Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science and Engineering

DIPLOMA THESIS ASSIGNMENT

Student: **Bc. Vojtěch Micka**

Study programme: Open Informatics
Specialisation: Artificial Intelligence

Title of Diploma Thesis: **Objects identification in signal processing of FMCW radar for Advanced Driver Assistance Systems**

Guidelines:

Propose and implement a method of objects identification in digital signal processing of FMCW radar (Frequency Modulated Continuous Wave) for Automotive Advanced Driver Assistance Systems. Focus especially on architectures of deep neural networks. Propose appropriate method for data preprocessing and encoding.

Bibliography/Sources:

- Juergen Schmidhuber: Deep Learning in Neural Networks: An Overview, Neural Networks, Vol 61, pp 85-117, Jan 2015, arXiv:1404.7828v4
- Christian Szegedy, Alexander Toshev, Dumitru Erhan: Deep Neural Networks for Object Detection, Advances in Neural Information Processing Systems, 2013

Diploma Thesis Supervisor: Ing. Zdeněk Buk, Ph.D.

Valid until the end of the winter semester of academic year 2016/2017

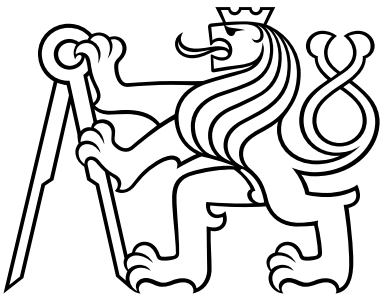
L.S.

doc. Ing. Filip Železný, Ph.D.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, October 8, 2015

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics



Master's Thesis

Objects identification in signal processing of FMCW radar for Advanced Driver Assistance Systems

Vojtěch Micka

Supervisor: Ing. Zdeněk Buk, Ph.D.

Study Programme: Open Informatics

Field of Study: Artificial Intelligence

January 9, 2017

Acknowledgements

I would like to express my very great appreciation to Ing. Zdenek Buk, Ph.D., my supervisor, for his patient guidance, enthusiastic encouragement and useful critiques of this work. His experience helped me overcame all the difficulties I have struggled with. Next, I want to thank Valeo company for such a pleasent cooperation and providing me with a workplace and all the necessary data. Further, I would like to thank the staff of Valeo for supporting me, especially to Ing. Jan Zahradnik for steering me in the right direction throughout this project. I am particularly grateful for the assistance given by Ing. Michal Mandlik, Ph.D. who was always there when I needed to answer questions about Radar technologies. I would also like to offer my special thanks to Mgr. Ing. Jakub Mikulka, M.A. for proofreading and correction of my work and for his sense for a detail that has helped me develop my thoughts consistently. Special thanks should be given to Jan Dvorsky, MSc and Mrs. Eva Mikulkova for help with linguistic adaptation. In addition, I would like to thank to PhDr. Martina Habova for helping me look at things from a broader perspective and to whole my family for their support and encouragement throughout my work.

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací

V Praze dne ....

Podpis autora práce

Abstract

The aim of this work is to propose a Artificial Neural Networks (ANNs)-based approach for signal processing of radar data used for Advanced Driver Assistance System (ADAS), including Blind Spot Detection (BSD) assist and Closing Vehicle Warning (CVW). Because vehicles now contain more and more computers, there is enough power for other systems, therefore, ADASs are becoming more important in automotive industry. All ADASs rely on inputs from single or multiple data sources, including Light Detection And Ranging (LiDAR), radars, cameras and others. Processing of data from any of those data sources needs to be done fast and reliably. So far the conventional methods of data processing are used, therefore, there is plenty of room for improvement. It is not easy to improve current engineered methods of signal processing, as it is difficult to understand the radar data. Convolutional Neural Networks (CNNs), which are currently used in state-of-the-art methods in many real-world applications (especially image processing), live at the core of my proposed alternative method. CNNs are used for feature extraction, which reducing data dimensionality and allowing to use standard classifiers on top of CNN. I have proposed two classification methods. To simplify the task, I have started with classification of a single frame, for that purpose, I have designed simple Multi-Layer Perceptron (MLP), however, the radar outputs stream of frames, for that, I have designed Long-Short Term Memory (LSTM) network.

Abstrakt

Cílem této práce je navrhnout metodu, založenou na neuronových sítích, pro zpracování signálu v radarových systémech použitých pro pokročilé asistenční systémy řidiče (ADAS), zejména detekce slepého úhlu a varování zezadu blížícího se vozidla. Protože vozidla v dnešní době obsahují čím dál více počítačů, je zde dostatečně výpočetního výkonu pro ostatní systémy, díky tomu ADAS jsou mnohem více důležité. Všechny systémy ADAS spolehají na vstup z jednoho či více senzorů (laserové radary, radary, camery a ostatní). Zpracování dat z jakéhokoliv senzoru musí být rychlé a spolehlivé. Na zpracování dat jsou zatím používány většinou standardní expertní metody, díky čemuž je zde hodně místa na vylepšení. Aktuální metody zpracování signálů je náročné vylepšovat, jelikož samotné porozumění radarovým datům je obtížné. Z tohoto důvodu jsem navrhl metodu založenou na konvolučních neuronových sítích, které jsou dnes používány jako nejvhodnější metody v mnoha reálných aplikacích (zejména zpracování obrazu). Konvoluční neuronové sítě se používají na extrahování příznaků z dat, čímž se především docílí redukce dimenzionality a daný problém se dá řešit běžnými klasifikátory. Já jsem navrhl dvě klasifikační metody. Pro zjednodušení problému jsem začal klasifikací jednotlivách radarovách snímků, pro tento účel jsem navrhl klasickou neuronovou síť (MLP), ale jelikož výstup z radaru je spíš podobný videu (kontinuální stream snímků), navrhl jsem druhou metodu používající LSTM sítě.

Contents

List of Figures

Glossary

ABS Anti-lock Breaking System. 1

ADAS Advanced Driver Assistance System. ix, 1, 2, 43

ANN Artificial Neural Network. ix, 2–4, 8, 17

BSD Blind Spot Detection. ix, 14, 19, 24, 26, 31, 33, 43

CFAR Constant false alarm rate. 13

chirp The change in the frequency in frequency modulated signal is known as *chirp*. 10, 11

CNN Convolutional Neural Network. ix, 3–7, 23–28, 33, 43

CPU Central Processing Unit. 29

CSV Comma Separated Values. 20, 21

CTA Cross Traffic Alert. 14

CUDA The Compute Unified Device Architecture is a parallel computing platform and programming model developed by NVIDIA. It allows developers to use CUDA-capable GPUs for general purpose processing. 28

cuDNN The NVIDIA CUDA® Deep Neural Network library is a library of primitives for deep neural networks. CuDNN provides optimised implementations for standard functions such as forward and backward convolution, normalisation, pooling and activation layers. 28

CVW Closing Vehicle Warning. ix, 14, 19, 26

Doppler The Doppler effect (Doppler shift) is a change in frequency of emitted waves produced by motion of an emitting source relative to an observer. Proposed in Prague 1842 by Austrian physicist Christian Doppler. 2, 10, 41

ESP Electronic Stability Program. 1

FFT Fast Fourier Transform. 2, 11, 12

FMCW Frequency Modulated Continuous Wave. 2, 3, 9, 10

FPS Frames Per Second. 39

GPS Global Positioning System. 1

GPU Graphics Processing Unit. 5, 28

HDF5 Hierarchical Data Format technologies address the problems of how to organise, store, discover, access, analyse, share, and preserve data in the face of enormous growth in size and complexity [38]. 21, 43

ILSVRC ImageNet Large Scale Visual Recognition Challenge is a benchmark in object category classification and detection on 1000 object categories and over 1.2 millions of images [32]. 5

LiDAR Light Detection And Ranging. ix, 1, 42

LSTM Long-Short Term Memory. ix, 8, 28, 33, 43

MCC Measure Cycle Count is a custom timer, which is used for timing and synchronising the frames. 21, 24

MLP Multi-Layer Perceptron. ix, 4, 6, 7, 24, 27, 33

RCS Radar Cross-Section. 39, 40

RDMatrix Range-Doppler Matrix. 39–41, 43

ReLU Rectified Linear Unit. 3, 7

RGB Red Green Blue. 4, 40

RNN Recurrent Neural Network. 3, 8, 24, 28

ROC Receiver Operating Characteristic. 37

SSD Single Shot MultiBox Detector. 19, 42

TORCS TORCS is a highly portable multi platform car racing simulation [44]. 28

TTC Time To Collision. 14

Chapter 1

Introduction

In the automotive industry, the main priorities are safety and driving experience. For decades the automobile manufacturers have been trying to integrate new technologies into their vehicles to create perfect combination of safety, comfort and engineering. Since the computers are small enough to fit into the car, the computer industry becomes part of the automotive evolution. It all started with the need for more fuel efficient engines, reducing emissions, but also to create better and more powerful engines. As computers are becoming smaller and more powerful, their utilization in the automotive industry is increasing. Today's vehicles contain many computers that are responsible for many different things, such as advanced climate controls, communications, entertainment devices, safety systems[1] and Advanced Driver Assistance System (ADAS). Advanced Driver Assistance System (ADAS) are systems intended to increase vehicle safety and driving comfort, by adding sensors perceiving the environment and assisting the driver by providing useful information (or alerts) or by intervening into the control of the vehicle. There are many examples of these systems such as: *adaptive cruise control*, *automatic braking*, *driver monitoring system*, *automatic parking* and *blind spot monitor*. Each of these systems relies on input from a single or multiple data sources, including Light Detection And Ranging (LiDAR), radar, cameras, ultrasonic sensors, Global Positioning System (GPS) and others.

Each manufacturer of ADAS can use different sets of sensors for the same system, because of different requirements. All sensors are different - some are more affected by weather conditions than others, some are more precise, but slow or expensive, each type has different advantages. Therefore, most of the crucial systems rely on multiple sensors, especially systems, which can take over control of the vehicle, whereas other systems use just one type of sensors, mostly because of price. In this thesis, I will focus purely on radar-based systems.

[1] Safety systems e.g.: Anti-lock Breaking System (ABS) or Electronic Stability Program (ESP) [39]

This thesis has been created with cooperation of Valeo company, which is automotive supplier based in France. It provides wide variety of automotive products including different types of sensors, ADAS, but also lights, air conditioning and more. Amongst other types, Valeo is developing many radar sensors and their related ADAS.

Radar technology has been known for a long time, the first mention of a system able of detecting objects using reflected electromagnetic waves is dated back to the 19th century [23]. But the first real applications of radar technology were developed in 1940s during World War II. Since then the methods of signal processing changed slightly with every new radar's type, in 1970s the modern Doppler systems were introduced, which were using Digital Fast Fourier Transform (FFT), because of availability of modern microprocessors. These radars were used by police for measuring the speed of vehicles.

At the same time, it has been shown that Artificial Neural Networks (ANNs) are robust classifiers and can solve some problems much better and more efficiently then conventional algorithms [21, 22]. Algorithms based on ANNs are now state-of-the-art in many fields, especially in image recognition. The aim of this work is to propose an ANN-based approach to detect objects for the needs of ADAS. In association with that, part of the current signal processing method will be replaced with an ANN, for which it is necessary to choose and gather the right training data.

Chapter 2 describes the neural networks in details with a brief introduction to Frequency Modulated Continuous Wave (FMCW) radars and current signal processing methods. Design of the datasets with proposed architectures of neural networks is described in Chapter 3. Proposed architectures are compared in Chapter 4 on real world data, with discussion and suggestions of future work in Chapter 5.

Chapter 2

Theory

This chapter is providing brief introduction of different types of Artificial Neural Networks (ANNs) such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) and of the current automotive radar technology, specifically Frequency Modulated Continuous Wave (FMCW) radars.

2.1 Artificial neural networks

ANNs were inspired by function of a human brain. The first conceptual model was developed by McCulloch and Pitts in 1943 [28], where they described the concept of a logical neuron. However, the first concept of a computational model with the ability to learn was invented in 1957 by Rosenblatt at the Cornell Aeronautical Laboratory [31]. He designed the simplest artificial neural network possible composed of a single neuron called *perceptron*. The perceptron is defined as a function mapping multiple inputs onto a single output, which can be used as a linear classifier. According to Equation 2.1 it is defined by weights w_i, a bias b and an activation function f.

$$y = f\left(\sum_i w_i x_i + b\right) \tag{2.1}$$

The original perceptron was using Heaviside step function [40] as the activation function, which is resulting in a binary output. However, but any other activation function can be used, such as sigmoid, hyperbolic tangent or Rectified Linear Unit (ReLU). Because the perceptron is a linear classifier, its abilities are limited to the classification of linearly separable problems - meaning that it is possible to separate two sets of samples using one hyperplane.

2.1.1 Multi-layer perceptron

To avoid this limitation, multiple perceptrons can be connected together. Such a network of interconnected perceptrons is called a Multi-Layer Perceptron (MLP). Whereas regular perceptron is limited to linear separable problems, MLP can approximate arbitrary continuous functions [5]. In MLP the neurons are organized into layers, where output of every neuron in one layer is connected to every neuron in the next adjacent layer up to output layer. This architecture is being referred to as fully connected feed-forward ANN. Example of such architecture is illustrated in Figure 2.1. This MLP contains two hidden layers and maps three inputs to single output.

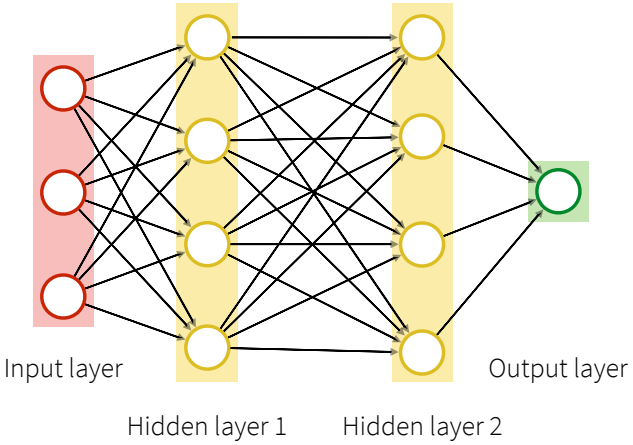


Figure 2.1: A multi-layer perceptron with two hidden layers

Designing a good architecture is a challenging task, because it is hard to choose the right number of layers and the right number of neurons in each layer. By designing a too complex architecture, with too many layers and neurons may result in overfitting on training data. On the other hand, by using too few layers and neurons may lead to a biased function. Each neuron can use any activation function, but most common are sigmoid function and hyperbolic tangent.

2.1.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are a special type of neural networks designed to take advantage of specific structure of input data. Typical example of input data is color image, which can be represented by 3D volume with **width**, **height** and **depth** (assuming Red Green Blue (RGB) representation, the *depth* here refers to the color depth / channels of an image). However, all other types of data, which can be represented by 3D volume, can be fed into the CNN.

Architecture of CNN is highly inspired by cat's visual cortex introduce by Hubel and Wiesel [14, 15] in the 1960-70s. First mention of CNN came from 1980s Fukushima's *neocognitron* [9], but the first practical application was shown by LeCun *et al.* [22] in 1989, with the CNN trained using backpropagation and gradient descent algorithm to classify handwritten digits.

In the last 5 years, the popularity of CNNs in the field of computer vision has significantly increased. Starting with the success of Krizhevsky et *al.* [21], who won ImageNet (ILSVRC) challenge using a deep CNN (now called "AlexNet"). Krizhevsky's CNN achieved a top-5 classification error [1] of 15.3 % on the test data, where the second-best entry reached just 26.2 %. Since then the CNNs-based algorithms dominates the ImageNet, recent winner of 2016 ImageNet in classification task reached top-5 error 2.991 %, full evolution is shown in Figure 2.2.

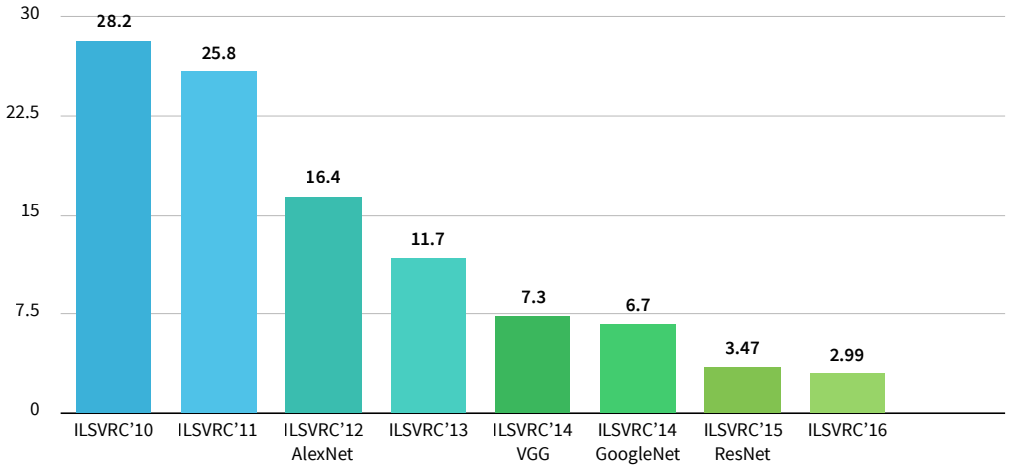


Figure 2.2: Evolution of ImageNet top-5 error (%) in classification task between years 2010 and 2016

At the same time, CNNs are successfully deployed into many practical applications, such as object detection [8] or facial recognition [37]. CNNs are highly computationally expensive, which is why the biggest breakthroughs were made in the last couple of years when high performance computers and Graphics Processing Units (GPUs) have become more widely available.

CNN is usually composed of two parts, *feature extraction part* and *classification part* as is shown in Figure 3.6, where the feature extraction is the main part of the CNN and it is typically built from a sequence of *convolutional layers*, *nonlinearity layers* and *pooling*

[1] Top-5 error (metric) means, the classification is consider correct if the target label is one of the top 5 predicted ones, i.e. top 5 with the highest probabilities.

(sub-sampling) layers. The simple MLP is commonly used for the classification part, but the whole classification part is not required and can be excluded for same type of tasks (autoencoders [3]).

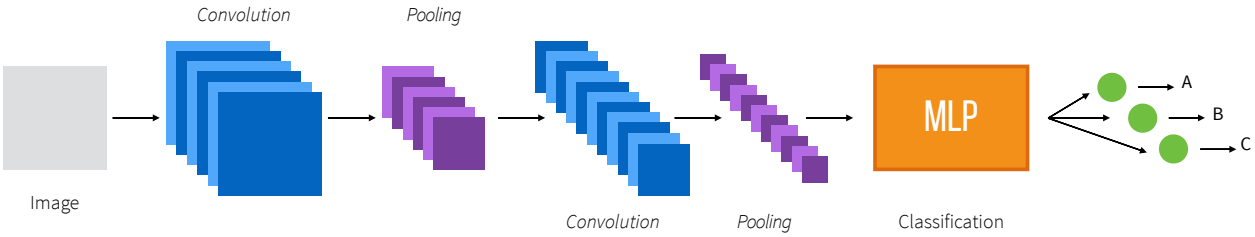


Figure 2.3: Basic structure of CNN

2.1.2.1 Convolutional layer

Key elements of the CNN are convolutional layers, where each layer contains K convolutional filters (kernels). Each kernel is then convolved with output of the previous layer resulting in a *feature map*. With multiple kernels in each layer, the network can extract multiple features for each location. Convolution at point m, n can be easily defined using Equation 2.2.

$$y[m,n] = k * x[m,n] = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} k[u,v], x[m-u, n-v] \tag{2.2}$$

Where the $*$ stands for convolutional operator, k is the kernel, x is the input and y is the outputted feature map. Graphical example of convolution is shown in Figure 2.4.

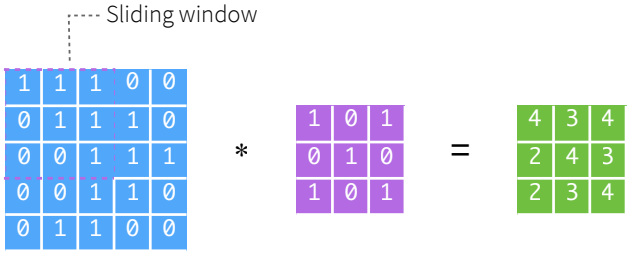


Figure 2.4: Graphical example of convolution

Convolutional layer is defined by following hyperparameters:

- number of kernels K
- kernel dimensions F

- stride S

- zero padding P

It accepts a volume of size $W_1 \times H_1 \times D_1$ and produces a volume of size $W_2 \times H_2 \times D_2$ where:

$$W_2 = (W_1 - F_1 + 2P)/S_1 + 1$$
$$H_2 = (H_1 - F_2 + 2P)/S_2 + 1$$
$$D_2 = K$$

Making the network deeper by adding more convolutional layers enables the network to extract features with different levels of abstraction. That could be intuitively illustrated on images - extracted features from the first layers are edges or color gradients, followed by more complex features, such as circles or other simple patterns, at the deepest layers the kernels match whole objects, faces, dogs, cars, etc. This is caused by the fact that deeper layers are able to build features from previous layers.

2.1.2.2 Nonlinearity layer

For the CNN it proved to be more appropriate to use *Rectified Linear Unit* (ReLU) instead of commonly used sigmoid function inside MLP [6]. ReLU is defined as follows:

$$y(x) = \max(0, x) \tag{2.3}$$

2.1.2.3 Pooling layer

The main purpose of pooling layer is to reduce dimensionality of an input. Pooling layer operates using a sliding window, by applying some window operation over the input volume resulting in one number per window. Commonly used operation are *maximum* (max-pooling) or *average* (avg-pooling).

The only hyperparameters defining the pooling layer are the following:

- operation O

- window dimensions F

- stride S

Accepting a volume of size $W_1 \times H_1 \times D_1$ and producing a volume of size $W_2 \times H_2 \times D_2$ where:

$$W_2 = (W_1 - F_1)/S_1 + 1$$
$$H_2 = (H_1 - F_2)/S_2 + 1$$
$$D_2 = D_1$$

It is worth mentioning that the most commonly used version is max-pooling with $F = (2, 2)$ and $S = (2, 2)$ (illustrated in Figure 2.5) or less commonly $F = (3, 3)$ and $S = (2, 2)$ (also called overlapping pooling).

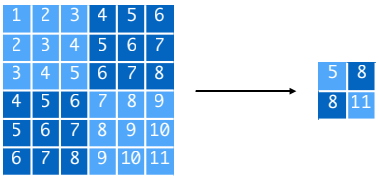


Figure 2.5: Illustration of max-pooling

2.1.2.4 Batch Normalization layer

Special type of layer, which normalizes the activations of the previous layer at each batch, such that the mean of activations approaches 0 and the activation standard deviation approaches 1. [16]. Using Batch Normalization layers allows to use much higher learning rates and still achieve the same accuracy.

2.1.3 Recurrent Neural Network

Recurrent Neural Networks (RNNs) are special type of ANNs, where connections between neurons form a directed cycle - information flows in both directions. These cycles create a sense of time and memory of previous state, which allows them to process arbitrary sequences of inputs. RNNs are useful for applications such as car driving [20], handwriting recognition [10] or sequence generation [11]. Example architecture of RNN is illustrated in Figure 2.6.

Traditional RNNs can suffer from *vanishing gradient problem* [13], which was the main motivation to develop Long-Short Term Memory (LSTM) model. The LSTM was proposed by Hichreiter and Schmidhuber in 1997 [12]. It contains small recurrent units (memory cells) consisting of input, output and three gates, which control the state changes of the cell - they enable to ignore, forget or not output the current state. Example of memory cell is shown in

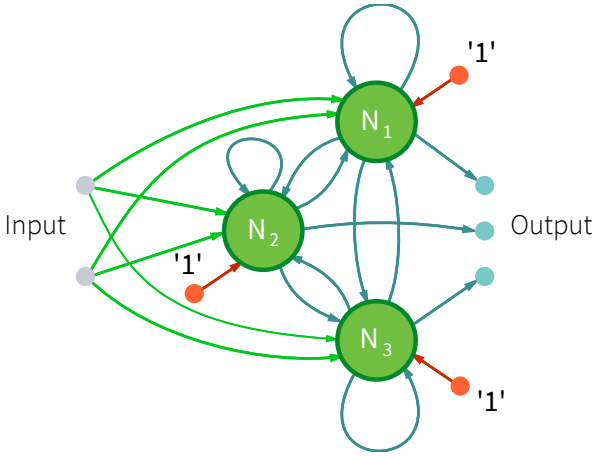


Figure 2.6: Example architecture of fully-connected RNN with 3 neurons, label '1' refers to bias

Figure 2.7. The key difference is that the recurrent component does not contain activation function, which means the stored value does not deprecate over time.

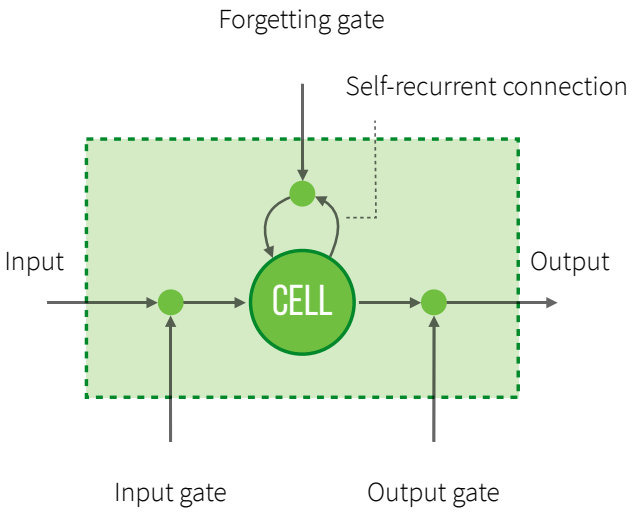


Figure 2.7: Example of LSTM memory cell

2.2 Radar

In the field of automotive industry the applications of radars are very popular. Radar sensor does not offer such a precise measurements especially in the angle, but it can be smaller and cheaper than other alternatives and it is less affected by the weather conditions. There are many different types of radar architecture, but the most broadly used one in the automotive is the FMCW radar [19, 34], because it offers more advantages than others.

2.2.1 FMCW radar

FMCW radar, similarly to Doppler radar [26, 30], relies on the Doppler effect to measure velocity, but it has the benefit of measuring distances of objects. In order for the FMCW radar to be able to measure distances of objects it produces a frequency shift proportional to distance using frequency modulation. Arbitrary signal can be used for modulation, but commonly used is sawtooth, triangle or sinusoidal signal. For my purpose, I will consider a sawtooth model of the FMCW signal (shown in Figure 2.8), which is used inside Valeo radars. One frequency ramp is called a chirp.

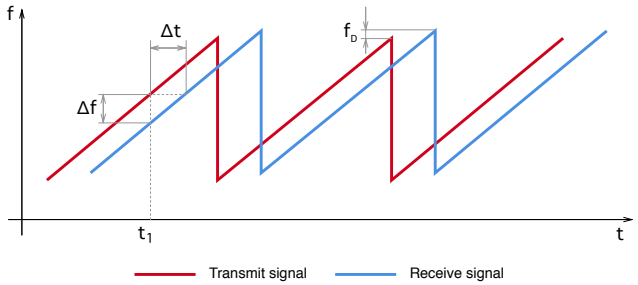


Figure 2.8: Frequency of the transmitted and received signals

Function of the FMCW radar is simple. The block diagram of the FMCW radar is shown in Figure 2.9. Radar transmits a modulated signal, which, after being reflected by an object, is received and mixed with transmitted signal. To get a differential signal the low-pass filter is applied. The resulting signal is called video signal, where the video signal is almost sinusoidal, with frequency Δf. The frequency Δf of the video signal consists of frequency caused by time delay and frequency f_{D} caused by Doppler effect.

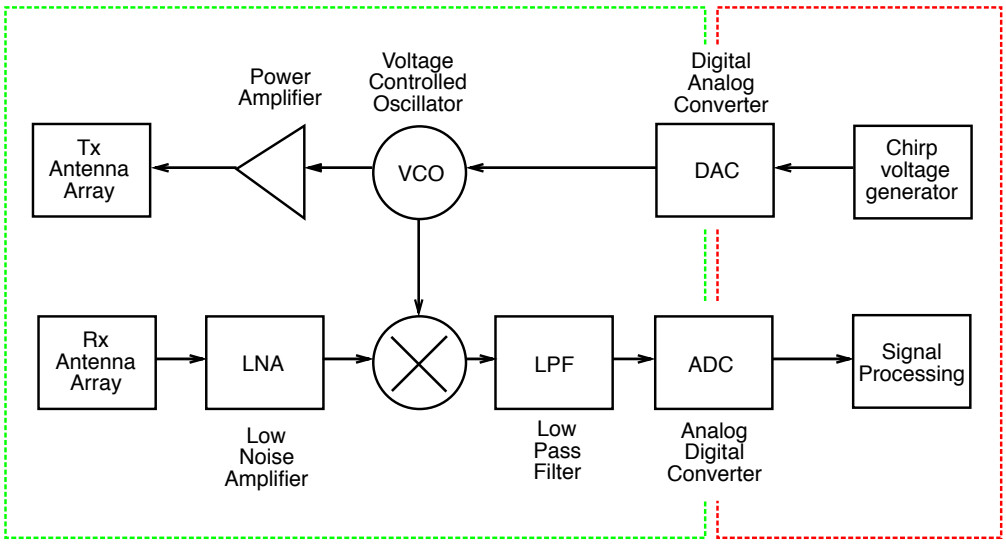


Figure 2.9: Block diagram of FMCW radar

To separate these two frequencies, the frequency spectrum of the signal over n consecutive chirps need to be analyze. For that the 2D Fast Fourier Transform (FFT) algorithm is used [35, 43]. At first the row-wise FFT is taken on the time samples, second, the column-wise FFT is taken on the output of the first FFT. The result of 2D FFT processing is Range-Doppler spectrum, which contains distance and velocity information about reflected objects (as is illustrated in Figure 2.10).

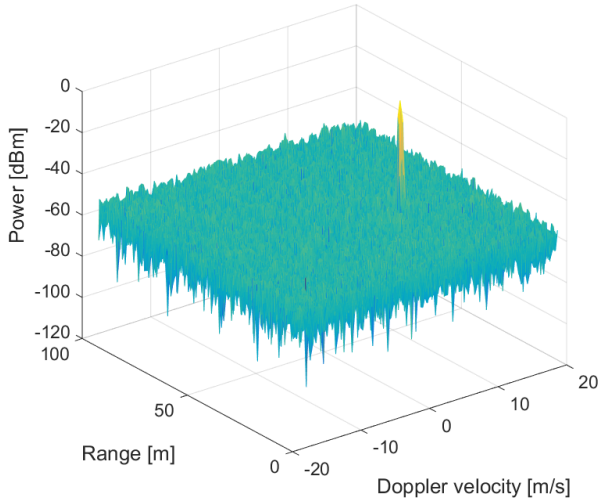


Figure 2.10: Example of Range-Doppler spectrum

To find an angular position of the object, the angle of the received wave needs to be calculated. The best way to do that is using technique called *Phase-comparison monopulse* with multiple receiving antennas separated by some distance d [27, Chapter 9.2.4]. The incoming wave will have different phase shift according to the distance d and the angle of incidence θ. Situation with two antennas is illustrated in Figure 2.11.

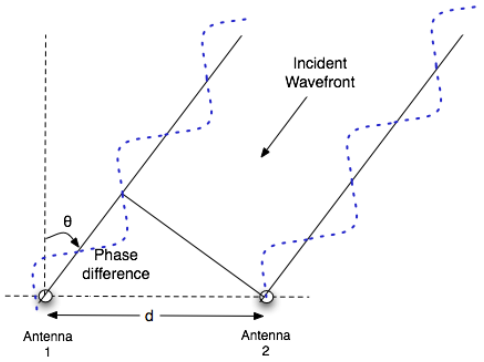
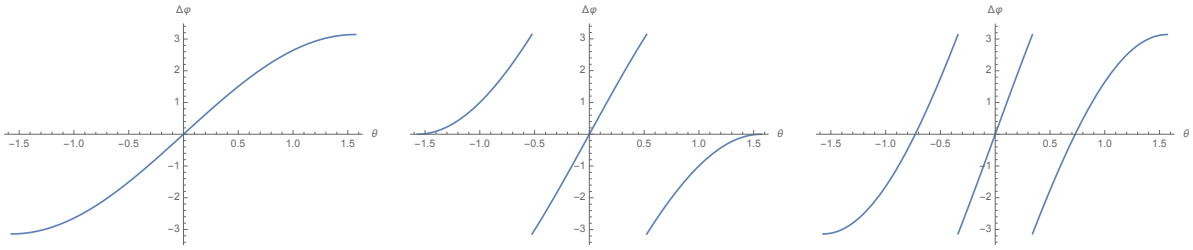


Figure 2.11: Principle of phase interferometry [41]

The phase difference is easy to calculated using Equation 2.4, where λ is wavelength of

the signal. To avoid ambiguity of the resulting angle it is clearly seen that the distance d should not be higher than $\lambda/2$. However, in reality d is usually higher then $\lambda/2$, because then the measurement of the phase difference is more precise. The ambiguity of the angle is shown in the Figure 2.12. The optimal case is shown in Figure 2.12a, where for each phase difference $\Delta\varphi$ there is only one angle θ, in the Figures 2.12b and 2.12c there can be found multiple angles θ for one phase difference $\Delta\varphi$. To solve this problem *multi-beam* radar needs to be introduced [17, 26].



(a) $d = \frac{1}{2}\lambda$ (b) $d = \lambda$ (c) $d = \frac{3}{2}\lambda$

Figure 2.12: Phase curve with different d

$$\Delta\varphi = \frac{2\pi d}{\lambda}\sin\theta, \quad \theta = \sin^{-1}\left(\frac{\lambda}{2\pi d}\Delta\varphi\right) \tag{2.4}$$

Regarding Valeo technology, the company is developing many different types of radar sensors. They mostly differ in the frequency and the number of beams. I was working with a radar with a 24 GHz antenna and with four single receiving partially overlapping beams yielding the total field of view about 150°, where rear-facing beam contains 64 chirps and other beams just 16 chirps.

2.2.2 Current signal processing

From each chirp I receive raw analog signal, which is then directly discretized by the onboard A/D unit. Data in this form is called video data as illustrated in Figure 2.13.

All video data from one beam (all chirps) are collected together and then processed together. Whole processing pipeline is shown in Figure 2.14.

Most of the processing steps are done for all beams separately. In the second step video data are transformed to Range-Doppler spectrum using 2D FFT. Range-Doppler spectrum is very useful because it is very descriptive for human eye and therefore, for engineered signal processing approaches. Sample of a frame in a form of Range-Doppler spectrum is shown in Figure 2.15.

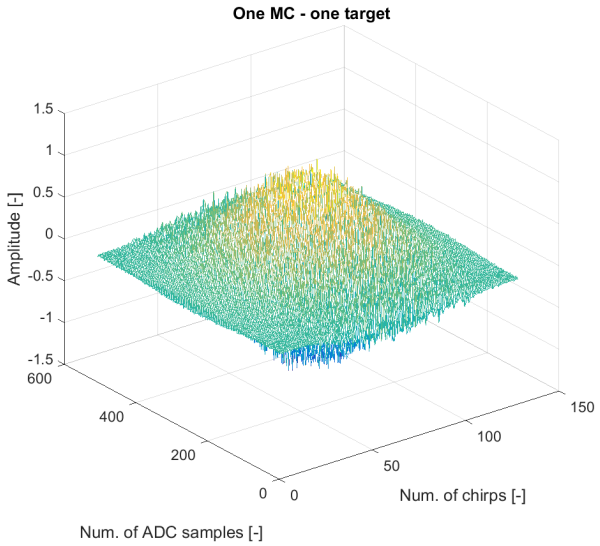


Figure 2.13: Sample video data for mutliple chirps

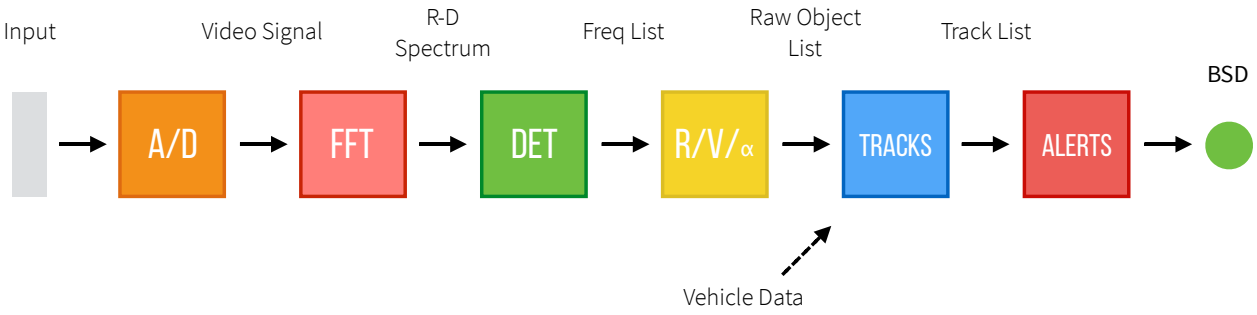


Figure 2.14: This digram simplifies the signal processing algorithm from receiving reflected signal as an input to triggering an alert at the end. The signal is at first discretized and processed using 2D FFT, which results in image-like structure called Range-Doppler spectrum. From the spectrum it is possible to obtain detections containing basic information about surrounding's objects. The detections are tracked using some sort of filter to get concrete objects. Just before the events can be evaluated, it is necessary to add vehicle data containing information about velocity and trajectory of the source vehicle. The final alerts are generated from associated events.

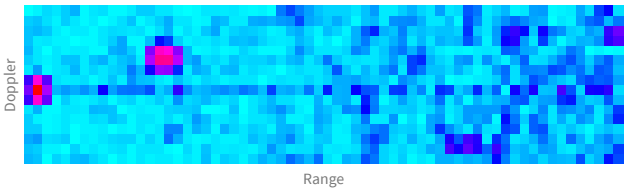


Figure 2.15: Example Range-Doppler spectrum frame. It shows aplitudes of received signal for every range and Doppler.

Next step is to detect peaks in the Range-Doppler spectrum, using algorithms like Constant false alarm rate (CFAR). Each peak represents a different reflection (object). The

reflection can be caused by many different objects, such as cars, trees, infrastructure and more. Sometimes the reflection can be bounced from other surfaces (usually infrastructure or large vehicles) in that case the object detected in this reflection is called a ghost-object. Detection of ghost-objects is a very hard problem as well as the classification of object type based on the reflection characteristic.

This is the last step where the beams are processed separately. Following steps take into account all beams to get the detailed information like velocity, position and possibly type of the object.

In the last step, all obtained information is evaluated and the important and relevant events are presented to the driver in a form of alerts. Basic events detected just using the radar sensor are Blind Spot Detection (BSD), Closing Vehicle Warning (CVW), Cross Traffic Alert (CTA).

2.2.3 Alert functions BSD and CVW

Common radar-based alert functions are BSD and CVW. These two alerts are detecting events of other vehicles approaching the blind spot zone. BSD alert is triggered if some other vehicle is located directly in the blind spot zone, whereas CVW gets triggered if other vehicle with its speed will arrived into blind spot zone in some time. This time is often called Time To Collision (TTC) and it is defined by requirements of the project. Blind spot zone is part of the space behind the vehicle as shown in Figure 2.16 and its dimensions and placement is also defined by project requirements. The Figure 2.16 is also illustrating all events when the BSD or CVW alerts will be triggered.

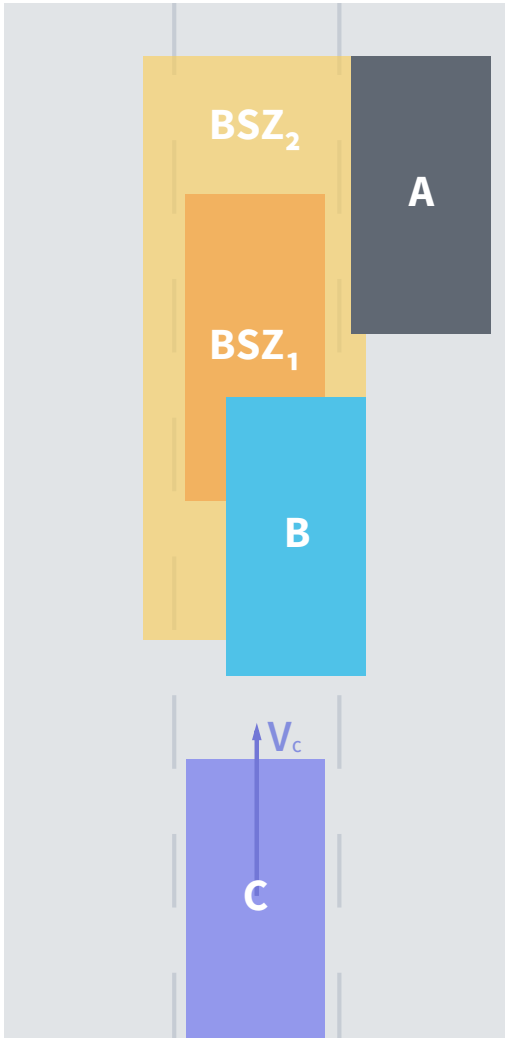


Figure 2.16: Blind spot of source vehicle **A** consists of two overlapping zones. Zone BSZ_1 representing space, where the BSD alert has to be triggered and BSZ_2 is space, where the BSD alert might be triggered, but the vehicle should be also visible by the driver, therefore, it is not required. The vehicle **B** illustrates the situation where the BSD alert will be triggered, because the vehicle is crossing BSZ_1. Vehicle **C** is showing vehicle approaching blind spot zone with higher speed and therefore, the CVW alert should be triggered. The CVW depends on time to collision.

Chapter 3

Implementation

The architecture of Artificial Neural Networks (ANNs), which is proposed at the end of this chapter, depends on a few tasks, at first it was necessary to decide what part of the current algorithm is possible to replace with an ANN and therefore, what should be the input. Based on availability of training datasets, similar decision was made for choosing the right output. Next in this chapter is described the format of the dataset as well as the file format used for storing all the datasets.

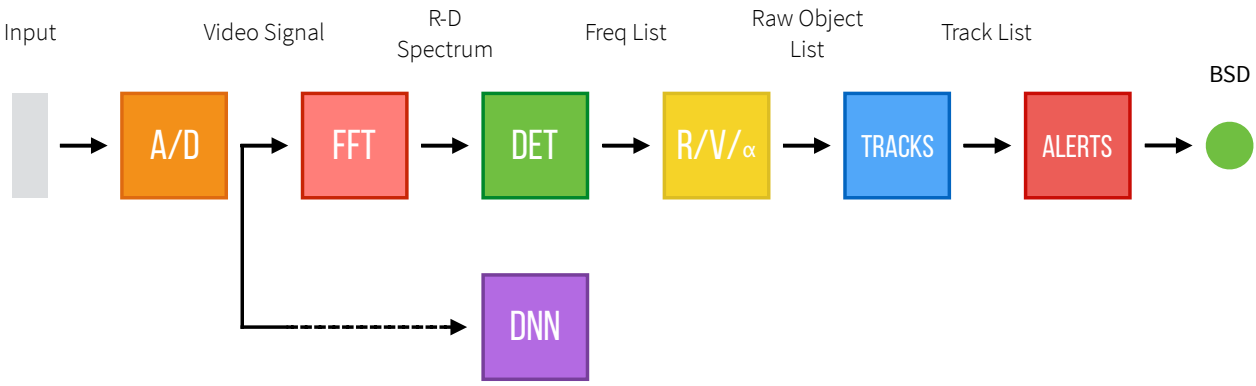
3.1 Choosing input data

Choosing the right input data is always a demanding and intricate task. Neural networks are capable of learning on arbitrary data, but the performance can be significantly worse if the network needs to deduce some information, which can be added as a preprocessing. In my case, I was considering two different approaches:

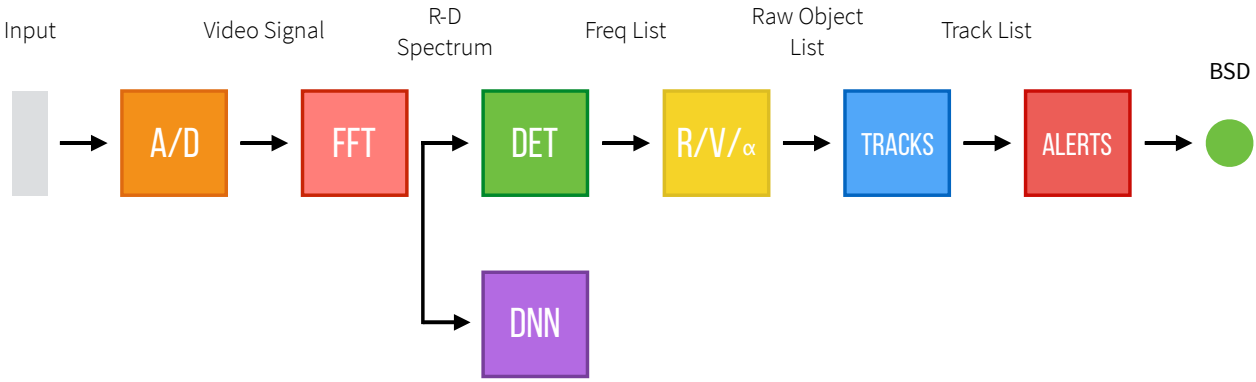
1. Using **video data** as an input to neural network (See figure 3.1a). This will enable the network to see the raw data without any preprocessing. Whole frame is needed for successfully evaluating any event. Each frame consists from multiple beams containing multiple video data per chirp. The video data can be interpreted as a time-series a fixed length. Therefore, the network would need to be able to handle input of multiple time-series, which are slightly delayed in time, because of the time delay between chirps. In addition to that all useful information are hidden in frequency domain such as velocity (Doppler effect). It would be hard for the network to extract all of this information.

2. Using **Range-Doppler spectrum** as an input (See figure 3.1b). Usually it is not desirable to use any kind of complicated preprocessing, which can add noise or inaccuracies to the data, whereas normalization or transformation is commonly used form

of preprocessing. Therefore, performing 2D Fourier transform, which actually extracts hidden information that should be deduced from the network anyway and not destroy anything else that can be useful.



(a) Video data



(b) Range-Doppler spectrum

Figure 3.1: Different types of input to a neural network

Because Range-Doppler spectrum is widely used for FMCW radar based signal processing and it is more human readable I decided to use it as an input. Thus I can easily examine the input data and analyze the network behavior. At the same time it allows me to use convolutional layers because of the data's structure.

3.2 Output of the network

The main task is detecting objects in radar data, therefore, obvious choice of output data would be **Track list**, list of track objects used for generating alerts (See figure 3.2). This way the network is generic and can be used for any kind of alert or any kind of future applications (for example 'sensor fusion').

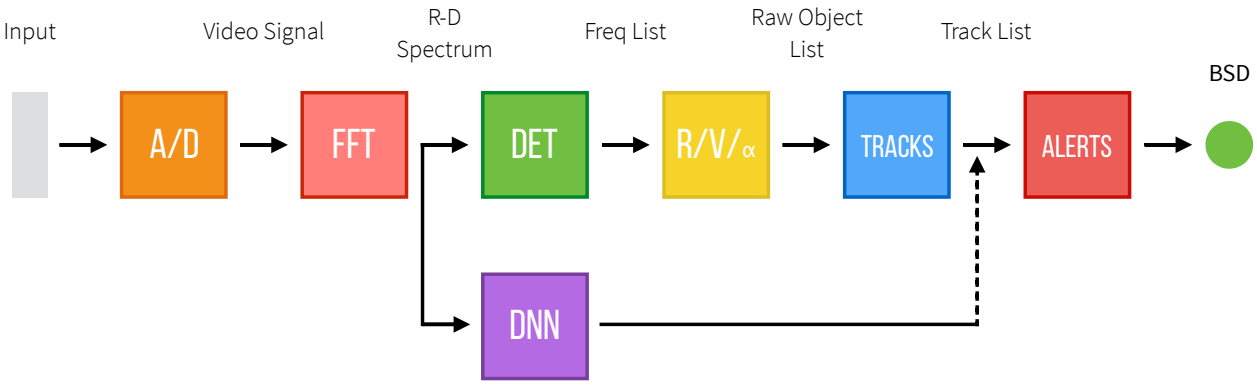


Figure 3.2: Optimal output.

To train the network to output multiple object detections can be achieved by doing either semantic segmentation [25, 29], Single Shot MultiBox Detector (SSD) [24] or similar. Unfortunately for any of these approaches I did not have training data.

For simplification of the task I chose to detect one specific alert. Data I had for the training was used for testing and verification of Blind Spot Detection (BSD) and Closing Vehicle Warning (CVW) functions and because CVW is highly dependent on the trajectory of the source vehicle, which means that I would need to include vehicle data (yaw rate, velocity) into the network, I chose to use BSD. The final proposed approach is shown in the figure 3.3.

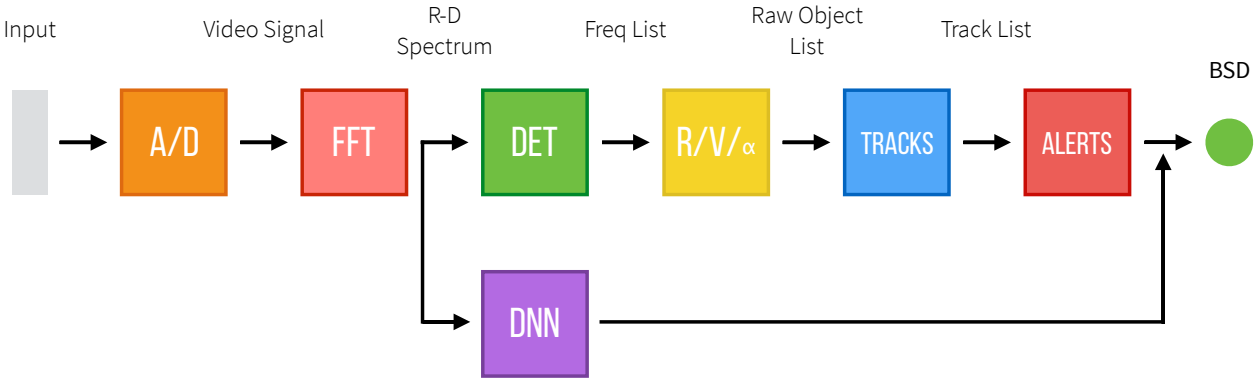


Figure 3.3: Final configuration.

3.3 File format

All experiments were done on a Linux system (more in Section 3.7), therefore, it was desirable to maintain compatibility with Linux. However, Valeo is using a proprietary file format for recording and storing all raw driving data. The parser for that format is included in an internal tool called 'Online tool' written in C++, but because it highly depends on

Windows environment, it would be complicated to rewrite it for multi-platform usage, similarly it would be complicated to separate the parser from the tool. For that reason, it was necessary to convert data into more convenient file format, which meant to implement an exporting module directly to the Online tool, so the data can be easily exported and used on any other system. There are multiple file formats that could be used for storing all datasets. The criteria the right file format should met are:

Linux / Windows compatible - parser has to exists for both platforms

Easy to read in Python - all the experiments are implemented in Python, therefore, is important to have easy way to read the file in it

Easy to write in C++ - because the Online tool is written in C++, I need to be able to write to the file from it

Compressed - training deep neural networks needs hundreds of thousands of images, so it is convenient if the file can be compressed as much as possible

Online access - the file is expected to be huge ($>$ 200 GB) so it would not be possible to fit the whole file into the memory, therefore, it is necessary have a format which can be read sequentially directly from hard drive

Easy to implement - I did not want too spend much time with integration of the file format

I was considering following options:

Comma Separated Values (CSV) :

- easy to write in C++ and easy to read in Python
- because CSV is a text file, it is human readable, but it also takes more memory
- can be loaded sequentially, but that would require a customized implementation

Pickle :

- binary format used for serializing objects in Python, easy to read and write in Python, but hardly accessible from any other programming language
- whole file has to loaded into the memory

Custom binary format :

- for each frame, we need to store couple of images and labels, therefore, it would not be so difficult to create a custom binary format
- flexible - contains anything I want
- hard to implement

HDF5 :

- file system inside a file, allows to use many different types of objects such as datasets, groups, links and attributes, which means I can better organize all the data inside datasets [38]
- all datasets can be automatically compressed during writing
- data are lazy read from the file in a moment when they are needed
- HDF5 is specially designed for storing large amount of data

From these options, HDF5 was an obvious choice even though I was aware of the integration to the Online tool will be harder. The Online tool is already able to export bunch of useful data, but in CSV format. At first I therefore, chose to work with HDF5 just inside Python and integrate it as a cache (preprocessed data) for the datasets. Each dataset is created directly from exported data (from CSV file) and saved into HDF5 file. Whenever then dataset needs to be loaded, Python automatically loads already processed datasets from HDF5 instead of recreating it again from the original data. This way the implementation of HDF5 is independent between Python and C++. At the end I have decided to implement exporting to HDF5 for the Online tool, because CSV was too slow and too memory intensive. But the Python implementation did not have to be changed, because the data were in the same form.

3.4 Dataset

Single dataset is represented by a single HDF5 file. It contains a single drive separated into batches - during the recording of a drive, the data are split to separate 1 GB files. The architecture of the dataset is shown in Figure 3.4.

Also the HDF5 file is allowing to save useful attributes with each dataset or group, which I take advantage of for saving some statistics and indices for faster filtering of the data. Dataset contains all RDMatrices for all possible sides of radar placements (rear left and rear right in my case) ordered by MCC. The labels are obtained by current version of signal processing algorithm, which means there is no guarantee of correctness of the labels. Dataset

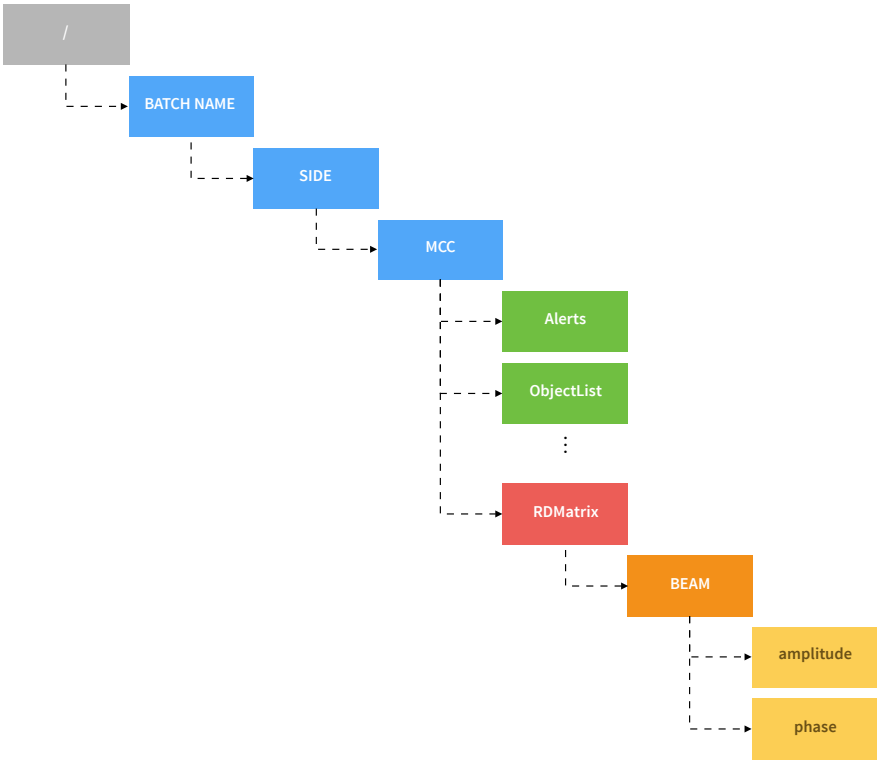


Figure 3.4: Architecture of a dataset

has to be fixed for one physical vehicle, because of different placement of the sensor, which is calibrated per vehicle and system requirements defined per project.

3.4.1 Range-Doppler Matrix

All values in the Range-Doppler matrices (RDMatrices) exported from Online tool are represented by complex numbers in standard form:

$$z(a, b) = a + bi \tag{3.1}$$

Which is fine and it can be also used as representation of an input, however, for better understanding of the input data it is convenient to transform the complex number from standard form to polar form:

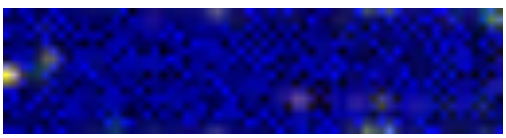
$$z(r, \varphi) = r \cdot e^{i\varphi} \tag{3.2}$$

This way, all the values are represented by amplitudes r and phases φ, which can be written as two matrices R containing just the amplitudes and Φ containing the phases. That means, the RDMatrix is a volume with dimensions $W \times H \times 2$.

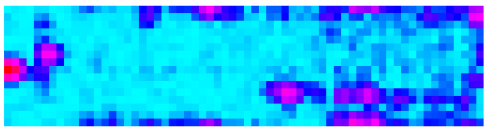
As it was already mentioned Valeo is using phase interferometry with two antennas as is shown in Figure 2.11 to get an estimate of an angle. Therefore, each frame for every

beam and side contains two RDMatrices - one for each antenna. At first the RDMatrices are transformed into set of $\{R_1, \Phi_1, R_2, \Phi_2\}$, which can be already used as an input volume of dimensions $(W \times H \times 4)$ into the Convolutional Neural Network (CNN). However, to simplify the problem for the network, it is possible to reduce the number of dimensions by calculating a phase difference directly. Therefore, the final input volume is $\{R_1, R_2, \Delta\Phi = \Phi_1 - \Phi_2\}$ with dimensions $W \times H \times 3$.

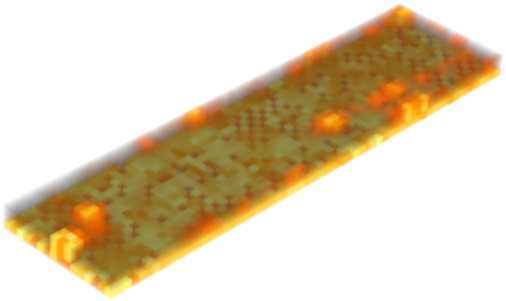
Using that structure RDMatrix can be visualized by slices through the last dimension such as R_1, which is shown in Figure 3.5b or as 3D image as is shown in Figure 3.5c, where all the layers $\{R_1, R_2, \Delta\Phi\}$ are distinguishable. But it is also possible to encode all slices as color channels for example R, G, B and visualize RDMatrix as a color image as is shown in Figure 3.5a.



(a) RGB image



(b) R_1 from RDMatrix



(c) 3D image

Figure 3.5: Different type of visualization of RDMatrix

Visualization of the RDMatrix as RGB image (as is shown in Figure 3.5a) allows to see an interesting phenomenons. Normally if the reflection is detected by both antennas, it should be visualized as a yellow-colored blob, because amplitudes from the R_1 are illustrated with the red color and amplitudes from the R_2 are using green color. However, sometimes the color of the blob is more close to the green or more close to the red, which means the signal was not received by both antennas with the same power.

3.5 Architecture of neural network

Designing a working architecture for given task can be challenging. It is really hard to define what the "working" architecture is, because the learning quality depends mostly on

training and validation data. As I have mentioned in Section 3.1, the input to the network is RDMatrix (Section 3.4.1), which can be represented as RGB image. That allowed me to get inspired by architectures used for image classification, such as AlexNet [21], VGG16 [33] and others.

Because I choose to detect just a single alert (Section 3.2, the task is simplify to binary classification problem. However, all alerts are time dependent, which means that alert function does not depend on current single RDMatrix (image) but it depends on sequence of previous RDMatrices. Mainly because the alert function is defined with hysteresis, the alert should stay on for a few seconds after the target car leaves the BSD zone, which is one of the system requirements. It is not possible to avoid this hysteresis, because it is already contained in the training data.

With that in mind I have decided to try simple approach with a single RDMatrix at first, to verify if the network is able to somehow fit the data. The general architecture is shown in Figure 3.6, this is overview of a typical use architecture that allowed me to simply switch between single RDMatrix input and sequence of RDMatrices in the future, just by replacing the Multi-Layer Perceptron (MLP) with some Recurrent Neural Network (RNN). The hardest part was to deal with multiple beams.

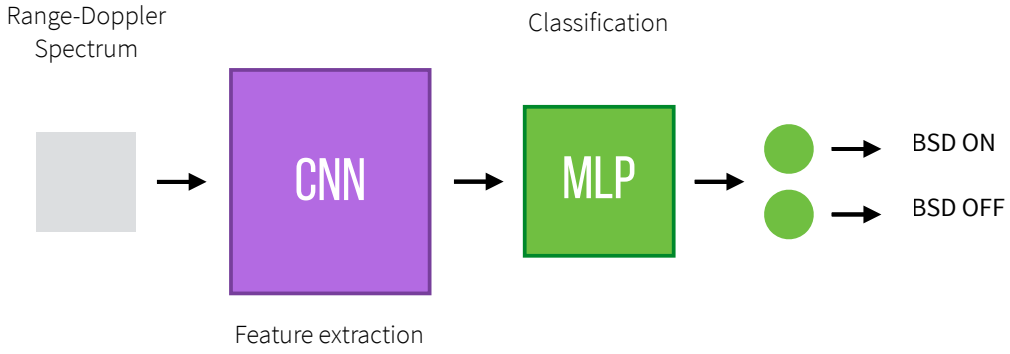


Figure 3.6: General architecture of CNN

3.5.1 Multiple beams

Because the radar outputs four RDMatrices each frame (MCC), it was necessary to find a way to process those matrices. The distribution of the beams is illustrated in Figure 3.7. Even if the energy is focused to narrower space, some smaller amount of the energy is still emitted to the whole field of view, that means I would received less energy, if the target object is outside of the beam, but I would be potentially able to detect the object. But because of this property it is possible to stack all RDMatrices on top of each other, to create a large input volume into the CNN. This is visualized in Figure 3.8. Only problem here is

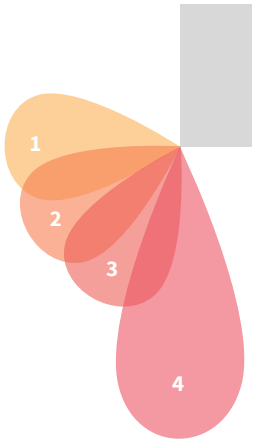


Figure 3.7: Diagram illustrating spatial orientation of multiple beams

different dimensions of the RDMatrices across the beams, I would need to crop the larger matrices or zero-pad the smaller ones.

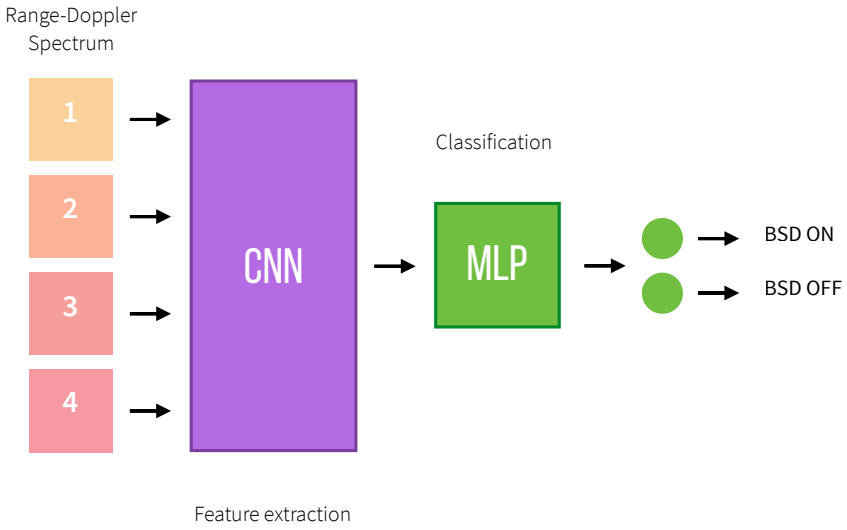


Figure 3.8: Architecture with one large CNN

Possibly a better approach is to create separate CNNs each one for extracting features from different beams. The first advantage is that there is no need for zero-padding or cropping the input data. But most importantly every beam is a representation of the environment from a different angle, that means some reflection may appear differently in some beams, because even if the reflection can come from the same object, it does not need to come from the same surface. This is easy to imagine with four different cameras pointing to different directions. Camera pointing rear (beam number 4) will almost always see other vehicles from the front, whereas the side camera (beam number 1) will mostly see vehicles from the side. It would make sense to create separate image processing for these cameras, because the training data will be different and even the features, which should be learned by the CNN

will differ. General idea of the multi-CNNs is illustrated in Figure 3.9.

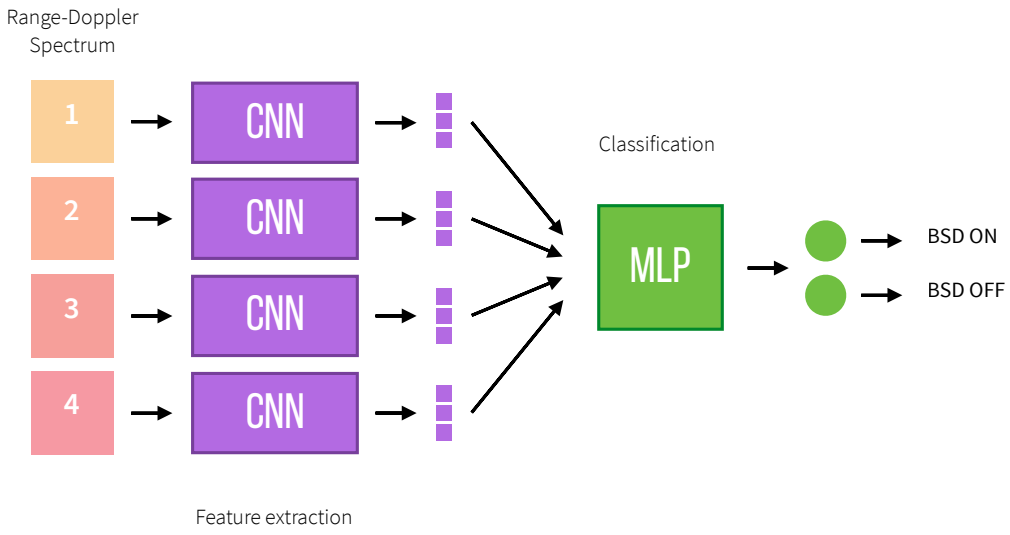


Figure 3.9: Architecture with multiple CNN

At the same time this approach allows me to develop one single CNN for one beam and add other beams later. From the orientation of the beams (illustrated in Figure 3.7) it is evident that some beams are not covering BSD zone very well. Beams number 2 and 3 seems to best cover the BSD zone, I have decided to use beam number 3. Other beams can be more suitable for other alerts, for example rear-facing beam number 4 could be used for CVW or detecting rear collision and beam number 1 can be used for detecting the infrastructure, or side collision.

3.5.2 Final architecture of Convolutional Neural Network

For the experiments I'm using two different architectures of CNN, one rather simple with 4 convolutional layers and *VGG16* for comparison. Both of them contains just 4 basic layers:

- **Convolutional** layer: $\{K, F, S, P\}$
- **Max-Pooling** layer: $\{O, F, S\}$
- **ReLU** layer
- **Batch Normalization** layer

The simple architecture contains the following:

- **Convolutional** layer $\{32, 3 \times 3, 1 \times 1, 1 \times 1\}$

- **ReLU** layer
- **Max-Pooling** layer $\{max, 2 \times 2, 2 \times 2\}$
- **Batch Normalization** layer
- **Convolutional** layer $\{32, 3 \times 3, 1 \times 1, 1 \times 1\}$
- **ReLU** layer
- **Max-Pooling** layer $\{max, 2 \times 2, 2 \times 2\}$
- **Batch Normalization** layer
- **Convolutional** layer $\{32, 3 \times 3, 1 \times 1, 1 \times 1\}$
- **ReLU** layer
- **Max-Pooling** layer $\{max, 2 \times 2, 2 \times 2\}$
- **Batch Normalization** layer
- **Convolutional** layer $\{32, 2 \times 2, 1 \times 1, 1 \times 1\}$
- **ReLU** layer
- **Max-Pooling** layer $\{max, 2 \times 2, 2 \times 2\}$
- **Batch Normalization** layer

3.5.3 Architecture of classifier

As I mentioned above, the CNN is used for feature extraction and its architecture is described above. For the classifier itself I used simple MLP with two **Dense** layers as follows:

- **Dense** layer - with 32 neurons
- **ReLU** layer
- **Batch Normalization** layer
- **Dense** layer - with 1 neuron as output
- **Sigmoid** layer

However, I was also playing with the thought, that RNN could be better, because it can take into account previous frames, so I designed a simple RNN. For that I was inspired by work of Koutnik et al. [20] with RNN controlling the car inside the TORCS racing simulator. They were using Long-Short Term Memory (LSTM) as recurrent layers right after CNN. But instead of neuroevolution I am using simple backpropagation. Architecture of my LSTM network is simple:

- **LSTM** layer - with 128 units
- **LSTM** layer - with 128 units
- **Dense** layer - with 1 neuron as output
- **Sigmoid** layer

3.6 Neural Network library

There are plenty of different neural networks frameworks I can use. I wanted to work in Python, which gave me most options. The most commonly used frameworks are *Theano* [2] and *TensorFlow* [1]. Both of them are powerful but really low-level. For fast experimenting there are wrappers, which simplify the syntax. There is a lightweight wrapper around Theano library called *Lasagne* [7], which allows to define a network as a sequence of layers per line, implementing the CNN is really simple. However, I have decided to use similar but newer library called *Keras* [4]. Keras is high-level library with minimalistic syntax working on top of Theano, but also on top of TensorFlow. Switching between Theano and TensorFlow is simple and it can be done right before running an experiment, which is convenient, because for some models one framework could be better then the other.

3.7 Hardware

To make the training time of the model as short as possible, I am using Python neural network library with support of a CUDA and cuDNN. To benefit from this I needed CUDA-capable Graphics Processing Unit (GPU). For that reason I built a custom computer with Nvidia Tesla K40, needed especially because it offers 12 GB of RAM, which I used for storing the model and data with batch size of 128. For quick access to disk space I was using 256 GB SSD with M.2 slot, that was used mainly for caching during creation of the datasets, because it was able to reach write speed around $1\,\mathrm{GB\,s^{-1}}$ and read speed around $2\,\mathrm{GB\,s^{-1}}$.

For archiving all the experiments (storing the weights of the networks and all results) as well as storing the datasets I had connected a external 1 TB SSD. The Central Processing Unit (CPU) used is an Intel i7-6800K with 6 cores and 12 threads and 3.4 GHz overclocked to 4.1 GHz.

All the neural network libraries are usually developed on Linux system and therefore, they offer best support for Linux-based operating systems (especially Ubuntu distribution). It is possible to set them on other environments, but its typically much harder work. To make the setup working as fast as possible, I used Ubuntu Linux 16.04 as the operating system, which enables me quickly configure all settings.

Chapter 4

Experiments

The following experiments were done on three different models and real datasets. To find the best training data is always a challenging task, I had access to two different sets of data recorded in real world conditions.

4.1 Datasets

I had two types of data: **Open Road Test**, which was recorded from long distance drive on public roads across the Czech Republic and **functional test** containing specific scenarios recorded in a controlled environment on a test track.

4.1.1 Open Road Test

The Open Road Test contains almost 1.74 millions frames of highways driving split into 7 drives. The statistics of this datasets are shown in Table 4.1. Each drive is quite long with average length of 247 987 frames, but because all the drives were recorded in natural environment they do not contain many alerts. There is less then 4.4 % (76 000 frames) of positive data in total (visualized per drive in Figure 4.1). All labels are calculated from the current version of Blind Spot Detection (BSD) algorithm, which is not perfect yet, also it contains hysteresis, which, for static frame classification, will be evaluated as false positive error, because the object is no longer visible in the frame. All of these flaws can make the training hard and slow. Therefore, I have decided to not use this dataset at the end.

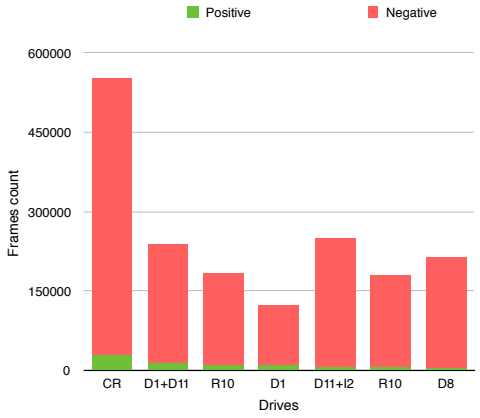


Figure 4.1: Number of frames per scenario in 2 000km drive

Table 4.1: Statistics for Open Road Test

	Total	Positives	Negatives	Pos. / total	Neg. / total
Total	1 735 909	75 977	1 659 932	4.38%	95.62%
Mean	247 987	10 854	237 133	4.43%	95.57%
Median	212 968	9 163	210 079	5.06%	94.94%
Std	140 008	8 131	132 911	2.00%	2.00%

4.1.2 Functional test

The scenarios in functional test drives are much more simple and better controllable. The labels were calculated almost correctly, because there is less noise from infrastructure and other objects. There is 84 scenarios per side, which gives around 572 000 frames in total, exact statistics is shown in Table 4.2. Just 6 % ($\sim$ 35 000) are positive samples, which is more then in the case of Open Road Test. Distribution (per scenario) of positive and negative frames is plotted in Figure 4.2. Exactly 10 scenarios do not contain any positive samples and almost 20 scenarios have less then 1 % of positive samples. However, 14 scenarios consist of more then 10 % of positive samples. Because the scenarios are short (6 974 frames in average) I do not need to split them to separate training and testing dataset, instead, I can just assign specific drives to different datasets. As a validation dataset I am using part of each drive assigned to testing dataset, but the test is perform on all drives separately.

Table 4.2: Statistics for Functional test

	Total	Positive	Negative	Pos. / Total	Neg. / Total
Total	571 862	34 865	536 997	6.10%	93.90%
Mean	6 974	425	6 549	6.42%	93.58%
Median	6 995	192	6 759	2.77%	97.23%
Std	2 120	589	2 155	9.76%	9.76%

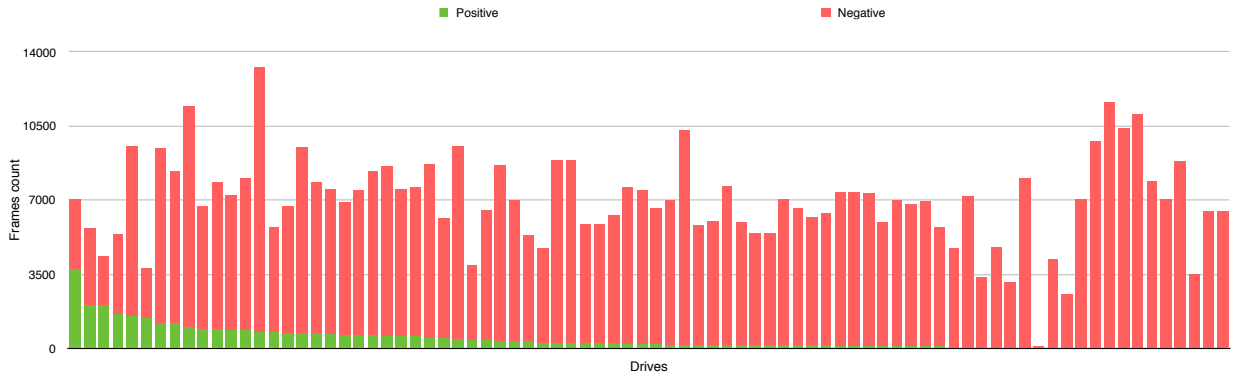


Figure 4.2: Number of frames per scenario in functional test drives

Table 4.3: Datasets used for models

	CNN + MLP	CNN + LSTM	VGG16
Num. of drives	12	12	15
Negatives	62 440	62 440	86 093
Positives	12 593	12 593	15 512
Training	52 519	52 519	71 117
Validation	22 514	22 514	30 488

4.2 Models

All experiments were done with three models, which were described in Section 3.5. For static frame classification I have used simple **Convolutional Neural Network (CNN) + Multi-Layer Perceptron (MLP)** (model A) and **VGG16** (model C) for comparison. At the same time I also tried to train the network for sequence classification using **CNN + Long-Short Term Memory (LSTM)** (model B).

I was experimenting with different configurations for different models, but in the end I stuck with following (shown in Table 4.3): Because the VGG16 is slightly deeper model I assigned more drives to the training set. I would use more, but I have had a problem with fitting all the data into memory. The parameters of the training are shown in Table 4.4, I have tried to play with different parameters values, but this is what was showing biggest potential. As the optimizer I have used **adamax** [18], because it is much more memory efficient. It worth noting that, because the training dataset is not well-balanced I assigned different weights to individual classes. The BSD alert is counted as 100 times more.

Table 4.4: Parameters of models

	CNN + MLP	CNN + LSTM	VGG16
Batch Size	128	128	128
Learning Rate	0.1	0.1	0.1
Num. of epochs	100	200	100
Class weights - 0	1	1	1
Class weights - 1	100	100	100
Num. of params	28 001	287 073	600 545
Shuffle	Yes	No	Yes

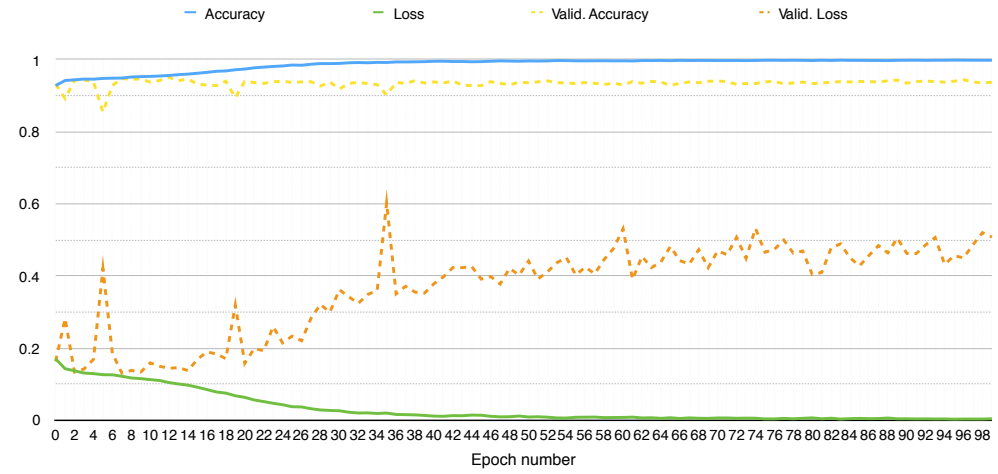
4.3 Results

From all experiments I made, I chose one that looks most promising. However, all experiments were quite similar and were expressing similar behaviors. From the training progress (shown in Figure 4.3) it is evident that first two models (A, B) are already overfitting the training data right after few epochs, whereas model C is barely able to reach to 95 % of training accuracy. The maximal validation accuracy is quite low, for models A and B it is slightly over 93 %. To put it into the perspective I have defined *beaten* criterion for each dataset. The beaten criterion tells what should be the accuracy of the model, so the model beats the *zero-model*, where zero-model is model, which always predict zeros. It is really conservative, but it can be used as a lower bound for measuring the model performance. The median of beaten criterion is 97.23 % among all drives, which means that no model should be able to beat the zero-model.

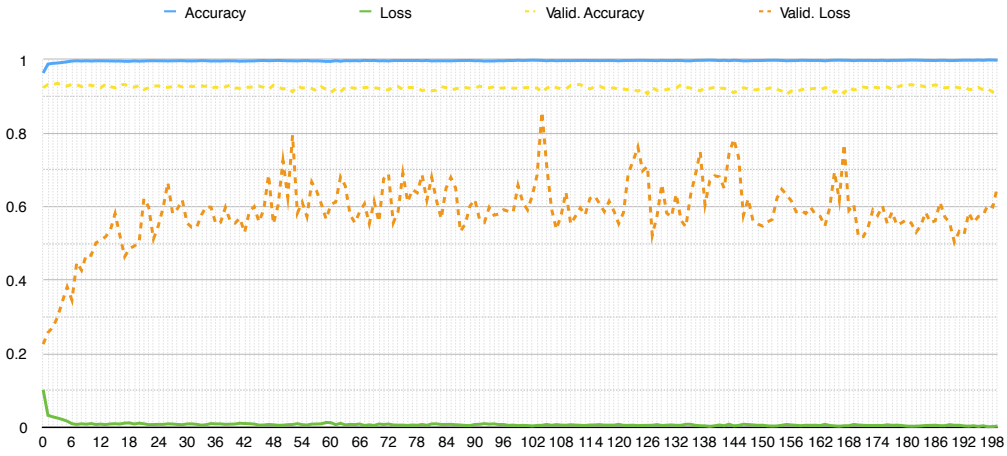
However, because the accuracy is measured on validation dataset, which is build from parts of drives, it is hard to compare these values. Therefore, I actually run all models on all data and let them predict the alerts. Then I was able to calculate how many drives the models have beaten, the result is shown in Figure 4.4. The model C was not able to beat any of the drives, which was expected from looking at the training progress. On the other hand, both model A and B have been able to beat more then 50 drives (from 84). There is around 20 of drives with no or small amount of alerts, the beaten criterion for these drives will be high, around 100 %, which is hard to beat, that is the reason why the score is so low, because all models have problems with these drives.

Despite the fact, that all models are overfitting and the training progress does not look very promising, the predictions look good. Figure 4.6 is presenting predictions for one sample drive from the testing dataset.

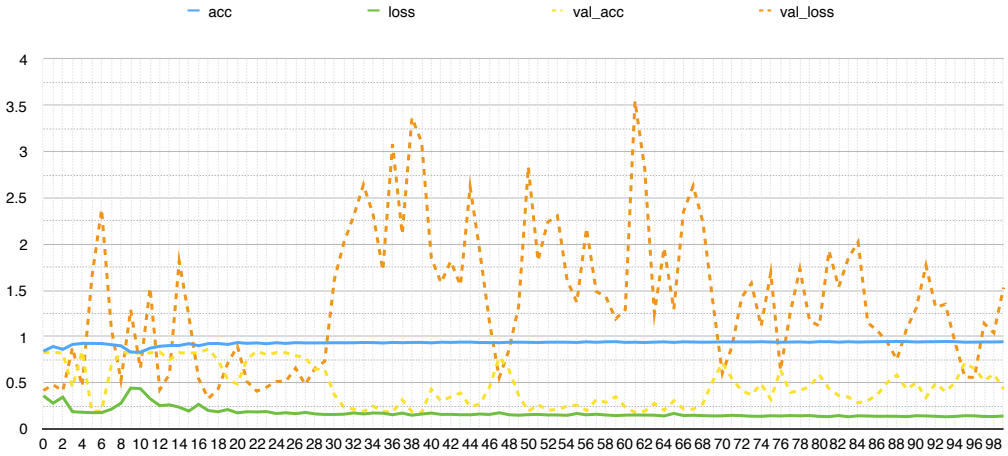
The model A (Figure 4.6a) appears to have best predictions, it has the smallest amount of false positives, but the predicted alert is not held during whole time of the original alert. This can be caused because model A is a static model and the reflection can sometime disappear.



(a) CNN + MLP



(b) CNN + LSTM



(c) VGG16

Figure 4.3: Progress of training different models

This problem can be handled by providing some basic filtering e.g., sliding average. That would help to fill the gaps in the alert, but can of course caused more false positives. The

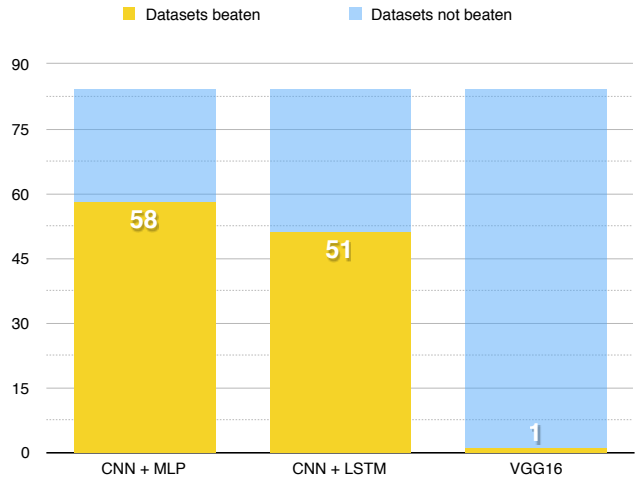


Figure 4.4: Comparison of beaten ratio between different models

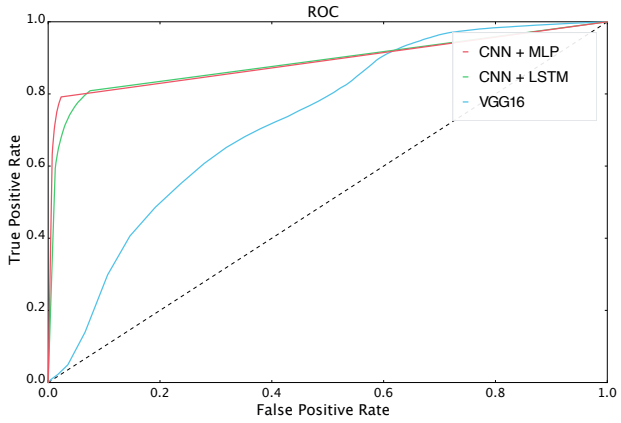


Figure 4.5: ROC for all models across all drives

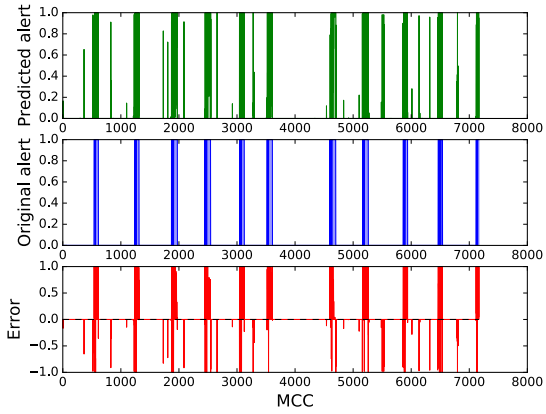
detail illustrated in Figure 4.6b is showing the worst predicted alert. There is visible the gap in the original alert, which would be filled with the filter.

Model B (Figure 4.6c) represents the only one model with internal memory and therefore, is should be able to hold the alert. It is obvious from the plots that this is not happening, at least not in the testing dataset. The reason could be in higher complexity of the model and therefore, in the insufficient size of training dataset or because the training dataset is not diverse enough to allow model generalize. However, it seems that model B suffers from the same problem as model A with the false positives, actually the problem is even more striking.

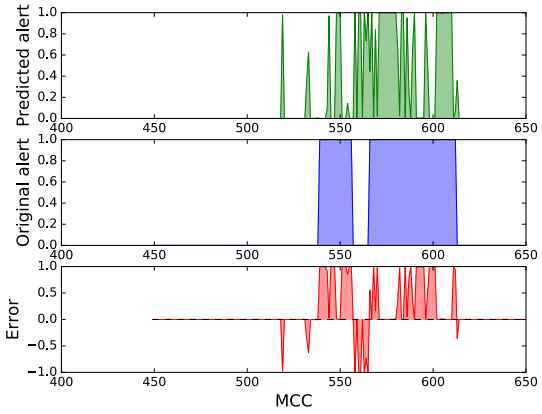
Regarding model C (Figure 4.6e), it is easy to say that model C was not able to learn on provided data. However, it is quite interesting, despite the fact it is almost always predicting the alert, it learned to turn off the alert some time after the original alert. This behavior is clear in the detail (Figure 4.6f), where it is also evident, that the model actually holds the

alert for almost the right amount of time and oscillate otherwise.

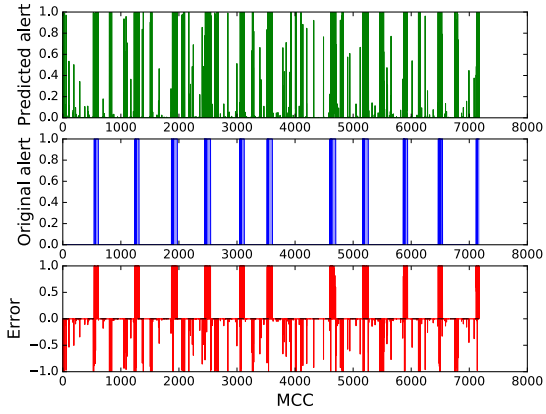
For better imagination of the quality of each model I have also calculated Receiver Operating Characteristic (ROC) curve across all drives (including drives assigned to training dataset). The ROC curve is shown in Figure 4.5. No model is perfect according to ROC, but models A and B are showing better results for this specific problem on this data, whereas results of model C are really bad.



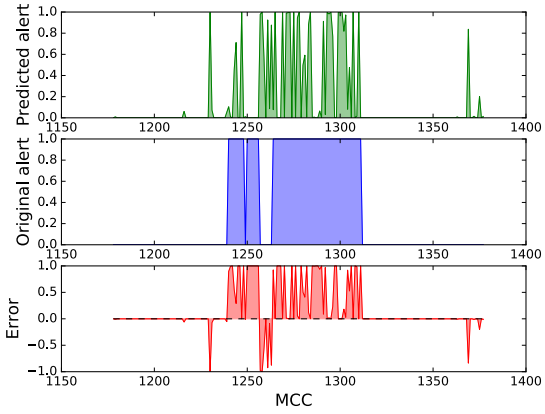
(a) CNN + MLP



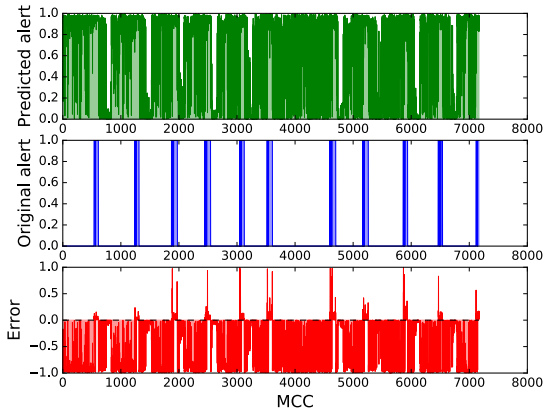
(b) CNN + MLP (detail)



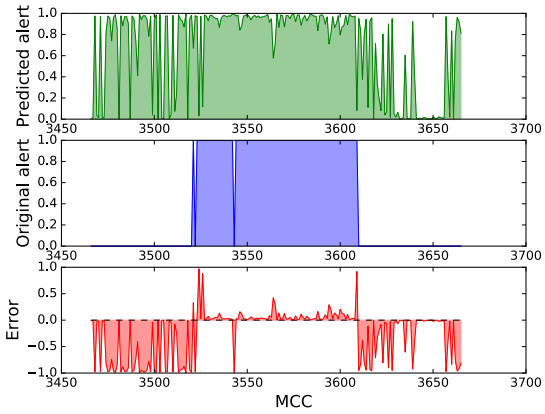
(c) CNN + LSTM



(d) CNN + LSTM (detail)



(e) VGG16



(f) VGG16 (detail)

Figure 4.6: Predictions of different models on one drive from testing dataset. There is always a example of full drive and detail, which illustrates the worst predicted alert. Prediction from the model is always a number in the range from 0 to 1, which can be interpreted as probability of an alert. The bottom part of each plot shows difference between predicted and original values.

Chapter 5

Discussion and future work

Results look promising, they prove that the task is solvable using neural networks. The fast overfitting, which was the issue of models A and B, means that the model is able to learn the data, but it is not able to generalize, which means that the model is too complex (in a sense of trained parameters) and it needs more data. Personally I would try to reduce the complexity of the model by removing some layers and try to use the same data. There could be also to many similar frames and therefore, the dataset is not diverse enough. That could be happening because of the high frame rate and low range resolution. The radar records around 20 Frames Per Second (FPS) and its range resolution is around 75 cm. The target car would need to travel less then 75 cm between two consecutive frames to be indistinguishable. Lets assume the target vehicle's relative velocity is 75 cm/frame, which is the lowest velocity the vehicle can drive so the frames are changing:

$$v_{target} = (\text{radar}_{\text{resolution}}) \cdot (\text{radar}_{\text{FPS}}) = 0.75 \cdot 20 = 15\text{m}\,\text{s}^{-1} = 54\text{km}\,\text{h}^{-1} \tag{5.1}$$

It is safe to assume that target vehicle will have velocity close to the source vehicle. Relative velocities on the highways can easily reach up to $40\,\text{km}\,\text{h}^{-1}$, therefore, to achieve a velocity around v_{target} is quite easy. Because everything slower then v_{target} will produce similar consecutive frames it should happen quite often. However, I did not notice this phenomenon in the data, in fact I actually noticed the RDMatrix is constantly changing without any duplicated frames, it can be observed in the Figure 5.1. From the same Figure, it is obvious that the frames are changing, but sometimes the reflection, which is clearly recognizable on one frame suddenly disappears in the next frame. That could be caused by power fluctuation of reflected signal [36]. These fluctuations are described on the diagram of the Radar Cross-Section (RCS) (example diagram is shown in Figure 5.2), which is demonstrating how the reception field strength is changing by moving the object against the radar.

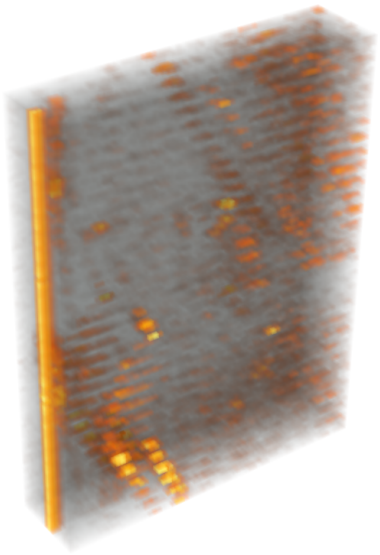


Figure 5.1: Visualization of data with two alerts from Open Road Test drive. The 3D image shows two concatenated sequences with alert triggered. Pixels opacity is calculated by amplitude of the received signal. The vertical axis represents the time, the short horizontal axis represents Doppler (relative speed) with the zero in the middle and the long horizontal axis represents range. Because radar can see it self, all frames contains detection when Doppler and range equal zero.

It is clear that large changes in a power can be caused by small changes in the object's state.

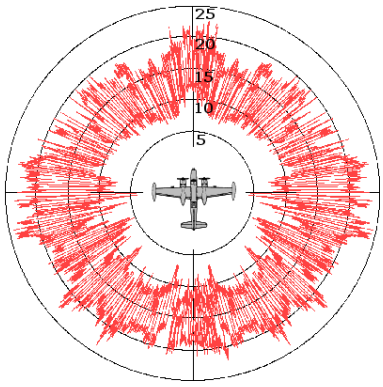
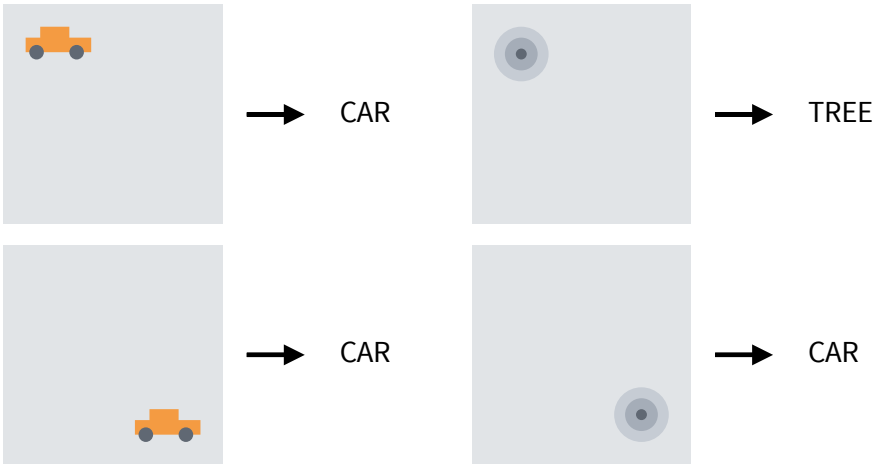


Figure 5.2: Typical RCS diagram of a plane [42]. Tho power is illustrated by the red curve - closer to the center, lower the power.

Next issue with the dataset are the labels. The labels are obtained automatically using the current version of the algorithm, which is already incorporating all system requirements. For the purpose of creating generic model, it would be necessary to have pure labels, not loaded by any other information that can be done in post-processing.

In the Figure 5.3 I am trying to address another issue with the RDMatrix. Since beginning I was assuming I can work with RDMatrix in similar fashion as with Red Green Blue (RGB)

image, however, I missed one important image's property - **stationarity** - the class of an object is translation invariant, that means the object is still the same wherever is placed in the image. Whereas RDMatrix does not share this property. Because class of an object in RDMatrix strongly depends on relative velocity represented by Doppler axis.



(a) Image (b) RDM

Figure 5.3: Illustration of stationarity property for image and RDM

Possible solution for this issue can be transformation of RDMatrix into different space, for example by cutting across Doppler axis - discretize the Doppler values. This proposed solution is illustrated in Figure 5.4. The RDMatrix for one antenna can be imagine as 4-dimensional space $\{R, D, A, \varphi\}$, where: R is range, D is Doppler, A is amplitude and φ is a phase. By discretization of the Doppler dimension the original space will be reduced into a 3-dimensional space, which can be used as an input. However, the model would need to be able to handle multiple inputs for each discretized Doppler's value.

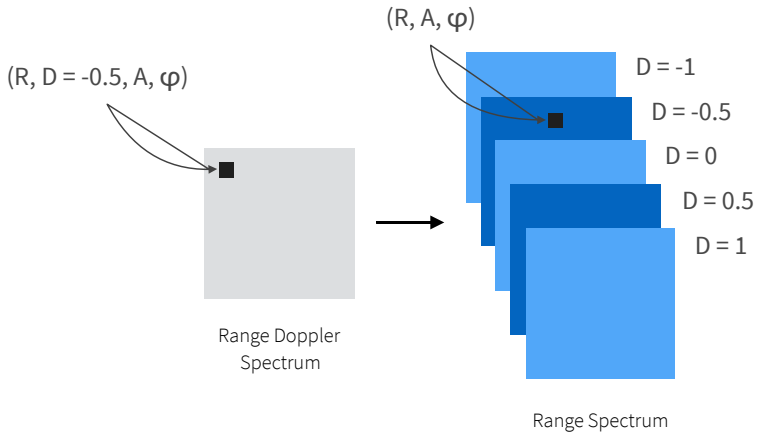


Figure 5.4: Possible solution for the stationarity issue

In the Section 3.2 I have mentioned the track list is an optimal output, which is a list of detected objects. One of the proposed approaches to detect multiple object is segmentation or Single Shot MultiBox Detector (SSD). However, for any of these methods, it is necessary to have correctly labeled data. Gathering the training data is always the problem, for that purpose I would suggest to implement a simulation and generate artificial data, because obtaining correct labels would be hard without any automatic labeling system. Another solution would be to attach some more precise system such as Light Detection And Ranging (LiDAR) based system to the vehicle and use its produced data to create labels. However, all of these approaches are beyond the scope of this work, but should be considered for future work.

Chapter 6

Conclusion

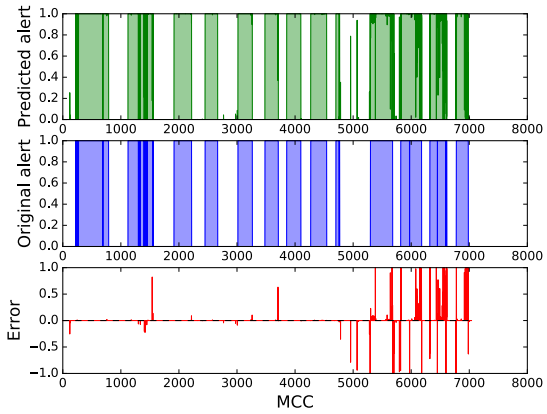
Main purpose of this work was to investigate neural networks based approaches for replicating the Advanced Driver Assistance System (ADAS) alert functions, using exclusively radar sensors. I have performed an analysis of input data and current signal processing algorithm to estimate the part of the current algorithm that would be convenient to replace with neural networks. I have decided to simplify the task by proposing to use neural networks for replication of only the Blind Spot Detection (BSD) alert function. The proposed architecture of neural networks was inspired by architectures winning the ImageNet competition in image classification [21, 32]. To obtain data I have to implement an exporting module into internal tool used in Valeo. This module provide me a HDF5 file, which I was able to process in Python. I have implemented Python library, which processes the HDF5 file and creates all datasets. To implement proposed architectures of neural networks I have used open source neural networks library called Keras [4].

The results suggest that it is possible to use neural networks for processing radar data and their classification based on the extracted features from Convolutional Neural Network (CNN). But the classification happens just by analyzing one single RDMatrix, which caused some problems, especially because the reflected power highly fluctuates and therefore, the object sometime disappear from the RDMatrix. The radar sensor is outputting a continuous stream of RDMatrices. To take that into account, I also implemented a recurrent version of classifier using Long-Short Term Memory (LSTM) layers. However, this approach did not show any significant improvement, probably because of data not being perfectly diverse, well balanced or there might not be enough data at all. To compare proposed models, I also implemented a network VGG16 [33], which won ImageNet in 2014. VGG16 model contains more trained parameters and failed to classify the RDMatrices, again probably due to lack of training data.

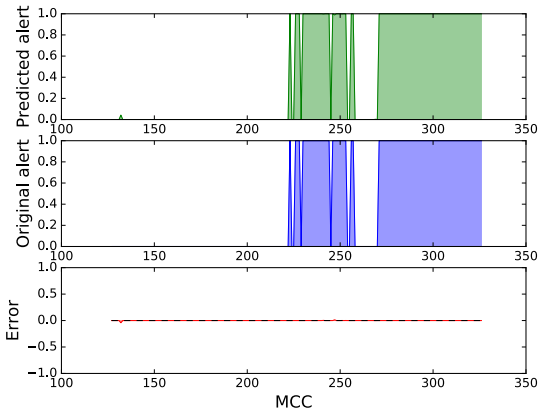
Appendix A

Experiments on the training datasets

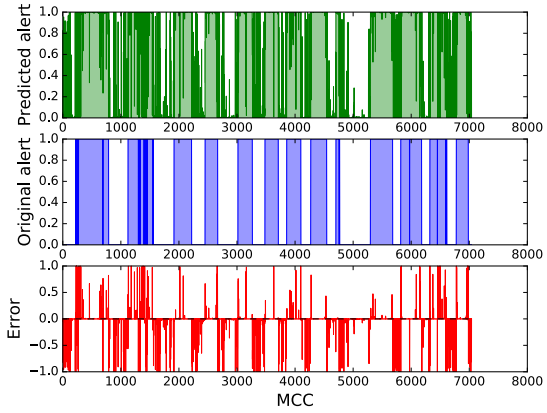
This chapter contains predictions of different models on one drive from *training* dataset. There is always a example of full drive and detail, which illustrates the worst predicted alert. Prediction from the model is always a number in the range from 0 to 1, which can be interpreted as probability of an alert. The bottom part of each plot shows difference between predicted and original values.



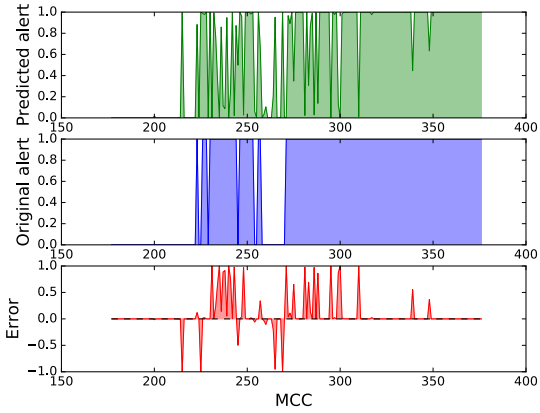
(a) CNN + MLP



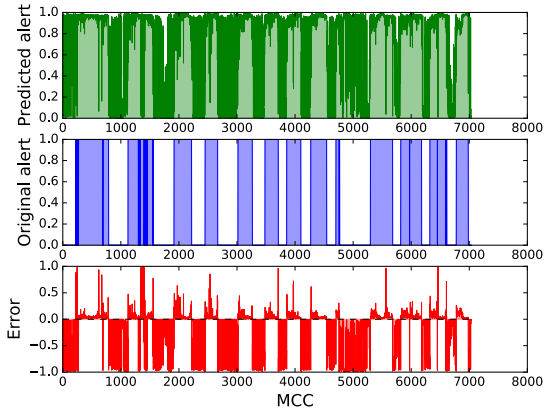
(b) CNN + MLP (detail)



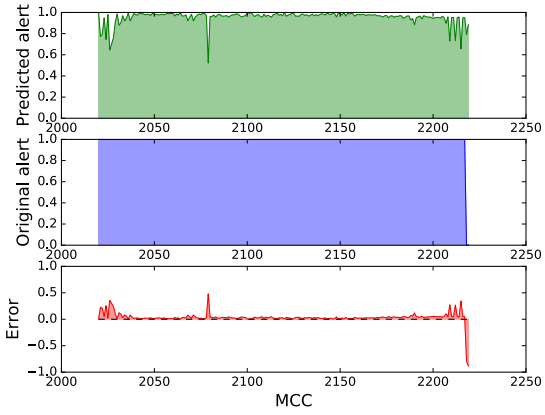
(c) CNN + LSTM



(d) CNN + LSTM (detail)

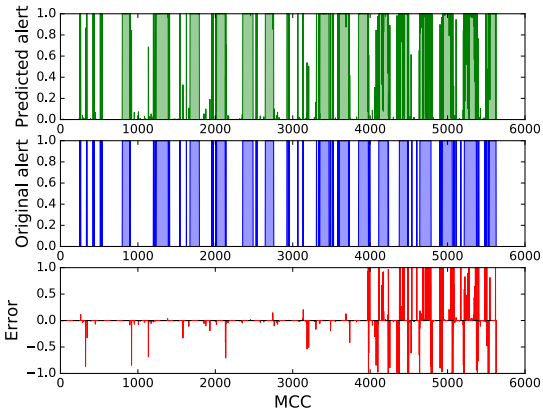


(e) VGG16

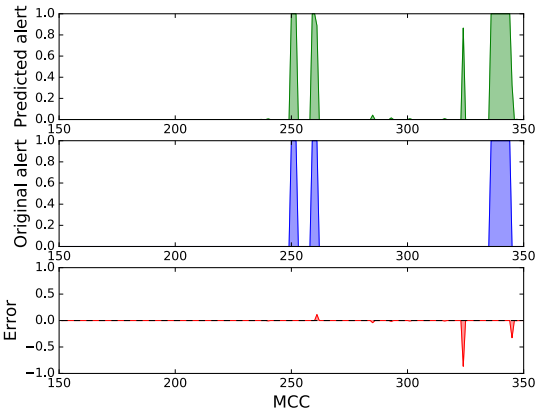


(f) VGG16 (detail)

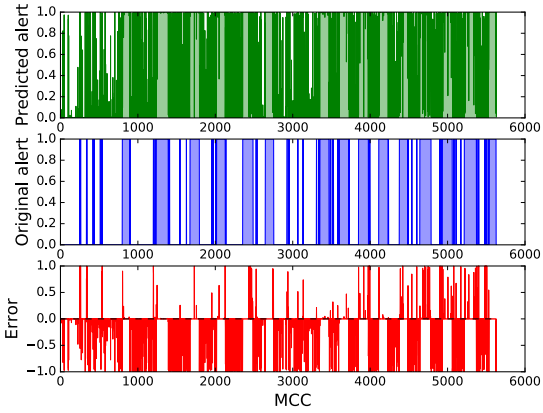
Figure A.1: Drive #1



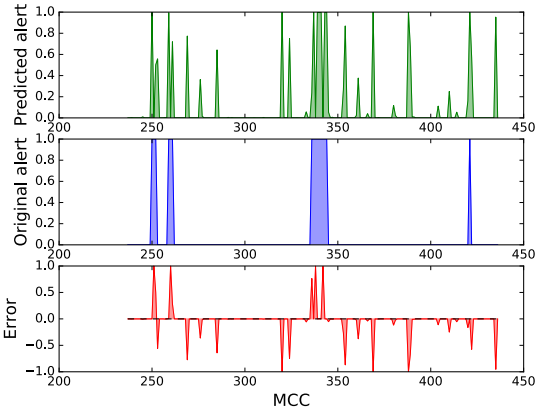
(a) CNN + MLP



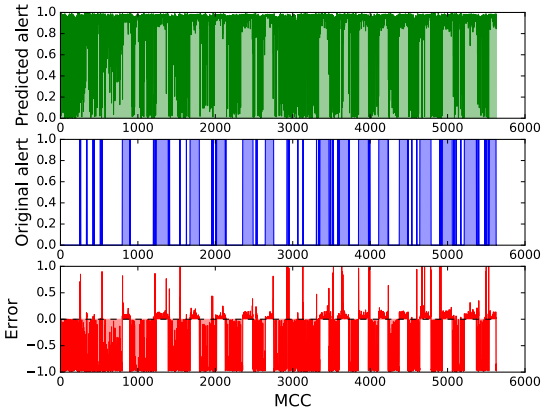
(b) CNN + MLP (detail)



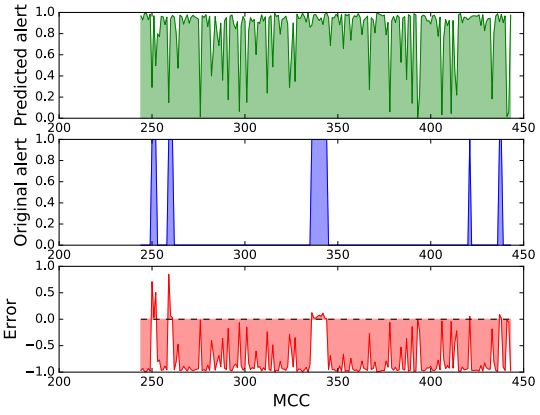
(c) CNN + LSTM



(d) CNN + LSTM (detail)

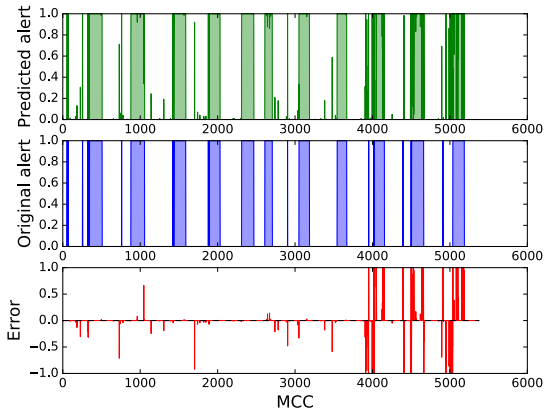


(e) VGG16

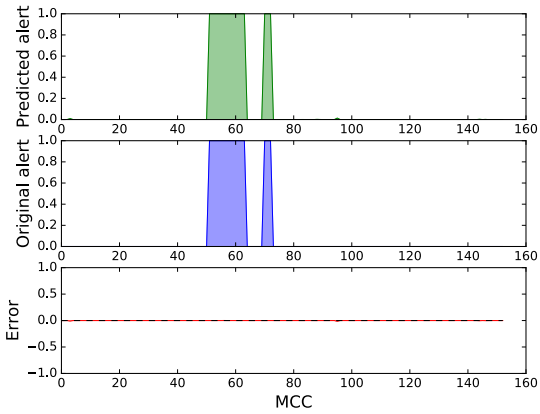


(f) VGG16 (detail)

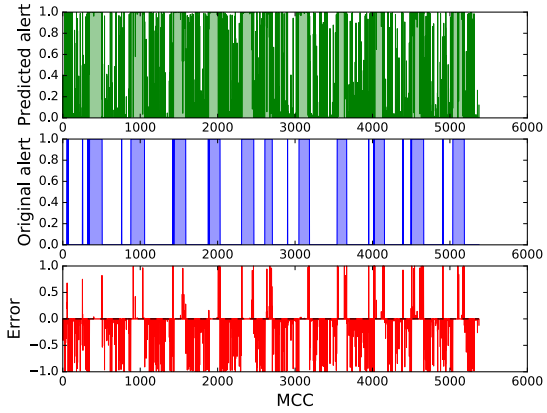
Figure A.2: Drive #2



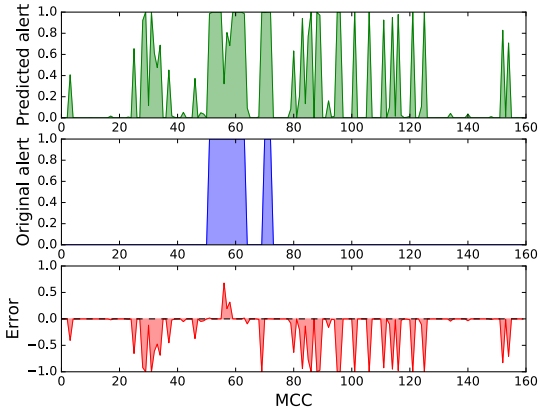
(a) CNN + MLP



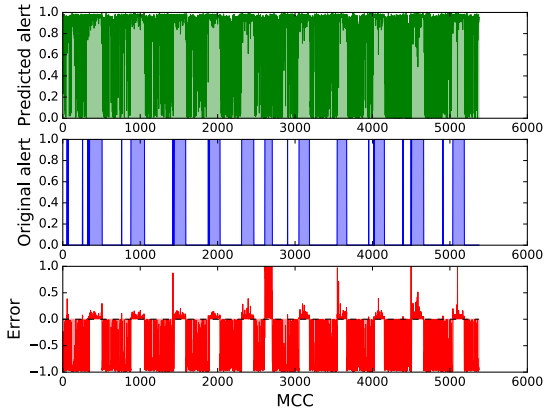
(b) CNN + MLP (detail)



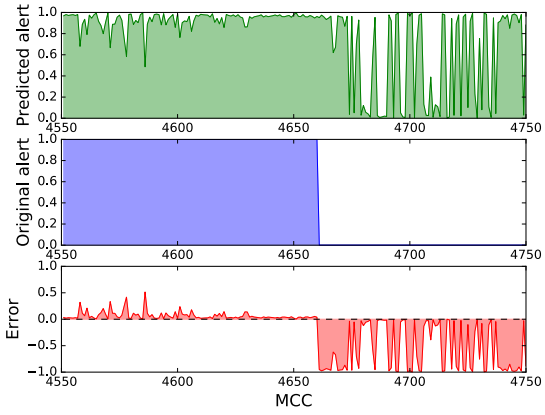
(c) CNN + LSTM



(d) CNN + LSTM (detail)

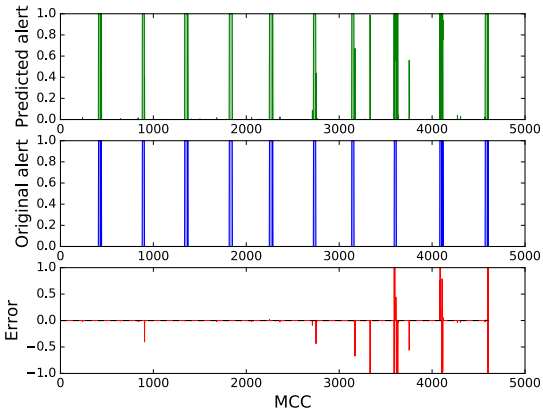


(e) VGG16

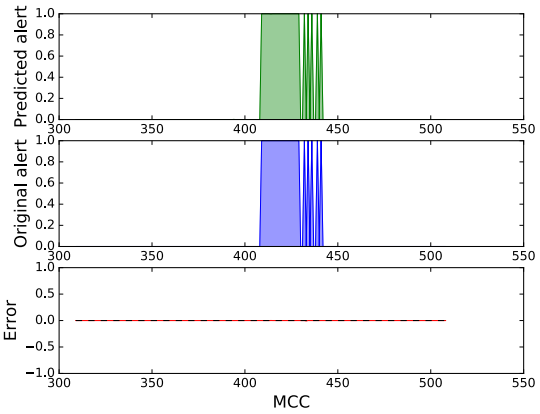


(f) VGG16 (detail)

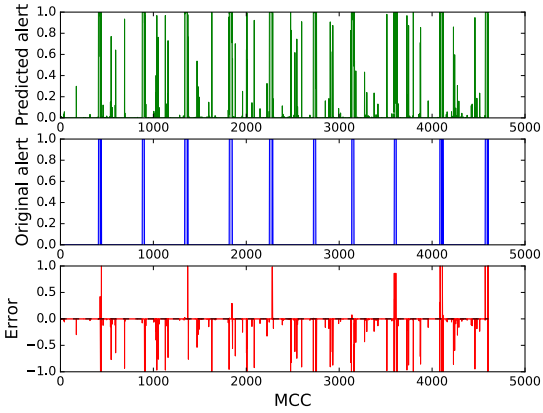
Figure A.3: Drive #3



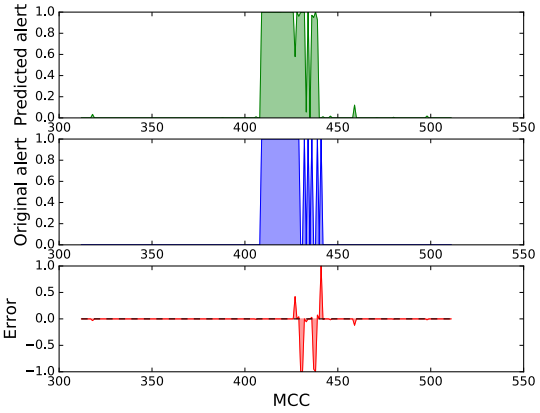
(a) CNN + MLP



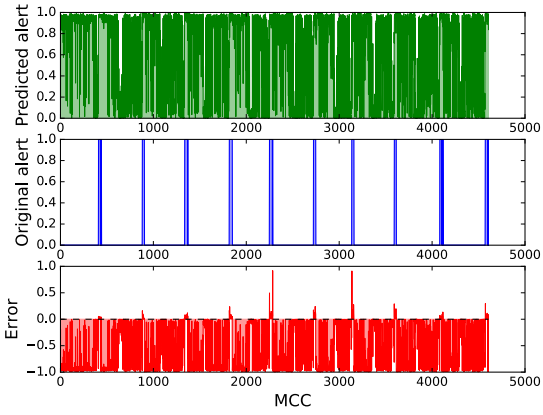
(b) CNN + MLP (detail)



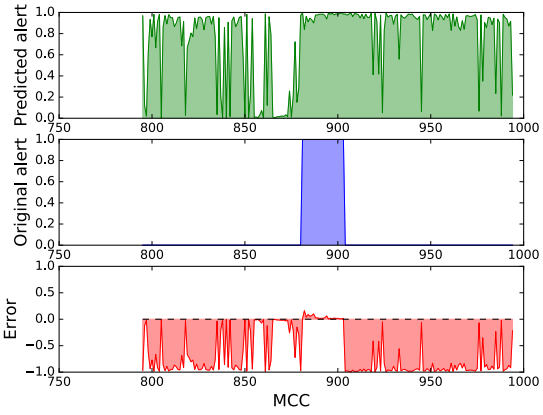
(c) CNN + LSTM



(d) CNN + LSTM (detail)



(e) VGG16



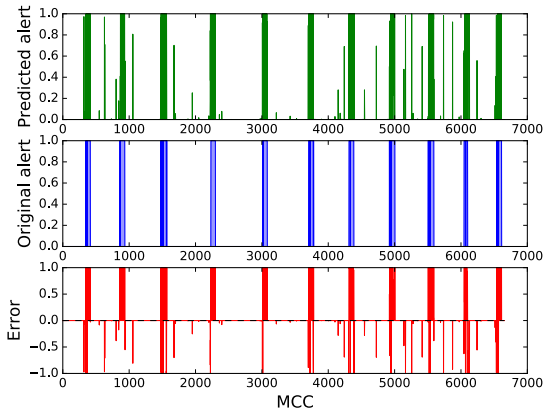
(f) VGG16 (detail)

Figure A.4: Drive #4

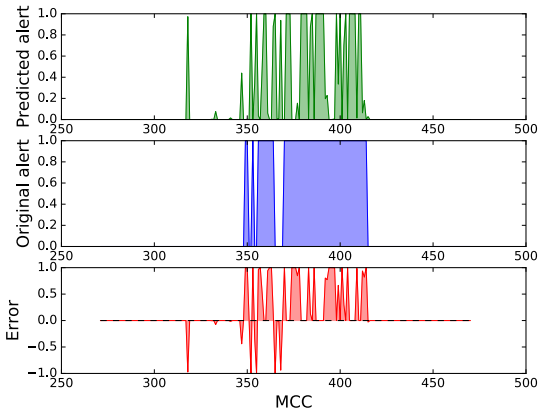
Appendix B

Experiments on the testing datasets

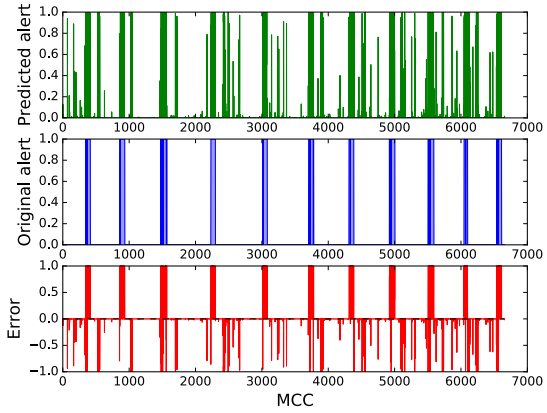
This chapter contains predictions of different models on one drive from *testing* dataset. There is always a example of full drive and detail, which illustrates the worst predicted alert. Prediction from the model is always a number in the range from 0 to 1, which can be interpreted as probability of an alert. The bottom part of each plot shows difference between predicted and original values.



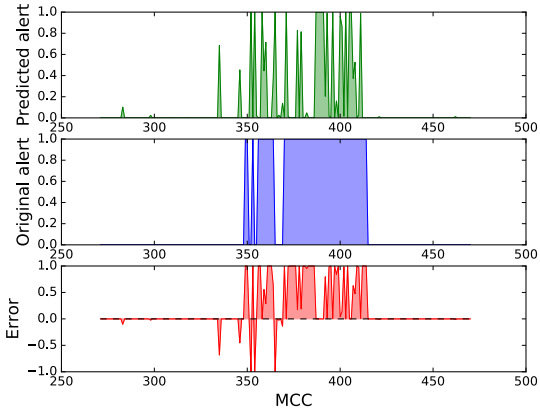
(a) CNN + MLP



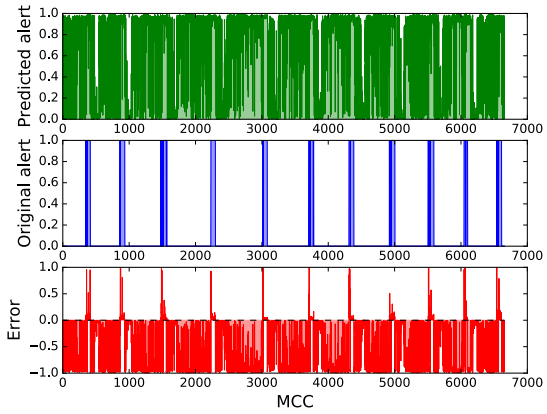
(b) CNN + MLP (detail)



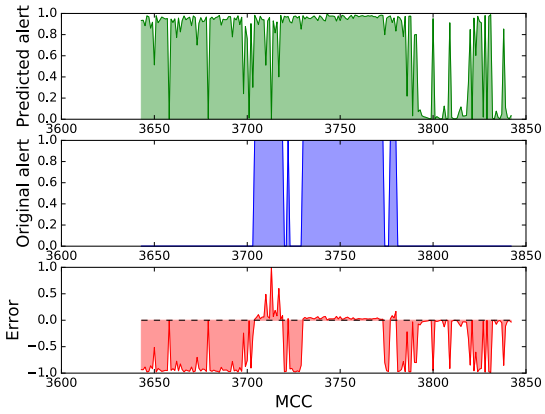
(c) CNN + LSTM



(d) CNN + LSTM (detail)

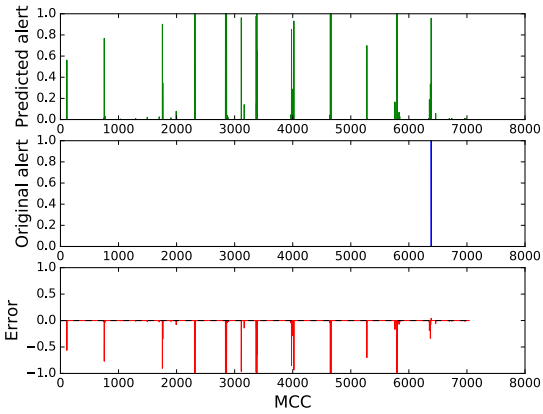


(e) VGG16

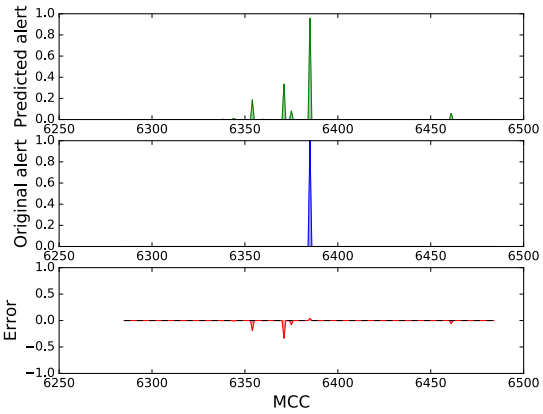


(f) VGG16 (detail)

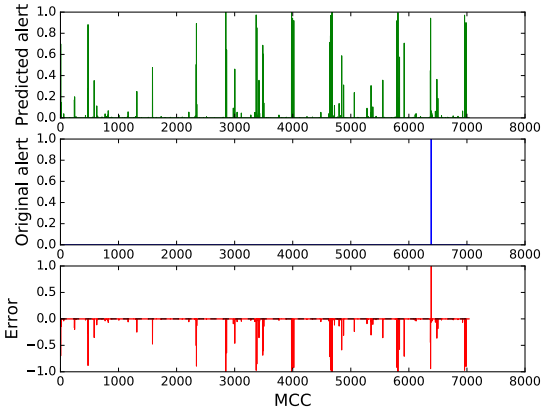
Figure B.1: Drive #1



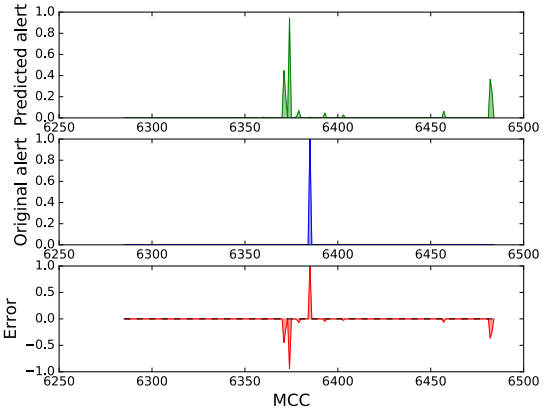
(a) CNN + MLP



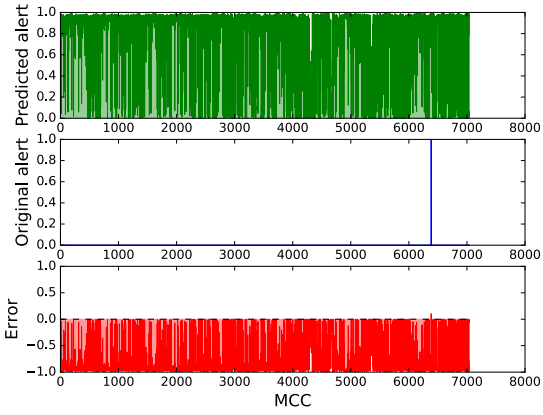
(b) CNN + MLP (detail)



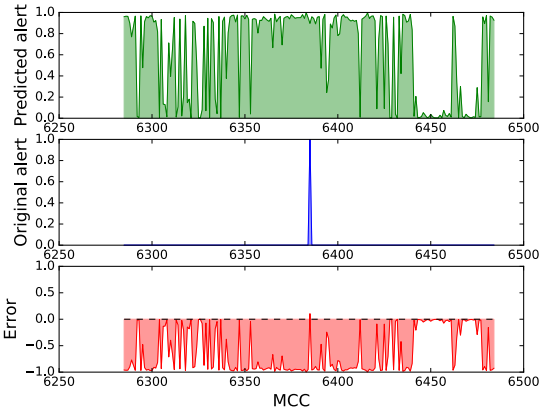
(c) CNN + LSTM



(d) CNN + LSTM (detail)

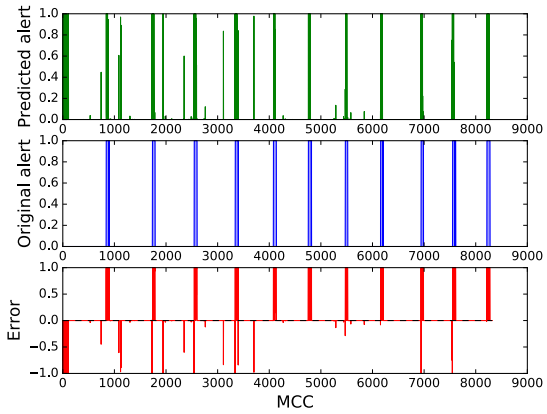


(e) VGG16

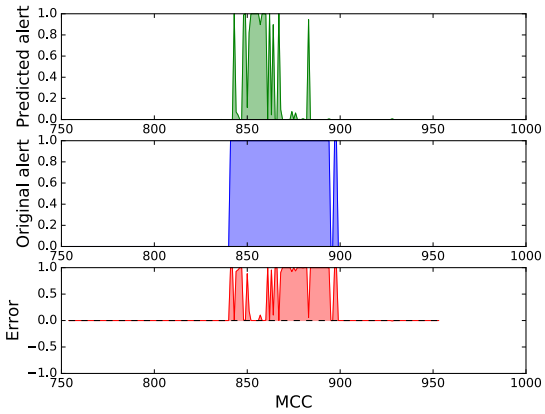


(f) VGG16 (detail)

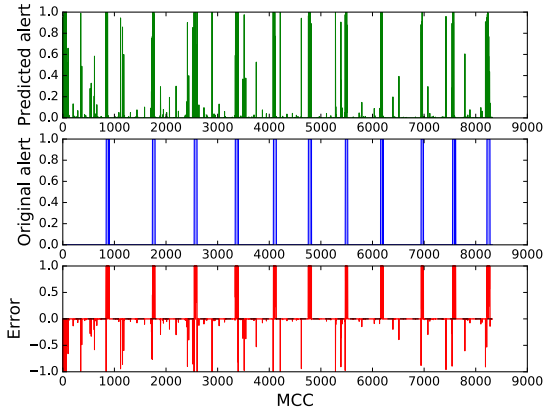
Figure B.2: Drive #2



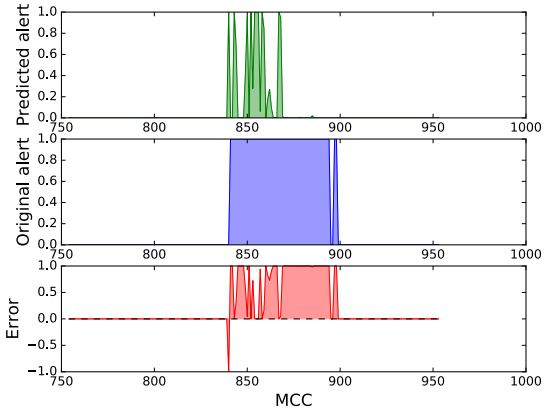
(a) CNN + MLP



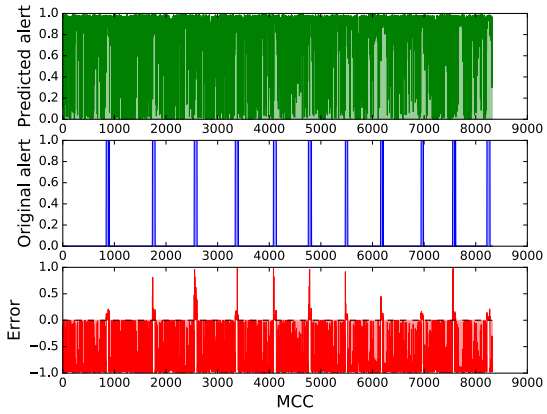
(b) CNN + MLP (detail)



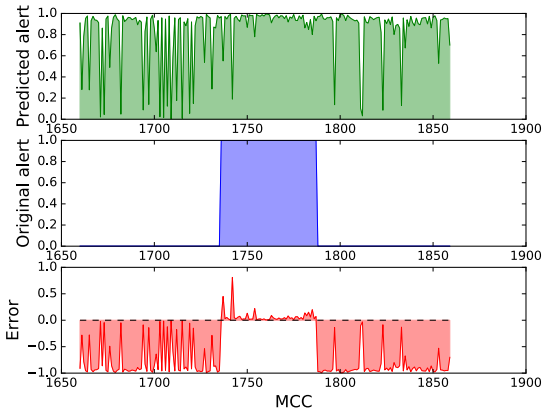
(c) CNN + LSTM



(d) CNN + LSTM (detail)



(e) VGG16



(f) VGG16 (detail)

Figure B.3: Drive #3

Bibliography

[1] ABADI, M. et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Source: <http://tensorflow.org/>. Software available from tensorflow.org.

[2] AL-RFOU, R. et al. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*. May 2016, abs/1605.02688. Source: <http://arxiv.org/abs/1605.02688>.

[3] BENGIO, Y. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*. 2009, 2, 1, s. 1–127. doi: 10.1561/2200000006. Also published as a book. Now Publishers, 2009.

[4] CHOLLET, F. Keras, 2015. Source: <https://github.com/fchollet/keras>.

[5] CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*. 1989, 2, 4, s. 303–314. ISSN 1435-568X. doi: 10.1007/BF02551274. Source: <http://dx.doi.org/10.1007/BF02551274>.

[6] DAHL, G. E. – SAINATH, T. N. – HINTON, G. E. Improving deep neural networks for LVCSR using rectified linear units and dropout. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, s. 8609–8613, May 2013.

[7] DIELEMAN, S. et al. Lasagne, 2014. Source: <https://github.com/Lasagne/Lasagne>.

[8] ERHAN, D. et al. Scalable Object Detection using Deep Neural Networks. *CoRR*. 2013, abs/1312.2249. Source: <http://arxiv.org/abs/1312.2249>.

[9] FUKUSHIMA, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*. 1980, 36, 4, s. 193–202. ISSN 1432-0770. doi: 10.1007/BF00344251. Source: <http://dx.doi.org/10.1007/BF00344251>.

[10] GRAVES, A. et al. A Novel Connectionist System for Unconstrained Handwriting Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. May 2009, 31, 5, s. 855–868. ISSN 0162-8828. doi: 10.1109/TPAMI.2008.137.

[11] GRAVES, A. Generating Sequences With Recurrent Neural Networks. *CoRR*. 2013, abs/1308.0850. Source: <http://arxiv.org/abs/1308.0850>.

[12] HOCHREITER, S. – SCHMIDHUBER, J. Long Short-Term Memory. *Neural Comput.* November 1997, 9, 8, s. 1735–1780. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. Source: <http://dx.doi.org/10.1162/neco.1997.9.8.1735>.

[13] HOCHREITER, S. et al. Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies. *IEEE*. 2001.

[14] HUBEL, D. H. – WIESEL, T. N. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*. October 1959, 148, s. 574–591. ISSN 0022-3751. Source: <http://view.ncbi.nlm.nih.gov/pubmed/14403679>.

[15] HUBEL, D. H. – WIESEL, T. N. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*. March 1968, 195, 1, s. 215–243. ISSN 1469-7793. Source: <http://jp.physoc.org/content/195/1/215.abstract>.

[16] IOFFE, S. – SZEGEDY, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR*. 2015, abs/1502.03167. Source: <http://arxiv.org/abs/1502.03167>.

[17] JEONG, S.-H. et al. A Multi-Beam and Multi-Range Radar with Fmcw and Digital Beam Forming for Automotive Applications. *Progress In Electromagnetics Research*. 2012, 124, s. 285–299. doi: 10.2528/PIER11110805. Source: <http://www.jpier.org/pier/pier.php?paper=11110805>.

[18] KINGMA, D. P. – BA, J. Adam: A Method for Stochastic Optimization. *CoRR*. 2014, abs/1412.6980. Source: <http://arxiv.org/abs/1412.6980>.

[19] KOMAROV, I. – SMOLSKIY, S. *Fundamentals of Short-range FM Radar*. Artech House radar library. Artech House, 2003. Source: <https://books.google.co.in/books?id=Am0FKx37IysC>. ISBN 9781580537339.

[20] KOUTNÍK, J. – SCHMIDHUBER, J. – GOMEZ, F. J. Evolving deep unsupervised convolutional networks for vision-based reinforcement learning. In *GECCO*, 2014.

[21] KRIZHEVSKY, A. – SUTSKEVER, I. – HINTON, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NIPS'12, s. 1097–1105, USA, 2012. Curran Associates Inc. Source: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.

[22] LECUN et al. Handwritten Digit Recognition with a Back-Propagation Network. In *Advances in Neural Information Processing Systems*, s. 396–404. Morgan Kaufmann, 1990.

[23] LI, J. et al. Extracting rebar's reflection from measured GPR data. In *Proceedings of the Tenth International Conference on Grounds Penetrating Radar, 2004. GPR 2004.*, 1, s. 367–370, June 2004.

[24] LIU, W. et al. SSD: Single Shot MultiBox Detector. *CoRR*. 2015, abs/1512.02325. Source: <http://arxiv.org/abs/1512.02325>.

[25] LONG, J. – SHELHAMER, E. – DARRELL, T. Fully Convolutional Networks for Semantic Segmentation. *CoRR*. 2014, abs/1411.4038. Source: <http://arxiv.org/abs/1411.4038>.

[26] MAHAFZA, B. R. *Radar Systems Analysis and Design Using MATLAB*. Boca Raton, FL, USA : CRC Press, Inc., 1st edition, 2000. ISBN 1584881828.

[27] MAHAFZA, B. *Introduction to Radar Analysis*. Advances in Applied Mathematics Series. Taylor & Francis, 1998. Source: <https://books.google.cz/books?id=HnbERplIrX0C>. ISBN 9780849318795.

[28] MCCULLOCH, W. S. – PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*. 1943, 5, 4, s. 115–133. ISSN 1522-9602. doi: 10.1007/BF02478259. Source: <http://dx.doi.org/10.1007/BF02478259>.

[29] NOH, H. – HONG, S. – HAN, B. Learning Deconvolution Network for Semantic Segmentation. *CoRR*. 2015, abs/1505.04366. Source: <http://arxiv.org/abs/1505.04366>.

[30] RICHARDS, M. A. (Ed.). *Principles of Modern Radar: Basic principles*. Radar, Sonar & Navigation. Institution of Engineering and Technology, 2010. Source: <http://digital-library.theiet.org/content/books/ra/sbra021e>.

[31] ROSENBLATT, F. *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory, 1957. Source: <https://books.google.cz/books?id=P_XGPgAACAAJ>.

[32] RUSSAKOVSKY, O. et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*. 2015, 115, 3, s. 211–252. doi: 10.1007/s11263-015-0816-y.

[33] SIMONYAN, K. – ZISSERMAN, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*. 2014, abs/1409.1556. Source: <http://arxiv.org/abs/1409.1556>.

[34] SKOLNIK, M. I. *Radar Handbook*. Electronic engineering series. McGraw-Hill, 1990. Source: <https://books.google.cz/books?id=m_rvUQswj3sC>. ISBN 9780070579132.

[35] SONG, M. – LIM, J. – SHIN, D. J. The velocity and range detection using the 2D-FFT scheme for automotive radars. In *2014 4th IEEE International Conference on Network Infrastructure and Digital Content*, s. 507–510, Sept 2014. doi: 10.1109/ICNIDC.2014.7000356.

[36] SWERLING, P. Probability of detection for fluctuating targets. *IRE Transactions on Information Theory*. April 1960, 6, 2, s. 269–308. ISSN 0096-1000. doi: 10.1109/TIT.1960.1057561.

[37] TAIGMAN, Y. et al. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, s. 1701–1708, June 2014. doi: 10.1109/CVPR.2014.220.

[38] The HDF Group. *Hierarchical Data Format, version 5* [online]. 1997-NNNN. /HDF5/.

[39] ZANTEN, A. T. Bosch ESP Systems: 5 Years of Experience. In *SAE Technical Paper*. SAE International, 05 2000. doi: 10.4271/2000-01-1633. Source: <http://dx.doi.org/10.4271/2000-01-1633>.

[40] WEISSTEIN, E. W. Heaviside Step Function — From MathWorld, A Wolfram Web Resource. http://mathworld.wolfram.com/HeavisideStepFunction.html.

[41] WIKIPEDIA. Phase-comparison monopulse — Wikipedia, The Free Encyclopedia, 2016. Source: <https://en.wikipedia.org/w/index.php?title=Phase-comparison_monopulse&oldid=745108284>. [Accessed 29-December-2016].

[42] WIKIPEDIA. Radar cross-section — Wikipedia, The Free Encyclopedia, 2016. Source: <https://en.wikipedia.org/w/index.php?title=Radar_cross-section&oldid=747135738>. [Accessed 31-October-2016].

[43] WOJTKIEWICZ, A. et al. Two-dimensional Signal Processing in FMCW Radars. In *XX th National Conference on Circuit Theory and Electronic Networks*, 2, 1997.

[44] WYMANN, B. et al. TORCS, The Open Racing Car Simulator. http://www.torcs.org, 2014.