# GPU ACCELERATED OBSTACLE DETECTION WITH DEEP LEARNING FOR ADAS

A Project Report

Submitted by

**Gowdham Prabhakar P G**
**EDS14M006**

in partial fulfillment for the award of the degree of

**Master of Design**

in

**Electronic Systems**

**Indian Institute of Information Technology**
**Design and Manufacturing, Kancheepuram, India**

May 2016

I would like to dedicate this thesis to my beloved parents and my sweet sister. Their continuous support helped me lead this project to the successful completion.

BONAFIDE CERTIFICATE

This is to certify that the thesis titled "**GPU ACCELERATED OBSTACLE DETECTION WITH DEEP LEARNING FOR ADAS**" submitted by **Mr. P G Gowdham Prabhakar (EDS14M006)** to the Indian Institute of Information Technology Design and Manufacturing, Kancheepuram, for the award of **Master of Design in Electronic Systems,** is a *bona fide* record of the project work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

**Dr.Binsu J Kailath**
Project Guide

Asst. Professor
Indian Institute of Information Technology
Design and Manufacturing, Kancheepuram
Chennai 600 127
India

Place: Chennai
Date:

## ACKNOWLEDGEMENTS

Firstly, I thank the Director of the Institute Prof. R. Gnanamoorthy for providing all infrastructural facilities required for my project as well as graduate study here. I wish to express my sincere gratitude to my Project Guide Dr. Binsu J. Kailath, Assistant Professor, IIITDM Kancheepuram, for the continuous support of my project work, for her patience, motivation and immense knowledge.

I also wish to express my sincere gratitude to my Co-guide Dr. Sudha Natarajan, Autonomous Vehicle Program - R&D, Tata Elxsi Limited, Chennai, for her support and guidance for my project.

I would like to thank research scholars Mr G Dinesh. and Mr K.K Abdul Majeed for their valuable suggestions regarding the usage of software tools. I also thank the technical staff in the Electronics and Computer Science labs for their timely help. My sincere thanks also go to the stream coordinator Dr. P. Damodharan and all the members of faculty in Electronics Engineering for their valuable suggestions and comments during the project reviews. Last but not the least; I thank all my friends for making my stay here a memorable one.

## ABSTRACT

Obstacle detection and classification is one of the key tasks in the perception system of ADAS and self-driving vehicles. Vision-based approaches are popular due to cost-effectiveness and appearance information associated with the vision data. In this paper, a deep learning system using region-based convolutional neural network is proposed for the detection and classification of on-road obstacles such as vehicles, pedestrians and animals. The proposed system is quite suitable for high-speed driving as it exploits the massive parallelism of neural network and GPU implementation. Results on images from road traffic benchmark datasets and Indian road images demonstrate the performance of the system invariant to object's shape and view, and different lighting and climatic conditions.

In the future, this algorithm is planned to be implemented in a car for real time object detection and recognition.

# Contents

# List of Figures

# List of Tables

# Nomenclature

GPU – Graphics Processing Unit

ADAS – Advanced Driver Assistance System

LIDAR – Light Detection and Ranging

CNN – Convolutional Neural network

RCNN – Region based Convolutional neural Network

ZF – Zeiler and Fergus

VOC – Visual Object Challenge

SVM – Support Vector Machine

FC – Fully Connected

RPN – Region Proposal Network

IoU – Intersection over Union

AP – Average Precision

mAP – mean Average Precision

# Chapter 1 Introduction

## 1.1 Motivation

In the present era, the gaining significance of autonomous robots has a key role in on road driving assistance for vehicles such as ADAS (Advanced Driver Assistance Systems). The ADAS techniques usually involve fusion of image processing and LIDAR (Light Detection and Ranging) sensors. These techniques work well for obstacle avoidance and intelligent navigation systems. Usage of LIDARs increases the cost of vehicles and moreover it does not contribute much to the intelligence. Object detection can be done using LIDAR but object recognition is possible only using image processing. By recognizing an object in real world, the characteristics of the object can be learnt and the response of the car for the detected object could be trained in a specific manner rather than generic methods.

## 1.2 Problem Statement

Object detection and recognition is to be implemented in GPU using Deep Learning methods for better accuracy and speed of detection. The system is to be tested on Indian road dataset.

## 1.3 Overview of the Project

Collision avoidance system is a key component in self-driving vehicles and vision-based such system is a cost-effective solution. Existing techniques for vision-based on-road obstacle detection techniques [1], [2] have not progressed into its mature form due to many issues such as variability in vehicle shapes, cluttered environment and illumination conditions. Deep learning [3] has shown great promise in recent years in the field of object detection and recognition. Convolutional Neural Networks (CNN) are dedicated to vision-based approaches and they are feasible for GPU acceleration in real-time applications.

## 1.4 Contributions of the Thesis

In this project, the detection and classification of on-road objects using Faster Region-based CNN (RCNN), a variant of CNN and its implementation in GPU is discussed. A pre-trained network model ZFNet, fine-tuned for 20 different objects of PASCAL VOC 2007 dataset [4], was employed in our proposed detection and classification system. During the real-time detection phase, the on-road object detections are filtered. The outputs of the system are the rectangular bounding boxes and class information of objects which are useful parameters for motion planning of the self-driving vehicle.

## 1.5 Organization of the Report

This report is organized as follows. The following chapter briefly describes the conventional CNN and its variant RCNN. Chapter 3 explains the proposed on-road obstacle detection and classification systems. The GPU implementation of the system using CAFFE framework along with the results and performance analyses are given in Chapter 4. Chapter 5 concludes the report.

# Chapter 2 Background

## 2.1 Convolutional Neural Network (CNN)

In the living era of Autonomous vehicles, there are several techniques to detect and classify the on-road obstacles. The state-of-the-art techniques for object detection and classification use CNN as the basis. CNNs are made up of neurons which have weights and biases. These neurons receive inputs from an image, perform a dot product with the weights and generate a scoring output. The entire network expresses a single score function which is always differentiable. These functions take the raw image pixels as input and output the class scores. The weights are updated by means of backpropagation with reference to the loss function. The loss function like Support Vector Machines (SVM)/Softmax is generally used in CNNs on the last Fully Connected (FC) layer. To build a ConvNet we use three layers namely convolutional layer, pooling layer and fully-connected layer. Each layer can input a 3D data and transform it to a 3D data output.

### 2.1.1 Region-based CNN (RCNN)

CNNs are able to give better mean Average Precision (mAP) in object detection and classification, but take a lot of time in training. In order to optimize the training time as well as detection time, a modified version of CNN called RCNN [5] was proposed which gave 58.5% mAP on PASCAL VOC 2007. It consists of three modules. The first module generates a set of category-independent region proposals. The second module is a large CNN that extracts a fixed-length feature vector from respective region proposals. The third module is a set of class-specific linear SVMs. The working of RCNN is illustrated in Figure. 2.1.

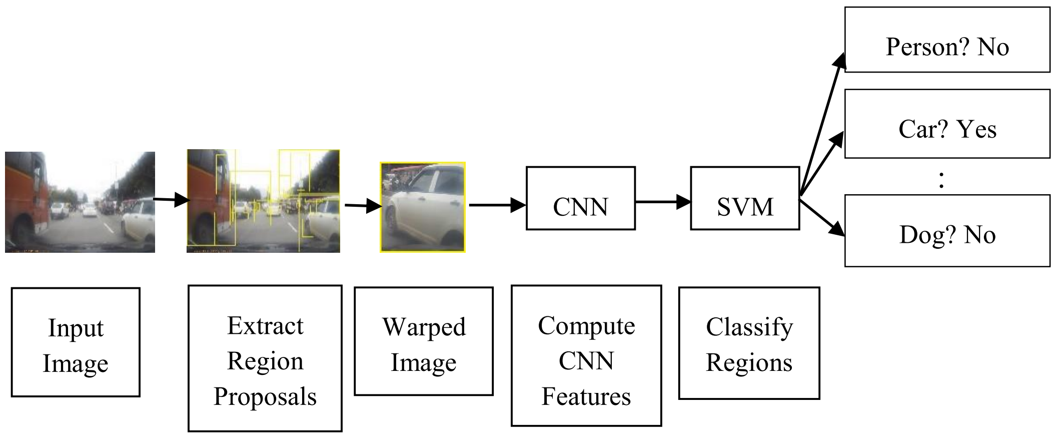

Figure 2.1: RCNN working model.

### 2.1.2 Faster RCNN

Faster RCNN [6] is a new method to realize RCNN for better performance on mAP as well as execution time. In this method, region proposals are made by a separate convolutional network called Region Proposal Network (RPN). This network shares the convolutional features with the Detection Network (RCNN). RPNs are trained to generate better region proposals, which are used for detection by the detection network. RPN and RCNN are together trained which share a set of convolutional features. The sliding window size is chosen as 3x3 and is mapped to a lower-dimensional vector (256-d). This vector is sent to teo layers called box-regression layer (*reg*) and box-classification layer (*cls*). The *k* region proposals are predicted such that the reg layer has 4*k* outputs (coordinates of *k* boxes) and the *cls* layer has 2*k* scores (object or not). These proposals are relative to *k* reference boxes, called anchors. The working of RPN is described in the Figure. 2.2. Faster RCNN achieved 73.2% mAP on PASCAL VOC 2007 using 300 proposals per image.

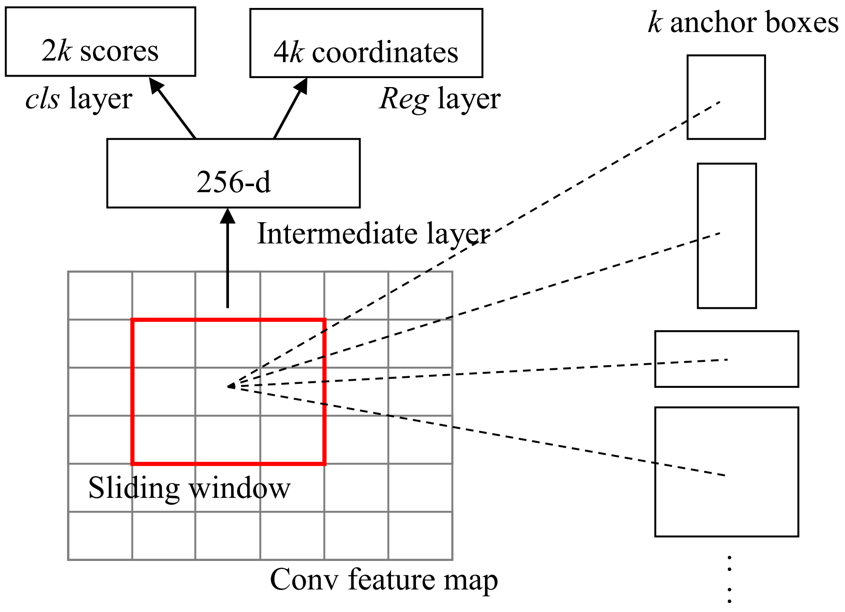

Figure 2.2: Region Proposal Network (RPN).

# Chapter 3 Proposed Design for Object Detection and Classification

## 3.1 Obstacle Detection and Classification

The proposed system employs the Faster RCNN implemented on GPU for detection and classification of on-road objects. The RCNN was trained as follows. An ImageNet pre-trained model ZF was used to initialize the weights. The PASCAL VOC 2007 dataset was used to fine-tune the network for detecting only 20 object classes. It includes aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, dining table, dog, horse, motorbike, person, potted plant, sheep, sofa, train, TV-monitor. Since the training set contains a lot of non-road objects, the network requires a retraining with only the on-road object classes. We have not however retrained the network as this requires a powerful GPU with a large graphics memory. Instead, the non-road object detections were masked in the detection phase efficiently.

There are three proposed versions of the system. The first version reads a set of images and processes for object classification. The second version reads a video file and processes the frames for object classification. The third version reads the live stream of a webcam and processes the frames for objects. These three versions are implemented in python with the help of OpenCV.

For real-time detection on video, each image frame is fed to the system. This image frame is processed by the trained RCNN module for the bounding boxes of various class-specific objects. Since this system is designed for detecting only the on-road objects, some classes are masked in such a way that only the on-road objects are annotated. This is done by a filter. The processed image is then annotated with bounding boxes tagged with the respective class name on top of each detected object. The bounding box colours are listed in Table 1.1. The entire workflow is illustrated by the block diagram in Figure. 3.1.

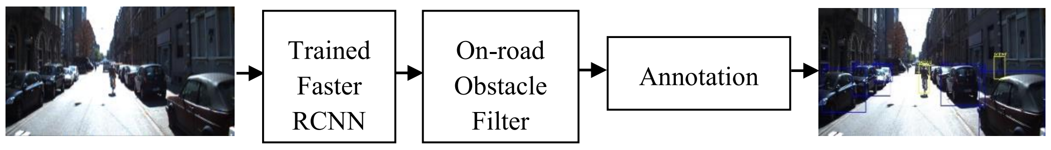

Figure 3.1: Block Diagram of the proposed system.

Each frame is processed for several region proposals by RPN. These region proposals are validated as several objects by CNN. Since this entire algorithm is focussed on object detection and recognition, the accuracy of detection is more significant than the speed of detection. Hence the following chapter discusses the detection accuracy calculations and comparisons among different datasets including Indian road datasets. Some datasets like Kitti, provide information about the bounding boxes of the ground truths in a file called annotation file. Some of the annotation files are not perfect and at times goes out of scope for our desired objects. So the accuracy is also calculated using heuristics where the detected objects are observed by human eyes and validated for correctness.

Table 3.1: List of on-road obstacles and their respective colours of the bounding-boxes.

| **Class Name** | **Colour of the Bounding-Box** |
|---|---|
| Bicycle | Red |
| Bus | Blue |
| Car | Blue |
| Cat | Cyan |
| Cow | Cyan |
| Dog | Cyan |
| Motorbike | Red |
| Person | Yellow |

## 3.2 Parameters for Module Integration

The image frames in the video version is processed for object proposals and the corresponding bounding boxes are stored in a variable called “dets”. These bounding boxes are parsed and stored in the form of  coordinates of top left corner (x1,y1) and bottom right corner (x2,y2) of the object along with the object class name, in a text file called “box.txt”. Refer Appendix A.5.

Since, all the other modules in the main project (Autonomous Vehicle) is implemented in C++ environment, a C++ wrapper code is implemented which invokes the python code for integrating this Object detection and recognition algorithm along with other modules.

In the future, this module is to be integrated along with other modules using sensor data fusion for optimised and better driving performance.

# Chapter 4    GPU Implementation and Performance Evaluation

## 4.1    System Configuration

The proposed system is implemented on Ubuntu workstation with NVIDIA Quadro K620 GPU. The GPU has 2GB graphics memory and the workstation is powered by Intel Xeon E3-1200 with 4GB RAM. There are two modules in the proposed system [7]. The main module runs on CPU. The second module that includes the CAFFE framework of RCNN runs on GPU. A wrapper code is developed for on-road obstacle filter.

## 4.2    Testing Results on Datasets

The implementation was tested on a variety of datasets in different climatic conditions. Images were from the public datasets such as Kitti [8] (size: 1392x512) and IRoads [9] (size: 640x360), Bangalore road (size: 1920x1080), Chennai road (size: 1920x1080), On-Road animals (size: 1025x680, 1001x608). Figures 4.1-4.6 show the performance of ZFNet model of faster RCNN on GPU. The objects in Figure 4.1-4.2 were detected in 0.271s for 300 object proposals. The objects in Figure 4.3 were detected in 0.289s for 169 object proposals. The objects in Figures 4.4-4.7 were detected in approximately 0.3s for 300 object proposals.

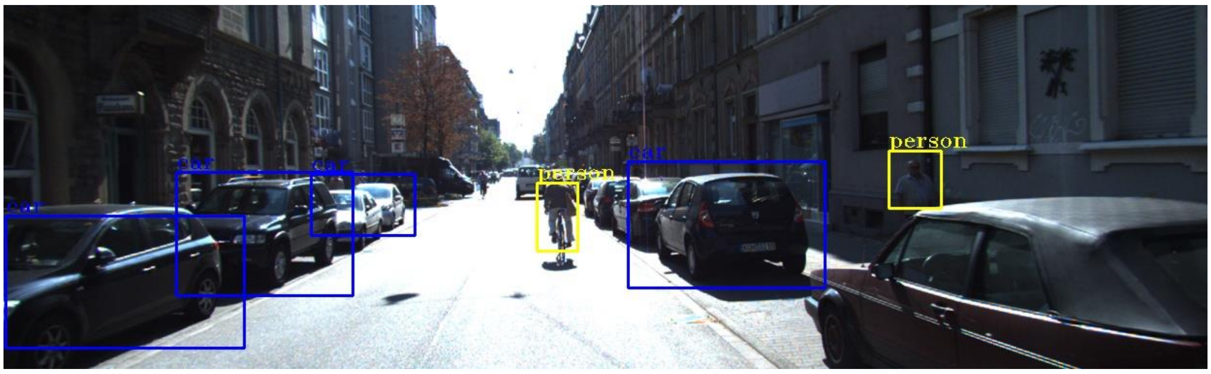

Figure 4.1: Person and cars are detected on KITTI_drive0005 image.

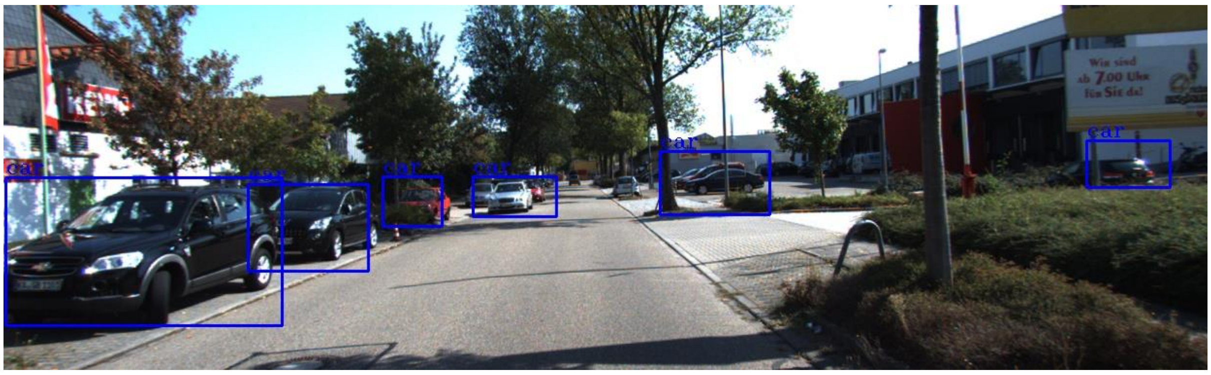

Figure 4.2: Cars are detected on KITTI_drive0009 image.

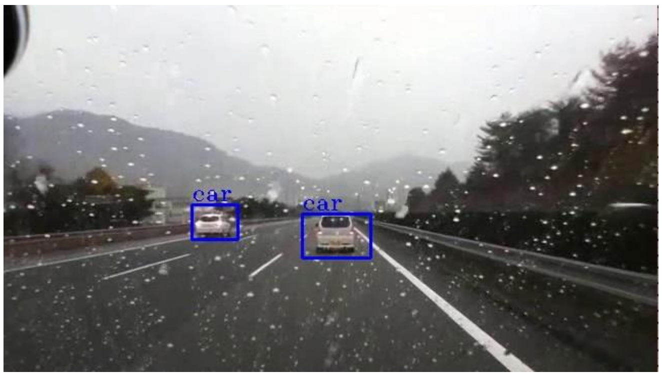

Figure 4.3: Cars detected during a rainy day on IRoads image.

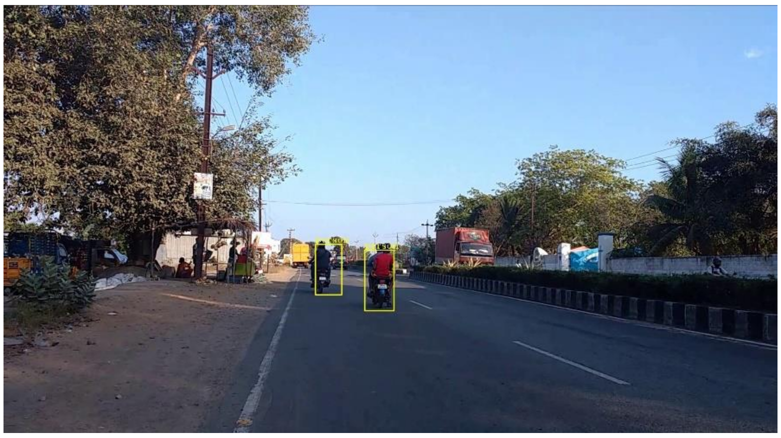

Figure 4.4: Persons detected on Chennai Highways.

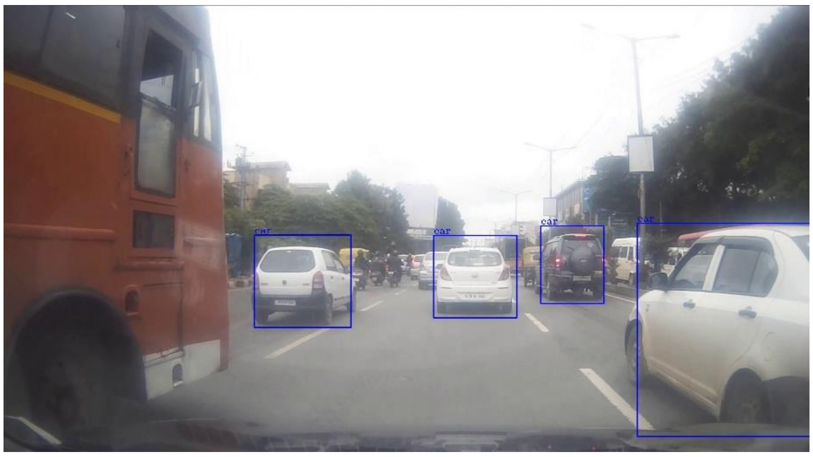

Figure 4.5: Cars detected on Bangalore Highways.

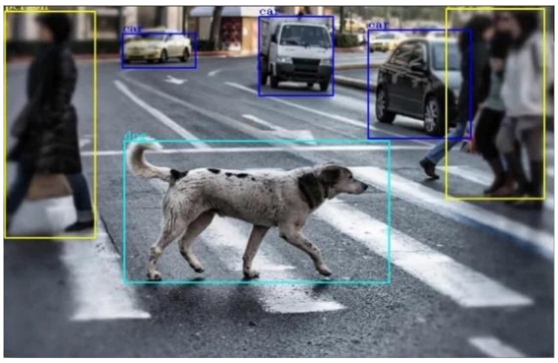

Figure 4.6: Dog detected along with persons and cars.

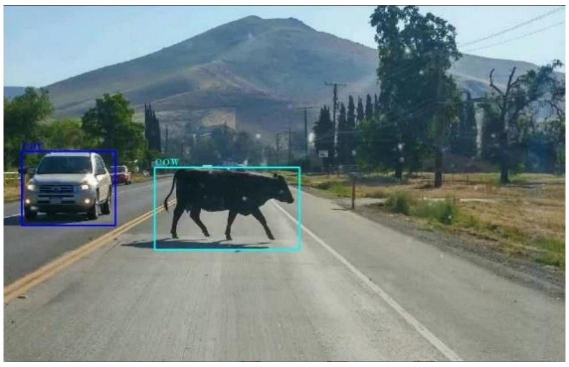

Figure 4.7: Cow detected along with car.

## 4.3 Performance Evaluation

The performance of the system is mainly focussed on the detection accuracy rather than the speed of detection/execution. The commonly used metric to measure the detection accuracy is the mean Average Precision (mAP). This is calculated by taking the average precision of each class at first and then averaging all the Average Precisions of every class as explained in the eqn. (4.1), eqn. (4.2) and eqn. (4.3).

$$Precision\ (P) = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{4.1}$$

$$Average\ Precision\ (AP) = \frac{\sum P\ \forall\ True\ Positives}{True\ Positives} \tag{4.2}$$

$$mean\ Average\ Precision\ (mAP) = \frac{\sum AP\ \forall\ classes}{Number\ of\ classes} \tag{4.3}$$

$$Intersection\ over\ Union\ (IoU) = \frac{Ground\ Truth\ BoundingBox \cap Detected\ BoundingBox}{Ground\ Truth\ BoundingBox \cup Detected\ BoundingBox} \tag{4.4}$$

The validation of True Positives in each class is done only if the Intersection over Union (IoU) >=0.5. The calculation of IoU is done as explained in the eqn (4.4). Except the Kitti Dataset, none of the test dataset is provided with the Ground Truth Bounding Boxes data in an annotation (.xml) file. The Chennai Road and Bangalore Road datasets are taken manually with the help of camera. This data is used to compare with the bounding boxes of the detection system for IoU. Since the bounding boxes are sometimes erroneous in the annotation file and the proposed classes are different from the detection system proposals e.g., vans in ground truth are considered as cars and cyclists in ground truth are considered as persons, the mAP is also calculated by means of heuristic methods where the images are individually investigated by eyes. Some classes in the detection algorithm sometimes are missing in the ground truth. This adds loss to the detection accuracy.

Table 4.1 shows mAP calculation of Kitti_drive0005 through Dataset Ground Truth and Table 4.2 shows the mAP calculation of Kitti_drive0005 through heuristics. The mAP was calculated over 153 images of Kitti_drive0005 in both cases. Table 4.3 shows the mAP calculation of Kitti_drive0009 through Dataset Ground Truth. The mAP was calculated over 426 images of Kitti_drive0009. Table 4.4 shows mAP calculation of Chennai Road dataset of 50 images through heuristics. Table 4.5 shows mAP calculation of Bangalore Road Dataset of 100 images through heuristics.

Table 4.1: mAP calculation for Kitti_drive0005 through Dataset Ground Truth.

| **Class Name** | **Average Precision (AP)** | **mAP (%)** |
|---|---|---|
| Car | 0.8498 | 62.14 |
| Person | 0.3930 | |

Table 4.2: mAP calculation for Kitti_drive0005 through Heuristics.

| **Class Name** | **Average Precision (AP)** | **mAP (%)** |
|---|---|---|
| Bicycle | 1 | 71.7 |
| Car | 0.916 | |
| Motorbike | 0 | |
| Person | 0.952 | |

Table 4.3: mAP calculation for Kitti_drive0009 through Dataset Ground Truth.

| **Class Name** | **Average Precision (AP)** | **mAP (%)** |
|---|---|---|
| Car | 0.8361 | 65.82 |
| Person | 0.4803 | |

Table 4.4: mAP calculation for Chennai Road through Heuristics.

| Class Name | Average Precision (AP) | mAP (%) |
|---|---|---|
| Bus | 0.62 | 90.5 |
| Car | 1 | |
| Motorbike | 1 | |
| Person | 1 | |

Table 4.5: mAP calculation for Bangalore Road through Heuristics.

| Class Name | Average Precision (AP) | mAP (%) |
|---|---|---|
| Bus | 0.9805 | 97.42 |
| Car | 0.9209 | |
| Motorbike | 1 | |
| Person | 0.9805 | |

# Chapter 5 Conclusion and Future Work

A vision-based object detection system for on-road obstacles was realized using Faster RCNN and implemented on GPU. The performance was evaluated on images of benchmark datasets and Indian roads. The deep learning network is found to be robust to variation in object's view, lighting and climatic conditions. The resulting bounding boxes of detected objects and their classes are useful for subsequent motion planning and control subsystems of self-driving vehicle. Although the proposed system detects almost all on-road obstacles, it sometimes fails to detect vehicles like auto that are commonly seen on Indian roads but not in the PASCAL training dataset.

Our future work will focus on improving the detection performance for Indian road scenario. We plan to create a dataset for Indian road vehicles and retrain the network with this dataset. The performance can be further improved by using a much deeper and wider network models such as VGG16 and GoogleNet. This necessitates a more powerful GPU and graphics memory for training and detection.

# References

[1] A. Mukhtar, L. Xia and T.B. Tang, “Vehicle detection techniques for collision avoidance systems: A review”, IEEE Transactions on Intelligent Transportation Systems, Vol. 16, No. 5, Oct. 2015.

[2] S. Sivaraman and M.M. Trivedi, “Looking at vehicles on the road: A survey of vision-based vehicle detection, tracking, and behavior analysis”, IEEE Transactions on Intelligent Transportation Systems, Vol. 14, No. 4, Dec. 2013.

[3] I. Arel, D.C. Rose and T.P. Karnowski, “Deep machine learning - A new frontier in artificial intelligence research”, IEEE Computational Intelligence Magazine, Nov. 2010.

[4] PASCAL VOC 2007 Dataset. [Online]. Avilable: http://host.robots.ox.ac.uk/pascal/VOC/voc2007/index.html

[5] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Region-Based Convolutional Networks for Accurate Object Detection and Segmentation," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 38, no. 1, pp. 142-158, Jan. 1 2016. doi: 10.1109/TPAMI.2015.2437384

[6] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sunar. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks arXiv: 1506.01497v2 [cs.CV] 13 Sep 2015

[7] Faster RCNN Python Code [Online]. Available: https://github.com/rbgirshick/py-faster-rcnn

[8] Kitti Dataset [Online]. Available: http://www.cvlibs.net/datasets/kitti/raw_data.php

[9] IRoads Dataset [Online]. Available: https://www.cs.auckland.ac.nz/~m.rezaei/Publications/iROADS%20Dataset.pdf

# Appendix A Demo setup

This guide will walk you through the installation and running the python implementation of Faster RCNN algorithm on a CUDA supported GPU system for a webcam, video as well as image feed.

Note: Please use the folder py-faster-rcnn for compiling and running the demo.

## A.1 Hardware Requirements

Workstation Specifications: Intel core i5, i7, Xeon processor, 4GB and above RAM, 250GB HDD, 2GB and above CUDA graphics card.

## A.2 Software Requirements

OS: Ubuntu 14.04

Software Packages: Python 2.7, OpenCV 2.4.9, G++

Python packages: cython, easydict

Build cython modules:

```
cd py-faster-rcnn/lib
make
```

Build Caffe and pycaffe:

```
cd py-faster-rcnn/caffe-fast-rcnn
# Now follow the Caffe installation instructions here:
#   http://caffe.berkeleyvision.org/install_apt.html
make -j8 && make pycaffe
```

## A.3 Demo for Webcam Feed

```
cd py-faster-rcnn
./tools/demowebcam.py --net ZF
```

### A.3.1 Running inside c++ wrapper

```
cd py-faster-rcnn
g++ demowebcam.cpp
./a.out
```

## A.4 Demo for Webcam Feed

Note: The link of the video file should be typed in line no.163

cap = cv2.VideoCapture("data/demo/vid1.mp4") of the demovideo.py  file inside tools folder.

```
cd py-faster-rcnn
./tools/demovideo.py --net ZF
```

### A.4.1 Running inside c++ wrapper

```
cd py-faster-rcnn
g++ demovideo.cpp
./a.out
```

## A.5 Demo for Image Feed

Note:

- The images should be located in /data/demo/. The name of the image file/files should be typed in line no.167 im_names = ['0000000000.png'] of the demoimagebox.py file inside tools folder.
- Uncomment line no. 138 for displaying the processed image with objects plotted.

### A.5.1 Running the demo in python

```
cd py-faster-rcnn
./tools/demoimagebox.py --net ZF
```

### A.5.2 Running inside c++ wrapper

```
cd py-faster-rcnn
g++ demoimagebox.cpp
./a.out
```

### A.5.3 Output text file (box.txt) of Demo for Image Feed

- The output file “box.txt” stored inside py-faster-rcnn contains the information about the bounding box coordinates and the class names for each image processed.
- Column 1 contains the image number, column 2 contains the class name, column 3,4,5,6 contains x1,y1,x4,y4 respectively where (x1,y1) is top left corner coordinate and (x4,y4) is bottom right corner coordinate of the bounding box.

```
0 car 373 195 459 243
0 car 738 198 1189 352
0 car 1049 123 1199 190
0 car 679 170 880 278
1 car 1104 127 1217 182
1 car 728 191 1198 352
1 car 694 178 895 288
1 car 368 181 444 255
2 car 713 179 897 302
2 car 341 184 436 248
2 car 738 197 1241 356
.
.
.
426 car 0 240 95 374
```

## A.6 Additional I/O information

The input of this system are images. There are two outputs in this system. The bounding box file ‘box.txt’ and display of processed image for visualising the bounding boxes on the image. The latter can be optional by commenting the corresponding line to display the image. The coordinates of the bounding boxes for each detected object with class name is parsed from the variable ‘dets’ in the RCNN program.