN-Skip is a traditional-style CNN with 422.152 parameters, presenting identity skip connections [45], in which the output to a layer is concatenated and given as input to some further layers. The skip connections, achieved through concatenation, facilitate a shortcut for the gradient to flow during backpropagation. This helps in mitigating the vanishing gradient problem, leading to improved convergence during training. Moreover, these connections allow the network to retain fine-grained features from earlier layers, making the network more expressive.
Inception-DenseNet, instead, has 10 million parameters and is a combination of an Inception-style network (with multiple branches) and Dense-net, a particular network architecture that presents dense blocks, where the output of each layer is concatenated and given as input to all the subsequent layers in the dense block [19]. Inception modules, as previously mentioned, process data at multiple scales, capturing diverse features efficiently with reduced computational cost. Dense modules, by contrast, enhance feature reuse and improve gradient flow, resulting in efficient learning and reduced overfitting. By merging Inception and Dense modules, the architecture captures multi-scale features and reuses them, offering both efficiency and versatility in recognizing complex patterns.
The results reported with these two more complex networks are achieved on raw data, meaning that no decluttering is applied. All the decluttering techniques previously explained have been tested on these more complex networks, but the results were similar to raw data, and in some cases worse. This is reasonable and can be explained because CNNs, through convolutional operations, extract non-linearity from the data, and these bigger networks have more convolutional layers, allowing them to learn more complex patterns. Essentially, they can "filter out" the noise or irrelevant information through the convolutional operations. This ability is lost with smaller networks like Fall-Net, making decluttering an essential step for a trade-off between good performance and deployability on device.
It's worth mentioning some hyperparameters of these model's trainings: `binary crossentropy` is used as loss function. The optimizer chosen is 'Adam', known for its efficiency and adaptability in adjusting learning rates. To enhance the training process, two callbacks, namely 'EarlyStopping' and 'ReduceLROnPlateau', are integrated. 'EarlyStopping' monitors the validation loss and halts training if it doesn't improve after 100 epochs, ensuring that the model doesn't overfit and saves computational resources. 'ReduceLROnPlateau' watches the validation loss and reduces the learning rate by a factor of 0.2 if no improvement is seen over 10 epochs. This allows the model to make finer adjustments when it's close to converging, avoiding overshooting the global minimum. Training the model is set for a maximum of 1000 epochs, using batches of 32 samples each.
The following images show the architectural characteristics of CNN-Skip and Inception-DenseNet.

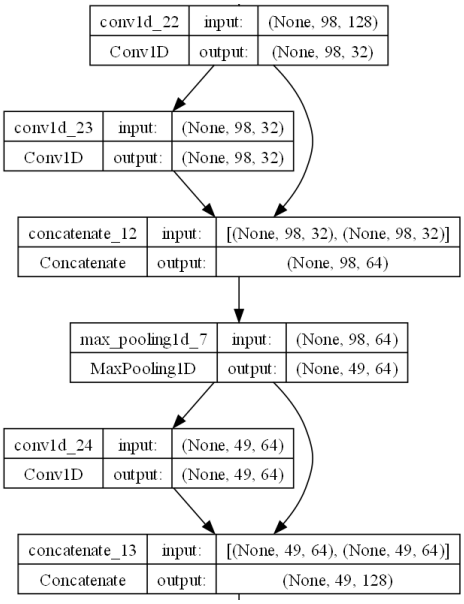


Figure 13: Skip connections of CNN-Skip

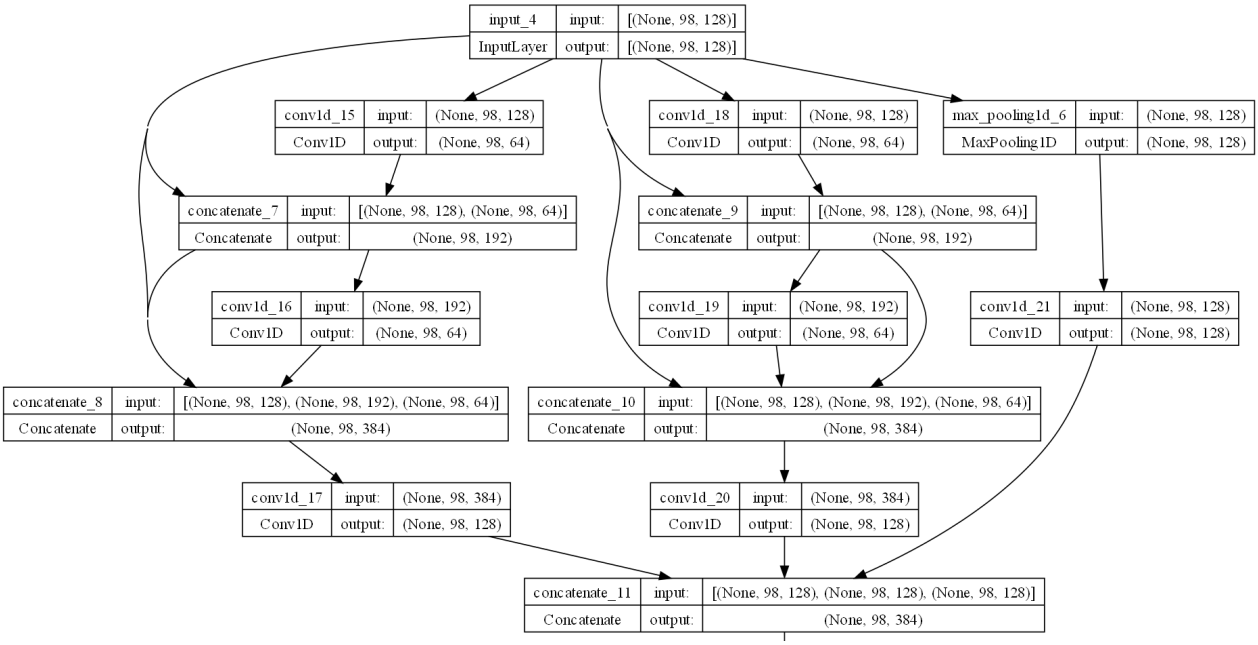


Figure 14: Inception and Dense module present in Inception-DenseNet

The dataset used, as previously mentioned, has been collected on subjects doing 7 different activities plus recordings of empty environments; for the fall detection task, the activities that weren't falls have been grouped under the "non-fall" label. For presence detection, all the activities were considered as "presence" and the empty environments as "non-presence".

Since the dataset was already categorized in different activities, we extended the task to Human Activity Recognition (HAR), which is a multi-class classification problem whose goal is to classify correctly the different activities, which are `non-presence, sitting and moving around, standing and moving, chair still, standing still, fall, pick up something, laying on the ground`.

The next table shows the results for Fall-Net, which reached 0.65 accuracy, and other tested architectures. The quantization process, in this specific case, made the performance drop, and this can be explained in multi-classification, `softmax` activation function is used, then, the `argmax()` among the probabilities is computed to

find the output class. These probabilities are often relatively close to each other, and converting from `float32` to `UINT8` inevitably transforms similar floating point numbers into the same int numbers. The next table reports the results of Fall-Net compared to more complex network architectures that resulted extremely precise on this problem, the activity-type detection task.

	Parameters	Accuracy	Quantized accuracy	Deployable on device
Fall-Net + EXP	45.832	0.65	0.36	Yes
CNN-Skip + Raw data	422.152	0.91	-	No
Inception-DenseNet + Raw data	10M	0.94	-	No

Table 4: Activity-type detection results

These results are reasonable and can be explained because bigger networks, with more parameters, are able to learn more complex and subtle patterns of the radar recordings and distinguish better between all different activities. For this task, Fall-Net has too few parameters to learn the relevant features of all the classes. The following confusion matrix shows the results of Fall-Net on HAR task.

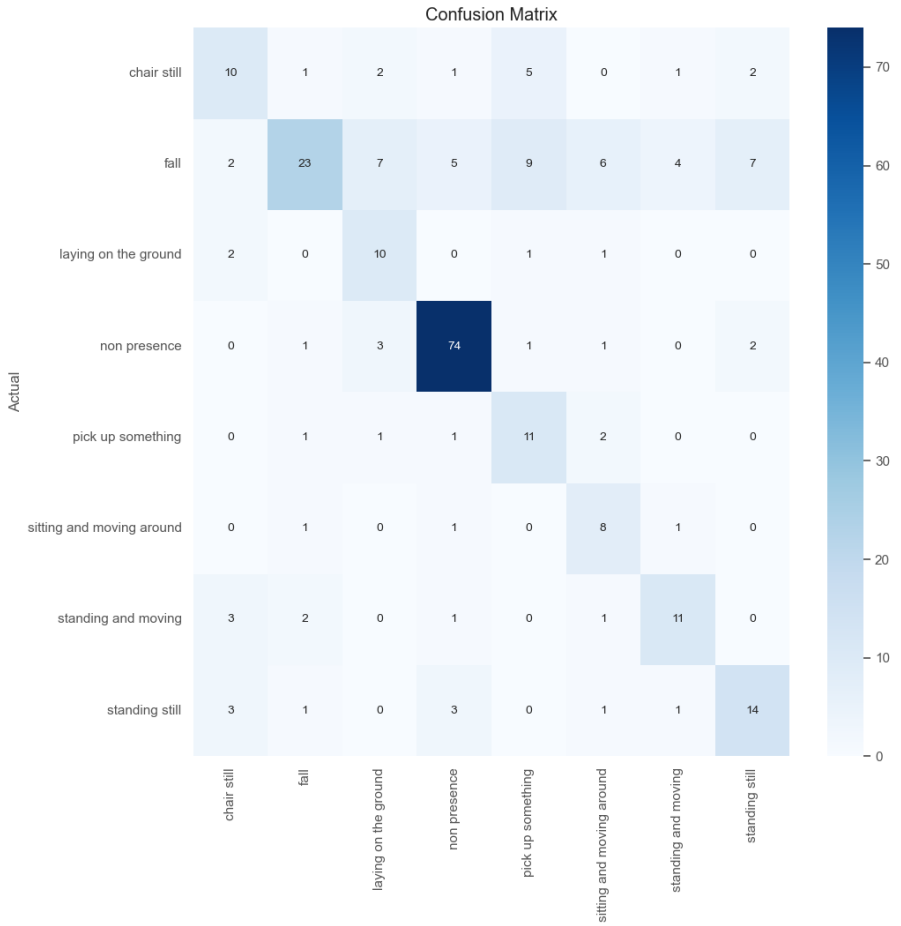


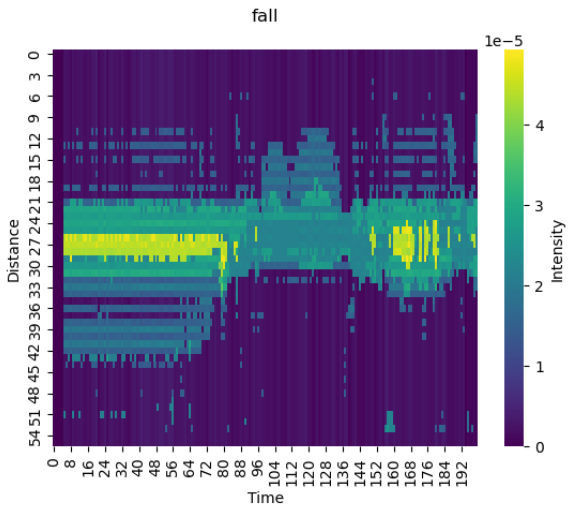
Figure 15: Activity-type detection confusion matrix of Fall-Net

8.2. Aria Sensing radar dataset

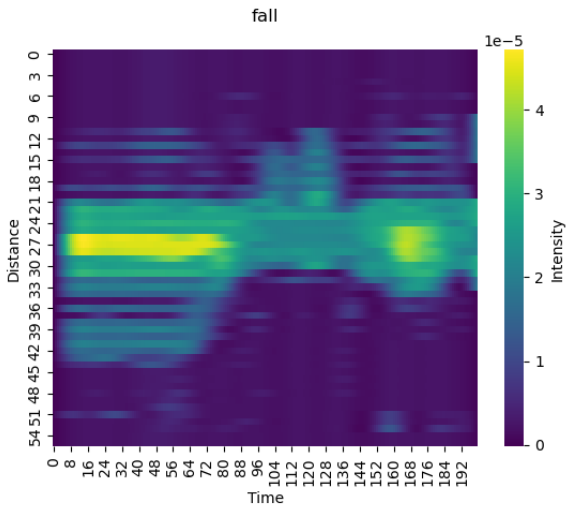
Although the proposed solution to this problem is based on the dataset collected with the NXP Semiconductors radar, the initial stage of this research employed another dataset, collected with a different radar, on which multiple experiments have been conducted. This dataset, which will be called Aria sensing dataset, consists of radar recordings collected with a frequency of the transmitting pulses is $10Hz$ and each pulse returned 106 complex values, one for each spatial bin. From these complex values, which enclose the amplitude and the phase of the radar return, their norm is calculated and considered for the subsequent process. Each recording ended up in a $56x200$ shape, where 56 represents the spatial dimension (the 106 initial spatial bins were cut to retrieve only the useful information of the movements) and 200 represents the time dimension (20 seconds).

8.2.1 Preprocessing

The preprocessing stage leverages the Butterworth low-pass filter. It is known for its maximally flat frequency response in the passband. Its primary purpose is to allow frequencies below a certain threshold to remain unaltered while attenuating or diminishing those frequencies above this threshold. This method is especially useful in filtering out high-frequency noise. By employing this filter, the data is rid of high-frequency perturbations, emphasizing the underlying genuine radar reflections and ensuring they are presented in their most unobstructed form [6].
When designing digital filters, such as the Butterworth filter, parameters are typically specified in the frequency domain, indicating desired behaviors like the cutoff frequency. The design process results in filter coefficients that dictate the filter's behavior in both time and frequency domains. These coefficients essentially serve as a set of weights. When applied to a time-domain signal, they emphasize or de-emphasize certain patterns in the data. High-frequency components, which represent rapid changes in a time-domain signal, can be recognized and attenuated based on these coefficients [39]. On the other hand, low-frequency components, manifesting as slower, more gradual changes, remain largely unaffected. Even though the design and understanding are in the frequency domain, the actual application of these filters frequently occurs in the time domain, leveraging the inherent frequency information found within the structure of time-domain data. There's no strict need to transform the data to the frequency domain to filter it, as time-domain filtering methods, like convolution, effectively address undesired frequencies by attenuating rapid temporal changes [28].
The Nyquist frequency also plays a pivotal role in this preprocessing [27]. Representing half of the sampling rate of the data, it's a fundamental concept that ensures no aliasing occurs, a phenomenon where higher signal frequencies become indistinguishable when sampled, and by considering the Nyquist frequency during filtering, the data integrity is upheld, preventing potential misinterpretations.



(a) Spectrogram of radar signal of a fall



(b) Spectrogram of filtered radar signal of a fall

Post-filtering, the data are normalized, scaling them in the interval [0, 1]. This is a common process for data used to train neural networks: it allows faster convergence and avoids saturation of the activation functions, which can lead to vanishing gradient. The data is then subject to clipping. During this process, a threshold derived from a percentage of the maximum value within each radar data sample is set. Any values that fall below this threshold are effectively set to a baseline, zero in this case. This technique is especially valuable when emphasizing stronger signals. By setting to zero weaker signals, which might be seen as noise or less significant, the analysis can focus on the most crucial and prominent features of the data, thereby offering clearer insights. Each transformation, from filtering out high-frequency noise, considering the Nyquist principle, normalizing the data, and emphasizing dominant signals through clipping, transforms the data such that only the relevant information is passed to the neural networks.

8.2.2 Results

The samples in which a fall is recorded are only a few in the Aria dataset; this fact makes the classes extremely unbalanced for fall detection. For this reason, the accuracy metric is not reasonable to use, since classifying all the samples as the majority class could lead to high accuracy, even if the network is not able to extract the relevant features. To solve this, the Area Under the Receiver Operating Characteristic Curve (AUC-ROC) metric was used. It offers a more comprehensive view of the model's performance across different decision thresholds. It quantifies the model's ability to discriminate between positive and negative classes, irrespective of their proportions. An AUC value of 1.0 signifies perfect classification, whereas 0.5 indicates no better performance than random guessing.
In particular, for the **fall detection** task, the best network was a MobileNet-style CNN with 145.000 parameters. It is a MobileNet-style because it uses depthwise separable convolutions, consisting of a depthwise convolution followed by a pointwise ($1x1$) convolution, which is a peculiarity of MobileNet. This architecture family is computationally efficient while still enabling the model to learn features. The AUC score on the test of this network was 0.75.
Although fall detection was a challenging task with this dataset because of the lack of samples, it's worth noting that, as the NXP dataset experiments, also presence detection and activity-type detection, experiments have been conducted on this dataset. For these two tasks, the dataset was a bit more balanced, hence the following results, shown in the next tables.
Concerning **presence detection**, Aria-CNN reported in the next table is a limited-size traditional-style CNN consisting of two convolutional layers followed by pooling, a dense layer, and a dropout for regularization, concluding with a sigmoid activation classification.
LSTM-1, instead, is an LSTM-based neural network. LSTMs are structures that present gates - namely the forget, input, and output gates - to control the flow of information and mitigate the vanishing gradient problem encountered by traditional CNNs. These gates ensure the network's ability to retain or discard information as needed, making LSTMs especially qualified at processing and remembering extended sequences. They have been first introduced by Hochreiter & Schmidhuber [18] and are known for their capability to retain long-term dependencies making them particularly congruent for sequential datasets like UWB radars. LSTM-1 architecture begins with an LSTM layer containing 16 units, capturing temporal dependencies in the data. This is succeeded by a series of dense layers, which facilitate the learning of intricate patterns from the sequential data. Dropout layers interleaved between the dense layers act as a regularization technique, reducing the risk of overfitting. The network concludes with a dense layer using a sigmoid activation, making the model suitable for binary classification tasks. This architecture effectively combines the power of LSTMs to process sequence data with the flexibility of dense layers to learn complex classifications. Both models are compiled with the Adam optimizer and binary cross-entropy loss. During training, early stopping and learning rate reduction are used for optimization. The models address class imbalances by computing class weights, ensuring that each class is adequately represented during training.

	Params	Test accuracy	Test loss
Aria-CNN	2.863	1.0	0.0014
LSTM-1	14.305	0.96	0.1

Table 5: Presence detection model comparison on Aria Sensing radar dataset

The Aria Sensing dataset, as previously mentioned, is divided into 5 classes: `'none'`, `'come-and-go'`, `'fall'`, `'agitazione'`, `'attività-normale'`, which are different subject's activities, making this an **Human Activity Recognition** problem. In the next tables are reported the results to this problem of two different model architectures. Aria VGG-style is a VGG-inspired CNN. Its model begins with two blocks of convolutional layers, reminiscent of the VGG's repetitive architectural style. Each of these blocks consists of a 1D convolutional layer with a rectified linear unit (ReLU) activation, followed by max-pooling to reduce spatial dimensions, and then a dropout layer to mitigate overfitting. Specifically, the first block has 32 filters and the second one has 64 filters, both using a kernel size of 3. The use of 'same' padding ensures that the spatial dimensions are preserved after the convolutions.
Post convolution, the model flattens its output to feed into fully connected dense layers. A larger dense layer with 128 units is employed with ReLU activation, again followed by dropout for regularization. Finally, a dense layer with 5 units and softmax activation to make multi-class classification.
LSTM-2 model initiates with an LSTM layer consisting of 64 units that process the sequence input and retain temporal dependencies, and it is set to return sequences, which means it passes the full sequence to the subsequent layer. This is followed by a Dropout layer to counteract overfitting. The second LSTM layer, consisting of 32 units, processes the sequences passed from the previous layer, collapsing the sequence information into a single vector, which is passed to the dense layers. The subsequent dense layers, interspersed with Dropout layers, enable the model to learn complex patterns from the data. The final dense layer, with 5 units and a softmax activation, allows the model to classify the input into one of five categories.
Both of these models have been trained using categorical cross-entropy loss which is typical for multi-class classification scenarios, Adam optimizer, and a learning rate reduction strategy that diminishes the learning rate when the validation loss plateaus, ensuring efficient and adaptive learning. Moreover, an early stopping mechanism monitors the validation loss, halting training if it doesn't improve significantly, and saving computational resources and time. To address potential class imbalances in the training data, the model is trained with computed class weights. It's worth mentioning the fact that architectures with more parameters with respect to presence detection were needed. This is because the network had the necessity of learning more complex features and understanding the differences between every class.
The following table shows the results of the activity-type detection task using these two explained architectures:

	Params	Test accuracy	Test loss
Aria VGG-style	140.901	0.93	0.12
LSTM-2	81.447	0.90	0.38

Table 6: Activity type detection model comparison on Aria Sensing radar dataset

The following image shows the confusion matrix on the activity-type detection task of the Aria VGG-style CNN with 140.901 parameters. It reached an impressive 0.93 accuracy on the test set, but since the number of samples was limited, the focus of the research moved to the collection to the NXP Semiconductors dataset.

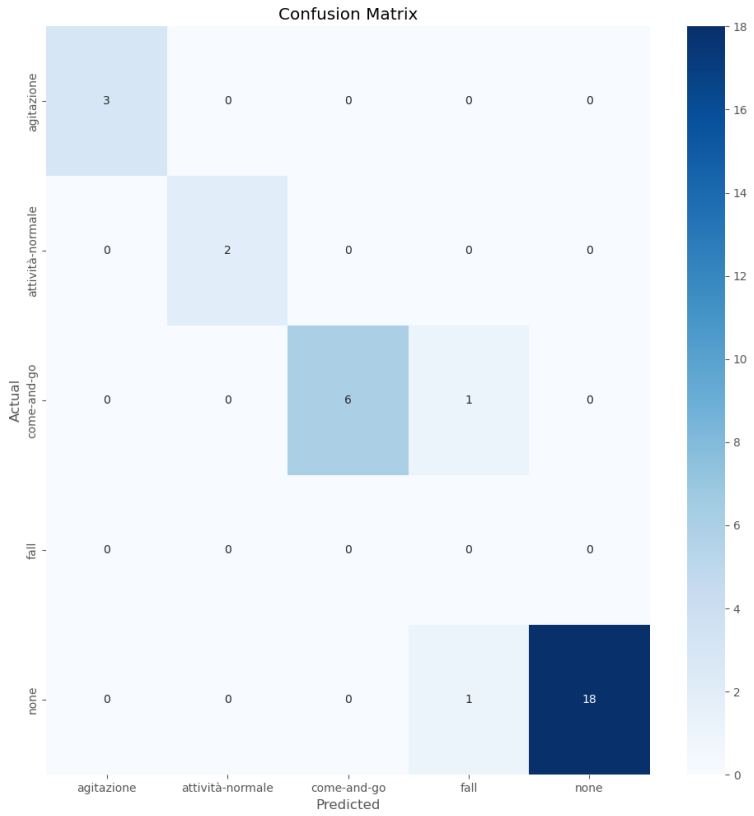


Figure 16: Activity type detection confusion matrix of Aria VGG-style

9. Conclusions

Operating within controlled environments, the developed algorithm shows the potential of TinyML combined with UWB radars as powerful tools for fall detection and beyond. The conducted experiments cover the entirety of the proposed solution, from the dataset collection to the deployment on device. Although a working solution has been provided, the limitations on the development were multiple, such as datasets collected only in a few environments, with few subjects doing limited activities, and low capabilities such as memory constraints on the device on which the solution has been deployed. Still, the goal of this project is achieved, demonstrating the effectiveness of this technology, and making it a starting point for future studies. This project shows a new path to make fall detection, setting a new starting point in the field of study of UWB radar combined with TinyML for fall detection and, furthermore, presence detection and human activity recognition.

9.1. Future directions

The solution proposed can be extended in various ways, an example would be to augment the dataset, including more human activities, conducted by different subjects in many different environments, so that the neural network could learn a wider range of human activities, understanding more precisely the difference from a fall (or different types of falls) from any other activity.
Another interesting path to follow is to find the right trade-off between model architecture (its structure and dimension) and memory needed on the device for on-device deployment, to find the best possible solution to the problem.
In this research, the Arduino microcontroller was just a placeholder for the radar, used to show that it's possible to deploy a pre-trained neural network model on-device and make inference directly on it, but the final application would need to deploy the model directly on the UWB radar, which would parse the recordings as soon as they are registered, and make inference in real-time.
Another direction that can be followed would be to add an alert system, an example could be to make the algorithm collaborate with an alarm system or domotics system which triggers an alert in case the algorithm detects a fall of a patient.
Furthermore, the technology used for this project can expand to multiple domains and has all it takes to become an ever-changing technology. The applications of UWB radars in collaboration with TinyML are broad. An example of it is to monitor vital signs such as heartbeats or a person's breath [1]. Moreover, it can be used in work environments to make safety monitoring, perhaps understanding when a person gets too close to a dangerous machine and behaving accordingly. It could be employed in cars, for example sending an alarm when passengers, even kids or pets, are left alone in a car for too much time, or even in the fitness and sports field, analyzing athlete's movements to improve techniques or ensure safety during training, and so on. This research is just a starting point for future studies.

References

[1] Fadel Adib, Hongzi Mao, Zachary Kabelac, Dina Katabi, and Robert C. Miller. Smart homes that monitor breathing and heart rate. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, page 837–846, New York, NY, USA, 2015. Association for Computing Machinery.

[2] Apple. Ultra wideband availability. `https://support.apple.com/en-us/HT212274/`.

[3] Bo Chen Menglong Zhu Matthew Tang Andrew Howard Hartwig Adam Dmitry Kalenichenko Benoit Jacob, Skirmantas Kligys. Quantization and training of neural networks for efficient integer-arithmetic-only inference.

[4] Kevin Bouchard, Julien Maitre, Camille Bertuglia, and Sébastien Gaboury. Activity recognition in smart homes using uwb radars. *Procedia Computer Science*, 170:10–17, 2020. The 11th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 3rd International Conference on Emerging Data and Industry 4.0 (EDI40) / Affiliated Workshops.

[5] Lathi BP. *Linear Systems and Signals.* Oxford University Press, 2010.

[6] S. Butterworth. On the Theory of Filter Amplifiers. *Experimental Wireless & the Wireless Engineer*, 7:536–541, October 1930.

[7] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep Learning with Low Precision by Half-wave Gaussian Quantization. *arXiv e-prints*, page arXiv:1702.00953, February 2017.

[8] Jingdong Chen, J. Benesty, Yiteng Huang, and S. Doclo. New insights into the noise reduction wiener filter. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(4):1218–1234, 2006.

[9] Moshfiq-Us-Saleheen Chowdhury, Ahnaf Tahmid, Mahfuz Ahmed Azmain, Mooaj Sadaqat Chowdhury, and Md Hossam-E-Haider. Exponential smoothing technique in filtration of distorted radar signal. In *2022 International Conference for Advancement in Technology (ICONAT)*, pages 1–5, 2022.

[10] Carmine Clemente. 'the micro-doppler effect in radar' by v.c. chen. *Aeronautical Journal*, 116(1176):5, February 2012.

[11] Dieter Coppens, Adnan Shahid, Sam Lemey, Ben Van Herbruggen, Chris Marshall, and Eli De Poorter. An overview of uwb standards and organizations (ieee 802.15.4, fira, apple): Interoperability aspects and future research directions. *IEEE Access*, 10:70219–70241, 2022.

[12] Mengyao Dong, Yihong Qi, Xianbin Wang, and Yiming Liu. A non-line-of-sight mitigation method for indoor ultra-wideband localization with multiple walls. *IEEE Transactions on Industrial Informatics*, 19(7):8183–8195, 2023.

[13] Moxa Doshi and Akson Varghese. Chapter 12 - smart agriculture using renewable energy and ai-powered iot. In Ajith Abraham, Sujata Dash, Joel J.P.C. Rodrigues, Biswaranjan Acharya, and Subhendu Kumar Pani, editors, *AI, Edge and IoT-based Smart Agriculture*, Intelligent Data-Centric Systems, pages 205–225. Academic Press, 2022.

[14] Yaxiang Fan, Martin D. Levine, Gongjian Wen, and Shaohua Qiu. A deep neural network for real-time detection of falling humans in naturally occurring scenes. *Neurocomputing*, 260:43–58, 2017.

[15] Estevao Gedraite and M. Hadad. Investigation on the effect of a gaussian blur in image filtering and segmentation. pages 393–396, 01 2011.

[16] Kishnaprasad G. Gunale and Prachi Mukherji. Fall detection using k-nearest neighbor classification for patient monitoring. In *2015 International Conference on Information Processing (ICIP)*, pages 520–524, 2015.

[17] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv e-prints*, page arXiv:1510.00149, October 2015.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[19] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2016.

[20] Robin John Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, Australia, 2nd edition, 2018.

[21] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

[22] Seung-Bae Jeon, Young-Hoon Nho, Sang-Jae Park, Weon-Guk Kim, Il-Woong Tcho, Daewon Kim, Dong-Soo Kwon, and Yang-Kyu Choi. Self-powered fall detection system using pressure sensing triboelectric nanogenerators. *Nano Energy*, 41:139–147, 2017.

[23] Pravin Kulurkar, Chandra kumar Dixit, V.C. Bharathi, A. Monikavishnuvarthini, Amol Dhakne, and P. Preethi. Ai based elderly fall prediction system using wearable sensors: A smart home-care technology with iot. *Measurement: Sensors*, 25:100614, 2023.

[24] Julien Maitre, Kévin Bouchard, and Sébastien Gaboury. Fall detection with uwb radars and cnn-lstm architecture. *IEEE Journal of Biomedical and Health Informatics*, 25(4):1273–1283, 2021.

[25] H. Nait-Charif and S.J. McKenna. Activity summarisation and fall detection in a supportive home environment. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 4, pages 323–326 Vol.4, 2004.

[26] Farzan M. Noori, Md. Zia Uddin, and Jim Torresen. Ultra-wideband radar-based activity recognition using deep learning. *IEEE Access*, 9:138132–138143, 2021.

[27] H. Nyquist. Certain topics in telegraph transmission theory. *Transactions of the American Institute of Electrical Engineers*, 47(2):617–644, 1928.

[28] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. Prentice Hall Press, USA, 3rd edition, 2009.

[29] Koray Ozcan and Senem Velipasalar. Wearable camera- and accelerometer-based fall detection on portable devices. *IEEE Embedded Systems Letters*, 8(1):6–9, 2016.

[30] S.U. Park, J.H. Park, M.A. Al-masni, M.A. Al-antari, Md.Z. Uddin, and T.-S. Kim. A depth camera-based human activity recognition via deep learning recurrent neural network for health and social care services. *Procedia Computer Science*, 100:78–84, 2016. International Conference on ENTERprise Information Systems/International Conference on Project MANagement/International Conference on Health and Social Care Information Systems and Technologies, CENTERIS/ProjMAN / HCist 2016.

[31] Massimo Pavan, Armando Caltabiano, and Manuel Roveri. Tinyml for uwb-radar based presence detection. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2022.

[32] Charissa Ann Ronao and Sung-Bae Cho. Human activity recognition with smartphone sensors using deep learning neural networks. *Expert Systems with Applications*, 59:235–244, 2016.

[33] Hoctor RT and Tomlinson HW. Delay-hopped transmitted-reference rf communications. pages 265 – 269, 02 2002.

[34] Laurence Rubenstein. Falls in older people: Epidemiology, risk factors and strategies for prevention. *Age and ageing*, 35 Suppl 2:ii37–ii41, 10 2006.

[35] NXP Semiconductors. Nxp semiconductors, 2023.

[36] Aria Sensing. Aria sensing. `https://www.liftt.com/en/portfolio-item/aria-sensing-en/`.

[37] Aria Sensing. Lt103oem uwb radar module. `https://usermanual.wiki/m/fe4f8353e3139c9e5f3d71c52fbdaf3d3f5fdda84bffb91dfde7c48bc69d5c74`.

[38] Anwar Shah, Javed Iqbal Bangash, Abdul Waheed Khan, Imran Ahmed, Abdullah Khan, Asfandyar Khan, and Arshad Khan. Comparative analysis of median filter and its variants for removal of impulse noise from gray scale images. *Journal of King Saud University - Computer and Information Sciences*, 34(3):505–519, 2022.

[39] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1997. Available at www.dspguide.com.

[40] Steven W. Smith. Chapter 15 - moving average filters. In Steven W. Smith, editor, *Digital Signal Processing*, pages 277–284. Newnes, Boston, 2003.

[41] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, D. Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2014.

[42] Li Tan and Jean Jiang. Chapter 6 - digital signal processing systems, basic filtering types, and digital filter realizations. In Li Tan and Jean Jiang, editors, *Digital Signal Processing (Second Edition)*, pages 161–215. Academic Press, Boston, second edition edition, 2013.

[43] The Arduino Team. Nano 33 ble sense.

[44] M.Z. Win and R.A. Scholtz. Ultra-wide bandwidth time-hopping spread-spectrum impulse radio for wireless multiple-access communications. *Communications, IEEE Transactions on*, 48:679 – 689, 05 2000.

[45] Dongxian Wu, Yisen Wang, Shutao Xia, James Bailey, and Xingjun Ma. Skip connections matter: On the transferability of adversarial examples generated with resnets. *ArXiv*, abs/2002.05990, 2020.

[46] Xinguo Yu. Approaches and principles of fall detection for elderly and patient. pages 42 – 47, 08 2008.

[47] Shugang Zhang, Zhiqiang Wei, Jie Nie, Lei Huang, Shuang Wang, and Zhen Li. A review on human activity recognition using vision-based method. *Journal of Healthcare Engineering*, 2017:1–31, 07 2017.

[48] Xu Zhou, Li-Chang Qian, Peng-Jie You, Ze-Gang Ding, and Yu-Qi Han. Fall detection using convolutional neural network with multi-sensor fusion. In *2018 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, pages 1–5, 2018.

Abstract in lingua italiana

Fall detection è il processo di identificazione delle cadute; si tratta di un problema di crescente importanza in vari settori, in particolare in quello sanitario e dell'assistenza agli anziani, dove le cadute si verificano di frequente. Le cadute possono portare a infortuni importanti, che spesso non vengono notati per lunghi periodi di tempo; nasce quindi l'esigenza di un metodo automatizzato per il rilevamento delle cadute. Questo progetto presenta un algoritmo che risolve il problema del rilevamento delle cadute utilizzando un radar a banda ultralarga (UWB) e un modello avanzato di rete neurale, implementandolo su un dispositivo Internet of things (IoT). Il modello, infatti, esegue inferenza direttamente su dispositivo, un microcontrollore Arduino con risorse limitate. L'addestramento del modello è basato su numerose registrazioni UWB-radar di cadute e altre attività umane, dalle quali impara a riconoscere i pattern tipici delle cadute umane. I radar a banda ultralarga offrono numerosi vantaggi rispetto alle tradizionali tecnologie di rilevamento delle cadute, quali accelerometri, giroscopi e telecamere. Si tratta infatti di un sistema non intrusivo che preserva la privacy dell'utente (i radar UWB non catturano o registrano immagini visive quindi volti e sembianze fisiche delle persone).
Questa tesi sottolinea anche il ruolo del Tiny Machine Learning (TinyML), che ha permesso di ottimizzare l'algoritmo per i dispositivi con risorse computazionali limitate, portando a una soluzione compatta ed efficiente dal punto di vista del consumo energetico e di memoria richiesta.
In sintesi, questa tesi integra con successo la tecnologia radar UWB, le reti neurali e il TinyML, fornendo una soluzione innovativa e rispettosa della privacy per il rilevamento delle cadute. La combinazione degli UWB e TinyML ha il potenziale per migliorare la qualità della vita, degli anziani e non solo, fungendo da punto di riferimento per i futuri progressi in questo critico settore della salute e della sicurezza.

Parole chiave: TinyML, Analisi delle cadute, Analisi della presenza, Analisi delle attività umane, Radar Ultrawideband

Acknowledgements

Questa tesi segna la fine di un lungo percorso, che con i suoi alti e bassi mi ha inevitabilmente trasformato in ciò che sono oggi. Sono qui grazie a varie figure che sono sempre state delle certezze nella mia vita. Non mi hanno mai fatto dubitare di me stesso e delle scelte fatte. In particolare la mia famiglia: Meri, Gigi, Dami, Olli e la nonna Ester. Senza di voi non sarei qui.
Grazie anche a Gino, mio amico e collega dal primo anno di università, l'unico sulla mia lunghezza d'onda e con cui paragonarsi e lamentarsi di molteplici vicende.
Ringrazio Massimo Pavan, il mio co-relatore, che è stato sempre disponibile durante lo sviluppo della tesi e devo riconoscergli una capacità invidiabile quando ci si trova davanti ad un problema: la calma. Ringrazio il mio relatore, il Prof. Roveri, anche lui è sempre stato disponibile durante tutta la tesi, lasciandomi grande libertà sulle decisioni del lavoro. Ringrazio Truesense, l'azienda con cui ho lavorato e che ci ha fornito i radar: Armando Caltabiano, Pierpaolo Lento, e Alessandro Bassi. Grazie anche a Gabriele Viscardi.
Si chiude un libro importante e questo lascia spazio per l'inizio di uno nuovo. Non temete, il futuro sarà un bel posto.

Ame