INDUSTRY INTERNSHIP

ON

VEHICLE DETECTION AND COUNTING SYSTEM

A report submitted
In fulfilment of the requirements of the degree of

BACHELOR OF ENGINEERING

By
SURAJ KUMAR CHOUDHARY

201304007

Supervised by

MS. JASMEEN KAUR

DEPARTMENT OF INFORMATION TECHNOLOGY

MAHANT BACHITTAR SINGH

COLLEGE OF ENGINEERING AND TECHNOLOGY

BABLIANA, JEEVAN NAGAR ROAD, (J&K), JAMMU – 181101 (INDIA)

June 2024

MAHANT BACHITTAR SINGH COLLEGE OF ENGINEERING & TECHNOLOGY, JAMMU

DECLARATION

I hereby certify that the work which is being presented in this internship report titled "**VEHICLE DETECTION AND COUNTING SYSTEM**" is for fulfilment of the requirement for the award of Degree of **Bachelor of Engineering** submitted in the **Department of Information Technology, MBSCET, Jammu** is an authentic record of my own work carried out under the supervision of Ms. Jasmeen Kaur**,** Asst. Professor.

The work has not formed the basis for the award of any other degree or diploma, in this or any other Institution or University. In keeping with the ethical practice in reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

Project Guide	**Project Coordinator**	**H.OD, IT**

MAHANT BACHITTAR SINGH

COLLEGE OF ENGINEERING & TECHNOLOGY,

JAMMU

CERTIFCATE

This is to certify that the internship titled "**VEHICLE DETECTION AND COUNTING SYSTEM**" submitted by **SURAJ KUMAR CHOUDHARY** bearing **Roll No 201304007** to **MBSCET**, Affiliated to **University of Jammu** in fulfillment for the award of the degree of **Bachelor of Engineering** is a bonafide record of work carried out by him. The contents of this report, in full or in parts, have not been submitted to any other Institution or University for the award of any degree or diploma.

HOD IT Dept PRINCIPAL

EXTERNAL EXAMINAR

ACKNOWLEDGEMENT

It gives me great pleasure in presenting the Industry Internship report on "**VEHICLE DETECTION AND COUNTING SYSTEM**".

This endeavour would have been impossible without the grace of **Almighty** and his Holiness **SHRIMAN MAHANT MANJIT SINGH JI** Sole Trustee MBSCET. I am eternally grateful for their perpetual blessings.

It has been a privilege and honour to work under the kind guidance and sharp vigil of my respected Guide, **Ms JASMEEN KAUR (Asst. Prof. IT Dept.**). Her patience, caring presence, encouragement and guidance has made it possible to put this work together. Her continuous encouragement helped me immensely in carrying out this work, I shall always remain indebted and thankful to her.

This work would not have been a reality without the permission of **Ms. Jasmeen Kaur**, Head of Department, for her wholehearted support guide to whom I owe, helped me in overcoming her teaching problems. I shall always remain indebted and thankful to her.

I would like to pay my sincere thanks to our esteemed and honourable Principal **Prof. (Dr.) Dinesh Kumar Gupta** for their unfailing support and providing me with all the necessary support and guidance whenever required.

It gives me great pleasure in acknowledging service Industry Internship with all necessary support for their unfailing support and providing me with all the resources that were vital for successful completion.

Finally, I would like to take this opportunity to thank all my teachers, family and friends who have helped us in this endeavour. My apologies to anyone else who helped me whom I have not mentioned here. I am grateful to them too.

Suraj Kumar Choudhary

201304007

LIST OF FIGURES

LIST OF TABLES

CONTENTS

ABSTRACT

Vehicle Counting and Classification Web App is an advanced object detection system engineered to monitor and analyze vehicular traffic using real-time video feeds. Leveraging the robust capabilities of the OpenCV library, along with state-of-the-art technologies such as YOLOv5 for object detection and DeepSORT for object tracking, this application aims to accurately count and classify vehicles, distinguishing between cars and buses. This report delves into the objectives, scope, implementation, and potential impact of the project, highlighting its significance in traffic management and urban planning.

Objectives

The core objectives of the Vehicle Counting and Classification Web App include:

1. **Vehicle Counting and Classification**: The primary function is to count vehicles passing through a specific area and classify them as either cars or buses using sophisticated object detection algorithms, specifically YOLOv5.
2. **User-Friendly Interface**: Designed with an intuitive interface, the application ensures that users with minimal technical knowledge can operate it effectively.
3. **Ease of Use**: The system is straightforward, supported by comprehensive documentation and a simple setup process, minimizing the learning curve.
4. **Quick and Responsive**: Optimized for rapid response times, the application facilitates real-time tracking and counting, essential for dynamic traffic environments.

Scope

The application's extensive scope offers significant advantages across multiple domains:

- **Aid for Traffic Police**: By centralizing traffic monitoring, the application enables traffic officers to efficiently manage vehicle flow using real-time data, aiding in informed decision-making and improved traffic flow.
- **Maintaining Records**: Automating the recording process, the system ensures accurate, real-time data capture, freeing officials from manual tasks and enhancing data analysis capabilities.

- **Traffic Surveillance Control**: Easily deployable with minimal infrastructure, the system helps monitor congestion levels, allowing traffic officers to manage traffic effectively and coordinate responses to varying traffic conditions.

Implementation Using OpenCV, YOLOv5, and DeepSORT

The project employs several key steps utilizing the OpenCV library, YOLOv5 for object detection, and DeepSORT for tracking:

- **Setting Up the Environment**: Installation of OpenCV, YOLOv5, DeepSORT, and necessary libraries such as NumPy, followed by setting up a camera feed or using prerecorded videos.
- **Preprocessing**: Video frames are converted to grayscale to reduce complexity, with techniques like Gaussian Blur applied to enhance detection accuracy.
- **Vehicle Detection**: YOLOv5 is employed to detect vehicles in the video frames, leveraging its advanced object detection capabilities to identify and classify vehicles accurately.
- **Vehicle Classification**: Detected objects are classified as cars or buses using the features extracted by YOLOv5, which is trained on a labeled dataset of vehicle images.
- **Counting and Tracking**: DeepSORT is utilized for tracking vehicles across frames, ensuring accurate counting and maintaining unique identifiers for each vehicle to avoid double counting.
- **User Interface**: A web-based interface is developed using Libraries like Streamlit , displaying real-time video feeds, vehicle counts, and classification results, with options for generating reports and viewing historical data.

Potential Impact

The Vehicle Counting and Classification Web App promises significant contributions to traffic management and urban planning:

- **Enhanced Traffic Management**: Real-time data facilitates dynamic traffic signal adjustments, reducing wait times and improving flow, while enabling quicker responses to incidents.
- **Data-Driven Decisions**: Accurate vehicle counts and classifications provide valuable insights for urban planning and infrastructure development, aiding in resource allocation and road expansion planning.

- **Improved Public Safety**: Better traffic management reduces accidents and improves emergency response times, with the system also aiding in traffic violation management.
- **Environmental Benefits**: Reduced congestion leads to lower emissions, contributing to cleaner air and a healthier environment.

The Vehicle Counting and Classification Web App is a powerful solution for modern traffic management, harnessing the capabilities of OpenCV, YOLOv5, and DeepSORT to deliver a user-friendly, efficient, and responsive system. Its benefits extend from aiding traffic police to enhancing public safety and environmental health, underscoring its value in urban management and planning. With continuous advancements in computer vision and machine learning, the system holds vast potential for future improvements and applications.

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

The Vehicle Counting and Classification Web App is a sophisticated object detection system designed to track vehicles using their coordinates in images or videos. This system leverages the capabilities of the OpenCV library, a powerful tool for real-time computer vision. The primary focus of this application is to provide accurate vehicle counting and classification, differentiating between cars and buses. This document outlines the objectives, scope, and potential impact of the project, providing a detailed overview of its implementation and benefits.

1.2 Objective

The main objectives of the Vehicle Counting and Classification Web App are:-

- **Vehicle Counting and Classification:** The core functionality is to count the number of vehicles passing through a designated area and classify each as either a car or a bus. This is achieved through advanced object detection algorithms that process video feeds in real-time.
- **User-Friendly Interface:** The application is designed with a user-friendly interface to ensure ease of use. Users with minimal technical expertise can operate the system efficiently, making it accessible to a broad audience.
- **Ease of Use:** The system is intuitive and straightforward, reducing the learning curve for new users. Detailed documentation and a simple setup process further enhance usability
- **Quick and Responsive:** The application is optimized for quick response times, ensuring real-time tracking and counting. This responsiveness is crucial for applications in dynamic environments, such as traffic management.

1.3 Scope

The scope of the Vehicle Counting and Classification Web App is extensive, offering numerous benefits across various domains.

Key areas of impact include:-

Aid for Traffic Police:

- The application provides a centralized monitoring system for traffic police. By installing cameras at key locations, traffic officers can monitor vehicle flow from a single location.
- This centralized system helps in efficiently managing traffic by providing real-time data on vehicle counts and types, which can be used to make informed decisions and improve traffic flow.

Maintaining Records:

- Manual recording of vehicle data is challenging and time-consuming. This application automates the process, recording data in real-time as vehicles pass by.
- Automated record-keeping ensures accuracy and saves time, allowing officials to focus on other critical tasks. The system can generate reports and statistics, aiding in data analysis and decision-making.

Traffic Surveillance Control:

- The system can be deployed at various locations with minimal infrastructure, requiring only a camera and a connection to the central system.
- In high-traffic areas, the application can help monitor congestion levels. Traffic officers can use this data to manage traffic more effectively, redirecting flows or taking preemptive measures to avoid bottlenecks.
- Information gathered can be forwarded to subsequent toll officers or traffic management centers, ensuring a coordinated response to traffic conditions.

Implementation Using OpenCV

- The implementation of this project involves several key steps, primarily utilizing the OpenCV library for image processing and object detection. OpenCV is an open-source computer vision library that provides tools for real-time image and video processing. Here's an outline of the implementation process:

1.4 Setting Up the Environment:

- Install OpenCV and other necessary libraries such as NumPy for numerical operations.
- Set up a camera feed or use pre-recorded videos for processing.

Preprocessing:

- Capture video frames and convert them to grayscale to reduce computational complexity.
- Apply image preprocessing techniques like Gaussian Blur to remove noise and improve detection accuracy.

Vehicle Detection:

- Use background subtraction methods to detect moving objects in the video frames.
- Apply contour detection to identify potential vehicles based on shape and size.

Vehicle Classification:

- Extract features from detected objects and use machine learning models to classify them as cars or buses.
- Train the classification model using a labeled dataset of vehicle images.

Counting and Tracking:

- Implement tracking algorithms to follow detected vehicles across frames, ensuring accurate counting.
- Use unique identifiers for each vehicle to avoid double counting.

User Interface:

- Develop a web-based interface using frameworks like Flask or Django.
- Display real-time video feed, vehicle count, and classification results on the dashboard.
- Provide options for generating reports and viewing historical data.

Potential Impact

- The Vehicle Counting and Classification Web App has the potential to significantly impact traffic management and urban planning. By providing accurate, real-time data on vehicle flows, it can help reduce congestion, improve traffic safety, and optimize the use of infrastructure. Here are some specific benefits:

Enhanced Traffic Management:

- Real-time data allows for dynamic traffic signal adjustments, reducing wait times and improving traffic flow.
- Traffic officers can respond more quickly to incidents and congestion, improving overall road safety.

Data-Driven Decisions:

- Accurate vehicle counts and classifications provide valuable data for urban planning and infrastructure development.
- Authorities can use this data to plan road expansions, optimize public transport routes, and allocate resources more effectively.

Improved Public Safety:

- Better traffic management leads to fewer accidents and improved emergency response times.
- The system can help in identifying traffic violations and managing enforcement more efficiently.

Environmental Benefits:

- Reduced congestion leads to lower emissions, contributing to cleaner air and a healthier environment.

The Vehicle Counting and Classification Web App is a powerful tool for modern traffic management. By leveraging the capabilities of the OpenCV library, this application provides a user-friendly, efficient, and responsive system for counting and classifying vehicles. Its wide-ranging benefits, from aiding traffic police to enhancing public safety and environmental health, make it a valuable asset in the realm of urban management and planning. With ongoing advancements in computer vision and machine learning, the potential for further improvements and applications of this system is vast.

CHAPTER 2

LITERATURE REVIEW

2.1 OVERVIEW

The literature survey explores the dynamic landscape of vehicle detection and counting systems, pivotal for enhancing traffic management and urban infrastructure planning. This chapter synthesizes existing research methodologies and technological advancements, spanning from traditional computer vision techniques to state-of-the-art deep learning models. It examines various algorithms, datasets, and evaluation metrics used to achieve accurate and efficient vehicle detection and counting. By reviewing these approaches, the study aims to provide a comprehensive understanding of current trends and challenges, laying the groundwork for proposing innovative solutions in the development of robust and scalable systems for real-world applications.

2.2 Different Reviews of Different Authors

[1] Huansheng Song, Haoxiang Liang, Huaiyu Li, Zhe Dai & Xu Yun, 'Vision-based vehicle detection and counting system using deep learning in highway scenes' , 2019

This study established a high-definition vehicle object dataset from the perspective of surveillance cameras and proposed an object detection and tracking method for highway surveillance video scenes. A more effective ROI area was obtained by the extraction of the road surface area of the highway. The YOLOv3 object detection algorithm obtained the end-to-end highway vehicle detection model based on the annotated highway vehicle object dataset. To address the problem of the small object detection and the multi-scale variation of the object, the road surface area was defined as a remote area and a proximal area. The two road areas of each frame were sequentially detected to obtain good vehicle detection results in the monitoring field. The position of the object in the image was predicted by the ORB feature extraction algorithm based on the object detection result. Then, the vehicle trajectory could be obtained by tracking the ORB features of multiple objects. Finally, the vehicle trajectories were analyzed to collect the data under the current highway traffic scene, such as driving direction, vehicle type, and vehicle number. The experimental results verified that the proposed vehicle detection and tracking method for highway surveillance video scenes has good performance and practicability. Compared with the traditional method of monitoring vehicle traffic by hardware, the method of this paper is low in cost and high in stability and does not require large-scale construction or installation work on existing monitoring equipment. According to the research reported in this paper, the surveillance camera can be further calibrated to obtain the internal and external parameters of the camera. The position information of the vehicle trajectory is thereby converted from the image coordinate system to the world coordinate system. The vehicle speed can be calculated based on the calibration result of the camera. Combined with the presented vehicle detection and tracking methods, abnormal parking events and traffic jam events can be detected to obtain more abundant traffic information.

[2] Md Abdur Rouf, Qing Wu, Xiaoyu Yu, Yuji Iwahori , 'Real-time Vehicle Detection, Tracking, and Counting System Based on YOLOv7' , 2019

The paper addresses the critical need for efficient traffic monitoring systems on highways through real-time vehicle detection, tracking, and counting. Emphasizing the pivotal role of these systems in traffic management and safety, the study focuses on enhancing the YOLOv7 algorithm to improve detection accuracy, especially for small targets and under varying environmental conditions. By integrating object detection, tracking, and counting functionalities, the system aims to provide robust traffic flow statistics and insights into vehicle movement patterns. Experimental validation on a modified MS COCO dataset demonstrates the system's capability to accurately detect and count vehicles in real-time videos. The research highlights the potential of YOLOv7 in enhancing real-time traffic surveillance and management, contributing to safer and more efficient transportation systems.

[3] Karthik Srivathsa D and Dr. Kamalraj, 'Real-time Vehicle Detection, Tracking, and Counting System Based on YOLOv7' , 2021

This research paper focuses on developing an effective method for vehicle detection and counting, utilizing OpenCV technologies to aid in traffic control. The proposed system addresses the critical issue of traffic congestion in urban areas, driven by an increasing number of vehicles and inadequate road infrastructure. The methodology leverages video-based techniques, specifically using background subtraction to identify moving vehicles within video sequences. The process involves several key steps: converting video frames from color to grayscale, subtracting the

background, and applying thresholding to distinguish between foreground (moving vehicles) and background pixels. Further refinement is achieved through adaptive morphological operations and hole filling to eliminate noise and shadows, ensuring a clearer identification of vehicles.
To improve the accuracy of vehicle detection, the study employs a virtual detection zone within a defined region of interest (ROI) on the road. Vehicles are counted when they enter this zone, with their movements tracked frame by frame. The centroids of detected vehicles are calculated, and each vehicle is counted only once when it crosses the virtual detection zone. This approach minimizes errors in vehicle counting and enhances the reliability of the system. The experimental results, depicted through various figures and flowcharts, demonstrate the high accuracy of the proposed method, achieving a vehicle counting accuracy ranging from 95% to 99% based on different video inputs.

The paper concludes that the implemented system, based on Python and computer vision techniques, is effective for real-time vehicle detection and counting, providing valuable data for traffic management. This data can be utilized by traffic authorities to optimize traffic flow and by road users to make informed travel decisions. The high accuracy of the system underscores its potential application in real-world traffic monitoring and control scenarios. The study acknowledges the guidance and support of academic mentors and references prior research in the field to contextualize its contributions.

[4] Shiva Kamkar and Reza Safabakhsh, Vehicle detection, counting and classification in various conditions, 2016

This paper present a comprehensive methodology aimed at enhancing intelligent transportation systems (ITS). The study addresses the pivotal task of vehicle detection within traffic monitoring systems, crucial for subsequent traffic management applications. Their proposed approach employs an active basis model (ABM) for detecting vehicle candidates in video frames, leveraging edge information for robustness against varying lighting, weather conditions, camera vibration, and image blurring. Vehicle verification incorporates symmetry checks to accommodate real-world scenarios like lane changes or varying viewpoints. Subsequently, vehicles are classified into small (e.g., cars), medium (e.g., vans), and large (e.g., buses, trucks) categories using features extracted from time-spatial images and grey-level co-occurrence matrices (GLCM), followed by classification using a random forest (RF) algorithm. The effectiveness of the method is demonstrated through experiments across seven video streams, showcasing its adaptability and high performance under diverse environmental challenges typical of highway conditions. The paper concludes with suggestions for further enhancements, such as GPU acceleration and additional texture-based features, to further improve algorithm efficiency and runtime performance in real-time applications.

[5] Mr. P. Nagaraj1 Raj, Varshith Reddy, Mohammed, SriChandana Reddy, V Santhosh, 'Automatic Vehicle Detection and Counting System' , 2020

This paper addresses the critical task of vehicle detection and counting using computer vision techniques, primarily leveraging OpenCV and object detection algorithms. The paper emphasizes the importance of such systems in traffic monitoring, toll collection, and other applications where real-time vehicle data is crucial. The methodology begins with video input, from which frames are extracted and background estimation is performed to isolate moving objects. The BackgroundSubtractorMOG2 algorithm is utilized for foreground/background segmentation, enhancing the accuracy of vehicle detection by identifying changes between frames. Pre-processing steps such as morphological operations (dilation and erosion) are applied to refine object boundaries, followed by vehicle detection using bounding boxes and centroid detection. Vehicles are classified based on their dimensional ratios, distinguishing between cars and larger vehicles like trucks. Importantly, the system includes a counting mechanism where vehicles passing through predefined regions of interest are tallied based on centroid crossings. The paper concludes by highlighting the system's potential applications in traffic management and its cost-effectiveness, while acknowledging limitations such as occlusion affecting detection accuracy. Overall, the study contributes to advancing intelligent transportation systems through accessible and efficient vehicle detection methodologies.

2.3 Emerging Trends and Technology used

- **NumPy**

NumPy, an open-source library, facilitates computing with multi-dimensional matrices and arrays. Widely utilized in scientific and industrial research, NumPy simplifies data manipulation and mathematical operations on arrays. Its extensive API is leveraged in various data science and scientific Python packages, enhancing efficiency and speed in data analysis and modeling tasks.

- **OpenCV**

OpenCV, short for Open Source Computer Vision Library, stands as a cornerstone in the realm of computer vision, wielding immense influence in the project's endeavors towards object detection.
In the realm of digital image processing, OpenCV emerges as a quintessential tool, offering a rich suite of functionalities tailored to the analysis and interpretation of visual data. Its versatile toolkit empowers developers to tackle a myriad of challenges spanning from face recognition to object detection, thus underpinning the project's overarching objectives with its robust capabilities.

At the heart of OpenCV lies its prowess in computer vision techniques, enabling the extraction of meaningful insights from digital images. Through a sophisticated array of algorithms and methodologies, OpenCV facilitates the transformation of raw pixel data into actionable information, thereby unlocking a realm of possibilities for image analysis and understanding. Within the project's context, this translates into the ability to discern and classify vehicles within input images, a task crucial for various applications ranging from traffic monitoring to autonomous navigation systems.

The project harnesses OpenCV's extensive arsenal of tools and functions to tackle the intricacies of object detection. Leveraging its robust framework, developers can implement sophisticated algorithms capable of detecting objects of interest within complex visual environments. OpenCV's object detection capabilities are particularly noteworthy, offering a range of approaches including Haar cascades, HOG (Histogram of Oriented Gradients), and deep learning-based methods. These methodologies empower the project to discern vehicles amidst diverse backgrounds and lighting conditions, thus bolstering its efficacy in real-world scenarios.

Central to OpenCV's utility is its seamless integration with the Python programming language, a feature that aligns seamlessly with the project's implementation framework. Python's simplicity and readability, coupled with OpenCV's powerful capabilities, form a symbiotic relationship that streamlines the development process. Developers can leverage Python's expressive syntax to orchestrate complex image processing workflows, while OpenCV's optimized algorithms ensure efficient execution and performance.

Moreover, OpenCV's open-source nature fosters a vibrant community ecosystem, characterized by collaborative development and knowledge sharing. This ecosystem serves as a valuable resource for the project, offering access to a wealth of documentation, tutorials, and pre-trained models. Developers can tap into this collective wisdom to expedite development cycles, troubleshoot issues, and explore innovative approaches to object detection.

In addition to its core functionalities, OpenCV offers a range of auxiliary features that augment the project's capabilities. These include tools for image filtering, feature extraction, and geometric transformations, each contributing to the project's versatility and adaptability. Furthermore, OpenCV's compatibility with various hardware platforms, including CPUs, GPUs, and embedded systems, ensures scalability and portability across diverse deployment scenarios.

Deep SORT

Deep SORT is an enhancement of the original SORT algorithm, which stands for Simple Online and Real time Tracking. SORT is a well-known tracking algorithm that uses Kalman filters and the Hungarian algorithm for data association. However, SORT has limitations, particularly in scenarios with occlusions and complex movements, where it can struggle to maintain accurate tracking.

Deep SORT addresses these limitations by integrating deep learning-based appearance descriptors with the traditional SORT framework. This combination allows for more robust tracking, especially in challenging environments. Here's a breakdown of how Deep SORT works:

1. **Detection:** The first step involves detecting objects (vehicles, in this case) in each frame of the video. This is typically achieved using a pre-trained object detection model like YOLO (You Only Look Once) or SSD (Single Shot MultiBox Detector).

2. **Feature Extraction:** Once the objects are detected, Deep SORT extracts appearance features using a deep convolutional neural network (CNN). These features help in distinguishing between different objects, even if they have similar shapes and sizes.

3. **Kalman Filter:** For each detected object, a Kalman filter predicts the object's future position based on its previous state (position, velocity, etc.). This prediction helps in maintaining a continuous tracking of the object across frames.

4. **Data Association:** The Hungarian algorithm is used for data association, which matches the predicted positions of objects from the Kalman filter with the newly detected objects. This step ensures that each object is consistently tracked over time.

5. **Re-identification:** The deep appearance features extracted earlier are used for re-identification. This means that if an object temporarily leaves the frame and reappears later, Deep SORT can recognize it as the same object, thus improving tracking accuracy.

Machine Learning

The concept of Machine Learning (ML) has been in existence since its inception by Arthur Samuel in 1959, and it has since undergone significant development and refinement through the contributions of numerous researchers. As a subset of Artificial Intelligence (AI), ML empowers computer applications to dynamically learn from data without explicit programming instructions, thereby enabling them to uncover hidden insights and patterns within the data.

ML achieves this by employing statistical techniques to analyze and recognize patterns and relationships between data and events. Through the utilization of computer algorithms, ML models are constructed to learn from data and make predictions based on the patterns observed. Unlike traditional software applications, which operate within predefined rules set by programmers, ML algorithms possess the capability to adapt and improve their performance over time through the process of learning.

The field of ML encompasses three fundamental types of learning:

1. **Supervised Learning:** Supervised learning is commonly employed for practical AI problems, where the algorithm is provided with input-output pairs for training. These algorithms endeavor to learn the mapping between input and output factors through a

mapping function. During the training phase, the algorithm is monitored to assess its performance, and adjustments are made to enhance its learning. Supervised learning algorithms can be categorized into regression and classification, with each serving specific purposes based on the nature of the problem being addressed.

Some commonly used supervised learning algorithms include:-

Support Vector Machines, Naive Bayes, Logistic Regression, Linear Regression, Decision Trees, k-Nearest Neighbor, and Neural Networks. However, the selection of the most suitable algorithm depends on the specific characteristics of the task at hand, as no single algorithm is universally optimal for all scenarios.

2. **Unsupervised Learning:** In contrast to supervised learning, unsupervised learning algorithms do not have access to labeled output data. Instead, they seek to identify hidden patterns and structures within the dataset without explicit guidance. Unsupervised learning tasks are typically categorized into clustering and association problems. Clustering involves partitioning data into groups based on similarities, while association aims to uncover relationships and associations between different data points.

Popular algorithms in unsupervised learning include:-

K-means clustering and the Apriori algorithm for association analysis. Despite the absence of labeled data, unsupervised learning algorithms play a crucial role in data exploration and pattern discovery, making them valuable tools in various domains.

3. **Reinforcement Learning:** Reinforcement learning operates on the principle of trial and error, analogous to a child learning through interactions with the environment. In this paradigm, an agent interacts with an environment, receiving rewards or punishments based on its actions. Over time, the agent learns to optimize its behavior to maximize rewards and minimize penalties, thereby achieving its objectives.

Reinforcement learning algorithms are particularly suited for scenarios where explicit feedback is not readily available, such as in autonomous navigation and game playing. By iteratively exploring and exploiting different actions, reinforcement learning agents can acquire sophisticated strategies to tackle complex tasks.

In summary, Machine Learning encompasses a diverse array of techniques and algorithms aimed at extracting insights and making predictions from data. From supervised and unsupervised learning to reinforcement learning, each approach offers unique capabilities and applications, contributing to the advancement of AI and enabling a wide range of practical solutions across various domains.

YOLOv5

YOLOv5 is a state-of-the-art object detection model that has gained popularity for its speed and accuracy. Developed by Ultralytics, YOLOv5 is the latest iteration in the YOLO family of models, known for their real-time detection capabilities. YOLOv5 introduces several enhancements over its predecessors, making it particularly well-suited for applications like vehicle detection. Here's an overview of its key features:

1. **Speed and Efficiency:** YOLOv5 is designed to be fast and efficient, capable of processing high-resolution images in real-time. This speed is crucial for applications that require immediate feedback, such as traffic monitoring.

2. **Accuracy:** YOLOv5 achieves high accuracy in object detection, thanks to its improved architecture and training techniques. It can accurately identify and localize multiple objects in a single frame, even in complex environments.

3. **Scalability:** YOLOv5 offers different model sizes (YOLOv5s, YOLOv5m, YOLOv5l, and YOLOv5x) to balance the trade-off between speed and accuracy. This flexibility allows users to choose a model that best fits their specific needs and computational resources.

4. **Ease of Use:** YOLOv5 is user-friendly, with straightforward installation and implementation processes. It supports popular deep learning frameworks like PyTorch, making it accessible to a wide range of developers and researchers.

Integration of YOLOv5 in Vehicle Detection

In our Vehicle Detection and Counting System, YOLOv5 serves as the primary object detection component. The process can be broken down into the following steps:

1. **Video Input:** The system receives video feeds from traffic cameras, which serve as the input for the detection and tracking processes.

2. **Frame Extraction:** Each video feed is processed frame-by-frame to enable real-time analysis.

3. **Vehicle Detection:** YOLOv5 is applied to each frame to detect vehicles. The model outputs bounding boxes around detected vehicles, along with class labels and confidence scores. These bounding boxes are essential for tracking and counting vehicles accurately.

Combined Application in Vehicle Detection and Counting

The integration of YOLOv5 for detection and Deep SORT for tracking provides a robust framework for vehicle detection and counting. Here's how it works:

1. Enhanced Accuracy in Detection and Tracking

By combining YOLOv5's high detection accuracy with Deep SORT's robust tracking capabilities, the system ensures precise vehicle detection and tracking. YOLOv5 effectively detects vehicles in each frame, while Deep SORT maintains continuous tracking, even in challenging conditions such as occlusions or overlapping vehicles.

2. Real-time Processing

Both YOLOv5 and Deep SORT are designed for real-time processing. YOLOv5's efficient architecture allows it to process frames quickly, while Deep SORT's lightweight tracking algorithm ensures that the system can handle real-time video feeds. This is crucial for applications like traffic monitoring and management, where immediate data is essential.

3. Handling Occlusions and Complex Movements

Traffic scenarios often involve vehicles being temporarily occluded by other vehicles, infrastructure, or environmental factors. Deep SORT’s re-identification capability allows it to handle these situations effectively. If a vehicle is temporarily hidden from view, the system can re-identify it once it reappears, ensuring continuity in tracking and counting.

4. Scalability

The system can be scaled to monitor multiple lanes and even entire road networks. YOLOv5’s flexibility in model sizes and Deep SORT’s robustness and accuracy mean the system can handle various traffic densities and patterns, making it suitable for urban, suburban, and highway environments.

Implementation Details

To implement the Vehicle Detection and Counting System using YOLOv5 and Deep SORT, we follow a structured approach:

1. **Pre-processing:** Video feeds from traffic cameras are pre-processed to enhance image quality and ensure consistent lighting and contrast. This step is crucial for accurate object detection.

2. **Object Detection with YOLOv5:** YOLOv5 is applied to each frame to detect vehicles. The model outputs bounding boxes, class labels, and confidence scores for each detected vehicle.

3. **Feature Extraction:** For each detected vehicle, deep appearance features are extracted using a pre-trained CNN. These features capture the unique visual characteristics of each vehicle

.

4. **Tracking Initialization:** Initial tracks are created for each detected vehicle. The Kalman filter predicts the vehicle's future position based on its initial state.

5. **Data Association and Re-identification:** The Hungarian algorithm matches the predicted positions with newly detected vehicles, and the deep appearance features are used to re-identify vehicles that re-enter the frame.

6. **Counting Logic:** A counting mechanism is implemented to increment the vehicle count each time a vehicle crosses a predefined virtual line. This ensures accurate counting even if vehicles reappear after being temporarily occluded.

Benefits and Applications

The integration of YOLOv5 and Deep SORT into the Vehicle Detection and Counting System offers several benefits:

- **Accurate Traffic Monitoring:** Provides precise vehicle counts, which are essential for traffic management and planning.

- **Improved Road Safety:** Enables real-time detection of traffic anomalies, such as sudden stops or accidents, allowing for quick response.

- **Data-Driven Decision Making:** Generates valuable data for urban planners and transportation authorities to make informed decisions regarding infrastructure development and traffic regulations.

- **Scalability and Flexibility:** Can be deployed across various environments, from busy city intersections to remote highways, adapting to different traffic conditions.

Streamlit Library

Streamlit is an open-source Python library designed to make it easy for data scientists and machine learning engineers to create interactive and shareable web applications. It focuses on simplicity and rapid prototyping, allowing users to build applications with minimal effort and code.

Key Features of Streamlit:

1. **Ease of Use**:
   - Requires minimal coding to convert Python scripts into web apps.
   - Uses a simple and intuitive API, enabling quick development.

2. **Interactive Widgets**:
   - Provides a variety of interactive widgets such as sliders, buttons, and text inputs to create dynamic applications.
   - Widgets can be used to filter data, adjust parameters, and update visualizations in real-time.

3. **Data Visualization**:
   - Supports integration with popular Python libraries like Matplotlib, Plotly, and Altair.
   - Allows for the easy display of static and interactive charts.

4. **Real-time Updates**:
   - Automatically updates the application as the underlying code changes.
   - Supports real-time data updates, which is useful for monitoring and dash boarding.

5. **Deployment**:
   - Applications can be easily shared and deployed via Streamlit sharing or other cloud platforms.
   - Streamlit apps can be run locally or hosted on the web.

6. **Customization and Theming**:
   - Users can customize the look and feel of their applications.
   - Offers a range of options for layout and design.

The Vehicle Detection and Counting System, powered by YOLOv5 and Deep SORT, represents a significant advancement in intelligent transportation systems. By combining YOLOv5’s high detection accuracy with Deep SORT’s robust tracking capabilities, the system enhances the accuracy, reliability, and real-time processing capabilities of vehicle tracking and counting systems

CHAPTER 3

PROBLEM ASSIGNED

3.1 Problem Overview

In today's bustling urban landscape, traffic congestion has become a ubiquitous challenge, stemming from a myriad of factors such as accidents, inefficient parking management, and the ever-expanding network of streets and highways. The escalating volume of vehicles navigating through cities worldwide exacerbates this issue, necessitating innovative solutions to alleviate congestion and enhance traffic management. Recognizing these challenges, our project focuses on leveraging advanced vehicle recognition and counting techniques to address the complexities of modern traffic environments.

3.2 Proposed Methodology

The cornerstone of our project lies in the development of a robust vehicle detection and counting system. Our proposed methodology entails the following steps:

1. **Vehicle Detection:** Employing state-of-the-art computer vision algorithms, we aim to accurately detect and identify vehicles within images or video streams. This initial step is crucial for subsequent counting and classification tasks.

2. **Counting Different Types of Vehicles:** Beyond mere detection, our system is designed to differentiate between various types of vehicles, such as cars, buses, and trucks. By meticulously tallying the occurrences of each vehicle type, we can glean valuable insights into traffic patterns and vehicular density.

3. **Data Storage and Analysis:** The data amassed through vehicle recognition and counting are systematically stored for further analysis and utilization. This repository of information serves as a valuable resource for traffic management authorities and urban planners, facilitating informed decision-making and policy formulation.

3.3 Motive of the Project

The objectives driving our project are multifaceted, encompassing various domains of traffic management and urban governance:

1. **Traffic Management:** By accurately counting and detecting vehicles, our system empowers traffic authorities to monitor and regulate the flow of vehicles effectively. This capability is particularly invaluable in congested urban areas or during peak traffic hours, where optimizing traffic flow is paramount.

2. **Accident Monitoring:** Real-time monitoring of traffic flow enables the timely detection of accidents or unusual activity on roads. Prompt identification of such incidents facilitates swift response measures, enhancing overall road safety and minimizing disruptions.

3. **Toll Collection Automation:** Integrating vehicle counting and detection capabilities into toll collection systems streamlines the process, reducing reliance on manual intervention and enhancing operational efficiency. Automated toll collection not only expedites traffic flow but also minimizes congestion at toll booths.

4. **Parking Management:** Our system extends its utility to parking management, enabling operators to monitor parking space occupancy and regulate the flow of vehicles in and out of parking lots. This proactive approach optimizes parking space utilization, mitigates congestion, and enhances the overall parking experience for motorists.

5. **Security and Surveillance:** Leveraging vehicle counting and detection technologies enhances security and surveillance efforts by enabling the monitoring of vehicular movement in sensitive areas or high-security zones. This capability aids in identifying suspicious vehicles, bolstering overall security measures.

This project endeavors to tackle the pervasive issue of traffic congestion through the innovative application of vehicle recognition and counting techniques. By harnessing the power of advanced computer vision algorithms, we aim to provide traffic management authorities with a comprehensive toolset for optimizing traffic flow, enhancing road safety, and improving overall urban mobility. With its diverse array of applications spanning from traffic management to security surveillance, our project holds the potential to significantly impact the way cities navigate the challenges of modern transportation.

CHAPTER 4

REQUIREMENT ANALYSIS

4.1 Introduction

Requirement analysis, often termed as feasibility study, serves as the cornerstone of software development projects. It involves a meticulous examination of the customer's needs and system specifications to ascertain the viability and scope of proposed software automation. This phase is pivotal in defining the parameters and functionalities of the envisaged system, laying the groundwork for subsequent development stages.

Understanding the Nature of Requirements

To comprehend the intricacies of the project's requirements, it is imperative for the development team to delve into the information domain and grasp the essential functions, behaviors, performance benchmarks, and interfacing intricacies. This holistic understanding forms the bedrock upon which the subsequent analysis and documentation are based.

4.2 Types of Requirements:

1. **Customer Requirements:**

   - These requirements delineate the expectations of the system from the customer's perspective. They encompass mission objectives, environmental constraints, and measures of effectiveness and suitability. Operational requirements, such as distribution/deployment, mission profiles, performance parameters, utilization environments, and effectiveness criteria, are meticulously delineated to address the core needs of stakeholders.

2. **Architectural Requirements:**

   Architectural requirements elucidate the necessary system architecture, outlining the structural framework within which the system functions. These requirements serve as a blueprint for system design and development, guiding the implementation of key architectural components.

3. **Structural Requirements:**

   Structural requirements define the necessary structure of the system, delineating the hierarchical organization of components and subsystems. These requirements elucidate the spatial and logical arrangement of system elements, ensuring coherence and scalability.

4. **Behavioural Requirements:**

   Behavioural requirements specify the expected behavior of the system, delineating the operational dynamics and interactions within the system. These requirements elucidate the system's response to external stimuli and internal processes, guiding the development of functional modules.

5. **Functional Requirements:**

   Functional requirements delineate the specific tasks, actions, or activities that the system must perform. They articulate the core functionalities and features of the system, providing a comprehensive overview of its operational capabilities.

6. **Non-functional Requirements:**

   Non-functional requirements specify criteria for judging the operation of the system, focusing on attributes such as performance, reliability, security, and scalability. These requirements complement functional specifications, ensuring that the system meets broader quality standards.

7. **Performance Requirements:**

Performance requirements quantify the extent to which system functions must be executed, encompassing parameters such as quantity, quality, timeliness, and readiness. These requirements are critical for assessing system efficiency and effectiveness, guiding performance optimization efforts.

8. **Design Requirements:**

Design requirements delineate the technical specifications and execution guidelines for system implementation. They specify the build, code, and procurement requirements for products, as well as the execution protocols for processes, ensuring adherence to technical standards and best practices.

4.3 Processing Requirements:

In this phase, the processing capabilities of the system are analyzed to determine the hardware and software specifications required for system development. The specification is categorized into hardware and software requirements, ensuring compatibility and performance optimization.

1. Hardware Requirements:

Table no. 4.1

1	System Type	64-bit operating system
2	RAM	8GB
3	SSD	500GB
4	Internet Connection	512Kbps
5	Processor	Inteli-5 8thGen

2. Software Requirements:

Table no. 4.2

1	Operating System	Windows 8 or 10
2	BackEnd	Artificial Intelligence , Machine Learning and Deep Learning
3	Web Server	Localhost
4	Designing Software	Python IDLE(Anaconda or Pycharm Community)

CHAPTER 5

SYSTEM ANALYSIS

5.1 Introduction

System analysis constitutes a crucial phase in the software development lifecycle, encompassing a comprehensive study of business processes and procedures to evaluate their efficacy and identify areas for improvement. This chapter delves into the investigation phase, feasibility study, economic analysis, technical analysis, and operational analysis, elucidating the key considerations and methodologies involved in each stage of system analysis.

5.2 Investigation Phase

The investigation phase, also known as the fact-finding stage, entails a detailed study of the current system to understand its functioning and identify information requirements. Various techniques, such as interviews and documentation review, are employed to gather pertinent data. Thorough investigation ensures a holistic understanding of the existing system, laying the foundation for subsequent analysis and decision-making.

5.3 Feasibility Study

Feasibility study is the process of determining whether a proposed project is viable and worthwhile. This entails assessing economic, technical, and operational feasibility to ascertain the project's potential success. Economic analysis involves cost-benefit analysis to weigh development costs against anticipated benefits. Technical analysis examines the technical merits and requirements of the proposed system, including performance, reliability, and maintainability. Operational analysis evaluates the system's operational aspects to ensure usability and user acceptance.

5.4 Economic Analysis

Cost-benefit analysis is a pivotal component of the feasibility study, delineating the costs and benefits associated with the proposed system. This analysis considers both tangible and intangible benefits, such as improved design quality and customer satisfaction. Economic feasibility is contingent upon the alignment of project costs with anticipated returns on investment, necessitating careful consideration of project scope and strategic objectives.

5.5 Technical Analysis

Technical analysis focuses on evaluating the technical merits and requirements of the proposed system. This includes assessing performance, reliability, maintainability, and development risks. Key considerations include the technologies and tools required for system implementation, as well as the feasibility of achieving desired system functionality. Technical soundness is paramount to ensuring the successful development and deployment of the system.

5.6 Operational Analysis

Operational analysis examines the operational aspects of the proposed system to assess its feasibility and usability. The system must be convenient and user-friendly, addressing the needs of stakeholders effectively. In the context of vehicle classification and counting, operational feasibility entails developing a system that streamlines processes and reduces the burden on users. User acceptance is crucial for the success of the system, and operational analysis ensures that the system meets user expectations and requirements.

System analysis is a critical phase in software development, providing insights into the viability, feasibility, and effectiveness of proposed projects. Through thorough investigation and analysis, stakeholders can make informed decisions and mitigate risks associated with system development. By considering economic, technical, and operational factors, system analysts can ensure the successful implementation and adoption of software systems, ultimately contributing to organizational efficiency and effectiveness.

CHAPTER 6

PLANNING

6.1 Introduction

Planning is a pivotal phase in the software development process, setting the foundation for subsequent stages. It involves strategizing the approach, determining tasks, scheduling, and resource allocation to ensure the effective completion of the project.

6.2 Planning Objectives

The primary objective of planning is to identify and schedule tasks necessary for project completion. A robust plan should be flexible enough to accommodate unforeseen events while considering economic, political, and human factors. While a detailed requirements document is not mandatory for planning, awareness of crucial requirements is essential.

6.3 Major Issues Addressed in Project Plan:-

1. **Cost Estimation:** Estimating the financial resources required for project execution.
2. **Schedule and Milestones:** Establishing a timeline with key milestones to track progress.
3. **Personnel Plan:** Allocating human resources effectively based on skill sets and availability.
4. **Software Quality Assurance Plans:** Outlining strategies to ensure software quality throughout the project.
5. **Configuration Management Plans:** Defining procedures for managing changes to the software configuration.
6. **Project Monitoring Plans:** Establishing mechanisms for monitoring project progress and addressing deviations.
7. **Risk Management:** Identifying potential risks and developing strategies to mitigate them.

6.4 Planning Activities

Planning activities include establishing the need for the system, conducting sensitivity assessments, performing initial risk assessments, and reviewing solicitation documents. These activities ensure that the project plan addresses all critical aspects and aligns with project objectives.

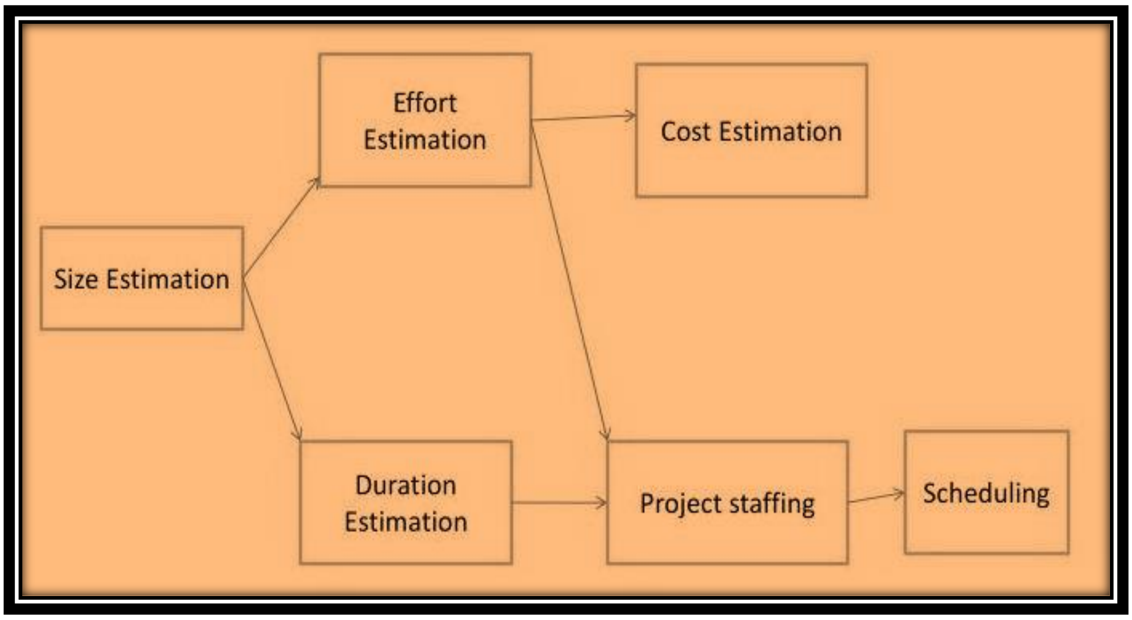


Fig.6.1–Planning activities

6.5 Waterfall Model

The Waterfall Model outlines the sequential execution of project phases:

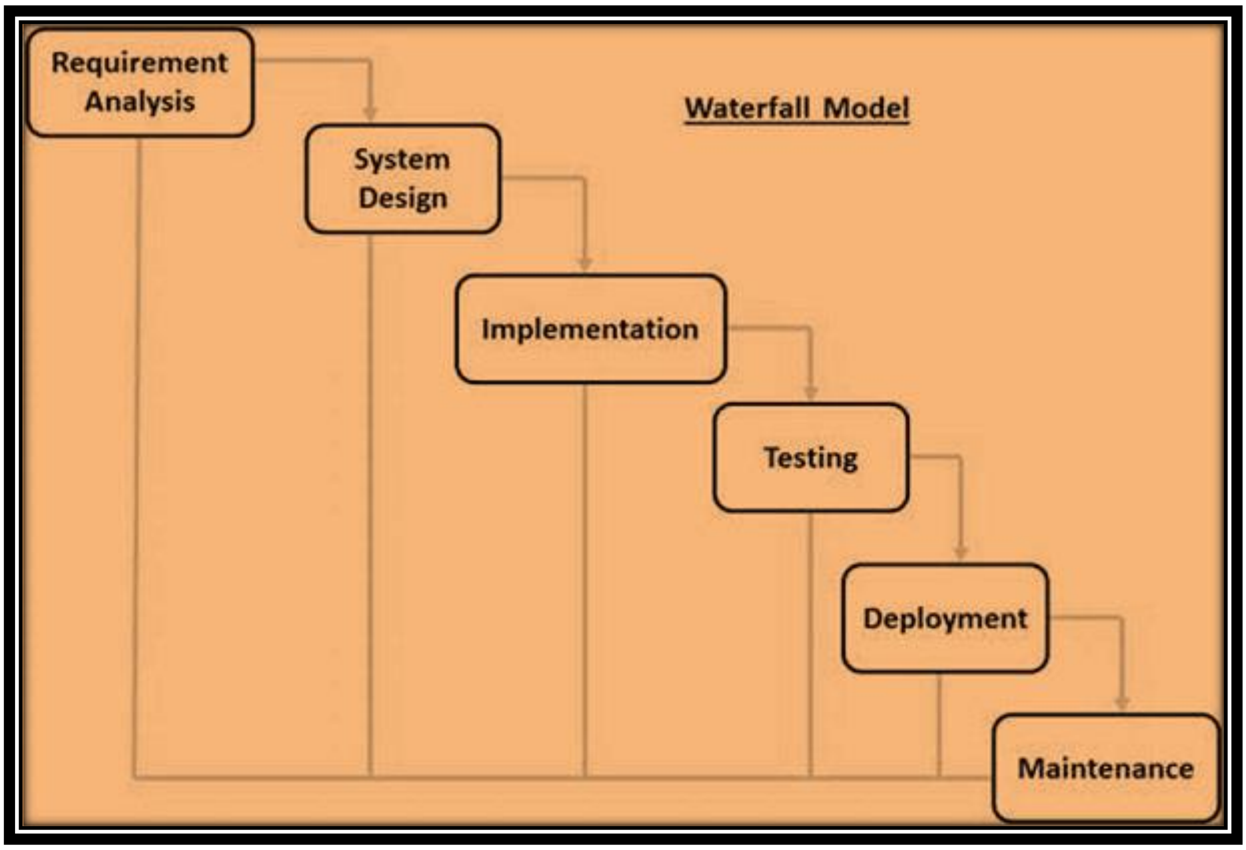


Fig.6.2–Waterfall model

- Gathering and analysis of needs – During this stage, all potential system requirementsaregatheredandrecordedinarequirementspecificationdocument.
- System Design – In this step, the required specifications from the first phase are examined, and a system design is created. This system design determining the overall system architecture as well as the hardware and system requirements.
- ImplementationThesystemisinitiallydevelopedasdiscreteprogrammesknown as units, which are then combined in the next phase, with input from the system design. Unit testing is the process of developing and evaluating each unit for functionality.

- IntegrationandTesting−Alltheunitsdevelopedintheimplementationphaseare integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

- Deployment of system − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.

- Maintenance − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

6.7 Functional Specifications

- **Accessibility:** Users should be able to access the vehicle counting and classification system without login credentials.

- **File Upload:** A functional file upload system should allow users to upload files from a repository.

- **Automation:** Upon file upload, the system should initiate vehicle counting and classification processes automatically.

Planning serves as the roadmap for successful project execution, ensuring that all aspects of the project are meticulously addressed and executed in a timely manner. It sets the stage for subsequent phases, guiding the development team towards project completion**.**

CHAPTER 7

SOFTWARE DESIGN

7.1 INTRODUCTION

In the realm of software development, the phase of design holds pivotal significance as it marks the foundational stage where methodologies and principles are meticulously crafted to articulate a system with the precision required for its eventual implementation. Software design encompasses a series of technical activities, each serving a distinct purpose in shaping the final product. System design involves conceptualizing the overall structure and functionality of the software, outlining how different components will interact and function together to achieve the desired outcomes. Architectural design delves deeper into defining the architecture of the software, specifying the high-level structures and patterns that will govern its organization and operation. Detailed design zooms in further to specify the intricacies of individual modules or components, detailing their interfaces, algorithms, and data structures. Together, these design activities not only lay the blueprint for the software but also establish the framework for rigorous verification and validation processes to ensure that the final product meets its intended specifications and performance expectations. Thus, the design phase serves as the bedrock upon which the entire software development lifecycle unfolds, guiding subsequent development, testing, and deployment efforts towards the realization of a robust and functional software solution.

7.2 System Design

System design, also known as top-level design, focuses on determining the necessary modules for the system, specifying their functionalities, and defining their interconnections. This level of design encompasses architectural design and detailed design.

7.3 Architectural Design

In architectural design, analysts organize and convert unstructured data collected through various means such as interviews, questionnaires, and observations into structured representations. Techniques such as system flowcharts, data flow diagrams, and decision tables are employed to illustrate how data are used or changed within the system. This level of design emphasizes functions over physical implementation and is a creative process aimed at providing solutions.

1. **External Design**: This involves planning and specifying the externally observable characteristics of the software product, including user interfaces, report formats, and performance requirements. External design begins during the analysis phase and extends into the design phase.

2. **Physical Design**: Physical design pertains to the actual input and output processes of the system, detailing how data is input, processed, and output. It does not refer to the tangible hardware design but focuses on the system's data flow and processing mechanisms.

3. **Logical Design**: The logical design represents an abstract representation of data flows, inputs, and outputs of the system. This is often achieved through modeling techniques such as data flow diagrams, entity-relationship diagrams, and use case diagrams.

    - **Data Flow Diagrams (DFD)**: Provide a graphical representation of the flow of data within the system, illustrating how data moves between processes and entities.

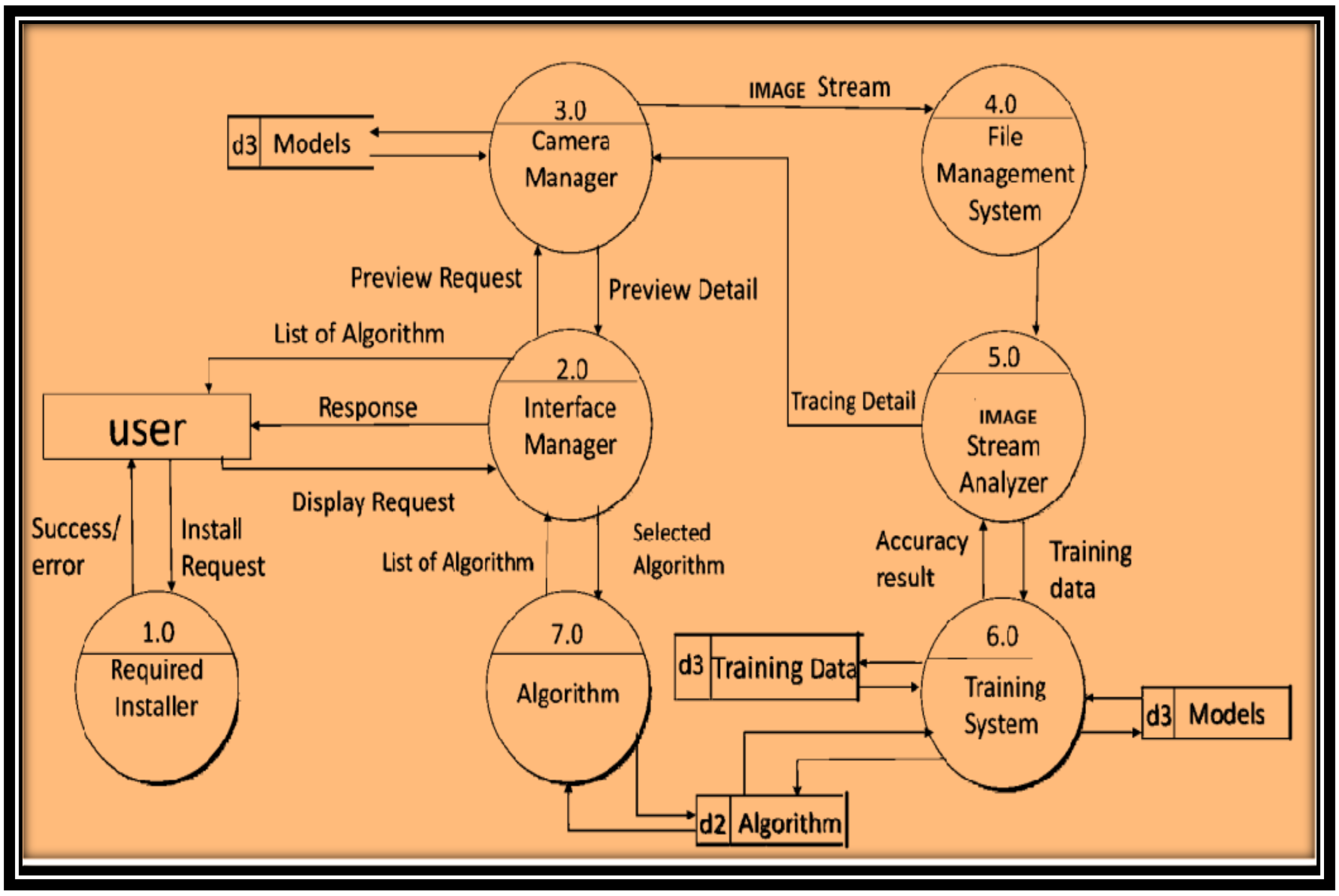


Fig.7.1 – 1 level DFD

- **Entity-Relationship Diagrams (ERD)**: Depict the relationships between different entities within the system, facilitating a clear understanding of data structure and dependencies.

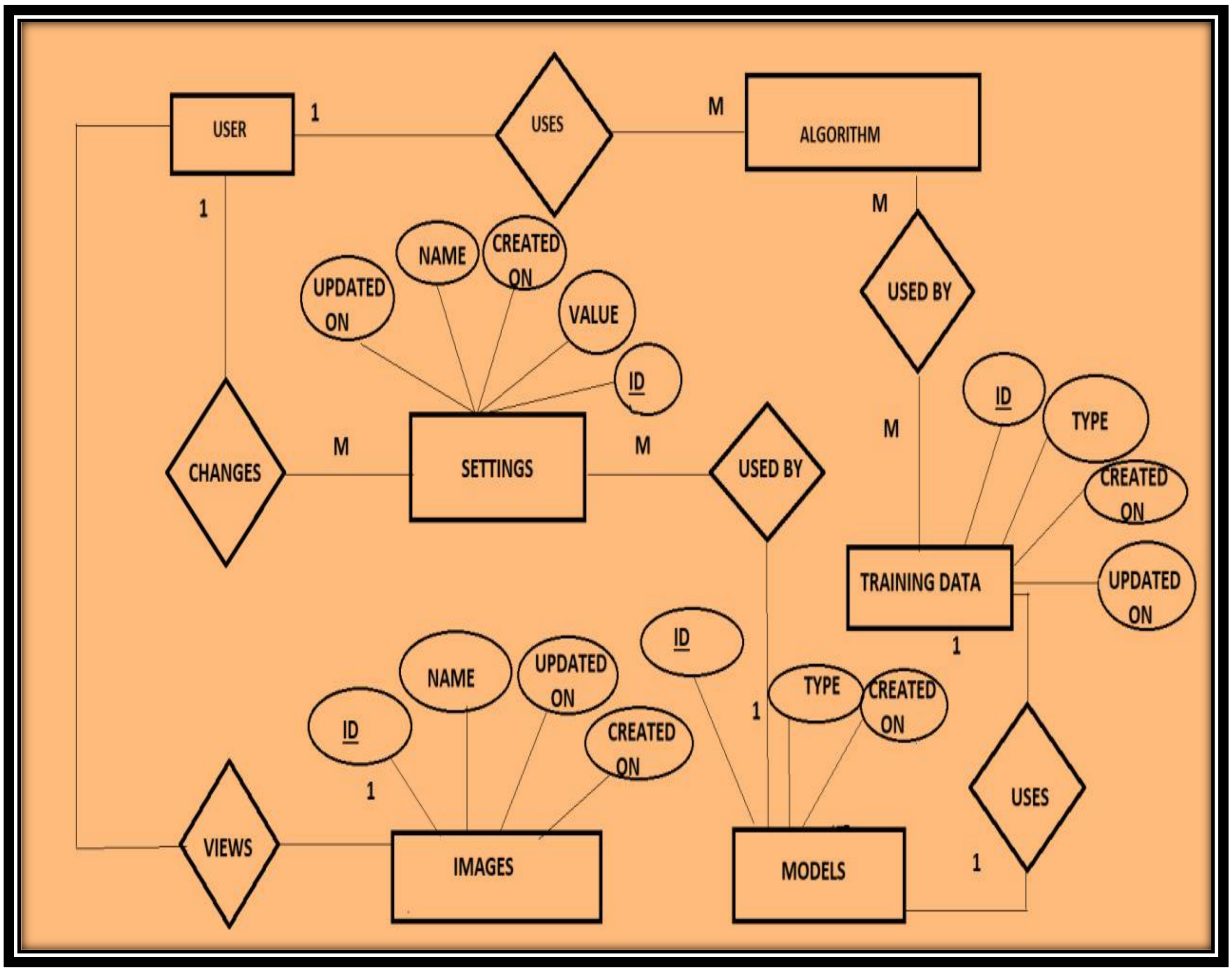


Fig.7.2 –ERD

- **Use Case Diagrams**: Model interactions between external actors and the system to achieve specific goals, capturing functional requirements and user interaction

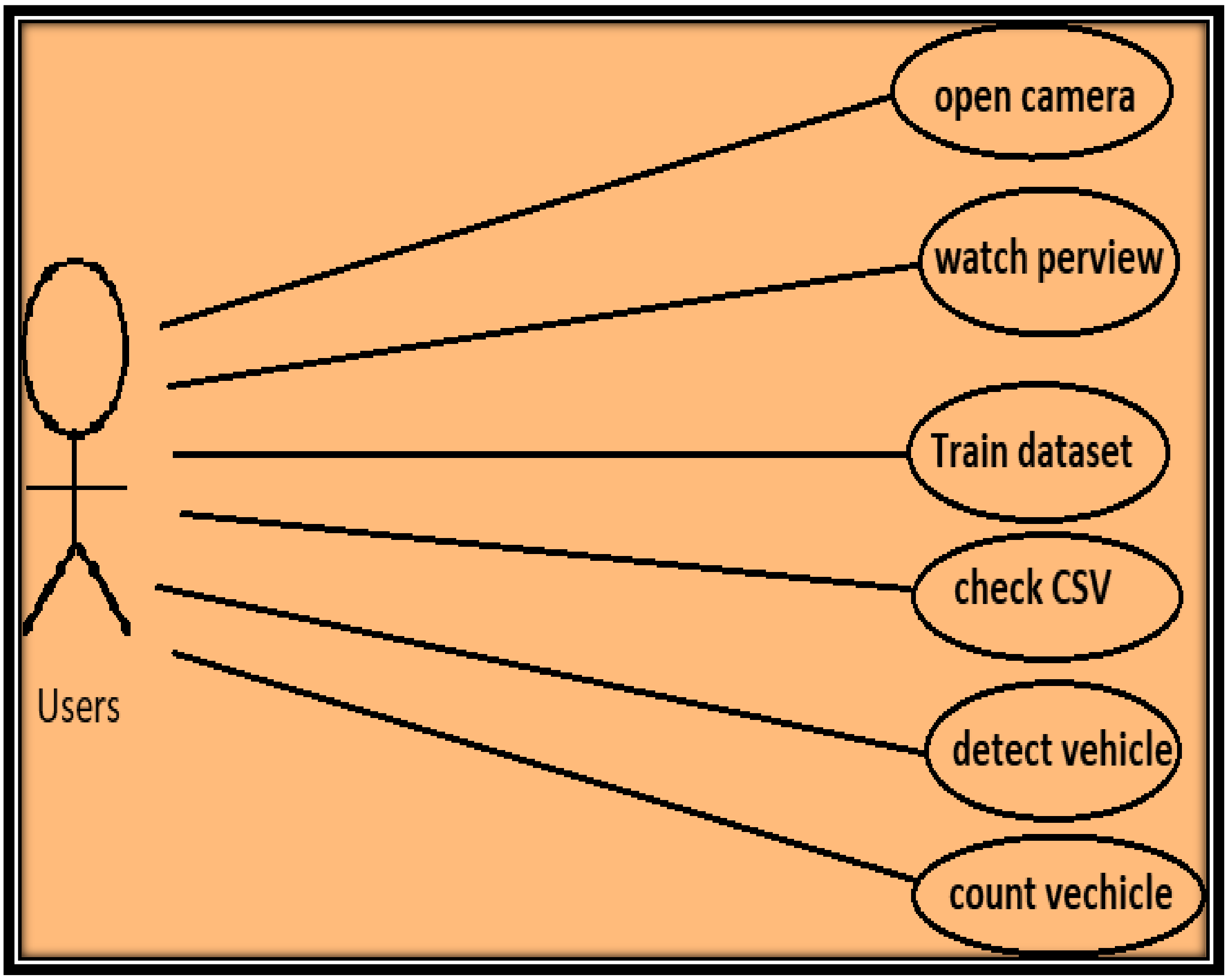


Fig 7.3 – Use case diagram

Flowchart:

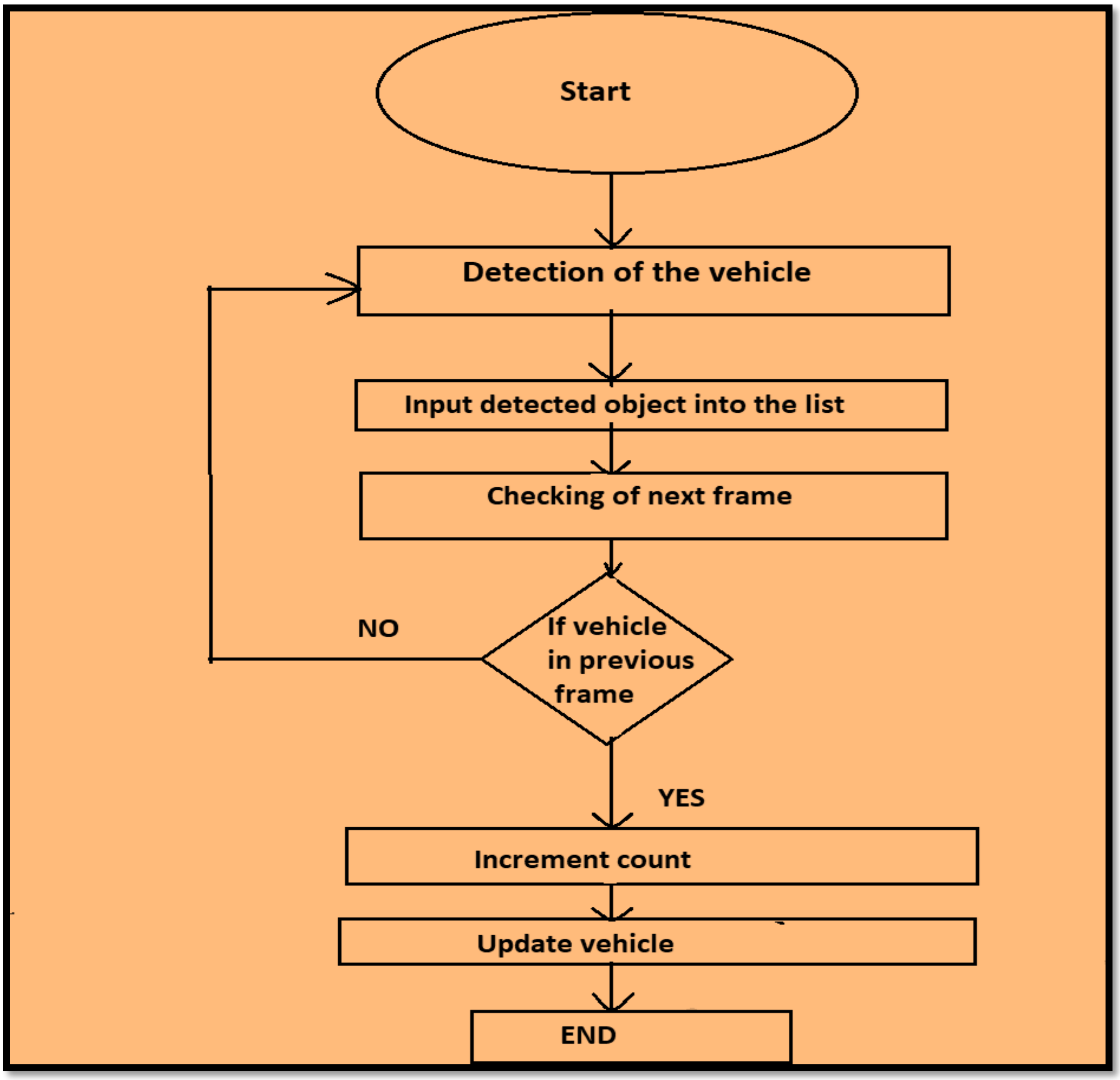


Fig 7.4–Flowchart

CHAPTER 8

TESTING

8.1 Introduction

Testing is a critical phase in the software development lifecycle where the program is evaluated against a set of test cases to determine if it behaves as expected. Dynamic testing, which involves executing the program with test cases, helps identify errors in the program's behavior. However, it does not determine the exact nature of errors. Testing forms the primary method of error detection in software.

The system is tested using testing firstly then all modules are integrated and again the system is tested using integrated testing and it was found that system is working according to its expectation.

8.2 Testing Process

1. **Individual Program Testing**: Initially, programs are tested individually to check for syntax and logical errors. Corrections are made, and tests are conducted to ensure the program functions as intended.
2. **System Testing**: Once individual programs are tested, the system as a whole is tested experimentally to verify that it adheres to its specifications. This involves executing the integrated system to identify any errors.
3. **Integrated Testing**: Modules are integrated, and the system is tested again to ensure it functions as expected in an integrated environment.

8.3 Testing Methods and Techniques

1. **Unit Testing**: Focuses on verifying the smallest unit of software, typically modules. Testing is conducted to uncover errors within the module's boundaries. All modules must pass unit tests before integration testing begins.

2. **Integration Testing**: This phase tests the integration of modules, focusing on module interactions and ensuring proper integration. It verifies that modules can communicate and function together as intended.

3. **System Testing**: The entire software system is tested against its requirements document to verify if it meets specified requirements. The entire system, referred to as 'HRRP,' is tested comprehensively.

4. **Acceptance Testing**: Performed with realistic client data to demonstrate the software's satisfactory operation. This testing focuses on the external behavior of the system rather than its internal logic.

5. **Black Box Testing**: Also known as functional testing, black-box testing validates the operational functionality of the software while maintaining the integrity of external information. It uncovers errors such as incorrect functions, interface issues, and data structure errors.

6. **White Box Testing**: White-box testing, or glass-box testing, utilizes the control structure of the procedural design. It ensures that all logical decisions, boundary conditions, and data structures are exercised during testing.

8.4 Merits of White-Box Testing:

- Implementation of all non-dependent paths at least once.
- Exercise of all logical decisions.
- Execution of all boundary conditions and loops.
- Maintenance of the validity of the internal data structure.

Testing is a crucial aspect of software development, ensuring that the software functions as intended and meets the specified requirements. Through various testing methods and techniques, errors are identified and rectified, ultimately contributing to the delivery of a reliable and high-quality software product.

TEST CASES AND TEST RESULT

Table no. 8.1

Test Subject	Test Method	Expected Result	Actual Result	Remarks
Complete testing of system on different browsers	The whole project is executed using different browsers like opera, Mozilla Firefox, and internet explorer.	The project should give the same output on all the Browsers.	Because High HD picture Quality, the size of the Website Become Large & on executing, it takes time to view.	Browser can sometime affect the working and performance of execution of the project.

CHAPTER 9

MAINTENANCE

9.1 Introduction

Software maintenance is the ongoing process of modifying a software product after it has been delivered to fix bugs, enhance performance, or adapt the product to new environments. This phase is critical as it ensures the software remains functional, secure, and efficient over time.

9.2 Maintenance Procedures

The international standard defines six key software maintenance procedures:

1. **Preparation and Transition Procedures**:
    - Developing a maintenance plan to guide future maintenance activities.
    - Preparing to address issues that surfaced during development.
    - Monitoring product configuration management to track and control changes.

2. **Problem and Modification Analysis**:
    - Once the maintenance team takes control of the application, each request for modification or issue resolution must be examined and verified.
    - The scenario is simulated to ensure the problem is valid.
    - Investigations are conducted, and solutions are proposed.
    - The maintenance programmer documents the request and the proposed solution, obtaining all necessary approvals before implementing adjustments.

3. **Modification Implementation Consideration**:
    - This procedure involves planning and executing the changes needed to address issues or enhance functionality.
    - It includes coding, testing, and deploying the modifications.

4. **Adjustment Acceptance**:
    - After implementing adjustments, it is essential to verify that the modifications solved the reported problems.
    - This involves communicating with the person who made the request to ensure their concerns are addressed.

5. **Migration Procedure**:
    - Unique maintenance activities such as platform migration, which involves moving the software to a new environment without changing its functionality.
    - This is not a routine maintenance task and usually requires a dedicated maintenance project team.

6. **Software Retirement**:
    - The final maintenance procedure involves retiring a piece of software.
    - This is a rare occurrence but necessary when the software is no longer useful or supported.

Maintenance-Specific Procedures and Practices:

Certain procedures, pursuits, and customs are exclusive to maintainers:

1. **Transition**:
    - A planned and coordinated series of actions take place while a system is gradually passed from the developer to the maintenance team.

2. **Modification Request Acceptance/Rejection**:
    - Maintainers may reject modification requests that exceed a specified size, effort, or complexity, redirecting them back to the developers for handling.

3. **Misconceptions About Maintenance:**
    A common misconception about software maintenance is that it solely involves bug fixes. However, research and surveys, such as those conducted by Pigosky in 1997, indicate that over 80% of the maintenance effort is used on non-corrective measures. These include enhancing system functionality and performance improvements. Users often report problems that are actually requests for functionality enhancements, contributing to this misconception.

4. **Software Maintenance and System Evolution:**
    Meir M. Lehman first addressed software maintenance and system evolution in 1969. His extensive research over 20 years led to the development of the Eight Laws of Evolution (Lehman, 1997). These laws emphasize the continuous growth and evolution of software systems, highlighting the need for ongoing maintenance to manage increasing complexity.

The Eight Laws of Software Evolution:

1. **Continuing Change**:
    - Software must continually adapt to meet the changing needs of its users and environment.
    - This requires regular updates and enhancements.

2. **Increasing Complexity**:
    - As software evolves, its complexity increases unless efforts are made to reduce it.
    - Techniques such as code restructuring and refactoring are essential to manage complexity.

3. **Self-Regulation**:
    - Software evolution is a self-regulating process.
    - This means it follows patterns and behaviors that can be observed and predicted.

4. **Conservation of Organizational Stability (invariant work rate)**:
    - The rate of work and productivity in software maintenance tends to remain stable over time.
    - This indicates that maintenance efforts are consistently required throughout the software's lifecycle.

5. **Conservation of Familiarity**:
    - As software evolves, all stakeholders must maintain a consistent understanding and familiarity with the system.
    - This requires documentation, training, and effective communication.

6. **Continuing Growth**:
    - The functionality of a software system must continually grow to satisfy user needs.
    - This involves adding new features and capabilities.

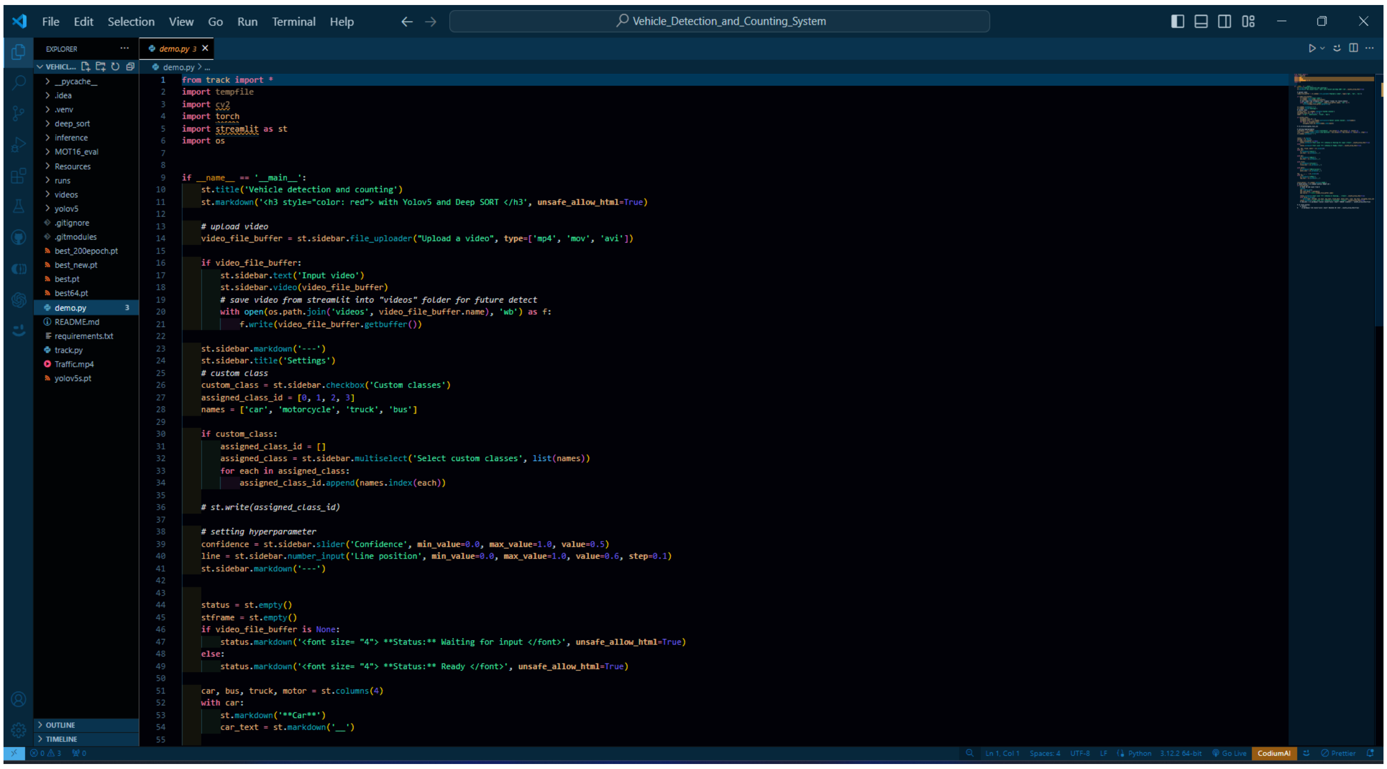
7. **Declining Quality**:
    - Unless rigorously maintained and adapted, the quality of software will decline over time.
    - This necessitates proactive maintenance and quality assurance practices.

8. **Feedback System**:
    - Software evolution is driven by feedback from users, developers, and the environment.
    - This feedback loop is critical for identifying areas for improvement and adaptation

Software maintenance is an essential aspect of the software lifecycle, ensuring that software products continue to meet user needs, perform efficiently, and adapt to changing environments. It encompasses a range of activities, from bug fixes to performance enhancements and migrations. Understanding and applying the principles of software maintenance and evolution, as articulated by Lehman's laws, can help manage the complexity and ensure the longevity of software systems. Effective maintenance requires meticulous planning, continuous monitoring, and a proactive approach to managing changes and improvements.

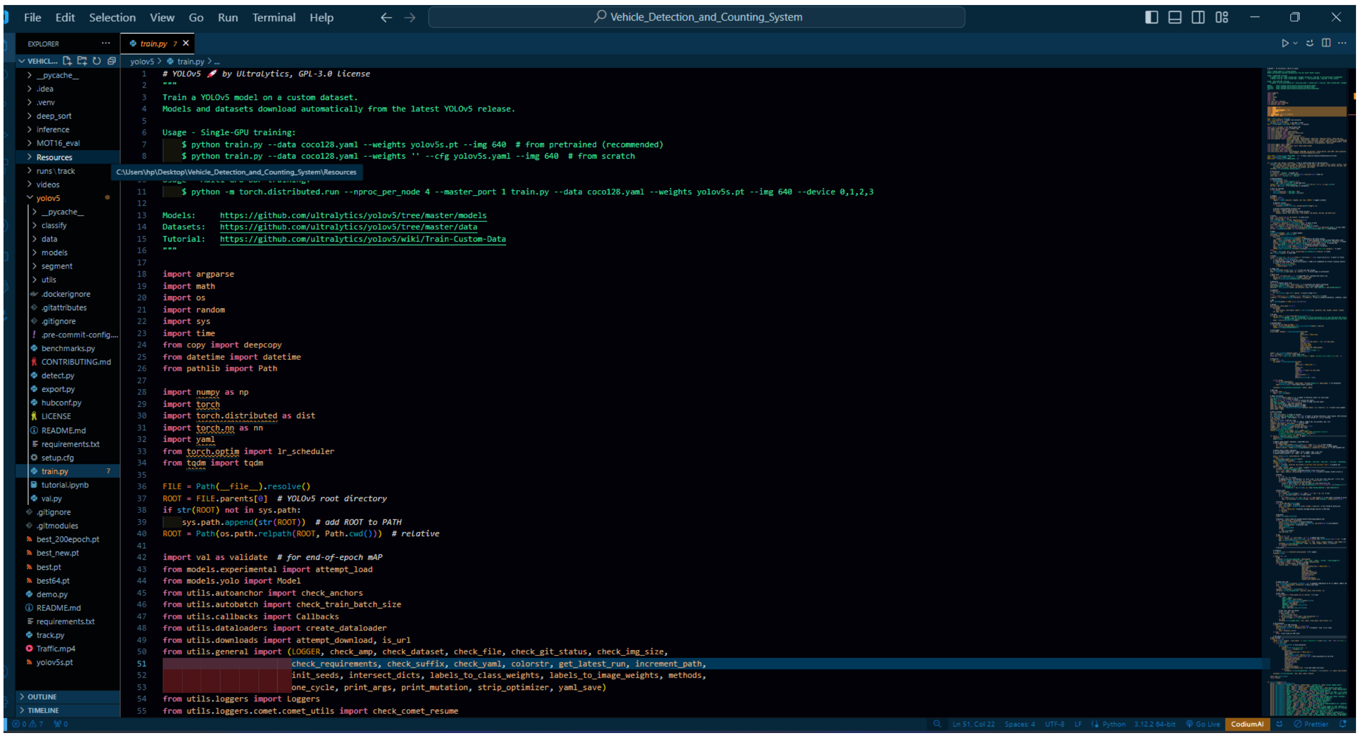
CHAPTER 10

SCREENSHOTS



```
from track import *
import tempfile
import cv2
import torch
import streamlit as st
import os


if __name__ == '__main__':
    st.title('Vehicle detection and counting')
    st.markdown('<h3 style="color: red"> with Yolov5 and Deep SORT </h3', unsafe_allow_html=True)

    # upload video
    video_file_buffer = st.sidebar.file_uploader("Upload a video", type=['mp4', 'mov', 'avi'])

    if video_file_buffer:
        st.sidebar.text('Input video')
        st.sidebar.video(video_file_buffer)
        # save video from streamlit into "videos" folder for future detect
        with open(os.path.join('videos', video_file_buffer.name), 'wb') as f:
            f.write(video_file_buffer.getbuffer())

    st.sidebar.markdown('---')
    st.sidebar.title('Settings')
    # custom class
    custom_class = st.sidebar.checkbox('Custom classes')
    assigned_class_id = [0, 1, 2, 3]
    names = ['car', 'motorcycle', 'truck', 'bus']

    if custom_class:
        assigned_class_id = []
        assigned_class = st.sidebar.multiselect('Select custom classes', list(names))
        for each in assigned_class:
            assigned_class_id.append(names.index(each))

    # st.write(assigned_class_id)

    # setting hyperparameter
    confidence = st.sidebar.slider('Confidence', min_value=0.0, max_value=1.0, value=0.5)
    line = st.sidebar.number_input('Line position', min_value=0.0, max_value=1.0, value=0.6, step=0.1)
    st.sidebar.markdown('---')


    status = st.empty()
    stframe = st.empty()
    if video_file_buffer is None:
        status.markdown('<font size= "4"> **Status:** Waiting for input </font>', unsafe_allow_html=True)
    else:
        status.markdown('<font size= "4"> **Status:** Ready </font>', unsafe_allow_html=True)

    car, bus, truck, motor = st.columns(4)
    with car:
        st.markdown('**Car**')
        car_text = st.markdown('__')

```

Fig 10.1–Coding Screenshot



```
# YOLOv5 🚀 by Ultralytics, GPL-3.0 license
"""
Train a YOLOv5 model on a custom dataset.
Models and datasets download automatically from the latest YOLOv5 release.

Usage - Single-GPU training:
    $ python train.py --data coco128.yaml --weights yolov5s.pt --img 640  # from pretrained (recommended)
    $ python train.py --data coco128.yaml --weights '' --cfg yolov5s.yaml --img 640  # from scratch

Usage - Multi-GPU DDP training:
    $ python -m torch.distributed.run --nproc_per_node 4 --master_port 1 train.py --data coco128.yaml --weights yolov5s.pt --img 640 --device 0,1,2,3

Models:     https://github.com/ultralytics/yolov5/tree/master/models
Datasets:   https://github.com/ultralytics/yolov5/tree/master/data
Tutorial:   https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data
"""

import argparse
import math
import os
import random
import sys
import time
from copy import deepcopy
from datetime import datetime
from pathlib import Path

import numpy as np
import torch
import torch.distributed as dist
import torch.nn as nn
import yaml
from torch.optim import lr_scheduler
from tqdm import tqdm

FILE = Path(__file__).resolve()
ROOT = FILE.parents[0]  # YOLOv5 root directory
if str(ROOT) not in sys.path:
    sys.path.append(str(ROOT))  # add ROOT to PATH
ROOT = Path(os.path.relpath(ROOT, Path.cwd()))  # relative

import val as validate  # for end-of-epoch mAP
from models.experimental import attempt_load
from models.yolo import Model
from utils.autoanchor import check_anchors
from utils.autobatch import check_train_batch_size
from utils.callbacks import Callbacks
from utils.dataloaders import create_dataloader
from utils.downloads import attempt_download, is_url
from utils.general import (LOGGER, check_amp, check_dataset, check_file, check_git_status, check_img_size,
                           check_requirements, check_suffix, check_yaml, colorstr, get_latest_run, increment_path,
                           init_seeds, intersect_dicts, labels_to_class_weights, labels_to_image_weights, methods,
                           one_cycle, print_args, print_mutation, strip_optimizer, yaml_save)
from utils.loggers import Loggers
from utils.loggers.comet.comet_utils import check_comet_resume
```

Fig 10.2- Coding Screenshot

Working Flow Of The Model

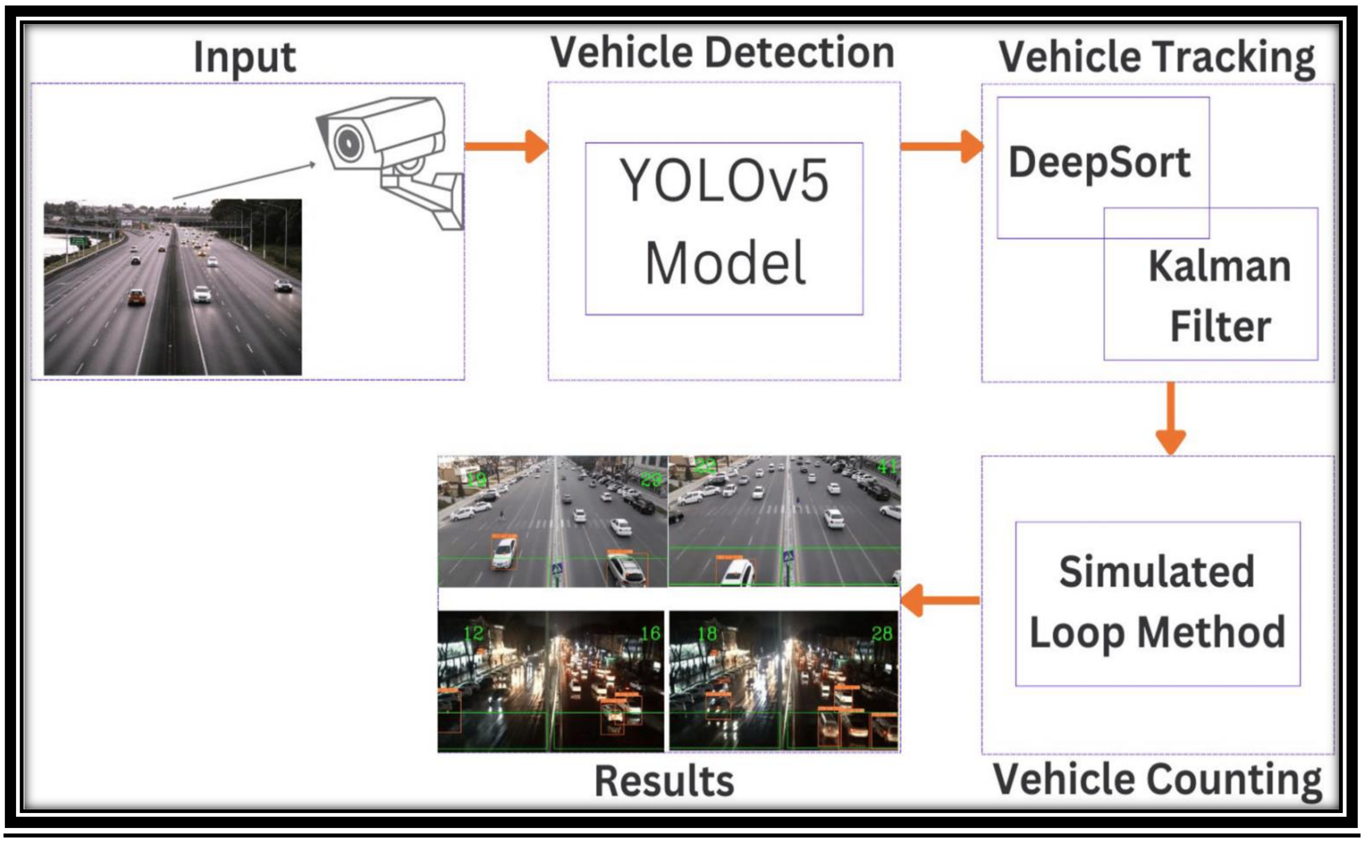


Fig 10.3 Working Flow of the Model

Bounding the objects by boxes (i.e Detecting the object)

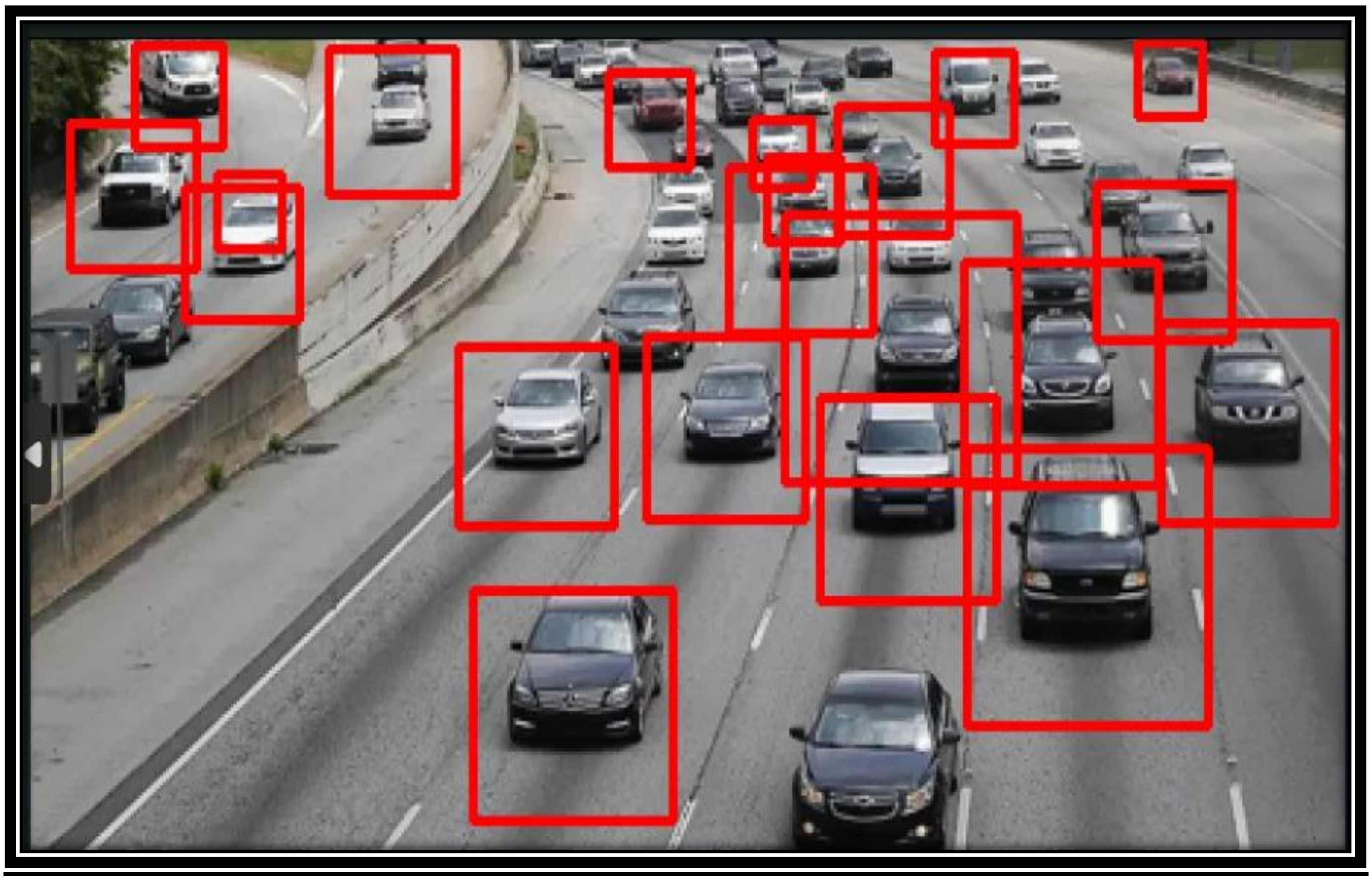


Fig 10.4 Bounding the objects by boxes

Marking the object with confidence

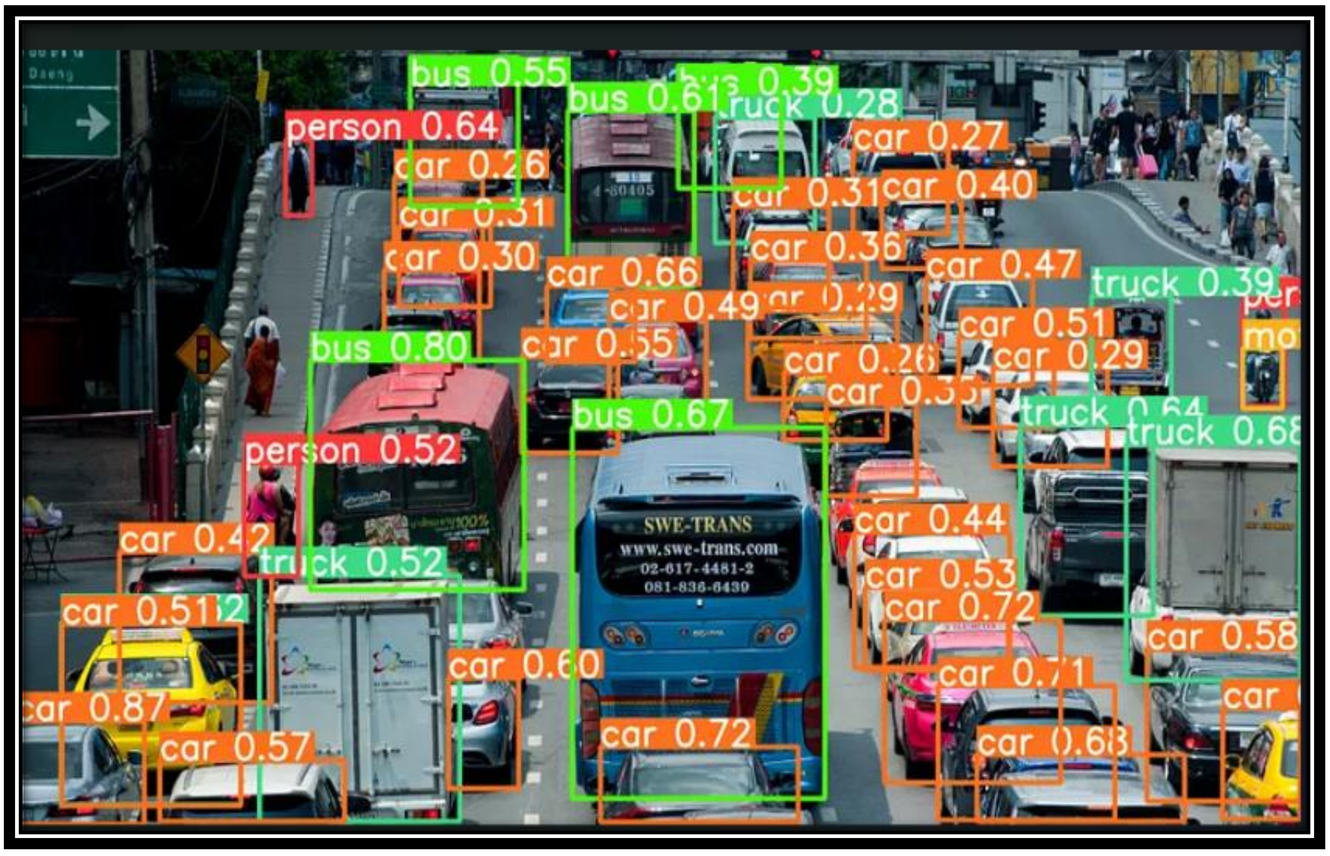
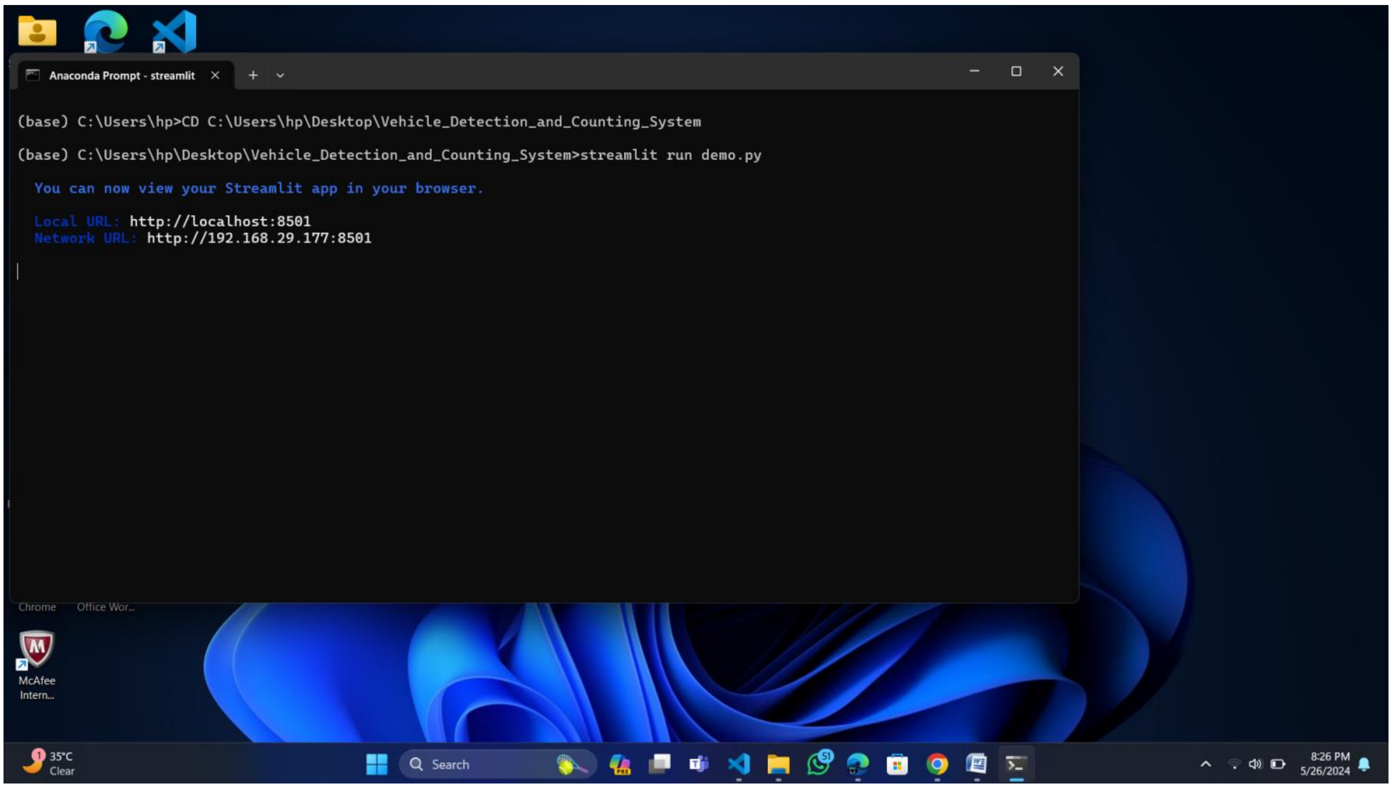


Fig 10.5 Marking the object with confidence

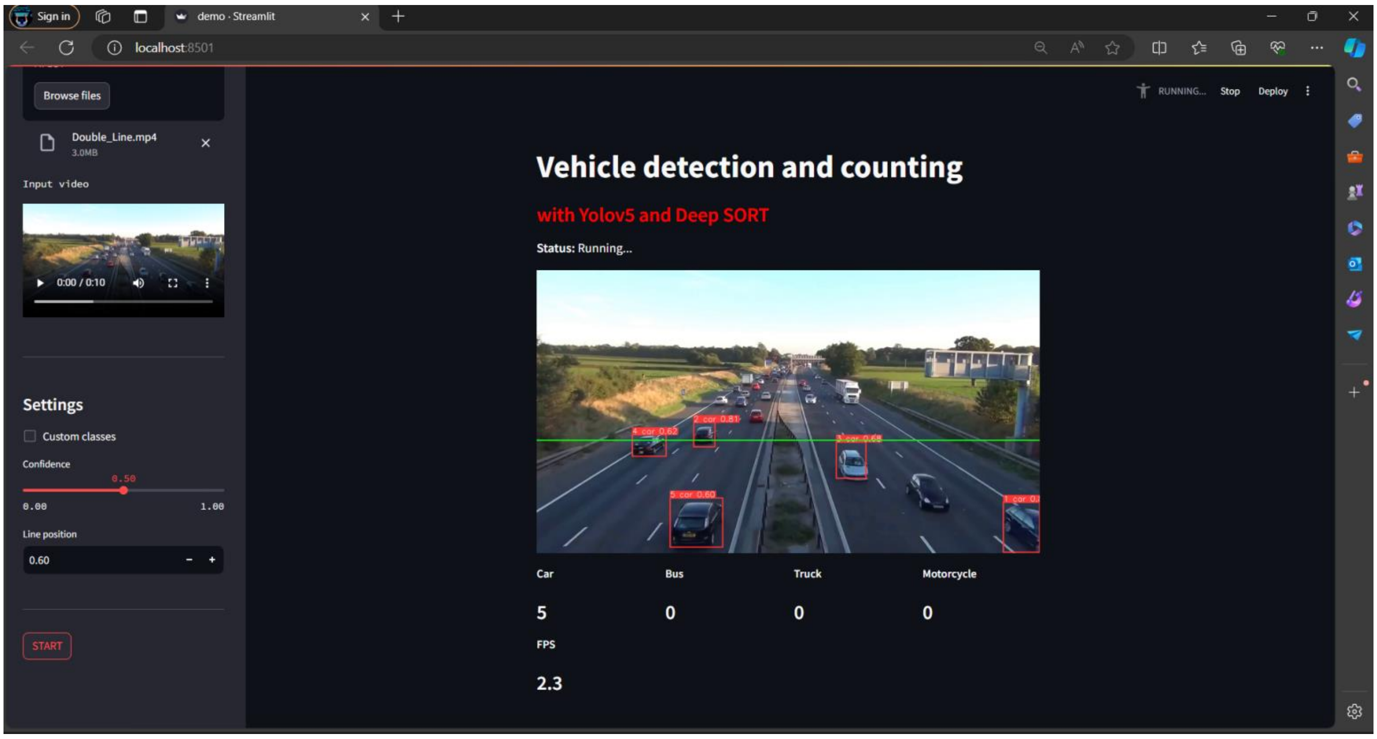
Opening Anaconda prompt



```
(base) C:\Users\hp>CD C:\Users\hp\Desktop\Vehicle_Detection_and_Counting_System

(base) C:\Users\hp\Desktop\Vehicle_Detection_and_Counting_System>streamlit run demo.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.29.177:8501
```

Fig 10.6–Anaconda prompt

Main Window Web Page



Vehicle detection and counting

with Yolov5 and Deep SORT

Status: Running...

Settings

Custom classes

Confidence 0.50 (0.00 – 1.00)

Line position 0.60

Car	Bus	Truck	Motorcycle
5	0	0	0

FPS

2.3

Fig 10.7 Main Window Web Page

<u>Browsing videos from files</u>

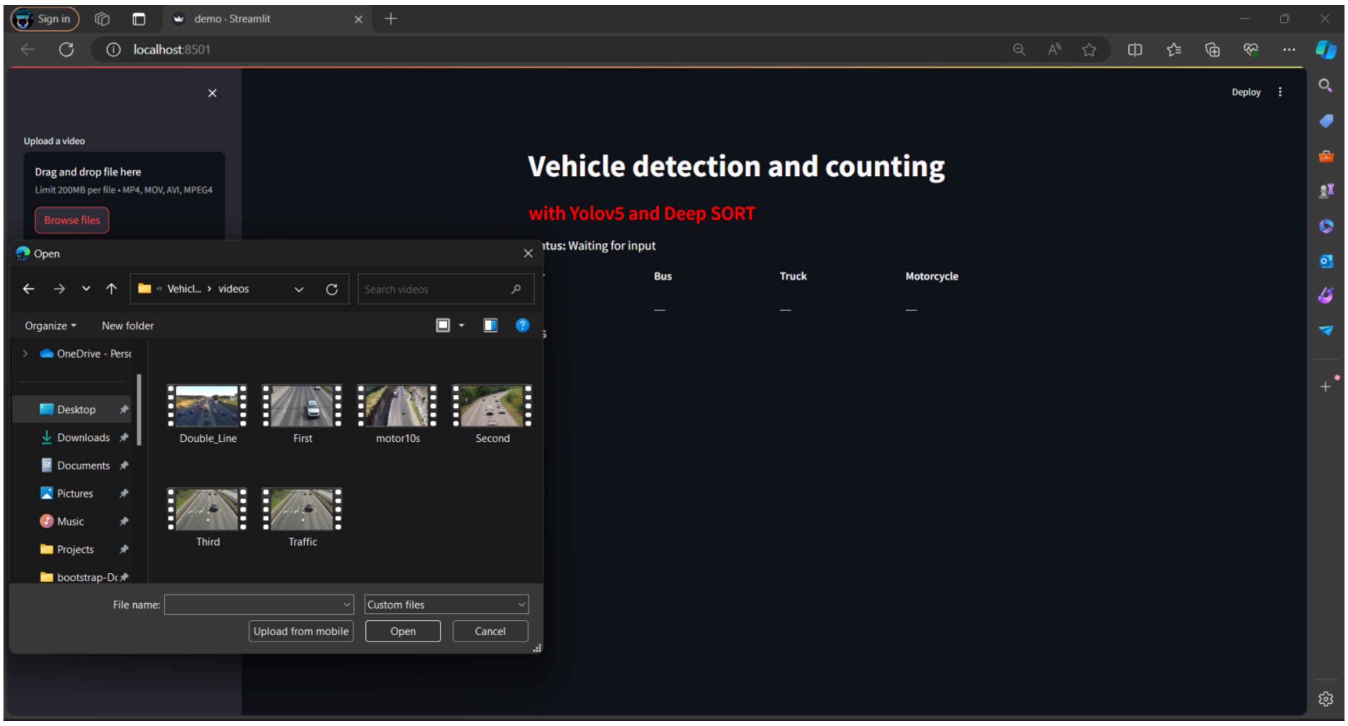
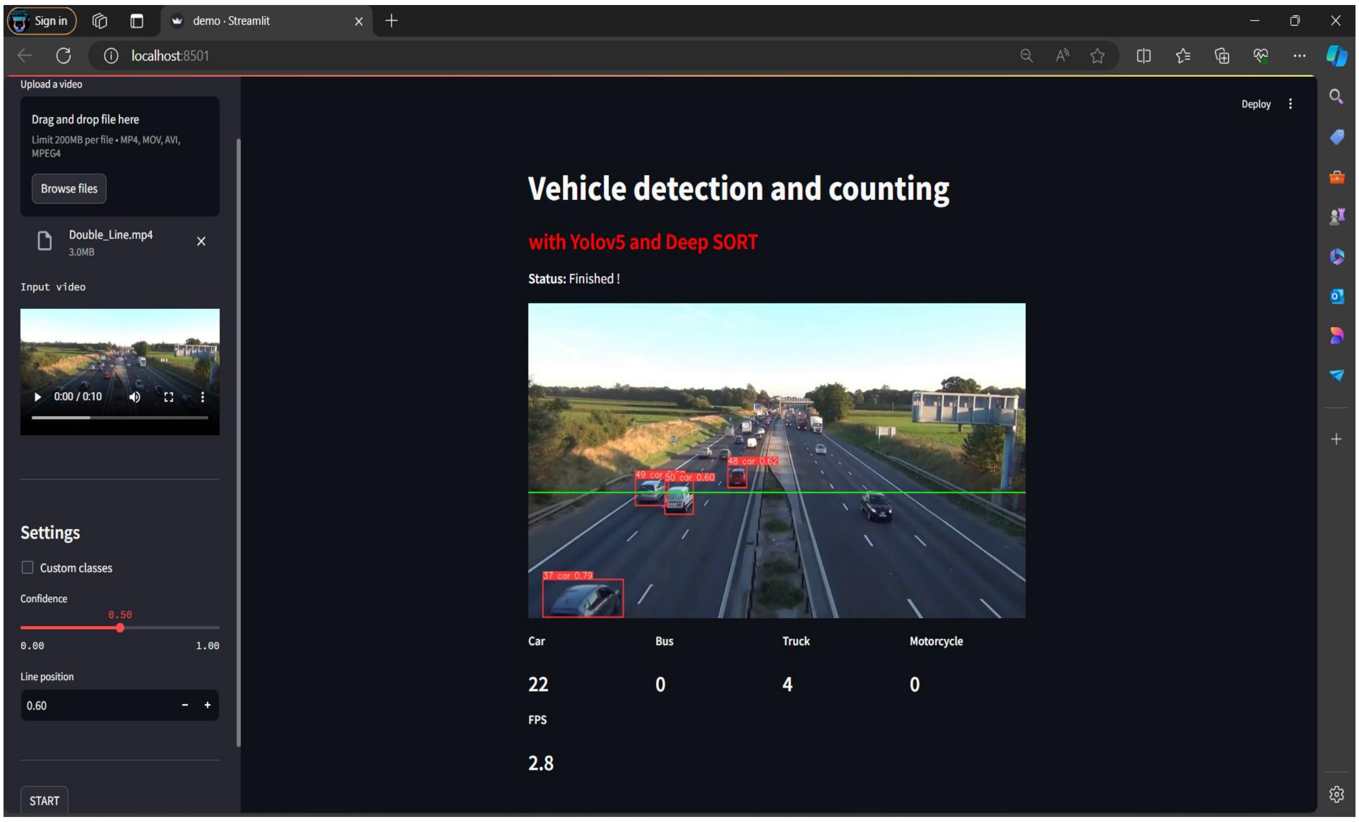


Fig 10.8 Browsing videos from files

Final Output



Upload a video

Drag and drop file here
Limit 200MB per file • MP4, MOV, AVI, MPEG4

Double_Line.mp4
3.0MB

Input video

0:00 / 0:10

Settings

Custom classes

Confidence
0.50
0.00 1.00

Line position
0.60

Vehicle detection and counting

with Yolov5 and Deep SORT

Status: Finished !

Car	Bus	Truck	Motorcycle
22	0	4	0

FPS

2.8

Fig 10.9 Final Output

REFERENCES

[1] X. Cheng, L. Yang, and X. Shen, “D2D for intelligent transportation systems: a feasibility study,” IEEE Transactions on Intelligent Transportation Systems, vol. 16, no. 4, pp. 1784–1793, 2015.

[2] W. Wang, Y. Song, J. Zhang, and H. Deng, “Automatic parking of vehicles: a review of literatures,” International Journal of Automotive Technology, vol. 15, no. 6, pp. 967–978, 2014.

[3] M. M. Ahmed and M. A. Abdel-Aty, “The viability of using automatic vehicle identification data for real-time crash prediction,” IEEE Transactions on Intelligent Transportation Systems, vol. 13, no. 2, pp. 459–468, 2012.

[4] S. Sivaraman and M. M. Trivedi, “Looking at vehicles on the road: a survey of vision-based vehicle detection, tracking, and behavior analysis,” IEEE Transactions on Intelligent Transportation Systems, vol. 14, no. 4, pp. 1773–1795, 2013.

[5] H. Y. Cheng, C. C. Weng, and Y. Y. Chen, “Vehicle detection in aerial surveillance using dynamic Bayesian networks,” IEEE Transactions on Image Processing, vol. 21, no. 4, pp. 2152–2159, 2012.

[6] Y. Tian, H. Dong, L. Jia, and S. Y. Li, “A vehicle reidentification algorithm based on multi-sensor correlation,” Journal of Zhejiang University SCIENCE C, vol. 15, no. 5,

[7] Tang, Y., Zhang, C., Gu, R. et al. Vehicle detection and recognition for intelligent traffic surveillance system. Multimed Tools Appl 76, 5817–5832 (2017).

[8] P. Prabhakar, P. Anupama and S. R. Resmi, "Automatic vehicle number plate detection and recognition," 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), Kanyakumari, India, 2014, pp. 185-190, doi: 10.1109/ICCICCT.2014.6992954.

[9] Saran K B and Sreelekha G, "Traffic video surveillance: Vehicle detection and classification," 2015 International Conference on Control Communication & Computing India (ICCC), Trivandrum, India, 2015, pp. 516-521, doi: 10.1109/ICCC.2015.7432948.

[10] L. Unzueta et al., “Adaptive multicue background subtraction for robust vehicle counting and classification,” IEEE Trans. Intell. Transp. Syst. 13, 527–540 (2012).

[11] S.-Y. Cheung, and P.P. Varaiya, “Traffic surveillance by wireless sensor networks: Final report”, PhD diss., University of California at Berkeley, 2006.

[12] S. Oh, S. Ritchie, and C. Oh, “Real-time traffic measurement from single loop inductive signatures”, Transportation Research Record: Journal of the Transportation Research Board, (1804), pp. 98-106, 2002.