University of Oxford

Engineering Science

4th Year Project Report

Learning to Interpret RADAR Data

Nihaar Shah

Lady Margaret Hall

Under the supervision of Prof. Ingmar Posner

With thanks to Robert Weston

2018

FINAL HONOUR SCHOOL OF ENG

UNIVERSITY OF OXFORD

DECLARATION OF AUTHORSHIP

You should complete this certificate. It should be bound into your fourth year project report, immediately after your title page. Three copies of the report should be submitted to the Chairman of examiners for your Honour School, c/o Clerk of the Schools, examination Schools, High Street, Oxford.

Name (in capitals): ……………………………………………………………………………………………

College (in capitals): ………………………………… **Supervisor:** …………………………………………

Title of project (in capitals): ………………………………...……………………………………………

Page count (excluding risk and COSHH assessments): ……………………………………………………

Please tick to confirm the following:

I have read and understood the University's disciplinary regulations concerning conduct in examinations and, in particular, the regulations on plagiarism (*The University Student Handbook. The Proctors' and Assessors' Memorandum, Section 8.8*; available at https://www.ox.ac.uk/students/academic/student-handbook)	☐
I have read and understood the Education Committee's information and guidance on academic good practice and plagiarism at https://www.ox.ac.uk/students/academic/guidance/skills.	☐
The project report I am submitting is entirely my own work except where otherwise indicated.	☐
It has not been submitted, either partially or in full, for another Honour School or qualification of this University (except where the Special Regulations for the subject permit this), or for a qualification at any other institution.	☐
I have clearly indicated the presence of all material I have quoted from other sources, including any diagrams, charts, tables or graphs.	☐
I have clearly indicated the presence of all paraphrased material with appropriate references.	☐
I have acknowledged appropriately any assistance I have received in addition to that provided by my supervisor.	☐
I have not copied from the work of any other candidate.	☐
I have not used the services of any agency providing specimen, model or ghostwritten work in the preparation of this project report. (See also section 2.4 of Statute XI on University Discipline under which members of the University are prohibited from providing material of this nature for candidates in examinations at this University or elsewhere: http://www.admin.ox.ac.uk/statutes/352-051a.shtml.)	☐
The project report does not exceed 50 pages (including all diagrams, photographs, references and appendices).	☐
I agree to retain an electronic copy of this work until the publication of my final examination result, except where submission in hand-written format is permitted.	☐
I agree to make any such electronic copy available to the examiners should it be necessary to confirm my word count or to check for plagiarism.	☐

Candidate's signature: …………………………………………….. **Date:** ………………………..

Abstract

Perception for autonomous vehicles relies on several modalities of which RADAR is a promising and relatively unexplored one. Although RADAR has lower resolution, it provides longer range and is resistant to adverse ambient conditions such as rain, snow, light etc. This project explores the classification and detection of cars using RADAR.

Training data is obtained from a state-of-the-art Laser Object Detector by projecting its detections to the corresponding radar scans in an automatic fashion. A linear SVM is trained using best-practices on these training examples, serving as a baseline classifier. To be able to classify cars at any orientation, the training examples are randomly rotated and the SVM is retrained. A Convolutional Neural Network (CNN) architecture is adapted for classifying our training examples, which results in superior performance for both aligned and rotated patches. CNN performance is further improved using unsupervised-pretraining. A sliding-window based car detection scheme is motivated and used for the RADAR scans. Our detector's performance is evaluated against a hand-labeled test set, also keeping in mind runtime speed.

This project demonstrates the promise of deep-learning in opening-up great possibilities within a valuable modality of perception for self-driving vehicles.

Acknowledgements

I would like to thank my supervisor Professor Ingmar Posner, for his valuable insights at various stages. I would also like to thank Robert Weston for his support and guidance. Last, but not least, I extend my gratitude to everyone at the Oxford Robotics Institute for making me feel at home in the lab.

Contents

Chapter 1

Introduction

1.1 Motivation

Perception, the ability to “see”, “hear” or “become aware of” something, is an indispensable quality for anyone that wishes to move - whether that be a microscopic cell navigating inside an organism or an autonomous mobile robot navigating on the road. For the latter, the challenge of reliably sensing the presence of surrounding objects then classifying and localising each object in varied and adverse ambient conditions in real-time is an active research problem. The modalities of perception for mobile robots are diverse and some notable examples include Vision, Sonar (for mainly aquatic vehicles), Laser and Radar.

Vision and Laser have been developed more widely by the Robotics Perception community partly due to those sensors’ ability to provide high-resolution information. However, there are significant drawbacks of each of those. Range: A typical hi-definition Laser is restricted to a radius of 100m. Ambient conditions: Laser sensors can saturate in the presence of direct sunlight (or direct high-beam headlights)[23], vision cameras perform poorly in the dark and both Vision and Laser suffer in rain, snow or fog. The value in using RADAR is twofold - 1) it provides information over longer distances (500m typically) 2) it is less affected by weather conditions such as snow, rain, sunlight or the lack of thereof.

This project investigates the use of deep-learning to build an object classifier and detector using the RADAR modality. It is hoped that our detection framework may inspire a sophisticated radar perception system for self-driving vehicles.

1.2 Our Approach

To utilise radar for this purpose, we must first overcome some hurdles. In particular, the raw Radar signal is considerably noisy and low-resolution which poses challenges for humans to manually label objects and for machines to correctly classify them. To tackle this, we first convert the radar signal into an image (as shown in

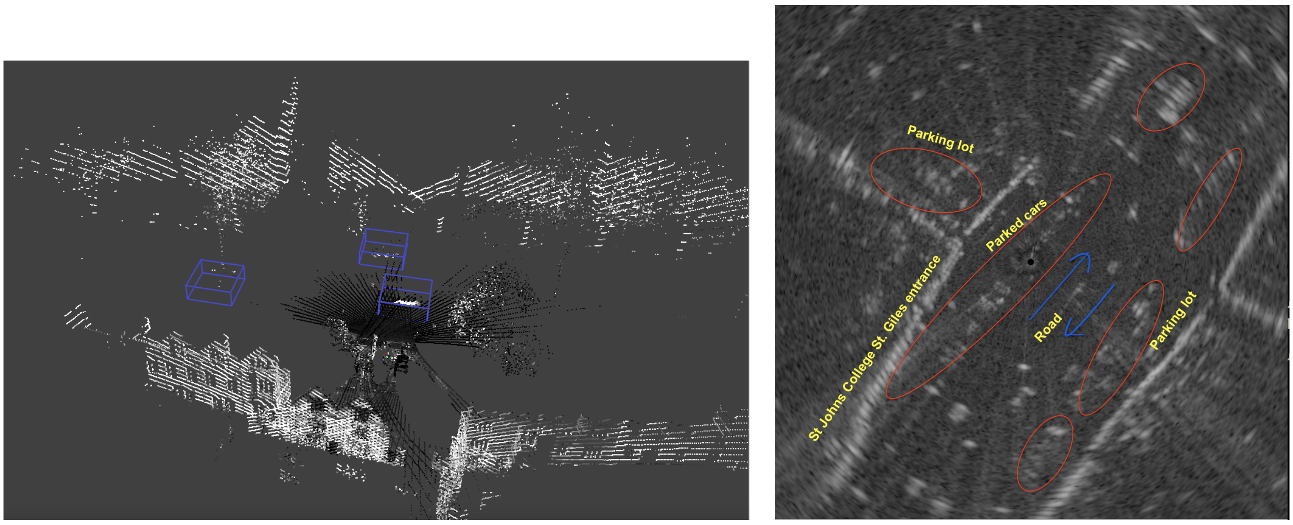


Figure 1.1: Contrasting a high resolution 3D Laser Pointcloud (left) versus a 2.5D radar scan of the same location (right) as would be seen in an aerial view. Notice that it is difficult for us to hand label the radar scene while the Laser pointcloud is richer in structure hence easier.

fig 1.1) as the input to our machine learning algorithms. Second, we project the detections made by a state-of-the-art Laser Detector to the corresponding radar scans from the same datasets. This is how we overcome the challenge of obtaining training labels sans hand-labelling. To do classification of cars in radar, we then extract these projected bounding boxes and train a classifier. As we want our detector to be able to identify cars at any orientation in the radar scene, the car patches are randomly rotated and the classifier is trained again on these rotated patches. Both an SVM and a deep network will be explored and compared as classifiers. Lastly, a sliding-window based detection is performed to localise where cars are found in a test radar scene.

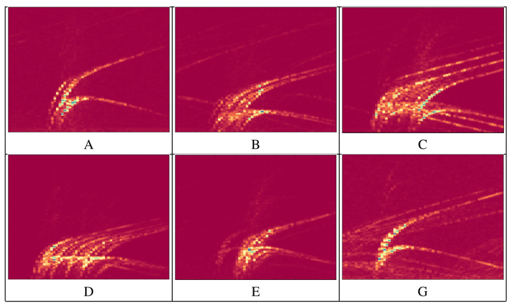
This project builds on the work of a previous 4YP, which focused on developing a calibration software toolbox to determine the relative transform between the lidar pointcloud and the radar frame of reference. Having determined the transform, the project went on to detect cars using an SVM classifier and a sliding window approach. Building upon this line of research, in this project we explore the use of deep learning for this task.

1.3 Literature Review

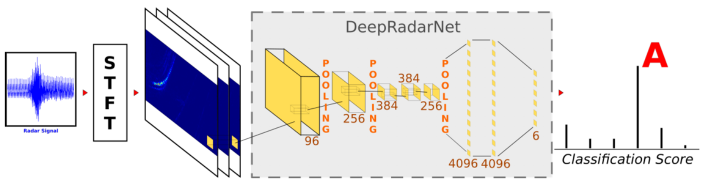
Object Classification in RADAR

Radar detection using deep learning for self driving cars has been unexplored relative to detection in lidar or vision in the academic community at least in the public domain. Most closely related is the recent paper by Capobianco et al.[2] from late October 2017. A 1D radar signal is transformed into a 3D range-doppler[1] signal by computing its STFT (Short term Fourier Transform). Examples of the average of the three computed channels for the 6 vehicle categories are shown in fig 1.2a. Then the three computed channels are stacked to

[1]Doppler radar uses the doppler effect i.e. frequency shift between transmitted and received signal to measure the radial velocity of an object



(a) Range-Doppler signature for different vehicle categories are shown above where letters have the following meanings: A: car, B: car-trailer, C: truck , D: cargo truck, E: bus, G: motorcycle [2]



(b) From a 1-D radar signal, using the Short Time Fourier Transformation, three range Doppler signatures (up-ramp, down-ramp, average-ramp) are computed and stacked into one tensor. A DeepRadarNet model maps the tensor into the classification score predicting the class of vehicle[2].

Figure 1.2: Illustration of the input image used to predict vehicle categories and the CNN used for this classification task by [2].

form a tensor and fed to a CNN which outputs the vehicle categories as depicted in fig 1.2b. This study achieved an average recognition accuracy of 0.961.

The above experimental setup, however, was specifically geared to the task of traffic monitoring: The radar was placed on a fixed structure at 5.3 m height with the antenna illuminating a single lane and pointing toward the back of the vehicles passing under the structure along that lane. Moreover, they have access to the vehicle velocities using the doppler frequency shift calculations and intend to use this system to augment traditional speed cameras. The results quoted in the paper are for the case when there is only one target vehicle and that it is passing directly under the structure on which the radar is mounted. The paper acknowledges that several complications arise if more general use cases are considered. For instance when there are multiple vehicles, especially if they are moving side by side. Difficulties also arise when there is a relative velocity between the radar sensor and vehicle. There has not been a mention of how the labels were generated or how many of them were used so it is most likely that labels were manually done.

In a different approach, Lombacher et al. [14] first choose to convert the radar scans into occupancy grids in which each cell of the grid covering a radar scan is associated with a value denoting its probability to be occupied. The radar occupancy grids were hand-labelled by drawing bounding boxes around the object and assigned a label from categories such as cars, fence, curb, field etc. Possible objects are extracted from the occupancy grid map via a connected component analysis[2]. The training data is obtained by cutting out a window of $8m \times 8m$ around each possible object as shown in fig 1.3. If the object is larger, which does happen for objects such as buildings and curbstones then multiple variations of the same object are created using a sliding window.

For classification, a CNN inspired by AlexNet[12] was used and one-vs-rest classifiers are trained for each class. An equally distributed prior on the number of objects in each class was assumed because of local variations

[2]an algorithm based on graph theory wherein portions of connected components - here, regions of the digital image - are uniquely labeled using a heuristic e.g. connectivity, relative values of neighboring pixels etc.

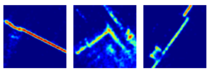
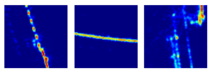
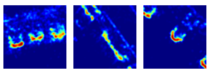
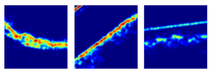
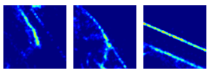
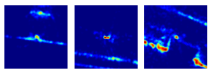
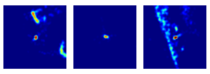
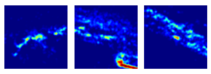
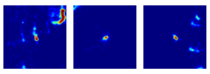
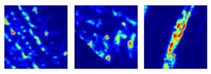
No of Samples	Label	Examples	No of Samples	Label	Examples
46	*Building*		118	*Fence*	
1669	*Car*		405	*Shrub*	
51	*Curbstone*		212	*Tree*	
133	*Pole*		69	*Field*	
104	*TrafficSign*		590	*Other*	

Figure 1.3: The hand-labelled training patches for various object categories used by Lombacher et al.[14] depicted along with the number of samples for each category. The availability of few samples is due to reliance on hand-labelling.

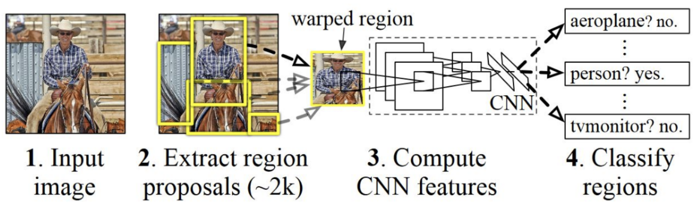
in a typical scan. Data balance was achieved by oversampling the unbalanced dataset in two steps. First, the multi-class dataset is balanced, so all classes are equally distributed. Then the data set is transformed into a one (the examined class, e.g Car) vs. rest (all other classes) data set and the examined class is over-sampled. The data set then has two classes which are equally distributed and the 'rest' class has an equal distribution among the sub-classes. The classification results varied among the 10 classes with all classes other than 'car' being in the range 0.60-0.75 but the "car" accuracy being 0.966. Detection was not studied in this paper.

The main limitation of this study, however, was the very few number of real training samples used (with as few as 46 examples in the worst case) as shown in fig. 1.3. This was because of reliance on hand-labelling which is tedious and time-consuming. There was also a heavy reliance on data augmentation to compensate for the few real labelled samples. As a result, the model trained will likely not generalise very well on a completely different radar scene.

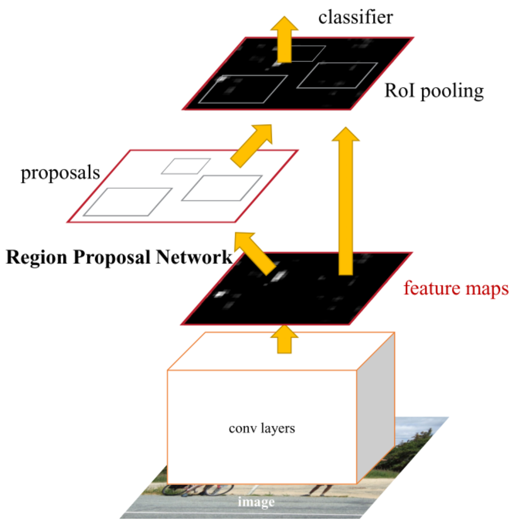
Our work is set apart from this prior work in that we do away with hand labelling altogether thereby being able to obtain a larger number of training samples, we apply deep learning purely on the power intensities returned by objects without needing doppler information and we develop a radar object detector. These are the novelties of our work.

Object Detection in Vision

There are broadly three paradigms prevalent in vision based object-detection today namely R-CNN, YOLO and Sliding-Window. Neither of these detection frameworks has been applied to the RADAR modality, to our knowledge. We will describe the approaches first, highlighting their respective merits and limitations if applied to our RADAR scans and then motivate our choice which is detailed in chapter 5.



(a) R-CNN [8] approach has stages: generate candidate regions, extract features, classify and regress bounding box around object.



(b) Faster-RCNN uses a unified network for both region proposal and classification. Features extracted by the CNN can be used to get region-proposals too - by running a sliding window on features and outputting k anchor-boxes and scoring them. This saves running a separate algorithm for region-proposal[22].

Figure 1.4: R-CNN and Faster RCNN paradigms illustrated.

R-CNN

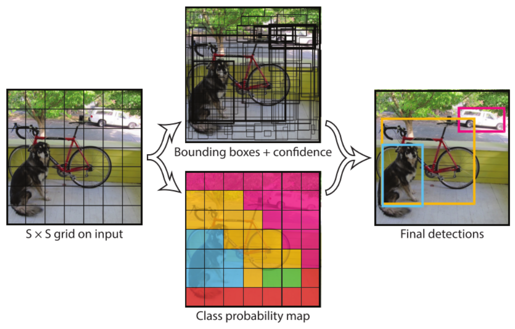
RCNN has three distinct parts as also illustrated in fig.1.4a:

1. **Category-independent region proposals:** Candidate regions are selected using a measure such as "objectness" which quantifies how likely it is for an image window to contain an object of any class.
2. **Feature extraction using a CNN:** Region proposals of various dimensions are warped and passed through a CNN to obtain a feature vector.
3. **Class specific linear SVMs:** The final stage performs multi-class classification by scoring the feature vector for every class - the highest score is the predicted class.

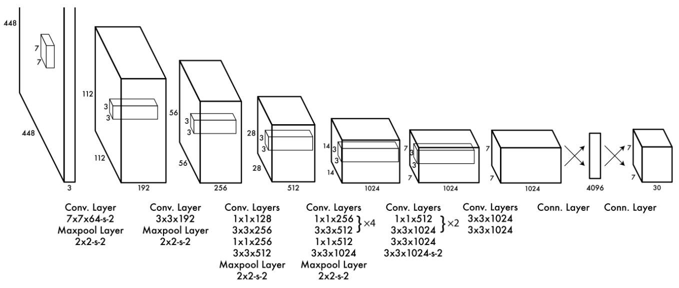
R-CNN [8] has a run-time performance of 53 seconds/image on a CPU and 13s/image on a GPU. This is slow for a real-time application like ours. There have been two major upgrades in performance to this R-CNN paradigm with modified pipelines namely Fast-RCNN and Faster-RCNN. Faster-RCNN [22] (as shown in fig.1.4b) using the VGG-16 architecture attains a frame rate of 5fps i.e. 0.20s/image frame. This is done on a K40-GPU enabling parallelisation. Accuracy is less relevant as we have a different modality, but they achieved a mean average precision in the range 0.65-0.78 on the PASCAL-VOC dataset. Thus, Faster-RCNN is one potentially promising approach for RADAR.

YOLO

You Only Look Once (YOLO)[21] has a multi-part loss function which is used to regress bounding-box(BB) coordinates, confidence of each BB and class probabilities. It is rapid compared with faster-RCNN with a frame



(a) YOLO models detection as a regression problem. It divides the image into grids and for each grid predicts bounding boxes, confidence of the boxes and class probabilities[21].


448
448
3
112
112
192
56
56
256
28
28
512
14
14
1024
7
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 ×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 ×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

(b) The architecture consists of 24 convolutional layers, 20 of which are pretrained on the Imagenet dataset[21].

Figure 1.5: Illustrations of YOLO approach of splitting an image into a grid and YOLO's architecture shown.

rate of 45 fps on a Titan-X GPU. However, it struggles to precisely localize some objects, especially small ones. The network has 24 convolutional layers as shown in in fig.1.5b and is pretrained on the Imagenet classification task and then converted to perform detection. This could be a drawback of using it in RADAR as we can't pretrain on data at the Imagenet scale along with the fact that it struggles on objects of small size (as cars in the radar scene are relatively small). YOLO relies on having accurate labels for the *entire* scene. We have good labels for some objects in the scene but we have no guarantees on having labelled all of the objects due to limitations in the lidar classifier and differences in the lidar and radar field of view. In contrast, in the sliding window approach, a classifier can be trained using only individual examples of the object.

Sliding-Window

Sliding window approach is more straightforward in that it exhaustively classifies a fixed sized patch iteratively over the entire image. A prominent case of this approach was in the Viola-Jones [26] paper for face-detection where they obtained a frame rate of 15fps on a 384×288 pixels image albeit using a simple Haar based classifier. This was back in 2001 on a 700MHz Pentium III processor. The sliding window approach is appealing to us as cars in radar scan are of fixed size. This is because of the absolute nature of the sensor, unlike the scaling ambiguity in vision. Thus, we can run a fixed size window to detect the cars. To be able to detect cars at various orientations, we must also train our classifier on randomly rotated car patches. In contrast to a fixed patch based sliding-window, Faster-RCNN's region proposal network outputs bounding-boxes of various aspect ratios, which may be an overkill. YOLO would not be suitable as it needs accurate ground-truth over the entire scan which we do not have.

1.4 Project Aims and Contributions

Our primary aim in this project is to explore deep-learning based object classification and detection in radar images. To this end, as will be described in chapter 2, we begin with extracting training labels from a state-of-the-art 3D LiDAR object detector on varied datasets and projecting these using an existing software toolbox on the corresponding radar scans building upon the approach developed in [27]. We then explore classification of the object of interest (i.e. a car) in chapter 3 and lay down a baseline classification performance using a linear SVM on aligned as well as randomly oriented car patches. Next, we utilise a CNN architecture for this classification task and improve its performance using unsupervised pretraining as described in chapter 4. Finally, we use our deep classifier and a sliding-window approach for object detection in the wider radar scene and evaluate the performance against a test-set of hand labelled frames in chapter 5. In accomplishing these aims, the project has the following two contributions:

- Demonstrating the superior performance of the CNN compared with the linear SVM, both trained using best practices, on aligned and rotated car patches
- Developing software tools for sliding-window based radar object detection and evaluation so that future improved CNN architectures maybe deployed easily on radar data

Chapter 2

Data Acquisition and Background

As discussed in the last chapter, noise and low resolution in radar signals are key challenges of the modality. To understand the origins of noise in RADAR, we first describe the working of our radar sensor. Then we outline the procedure of detecting in Lidar and extracting projected detections. Finally, we introduce the metrics with which we evaluate our algorithms in following chapters.

2.1 Working of RADAR

RADARs (Radio Detection and Ranging) are active sensors, i.e. they project microwaves to detect rather than just responding to external stimuli (passive sensor) which makes them more versatile. Two notable types of RADARs used in robotics are: (1) Pulsed RADAR which transmits a pulse first then activates the receiver - the elapsed time between pulse transmission and reception is a measure of the range to the target and (2) Continuous RADAR which operates the transmitter and receiver continuously and so separates the transmitted signal from the received signal on the basis of frequency. As pulsed radars have lower range resolution and consume higher peak power, we look towards FMCW radars which is what we use in this project. A schematic of an FMCW is depicted in fig. 2.1 showing the coupling of transmitted and received signals to obtain the beat signal which encodes range information of objects.

In FMCW radars, first the received and transmitted signals are mixed and low pass filtered to give us the beat signal equation where δf is the chirp bandwidth, T_d is the time between chirps, τ_i is the time of flight and ω_c is the carrier frequency:

$$\nu_{beat}(t, \tau_i) \propto cos[2\pi((\frac{\Delta f}{T_d}\tau_i)t - (\omega_c + \frac{\Delta f}{T_d}\tau_i)\tau_i] \tag{2.1}$$

The signal is then smoothed and digitized for each azimuth before performing FFT. The FFT is multiplied by its complex conjugate to generate a power spectrum. Each frequency in this spectrum corresponds to a particular amplitude which is proportional to the Time of Flight to an object. The final sensor measurements are output as intensity values corresponding to each azimuth indexed by range as seen in fig2.2b.

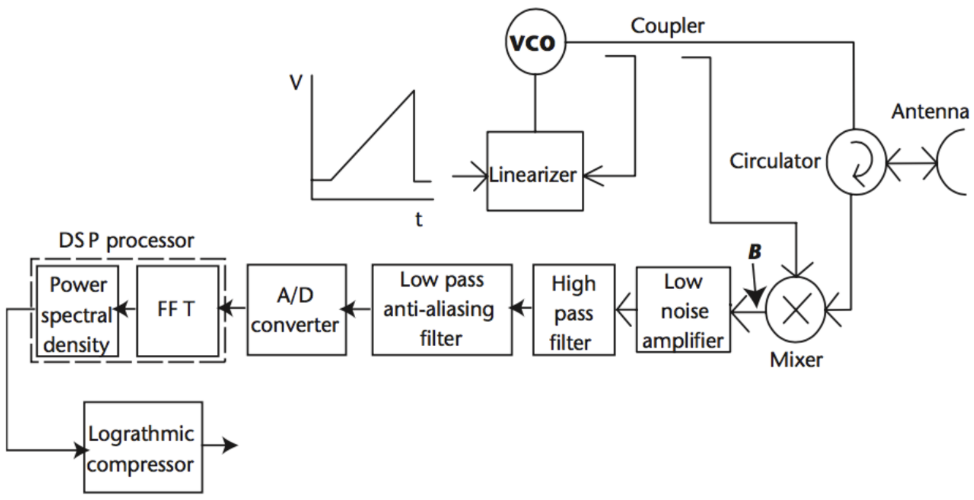
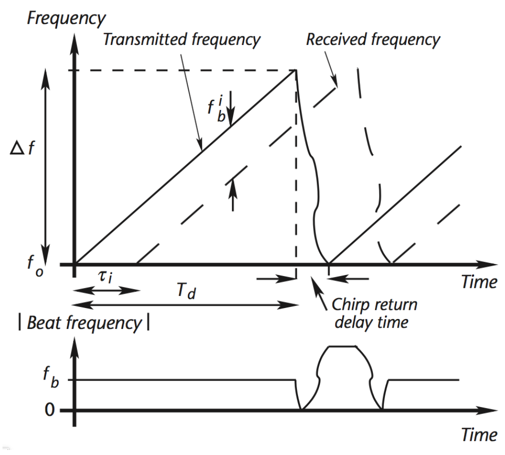
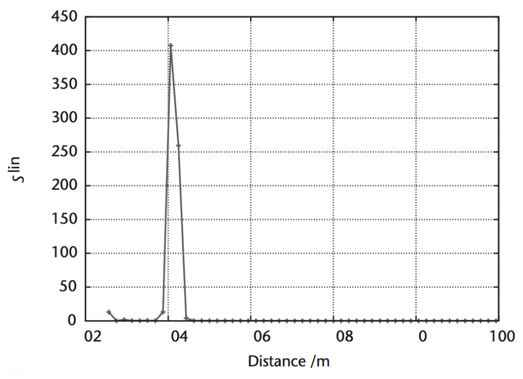


Figure 2.1: The oscillator generates a linearly increasing frequency signal over a time-period t [1]. At the receiver, some part of the transmitted signal is mixed with the received signal and then filtered and amplified to retrieve the beat signal as shown in eq.2.1; For every azimuth, the beat signal is collected and FFT is performed to then obtain a power spectrum. The amplitude corresponding to each frequency in this spectrum is proportional to the Time of Flight (ToF) to an object. The final output is intensity values indexed by range for every azimuth as can be seen in fig. 2.2b



(a) Beat is the difference in frequency between the transmitted signal and received echo and is most informative for range measurement. T_d is the time between chirps, Δf is the chirp bandwidth and τ_i is time of flight[1].

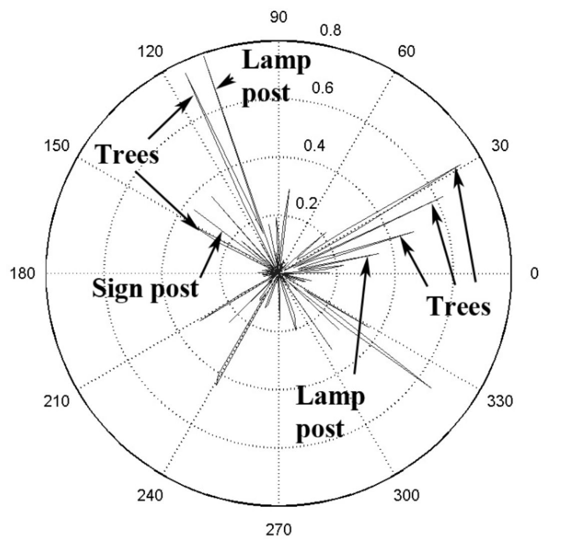


(b) A-scope consists of discrete received power values corresponding to a range bin in every azimuth [1].

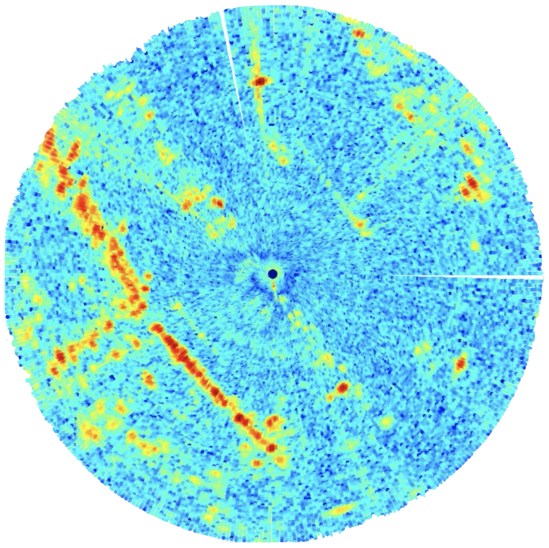
Figure 2.2: Schematic of the transmitted and received signal along with the beat signal obtained from them shown on the left. The beat signal frequency and amplitude contain information about the range and intensity of objects. This information can be plotted as shown in the figure on the right.

It can be seen that the received power (S^{lin} in fig 2.2b) in each radar range bin is derived from the amplitude of the beat signal and that the beat frequency carries the range information. In particular, if the beat frequency is $f_b^i = \frac{\delta f}{T_d}\tau_i$, assuming the speed of radio waves to be c, then the delay time between the transmitted and received chirps will be $\tau_i = 2\frac{r^i}{c}$ where r_i is the range to the object being sensed. Thus we can get the relationship between the electronically measured beat frequency and range: $r^i = \frac{cT_d}{2\Delta f} f_b^i$. Hence, closer objects produce signals with lower beat frequencies and vice versa.

Noise and Data Characteristics



(a) Intensity values for every azimuth is shown for the whole radar scan [1]. In reality the scan is not as clean as this due to spurious reflections and noise as shown in the adjacent figure2.3b.



(b) Our real raw data is in the form of a scatter plot of intensities, here shown by scaling grayscale to colour (red is high intensity and blue is low). This is a scatter of St.Giles road, Oxford. Noise is visible as brighter and darker dots among surroundings.

Figure 2.3: Our raw radar signal is a scatter plot of intensities because for every range bin in each azimuth, the radar returns an intensity value.

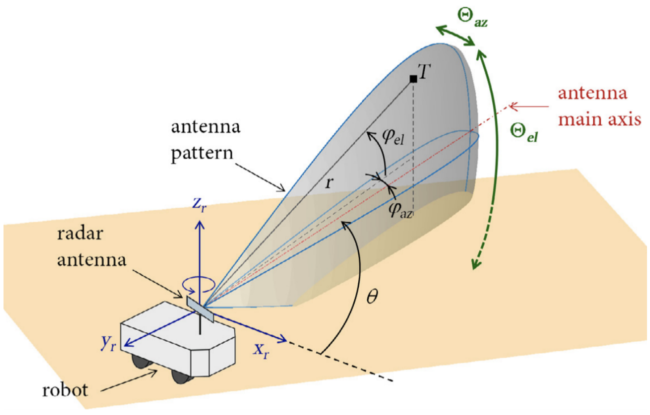
As seen in fig. 2.3b, the presence of noise makes a scan harder to interpret by covering finer details of objects. This means either we must mitigate noise or learn how to incorporate it into our model. Primary noise sources are[1]:

- **Clutter:** unwanted echoes that exceed the detection threshold of the radar.
- **Speckle noise:** a result of constructive and destructive interference between multiple objects.
- **Cross-talk:** leakage of the signal from the transmitter to the receiver because of stray capacitances and high frequencies of the signals
- **Receiver Noise:** thermal noise at the receiver. As range increases the signal becomes weaker until it is of an equivalent magnitude as the noise power. This defines the maximum range of the radar.
- $\frac{1}{f}$ **noise:** generated in the amplification stages and is a result of thermal noise. This is particularly prevalent at ranges close to the radar corresponding to lower frequencies.

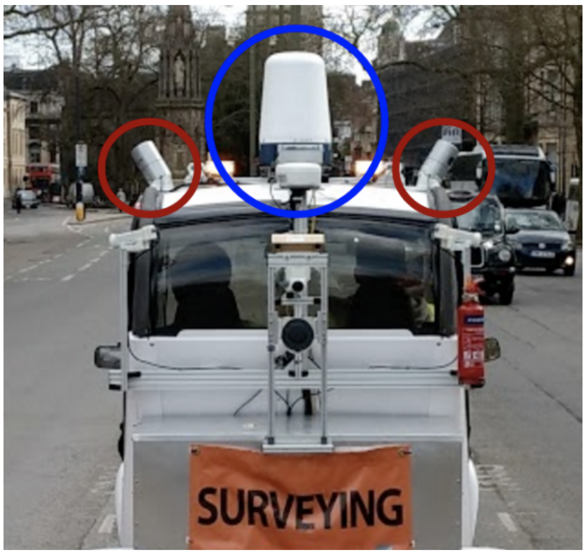
In the RADAR domain, cars at long distances don't look smaller than nearer cars. This is because the measurements we make are absolute and not relative. We know the range and azimuth to every return value which means our pixels represent known distances and so a car will always take up the same amount of space as can be noticed from fig 1.1 from the previous chapter. This is quite different from a computer-vision problem for cameras where there is this depth-scale ambiguity. For the detection task, this would usually be tackled using a Gaussian pyramid scaling i.e. scaling the image to different sizes such that stacked they resemble a

Gaussian and running a sliding window. As we shall see in chapter 5, this quality makes it easier and faster for us to use a sliding window approach to detection.

2.2 RADAR sensor



(a) θ_{az} and θ_{el} are the angles of the antenna (azimuth and elevation respectively). A target T situated in the antenna beam is localized through its polar coordinates (θ,r) while elevation angle and azimuth angles (ϕ_{el}, ϕ_{az}) of target T are unknown parameters[24].



(b) RADAR sensor was mounted on the GEM surveying vehicle and the data used by us was captured by the ORI. Blue circle shows the radar and red shows two laser sensors[27].

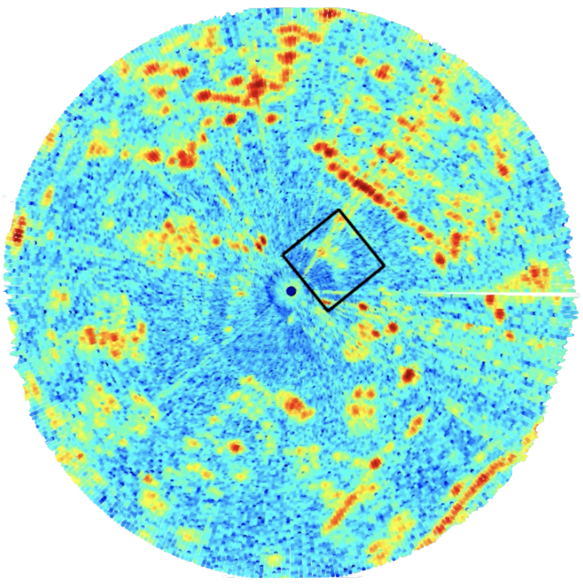
Figure 2.4: Schematic showing what is meant by azimuth and elevation of the radar sensor along with how the range and direction of a target object are obtained. On the right, we show the actual vehicle that was used as per this setup.

The data in this project is captured using a Navtech CTS350x FMCW mounted on the rear of the GEM surveying vehicle as shown in fig2.4b. The radar rotates (at $4Hz$) around its vertical axis continuously transmitting waves as an angular beam of width 2° and spreads from ±0.9 to ±25 from the horizontal[3]. Navtech CTS350x records $N_{ra} = 2000$ range readings for each of the $N_{az} = 400$ azimuths and has a range of $500m$. Thus its range resolution is $0.25m$ and angular resolution is 0.9°. The polar intensity array, corresponding to a single sweep of the radar is converted into a Cartesian image using bi-linear[1] interpolation in order to get an image that is without discontinuities. This leads to further distortions in how objects are represented.

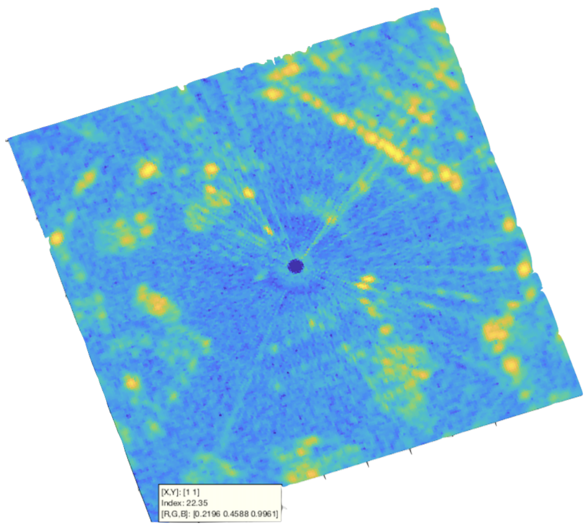
2.3 LiDAR sensor

Light Detection and Ranging (LiDAR) operate in the infrared portion of the light spectrum. The data in this project was captured from two Velodyne HDL32E lidars. Lidar produces a sparse set of points measured in 3D cartesian co-ordinates. Every point $p(i) = [x_L, y_L, z_L]^T$ within the current sweep corresponds to the intensity value $I^{(i)}$ and time stamp $T^{(i)}$. The two velodynes rotate at $10Hz$ and their output was combined to form a

[1]It considers the closest 2×2 neighborhood of known pixel values surrounding the unknown pixel's computed location and takes a weighted average of these 4 pixels to arrive at its final, interpolated value



(a) Radar raw scan is obtained as range and intensity values in polar coordinates best shown as a scatter plot. This is an actual scan from Lamb-and-Flag-2 dataset along with a projected Laser detection in black.



(b) The scatter plot is transformed into a cartesian image using a transformation of coordinates followed by bilinear interpolation. Original grayscale images are scaled here to colour.

Figure 2.5: Illustrations of the raw radar signal along with the processed image that we choose to use for our project.

single pointcloud. The lidar we use in this project has a range resolution of $0.02m$ which is superior than that of RADAR which is $0.25m$.

Combining pointclouds

First, we had to combine data from the two concurrent laser scan sources into a single combined pointcloud. This was done using ORI's application CombineVelodyneScansOnline, written in C++, which accepts separate source streams along with a set of relative poses (obtained from Visual Odometry [VO]) and outputs a single combined pointcloud. This was done for each of the 5 datasets used in the project before we ran Vote3Deep. The relaying of the 2 laser streams and relative poses was done via MOOS which is a powerful low latency cross platform inter-process communication (IPC) system, that uses a lightweight communication hub (MOOSDB) to communicate with respective processes (clients)[16].

LiDAR object detector

This project uses a LiDAR detection system named Vote3deep[5] to generate training labels. Vote3deep is capable of multi-class detections and is found to be more accurate than its predecessor Vote3D - an SVM based laser detector. We shall briefly describe the essence of how Vote3Deep works and how it can perform so efficiently.

Processing 3D pointclouds is exponentially more expensive than a 2D image due to dimensionality. However, if the pointcloud is sparse, then performing operations such as convolution can be realised. Vote3Deep's key

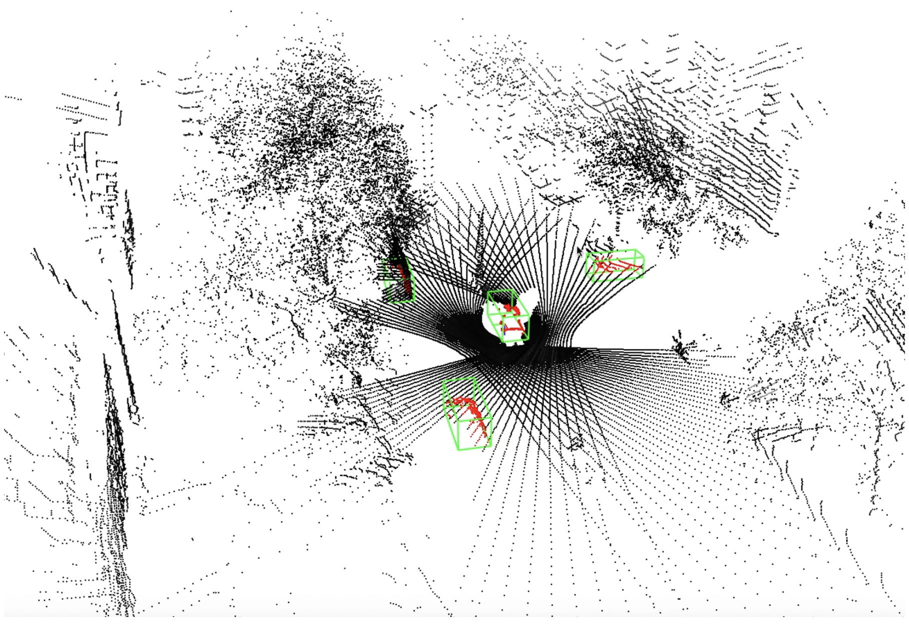
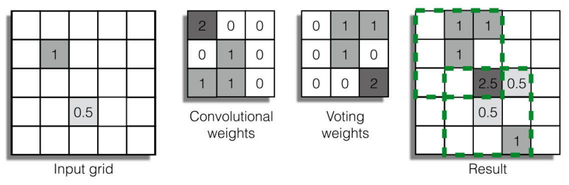
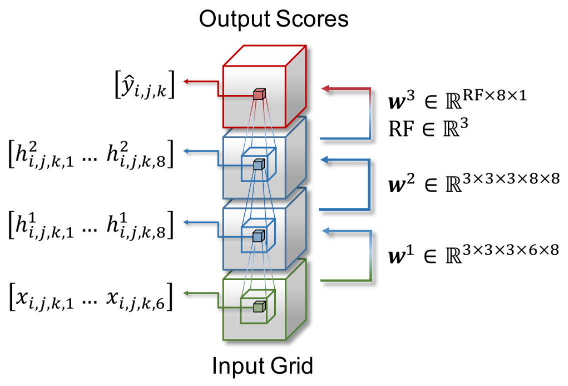


Figure 2.6: Laser pointcloud and detections made by Vote3Deep as visualised in the native visualiser. In this scene all the cars are detected.

insight was writing a convolution as a voting scheme as shown in fig 2.7a.



(a) While a standard convolution slides the kernel along every element of the input tensor, the voting procedure only needs to be applied at each non-zero location to obtain an equivalent result. The voting weights are simply the flipped convolutional weights along each dimension[5]. This key insight of Vote3Deep make CNNs feasible for sparse pointclouds.


Output Scores
$[\hat{y}_{i,j,k}]$
$[h^2_{i,j,k,1} \dots h^2_{i,j,k,8}]$
$[h^1_{i,j,k,1} \dots h^1_{i,j,k,8}]$
$[x_{i,j,k,1} \dots x_{i,j,k,6}]$
$\boldsymbol{w}^3 \in \mathbb{R}^{\mathrm{RF} \times 8 \times 1}$
$\mathrm{RF} \in \mathbb{R}^3$
$\boldsymbol{w}^2 \in \mathbb{R}^{3 \times 3 \times 3 \times 8 \times 8}$
$\boldsymbol{w}^1 \in \mathbb{R}^{3 \times 3 \times 3 \times 6 \times 8}$
Input Grid

(b) There are 3 layers that perform convolutions with a voting kernel w^c and incorporate ReLU nonlinearities. On a CPU it takes on average 0.8 seconds for car detections per scan. Average precison was 0.75 for cars[5].

Figure 2.7: Illustrations of Voting as a convolution on sparse data and the architecture used by the Vote3deep detector.

Generating Labels

Vote3deep's Detect app was run on the combined pointcloud file to generate a "labels" file containing the $[x, y, z, t, \text{frame}_{\text{number}}]$ coordinates for every detection. To do this, firstly an XML file containing specifications of which pretrained vote3deep model was to be used as well as the operating point of the detector (i.e. threshold that sets desired Precision and Recall) was modified.

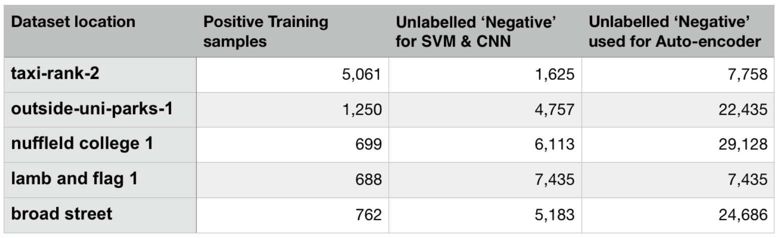
To set a suitable operating-point, the detections were first visualised using the native visualiser app. We take only the highest confidence detections made by Vote3Deep favouring high precision over recall as the presence of false positives in our training data can harm our RADAR detector model later. Another parameter to be tuned in the Vote3deep XML file was the Non-Max Suppression (NMS) value.

Bash scripts were written to automate the various steps in the sequence so far such as opening a Monolithics Player instance, MOOSDB instance, Monitor the MOOS (mtm), CombineVelodyneScansOnline and Vote3deep DetectApp supplied with an XML file. It was all processed remotely on a cluster named Smaug which has 160 cores (10 x 16-core 2.6GHz Intel Xeon) and so enables big speed-ups for processing. All of the above was repeated for the 5 datasets that were used in this project as outlined later in this chapter.

Projecting into Radar frame

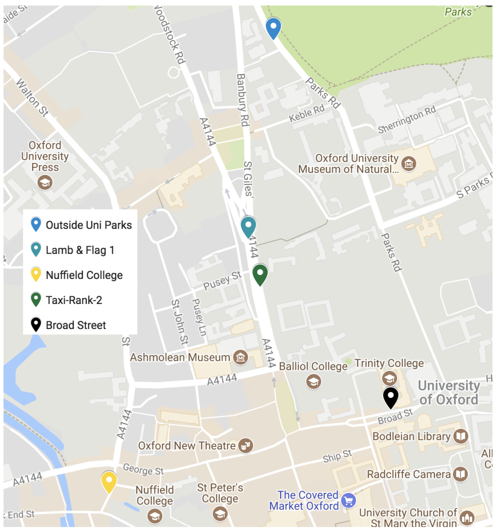
Projecting labels into the RADAR frame was done using a tool developed by the previous 4YP. The patch size chosen is of dimensions 5m by 5m square patch. This was because we observed that Vote3Deep about 30% of the time put a bounding box perpendicular to the true orientation of the car at our chosen operating point. The square patch would mean that the car is always entirely in the patch. The patches are all automatically made to align in one direction during extraction but a corresponding angle obtained from vote3deep indicating their original orientation is saved in the Monolithic file.

2.4 Datasets



Dataset location	Positive Training samples	Unlabelled 'Negative' for SVM & CNN	Unlabelled 'Negative' used for Auto-encoder
taxi-rank-2	5,061	1,625	7,758
outside-uni-parks-1	1,250	4,757	22,435
nuffleld college 1	699	6,113	29,128
lamb and flag 1	688	7,435	7,435
broad street	762	5,183	24,686

(a) Table indicates the various locations and the corresponding number of training data patches obtained.



(b) The map with pins for the locations described in the table above are shown for context.

Figure 2.8: Locations and details about number of positive car samples and negatives recorded by our RADAR and laser are shown.

Data was collected by the ORI from 5 different locations around Oxford which we used in this project. As will be seen later, data from 4 of these was used for training and validation, one was kept reserved solely for testing namely "Lamb and Flag-1". Figure 2.8b gives the locations and details of each of the datasets used in this project.

2.5 Evaluation Metrics

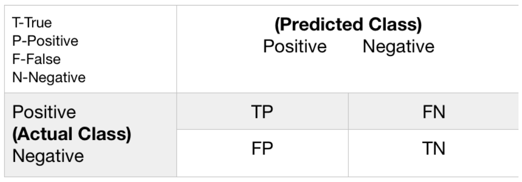
Defining an appropriate evaluation metric based on the model and task at hand is crucial before embarking on trying various approaches. We explore some of the metrics appropriate for classification and detection. All of these will later be used as ways to judge our algorithms and select the right one.

Accuracy

Accuracy is the ratio of correct predictions to total predictions. This can be a misleading metric if the test dataset is imbalanced i.e. the positive class or negative class examples are over-represented. Consider the case that there are only 10% positive samples in the test set and if the classifier predicts negative class for all the test examples then the accuracy will be 90% which suggests that the classifier is apparently good, while in reality, it is not. We use accuracy in monitoring the training performance because our training dataset is balanced.

Confusion Matrix

The general idea is to count the number of times instances of class A are classified as class B as illustrated in the figure below. A perfect classifier would have only True Positives (TP) and True Negatives (TN) so its confusion matrix will have non-zero values only on its main diagonal. We use this to evaluate our classifier's test set performance.



T-True P-Positive F-False N-Negative	**(Predicted Class)** Positive	 Negative
Positive **(Actual Class)**	TP	FN
Negative	FP	TN

Figure 2.9: Each row in the confusion matrix represents the Actual Class while each column the predicted class.

Precision and Recall

An interesting and more concise metric is the accuracy of the positive predictions which is called the Precision(Pr) of the classifier[7].

$$Pr = \frac{TP}{(TP + FP)} = \frac{\text{Relevant samples Retrieved}}{\text{Total samples Retrieved}} \tag{2.2}$$

Intuitively this is a ratio between the relevant (in our project "car") samples that exist in the test set and the retrieved/predicted samples i.e. that which the classifier "thinks" are relevant.

Precision is typically used along with another metric called Recall: the ratio of positive instances correctly classified. Intuitively this can also be written as the ratio of the relevant samples that were retrieved by the total number of relevant samples present in the test set.

$$Re = \frac{TP}{(TP+FN)} = \frac{\text{Relevant samples Retrieved}}{\text{Total Relevant samples}} \tag{2.3}$$

It is often convenient to combine precision and recall into a single metric called the $\boldsymbol{F_1 score}$. It can be a single way to compare two classifiers. The F_1 score is the harmonic mean of precision and recall, as a result, the classifier will only get a high F_1 score if both recall and precision are high.

$$F_1 = \frac{2}{(\frac{1}{Pr} + \frac{1}{Re})} = \frac{2 \times (Pr \times Re)}{(Pr + Re)} \tag{2.4}$$

We use precision, recall and F_1 scores to evaluate our radar classifier in this project.

PR curves

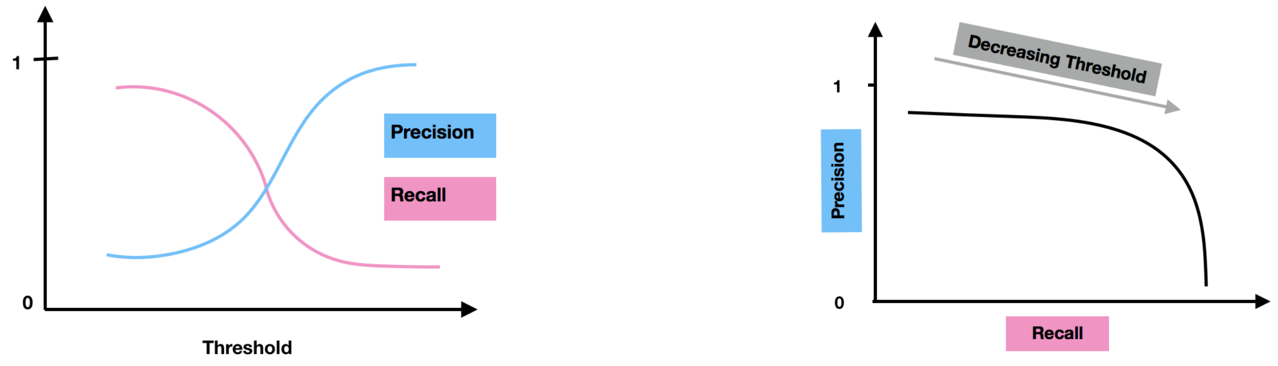


Figure 2.10: Illustration of how precision and recall individually vary with classifier's threshold. We can even vary Recall by implicitly adjusting the threshold to get various values of Precision as shown on the right.

We can plot precision and recall as a function of the threshold value. From this the **operating point** i.e. the threshold value that gives the best precision/recall tradeoff can be selected as per the task. Another way to select a good operating point is to plot precision directly against recall as shown in fig 2.10. Usually, it can be seen that precision suddenly drops after a certain recall value so it is usually suitable to select an operating point just before that drop.[7]

FPPI vs Miss rate

Object detection is a different task from classification and thus needs another metric from the above. A detector's per-image performance is assessed by analysing the FPPI vs miss-rate as a function of thresholded detection score c. FPPI stands for False Positives per Image and Miss rate is defined as the ratio of ground truth bounding boxes that were missed out.

If detection bounding boxes are denoted BB_{dt}. We then attempt to match each ground truth BB_{gt} to a detection BB_{dt} if their area of overlap exceeds 50%.

When we have a low threshold probability there will be some false positives whereas, at a higher threshold i.e. stricter criterion for classification, we expect fewer false positives. Similarly, because there are simply more detections made at a higher threshold, chances are that they cover the ground truth bounding boxes as well. This means we would expect fewer misses ie. low miss rate for high FPPI. Conversely, for a high threshold, there are fewer detections so lower FPPI but statistically higher possibility of misses so high miss rate. For every threshold between 0 and 1 we can plot miss-rates vs FPPI as a curve. The lower the curve is, the better the detection system works. This method of evaluating a detector was proposed in [28].

Chapter 3

SVM Based RADAR Classification

3.1 Motivation

We wish to develop a classifier that we can use as part of a sliding window detection scheme. For this, we need a classifier that is (1) quick to run and (2) very accurate. A classifier operating on randomly orientated patches is preferable to a classifier operating on aligned patches as the former only needs to be evaluated once at each location in the image. This is in contrast to the latter in which multiple angles must be considered and leads to a significant improvement in the speed of the detector.

Building on the work of [27], we evaluate the performance of an SVM detector operating on the aligned case and treat this as our baseline. We then evaluate the performance of an SVM detector on the non-aligned patch case and see a significant decrease in the performance of the SVM. We also experiment with some basic static thresholding and see that this results in a reduction in accuracy and an increase in classification error. This indicates that there is relevant information in all of the pixel values and simple decisions made on the magnitude of intensity values are not enough to correctly classify the dataset. We hope to utilise this information using the deep learning classifier developed in chapter 4.

3.2 Theory

The linear SVM classifier model predicts the class of a new instance $\boldsymbol{x}$ by simply computing the decision function $\boldsymbol{w}^T\boldsymbol{x}$ + b where $\boldsymbol{w}$ is the model weights and b the bias - this equation represents a plane in hyperspace. If the result of computing the equation for a test example $\boldsymbol{x}$ is positive then the example lies on one side of the plane and vice-versa.

$$\hat{y} = \begin{cases} 0 & \text{if } w^T x + b < 0 \\ 1 & \text{if } w^T x + b \geq 0 \end{cases} \tag{3.1}$$

This can be written for a hard-margin[1] linear SVM as a constrained optimisation problem:

$$\begin{aligned} \underset{\boldsymbol{w}, b, \eta}{\text{minimize}} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} \\ \text{subject to} \quad & t^{(i)}((\boldsymbol{w}^T.x^{(i)} + b) \geq 1, i = 1, \ldots, m \end{aligned} \tag{3.2}$$

To get a soft-margin[2] objective, we need to introduce a *slack variable* $\zeta^{(i)} \geq 0$ for each instance, $\zeta^{(i)}$ measures how much the i^{th} instance is allowed to violate the margin. We now have two conflicting objectives: make the slack variables as small as possible to reduce the margin violations and making the objective as small as possible to increase the margin. The C hyperparameter controls this balance: it allows us to define the trade-off between these two objectives. This gives the constrained optimisation problem:

$$\begin{aligned} \underset{\boldsymbol{w}, b, \eta}{\text{minimize}} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{m}\zeta^{(i)} \\ \text{subject to} \quad & t^{(i)}((\boldsymbol{w}^T x^{(i)} + b) \geq 1, \quad i = 1, \ldots, m, \\ & \zeta^{(i)} \geq 0, \quad \forall i \end{aligned} \tag{3.3}$$

Both the above problems are convex quadratic optimisation problems with linear constraints and there are several off-the-shelf QP solvers available.

Hyper-parameter tuning via Cross Validation

The hyperparameter C or $\frac{1}{\lambda}$ of the SVM denotes the width of the margin between the two support vector - larger C shorter the width and vice versa. It sets the strength of the regularization applied on the weight vector because it allows some misclassifications and margin violations so as to not overfit to the outliers and damage our model. C decides how strictly to segregate points which may be on the correct side of the decision surface but not far enough for the margin to allow. This, however, means we run the risk of disallowing correctly classified points to be labelled as such and that may reduce our accuracy. The optimal trade-off is found using cross-validation.

A validation-set[3] is split into k-folds or segments and training is done on k-1 folds and validating on the remaining 1 fold for several C values. We then perform a line search on C to narrow down to an optimal value[20]. Starting with values that are separated by a power of 10 to cover a wide range and then narrowing down gradually between which values are we able to get the best validation performance. After having recognised roughly region of this sweet-spot we can fine tune our estimate by dividing that range further and repeating the validation.

[1]When the classifier strictly encforces all instances to be outside the support vector margins and on the correct side

[2]When the objective is to find a balance between making the margin between support vectors as large as possible and limiting margin violations.

[3]A subset of a data that has been split for the purpose evaluating the trained model's performance and fine-tuning hyperparameters

3.3 Experiments

We begin by setting our baseline in the case of aligned car patches as this was done by the previous 4YP. Building on that, we augment the training data because our goal is to apply a sliding window to detect cars at any orientation in the scan. We retrain the SVM on augmented data and contrast its performance with the aligned case. We perform Hard Negative Mining (HNM) after realising that there is a need to select key negative examples instead of passing all the randomly samples negatives to the classifier. Lastly, in an effort to mitigate noise, we apply a simple static-threshold and train the SVM on thresholded patches.

3.3.1 Aligned car patches

Dataset: The dataset of aligned car patches consists of 7,772 samples in total over 5 different locations around Oxford. Excluding one of these locations as a testing set, we are left with 7,084 aligned positive training samples. The number of negatives is 25,113 in total and 17,678 of these are for training. Each image is of size 36×36 and we flatten the pixels into a vector to feed the SVM as features.

Training

For cross-validation, several scoring functions are available to evaluate an optimal C value - we choose cross-entropy loss as we are converting SVM scores into probabilities (Platt[4] scaling). Higher the probability of predicting a class correctly, the better. Log loss increases as the predicted probability diverges from the actual label. The scoring function used was negative log loss (NLL) hence lower the NLL, the better.

5 fold cross validation was performed for a range of C values using a grid search: first $C = [0.001, 0.01, 0.1, 1, 10]$ was tried. $C = 0.1$ gave the best score. To fine tune this $C = [0.05, 0.1, 0.8]$ was tried and $C = 0.05$ gave the lowest NLL as shown in in fig 3.1. The SVM was then re-trained using this value. Note that it is very computationally expensive to try new values of C as it involves training and validating 5 times for each new C. Training was done using the python scikit-learn.LinearSVC library with the following parameters:

Parameter	**Used quantity**	**Reason**
Penalty on weights	L2 norm	L1 norm leads to sparse svm weights
Loss function	Hinge loss	Penalises margin violations so suitable for soft-margin
Dual/Primal objective	Primal	Primal faster if samples(24,762) > features(1296)
Tolerance	1e-4	Library's default stopping criteria for the optimizer
C hyperparameter	0.05	Cross-validated via grid search

Results

The metric of interest in the table in fig. 3.2 is F_1 score for the car which is 0.69 at a probability threshold of 0.5 Although, we can get a higher F_1 score by choosing a suitable operating point, the performance of this

[4]The probabilities are calibrated using logistic regression on the SVMs scores.

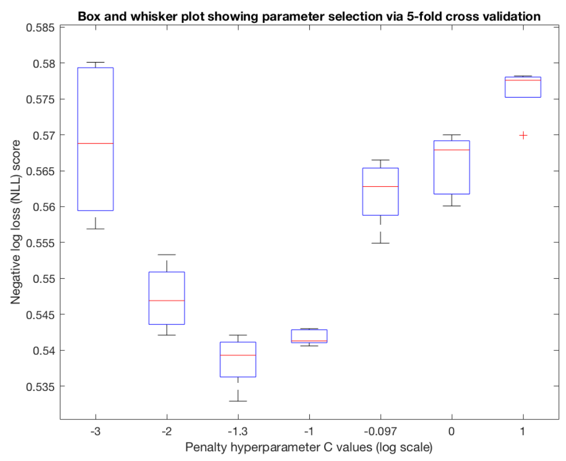
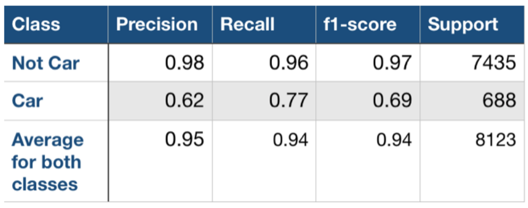
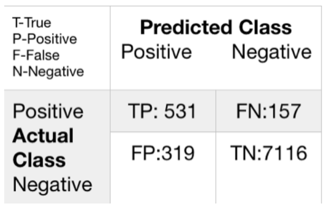


Figure 3.1: A box-whisker plot showing the spread of 5 NLL scores obtained for various C values tried using line-search. The median was taken as a summary with which to compare the C values. Lower the NLL, the better.

classifier is not competitive with generally what is expected of SVMs from other tasks such as vision. Our main hypothesis for why that is the case is that the raw image pixel features are not sufficient descriptors to linearly separate our classes. Negative samples are sometimes confusingly similar to the positive samples and vice-versa, as will be later visualised in section 3.4 adding impetus to this hypothesis.



Class	Precision	Recall	f1-score	Support
Not Car	0.98	0.96	0.97	7435
Car	0.62	0.77	0.69	688
Average for both classes	0.95	0.94	0.94	8123



T-True P-Positive F-False N-Negative	Predicted Class Positive	Predicted Class Negative
Actual Class Positive	TP: 531	FN:157
Actual Class Negative	FP:319	TN:7116

Figure 3.2: Summary of Precision, Recall, F_1 score and the confusion matrix of the SVM trained and tested on aligned data shown here for a probability threshold of 0.5 (converted using Platt scaling).

3.3.2 Randomly oriented car patches

In reality, we aim to detect cars at any orientation in the radar scan. So, the "positive" car containing patches were randomly oriented. The negative label patches were left unchanged. After another round of cross-validation to choose a renewed $C = 1.20$ hyperparameter, we trained and tested on these new patches. This time the number of car patches was increased (oversampled) three-fold due to augmentation.

Data Augmentation

We are interested to learn to classify a car at any orientation in a radar scene. Thus, the extracted car patches are rotated at an angle chosen from a uniform distribution between $0°$ and $360°$. To do this, we extracted larger patches by a factor $\sqrt{2}$ in each axis of the image so that upon rotating we can still extract the patch size of interest without having to pad the patches. After applying the rotation to this larger patch, the central 36×36 patch was cropped and saved. Three-fold more data than our initial number of positive patches was generated

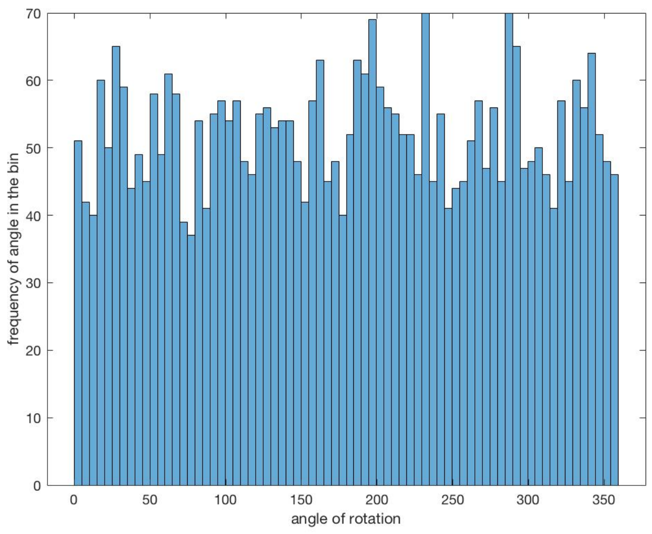
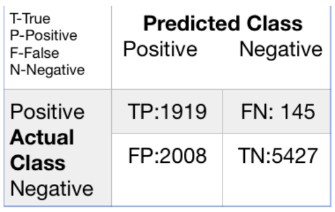


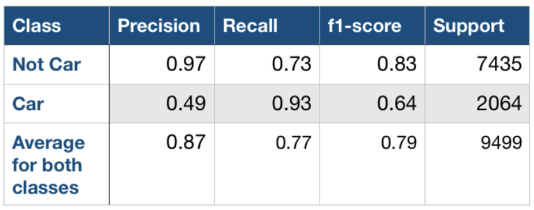
Figure 3.3: Histogram showing distribution of angles at which (one of the) datasets was augmented.

by looping over the training set. A histogram showing the distribution of angles in bins of 5° is shown in fig. 3.3.

Results



T-True P-Positive F-False N-Negative	Predicted Class Positive	 Negative
Positive **Actual Class**	TP:1919	FN: 145
Negative	FP:2008	TN:5427



Class	Precision	Recall	f1-score	Support
Not Car	0.97	0.73	0.83	7435
Car	0.49	0.93	0.64	2064
Average for both classes	0.87	0.77	0.79	9499

Figure 3.4: Summary of Precision, Recall, F_1 score and the confusion matrix of the SVM trained and tested on augmented data shown here for a probability threshold of 0.5

The linear SVM performed better classification on the aligned patches as compared to randomly rotated ones. This is as per the hypothesis because the linear SVM is not inherently a rotationally invariant classifier which means it will treat each patch even if it is the same car but at different angles as two unrelated patches. Learning a decision boundary for so many more points is naturally a harder task. It could be argued that the training was also done with a three-fold larger dataset which might take care of the greater complexity of the task. However, given the results, we hypothesize that the task was simply not linearly separable for an SVM to learn a good decision boundary.

3.3.3 Hard Negative Mining

So far, the SVM has been learned using a small and randomly sampled number of negative examples. However, in principle, every single patch that does not contain the object can be considered as a negative sample which is sometimes too many to be used in practice. Random sampling is not as effective because the most interesting

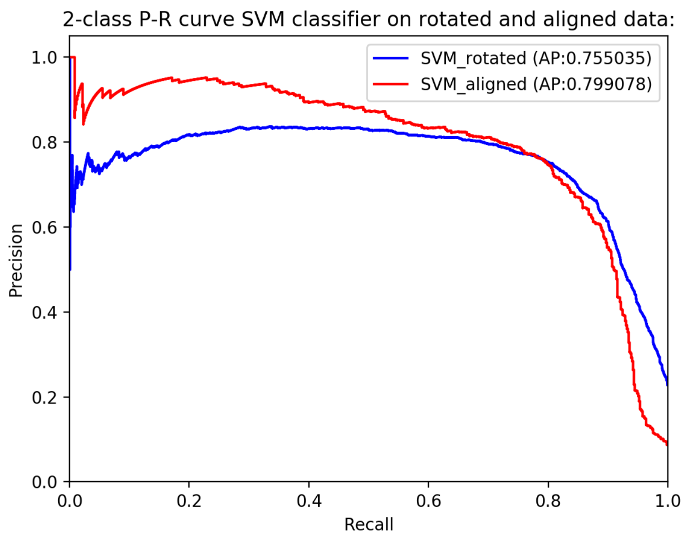


Figure 3.5: PR curves comparing SVM performance on rotated patches and aligned patches for various thresholds.

(confusing) negative samples are a small subset of all the possible ones[18].

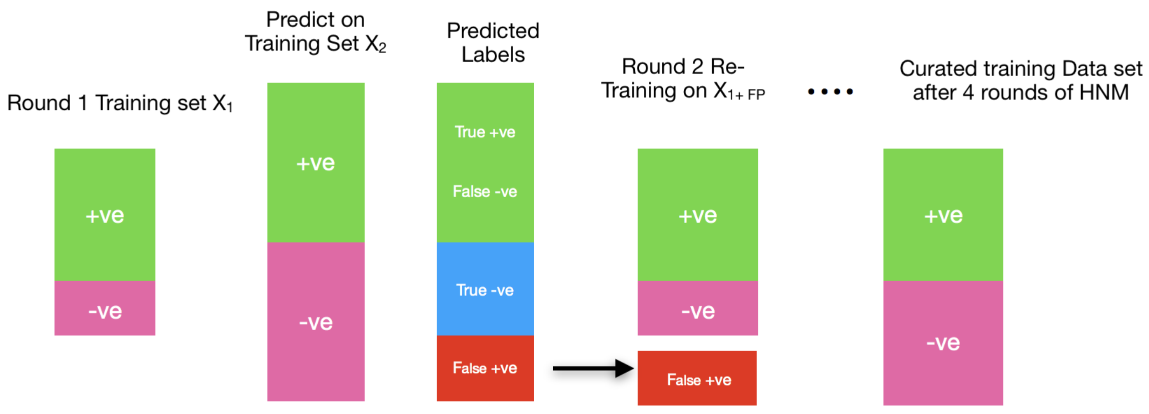
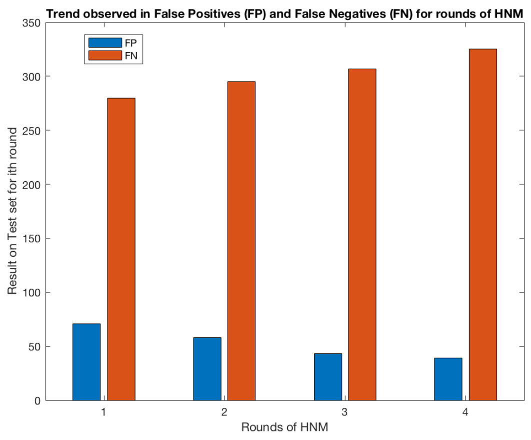


Figure 3.6: A diagrammatic way of showing how HNM curates a training set with key negatives from a larger set of all negative instances available.

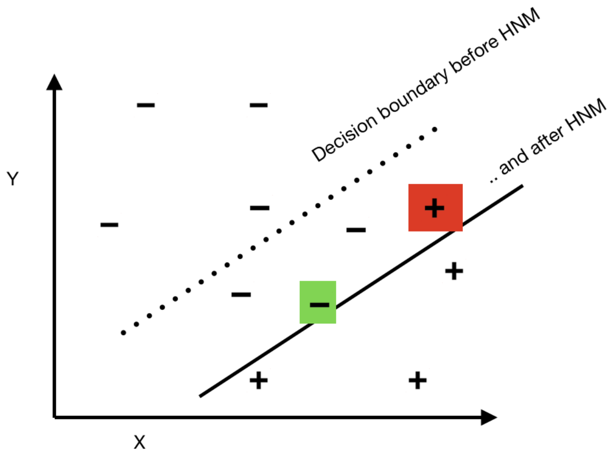
Hard negative mining is a technique that allows finding a small set of key negative examples as illustrated in fig. 3.6. We make the model focus on the difficult negatives - those training samples close to the boundary - and encourage the model to get them right. This is useful when we have a few positive examples and several times more negative examples as is our case where we can randomly sample the surroundings of the radar scene and so it is quite likely that the negative samples picked are in fact very easy for the classifier. Thus, when training, the model spends most of its time getting the easy negatives right because they are in multitude.

We start by training a model with a very small number of negatives and then we alternate between evaluating the model on the training data to find erroneous responses and adding the corresponding examples to the training set as shown in fig.3.6.

Results



(a) The outcome of HNM was a decrease in FP rate but an increase in FN rate round after round after the SVM was re-trained on the top 10% false positives and then re-tested on the same dataset.



(b) Hypothesis for results obtained: Green indicates a slightly anomalous negative example that would have been classified as a false positive before tightening of decision boundary but after HNM is a true negative. Thus HNM can reduce the number of false positives. Red indicates an anomalous example that would have been classified as a true positive before HNM but now is a false negative thus HNM increases the number of false negatives.

Figure 3.7: HNM results and a hypothesis for why we observed this are illustrated above.

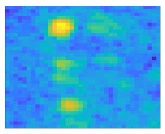
On the test set, we see that after each round, the number of false positives has decreased but the number of false negatives has increased. This was initially surprising as we had hypothesised that HNM would help select some key negative examples so that the SVM hyperplane can be drawn so as to account for the hardest cases thus tightening the boundary.

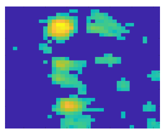
However, this worsening of classifier performance due to HNM can be explained if all of the training data that we have is not linearly separable in the first place. If this is the case then there is no single hyperplane that can be drawn to give us zero training loss for the C-hyperparameter value used. Therefore HNM might reduce false positive rate but increase false negative rate as a result of the re-trained boundary being drawn closer and closer to the positive class portion of the hypervolume after each HNM round.

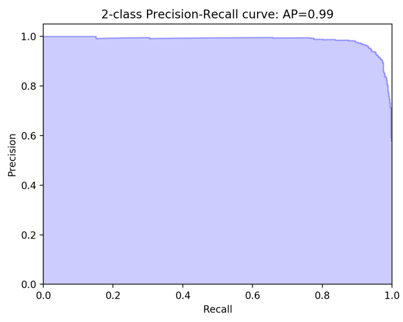
Linear Separability: The training loss will tell us how linearly separable is the data. Out of 15,510 training examples, if we train and test on the same set we get 608 misclassifications (i.e. 3.9 % misclassifications). This is for an optimally chosen regularisation hyper parameter value 'C' of 0.05. If the regularizer was made very high C = 100 (i.e. hard margin classifier) the training misclassifications were 11%. We thus confirm that the entire training dataset is not linearly separable indeed adding impetus to the justification for HNM provided above.

3.3.4 Denoising by Static Thresholding

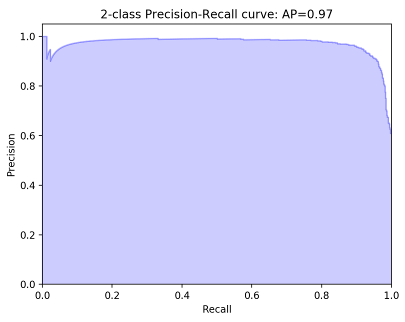
In an effort to reduce speckle noise, an experiment of introducing a static threshold in our grayscale images with intensity range 0-1 was performed. Empirically we found a threshold intensity of 0.4 to preserved the most relevant information about the structure of a car. This means all the intensities below 0.4 were set to 0. Both


(a) Original patch


(b) Thresholded patch



(c) Not-Thresholded PR curve



(d) Thresholded PR curve

Figure 3.8: PR curves for thresholded and original patches.

these PR curves were generated for training and testing on the radar data taken from the same place but split such that the training set is separate from the test set. The performance of thresholded patches was 2% worse than original patches as seen in fig. 3.8 and we conjecture this is because the nature of noise is not captured by a static threshold. Noise can often be of high intensities and the car in the patch can be of low intensities (further away from the sensor they are).

Secondly, we notice that the performance (average precision 0.97) is much higher than the AP of 0.85 for the varied datasets used in the experiment 3.3.1. This gives us insight that the problem of generalising a classifier to differentiate a car patch over varying locations is much harder than to do that for the same/similar dataset.

This is also a good place to compare the work of the 4yp by Weston[27], which also used a linear SVM. They obtained a precision of 90% for 80% recall at the operating point while we get a precision of about 80% for 80% recall. Comparing the two is not a fair competition because of different data splits and datasets used, however, one reason for their better performance could be attributed to a very similar dataset being used in both training (namely Tax-Rank-2) and testing (namely Taxi-Rank-3) as hinted to us in this experiment.

3.4 Success and Failures cases

In this section, we conduct a qualitative study of the SVM classifier. First, we shall visualise instances from the test set that the SVM could correctly classify albeit just narrowly making the cut. This might shed light on what kind of instances are difficult for the SVM and whether they are also difficult for the human eye to distinguish?

We choose a threshold of 0.5 to 0.6 to categorize a correctly classified positive instance as "hard true positive". We choose a threshold of 0.4-0.5 to categorize a correctly labelled negative instance as "hard true negative". Intuitively these can be thought of as cases where the SVM had low confidence in making the correct prediction.

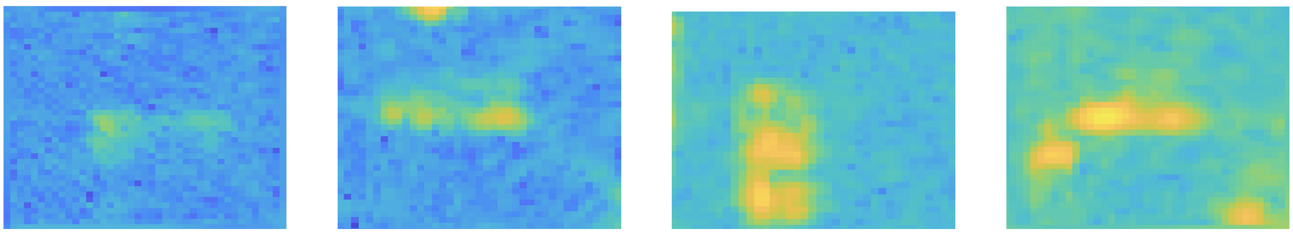


Figure 3.9: Easy True Positives. There are 7,316 in total.

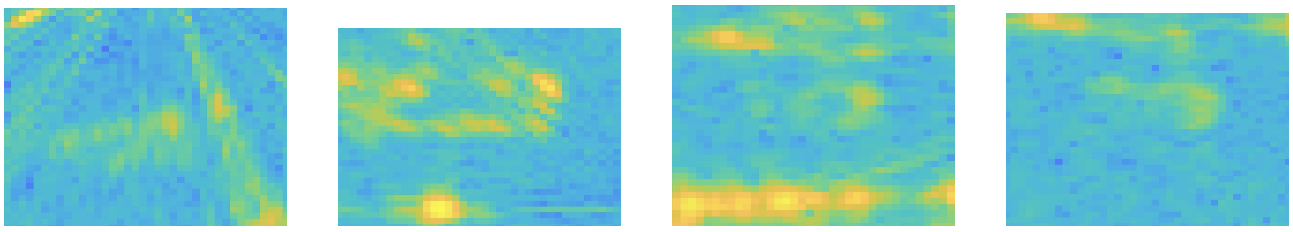


Figure 3.10: Hard True Positives. There are 192 in total.

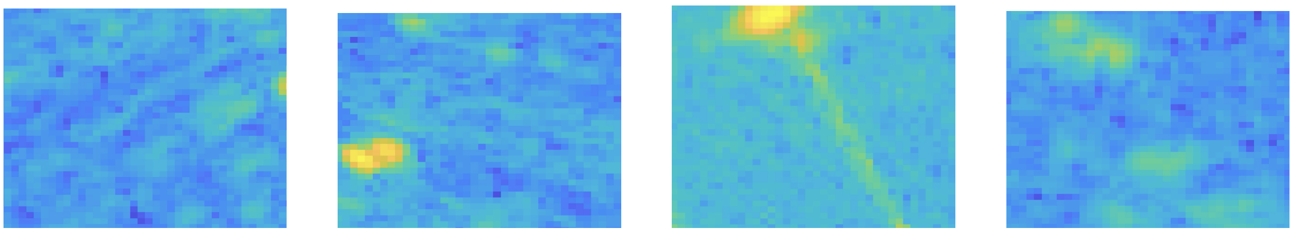


Figure 3.11: Easy True Negatives. There are 7,201 in total.

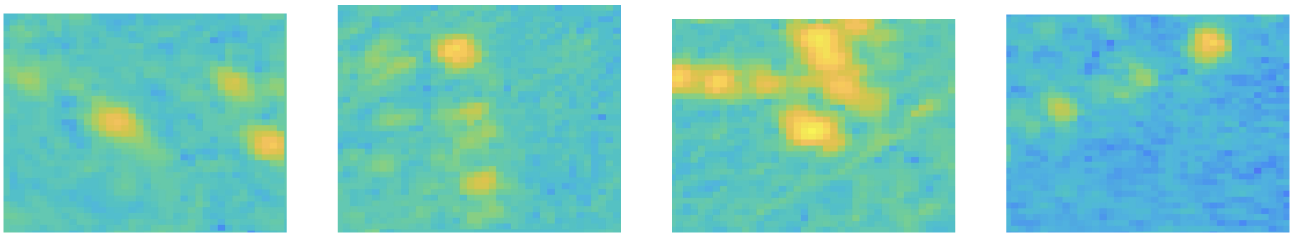


Figure 3.12: Hard True Negatives. There are 215 in total.

From fig 3.12 we can observe the presence of artefacts such as boundary effects (seen in the third image in the second row) due to surrounding objects entering the patch. Noise artefacts from multiple scattering objects are visible in the first image in the second row. Hard true-negatives are quite difficult to tell apart from positives, but this is simply due to randomly sampling similar objects on some occasions. Indeed, it is also not trivially simple for a human to reason the category to which these hard examples belong. Finally, let us also visualise misclassifications of the SVM.

Again, we see nearby objects falling in the patches in some false negatives (first image) and the presence of noise artefacts (fourth image). It is quite clear that this classification problem is not trivial and the distinguishing features are not apparent easily. We now are motivated strongly to look at algorithms that can extract abstract

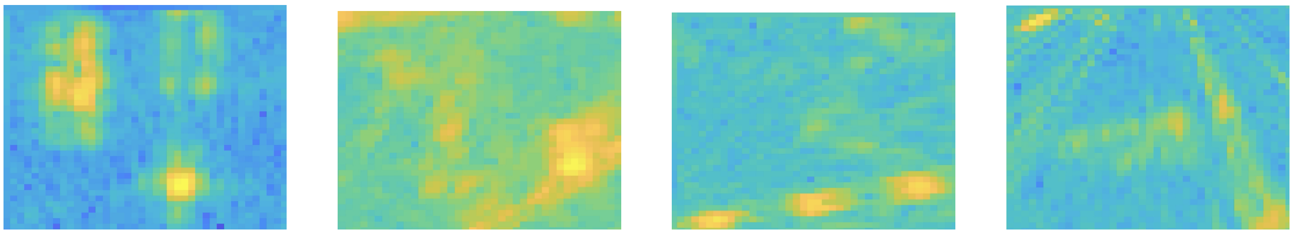


Figure 3.13: Chosen Hard False Negatives. There are 323 in total. Note that they visually resemble true negatives.

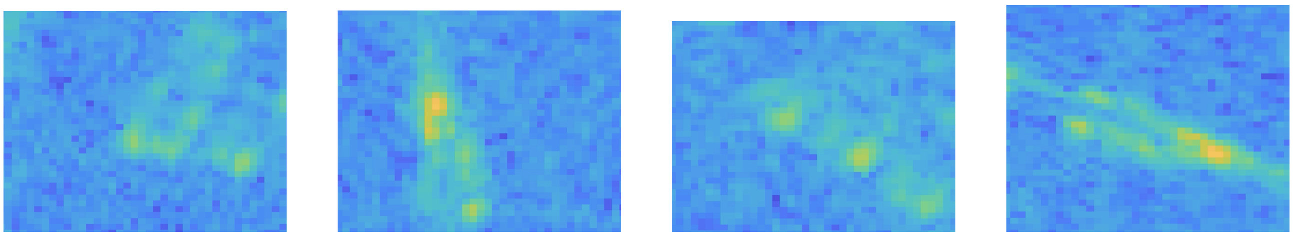


Figure 3.14: Chosen Hard False Positives. There are 285 in total. Note that they visually resemble true positives.

features that can aid in distinguishing difficult cases like the ones that we have visualised.

3.5 Conclusion

In this chapter, we have used raw pixel intensities as features of radar patches to perform classification. A soft margin linear SVM was trained and an optimal C hyperparameter that controls the width of this margin was selected via cross-validation. Further, Hard Negative Mining (HNM), a popular technique to curate training data to select key negative examples was used. Lastly, a qualitative understanding of the data was gained by visualising examples of correct and incorrect SVM classifications.

We noticed that the data was not trivial for even a human eye to classify which suggests that the most distinguishing features of each class may not lie in raw pixel intensities. Thus, we look towards algorithms that can extract more descriptive features for classification.

We can hand-engineer features suitable for our data or use popular features such as HoG (Histogram of Gradients). We also have the option of applying the kernel-trick which transforms the features into a high dimensional space such that it is easier to apply a linear classifier. The choice of the feature descriptor and transformation requires intuition about the structure of data in high dimensions and is not trivial. Thus, we refrain from this approach and instead turn towards deep learning which will “learn” optimal filters and apply them to our raw pixel input to obtain abstract features from the image. Deep learning classifiers will have a non-linear decision boundary which may be a powerful tool to separate that data which the linear SVM was not able to. Our work in this chapter and the SVM results obtained after using best training practices will serve as a baseline for future approaches, both in the case of aligned car patches and randomly rotated patches.

Chapter 4

CNN Based RADAR Classification

4.1 Motivation

In this project so far, we have set the stage for employing deep learning by making a pipeline to obtain training data of radar image patches and laying down a baseline for classifying them using a linear SVM using raw intensities as features both for aligned and randomly orientated car patches. Quantitive analysis showed that the randomly orientated case was a significantly more challenging task with a significant decrease in classification accuracy. We hypothesise that a greater classification accuracy can be achieved by using a more expressive feature representation than raw intensity values alone.

Deep learning has enjoyed success over other machine learning algorithms in vision classification tasks. Now, we turn towards Convolutional Neural Networks (CNNs) for classification with the expectation that they are able to learn abstract radar image features. CNNs apply a non-linear decision boundary to classify these extracted features, thus making them a powerful classifier.

4.2 Theory

CNNs can be viewed as a composition of functions where every layer is the result of a function's operation on the previous layer's output. Commonly used function operators are spatial-convolution, ReLU and max pool. This compositional nature allows the CNN to "learn" features in a hierarchy: low level features such as in our case say the edges of car blob in the patch and then the higher level features, for instance, say the shape of the car (which is very often an L shaped structure) at different orientations. Having extracted features, the network maps them to a scalar output indicating the class. A loss function must be decided between the predicted class and the actual class such that it penalises misclassifications and rewards correct ones. Then, minimimising this loss function with respect to the weight parameters of all of the composite functions results in learning optimal weights for the task.

4.2.1 Encoder

Convolution is an operator that applies a weighting kernel (K) to a function to produce a feature map. In our case we treat each image (I) which is a radar patch of dimensions 36×36 as a spatially varying function of pixel intensities. Mathematically, a feature map S is computed at each pixel position (i,j) using a kernel K applied as a sliding window with m and n acting as iterators:

$$S(i,j) = \sum_{m} \sum_{n} I(m,n)K(i-m, j-n) \tag{4.1}$$

While a convolution with an image can be written as a matrix multiplication with a large and sparse Convolution matrix, the kernels are much smaller than the image size itself and hence we need to store much fewer parameters. We shall look at the idea of convolution matrices in section 4.2.6 while discussing the convolution transpose operation. Note that convolution is a linear operator and so we need to add a non linearity between successive convolutional layers otherwise it is equivalent to having just one single convolutional layer. For this reason we use a nonlinear activation function such as rectified linear unit (ReLU)[17] defined as $\max(0, S(i,j))$ where $S(i,j)$ is a certain location of the feature map. ReLU simply clamps all negative activations to zero and leaves positive activations unchanged.

Max pool is an operator that replaces a specified window of the activations with a single maximum value among the values in that window. It also does downsampling. Max pool provides a small degree of translational invariance because max pooling units are sensitive to only the maximum value in the neighbourhood, not its exact location. This is an important property for classifying our radar car patches because often the position of the car in the patch is not perfectly centred as seen from the visualisations in the previous chapter.

4.2.2 Loss

We wish to obtain a probability that a test sample belongs to the predicted class. Logistic loss allows this as it outputs the logistic function (a mapping from any $\mathbb{R}$ to a number between 0 and 1) of the resulting score: $\hat{p} = \sigma(f_n(\theta^T \mathbf{x}))$ where θ are the model weights, x the features of the input image and f_n the last function in the composition of functions in the CNN. So this model predicts 1 if $\theta^T \mathbf{x}$ is positive and 0 if negative. We want a cost function that grows very large if the model estimates a probability close to 0 for a positive instance and also very large if the model estimates a probability close to 1 for a negative instance. On the other hand, loss close to 0 for when $\hat{p}$ is close to 0 for a negative example or close to 1 for a positive one. Applying a log to $\hat{p}$ satisfies all the aforementioned desirable properties of the cost function. So cost $c(\theta)$ for one sample is:

$$c(\theta) = \begin{cases} -log(\hat{p}) & y = 1 \\ log(1-\hat{p}) & y = 0 \end{cases} \tag{4.2}$$

The cost function over the whole training dataset is simply the average cost over all the m instances where $J(\theta)$ is the total cost:

$$J(\theta) = -\frac{1}{m_i}\sum_{n=1}^{m}[y^{(i)}log(\hat{p}(\theta)^{(i)}) + (1-y^{(i)})log(1-\hat{p}(\theta)^{(i)})] \tag{4.3}$$

4.2.3 Learning by Optimisation

Our goal for learning is to update each of the weights (θ) in the network so that they cause the actual output to be closer to the target output, thereby minimizing the error $J(\theta)$.

$$\frac{\partial J}{\partial \theta_{n-1}} = \frac{\partial J}{\partial f_n}\frac{\partial f_n}{\partial \theta_{n-1}} \tag{4.4}$$

And a weight update in the same way as gradient descent on all connection weights using the error gradients calculated above:

$$\theta_{t+1} \leftarrow \theta_t + \eta\frac{\partial E}{\partial \theta_t} \tag{4.5}$$

To understand the problem with the above calculation, focus on the size of each derivative. If the size of the penultimate hidden layer x_{n-1} is $H_{n-1}W_{n-1}C_{n-1}$ and that of the layer f_n is $H_nW_nC_n$ then the size of $\frac{\partial f_n}{\partial x_{n-1}}$ is $H_{n-1}W_{n-1}C_{n-1} \times H_nW_nC_n$ which for $H_n = W_n = 16$ and $C_i = 256$ is $65,536 \times 65,536 = 4$ billion entries and would take 16GB of memory to store.

4.2.4 Back Propagation

Backpropagation is a memory efficient method of computing the derivative of function compositions. We have:

$$\frac{dvec(\boldsymbol{f}_n \circ \boldsymbol{f}_{n-1})}{d(vec\boldsymbol{x}_{n-2})^T} = \frac{dvec\boldsymbol{f}_n}{d(vec\boldsymbol{x}_{n-1})^T}\frac{dvec\boldsymbol{f}_{n-1}}{d(vec\boldsymbol{x}_{n-2})^T} \tag{4.6}$$

By using a back projection function which is just a dot product (projection) of a layer (a tensor) on its previous layer (also a tensor) we can reduce this to a scalar. Then taking a derivative of this scalar wrt to a tensor greatly reduces the dimension of the derivative. This is shown for the output f_n (a scalar usually) and penultimate layer x_{n-1} as:

$$\frac{dvecf_n}{d(vec\boldsymbol{x}_{n-1})^T} = \frac{\langle 1, f_n\rangle}{d(vec\boldsymbol{x}_{n-1})^T} = d\boldsymbol{x}_{n-1} \tag{4.7}$$

Similarly for the previous layer where we are dealing with a derivative of a tensor with another tensor :

$$\frac{dvecf_n}{d(vec\boldsymbol{x}_{n-1})^T}\frac{dvec\boldsymbol{f}_{n-1}}{d(vec\boldsymbol{x}_{n-2})^T} = (\text{vec d}\boldsymbol{x}_{n-1})^T\frac{dvec\boldsymbol{f}_{n-1}}{d(vec\boldsymbol{x}_{n-2})^T} = \frac{\langle \boldsymbol{dx}_{n-1}, \boldsymbol{f}_{n-1}\rangle}{d(vec\boldsymbol{x}_{n-2})^T} \tag{4.8}$$

The intermediate results dx_{n-1} and dx_{n-2} have the same size as x_{n-1} and x_{n-2} instead of what would have been a Jacobian matrix with dimensions that are a product of dimensions of each of the two layer tensors.

4.2.5 Representation learning

In this project, we are able to extract large amounts of negatively labelled data but have relatively fewer positively labelled samples. This motivates the question that can we learn a representation of radar images which can help in the classification task? Using autoencoders, it is possible to pre-learn low-level features and re-use them. An autoencoder shrinks the input down to an encoded representation and then inverts the encoded portion to mimic the input as shown in fig 4.1. Learning an identity function like this might not seem useful but the key is that the encoded representation is compressed, thus encouraging the network to learn essential features that are needed to reconstruct the image. Such an architecture is called Undercomplete Autoencoder[9]. The filters that we have learned might be useful in the classification task later. Therefore we look towards representation learning to learn a distribution of the input data in a self-supervised fashion.

4.2.6 Autoencoder

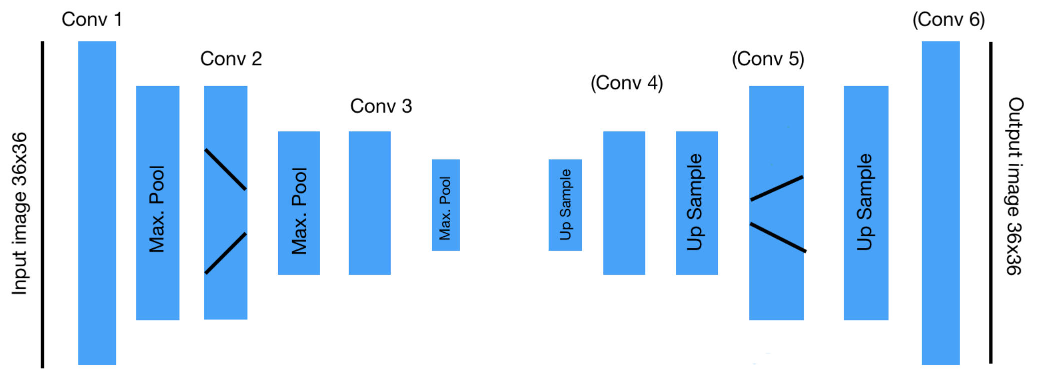


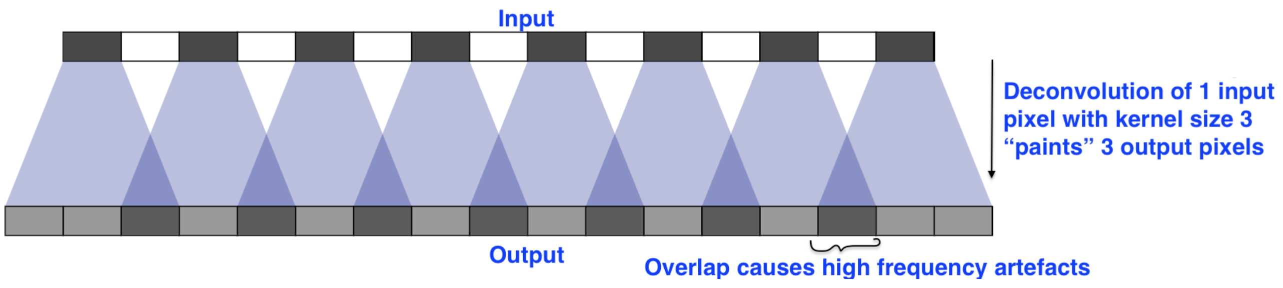
Figure 4.1: Our undercomplete-autoencoder architecture showing the Encoder and the Decoder. The converging lines in conv2 layer show the shrinking action of the encoder while the diverging lines in conv5 show the enlarging action of the decoder.

An autoencoder consists of an encoder and a decoder. The encoder shrinks the input layer by layer using convolutions and max pools as we show in section 4.2.1. The decoder inverts the operations of the encoder using convolution-transpose and up-sampling back to the same dimensions as the input. The loss is defined as a difference measure between each input pixel and output pixel so as to “ask” the network to replicate the input image.

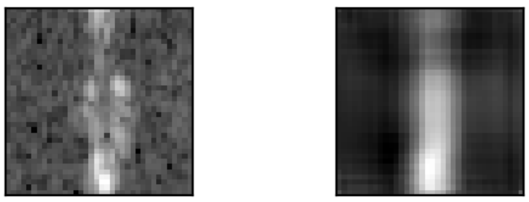
Decoder

Having discussed how an encoder works in section 4.2.1, we need a way to go from a lower dimensional encoded image to a full sized one. This is achieved either using deconvolution layers or by performing convolution followed by upsampling. We explore both and motivate the reasons for choosing the latter.

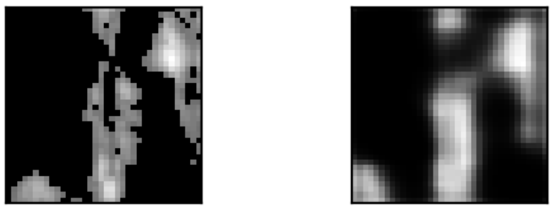
Convolution Transpose: The convolution operation can be represented as a matrix multiplication of image with a sparse matrix **C**[9]. It can be shown that the backward pass (“return” stage of backpropagation) of a convolution operation is represented as this convolution matrix transposed C^T[4]. Similarly, for a convolution-transpose operator, the forward pass is a matrix multiplication C^T and the backward pass is a multiplication with **C** [4]. Intuitively, the convolution transpose operation maps every point in the compressed image to a square in the larger one. This is shown in fig 4.2a. In it, we can see that there is an uneven overlap which results in the checkerboard pattern. We experimented with the convolution transpose layer which showed a similar checkerboard pattern as seen in the fig 4.2. The uneven overlap tends to be more extreme in two dimensions[19].


Input
Deconvolution of 1 input pixel with kernel size 3 "paints" 3 output pixels
Output
Overlap causes high frequency artefacts

(a) Deconvolution operation showing the creation of checkerboard artefacts[19].



(b) Checkerboard artefacts when radar patches were passed into the AEC.



(c) Checkerboard artefacts when thresholded radar patches were passed into the AEC.

Figure 4.2: The presence of checkerboard pattern was observed in image reconstructions if the convolution-transpose operator was used. This motivated the use of convolution followed by upsampling.

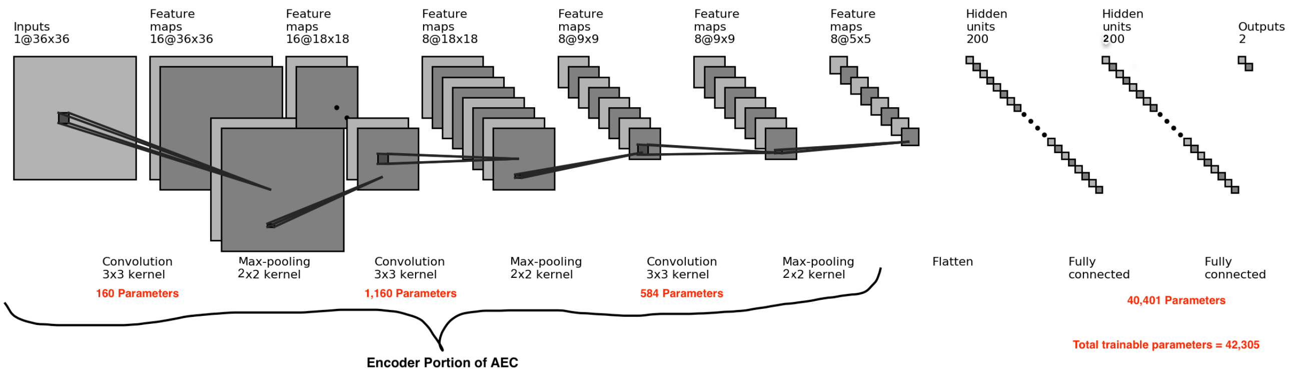
Convolution followed by Upsampling: An alternative approach is to separate out up-sampling to a higher resolution from convolution. We can resize the image by using nearest-neighbour interpolation, bilinear interpolation or just repeating rows and columns[25]. We choose the latter method to resize the image and then apply convolution that preserves dimensions[11]. The results of reconstruction are seen in fig 4.8.

Loss

Three different loss functions namely binary-cross entropy, mean-squared-error and Kullback-Leibler-divergence were tried[15]. Training loss values at the end of training were checked (lower the better) along with visualising if the image reconstructions were faithful in order to ascertain a good choice of the loss function. The loss function chosen is mean square error between the input image pixels p and the reconstructed pixels $\hat{p}$. So, for a total of N pixels in an image, the loss function is $\text{Loss}(\hat{p}) = \frac{1}{N}\sum_1^N (p-\hat{p})^2$.

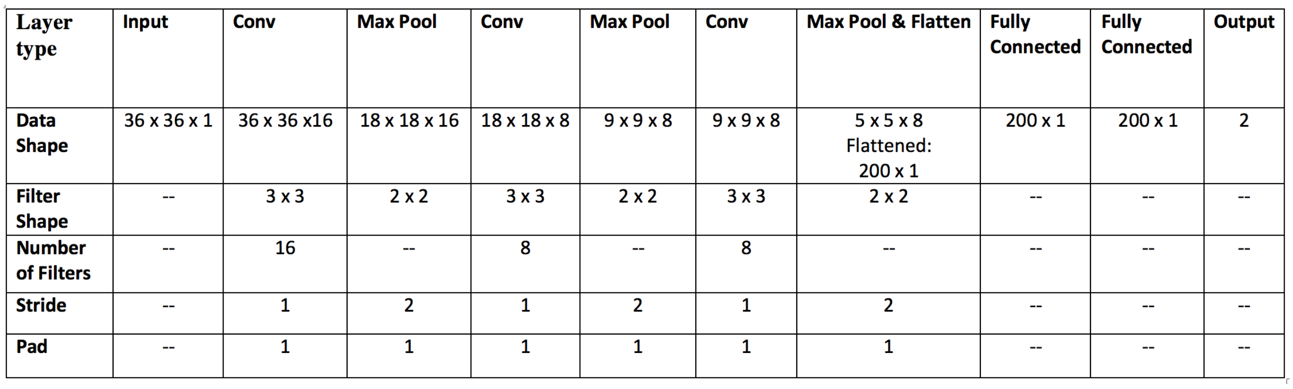
4.3 Classifier Architecture

The architecture we use is inspired by LeNet-5 [13] which is a pioneering and relatively straightforward network that was first used on 32×32 pixel greyscale input images to classifies digits. LeNet-5 has two pairs of convolution and maxpool layers followed by a fully connected layer. Our architecture has *three* pairs of convolutions and max pool followed by two fully connected layers and convolution filter size of 3×3 instead. One reason to use LeNet-5 as an inspiration was the similarity between the input sizes of digits used by LeNet and our car patches. There is also a resemblance in the structure of digit images and car patches - they are both grayscale with a brighter structure in the center and relatively dark background. Another rationale was that this is a simpler architecture than many other established architectures available. Keeping in mind that our CNN is for a robotics application, we require close to real-time computations which becomes difficult on deeper, complex architectures. We acknowledge that various iterations in architecture designs are possible as future work to improve the performance, but that is outside the scope of this project.



(a) Diagram[a] of the architecture used for our classifier showing the Encoder and number of trainable parameters.

[a]This figure is generated by adapting the code from `https://github.com/gwding/draw_convnet`



Layer type	Input	Conv	Max Pool	Conv	Max Pool	Conv	Max Pool & Flatten	Fully Connected	Fully Connected	Output
Data Shape	36 x 36 x 1	36 x 36 x16	18 x 18 x 16	18 x 18 x 8	9 x 9 x 8	9 x 9 x 8	5 x 5 x 8 Flattened: 200 x 1	200 x 1	200 x 1	2
Filter Shape	--	3 x 3	2 x 2	3 x 3	2 x 2	3 x 3	2 x 2	--	--	--
Number of Filters	--	16	--	8	--	8	--	--	--	--
Stride	--	1	2	1	2	1	2	--	--	--
Pad	--	1	1	1	1	1	1	--	--	--

(b) Table summarising the various layers and their hyperparameters used such as filter sizes, stride, padding and dimensions.

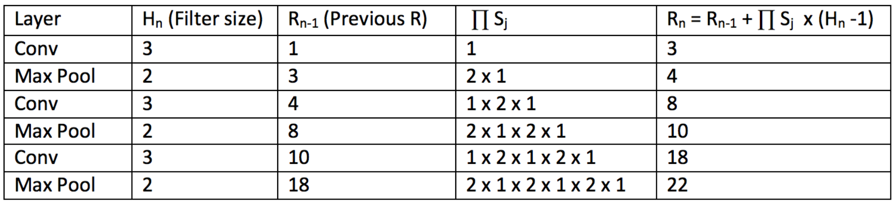
Figure 4.3: Our architecture is inspired from LeNet-5 [13] because of the resemblance in the data dimensions and appearance of our grayscale radar car patches with grayscale low-res handwritten digits. In addition, the speed and simplicity due to fewer parameters was appealing. Our architecture is illustrated and summarised above.

Receptive field calculation

The receptive field of element $\boldsymbol{x}_{vuc}$ of tensor $\boldsymbol{x}$ in the CNN is the subset of the input image $\boldsymbol{x}_0$ that can affect $\boldsymbol{x}_{vuc}$. The receptive field of the output from the classifier was calculated to ensure that the filter sizes and overall architecture adequately covers the size of interest of the input radar patch. We can calculate Receptive field (R) from:

$$R_n = R_{n-1} + (\prod_j S_j)(H_n - 1) \tag{4.9}$$

The receptive field of the representation after 3 pairs of convolution and max pool is of size 22×22 but after



Layer	H_n (Filter size)	R_{n-1} (Previous R)	$\prod S_j$	$R_n = R_{n-1} + \prod S_j \times (H_n - 1)$
Conv	3	1	1	3
Max Pool	2	3	2 x 1	4
Conv	3	4	1 x 2 x 1	8
Max Pool	2	8	2 x 1 x 2 x 1	10
Conv	3	10	1 x 2 x 1 x 2 x 1	18
Max Pool	2	18	2 x 1 x 2 x 1 x 2 x 1	22

Figure 4.4: Table summarising the calculation of the receptive field of the encoder.

applying fully connected layers (of which there are two), the receptive field of the output naturally covers the entire image patch of 36×36. While it maybe argued that a deeper network with receptive field bigger than the image must be tried, it is not clear that this is mandatory because the fully connected layers already are able to cover the entire image.

4.4 Experiments

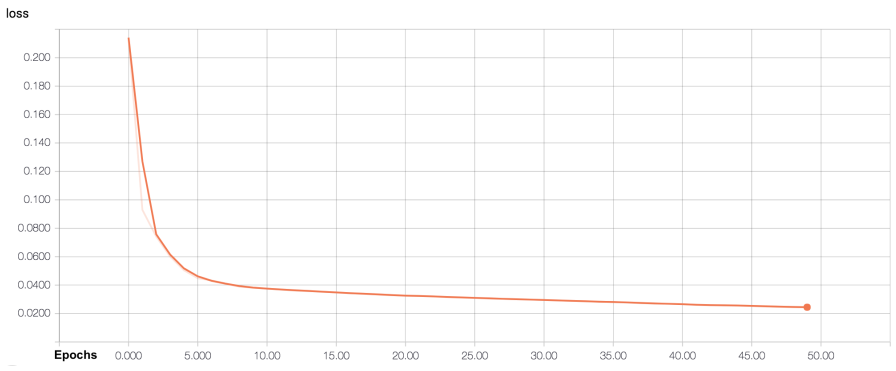
In order to investigate the performance of the proposed architecture on our dataset, several experiments have been designed. To compare with the baselines set in the case of linear SVM, we start with classifying car patches that are all aligned. As our actual goal is to detect cars at various angles using a sliding-window, we artificially rotate the patches randomly and again classify those. The two above cases shall be juxtaposed to investigate any differences in their performances. As we have fewer positive samples than typical for CNNs but plenty of negatives. we explore the potential of unsupervised pretraining using an autoencoder in improving our classification performance on rotated car patches. Lastly, we explore the possibility of building a regressor to predict the orientation angle of cars in patches.

4.4.1 Aligned Patches

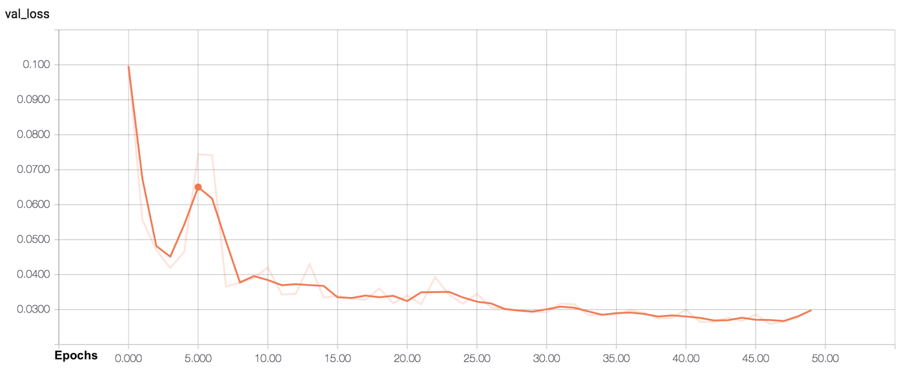
Dataset

The dataset trained on was identical to the SVM in the experiment in chapter 3.3.1. We trained and tested on image patches of size 36×36. The training set consists of a total of 7,772 car patches and 7,738 negative patches from different datasets as mentioned in chapter 2.4. The division of positive and negative examples in the training set was roughly $50 - 50\%$ so that we do not bias the model against one category. The training set was further randomly split up into a validation set with the ratio of 33% validation and 77% for solely training. The test set was gathered from a completely separate location so as to be indicative of the model's generalisation ability. It consisted of 688 positives and 7,435 negatives. Thus, the division of samples in the test set was, 92% negative samples and only 8% positive samples in order to simulate reality - a usual radar scan will contain much fewer instances of cars than surroundings.

Training and Validation



(a) Training loss versus the number of epochs indicated when to stop learning.



(b) Validation loss although not monotonically decreasing, has the same trend and similar magnitude as training loss, suggesting no overfitting.

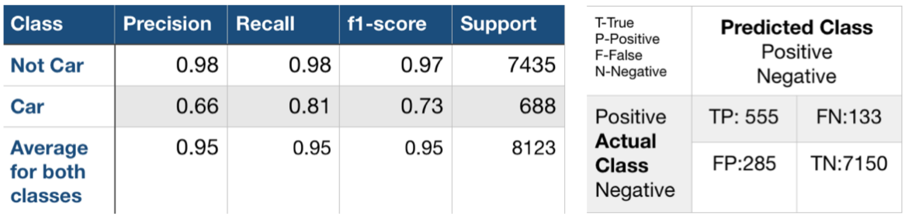
Figure 4.5: These plots were monitored to ensure there were no anomalies while training as well as later to verify that no significant overfitting has occured.

Each sample was labelled as either 1 (positive class) and 0 (negative class). The labels were stacked into a vector while the corresponding patches were stacked as a tensor. Training data was shuffled so that samples from the same class are not all consecutive, in case that has any biasing effect. Training was done using Ada Delta optimizer which dynamically adjusts the learning rate so that manual tuning is not required [29]. No regularisation penalty was applied. As can be seen in fig 4.5 the training loss starts to plateau after 40 epochs. We let the training continue until 50 epochs. The training and validation losses are not drastically different, with both plateauing to a value of 0.03 suggesting that there has not been overfitting[1]. Validation and training accuracies[2] were also comparable - both being close to 97%.

[1]When training loss is much lower than validation or testing loss, our model has learnt the training data too closely and will not generalise well.

[2]Accuracy in this case is a good evaluation metric because the training data is balanced

Testing Results



Class	Precision	Recall	f1-score	Support
Not Car	0.98	0.98	0.97	7435
Car	0.66	0.81	0.73	688
Average for both classes	0.95	0.95	0.95	8123

T-True P-Positive F-False N-Negative	Predicted Class Positive	Negative
Positive **Actual Class**	TP: 555	FN:133
Negative	FP:285	TN:7150

Figure 4.6: Summary of Precision, Recall, F_1 score and the confusion matrix of the CNN trained and tested on aligned data shown here for a probability threshold of 0.5.

Upon testing the CNN on aligned patches, we get 90% precision for 80% recall at the operating point chosen to be the threshold when the PR curve in figure 4.7a drops sharply. In comparison to the SVM tested on the same dataset, we have obtained a superior performance: Average Precision[3] (AP) of 87.2 Vs 79.9 for SVM. This result meets with our expectation that deep learning should perform better than the linear SVM. The reason for this is that the CNN is able to extract features and draw a nonlinear decision boundary to separate them.

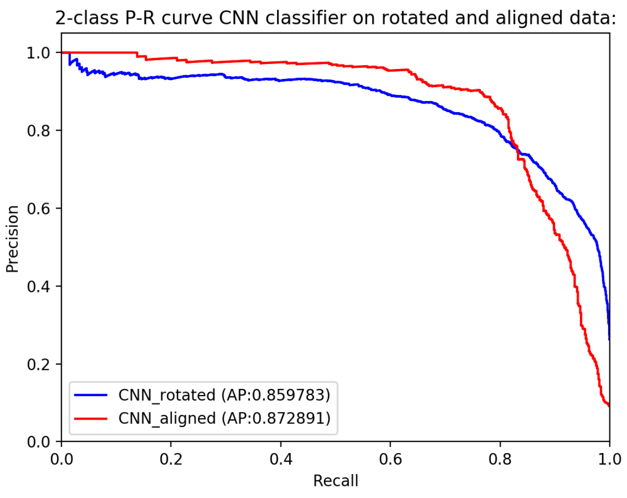
4.4.2 Randomly oriented patches

In this section, we investigate the CNNs performance on randomly rotated data because in reality, we aim to learn to classify cars at any orientation. The data augmentation process was identical to the one described in section 3.3.2. As a result of augmentation, the number of positive examples has been increased threefold to 23,316. The negative examples have been kept the same as before. The training procedure for the augmented data is identical to the one we described in the aligned experiment (section 4.4.1). We test this classifier on the same dataset as in the previous aligned experiment except that we augmented this test dataset as well.

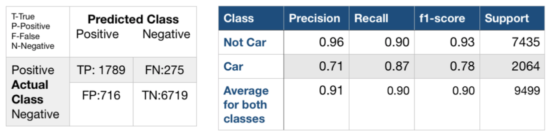
Results

Juxtaposing the two PR-curves as shown in fig 4.7a shows that the CNN does better in general for aligned patches (Average Precision = 0.873) compared to rotated patches (AP = 0.859). One hypothesised reason for this is that classifying rotated patches is beyond the CNNs model capacity. CNN is not inherently rotationally invariant i.e. there are no operators for preserving rotational invariance. It could be that the model capacity or the filter weights are used up in learning other features and not the rotational differences. Therefore, more training data would not necessarily take care of the problem. Future work could look at using spatial transformer networks [10] which gives the network the ability to actively spatially transform feature maps so that the model can learn invariance to rotation (among other transformations).

[3]One way to summarize a PR curve by calculating the area under the curve



(a) PR curves of the CNN tested on aligned data (shown in red) and on rotated data (shown in blue) are juxtaposed for comparison.



T-True P-Positive F-False N-Negative	Predicted Class Positive	Negative
Positive	TP: 1789	FN:275
Actual Class Negative	FP:716	TN:6719

Class	Precision	Recall	f1-score	Support
Not Car	0.96	0.90	0.93	7435
Car	0.71	0.87	0.78	2064
Average for both classes	0.91	0.90	0.90	9499

(b) Summary of Precision, Recall, F_1 score and the confusion matrix of the CNN trained and tested on rotated data shown here for a probability threshold of 0.5.

Figure 4.7: Results of the CNN on the test set are summarised - the table shows for the rotated data while the PR curve is a comparison of aligned and rotated data.

4.4.3 Unsupervised Pre-Training

Dataset

The training set for the autoencoder consisted of 99,214 unlabelled samples which is much more than the number used in the supervised classifier training. These were obtained by randomly sampling a radar scene to get as much unlabelled but unique (non overlapping) data as possible. The target output of the network was identical to the input tensor consisting of all the sample images stacked.

Training

Just like in the supervised case the Ada optimiser was used and training was continued for 50 epochs which was after the loss began to plateau to a magnitude of 0.006. Note that the pixel intensity is a floating point number between 0 and 1 hence a loss of this magnitude. Once trained, testing was done on a separate smaller (2,000 images) subset of the data and reconstructed images were visualised as shown in fig 4.8. There are two approaches to training[6] - 1) greedy layer wise (training done in stages) and 2) all layers at once i.e. the usual way of backpropagating with all layers present. Both these were tried. We outline the greedy approach below for clarity.

Greedy approach

The unsupervised feature learning algorithm $\mathcal{L}$ takes as input, a set of training samples and returns an encoder or feature function f. The raw input data is $\boldsymbol{X}$ and $f^{(1)}(\boldsymbol{X})$ is the output of the first stage encoder on $\boldsymbol{X}$.

Fine tuning is when the weights learned in the unsupervised task above are used as starting values for the updated weights learned in the supervised task. In that case, we use a learner $\mathcal{T}$, which takes an initial

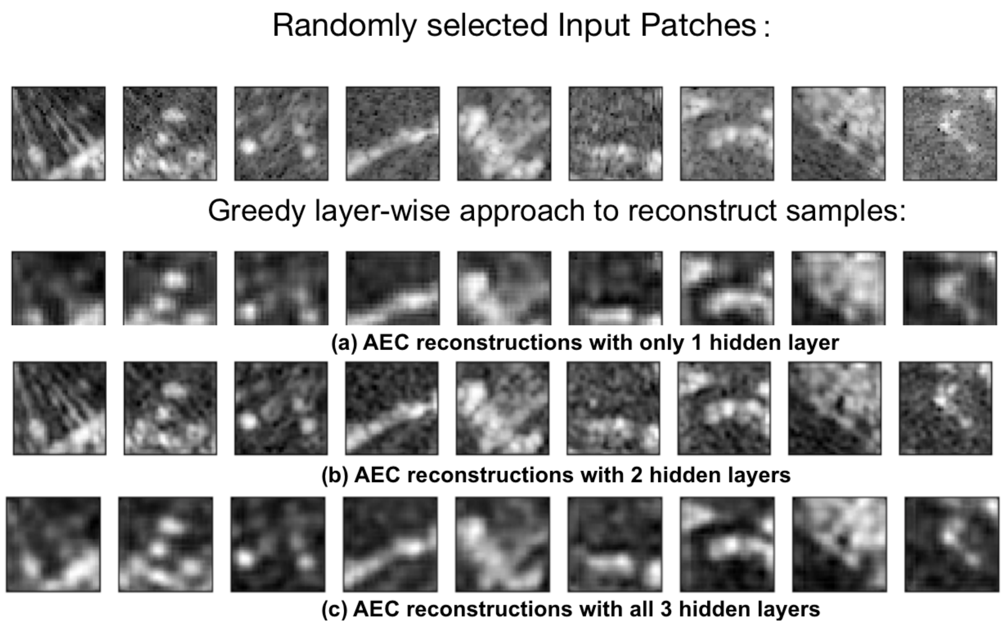

Randomly selected Input Patches :
Greedy layer-wise approach to reconstruct samples:
(a) AEC reconstructions with only 1 hidden layer
(b) AEC reconstructions with 2 hidden layers
(c) AEC reconstructions with all 3 hidden layers

Figure 4.8: Some randomly chosen unlabelled samples being reconstructed using the undercomplete AEC layer wise.

function f, input examples $\boldsymbol{X}$ and associated targets $\boldsymbol{Y}$ and returns a tuned function [9]. So, at the end of the

Algorithm 1 Greedy Layer Wise Pre-training algorithm[9]

```
1: $f \leftarrow$ Identity function;
2: for k = 1,...,m do                                ▷ Number of stages in the autoencoder is m.
3:     $f^{(k)} = \mathcal{L}(\hat{(X)})$;
4:     $f \leftarrow f^{(k)} \circ f$;
5:     $(\hat{X}) \leftarrow f^{(k)}(\hat{(X)})$
6: end for
7: if fine-tuning then
8:     $f \leftarrow \mathcal{T}(f, X, Y)$
9: end if
```

unsupervised learning we have learned the encoder function weights: $f \leftarrow f^{(m)} \circ f^{(m-1)} \circ \ldots \circ f^{(2)} \circ f^{(1)} \circ f_{identity}$. This we then fine-tune in a supervised way using target outputs as will be done in the next section4.4.4.

4.4.4 Supervised Fine-Tuning

An experiment of comparing the effect of pretraining was performed by training the same network architecture - one with randomly initialized weights while the other with initial weights loaded from the Autoencoder model that was trained in an unsupervised fashion. The classifier neural network then fine tunes these initialized weights in a supervised fashion.

On one hand, we chose to freeze the lower layer of the network and fine tune only the upper layers with the hypothesis that some features of the input data may have been learned by the autoencoder weights already and need not be fine tuned. On the other hand, we wished to test the hypothesis that the features would not directly be useful for the classification task and hence the initial weights were not frozen and all the layers could be fine-tuned. It must be noted that this Autoencoder was not trained in a greedy manner.

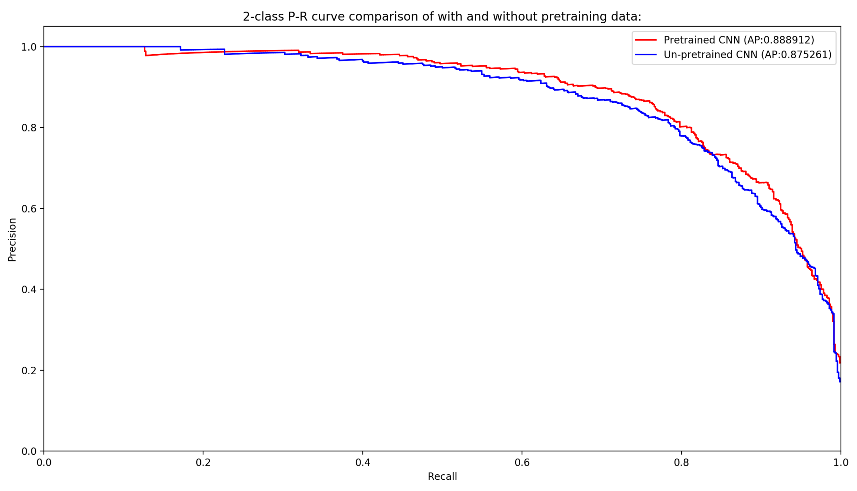


Figure 4.9: Comparing performance of classifier with and without pretraining; The former consistently does better.

Unsupervised pretraining helps in some cases because the choice of initial parameters for a deep neural network is thought to have a regularising effect on the model - in particular [6] claims that the initialisiation point is such that model parameters are restricted in a so called "basin of attraction" of the supervised fine-tuning cost function. We might expect pretraining to be more effective when the number of labelled examples is a small portion of the training data, which is our main motivation to use this semi-supervised approach. The idea behind this is that the model may have learned some features in the unsupervised phase which it uses in the supervised phase to perform better.

Results

Both approaches were tried and their PR-curves plotted in fig 4.10. While both approaches gave better results than just a classifier without any pretraining, there was not much difference between the two approaches. The

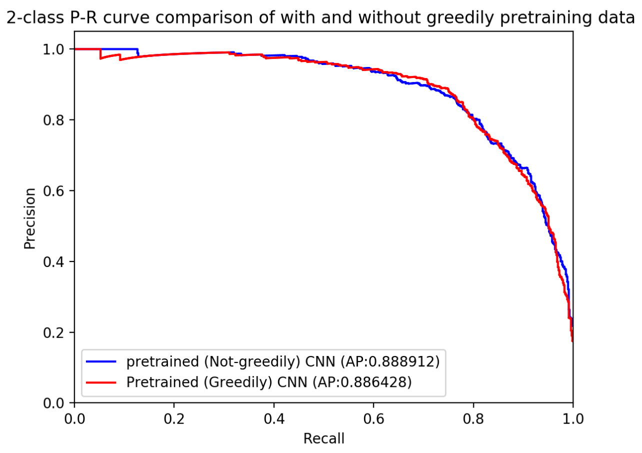


Figure 4.10: Comparing the greedy approach to pre-training and the one where all layers are simultaneously trained; they are nearly identical.

result of a pretrained CNN performing better than a randomly initialised CNN is as per the hypothesis based on the reasoning discussed in 4.4.4. The other result of the greedy layer-wise performance being not any better than all of the layers fine-tuned together.

4.4.5 Regression for predicting orientation of car

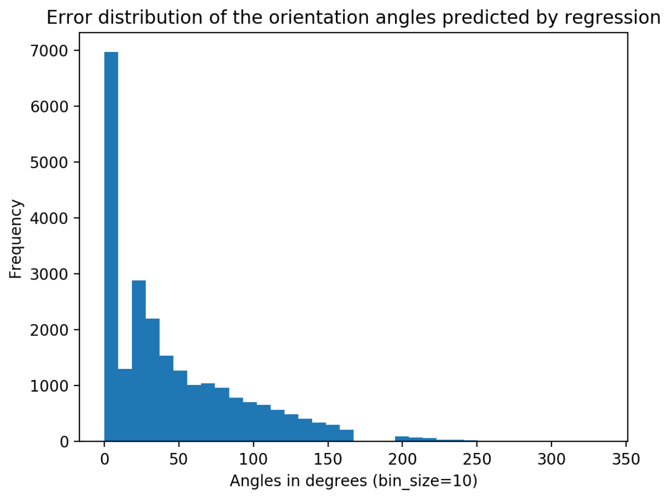


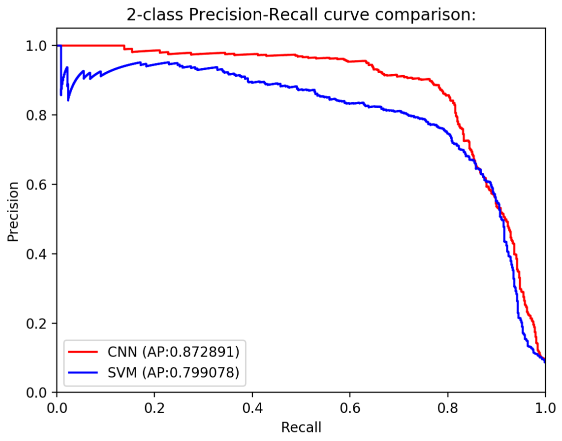
Figure 4.11: Histogram of the absolute error in predicted orientation of cars in degrees with a bin size = 10.

In addition to classifying a patch, we are also interested in knowing the orientation of the car in order to possibly use this information in tracking its direction and velocity. We augment the aligned car patches artificially and so keep a record of each training patch's orientation which we utilise here. Using the same classifier architecture and altering the loss function as well as number of outputs, the regressor was built for experimentation. Instead of two outputs predicting the probability of each class, we only have one that outputs an angle between 0° and 360°. The loss function is changed to mean squared error between the predicted angle and actual one. We plot the absolute error $|\theta - \hat{\theta}|$ and observe that most of the errors are concentrated in a range of 0°-10° i.e. ± 10° as shown in fig. 4.11.

4.5 SVM classifier Vs. CNN classifier

Upon juxtaposing the CNN PR-curve and SVM PR-curve on the same test sets, it is clear that CNN has a superior performance consistently (for all thresholds) and for both rotated as well as aligned cases. The test time taken on a CPU by the SVM was 128 seconds while the CNN took 2.8 seconds. Note that the SVM was made to predict probabilities which has the overhead of converting scores to probabilities using logistic regression. The CNN performance was so rapid because of its small size and just 42,305 parameters.

Note that the above PR-curves are the best baseline we could get in both the SVM and CNN case after the rigorous training procedure involving HNM, Hyper-parameter tuning for SVM and Unsupervised pretraining for the CNN. The reason for this better performance is that deep learning is able to extract features and apply a non linear decision boundary to classify them which is more powerful.



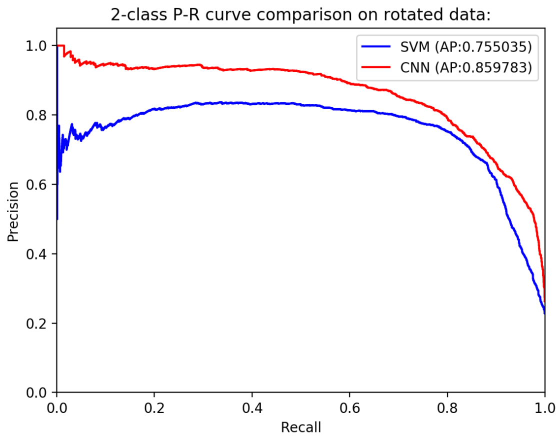


Figure 4.12: PR curves of SVM and CNN are juxtaposed here for cases of both aligned data (left) and rotated data (right). The CNN outperforms the SVM for each task.

4.6 Conclusion

At the beginning of this chapter, we had hypothesised that more expressive descriptors than simply raw intensities of radar images will be more suitable for classification. This chapter has confirmed that Convolutional Neural Networks have proved to be promising models to learn representations of cars, especially at random orientations if compared with the SVM. This is evidenced by a performance of 90 % precision for 80% recall on aligned cars and 82% precision for 80% recall on rotated cars. We further improved on this by unsupervised pretraining, by an Average Precision increase of 1.3% on rotated patches. Finally, the run-times were compared for which again the CNN performed better by about 60 times on a CPU, the credit for this lies in the lightweight architecture of the CNN used with only 40,305 parameters. In the next chapter, we shall use this CNN classifier to perform object detection using a sliding-window on the wider radar scan.

Chapter 5

Object Detection in a RADAR scan

5.1 Motivation

In the project so far, we have trained a classifier for whether a radar patch is a car or not. The next aim is to be able to localise where in the entire radar sweep are the cars located. As discussed in chapter 1.3 there are three prominent approaches to detection in Vision. It was also discussed that Faster-RCNN showed promise in terms of speed (5fps) and accuracy in Vision (mean-average-precision = 0.70 in general). However, it involved applying a CNN on the entire image and a Region Proposer Network to obtain potential bounding-boxes (anchor-boxes) of various aspect-ratios. Due to the nature of our co-training approach of using a laser detector to generate labels, we do not have all the ground truth bounding boxes which makes it less suitable to use YOLO. We also discussed that sliding-window approach was simpler and suitable for the RADAR case as cars are of a fixed size enabling a fixed window size to be slid on the image. This chapter explores: how good is the speed and accuracy of a sliding-window approach for the RADAR modality?

5.1.1 Sliding window

In this approach, a window classification model is repeatedly applied at all locations in the image. We have chosen a radar image of size 561×561 which is about $140m \times 140m$ in area. If the step size is 1 pixel and the sliding-window size is 36×36 there are a total of $525 \times 525 = 275,625$ potential regions to classify per image. By increasing the stride to 10, there are $52 \times 52 = 2704$ regions which is 100 times less. In fact, the number of regions and hence time-performance of sliding window varies inversely by $(\text{stride})^2$.

In robotics applications, accuracy is not enough. We are aiming for a real-time object detection system and which would mean keeping up with a radar sensor that is 4Hz. As a corner case, let us consider an autonomous car driving at 50 mph on a highway. Then the relative velocity of the sensor with respect to an oncoming vehicle also at 50mph is 100mph which is 44 metres per second. Considering the sensor is in the centre of the 140 metres $\times$ 140 metres scan, that means the sensor can "see" objects in a radius of about 70 metres. This gives the oncoming vehicle $70/44 = 1.5$ seconds to travel from the edge of the radar image and hit the autonomous

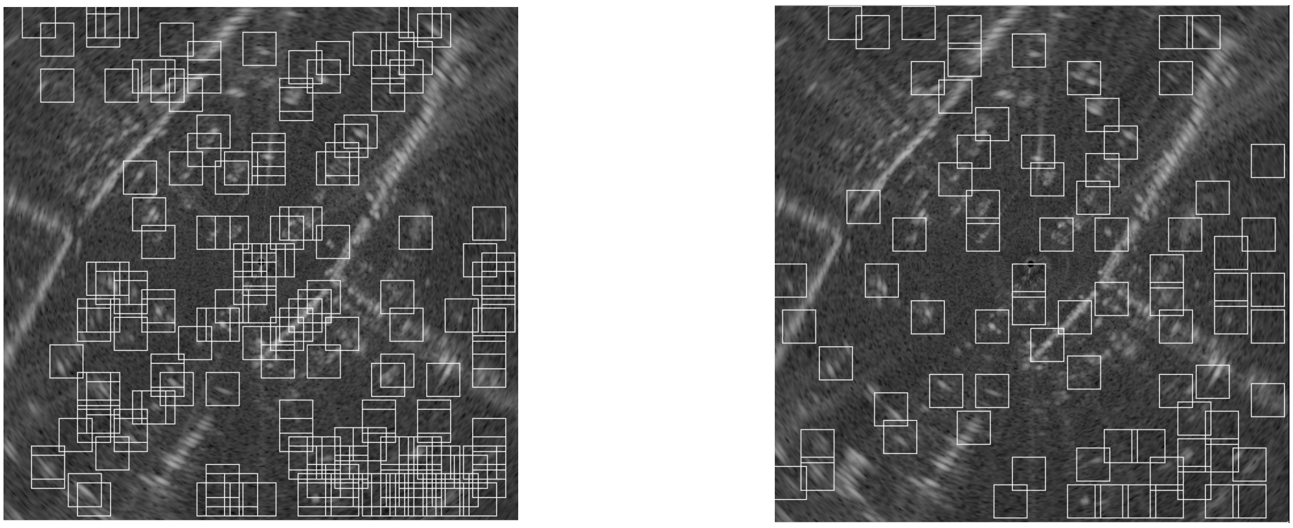


Figure 5.1: The pre-filtered overlapping detections illustrated in contrast to after NMS filtering.

car. Thus, we need to perform better than 1.5Hz to avoid this hypothetical corner case.

5.1.2 Non-Maximum Suppression

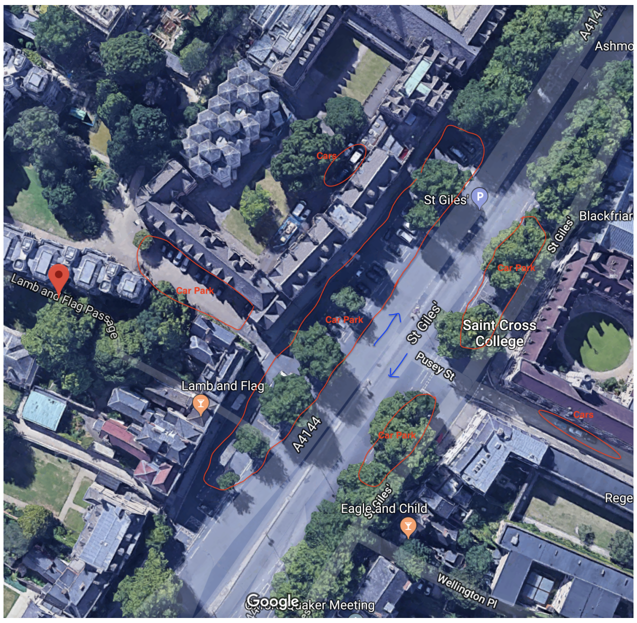
When several detections are located at a similar point in the broader image there is likely high overlap between the detection boxes. It is also very likely that these overlapping detections represent the same object but because the nature of sliding window with a small step size, we are capturing parts of the same object again. One solution to filter out these overlapping detections is that if boxes overlap by more than a certain threshold then we can keep only the box with the highest score among the overlapping boxes. We do this by sorting all the detections by score in descending order and iteratively finding the overlapping boxes for each of the indexes in this order. Our measure of overlap is the ratio of the overlapping areas and the total area of the overlapping boxes. The threshold chosen is 0.3 after trying some values empirically.

5.1.3 Hand Labelled Test Set

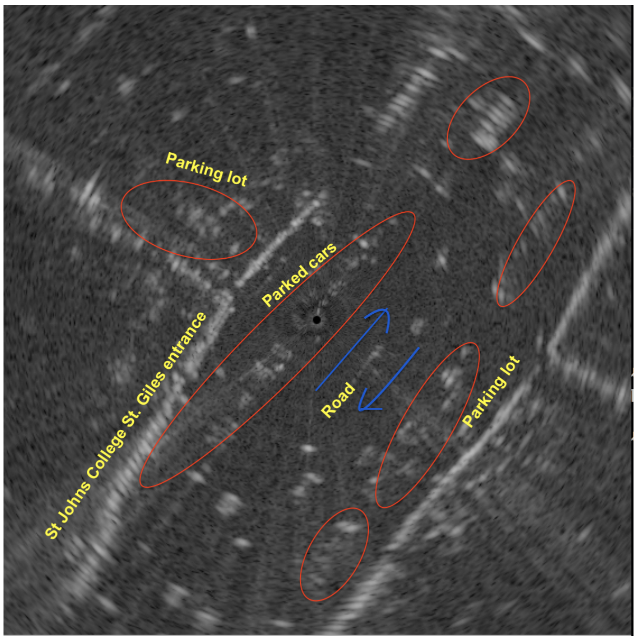
As noted earlier, the detections output from Vote3Deep are not perfect. To assess the performance of our detection scheme, we therefore hand label a portion of the lamb and flag dataset. Our radar object detector is evaluated against this hand labelled ground truth dataset of 90 frames from a span of 900 frames (leaving a gap of 10 between each frame for varied data). We treat these labels as a gold standard test set to evaluate our detection performance.

Hand labelling was done using ORIs software ActiveLabelling3DGUI in which 3D LiDAR data can be visualised and boxes can be drawn around cars in the scene. There were 230 labels in total. The limitation of LiDAR's range became very clear while doing this manual job. It is only clearly able to pick up an area of approximately 50m radius in practice. Cars beyond that appear in low resolution and it was hard to put bounding boxes. The same radar scene will contain cars from a much longer distance as well. We should therefore be prepared for an overestimate of false positives in the radar detector.

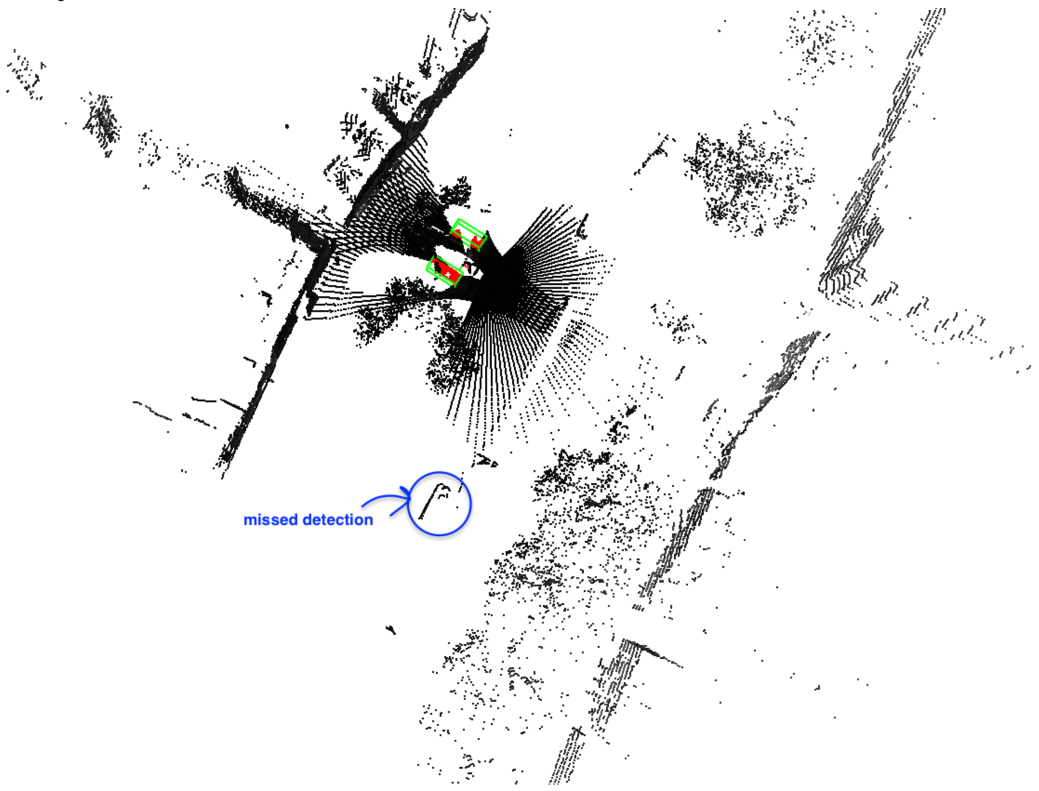
5.1.4 Visualising Detections Radar



(a) Satellite image of test scene with annotations of where to expect cars.



(b) Aerial view of correspondingly aligned radar scene.



(c) The corresponding scene in LiDar from where we get our detections. Noticeably able to capture a shorter range.

We wish to compare the radar detector with the hand labels. The white boxes in fig 5.3 are the detections made by our radar detector which could not be compared against any hand labels. This does not necessarily mean that they are false detections because as discussed the hand labels are not comprehensive due to the limited range of the Laser. The green boxes are those detections of our detector that we could match with the hand labels. The threshold for the radar detector was chosen to be 0.9 which is high to reduce the false

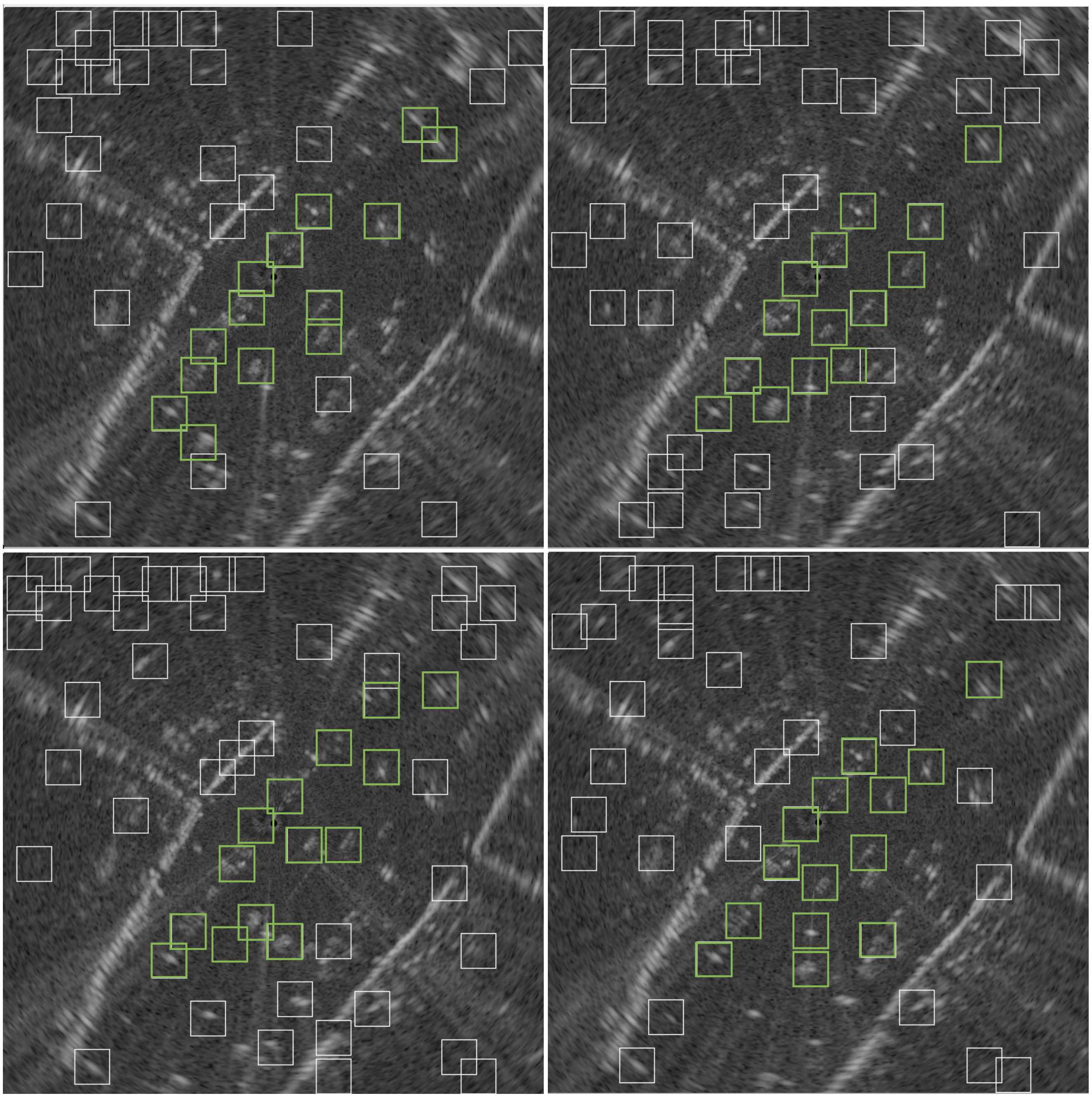


Figure 5.3: 4 consectutive Radar scans from the test set are displayed with the detections made at a threshold 0.9. Green boxes are those detections that could be corroborated with the manually labelled dataset.

positives.

In these 4 images that have been visualised, there have been 3 misses (per image). The average miss rate over the 90 frames at a threshold of 0.9 was 0.4. This is of course for a high threshold. If the threshold is lowered to 0.3 the miss rate sharply reduces to 0.02. The plot of average miss rate vs false positives per image is plotted in fig. 5.4. The number of detections that the radar could pick up were a lot more in general due to its range. From prior knowledge of the context we noticed that the detector made detections in the likely places such as a parking lot as shown in the satellite image 5.2a thus suggesting that our detector may be useful in longer range predictions as well.

5.2 Evaluating Detection

For every ground-truth label, the nearest neighbour label generated by our radar detector is first found. Then, we calculate the overlap (if any) between this nearest neighbour and the ground truth bounding-box label. The overlap measure used is intersection over union (IoU). If IoU is greater than a certain threshold (0.2 in our case), then we conclude that detection to be a true positive (TP). Otherwise, that detection is a false positive (FP) and we also increase the number of missed detections (FN) by one. The miss-rate for the whole image is then calculated as $MR = \frac{FN}{FN+TP}$. The miss rate and FP are calculated for each confidence threshold of the detector for an image. Then we take an average miss rate and FPPI over all the 90 frames in the test set and plot this as shown in 5.4 where lower the plotline, the better.

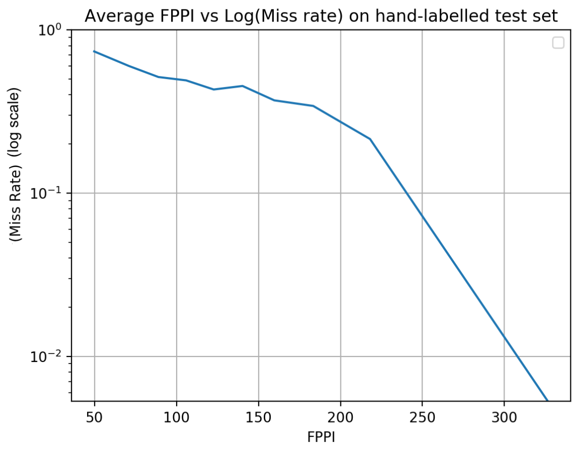


Figure 5.4: Miss Rate vs FPPI showing detector performance averaged over 90 frames.

From the fig 5.4 we can see that the miss-rate is quite low (0.02) after a threshold of 0.5 when there is a sharp drop in the curve. This is of course at the cost of a high number of false positives (220 FP at that threshold). While this is an overestimate of FP as discussed, we can still do better than this if we use tracking to filter out spurious detections as future work.

5.2.1 Analysis of detection visuals

The purpose of the above visualisation was to explore the effectiveness of our new radar detector in picking out cars both mobile and stationary at various angles. Radar could clearly detect cars correctly much beyond the purview of the LiDAR which is one key strength apparent from above. It could also pick up cars that were at varying angles as the parked cars which are in a row in the figure were parked perpendicular to the mobile cars on the road. We noticed some clear false positives for example the 3 detections (in each image) that were actually part of the wall of St.Cross college. This is not very surprising as that portion of the wall does resemble a car's representation, thus we may have to incorporate contextual information in a future approach as one solution.

5.2.2 Time performance

The pipeline of loading a radar image of size 561×561 pixels, running a sliding window (stride 10), applying a CNN classifier on each window followed by non-maximum suppression to the detections takes on average $2.05 \pm .18$ seconds over 90 frames. This performance was measured on Python code on a MacBookPro(Late 2016) equipped with 2.9 GHz Intel Core i5 CPU and 8 GB RAM. This is without the use of any multi-threaded parallelisation. To make this real-time or very close to it (about 2 to 4Hz) we need a 4 to 8 factor of improvement. In the two seconds there are 8 images supplied by the radar so we might be able to parallelise over two or three images simultaneously to obtain speed because processing images is *independent* of one another. Multi-threaded implementation of the for-loop that processes one frame per loop was tried on a CPU - parallelising each frame by spawning 4 separate threads. However, the overheads in creating these threads led to an exponential increase in time taken instead of reducing it. Thus, a GPU implementation is recommended as future work.

Note that the above calculation is done for quite a large image size (561×561) which is equivalent to a distance of 140 metres in each direction. The time taken reduces with a smaller image if we choose to, depending on the application. For instance a respectable image size of 261×261 took on average 0.82 seconds $\pm .13$.

5.3 Conclusion

This project was concerned with exploring the potential of machine learning and deep learning methods in being able to classify and detect objects (cars as a proof of concept) in the RADAR domain. The primary difficulty towards this goal was that there were not enough labelled datasets available to train such a model, especially because the task of identifying a car in RADAR is not trivial for humans to manually label.

To overcome this, a novel idea of projecting labels from an existing state-of-the-art 3D LiDAR detector on the RADAR frame on the same datasets was explored. After visualising and selecting a suitable threshold for the LiDAR detector, labels in LiDAR frame were generated. A calibration toolbox developed by ORI was utilised to project these labels and extract them as training patches - all aligned at the same orientation. This was repeated for five datasets from different locations in Oxford to increase variability and a total of 8,460 car detections were obtained.

Serving as a baseline for other approaches, a linear SVM approach to classification of cars showed some success with an Average Precision (AP) of 0.8 and at the operating point and a Precision of 0.8 for a Recall of 0.7 when trained and tested on aligned car patches. To understand how the SVM performs on the more realistic case when cars can be found at any orientation in the radar scene, the patches were randomly rotated. The SVM performed worse overall with and average precision of 0.76. Hard negative mining yielded a slight overall improvement.

The primary aim of this project was to investigate deep learning for radar image classification. We found that the CNN outperformed the SVM both in the aligned and rotated cases as confirmed by juxtaposing their PR curves. The improvement was stark especially in the rotated case. In the aligned case an average precision of 0.872 and at chosen operating point (Precision, Recall) = (0.8,0.9) was found. For the rotated case average precision of 0.86 and at chosen operating point (Precision, Recall) = (0.7,0.85) was found. After training an autoencoder in a self-supervised fashion, unsupervised pretraining was experimented to bolster the performance of this CNN. It was empirically found to have pushed up performance by 1.3 % to an AP of 0.89 without a single extra labelled instance.

Finally, we chose a sliding window approach for detection because of not having a complete set of ground truth bounding boxes and because cars in radar are of the same size everywhere in the scan. Using the CNN built above, the detection task was performed on a test set. To evaluate this, 90 radar scans were hand-labelled by projecting from LiDAR. Miss rate Vs. FPPI curve was plotted for the average over 90 frames. There was a miss rate of 0.2 and an FPPI of 220 at a chosen operating point with threshold 0.5. This FPPI rate however, is to be taken as an overestimate because of the longer range of the RADAR. It was verified using satellite image that there was a high likelihood of true detections in several places where a car was detected while missed/not reached by the Laser.

Further work may involve reducing the false positive rate in detection using another detection paradigm e.g. Faster-R-CNN. Tracking using a Kalman filter or a Recurrent Neural Network can also be done to filter out some spurious detections by considering the temporal dimension. A GPU implementation is also recommended for speed increases. Co-training i.e. projecting these RADAR detections to provide complementary views to aid in improving the LiDAR detector can also be another direction of work.

Bibliography

[1] Martin David. Adams. *Robotic navigation and mapping with radar / Martin Adams .... [et al.]* English. Artech House Boston ; London, 2012, xxix, 346 p. : ISBN: 9781608074822 160807482.

[2] Samuele Capobianco et al. "Vehicle classification based on convolutional networks applied to FM-CW radar signals". In: (2017), pp. 1–13. arXiv: 1710.05718. URL: http://arxiv.org/abs/1710.05718.

[3] *ClearWay CTS350 Specifications.* https://www.navtechradar.com/automatic-incident-detection/clearway-cts350-data/. Accessed: 2018-04-18.

[4] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: (2016), pp. 1–31. ISSN: 16113349. DOI: 10.1051/0004-6361/201527329. arXiv: 1603.07285. URL: http://arxiv.org/abs/1603.07285.

[5] M. Engelcke et al. "Vote3Deep: Fast Object Detection in 3D Point Clouds Using Efficient Convolutional Neural Networks". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).* June 2017.

[6] Dumitru Erhan, Aaron Courville, and Pascal Vincent. "Why Does Unsupervised Pre-training Help Deep Learning ?" In: 11 (2010), pp. 625–660.

[7] A. Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media, 2017. ISBN: 9781491962244. URL: https://books.google.co.uk/books?id=bRpYDgAAQBAJ.

[8] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2014), pp. 580–587. ISSN: 10636919. DOI: 10.1109/CVPR.2014.81. arXiv: 1311.2524.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* http://www.deeplearningbook.org. MIT Press, 2016.

[10] Max Jaderberg et al. "Spatial Transformer Networks". In: *CoRR* abs/1506.02025 (2015). arXiv: 1506.02025. URL: http://arxiv.org/abs/1506.02025.

[11] *Keras-upsampling-code.* https://github.com/keras-team/keras/blob/236757df/keras/layers/convolutional.py#L1552. Accessed: 2018-05-06.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances In Neural Information Processing Systems* (2012), pp. 1–9. ISSN: 10495258. DOI: http://dx.doi.org/10.1016/j.protcy.2014.09.007. arXiv: 1102.0183.

[13] Yann Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE.* 1998, pp. 2278–2324.

[14] Jakob Lombacher et al. "Potential of radar for static object classification using deep learning methods". In: *2016 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility (ICMIM)* (2016), pp. 1–4. DOI: 10.1109/ICMIM.2016.7533931. URL: http://ieeexplore.ieee.org/document/7533931/.

[15] *Loss functions.* https://keras.io/losses/. Accessed: 2018-05-06.

[16] *MOOS:Documentation Guide.* http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php/Main/Documentation. Accessed: 2018-04-18.

[17] Vinod Nair and Geoffrey E. Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML’10. Haifa, Israel: Omnipress, 2010, pp. 807–814. ISBN: 978-1-60558-907-7. URL: `http://dl.acm.org/citation.cfm?id=3104322.3104425`.

[18] *Object category detection practical*. `http://www.robots.ox.ac.uk/~vgg/practicals/category-detection/index.html`. Accessed: 2018-04-17.

[19] Augustus Odena, Vincent Dumoulin, and Chris Olah. “Deconvolution and Checkerboard Artifacts”. In: *Distill* (2016). DOI: `10.23915/distill.00003`. URL: `http://distill.pub/2016/deconv-checkerboard`.

[20] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[21] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: (2015). ISSN: 01689002. DOI: `10.1109/CVPR.2016.91`. arXiv: `1506.02640`. URL: `http://arxiv.org/abs/1506.02640`.

[22] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 91–99. URL: `http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf`.

[23] *Researchers blind autonomous cars by tricking LIDAR*. `https://www.theregister.co.uk/2017/06/27/lidar_spoofed_bad_news_for_self_driving_cars/`. Accessed: 2018-05-3.

[24] Raphael Rouveure, Patrice Faure, and Marie-Odile Monod. “PELICAN: Panoramic millimeter-wave radar for perception in mobile robotics applications, Part 1: Principles of FMCW radar and of 2D image construction”. In: 81 (Apr. 2016).

[25] Wenzhe Shi et al. “Is the deconvolution layer the same as a convolutional layer?” In: (2016). arXiv: `1609.07009`. URL: `http://arxiv.org/abs/1609.07009`.

[26] Paul Viola and Michael Jones. “Robust Real-time Object Detection”. In: *International Journal of Computer Vision*. 2001.

[27] Robert Weston. “Learning to interpret Radar via co-training (4yp)”. unpublished. 2017.

[28] Christian Wojek et al. “Pedestrian Detection : An Evaluation of the State of the Art”. In: (), pp. 1–20.

[29] Matthew D. Zeiler. “ADADELTA: An Adaptive Learning Rate Method”. In: *CoRR* abs/1212.5701 (2012). arXiv: `1212.5701`. URL: `http://arxiv.org/abs/1212.5701`.

Department of Engineering Science

UNIVERSITY OF OXFORD

Supplementary Questions for 4th Year Project Students

Risk Factor	Answer	Things to Consider	Record details here
Has the checklist covered all the problems that may arise from working with the VDU?	☐ Yes ☐ No		
Are you free from experiencing any fatigue, stress, discomfort or other symptoms which you attribute to working with the VDU or work environment?	☐ Yes ☐ No	Any aches, pains or sensory loss (tingling or pins and needles) in your neck, back shoulders or upper limbs. Do you experience restricted joint movement, impaired finger movements, grip or other disability, temporary or permanently	
Do you take adequate breaks when working at the VDU?	☐ Yes ☐ No	Periods of two minutes looking away from the screen taken every 20 minutes and longer periods every 2 hours Natural breaks for taking a drink and moving around the office answering the phone etc.	
How many hours per day do you spend working with this computer?	☐ 1-2 ☐ 3-4 ☐ 5-7 ☐ 8 or more		
How many days per week do you spend working with this computer?	☐ 1-2 ☐ 3-5 ☐ 6-7		
Please describe your computer usage pattern			