Classification, Object Detection and Tracking in High Resolution Radar Imagery for Autonomous Driving Using Deep Learning

By

Ana Raluca Stroescu

A thesis submitted to
the University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Engineering
College of Engineering and Physical Sciences
University of Birmingham
March 2022

To my parents, Anca and Sorin.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to my supervisors, Prof. Marina Gashinova, Prof. Mikhail Cherniakov and Dr. Liam Daniel, for their guidance, encouragement and support and for giving me the opportunity to carry out this research project and to attend international conferences.

I would also like to thank Jaguar Land Rover for funding my MRes programme and making this opportunity possible.

Many thanks to my colleagues, Dom, Müge, Ali, Alan, Shahrzad, Fatemeh, Anum, Emidio and the rest of the MISL team for their continuous support. Special thanks to Dr. Liam Daniel and Dr. Dominic Phippen for providing radar imagery datasets, as well as for their assistance and helpful advice throughout my PhD.

Finally, I am grateful for all the love and support that I have received from my parents, Anca and Sorin, from my boyfriend Jack and from my best friends, Ana, Silviana, Costina and Daisy, throughout this period of my life and for always being there for me when I needed encouragement.

ABSTRACT

Over the past years, there has been an increase in research for artificial intelligence methods in the field of autonomous driving. Significant advances have been made recently and public datasets for urban scenes are now available for object detection and segmentation on optical images for self-driving vehicles. However, the real world traffic environment is very complex and the development of a sensing system that can perform in adverse weather conditions, such as fog, heavy rain and snow, when the capabilities of the optical vision are diminished, still remains a challenge for the automotive industry.

Therefore, this thesis presents a method for classification, object detection and tracking of different roadside targets in high frequency radar imagery, using deep neural networks. The current work confirms that, despite their numerous applications, such as image and speech recognition, health care, finance, web search engines, advertising, etc., neural networks can also be successfully employed in high frequency automotive radar imagery. Classification and object detection methods using deep neural networks have the potential to overcome the current object identification issues in radar and can support the development of all-weather sensing systems for autonomous vehicles.

"Our intelligence is what makes us human,

and AI is an extension of that quality."

Yann LeCun, the founding father of convolutional nets.

Contents

List of Figures

List of Tables

Acronyms

ABS Anti-lock Braking System. 14

ACC Adaptive Cruise Control. 6, 15

ADAS Advanced Driver-Assistance Systems. 1, 14, 15

AE Autoencoder. 23, 27, 37

AEB Automatic Emergency Braking. 6, 15

AI Artificial Intelligence. 18, 19

ANN Artificial Neural Network. 24, 37–39, 43, 83

AP Average Precision. 114

API Application Programming Interface. 67

CE Cross-Entropy. 78

CFAR Constant False Alarm Rate. 98

CNN Convolutional Neural Network. 21, 23, 27, 29, 36, 38–41, 43–48, 59, 61, 67, 72, 80–84, 87, 89–92, 97, 100, 116, 153

CPU Central Processing Unit. 19, 24, 66, 67, 70

DAS Driver-Assistance Systems. 14

DL Deep Learning. 19, 21

DNN Deep Neural Network. 24, 27, 28, 34–36, 39, 41, 46, 66, 73, 92, 94–98, 122, 124, 125

EKF Extended Kalman Filter. 134, 135

FC Fully Connected. 41, 74

FCN Fully Convolutional Network. 102

FMCW Frequency Modulated Continuous Wave. 7

GAN Generative Adversarial Network. 23

GPS Global Positioning System. 12, 16, 33

GPU Graphics Processing Unit. 19, 66, 67, 70

HMM Hidden Markov Model. 127, 129

HPC High Performance Computing. 67

IDE Integrated Development Environment. 67

IMU Inertial Measurement Unit. 12

IoT Internet of Things. 17, 34

IoU Intersection over Union. 115, 116

IoV Internet of Vehicles. 34

KF Kalman Filter. 126, 127, 133, 134

LFMCW Linear Frequency Modulated Continuous Wave. 63, 109, 110, 145

LR Learning Rate. 77, 92

mAP Mean Average Precision. 114–116, 118

ML Machine Learning. 19, 21, 22, 38–40, 47

MLP Multilayer Perceptron. 27, 41

MNIST Modified National Institute of Standards and Technology. 46–48, 53–55, 57–59, 61, 70, 76, 77, 91

MTT Multi Target Tracking. 125, 136, 138, 141, 146, 150

NN Neural Networks. 19

OOP Object Oriented Programming. 139, 140

PCA Principle Component Analysis. 37

PF Particle Filter. 129, 130, 132–139, 141–143, 146, 148, 150, 151

R-CNN Region Based Convolutional Neural Network. 29, 96, 100, 102, 103, 106, 112, 116–118, 122, 138, 145, 152, 154

RBM Restricted Boltzmann Machine. 23

RCS Radar Cross Section. 7, 9, 62

ReLU Rectified Linear Unit. 41, 42, 73, 74, 80, 83

RGB Red-Green-Blue. 72, 81, 82, 92, 108, 138

RNN Recurrent Neural Network. 23, 27

RoI Region of Interest. 100, 102, 105, 106

RPN Region Proposal Network. 100, 102–104, 106

SAE Society of Automotive Engineers. 14, 17

SAR Synthetic Aperture Radar. 36, 96

SGD Stochastic Gradient Descent. 73, 75, 76

SIS Sequential Importance Sampling. 130, 132

SNR Signal to Noise Ratio. 98, 126, 127, 133, 134

SSD Single Shot Multibox Detector. 29, 96, 100, 106, 112, 116–118, 122, 145, 154

SVM Support Vector Machine. 37–39, 43

TBD Track Before Detect. 127, 129

UKF Unscented Kalman Filter. 134, 135

Chapter One

Introduction

Self-driving vehicles have been in the spotlight lately and this domain is becoming more and more popular because of the recent breakthroughs in this field. Radars are important sensors for autonomy, owing to their capabilities of sensing the environment in adverse weather conditions and darkness, when other sensors might struggle. However, one shortcoming of the current technology for autonomous vehicles is the limited ability of performing object identification on radar data. The goal of this work is to investigate the benefits of artificial intelligence methods, such as deep neural networks, for classification, object detection and tracking in high frequency radar imagery.

This chapter will be focused on introducing the main concepts used throughout this thesis: radars, autonomous vehicles and artificial intelligence. Firstly, Section 1.1 will present an overview of the radar's principles and operation and the main types of radars that are of interest in this work: automotive radars and the emerging low-THz radars. This section will be followed by an outline of the current technology and sensors for autonomous vehicles, the levels of automation and examples of popular Advanced Driver-Assistance Systems (ADAS) technologies. Section 1.3 will introduce the necessary artificial intelligence concepts, from machine learning to deep learning and image recognition algorithms. Finally, the motivation

of this work and the research contribution will be stated.

1.1 Radar

Radar (radio detection and ranging) is a detection system which has been widely used since the 1900s to determine the range, velocity and angle of objects of interest, using radio waves. Experiments that demonstrated that radio waves were reflected by metallic targets were conducted in the late 19^{th} century by Heinrich Hertz. In 1897, more experiments with radio communication in the Baltic Sea were also conducted by the Russian scientist Alexander Popov, who made the first observation about the possibility of object detection using radio waves when a warship crossed the radio communication between two other ships [1]. The first radar application, the "Telemobiloskop", was a ship detector for collision avoidance built by the German inventor Christian Hülsmeyer in 1904 [2]. However, widely applicable radar systems became available three decades after.

Radar is used as an effective tool in many disciplines, from a radar gun in speed limit enforcement to military and defence radars, such as airborne early warning and control systems. Some common applications are weather radar, mapping radar, air traffic control, spacecraft, ship navigation and safety, missile guidance, law enforcement, automotive, etc. [3], although they are all built upon the same basic principles.

At present, radar is the most popular system for active sensing and detection, due to its efficiency and accuracy. It can detect objects that cannot be perceived by the human eye or optical cameras and it works in the dark and in adverse weather conditions such as rain, snow, fog, etc. This section will be focused on automotive radar and low-THz radar, as they are of interest for this thesis.

1.1.1 Basic Principles and Operation

The basic radar operating principle is straightforward: the transmitter emits an electromagnetic wave and the echo wave reflected by an object is received and further analysed through digital signal processing and amplification. The position (range, azimuth, elevation), the direction and the velocity can therefore be calculated for the target. The functioning principle and the basic components for radar transmission and reception are illustrated in Figure 1.1.

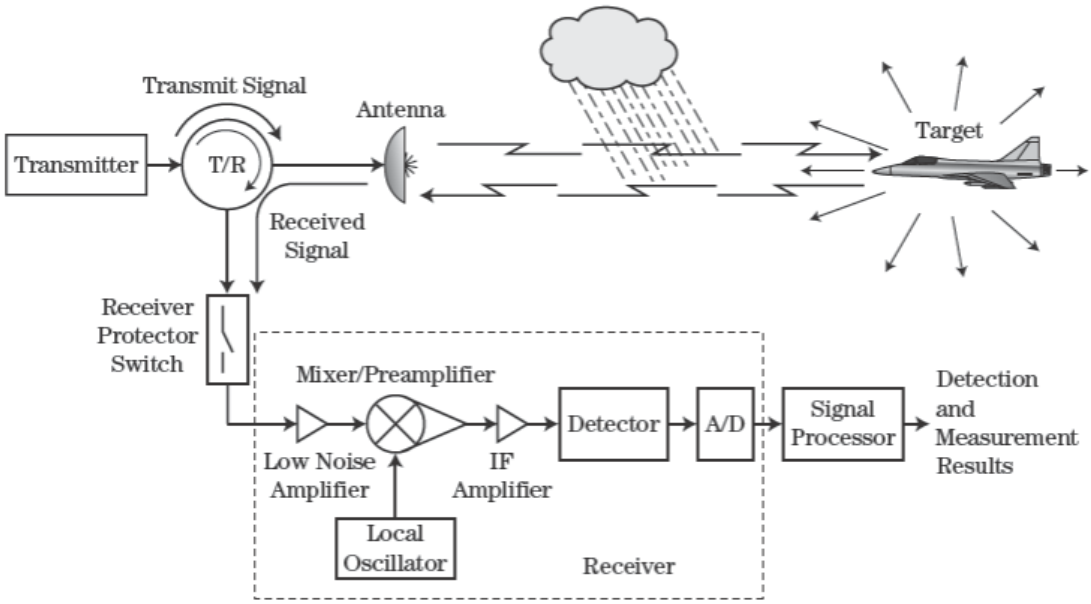


Figure 1.1: Radar transmission and reception process. [4]

The theory behind this principle is, however, very complex, as it involves a wide range of disciplines such as microwave engineering, electrical engineering, mechanical engineering and advanced signal processing techniques.

Each functioning radar system is designed and implemented based on these three basic principles:

1. The reflection of electromagnetic waves.

   If an echo wave is received back, it means there is a reflective object in the direction of propagation.

2. The constant travelling speed of electromagnetic waves through air (approximately the speed of light in vacuum $c \approx 3\text{x}10^8$ m/s).

   The distance to the object can be determined by measuring the time between transmission and reception.

3. The wave usually travels through atmosphere in a straight line, if not deviated by weather conditions.

   The wave can be directed to a specific location, therefore the direction of the target can be found.

The spectrum of electromagnetic waves is very large (up to 10^{24} Hz), therefore it was divided into several bands due to the different properties of interaction with the medium. There are several designation systems used for frequency bands, defined by convenience and for different applications. Figure 1.2 shows an overview of the the radar frequency ranges for both IEEE (industry) and NATO (military) band designations.

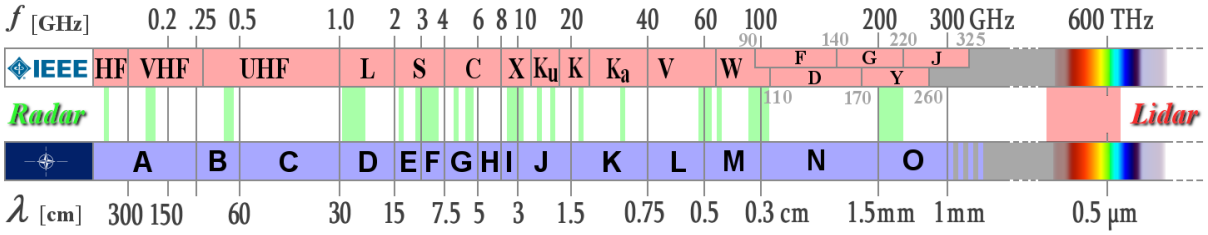


Figure 1.2: Radar waves and frequency ranges with the radar allocated sub-bands (green) [5].

The IEEE designation system [6] originates from the second world war, when the operating frequencies were intended to be kept secret, hence the inconsistent order of letters.

The NATO system is a new designation system which tailors the frequency band boundaries to the existing technologies and measurement capabilities in different frequency ranges.

The automotive radars that are currently in use operate in the range of 24 GHz, 60 GHz and 77 GHz. The frequency ranges that are of interest in this project are above 60 GHz, which correspond to automotive radars and low-THz radars. The low-THz (or sub-THz) radars operate between 0.1 THz - 1 THz.

The radars with the highest frequency (over 200 THz) are laser systems called LiDARs (light detection and ranging), usually used as imaging sensors for mapping the environment into point clouds (3D models).

1.1.2 Automotive Radar

Automotive radar is currently a key technology for providing autonomous features in modern vehicles, because radars are effective in difficult weather conditions and can penetrate foliage. Automotive radars can be mounted on the front, sides and rear of the vehicle to scan different directions simultaneously. These can provide intelligent functions to significantly reduce the driver's stress by alleviating the monotonous, repetitive tasks. They also provide comfort features and improve the safety of the vehicle by automatically taking over in crucial, life saving situations.

An example of such system is Automatic Emergency Braking (AEB) which is one of the most important recent developments in terms of road safety and collision avoidance. Researchers at the University of Adelaide predicted that AEB could reduce fatal crashes by 20-25% and injury crashes by 25-35% [7]. Another example of intelligent automotive system is the Adaptive Cruise Control (ACC) that automatically adjust the acceleration and braking of the vehicle.

The International Telecommunication Union (ITU) document [8] anticipates that by 2030 the automotive radars will be used in 65% of the cars in Europe and 50% in the United States. Automotive radars fall into two main categories, depending on their ranging possibilities and safety demands [9]:

1. Category 1: for long distances up to 250 m, used in applications such as ACC and AEB;

2. Category 2: for short and medium distances up to 50 - 100 m, used in applications such as lane change assistance, rear cross traffic alert, backup aid and rear crush collision alert.

Automotive radars are typically Frequency Modulated Continuous Wave (FMCW) radars operating in the mm-wave spectrum, where the wavelengths for these frequencies are under 10 mm. As mentioned before, the established radar operating frequencies for automotive radars are 24 GHz, 60 GHz and 77 GHz. Currently, there are trends to move higher in frequency up to 300 GHz.

The *radar equation* is known in literature for describing the power of the received signal, reflected by the target. The modified mm-wave radar equation, which includes an additional factor η to account for losses due to atmospheric absorption is [10]:

$$S = \frac{P_0 \eta G_t}{4\pi R^2} \times \frac{\sigma_c}{4\pi R^2} A_e, \tag{1.1}$$

where the first operand denotes the incident signal and the second operand denotes the reflected signal. S is the received power, R is the distance to the target (range), P_o is the radar transmit power, G_t is the transmit antenna gain, A_e is the effective area of the receive antenna, and σ_c is the Radar Cross Section (RCS) of the target.

The radar equation can be rewritten as:

$$S = \gamma_1 \gamma_2 P_o R^{-4} \eta, \tag{1.2}$$

where

$$\gamma_1 = \frac{G_t A_e}{4\pi} = G_t G_r (\frac{\lambda}{4\pi})^2, \tag{1.3}$$

where λ is the wavelength of the signal, G_r is the receiver antenna gain and

$$\gamma_2 = \frac{\sigma_c}{4\pi}. \tag{1.4}$$

A typical automotive scenario that illustrates the working principle of a mm-wave radar is shown in Figure 1.3.

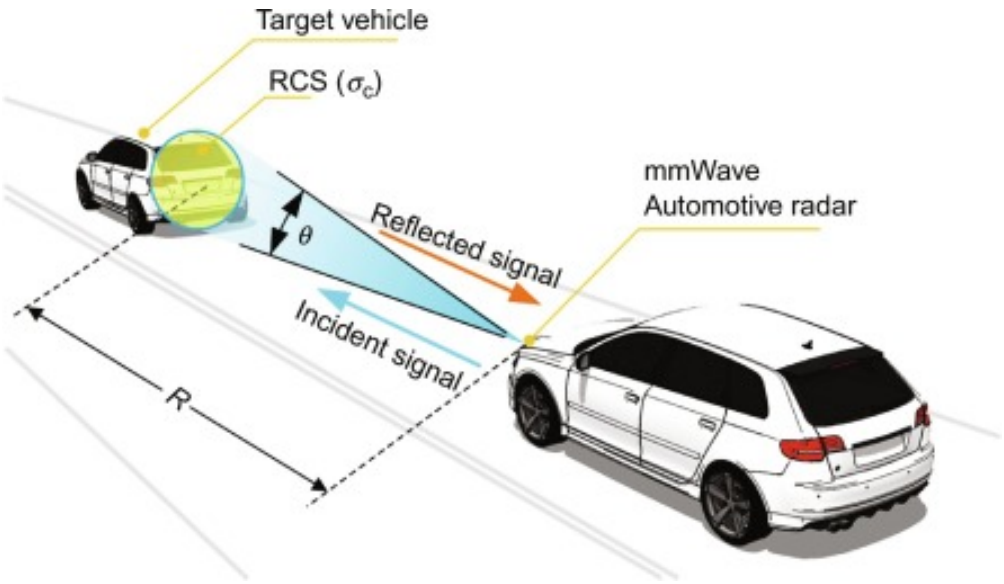


Figure 1.3: Typical automotive scenario for a mm-wave radar sensor [10].

1.1.3 Low-THz Radar

Radars that operate in high frequency bands can produce highly detailed radar imagery and they are used for applications at very short ranges, due to the low RCS of the targets. For example, mm-wave airport security scanners are used for luggage and human scanning, aiming to find metallic and non-metallic threats concealed beneath clothing [11], or scanning heavy goods vehicles for detection of illegal passengers [12].

Currently, research is being carried out in the band referred to as low-THz, or sub-THz, with frequencies between 100 GHz - 1 THz. This is promising for automotive applications, due to the imaging capabilities provided by the radars functioning at these frequencies [13]. This emerging technology provides a high resolution imaging system that can complement the existing automotive sensors.

Low-THz radars are suitable for automotive imaging radar because they can produce highly detailed imagery due to diffuse surface scattering, so a signal will be scattered from the whole area of an extended target, allowing for the potential reconstruction of a full object. Therefore, low-THz radar can provide a finer resolution imagery when compared to other mm-wave radars and can facilitate the detection and tracking of road objects. The highly detailed radar imagery allows for computer vision and state of the art deep learning techniques to be implemented for image analysis.

Throughout this project, two LFMCW radars are utilised. One of them is a low-THz radar prototype with a frequency of 300 GHz and multiple receivers, being used in extensive research, such as low-THz automotive 3D imaging [14] and low-THz signal attenuation due to rain, ice, snow, leaves, sand and antenna radome [15-21]. In this project, the low-THz imagery was used for classification and object detection in controlled environments, by virtue of its highly detailed radar imagery.

1.2 Sensing and Perception in Autonomous Vehicles

People have been enthusiastic about autonomous cars since the early 20^{th} century. The idea of full autonomy, with the user only setting the destination, while the vehicle completes the journey by itself, appears to be very exciting nowadays, considering that important advances have been made in this direction recently.

This section will present an overview on how autonomous vehicles work and how do they "see" the outer world.

1.2.1 Technology and Sensors

Autonomous vehicles need sensors to function and complement (or fully replace) the human's driving capabilities. This section will outline the major types of sensors that allow vehicles to sense their surroundings.

LiDAR

LiDAR, or 3D laser scanning and mapping, is an important tool that allows a self-driving vehicle to scan the environment using lasers. The functioning principle of a LiDAR is very similar to Radar: it transmits high frequency energy pulses and listens to the reflected beams from the surrounding objects. Usually, the LiDAR wavelength is mainly located in the near infrared part of the electromagnetic spectrum, 750 nm to 1.5 µm, hence a frequency range between 200 THz - 400 THz. The received signals are reconstructed into point clouds, which represent the surrounding environment in 3D space. Although LiDARS can detect small objects with high accuracy, they are not as effective in bad weather conditions such as

heavy rain, snow and low hanging clouds, due to the effects of atmospheric attenuation.

Radar

As mentioned in the previous section, automotive radars are indispensable sensors for self-driving cars, that provide essential information about the ranges, angles and velocities of surrounding objects. Radars function well in most weather conditions and over long distances. However, the shortcomings of radars are false detections and the difficult (if not nonexistent) object identification. In this project, research is being carried out for mitigating these issues with the aid of emerging low-THz radars and computer vision techniques.

Optical Cameras

Optical cameras are crucial sensors for autonomy, they allow the vehicle to visualise the surroundings in ways that resemble the human vision. For computer vision and object detection algorithms, optical cameras are probably the best automotive sensors in terms of the data provided. Deep neural networks can be trained for object detection in urban scenes and there are several publicly available datasets, such as ImageNet [22], KITTI [23] and Pascal VOC [24]. Unfortunately, optical cameras are not that efficient during night time and adverse weather conditions, so other sensors, most probably radars, must take over in such situations.

Ultrasonic Sensors

Ultrasonic sensors are used for echolocation, they can passively listen for sound waves from nearby objects, or measure the distance to objects. They can also detect an approaching or receding object by transmitting sound pulses and listening for the echoes. The disadvan-

tage of ultrasonic sensors is that they are limited by the speed of sound and can have a high false detection rate.

Sensors for Positioning and Location

1. **Inertial Navigation Sensors**

   Inertial Measurement Unit (IMU) is used to determine the acceleration, heading angle, and relative position of a vehicle, using a combination of gyroscopes, accelerometers and sometimes magnetometers. By detecting the physical movements of the car, safety features can be triggered, such as airbags. It can also stabilise the vehicle and prevent it from rolling over.

2. **Global Positioning System**

   Global Positioning System (GPS) is a satellite navigation system owned by the US government and it currently has 31 active satellites. GPS is the most popular navigation system for commercial applications, from smartphones to self-driving cars. Both geolocation and time information from atomic clocks can be obtained by users if they have a GPS receiver. Depending on factors such as the available satellites or reflections, a GPS location can vary by 5-metre radius, although the accuracy of the location can now be improved by advanced processing techniques such as particle filters.

The process of perception in autonomous vehicles to sense the outer environment is based on a fusion of information from all mentioned sensors, combined with real-time state of the art processing algorithms. Data from multiple sensors is fused to produce more accurate decisions and reduce the uncertainty. Figure 1.4 shows an example of sensory output in a driving environment: an optical image of a multi-storey car park entrance and the associated LiDAR 3D point cloud (black) and radar image (colour), obtained in the

MISL at the University of Birmingham.

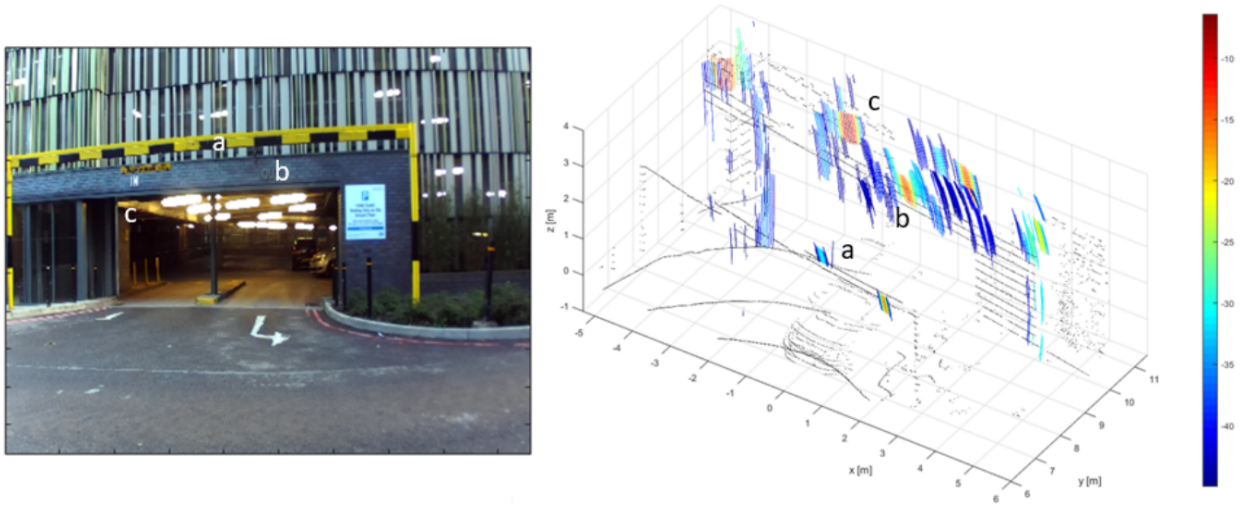


Figure 1.4: Left: optical image; Right: 3D reconstruction with lidar point-cloud in black and radar image in colour [25].

1.2.2 Applications

Although the fully autonomous vehicle is not on the road yet, there are already semi-autonomous vehicles available. The autonomy of a vehicle can take many forms, depending on the requirements of the manufacturer, the costs and the market's demands. The technology that assist drivers and increase vehicle and road safety is called Advanced Driver-Assistance Systems (ADAS). The first ADAS system, the Anti-lock Braking System (ABS), dates back to the 1950s. ADAS differs from Driver-Assistance Systems (DAS) because it takes into account the surrounding environment of the vehicle. Modern vehicles nowadays have ADAS integrated into their electronics. ADAS is categorised into several levels, based on the autonomy that it provides to the vehicles.

Vehicle automation is grouped into 6 levels (from 0 to 5) by the Society of Automotive Engineers (SAE) [26], illustrated in Figure 1.5. Levels 0-2 correspond to low to moderate autonomy and the vehicles are called "traditional", because the user is still required to drive and the driving support systems only provide assistance, such as lane departure warning or cruise control. Levels 3-5 are considered highly autonomous, especially levels 4-5, where human input is not required. The vehicle interprets the environment and pedals and steering wheels do not need to be installed anymore.

Figure 1.5: SAE levels of driving automation [23].

The levels of automation are briefly described, as follows:

1. Level 0 - No driving automation: the driver support features are limited to providing warnings and momentary assistance, such as Automatic Emergency Braking (AEB), lane departure warning and blind spot warning.

2. Level 1 - Driver assistance: the driver support features provide steering or brake/acceleration support, such as Adaptive Cruise Control (ACC) and lane centering.

3. Level 2 - Partial driving automation: driver support features provide steering and brake/acceleration support. It is basically level 1, but ACC and lane centering are provided at the same time.

4. Level 3 - Conditional driving automation: automated driving features might request the user to drive, so this is not a fully autonomous level. These features can drive the vehicle under limited conditions and the required conditions must be met. An example of application is the traffic jam chauffeur.

5. Level 4 - High automation: this is the first level of high autonomy - the automated driving features will not require the user to take over driving, pedals and steering wheel may not be installed. However, similarly to level 3, the features can drive the vehicle under certain conditions.

6. Level 5 - Full automation: this is the highest level of driving automation - the user will not be required to drive and the automated driving features can drive the vehicle under all conditions.

Examples of other ADAS technologies (apart from AEB, ACC and lane change assist) that have become commonly available in the last decade are [27-34]:

1. Automatic parking: uses parking sensors, such as proximity sensors and video cameras to fully take over parking for the driver.

2. Backup camera system: working in fusion with proximity sensors, provides audio warning and a view of the rear of the vehicle and the potential blind spots, when the driver puts the car in reverse.

3. Automotive navigation system: uses GPS to detect the location of the vehicle and provide real-time information about the traffic and navigation.

4. Collision avoidance system: use radar detectors to inform the driver in the event of a potential crash by sounding an alarm and can also take other actions, such as raising the reclined seats or tensing up the seat belts.

5. Crosswind stabilisation: prevents a vehicle from overturning in the event of strong winds, by analysing the velocity, the steering angle and the lateral acceleration of the vehicle.

6. Driver monitoring system: or driver drowsiness detection, is designed to prevent collisions by monitoring the alert levels of the driver, using information such as steering patterns, facial expressions, eye tracking, driving velocity and other indicators of human fatigue.

7. Hill assist: hill start or descent control, it assists with preventing a vehicle from rolling backward down and with maintaining a safe speed when driving down a hill, by holding the brake and by controlling each wheel independently.

8. Intelligent speed adaptation (ISA): assists drivers by notifying them if the speed limits are exceeded and if the user requires, by automatically adjusting the speed as well, to comply with the speed limit.

Challenges

At the moment, there are several automobile manufactures, such as Tesla, Honda, Mercedes and AUDI that offer vehicles in the SAE level 3 of autonomy, or even in SAE level 4 (Waymo's commercial taxi service in Phoenix). Although autonomous vehicles can contribute to our safety and revolutionise the road traffic, there are several restrictions or conditions that need to be fulfilled for accomplishing the final goal of full autonomy.

Realistically, although significant progress has been made so far, fully autonomous vehicles in SAE level 5 will not be commercialised too soon, because there are still challenges that need to be addressed. The sensing system must be able to cooperate smoothly with the vehicle and the environment. Also, a powerful and reliable communication network is needed to be able to transmit large amounts of data at high speeds, in all weather conditions and without interference. While the technology is still developing, legal and ethical issues should be taken into consideration too. An autonomous vehicle will be impractical if it behaves individually, hence the key to autonomy is a cooperative environment (between vehicles, passengers, the cloud, Internet of Things (IoT) devices, the road infrastructure, etc.). Autonomous vehicles are a complex task and will likely not be feasible for public use without having any strategies for cooperative traffic management in place and a driving environment adapted to self-driving vehicles [35].

1.3 Artificial Intelligence

Artificial Intelligence (AI) is a computer science field involved with automation of human intelligent behaviour [36]. The term "intelligence" has not yet been fully understood, hence the architectures of profound human actions, such as learning, reasoning, creativity and intuition are not entirely known. Therefore, research in the field of AI is accelerating, with the purpose of designing methods and algorithms that resemble as much as possible with human and animal approach to reasoning.

Some of the AI technologies that were successfully developed include machine learning, neural networks, fuzzy logic, genetic algorithms and evolutionary computation. These methods are also known as "soft computing", term introduced by Lotfi A. Zadeh [37] at the University of California and according to him, "soft computing differs from hard computing (conventional computing) in its tolerance to imprecision, uncertainty and partial truth". In other words, AI algorithms are prone to imprecision, like the human mind and trial and error method is often used, while the hard computing algorithms require a high level of precision and mathematical approach.

According to [38], the most important AI technologies can be seen in Figure 1.6. Neural networks, fuzzy systems and evolutionary algorithms can also be combined to obtain hybrid intelligent systems, that are usually more efficient, by comparison with each algorithm used individually.

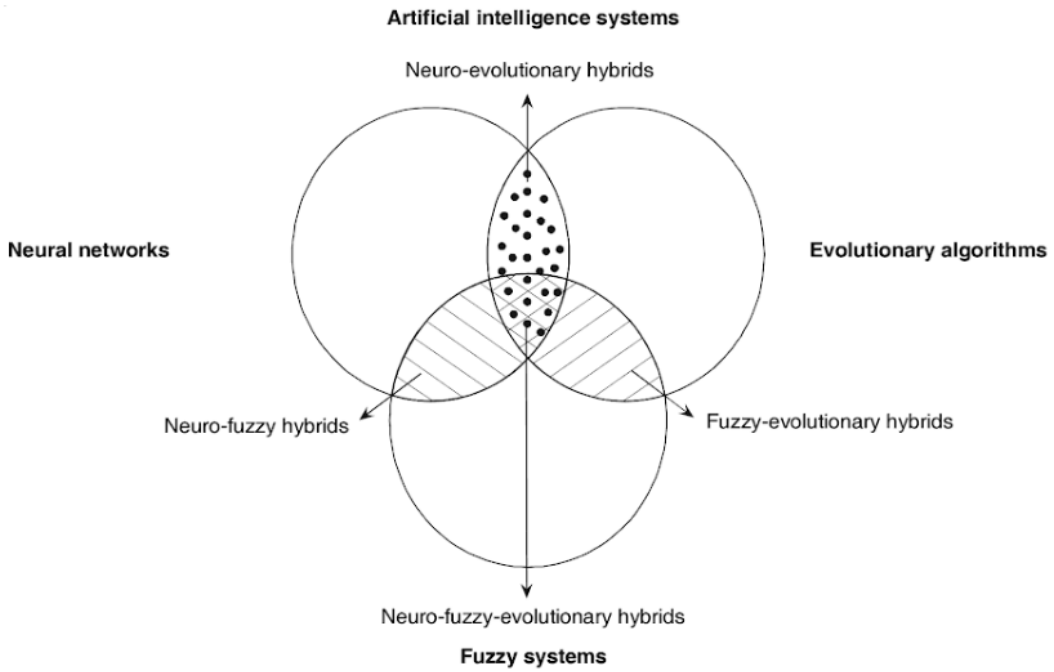


Figure 1.6: Integration of neural networks, fuzzy systems and evolutionary algorithms techniques [38].

The taxonomy of AI is shown in Figure 1.7. Machine Learning (ML) is a subset of AI that deals with algorithms that are not programmed for specific tasks, but rather learn from observational data. Neural Networks (NN) is a brain-inspired, revolutionary subfield of ML, which spawned Deep Learning (DL), a state of the art field of NNs with deep architectures. DL has become popular recently with the hardware advances and the switch from Central Processing Unit (CPU) programming to parallel programming using powerful Graphics Processing Unit (GPUs) [39].

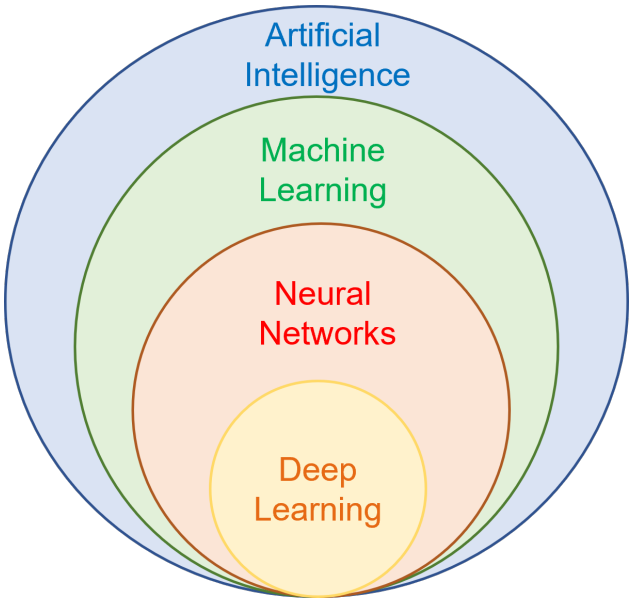


Figure 1.7: AI taxonomy: ML, NN and DL.

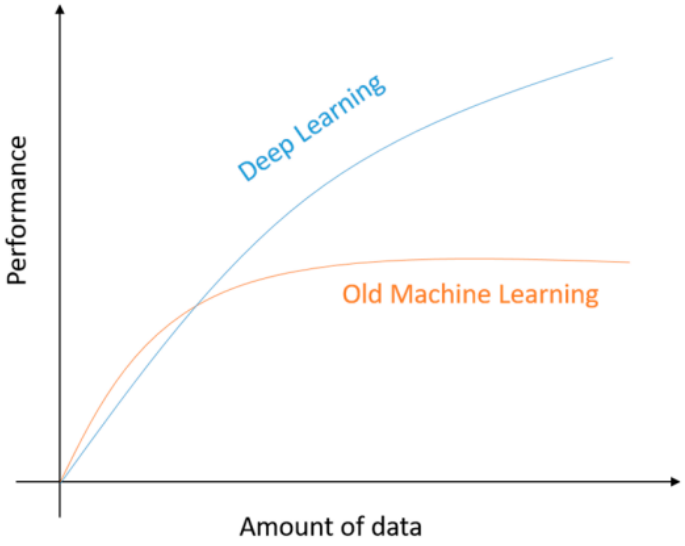


Figure 1.8: The performance of deep learning versus traditional ML depending on the amount of data [40].

DL can deal better with large amounts of data, when compared to traditional ML techniques. The advantages of DL in big data problems are demonstrated in [41-43]. Figure 1.8 shows that the old ML learning methods are more effective than DL for lower amounts of input data, although the performance becomes steady if the amount of data increases. On the other hand, the performance of DL algorithms increases directly with the size of the input data.

This project will focus on neural networks for image classification and object detection, therefore different types of neural networks that are feasible for this kind of application will be detailed in the following chapters, such as Convolutional Neural Networks (CNNs), Siamese Neural Networks and deeper networks (object detectors), along with their functionality, parameters, architecture considerations and results. Next, the concept of ML and the most popular methods for image recognition using DL will be defined.

1.3.1 Machine Learning

Machine Learning (ML) is one of the core techniques for data analytics, forecasting and classification. ML algorithms focus on learning patterns in input data and then generalise on unseen data. The result of a ML algorithm is known as *the model* and it represents the learning outcome of the algorithm that has been trained on input data. The model consists of parameters that can be adjusted or refined for better learning. As seen in Figure 1.9 below, the ML algorithms can be grouped in three main categories, based on the learning method: supervised learning, unsupervised learning and reinforced learning.

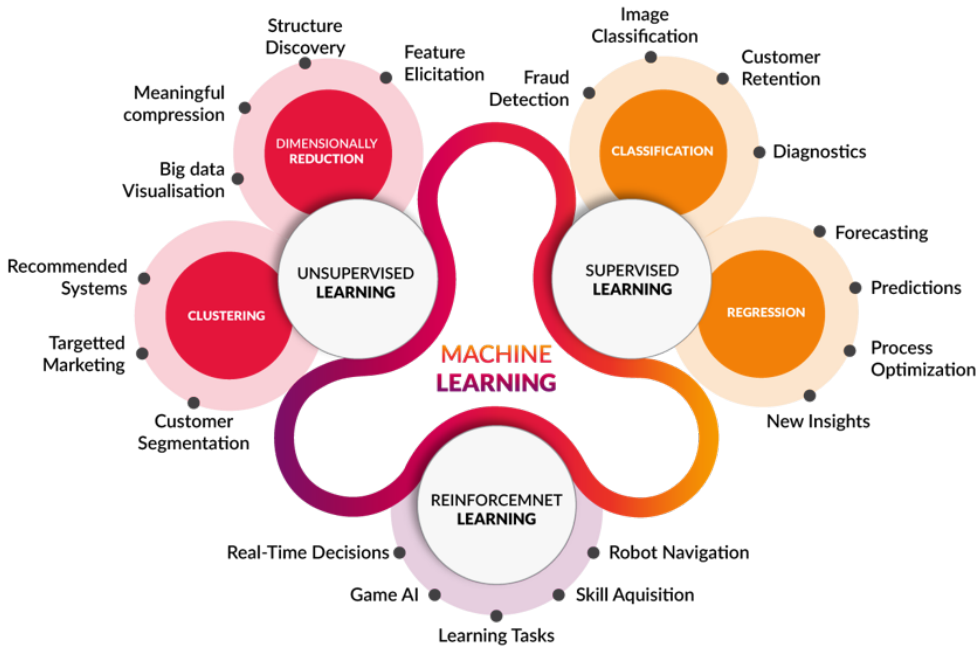


Figure 1.9: Branches of ML with examples of common applications: supervised learning, unsupervised learning and reinforcement learning [44].

1. **Supervised Learning**

   Supervised learning algorithms require labelled data, a set of inputs with corresponding outputs, which are labelled by users. There are two main supervised learning

techniques: classification and regression. For classification, the input is assigned a category and the desired classes are known beforehand. Examples of applications include image classification with CNNs, fraud detection and customer retention. Regression is used in applications where numerical values need to be predicted, such as forecasting in marketing and weather or predicting house prices.

2. **Unsupervised Learning**

   Unsupervised learning systems do not require labelled data, they learn patterns and important features to form relationships between the input data and model an internal structure. Unsupervised learning is mostly used for clustering and dimensionality reduction. In this method there are no correct answers, this approach can be used to discover unknown data patterns in consumer behaviour for targeted marketing, or to compress the input data for finding only the most important features. Some examples of such algorithms are Autoencoder (AEs) [45], K-Means Clustering [46], Restricted Boltzmann Machines (RBMs) [47] and Recurrent Neural Networks (RNNs) [48].

   There are also semi-supervised learning methods that require partially labelled datasets, such as the novel Generative Adversarial Networks (GANs) [49].

3. **Reinforcement Learning**

   Reinforcement learning techniques are used when the environment is unknown. The idea is to reward the model when a positive action is taken, otherwise punish it. Rewards and punishments have numerical forms and the model can use them to update its parameters (learn) and perform better. This approach is often used for robot navigation in finding the best path, or in Game AI (for example Google DeepMind's Deep Q-learning playing Atari Breakout [50]).

1.3.2 Deep Neural Networks

Neural networks have become fundamental tools for many applications such as image processing, image segmentation, pattern recognition, efficient web surfing, self-driven systems, speech recognition, etc. By using such machine learning techniques, algorithms no longer need to be programmed for specific tasks, but are capable of learning from previous computations and make predictions and decisions on their own, based on a set of input data that is processed for a particular task.

Deep Neural Networks (DNNs) originated from Artificial Neural Networks (ANNs), therefore both share the same main components of a basic neural network, as seen in Figure 1.10: input layer, hidden layers (a neural network is considered *deep* if it has more than two hidden layers of neurons) and output layer. The neural network is assigning every input x a certain probability of being associated to a particular notion ω_c. Each layer is made of artificial neurons, which have a similar structure with the brain neuron, but different functionality (for example, the human brain is known to have approximately 100 billion neurons and 100 trillion synapses operating on 20 Watts, while the largest neural network developed only has 10 million neurons and 1 billion connections, operating on 16,000 CPUs, so 3 million Watts). Therefore, DNNs perform well at learning from data, although they cannot be considered a good mechanism for reasoning, yet.

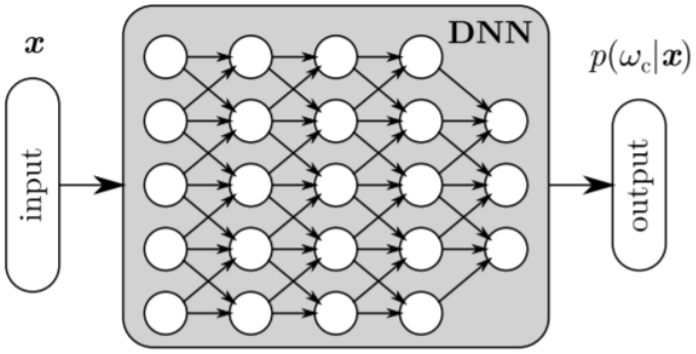


Figure 1.10: Architecture of a neural network composed of interconnected neurons [51].

Current neural networks drawbacks include:

- Supervised models require labelled data, thus real world data must be annotated first.
- The network structure has to be selected manually, depending on the application and the end goal.
- The hyperparameters (learning rate, loss function, number of training iterations, batch size, optimisation algorithm, etc.) must be tuned before training the network.

On the other hand, while the human brain only has limited types of inputs from physical senses of the body, neural networks do not have such limitation on input data. Neural networks also follow the universal approximation theorem (proved by George Cybenko in 1989 [52], using the sigmoid function), which states that for any arbitrary continuous function $f(x)$, there exists a feed-forward neural network with a single hidden layer of finite number of neurons, that can closely approximate $f(x)$ for any input x.

The process of learning for an artificial neural network functions by passing the input data through the hidden layers of units (neurons) and feed-forwarding the data to the output unit, along with feedback and backpropagation. The inputs of a neural network are diverse

and can vary from numerical data and images to sounds and words, though all types of inputs must be converted to vectors or tensors before being presented to a neural network.

In a feed-forward neural network, each neuron receives an input from the previous layer and multiplies it by the weight of the connection the information travels along. Each neuron adds up all the inputs and a bias and if the resulted value is higher than a certain threshold, the neuron $fires$ and sends information to the next layer of neurons. Therefore, not all units are active during feed-forward, which also helps to avoid overfitting the data (overfitting happens when a model learns the details and noise in the training set excessively, to the point that it is unable to perform well on the testing set).

The output of a neuron has the following equation

$$Z = f\Big(\sum_{i=1}^{n} w_i x_i + b\Big), \tag{1.5}$$

where f is the activation function, w_i is the weight of the i^{th} connection between neurons, n is the total number of connections, x_i is the i^{th} input neuron and b is a bias (a parameter which shifts the activation function to the right or left for better learning).

A neural network must include a repetition element in order to be able to learn, similar to the biological human brain. This process is called backpropagation and implies comparing the actual output of the network with the desired output and using their difference to update the weights of the connections between neurons in a reversed direction, from the output unit through the hidden layers and back to the input layer. Hence, the network is using feedback and backpropagates until the output corresponds to the expected value and the learning process is accomplished.

The more hidden layers of artificial neurons a network has, the more details it learns, therefore a deep network where each layer learns a new concept from the previous one is preferred, rather than a wide network with a large number of neurons in each layer.

The field of DNNs is continuously developing and evolving to different types of networks, hence it would be exhaustive to mention all types of DNNs available today, especially considering all the combinations between algorithms that are being made in order to enhance network's capabilities to perform particular tasks and develop hybrid systems. However, the basic types of deep neural networks that can be distinguished are: Multilayer Perceptron (MLP), AEs, CNNs and RNNs.

Firstly, this thesis will focus on CNNs, which are described in section 2.1.3, followed by one of its variants, Siamese Neural Networks in section 2.1.4, since they are widely used in applications such as image classification and pattern recognition.

1.3.3 Image Recognition Algorithms

Image recognition algorithms using neural networks have been making significant progress over the last decade. In certain applications, DNNs based image recognition algorithms can achieve even better accuracy than the human expert, especially for 2-D images. These algorithms are generally based on supervised learning and can be grouped into three main categories, illustrated in Figure 1.11: classification, object detection and segmentation.

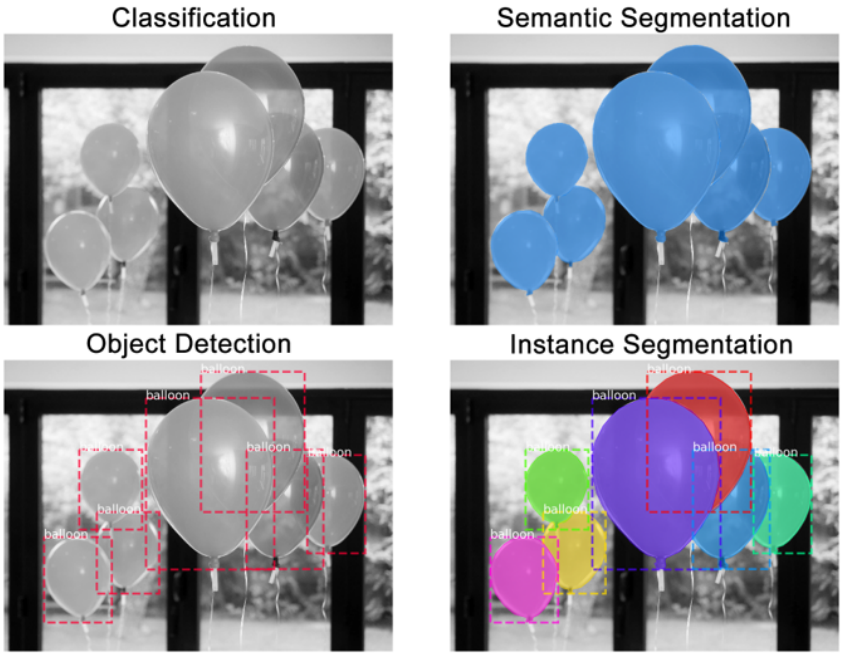


Figure 1.11: Image recognition algorithms using NNs [53].

1. **Classification**

   Classification algorithms are usually used for assigning an object into a predefined group or class, based on a number of observed attributes related to that object. Classification has a large number of applications in various fields, such as face recognition, speech recognition, handwriting recognition and medical diagnosis. A survey of neural networks for classification can be found in [54].

2. **Object Detection**

   Object detection is one of the most fundamental and challenging tasks in computer vision. Object detection algorithms are basically advanced classifiers, based on deeper neural networks, which can detect multiple instances of objects in the same image and also indicate the locations of the objects, by including them in rectangle bounding boxes. A survey of object detection over the past 20 years can be found in [55].

3. **Segmentation**

   Segmentation, also known as pixel level classification, is used for finding the shapes of the objects. There are two types of segmentation algorithms that can be distinguished: semantic segmentation, for classifying each pixel of the image into a class, and instance segmentation, where individual objects of the same class are distinguished. A review of semantic segmentation based on neural networks is provided by Guo et al in [56].

In this thesis, Chapter 2 will be focused on radar image classification using CNNs, while Chapter 3 will describe the radar object detection using two performant detectors, Faster R-CNN and SSD.

1.4 Motivation and Contribution

1.4.1 Aim

One of the major difficulties that the field of autonomous vehicles currently faces is the lack of technology for object detection during adverse weather conditions. Automotive radars partially solve this issue, since radars sensors can be effective in all weather conditions and at night-time. However, the downside is that objects are difficult to be identified in the current automotive radar imagery.

The radar department at the University of Birmingham, Microwave Integrated Systems Laboratory (MISL) is one of the biggest radar academic teams in the UK, carrying out state of the art research in radar and remote sensing for autonomous vehicles. One of the research topics of the group is the emerging low-THz radar, a promising solution that should be further explored, considering that the radar imagery at very high frequencies is highly detailed and has finer resolution.

Overcoming the difficulties of object identification in radar imagery has the potential to revolutionise the current approaches in the field of autonomous vehicles and make this technology available for the public sooner. Therefore, the aim of this project is to investigate novel deep learning techniques, based on neural networks, for automotive radar image classification, object detection and tracking in high resolution radar imagery.

The following research questions will be addressed throughout the thesis:

- Achieve high accuracy classification of several roadside targets using low-THz radar imagery and deep neural networks, thus proving that the low-THz radar is a valuable component for the development of future autonomous vehicles;
- Perform object detection on indoor and outdoor automotive high frequency radar imagery to classify and detect the location of several roadside targets;
- Propose and implement a novel method for improving the object detection accuracy by combining it with a particle filter tracker and demonstrating the performances of the system on radar frames of moving pedestrians.

1.4.2 Publications

Throughout this project, three notable research papers were published, that correspond to the chapters of this thesis, in the order of publication. Two of these publications have received awards at the international radar conferences where they were presented by the author of this thesis. A journal paper on the survey of the deep learning methods used in this project for radar imagery recognition is being prepared.

1. **Classification** [57]: Ana Stroescu, Mikhail Cherniakov, Marina Gashsinova, *Classification of High Resolution Automotive Radar Imagery for Autonomous Driving Based on Deep Neural Networks.*, presented at the International Radar Symposium, Ulm, Germany 2019.

   This paper was awarded the *Best Paper Award* at the IRS Conference and was nominated for the *Young Scientist Award*.

2. **Object Detection** [58]: Ana Stroescu, Liam Daniel, Dominic Phippen, Mikhail Cherniakov, Marina Gashinova, *Object Detection on Radar Imagery for Autonomous Driving Using Deep Neural Networks.*, presented at the European Microwave Week, EuRAD 2020, Utrecht, the Netherlands.

3. **Combined Object Detection and Tracking** [59]: Ana Stroescu, Liam Daniel and Marina Gashinova, *Combined Object Detection and Tracking on High Resolution Radar Imagery for Autonomous Driving Using Deep Neural Networks and Particle Filters*, presented at the IEEE Radar Conference, Florence, Italy, 2020.

   This paper was awarded the 3^{rd} place at the *Student Paper Competition*.

Chapter Two

Image Classification

2.1 Introduction

Autonomous vehicles are now the cutting edge innovation in automotive industry. The fully autonomous driving technology is already used in controlled environments such as farming, though it is not available for customers and on-road vehicles yet. Currently, there are two ways for providing autonomy: GPS navigation and sensors for comprehensive optical avoidance. Information from sensors is combined for a robust classification of objects. Research in the field of self-driving vehicles is currently intensively carried out by automotive companies, so that better technologies can be developed for on-road use. Although prototypes exist and tests are being performed on private roads, there is a long way until these technologies will be fully reliable for day to day operation. It is predicted that autonomous vehicles will become the primary means of transport by 2050, on-highway trucks being the first to feature the technology on public roads, by 2040.

Some of the numerous advantages that autonomous vehicles are expected to offer are reducing the accidents by 90%, lowering pollution by cutting the carbon dioxide emissions by 60% as a result of an optimised driving style and increased traffic management, providing

an overall better experience and a higher level of safety and comfort to passengers. Researchers also predict that an Internet of Vehicles (IoV) network will be developed, similar to the Internet of Things (IoT) that we are currently witnessing, allowing the vehicles to communicate and share sensor data in real time, such as road and weather conditions [60].

However, video cameras and lidar cannot perform well during adverse weather conditions, therefore radar imagery is preferred in situations when video and lidar may fail. The aim of the work presented in this chapter is to develop a DNN which can classify objects in low-THz band radar imagery. The neural network will be able to distinguish between different types of roadside targets that were used in the experiment: bicycles, trolleys, mannequins, dogs, traffic cones and traffic signs. The radar object classification performed by the DNN, together with information received from other sensors (lidar, stereo-video), will have the potential to produce a detailed map of the surroundings and an all-weather sensing system. This chapter is focused on presenting the process of radar image classification, from methodology to data collection and the classification results.

2.1.1 State of the Art

DNNs are being used in many applications for semantic urban scene segmentation, based on optical images captured by video cameras which are mounted on vehicles. Large scale, public datasets of urban scenes, such as ImageNet [22], KITTI [23], Pascal VOC [24] and Cityscapes [61] have sustained the development of neural networks for scene recognition. However, these datasets are not large enough to capture the full complexity of the real world traffic environment and exploit the full potential of neural networks in this field, yet.

Despite the significant progress that has been made recently in this domain, the visual scene understanding still remains a challenging issue for autonomous driving industry [61]. An example of the current state-of-the-art image annotation using DNNs on the Cityscapes dataset can be seen below, in Figure 2.1. The image shows the results of a semantic segmentation algorithm (or pixel level classification), where the neural network was trained to recognise different types of objects and classify each pixel into a class. For a better visualisation, entity classes are associated with a colormap (dark blue - vehicles, green - vegetation, light blue - sky, grey - buildings, purple - roads).

Figure 2.1: Example of Cityscapes annotations [61].

However, optical images may become unusable in bad weather conditions, therefore radar imagery is preferred. State of the art radar cannot ensure good resolution in order to enable such imagery, hence low-THz radar has been proposed as a potential candidate to provide this object classification capability for automotive systems [62]. The main advantage

that low-THz radar systems have over the current automotive mm-wave radars is that they provide larger bandwidths, hence a finer range resolution. The very small wavelengths cause diffuse backscatters from all over the object's surface, which provide much more information about the profile of the target, such as texture, curvature and depth of shadow [63]. Therefore, the detail provided by low-THz radar images is more similar to optical/video images, facilitating the deployment of electro-optical object classification algorithms in automotive radar applications.

At present, attempts have been made to classify radar and lidar images [64-65], but a fully reliable system that annotates the entire vehicle's environment based on these images has not been developed yet. The majority of research into neural networks for radar imagery at the moment is being carried out for Synthetic Aperture Radar (SAR), such as noise reduction in SAR images [66] and target classification [67-69] using CNNs, but there is little evidence for research on object recognition in low-THz radar imagery for autonomous vehicles, thus far. To the knowledge of the author, the current work is a novel radar image classification for autonomous driving and it was presented in 2019 at the International Radar Symposium [57]. The paper has received great interest from specialised audiences and was awarded "The Best Paper" award.

This chapter will focus on a method for classification of six different targets in low-THz radar imagery, using DNNs. The proposed method, along with the results of different types of networks and their hyperparameters tuning process will be described hereinafter.

2.1.2 Classification in Machine Learning

Classification is often considered "the building block" of machine learning and it is a statistical method which aims to group/categorise a specific set of data into different classes. This method uses a supervised learning approach, meaning that the data is classified into certain classes that are pre-defined by the user. Training is accomplished by using a sequence of training vectors, each with an associated target output vector. The weights are adjusted according to the learning algorithm [70]. Usually, classes are referred to as labels. On the other hand, unsupervised learning methods are used when the data is unlabelled by the user and the model is forced to build patterns on the input data and find structures on its own. This means that a sequence of vectors is provided, though no target vectors are defined. In neural networks, the model modifies the neuron weights in order to build clusters. Most popular unsupervised machine learning algorithms are Principle Component Analysis (PCA), clustering with k-means and AEs. The key differences between supervised and unsupervised methods are stated in Table 2.1 below.

Table 2.1: Supervised vs unsupervised methods.

Criteria	**Supervised**	**Unsupervised**
Input Data	Data is labelled	Data is not labelled
Number of classes	Known	Unknown
Output variables	Are given	Are not given
Process	Finds a connection between input and output	Finds patterns in the input data
Accuracy	Very accurate method	Less accurate method
Complexity	Less complex	Very computationally expensive
Methods	ANNs, Regression, Support Vector Machine (SVM), Random Forest	Clustering, K-Means, AEs, PCA

Only supervised methods will be used throughout this project, as the discreet classes are already known. The data is labelled by the user and the aim of the classification model is to learn how the input data is related to the classes and then generalise on unseen, test

data. Nevertheless, a supervised model is not very practical in the complex real-world traffic scenario because it is difficult to account for each and every object in the environment. Thus, a combination of supervised and unsupervised methods is preferable, to classify the most common targets (particularly the moving targets, such as pedestrians and other vehicles, that might have unpredictable behaviours), as well as to detect and cluster other road-side targets, especially the ones that might represent a risk of collision, but are not labelled.

Classification methods are most commonly used for applications such as face detection, speech recognition, market forecasting, document classification, identity fraud detection, handwriting recognition etc. Based on the output, there are two main types of classifiers:

- Binary Classification - there are only two outputs (i.e. true or false). For example, medical applications such as melanoma detection using CNNs, where the outcome is binary (melanoma is either present or not in an image). Other example of binary classification is the Siamese Neural Network, presented later in Section 2.1.4, which classifies two inputs based on their similarity (binary 1 - similar objects, binary 0 - different objects).
- Multi Class Classification - there are more than two classes. For example, a six-class CNN is used in this project for detecting six types of roadside targets.

The most popular ML methods for classification tasks are Regression, SVMs, Random Forests and ANNs, briefly described below.

- Regression Analysis - a statistical method mostly used in financial applications for finding the relationships between a known variable and several others independent variables. Regression can be logistic or linear (also multilinear and non-linear), depending on the type of relationship between the variables. For example, the equation

for the multilinear regression is:

$$Y = a + bX_1 + cX_2 + dX_3 + \epsilon, \tag{2.1}$$

where Y is the dependent variable, X_1, X_2 and X_3 are independent variables, a is the intercept (the point where the graph of the equation intersects the axis), b, c and d are the slopes, and ϵ is the residual error. While linear regression is used for predicting values, logistic regression is preferred for classification tasks, since the output is categorical (binary, 1 or 0, true or false). The relationship between the dependent and independent variables is not necessary linear. A graphical comparison of linear and logistic regression can be seen in Figure 2.2.

- Support Vector Machines (SVMs) - a ML algorithm widely used for classification tasks due to its high accuracy and reduced computational cost. The aim of SVM algorithm is to find the optimal hyperplane that separates (classifies) the data. There are many hyperplanes that would successfully divide input data points, however the aim is to find the optimal hyperplane which has the maximum margin distance between points and reinforces the model, so that the points will be classified more accurately in the future.

- Random Forest - as its name suggests, this algorithm is structured as a "forest" (ensemble) of multiple decision trees, which are merged to produce a more accurate forecast. The model averages the results of several decision trees. The trees are built based on features and observations which are selected arbitrary.

- Artificial Neural Networks (ANNs) - the most advanced tools that are currently used in ML, which aim to imitate the functions and the learning process of the biological human brain, with a structure of interconnected layers of artificial neurons. ANNs have been described in the previous chapter as being the precursors of DNNs. A very important breakthrough in terms of classification with ANNs was the CNN. This is

the type of machine learning algorithm that was used in this project for classifying automotive radar imagery and the next section is dedicated solely to the description of the working principles of CNNs and their hyperparameters.

A graphical comparison between the above mentioned ML classification methods is illustrated below in Figure 2.2.

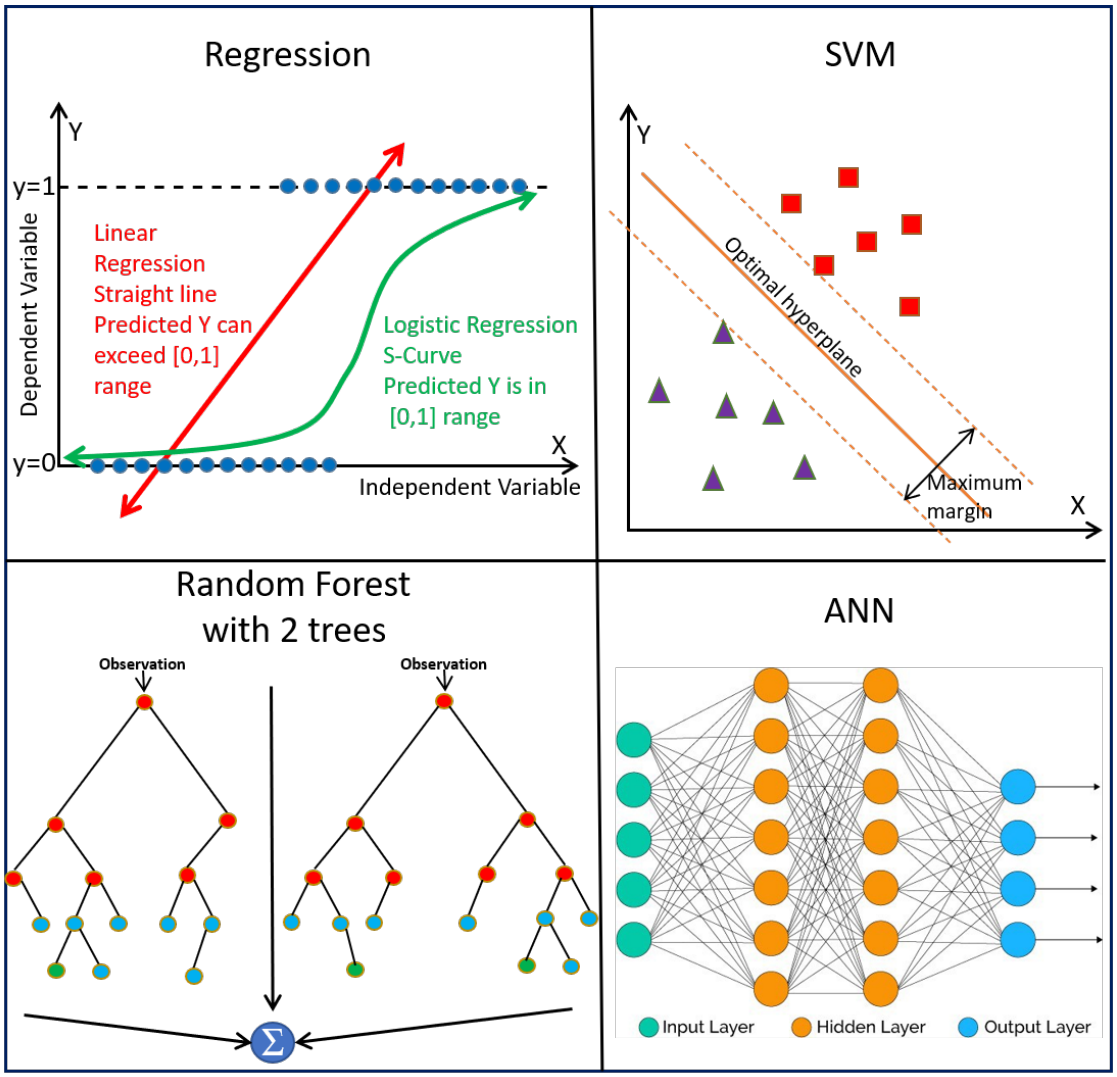


Figure 2.2: Graphical comparison between Regression, SVM, Random Forest and ANN.

2.1.3 Convolutional Neural Networks (CNNs)

CNNs are the first successful DNNs owing to their reduced complexity, decreased number of weights, and computational resources. The standard array multiplication used in neural networks is replaced by the convolution operation. Pre-processing for feature vectors is no longer required for the input data, therefore the raw images can be directly loaded into the network. CNNs have already been efficiently implemented for numerous applications, such as speech recognition, image classification, handwriting recognition, and face recognition.

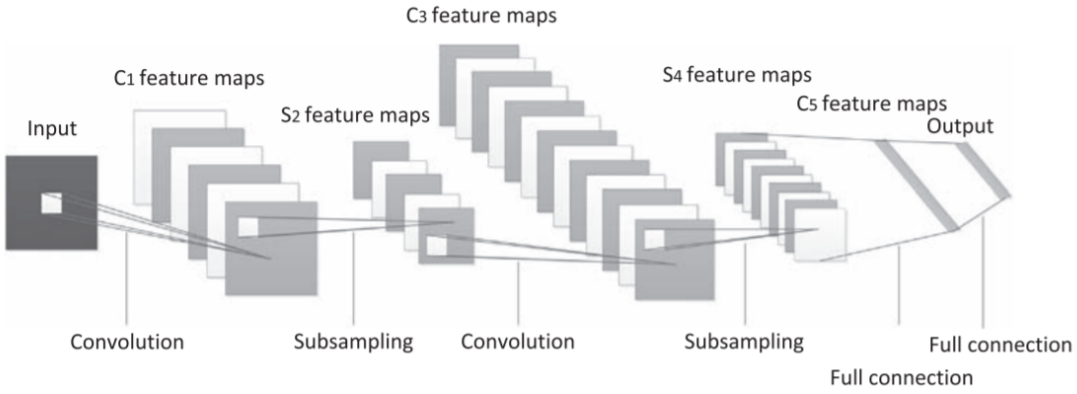


Figure 2.3: Schematic structure of CNNs [71].

As shown in Figure 2.3, the structure of a CNN consists of two main types of layers: convolution layers and sub-sampling layers. These layers are usually followed by Fully Connected (FC) layers in the end for predicting the class scores, which are standard MLP layers and can be omitted in fully convolutional networks. In the output layer, each output neuron represents a different class. Common sub-sampling layers are Rectified Linear Unit (ReLU), sigmoid, pooling and dropout. A frequently used pattern for a CNN is:

$$[CONV \rightarrow ReLU \rightarrow Pool \rightarrow CONV \rightarrow ReLU \rightarrow Pool \rightarrow FC]$$

Convolutional Layer

Convolutional layers are mainly implemented in a neural network for extracting features from images. The deeper the convolutional layer is in the network, the more complicated features it learns. The input image is represented as an array of pixel values and the convolution operation involves sliding a smaller convolution kernel (usually with dimensions 3x3 or 5x5) over the input image array and, for each position, multiply the pixel values with the convolution kernel. The resulting output feature map will have a reduced size, though it will contain the relevant information from the original image. A visual example of how the 2D convolution operation is performed over a binary (black and white) input image of size 5x5 with a convolution kernel of size 3x3 is shown in Figure 2.4.

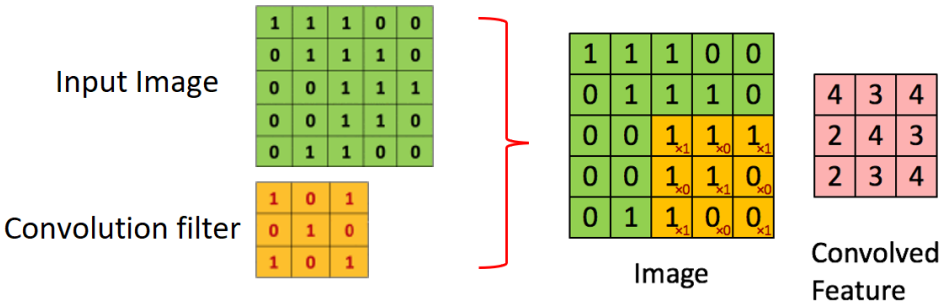


Figure 2.4: Convolution operation.

Convolution is a linear operation, therefore a non-linear activation function must be added after a convolutional layer to achieve a non-linear transformation of data. This ensures that the neural network can learn non-linear behaviour and that subsequent layers build off each other. Summing linear layers would give a linear function, hence without non-linear activations between linear layers, a neural network would behave like a single layer perceptron, regardless the number of layers. The most common non-linear functions are ReLU, which only keeps the positive values (detailed later in Section 2.2.5) and the sigmoid.

Pooling Layer

The pooling layer helps reducing the spatial dimensions of the network, thus gaining more computation performance, while retaining the most important information in the image. Also, the chance that the network will over-fit reduces significantly. The pooling kernel performs in a similar fashion with the convolutional window, except it reduces dimensions by keeping only the important information in the image and performing basic operations on the pixel values in the window (MaxPool, MinPool, AveragePool).

Hyperparameters

The parameters of a CNN (or other neural network architecture) fall into two main categories:

- Model Parameters - the parameters that are estimated by the model from the input data (e.g. the neuron weights in ANNs, the features in regression, or the support vectors in SVMs). In CNNs, model parameters are not pre-defined by the user and they are considered to be part of the final, trained model.
- Hyperparameters - the parameters that are external to the model and cannot be estimated from the input data. They are used to estimate the model parameters and they are configured by the user.

  The CNN must be tuned in order to generate the best results for a particular application. This process implies tuning the hyperparameters, so that the model parameters will generate the best predictions. Hyperparameters can be tuned manually, or automatically (fortunately, at the time of writing this thesis, considering the extensive research that is being carried out in this field, there are a number of available frame-

works for hyperparameters tuning, such as Keras Tuner). The Hyperparamters that play a crucial role in the performance of a CNN are:

- Batch size;
- Number of training epochs;
- Activation function;
- Optimisation algorithm;
- Learning rate;
- Loss function.

The process of manually tuning the above hyperparameters will be described further in this chapter.

2.1.4 Siamese Neural Networks

A siamese network, like its name suggests, is a type of neural network that contains two or more identical sub-networks which share the same architecture, weights and hyperparameters. Siamese networks were first introduced in 1993 by Bromley and LeCun as a solution for image matching problems, in particular signature verification [72]. Each subnetwork represents a CNN with identical weights and hyperparameters. This type of network is usually used when finding similarities and relationships between related entities, rather than assigning them to different classes [73].

The approach used by siamese neural networks is to feed two inputs separately to the sub-networks and compute their feature maps using a series of processing layers (convolutional, ReLU, max pool, etc.), depending on the network's architecture. Afterwards, the feature vectors are compared in the distance layer, using absolute difference, cosine similarity or contrastive loss functions. Based on these learnt parameters, the model outputs a binary value which represents the similarity between the two inputs (Figure 2.5).

Fundamentally, a siamese network is looking for similarities and dissimilarities between data, rather then classifying it to certain classes. Thus, the siamese network is trained and tested on random pairs of inputs chosen from the datasets. The pairs can be images belonging to the same class, in which case the pair will be assigned a logical value 1, or images belonging to different classes, in which case the pair will be assigned the value 0.

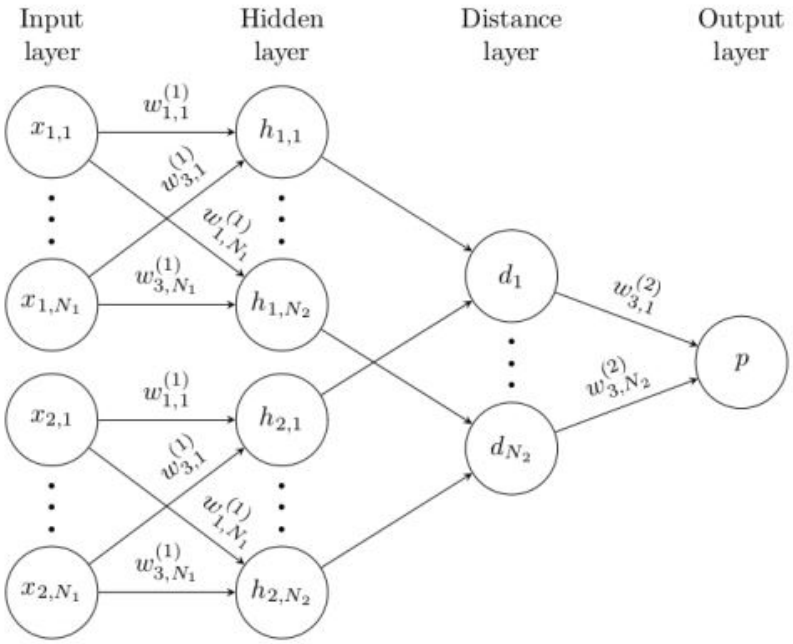


Figure 2.5: Siamese network for binary classification with logistic prediction p. The twin networks share the same weights and parameters. [74]

Siamese networks are used in particular for one-shot learning, which is a type of classification where examples are presented to the network just once, before making a prediction on a test entity. One-shot learning is helpful to facilitate data collection, given that a large amount of labelled examples are no longer required to obtain reasonable performance [75]. This is convenient when the training data set is not sufficiently large to train a DNN for classification. A CNN usually requires at least 500 training examples for each entity to be classified correctly, as demonstrated in the following section (training set dimensions, Figure 2.19).

Therefore, the one-shot learning method can be a suitable approach for this project, considering that the radar image dataset used for training the network is not very large. Therefore, siamese neural networks represent a potential solution to this issue and will be further investigated in this project, both on the MNIST and on the automotive radar dataset.

2.1.5 MNIST Dataset

Modified National Institute of Standards and Technology (MNIST) [76-77] is a large scale computer vision database of handwritten digits and a subset of a larger set from NIST. It is used for ML tasks, mainly for classification and pattern recognition purposes. The dataset contains 60,000 training images and 10,000 testing images.

The MNIST dataset was used in this work prior to the automotive radar dataset, to get accustomed to CNNs in Python using Tensorflow and Keras, data augmentation and different input data formats. Throughout this chapter, different studies will be performed to determine how well the neural network can generalise on unseen data that is different from training data, due to the target's position, scale and noise and also to analyse the influence of different hyperparameters on the performance of the model. These stages are performed because they have parallels in the radar dataset, with radar targets placed at different ranges and several rotation angles. Moreover, it is not known how the classification model will perform on the radar dataset because there is no evidence of neural networks applied for classification on similar radar datasets. Hence, the following analysis on the MNIST dataset will give a good indication of the expected results, as well as a way of debugging potential issues.

Several neural network models were trained on MNIST and tested on a similar dataset of handwritten digits. The dataset was created in accordance with the MNIST design specifications in three stages:

1. Stage 1: Reshape digits to 28x28 pixels images and invert the colours (white digit, black background);

2. Stage 2 : Fit the digit into a 20x20 pixel box and pad it with eight black rows and columns;

3. Stage 3: Center the digit by computing the centre of the mass of the pixels and position this point at the centre of the 28x28 pixels image.

Test-time Data Augmentation

In order to assess how the model performs when different variations of the same digit are tested (different scales and rotation angles), the dataset was modified to include 3 additional scales (70%, 40% and 20%) and seven other rotation angles (30°, 90°, 150°, 180°, 210°, 270° and 330°). This test-time data augmentation can be seen in Figure 2.6.

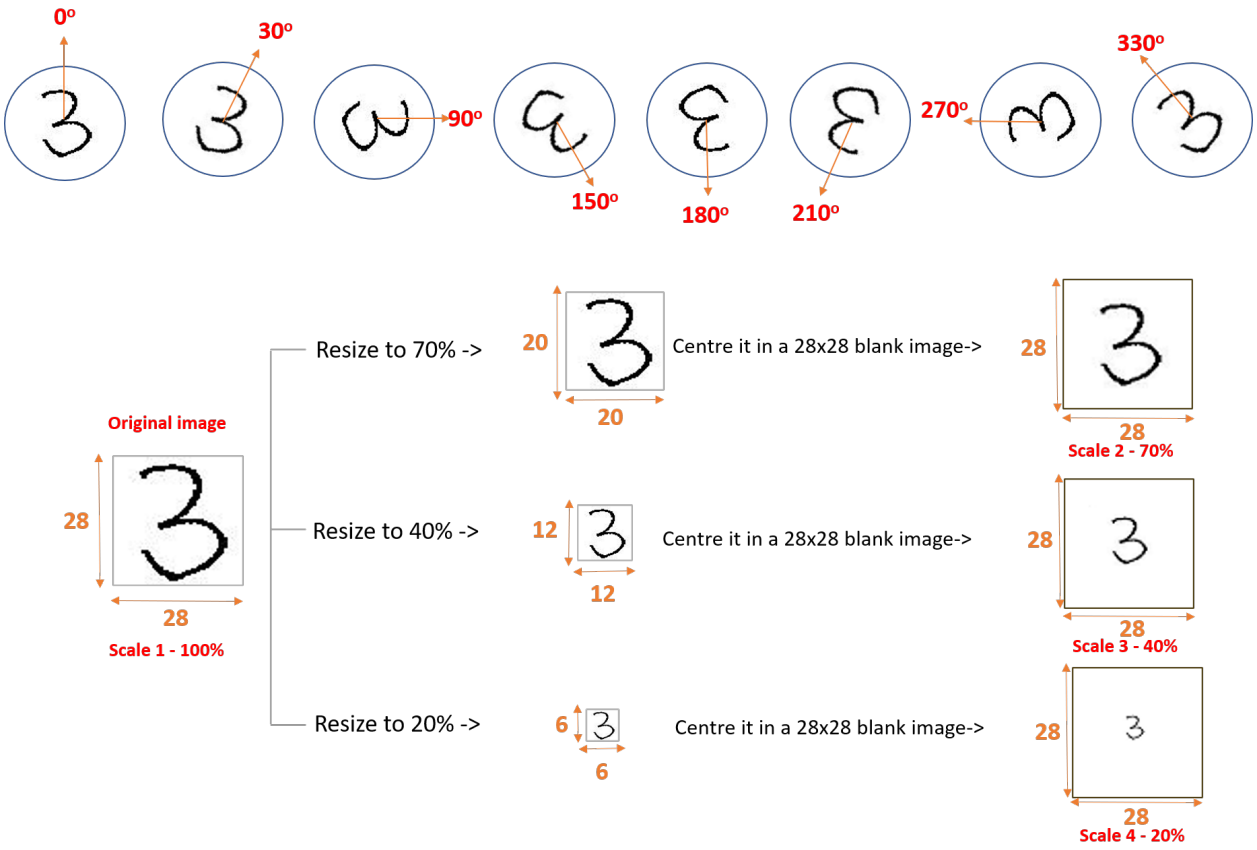


Figure 2.6: Test-time data augmentation for MNIST dataset.

A CNN with two convolutional layers was trained on the original MNIST dataset

and tested on three different datasets corresponding to different scales (70%, 40% and 20%), after each image processing step. The model hyperparameters are 256 batch size, 10 training epochs and Adadelta optimisation algorithm with a default learning rate of 1. The testing results after each processing step and for each scale can be seen in Figure 2.7. It is evident that the accuracy increases after each processing step, as expected, because the features of the testing images begin to match the features of the training images. The accuracy for the 3^{rd} scale is lower, due to the fact that the training set does not include similar scales and the model cannot generalise well enough.

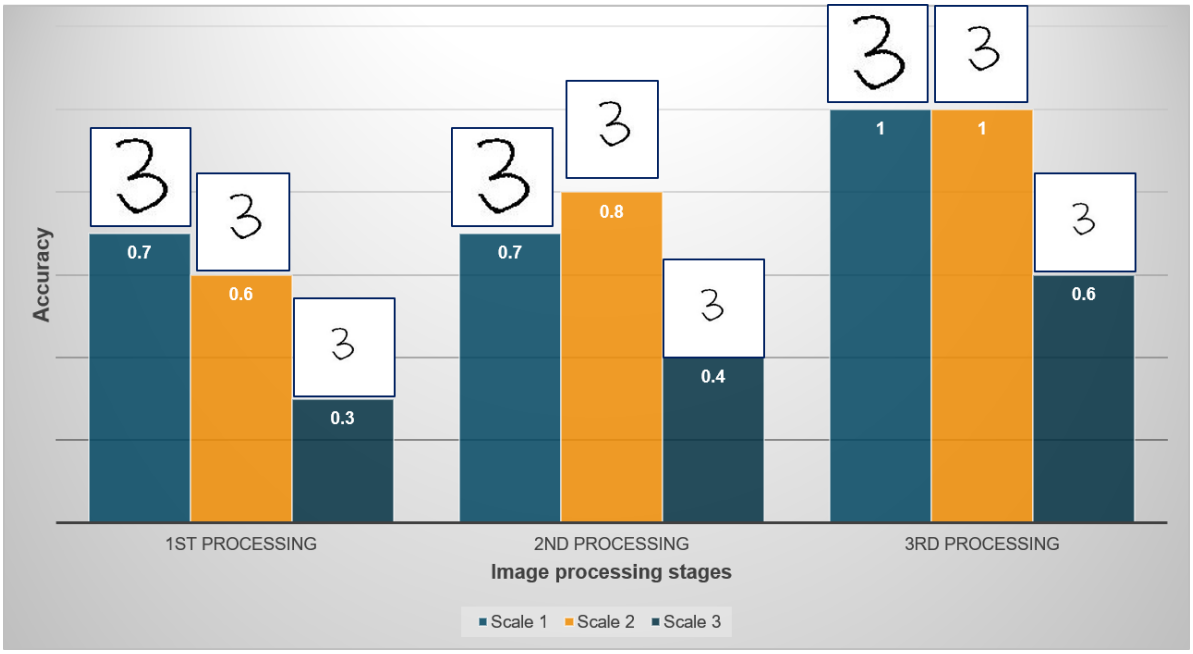


Figure 2.7: Test results for each image processing stage on test-time augmentation with different scales. The model was trained on the original MNIST training dataset.

Oppositely, when the test set is augmented with images of the digits from different rotation angles, the accuracy is very low. Figure 2.8 shows the test results for different scales and rotation angles. As expected, the features of the rotated digits are too distinct from the training examples to allow the model to generalise on the unseen data. The accuracy improves slightly for 180° rotation, because some digits can still be correctly classified (i.e

1, 0 and 5 are very similar to their 180° rotated version).

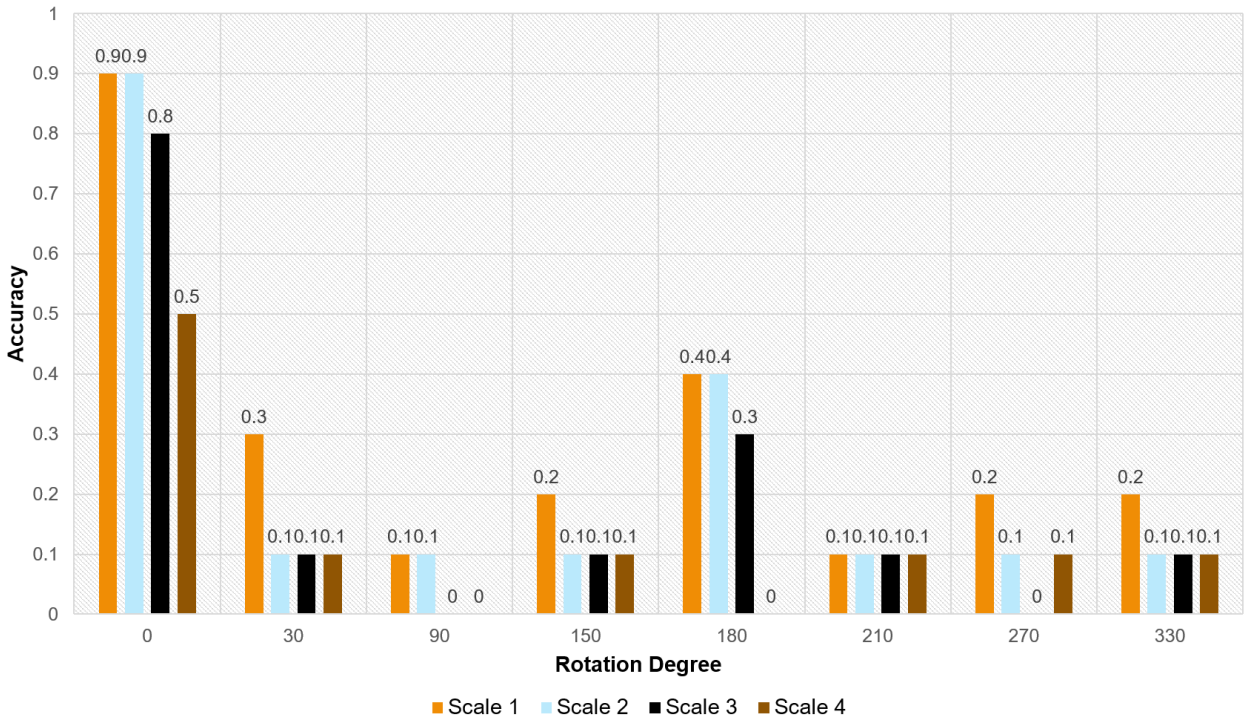


Figure 2.8: Test results on test-time augmentation with different scales and rotation angles. The model was trained on the original MNIST training dataset.

Cohen's Kappa Coefficient

Another measure of performance, apart from accuracy (which can be simply defined as the percentage of the correct classified instances) is the Cohen's Kappa Coefficient [78]. This metric measures how well the classifier performed, as compared to how well it would have performed simply by chance. The Kappa statistic based on the value of the Kappa coefficient can be seen in Table 2.2.

Table 2.2: The Kappa Statistic.

Kappa coefficient value	**Agreement level between classes**
< 0	agreement less than chance
0	agreement equivalent to chance
0.1 - 0.2	slight agreement
0.21 - 0.4	fair agreement
0.41 - 0.6	moderate agreement
0.61 - 0.8	substantial agreement
0.81 - 0.99	near perfect agreement
1	perfect agreement

The Kappa coefficient can be calculated using the equation:

$$Kappa = \frac{Observed\ Accuracy - Expected\ Accuracy}{1 - Expected\ Accuracy}, \tag{2.2}$$

where the Observed Accuracy and the Expected Accuracy are calculated using a confusion matrix. The confusion matrix for Scale 1 with 150° rotation is represented in Figure 2.9.

Digits	0	1	2	3	4	5	6	7	8	9	Total
0	14	0	11	0	0	7	0	0	0	0	32
1	1	6	6	0	8	8	1	0	2	0	32
2	0	0	10	1	6	10	0	2	3	0	32
3	3	0	8	6	1	7	4	0	3	0	32
4	2	0	8	2	2	15	1	2	0	0	32
5	2	0	8	2	0	13	3	1	0	3	32
6	5	0	7	2	0	9	6	2	1	0	32
7	0	0	12	0	7	8	0	4	1	0	32
8	1	0	9	1	4	9	2	0	6	0	32
9	3	0	8	1	0	11	3	0	5	1	32
Ground truth	**31**	**6**	**87**	**15**	**28**	**97**	**20**	**11**	**21**	**4**	320

Figure 2.9: The Confusion Matrix for MNIST test set with scale 1 and 150° rotation.

The observed accuracy is calculated as the accuracy of the instances that were cor-

rectly classified throughout the entire confusion matrix:

$$Observed\ Accuracy = \frac{\sum diag(M)}{\sum_{i=1}^{C} T_i}, \tag{2.3}$$

where M is the confusion matrix, $diag(M)$ is the green diagonal, C is the total number of classes and T_i is the total number of instances for a class i.

The expected accuracy is the accuracy that any random classifier would be expected to achieve only by chance:

$$Expected\ Accuracy = \frac{\sum_{i=1}^{C} G_i T_i}{[\sum_{i=1}^{C} T_i]^2}, \tag{2.4}$$

where G_i is the ground truth for a class i.

Therefore, the Kappa coefficient for the above confusion matrix is:

$$Kappa = \frac{0.21 - 0.1}{1 - 0.1} = 0.12. \tag{2.5}$$

As expected, the Kappa Coefficient will have a low value, 0.12. According to table 2.2, this corresponds to a slight agreement between classes.

Train-time Data Augmentation

The original MNIST training dataset was augmented with different scales and rotating angles in order to determine how this influences the classification results. Figure 2.11 illustrates the classification results after three different scales and seven different rotating angles were added to the training dataset. Hence, the original MNIST dataset consisting of 60,000 training images was augmented to 480,000 images to include all rotating angles (Figure 2.10) and then further augmented to include all four scales. The final training dataset comprised of 1,920,000 images (Figure 2.11).

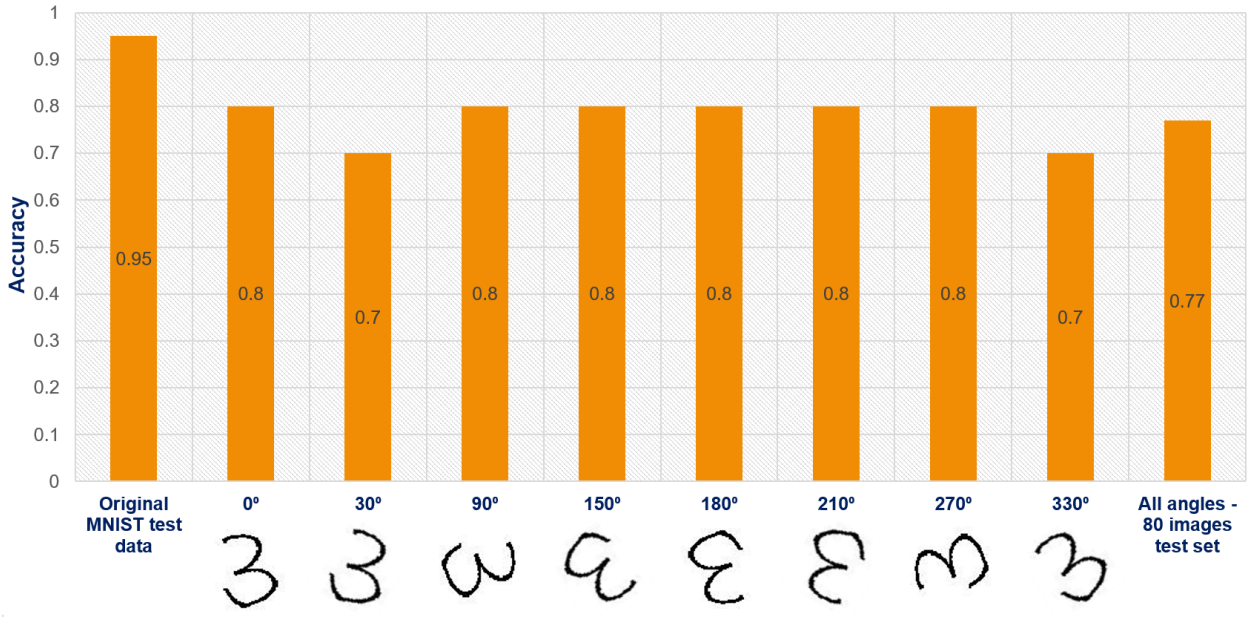


Figure 2.10: Test results on train-time augmentation with different rotation angles. The model was trained on the augmented MNIST training dataset with 8 rotation angles - 480,000 images.

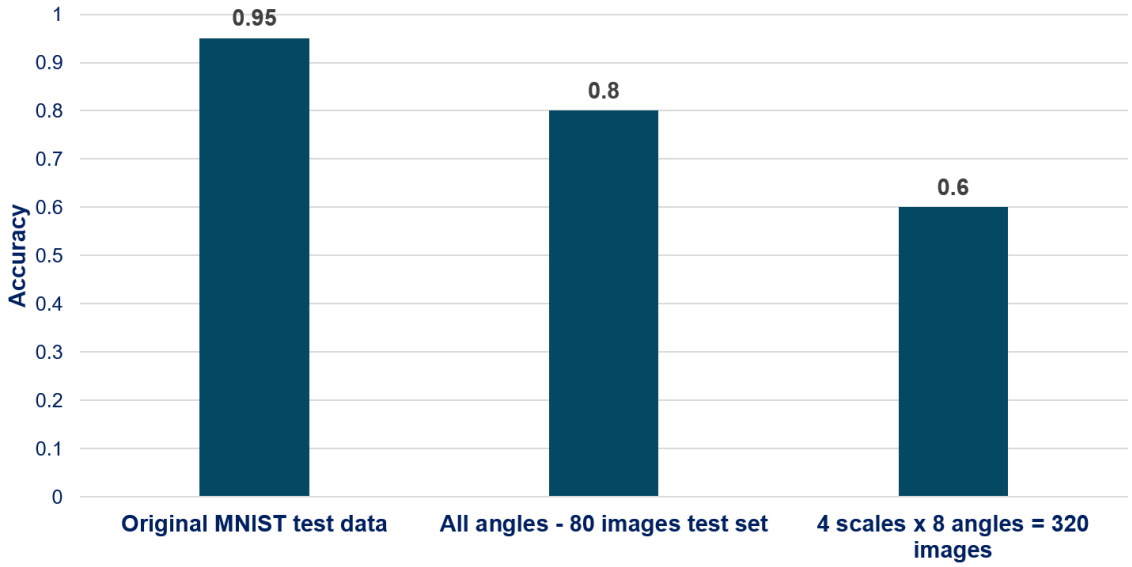


Figure 2.11: Test results on train-time augmentation with different rotation angles and scales. The model was trained on the augmented MNIST training dataset with 8 rotation angles and 4 scales - 1,920,000 images.

Figures 2.10 and 2.11 show that testing accuracy improved considerably after the training set was augmented with images of digits from different rotation angles and scales and the model was able to generalise better. A similar approach was explored for classifying radar images with different views (targets rotated at different angles) and with different scales (distances to the targets). The results and methodology will be presented in this chapter. More references will be made to the MNIST dataset experiment throughout this chapter, for hyperparameters tuning and determining the suitable training set dimensions.

Improving the test accuracy by using filtering techniques

Further digital image processing techniques were experimented for improving the overall classification accuracy on the augmented MNIST dataset. The digits from the testing set were manually written, therefore an inherent noise was added in the digital scanning process. The following image processing techniques were applied on the images to filter the redundant noise:

- **Mean Filter** - a linear filter applied for reducing the noise in images by smoothing them [79-81]. It is similar to a low pass filter and it reduces the variation in intensity between neighbouring pixels. It is used for removing white, additive, Gaussian noise. The working principle of the mean filter is to replace the value of each pixel in the image with the average (mean) value of the neighbouring pixels, including itself. The aim is to eliminate pixels which are unrepresentative of their surroundings, therefore the values of the pixels after filtering will change. The mean filter is usually thought as a convolution filter. As seen in Figure 2.12, the mean filter has a kernel of size 3x3 (orange rectangle) which represents a centre pixel and eight neighbouring pixels. The kernel is shifted over the whole image in order to calculate the mean value which will replace the centre value. Larger kernels can be used for a stronger smoothing effect.

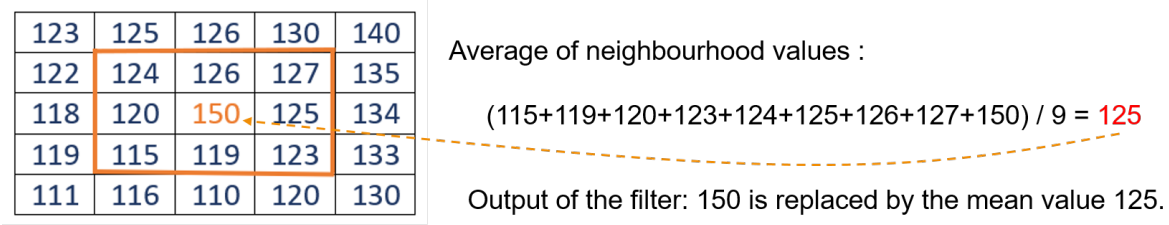


Figure 2.12: Mean Filter - working principle.

The MNIST images were filtered with a mean filter, though the results were not satisfactory, due to the salt and pepper type of noise that was present in the images.

As seen in Figure 2.13, the mean filter has the effect of smoothing the noise, rather than eliminating it. The main disadvantage of the mean filter is that if a pixel has a very different value when compared to its surroundings (for example an edge pixel) it will significantly affect the mean values of the neighbouring pixels. Hence, sharp edges will rather be blurred than preserved. These problems are better tackled using the median filter, which is usually a better filter than the mean filter, but it takes longer to compute.

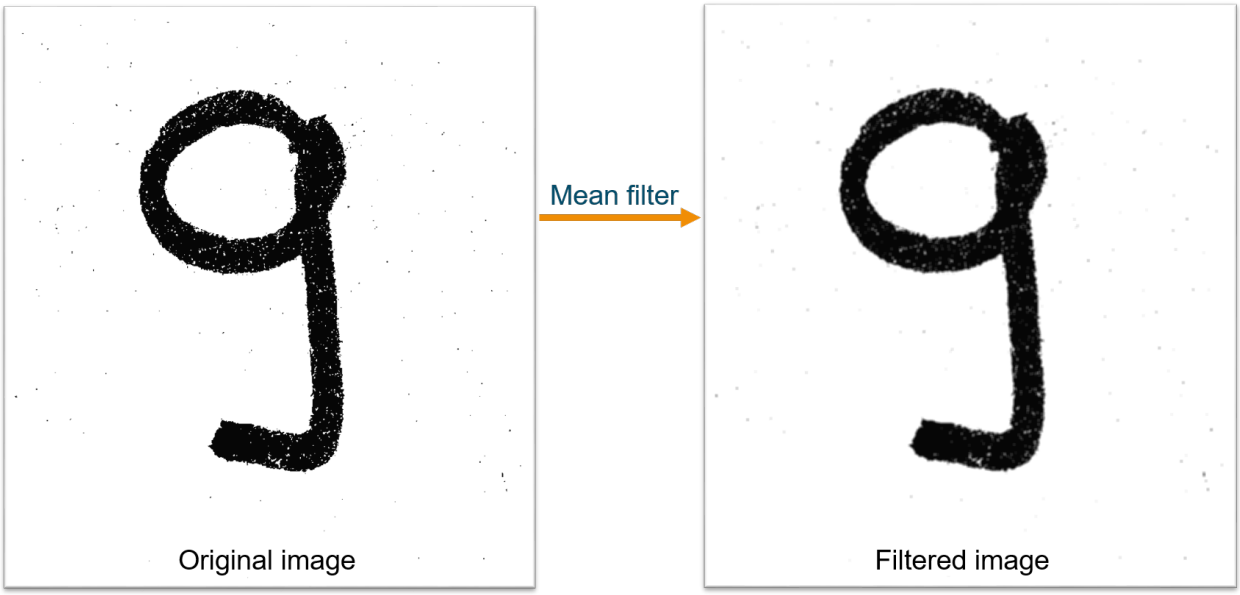


Figure 2.13: Mean Filter applied on a MNIST image.

- **Median Filter** - a nonlinear filter often used to reduce the noise in an image, similar to the mean filter. However, the median filter [81-83] is preferred for preserving useful details in the image. The working principle of the median filter resembles the mean filter. A kernel is slid over the image and the centre value is replaced with the median value of the neighbouring pixels in the kernel (Figure 2.14). The first step is to sort the values of the pixels into numerical order, then determine the median value and ultimately replace the centre value with the median. If the kernel contains an even number of pixels, then the average of the two median values will be calculated.

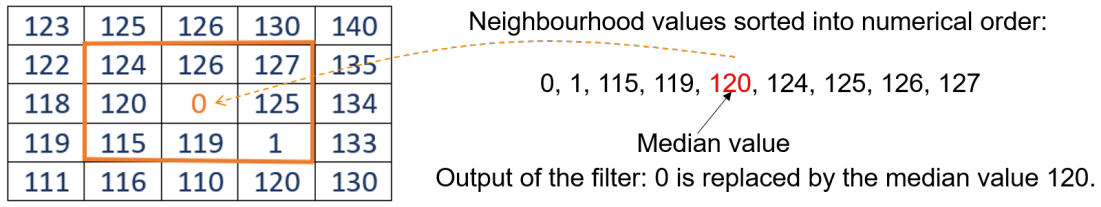


Figure 2.14: Median Filter - working principle.

The drawbacks of the median filter are that it is relatively complex to compute for large images and it is not as good as the mean filter at dealing with large amounts of Gaussian noise. However, the median is a more robust average than the mean, thus a pixel with a very unrepresentative value will not affect the median value considerably. Also, unlike the mean filter, the median filter does not create new, unrealistic pixel values, since the median is actually a pixel value that already exists in the image. Therefore, the median filter will perform better at removing noise that has extreme values (salt and pepper noise) and at preserving edges.

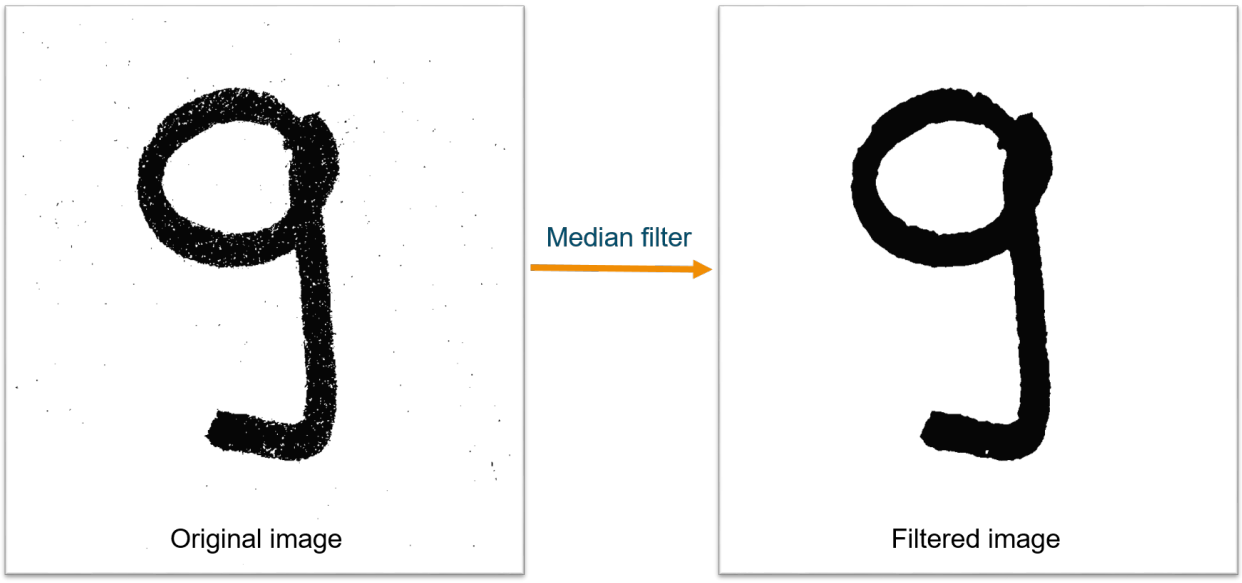


Figure 2.15: Median Filter applied on a MNIST image.

The median filter was applied on the MNIST images using the OpenCV library for

Python. Figure 2.15 shows that the median filter successfully removes the noise in the image, without altering the edges of the digit.

Hence, the MNIST images from the test set were filtered with a median filter. Figure 2.16 shows that the accuracy improved by 33%, from 0.6 to 0.8. The training set contained 30,000 original MNIST images, which were augmented with two other scales and seven rotation angles. The final training set contained 720,000 images.

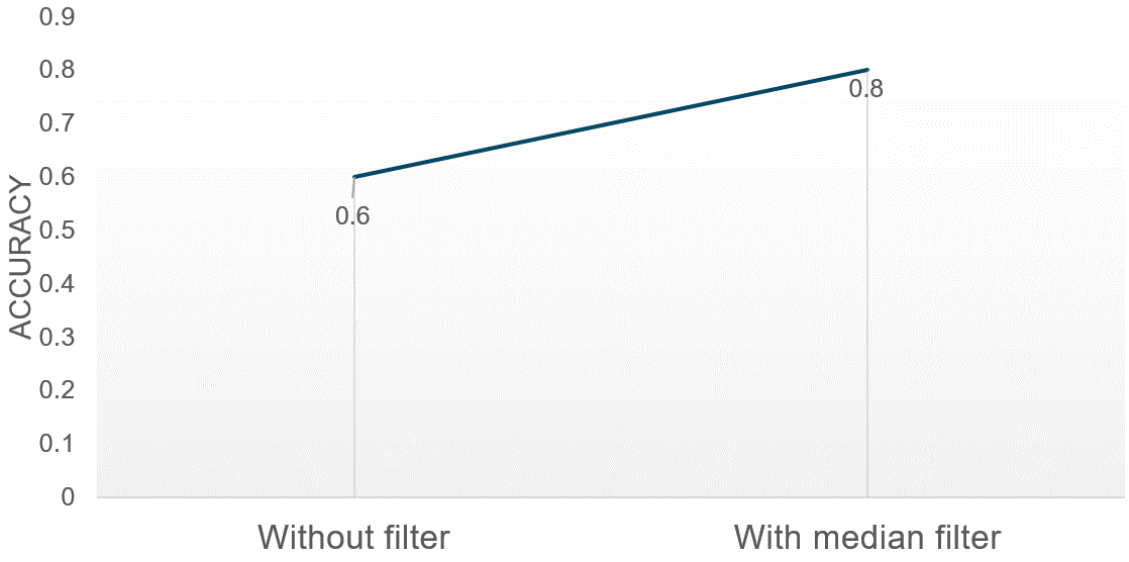


Figure 2.16: Test results on MNIST augmented dataset, before and after median filtering.

The confusion matrix for the final MNIST experiment, after median filtering, is shown in Figure 2.17. This confusion matrix was used to evaluate Cohen's Kappa Coefficient. The results are as follows:

$$\begin{aligned} &Observed\ Accuracy = 0.79. \\ &Expected\ Accuracy = 0.1. \\ &Kappa = 0.76. \end{aligned} \tag{2.6}$$

After the median filtering, the Kappa value corresponded to a substantial agreement between classes, according to Table 2.2.

Digits	0	1	2	3	4	5	6	7	8	9	Total
0	24	0	0	0	0	0	0	0	0	0	24
1	0	18	0	0	0	0	0	6	0	0	24
2	0	0	23	0	0	0	0	1	0	0	24
3	0	0	0	24	0	0	0	0	0	0	24
4	0	0	0	1	17	4	2	0	0	0	24
5	0	0	0	0	0	24	0	0	0	0	24
6	0	0	0	0	0	0	24	0	0	0	24
7	0	4	8	0	8	0	0	4	0	0	24
8	0	0	3	0	0	0	0	0	21	0	24
9	0	0	0	1	0	0	9	0	3	11	24
Ground truth	24	22	34	26	25	28	35	11	24	11	240

Figure 2.17: The Confusion Matrix for MNIST after median filtering.

Siamese Neural Network

A siamese neural network architecture was applied on the MNIST dataset. Given any two input images, the network predicted whether those images depicted the same digit. The model learnt to identify the similarity level between input test pairs according to the probability that they belonged to the same class or not. The pair with the highest score was awarded the highest probability for classification. Three siamese architectures were analysed on the MNIST dataset:

1. Six dense, fully connected layers.

2. Four convolutional layers with ReLU activations.

3. Five convolutional layers with ReLU activations.

The testing accuracy results on a 60,000 images MNIST training set (2,500 original MNIST images, with three different scales and eight rotation angles) are shown below in Figure 2.18, by contrast with the CNN result. The siamese architecture with five convolutional layers achieved the accuracy of 0.96, after median filtering, when testing 240 different pairs of digits for similarities.

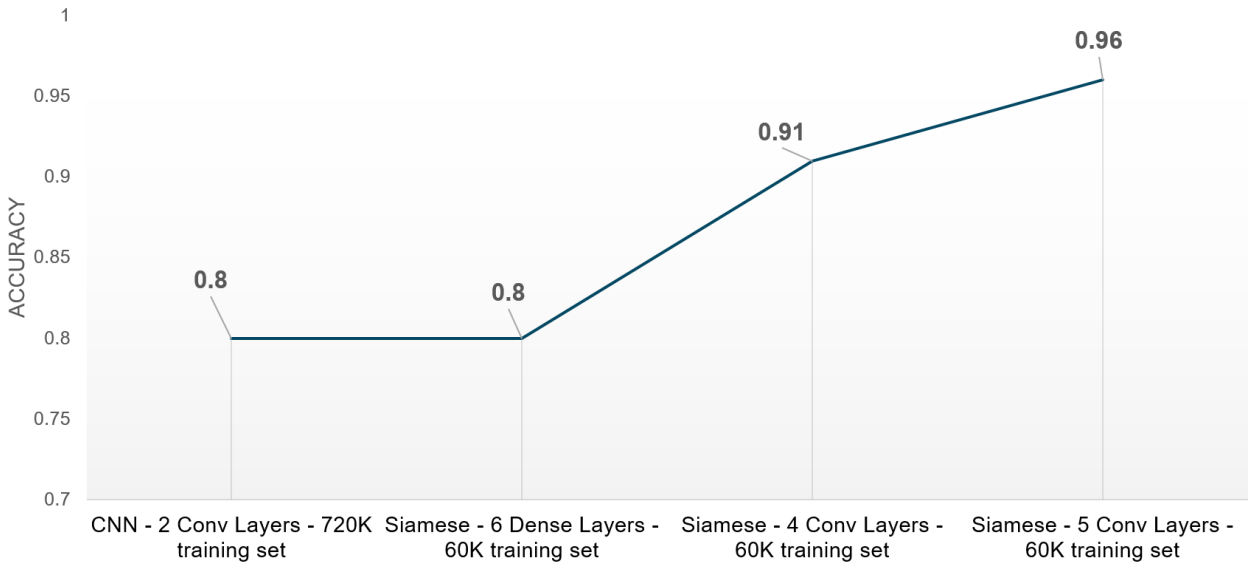


Figure 2.18: CNN vs Siamese results on the MNIST dataset, after median filtering.

Training Set Dimensions

The effect of training set dimension on the MNIST data set can be observed in Figure 2.19, which shows some experimental results with CNNs and MNIST images, which were achieved prior to moving on to analyse the radar dataset. The results show that the threshold for a satisfactory accuracy occurs for a training dataset of 120,000 images. The dataset consisted of 240 different entities: 10 handwritten digits, each one rotated to 8 different angles and re-scaled to 3 different sizes. Thus, it has been shown that 500 images of each entity are required for training a CNN and obtaining a suitable classification accuracy.

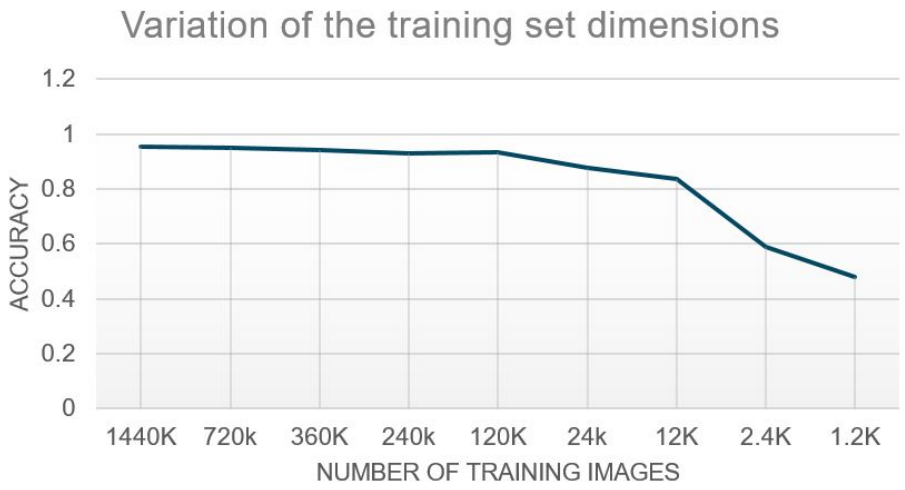


Figure 2.19: Effects of different training set dimensions on the testing accuracy for the MNIST dataset.

2.2 Automotive Radar Imagery Classification

2.2.1 Dataset Requirements

The radar dataset was designed to capture some of the roadside objects from a vehicle's outdoor environment: a bicycle, a cone, a dog, a mannequin, a traffic sign and a trolley. Neural networks were trained and tested using the radar imagery of the six different objects, placed at different distances to the radar and for several rotation angles, which will be described in the next section. These data augmentations were performed as in scanning radar the object occupies different numbers of resolution cells with distance and its reflectivity may change due to fluctuation of RCS over look angle.

Examples of optical images for the six targets and their associated range profiles in low-THz radar imagery are shown in Figure 2.20. All objects are placed at 3.8 m distance from the radar.

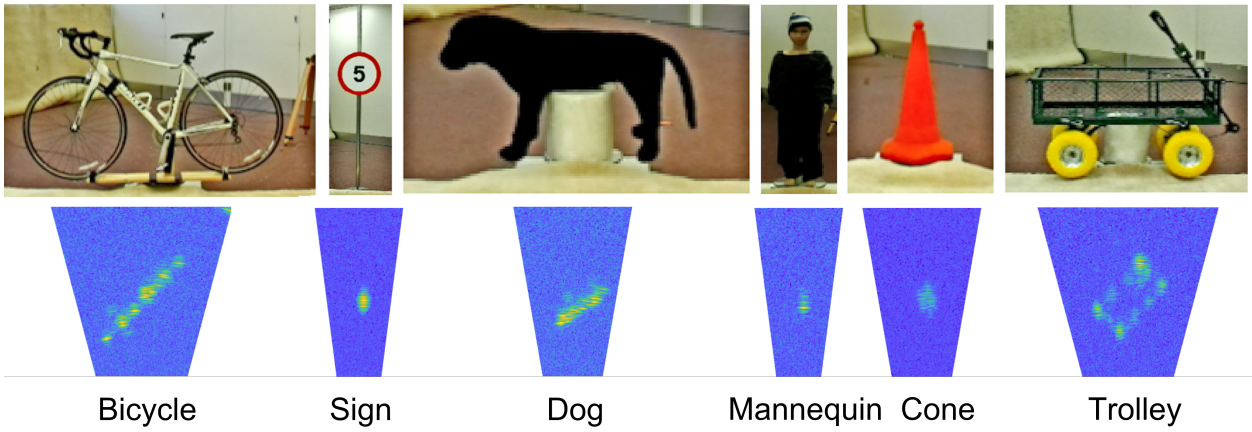


Figure 2.20: Optical and radar images of the six targets placed at 3.8 m distance from the radar.

2.2.2 Experimental Setup and Data Acquisition

The radar system was designed at The University of Birmingham and the measurements for obtaining the imagery dataset were performed in controlled laboratory environment. The system comprised of a ZED stereo camera and a bespoke Linear Frequency Modulated Continuous Wave (LFMCW) radar, with one transmitter and three receivers, to obtain multiaspect data for the elevation profile reconstruction. A picture of the setup configuration is shown in Figure 2.21. For this project, the indicated receivers and transmitter were mounted at different heights, which are also given in Table 2.3.

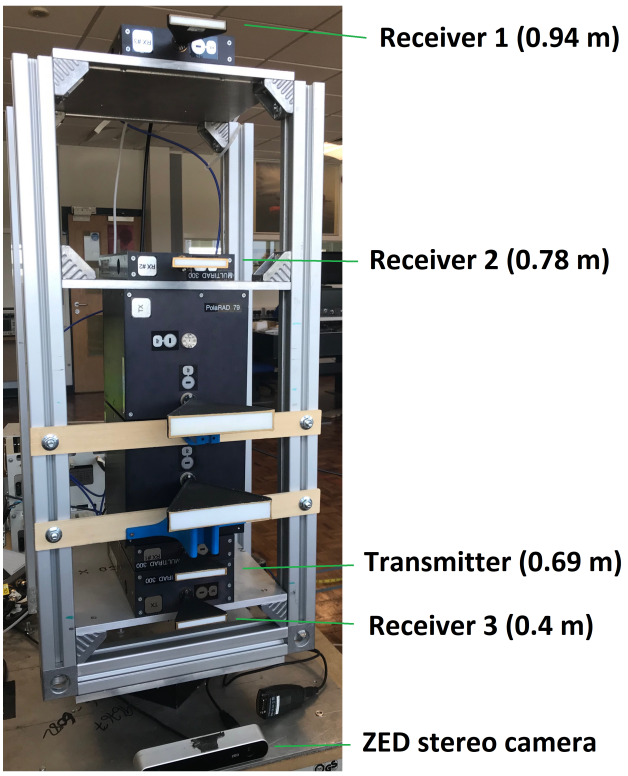


Figure 2.21: Experimental setup.

A full radar scan was taken for each target azimuth position by placing the objects on a turntable and rotating it in a stop-and-go mode. The radar parameters used for the

measurements in this experiment are shown in Table 2.3.

Table 2.3: Parameters used for measurements.

Parameter	**Value**
Sweep Bandwidth	20 GHz
Chirps per Dwell	10
Dwell Time	11 ms
Chirp Pattern	Repeated Up Chirps
Centre Frequency	290 GHz
Transmitted Power	0 dBm
Azimuthal Beamwidth	1.0-1.1° (3 dB) 1.4-1.5° (6 dB)
Elevation Beamwidth	6.2-7.0° (3 dB) 8.8-9.1° (6 dB)
Receiver 1 height	0.94 m
Receiver 2 height	0.78 m
Receiver 3 height	0.4 m
Transmitter height	0.69 m

Each object (except for the dog and the traffic cone) was mounted on the turntable and rotated 360°, taking a full radar scan every 4°, thus 90 different scans. The dog and the traffic cone were rotated 180° and 90° respectively and scanned every 4°. The data was taken from three receivers which were placed according to Figure 2.21, hence three different images were acquired for one single position of each object. Moreover, scans were taken with targets placed at 3.8 m and also at 6.3 m distance from the radar.

Figure 2.22 shows an optical image of the setup for the bicycle target and the corre-

sponding radar range profile. We can observe the extent, shape and outline of the bicycle in the radar image, owing to diffuse scattering, which makes it suitable for classification using computer vision methods.

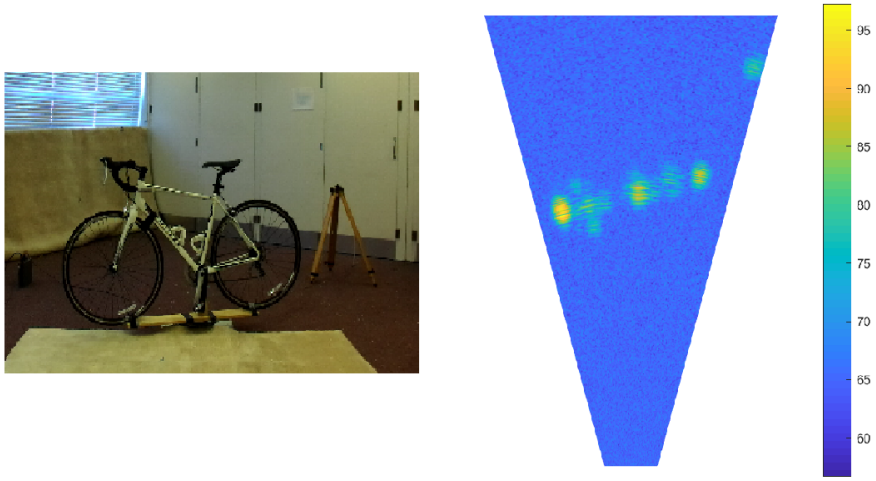


Figure 2.22: Left: the ZED optical image of a bicycle target at 12° rotation. Right: the corresponding radar image.

2.2.3 System Resources

Software

TensorFlow and Keras were used for developing neural networks during this project, due to the numerous advantages that they provide over other machine learning back-ends.

TensorFlow is a public, open-source software library, developed by the Google Brain team for internal use and released to the public in 2015. TensorFlow is mainly used for machine learning applications and other numerical computations. Its name derives from the operations that neural networks perform on multidimensional arrays, also called "tensors" [84].

Some of the main features offered by TensorFlow are:

- Fast computing: it can run on multiple CPU and GPU platforms, reducing the computational time for training DNNs.
- Flexibility: it provides multiple powerful mathematical functions that can solve most problems, not only machine learning related.
- Portability: runs on most operating systems, as well as on mobile computing platforms such as Android.
- Easy debugging: it provides a tool for analysing and debugging developed models, called TensorBoard.
- Wide adaption: ARM, Google, Intel, eBay, Qualcomm, DropBox, Twitter, Airbnb are just some of the numerous tech giants that use TensorFlow to improve their business intelligence.

Keras is a top-level neural network Application Programming Interface (API) that runs on top of other backends, such as TensorFlow or Theano and it is written in Python. Keras provides a more understandable interface for TensorFlow, being more user friendly and having an extended modularity. Standalone modules can be combined in order to develop the desired neural network architecture, with as less restrictions as possible.

The Python code for the neural networks was implemented in this project using Keras, under TensorFlow backend, in the Anaconda Spyder Integrated Development Environment (IDE). The full code for CNN and siamese is provided in the Appendix section of this thesis.

Hardware

Initially, training and testing of neural networks were performed on a single CPU and later on a GPU. However, as the complexity of the neural network was increased by adding more convolutional layers, the computational load of the model became too heavy and training the network required extended amount of time. Therefore, the training and testing procedures of the neural networks have also been performed using the hardware resources from University of Birmingham's BlueBear High Performance Computing (HPC) service (Linux server) [85].

2.2.4 Radar Dataset for Classification

Image Processing

The range profiles were generated by processing the radar data in MATLAB. The targets were cropped from the original images to the same dimension. Due to the fact that all images should have the same dimensions when presented to the neural network, the image dimension was set to include the largest of the six targets, in this case the bicycle. A 220x220 pixel cropping box dimension was determined as being suitable to include all objects.

Examples of processed images that were used for training and testing the neural networks are shown in Figure 2.23.

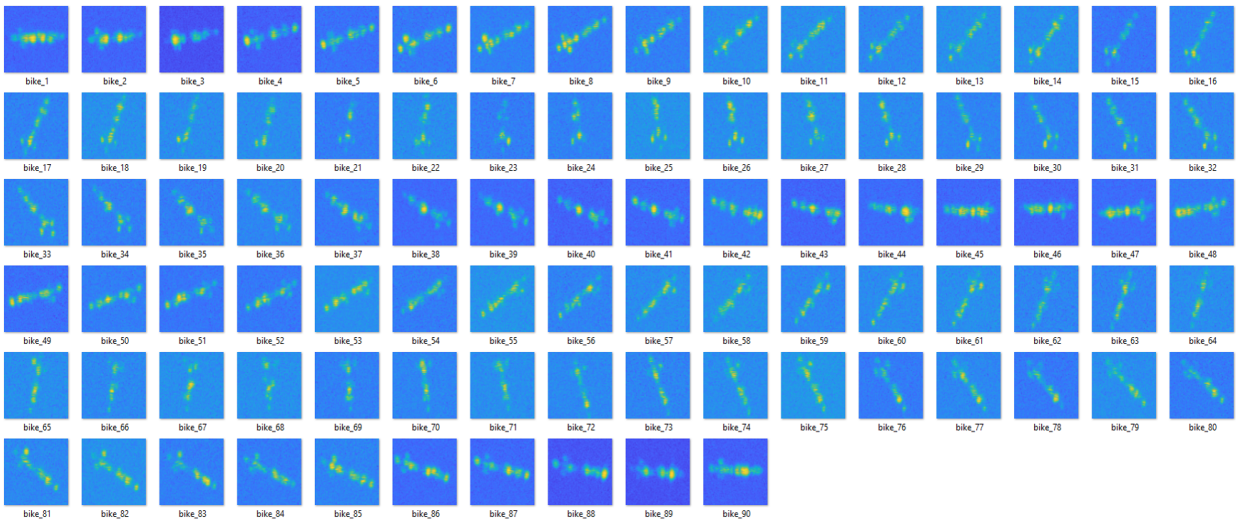


Figure 2.23: Cropped images of a bicycle, 220x220 pixels, 360° rotation with a scan every 4°.

Dataset split

The purpose of dividing the data into train and test sets is to train the model on the training images until a suitable training accuracy is achieved. Afterwards, the network's ability to generalise is verified on the unseen images from the test set, which has to be different from the train set.

Ultimately, the final, full dataset of radar imagery consisted of 2,442 different images of the six roadside targets. The training dataset has to be larger than the testing dataset in order to expose the network to more training examples and to avoid overfitting, therefore the dataset was split between training and testing as follows: 70% for training (1,620 images) and 30% for testing (822 images), chosen randomly.

This dataset is available online in a public repository at the University of Birmingham [86].

2.2.5 Hyperparameters Tuning

As previously stated, one of the shortcomings of neural networks is that suitable values for the hyperparameters need to be selected prior to the network training. This involves multiple tests, performed with different parameter values, until the network's performance proves to be satisfying for a given application. This process is based on a trial and error method, therefore it is time and resource consuming, as the neural network is trained numerous times with different parameters. However, this process is essential and once the suitable hyperparameters are found, the network will achieve reliability and will require further training only in case the dataset changes.

Batch size

The batch size represents the number of samples from the test set that are passed through to the neural network at one time. Samples can also be passed to the network one by one (batch size 1), but typically the larger the batch size is, the faster the epochs will be completed, therefore the neural network will be trained quicker.

However, even though CPUs and GPUs can handle very large batches, the model's features may degrade with large batches. Moreover, this parameter also has to be tested in terms of resource utilisation, as less performant machines do not have enough computational power to process all samples in parallel.

Tests regarding the batch size have been performed on the MNIST data prior to the radar data, as seen in Figure 2.24. Results show that there is a threshold at batch size 256, after which the model starts to deteriorate and the testing accuracy drops significantly. The reason for this decline is that large batch methods tend to converge to a sharp minimum, leading to overfitting and poorer generalisation. By contrast, small batch methods

consistently converge to a flat minimum [87].

Therefore, the optimal batch size was considered to be 256, meaning that 256 images will be passed as a group at once to the network.

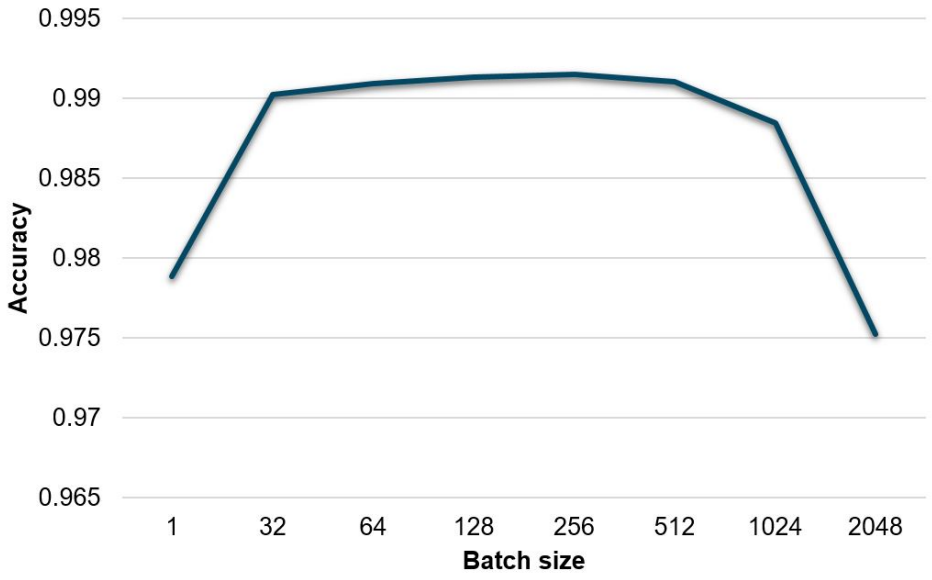


Figure 2.24: Determining the optimal batch size.

Epochs

All training data is passed to the network once during an epoch. An epoch is divided into several batches due to the high computational load. In order for the model to be able to generalise well on unseen data, it is not sufficient to pass the dataset to the network just once, hence the network should iterate through the training data multiple times. This can be achieved by setting the number of epochs parameter to a suitable value.

Unfortunately, this hyperparameter depends on the dataset and its diversity. The algorithm cannot update the weights to the right value if it has not iterated through the data enough times (underfit), yet too many iterations can cause the network to overfit, as

seen in Figure 2.25. Hence, the amount of neurons used in the network should be balanced, because large amounts of neurons overfit the data, while too less neurons can underfit it.

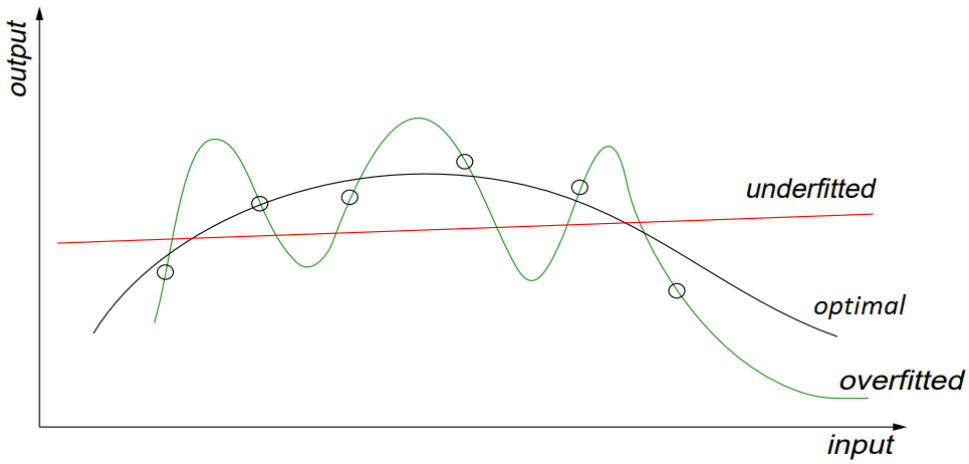


Figure 2.25: Underfitting and overfitting a neural network. The black curve represents an optimal neural network which provides a good compromise between approximation on trained data and generalisation on unseen data [88].

Therefore, the optimum balance must be found through trial and error for each type of network and dataset, by changing the number of training epochs and training the model until the best accuracy is achieved.

The testing accuracy was monitored for different number of training epochs, varying from 1 to 200, for a CNN with 5 convolutional layers. The dataset used was the full radar dataset of 2,442 images, 110x110 Red-Green-Blue (RGB) pixels. The best accuracy (98.78%) was achieved after epoch 139 (Figure 2.26).

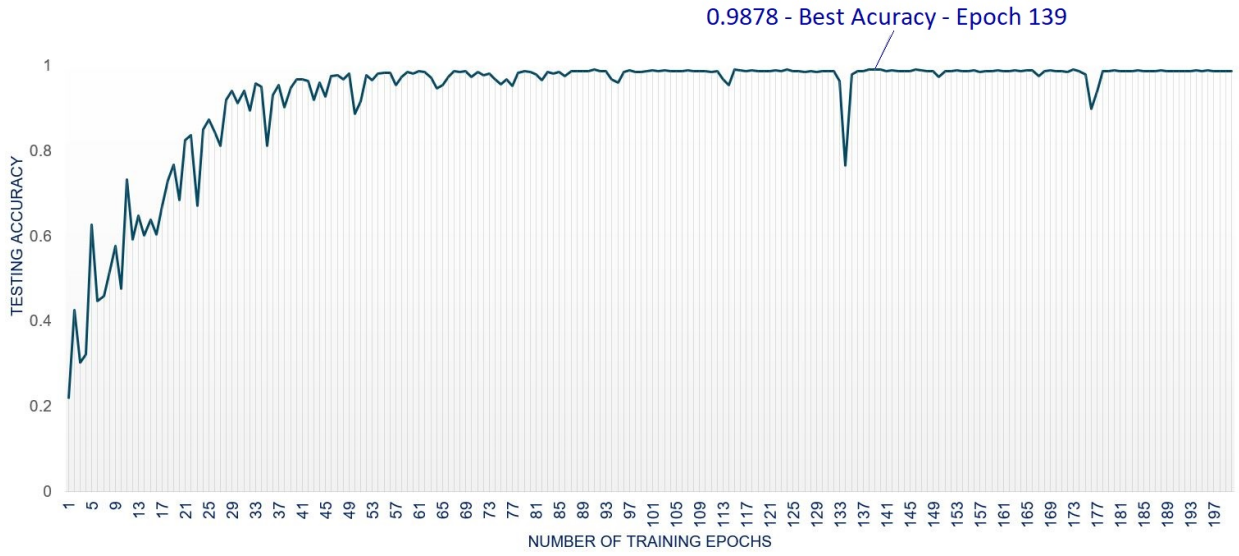


Figure 2.26: Testing accuracy depending on the number of training epochs for a CNN with 5 convolutional layers. The best accuracy is achieved after epoch 139.

Activation Functions

An activation function decides whether or not the neurons should activate (or $fire$) and trigger the others. Those are usually threshold based functions, like step functions, or linear, sigmoid, tanh, ReLU, Softmax, even custom functions.

- ReLU is usually used as an activation function for the hidden layers of a DNN. It is considered to be an easy, general approximator for neural networks, hence it was used as an activation function in this project.

  ReLU is a non linear operation, which has to be used after every convolutional layer. It replaces all negative pixel values in the feature maps (output from convolutional layer) by 0. One of the advantages of ReLU is that it is very fast to calculate and, unlike sigmoid or tanh functions, it does not require any exponential computation. Moreover, it was confirmed that the Stochastic Gradient Descent (SGD) optimisation algorithm

converges considerably faster with the ReLU activation, compared to sigmoid and tanh activation functions [90].

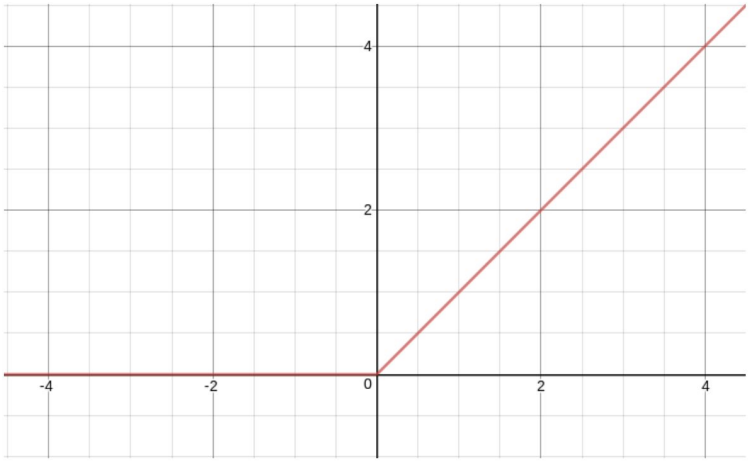


Figure 2.27: ReLU activation function [89].

- Softmax is an activation function which is often used in the FC layer for classification purpose. The fundamental objective of the Softmax function is to convert the output values into probabilities for different classes, that sum to 1.

  The softmax function has the following form

$$S(y_i) = \frac{e^{y_i}}{\sum_i e^{y_i}}, \tag{2.7}$$

  where y_i is the logit input. Softmax is an exponential function, therefore the output is not uniform. If the input logit has a large value, the difference will also be enlarged and the output probability will be closer to 1. Similarly, if the logit is small, the probability of assigning the input to the corresponding class will be closer to 0. The softmax activation function was used in this project in the last FC layer of neurons for determining the probabilities of the input radar image of a target to be assigned to one of the six classes of roadside objects.

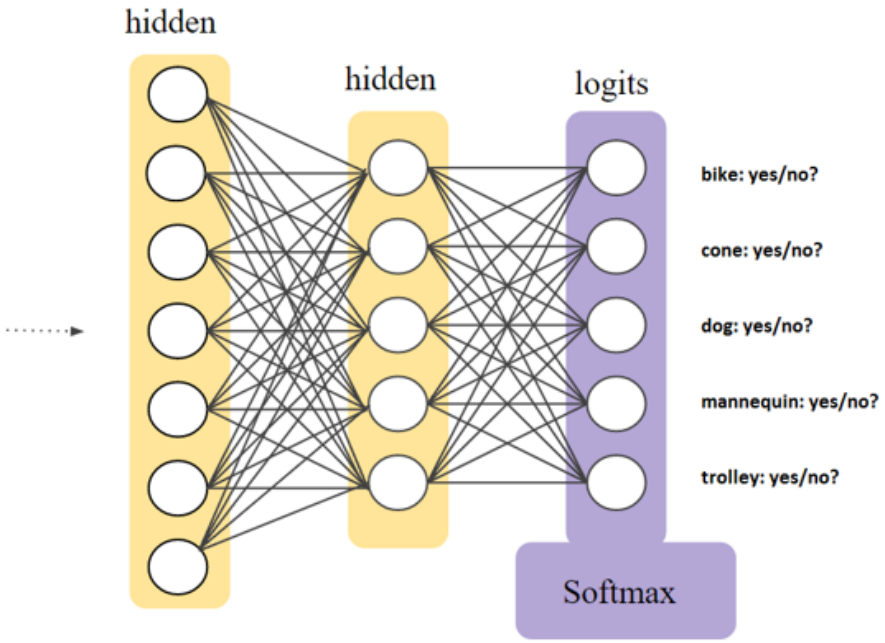


Figure 2.28: Softmax activation function in the last layer of neurons.

Optimisation Algorithm

An optimisation algorithm is a mathematical function which depends on the internal parameters, such as the weights and the bias, which are also called the learnable parameters, due to their ability to update their values with each iteration and get as close as possible to the optimal solution.

There are two types of optimisation algorithms:

- First order optimisation algorithms - they usually minimise an objective function, called loss function, cost function or error function, or maximise it (depending on the type of application) by using its gradient values, thus first order derivatives, which give information about whether the function is increasing or decreasing for certain parameter values [91]. The most common type is the SGD algorithm.
- Second order optimisation algorithms - these methods use the second order derivative

of a function, also called the Hessian matrix, to minimise or maximise the loss function. These algorithms are usually costly and very slow to compute.

A frequent method for stochastic optimisation used in neural networks is the Adam algorithm [92], because it is very straightforward and easy to implement. However, in this project, the optimisation algorithm used was Adadelta, a first order SGD algorithm, which appears robust to different selection of hyperparameters, model architectures and noisy gradient information [93]. Figure 2.29 shows a comparison of different first order optimisation algorithms (SGD, Momentum, Adagrad and Adadelta) on the MNIST handwritten digits classification for 50 epochs, where Adadelta proves to have the best convergence rate.

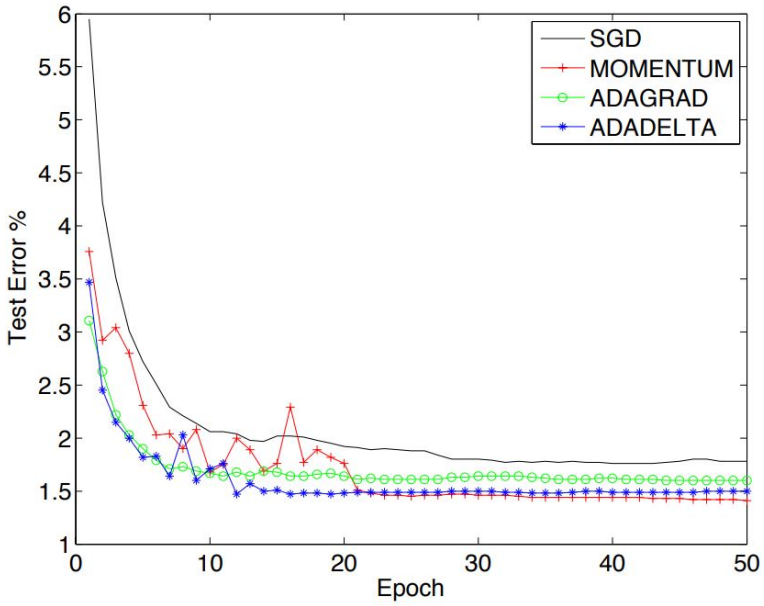


Figure 2.29: Comparison of different optimisation algorithms on the MNIST digit classification for 50 epochs [93].

Learning Rate (LR)

LR is a hyperparameter which controls the weight adjustment with respect to the optimisation algorithm. If the LR is too small, the algorithm will converge very slowly, yet if the LR is too large, the optimisation algorithm may fail to converge, or even diverge from the optimal solution.

The LR must be carefully chosen, therefore in this project a LR of 1 (which is also the default LR value for the Adadelta algorithm in Keras) was used for the Adadelta optimisation algorithm, as it proved to have better results than a lower LR on the MNIST dataset (Figure 2.30). By using an appropriate LR, the optimisation algorithm converges faster and less training epochs are required for achieving a suitable classification accuracy.

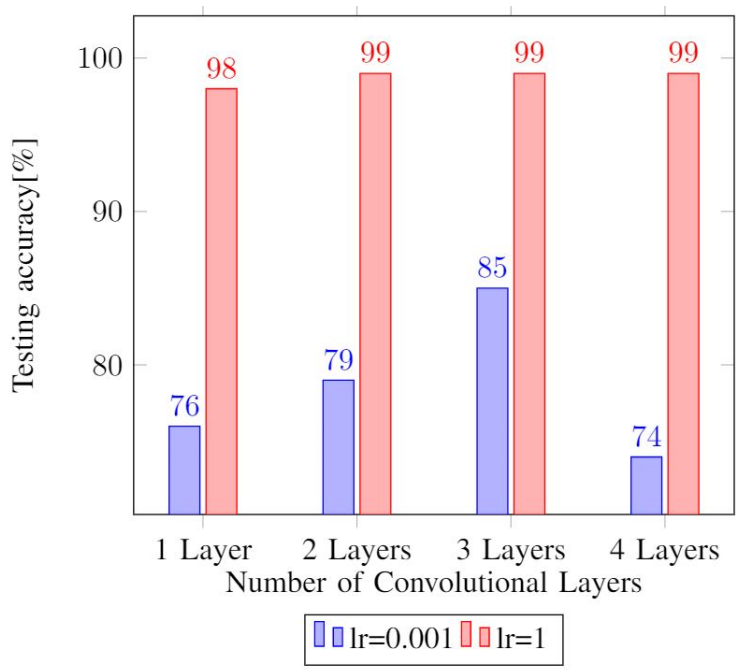


Figure 2.30: Adadelta accuracy on the MNIST dataset for different LRs and number of convolutional layers.

Loss Function

The loss function finds the optimal gradients for the neural network by minimising the error, i.e. the difference between the actual output of the model and the predicted one. Changing the loss function would also modify the error and influence the performance of the model, therefore the choice of the loss function depends on the type of application addressed by the neural network.

Most common loss functions are: Squared Error (SE), Mean Squared Error (MSE), Categorical Cross-Entropy (CE), Binary Cross Entropy, Hinge Loss, Cosine Similarity, Absolute Error, etc. In this project, the loss function used was the Categorical CE loss, which is suitable for multi-class classification. It is also called a Softmax loss because it consists of a Softmax activation, followed by a CE loss. The loss function is minimised by the optimisation algorithm, Adadelta.

CE is defined as

$$CE = -\sum_{i=1}^{C} y_i' \log(y_i), \tag{2.8}$$

where C is the total number of classes, y_i is the predicted probability for class i and y_i' is the true probability.

The loss function is one of the most important hyperparameters of a model, as it provides important information about the efficiency of the model and it is inversely proportional with the accuracy. By analysing the loss function (Figure 2.31) we can determine whether the model is overfit or underfit and when to stop training. Ideally, the loss function of an adequate model should decrease with each training epoch and reach a value as close to 0 as possible.

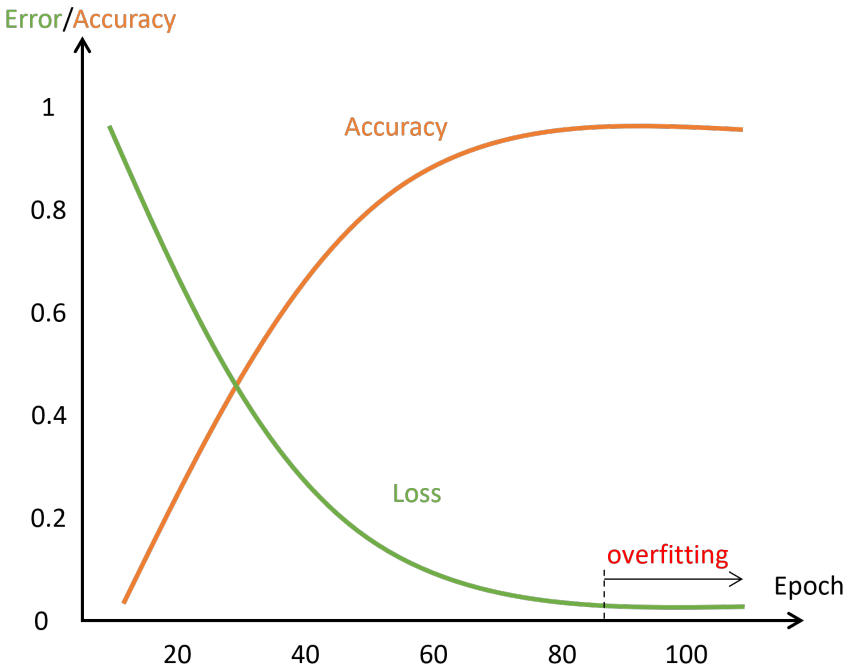


Figure 2.31: A typical graph example of loss function vs accuracy.

2.2.6 CNN Results

The full radar dataset, as well as several subsets were used for training and testing different types of CNNs with the hyperparameters described in the previous section. The process of optimising the model architecture and computational load for obtaining the best accuracy is presented in this section, alongside the interpretation of the results.

The basic flow architecture of the CNN with the typical components (ReLU activations, pooling layers) is represented below in Figure 2.32.

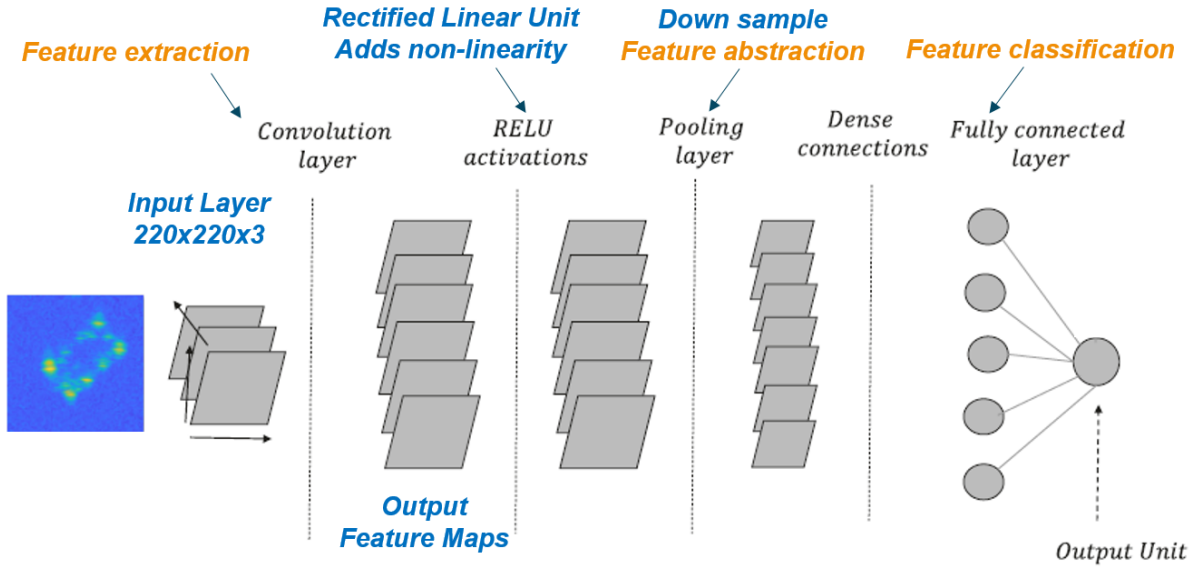


Figure 2.32: Basic flow diagram of the CNN.

Optimising computational load

A subset of the final dataset, which included only the images with the targets placed at 3.8 m from the radar (858 images for training the model and 435 images for testing it) was used in order to determine whether the grayscale images provide enough information for the network to learn the features of this specific radar imagery dataset, or the model can

deduct even more information from the RGB images.

The RGB images have three channels of information and are represented by the colormaps generated in MATLAB, which correspond to the radar signal amplitude in dB. The same images were loaded in Python in grayscale mode, using the OpenCV (Open Source Computer Vision Library - a computer vision and machine learning software library) and have a single channel, with pixel values normalised between 0 and 1. Therefore, when using RGB images, the convolutional filter will also have three dimensions instead of one, proportionally increasing the computational load (Figure 2.33).

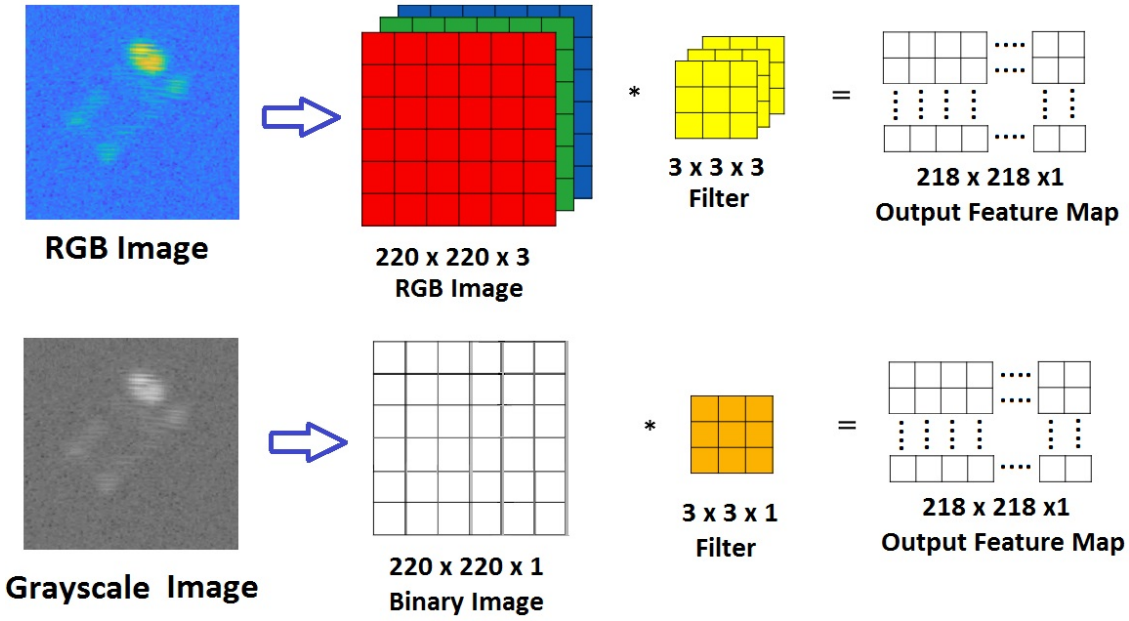


Figure 2.33: Convolutional operation for RGB radar image vs grayscale radar image of the trolley. The RGB image requires 3 convolutional filters, one for each channel, whereas the grayscale image requires only one convolutional filter.

Moreover, all images were cropped to the same dimension before loading them into the CNN. This implied determining suitable dimensions for the largest target from all six objects (in this case, the bicycle), which was found to be 220x220 pixels. In order to optimise the model, the CNNs were also trained with data resized to 110x110 pixels, thus approximately

reducing the computational load by 75%. A comparison of testing accuracies for various CNNs with the aforementioned computation savings is shown in table 2.4.

Table 2.4: Testing accuracy for four different CNNs (2, 3, 4 and 5 convolutional layers), with RGB and Grayscale images, and two different image sizes (110x110 and 220x220 pixels).

No. of Conv Layers	110x110 pixels		220x220 pixels	
	RGB Acc. [%]	Grayscale Acc. [%]	RGB Acc. [%]	Grayscale Acc. [%]
2 Conv	97.01	94.94	95.86	95.4
3 Conv	97.7	97.24	96.55	94.94
4 Conv	98.39	98.16	98.16	97.7
5 Conv	98.62	97.01	97.56	97.37

It can be concluded that the accuracy increases with the number of convolutional layers added to the network and the 3D convolutional filter for RGB images performed better at extracting useful features for classification, as opposed to the one dimensional convolutional filter used for the grayscale images. Moreover, it was found that the datasets with 50% reduced dimensions (110x110 pixels) led to more accurate results.

The trade-off between dimension and colour information appears to be a good choice, as the model learns more information from the depth of an image (three channels) than from an image with only one channel, but larger dimensions, where the redundant pixels do not contribute to improve the feature maps.

Testing Accuracy for Different CNNs

The final dataset of 2,442 radar images (1,620 for training the model and 822 for testing its performance) was used with eight different CNNs, with an increasing number of convolutional layers. The goal was to determine how deep the CNN should be in order to

achieve the best results for this particular type of application.

The architecture of the CNN with 2 convolutional layers is as follows:

$$[CONV \rightarrow MaxPool \rightarrow CONV \rightarrow MaxPool \rightarrow Dropout\ 0.2 \rightarrow Flatten \rightarrow Dense],$$

where every CONV layer is followed by a ReLU activation function, the MaxPool layer reduces the spatial dimensions of the network by keeping only the maximum pixel values in the image, and the flatten layer flattens the input before the dense layer, which is an ANN classifier with six outputs, one for each class.

The actual architecture of the CNN with four convolutional layers is shown in Figure 2.34.

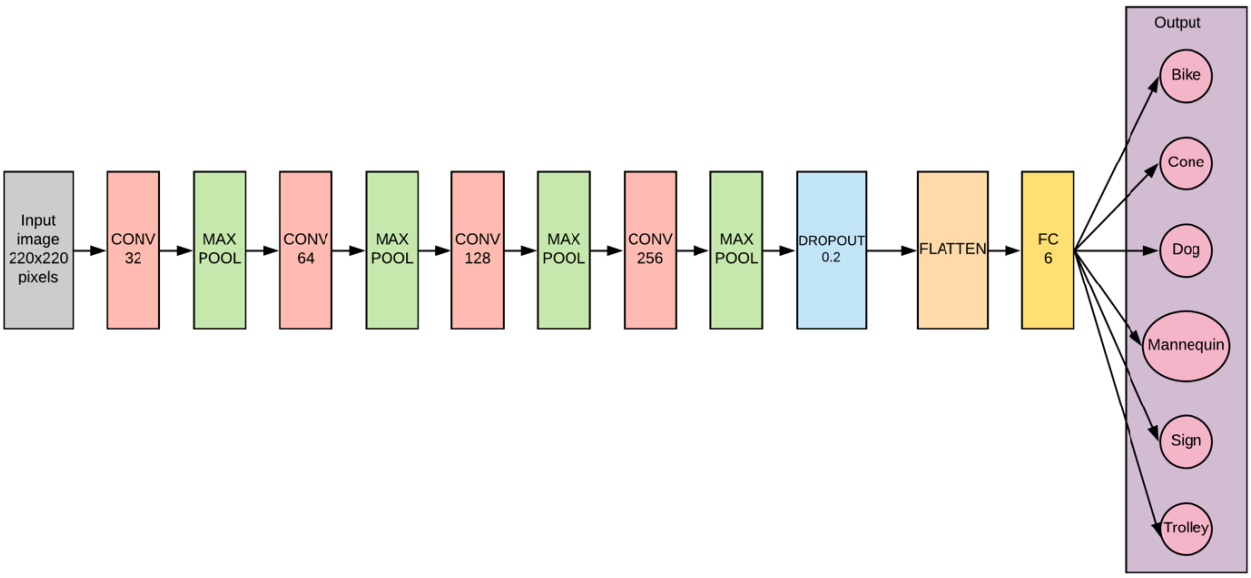


Figure 2.34: CNN architecture with 4 convolutional layers.

The dropout layer prevents the overfitting of the model by deactivating a fraction rate of the input neurons during training. The dropout level has been set to 0.2 (20% of neurons were inactive) after it was found to carry out the best results for a CNN with five convolutional layers (Figure 2.35).

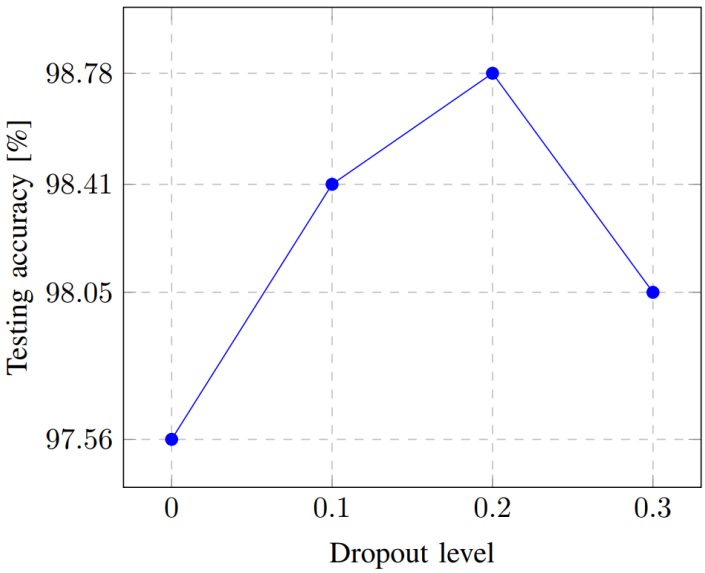


Figure 2.35: The testing accuracy for different values of the dropout parameter for a CNN with 5 convolutional layers, using the RGB, 110x110 pixels image dataset.

The architectures of the following four CNNs (up to 6 convolutional layers) were built by adding the CONV layers one by one, each one followed by a MaxPool layer. For the CNNs with seven and eight CONV layers respectively, MaxPool was added after every two CONV layers, thus for the deepest CNN with eight CONV layers the architecture had the following structure:

$$[CONV \rightarrow CONV \rightarrow MaxPool \rightarrow CONV \rightarrow CONV \rightarrow MaxPool \rightarrow CONV \rightarrow CONV \rightarrow \\ \rightarrow MaxPool \rightarrow CONV \rightarrow CONV \rightarrow MaxPool \rightarrow Dropout\ 0.2 \rightarrow Flatten \rightarrow Dense].$$

Each one these CNNs were trained and tested with the same datasets, and the results can be seen in Figure 2.36. The testing accuracy increases with the number of convolutional layers until the maximum testing accuracy is achieved with 5 convolutional layers (98.78%), after which the accuracy starts to decrease. Moreover, all training and testing images are distinct, thus each image is presented only once to the network. Hence, in this case, the CNN performs a one-shot classification.

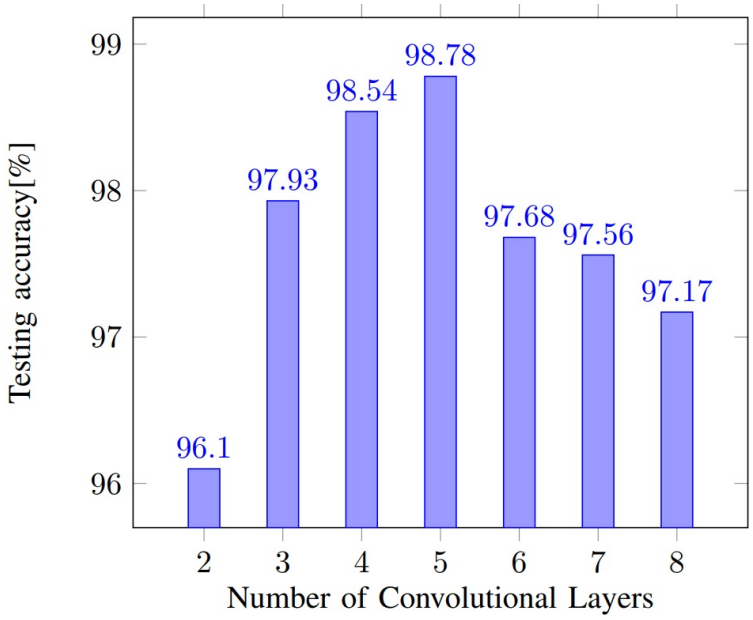


Figure 2.36: CNN testing accuracy for different numbers of convolutional layers.

With each convolutional layer added to the network, the model extracts more features, however, this occurs only to a certain extent (in this case, the 5^{th} layer), after which the model starts overfitting the data. The accuracy declines because the model is learning too many features from the training images and thus becomes unable to generalise on new data. Therefore, it can be concluded that the best architecture for this type of application, given the current dataset and parameters, should be no deeper than five convolutional layers.

The confusion matrix for the model with the best accuracy (5 convolutional layers) is represented in Figure 2.37. The confusion matrix is also referred to as an error matrix and it is used for visualising the performances of a supervised model (labels are known prior to classification), such as observing how the network is confusing between classes [94].

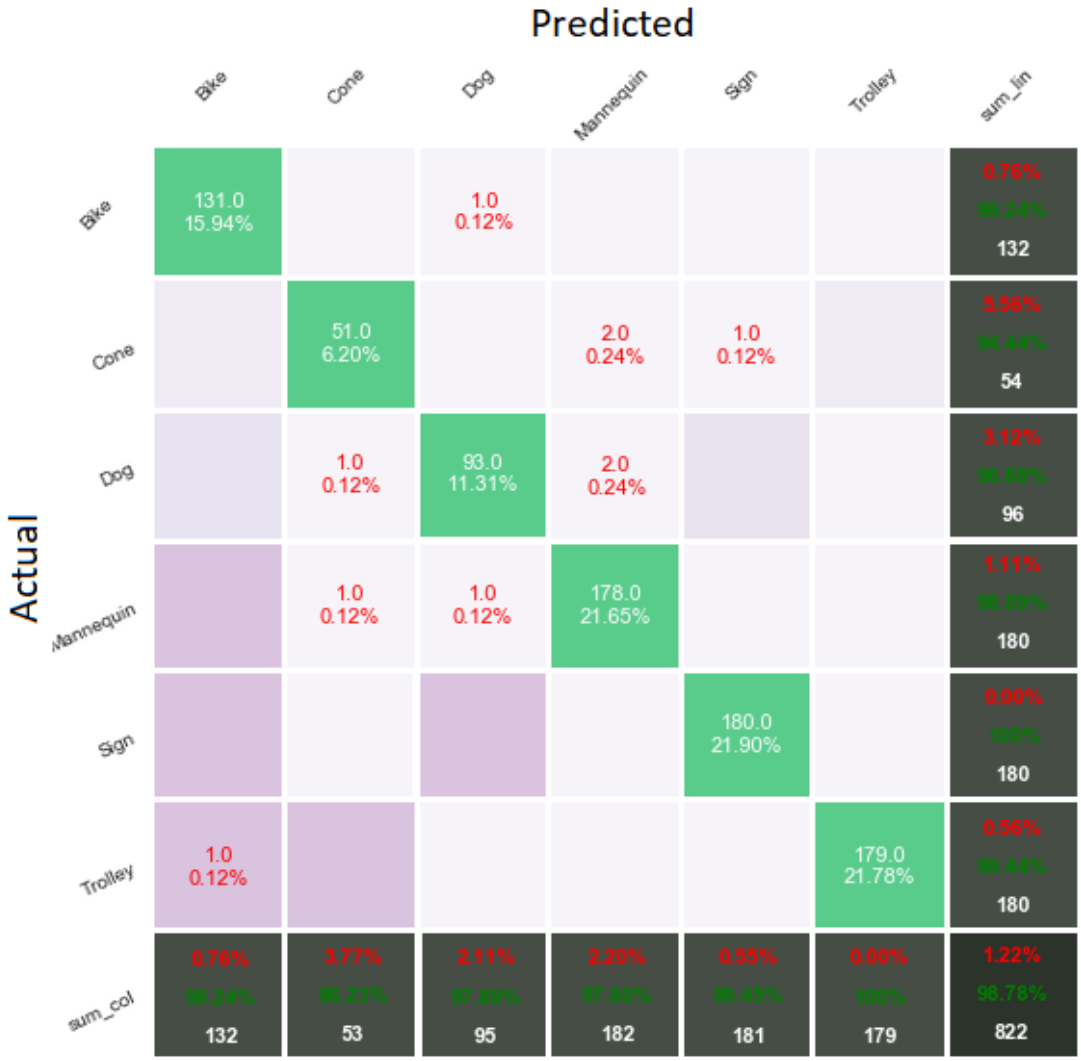


Figure 2.37: CNN confusion matrix.

The confusion matrix can be interpreted as follows: 131 out of 132 bike images were classified as bikes and only one instance was mistaken with a dog; 51 out of 54 cone images were classified correctly, while 2 images were classified as mannequins and 1 image as traffic sign, etc. Remarkably, all traffic sign instances were classified accurately.

Furthermore, Cohen's Kappa coefficient was calculated based on the confusion matrix, to evaluate the agreement between classes, when compared to the chance agreement. Kappa's

values vary from -1 to 1, where 1 implies a perfect agreement between classes and 0 means an agreement equivalent to chance (Table 2.2). In some cases Kappa can be negative, which indicates an agreement poorer than chance.

The observed accuracy is the number of instances that were classified correctly, hence the testing accuracy of the model, 98.78%. The expected accuracy is the accuracy that every random classification model is expected to achieve and it is calculated using the ground truth values, sum_col and sum_lin:

$$\textit{Expected Accuracy} = [(132\times132+53\times54+95\times96+180\times(182+181+179))]/822]/822 = 0.5945 \tag{2.9}$$

Therefore, the Kappa coefficient for this model, calculated using Equation (2.2), is 0.97, which falls under a near perfect agreement, according to the Kappa statistic from Table 2.2.

Wrongly classified images

Throughout the experiment with different types of CNNs, some particular images were noticed as they were consistently mistaken by the same, incorrect objects. Seven of the most common images that were wrongly classified by the CNN are represented in Figure 2.38.

The training dataset was analysed and specific images from other classes, that resembled the original image, were found. For example, certain images where the traffic cone had higher reflectivity were mistaken by images of a traffic sign; some mannequin images were classified as dogs, because at 90° rotation, dog scans look similar to the mannequin scans.

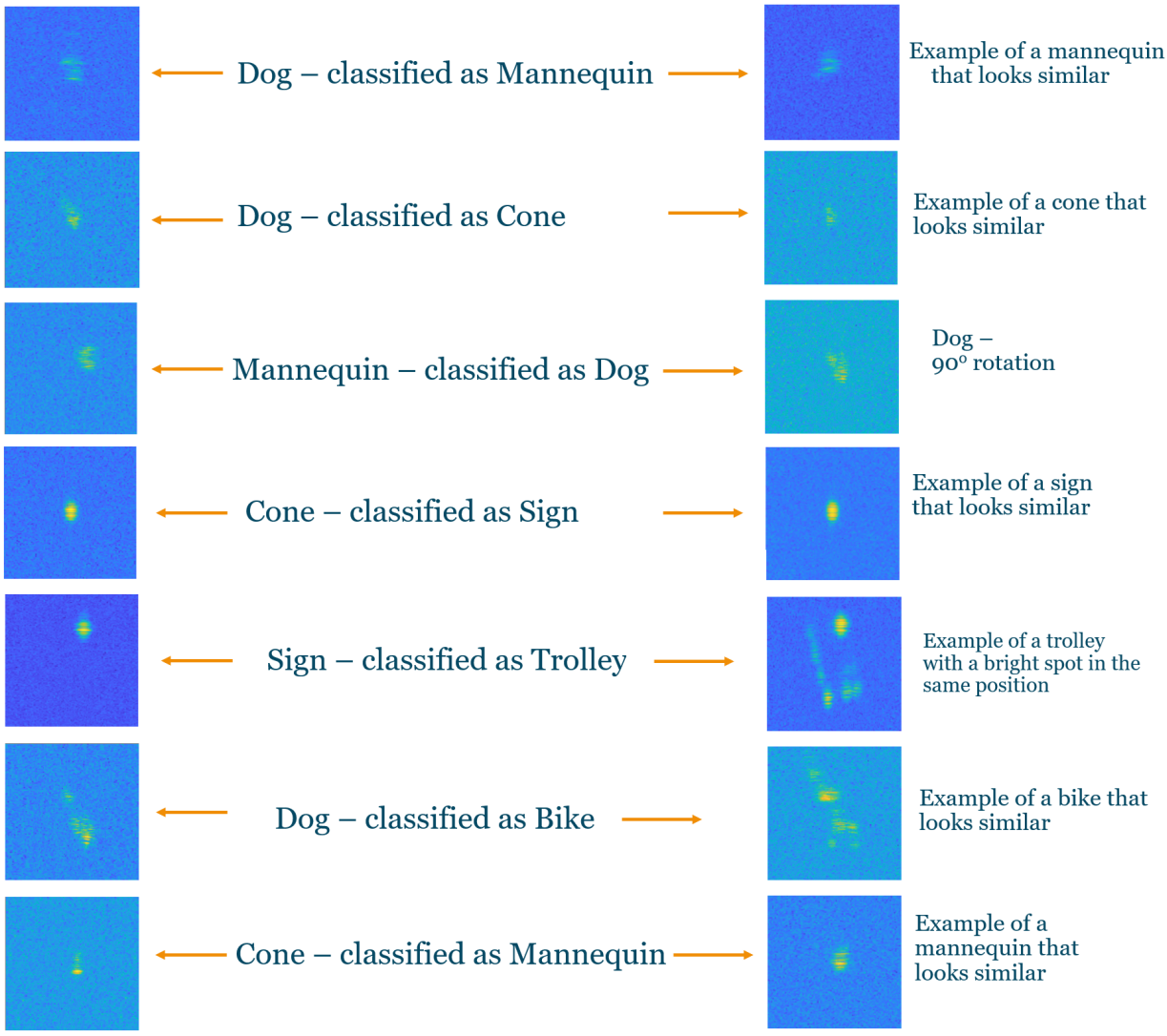


Figure 2.38: Examples of wrongly classified images.

The reason for these misinterpretations is that the output feature maps of the images were too similar to certain output feature maps of images from other classes and the one-shot learning approach was not suitable. In order to overcome these drawbacks, the dataset can be extended so that more training images will be presented to the neural network. Alternatively, data augmentation and shuffling the training and testing data sets might be appropriate solutions to address this issue.

2.2.7 Siamese Results

A different method to classify radar imagery objects that was analysed in this project is the siamese neural network. As presented in Section 2.1.4, the architecture of a siamese network is composed by two identical sub-networks with the same weights and parameters. In this project, each sub-network is implemented as a CNN.

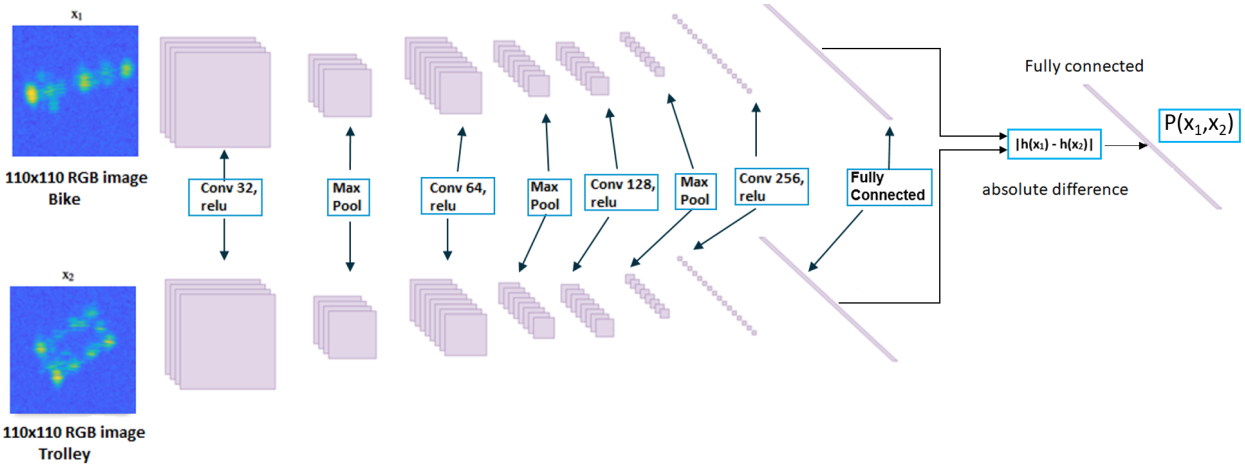


Figure 2.39: Siamese neural network architecture with 4 convolutional layers.

For this experiment, three different architectures of siamese neural network were analysed, each one consisting of two identical CNN subnetworks with two, three and four convolutional layers each (Figure 2.39). The architecture of the network layers is detailed in Section 2.2.6. Each network was trained on 255 pairs of images from the training dataset and tested on 176 pairs of images from the testing dataset, chosen randomly. Each pair was assigned a logical value of 1 if the images belonged to the same class and 0 otherwise (Figure 2.40).

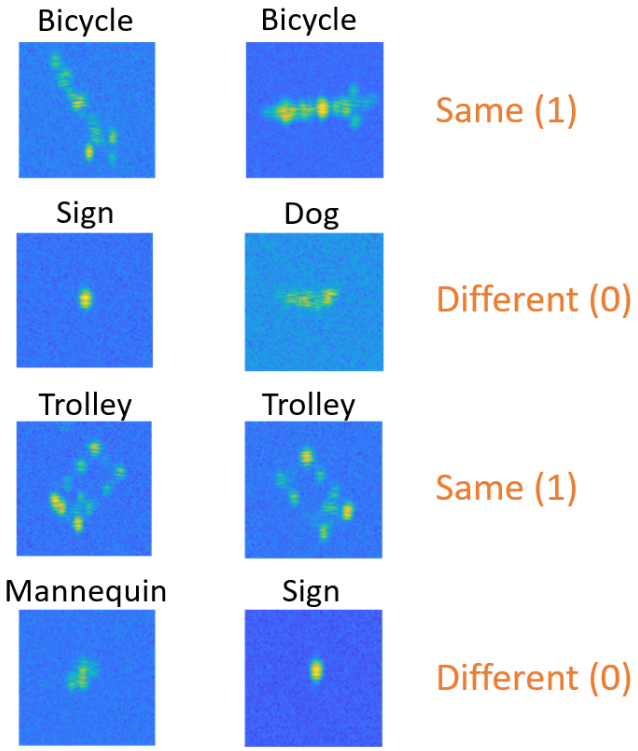


Figure 2.40: Example of radar training pairs for the siamese neural network.

Figure 2.41 shows the training and testing accuracies for different types of siamese networks applied on the radar imagery dataset. The best testing accuracy was 95.86%, obtained with four convolutional layers. It can be observed that the siamese testing accuracies for networks with different number of convolutional layers should not be neglected, though they are lower than the CNN testing accurracies (Figure 2.36). Therefore, increasing the number of convolutional layers for the siamese network beyond four layers was considered redundant. Moreover, due to the fact that a siamese network employs two convolutional networks which function in parallel, the computational load for a siamese network is doubled by comparison with the CNN workload. The training accuracy was consistently higher than the testing accuracy which is a normal behaviour, hence the networks performed well on both testing and training sets and did not overfit the data.

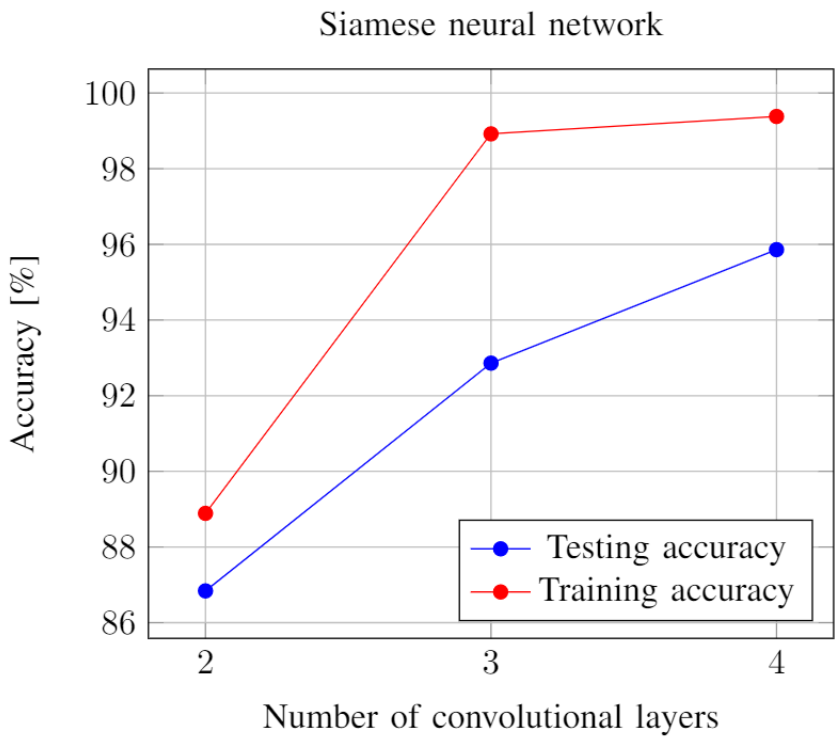


Figure 2.41: Testing and training accuracies for different siamese neural networks.

Although the siamese network achieved better results than the CNN on the MNIST dataset, it can be concluded that, for this particular type of application on a radar imagery dataset and given the current siamese and CNN architectures and parameters, the CNN with 5 convolutional layers still remains the effective method for classification.

2.3 Discussion

Throughout this chapter, the research findings have suggested that by setting the right hyperparameters, such as the batch size, the number of training epochs, the activation and the loss function, the optimisation algorithm and the LR, and by carefully optimising the computational load and the format of the input data, a DNN can be successfully utilised for classifying several roadside objects in low-THz radar imagery.

After the input images have been processed to a convenient form to be used by the neural networks, tests have been carried out to optimise the computational load of the model. These experiments involved reducing the image dimensions by 75%, from 220x220 pixels down to 110x110 pixels and lowering the depth of the images by converting the original RGB images with three dimensions into single dimension, grayscale images. It was found that, for this type of image, RGB images are more accurate than the grayscale ones and reduced size images with 110x110 pixels lessen the computational load of the model, without affecting the training and the testing accuracies.

Moreover, tests have been performed for improving the architecture of the neural network model by varying the number of the convolutional, max pooling and dropout layers. It can be concluded that the best testing accuracy on the radar imagery dataset was as high as 98.78%, obtained with a CNN which consisted of five convolutional layers and 20% dropout of the activations. In addition, a siamese neural network was implemented, although, unlike CNNs which are used for classifying images, the objective of a siamese network is to find the similarities and dissimilarities between pairs of objects. The best accuracy for the siamese network was 95.86%, achieved with four convolutional layers per sub-network. It should be noted that the radar dataset consisted of different images, thus the CNN achieved outstanding results for one-shot recognition.

In general, one of the drawbacks of neural networks is that the model has to be trained multiple times with different parameters and architectures, until an optimal model is achieved. In this project, neural networks have been trained on the same dataset over 60 times to produce a model with a fine adjustment of the architecture and hyperparameters. This process is complex and laborious, though essential for developing a neural network for specific types of applications and datasets, that does not overfit and can generalise well on unseen data.

The shortcoming of this work is that the radar imagery database for training and testing is limited, comprising of almost 2,500 distinct images of six roadside targets in controlled environment. Extra targets and images are required for the model to be able to recognise more sophisticated and distinct patterns.

2.4 Conclusion

In this chapter, a method for classification of different roadside targets in low-THz radar imagery using DNNs has been presented. Two different types of neural networks have been considered for image classification and for finding similarities/dissimilarities between pairs of objects: Convolutional Neural Networks (CNNs) and Siamese Neural Networks. This is the first publication on classification of roadside targets for autonomous driving on low-THz radar imagery [57].

For both types of networks, the architecture and the hyperparameters have been adjusted to achieve the best performance. The present findings confirm that DNNs can be successfully used for this type of application, due to the high accuracy on testing dataset (98.78%) of radar imagery that was achieved throughout this work.

In conclusion, although the presented method has its limitations, the current results on a radar imagery dataset provide a basis for developing an all-weather sensing system for autonomous vehicles, therefore further work will be carried out in this direction in the following chapters. Firstly, the existing radar imagery dataset will be extended with more radar scans of other roadside targets, especially in outdoor environment, so that the model will be reliable for on-road use. Secondly, a DNN will be developed with a more elaborate architecture that would be able to detect multiple objects in the same frame, as well as to generalise better and make precise predictions in real situations.

Chapter Three

Object Detection

3.1 Introduction

This chapter will present a solution to the current challenges of the imaging radar to respond the demands of autonomy for detection and classification of targets in radar imagery, which traditionally has been considered as clutter. The proposed object detection method is defined in a new way, as opposed to the traditional object detection methods in the radar related contexts. The current works is the first application of this novel approach and follows the previous work on classification. This approach is based on deep neural networks (DNNs) for object detection on outdoor radar images, as well as indoor images taken in controlled environment. Object detection was performed using two detectors based on neural networks and the evaluation proved that this method can be successfully used on radar imagery for autonomous applications. The principles of object detection in machine learning, the experimental setup, datasets, and the implementation of the two detectors will be detailed hereinafter.

3.1.1 State of the Art

The increased need for developing intelligent driving systems involves extensive research for innovations in terms of vehicle safety. For obstacle detection and collision avoidance systems in autonomous driving, radar imagery is preferred when optical cameras and lidar cannot provide usable imagery in adverse weather conditions, such as fog, spray, heavy rain or snow. Therefore, a data fusion between the information obtained as a result of target classification performed by the deep neural network and other information received from electro-optical sensors, such as lidar and stereo-video camera, is preferred and has the potential to produce a detailed map of the surroundings to help implement an all-weather sensing system. A recent, comprehensive survey on fusion methods and challenges is presented in [95].

This chapter aims to present the results of object detection with DNNs, applied on both indoor and outdoor radar images of roadside targets. This work follows previous research on high resolution automotive radar imagery classification [57] and focuses on two better known object detection architectures, the Faster Region Based Convolutional Neural Network (Faster R-CNN) [96] and the Single Shot Multibox Detector (SSD) [97]. Therefore, the single target classification from Chapter 2 is extended to multiple object detection in the same frame, where the objects are classified using rectangle bounding boxes with an associated classification score. Hence, the location of targets in the image can also be estimated using the coordinates of the bounding box.

Machine learning and computer vision techniques for radar applications have been investigated for decades. Example of state of the art methods that use artificial neural networks for segmentation of doppler radar images or LiDAR images, as well as classification in SAR images, have been mentioned in the previous chapter. Fortunately, the recent develop-

ments in terms of hardware capabilities supported the implementation of deeper and more sophisticated neural network architectures, such as deep object detectors.

In terms of autonomous applications, Lombacher et al. have achieved notable research on the potential of radar-based object recognition using deep learning methods [98] and semantic radar grid maps for static object classification in autonomous driving [99]. The implementation of a CNN classifier for roadside targets in low-THz radar imagery was presented in [57], although a fully reliable system that annotates the entire vehicle's environment based on these images has not yet been developed.

Currently, there is little published research on object detection with DNNs for automotive radar applications. Simon Chadwick et al.'s notable publication [100] incorporates radar data in order to increase the performance of optical object detection algorithms in difficult situations, such as detecting small, distant objects. However, this approach focuses only on vehicle detection, therefore there is still a gap in object detection performed on radar imagery of any type. To the knowledge of the author, this work is the first attempt at object detection on radar images for multiple roadside targets and was published at the 2020 European Microwave Week [58].

3.1.2 Traditional Radar Target Detection

Threshold Detection

In traditional radar, the process of detecting a target is based on the concept of "threshold detection" [4]. The radar measurement is compared with a set threshold and there are two hypothesis to choose from. If the measurement exceeds the threshold, then the measurement corresponds to target and interference, hence target-plus-interference hypothesis (H1 hypothesis). On the other hand, if the measurement is below the threshold it corresponds to interference (receiver noise), clutter (background returns, reflections from other objects in the scene that are not targets), or thermal noise, and it is associated with the null hypothesis (H0 hypothesis). The threshold should be carefully chosen in order to achieve the highest probability of detection for a given Signal to Noise Ratio (SNR).

A radar system should be able to maintain a specified probability of false alarm, which arises when a measured radar value exceeds the threshold, although it was caused by a source of interference and not a target, hence a false detection. False alarms can impair the performance of a radar system significantly, due to the additional processing and resources required in such situations. Therefore, Constant False Alarm Rate (CFAR) detectors were implemented, which can account for changes in interference and modify the detection threshold accordingly, so that the false alarm rate of the system is maintained.

It is essential to emphasise the fact that in this chapter, the concept of "Object Detection" does not involve the traditional radar detection techniques mentioned above. It refers to the concept of object detection in machine learning, which uses DNNs and computer vision algorithms. This method is implemented in a novel way for detecting roadside targets in radar imagery and it is described in the following sections.

3.2 Object Detection in Machine Learning

Object detection is a deep learning concept in conventional optical imagery, used when multiple objects are present in the same image; it implies identifying the location of each single object and classifying them. Ground-truth bounding boxes are compared with the predicted boxes and a classification score is calculated for each. In machine learning terminology, ground truth refers to the information that is known to be real, from measurements and direct observations, which are often represented as labels. For this application, the ground truth was obtained by manually labelling the objects in radar images, using optical images as reference.

An example of object detection for autonomous driving based on optical images can be seen in Figure 3.1. For better visualisation, bounding boxes have distinct colours, corresponding to the class colour (vehicle - red, road sign - blue, person - green, traffic light - yellow). Bounding boxes have an associated classification score which represents how confident the system is that the detected object belongs to the proposed class.

Figure 3.1: Object detection on a complex urban scene scenario for autonomous driving [95].

As opposed to classification, object detection is a more advanced method which uses deeper neural networks in order to classify and find the location of multiple objects in an image. Object detection is the preferred method for real-time and real-life applications such as object classification in autonomous driving.

The main idea of traditional machine learning object detection algorithms is to take multiple Regions of Interest (RoIs) and apply a classifier in each region [101]. This method is often referred to as a two-step approach. Essentially, each region is passed to a CNN for classification, after which the regions are re-combined to form the original image with the detected objects. However, this basic approach in region-based networks (R-CNN [101] and Fast R-CNN [102]) has major disadvantages in terms of computational effort, given by the unknown ratio of objects in the image, which requires a selective search over a large number of regions. Therefore, algorithms have been implemented in order to reduce the number of RoIs, by selecting the RoIs with a Region Proposal Network (RPN). This was implemented in Faster R-CNN [96].

Another powerful, yet easy and widely used object detection model is the Single Shot Multibox Detector (SSD) [97]. SSD is considered a one-stage detector because it predicts all the bounding boxes in an image in one pass. The fully-convolutional approach of SSD increases the computation speed and is preferred in applications where high accuracy is not crucial. More details about the image processing, the anchor boxes generation and the detection steps can be found in [96] and [97], though the block diagrams and a short description of each detector will also be covered in the following sections.

The performances of Faster R-CNN and SSD models were evaluated in this chapter for object detection in automotive radar imagery because they provide up to 30 times greater processing speed (with a prediction time per image of ≈ 0.2 seconds) when compared to selective search models. This facilitates the real time detection. However, the choice of the

right object detector depends on the application and the desired trade off between speed and accuracy.

3.2.1 Object Detection versus Classification

Image classification is used for identifying the class of a single object contained in an image. For example, if the Figure 3.2 (left) is passed as input into a classifier, it will output the class it belongs to (a bicycle, in this instance). On the other hand, Figure 3.2 (right) contains both a bicycle and a trolley. A multilabel classifier could be trained, though the issue that arises is that the locations of the objects will be unknown.

Image Localization can be used to specify the location of a single object in an image. In case there are multiple objects present, the concept of Object Detection is preferred, which specifies the location of multiple objects in the image. The locations of objects, along with the classes, can be predicted using Object Detection.

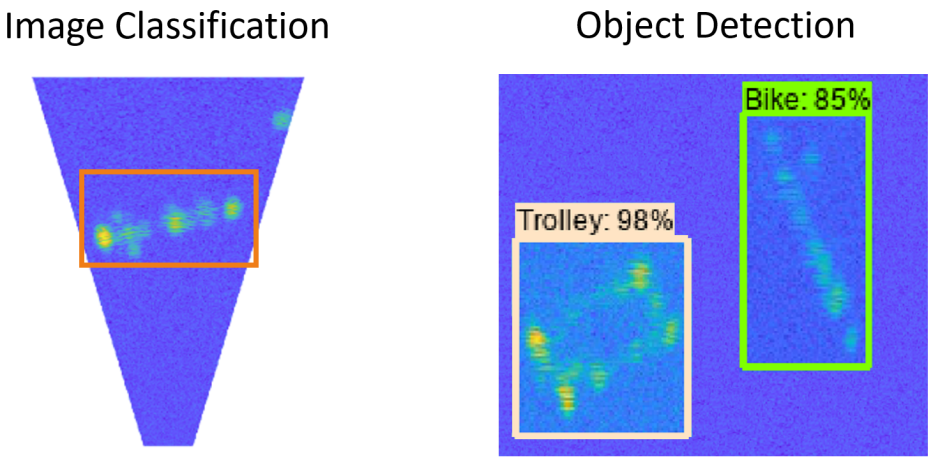


Figure 3.2: Object Detection versus Classification.

3.2.2 Faster R-CNN

Faster R-CNN is composed of two main modules: the first module is a deep Fully Convolutional Network (FCN) that proposes the RoIs for the objects (also known as the Region Proposal Network (RPN)) and the second module is the actual Faster R-CNN detector that takes in the object proposals and outputs the classified objects.

The block diagram in Figure 3.3 briefly describes how the Faster R-CNN detector works. It takes the image as an input (in this case a radar range profile) and passes it to the FCN, which generates the convolutional feature maps. A RPN is applied on these maps, which returns the object proposals. A RoI pooling layer is applied to reshape all the proposals into a fixed size, so that they can be fed into a fully connected network, to classify the object and also to output the bounding boxes.

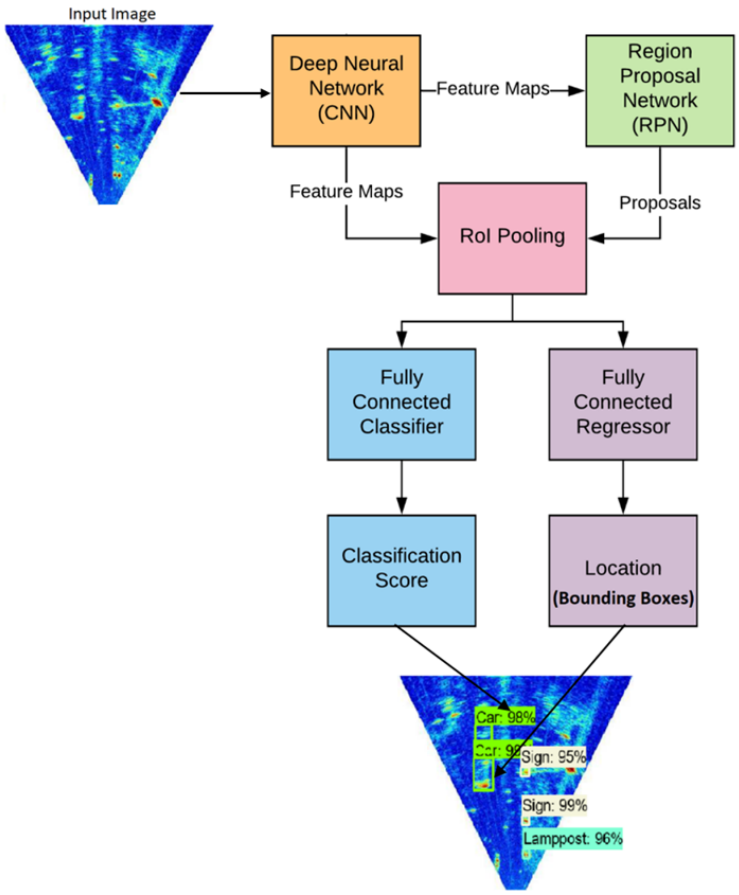


Figure 3.3: Faster R-CNN block diagram.

Region Proposal Network (RPN)

RPN is one module of Faster R-CNN, the network that predicts the regions where the objects of interest might be located. R-CNN's accuracy of classifying the objects depends directly on the performance of the RPN. This module is often called the "attention" mechanism, as it directs where the detector looks. Figure 3.4 (from the official Faster R-CNN paper [96]) shows the functionality of the RPN module within the Faster R-CNN object detector.

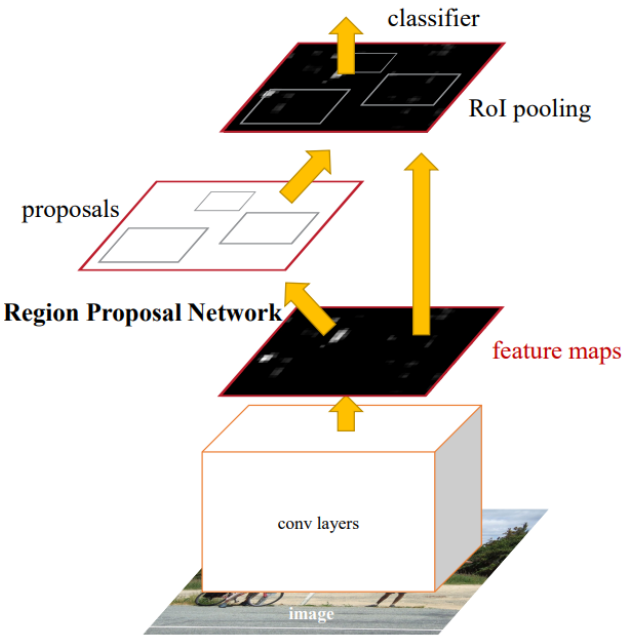


Figure 3.4: RPN module as part of the Faster R-CNN object detector [96].

RPN, as its name suggests, outputs a set of proposals of different sizes with associated objectness scores (the probability that the object belongs to the specified set of object classes). The proposals (rectangle windows) are generated by sliding a small network over the feature maps (the output of the last convolutional layer). The RPN network that works in a sliding window fashion is illustrated in Figure 3.5.

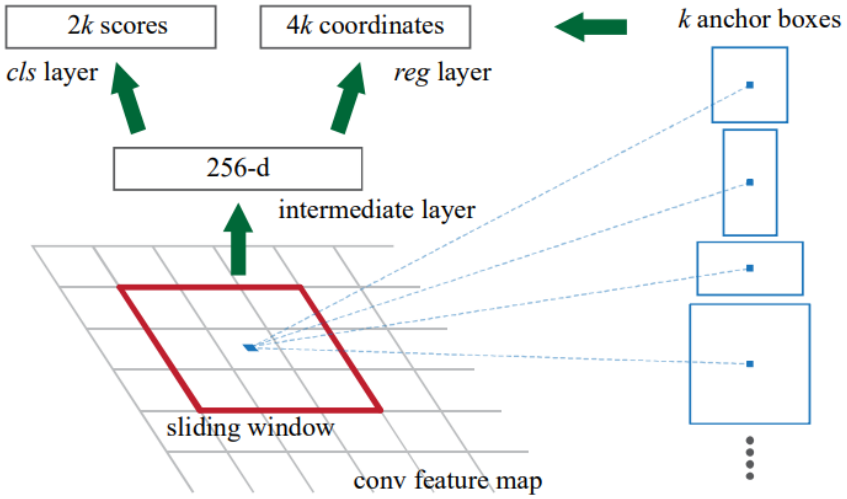


Figure 3.5: RPN structure [96].

At each sliding window there are maximum $k = 9$ anchor boxes generated (3 different scales x 3 different aspect ratios). The anchor boxes are centred in the sliding window. The total number of anchors generated for each feature map is W x H x k, where W and H are the width and the height of the feature map. The anchor boxes are ultimately fed into the classifier (cls) and the regression (reg) twin layers (one convolutional layer). The classification step here means finding the probability that the anchor is an object (the objectness score) without considering the class an object belongs to. The regressor adjusts the anchors to better fit the object. The regression layer will output $4k$ coordinates (4 coordinates for each box) and the classification layer will output $2k$ scores (each box has two scores, the probability of the anchor to be an object of interest and the opposite probability of not being an object of interest).

After the RPN step, the RoI pooling layer will take each proposal and crop it to contain just the object, so it will extract a fixed size feature map for each anchor. These are the final feature maps that are passed to the fully connected layer for classification (the softmax layer) and bounding box prediction.

3.2.3 SSD

Another powerful and widely used detector is the Single Shot Multibox Detector (SSD). This detector is considered a one-stage detector because it predicts all the bounding boxes in an image in one forward pass (single shot).

In the block diagram (Figure 3.6) it can be seen that SSD performs in a similar fashion with Faster R-CNN, but it skips the region proposal step before finding the RoIs. The technique for bounding box regression in SSD is called Multibox [103]. Multibox uses the K-means clustering algorithm to generate 800 anchors. According to the authors of RPN, the RPN method reduces the model size considerably by 2 orders of magnitude, when compared to Multibox. Considering that proposals are the main computational bottleneck for state of the art object detection at test time, this is an important benefit of RPN. Another advantage that RPN has over Multibox is that the anchor boxes used in RPN are also translation invariant, meaning if an object is translated in an image, the proposal is also translated.

SSD has a fully-convolutional architecture and an increased computing speed, though results showed that it performed much worse at detecting small objects. Hence, SSD is preferred in applications where speed is a priority, while Faster R-CNN is more accurate. This research concluded that Faster R-CNN performed better than SSD on the radar dataset and the results will be presented in the following sections.

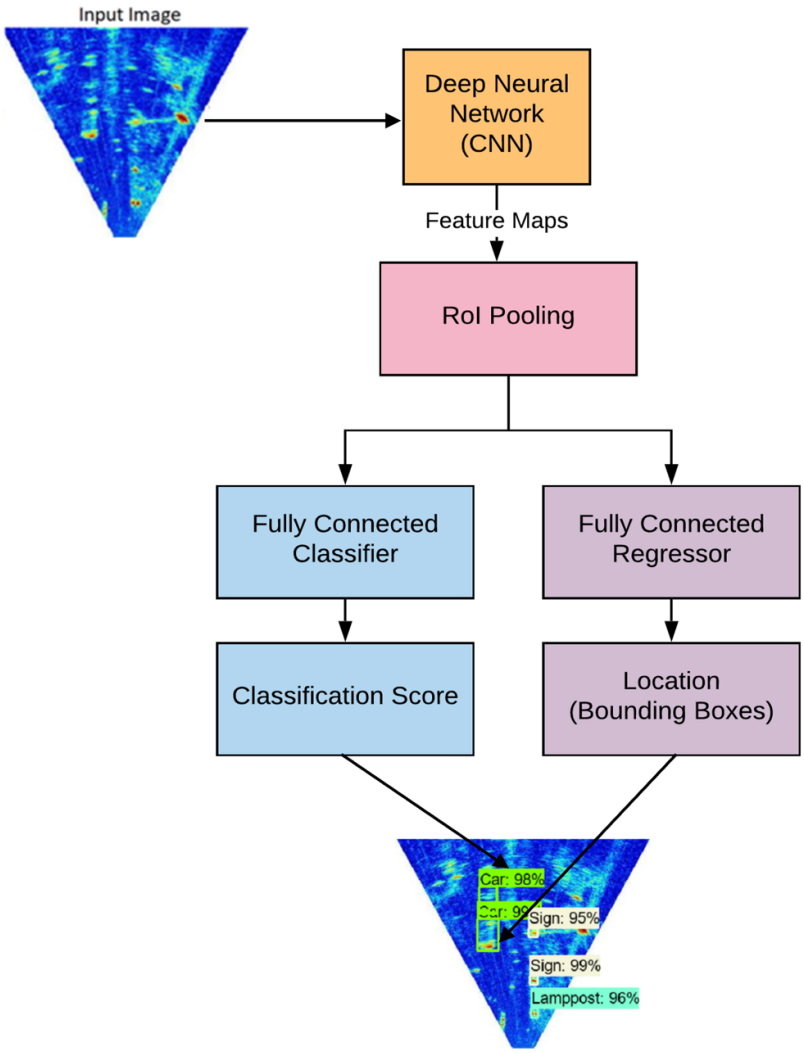


Figure 3.6: SSD block diagram.

3.2.4 Transfer Learning

Transfer learning is a deep learning method where a model trained for a purpose is reused as the starting point for another task. The pre-trained model is fine tuned on another dataset, specific to its task. This technique allows for rapid progress as the detector does not spend a lot of time on learning the initial features of a model at training time, as it inherits them from a pre-trained model and then it modifies them to fit the current dataset.

Tensorflow Object Detection API provides a collection of detection models pre-trained on several well known datasets, such as the COCO dataset [104], the KITTI dataset [23], the Open Images dataset [105], etc. The models used in this project were pre-trained on the COCO dataset, a large scale dataset for object detection, published by Microsoft. At the time the model was used, it already knew that the aim was to find objects of interest in RGB images. The optical targets were swapped for radar targets and the model was fine tuned during the training process.

3.3 Object Detection in Automotive Radar Imagery

3.3.1 Experimental Setup and Datasets

Object detection was performed on two datasets of scanned scene maps, indoor and outdoor, obtained with two LFMCW radars, custom-made by ELVA-1 [106].

Indoor Dataset

The indoor experiment was performed in laboratory environment at the University of Birmingham and the dataset consisted of radar images of six single roadside targets (a dog, a cone, a mannequin, a traffic cone, a traffic sign and a trolley). This dataset, which was taken with a 300 GHz LFMCW radar, is the same dataset that was used for classification and the experimental setup was described previously in Section 2.2.2. Also, the description of signal generation and data acquisition with the low-THz radar can be found in [62].

However, the main purpose of this study is to detect multiple objects in the same scene (as opposed to the previous classification-only algorithms described in Chapter 2). Hence, more than one target is required in the same image for an efficient object detection. Therefore, the images of single targets were merged into a larger image, similar to Figure 3.7. The dataset contained $\approx$ 1,000 such images, split into training (80%) and testing (20%).

Object detection was firstly performed on the indoor, controlled environment images, to determine whether this technique is feasible or not for radar imagery, prior to starting the labelling process for the outdoor images. The 300 GHz radar has not been used for the outdoor measurements due to the lack of power of the current experimental prototype. Instead, a 79 GHz radar with the same operation principle as the 300 GHz radar was used.

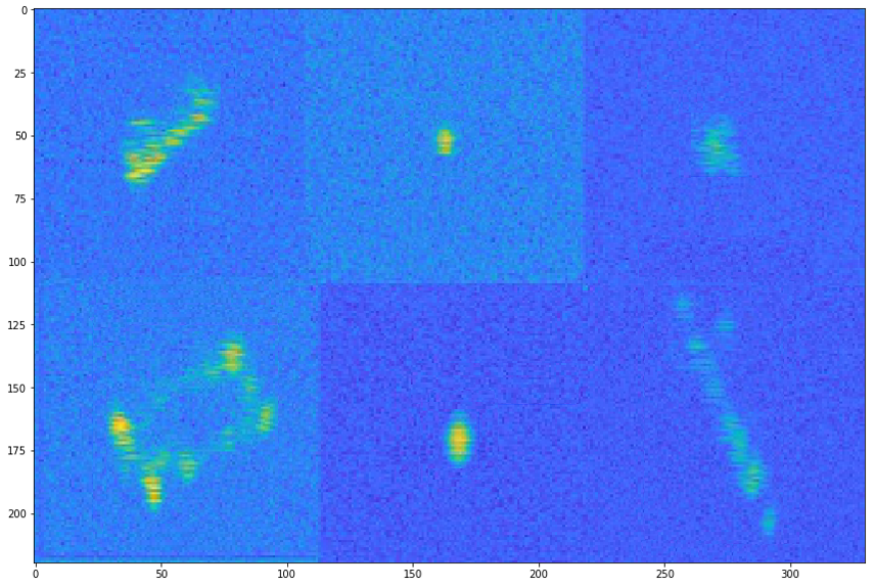


Figure 3.7: Image for object detection composed of merged radar images of single roadside targets, taken in laboratory environment.

Outdoor Dataset

For the outdoor experiment, the setup was mounted on a vehicle and consisted of a 79 GHz wideband, experimental, LFMCW radar with one transmitter and two receivers positioned at different heights and a ZED stereo camera for ground truth reference. The setup can be seen in Figure 3.8 and the radar parameters are found in Table 3.1.

The radar rotated on a turntable to scan the scene in azimuth, forming a single frame of the radar video. The measurements were taken while driving around the campus of the University of Birmingham during daytime and night time and the dataset consisted of 27 radar videos of different lengths, giving a total of $\approx$ 2,000 frames. The speed of the vehicle was chosen in order to avoid range walk ($\approx$ 5 mph). Although two receivers were used to estimate the height of the objects, in this experiment we consider only 2D radar maps from receiver 1.

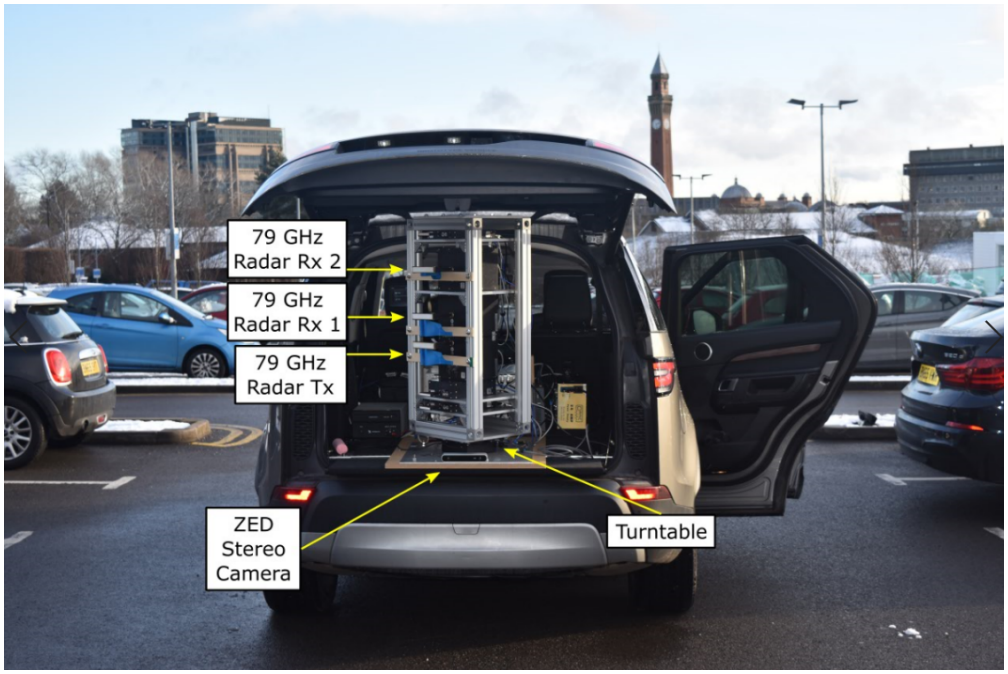


Figure 3.8: Radar experimental setup for the outdoor environment data collection.

Table 3.1: Radar parameters used for the outdoor environment measurements.

Parameter	**Value**
Sweep bandwidth	5 GHz
Sweep duration	1 ms
Range resolution	∼3 cm
Centre frequency	78.5 GHz
Field of view	90°
Number of chirps across field of view	132
Pulse repetition interval	4.3 ms
Scan duration	1 s
Frame rate	1 Hz
Output power	15 dBm
Azimuthal beamwidth (2-way)	1.7°
Elevation beamwidth (2-way)	8.5°
Transmitter height (from ground)	1.3 m
Receiver 1 height (from ground)	1.42 m
Receiver 2 height (from ground)	1.59 m

Annotations

Five classes of roadside objects were considered for this application (cars, cyclists, lampposts, pedestrians and traffic signs). The targets were manually labelled with rectangular boxes in each frame, using the accompanying optical images from the ZED stereo camera for guidance and an open-source Python graphical image annotation tool [107]. The annotations process is illustrated in Figure 3.9.

A summary of the dataset, which was split into training (80%) and testing (20%), can be found in Table 3.2. This dataset was successfully used for object detection with Faster R-CNN and SSD and the outcomes are presented in this chapter.

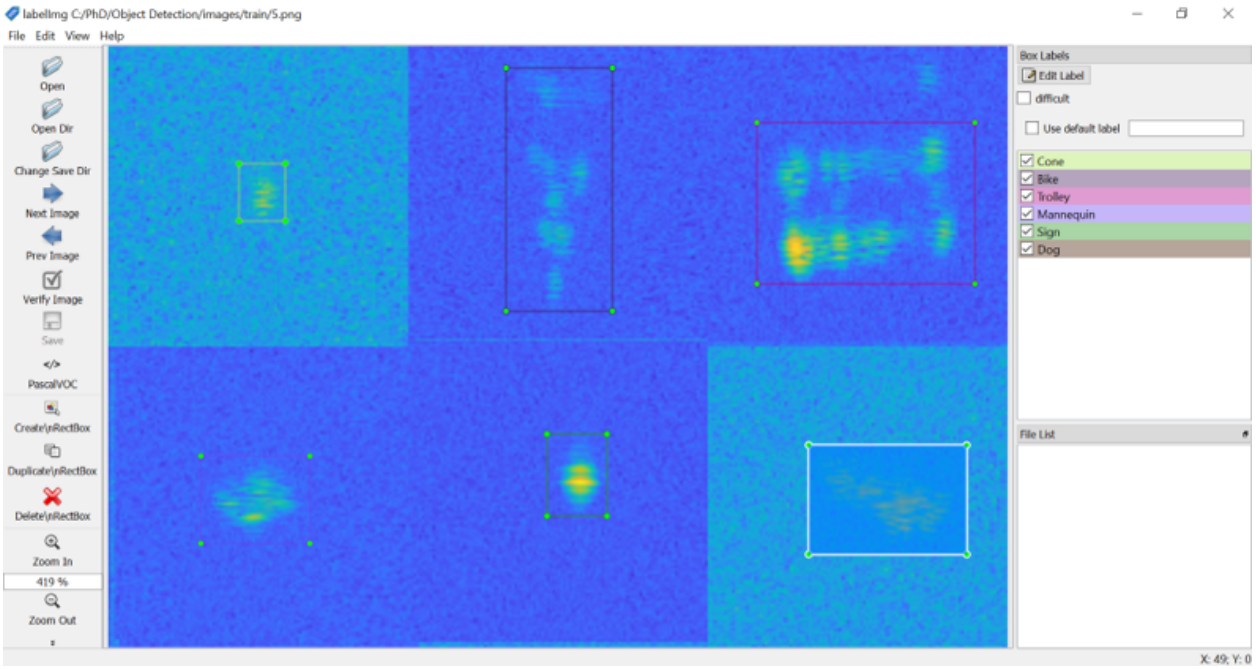


Figure 3.9: The labelling process.

Table 3.2: Training and testing labels for the outdoor dataset.

	Car	**Cyclist**	**Lamppost**	**Pedestrian**	**Sign**
Train labels	2,603	362	917	703	1,209
Test labels	851	91	244	194	302
Total labels	3,454	453	1,161	897	1,511

3.3.2 Evaluation Metrics for Object Detection

Precision and Recall

Precision (also known as positive predictive value) measures how accurate the predictions of the detector are, the percentage of the correct predictions. Recall (also called sensitivity) identifies the proportion of actual positives identified correctly.

$$Precision = \frac{TP}{TP + FP}. \tag{3.1}$$

$$Recall = \frac{TP}{TP + FN}. \tag{3.2}$$

where TP = true positives, FP = false positives, TN = true negatives and FN = false negatives.

Precision can be plotted against the recall and the resulting plot is called the precision-recall curve.

AP and mAP

Average Precision (AP) is a popular metric used to determine the accuracy of object detectors and Mean Average Precision (mAP) is the mean AP over all classes of objects. AP is calculated as the area under the precision-recall curve:

$$AP = \int_0^1 p(r)\, dr. \tag{3.3}$$

AP falls within 0 and 1, as precision and recall are between 0 and 1.

IoU

Intersection over Union (IoU), also known as the Jaccard index, measures the overlapping level between the bounding box predicted by the neural network and the ground truth bounding box, which was manually labelled.

The evaluation was performed using the Microsoft COCO evaluation metrics [104] and included the mAP over different IoU thresholds: from 0.5 to 0.95, 0.5 and 0.75. Traditionally, mAP is computed at IoU=0.5, but according to the COCO evaluation metrics, mAPs are averaged over 10 IoU thresholds of .50:.05:.95. Averaging over IoUs can help detectors to improve their localisations.

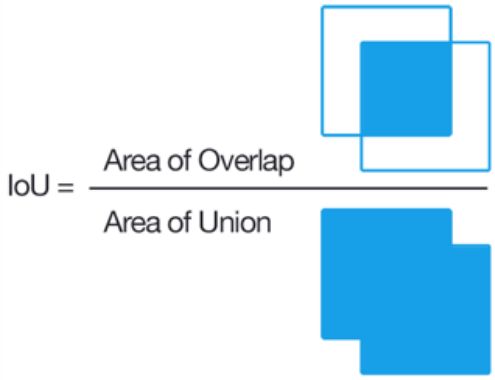


Figure 3.10: IoU calculation.

3.3.3 Object Detection Results

The methods used for object detection were the two detectors decribed above, Faster R-CNN with a Resnet 101 backbone (feature extractor) [108] and SSD with an Inception V2 backbone [109]. The backbones are very performant deep CNNs (101 layers and 164 layers, respectively). The two detectors were applied on the indoor and outdoor datasets from Section 3.3.1.

The results for the indoor and outdoor environment data sets can be found in Table 3.3. The performance of both detectors on the MS COCO data set is listed as well, for comparison. Figure 3.11 shows an example of object detection performed on a controlled radar image using Faster R-CNN, which led to better mAPs than the SSD detector, while training for 1,000 epochs. The mAP with a 50% IoU between the predictions and the ground truth achieved the value of 100%.

Table 3.3: Object Detection Evaluation

Data Set	**mAP@0.5:0.95 [%]**		**mAP@0.5 [%]**		**mAP@0.75 [%]**	
	Faster R-CNN	**SSD**	**Faster R-CNN**	**SSD**	**Faster R-CNN**	**SSD**
Indoor Environment	67.9	42.9	100	79.6	81.8	38.2
Outdoor Environment	28.6	20.5	61.5	57.2	21.5	9.1
MS COCO [104]	24.2	23.2	45.3	41.2	23.5	23.4

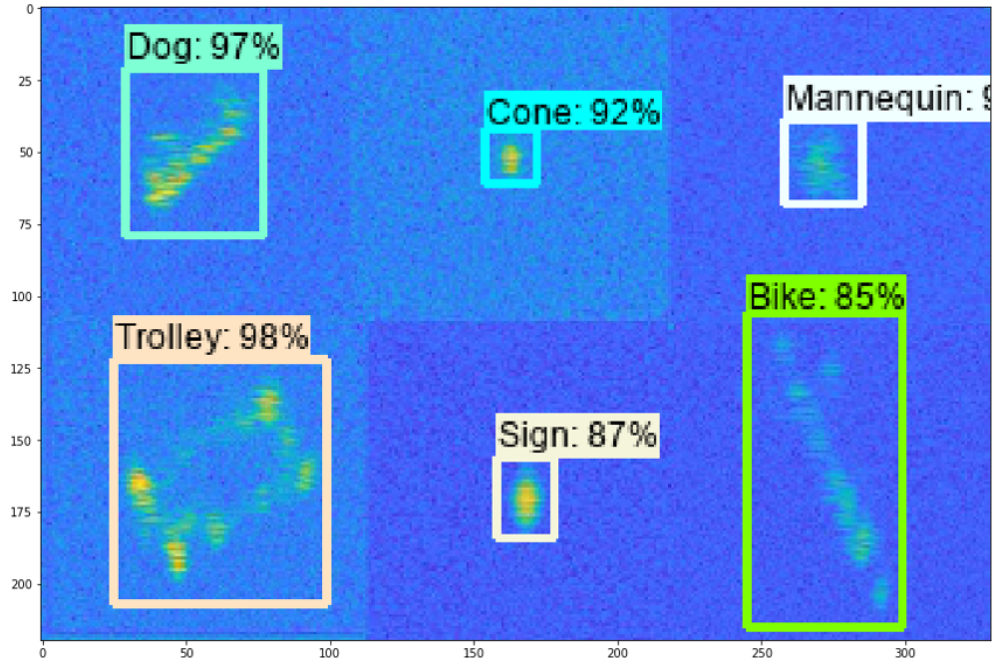


Figure 3.11: Object detection results on a radar image composed of merged, single images of roadside targets, taken in laboratory environment.

The outdoor radar data set was trained for 40,000 epochs with Faster R-CNN and SSD, respectively. The models were trained and tested in parallel, until the best evaluation metrics were achieved. The total training loss functions (the sum of the classification and localisation losses) for both detectors are shown in Figure 3.12.

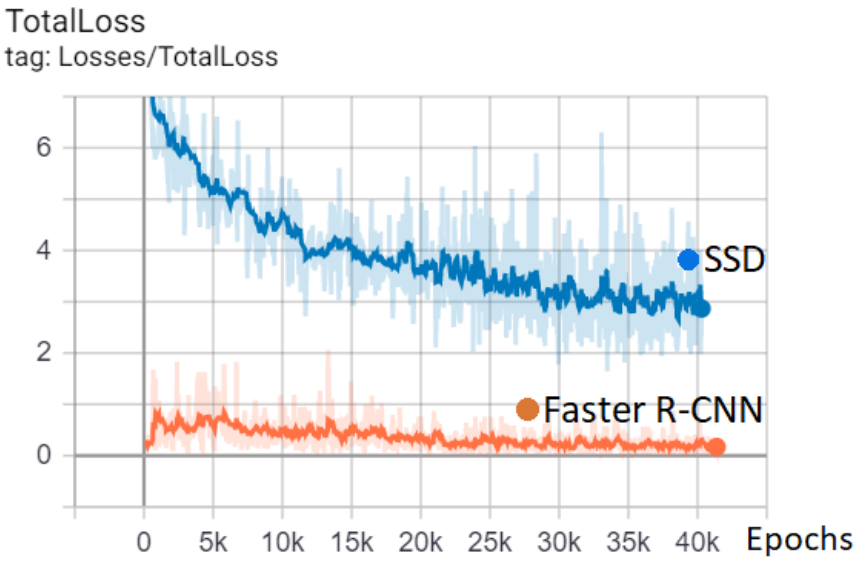


Figure 3.12: Total training losses for Faster R-CNN (orange) and SSD (blue) over 40,000 epochs, on the outdoor dataset.

Although the SSD training loss is steadily decreasing with each training epoch, the Faster R-CNN loss has much lower values, therefore greater mAPs (Table 3.3). A possible explanation for the better performance of Faster R-CNN can be found in a report by Google Research [110] about the trade-off between speed and accuracy for different detectors performing on the COCO data set. The study concluded that Faster R-CNN had the best accuracy, while SSD was faster and performed much worse at detecting small objects.

Examples of object detection performed on test images with Faster R-CNN, accompanied by the corresponding optical images and their 2D maps are illustrated in Figures 3.13, 3.14 and 3.15.

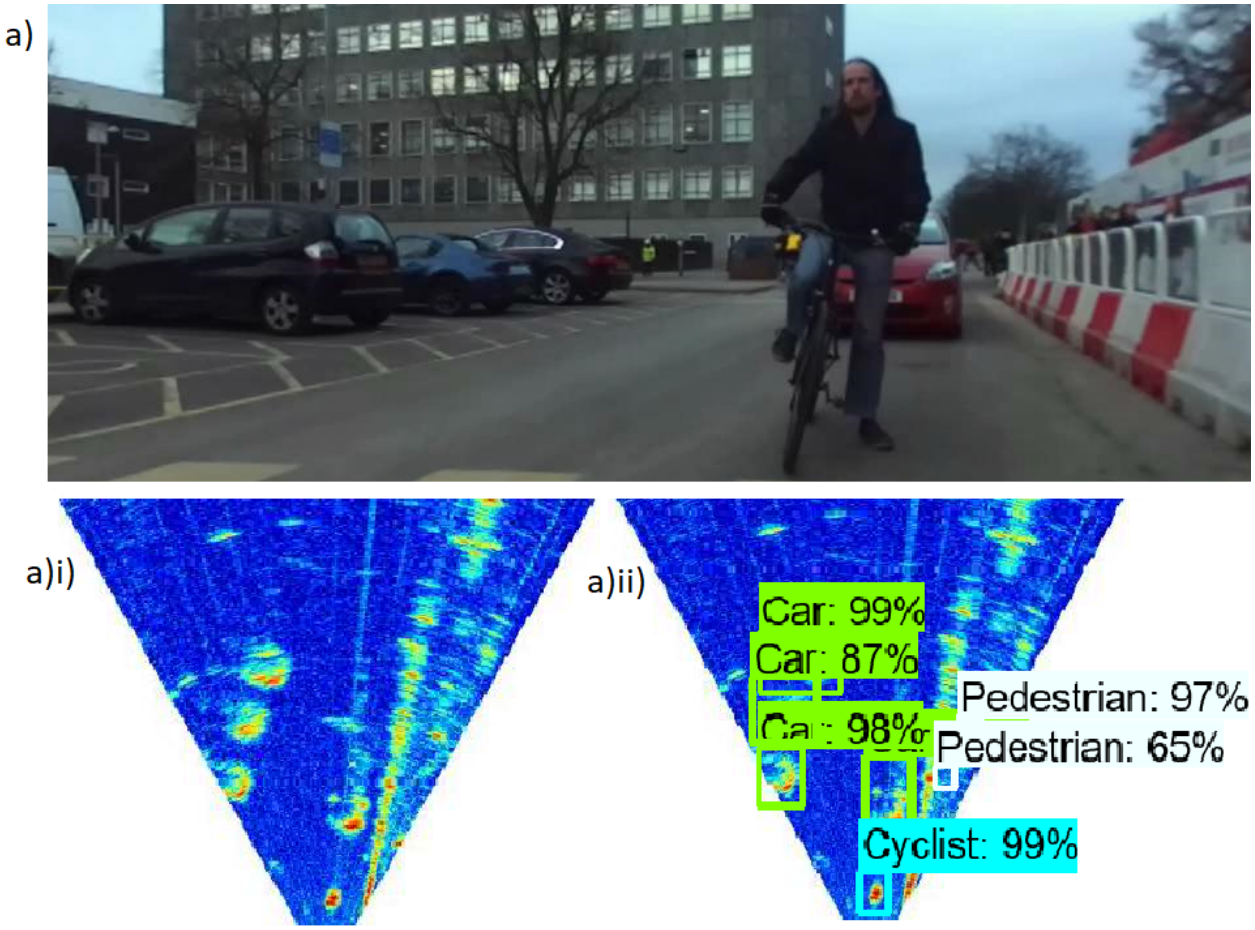


Figure 3.13: Example of an outdoor optical image (a), its corresponding 2D map (a)i), and the object detection results (a)ii) performed with Faster R-CNN.

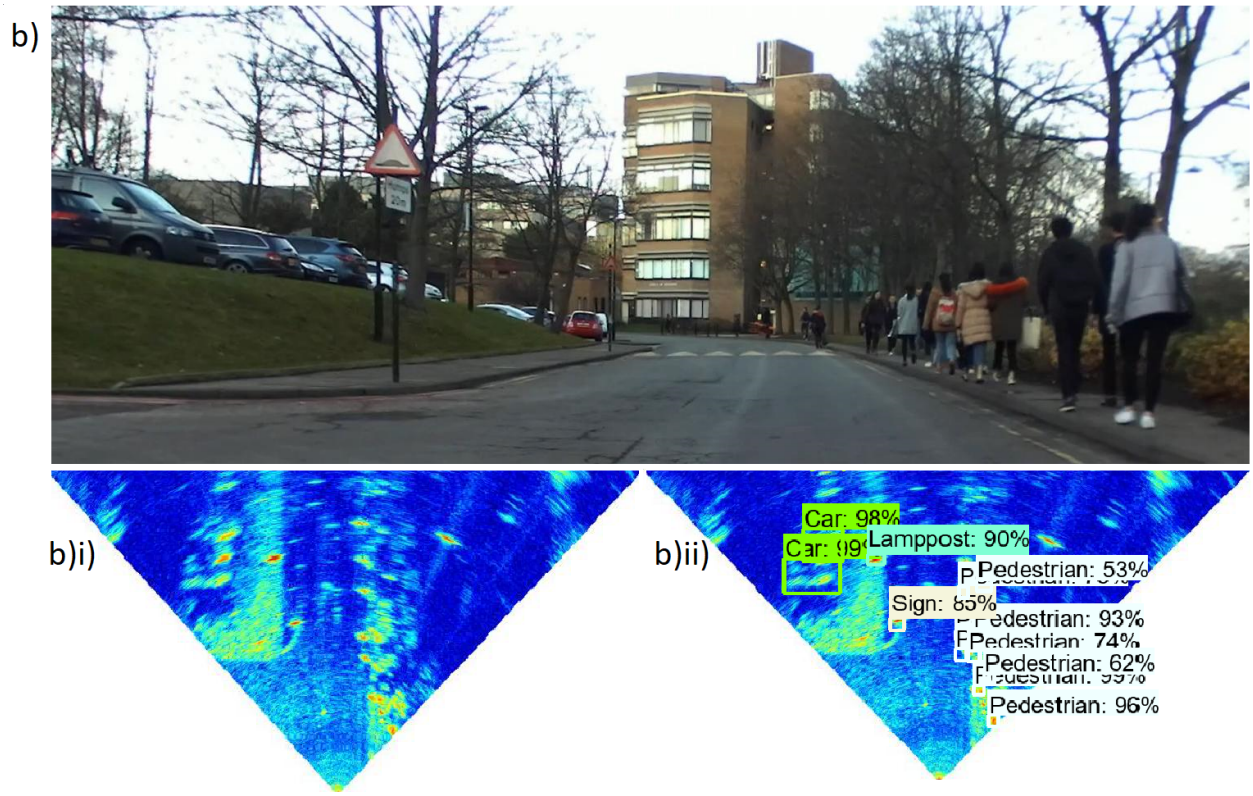


Figure 3.14: Example of an outdoor optical image (b), its corresponding 2D map (b)i), and the object detection results (b)ii) performed with Faster R-CNN.

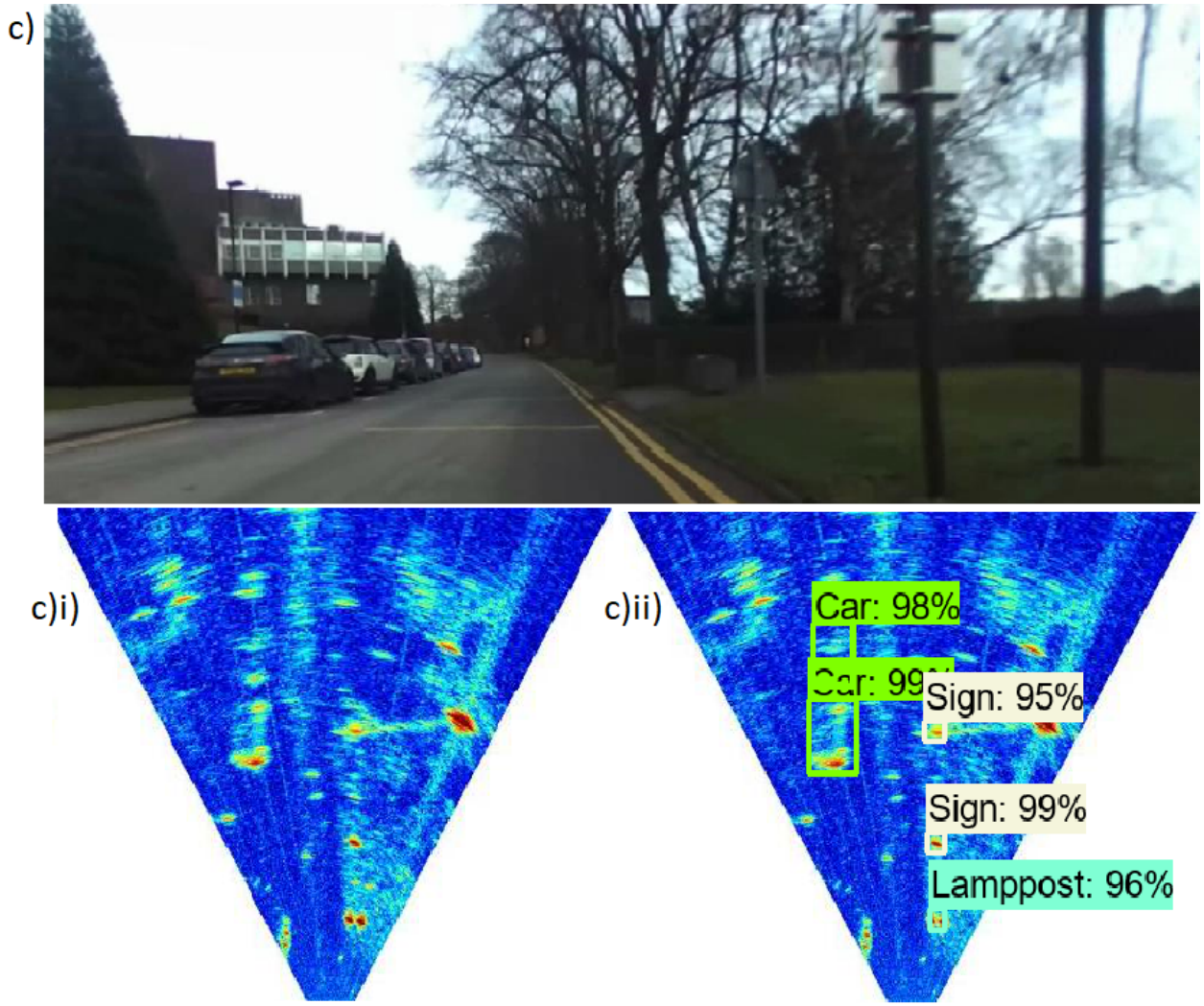


Figure 3.15: Example of an outdoor optical image (c), its corresponding 2D map (c)i), and the object detection results (c)ii) performed with Faster R-CNN.

3.4 Conclusion

This chapter presented two DNN object detection methods which were performed in a novel way on indoor and outdoor radar imagery, using the Faster R-CNN and the SSD detectors. The evaluation metrics and the results on the test data for Faster R-CNN confirmed that object detection using DNNs can be successfully performed on radar imagery to detect roadside objects for autonomous driving applications. This is the first application of DNN object detectors on indoor and outdoor automotive radar imagery for detecting the position and classifying several roadside targets and was presented at EuRAD in 2020.

The current results on radar imagery datasets prove that this concept can be further used for object detection and classification of dynamic radar videos in an all-weather sensing system for autonomous vehicles. The current work can be further improved by extending the current radar imagery datasets with more outdoor radar scans, as well as performing fusion between optical images and radar images and target tracking. The next chapter will present a novel system that combines the current object detector with a particle filter tracker to improve the performance and the computational cost.

Chapter Four

Object Detection and Tracking

4.1 Introduction

This chapter presents a novel approach for target detection in radar imagery, which combines an object detector and a multi target particle filter tracker. Object detection is implemented using deep neural networks, as opposed to the traditional radar object detection methods, as described in Chapter 3. This technique is applied to a dataset collected with the 79 GHz FMCW radar mounted on a vehicle. In this approach, object detection and tracking of roadside objects are performed in an alternating fashion to reduce the computational load required by the real time processing. The results and the thorough analysis of the parameters showed that this approach is feasible and can be successfully utilised in radar imagery for autonomous driving.

The increased need for developing intelligent driving systems attracts an extensive research on innovative sensing, processing and presentation of information. It was mentioned previously that for obstacle detection and collision avoidance systems in autonomous driving, radar imagery is preferred when optical cameras and LiDAR cannot provide usable imagery in adverse weather conditions, such as fog, spray, heavy rain or snow. Therefore,

the development of imaging automotive radar is imminent and a necessary demand to effectively provide fusion of data from radar and optical sensors, where currently DNNs are used for target detection. This data fusion has the potential to produce a detailed map of the surroundings to help implement an all-weather sensing system.

In this chapter, a novel method to process radar imagery will be introduced, which combines a DNN object detector and a particle filter tracker, with the purpose of improving target detection and recognition for autonomous driving on outdoor radar images for multiple roadside targets. This work is the first attempt at combining an object detection with a particle filter tracker and was published at the IEEE Radar Conference 2020, in Florence, Italy [59].

By combining two techniques which are competent on their own, we aim to increase their individual performances. To reduce the processing time required for autonomous applications, detection and tracking are proposed to be applied in turns on the radar frames.

This chapter will present the functionality of this technique, the results and the advantages of this application on outdoor radar videos of roadside targets for autonomous driving purposes.

4.1.1 State of the Art

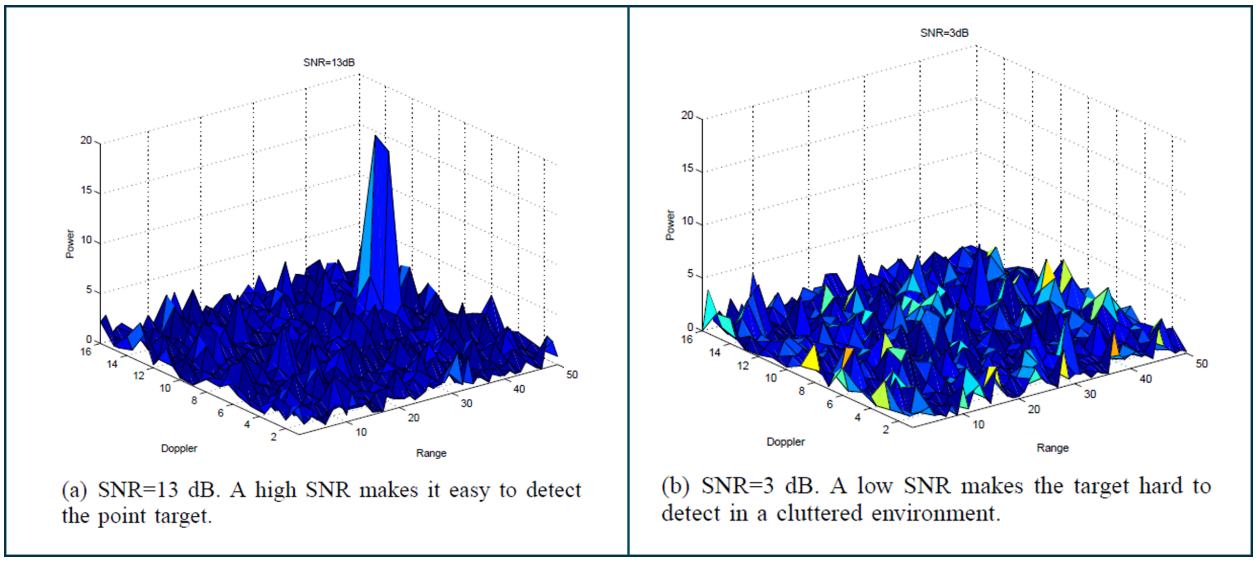
Methods for classification of radar targets using computer vision techniques based particularly on machine learning have been investigated over decades. Examples of such notable publications have been mentioned in Chapter 3. So far in this thesis, a novel process of object detection on radar images for autonomous driving has been detailed in Chapter 3 [58], as part of our previous research. The preceding work on classification of roadside targets performed on indoor radar imagery was presented in Chapter 2 [57].

The problem of tracking targets in radar imagery with particle filters has been investigated for a long time. A survey for existing particle filters for Multi Target Tracking (MTT) is available in [111]. In [112], a track-before-detect algorithm with particle filters is implemented for real-world data of X-band marine radar. This study assumes that the targets have elliptical shape and disregards the MTT case by dividing the image in separate regions which contain only single targets. The aim of this automotive radar application is to find the locations of the targets, as well as to classify and to track them, therefore our approach is to implement a multi target tracker which can detect multiple targets in the whole radar frame.

Particle filters have been confirmed to be a superior method for tracking extended targets when compared to algorithms based on a point target model assumption. Particle filters have also been used in research for automotive radar applications, such as for increasing pedestrian's safety [113], although this experiment was focused on tracking a single pedestrian only. To the best of our knowledge, there were no other previous attempts to combine DNN based object detection and tracking for automotive radar, therefore this chapter will focus on developing and analysing this kind of algorithm.

4.1.2 Traditional Radar Tracking Methods and Track Before Detect

In classic radar processing, traditional tracking methods are based on thresholding the raw data [114], as discussed in Section 3.1.2. Each detection, which corresponds to either a real target or a false alarm (clutter), can be filtered using a Kalman Filter or its variants to estimate the target state. However, this traditional approach has a number of limitations when tracking targets with low Signal to Noise Ratio (SNR), because the threshold must be low, as well. This leads to a high rate of false detections. A visual explanation of this issue can be found below, in Figure 4.1. In Figure 4.1 (a), the point target has a high-SNR, therefore it is easily detected because the received energy is above the detection threshold. In Figure 4.1 (b), the target is much weaker, so it is difficult to detect the target using classical extraction methods.



(a) SNR=13 dB. A high SNR makes it easy to detect the point target.

(b) SNR=3 dB. A low SNR makes the target hard to detect in a cluttered environment.

Figure 4.1: Illustrations of radar measurements for detecting a point target in (a) high-SNR and (b) low-SNR, where the power is plotted as a function of range and Doppler cells for one fixed bearing angle, P(r_t, d_t, b_t) [115].

Therefore, another approach has been developed in the past years to overcome the

drawbacks of traditional radar tracking methods, Track Before Detect (TBD) [115-118], which uses the whole available information from the radar, instead of thresholding. It improves tracking accuracy by allowing the tracker to follow low-SNR targets. TBD algorithms are designed to process unthresholded measurements. Despite its name, in TBD detection and tracking are done simultaneously, as seen in Figure 4.2. The steps of a classical radar tracking setup are: thresholding, clustering, extraction and tracking. The features are extracted to obtain the radar plots. Afterwards, the plots are filtered and merged to obtain trajectories in the tracker.

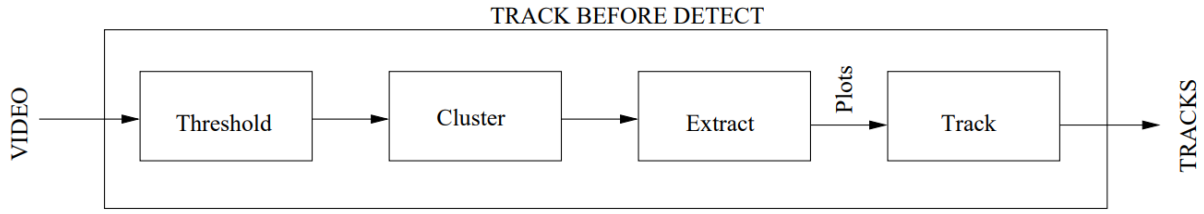


Figure 4.2: Traditional radar tracking method based on thresholded data and the TBD method, where tracking and detection are performed simultaneously [115].

The drawbacks of TBD are that it is more computationally intensive, since the detection stage is incorporated in the tracker and it also requires a longer observation time before the track initiation is declared.

TBD is a highly non-linear problem and cannot be solved (without approximations) using KF, which can make predictions only in linear, Gaussian systems. The non-linear space must be discretised using techniques such as Hidden Markov Model (HMMs), Viterbi, Particle Filters or the Histogram Probabilistic Multihypothesis Tracker, as seen in [119]. An overview of these TBD methods is shown in Figure 4.3.

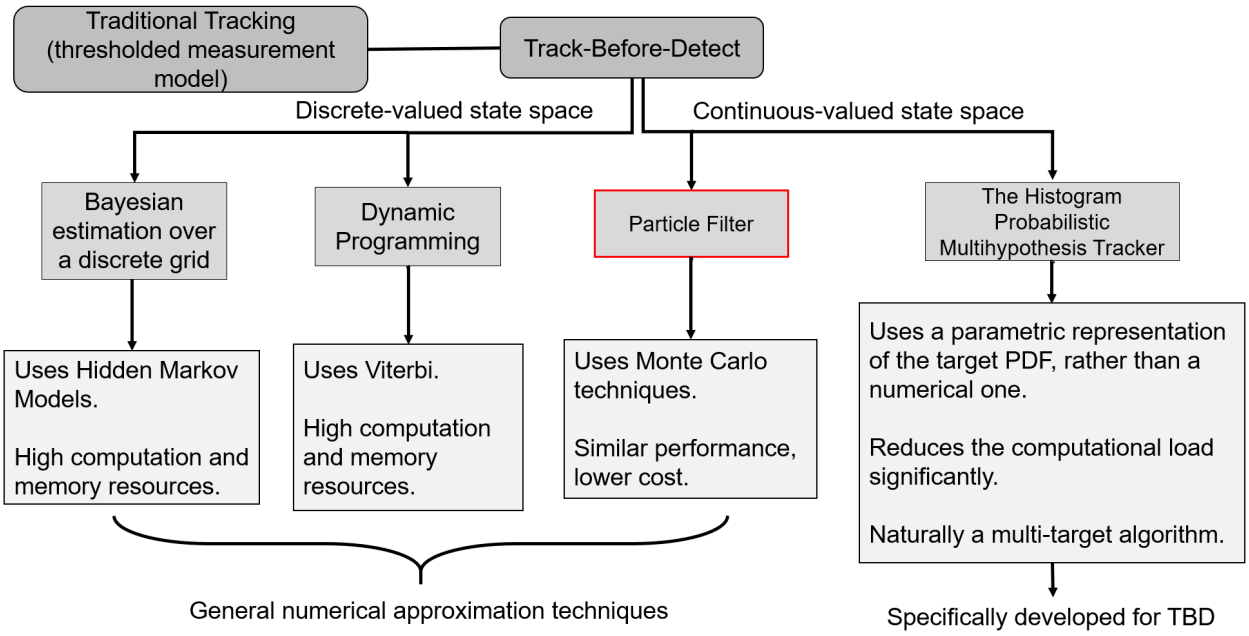


Figure 4.3: An overview of popular Track-Before-Detect Methods.

4.2 Particle Filters

Particle Filters (PFs) (or Sequential Monte Carlo [120]) are a class of TBD algorithms and were firstly introduced in 1996 by Del Moral [121] as a nonlinear filtering technique which emerged from genetic algorithms. Since then, PFs have been widely used to solve Hidden Markov Models (HMMs) and non-linear filtering problems, by using Bayesian statistical inference and stochastic sampling for estimating the internal state of dynamic systems, when partial observations and random perturbations are present. PFs use a non-parametric approach and model arbitrary distributions by using samples (or particles) with associated importance weight. The more samples it uses, the better is the estimate, although too many particles can increase the system load significantly.

4.2.1 Working Principles

PFs have a well established methodology to generate samples from the required distribution, so there is no need to make assumptions about the distribution of states or the state space model. In other words, PFs can be beneficial in estimating the state of a system by taking advantage of the available measurements.

For a better understanding, consider the following analogy. Commonly, a PF technique can handle a situation such as:

1. There is something that we want to know, supposedly called X.
2. It is possible to measure some parameter(s) related to X.
3. There is a known relationship between the measurements and X.

The basic idea of a PF is to use multiple samples to represent arbitrary distributions. Unlike Kalman filters and its variants, PFs can model non-Gaussian distributions. PFs consist of roughly five steps, detailed below. First, the particles are sampled using the proposal distribution and the importance weights are computed. During the resampling step, the unlikely samples are replaced by the most likely ones, also called "survival of the fittest". This step is crucial, as the number of particles is limited and it is preferable to avoid situations in which many samples cover unlikely states. Particles are then propagated following the motion model and they are weighted according to the likelihood of the observation. The typical PF processing steps are:

1. ***Initialisation:*** compute a transition probability for the spatial distribution of particles $p(x_0)$ using a proposal distribution (for example a Gaussian). Generate a set of N particles $\{x_0^i\}_{i=1}^N$ from this initial distribution and set the time step $t = 1$.

   During this step, a set of particles is randomly generated. Particles can have position, velocity or any other state variable that needs to be estimated. Each particle has an associated weight, which is the probability of the particle to match the state of the system. The particles are initialised with the same weight.

2. ***Prediction***: draw the predicted samples $x_t^i \sim p(x_t|x_{t-1}^i)$, $i = 1, ..., N$;

   In this step, the next state of the particles is predicted. The particles are moved based on the prediction on how the real system is behaving.

3. ***Update:*** also called Sequential Importance Sampling - make a set of observations y_t and compute the importance weights of each sample $w_t^i = w_{t-1}^i p(y_t|x_t^i)$ to account for the differences between the measurement and the proposal;

   Particles that closely match the measurements are weighted higher than other particles which do not match the measurements that well.

4. ***Normalisation:*** normalise the importance weights $w_t^i = \frac{w_t^i}{\sum_{n=1}^{N} w_t^n}, n = 1, ..., N$;

5. ***Resampling:*** replace unlikely samples by more likely ones and generate a new sample set $\{x_t^j\}_{j=1}^n$;

   Set the next time step $t = t + 1$ and repeat from step 2.

   Optionally, a state estimate can be obtained by computing the weighted mean and the covariance of the particles.

Figure 4.4 shows a visual representation of the algorithm, based on the description from [122]. The black curve represents the distribution of points of an initial measurement at time t, while the next iterative steps allow resampling, updating weights and making predictions.

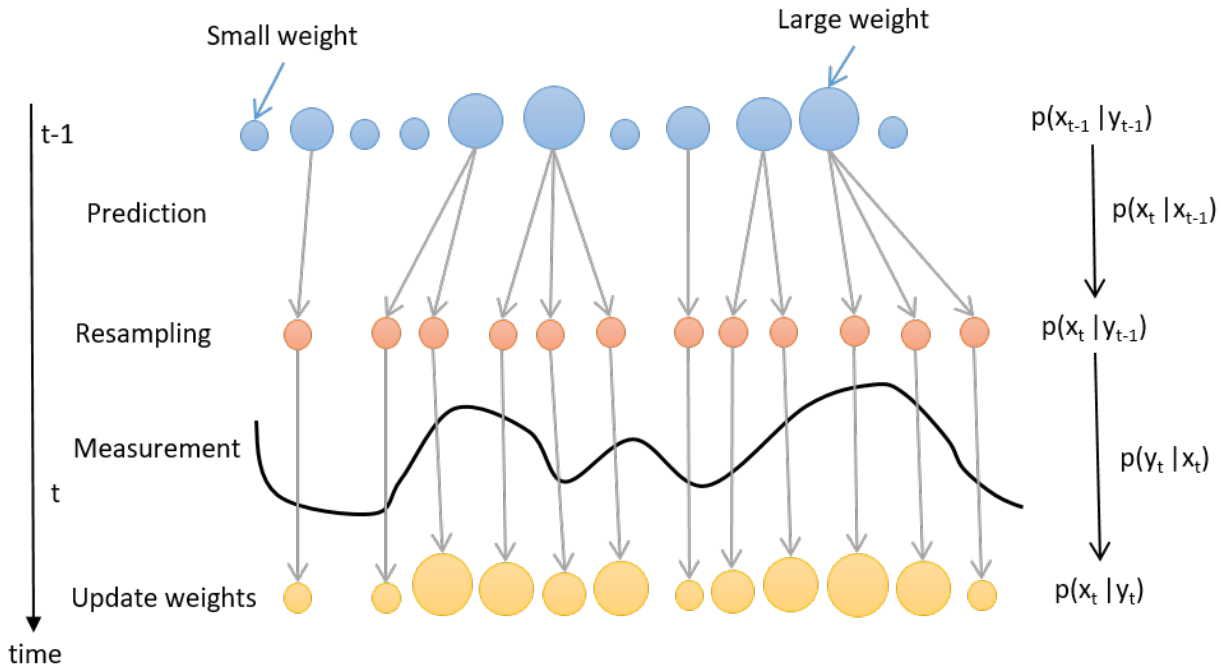


Figure 4.4: Principle of particle filters.

4.2.2 Common Problems

Degeneracy Problem

One of the most troublesome situations that can arise with PFs is the unlucky event when only a small number of the particles can contribute to the state estimate, while the other particles can have very low weights. The SIS algorithm suffers from this problem, also known as the degeneracy problem. Suppose a PF was used to track an object and during initialisation there might be little to no particles in the proximity of the target. As the algorithm runs, the other particles which do not match the observations will acquire extremely low weights. The only particles contributing to the state estimate will be the very few ones which are near the target and have appreciable weights. If this situation arises, we say that the filter has degenerated. Fortunately, the crucial step of resampling aims to solve this issue by duplicating particles with high probability and discarding the ones with negligible weights.

Sample Impoverishment

However, the degeneracy problem can convert into impoverishment, as a direct consequence of the resampling step [123]. Sample impoverishment refers to the situation in which particles are over concentrated on a target. The most straightforward method to avoid sample impoverishment is to add a zero-mean noise to the state of each particle after resampling, to improve particles' diversity. This method is also known as "roughening". This approach is very useful, although we have to be careful with adding artificial dynamics to the particles, because this can cause issues such as over-dispersed posteriors, also known as the variance inflation problem.

4.2.3 Motivation

The motivation behind choosing the PF approach is supported by these particular characteristics required by the application:

1. Multimodality: can track more than one target simultaneously;
2. Multivariation: can track several attributes simultaneously, such as position, velocity, etc;
3. Continuity: the state space, or the target's attributes can vary over time;
4. Non-linear behaviour and measurements: In radar measurements, the non-linearity comes from the measurements' coordinates. When converting the distance to an object from polar coordinates to a linear system of coordinates, it requires square root and other non-linear trigonometry operations;
5. Can handle non-Gaussian noise: assume we have a signal influenced by clutter, which is most probably not a Gaussian noise;
6. Can handle occlusions (both partial and full occlusions, to some extent): one target can hide another, resulting in one observation for multiple targets;
7. Can handle extended targets in low-SNR;
8. Can handle an unknown process model: the process model of the system is unknown.

PFs can address the above constraints, unlike some of the other well known filter alternatives:

1. Kalman Filter [124]: this method relies on linearity and normality assumptions, it produces optimal estimates for linear systems with Gaussian noise [125], which is not

practical for this application. KF is suitable for LiDAR applications, since the measurements are linear. The Gaussian error estimations in the KF remain Gaussian after the *Update* and *Prediction* steps, assuming they are linear functions. Moreover, KF estimates the position and velocity (change in position) of a radar target and then iteratively updates the position of the target. The position is estimated using the maximum in the radar intensity image. Assuming this is an unbiased estimator in additive Gaussian white noise, the best this method could do is dictated by the Cramér-Rao bound, which states that there is a large error in this position estimate, because of the low-SNR.

On the other hand, PFs can reproduce the work of a KF in non-linear environments and cluster a set of particles around the whole target, whereas the KF would find a bright point and analyse how it moves. An extended target would be a collection of nearby bright points and a PF would find the whole shape and analyse its movement, therefore PF is the optimal choice for tracking an extended target, where individuals pixels are fluctuating and the noise realisation changes each with measurement.

2. Unscented Kalman Filter [126]: this method can handle non-linear problems, basically by linearising the equations using approximations. On the other hand, a PF uses simulations, which are more accurate than approximations. Moreover, UKF is unimodal and it cannot handle occlusions. It can modestly undertake non-Gaussian noise, though it does not perform very well with distributions that are very non-Gaussian. UKF also assumes that the filtering distribution of previous steps is Gaussian, therefore it is not a truly global estimator.

3. Extended Kalman Filter [127]: has the same strengths and limitations as UKF, although it is even more sensitive to strong non-linearities and non Gaussian noise. The non-linear functions are approximated using the first derivative of Taylor series, also called the Jacobian Matrix. EKF is more computationally intensive than UKF, be-

cause it calculates Jacobians at each step. UKF approximates around multiple weighted sigma points, while EKF approximates around the mean.

4. Discrete Bayes Filter [128]: this filter has some of the attributes required by our application, such as multimodality and non-linearity. However, it is discrete and univariate (cannot track multiple targets simultaneously).

Other alternatives are the Gaussian sum approximations or grid-based filters, although they are of limited use and many of the above mentioned methods, including EKF, fail asymptotically. Therefore, the PF is the most adequate, accurate and robust filtering technique that can tackle inference for non-linear systems in this application. The drawback of PF techniques is that they can be computationally more expensive than other filters.

4.2.4 Multi-target Particle Filter

The performance assessment for a Multi Target Tracking (MTT) filter is done in terms of reliability, accuracy and computational efficiency. Some of the challenges of MTT are that the number of objects of interest in the frame is unknown and varying over time. The filter will need to account for each target appearance and disappearance from the field of view, as well as for occlusions and targets merging or splitting. A MTT filter will also need to take into consideration false alarms, missing reports, and unresolved targets.

There are two main methods of implementing PFs for tracking multiple targets simultaneously [129]:

1. Method 1 - use multiple instances of single object tracking techniques, thus using multiple PFs for tracking each target individually [130-131].

2. Method 2 - dynamically change the dimension of the state-space to include components for all targets, or add a set of indicator parameters, signifying whether an object is present or not [132-133].

In this project, the first method is used for implementing the multi-target tracker, by allocating a PF for each object of interest. To prevent the PFs from tracking the same target, the particles of different filters are limited from getting near each other by restricting a PF to track a target that it is already being tracked by another PF and also by randomly scattering the particles if the means of the particles of the two PFs are too close. This approach is explained in more detail in the next section. A second approach was to decrease the weights of particles if they come in proximity with particles from another filter. However, this approach was found to be very computational expensive and also interfering with the update step of PFs.

4.2.5 Particle Filter Estimators

Mean

Each particle has a position parameter, $pos(x, y)$. The mean of the particles of a PF is calculated as follows:

$$\text{mean} = \frac{1}{N}\sum_{i=1}^{N}(x_i, y_i) = (\mu_x, \mu_y), \tag{4.1}$$

where N is the total number of particles.

Standard Distance Deviation

Standard deviation is not a single summary statistic, it is actually two separate statistics in the linear system of coordinates (x,y). Therefore, we can use the 2D equivalent, also known as Standard Distance Deviation, which is the standard deviation of each point from the mean centre:

$$S_{xy} = \sqrt{\frac{1}{N-2}\sum_{i=1}^{N} d_{iMC}^2}, \tag{4.2}$$

where d_{iMC} is the distance between particle i and the mean centre (μ_x, μ_y):

$$d_{iMC} = L_2 \text{ Norm} = \text{Euclidean distance} = \sqrt{(x_i - \mu_x)^2 + (y_i - \mu_y)^2}. \tag{4.3}$$

Note that for S_{xy}, 2 is subtracted from N in order to produce an unbiased estimate of standar distance devition, since this distance is measured from two constants, μ_x and μ_y.

Variance

Variance is defined as the square of the standard distance deviation:

$$\text{variance} = S_{xy}^2 = \frac{1}{N-2}\sum_{i=1}^{N} d_{iMC}^2. \tag{4.4}$$

For this application, it was found that the PF tracks a target when the variance is lower than ≈ 300.

4.3 Proposed Method

4.3.1 Description

The combined object detector and tracker with Faster R-CNN and particle filters is implemented in Python, using Keras under TensorFlow backend and has the following functionalities:

1. MTT is implemented by using multiple instances of PFs which can track different targets. A fixed number of PFs are initialised in the beginning, although the functionality allows for more PFs to be dynamically created when a new target is detected by the object detector, making this process an automatic detection and tracking process.

2. Particles are re-scattered if they come in proximity with particles from another filter which is already tracking a target. This prevents multiple filters from tracking the same target.

3. The object detector can analyse all frames, or omit a certain number of frames set by the user. If targets are found in the frames, the object detector aids the PF to track the target by placing the particles of an unallocated PF in the bounding box (if the target was not being tracked by a PF already). The PF on its own is set to find targets based on their intensity maps. In this case we consider intensity map as an RGB image and detection is triggered in pixels where specific values are found. Therefore, though the input from the object detector is not compulsory, it can guide the particles to find

the target faster. PFs will always perform worse on their own than when combined with an object detector, because of the additional time needed by the particles to find targets.

Therefore, in order to reduce the computational load, object detection does not need to be performed in each frame, as the PFs will assist it by keeping track of the previously detected targets. At mm-wave frequencies and in automotive scenarios, targets are slowly fluctuating, hence we can expect targets to have the same order of magnitude within few seconds. Finally, the object detector and the PFs assist each other and mutually improve the functionality of the overall system.

4.3.2 Implementation

Object Oriented Programming (OOP)

The proposed method is implemented in the Python programming language, using an OOP approach. OOP is a computer programming technique that organises the code around data (objects), rather than the traditional programming approach based on functions and logic. An object belongs to a type of class and has attributes (the state of the object) and behaviours (methods, functions). The concept is similar to traditional structures, although OOP is a lot more complex and works well for large programs that require regular updates and maintenance. Other benefits of OOP that make it suitable for this application are its modularity structure, code reusability and scalability. Therefore, OOP is the preferred programming technique when combining two or more standalone applications, such as an object detector and a PF tracker, because it is simple and straightforward to modify or replace individual parts of the code, without affecting the rest of it.

The UML Class diagram for this program is illustrated in Figure 4.6 below. The class

diagram shows the structure of the OOP code: the classes, the attributes and the methods for each class and the relationships between classes.

Class Diagram

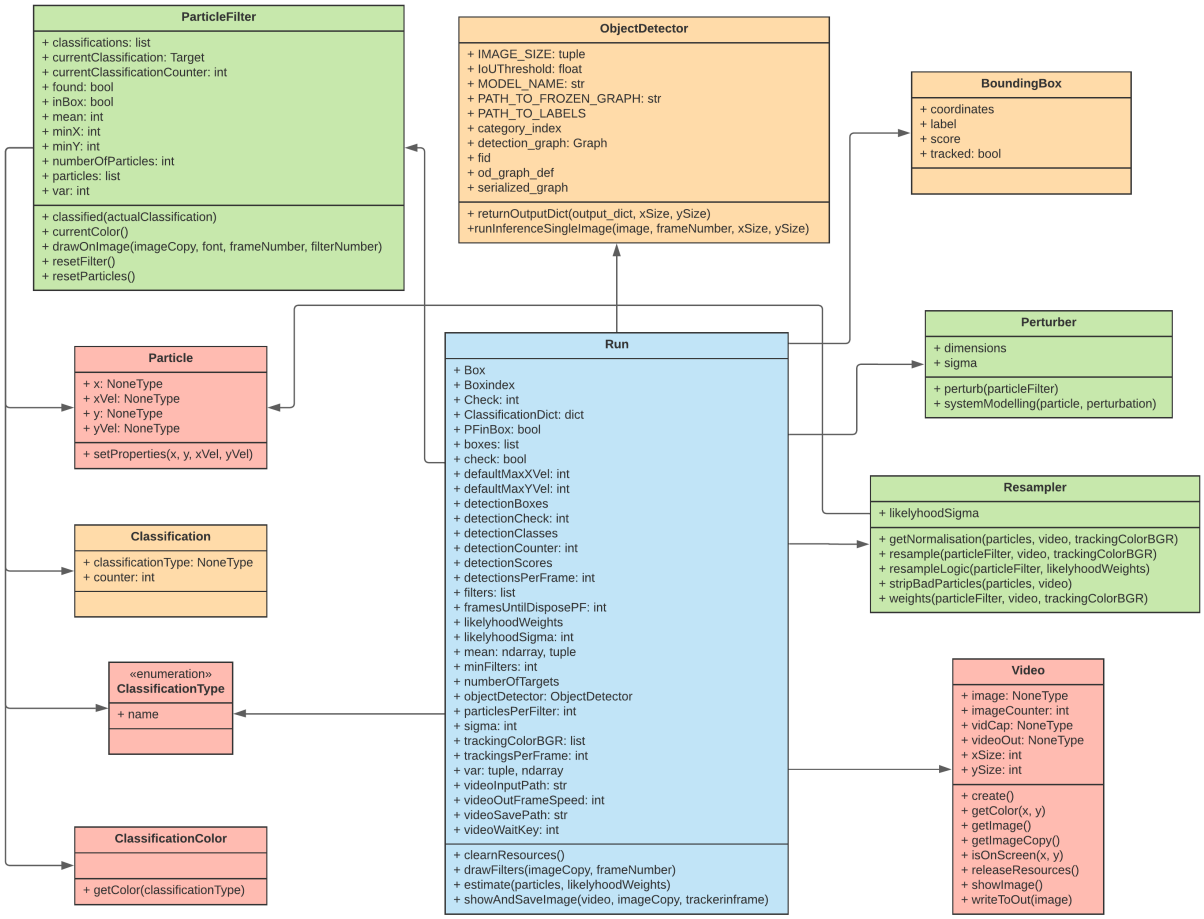


Figure 4.6: Class diagram for the implementation of the proposed method of combining an object detector with a PF tracker.

Class Description

1. `Particle`: this is a class which has a setter method, `setProperties(x, y, xVel, yVel)` for setting the properties of the particles, $position(x, y)$ and $velocity(xVel, yVel)$. The method is called in the class constructor for each particle that is instantiated in the `Run` class.

2. `ClassificationType`: this is an enumeration type of class, which specifies the types of targets and their associated number (i.e Car = 1, Cyclist = 2, Lamppost = 3, Pedestrian = 4, Sign = 5, Target = None).

3. `Classification`: this class has a constructor which assigns the actual `classificationType` to the PF.

4. `ClassificationColor`: this class has a getter method, `getColor(classificationType)` which returns the colour for the particles specified in a `ColourDict`, depending on the target number. This is only for visualisation purposes.

5. `Video`: this class manages the radar video through its methods. It extracts the frames, shows them on screen and merges the processed frames to obtain the output video.

6. `BoundingBox`: this class assigns the properties of the bounding box through its constructor, the bounding box coordinates, the label from the object detector, the detection score and a boolean parameter `tracked` to indicate whether the bounding box also contains a PF which tracks the target or not. This helps implementing MTT.

7. `ObjectDetector`: this is the class which runs the object detector in each frame (in method `runInferenceSingleImage`) and returns the output dictionary (in method `returnOutputDict`), which contains the number of targets found, their bounding boxes coordinates, the classification scores and the labels.

8. `ParticleFilter`: this class initialises the PF in the constructor, specifying the `classificationType`, the number of particles and resetting the particles to their initial positions and velocities. It contains a `resetFilter` method for resetting the classifications and a `resetParticles` method which randomly scatters the particles in the frame and assigns random, uniform distributions for their velocities. The `drawOnImage` method draws the particles and the centre mean on the image, at the specified locations. Aditionally, a variance bar can be included. This method also writes on the image when the PF finds a target.

9. `Resampler`: this class implements the resampling step of the PF and contains the resampling algorithm (`resampleLogic`). It has methods for removing the unlikely particles (`stripBadParticles`), for calculating the weights based on the observation and for normalising them (`getNormalisation` and `weights`).

10. `Perturber`: this class adds a random noise to the state of the particles, to avoid the sample impoverishment issue.

11. `Run`: this is the class where the main function is executed, which runs the application. It contains all the input/output paths and the parameters' values, such as the number of trackings per frame, the number of object detections per frame, the numbers of particles, etc. It has several methods to calculate the mean and the variance of the particles.

Programming algorithm

The proposed algorithm is generally implemented as follows (in the `main` method of the `Run` class):

1. Initially, apply a number of PFs on the frame and run them for a number times. If

targets are found, based on intensity levels, label them as undefined, generic "Target";

2. Send the frame to the object detector for processing. Generate labels, bounding box coordinates and classification scores for each target;

3. Check if a PF is already in a bounding box (shortly BBox), tracking the object. If so, assign the target's label to the PF (change the colour of the particles according to the `ColourDict` in `ClassificationColor`) and display the label. Otherwise, release a new PF with all its particles scattered in the bounding box;

4. Discard the unnecessary PFs, and keep a couple extra in the frame, in case the object detector misses a new target;

5. Keep tracking the undefined targets that were not classified;

6. If a target disappears from the frame and the PF is not tracking any object for a number of consecutive frames, discard the PF;

7. Implement MTT by preventing a PF to enter a bounding box if another PF is already tracking that target. Randomly scatter the particles if the means are too close;

8. Get the estimators for the PFs (mean, variance).

Repeat steps 2-8 above for each frame of the radar video that requires both object detection and tracking. More details about how and when object detection and tracking are performed/skipped are found in the Results Section 4.3.4.

For a better understanding, the above algorithm is briefly illustrated in the flowchart diagram below, in Figure 4.7.

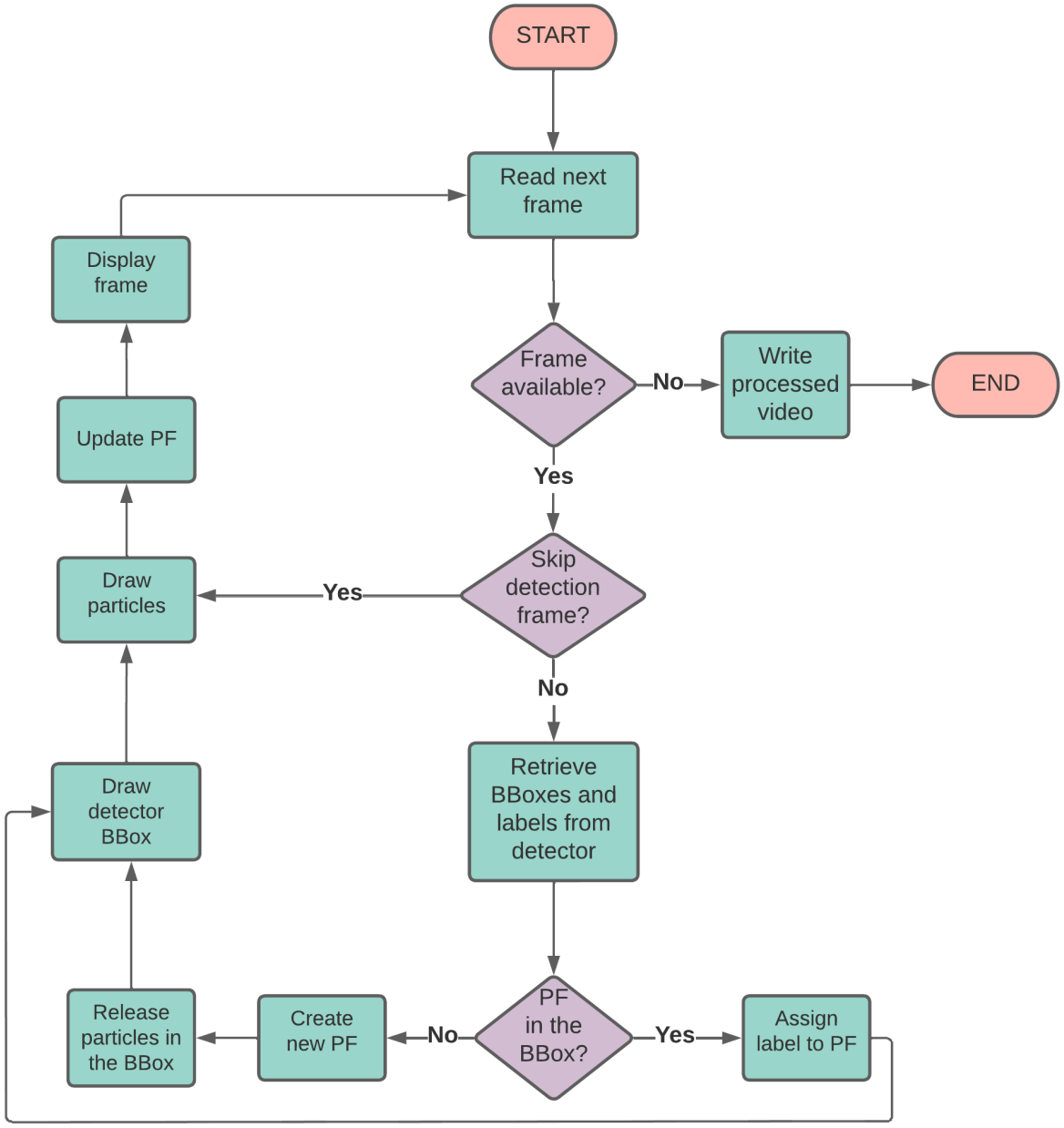


Figure 4.7: Flowchart diagram for the proposed algorithm.

4.3.3 Experimental Setup and Dataset

The object detector was trained on a dataset of scanned scene maps in outdoor environment, obtained with a 79 GHz wideband experimental LFMCW radar, custom-made by ELVA-1 [106] by the specification of The University of Birmingham. The specifications of this dataset and the radar parameters for the data acquisition are found in Section 3.3.1 Experimental Setup and Datasets - Outdoor Dataset.

The same five classes of roadside objects were considered for this application: cars, cyclists, lampposts, pedestrians and traffic signs. The targets were manually labelled with rectangular boxes in each frame, using the accompanying optical images from the ZED stereo camera for guidance and an open-source Python graphical image annotation tool [107], as described in Section 3.3.1 Experimental Setup and Datasets - Annotations.

The performance of the object detector on the testing data can be found in Section 3.3.3 Object Detection Results. The experiment concluded that Faster R-CNN performed better than SSD on the radar imagery dataset. Although the fully-convolutional approach of SSD increases the computing speed, it performed much worse at detecting small objects, therefore SSD is preferred in applications where speed is a priority. For this reason, Faster R-CNN is considered as a feasible object detection method for this application. More details about the image processing, the anchor boxes generation and the detection steps can be found in Section 3.2.2 Faster R-CNN.

4.3.4 Results

This section shows a detailed analysis of the parameters which influence the performance of the combined tracking and detection system. Usually, defining the performance metrics of a PF can be difficult. An accurate PF typically requires a large number of particles, and analysing how the tracking accuracy depends on the number of particles may be difficult, especially for MTT where targets are extended and have different sizes. Here, the accuracy of the object detector is regarded as the percentage of the targets detected by the object detector (included in bounding boxes and classified in the correct class), while the combined system accuracy is regarded as the percentage of targets detected and/or tracked by the particle filters, out of the total number of targets in the test set. PFs are also non deterministic and the exact results cannot be predicted, different outputs can be achieved with the same input. Therefore, in this thesis, the performances of the joint object detector and PF were assessed by varying the values of different parameters which influence the accuracy.

The proposed approach has been applied on a radar video with moving pedestrians. The radar video here represents a sequential set of radar maps or frames, which were collected with the 79 GHz FMCW radar described previously. Figure 4.8 shows a sequence of six frames (frames 27-32) and their corresponding optical frames from the ground truth video, time stamped with the radar data. The output from the object detector is represented by the bounding boxes with associated labels and confidence scores. The PFs are pictured by pink or yellow particles and the mean value of the positions of particles for each filter is represented by a cross. When PFs are tracking targets which are also detected by the object detector, the labels are transferred and the particles change their colours to yellow. Initially, object detection was performed in every frame to assess how the PF can assist the object detector when it fails to detect targets.

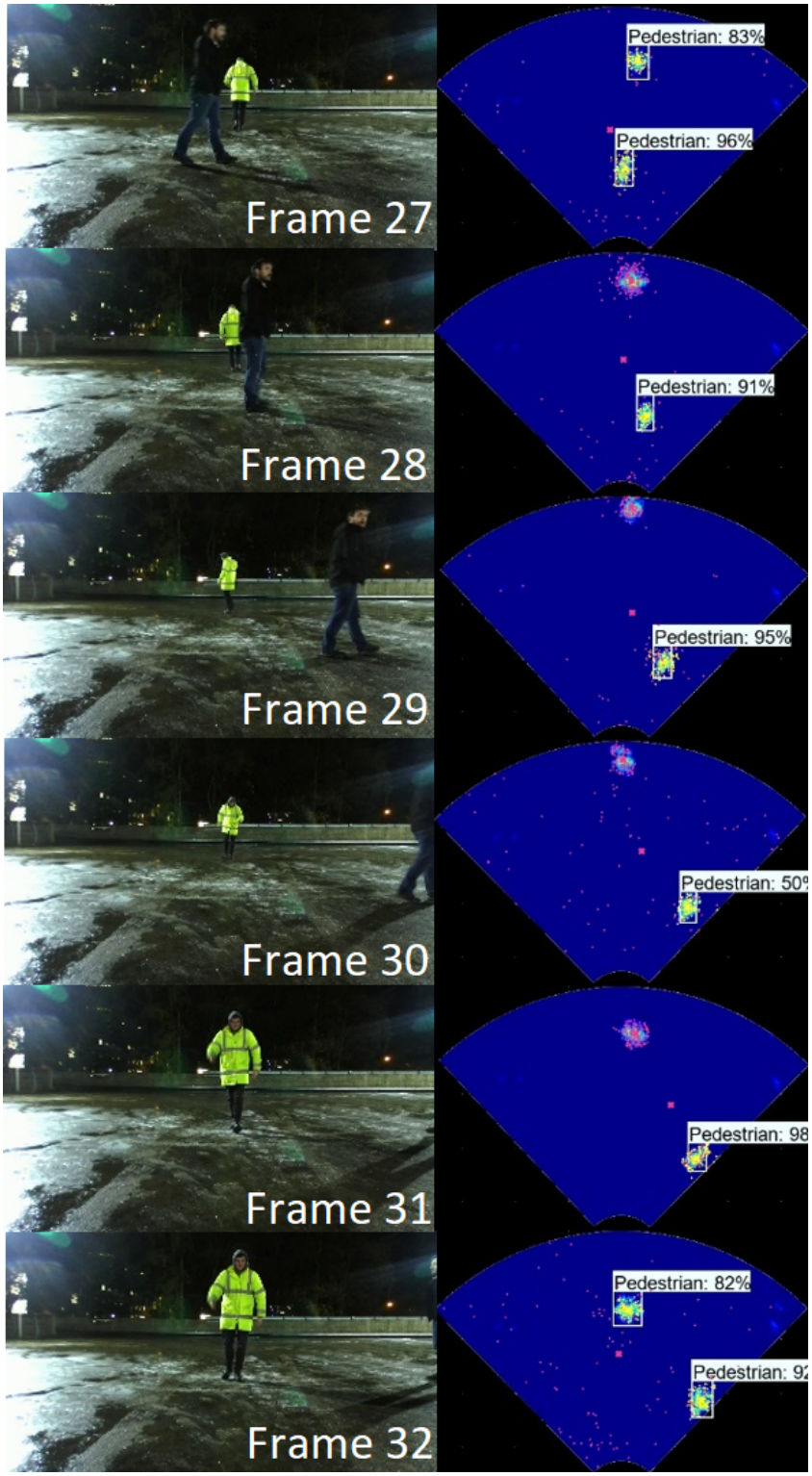


Figure 4.8: Object detection and MTT with three PFs in a sequence of six consecutive frames of moving pedestrians. The PFs keep track of both targets, even though the object detector fails to detect one of the pedestrians in four consecutive frames (28-31).

In this particular scenario, there are three different PFs initialised, each one represented by the mean value of the positions of their particles. The object detector fails to detect one of the pedestrians for four consecutive frames (28-31), although the PF still keeps track of both pedestrians. The third PF remains unallocated, since there are only two targets in each image.

1) Varying the frequency of the object detector

Object detection can be performed every N^{th} frame of the sequence, where N is chosen by the user and $N-1$ represents the number of omitted frames for object detection. Figure 4.9 shows how the number of the targets tracked and detected depend on the number of omitted frames (up to 9)– the frames not used for object detection. The results show that a suitable trade off between performance and speed would be to perform object detection once every two or three frames to achieve tracking accuracy for the combined system of $\approx 90\%$ of the total number of targets in the video. This analysis was performed with 200 particles per PF and 10 tracking runs per frame.

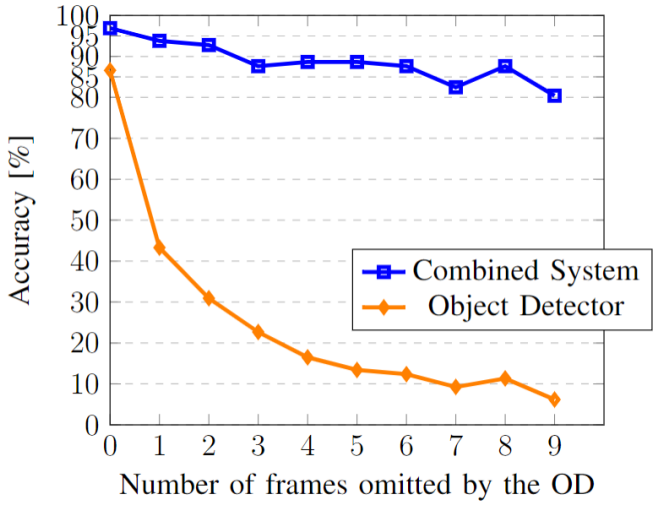


Figure 4.9: Combined system accuracy and object detection accuracy as a function of the number of frames omitted from the object detector.

2) Varying the number of trackings per frame

In order to improve the tracking accuracy, the tracking process can be performed multiple times per frame, although this increases the computational load. An analysis can be found in Figure 4.10, where the tracking runs per frame were varied from 1 to 20 and the frames omitted by the object detector were varied from 0 to 9. It can be seen that the accuracy increases with the number of trackings per frame and a suitable value is achieved with around 15 trackings per frame. Also, each curve represents the number of frames omitted by the object detector. This analysis was performed with 200 particles per filter.

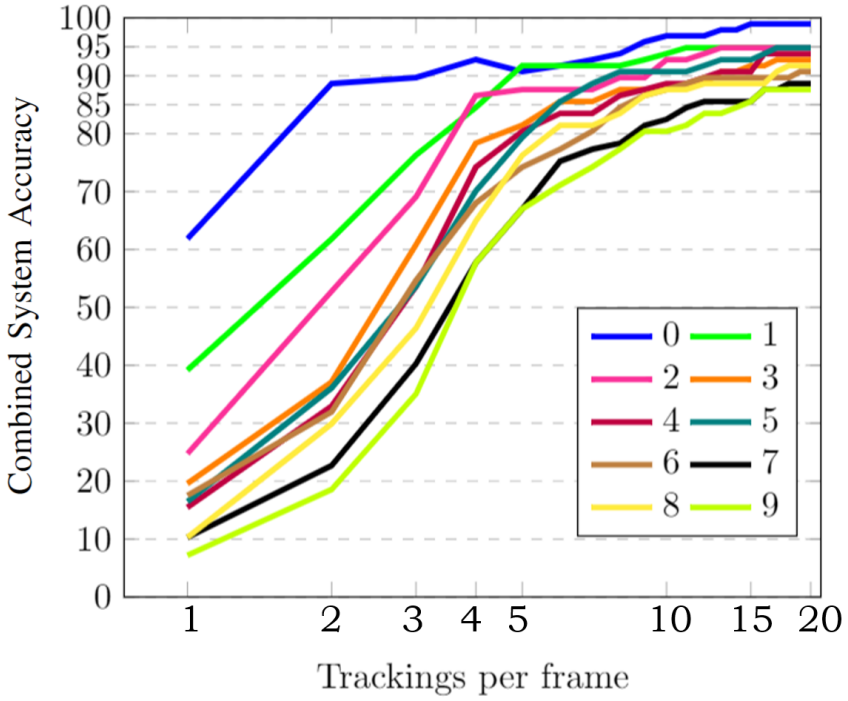


Figure 4.10: Combined system accuracies for different number of frames omitted by the object detector vs. the number of trackings per frame. The legend represents the number of frames omitted by the object detector.

3) Varying the number of particles

Another parameter which highly influences the tracking performances is the number of particles in each PF. This parameter should be carefully chosen, because a large number of particles can increase the computation time and resources significantly. For MTT, this problem is even more difficult to solve because targets can have different sizes and require a different number of particles.

The analysis of a varying number of particles is shown in Figure 4.11, for the same radar dataset with pedestrians moving within a range of ≈ 15 meters. The analysis considers the worst case scenario in Figure 4.10, with object detection performed every 10 frames. It is observed that the accuracy of the tracker can be improved from 87.6% to 96%, if the number of particles increases from 200 to 450.

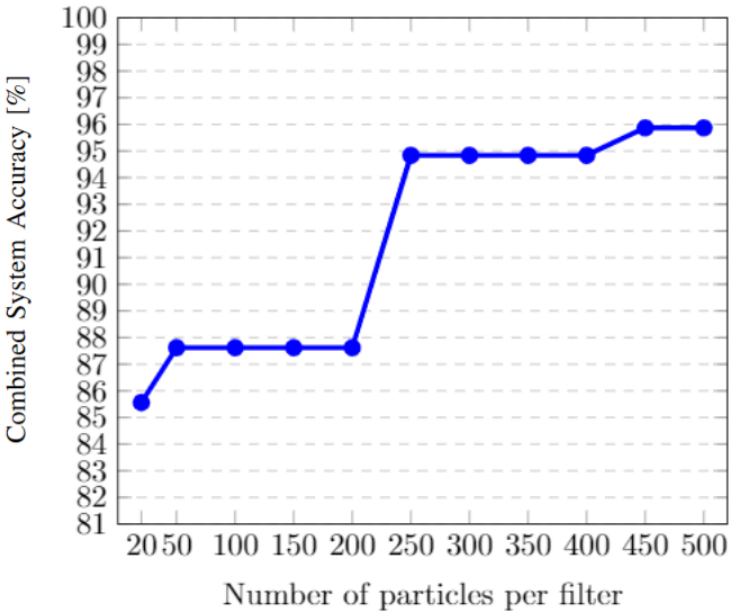


Figure 4.11: Combined system accuracy vs. the number of particles per PF.

4.3.5 Discussion

The performance of the proposed system was analysed on a radar video with moving pedestrians because the targets were considered relevant for tracking purposes. This method can be applied on other radar videos with different moving targets and achieve comparable results, providing the radar scans have low clutter, the particle filter's parameters are adjusted for tracking other target or other filter instances are created for specific target detection and the object detector is trained for detecting the target.

Analysis shows that the same results can be achieved by varying the three parameters which were analysed above: frequency of the object detector, PF trackings per frame and the number of particles. For example, same results (96% accuracy) are achieved with 9 trackings per frame, object detection performed in each frame and 200 particles per PF, as well as with 15 trackings per frame, object detection performed every 10 frames and 450 particles per PF.

Therefore, there are three alternatives for addressing this proposed method of a combined object detector and tracker. The parameters should be adjusted by the user depending on the application, the configurations of the system and the computational load supported, in order to achieve the desired performance and speed. For this specific implementation, the best trade-off between accuracy and speed was achieved by increasing the number of trackings per frame and decreasing the frequency of the object detector.

4.4 Conclusion

This chapter presented a novel radar imaging method of combining an object detector with a multi target particle filter tracker for automotive radar imagery. The object detection was performed with a deep neural network (Faster R-CNN).

The analysis of the system's performances proved that this combination is beneficial, given the fact that the tracker and the detector assist each other. The detector guides the particles to find the target, while the particle filters can still keep track of the targets when the detector fails. The combined system achieved 98.9% accuracy, as opposed to 86% object detection accuracy when the object detector is used without tracker. It is also shown that object detection is not required in each frame, therefore the computational load of the combined system can be reduced significantly.

Chapter Five

Conclusions and Further Work

5.1 Conclusion

This thesis presented an investigation of deep learning techniques for object identification in high frequency and low-THz radar imagery for autonomous vehicles. The impact of our research in this field was demonstrated for specialist audiences and it has received two awards at top radar conferences.

Chapter 1 introduced an overview of the main concepts used in this work, from radars to autonomous driving and artificial intelligence. Chapter 2 presented the first part of this work, classification of objects in low-THz radar imagery using CNNs, for six different roadside targets in controlled environment: a trolley, a dog, a bicycle, a mannequin, a traffic sign and a traffic cone.

After classification of single targets in controlled environment, the next approach towards real-time detection was to classify multiple objects in the same frame, hence the concept of object detection using deep neural networks was introduced in Chapter 3. Object detection in high frequency radar imagery was performed for both indoor and outdoor high

frequency radar imagery, using two state of the art object detectors, faster R-CNN and SSD.

Chapter 4 presented the last part of this project, an integrated system for object detection and tracking in automotive radar imagery. The novel system combined the object detector described in Chapter 3 with a particle filter tracker. The analysis of the combined system proved that a particle filter tracker can benefit from the input of an object detector and find the targets faster. Also, the object detector benefits from the input of the particle filter in reducing the computational load, since object detection is not required for each frame anymore after tracking is initiated. Nevertheless, the combined system increased the overall accuracy of both tracker and object detector.

5.2 Further Work

Another appropriate deep learning technique which could be further investigated is the instance segmentation for roadside objects in automotive radar scenes using deep neural networks. Initial evaluations were made using Mask R-CNN [134], a deep learning method for instance segmentation. However, this method requires highly accurate manual annotations of the shape of the object, which are time consuming. Early results on two outdoor radar scans can be seen in Figure 5.1.

Segmentation can potentially yield better results when combined with a particle filter tracker because the segmentation mask (the actual shape of the object) is more accurate than the bounding box for detecting objects of the same class which are in close proximity. However, this method is basically pixel level classification, hence it is more susceptible to clutter in radar images. Therefore, research is being carried out at the moment for filtering radar images using computer vision techniques, starting with the bilateral filter [135]. The initial results can be seen in Figure 5.2. Once this issue is solved, a dataset can be properly

labelled for segmentation.

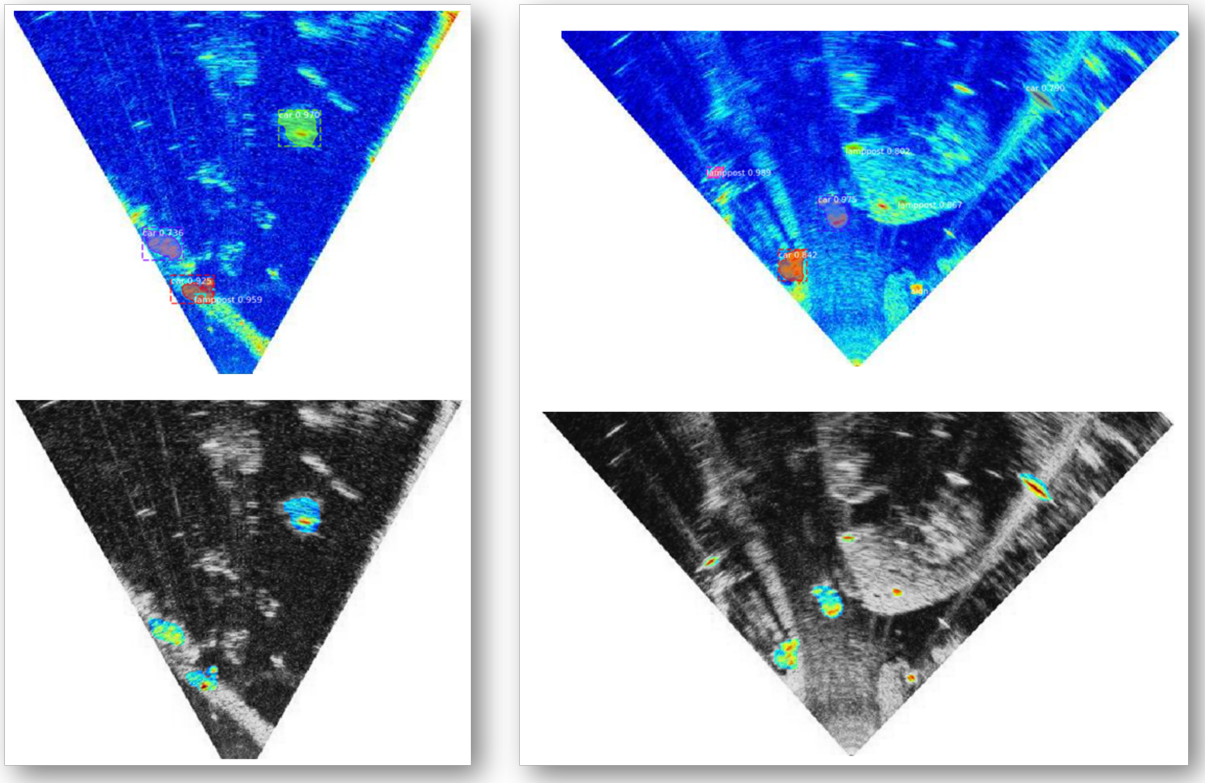


Figure 5.1: Preliminary results of instance segmentation using Mask R-CNN on two outdoor radar scans (left and right) and the highlighted versions (bottom) for better visualisation.

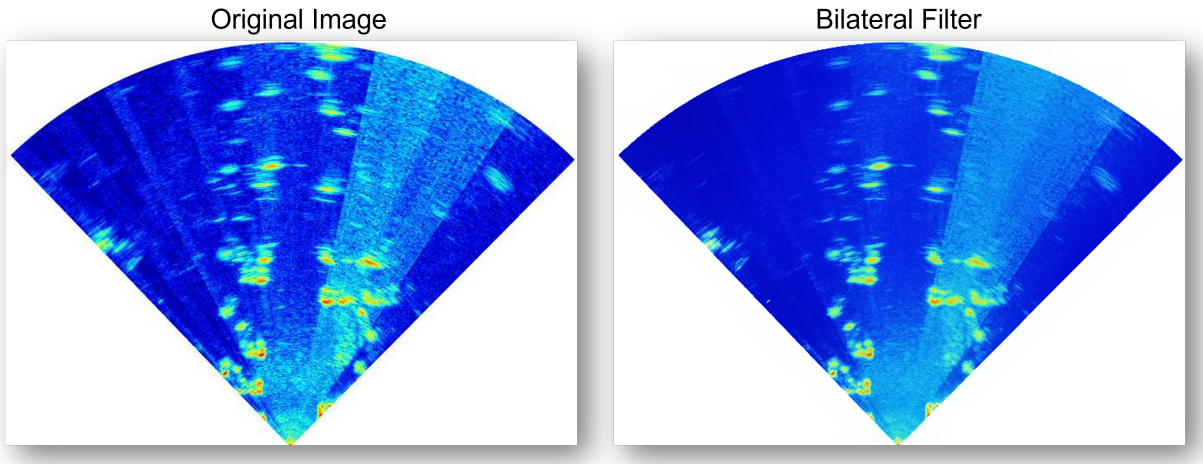


Figure 5.2: Preliminary results of radar image filtering using a bilateral filter.

Appendix One

CNN Python code

```
from __future__ import print_function
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
import os
import numpy as np

batch_size = 256 #set the batch size
num_classes = 6  #set the total number of classes
epochs = 150     #set the number of training epochs
LOGDIR="C:/Python_work/CNN_KERAS/"

save_dir = os.path.join(os.getcwd(), 'saved_models') #saving directory
model_name = 'CNN_radar.h5'
```

```
# input image dimensions
img_rows, img_cols = 220,220 #image size
no_train=1620
no_test=822
# annotations files
y_train=np.load("y_train_300GHz_3receivers_220x220.npy")
y_test=np.load("y_test_300GHz_3receivers_220x220.npy")

# the data, shuffled and split between train and test sets
x_train=np.load("x_train_300GHz_3receivers_220x220_COLOR.npy")
x_test=np.load("x_test_300GHz_3receivers_220x220_COLOR.npy")



if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
```

```
print('x_test shape:', x_test.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

#define the CNN architecture
model = Sequential()
# 1st CONV layer
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(img_rows,img_cols,3)))
#MaxPool
model.add(MaxPooling2D(pool_size=(2, 2)))
# 2nd CONV Layer
model.add(Conv2D(64, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(img_rows,img_cols,3)))
#MaxPool
model.add(MaxPooling2D(pool_size=(2, 2)))
# 3rd CONV Layer
model.add(Conv2D(128, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(img_rows,img_cols,3)))
#MaxPool
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))
# 4th CONV Layer
model.add(Conv2D(256, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(img_rows,img_cols,3)))
#MaxPool
model.add(MaxPooling2D(pool_size=(2, 2)))
# 5th CONV Layer
model.add(Conv2D(512, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(28,28,1)))
#MaxPool
model.add(MaxPooling2D(pool_size=(2, 2)))
# Dropout 20% of the neurons
model.add(Dropout(.2))
# Flatten the input
model.add(Flatten())
# FC layer
model.add(Dense(num_classes, activation='softmax'))
# Compile the model, loss function and optimisation algorithm
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
# Train the model
model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
```

```
          verbose=1,
          validation_data=(x_test, y_test))
# Save model
if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
print('Saved trained model at %s ' % model_path)
# Compute the training and testing accuracies
score = model.evaluate(x_test, y_test, verbose=0)
pred = model.predict_classes(x_test, verbose=2)

print('Test loss:', score[0])
print('Test accuracy:', score[1])
print('Prediction:',pred)
```

Appendix Two

Siamese Python code

```
from __future__ import absolute_import
from __future__ import print_function
import numpy as np
from keras.layers import Dense, Dropout, Input, Lambda, Conv2D,
MaxPooling2D, Flatten
np.random.seed(1337)  # for reproducibility
import random
from keras.models import Sequential, Model
from keras.optimizers import RMSprop
from keras import backend as K
import numpy.random as rng

def W_init(shape,name=None):  #initialise weights
    values = rng.normal(loc=0,scale=1e-2,size=shape)
    return K.variable(values,name=name)
def b_init(shape,name=None):  #initialise bias
    values=rng.normal(loc=0.5,scale=1e-2,size=shape)
```

```
    return K.variable(values,name=name)
#Euclidean distance between outputs
def euclidean_distance(vects):
    x, y = vects
    return K.sqrt(K.sum(K.square(x - y), axis=1, keepdims=True))


def eucl_dist_output_shape(shapes):
    shape1, shape2 = shapes
    return (shape1[0], 1)
#contrastive loss function
def contrastive_loss(y_true, y_pred):
    margin = 1
    return K.mean(y_true * K.square(y_pred) + (1 - y_true) *
                  K.square(K.maximum(margin - y_pred, 0)))
# Create positive and negative pairs
def create_pairs(x, digit_indices):
    pairs = []
    labels = []
    n = min([len(digit_indices[d]) for d in range(6)]) - 1
    for d in range(6):
        for i in range(n):
            z1_s, z2_s = digit_indices[d][i], digit_indices[d][i + 1]
            pairs += [[x[z1_s], x[z2_s]]]
            inc = random.randrange(1, 6)
            dn = (d + inc) % 6
            z1_d, z2_d = digit_indices[d][i], digit_indices[dn][i]
            pairs += [[x[z1_d], x[z2_d]]]
```

```
            labels += [1, 0]
            ##Write pairs
            if (train==1):
             print("Same: ",y_train[z1_s] ,y_train[z2_s])
             print("Different: ",y_train[z1_d] ,y_train[z2_d])
            else:
             print("Same: ",y_test[z1_s] ,y_test[z2_s])
             print("Different: ",y_test[z1_d] ,y_test[z2_d])
    return np.array(pairs), np.array(labels) ##tr_pairs, tr_y

# Create network architecture
def create_base_network(input_dim):
    model = Sequential()
# 1st CONV layer
    model.add(Conv2D(32, kernel_size=(3, 3),
                activation='relu',
                input_shape=(img_rows,img_cols,1)))
# Max Pool
    model.add(MaxPooling2D(pool_size=(2, 2)))
# 2nd CONV layer
    model.add(Conv2D(64, kernel_size=(3, 3),
                activation='relu',
                input_shape=(img_rows,img_cols,1)))
# Max Pool
    model.add(MaxPooling2D(pool_size=(2, 2)))  # downsample
# 3rd CONV layer
    model.add(Conv2D(128, kernel_size=(3, 3),
```

```
                activation='relu',
                input_shape=(img_rows,img_cols,1)))
# Max Pool
    model.add(MaxPooling2D(pool_size=(2, 2)))
#Dropout 20% of the activations
    model.add(Dropout(.2))
# Flatten the output
    model.add(Flatten())
# FC layer
    model.add(Dense(300,activation='relu'))
# Dropout 10%
    model.add(Dropout(.1))
# FC layer
    model.add(Dense(100,activation='relu'))
    return model


#Hyperparameters
nb_epoch = 100 #number of epochs
img_rows =220
img_cols = 220
no_train=1620
no_test=822
train=0
classes=6
def compute_accuracy(predictions, labels):

    return labels[predictions.ravel() < 0.5].mean()
```

```
#Load train and test labels
y_train=np.load("y_train_300GHz_3receivers_220x220.npy")
y_test=np.load("y_test_300GHz_3receivers_220x220.npy")

#Load train and test dataset
x_train=np.load("x_train_300GHz_3receivers_220x220.npy")
x_test=np.load("x_test_300GHz_3receivers_220x220.npy")

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
input_dim =(img_rows,img_cols,1)



inp_shape = x_train.shape[1:]
# create training+test positive and negative pairs
digit_indices_tr = [np.where(y_train == i)[0] for i in range(classes)]
print("Training pairs")
train=1
tr_pairs, tr_y = create_pairs(x_train, digit_indices_tr)
train=0
digit_indices_te = [np.where(y_test == i)[0] for i in range(classes)]
print("Testing pairs")
te_pairs, te_y = create_pairs(x_test, digit_indices_te)
```

```
# base network definition
base_network = create_base_network(100)

input_a = Input(shape=(img_rows,img_cols,1))
input_b = Input(shape=(img_rows,img_cols,1))

# siamese networks
processed_a = base_network(input_a)
processed_b = base_network(input_b)

distance = Lambda(euclidean_distance, output_shape=eucl_dist_output_shape)
([processed_a, processed_b])

model = Model(input=[input_a, input_b], output=distance)

# train data
rms = RMSprop()
model.compile(loss=contrastive_loss, optimizer=rms)
model.fit([tr_pairs[:, 0], tr_pairs[:, 1]], tr_y,
          validation_data=([te_pairs[:, 0], te_pairs[:, 1]], te_y),
          batch_size=128,
          nb_epoch=nb_epoch)

# compute training and testing accuracies
pred = model.predict([tr_pairs[:, 0], tr_pairs[:, 1]])
tr_acc = compute_accuracy(pred, tr_y)
pred = model.predict([te_pairs[:, 0], te_pairs[:, 1]])
```

```
te_acc = compute_accuracy(pred, te_y)



print('* Accuracy on training set: %0.2f%%' % (100 * tr_acc))
print('* Accuracy on test set: %0.2f%%' % (100 * te_acc))
print(model.summary())
pred = model.predict_classes(x_test, verbose=2)
print(pred)
```

Bibliography

[1] Victor Chernyak and I. Ya Immoreev, *A Brief History of Radar.*, IEEE Aerospace and Electronic Systems Magazine, 2009.

[2] Christian Hüllsmeyer, *Das Telemobiloskop.*, Düsselforf, Deutsche Patentschrift Nr. 165546, 30.04.1904.

[3] Niraj Prasad Bhatta and M. Geetha Priya, *RADAR and its applications.*, International Journal of Circuit Theory and Applications, 10(03), 2017, pp. 1-9.

[4] Mark A. Richards, James A. Scheer, William A. Holm *Principles of Modern Radar. Vol. 1: Basic Principles.*, 2010.

[5] Konstantinos Zikidis, *Early Warning Against Stealth Aircraft, Missiles and Unmanned Aerial Vehicles*, Surveillance in Action (pp.195-216), 2018.

[6] IEEE, *Standard Letter Designations for Radar-Frequency Bands.*

[7] Anderson RWG, Doecke SD, Mackenzie JRR, Ponte G, *Potential benefits of autonomous emergency braking based on in-depth crash reconstruction and simulation.*, 23rd International Technical Conference on the Enhanced Safety of Vehicle, 2013.

[8] International Telecommunications Union (ITU), *Characteristics of ultra-wideband technology.*, ITU-R SM.1755, 2006.

[9] International Telecommunications Union (ITU), *Systems characteristics of automotive radars operating in the frequency band 76–81 GHz for intelligent transport systems applications.*, ITU-R M.2057-0, 2014.

[10] Akram Al-Hourani, Robin J.Evans, Peter M.Farrell, Bill Moran, Marco Martorella, Sithamparanathan Kandeepan, Stan Skafidas and Udaya Parampalli, *Academic Press Library in Signal Processing, Volume 7 - Array, Radar and Communications Engineering.*, edited by Rama Chellappa and Sergios Theodoridis, 2018.

[11] Kevin St. J. Murphy, Roger Appleby, Gordon Sinclair, Andrew McClumpha, Kerry Tatlock, Richard Doney and Ian Hutcheson, *Millimetre Wave Aviation Security Scanner.*, 36th Annual 2002 International Carnahan Conference on Security Technology, Atlantic City, NJ, USA, 2002.

[12] G.N. Sinclair, P.R. Coward, R.N. Anderto, R. Appleby, T. Seys, P. Southwood, *Detection of illegal passengers in lorries using a passive millimetre wave scanner.*, 36th Annual 2002 International Carnahan Conference on Security Technology. Atlantic City, NJ, USA, 2002.

[13] Fatemeh Norouzian, Edward G. Hoare, Emidio Marchetti, Mikhail Cherniakov and Marina Gashinova, *Next Generation, Low-THz Automotive Radar – the potential for frequencies above 100 GHz.*, in 2019 20th International Radar Symposium (IRS), pp. 1–7, June 2019.

[14] Dominic Phippen, Liam Daniel, Edward Hoare, Shahzad Gishkori, Bernard Mulgrew, Mikhail Cherniakov and Marina Gashinova *Height Estimation for 3D Automotive Scene Reconstruction using 300 GHz Multi-Receiver Radar.*, IEEE Transactions on Aerospace and Electronic Systems, 2021.

[15] Fatemeh Norouzian, Emidio Marchetti, Marina Gashinova, Edward G. Hoare, Costas Constantinou, Peter Gardner and Mikhail Cherniakov, *Rain Attenuation at Millimeter*

Wave and Low-THz Frequencies., IEEE Transactions on Antennas and Propagation, vol. 68, pp. 421–431, Jan. 2020.

[16] Fatemeh Norouzian, Rui Du, Emidio Marchetti, Marina Gashinova, Edward G. Hoare, Costas Constantinou, Peter Gardner and Mikhail Cherniakov, *Transmission through uniform layer of ice at low-THz frequencies*, 2017 European Radar Conference (EURAD), pp. 211–214, Oct. 2017.

[17] Fatemeh Norouzian, Emidio Marchetti, Edward G. Hoare, Marina Gashinova, Costas Constantinou, Peter Gardner and M. Cherniakov, *Low-THz Wave Snow Attenuation.*, in 2018 International Conference on Radar (RADAR), pp. 1–4, Aug. 2018.

[18] Shahrzad Sabery, Fatemeh Norouzian, Peter Gardner, Edward G. Hoare, Mikhail Cherniakov and Marina Gashinova, *Signal Reduction by Tree Leaves in Low- THz Automotive Radar.*, 2018 48th European Microwave Conference (EuMC), pp. 1445–1448, Sept. 2018.

[19] Rui Du, Fatemeh Norouzian, Emidio Marchetti, Ben Willetts, Marina Gashinova and Mikhail Cherniakov, *Characterisation of attenuation by sand in low-THz band.*, 2017 IEEE Radar Conference (RadarConf), pp. 0294–0297, May 2017.

[20] Fatemeh Norouzian, Rui Du, Marina Gashinova, Edward G. Hoare, Costas Constantinou, Mike Lancaster, Peter Gardner and Mikhail Cherniakov, *Signal reduction due to radome contamination in low-THz automotive radar.*, 2016 IEEE Radar Conference (RadarConf), pp. 1–4, May 2016.

[21] Fatemeh Norouzian, Rui Du, Edward G. Hoare, Peter Gardner, Costas Constantinou, Mikhail Cherniakov and Marina Gashinova, *Low-THz Transmission Through Water-Containing Contaminants on Antenna Radome.*, IEEE Transactions on Terahertz Science and Technology, vol. 8, pp. 63–75, Jan. 2018.

[22] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Li Fei-Fei, *ImageNet: A Large-Scale Hierarchical Image Database.*, Department of Computer Science, Princeton University, USA, IEEE CVPR, 2009.

[23] Andreas Geiger, Philip Lenz, Christoph Stiller and Raquel Urtasun, *Vision meets Robotics: The KITTI Dataset.*, The International Journal of Robotics Research, 2013.

[24] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, Andrew Zisserman, *The PASCAL Visual Object Classes (VOC) Challenge.*, International Journal of Computer Vision, September 2009.

[25] Dominic Phippen, Liam Daniel, Edward Hoare, Marina Gashinova and Mikhail Cherniakov, *3D Images of Elevated Automotive Radar Targets at 300GHz.*, 2019 International Radar Conference (RADAR).

[26] On-Road Automated Driving (ORAD) committee, *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles.*, Tech. Rep. J3016_201806, SAE International, June 2018.

[27] Chenxu Li, Haobin Jiang, Shidian Ma, Shaokang Jiang and Yue Li, *Automatic Parking Path Planning and Tracking Control Research for Intelligent Vehicles.*, MDPI, 2020.

[28] Z. Stamenkovic, K. Tittelbach-Helmrich, J. Domke, C. Lörchner-Gerdaus, J. Anders, V. Sark, M. Eric, and N. Šira, *Rear View Camera System for Car Driving Assistance.*, Poc. 28th International Conference on Microelectronics (MIEL 2012), Serbia, 13-16 MAY, 2012.

[29] Isaac Skog and Peter Händel, *In-Car Positioning and Navigation Technologies—A Survey.*, IEEE Transactions on Intelligent Transportation Systems, April 2009.

[30] Vikas Nyamati, Tridha Chaudhuri and Kayalvizhi Jayavel, *Intelligent Collision Avoidance and Safety Warning system For Car Driving.*, International Conference on Intelligent Computing and Control Systems (ICICCS), 2017.

[31] Adam Brandt, Bengt Jacobson and Simone Sebben, *High speed driving stability of road vehicles under crosswinds: an aerodynamic and vehicle dynamic parametric sensitivity analysis.*, Vehicle System Dynamics, 2021.

[32] Muhammad Qasim Khan and Sukhan Lee, *A Comprehensive Survey of Driving Monitoring and Assistance Systems.*, Sensors MDPI, 2019.

[33] Pai Peng, Hongliang Wang, Xianhui Wang, Weihua Wang, Dawei Pi and Tianle Jia, *Research on the Hill Start Assist of Commercial Vehicles Based on Electronic Parking Brake System.*, Journal of Mechanical Engineering, 2019.

[34] H. L. Oei and P. H. Polak, *Intelligent Speed Adaptation (ISA) and Road Safety.*, IATSS Research, 2002.

[35] Margarita Martínez-Díaz and Francesc Soriguera, *Autonomous vehicles: theoretical and practical challenges.*, XIII Conference on Transport Engineering, 2018.

[36] George F. Luger, William A. Stubblefield, *Artificial intelligence: structures and strategies for complex problem solving.* Benjamin Cummings Pub. Co., 1993.

[37] Lotfi A. Zadeh, *Soft Computing and Fuzzy Logic.* IEEE Computer Society Press, 1994.

[38] S. Rajasekaran, G.A. Vijayalakshmi Pai, *Neural Networks, Fuzzy Systems, and Evolutionary Algorithms - Synthesis and Applications - Second Edition.* PHI Learning, 2017.

[39] Akshay L. Chandra, Sai Vikas Desai, Wei Guo and Vineeth Balasubramanian, *Computer Vision with Deep Learning for Plant Phenotyping in Agriculture: A Survey.*, Journal of Advanced Computing and Communications, April 2020.

[40] Zahangir Alom, Tarek M. Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Mahmudul Hasan, Brian C. Van Essen, Abdul A. S. Awwal and Vijayan K. Asari, *A State-of-the-Art Survey on Deep Learning Theory and Architectures.*, MDPI Electronics, 2019.

[41] Xue-Wen Chen and Xiaotong Lin, *Big Data Deep Learning: Challenges and Perspectives.*, IEEE Access, 2014.

[42] Zhi-Hua Zhou, Nitesh V. Chawla, Yaochu Jin and Graham J. Williams, *Big Data Opportunities and Challenges: Discussions from Data Analytics Perspectives.*, IEEE Computational Intelligence Magazine, November 2014.

[43] Maryam M Najafabadi, Flavio Villanustre, Taghi M Khoshgoftaar, Naeem Seliya, Randall Wald1 and Edin Muharemagic, *Deep learning applications and challenges in big data analytics.*, Journal of Big Data, 2015.

[44] COGNUB, *Cognitive Computing and Machine Learning.*, Source: https://www.cognub.com/index.php/cognitive-platform/.

[45] Dor Bank, Noam Koenigstein and Raja Giryes, *Autoencoders.*, 2020.

[46] Xin Jin and Jiawei Han, *K-Means Clustering.*, in Encyclopedia of Machine Learning. Springer, Boston, MA, 2011.

[47] Guido Montúfar, *Restricted Boltzmann Machines: Introduction and Review.*, International conference on information geometry and its applications, 2018.

[48] Alex Sherstinsky, *Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network.*, in Elsevier "Physica D: Nonlinear Phenomena" journal, Volume 404, March 2020: Special Issue on Machine Learning and Dynamical Systems.

[49] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville and Yoshua Bengio, *Generative Adversarial Networks.*, NIPS, 2014.

[50] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra and Martin Riedmiller, *Playing Atari with Deep Reinforcement Learning.*, Google DeepMind Technologies, Conference on Neural Information Processing Systems, 2013.

[51] Grégoire Montavon, Wojciech Samek, Klaus-Robert Müller, *Methods for interpreting and understanding deep neural networks.* Digital Signal Processing, Volume 73, February 2018, Pages 1-15.

[52] George Cybenko, *Approximation by superpositions of a sigmoidal function.*, Mathematics of Control, Signals, and Systems. 2 (4): 303–314.

[53] Waleed Abdulla, *Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow.* 2017, Source: https://github.com/matterport/Mask_RCNN.

[54] Peter G. Zhang, *Neural Networks for Classification: A Survey.*, IEEE Transactions on Systems Man and Cybernetics Part C (Applications and Reviews), 2000.

[55] Zhengxia Zou, Zhenwei Shi, Yuhong Guo and Jieping Ye, *Object Detection in 20 Years: A Survey.*, Submitted to the IEEE TPAMI, arXiv:1905.05055, 2019.

[56] Guo Y, Liu Y, Georgiou T and Lew MS, *A review of semantic segmentation using deep neural networks.*, International Journal of Multimedia Information Retrieval 7(2):87–93, 2018.

[57] Ana Stroescu, Mikhail Cherniakov, Marina Gashsinova, *Classification of High Resolution Automotive Radar Imagery for Autonomous Driving Based on Deep Neural Networks.* International Radar Symposium, 2019.

[58] Ana Stroescu, Liam Daniel, Dominic Phippen, Mikhail Cherniakov, Marina Gashinova, *Object Detection on Radar Imagery for Autonomous Driving Using Deep Neural Networks.*, European Microwave Week, EuRAD 2020, Utrecht, the Netherlands.

[59] Ana Stroescu, Liam Daniel and Marina Gashinova, *Combined Object Detection and Tracking on High Resolution Radar Imagery for Autonomous Driving Using Deep Neural Networks and Particle Filters.*, IEEE Radar Conference, Florence, Italy, 2020.

[60] Mario Gerla, Eun-Kyu Lee, Giovanni Pau, and Uichin Lee, *Internet of Vehicles: From Intelligent Grid to Autonomous Cars and Vehicular Clouds.* Korea Advanced Institute of Science and Technology, Daejeon, Korea.

[61] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke1, Stefan Roth, Bernt Schiele. *The Cityscapes Dataset for Semantic Urban Scene Understanding.*Daimler AG R/D, TU Darmstadt, MPI Informatics, TU Dresden, 2016.

[62] Liam Daniel, Andy Stove, Edward Hoare, Dominic Phippen, Mikhail Cherniakov, Bernie Mulgrew, Marina Gashinova, *Application of Doppler Beam Sharpening for Azimuth Refinement in Prospective Low-THz Automotive Radars.* IET Radar, Sonar Navigation, 2018.

[63] D. Jasteh, E. G. Hoare, M. Cherniakov and M. Gashinova, *Experimental Low-Terahertz Radar Image Analysis for Automotive Terrain Sensing.*, IEEE Geoscience and Remote Sensing Letters, vol. 13, no. 4, pp. 490-494, April 2016.

[64] Tom Ziemke, *Radar Image Segmentation using Recurrent Artificial Neural Networks.* The Connectionist Research Group, Dept. of Computer Science, University of Skovde, Sweden, 1995.

[65] G. Priestnall, J. Jaafar, A. Duncan, *Extracting urban features from LiDAR digital surface models.* Computers, Environment and Urban Systems, 24 (2000), 65-78.

[66] Puyang Wang, He Zhang and Vishal M. Patel, *SAR Image Despeckling Using a Convolutional Neural Network.* Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ, June 2018.

[67] Zhongling Huang, Zongxu Pan and Bin Lei, *Transfer Learning with Deep Convolutional Neural Network for SAR Target Classification with Limited Labeled Data.* MDPI, Published: 31 August 2017.

[68] Jiaqi Shao, Changwen Qu, Jianwei Li and Shujuan Peng, *A Lightweight Convolutional Neural Network Based on Visual Attention for SAR Image Target Classification.* MDPI, Published: 11 September 2018.

[69] Jie Geng, Jianchao Fan, Hongyu Wang, Xiaorui Ma, Baoming Li, Fuliang Chen, *High-Resolution SAR Image Classification via Deep Convolutional Autoencoders.*, IEEE Geoscience and Remote Sensing Letters (Volume: 12 , Issue: 11 , Nov. 2015).

[70] Laurene V. Fausett, *Fundamentals of neural networks: Architectures, Algorithms and Applications.* Florida Institute of Technology, Prentice-Hall, 1994.

[71] Weibo Liua, Zidong Wanga, Xaohui Liua, Nianyin Zengb, Yurong Liuc, Fuad E. Alsaadid, *A survey of deep neural network architectures and their applications.* Neurocomputing, Volume 234, 19 April 2017, Pages 11-26.

[72] J. Bromley, I. Guyon, Y. LeCun, and E. Säckinger, and R. Shah, *Signature verification using a "Siamese" time delay neural network.* in Proc. Adv. Neural Inf. Process. Syst., 1994, pp. 737–744.

[73] S. Zagoruyko and N. Komodakis, *Learning to compare image patches via convolutional neural networks.* Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2015, pp. 4353–4361.

[74] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, *Siamese Neural Networks for One-shot Image Recognition.* in Proc. ICML Deep Learn. Workshop, 2015.Department of Computer Science, University of Toronto. Toronto, Ontario, Canada.

[75] N. O' Mahonya, S. Campbella, A. Carvalhoa, L. Krpalkovaa , G. Velasco Hernandeza, S. Harapanahallia, D. Riordana and J. Walsh, *One-Shot Learning for Custom Identification Tasks; A Review.* 29th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2019), June 24-28, 2019, Limerick, Ireland.

[76] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, *Gradient-based learning applied to document recognition.* Proceedings of the IEEE, 86, 2278–2324. 1998.

[77] Yann LeCun, Corinna Cortes, Christopher J.C. Burges, *The MNIST database of handwritten digits* Viewed 18th June 2021. Source: http://yann.lecun.com/exdb/mnist/.

[78] Joseph L. Fleiss and Jacob Cohen, *The Equivalence of Weighted Kappa and the Interclass Correlation Coefficient as Measures of Reliability.* Biometrics Research, New York University, Educational and Psychological Measurement, 1973, 33, 613-619.

[79] Roger Boyle, Richard C. Thomas, *Computer Vision: A first course.* Blackwell Scientific Publications, 1988.

[80] E. R. Davies, *Computer and Machine Vision: Theory, Algorithms, Practicalities.* Academic Press, 1990, Chapter 3.

[81] David Vernon, *Machine Vision: Automated Visual Inspection and Robot Vision.* Prentice Hall, 1991, Chapter 4.

[82] André Marion, *An Introduction to Image Procesing.* Springer, 1991.

[83] Anil K. Jain, *Fundamentals of Digital Image Processing.* Prentice Hall, 1988, Chapter 7.

[84] Aurélien Géron, *Hands-On Machine Learning with Scikit-Learn & TensorFlow - Concepts, Tools and Techniques to Build Intelligent Systems.* Published by O'Reilly Media, Inc., 2017.

[85] More details about University of Birmingham's HPC service can be found on http://www.bear.bham.ac.uk/bluebear

[86] Liam Daniel, Dominic Phippen, Edward Hoare, Ana Stroescu, Mikhail Cherniakov, Peter Gardner, Marina Gashinova, *Dataset of radar and optical imagery of road actors.* 2019, available at: https://edata.bham.ac.uk/314/

[87] N.S.Keskar, D.Mudigere, M.Smelyanskiy, J.Nocedal, P.T.P.Tang, *On Large-Batch Training For Deep Learning: Generalization Gap And Sharp Minima.*

[88] Radek Grzeszczuk, Demetri Terzopoulos, Geoffrey Hinton, *NeuroAnimator: Fast Neural Network Emulation and Control of Physics-Based Models.* Intel Corporation and University of Toronto.

[89] Abien Fred M. Agarap, *Deep Learning using Rectified Linear Units (ReLU).* Department of Computer Science, Adamson University, Manila, Philippines.

[90] Giancarlo Zaccone, Md. Rezaul Karim, Ahmen Menshawy, *Deep Learning with TensorFlow.*

[91] Ian Goodfellow, Yoshua Bengio and Aaron Courville, *Deep Learning.* 2016.

[92] Diederik P. Kingma and Jimmy Lei Ba, *ADAM: A Method for Stochastic Optimization.* Published as a conference paper at ICLR 2015.

[93] Matthew D. Zeiler, *Adadelta: An Adaptive Learning Rate Method.* Google Inc., New York University, USA.

[94] Stephen V. Stehman, *Selecting and interpreting measures of thematic classification accuracy.* Remote Sensing of Environment, 1997.

[95] Di Feng, Christian Haase-Schütz, Lars Rosenbaum, Heinz Hertlein, Claudius Gläser, Fabian Timm, Werner Wiesbeck and Klaus Dietmayer *Deep Multi-modal Object Detection and Semantic Segmentation for Autonomous Driving: Datasets, Methods, and Challenges.*, 2019.

[96] Shaoqing Ren, Kaiming He, Ross Girshick and Jian Sun, *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2015.

[97] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu and Alexander C. Berg, *SSD: Single Shot MultiBox Detector.*, ECCV, 2016.

[98] Jakob Lombacher, Markus Hahn, Juergen Dickmann, Christian Wöhler, *Potential of Radar for Static Object Classification using Deep Learning Methods.*, IEEE MTT-S International Conference on Microwaves for Intelligent Mobility (ICMIM), 2016.

[99] Jakob Lombacher, Kilian Laudt, Markus Hahn, Juergen Dickmann, Christian Wöhler, *Semantic radar Grids.*, IEEE Intelligent Vehicles Symposium (IV), 2017.

[100] Simon Chadwick, Will Maddern, Paul Newman, *Distant Vehicle Detection Using Radar and Vision.*, International Conference on Robotics and Automation (ICRA), Montreal, Canada, 2019.

[101] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation.*, CVPR, 2014.

[102] Ross Girshick, *Fast R-CNN.*, ICCV, 2015.

[103] C. Szegedy, S. Reed, D. Erhan, and D. Anguelov, *Scalable, high-quality object detection.*, arXiv:1412.1441 (v1), 2015.

[104] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollár, *Microsoft COCO: Common Objects in Context.*, European Conference on Computer Vision, 2014.

[105] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, Tom Duerig and Vittorio Ferrari, *The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale.*, International Journal of Computer Vision (IJCV), 2020.

[106] Elva-1 Millimeter Wave Division, St. Petersburg, Russia. Source: http://elva-1.com/

[107] Tzutalin, *LabelImg - Graphical image annotation tool*, Git code (2015). Source: https://github.com/tzutalin/labelImg

[108] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, *Deep Residual Learning for Image Recognition.*, CVPR, 2016.

[109] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna, *Rethinking the Inception Architecture for Computer Vision.*, CVPR, 2016.

[110] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, Kevin Murphy, *Speed/accuracy trade-offs for modern convolutional object detectors.*, CVPR, 2017.

[111] Wei Yi, Mark R. Morelande, Lingjiang Kon and Jianyu Yang, *A Computationally Efficient Particle Filter for Multitarget Tracking Using an Independence Approximation.*, IEEE Transactions on Signal Processing (Volume: 61, Issue: 4, Feb.15, 2013).

[112] B. Errasti-Alcala and P. Braca, *Track Before Detect Algorithm for Tracking Extended Targets applied to Real-World Data of X-band Marine Radar*, 17th International Conference on Information Fusion (FUSION), Salamanca, Spain, 2014.

[113] M. Heuer, A. Al-Hamadi, A. Rain and M. M. Meinecke, *Detection and tracking approach using an automotive radar to increase active pedestrian safety.*, IEEE Intelligent Vehicles Symposium Proceedings, Dearborn, USA, 2014.

[114] Alexandre Lepoutre, Olivier Rabaste, François Le Gland, *Multitarget likelihood for Track-Before Detect applications with amplitude fluctuations.*, IEEE Transactions on Aerospace and Electronic Systems, Institute of Electrical and Electronics Engineers, 2016.

[115] Y. Boers, H. Driessen, J. Torstensson, M. Trieb, R. Karlsson and F. Gustafsson, *Track before-detect algorithm for tracking extended targets.*, IEE Proceedings - Radar, Sonar and Navigation, vol. 153, no. 4, pp. 345-351, August 2006.

[116] D.J. Salmond and H. Birch, *A particle filter for track-before-detect.*, In Proceedings of the 2001 American Control Conference, Pages: 3755–3760, Arlington, USA, 2001.

[117] C. Kreucher, K. Kastella, and A.O. Hero, *Multitarget Tracking using the Joint Multitarget Probability Density.*, IEEE Transactions on Aerospace and Electronic Systems (Volume: 41, Issue: 4, Oct. 2005), Pages :1396–1414.

[118] Y. Boers and J.N. Driessen, *Multitarget particle filter track-before-detect application.*, In IEE Proceedings - Radar, Sonar and Navigation, Volume 151, pages 351–357, 2004.

[119] S. J. Davey, M. G. Rutten and B. Cheung, *A comparison of detection performance for several track-before-detect algorithms.*, URASIP Journal on Advances in Signal Processing, 2008.

[120] Jun S. Liu and Rong Chen, *Sequential Monte Carlo Methods for Dynamic Systems*, Journal of the American Statistical Association (1998), Vol. 93, No. 443, Pages: 1032-1044.

[121] Pierre Del Moral, *Nonlinear filtering: Interacting particle resolution.*, 1996.

[122] D. Watzenig, M. Brandner and G. Steiner, *A particle filter approach for tomographic imaging based on different state-space representations.*, 2006, Measurement Science and Technology, Volume 18, Number 1.

[123] Xuedong Wang, Tiancheng Li, Shudong Sun and Juan M. Corchado, *A Survey of Recent Advances in Particle Filters and Remaining Challenges for Multitarget Tracking*, MDPI Sensors, 2017.

[124] Rudolph Emil Kalman, *A New Approach to Linear Filtering and Prediction Problems.*, Transactions of the ASME–Journal of Basic Engineering, Vol. 82, No. Series D, Pages: 35-45, 1960.

[125] Fernández-Villaverde, *Kalman and particle filtering.*, Palgrave Macmillan (eds) The New Palgrave Dictionary of Economics. Palgrave Macmillan, London, 2016.

[126] E.A. Wan and R. Van Der Merwe, *The unscented Kalman filter for nonlinear estimation.*, Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium, Lake Louise, Canada, 2000.

[127] S. J. Julier and J. K. Uhlmann, *A new extension of the Kalman filter to nonlinear systems.*, Proceeding of AeroSense: The 11th International Sysmpsium on Aerospace/ Defence Sensing, Simulation and Controls, Orlando, Florida, 1997. Vol. Muti Sensor Fusion, Tracking and Resource Mangement II, Pages: 182–193.

[128] T. R. Bayes, *Essay towards solving a problem in the doctrine of chances.*, Phil. Trans. Roy. Soc. Lond., vol. 53, Pages: 370–418, 1763. Reprinted in Biometrika, vol. 45, 1958.

[129] Jaco Vermaak, Arnaud Doucet and Patrick Pérez, *Maintaining multimodality through mixture tracking.*, Proceedings Ninth IEEE International Conference on Computer Vision, Nice, France, 2003.

[130] S. L. Dockstader and A. M. Tekalp, *Tracking multiple objects in the presence of articulated and occluded motion.*, In Proceedings of the IEEE Workshop on Human Motion, pages 88–95, 2000.

[131] C. Rasmussen, *Joint likelihood methods for mitigating visual tracking disturbances.*, Proceedings of the IEEE Workshop on Multi-Object Tracking, pages 69–76, 2001.

[132] C. Hue, J.-P. Le Cadre, and P. Perez, *Tracking multiple objects with particle filtering.*, IEEE Transactions on Aerospace and Electronic Systems, 38(3):791–812, 2002.

[133] M. Isard and J. MacCormick, *A Bayesian multiple-blob tracker.*, Proc. Int. Conf. Computer Vision, pages II: 34–41, 2001.

[134] Kaiming He, Georgia Gkioxari, Piotr Dollár and Ross Girshick, *Mask R-CNN.*, Facebook AI Research (FAIR), arXiv:1703.06870v3, 24 Jan 2018.

[135] C. Tomasi and R. Manduchi, *Bilateral filtering for gray and color images.* Sixth International Conference on Computer Vision, 1998.