COGNITIVE REAL-TIME TRAFFIC ANALYSIS

Project Report submitted by

Hema Upadhyaya
(4NM19CS078)

Lahari R Rao
(4NM19CS101)

Medha S
(4NM19CS107)

U Sneha Shet
(4NM19CS207)

Under the guidance of

Dr. Sannidhan M S

Associate Professor

Department of Computer Science and Engineering

in partial fulfillment of the requirements for the award of the Degree of

Bachelor of Engineering
(Computer Science & Engineering)

from

Visvesvaraya Technological University

Department of Computer Science and Engineering

NMAM Institute of Technology, Nitte – 574110

(An Autonomous Institution affiliated to VTU Belgaum)

MAY 2023

NITTE EDUCATION TRUST | N.M.A.M. INSTITUTE OF TECHNOLOGY
(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi)
Nitte – 574 110, Karnataka, India
(ISO 9001:2015 Certified), Accredited with 'A' Grade by NAAC
☎: 08258 - 281039 - 281263, Fax: 08258 - 281265
Department of Computer Science and Engineering
B.E. CSE Program Accredited by NBA, New Delhi from 1-7-2018 to 30-6-2021

CERTIFICATE

Certified that the project work entitled

"Cognitive Real-Time Traffic Analysis "

is a bonafide work carried out by

Hema Upadhyaya (4NM18CS078)
Lahari R Rao (4NM19CS101)
Medha S (4NM19CS107)
U Sneha Shet (4NM19CS207)

in partial fulfillment of the requirements for the award of
Bachelor of Engineering Degree in Computer Science and Engineering
prescribed by the Visvesvaraya Technological University, Belagavi
during the year 2022-23

It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library.
The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the Bachelor of Engineering Degree.

Signature of Guide

Signature of HOD

ACKNOWLEDGEMENT

The satisfaction that accompanies the completion of any task would be incomplete without the mention of all the people, without whom this endeavor would have been a difficult one to achieve. Their constant blessings, encouragement, guidance and suggestions have been a constant source of inspiration.

First and foremost, our gratitude to our project guide, **Dr. Sannidhan M S** for his constant guidance and support throughout the course of this project and for his valuable suggestions.

We also take this opportunity to express a deep sense of gratitude to the project coordinators, for their valuable guidance and support.

We acknowledge the support and valuable inputs given by **Dr. Jyothi Shetty,** the Head of the Department, Computer Science and Engineering, NMAMIT, Nitte.

Our sincere thanks to our beloved principal, **Dr. Niranjan N Chiplunkar** for permitting us to carry out this project at our college and providing us with all needed facilities.

We also thank all those who have supported us throughout the entire duration of our project.

Finally, thanks to all the staff members of the Department of Computer Science and Engineering and our friends for their honest opinions and suggestions throughout the course of our project.

Hema Upadhyaya (4NM19CS078)
Lahari R Rao (4NM19CS101)
Medha S (4NM19CS107)
U Sneha Shet (4NM19CS207)

ABSTRACT

Traditional traffic management approaches are insufficient in addressing urban traffic congestion and improving transportation efficiency. This necessitates a smart framework for traffic management that can address congestion and improve transportation efficiency. By leveraging intelligent systems and real-time traffic data, it enables accurate traffic predictions, optimization of traffic flow, and overall improvement in transportation efficiency.

Existing traffic management approaches suffer from reliance on human judgment, scalability challenges, and limited quantitative studies. To overcome these drawbacks, the proposed model emphasizes the importance of data collection, extraction, and analysis for traffic forecasting. Deep learning algorithms and big data analytics play a crucial role in forecasting traffic. By analyzing large volumes of traffic data and incorporating deep learning algorithms, proactive measures can be taken to mitigate congestion and optimize traffic flow.

The suggested model integrates data extraction from traffic videos, image annotation using LabelImg, object detection using YOLOv6, and future traffic prediction using LSTM. This integration helps to minimize traffic congestion which in turn helps road users to have smoother commutes and improved travel experiences.

TABLE OF CONTENTS

CONTENTS PAGE NO.

TITLE PAGE..... I

CERTIFICATE..... II

ACKNOWLEDGEMENT..... III

ABSTRACT..... IV

TABLE OF CONTENTS..... V-VI

LIST OF FIGURES..... VII

CHAPTER 1 INTRODUCTION **1-4**

1.1 Overview 1-2

1.2 Problem Statement 2-3

1.3 Study Area 3

1.4 Objectives 3

1.5 Motivation 3-4

1.6 Organization of Chapters 4

CHAPTER 2 LITERATURE SURVEY **5-10**

2.1 Existing Systems 5-10

CHAPTER 3 SYSTEM REQUIREMENTS SPECIFICATIONS **11-12**

CHAPTER 4 SYSTEM DESIGN **13-14**

CHAPTER 5 SOFTWARE APPROACH **15-21**

5.1 Python 15

5.2 Machine Learning 15-16

5.3 Deep Learning 16

5.4 Neural Networks 16-19

5.4.1 YOLOv6 17-18

5.4.2 Long Short-Term Memory (LSTM) 18-19
5.5 Tensorflow 19
5.6 Keras 20
5.7 Sklearn 20

CHAPTER 6 IMPLEMENTATION 21-32
6.1 Dataset 21
6.2 Libraries Used 23-26
6.2.1 OpenCV (cv2) 23
6.2.2 NumPy 23-24
6.2.3 Pandas 24
6.2.4 Random 24
6.2.5 Tqdm 24
6.2.6 Os 25
6.2.7 Csv 25
6.2.8 Glob 25
6.2.9 DefaultDict 25-26
6.2.10 Matplotlib 26
6.2.11 Train_test_split 26
6.2.12 Timeseriesgenerator 26
6.3 Object Detection Using YOLOv6 26-30
6.3.1 Training 27-28
6.3.2 Testing 28-29
6.3.3 Inference/Detection 29-30
6.4 Counting 30
6.5 Future Prediction Using LSTM 30-32
6.5.1 Training 31
6.5.2 Testing and Prediction 32

CHAPTER 7 RESULTS 33-35

CHAPTER 8 CONCLUSION 36

CHAPTER 9 REFERENCES 37-38

LIST OF FIGURES

FIGURE CAPTION	PAGE NO.
Fig 1 LabelImg Tool	12
Fig 2 System Design	13
Fig 3 Deep Neural Network	17
Fig 4 YOLOv6 Model Architecture	17
Fig 5 LSTM Unit	19
Fig 6 Heavy-Traffic Image Frame	21
Fig 7 Image Frame with Bounding Boxes	22
Fig 8 Annotation Text file of an Image	23
Fig 9 Training the dataset	27
Fig 10 Evaluation Results I	28
Fig 11 Evaluation Results II	29
Fig 12 Confusion Matrix	29
Fig 13 Inference Result I	30
Fig 14 Inference Result II	30
Fig 15 Count of each class of vehicles	30
Fig 16 LSTM Training Model	31
Fig 17 LSTM Training Process	31
Fig 18 Measured Regression Errors	32
Fig 19 y_pred vs y_test Truth Graph	32
Fig 20 Input Image	33
Fig 21 Output Image with Bounding Boxes	33
Fig 22 Annotation Text File	34
Fig 23 Count of vehicles	34
Fig 24 Actual and Predicted Count of Vehicles of each class label	35

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

With the increasing complexity and volume of urban traffic, traditional traffic management approaches have become inadequate in effectively addressing congestion and improving transportation efficiency. The impact of traffic congestion goes beyond just transportation, affecting individuals on multiple levels. It leads to time loss, mental stress, and contributes to pollution and global warming. This poses the need for a smart framework, which could identify heavy traffic zones in cross lines and intersections to provide an energy efficient transportation system. Cognitive real-time traffic analysis is a field of study that focuses on utilizing advanced technologies, such as artificial intelligence and machine learning, to analyze and manage traffic in real-time. The primary goal of cognitive real-time traffic analysis is to enhance the efficiency and safety of urban transportation networks. Traffic analysis has gained increasing importance in the present day as a means of monitoring and managing traffic.

Cognitive real-time traffic analysis aims to overcome these challenges by leveraging intelligent systems that can perceive, understand, and make informed decisions based on real-time traffic data. To ensure economic growth and the comfort of road users, smooth traffic flow is essential. Effective traffic congestion management is crucial in meeting these requirements. With advancements in the transportation sector, authorities are increasingly focusing on monitoring traffic congestion. By collecting and analyzing traffic information, authorities can predict congestion, enabling them to allocate resources and plan ahead for smoother journeys. The precise and timely collection of this data is essential for generating accurate traffic predictions. Real-time data can be utilized to optimize traffic flow, reduce congestion, and improve overall transportation efficiency.

The existing model suffers from certain problems, one of which is the reliance on human judgment in manual traffic control. This dependence can lead to inconsistencies and inefficiencies in the system. Scalability and feasibility are challenges due to the need for trained personnel at each intersection. Limited quantitative studies hinder evaluating its effectiveness compared to other traffic control methods. The HOG-based model for

object detection has drawbacks including complex background difficulties, high computational cost, reliance on specific datasets, and performance variation across use cases. The statement emphasizes the need for specific insights and discussions on limitations and evaluation of pre-training models with Faster R-CNN, highlighting the importance of addressing challenges and identifying areas for improvement. The statement highlights the effectiveness of RNN-based models, including GRNN-LF on a GPU, for long-term traffic flow prediction. It suggests addressing the limitations and challenges of these models in this context. Ultimately, the choice of model depends on factors like data characteristics, computational resources, and specific requirements of the prediction task.

In light of the aforementioned drawbacks, we suggest a model in which data collection, data extraction, and data analysis are crucial for the forecast of traffic. In order to forecast and manage traffic, deep learning algorithms and big data analytics are essential. We extracted image frames from a traffic video and performed image annotation manually, using LabelImg. The images along with their annotation files formed the dataset. Object detection was done using YOLOv6. You only look once (YOLO) is a detection algorithm for images. It is the most famous and fastest algorithm in the field of computer vision. Using LSTM we predict the future traffic count at an intersection, based on previous count of vehicles at different timestamps. Long short-term memory LSTM is an artificial neural network used in the fields of artificial intelligence and deep learning. By analyzing large volumes of traffic data and incorporating deep learning algorithms, it becomes possible to forecast traffic conditions in advance, enabling proactive measures to mitigate congestion. This integration empowers intelligent traffic management systems to optimize traffic flow, make informed decisions, and enhance transportation efficiency. By minimizing traffic congestion, road users can experience smoother commutes and improved travel experiences.

1.2 PROBLEM STATEMENT

The complexity and volume of urban traffic have made traditional traffic management approaches inadequate, causing congestion and affecting individuals on multiple levels. Cognitive real-time traffic analysis leverages advanced technologies to manage

traffic in real-time, but existing models have limitations, such as reliance on human judgment and limited quantitative studies. We propose a model that integrates YOLOv6 for object detection and LSTM for predicting traffic count at intersections, to optimize traffic flow and enhance transportation efficiency, resulting in improved travel experiences for road users.

1.3 STUDY AREA

Cognitive real-time traffic analysis focuses on studying traffic patterns, flow, and congestion in urban environments, transportation networks, and intelligent transportation systems. The study aims to develop a system that can detect and track vehicles in real-time using Yolov6 and predict traffic flow using LSTM. The system will be tested on a dataset captured from a busy intersection. The results of the study can potentially improve traffic management and reduce congestion in urban areas.

1.4 OBJECTIVES

- To develop a predictive model for traffic forecasting and management using deep learning algorithms and big data analytics.
- To extract image frames from traffic videos and perform manual image annotation using LabelImg to form the dataset.
- To implement object detection using the YOLOv6 algorithm to identify and track vehicles in the dataset.
- To use LSTM, an artificial neural network, to predict future traffic counts at an intersection based on previous counts at different timestamps.

1.5 MOTIVATION

The motivation for cognitive real-time traffic analysis arises from the need to address the challenges of traffic management in urban areas. Traditional methods fall short in providing accurate and timely information for effective traffic control. Cognitive real-time traffic analysis utilizes advanced technologies like AI and machine learning to process large volumes of real-time traffic data. By leveraging YOLOv6, a fast and accurate object detection algorithm, and LSTM, a recurrent neural network for sequential data analysis, it becomes possible to achieve precise object detection and accurate

prediction of traffic conditions. This technology has the potential to improve traffic management, enhance road safety, and optimize transportation systems. This enables the development of predictive models and optimization strategies to improve traffic flow, reduce congestion, and enhance transportation systems. The benefits include improved traffic management, reduced travel times, increased safety, and more efficient urban transportation.

1.6 ORGANIZATION OF THE CHAPTERS

The project report has been organized into eight chapters, which are as follows:

CHAPTER I : Introduces the main idea of the project. It gives a brief knowledge about the objectives, methodology and motivation for the same.

CHAPTER II : It includes a literature survey of related works.

CHAPTER III : Discusses the system requirements and specifications that are needed for the project.

CHAPTER IV : Includes the system design details.

CHAPTER V : Explains in detail about the software approaches and training model used in developing the project.

CHAPTER VI : Includes the implementation details of the project such as the dataset details, training, testing and prediction details.

CHAPTER VII : Discusses the results of the project.

CHAPTER VIII : Outlines the conclusion and the future work of the project.

CHAPTER 2
LITERATURE SURVEY

2.1 EXISTING SYSTEMS

Considering few research gaps in existing systems, we have concentrated on investigating the areas and technologies that can fill up the existing gaps to improve a better performing system. Hence our literature review focused in exploring the technologies and research areas related to traffic analysis, object detection and future prediction.

Traffic Analysis:

The study by B M Williams et al. highlights the importance of short-term traffic condition forecasting in intelligent transportation systems (ITS). It emphasizes the need for accurate predictions to optimize transportation operations and discusses the shift from physical capacity construction to operational efficiency. The survey proposes the seasonal autoregressive moving average (SARMA) process as a suitable model for univariate short-term traffic forecasts at fixed network locations. It acknowledges the limitations of longer time intervals and underscores the fundamental role of accurate predictions for uncongested entry points and averaged traffic data. The approach is suitable to deal with regular variations, but the performance is undesirable when the traffic data show significant stochastic and nonlinear characteristics. [1]

Lippi et al. explored developments in short-term traffic flow forecasting. They devised two new support vector regression models by combining the SARIMA model with a Kalman filter. This Hybrid approach delivered plausible results in various practices. But when enlarging the training data size, the performance quickly reached its bottleneck, failing to take advantage of larger datasets. [2]

Using real-time traffic analysis based on UAV recordings and deep learning algorithms, Zhang et al.'s 2019 study focused on reducing traffic congestion in urban areas. It utilizes a position-fixed UAV to capture road traffic videos, and applies state-of-the-art deep learning methods to identify moving objects in the footage. The system calculates relevant mobility metrics to analyse traffic congestion and measure its consequences.

The proposed approach is validated through manual analysis results and visualization. However, some disadvantages include limited coverage due to UAV flight limitations, potential data quality and reliability issues, computational complexity for real-time analysis, challenges in obtaining diverse training data. [3]

Jyoti Tiwari et al 2020 aimed to redesign the traffic signal system that is static switching to signal switching, which can perform real-time signal monitoring and handling. The switching time of signal will be decided based on real time image detection with good accuracy in dense traffic. This practice proved its most effectiveness in releasing the congested traffic at an efficient and faster rate. The challenges that need to be addressed include real-time processing of large amount of image data, handling occlusions and complex traffic scenarios, and ensuring system scalability and cost-effectiveness. [4]

Shuja Ali et al 2022 proposed a method for detecting vehicles and also tracking them through the use of cascade classifier and centroid tracking. They also incorporated georeferencing and coregistration of acquired images and then proceeded on to extract lanes. After segmenting out the region of interest, they proceeded with the detection and tracking tasks. The disadvantage in this system is the difficulty of accurately detecting and tracking objects in aerial images due to factors such as high object density, challenging view angles, varying altitudes, and illumination changes. [5]

Object Detection:

Dalal and Triggs investigated feature sets for reliable optical object recognition, focusing on linear SVM-based human detection. For this assignment, they discovered that grids of Histograms of Oriented Gradient (HOG) descriptors worked well. They looked at the effects of different computation phases and pinpointed crucial success criteria, such as fine-scale gradients, accurate orientation binning, fairly coarse spatial binning, and excellent local contrast normalisation. However, potential limitations include sensitivity to appearance variations, computational complexity, limitations in detecting non-visible or non-upright poses, and the need for further research on the generalizability of the system to other shape-based object classes. [6]

Wenze Li's research involved an in-depth examination of Faster R-CNN models, assessing their performance alongside R-CNN and Fast R-CNN. The experimental

results showed the accuracy and detection speed of R-CNN, fast R-CNN and faster R-CNN based on three different data sets. By considering accuracy and detection speed, the study aimed to provide a comprehensive and unbiased evaluation of these object detection models. However, faster R-CNN exhibits slower detection speed compared to alternative object detection systems like YOLO. The other disadvantages include higher complexity and higher resource requirements. [7]

The classification accuracy of Single Shot Multibox Detector (SSD) algorithm was improved by Kumar, Zhang, and Lyu while maintaining its speed. Advancements were made through the use of multilayer convolutional neural networks, namely depth-wise separable convolution and spatial separable convolution. The convolutional layers of the SSD algorithm received these improvements. The suggested method successfully identified objects in multiple images and sorted them into predefined classes. The multilayer convolutional neural network and large number of default boxes were used in the revised SSD algorithm to obtain high object detection accuracy. One potential drawback of the proposed approach is that it does not explicitly consider the trade-off between accuracy and computational complexity. While achieving high accuracy, the algorithm may require significant computational resources and processing time, which could limit its feasibility for real-time applications on resource-constrained devices. [8]

Jifeng Dai et al in the year 2016, introduced a region-based, fully convolutional network approach for accurate and efficient object detection. R-FCN is a fully convolutional system that shares computations across the entire image, achieving competitive results on datasets like PASCAL VOC while being significantly faster than previous methods. The method utilizes position-sensitive score maps and Residual Networks (ResNets) as backbones for improved performance. While RFCN offers accurate detection, it is slower and less precise than YOLO, especially for small objects. Training RFCN models is also more complex, and it struggles with overlapping objects. [9]

In 2020, Asmara et al. aimed to estimate vehicle traffic density using the Double Exponential Smoothing forecasting model, which utilizes historical data to predict future quantities. The accuracy of the model was assessed using the Mean Absolute Percentage Error (MAPE) to quantify the error rate. Additionally, the study employed the You Only Look Once (YOLO) method, a Convolutional Neural Network (CNN) object detection technique, to identify vehicles as the primary contributors to traffic congestion. The utilization of YOLO in this approach brings accurate vehicle

identification, real-time processing, seamless integration, and high forecasting accuracy, contributing to effective traffic management and congestion alleviation efforts. [10]

Chuyi Li et al 2022 focused on advancing the capabilities of the YOLO series, which has long been considered the industry standard for efficient object detection. The authors present YOLOv6, an enhanced version that incorporates advancements in various components of object detectors. This includes exploring network design, training strategies, testing techniques, quantization, and optimization methods. YOLOv6 achieves high accuracy and fast throughput, outperforming other real-time detectors. [11]

Future Prediction:

In 2020, Belhadi et al. explored the application of recurrent neural networks for long-term traffic flow prediction in urban areas. They introduced a representation of long-term flows combined with weather and contextual data. A recurrent neural network method, called RNN-LF, was proposed to predict long-term flows using multiple data sources. Additionally, they developed a parallel implementation of the solution on a GPU (GRNN-LF) to enhance the performance of RNN-LF. [12]

Yijing Zhang 2020 proposed a vehicle management system which plays a crucial role in smart transportation, allowing traffic management departments to effectively control urban road traffic. Short-term traffic flow forecasting is an essential component of car guidance systems and intelligent transport. It provides drivers with optimal routes and serves as a fundamental mathematical foundation for managing traffic flow and improving traffic management schemes. This paper focuses on incorporating various mainstream approaches, such as ARIMA, RNN, and Sparse AutoEncoder (SAE), to address short-term traffic flow prediction. By exploring these approaches, the article aims to provide quick insights for those interested in this field. [13]

A team consisting of Yuelei Xiao and Yang Yin designed an innovative approach to enhance traffic prediction by developing a hybrid LSTM neural network that builds upon the LSTM model. To achieve optimal performance, they conducted experiments to fine-tune the structure and parameters of the hybrid LSTM network across various traffic conditions. Comparative analysis was carried out against alternative models.

The findings demonstrated that the hybrid LSTM model surpassed the other models in terms of prediction accuracy, while incurring only a marginal increase in execution time compared to the LSTM model.[14]

Tanzina Afrin, Nita Yodo proposed a Long Short-Term Memory (LSTM)-based framework called LSTM-CTP for predicting correlated traffic data. This type of data consists of time series recorded simultaneously in different regions of the same transportation network route. The proposed framework captures temporal and spatial trends and correlations in the data. LSTM models are used to predict temporal and spatial trends, and the Kalman-filter approach is employed to aggregate the predictions. The performance of LSTM-CTP is compared to other prediction algorithms, showing significant improvements. The framework aims to enhance traffic congestion control and establish a robust traffic management system. [15]

Zilin Huang et al 2020 proposed a framework for short-term traffic speed prediction. It includes a data preprocessing module to handle missing data and a prediction module that utilizes an attention-based LSTM (ATT-LSTM) model. The framework was tested on a 30-day traffic speed dataset from an urban road network, demonstrating superior performance compared to other deep learning algorithms like RNN and CNN. The attention mechanism reduces LSTM model error by up to 12.4% and improves prediction accuracy. [16]

Yassine Touzani and Khadija Douzi proposed a trading strategy for the Moroccan stock market using LSTM and GRU deep learning models to predict short and medium-term closing prices. The strategy outperforms benchmark indices, achieving an annualized return of 27.13% compared to lower returns of the market indices. The approach involves implementing decision rules for buying and selling stocks based on the model's forecasts and conducting simulations to optimize parameters and select the best-performing stocks. The proposed strategy demonstrates promising results and accounts for brokerage fees for realistic performance evaluation. [17]

Traditional traffic management approaches have proven insufficient in handling the complexity and volume of urban traffic, leading to congestion and negative impacts on individuals. To address these challenges, cognitive real-time traffic analysis has been proposed, leveraging advanced technologies. However, existing models have limitations, including the need for human judgment and limited quantitative studies. To

improve transportation efficiency and optimize traffic flow, we propose an innovative model that integrates YOLOv6 for object detection and LSTM for predicting traffic count at intersections. By utilizing deep learning techniques, our model can predict traffic conditions accurately in real-time, resulting in improved travel experiences for road users. This approach can be crucial for developing smarter transportation systems and mitigating traffic-related issues in urban areas.

CHAPTER 3

SYSTEM REQUIREMENTS SPECIFICATION

- **Central Processing Unit**

  Intel Core i5 6th Generation processor or higher. An AMD equivalent processor will also be optimal.

- **Operating System**

  Ubuntu or Microsoft Windows 10 is recommended.

- **RAM**

  4GB (minimum) or higher is recommended.

- **Google Colab**

  Colaboratory, commonly known as Colab, is a web-based tool developed by Google Research. It offers users the convenience of writing and executing Python code seamlessly. This platform is specifically designed to cater to tasks such as machine learning, data analysis, and educational purposes. Colab serves as a hosted Jupyter notebook service, eliminating the need for complex setups. Moreover, it provides free access to computational resources, including GPUs. By removing the requirement of downloading, installing, or managing software, Colab simplifies the process of creating and collaborating on Jupyter notebooks, enabling easy sharing of projects with others.

  Google Colab is highly regarded among deep learning enthusiasts, providing a platform for testing machine learning models, gaining experience, and developing intuition in areas such as hyperparameter tuning, data preprocessing, model complexity, and overfitting. It is compatible with popular browsers like Chrome, Firefox, and Safari, with thorough testing conducted on the latest versions. Colab is focused on supporting Python and its ecosystem of third-party tools, allowing users to import image datasets, train image classifiers, and evaluate models with just a few lines of code. By executing code on Google's cloud servers, Colab enables users to leverage powerful hardware resources like GPUs and TPUs, irrespective of their local machine's capabilities.

- **GPU**

  GPUs have become crucial for personal and business computing, offering parallel processing capabilities in various applications like graphics rendering and AI. Particularly in AI and machine learning, GPUs provide significant acceleration due to their immense computational power, making them ideal for image recognition and deep learning. Google Colab, our preferred environment for training models, utilizes the NVIDIA Tesla K80 GPU. This GPU significantly reduces data center costs, delivers exceptional performance, and increases throughput by 5-10x compared to CPU-only systems. It enables more scientific discoveries and enhanced user engagement for web services while saving resources.

- **LabelImg**

  LabelImg is a popular annotation tool for labeling images in object detection tasks like YOLO. With an intuitive interface, users can draw bounding boxes around objects and assign labels. It simplifies and enhances the efficiency of creating annotated datasets for YOLO models. LabelImg is a valuable tool for computer vision projects, aiding researches, developers, and data scientists in training and deploying YOLO models.

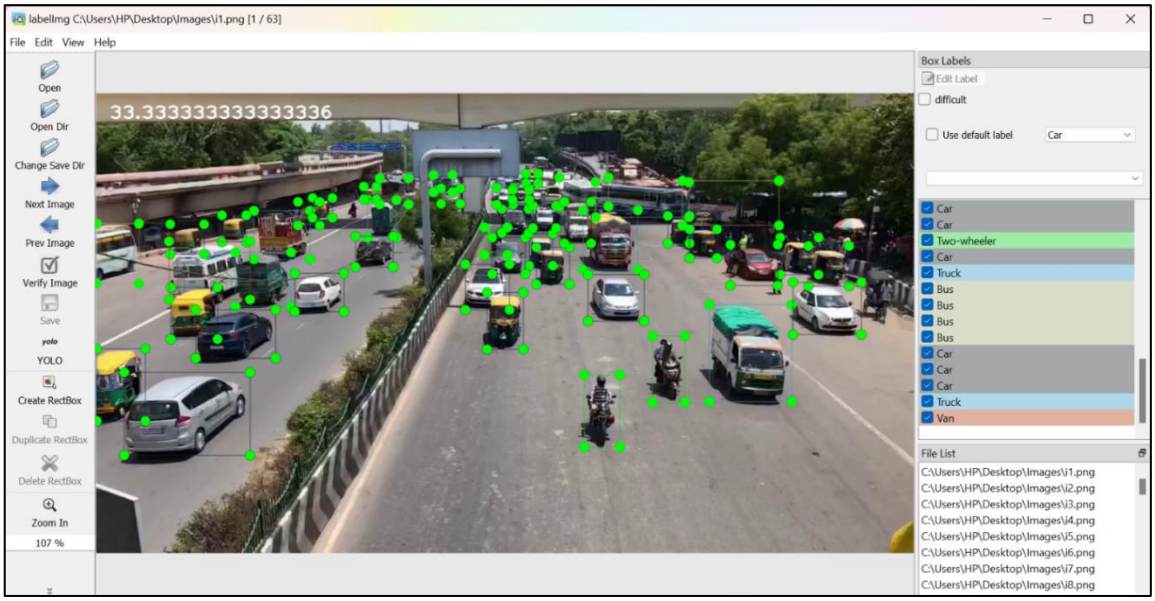


Fig.1 LabelImg Tool

CHAPTER 4

SYSTEM DESIGN

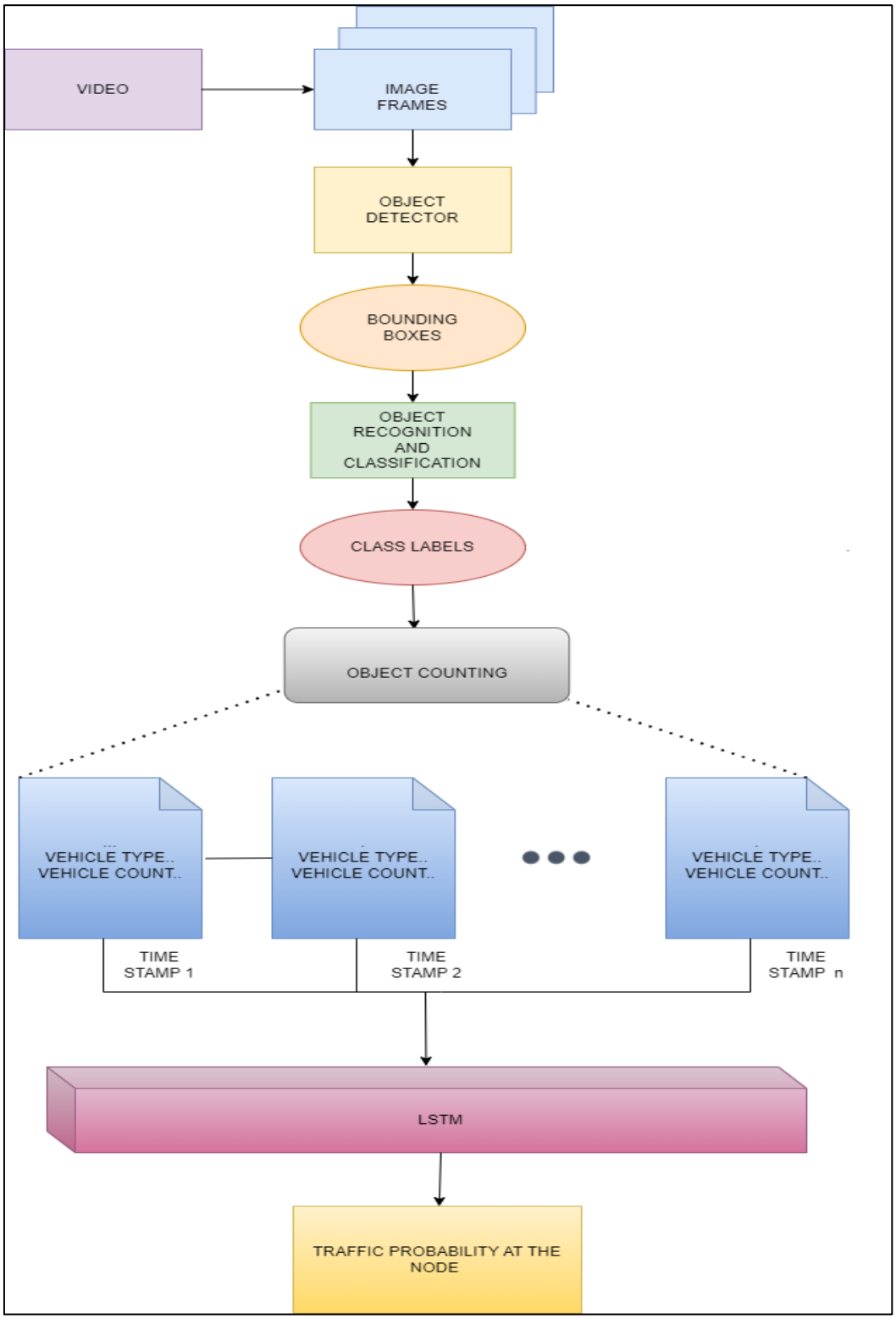


Fig. 2 System Design

The process of defining a system's architecture, product design, modules, interfaces, and data in order to meet predetermined requirements is known as system design. System design could be seen as the application of system theory to product development. This gives better understanding of how you solve a particular design problem, how you respond when there is more than expected traffic on your system, how you design the database of your system and many more. All these decisions are required to be taken carefully keeping in mind Scalability, Reliability, Availability, and Maintainability.

Fig. 2 is the system design for the framework we developed. The input to this model is a video, which was converted into image frames. Annotations were performed on the image frames by drawing the bounding boxes and corresponding text files were obtained. Annotations along with the text files was used as the dataset and was given to an object detector. The object detector performed object detection on each image frame and produced output images, with bounding boxes around the detected objects. The object detector also recognized and classified the objects into different class labels. We obtained the count of vehicles for each of these class labels at different time stamps. This count of vehicles was then given to LSTM model to predict the future count of vehicles, which gives the traffic probability at the node.

CHAPTER 5
SOFTWARE APPROACH

5.1 Python

Python is an open-source, interpreted, object-oriented programming language praised for its readability and clarity. It is frequently used for scripting, linking components, and developing applications quickly. Python is adaptable and ideal for a variety of uses because to its dynamic typing and high-level data structures. It promotes modularity and code reuse through modules and packages. The comprehensive standard library and the language are freely distributable and are supported on all major platforms. With libraries like SciPy and NumPy, which provide easy syntax, tools for linear algebra, and kernel methods, Python excels in machine learning.

The benefits of Python include its readability and ease of learning, which make it simpler to build and maintain code. For prototyping and experimentation, it has numerous applications and is favoured by scientists, engineers, and mathematicians. Python might be less effective for game and mobile app development, though. Due to design restrictions, it may experience runtime faults, necessitating careful testing. Python also uses a lot of memory and isn't frequently utilised in web browsers because of security issues.

5.2 Machine learning

A key component of data science is machine learning, which uses statistical techniques to train algorithms for categorization and prediction tasks. Data mining reveals important insights that help with decision-making and promote corporate success. Big data's exponential expansion is driving up need for data scientists, who are essential to identifying pertinent business problems and developing data-driven solutions.

Machine learning is a field that enables computers to make accurate predictions by leveraging data. Its applications are widespread and diverse, encompassing tasks such as recognizing objects in images, detecting pedestrians in autonomous vehicles, differentiating word meanings in sentences, filtering out spam emails, and generating spoken captions for movies. What sets machine learning apart from traditional software

is its ability to learn from vast datasets instead of relying solely on explicit code instructions, allowing for dynamic and adaptable solutions to problems.

5.3 Deep learning

"Deep learning" is a branch of machine learning and artificial intelligence that mimics human learning. It has a significant impact on data science and includes both statistics and predictive models. Data scientists can benefit from deep learning in a number of ways since it simplifies and accelerates the collection, analysis, and interpretation of large datasets.

Deep learning essentially automates predictive analytics. Deep learning uses a layered framework of increasing complexity and abstraction as opposed to classical machine learning, which uses linear methods. To understand, picture a little child learning the word "dog." The youngster eventually learns to recognise the distinctive characteristics of a dog by pointing to objects and getting feedback from a parent. As a result, a hierarchy of abstractions is produced, with each layer drawing on the knowledge of the one before it. Deep learning also employs a hierarchical structure.

5.4 Neural networks

In the domains of AI, machine learning, and deep learning, neural networks enable computer programmes to identify patterns and resolve common issues by mimicking the behaviour of the human brain. In general, a neural network is a piece of software designed to mimic the functions of the human brain, particularly pattern recognition and the transmission of input across several layers of artificial neural connections.

Deep learning techniques based on neural networks, sometimes referred to as artificial neural networks (ANNs) or simulated neural networks (SNNs) are a subset of machine learning. Their structure and nomenclature are modelled after the human brain, mirroring the communication between organic neurons.

Training data is essential for neural networks to develop and enhance their accuracy over time. However, these learning algorithms become effective tools in computer science and artificial intelligence once they are adjusted for accuracy, enabling us to quickly classify and cluster data. When compared to manual identification by human

experts, tasks in speech recognition or picture recognition can be completed in minutes as opposed to hours. Google's search algorithm is one of the most well-known neural networks.

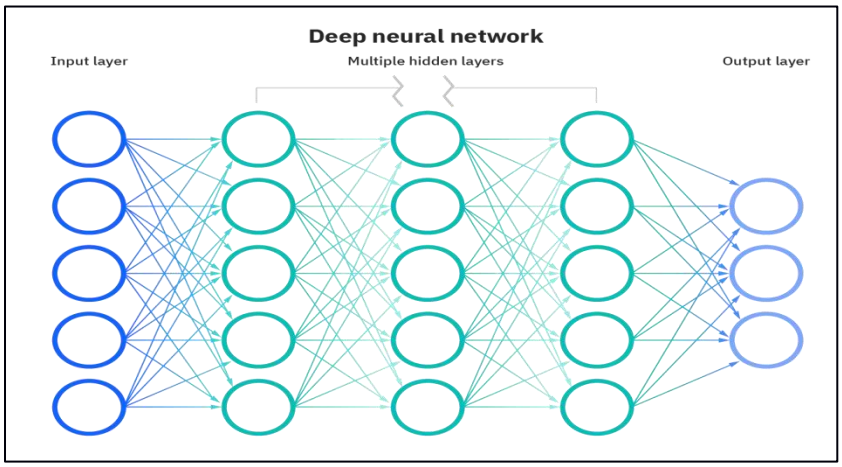


Fig. 3 Deep Neural Network

5.4.1 YOLOv6

YOLO (You Only Look Once) is a highly efficient object detection algorithm used in computer vision, known for its speed and accuracy. YOLOv6 is an advanced version that further enhances detection accuracy and inference speed. It adopts an anchor-free method, resulting in a 51% increase in speed compared to previous versions.

The aim of YOLOv6 is to achieve real-time object detection with high accuracy. It uses deep convolutional neural networks to extract features and make predictions on object classes and bounding box coordinates. YOLOv6 architecture incorporates various improvements, including advanced network architectures, feature fusion techniques, and attention mechanisms, improving accuracy and handling of small or complex objects.

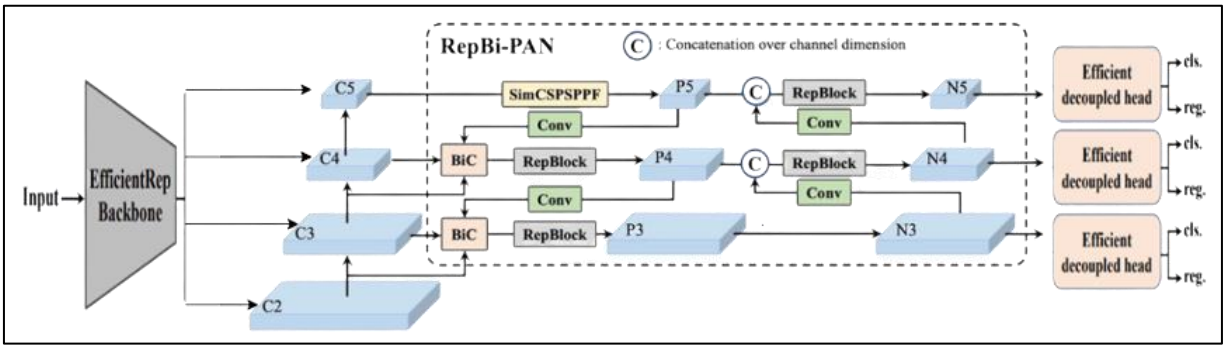


Fig.4 YOLOv6 Model Architecture

The YOLOv6 Backbone Architecture:

The backbone in YOLOv6 has a critical function of extracting features, which are then passed to the network's neck and head. It generates a map representation of the input and can leverage hardware computing power, such as GPUs. The EfficientRep is the comprehensive backbone architecture employed in YOLOv6.

The YOLOv6 Neck Architecture:

In YOLOv6, the neck component integrates multi-scale feature maps using PAN (Path Aggregation Networks). It employs a Bi-directional Concatenation (BiC) module to augment localization signals without excessive computation. The PAN in YOLOv6 performs feature concatenation from various reparametrized blocks. The neck used here is the RepBiF-PAN Neck (Reparametrized Bidirectional Feature PAN Neck).

The YOLOv6 Detection Head:

In contrast to YOLOv4 and YOLOv5, YOLOv6 introduces the Efficient Decoupled Head, which handles the computation of the output. The head is decoupled, meaning there exists a distinct layer between the network and the final head. This architectural modification leads to improved performance. The classification and detection branches in YOLOv6 operate independently without sharing parameters. They branch out separately from the backbone, resulting in reduced computations and enhanced accuracy.

5.4.2 Long Short-Term Memory (LSTM)

A popular type of artificial neural network used in AI and deep learning is called LSTM (Long Short-Term Memory). It is particularly adept at handling time series and other sequential data. Due to its distinctive memory cells, LSTM is well suited for tasks like sentiment analysis and speech recognition since it can remember information for extended periods of time. Recurrent connections and specialised gates found in LSTM, which differ from typical neural networks, handle long-term dependencies and overcome the vanishing gradient problem, effectively.

The information flow inside the network is controlled by the LSTM using parts like the input, output, and forget gates. The input gate adds fresh information to the current state, the forget gate deletes extraneous information from the previous state, and the

output gate only outputs pertinent information. The LSTM can identify patterns and generate precise predictions in sequential data because of its selective processing.

Overall, LSTM is an effective method for modelling and forecasting sequential data, finding use in computer vision, natural language processing, and other fields where sequential data is essential.

LSTM networks were specifically designed to overcome the vanishing gradient problem that commonly arises when training traditional recurrent neural networks.

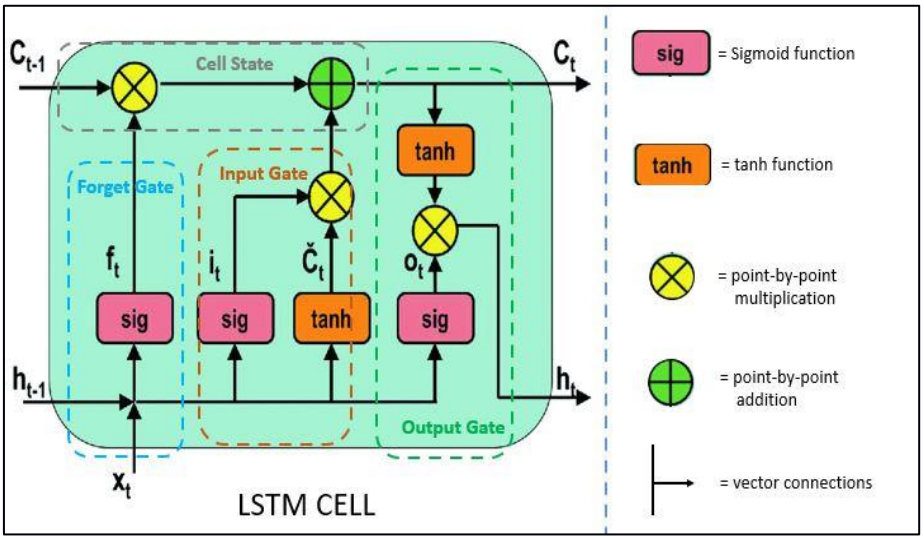


Fig.5 LSTM unit

5.5 Tensorflow

TensorFlow is a free and open-source library for Machine Learning and Artificial Intelligence. Originally designed for numerical computations, it became popular for deep learning and was open-sourced by Google. TensorFlow-gpu enables parallel computing on GPUs. It operates on multi-dimensional arrays called tensors and uses data flow graphs for execution, allowing distributed computing across clusters of computers with GPU support.

5.6 Keras

Keras is a high-level neural networks API in Python, widely used for deep learning. It offers a user-friendly interface for building and training models, allowing easy

definition of network architecture using a sequential or functional API. With pre-built layers and compatibility with popular backends like TensorFlow, Theano, and CNTK, Keras simplifies the process of creating complex models. It also provides utilities for data preprocessing, model evaluation, and visualization. Overall, Keras is a popular choice for deep learning due to its simplicity and extensive functionality.

5.7 Sklearn

Scikit-learn (sklearn) is a popular Python machine learning library built on NumPy, SciPy, and matplotlib. It offers a simple and consistent API, making it suitable for beginners and experts alike. Sklearn provides a wide range of algorithms for classification, regression, clustering, and more. It also includes preprocessing tools for data cleaning and transformation. With evaluation metrics and techniques for model selection, sklearn helps optimize performance. It integrates well with other Python libraries, making it a valuable resource for machine learning tasks in academia and industry.

CHAPTER 6

IMPLEMENTATION

6.1 DATASET

A dataset in machine learning is, quite simply, a collection of data pieces that can be treated by a computer as a single unit for analytic and prediction purposes. This means that the data collected should be made uniform and understandable for a machine that doesn't see data the same way as humans do. For this, after collecting the data, it's important to preprocess it by cleaning and completing it, as well as annotate the data by adding meaningful tags readable by a computer. Moreover, a good dataset should correspond to certain quality and quantity standards. You should make sure your dataset is pertinent and well-balanced for a quick and easy training process.

We considered a 5-minutes heavy-traffic video and converted it into image frames. These image frames were separated by equal time intervals between them. One such image frame is shown in **Fig. 6**.

Fig. 6 Heavy-traffic image frame

For each image frame, annotation was performed manually, using the LabelImg tool. Annotation refers to the process of labeling objects in an image with bounding boxes and corresponding class labels. The purpose of annotation is to provide ground truth information to train the YOLO model to accurately detect and classify objects in new images.

To annotate an image for YOLOV6, we manually drew bounding boxes around each object in the image and assigned a class label to each box, as shown in **Fig. 7**. The class labels represent the categories of objects that the YOLO model will be trained to detect, such as "car," "person," or "dog." The class labels that we have used include, “Car”, “Two-wheeler”, “Bus”, “Three-wheeler” and “Truck”.

Fig. 7 Image frame with bounding boxes

As a result of the annotation process, we obtain an annotation text file for each image frame, as shown in **Fig. 8**. Here, each line corresponds to bounding box of one object in the image with the following format: `<class_label> <x_center> <y_center> <width> <height>`

- `class_label`: An integer value indicating the class of the object detected in the image.
- `x_center`: The x-coordinate of the center of the object's bounding box, normalized to the range 0 to 1.
- `y_center`: The y-coordinate of the center of the object's bounding box, normalized to the range 0 to 1.
- `width`: The width of the object's bounding box, normalized to the range 0 to 1.
- `height`: The height of the object's bounding box, normalized to the range 0 to 1.

The images along with the corresponding annotation text files form the dataset of the project. There are around 250 images along with their annotation text files in the dataset.

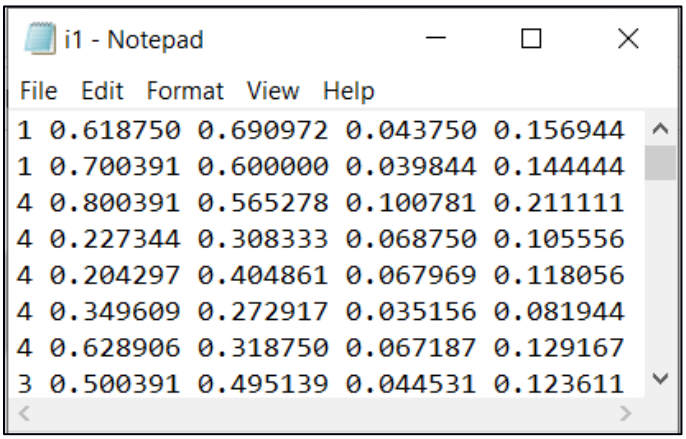


Fig. 8 Annotation text file of an image

6.2 LIBRARIES USED

A Python library is a collection of related modules containing pre-compiled codes, documentation, and configuration data that can be used repeatedly in different programs. This makes programming simpler and more convenient, as it eliminates the need to write the same code repeatedly. Python libraries are particularly important in fields such as Machine Learning, Data Science, and Data Visualization.

6.2.1 OpenCV (cv2)

OpenCV is an open-source computer vision and machine learning library with over 2500 optimized algorithms. It provides a common infrastructure for computer vision applications, accelerating the use of machine perception in commercial products. With support for multiple programming languages and operating systems, including C++, Python, Java, and MATLAB, OpenCV caters to a wide range of developers. It excels in real-time vision tasks and leverages specialized instruction sets like MMX and SSE for improved performance on compatible hardware platforms.

6.2.2 NumPy

NumPy is a popular Python library specifically designed for efficient array manipulation. It offers a wide range of functions for various mathematical operations such as linear algebra, Fourier transform, and matrix operations. Being an open-source project, NumPy is freely available for use. The name "NumPy" is derived from "Numerical Python." Although Python provides lists that can be used as arrays, they can be slow

for certain computations. NumPy addresses this limitation by introducing the ndarray object, which is significantly faster (up to 50 times) than traditional Python lists. The ndarray object in NumPy comes with a rich set of functions that simplify array handling and manipulation.

6.2.3 Pandas

Pandas is a well-known Python library utilized for data analysis and manipulation. It offers a range of data structures and functions that enhance the efficiency and ease of working with structured data, particularly tabular data. Pandas finds extensive application in domains like scientific research, finance, and data analysis where structured data manipulation is essential. Its intuitive and robust features make it a preferred choice within the Python ecosystem for tasks involving data analysis and manipulation.

6.2.4 Random

The random library in Python is a pre-installed module that offers functionalities for generating random numbers, selecting random elements, and shuffling sequences. It is commonly employed in various applications, including simulations, games, statistics, and cryptography. With the shuffle() function, it is possible to randomly rearrange a list directly. Additionally, the seed() function initializes the random number generator using a seed value, enabling the reproduction of the same sequence of random numbers. Nonetheless, it is essential to understand that the generated random numbers are pseudorandom and determined by an algorithm and initial seed value.

6.2.5 Tqdm

The tqdm library is a Python library that provides a fast, extensible progress bar for loops and other iterable processes. It allows you to easily add progress indicators to your code, giving you visibility into the execution progress and estimated time remaining. The tqdm library is widely used in various domains, including data analysis, machine learning, and scientific computing. Its simplicity and versatility make it a popular choice for adding progress bars to iterative processes, providing a visual representation of the execution progress and enhancing the user experience.

6.2.6 Os

The Python OS module is a part of the standard utility modules and offers a range of functions to interact with the operating system. It provides a platform-independent approach to access operating system-specific features. The module includes two important sub-modules, namely *os* and *os.path*, which offer numerous functions for file system operations. The *os.path* module is particularly useful when working with files located at different locations within the system. It offers convenient functions for tasks like merging, normalizing, and retrieving file path names in Python. With its extensive capabilities, the OS module facilitates seamless interaction with the underlying operating system and simplifies file system operations in Python programs.

6.2.7 Csv

The csv library in Python is a built-in module that provides functionality for working with CSV (Comma-Separated Values) files. CSV files are a common format for storing tabular data, where each row represents a record, and the values are separated by commas or other delimiters. The `csv` library makes it easy to read from and write to CSV files. The csv library is simple and efficient for basic CSV file handling in Python. It is well-suited for reading and writing CSV files with simple structures and doesn't offer more advanced data manipulation capabilities.

6.2.8 Glob

The Python glob library is a built-in module that provides file and directory searching functionality using wildcard patterns. It is useful for performing operations on multiple files that share a common pattern in their names or locations. The glob.glob() function returns a list of file or directory paths that match the specified pattern, making it an efficient solution for tasks such as batch processing, file manipulation, or collecting file lists.

6.2.9 Defaultdict

The defaultdict is not a separate library in Python but rather a class within the collections module, which is part of the Python standard library. The defaultdict class provides an alternative to the built-in dict class and offers a default value for keys that are not present in the dictionary. defaultdict can be particularly useful when dealing with data structures, counting frequencies, or grouping data. It helps avoid unnecessary conditional checks and enhances code readability.

6.2.10 Matplotlib

Matplotlib is a widely used Python plotting library for creating high-quality visualizations and figures. It is highly customizable and flexible, with extensive documentation and examples available. The matplotlib.pyplot module is a simplified interface for creating and customizing plots, often imported under the alias plt for convenience. It is commonly used for data visualization, exploratory data analysis, and creating publication-quality figures.

6.2.11 Train_test_split

train_test_split is a function provided by the sklearn.model_selection module in the scikit-learn library. It is commonly used for splitting a dataset into training and testing subsets to evaluate the performance of machine learning models. The train_test_split function is a convenient tool for creating training and testing sets from a given dataset. It helps in assessing the generalization capability of a machine learning model by evaluating its performance on unseen data.

6.2.12 TimeseriesGenerator

TimeseriesGenerator is a class provided by the tensorflow.keras.preprocessing. sequence module in TensorFlow. It is used to generate time series sequences for training recurrent neural networks (RNNs) or other sequential models. The TimeseriesGenerator class is particularly useful when working with time series data and training models that require sequential input. It helps in generating batches of input-output pairs from the time series data, making it easier to train models that learn patterns over time.

6.3 OBJECT DETECTION USING YOLOv6

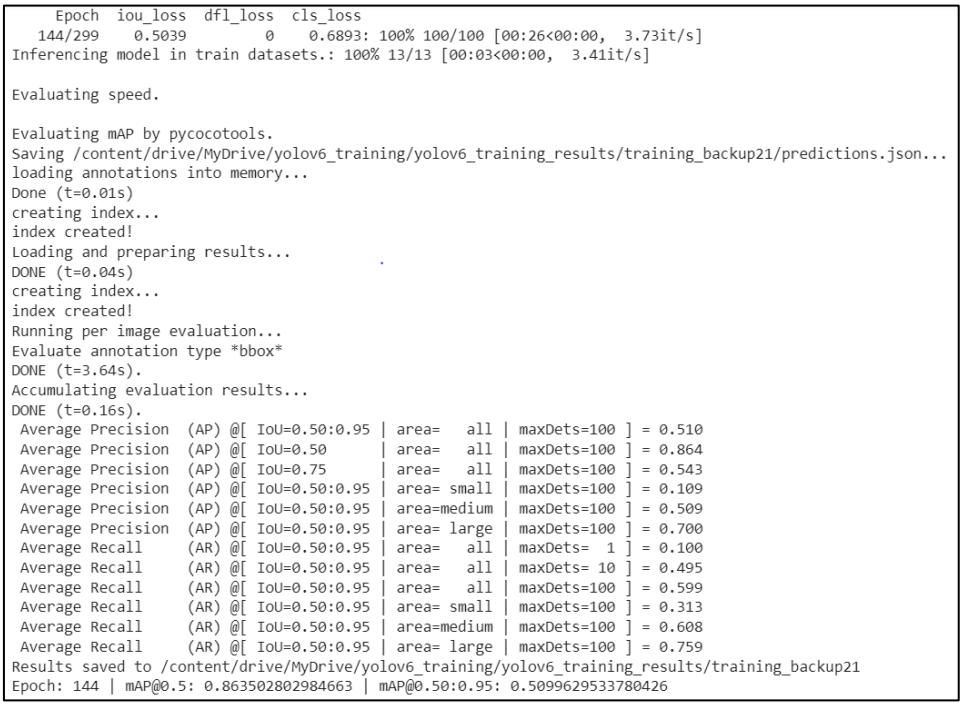
Object detection is a computer vision task where an algorithm locates and identifies objects in an image or video. It involves feature extraction, object proposals, classification, and bounding box regression. Object detection is used in many applications like self-driving cars, surveillance systems, and robotics, and is done using frameworks like YOLO, Faster R-CNN, and SSD.

6.3.1 Training

In machine learning, training a model means finding good values for weights and bias from labeled examples using empirical risk minimization. The process involves using a training dataset to run input data through an algorithm and modify the model through an iterative process called model fitting, with the accuracy of the dataset being crucial to the model's precision.

Supervised learning is possible when input and output values are available, while unsupervised learning involves determining patterns in the data without reference output data. Additional data is then used to fit patterns or clusters in an iterative process that improves accuracy based on correlation to expected patterns or clusters.

In our project, the model used for training is YOLOV6 and we have used 80% of the total dataset for training. The dataset is trained for 300 epochs with a batch size of 2.



```
     Epoch  iou_loss  dfl_loss  cls_loss
   144/299    0.5039         0    0.6893: 100% 100/100 [00:26<00:00,  3.73it/s]
Inferencing model in train datasets.: 100% 13/13 [00:03<00:00,  3.41it/s]

Evaluating speed.

Evaluating mAP by pycocotools.
Saving /content/drive/MyDrive/yolov6_training/yolov6_training_results/training_backup21/predictions.json...
loading annotations into memory...
Done (t=0.01s)
creating index...
index created!
Loading and preparing results...
DONE (t=0.04s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=3.64s).
Accumulating evaluation results...
DONE (t=0.16s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.510
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.864
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.543
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.109
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.509
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.700
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.100
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.495
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.599
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.313
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.608
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.759
Results saved to /content/drive/MyDrive/yolov6_training/yolov6_training_results/training_backup21
Epoch: 144 | mAP@0.5: 0.863502802984663 | mAP@0.50:0.95: 0.5099629533780426
```

Fig. 9 Training the dataset

We have used yolov6s.pt which is a pre-trained model file for the YOLOv6s object detection algorithm. It contains the learned weights of the neural network that can be used as a starting point for training a custom object detection model. By using a pre-trained model such as yolov6s.pt, developers can leverage the knowledge that has already been gained by the model and fine-tune it on a smaller, custom dataset to achieve higher accuracy on a specific task.

The YOLOv6s model architecture consists of an EfficientRep backbone, a RepBiFPANNeck neck, and an EffiDeHead head. EfficientRep is a feature extractor using EfficientNet architecture. RepBiFPANNeck is a feature fusion network combining multi-scale features via repeated BiFPN structure. EffiDeHead is the detection head predicting object classes and bounding boxes using convolutional layers and output layer. The optimizer used for training the model is Stochastic Gradient Descent (SGD).

6.3.2 Testing

Testing is referred to as the process where the performance of a fully trained model is evaluated on a testing set. The purpose of testing is to compare the outputs from the neural network against targets in an independent set (the testing instances). Testing ensures that the software meets requirements and works as expected. It also helps identify defects and flaws during development, preventing issues from arising post-release. In our project, 20% of the dataset is used for testing.

Fig. 10 shows a report on object detection performance on a validation dataset, using a trained model. The best mF1 threshold achieved was around 0.508. The evaluation metrics include Precision, Recall, F1 score, and mAP at different IOU thresholds for one class labeled "all", representing all objects in the dataset. The model achieved high precision and recall, with an F1 score of 0.844 and a mean average precision of 0.881.

```
Inferencing model in val datasets.: 100% 2/2 [00:03<00:00,  1.99s/it]
IOU 50 best mF1 thershold near 0.508.
Class                 Images      Labels     P@.5iou     R@.5iou    F1@.5iou      mAP@.5  mAP@.5:.95
all                       50        2496       0.896       0.801       0.844       0.881       0.549
```

Fig. 10 Evaluation results I

Mean average precision (mAP) is evaluated using pycocotools. The evaluation is done to measure the performance of an object detection model on a test set. The mAP is calculated at different Intersection over Union (IoU) thresholds and different object sizes, as shown in **Fig. 11**.

```
Accumulating evaluation results...
DONE (t=0.14s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.548
 Average Precision  (AP) @[ IoU=0.50       | area=   all | maxDets=100 ] = 0.876
 Average Precision  (AP) @[ IoU=0.75       | area=   all | maxDets=100 ] = 0.611
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.149
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.554
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.733
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.117
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.529
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.631
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.335
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.647
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.786
Results saved to /content/drive/MyDrive/yolov6_training/evaluation_results/results
```

Fig. 11 Evaluation results II

The confusion matrix is plotted and is as shown in **Fig. 12**.

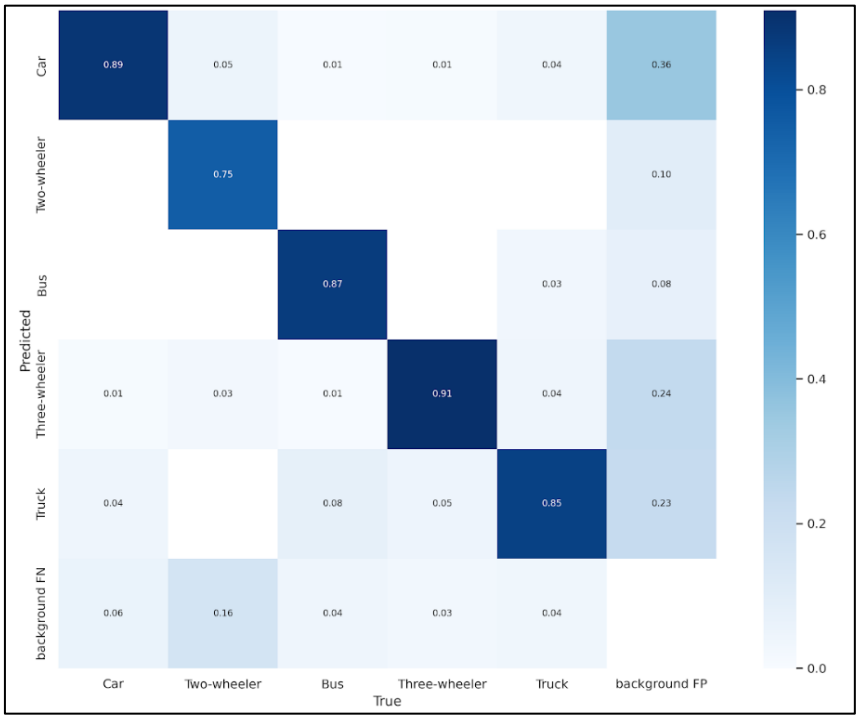


Fig. 12 Confusion matrix

6.3.3 Inference/Detection

Inference or detection of new images in YOLO (You Only Look Once) is the process of using a trained YOLO model to detect objects in new, unseen images. This involves passing the new images through the YOLO model and using the model's learned weights and architecture to predict the location, class, and confidence score of objects within the image.

In our project, 201 new images were used for inference or detection.

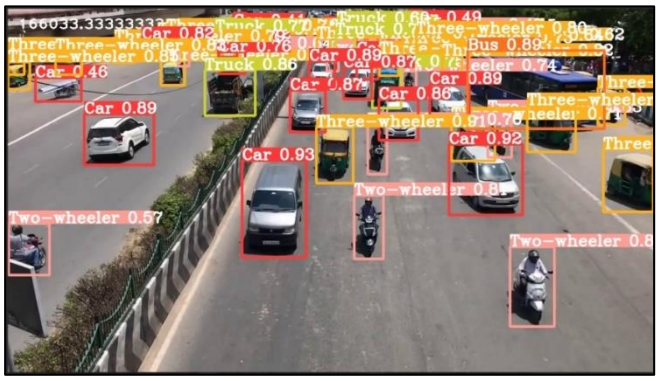


Fig. 13 Inference result I

```
0 0.359375 0.0263889 0.0203125 0.0305556 0.413628
0 0.482422 0.0569444 0.0242187 0.05 0.429316
3 0.596094 0.0444444 0.021875 0.0388889 0.445517
4 0.482422 0.0583333 0.0242187 0.05 0.448698
4 0.255078 0.176389 0.0367187 0.0666667 0.450467
3 0.240625 0.0916667 0.0171875 0.0416667 0.452085
0 0.0777344 0.221528 0.0710938 0.0652778 0.45926
1 0.601953 0.0854167 0.0226563 0.0541667 0.470996
3 0.0125 0.146528 0.025 0.0680556 0.478804
0 0.557422 0.127083 0.0335938 0.0597222 0.490227
0 0.630078 0.0159722 0.0179687 0.0263889 0.494041
4 0.565625 0.0673611 0.034375 0.0736111 0.51829
4 0.541406 0.0479167 0.0234375 0.0541667 0.552851
1 0.0324219 0.661111 0.0632813 0.141667 0.569802
```

Fig. 14 Inference result II

6.4 COUNTING

We have used a python code to count the vehicles belonging to each class label using the text file obtained as output from the above inference. Later we stored this count of each class label in a CSV file, as shown in **Fig. 15**.

Image	Car	Two-wheeler	Bus	Three-wheeler	Truck
i250	18	9	1	21	9
i251	17	11	1	21	10
i252	18	10	1	20	9
i253	13	5	1	20	10
i254	15	5	1	19	8
i255	13	9	1	20	10
i256	12	8	1	16	10
i257	14	6	1	19	12
i258	14	6	1	17	12
i259	19	6	1	18	10
i260	23	5	2	15	10
i261	22	7	1	18	12

Fig. 15 Count of each class of vehicles

6.5 FUTURE PREDICTION USING LSTM RECURRENT NEURAL NETWORK

Future prediction is the act of using information from the past and present to make assumptions and estimations about what will happen in the future. It involves analyzing current trends, patterns, and data to forecast possible outcomes, events, and changes that may occur in the future.

Future prediction using LSTM is a type of machine learning technique that is used to make predictions based on time series data. LSTM is a type of recurrent neural network (RNN) that is specifically designed to handle long-term dependencies and time-series data.

6.5.1 Training

Training model involves a Keras sequential model with 3 LSTM layers and a dense layer. The first LSTM layer has 8 units, takes input of shape (2,1), and returns sequences. The second LSTM layer also has 8 units and returns sequences. A LeakyReLU activation function is applied after each of these two layers. The third LSTM layer has 4 units and returns a single output sequence. Dropout regularization is applied after the second and third LSTM layers. Finally, there is a dense layer with a single output unit.

```
Model: "sequential_48"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_144 (LSTM)             (None, 2, 8)              320

 leaky_re_lu_96 (LeakyReLU)  (None, 2, 8)              0

 lstm_145 (LSTM)             (None, 2, 8)              544

 leaky_re_lu_97 (LeakyReLU)  (None, 2, 8)              0

 dropout_96 (Dropout)        (None, 2, 8)              0

 lstm_146 (LSTM)             (None, 4)                 208

 dropout_97 (Dropout)        (None, 4)                 0

 dense_48 (Dense)            (None, 1)                 5

=================================================================
```

Fig. 16 LSTM training model

The above CSV file with the count of vehicles was given as input to the LSTM model. 80% of the above data was trained for 50 epochs using the LSTM model, as shown in **Fig. 17**.

```
Epoch 1/50
<ipython-input-21-fb0b3525a387>:37: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use
  history=model.fit_generator(train_generator, epochs=50, validation_data=test_generator, shuffle=False)
158/158 [==============================] - 12s 29ms/step - loss: 0.1401 - mean_absolute_error: 0.3067 - root_mean_squared_error: 0.3743
Epoch 2/50
158/158 [==============================] - 2s 12ms/step - loss: 0.0611 - mean_absolute_error: 0.1953 - root_mean_squared_error: 0.2471
Epoch 3/50
158/158 [==============================] - 2s 12ms/step - loss: 0.0436 - mean_absolute_error: 0.1663 - root_mean_squared_error: 0.2088
Epoch 4/50
158/158 [==============================] - 2s 11ms/step - loss: 0.0471 - mean_absolute_error: 0.1758 - root_mean_squared_error: 0.2170
Epoch 5/50
158/158 [==============================] - 2s 11ms/step - loss: 0.0410 - mean_absolute_error: 0.1681 - root_mean_squared_error: 0.2025
Epoch 6/50
158/158 [==============================] - 2s 12ms/step - loss: 0.0413 - mean_absolute_error: 0.1661 - root_mean_squared_error: 0.2033
Epoch 7/50
158/158 [==============================] - 3s 17ms/step - loss: 0.0366 - mean_absolute_error: 0.1579 - root_mean_squared_error: 0.1914
```

Fig. 17 LSTM training process

6.5.2 Testing and Prediction

After training the model, 20% of the data was tested using mean squared error loss function, the Adam optimizer, and several evaluation metrics including mean absolute error, root mean squared error, mean absolute percentage error, and mean squared logarithmic error. The results are as shown in **Fig. 18**.

```
------------------------------------------------------------------
Mean Squared Error= 0.019209451973438263
Mean Absolute Error= 0.10108733177185059
Root Mean Squared Error= 0.13859817385673523
Mean Absolute Percentage Error= 19.15162467956543
Mean Squared Logarithmic Error= 0.007909279316663742
------------------------------------------------------------------
```

Fig. 18 Measured Regression Errors

A graph showing the difference between actual and predicted values was plotted for each class label. One such graph is shown in **Fig. 19**.

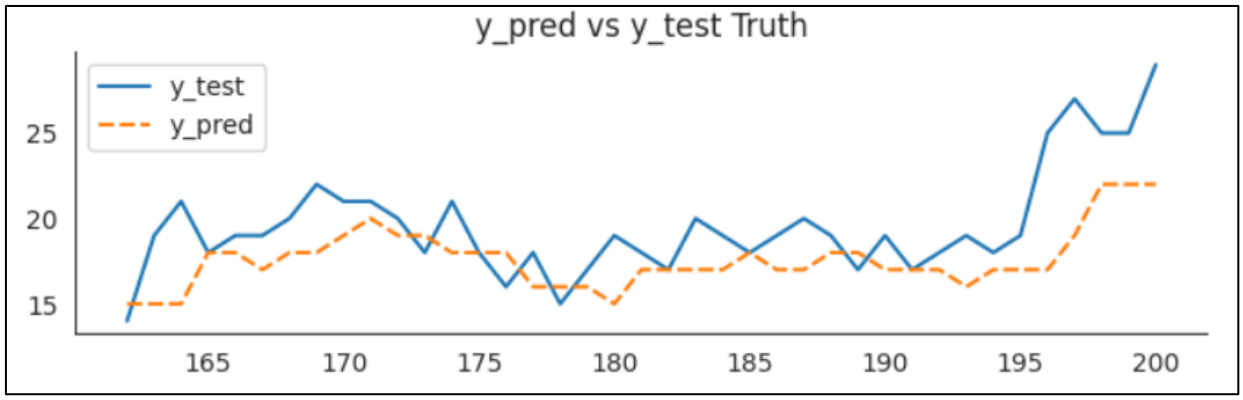


Fig. 19 y_pred vs y_test truth graph

CHAPTER 7

RESULT

After training and testing the YOLOV6 model, the challenging part is to detect and classify the image that is totally new to the model. When **Fig.20** is given as input, the model gives the output as shown in **Fig.21**. The output image has bounding boxes drawn around the detected objects along with confidence score of these bounding boxes.

Fig. 20 Input image

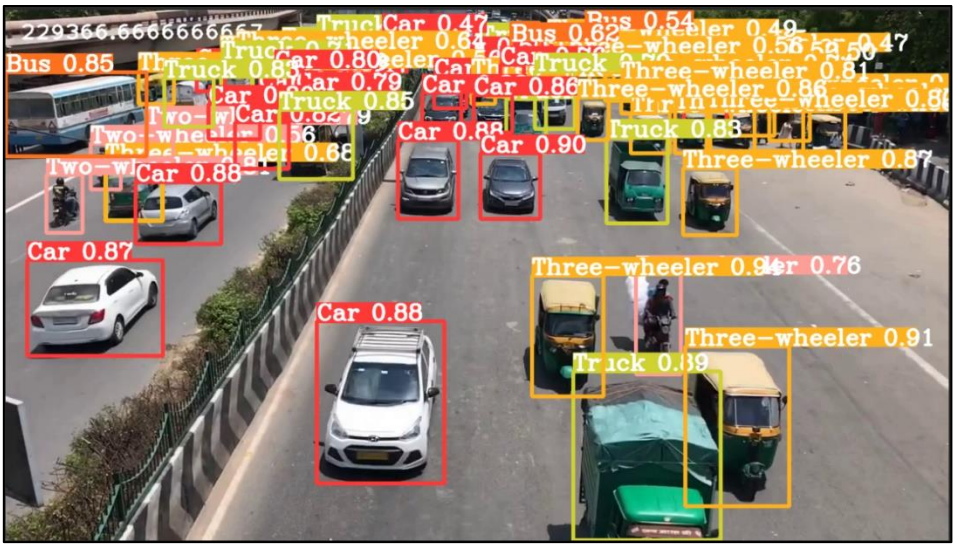


Fig. 21 Output image with bounding boxes

In addition to the above output image, we also obtain corresponding annotation text file as output, as shown in **Fig. 22**. The annotation text file for each image provides information about the objects detected in the image. Each line in the file represents a bounding box with the format `<class_label> <x_center> <y_center> <width> <height>`. The class_label is an integer indicating the object's class, and the x_center, y_center, width, and height values describe the location and size of the object's bounding box in the image.

```
4 0.308594 0.0944444 0.03125 0.0555556 0.439731
3 0.239453 0.100694 0.0242187 0.0375 0.439872
0 0.508203 0.0993056 0.0257812 0.0541667 0.448272
3 0.597656 0.0965278 0.0234375 0.0513889 0.452438
4 0.339453 0.0625 0.0195312 0.0388889 0.461698
4 0.51875 0.05625 0.03125 0.0597222 0.463801
3 0.702344 0.0965278 0.0203125 0.0375 0.466394
0 0.40625 0.0569444 0.0140625 0.0305556 0.467132
4 0.550781 0.0548611 0.0296875 0.0625 0.471967
0 0.328125 0.136806 0.0234375 0.0375 0.472237
3 0.583984 0.0708333 0.0195312 0.0388889 0.485744
0 0.476562 0.110417 0.028125 0.0513889 0.495057
3 0.519922 0.138889 0.0210938 0.0472222 0.504069
3 0.671875 0.117361 0.028125 0.0458333 0.504321
3 0.631641 0.123611 0.0257812 0.0611111 0.516251
4 0.135937 0.340278 0.059375 0.111111 0.517199
0 0.476953 0.131944 0.0304688 0.0444444 0.531397
```

Fig. 22 Annotation text file

Using python code on the above annotation text file, the count of vehicles belonging to each class label is calculated and stored in a CSV file as shown in **Fig. 23**.

Image	Car	Two-wheeler	Bus	Three-wheeler	Truck
i250	18	9	1	21	9
i251	17	11	1	21	10
i252	18	10	1	20	9
i253	13	5	1	20	10
i254	15	5	1	19	8
i255	13	9	1	20	10
i256	12	8	1	16	10
i257	14	6	1	19	12
i258	14	6	1	17	12
i259	19	6	1	18	10
i260	23	5	2	15	10
i261	22	7	1	18	12

Fig. 23 Count of vehicles

This count of vehicles is given as input to the LSTM model for time series prediction. After training the LSTM model, the future count is predicted based on the previous two counts. The result of the prediction along with the actual values are stored in a CSV file, as shown in **Fig. 24**.

	Car	Car_pred	Two-wheeler	Two-wheeler_pred	Bus	Bus_pred	Three-wheeler	Three-wheeler_pred	Truck	Truck_pred
162	14	15	7	4	1	1	18	17	8	9
163	19	15	8	6	1	1	17	17	9	8
164	21	15	9	7	1	1	16	17	6	8
165	18	18	8	7	1	1	18	16	7	8
166	19	19	5	7	1	1	19	17	7	7
167	19	17	6	5	1	1	13	17	9	7
168	20	18	6	6	1	1	12	17	13	8
169	22	18	7	6	1	1	13	15	9	10
170	21	19	9	6	1	1	17	15	9	9
171	21	20	6	7	2	1	14	15	13	9
172	20	19	9	6	1	2	16	16	11	10
173	18	19	9	7	1	1	16	16	9	10
174	21	18	7	8	2	1	15	16	11	9
175	18	18	7	6	3	2	17	16	14	9
176	16	19	7	6	1	2	15	16	13	11
177	18	16	7	6	1	2	15	16	11	11

Fig. 24 Actual and predicted count of vehicles of each class label

The mAP is calculated at different Intersection over Union (IoU) thresholds and different object sizes. When the Intersection over Union (IoU) criterion was set at 0.5, the YOLOV6 model performed best, with a score of 0.8737.

Mean squared error of the LSTM algorithm for the classes “Car”, “Two-wheeler”, “Bus”, “Three-wheeler” and “Truck” are 0.0192, 0.0238, 0.0221, 0.0110 and 0.0255, respectively.

CHAPTER 8
CONCLUSION

The study presents a model that uses deep learning algorithms and big data analytics for traffic forecasting and management. The YOLOV6 model successfully detects and classifies vehicles in real-time with a high mAP score. The LSTM algorithm accurately predicts future traffic counts at an intersection based on previous counts, with low mean squared error values for each class label.

By analyzing large volumes of traffic data and incorporating deep learning algorithms, it becomes possible to forecast traffic conditions in advance, enabling proactive measures to mitigate congestion. This integration empowers intelligent traffic management systems to optimize traffic flow, make informed decisions, and enhance transportation efficiency, ultimately resulting in smoother commutes and improved travel experiences for road users.

REFERENCES

1. B. M. Williams and L. A. Hoel, “Modeling and forecasting vehicular traffic flow as a seasonal ARIMA process: Theoretical basis and empirical results,” Journal of Transportation Engineering, vol. 129, no. 6, pp. 664-672, 2003.
2. M. Lippi, M. Bertini, and P. Frasconi, "Short-term traffic flow forecasting: An experimental comparison of time-series analysis and supervised learning,” IEEE Transactions on Intelligent Transportation Systems, vol. 14, no. 2, pp. 871-882, 2013.
3. Zhang, Huaizhong & Liptrott, Mark & Bessis, Nik & Cheng, Jianquan. (2019). Real-Time Traffic Analysis using Deep Learning Techniques and UAV based Video. 10.1109/AVSS.2019.8909879.
4. J. Tiwari, A. Deshmukh, G. Godepure, U. Kolekar and K. Upadhyaya, "Real Time Traffic Management Using Machine Learning," 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), Vellore, India, 2020, pp. 1-5, doi: 10.1109/ic-ETITE47903.2020.462.
5. S. Ali, M. Hanzla and A. A. Rafique, "Vehicle Detection and Tracking from UAV Imagery via Cascade Classifier," 2022 24th International Multitopic Conference (INMIC), Islamabad, Pakistan, 2022, pp. 1-6, doi: 10.1109/INMIC56986.2022.9972959.
6. N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 2005, pp. 886-893 vol. 1, doi: 10.1109/CVPR.2005.177.
7. Li, W. (2021). Analysis of Object Detection Performance Based on Faster R-CNN. *Journal of Physics*, *1827*(1), 012085. https://doi.org/10.1088/1742-6596/1827/1/012085
8. Kumar, A., Zhang, Z., & Lyu, H. (2020). Object detection in real time based on improved single shot multi-box detector algorithm. *Eurasip Journal on Wireless Communications and Networking*, *2020*(1). https://doi.org/10.1186/s13638-020-01826-x
9. Dai, J. (2016, May 20). *R-FCN: Object Detection via Region-based Fully Convolutional Networks*. arXiv.org. https://arxiv.org/abs/1605.06409

10. Andrie, Rosa & Syahputro, Bimo & Suprianto, Dodit & Handayani, Anik. (2020). Prediction of Traffic Density Using YOLO Object Detection and Implemented in Raspberry Pi 3b + and Intel NCS 2. 391-395. 10.1109/ICOVET50258.2020.9230145.
11. Li, C. (2022, September 7). *YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications*. arXiv.org. https://arxiv.org/abs/2209.02976
12. Belhadi, A., Djenouri, Y., Djenouri, D., & Lin, J. C. (2020). A recurrent neural network for urban long-term traffic flow forecasting. *Applied Intelligence*, *50*(10), 3252–3265. https://doi.org/10.1007/s10489-020-01716-1
13. Zhang, Y. (2020). Short-Term Traffic Flow Prediction Methods: A Survey. *Journal of Physics: Conference Series*, *1486*(5), 052018. https://doi.org/10.1088/1742-6596/1486/5/052018
14. Xiao, Y., & Yin, Y. (2019). Hybrid LSTM Neural Network for Short-Term Traffic Flow Prediction. *Information*, *10*(3), 105. https://doi.org/10.3390/info10030105
15. Afrin, T., & Yodo, N. (2021). A Long Short-Term Memory-based correlated traffic data prediction framework. *Knowledge Based Systems*, *237*, 107755. https://doi.org/10.1016/j.knosys.2021.107755
16. Pan Wu, Zilin Huang, Yuzhuang Pian, Lunhui Xu, Jinlong Li, Kaixun Chen, "A Combined Deep Learning Method with Attention-Based LSTM Model for Short-Term Traffic Speed Forecasting", Journal of Advanced Transportation, vol. 2020, Article ID 8863724, 15 pages, 2020. https://doi.org/10.1155/2020/8863724
17. Touzani, Y., Douzi, K. An LSTM and GRU based trading strategy adapted to the Moroccan market. *J Big Data* **8**, 126 (2021). https://doi.org/10.1186/s40537-021-00512-z
18. https://stackoverflow.com/
19. https://www.geeksforgeeks.org/
20. https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/
21. https://github.com/meituan/YOLOv6
22. https://www.javatpoint.com/