


Real-time Traffic Object Detection Using DNN Frameworks Centering Bangladesh

Approved by:



Dr. Golam Sarowar

Supervisor and Professor,
Department of Electrical and Electronic Engineering,
Islamic University of Technology (IUT),
Boardbazar, Gazipur-1704.

Date:/05/2022

Declaration of Authorship

This is to clarify that the work in this thesis paper is the outcome of research carried out by the students under the supervision of Dr. Golam Sarowar, Professor, Department of Electrical & Electronic Engineering (EEE), Islamic University of Technology (IUT).

Authors

Nafis

Nafis Shahriar Munir

ID: 170021026

Nazia Hossain

Nazia Hossain

ID: 170021066

Zame

Raghib Ryyan Zame

ID: 170021156

Table of Contents

| | |
|--|----|
| List of Figures | 1 |
| List of Tables | 3 |
| List of Acronyms | 4 |
| Acknowledgement | 5 |
| Abstract | 6 |
| Chapter 1 | 7 |
| Introduction | 7 |
| 1.1 Vehicle Detection..... | 7 |
| 1.2 Object Detection vs. Image Classification..... | 7 |
| 1.3 Real Time Object Detection..... | 8 |
| Chapter 2 | 9 |
| Literature Review | 9 |
| 2.1 Literature Review..... | 9 |
| 2.2 Research Objective | 10 |
| Chapter 3 | 12 |
| How YOLO Works | 12 |
| 3.1 Introduction..... | 12 |
| 3.2 Concept of YOLO..... | 12 |
| 3.2 Architecture of YOLOv1 | 14 |
| 3.3 YOLOv5 | 16 |
| 3.3.1 Architecture..... | 17 |
| 3.3.2 Activation Function | 20 |
| 3.3.3 Optimization Function | 20 |
| 3.3.4 Cost Function..... | 20 |
| 3.3.5 Detection in YOLO..... | 20 |
| 3.3.6 Bag of Freebies | 22 |
| 3.3.7 Bag of Specials | 23 |
| Chapter 4 | 24 |
| How MobileNet Works | 24 |
| 4.1 Introduction..... | 24 |
| 4.2 Convolutional Blocks for MobileNetv2..... | 25 |

| | |
|--|----|
| 4.3 Overall Architecture..... | 26 |
| 4.4 Depthwise Separable Convolution..... | 27 |
| Chapter 5 | 30 |
| Research Methodology | 30 |
| 5.1 Dataset..... | 30 |
| 5.2 Training Parameters | 34 |
| 5.2.1 MobileNetV2 | 34 |
| 5.2.2 YOLOv5 | 34 |
| 5.3 Requirements | 34 |
| 5.3.1 MobileNetV2 | 34 |
| 5.3.2 YOLOv5 | 34 |
| 5.4 Installation..... | 35 |
| 5.4.1 MobileNetV2 | 35 |
| 5.4.2 YOLOv5 | 35 |
| 5.5 Platform and System Specifications | 35 |
| 5.5.1 Local Machine | 35 |
| 5.5.2 Cloud..... | 36 |
| 5.6 Training..... | 36 |
| 5.6.1 MobileNetV2 | 36 |
| 5.6.2 YOLOv5 | 36 |
| 5.7 Research Timeline | 39 |
| 5.7.1 Phase 1 | 39 |
| 5.7.2 Phase 2 | 40 |
| 5.7.3 Phase 3 | 40 |
| 5.7.4 Phase 4 | 40 |
| Chapter 6 | 42 |
| Results | 42 |
| 6.1 MobileNetV2 | 42 |
| 6.1.1 Inference on Images..... | 42 |
| 6.1.2 Inference on Video..... | 45 |
| 6.2 YOLOv5 | 45 |
| 6.2.1 Inference on Images..... | 47 |
| 6.2.2 Inference on Video..... | 49 |
| 6.2.3 Data Augmentation and Training on YOLOv5 | 49 |
| 6.3 Result Comparison..... | 50 |

| | |
|--------------------------------|----|
| Chapter 7 | 52 |
| Conclusion | 52 |
| 7.1 Project Significance | 52 |
| 7.2 Future Prospects..... | 53 |
| References | 54 |

List of Figures

| | |
|--|----|
| Figure 3.1: A 7x7 grid cell YOLO Model Applied on an Image..... | 13 |
| Figure 3.2: Parameters of a Bounding Box..... | 13 |
| Figure 3.3: Architecture of YOLOv1 | 14 |
| Figure 3.4: Layers, Filters and Output Dimensions of YOLO | 15 |
| Figure 3.5: Accurate and Predicted Box From the Model..... | 16 |
| Figure 3.6: Calculating IOU | 16 |
| Figure 3.7: Double-staged Detector Concept | 17 |
| Figure 3.8: A Visual Representation of YOLOv4 Design..... | 18 |
| Figure 3.9: A Deep Network Consisting of Three Dense Blocks..... | 18 |
| Figure 3.10: (a) FPN (b) PAN (c) Bottom-up Augmentation Path Connection | 19 |
| Figure 3.11: YOLOv5 Architecture Using Block Diagram..... | 19 |
| Figure 3.12: Anchor boxes and output vectors..... | 21 |
| Figure 3.13: Anchor boxes 1 and 2..... | 22 |
| Figure 3.14: Multiple Optimization Methods That Were Experimented..... | 22 |
| Figure 4.1: Image Processing by MobileNet Architecture | 24 |
| Figure 4.2: MobileNetV1 and MobilenetV2 comparison..... | 25 |
| Figure 4.3: Overall Architecture of MobileNetV2 | 26 |
| Figure 4.4: Gx(left), Gy (right)..... | 27 |
| Figure 4.5: Depthwise Separable Convolution..... | 28 |
| Figure 4.6: Depthwise Convolution..... | 29 |
| Figure 4.7: Pointwise Convolution | 29 |
| Figure 5.1: Summary of the Dataset | 30 |
| Figure 5.2: Class Balance of the Dataset | 31 |
| Figure 5.3: Different Classes of Vehicles in Dhaka City (1)..... | 32 |
| Figure 5.4: Different Classes of Vehicles in Dhaka City (2)..... | 32 |
| Figure 5.5: Image Size Distribution of the Dataset | 33 |
| Figure 5.6: Loss vs Step Curve of MobileNetV2 | 36 |
| Figure 5.7: Performance Matrix vs Epochs | 37 |
| Figure 5.8: Confusion Matrix of YOLOv5 | 37 |
| Figure 5.9: Label Instances..... | 38 |
| Figure 5.10: Research Progression Timeline..... | 39 |
| Figure 6.1: Detection Box mAP of MobileNetV2 | 42 |
| Figure 6.2: MobileNetV2 Image Inference 1..... | 43 |
| Figure 6.3: MobileNetV2 Image Inference 2..... | 43 |
| Figure 6.4: MobileNetV2 Image Inference 3..... | 44 |
| Figure 6.5: MobileNetV2 Image Inference 4..... | 44 |
| Figure 6.6: QR Code of MobileNetV2 Video Inference | 45 |
| Figure 6.7: Mean Average Precision of YOLOv5 | 45 |
| Figure 6.8: Precision vs Confidence Curve | 46 |
| Figure 6.9: Precision vs Recall Curve | 46 |
| Figure 6.10: Precision vs Confidence Curve | 47 |
| Figure 6.11: YOLOv5 Image Inference 1..... | 47 |
| Figure 6.12: YOLOv5 Image Inference 2..... | 48 |

| | |
|---|----|
| Figure 6.13: YOLOv5 Image Inference 3 | 48 |
| Figure 6.14: YOLOv5 Image Inference 4 | 48 |
| Figure 6.15: QR Code of YOLOv5 Video Inference | 49 |
| Figure 6.16: Dataset Augmentation Summary | 50 |
| Figure 6.17: mAP for Augmented Dataset | 50 |
| Figure 6.18: Comparison between YOLO and MobileNetV2 | 51 |

List of Tables

Figure 3.1: Parameters of the Anchor Boxes.....21

List of Acronyms

| | |
|---------------|--|
| YOLO | You Only Look Once |
| IP | Internet Protocol |
| CNN | Convolutional Neural Network |
| R-CNN | Region Based Convolutional Neural Networks |
| FPS | Frames Per Second |
| GPU | Graphics Processing Unit |
| SVM | Support Vector Machine |
| mAP | mean Average Precision |
| UAV | Unmanned Aerial Vehicle |
| PC | Personal Computer |
| IoU | Intersection over Union |
| CPU | Central Processing Unit |
| VGG | Visual Geometry Group |
| FPN | Feature Pyramid Network |
| MSE | Mean Square Error |
| SPM | Spatial Pyramid Matching |
| SE | Squeeze and Excitation |
| ReLU | Rectified Linear Unit |
| NMS | Non-Maximum Suppression |
| CUDA | Compute Unified Device Architecture |
| OpenCV | Open Source Computer Vision |
| cuDNN | CUDA Deep Neural Network |
| CC | Communicating Cores |
| MSVC | Microsoft Visual C++ |
| CMD | Command Prompt |
| RAM | Read Only Memory |
| SUV | Sport Utility Vehicles |
| AI | Artificial Intelligence |
| QR | Quick Response |
| PANet | Path Aggregation Network for Instance Segmentation |
| CNN | Convolutional Neural Network |
| RNN | Residual Neural Networks |
| DWSC | Depth wise Separable Convolution |

Acknowledgement

Praise and glory be to Allah Subhanahu wa Ta'ala, the All-Merciful, Almighty for enabling this research via His unending bounty.

We are eternally grateful to our honorable supervisor Professor Dr. Golam Sarowar sir, for providing us the chance to do this research work. Throughout the project, he provided us with invaluable advice and support which helped us greatly while doing this this work.

We'd like to express our gratitude to the other members of the thesis committee for their invaluable assistance. Lastly, we would like to convey our sincere gratitude to our family members for their unconditional love and support and for their prayers that always kept us firmly motivated.

Abstract

In this thesis, the research is mainly focused on traffic object detection by different computer vision algorithm like MobileNetV2 and YOLOv5. The main focus is on real-time detection and identification of various types of automobiles specific to Bangladesh. Vehicle detection is a necessary step for traffic surveillance and autonomous vehicles. Vehicle counting in complex transportation conditions requires the detection and tracking of mobile vehicles. Vehicle detection on the road are used for vehicle tracking, vehicle traffic assessment, average velocity of each individual vehicle, motion analysis, and vehicle classification, and may be applied in a variety of contexts. Because of the irregular traffic, the variety of vehicles, and the absence of a good dataset, it is more difficult in Bangladesh to adopt a smart traffic. One of the most important aspects of algorithm training is data quality. Here, the dataset, “Dhaka Traffic Detection Challenge Dataset” was cleaned and augmented to get better results. The dataset was trained on two neural network architecture, YOLOv5 and MobileNetV2. YOLOv5 ran on PyTorch and the model was trained on Google Colaboratory, a cloud-based platform. Codes were written in Python 3.8 and Python 3.9. Roboflow, an online-based computer vision application, was used to organize the dataset for training. For image categorization and mobile vision, MobileNet is built on the CNN architectural model. There are other models available, but MobileNet is unique in that it uses very minimal compute resources to operate and also applies transfer learning. As a result, MobileNet is ideal for both mobile devices and web browsers. There are 28 levels in MobileNet. MobileNet contains 4.2 million parameters by default. YOLOv5 performed better than MobileNetV2, in terms of accuracy and inference time. This research will be a vital step towards intelligent traffic detection system that can detect unauthorized vehicles like rickshaw/CNGs in highways, or to develop a traffic plan that minimizes traffic congestion on the road.

Chapter 1

Introduction

1.1 Vehicle Detection

Vehicle detection and tracking systems has a number of applications, including regulation of highway traffic, assessing, and controlling. These techniques are used for vehicle tracking, counts, measuring vehicle speeds, traffic analysis, and vehicle classification, and also utilized in a number of situations. With recent development in technology, adoption of smart traffic system is becoming increasingly popular. Vehicle detection is the process of allowing autonomous systems to view and recognize traffic vehicles in order to make essential judgments about vehicle management, resource allocation, and traffic control based on the data collected. Many approaches have been employed in this sector. However, they can be challenging to apply in real-time traffic monitoring and self-driving car applications. Because of our unique cars and demanding traffic situation, the work is considerably more difficult for Bangladesh. For example, making any autonomous system grasp the complicated geometry of a Rickshaw and distinguishing it from that of a bicycle in a congested traffic scenario in real time using traditional approaches may be rather challenging. To make the job easier and more feasible, we used machine learning-based computer vision to train a neural network to recognize and classify 21 types of common automobiles in Bangladesh. We used the YOLO [1] and the MobileNet [2] architecture to make the systems see cars in real time.

1.2 Object Detection vs. Image Classification

However, there is a slight but significant distinction between image and object recognition. The AI model assigns a single high-level label to an image or video in image recognition. The AI model detects each and every significant thing in the picture or video in object recognition. In classification, we typically have been using a CNN to predict merely the object's name. However, in order to find the item within the image, we construct bounding boxes around it [3]. This is more

challenging for a variety of reasons. Because several items may exist inside the same picture, the output layer length cannot be fixed. A crude way to solve this issue would be to first select different regions of interest from the image, and then use a CNN, such as R-CNN, to classify the object within that region. The drawback is that the items might appear in various positions and have various aspect ratios. As a result, the algorithms must choose a large number of areas, which takes a lot of computing power.

1.3 Real Time Object Detection

Real time object recognition and tracking is a broad, exciting, yet inconclusive and difficult topic in computer vision. Researchers are constantly devising more efficient and competitive algorithms as a result of its expanding use in surveillance, tracking systems utilized in security, and many other applications. For our car detection challenge, we need our system to be able to identify objects quickly enough to be used in real-time applications. Our detection system will receive a live video feed from a camera, which will consist of several photos collected in a short period of time, rather than a single image. As a result, the detecting algorithm must process a large number of photos quickly. This is a significant difficulty that necessitates an efficient design as well as machines with sufficient processing capability. R-CNN [4], Fast R-CNN [5], and Faster R-CNN [6] are examples of machine learning-based object detection architectures.

Chapter 2

Literature Review

2.1 Literature Review

Object detection has progressed significantly in the last decade thanks to developments in deep learning and convolutional neural networks. Computer vision has made significant progress in terms of accuracy, instance segmentation, and object identification. With the advancement of the Graphic Processing Unit (GPU), real-time object identification has become more feasible. We'll look at some fantastic works based on object detection.

1. The authors of this paper [7] looked at the consequences and reasoning behind several architectural and model changes made to the popular YOLOv5 object detector in order to improve its small-object recognition capabilities.
2. This study suggests [8] that in the near future, it might be used in autonomous vehicle situations to identify various road assets in various weather circumstances.
3. With the aid of the PYNQ Z2 board and Movidius NCS, the FPGA-based implementation of object detection [9] and recognition results in improved performance. The YOLO model is the quickest and easiest of the three, although it has lower accuracy and FPS than FRCNN. The SSD provides the maximum frame rate while also balancing calculation time and accuracy.
4. A comparison was proposed by Bilel et al. [10] for vehicle detection between Faster RCNN and YOLOv3. It was based on precision, F1 score, recall, quality and processing time. The study shows that YOLO outperforms R_CNN for most of the pictures.
5. Hou-Ning et al. [11] suggest an online network architecture from a series of pictures to track and detect vehicles. Still images of different angles are used for this purpose which extend the working load for detecting vehicles.

6. A bidirectional cooperation between recognition and tracking is analysed by Foresti et al. [12], these assign semantic levels, establishing identity and pose correspondence between objects detected at various time instants.
7. Zehang Sun et al. [13] explore multi-scale driven hypothesis generation and appearance based hypothesis verification to present an in-vehicle real-time monocular precrash vehicle detection system. Haar Wavelet decomposition for feature extraction and Support Vector Machines (SVMs) for classification were used here for appearance based hypothesis verification.
8. In another research paper [14], they used the same methodology to detect vehicles based on on-road video where the camera was mounted on the vehicle itself.
9. YOLOv2 is used. Jun Sang et al. [15] proposed their vehicle detection research, which employed the k-means++ clustering technique to cluster the vehicle bounding boxes with six anchor boxes of various sizes. According to their findings, the average precision (mAP) might reach 94.78 percent.
10. The optimum dense YOLO approach proposed by Zhi Xu et al. [16] is used to detect vehicles in images captured by UAVs. This study is particularly useful for detecting tiny targets and is tailored to vehicle targets' features.
11. Shaobin Chen presented real-time detection for embedded systems in [17], using Yolo v3-live as the technique that minimized the complexity of embedded operating device computation.
12. This study suggests traffic object detection under bad weather conditions using YOLOv5. [18]

2.2 Research Objective

Our research's major goal is to familiarize Bangladeshi automobiles with smart traffic management and regulation. Some of Bangladesh's unique vehicles are not recognized by existing autonomous vehicles or traffic control systems. Bangladesh requires its own vehicle detection system that can identify all cars with unique characteristics. Our main research goals are:

1. To help Bangladesh establish a smart transportation system that would minimize traffic congestion and accident rates.
2. To detect Bangladeshi automobiles in difficult environments. Our main objective is to create a system that can recognize some of Bangladesh's distinctive vehicles.
3. We intended to use computer vision algorithms based on machine learning, such as YOLOv5 and Mobilenet-v5, for detecting various classes of vehicles.
4. To implement traffic object detection with deep neural network architecture.

Chapter 3

How YOLO Works

3.1 Introduction

YOLO made its computer vision debut in 2015 with a work published by Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection," garnered quick interest from other computer vision experts. Prior to the invention of YOLO, Convolutional Neural Networks (CNNs) such as Region Convolutional Network (R-CNN) used Region Proposal Networks (RPNs) to generate proposal bounding boxes on the input image, then run a classifier on the bounding boxes, and finally perform post-processing to remove duplicate detections and refine the bounding boxes. It was not suited for training the R-CNN network's various phases independently. Optimizing the R-CNN network was both challenging and time consuming. The author's objective is to develop a neural network model that encompasses all phases. After running the input picture through a single neural network composed of many convolutional networks, the system generates prediction vectors for each item present in the image. Rather of iteratively categorizing distinct parts on the picture, the YOLO system computes all of the image's attributes and provides predictions for all items simultaneously. This is the concept behind "You Only Look Once. [1]

3.2 Concept of YOLO

The primary concept behind YOLOv1 is to insert a grid cell with a size of $S \times S$ (7×7 by default) onto a picture. If an item's center is within a grid cell, that grid cell is responsible for detecting the object (Figure 3.1). As a result, all subsequent cells ignore the presence of the object displayed in numerous cells.

To execute object detection, each grid cell forecasts B bounding boxes together with their associated attributes and confidence ratings in Figure 1 [19] . These confidence scores indicate whether an object is present or absent within the enclosing box. The confidence score is defined as follows:

$$C = P_c * IOU$$

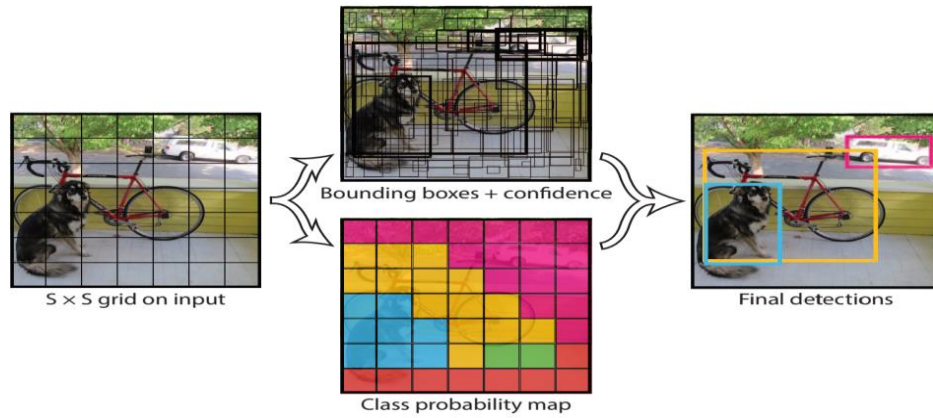


Figure 3.1: A 7x7 grid cell YOLO Model Applied on an Image.

Where P_c is the probability that an object exists inside the cell, and IOU is the intersection of the prediction box and the ground truth box. Because P_c between 0 and 1, the confidence score is close to zero if no object exists in that cell, and IOU is the intersection of the prediction box and the ground truth box. Because P_c between 0 and 1, the confidence score is close to zero if no object exists in that cell. Otherwise, the score is equal to the IOU. There are four more parameters for each bounding box, which correspond to the (center coordinates (b_x, b_y) , width, and height) of the bounding box (Figure 2). Incorporating the confidence score each bounding box has five parameters: a confidence score.

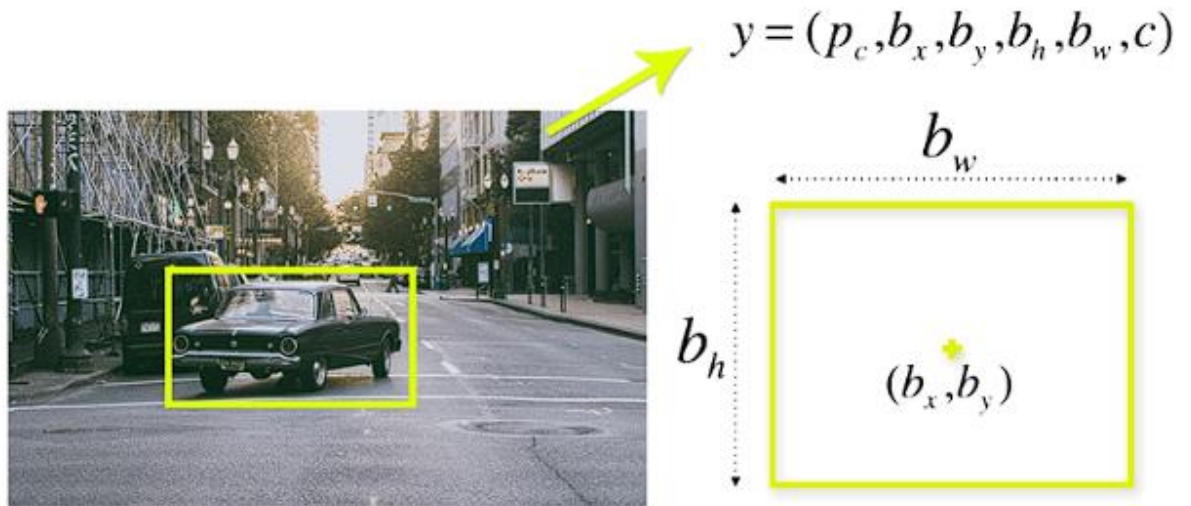


Figure 3.2: Parameters of a Bounding Box

3.2 Architecture of YOLOv1

When it comes to object bounding box prediction, there are two levels in the YOLO model that are entirely connected to each other (the authors referred to this as the "Darknet architecture") (Figure 3). For the assessment of this model, the $S = 7$, $B = 2$, and $C = 20$ values from the Pascal VOC dataset were used. This explains why the final feature maps are 7 by 7, as well as why the output size was 7 by 7 (2 by 5 Plus 20). According to the authors, the normal-YOLO model with 24 CNN layers in Darknet architecture can handle more complicated datasets and deliver more accuracy than the fast-YOLO model with 9 CNN levels (Figure 4). When GoogleNet was created, it advocated the usage of 1x1 and 3x3 convolutional layers in order to reduce the amount of feature space in the prior layers.

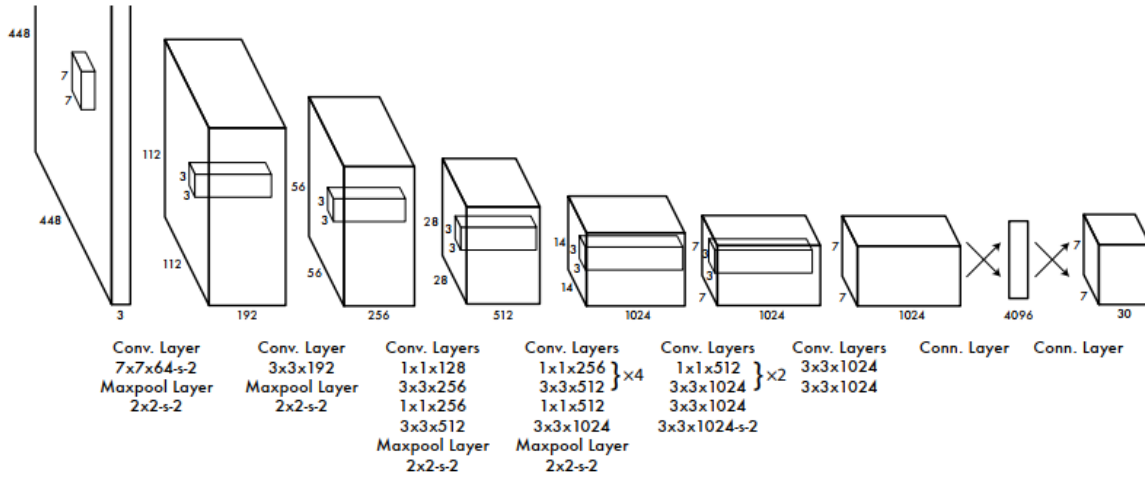


Figure 3.3: Architecture of YOLOv1

the activation function used in the last layer is swish/mish instead of leaky Relu. (Menegaz, 2018)

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases}$$

| Name | Filters | Output Dimension |
|------------|------------------------|-------------------|
| Conv 1 | 7 x 7 x 64, stride=2 | 224 x 224 x 64 |
| Max Pool 1 | 2 x 2, stride=2 | 112 x 112 x 64 |
| Conv 2 | 3 x 3 x 192 | 112 x 112 x 192 |
| Max Pool 2 | 2 x 2, stride=2 | 56 x 56 x 192 |
| Conv 3 | 1 x 1 x 128 | 56 x 56 x 128 |
| Conv 4 | 3 x 3 x 256 | 56 x 56 x 256 |
| Conv 5 | 1 x 1 x 256 | 56 x 56 x 256 |
| Conv 6 | 1 x 1 x 512 | 56 x 56 x 512 |
| Max Pool 3 | 2 x 2, stride=2 | 28 x 28 x 512 |
| Conv 7 | 1 x 1 x 256 | 28 x 28 x 256 |
| Conv 8 | 3 x 3 x 512 | 28 x 28 x 512 |
| Conv 9 | 1 x 1 x 256 | 28 x 28 x 256 |
| Conv 10 | 3 x 3 x 512 | 28 x 28 x 512 |
| Conv 11 | 1 x 1 x 256 | 28 x 28 x 256 |
| Conv 12 | 3 x 3 x 512 | 28 x 28 x 512 |
| Conv 13 | 1 x 1 x 256 | 28 x 28 x 256 |
| Conv 14 | 3 x 3 x 512 | 28 x 28 x 512 |
| Conv 15 | 1 x 1 x 512 | 28 x 28 x 512 |
| Conv 16 | 3 x 3 x 1024 | 28 x 28 x 1024 |
| Max Pool 4 | 2 x 2, stride=2 | 14 x 14 x 1024 |
| Conv 17 | 1 x 1 x 512 | 14 x 14 x 512 |
| Conv 18 | 3 x 3 x 1024 | 14 x 14 x 1024 |
| Conv 19 | 1 x 1 x 512 | 14 x 14 x 512 |
| Conv 20 | 3 x 3 x 1024 | 14 x 14 x 1024 |
| Conv 21 | 3 x 3 x 1024 | 14 x 14 x 1024 |
| Conv 22 | 3 x 3 x 1024, stride=2 | 7 x 7 x 1024 |
| Conv 23 | 3 x 3 x 1024 | 7 x 7 x 1024 |
| Conv 24 | 3 x 3 x 1024 | 7 x 7 x 1024 |
| FC 1 | - | 4096 |
| FC 2 | - | 7 x 7 x 30 (1470) |

Figure 3.4: Layers, Filters and Output Dimensions of YOLO

Now we'll look into the insights of IOU which is used in all YOLO models. [20]

- The amount to which two boxes overlap is referred to as the "IOU" (Intersection over Union). Bigger the area in which the IOU overlaps, the greater the amount.
- When it comes to object detection, IOU is mostly utilized to create boxes that fit exactly around an item once a model is trained to do so, a green box and a blue box are seen in the image below. The accurate box is shown in green, while the model's forecast is shown in blue. For this model, the ultimate goal is for the blue and green boxes to exactly overlap, such that the IOU between them is equal to one. To eliminate numerous boxes surrounding the same item based on which box has a higher degree of confidence, non-max suppression is a technique that is used in Intersection of Union.

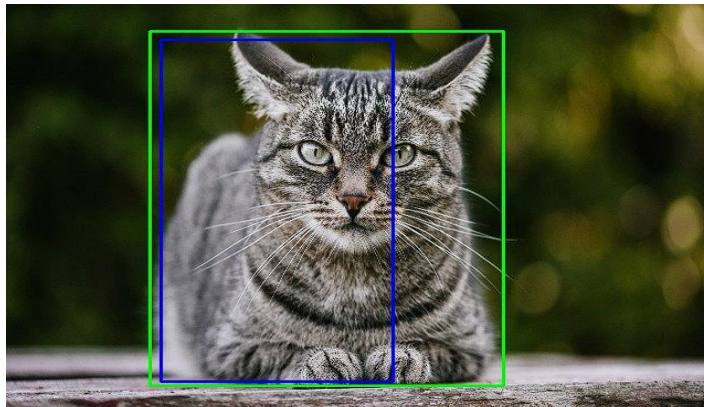


Figure 3.5: Accurate and Predicted Box From the Model

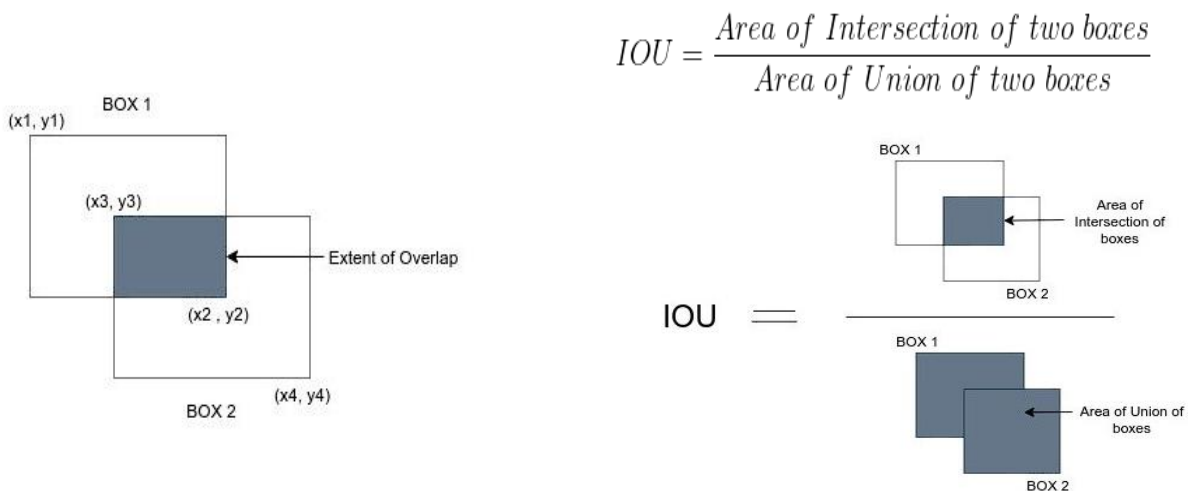


Figure 3.6: Calculating IOU

3.3 YOLOv5

The latest version YOLO is version 5 which encompasses total architecture of the previous version. We've used here the "small" version of it. There are a few versions and the difference among them is mainly the layers and connections. Now, we'll take a look at the architecture, activation function, cost function, weights and biases of this algorithm.

3.3.1 Architecture

Yolo version 5 is based on two concept two ideas of detection. The single-stage detector and the two-staged detector [21]. Figure 3.7 shows the basic ideas.

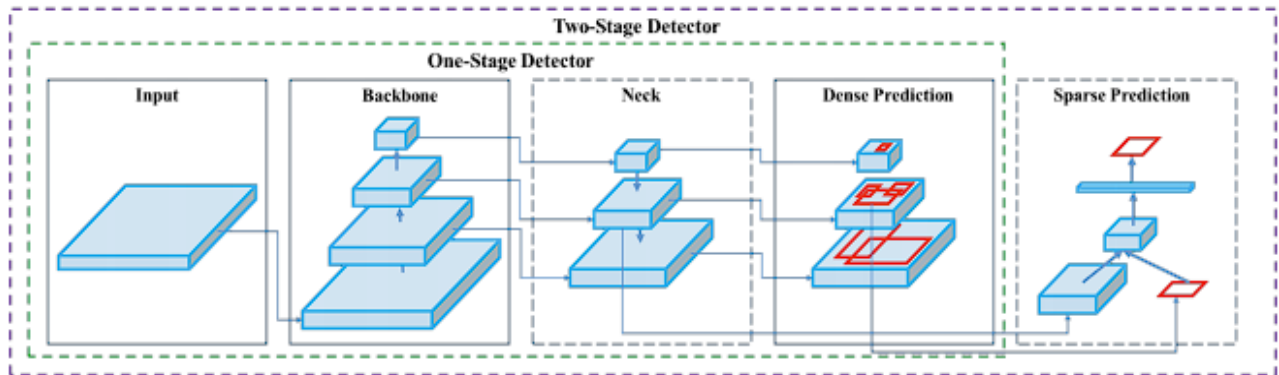


Figure 3.7: Double-staged Detector Concept

As shown in Figure 3.7, all object detection designs begin with a feature extractor (Backbone) compressing the input picture features before sending them on to the detector (which includes the Neck and Head components used for detection). The detection neck aggregates the characteristics produced by the backbone. Here, that head is responsible for detecting the bounding boxes, as well as localizing and classifying them. The one-stage detector (Dense Detection) does both jobs simultaneously, whereas the two-stage detector (Sparse Detection) does so sequentially and then combines the findings [22]. It's a one-step detector, so You Only Look Once.

So, there are 3 parts of in the whole architecture,

1. Backbone
2. Neck
3. Head

The authors of YOLOv4 have used different layers and activation function for better optimizing the algorithm. [23][24]

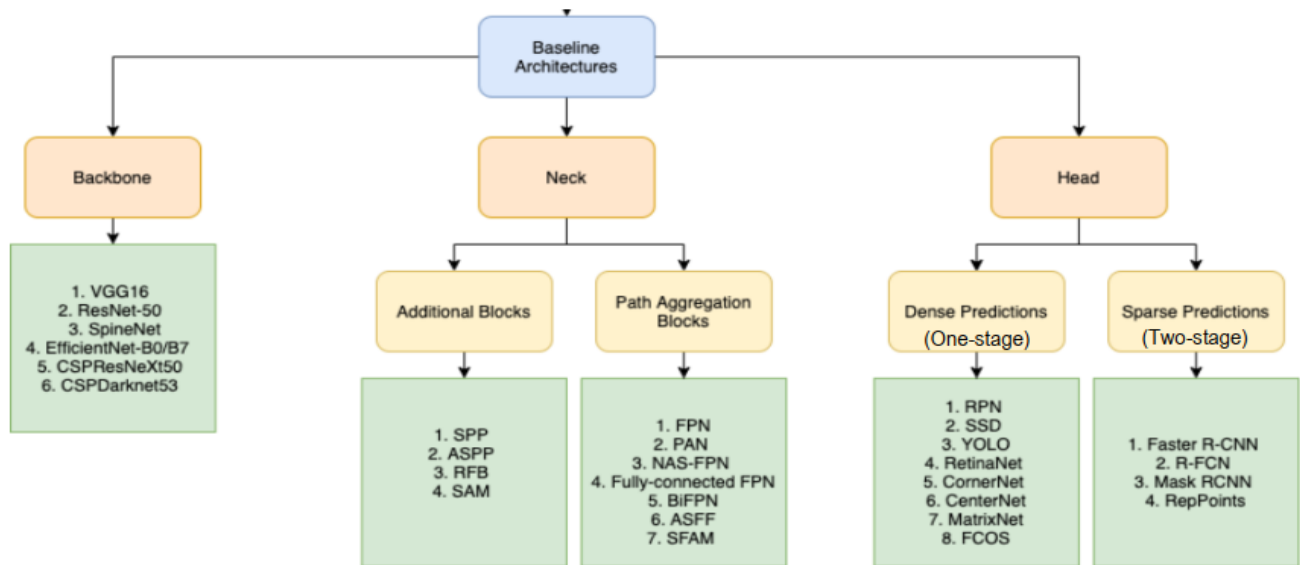


Figure 3.8: A Visual Representation of YOLOv4 Design.

Cross Stage Partial (CSP) architectures such as the CSPResNext50 and the CSPDarknet53 (where CSP stands for Cross Stage Partial) are both developed from the DenseNet design, which takes the prior and adds it to the already-existing input before proceeding to the dense layer. DenseNet was created to link layers in a very deep neural network with the goal of alleviating vanishing gradient difficulties (as ResNet). DenseNet was intended in making connections between neurons of a highly deep neural network in order to alleviate gradient difficulties. [25]

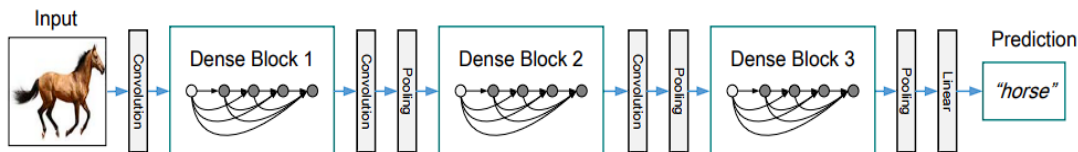


Figure 3.9: A Deep Network Consisting of Three Dense Blocks

In figure 3.9, a deep network of three dense blocks is depicted. The layers between two neighboring blocks are known as transition layers and vary in composition.

YOLOv4 uses path aggregation network in the neck layer which is an advanced version of FPN. FPN architecture's top-down flow means that only the large-scale detector in the lateral backbone of FPN can concurrently accept high-level semantic information and fine-grained characteristics from lower levels. For the time being, FPN's small-scale detector relies solely on semantic traits

to identify things. Semantic features and fine-grained characteristics might be concatenated at high-level layers to make the small and medium-sized detector work better.

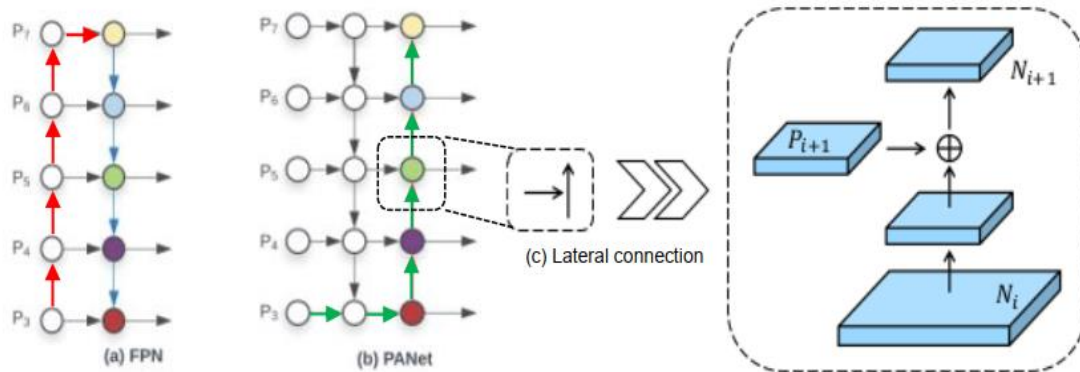


Figure 3.10: (a) FPN (b) PANet (c) Bottom-up Augmentation Path Connection

The block diagram of YOLOv5 looks like this,

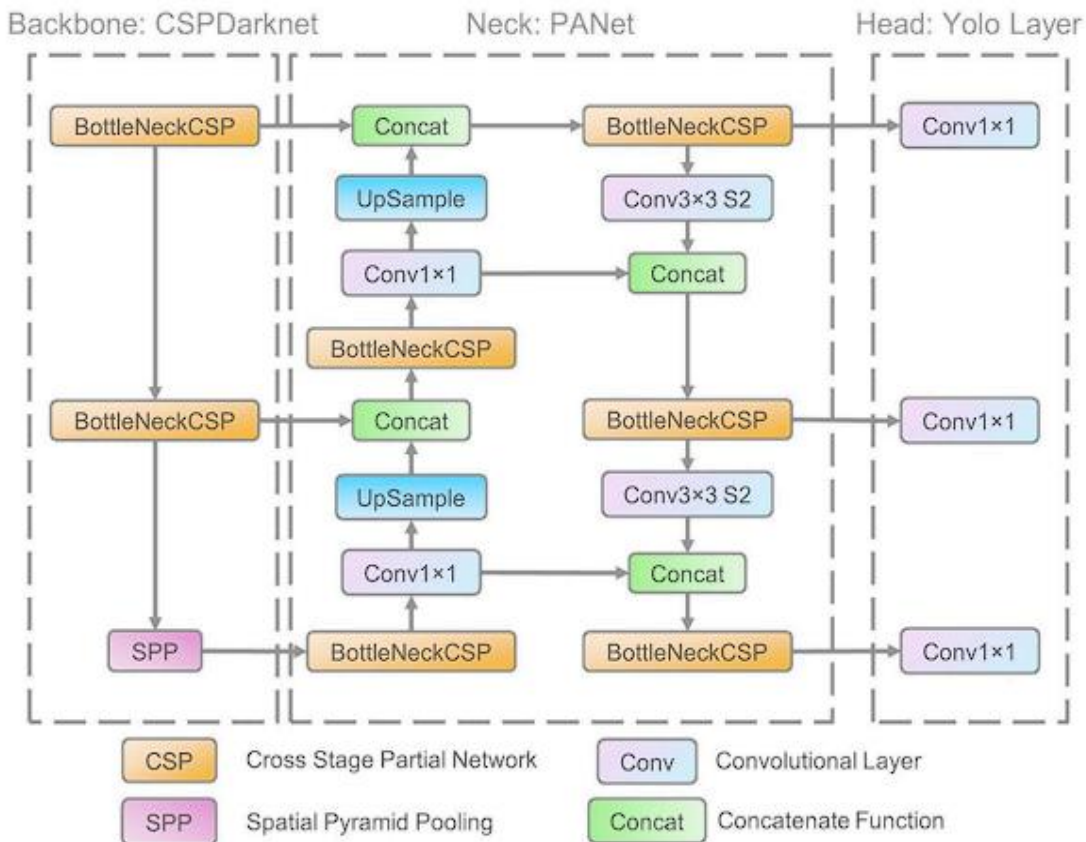


Figure 3.11: YOLOv5 Architecture Using Block Diagram

3.3.2 Activation Function

The selection of activation functions is crucial in the development of any deep neural network. Recently, a slew of activation functions, such as Leaky ReLU, mish, swish, and others, have been added. The authors of YOLOv5 selected the Leaky ReLU and Sigmoid activation functions as the best fit for their research. When it comes to the middle/hidden layers of YOLOv5, the Leaky ReLU activation function is used, as is the sigmoid activation function when it comes to the final detection layer.

3.3.3 Optimization Function

In YOLO v5, we have two alternatives for optimizing the function:

1. SGD
2. Adam

For training in YOLO v5, the SGD optimization algorithm is the default. However, the command-line parameter "`— — adam`" can be used to change the default one. [26]

3.3.4 Cost Function

YOLO's compound loss is formed from the objectness score, class probability, and bounding box regression score, and it belongs to the YOLO family. Class probability and object score loss have been calculated using PyTorch's Binary Cross-Entropy with Logits Loss function. We also have the option of calculating the loss using the Focal Loss function, which is described below. In order to train using Focal Loss, we may make use of the `fl-gamma` hyper parameter.

3.3.5 Detection in YOLO

Suppose we want to train 3 types of object with YOLO: pedestrian, cars, motorcycles. As we have three background class we need to define three labels. For example, we are using two anchor boxes to hold things in place (these anchor boxes are presented in the figure 3.12). This means that our

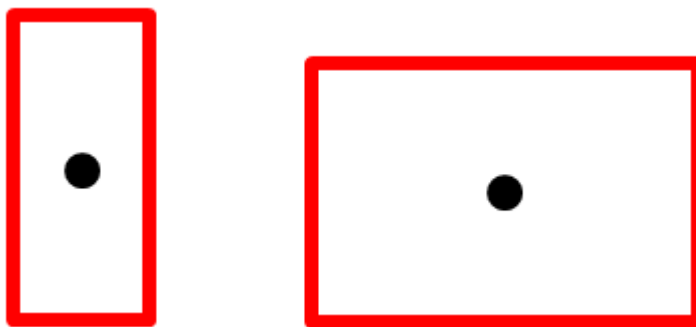


Figure 3.13: Anchor boxes 1 and 2.

While it's not ideal, our car will be placed to the anchor box 2 because it's taller and wider than our image suggests. An easy technique to determine which anchor box has the maximum IoU with an ideal object box is to just check that one. We don't need to worry about the other values as pc associate anchor box 1 is equal to 0. We assigned $P_c=1$ to the anchor box 2 because we found a vehicle equipped with it. As a result, we'll produce a 16-dimensional vector for each of our nine grid places (3x3 grid positions).

3.3.6 Bag of Freebies

With the goal of increasing the accuracy and performance of the YOLOv4 algorithm, the authors have experimented with and implemented a variety of optimization strategies. To describe these enhancements, the writers use the terms "Bag of Freebies" and "Bag of Specials," respectively. [24]

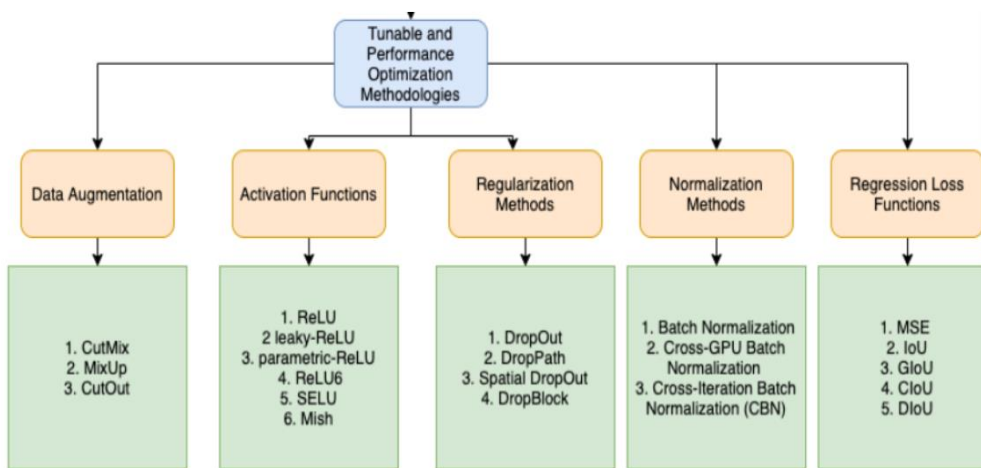


Figure 3.14: Multiple Optimization Methods That Were Experimented.

An improvement approach is referred to by the term "Bag," and the term "Freebies" indicates that this improvement method can help improve the model's performance and accuracy without costing any hardware. As a result, the architecture will be able to take advantage of extra performance without having to pay for it. After conducting many tests and analyzing the data, the authors of YOLOv4 identified and implemented the following Bag of Freebies improvement approaches and applied them to YOLOv4.

1. **For Backbone:** CutMix and Mosaic data augmentations, DropBlock regularization, Class label smoothing. [24]
2. **For detector:** Complete Intersection over Union (CIoU-loss), Cross-mini-Batch Norm (CmBN), DropBlock regularization, Mosaic data augmentation, Self-Adversarial Training (SAT), Eliminate grid sensitivity, using multiple anchors for a single ground truth, Cosine annealing scheduler, Optimal hyper-parameters, Random training shapes. [24]

3.3.7 Bag of Specials

After the Bag of Freebies, the authors came up with another way to make things better, called the Bag of Specials. It means getting something of value for a discount or cheap. There are advanced optimization strategies that demand a minimal amount of architecture expense in order to considerably improve the performance accuracy of object detection in the Bag of Specials collection. [24]

1. **For backbone:** Mish activation, Cross-stage partial connections (CSP), Multi-input weighted residual connections (MiWRC). [24]
2. **For detector (neck and head):** Mish activation, Spatial Pyramid Pooling block (SPP-block), Spatial Attention Module (SAM-block), Path Aggregation Network (PANet), Distance Intersection over Union Non-Maximum Suppression (DIoU-NMS).

Chapter 4

How MobileNet Works

4.1 Introduction

It is the first mobile computer vision model developed by TensorFlow, and it is named MobileNet since it is designed to be used in mobile applications, as the name indicates. In the previous version of MobileNetV1, Depth wise Separable Convolution was introduced, which significantly lowered the network's complexity cost and model size, making it suitable for mobile devices or other low-power devices, as well as for high-performance computing. It has been introduced to MobileNetV2 a new upgraded module with an inverted residual structure. Non-linearities in thin layers have been eliminated this time around. With MobileNetV2 as the foundation for feature extraction, it is also feasible to perform state-of-the-art object recognition and semantic segmentation techniques. Figure 4.1 illustrates how image processing is carried out using MobileNetV2 software. The changes between MobileNetV1 and MobileNetV2 are seen in Figure 4.2.[27] [2]

When contrasted to a network with traditional convolutions of same depth in the nets, it results in a significant reduction in the number of parameters. As a result, deep neural networks that are lightweight are built.

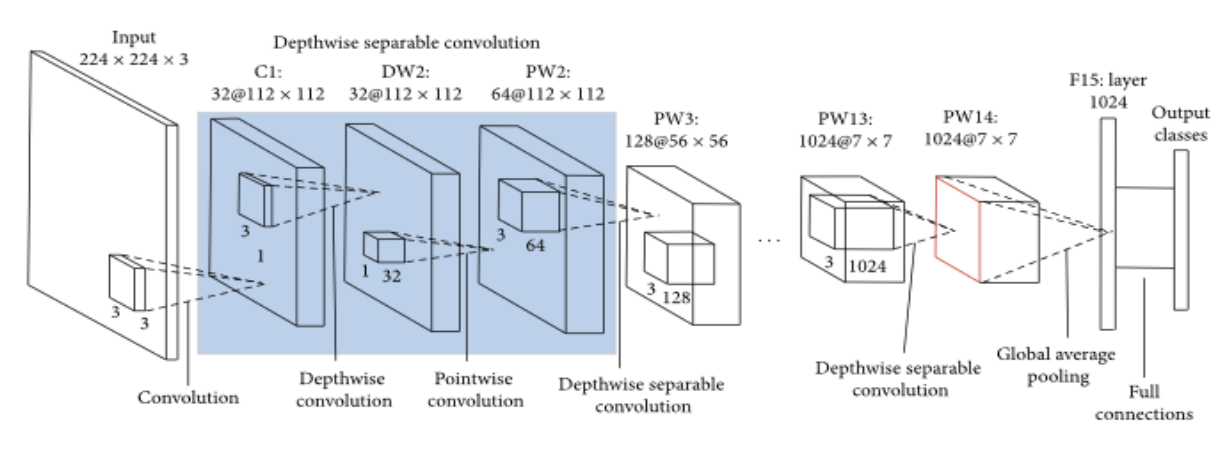


Figure 4.1: Image Processing by MobileNet Architecture

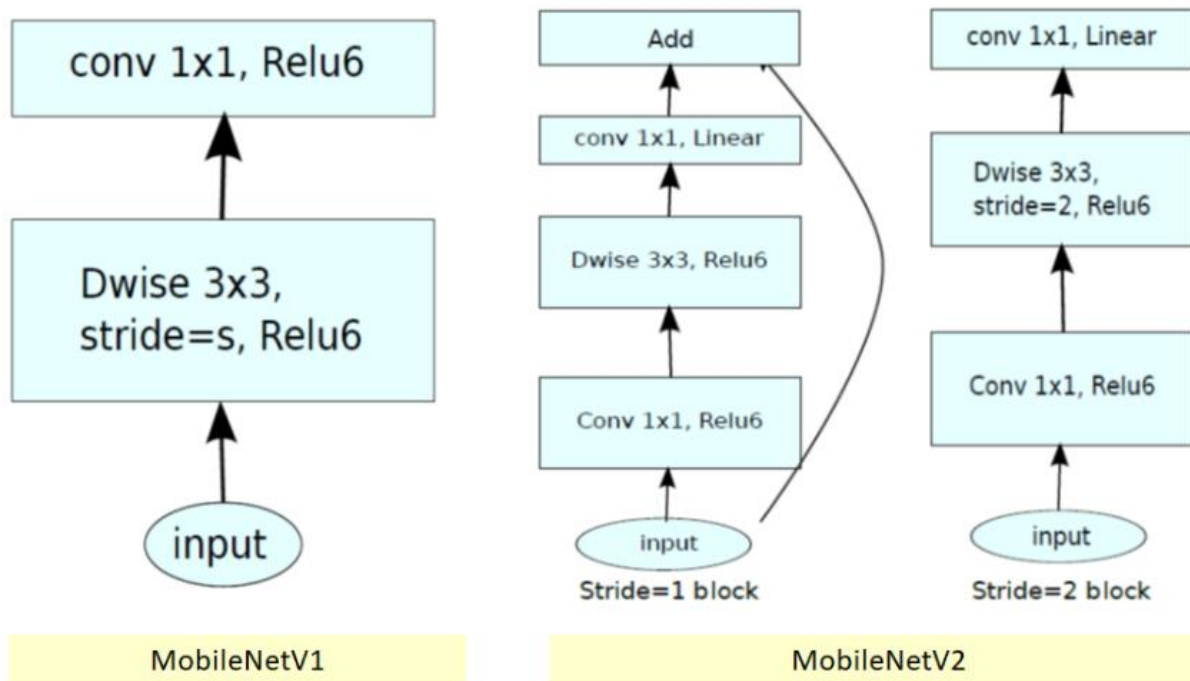


Figure 4.2: MobileNetV1 and MobilenetV2 comparison [2]

4.2 Convolutional Blocks for MobileNetv2

- A DWSC is made from two methods: Depth-wise convolution and Pointwise convolution.
- In MobileNetV2, there are two types of network blocks. A one-stride residual network block is one of them. A two-stride block is appropriate for downsizing.
- Both kinds of blocks have three layers each.
- The first layer is 1*1 convolution using Rectified Linear activation of type 6 this time.
- The depth wise convolution is the second layer.
- The third layer is a 1X1 network layer convolution with no non-linearity. If ReLU is applied again, it is suggested that Deep networks will only have the power of a linear classifier in the non-zero volume output domain, which is the case for most applications.
- There is also a t expansion factor. For all major experiments, t=6.
- If there are 64 channels in the input, the internal output would become $64 \times t = 64 \times 6 = 384$ network channels.

4.3 Overall Architecture

| Input | Operator | t | c | n | s |
|--------------------------|-------------|-----|------|-----|-----|
| $224^2 \times 3$ | conv2d | - | 32 | 1 | 2 |
| $112^2 \times 32$ | bottleneck | 1 | 16 | 1 | 1 |
| $112^2 \times 16$ | bottleneck | 6 | 24 | 2 | 2 |
| $56^2 \times 24$ | bottleneck | 6 | 32 | 3 | 2 |
| $28^2 \times 32$ | bottleneck | 6 | 64 | 4 | 2 |
| $14^2 \times 64$ | bottleneck | 6 | 96 | 3 | 1 |
| $14^2 \times 96$ | bottleneck | 6 | 160 | 3 | 2 |
| $7^2 \times 160$ | bottleneck | 6 | 320 | 1 | 1 |
| $7^2 \times 320$ | conv2d 1x1 | - | 1280 | 1 | 1 |
| $7^2 \times 1280$ | avgpool 7x7 | - | - | 1 | - |
| $1 \times 1 \times 1280$ | conv2d 1x1 | - | k | - | - |

Figure 4.3: Overall Architecture of MobileNetV2

- For spatial convolution, 3X3 kernels are employed. where t : expansion factor, c : number of output channels, n : repeating number, s : stride.
- The core network (width multiplier 1, 224X224) employs 3.4 million values and has a computational complexity of 300 million multiply-adds.
- For input resolutions ranging from 96 to 224 and width multipliers ranging from 0.35 to 1.4, tradeoffs are further investigated. The network computation took up to 585 million MAdds, while the model size ranged from 1.7 million to 6.9 million parameters.
- The network is trained on 16 GPUs with a batch size of 96.

4.4 Depthwise Separable Convolution

When a filter can be separated into its depth and spatial dimensions, the term separable is used. Take the Sobel filter, for example (figure 4.4), which is used in image processing to identify edges

| | | |
|---|---|---|
| 1 | 2 | 1 |
| 0 | 0 | 0 |
| 2 | 1 | 1 |

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 0 | 0 | 0 |
| 2 | 2 | 1 |

Figure 4.4: Fx(left), Fy (right)

Fx is used to identify the vertical edge, whereas Fy is used to detect the horizontal edge. These filters may be distinguished in terms of their height and width. Gx is the matrix product of $[1 \ 1 \ 1]$ and $[1 \ 2 \ 1]$. The filter had changed its appearance. It shows nine parameters, but only six are available. This was possible since the measurements of height and breadth were separated. The same principle may be used to separate the depth dimension from the horizontal (width*height) dimension, resulting in DWSC. The depth dimension is then covered using a $1*1$ matrix.

One thing to note is how much this convolution reduces the number of parameters while maintaining the same number of channels. $3*3*3$ parameters for depth-wise convolution and $1*3$ parameters for additional convolution in the depth dimension are required to create one channel.

However, if three output channels are required, we only need 31×3 depth matrix, totaling $36 (= 27 + 9)$ parameters, while standard convolution requires $33 \times 3 \times 3$ filters, totaling 81 parameters.

DWSC (figure 4.5) is followed by a pointwise convolution as given below:

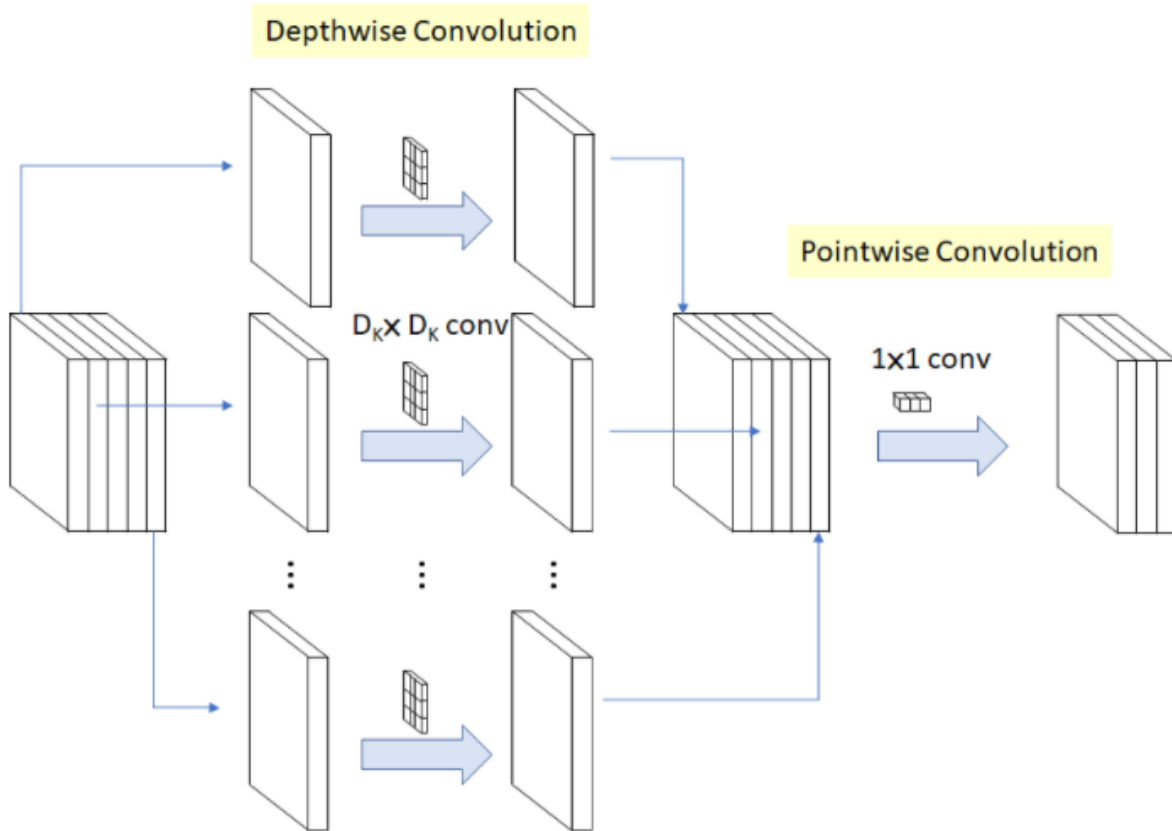


Figure 4.5: Depthwise Separable Convolution

The channel-wise $D_k \times D_k$ spatial convolution is known as depthwise convolution. If, as shown in figure 4.5, we have five channels, we may expect to have five $D_k \times D_k$ spatial convolutions as a result. One convolution may have different maps for each of the input channels. That's why there are exactly as many input and output ports as there are devices connected to them. It costs $D_f^2 \times M \times D_k^2$ to compute. [28]

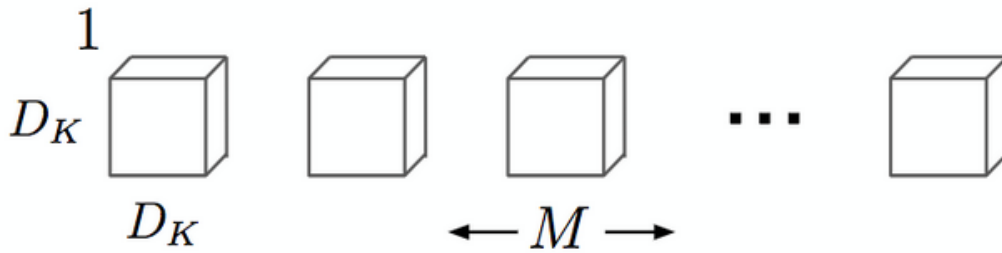


Figure 4.6: Depthwise Convolution

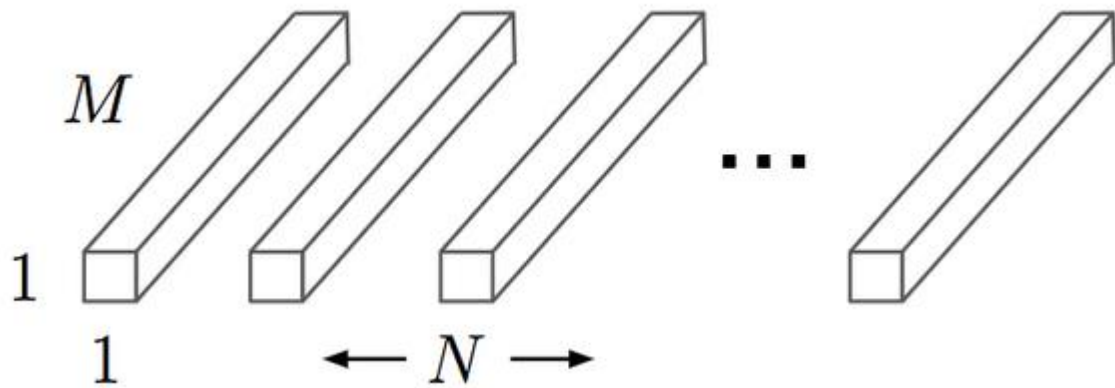


Figure 4.7: Pointwise Convolution

Convolution in the form of pointwise convolution is a one-to-one transformation. It is a convolution with a kernel size of 1×1 that simply mixes the depthwise convolution's characteristics. It costs $M * N * D_f^2$ to compute.

The batch norm and the ReLU are utilized instead of a single 3×3 convolution layer in MobileNet's design. Using a 3×3 depth-wise convolution and a 1×1 pointwise convolution, Mobile Nets separate the convolution.

Chapter 5

Research Methodology

5.1 Dataset

The dataset for training was provided by Dhaka-AI [39], named “Dhaka Traffic Detection Challenge Dataset” [40] [41]. The Dataset is composed of the most common 21 different classes of vehicles of Dhaka city. There are total 3002 images in the dataset and 24,368 annotations. Figure 5.1 shows a general summary of the dataset.



Figure 5.1: Summary of the Dataset

List of the class names and their number of appearance in the dataset:

- Ambulance - 70
- Auto-rickshaw - 43
- Bicycle - 459
- Bus - 3340
- Car - 5476
- Garbage van - 3
- Human hauler - 169
- Minibus - 95
- Minivan - 935
- Motorbike - 2284
- Pickup - 1225

- Army vehicle - 43
- Police car - 32
- Rickshaw - 3549
- Scooter - 38
- SUV - 860
- Taxi - 60
- CNG - 2990
- Truck - 1492
- Van - 756
- Wheelbarrow - 120

The dataset is an imbalanced dataset. Car, rickshaw, bus and CNG are overrepresented where except only motorbike, truck, pickup and minivan, the rest of the vehicles are underrepresented. Figure 5.2 shows the class balance report generated by Roboflow . As the data set was unbalanced, different methods can be used like Cutmix, mixup, rotation etc. to better train the model. But as the training time would increase significantly with low performance workstation that we have we couldn't do so at the beginning. Afterwards we tried some data augmentation techniques but the results didn't improve that much. Figure 5.3 and 5.4 show a typical image for each of the classes of vehicles. Figure 5.5 shows the image size distribution of the dataset.

| Class Balance | | |
|----------------------|-------|-------------------|
| car | 5,476 | over represented |
| rickshaw | 3,549 | over represented |
| bus | 3,340 | over represented |
| three wheelers (C... | 2,990 | over represented |
| motorbike | 2,284 | over represented |
| truck | 1,492 | |
| pickup | 1,225 | |
| minivan | 935 | |
| suv | 860 | under represented |
| van | 756 | under represented |
| bicycle | 459 | under represented |
| auto rickshaw | 372 | under represented |
| human hauler | 169 | under represented |
| wheelbarrow | 120 | under represented |
| minibus | 95 | under represented |
| ambulance | 70 | under represented |
| taxi | 60 | under represented |
| army vehicle | 43 | under represented |
| scooter | 38 | under represented |
| policecar | 32 | under represented |
| garbagevan | 3 | under represented |

Figure 5.2: Class Balance of the Dataset



Figure 5.3: Different Classes of Vehicles in Dhaka City (1)



Figure 5.4: Different Classes of Vehicles in Dhaka City (2)

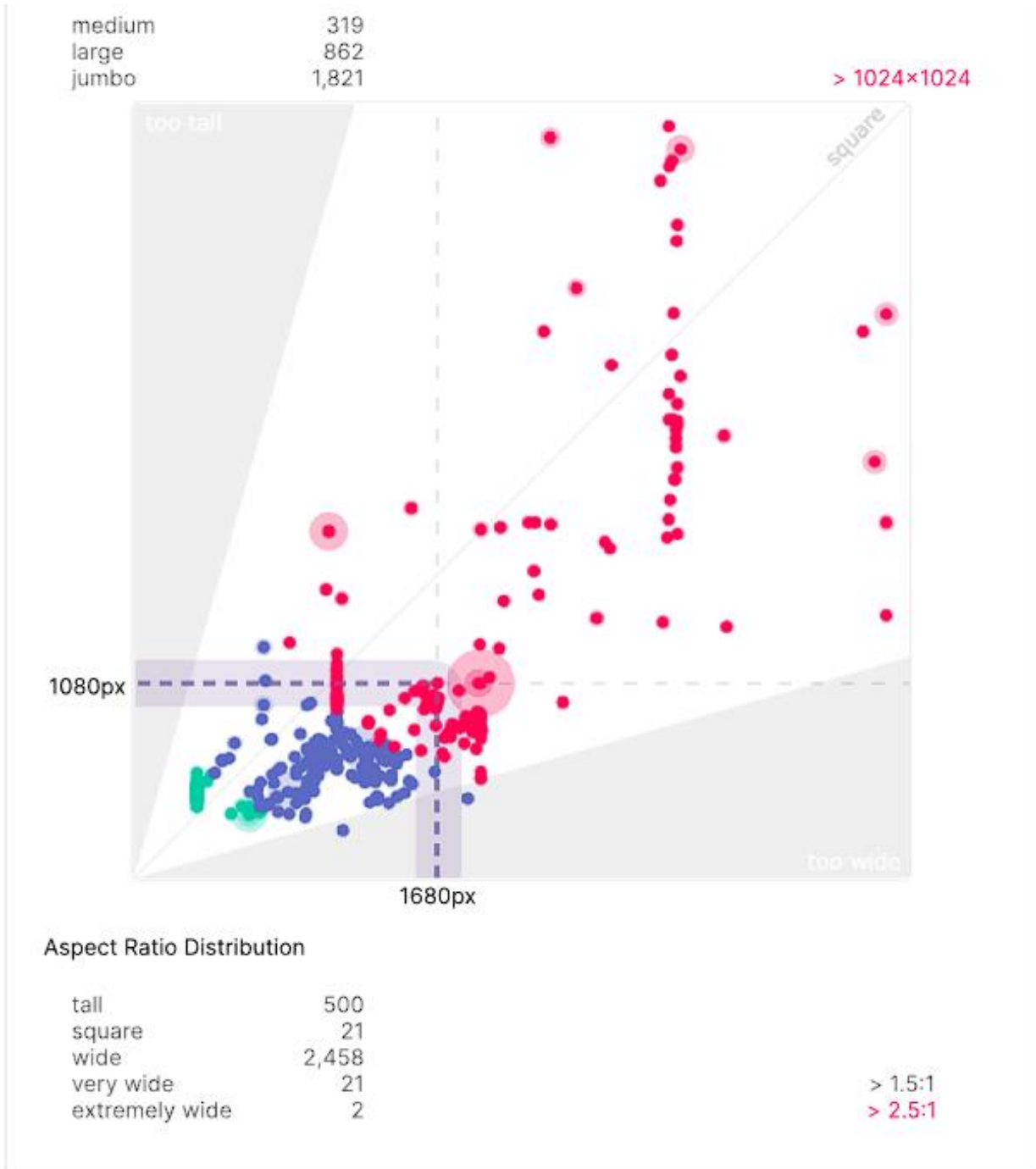


Figure 5.5: Image Size Distribution of the Dataset

5.2 Training Parameters

5.2.1 MobileNetV2

1. Batch Size: 8
2. Number of Steps: 10,000
3. Warm-up Steps: 2000
4. Number of classes: 21

5.2.2 YOLOv5

1. Batch Size: 16
2. Epochs: 2000
3. Number of classes: 21

5.3 Requirements

5.3.1 MobileNetV2

1. Windows or Linux
2. Python ≥ 3.8
3. TensorFlow ≥ 2.5
4. Anaconda ≥ 3.8
5. Nvidia GPU GTX 650 or newer
6. CUDA Toolkit v11.2
7. CuDNN 8.1.0
8. Object Detection API
9. Microsoft Visual Studio

5.3.2 YOLOv5

1. Windows or Linux
2. Python ≥ 3.9
3. Tensorflow $\geq 2.4.1$
4. matplotlib $\geq 3.2.2$
5. opencv-python $\geq 4.1.1$

6. Pillow \geq 7.1.2
7. PyYAML \geq 5.3
8. scikit-learn \leq 0.19.2

5.4 Installation

5.4.1 MobileNetV2

We used local machine for training the dataset using MobileNetV2. The following steps were done sequentially in the training process:

1. A new environment in Anaconda was created.
2. Inside the environment, first we installed TensorFlow 2.5.
3. CUDA Toolkit and CUDNN was installed and added to path.
4. From TensorFlow Models repository, the Object Detection API was installed.
5. Downloaded the pre-trained MobileNetV2 FPNlite 640x640 from TensorFlow Model Zoo.
6. Appropriate Label Map and TFrecords were generated for the dataset.

5.4.2 YOLOv5

As we used Google colab platform for training with YOLO we did not need any installations in the local machine. All the installations were done in colab. They are as following,

- Roboflow for training the model
- Pytorch
- OS

We used Roboflow developed by Ultralytics Corporation. It has all the dependencies installed within it like Numpy, Open CV, DarkNet repository. We only had to choose the model we wanted to train and prepare the dataset accordingly.

5.5 Platform and System Specifications

5.5.1 Local Machine

MobileNetV2 model was trained on a desktop pc. The specifications of the system were:

- CPU: AMD Ryzen-5 3600 6 Core Processor @ 3.60 GHz

- GPU: GIGABYTE GamingX NVIDIA GTX 1060 6 GB VRAM
- RAM: 16 GB DDR4

5.5.2 Cloud

The YOLOv5 model was trained on Google Colaboratory Pro, which is a paid cloud platform. The specifications of the platform were:

- CPU: Intel Xeon(R) 2.30 GHz
- GPU: Tesla P100-PCIE-16GB VRAM
- RAM: 12.8 GB DDR4

5.6 Training

5.6.1 MobileNetV2

The MobileNetV2 model was trained on local machine for ten thousand steps for an approximate training time of 28 hours. The model was trained using a GTX 1060 6 GB gpu. The loss vs steps curve is shown in figure 5.6.

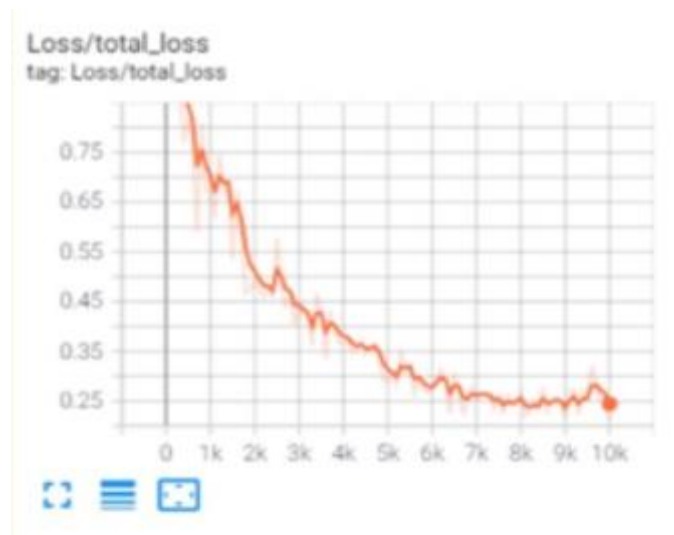


Figure 5.6: Loss vs Step Curve of MobileNetV2

5.6.2 YOLOv5

YOLOv5 was trained using 1000 epochs which took approximately 12 hours to train. Batch size was 16. The learning rate saturated after 700 epochs. Figure 5.7 depicts the relation between performance matrix and epochs of the model. Figure 5.8 depicts the confusion matrix for YOLOv5 and figure 5.9 shows the label instances of the particular model.

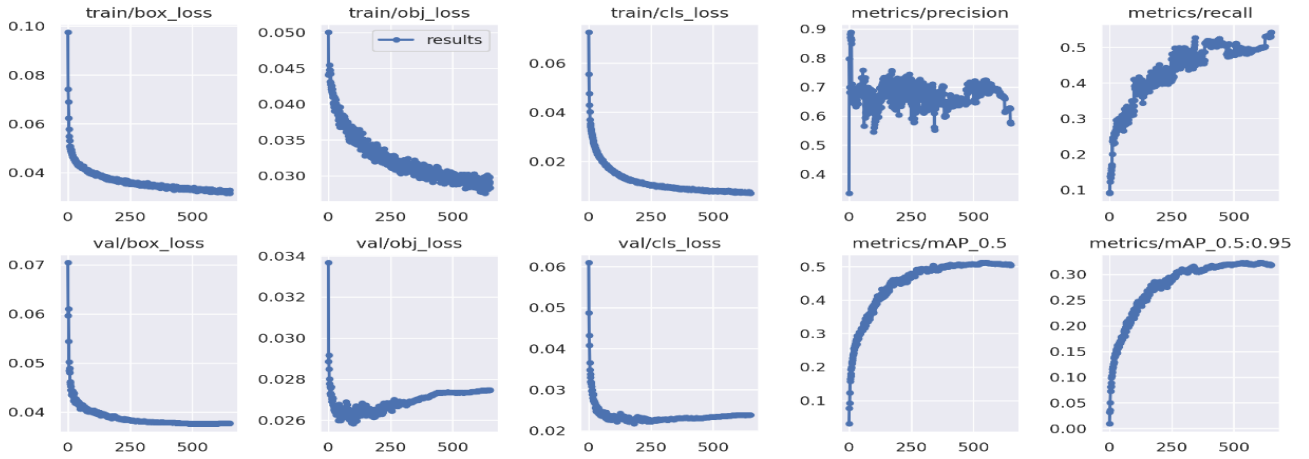


Figure 5.7: Performance Matrix vs Epochs

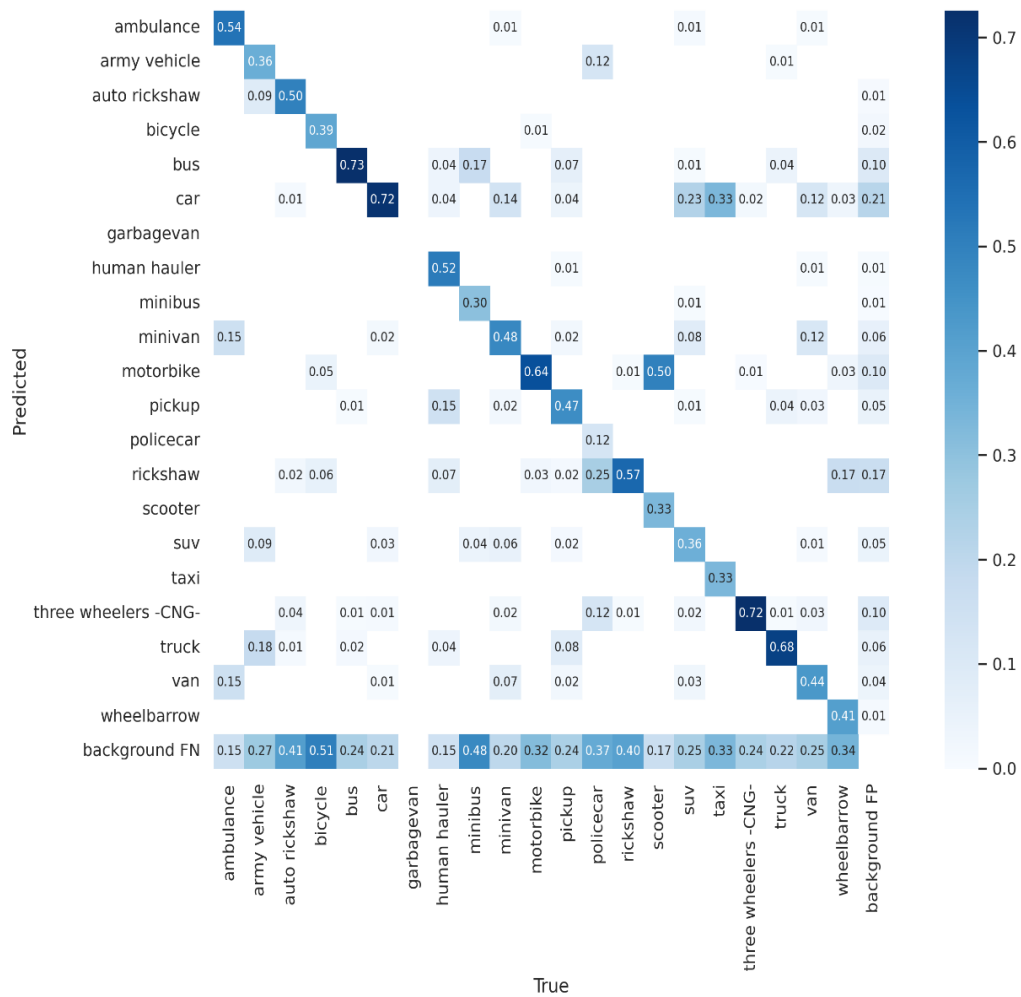


Figure 5.8: Confusion Matrix of YOLOv5

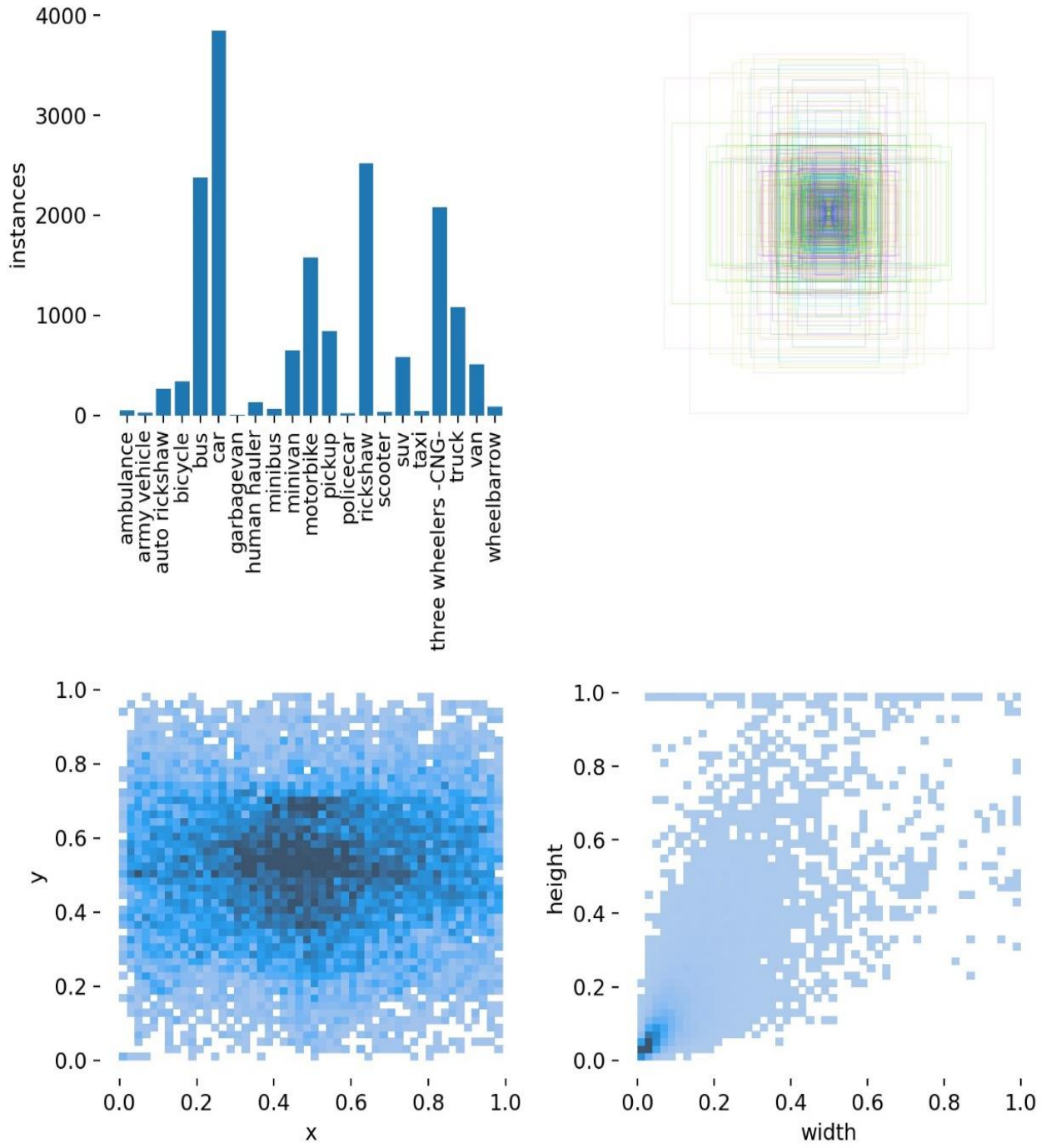


Figure 5.9: Label Instances

5.7 Research Timeline

We methodically split our research activity into four parts to better assist our development and guarantee that we followed a strict plan that would keep us on track. We chose to adopt a more systematic approach to our learning and work rather than rushing to the finish line and stressing about the ultimate product. For each step, we identified specific targets that we wanted to achieve and were flexible with time so that we could do our tasks properly rather than half-heartedly.

We all shared a single goal: to learn and explore. We wanted to establish a good foundation in anything we did because machine learning and deep learning were unfamiliar to us. As we embarked on this journey, we concentrated on the theoretical parts of learning and putting what we had learned into practice. We worked on a variety of initiatives to help us learn more effectively, which finally led to our dissertation work. Figure 5.10 depicts a summary of our research timeline.

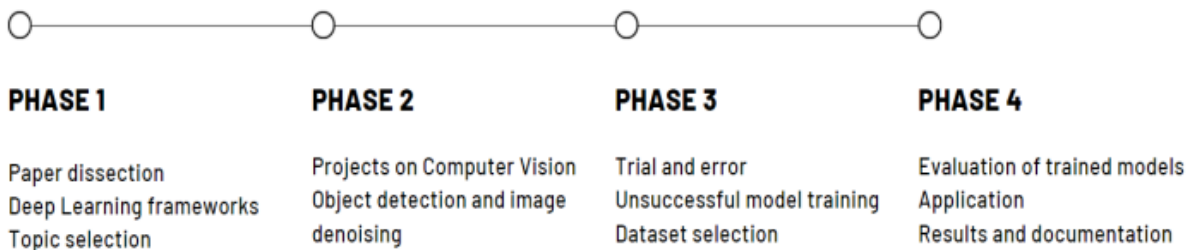


Figure 5.10: Research Progression Timeline

5.7.1 Phase 1

The first stage took the most time. It began after our third-year final examinations were completed. To learn about Python programming, machine learning, and eventually deep learning, we began taking online classes, watching YouTube videos, and participating in different online courses. We also began reading research papers on various machine and deep learning approaches and strategies. The entire process took around four months and provided us with adequate information to choose a dissertation subject. We selected Computer Vision because we were intrigued by the thought that robots could perceive and recognize objects in the same way that people could. This is what we wanted to investigate and expand on.

5.7.2 Phase 2

The second step was significantly more difficult and demanding. This was the time that we studied Computer Vision in depth, both theoretically and practically. We learnt both traditional and cutting-edge computer vision techniques. We looked for projects that sparked our attention on Github [29] and Stack Overflow [30]. We worked on a variety of modest projects, such as a real-time attendance system and picture denoising, to improve our understanding of this field of study and generate ideas for our own thesis. This stage lasted around three months.

5.7.3 Phase 3

We spent hours during this step attempting to come to an agreement on the subtopic and dataset we should use for our thesis work. We looked through Kaggle [31] and UCI Datasets for an enjoyable topic to work on. While there were many interesting things to choose from, we always felt compelled to concentrate on something that was relevant to our own lives. We found that there were no datasets particular to Bangladesh. As a result, we decided to make our own dataset and contribute to the Deep Learning community.

Fortunately for us, we came across the "Dhaka-AI" competition, which was an AI-based competition held in Bangladesh using a Dhaka-specific dataset. As a result, we used the Dhaka traffic dataset to develop a deep learning-based real-time vehicle recognition system. We tried a variety of methods and spent a lot of time experimenting. Finally, we chose MobileNetV2 and YOLOv5 framework as the foundation for our thesis work. This stage lasted around three months.

5.7.4 Phase 4

After we'd chosen our topic, dataset, and method of action, all that was left for us to do was put our knowledge into practice and bring our thesis work to life. We spent the following four months improving and training our models to achieve the best potential outcomes.

We documented all of our discoveries and were prepared for our defense as we approached the end of the process. To improve our outcomes, we adjusted our models and explored other

strategies. We wanted to make certain that no stone was left unturned and that we could confidently and effectively defend our thesis before the thesis committee.

We were able to overcome all barriers and deliver our thesis thanks to the blessing of Almighty Allah, the direction and assistance of our supervisor, Professor Dr. Golam Sarowar sir, and the love and support of our parents.

Chapter 6

Results

6.1 MobileNetV2

For MobileNetV2 we got the mean average precision or mAP to be around 0.26. The value is fairly acceptable since the COCO API metric of MobileNetV2 is 0.28. The precision value is can be improved by enhancing the dataset and by making it more balanced in terms of class representation. Figure 6.1 shows the the mAP of detection boxes for our MobileNetV2 model.

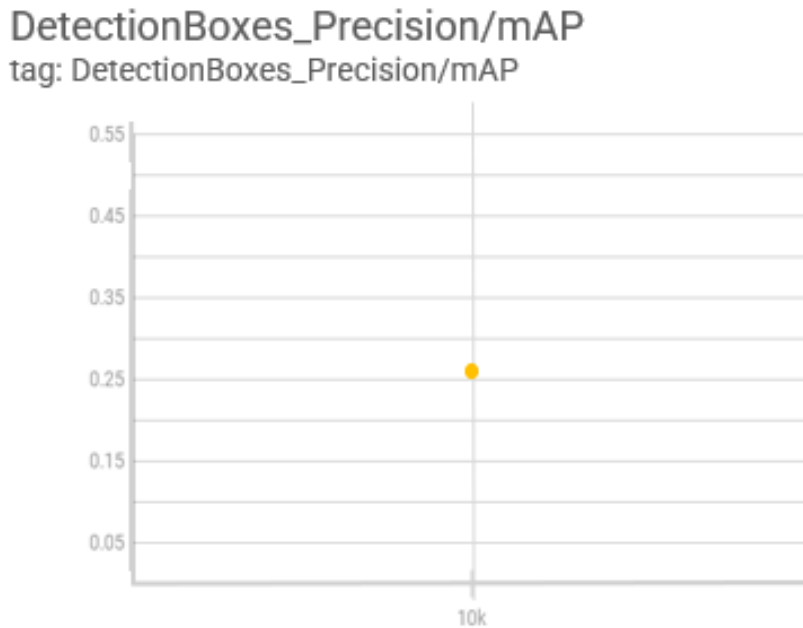


Figure 6.1: Detection Box mAP of MobileNetV2

6.1.1 Inference on Images

We ran inference on a few images randomly to test our models working capacity. The results are shown in figure 6.2, 6.3, 6.4 and 6.5.

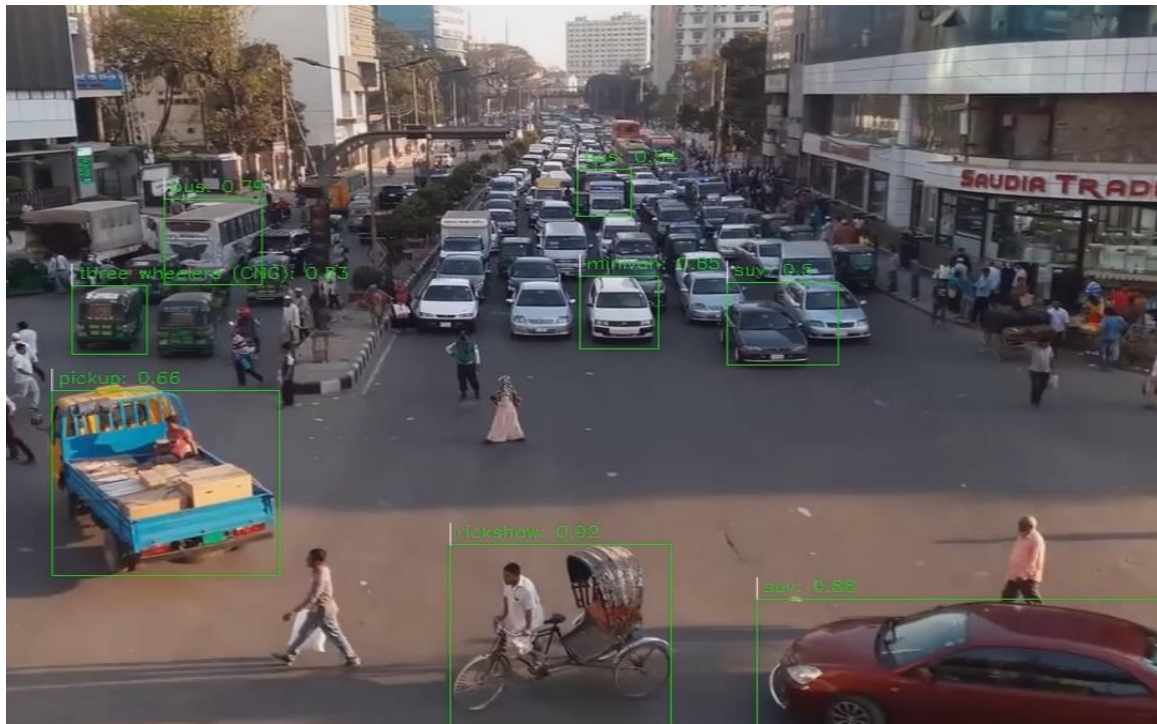


Figure 6.2: MobileNetV2 Image Inference 1

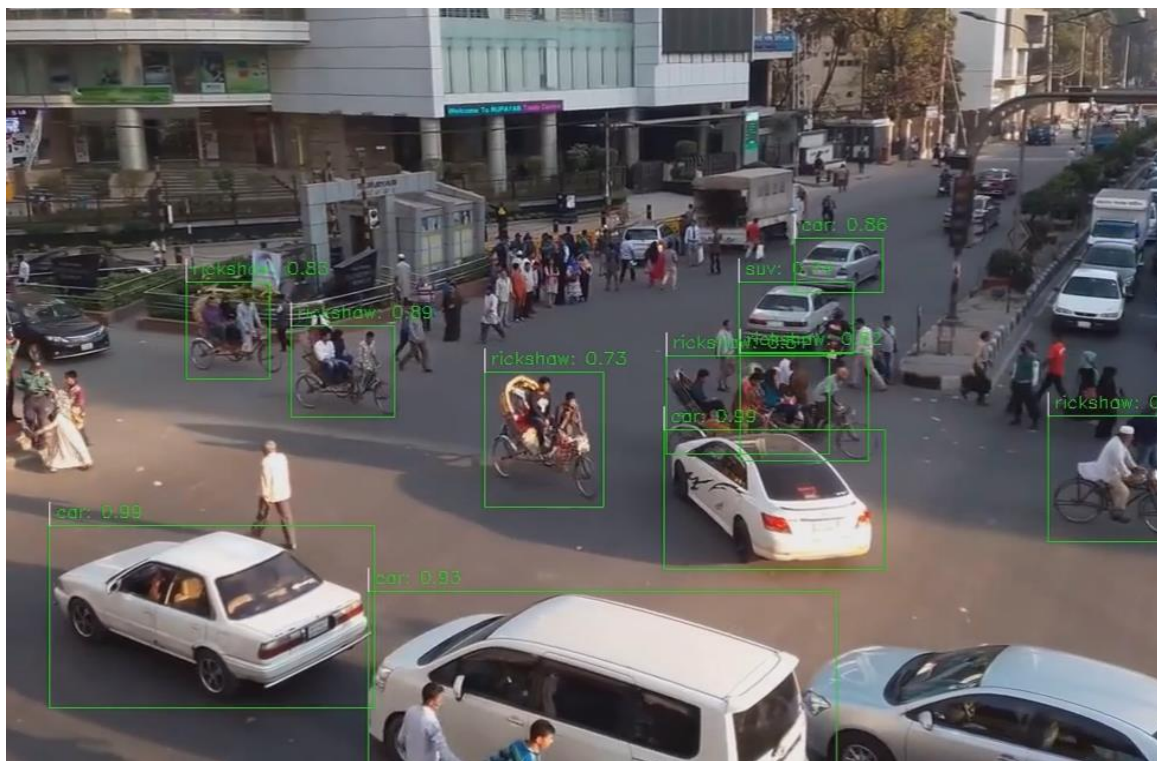


Figure 6.3: MobileNetV2 Image Inference 2



Figure 6.4: MobileNetV2 Image Inference 3



Figure 6.5: MobileNetV2 Image Inference 4

6.1.2 Inference on Video

The inference video was created by running inference on a real time traffic video using the MobileNetV2 exported model. A QR code containing the video is depicted in figure 6.6.



Figure 6.6: QR Code of MobileNetV2 Video Inference

6.2 YOLOv5

For YOLOv5s the mean average precision or mAP was found to be .51 approximately. The mAP curve is shown in figure 6.7. As noticeable in the figure, the mAP increases initially with increasing epochs but saturated after 700 epochs approximately. The mAP for YOLOv5s we've achieved is fairly good considering it's a multiclass database.

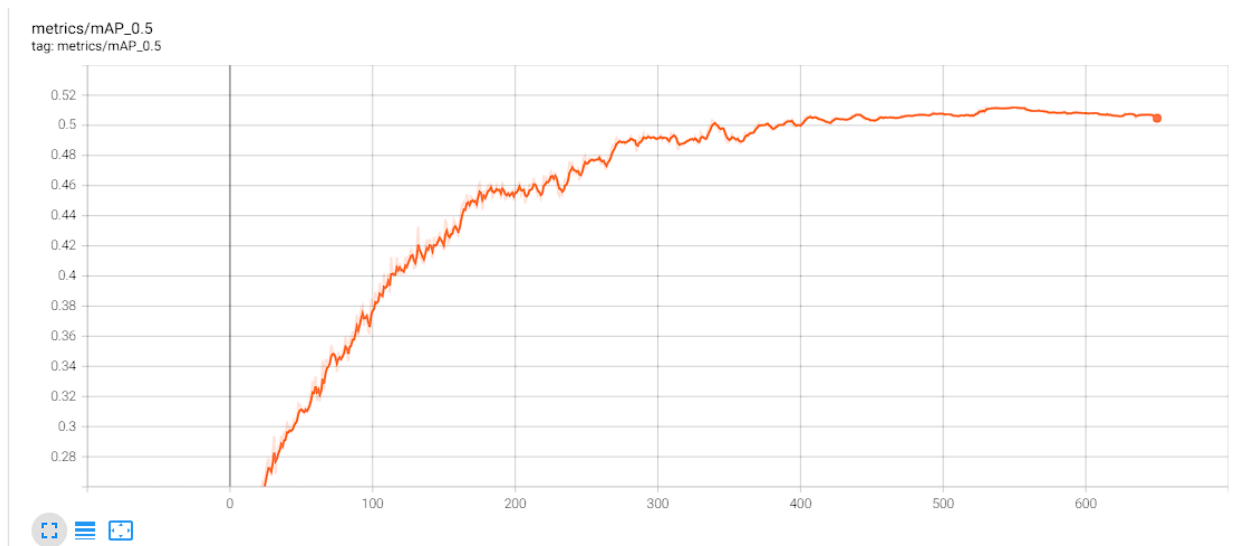


Figure 6.7: Mean Average Precision of YOLOv5

We can measure the performance of the YOLOv5 model from different parameters' values. Tensorboard API was used for generating these performance parameters. Figure 6.8 depicts precision vs confidence, figure 6.9 depicts precision vs recall and figure 6.10 depicts recall vs confidence for the 21 different classes in the database.

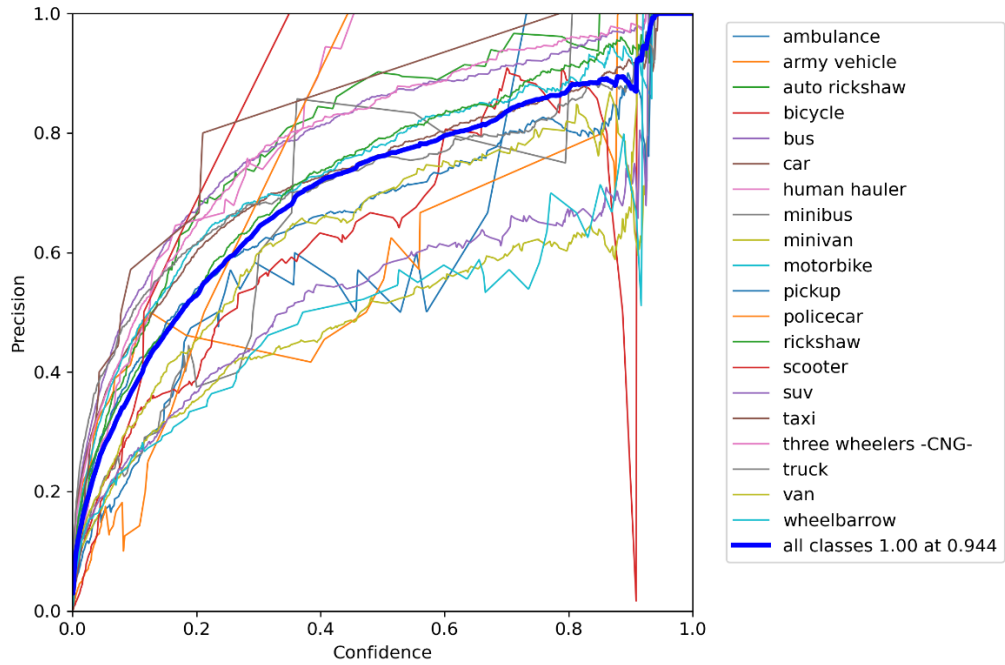


Figure 6.8: Precision vs Confidence Curve

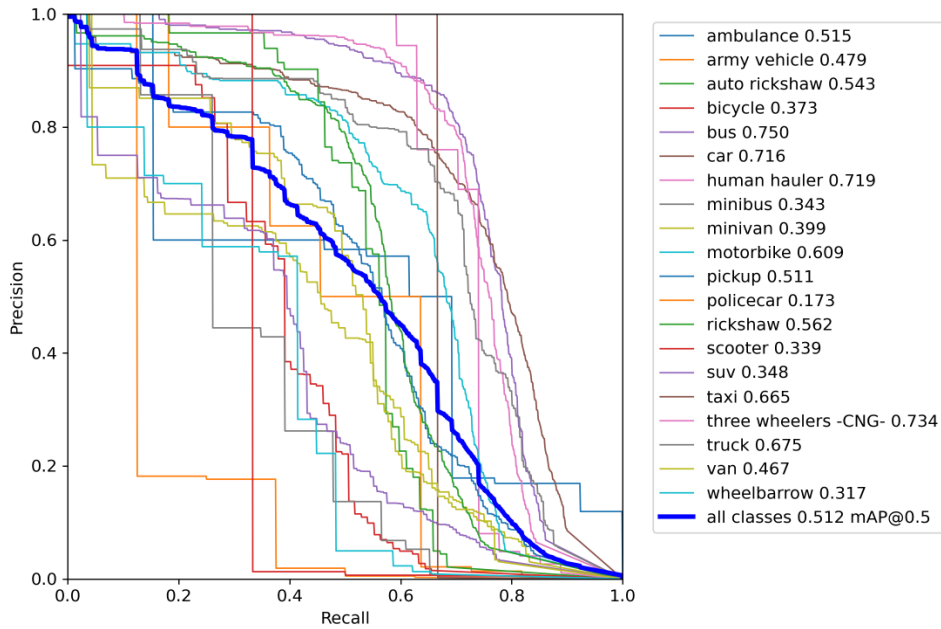


Figure 6.9: Precision vs Recall Curve

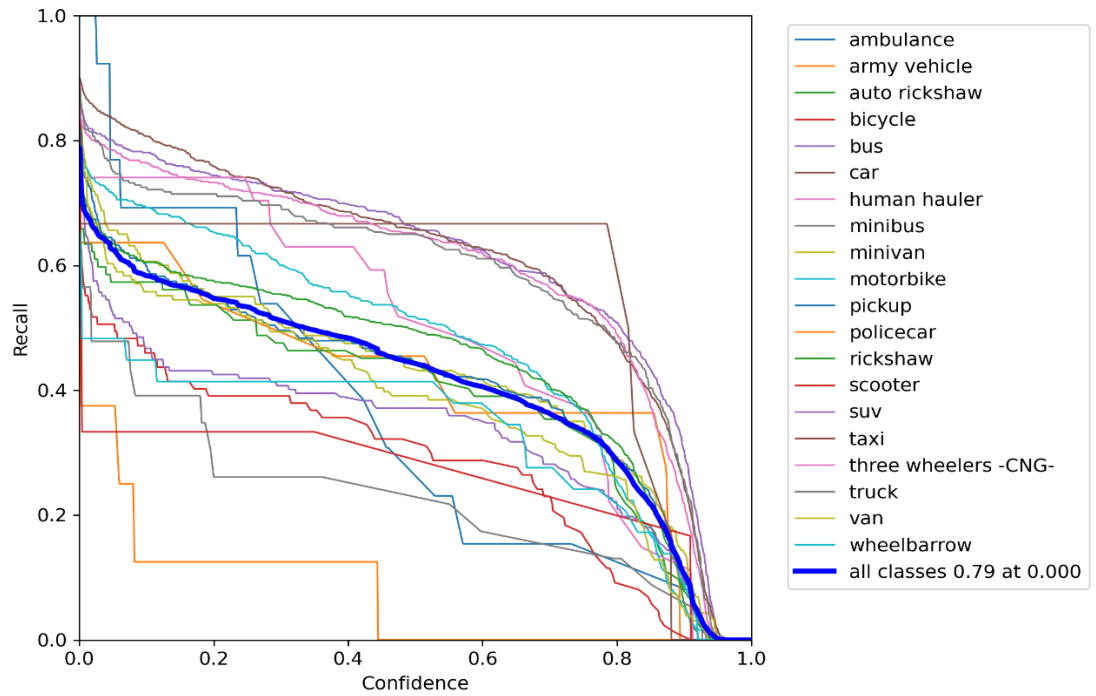


Figure 6.10: Precision vs Confidence Curve

6.2.1 Inference on Images

To test our YOLOv5 model we ran inference on a few images. The results are shown in figures 6.11, 6.12, 6.13, 6.14.

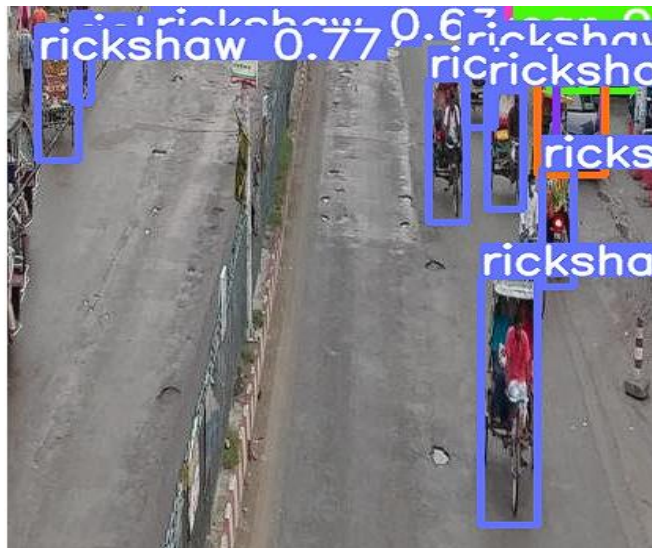


Figure 6.11: YOLOv5 Image Inference 1



Figure 6.12: YOLOv5 Image Inference 2



Figure 6.13: YOLOv5 Image Inference 3

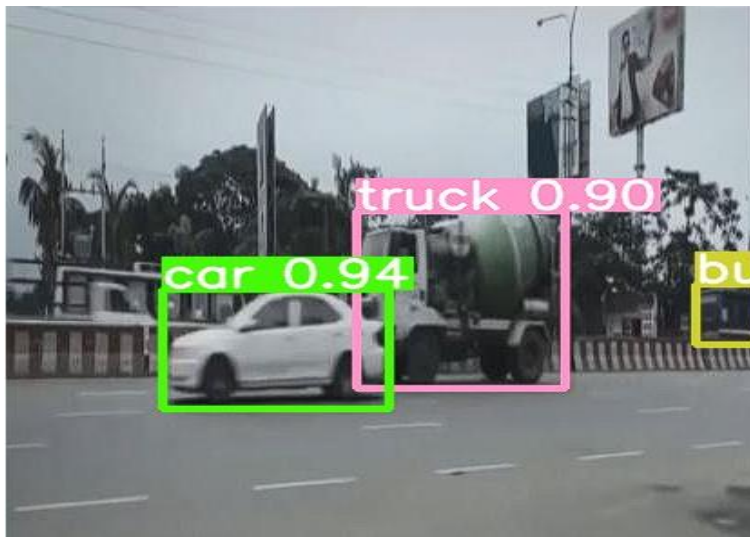


Figure 6.14: YOLOv5 Image Inference 4

6.2.2 Inference on Video

For the same video used before, video inference was done using the YOLOv5 exported model. The result is stored in a drive link and figure 6.15 depicts the QR code that can access it.



Figure 6.15: QR Code of YOLOv5 Video Inference

6.2.3 Data Augmentation and Training on YOLOv5

We have also used data preprocessing for better results in YOLOv5. Roboflow was used to augment the data and generate new dataset. Augmentations applied are:

1. Auto rotation
2. Auto orientation
3. Horizontal flip
4. Rotation(between -15° to 15°)
5. Cutout: 3 boxes with 10% of the size.
6. Bounding box crop.

After augmentation we had 6.3k images of which 612 were used for validation and 299 were used for testing.

TRAIN / TEST SPLIT

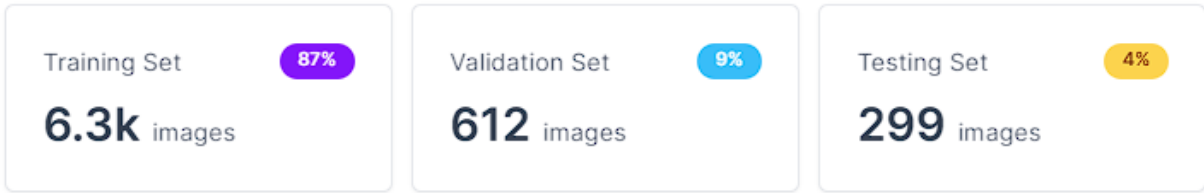


Figure 6.16: Dataset Augmentation Summary

We only could train it for 70 epochs because of resource constraints as the dataset is huge. But judging by the results it appears that under represented class detection should improve if trained for 500 epochs or more. Figure 6.17 depicts the mAP of the augmented dataset.

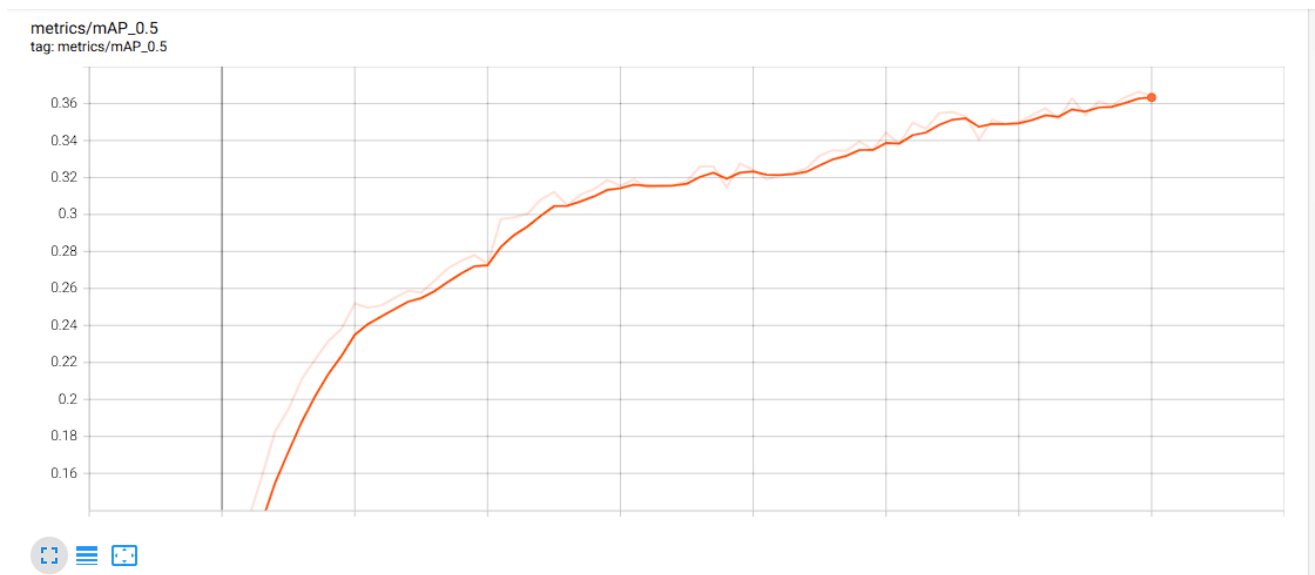


Figure 6.17: mAP for Augmented Dataset

6.3 Result Comparison

As we can see from all the training results, YOLOv5 with the original dataset trained for 1000 epochs has given us the best result closely succeeded by Mobilenetv2. The second training of YOLOv5 that we've done has the worst result but that's only because we couldn't train the model for sufficient epochs because of resource constraints.

So, it is clear that YOLOv5 is superior than Mobilenetv2. It's also the newer of the two algorithms. For the quantitative analysis, we emphasized on the detection process speed and on the number

of classes each framework could detect accurately. The results were synonymous with our qualitative hypothesis and shows a numerical representation of the results obtained. And also YOLOv5 is faster than Mobilenetv2 in our experiments and also in theory. We can see the same result in comparative studies between them. Though Mobilenetv2 has better accuracy but YOLOv5 is much faster. We can use them for object detection according to our need and balance.

Here's a comparison of accuracy between SSD Mobilenetv2 and YOLO from a research work. [32]

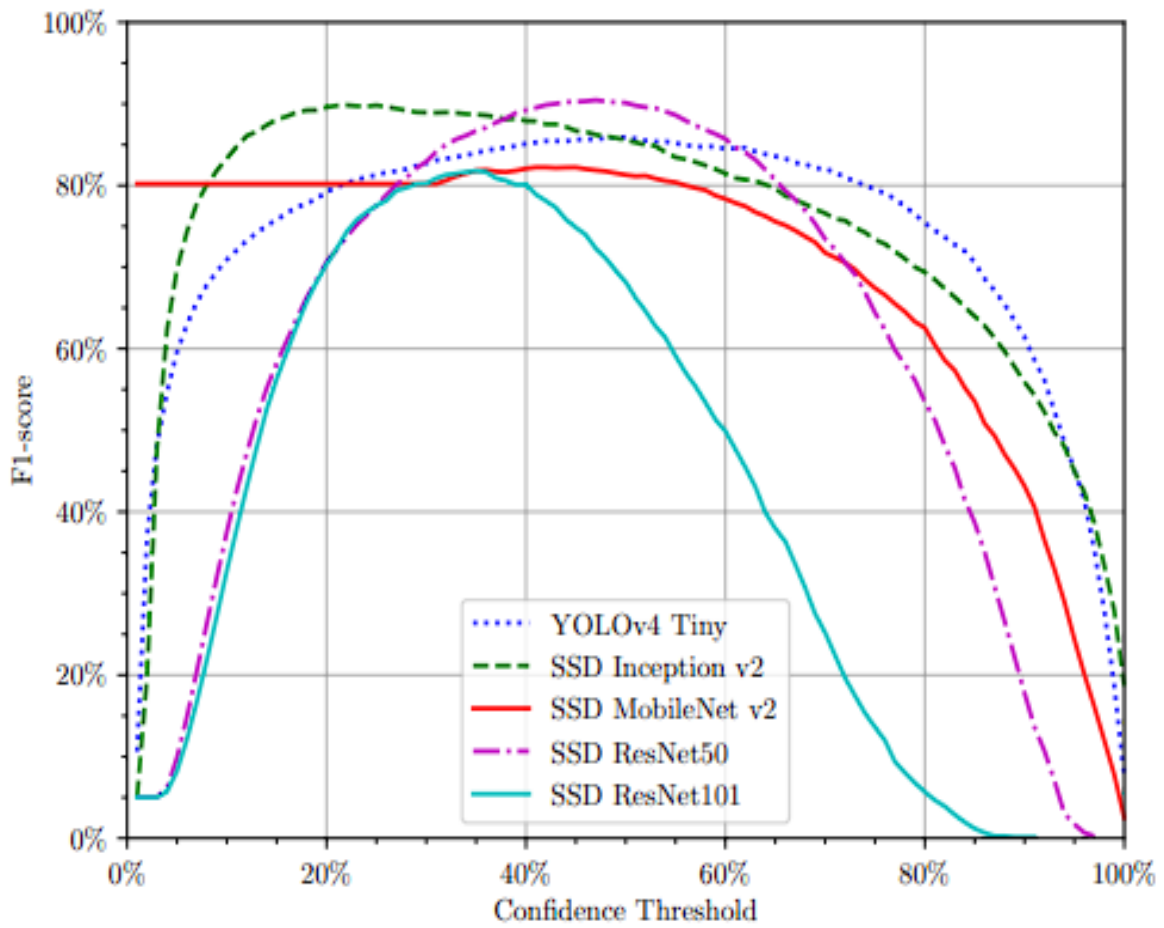


Figure 6.18: Comparison between YOLO and MobileNetV2.

Chapter 7

Conclusion

7.1 Project Significance

We think that our research is important in a number of ways. In terms of its scope, this project makes it possible for Bangladesh to put in place an intelligent traffic surveillance system.

Also, self-driving cars will be able to tell the difference between some of the unique cars in Bangladesh that they have never seen before when equipped with the trained models.

In terms of resources, we trained and ran our models on a moderately powerful workstation and in the Google Colab cloud, both of which can be used by anyone. Improvements in cloud computation are paving a way to the use of this kind of object detection techniques even more.

We were able to get real-time results because the inferences were done pretty quickly. But if our GPU was better, we could train our model faster and get results faster. Uses of this algorithms are given below.

1. Yolo can improve small object detection more than any other algorithm. So, for more speedy and accurate detection of small objects in an image YOLOv5 can be used efficiently.
2. YOLOv5 can detect vehicle even in bad weather condition with higher accuracy. YOLOv5 and also Mobilenet can predict different vehicles, persons, traffic lights even when the weather is not ideal.
3. Object detection algorithms can also be implemented successfully using FPGA devices with low power. As this types of algorithms take low computational power to run in inference.

4. Intelligent ways to find and count vehicles are becoming more and more important for managing highways. Using Object detection algorithm like YOLOv3 and YOLOv5 we can track, detect and count vehicles in a highway using surveillance camera footage.
5. Smart camera vehicle monitoring software is in production as of now. One of them is Camlytics. It uses object detection and counting system of different DNN model to accurately measure the flow of vehicles.

7.2 Future Prospects

The future prospects of this project is exciting and huge. More classifications, such as people and traffic signs, will be added as we enhance the current dataset to better train our models. Premium cloud training facilities will also be used so that we may acquire unlimited training time and increase our model performances even more. Our project is a derivative work but it is critical for traffic management and monitoring, and also for driverless cars. Our research might potentially be used to intelligent parking systems. Some other scopes of our research are:

- Real-time traffic management systems
- Autonomous vehicles
- Measurement of distance between vehicles
- Cloud-based detection and classification
- Automated parking systems
- Surveillance and monitoring

References

- [1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 779–788, 2016, doi: 10.1109/CVPR.2016.91.
- [2] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 4510–4520, 2018, doi: 10.1109/CVPR.2018.00474.
- [3] “No Title”, [Online]. Available: https://en.wikipedia.org/wiki/Object_detection
- [4] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 580–587, 2014, doi: 10.1109/CVPR.2014.81.
- [5] R. Girshick, “Fast R-CNN,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 Inter, pp. 1440–1448, 2015, doi: 10.1109/ICCV.2015.169.
- [6] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017, doi: 10.1109/TPAMI.2016.2577031.
- [7] A. Benjumea, I. Teeti, F. Cuzzolin, and A. Bradley, “YOLO-Z: Improving small object detection in YOLOv5 for autonomous vehicles,” 2021, [Online]. Available: <http://arxiv.org/abs/2112.11798>
- [8] T. Sharma, B. Debaque, N. Duclos, A. Chehri, B. Kinder, and P. Fortier, “Deep Learning-Based Object Detection and Scene Perception under Bad Weather Conditions,” *Electron.*, vol. 11, no. 4, pp. 1–11, 2022, doi: 10.3390/electronics11040563.
- [9] S. P. Kaarmukilan, S. Poddar, and A. K. Thomas, “FPGA based Deep Learning Models for Object Detection and Recognition Comparison of Object Detection: Comparison of object detection models using FPGA,” *Proc. 4th Int. Conf. Comput. Methodol. Commun. ICCMC 2020*, no. Iccmc, pp. 471–474, 2020, doi: 10.1109/ICCMC48092.2020.ICCMC-00088.

- [10] B. Benjdira, T. Khursheed, A. Koubaa, A. Ammar, and K. Ouni, “Car Detection using Unmanned Aerial Vehicles: Comparison between Faster R-CNN and YOLOv3,” *2019 1st Int. Conf. Unmanned Veh. Syst. UVS 2019*, pp. 1–6, 2019, doi: 10.1109/UVS.2019.8658300.
- [11] H. N. Hu *et al.*, “Joint monocular 3D vehicle detection and tracking,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2019-Octob, pp. 5389–5398, 2019, doi: 10.1109/ICCV.2019.00549.
- [12] G. L. Foresti, V. Murino, and C. Regazzoni, “Vehicle recognition and tracking from road image sequences,” *IEEE Trans. Veh. Technol.*, vol. 48, no. 1, pp. 295–300, 1999, doi: 10.1109/25.740109.
- [13] Z. Sun, R. Miller, G. Bebis, and D. DiMeo, “A real-time precrash vehicle detection system,” *Proc. IEEE Work. Appl. Comput. Vis.*, vol. 2002-Janua, pp. 171–176, 2002, doi: 10.1109/ACV.2002.1182177.
- [14] Z. Sun, G. Bebis, and R. Miller, “On-road vehicle detection: A review,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 5, pp. 694–711, 2006, doi: 10.1109/TPAMI.2006.104.
- [15] J. Sang *et al.*, “An improved YOLOv2 for vehicle detection,” *Sensors (Switzerland)*, vol. 18, no. 12, 2018, doi: 10.3390/s18124272.
- [16] Z. Xu, H. Shi, N. Li, C. Xiang, and H. Zhou, “Vehicle Detection under UAV Based on Optimal Dense YOLO Method,” *2018 5th Int. Conf. Syst. Informatics, ICSAI 2018*, no. Icsai, pp. 407–411, 2019, doi: 10.1109/ICSAI.2018.8599403.
- [17] B. N. Li, J. Qin, R. Wang, M. Wang, and X. Li, “Selective level set segmentation using fuzzy region competition,” *IEEE Access*, vol. 4, pp. 4777–4788, 2016, doi: 10.1109/ACCESS.2016.2590440.
- [18] T. Sharma, B. Debaque, N. Duclos, A. Chehri, B. Kinder, and P. Fortier, “Deep Learning-Based Object Detection and Scene Perception under Bad Weather Conditions,” *Electron.*, vol. 11, no. 4, 2022, doi: 10.3390/electronics11040563.
- [19] A. V. Thatte, “Evolution of YOLO — YOLO version 1.” <https://towardsdatascience.com/evolution-of-yolo-yolo-version-1-afb8af302bd2>

- [20] M. Menegaz, “Understanding YOLO.” <https://hackernoon.com/understanding-yolo-f5a74bbc7967>
- [21] J. Solawetz, “YOLOv4 Explained,” 2020. <https://blog.roboflow.com/a-thorough-breakdown-of-yolov4/>
- [22] J. Solawetz, “YOLOv5 New Version - Improvements And Evaluation.” <https://blog.roboflow.com/yolov5-improvements-and-evaluation/>
- [23] cenit, “AlexeyAB / darknet,” 2021. <https://github.com/AlexeyAB/darknet>
- [24] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” 2020, [Online]. Available: <http://arxiv.org/abs/2004.10934>
- [25] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-January, pp. 2261–2269, 2017, doi: 10.1109/CVPR.2017.243.
- [26] Patel, “YOLO V5 — Explained and Demystified,” 2020. <https://towardsai.net/p/computer-vision/yolo-v5-explained-and-demystified>
- [27] A. G. Howard *et al.*, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” 2017, [Online]. Available: <http://arxiv.org/abs/1704.04861>
- [28] “MobileNet V1 Architecture”, [Online]. Available: <https://iq.opengenus.org/mobilenet-v1-architecture/>
- [29] “No Title.” <https://github.com/team>
- [30] “No Title.” <https://stackoverflow.com/>
- [31] “No Title.” <https://www.kaggle.com/>
- [32] S. A. Magalhães *et al.*, “Evaluating the single-shot multibox detector and yolo deep learning models for the detection of tomatoes in a greenhouse,” *Sensors*, vol. 21, no. 10, pp. 1–30, 2021, doi: 10.3390/s21103569.