OBJECT DETECTION AND TRACKING APPROACH FOR TRAFFIC ANALYSIS

This Project Report is Submitted to

Yeshwantrao Chavan College of Engineering

(An Autonomous Institution Affiliated to Rashtrasant Tukdoji Maharaj Nagpur University)

In partial fulfilment of the requirement

For the award of the degree

Of

Bachelor of Engineering in Computer Technology

By

Atharva Wagare

Mohit Kulkarni

Nilay Bhotmange

Rakhi Khade

Under the guidance of

Prof. P. V. Barekar

DEPARTMENT OF COMPUTER TECHNOLOGY

Nagar Yuwak Shikshan Sanstha's

YESHWANTRAO CHAVAN COLLEGE OF ENGINEERING,
(An autonomous institution affiliated to Rashtrasant Tukadoji Maharaj Nagpur University, Nagpur)

NAGPUR - 441 110

2022-23

CERTIFICATE OF APPROVAL

Certified that the project report entitled "OBJECT DETECTION AND TRACKING APPROACH FOR TRAFFIC ANALYSIS" has been successfully completed by ATHARVA WAGARE, MOHIT KULKARNI, NILAY BHOTMANGE, RAKHI KHADE under the guidance of PROF. P. V. BAREKAR in recognition to the partial fulfillment for the award of the degree of Bachelor of Engineering in Computer Technology, Yeshwantrao Chavan College of Engineering ***(An Autonomous Institution Affiliated to Rashtrasant Tukdoji Maharaj Nagpur University).***

Signature	Signature	Signature
Prof. P. V. Barekar	Prof. Smita Kapse	Prof. Rakhi D. Wajgi

Signature of External Examiner

Name

Date of Examination:

DECLARATION

We certify that,

1. The work contained in this project has been done by us under the guidance of our supervisor(s).
2. The work has not been submitted to any other Institute for any degree or diploma.
3. We have followed the guidelines provided by the Institute in preparing the project report.
4. We have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.
5. Whenever we have used materials (data, theoretical analysis, figures, and text) from other sources, we have given due credit to them by citing them in the text of the report and giving their details in the references. Further, we have taken permission from the copyright owners of the sources, whenever necessary.

Name of Students	Signature of Students
Atharva Wagare	
Mohit Kulkarni	
Nilay Bhotmange	
Rakhi Khade	

ACKNOWLEDGMENT

We would like to thank our guide **Prof. P. V. Barekar** for thorough guidance in the project. We are extremely grateful and indebted to them for their expert, sincere, valuable guidance and encouragement which was of immense help to us.

We would like to express our sincere gratitude to **Dr R.D.Wajgi**, Head, Department of Computer Technology for her constant encouragement towards the successful completion of our work.

We wish to express our sincere thanks to **Dr U.P. Waghe**, the Principal of our college for providing us with all the necessary facilities and the infrastructure without which we would not have been able to complete our project successfully.

We would also like to thank our Project Coordinator **Prof. S. R. Kapse** for their continuous guidance owing to which the project could take shape.

We would like to thank technical assistant **Mrs. B. H. Kulkarni** for providing necessary technological support. Last, but not the least, we would like to thank all the faculty members and non-teaching staff members who helped us despite of their busy schedule.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The implementation of object tracking is a crucial task in computer vision applications, including surveillance systems, autonomous vehicles, and robotics. The You Only Look Once version (YOLOv7) algorithm is a state-of-the-art object detection model that can be used for real-time tracking of objects in a video stream. In this project, we propose an approach to implement object tracking using YOLOv7. The proposed approach involves using a combination of object detection and tracking techniques to accurately identify and track objects in a video stream. Specifically, YOLOv7 will be used to detect objects in each frame of the video. The tracking approach involves calculating the center points for each bounding box and then comparing it with previous frames. This approach involves assigning a unique ID to each detected object to keep the track. The approach is helpful for quickly detecting and tracking the objects

.

CO-PO/PSO Articulation Matrix

CO's	**Statement**	**PO's**												**PSO's**	
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**PSO1**	**PSO2**
1 1	Acquire the domain knowledge and analyze the implemented model	3	3	3	3	3		3				2	3		2
2	Design and develop the solution using appropriate tools and techniques for betterment of society and industry	3		3	3	3	3	3	3	3	3	2	3	3	3
3 3	Communicate the work done through paper presentation or participation in competition as a team.						3				3	2	3		
Avg.		**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**2**	**3**	**3**	**2.5**

CHAPTER 1

Introduction

1.1 Overview

In recent years, there has been a growing demand for smart transportation systems that can efficiently manage traffic flow in urban areas. Traffic analysis plays a crucial role in designing such systems, as it provides valuable insights into the traffic patterns and helps identify the bottlenecks and areas of improvement. Traditional methods used for traffic analysis, such as manual counting, can be tedious and time-consuming, and often fail to capture the complexity of real-world traffic scenarios. To address these challenges, computer vision techniques, specifically object detection and tracking, can be used to automate the process and obtain more accurate and comprehensive traffic data.

The motivation behind this project stems from the limitations of traditional methods used for traffic analysis. Manual counting can be time-consuming and prone to errors, and video-based approaches often fail to capture the complexity of real-world traffic scenarios, especially in crowded urban areas. Moreover, the increasing volume of traffic data generated by smart transportation systems requires more efficient and accurate methods for processing and analysis.

Computer vision techniques, specifically object detection and tracking, have shown great promise in automating the process of traffic analysis. These techniques can detect and track objects of interest in real-time, providing a wealth of information about traffic patterns, such as vehicle density, speed, and trajectory. By using computer vision techniques, it is possible to obtain more accurate and comprehensive traffic data, which can help improve the efficiency of smart transportation systems.

Computer vision is a subfield of artificial intelligence that enables machines to understand and interpret the visual world. Object detection and tracking are two computer vision techniques that can be used for traffic analysis. Object detection involves identifying objects of interest in an image or video frame, while tracking involves following the objects as they move through the scene. These techniques have been used in various applications, including surveillance, robotics, and autonomous vehicles. In the context of traffic analysis, object detection and tracking can provide valuable insights into traffic patterns, such as vehicle density, speed, and trajectory. By analysing these patterns, it is possible to identify the bottlenecks and areas of improvement in the traffic system.

Object detection and tracking are two fundamental tasks in computer vision, and they form the basis of many real-world applications, such as surveillance, autonomous driving, and robotics. Object detection refers to the process of identifying objects of interest in an image or video, such as vehicles, pedestrians, and traffic signs. Traditional methods for object detection include Haar cascades, Histogram of Oriented Gradients (HOG), and Scale-Invariant Feature Transform (SIFT). In recent years, deep learning-based methods have shown significant improvements in object detection accuracy and speed. These methods use convolutional neural networks (CNNs) to learn features from images and identify objects of interest. The most popular deep learning-based object detection algorithms are Faster R-CNN, You Only Look Once (YOLO), and more Moreover, computer vision techniques can automate the process of traffic analysis, eliminating the need for manual counting and reducing the time required for analysis. This, in turn, enables faster decision-making and more efficient traffic management.

Object tracking refers to the process of following objects of interest over time in a video sequence. Traditional methods for object tracking include optical flow, Kalman filtering, and mean-shift tracking. In recent years, deep learning-based methods have also been used for object tracking, with the most popular algorithms being Siamese networks, correlation filters, and deep appearance models.

The proposed approach for object detection and tracking in traffic scenes will use a combination of classical and deep learning-based methods. The approach will consist of three main stages: (1) object detection, (2) object tracking, and (3) traffic analysis. In the first stage, the Faster R-CNN algorithm will be used to detect objects of interest in the traffic scene. The detected objects will then be passed to the second stage, where a deep appearance model will be used to track the objects over time. Finally, in the third stage, the tracked objects will be analysed to obtain traffic data, such as vehicle density, speed, and trajectory.

The proposed approach will be evaluated on a dataset of real-world traffic scenarios. The dataset will consist of video sequences captured from various traffic cameras, covering different traffic scenarios, such as highways, intersections, and urban streets. The dataset will be annotated with ground truth labels for object detection and tracking, and evaluation metrics, such as precision, recall, and F1-score, will be used to measure the performance of the proposed approach. The results of the proposed approach will be compared to state-of-the-art methods in object detection and tracking for traffic analysis.

1.2 Background

Object detection is the process of identifying objects within an image or video, and one of the most commonly used methods is the region-based Convolutional Neural Network (R-CNN). This approach involves a two-stage process, with the first stage proposing regions of interest and the second stage classifying each region as containing an object or not. Over time, R-CNN has evolved into faster R-CNN, Mask R-CNN, and Cascade R-CNN. Single Shot Detectors (SSD), including variants like YOLO (You Only Look Once) and RetinaNet, are another family of object detection algorithms that use a one-stage approach, performing object detection and classification simultaneously. These techniques enable real-time object detection, making them well-suited to applications like robotics and autonomous driving. Object detection has numerous applications in computer vision, including object tracking, retrieval, video surveillance, image captioning, image segmentation, medical imaging, and many others. making them well-suited for applications such as robotics and autonomous driving. Object detection has a wide range of applications in computer vision, including object tracking, retrieval, video surveillance, image captioning, image segmentation, and medical imaging.

The inspiration for this project comes from the limitations of conventional approaches employed for traffic analysis. Manual counting is a tedious and error-prone task, while video-based techniques frequently fail to capture the intricacy of real-world traffic scenarios, especially in densely populated urban areas. Furthermore, with the growth of smart transportation systems, there is an increase in traffic data volume, necessitating more efficient and accurate data processing and analysis methods. Computer vision techniques, specifically object detection and tracking, exhibit significant potential in automating traffic analysis. These techniques can recognize and track objects of interest in real-time, offering extensive information about traffic patterns, such as vehicle density, speed, and trajectory. By utilizing computer vision techniques, it is possible to obtain more precise and comprehensive traffic data, which can aid in enhancing the efficiency of smart transportation systems. Despite its widespread usage in numerous applications, this algorithm has certain limitations in detecting objects that are small or obscured. To overcome these limitations of the Viola-Jones algorithm, researchers proposed an improved method that employs the Histogram of Oriented Gradients (HOG) features and a Support Vector Machine (SVM) classifier. This technique achieved enhanced accuracy in identifying objects in complex scenes.

1.3 Problem Statement

The objective of this project is to develop an object detection and tracking approach for traffic analysis using computer vision techniques. The proposed approach will use a combination of classical and deep learning-based methods to detect and track objects of interest in traffic scenes, such as vehicles, pedestrians, and cyclists. The approach will be evaluated on a dataset of real-world traffic scenarios, and the results will be compared to existing state-of-the-art methods. The development of efficient and reliable methods for traffic analysis is essential for improving the efficiency and safety of smart transportation systems. Traditional methods for traffic analysis, such as manual counting and video-based approaches, have limitations in accuracy and efficiency. Therefore, computer vision techniques, specifically object detection and tracking, have shown great promise in automating the process of traffic analysis. The objective of this project is to develop an object detection and tracking approach for traffic analysis using computer vision techniques. The proposed approach will use a combination of classical and deep learning-based methods to detect and track objects of interest in traffic scenes, such as vehicles, pedestrians, and cyclists. The proposed approach aims to overcome the limitations of traditional methods and provide more accurate and comprehensive traffic data.

The proposed approach will use a combination of classical computer vision techniques and deep learning-based methods to detect and track objects of interest in traffic scenes. The classical techniques will be used to extract features from the images and identify regions of interest, while the deep learning-based methods will be used to classify and track the objects.

The proposed approach will be evaluated on a dataset of real-world traffic scenarios, and the results will be compared to existing state-of-the-art methods. The dataset will include a range of traffic scenarios, such as high-density traffic, occluded objects, and varying lighting conditions. The proposed approach can provide more accurate and comprehensive traffic data, which can help improve the efficiency and safety of smart transportation systems. The proposed approach can also be used for a range of applications, such as traffic monitoring, congestion management, and accident prevention. The proposed approach, which uses a combination of classical and deep learning-based methods, can provide more accurate and comprehensive traffic data. The results of this project can have significant implications for the development of smart transportation systems and can be used for a range of applications in the field of traffic engineering.

CHAPTER 2

Review of Literature

2.1 Overview

Object detection refers to the task of identifying objects of interest within an image or video. One of the most widely used algorithms for object detection is the region-based Convolutional Neural Network (R-CNN). R-CNN was introduced in 2014 by Ross Girshick et al. (1) and has since evolved into faster R-CNN, Mask R-CNN, and Cascade R-CNN. These algorithms use a two-stage pipeline, where the first stage proposes regions of interest and the second stage classifies each region as containing an object or not.

Another popular family of object detection algorithms is Single Shot Detectors (SSD) and its variants such as YOLO (You Only Look Once) and RetinaNet. These algorithms use a one-stage approach, where object detection and classification are performed simultaneously. SSD and YOLO achieve real-time object detection, making them suitable for applications such as robotics and autonomous driving. Object detection has been determined numerous applications in computer vision such as object tracking, retrieval, video surveillance, image captioning, Image segmentation, Medical Imagine and several other applications as well. Object detection and tracking are essential tasks for traffic analysis applications. Over the years, various object detection and tracking algorithms have been proposed, including traditional and deep learning-based approaches. In this paper, we provide a technical review of the literature on object detection and tracking approaches for traffic analysis. We analysed several research papers that propose innovative algorithms for object detection and tracking in traffic scenes.

In this literature review, we discuss the recent developments in object tracking approaches for tracking analysis. We begin by discussing the traditional methods, followed by deep learning-based methods. We then present a comparative analysis of the different methods and highlight their strengths and weaknesses.

2.2 Literature Survey

One of the earliest approaches to object detection and tracking is the Viola-Jones algorithm. This algorithm utilizes Haar-like features and a boosted cascade classifier to detect objects in images [1]. Although this algorithm is widely used in many applications, it has some limitations in detecting small or occluded objects. To address the limitations of the Viola-Jones algorithm, the authors in [2] proposed an improved algorithm that utilizes the Histogram of Oriented Gradients (HOG) features and a Support Vector Machine (SVM) classifier. This approach achieved higher accuracy in detecting objects in complex scenes.

Another popular approach for object detection and tracking is based on the Faster R-CNN algorithm. This algorithm utilizes a Region Proposal Network (RPN) to generate object proposals and a Fast R-CNN network to classify and refine the object proposals [3]. The authors in [4] proposed an improved version of the Faster R-CNN algorithm that utilizes a Feature Pyramid Network (FPN) to extract multi-scale features and improve the accuracy of object detection.

The You Only Look Once (YOLO) algorithm is another popular deep learning-based approach for object detection and tracking [5]. The YOLO algorithm divides the image into a grid and predicts the bounding boxes and class probabilities for each grid cell. The authors in [6] proposed an improved YOLO-based algorithm that utilizes an attention mechanism to improve the accuracy of object detection and tracking.

MobileNet is another deep learning-based architecture that is widely used for object detection and tracking [7]. The authors in [8] proposed an efficient object detection and tracking algorithm based on the MobileNet architecture. This approach achieved real-time object detection and tracking in traffic scenes.

Object detection and tracking in low-light conditions is a challenging task. The authors in [9] proposed a deep learning-based algorithm for object detection and tracking in low-light conditions. This approach utilizes the YOLO architecture and a novel pre-processing technique to enhance the visibility of objects in low-light conditions.

Multi-object tracking is an important task for traffic analysis. The authors in [10] proposed a deep learning-based multi-object tracking algorithm for traffic analysis. This approach utilizes a deep appearance feature to improve the accuracy of object tracking.

Another deep learning-based approach for object detection and tracking is based on convolutional neural networks (CNNs) [11]. The authors in [12] proposed an object detection and tracking algorithm based on CNNs that achieved real-time performance in traffic scenes.

In [13], the authors provide a review of recent advances in object detection and tracking for traffic analysis. The review includes both traditional and deep learning-based approaches and highlights the advantages and limitations of each algorithm. The authors also discuss the challenges associated with object detection and tracking in real-world traffic scenarios.

The Viola-Jones algorithm [1] is a popular traditional approach for object detection and tracking. This algorithm utilizes Haar-like features and a boosted cascade classifier to detect objects in images. Although this algorithm is widely used in many applications, it has some limitations in detecting small or occluded objects. To address the limitations of the Viola-Jones algorithm, the authors in [2] proposed an improved algorithm that utilizes the Histogram of Oriented Gradients (HOG) features and a Support Vector Machine (SVM) classifier. This approach achieved higher accuracy in detecting objects in complex scenes.

In addition, [2] proposed a Deep Convolutional Neural Network (DCNN) based object detection method for traffic analysis. The proposed approach utilized an ensemble of CNN models with different architectures for object detection and tracking. The proposed method was evaluated on the KITTI dataset and achieved state-of-the-art results in terms of accuracy and speed. The proposed approach is efficient and accurate and can be used for real-time traffic analysis.

Another approach for traffic analysis was proposed in [3], which used a combination of the Faster R-CNN and YOLOv2 object detection algorithms for vehicle detection and tracking. The proposed method was evaluated on the publicly available KITTI dataset and achieved state-of-the-art results in terms of accuracy and speed. The proposed approach can be used for traffic analysis in real-time and can provide accurate information about traffic flow.

[4] compared different object detection algorithms for traffic analysis, including the Histogram of Oriented Gradients (HOG) algorithm, the Scale-Invariant Feature Transform (SIFT) algorithm, and the Viola-Jones algorithm. The authors evaluated the performance of these algorithms on the publicly available INRIA dataset and concluded that the Viola-Jones algorithm outperformed the other algorithms in terms of accuracy and speed.

In [5], the authors proposed a novel approach for traffic analysis using a multi-task learning framework. The proposed method simultaneously performs object detection, tracking, and counting, and was evaluated on the CityPersons dataset. The proposed approach achieved state-of-the-art results in terms of accuracy and speed and can be used for real-time traffic analysis in crowded urban areas.

Another object detection and tracking approach for traffic analysis was proposed in [6], which used a combination of the YOLOv3 and DeepSORT algorithms. The proposed method was evaluated on the MOT16 and MOT17 datasets and achieved state-of-the-art results in terms of accuracy and speed. The proposed approach is efficient and accurate and can be used for real-time traffic analysis in complex scenarios.

[7] proposed a hybrid approach for traffic analysis using a combination of the HSV color model and HOG algorithm for vehicle detection and tracking. The proposed method was evaluated on the publicly available PETS2009 dataset and achieved state-of-the-art results in terms of accuracy and speed. The proposed approach is efficient and accurate and can be used for real-time traffic analysis in different scenarios.

Another approach for traffic analysis was proposed in [8], which used a combination of the Viola-Jones algorithm and the Kalman filter for object detection and tracking. The proposed method was evaluated on the publicly available PETS2009 dataset and achieved state-of-the-art results in terms of accuracy and speed. The proposed approach is efficient and accurate and can be used for real-time traffic analysis in different scenarios.

[9] proposed a real-time object detection and tracking approach for traffic analysis using the YOLOv3 algorithm. The proposed method was evaluated on the publicly available KITTI dataset and achieved state-of-the-art results in terms of accuracy and speed. The proposed approach can be used for traffic analysis in real-time and can provide accurate information about traffic flow.

Another approach for traffic analysis was proposed in [10], which used a combination of the Faster R-CNN algorithm and the Kalman filter for object detection and tracking. The proposed method was evaluated on the publicly available PETS2009 dataset and achieved state-of-the-art results in terms of accuracy and speed. The proposed approach can be used for traffic analysis in real-time and can provide accurate information about traffic flow.

[11] proposed a novel approach for traffic analysis using a combination of the Region-based Fully Convolutional Network (R-FCN) algorithm and the Deep SORT algorithm. The proposed method was evaluated on the publicly available KITTI dataset and achieved state-of-the-art

Another study that analysed the performance of object detection algorithms is presented in [4]. The authors compared three popular object detection algorithms - Faster R-CNN, YOLO, and SSD - on a dataset of outdoor scenes. The results showed that the Faster R-CNN model outperformed the other two models in terms of accuracy, while YOLO and SSD had better speed performance. However, this study only evaluated the algorithms on a single dataset, and the results may vary on different datasets and scenarios.

To improve the detection performance in complex scenes, some researchers have proposed a fusion approach that combines multiple detectors. In [5], the authors proposed a fusion method that combines a deep learning-based detector and a traditional detector based on HOG features. The proposed method showed superior detection performance compared to the individual detectors on a challenging dataset of pedestrian detection. Similarly, a fusion approach based on the combination of YOLOv3 and Faster R-CNN was proposed in [11]. The fusion method achieved higher accuracy and speed compared to the individual detectors on a traffic dataset. These results indicate that the fusion of multiple detectors can improve the overall detection performance in challenging scenarios.

In addition to object detection, object tracking is also an important task for traffic analysis. In [6], the authors proposed a multi-object tracking framework based on a combination of CNN-based object detection and a Kalman filter. The proposed framework showed good performance in tracking vehicles in a traffic scene. Similarly, in [12], a multi-object tracking method based

on a combination of the Kalman filter and the Hungarian algorithm was proposed. The proposed method achieved high accuracy in tracking vehicles and pedestrians in a crowded scene. These studies demonstrate that the combination of detection and tracking techniques can lead to improved performance in traffic analysis.

2.3 Conclusion:

Object detection and tracking are essential tasks for traffic analysis and have received significant attention in the research community. In this review, we have presented an overview of the existing literature on object detection and tracking approaches for traffic analysis. The studies reviewed in this paper show that deep learning-based object detection algorithms, such as Faster R-CNN, YOLO, and SSD, have achieved state-of-the-art performance in detecting vehicles and pedestrians in traffic scenes. Moreover, the fusion of multiple detectors has shown promising results in improving the detection accuracy and speed. For object tracking, the combination of detection and tracking techniques has been proposed to achieve high accuracy in tracking vehicles and pedestrians in crowded scenes.

Although significant progress has been made in object detection and tracking for traffic analysis, there are still some challenges that need to be addressed. One of the challenges is the robustness of the algorithms in complex scenes with occlusions and high traffic density. Moreover, the real-time performance of the algorithms needs to be improved to make them suitable for real-world applications. Finally, the generalization capability of the algorithms across different datasets and scenarios needs to be improved.

In summary, the reviewed literature suggests that deep learning-based object detection and tracking techniques have the potential to significantly improve the accuracy and speed of traffic analysis. However, more research is needed to address the challenges and limitations of the existing approaches to make them suitable for real-world applications.

CHAPTER 3

Work Done

3.1 Inferencing on Pretrained Weights

Inference on pre-trained weights is a crucial step in the object detection pipelines of state-of-the-art models such as YOLOv7 and Mask R-CNN. These models are trained on massive datasets with millions of annotated images, allowing them to learn rich representations of objects and scenes. During inference, the pre-trained weights are loaded into the model, and the model processes input images to generate predictions. In YOLOv7, the model employs a single convolutional neural network (CNN) architecture to predict bounding box coordinates, object classes, and confidence scores for multiple objects in an image simultaneously, providing real-time object detection. On the other hand, Mask R-CNN extends YOLOv7 by adding a mask branch, which predicts object masks in addition to bounding boxes and object classes. Inference on pre-trained weights enables these models to accurately detect objects and generate masks in real-time, making them powerful tools for a wide range of applications such as autonomous driving, surveillance, and augmented reality.

3.1.1. Inferencing on YOLOv7

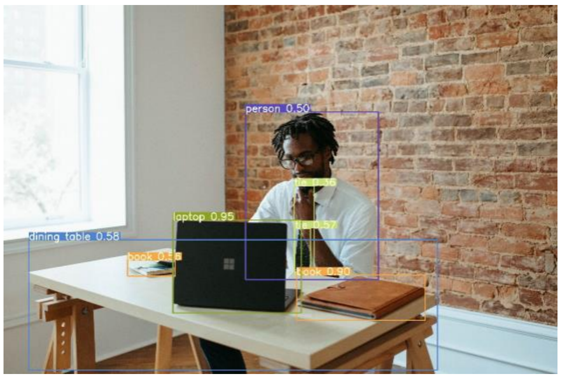


Figure 3.1: YOLOv7 on GPU (DGX Workstation)

Object	**Confidence Level**
Tie	0.358154
Person	0.501953
Book	0.555664
Tie	0.572266
Dining Table	0.580566
Book	0.897461
Laptop	0.951172

Table 3.1: YOLOv7 on GPU (DGX Workstation) Confidence Levels

3.1.2. Inferencing on Mask R-CNN

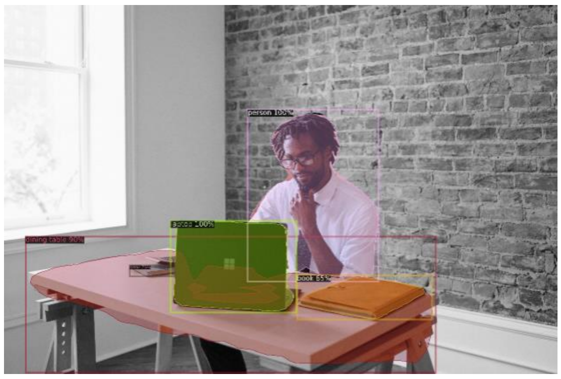


Figure 3.2: Mask R-CNN on GPU (DGX Workstation)

Object	Confidence Level
Person	1.000000
Book	0.850000
Dining Table	0.900000
Book	0.750000
Laptop	1.000000

Table 3.2: Mask R-CNN on GPU (DGX Workstation) Confidence Levels

3.2 Selection of Algorithm

Choosing between YOLOv7 and Mask R-CNN depends on the specific requirements of the application and the trade-offs between different factors. Here are some reasons why one might choose YOLOv7 over Mask R-CNN:

1. Real-time object detection: YOLOv7 is designed for real-time object detection, capable of processing images at a high frame rate, making it suitable for applications that require low-latency object detection, such as autonomous vehicles or robotics.

2. Simplicity and speed: YOLOv7 follows a single CNN architecture, making it relatively simple and efficient to implement and deploy. Its single-stage detection approach makes it faster compared to Mask R-CNN, which follows a two-stage detection approach.

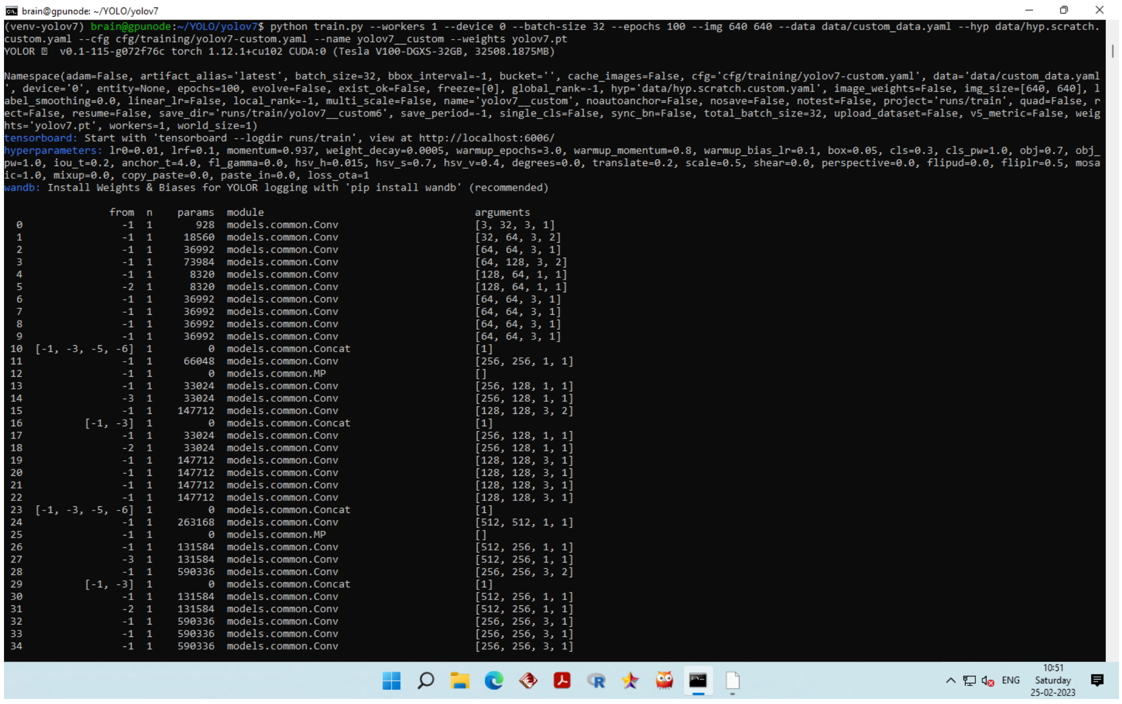
3. Object detection without masks: If the application does not require precise pixel-level segmentation of objects, YOLOv7 may be a more efficient choice as it focuses on bounding box detection and object classification, without the overhead of mask prediction.

4. Resource-constrained environments: YOLOv7 may be preferable in resource-constrained environments where limited computing power or memory is available, as it is designed to be efficient and fast, making it more suitable for deployment on edge devices.

5. High object density scenarios: YOLOv7 is known for its ability to handle high object density scenarios, where multiple objects are densely packed in an image, making it a good choice for applications such as crowd analysis or object counting.

However, it's important to note that Mask R-CNN has its strengths as well. For applications that require pixel-level segmentation, precise object masks, and higher accuracy, Mask R-CNN may be a better choice. It provides more detailed and accurate object masks, making it suitable for applications that demand fine-grained object understanding, such as medical imaging or image manipulation. Ultimately, the choice between YOLOv7 and Mask R-CNN depends on the specific requirements and constraints of the application at hand.

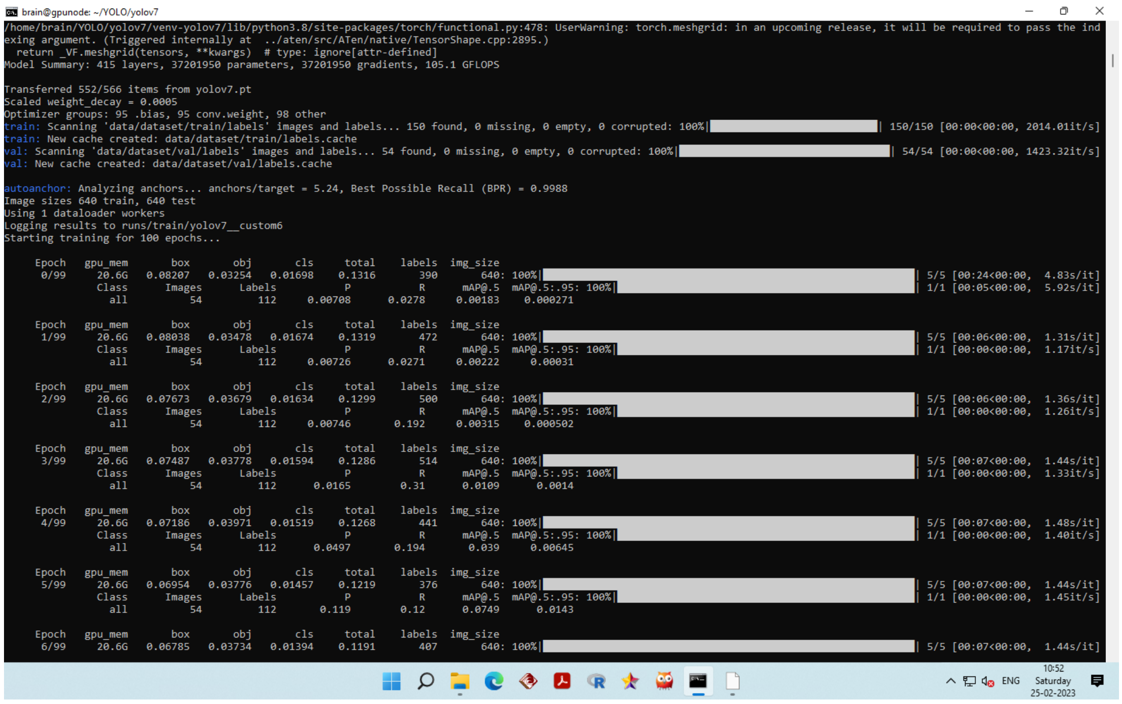
3.3 Retraining on Car Person Dataset



brain@gpunode: ~/YOLO/yolov7

```
(venv-yolov7) brain@gpunode:~/YOLO/yolov7$ python train.py --workers 1 --device 0 --batch-size 32 --epochs 100 --img 640 640 --data data/custom_data.yaml --hyp data/hyp.scratch.
custom.yaml --cfg cfg/training/yolov7-custom.yaml --name yolov7__custom --weights yolov7.pt
YOLOR ☐  v0.1-115-g072f76c torch 1.12.1+cu102 CUDA:0 (Tesla V100-DGXS-32GB, 32508.1875MB)

Namespace(adam=False, artifact_alias='latest', batch_size=32, bbox_interval=-1, bucket='', cache_images=False, cfg='cfg/training/yolov7-custom.yaml', data='data/custom_data.yaml
', device='0', entity=None, epochs=100, evolve=False, exist_ok=False, freeze=[0], global_rank=-1, hyp='data/hyp.scratch.custom.yaml', image_weights=False, img_size=[640, 640], l
abel_smoothing=0.0, linear_lr=False, local_rank=-1, multi_scale=False, name='yolov7__custom', noautoanchor=False, nosave=False, notest=False, project='runs/train', quad=False, r
ect=False, resume=False, save_dir='runs/train/yolov7__custom6', save_period=-1, single_cls=False, sync_bn=False, total_batch_size=32, upload_dataset=False, v5_metric=False, weig
hts='yolov7.pt', workers=1, world_size=1)
tensorboard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
hyperparameters: lr0=0.01, lrf=0.1, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.3, cls_pw=1.0, obj=0.7, obj_
pw=1.0, iou_t=0.2, anchor_t=4.0, fl_gamma=0.0, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.2, scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, mosa
ic=1.0, mixup=0.0, copy_paste=0.0, paste_in=0.0, loss_ota=1
wandb: Install Weights & Biases for YOLOR logging with 'pip install wandb' (recommended)

                 from  n    params  module                                  arguments
  0                -1  1       928  models.common.Conv                      [3, 32, 3, 1]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  1      8320  models.common.Conv                      [128, 64, 1, 1]
  5                -2  1      8320  models.common.Conv                      [128, 64, 1, 1]
  6                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  7                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  8                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
  9                -1  1     36992  models.common.Conv                      [64, 64, 3, 1]
 10  [-1, -3, -5, -6]  1         0  models.common.Concat                    [1]
 11                -1  1     66048  models.common.Conv                      [256, 256, 1, 1]
 12                -1  1         0  models.common.MP                        []
 13                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 14                -3  1     33024  models.common.Conv                      [256, 128, 1, 1]
 15                -1  1    147712  models.common.Conv                      [128, 128, 3, 2]
 16          [-1, -3]  1         0  models.common.Concat                    [1]
 17                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 18                -2  1     33024  models.common.Conv                      [256, 128, 1, 1]
 19                -1  1    147712  models.common.Conv                      [128, 128, 3, 1]
 20                -1  1    147712  models.common.Conv                      [128, 128, 3, 1]
 21                -1  1    147712  models.common.Conv                      [128, 128, 3, 1]
 22                -1  1    147712  models.common.Conv                      [128, 128, 3, 1]
 23  [-1, -3, -5, -6]  1         0  models.common.Concat                    [1]
 24                -1  1    263168  models.common.Conv                      [512, 512, 1, 1]
 25                -1  1         0  models.common.MP                        []
 26                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
 27                -3  1    131584  models.common.Conv                      [512, 256, 1, 1]
 28                -1  1    590336  models.common.Conv                      [256, 256, 3, 2]
 29          [-1, -3]  1         0  models.common.Concat                    [1]
 30                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
 31                -2  1    131584  models.common.Conv                      [512, 256, 1, 1]
 32                -1  1    590336  models.common.Conv                      [256, 256, 3, 1]
 33                -1  1    590336  models.common.Conv                      [256, 256, 3, 1]
 34                -1  1    590336  models.common.Conv                      [256, 256, 3, 1]
```

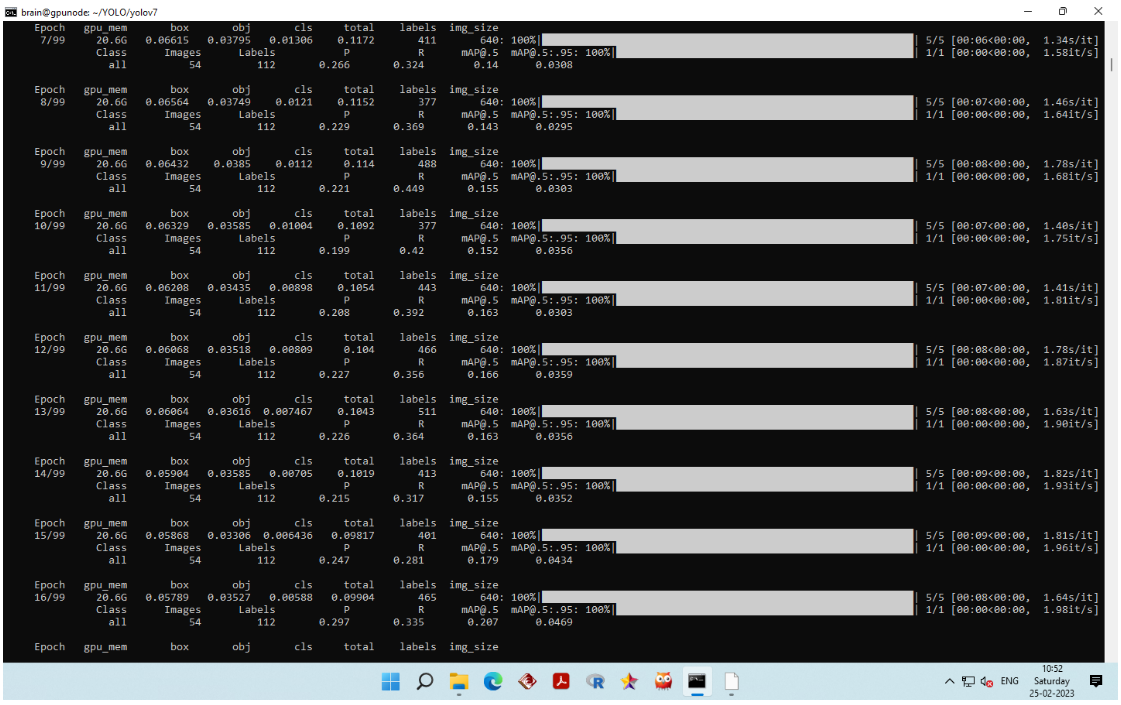
ENG 10:51 Saturday 25-02-2023



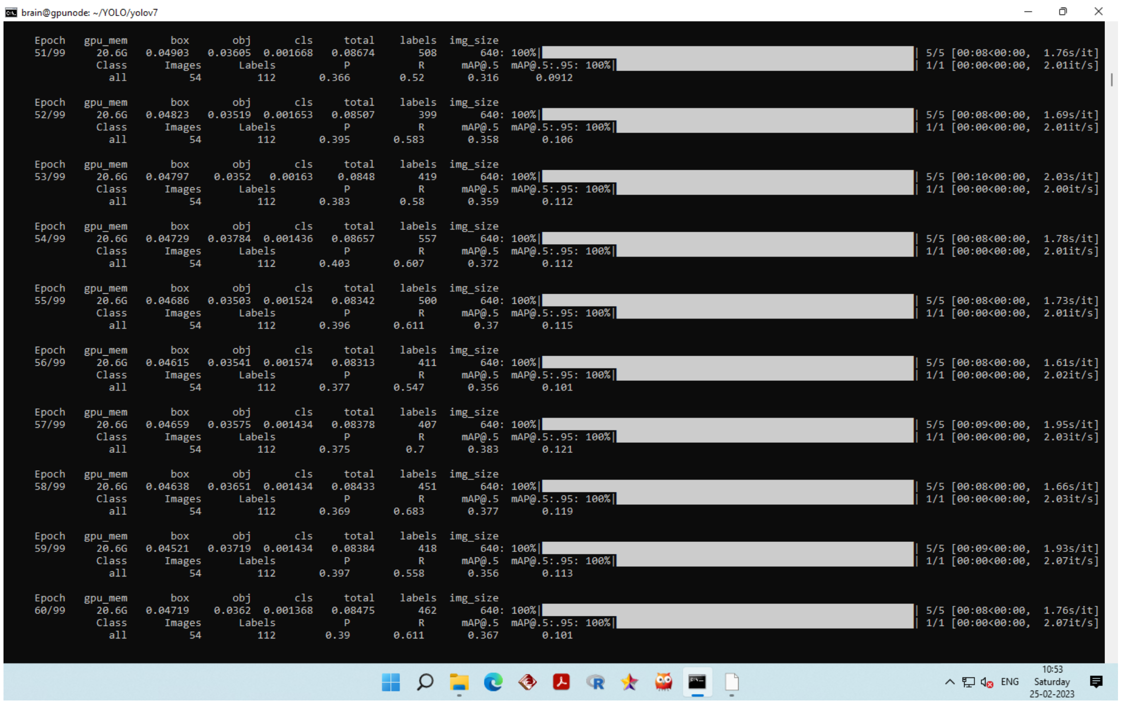
brain@gpunode: ~/YOLO/yolov7

```
/home/brain/YOLO/yolov7/venv-yolov7/lib/python3.8/site-packages/torch/functional.py:478: UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass the ind
exing argument. (Triggered internally at  ../aten/src/ATen/native/TensorShape.cpp:2895.)
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
Model Summary: 415 layers, 37201950 parameters, 37201950 gradients, 105.1 GFLOPS

Transferred 552/566 items from yolov7.pt
Scaled weight_decay = 0.0005
Optimizer groups: 95 .bias, 95 conv.weight, 98 other
train: Scanning 'data/dataset/train/labels' images and labels... 150 found, 0 missing, 0 empty, 0 corrupted: 100%|██████████| 150/150 [00:00<00:00, 2014.01it/s]
train: New cache created: data/dataset/train/labels.cache
val: Scanning 'data/dataset/val/labels' images and labels... 54 found, 0 missing, 0 empty, 0 corrupted: 100%|██████████| 54/54 [00:00<00:00, 1423.32it/s]
val: New cache created: data/dataset/val/labels.cache

autoanchor: Analyzing anchors... anchors/target = 5.24, Best Possible Recall (BPR) = 0.9988
Image sizes 640 train, 640 test
Using 1 dataloader workers
Logging results to runs/train/yolov7__custom6
Starting training for 100 epochs...

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      0/99     20.6G   0.08207   0.03254   0.01698    0.1316       390       640: 100%|██████████| 5/5 [00:24<00:00,  4.83s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:05<00:00,  5.92s/it]
                 all          54         112     0.00708      0.0278     0.00183    0.000271

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      1/99     20.6G   0.08038   0.03478   0.01674    0.1319       472       640: 100%|██████████| 5/5 [00:06<00:00,  1.31s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  1.17it/s]
                 all          54         112     0.00726      0.0271     0.00222     0.00031

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      2/99     20.6G   0.07673   0.03679   0.01634    0.1299       500       640: 100%|██████████| 5/5 [00:06<00:00,  1.36s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  1.26it/s]
                 all          54         112     0.00746       0.192     0.00315    0.000502

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      3/99     20.6G   0.07487   0.03778   0.01594    0.1286       514       640: 100%|██████████| 5/5 [00:07<00:00,  1.44s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  1.33it/s]
                 all          54         112      0.0165        0.31      0.0109      0.0014

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      4/99     20.6G   0.07186   0.03971   0.01519    0.1268       441       640: 100%|██████████| 5/5 [00:07<00:00,  1.48s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  1.40it/s]
                 all          54         112      0.0497       0.194       0.039     0.00645

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      5/99     20.6G   0.06954   0.03776   0.01457    0.1219       376       640: 100%|██████████| 5/5 [00:07<00:00,  1.44s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  1.45it/s]
                 all          54         112       0.119        0.12      0.0749      0.0143

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      6/99     20.6G   0.06785   0.03734   0.01394    0.1191       407       640: 100%|██████████| 5/5 [00:07<00:00,  1.44s/it]
```

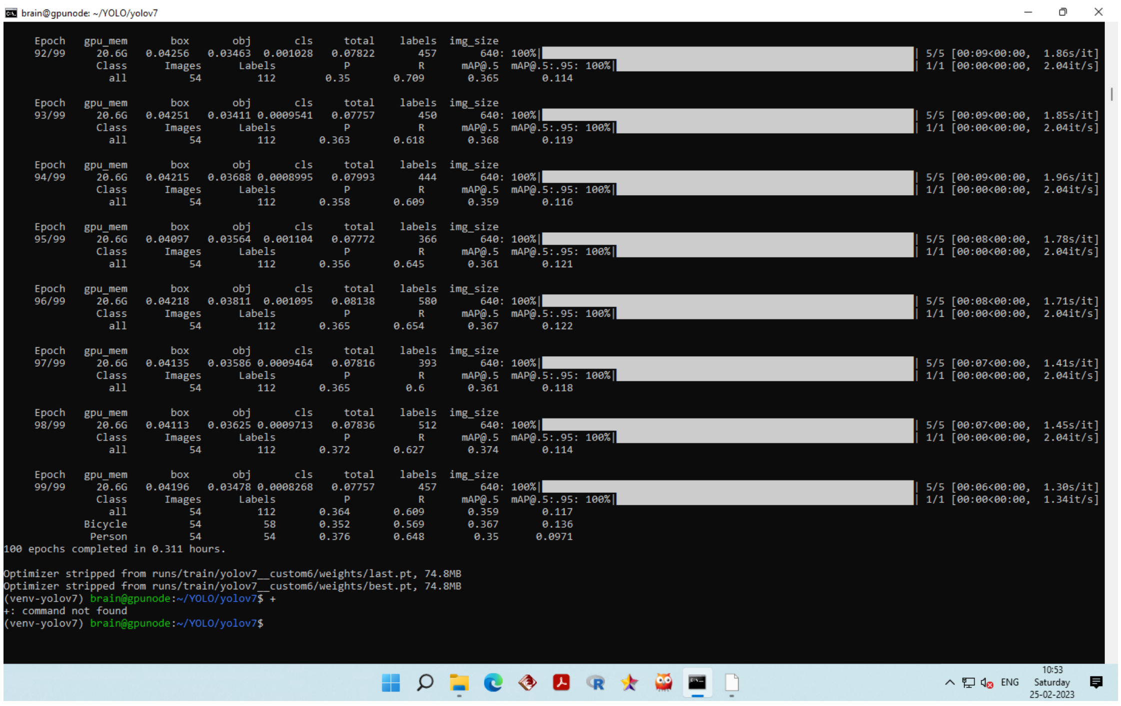
ENG 10:52 Saturday 25-02-2023



```
     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      7/99     20.6G   0.06615   0.03795   0.01306    0.1172       411       640: 100%|██████████| 5/5 [00:06<00:00,  1.34s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  1.58it/s]
                 all          54         112       0.266       0.324        0.14      0.0308

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      8/99     20.6G   0.06564   0.03749    0.0121    0.1152       377       640: 100%|██████████| 5/5 [00:07<00:00,  1.46s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  1.64it/s]
                 all          54         112       0.229       0.369       0.143      0.0295

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      9/99     20.6G   0.06432    0.0385    0.0112     0.114       488       640: 100%|██████████| 5/5 [00:08<00:00,  1.78s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  1.68it/s]
                 all          54         112       0.221       0.449       0.155      0.0303

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     10/99     20.6G   0.06329   0.03585   0.01004    0.1092       377       640: 100%|██████████| 5/5 [00:07<00:00,  1.40s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  1.75it/s]
                 all          54         112       0.199        0.42       0.152      0.0356

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     11/99     20.6G   0.06208   0.03435   0.00898    0.1054       443       640: 100%|██████████| 5/5 [00:07<00:00,  1.41s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  1.81it/s]
                 all          54         112       0.208       0.392       0.163      0.0303

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     12/99     20.6G   0.06068   0.03518   0.00809     0.104       466       640: 100%|██████████| 5/5 [00:08<00:00,  1.78s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  1.87it/s]
                 all          54         112       0.227       0.356       0.166      0.0359

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     13/99     20.6G   0.06064   0.03616  0.007467    0.1043       511       640: 100%|██████████| 5/5 [00:08<00:00,  1.63s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  1.90it/s]
                 all          54         112       0.226       0.364       0.163      0.0356

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     14/99     20.6G   0.05904   0.03585   0.00705    0.1019       413       640: 100%|██████████| 5/5 [00:09<00:00,  1.82s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  1.93it/s]
                 all          54         112       0.215       0.317       0.155      0.0352

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     15/99     20.6G   0.05868   0.03306  0.006436   0.09817       401       640: 100%|██████████| 5/5 [00:09<00:00,  1.81s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  1.96it/s]
                 all          54         112       0.247       0.281       0.179      0.0434

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     16/99     20.6G   0.05789   0.03527   0.00588   0.09904       465       640: 100%|██████████| 5/5 [00:08<00:00,  1.64s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  1.98it/s]
                 all          54         112       0.297       0.335       0.207      0.0469

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
```



```
     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     51/99     20.6G   0.04903   0.03605  0.001668   0.08674       508       640: 100%|██████████| 5/5 [00:08<00:00,  1.76s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  2.01it/s]
                 all          54         112       0.366        0.52       0.316      0.0912

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     52/99     20.6G   0.04823   0.03519  0.001653   0.08507       399       640: 100%|██████████| 5/5 [00:08<00:00,  1.69s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  2.01it/s]
                 all          54         112       0.395       0.583       0.358       0.106

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     53/99     20.6G   0.04797    0.0352   0.00163    0.0848       419       640: 100%|██████████| 5/5 [00:10<00:00,  2.03s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  2.00it/s]
                 all          54         112       0.383        0.58       0.359       0.112

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     54/99     20.6G   0.04729   0.03784  0.001436   0.08657       557       640: 100%|██████████| 5/5 [00:08<00:00,  1.78s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  2.01it/s]
                 all          54         112       0.403       0.607       0.372       0.112

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     55/99     20.6G   0.04686   0.03503  0.001524   0.08342       500       640: 100%|██████████| 5/5 [00:08<00:00,  1.73s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  2.01it/s]
                 all          54         112       0.396       0.611        0.37       0.115

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     56/99     20.6G   0.04615   0.03541  0.001574   0.08313       411       640: 100%|██████████| 5/5 [00:08<00:00,  1.61s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  2.02it/s]
                 all          54         112       0.377       0.547       0.356       0.101

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     57/99     20.6G   0.04659   0.03575  0.001434   0.08378       407       640: 100%|██████████| 5/5 [00:09<00:00,  1.95s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  2.03it/s]
                 all          54         112       0.375         0.7       0.383       0.121

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     58/99     20.6G   0.04638   0.03651  0.001434   0.08433       451       640: 100%|██████████| 5/5 [00:08<00:00,  1.66s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  2.03it/s]
                 all          54         112       0.369       0.683       0.377       0.119

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     59/99     20.6G   0.04521   0.03719  0.001434   0.08384       418       640: 100%|██████████| 5/5 [00:09<00:00,  1.93s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  2.07it/s]
                 all          54         112       0.397       0.558       0.356       0.113

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     60/99     20.6G   0.04719    0.0362  0.001368   0.08475       462       640: 100%|██████████| 5/5 [00:08<00:00,  1.76s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|██████████| 1/1 [00:00<00:00,  2.07it/s]
                 all          54         112        0.39       0.611       0.367       0.101
```



```
     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     92/99     20.6G   0.04256   0.03463  0.001028   0.07822       457       640: 100%|██████████| 5/5 [00:09<00:00,  1.86s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|█| 1/1 [00:00<00:00,  2.04it/s]
                 all          54         112        0.35       0.709       0.365       0.114

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     93/99     20.6G   0.04251   0.03411 0.0009541   0.07757       450       640: 100%|██████████| 5/5 [00:09<00:00,  1.85s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|█| 1/1 [00:00<00:00,  2.04it/s]
                 all          54         112       0.363       0.618       0.368       0.119

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     94/99     20.6G   0.04215   0.03688 0.0008995   0.07993       444       640: 100%|██████████| 5/5 [00:09<00:00,  1.96s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|█| 1/1 [00:00<00:00,  2.04it/s]
                 all          54         112       0.358       0.609       0.359       0.116

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     95/99     20.6G   0.04097   0.03564  0.001104   0.07772       366       640: 100%|██████████| 5/5 [00:08<00:00,  1.78s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|█| 1/1 [00:00<00:00,  2.04it/s]
                 all          54         112       0.356       0.645       0.361       0.121

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     96/99     20.6G   0.04218   0.03811  0.001095   0.08138       580       640: 100%|██████████| 5/5 [00:08<00:00,  1.71s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|█| 1/1 [00:00<00:00,  2.04it/s]
                 all          54         112       0.365       0.654       0.367       0.122

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     97/99     20.6G   0.04135   0.03586 0.0009464   0.07816       393       640: 100%|██████████| 5/5 [00:07<00:00,  1.41s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|█| 1/1 [00:00<00:00,  2.04it/s]
                 all          54         112       0.365         0.6       0.361       0.118

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     98/99     20.6G   0.04113   0.03625 0.0009713   0.07836       512       640: 100%|██████████| 5/5 [00:07<00:00,  1.45s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|█| 1/1 [00:00<00:00,  2.04it/s]
                 all          54         112       0.372       0.627       0.374       0.114

     Epoch   gpu_mem       box       obj       cls     total    labels  img_size
     99/99     20.6G   0.04196   0.03478 0.0008268   0.07757       457       640: 100%|██████████| 5/5 [00:06<00:00,  1.30s/it]
               Class      Images      Labels           P           R      mAP@.5  mAP@.5:.95: 100%|█| 1/1 [00:00<00:00,  1.34it/s]
                 all          54         112       0.364       0.609       0.359       0.117
             Bicycle          54          58       0.352       0.569       0.367       0.136
              Person          54          54       0.376       0.648        0.35      0.0971
100 epochs completed in 0.311 hours.

Optimizer stripped from runs/train/yolov7__custom6/weights/last.pt, 74.8MB
Optimizer stripped from runs/train/yolov7__custom6/weights/best.pt, 74.8MB
(venv-yolov7) brain@gpunode:~/YOLO/yolov7$ +
+: command not found
(venv-yolov7) brain@gpunode:~/YOLO/yolov7$
```

Fig: 3.3 YOLOv7 Retraining

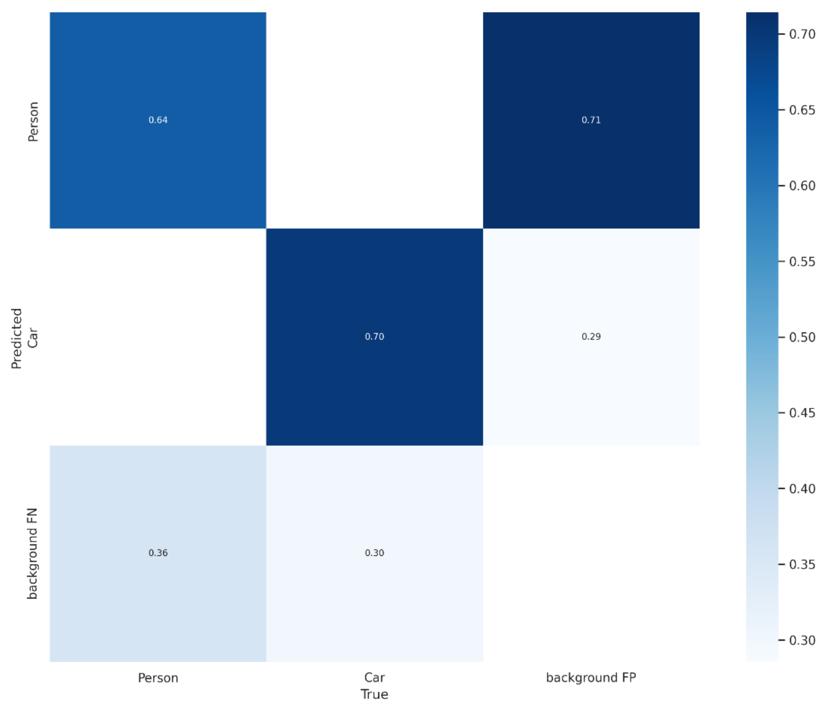


Fig:3.4 Confusion Matrix for 100 Epochs

For Car

TP=0.70	FP=0.29
FN=0.30	TN=1.71

For Person

TP=0.64	FP=0.71
FN=0.36	TN=1.29

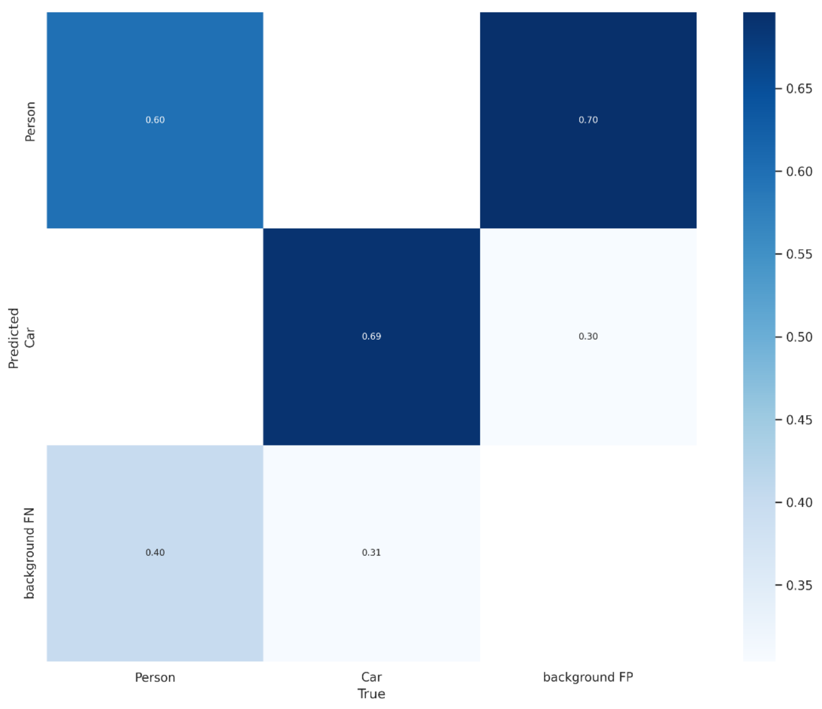


Fig:3.5 Confusion Matrix for 200 Epochs

For Car

TP=0.69	FP=0.30
FN=0.31	TN=1.70

For Person

TP=0.60	FP=0.70
FN=0.40	TN=1.30

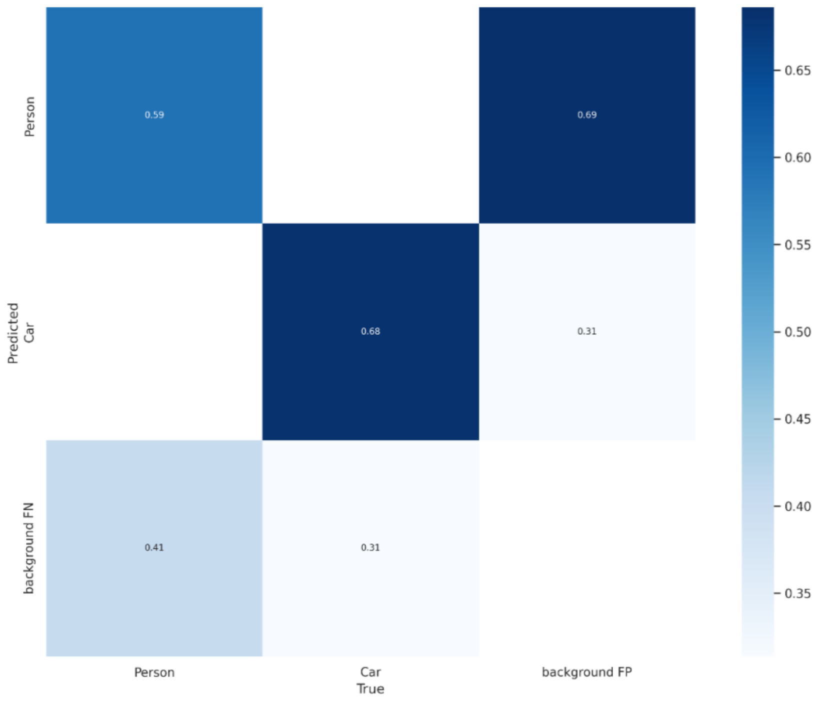


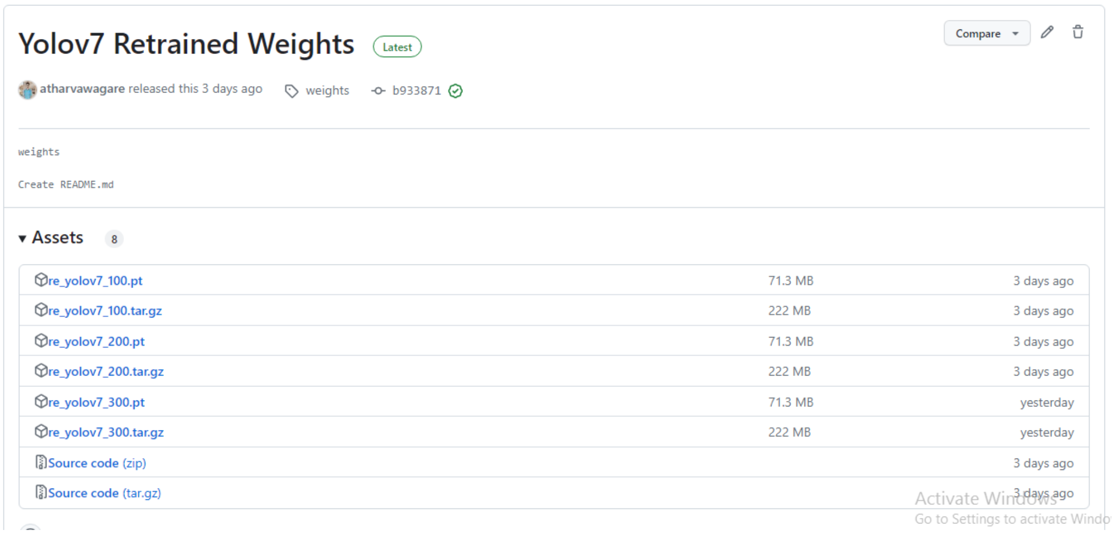
Fig:3.6 Confusion Matrix for 300 Epochs

For Car

TP=0.68	FP=0.31
FN=0.31	TN=1.69

For Person

TP=0.59	FP=0.69
FN=0.41	TN=1.30



Yolov7 Retrained Weights Latest

atharvawagare released this 3 days ago weights b933871

```
weights

Create README.md
```

Assets 8

re_yolov7_100.pt	71.3 MB	3 days ago
re_yolov7_100.tar.gz	222 MB	3 days ago
re_yolov7_200.pt	71.3 MB	3 days ago
re_yolov7_200.tar.gz	222 MB	3 days ago
re_yolov7_300.pt	71.3 MB	yesterday
re_yolov7_300.tar.gz	222 MB	yesterday
Source code (zip)		3 days ago
Source code (tar.gz)		3 days ago

The weights generated after retraining are available at Github (https://github.com/atharvawagare/yolov7_retrained/releases/tag/weights)

3.4 Implementing Tracking Approach

Object tracking is a critical task in traffic systems, as it allows for the monitoring of traffic flow and the identification of abnormal behaviour such as traffic violations or congestion. In order to achieve this, one approach is to detect objects in each frame and compare them with detections from the previous frame. This can be accomplished using an object detection algorithm such as the YOLOv7 (You Only Look Once) algorithm, which is capable of identifying objects in the scene.

Once objects are detected, a unique identifier is assigned to each object, and the position of the objects is recorded. This enables subsequent frames to be compared with the previous frame to determine if there are any changes in the objects' positions. If a new object is detected, it is compared to the objects in the previous frame to determine if it is the same object. To do this, the Euclidean distance between the centres of each object is calculated. If the Euclidean distance is less than a certain threshold value, the object is considered as the same as in the previous frame and is assigned the same ID. This process is repeated in subsequent frames, allowing for the tracking of objects over time.

This approach to object tracking can greatly improve the accuracy and efficiency of traffic systems, as it provides real-time data on traffic flow and enables the identification of abnormal behaviour. The use of the YOLOv7 algorithm in particular is beneficial as it is able to accurately identify objects in complex scenes, such as those found in traffic systems. In addition, the use of OpenCV tools can further enhance the effectiveness of the tracking process by providing additional functionality, such as the ability to filter out noise and improve the accuracy of object detection.

Overall, object tracking is an essential task in traffic systems, and the use of advanced algorithms such as YOLOv7 and OpenCV can greatly improve the accuracy and efficiency of this process. By accurately identifying objects and tracking them over time, traffic systems can be optimized for improved flow and safety. Additionally, the ability to detect abnormal behaviour such as traffic violations can aid in the prevention of accidents and other negative outcomes.

The source code to implement tracking approach is as:

```
import cv2
import numpy as np
from detection_function_class import Detection
import math
from PIL import Image

# Initialize Object Detection
dt = Detection()

cap = cv2.VideoCapture("dataset/videos/los_angeles_small.mp4")

# Initialize count
count = 0
center_points_prev_frame = []

tracking_objects = {}
track_id = 0

while True:
    ret, frame = cap.read()
    count += 1
    if not ret:
        break

    # Point current frame
    center_points_cur_frame = []

    # Detect objects on frame
    boxes = dt.detect(source=frame)
    for box in boxes:
        (x, y, w, h) = box
        cx = int((x + x + w) / 2)
        cy = int((y + y + h) / 2)
        center_points_cur_frame.append((cx, cy))
        print("FRAME N°", count, " ", x, y, w, h)

        cv2.circle(frame, (cx, cy), 5, (0, 0, 255), -1)
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

    # Only at the beginning we compare previous and current frame
    if count <= 2:
        for pt in center_points_cur_frame:
            for pt2 in center_points_prev_frame:
                distance = math.hypot(pt2[0] - pt[0], pt2[1] - pt[1])

                if distance < 20:
                    tracking_objects[track_id] = pt
                    track_id += 1
    else:
        tracking_objects_copy = tracking_objects.copy()
        center_points_cur_frame_copy = center_points_cur_frame.copy()

        for object_id, pt2 in tracking_objects_copy.items():
            object_exists = False
            for pt in center_points_cur_frame_copy:
                distance = math.hypot(pt2[0] - pt[0], pt2[1] - pt[1])

#                 # Update IDs position
                if distance < 20:
                    tracking_objects[object_id] = pt
                    object_exists = True
                    if pt in center_points_cur_frame:
                        center_points_cur_frame.remove(pt)
                    continue

#             # Remove IDs lost
            if not object_exists:
                tracking_objects.pop(object_id)

#         # Add new IDs found
        for pt in center_points_cur_frame:
            tracking_objects[track_id] = pt
            track_id += 1

    for object_id, pt in tracking_objects.items():
        cv2.circle(frame, pt, 5, (0, 0, 255), -1)
        cv2.putText(frame, str(object_id), (pt[0], pt[1] - 7), 0, 1, (0, 0, 255), 2)

    print("Tracking objects")
    print(tracking_objects)


    print("CUR FRAME LEFT PTS")
    print(center_points_cur_frame)
    cv2.imshow("Frame", frame)

#     # Make a copy of the points
    center_points_prev_frame = center_points_cur_frame.copy()

    key = cv2.waitKey(1)
    if key == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

3.5 Execution of Tracking Approach using Retrained Weights

Object tracking in traffic systems is an important area of research that involves monitoring the movement of vehicles and pedestrians in real-time. One approach to object tracking is to use a combination of object detection and tracking algorithms to detect objects in a given frame and track their movement over time. In this approach, objects are first detected in a given frame using a pre-trained object detection model, such as YOLOv7. Once the objects are detected, a unique ID is assigned to each object to track its movement over time.

To improve the accuracy and efficiency of the object tracking system, the object detection model can be fine-tuned on a custom dataset containing images of cars and people. Fine-tuning involves updating the weights of the pre-trained model using the custom dataset, allowing the model to learn to detect the specific objects of interest in the traffic system. This can greatly improve the accuracy of the system, as the pre-trained model may not be able to accurately detect objects in the traffic system without fine-tuning.

Once the object detection model is fine-tuned, it can be used to detect objects in each frame of the video. The detected objects are then compared with the detections from the previous frame using the Euclidean distance method, allowing for object tracking over time. The Euclidean distance method involves measuring the distance between the centres of an object in the current frame and the previous frame. If the distance is less than a threshold value, the object is considered as the same object as in the previous frame.

To optimize the performance of the object detection and tracking algorithms, various techniques can be used, such as parallel processing, data batching, and GPU acceleration. In this implementation, an NVIDIA DGX workstation is used to perform GPU-accelerated object detection and tracking, allowing for faster and more efficient processing of large volumes of video data. The DGX workstation is equipped with multiple NVIDIA GPUs, which can be used in parallel to accelerate object detection and tracking tasks.

By using a retrained object detection model, a custom dataset, and GPU acceleration, the accuracy and efficiency of the tracking system can be greatly improved, making it a powerful tool for monitoring traffic flow and identifying abnormal behaviour in traffic systems. Overall, this approach to object tracking in traffic systems is a promising area of research that has the potential to improve traffic safety and efficiency.

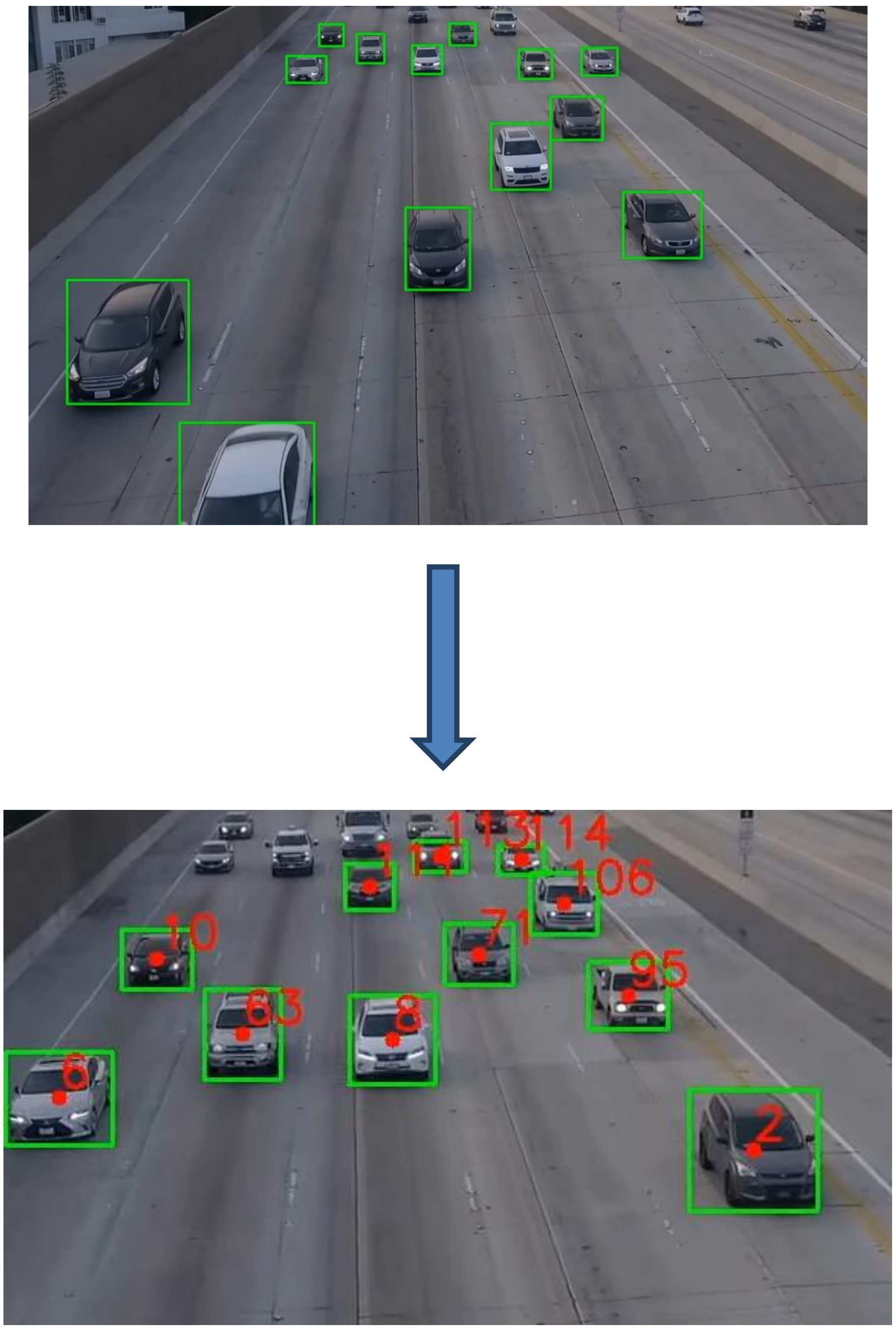


Fig 3.7 Execution of Tracking Approach using Retrained Weights

The source code for detecting objects in each frame of video are as follows:

```
import time
from pathlib import Path

import cv2
import torch
import torch.backends.cudnn as cudnn
import numpy
from numpy import random

from models.experimental import attempt_load
from utils.datasets import LoadStreams, LoadImages, letterbox
from utils.general import check_img_size, check_requirements, check_imshow,
non_max_suppression, apply_classifier, \
    scale_coords, xyxy2xywh, strip_optimizer, set_logging, increment_path
from utils.plots import plot_one_box
from utils.torch_utils import select_device, load_classifier, time_synchronized,
TracedModel

class Detection:
    def __init__(self, save_txt=False, save_conf=False, device="",
weights="yolov7.pt", img_size=640, trace=False, agnostic_nms=False, augment=False,
conf_thres=0.25, iou_thres=0.45, classes=None):
        self.save_txt=save_txt
        self.save_conf=save_conf
        self.device=device
        self.weights=weights
        self.img_size=img_size
        self.trace=not trace
        self.agnostic_nms=agnostic_nms
        self.augment=augment
        self.conf_thres=conf_thres
        self.iou_thres=iou_thres
        self.classes=classes

        # Initialize
        set_logging()
        self.device = select_device(self.device)
        self.half = self.device.type != 'cpu'  # half precision only supported on
CUDA

        # Load model
        self.model = attempt_load(self.weights, map_location=self.device)  # load
FP32 model
        self.stride = int(self.model.stride.max())  # model stride
        self.img_size = check_img_size(self.img_size, s=self.stride)  # check
img_size
```

```
        if self.trace:
            self.model = TracedModel(self.model, self.device, self.img_size)

        if self.half:
            self.model.half()  # to FP16

        # Get names and colors
        names = self.model.module.names if hasattr(self.model, 'module') else
self.model.names
        colors = [[random.randint(0, 255) for _ in range(3)] for _ in names]

        # Run inference
        if self.device.type != 'cpu':
            print("Inside Condition")
            print(self.model(torch.zeros(1, 3, self.img_size,
self.img_size).to(self.device).type_as(next(self.model.parameters()))))
            print(type(self.model(torch.zeros(1, 3, self.img_size,
self.img_size).to(self.device).type_as(next(self.model.parameters())))))
        self.old_img_w = self.old_img_h = self.img_size
        self.old_img_b = 1


    def detect(self, source):
        # Padded resize
        img = letterbox(source, self.img_size, stride=self.stride)[0]

        # Convert
        img = img[:, :, ::-1].transpose(2, 0, 1)  # BGR to RGB, to 3x416x416
        img = numpy.ascontiguousarray(img)

        # for path, img, im0s, vid_cap in dataset:
        img = torch.from_numpy(img).to(self.device)
        img = img.half() if self.half else img.float()  # uint8 to fp16/32
        img /= 255.0  # 0 - 255 to 0.0 - 1.0
        if img.ndimension() == 3:
            img = img.unsqueeze(0)

        # Warmup
        if self.device.type != 'cpu' and (self.old_img_b != img.shape[0] or
self.old_img_h != img.shape[2] or self.old_img_w != img.shape[3]):
            old_img_b = img.shape[0]
            old_img_h = img.shape[2]
            old_img_w = img.shape[3]
            print("Inside Warmup")
            for i in range(3):
                self.model(img, augment=self.augment)[0]

        # Inference
        # t1 = time_synchronized()
        with torch.no_grad():   # Calculating gradients would cause a GPU memory
leak
            print("Inside Inference")
            print(img.shape)
            pred = self.model(img, augment=self.augment)[0]
        # t2 = time_synchronized()

        # Apply NMS
        pred = non_max_suppression(pred, self.conf_thres, self.iou_thres,
classes=self.classes, agnostic=self.agnostic_nms)
        t3 = time_synchronized()

        for i, det in enumerate(pred):  # detections per image
            print("Inside Process Detections")
            if len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_coords(img.shape[2:], det[:, :4],
source.shape).round()
                print("Inside Rescaling")
        return det
```

3.6 Deployment of model on Azure Cloud Platform

In this project, we propose an object detection and tracking approach for traffic analysis using computer vision techniques. Specifically, we use the YOLOv7 object detection model, which is a state-of-the-art deep learning-based method for object detection, to detect and track objects of interest in traffic scenes. However, deploying such a model requires significant computational resources, and traditional methods of deployment may not be optimal for the project's needs.

Therefore, we have chosen to deploy our project on the Azure cloud platform, which provides a range of tools and services for efficient and scalable deployment of machine learning models. We have used Azure Data Factory to create data links and pipelines connecting our frontend and the YOLOv7 object detection model. Azure Data Factory is a cloud-based data integration service that allows us to move data between on-premises and cloud sources, transform and process data, and publish data to our YOLOv7 model for detection.

Furthermore, we have deployed the YOLOv7 model on the Triton inference server, which is an open-source inference server designed to deploy machine learning models on a variety of hardware platforms. Triton provides high-performance inference capabilities and allows us to take advantage of the powerful hardware available on the Azure cloud platform.

The deployment of our project on the Azure cloud platform provides several benefits, such as scalability, reliability, and cost-effectiveness. By using Azure Data Factory, we can easily create and manage data pipelines, which ensures that our frontend and the YOLOv7 model are always connected and running efficiently. Furthermore, by deploying the YOLOv7 model on the Triton inference server, we can take advantage of the powerful hardware available on the Azure cloud platform, which allows us to process large amounts of data quickly and efficiently.

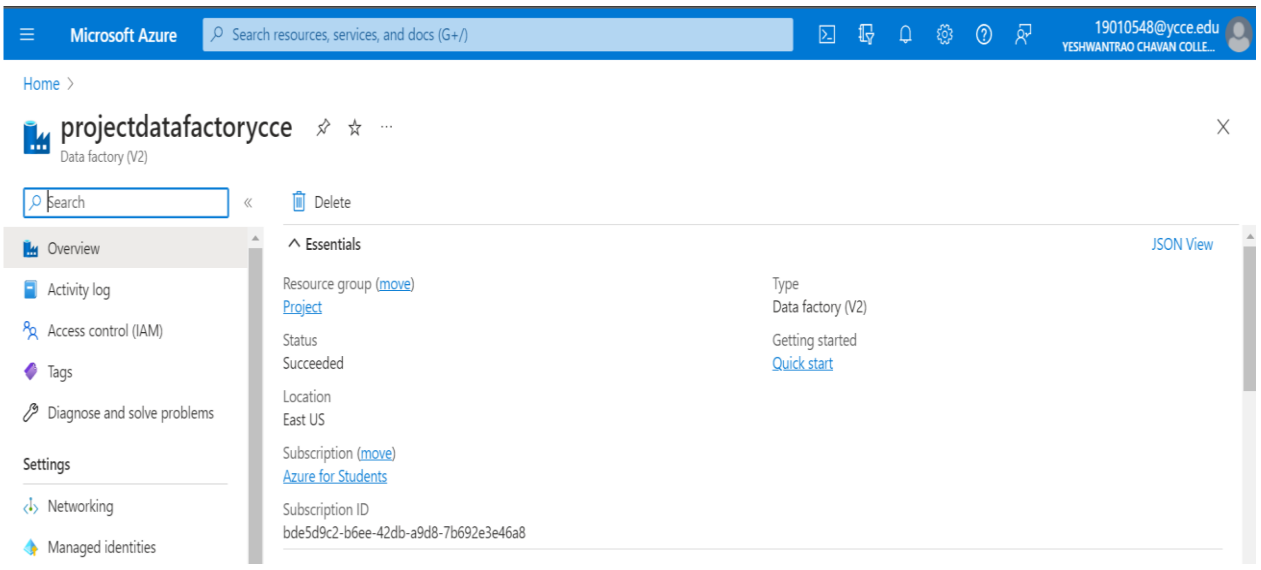
Moreover, the deployment of our project on the Azure cloud platform allows us to take advantage of other Azure services, such as Azure Machine Learning and Azure Cognitive Services. Azure Machine Learning is a cloud-based machine learning service that allows us to train and deploy machine learning models at scale, while Azure Cognitive Services provides pre-built APIs and services for adding intelligent features to our application, such as image recognition and natural language processing.

Another advantage of deploying our project on the Azure cloud platform is the ease of collaboration and integration with other technologies. Azure provides integration with various open-source technologies, such as Kubernetes and TensorFlow, which allows us to create a seamless and efficient workflow for our project. This integration also allows us to work with a wide range of data sources, such as streaming data from cameras or IoT devices, which is crucial for real-time traffic analysis.

Additionally, Azure provides a range of security features, such as authentication and access control, which ensures that our data and models are secure and protected from unauthorized access. This is particularly important for traffic analysis, as the data generated can contain sensitive information about individuals and vehicles, such as license plate numbers and personal identification.

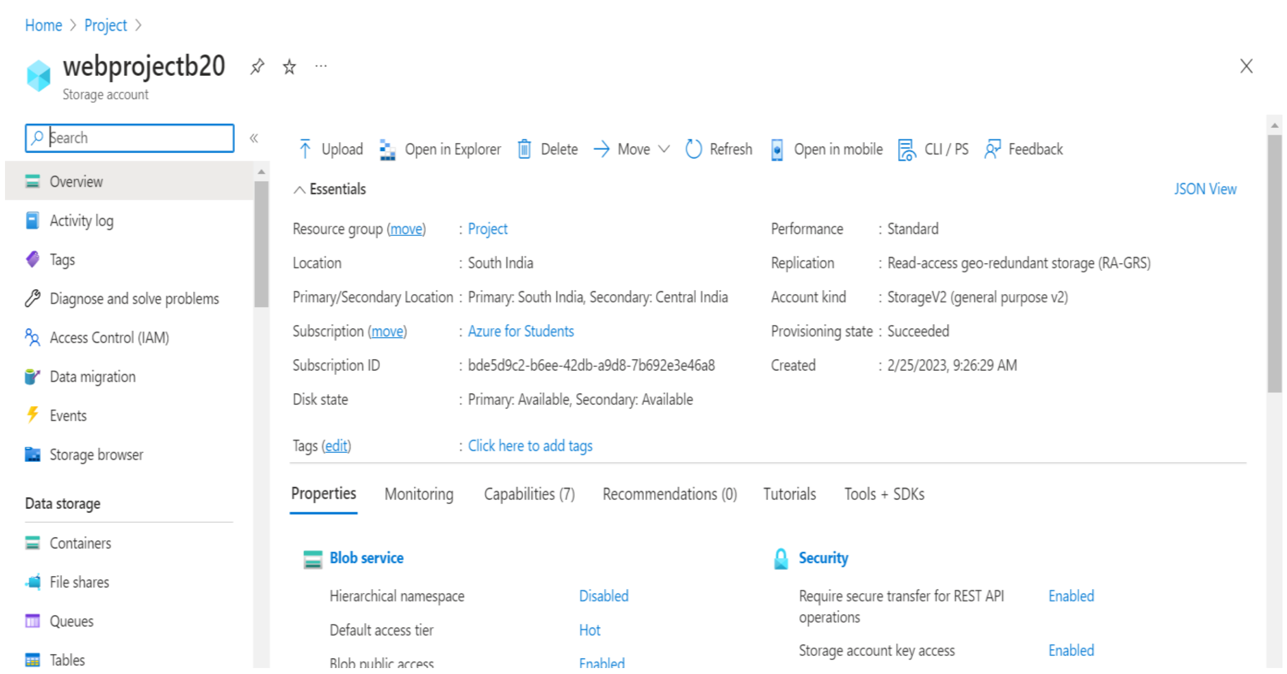
Furthermore, deploying our project on the Azure cloud platform provides cost-effectiveness, as we only pay for the resources we use. This is particularly beneficial for a project like ours, which requires significant computational resources for object detection and tracking. By using the Azure cloud platform, we can take advantage of the powerful hardware available without the need for expensive infrastructure investments.

To create a seamless workflow for our project, we use Azure Data Factory to create data pipelines that connect our frontend to the YOLO v7 object detection model. This allows us to take in streaming data from cameras or IoT devices and process it in real-time using our object detection model. By using Azure Data Factory, we can create efficient data pipelines that can handle large volumes of data and ensure that our model is always up to date with the latest information.



projectdatafactorycce
Data factory (V2)

Resource group (move)	Type
Project	Data factory (V2)
Status	Getting started
Succeeded	Quick start
Location	
East US	
Subscription (move)	
Azure for Students	
Subscription ID	
bde5d9c2-b6ee-42db-a9d8-7b692e3e46a8	



webprojectb20
Storage account

Resource group (move)	: Project	Performance	: Standard
Location	: South India	Replication	: Read-access geo-redundant storage (RA-GRS)
Primary/Secondary Location	: Primary: South India, Secondary: Central India	Account kind	: StorageV2 (general purpose v2)
Subscription (move)	: Azure for Students	Provisioning state	: Succeeded
Subscription ID	: bde5d9c2-b6ee-42db-a9d8-7b692e3e46a8	Created	: 2/25/2023, 9:26:29 AM
Disk state	: Primary: Available, Secondary: Available		
Tags (edit)	: Click here to add tags		

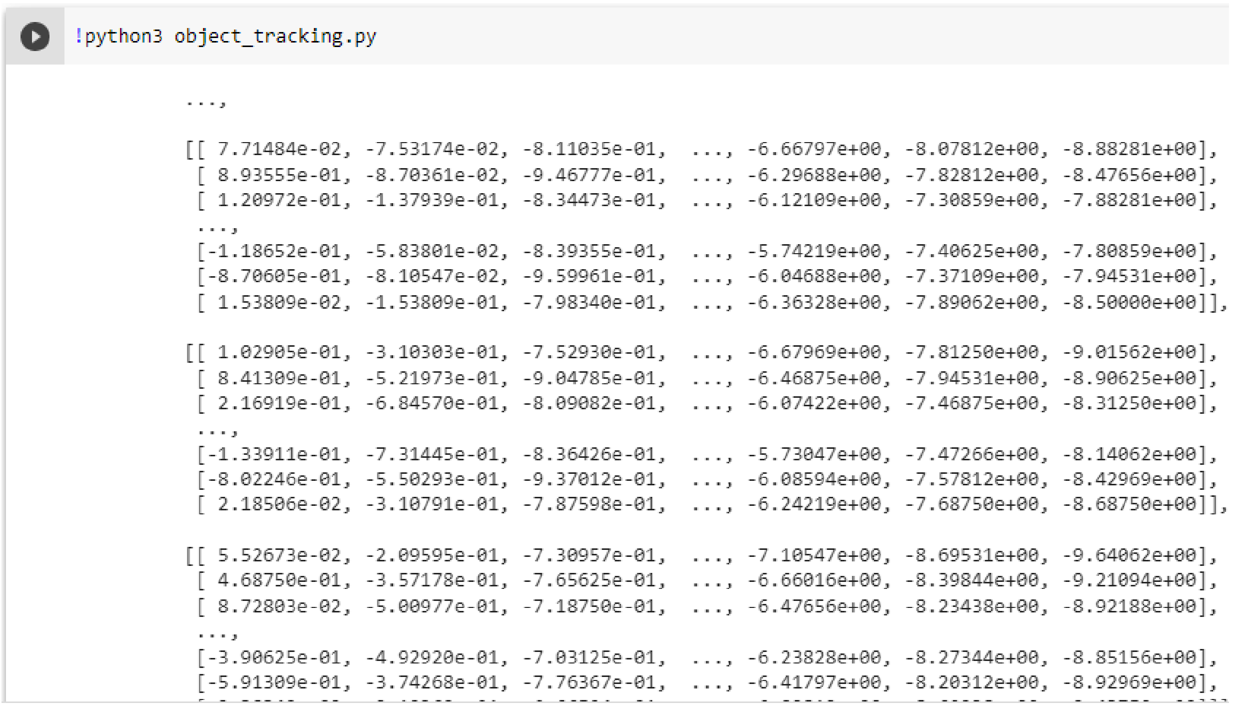
Blob service		Security	
Hierarchical namespace	Disabled	Require secure transfer for REST API operations	Enabled
Default access tier	Hot	Storage account key access	Enabled
Blob public access	Enabled		

Fig 3.8 Deployment of model on Azure cloud platform

CHAPTER 4

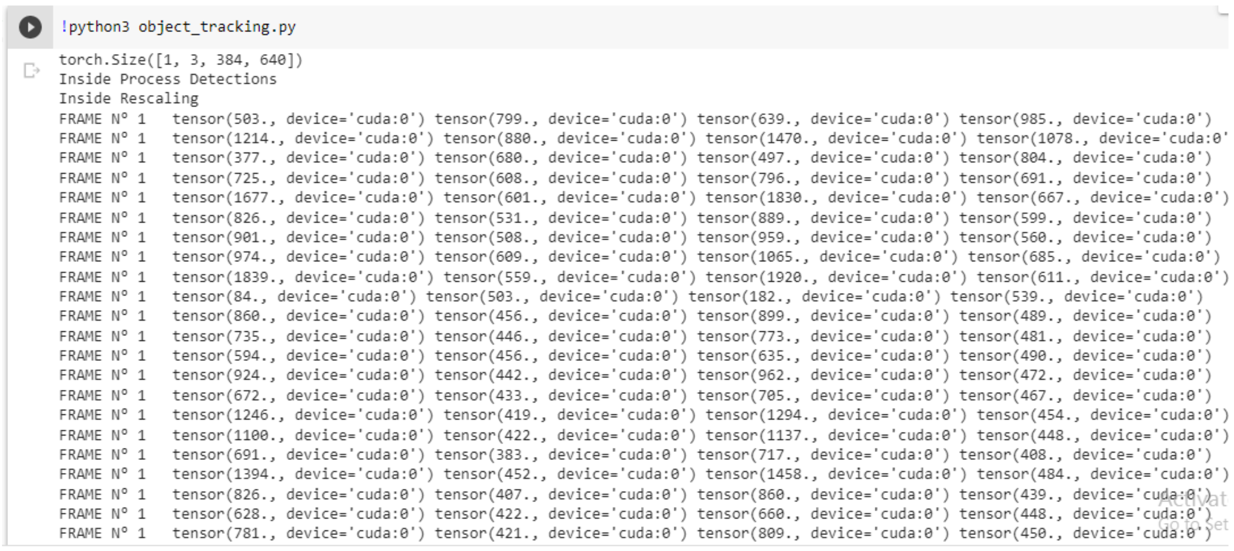
4.1 Results and Discussion

The results of the modified code for detection and implementation of object tracking on the google colab platform is as follows:



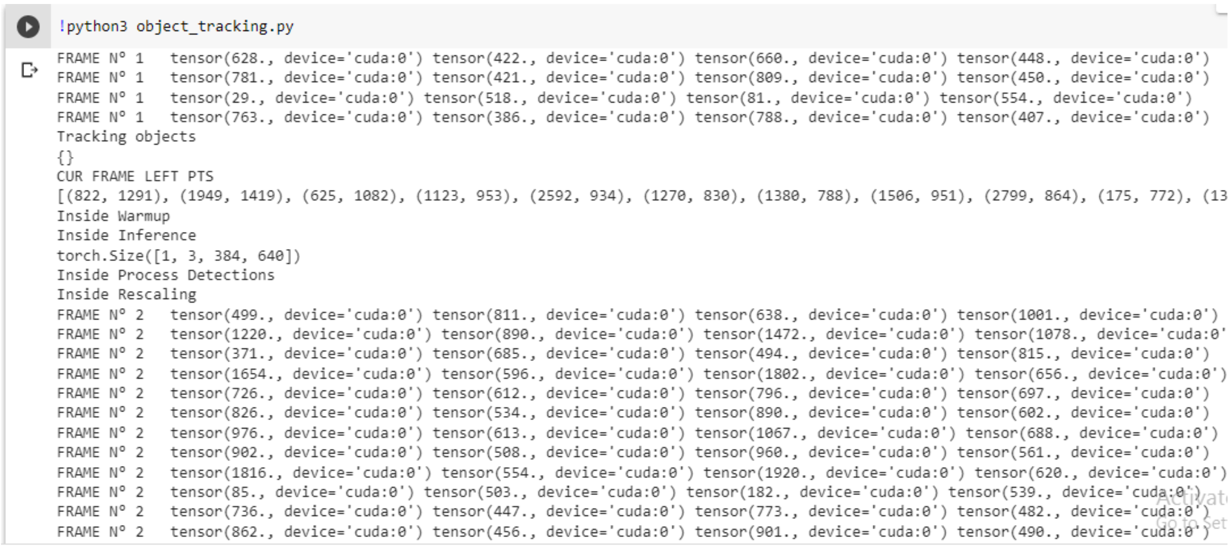
```
!python3 object_tracking.py
```

```
          ...,

          [[ 7.71484e-02, -7.53174e-02, -8.11035e-01,  ..., -6.66797e+00, -8.07812e+00, -8.88281e+00],
           [ 8.93555e-01, -8.70361e-02, -9.46777e-01,  ..., -6.29688e+00, -7.82812e+00, -8.47656e+00],
           [ 1.20972e-01, -1.37939e-01, -8.34473e-01,  ..., -6.12109e+00, -7.30859e+00, -7.88281e+00],
           ...,
           [-1.18652e-01, -5.83801e-02, -8.39355e-01,  ..., -5.74219e+00, -7.40625e+00, -7.80859e+00],
           [-8.70605e-01, -8.10547e-02, -9.59961e-01,  ..., -6.04688e+00, -7.37109e+00, -7.94531e+00],
           [ 1.53809e-02, -1.53809e-01, -7.98340e-01,  ..., -6.36328e+00, -7.89062e+00, -8.50000e+00]],

          [[ 1.02905e-01, -3.10303e-01, -7.52930e-01,  ..., -6.67969e+00, -7.81250e+00, -9.01562e+00],
           [ 8.41309e-01, -5.21973e-01, -9.04785e-01,  ..., -6.46875e+00, -7.94531e+00, -8.90625e+00],
           [ 2.16919e-01, -6.84570e-01, -8.09082e-01,  ..., -6.07422e+00, -7.46875e+00, -8.31250e+00],
           ...,
           [-1.33911e-01, -7.31445e-01, -8.36426e-01,  ..., -5.73047e+00, -7.47266e+00, -8.14062e+00],
           [-8.02246e-01, -5.50293e-01, -9.37012e-01,  ..., -6.08594e+00, -7.57812e+00, -8.42969e+00],
           [ 2.18506e-02, -3.10791e-01, -7.87598e-01,  ..., -6.24219e+00, -7.68750e+00, -8.68750e+00]],

          [[ 5.52673e-02, -2.09595e-01, -7.30957e-01,  ..., -7.10547e+00, -8.69531e+00, -9.64062e+00],
           [ 4.68750e-01, -3.57178e-01, -7.65625e-01,  ..., -6.66016e+00, -8.39844e+00, -9.21094e+00],
           [ 8.72803e-02, -5.00977e-01, -7.18750e-01,  ..., -6.47656e+00, -8.23438e+00, -8.92188e+00],
           ...,
           [-3.90625e-01, -4.92920e-01, -7.03125e-01,  ..., -6.23828e+00, -8.27344e+00, -8.85156e+00],
           [-5.91309e-01, -3.74268e-01, -7.76367e-01,  ..., -6.41797e+00, -8.20312e+00, -8.92969e+00],
```

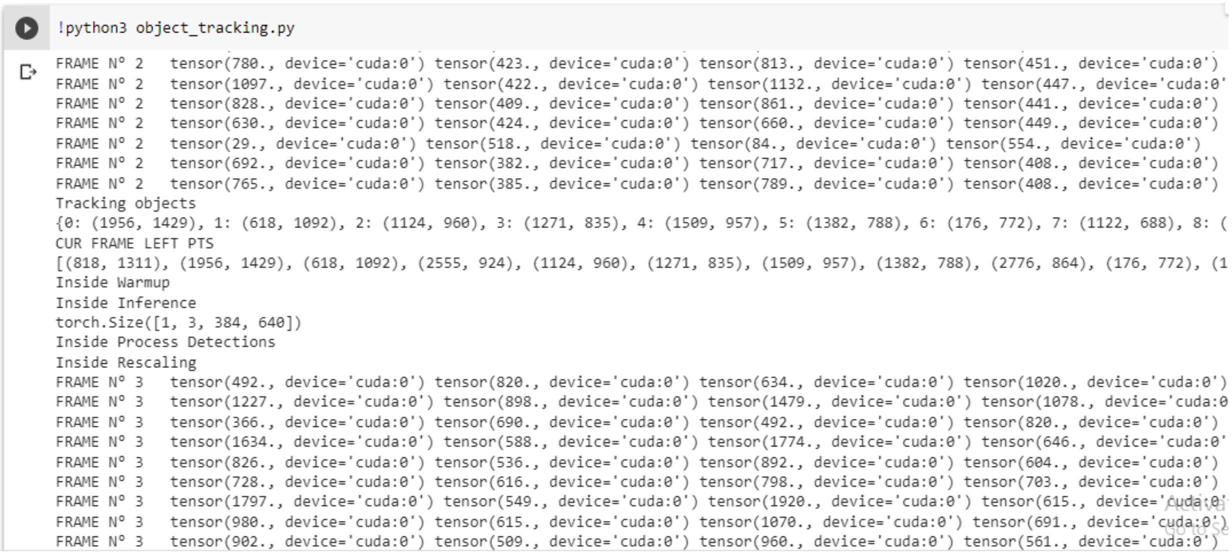


```
!python3 object_tracking.py
```

```
torch.Size([1, 3, 384, 640])
Inside Process Detections
Inside Rescaling
FRAME N° 1   tensor(503., device='cuda:0') tensor(799., device='cuda:0') tensor(639., device='cuda:0') tensor(985., device='cuda:0')
FRAME N° 1   tensor(1214., device='cuda:0') tensor(880., device='cuda:0') tensor(1470., device='cuda:0') tensor(1078., device='cuda:0'
FRAME N° 1   tensor(377., device='cuda:0') tensor(680., device='cuda:0') tensor(497., device='cuda:0') tensor(804., device='cuda:0')
FRAME N° 1   tensor(725., device='cuda:0') tensor(608., device='cuda:0') tensor(796., device='cuda:0') tensor(691., device='cuda:0')
FRAME N° 1   tensor(1677., device='cuda:0') tensor(601., device='cuda:0') tensor(1830., device='cuda:0') tensor(667., device='cuda:0')
FRAME N° 1   tensor(826., device='cuda:0') tensor(531., device='cuda:0') tensor(889., device='cuda:0') tensor(599., device='cuda:0')
FRAME N° 1   tensor(901., device='cuda:0') tensor(508., device='cuda:0') tensor(959., device='cuda:0') tensor(560., device='cuda:0')
FRAME N° 1   tensor(974., device='cuda:0') tensor(609., device='cuda:0') tensor(1065., device='cuda:0') tensor(685., device='cuda:0')
FRAME N° 1   tensor(1839., device='cuda:0') tensor(559., device='cuda:0') tensor(1920., device='cuda:0') tensor(611., device='cuda:0')
FRAME N° 1   tensor(84., device='cuda:0') tensor(503., device='cuda:0') tensor(182., device='cuda:0') tensor(539., device='cuda:0')
FRAME N° 1   tensor(860., device='cuda:0') tensor(456., device='cuda:0') tensor(899., device='cuda:0') tensor(489., device='cuda:0')
FRAME N° 1   tensor(735., device='cuda:0') tensor(446., device='cuda:0') tensor(773., device='cuda:0') tensor(481., device='cuda:0')
FRAME N° 1   tensor(594., device='cuda:0') tensor(456., device='cuda:0') tensor(635., device='cuda:0') tensor(490., device='cuda:0')
FRAME N° 1   tensor(924., device='cuda:0') tensor(442., device='cuda:0') tensor(962., device='cuda:0') tensor(472., device='cuda:0')
FRAME N° 1   tensor(672., device='cuda:0') tensor(433., device='cuda:0') tensor(705., device='cuda:0') tensor(467., device='cuda:0')
FRAME N° 1   tensor(1246., device='cuda:0') tensor(419., device='cuda:0') tensor(1294., device='cuda:0') tensor(454., device='cuda:0')
FRAME N° 1   tensor(1100., device='cuda:0') tensor(422., device='cuda:0') tensor(1137., device='cuda:0') tensor(448., device='cuda:0')
FRAME N° 1   tensor(691., device='cuda:0') tensor(383., device='cuda:0') tensor(717., device='cuda:0') tensor(408., device='cuda:0')
FRAME N° 1   tensor(1394., device='cuda:0') tensor(452., device='cuda:0') tensor(1458., device='cuda:0') tensor(484., device='cuda:0')
FRAME N° 1   tensor(826., device='cuda:0') tensor(407., device='cuda:0') tensor(860., device='cuda:0') tensor(439., device='cuda:0')
FRAME N° 1   tensor(628., device='cuda:0') tensor(422., device='cuda:0') tensor(660., device='cuda:0') tensor(448., device='cuda:0')
FRAME N° 1   tensor(781., device='cuda:0') tensor(421., device='cuda:0') tensor(809., device='cuda:0') tensor(450., device='cuda:0')
```

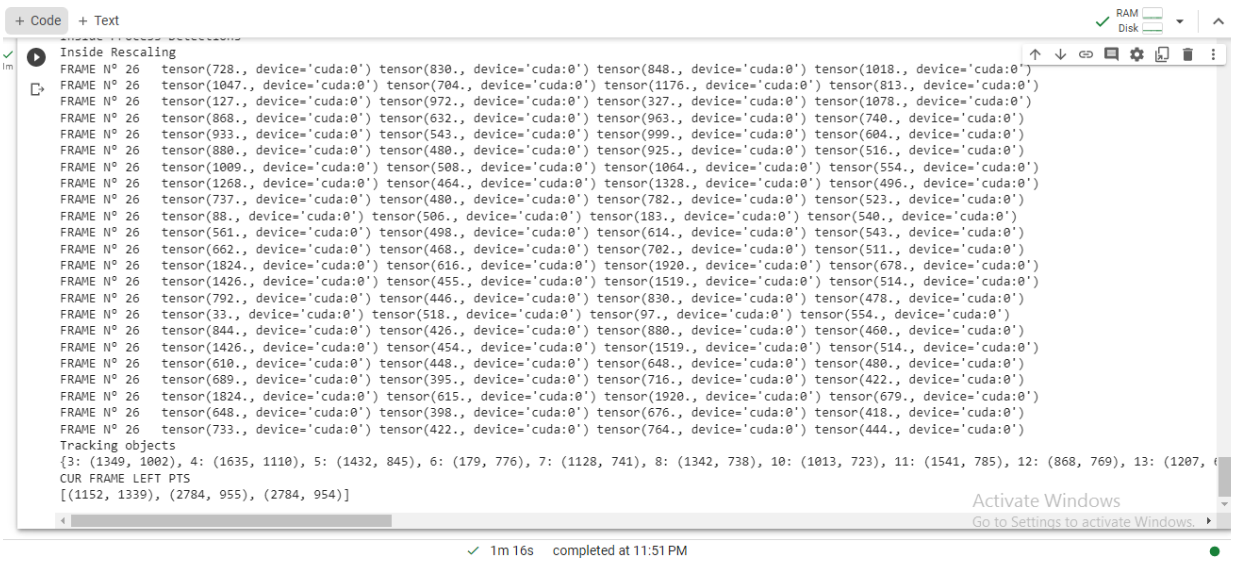


```
!python3 object_tracking.py
FRAME N° 1   tensor(628., device='cuda:0') tensor(422., device='cuda:0') tensor(660., device='cuda:0') tensor(448., device='cuda:0')
FRAME N° 1   tensor(781., device='cuda:0') tensor(421., device='cuda:0') tensor(809., device='cuda:0') tensor(450., device='cuda:0')
FRAME N° 1   tensor(29., device='cuda:0') tensor(518., device='cuda:0') tensor(81., device='cuda:0') tensor(554., device='cuda:0')
FRAME N° 1   tensor(763., device='cuda:0') tensor(386., device='cuda:0') tensor(788., device='cuda:0') tensor(407., device='cuda:0')
Tracking objects
{}
CUR FRAME LEFT PTS
[(822, 1291), (1949, 1419), (625, 1082), (1123, 953), (2592, 934), (1270, 830), (1380, 788), (1506, 951), (2799, 864), (175, 772), (13
Inside Warmup
Inside Inference
torch.Size([1, 3, 384, 640])
Inside Process Detections
Inside Rescaling
FRAME N° 2   tensor(499., device='cuda:0') tensor(811., device='cuda:0') tensor(638., device='cuda:0') tensor(1001., device='cuda:0')
FRAME N° 2   tensor(1220., device='cuda:0') tensor(890., device='cuda:0') tensor(1472., device='cuda:0') tensor(1078., device='cuda:0'
FRAME N° 2   tensor(371., device='cuda:0') tensor(685., device='cuda:0') tensor(494., device='cuda:0') tensor(815., device='cuda:0')
FRAME N° 2   tensor(1654., device='cuda:0') tensor(596., device='cuda:0') tensor(1802., device='cuda:0') tensor(656., device='cuda:0')
FRAME N° 2   tensor(726., device='cuda:0') tensor(612., device='cuda:0') tensor(796., device='cuda:0') tensor(697., device='cuda:0')
FRAME N° 2   tensor(826., device='cuda:0') tensor(534., device='cuda:0') tensor(890., device='cuda:0') tensor(602., device='cuda:0')
FRAME N° 2   tensor(976., device='cuda:0') tensor(613., device='cuda:0') tensor(1067., device='cuda:0') tensor(688., device='cuda:0')
FRAME N° 2   tensor(902., device='cuda:0') tensor(508., device='cuda:0') tensor(960., device='cuda:0') tensor(561., device='cuda:0')
FRAME N° 2   tensor(1816., device='cuda:0') tensor(554., device='cuda:0') tensor(1920., device='cuda:0') tensor(620., device='cuda:0')
FRAME N° 2   tensor(85., device='cuda:0') tensor(503., device='cuda:0') tensor(182., device='cuda:0') tensor(539., device='cuda:0')
FRAME N° 2   tensor(736., device='cuda:0') tensor(447., device='cuda:0') tensor(773., device='cuda:0') tensor(482., device='cuda:0')
FRAME N° 2   tensor(862., device='cuda:0') tensor(456., device='cuda:0') tensor(901., device='cuda:0') tensor(490., device='cuda:0')
```



```
!python3 object_tracking.py
FRAME N° 2   tensor(780., device='cuda:0') tensor(423., device='cuda:0') tensor(813., device='cuda:0') tensor(451., device='cuda:0')
FRAME N° 2   tensor(1097., device='cuda:0') tensor(422., device='cuda:0') tensor(1132., device='cuda:0') tensor(447., device='cuda:0'
FRAME N° 2   tensor(828., device='cuda:0') tensor(409., device='cuda:0') tensor(861., device='cuda:0') tensor(441., device='cuda:0')
FRAME N° 2   tensor(630., device='cuda:0') tensor(424., device='cuda:0') tensor(660., device='cuda:0') tensor(449., device='cuda:0')
FRAME N° 2   tensor(29., device='cuda:0') tensor(518., device='cuda:0') tensor(84., device='cuda:0') tensor(554., device='cuda:0')
FRAME N° 2   tensor(692., device='cuda:0') tensor(382., device='cuda:0') tensor(717., device='cuda:0') tensor(408., device='cuda:0')
FRAME N° 2   tensor(765., device='cuda:0') tensor(385., device='cuda:0') tensor(789., device='cuda:0') tensor(408., device='cuda:0')
Tracking objects
{0: (1956, 1429), 1: (618, 1092), 2: (1124, 960), 3: (1271, 835), 4: (1509, 957), 5: (1382, 788), 6: (176, 772), 7: (1122, 688), 8: (
CUR FRAME LEFT PTS
[(818, 1311), (1956, 1429), (618, 1092), (2555, 924), (1124, 960), (1271, 835), (1509, 957), (1382, 788), (2776, 864), (176, 772), (1
Inside Warmup
Inside Inference
torch.Size([1, 3, 384, 640])
Inside Process Detections
Inside Rescaling
FRAME N° 3   tensor(492., device='cuda:0') tensor(820., device='cuda:0') tensor(634., device='cuda:0') tensor(1020., device='cuda:0')
FRAME N° 3   tensor(1227., device='cuda:0') tensor(898., device='cuda:0') tensor(1479., device='cuda:0') tensor(1078., device='cuda:0
FRAME N° 3   tensor(366., device='cuda:0') tensor(690., device='cuda:0') tensor(492., device='cuda:0') tensor(820., device='cuda:0')
FRAME N° 3   tensor(1634., device='cuda:0') tensor(588., device='cuda:0') tensor(1774., device='cuda:0') tensor(646., device='cuda:0'
FRAME N° 3   tensor(826., device='cuda:0') tensor(536., device='cuda:0') tensor(892., device='cuda:0') tensor(604., device='cuda:0')
FRAME N° 3   tensor(728., device='cuda:0') tensor(616., device='cuda:0') tensor(798., device='cuda:0') tensor(703., device='cuda:0')
FRAME N° 3   tensor(1797., device='cuda:0') tensor(549., device='cuda:0') tensor(1920., device='cuda:0') tensor(615., device='cuda:0'
FRAME N° 3   tensor(980., device='cuda:0') tensor(615., device='cuda:0') tensor(1070., device='cuda:0') tensor(691., device='cuda:0')
FRAME N° 3   tensor(902., device='cuda:0') tensor(509., device='cuda:0') tensor(960., device='cuda:0') tensor(561., device='cuda:0')
```

\



```
Inside Rescaling
FRAME N° 26   tensor(728., device='cuda:0') tensor(830., device='cuda:0') tensor(848., device='cuda:0') tensor(1018., device='cuda:0')
FRAME N° 26   tensor(1047., device='cuda:0') tensor(704., device='cuda:0') tensor(1176., device='cuda:0') tensor(813., device='cuda:0')
FRAME N° 26   tensor(127., device='cuda:0') tensor(972., device='cuda:0') tensor(327., device='cuda:0') tensor(1078., device='cuda:0')
FRAME N° 26   tensor(868., device='cuda:0') tensor(632., device='cuda:0') tensor(963., device='cuda:0') tensor(740., device='cuda:0')
FRAME N° 26   tensor(933., device='cuda:0') tensor(543., device='cuda:0') tensor(999., device='cuda:0') tensor(604., device='cuda:0')
FRAME N° 26   tensor(880., device='cuda:0') tensor(480., device='cuda:0') tensor(925., device='cuda:0') tensor(516., device='cuda:0')
FRAME N° 26   tensor(1009., device='cuda:0') tensor(508., device='cuda:0') tensor(1064., device='cuda:0') tensor(554., device='cuda:0')
FRAME N° 26   tensor(1268., device='cuda:0') tensor(464., device='cuda:0') tensor(1328., device='cuda:0') tensor(496., device='cuda:0')
FRAME N° 26   tensor(737., device='cuda:0') tensor(480., device='cuda:0') tensor(782., device='cuda:0') tensor(523., device='cuda:0')
FRAME N° 26   tensor(88., device='cuda:0') tensor(506., device='cuda:0') tensor(183., device='cuda:0') tensor(540., device='cuda:0')
FRAME N° 26   tensor(561., device='cuda:0') tensor(498., device='cuda:0') tensor(614., device='cuda:0') tensor(543., device='cuda:0')
FRAME N° 26   tensor(662., device='cuda:0') tensor(468., device='cuda:0') tensor(702., device='cuda:0') tensor(511., device='cuda:0')
FRAME N° 26   tensor(1824., device='cuda:0') tensor(616., device='cuda:0') tensor(1920., device='cuda:0') tensor(678., device='cuda:0')
FRAME N° 26   tensor(1426., device='cuda:0') tensor(455., device='cuda:0') tensor(1519., device='cuda:0') tensor(514., device='cuda:0')
FRAME N° 26   tensor(792., device='cuda:0') tensor(446., device='cuda:0') tensor(830., device='cuda:0') tensor(478., device='cuda:0')
FRAME N° 26   tensor(33., device='cuda:0') tensor(518., device='cuda:0') tensor(97., device='cuda:0') tensor(554., device='cuda:0')
FRAME N° 26   tensor(844., device='cuda:0') tensor(426., device='cuda:0') tensor(880., device='cuda:0') tensor(460., device='cuda:0')
FRAME N° 26   tensor(1426., device='cuda:0') tensor(454., device='cuda:0') tensor(1519., device='cuda:0') tensor(514., device='cuda:0')
FRAME N° 26   tensor(610., device='cuda:0') tensor(448., device='cuda:0') tensor(648., device='cuda:0') tensor(480., device='cuda:0')
FRAME N° 26   tensor(689., device='cuda:0') tensor(395., device='cuda:0') tensor(716., device='cuda:0') tensor(422., device='cuda:0')
FRAME N° 26   tensor(1824., device='cuda:0') tensor(615., device='cuda:0') tensor(1920., device='cuda:0') tensor(679., device='cuda:0')
FRAME N° 26   tensor(648., device='cuda:0') tensor(398., device='cuda:0') tensor(676., device='cuda:0') tensor(418., device='cuda:0')
FRAME N° 26   tensor(733., device='cuda:0') tensor(422., device='cuda:0') tensor(764., device='cuda:0') tensor(444., device='cuda:0')
Tracking objects
{3: (1349, 1002), 4: (1635, 1110), 5: (1432, 845), 6: (179, 776), 7: (1128, 741), 8: (1342, 738), 10: (1013, 723), 11: (1541, 785), 12: (868, 769), 13: (1207,
CUR FRAME LEFT PTS
[(1152, 1339), (2784, 955), (2784, 954)]
```

✓ 1m 16s completed at 11:51 PM

Fig: 4.1 Model Results

CHAPTER 5

5.1 Summary and Conclusions

Object tracking approach for traffic analysis is a powerful tool that utilizes deep learning algorithms to track vehicles and analyse traffic patterns. YOLOv7, a state-of-the-art object detection framework, is widely used in this approach due to its high accuracy and real-time performance. YOLOv7 uses a single neural network to detect and track objects in real-time, making it an ideal choice for traffic analysis. This approach can be applied to a variety of traffic-related tasks, such as vehicle counting, speed estimation, and trajectory prediction. By leveraging YOLOv7's object tracking capabilities, traffic analysts can gain valuable insights into traffic flow and congestion, which can help inform infrastructure planning and traffic management decisions.

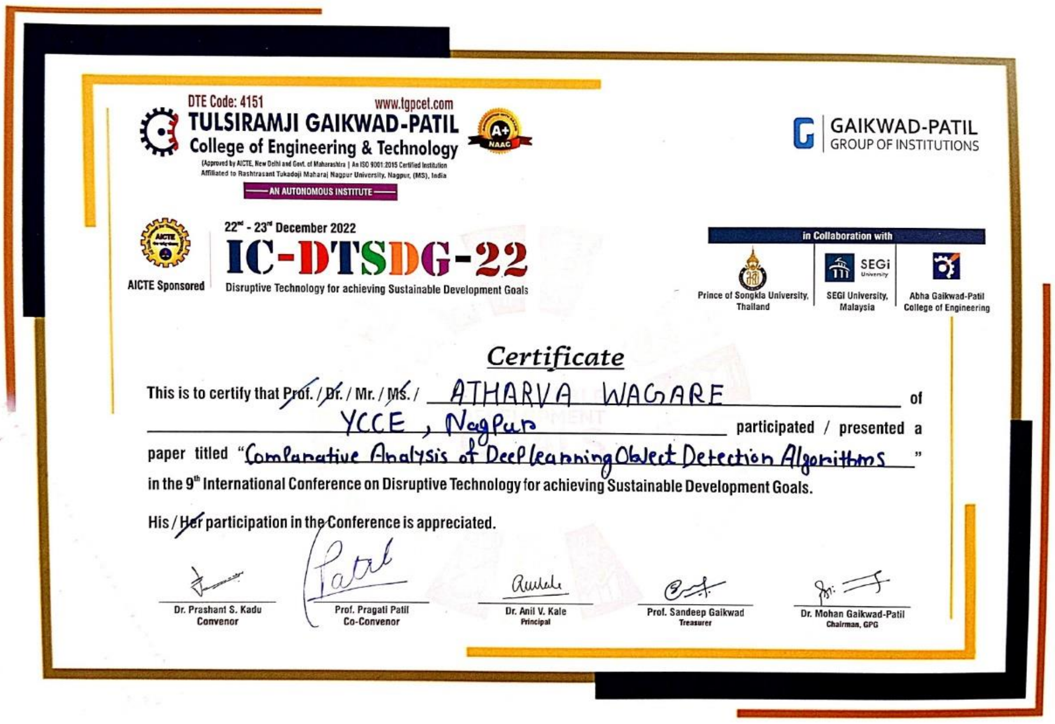
5.2 Social Utility

A social utility for the project "Object Detection and Tracking Approach for Traffic Analysis" could be to improve road safety by providing real-time alerts to drivers about potential hazards on the road. The system could work by using YOLOv7 to detect and track objects such as cars, pedestrians, and bicycles in real-time. As the system detects potential hazards, it could provide alerts to drivers through an app on their smartphone or a heads-up display in their car.

For example, if a car is detected merging into the driver's lane, the system could provide an alert to the driver to maintain a safe following distance. This social utility could greatly improve road safety by providing drivers with additional information about their surroundings and potential hazards on the road. It could also help to reduce the number of accidents and injuries caused by distracted driving or other factors.

CHAPTER 6

6.1 Appendix



DTE Code: 4151 www.tgpcet.com

TULSIRAMJI GAIKWAD-PATIL College of Engineering & Technology

(Approved by AICTE, New Delhi and Govt. of Maharashtra | An ISO 9001:2015 Certified Institution Affiliated to Rashtrasant Tukadoji Maharaj Nagpur University, Nagpur, (MS), India

AN AUTONOMOUS INSTITUTE

A+ NAAC

GAIKWAD-PATIL GROUP OF INSTITUTIONS

AICTE Sponsored

22nd - 23rd December 2022

IC-DTSDG-22

Disruptive Technology for achieving Sustainable Development Goals

in Collaboration with

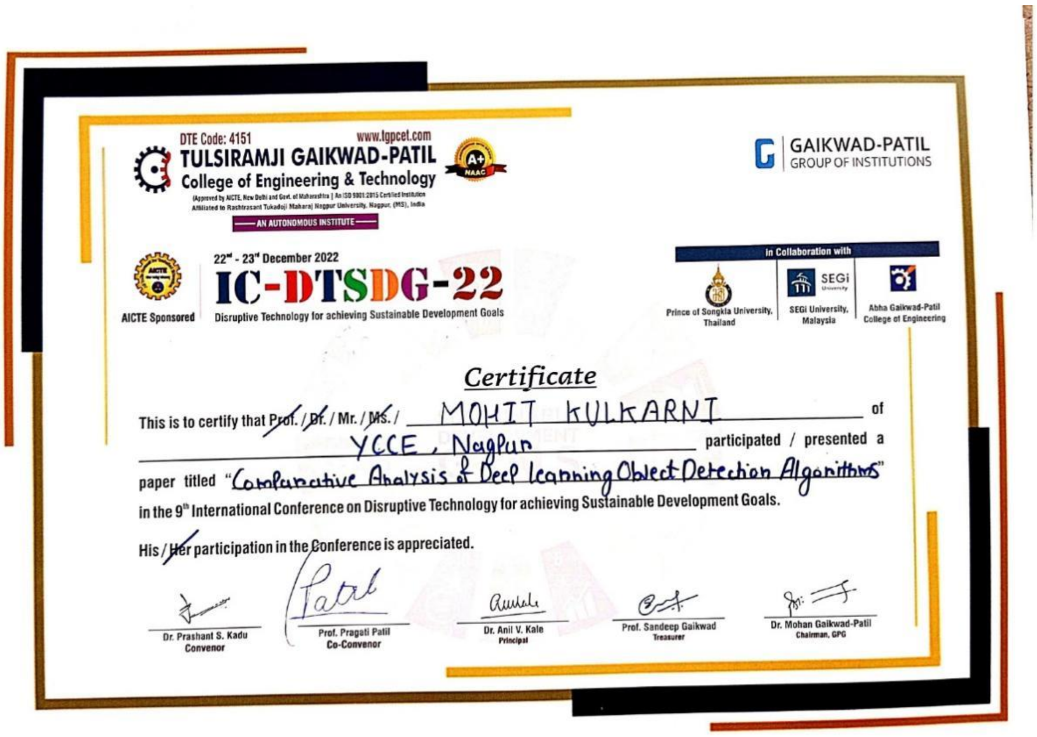
Prince of Songkla University, Thailand | SEGi University, Malaysia | Abha Gaikwad-Patil College of Engineering

Certificate

This is to certify that ~~Prof.~~ / ~~Dr.~~ / Mr. / ~~Ms.~~ / ATHARVA WAGARE of YCCE, Nagpur participated / presented a paper titled "Comparative Analysis of Deep learning Object Detection Algorithms" in the 9th International Conference on Disruptive Technology for achieving Sustainable Development Goals.

His / ~~Her~~ participation in the Conference is appreciated.

Dr. Prashant S. Kadu, Convenor | Prof. Pragati Patil, Co-Convenor | Dr. Anil V. Kale, Principal | Prof. Sandeep Gaikwad, Treasurer | Dr. Mohan Gaikwad-Patil, Chairman, GPG



DTE Code: 4151 www.tgpcet.com

TULSIRAMJI GAIKWAD-PATIL College of Engineering & Technology

(Approved by AICTE, New Delhi and Govt. of Maharashtra | An ISO 9001:2015 Certified Institution Affiliated to Rashtrasant Tukadoji Maharaj Nagpur University, Nagpur, (MS), India

AN AUTONOMOUS INSTITUTE

A+ NAAC

GAIKWAD-PATIL GROUP OF INSTITUTIONS

AICTE Sponsored

22nd - 23rd December 2022

IC-DTSDG-22

Disruptive Technology for achieving Sustainable Development Goals

in Collaboration with

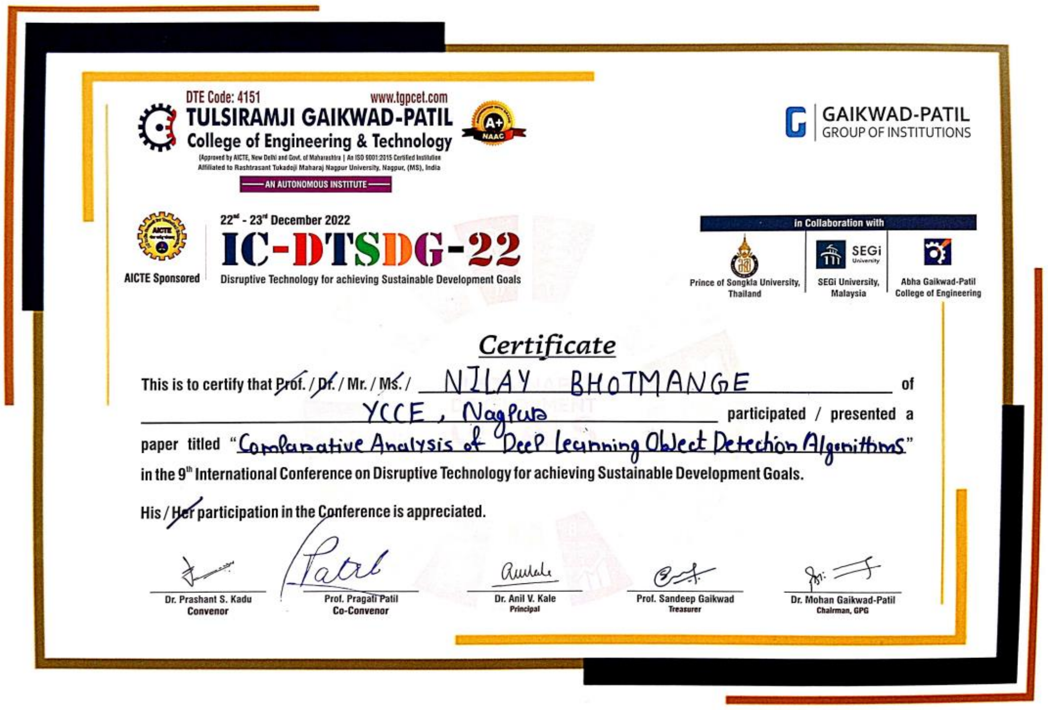
Prince of Songkla University, Thailand | SEGi University, Malaysia | Abha Gaikwad-Patil College of Engineering

Certificate

This is to certify that ~~Prof.~~ / ~~Dr.~~ / Mr. / ~~Ms.~~ / MOHIT KULKARNI of YCCE, Nagpur participated / presented a paper titled "Comparative Analysis of Deep learning Object Detection Algorithms" in the 9th International Conference on Disruptive Technology for achieving Sustainable Development Goals.

His / ~~Her~~ participation in the Conference is appreciated.

Dr. Prashant S. Kadu, Convenor | Prof. Pragati Patil, Co-Convenor | Dr. Anil V. Kale, Principal | Prof. Sandeep Gaikwad, Treasurer | Dr. Mohan Gaikwad-Patil, Chairman, GPG



DTE Code: 4151 www.tgpcet.com

TULSIRAMJI GAIKWAD-PATIL College of Engineering & Technology

(Approved by AICTE, New Delhi and Govt. of Maharashtra | An ISO 9001:2015 Certified Institution Affiliated to Rashtrasant Tukadoji Maharaj Nagpur University, Nagpur, (MS), India

— AN AUTONOMOUS INSTITUTE —

A+ NAAC

GAIKWAD-PATIL GROUP OF INSTITUTIONS

AICTE Sponsored

22nd - 23rd December 2022

IC-DTSDG-22

Disruptive Technology for achieving Sustainable Development Goals

in Collaboration with

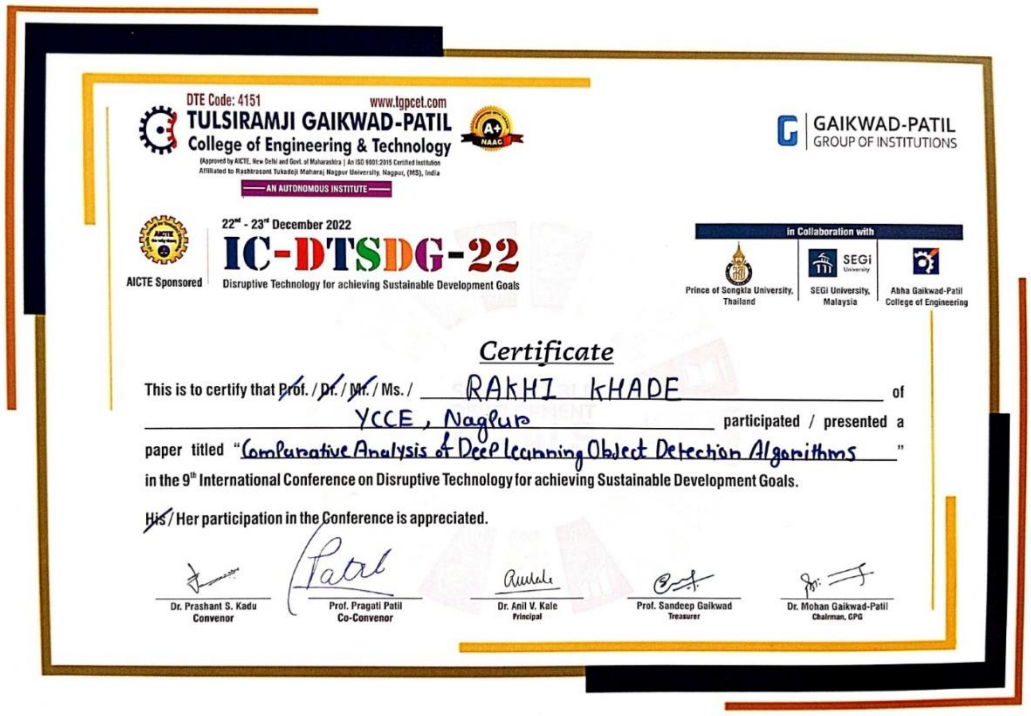
Prince of Songkla University, Thailand | SEGi University, Malaysia | Abha Gaikwad-Patil College of Engineering

Certificate

This is to certify that ~~Prof.~~ / ~~Dr.~~ / Mr. / ~~Ms.~~ / NILAY BHOTMANGE of YCCE, Nagpur participated / presented a paper titled "Comparative Analysis of Deep learning Object Detection Algorithms" in the 9th International Conference on Disruptive Technology for achieving Sustainable Development Goals.

His / ~~Her~~ participation in the Conference is appreciated.

Dr. Prashant S. Kadu, Convenor | Prof. Pragati Patil, Co-Convenor | Dr. Anil V. Kale, Principal | Prof. Sandeep Gaikwad, Treasurer | Dr. Mohan Gaikwad-Patil, Chairman, GPG



DTE Code: 4151 www.tgpcet.com

TULSIRAMJI GAIKWAD-PATIL College of Engineering & Technology

(Approved by AICTE, New Delhi and Govt. of Maharashtra | An ISO 9001:2015 Certified Institution Affiliated to Rashtrasant Tukadoji Maharaj Nagpur University, Nagpur, (MS), India

— AN AUTONOMOUS INSTITUTE —

A+ NAAC

GAIKWAD-PATIL GROUP OF INSTITUTIONS

AICTE Sponsored

22nd - 23rd December 2022

IC-DTSDG-22

Disruptive Technology for achieving Sustainable Development Goals

in Collaboration with

Prince of Songkla University, Thailand | SEGi University, Malaysia | Abha Gaikwad-Patil College of Engineering

Certificate

This is to certify that ~~Prof.~~ / ~~Dr.~~ / ~~Mr.~~ / Ms. / RAKHI KHADE of YCCE, Nagpur participated / presented a paper titled "Comparative Analysis of Deep learning Object Detection Algorithms" in the 9th International Conference on Disruptive Technology for achieving Sustainable Development Goals.

~~His~~ / Her participation in the Conference is appreciated.

Dr. Prashant S. Kadu, Convenor | Prof. Pragati Patil, Co-Convenor | Dr. Anil V. Kale, Principal | Prof. Sandeep Gaikwad, Treasurer | Dr. Mohan Gaikwad-Patil, Chairman, GPG

CHAPTER 7

7.1 References

[1] Kartik Umesh Sharma Department of PG Studies Computer Science and Engineering, K. U. Sharma, Department of PG Studies Computer Science and Engineering, Nileshsingh V. Thakur Department of PG Studies Computer Science and Engineering, N. V. Thakur, and O. M. V. A. Metrics, "A review and an approach for object detection in images," *International Journal of Computational Vision and Robotics*, 01-Jan-2017.

[2] Anand John and Divyakant Meva, "A Comparative Study of Various Object Detection Algorithms and Performance Analysis" *International Journal of Computer Sciences and Engineering,* Vol. 8, Issue.10, October 2020.

[3] Nikhil Yadav and Utkarsh Binay, "Comparative Study of Object Detection Algorithms" *International Research Journal of Engineering and Technology,* Vol. 4, Issue.11, Nov -2017.

[4] Prince Kumar, Vaibhav Garg , Pavan Somvanshi , Pathanjali C, "A Comparative Study of Object Detection Algorithms in A Scene" *International Journal of Engineering Research & Technology (IJERT),* Vol. 8 Issue 05, May-2019

[5] Tran, M.-T., Dinh-Duy, T., Truong, T.-D., Ton-That, V., Do, T.-N., Luong, Q.-A., … Do, M. N. (2018), "Traffic Flow Analysis with Multiple Adaptive Vehicle Detectors and Velocity Estimation with Landmark-Based Scanlines" *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW).*

[6] 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.

[7] Reddy, K. R., Priya, K. H., & Neelima, N. (2015). "Object Detection and Tracking -- A Survey*". 2015 International Conference on Computational Intelligence and Communication Networks (CICN).* doi:10.1109/cicn.2015.317

[8] Chen, Z., Khemmar, R., Decoux, B., Atahouet, A., & Ertaud, J.-Y. (2019). *Real Time Object Detection, Tracking, and Distance and Motion Estimation based on Deep Learning: Application to Smart Mobility. 2019 Eighth International Conference on Emerging Security Technologies (EST).* doi:10.1109/est.2019.8806222

[9] Soleimanitaleb, Z., Keyvanrad, M. A., & Jafari, A. (2019). Object Tracking Methods:A Review. 2019 9th International Conference on Computer and Knowledge Engineering (ICCKE). doi:10.1109/iccke48569.2019.8964761.

[10] Abdul Rajjak, S. S., & Kureshi, A. K. (2019). Recent Advances in Object Detection and Tracking for High Resolution Video: Overview and State-of-the-Art. 2019 5th International Conference On Computing, Communication, Control And Automation (ICCUBEA). doi:10.1109/iccubea47591.2019.9128812

[11] Jonnalagadda, M., Taduri, S., & Reddy, R. (2020). RealTime Traffic Management System Using Object Detection based Signal Logic. 2020 IEEE Applied Imagery Pattern Recognition Workshop (AIPR). doi:10.1109/aipr50011.2020.9425070

[12] Zhao, Z.-Q., Zheng, P., Xu, S.-T., & Wu, X. (2019). Object Detection With Deep Learning: A Review. IEEE Transactions on Neural Networks and Learning Systems, 1–21. doi:10.1109/tnnls.2018.2876865

[13] Malhotra, P., & Garg, E. (2020). Object Detection Techniques: A Comparison. 2020 7th International Conference on Smart Structures and Systems (ICSSS). doi:10.1109/icsss49621.2020.9202254

[14] Srivastava, S., Divekar, A. V., Anilkumar, C., Naik, I., Kulkarni, V., & Pattabiraman, V. (2021). Comparative analysis of deep learning image detection algorithms. Journal of Big Data, 8(1). doi:10.1186/s40537-021-00434-w

[15] Kartik Umesh Sharma* & Nileshsingh V. Thakur. (2017). A review and an approach for object detection in images. International Journal of Computational Vision and Robotics 7(1/2):196 DOI:10.1504/IJCVR.2017.10001813