Dissertation on

Detection of Traffic during Low Illumination and Adverse Weather Conditions

Thesis submitted towards partial fulfilment of the requirements for the degree of

Master in Multimedia Development

Submitted by

Sk Anisuddin Afridi
EXAMINATION ROLL NO.: M4MMD22001
UNIVERSITY REGISTRATION NO.: 154509 of 2020-21

Under the guidance of

Mr. Joydeep Mukherjee
School of Education Technology
Jadavpur University

Course affiliated to
Faculty of Engineering and Technology
Jadavpur University
Kolkata-700032
India

2022

Master in Multimedia Development
Course affiliated to
Faculty of Engineering and Technology
Jadavpur University
Kolkata, India

CERTIFICATE OF RECOMMENDATION

This is to certify that the thesis entitled **"Detection of Traffic during Low Illumination and Adverse Weather Conditions"** is a bonafide work carried out by Sk Anisuddin Afridi under our supervision and guidance for partial fulfilment of the requirements for the degree of Master in Multimedia Development in School of Education Technology, during the academic session 2021-2022.

SUPERVISOR
School of Education Technology
Jadavpur University,
Kolkata-700 032

DIRECTOR
School of Education Technology
Jadavpur University,
Kolkata-700 032

DEAN - FISLM
Jadavpur University,
Kolkata-700 032

Master in Multimedia Development
Course affiliated to
Faculty of Engineering and Technology
Jadavpur University
Kolkata, India

CERTIFICATE OF APPROVAL **

This foregoing thesis is hereby approved as a credible study of an engineering subject carried out and presented in a manner satisfactory to warranty its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not endorse or approve any statement made or opinion expressed or conclusion drawn therein but approve the thesis only for purpose for which it has been submitted.

Committee of final examination for evaluation of Thesis

** Only in case the thesis is approved.

DECLARATION OF ORIGINALITY AND COMPLIANCE OF ACADEMIC ETHICS

I hereby declare that this thesis contains literature survey and original research work by the undersigned candidate, as part of his **Master in Multimedia Development** studies.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by this rule and conduct, I have fully cited and referenced all materials and results that are not original to this work.

NAME	:	**Sk Anisuddin Afridi**
EXAM ROLL NUMBER	:	**M4MMD22001**
THESIS TITLE	:	**Detection of Traffic during Low Illumination and Adverse Weather Conditions**

SIGNATURE: DATE:

Acknowledgement

This dissertation would not been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study.

First and foremost, my utmost gratitude to my guide **Mr. Joydeep Mukherjee**, for his supervision, advice, and guidance form the very early stage of this research as well as giving me extraordinary experiences throughout the work. He spent his precious time in reviewing the project work and provided many insightful comments and constructive criticism, He has been my inspiration as I hurdle all the obstacles in the completion this research work. Without his enthusiasm, advice and unique support this project would never have become a reality.

Needless to mention that, I am grateful to, **Dr. Matangini Chattopadhyay,** Director of School of Education Technology, who had been a source of inspiration and for his timely guidance in the conduct of this dissertation. I am really indebted of **Dr. Ranjan Parekh** and **Dr. Saswati Mukherjee** for their valuable assistance during the entire course of the research work. Their advice and support are highly inspirational and motivating.

Words are inadequate in offering my thanks to all of my classmates of Master in Multimedia Development and M. Tech. IT (Courseware Engineering) who motivated me to complete my research work successfully. I do wish to thank all of our support staff and all of those who were associated with this work contributed in some from or the others.

Last but not the least, I would like to acknowledge and thank my family members for their strong, Unconditional support, Inspiration and encouragement that they have provided me, without which I would be unable to complete this work.

Thanks and regards,

Sk Anisuddin Afridi

Date:

Class Roll No: 002030401001
Exam Roll No: M4MMD22001
Registration No: 154509 of 2020-21
Master in multimedia Development
School of Education Technology,
Jadavpur University,
Kolkata-700032

DEDICATED TO

MY LOVING PARENTS

&

MY PROJECT GUIDE

Mr. Joydeep Mukherjee

Contents

Table of Figures

Chapter 1 Introduction

India gloats of being the second-biggest street organizer on earth. The complete stretch of the Indian street systems remains at an astounding 5.4 million km. Thus, it shapes a gigantic final proposal for the Indian Government to give impeccable streets at each progression. For any Indian, be it normal or millennial, passing through the Indian lanes is absolutely an issue that no one might want to experience. As request moves toward the limit of a street (or of the convergences along the street), outrageous traffic blockage sets in. At the point when vehicles are completely halted for timeframes, this is known as a traffic jam or (casually) a traffic growl up. Traffic blockage can prompt drivers getting baffled and delay which may lead them participating in street rage.

Due to the rising traffic congestions, it is important to organize them to reduce traffic delay. In order to that, information about the traffic conditions can be used by traffic management centres in many ways, including to synchronize traffic lights, to assist drivers in the route selection, and to assist government in intercity as well as intracity connections and new roads planning. Traffic management not only provides benefits to the road users, but also to the municipals and government, and the environment. Drivers could benefit with less time spent in travelling in urban and rural roads, while government can acquire those data for analysis and use those data to improve traffic management. From the perspective of a healthy environment, this effort could reduce the emission of pollutants and have lesser fuel consumption, which collectively reduces the scarcity of resources and the emerging global warming.

1.1 Motivation

The motivation for developing Traffic control system come from many reasons but the biggest motivation behind Automatic Traffic Light Control system is the convenience. Convenience is really another way of saying "time saver" and in today's world where everything moving faster, every second has a value. Most of the technology we use today is based of convenience, for example phones get us information from other people faster. The main aspiration of the designed system is to compute total traffic density at targeted area which is then further used to reduce the traffic congestion caused by vehicles. During the busy hours of a day, the traffic is at its peak and there are various problems related to traffic congestion and delay. This project will be very useful and will be widely used. It can be implemented where ever necessary.

1.2 Historical and Background Concepts

The first automated system for controlling traffic signals was developed by inventors Leonard Casciato and Josef Kates and was used in Toronto in 1954. Several techniques exist to reduce delay of traffic. Generally, the algorithms attempt to reduce delays (user time), stops, exhaust gas emissions, or some other measure of effectiveness. Many optimizations software is geared towards pre-timed coordinated systems.

Normally optimization of signals along a road is a challenging and expensive task, because the sources for traffic monitoring have been limited to inductive loops, cameras or manually counting. However, due to recent advances in information technology, portable devices with Bluetooth and Wi-Fi communication are becoming more common, enabling real-time continuous traffic monitoring and adjustments to traffic signal timing.

By placing sensors along roads, tracking Bluetooth and Wi-Fi devices in passing vehicles, the solution is able to accurately detect and record how long it takes a car to drive along a corridor, segment by segment and in total. This provides historic data for traditional timing methods but also enables real-time feedback to changes in signal programs along with the ability to continuously detect traffic levels and travel time to trigger transitions among programs.

Excessive city traffic can be the most frustrating part about living in urban areas. The solution to metro traffic is a well-balanced mixture of expanded public transit options, remote work, differentiated hubs within the metro area, and electronic tolls. Public transit will help commuters arrive safely at their place of work while eliminating the stress of bumper-to-bumper traffic. Toll roads, especially those with fluctuating tolls dependent upon the time of day and/or level of traffic, also help to improve the level of traffic by deterring some from driving the roadways. Some may choose to work remotely, or run their errands at a different time of day. Finally, differentiated business hubs within a metro area can help alleviate traffic by directing it to several different areas, rather than one centralized location. Cities that spend time and resources on making travel efficient within their borders will attract a healthier, wealthier, and happier population.

1.3 Problem Statement

Abdullah A. et. al. (2020) developed the model which fails to detect vehicles during low illuminations conditions. Earlier studies have been performed based on models which provide a good accuracy in daylight condition, but with the low illumination conditions its accuracy drops drastically.

1.4 Aim and Goals

Since the last decade traffic optimisation has been a great concern to reduce the delay in Traffic. Automatic traffic surveillance for vehicle detection, classification and tracking using vision-based methods has attracted a great deal of interest from researchers in the past decade. Even though many solutions have been proposed, real-time vehicle detection automatic systems need to be further developed.

In this study, YOLO v4 is used to detect vehicles and is combined with a vehicle-counting method to calculate traffic flow. With the help of YOLO object detector, objects like car, pedestrian, by-cycle, trucks, and much more are being detected in video streams using Deep Learning, OpenCV, and Python, during low illumination and adverse weather conditions.

Here OpenCV, Python, and deep learning are used to apply YOLO v4 to video streams.

The aim of this dissertation is to develop a framework for automatic vehicle detection, classification and tracking in video frames, also implement a system for vehicle counting.

Therefore, the main goals are as follow:

- Implement a server that can receive video frames and feed them to a pipeline for processing.
- Implement a multi-object tracking algorithm.
- Perform vehicle detection and its accuracy.
- Visualize the detections, classification and tracking of the vehicles.

1.5 Structure of the Thesis

This thesis is organised as follows:

Chapter1 It gives the Introduction, includes background concepts, problem statement, objectives and overview of the thesis.

Chapter2 Includes Literature Survey.

Chapter3 Object Detection and its Methodology

Chapter4 Implementation of the system using YOLO v4, Google colab, python, VS code.

Chapter5 Conclusion and future scope of the study are described.

1.6 Terminology

In this section, terms used in the report are briefly explained.

- AI Artificial Intelligence
- CCTV Closed-circuit television
- CNN Convolutional Neural Network
- COCO Common Objects in Context
- CPU Central Processing Unit
- CUDA Compute Unified Device Architecture
- CV Computer Vision
- Fast R-CNN Fast Regions-based Convolutional Neural Network
- FCNN Fully Connected Neural Networks
- GPU Graphics Processing Unit
- Python A programming language
- RCNN Region-Based Convolutional Neural Network
- YOLO You Only Look One

Chapter 2 Literature Review

In the past few decades researchers had a great interest in vehicle detection and tracking. The topic attracted the attention quite much. The primary goal and target of the thesis is to develop a system in which the system should be able to detect and track the vehicles automatically whether they are static or moving. There are various studies that have been made to analyse traffic characteristics and their analysis to optimise traffic signal to reduce traffic delay.

Kabara N et. al. (2022) discussed a smart way of managing the traffic using computer vision an object detection for performing various processes for smooth flow of traffic. By using the OpenCV library, and YoloV3 algorithm, they tried to control the traffic lights at the junctions by finding the vehicle density at the signal. The system would also be able to track the vehicles that are breaking the signal and the details would be transferred to the authorities. Majority of the code is written in Python using various libraries and the dataset that will be used is an open-source large-scale object detection, segmentation, and captioning dataset. As per the collected data, the proposed model is projected to perform better. It would be feasible to make more accurate forecasts. It can be used to automate traffic using advanced technologies such as machine learning and computer vision.

Lin J P and **Min T S**. (2018) used YOLO as the infrastructure for image recognition. After YOLO identifies a vehicle in a frame, the counting system has to know whether the vehicles recognized across different frames are the same vehicle or not. If the counting system is capable of such a task, it can then count how many vehicles passing through a specific checkpoint. The system architecture consists of the Detector to generate the bounding box of vehicles, the Buffer to store coordinates of vehicles, and the Counter responsible for vehicle counting.

Azizi A and **Jaison O**. (2020) proposed to perform data annotation and transfer learning from an existing model to construct a new model for vehicle detection and counting in the real-world urban traffic scenes. This was performed using YOLOv3 DarkNet19, which performs worse in the morning and night condition of the custom dataset. Thus, the solution is to retrain the model with a custom dataset from the poor illumination condition environment using a data annotation tool and employs transfer learning with the weight training initialization method. The resulting model improves the counting accuracy very significantly. A tracking mechanism based on consecutive frames comparison was also proposed to aid the counting system. This mechanism may work only on vehicles moving in one direction without occlusion.

Zulaikha K et. al. (2021) focused on a real-time vehicle counting in a complex scene for traffic flow estimation using a multi-level convolutional neural network is proposed. By exploiting the capabilities of deep-learning models in delineating and classifying objects in an image. But the limitation of the proposed method is the system has yet to be tested in more challenging weather and night environment where the lighting condition is low and the image may be distorted with the light beam from the vehicles.

Mattias G and **S H.** (2018) focused on the task of counting vehicles in video using machine learning and neural networks, using the bitrate and the QP-values as the basis for training the models. However, there is a variety of other metadata that could have been used instead. They have also mentioned SNR and motion vectors as two other options but there are countless others that both stems from H.264 and the Axis Zipstream system. While QP and bitrate are relatively easy to extract from videos and have an intuitive interpretation that cannot be sure that the result in the highest performance possible.

Cheng-J L and **J-Y J**. (2022) developed an intelligent traffic-monitoring system based on YOLO and a CFNN. which record traffic volume, and vehicle type information from the road. In this system, YOLO is first used to detect vehicles and is combined with a vehicle-counting method to calculate traffic flow.

Aderonke A O and **Nicholas K**. (2019) focus is on creating a vehicle counting system to be used on urban roads in Nigeria, specifically to be installed on pedestrian and overhead bridges. The goal is to study existing vehicle counting systems in Nigeria and develop an improved and robust system by exploiting state-of-the-art Computer Vision techniques and algorithms for object detection and counting. Developed a vehicle counting system based on YOLO detection algorithm.

Aswathi R et. al. (2022) took a CCTV camera mounted on the road captures videos, and these video frames were fed to a YOLO based CNN model, which performs the dual task of detecting a vehicle on the road and identifying the type of vehicle. Once the vehicles were detected in a frame, other parameters such as vehicle count, vehicle density, vehicle occupancy and traffic congestion were computed for traffic congestion analysis.

Manoj B et. al. (2021) focused on the task of counting vehicles autonomously in an edge device using computer vision and deep learning. Estimating classified counts using cameras can be seen as the classic object counting task in computer vision. In this paper, they have presented a low-compute framework to estimate the traffic counts. Solution consists of a fast object detector (Tiny YOLO) trained using Knowledge Distillation from 3 teacher networks (Faster RCNN, YOLOv3, and SSD), Tensor-RT Optimization of the learned model, and estimating the count of the traffic using the SORT Algorithm in real-time on edge devices.

Both the Effectiveness and Efficiency of solution are experimentally illustrated using multiple data sources.

H R-Rangel et. al. (2022) used this data set to compare five different statistical methods and three machine learning methods to evaluate their accuracy in estimating the cars' speed in real-time. For vehicle estimation requires a feature extraction process using YOLOv3 and Kalman filter to detect and track vehicles.

Chieh M L and **Jyh C J** (2021) proposed a neural network that fuses the data received from a camera system on a gantry to detect moving objects and calculate the relative position and velocity of the vehicles traveling on a freeway. This information is used to estimate the traffic flow. To estimate the traffic flows at both microscopic and macroscopic levels, this paper used YOLO v4 and Deep SORT for vehicle detection and tracking. The number of vehicles passing on the freeway was then calculated by drawing virtual lines and hot zones. Need to improve the prediction accuracy, especially that of lane-specific counting. Cannot automatically describe the complete traffic situation on a larger scale.

Rosaldo J F R et. al. (2018) used the YOLOv3 model with pre-trained weights on the COCO dataset for vehicle detection and classification. The results are fed into a multi-object tracking algorithm. The implemented tracker is based on the IOU tracker a track-by-detection approach where first an object detector is applied to a frame and second a tracker is used to associate the detections to tracks. Then a tracking zone is identified in the video image where the vehicles are tracked and counted.

Huansheng Song et. al. (2019) proposed vehicle detection and counting system, the highway road surface in the image is first extracted and divided into a remote area and a proximal area by segmentation method; the method is crucial for improving vehicle detection. Then, the above two areas are placed into the YOLOv3 network to detect the type and location of the vehicle. Finally, the vehicle trajectories are obtained by the ORB algorithm. In summary, vehicles in Europe, such as in Germany, France, the United Kingdom, and the Netherlands, have similar characteristics to the vehicles in this vehicle dataset, and the angle and height of the road surveillance cameras installed in these countries can also clearly capture the long-distance road surface. Therefore, the methodology and results of the vehicle detection and counting system provided in this analysis will become references for mainly European transport studies.

Vishal Mandal et. al. (2020) approach to automatically monitor real time traffic footage using deep convolutional neural networks and a stand-alone graphical user interface. They describe the results of research received in the process of developing models that serve as an integrated framework for an artificial intelligence enabled traffic monitoring system. The proposed system deploys several state-of-the-art deep learning algorithms to automate

different traffic monitoring needs. Taking advantage of a large database of annotated video surveillance data, deep learning-based models are trained to detect queues, track stationary vehicles, and tabulate vehicle counts. An automatic traffic monitoring system is developed that builds on robust deep learning models and facilitates traffic monitoring using a graphical user interface. Deep learning algorithms, such as Mask R-CNN, Faster R-CNN, YOLO and CenterNet.

Chapter 3 Brief Overview of Object Detection and its Methodology

In this chapter, the details of the computer vision approaches and its complimentary platforms have been discussed briefly.

3.1 Computer Vision

Computer vision is the field of computer science that focuses on creating digital systems that can process, analyse, and make sense of visual data (images or videos) in the same way that humans do. The concept of computer vision is based on teaching computers to process an image at a pixel level and understand it. Technically, machines attempt to retrieve visual information, handle it, and interpret results through special software algorithms.

It is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do.

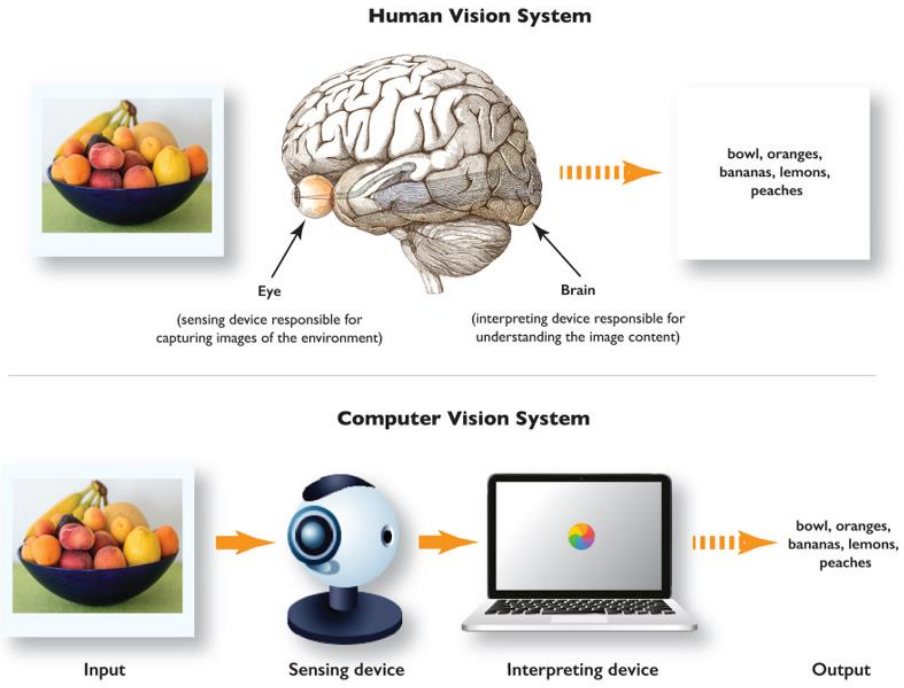


Figure 3.1: Human vision and computer vision systems process visual data in a similar way. (Image source: https://freecontent.manning.com/wp-content/uploads/human-and-artificial-sensing.png)

- **How does computer vision work?**

Computer vision technology tends to mimic the way the human brain works. But how does our brain solve visual object recognition? One of the popular hypothesis states that our brains rely on patterns to decode individual objects. This concept is used to create computer vision systems.

Computer vision algorithms that we use today are based on pattern recognition. We train computers on a massive amount of visual data—computers process images, label objects on them, and find patterns in those objects. For example, if we send a million images of flowers, the computer will analyse them, identify patterns that are similar to all flowers and, at the end of this process, will create a model "flower." As a result, the computer will be able to accurately detect whether a particular image is a flower every time we send them pictures.

Golan Levin, in his article Image Processing and Computer Vision, provides technical details about the process that machines follow in interpreting images. In short, machines interpret images as a series of pixels, each with their own set of colour values. For example, below is a picture of Abraham Lincoln. Each pixel's brightness in this image is represented by a single 8-bit number, ranging from 0 (black) to 255 (white). These numbers are what software sees when you input an image. This data is provided as an input to the computer vision algorithm that will be responsible for further analysis and decision making.

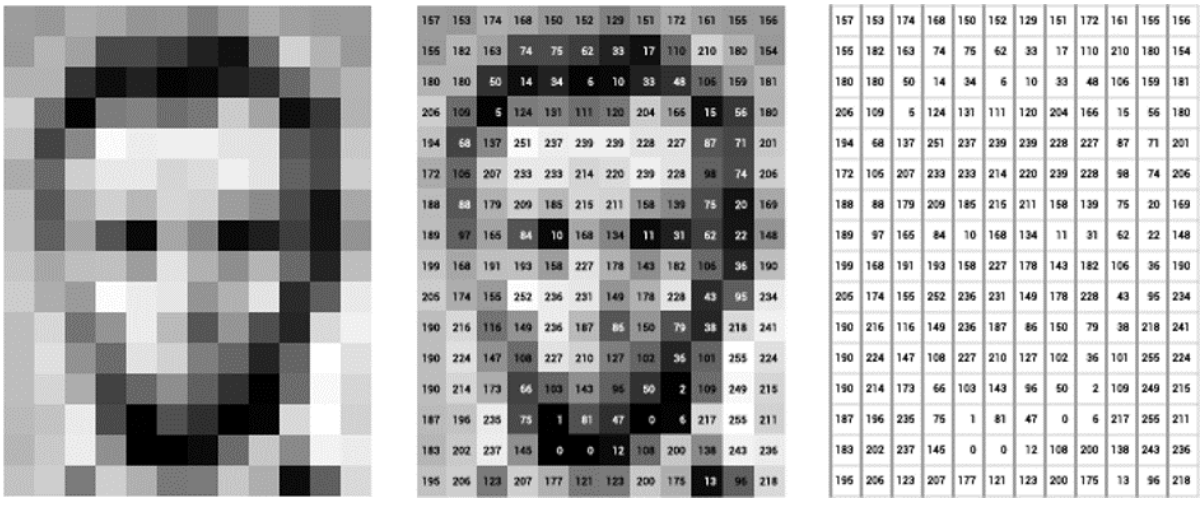


Figure 3.2: Colour values of individual pixels are converted into a simple array of numbers used as input for a computer vision algorithm. (Image credit: Lin CJ et.al. (2022))

OpenCV-Python is a library of Python bindings designed to solve computer vision problems.

3.2 Object Detection

In computer vision, there are so many applications and uses, one of which is object detection. Object detection is a subset of computer vision that is used to detect the presence, location, and type of objects in images. Object detection is also a combination of three functions; Object recognition, to find objects in an image, Object localization, to find where exactly in the image the objects are located, and Object classification, to detect what particular objects are in that image.

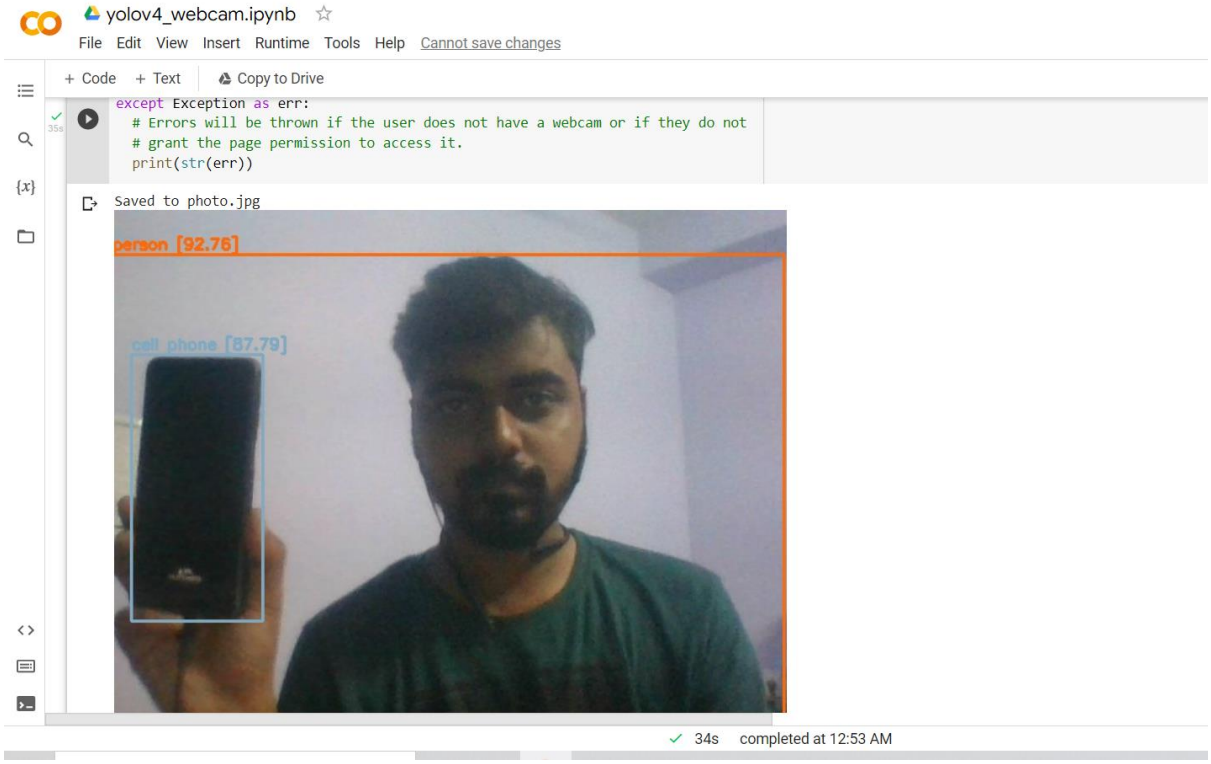


Figure 3.3: Preliminary object detection

As per Figure 3.3, the objects have been recognized (two objects in the image), they have been localized (bounding boxes around the two images) and they have been classified (person and cell phone with 92.78% and 87.79% respectively). This is the working of object detection. Object detection has several uses, some of which are:

- For tracking objects, example, tracking a ball during a football match or tennis game, also for traffic checking
- For automated CCTV surveillance, to help increase security and protect people and property

- For detecting people, mostly for the police and security services, to use to detect any unscrupulous activities by individuals
- For vehicle detection, mostly for self-driving cars, to detect vehicles and respond accordingly

Over the past few years, deep learning architectures are achieving extraordinary and state-of-the-art performances for object detection. There are many families of object detection created over the years, but the most popular ones are the YOLO and RCNN object detection families. This article would focus on the YOLO architecture.

3.3 Overview of You only look once (YOLO)

The YOLO algorithm was developed in 2015.

3.3.1 What is YOLO:

YOLO is an abbreviation for the term 'You Only Look Once'. This is an algorithm that detects and recognizes various objects in a picture (in real-time). Object detection in YOLO is done as a regression problem and provides the class probabilities of the detected images.

YOLO algorithm employs convolutional neural networks (CNN) to detect objects in real-time. As the name suggests, the algorithm requires only a single forward propagation through a neural network to detect objects.

This means that prediction in the entire image is done in a single algorithm run. The CNN is used to predict various class probabilities and bounding boxes simultaneously.

3.3.2 Why the YOLO algorithm is important

YOLO algorithm is important because of the following reasons:

- **Speed:** This algorithm improves the speed of detection because it can predict objects in real-time.
- **High accuracy:** YOLO is a predictive technique that provides accurate results with minimal background errors.
- **Learning capabilities:** The algorithm has excellent learning capabilities that enable it to learn the representations of objects and apply them in object detection.

3.3.3 How the YOLO algorithm works

YOLO algorithm employs convolutional neural networks (CNN) to detect objects in real-time. As the name suggests, the algorithm requires only a single forward propagation through a neural network to detect objects.

This means that prediction in the entire image is done in a single algorithm run. The CNN is used to predict various class probabilities and bounding boxes simultaneously.

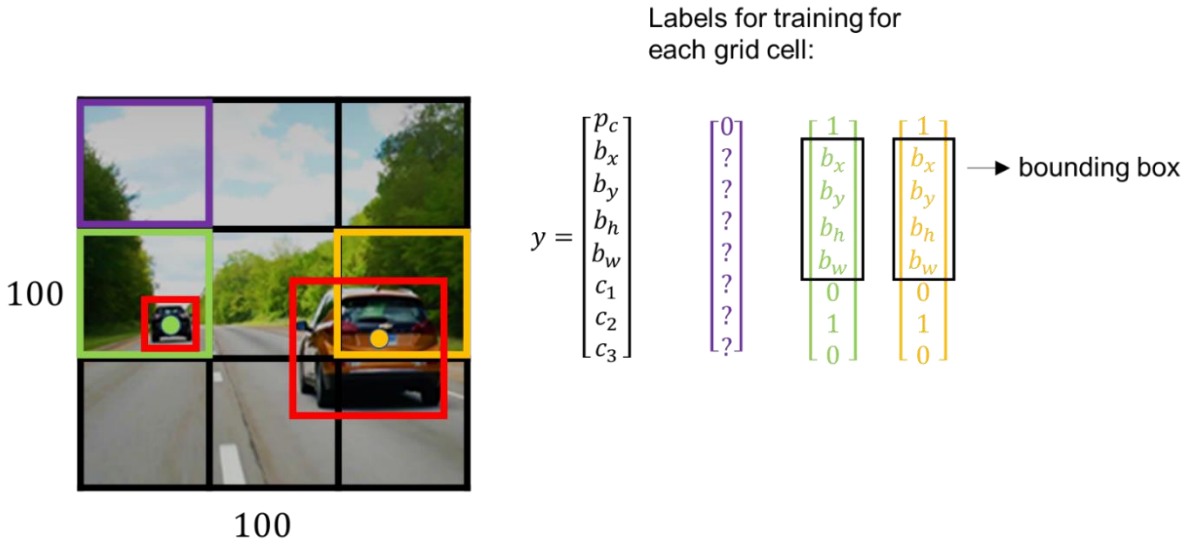


Figure 3.4: Grid cell (source: https://www.irjet.net/archives/V8/i6/IRJET-V8I6703.pdf)

An input image is split into several grid cells, where each cell has a duty to predict a bounding box if the middle of the bounding box falls within that cell. The predicted bounding box has x, y coordinates and height, and width. Considering the image above, the image is divided into 9 grids, where 5 of them have part of the object(cars) in it, but only 2 of the cells contain the middle of the car, those are the cells that would be chosen (anchor box mechanism).

Each cell would be represented by vectors, with the height, width, x and y coordinates, if there's a presence of an object and the type of object in the cell, as the vectors. If there's no object, that cell does not proceed with any detection. If an object is detected, then the type of object is indicated and the x and y coordinates, together with the bounding boxes are indicated and the final prediction is made.

3.4 Understanding YOLO object detection: the YOLO algorithm

To understand the YOLO algorithm, it is necessary to establish what is actually being predicted. Ultimately, we aim to predict a class of an object and the bounding box specifying object location. Each bounding box can be described using four descriptors:

- Centre of a bounding box (**bxby**)
- Width (**bw**)
- Height (**bh**)
- Value cis corresponding to a class of an object (e.g., car, traffic lights, etc.)

In addition, we have to predict the pc value, which is the probability that there is an object in the bounding box.

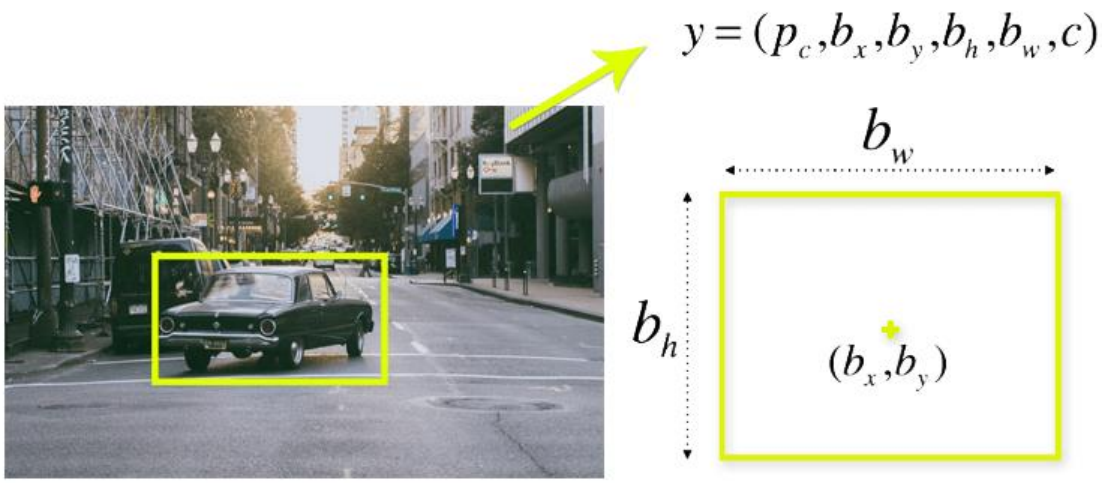


Figure 3.5: Bounding box probability calculation (source: https://www.kdnuggets.com/2018/09/object-detection-image-classification-yolo.html)

As we mentioned above, when working with the YOLO algorithm we are not searching for interesting regions in our image that could potentially contain an object.

Instead, we are splitting our image into cells, typically using a 19×19 grid. Each cell is responsible for predicting 5 bounding boxes (in case there is more than one object in this cell). Therefore, we arrive at a large number of 1805 bounding boxes for one image. Rather than seizing the day with YOLO and Carpe Diem, we're looking to seize object probability. The exchange of accuracy for more speed isn't reckless behaviour, but a necessary requirement for faster real-time object detection.

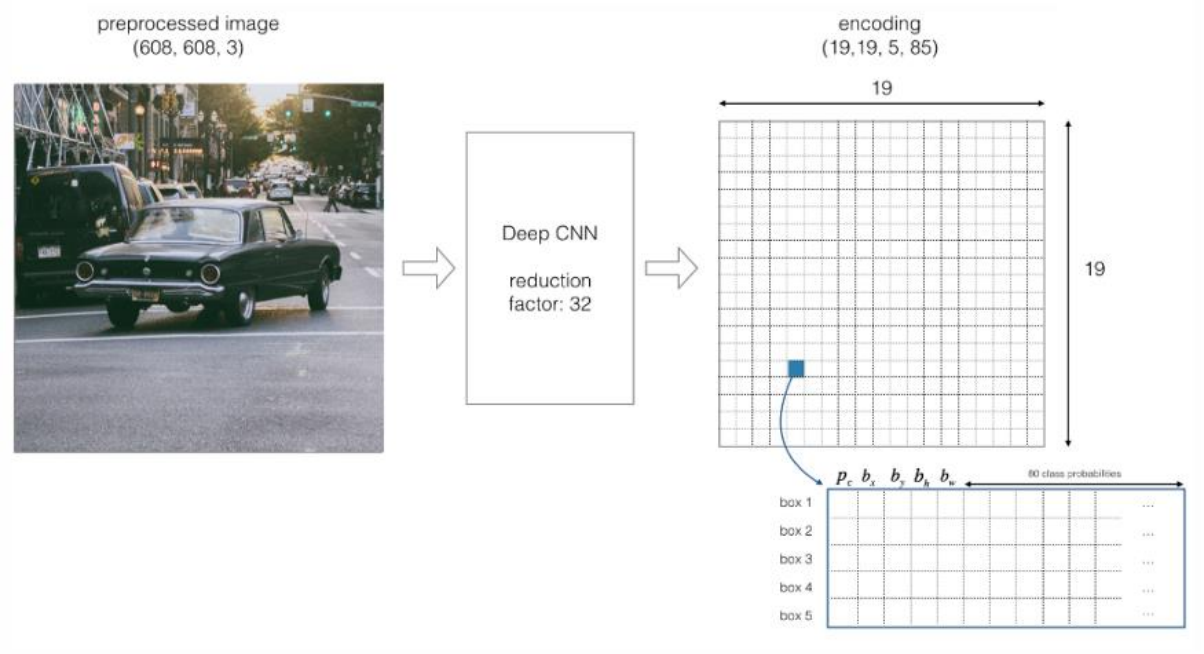


Figure 3.6: Image size-reduction (source: https://towardsdatascience.com/yolo-you-only-look-once-3dbdbb608ec4)

Most of these cells and bounding boxes will not contain an object. Therefore, we predict the value pc, which serves to remove boxes with low object probability and bounding boxes with the highest shared area in a process called non-max suppression.

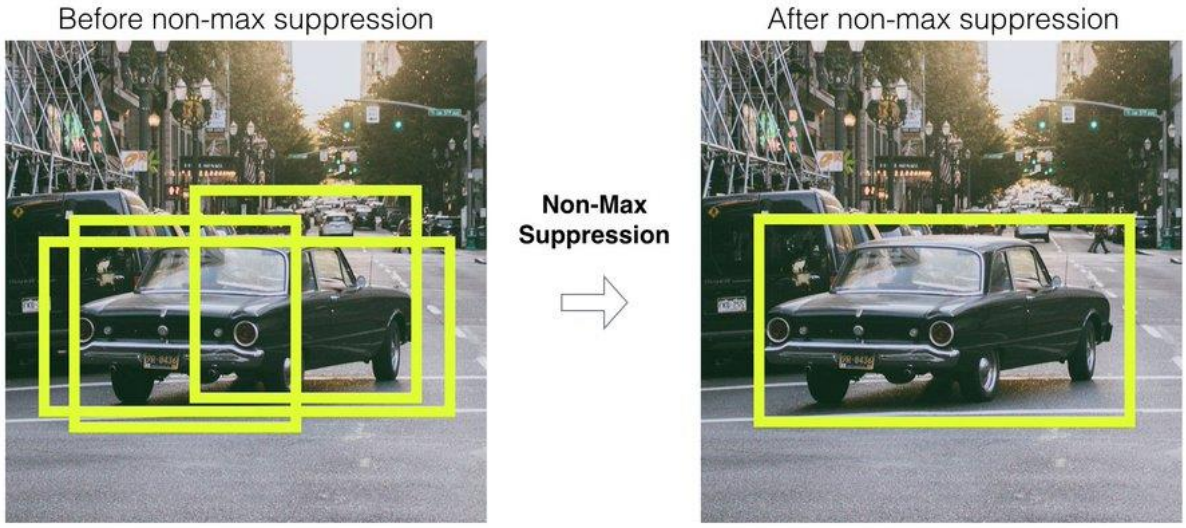


Figure 3.7: Non-max suppression (source: https://medium.com/swlh/beginners-guide-to-everything-image-recognition-50771e786601)

3.4.1 YOLO V3 ARCHITECTURE

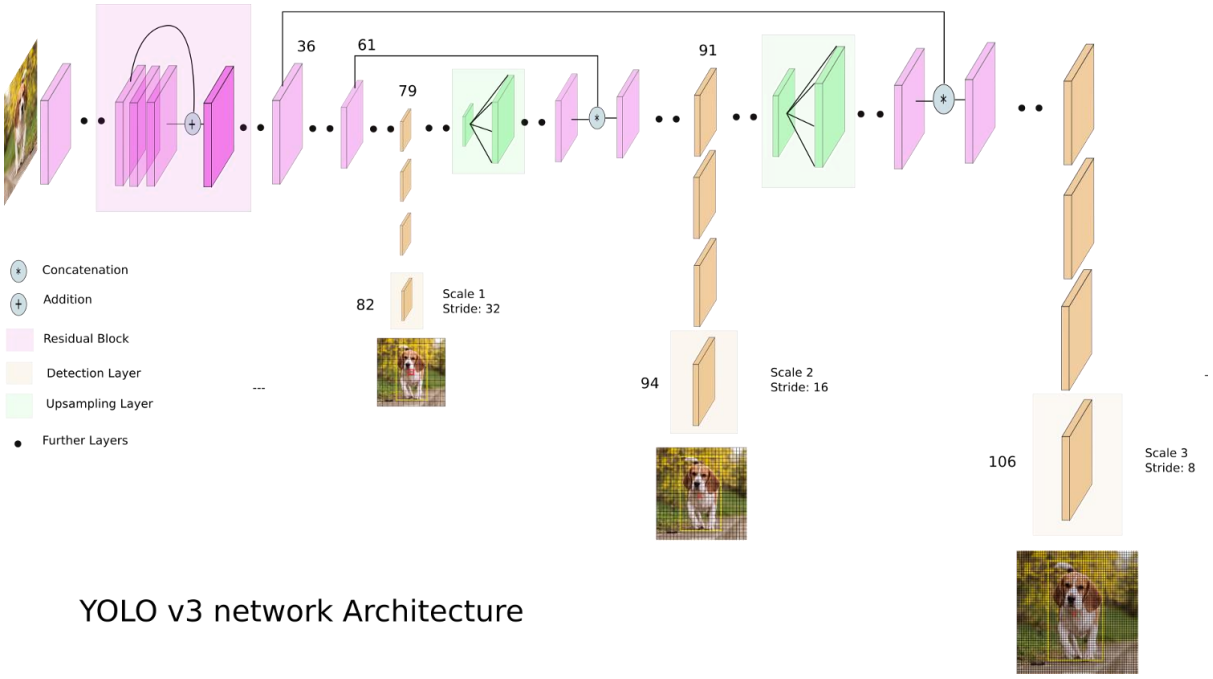


Figure 3.8: YOLO v3 network Architecture (source: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b)

The Yolov3 architecture has residual skip connections and an up-sampling layer. The key novelty is this algorithm is that it makes its detections at three different scales. The Yolo algorithm is a fully connected layer and the detection is done by using a 1×1 kernel on the feature maps to make the detections at three different locations using three different scales, as can be seen in the diagram above.

As mentioned, the shape of the kernel for detection is a 1×1× (B×(A+C)), where B indicates the number of bounding boxes, C refers to the number of classes and the A refers to the 4 bounding box attributes (height, width, x and y coordinates). The Yolov3 algorithm was trained on a dataset known as the coco dataset, which has 80 classes and the bounding box attributes summing to 3, so in effect, the kernel size becomes 1×1×255.

Moving forward, an assumption would be made that there's an image of size 480×480, that we are using the yolov3 to detect the objects in the image. As mentioned earlier, the yolov3 makes detection at three scales, it down samples the input images by 32,16, and 8. For the first 81 layers, the image is down sampled by the stride of 32 of the 81st layer. Regarding our image of 480×480, the resulting feature map would be 480/32, which is 15×15. The first detection is performed here using the aforementioned 1×1 detection kernel, resulting in a feature map of 15×15×255.

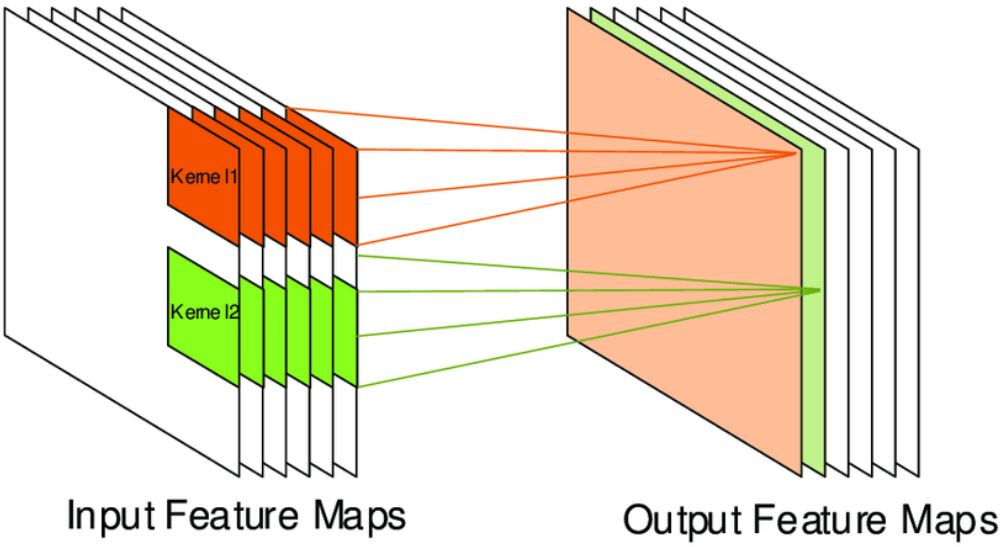


Figure 3.9: YOLO v3 feature map (source: https://www.researchgate.net/figure/Input-and-output-feature-maps-of-a-convolutional-layer_fig1_334819564)

From the 79th layer, the feature map is up sampled to twice its dimensions (30×30), which is then concatenated with the feature map from the 61st layer. The second detection is made at the 94th layer, generating a feature map of 30×30×255. A similar process like before is followed, where the feature map from the 91st layer is concatenated with the feature map in the 36th layer, which is then subjected to a few 1×1 convolution layers to combine the previous layers. The final detection is made at the 106th layer, generating a feature map of 60×60×255. Each layer has a role in the detection, the 15×15 helps detect the larger objects, the 30×30 helps detect medium-sized objects, and the 60×60 helps detect large-sized objects.

Yolov3 uses independent logistic classifiers in place of the SoftMax function to determine the class of an input image. It also replaces the mean squared error with the binary cross-entropy loss, in simpler terms, the probability of object in the image and the class predictions are done using logistic regression.

3.4.2 The architecture of YOLOv4

YOLOv4 is designed based on recent research findings, using CSPDarknet53 as a Backbone, SPP (Spatial pyramid pooling) and PAN (Path Aggregation Network) for what is referred to as “the Neck”, and YOLOv3 for “the Head”.

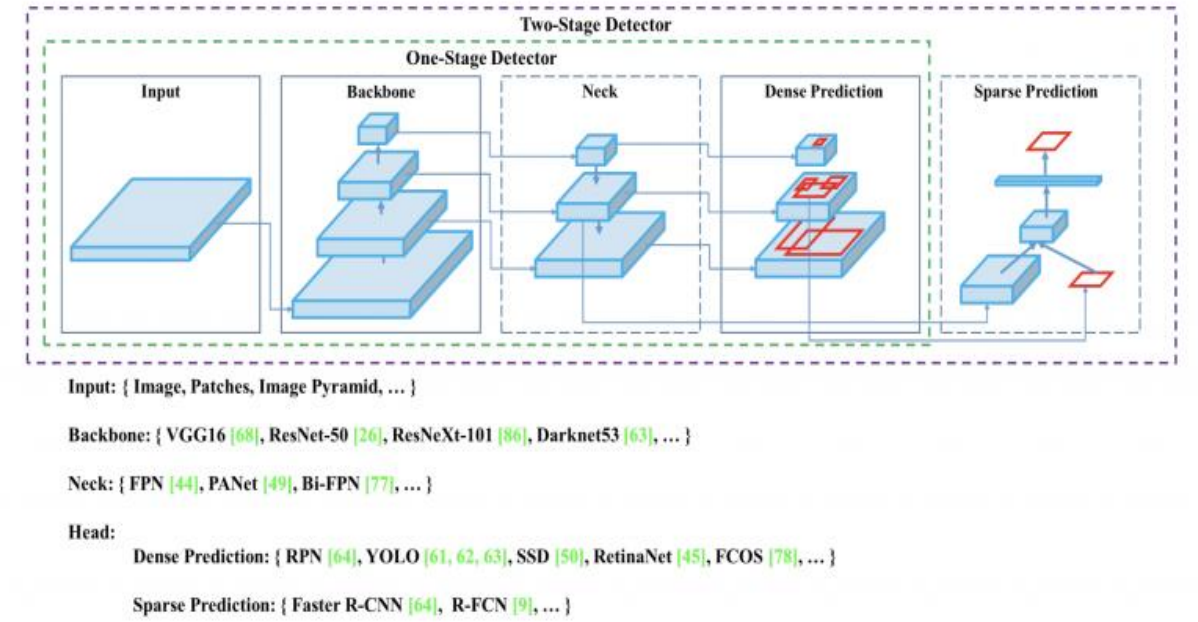


Figure 3.10: Object Detection Workflow (source: https://pyimagesearch.com/2022/05/16/achieving-optimal-speed-and-accuracy-in-object-detection-yolov4/)

CSPNet is an optimization method aiming at partitioning feature map of the base layer into two parts and then merging them through a cross-stage hierarchy presented below.

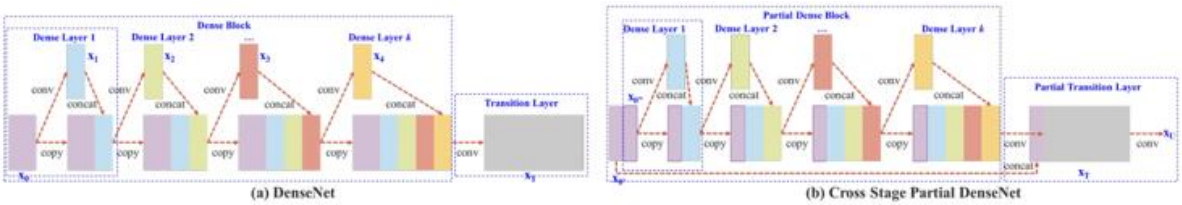


Figure 3.11: Illustrations of DenseNet and Cross Stage Partial DenseNet (source: https://darrencarvalho.medium.com/cspnet-reason-why-sota-algorithms-are-fast-and-efficient-1fd2e245c701)

Yolov4 is an improvement on the Yolov3 algorithm by having an improvement in the mean average precision(mAP) by as much as 10% and the number of frames per second by 12%. The Yolov4 architecture has 4 distinct blocks as shown in the image above, The backbone, the neck, the dense prediction, and the sparse prediction.

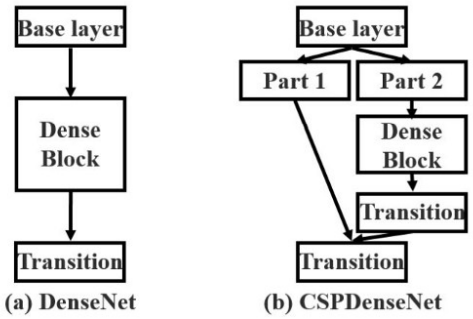


Figure 3.12: Schematic layer view of DenseNet and CSPDenseNet(source: https://darrencarvalho.medium.com/cspnet-reason-why-sota-algorithms-are-fast-and-efficient-1fd2e245c701)

The backbone is the feature extraction architecture which is the CSPDarknet53. This CSPDarknet53 stands for Cross-Spatial -Partial connections, which is used to split the current layer into two parts, one to pass through convolution layers and the other that would not pass-through convolutions, after which the results are aggregated. Above is an example with DenseNet.

The neck helps to add layers between the backbone and the dense prediction block(head), which is a bit like what the ResNet architecture does. The yolov4 architecture uses a modified Path aggregation network, a modified spatial attention module, and a modified spatial pyramid pooling, which are all used to aggregate the information to improve accuracy. The image above shows spatial pyramid pooling.

The head (Dense prediction) is used for locating bounding boxes and for classification. The process is the same as the one described for Yolo v3, the bounding box coordinates (x,y, height, and width) are detected as well as the score. Remember, the main goal of the Yolo algorithm is to divide an input image into several grid cells and predict the probability that a cell contains an object using anchor boxes. The output is then a vector with the bounding box coordinates and the probabilities of the classes.

There are other techniques the authors of the yolov4 algorithm used to improve accuracy during training and afterward. They are the bag of freebies and bag of specials.

The bag of freebies helps during training and without increasing inference time. The bag freebies have 2 techniques, the first being the Bag of freebies for the backbone, which uses the cut mix and mosaic of data augmentation and drop block for regularization, and the second being the bag of freebies for detection, which adds more to the backbone, such as self-adversarial training, random training shapes and the rest.

The bag of specials changes the architecture and increases the inference time by a little bit. The bag of specials also has 2 techniques, the first is the bag of specials for the

backbone which uses the mish activation, cross-stage partial connections, the second is the bag of specials for detection, which uses the SPP-block, the SAM block, and others.

There's a lot of information regarding the bag of freebies and bag of specials, but to make it simple, these are the key takeaways, both help greatly during training and after training. Bag of freebies uses techniques such as data augmentation and dropout and the bag of specials involves the neck, non-maximum suppression, and the likes. The below image shows the architecture of the main blocks of the Yolov4.

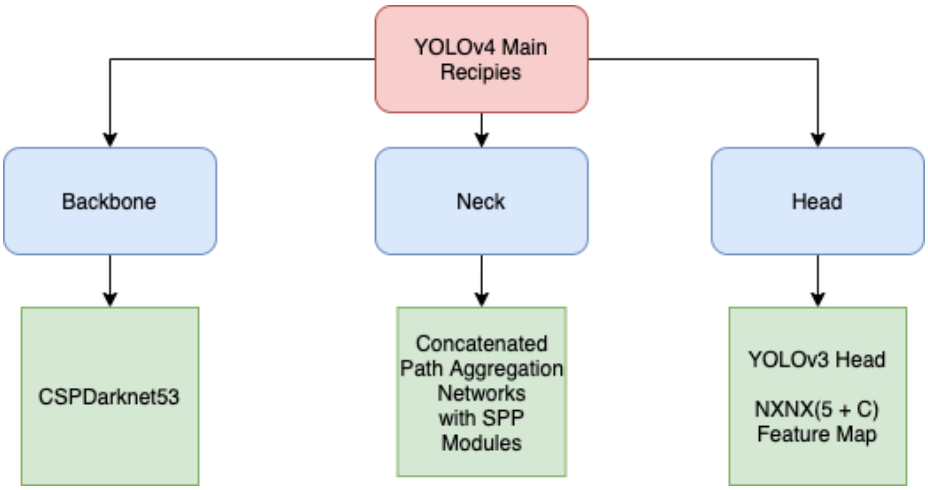


Figure 3.13: Flowchart of YOLO v4 (source: https://medium.com/visionwizard/yolov4-version-0-introduction-90514b413ccf)

3.4.3 YOLO v3 and YOLO v4

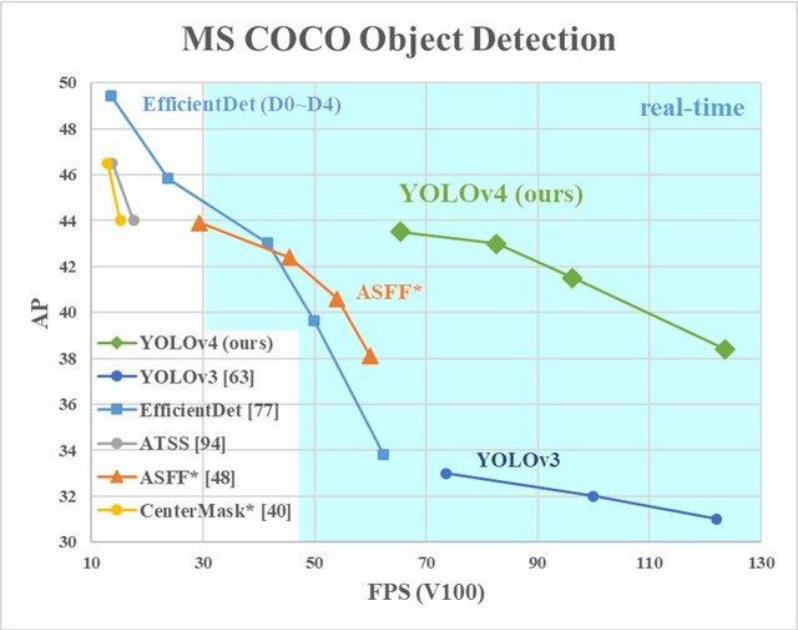


Figure 3.14: MS COCO Object Detection (source: https://neurohive.io/en/news/yolo-v4-is-the-new-state-of-the-art-object-detector/)

The y-axis is the absolute precision and the x-axis is the frame per second. The blue shaded part of the graph is for real-time detection (webcam, street cameras, etc), and the white is for still detection(pictures). It can be seen that the yolov4 algorithm does very well in real-time detection, achieving an average precision between 38 and 44, and frames per second between 60 and 120. The yolov3 achieves an average precision between 31 and 33 and frames per second between 71 and 120.

3.5 What is Google Colab

Google is quite aggressive in AI research. Over many years, Google developed AI framework called TensorFlow and a development tool called Colaboratory. Today TensorFlow is open-sourced and since 2017, Google made Colaboratory free for public use. Colaboratory is now known as Google Colab or simply Colab.

Another attractive feature that Google offers to the developers is the use of GPU. Colab supports GPU and it is totally free. The reasons for making it free for public could be to make its software a standard in the academics for teaching machine learning and data science. It may also have a long-term perspective of building a customer base for Google Cloud APIs which are sold per-use basis.

3.5.1 What Colab Offers You

As a programmer, you can perform the following using Google Colab.

- Write and execute code in Python
- Document your code that supports mathematical equations
- Create/Upload/Share notebooks
- Import/Save notebooks from/to Google Drive
- Import/Publish notebooks from GitHub
- Import external datasets e.g., from Kaggle
- Integrate PyTorch, TensorFlow, Keras, OpenCV
- Free Cloud service with free GPU

Chapter 4 Implementation and Results

In this chapter, the implementation of an intelligent traffic surveillance according to the methods are as per previous chapter are described. Some experiments has been performed to test and present the results.

Detailed steps of object tracking have been described further.

4.1 Traffic Monitoring and Video Processing

The implementation of Running DeepSort Object Tracking with Yolov4 Object Detections in Google Colab is preseented. Object tracking implemented with YOLOv4, DeepSort, and TensorFlow. YOLOv4 is a state-of-the-art algorithm that uses deep convolutional neural networks to perform object detections. The output of YOLOv4 feed these object detections into Deep SORT (Simple Online and Realtime Tracking with a Deep Association Metric) in order to create a highly accurate object tracker can be obtained.

The proposed solution is to process images received from a traffic camera that monitors of the motorway and performs the tasks of detecting, classifying, tracking, and counting vehicles. The detection and classification of vehicles is done using YOLOv4 and a pre-trained model in the COCO dataset, which has 80 classes including car, truck, bus, motorcycle and pedestrian etc. As a result of these processes, the application will be able to detect the vehicles and locate their exact position in the image, drawing a bounding box around them and an id. Finally, vehicles passing in the tracking and counting zones are tracked and counted. The resulting images and the counters results will be displayed in a web application so that the end user can view them.

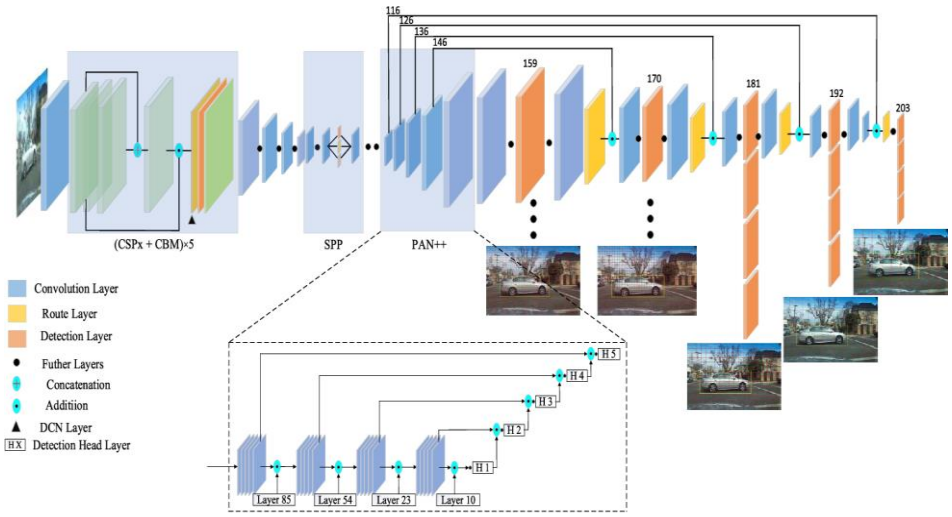


Figure 4.1:Traffic monitoring application diagram (source: https://towardsdatascience.com/yolov4-5d-an-enhancement-of-yolov4-for-autonomous-driving-2827a566be4a)

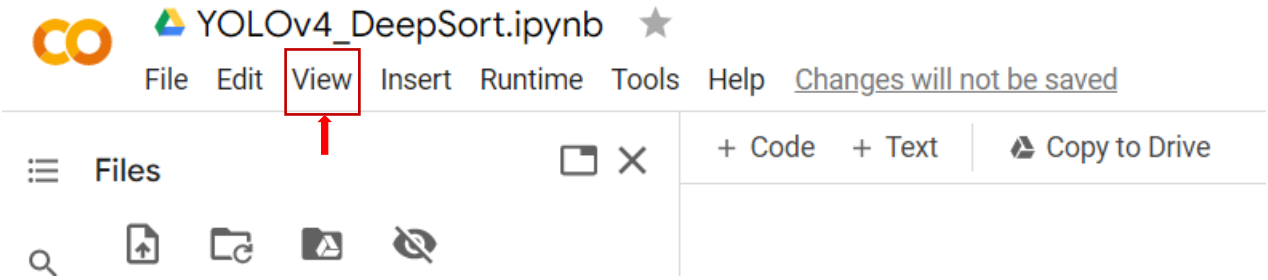
4.2 Set Up System for Object Detection Models

In this study Colab environment is used. The Google Collaboration is free service for the developers to try TensorFlow on CPU and GPU over cloud instance of Google. This service is totally free for improving Python programming skills. It can used by logging in with Google Gmail account and connect to this service. Here deep learning applications using popular machine learning libraries such as Keras, TensorFlow, PyTorch, OpenCV and others can be performed.

It requires to enable GPU acceleration within the Colab notebook so that the object tracker will be able to process detections over 100 times faster than if it were on CPU.

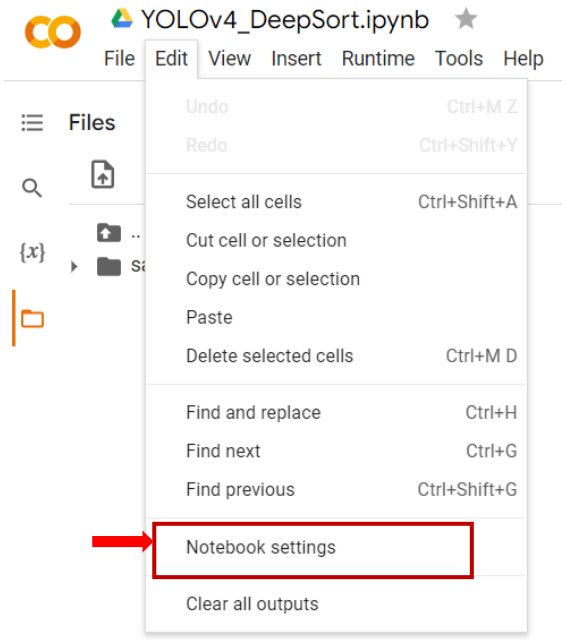
4.2.1 Step 1:

Click **Edit** at top left of the notebook.



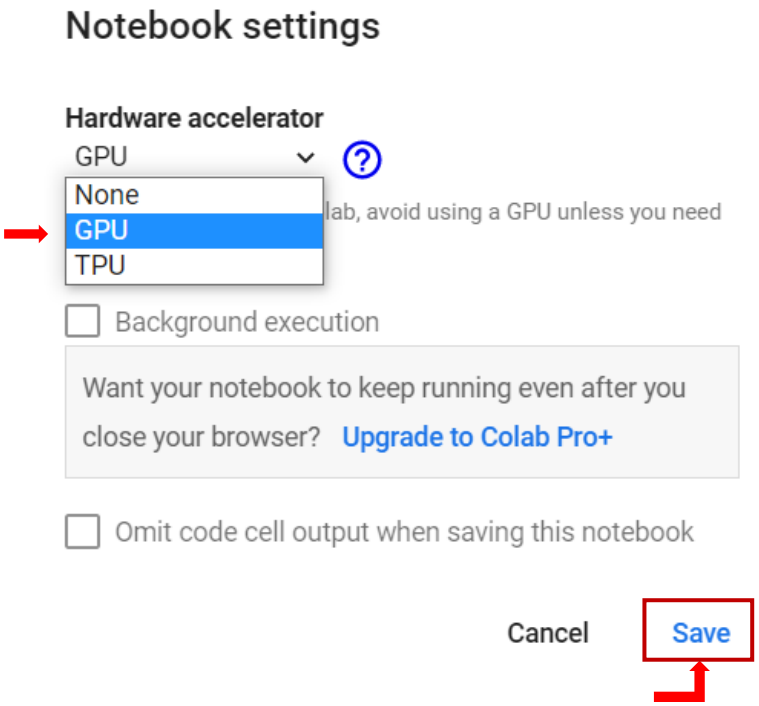
4.2.2 Step 2:

Click Notebook settings within dropdown.



4.2.3 Step 3:

Under 'Hardware Accelerator' select **GPU** and then hit **Save.**



4.3 Upload File from Local Machine

Upload code from the local machine which was written in VS Code using Abseil flags library. The Abseil flags library includes the ability to define flag types (boolean, float, integer, list), auto generation of help (in both human and machine-readable format) and reading arguments from a file. It also includes the ability to automatically generate manual pages from the help flags.

4.4 Vehicle Detection and Classification

Cloning of the yolov4-deepsort repository from github, to enable the rest of the process and grab the code. For vehicle detection and classification, YOLOv4 model is used. The YOLOv4 code was published by Alexey B.

Get YOLOv4 Pre-trained Weights

The pre-trained YOLOv4 model is used, trained on over 80 classes. Yolov4 model weights were downloaded to data folder.

```
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.weights -P data/

--2022-08-12 12:46:30--  https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov
Resolving github.com (github.com)... 192.30.255.112
Connecting to github.com (github.com)|192.30.255.112|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/75388965/ba4b6380-88
--2022-08-12 12:46:30--  https://objects.githubusercontent.com/github-production-release-asset-2e65be/753889
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.108.133, 185.199.109.133,
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.108.133|:443... connecte
HTTP request sent, awaiting response... 200 OK
Length: 257717640 (246M) [application/octet-stream]
Saving to: 'data/yolov4.weights'

yolov4.weights      100%[===================>] 245.78M   317MB/s    in 0.8s

2022-08-12 12:46:31 (317 MB/s) - 'data/yolov4.weights' saved [257717640/257717640]
```

Installed tf2_yolov4 + TensorFlow

Google is releasing the test versions of its upcoming TensorFlow 2.0 for testing and experimenting with it. TensorFlow 2.0 is highly upgraded version of TensorFlow and it comes with many new features and enhancement for development of next generation deep learning applications. I install TensorFlow 2.0 in Colab environment. I will be able to connect to the Colab environment for testing and running the sample applications. Google Colab provides web browser interface for running code.

```
!pip install --no-cache-dir https://github.com/sicara/tf2-yolov4/archive/master.zip
!pip install tensorflow==2.2.0

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/
Collecting https://github.com/sicara/tf2-yolov4/archive/master.zip
  Downloading https://github.com/sicara/tf2-yolov4/archive/master.zip
     \ 2.7 MB 4.6 MB/s
Requirement already satisfied: click>=6.7 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numpy>=1.10 in /usr/local/lib/python3.7/dist-packages
Collecting tensorflow-addons>=0.9.1
  Downloading tensorflow_addons-0.17.1-cp37-cp37m-manylinux_2_17_x86_64.manylinux201
     |████████████████████████████████| 1.1 MB 8.9 MB/s
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: typeguard>=2.7 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/
Building wheels for collected packages: tf2-yolov4
  Building wheel for tf2-yolov4 (setup.py) ... done
  Created wheel for tf2-yolov4: filename=tf2_yolov4-0.1.0-py3-none-any.whl size=2015
  Stored in directory: /tmp/pip-ephem-wheel-cache-nryw30df/wheels/56/ba/d4/430cf7011
Successfully built tf2-yolov4
Installing collected packages: tensorflow-addons, tf2-yolov4
Successfully installed tensorflow-addons-0.17.1 tf2-yolov4-0.1.0
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/
Collecting tensorflow==2.2.0
```

Convert YOLOv4 Darknet Weights to TensorFlow model

I will be running the DeepSort object tracker using TensorFlow. In order to accomplish this, we must first convert the yolov4 weights into a tensorflow model.

```
2022-08-12 12:48:07.109305: I tensorflow/stream_executor/platform/default/dso_loader.cc:44]
2022-08-12 12:48:07.176645: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:981] succ
2022-08-12 12:48:07.177302: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1561] Found d
pciBusID: 0000:00:04.0 name: Tesla T4 computeCapability: 7.5
coreClock: 1.59GHz coreCount: 40 deviceMemorySize: 14.75GiB deviceMemoryBandwidth: 298.08GiB
2022-08-12 12:48:07.191428: I tensorflow/stream_executor/platform/default/dso_loader.cc:44]
2022-08-12 12:48:07.392873: I tensorflow/stream_executor/platform/default/dso_loader.cc:44]
2022-08-12 12:48:07.427307: I tensorflow/stream_executor/platform/default/dso_loader.cc:44]
2022-08-12 12:48:07.445628: I tensorflow/stream_executor/platform/default/dso_loader.cc:44]
2022-08-12 12:48:07.659327: I tensorflow/stream_executor/platform/default/dso_loader.cc:44]
2022-08-12 12:48:07.681890: I tensorflow/stream_executor/platform/default/dso_loader.cc:44]
2022-08-12 12:48:08.074562: I tensorflow/stream_executor/platform/default/dso_loader.cc:44]
2022-08-12 12:48:08.074774: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:981] succ
2022-08-12 12:48:08.075756: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:981] succ
2022-08-12 12:48:08.076697: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1703] Adding
2022-08-12 12:48:08.078635: I tensorflow/core/platform/cpu_feature_guard.cc:143] Your CPU su
2022-08-12 12:48:08.089851: I tensorflow/core/platform/profile_utils/cpu_utils.cc:102] CPU F
2022-08-12 12:48:08.090102: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x
2022-08-12 12:48:08.090134: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecut
2022-08-12 12:48:08.378154: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:981] succ
2022-08-12 12:48:08.379171: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x
2022-08-12 12:48:08.379214: I tensorflow/compiler/xla/service/service.cc:176]   StreamExecut
2022-08-12 12:48:08.379463: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:981] succ
2022-08-12 12:48:08.380254: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1561] Found d
pciBusID: 0000:00:04.0 name: Tesla T4 computeCapability: 7.5
```

4.5 Vehicle Tracking and Counting

Unfortunately, Google Colab doesn't support displaying video while its being processed. So, it will be suppressing the output of the object tracker as it runs and then it will display the entire video once it has finished processing.

Then run DeepSort with YOLOv4 Object Detections as backbone (enable --info flag to see info about tracked objects).

Detailed steps to tracking and counting object with in bounding box:

1. Format bounding boxes from normalized ymin, xmin, ymax, xmax ---> xmin, ymin, width, height

```
150 |   original_h, original_w, _ = frame.shape
151 |   bboxes = utils.format_boxes(bboxes, original_h, original_w)
```

2. Store all predictions in one parameter for simplicity when calling functions

```
154 |   pred_bbox = [bboxes, scores, classes, num_objects]
```

3. Read in all class names from config.
4. Loop through objects and use class index to get class name, allow only classes in allowed_classes list
5. Delete detections that are not in allowed_classes

```
181         bboxes = np.delete(bboxes, deleted_indx, axis=0)
182         scores = np.delete(scores, deleted_indx, axis=0)
```

6. Encode yolo detections and feed to tracker
7. Initialize color map

```
189         cmap = plt.get_cmap('tab20b')
190         colors = [cmap(i)[:3] for i in np.linspace(0, 1, 20)]
```

8. Run non-maxima suppression

```
193         boxs = np.array([d.tlwh for d in detections])
194         scores = np.array([d.confidence for d in detections])
195         classes = np.array([d.class_name for d in detections])
196         indices = preprocessing.non_max_suppression(boxs, classes, nms_max_overlap, scores)
197         detections = [detections[i] for i in indices]
```

9. Call the tracker
10. Update tracks

```
204         for track in tracker.tracks:
205             if not track.is_confirmed() or track.time_since_update > 1:
206                 continue
207             bbox = track.to_tlbr()
208             class_name = track.get_class()
```

11. Draw BBox on screen
12. If enable info flag then print details about each track

```
if FLAGS.info:
    print("Tracker ID: {}, Class: {},  BBox Coords (xmin, ymin, xmax, ymax): {}".format(str(track.track_id), class_name, (int(bbox[0]), int(bbox[1]), int(bbox[2]), int(bbox[3]))))
```

13. Calculate frames per second of running detections

```
222         fps = 1.0 / (time.time() - start_time)
223         print("FPS: %.2f" % fps)
224         result = np.asarray(frame)
225         result = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
```

14. If output flag is set, save video file

```
231         if FLAGS.output:
232             out.write(result)
233         if cv2.waitKey(1) & 0xFF == ord('q'): break
234     cv2.destroyAllWindows()
```

Output of frame 6

```
Frame #:  6
Tracker ID: 1, Class: bus,  BBox Coords (xmin, ymin, xmax, ymax): (105, 268, 185, 359)
Tracker ID: 2, Class: car,  BBox Coords (xmin, ymin, xmax, ymax): (894, 330, 971, 375)
Tracker ID: 3, Class: person,  BBox Coords (xmin, ymin, xmax, ymax): (795, 545, 831, 597)
Tracker ID: 4, Class: car,  BBox Coords (xmin, ymin, xmax, ymax): (2, 477, 124, 597)
Tracker ID: 5, Class: car,  BBox Coords (xmin, ymin, xmax, ymax): (180, 318, 230, 368)
Tracker ID: 6, Class: person,  BBox Coords (xmin, ymin, xmax, ymax): (150, 335, 172, 390)
Tracker ID: 7, Class: bus,  BBox Coords (xmin, ymin, xmax, ymax): (373, 364, 514, 544)
Tracker ID: 8, Class: truck,  BBox Coords (xmin, ymin, xmax, ymax): (555, 374, 763, 583)
Tracker ID: 10, Class: person,  BBox Coords (xmin, ymin, xmax, ymax): (834, 454, 865, 543)
Tracker ID: 11, Class: truck,  BBox Coords (xmin, ymin, xmax, ymax): (189, 394, 265, 471)
Tracker ID: 12, Class: car,  BBox Coords (xmin, ymin, xmax, ymax): (244, 347, 386, 428)
Tracker ID: 13, Class: bus,  BBox Coords (xmin, ymin, xmax, ymax): (295, 265, 441, 354)
Tracker ID: 15, Class: bus,  BBox Coords (xmin, ymin, xmax, ymax): (632, 337, 753, 415)
Tracker ID: 16, Class: person,  BBox Coords (xmin, ymin, xmax, ymax): (-10, 241, 1050, 598)
Tracker ID: 18, Class: person,  BBox Coords (xmin, ymin, xmax, ymax): (918, 541, 949, 598)
Tracker ID: 20, Class: bus,  BBox Coords (xmin, ymin, xmax, ymax): (209, 255, 262, 309)
Tracker ID: 21, Class: truck,  BBox Coords (xmin, ymin, xmax, ymax): (510, 346, 664, 496)
Tracker ID: 22, Class: car,  BBox Coords (xmin, ymin, xmax, ymax): (189, 396, 266, 474)
Tracker ID: 25, Class: bus,  BBox Coords (xmin, ymin, xmax, ymax): (555, 374, 763, 583)
Tracker ID: 26, Class: person,  BBox Coords (xmin, ymin, xmax, ymax): (880, 483, 923, 585)
Tracker ID: 27, Class: bus,  BBox Coords (xmin, ymin, xmax, ymax): (288, 246, 345, 288)
FPS: 27.63
```

From the algorithm used in the study, the final output of a particular frame as shown above, for the case of frame#6, the detection processing time has come out to be 27 fps without any error. As per BCAS (Bureau of Civil Aviation Security) the standard CCTV footage frame rate is 25 fps. Which in return the algorithm is able to detect the traffic with lesser processing time to provide faster results without any delay.

After detection it will save as avi format, Then I convert from avi to mp4 format and I defined helper function to display videos.

```
import io
from IPython.display import HTML
from base64 import b64encode
def show_video(file_name, width=640):
  # show resulting deepsort video
  mp4 = open(file_name,'rb').read()
  data_url = "data:video/mp4;base64," + b64encode(mp4).decode()
  return HTML("""
  <video width="{0}" controls>
        <source src="{1}" type="video/mp4">
  </video>
  """.format(width, data_url))
```

Then convert resulting video from avi to mp4 file format.

```
import os
path_video = os.path.join("outputs","tracker.avi")
%cd outputs/
!ffmpeg -y -loglevel panic -i tracker.avi output.mp4
%cd ..

# output object tracking video
path_output = os.path.join("outputs","output.mp4")
show_video(path_output, width=960)
```

4.6 Experimental Results and Analysis

In this chapter, I provided the steps for the implementation of a traffic surveillance application to analyse and extract information from images. For each object detected coordinates of the bounding boxes were taken and a class was predicted according to its features.

In this work, 6 sample traffic video clips from difference location in different times of the day (day and night) are used in the experiments. The video was recorded in Kolkata, India under two difference weather condition, Under clear-sky condition and another during rainfall

After successfully validating the proposed model on images from datasets, the proposed model was subjected to testing of the traffic congestion analysis from a footbridge near the Lake Town in Kolkata, as shown in Figure 4.2. It was noticed due to free-flowing traffic. In order to assess the performance of the proposed model under peak traffic conditions, similar experiments were carried out on several other footbridge in Kolkata too, wherein the traffic congestion was higher. During these experiments, the capture video was fed to the laptop, and the traffic congestion percentage was displayed.

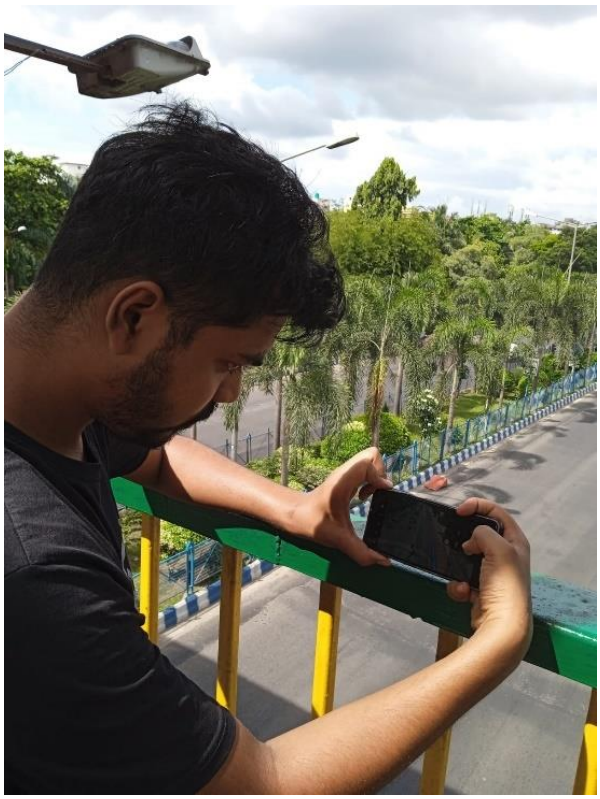


Figure 4.2: Video capturing moment

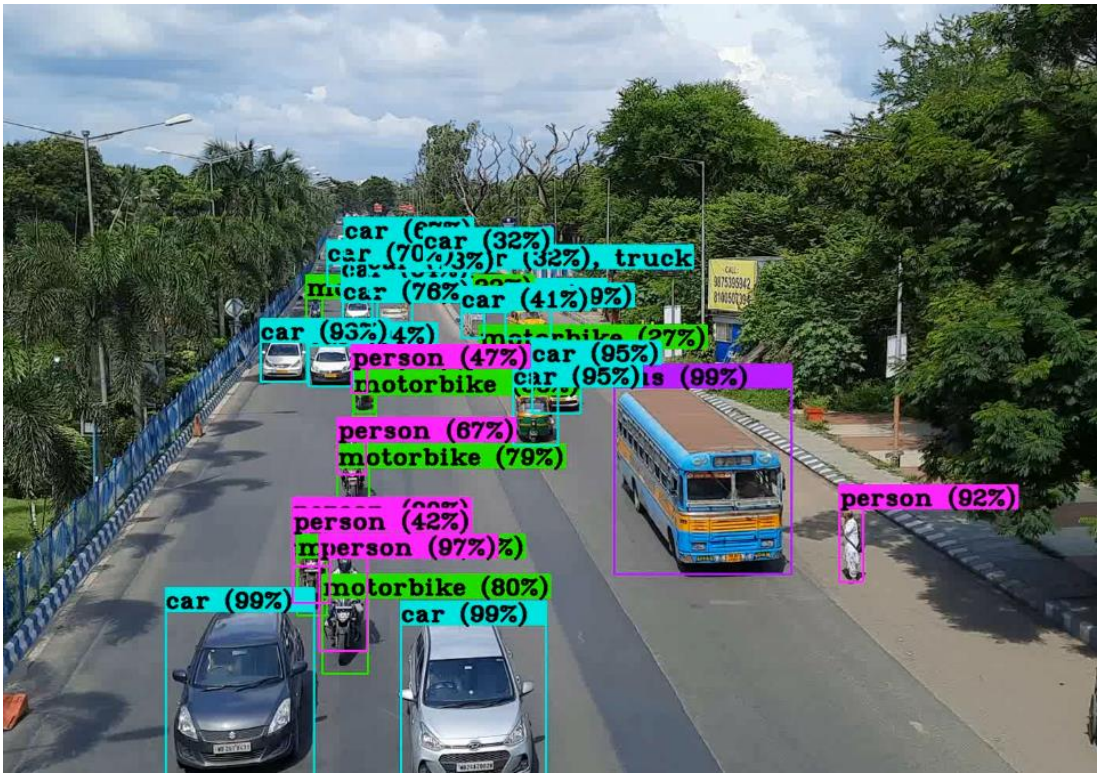


Figure 4.3: Traffic during 8am in the morning at Lake town, Kolkata

```
Tracker ID: 1, Class: person,  BBox Coords (xmin, ymin, xmax, ymax): (242, 789, 362, 1080)
Tracker ID: 2, Class: person,  BBox Coords (xmin, ymin, xmax, ymax): (717, 223, 783, 409)
Tracker ID: 3, Class: person,  BBox Coords (xmin, ymin, xmax, ymax): (1605, 611, 1715, 864)
Tracker ID: 4, Class: person,  BBox Coords (xmin, ymin, xmax, ymax): (882, 73, 947, 228)
Tracker ID: 5, Class: person,  BBox Coords (xmin, ymin, xmax, ymax): (803, 239, 858, 417)
Tracker ID: 6, Class: person,  BBox Coords (xmin, ymin, xmax, ymax): (861, 390, 984, 593)
Tracker ID: 7, Class: person,  BBox Coords (xmin, ymin, xmax, ymax): (1646, 118, 1715, 279)
Tracker ID: 8, Class: bicycle,  BBox Coords (xmin, ymin, xmax, ymax): (1350, 70, 1402, 147)
Tracker ID: 9, Class: person,  BBox Coords (xmin, ymin, xmax, ymax): (280, 291, 372, 483)
Tracker ID: 10, Class: person,  BBox Coords (xmin, ymin, xmax, ymax): (1452, 27, 1533, 185)
Tracker ID: 11, Class: person,  BBox Coords (xmin, ymin, xmax, ymax): (1573, 918, 1708, 1080)
Tracker ID: 12, Class: person,  BBox Coords (xmin, ymin, xmax, ymax): (1801, 176, 1867, 342)
Tracker ID: 13, Class: bicycle,  BBox Coords (xmin, ymin, xmax, ymax): (1508, 89, 1547, 188)
Tracker ID: 14, Class: person,  BBox Coords (xmin, ymin, xmax, ymax): (606, 1002, 802, 1080)
Tracker ID: 15, Class: person,  BBox Coords (xmin, ymin, xmax, ymax): (1154, 0, 1194, 75)
Tracker ID: 16, Class: bicycle,  BBox Coords (xmin, ymin, xmax, ymax): (879, 488, 1016, 641)
```

```
FPS: 13.81
Frame #:  510
FPS: 15.48
Frame #:  511
FPS: 14.39
Frame #:  512
FPS: 14.19
Frame #:  513
FPS: 18.14
Frame #:  514
FPS: 16.00
Frame #:  515
FPS: 17.84
Frame #:  516
FPS: 16.17
Frame #:  517
```

```
car: 98%
car: 98%
car: 97%
car: 96%
car: 87%
car: 78%
car: 73%
car: 57%
car: 55%
car: 54%
car: 54%
car: 54%
car: 51%
car: 28%
```

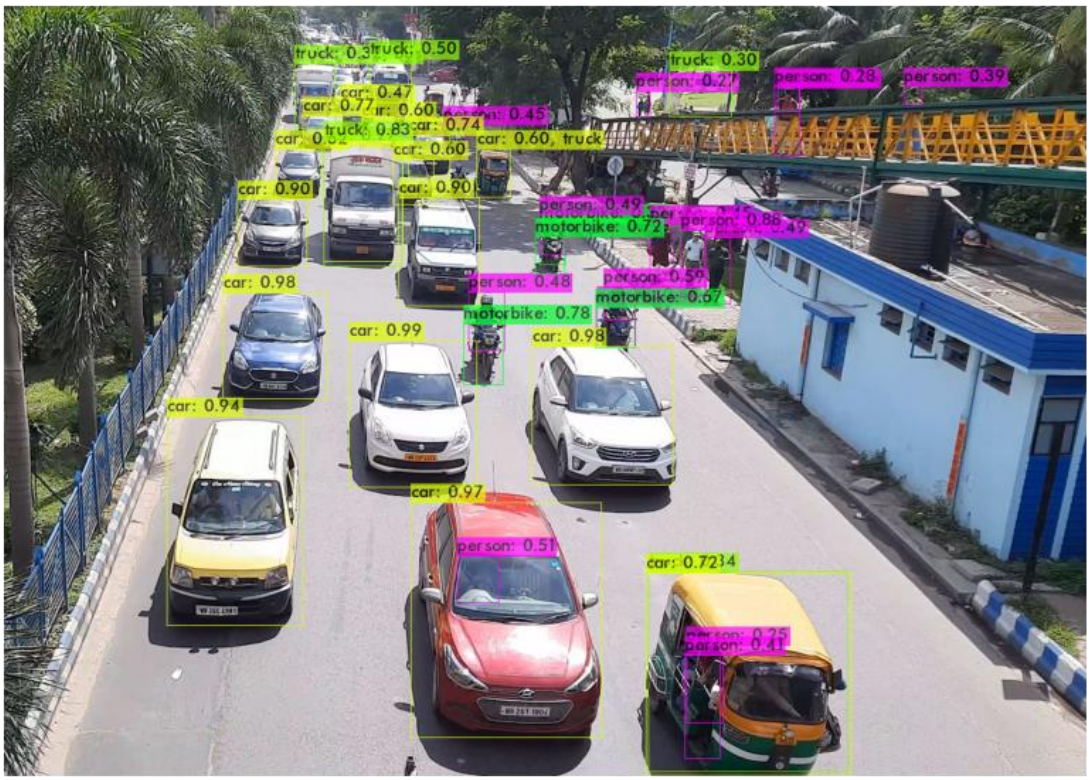


Figure 4.4: Traffic during 2pm in the afternoon at Lake town, Kolkata

```
Frame #:  1
FPS: 0.35
Frame #:  2
FPS: 16.29
Frame #:  3
FPS: 14.47
Frame #:  4
FPS: 13.71
Frame #:  5
FPS: 14.00
Frame #:  6
FPS: 12.35
Frame #:  7
FPS: 14.14
Frame #:  8
FPS: 13.44
Frame #:  9
FPS: 13.56
Frame #:  10
FPS: 12.57
Frame #:  11
FPS: 12.57
```

```
 cvWriteFrame
Objects:

car: 97%
car: 96%
car: 95%
car: 95%
car: 91%
car: 81%
car: 76%
car: 74%
car: 64%
car: 64%
car: 43%
car: 32%
Stream closed.
car: 31%
car: 30%

FPS:28.2
```

Figure 4.5: Traffic during 7pm in the evening at Chingrihata, Kolkata

Object tracking report of Figure 4.5

```
FPS: 15.88
Frame #:  479
FPS: 17.22
Frame #:  480
FPS: 15.71
Frame #:  481
FPS: 17.57
Frame #:  482
FPS: 18.62
Frame #:  483
FPS: 16.04
Frame #:  484
FPS: 15.86
Frame #:  485
FPS: 16.33
Frame #:  486
FPS: 16.63
Frame #:  487
FPS: 17.04
Frame #:  488
FPS: 15.36
Frame #:  489
FPS: 17.92
Frame #:  490
FPS: 15.92
```

```
car: 91%
car: 97%
car: 61%
car: 93%
traffic light: 30%
car: 90%
car: 53%
car: 30%
car: 52%
car: 91%
car: 47%
truck: 26%
car: 84%
car: 98%
car: 98%
car: 29%
car: 97%
car: 43%
car: 95%
car: 89%
car: 89%
car: 53%
car: 98%
car: 99%
```

Performance evaluation of proposed model under lousy weather and low illumination conditions,

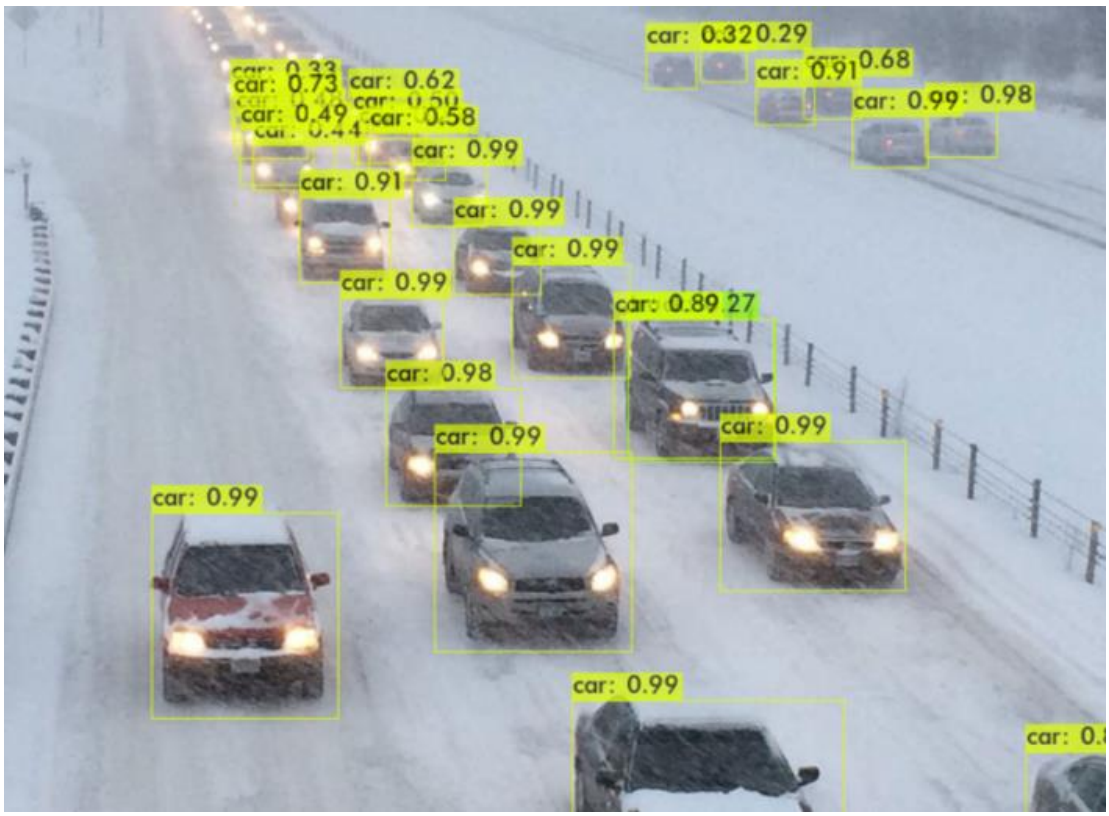


Figure 4.6*:* Vehicle detection during snowfall

```
Frame #:  253
FPS: 15.71
Frame #:  254
FPS: 12.14
Frame #:  255
FPS: 13.77
Frame #:  256
FPS: 16.03
Frame #:  257
FPS: 15.02
Frame #:  258
FPS: 20.38
Frame #:  259
FPS: 18.99
Frame #:  260
FPS: 15.73
Frame #:  261
FPS: 19.66
Frame #:  262
FPS: 19.36
Frame #:  263
FPS: 19.91
Frame #:  264
FPS: 19.72
```

```
car: 90%
car: 53%
car: 30%
car: 52%
car: 91%
car: 47%
truck: 26%
car: 84%
car: 98%
car: 98%
car: 29%
car: 97%
car: 43%
car: 95%
car: 89%
car: 89%
car: 53%
car: 98%
car: 99%
```

Figure 4.7: During rainfall at Kolkata

```
 CUDA-version: 11010 (11020), cuDNN: 7.6.5, GPU count: 1
 OpenCV version: 3.2.0
 0 : compute_capability = 750, cudnn_half = 0, GPU: Tesla T4
net.optimized_memory = 0
mini_batch = 1, batch = 1, time_steps = 1, train = 0
   layer   filters  size/strd(dil)      input                output
   0 Create CUDA-stream - 0
 Create cudnn-handle 0
conv     32       3 x 3/ 1    416 x 416 x   3 ->  416 x 416 x  32 0.299 BF
   1 conv     64       3 x 3/ 2    416 x 416 x  32 ->  208 x 208 x  64 1.595 BF
   2 conv     32       1 x 1/ 1    208 x 208 x  64 ->  208 x 208 x  32 0.177 BF
   3 conv     64       3 x 3/ 1    208 x 208 x  32 ->  208 x 208 x  64 1.595 BF
   4 Shortcut Layer: 1,  wt = 0, wn = 0, outputs: 208 x 208 x  64 0.003 BF
   5 conv    128       3 x 3/ 2    208 x 208 x  64 ->  104 x 104 x 128 1.595 BF
   6 conv     64       1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x  64 0.177 BF
   7 conv    128       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x 128 1.595 BF
   8 Shortcut Layer: 5,  wt = 0, wn = 0, outputs: 104 x 104 x 128 0.001 BF
   9 conv     64       1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x  64 0.177 BF
  10 conv    128       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x 128 1.595 BF
  11 Shortcut Layer: 8,  wt = 0, wn = 0, outputs: 104 x 104 x 128 0.001 BF
  12 conv    256       3 x 3/ 2    104 x 104 x 128 ->   52 x  52 x 256 1.595 BF
  13 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  14 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  15 Shortcut Layer: 12,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  16 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  17 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  18 Shortcut Layer: 15,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  19 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  20 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  21 Shortcut Layer: 18,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
```

```
Frame #:  16
FPS: 21.10
Frame #:  17
FPS: 20.01
Frame #:  18
FPS: 20.45
Frame #:  19
FPS: 20.00
Frame #:  20
FPS: 20.80
Frame #:  21
FPS: 20.34
Frame #:  22
FPS: 21.42
Frame #:  23
FPS: 19.38
Frame #:  24
FPS: 19.00
Frame #:  25
FPS: 20.13
Frame #:  26
FPS: 20.57
```

```
data/in_kol_1.jpg: Predicted in 40.933000 milli-seconds.
umbrella: 77%
car: 71%
person: 51%
person: 99%
person: 99%
bus: 100%
person: 99%
backpack: 36%
person: 95%
bus: 100%
umbrella: 93%
person: 92%
person: 85%
person: 34%
```

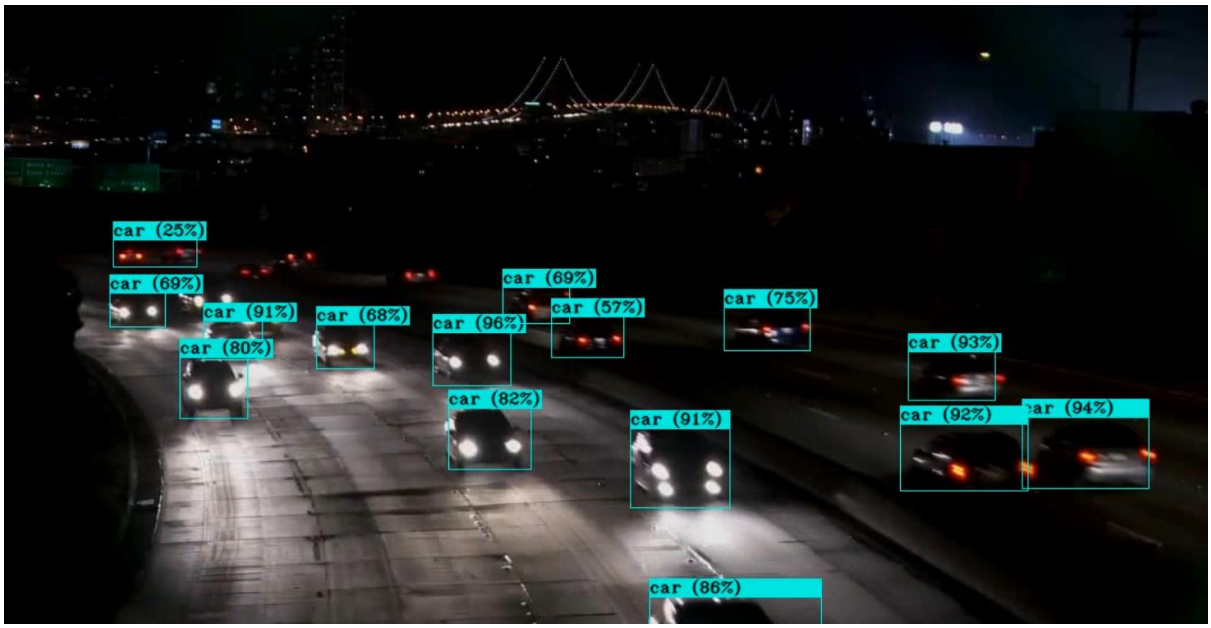
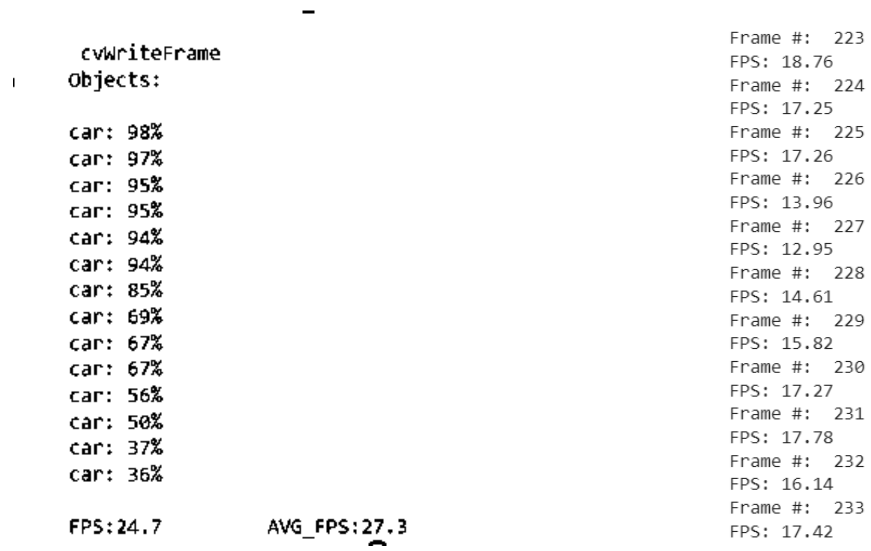


Figure 4.8: Vehicle detection at night where no street light present



In order to assess the performance of the proposed model for other weather conditions, few images were collected online, as I couldn't get real-time images/videos during snowfall is impossible in this geographical region. The proposed model for rainy, snowfall and at midnight results is shown in Figure 4.6, Figure 4.7, Figure 4.8, wherein it can be observed that the performance has little degraded due to the bad weather conditions, but it can also be noted that the vehicles do get detected correctly once they arrive closer to the camera. It can be concluded that the performance varies concerning the intensity of rainfall or snowfall and also at midnight.

Table 4.1: Accuracy of vehicle detection in low illumination conditions

Conditions	**Accuracy**
Rainy time	66.66%
At midnight	75%
During snowfall	73.33%

Final comparison of all the five experimental data as shown in Figure 4.3-Figure 4.8.

Table 4.2: Cumulative Results of all Experimental Data

Sl No.	Figure reference	Illumination and Weather condition	Frames per second (fps)	Maximum Accuracy
1.	**Figure 4.3**	Morning	18.14	98%
2.	**Figure 4.4**	Afternoon	16.29	97%
3.	**Figure 4.5**	Evening	18.62	99%
4.	**Figure 4.6**	Snowfall	19.72	99%
5.	**Figure 4.7**	Rainfall	20.57	100%
6.	**Figure 4.8**	Night	27.3	98%

In this work, vehicle accuracy counting is used to evaluate the performance of each detection model. It is determined by counting vehicles image frames. The vehicle counting accuracy (VCA) is computed in equation as follows:

$$\text{VCA\%} = \frac{\text{Number of Detected V ehicles}}{\text{Total Number of V ehicles}} \times 100$$

Table 4.3: The overall accuracy of vehicle detection

Video Time	**YOLOv4**	**YOLOv3**
6am -11am	32%-99.91%	2.8%-83.93%
12pm-2pm	99.00%	96.32%
4pm – 9pm	26%-99%	2.47%-74.02%

During the evaluation of the proposed model, it was observed that the network size of the YOLO v3 and YOLO v4 Darknet architecture for object detection could be modified based on the size of the input video resolution. Experiments were carried out to find the best network size for the proposed application.

Table 4.4: Average Counting Accuracy and Processing Time

Model	Counting Accuracy (%)	Processing Time (Frame per second)
YOLOv4	90.10 ± 8.50	0.27 ± 0.010
Tiny-yolov3	66.29 ± 33.35	0.26 ± 0.013

Chapter 5 Conclusion

The model is able to determine what the object is but also where the given object resides. It also counts the objects of a pericellular class and displace them accordingly. Figures in Section 4.6 shows the vehicles detections, tracked, counts made by the YOLOV4 model on a sample traffic scene.

This study explores as aspect of building a robust transportation management system that can handle the challenges that plague transportation in India and indeed other countries of the world. It lays a good foundation for fixing a key component crucial in the development of a robust solution: the vehicle counting system. The vehicle counting system is an invaluable tool in transportation management agencies and systems. It provides information about the flow of vehicular traffic along a given road which is useful in optimizing transportation and planning for the future, so that people and goods can be conveyed from one place to the other faster and resources can be used efficiently. During the implementation of this system, several detection algorithms were experimented with. The YOLO deep learning algorithm was the detector of choice for the vehicle counting system as it produced the most accurate results while being relatively fast.

While existing vehicle counting systems such as the manual counting done and the pneumatic road tubes used by FRSC provide a manageable solution, their limitations make handling the traffic challenges of today very difficult. This study proposes a video-based vehicle counting solution which solves some of the limitations of existing systems such as increased accuracy, flexibility and scalability. There are improvements to be made to make the system faster, more accurate and cheaper to operate. As new computer vision techniques and algorithms are developed, the video-based vehicle counting system would improve accordingly and eventually become the standard for collecting and processing traffic flow data.

5.1 Limitation of the study

The limitations and drawbacks of the YOLO object detector, are described as below.

5.1.1 Limitations and drawbacks of the YOLO object detector

Arguably the largest limitation and drawback of the YOLO object detector is that:

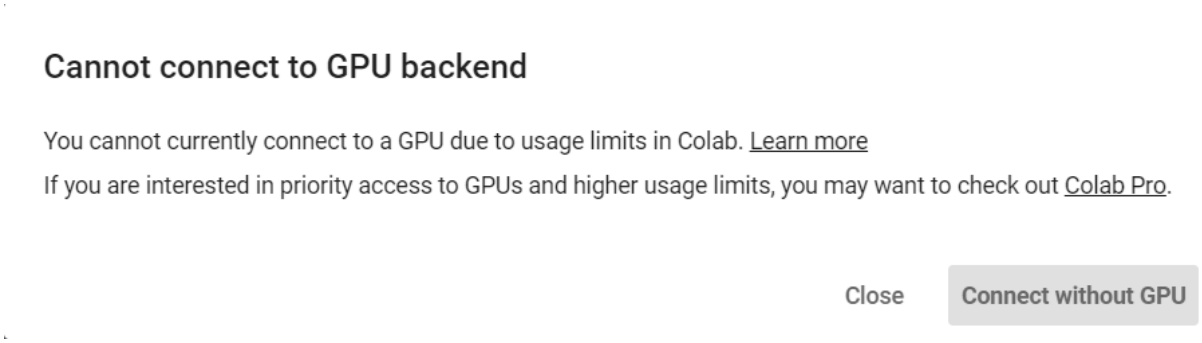
- It does not always handle small objects well.
- It especially does not handle objects grouped close together.

The reason for this limitation is due to the YOLO algorithm itself:

- The YOLO object detector divides an input image into an SxS grid where each cell in the grid predicts only a single object.
- If there exist multiple, small objects in a single cell then YOLO will be unable to detect them, ultimately leading to missed object detections.

5.2 Drawbacks of the Google Colab

In this study Google Colab is used. There are some problems that that was encounter while training a Computer Vision Model in Google Colab.



5.3 Future Work

During the dissertation work a number of challenges appeared that would be interesting to address.

- **Create a specific vehicle dataset to train deep learning frameworks:** If deep learning models for real traffic application is used it is essential to have large datasets with accurate annotations. It would be interesting to use semi-supervised or supervised techniques to create these datasets as there is a lack of specific training datasets for vehicle detection in traffic environments.

- **Dynamic detection of the tracking zones:** In this work the traffic cameras are used in a fixed position but in real life, cameras can be dynamic. The automatic detection of the tracking zones would be an interesting addition to the system.

- **Train YOLOv4 on a specific vehicle dataset with different classes:** It would be interesting to train YOLOv4 on a specially crafted dataset in images of vehicles of different classes for the context of real traffic applications as the YOLOv4 shown promising results in doing these tasks.

Chapter 6 References

[1] Adewumi, A., Odunjo, V., & Misra, S. (2015, December). Developing a mobile application for taxi Service Company in Nigeria. In 2015 International Conference on Computing, Communication and Security (ICCCS) (pp. 1-5). IEEE.

[2] Azizi, Abdullah, and Jaison Oothariasamy. "Vehicle Counting using Deep Learning Models: A Comparative Study." International Journal of Advanced Computer Science and Applications 11.7 (2020).

[3] Bush, I.J., Dimililer, K.: Static and dynamic pedestrian detection algorithm for visual based driver assistive system. J. ITM Web Conf. 9, 03002 (2017). EDP Sciences

[4] C. -J. Lin and J. -Y. Jhang, "Intelligent Traffic-Monitoring System Based on YOLO and Convolutional Fuzzy Neural Networks," in IEEE Access, vol. 10, pp. 14120-14133, 2022, doi: 10.1109/ACCESS.2022.3147866.

[5] Dimililer, K., Ever, Y.K., Mustafa, S.M. (2020). Vehicle Detection and Tracking Using Machine Learning Techniques. In: Aliev, R., Kacprzyk, J., Pedrycz, W., Jamshidi, M., Babanli, M., Sadikoglu, F. (eds) 10th International Conference on Theory and Application of Soft Computing, Computing with Words and Perceptions - ICSCCW-2019. ICSCCW 2019. Advances in Intelligent Systems and Computing, vol 1095. Springer, Cham. https://doi.org/10.1007/978-3-030-35249-3_48

[6] H. Shin, K. Chung and C. Park, "Prediction of Traffic Congestion Based on LSTM Through Correction of Missing Temporal and Spatial Data", IEEE Access, vol. 8, pp. 150784-150796, August 2020.

[7] Huaizhong Zhang, Mark Liptrott, Nik Bessis and Jianquan Cheng, "Realtime traffic analysis using deep learning techniques and uav based video", 2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), pp. 1-5, 2019.

[8] J. Lin and M. Sun, "A YOLO-Based Traffic Counting System," 2018 Conference on Technologies and Applications of Artificial Intelligence (TAAI), 2018, pp. 82-85, doi: 10.1109/TAAI.2018.00027.

[9] K. Sakai, T. Seo and T. Fuse, "Traffic density estimation method from small satellite imagery: Towards frequent remote sensing of car traffic", Int. Transp. Systems Conf., pp. 1776-1781, Oct 2019.

[10] Lin, Cheng-Jian, and Jyun-Yu Jhang. "Intelligent Traffic-Monitoring System Based on YOLO and Convolutional Fuzzy Neural Networks." IEEE Access 10 (2022): 14120-14133.

[11] Liu, Chieh-Min, and Jyh-Ching Juang. 2021. "Estimation of Lane-Level Traffic Flow Using a Deep Learning Technique" Applied Sciences 11, no. 12: 5619. https://doi.org/10.3390/app11125619

[12] Mandal, Vishal, Abdul Rashid Mussah, Peng Jin, and Yaw Adu-Gyamfi. 2020. "Artificial Intelligence-Enabled Traffic Monitoring System" Sustainability 12, no. 21: 9177. https://doi.org/10.3390/su12219177

[13] Mayank Singh Chauhan, Arshdeep Singh, Mansi Khemka, Arneish Prateek and Rijurekha Sen, "Embedded CNN based vehicle classification and counting in non-laned road traffic", Int. Conf. on Inf. and Comm. Techn. and development, pp. 1-11, Jan 2019.

[14] Mohammed A.A. Al-Qaness, Mohamed Abd Elaziz, Ammar Hawbani, Aaqif Afzaal Abbasi, Liang Zhao and Sunghwan Kim, Real-Time Traffic Congestion Analysis Based on Collected Tweets. Int. Conf. on Ubiq. Comp. & Comm. and Data Sci. and Comp. Intell. and Smart Comp. Netw. and Services, pp. 1-8, Oct 2019.

[15] Nere, Darshan & Kabara, Neilkrishna & Patil, Nirmal & Niturkar, Gaurav & Patil, Devyani. (2022). ATMOS: Advanced Traffic Management and Optimizing System. 10.9790/0661-2401025559.

[16] Nilani et al., "Towards Real-time traffic flow estimation using YOLO and SORT from Surveillance Video Footage", 18th Int. Conf. on Inf. Sys. for Crisis Resp. and Manag., pp. 40-48, May 2021.

[17] Nilani et al., "Towards Real-time traffic flow estimation using YOLO and SORT from Surveillance Video Footage", 18th Int. Conf. on Inf. Sys. for Crisis Resp. and Manag., pp. 40-48, May 2021.

[18] Noh, S., Shim, D., Jeon, M.: Adaptive sliding-window strategy for vehicle detection in highway environments. IEEE Trans. Intell. Transpt. Syst. 17, 323–335 (2015)

[19] O. Olayode, K. Tartibu and O. Okwu, "Traffic flow Prediction at Signalized Road Intersections: A case of Markov Chain and Artificial Neural Network Model", Int. Conf. on Mech. and Intell. Manuf. Techn. Cape Town, pp. 287-292, May 2021

[20] Osebor, I., Misra, S., Omoregbe, N., Adewumi, A., & Fernandez-Sanz, L. (2017). Experimental simulationbased performance evaluation of an SMS-based emergency geolocation notification system. Journal of healthcare engineering, 2017.

[21] P. Naresh, T. Tallam and N. Kumar, "Analysis of Urban Traffic Bottleneck in Hyderabad city using Machine Learning Techniques", Fourth Int. Conf. on IoT in Soc. Mob. Analy. and Cloud Palladam, pp. 756-762, Oct 2020

[22] R Rangel, Héctor, Luis A M, R I Rojo, Mario A R G, Gloria E P Peñuñuri, and Mariana L B. 2022. "Analysis of Statistical and Artificial Intelligence Algorithms for Real-Time Speed Estimation Based on Vehicle Detection with YOLO" Applied Sciences 12, no. 6: 2907. https://doi.org/10.3390/app12062907

[23] R, J. U, Y. H, A. A and M. J, "Design and Development of a Real-Time Traffic Congestion Analysis System," 2022 International Conference on Computer Science and Software Engineering (CSASE), 2022, pp. 230-235, doi: 10.1109/CSASE51777.2022.9759708.

[24] Real-time vehicle counting in complex scene for traffic flow estimation using multi-level convolutional neural network. Kadim, Zulaikha; K Mohd. Johari; Fairol, Den; Yuen Shang Li; Hock W H. International Journal of Advanced Technology and Engineering Exploration; Bhopal Vol. 8, Iss. 75, (Feb 2021): 338-351.

[25] Song, H., Liang, H., Li, H. et al. Vision-based vehicle detection and counting system using deep learning in highway scenes. Eur. Transp. Res. Rev. 11, 51 (2019). https://doi.org/10.1186/s12544-019-0390-4

[26] Tamrakar, Yunik, "Empirical Evaluation of Vehicle Detection, Tracking And Recognition Algorithms Operating On Real Time Video Feeds" (2020). Honors Theses. 1507. https://egrove.olemiss.edu/hon_thesis/1507

[27] The AI GUY, Yolov4_objectdetection Github,

[28] Vlahija, C., & Abdulkader, A. (2020). Real-time vehicle and pedestrian detection, a data-driven recommendation focusing on safety as a perception to autonomous vehicles (Dissertation, Malmö universitet/Teknik och samhälle). Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:mau:diva-20089

Chapter 7 Appendix

7.1 Resource Specifications

7.1.1 Hardware requirements

The following GPU-enabled devices are supported:

- NVIDIA GPU card with CUDA architectures 3.5, 5.0, 6.0, 7.0, 7.5, 8.0 and higher.
- For GPUs with unsupported CUDA architectures, or to avoid JIT compilation from PTX, or to use different versions of the NVIDIA libraries.
- Packages do not contain PTX code except for the latest supported CUDA architecture; therefore, TensorFlow fails to load on older GPUs.

7.1.2 System requirements

- Ubuntu 16.04 or higher (64-bit)
- macOS 10.12.6 (Sierra) or higher (64-bit) (no GPU support)
- Windows 7 or higher (64-bit)

7.1.3 Software requirements

- Python 3.7 – 3.10
- pip version 19.0 or higher for Linux (requires manylinux2010 support) and Windows, version 20.3 or higher for macOS

NOTE: - NVIDIA software is only required for GPU support.

7.2 Program Code

7.2.1 Code 1

```
import os
# comment out below line to enable tensorflow logging outputs
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import time
import tensorflow as tf
physical_devices = tf.config.experimental.list_physical_devices('GPU')
if len(physical_devices) > 0:
    tf.config.experimental.set_memory_growth(physical_devices[0], True)
from absl import app, flags, logging
from absl.flags import FLAGS
import core.utils as utils
from core.yolov4 import filter_boxes
from tensorflow.python.saved_model import tag_constants
from core.config import cfg
from PIL import Image
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession


from deep_sort import preprocessing, nn_matching
from deep_sort.detection import Detection
from deep_sort.tracker import Tracker
from tools import generate_detections as gdet
flags.DEFINE_string('framework', 'tf', '(tf, tflite, trt')
flags.DEFINE_string('weights', './checkpoints/yolov4-416',
                    'path to weights file')
flags.DEFINE_integer('size', 416, 'resize images to')
flags.DEFINE_boolean('tiny', False, 'yolo or yolo-tiny')
flags.DEFINE_string('model', 'yolov4', 'yolov3 or yolov4')
flags.DEFINE_string('video', './data/video/test.mp4', 'path to input video or set to 0 for webcam')
flags.DEFINE_string('output', None, 'path to output video')
flags.DEFINE_string('output_format', 'XVID', 'codec used in VideoWriter when saving video to file')
flags.DEFINE_float('iou', 0.45, 'iou threshold')
flags.DEFINE_float('score', 0.50, 'score threshold')
flags.DEFINE_boolean('dont_show', False, 'dont show video output')
flags.DEFINE_boolean('info', False, 'show detailed info of tracked objects')
flags.DEFINE_boolean('count', False, 'count objects being tracked on screen')

```

```
41  def main(_argv):
42      # Definition of the parameters
43      max_cosine_distance = 0.4
44      nn_budget = None
45      nms_max_overlap = 1.0
46
47      # initialize deep sort
48      model_filename = 'model_data/mars-small128.pb'
49      encoder = gdet.create_box_encoder(model_filename, batch_size=1)
50      # calculate cosine distance metric
51      metric = nn_matching.NearestNeighborDistanceMetric("cosine", max_cosine_distance, nn_budget)
52      # initialize tracker
53      tracker = Tracker(metric)
54
55      # load configuration for object detector
56      config = ConfigProto()
57      config.gpu_options.allow_growth = True
58      session = InteractiveSession(config=config)
59      STRIDES, ANCHORS, NUM_CLASS, XYSCALE = utils.load_config(FLAGS)
60      input_size = FLAGS.size
61      video_path = FLAGS.video


64      if FLAGS.framework == 'tflite':
65          interpreter = tf.lite.Interpreter(model_path=FLAGS.weights)
66          interpreter.allocate_tensors()
67          input_details = interpreter.get_input_details()
68          output_details = interpreter.get_output_details()
69          print(input_details)
70          print(output_details)
71      # otherwise load standard tensorflow saved model
72      else:
73          saved_model_loaded = tf.saved_model.load(FLAGS.weights, tags=[tag_constants.SERVING])
74          infer = saved_model_loaded.signatures['serving_default']
75
76      # begin video capture
77      try:
78          vid = cv2.VideoCapture(int(video_path))
79      except:
80          vid = cv2.VideoCapture(video_path)
81
82      out = None
```

```
    if FLAGS.output:
        # by default VideoCapture returns float instead of int
        width = int(vid.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = int(vid.get(cv2.CAP_PROP_FPS))
        codec = cv2.VideoWriter_fourcc(*FLAGS.output_format)
        out = cv2.VideoWriter(FLAGS.output, codec, fps, (width, height))

    frame_num = 0
    # while video is running
    while True:
        return_value, frame = vid.read()
        if return_value:
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            image = Image.fromarray(frame)
        else:
            print('Video has ended or failed, try a different video format!')
            break
        frame_num +=1
        print('Frame #: ', frame_num)
        frame_size = frame.shape[:2]
        image_data = cv2.resize(frame, (input_size, input_size))
        image_data = image_data / 255.
        image_data = image_data[np.newaxis, ...].astype(np.float32)
        start_time = time.time()


     if FLAGS.framework == 'tflite':
         interpreter.set_tensor(input_details[0]['index'], image_data)
         interpreter.invoke()
         pred = [interpreter.get_tensor(output_details[i]['index']) for i in range(len(output_details))]
         # run detections using yolov3 if flag is set
         if FLAGS.model == 'yolov3' and FLAGS.tiny == True:
             boxes, pred_conf = filter_boxes(pred[1], pred[0], score_threshold=0.25,
                                             input_shape=tf.constant([input_size, input_size]))
         else:
             boxes, pred_conf = filter_boxes(pred[0], pred[1], score_threshold=0.25,
                                             input_shape=tf.constant([input_size, input_size]))
     else:
         batch_data = tf.constant(image_data)
         pred_bbox = infer(batch_data)
         for key, value in pred_bbox.items():
             boxes = value[:, :, 0:4]
             pred_conf = value[:, :, 4:]

     boxes, scores, classes, valid_detections = tf.image.combined_non_max_suppression(
         boxes=tf.reshape(boxes, (tf.shape(boxes)[0], -1, 1, 4)),
         scores=tf.reshape(
             pred_conf, (tf.shape(pred_conf)[0], -1, tf.shape(pred_conf)[-1])),
         max_output_size_per_class=50,
         max_total_size=50,
         iou_threshold=FLAGS.iou,
         score_threshold=FLAGS.score
     )

```

```
            # convert data to numpy arrays and slice out unused elements
            num_objects = valid_detections.numpy()[0]
            bboxes = boxes.numpy()[0]
            bboxes = bboxes[0:int(num_objects)]
            scores = scores.numpy()[0]
            scores = scores[0:int(num_objects)]
            classes = classes.numpy()[0]
            classes = classes[0:int(num_objects)]

            # format bounding boxes from normalized ymin, xmin, ymax, xmax ---> xmin, ymin, width, height
            original_h, original_w, _ = frame.shape
            bboxes = utils.format_boxes(bboxes, original_h, original_w)

            # store all predictions in one parameter for simplicity when calling functions
            pred_bbox = [bboxes, scores, classes, num_objects]

            # read in all class names from config
            class_names = utils.read_class_names(cfg.YOLO.CLASSES)

            # by default allow all classes in .names file
            allowed_classes = list(class_names.values())
```

```
        names = []
        deleted_indx = []
        for i in range(num_objects):
            class_indx = int(classes[i])
            class_name = class_names[class_indx]
            if class_name not in allowed_classes:
                deleted_indx.append(i)
            else:
                names.append(class_name)
        names = np.array(names)
        count = len(names)
        if FLAGS.count:
            cv2.putText(frame, "Objects being tracked: {}".format(count), (5, 35), cv2.FONT_HERSHEY_COMPLEX_SMALL, 2, (0, 255, 0), 2)
            print("Objects being tracked: {}".format(count))
        # delete detections that are not in allowed_classes
        bboxes = np.delete(bboxes, deleted_indx, axis=0)
        scores = np.delete(scores, deleted_indx, axis=0)

        # encode yolo detections and feed to tracker
        features = encoder(frame, bboxes)
        detections = [Detection(bbox, score, class_name, feature) for bbox, score, class_name, feature in zip(bboxes, scores, names, features)]
```

```
            cmap = plt.get_cmap('tab20b')
            colors = [cmap(i)[:3] for i in np.linspace(0, 1, 20)]

            # run non-maxima supression
            boxs = np.array([d.tlwh for d in detections])
            scores = np.array([d.confidence for d in detections])
            classes = np.array([d.class_name for d in detections])
            indices = preprocessing.non_max_suppression(boxs, classes, nms_max_overlap, scores)
            detections = [detections[i] for i in indices]

            # Call the tracker
            tracker.predict()
            tracker.update(detections)
```

```
        for track in tracker.tracks:
            if not track.is_confirmed() or track.time_since_update > 1:
                continue
            bbox = track.to_tlbr()
            class_name = track.get_class()

        # draw bbox on screen
            color = colors[int(track.track_id) % len(colors)]
            color = [i * 255 for i in color]
            cv2.rectangle(frame, (int(bbox[0]), int(bbox[1])), (int(bbox[2]), int(bbox[3])), color, 2)
            cv2.rectangle(frame, (int(bbox[0]), int(bbox[1]-30)), (int(bbox[0])+(len(class_name)+len(str(track.track_id)))*17, int(bbox[1])), color, -1)
            cv2.putText(frame, class_name + "-" + str(track.track_id),(int(bbox[0]), int(bbox[1]-10)),0, 0.75, (255,255,255),2)

        # if enable info flag then print details about each track
            if FLAGS.info:
                print("Tracker ID: {}, Class: {},  BBox Coords (xmin, ymin, xmax, ymax): {}".format(str(track.track_id), class_name, (int(bbox[0]), int(bl

        # calculate frames per second of running detections
        fps = 1.0 / (time.time() - start_time)
        print("FPS: %.2f" % fps)
        result = np.asarray(frame)
        result = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)

        if not FLAGS.dont_show:
            cv2.imshow("Output Video", result)

        # if output flag is set, save video file
        if FLAGS.output:
            out.write(result)
        if cv2.waitKey(1) & 0xFF == ord('q'): break
    cv2.destroyAllWindows()


if __name__ == '__main__':
    try:
        app.run(main)
    except SystemExit:
        pass
```

7.2.2 Code 2

```
import tensorflow as tf
from absl import app, flags, logging
from absl.flags import FLAGS
from core.yolov4 import YOLO, decode, filter_boxes
import core.utils as utils
from core.config import cfg

flags.DEFINE_string('weights', './data/yolov4.weights', 'path to weights file')
flags.DEFINE_string('output', './checkpoints/yolov4-416', 'path to output')
flags.DEFINE_boolean('tiny', False, 'is yolo-tiny or not')
flags.DEFINE_integer('input_size', 416, 'define input size of export model')
flags.DEFINE_float('score_thres', 0.2, 'define score threshold')
flags.DEFINE_string('framework', 'tf', 'define what framework do you want to convert (tf, trt, tflite)')
flags.DEFINE_string('model', 'yolov4', 'yolov3 or yolov4')

def save_tf():
  STRIDES, ANCHORS, NUM_CLASS, XYSCALE = utils.load_config(FLAGS)

  input_layer = tf.keras.layers.Input([FLAGS.input_size, FLAGS.input_size, 3])
  feature_maps = YOLO(input_layer, NUM_CLASS, FLAGS.model, FLAGS.tiny)
  bbox_tensors = []
  prob_tensors = []
  if FLAGS.tiny:
    for i, fm in enumerate(feature_maps):
      if i == 0:
        output_tensors = decode(fm, FLAGS.input_size // 16, NUM_CLASS, STRIDES, ANCHORS, i, XYSCALE, FLAGS.framework)
      else:
        output_tensors = decode(fm, FLAGS.input_size // 32, NUM_CLASS, STRIDES, ANCHORS, i, XYSCALE, FLAGS.framework)
      bbox_tensors.append(output_tensors[0])
      prob_tensors.append(output_tensors[1])
  else:
    for i, fm in enumerate(feature_maps):
      if i == 0:
        output_tensors = decode(fm, FLAGS.input_size // 8, NUM_CLASS, STRIDES, ANCHORS, i, XYSCALE, FLAGS.framework)
      elif i == 1:
        output_tensors = decode(fm, FLAGS.input_size // 16, NUM_CLASS, STRIDES, ANCHORS, i, XYSCALE, FLAGS.framework)
      else:
        output_tensors = decode(fm, FLAGS.input_size // 32, NUM_CLASS, STRIDES, ANCHORS, i, XYSCALE, FLAGS.framework)
      bbox_tensors.append(output_tensors[0])
      prob_tensors.append(output_tensors[1])
  pred_bbox = tf.concat(bbox_tensors, axis=1)
  pred_prob = tf.concat(prob_tensors, axis=1)
  if FLAGS.framework == 'tflite':
    pred = (pred_bbox, pred_prob)
  else:
    boxes, pred_conf = filter_boxes(pred_bbox, pred_prob, score_threshold=FLAGS.score_thres, input_shape=tf.constant([FLAGS.input_size, FLAGS.input_size]))
    pred = tf.concat([boxes, pred_conf], axis=-1)
  model = tf.keras.Model(input_layer, pred)
  utils.load_weights(model, FLAGS.weights, FLAGS.model, FLAGS.tiny)
  model.summary()
  model.save(FLAGS.output)

def main(_argv):
  save_tf()

if __name__ == '__main__':
    try:
        app.run(main)
    except SystemExit:
        pass
```

7.2.3 Utlis code

```
import cv2
import random
import colorsys
import numpy as np
import tensorflow as tf
from core.config import cfg

def load_freeze_layer(model='yolov4', tiny=False):
    if tiny:
        if model == 'yolov3':
            freeze_layouts = ['conv2d_9', 'conv2d_12']
        else:
            freeze_layouts = ['conv2d_17', 'conv2d_20']
    else:
        if model == 'yolov3':
            freeze_layouts = ['conv2d_58', 'conv2d_66', 'conv2d_74']
        else:
            freeze_layouts = ['conv2d_93', 'conv2d_101', 'conv2d_109']
    return freeze_layouts

def load_weights(model, weights_file, model_name='yolov4', is_tiny=False):
    if is_tiny:
        if model_name == 'yolov3':
            layer_size = 13
            output_pos = [9, 12]
        else:
            layer_size = 21
            output_pos = [17, 20]
    else:
        if model_name == 'yolov3':
            layer_size = 75
            output_pos = [58, 66, 74]
        else:
            layer_size = 110
            output_pos = [93, 101, 109]
    wf = open(weights_file, 'rb')
    major, minor, revision, seen, _ = np.fromfile(wf, dtype=np.int32, count=5)


     j = 0
     for i in range(layer_size):
         conv_layer_name = 'conv2d_%d' %i if i > 0 else 'conv2d'
         bn_layer_name = 'batch_normalization_%d' %j if j > 0 else 'batch_normalization'

         conv_layer = model.get_layer(conv_layer_name)
         filters = conv_layer.filters
         k_size = conv_layer.kernel_size[0]
         in_dim = conv_layer.input_shape[-1]


        if i not in output_pos:
            # darknet weights: [beta, gamma, mean, variance]
            bn_weights = np.fromfile(wf, dtype=np.float32, count=4 * filters)
            # tf weights: [gamma, beta, mean, variance]
            bn_weights = bn_weights.reshape((4, filters))[[1, 0, 2, 3]]
            bn_layer = model.get_layer(bn_layer_name)
            j += 1
        else:
            conv_bias = np.fromfile(wf, dtype=np.float32, count=filters)

        # darknet shape (out_dim, in_dim, height, width)
        conv_shape = (filters, in_dim, k_size, k_size)
        conv_weights = np.fromfile(wf, dtype=np.float32, count=np.product(conv_shape))
        # tf shape (height, width, in_dim, out_dim)
        conv_weights = conv_weights.reshape(conv_shape).transpose([2, 3, 1, 0])
```

```
        if i not in output_pos: ...
        else:
            conv_layer.set_weights([conv_weights, conv_bias])

    # assert len(wf.read()) == 0, 'failed to read all data'
    wf.close()


def read_class_names(class_file_name):
    names = {}
    with open(class_file_name, 'r') as data:
        for ID, name in enumerate(data):
            names[ID] = name.strip('\n')
    return names

def load_config(FLAGS):
    if FLAGS.tiny:
        STRIDES = np.array(cfg.YOLO.STRIDES_TINY)
        ANCHORS = get_anchors(cfg.YOLO.ANCHORS_TINY, FLAGS.tiny)
        XYSCALE = cfg.YOLO.XYSCALE_TINY if FLAGS.model == 'yolov4' else [1, 1]
    else:
        STRIDES = np.array(cfg.YOLO.STRIDES)
        if FLAGS.model == 'yolov4':
            ANCHORS = get_anchors(cfg.YOLO.ANCHORS, FLAGS.tiny)
        elif FLAGS.model == 'yolov3':
            ANCHORS = get_anchors(cfg.YOLO.ANCHORS_V3, FLAGS.tiny)
        XYSCALE = cfg.YOLO.XYSCALE if FLAGS.model == 'yolov4' else [1, 1, 1]
    NUM_CLASS = len(read_class_names(cfg.YOLO.CLASSES))

    return STRIDES, ANCHORS, NUM_CLASS, XYSCALE


def get_anchors(anchors_path, tiny=False):
    anchors = np.array(anchors_path)
    if tiny:
        return anchors.reshape(2, 3, 2)
    else:
        return anchors.reshape(3, 3, 2)

def image_preprocess(image, target_size, gt_boxes=None):

    ih, iw    = target_size
    h,  w, _  = image.shape

    scale = min(iw/w, ih/h)
    nw, nh  = int(scale * w), int(scale * h)
    image_resized = cv2.resize(image, (nw, nh))

    image_paded = np.full(shape=[ih, iw, 3], fill_value=128.0)
    dw, dh = (iw - nw) // 2, (ih-nh) // 2
    image_paded[dh:nh+dh, dw:nw+dw, :] = image_resized
    image_paded = image_paded / 255.

    if gt_boxes is None:
        return image_paded

    else:
        gt_boxes[:, [0, 2]] = gt_boxes[:, [0, 2]] * scale + dw
        gt_boxes[:, [1, 3]] = gt_boxes[:, [1, 3]] * scale + dh
        return image_paded, gt_boxes
```

```
def format_boxes(bboxes, image_height, image_width):
    for box in bboxes:
        ymin = int(box[0] * image_height)
        xmin = int(box[1] * image_width)
        ymax = int(box[2] * image_height)
        xmax = int(box[3] * image_width)
        width = xmax - xmin
        height = ymax - ymin
        box[0], box[1], box[2], box[3] = xmin, ymin, width, height
    return bboxes

def draw_bbox(image, bboxes, info = False, show_label=True, classes=read_class_names(cfg.YOLO.CLASSES)):
    num_classes = len(classes)
    image_h, image_w, _ = image.shape
    hsv_tuples = [(1.0 * x / num_classes, 1., 1.) for x in range(num_classes)]
    colors = list(map(lambda x: colorsys.hsv_to_rgb(*x), hsv_tuples))
    colors = list(map(lambda x: (int(x[0] * 255), int(x[1] * 255), int(x[2] * 255)), colors))

    random.seed(0)
    random.shuffle(colors)
    random.seed(None)

    out_boxes, out_scores, out_classes, num_boxes = bboxes
    for i in range(num_boxes):
        if int(out_classes[i]) < 0 or int(out_classes[i]) > num_classes: continue
        x,y,w,h = out_boxes[i]
        fontScale = 0.5
        score = out_scores[i]
        class_ind = int(out_classes[i])
        class_name = classes[class_ind]
        bbox_color = colors[class_ind]
        bbox_thick = int(0.6 * (image_h + image_w) / 600)
        c1, c2 = (x, y), (x + w, y + h)
        cv2.rectangle(image, c1, c2, bbox_color, bbox_thick)


    if info:…

        if show_label:
            bbox_mess = '%s: %.2f' % (class_name, score)
            t_size = cv2.getTextSize(bbox_mess, 0, fontScale, thickness=bbox_thick // 2)[0]
            c3 = (c1[0] + t_size[0], c1[1] - t_size[1] - 3)
            cv2.rectangle(image, c1, (np.float32(c3[0]), np.float32(c3[1])), bbox_color, -1) #filled

            cv2.putText(image, bbox_mess, (c1[0], np.float32(c1[1] - 2)), cv2.FONT_HERSHEY_SIMPLEX,
                        fontScale, (0, 0, 0), bbox_thick // 2, lineType=cv2.LINE_AA)
    return image

def bbox_iou(bboxes1, bboxes2):…


def bbox_giou(bboxes1, bboxes2):…


def bbox_ciou(bboxes1, bboxes2):…

def nms(bboxes, iou_threshold, sigma=0.3, method='nms'):…

def freeze_all(model, frozen=True):
    model.trainable = not frozen
    if isinstance(model, tf.keras.Model):
        for l in model.layers:
            freeze_all(l, frozen)
def unfreeze_all(model, frozen=False):
    model.trainable = not frozen
    if isinstance(model, tf.keras.Model):
        for l in model.layers:
            unfreeze_all(l, frozen)
```