Table of Contents

[1. Introduction 2](#_Toc132333119)

[2. Background of the Algorithm used 3](#_Toc132333120)

[2.1 Working of Yolov5 3](#_Toc132333121)

[3. Methodology 4](#_Toc132333122)

[3.1 Dataset Collection 4](#_Toc132333123)

[3.2 Importing of libraries, mounting drive and other pre-requisites 5](#_Toc132333124)

[3.3 Training process 6](#_Toc132333125)

[3.4 Validation score 6](#_Toc132333126)

[3.5 Detection Process 6](#_Toc132333127)

[3.6 Feature Extraction 7](#_Toc132333132)

[3.6.1 Performing Size Extraction 10](#_Toc132333135)

[3.6.2 Performing Colour Extraction 12](#_Toc132333136)

[3.6.2 Performing an unique feature extraction (here, Number Plate) 13](#_Toc132333137)

[3.7 Problems faced during the whole coding process 14](#_Toc132333139)

[4. Conclusion 14](#_Toc132333140)

[5. References 14](#_Toc132333141)

# Introduction

Object recognition in images is thought to be a common task for the human brain. Humans are blessed with a pair of eyes that aids our ability to visualise objects as well as our perception of light, colour, and depth. Even though machines have not become sophisticated enough to have all of these features with perfect accuracy, it does have a concept of eye, and the main focus would be on the aspect of object detection, which is a computer vision task that identifies and locates things in images. Several methods have been developed in recent years to address the issue. YOLO (You Only Look Once) is one of the most well-liked real-time object identification techniques available today. There are several versions of Yolo. The one that will be used for this report will be Yolov5. Traffic issues are getting worse and worse as a result of the fast urbanisation and rapid development of the social economy. Serious traffic issues can be resolved with effective traffic monitoring. Monitoring of vehicle traffic plays a significant role in the administration of public policy in metropolitan regions. Vehicle flow data helps authorities decide where to devote funds for things like building new roads, traffic signals, roadside areas, and other infrastructure. These data may also be utilised to develop strategies for easing urban traffic congestion and synchronising traffic lights. Keeping in mind, the traits of quick identification speed, high accuracy, and good detection effect, the upgraded YOLO algorithm can fully exploit the benefits of auxiliary decision-making in a variety of difficult traffic scenarios.

# Background of the Algorithm used

Object detection is a computer vision task in which objects in images or videos are identified and located. It is an essential component of many applications, including surveillance, self-driving cars, and robotics. It focuses on locating a region of interest inside an image and classifying this region in the same way that a standard image classifier would. Multiple areas of interest pointing to various things can be present in a single picture. This elevates object detection to a more complex image classification issue. A well-liked object detection model called YOLO (You Only Look Once) is renowned for its quickness and precision. You Only Look Once (YOLO) uses an end-to-end neural network to predict bounding boxes and class probabilities simultaneously. It differs from prior object detection algorithms, which reworked classifiers to detect objects. YOLO achieved state-of-the-art results by taking a fundamentally different approach to object detection, outperforming other real-time object detection algorithms by a wide margin. While Faster RCNN and other algorithms identify potential regions of interest using the Region Proposal Network and then perform recognition on those regions independently, YOLO performs all of its predictions using a single fully connected layer.

## Working of Yolov5

The YOLO algorithm uses a simple deep convolutional neural network to detect objects in an image as input. The model's first 20 convolution layers are pre-trained using ImageNet by inserting a temporary average pooling and fully connected layer. This pre-trained model is then converted for detection. Previous research demonstrated that adding convolution and connected layers to a pre-trained network improves performance. The final fully connected layer of YOLO predicts class probabilities as well as bounding box coordinates. An input image is divided into a SxS grid by YOLO. If the centre of an object falls into a grid cell, that grid cell is in charge of detecting it. Each grid cell predicts B bounding boxes and their confidence scores. These confidence scores indicate how certain the model is that the box contains an object and how accurate it believes the predicted box is. YOLO predicts multiple bounding boxes per grid cell. Only one bounding box predictor needs to be responsible for each object during training.

YOLO makes one predictor "responsible" for predicting an object based on which prediction has the highest current IOU (the total size of the intersection divided by the total size of the union of both the bounding boxes) with the ground truth. This results in specialisation among the bounding box predictors. Each predictor improves at forecasting specific sizes, aspect ratios, or object classes, increasing the overall recall score. On-maximum suppression is a key technique in the YOLO models (NMS). NMS is a post-processing step used to improve object detection accuracy and efficiency. Multiple bounding boxes are frequently generated for a single object in an image during object detection. The same team that created the original YOLO algorithm released YOLO v5 in 2020 as an open-source project, which is now maintained by Ultralytics. The success of earlier versions is built upon by YOLO v5, which also includes a number of enhancements and new features. In contrast to YOLO, YOLO v5 uses a more intricate architecture called EfficientDet, which is based on the EfficientNet network architecture.

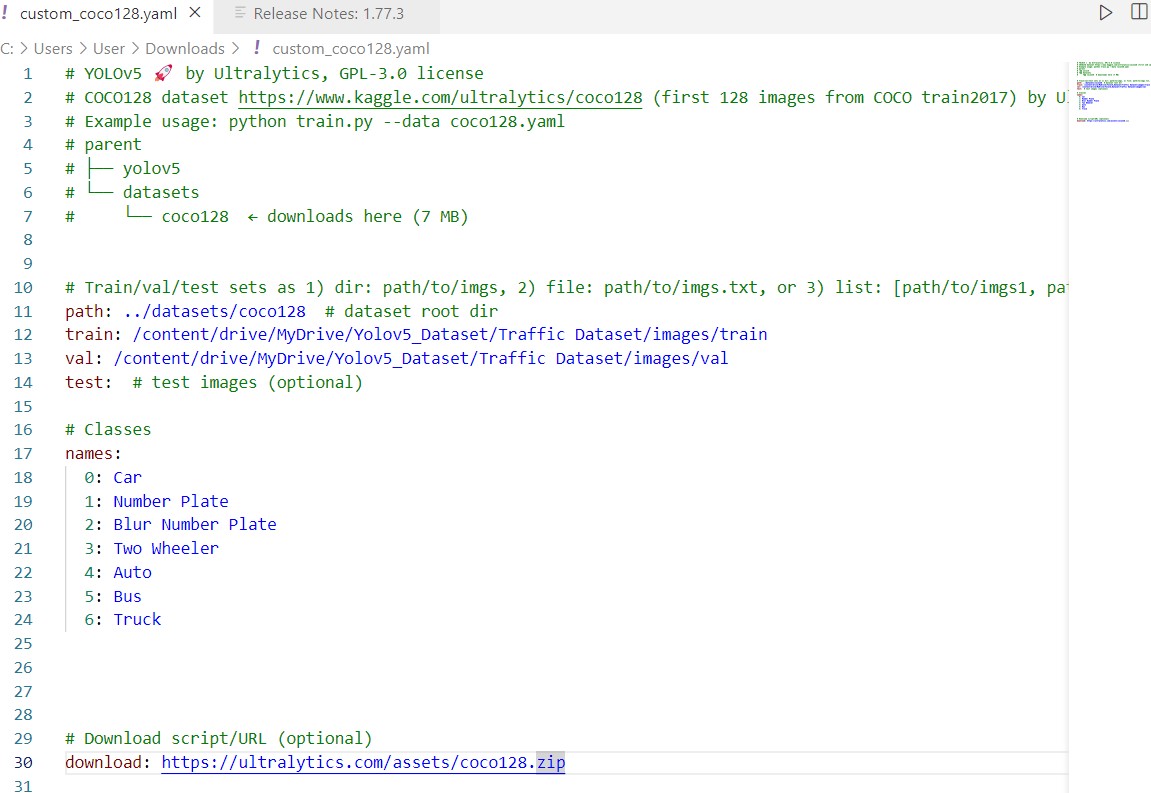
YOLO v5 can achieve greater accuracy and better generalisation to a larger variety of item categories thanks to the use of a more complicated architecture. YOLO v5 was trained using D5, which is a large and more varied dataset that consists of a total of 600 object types. The anchor boxes are created using a new technique in YOLO v5 called "dynamic anchor boxes." The ground truth bounding boxes are first grouped into clusters using a clustering method, and then the centroids of those clusters are used as the anchor boxes. As a result, the anchor boxes can match the size and shape of the identified objects more closely.The idea of "spatial pyramid pooling" (SPP), is also introduced in Yolov5 , which is a kind of pooling layer used to lower the spatial resolution of the feature maps. Since SPP enables the model to view the objects at various scales, it is employed to enhance the detection performance for small objects .Both YOLO v4 and v5 train the model using a comparable loss function. A new concept known as "CIoU loss," a variation of the IoU loss function, is however introduced in YOLO v5 and is intended to enhance the model's performance on imbalanced datasets.

# Methodology

## Dataset Collection

The Traffic dataset was retrieved from Kaggle. The dataset had two folders, namely images and labels, with each folder further divided into training and validation set. It contained the combination of 7 different classes. They are Car, Number Plate, Blur Number Plate, Two Wheeler, Bus and Truck.

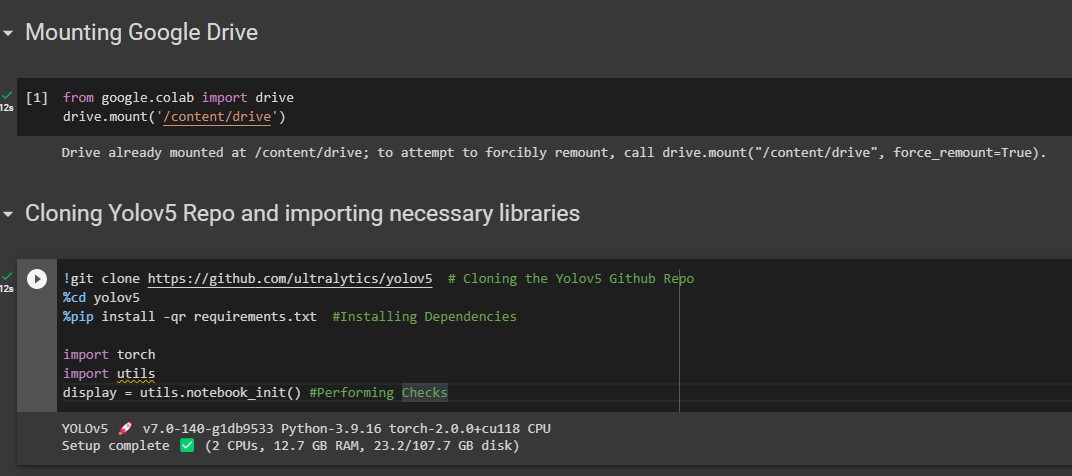
The pre-existing coco128.yaml dataset was downloaded from the yolov5 repository and then modified locally. The modifications included the addition of the path to the dataset stored in Google drive and deletion of all the unnecessary classes. Only 7 classes were included along with its index in the .yaml file, which we want to be detected as an object during Yolov5 detection.



The modified coco128.yaml was renamed as custom\_coco128.yaml and then uploaded to the Google Colab alongside yolov5 repository, to be used later during the training process.

## Importing of libraries, mounting drive and other pre-requisites

Before the training process could be started, at first we needed to mount our Google drive containing the dataset, which can then be accessed by the custom\_coco128.yaml. After that important libraries were imported and Yolov5 repository was cloned. All the necessary dependencies were installed using the ”%pip install -qr requirements.txt“ command.



## Training process

Now the training of the dataset is conducted using the “!python train.py --img 640 --batch 2 --epochs 20 --data custom\_coco128.yaml --weights yolov5s.pt --cache ram --cache disk“ command.

At first, batch size was kept quite high, and the number of epochs was kept at 10.

After the training was done, it was observed that the validation check was showing inaccurate results, and a lot of classes were confused with the class ‘car’. All of the two wheelers coming into the frame were misidentified as cars. For trying to resolve this issue, the batch size was decreased to 2, and the number of epochs was increased to 20.

During this time, the accuracy increased considerable, and all classes were pretty much detected correctly. 20 epochs were completed in about.320 hours, and the weight that was created namely best.pt after the training process, got saved to runs/train/exp/weights/best.pt directory of Yolov5.

The **problem** with Google Colab being that when the notebook gets recycled after a certain period of time, it deletes all the uploaded files and all the training data including the weight that got saved gets removed as well. So, the best.pt thus created was downloaded locally and renamed as Vehicles\_trained\_weight.pt to avoid losing it due to recycling taking place, since it will be required for the upcoming detection purposes.

## Validation score

All the validation score, namely f1 score, recall score, confusion matrix gets created during the training process itself, and is not required to be done externally to get these values. These provides an important metric for the evaluation of the model performance. These scores too gets deleted after the runtime disconnects, but will again reappear if the training is repeated.

### Detection Process

Now that the training process has been completed, we are left with detection. A sample video was taken and was then used for the purpose of detecting objects in each individual frames of the video, using the command ”!python detect.py --weights /content/Vehicles\_trained\_weight.pt --img 640 --conf 0.5 --source /content/video3-1min.mp4”

The confidence threshold was set to 0.5 and the trained weight “Vehicles\_trained\_weight.pt” was used for detection. After the process concluded, the resulting sample video was then saved in the runs/detect/exp directory. The video was then downloaded locally for viewing the result.

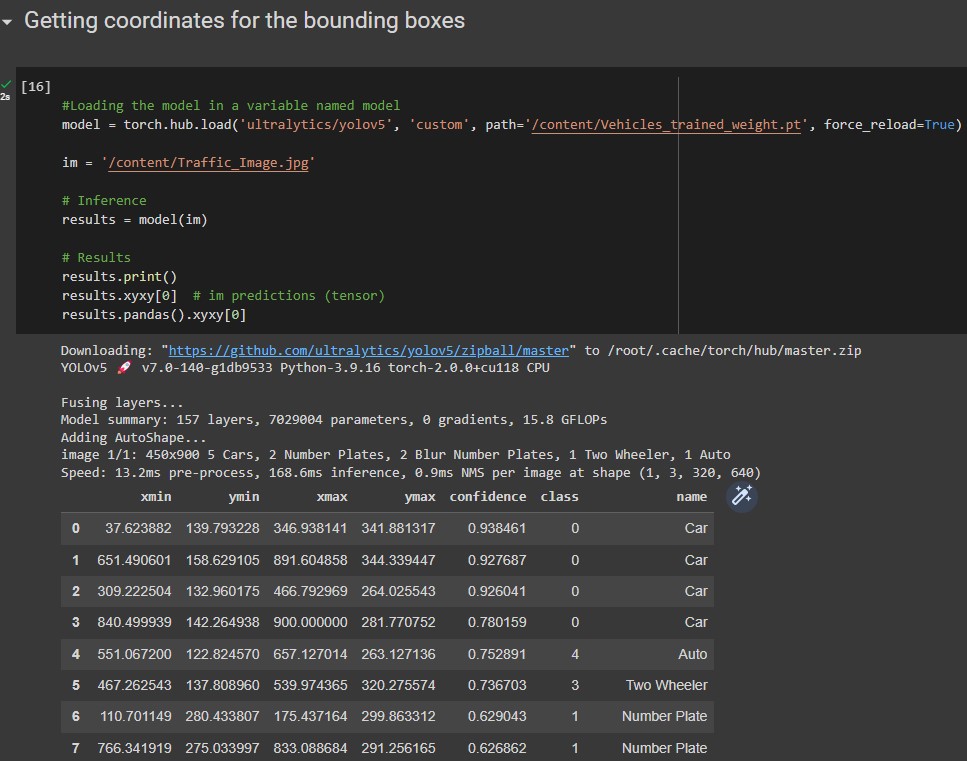
Now it was time for feature extraction of objects from a single image. So a traffic image found online was uploaded to the Colab notebook, and then detected using the “!python detect.py --weights /content/Vehicles\_trained\_weight.pt --img 640 --conf 0.5 --source /content/Traffic\_Image.jpg” command. The resulting image was then displayed in the notebook using cv2\_imshow function.



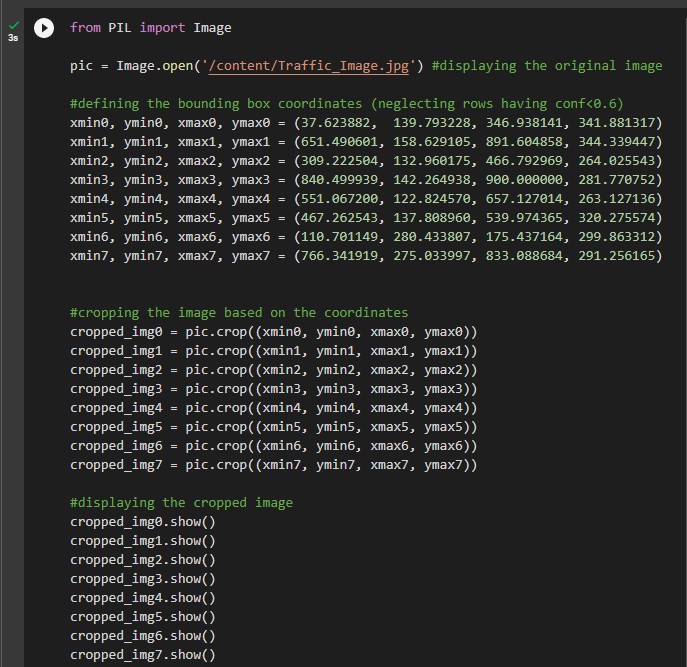
The above image is the result of the detection done by Yolov5 using the weight that got created after the training process.

## Feature Extraction

Feature extraction involves the identification and selection of relevant characteristics of the input image that can then be used to improve object detection accuracy or it can be used for other purposes. For the feature extraction, firstly we would require the coordinates of the bounding box of the detected object. For that purpose, the model needed to be loaded and the trained weight was to be included. The XY coordinates are then retrieved from the results, which are stored in the form of tensors in Yolov5 after the training is completed.



These coordinates are then inputted for the purpose of cropping the bounding box of the detected object. The cropped images are then resized to the size 224x224. This is done to ensure the features that are extracted are consistent across all detected objects. The output gave a total of 11 object count. The last three detection having confidence <0.6 were neglected.



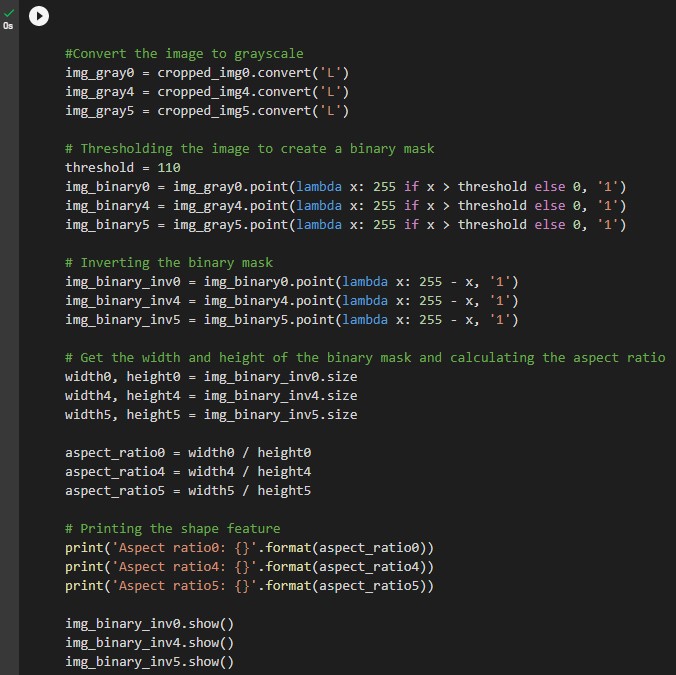
The cropped images were then displayed in the notebook. Going through the cropped images it was evident that the number of objects detected having confidence >0.6 were 4 cars, 1 auto, 1 two wheeler and two number plates. The displayed images are shown below.



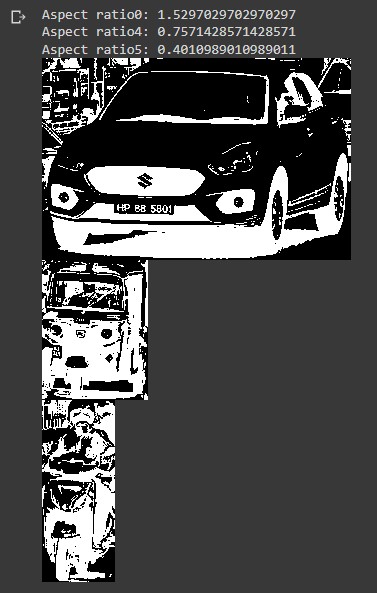
After selecting one of each class from the detected object and resizing it, we are left with three object namely car, auto and two wheeler (not considering the number plate, as it is put under the unique feature itself) indexed as 0, 4 and 5.

### Performing Size Extraction

For performing size extraction of the detected object, we first convert the image to grayscale .The convert() function was used to convert the image to grayscale. The point() method was then used to threshold the grayscale image to create a binary mask, where the pixels above a certain threshold are set to white and pixels below the threshold are set to black. The threshold value can be set using hit and trial method, by getting to view the output each time the threshold is set. The point() method is again used to invert the binary mask. The width and height of the inverted binary mask are then extracted using the size attribute. Finally, the aspect ratio of the inverted binary mask is calculated by dividing the width by the height, and is then printed to the console.

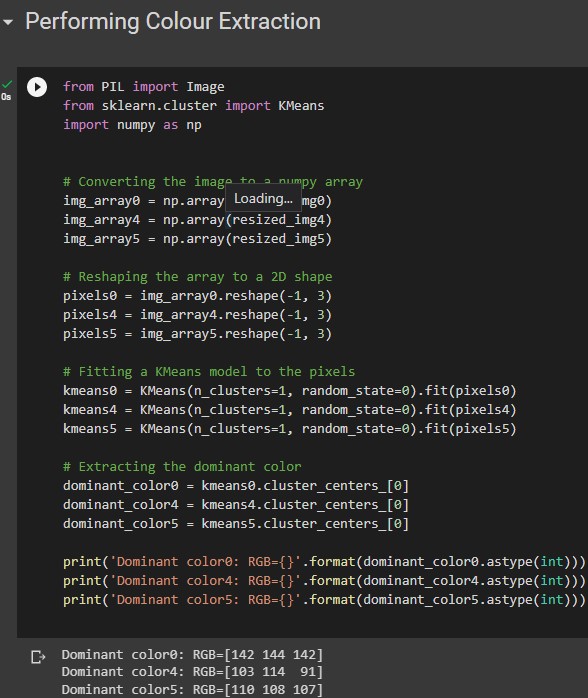


The output is then displayed using the show() function, as given below:-

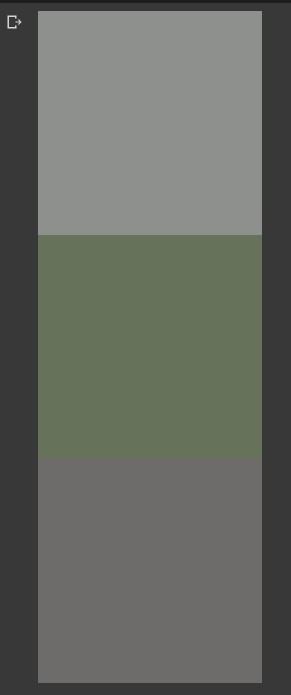


### 3.6.2 Performing Colour Extraction

For colour extraction, KMeans method was used to extract the dominant colour of the detected object. First, the resized image is loaded and is then converted into a numpy array using numpy.array () function. The array thus formed is reshaped into a 2D shape, where each row represents a pixel, and each column represents a RGB value. A KMeans model having one cluster is fit to the pixel values and the centroid of that cluster is taken to be the dominant colour. The dominant colour is then displayed in the Colab notebook using .show() function.



The output thus obtained is shown below in the order of index 0, 4 and 5.



* + 1. Performing an unique feature extraction (here, Number Plate)

This feature extraction was already done during the initial detection of objects in the traffic image. It was already included in the custom\_coco128.yaml dataset as a class, and has been extracted in the form of a resized image in the later stage.



## Problems faced during the whole coding process

Some of the problems faced while coding included the following:-

While cloning Yolov5 Repo and importing necessary libraries, there were some instances where we faced with runtime error, and was resolved by just restarting the kernel.

While trying to display the colour using the dominant RGB values from the previous output directly, we were met with “TypeError: 'Image' object is not callable“. This was fixed by manually typing in the RGB values.

“AttributeError: 'NoneType' object has no attribute 'clip” was thrown while trying to display a picture using cv2\_imshow(). Turns out, that improper file directory of the picture lead to this error.

“AttributeError: 'numpy.ndarray' object has no attribute 'convert'” was shown when improper indexing of cropped image was done. This was later fixed.

Context extraction and distance measurement were some of the tasks that failed, due to errors and confusion while trying to code them.

# Conclusion

Yolov5 has proven to be one of the most popular and widely used algorithm used for the purpose of object detection. It detects objects seamlessly and with great deal of accuracy. Imbalanced datasets are also internally handled in yolov5 which can mitigate over fitting uses or bias to any specific class. Object detection, counting and feature extraction from a given detection can prove to be useful for further increasing the accuracy of the detection, and aid in traffic management and safety monitoring. In order to precisely locate the vehicle position and accurately classify the vehicle type, vehicle detection is used to detect the target vehicle from the inspected area. Due to the characteristics of vehicles and the variety of detection states, object detection has long been a hot topic in the field of computer vision, with vehicle detection being the most fundamental and challenging problem.

# References

Zhang, Yu & Guo, Zhongyin & Wu, Jianqing & Tian, Yuan & Tang, Haotian & Guo, Xinming. (2022). Real-Time Vehicle Detection Based on Improved YOLO v5. Sustainability. 14. 12274. 10.3390/su141912274.

Rahman, Raian & Azad, Zadid & Hasan, Bakhtiar. (2021). Densely-Populated Traffic Detection using YOLOv5 and Non-Maximum Suppression Ensembling.

Lihi Gur Arie. (2022). The practical guide for Object Detection with YOLOv5 algorithm

Oluwaseyi, Olorunshola & Irhebhude, Martins & Evwiekpaefe, Abraham. (2023). A Comparative Study of YOLOv5 and YOLOv7 Object Detection Algorithms. Journal of Computing and Social Informatics. 2. 1-12. 10.33736/jcsi.5070.2023.

Additional Note

Traffic\_Image.jpg , Vehicles\_trained\_weight.pt, custom\_coco128.yaml , video3-1min.mp4 are all the files that needs to be uploaded to the Google Colab for the proper working of the code, without facing any errors along the way.