UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

DESIGN OF A TINY MACHINE LEARNING SYSTEM FOR UWB RADAR BASED MULTI TARGET DETECTION

LUIS GONZÁLEZ NAVARRO

2022

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

Master's Thesis

Title: Design of a Tiny Machine Learning system for UWB radar based multi target detection

Author: Luis González Navarro

Tutor: Luis Hernández Gómez (ETSIT)

Department: Señales, Sistemas y Radiocomunicaciones (ETSIT)

Group: Grupo de Aplicaciones del Procesado de Señal (ETSIT)

Tutor: Manuel Roveri (Politecnico di Milano)

Department: Electronics and Information (Politecnico di Milano)

Members of the Tribunal

President:

Vocal:

Secretary:

Substitute:

Grading:

Madrid, on October, 2022

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros de Telecomunicación

Master's Thesis

Design of a Tiny Machine Learning system for UWB radar based multi target detection

Author:

Luis González Navarro

Tutor:

Luis Hernández Gómez // Manuel Roveri

October 2022

Abstract

Tiny Machine Learning (TinyML) is a novel field of research which consists on designing machine and deep learning models at a reduced size, enabling them to be executed on tiny devices such as Internet-of-Things units, edge devices or embedded systems. The increasing use an research on TinyML follows the scientific trend of moving data processing closer to where data is generated. This offers an increase in the autonomy of tiny devices and achieving the most efficient use of energy possible.

One of the most promising applications for TinyML is presence detection. However, current automated presence detection systems rely on the processing of images taken from video cameras or microphones. While this is a very appropriate task for Machine Learning-based applications, the acquisition and processing of this data can lead to a risk in privacy vulneration, since images, videos or audio of people can be considered sensitive information. The use of Ultra-Wideband (UWB) radar might be a solution for this privacy issue, and aims to be a promising technology for tiny devices, since it is characterized by high precise recordings, low energy consumption and fast acquisition of data.

Combining these two technologies, we propose the development of automated presence detection solutions with the use of Ultra-Wideband (UWB) radar and TinyML. The goal of this research is to develop functional deep learning solutions for person counting and object detection with the use of UWB radar and applying TinyML algorithms to be able to deploy them on tiny devices. Different approaches have been considered, and several optimizations are offered in order to further shrink down the networks or improve their results.

Keywords

Tiny Machine Learning, Deep Learning, Ultra Wide Band radar, presence detection, person counting, object detection

Resumen

Tiny Machine Learning (TinyML) es un novedoso campo de investigación que consiste en el diseño de modelos de aprendizaje automático y profundo de tamaño reducido, lo que permite su ejecución en dispositivos pequeños como unidades de Internet de las Cosas, dispositivos periféricos o sistemas embebidos. El creciente uso e investigación en TinyML sigue la tendencia científica de acercar el procesado de datos al lugar donde los datos se generan. Esto permite aumentar la autonomía de los dispositivos pequeños y lograr un uso lo más eficiente posible de la energía.

Una de las aplicaciones más prometedoras de TinyML es la detección de presencia. Sin embargo, los actuales sistemas automatizados de detección de presencia se basan en el procesamiento de imágenes tomadas por videocámaras o micrófonos. Aunque se trata de una tarea muy apropiada para las aplicaciones basadas en el aprendizaje automático, la adquisición y el procesamiento de estos datos puede suponer un riesgo de vulneración de la privacidad, ya que las imágenes, vídeos o audio de personas puede considerarse información sensible. El uso del radar de banda ultraancha (UWB) podría ser una solución para este problema de privacidad, y apunta a ser una tecnología prometedora para dispositivos pequeños, ya que se caracteriza por la alta precisión de los registros, el bajo consumo de energía y la rápida adquisición de datos.

Combinando estas dos tecnologías, proponemos el desarrollo de soluciones automatizadas de detección de presencia con el uso del radar de banda ultraancha (UWB) y TinyML. El objetivo de esta investigación es desarrollar soluciones funcionales de aprendizaje profundo para el recuento de personas y la detección de objetos con el uso del radar UWB y aplicando algoritmos TinyML para poder desplegarlos en dispositivos pequeños. Para ello se han considerado diferentes enfoques y se ofrecen varias optimizaciones que permiten reducir aún más las redes o mejorar sus resultados.

Palabras clave

Tiny Machine Learning, aprendizaje profundo, radar de banda ultraancha, detección de presencia, conteo de personas, detección de objetos

Contents

List of Figures

List of Tables

Glossary

UWB	*Ultra-Wideband*
TinyML	*Tiny Machine Learning*
DFT	*Discrete Fourier Transform*
FFT	*Fast Fourier Transform*
NN	*Neural Netowrk*
HAR	*Human Activity Recognition*
CNN	*Convolutional Neural Networks*
RTC	*Real Time Clock*
TFLite	*TensorFlow Lite*
Dw convolution	*Depthwise convolution*

1 Introduction

This section includes a brief introduction to the research contained in this document. Starting with a description of the problem statement, followed by a general introduction to the main fields of study of the thesis, then a quick summary of the contributions of the work and ending with an explanation of the structure of the document.

1.1 Problem statement

In the most recent technological landscape, tiny devices are becoming one of the main areas of technological breakthrough. As Internet-of-Things (IoT) units, embedded systems and edge devices become more present in the technological environment, the scientific trend reflects the displacement of data processing closer to where the data is generated. This aims to increase the autonomy of tiny devices and to achieve the most efficient use of energy possible, while also reducing the required bandwidth and the latency of decision-making. Designing Machine Learning models for this devices results in a complete change of paradigm due to their severe constraints on memory, computation and power consumption, and this is precisely where Tiny Machine Learning (TinyML) plays a big role. TinyML offers a variety of tools to shrink down a neural network as much as possible, using methods such as quantization or pruning, in order to accomplish the aforementioned constraints found in tiny devices.

One of the most promising applications for TinyML is presence detection, meaning the identification of a person in a certain environment. However, current automated presence detection systems rely on the processing of images taken from video cameras or microphones. While this is a very appropriate task for Machine Learning-based applications, the acquisition and processing of this data can lead to a risk in privacy vulneration, since images, videos or audio of people can be considered sensitive information.

In order to address this privacy issues related to the current presence detection landscape, we propose the use of Ultra-Wideband (UWB) radar, which shows to be a promising radar technology for tiny devices. These devices are characterized by precise recordings (in the order of the mm), low energy consumption (generally under 0.1 W) and fast acquisition of data (in fractions of seconds). We believe firmly that the development of TinyML solutions for UWB-radar will pave the way for a great number of interesting applications based on tiny devices and guaranteeing the right to privacy of the users.

The aim of this research is to escalate the work of [2, 3] on Tiny Convolutional Neural Networks (TyCNNs) in order to expand their capabilities beyond presence detection. This novel family of CNNs is characterized by its custom design of tiny dilated convolutional blocks and the use quantization of the CNN architecture to reduce the computational and memory demands. There are two main goals to the thesis: firstly, to develop a TinyML solution for an automated person counting system inside a car with the use of UWB radar; secondly, to develop an automated object detection solution, developed in an indoor setting, which paves the way to developing person

tracking and distance measuring solutions for tiny devices with the help of UWB radar technology.

1.2 General framework

Tiny Machine Learning is a growing field of research that combines Machine Learning and Embedded Systems. It aims at the design of deep learning algorithms specially designed for tiny devices, such as microcontrollers. These are incredibly small devices, with very low power consumption and computing capacity, that can be powered by a single coin battery for weeks, months or even years [4]. TinyML is a novel research area, and it gives us the possibility to explore it further while tackling the already mentioned resource consumption concerns.

Ultra-Wideband Radar technology does not have a strict definition, but can be understood as radar systems that use wide relative bandwidth signals called Ultra-Wideband waveforms [5]. This technology has been used in human detection systems with good results in closed spaces, and is a very appropriate tool to tackle the privacy concerns derived from the use of camera-based person detection systems.

1.3 Brief description of the work

The research has been conducted in three main steps:

- The first step consists in the search of the best possible architecture for our neural network to be able to extract information from the UWB radar signals while not compromising its size and number of operations. The radar signals, in the form of range-Doppler maps, come from a dataset of around 500 UWB samples collected with a microcontroller unit that includes a UWB radar module, and have been captured inside a car, in order to detect the passengers present in the backseat. Both the dataset and the microcontroller have been provided to me by the company Truesense srl. The images have been pre-processed using Fast Fourier Transform and some feature extraction algorithms in order to improve the perception of the neural network. The goal is to implement a person counting solution, and different approaches have been used, such as classification and regression neural networks. To prove the strength of the selected architectures, the networks have been tested on an image classification task, where a regular image dataset of 64x64 images is used. It is the Tiny ImageNet dataset [6].

- The second step focuses on the further development of the previous architecture, exploring different optimizations to increase the capabilities of the network without compromising the size and number of operations. For this step, the same dataset has been used, containing around 500 samples captured from the backseat of the car.

- The third step approaches the problem as an object detection one. For this purpose, an object detection network has been designed from scratch. It takes inspiration from the design of the state of the art network for object detection on visual data (YOLO [7]). Its purpose is to

be able to perform object detection on our UWB radar signals, instead of regular colored images. However, the previous dataset can not be used for this task, mainly due to the difficulty of applying bounding box labels to it. To tackle this issue, a new dataset has been manually collected in a closed room environment with the same microcontroller unit as the first dataset.

This research lays its main foundations on the work carried out by Massimo Pavan on his Master Thesis [2] and the follow-up publication [3], and it aims to continue expanding the groundbreaking work developed by him and professor Manuel Roveri at the Department of Electronics and Information of Politecnico di Milano in colaboration with Truesense srl.

1.4 Document structure

The thesis is structured as follows.

Chapter 2 discusses the current state of the art, and some background on UWB radar technology and image processing in TinyML is included, as well as a summary of the methods and algorithms that have been used in the project.

Chapter 3 includes the previously mentioned architecture study, which details the research work done in order to achieve the best possible architecture to implement a person counting solution, including the different approaches explored.

The **chapter 4** follows the optimizations of this architectures to achieve a better performance or to reduce the computation of the networks. It includes the different approaches that have been followed to achieve this task, and the reasoning behind them.

Chapter 5 discusses the implementation of an object detection algorithm to the radar signals, with the intention of expanding it to other useful features such as person tracking and distance measuring.

Chapter 6 summarizes the results of the previous 3 chapters, while **chapter 7** sums up the conclusions of the results and hints at the future developments that could follow the current work of the thesis.

Appendix A includes the main ethical, economic, social and environmental aspects of the thesis, and **appendix B** details the financial budget of the project.

Finally, **Appendices C, D** and **E** contain grid search results and other tables from the results of chapters 3, 4 and 5.

2 State of the Art

This chapter includes a thorough analysis of the current state of the art on the fields of UWB radar and image processing in deep learning and TinyML. A collection of the methods, algorithms and metrics that have been used in the project is also included.

2.1 UWB Radar

Most of the literature that focuses on the use of UWB radar for similar purposes is grouped in two main categories: human detection and human activity recognition.

Human detection is the closest to the goals of this project. Most of the examples [8] are single target person detectors, and they use CNN to process a micro-Doppler representation of the radar data. In the case of multi-target person detectors, the tendency shows a preference for multi-antenna solutions [9, 10], while there are not many examples of a solution with a single transmission-reception pair. The aforementioned [2, 3] lay a strong groundwork in this scenario.

Various solutions with deep learning can be found, however, in the field of human activity recognition (HAR) [11]. Mainly present is the use of Convolutional Neural Networks (CNN), while others like Recurrent Neural Networks (RNN) and non-ML solutions are less frequent in the literature. In the case of CNN solutions, the input to the network can vary through the different examples. In [12] and [13] a radar micro-Doppler spectrogram is used, while, in examples like [14], a time-range map is used. Some solutions with radar that use range-Doppler maps can be found, like [15], but not one that uses them with UWB radar technology.

Of significant interest is the research on [16], where detection and localization of human targets is carried away with the help of CNN and range-time maps. An IR-UWB radar module is used for this task.

The absence of object detection solutions with UWB radar should be noted. No other implementations or portings of the previously mentioned solutions to an embedded device have been found, except for the one in [2, 3]. Very few examples of the use of UWB radar on embedded devices have been found either, other than [2, 3].

2.2 Tiny Machine Learning

Tiny Machine Learning or TinyML is a field of study that combines Embedded Systems and Machine Learning. It explores models and architectures reduced to the bare minimum, enabling small and low-power devices to run them. In comparison, a regular CPU consumes around 70 watts and a standard GPU consumes between 200 and 500 watts, while a microcontroller consumes power in the order of milliwatts. This low power allows TinyML devices to run for weeks or even months on a single coin battery. In order to design these kind of applications, strong constraints in terms of memory, computation and power have to be taken into account, and in fact constitute one of the main challenges to overcome [4]. Techniques such as weight quantization, pruning and

separable convolutions have been developed in order to address this challenge. An up-to-date review on the TinyML state of the art can be found in [17].

The progress in this emerging field is very fast, with new applications and usage ideas coming up on a daily basis. For this research, we take a special interest in the field of image processing.

Image processing in deep learning often relies on the aforementioned Convolutional Neural Networks. In the case of TinyML, a reduced architecture with CNN can work, but most of the time quantization and/or separable convolutions need to be used. A very notable research can be found in [18], where another type of optimization is explored: log-gradient (log∇) preprocessing, and it shows promising results in combination with quantization and pruning. Another interesting research can be found in [19], where visual attention condensers are used to reduce the parameters and number of operations of the network while improving the accuracy in comparison with previous networks like MobileNet.

Most of the research found on this field comes from very recent years, and not many publications on the topic of image recognition using TinyML can be found. Very few object detection solutions have been found on the field of TinyML either, as most of these implementations result on heavy networks with a big computational cost. The very recent solution found in FOMO by Edge Impulse [20] should be highlighted.

2.3 Background

A summary of the main methods and algorithms that have been used in the project is included in this section.

2.3.1 FFT

Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa. The Discrete Fourier Transform (DFT) is obtained by decomposing a sequence of values into components of different frequencies. This operation is useful in many fields, but computing it directly from the definition is often too slow to be practical. This is where the Fast Fourier Transform comes in.

The Fast Fourier Transform (FFT) is an efficient algorithm to calculate the DFT of a sequence. It is a divide and conquer algorithm that recursively breaks the DFT into smaller DFTs to bring down the computation. The answer to how FFT speedup the computing of DFT lies in the exploitation of the symmetries in the DFT. Cooley and Tukey showed that we can calculate DFT more efficiently if we continue to divide the problem into smaller ones. As a result, it successfully reduces the complexity of the DFT from O(n^2) to O($n \log n$), where n is the size of the data [21]. The algorithm was described first in Cooley and Tukey's classic paper in 1965 [22].

Algorithm 1 Cooley-Tukey's FFT algorithm

Require: N integer power of 2

$X_{0,\dots,N-1} \leftarrow dit\text{fft2}(x, N, s)$:
if $N = 1$ **then**
    $X_0 \leftarrow x_0$
else
    $X_{0,\dots,(N/2)-1} \leftarrow dit\text{fft2}(x, N/2, 2s)$:
    $X_{N/2,\dots,N-1} \leftarrow dit\text{fft2}(x + s, N/2, 2s)$:
    for $k = 0$ to $(N/2) - 1$ **do**
        DFT:
        $p \leftarrow X_K$
        $q \leftarrow exp - 2\pi/NkX_k + N/2$
        $X_K \leftarrow p + q$
        $X_K + N/2 \leftarrow p - q$
    end for
end if

2.3.2 CNN

A Convolutional Neural Network (CNN) a deep learning algorithm that can take in an input image, assign importance weights and biases to various aspects or objects in the image and be able to differentiate one from the other [23]. They have at least a convolutional layer, which a mathematical operation called convolution designed specifically to process image-like array data. The objective of the convolution operation is to extract the high-level features such as edges, from the input image.

The input of a CNN is usually some type of image, while the output can be adapted to whatever type of information is wanted from the input images. They are multi-layer networks, where the hidden layers are mostly made up of convolutional layers, which extract the information from the image, pooling layers, to reduce the image size after the convolution is performed, and activation layers.

2.3.3 K-Fold Cross-Validation

Cross-validation is a statistical method used to evaluate machine learning models on a limited data sample. It is commonly used in applied machine learning to compare and select a model for a given predictive modeling problem because it is easy to implement and results in skill estimates that generally have a lower bias than other methods. The procedure is as follows:

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times [24].

Algorithm 2 K-Fold Cross-Validation

```
Shuffle the dataset randomly
Split the dataset into K groups
for each unique group do
    Take the group as a hold out or test data set
    Take the remaining groups as a training data set
    Fit a model on the training set and evaluate it on the test set
    Retain the evaluation score and discard the model
end for
Summarize the skill of the model using the sample of model evaluation scores
```

2.3.4 Accuracy

Accuracy is a metric to evaluate machine learning models mainly used in classification problems. It can be defined as the ratio of the number of correctly classified cases to the total of cases under evaluation. The best value of accuracy is 1 and the worst value is 0. Accuracy can be a very useful metric if used in the correct scenario. However, it should be taken into account that it gives biased results for data with unbalanced classes [25].

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Three different types of accuracies can be obtained from a neural network:

- Training accuracy (accuracy obtained from the training samples)
- Max validation accuracy (highest accuracy value obtained from the validation samples)
- Last validation accuracy (accuracy value obtained from the validation samples at the last epoch)

The last validation accuracy should be the main accuracy to take into account, since it is the closest we can get to a real scenario. However, training accuracy can be a useful metric to check the overfitting of the network.

In the case of using cross-validation, an average of the accuracies of each fold has been used.

$$Accuracy_{avg} = \frac{1}{K} \sum_{n=1}^{K} Accuracy_n$$

2.3.5 Confusion Matrix

A confusion matrix is a metric for summarizing the performance of a classification algorithm. Classification accuracy alone can be misleading in the case of dataset unbalance, which happens most of the time when training a neural network. The confusion matrix can give you a good idea of what your classification model is getting right and what types of errors it is making.

A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. This way, it shows the ways in which your classification model is confused when it makes predictions. It gives you insight not only into the errors being made by your classifier but more importantly the types of errors that are being made. It is this breakdown that overcomes the limitation of using classification accuracy alone.

The process of calculating the confusion matrix is the following:

Algorithm 3 Confusion Matrix

1: Take the test/validation dataset with expected outcome values
2: Make a prediction for each row in your test dataset
3: **for** each set of predictions **do**
4: Count number of correct predictions for each class
5: Count number of incorrect predictions for each class, organized by the class that was predicted
6: **end for**
7: Fill the counts of correct and incorrect classification into a table

The result is a table or matrix where each row of the matrix corresponds to a predicted class and each column of the matrix corresponds to an actual class [26]. In a multi-class classification problem, an example of a graphic representation of the confusion matrix can be found in Figure 1, along with the true-positive, false-positive, true-negative and false-negative distribution relative to the first class.

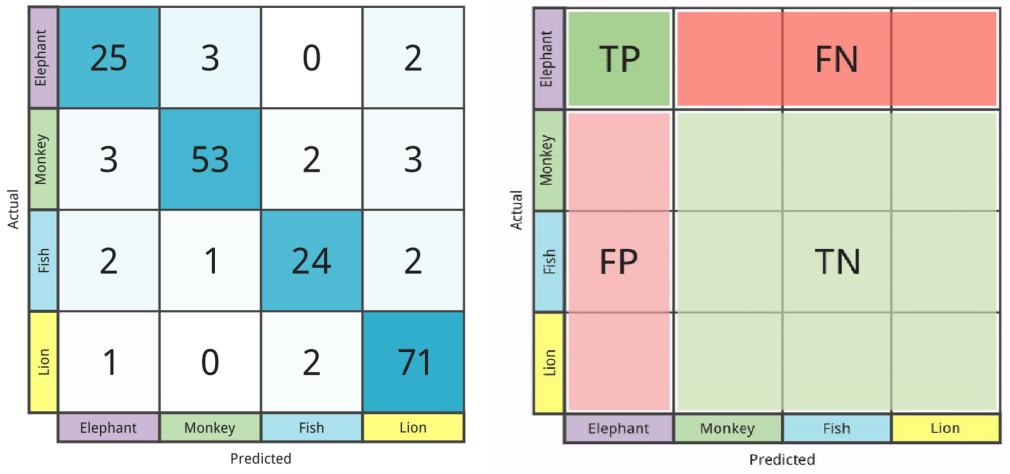

Figure 1: Confusion matrix example of an animal classification problem [1]

2.3.6 IOU

Intersection over Union is an evaluation metric used to measure the accuracy of the predicted bounding boxes of any algorithm. It's a very common metric in CNN detectors (R-CNN, Faster R-CNN, YOLO, etc.) [27]. In order to apply IOU to an object detector, we need:

- The ground truth bounding boxes, hand labeled, taken from the testing set and assumed to be the most correct.
- The predicted bounding boxes from our model.

With this two sets of bounding boxes we can calculate the IOU with this equation, which is simply a ratio between the area of overlap and the area of union:

$$IOU = \frac{\textit{Area of Overlap}}{\textit{Area of Union}}$$

This is a metric for a single sample, however. When the aim is validating a full set of predictions, average IOU is used:

$$IOU_{avg} = \frac{1}{K} \sum_{n=1}^{K} IOU_n$$

In the case of multi-class labeling for object detection algorithms, Mean Average Precision is used to combine the IOU values of each class. However, this is not necessary for this research, which is focused on one class object detection.

2.3.7 Grid search

A simple method that can be used to test out different parameters in the network, and one that helps us to find the best combination of them, is grid search. We assign a set of values ($L_1 ... L_k$) for each variable parameter in the network, and they are tested in every possible combination:

$$s = \prod_{k=1}^{|L|} L_k$$

While usually the main problem with grid search is the exponentially growing number of parameters, the limitations in terms of memory and computation only leaves us with a small enough set of parameters that make grid search viable as an optimization method.

3 Architecture study

The goal of this part of the thesis is to find the best neural network architecture possible be able to extract information from the UWB radar signal in order to perform a classification task. At the same time, we want to avoid compromising its memory footprint and computational demand. The radar signals, in the shape of range-Doppler maps, come from a dataset of around 500 samples captured in a car setting. The images have been pre-processed using Fast Fourier Transform and some feature extraction algorithms in order to improve the predictions and the memory occupation of the neural network. The problem will then be tackled with a regression approach that makes use of transfer learning to switch the classification output of the network for a regression one. The results will be compared to the classification approach. Finally, to prove the strength of the selected architectures, the networks have also been tested on an image classification task, where a regular image dataset of 64x64 images is used.

3.1 Dataset

The dataset is composed of 478 samples of range-time maps, recorded inside a car environment. They have been acquired by an ESP32 microcontroller unit with a UWB radar module, equipped with a couple of TX and RX antennas. The equipment is an ultra-high precision radar sensor developed by ARIA sensing. In all the recordings the device was placed above one of the back lateral windows of the car, scanning the backseats of the car in order to detect the passengers present, as can be seen in Figure 2.

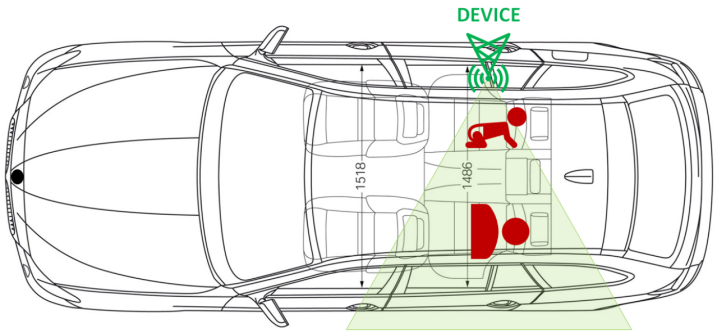

Figure 2: Sample acquisition example [2, 3]

The radar module is capable of performing a scan of the environment, returning a 65-values-long series of imaginary values representing energy intensities, where the first value represents the spatial bin closer to the device, and the 65th the most distant one. The radar can detect subjects in a 60º cone from where it's directed. The same scan is repeated over a 20 second period, at 10 Hz, resulting on a range-time map of 65 x 200 for each dataset sample.

The dataset contains information regarding each radar sample, such as the serial number and battery level of the device, but, most importantly, the seats that were occupied at the moment of recording. A breakdown of each seat is given, starting from seat 1 (the closest one to the

device) up until seat 3. The dataset includes a breakdown of each seat in 6 categories. The main distinction is made by age groups, starting from 'Baby', then 'Toddler', 'Teen' and finally 'Adult'. The other two classes correspond to 'Pet', for animals, and 'None' for empty seats.

Although these classes have been used for experimentation, our main focus is people counting. Thus, the number of people present in the car is the value used in the final research, while the age group distinctions have been not been used.

Here are some samples taken from the dataset:

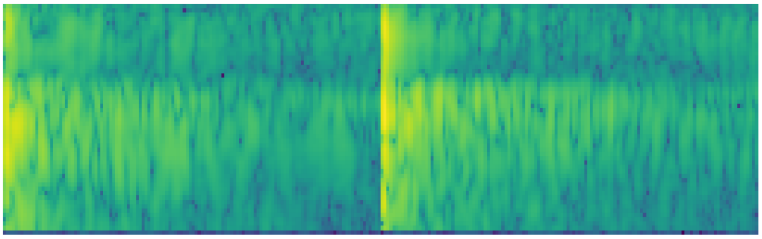

Figure 3: FFT Data Example. The X axis represents Frequency bins, while the Y axis represents space. The left and right parts of the image represent the real and imaginary part respectively of the same imaginary value. This sample represents an adult on seat 1, while the rest are empty

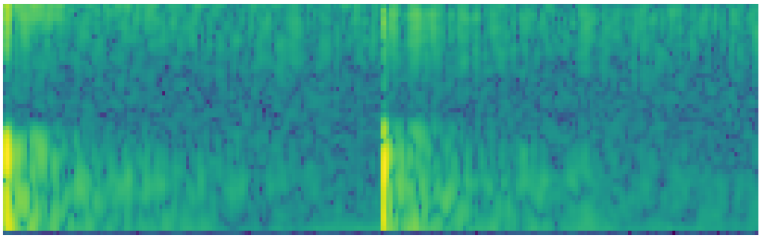

Figure 4: FFT Data Example. This sample represents an adult on seat 3, while the rest are empty

3.2 Problem formulation

The problem is divided into two classification sub-problems. The classes correspond to the number of people present in the backseats of the vehicle:

- A multi-label classification problem with 4 classes.
- A multi-label classification problem with 3 classes.

The first problem can be formalized as a classification task where each class corresponds strictly to the number of passengers in the backseat of the car. A multi-class classification task requires to learn the following function:

$$f(x) = \begin{cases} 3 & \text{if there are 3 passengers in any seat} \\ 2 & \text{if there are 2 passengers in any seat} \\ 1 & \text{if there is 1 passenger in any seat} \\ 0 & \text{if all the seats are empty} \end{cases}$$

The balancing of the dataset for this sub-problem is far from optimal, with a notable unbalance among the different classes, as it can be seen in Figure 5.

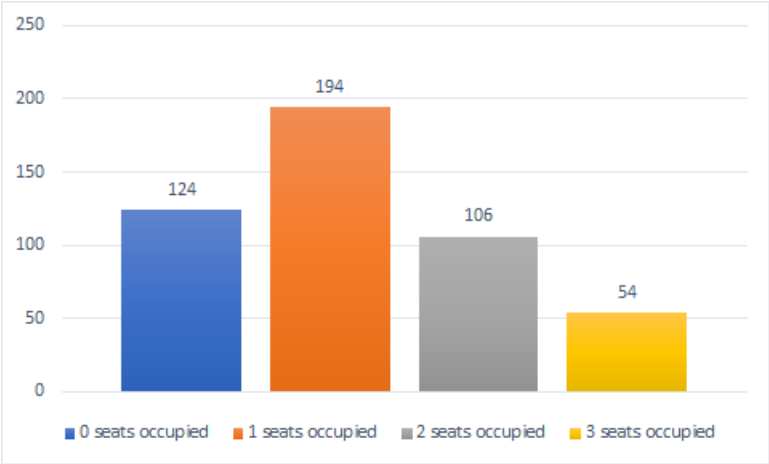

Figure 5: Data distribution for 4 classes

The second problem can be formalized as a classification task where each class corresponds to an indication of the number of passengers in the backseat of the car. This variation comes from the fact that the majority of the mistakes made by the network are between classes 2 and 3. It also helps with the class balancing of the dataset. A multi-class classification task requires to learn the following function:

$$g(x) = \begin{cases} 2 & \text{if there are 2 or more passengers in any seat} \\ 1 & \text{if there is 1 passenger in any seat} \\ 0 & \text{if all the seats are empty} \end{cases}$$

The balancing of the dataset for this sub-problem is certainly improved, while still showing some class unbalance, as it can be seen in Figure 6.

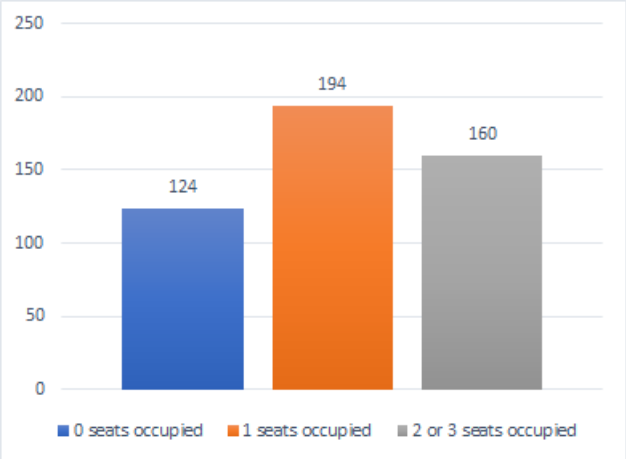

Figure 6: Data distribution for 3 classes

The goal of this problem is to approximate both functions as accurately as possible by training

on the records collected by the company directly on the device, while keeping the size and number of operations of the network as low as possible.

3.3 Data preprocessing

In order for a neural network to learn from the dataset, while also keeping its size as small as possible without hurting the performance of the network, a series of preprocessing steps have been taken in order to convert these samples into input data for the network:

- Firstly, Fast Fourier Transform algorithm is applied to the raw data. In this dataset, the FFT was directly provided by the microcontroller using the incorporated FFT function available in the library of the device. The result is a range-Doppler map, sized 65 x 256, where the y axis represents frequency instead of time. The noticeable shape change, from 200 to 256 width, comes from the limitations of the on-device FFT processing, where data needs to be arranged in a power of two shape. In order to reshape the input data of the range-time maps a Hann window is used.
- The FFT values are then scaled logarithmically, and the norm of the imaginary values is calculated. This results in the transformation of the FFT data into real values.
- Some of the first horizontal bins are cut as well, due to the high presence of noise present to the bins closer to the device. The result is a cut in size, leaving the final range-Doppler map with 53 x 256 values. See Figure 7.

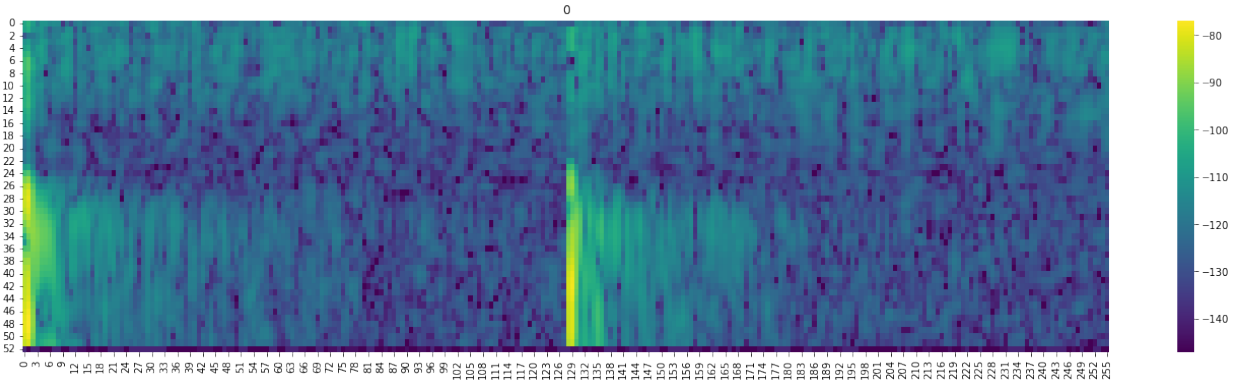

Figure 7: 53 x 256 range-Doppler map after preprocessing step 3

- Next step is the frequency selection. This step consists on keeping only a fraction of all the frequencies present in the range-Doppler map and discarding the rest. The objective is obtaining a reduction in terms of memory footprint, as well as avoiding the overfitting of the algorithm to the data at disposal. It can be done due to the fact that human-related activities, once ported in the frequency domain, tend to concentrate in the first bins, that can be roughly translated in frequencies that go from 0 Hz to 1,66 Hz. This allows us to cut up to two thirds of the data in the range-Doppler maps without worsening the performance of the network. The network input size is finally 53 x 86. See Figure 8.

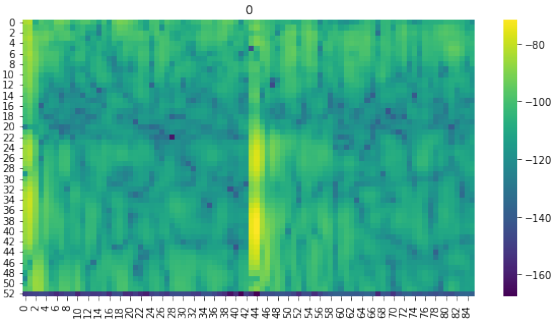

Figure 8: 53 x 86 range-Doppler map after preprocessing step 4

- Finally, a Z-score normalization is performed, in order to feed the data into the network. Z-score normalization is the process of normalizing every value in a dataset such that the mean of all of the values is 0 and the standard deviation is 1 [28]. This allows the network to process the information.

$$Z = \frac{x - \mu}{\sigma}$$

All the preprocessing steps are mainly based on the previous research carried out in [2, 3].

3.4 Network requirements

Keeping in mind that we are developing a TinyML application, the limitations of the hardware have to be taken into account. The application is expected to be run in the same device where the dataset samples are taken, the ESP32 Microcontroller. It includes the following components: a Flash memory of size 4MB, two RAM memories, sized 520KB SRAM and 16KB SRAM respectively (in RTC), and a 1800 mAh battery, which translates into an estimated duration of 6/7 months of normal use. However, not all the resources on the device can be dedicated to the algorithms. The actual limits that each algorithm needs to meet to be ported and used in the device are summarized in *Table 1*.

Resource	Available Values
Flash Memory	3 MB
RAM	150 KB

Table 1: Maximum available memory dedicated to the neural network in the microcontroller

These parameters have to be taken into account when designing the network architecture. For this purpose, a TFLite model is compiled for each model to study the flash memory occupation, and an estimation of the number of operations is carried out to consider the RAM use.

3.5 Baseline network design

As a blueprint for our proposed network architecture, the architecture study of [2, 3] is used. The main reasons for this decision are the use of the same input information and the TinyML

requirements. The philosophy behind this architecture is to replicate small parts of other popular image recognition architectures while keeping the network as small as possible.

The baseline architecture is divided in two main blocks:

- The convolutional part, which is a repetition of several convolutional blocks. These convolutional blocks are made up of two convolutional layers, to extract the main features of the image, and a max pooling layer, to reduce the image size.
- The classification part has a very standard build, composed of a flattening layer, a dropout layer and a dense layer. The dense layer then outputs the probabilities for each class. In the original architecture, the output is binary.

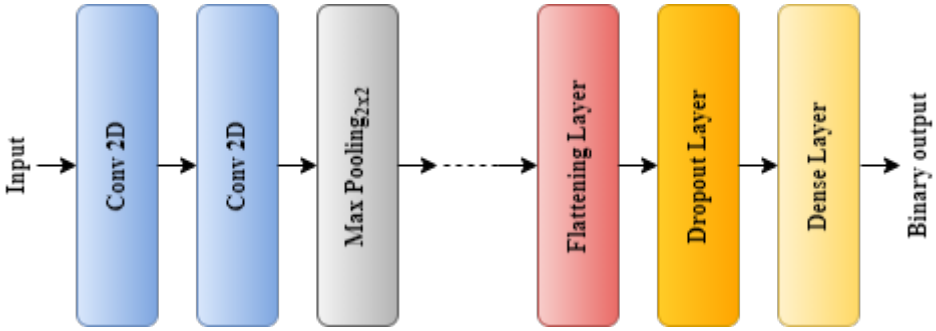

Figure 9: Baseline Network Architecture

This architecture has some variable parameters that can be modified to adapt the network. The change of these parameters will result in a modification in network size. The main variable parameters are are:

- **Convolutional kernel size:** It represents both height and width of the 2D convolution window. The model uses square convolution windows of odd dimensions.
- **Number of convolutional blocks:** number of convolutional blocks present in the network.
- **Number of filters:** number of output filters for each convolutional layer. It is the same for every convolutional block.
- **Dropout rate:** the dropout rate of the dropout layer, which represents the fraction of the input units to drop. It helps prevent overfitting.
- **Dilation rate:** integer specifying the dilation rate to use for dilated convolution.

3.6 Proposed network design

The main modification from the baseline architecture to our proposed architecture is the output. The binary class output is substituted by a multi-class output. This means a change in the final dense layer, from ’sigmoid’ activation to ’softmax’ activation.

The rest of the architecture, both the convolutional and classification parts, remains the same, as well as the variable parameters of the network. A max pooling layer is added at the start of the network, greatly reducing the weight and computational cost of the network. The resulting architecture is represented in Figure 10.

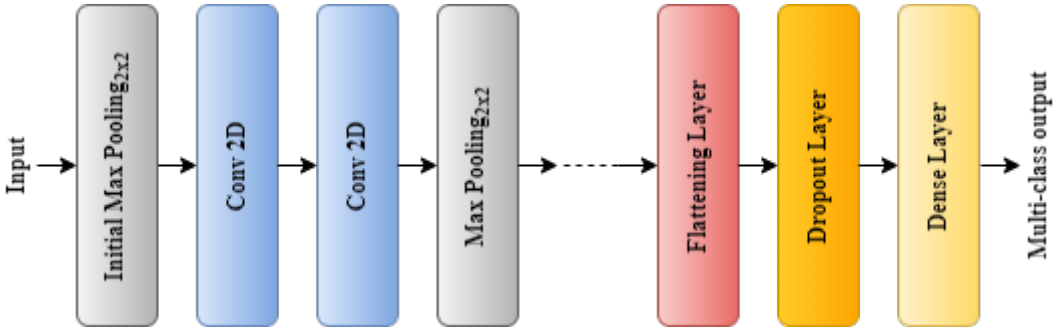

Figure 10: Proposed Network Architecture

Due to the size of the dataset, and to the convolutional networks being particularly data hungry, 10-fold cross-validation has been implemented. Class balancing is taken into account in the training phase, since the dataset is not evenly distributed. The '*class weight*' parameter of *model.fit()* helps the model to "pay more attention" to samples from an under-represented class.

The same variables from the baseline network are used. In order to check the different combination of these variables, a grid search is performed. Table 2 includes a breakdown of the variables tested for the grid search.

Parameter	Min. Value	Max. value
conv kernel dim	3	7
n convolutions	2	4
n filters	6	22
drop out rate	0.3	0.3
dilation rate	1	2

Table 2: Grid search parameter range

The values selected for the grid search are inspired by the ones used for the baseline architecture, and respond to the computational and memory limitations and the to the available training time and resources. The increasing of convolutional block number and dimension of the kernels come with an exponential increase on training time.

3.7 Regression approach

In order to test a new approach to the problem, a regression output is implemented. The idea behind this experiments lays its foundation behind the frequent use of regression in person and item counting networks [29]. In order to implement this changes, a transfer learning method is used. The process of modifying the network follows these steps:

- Firstly, the preprocessing of the data is computed normally, using FFT, frequency selection

and Z-score normalization as detailed previously in **Chapter 3.3**.

- Then, the neural network from the previous 4-class classification experiment with the best results is loaded.
- Transfer learning is applied, removing the last layer of the network where the multi-class classification is performed. This layer is then replaced by another dense layer with one output and linear activation.
- The model is finally compiled with the 'mean squared error' loss function and trained with the 'mean average error' metric. The model will be trained on the whole dataset, averaging the results among different splits of test and validation data.
- Different numbers of epochs will be tested, due to transfer learning being specially sensible to this value.
- In order to calculate the accuracy of the network and be able to compare it to our classification networks, the outputs are rounded to match the integer values corresponding to how many people are present in the vehicle. Accuracy is then calculated against the true values.

In order to compare to both 4-class classification and 3-class classification models, we will also compute the accuracy obtained after combining classes 2 and 3 (2 and 3 passengers respectively). In regression, the combining of these classes comes after training the network, and not before. This is because, while a classification network can easily classify two different kinds of images in the same category with ease, regression is trying to approximate a function, where it is more difficult for different inputs to have the same output.

3.8 Adaptation to image data

In order to prove the strength of this architecture, an image classification experiment has been carried out. This time, a regular image dataset is used: Tiny ImageNet [6]. This test can give us some hints towards whether or not the architecture can be used in a different scenario, and also whether the results would improve if a larger and more complete dataset were available to train our network.

The training dataset is composed of 100,000 images, divided in 200 classes. The images are sized 64x64 pixels. A test dataset is also included, with a total of 10,000 images (50 per class). However, in order to create a similar scenario to our own, only 3 classes will be used for training and classified by the network.

In order to adapt the baseline architecture, the main difference is the input size, which changes from 53 x 86 x 1 images to 64 x 64 x 3. The most meaningful change is the addition of 2 extra color spaces, which adds to the network's size. However, since this dataset is used only for comparison and there is no plans for deploying this network into the microcontroller, this is not a big concern.

Some of the optimizations among the ones explained in the following chapter have also been tested in this network, in order to check the efficiency of these optimizations in a bigger and more complete dataset.

The results of all of these experiments will be discussed in **Chapter 6**.

4 Person counting

The goal of this part of the thesis is to improve the proposed network architecture from the previous chapter. The goal remains the same, person counting in a car environment, and thus the same dataset is used. These network modifications, or "optimizations", aim to improve the architecture in two main ways: improving the results of the network, in the case of the RAM usage optimization, and reducing the computational and memory requirements of the network, in the case of the depthwise convolution optimization and the average pooling optimization. Finally, to prove the strength of these optimizations, they have also been tested on an image classification network, where a regular image dataset of 64x64 images is used.

4.1 Dataset

The same dataset from the previous section is used, containing 478 samples recorded in the backseat of a car. For more information on the dataset see **Chapter 3.1**.

4.2 Problem formulation

The problem is a multi-label classification problem with 3 classes. The output corresponds to the number of people present in the backseats of the vehicle.

The problem is equivalent to the second problem of the previous chapter, see **Chapter 3.2** for more information. The main goal of this chapter is to improve on the previous architecture, both in results and computation. The multi-class classification task requires to learn the following function:

$$f(x) = \begin{cases} 2 & \text{if there are 2 or more passengers in any seat} \\ 1 & \text{if there is 1 passenger in any seat} \\ 0 & \text{if all the seats are empty} \end{cases}$$

The balancing of the dataset for this problem is the same as the previous chapter, as it can be seen in Figure 11.

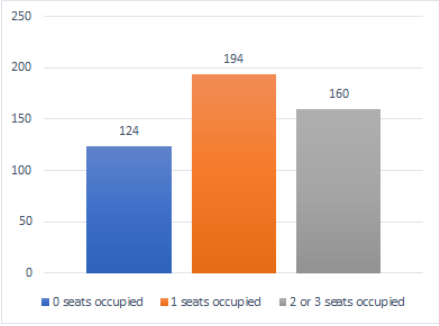

Figure 11: Data distribution for 3 classes

4.3 Data preprocessing

Data preprocessing remains the same as the previous section of the thesis. Most notably, the use of FFT, frequency selection and Z-score normalization is unchanged. For more information on the data preprocessing see **Chapter 3.3**.

4.4 Network requirements

The same network requirements are present, taking into account the limitations of the ESP32 Microcontroller where the networks are planned to be deployed. For more information on the network requirements check **Chapter 3.4**.

4.5 Computational load and memory footprint evaluation

For the first problem of this chapter, our goal is to reduce the computational and memory requirements of the network. In order to do this, an in-depth understanding of these parameters is required.

While checking the memory footprint of the network can be as straight forward as compiling the tflite model and inspecting its size, checking the computational load can be a daring task to check, and may involve the full deployment of the network on a functioning device. In order to understand the RAM optimization, a better criteria for calculating the computational load must be specified. We can find a definition for computational load c and memory footprint m in [3], and it is stated as follows:

$$c = n_{ops}$$
$$m = (\hat{m_w} + \hat{m_a}) \cdot m_p$$

being n_{ops} the number of multiplications required to compute the inference, $\hat{m_w}$ the cardinality of the parameters of the network, $\hat{m_a}$ the memory needed to store the maximum sum of two consecutive activations and m_p the memory required to store a value of a parameter or an activation. In our case, since both the weights of the network and the input data are quantized, m_p will be equal to 1 Byte. Only two arrays are used to store intermediate activations, and are shared among the processing layers of the network.

In order to obtain the values of memory footprint and number of operations for each different layer of the network, a specific calculation is required. The detailed calculation for each layer can be found in [3]. In regards to our network, a detailed breakdown with the number of operations and memory footprint for our proposed model layer by layer can be found in Table 3, where M and L correspond to the columns and rows of the image respectively, n to the number of filters, K to the index of each convolutional block and r the side length of the square 2D kernel.

	Memory footprint (Bytes)	Number of operations
Input	$M \cdot L \cdot 1$	-
Pool0 (Weights)	-	-
Input-Pool0 (Activations)	$\frac{M}{2} \cdot \frac{L}{2} \cdot 1$	$2 \cdot 2 \cdot M \cdot L$
Conv1_00 (Weights)	$r^2 \cdot n + n$	-
Conv1_00 (Activations)	$\frac{M}{2} \cdot \frac{L}{2} \cdot n$	$r^2 \cdot n \cdot \frac{M}{2} \cdot \frac{L}{2}$
Conv1_01 (Weights)	$r^2 \cdot n^2 + n$	-
Conv1_01 (Activations)	$\frac{M}{2} \cdot \frac{L}{2} \cdot n$	$r^2 \cdot n^2 \cdot \frac{M}{2} \cdot \frac{L}{2}$
Pool1 (Weights)	-	-
Pool1 (Activations)	$\frac{M}{4} \cdot \frac{L}{4} \cdot n$	$2 \cdot 2 \cdot \frac{M}{2} \cdot \frac{L}{2}$
ConvK_00 (Weights)	$r^2 \cdot n^2 + n$	-
ConvK_00 (Activations)	$\frac{M}{2^K} \cdot \frac{L}{2^K} \cdot n$	$r^2 \cdot n^2 \cdot \frac{M}{2^K} \cdot \frac{L}{2^K}$
ConvK_01 (Weights)	$r^2 \cdot n^2 + n$	-
ConvK_01 (Activations)	$\frac{M}{2^K} \cdot \frac{L}{2^K} \cdot n$	$r^2 \cdot n^2 \cdot \frac{M}{2^K} \cdot \frac{L}{2^K}$
PoolK (Weights)	-	-
PoolK (Activations)	$\frac{M}{2^{K+1}} \cdot \frac{L}{2^{K+1}} \cdot n$	$2 \cdot 2 \cdot \frac{M}{2^K} \cdot \frac{L}{2^K}$
FC Classifier (Weights)	$\frac{M}{2^{K+1}} \cdot \frac{L}{2^{K+1}} \cdot n \cdot 3 + 3$	-
FC Classifier (Activations)	3	$\frac{M}{2^{K+1}} \cdot \frac{L}{2^{K+1}} \cdot n \cdot 3$

Table 3: Memory footprint and computational demand of the proposed network

4.6 RAM usage optimization

Having established the calculations of memory footprint and computational demand, the aim of this first optimization is to take advantage of the microcontroller's available RAM as much as possible. As we can see in Table 3, the activation memory as well as the number of operations decreases exponentially with each new convolutional block:

$$m_a^C = \frac{M}{2^K} \cdot \frac{L}{2^K} \cdot n$$
$$n_{ops}^C = r^2 \cdot n^2 \cdot \frac{M}{2^K} \cdot \frac{L}{2^K}$$

In order to take advantage of this decrease, we propose to increase the number of filters exponentially; this allows us to fully take advantage of the device's RAM with a the increase in network size as tradeoff:

$$m_a^C = \frac{M}{2^K} \cdot \frac{L}{2^K} \cdot n \cdot 4^{K-1} = \frac{M}{2} \cdot \frac{L}{2} \cdot n$$
$$n_{ops}^C = r^2 \cdot n^2 \cdot \frac{M}{2^K} \cdot \frac{L}{2^K} \cdot 4^{K-1} = r^2 \cdot n^2 \cdot \frac{M}{2} \cdot \frac{L}{2}$$

It should be noted that, while the activation size $\hat{m_a}$ remains constant, the total amount of memory will increase due to the increase of $\hat{m_w}$. The resulting changes to the network in terms of memory and number of operations are collected in Table 4.

	Memory footprint (Bytes)	Number of operations
Input	$M \cdot L \cdot 1$	-
Pool0 (Weights)	-	-
Input-Pool0 (Activations)	$\frac{M}{2} \cdot \frac{L}{2} \cdot 1$	$2 \cdot 2 \cdot M \cdot L$
Conv1_00 (Weights)	$r^2 \cdot n + n$	-
Conv1_00 (Activations)	$\frac{M}{2} \cdot \frac{L}{2} \cdot n$	$r^2 \cdot n \cdot \frac{M}{2} \cdot \frac{L}{2}$
Conv1_01 (Weights)	$r^2 \cdot n^2 + n$	-
Conv1_01 (Activations)	$\frac{M}{2} \cdot \frac{L}{2} \cdot n$	$r^2 \cdot n^2 \cdot \frac{M}{2} \cdot \frac{L}{2}$
Pool1 (Weights)	-	-
Pool1 (Activations)	$\frac{M}{4} \cdot \frac{L}{4} \cdot n$	$2 \cdot 2 \cdot \frac{M}{2} \cdot \frac{L}{2}$
ConvK_00 (Weights)	$r^2 \cdot n^2 \cdot 4^{K-1} + n \cdot 4^{K-1}$	-
ConvK_00 (Activations)	$\frac{M}{2} \cdot \frac{L}{2} \cdot n$	$r^2 \cdot n^2 \cdot \frac{M}{2} \cdot \frac{L}{2}$
ConvK_01 (Weights)	$r^2 \cdot n^2 \cdot 4^{2(K-1)} + n \cdot 4^{K-1}$	-
ConvK_01 (Activations)	$\frac{M}{2} \cdot \frac{L}{2} \cdot n$	$r^2 \cdot n^2 \cdot \frac{M}{2} \cdot \frac{L}{2} \cdot 4^{K-1}$
PoolK (Weights)	-	-
PoolK (Activations)	$\frac{M}{2^K+1} \cdot \frac{L}{2^K+1} \cdot n$	$2 \cdot 2 \cdot \frac{M}{2^K} \cdot \frac{L}{2^K}$
PoolK (Activations)	$\frac{M}{2^K} \cdot \frac{L}{4} \cdot n$	$2 \cdot 2 \cdot \frac{M}{2^K} \cdot \frac{L}{2^K}$
FC Classifier (Weights)	$\frac{M}{2^{K+1}} \cdot \frac{L}{2^{K+1}} \cdot n \cdot 3 + 3$	-
FC Classifier (Activations)	3	$\frac{M}{2^{K+1}} \cdot \frac{L}{2^{K+1}} \cdot n \cdot 3$

Table 4: Memory footprint and computational demand of the proposed classification network after the first optimization

4.7 Average pooling optimization

The second optimization consists in the addition of a final average pooling layer, right after the last convolutional block and before the classification part of the network. This solution is inspired by many image recognition networks, like [30], and it helps reduce the size of the network. The aim of this optimization is to evaluate weather or not the implementation of an average pooling layer improves the results of the network and if it translates into a noticeable reduction of the network's demands.

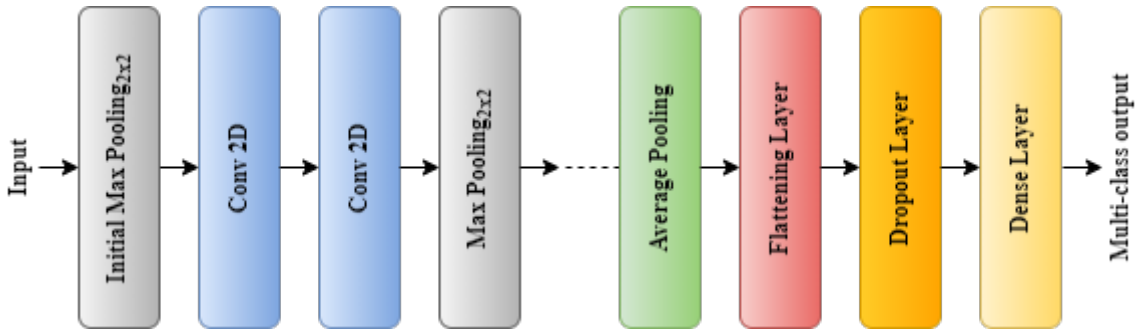

Figure 12: Proposed Network Architecture with the additional Average Pooling optimization

An average pooling layer returns the average value of the selected patch of the image. In the case of our network, we want to reduce the whole image to a single value. However, depending on how many convolutional blocks come before this layer, the input size of this pooling layer is variable. The algorithm to calculate the input size is as follows:

Algorithm 4 Calculation for the average pooling input size

1: $width = math.floor(53/2)$
2: $height = math.floor(86/2)$
3: **for** each convolutional block **do**
4: $\quad width = math.floor(width/2)$
5: $\quad height = math.floor(height/2)$
6: **end for**

The modification of the network in terms of memory footprint and computational demand comes after the convolutional part of the network. The details are collected in Table 5

	Memory footprint (Bytes)	Number of operations
ConvK_00 (Weights)	$r^2 \cdot n^2 + n$	-
ConvK_00 (Activations)	$\frac{M}{2^K} \cdot \frac{L}{2^K} \cdot n$	$r^2 \cdot n^2 \cdot \frac{M}{2^K} \cdot \frac{L}{2^K}$
ConvK_01 (Weights)	$r^2 \cdot n^2 + n$	-
ConvK_01 (Activations)	$\frac{M}{2^K} \cdot \frac{L}{2^K} \cdot n$	$r^2 \cdot n^2 \cdot \frac{M}{2^K} \cdot \frac{L}{2^K}$
PoolK (Weights)	-	-
PoolK (Activations)	$\frac{M}{2^{K+1}} \cdot \frac{L}{2^{K+1}} \cdot n$	$2 \cdot 2 \cdot \frac{M}{2^K} \cdot \frac{L}{2^K}$
PoolAvg0 (Weights)	-	-
PoolAvg0 (Activations)	$1 \cdot 1 \cdot n$	$\frac{M}{2^{K+1}} \cdot \frac{L}{2^{K+1}}$
FC Classifier (Weights)	$1 \cdot 1 \cdot n \cdot 3 + 3$	-
FC Classifier (Activations)	3	$1 \cdot 1 \cdot n \cdot 3$

Table 5: Memory footprint and computational demand of the proposed classification network after the second optimization

4.8 Depthwise separable convolution optimization

Depthwise separable convolutions were designed as an alternative to regular convolutions. Their main distinction is having less parameters, in order to avoid overfitting. They achieve this feat by combining deptwhise convolutions followed by pointwise convolutions.

The basic principle to achieve this low number of parameters is to split the input into single channels, then applying the filters and finally combining the results with a pointwise convolution (of kernel size 1x1). Table 6 shows the number of operations comparison between a regular convolution and a depthwise separable convolution; in the example, an image of size 8x8x3 is passed through a convolution of kernel 5x5 with 256 filters.

	Regular conv.	Dw separable conv.	
Number of operations	$(8 \cdot 8) \cdot (5 \cdot 5 \cdot 3) \cdot 256$	$(8 \cdot 8) \cdot (5 \cdot 5 \cdot 1) \cdot 3$	Filtering
		$(8 \cdot 8) \cdot (1 \cdot 1 \cdot 3) \cdot 256$	Combining
Total operations	$1,228,800$	$53,952$	

Table 6: Comparison example between regular convolution and depthwise separable convolution

This optimization aims to use depthwise separable convolutions, replacing the regular ones, in order to both reducing overfitting in the network and optimizing its memory footprint and

computational demands. Many prevalent examples can be found in the literature that use depthwise convolutions. 2017's MobileNet [30] stands out.

For each depthwise separable convolution, we can compute m_w^{DW}, m_a^{DW} and n_{ops}^{DW} as follows:

$$m_w^{DW} = r^2 \cdot n + n$$
$$m_a^{DW} = M_{in} \cdot L_{in} \cdot n$$
$$n_{ops}^{DW} = (M_{in} \cdot L_{in}) \cdot n_{in} \cdot (r^2 + n)$$

When the regular convolutions are substituted by depthwise separable convolutions in the proposed architecture, we can calculate the updated memory footprint and computational demand of the network:

	Memory footprint (Bytes)	Number of operations
Input	$M \cdot L \cdot 1$	-
Pool0 (Weights)	-	-
Input-Pool0 (Activations)	$\frac{M}{2} \cdot \frac{L}{2} \cdot 1$	$2 \cdot 2 \cdot M \cdot L$
DWConv1_00 (Weights)	$r^2 \cdot n + n$	-
DWConv1_00 (Activations)	$\frac{M}{2} \cdot \frac{L}{2} \cdot n$	$(r^2 + n) \cdot \frac{M}{2} \cdot \frac{L}{2}$
DWConv1_01 (Weights)	$r^2 \cdot n + n$	-
DWConv1_01 (Activations)	$\frac{M}{2} \cdot \frac{L}{2} \cdot n$	$(r^2 + n) \cdot n \cdot \frac{M}{2} \cdot \frac{L}{2}$
Pool1 (Weights)	-	-
Pool1 (Activations)	$\frac{M}{4} \cdot \frac{L}{4} \cdot n$	$2 \cdot 2 \cdot \frac{M}{2} \cdot \frac{L}{2}$
DWConvK_00 (Weights)	$r^2 \cdot n + n$	-
DWConvK_00 (Activations)	$\frac{M}{2^K} \cdot \frac{L}{2^K} \cdot n$	$(r^2 + n) \cdot n \cdot \frac{M}{2^K} \cdot \frac{L}{2^K}$
DWConvK_01 (Weights)	$r^2 \cdot n + n$	-
DWConvK_01 (Activations)	$\frac{M}{2^K} \cdot \frac{L}{2^K} \cdot n$	$(r^2 + n) \cdot n \cdot \frac{M}{2^K} \cdot \frac{L}{2^K}$
PoolK (Weights)	-	-
PoolK (Activations)	$\frac{M}{2^{K+1}} \cdot \frac{L}{2^{K+1}} \cdot n$	$2 \cdot 2 \cdot \frac{M}{2^K} \cdot \frac{L}{2^K}$
FC Classifier (Weights)	$\frac{M}{2^{K+1}} \cdot \frac{L}{2^{K+1}} \cdot n \cdot 3 + 3$	-
FC Classifier (Activations)	3	$\frac{M}{2^{K+1}} \cdot \frac{L}{2^{K+1}} \cdot n \cdot 3$

Table 7: Memory footprint and computational demand of the proposed classification network after the third optimization

4.9 Optimizations in image classification

Similarly to the work in **Chapter 3.7**, we will also test the strength of these optimizations in an image classification environment. Again, we will be using the Tiny ImageNet dataset [6]. This test will allow us to better understand the results obtained from these optimizations in our UWB radar environment.

The same adaptation has been carried out, with the main modification being the input size, from 53 x 86 x 1 inputs to 64 x 64 x 3. While the image sizes are similar, the 3 color spaces add to the weight of the network. This is not a big concern, since the networks are trained only for comparison and there are no plans for deploying the network into the microcontroller.

Because the input size changes, the algorithm for calculating the average pooling input dimensions has been modified:

Algorithm 5 Calculation for the average pooling input size in image classification

1: $width = math.floor(64/2)$
2: $height = math.floor(64/2)$
3: **for** each convolutional block **do**
4: $\quad width = math.floor(width/2)$
5: $\quad height = math.floor(height/2)$
6: **end for**

For the rest of the optimizations, no modifications have to be applied. The results of all of these experiments will be discussed in **Chapter 6**.

5 Person tracking and distance measuring

This part of the thesis aims to approach the problem as an object detection one. For this purpose, an object detection network has been designed from scratch. It takes inspiration from the design of the state of the art network for object detection on visual data (YOLO [7]). Its purpose is to be able to perform object detection on our UWB radar signals, instead of regular colored images. However, the dataset that we have used in our previous experiments can not be used for this task, mainly due to the difficulty of applying bounding box labels to it. To tackle this issue, a new dataset has been manually collected in a closed room environment with the same device.

5.1 Dataset

In order to implement an object detection solution, some sort of bounding boxes have to be applied, referencing where the people are in the range-Doppler map. The previous dataset was particularly difficult to label in this way. As we can see in Figures 13 and 14, it is very difficult to visualize how many people are in the heat map and more so exactly where they are. This is due to noise, the presence of multiple people in a short distance and the lack on information on the distance of each person.

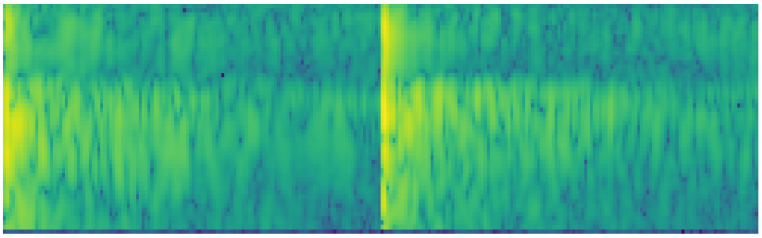

Figure 13: Original dataset sample representing an adult on seat 1, while the rest are empty

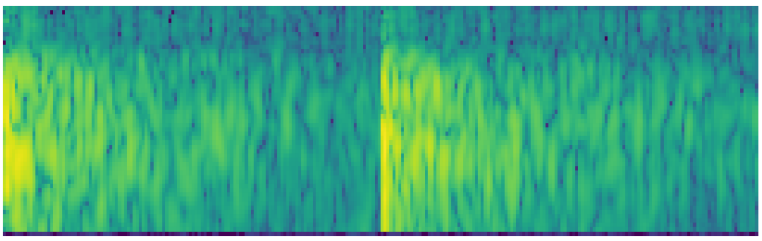

Figure 14: Original dataset sample representing a toddler, an adult and a pet on seats 1 to 3

To tackle this problem, a new dataset has been collected. The data has been manually acquired in an indoor setting by an ESP32 microcontroller unit with a UWB radar module, equipped with a couple of TX and RX antennas. The radar sensor is the same device used in the previous chapters. Most of the samples have 1 still person present, at varying distances from the sensor, which was also placed in different rooms and positions. The network has been trained with 1 person samples.

The radar module is capable of performing a scan of the environment, this time returning a 149-values-long series of imaginary values representing energy intensities, where the first value

represents the spatial bin closer to the device, and the 149th the most distant one. The detection range has been increased due to the change of environment, from a car to an indoor space. The radar can detect subjects in a 60º cone from where it's directed. The same scan is repeated over a 20 second period, at 10 Hz, resulting on a range-time map of 149 x 201 for each dataset sample.

The dataset only includes the folder where the sample was saved, the number of people in the sample and the data itself. As can be seen in the following examples, the presence and position of the individual is clearer, and allows us to allocate bounding boxes to the image with ease:

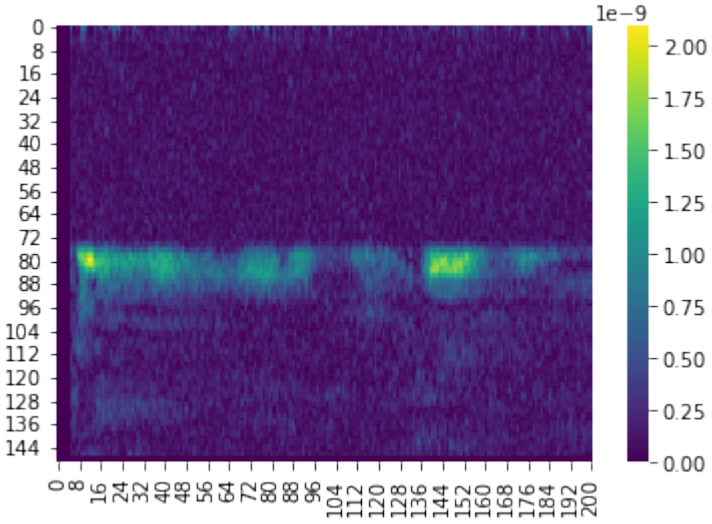

Figure 15: Raw Data Example. The person captured is represented by a higher signal intensity in the middle of the image, meaning they are at a medium distance from the sensor

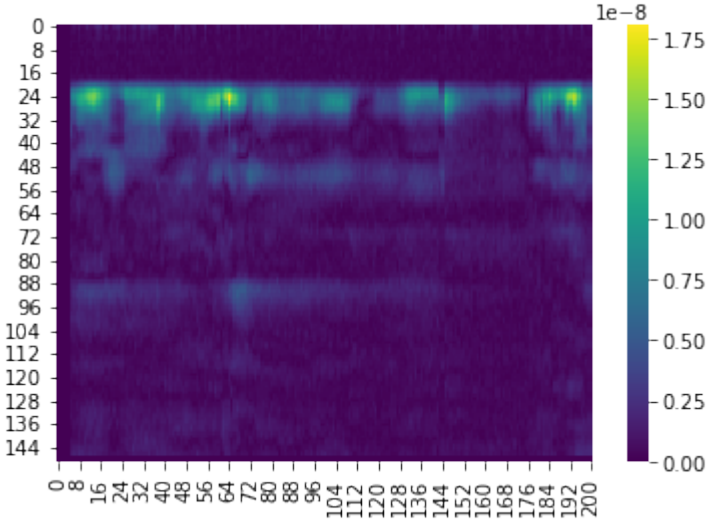

Figure 16: Raw Data Example. The person captured is represented by a higher signal intensity in the top part of the image, meaning they are closer to the sensor

The bounding box labels follow the principle of the YOLO algorithm [7]. However, they have been simplified in order to suit our problem and our network requirements. They have only an upper and a lower bound, since the person is standing still during recording, while horizontally they occupy the whole image. We are able to label the images like this because of the targets

being still. Some examples can be found in Figure 17:

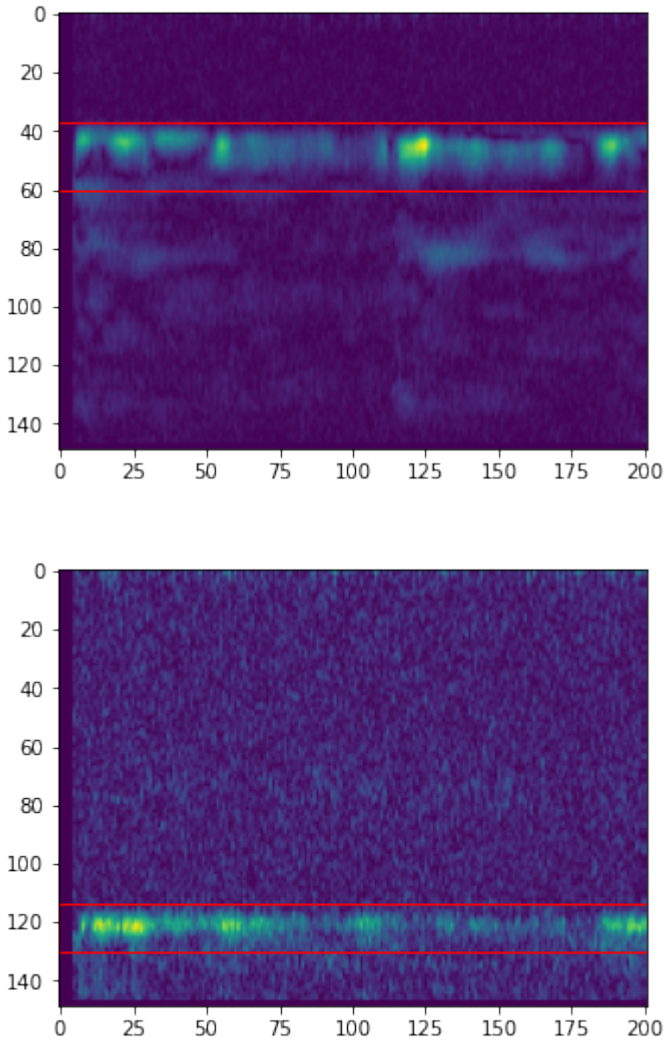

Figure 17: Raw Data bounding box examples

The images are divided in 7 horizontal segments in order to train the network, as will be detailed later. It is also a very useful tool to check the balancing of the dataset. If the center of the bounding box is found inside segment 0 (the top segment, closest to the sensor), then it is classified as a segment 0 bounding box. This way we can check the balancing of the data.

5.2 Problem formulation

The problem of this chapter can be formalized as an object detection task where the output must determine the position of a person in the image. In order to do this, the network approximates the center and height of the bounding box that contains the person. A regression task requires to approximate the following functions:

$$f(x) = \text{the position of the center of the bounding box}$$

$$g(x) = \text{the height of the bounding box}$$

The dataset for this problem has some unbalance among the different segments. As we can see in Figure 18, there are only a few examples of bounding boxes in the last segments, due to the limitations of space at the time of recording.

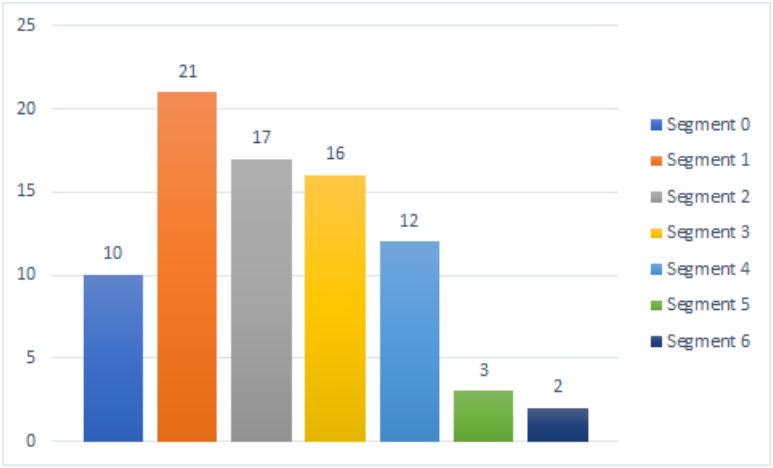

Figure 18: Data distribution for each segment

5.3 Data preprocessing

As we discussed previously, the bounding boxes are labeled with two values: an upper and a lower bound. Horizontally they occupy the whole image. In order to train our network, however, we want to modify this values as inspired by the YOLO algorithm [7], mainly for lone reason: although our dataset is composed by single-target samples, we want our model to be a generalist, i.e.,to be able to use it for multi-target detection. The YOLO algorithm allows us to do this. The steps to modify these labels are the following:

- Firstly, the image is divided in 7 segments, as explained previously:

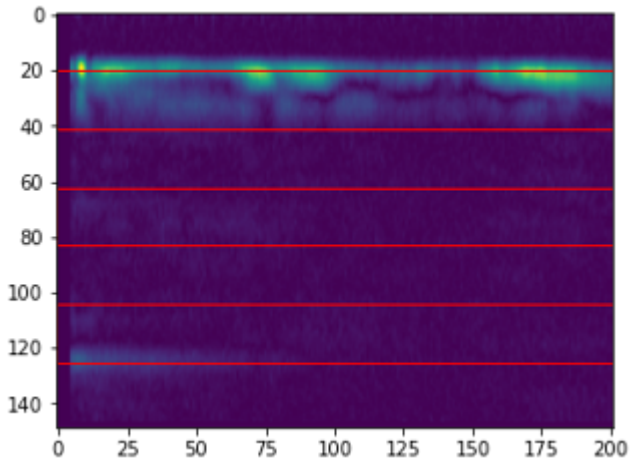

Figure 19: Segment distribution

- For each sample, we need to check in what segment the center is, and also the height of the bounding box. We use a simple algorithm:

Algorithm 6 Calculation for the center and height of a bounding box

1: $x_1, x_2 =$ lower bound, upper bound
2: $c = (x_1 + x_2)/2$
3: $h = x_2 - x_1$

- Now we check the segment where the center is located. Since segments are 21 values tall, this is a simple floor division between center and segment height.
- Finally, we use another algorithm to be able to transform this values into input data:

Algorithm 7 Transformation into input data

```
1: img_boxes = []
2: for each segment do
3:     segment_box = []
4:     if bounding box center is in this segment then
5:         segment_box = [1, ((box[0]-21*math.floor(box[0]/21))/21), (box[1]/21)]
6:     else
7:         segment_box = [0, 0, 0]
8:     end if
9:     bboxes.append(img_boxes)
10: end for
```

For each sample, each of the 7 segments has 3 values: the first is a binary value that indicates whether or not the bounding box center is in this segment, the second is a value between 0 and 1 which indicates the center's position relative to the segment size, and the last value is the height of the bounding box, also relative to the segments height (it can be higher than 1 if the bounding box is taller than the segment). The result is an array of 7x3 values. An example can be found in Figure 20. The resulting array will be falttened to feed it into the network as an input.

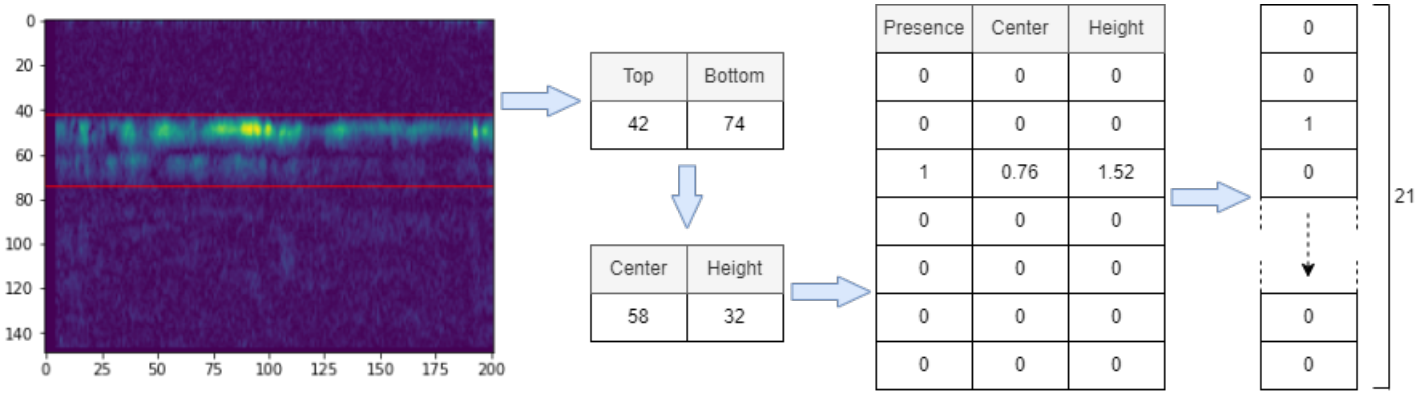

Figure 20: Preprocessing steps and results

In this data preprocessing we can see a very simplified implementation. The aim is small network and thus a small enough input so that the whole process can be carried out by a microcontroller. This, of course, comes with some drawbacks. With this segmentation, the theoretical maximum number of people detected in one sample is 7, if they are distant apart enough from each other. At the same time, if two targets stand at a similar distance from the

radar, they will most likely be categorized as one. This last topic is not a big concern, since it is already very difficult for UWB technology to distinguish people at the same distance. Different segmentation values are at this time under investigation. In the case of our implementation, with the use of single-target samples, this is not relevant.

5.4 Network requirements

We face the same network requirements as the previous chapters, taking into account the limitations of the ESP32 Microcontroller where the networks are planned to be deployed. However, a more relaxed approach will be followed, as the main aim of this chapter is to find a functional solution, and not so much optimizing the computational load and network size. For more information on the network requirements check **Chapter 3.4**.

5.5 Network architecture

For this problem, the proposed network architecture is based on the architecture discussed in **Chapter 3**. The main difference is, once again, the output: while the original model has a multi-class classification output, the output for this model has a regression output of 21 values. These correspond to the 7x3 arrays, flattened, that we use as input to the network. The proposed architecture is the following:

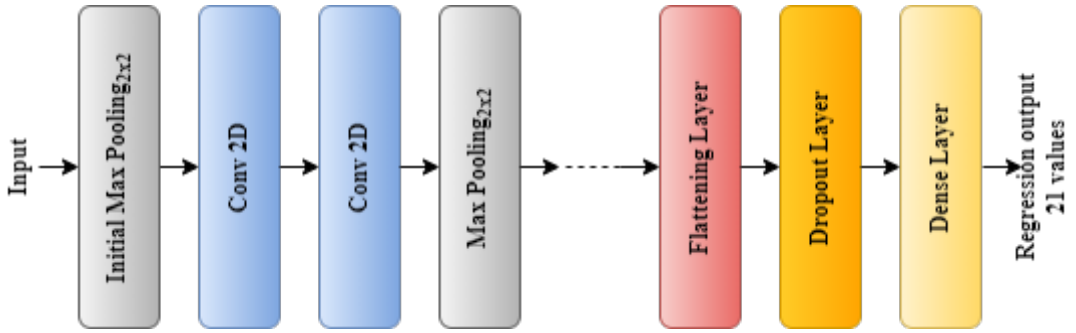

Figure 21: Object Detection Network Architecture

Since we have a regression output, the activation is linear, and the loss function used is 'mean squared error'. However, inspired by the YOLO model [7], we will also implement a custom loss function that aims to adapt to our network more efficiently.

5.6 Custom loss function

As we explained previously, the output of the network consists on a 7x3 array, where each column represents a different value in each segment. As inspired by YOLO model [7], a custom loss function has been tested in addition to MSE. In this algorithm, the following custom loss function is used:

$$\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (3)$$

Figure 22: YOLO custom loss function

In this formula, S^2 corresponds to the number of segments, B to the number of bounding boxes per segment, x and y to the coordinates of the center of the bounding box, w and h to the width and height of the bounding box, C to the presence of any object and p to each probability that the object is part of each class. $\mathbb{1}^{obj}$ and $\mathbb{1}^{noobj}$ are boolean values that correspond to whether there's an object or not in the segment, respectively, and all the different λ are different weightings to each part of the equation.

In the YOLO model, among other features, the network can distinguish 20 distinct objects from each other, and the segments that divide the image consist of horizontal and vertical divisions. Our model is much more simple, with only 1 class of object (a person), vertical segmentation and only one predicted bounding box per segment. This way, we are able to simplify this function to suit our needs:

$$\lambda_{obj} \sum_{i=0}^{S} \mathbb{1}_{i}^{obj} (y_i - \hat{y}_i)^2 + \lambda_{obj} \sum_{i=0}^{S} \mathbb{1}_{i}^{obj} (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 + \sum_{i=0}^{S} \mathbb{1}_{i}^{obj} (p_i - \hat{p}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S} \mathbb{1}_{i}^{noobj} (p_i - \hat{p}_i)^2$$

In our adapted formula, S corresponds to the number of vertical segments, y to the vertical coordinate of the center of the bounding box, h to the height of the bounding box and p to the probability that there is a bounding box. $\mathbb{1}^{obj}$ and $\mathbb{1}^{noobj}$ are boolean values that correspond to whether there's an object or not in the segment, respectively, and all the different λ are different weightings to each part of the equation.

5.7 Metrics

The metrics used to evaluate the performance of the network are not as straight forward as they may seem.

In the YOLO algorithm, a metric called 'mean average precision' is used. It is a pretty complex metric that takes into account the IOU of each object predicted, while also factoring in the False

Positives and False Negatives of the network. In our case this is not necessary, since our model does not predict multiple classes of objects and it is trained and validated on 1 person images.

The main metric used to evaluate our model is average IOU, which is the average of IOU obtained for every prediction, as detailed in **Chapter 2.3.6**. Other metrics are also considered, such as mean average error and mean squared error. However, none of this metrics are ideal. Since the bounding boxes are labelled manually, an optimal IOU value, or 'mse' or 'mae' is close to impossible to obtain. We will see this in practice in the results chapter.

It should be noted that, if the model were to be used in further scenarios, like multi-class or multi-target object detection, a more complex set of metrics must be put in place, such as the aforementioned mean average precision.

6 Results

This chapter collects the results of the experiments detailed in the previous chapters. It also includes a detailed description of the parameter selection phase, the memory and computation calculations of the selected networks and the procedure to obtain the final results when training on the full dataset. The most relevant notebooks have been uploaded to GitHub [31].

6.1 Architecture study

Table 8 sums up the results from the different problems in **Chapter 3**. As we discussed earlier, the memory and computation of the image classification problem is not relevant to us, and thus hasn't been calculated.

Architecture	Accuracy (%)	Memory (kB)	c (10^6)
4-class classification	80.92 ± 1.36	42.5	13.27
3-class classification	90.83 ± 1.09	122.5	40.70
4-class regression	77.08 ± 1.56	42.5	13.27
3-class regression	84.61 ± 1.40	42.5	13.27
Image classification	85.07	–	–

Table 8: Chapter 3 results

As we can observe, the accuracy in the 4-class classification model is around 80%. We are more than satisfied with the results offered, specially taking into account the size of the dataset (having less than 500 samples) and the difficulty to distinguish how many people are there in a sample with the human eye.

When analyzing the 3-class classification results, we see a jump to around 90% accuracy. This increase should be coming from the grouping of classes 2 and 3, which correspond to 2 and 3 passengers respectively. This leads to a more balanced dataset, given that there were very few samples of 3 people in the car as we have seen in **Chapter 3.2**. While the best model shows a bigger network in size and computation, all of the models with 3-class classification outperformed the previous 4-class models.

The regression results are not very positive, with a drop to 77% in the 4-class scenario and a drop to 84.5% in 3 classes. We believe that the classification approach is showing better results because we are counting a small number of people. As we can see, the difference between regression and classification is greater in the 3-class scenario than in the 4-class one. While counting a small number of people may be a better task for a classification algorithm, regression might improve the results in a bigger number of people.

Finally, the 3-class image classification results go up to 85% accuracy. While this is not an impressive result, it shows the limits in capabilities of our tiny models, which are not that of the bigger CNN models.

We will now detail the process carried out to achieve these results, starting from the parameter

selection phase, followed by the calculation of memory and computation of the selected network and finally the results when training on the full dataset.

6.1.1 4-class classificaiton

Table 20, included in **Appendix C**, summarize the results obtained by the grid search using 10-fold Cross Validation. This grid search is carried out to obtain the best combination of parameters of the network. In total, 60 combinations of parameters were tested.

The best model obtained by the grid search consists of 7x7 kernels, 3 convolutional blocks, 12 filters and dilation rate 1. The memory and computational load of this model is detailed in Table 9.

	Memory footprint (Bytes)	Number of operations
Input	$*53 \cdot 86 \cdot 1 = 4558$	-
Pool0 (Weights)	-	-
Input-Pool0 (Activations)	$*26 \cdot 43 \cdot 1 = 1118$	$2 \cdot 2 \cdot 53 \cdot 86 = 18232$
Conv1_00 (Weights)	$49 \cdot 12 + 12 = 600$	-
Conv1_00 (Activations)	$26 \cdot 43 \cdot 12 = 13416$	$49 \cdot 12 \cdot 26 \cdot 43 = 657384$
Conv1_01 (Weights)	$49 \cdot 144 + 12 = 7086$	-
Conv1_01 (Activations)	$26 \cdot 43 \cdot 12 = 13416$	$49 \cdot 144 \cdot 26 \cdot 43 = 7888608$
Pool1 (Weights)	-	-
Pool1 (Activations)	$*13 \cdot 21 \cdot 12 = 3276$	$2 \cdot 2 \cdot 26 \cdot 43 = 9116$
Conv2_00 (Weights)	$49 \cdot 144 + 12 = 7086$	-
Conv2_00 (Activations)	$*13 \cdot 21 \cdot 12 = 3276$	$49 \cdot 144 \cdot 13 \cdot 21 = 1926288$
Conv2_01 (Weights)	$49 \cdot 144 + 12 = 7086$	-
Conv2_01 (Activations)	$*13 \cdot 21 \cdot 12 = 3276$	$49 \cdot 144 \cdot 13 \cdot 21 = 1926288$
Pool2 (Weights)	-	-
Pool2 (Activations)	$*6 \cdot 10 \cdot 12 = 720$	$2 \cdot 2 \cdot 13 \cdot 21 = 1092$
Conv3_00 (Weights)	$49 \cdot 144 + 12 = 7086$	-
Conv3_00 (Activations)	$*6 \cdot 10 \cdot 12 = 720$	$49 \cdot 144 \cdot 6 \cdot 10 = 423360$
Conv3_01 (Weights)	$49 \cdot 144 + 12 = 7086$	-
Conv3_01 (Activations)	$*6 \cdot 10 \cdot 12 = 720$	$49 \cdot 144 \cdot 6 \cdot 10 = 423360$
Pool3 (Weights)	-	-
Pool3 (Activations)	$*3 \cdot 5 \cdot 12 = 180$	$2 \cdot 2 \cdot 6 \cdot 10 = 240$
FC Classifier (Weights)	$3 \cdot 5 \cdot 12 \cdot 4 + 4 = 724$	-
FC Classifier (Activations)	$*4$	$3 \cdot 5 \cdot 12 \cdot 4 = 720$
Total	42484	13274688

Table 9: Memory footprint and computational demand of the best 4-class classification network (an asterisk marks the activations re-using such arrays)

Finally, our best architecture is trained on the full dataset, averaging the results of 40 different splits of train and test sets and calculating the confidence interval. The results are included in Table 8.

6.1.2 3-class classificaiton

Table 21, included in **Appendix C**, summarizes the results obtained by the grid search using 10-fold Cross Validation. This grid search is carried out to obtain the best combination of parameters of the network. In total, 31 combinations of parameters were tested.

The best model obtained by the grid search consists of 7x7 kernels, 2 convolutional blocks, 22 filters and dilation rate 1. The memory and computational load of this model is detailed in Table 10.

Finally, our best architecture is trained on the full dataset, averaging the results of 40 different splits of training and validation sets and calculating the confidence interval. The results are included in Table 8.

	Memory footprint (Bytes)	Number of operations
Input	$*53 \cdot 86 \cdot 1 = 4558$	-
Pool0 (Weights)	-	-
Input-Pool0 (Activations)	$*26 \cdot 43 \cdot 1 = 1118$	$2 \cdot 2 \cdot 53 \cdot 86 = 18232$
Conv1_00 (Weights)	$49 \cdot 22 + 22 = 1100$	-
Conv1_00 (Activations)	$26 \cdot 43 \cdot 22 = 24596$	$49 \cdot 22 \cdot 26 \cdot 43 = 1205204$
Conv1_01 (Weights)	$49 \cdot 484 + 22 = 23738$	-
Conv1_01 (Activations)	$26 \cdot 43 \cdot 22 = 24596$	$49 \cdot 484 \cdot 26 \cdot 43 = 26514488$
Pool1 (Weights)	-	-
Pool1 (Activations)	$*13 \cdot 21 \cdot 22 = 6138$	$2 \cdot 2 \cdot 26 \cdot 43 = 4472$
Conv2_00 (Weights)	$49 \cdot 484 + 22 = 23738$	-
Conv2_00 (Activations)	$*13 \cdot 21 \cdot 22 = 6006$	$49 \cdot 484 \cdot 13 \cdot 21 = 6474468$
Conv2_01 (Weights)	$49 \cdot 484 + 22 = 23738$	-
Conv2_01 (Activations)	$*13 \cdot 21 \cdot 22 = 6006$	$49 \cdot 484 \cdot 13 \cdot 21 = 6474468$
Pool2 (Weights)	-	-
Pool2 (Activations)	$*6 \cdot 10 \cdot 22 = 1386$	$2 \cdot 2 \cdot 13 \cdot 21 = 1092$
FC Classifier (Weights)	$6 \cdot 10 \cdot 22 \cdot 3 + 3 = 3963$	-
FC Classifier (Activations)	$*3$	$6 \cdot 10 \cdot 22 \cdot 3 = 3960$
Total	125469	40696384

Table 10: Memory footprint and computational demand of the best 3-class classification network (an asterisk marks the activations re-using such arrays)

6.1.3 Image classification

Table 22, included in **Appendix C**, summarizes the results obtained by the grid search. This time, the use of K-folds has been deemed unnecessary, due to the dataset being complete and balanced. Tiny ImageNet offers us 1000 samples per class only to train, and other 100 samples for validation. In total, 24 combinations of parameters were tested.

The best model obtained by the grid search consists of 5x5 kernels, 4 convolutional blocks, 20 filters and dilation rate 1. The memory and computational load of this model has not been calculated, since it only serves as comparison to the results of the network that use our own

dataset.

This architecture is already trained on the full dataset, and no averaging of results or confidence intervals have been computed. This is because the dataset is already split in training and validation sets, and there is no unbalance among classes.

6.1.4 Regression approach

While usually regression models use metrics such as 'mean average error', we have decided to compute the accuracy to better compare the results with our classification networks. The process is to train the network normally, with mean squared error loss, and then rounding the predicted values to integers. We can now calculate the accuracy of the regression networks.

No grid search or K-fold Cross Validation is used in this model's training. We depart from the 4-class model with the best results, which consists of 7x7 kernels, 2 convolutional blocks, 22 filters and dilation rate 1. The memory and computational load are approximately the same as the 4-class classification scenario, detailed in Table 9, since the only change is the last layer.

Different numbers of epochs were tested on this model, and the best results were obtained with 200 epochs, as Table 11 details:

Epochs	N. classes	Accuracy (%)	N. classes	Accuracy (%)
50	4	75.41 ± 1.42	3	83.26 ± 1.36
100	4	76.63 ± 1.38	3	84.34 ± 1.23
200	4	77.08 ± 1.56	3	84.61 ± 1.40
300	4	76.77 ± 1.48	3	84.24 ± 1.25

Table 11: Regression accuracy results by number of epochs

6.2 Person counting

Table 12 sums up the results from the different optimizations in **Chapter 4**.

Architecture	Accuracy (%)	Memory (kB)	c (10^6)
3-class classification	90.83 ± 1.09	122.5	40.70
3-class classification (RAM usage opt.)	95.73 ± 0.86	75.3	21.78
3-class classification (average pooling opt.)	78.82 ± 1.50	66.7	11.89
3-class classification (dw convolution opt)	88.99 ± 1.19	53.6	0.08

Table 12: Chapter 4 results: optimizations in UWB radar 3-class classification

As we can observe, the accuracy in the RAM usage optimization model is around 96%. We can see a considerable improvement in accuracy compared to the proposed architecture. The comparison between memory and computation is also very positive, due mainly to the low parameters of the network that obtained the best results.

The average pooling optimization shows the worst results by far, with a drop to around 79% accuracy. The decrease in memory and, specially, in computation, should be noted.

Finally, the dw optimization showed promising results. While the grid search showed poor results overall compared to the proposed network, the final results show impressive results in accuracy (up to 89%) and computation. Several architectures were tested in this experiment, and the one with the best results was surprisingly compact, as we will detail later.

For comparison, as it was done for the previous chapter, Table 13 sums up the results for the different optimizations in a 3-class image classification scenario. This comparison aims to shed more light to the reasons behind the poor results of applying the optimizations to our main scenario.

Architecture	Accuracy (%)
Image classification	85.07
Image classification (RAM usage opt)	87.50
Image classification (average pooling opt)	86.11
Image classification (RAM usage + avg. pooling)	90.28
Image classification (dw convolution opt)	84.38
Image classification (RAM usage + dw convolution)	87.15

Table 13: Chapter 4 results: optimizations in image 3-class classification

In the RAM usage optimization we can see an increase, from 85% to 87.5%, though smaller than the one in our UWB radar scenario. This shows that this optimization can also be useful in an image recognition scenario.

The average pooling optimization does show an improvement as well, up to 86%. This suggests that, while this optimization is useful for distinguishing 3-color images, it is not so useful when applying it to our UWB radar range-Doppler maps.

The combination of these two optimizations results in a further improvement up to 90% in

accuracy. This solidifies our previous conjectures towards these optimizations.

The dw optimziation shows a bit lower accuracy (85%) compared to the proposed network, which is to be expected since its purpose is to reduce the weight of the network. The result is similar to our main scenario.

When combining both RAM usage and dw optimizations, the result is an increase in accuracy, up to 87%, close to the RAM usage optimization on its own.

We will now detail the process carried out to achieve these results, starting from the parameter selection phase, followed by the calculation of memory and computation of the selected network and finally the results when training on the full dataset.

6.2.1 RAM usage optimization

Table 23, included in **Appendix D**, summarizes the results obtained by the grid search using 10-fold Cross Validation. This grid search is carried out to obtain the best combination of parameters of the network. In total, 25 combinations of parameters were tested.

The best model obtained by the grid search consists of 7x7 kernels, 2 convolutional blocks, 16 filters and dilation rate 1. The memory and computational load of this model is detailed in Table 14.

	Memory footprint (Bytes)	Number of operations
Input	$*53 \cdot 86 \cdot 1 = 4558$	-
Pool0 (Weights)	-	-
Input-Pool0 (Activations)	$*26 \cdot 43 \cdot 1 = 1118$	$2 \cdot 2 \cdot 53 \cdot 86 = 18232$
Conv1_00 (Weights)	$49 \cdot 16 + 16 = 800$	-
Conv1_00 (Activations)	$26 \cdot 43 \cdot 16 = 17888$	$49 \cdot 16 \cdot 26 \cdot 43 = 876512$
Conv1_01 (Weights)	$49 \cdot 256 + 16 = 12560$	-
Conv1_01 (Activations)	$26 \cdot 43 \cdot 16 = 17888$	$49 \cdot 256 \cdot 26 \cdot 43 = 14024192$
Pool1 (Weights)	-	-
Pool1 (Activations)	$*13 \cdot 21 \cdot 16 = 4464$	$2 \cdot 2 \cdot 26 \cdot 43 = 4472$
Conv2_00 (Weights)	$49 \cdot 256 + 16 = 12560$	-
Conv2_00 (Activations)	$*13 \cdot 21 \cdot 16 = 4368$	$49 \cdot 256 \cdot 13 \cdot 21 = 3424512$
Conv2_01 (Weights)	$49 \cdot 256 + 16 = 12560$	-
Conv2_01 (Activations)	$*13 \cdot 21 \cdot 16 = 4368$	$49 \cdot 256 \cdot 13 \cdot 21 = 3424512$
Pool2 (Weights)	-	-
Pool2 (Activations)	$*6 \cdot 10 \cdot 16 = 1008$	$2 \cdot 2 \cdot 13 \cdot 21 = 1092$
FC Classifier (Weights)	$6 \cdot 10 \cdot 16 \cdot 3 + 3 = 2883$	-
FC Classifier (Activations)	$*3$	$6 \cdot 10 \cdot 16 \cdot 3 = 2880$
Total	77139	21776404

Table 14: Memory footprint and computational demand of the best RAM usage optimized network (an asterisk marks the activations re-using such arrays)

Finally, our best architecture is trained on the full dataset, averaging the results of 40 different splits of training and validation sets and calculating the confidence interval. The results are

included in Table 12.

6.2.2 Average pooling optimization

Table 24, included in **Appendix D**, summarizes the results obtained by the grid search using 10-fold Cross Validation. This grid search is carried out to obtain the best combination of parameters of the network. In total, 36 combinations of parameters were tested.

The best model obtained by the grid search consists of 5x5 kernels, 3 convolutional blocks, 16 filters and dilation rate 1. The memory and computational load of this model is detailed in Table 15.

	Memory footprint (Bytes)	Number of operations
Input	$*53 \cdot 86 \cdot 1 = 4558$	-
Pool0 (Weights)	-	-
Input-Pool0 (Activations)	$*26 \cdot 43 \cdot 1 = 1118$	$2 \cdot 2 \cdot 53 \cdot 86 = 18232$
Conv1_00 (Weights)	$25 \cdot 16 + 16 = 416$	-
Conv1_00 (Activations)	$26 \cdot 43 \cdot 16 = 17888$	$25 \cdot 16 \cdot 26 \cdot 43 = 447200$
Conv1_01 (Weights)	$25 \cdot 256 + 16 = 6416$	-
Conv1_01 (Activations)	$26 \cdot 43 \cdot 16 = 17888$	$25 \cdot 256 \cdot 26 \cdot 43 = 7155200$
Pool1 (Weights)	-	-
Pool1 (Activations)	$*13 \cdot 21 \cdot 16 = 4464$	$2 \cdot 2 \cdot 26 \cdot 43 = 4472$
Conv2_00 (Weights)	$25 \cdot 256 + 16 = 6416$	-
Conv2_00 (Activations)	$*13 \cdot 21 \cdot 16 = 4368$	$25 \cdot 256 \cdot 13 \cdot 21 = 1747200$
Conv2_01 (Weights)	$25 \cdot 256 + 16 = 6416$	-
Conv2_01 (Activations)	$*13 \cdot 21 \cdot 16 = 4368$	$25 \cdot 256 \cdot 13 \cdot 21 = 1747200$
Pool2 (Weights)	-	-
Pool2 (Activations)	$*6 \cdot 10 \cdot 16 = 1008$	$2 \cdot 2 \cdot 13 \cdot 21 = 1092$
Conv3_00 (Weights)	$25 \cdot 256 + 16 = 6416$	-
Conv3_00 (Activations)	$*6 \cdot 10 \cdot 16 = 960$	$25 \cdot 256 \cdot 6 \cdot 10 = 384000$
Conv3_01 (Weights)	$25 \cdot 256 + 16 = 6416$	-
Conv3_01 (Activations)	$*6 \cdot 10 \cdot 16 = 960$	$25 \cdot 256 \cdot 6 \cdot 10 = 384000$
Pool3 (Weights)	-	-
Pool3 (Activations)	$*3 \cdot 5 \cdot 16 = 240$	$2 \cdot 2 \cdot 6 \cdot 10 = 240$
PoolAvg0 (Weights)	-	-
PoolAvg0 (Activations)	$*1 \cdot 1 \cdot 16 = 16$	$3 \cdot 5 = 15$
FC Classifier (Weights)	$1 \cdot 1 \cdot 16 \cdot 3 + 3 = 51$	-
FC Classifier (Activations)	$*3$	$1 \cdot 1 \cdot 16 \cdot 3 = 48$
Total	68323	11888899

Table 15: Memory footprint and computational demand of the best average pooling optimized network (an asterisk marks the activations re-using such arrays)

Finally, our best architecture is trained on the full dataset, averaging the results of 40 different splits of train and test sets and calculating the confidence interval. The results are included in Table 12.

6.2.3 Depthwise convolution optimization

Many experiments have been conducted using depthwise separable convolutions, and different architectures have been tested. This is due to the strong under-performance of the network. Finally, the architecture with the best results was found, using only one convolution. Apart from the positive results, this architecture is also very light, specially in the number of operations.

Table 25, included in **Appendix D**, summarizes the results obtained by the grid search using 10-fold Cross Validation. This grid search is carried out to obtain the best combination of parameters of the network. In total, 45 combinations of parameters were tested.

The best model obtained by the grid search consists of 3x3 kernels, 1 convolutional blocks, 12 filters and dilation rate 2. The memory and computational load of this model is detailed in Table 16.

	Memory footprint (Bytes)	Number of operations
Input	$*53 \cdot 86 \cdot 1 = 4558$	-
Pool0 (Weights)	-	-
Input-Pool0 (Activations)	$26 \cdot 43 \cdot 1 = 1118$	$2 \cdot 2 \cdot 53 \cdot 86 = 18232$
Conv1_00 (Weights)	$9 \cdot 12 + 12 = 120$	-
Conv1_00 (Activations)	$26 \cdot 43 \cdot 12 = 13416$	$(9 + 12) \cdot 26 \cdot 43 = 23478$
FC Classifier (Weights)	$26 \cdot 43 \cdot 12 \cdot 3 + 3 = 40251$	-
FC Classifier (Activations)	$*3$	$26 \cdot 43 \cdot 12 \cdot 3 = 40248$
Total	54905	81958

Table 16: Memory footprint and computational demand of the best depthwise convolution optimized network (an asterisk marks the activations re-using such arrays)

Finally, our best architecture is trained on the full dataset, averaging the results of 40 different splits of train and test sets and calculating the confidence interval. The results are included in Table 12.

6.2.4 Optimizations in image classification

Tables 26, 27, 28, 29 and 30, included in **Appendix D**, summarize the results obtained by the grid search. As with the previous image classification experiment, the use of K-folds has been deemed unnecessary, due to the dataset being complete and balanced. Tiny ImageNet offers us 1000 samples per class only to train, and other 100 samples for validation.

Since 5 different optimizations and combinations of them have been tested, Table 17 summarizes the details of each architecture: the number of parameter combinations tested, the best model and its final parameters.

The memory and computational load of these models have not been calculated, since they only serve as comparison to the results of the networks that use our own dataset. All the architectures that use Tiny ImageNet are already trained on the full dataset, and no averaging of results or confidence intervals have been computed. This is because the dataset is already split in training

Architecture	Combinations	Kernels	Conv. blocks	Filters	Dil. Rate
RAM usage opt	23	3x3	3	12	1
average pooling opt	24	3x3	3	16	1
RAM usage + avg. pooling	22	5x5	3	16	1
dw convolution opt	18	3x3	2	20	1
RAM usage + dw convolution	18	3x3	2	20	1

Table 17: Grid search summary with combinations of parameters tested, best models and their final parameters

and validation sets, and there is no unbalance among classes.

6.3 Person tracking and distance measuring

Table 18 sums up the results from the problem in **Chapter 5**.

Loss	Average IOU	Memory (kB)	c (10^6)
MSE loss	0.597 ± 0.022	340.913	100.72
Custom loss	0.460 ± 0.029	464.96	247.79

Table 18: Chapter 5 results

As we can observe, the average IOU value obtained through all the predictions of the network is around 0.60 in the MSE loss model. This may seem like poor performance, but it needs to be taken into account that IOU is a very sensible metric. While a perfect prediction, where the predicted bounding box aligns perfectly compared to the 'ground truth' manually assigned box (which already has a human error factor), would be equivalent to an IOU of 1, it is generally considered a value over 0.5 to be a good prediction, as stated in [27]. We can see an example of this in Figure 23. The memory and computational cost is too high for our microcontroller, and the model should be further optimized to reduce its size. The main source of the large dimensions of the network is the input size; reducing it might be the best path to follow in order to lower the network's requirements.

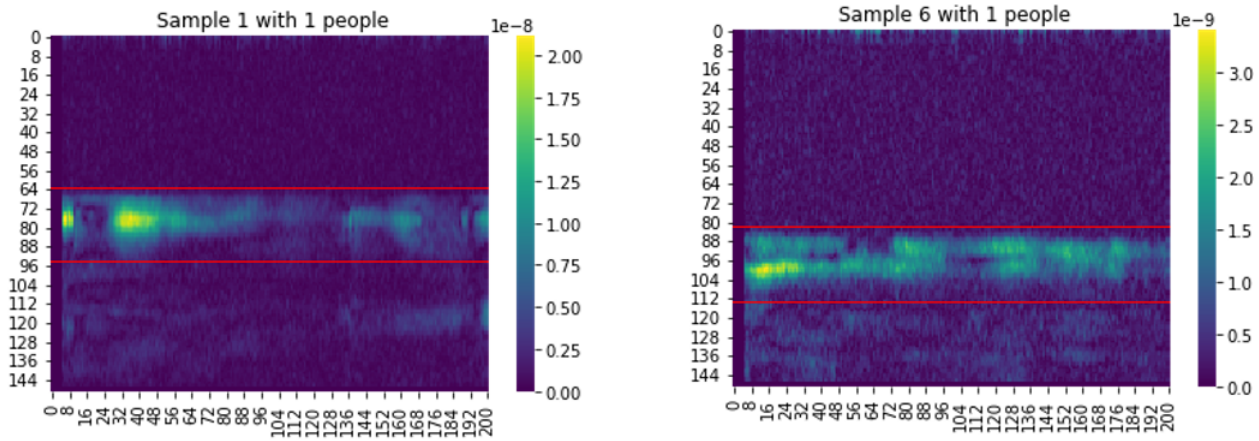

Figure 23: Prediction comparison of 0.95 IOU (left) vs 0.48 IOU (right)

The custom loss model, with a custom loss function detailed in **Chapter 5.6**, shows an average IOU of 0.40. The reason behind this under-performance might be related to the number of outputs of the network. While the YOLO model [7] used a custom loss function, the output was massive in comparison. Our model has 21 output parameters in total, while YOLO has 30 parameters per segment, and a greater number of segments per image. This may be the reason behind why MSE loss works better in our model.

Even though the results are far from optimal, we want to highlight the small size of the dataset, with 81 only samples. We want to note as well that, while a more specific model could have obtained much better results for the specific 1-person object detection task, the network has been designed as a generalist network, that trained with different data could perform multiple-person

object detection. We are convinced that, with a dataset enlargement, the results would increase by a good amount, and we are proud to pave the way to object detection in tiny devices with the use of UWB radar.

We will now detail the process carried out to achieve these results, starting from the parameter selection phase, followed by the calculation of memory and computation of the selected networks and finally the results when training on the full dataset.

6.3.1 Network architecture

After collecting all the samples in the dataset, the first tests on the complete dataset were performed. Table 31 details the first grid search performed by the MSE loss model, using 10-fold Cross Validation on 85% of the dataset. 48 combinations of parameters were tested. The result is the average IOU obtained by the best prediction on every one of the 81 samples, reaching a value of 0.15. This test helps us discard the use of average pooling, which almost always shows a worse performance than not using it.

In the second grid search, detailed in Table 32, average pooling was not considered, and more numbers of filters were tested. The full dataset was used this time, with 10-fold Cross Validation. In total, 40 combinations of parameters were tested. The result is the average IOU obtained by the best prediction on every one of the 81 samples, and shows an improvement over the previous grid search obtaining 0.22 average IOU. The selected model has 5x5 kernels, 3 convolutional blocks, 18 filters, dilation rate 1 and no average pooling.

After selecting the best parameters for the network, the memory and computation are calculated. These calculations are detailed in Table 33.

Finally, our best architecture is trained on the full dataset, averaging the results of 40 different splits of training and validation sets and calculating the confidence interval. The results are included in Table 18. All the tables can be doun in **Appendix E**.

6.3.2 Custom loss function

The same procedure is carried out with the custom loss function. Starting with the standard parameters for λ_{coord} and λ_{noobj}, which are 5 and 0.5 respectively, Table 34 details the first grid search performed by the custom loss model, using 10-fold Cross Validation on 85% of the dataset. 48 combinations of parameters were tested. The result is the average IOU obtained by the best prediction on every one of the 81 samples, reaching a value of 0.12. This test helps us discard the use of average pooling, which almost always shows a worse performance than not using it.

In the second grid search average pooling was not considered, and different values for λ_{coord} and λ_{obj} are tried, while λ_{noobj} remains 0.5. The full dataset was used this time, with 10-fold Cross Validation. In total, 144 combinations of parameters were tested, detailed in Tables 35 and 36. Since The result is the average IOU obtained by the best prediction on every one of the

81 samples, and shows an improvement over the previous grid search obtaining 0.15 average IOU. The selected model has 7x7 kernels, 5 convolutional blocks, 20 filters, dilation rate 1 and no average pooling, while λ_{coord} is 10 and λ_{obj} equals 3.

After selecting the best parameters for the network, the memory and computation are calculated. These calculations are detailed in Table 37. All the tables can be found in **Appendix E**.

Finally, our best architecture is trained on the full dataset, averaging the results of 40 different splits of training and validation sets and calculating the confidence interval. The results are included in Table 18.

7 Conclusions

In this section we present the contributions of the thesis, both general and academic, as well as future directions that are considered more interesting to be explored, ending with some acknowledgements.

7.1 General contributions

The first part of this research proposes several solutions to person counting in a car environment, and their corresponding convolutional neural network architectures that use UWB radar data have been designed. All these solutions accomplish our goals in privacy, memory and computation requirements. The best accuracy results come from the classification networks:

accuracy of 3-class classification model = **90.83**

accuracy of 4-class classification model = **80.92**

Different optimizations to these networks have been proposed as well, in order to either improve the results of the network or to improve the memory and computation:

accuracy of 3-class classification model (RAM usage opt.) = **95.73**

accuracy of 3-class classification model (dw convolution opt) = **88.99**

These classification networks and optimizations have also been successfully tested in an image classification scenario, proving that these tiny architectures can also be used in image classification.

The second part of this research proposes an object detection solution with he use of UWB radar signals in an indoor environment, and the corresponding convolutional neural network architecture has been designed. The proposed solution is a functional object detection model, while it does not achieve the memory and computation requirements. The results obtained by this network are the following:

average IOU of object detection model = **0.597**

While some of the requirements have not been accomplished, this model can serve as a baseline for object detection with UWB radar in TinyML, with the possibility of expanding the predictions to multiple targets.

7.2 Academic contributions

The research contributes to the few publicly available research on the topic of multi-target classification and person counting through the use of radar data, and specially to the field of

object detection with radar data. It also contributes to the relatively new and expanding field of TinyML, where very few object detection solutions can be found. To our knowledge, this is the first research that tries to use UWB radar on person counting and object detection.

7.3 Future directions

The main directions that we would like to take the research in the future are the following:

- **Deployment of the networks:** the full deployment of these networks on a microcontroller is the first logical step after this research. We have at our disposal several TinyML models designed for person counting, and testing them on a real scenario would add further value to our research, as well as helping us understand the differences and limitations of fully deploying these networks.
- **Enlarging the datasets:** one of the main problems when training the networks is the dataset size, specially on a convolutional network. While the first dataset has around 500 samples, there is a big unbalance, specially with the lack of 2 and 3 people samples. The second dataset only contains 81 samples, and an enlargement of this dataset would certainly improve our object detection networks.
- **Reducing the object detection network:** while we managed to design a very small object detection network, the requirements to be deployed on the ESP32 microcontroller have not been achieved. While the object detection network model is very similar to the classification models, the main difference is the input size. The best way to reduce the network would be further preprocessing the radar signals to reduce their size while losing the minimum amount of information. Some of the optimizations like dw convolutions could also be implemented.
- **Exploring other uses of the object detection network:** the object detection implementation opens the door to some other functionalities, like person tracking and distance measuring. With very simple changes to the network, solutions for these tasks could be implemented.
- **Performing multi-target object detection:** the object detection network has been designed to be able to escalate the number of the detections. With the current implementation, up to 7 targets could be detected at once, theoretically. While some preliminary experimentation has been carried out, the dataset plays a major factor again. We believe with a larger, more varied dataset, the network could perform multi-target object detection without modifications. We would need, however, more than one person samples for the training of this network.

7.4 Acknowledgments

The author would like to thank the following people:

Professor Manuel Roveri and Massimo Pavan, from Politecnico di Milano, for the opportunity that is working on this project, as well as for their professionalism and availability.

Luis Hernández Gómez, from ETSIT, UPM, for greenlighting, supervising and supporting the project, as well as putting his time and interest on it.

A particular thank you to Icíar Gómez and Isabel Santidrián, for their invaluable help in collecting dataset samples.

Gratitude to the Erasmus+ programme and everybody behind it, that has allowed me to carry out this work of research that I am so very proud of, in the hospitality of a neighbouring country.

A final thank you to my family and friends, those that have always supported me and have helped me achieve my goals, thanks to you I am the person that I am today.

A Ethical, economic, social and environmental aspects

A.1 Introduction

Our proposed solution consists of a UWB radar-based TinyML system for people counting and object detection. With it, we aim to tackle two main problems: the privacy problem related to current presence detection systems, which rely on video cameras and/or microphones, and the use of energy and resources, usually related to big-scale automated systems.

In this appendix we will discuss the main ethical, economic, social and environmental aspects, detail one of them and finally reflect on the impact of our proposed solution.

A.2 Description of relevant impacts related to the project

This section details the different impacts of the project:

Ethical and social impact: current automated presence detection systems rely on the processing of images taken from video cameras or microphones. While this is a very appropriate task for Machine Learning-based applications, the acquisition and processing of this data can lead to a risk in privacy vulneration, since images, videos or audio of people can be considered sensitive information. The use of UWB radar offers a solution to this, enabling us to design ML systems for person counting and object detection while tackling these privacy concerns. With our project we want to offer a solution that does not rely on sensitive information, lowering the privacy vulneration risks of our day-to-day life.

Economic and environmental impact: in the most recent technological landscape, tiny devices are becoming one of the main areas of technological breakthrough. As Internet-of-Things (IoT) units, embedded systems and edge devices become more present in the technological environment, the scientific trend reflects the displacement of data processing closer to where the data is generated. This aims to increase the autonomy of tiny devices, but also to achieve the most efficient use of energy possible, reducing as well the required bandwidth for this devices. TinyML offers a variety of tools to shrink down a neural network as much as possible, using methods such as quantization or pruning, in order to deploy them directly into tiny devices with very low resources, enabling their continued use for several weeks, or even months, with a single coin battery. With our project we want to offer a solution that makes the most efficient cost and energy use possible.

A.3 Detailed analysis of one of the main impacts

The main impact of the project is tackling the privacy vulneration risks of current automated person counting and object detection systems. The protection of privacy is paramount in our current societies, carrying with it an important ethical and social impact. The use of UWB radar offers a safer alternative to cameras and microphones in terms of privacy, allowing automated

systems to perform various functions like presence detection and person counting without any people involved being identifiable.

Our project is one of the first in the literature to successfully use this technology on person counting and object detection tasks, and we want our project to serve as a stepping stone for future research on this technology. There is no doubt that this technology is very promising, and in combination with TinyML, better and more varied solutions for UWB radar will come, and with them the consequent reduction in privacy vulneration risks.

A.4 Conclusions

The project accomplishes our goals in ethical, social, economic and environmental aspects. From a social and ethical standpoint, it contributes to privacy and data protection, directly (for the users to deploy our solutions) and indirectly (for those affected by them). From an economical and environmental standpoint, it accomplishes to reduce the cost and energy consumption of data processing, enabling cheap small devices like microcontroller to process data directly after collecting it by making use of energy in the most efficient way possible.

B Financial budget

This section presents an estimation of the economical budget. The total cost of the project is 13,614.44€.

Labour cost (direct cost)		**Hours**	**Price/hour**	**Total**
		810	15.00€	12,150.00€
Material resources (direct cost)	**Purchase price**	**Use (months)**	**Deprecation (years)**	**Total**
Personal computer	600.00€	6	5	60.00€
Other resources	40.00€	6	5	4.00€
Total material resources cost				64.00€
General costs (indirect cost)			15%	1,832.10€
Subtotal budget				14,046.10€
Aplicable vat			21%	2,949.68€
Total budget				16,995.78€

Table 19: Budget of the project

C Architecture study: grid search results

Conv kernel dim	n conv blocks	filters	dil. rate	accuracy	Conv kernel dim	n conv blocks	filters	dil. rate	accuracy
5x5	2	12	1	0.643	7x7	2	12	1	0.687
5x5	2	12	2	0.658	7x7	2	12	2	0.631
5x5	2	14	1	0.660	7x7	2	14	1	0.655
5x5	2	14	2	0.635	7x7	2	14	2	0.670
5x5	2	16	1	0.665	7x7	2	16	1	0.682
5x5	2	16	2	0.645	7x7	2	16	2	0.680
5x5	2	18	1	0.672	7x7	2	18	1	0.663
5x5	2	18	2	0.677	7x7	2	18	2	0.672
5x5	2	20	1	0.675	7x7	2	20	1	0.670
5x5	2	20	2	0.677	7x7	2	20	2	0.687
5x5	3	12	1	0.650	7x7	3	12	1	0.700
5x5	3	12	2	0.648	7x7	3	12	2	0.655
5x5	3	14	1	0.650	7x7	3	14	1	0.675
5x5	3	14	2	0.658	7x7	3	14	2	0.677
5x5	3	16	1	0.677	7x7	3	16	1	0.672
5x5	3	16	2	0.653	7x7	3	16	2	0.680
5x5	3	18	1	0.663	7x7	3	18	1	0.675
5x5	3	18	2	0.653	7x7	3	18	2	0.677
5x5	3	20	1	0.653	7x7	3	20	1	0.685
5x5	3	20	2	0.682	7x7	3	20	2	0.690
5x5	4	12	1	0.672	7x7	4	12	1	0.685
5x5	4	12	2	0.628	7x7	4	12	2	0.635
5x5	4	14	1	0.650	7x7	4	14	1	0.660
5x5	4	14	2	0.621	7x7	4	14	2	0.643
5x5	4	16	1	0.643	7x7	4	16	1	0.670
5x5	4	16	2	0.635	7x7	4	16	2	0.635
5x5	4	18	1	0.667	7x7	4	18	1	0.677
5x5	4	18	2	0.655	7x7	4	18	2	0.655
5x5	4	20	1	0.685	7x7	4	20	1	0.667
5x5	4	20	2	0.658	7x7	4	20	2	0.638

Table 20: 4-class classification grid search results

Conv kernel dim	n conv blocks	filters	dil. rate	accuracy
3x3	2	10	1	0.741
3x3	2	12	1	0.754
3x3	2	14	1	0.764
3x3	2	16	1	0.756
3x3	2	20	1	0.751
5x5	2	16	1	0.766
5x5	3	16	1	0.768
5x5	4	12	1	0.771
5x5	4	12	2	0.727
5x5	4	14	1	0.756
5x5	4	14	2	0.754
5x5	4	16	1	0.776
5x5	4	16	2	0.766
5x5	4	18	1	0.751
5x5	4	20	1	0.764
7x7	2	10	1	0.759
7x7	2	16	1	0.781
7x7	2	18	1	0.776
7x7	2	20	1	0.746
7x7	2	22	1	0.783
7x7	2	6	1	0.734
7x7	2	8	1	0.768
7x7	3	16	1	0.766
7x7	4	12	1	0.771
7x7	4	12	2	0.749
7x7	4	14	1	0.756
7x7	4	14	2	0.761
7x7	4	16	1	0.764
7x7	4	16	2	0.741
7x7	4	18	1	0.781
7x7	4	20	1	0.761

Table 21: 3-class classification grid search results

Conv kernel dim	n conv blocks	filters	dil. rate	accuracy
3x3	1	12	1	0.722
3x3	1	16	1	0.764
3x3	1	20	1	0.795
3x3	2	12	1	0.812
3x3	2	16	1	0.792
3x3	2	20	1	0.816
3x3	3	12	1	0.812
3x3	3	16	1	0.802
3x3	3	20	1	0.819
3x3	4	12	1	0.806
3x3	4	16	1	0.840
3x3	4	20	1	0.833
5x5	1	12	1	0.785
5x5	1	16	1	0.785
5x5	1	20	1	0.774
5x5	2	12	1	0.819
5x5	2	16	1	0.788
5x5	2	20	1	0.809
5x5	3	12	1	0.812
5x5	3	16	1	0.799
5x5	3	20	1	0.819
5x5	4	12	1	0.816
5x5	4	16	1	0.806
5x5	4	20	1	0.851

Table 22: Image classification grid search results

D Person Counting: grid search results

Conv kernel dim	n conv blocks	filters	dil. rate	accuracy
5x5	2	12	1	0.761
5x5	2	14	1	0.776
5x5	2	16	1	0.764
5x5	2	18	1	0.781
5x5	2	20	1	0.761
5x5	3	12	1	0.771
5x5	3	14	1	0.756
5x5	3	16	1	0.747
5x5	3	18	1	0.749
5x5	3	20	1	0.761
7x7	1	12	1	0.778
7x7	1	14	1	0.778
7x7	1	16	1	0.769
7x7	1	18	1	0.743
7x7	1	20	1	0.763
7x7	2	12	1	0.758
7x7	2	14	1	0.746
7x7	2	16	1	0.783
7x7	2	18	1	0.772
7x7	2	20	1	0.778
7x7	3	12	1	0.754
7x7	3	14	1	0.764
7x7	3	16	1	0.746
7x7	3	18	1	0.738
7x7	3	20	1	0.771

Table 23: RAM usage optimization grid search results

Conv kernel dim	n conv blocks	filters	dil. rate	accuracy
3x3	1	12	1	0.419
3x3	1	12	2	0.446
3x3	1	16	1	0.460
3x3	1	16	2	0.485
3x3	1	20	1	0.443
3x3	1	20	2	0.475
3x3	2	12	1	0.586
3x3	2	12	2	0.611
3x3	2	16	1	0.606
3x3	2	16	2	0.656
3x3	2	20	1	0.636
3x3	2	20	2	0.689
3x3	3	12	1	0.680
3x3	3	12	2	0.702
3x3	3	16	1	0.719
3x3	3	16	2	0.746
3x3	3	20	1	0.744
3x3	3	20	2	0.749
5x5	1	12	1	0.478
5x5	1	12	2	0.549
5x5	1	16	1	0.512
5x5	1	16	2	0.584
5x5	1	20	1	0.503
5x5	1	20	2	0.606
5x5	2	12	1	0.712
5x5	2	12	2	0.719
5x5	2	16	1	0.712
5x5	2	16	2	0.717
5x5	2	20	1	0.741
5x5	2	20	2	0.744
5x5	3	12	1	0.741
5x5	3	12	2	0.749
5x5	3	16	1	0.763
5x5	3	16	2	0.752
5x5	3	20	1	0.744
5x5	3	20	2	0.757

Table 24: Average pooling optimization grid search results

Conv kernel dim	n conv blocks	filters	dil. rate	accuracy
3x3	1	10	1	0.754
3x3	1	10	2	0.778
3x3	1	12	1	0.741
3x3	1	12	2	0.791
3x3	1	14	1	0.759
3x3	1	14	2	0.761
3x3	1	16	1	0.749
3x3	1	16	2	0.783
3x3	1	18	1	0.768
3x3	1	18	2	0.786
3x3	1	20	1	0.744
3x3	1	20	2	0.776
3x3	1	22	1	0.773
3x3	1	22	2	0.781
3x3	1	6	1	0.783
3x3	1	6	2	0.771
3x3	1	8	1	0.766
3x3	1	8	2	0.768
5x5	1	10	1	0.764
5x5	1	10	2	0.776
5x5	1	12	1	0.786
5x5	1	12	2	0.771
5x5	1	14	1	0.761
5x5	1	14	2	0.786
5x5	1	16	1	0.749
5x5	1	16	2	0.768
5x5	1	18	1	0.736
5x5	1	18	2	0.768
5x5	1	20	1	0.751
5x5	1	20	2	0.778
5x5	1	22	1	0.741
5x5	1	22	2	0.771
5x5	1	6	1	0.771
5x5	1	6	2	0.749
5x5	1	8	1	0.744
5x5	1	8	2	0.771
7x7	1	10	2	0.768
7x7	1	12	2	0.778
7x7	1	14	2	0.773
7x7	1	16	2	0.756
7x7	1	18	2	0.783
7x7	1	20	2	0.766
7x7	1	22	2	0.776
7x7	1	6	2	0.746
7x7	1	8	2	0.749

Table 25: Depthwise convolution optimization grid search results

Conv kernel dim	n conv blocks	filters	dil. rate	accuracy
3x3	1	12	1	0.781
3x3	1	16	1	0.799
3x3	1	20	1	0.781
3x3	2	12	1	0.812
3x3	2	16	1	0.830
3x3	2	20	1	0.847
3x3	3	12	1	0.875
3x3	3	16	1	0.847
3x3	3	20	1	0.854
3x3	4	12	1	0.830
3x3	4	16	1	0.844
3x3	4	20	1	0.851
5x5	1	12	1	0.743
5x5	1	16	1	0.757
5x5	1	20	1	0.792
5x5	2	12	1	0.826
5x5	2	16	1	0.823
5x5	2	20	1	0.833
5x5	3	12	1	0.840
5x5	3	16	1	0.816
5x5	3	20	1	0.826
5x5	4	12	1	0.826
5x5	4	16	1	0.854

Table 26: Image classification with RAM usage optimization grid search results

Conv kernel dim	n conv blocks	filters	dil. rate	accuracy
3x3	1	12	1	0.691
3x3	1	16	1	0.729
3x3	1	20	1	0.740
3x3	2	12	1	0.757
3x3	2	16	1	0.788
3x3	2	20	1	0.799
3x3	3	12	1	0.812
3x3	3	16	1	0.861
3x3	3	20	1	0.861
3x3	4	12	1	0.809
3x3	4	16	1	0.833
3x3	4	20	1	0.837
5x5	1	12	1	0.719
5x5	1	16	1	0.722
5x5	1	20	1	0.757
5x5	2	12	1	0.819
5x5	2	16	1	0.816
5x5	2	20	1	0.840
5x5	3	12	1	0.851
5x5	3	16	1	0.844
5x5	3	20	1	0.861
5x5	4	12	1	0.858
5x5	4	16	1	0.847
5x5	4	20	1	0.830

Table 27: Image classification with average pooling optimization grid search results

Conv kernel dim	n conv blocks	filters	dil. rate	accuracy
3x3	1	12	1	0.677
3x3	1	16	1	0.743
3x3	1	20	1	0.722
3x3	2	12	1	0.812
3x3	2	16	1	0.865
3x3	2	20	1	0.896
3x3	3	12	1	0.878
3x3	3	16	1	0.878
3x3	3	20	1	0.885
3x3	4	12	1	0.830
3x3	4	16	1	0.878
3x3	4	20	1	0.851
5x5	1	12	1	0.733
5x5	1	16	1	0.750
5x5	1	20	1	0.764
5x5	2	12	1	0.868
5x5	2	16	1	0.854
5x5	2	20	1	0.899
5x5	3	12	1	0.840
5x5	3	16	1	0.903
5x5	3	20	1	0.858
5x5	4	12	1	0.809

Table 28: Image classification with RAM usage optimization and average pooling optimization grid search results

Conv kernel dim	n conv blocks	filters	dil. rate	accuracy
3x3	1	12	1	0.778
3x3	1	16	1	0.778
3x3	1	20	1	0.812
3x3	2	12	1	0.806
3x3	2	16	1	0.799
3x3	2	20	1	0.844
3x3	3	12	1	0.771
3x3	3	16	1	0.802
3x3	3	20	1	0.802
5x5	1	12	1	0.778
5x5	1	16	1	0.799
5x5	1	20	1	0.802
5x5	2	12	1	0.840
5x5	2	16	1	0.837
5x5	2	20	1	0.816
5x5	3	12	1	0.788
5x5	3	16	1	0.792
5x5	3	20	1	0.795

Table 29: Image classification with dw convolution optimization grid search results

Conv kernel dim	n conv blocks	filters	dil. rate	accuracy
3x3	1	12	1	0.781
3x3	1	16	1	0.802
3x3	1	20	1	0.792
3x3	2	12	1	0.872
3x3	2	16	1	0.861
3x3	2	20	1	0.847
3x3	3	12	1	0.851
3x3	3	16	1	0.833
3x3	3	20	1	0.865
5x5	1	12	1	0.785
5x5	1	16	1	0.806
5x5	1	20	1	0.837
5x5	2	12	1	0.844
5x5	2	16	1	0.840
5x5	2	20	1	0.840
5x5	3	12	1	0.858
5x5	3	16	1	0.837
5x5	3	20	1	0.868

Table 30: Image classification with RAM usage optimization and dw convolution optimization grid search results

E Person tracking and distance measuring: grid search results

n conv blocks	filters	dil. rate	avg. pooling	Conv kernel dim	avg. IOU	Conv kernel dim	avg. IOU
3	12	1	False	5x5	0.10	7x7	0.15
3	12	1	True	5x5	0.03	7x7	0.07
3	16	1	False	5x5	0.10	7x7	0.13
3	16	1	True	5x5	0.04	7x7	0.07
3	20	1	False	5x5	0.10	7x7	0.14
3	20	1	True	5x5	0.06	7x7	0.07
4	12	1	False	5x5	0.10	7x7	0.08
4	12	1	True	5x5	0.08	7x7	0.05
4	16	1	False	5x5	0.12	7x7	0.08
4	16	1	True	5x5	0.06	7x7	0.07
4	20	1	False	5x5	0.11	7x7	0.09
4	20	1	True	5x5	0.10	7x7	0.08
5	12	1	False	5x5	0.07	7x7	0.10
5	12	1	True	5x5	0.04	7x7	0.07
5	16	1	False	5x5	0.09	7x7	0.11
5	16	1	True	5x5	0.07	7x7	0.07
5	20	1	False	5x5	0.09	7x7	0.10
5	20	1	True	5x5	0.09	7x7	0.12
6	12	1	False	5x5	0.06	7x7	0.05
6	12	1	True	5x5	0.06	7x7	0.07
6	16	1	False	5x5	0.06	7x7	0.06
6	16	1	True	5x5	0.08	7x7	0.06
6	20	1	False	5x5	0.11	7x7	0.09
6	20	1	True	5x5	0.08	7x7	0.10

Table 31: MSE loss model grid search results 1

Conv kernel dim	n conv blocks	filters	dil. rate	avg. pooling	avg. IOU
5x5	3	12	1	False	0.13
5x5	3	14	1	False	0.12
5x5	3	16	1	False	0.15
5x5	3	18	1	False	0.22
5x5	3	20	1	False	0.17
5x5	4	12	1	False	0.13
5x5	4	14	1	False	0.16
5x5	4	16	1	False	0.16
5x5	4	18	1	False	0.12
5x5	4	20	1	False	0.14
5x5	5	12	1	False	0.11
5x5	5	14	1	False	0.13
5x5	5	16	1	False	0.15
5x5	5	18	1	False	0.14
5x5	5	20	1	False	0.18
5x5	6	12	1	False	0.07
5x5	6	14	1	False	0.10
5x5	6	16	1	False	0.08
5x5	6	18	1	False	0.06
5x5	6	20	1	False	0.09
7x7	3	12	1	False	0.15
7x7	3	14	1	False	0.15
7x7	3	16	1	False	0.15
7x7	3	18	1	False	0.15
7x7	3	20	1	False	0.11
7x7	4	12	1	False	0.14
7x7	4	14	1	False	0.11
7x7	4	16	1	False	0.17
7x7	4	18	1	False	0.11
7x7	4	20	1	False	0.18
7x7	5	12	1	False	0.14
7x7	5	14	1	False	0.12
7x7	5	16	1	False	0.13
7x7	5	18	1	False	0.18
7x7	5	20	1	False	0.17
7x7	6	12	1	False	0.10
7x7	6	14	1	False	0.10
7x7	6	16	1	False	0.10
7x7	6	18	1	False	0.09
7x7	6	20	1	False	0.10

Table 32: MSE loss model grid search results 2

	Memory footprint (Bytes)	Number of operations
Input	$*149 \cdot 201 \cdot 1 = 29949$	-
Pool0 (Weights)	-	-
Input-Pool0 (Activations)	$*74 \cdot 100 \cdot 1 = 7437$	$2 \cdot 2 \cdot 149 \cdot 201 = 119796$
Conv1_00 (Weights)	$25 \cdot 18 + 18 = 468$	-
Conv1_00 (Activations)	$74 \cdot 100 \cdot 18 = 133200$	$25 \cdot 18 \cdot 74 \cdot 100 = 3330000$
Conv1_01 (Weights)	$25 \cdot 324 + 18 = 8118$	-
Conv1_01 (Activations)	$74 \cdot 100 \cdot 18 = 133200$	$25 \cdot 324 \cdot 74 \cdot 100 = 59940000$
Pool1 (Weights)	-	-
Pool1 (Activations)	$*37 \cdot 50 \cdot 18 = 33300$	$2 \cdot 2 \cdot 74 \cdot 100 = 29600$
Conv2_00 (Weights)	$25 \cdot 324 + 18 = 8118$	-
Conv2_00 (Activations)	$*37 \cdot 50 \cdot 18 = 33300$	$25 \cdot 324 \cdot 37 \cdot 50 = 14985000$
Conv2_01 (Weights)	$25 \cdot 324 + 18 = 8118$	-
Conv2_01 (Activations)	$*37 \cdot 50 \cdot 18 = 33300$	$25 \cdot 324 \cdot 37 \cdot 50 = 14985000$
Pool2 (Weights)	-	-
Pool2 (Activations)	$*18 \cdot 25 \cdot 18 = 8100$	$2 \cdot 2 \cdot 37 \cdot 50 = 7400$
Conv3_00 (Weights)	$25 \cdot 324 + 18 = 8118$	-
Conv3_00 (Activations)	$*18 \cdot 25 \cdot 18 = 8100$	$25 \cdot 324 \cdot 18 \cdot 25 = 3645000$
Conv3_01 (Weights)	$25 \cdot 324 + 18 = 8118$	-
Conv3_01 (Activations)	$*18 \cdot 25 \cdot 18 = 8100$	$25 \cdot 324 \cdot 18 \cdot 25 = 3645000$
Pool3 (Weights)	-	-
Pool3 (Activations)	$*9 \cdot 12 \cdot 18 = 2016$	$2 \cdot 2 \cdot 18 \cdot 25 = 1800$
FC Classifier (Weights)	$9 \cdot 12 \cdot 18 \cdot 21 + 21 = 40845$	-
FC Classifier (Activations)	$*21$	$9 \cdot 12 \cdot 18 \cdot 21 = 40824$
Total	348303	100729420

Table 33: Memory footprint and computational demand of the best mse loss network

n conv blocks	filters	dil. rate	avg. pooling	Conv kernel dim	avg. IOU	Conv kernel dim	avg. IOU
3	12	1	False	5x5	0.05	7x7	0.08
3	12	1	True	5x5	0.01	7x7	0.02
3	16	1	False	5x5	0.07	7x7	0.10
3	16	1	True	5x5	0.01	7x7	0.01
3	20	1	False	5x5	0.11	7x7	0.12
3	20	1	True	5x5	0.01	7x7	0.01
4	12	1	False	5x5	0.07	7x7	0.09
4	12	1	True	5x5	0.01	7x7	0.01
4	16	1	False	5x5	0.06	7x7	0.07
4	16	1	True	5x5	0.02	7x7	0.01
4	20	1	False	5x5	0.05	7x7	0.12
4	20	1	True	5x5	0.02	7x7	0.03
5	12	1	False	5x5	0.07	7x7	0.04
5	12	1	True	5x5	0.01	7x7	0.02
5	16	1	False	5x5	0.04	7x7	0.11
5	16	1	True	5x5	0.02	7x7	0.03
5	20	1	False	5x5	0.10	7x7	0.09
5	20	1	True	5x5	0.02	7x7	0.04
6	12	1	False	5x5	0.02	7x7	0.01
6	12	1	True	5x5	0.02	7x7	0.03
6	16	1	False	5x5	0.03	7x7	0.03
6	16	1	True	5x5	0.02	7x7	0.01
6	20	1	False	5x5	0.03	7x7	0.02
6	20	1	True	5x5	0.04	7x7	0.03

Table 34: Custom loss model grid search results 1

Conv kernel dim	filters	dil. rate	λ_{coord}	λ_{obj}	n conv blocks	avg. IOU	n conv blocks	avg. IOU
7x7	12	1	10	1	3	0.08	4	0.08
7x7	12	1	10	3	3	0.08	4	0.08
7x7	12	1	10	5	3	0.12	4	0.10
7x7	12	1	20	1	3	0.05	4	0.08
7x7	12	1	20	3	3	0.08	4	0.07
7x7	12	1	20	5	3	0.09	4	0.09
7x7	12	1	3	1	3	0.11	4	0.09
7x7	12	1	3	3	3	0.06	4	0.09
7x7	12	1	3	5	3	0.10	4	0.08
7x7	12	1	5	1	3	0.09	4	0.10
7x7	12	1	5	3	3	0.09	4	0.09
7x7	12	1	5	5	3	0.07	4	0.07
7x7	16	1	10	1	3	0.11	4	0.08
7x7	16	1	10	3	3	0.10	4	0.07
7x7	16	1	10	5	3	0.12	4	0.08
7x7	16	1	20	1	3	0.10	4	0.11
7x7	16	1	20	3	3	0.07	4	0.08
7x7	16	1	20	5	3	0.11	4	0.07
7x7	16	1	3	1	3	0.08	4	0.14
7x7	16	1	3	3	3	0.09	4	0.07
7x7	16	1	3	5	3	0.10	4	0.10
7x7	16	1	5	1	3	0.09	4	0.10
7x7	16	1	5	3	3	0.08	4	0.08
7x7	16	1	5	5	3	0.09	4	0.08
7x7	20	1	10	1	3	0.13	4	0.09
7x7	20	1	10	3	3	0.10	4	0.09
7x7	20	1	10	5	3	0.11	4	0.10
7x7	20	1	20	1	3	0.09	4	0.09
7x7	20	1	20	3	3	0.14	4	0.11
7x7	20	1	20	5	3	0.12	4	0.08
7x7	20	1	3	1	3	0.13	4	0.08
7x7	20	1	3	3	3	0.11	4	0.11
7x7	20	1	3	5	3	0.11	4	0.08
7x7	20	1	5	1	3	0.14	4	0.07
7x7	20	1	5	3	3	0.12	4	0.10
7x7	20	1	5	5	3	0.14	4	0.14

Table 35: Custom loss model grid search results 2 part 1

Conv kernel dim	filters	dil. rate	λ_{coord}	λ_{obj}	n conv blocks	avg. IOU	n conv blocks	avg. IOU
7x7	12	1	10	1	5	0.07	6	0.03
7x7	12	1	10	3	5	0.07	6	0.05
7x7	12	1	10	5	5	0.10	6	0.01
7x7	12	1	20	1	5	0.09	6	0.02
7x7	12	1	20	3	5	0.10	6	0.03
7x7	12	1	20	5	5	0.09	6	0.02
7x7	12	1	3	1	5	0.09	6	0.04
7x7	12	1	3	3	5	0.11	6	0.02
7x7	12	1	3	5	5	0.08	6	0.04
7x7	12	1	5	1	5	0.09	6	0.03
7x7	12	1	5	3	5	0.07	6	0.03
7x7	12	1	5	5	5	0.10	6	0.03
7x7	16	1	10	1	5	0.11	6	0.03
7x7	16	1	10	3	5	0.12	6	0.02
7x7	16	1	10	5	5	0.10	6	0.03
7x7	16	1	20	1	5	0.09	6	0.04
7x7	16	1	20	3	5	0.08	6	0.01
7x7	16	1	20	5	5	0.09	6	0.04
7x7	16	1	3	1	5	0.09	6	0.06
7x7	16	1	3	3	5	0.11	6	0.04
7x7	16	1	3	5	5	0.12	6	0.06
7x7	16	1	5	1	5	0.09	6	0.04
7x7	16	1	5	3	5	0.09	6	0.04
7x7	16	1	5	5	5	0.09	6	0.04
7x7	20	1	10	1	5	0.06	6	0.04
7x7	20	1	10	3	5	0.15	6	0.04
7x7	20	1	10	5	5	0.11	6	0.06
7x7	20	1	20	1	5	0.09	6	0.03
7x7	20	1	20	3	5	0.10	6	0.06
7x7	20	1	20	5	5	0.11	6	0.04
7x7	20	1	3	1	5	0.09	6	0.03
7x7	20	1	3	3	5	0.13	6	0.08
7x7	20	1	3	5	5	0.09	6	0.04
7x7	20	1	5	1	5	0.10	6	0.05
7x7	20	1	5	3	5	0.12	6	0.06
7x7	20	1	5	5	5	0.09	6	0.05

Table 36: Custom loss model grid search results 2 part 2

	Memory footprint (Bytes)	Number of operations
Input	$*149 \cdot 201 \cdot 1 = 29949$	-
Pool0 (Weights)	-	-
Input-Pool0 (Activations)	$*74 \cdot 100 \cdot 1 = 7437$	$2 \cdot 2 \cdot 149 \cdot 201 = 119796$
Conv1_00 (Weights)	$49 \cdot 20 + 20 = 1000$	-
Conv1_00 (Activations)	$74 \cdot 100 \cdot 20 = 148000$	$49 \cdot 20 \cdot 74 \cdot 100 = 7252000$
Conv1_01 (Weights)	$49 \cdot 400 + 20 = 19620$	-
Conv1_01 (Activations)	$74 \cdot 100 \cdot 20 = 148000$	$49 \cdot 400 \cdot 74 \cdot 100 = 145040000$
Pool1 (Weights)	-	-
Pool1 (Activations)	$*37 \cdot 50 \cdot 20 = 37000$	$2 \cdot 2 \cdot 74 \cdot 100 = 29600$
Conv2_00 (Weights)	$49 \cdot 400 + 20 = 19620$	-
Conv2_00 (Activations)	$*37 \cdot 50 \cdot 20 = 37000$	$49 \cdot 400 \cdot 37 \cdot 50 = 36260000$
Conv2_01 (Weights)	$49 \cdot 400 + 20 = 19620$	-
Conv2_01 (Activations)	$*37 \cdot 50 \cdot 20 = 37000$	$49 \cdot 400 \cdot 37 \cdot 50 = 36260000$
Pool2 (Weights)	-	-
Pool2 (Activations)	$*18 \cdot 25 \cdot 20 = 9000$	$2 \cdot 2 \cdot 37 \cdot 50 = 7400$
Conv3_00 (Weights)	$49 \cdot 400 + 20 = 19620$	-
Conv3_00 (Activations)	$*18 \cdot 25 \cdot 20 = 9000$	$49 \cdot 400 \cdot 18 \cdot 25 = 8820000$
Conv3_01 (Weights)	$49 \cdot 400 + 20 = 19620$	-
Conv3_01 (Activations)	$*18 \cdot 25 \cdot 20 = 9000$	$49 \cdot 400 \cdot 18 \cdot 25 = 8820000$
Pool3 (Weights)	-	-
Pool3 (Activations)	$*9 \cdot 12 \cdot 20 = 2240$	$2 \cdot 2 \cdot 18 \cdot 25 = 1800$
Conv4_00 (Weights)	$49 \cdot 400 + 20 = 19620$	-
Conv4_00 (Activations)	$*9 \cdot 12 \cdot 20 = 2160$	$49 \cdot 400 \cdot 9 \cdot 12 = 2116800$
Conv4_01 (Weights)	$49 \cdot 400 + 20 = 19620$	-
Conv4_01 (Activations)	$*9 \cdot 12 \cdot 20 = 2160$	$49 \cdot 400 \cdot 9 \cdot 12 = 2116800$
Pool4 (Weights)	-	-
Pool4 (Activations)	$*4 \cdot 6 \cdot 20 = 480$	$2 \cdot 2 \cdot 9 \cdot 12 = 432$
Conv5_00 (Weights)	$49 \cdot 400 + 20 = 19620$	-
Conv5_00 (Activations)	$*4 \cdot 6 \cdot 20 = 480$	$49 \cdot 400 \cdot 4 \cdot 6 = 470400$
Conv5_01 (Weights)	$49 \cdot 400 + 20 = 19620$	-
Conv5_01 (Activations)	$*4 \cdot 6 \cdot 20 = 480$	$49 \cdot 400 \cdot 4 \cdot 6 = 470400$
Pool5 (Weights)	-	-
Pool5 (Activations)	$*2 \cdot 3 \cdot 20 = 120$	$2 \cdot 2 \cdot 4 \cdot 6 = 96$
FC Classifier (Weights)	$2 \cdot 3 \cdot 20 \cdot 21 + 21 = 2541$	-
FC Classifier (Activations)	$*21$	$2 \cdot 3 \cdot 20 \cdot 21 = 2520$
Total	476121	247788044

Table 37: Memory footprint and computational demand of the best custom loss network

References

[1] MLee. Visual guide to the confusion matrix, 2021. URL https://towardsdatascience.com/visual-guide-to-the-confusion-matrix-bb63730c8eba.

[2] Massimo Pavan. Tiny machine learning for UWB-radar based subject recognition in cars. Master's thesis, Politecnico di Milano, 2022. URL https://www.politesi.polimi.it/handle/10589/179542.

[3] Massimo Pavan, Armando Clatabiano, and Manuel Roveri. Tinyml for uwb-radar based presence detection. *Proceedings of WCCI 2022, IEEE*, page 5, Jul. 2022.

[4] Pete Warden and Daniel Situnayake. *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-low-power Microcontrollers*. O'Reilly, 2020. ISBN 978-1-4920-5204-3. Google-Books-ID: sB3mxQEACAAJ.

[5] James D. Taylor, editor. *Introduction to Ultra-Wideband Radar Systems*. CRC Press, 1995. ISBN 978-1-00-306811-2. doi: 10.1201/9781003068112.

[6] Andrej Karpathy Fei-Fei Li and Justin Johnson. Tiny ImageNet. Technical report, Stanford University, 2017. URL https://kaggle.com/competitions/tiny-imagenet.

[7] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. pages 779–788, 2016. URL https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Redmon_You_Only_Look_CVPR_2016_paper.html.

[8] Youngwook Kim and Taesup Moon. Human detection and activity classification based on micro-doppler signatures using deep convolutional neural networks. *IEEE Geoscience and Remote Sensing Letters*, 13(1):8–12, 2016. doi: 10.1109/LGRS.2015.2491329.

[9] Taehyeong Ha and Jeongtae Kim. Detection and localization of multiple human targets based on respiration measured by IR-UWB radars. In *2019 IEEE SENSORS*, pages 1–4, 2019. doi: 10.1109/SENSORS43011.2019.8956687. ISSN: 2168-9229.

[10] Fulai Liang, Fugui Qi, Qiang An, Hao Lv, Fuming Chen, Zhao Li, and Jianqi Wang. Detection of multiple stationary humans using UWB MIMO radar. *Sensors (Basel, Switzerland)*, 16(11): E1922, 2016. ISSN 1424-8220. doi: 10.3390/s16111922.

[11] Xinyu Li, Yuan He, and Xiaojun Jing. A survey of deep learning-based human activity recognition in radar. *Remote Sensing*, 11(9):1068, 2019. ISSN 2072-4292. doi: 10.3390/rs11091068. URL https://www.mdpi.com/2072-4292/11/9/1068. Number: 9 Publisher: Multidisciplinary Digital Publishing Institute.

[12] Jinhee Park, Rios Jesus Javier, Taesup Moon, and Youngwook Kim. Micro-doppler based classification of human aquatic activities via transfer learning of convolutional neural networks. *Sensors*, 16(12):1990, 2016. ISSN 1424-8220. doi: 10.3390/s16121990. URL https://www.mdpi.com/1424-8220/16/12/1990. Number: 12 Publisher: Multidisciplinary Digital Publishing Institute.

[13] Yue Lang, Chunping Hou, Yang Yang, Danyang Huang, and Yuan He. Convolutional neural network for human micro-doppler classification. In *European Microwave Conference*, 2017.

[14] Yuming Shao, Sai Guo, Lin Sun, and Weidong Chen. Human motion classification based on range information with deep convolutional neural network. In *2017 4th International Conference on Information Science and Control Engineering (ICISCE)*, pages 1519–1523, 2017. doi: 10.1109/ICISCE.2017.317.

[15] Don Koks. How to create and manipulate radar range–doppler plots. page 95, 2014.

[16] Runhan Bao and Zhaocheng Yang. CNN-based regional people counting algorithm exploiting multi-scale range-time maps with an IR-UWB radar | IEEE journals & magazine | IEEE xplore. *IEEE Sensors Journal*, 2022. URL https://ieeexplore.ieee.org/abstract/document/9398946?casa_token=QCH9Du5S1a4AAAAA:LVI-WtqX0lcO4htEZ2z0yiKUspNOJu_DCrtxRZ8Gk1EB_uGp7gZwiUhTvgjuNg68EpJZxOe7pQ.

[17] Partha Pratim Ray. A review on TinyML: State-of-the-art and prospects. 34(4):1595–1623, 2022. ISSN 1319-1578. doi: 10.1016/j.jksuci.2021.11.019. URL https://www.sciencedirect.com/science/article/pii/S1319157821003335.

[18] Qianyun Lu and Boris Murmann. Improving the energy efficiency and robustness of tinyML computer vision using log-gradient input images, 2022. URL https://cms.tinyml.org/wp-content/uploads/talks2022/2203.02571.pdf.

[19] Alexander Wong, Mahmoud Famouri, and Mohammad Javad Shafiee. Attendnets: Tiny deep image recognition neural networks for the edge via visual attention condensers, 2020. URL https://www.researchgate.net/publication/344436595_AttendNets_Tiny_Deep_Image_Recognition_Neural_Networks_for_the_Edge_via_Visual_Attention_Condensers.

[20] Mat Kelcey Louis Moreau. Announcing FOMO (faster objects, more objects), 2022. URL https://www.edgeimpulse.com/blog/announcing-fomo-faster-objects-more-objects.

[21] Alexandre Bayen Qingkai Kong, Timmy Siauw. *Fast Fourier Transform (FFT) — Python Numerical Methods*. Academic Press, 2020. URL https://pythonnumericalmethods.berkeley.edu/notebooks/chapter24.03-Fast-Fourier-Transform.html.

[22] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. page 5, 1965.

[23] Sumit Saha. A comprehensive guide to convolutional neural networks — the ELI5 way, 2018. URL https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[24] Jason Brownlee. A gentle introduction to k-fold cross-validation, 2018. URL https://machinelearningmastery.com/k-fold-cross-validation/.

[25] Rajkumar Lakshmanamoorthy. Python code for evaluation metrics in ML/AI for classification problems, 2021. URL https://analyticsindiamag.com/evaluation-metrics-in-ml-ai-for-classification-problems-wpython-code/.

[26] Jason Brownlee. What is a confusion matrix in machine learning, 2016. URL https://machinelearningmastery.com/confusion-matrix-machine-learning/.

[27] Adrian Rosebrock. Intersection over union (IoU) for object detection, 2016. URL https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.

[28] Zach. Z-score normalization: Definition & examples, 2021. URL https://www.statology.org/z-score-normalization/.

[29] Xin Tan, Chun Tao, Tongwei Ren, Jinhui Tang, and Gangshan Wu. Crowd counting via multi-layer regression, 2019. URL https://doi.org/10.1145/3343031.3350914.

[30] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications, 2017. URL https://dblp.org/rec/journals/corr/HowardZCKWWAA17.html.

[31] Luis Gonzalez Navarro. Design of a Tiny Machine Learning system for UWB radar based multi target detection, 9 2022.