FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

LTE
Lehrstuhl für Technische Elektronik

Lehrstuhl für Technische Elektronik
Prof. Dr.-Ing. Dr.-Ing. habil. Robert Weigel
Prof. Dr.-Ing. Georg Fischer

Master Thesis

in the course of study
"Communications and Multimedia Engineering (CME)"

from

Venkat Ramana Madhavan

on the topic

Deep Learning based Target Detection on Radar Data

Supervisor:	Anand Dubey
	Jonas Fuchs
Begin:	15.08.2019
End:	09.03.2020

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

LTE
Lehrstuhl für Technische Elektronik

Lehrstuhl für Technische Elektronik
Prof. Dr.-Ing. Dr.-Ing. habil. Robert Weigel
Prof. Dr.-Ing. Georg Fischer

Masterarbeit

im Studiengang
"Communications and Multimedia Engineering (CME)"

von

Venkat Ramana Madhavan

zum Thema

Deep Learning based Target Detection on Radar Data

Betreuer:	Anand Dubey
	Jonas Fuchs
Beginn:	15.08.2019
Abgabe:	09.03.2020

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.

Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den 09.03.2020

Venkat Ramana Madhavan

Kurzfassung

Die Zieldetektion ist eine der wichtigsten Anwendungen für Radar im Bereich des autonomen Fahrens. Die Zieldetektion dient der korrekten Erkennung von Zielen im Radarbild. Dies kann durch den Vergleich des Frequenzspektrums des gemessenen Signals mit einer bestimmten Detektionsschwelle erreicht werden. Die traditionelle Radarsignalverarbeitung beinhaltet mehrere Schritte: der Matched-Filter Filterung, die Doppler-Verarbeitung und zusätzliche Vorbverarbeitungsschritte. Die Verwendung eines konstanten Schwellenwertes kann eine große Anzahl falscher Detektionen zur Folge haben. Es wird die klassische Methode der constant false alarm rate (CFAR) verwendet, welche in der Lage ist, einen variablen Schwellenwert unter Berücksichtigung des geschätzten Grundrauschens zu berechnen. Es werden verschiedene Arten von CFAR-Algorithmen entwickelt, um die Genauigkeit zu erhöhen und die Rechenleistung zu reduzieren. Doch obwohl die Algorithmen nach dem aktuellen Stand der Technik gut funktionieren, geraten sie bei Störung durch Mehrwegeausbreitung an ihre Grenzen. Mögliche Verbesserungen in diesen Bereichen können durch den Einsatz verschiedener Methoden der Künstlichen Intelligenz (KI) erreicht werden. Das Konzept der Segmentierung wird in der Zielerkennung verwendet, um Objekte zu erkennen und zu kennzeichnen.

Diese Masterarbeit widmet sich der Zielerkennung mit einem Frequency Modulated Continous Waveform (FMCW). Der erste Teil befasst sich mit der Implementierung traditioneller Algorithmen basierend auf CFAR. Danach wird ein Deep Learning Framework implementiert, um eine genaue Objekterkennung durchzuführen. Bei der Aufbereitung der Radar-Trainingsdaten werden Entfernungs-Doppler-Bildern aus einem Simulationsszenario generiert. Die Ground Truth eines Objekts, auch genannt Labels, werden nach den implementierten CFAR-Erkennungsalgorithmen ermittelt. Zur Lösung der Zieldetektion werden im Wesentlichen zwei verschiedene Segmentierungstechniken eingesetzt. Zunächst werden die Netzwerke auf einem gegebenen Einzeldatensatz trainiert. Das System wird nach entsprechendem Training in der Lage sein, das Ziel in einer bestimmten Umgebung zu erkennen. Der nächsten Schritte wird die Leistung eines solchen Detektors mit den traditionellen Architekturen verglichen. Abschließend wird eine Zusammenfassung sowie ein Ausblick auf der Arbeit gegeben.

Abstract

Target detection is one of the most important applications for radar. The goal of target detection is to correctly detect targets in the radar scene without missing any targets while also reducing false alarms. This can be achieved by comparing the frequency spectrum of the measured signal to a specific detection threshold. The traditional signal processing of radar includes several steps, which are matched filtering, doppler processing, and further pre-processing. Using a constant threshold value may cause a large number of false detections. The classical hypothesis called constant false alarm rate (CFAR) algorithms are used, which are able to calculate an adaptive threshold value due to the estimated noise floor. Different types of CFAR algorithms are developed in order to increase the precision and reduce the computing power. Although the state-of-the-art algorithms perform well, they have limitations in case of multiple targets and multi-path interference. The areas of improvement can be brought by applying different artificial intelligence (AI) techniques. The idea of segmentation is used in the object detection in order to classify and label objects.

This thesis considers object detection using an frequency modulated continuous waveform (FMCW) radar. The state-of-the-art deep learning framework is employed after the traditional signal processing methods. In preparing the radar training data, the range-Doppler maps are generated from a simulation. The ground truth of an object, so called labels is obtained after the state-of-the-art CFAR detection algorithms. Mainly, two different segmentation techniques are used to solve the classification task. Initially, the networks are trained with respect to a given single data set. In this regard, deep neural networks are used in the architecture. The system after proper training will be able to detect targets in a given environment. The performance of the implemented target detectors is then compared with the traditional architectures. Finally, a summary as well as an outlook over the complete thesis is given.

Contents

List of Abbreviations

ACC	Autonomous Cruise Control
AF	Activation function
AP	Average Precision
CASH-CFAR	Cell Averaging Statistic Hofele
CFAR	Constant False Alarm Detection
ConvNet	Convolutional Neural Network
CUT	cell under test
DBSCAN	Density based spatial clustering applications with noise
DFT	Discrete Fourier Transformation
DL	Deep learning
FC	Fully connected
FFT	Fast fourier transformation
FMCW	Frequency Modulated Continuous Waveform
FN	False negative
FP	False positive
FPN	Feature Pyramid Network
GOCA-CFAR	Greatest-of Cell-averaging CFAR
IoU	Intersection of Union
mAP	mean Average Precision
Mask R-CNN	Mask Region based convolutional neural network
OPTICS	Ordering Points To Identify Cluster Structure
OS-CFAR	Ordered Statistic CFAR
RD	range-Doppler
ReLU	Rectified Linear Unit
RPN	Region Proposal Network
SNR	signal-to-noise ratio
SOCA-CFAR	Smallest-of Cell-averaging CFAR
SVM	Support vector machine
TN	True negative
TP	True positive
VIA	VGG Image Annotator
YOLO	You only look once

1 Introduction

1.1 Motivation

Radar systems play a vital role in applications like autonomous driving, intelligent road infrastructure and surveillance systems. There are many reasons for choosing radar sensors over for normal image sensors. Firstly, radar sensors are not affected by weather and light conditions. Image sensors would be of limited use in low light conditions or during bad weather conditions. Furthermore, certain radar types like Frequency Modulated Continuous Waveform (FMCW) radars are able to directly measure the distance to an object (range) as well as the radial velocity (Doppler) of an object. The unique properties of radars can be as follows. Design freedom is an important factor and can include the ability to adjust the field pattern of the sensor as well as the appearance of the sensor to the end users. Radar sensors can be designed aesthetically behind a wall or a fixture so that there is no visible sensor seen. This could be useful for security sensors that can monitor motion behind a wall. Radar sensors can be combined with optical cameras where radar serves as a longer distance activator for the camera.

Being able to detects objects correctly is an important aspect for these applications. In image signal processing domain a lot of progress has been achieved, especially by the introduction of Convolutional Neural Network (ConvNet). However, it is not clear if the same networks can be applied to radar sensors as well. First of all, radar sensors have lower spatial resolution than image sensors making it harder to determine the shape and size of objects. Furthermore, only when objects are moving so-called Doppler information is available to perform classification. Lastly, the assumption upon which convolutional neural networks are based, namely that the input is translation-invariant, meaning that a shape at one location of the image has the same meaning at different locations, does not hold for radar images.

This discussion for complex applications like autonomous driving, classification alone is not sufficient; the task of tracking these objects over time needs to be addressed as well. In [1], researchers presented an algorithm to track multiple objects in FMCW radar data. This algorithm consists of several steps to perform the task: thresholding, clustering, tracking based on Kalman filter, feature extraction and finally classification. However, looking again back to the image domain where several neural network based multiple object detection methods have been proposed such as You only look once (YOLO) object detection system, variants of ConvNets, Autoencoders and many more. This raises the question if these methods can be successfully applied in the radar domain as well and how such a neural network based approach can be compared to the traditional algorithmic approach.

1.1.1 Automotive Radar

Radar has been used in automotive applications for a long time. Already in 1949, unfortunate car drivers were issued speeding tickets based on speed measurements obtained from the radar speed gun, invented by John L. Barker Sr. However, on-board automotive radar was not made commercially available until 1999, when it was introduced for collision warning and Autonomous Cruise Control (ACC). The second point to discuss is interference, which increases with more interfering radars and leads to false alarms and as a result false detections. In recent years, there has been a strong push to increase the integration level of millimeter wave electronics used for automotive radar and industrial radar sensors.

Today most radar transmissions are uncoordinated, meaning that there is no apriori agreement on who is allowed to transmit and when. A number of recent studies in [2] have indicated the interference situations which are likely to rise as the automotive radar transceiver market penetration increases. FMCW waveforms can, up to a point, relatively easily be repaired in case it is intermittently corrupted by interference, which is why they are still operational [3].

1.2 Tasks

This thesis serves as a purpose for radar target detection methods with the principles and fundamentals in the beginning. The remainder of this thesis is structured as follows: Section 2 provides the fundamentals. Section 3 discusses the requirements analysis and data generation. Section 4 introduces the state-of-art algorithms and deep learning approaches used for target detection. Section 5 throws light on the simulation results and comparative analysis. The conclusion follows with the future scope of the work that can be implemented.

2 Fundamentals

2.1 Radar Systems

Pulse-modulated continuous waves have short power pulses. Two targets can be separated in the range domain only if they produce non-overlapping returns in the time domain. This determines a performance measure of radar namely range resolution. Range resolution tells the ability to distinguish closely spaced targets. Hence, the range resolution is proportional to the pulse width T_p. In other words, finer pulses provide higher resolution. On the other hand, they have low energy which result in poor signal-to-noise ratio (SNR) and detection performance. Therefore we use a technique called *pulse compression* which uses frequency modulated pulses in order to overcome this problem.

2.1.1 FMCW Basics

In any radar, the electromagnetic wave is sent into the environment containing various objects. Then the echo of the wave is captured at a receiver. A simplified block diagram of such a system is shown in Figure 2.1, in which both the transmitter and the receiver are at the same location. Each chirp at the output of the FMCW generator is a sinusoid signal whose frequency is swept from fmin to fmax in Figure 2.2. Here the frequency is swept linearly with a positive slope of K and a duration of T_r implying that the sweeping bandwidth B is $B = KT_r$. The received signal at the output port of the receiver antenna is amplified and correlated with the transmit signal, which results in a signal called beat signal. The beat signal contains information about the objects in the scene. Particularly, the delay in the reflected signal is translated to an instantaneous frequency difference between the transmitted and the received chirps.

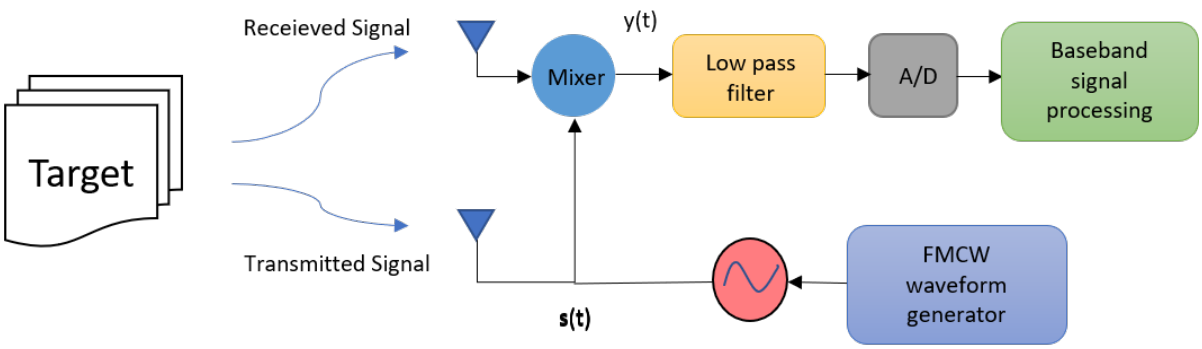


Figure 2.1: FMCW block diagram [4]

The basic idea in FMCW is to generate a linear frequency ramp. The transmit frequency for one ramp with bandwidth B and duration T between a given interval [−T/2, T/2] can be expressed as

$$f_T(t) = f_c + \frac{B}{T}t \tag{2.1}$$

The chirp signal of an FMCW radar system can be modeled as in the Equation (2.3)

$$s(t) = A_t exp(j(2\pi f_{min} + \pi K t^2)) \qquad 0 \leq t \leq T_r \tag{2.2}$$

f_{min} is the start frequency (λ_{max} is the corresponding wavelength) and A_t is the magnitude related to the transmit power. Suppose that there is only a single small object situated at the distance of R_0 to the radar but it is moving around R_0, which results in a time-varying distance to the radar. Let us denote the time-varying distance by $R(t) = R_0 + x(t)$ and $x(t)$ is a function represents the distance variations around R_0. Furthermore, the reflected wave off the object at the receiver is the delayed version of $s(t)$ with a delay of $t_d = 2R(t)/c$, which is the round-trip time of the wave and c is the speed of light. The beat signal, $y(t)$, can be expressed as follows in Equation (2.3)

$$y(t) = s(t)s^*(t - t_d) \qquad \text{and} \qquad y(t) \approx A_t A_r exp(j(\psi(t) + \omega_b t)) \tag{2.3}$$

$$\psi(t) = 4\pi\frac{R_0 + x(t)}{\lambda_{max}} \qquad \text{and} \qquad \omega_b = 4\pi\frac{KR_0}{c} \tag{2.4}$$

2.1.2 Range-Doppler Estimation

The signal generated after the mixer stage with signals from transmitted and reciever, the output signal is as follows in Equation (2.5). The fast time and slow time is described in Chapter A.

$$y(t) = A_t A_r exp(j(4\pi\frac{R_0}{\lambda_{max}} + 4\pi K\frac{R_0}{c}t + 4\pi K\frac{v}{c}t^2 + 4\pi\frac{v}{\lambda_{max}}t)) \tag{2.5}$$

2.1.2.1 Range Estimation

The range R_0, is determined as the round-trip time delay that the electromagnetic waves take to propagate to and from a given target. $R = (c\tau/2)$, where τ is the round-trip time delay in seconds and c is the speed of light.

The frequency spectrum of the signal computed over one modulation period will give us $4\pi K\frac{R_0}{c}$ as a main frequency component which is the beat frequency. The estimation of the beat frequency is usually based on the Fast Fourier Transform (FFT) algorithm which efficiently computes the Discrete Fourier Transformation (DFT) of the digital sequence. Consequently, by applying the FFT algorithm over one signal period, we can find the beat frequency (f_b) and thus the range to the target.

$$f_b = \frac{2BR_0}{cT_r} \qquad \text{and} \qquad R_0 = \frac{cf_bT_r}{2B} \tag{2.6}$$

2.1.2.2 Doppler Estimation

The Doppler D is determined based on the phenomenon called Doppler effect. Consider a scenario with relative motion between two cars. The time delay which is responsible for a frequency shift is given by the $\tau = (2(R \pm x(t))/c)$, where v is the radial velocity of the target and c is the speed of light in m/s. The Doppler shift $f_d = (\pm 2v/\lambda)$, where λ_{max} being the wavelength, tells whether the target is approaching the radar (positive) or moving away from the radar (negative). The configuration in [5] illustrates the range-Doppler estimation using FMCW pulses.

There is also a phase $(4\pi \frac{v}{\lambda_{max}})$ associated with f_b in Equation (2.5) which changes linearly with the number of sweeps. The change of the phase indicates how the frequency of the signal changes over consequent number of periods. This change is based on the Doppler frequency shift (f_d) which is the shift in frequency that appears as a result of the relative motion of two objects. The Doppler shift can be used to find the velocity of the moving object.

$$f_d = \frac{2 f_{min} v}{c} \qquad \text{and} \qquad v = \frac{f_d c}{2 f_{min}} \tag{2.7}$$

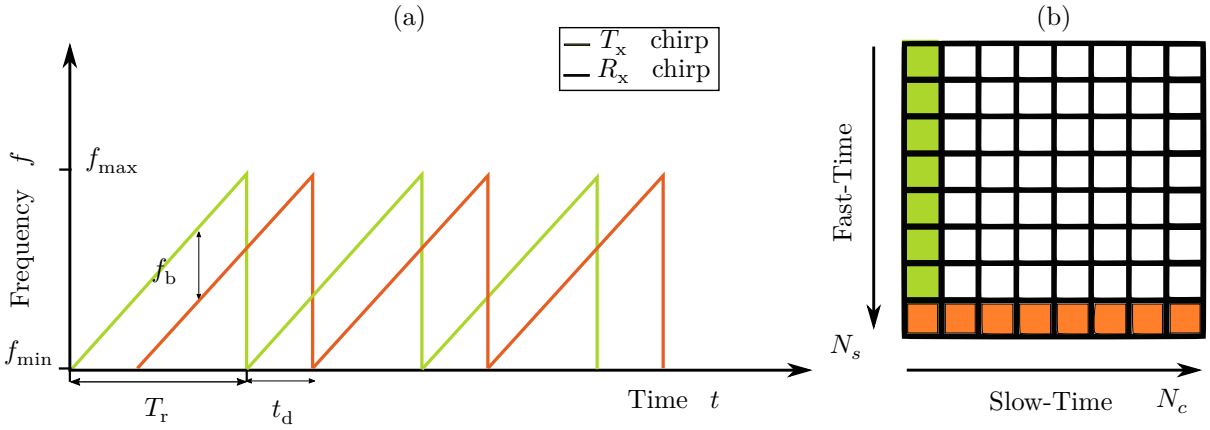


Figure 2.2: (a) Time domain representation of FMCW waveform and (b) 2D ADC output matrix over N_s x N_c

The Doppler shift of the signal can be found by looking at the frequency spectrum of the signal over n consecutive periods. The two-dimensional Fast fourier transformation (FFT) operation is summarized in Fig.2.2(b). The first step consists of 1D FFT along fast-time to obtain Range. Thereafter, a 1D FFT along slow-time for each range bin is performed and to obtain the Doppler frequency. After two dimensional FFT processing, the range-Doppler map contains range and velocity information of all targets is obtained.

2.1.3 Signal Detection

The need of signal detection is to identify the corresponding targets on the range-Doppler map. The map taken from a frame of radar simulation is shown below in Figure 2.3. Here the target is a cycle which is inscribed within a yellow elliptical region. The target in this case is already known in advance during generation of raw data. This situation is

not often seen in a practical scenario. Therefore, the detection of signal plays a key role in object detection.

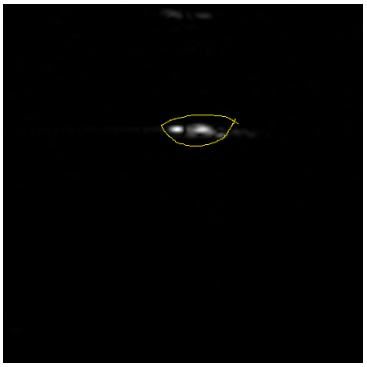


Figure 2.3: Visualization of signal in range-Doppler map

In automotive applications, targets of interest are dynamic. Noise and clutter in real environments are non-stationary random process varying with time. Applying a fixed threshold in real data will cause an enormous number of false alarms. The false alarms are not good in the field of object detection. Let us take a scenario of a single target with two clutter components on a range-Doppler map. If there are high false alarm detections present, the number of clusters in the scenario are detected to be three instead of the single target. In another scenario, it may also happen that extended target component is detected as the main target. These problems can bring errors in object analysis. As a result, different dimensions like range, doppler or azimuth are used to detect and identify them uniquely. The proposed detection pipeline uses range-Doppler (RD) maps as 2D input. Therefore, chirp configuration parameters are chosen considering δr_{min} as range resolution and δv_{min} as Doppler resolution.

Constant false alarm detectors are designed to maintain the probability of false alarm of the background noise or clutter at a fixed level. The CFAR estimates the background power level from the surrounding samples to set a detection threshold adaptively varies with the power level of the noise or the clutter. Heterogeneous environments, such as clutter transitions, and multiple targets, are the kind of environments that the CFAR detectors can operate in and still maintain the probability of false alarm.

2.1.4 CFAR Architecture

2.1.4.1 Probability of False Alarm

The probability of false alarm P_{fa}, is the chance that noise or clutter is mistaken by the CFAR detector as a target. Although noise and clutter distributions are continuous, usually amplitudes are very close to zero. No matter how high thresholds are set, there will always be a finite probability of random noise or clutter exceeding the threshold. So rather than eliminating false alarms all together (this would be impossible), the goal of CFAR algorithms is to reliably estimate the mean noise, and scaling the estimated mean by a CFAR multiplier to obtain the threshold set high enough to limit false alarm rate to a tolerably small rate.

2.1.4.2 Probability of Detection

The reference window is used to set the threshold and differentiate between target and noise. If signal power of a range cell lies below the threshold, a target is not detected and vice-versa. Inspite of setting an ideal threshold covering all the targets, there is always a chance that a target can be missed. The probability of detection P_D helps to characterize such scenarios. [6]

2.1.4.3 CFAR Loss

Although P_D is a good measure of CFAR performance, due to varying signal-to-noise ratio (SNR) it can have a high relative variance. CFAR loss is the metric that can be used to judge performance. The estimation of noise in a specific range bin is done with the help of neighbouring bins. The use of sliding window can be helpful for such task. The general architecture of the CFAR processor is explained as shown in Figure 2.4.

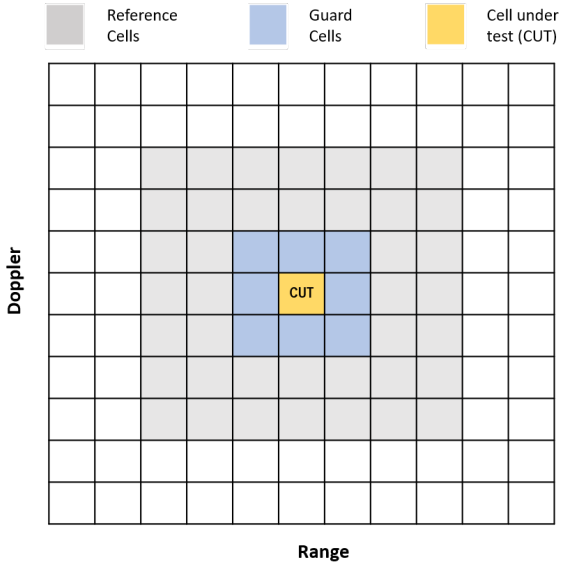


Figure 2.4: 2-D CFAR Processor

The CFAR detector contains four main elements as follows. These four elements assist the processor to set a varying threshold which follows the structure of the background noise or clutter.[6]

- cell under test (CUT)
- guard cells
- reference cells
- CFAR multiplier α

A CUT is the position where the threshold is going to be applied. If the target is in the CUT, then it should exceed the threshold and declare a detection. The cell under test

is located at the middle of the CFAR processor. Guard cells are located next to the CUT usually on both sides as shown in Figure 4.1. The samples contained in these cells are not used. The reason of having the guard cells is to eliminate any spill over from the target if the target extends to more than one sample. This provides a better estimation of the background noise. Reference cells are the outer cells of the CFAR processor. These cells estimate the threshold of the background noise. The more samples of the background noise in the reference cells, the better estimation of the threshold. [6]

CFAR multiplier (α) is also called CFAR constant. This constant is chosen based on the desired probability of false alarm to set the estimated threshold from the reference cells at the appropriate level. Usually the calculation to find the CFAR multiplier is complicated due to the complex relationship between the probability of false alarm and the CFAR multiplier. The data window could also be a two dimensional data set, such as in FMCW radar, which consists of range and doppler [6]. An example of two dimensional CFAR architecture applied in 2D data is illustrated in Figure 2.4.

In this thesis, detection pipelines are evaluated over two dimensional RD map, but this can be extended to 3D using azimuth information. The conventional target detection pipeline in radar signal processing is a two-stage process applied over RD map. Assuming each target of interest has a stronger reflection strength than clutter noise in the corresponding received signal, the first stage consist of peak detection in 2D-RD space. While the most common approach for peak detection is constant threshold based, it may result in a large number of false detections due to varying noise caused by interference, clutter, jamming etc. The preferred method to prevent these erroneous detections is a constant false alarm rate (CFAR) detector. It calculates the detection probability threshold $T = P_n$ for each bin by estimating the actual noise power from neighboring cells where T refers to detection threshold, is scaling factor and P_n is estimated noise power.

As an alternative, Ordered Statistic CFAR (OS-CFAR) is being used for detection of extended targets. This helps to select threshold using kth ordered data instead of averaging over the samples. The other types of CFAR implemented in this thesis are Greatest-of Cell-averaging CFAR (GOCA-CFAR), Smallest-of Cell-averaging CFAR (SOCA-CFAR) and Cell Averaging Statistic Hofele (CASH-CFAR) [7].

2.1.5 Clustering

Cluster analysis groups data objects based only on information found in the data that describes the objects and their relationships. The goal is that the objects within a group are similar to one another and different from the objects in other groups. The greater the similarity (or homogeneity) within a group and the greater the difference between groups, the better or more distinct the clustering. An entire collection of clusters is commonly referred to as clustering. The task of clustering is usually referred to a subjective quantity. Each methodology defines a set of rules to find the similarity measure.

Distance and similarity are the basis for constructing clustering algorithms. As for quantitative data features, distance is preferred to recognize the relationship among data. And similarity is preferred when dealing with qualitative data features. There are

clustering algorithms commonly categorized based on partition, hierarchy, density and distribution.

Partition models: These are iterative clustering algorithms in which the notion of similarity is derived by the closeness of a data point to the centroid of the clusters. k-means clustering algorithm is a popular algorithm that falls into this category. In these models, the number of clusters required at the end have to be mentioned beforehand, which makes it important to have prior knowledge of the dataset. These models run iteratively to find the local optima. [8]

Density Models: These models search the data space for areas of varied density of data points in the data space. It isolates various different density regions and assign the data points within these regions in the same cluster. Popular examples of density models are Density based spatial clustering applications with noise (DBSCAN) and Ordering Points To Identify Cluster Structure (OPTICS). [8]

2.2 Neural Networks

The human brain and vision is one of the best inventions that had ever happened. Consider, an image of cat and think how does the brain recognize it. The human brain has a hemisphere named primary visual cortex, which contains millions of neurons with billions of connections between each of the neurons. This task of identifying objects can sound easy to this intellectual system. The difficulty of pattern recognition comes into picture when such kind task has to be performed by a machine. In order to make machine understand this task, we need to build artificial neurons and connect them resulting in a network. One of such an artificial neuron is called a perceptron. A basic perceptron takes several inputs such as x_1,x_2,.., and produces a single output. [9]

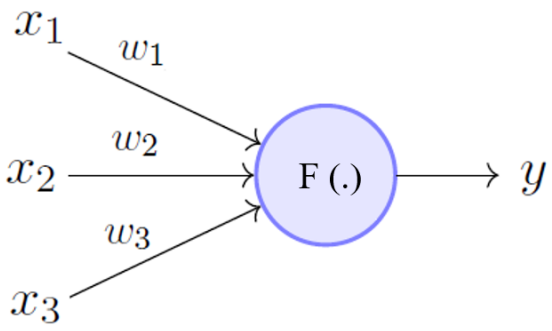


Figure 2.5: Perceptron [10]

A perceptron shown in Figure 2.5 is a mathematical model that makes the decisions by weighing up the evidence. For example, summing up the weighted inputs (linear combination of x, w) will produce a weighted net sum. The weighted net sum is then applied to an activation function $f(.)$ which then standardizes the value, producing an output y.

2.2.1 Activation Functions

Activation functions form an important feature of the artificial neural networks. They basically decide whether a neuron should be activated or not and whether the information that the neuron is receiving is relevant or should it be ignored.

The Activation function (AF) is the non linear transformation applied on the input signal. This transformed output is then sent to the next layer of neurons as an input. In literature, different activation functions such as sigmoid, tanh, Rectified Linear Unit (ReLU), leaky rectified linear unit are used to inculcate non-linear combination of features. In this thesis, the following functions are used in the implementation.

2.2.1.1 Sigmoid Function

The Sigmoid AF is often referred to as the logistic function. The function is given as in Equation (2.8)

$$f(x) = \frac{1}{(1 + e^{-x})} \tag{2.8}$$

The sigmoid function is used in the output layers of Deep learning (DL) architectures and can be applied to applications such as binary classification, logistic regression and other neural network domains. The problem with this function is its sharp damp gradient, slow convergence due to exponential function.

2.2.1.2 Rectified Linear Unit (ReLU) Function

The ReLU AF is the most widely used activation function used in the state-of-the-art architectures. It is a faster learning AF and offers better performance than sigmoid. The structure of this function resembles to a near linear function, which helps in achieving proper gradient descent. The function is given as follows in the Equation (2.9).

$$f(x) = \max(0, x) = \begin{cases} x_i, & \text{if } x_i \geq 0 \\ 0, & \text{if } x_i < 0 \end{cases} \tag{2.9}$$

The main advantage of rectified linear units is that, they guarantee faster computation as the function does not contain any exponentials or divisions. The other property is the squishing of sparsity in the hidden units from zero to maximum. The problem with the ReLU is that sometimes it acts fragile making some gradients die and therefore leads to null activation. To solve such a problem, a version of ReLU named Leaky ReLU is introduced in the literature to solve the problem of dying gradients.

2.2.1.3 Leaky ReLU function

The intuitive idea of leaky version of ReLU is by adding a negative slope to ReLU function. This is done in order to keep the weight update active during the entire propagation process. The parameter α is to prevent zero-gradients during training. The leaky ReLU is given by the following Equation (2.10) below.

$$f(x) = \alpha x + x = \begin{cases} x, & \text{if x} > 0 \\ \alpha x, & \text{if } x \leq 0 \end{cases} \tag{2.10}$$

2.2.1.4 Loss Functions

Loss function helps in optimizing the parameters of the neural networks. The objective is to minimize the loss for a neural network by optimizing its parameters (weights). The loss is calculated using loss function by matching the actual value (ground truth) and predicted value by a neural network. Then the principle of optimization is applied to update the weights of the neural network such that the loss is minimized.

There are several loss functions defined in the literature of deep learning. The commonly used ones are the Mean squared error (MSE) and Cross Entropy loss. In this report, the loss function used in the U-Net architecture is Focal loss, which is a type of categorical cross entropy loss function that has been described with details in the [11].

2.2.1.5 Optimizers

An optimization algorithm is very much required in order to train the neural network in given time and achieve better accuracy. The basic algorithm used in the convergence of neural networks is gradient descent. Gradient descent is a process in the backpropagation phase, where the goal is to continuously resample the model parameters in the opposite direction of the weights w, updating such value of weight until the cost function reaches global minimum.

There are different kinds of optimizers mentioned in the Keras documentation [12] used in training a neural network. Commonly used ones are the Stochastic gradient descent (SGD), Batch gradient descent (BGD). The optimizer that is used in this report is Adaptive Moment Estimation (Adam) optimizer, an extension of stochastic gradient optimization momentum algorithm as described detail in the paper [13]. The advantage of Adam optimizer is that it uses first-order gradients which requires less memory for computation. Adam shows marginal improvement over SGD with momentum as shown in the paper cited above, it adapts learning rate scale for different layers in case of convolutional neural networks. [13]

2.3 Basic Neural Network Architectures

There are several neural network architectures for different purposes. For example, in order to perform object detection, in the applications of machine vision the commonly used architecture is Convolutional Neural Network (ConvNet) as they yield good features after an input is passed through them. In a similar fashion, there are several networks that can be trained across different models in order to implement certain functionality.

2.3.1 Convolutional Neural Networks

Convolutional Neural Network (ConvNet) is a well-known deep learning architecture inspired by the natural visual perception mechanism of the human creatures. ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

ConvNets transform the original image layer by layer from the original pixel values to the final class outputs. Note that some layers contain parameters and others do not. In particular, the convolutional layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons). On the other hand, the pooling layers will implement a fixed function. The parameters in the fully connected layer will be trained with gradient descent so that the class outputs proper label associated to it.

2.3.1.1 Architecture

ConvNet is a sequence of layers, and every layer of the network transforms one volume of activations to another through a differentiable function. We use mainly three types of layers namely Convolutional Layer, Pooling Layer and Fully-Connected Layer (FC). These layers are stacked together to form a ConvNet structure.

2.3.1.2 Convolutional Layer

Convolutional layer is composed of a set of convolutional kernels (each neuron acts as a kernel). These kernels are associated with a small area of the image known as a receptive field. It works by dividing the image into small blocks (receptive fields) and convolving them with a specific set of weights (multiplying elements of the filter with the corresponding receptive field elements) as shown in Figure 2.6.

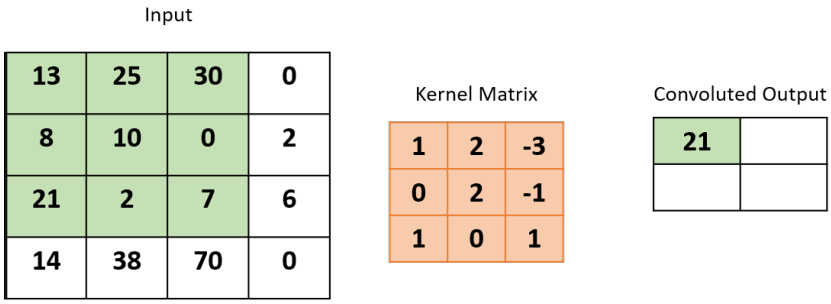


Figure 2.6: Convolution Layer

The operation of convolution is expressed as follows in the Equation (2.9), where $I_{x,y}$ is an input image at spatial location (x, y) and K_l^k is the l^{th} convolutional kernel of k^{th} layer.

$$F_l^k = I_{x,y} * K_l^k \tag{2.11}$$

The objective of the convolution operation is to extract the high-level features such as edges, corners from the input image. ConvNets need not be limited to only one convolutional layer. Conventionally, the first ConvLayer is responsible for capturing the low-level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the high-level features as well.

There are two types of results to the operation — one in which the convolved feature is reduced in dimensionality as compared to the input, and the other in which the

dimensionality is either increased or remains the same. This is done by applying valid padding in case of the former, or same padding in the case of the latter.

2.3.1.3 Pooling Layer

Pooling layer is responsible for reducing the spatial size of the convolved feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effective training of the model.

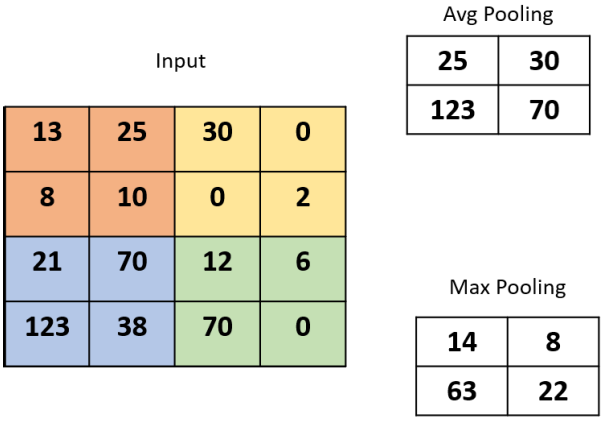


Figure 2.7: Pooling Layer

Pooling operation is a downsampling operation which is described in the Equation (2.11), where P_l represents the l^{th} output feature map. $F^l_{x,y}$ shows the l^{th} input feature map and $f_p(.)$ is the pooling function.

$$P_l = f_p(F^l_{x,y}) \tag{2.12}$$

There are two types of pooling, Average pooling and Max pooling. Max pooling returns the maximum value from the portion of the image covered by the kernel. On the other hand, Average pooling returns the average of all the values from the portion of the image covered by the kernel.

2.3.1.4 Fully Connected Layer

As their name prompts, Fully connected (FC) layers are layers where each neuron in a layer is connected to each neuron of the previous layer. Their activations can be seen simply as a matrix multiplication enhanced by a bias. The purpose of FC layers is to take the high-level feature map as an input and return a classification vector as an output. Each value in the output refers to one class occurrence. For example, The length of the output vector is n where n is the number of classes. FC layers are not so hard to train to use non-linear combinations of features in input which is widely used whereas the combinations of high-level features are the things we are looking for.

For example, if we are looking for a cycle, the last layer output will have high values in the neurons that represent things like a wheel, two tires or a cycle model; if we are

looking for a car, we will most probably not be interested in any of these features. Using popular softmax classifier, the output is a vector of probabilities representing each class. Other classifiers like Support vector machine (SVM) can also be used.

2.3.1.5 Batch Normalization

Batch normalization is used to address the issues related to internal covariance shift within feature maps. The internal covariance shift is a change in the distribution of hidden units values, which can slow down the convergence (by forcing learning rate to small value) and requires careful initialization of parameters. It enables user to use high learning rates without having much attention over initialization.

2.3.2 ResNet

ResNet stands for Residual Networks. The need for ResNet [14] arose due to the problem that when deeper networks starts converging, a degradation problem occurs. With the network depth increasing, accuracy gets saturated and then degrades rapidly.

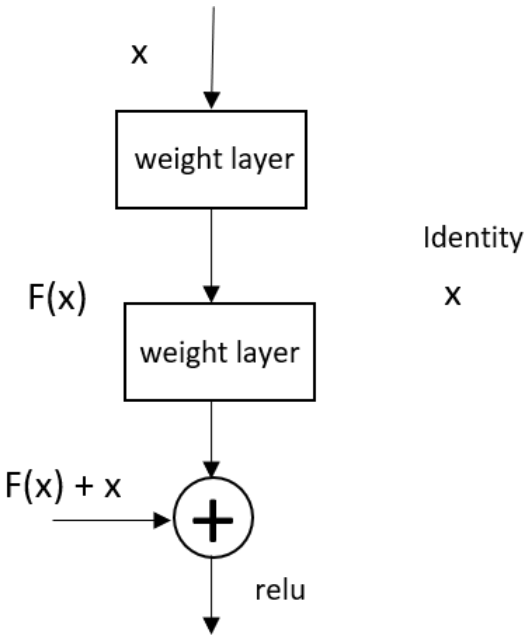


Figure 2.8: Residual block [14]

ResNet-50 is a deep ConvNet, with 152 layers. As the name suggests it contains 50 layers. Instead of hoping every few stacked layers directly fit a desired underlying mapping, they explicitly let these layers fit a residual mapping. The formulation of $F(x)+x$ can be realized by feedforward neural networks with shortcut connections. Shortcut connections are those skipping one or more layers shown in Figure 2.8. The shortcut connections perform identity mapping, and their outputs are added to the outputs of the stacked layers. By using the residual network, there are many problems which can be solved such as optimization and good accuracy from increased depth.

2.4 Image Segmentation

Image segmentation can be basically classified into two types, Semantic segmentation and Region Instance segmentation.

2.4.1 Semantic Segmentation

Pixel-based semantic segmentation is process of linking each pixel in an image to a class label. These labels can be a car, pedestrian, a bicycle and other major components that form a part of autonomous driving. One of the approaches to semantic segmentation is to classify patches. Each pixel is assigned to a class based on the path of the image around it.

This kind of segmentation is an image classification task done at pixel level. The labels in this kind of segmentation are class-aware. For example, in an image that has many cars, this segmentation will label all the objects as cars.

Figure 2.9: Semantic Segmentation

A general semantic segmentation architecture can be broadly thought of as an encoder network followed by a decoder network.

- The encoder is usually is a pre-trained classification network like normal deep convolutional network followed by a decoder network

- The task of the decoder is to semantically project the discriminative features (lower resolution) learnt by the encoder onto the pixel space (higher resolution) to get a dense classification

2.4.2 U-Net : ConvNet for Semantic Segmentation

We see that deep convolutional networks such as deep ConvNets have already outperformed the state of the art in many visual recognition tasks. The problem of using ConvNets lies in the size of training sets and network with millions of parameters.

The typical use of convolutional networks is on classification tasks, where the output to an image is a single class label. But in case of range-doppler information extraction, we need to assign class label to each pixel and determine if it is target or not. The architecture looks like a '*U*' in Figure 2.10 which justifies its name. The architecture basically consists of three sections: The contraction, bottleneck and the expansion section. The contraction section is made of many blocks. Each block takes an input that is applied on two 3x3 convolution layers followed by a 2x2 max pooling. The number of kernels or feature maps after each block doubles so that the network can learn the complex structures effectively. The bottommost layer called the bottleneck mediates between the contraction layer and the expansion layer. It uses two 3x3 convolution layers and 2x2 up-convolution layers.

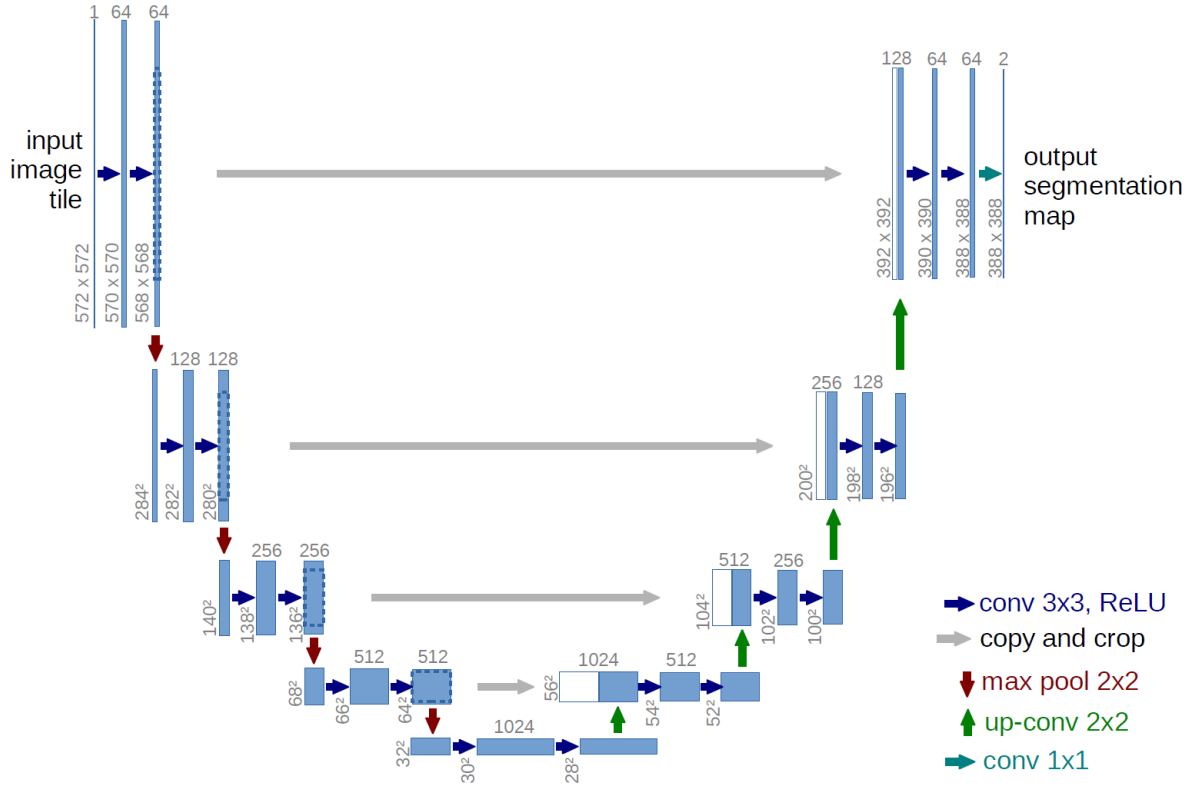


Figure 2.10: U-Net Architecture [15]

The heart of the architecture lies in the expansion section. Similar to the contraction layer, it also consists of several expansion blocks. Each block passes the input to two $3x3$ convolution layers folowed by a $2x2$ upsampling layer. Also after each block, number of feature maps get halved to maintain the symmetry. However, every time the input gets also appended by the feature maps of the corresponding contraction layer. This append operation ensures that the features learnt while contracting the images will be used to

reconstruct it. The number of expansion blocks are same as the number of contraction blocks. At the end, the resultant mapping passes through another 3x3 convolution layer with the number of feature maps equal to the number of segments desired.

The loss function uses the weighting scheme for each pixel, such that there is a higher weight at the border of the segmented objects. This scheme helps to segment the cells in a discontinued fashion such that the individual regions of interest can be easily identified within the segmentation map. The pixel-wise softmax is applied on the resultant image followed by a cross-entropy loss function. Based on the application of segmentation, different loss functions can be used.

The proposed architecture in the [15] works with training images and yield more precise segmentations. The main idea is to supplement a usual contracting network by successive layers, where pooling operators are replaced by upsampling operators. Hence, these layers increase the resolution of the output. In order to localize, high resolution features from the contracting path are combined with the upsampled output. A successive convolution layer can then learn to assemble a more precise output based on this information. The architecture was so modified such that the filter sizes are set according he target size. This helps the network to segment on to a narrrower level and give the resultant output.

2.4.3 Instance Segmentation

Instance segmentation is a process of identifying each object instance of each pixel for every known object within an image. The type of instance segmentation is region-based. The example of instance segmentation is shown in Figure 2.11 below.

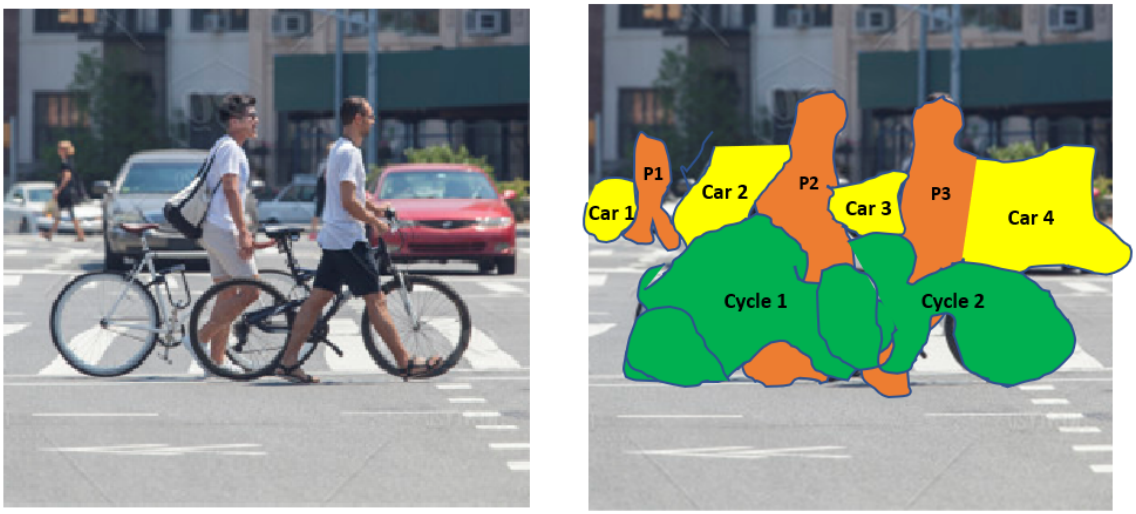


Figure 2.11: Instance Segmentation

It can be referred to object detection in combination with semantic segmentation. On the other hand, the labels in this type of segmentation are instance-aware. Here each object has its own characteristics after segmentation and hence gets an each unique label associated to it.

2.4.4 Mask R-CNN : ConvNet for Instance Segmentation

Deep convolutional networks such as Faster R-CNN [16] and Mask R-CNN [17] have already outperformed in the state-of-the-art object detection in many visual recognition tasks. This thesis considers the object detection network Mask Region based convolutional neural network (Mask R-CNN) for the instance segmentation.

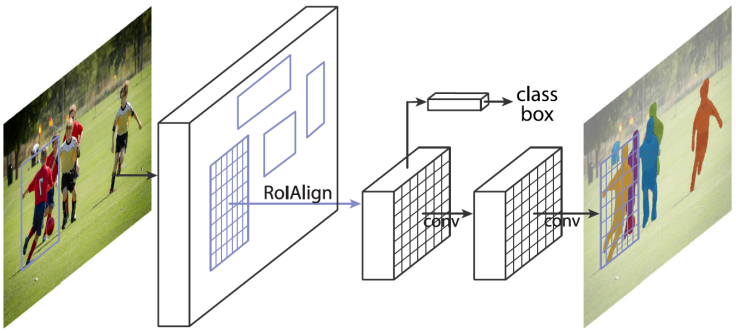


Figure 2.12: Mask R-CNN framework [17]

Mask R-CNN (Region based convolutional neural network) is a two stage framework: the first stage scans the image and generates proposals (areas likely to contain an object). And the second stage classifies the proposals and generates bounding boxes and masks. [17]

2.4.4.1 Network backbone

This is a standard convolutional neural network (typically, ResNet50 or ResNet101) that serves as a feature extractor. The early layers detect low level features (edges and corners), and later layers successively detect higher level features (car, pedestrian, cycle).

In the original paper [17], the authors used backbone as a ResNet architecture with Feature Pyramid Network (FPN). This feature map becomes the input for the following stages. The implementation of Mask R-CNN can be either configured with a ResNet-50, ResNet-101 or any custom backbone. [17]

2.4.4.2 Feature Pyramid Network

FPN creates two pathways in order to obtain feature maps at different scales. One pathway is bottom-up, other one is top-down pathway. The two pathways combined with skip connections is shown in Figure 2.13. [17]

The features extracted in bottom-up pathway are on a higher level with semantically strong features and decrease in spatial resolution. The bottom layers hence perform worse when detecting small objects. Therefore, the top-down pathway is used where the upsampling is performed on feature maps from higher level. This two-way approach brings good results with the help of skip connections.

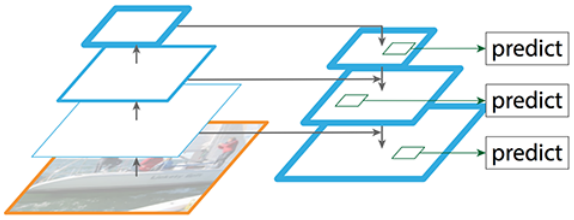


Figure 2.13: FPN Network with two pathways [18]

FPN improves the standard feature extraction pyramid by adding a second pyramid that takes the high level features from the first pyramid and passes them down to lower layers. By doing so, it allows features at every level to have access to both, lower and higher level features.

2.4.4.3 Region Proposal Network

The Region Proposal Network (RPN) is a lightweight neural network that scans the image in a sliding-window fashion and finds areas that contain objects, where the number of maximum possible proposals for each location is denoted as k. The reg layer has $4k$ outputs encoding the coordinates of k boxes, and the cls layer outputs $2k$ scores that estimate probability of object or not object for each proposal as shown in Figure 2.14. [17]

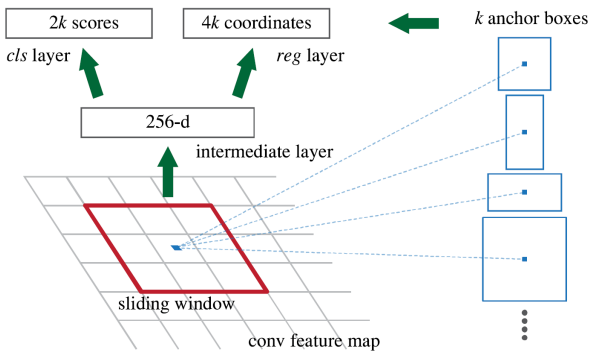


Figure 2.14: RPN Network with layers and anchor boxes [16]

The center of the sliding window at each position is called anchor. For each anchor, a set of predefined bounding boxes are distributed over the image area. These boxes are used as a reference for prediction of location of different objects. The theoritical number of anchor boxes generated per anchor is usually nine. In practice, there can be several anchors of different sizes and aspect ratios, and they overlap to cover as much of the image as possible. The RPN scans over the backbone feature map. This allows the RPN to reuse the extracted features efficiently and avoid duplicate calculations.

The RPN generates two outputs for each anchor:

- Anchor Class: One of two classes: foreground or background. The foreground class implies that there is likely an object in that box

- Bounding Box Refinement: Refinement of bounding box over the object

The RPN estimates a delta (percentage change in x, y, width, height) to refine the anchor box to fit the object better. Using the RPN predictions, top anchors are picked which are likely to contain objects and refine their location and size. If several anchors overlap too much, the ones with the highest foreground score are preserved and the rest are discarded (using non-maximum suppression). After suppression, the final proposals (regions of interest) are passed to the next stage.

2.4.4.4 ROI Align

This stage runs on the regions of interest (ROIs) proposed by the RPN. And just like the RPN, it generates two outputs for each ROI:

- Class: The class of the object in the ROI. Unlike the RPN, which has two classes (FG/BG), this network is deeper and has the capacity to classify regions to specific classes (person, car, cycle). It can also generate a background class, which causes the ROI to be discarded.
- Bounding Box Refinement: Very similar to how it is done in the RPN, and its purpose is to further refine the location and size of the bounding box to encapsulate the object.

There is small problem to be solved before the next processing to be continued. Classifiers do not handle variable input size very well. They typically require a fixed input size. But, due to the bounding box refinement step in the RPN, the ROI boxes can have different sizes. That is where ROI pooling comes into picture.

ROI pooling refers to cropping a part of a feature map and resizing it to a fixed size. It is similar in principle to cropping part of an image and then resizing it (but there are differences in implementation details). The extension of Faster R-CNN is made by he introduction of ROI Align layer by converting the anchor boxes to same size. This layer overcomes the higher complexity of processing generated anchors of different sizes over the feature maps. [17]

The implementation in [17] suggests the above method ROIAlign in Figure 2.15, in which the feature map is sampled at different points and a bilinear interpolation is applied. This interpolation improves the accuracy of predicted masks. [17]

2.4.4.5 ROI Classifier and Bounding Box Regressor

After the ROI pooling, the output is then transferred to creation of segmentation mask. The mask branch is a convolutional network that takes the positive regions selected by the ROI classifier and generates masks for them.

The generated masks are low resolution: For Example, 28x28 pixels. But they are soft masks, represented by float numbers, which hold more details than binary masks. The small mask size helps keep the mask branch light. During training, the ground-truth

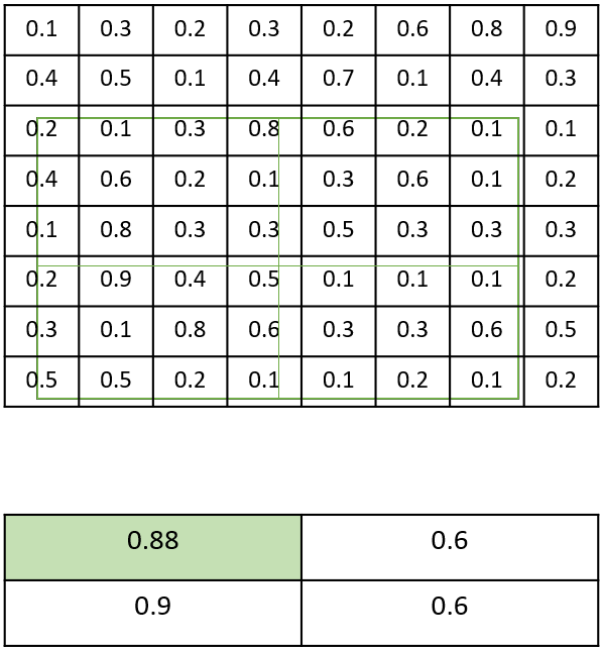


Figure 2.15: ROI Align with bilinear interpolation mapping

masks are scaled down to 28x28 to compute the loss, and during inferencing the predicted masks are scaled back to the size of the ROI bounding box and that gives the final masks, one per object. The regions already generated from ROI Align is used for the generation of predicted masks. The mask prediction is performed in parallel to the bounding box classifier and regression layer. [17]

2.4.4.6 Training Heads

In the head architecture, there was the most important change to allow the instance segmentation including the mask branch in parallel with branches for classification and bounding box regression. The mask branch is a pixel-to-pixel FCN predicting a mask individually for each ROI, where the mask is a binary matrix of ones (object location) and zeros (elsewhere).

3 Requirements Analysis

3.1 Architecture

The architecture for the following thesis is described below in Figure 3.1. In Figure 3.1(a), the initial pre-processing is performed over the input signal and range-doppler maps are generated. This serves as an input to the traditional target detection using the principles of signal processing and as well as for the proposed deep learning architecture. [19]

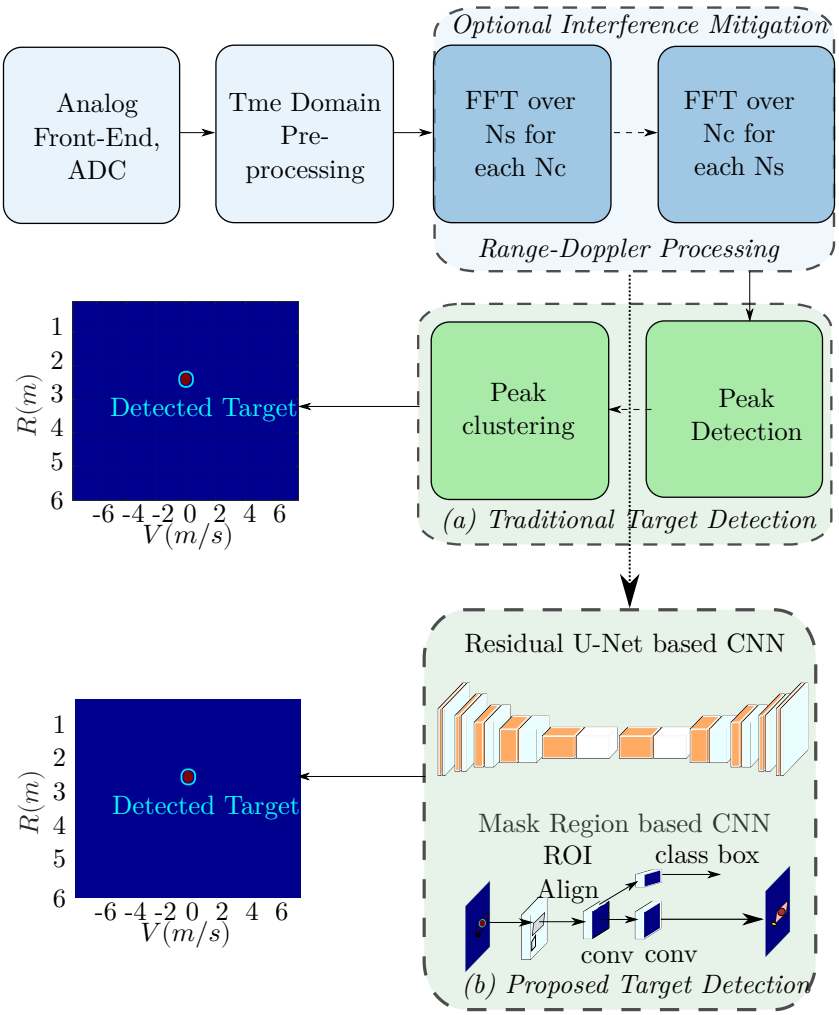


Figure 3.1: (a) Traditional 2-stage target detection approach and (b) proposed deep learning based single stage target detection [19]

3.2 Data Preparation

The data is prepared with ideal software simulator and realistic simulation scenario for detection with automotive radar sensors. A radar sensor is placed on a setup bench and various objects are captured by the object from several different directions. As objects we selected a single instance of each of the following three different objects namely car, bicycle and pedestrians. Although these objects are visually easy to distinguish, they pose a greater challenge for detection algorithms when working in the radio frequency spectrum. Furthermore, when the scene is static and thus does not allow identifying the objects solely through the doppler spectrum. For radar, dynamic objects with different doppler spectra are easier to classify, due to their micro Doppler signature, but harder to record and annotate. For example, a moving person with arms and toes movement may have different signatures in the doppler domain. For every object, the range R, the relative radial velocity v, and the Direction of Arrival (DOA) θ are measured.

In this thesis, two sources are used in order to generate input data. The first source is the radar simulator built in the MATLAB using drive scene scenario and signal processing toolbox and the another set of data is realtime measurements generated from radarbook. The software requirements are listed in the Chapter A

3.2.1 Signal Parameters

The choice of the radar signal processing pipeline and the choice of the FMCW parameters are important for accurately estimating the range and doppler components of the target. The parameters can also be altered and simulations with different specifications can be generated. The details of the chosen FMCW signal parameters are described in Table 3.1.

Radar Parameters	Values
Cut-off frequency, f_0	77 GHz
Speed of light, c	3 x 10^8 m/s
Bandwidth (B)	150MHz
Sampling frequency, fs	187 MHz
Number of samples/chirp, N_s	188
Number of chirps/frame, N_c	128
Number of Tx antennas, N_{Tx}	1
Number of Rx antennas, N_{Rx}	1
Chirp time, T_c	10 μs

Table 3.1: FMCW signal parameters in the simulator

The details of the chosen FMCW signal parameters used in the real-time measurments are described in Table 3.2. However, the signal parameters are chosen to meet the hardware requirements.

Radar Parameters	Values
Ramp start frequency, f_{min}	76 GHz
Ramp stop frequency, f_{max}	77 GHz
Bandwidth (B)	1GHz
Sampling frequency, fs	1 MHz
Number of samples/chirp, N_s	256
Unambiguous range, R_{max}	50 m
Range resolution, δ_R	15 cm
Chirp time, T_c	128 μs
Chirp repetition time T_{PRT}	256 μs
Number of chirps/frame, N_c	64
Doppler resolution, δ_V	0.15 m/s
Unambiguous velocity, V_{max}	3.9 m/s
Number of Tx antennas, N_{Tx}	1
Number of Rx antennas, N_{Rx}	8
Range FFT points, $NFFT_r$	1024
Doppler FFT points, $NFFTd$	512

Table 3.2: FMCW signal parameters used for measurements

3.2.2 Data from Radar Simulator

Driving Scenario Radar Simulation
The framework from Autonomous driving toolbox in MATLAB lets users create any driving scenarios with the DrivingScenarioDesigner App. It enables creation of raw radar data for the ego vehicles front radar. The Figure 3.2 shows a snapshot of the designer app. This feature enables users to directly obtain raw data without measurements.

- Open the DrivingScenarioDesigner using MATLAB
- Create road and actor models using the interface.
- Configure sensors on the objects and use these sensors to simulate detections of actors and lane boundaries in the scenario
- The objects can be configured with parameters such as initial position, velocity and waypoints
- After the configuration, save the scenario and export it to your workspace

In addition to the above designer application, the signal processing for pre-processing is added into the simulator using user-defined functions. This enables us tho get the RD maps with better information. As discussed in the beginning, this tool is easy to generate raw data for testing. The limitation of this tool is that it does not account many real-world parameters like noise from the hardware, environment and other factors. The ideality of the tool makes it easy but does not fit the real-time scenario.

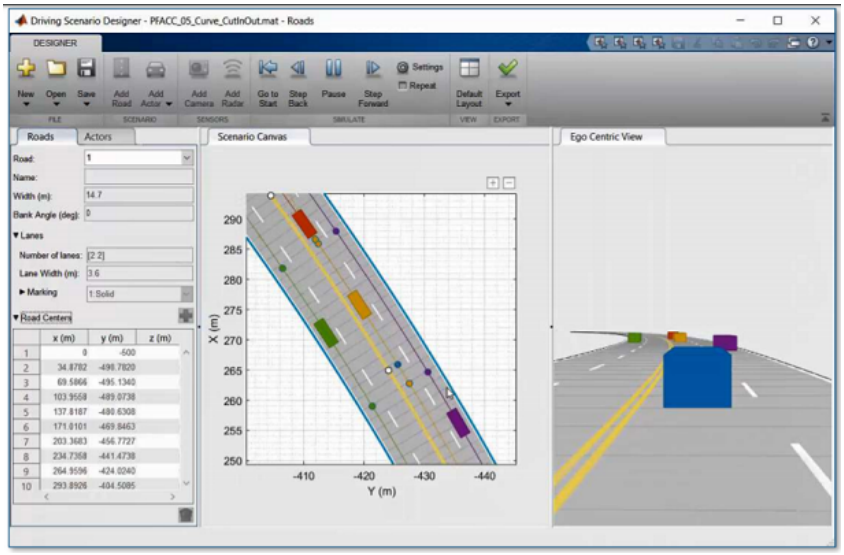


Figure 3.2: Driving Scenario Designer MATLAB

3.2.3 Data from Real-time Measurements

In the previous subsection, we have seen that the synthetic data can be generated using a software tool. Due to its limitations, the following section explains the data generation with real-time radar sensor. The radarbook hardware in Figure 3.3 is a versatile signal processing platform, developed to operate modern continuous wave radar systems with multiple receive and transmit channels in real-time. A standardized RF connector allows the control of different radar frontends from a single baseband platform.

Features	Usage
MIMO Frontend	Infineon 77 GHz
Signal processing	Rate reduction, FFT, Reconfigurable
Available frameworks	Sampling, Range-Doppler, Beamforming
Interfaces supported	LAN, WLAN, USB

Table 3.3: Radarbook Features

It operates over a frequency of 77 GHz with four transmit (T_x) and eight recieve (R_x) antennas. Morever it supports programming in $\#C$ and Octave/MATLAB. The features of the radarbook is listed in Table 3.3

Additional details in detail about the radarbook is available in the [20]. The setup must be connected to power supply and to the Linux/Windows machine with USB. Once the device is installed and is in ready to use, the link cited above contains information about how to establish connection to the board and initiate the device. When the program is executed, the sensor emits the radiations and as a result the reflections as well as the targets in the test environment are generated over the doppler map. The pre-processing enables to eliminate some clutter and gives filtered output at the component end.

Figure 3.3: Radarbook

3.2.4 Instance Labeling

VGG Image Annotator (VIA) is a manual image annotation tool to define and describe regions in an image. The region shape can be rectangle, circle, ellipse, polygon, polyline, or a single point. Region descriptions can be plain text or a set of predefined options presented to manual annotators as checkbox, dropdown menu, radio buttons or image list. This tool [21] supports bulk update of annotations corresponding to a large set of images.

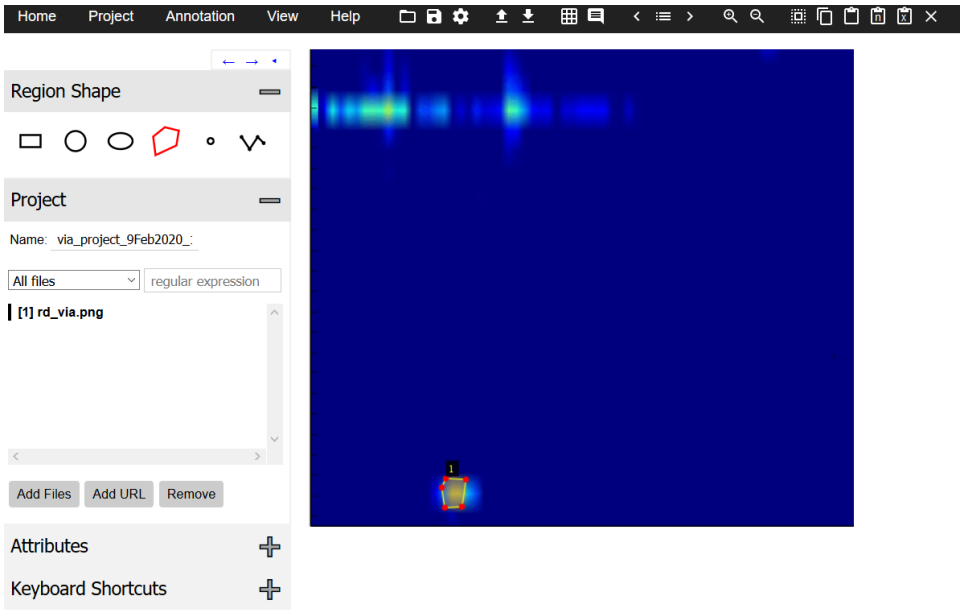


Figure 3.4: VIA Annotator

The steps of creating labels manually is possible with the tool shown in Figure 3.4 are as follows.

- First click on 'Add Files' button, then upload your image from local machine
- The image is loaded and click on the desired shape with the given shape

- Hover the mouse with right click on the target and capture the path around it and click 'Enter'
- After the masks are created, click on File Menu -> Export Annotations as (.json)

The project created above can be saved at any point of time. This saved file can be loaded on the (.html). If further more masks are needed to be created, this can be done with the previously saved file. There is also the possibility of importing the saved (.json) file, make changes and export it again back to the required file. [21]

4 Detection Algorithms

4.1 State of the art

The major challenge in object detection is to decide if a peak in the spectrum corresponds to a potential target or not. Comparing the frequency spectrum to a fixed threshold value could work good for an ideal spectrum. In a real time scenario, the presence of noise with unknown power may cause many false alarms if the threshold value was chosen too low. Conversely, if it is set too high, fewer objects will be detected. Thereby, a new set of adaptive threshold detection namely Constant False Alarm Rate (CFAR) has been introduced to mitigate above problems.

4.1.1 Effect of Unknown Interference

The false alarm probability of an unknown interference modeled by square detector in a white Gaussian noise environment with unnormalized samples is given by Equation (4.1).

$$P_{fa} = e^{\frac{-T}{\beta^2}} \tag{4.1}$$

The threshold T is proportional to the noise power. Accurate setting of threshold requires appropriate knowledge of noise power β^2. This interference noise mostly is unknown and varies with different factors. Small changes in the interference power would lead to changes in threshold as a result in the false alarm probability. In order to avoid such a problem, a radar system designer prefers to have a constant false alarm rate.

4.1.2 General Concept of CFAR

The detector uses range-Doppler processing as discussed in the sub-section 2.1.2. It looks for target in each available sample. The cell to be tested in labelled as x_i as shown in Figure 4.1 below.

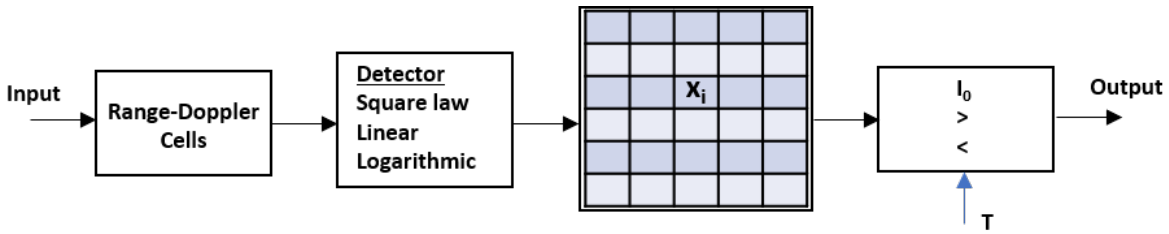


Figure 4.1: CFAR block diagram

The CFAR processing is based on two major assumptions

- Homogeneous Interference is exhibited between cells.
- Neighboring cells contain no targets

4.1.3 Cell Averaging CFAR

Cell-averaging CFAR (CA-CFAR) is obtained by the maximum likelihood estimate resulting in the average of available samples. The threshold is a multiple of the estimated interference power. The principle of CA-CFAR algorithm is shown in the figure below. Averaging over the reference windows consisting of N values, presents the background noise estimation of this CFAR algorithm.
The cells adjacent to the cell under test are called the guard cells. The resultant averaging yield is defined as shown in the Equation (4.2)

$$Z = \frac{1}{N}\sum_{i=1}^{N} X_i \tag{4.2}$$

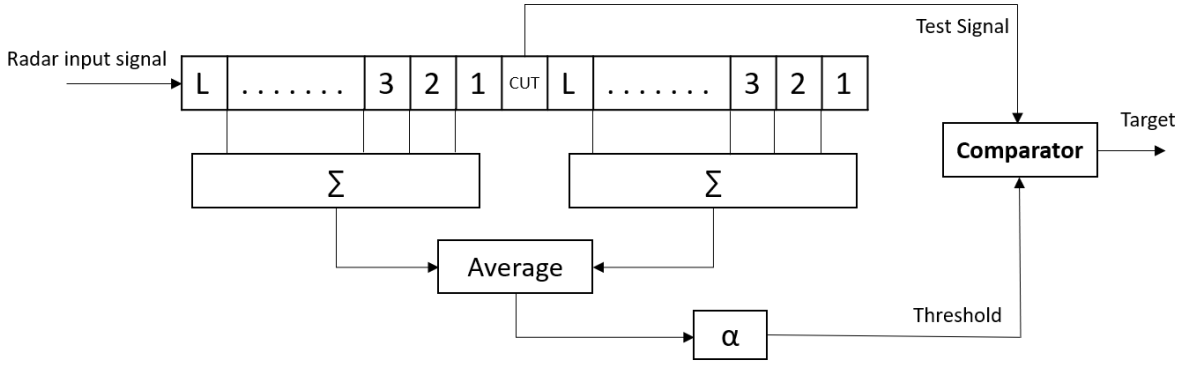


Figure 4.2: CA-CFAR [22]

4.1.3.1 Calculation of the threshold T

To get an adaptive threshold value for comparison, a scaling factor has to be found in order to properly scale the result of the averaging of the above obtained value of Z. For a given probability of false alarm P_{fa} and a fixed window size N, a constant threshold T is determined. The input of CFAR algorithms is exponentially distributed if a square law detector is used.

Assuming that the interference is independent identically distributed (i.i.d.), the probability density function (pdf) of a cell x_i is given by (4.3)

$$p_{x_i}(x_i) = \frac{1}{\beta^2} e^{\frac{-x_i}{\beta^2}} \tag{4.3}$$

Using the standard results from probability and Equation (4.4), we can write the pdf of z_i and T as follows

$$p_{z_i}(z_i) = \left(\frac{N}{\beta^2}\right) e^{\frac{-Nz_i}{\beta^2}}, \quad p_T(T) = \frac{N}{\beta^2}^N \frac{(T^{N-1})}{(N-1)!} e^{\frac{-NT}{\beta^2}} \tag{4.4}$$

The P_{fa} observed with the estimated threshold is $exp(\frac{N}{\beta^2})$. As this value is a random variable, the expectation of the P_{fa} is taken and then standard integrals is computed. After some algebraic manipulations as followed in [6], the final result is given by

$$P_{fa} = \left(1 + \frac{T}{N}\right)^{-N} \tag{4.5}$$

The scaling factor T is constant for fixed values of P_{fa} as well as N, and it can be therefore in prior calculated. With the multiplication of T, the determined average Z is an adaptive threshold for object detection, which satisfies the specified probability of false alarm P_{fa} for a given Additive gaussian white noise (AWGN) channel. The CA-CFAR detection outputs a boolean, stating if a cell exceeds the threshold or not.

4.1.3.2 Limitations of CA-CFAR

The detection procedure in CA-CFAR is not designed for multiple target detection. Other objects present in the reference window distort the noise estimation and increase the threshold value as a consequence. Therefore, an algorithm which can solve such a kind of scenario is discussed in the sub-section below.

4.1.4 Ordered Statistics CFAR

In contrast to the CA-CFAR procedure, which uses all signal amplitudes in the reference window to determine a threshold, the Ordered Statistic CFAR (OS-CFAR) algorithm only selects a single amplitude value for estimation of the interference. The selection of the the amplitude is based on the rank-procedure. The reference window data samples are sorted in the ascending numerical order and stored into list called ordered list. This kth element of the ordered list is called kth ordered statistic. This ordered statistic is selected as representative of the interference level and a threshold is a multiple of it.

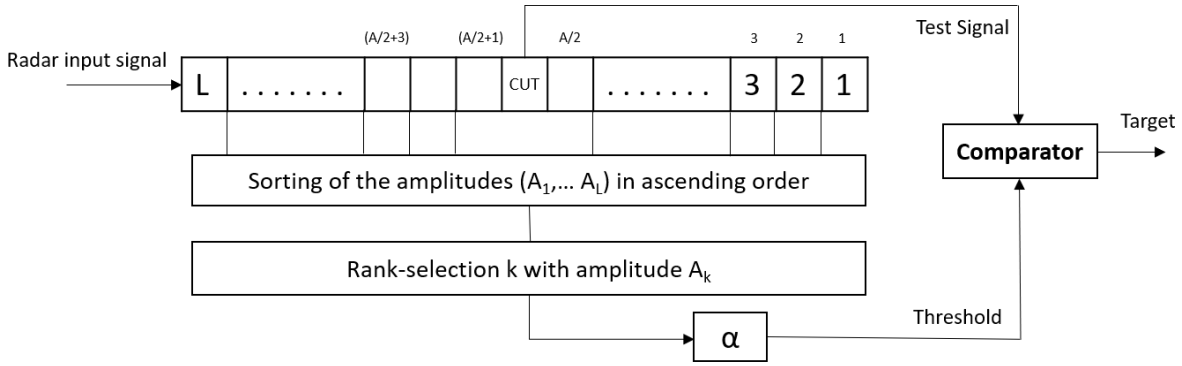


Figure 4.3: OS-CFAR [22]

Here the important note is that threshold depends on all of the given data, since all of the samples are required to determine the kth largest of all the samples.

$$P_{fa} = k\binom{N}{k}\frac{(k-1)!(T_{OS}+N-k)!}{(T_{OS}+N)!} \tag{4.6}$$

The OS-CFAR algorithm comprises of a shift-register containing A storage cells, and the cell under test. The amplitudes are passed as an input to the sorting algorithm as described above. In this rank selection process, we select the kth ordered statistic multiplied with α and compare the input values to this threshold.

4.1.4.1 Calculation of the threshold T

At first, a suitable value for k in the ordered statistic must be found. The factor T can be calculated using the below Equation (4.6). For a given probability of false alarm and kth value of an ordered-statistic array of exponentially distributed values [6]. Using the plots from the Figure 3.3, the ordered statistic is chosen approximately as $k = 3N/4$.

4.1.4.2 Limitations of OS-CFAR

The limitation of Ordered statistic CFAR is the processing power. It requires high computing power due to the presence of sorting algorithm.

4.1.5 Smallest of Cell Averaging CFAR

Smallest-of Cell-averaging CFAR (SOCA-CFAR) also known as least of CA-CFAR. This technique is intended to combat masking effect casued by interference among CFAR reference cells.

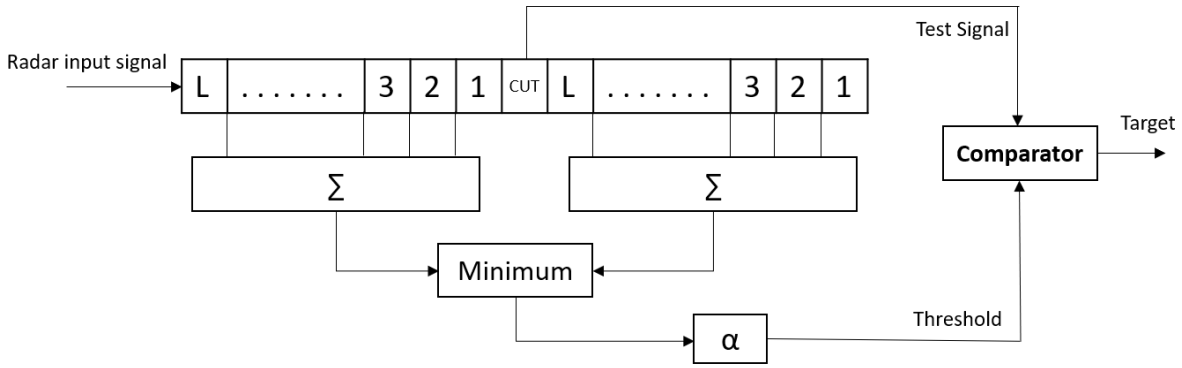


Figure 4.4: SOCA-CFAR [22]

In a N-cell SOCA method, the lead and lag windows are averaged separately to make two different estimates such as β_1 and β_2. Each of these beta-values are computed over $N/2$ reference cells. The threshold is then computed by finding the minimum of both the above estimates.

$$P_{fa} = \frac{1}{2}\left(2 + \frac{\alpha_{SO}}{N/2}\right)^{-N/2}\left[\sum_{k=0}^{\frac{N}{2}-1}\binom{\frac{N}{2}-1+k}{k}\left(2 + \frac{\alpha_{SO}}{N/2}\right)^{-k}\right] \tag{4.7}$$

The inference can be drawn as follows. If the interfering target is present in one of the two windows, it will raise the interference power estimate in that window. Thus, lesser the amplitude of two estimates, there is high probability that it is the representation of interference level. This level can be used to set the threshold in the detection.

4.1.5.1 Limitations of SOCA-CFAR

Thresholding is often ineffective. The reason for this is that the mean-value calculation inevitably raises the threshold in the neighbourhood of targets. Consequently, far spaced targets can mask each other, particularly if a large target is located close to a small or extended target.

Furthermore, because of the minimum selection process, the SOCA threshold cannot immediately follow an abrupt rise or fall in clutter level, e.g. the front of a thunderstorm. Therefore the threshold will lead in time in case of a steep rise in clutter level and will lag in case of a steep fall. Both these effects can result in target losses.

4.1.6 GOCA-CFAR

Greatest-of Cell-averaging CFAR (GOCA-CFAR) also known as least of CA-CFAR. This technique is intended to combat masking effect casued by interference among CFAR reference cells as shown below.

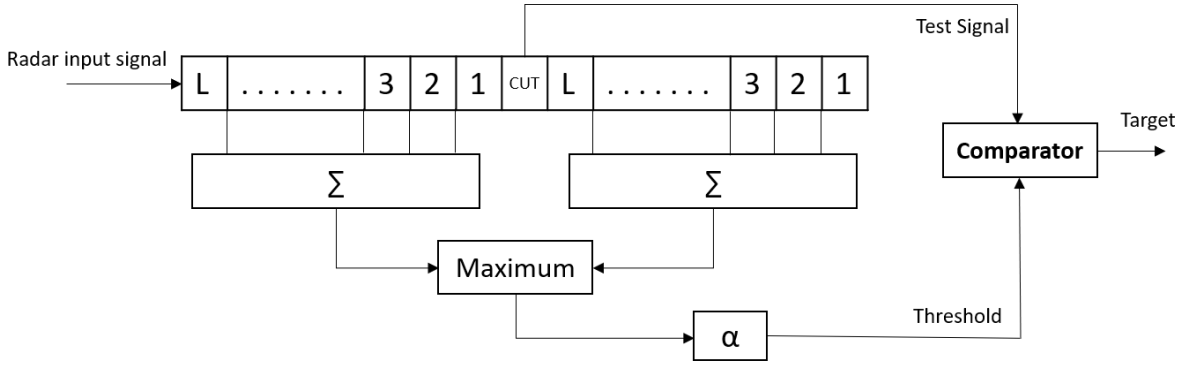


Figure 4.5: GOCA-CFAR [22]

In a N-cell GOCA method, the lead and lag windows are averaged separately to make two different estimates such as β_1 and β_2 as in the GOCA-CFAR technique . Each of these beta-values are computed over $N/2$ reference cells. The difference lies here where the threshold is then computed by finding the maximum of both the above estimates.

$$P_{fa} = \frac{1}{2}\left(1+\frac{\alpha_{GO}}{N/2}\right)^{-N/2} - \left(2+\frac{\alpha_{GO}}{N/2}\right)^{-N/2}\left[\sum_{k=0}^{\frac{N}{2}-1}\binom{\frac{N}{2}-1+k}{k}\left(2+\frac{\alpha_{GO}}{N/2}\right)^{-k}\right] \tag{4.8}$$

GOCA used in nonhomogeneous areas makes that the false alarms drop but also the probability of detection is slightly reduced. Therefore, this technique shows a good advantage in transition areas of big power level difference but at the same time drops the sensitivity in homogeneous background noise situations.

4.1.6.1 Limitations of GOCA-CFAR

Thresholding is often ineffective. The reason for this is that the mean-value calculation inevitably raises the threshold in the neighbourhood of targets. Consequently, closely

spaced targets can mask each other, particularly if a small target is located close to a large or extended target.

Furthermore, because of the MAX-process, the GOCA threshold cannot immediately follow an abrupt rise or fall in clutter level, e.g. the front of a thunderstorm. Therefore the threshold will lead in time in case of a steep rise in clutter level and will lag in case of a steep fall. Both these effects can result in target losses.

4.1.7 Cell Averaging Statistic Hofele CFAR

The Cell Averaging Statistic Hofele CFAR (CASH-CFAR) comprises of a circuit of A sub-registers, each containing L storage cells. By means of a special maximum-minimum process, a clutter representative sum-value S_l, will be selected from A sum-values, S_1,...S_A. The threshold value is calculated based on this above maximum or minimum circuit. By means of a special maximum-minimum process, a clutter representative sum-value S_l, will be selected from A sum-values, S_1,...S_A. The threshold value is calculated based on this above maximum or minimum circuit.

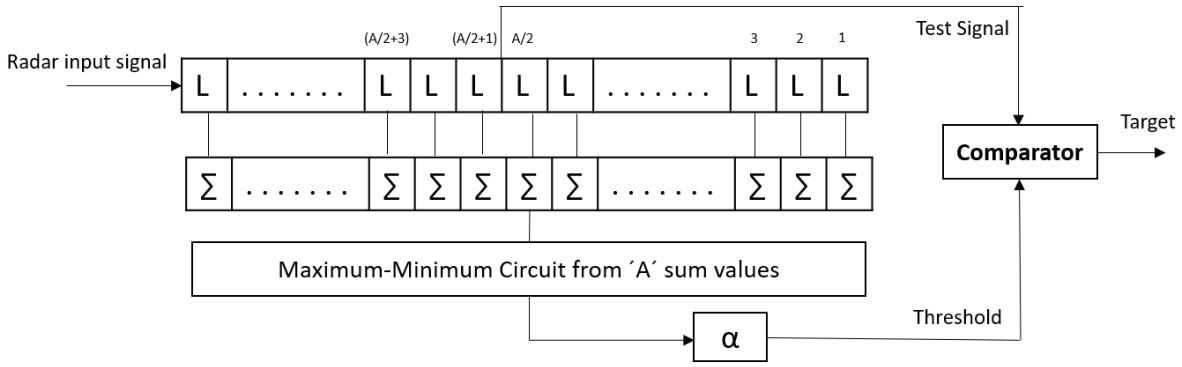


Figure 4.6: CASH-CFAR [22]

4.1.7.1 Limitations of CASH-CFAR

The complexity of this algorithm occurs in terms of hardware complexity. The reason is due to the maximum-minimum circuit which needs registers for both maximum and minimum circuits separately.

4.1.8 Effect of Algorithm Parameters

Each CFAR algorithm basically has three parameters reference window size, guard size and threshold. To achieve the aimed objective, one should make three fundamental decisions regarding the parameters optimal for the algorithm.

First, it is necessary to select the size of the CFAR sliding window. This window moves around the entire detection area so that each cell has a chance to occupy the central cell of the window at a given moment. The central cell is always the one that is evaluated to decide whether a target exists.

Lets take the the example of CA-CFAR to understand in detail. The CA-CFAR mechanism estimates the average size of the clutter by averaging the values of the window cells. Therefore, higher window sizes achieve a high precision in the estimated mean value, while smaller sizes are associated with a lower processing load. Based on the input image size, one must properly choose reference cells and guard cells that offer a good balance between gain and resource consumption.

Secondly, the samples in the given data is an important factor. A very small number of samples leads to a poor estimate of the resulting optimal threshold T, while processing too many samples can slow down the execution of the algorithm considerably.

Finally it had to be decided which P_{fa} values should be considered in the detection procedure. It should be noted that a minimum P_{fa} requires data with more sample point. It is important to understand that the increase in T always benefits the P_{fa}, but also decreases the probability of detection. Therefore, choosing a too high T-value is not a reasonable option. The exact trade-off between P_{fa} and P_d is to be adjusted on a corresponding T value without additional loss. The threshold that is too low will produce a very good P_d because it will cause the threshold to remain low, which will result in the detection of most targets. However, this low threshold will also result in clutter samples of large size being classified as targets, which will result in a very poor P_{fa} value. The opposite effect is achieved when a very high threshold value is selected.

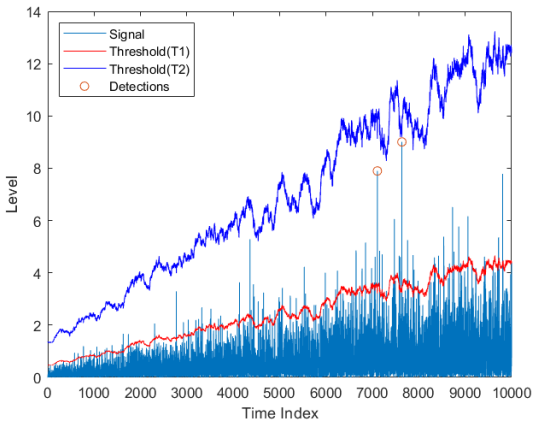


Figure 4.7: CFAR Threshold

The Figure 4.7 above shows the variation of threshold with respect to targets and cluster. The data is generated using Rand stream from MATLAB with 200 training cells. The corresponding custom thresholds are $T_1 = 4.25$ (red line) and $T_2 = 12.35$ (blue line). This simple example is used only for illustration in the variation of threshold.

4.2 Clustering

4.2.1 K-Means Clustering

Consider a data set of N data points $(x^i)_{1 \leq i \leq N}$ which live in the d-dimensional feature space. The primary goal is to partition the data set into K number of clusters. It can be

formalized with the notion by first introducing a set of n-dimensional vectors μ_k , where $k = 1, ..., K$, in which μ_k is the mean associated with the kth cluster. The goal is to find an assignment of data points to clusters, as well as a set of vectors μ_k, such that the sum of the squares of the distances of each data point to its closest vector μ_k, is a minimum. The objective function J represents the sum of square of distances of each point to its assigned vector μ_k.

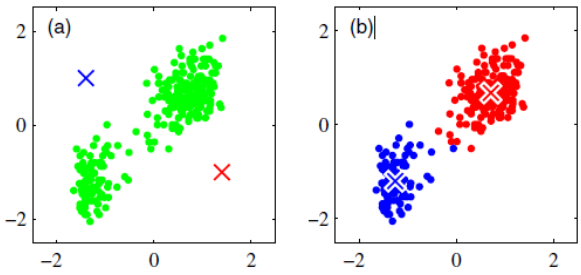


Figure 4.8: K-Means Clustering

The objective function is to be minimzed by iteratively finding the values of r_{nk} and μ_k. The two parameters are estimated using Expectation-Maximization algorithm as discussed detail in [10]. After applying the E-M algorithm we get the values in the Equation (4.9). The (a) and (b) in the Figure 4.8 above represents a dataset before and after clustering.

$$f(x) = r_{nk} = \begin{cases} 1, & \text{if } k = \arg\max_x \|x_n - \mu_j\|^2 \\ 0, & \text{otherwise} \end{cases} \tag{4.9}$$

$$Z = \frac{\sum_n r_{nk} x_n}{\sum_n r_{nk}} \tag{4.10}$$

The denominator in this expression is equal to the number of points assigned to cluster k, and so this result has a simple interpretation, namely set μ_k equal to the mean of all of the data points x_n assigned to cluster k. For this reason, the procedure is known as the K-means algorithm.

4.2.2 DBSCAN

A category of Density-based clustering, DBSCAN stands for Density-based spatial clustering of applications with noise. The density in center based approach is estimated for a particular point in the dataset by counting the number of points within a specified radius, ϵ of that point [8].

The density of any point will depend on the specified radius. The center-based approach to density allows us to classify a point as being

- Interior of a dense region (a core point)
- Edge of a dense region (a border point)

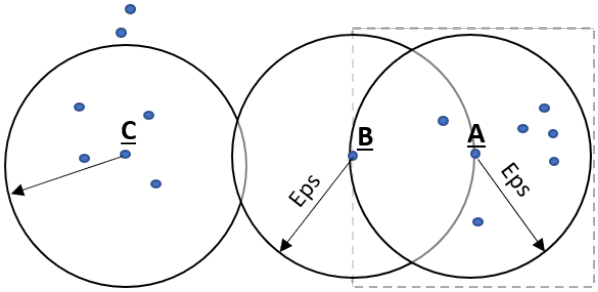


Figure 4.9: DBSCAN points [23]

- Sparsely occupied region (a noise or background point)

Core points: These points are in the interior of a density-based cluster. A point is a core point if the number of points within a given neighborhood around the point as determined by the distance function and a user-specified distance parameter, ϵ, exceeds a certain threshold, MinPts, which is also a user-specified parameter.

Border points: A border point is not a core point, but falls within the neighborhood of a core point. A border point can fall within the neighborhoods of several core points.

Noise points: A noise point is any point that is neither a core point nor a border point.

In the Figure 4.9, A is a core point, B is a border point and C is a noise point. With the knowledge of points in the density based dataset. The algorithm can be explained as follows. Any two core points that are close enough within a distance ϵ of one another are put in the same cluster. Likewise, any border point that is close enough to a core point is put in the same cluster as the core point. (Ties may need to be resolved if a border point is close to core points from different clusters). Noise points are discarded. The formal details are given in the algorithm in A. This algorithm uses the same concepts and finds the clusters.

DBSCAN can find many clusters that could not be found using K-means. However, DBSCAN has trouble with high-dimensional data because density is more difficult to define for such data. The problem lies in the computational complexity of finding nearest neighbours using region query. In this report, the method is used as the range-doppler of radar mostly doesnot have high problems with density, except in the situations of high noise. For such kind of situations, preprocessing of data is highly essential before performing clustering. [23]

4.3 Neural Network Implementation

4.3.1 Implementation using U-Net

Structure of neural network

A Fully Connected Network (U-Net) is implemented with following layers in order. It has 3 encoder blocks and one encoder-decoder bottleneck and 3 decoder blocks. Each encoder block has 2 X ConvNet followed by a LeakyRELU. The output of the encoder block is then passed to Batch Normalization and then the output from the layer is passed to the

Max Pooling layer. The output of the encoder block is then passed to a Concatenate layer with the input from encoder block. Similarly, the decoder has 2 X ConvNet followed by a LeakyRELU. The output of the above block is then concatenated with the encoder layers. At the final stage, the output from decoder is passed to the ConvNet with sigmoid activation to get the output class.

For training this network, Adam Optimizer is used with a learning rate of $1e-4$ and categorical focal loss as the loss function. The architecture used for the implementation is shown in Figure 4.10 below.

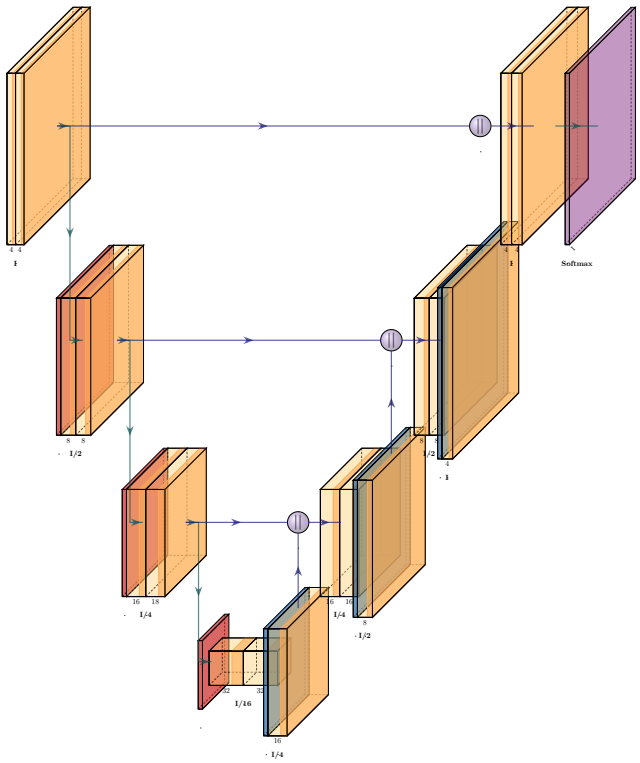


Figure 4.10: U-Net

The network also yields good results when trained using a binary cross-entropy loss function.

4.3.2 Implementation using Mask R-CNN

The implementation of Mask R-CNN is based on the implementation of matterport on git repository. As already discussed in the Section 2, the architecture of Mask R-CNN (regional convolutional neural network) is a two stage framework: the first stage scans the image and generates proposals(areas likely to contain an object). And the second stage classifies the proposals and generates bounding boxes and masks.

The implementation can be explained on an higher level. Custom backbone is the trained convolutional network that is used for early detection of features. In this thesis, ResNet-50 architecture is used as a custom backbone. To improve the object detection with respect to feature scalability, Feature pyramid network is used in combination with

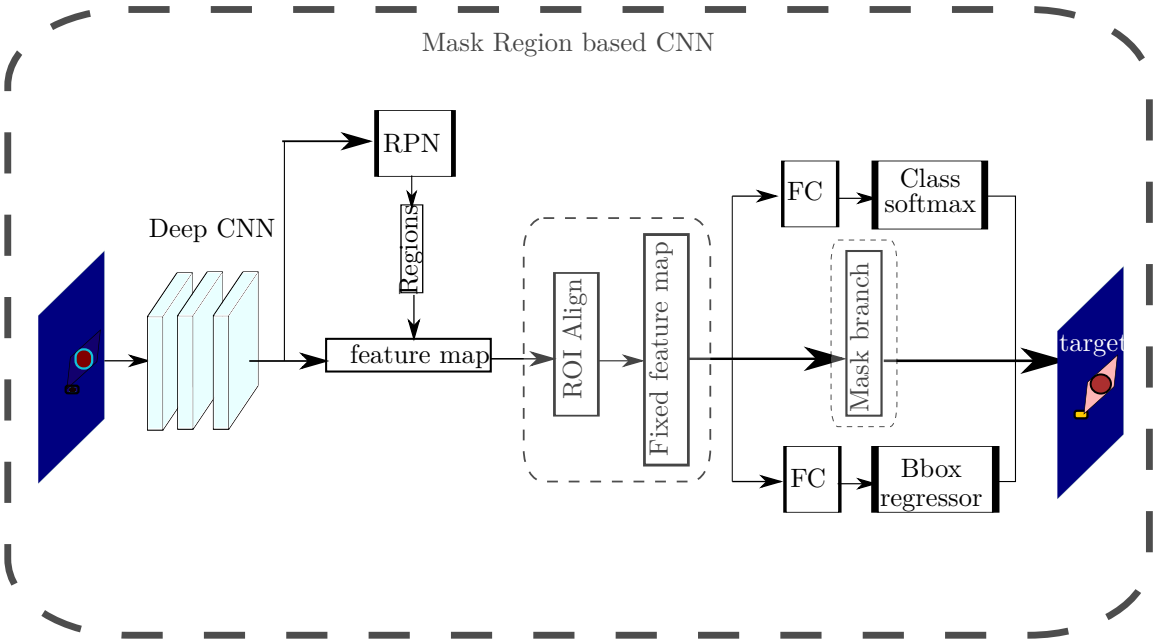


Figure 4.11: Proposed deep learning region based cnn architecture

the ResNet. The network has currently understanding of different features in the given input. The RPN comes into picture where anchors of aspect ratio (0.5,1,2) with 256 training anchors per image are chosen. The RPN scans over the backbone feature map generated earlier and generates two outputs for each anchor. The first output is the anchor class, bounding box parameter. The anchor class decides whether the object is in the box or not. The bounding box parameter helps in the refinement of the highest overlap score using the principle of non-maximum suppression. Once the anchors are correctly overlapped the regions of interest is given to the second stage.

The proposals given by the RPN are subjected to ROI pooling. This is similar to the RPN network but has the ability to find the deeper features of the image. The ROI pooling generates two outputs class and bounding box parameter. The ROI pooling performs resize and crop operations on feature map to limit it to a fixed size. This operation enables classification process without ambiguity. The mask branch is a convolutional network that takes the positive regions selected by the ROI classifier and generates masks for them. The generated masks are of low resolution. But they are soft masks, represented by float numbers, so they hold more details than binary masks. The small mask size helps keep the mask branch light. During training, scaling down the ground-truth masks to above resolution helps to compute the loss, and during inference, scaling up the predicted masks to the size of the ROI bounding box and which yields the final masks, one per object.

4.3.2.1 Mask R-CNN Library

Mask R-CNN tools created for the practical part of the thesis consist of two modules. This implementation is based on a Python implementation of Mask R-CNN [24] written

by Waleed Abdulla from Matterport, Inc published their implementation under the MIT License.

The library consists of four Python modules.

- config.py: The configuration file for the model
- model.py: The core of the Mask R-CNN model. It builds up the model
- target.py: The target or custom named file contains the different parameters and calls to the functions for defined in the model
- utils.py: Utilities for the model

5 Experimental results

5.1 Evaluation of state-of-the-art

The results of the state-of-art algorithms are generated from MATLAB using Constant False Alarm Detection (CFAR) algorithms. The sources of generation already discussed detail in the Chapter 3.

5.1.1 Results of CFAR Algorithms - Simulator

The driving simulator is generated with two pedestrians walking against each other in opposite directions. The above reference image is generated with respect to frame '13' (whole dataset containing 300 frames). The parameters for the CFAR algorithms are listed in the Table 5.1 below.

CFAR Parameters	Values
Dataset	rawdatacreateComplexScenario
Probability of false alarm, P_{fa}	9e-3
Reference cells, ref_{cells}	10
Guard cells, g_{cells}	7
Order (applicable only for OS-CFAR), k	7

Table 5.1: CFAR parameters in the real-time measurements

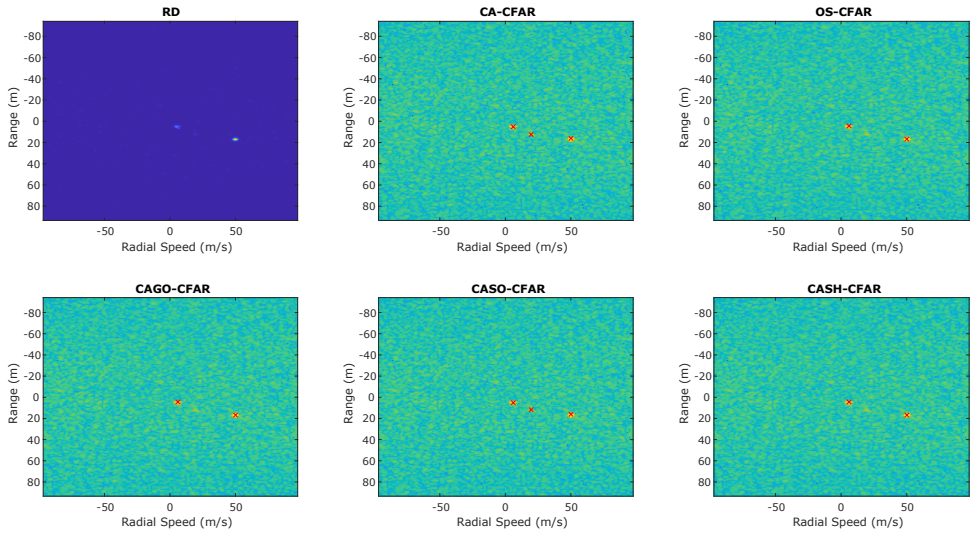


Figure 5.1: CFAR simulator results

5.1.2 Results of CFAR Algorithms - Realtime measurements

The simulation [20] is generated with the scene of single pedestrian, cycle moving in a cycle trajectory and two pedestrians walking against each other in opposite directions. The above reference image is generated with respect to frame '213' (whole dataset containing 300 frames). The parameters for the CFAR algorithms are listed in the Table 5.2 below.

CFAR Parameters	Values
Dataset	h1cycleh2
Probability of false alarm, P_{fa}	1e-7
Reference cells, ref_{cells}	24
Guard cells, g_{cells}	20
Order (applicable only for OS-CFAR), k	18

Table 5.2: CFAR parameters in the real-time measurements

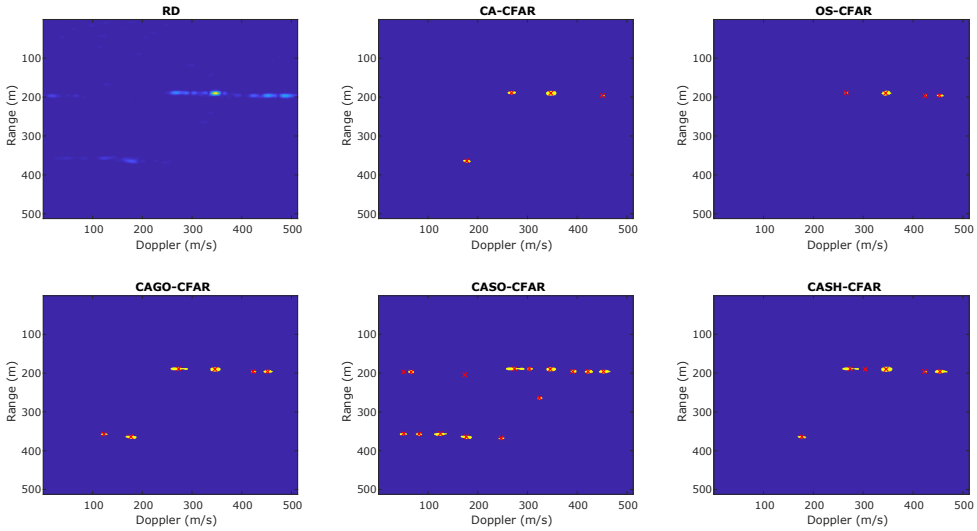


Figure 5.2: CFAR realtime results

The CFAR algorithms applied on the data simulated from radarbook consists of high scatter of information. The data scenario has two pedestrians walking against each other with some distance between them. The expected result after CFAR must have only two objects in the map. The scatter observed from interference, multi-path reflections are not handled properly. In order to get better detection accuracy, the next subsection shows the result of clustering on a spread data. The idea of grouping into useful and unwanted information brings the target of interest to the forground and rest everything to background (noise).

5.1.3 Results of Clustering Algorithm - Real-Time with k-means

The range-doppler map is generated from the dataset with the cycle movement around a circle. The frame '41' was used in the dataset with radarbook. The above figure 5.3

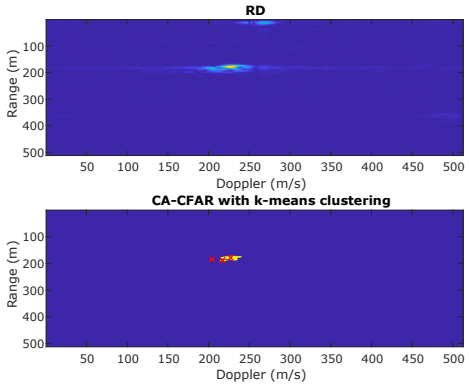


Figure 5.3: CA-CFAR with k-means clustering

depicts the k-means clustering algorithm applied on CA-CFAR. The clusters can be seen with their centers at each target having a 'x'. The problem in the algorithm is regarding division into three clusters although original image has only single target, which essentially must create a single cluster.

5.1.4 Results of Clustering Algorithm - Real-Time with DBSCAN

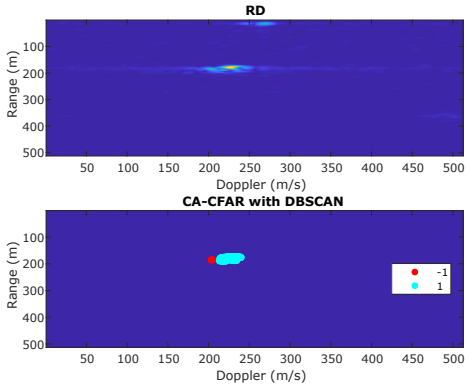


Figure 5.4: CA-CFAR with DBSCAN

The range-doppler map is generated from the dataset with the cycle movement around a circle. The frame '41' was used in the dataset with radarbook. The parameters for this algorithm used were $MinPts = 4$ and $epsilon = 4$. The above figure 5.4 depicts the density based clustering algorithm applied on CA-CFAR. The clusters can be seen clearly with the cluster labelled as '1'. The label '-1' is referred to noise clusters (which is not target of interest). This algorithm has improved clustering accuracy.

5.1.5 Performance Analysis of CFAR

For analysis, the implemented algorithms will be tested in defined scenarios containing clutter, multiple object scenarios. The test samples are generated using simulator and

with real-time measurements. As a performance measure it will be checked if all potential objects can be detected or not. In the following, the scenarios will be explained and compared using Matlab simulations. There are a few problems when it comes to object detection using CFAR which either modify the specified false alarm rate or minimize the detection probability.

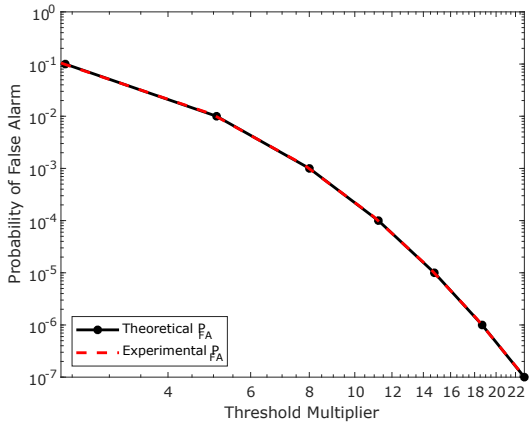


Figure 5.5: CA-CFAR Theoritical vs Experimental

As shown in Figure 5.5, the experimental and theoritical results are closely related with small differences in P_{fa}. Although CA-CFAR performs very well in homogeneous environments, its detection performances suffers drastically with closely spaced multiple targets. At the clutter wall, false alarm probabilities increase by as much as an order of 10^2. The following result is obtained after the testing with respect to a dataset with closely spaced targets. This renders the algorithm not very helpful in clutter wall environments.

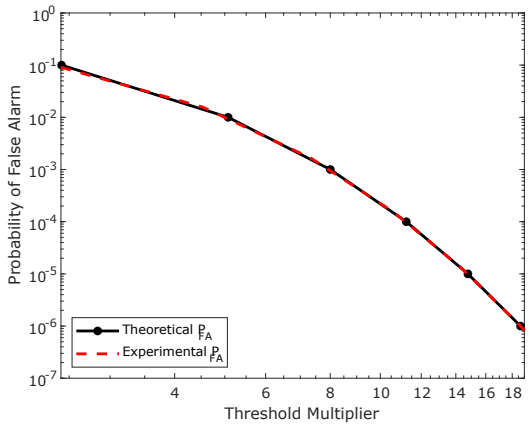


Figure 5.6: OS-CFAR Theoritical vs Experimental

Ordered Statistics CFAR performs similarly to cell averaging CFAR at the clutter wall. Both algorithms only gradually slope upwards their thresholds approaching the clutter wall, and past the clutter edge, the thresholds of both algorithms get as low as

to frequently allow false alarms. OS-CFAR algorithms selected for different ranked k^{th} values perform differently at the clutter wall. In general, the greater the k, the less the probability of false alarm will increase at the clutter edge. CA-CFAR for window length 32, at the clutter wall, suffers roughly a 100-fold increase in false alarm rate from the designed 10^{-4} to nearly 10^{-4}.

Probability of detection performance at the clutter wall for the various k value is reversed from that for probability of false alarm. The smaller the k value, the less the target masking effect before the clutter wall. But in case of the current dataset used in this thesis, $k = 0.5N$ gave better detection results than $k = 0.7N$ and $k = 0.9N$.

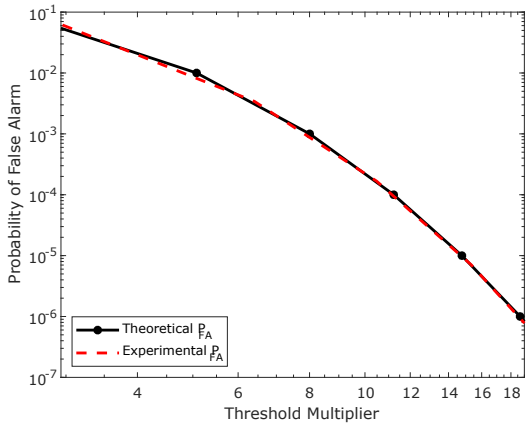


Figure 5.7: SOCA-CFAR Theoritical vs Experimental

Although SOCA-CFAR and GOCA-CFAR outperform CA-CFAR in some cases, they exhibit resistance to target masking and also are vulnerable to the clutter edge and wall.

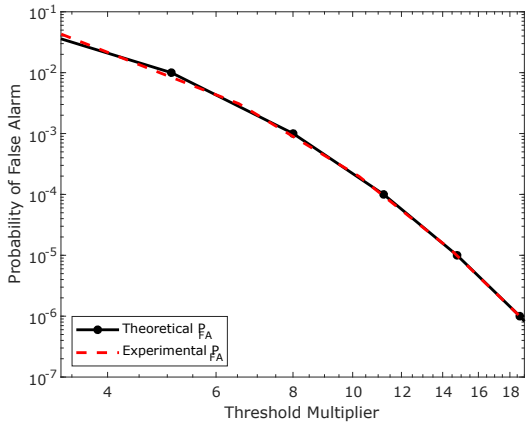


Figure 5.8: GOCA-CFAR Theoritical vs Experimental

Note: The results displayed in the metric figures with respect to GOCA-CFAR and SOCA-CFAR are referred to GOCA and SOCA.

5.1.6 Clutter and Object Masking

Clutter can be defined as the undesirable signal from the aspect of object detection as presented in [25]. In automotive applications, multipath reflections can cause undesirable echos in the recording area of the sensor. The CFAR algorithms are forced to identify this clutter as irrelevant. Due to the presence of clutter, the threshold value in such regions must be increased. The specified false alarm rate does not meet such environments containing clutter.

In the regard to the window size, if the sliding window is chosen larger than the clutter window, the clutter samples has less weight.

5.1.7 Comparison of different CFAR methods

The comparison of different types of CFAR methods are compared with respect to the performance metric. The metric used is Probability of false alarm v. Threshold multiplier.

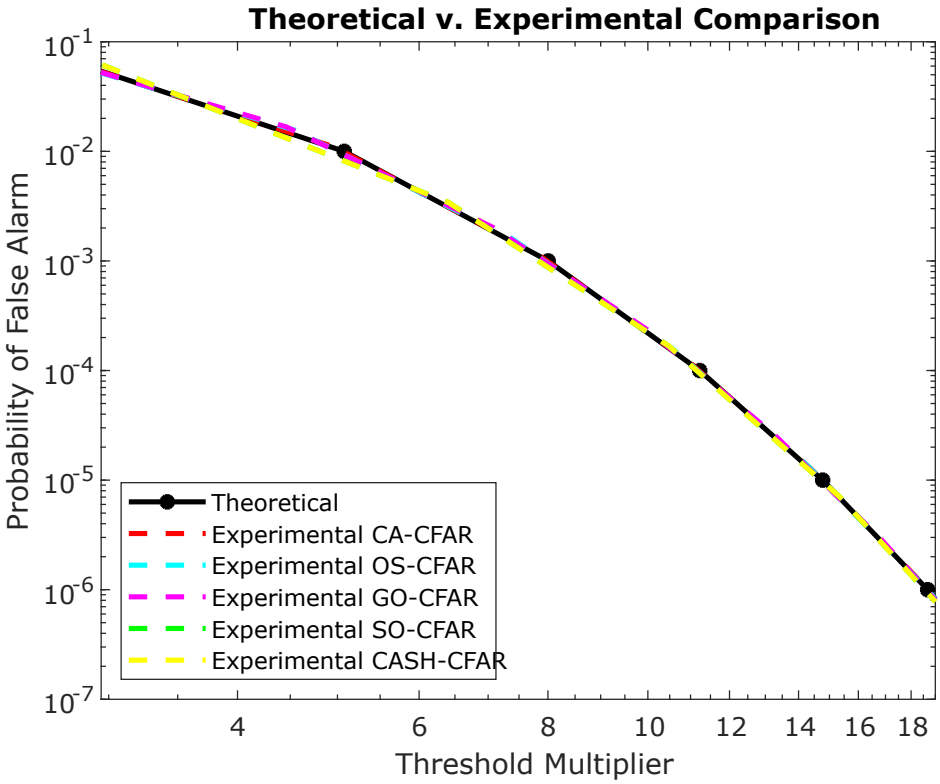


Figure 5.9: Analysis of different CFAR methods

From this Figure 5.9, due to the semilogarithmic axis plot of all algorithms, the observation is that all the methods are closer to the theoritical metric. But, when seen closely by individual plot they vary a lot with respect to theoritical. The threshold dependency also varies from different inputs and CFAR algorithms.

5.2 Evaluation of segmentation

5.2.1 U-Net Implementation Results

The results of U-Net implementation are provided below in Figure 5.10. The implementation is performed with the help of radar dataset generated from radar simulation. It shows the two pedestrian scene walking across each other.

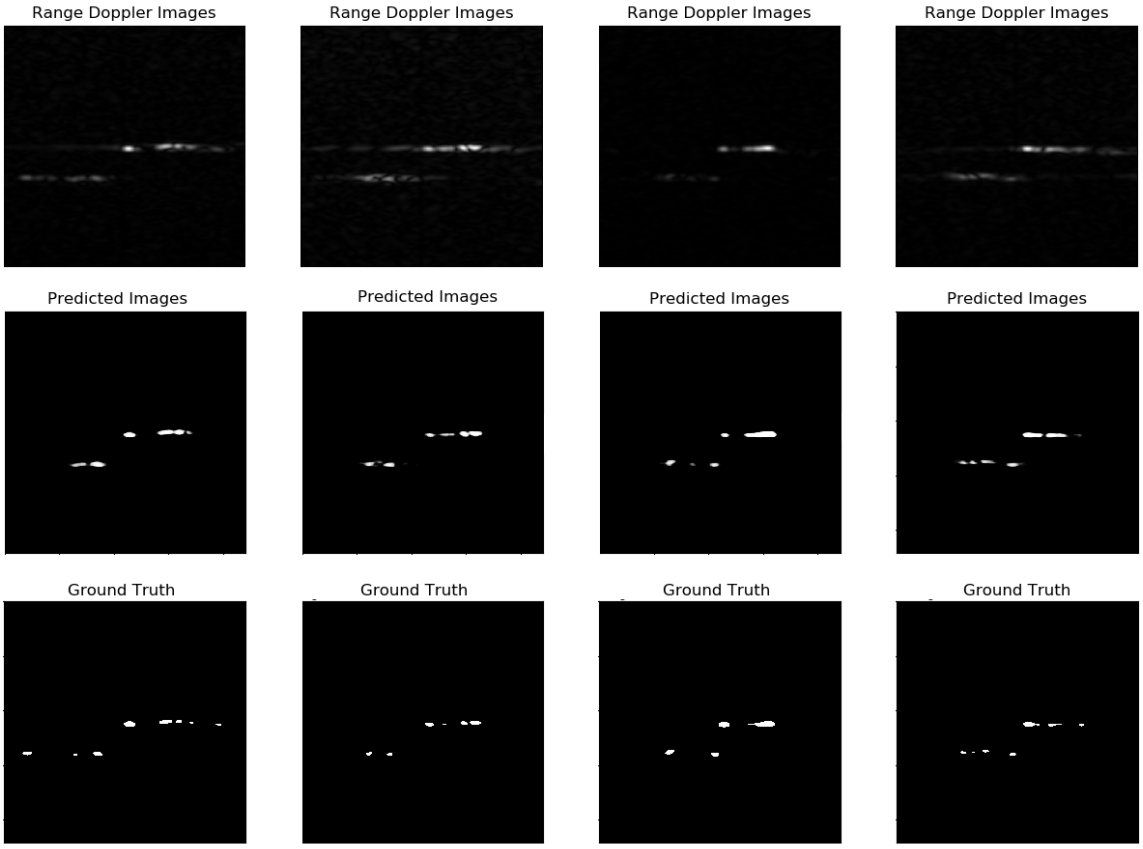


Figure 5.10: U-Net results

The output shown in Figure 5.11a shows target image. The prediction at end of the network is displayed in Figure 5.11b.

(a) Ground truth (b) Predicted Image

Figure 5.11: (a) U-Net ground truth (b) U-Net predicted image

The plots related to metrics are visualized by the plots in Figures 5.12b and 5.12a. The metrics in the 5.12b shows the F1-score of the U-Net which has exponential increase on training with number of epochs. The Figure 5.12a shows the sharp decline in the loss after some epochs, with gradient falling down very sharply.

(a) U-Net validation loss (b) U-Net validation F1-Score

Figure 5.12: (a) U-Net Validation loss (b) U-Net Validation F1-Score

5.2.2 Mask R-CNN Implementation Results

The configuration parameters for the model to be trained is listed in the section configuration in the Chapter A. The model is trained with repsect to the dataset generated with respect to three scenarios involved in the dataset as already generated in the Chapter 3. The results shown in Figure 5.13 above depicts the test images and predicted images from the radar dataset. The frame 213 in Figure 5.13 clearly shows the difference in detection results between U-Net and Mask R-CNN.

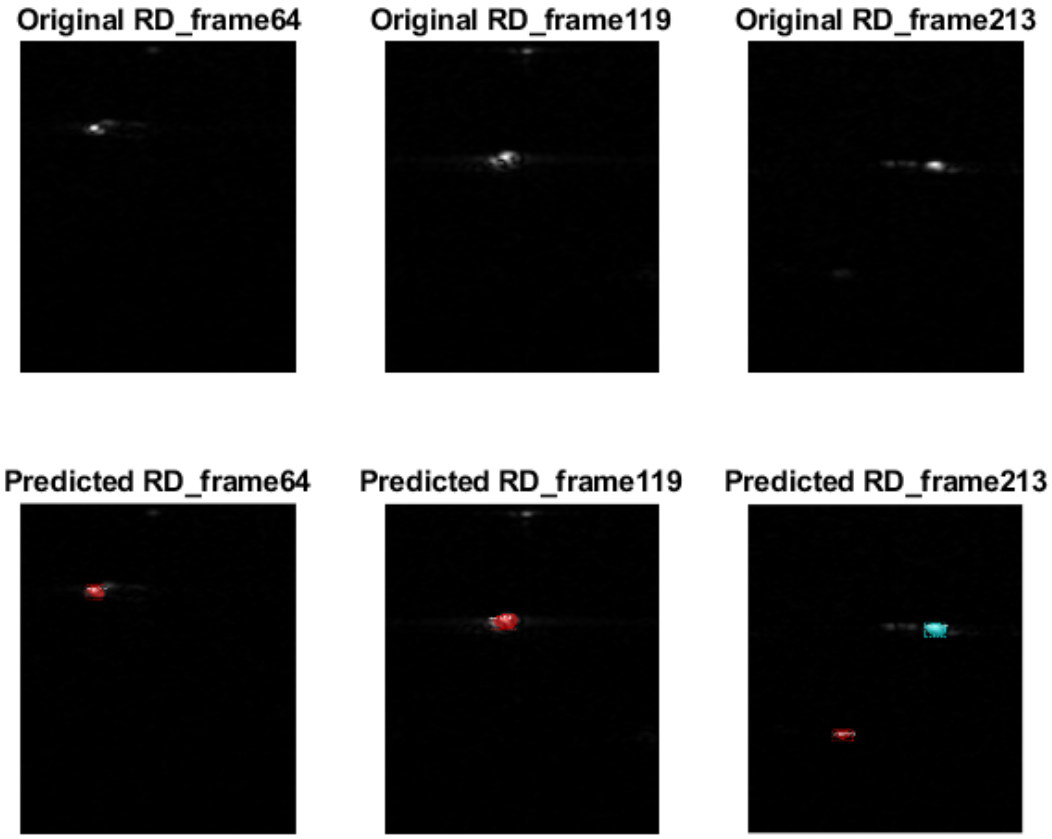
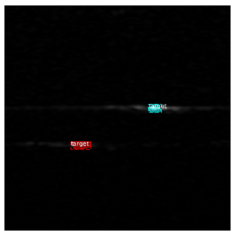
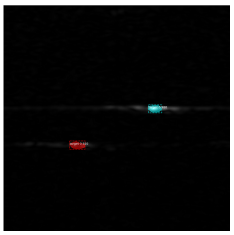


Figure 5.13: Mask R-CNN with real-time dataset

The output shown in Figure 5.14a shows target image. The instance preserved at the end of the network is displayed in Figure 5.14b.



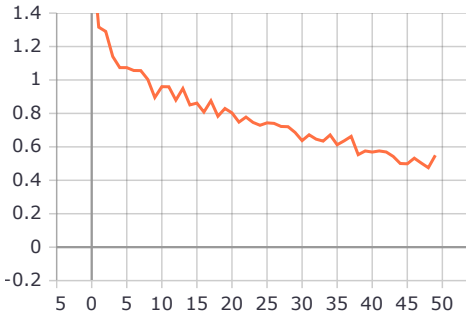
(a) Ground truth



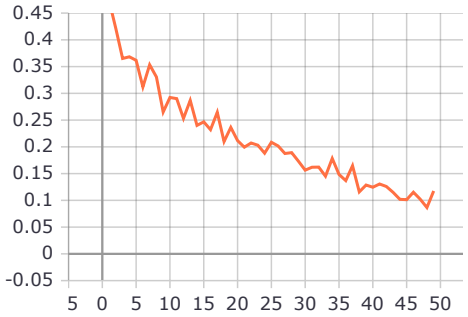
(b) Predicted instance

Figure 5.14: (a) Mask R-CNN ground truth (b) Mask R-CNN predicted instance

The losses of Mask R-CNN like training loss and bbox loss are depicted below in Figures 5.15a and 5.15b.



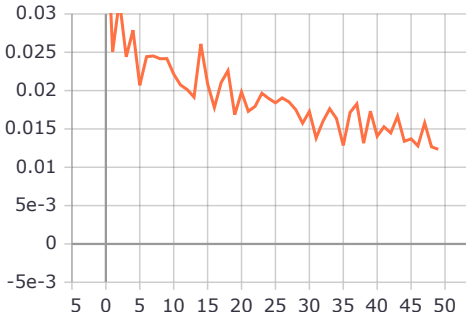
(a) Mask R-CNN training loss



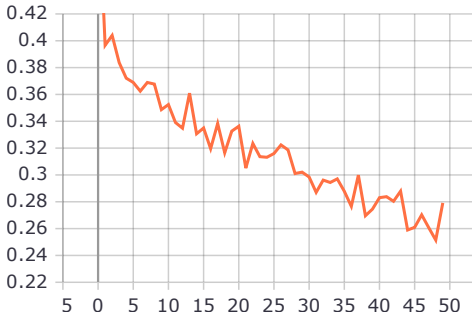
(b) Mask R-CNN bounding box loss

Figure 5.15: (a) Mask R-CNN training loss (b) Mask R-CNN bounding box loss

The losses of Mask R-CNN like mask loss and class loss and are depicted below in Figures 5.16b and 5.16a.



(a) Mask R-CNN class loss



(b) Mask R-CNN mask loss

Figure 5.16: (a) Mask R-CNN class loss (b) Mask R-CNN mask loss

The Average precision vs Intersection of Union (IoU) of Mask R-CNN for different anchor ratios is shown below and are depicted below in Figure 5.17.

Anchor scales	Anchor ratios	Average precision
(8, 16, 32, 64, 128)	[0.5, 1, 1.5]	0.59514
(16, 32, 64, 128, 256)	[0.5, 1, 2]	0.63221

Table 5.3: Average precision for different anchor ratio

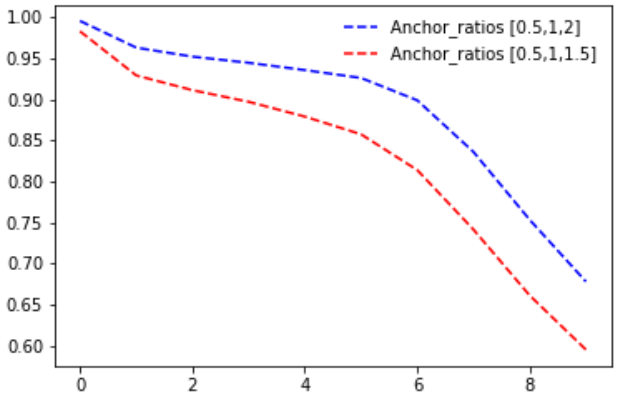


Figure 5.17: mAP vs IoU for two sets of anchor ratios

5.2.3 Comparison of neural network methods

The comparison of different neural network implementation is done with the help of performance metrics namely Sensitivity, Precision and Specificity. [26]

To understand the metrics, we need to understand the following parameters. In a radar image with single target, we need to find existence one target.

- True positive (TP): Prediction is +ve and the radar has a target (desired)
- True negative (TN): Prediction is -ve and the radar has no target (also desired)
- False positive (FP): Prediction is +ve and the radar has no target (false alarm - bad)
- False negative (FN): Prediction is -ve and the radar has a target (worst)

With the above definitions of positives and negatives, the peformance metrics are defined as follows

$$\text{Sensitivity (Recall)} = \frac{TP}{TP + FN} \tag{5.1}$$

$$\text{Specificity} = \frac{TN}{TN + FP} \tag{5.2}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.3}$$

$$\text{IoU (Intersection of Union)} = \frac{TP}{TP + FP + FN} \tag{5.4}$$

$$\text{F1-Score} = 2 * \frac{Precision * Sensitivity}{Precision + Sensitivity} \tag{5.5}$$

Average Precision (AP) is a weighted mean of precisions at a given threshold with the increase in recall from the previous threshold used as weights. Averaging of AP across all classes gives mean Average Precision (mAP). [26]

The metrics of precision, recall and mean average precision are shown in the below tables for two datasets mainly masktrain11 (cycle movement) and masktrain21 (pedestrians movement) at threshold of 0.5 (trained for 20 epochs).

Model	mAP	Precision	Recall	F1-Score
U-Net	0.4543	0.6162	0.6781	**0.6457**
Mask RCNN	**0.6192**	**0.8191**	0.3412	0.4817

Table 5.4: Metrics comparison of U-Net and Mask R-CNN for masktrain11

Model	mAP	Precision	Recall	F1-Score
U-Net	0.3341	0.5362	0.5771	**0.5559**
Mask R-CNN	**0.5167**	**0.7423**	0.3114	0.4387

Table 5.5: Metrics comparison of U-Net and Mask R-CNN for masktrain21

The metrics in Tables 5.4 and 5.5 clearly shows that U-Net has good F1-score by the segmentation network and Mask R-CNN has good precision and mAP due to the fitting of masks. In order to evaluate the model performance with respect to Mask R-CNN, the metric mAP is needed for different threshold IoU (trained for 50 epochs with h1cycleh2 dataset). The results shown in the Table 5.6 are reported using the average over IoU for different mean average precisions mAP (AP0.50, AP0.70, AP0.90) for both network implementations.

Model	Backbone	AP@0.5	AP@0.7	AP@0.9
U-Net	FCNN	0.6322	0.5222	0.3747
Mask R-CNN	ResNet-50	0.8991	0.8362	0.6781

Table 5.6: Average precision for different IoU thresholds

6 Conclusion and Outlook

6.1 Conclusion

The results and discussions in this thesis shows that deep learning methods can be effectively applied to achieve object detection. The thesis work covered the practical application of different deep learning components, specifically related to the use of image data for perception in an autonomous vehicle. The goal of this thesis was to implement detection techniques for radar signal detection. With the rising interest in the area of research in artificial intelligence, mainly regarding deep learning, the object detection architectures can also be used for the range-doppler images which are post-processed generated from a radar simulation or virtual simulator.

The first part of the thesis was dedicated to a theoretical background behind radar signal processing, understanding of neural networks mainly ConvNets and components of a deep neural network. The second part of the thesis was dedicated for the implementation of various types of constant false alarm rate algorithms and clustering algorithms with basis on theoritical aspect. Then with the concept of segmentation, the architectures using deep learning were exploited for the radar target detection. Firstly, the one kind of segmentation called semantic segmentation is implemented and then the limitations for the network is observed. Then the idea of instance segmentation helped to analyze the previous limitation and overcome the limitations successfully.

The research flows in the background of the theoretical part and is briefly summarized at the beginning of the chapter on the implementation. The developed system in the initial phase using state-of-the-art algorithms and the deep learning techniques gives a user to analyze the margin required to achieve the goals of accuracy and proper solution to the given title of the thesis. The results from the implementations reveal that deep learning methods hold high promise for the future and also much scope for improvement.

6.2 Outlook and Future scope

The outlook of the thesis can be discussed with the summary of theory and implementation. The research in the field with best of the radar sensors and high demands of the artificial intelligence inspires the community of engineers to bring up new ideas and techniques to enable the autonomous support for the society. The future of object detection observed using image sensor and radar sensor with artificial intelligence is shown in Figure 6.1.

The possible extensions to the above developed target detection schemes can be as discussed as follows. The architecture used in the instance segmentation currently works on a ResNet backbone. The targets that are aimed in the field of autonomous driving are basically limited and not too many in the area of vicinity. The custom backbone can be designed using an autoencoder network. This will enable the system to be working with reduced time and space constraints. The other scope of extension would lie in the type of convolutional network. Instead of using a conventional convolutional type of network, the improvement can be observed with densenets which actually have different characteristics of providing the specific targets as outputs.

These approaches offer a lot of advantages, while the biggest challenge to practical applicability is the high demand for computational resources. Rapid progress in research and development of both hardware and algorithms is expected, to ease the computational requirements in the future. Hence, the detailed analysis of deep learning methods presented in the thesis offer the potential for more understanding and continuing research to improve current methods.

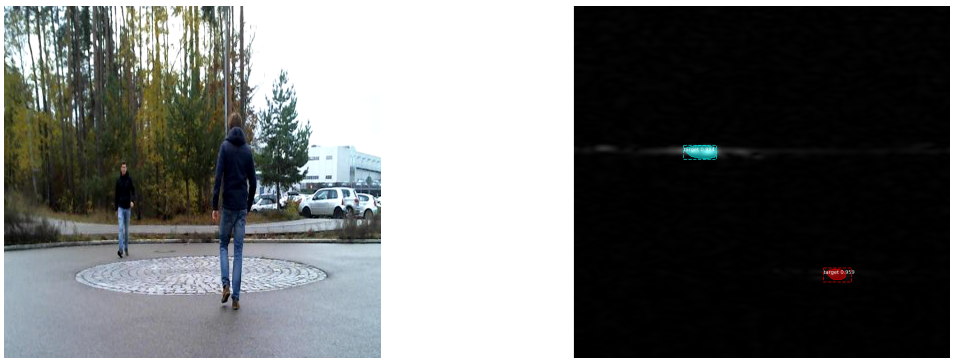


Figure 6.1: Image sensor perception vs Radar sensor with Artificial intelligence perception

A Appendix

A.1 Software Requirements

The software requirements for the thesis are classified as follows. The first one is Matrix Laboratory (MATLAB) and the second one is Python 3.7 as programming language with the libraries for deep learning such as Keras and Tensorflow. In addition to above libraries, the common libraries for standard input/ouptut, mathematical functions, graphical plots are also used in the implementation.

(a) MATLAB

(b) Keras

Figure A.1: Software Requirements

The Phased Array System Toolbox in the MATLAB gives different functions for range and doppler estimation. It also has functions for the implementation of CFAR. Keras is used as library for implementation and training of neural networks. Tensorflow is used in the combination with Keras as frontend to provide different built-in functionalities.

Radar Datacube

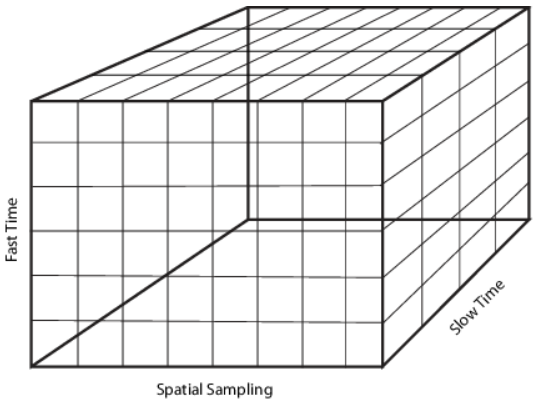


Figure A.2: Radar datacube

The radar data cube in Figure A.2 is a convenient way to conceptually represent space-time processing. To construct the radar data cube, assume that preprocessing converts the RF signals received from multiple pulses across multiple array elements to

complex-valued baseband samples. Arrange the complex-valued baseband samples in a three-dimensional array of size K x N x L. The fast time dimension is also referred to as the range dimension and the fast time sample intervals (N_s), when converted to distance using the signal propagation speed, are referred to as range bins. Typical Pulse repetition intervals are much longer than the fast-time sampling interval. Because of the long sampling intervals, samples taken across multiple pulses are referred to as slow time. Processing data in the slow-time dimension allows you to estimate the Doppler spectrum at a given range bin.

A.2 Mask R-CNN Configuration

The general configuration such as parameters like backbone, strides and image shape can be adapted based on the requirement of the application. The file for this configuration is available under the name *config.py* in [24]. The configuration can also contain custom parameters which can support the neural network during the training and inference.

Parameters	Values
BACKBONE	resnet50
BACKBONE_STRIDES	[4, 8, 16, 32, 64]
BATCH_SIZE	1
BBOX_STD_DEV	[0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE	None
DETECTION_MAX_INSTANCES	100
DETECTION_MIN_CONFIDENCE	0.9
DETECTION_NMS_THRESHOLD	0.3
FPN_CLASSIF_FC_LAYERS_SIZE	1024
GPU_COUNT	1
NAME	target
IMAGES_PER_GPU	1
IMAGE_CHANNEL_COUNT	3
IMAGE_MAX_DIM	512
IMAGE_META_SIZE	14
IMAGE_MIN_DIM	512
IMAGE_MIN_SCALE	0
IMAGE_SHAPE	[512 512 1]
LEARNING_MOMENTUM	0.9
LEARNING_RATE	0.001
VALIDATION_STEPS	50
WEIGHT_DECAY	0.0001

Table A.1: Mask R-CNN config parameters

A.3 Mask R-CNN Mask Configuration

The table A.2 shows the various types of configurations related to mask in the Mask-RCNN network.

Parameters	Values
MASK_POOL_SIZE	14
MASK_SHAPE	[28, 28]
MAX_GT_INSTANCES	100
MEAN_PIXEL	[123.7 116.8 103.9]
MINI_MASK_SHAPE	(56, 56)
NUM_CLASSES	2
POOL_SIZE	7
POST_NMS_ROIS_INFERENCE	1000
POST_NMS_ROIS_TRAINING	2000
ROI_POSITIVE_RATIO	0.33
RPN_ANCHOR_RATIOS	[0.5, 1, 2]
RPN_ANCHOR_SCALES	(32, 64, 128, 256, 512)
RPN_ANCHOR_STRIDE	1
RPN_BBOX_STD_DEV	[0.1 0.1 0.2 0.2]
RPN_NMS_THRESHOLD	0.7
RPN_TRAIN_ANCHORS_PER_IMAGE	256
STEPS_PER_EPOCH	100
TOP_DOWN_PYRAMID_SIZE	256
TRAIN_ROIS_PER_IMAGE	200

Table A.2: Mask R-CNN mask parameters

A.4 DBSCAN Algorithm

Algorithm 1: Pseudo Code of DBSCAN

Data: Given data
Result: Indices of Clusters

- Label all points as core, border, or noise points
- Eliminate noise points by performing region query with ϵ and check with MinPts
- Put an edge between all core points that are within ϵ of each other
- Make each group of connected core points expanding the cluster into a separate cluster by
- Assign each border point to one of the clusters of its associated core points.

Bibliography

[1] T. Wagner, R. Feger, and A. Stelzer, "Radar Signal Processing for Jointly Estimating Tracks and Micro-Doppler Signatures", *IEEE Access*, vol. 5, pp. 1220–1238, 2017 (cit. on p. 1).

[2] W. Buller, J. Wilson, J. Kelly, N. Subotic, B. Thelen, and B. Belzowski, "Radar congestion study, Washington DC", Report Number - DOT HS 812 632, pp.35–42, 2018 (cit. on p. 2).

[3] C. Aydogdu, G. K. Carvajal, O. Eriksson, and H. Hellsten, "Radar Interference Mitigation for Automated Driving", 2019 (cit. on p. 2).

[4] J. Lin, Y. Li, and W. Hsu, "Design of an FMCW radar baseband signal processing system for automotive application", *SpringerPlus 5*, vol. 42, 2016, pp.3–10 (cit. on p. 3).

[5] S. M. Patole, M. Torlak, D. Wang, and M. Ali, "Automotive radars: A review of signal processing techniques", *IEEE Signal Processing Magazine*, vol. 34 (2), pp. 22–35, Mar. 2017 (cit. on p. 5).

[6] M.A.Richards, "Fundamentals of radar signal processing", 2nd. McGraw-Hill, 2014 (cit. on pp. 7, 8, 30, 31).

[7] F. X. Hofele, "An innovative CFAR algorithm", in *2001 CIE International Conference on Radar Proceedings*, Oct. 2001, pp. 329–333 (cit. on p. 8).

[8] D. Xu and Y. Tian, "A Comprehensive Survey of Clustering Algorithms", *Annals of Data Science*, vol. 2, 2015, pp.165–193 (cit. on pp. 9, 35).

[9] A. C. Ian Goodfellow, "Deep Learning". MIT, 2017 (cit. on p. 9).

[10] C. M. Bishop, "Pattern Recognition and Machine Learning". Springer, 2006 (cit. on pp. 9, 35).

[11] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection", 2017 (cit. on p. 11).

[12] F. Chollet, "Keras", `https://keras.io`, Online, accessed 20.January.2020, 2015 (cit. on p. 11).

[13] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", 2014 (cit. on p. 11).

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", in *2016 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2016, pp. 770–778 (cit. on p. 14).

[15] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation", 2015 (cit. on pp. 16, 17).

[16] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39 (6), pp. 1137–1149, Jun. 2017 (cit. on pp. 18, 19).

[17] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42 (2), pp. 386–397, Feb. 2020 (cit. on pp. 18–21).

[18] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature Pyramid Networks for Object Detection", in *2017 IEEE Conference on Computer Vision and Pattern Recognition*, Jul. 2017, pp. 936–944 (cit. on p. 19).

[19] A. Dubey, J. Fuchs, M. Lübke, R. Weigel, and F. Lurz, "Generative Adversial Network based Extended Target Detection for Automotive MIMO Radar", in *IEEE International Radar Conference* (Washington DC, USA, USA), Apr. 2020, *accepted contribution* (cit. on p. 22).

[20] I. GmbH, "Inras Products", `https://www.inras.at/en/products/radarbook.html`, Online, accessed 20.January.2020 (cit. on pp. 25, 41).

[21] A. Dutta, A. Gupta, and A. Zissermann, "VGG Image Annotator (VIA)", `http://www.robots.ox.ac.uk/~vgg/software/via/`, Version: 2.0.8, Accessed: Online, 20.January.2020 (cit. on pp. 26, 27).

[22] W. Wiesbeck, "Radar Systems Engineering, IHE, Karlsruhe Institute of Technology", `https://www.ihe.kit.edu/download/RSE_script__2009.pdf`, Accessed: Online, 20.January.2020 (cit. on pp. 29–33).

[23] Xiaowei Xu, M. Ester, H. .-.-. Kriegel, and J. Sander, "A distribution-based clustering algorithm for mining in large spatial databases", in *Proceedings 14th International Conference on Data Engineering*, Feb. 1998, pp. 324–331 (cit. on p. 36).

[24] W. Abdulla, "Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow", `https://github.com/matterport/Mask_RCNN`, accessed 20.January.2020 (cit. on pp. 38, 54).

[25] H. Rohling, "Ordered statistic CFAR technique - an overview", in *2011 12th International Radar Symposium*, Sep. 2011, pp. 631–638 (cit. on p. 45).

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, and V. Michel, "Scikit-learn: Machine Learning in Python", 2012 (cit. on pp. 49, 50).

List of Figures

List of Tables

Curriculum Vitae

Venkat Ramana **Madhavan**, born 30.June.1995 received the Bachelor of Technology degree in *Electronics and Communication Engineering* from Jawaharlal Nehru Technological University, Hyderabad, India in 2016. In July 2016, he joined Ntt Data Incorporation, Bangalore, India, as an Associate Consultant. After working there for one year, he enrolled for Master of Science in *Communication and Multimedia Engineering* at Friedrich-Alexander-Universität Erlangen-Nuremberg.

He completed his research internship in "Development of tool for the analysis of video decoding with MPEG-TS and H.264 container formats" at e.solutions GmbH, Erlangen. During his study, he worked as a working student at Siemens AG, Nuremberg in the field of Augmented Reality development. His interests are in the fields of Software Development, Image Signal Processing and Deep Learning.