TRAFFIC DENSITY ESTIMATION

Dissertation submitted to

Shri Ramdeobaba College of Engineering & Management, Nagpur in partial fulfillment of requirement for the award

Masters in Computer Application

Fourth Semester

By

AYUSH MUKESH GUPTA

Guide

Prof. SATYAJIT SIDHESHWAR UPARKAR

Department of Computer Application

Shri Ramdeobaba College of Engineering & Management, Nagpur 440013

(An Autonomous Institute affiliated to Rashtrasant Tukdoji Maharaj Nagpur University Nagpur)

MAY 2024

TRAFFIC DENSITY ESTIMATION

SHRI RAMDEOBABA COLLEGE OF ENGINEERING & MANAGEMENT, NAGPUR

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

Department of Computer Applications

CERTIFICATE

This is to certify that the Thesis on **"Traffic Density Estimation"** is a bonafide work of Ayush Mukesh Gupta submitted to the Rashtrasant Tukdoji Maharaj Nagpur University, Nagpur in partial fulfillment of the award of a Masters in Computer Applications has been carried out at the Department of Computer Application, Shri Ramdeobaba College of Engineering and Management, Nagpur during the academic year 2024.

Date:

Place: Nagpur

Prof. Satyajit Sidheshwar Uparkar
Project guide
Department of Computer Applications

Dr. P.S. Voditel
H.O.D
Department of Computer Applications

Dr. R. S. Pande Principal

TRAFFIC DENSITY ESTIMATION

DECLARATION

I, hereby declare that the thesis titled **"Traffic Density Estimation"** submitted here in, has been carried out Shri Ramdeobaba College of Engineering & Management, Nagpur. The work is original and has not been submitted earlier as a whole or part for the award of any degree / diploma at this or any other institution / University

Date: 06/05/2024

Place: Nagpur

(Signature)

_____

(Ayush Mukesh Gupta)

Roll No : 12

TRAFFIC DENSITY ESTIMATION

APPROVAL SHEET

This thesis entitled **Traffic Density Estimation** by Ayush Mukesh Gupta is approved for the degree of Masters of Computer Applications.

Name & signature of guide
Dr. S. Uparkar

name & signature of co-guide prof.
Prof.P.Karmore

Name & signature of HOD
Dr. P.S. Voditel

Date:

Place: Nagpur

TRAFFIC DENSITY ESTIMATION

ACKNOWLEDGEMENTS

My major project title, “TRAFFIC DENSITY ESTIMATION” requires a lot of hard work, sincerity and systematic work methodologies. I express our deepest gratitude to my project guide **Prof. Satyajit Sidheshwar Uparkar** and project co-guide **Prof.Pravin Yashwantrao Karmore** for giving me an opportunity to be a part of this exciting project and guiding me at every step of the project.

I would like to thank them for the valuable guidance and advice. They inspired me greatly to work on this project. There willingness to motivate me contributed tremendously to my project. I would also like to thank them for explaining me some examples that related to the topic of my project.

I would also like to thank **Dr. P. S. Voditel**, Head of Department of Computer Application and all MCA faculty members who gave me important suggestions for the future improvement of my project. I am highly indebted to them.

I am also grateful to **Dr. R. S. Pande**, Principal, RCOEM for providing me the facilities of computer lab and other required infrastructure. I would like to thank MCA Library Department for providing me useful books related to my project.

Name of Project group members

Ayush Gupta

Abstract

This project is focused on leveraging YOLOv8, a popular object detection algorithm, to estimate real-time traffic density in urban and traffic management systems. The objective is to accurately count the number of vehicles in video frames to gain insights into traffic flow patterns.

One of the key achievements of this project is the development of a Top-View Vehicle Detection Image Dataset, which is specifically tailored for YOLOv8. This dataset provides aerial views of roads and highways, which are crucial for training computer vision models to detect vehicles accurately. By using this dataset, AI models can analyze traffic patterns more effectively and efficiently.

In terms of real-time application, the project demonstrates a YOLOv8-based model for traffic analysis, which can be used to enhance urban traffic management and planning strategies. This model can process video frames in real-time, making it ideal for monitoring traffic flow during peak hours.

CONTENT **PAGE NO.**

CHAPTER 1
INTRODUCTION

1.1 Title

Traffic Density Estimation

1.1 Introduction

Traffic density estimation is a crucial tool in managing urban traffic flow. It involves the use of sensors and cameras to collect data on the number and speed of vehicles on roads, which is then analyzed and used to optimize traffic light signals, identify bottlenecks, and improve overall road network performance. This technology has become increasingly important as cities continue to grow and traffic congestion becomes a major problem.

One of the key benefits of traffic density estimation is its ability to provide real-time data on traffic conditions. This allows traffic management systems to adjust traffic signals and other infrastructure in real-time, based on the current traffic flow. For example, if a particular intersection is experiencing heavy congestion, the traffic light signals can be adjusted to give more time to the main flow of traffic, reducing delays and improving overall traffic flow.

Another important use of traffic density estimation is in identifying and addressing bottlenecks in the road network. By analyzing data on traffic density and speed, traffic management systems can pinpoint areas where traffic is slowing down or coming to a standstill. This information can then be used to identify the root cause of the bottleneck, such as a poorly-designed intersection or a highway on-ramp that is too narrow, and take steps to address it.

Overall, traffic density estimation is a critical component of modern traffic management systems. It enables cities to better manage the flow of traffic, reduce congestion, and improve safety on the roads. Researchers are constantly working to refine methods and tools to enhance the accuracy and real-time capabilities of traffic density estimation technology, ensuring that it remains an effective tool for managing urban traffic flow well into the future.

1.2 Objective

YOLOv8 Model Selection and Initial Assessment: Starting with YOLOv8's pre-trained model selection, assessing its initial performance on COCO dataset for vehicle detection.

Specialized Vehicle Dataset Preparation: Curating and annotating a vehicle-specific dataset to refine the model's detection capabilities for diverse vehicle types.

Model Fine-Tuning for Enhanced Vehicle Detection: Employing transfer learning to fine-tune the YOLOv8 model, focusing on vehicle detection from aerial perspectives for improved precision and recall.

Comprehensive Model Performance Evaluation: Analyzing learning curves, evaluating confusion matrix, and assessing performance metrics to validate the model's accuracy and generalization capabilities.

Inference and Generalization on Test Data: Testing the model's generalization on validation images, an unseen test image, and a test video to demonstrate its practical application and effectiveness.

Real-Time Traffic Density Estimation: Implementing an algorithm to estimate traffic density by counting vehicles and analyzing traffic intensity in real-time on test video data.

Cross-Platform Model Deployment Preparation: Exporting the fine-tuned model in ONNX format for versatile use across different platforms and environments.

CHAPTER 2

PRELIMINARY SYSTEM ANALYSIS

2.1 Existing System

Traffic density estimation systems utilize various technologies and approaches to estimate the volume of vehicles on roads. Here are some existing systems commonly used for traffic density estimation:

a) **Inductive Loop Detectors**: These are electromagnetic sensors embedded in the road surface. They detect the presence of vehicles by measuring changes in inductance caused by the metal components of vehicles passing over them. The frequency of vehicle passages can be used to estimate traffic density.

b) **Video-Based Systems**: Cameras mounted alongside roads capture video footage of traffic. Computer vision algorithms analyze the footage to detect and track vehicles, estimating their speed, count, and density. These systems can use techniques like object detection, background subtraction, and optical flow analysis.

c) **Radar and LiDAR Sensors**: Radar and LiDAR sensors emit radio or laser pulses, respectively, and measure the time it takes for the pulses to reflect from objects, such as vehicles. By analyzing the returned signals, these sensors can determine the presence, speed, and position of vehicles, enabling traffic density estimation.

d) **Bluetooth and Wi-Fi Detection**: Some systems utilize Bluetooth or Wi-Fi signals emitted by smartphones or in-vehicle devices to detect and track vehicles. By analyzing the signal strength and changes in device proximity, these systems estimate traffic density and flow.

e) **GPS Data Analysis**: GPS-equipped vehicles continuously transmit location data, which can be collected and analyzed to estimate traffic density and flow. By aggregating GPS data from multiple vehicles, traffic patterns can be inferred and congestion areas identified.

f) **Mobile Apps and Crowdsourcing**: Mobile applications allow users to report traffic conditions in real time, including congestion and traffic density. By aggregating and analyzing user-generated data, these apps provide insights into current traffic conditions.

g) **Machine Learning Models**: Machine learning techniques, such as neural networks, can be trained on historical traffic data to predict traffic density based on various factors like time of day, day of the week, weather conditions, and events. These models can continuously learn and adapt to changing traffic patterns.

h) **Integration of Multiple Sensors**: Many systems integrate multiple sensor technologies, such as cameras, radar, and inductive loops, to improve the accuracy and reliability of traffic density estimation.

2.1.1. Feasibility Study

a. **Accuracy:** YOLOv8 is capable of detecting various objects with high accuracy, including vehicles. However, accurately estimating traffic density requires more than just object detection. You would need to implement additional logic to count and track vehicles over time to estimate density accurately.

b. **Speed:** YOLOv8 is optimized for real-time inference, making it suitable for processing video streams at high frame rates. However, the speed of inference may vary depending on the hardware you're using (CPU, GPU, or specialized accelerators) and the complexity of the scene.

c. **Training Data:** Training YOLOv8 for traffic density analysis would require a large dataset of annotated traffic scenes. This dataset should cover various lighting conditions, weather conditions, camera angles, and traffic densities to ensure robustness.

d. **Hardware Requirements:** Real-time inference with YOLOv8 typically requires powerful hardware, especially for high-resolution video streams. GPUs or specialized accelerators like TPUs are often used to accelerate inference speed.

e. **Integration with Tracking Algorithms:** To estimate traffic density accurately, you would need to integrate YOLOv8 with tracking algorithms to maintain consistent vehicle identities across frames. This helps in counting vehicles and measuring their speeds accurately.

f. **Environmental Factors:** Real-world traffic scenes can be complex, with factors like occlusions, varying lighting conditions, and camera motion affecting object detection performance. Your system should be robust enough to handle these challenges.

g. **Computational Resources:** Running YOLOv8 in real-time for traffic density analysis may consume significant computational resources, especially if you're processing multiple video streams simultaneously. You need to ensure that your infrastructure can handle the computational load efficiently.

h. **Deployment Considerations:** Once you have a trained model, deploying it in a real-time traffic monitoring system requires careful consideration of scalability, reliability, and latency requirements. You may need to optimize your deployment pipeline for efficient model serving.

CHAPTER 3

PROJECT CATEGORY SOFTWARE/ HARDWARE REQUIREMENTS

3.1 Technology Used

3.1.1 Hardware

1. **Graphics Processing Unit (GPU):**

   a. GPUs are crucial for accelerating the inference process, especially for deep learning tasks like object detection with YOLOv8.

   b. Modern GPUs with parallel processing capabilities significantly speed up the inference time compared to using CPUs alone.

   c. NVIDIA GPUs, such as those from the GeForce, Quadro, or Tesla series, are commonly used for deep learning tasks due to their CUDA support and optimized performance for neural network computations.

2. **Memory (RAM):**

   a. Sufficient RAM is necessary to store model parameters, intermediate computations, and temporary data during inference.

   b. Deep learning models like YOLOv8 can require significant memory, especially for processing high-resolution video frames.

3. **Storage:**

   a. Storage is needed for storing the YOLOv8 model files, training data, and any intermediate or processed data generated by the system.

   b. Solid-state drives (SSDs) are preferred for faster data access and retrieval compared to traditional hard disk drives (HDDs).

4. **Data Capture Devices:**

   a. Cameras or video feeds are required to capture traffic scenes and provide input to the system.

   b. The type and number of cameras depend on the coverage area and resolution requirements of the traffic monitoring system.

5. **Networking Components:**

    a. Networking hardware is necessary for transmitting video feeds, inference results, and other data between different components of the system.

    b. This includes routers, switches, and network cables for establishing connectivity between cameras, processing units, and storage devices.

3.1.2. Software

Operating System: Windows

Language: Python

Tools: Google Colab

3.1.2.1 Software

1. **Setting up Environment**: Google Colab provides an environment where you can install required libraries, including deep learning frameworks like PyTorch or TensorFlow, and other dependencies needed for your project. You can specify the runtime type to use GPU acceleration for training and inference.
2. **Accessing Data**: You can upload your dataset directly to Google Colab or mount Google Drive to access datasets stored there. Alternatively, you can download datasets directly from the internet within the Colab environment.
3. **Training YOLOv8**: You can write Python code in Colab notebooks to train YOLOv8 on your traffic dataset. You'll utilize frameworks like PyTorch or TensorFlow for this purpose. Colab's GPU acceleration can significantly speed up the training process compared to running on a CPU.
4. **Evaluation and Validation**: After training, you can evaluate the performance of your YOLOv8 model using validation datasets. You can visualize metrics such as precision,

recall, and mAP (mean Average Precision) directly within the Colab notebook.

5. **Real-Time Inference**: Once your YOLOv8 model is trained and validated, you can use it for real-time inference on traffic video streams. Colab provides options for reading video files or accessing live camera feeds, allowing you to process frames and perform object detection in real-time.
6. **Visualization and Analysis**: You can utilize libraries like OpenCV, Matplotlib, or Seaborn to visualize the results of traffic density analysis within the Colab notebook. This includes displaying annotated video frames with detected vehicles, plotting traffic density over time, or visualizing statistical metrics.
7. **Sharing and Collaboration**: One of the benefits of Google Colab is its ability to easily share and collaborate on notebooks. You can share your Colab notebook with collaborators for review, feedback, or collaborative development.
8. **Saving Models and Results**: You can save trained YOLOv8 models, evaluation metrics, and analysis results directly to your Google Drive or download them for further use or integration into other systems.

Google Colab provides a convenient and cost-effective platform for developing and prototyping deep learning projects like real-time traffic density analysis with YOLOv8, especially if you require GPU acceleration and collaboration features without the need for local hardware resources.

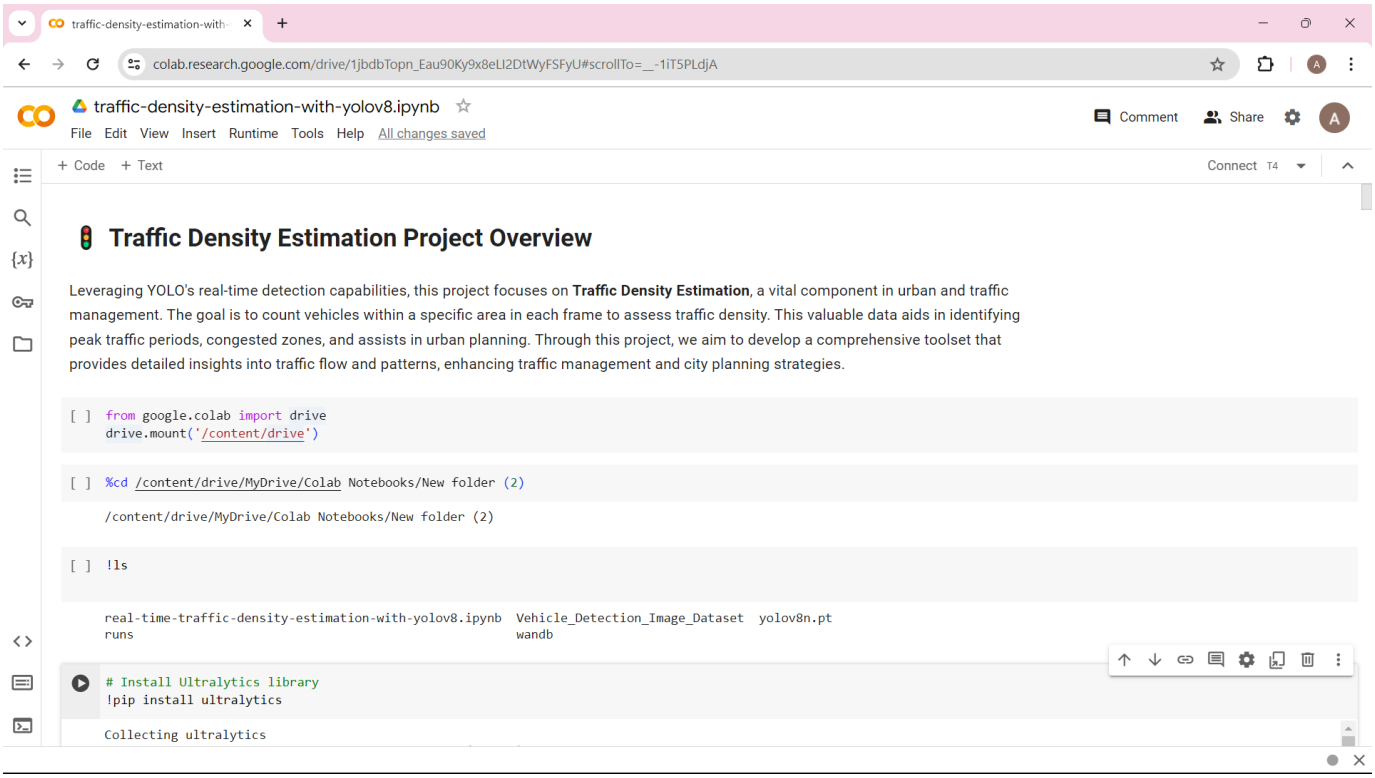


Figure 1: Google Colab

3.2 About language

3.2.1. Python Library:

Python libraries used in the code snippet you provided:

- **os:** This library provides functions for interacting with the operating system, such as navigating directories, manipulating file paths, and accessing environment variables.
- **shutil:** The shutil module offers a higher-level interface for file operations, such as copying, moving, and deleting files and directories.
- **numpy (np):** NumPy is a fundamental package for numerical computing in Python. It provides support for multi-dimensional arrays, matrices, and mathematical functions to operate on these arrays efficiently.
- **pandas (pd):** Pandas is a powerful library for data manipulation and analysis. It provides data structures like DataFrame and Series, along with functions for reading and writing data from various file formats, data cleaning, transformation, and analysis.
- **matplotlib.pyplot (plt):** Matplotlib is a comprehensive plotting library for creating static, interactive, and animated visualizations in Python. The pyplot module provides a MATLAB-like interface for creating plots and visualizations.
- **seaborn (sns):** Seaborn is a statistical data visualization library built on top of Matplotlib. It provides high-level functions for creating informative and attractive statistical graphics.
- **cv2:** OpenCV (Open Source Computer Vision) is a popular library for computer vision tasks, such as image and video processing, object detection, and feature extraction. The cv2 module provides functions and classes for these tasks.
- **yaml:** This library provides functions for parsing and serializing YAML (YAML Ain't Markup Language) files, which are commonly used for configuration and data serialization tasks.
- **PIL.Image:** The Python Imaging Library (PIL) provides functions for opening, manipulating, and saving many different image file formats. The Image module within PIL is used for working with images in various formats.
- **ultralytics.YOLO:** This seems to be a library or module for working with the YOLO (You Only Look Once) object detection algorithm. It likely provides functions and classes for training, testing, and deploying YOLO models.
- **IPython.display.Video:** This module from IPython.display provides functions for displaying video files directly within Jupyter notebooks or IPython environments.

3.2.2 Database

The dataset exclusively focuses on the 'Vehicle' class, encompassing a wide array of vehicles including cars, trucks, and buses, providing a comprehensive scope for vehicle detection models**.**

Pre-processing:

Each image has been standardized to a 640x640 resolution.

🔢 **Dataset Split:**

The dataset is divided into:

- **Training Set**: 536 images
- **Validation Set**: 90 images

🎭 **Augmentation on Training Set:**

Augmentation, including a 50% probability of horizontal flip, was exclusively applied to the training set to maintain the validation set's integrity.

This dataset facilitates the development of advanced vehicle detection models and contributes to smarter transportation and urban systems. Embark on this journey to create cutting-edge computer vision solutions for real-world traffic and urban challenges!

.

Vehicle_Detection_Image_Dataset (2 directories, 5 files)

1. **train:** This folder contains the training set, organized into two subfolders:
    - **images:** Holds 536 images used for training the model.
    - **labels:** Contains YOLOv8 format labels corresponding to the training images.
2. **valid:** This folder includes the validation set, also divided into two subfolders:
    - **images:** Contains 90 images for model validation.
    - **labels:** Contains YOLOv8 format labels for the validation images.

Files:

1. **data.yaml:**
    - The YAML file specifies paths to the training and validation datasets. It also defines the number of classes and the class name, essential for model training and validation.
2. **README.dataset.txt:**

Figure 3.2 Dataset

CHAPTER 4
SYSTEM DESIGN SPECIFICATION

4.1 Problem Definition

Problem Statement:

Design and implement a system for real-time traffic density analysis to monitor and analyze traffic flow on roads and highways.

Background:

Traffic congestion is a significant issue in urban areas and along major transportation routes. Understanding traffic patterns, density, and flow is essential for effective traffic management, infrastructure planning, and congestion mitigation efforts. Traditional manual methods of traffic monitoring are labor-intensive, time-consuming, and often lack real-time insights. Therefore, there is a need for automated systems that can accurately and efficiently analyze traffic density in real-time.

Project Objectives

Leveraging YOLO's real-time detection capabilities, this project focuses on **Traffic Density Estimation**, a vital component in urban and traffic management. The goal is to count vehicles within a specific area in each frame to assess traffic density. This valuable data aids in identifying peak traffic periods, congested zones, and assists in urban planning. Through this project, we aim to develop a comprehensive toolset that provides detailed insights into traffic flow and patterns, enhancing traffic management and city planning strategies

4.2 Proposed System Requirement Analysis

Hardware Requirements:

CPU: A multicore processor with sufficient processing power to handle real-time video processing tasks.

GPU: A dedicated GPU with CUDA support for accelerating deep learning inference tasks, especially for object detection with YOLOv8.

Memory (RAM): At least 16 GB of RAM to accommodate the computational demands of video processing, deep learning, and data analysis tasks.

Storage: Adequate storage space for storing datasets, trained models, intermediate results, and system logs. A solid-state drive (SSD) is recommended for faster data access.

Traffic Cameras: High-quality cameras capable of capturing clear and high-resolution video streams of traffic scenes. The number and placement of cameras depend on the coverage area and monitoring requirements.

Software Requirements:

Operating System: Support for Window operating systems is preferred due to better compatibility with deep learning frameworks and libraries.

Deep Learning Frameworks: Installation of PyTorch or TensorFlow for training and inference of deep learning models, including YOLOv8 for object detection.

Python Environment: Python 3.x environment with necessary libraries and dependencies installed, including OpenCV, NumPy, pandas, Matplotlib, Seaborn, and other required packages.

Video Processing Libraries: OpenCV for video input/output operations, frame manipulation, and real-time video processing tasks.

Annotation Tools: Optional annotation tools for labeling and annotating training datasets, such as LabelImg or LabelMe, if custom training datasets are required.

Development Tools: Integrated development environments (IDEs) like Google colab for writing, testing, and debugging code. Git for version control and collaboration.

Network Requirements:

Internet Connectivity: Stable internet connection for accessing external datasets, libraries, and resources, as well as for sharing and collaborating on project development.

Local Network: Local network infrastructure for connecting traffic cameras, data storage servers, and processing units within the system architecture.

Regulatory and Compliance Considerations:

Data Privacy and Security: Compliance with data privacy regulations and security standards, ensuring the protection of sensitive information captured by traffic cameras.

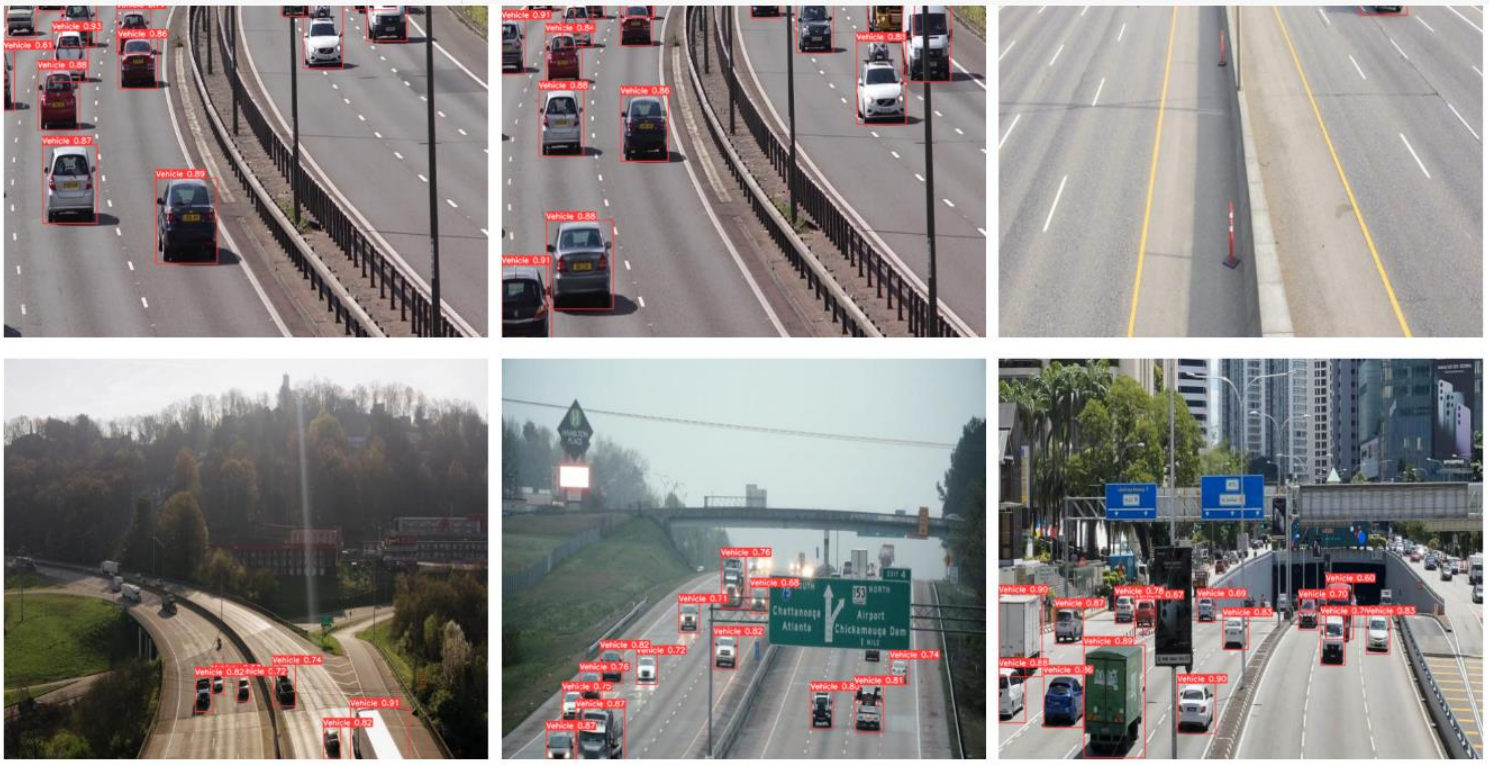
Legal Permissions: Obtaining necessary permissions and licenses for accessing and using traffic camera feeds, complying with local regulations and laws governing surveillance and data collection.

Scalability and Performance Requirements:

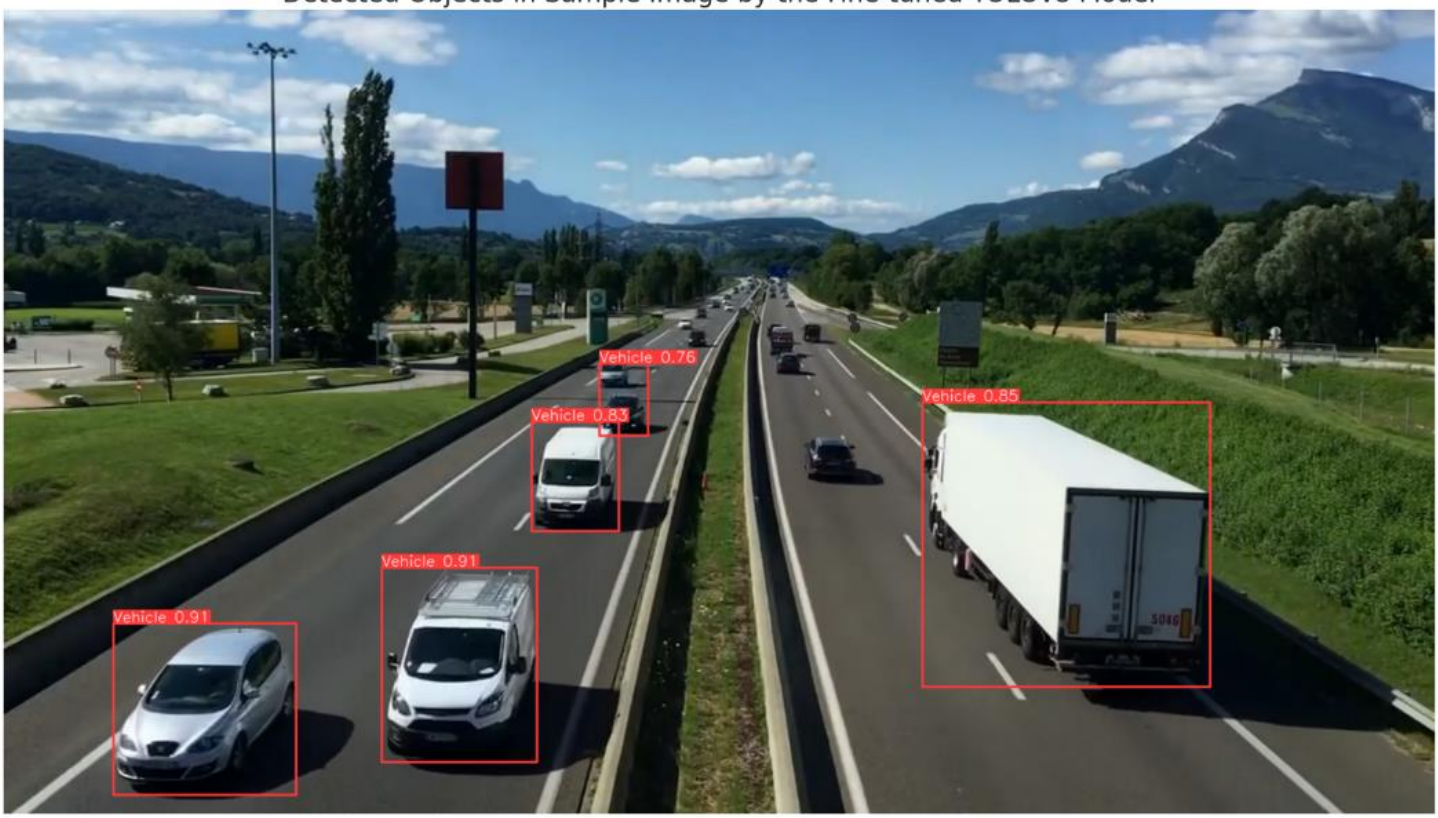
Scalability: The system should be scalable to accommodate additional traffic cameras, handle increased traffic volume, and support future expansion.

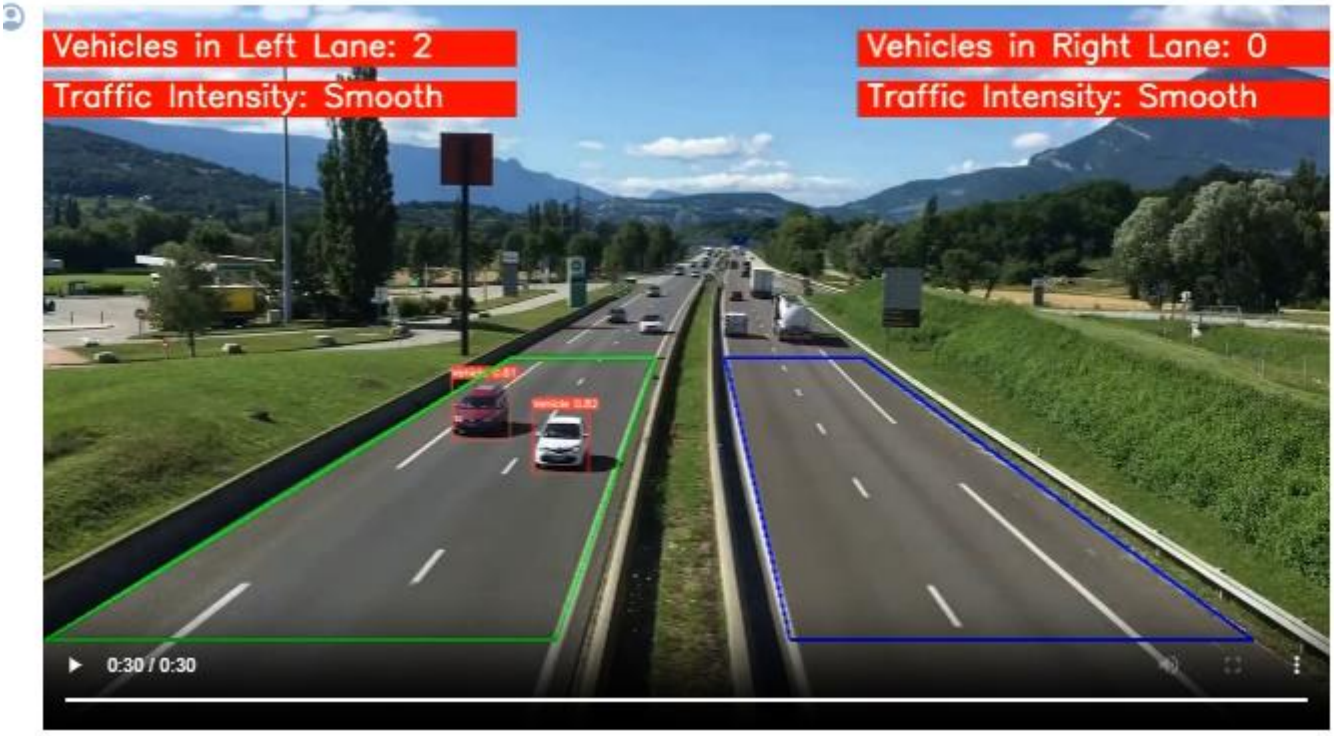
Performance: Real-time processing capability with low latency for analyzing video streams, ensuring timely detection and response to traffic incidents and congestion.

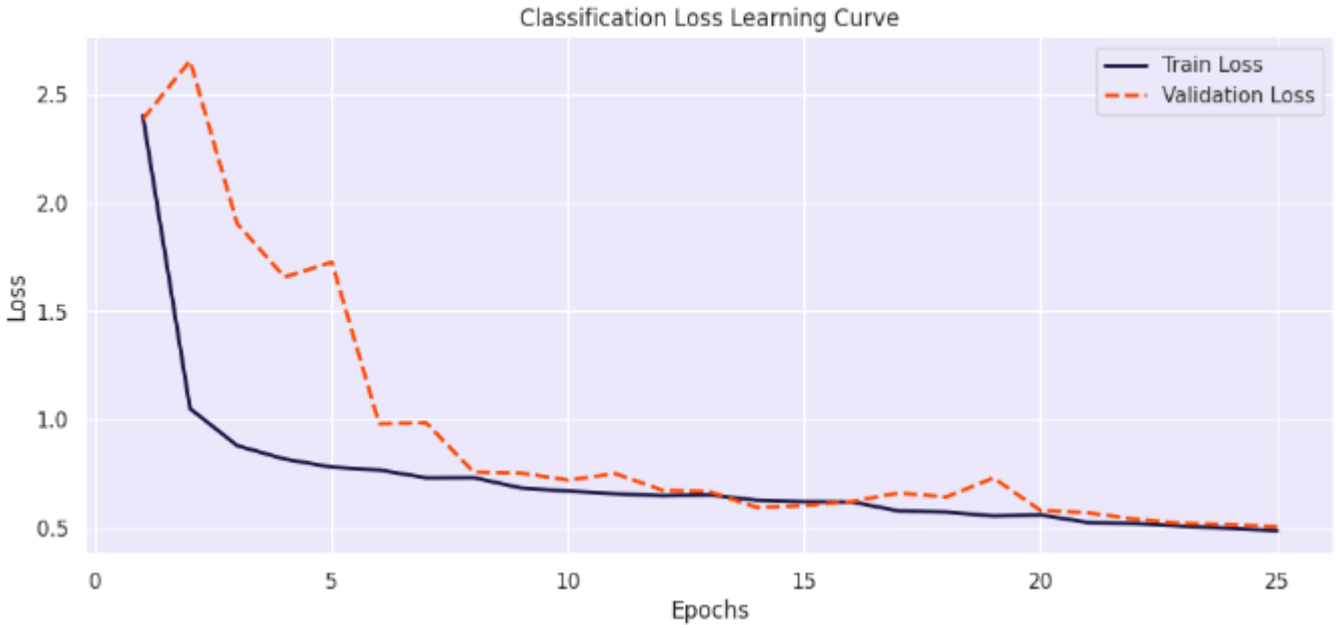
4.3 Result

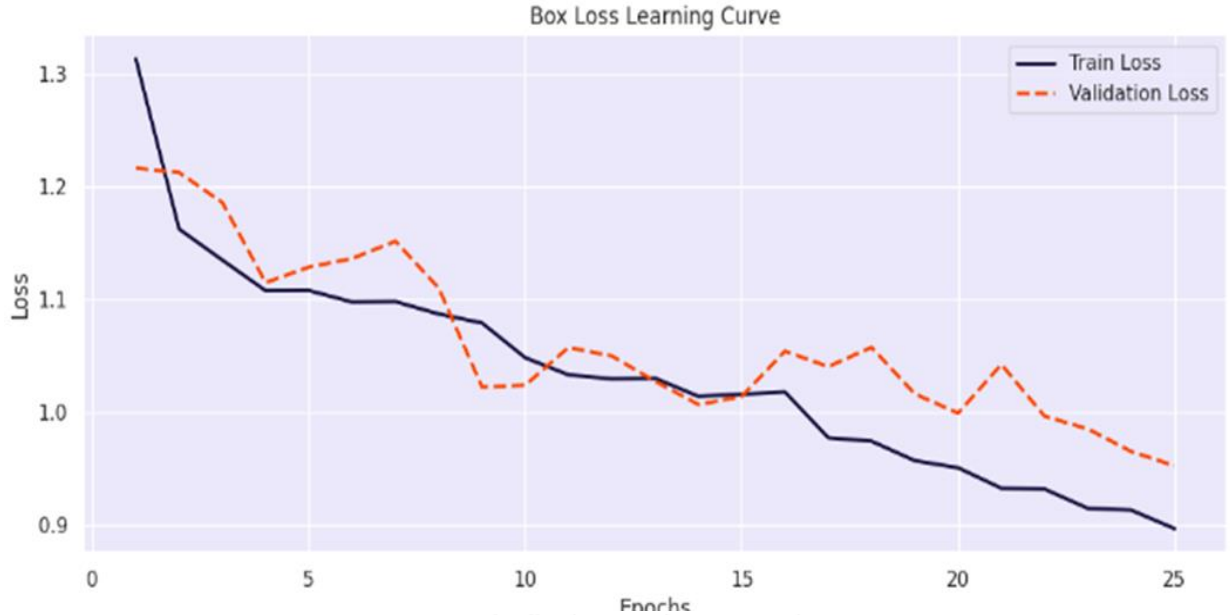


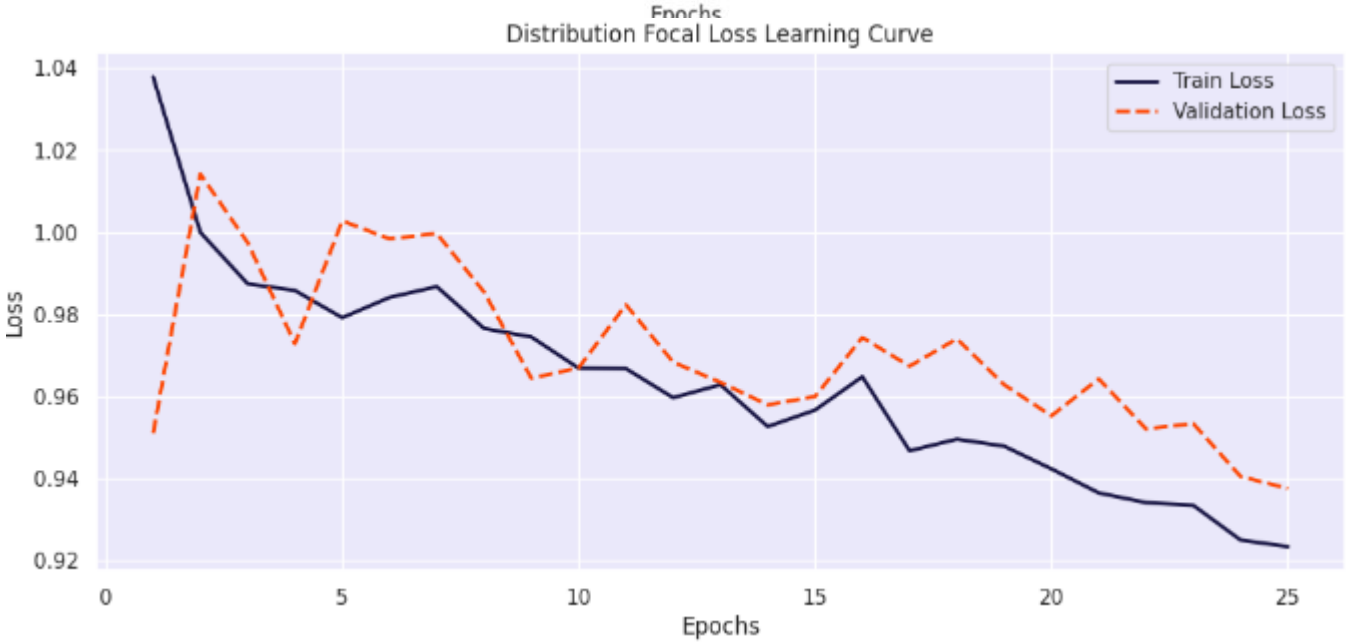
Detected Objects in Sample Image by the Fine-tuned YOLOv8 Model




Vehicles in Left Lane: 2
Traffic Intensity: Smooth
Vehicles in Right Lane: 0
Traffic Intensity: Smooth
0:30 / 0:30


Classification Loss Learning Curve
Train Loss
Validation Loss
Loss
2.5
2.0
1.5
1.0
0.5
0
5
10
15
20
25
Epochs


Box Loss Learning Curve
Train Loss
Validation Loss
Loss
1.3
1.2
1.1
1.0
0.9
0
5
10
15
20
25
Epochs


Distribution Focal Loss Learning Curve
Train Loss
Validation Loss
Loss
1.04
1.02
1.00
0.98
0.96
0.94
0.92
0
5
10
15
20
25
Epochs

CHAPTER 5
IMPLEMENTATION

5.1 Coding Explanation:

1. **Installation of Ultralytics Package**: We started by installing the Ultralytics package, which provides an easy-to-use interface for working with YOLOv8 models.
2. **Model Selection**: We opted for the YOLOv8 nano pre-trained model (yolov8n.pt) due to its fast inference time, making it suitable for real-time applications like traffic density estimation.
3. **Model Evaluation**: We evaluated the performance of the pre-trained YOLOv8 model on a sample image. However, it missed detectable vehicles due to its broad training on the COCO dataset, which contains diverse object categories.
4. **Fine-Tuning Process**: Recognizing the need for better accuracy in vehicle detection, we fine-tuned the YOLOv8 model on a specialized dataset focusing solely on vehicles. This involved resizing images to a uniform resolution of 640x640 pixels and applying augmentations to enhance model generalization.
5. **Training Output Files**: After fine-tuning, we obtained various output files, including trained weights, confusion matrix, learning curves, and performance metrics, which provide insights into model performance and training progress.
6. **Model Evaluation**: We tested the fine-tuned model on a sample image, demonstrating its improved accuracy in vehicle detection compared to the pre-trained model.
7. **Video Evaluation**: Finally, we assessed the generalization capabilities of the fine-tuned model on a new video, unseen during training, and converted the output to .mp4 format for easy playback and display.
8. **Installation of Ultralytics Package**: We started by installing the Ultralytics package, which provides an easy-to-use interface for working with YOLOv8 models.
9. **Model Selection**: We opted for the YOLOv8 nano pre-trained model (yolov8n.pt) due to its fast inference time, making it suitable for real-time applications like traffic density estimation.
10. **Model Evaluation**: We evaluated the performance of the pre-trained YOLOv8 model on a sample image. However, it missed detectable vehicles due to its broad training on the COCO dataset, which contains diverse object categories.
11. **Fine-Tuning Process**: Recognizing the need for better accuracy in vehicle detection, we

fine-tuned the YOLOv8 model on a specialized dataset focusing solely on vehicles. This involved resizing images to a uniform resolution of 640x640 pixels and applying augmentations to enhance model generalization.

12. **Training Output Files**: After fine-tuning, we obtained various output files, including trained weights, confusion matrix, learning curves, and performance metrics, which provide insights into model performance and training progress.

13. **Model Evaluation**: We tested the fine-tuned model on a sample image, demonstrating its improved accuracy in vehicle detection compared to the pre-trained model.

14. **Video Evaluation**: Finally, we assessed the generalization capabilities of the fine-tuned model on a new video, unseen during training, and converted the output to .mp4 format for easy playback and display.

5.2 Coding screenshot:

```
[ ] # Load a pretrained YOLOv8n model from Ultralytics
    model = YOLO('yolov8n.pt')
```

```
    # Path to the image file
    image_path = '/content/drive/MyDrive/Colab Notebooks/New folder (2)/Vehicle_Detection_Image_Dataset/sample_image.jpg'


    # Perform inference on the provided image(s)
    results = model.predict(source=image_path,
                            imgsz=640,  # Resize image to 640x640 (the size pf images the model was trained on)
                            conf=0.5)   # Confidence threshold: 50% (only detections above 50% confidence will be considered)

    # Annotate and convert image to numpy array
    sample_image = results[0].plot(line_width=2)

    # Convert the color of the image from BGR to RGB for correct color representation in matplotlib
    sample_image = cv2.cvtColor(sample_image, cv2.COLOR_BGR2RGB)

    # Display annotated image
    plt.figure(figsize=(20,15))
    plt.imshow(sample_image)
    plt.title('Detected Objects in Sample Image by the Pre-trained YOLOv8 Model on COCO Dataset', fontsize=20)
    plt.axis('off')
    plt.show()
```

```python
# Define the dataset_path
dataset_path = '/content/drive/MyDrive/Colab Notebooks/New folder (2)/Vehicle_Detection_Image_Dataset'


# Set the path to the YAML file
yaml_file_path = os.path.join(dataset_path, 'data.yaml')

# Load and print the contents of the YAML file
with open(yaml_file_path, 'r') as file:
    yaml_content = yaml.load(file, Loader=yaml.FullLoader)
    print(yaml.dump(yaml_content, default_flow_style=False))
```

```
names:
- Vehicle
nc: 1
roboflow:
  license: CC BY 4.0
  project: vehicle_detection_yolov8
  version: 3
train: /content/drive/MyDrive/Colab Notebooks/New folder (2)/Vehicle_Detection_Image_Dataset/train/images
val: /content/drive/MyDrive/Colab Notebooks/New folder (2)/Vehicle_Detection_Image_Dataset/valid/images
```

```python
# Set paths for training and validation image sets
train_images_path = os.path.join(dataset_path, 'train', 'images')
valid_images_path = os.path.join(dataset_path, 'valid', 'images')

# Initialize counters for the number of images
num_train_images = 0
num_valid_images = 0

# Initialize sets to hold the unique sizes of images
train_image_sizes = set()
valid_image_sizes = set()

# Check train images sizes and count
for filename in os.listdir(train_images_path):
    if filename.endswith('.jpg'):
        num_train_images += 1
        image_path = os.path.join(train_images_path, filename)
        with Image.open(image_path) as img:
            train_image_sizes.add(img.size)

# Check validation images sizes and count
for filename in os.listdir(valid_images_path):
    if filename.endswith('.jpg'):
        num_valid_images += 1
        image_path = os.path.join(valid_images_path, filename)
        with Image.open(image_path) as img:
            valid_image_sizes.add(img.size)

# Print the results
print(f"Number of training images: {num_train_images}")
print(f"Number of validation images: {num_valid_images}")

# Check if all images in training set have the same size
if len(train_image_sizes) == 1:
    print(f"All training images have the same size: {train_image_sizes.pop()}")
else:
    print("Training images have varying sizes.")

# Check if all images in validation set have the same size
if len(valid_image_sizes) == 1:
    print(f"All validation images have the same size: {valid_image_sizes.pop()}")
else:
    print("Validation images have varying sizes.")
```

```
Number of training images: 536
Number of validation images: 90
All training images have the same size: (640, 640)
All validation images have the same size: (640, 640)
```

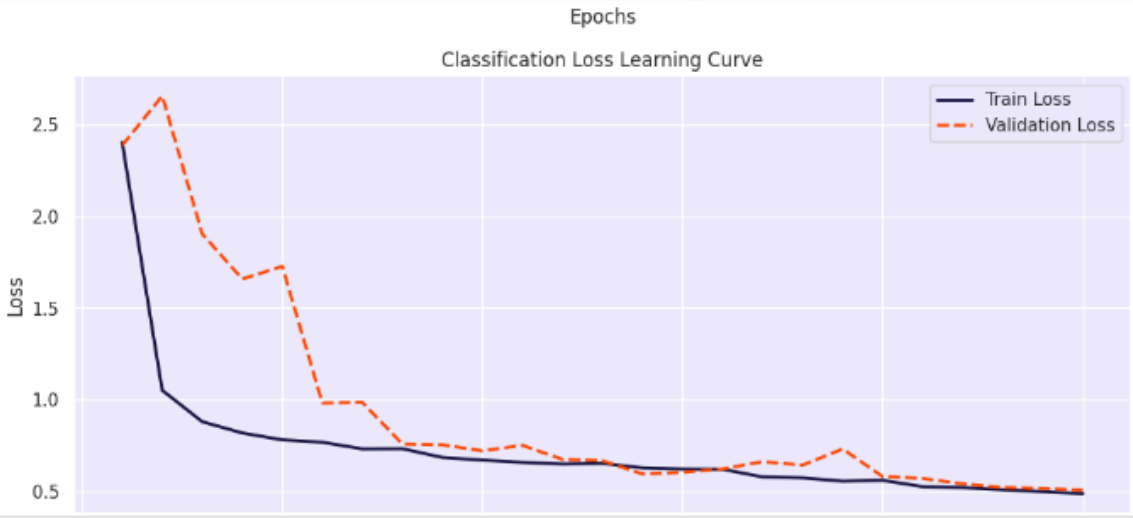
```
[ ] # Define a function to plot learning curves for loss values
    def plot_learning_curve(df, train_loss_col, val_loss_col, title):
        plt.figure(figsize=(12, 5))
        sns.lineplot(data=df, x='epoch', y=train_loss_col, label='Train Loss', color='#141140', linestyle='-', linewidth=2)
        sns.lineplot(data=df, x='epoch', y=val_loss_col, label='Validation Loss', color='orangered', linestyle='--', linewidth=2)
        plt.title(title)
        plt.xlabel('Epochs')
        plt.ylabel('Loss')
        plt.legend()
        plt.show()
```

```
# Create the full file path for 'results.csv' using the directory path and file name
results_csv_path = os.path.join(post_training_files_path, 'results.csv')

# Load the CSV file from the constructed path into a pandas DataFrame
df = pd.read_csv(results_csv_path)

# Remove any leading whitespace from the column names
df.columns = df.columns.str.strip()

# Plot the learning curves for each loss
plot_learning_curve(df, 'train/box_loss', 'val/box_loss', 'Box Loss Learning Curve')
plot_learning_curve(df, 'train/cls_loss', 'val/cls_loss', 'Classification Loss Learning Curve')
plot_learning_curve(df, 'train/dfl_loss', 'val/dfl_loss', 'Distribution Focal Loss Learning Curve')
```



CHAPTER 6

TESTING

6.1 Testing

Software testability is simply how easily computers programs can be tested. The checklist that follows provides a set of characteristics that lead to testable software.

- ➢ Portability.
- ➢ Observable.
- ➢ Controllability.
- ➢ Decomposability.
- ➢ Simplicity.
- ➢ Stability.
- ➢ Understandability

6.1.1 Why Testing is Important?

- ➢ Meets the requirements that guided its design and development,
- ➢ Works as expected,
- ➢ Can be implemented with the same characteristics, and
- ➢ Satisfies the needs of stakeholders.

Software testing, depending on the testing method employed, can be implemented at any time in the software development process. Traditionally most of the test effort occurs after the requirements have been defined and the coding process has been completed, but in the Agile approaches most of the test effort is on-going. As

such, the methodology of the test is governed by the chosen software development methodology.

Testing is the process of evaluating a system or application to ensure that it meets its specified requirements and works correctly. Testing is essential in software development as it helps identify defects, errors, and vulnerabilities in the system or application, and ensures that it meets the user's requirements and expectations. Testing also helps to ensure the quality of the software and provides a means to validate that the application is functioning as intended.

For the current application, which involves virtual try-on of necklaces using computer vision and machine learning, different types of testing are required to ensure its functionality and reliability. Some of the testing that needs to be done for this application are:

1. Unit Testing: This testing is done to verify the functionality of individual units or components of the software. In the case of this application, unit testing can be done to verify that the pose detection, landmark detection, and image processing components are working as expected.

2. Integration Testing: This testing is done to verify that different components of the software work correctly when integrated with each other. For this application, integration testing can be done to verify that the pose detection, landmark detection, and image processing components work correctly together.

3. Functional Testing: This testing is done to verify that the software meets its specified requirements and performs its intended functions. For this application, functional testing can be done to ensure that the virtual try-on functionality works as intended and that the user can select and view different necklaces.

4. Performance Testing: This testing is done to verify that the software meets its performance requirements, such as response time, scalability, and resource utilization. For this application, performance testing can be done to ensure that the application runs smoothly, even when multiple users are trying to access it simultaneously.

5. Security Testing: This testing is done to verify that the software is secure and free from vulnerabilities that could be exploited by attackers. For this

application, security testing can be done to ensure that the user's personal information and images are stored securely, and the application is protected from potential attacks.

6. Usability Testing: This testing is done to verify that the software is user-friendly and easy to use. For this application, usability testing can be done to ensure that the user interface is intuitive and easy to navigate, and the virtual try-on functionality is easy to use and understand.

In summary, testing is a critical aspect of software development, and for the current application, different types of testing need to be done to ensure that it meets its specified requirements, works correctly, and is reliable, secure, and user-friendly.

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 Conclusion

In conclusion, the landscape of urban mobility is undergoing a transformative shift driven by innovation, efficiency, and safety, all propelled by AI-powered traffic density analysis systems. As we strive for a more sustainable future, these systems play a crucial role in optimizing transportation networks and enhancing the overall mobility experience.

The integration of AI-driven traffic density analysis with sustainable mobility options such as public transportation, ride-sharing, and electric vehicles is reshaping urban transportation paradigms. By providing real-time insights into traffic patterns, these systems enable more efficient route planning, reduce congestion, and minimize environmental impact.

Furthermore, AI-powered traffic density analysis systems enhance safety and security on the roads by enabling proactive measures to mitigate traffic incidents and improve emergency response times. By leveraging advanced technologies such as computer vision and machine learning, these systems can identify potential hazards and alert authorities and drivers in real-time, ultimately saving lives and reducing accidents.

However, as we embrace the benefits of AI in traffic density analysis, it is essential to address ethical considerations and privacy concerns. Striking a balance between data-driven insights and individual privacy rights is paramount to ensure the responsible and ethical use of these technologies.

Looking ahead, the future prospects for AI-powered traffic density analysis are promising, with continued advancements in AI algorithms, sensor technology, and data analytics. By harnessing the power of AI, we can create smarter, safer, and more sustainable transportation systems

7.2 Future Scope

The future scope for AI-powered traffic density analysis is vast and holds great potential for further advancements and applications. Here are some key areas of future development and opportunities:

1. **Advanced Machine Learning Algorithms**: Continued advancements in machine learning algorithms, including deep learning techniques, can lead to more accurate and efficient traffic density analysis. Research into novel architectures and training strategies can further improve the performance of AI models in understanding complex traffic patterns.

2. **Integration with Smart Cities**: AI-powered traffic density analysis can be integrated into broader smart city initiatives to optimize urban transportation systems. This integration can enable real-time traffic management, adaptive traffic signal control, and predictive modeling to reduce congestion and enhance mobility.
3. **Multi-Modal Transportation**: Future developments in AI can enable the analysis of multi-modal transportation systems, including not only vehicles but also pedestrians, cyclists, and public transit. This holistic approach can provide a comprehensive understanding of urban mobility patterns and facilitate more seamless integration between different modes of transportation.
4. **Edge Computing and IoT**: The use of edge computing and IoT (Internet of Things) devices can enable real-time data collection and analysis at the network edge, reducing latency and improving scalability for traffic density analysis systems. This can lead to more responsive and adaptive traffic management solutions.
5. **Autonomous Vehicles**: As autonomous vehicle technology continues to evolve, AI-powered traffic density analysis will play a crucial role in supporting safe and efficient deployment. AI algorithms can help autonomous vehicles navigate complex traffic environments, anticipate traffic conditions, and make informed decisions in real-time.
6. **Predictive Analytics**: Future advancements in AI can enable predictive analytics capabilities for traffic density analysis, allowing transportation authorities to anticipate congestion, plan for future infrastructure investments, and optimize transportation services based on anticipated demand.
7. **Environmental Impact Assessment**: AI-powered traffic density analysis can be extended to assess the environmental impact of transportation systems, including emissions, air quality, and carbon footprint. This can inform policy decisions and initiatives aimed at promoting sustainable transportation practices.
8. **Privacy-Preserving Solutions**: Addressing privacy concerns and ensuring the ethical use of data will be critical for the future of AI-powered traffic density analysis. Developing privacy-preserving AI algorithms and adopting transparent data governance frameworks can help build trust and ensure responsible deployment of these technologies.

Overall, the future of AI-powered traffic density analysis is bright, with opportunities for innovation and positive impact on urban mobility, sustainability, and safety. By leveraging AI technologies responsibly and collaboratively, we can create more efficient, equitable, and resilient transportation systems for the cities of tomorrow.

8.1 Bibliography

- References: A.D. May
- Traffic Flow Fundamentals

  Prentice Hall, Englewood Cliffs (1990)
- R.P. Roess, E.S. Prassas, W.R. McShane

  Traffic Engineering (third ed.), Pearson Prentice Hall (2004)
- Highway Capacity Manual

  TRB

  National Research Council, Washington, DC (2010)
- O. Thabet, Modeling and Macroscopic Simulation Of Traffic Streams on Multi-Lane Highways (Master Thesis), Cairo University, Giza, 2010.
- Al Kherret, A. Al Sobky, R. Mousa, Video-based detection and tracking model for traffic surveillance. Paper No. 15-1465 Presented at the 94th TRB Annual Meeting, Washington, DC, January 2015.
- Al Kherret, Video-Based Detection and Tracking Model for Acquiring Traffic Data (Ph.D. Dissertation), Cairo University, 2015.