Department of Computer Science and Engineering

Chittagong Independent University

Report for Senior Project (CSE 498)

Real-Time Bangla License Plate Recognition using TensorFlow Object Detection API

Submitted By

Tariqul Islam

ID: 16302007

Supervised By

Risul Islam Rasel

Assistant Professor, Dept. of CSE

Chittagong Independent University.

Table of Content

Content Page

List of Figures

Figure No Page

List of Tables

Table No Page

Candidate's Declaration

This is to certify that the work presented in this paper, titled "Real-Time Bangla

License Plate Recognition using TensorFlow Object Detection API", is the outcome of the investigation and research carried out under the supervision of **Mr. Risul Islam, Assistant Professor, Department of Computer Science and Engineering, Chittagong Independent University.**

It is also declared that neither this work nor any part of it has been submitted anywhere else for the award of any degree, diploma or other qualifications.

Real-Time Bangla License Plate Recognition using TensorFlow Object Detection API.

By

Tariqul Islam (ID: 16302007)

PROJECT SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENT
FOR THE DEGREE OF
BACHELOR OF COMPUTER SCIENCE AND ENGINEERING

Approved By:

Mr. Risul Islam

Assistant Professor

Department of Computer Science & Engineering

Chittagong Independent University

Dedication

This project is dedicated to my departed mother.

Abstract

Traffic control and vehicle owner identification have become a major problem in Bangladesh. Most of the time it is difficult to identify the owner of the vehicle who has violated the traffic rule. Also, it might turn out to be very time consuming for an officer to physically check the license plate of every car. There must exist a way of detecting and identifying license plates without constant human intervention. Therefore, this project is developed as a solution to the problem. The existing license plate detection systems are mostly based on character segmentation. They heavily rely on several image filtrations followed by character segmentation. Moreover, these methods were not implemented in real-time. The proposed method in this project for detecting characters on a license plate is based on Tensorflow Object Detection API, where two separate Deep Convolutional Neural Networks are used to identify the license plate and the characters on the license plate. One main feature of Object Detection is that it detects the object on an image, and also finds the position of that particular object on the image by drawing a bounding box around the object. The characters can be detected as individual objects and then rearrange the characters and numbers according to their positions to identify the license plate. After detecting the license plate, the portion containing the license plate can be cropped from the frame. Then the cropped image can be again fed to another CNN for the detection of the characters. After arranging the detected characters in right order, we will reach our objective of detecting the license plate. Owner identification would become much easier by implementing this in real world. Also, the application can be useful for automatic toll collection and issuing fine tickets. The application developed using proposed methodology gains a 100% precision on detecting the license plate, and 91.67% precision for detecting the characters on the license plate, for given test data (18 videos and 6 still images).

Chapter 1

INTRODUCTION

1.1 Real time Bangla License Plate Recognition

License plate recognition or Automatic License Plate Recognition (ALPR) is the technology that is used to read the license plate characters. Bangla license plate recognition or automatic Bangla license plate recognition can be implemented for recognizing the license plate issued by Bangladesh Road Transport Authority (BRTA), for still images or video streams. The detection of the license plate can be done in real time using the same mechanism used for detecting from still images. We can take a video from the camera and feed the frames of the video to a convolutional neural network one after another. After successfully detecting the characters, we can then store the record of the vehicle in a database.

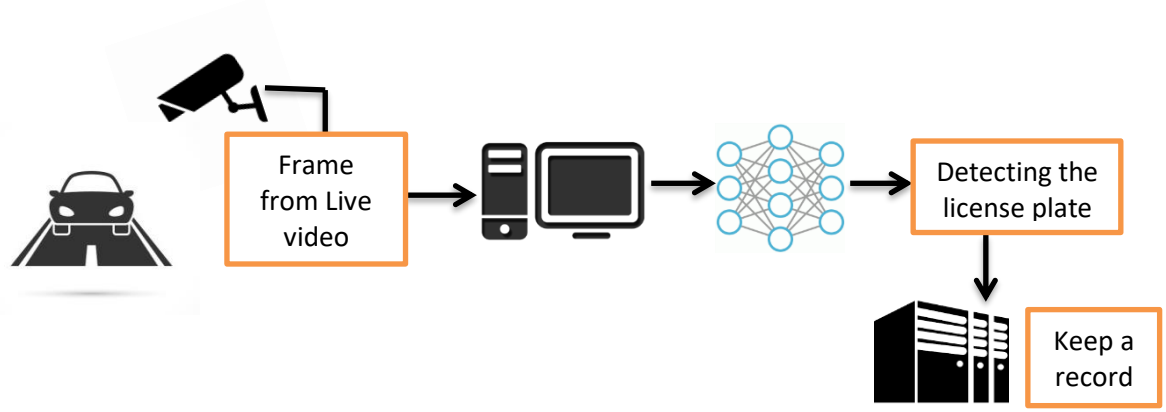

Figure 1.1: Outer view of the whole system

Figure 1.1 shows an outer glimpse of the whole system and how it will work.

In this work, the characters and numbers of a license plate are considered as individual objects and to detect the characters, Tensorflow Object detection API was used.

For the detection purpose, two several classification models are trained. The first model is used for detecting the license plate and the second model is used for detecting the characters and numbers on the plate.

The overall license plate recognition is done in two steps:

1.1.1 Detecting the license plate:

- Get image frame from live camera
- Detecting the license plate
- Crop the detected portion from the image frame

1.1.2 Recognizing the characters on the license plate:

- Take the image of the license plate
- Detecting the characters
- Arranging the characters and numbers in right order
- Keep a record of the detected license plate

1.2 Applications of Real-Time Bangla License Plate Recognition

The most important application of ALPR system is traffic monitoring. It can be very helpful for stolen vehicle identification, over speeding vehicles and traffic lawbreaker. This system can help traffic police to efficiently detect the license plate of vehicles and automatically keep a database of those vehicles. It also can be useful for private vehicle surveillance by distinguishing if a vehicle is authorized to enter a private premise or restricted area where a certain number of vehicles are allowed only. Another great approach is automatic toll collection for highways, flyovers and bridges. This automatic approach can save time and reduce traffic congestion during toll collection. Even in gas stations and shopping mall parking it can come handy.

1.3 Motivation

Automatic license plate detection system is becoming very popular lately for its practicality in real life. It can be implemented for a variety of applications. Many algorithms and techniques have been implemented for optical character recognition. But all the previous implementations had some flaws and weaknesses. The previous implementations of this type of application had variety of technical issues and had some limitations in preprocessing step. Most of the implementations have not considered the distorted, noisy, tilted, low contrasted images. Moreover those implementations were subjected to still images, and relied on hand coded threshold for the preprocessing filters. To have more accurate result in license plate recognition, those problems and limitations should be solved. Hence particular effort is needed to develop and application to make the most accurate character recognition application that can detect license plate in real-time.

1.4 Objectives

The objectives of this project are:

1. Detecting a vehicle license plate for different angles of the camera.
2. Detecting the characters and numbers on the license plate.
3. Storing the license plate number in the local machine with the date and time of the detection.

1.5 Scope of the Project

- License plates can be of different sizes and shapes. To detect characters from every shape of license plate is beyond the scope of this project.
- The project only focuses on the general license plates which are issued by BTRA, which are white in color and written in black characters using standard Bangla font.

- The application will be able to detect multiple license plates from a single frame, but for the simplicity of the project, multiple license plate detection is not considered in this work.
- Minor damaged plates can be detected but it won't recognize if the plate is very rusty and blurry.
- Detecting license plate of high speed vehicles (over 20kmph) was not tested for this system.
- To detect the license plate in night time, the lighting must be sufficient. The testing of the application was done in evening time with moderate lighting.

1.6 Organization of the Report

The rest of the document is organized as follows: In the Chapter 2, Literature Reviews of several existing license plate character recognition techniques are discussed. In Chapter 3, the Methodology is described. In Chapter 4, the Result and Performance of the application is described. Chapter 5 is about the Summary and the Future works which can be done to increase the performance of this system.

Chapter 2

LITERATURE REVIEW

2.1 Introduction

Usually, license plate recognition process of an ALPR system is comprised of three stages: segmentation, feature extraction and recognition. Due to the various applications of license plate recognition, various techniques have been developed for these stages. In this chapter an overview of these techniques that are used in segmentation and recognition stage of ALPR are briefed.

2.2 Overview of Bangla Characters on a License Plate

The general format of vehicle registration plates in Bangladesh is "city - vehicle class letter - vehicle number". For example: "DHAKA-KA 11-9999". The "DHAKA" field represents the city name in Bengali letters, the "KA" field represents the vehicle class in Bengali letters, the "11-9999" field represents the vehicle number of the vehicle in Bengali numerals.[1] The letters permitted in the vehicle registration plate are:
অ, ই, উ, এ, ক, খ, গ, ঘ, ঙ, চ, ছ, জ, ঝ, ত, থ, ঢ, ড, ট, ঠ, দ, ধ, ন, প, ফ, ব, ভ, ম, য, র, ল, শ, স and হ.

The numerals permitted in the vehicle registration plate are:
০, ১, ২, ৩, ৪, ৫, ৬, ৭, ৮ and ৯.
For this project, only the private vehicle License plates of Dhaka and Chattogram metropolitan area that are issued by BRTA, are taken into account. Private vehicles belong to the following classes only:

ক, খ, গ, ঘ, চ, ছ, ব, ভ, ল and হ.

The project can easily be extended to detect license plate of every class and cities, by providing proper dataset.

2.3 Related work

Feature extraction and recognition using neural network: Himel Das Gupta, Ahmad Shadi Shaon, Mujahid Al Rafi developed a system for Automatic license plate recognition system [2]. At first the image was converted to gray scale. Then Noise Elimination and Connected Component Technique were used to segment the characters. Then several feature extraction algorithms were used. And for the learning, a 3 layer ANN was trained which could classify different characters.

Morphological template matching: Template matching is a technique in digital image processing to find small parts of images that match the image template. It is a digital image processing method to compare between two similar objects by using small parts of an image that matches to the original image. Mohammad Jaber Hossain, Md Hasan Uzzaman and A.F.M. Saifuddin Saif developed a system called Bangla Digital Number Plate Recognition using Template Matching for Higher Accuracy and Less Time Complexity [3]. The paper represents a method of Bangla number plate recognition. The propose method have four main steps-preprocessing, number plate extraction, segmentation of characters and character recognition. In preprocessing step need to convert the test image in gray level first and then binarized using Otsu method. Sobel edge operator, morphological dilation and erosion are used to find the plate location. Boundary box feature in used for segmentation of characters. Then template matching technique was applied for recognition of the characters.

Line Segmentation, Orientation Algorithm and cross correlation: Another project was developed by Md. Rokibul Haque, Saddam Hossain, Sagor Roy, Nashid Alam and M. Jahirul Islam on Line Segmentation and Orientation Algorithm for Automatic Bengali License Plate Localization and Recognition [4]. For highlighting the license

plate region image morphology, horizontal and vertical extraction was used. Line Segmentation and Orientation (LSO) algorithm was proposed for segmentation which is also effective in efficient searching for signs of the license plate and finally template matching has been used for recognition. The method used was correlation with Sign House database.

All of the methods used earlier were based on basic character segmentation after applying noise reduction filter and segmentation algorithms. Most of the methods have not considered the distorted, noisy, tilted, low contrasted images that much. That is why particular effort is needed to consider those problems and hence make such feature to make the most accurate character recognition. Also all the previous methods were focused on only one car in the frame. But the proposed method can be extended to identify several license plates simultaneously from a single frame. Also using convolutional neural networks makes the proposed method more robust, accurate on real life data and faster at the same time.

2.4 Convolutional Neural Network

A Convolutional Neural Network (Con-vNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.[5] And with the usage of pooling layer, the convolutional layer becomes much more efficient and powerful. Some famous CNNs are LeNet, AlexNet, VGGNet, GoogLeNet, ResNet etc. The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. **Figure 2.1** shows the architecture of a simple Convolutional Neural Network.

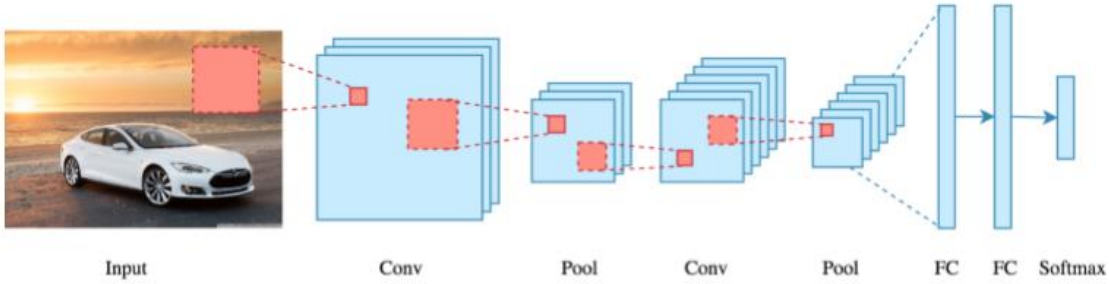

Figure 2.1: A simple Convolutional Neural Network architecture

2.5 Base Convolutional Neural Networks

A base convolutional network is needed to implement different Object Detection techniques. VGG-16, GoogLeNet (Inception-V1), and Mobilenet are the most used base networks.

2.5.1 VGG-16

VGG-16[6] is a convolutional neural network model introduced by K. Simonyan and A. Zisserman from the University of Oxford, by the renowned Visual Geometry Group. It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. **Figure 2.2** shows the architecture of the VGG-16 network.

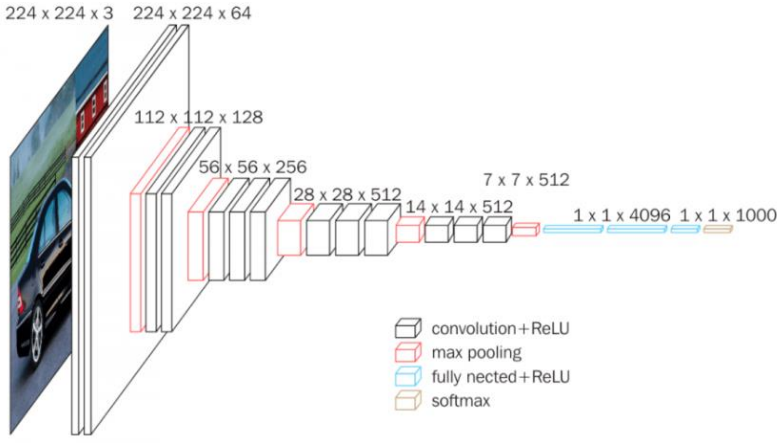

Figure 2.2: VGG-16 network architecture

The input to cov1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional (conv.) layers. The Convolution stride is fixed to 1 pixel. The spatial padding of conv layer input is such that the spatial resolution is preserved after convolution. Three Fully-Connected (FC) layers follow a stack of convolutional layers. All hidden layers are equipped with the rectification (ReLU) non-linearity.

2.5.2 Inception-v2

GoogleNet (Inception-v1), a deep CNN proposed by Szegedy et al. [7] marked another important contribution in the field of Object Detection by winning the ImageNet challenge in 2014 with a top-5 error rate of 6.7% or 93.3% accuracy. It was the first time a network exceeded 90% accuracy.

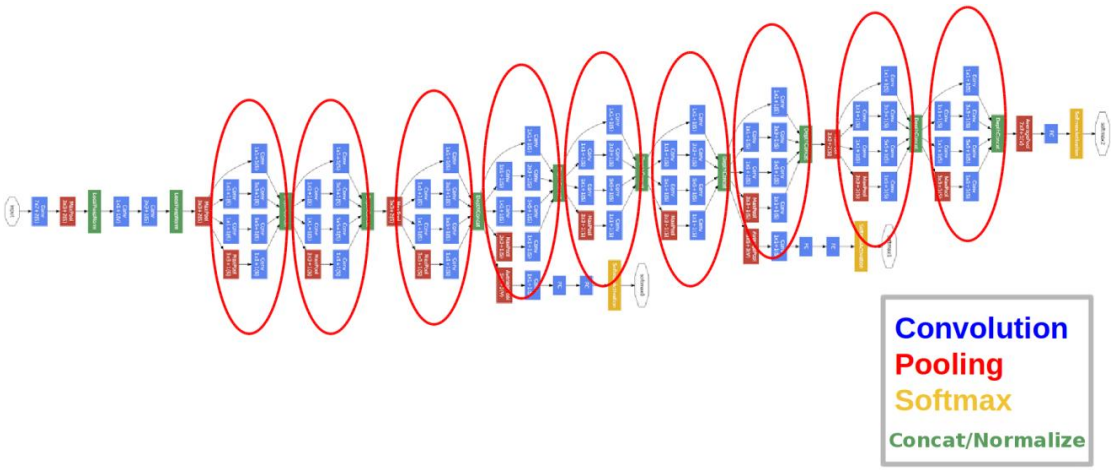

Figure 2.3: Inception-v1 network architecture

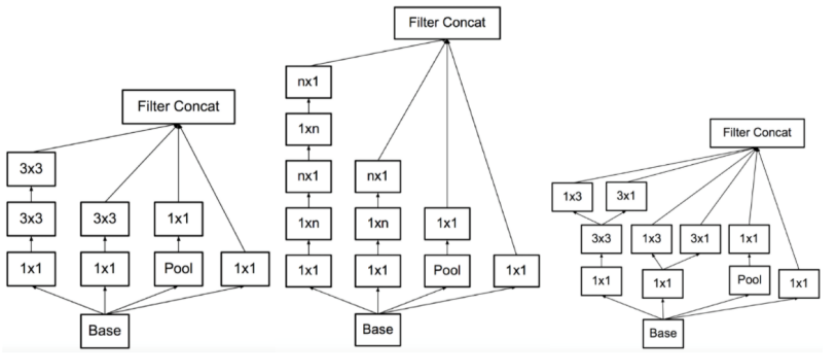

Figure 2.4: Inception blocks[22]

The network consists of several inception blocks. An inception block consists of several convolutional and pooling layers and the outputs are concatenated and fed to the next layer. **Figure 2.3** shows the architecture of Inception network. **Figure 2.4** shows different of Inception blocks. Later batch normalization was introduced to the Inception architecture and was named as Inception-V2[8].

2.5.3 Mobilenet

A group of researchers at Google proposed MobileNet, a family of architectures which encapsulates the CNN that runs efficiently on mobile devices [9]. The authors claim that the accuracy of MobileNet is comparable to VGG16 although it is 32 times smaller than VGG16. The principle idea behind MobileNet is that it uses Depthwise Separable Convolution to build quite small, low-latency Deep Neural Networks. In standard CNN, all the convolutional layers deploy regular convolution. Regular convolution means filtering all the input channels and combining them into a single output channel in one step. However, in MobileNet architecture, the first layer still uses standard convolution while all the layers use Depthwise Separable Convolution. The Depthwise Separable Convolution is factorized into two different operations of convolution: a Depthwise Convolution and a Pointwise Convolution. Unlike regular CNN, MobileNet filters the input channels in depthwise convolution and then combines them in pointwise convolution.

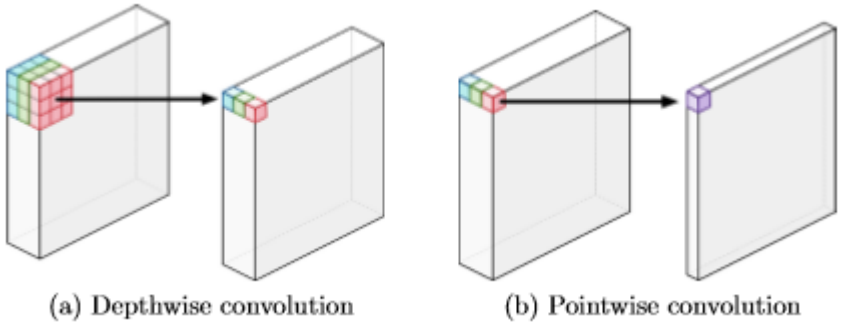

Figure 2.5: Depthwise and Pointwise Convolution

2.6 Object Detection Techniques

Object Detection is not only about recognizing or classifying objects in an image, but also about localizing them and drawing bounding boxes around them. In other words, Object Detection is an extension of image recognition.

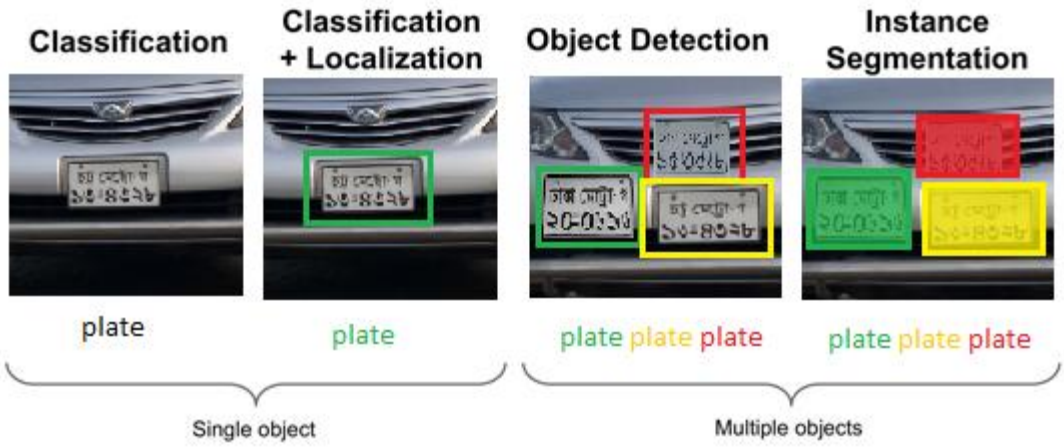

Figure 2.6: Difference between Classification, Localization, Detection and Instance Segmentation

The Object Detection method mainly consists of three steps:

• In the first step, region proposals are generated from an input image. Region proposals, also known as Regions of Interest (RoI), are a large set of bounding boxes produced by scanning the whole input image.

• In the next step, visual features are extracted given the region proposals and these features help to detect objects in an image.

• In the final step of Non-maximum Suppression, highly-overlapping boxes are grouped into a single box.

Most of the latest Object Detection routines use the Deep Convolutional Neural Networks (CNN). The most used models are Faster Region-based Convolutional Neural Network (Faster R-CNN), Region-based Fully Convolutional Network (R-FCN), and Multibox Single Shot Detector (SSD). In this section, various Object Detection techniques are discussed along with their architectures.

2.6.1 Region-based Convolutional Neural Network (R-CNN)

In 2014, Girshick et al. [10] leveraged the advantages of the Convolutional Neural Network to devise a network for Object Detection referred to as Region-based Convolutional Neural Network or R-CNN. The architecture is illustrated in **Figure 2.7**

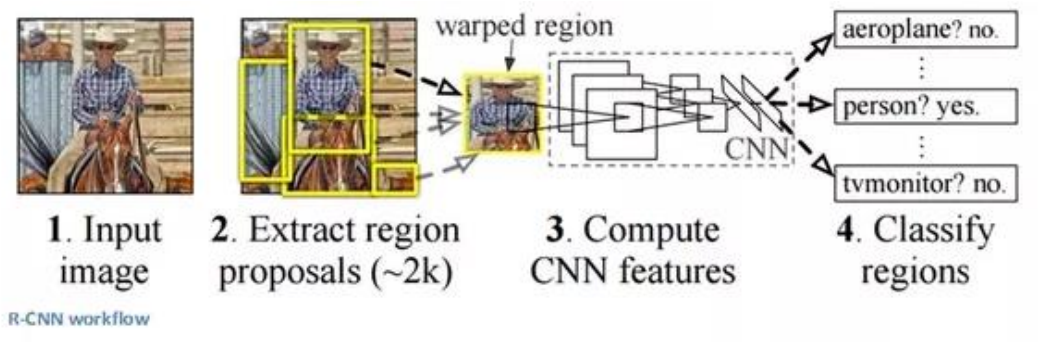

Figure 2.7: R-CNN network architecture[6]

Training a network with R-CNN involves the following three steps:

• Firstly, around 2,000 region proposals are extracted by scanning an input image with an algorithm known as Selective Search. All the proposed regions are warped to the same size. Since the size of regions generated by selective search varies and CNN accepts a fixed size input, these regions are warped (scaled) to a fixed size required by the CNN.

• Secondly, features are computed for each of the proposed regions using an independent Deep Convolutional Neural Network.

• Lastly, based on the features derived by each CNN, multiple linear SVMs are deployed to classify regions and tighten bounding boxes. Selective Search is an algorithm that generates region proposals 11].

The algorithm follows the three steps described below:

• At first, many small regions are generated using the fast method described by Felzenszwalb et al. [12]. Typically, at most one object resides in each of the generated regions.

• Next, apply a greedy algorithm to combine regions recursively.

• The regions produced in the second step are used to derive the final candidate object proposals.

To train a network with R-CNN required immense annotated datasets. On PASCAL VOC 2012 dataset, R-CNN achieved mAP (Mean Average Precision) score of 62.4%, which is 22.0% more than the second best model. Similarly, on the ImageNet 2013 dataset, R-CNN achieved 31.4% mAP score, 7% more than the second-best. Therefore, the approach adopted by the author, supervised pre-training on a large secondary dataset, followed by domain-specific fine-tuning improved the performance considerably [10].

2.6.2 Fast Region-based Convolutional Neural Network (Fast R-CNN)

As discussed above, the training process for R-CNN is multi-stage. First, a CNN is fine-tuned for each of the 2,000 region proposals; then multiple class-specific SVMs are used to classify objects followed by Linear Regressors to adjust bounding boxes. Also, the three-stage training makes the process time-consuming and expensive. Additionally, it takes 47 seconds with VGG16 (on one GPU) to detect objects for one test image, which makes R-CNN unsuitable for Object Detection in real-time applications. To improve upon the training process of R-CNN, Girshick [13] came up with a faster Object Detection algorithm in 2015 known as Fast R-CNN. The architecture is illustrated in **Figure 2.8**

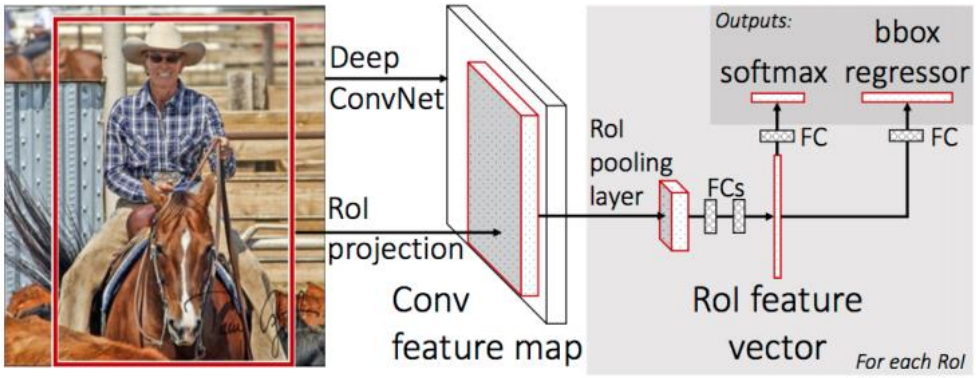

Figure 2.8: Fast R-CNN network architecture

The working of the network is explained below:

• In Fast R-CNN, an input image is fed to a single CNN with multiple Convolutional layers to generate a Convolution feature map.

• The Regions of Interest are still created with the Selective Search algorithm and provided as an input to Fast R-CNN. Each of the region proposals is warped and fed into a RoI pooling layer.

• RoI pooling layer produces a fixed-length feature vector from the feature map for each of the object proposals and passes it as an input to a series of fully connected layers.

• The fully connected layers are finally divided into two branches: the Softmax classifier predicts the class of the proposed region and the output of regression provides the four offset values for adjusting the bounding boxes.

The advantages of Fast R-CNN over R-CNN are as follows:

• Fast R-CNN involves training only one CNN for an entire image instead of training multiple CNNs for all the region proposals generated for an image.

• Multiple class-specific SVMs are replaced by single Softmax classifier, extending the neural network without training a different module.

• The above factors make the training single-stage and faster as compared to RCNN. The testing speed also improves considerably. The author claims that the speed of Fast R-CNN for training the VGG-16 network has improved nine times than the speed of R-CNN. Likewise, the test speed for Fast R-CNN has increased by 213 times as compared to that of R-CNN [13].

• Moreover, Fast R-CNN achieved the mAP score of 68.4% for the PASCAL VOC 2012 dataset improving upon the detection accuracy [13].

2.6.3 Faster Region-based Convolutional Neural Network (Faster R-CNN)

Fast R-CNN advanced the training and testing time, but the Selective Search algorithm used to generate region proposals is still one of the major bottlenecks of these networks. Selective Search is a slow and computationally expensive process to produce region proposals. Therefore, in 2015 Ren et al. [14] replaced the Selective Search with Region Proposal Network (RPN) in an Object Detection algorithm known as Faster R-CNN. Region Proposal Network, a deep fully convolutional network is used to generate a fixed number of region proposals. Faster R-CNN unifies RPN with Fast R-CNN to detect objects. The network used to generate region proposals is shown in **Figure 2.9**

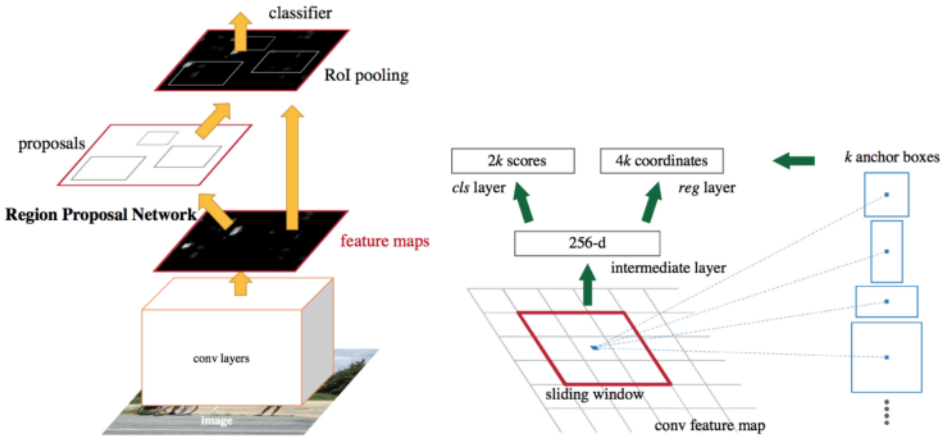

Figure 2.9: Faster RCNN network architecture

The working is described below:

• Similar to Fast R-CNN, an input image is fed to a CNN to generate convolutional feature map.

• A filter/window of size n x n is slid over the feature map generated by the last convolution layer and maps it to a lower-dimensional (256-d) feature vector.

• The output feature vector is connected to two fully connected layers, one for classification and one for regression. For each sliding window location, RPN generates a set of k region proposals with different aspect ratios. The k regions proposed by sliding window are known as anchor boxes. Each anchor box is associated with an objectness score of the box and four coordinates of the bounding box. Objectness score classifies the anchor boxes into object classes or background. Since RPN generates a maximum of k regions for each location, the output of the classification layer is 2k scores (to predict the probability whether each region is an object or not) and the output of the regression layer is 4k (corresponding to the four coordinates of each region). After detecting all the anchor boxes, the relevant region proposals are selected by applying a threshold to the objectness score. In essence, the regions whose objectness score is above a certain threshold are chosen and passed as an input to Fast R-CNN detector along with the convolutional feature map. Since RPN shares the feature map generated by CNN, the region proposal step is almost free of cost. Secondly, Faster R-CNN achieved the mAP score of 75.9% for the PASCAL VOC 2012 dataset, around 7% more than that of Fast R-CNN. Lastly, in the ImageNet Challenge and the

COCO 2015 competition, RPN and Faster R-CNN are the backbones of various approaches that won in these events [14].

2.6.4 Single-Shot Multibox Detector (SSD)

All the region-based Object Detection algorithms discussed work in two stages: first, generate region proposals and then classify those regions in a separate step. Single-Shot Detector, an Object Detection algorithm proposed by Liu et al. [15] in 2016 predicts the bounding box (region) and the class simultaneously in a single shot. Doing both the operations in one step makes SSD faster and a viable option for real-time detections. The architecture of SSD is shown in **Figure 2.10**

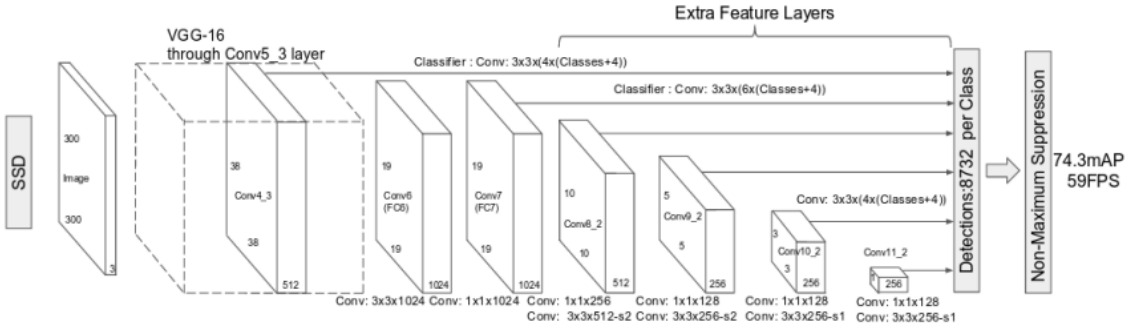

Figure 2.10: SSD Mobilenet network architecture

The working is described below:

• For the base network, the author has chosen the truncated version of VGG-16, which is image classification architecture. VGG-16 is truncated to remove the classification part and extended with extra feature layers with sizes in decreasing order. An input image passes through all the convolution layers to generate multiple convolutional feature maps of different sizes.

• Then, for each of the feature maps produced above, anchor boxes are generated in a similar way as for Faster R-CNN. A convolution filter of size says, 3x3 is slid over each feature map to create anchor boxes at different scales and aspect ratios. Each predicted box has four coordinates and a vector of class probabilities.

• At the training time, all the predicted boxes are matched with the ground truth boxes based on IoU (Intersection over Union). As a result, all the boxes with IoU > 0.5 are considered as positives, and others as negatives.

• Since there are multiple feature maps of different sizes, the numbers of bounding boxes generated are quite large. Also, most of the anchor boxes are negatives.

• To overcome the shortcomings mentioned above, firstly non-maximum suppression is deployed at the end to combine the overlapping boxes. Secondly, the technique of Hard Negative Mining (HNM) is used to balance the negative and positive training examples. In training, only a subpart of negative examples is used. The subpart is created by sorting all the negatives by the confidence loss and selecting the top ones such that the ratio of negatives to positives is 3:1. To conclude, SSD predicts the bounding boxes and class probabilities in one shot, which makes it faster than the region-based algorithms discussed above. Besides, accuracy is comparable to Faster R-CNN. Moreover, the variable sizes of the feature maps assist in generating boxes of varying dimensions, which in turn helps detect objects of different sizes [15].

2.7 Open Source Frameworks for Object Detection

Google's Tensorflow provides various pre-trained models for image classification trained on the ImageNet database and also presents several pre-trained models for Object Detection trained on datasets such as COCO, Kitti, and Open Images. Tensorflow Object Detection API supports Object Detection models such as SSD, Faster R-CNN, and R-FCN in association with image classifiers including MobileNet-v1, MobileNetv2, Inception-v2, and ResNet-101.

The following section describes Tensorflow Object Detection API in detail.

2.8 Tensorflow Object Detection API

Creating accurate machine learning models capable of localizing and identifying multiple objects in a single image remains a core challenge in computer vision. Google's

Tensorflow Object Detection API is an open source framework for Object Detection; it allows a user to define, train and utilize the models for Object Detection. Tensorflow is widely used to develop applications with deep learning [16, 17]. This API can be used to detect, with bounding boxes, objects in images and/or video using either some of the pre-trained models made available or fine tuning a pre trained model for a custom dataset. Another interesting feature of Tensorflow is that it supports a tool known as Tensorboard for in-depth visualization through a web interface of models during the training process.

To fine-tune our specific object detectors, we can choose among the pre-trained models to initialize our training. The selection of the pre-trained model is based on the purpose of our application. The API also gives an insight into speed and accuracy trade-offs of different Object Detection models.

The most important requisite of modern Object Detection applications is to achieve real-time speed. The models trained with SSD networks are pretty fast, but this speed comes at the cost of reduced accuracy. On the contrary, models trained with Faster R-CNN are more accurate but are expensive in time [18].

2.9 Fine Tuning

Fine tuning is a process to take a network model that has already been trained for a given task, and make it perform a second similar task. Fine tuning machine learning predictive models is a crucial step to improve accuracy of the forecasted results. Fine tuning means taking weights of a trained neural network and use it as initialization for a new model being trained on data from the same domain (often e.g. images). It is used to:

1. Speed up the training
2. Overcome small dataset problem

To fine tune a network, the earlier layers are kept frozen. The final fully connected layer is replaced. Only the weights of the final layer are updated while training. **Figure 2.11** shows the removal of the final fc layer having 1000 outputs with 21 outputs.

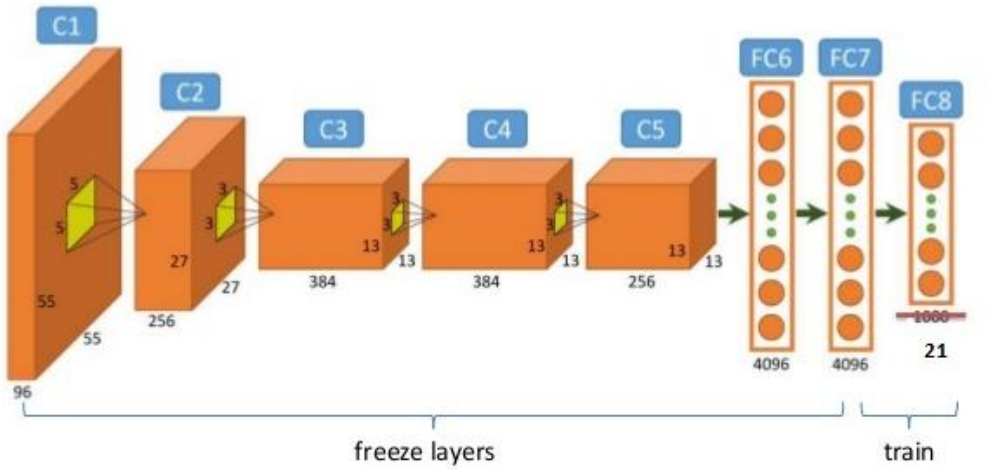

Figure 2.11: Fine tuning and replacing the final layer

2.10 Summary

In this chapter, different techniques about detecting characters on a license plate are elaborated. These methods can detect characters on a single license plate from a given still image input. Also, the state of the art approaches for Object Detection and several Object Detection network architectures were discussed, which will be used later on.

Chapter 3

METHODOLOGY

3.1 Introduction

In the previous chapter, different methods developed for Bangla character detection were briefed, but most of them were implemented for still images. Also, those proposed methods were based on hand coded threshold values for a particular lighting condition. The system might not work well with different angles of the camera and slightly deformed license plates. By eliminating these two limitations, the license plate detection can be more robust and practical in terms of usability. The TensorFlow Object Detection API can come handy handling this kind of situation. The API allows us to use a pre-trained model and fine-tune the model for our image data. Also, it performs preprocessing and data augmentation for better results.

3.2 The Principle Mechanism

When running an inference using a trained network under a Tensorflow session, an Object Detection model provides the following outputs:

```
detection_boxes ; matrix of shape (1,300,4)
detection_scores ; matrix of shape (1, 300)
detection_classes ; matrix of shape (1, 300)
num_detections ; matrix of shape (1,)
```

The model will predict 300 possible objects having 300 bounding boxes.

The detection_boxes is a matrix of shape (1, 300, 4). For 300 predicted objects, it will have the 4 bounding box coordinates – X_{min} (lower leftmost point), X_{max} (lower

rightmost point), Y_{min} (Upper leftmost point) and Y_{max} (Upper rightmost point) thus the shape (1, 300, 4)

detection_scores has a shape of (1, 300) that holds the confidence score for each of the predicted objects.

detection_classes has a shape of (1, 300) that contains the corresponding class of the 300 bounding boxes.

The matrixes are organized according to their scores. The object with the highest confidence score will be the first element of the matrixes. In this case, the characters can be considered as objects.

By taking the X_{min} coordinates of the first eight predicted bounding boxes and their corresponding class names, the characters on the license plate can be detected in the right order. As the matrixes are organized according to their confidence scores and not their position, the coordinates of the bounding boxes are taken into consideration to arrange the characters.

3.3Workflow

Two different detection models are trained individually on different datasets. The two models are then merged together for the task. The first classifier will classify the license plate from an image/frame. The second classifier will classify the characters and digits on the license plate.

3.3.1 Choosing a Pre-Trained Model

For pretraining purpose, Tensorflow detection model zoo provides a collection of detection models pre-trained on the COCO dataset, the Kitti dataset, the Open Images dataset, the AVA v2.1 dataset and the iNaturalist Species Detection Dataset. Choosing a suitable pre-trained model and fine-tuning it saves a lot of training time.

Here is the list of available pre-trained models on the COCO dataset.

Table 3.1: COCO-trained models

Model Name	Speed(ms)	mAP	Ouput
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v1_0.75_depth_coco ☆	26	18	Boxes
ssd_mobilenet_v1_quantized_coco ☆	29	18	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco ☆	29	16	Boxes
ssd_mobilenet_v1_ppn_coco ☆	26	20	Boxes
ssd_mobilenet_v1_fpn_coco ☆	56	32	Boxes
ssd_resnet_50_fpn_coco ☆	76	35	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssd_mobilenet_v2_quantized_coco	29	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_lowproposals_coco	64	NG[1]	Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82	NG	Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
fast- er_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241	NG	Boxes
faster_rcnn_nas	1833	43	Boxes
faster_rcnn_nas_lowproposals_coco	540	NG	Boxes
mask_rcnn_inception_resnet_v2_atrous_coco	771	36	Masks
mask_rcnn_inception_v2_coco	79	25	Masks
mask_rcnn_resnet101_atrous_coco	470	33	Masks

Note: The asterisk (☆) at the end of model name indicates that this model supports TPU training. [19]

The table shows the speed and mAP (Mean Average Precision) of different pre trained models on different network architectures.
Choosing a suitable model is all about tradeoffs between speed and accuracy.
From this list, faster_rcnn_inception_v2_coco was chosen for the task as it was more accurate detecting the characters. Also, the ssd_mobilenet_v1_coco was used for testing purpose to compare the detection time and accuracy.

[1] NG: Not given in the COCO Model Zoo

3.3.2 Mean Average Precision (mAP)

mAP (mean average precision) is the average of AP. Tensorflow Object Detection API uses mAP as an evaluation protocol to measure and compare the accuracy of Object Detection models. According to PASCAL metrics, mAP is calculated as the mean of average precision (AP) of all the object classes. [20]

3.3.3 Creating the dataset

To create the dataset for the first CNN which is for detecting the license plate more than 300 sample photos are being taken. After deleting some blurry, rustic and vague samples, 292 photos are chosen after refinement. The photos are then being divided for training and testing. For training, 262 photos are taken; the other 30 photos are kept for testing.

Figure 3.1: Train data samples

Figure 3.2: Test data samples

3.3.4 Selection and Annotation of Images

To label and annotate the train and test images, LabelImg is used. LabelImg is a graphical image annotation tool.[21] It is written in Python and uses Qt for its graphical interface. Annotations are saved as XML files in PASCAL VOC format, the format used by ImageNet. Besides, it also supports YOLO format. Using LableImg, a bounding box is drawn around each license plate, and the 'plate' label name is given. This creates an XML file that holds the coordinates of the bounding box along with the label name.

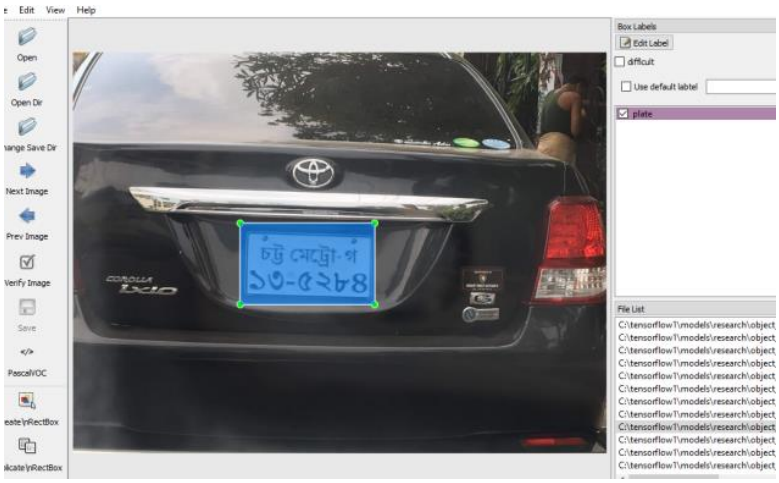

Figure 3.3: plate annotation using LabelImg

For the second CNN, two separate datasets are combined to make a larger dataset for gaining better accuracy. After training the first CNN for detecting license plates, a TensorFlow session is created, and the training and testing images are fed for detecting the license plate. Also, an OpenCV script is written to save only the detected portion of the image that is only the portion of the license plate. All the cropped license plate images are stored into a folder and annotated afterwards.

Figure 3.4: Character annotation using LabelImg

The label names were given in English, where each label name denotes the Bangla class name of the vehicle. For example, the label name for ক is given 'ka'.

Table 3.2: List of labels

Model Name	**Number of classes/Labels**	**Label Names**
Model 1 for License Plate	1	Plate
Model 2 for Characters and Numbers	21	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, bha, cha, chha, ga, gha, ha, ka, kha, la, chattametro and dhakametro

3.3.5 Creating Synthetic Data

The dataset is not enough for training because around 90% of the license plate images contained only the letter ‘গ’ in it, as most of the cars are 1500 cc vehicles belonging to the class ‘গ’, and some samples of the character ‘ঘ ‘are found, but the other characters are not found at all. For this reason, synthetic data is used for the training.

BanglaLekha-Isolated[22] is an open source dataset that is used so serve the purpose. This dataset was created for handwriting recognition. The reason behind using a handwritten character dataset for detecting printed characters is that some handwritten characters of the mentioned dataset were very close to printed characters. Only the samples which are very close to printed Bangla characters are chosen. The samples are then labeled using LabelImg along with the previous images. The BanglaLekha-Isolated dataset comes in white font on black background. For this reason all the selected images are inverted before using for the dataset.

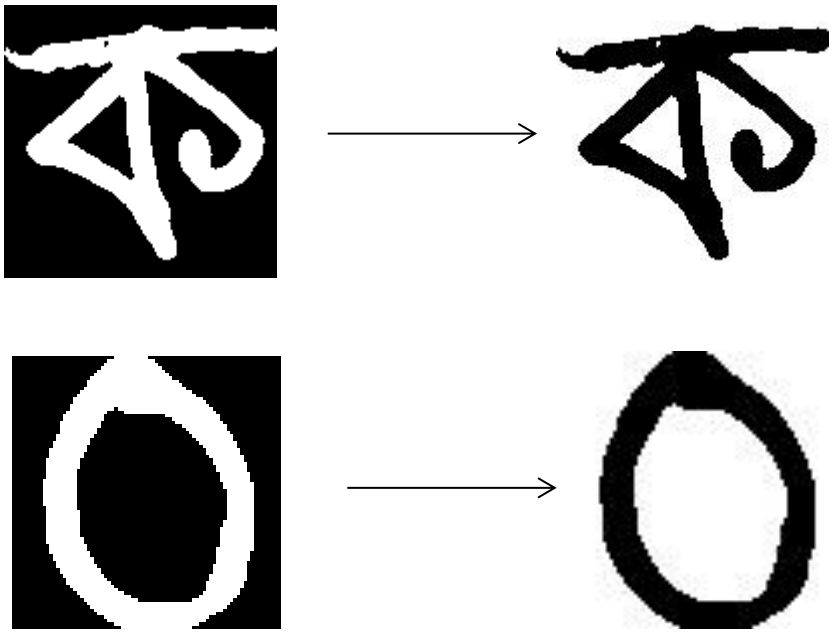

Figure 3.5: Image samples before and after the inversion.

An OpenCV script is written to go through all the directory and inverting the images on after another. The inverted images are also labeled afterwards using the LabelImg tool.

3.3.6 Creating the train and test Datasets

After completing the annotation process, the images are divided into two sets: train dataset and test dataset; for both the models. The following table shows the number of annotations and pictures for the training of two models.

Table 3.3: Total number of Annotations

Model No / Name	**No of images (train)**	**No of images (test)**	**Total No of images**	**No of Annotations (train)**	**No of Annotations (test)**	**Total No of Annotations**
Model 1 for Plate	262	30	292	262	30	292
Model 2 for Characters	622	150	772	1573	408	1981

3.3.7 Creating the TFRecords

The XML files are then converted into CSV files, and the CSV files are then converted into tf record format. The input data for training the CNNs will be fed through the tf record files. While working with large datasets, using a binary file format for storage can affect the performance of the import pipeline and as a result the training will be slow. Tfrecord file system is optimized for Tensorflow in multiple ways. Another important aspect is that with TFRecords, it is possible to store a sequence of data[23]. The helper code for generating the tfrecord files, generate_tfrecord.py[24] was taken from a Github repository by dattrain.

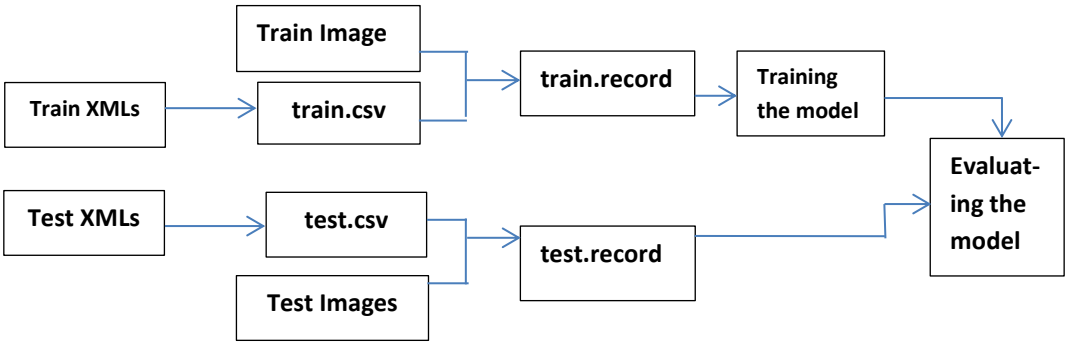

Figure 3.6: Workflow diagram of the training and evaluation process

3.3.8 Creating the label map

Label map is the mapping of the classes with its corresponding identification number (id). The ID number starts with ID: 1. The label map file is saved with the .pbtxt extension which is a protobuff file in a human-readable format. The label map will contain the number of labels that the model will have. For the first model there is only one label, and for the second model, the number of labels is 21.

```
item {
  id: 1
  name: 'plate'
}
```

Figure 3.7: label map for model detecting license plate.

```
item {
  id: 1
  name: '0'
}

item {
  id: 2
  name: '1'
}

item {
  id: 3
  name: '2'
}

item {
  id: 4
  name: '3'
}

item {
  id: 5
  name: '4'
}

item {
  id: 6
  name: '5'
}
item {
  id: 7
  name: '6'
}
```

```
item {
  id: 8
  name: '7'
}

item {
  id: 9
  name: '8'
}

item {
  id: 10
  name: '9'
}

item {
  id: 11
  name: 'bha'
}

item {
  id: 12
  name: 'cha'
}

item {
  id: 13
  name: 'chha'
}

item {
  id: 14
  name: 'ga'
}
```

```
item {
  id: 15
  name: 'gha'
}

item {
  id: 16
  name: 'ha'
}

item {
  id: 17
  name: 'ka'
}

item {
  id: 18
  name: 'kha'
}

item {
  id: 19
  name: 'la'
}

item {
  id: 20
  name: 'chattametro'
}

item {
  id: 21
  name: 'dhakametro'
}
```

Figure 3.8: label map for model detecting characters

3.3.9 Setting up the Pipeline Configuration File

The configuration file is the file that contains all the training parameters for fine tuning. All the parameters of the config file are not fine-tuned in this case. The parameters are kept unchanged except for few. The fine tune checkpoint is pointed towards the checkpoint file of the downloaded pertained model. Also the path for tfreord for train and test are pointed in the config file.

After generating the tfrecords for train and test, the training is initialized. The helper code for starting the training comes with the API itself.

3.3.10 Training the Models

After collecting all the input data for training, the next job is to set up the configuration and labelmap for the training. The pre-trained models are downloaded from Ten-

sorflow detection model zoo [19]. The models contain a checkpoint file, which will be pointed by the config file. The training will start on the pre-trained models, changing the weights, biases and also the number of output labels. After updating the configuration file according to our dataset, the Python commands to run the training and evaluation script is presented in **Figure 3.9**:

```
# Command to train a model
python train.py --logtostderr
 --train_dir=path/to/train-directory
 --pipeline_config_path=path/to/config-file

# Command to evaluate a model while training
python eval.py --logtostderr
 --checkpoint_dir=path/to/train-directory
 --pipeline_config_path=path/to/config-file
 --eval_dir=path/to/eval-directory
```

Figure 3.9: Commands for training and evaluation

3.3.11 Training Parameters, resulting Loss and mAP

The training parameters tuned in the config file for each models are described below. Also the graphs for loss and mAP after finishing the training job are illustrated and described in this section.

Model 1: Two separate models are trained for detecting the license plate, one using the faster RCNN and another using the SSD Mobilenet.

Faster RCNN

Training Parameters:

batch_size: 1

initial_learning_rate: 0.0002

optimizer: momentum_optimizer

Activation: Softmax

Steps: 53000

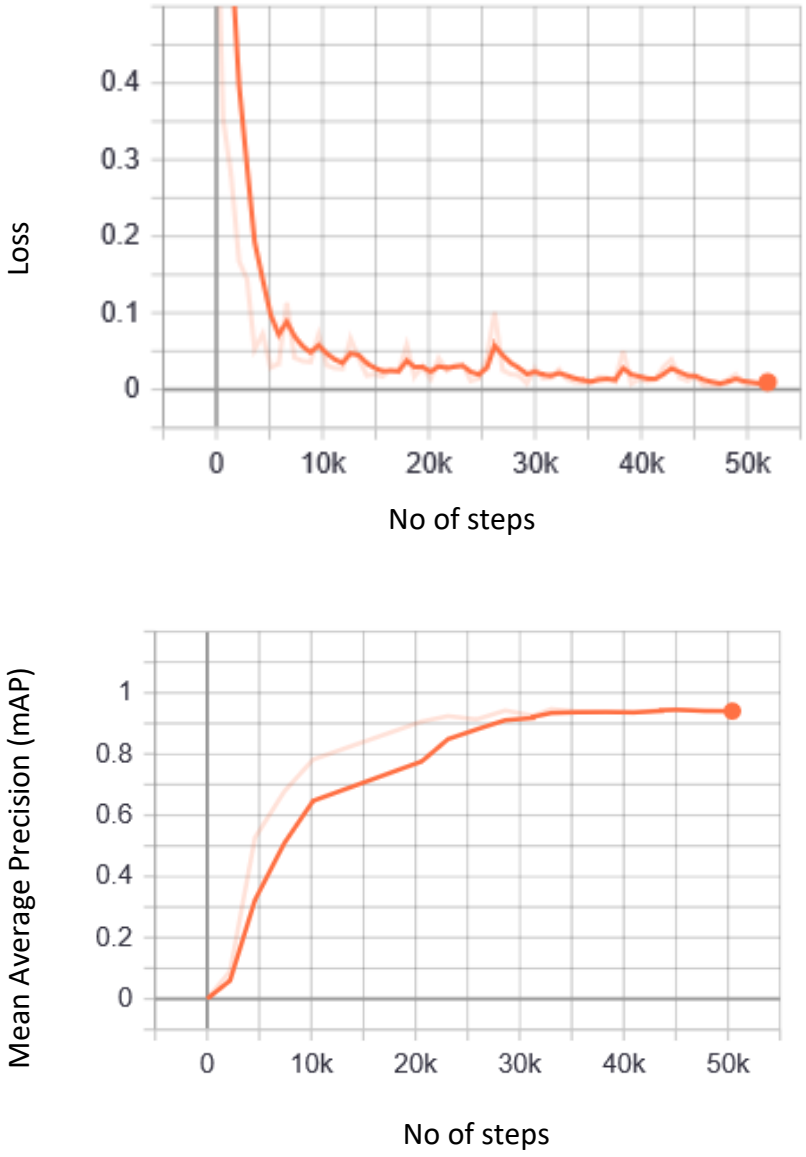

Figure 3.10: Total loss and mAP for faster RCNN (plate)

Loss and mAP: Loss is around 0.05 and mAP is around 0.93 as shown in figure 3.10

SSD Mobilenet

Training Parameters:

batch_size: 24

optimizer: rms_prop_optimizer

initial_learning_rate: 0.004

Steps: 82000

Activation: RELU_6

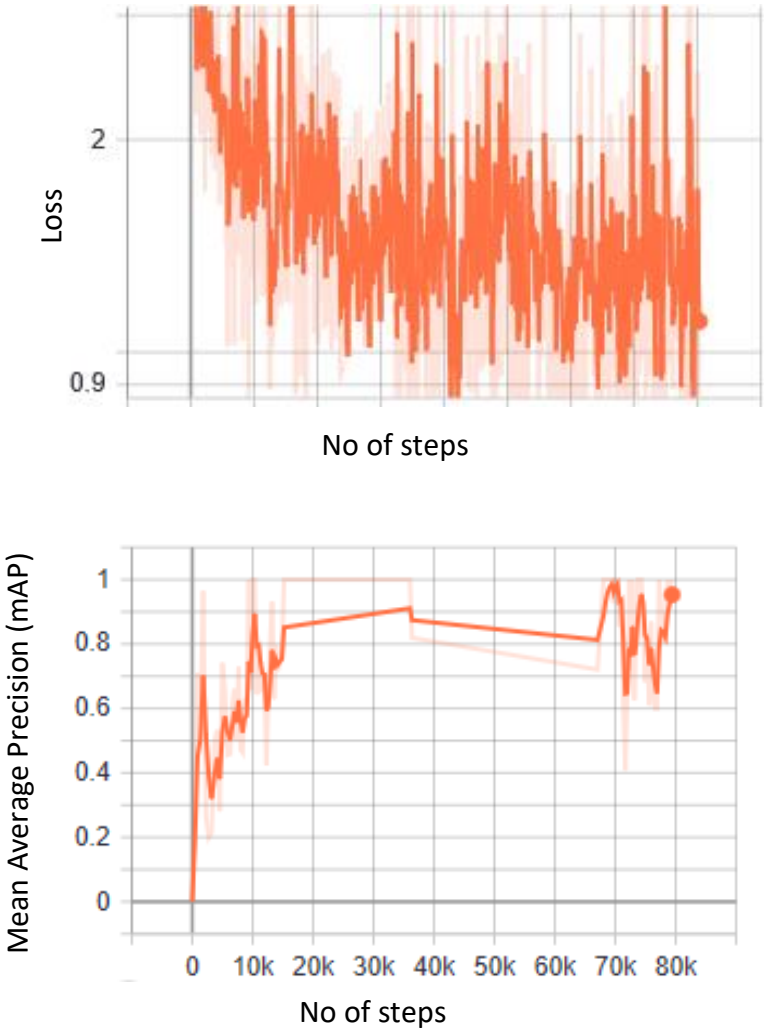

Figure 3.11: Total Loss and mAP for SSD Mobilenet (plate)

Loss and mAP: Loss is around 1.5 and mAP is around 0.95 as shown in **Figure 3.11**

Model 2: Similarly**,** 2 separate models are also trained for detecting characters on the license plate. The first one is using the Faster RCNN and the second one using the SSD Mobilenet.

Faster RCNN

Training Parameters:

batch_size: 1

initial_learning_rate: 0.0002

After 90000 steps, learning_rate: 0.00002

After 120000 steps, learning_rate: 0.000002

optimizer: momentum_optimizer

steps: 150000

Activation: Softmax

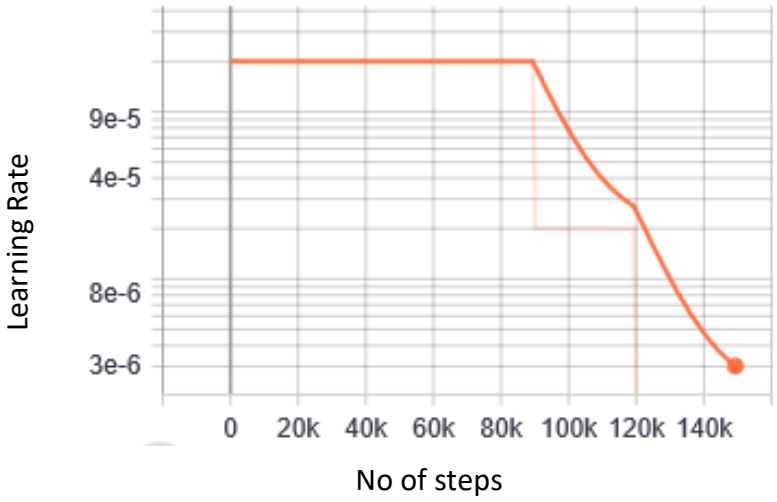

Figure 3.12: Learning Rate for Faster RCNN (character)

Figure 3.12 shows the decrement of learning rate. Initially the learning rate was set to 0.0002. After 90000 steps, it was reduced to 0.00002. And after 120000 steps, it was again reduced to 0.000002.

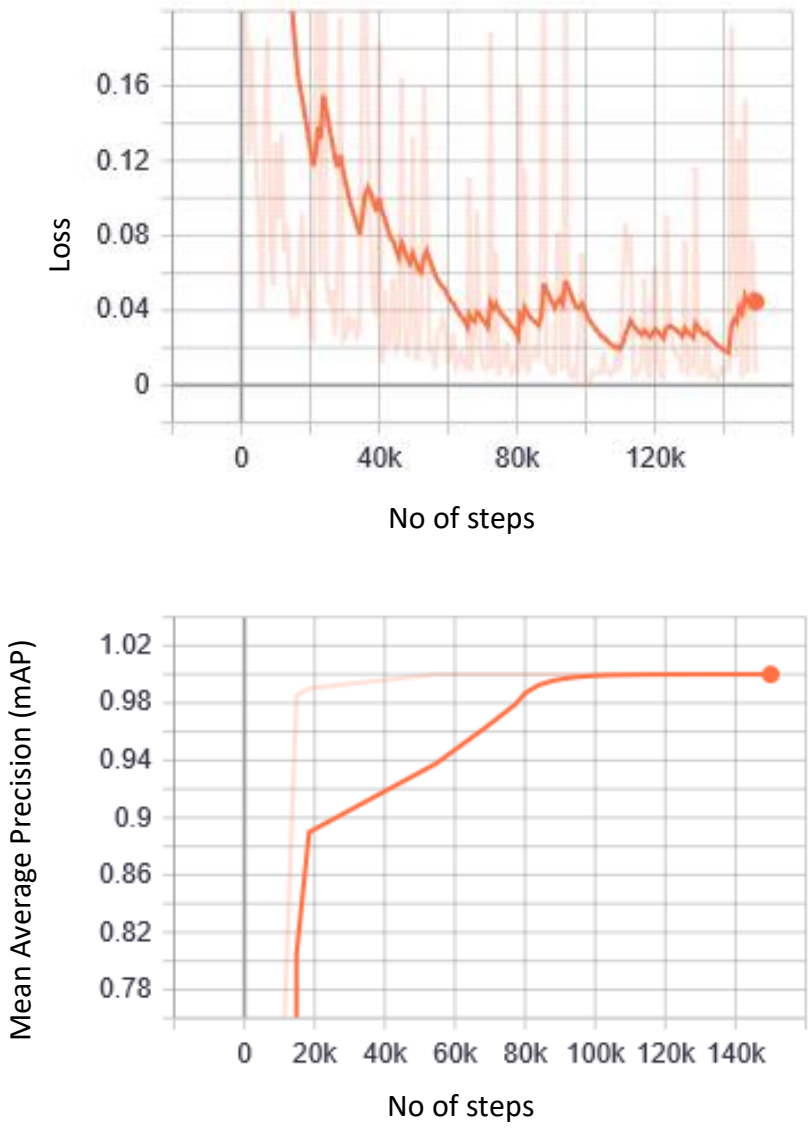

Figure 3.13: Total Loss and mAP for Faster RCNN (character)

Loss and mAP: Loss is around 0.05 and mAP is around 0.99 as shown in **Figure 3.11**

SSD Mobilenet

Training Parameters:

batch_size: 24

optimizer: rms_prop_optimizer

initial_learning_rate: 0.004

Steps: 54000

Activation: RELU_6

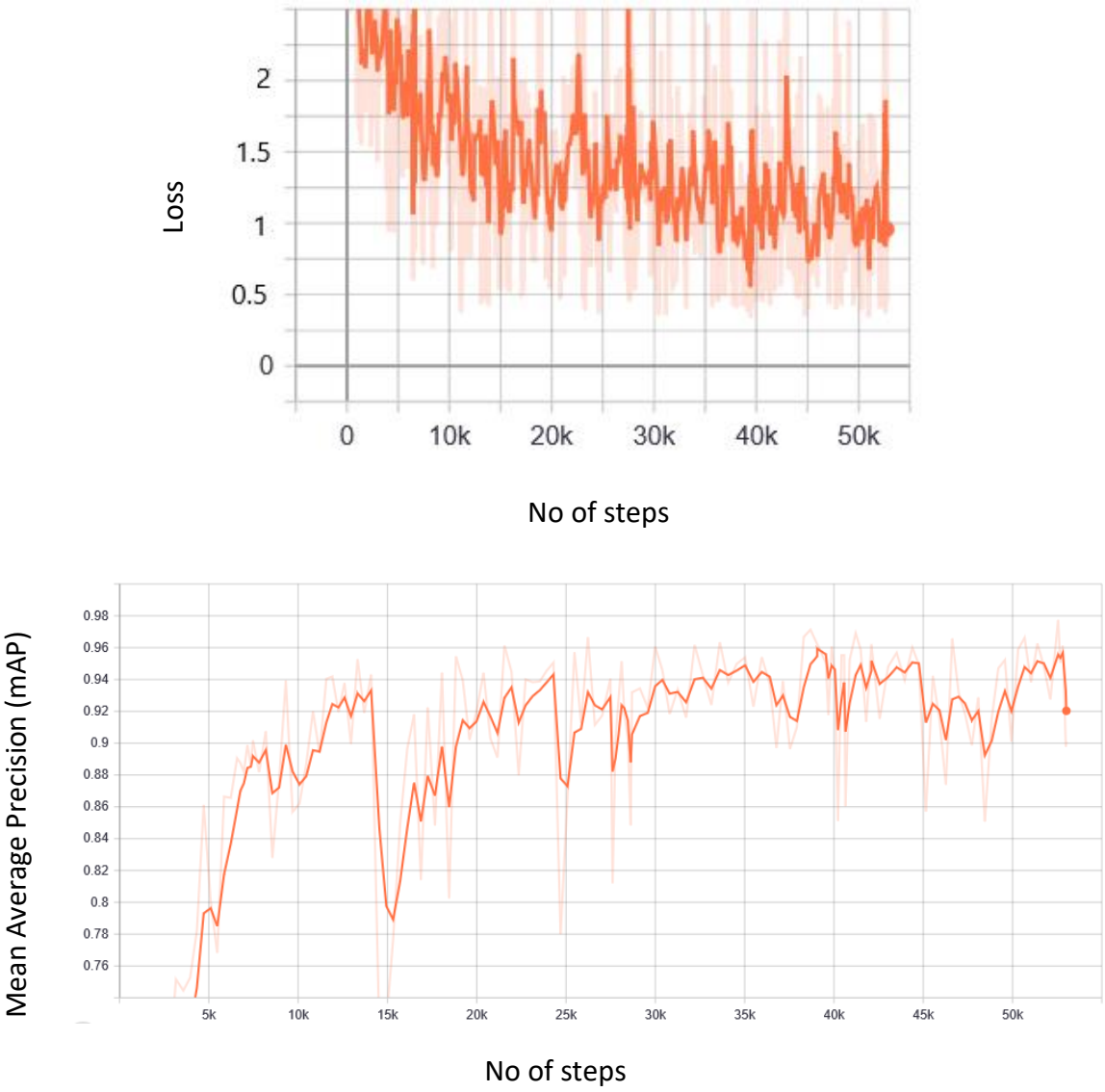

Figure 3.14: Total Loss and mAP for SSD Mobilenet (character)

Loss and mAP: Loss is around 1.5 and mAP is around 0.92 as shown in **figure 3.14**

3.3.12 Exporting the Inference Graph

The trained models are then being frozen for later inference. This is called the frozen inference graph. All the four graphs are saved and will later be used for detecting the characters. The script to export the inference graph "export_inference_graph.py" is provided with the API.

3.4 Summary

The implementation of vehicle license plate detection using Tensorflow is discussed in this chapter. To detect the license plate two several models are trained. At first the data collection technique is described and also how some data are pre-processed according to our requirements. Then the training procedure is discussed. After that, the inference graph is exported from the final checkpoint of the trained model. The next chapter the models will be deployed for inference and also the performance of the application will be analyzed.

Chapter 4

RESULT AND PERFORMANCE

4.1 Introduction

After successfully training both the classifiers, the system was tested on some new input data. All the four models which are analyzed and was tested on several input images and videos having different resolution. The analysis is done using the Nvidia Geforce GT 740M GPU.

4.2 License Plate Detection Result

After running two inferences on a single input image, the output is shown in two separate windows. The first window shows the output after detecting the license plate and drawing bounding boxes around the detected plate. The minimum threshold for detecting the license plate is set to 0.99. **Figure 4.1** shows the output after detecting the plate and drawing bounding box

Figure 4.1: Output after detecting the plate

4.3 Character Recognition Result

The second window shows the output after detecting the characters on it. The minimum threshold is set to 0.6 for detecting the characters.

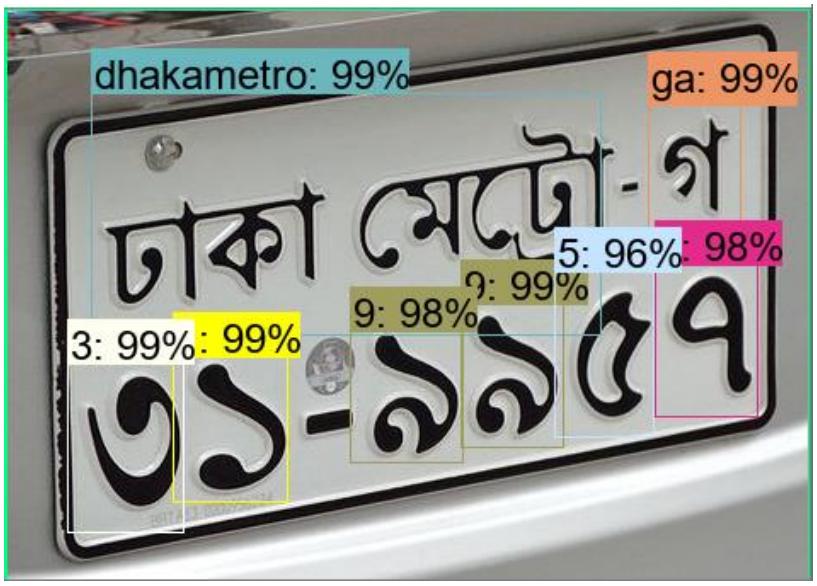

Figure 4.2: Output after detecting the characters

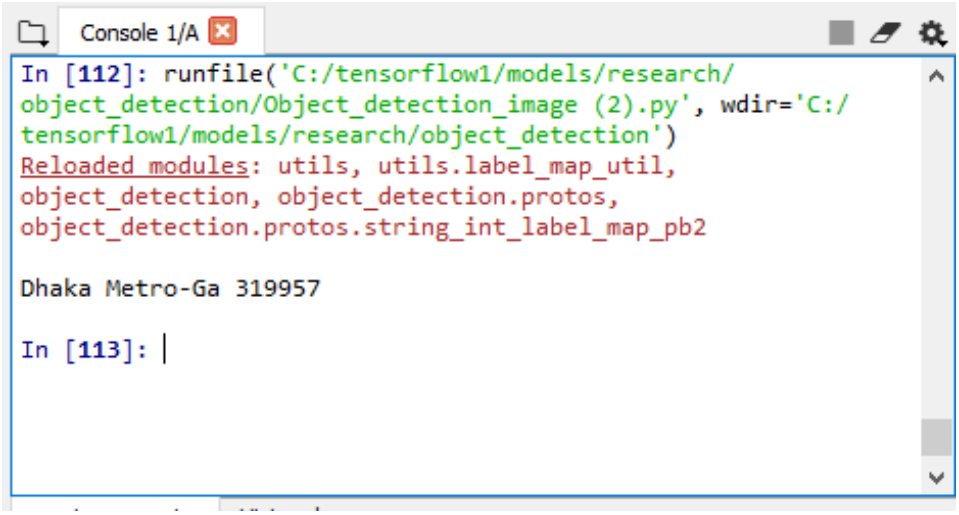

Figure 4.3: Console output

Figure 4.2 shows the output after detecting characters and drawing bounding boxes around those characters. **Figure 4.3** shows the console output.

4.4 Detection time (Merging different models)

For license plate detection, we trained both faster_rcnn_inception_v2_coco and ssd_mobilenet_v1_coco. For detecting characters on the license plate, two different models are trained similarly. The system is tested for different combinations of the models.

To measure the time it takes to process a single image and how accurately it does the job, we tested the models on several still images rather than using a video. Two of the sample images are shown in figure 4.1 and 4.2. The result is given in the following tables:

4.4.1 Sample Picture 1:

Figure 4.4: Sample picture 1 for measuring detection time.

Table 4.1: Detection time comparison of the models for sample picture 1

Model 1 for Detecting License Plate	**Model 2 for detecting Characters**	**Output**	**Elapsed time in second**
faster_rcnn_inception_v2_coco	faster_rcnn_inception_v2_coco	Detected	20.47
faster_rcnn_inception_v2_coco	ssd_mobilenet_v1_coco	Detected	17.30
ssd_mobilenet_v1_coco	ssd_mobilenet_v1_coco	Detected	12.50
ssd_mobilenet_v1_coco	faster_rcnn_inception_v2_coco	Detected	16.58

4.4.2 Sample picture 2:

Figure 4.5: Sample picture 2 for measuring detection time.

Table 4.2: Detection time comparison of the models for sample picture 2

Model 1 for Detecting License Plate	**Model 2 for detecting Characters**	**Output**	**Elapsed time in second**
faster_rcnn_inception_v2_coco	faster_rcnn_inception_v2_coco	Detected	23.72
faster_rcnn_inception_v2_coco	ssd_mobilenet_v1_coco	Not Detected	N/A
ssd_mobilenet_v1_coco	ssd_mobilenet_v1_coco	Not Detected	N/A
ssd_mobilenet_v1_coco	faster_rcnn_inception_v2_coco	Detected	17.11

4.5 Speed. Accuracy and Resolution Tradeoffs

The SSD Mobilenets worked considerably better for low resolution images, but the faster RCNN networks had better accuracy with larger images. The SSD Mobilenet is faster compared to the Fastre RCNN, but accuracy is its main drawback. Also, the SSD Network couldn't detect objects (characters) and considered ground truth, whereas there are actual objects (characters) on the image. Choosing different networks for inference is a tradeoff between these three factors: speed, accuracy, and resolution. For this project, Faster RCNN is chosen for both plate and character recognition.

4.6 Results for video inputs taken from different angles

The system is tested for different angle of the stationary camera. Two random number plates are taken for the testing purpose. The speed of the vehicle is 20Kmph. The resolution of the video files are 1080p(30 fps), 2k(30 fps), and 4k(60 fps). The devices used for the recording purpose are:

1. iPhone 8 Plus (12 Megapixel)
2. OnePlus 6 (20 Megapixel)
3. Huwawei (13 Megapixel)

The result is shown in the following two tables:

Table 4.3: Output for **Dhaka Metro – Ga 105244**

Vertical Angle in Degree	**Horizontal Angle in Degree**	**Resolution and FPS**	**Output**
0	15	1080P 30FPS	Detected
0	30	2K 30FPS	Detected
0	40	4K 60FPS	Detected
15	15	1080P 30FPS	Detected
15	30	2K 30FPS	Detected
15	40	4K 60FPS	Detected
30	15	1080P 30FPS	Detected
30	30	2K 30FPS	Detected
30	40	4K 60FPS	Detected

Table 4.4: Output for **Chatta Metro – Ka 129021**

Vertical Angle in Degree	**Horizontal Angle in Degree**	**Resolution and FPS**	**Output**
0	15	1080P 30FPS	Detected
0	30	2K 30FPS	Detected
0	40	4K 60FPS	Detected
15	15	1080P 30FPS	Detected
15	30	2K 30FPS	Detected
15	40	4K 60FPS	Detected
30	15	1080P 30FPS	Detected
30	30	2K 30FPS	Detected
30	40	4K 60FPS	Detected

4.7 Performance Analysis

Total 24 samples (18 videos and 6 still images) are taken to test the application. 4 of the still images are of lower resolution and 2 of them are of high resolution. The system successfully detected the plate from different resolution images but could not detect characters on the plate from images with lower resolution.

4.7.1 Plate detection Precision: $\frac{24 \times 100}{24} = 100\%$

4.7.2 Character recognition Precision: $\frac{22 \times 100}{24} = 91.67\%$

4.7.3 Testing Domain

The angle between the cameraand the vehicle is kept within a range of -15 to 45 degree. The horizontal angle is in a range of 0 to 45 degree. The analysis is done for a speed of 20Kmph of the vehicle. The lighting was moderate as all the videos are taken in the evening time.

4.7.4 System Constraints

The system did not work effectively for lower resolution images and videos as Faster RCNN network was chosen for both plate detection and character recognition. The SSD Mobilenet works considerably better with lower resolution images.

4.8 Summary

In this chapter, the performance of the application in different condition is discussed. We tested the models and selected the combination which is the best considering the

accuracy. Also, we tested the application for different angles of the stationary camera. Finally, we calculated the precision on the given test data. The application is 100% accurate in detecting the license plate and 91.6% accurate in detecting the characters on the license plate.

Chapter 5

CONCLUSION

5.1 Summary

In this project, a new approach for license plate detection and character recognition is introduced, that is based on Object Detection. Object detection is a technology which is a computer vision technique for detecting multiple semantic objects in an image. Tensorflow Object Detection API is widely used for detecting objects, which includes localization as well. The main objective of our thesis is to deploy an Object Detection model to recognize the plate from an image frame and also to recognize the characters on it. To serve this purpose, we trained two image classifiers; one detecting license plate and another for recognizing the characters. We used this API because to detect a license plate, we not only need to classify the characters on a license plate but also need to know the location of the characters on the plate so that the license plate numbers and characters can be identified in the right order.

5.2 Future Work

This section discusses selected ideas for future work that can further enhance the accuracy and usefulness of the License plate Recognition Application:

1. This project can be extended to detect multiple license plates from a single frame.
2. The application will be able to predict more precisely if a larger dataset is provided for training.
3. The application will work much faster if it can be trained on SSD Mobilenet with a larger dataset.

4. The project can easily be extended to detect license plate of every class and cities, by providing a proper dataset.

Bibliography

[1] Vehicle registration plates of Bangladesh. 2011. Wikipedia contributors. https://en.wikipedia.org/w/index.php?title=Vehicle_registration_plates_of_Bangladesh&oldid=890896273

[2] Himel Das Gupta, Ahmad Shadi Shaon, Mujahid Al Rafi. (2017). Automatic Bangla License Plate Recognition. (2017). Bachelor's Thesis, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology.

[3] Mohammad Jaber Hossain, Md Hasan Uzzaman, Saifuddin Saif. 2018. Bangla Digital Number Plate Recognition using Template Matching for Higher Accuracy and Less Time Complexity. Published on International Journal of Computer Applications 181(29):15-21.

[4] by Md. Rokibul Haque, Saddam Hossain, Sagor Roy, Nashid Alam, M. Jahirul Islam. 2016. Published on International Journal of Computer Applications (0975 – 8887) Volume 154 – No.9.

[5] A Comprehensive Guide to Convolutional Neural Networks. 2018. Sumit Saha. Published on https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014. 1, 2, 10, 13, 16, 35

[7] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In Proceedings IEEE Conference on Computer Vision and Pattern Recognition, pages 1–9. IEEE, 2015.

[8] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[9] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017.

[10] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings IEEE Conference on Computer Vision and Pattern Recognition, pages 580–587. IEEE, 2014. x, 22, 23, 24

[11] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. International Journal of Computer Vision, 104(2):154–171, 2013.

[12] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. International Journal of Computer Vision, 59(2):167–181, 2004.

[13] Ross Girshick. Fast r-cnn. In Proceedings IEEE International Conference on Computer Vision, pages 1440–1448. IEEE, 2015.

[14] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in Neural Information Processing Systems, pages 91–99, 2015.

[15] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In European Conference on Computer Vision, pages 21–37. Springer, 2016.

[16] Ladislav Rampasek and Anna Goldenberg. Tensorflow: Biology's gateway to deep learning? Cell Systems, 2(1):12–14, 2016.

[17] Peter Goldsborough. A tour of Tensorflow. arXiv preprint arXiv:1610.01178, 2016.

[18] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In Proceedings IEEE Conference on Computer Vision and Pattern Recognition, volume 4. IEEE, 2017.

[19] pkulzc, tombstone, nealwu. github.com, 'Tensorflow detection model zoo ', July, 2019. [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md#tensorflow-detection-model-zoo [Accessed: 1- Oct- 2019].

[20] Jonathan Hui, medium.com,' mAP (mean Average Precision) for Object Detection', March, 2018. [Online]. Available: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173 [Accessed: 1- Oct- 2019].

[21] Tzutalin. LabelImg. Git code (2015). https://github.com/tzutalin/labelImg

[22] Nabeel Mohammed, Sifat Momen, Anowarul Abedin, Mithun Biswas, Rafiqul Islam, Gautam Shom, Md. Shopon . 2017. BanglaLekha-Isolated. https://data.mendeley.com/datasets/hf6sf8zrkc/2

[23] Thomas Gamauf, medium.com, Tensorflow Records? What they are and how to use them', March, 2018. [Online]. Available: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173 [Accessed: 1- Oct- 2019].

[24] datitran, github.com, 'Raccoon Detector Dataset ', Dec, 2018. [Online]. Available: https://github.com/datitran/raccoon_dataset [Accessed: 1- Oct- 2019].