Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Detection and Classification of Small Concealed Objects using mmWave FMCW Radars.

Author:
Kheil Ziad

Supervisor:
Knottenbelt William

Submitted in partial fulfillment of the requirements for the MSc degree in Machine Learning & AI of Imperial College London

September 2, 2021

Abstract

Concealed Weapon Detection (CWD) is a longstanding problem with global repercussions, all existing solutions leverage some, or a combination of sensors which span across the frequency spectrum to detect lethal weapons. Most of these usually require some form of handheld device, or portal based scanning, and consequently their range of detection only encompass a few centimeters or roughly a meter.

Recently, research has been more inclined to tackle the issue from a more functional and handy standpoint, that is: detecting a threat from a distance or amongst a larger crowd.

Through the lens of poaching detection and prevention, we attempt to study the feasibility of a mid-range, low consumption solution to detect and eventually classify concealed weapons using Millimetre Wave sensors. In particular the goal is to show the feasibility of object recognition and classification and study the reliability of such a system on a longer range. More specifically this is done here, on small everyday objects: knives and spoons.

Acknowledgments

I would like to start things by thanking the various people who helped make this project successful and guided me at times of uncertainty.

First of all to my Supervisor Prof. W. Knottenbelt, for suggesting the projects, and suggesting ideas and general directions throughout my work at times were I was unsure.

I am also grateful for the frequent meetings with C. Charalmbous and F. Mouridsen from Stofl Ltd. who also were never short of amazing ideas, and generally very supportive of the work presented here.

Contents

Chapter 1

Introduction

> *" People shot you instantly if they thought you were going to shoot them. But if you were unarmed, they often stopped to talk. "*
>
> — Terry Pratchett, *The Last Continent*

Wildlife trafficking was recently estimated to belong in the top 5 highest sources of illicit cash income, if we take into account fishing and logging, it ranks 2^{nd} , right after drug trafficking and before human trafficking. Estimating such a market is complicated due to its illicit nature, and the complications in properly defining it across borders, but some estimates cite roughly a $ 19 billion dollar industry (ranging from $ 5 to 23 billion) [1, 2]. With such financial incentives, poaching persists in thriving throughout the world. In January of 2021, an incident where 6 rangers where assaulted and killed by poachers in the Democratic Republic of Congo [3]. This tragedy reminds us how dangerous such activities can be, and when we learn that nearly 2 rangers a week are killed while protecting wildlife [4], it is easy to understand why a solution must be found, not only to detect intruders, but especially to obtain early warning of what type of weapons rangers might be facing. Currently this is done using highly trained dogs, which not only are complicated to obtain and train, but also not always available. This project attempts to leverage FMCW Mmwave based radars and Deep Learning to help tackle such issues with some potential, broader applications for Concealed Weapon Detection.

Ideally, our solution would have a low false positive rate, and function on a reasonable range such that it can be deployed on a vast terrain. The idea being to detect intruders over a large terrain, and acquire information about their general direction, number of individuals and weapons they might be carrying to assist rangers.

In the scope of our project, the focus was aimed at demonstrating the capacity of learning based approaches on data issued from Mmwave sensors with the aim to classify concealed objects from a certain range.

Chapter 2

Background

2.1 The poaching industry and its complications

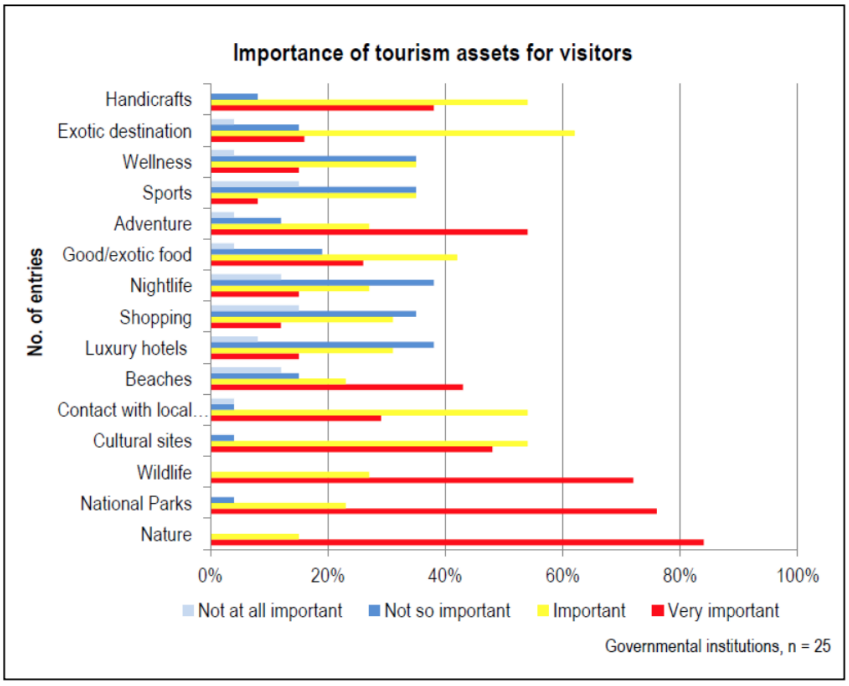
As previously stated, as with most illicit activities, estimating poaching related revenues is unfeasible, on the other hand the ecological toll of such practices is easily visible. Many agencies keep track of wildlife population estimates, the South African department of Forestry, Fisheries and the Environment, declared that in 2020 roughly one rhino was killed every 22 hours, and this represents a 33% decline compared to 2019. But this decline is thought to be directly linked to the Covid pandemic and travel restrictions, and is expected to rise. It is noteworthy that since 2012, elephant poaching rates have surpassed their natural growth, or replacement rates [5, 6].

Other than the environmental and ethical toll posed by the poaching industry, decline in animal population immediately translates to decrease in GDP for many regions. With roughly 80% of Africa's tourists being attracted by wildlife, a shift from "Big Five" to "Big Four" could have unfathomable repercussions on the economy. In fact it is estimated that a 1% decrease in elephant population equates to $ 4 billion lost annually (5% of the annual tourism income) [5, 7]. The impact of wildlife conservation on tourism in Africa is quantified further in figure 2.1 which was published by the UNWTO [8]. Keep in mind that over the course of 7 years, we have seen a 30% decline in African Savannah elephants' population.

Finally, as we will see later, although sometimes poachers do not pose a threat to rangers, often they are heavily armed, and well trained. Without sufficient information about the situation, rangers who intervene are constantly living on the brink, with their lives on the line.

To counteract this, plenty is already being done, albeit not enough. Such measures range from social awareness such as debunking the supposed health benefits of tusks, to concrete preventive measures which are accomplished by rangers in several natural reserves. And the numbers show that this has helped reverse the curve, since the peak in 2006, the numbers of rhinos poached has steadily declined, but remain unsustainable for the population.

In order to help rangers with their task, this work aims at developing a system to proactively detect and classify armed intrusions on reserve grounds. This idea is



Source: UNWTO, *Towards Measuring the Economic Value of Wildlife Watching Tourism in Africa*.

Figure 2.1: Economic Value of Wildlife in Africa. Published by [8]

less intrusive, and less risky [9] for the wildlife than directly tracking animals, as this information, if not handled correctly could also benefit poachers.

2.2 Weapon Detection

Weapon detection is a relatively well studied subject and usually referred to as Concealed Weapon Detection (CWD). Numerous systems based on a vast array of different sensors have been tested and are actively used. Each presents it's own advantages and drawbacks, and choosing one highly depends on the use case's specifications and constraints. Here are a few of the most common ones from the literature:

2.2.1 Metal Detectors

The most widespread, and known CWD method is the metal detector. The general term ”metal detector” is usually used to refer to systems which make use of some metals' ferromagnetic properties to detect them.

One way to achieve this, is through gradiometers, which are themselves composed of several magnetometers [10]. The idea being that we can measure the local difference, or distortion, of Earth's magnetic field due to the movement of a ferromagnetic metallic object. We know that when such an object is in movement and subject to a magnetic field (here Earth's magnetic field), it in turn produces it's own magnetic field, due to mutual inductance. This will cause a small local change in the

measured magnetic field which is measured and used as a signature to characterize an object, then discriminate as dangerous or not using a previously established database. Such systems are usually used as handheld devices which need to be swiped across the person we are searching, or portals through which the subject needs to walk through.

"Electromagnetic inductive techniques" are techniques which rely on the inductive properties of metallic objects. Other devices, instead of leveraging Earth's magnetic field and the movement of the object to induce a magnetic field, generate their own magnetic fields, which vary through time. Inductive laws also state that a coil subject to such a varying field would also generate their own field which again can be measured and used as a signature.

Metal Detectors present a few drawbacks, in particular, since the human body incorporates a small conductivity as well, depending on the object's size and inductive properties the accuracy of detection can be affected [10]. Furthermore, this method only functions when dealing with metallic objects, and typically works on limited ranges.'

2.2.2 Acoustic and ultrasonic detection

Today, some hand held CWD devices make use of acoustic or ultrasound waves coupled with signal processing techniques to identify signatures of the resonant response of an object when exposed to such waves.

Put simply, in function of the type of material, size and geometry of an object, acoustic waves will bounce off objects in different ways, which we can later use to infer information about the object itself. Typically, "soft" objects will absorb acoustic waves, while "hard" ones present higher reflectivity [10].

In practice, acoustic detectors are rarely used since they present several drawbacks including lofty false alarm rates: since they generally react well to hard objects, some issues arise in differentiating these objects. Also the signature of a plain human body, and a human body with a concealed weapon shows hardly any difference. Researchers have tried to make this system more robust, but this implied hard constraints such as the subject being positioned a specific way during search.

2.2.3 Infrared Detection

Albeit often being associated with nigh or low-light condition vision, Infrared devices are also commonly used for CWD. Since no material being at 0°K, all objects emit radiation and the peak wavelength of this radiation depends on the temperature of the object. This also being true for the human body, devices are set with a peak sensitivity for the typical wavelength emitted by the human body (a few μm). Since clothing will re-emit the heat absorbed from the human body, we can get a general idea of concealed object under clothing through the form of an Infrared image, where heat blurs represent hidden objects [10], an example can be seen in Figure 2.2.

But we should note that if concealed objects approach human body temperature, detection becomes impossible. Also, clothing can greatly affect detection, usually tight clothes present no challenge, but looser clothing induce blurry heat emission, and so disturb detection.

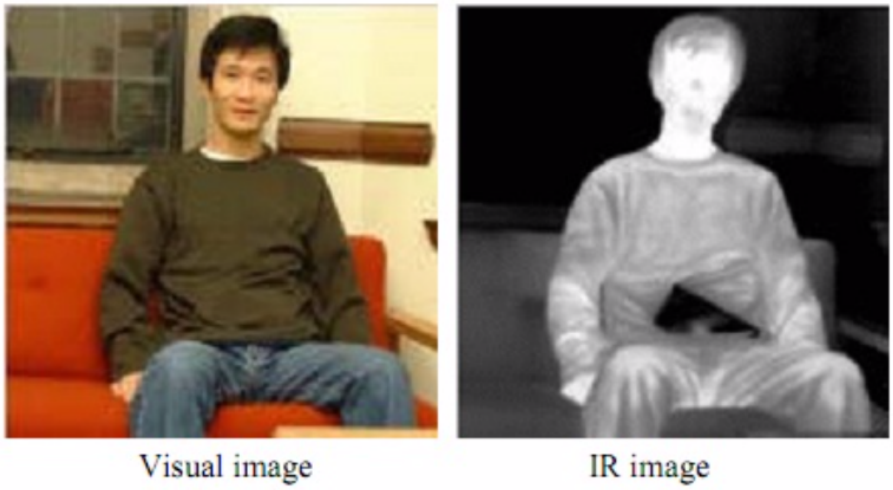


Figure 2.2: Infrared Imaging output example. Image from [11]

2.2.4 THz Imaging

Making use of THz frequency waves has gained popularity, not only is it a reliable method, but it can also detect a vast array of objects ranging from traditional weapons, to explosives and bio-hazards.

The idea is to leverage the big differences in absorption and reflection of different materials when exposed to different wavelengths of THz waves, this is known as spectroscopy. For example, meanwhile metals will completely reflect such waves, clothing, paper and plastic will all be penetrated quite easily. Ceramics will partially reflect the waves, and the human body, due to its high concentration in water will almost entirely absorb these waves. Note that they are not harmful, they will simply dissipate as heat in the first 100 microns of our tissues [10].

Granting THz imaging seems to pick up the vastest array of potential threats, we should point out that the atmosphere immensely attenuates signals, and rain and fog even more. So there is a quality/range trade off. More so, generating THz waves is very tricky [12] , most systems have a very low energy conversion rate. In other words, to generate even a weak THz signal, much power is needed.

2.2.5 Mmwave Sensors

Two types of Millimeter-Wave screening methods exist, passive ones where we rely on the naturally emitted radiations and read them. What is more interesting is the active method, where we generate waves with known properties, and analyze the

reflected signal to infer what we are looking for. MMwave sensors require very low radiation power, and so are also low-consumption, and tend to work equally well in low-light, rain of foggy conditions due to the nature of the waves they generate: these typically range from 30 to 300 GHz.

2.3 On Poaching Techniques

As seen previously, numerous techniques exist allowing for detection and classification of weapons with different advantages and drawbacks. The question that arises is which one to use? The answer lies in what we are trying to accomplish: detect, characterize and eventually identify poachers. Poachers typically operate in natural reserves, which span over several hundred thousand hectares, to millions. Furthermore, our results must be accurate no matter the conditions, sunny or dark, scorching hot or pouring cats and dogs. Note that, for deployment in reserves like the Mohololoholo wildlife reserve which span over a few dozen thousand hectares, our solution must also ideally be low consumption, and range over at least a few meters to realistically cover a substantial area.

Finally the biggest factor to consider are what types of weapons we are trying to detect. A study [6] on poachers' weapons and methods shows that two poaching "styles" exist. Large-scale poaching which is characterized by a short-time frame during which a large population of wildlife is targeted strenuously, can be contrasted with small-scale poaching during which only a limited amount, or even an individual animal is targeted, over longer periods of time. Both types are equally lethal for wildlife.

In terms of weapons and techniques, both styles usually converge to a divergence of weapons, which range from military grade automatic firearms to traditional weapons. These can also be classified in two large types:

- Firearms: the most commonly used poaching firearms can also be categorized broadly as hunting rifles, military-style automatic firearms and shotguns. Although we should note that information on seized weapons are not consistently collected or analyzed, none really prevail over others, in general each have their own uses,advantages and disadvantages. Hunting rifles with .375 or .458 calibres have a larger range range, and can kill large game with one shot, meanwhile Kalashnikov-pattern rifles use smaller chambers (typically 7.62 $\times$ 39 mm) which gives them smaller ranges and penetrating power, but are usually less expensive than hunting rifles. In groups, poachers would typically use hunting rifles for the game, and automatic weapons to ensure a protective perimeter for the designated shooter [6].

- Traditional and Craft weapons: although traditional weapons take more time to kill large species, which in turn requires patience, and usually tracking the animal for several hours, all the while evading detection, they have the advantage of not attracting rangers' attention due to gunshot sounds. They are usually comprised of poisoned tipped spears and arrows or even simply poison

(e.g Zimbabwe 2013, mass poisoning was responsible for the loss of hundreds of elephants) [6]. Craft firearms also exist, although in sparser quantities.

In terms of which types are most prevalent, no clear conclusions can really be drawn, for example in 2012 in Kenya's North Rift Region , 85% of killed elephants presented gunshot wounds, yet this dropped to 34% in the Coast Province [6]. Also, groups can range from 3 poachers (eventually even lone ones), to over a dozen,Ṫheir composition also presents high variance, with actors ranging from random civilians (usually bush meat poaching) to rogue military units, non-governmental armed groups, and even sometimes national armed forces.

If we head back to our potential detection mechanisms, we realize that first of all in light of the vast array of weapons used, and the non-existent prevalence of certain weapons, we can't solely rely on magnetic based detection of weapons, or even simply weapons detection for that matter, in some instances simply detecting human presence must suffice. Also, conditions are such that acoustic and infrared detection would not work properly, and THz imaging requires a power consumption which we don't have access to. This leaves us with Mmwave sensors as a viable candidate.

2.4 TI IWR1443 FMCW Radar

In light of the different practical aspects of poaching we had to consider, we decided to go forward with the Mmwave sensor for this use case. The chosen sensor was the Texas Instruments IWR1443BOOST Evaluation board. This section discusses more in depth aspects of radar theory.

FMCW (frequency modulated continuous wave) radars emit electromagnetic waves called *chirps*, and measure the resulting reflection of these chips on objects in their field of view. Chirps are signals such that their frequency increases linearly with time (see figure 2.3). The following sections explain how a FMCW can provide a reading for the objects' range, velocity and angle (radial distance).

Throughout this section, we will use the following notation to describe the emitted chirp:

- f_s, f_e : start and end frequency of the chirp.
- S : slope of the chirp's linear frequency increase
- T_c : total chirp time

2.4.1 Measuring Range

Suppose we have an object in the field of view of the radar, at a distance d, when a chirp signal x_1 is emitted, it reaches the object and bounces back, all in all the chirp is received by the radar at time $\tau = \frac{2d}{c}$ (where c is the speed of light). When the

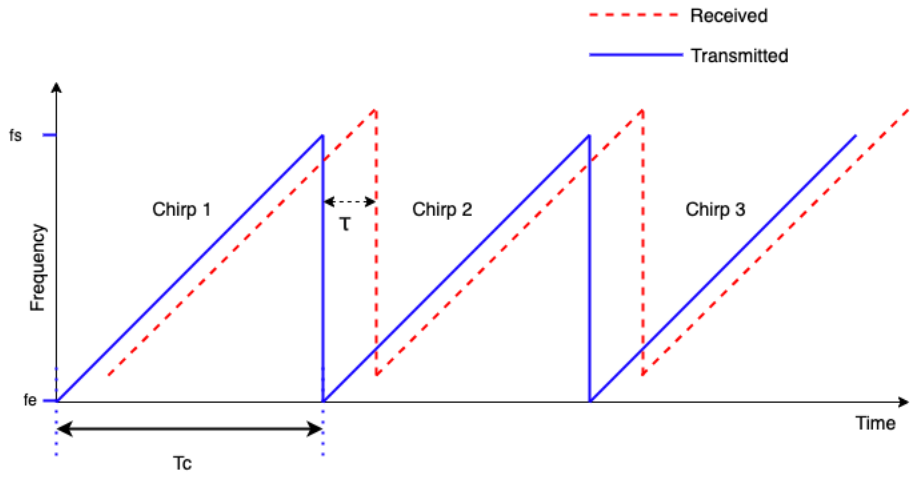


Figure 2.3: Visualisation of Chirps and reflected signal

chirp is received by the radar it constitutes a new signal x_2. Both these signals are fed into a *mixer* which outputs:

$$f_{mixer}(x_1, x_2) = f_{mixer}(\sin(\omega_t t + \phi_t), \sin(\omega_r t + \phi_r))$$

$$= \sin((\omega_t - \omega_r)t + (\phi_t - \phi_r))$$

- If we suppose there is a single object, the received signal is the same as the emitted, but shifted in time by a delay τ, thus the emitted and reflected frequencies will be identically ramping by S, and the output of the mixer has a constant frequency $f_c = S\tau$, and given the definition of τ, we have :

$$\boxed{d = \frac{cf_c}{2S}} \tag{2.1}$$

- On the other hand, if there are several objects, each will reflect the signal at a different time, so for a single chirp, the radar will receive different signals, and the output of the mixer will be comprised of several frequencies. In that case, by computing the range-FFT we obtain the different distances of each object [13].

  Finally, note that the range resolution d_{res} and maximum unambiguous range d_{max} depend on the chirp configuration, and the analog to digital converter (ADC) as follows [14]:

$$d_{res} = \frac{c}{2B} \tag{2.2}$$

$$d_{max} = \frac{F_s c}{2S} \tag{2.3}$$

  (F_s : ADC sampling rate).

2.4.2 Measuring Angle

The estimation of Angle of Arrival (AoA) requires at least 2 receiving antennas [14], let these be RX1 and RX2, as per 2.4, the chirp emitted by the object arrives at RX2 with a time delay of $\frac{\Delta d}{c}$, thus the phase difference $\Delta\phi$ between the two received signals for the difference in distance to the object Δd will be :

$$\Delta\phi = \frac{2\pi\Delta d}{\lambda}$$

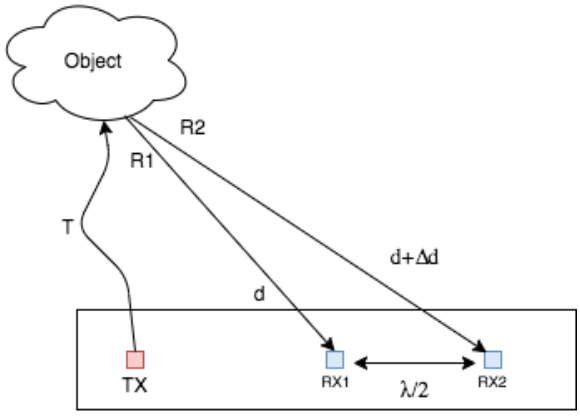


Figure 2.4: Radar representation. transmitting antenna in red, receiving ones in blue. Made in DrawIO.

Furthermore, if we consider the distance between TX1 and RX2 negligible compared to the distance of the object, we can make the reasonable approximation that the reflected signals arrive approximately parallel to one another at RX1 and RX2, thus geometrically (see Figure 2.5) :

$$\Delta d = l \sin(\theta)$$

and by combining these two equations, we get a formula for estimating the AoA :

$$\boxed{\theta = \sin^{-1}\left(\frac{\lambda\Delta\phi}{2\pi l}\right)} \tag{2.4}$$

2.4.3 Measuring Velocity

Finally, measuring the velocity of an object requires 2 chirps [14], and relies on the phase delay. By looking at the mixer output formula stated previously, the initial phase ϕ_0 of the mixer output signal is the difference of the initial phases of the mixer inputs. In other words, if an object is initially at a distance d when it is reached by the first chirp, the mixer output will be sinusoidal: $A\sin(2\pi f_s t + \phi_0)$.

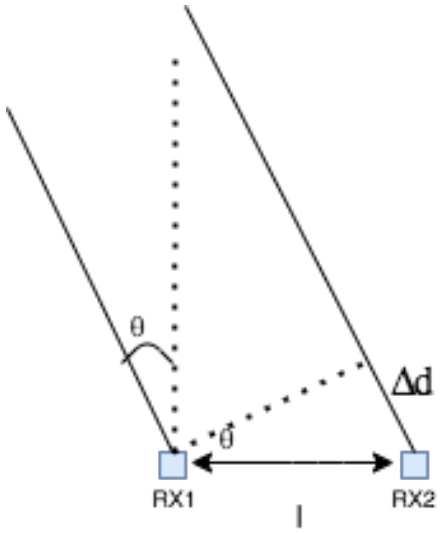


Figure 2.5: Visualisation of relations between the incoming reflected signals which can be considered parallel. Made in DrawIO

Now if we suppose we emit the second chirp delayed by T_c after the first one, and the object has moved by Δd, the round-trip delay of the signal has thus changed by $\Delta\tau$, from the mixer output formula, we can see that the phase difference will be $\Delta\phi = 2\pi f_s \Delta\tau$. The round-trip delay formula stated above ($\tau = \frac{2d}{c}$) allows us to rewrite this as : $\Delta\phi = \frac{4\pi\Delta d}{\lambda_s}$ (where λ_s is the starting wavelength of the chirp).

Note that this only works if at a given range, there is only one or no object. If that is not the case, we need several chirps to separate them. As was done for detecting several ranges previously, we apply a range-FFT to every chirp, this allows us to separate objects by their range. And we then apply a second FFT called Doppler-FFT which separates objects at the same range, by their velocity.

Finally, using $\Delta d = vT_c$ we are left with:

$$\boxed{v = \frac{\lambda_s \Delta\phi}{4\pi T_c}} \tag{2.5}$$

2.4.4 Radar Cube

The commonly accepted and widespread structure to store radar data is called the Radar Cube. First we construct a Data Matrix which contains the received signal reflected by objects. It is organised as follows:

- A chirp is subdivided into different time slots defined by the sampling rate. This is known as *Fast Time*.
- Each chirps contain another series of time steps called the *Slow Time*.

This yields a Data Matrix comprised of received IQ signals, with the dimensions being the Fast and Slow Times [15].

Since most FCMWs incorporate several receiving channels, the *Radar Cube* is defined as an extension of the Data Matrix obtained by stacking the Data Matrices obtained by each channel [15].

Figures 2.6 and 2.7 found in [15] are good illustrations of these structures in the case of L time bins, M chirps and N channels.

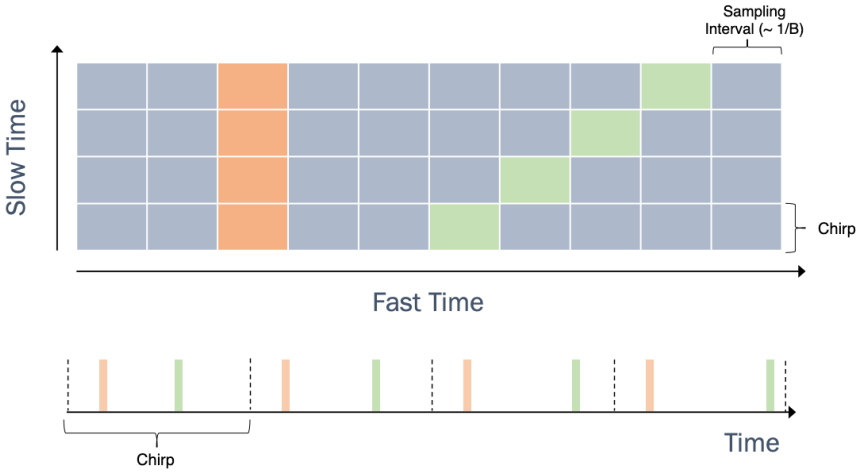


Figure 2.6: Visualisation of Data Matrix and chirp returns for a static target (orange) and a moving one (green). Figure from [15]

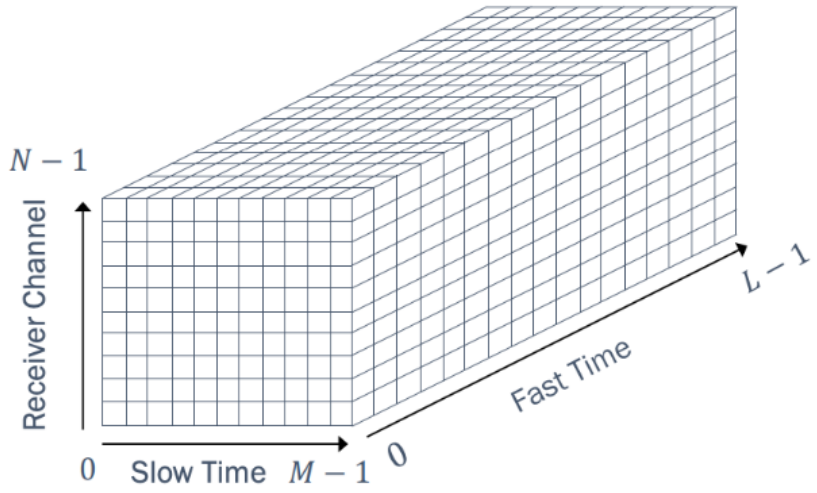


Figure 2.7: Visualisation of Radar Cube format for configuration of L time bins, M chirps and N channels. Figure from [15]

2.4.5 Configuration

The chosen sensor – IWR1443BOOST evaluation board [16] – operates on frequencies in the range of 76 to 81 GHz and incorporates 4 receivers and 3 transmitters.

In terms of raw outputs we have access to, there is a trade off between the large quantities of data that is generated and the output streaming speed. The evaluation board was not made to stream large quantities of data. In order to extract the full raw data, that is the *radar cube*, we would need the DCA1000 to increase the data transmission rate.

The radar cube represents the complex IQ signal received from each chirp on the receiver array. It is the generally accepted way to represent space-time processed radar data. But since the radar cube can be very large (depending on the configurations of our sensor, it is not possible to stream it out through the UART-to-USB connection of the evaluation board (UART speed limit $<$ 1 Mbps), instead we only have access to pre-processed parts of the data, and a certain number of other useful information [17] as described in the following section.

2.4.6 IWR1443BOOST Outputs

When the Data Cube is constructed, a series of on-chip processing functions are launched, these can be regulated through the sensor configuration file (section 3.1.3).

Globally, the radar cube is processed through 2 succesive FFTs, further details on this can be found in the TI user doyxgen files.

The 1st dimension FFT is launched during the chirps, while the 2nd is launched after the end of the chirps, and before the next ones (during the "Inter Frame Period").

- **1st Dimension FFT Processing** : The FFT of each row in the Fast Time dimension is computed, this allows to interpret the Fast and Slow Times respectively as *Range* and *Doppler* bins. Typically this is used to discriminate objects at the same range index but different velocities.

- **2nd Dimension FFT Processing** :
  - 1) If enabled, initiate Static Clutter Removal.
  - For Range-Doppler Heatmap : 2D-FFT across all chips (on the previous 1D-FFT), Then accumulate the samples across all receiving channels, this collapses the radar cube structure back to a matrix.
  - For Azimuth-Static heatmap: 2D-FFT across the range bins and take only 0th doppler value of the cube, which yields a matrix.
  - for Object Detection : CFAR detection and peak grouping.

These data processing steps leave us with the following data structures which can be transmitted through the UART interface:

- *Range Profile* : An array of profile points at 0th Doppler (stationary objects) for each range bin. Concretely, the array values are the average of log2 magnitudes of the received signal by the antennas.

- *Noise Floor Profile* : Same format as Range Profile, but represents the maximum Doppler bin, so objects at maximum speed. Since in general for a scene there are no objects at that speed, this profile represents the noise floor of the receiver.

- *Static Azimuth Heatmap* : Part of the radar cube, the Azimuth Static heatmap is the 2D FFT array in the range direction, but only at Doppler index 0 (so the stationary objects). It contains received I/Q signals from each antenna.

- *Range Doppler Heatmap* : Known as the detection matrix, it contains the log2 magnitudes, at each range and Doppler index combination.

- *List of Detected Objects* : Objects are detected using CFAR detection coupled with a peak grouping algorithm. Parameters such as the CFAR threshold can be uploaded with the configuration file beforehand. We obtain a list of detected objects comprised of the peak magnitude, and the Doppler and range indexes (which can then be translated to a speed and distance).

- *Stats information* : Holds various information about the measured frame, such as the CPU load usage, processing times etc...

2.5 Deep Learning

2.5.1 Learning theory

In order to compose a model which can predict the outcome given input data, we rely on Supervised Learning.

In such problems, given input data x where we desire a certain output label y, the problem can be written as :

$$f(x) = y$$

Where f is the function we want to learn. When our output labels y are finite integers, say $y \in \{0, 1\}$ for example, the problem is called a classification problem (in the previous example, Binary Classification).

In order to learn the function f we need a *training dataset* comprised of a series of input data along with the appropriate label. We then ask an algorithm to predict the output label of each input data. This output is compared to the correct label, his gives a loss value, which we can differentiate with respect to the inputs in order to adjust the model. This adjustment process can be the Stochastic Gradient Descent (SGD) whcih uses a few samples a batch Gradient Descent which uses a batch of samples.., Gradient Descents come in numerous possible algorithms. The goal of the SGD is to minimize the loss value, and the loss function is defined such as when it reaches it's minimal value, the model's predictions are "often" correct. An example of such a loss value is the Cross Entropy loss used for binary classification:

$$L_{cross-entropy}(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log 1 - \hat{y}$$

If the label to predict is $y = 0$, then we have $L_{cross-entropy}(y, \hat{y}) = -\log 1 - \hat{y}$ which is minimized for $\hat{y} = 0$, and if $y = 1$, then we have $L_{cross-entropy}(y, \hat{y}) = -\log \hat{y}$ minimal for $\hat{y} = 1$.

The models typically used for such predictions are called artificial Neural Networks. Neural Networks are composed of *perceptrons* which are essentially weight matrices and bias matrices which are multiplied by their input. They take input data passed as a vector, process them through multi-layer perceptrons and apply activation functions. This process is done several times and across several dimensions, til we reach the output layer which outputs the final prediction in the desired output dimension. A graphical representation of a neural network and a perceptron can be found in figures 2.8 and 2.9.

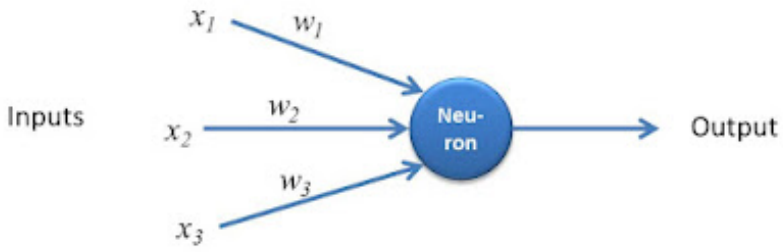


Figure 2.8: Perceptron representation. Figure from [18]

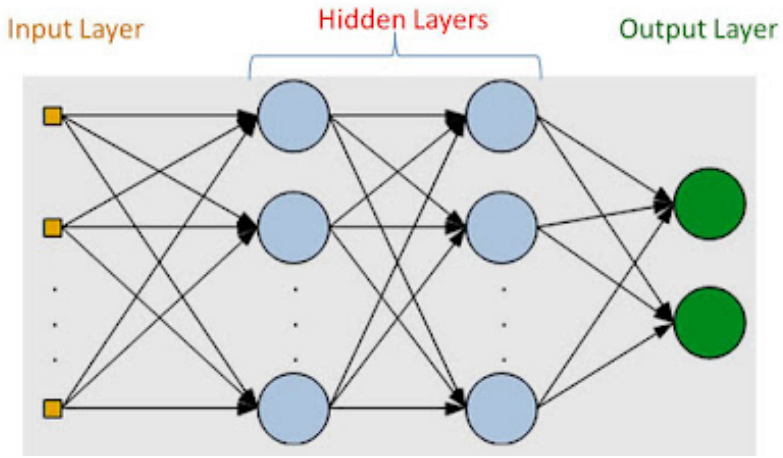


Figure 2.9: Multi Layer Perceptron (Neural Network) representation. Figure from [18]

2.5.2 CNNs

CNNS (Convolutionnal Neural Networks) are a particular class of Neural Networks which are more commonly used today, especially for deep learning on images. Instead of fully-connected and linear layers, the idea is to use a weight matrix composed of a small convolutional which we slide over the input matrix. An example of a CNN architecture and sliding windows is visible in figure 2.10.

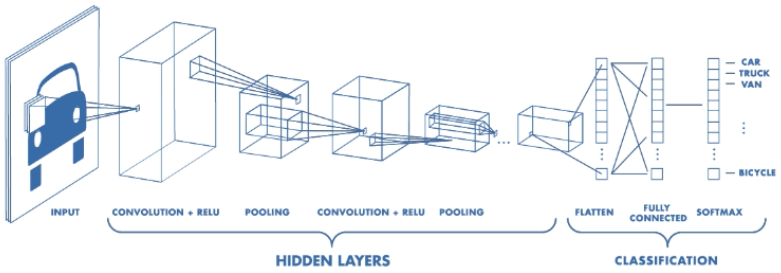


Figure 2.10: CNN representation. Figure from [19]

This allows for much faster processing times as we have much less operations to compute, and weights to learn, but works as well, and even better than the classic fully connected layers, it is usually applied on data such as images because of its invariance and approximate equivariance properties to certain transformations. The weights of the convolutions are still learned through gradient descent.

Research suggests that networks comprised of small but deep convolutional layers (deep here refers to the number of layers) outperform their counterparts with a few big convolutions.

CNNS exist in various and unlimited forms, some notable ones are LeNet[20], AlexNet[21], VGG [22] and ResNet [23]. But custom tailored CNNS also exist.

Also note that in practice CNNs are mostly used when the input data is an image (matrix), but 3D convolutions also exist and work in exactly the same fashion. A common practice is to employ a CNN architecture to transition from the input data to a denser, but more meaningful vector representations, and use this vector as a feature input to a fully connected network in order to predict the output label.

Optimizing

Once a Neural Network is designed and the layers are initialised, we iteratively try to fit the network to the data after each prediction (or batch of predictions). In order to do this, we require two other components: a loss function, and an optimization algorithm. As explained previously, the loss function enables us to quantify if the prediction matches the target, or how far off the ground truth it is. The optimization algorithm allows us to backpropagate the loss and update all of the weights of the model, in order to hopefully obtain better results on the next attempt. Thus the model is updated by alternating between evaluation on the data, and backpropagating errors measured with the loss function.

There exists many possible loss functions, as well as optimizers, for this project I chose the Cross-Entropy Loss which was presented above, and the Adam Optimizer. This will be discussed in furhter details in the rest of the report.

Adam

Adam stands for Adaptive Movement Estimation, dating back from 2015, it is frequently used for machine learning optimization. Like most optimizers, its core idea is the same as stochastic gradient descent. classical SGD makes use of a *learning rate* , which is the ”step size” to update the weights in the direction dictated by the optimization problem. In Adam, we keep a learning rate **for each** weight parameter instead of a single one for the entire network, and these learning rates are not fixed like in the classical SGD, but are adapted after each step based on the first and second order momentum of the gradient (mean and variance).

2.6 Deep Learning with Mmwave sensors

A fair amount of recent papers have leveraged Deep Learning techniques coupled with the outputs of Mmwave sensors to achieve astounding results. Usually, for such results, not the whole radar cube is needed, although it entirely depends on the application itself.

For instance, Gesture Recognition adopting Mmwave sensors showed promising results [24], which was later extended to to long range gesture recognition [25]. Another interesting application was that of indoor human tracking [26] which not only showed we could reliably detect a person, or a group of people, but also uniquely identify using information such as their gait. Note that uniquely identifying someone's gait could be extremely useful in reliably convicting a poacher once he/she has been arrested.

Other notable approaches include classification of objects from driving scenes, surface material classification and even high resolution imaging using Mmwave sensors [27, 28, 29, 30].

A good overview of recent learning based approaches with Mmwave sensors is available in table 2.1 inspired from [31], it depicts what sensor data was used, what they achieved and using what types of learning technique. The paper[31] as a whole does an excellent job at describing the most recent advances in the field.

2.7 Problem Statement

From the previous chapter, we can see some common denominators from previous Mmwave classification tasks: Most rely on movement(usually Doppler signatures) as features for the classification task.

Although some papers do classify static objects, or make use of heatmaps instead of Doppler Signatures for object recognition, these objects are usually fairly big (cars, bikes, street signs etc..).

This paper aims to introduce a novel problem in terms of millimeter-wave sensing: classifying small static objects. We also impose a limit in terms of data structure used. Having no access yet to the DCA10000, this project will rely solely on the data

Referenc	Data	Network Model	Task	Objects
O. Schumann et al.,[32]	Radar cube	CNN, RNN	Segmentation & Classification	Cars, buildings, pavements, poles, trees, and more
M. Sheeny et al.,[33]	Range profile	CNN	Detection & Classification	Bike, trolley, mannequin, cone, traffic sign, stuffed toy
K. Patel et al., [34]	Range-Azimuth	CNN, SVM and KNN	Classification	Barrier, motorcycle, baby carriage, bicycle, garbage container, car, and stop sign
Y. Kim and T.Moon [35]	Micro-Doppler signatures	CNN	Human discriminator & pace classification	Human, dog, horse, and car

Table 2.1: A few Mmwave and ML based applications. Adapted from [31]

classes described above, as we can not make use of the entire Radar Cube, and in particular, not compute Doppler Signatures.

With this in mind, we propose a method to detect and classify small objects – in this case a recognize handheld knife from a handheld spoon. This method is then evaluated through different test beds in order to determine if such a method is indicative of the potential for simple system to solve long distance, low-cost CWD.

Chapter 3

Contribution

3.1 Choice of Data and Configuration

3.1.1 Parsing Data

The output data we discussed in section 2.4.6 is streamed as a hexadecimal file representing output bytes. This must be parsed in order to obtain readable and usable data (strings, integers etc..). The first step of parsing this output data is to use the sensor configuration file (*.cfg*) in order to obtain necessary information such as Range, Doppler and Angular resolution, number of range and Doppler bins and which data outputs were selected. The code for this can be found on the github [1] and an overview of the profile.cfg file syntax in figure 3.1.

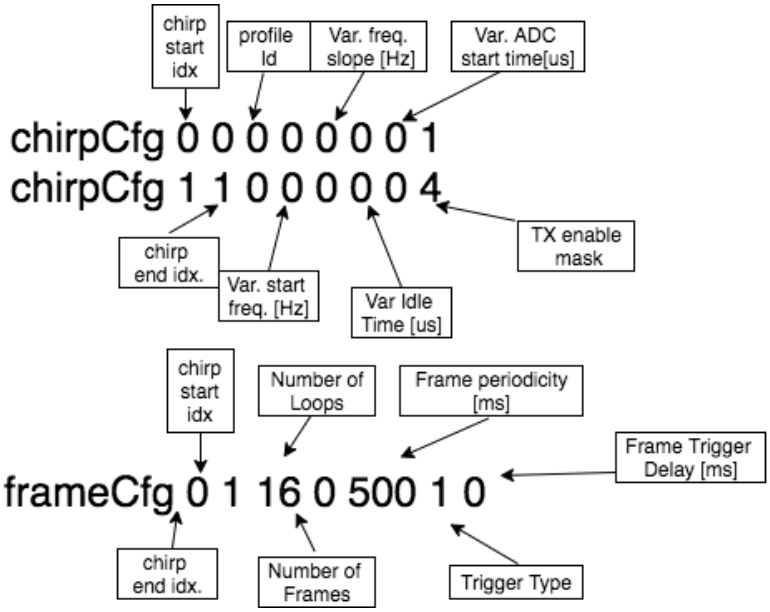


Figure 3.1: Chirp and Frame configuration parameters to be included in the sensor profile configuration file

[1] `https://github.com/Kheil-Z/MmWave_Classification`

Once this information is available, we can use it to parse the streamed *.dat* file. It is organised as a series of packets for each frame [17], that is after each frame is recorded, the packets are sent over the UART interface during the inter-chirp processing time. In order to find the beginning of each packet, a *magic word* (*'\x02\x01\x04\x03\x06\x05\x08\x07'*) is appended to the beginning of the structure. Once this magic word is found in the stream, the packets have a standard structure which encompasses: (see figure 3.2)

- **Header** which indicates the number of TLV items selected in the configuration file. The header also contains an output message with diverse information (always starts with a *Magic Word*, number of frames, system version etc..).
- **TLV items**, comprised of: *TLV type* (6 types stated previously, see section 2.4.6), *TLV length* and the *Payload* (data) in a format specified on the mmWave SDK docs [36].
- Finally, the end of the packet is **padded** so that the total packet length is always multiple of 32 Bytes.

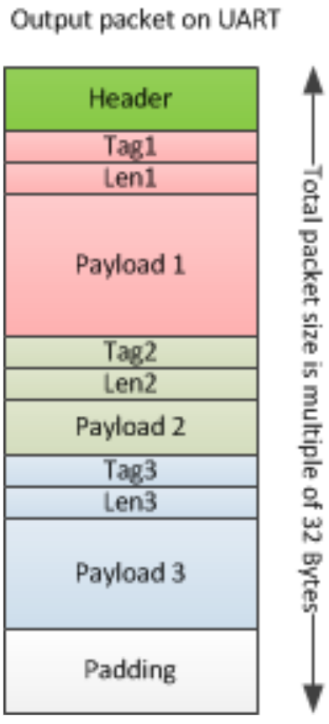


Figure 3.2: Packet structure - Figure from SDK doxygen [36]

Contents of the *Parsing* folder are used for this, and return the data as a *.txt* file with a custom structure, or eventually directly a *.csv* to be used for the learning process.

3.1.2 Data Choice

From the available data structures, each incorporate a different subset of physical information about the scene, not all are particularly useful for our problem. Classifying small handheld objects requires information such as the object's geometry and material composition (i.e reflective properties). In light of this, the Range Profile is not very useful as it contains the average intensity at each range bin, thus information is loss during the on-chip averaging operation. The Noise Floor Profile and Statistics Profile are also meaningless here.

This leaves us with the Range-Azimuth heatmap and the list of detected objects as viable structures. The Range-Azimuth heatmap directly provides information on the intensity reflected by the object, and somewhat its geometry. Meanwhile, the sensor can be configured such that there is no on-chip clustering of detected object lists and also we can lower the CFAR detection threshold such that points with lower intensities are also listed. This essentially means it becomes a list of detected points presenting an intensity over a certain threshold. This list can be used as a point cloud to infer the geometry of the object.

3.1.3 Sensor Configuration

We must also consider the data transmission rate of the board, the technical sheets indicate a baud rate of 115 200 bytes per second. Since the heatmap sizes' are linked to the number of bins configured.

The size of the Range-Azimuth heatmap is given by the number of range bins, and the number of angular bins which is hardcoded to 64. The number of range bins was fixed to 64.

$$\text{Size} = \#\ \text{rangeBins} * 64 * 4\ \text{bytes} = 64 * 64 * 4 = 16384$$

The size of the detected objects list depends on the ammount of detected objects, our configuration typically yielded about 20 objects, but the board allows a maximum of 100 points :

$$\text{Size} = 12 + \#\ \text{Objects} * 12\ \text{bytes} = 132$$

To which we must add roughly 40 bytes of header. The total is 16 556 bytes, which takes $\frac{16516}{115200} = 0.14$ seconds to transmit, the inter-frame time is fixed to 0.033 seconds, so we have a total of 0.17 seconds. So roughly 5 frames per second.

With this in mind, the Frame Rate was set to 4 fps, Furthermore, since this experiment was launched in a confined space, with roughly 3 meters of length, anything farther than that would be pointless, so the Maximum Unambiguous Range, which we set to 2.41 m, and this required setting the range resolution to 0.047m. The Velocity resolutions and bins were left to their default values, and the Antenna Configurations were set to 4RX,3TX in order to obtain elevation information.

Furthermore, advanced parameters were set to Improve the Detected Objects data. Peak grouping was disabled in both directions in order to obtain all points and not just one point representing the cluster, and the CFAR threshhold was lowered to

10dB in order to pickup more points. The full configuration file used is available in appendix figure 6.2.

The data being now ready to collect, and the sensor configured, we will proceed to discuss how exactly we gathered our data and trained the model.

3.2 Algorithms and Pipeline

One of the most critical steps to obtain a model with a good performance is the data handling. Clean and meaningful data will be easier to work with, for both the human, and the machine learning algorithms. This section discussed how we collected and processed our data.

3.2.1 Point Cloud processing

After parsing the raw data for each frame, the detected points where filtered according to their range index, any point with a range index smaller than 2 (roughly 7cm, chosen empirically) was considered near-antenna noise and ignored, also very large indexes (roughly 2m), belonged to background objects (walls or behind).

This left us with a list of point structures all belonging to the scene, these structures are composed of: range and Doppler indexes, x,y,z and velocity coordinates, intensity values and an ID number.

Since not all points necessarily belonged to the same object, another pre-processing step was to cluster the detected points to form groups belonging to the same object. Many unsupervised learning clustering algorithms were possible (K-Means, DBSCAN [37], Gaussian Mixture Models ...), but instead we chose to do this deterministically and not adopt a learning approach.

This choice was motivated by the sensor parameter we had deactivated in order to obtain more points: *peak grouping*. The problem was that when peak grouping was activated, the sensor did not return all the points found in groups, but it replaced the grouped points by a single point. Thus we had to disable peak grouping, and code the process manually in order to obtain the group, but also keep individual information about each point. Furthermore, this was much quicker to compute for a real-time application than looping over an unsupervised learning method.

The grouping was done by arranging points first by their range index, grouping the adjacent ones, then by Doppler index, and grouping adjacent groups. This is referred to as Range and Doppler Direction grouping. Note that this method was generally equivalent to DBSCAN results (sometimes a point or 2 would differ).

Once the points were grouped as belonging to different objects, we needed to voxelize them.

The problem with point clouds is that they are an un-ordered data structure, hence not well suited for learning techniques. Although some learning techniques such as PointNet, PointNet++ and MeshNet have been developed, and work quite well, in our case working directly on the point clouds was not appropriate. This is due to the fact that our point clouds are sparse and contain very few points, which is not suited for point cloud based learning. Furthermore, a recent paper has

shown that point-cloud based learning shows an inconvenient: ”up to 80% of the [computation] time is wasted on structuring the sparse data” , this paper suggested a mix of voxelisation and point cloud based learning.

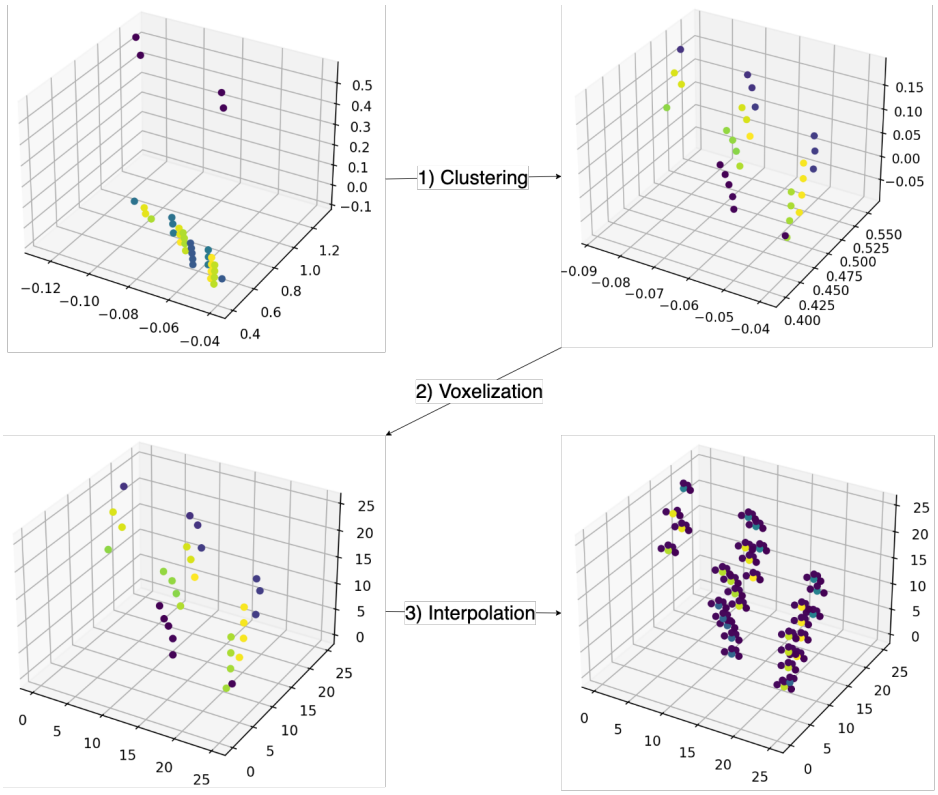


Figure 3.3: Voxel Processing Steps. Results from step 2 are used for the Heatmap.

Thus, we went down the voxelisation route, this proved to be adequate as we could simply apply 3d Convolutions to learn from the voxels.

In order to voxelize our point cloud, we min-max scaled the groups' points' coordinates to the [0,1] interval, then to [0,25]. The xyz coordinates between 0 and 25 were rounded, and used as voxel coordinates , with the value being the intensity of the detected point. The value of 25 was chosen empirically, as a tradeoff between small values which would lose resolution as points too close could be merged, and large values which would create a very sparce voxel (computationally efficient as the complexity grows *cubically*). This yielded a 26x26x26 voxel.

Finally, since we had 15-20 points for the object on average, only a handful of these 26x26x26 values would be filled. During training, different pre-processing interpolation methods were evaluated and checked for a positive influence on the learning metrics:

- Neighbouring voxels of each point were also filled, by either adding/subtracting the square root value of the intensity, or using the same value as the intensity, in function of which neighbour we were dealing with. This law was chosen due to the properties of reflected signals, their energy decreases proportionally to inverse square of the distance.

- Using scipy, we computed the convex hull of the point cloud, and generated additional points along the convex hull with intensities being the mean intensity of the 2 neighbouring points which were used.

The aggregation of voxel processing steps are represented in figure **??**, meanwhile the next section deals with the Heatmap processing steps.

3.2.2 Azimuth-Static Heatmap pre-processing

As previously stated, the Range-Azimuth Heatmap consists of the raw time-domain signal for each azimuth direction. We transformed this complex signal by applying an FFT on the azimuth signals, which results in a range profile with intensities at each range/azimuth bin. Furthermore, range profile was shifted in order to visualize the zero-frequency component at the center of the spectrum. The final result is what we call a heatmap, and can be projected from polar coordinates to cartesian ones for better visualisation. (cf figure 3.4)

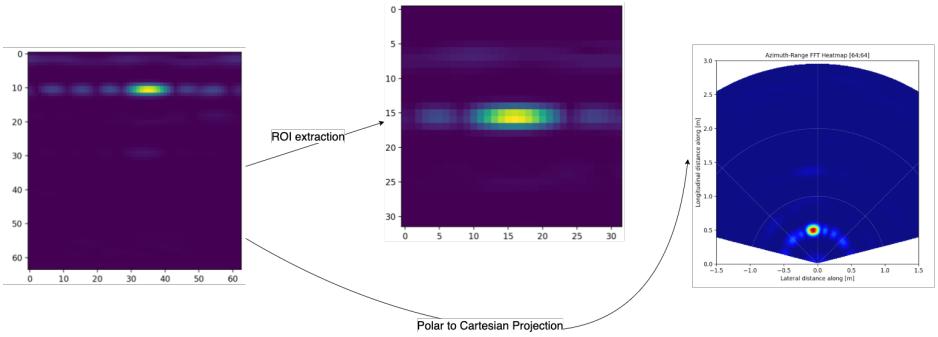


Figure 3.4: Heatmap Processing steps. The ROI is extrcated using data from the point cloud.

The heatmap represents data from all reflectors present in the field-of-view of the sensor, this could include static clutter, or other random objects present in the scene at the time of scan. Since not all of this information is important in order to learn the representations of our chosen classes, we cropped a fixed-size bounding box with the highest intensity point of the object in the center of the box. This bounding box called a *Region of Interest* (ROI) is composed of 32x32 cells independently of the distance from the sensor, this arbitrary value was a good compromise between the occupation capacity of the objects when up close or far from the sensor, and dealt well with resonant signals from the reflection when present.

In order to find the ROIs in the scene, we used the information from the detected objects data, after clustering them, for training and validation data the center was defined as the highest intensity from the cluster with the most points. We can imagine a scenario where at inference time, this would be done for all clusters and function as Region Proposal algorithm.

We should also note, that some Deep Learning networks, notably YOLO [38], incorporate Region Proposal directly in their network, and learn it alongside learning the classification network weights. This has proven to be quite efficient and require

less computation. The reason this was not done here, was first of all the simplicity of the application, labelling data with our current point-cloud based Region Proposal method works quite well, so there was no particular need to learn it, and also it is efficient to compute as the required data is readily available and requires no further computing. But we can imagine studying this in future work when more classes are added to the classification problem. It is noteworthy that this would probably require a Lidar to generate accurate training data for the bounding boxes.

Finally, the definite data used for learning was composed of the voxelized point cloud, and the ROI from the heatmap. This was a given a label corresponding to the object (0 for a spoon, and the positive label 1 for a knife). Both data structures were saved as separate .csv files, and we generated a third csv which contained the labels for the objects, along with the path to find the corresponding voxels and heatmap csv file.

3.2.3 Data Augmentation

Withing the framework of this project, generating a substantial training database was not feasible, hence with the intention to obtain more training data and reduce overfitting, we resorted to data augmentation techniques on both data structures. In both cases, we used Pytorch's *transforms* module, and randomly applied certain transformations to the input data.

For the Range-Azimuth heatmap ROI :

- Random Vertical flip.
- Random Horizontal flip.
- Random affine transformation with parameters defining an affine transformation with a maximum rotation in the range of $\pm 2°$, translation of ± 0.1 (fraction of image size) and scaling of ± 0.05.
- Gaussian Blurring.
- Adding Gaussian Noise to the image.

and for the voxels:

- Random Flip along dimension 0.
- Random Flip along dimension 1.
- Random Flip along dimension 2.
- Adding Gaussian Noise to the voxelized point cloud.

These transformations were picked among a myriad of possibilities mainly because of their physical signification, flipping, affine transforms all resulted in input data which could be seen in real cases. Moreover, different degrees of transformations (which ones to apply, with which probability and their parameters) were toyed with during the hyper-parameter tuning phase.

A final transformation was applied to that data: Normalization. We normalized all the training, validation and test data with the same parameters. The chosen normalization was operation was the Gaussian Normalization, it enables us to fit our data to a Gaussian of mean 0 and standard deviation of 1. This was done by finding the distribution of the collected data for the voxels and heatmaps separately. Normalizing the training data allows the network to converge faster, and avoids exposing it to very large numbers at input.

3.2.4 Experimental Setup and Data Collection

Having defined what sensor configuration was to be applied, which data types were relevant and how to process them, we defined our experimental setup and started collecting data.

The setup was fairly simple, the sensor was mounted horizontally on a flat table, facing the user (see Figure 3.5). The user alternated holding a knife, and a spoon while recording data using a custom script (also available on github). Since this was done at home, the cooking knife and spoon were kept the same during all the data gathering process, but the posture, orientation and distance they were held at were left to the discretion of the subject, but never exceeded roughly a meter.

In order to correctly compare network performances, and evaluate our final results 3 completely independent batches of data were recorded. They constituted the training set, validation set and test set. Only the training set was used for training the networks, which were then evaluated on the validation set as we will discuss later. Finally the test set was only used to evaluate the final model, which was trained on the aggregation of training and validation set data.

This subdivision in 3 sets is typically done in order to avoid data leakage, and is intended to reflect the real metrics of the model (otherwise we could be overfitting to the training set and consequently completely miss-evaluate the model on the true underlying data distribution). Moreover, it is noteworthy that the train/validation/test sets were recorded completely separately, and not just in one big batch which would traditionally then be shuffled and split (typically 70%/20%/10%). This was intentional, as although we had to reduce the sensor rate compared to what it can achieve in order to stream the data over the UART interface, the rate was still close to 4fps, and a quick analysis of the data revealed that this speed caused many frames to be quite similar, or almost identical depending on how far in time they were separated, and the behaviour of the subject. Thus, by shuffling and splitting one big batch of data, we would erroneously leak data from the training set to the validation and test sets. Thus the sets were recorded independently. Another custom solution to mitigate this effect, was to record a maximum of 50 frames in one experiment instead of the whole dataset, and aggregate recordings to constitute the datasets. The final distribution of the recorded data used for training and validation can be found in table 3.1.

Furthermore, note that these numbers differ from the actual number of recorded

Set	Type	Total
Training	knife	*737*
	spoon	*823*
Validation	knife	*150*
	spoon	*121*

Table 3.1: Actual frames kept for Training and Validation sets. N.B: Noisy background was incorporated in the different distances.

Set	Type	0.5m	1m	Moving	Noisy Background	Total
Training	knife	370	200	110	210	*890*
	spoon	350	200	110	200	*860*
	Total	720	400	220	410	*1750*
Validation	knife	50	50	50	-	*150*
	spoon	50	50	50	-	*150*
	Total	100	100	100	-	*300*

Table 3.2: Collected data distribution for Training and Validation sets.

frames as some frames do not meet the requirements for constituting ”valid” data (number of points in cloud inferior to 4, etc) which is summarised in Table 3.2.

All of the raw data frames, along the parsed and processed csv datasets can be found on the project's github. The next section introduces exactly how we constructed our Classifier.

3.2.5 Network

With the input data dealt with, we can now investigate the architecture of the network. Our problem consists in classifying objects belonging to 2 classes, from 2 different data structures described above. The custom network constructed is a combination of 2 independent feature extractor networks applied to each input structure separately. These networks transform the inputs into vectors of features. These vectors being much smaller, but more meaningful than the input data are then classified by a classification network which takes as input the two previous outputs.

Our inputs essentially being an image structure and voxels, using fully connected linear networks was not feasible, we investigated using different Convolutional Neural Networks (CNNs) through various forms for the feature extraction process. VGG (Visual Geometry Group) architectures, Residual Networks (ResNet) and custom CNNs were tested both for the heatmap and and for the voxels (by using

3D convolutional layers). The outputs of theses feature extractors were all set to the same sizes (chosen arbitrarily) :

- Heatmap : the input heatmap is of size 32x32, and the output feature vector was
$$\|\text{Heatmap Features}\| = (4 * 4) * \#\ \text{outputChannels}$$
- Voxels : the input voxel measured 26x26x26, for a desired output :
$$\|\text{Voxel Features}\| = (3 * 3 * 3) * \#\ \text{outputChannels}$$

The number of Output Channels was also used to define the breadth of the network as can be seen in Figure 4.3 for example.

Once the features were extracted, we flattened them into vectors which we concatenated to from a single feature vector of size (4x4 + 3x3x3)x # Output Channels. This vector was then passed through a fully connected linear network with an output of size 1 for the classification.

All networks incorporated ReLU activation functions, except for the final output layer which used a sigmoid activation. Furthermore, the use of Batch Normalisation after each layer was investigated, and also Dropout Layers with different probabilities.

What we obtain is a custom network composed of 2 independent feature extractors (heatmapFeatures and voxelsFeatures) whose outputs pass through a fully connected Network (Classifier). An example of such a network is summarized in figures 4.4, 4.3 and **??**.

The network consequently makes use of three additional hyper-parameters:

- Batchnorm : True or False
- Dropout Rate : ranging from 0 to 0.5
- Channels : used to specify the feature extractor networks' sizes and the classifier's input size.

The plethora of tested network architectures are described in appendix (figures 6.3, 6.4, 6.5, 6.6), and can still be used in the training script available on github.

The final chosen network architecture, and parameters are presented in section 4.0.1. Meanwhile the next section describes how we orchestrated the training process.

3.2.6 Training and grid search

With our different architectures in place, we now can train combinations of these three networks, along with various combinations of BatchNorms and dropout layers.

The training process was done using Pytorch's Adam optimizer, with most parameters left to their default values (β=(0.9, 0.999), $\epsilon = 10^{-8}$, weight_decay=0, amsgrad=False). Solely the *learning rate* parameter was used as a hyper-parameter and varied incrementally in the set [0.01;0.0001].

Considering that we have a binary classification problem, our output was of dimension 1. During inference we of course applied a Sigmoid activation in order to interpret the result, but during training we only output the logits, consequently, we made use of the Binary Cross Entropy Loss applied directly on logits (Pytorch's *BCEWithLogitsLoss*).

The training loop was applied on batches of inputs, with the batch size also being a hyper-parameter (64 or 128) and the number of epochs to loop was fixed to 30. At each epoch we trained the constructed network on the training set, then evaluated it on the validation set. This project made use of the Weights and Biases (WandB) tool in order to log all results from the tuning process. The logged variables included the hyper-parameters used to construct the network and parameterize the training process and are summarized below. These logged variables were completed by the training and validation losses, and typical classification metrics evaluated on the validation set: (Accuracy, Recall, Precision and F1 score) for each epoch. So as to achieve reproducibility in the experiments, all random generator seeds were set to 0, and in order to avoid further computation later on, for each hyper-parameter configuration and at each epoch, the validation results were checked against a preset minimum metric requirement, and saved in their current stated if validated. Thus, all saved models can directly be accessed from the github repository, and their metrics cross-referenced using their name on the WandB project page [2].

Bearing in mind the previous sections, the set of hyper-parameters is summarised in Table 3.3.

We should also underline the fact that in additional to these hyper-parameters, the usefulness of the input data itself was investigated. Although theoretically the network would learn to discard irrelevant input data and features, bear in mind the future use case of such a system would be on real-time and potentially on-chip predictions, thus the lighter the model, and the less inputs it requires, even at the cost of a slight decrease in metrics, the more viable the project becomes. Ergo, a few experiments were run with solely the heatmap, or voxels as input.

Hyper-parameters' influence was studied by implementing a grid search type training. We constructed combinations of hyper-parameters and used each combination to train and evaluate the model on the training/validation sets.

Finally, the whole training process and hyper-parameter grid search was run on a Collab notebook (available in the repository) such that we could make use of the CUDA environment's GPUs to speed up the process. In corcodance with the results of the validation set, the "best" model was selected and is discussed in further detailed in the following chapter.

[2] `https://wandb.ai/z_k/mmmwaveKnife_Spoon_HeatVox` and `https://wandb.ai/z_k/mmmwave_ROI_32x32`

Learning Rate	[0.01 ; 0.0001]
Batch Size	64 or 128
Heatmap Transforms	Combinations of Vertical Flip , Horizontal Flip, Random Affine, Gaussian Blurring, Gaussian Noise, None
Voxel Transforms	Combinations of: Random flip along dim 0,1,2 Gaussian Noise and None
Dropout Layer Proba.	[0 ; 0.5]
No. Output Channels	4 or 8
BatchNorm	True or False
Heatmap Network type	Custom CNN1.1, CNN1.2 VGGNet, ResNet
Voxel Network type	Custom CNN 2.1, VGGNet_3D

Table 3.3: Hyper-parameters Investigated

Figure 3.5: Data was gathered by placing the sensor facing a user, while the user held a spoon or knife in it's field of view.

Chapter 4

Experimental Results

The previous chapter went over the training process and the hyper-parameter tuning. The discussed hyper-parameters yielded a total of 194 runs which are documented on the WandB pages. The selected model was based on the validation set results, and the following sections detail the architecture and training parameters. The definitive results were evaluated on a final test set which was not used at all in the process.

4.0.1 Retained Model

The retained model (id: *young-dragon-39* [1]) was trained with a **learning rate** of **0.001**, and took input **batches of size 64**. The best input data did turn out to be both **heatmap and voxel** structures, with **all augmentation techniques** applied. The network architecture was composed of **8 output channels**, **BatchNorm layers activated** and **Dropout Layers were not used**. Finally the networks used were **ResNet** for the **Heatmap input** and **VGGNet** for the **Voxel input**.

Hence, the full network is:

- Heatmap Feature Exctractor (ResNet, see Fig 4.4): The input data is passed through a succession of Convolutional Blocks and BatchNorm blocks then Residual Connections are applied and finally a ReLU activation. This is done 3 times, with channels spanning from the 1 to 16, 32 down to 16 and to the output 8.
- Voxel Feature Exctractor (VGGNet, see Fig 4.3): Sequentially we apply Visual Geometry Groups composed of a 3d Convolution (kernel size 3 and padding of 1), Batchnorm, ReLu twice, which leaves the image size indentical, then a MaxPooling which halves the voxel size. This block is repeated 6 times, witch channels spanning as follows : 1-16-16-32-16-8.

Which both output their results to the Classification Network composed of 2 fully connected layers (see Fig 4.5):

- fc1 : 8*(4*4 + 3*3*3) to 128.
- ReLU activation

[1] `https://wandb.ai/z_k/mmmwaveKnife_Spoon_HeatVox/runs/3u6vs4fl`

- fc2: 128 to 1 output.

The whole network can be visualized in the appendix (figure 6.1).

Finally, our best model was saved at the 11^{th} epoch, after which the model begun showing signs of overfitting. Thus **early stopping** was applied to stop training after epoch 11. The loss over the 30 epochs, and truncated to our model can be seen in Figures 4.1 and 4.2.

Figure 4.1: Train and Validation loss plot for the full 30 epochs training

Figure 4.2: Train and Validation loss plot when stopped at 11 epochs.

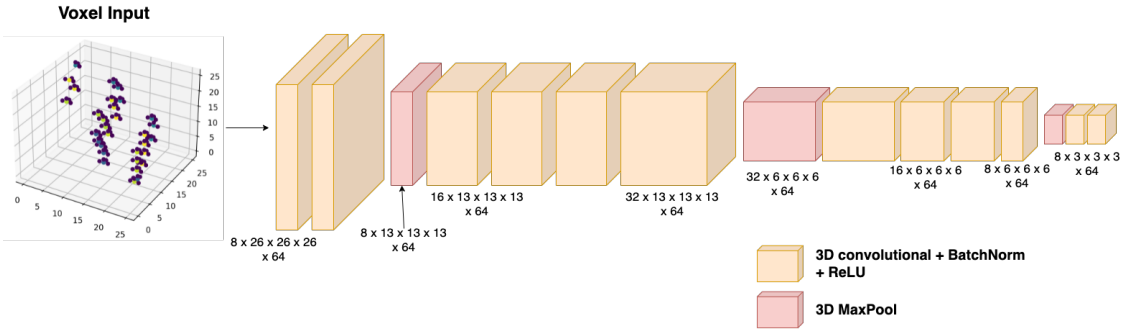


Figure 4.3: Voxel Network : VGGNet to exctract features from the voxelized pointcloud.

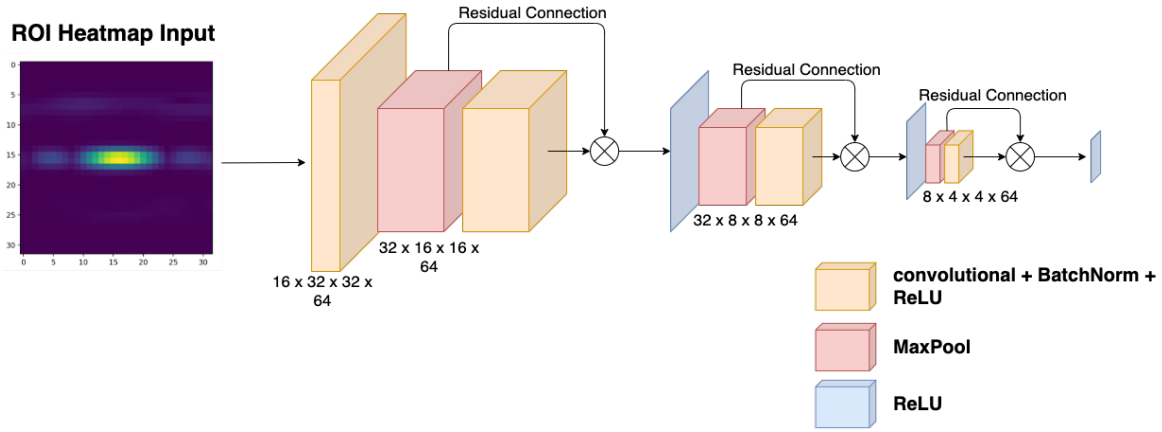


Figure 4.4: Heatmap Network : ResNet to exctract features from the ROI of the heatmap.

4.0.2 Benchmarks and Results

In order to correctly evaluate the final model, we now made use of the 3rd dataset, the test set which was never used previously.

The model presented an **accuracy of 0.82**, **recall of 0.8**, **precision of 0.81** and **F1 score of 0.8** . Given that our positive label is the one associated to the knife and our Recall score being fairly high, this means our model is particularly well suited for avoiding False Negatives, i.e a spoon will hardly ever be predicted as a knife. Considering the underlying weapon detection application, this is what seems to be most important. But note that if a specific use case required a better precision rather than recall, our best model at maximizing precision reached roughly 0.9. precision.

The Confusion matrix of the model is represented in figure 4.1.

Additional testing was conducted in specific cases such as evaluating the metrics in specific settings, and even settings not addressed in the training sets.

The model shows a high sensitivity to distance: the same object will not be categorized the same in function of the distance. As we can see from table 4.2, the model is still not well suited for long-range detection. It is essentially unusable for

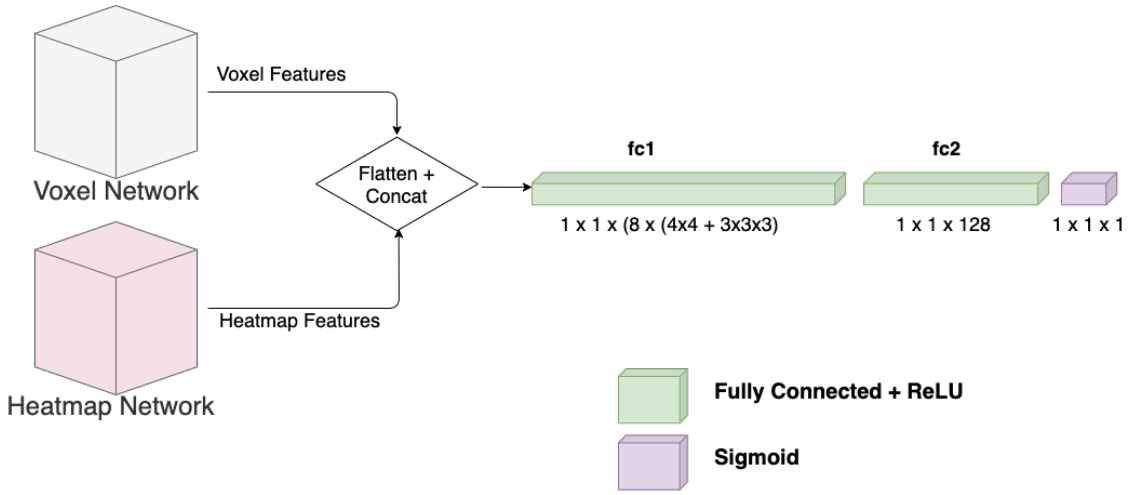


Figure 4.5: Classifier Network : Fully Connected Layers, to classify outputs from Heatmap and Voxel Networks (Figures 4.4 and 4.3)

		True Class	
		Positive	Negative
Predicted Class	Positive	225	57
	Negative	58	243

Table 4.1: Confusion matrix of out trained model. N.B : the Positive class represents the knife. (Results : Acc = 0.83, Pr=0.81 , Recall=0.8, F1=0.8)

over 1.5m. Further investigation on the results as the distance grows reveal that the model will often output results as being spoons. This is due to 2 factors:

- Since we are using the concave side of the spoon when training and testing, the spoon is by definition a *strong reflector*. Thus it's capacity at reflecting signals is superior to that of a flat object (such as the knife).
- Furthermore, Since the angular resolution is constant, the farther we are from the sensor, the less the greater the resolution of the grid, ergo the less information is available on the heatmap, but even more so in the point cloud. Thus the network has less information to decide. Also, given our network structure (predictions of 1 = knife, 0= spoon), the more our inputs are close to being null, the more likely we could be to output small values. This issue could be fixed by instead using one-hot vectors, and outputting 2 values which would the probability of a spoon, the one of a knife.

Another potential solution to this problem is expanding our training set by gathering more datasets from a greater distance, this would highlight whether the problem emanates from the training process or a hardware restriction from the sensor itself.

Distance	True Positive	False Negative	False Positive	True Negative	Accuracy
0.5m	100	0	14	86	0.93
1m	70	30	0	100	0.85
1.5m	3	97	0	100	0.51

Table 4.2: Influence of Distance on predictions.

(a) (b) (c) (d) (e) (f)

Figure 4.6: Knives and Spoons used .4.6(a) and 4.6(b) were used for training. Meanwhile 4.6(c),4.6(d),4.6(e)and 4.6(f) were never seen by the model except at test time.

On the other hand, our model proved to generalize fairly well. A potential problem in our training pipeline was the fact we only exposed the network to the same spoon and knife, hence there was a non negligible risk that our model would overfit by only learning the representation of this particular knife and spoon. Testing showed that using other knives and spoons did not affect our metrics, the model learned the underlying representation of what differentiates a spoon from a knife. The benchmarks for the objects from figure 4.6 were substantially similar to the original knife and spoon (Accuracy ≈ 0.81).

Also on the brighter side of results, since this whole project is after all done in the frame of Concealed Weapon Detection, we carried out further testing with the sensor being obstructed by a few sheets of paper, and also by a typical t-shirt. Remarkably, even though none of the training data was collected in this concealed manner, the network generalized well to this use case, and presented an accuracy of 0.78. The confusion matrix for the concealed test set can be seen in table 4.3.

		True Class	
		Positive	Negative
Predicted Class	Positive	66	13
	Negative	31	86

Table 4.3: Confusion matrix of the model on objects concealed by sheets of paper and a t-shirt. (Results : Accuracy = 0.78, Prcision=0.84 , Recall=0.68, F1=0.75)

Chapter 5

Conclusion

This thesis paves the way for a long-range, low-cost and low-maintenance system for Concealed Weapon Detection by making use of modern and robust Millimeter Wave sensors.

Although CWD systems are widely used today and present very solid performances compared to our current system, they also inherently contain strong constraints which can not always be respected. In particular, they usually require lavish hardware, considerable power sources, and are not effective at more than a few meters. Our initial use case for poaching prevention is a poignant example of the inefficiencies of CWD today. Through this project, we highlighted the possibility of using MMwave sensors in order to detect and classify small and concealed objects, whether static or in movement.

In conclusion, compared to previous Mmwave based classification works, our work presents a decrease in accuracy, but recognizes smaller objects than previous experiments. Also, this paper demonstrates classification abilities without the use of Doppler information, such as the widely used Doppler Signature, thus there is no inherent constraint on the exact movement of the subjects.

5.0.1 Future Work

This thesis presents a few shortcomings which we wish to address. First of all the range of detection. Since the drastic decrease in accuracy over a few meters is due to the inherit resolution of the sensor, we wish to investigate the use of an array of sensors in order to increase the resolution of our system, and potentially increase the accuracy of classification for longer ranges.

Furthermore, when exposed to concealed data the system sometimes presented a few issues when suggesting Regions of Interest, this could be improved by implementing a YOLO-style network which would also learn the region proposal algorithm, but be slightly more time-consuming.

Finally, the use of of higher or lower signal frequencies which will affect resolution, but also penetration of the waves should be investigated, this would require different sensors than the one we currently have.

Bibliography

[1] World Bank. *Illegal Logging, Fishing, and Wildlife Trade*. World Bank, Washington, DC, October 2019. doi: 10.1596/32806. URL http://hdl.handle.net/10986/32806. pages 1

[2] W. Avis. Criminal networks and illicit wildlife trade. Technical Report 150, Institute of Development Studies, Brighton, UK, 2017. URL https://assets.publishing.service.gov.uk/media/5975df3ded915d59ba00000a/150-Illicit-Wildlife-Trade.pdf. pages 1

[3] Six rangers killed in DR Congo's Virunga National Park. *BBC News*, January 2021. URL https://www.bbc.com/news/world-africa-55611203. pages 1

[4] Over One Thousand Park Rangers Die in 10 Years Protecting Our Parks and Wildlife | News | Global Conservation. URL https://globalconservation.org/news/over-one-thousand-park-rangers-die-10-years-protecting-our-parks/. pages 1

[5] Smith Lucy Olivia and Lucas Porsch. The Costs of Illegal Wildlife Trade: Elephant and Rhino. Technical report, Ecologic Insititute, Berlin, 2015. URL https://efface.eu/sites/default/files/EFFACE%20D3.2c%20-%20Quantitative%20and%20monetary%20analysis%20of%20Elephant%20and%20Rhino%20hunting.pdf. pages 2

[6] Khristopher Carlson. *In the Line of Fire: Elephant and Rhino Poaching in Africa*. 06 2015. ISBN 9781107690677. pages 2, 6, 7

[7] Office on Drugs and Crime. *World wildlife crime report: trafficking in protected species, 2020*. United Nations publication. United Nations, New York, 2020. ISBN 978-92-1-148349-9. URL https://www.unodc.org/documents/data-and-analysis/wildlife/2020/World_Wildlife_Report_2020_9July.pdf. pages 2

[8] Organitzacio Mundial del Turisme. *Towards Measuring the Economic Value of Wildlife Watching Tourism in Africa - Briefing Paper*. Organizacion Mundial del Turismo, Madrid, 2016. ISBN 9789284416752. URL https://www.e-unwto.org/doi/book/10.18111/9789284416752. OCLC: 1120647481. pages 2, 3

[9] Jacob Kamminga, Eyuel Ayele, Nirvana Meratnia, and Paul Havinga. Poaching Detection Technologies—A Survey. *Sensors*, 18(5):1474, May 2018. ISSN 1424-8220. doi: 10.3390/s18051474. URL http://www.mdpi.com/1424-8220/18/5/1474. pages 3

[10] Alan Agurto Goya. New proposal for the detection of concealed weapons: Electromagnetic weapon detection for open areas. July 2009. URL http://eprints.hud.ac.uk/id/eprint/7067/. pages 3, 4, 5

[11] Wang Yajie and Lu Mowu. Image fusion based concealed weapon detection. 12 2009. doi: 10.1109/CISE.2009.5364581. pages 5

[12] The Truth About Terahertz. URL https://spectrum.ieee.org/aerospace/military/the-truth-about-terahertz. pages 5

[13] S. Rao. Introduction to mmwave sensing: Fmcw radars. URL https://training.ti.com/sites/default/files/docs/mmwaveSensing-FMCW-offlineviewing_0.pdf. publisher: TI. pages 8

[14] C. Iovescu and S. Rao. The fundamentals of millimeter wave radar sensors, 2017. URL https://www.ti.com/lit/wp/spyy005a/spyy005a.pdf?ts=1622624610449&ref_url=https%253A%252F%252Fwww.ti.com%252Fsensors%252Fmmwave-radar%252Foverview.html. publisher: TI. pages 8, 9

[15] Christos Ilioudis. Introduction to radar signal processing. URL https://udrc.eng.ed.ac.uk/search/node/INTRODUCTIONTORADARSIGNALPROCESSING. pages 10, 11

[16] IWR1443BOOST Evaluation board | TI.com. URL https://www.ti.com/tool/IWR1443BOOST. pages 11

[17] Texas Instuments. Mmw demo data structure v0.1. URL https://e2e.ti.com/cfs-file/__key/communityserver-discussions-components-files/1023/mmw-Demo-Data-Structure_5F00_8_5F00_16.pdf. pages 12, 19

[18] by. Basics of Multilayer Perceptron - A Simple Explanation of Multilayer Perceptron, January 2018. URL https://kindsonthegenius.com/blog/basics-of-multilayer-perceptron-a-simple-explanation-of-multilayer-perceptron/. pages 14

[19] Mayank Mishra. Convolutional Neural Networks, Explained, September 2020. URL https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939. pages 15

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. ISSN 00189219. doi: 10.1109/5.726791. URL http://ieeexplore.ieee.org/document/726791/. pages 15

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6): 84–90, May 2017. ISSN 0001-0782, 1557-7317. doi: 10.1145/3065386. URL https://dl.acm.org/doi/10.1145/3065386. pages 15

[22] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. Very Deep Convolutional Networks for Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1107–1116, Valencia, Spain, 2017. Association for Computational Linguistics. doi: 10.18653/v1/E17-1104. URL http://aclweb.org/anthology/E17-1104. pages 15

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. URL http://arxiv.org/abs/1512.03385. arXiv: 1512.03385. pages 15

[24] Johan Jaxing, Thomas Johansen, Andreas Lindström, and Aren Moosakhanian. Deep machine learning for gesture recognition in IoT applications. 2019. URL https://odr.chalmers.se/handle/20.500.12380/257448. pages 16

[25] Yu Liu, Yuheng Wang, Haipeng Liu, Anfu Zhou, Jianhua Liu, and Ning Yang. Long-Range Gesture Recognition Using Millimeter Wave Radar. *arXiv:2002.02591 [cs, eess]*, February 2020. URL http://arxiv.org/abs/2002.02591. arXiv: 2002.02591. pages 16

[26] Jacopo Pegoraro, Domenico Solimini, Federico Matteo, Enver Bashirov, Francesca Meneghello, and Michele Rossi. Deep learning for accurate indoor human tracking with a mm-wave radar. In *2020 IEEE Radar Conference (RadarConf20)*. IEEE, 2020. pages 16

[27] Tokihiko Akita and Seiichi Mita. Object tracking and classification using millimeter-wave radar based on LSTM. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019. pages 16

[28] Tokihiko Akita and Seiichi Mita. Accurate parking scene reconstruction using high-resolution millimeter-wave radar. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020. pages 16

[29] Jonas Weis and Avik Santra. One-Shot Learning for Robust Material Classification Using Millimeter-Wave Radar System. *IEEE Sensors Letters*, 2(4), December 2018. ISSN 2475-1472. doi: 10.1109/LSENS.2018.2878041. URL https://ieeexplore.ieee.org/document/8509596/. pages 16

[30] Junfeng Guan, Sohrab Madani, Suraj Jog, Saurabh Gupta, and Haitham Hassanieh. Through fog high-resolution imaging using millimeter wave radar. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2020. pages 16

[31] Fahad Jibrin Abdu, Yixiong Zhang, Maozhong Fu, Yuhan Li, and Zhenmiao Deng. Application of deep learning on millimeter-wave radar signals: A review. *Sensors (Basel)*, 21(6):1951, 2021. pages 16, 17

[32] Ole Schumann, Jakob Lombacher, Markus Hahn, Christian Wohler, and Jurgen Dickmann. Scene Understanding With Automotive Radar. *IEEE Transactions on Intelligent Vehicles*, 5(2):188–203, June 2020. ISSN 2379-8904, 2379-8858. doi: 10.1109/TIV.2019.2955853. URL https://ieeexplore.ieee.org/document/8911477/. pages 17

[33] Marcel Sheeny, Andrew Wallace, and Sen Wang. 300 GHz radar object recognition based on deep neural networks and transfer learning. *IET Radar, Sonar & Navigation*, 14(10):1483–1493, October 2020. ISSN 1751-8784, 1751-8792. doi: 10.1049/iet-rsn.2019.0601. URL https://onlinelibrary.wiley.com/doi/10.1049/iet-rsn.2019.0601. pages 17

[34] Kanil Patel, Kilian Rambach, Tristan Visentin, Daniel Rusev, Michael Pfeiffer, and Bin Yang. Deep Learning-based Object Classification on Automotive Radar Spectra. In *2019 IEEE Radar Conference (RadarConf)*, pages 1–6, Boston, MA, USA, April 2019. IEEE. ISBN 9781728116792. doi: 10.1109/RADAR.2019.8835775. URL https://ieeexplore.ieee.org/document/8835775/. pages 17

[35] 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, January 2016. ISSN 1545-598X, 1558-0571. doi: 10.1109/LGRS.2015.2491329. URL http://ieeexplore.ieee.org/document/7314905/. pages 17

[36] Texas Instruments. [SDK_directory]/mmwave_sdk_02_00_00_04/packages/ti/demo/xwr16xx/
in MMWAVE-SDK. URL http://www.ti.com/tool/MMWAVE-SDK. pages 19

[37] Anant Ram, Sunita Jalal, Anand S. Jalal, and Manoj Kumar. A Density Based Algorithm for Discovering Density Varied Clusters in Large Spatial Databases. *International Journal of Computer Applications*, 3(6):1–4, June 2010. ISSN 09758887. doi: 10.5120/739-1038. URL http://www.ijcaonline.org/volume3/number6/pxc3871038.pdf. pages 21

[38] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, Las Vegas, NV, USA, June 2016. IEEE. ISBN 9781467388511. doi: 10.1109/CVPR.2016.91. URL http://ieeexplore.ieee.org/document/7780460/. pages 23

Chapter 6

Appendix

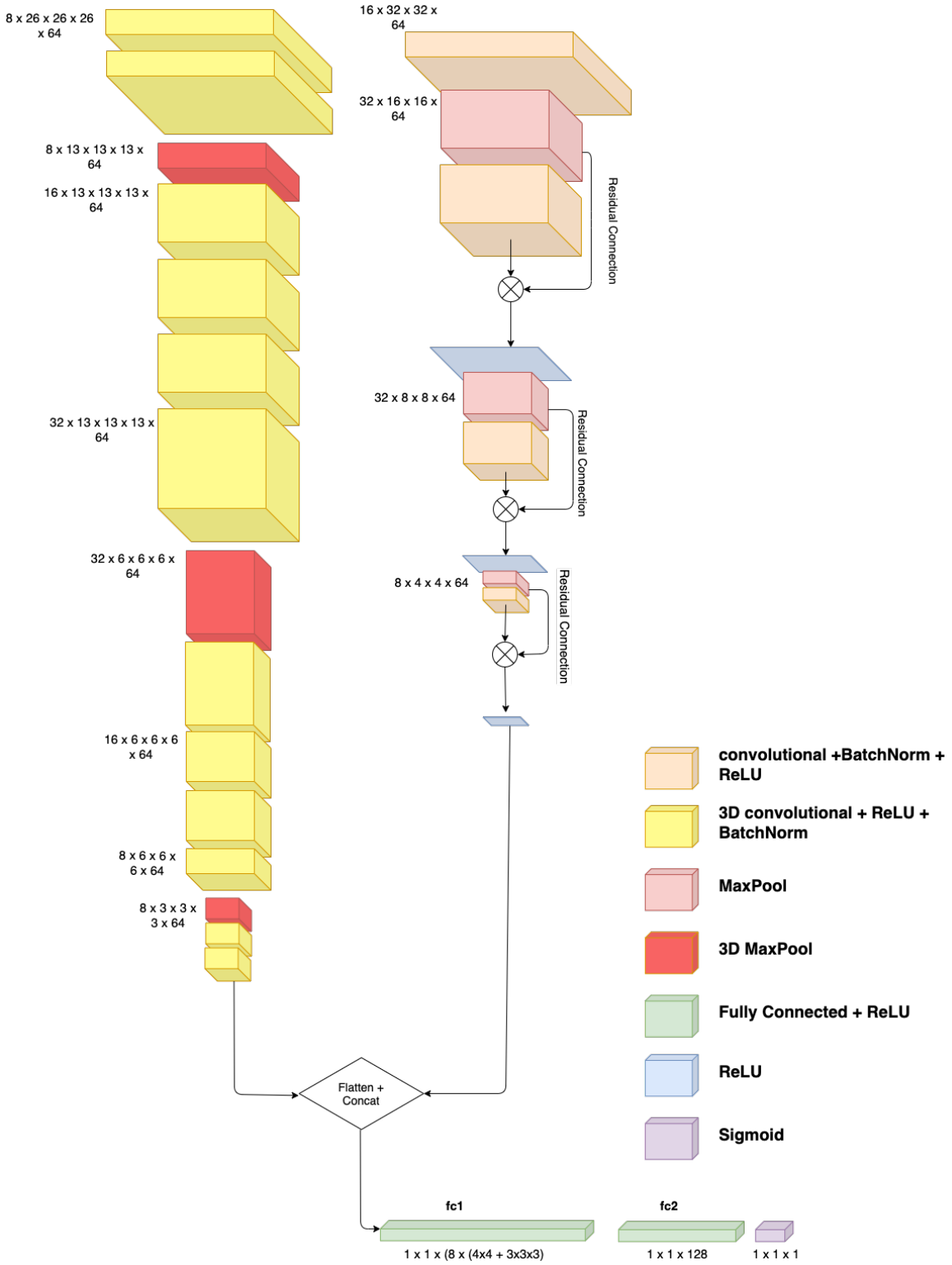


Figure 6.1: Representation of the retained Network in its entirety

```
% ****************************************************************
% Created for SDK ver:02.01
% Created using Visualizer ver:2.1.0.3
% Frequency:77
% Platform:xWR14xx
% Scene Classifier:best_range_res
% Azimuth Resolution(deg):15 + Elevation
% Range Resolution(m):0.047
% Maximum unambiguous Range(m):2.41
% Maximum Radial Velocity(m/s):1
% Radial velocity resolution(m/s):0.13
% Frame Duration(msec):250
% Range Detection Threshold (dB):15
% Range Peak Grouping:disabled
% Doppler Peak Grouping:disabled
% Static clutter removal:disabled
% ****************************************************************
sensorStop
flushCfg
dfeDataOutputMode 1
channelCfg 15 7 0
adcCfg 2 1
adcbufCfg 0 1 0 1
profileCfg 0 77 284 7 40 0 0 100 1 64 2000 0 0 30
chirpCfg 0 0 0 0 0 0 0 1
chirpCfg 1 1 0 0 0 0 0 4
chirpCfg 2 2 0 0 0 0 0 2
frameCfg 0 2 16 0 250 1 0
%frameCfg 0 2 16 0 1000 1 0
lowPower 0 1
guiMonitor 1 1 0 1 0 0
cfarCfg 0 2 8 4 3 0 1280
peakGrouping 1 0 0 1 229
% 229
multiObjBeamForming 1 0.5
clutterRemoval 0
calibDcRangeSig 0 -5 8 256
compRangeBiasAndRxChanPhase 0.0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
measureRangeBiasAndRxChanPhase 0 1.5 0.2
CQRxSatMonitor 0 3 4 99 0
CQSigImgMonitor 0 31 4
analogMonitor 1 1
sensorStart
```

Figure 6.2: Configuration file used throughout the project. Generated using TI's Tools.

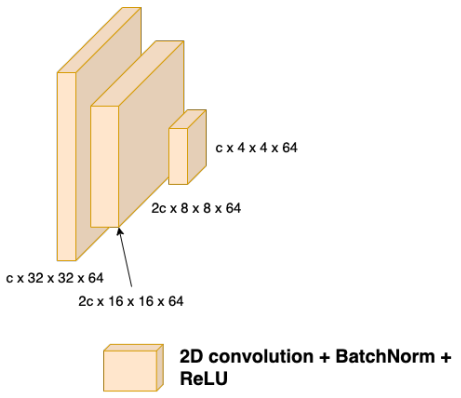


Figure 6.3: Other heatmap Network investigated (Model 1)

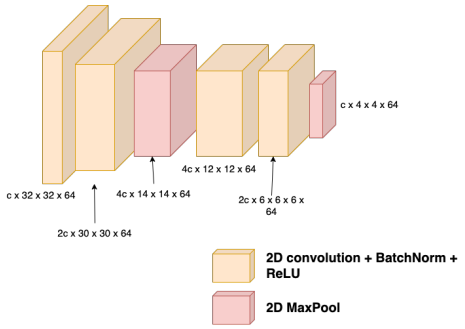


Figure 6.4: Other heatmap Network investigated (Model 2)

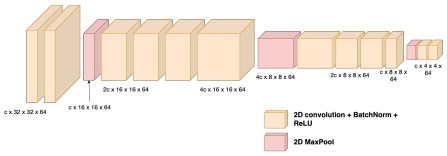


Figure 6.5: Other heatmap Network investigated (Model VGG)

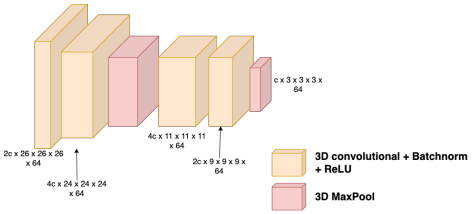


Figure 6.6: Other voxel Network investigated (Model Voxel VGG)