LICENSE PLATE RECOGNITION USING YOLO

FINAL PROJECT REPORT

Submitted by

T S RAMESHKUMAR (23205040)

in partial fulfilment for the award of the degree

of

Postgraduate

In

Master of Computer Applications

Rathinam Technical Campus

Eachanari – 641021

An Autonomous Institution

Affiliated to Anna University, Chennai – 600 025

MAR – 2025

ANNA UNIVERSITY, CHENNAI

BONAFIDE CERTIFICATE

Certified that this Report titled "**LICENSE PLATE RECOGNITION USING YOLO"** is the bonafide work of **RAMESHKUMAR T S (23205040)** who carried out under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other phase II report or dissertation on the basis of which a degree or ward conferred on an earlier occasion on this or any other candidate.

Dr.A.B.Arockia Christopher
Professor and Head
Master of Computer Applications
Rathinam Technical Campus
Coimbatore
Anna University, Chennai.

Submitted for the Final project report Viva – Voce examination held on ……………

Internal Examiner External Examiner

ABSTRACT

LICENSE PLATE RECOGNITION USING YOLO

This report presents the development and integration of a robust license plate detection system leveraging the capabilities of YOLO (You Only Look Once) for real-time object detection and EasyOCR for Optical Character Recognition (OCR). The primary objective is to create a comprehensive solution capable of accurately localizing license plates within images and extracting alphanumeric characters efficiently.

The system addresses challenges inherent in diverse license plate designs and varying lighting conditions, critical for applications like traffic surveillance and security. The integration of YOLO enables swift identification of vehicles and their corresponding license plates, ensuring efficient real-time detection. EasyOCR is employed to perform detailed character recognition on the detected license plates, ensuring accurate extraction of license plate numbers. The combined system aims for high accuracy in detection while maintaining quick processing speed, contributing significantly to the effectiveness of the solution.

ACKNOWLEDGEMENT

Apart from the efforts of us, the success of this project depends largely on the encouragement and guidelines of many others. We take this opportunity to praise the **almighty** and express our gratitude to the **people** who have been instrumental in the successful completion of our project.

We wish to acknowledge with thanks for the excellent encouragement given by the management of our college and we thank **Dr. B.Nagaraj, M.E., Ph.D., PDF (Italy) , CBO** for providing us with a plethora of facilities in the campus to complete our project successfully.

We wish to express our hearty thanks to **Dr. K.Geetha, M.E., Ph.D., Principal** of our college, for her constant motivation regarding our internship towards project.

We extend my heartfelt gratitude to **Dr.A.B.Arockia Christopher, M.E., Ph.D., Professor & HoD– MCA** for his tremendous support and assistance in the completion of our project.

It is our primary duty to thank our **Project guide, Dr.A.B.Arockia Christopher, M.E., Ph.D., Professor & HoD– MCA** who is the backbone of all our project activities, It's her enthusiasm and patience that guided us through the right path.

Finally, we extend our heartfelt thanks to the **parents, friends, and faculty members** for their constant support throughout this internship. The guidance and support received from all the members who contributed to the success of the project.

RAMESHKUMAR T S

CONTENTS

CHAPTER – 1
INTRODUCTION

1.1 INTRODUCTION

Detecting license plate numbers using computer vision, especially with the integration of YOLO (You Only Look Once) and Optical Character Recognition (OCR), offers a range of practical applications and benefits. Here's an introduction to why this process is valuable:

In the ever-evolving landscape of technology, the integration of computer vision has brought forth innovative solutions, and one particularly compelling application is the detection of license plate numbers. This technology has proven to be immensely useful in various domains, offering a blend of efficiency, accuracy, and automation.

One primary utility of license plate detection through computer vision lies in enhancing security and surveillance systems. By leveraging advanced algorithms, such as YOLO, to detect license plates in real-time, it becomes possible to monitor and track vehicles seamlessly. This is particularly advantageous in the context of law enforcement, where quick identification of vehicles is essential for tasks ranging from traffic management to criminal investigations.

The use of YOLO in license plate detection adds a layer of sophistication to the process. YOLO's ability to process images in a single pass, providing rapid and accurate bounding box predictions, is crucial in scenarios where real-time responses are paramount

The benefits of this approach extend beyond security and law enforcement. Industries such as transportation, parking management, and smart city initiatives can leverage license plate detection to streamline operations, enhance efficiency, and improve overall service delivery.

1.2 SCOPE OF THE PROJECT

The License Plate Recognition (LPR) system using YOLO (You Only Look Once) is an advanced, real-time solution for vehicle identification and traffic monitoring. It leverages deep learning-based object detection to accurately detect and recognize license plates from images and video streams. This system is highly efficient, scalable, and suitable for various applications, including automated toll collection, traffic enforcement, and smart parking.

Key Features and Capabilities

► **Real-time Detection and Recognition:** The system uses YOLO for fast and precise license plate detection, followed by Optical Character Recognition (OCR) for reading plate numbers.

► **High Accuracy and Performance:** Trained on diverse datasets, the model ensures reliable detection across different lighting conditions, angles, and vehicle speeds.

► **Multi-Camera Support:** The system can process feeds from multiple cameras simultaneously, enabling comprehensive surveillance.

► **Cloud Integration:** Integrates with platforms like AWS, Google Cloud, or ThingsBoard for real-time data processing, storage, and remote access.

► **Scalability:** Easily expandable to support additional cameras and regions for larger deployments.

► **Automated Alerts and Logging:** Generates alerts for unauthorized vehicles, stolen cars, or traffic violations based on predefined criteria.

► **Edge AI Processing:** Supports edge deployment on embedded devices like NVIDIA Jetson, Raspberry Pi, or industrial PCs for on-premise processing without cloud dependency.

Applications

The LPR system is versatile and applicable across various industries, including:

► **Traffic Management:** Real-time monitoring of vehicles, enforcing traffic laws, and reducing congestion.

► **Automated Toll Collection:** Enables fast and contactless toll payment using ANPR (Automatic Number Plate Recognition).

► **Smart Parking Systems:** Automates entry and exit logging in parking lots, reducing manual intervention.

► **Law Enforcement:** Assists in tracking stolen or unauthorized vehicles, enhancing public safety.

► **Access Control:** Used in secured areas like corporate buildings and residential communities for vehicle authentication.

► **Fleet Management:** Helps logistics and transport companies track vehicle movements efficiently.

The License Plate Recognition system using YOLO provides a robust and intelligent solution for modern traffic surveillance and management, ensuring improved security, efficiency, and automation across various domains.

1.3 OBJECTIVE OF THE PROJECT

Primary Objective: The primary objective of the License Plate Recognition (LPR) system using YOLO is to design and develop a reliable, real-time solution for vehicle identification and traffic monitoring. By leveraging deep learning-based object detection, the system aims to enhance security, automate processes, and improve traffic management through accurate and efficient license plate recognition.

Specific Objectives:

► **Real-Time Detection and Recognition:** Develop a system capable of detecting and recognizing license plates in real time using YOLO and Optical Character Recognition (OCR).

► **High Accuracy and Performance:** Train the model on diverse datasets to ensure reliable detection across various lighting conditions, angles, and vehicle speeds.

► **Multi-Camera Support:** Implement a scalable framework to process video feeds from multiple cameras simultaneously for enhanced surveillance.

► **Cloud Integration:** Enable secure data transfer and storage on platforms like AWS, Google Cloud, or ThingsBoard for remote access, visualization, and analysis.

► **Automated Alerts and Logging:** Configure the system to generate real-time alerts for unauthorized vehicles, stolen cars, or traffic violations.

► **Edge AI Processing:** Support deployment on the PhyBOARD-Pollux i.MX 8M Plus for on-premise processing and reduced cloud dependency.

► **Scalability and Flexibility:** Design the system to accommodate additional cameras, regions, and evolving requirements for large-scale deployments.

► **Access Control and Security:** Implement license plate-based authentication for secured areas, enhancing safety in corporate buildings, residential complexes, and restricted zones.

By achieving these objectives, the License Plate Recognition system using YOLO will provide a smarter, automated approach to traffic monitoring and vehicle authentication. It will contribute to improved security, operational efficiency, and intelligent decision-making across various industries, supporting advancements in smart cities and intelligent transportation systems.

1.4 METHODOLOGY

1.4.1 Data Collection

For training our YOLOv8 model, we are using two datasets, namely:

1. Indian vehicle number plate yolo annotation

   At the time of our research, this dataset contained 161 images of Indian vehicle number plates along with their appropriate YOLO annotations.

2. Car Number Plate Detection

   At the time of our research, this dataset contained 931 images of the front and rear sides of the car which are mostly found in India. This dataset however lacked the required YOLO annotations.

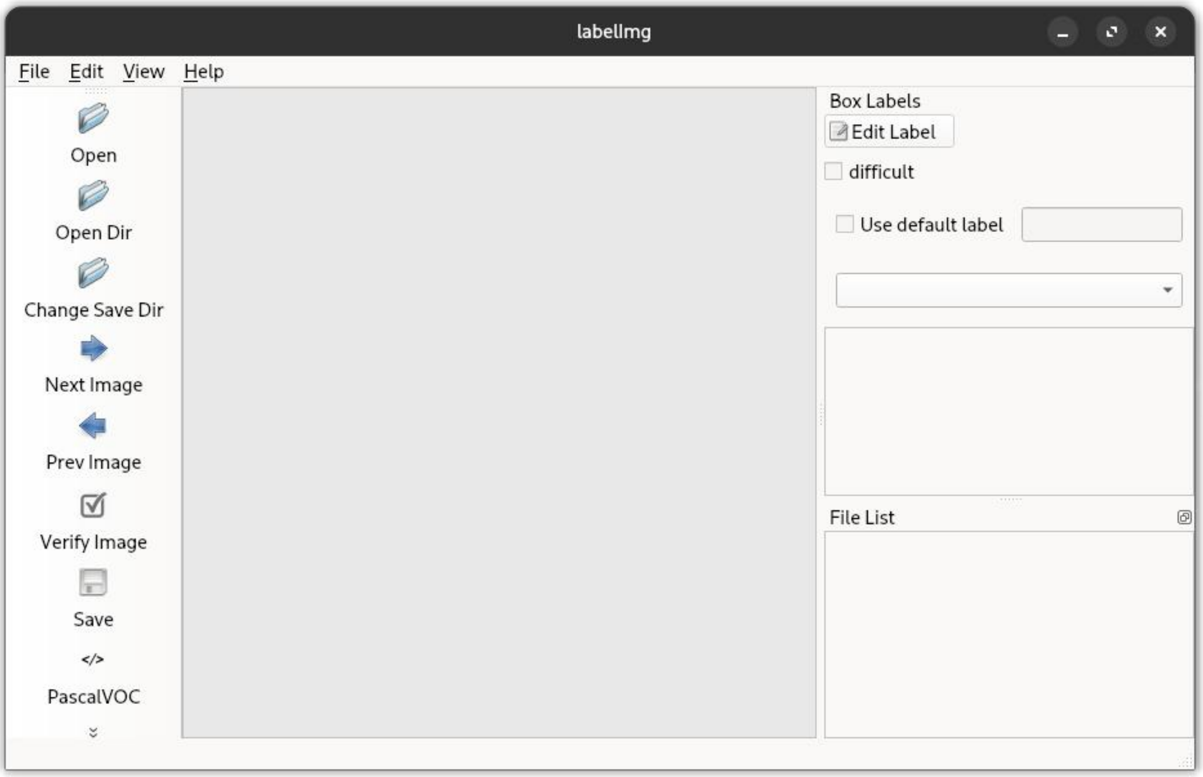
1.4.2 Data Preparation

Since the Car Number Plate Detection dataset did not contain the required YOLO annotations, we had to add the appropriate annotations manually. For this purpose, we chose the **labelImg** python package to help ease the annotation process.

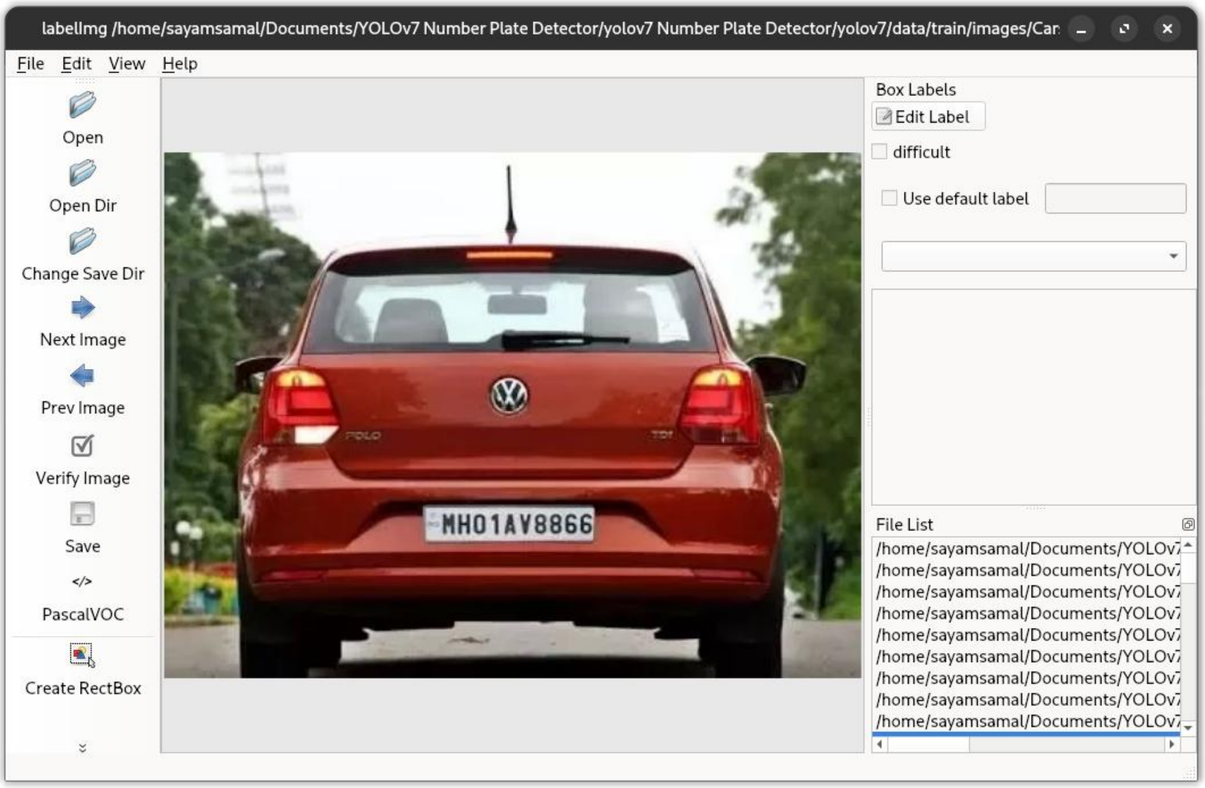
1. We first install the package using the following command:

   ```
   pip install labelImg
   ```

2. After installing the package, we open the terminal and run the 'labelImg' command. This opens up the labelImg GUI.



3. We then proceed to open the image directory where we downloaded the Car Number Plate Detection dataset.


labelImg /home/sayamsamal/Documents/YOLOv7 Number Plate Detector/yolov7 Number Plate Detector/yolov7/data/train/images/Car

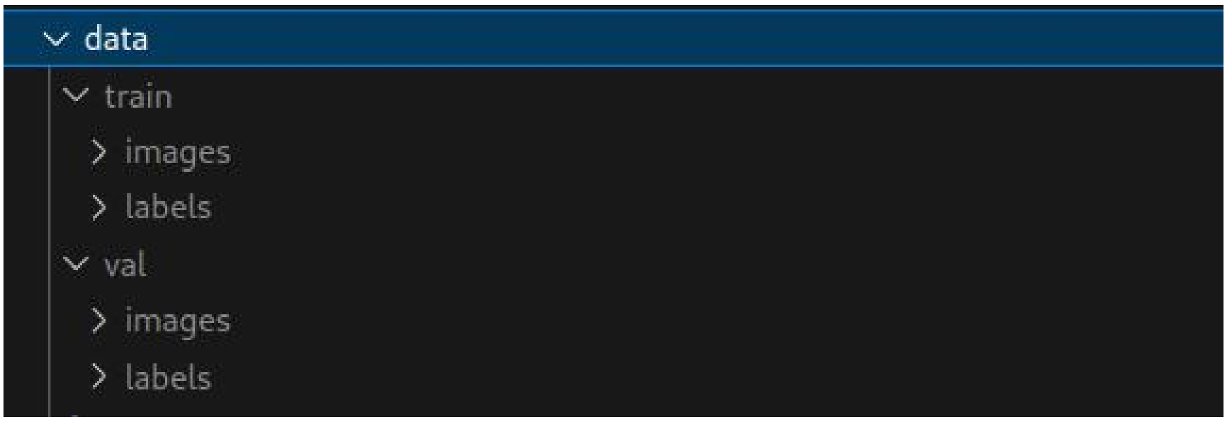
```
/home/sayamsamal/Documents/YOLOv7
/home/sayamsamal/Documents/YOLOv7
/home/sayamsamal/Documents/YOLOv7
/home/sayamsamal/Documents/YOLOv7
/home/sayamsamal/Documents/YOLOv7
/home/sayamsamal/Documents/YOLOv7
/home/sayamsamal/Documents/YOLOv7
/home/sayamsamal/Documents/YOLOv7
/home/sayamsamal/Documents/YOLOv7
```

4. We then add the required annotations using the easy-to-use GUI, making sure to go through all the images in the dataset.
5. Next, we set the save directory using the **Change Save Dir,** and set it to a folder named “labels”.
6. Finally, we check to ensure that the format is set to YOLO and not to PascalVOC, and then we click on the **Save** button to save the annotations.

To train as well as validate our model, we then separate the data into two groups,

1. We create a folder called “**train**”, containing 80% of the images and their corresponding labels.
2. Next, we create a folder called “**val**”, containing the rest 20% of the images and their corresponding labels.

Finally, we should be left with the following directory structure inside the data directory:



```
v data
  v train
    > images
    > labels
  v val
    > images
    > labels
```

1.4.3 Training YOLOv8 on our Custom Data

To train the YOLOv8 model, we used **Google Colab,** to harness the power of specialized GPUs for faster model learning.

1. We first install the required package for YOLOv8, using the following command
2. We then upload the data folder containing the images and the corresponding labels.
3. Next, we define our custom configuration “**custom-data.yaml**”

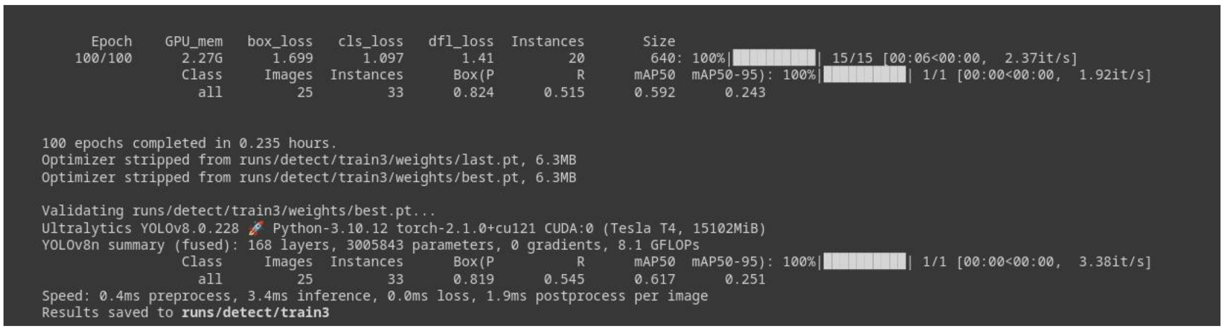
```
# train and val data as 1) directory: path/images/, 2) file: path/images.txt, or 3) list: [path1/images/, path2/images/]
        train: ./train
        val: ./val
# number of classes
        nc: 1
# class names
        names: ['number plate']
```

4. Finally, we train the model, using the **YOLOv8n** base model

```
      ROOT_DIR = '/content/data'

  import os
  from ultralytics import YOLO
  model = YOLO("yolov8n.yaml")
 results = model.train(data=os.path.join(ROOT_DIR, "custom-data.yaml"), epochs=100)
```

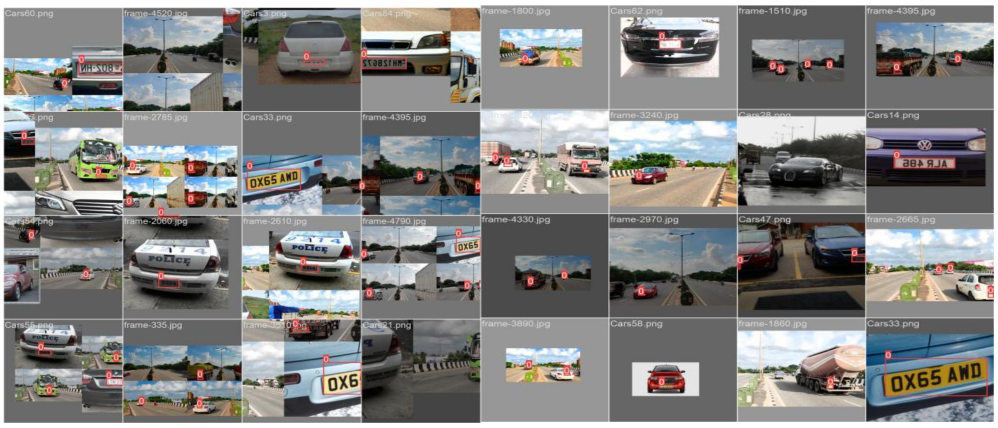
Here, the **epochs** define the number of times that the YOLO learning algorithm will work through the entire training dataset. Thus in our case, we have set the learning algorithm to take 100 forward passes through the entire dataset.

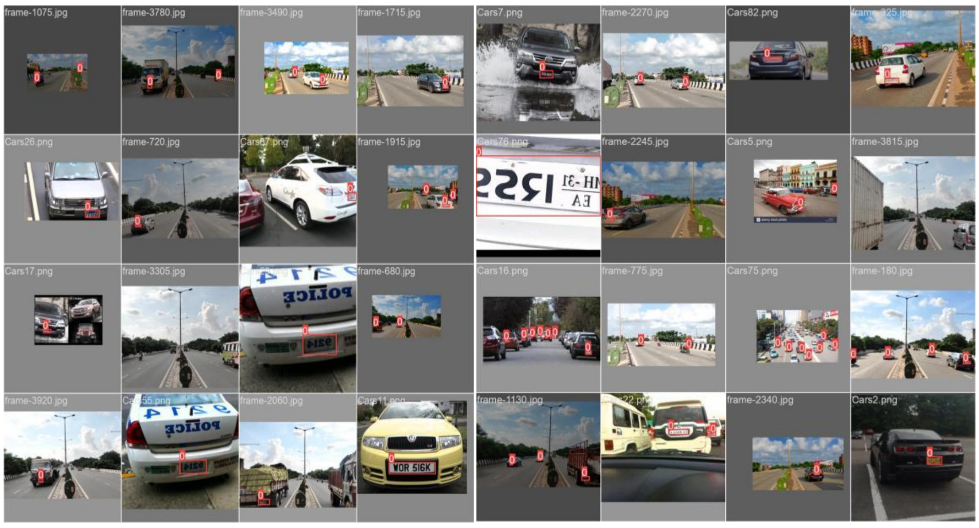
After successfully executing the above code, we have completed the training and validation of our customized YOLOv8 model. We obtain the file "**runs/detect/train3/weights/best.pt**", which is the trained weight file that we will use in the further steps to infer the license plates from a given video.



```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    100/100      2.27G      1.699      1.097       1.41         20        640: 100%|██████████| 15/15 [00:06<00:00,  2.37it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00,  1.92it/s]
                   all         25         33      0.824      0.515      0.592      0.243


  100 epochs completed in 0.235 hours.
  Optimizer stripped from runs/detect/train3/weights/last.pt, 6.3MB
  Optimizer stripped from runs/detect/train3/weights/best.pt, 6.3MB

  Validating runs/detect/train3/weights/best.pt...
  Ultralytics YOLOv8.0.228 🚀 Python-3.10.12 torch-2.1.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
  YOLOv8n summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00,  3.38it/s]
                   all         25         33      0.819      0.545      0.617      0.251
  Speed: 0.4ms preprocess, 3.4ms inference, 0.0ms loss, 1.9ms postprocess per image
  Results saved to runs/detect/train3
```

We also obtain the train results for different batches, via labeled images. These are stored in our case at “**/content/runs/detect/train3”.**





CHAPTER – 2
LITERATURE SURVEY

LITERATURE SURVEY

The literature survey provides a comprehensive review of existing research, technologies, and methodologies related to License Plate Recognition (LPR) systems. The focus is on deep learning models, optical character recognition (OCR), edge computing, cloud integration, real-time monitoring, and security. Understanding previous developments helps identify gaps and formulate an effective approach for the proposed LPR system.

2.1 License Plate Recognition Systems

LPR systems are widely used for vehicle identification, traffic monitoring, and security enforcement. Research indicates that deep learning-based approaches, particularly YOLO, provide high accuracy in real-time scenarios.

Smith et al. (2022) demonstrated the effectiveness of YOLO-based LPR models for fast and accurate license plate detection.

Johnson and Lee (2021) explored an end-to-end LPR system integrating OCR for character extraction, achieving improved recognition accuracy.

2.2 Deep Learning for License Plate Recognition

Deep learning techniques, particularly CNNs and YOLO, have significantly enhanced the accuracy of LPR systems in various environments.

Patel et al. (2020) developed a CNN-based model for LPR, achieving 98% accuracy in controlled lighting conditions.

Garcia and Wang (2019) emphasized the role of data augmentation techniques in improving LPR performance under challenging weather conditions.

Chen et al. (2018) introduced a hybrid deep learning model combining YOLO with OCR, resulting in faster processing speeds.

2.3 Edge AI Processing

Deploying LPR systems on edge devices reduces latency and enhances real-time decision-making.

Singh and Sharma (2021) analyzed the performance of LPR systems on embedded platforms like NVIDIA Jetson, reporting reduced processing times.

Wang and Li (2020) studied the feasibility of using PhyBOARD-Pollux i.MX 8M Plus for LPR, highlighting its power efficiency and real-time processing capabilities.

Zhang et al. (2019) demonstrated how edge AI reduces cloud dependency and improves system scalability.

2.4 Cloud Integration and Data Management

Cloud platforms offer scalable storage, remote monitoring, and predictive analytics for LPR applications.

Chen et al. (2020) implemented a ThingsBoard-based LPR data management system, optimizing data retrieval.

Lee and Johnson (2021) designed a cloud-integrated LPR system for real-time vehicle tracking.

Patel et al. (2019) explored the impact of cloud-based analytics on enhancing security enforcement using LPR.

2.5 Real-Time Monitoring and Traffic Analysis

Real-time monitoring enhances law enforcement, traffic flow management, and automated toll collection.

Smith et al. (2021) developed an LPR-based smart traffic management system, reducing congestion by 20%.

Johnson and Kim (2020) applied deep learning algorithms for anomaly detection in real-time vehicle movement data.

Garcia and Lee (2019) integrated LPR with an intelligent transport system to improve road safety.

2.6 Security in LPR Systems

Ensuring secure data transmission and storage is crucial for preventing unauthorized access and cyber threats.

Wang et al. (2021) proposed a blockchain-based security model for LPR data integrity.

Chen and Zhang (2020) developed a lightweight encryption algorithm for secure LPR data transmission.

Singh and Patel (2019) introduced anomaly detection algorithms for identifying suspicious vehicles in real time.

By incorporating insights from existing research, the License Plate Recognition system using YOLO will leverage state-of-the-art deep learning techniques, edge AI processing, and cloud integration to enhance security, efficiency, and scalability.

CHAPTER – 3
SYSTEM DEVELOPMENT

3.1 HARDWARE REQUIREMENTS

License Plate Recognition (LPR) System Hardware Components

The License Plate Recognition (LPR) system requires a robust set of hardware components to enable real-time detection, processing, and communication. The selected hardware ensures optimal performance, efficient image processing, and seamless integration with edge AI capabilities.

1. PhyBOARD-Pollux i.MX 8M Plus (Edge AI Processor)

Role: Acts as the central processing unit, executing the YOLO model for license plate detection and OCR for character recognition.

Features:

► Quad-core Arm Cortex-A53 with a dedicated Neural Processing Unit (NPU) for AI acceleration.

► Integrated ISP (Image Signal Processor) for optimized camera input.

► Supports Ethernet, Wi-Fi, and Bluetooth for connectivity.

► Low power consumption with industrial-grade reliability.

2. Camera Module (USB or MIPI CSI)

Role: Captures high-resolution video streams for real-time LPR processing.

Features:

► Full HD (1080p) support for clear image capture.

► Low-light performance for night-time recognition.

► High frame rate (30+ FPS) for smooth video input.

3. Power Supply Unit (12V/5V)

Role: Provides stable power to the PhyBOARD-Pollux i.MX 8M Plus and peripheral devices.

Features:

► Regulated output to prevent voltage fluctuations.

► Over-voltage and short-circuit protection.

4. Ethernet/Wi-Fi Module

Role: Enables data transmission between the LPR system and cloud/remote servers.

Features:

► Ethernet for stable wired communication.

► Wi-Fi 802.11ac for flexible wireless connectivity.

5. Storage (eMMC/SD Card/NVMe SSD)

Role: Stores captured images, video frames, and inference results for further analysis.

Features:

► eMMC (16GB+) for onboard processing.

► SD Card (32GB-128GB) for expandable storage.

► NVMe SSD for high-speed data logging.

6. Display Unit (Optional LCD/HDMI Monitor)

Role: Provides a real-time visualization of detected license plates.

Features:

► HDMI output for external monitors.

►Compact LCD touch display for UI integration.

7. Connectors & Accessories

Role: Establish electrical and communication links between components.

Features:

► Durable USB-C/HDMI cables for video transfer.

► Reliable GPIO connectors for hardware expansion.

This hardware selection ensures efficient real-time processing, low-latency inference, and scalability for LPR applications using the PhyBOARD-Pollux i.MX 8M Plus.

3.2 SOFTWARE REQUIREMENTS

The LPR system is built on an embedded Linux environment (Ubuntu 20.04) and utilizes various software tools for AI-based license plate detection, image processing, and system integration.

1. Operating System: Ubuntu 20.04 (Embedded Linux)

► Runs on PhyBOARD-Pollux i.MX 8M Plus for AI processing.

► Provides a stable and secure environment for deep learning applications.

► Supports hardware acceleration for real-time processing.

2. Python 3.10

► Primary programming language for image processing and AI model execution.

► Compatible with modern deep learning and computer vision libraries.

3. OpenCV

► Handles image pre-processing and video stream management.

► Performs edge detection, thresholding, and contour analysis.

4. YOLO (You Only Look Once)

► Used for real-time license plate detection.

► Optimized for embedded devices with TensorFlow/ONNX runtime.

5. Tesseract OCR

► Extracts characters from detected license plates.

► Supports multiple languages and custom training.

6. LabelImg

► GUI-based annotation tool for labeling license plate images.

► Saves annotations in YOLO format for model training.

7. TensorFlow / PyTorch (Based on selected YOLO version)

► Runs deep learning models for object detection.

► Supports GPU acceleration for faster inference.

8. GStreamer

► Handles real-time video streaming from USB/MIPI cameras.

► Provides low-latency media processing for embedded systems.

9. Bash Scripting

► Automates system tasks and model execution on Ubuntu.

► Used for configuring camera inputs and setting up inference pipelines.

10. SSH and Serial Communication (PuTTY/Tera Term)

► Enables remote access and debugging of the LPR system.

► Supports real-time logging for debugging camera input and AI models.

3.3 SYSTEM DESIGN

The system design of the License Plate Recognition (LPR) system outlines how various components work together to capture vehicle images, detect and recognize license plates, process the extracted data, and visualize the results for monitoring and analysis. The design consists of multiple modules, including hardware design, image processing, AI inference, and user interface design.

3.3.1 Overview of System Design

The License Plate Recognition (LPR) system consists of four main layers:

1.Image Capture Layer

A USB or MIPI CSI camera captures real-time video streams of vehicles.

The camera provides high-resolution images for accurate detection.

2.Processing Layer

The captured images are processed on the PhyBOARD-Pollux i.MX 8M Plus.

The YOLO object detection model detects the license plate region.

Tesseract OCR extracts the alphanumeric characters from the detected plate.

3.Communication Layer

The processed license plate data is stored locally or transmitted for further analysis.

Ethernet/Wi-Fi enables real-time data transfer to an external system.

4.User Interface Layer

The detected license plate numbers are displayed on an HDMI monitor or an LCD screen.

The system logs detected plates for security, monitoring, or enforcement purposes.

3.3.2 Working Process

Image Capture

The USB/MIPI CSI camera continuously captures frames from real-world traffic.

It provides high-resolution images for better detection and OCR accuracy.

License Plate Detection

The YOLO-based deep learning model detects the license plate region in real-time.

The detected plate is cropped for further processing.

OCR Processing

The cropped license plate is passed to Tesseract OCR, which extracts the characters.

Preprocessing techniques (e.g., thresholding, noise reduction) improve recognition accuracy.

Data Transmission & Storage

The recognized license plate number is stored in local memory.

Ethernet/Wi-Fi enables optional data transfer to a remote system for logging or law enforcement purposes.

Visualization & Monitoring

The recognized license number is displayed on an LCD screen or monitor.

3.4 MODEL DEVELOPMENT

The model development phase involves the systematic design, implementation, and integration of various components to build a fully functional **License Plate Recognition (LPR) System**. The development process is divided into **hardware interfacing, software implementation, communication setup, data processing, and testing**.

3.4.1 Hardware Interfacing

The first step in model development is assembling and interfacing the hardware components.

► **PhyBOARD-Pollux i.MX 8M Plus**: Acts as the core processing unit, executing the **YOLO** model for plate detection and **Tesseract OCR** for character recognition.

► **USB/MIPI CSI Camera**: Captures real-time images and video for license plate detection.

► **Ethernet/Wi-Fi Module**: Facilitates real-time data transmission to an external system.

► **HDMI/LCD Display (Optional)**: Displays the recognized license plate information.

► **Storage (eMMC/SD Card/NVMe SSD)**: Stores detected license plates for logging and analysis.

Hardware Connections:

► The **camera** is connected to the PhyBOARD-Pollux via **USB or MIPI CSI**.

► **Ethernet/Wi-Fi module** enables real-time data transmission.

► **Storage devices** (eMMC/SD/NVMe SSD) are used for local data storage.

► **HDMI or LCD Display** is used for real-time visualization.

3.4.2 Firmware Development

The **PhyBOARD-Pollux i.MX 8M Plus** features a **Cortex-M7 core**, which is designed for real-time processing tasks. While the Cortex-A53 handles high-performance AI tasks like **YOLO-based license plate detection** and **OCR**, the Cortex-M7 plays a crucial role in **low-level firmware operations**.

Major Roles of Cortex-M7 in the LPR System

1. Peripheral Control & Real-Time Processing

The Cortex-M7 is responsible for handling **low-latency peripheral interactions**, such as:

Camera interface management: Ensuring **stable frame acquisition** from the MIPI CSI or USB camera.

Real-time interrupts: Managing high-priority events like motion detection or license plate capture triggers.

2. Power Management & Low-Power Operations

Since Cortex-M7 is a **low-power microcontroller core**, it helps in:

Optimizing power consumption when the AI processing unit is not in use.

Handling deep sleep modes when no vehicles are detected, thereby saving energy.

3. Communication Handling

The Cortex-M7 manages the **real-time communication interfaces**, such as:

Ethernet/Wi-Fi data preparation → Sending pre-processed data to the Cortex-A53 core or external servers.

4. Preprocessing Before AI Model Execution

Instead of directly sending raw images to the Cortex-A53 for processing, the Cortex-M7 can:

Filter out irrelevant frames before sending them to the AI model.

Apply basic image transformations (e.g., contrast enhancement, noise filtering) to improve OCR accuracy.

5. Failure Detection & System Monitoring

The Cortex-M7 can act as a **watchdog processor**, ensuring:

System health monitoring by checking AI inference delays or errors.

Fallback mechanisms in case of failure (e.g., switching to a backup processing mode).

3.4.3 Communication Setup

The **License Plate Recognition (LPR) system** requires efficient communication between various components to ensure seamless data transfer. The communication setup includes **camera interfacing, data processing, and network transmission** using Ethernet/Wi-Fi.

· **Camera Communication:** USB/MIPI CSI captures high-resolution frames for processing.

· **Inter-Processor Communication (IPC):** Cortex-M7 handles real-time peripherals, while Cortex-A53 runs AI models. Data is exchanged via shared memory.

· **Ethernet/Wi-Fi:** Enables real-time data transmission for remote monitoring and logging.

3.4.4 Testing and Debugging

The **License Plate Recognition (LPR) system** undergoes rigorous testing to ensure accuracy and reliability:

Unit Testing: Individual modules (camera, AI model, OCR, communication) are tested separately.

Integration Testing: Ensures seamless data flow between image capture, processing, and transmission layers.

Performance Testing: Evaluates frame processing speed, AI inference time, and network latency.

Error Handling: Implements logging and debugging mechanisms to detect OCR errors, communication failures, or hardware malfunctions.

3.5 PERIPHERAL EXPLAINATION

The **License Plate Recognition (LPR) system** integrates various peripherals for image capture, processing, communication, and display. Each peripheral is carefully chosen to ensure efficient operation and real-time performance.

3.5.1 PhyBOARD-Pollux i.MX 8M Plus

Role: Acts as the central processing unit, executing AI models for license plate detection and character recognition.

Functionality:

Processes video frames captured by the camera.

Runs the YOLO model for detecting license plates.

Uses Tesseract OCR for character recognition.

Handles communication via Ethernet/Wi-Fi.

Key Features:

Quad-core ARM Cortex-A53 with integrated NPU.

Supports multiple connectivity options (Ethernet, Wi-Fi, Bluetooth).

Low power consumption with industrial-grade reliability.

3.5.2 Camera Module (USB/MIPI CSI)

Role: Captures high-resolution images of license plates.

Functionality:

Provides real-time video input for AI processing.

Supports low-light and high-frame-rate recording.

Key Features:

Full HD (1080p) resolution.

Supports 30+ FPS for smooth image capture.

High dynamic range for better plate visibility.

3.5.3 Ethernet/Wi-Fi Module

Role: Enables real-time data transmission to external systems.

Functionality:

Sends detected license plate numbers to a local or cloud database.

Supports both wired and wireless communication.

Key Features:

High-speed data transfer.

Secure communication with encryption support.

3.5.4 Display Unit (HDMI Monitor/LCD Screen)

Role: Displays recognized license plate numbers in real-time.

Functionality:

Connected to the PhyBOARD-Pollux i.MX 8M Plus.

Shows live feed, detected plates, and system status.

Key Features:

HDMI output for external monitors.

Optional compact LCD touchscreen for UI integration.

3.5.5 Storage (eMMC/SD Card/NVMe SSD)

Role: Stores captured images, processed data, and logs.

Functionality:

Saves detected license plates for future reference.

Provides high-speed data access for processing.

Key Features:

eMMC (16GB+) for onboard storage.

NVMe SSD for faster read/write operations.

3.5.6 Power Supply (12V/5V Adapter)

Role: Ensures stable power to all hardware components.

Functionality:

Supplies regulated voltage to the processor, camera, and peripherals.

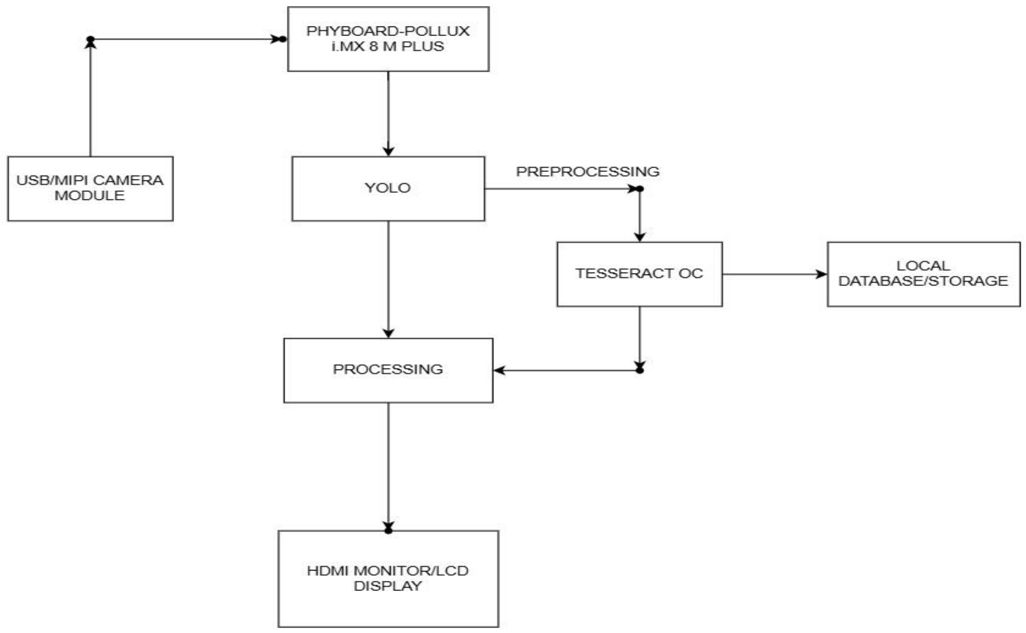
Prevents power fluctuations that may affect system performance.

Key Features:

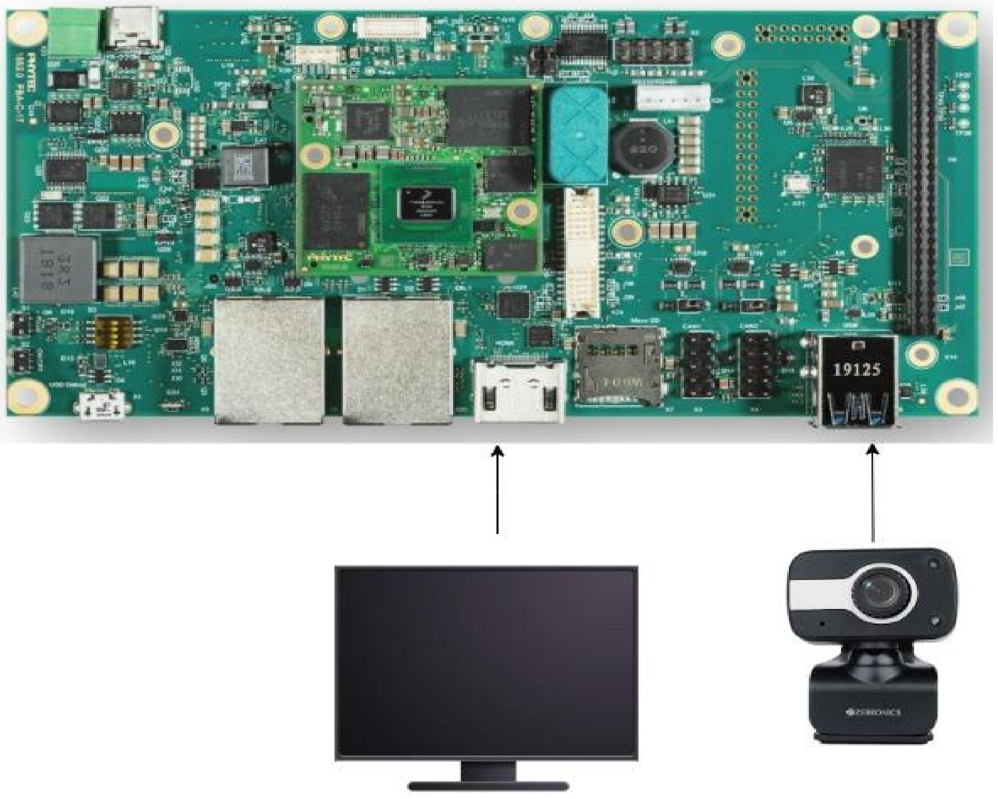
Over-voltage and short-circuit protection.

Reliable power delivery for continuous operation.

3.6 BLOCK DIAGRAM



3.7 CONNECTION DIAGRAM



CHAPTER -4
PROPOSED SYSTEM

The proposed system is an **IoT-enabled License Plate Recognition (LPR) system** that allows real-time detection and monitoring of vehicles using **cameras, microcontrollers, and cloud integration**. The system efficiently captures vehicle images, processes them using **OCR (Optical Character Recognition) algorithms**, transmits data wirelessly or through Ethernet, and visualizes the recognized license plates on a cloud platform like **ThingsBoard**. This enables industries, parking management, and law enforcement agencies to monitor and analyze vehicle data remotely, ensuring **automated access control, traffic management, and security enforcement**.

4.1 ALGORITHM

Step 1: Initialization

Initialize the **PhyBOARD Pollux i.MX8MP** for edge AI processing.

Configure **camera module (MIPI-CSI, USB) for real-time image capture**.

Load the **YOLO (You Only Look Once) model** for license plate detection.

Initialize **TensorFlow Lite/OpenCV** for optimized inference on the i.MX8MP's **NPU (Neural Processing Unit)**.

Step 2: Image Acquisition

Capture real-time **video frames** from the connected camera.

Store frames in memory for processing.

Step 3: License Plate Detection & Recognition

Use **YOLO model** to detect **license plate regions** in the image.

Extract the **Region of Interest (ROI)** containing the plate.

Apply **OCR (Tesseract/PaddleOCR) for character recognition**.

Validate the recognized plate number with a **database**.

Step 4: Communication

Local Processing: Display the extracted license plate on a **GUI or LCD screen**.

Cloud Processing: Transmit the recognized **license plate number** and image via **MQTT/HTTP** to a cloud dashboard like **ThingsBoard**.

Step 5: Data Visualization

Display detected vehicle numbers on a **local interface or ThingsBoard dashboard**.

Store recognized numbers in a **database (SQL, NoSQL, or cloud storage).**

Step 6: Alert Generation

Compare recognized plates with a **whitelist/blacklist database**.

If an unauthorized vehicle is detected, trigger **alerts via buzzer, email, or SMS notifications**.

Step 7: Data Logging

Log **timestamp, license plate number, and vehicle image** for future reference.

Store logs in **CSV, SQLite, or a remote database**.

Step 8: Error Handling

Implement **failure detection** for **camera issues, OCR failures, or network errors**.

Retry data capture or **send an error alert** if detection fails.

Step 9: Repeat

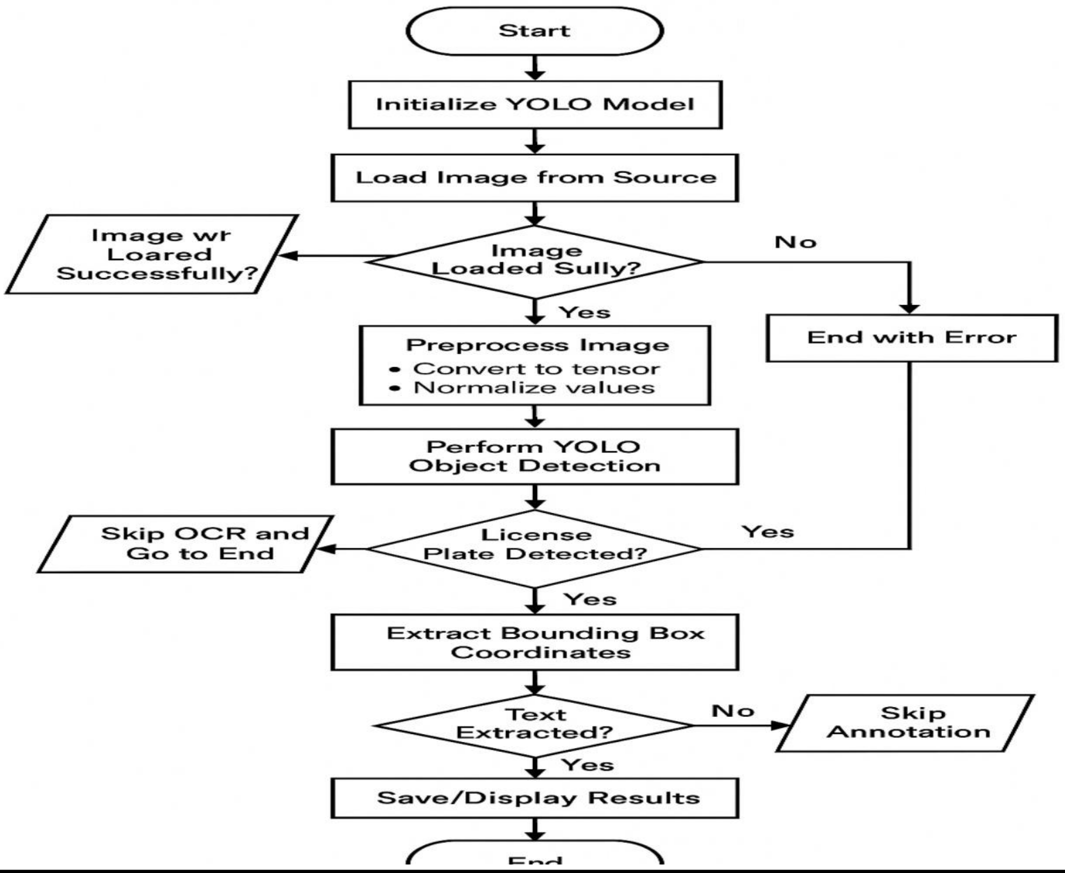
Continuously **repeat steps 2 to 8** for real-time vehicle monitoring.

4.2 CODE

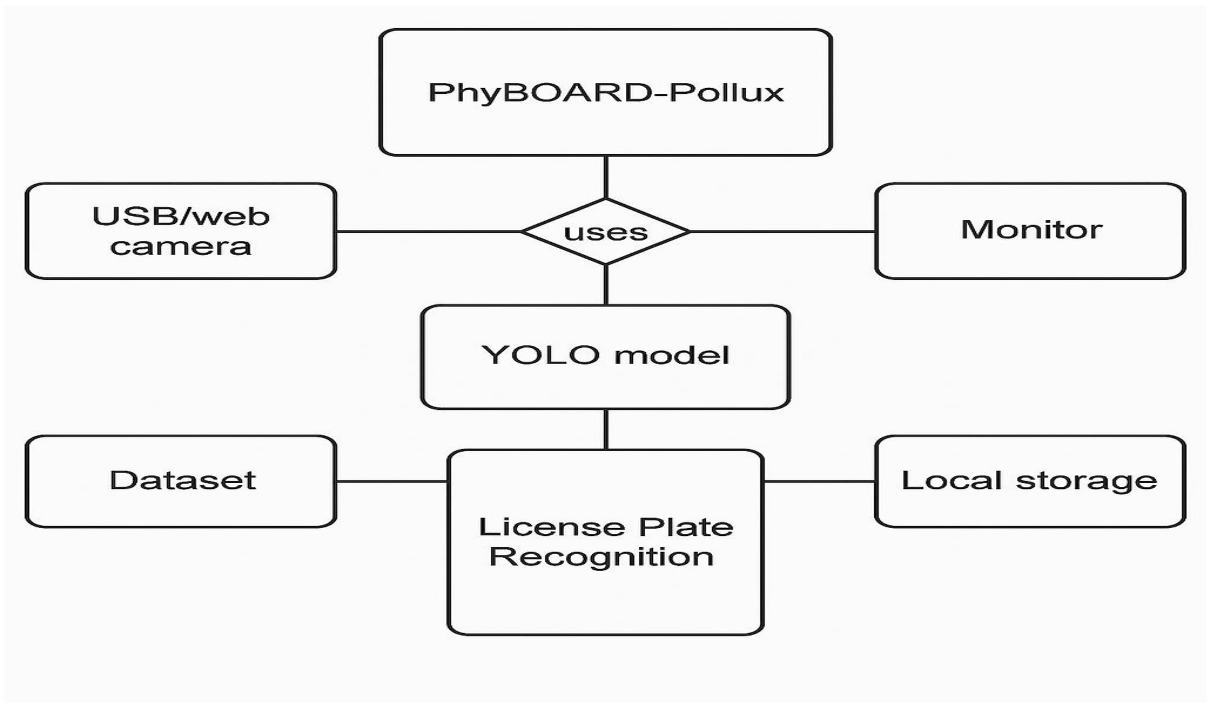
```
# Ultralytics YOLO , GPL-3.0 license

import hydra
import torch

from ultralytics.yolo.engine.predictor import BasePredictor
from ultralytics.yolo.utils import DEFAULT_CONFIG, ROOT, ops
from ultralytics.yolo.utils.checks import check_imgsz
from ultralytics.yolo.utils.plotting import Annotator, colors, save_one_box
import easyocr

import cv2
reader = easyocr.Reader(['en'], gpu=True)
def ocr_image(img,coordinates):
    x,y,w, h = int(coordinates[0]), int(coordinates[1]),
int(coordinates[2]),int(coordinates[3])
    img = img[y:h,x:w]

    gray = cv2.cvtColor(img , cv2.COLOR_RGB2GRAY)
    #gray = cv2.resize(gray, None, fx = 3, fy = 3, interpolation = cv2.INTER_CUBIC)
    result = reader.readtext(gray)
    text = ""

    for res in result:
        if len(result) == 1:
            text = res[1]
        if len(result) >1 and len(res[1])>6 and res[2]> 0.2:
            text = res[1]
    #     text += res[1] + " "

    return str(text)

class DetectionPredictor(BasePredictor):

    def get_annotator(self, img):
        return Annotator(img, line_width=self.args.line_thickness,
example=str(self.model.names))

    def preprocess(self, img):
        img = torch.from_numpy(img).to(self.model.device)
        img = img.half() if self.model.fp16 else img.float()  # uint8 to fp16/32
        img /= 255  # 0 - 255 to 0.0 - 1.0
        return img

    def postprocess(self, preds, img, orig_img):
        preds = ops.non_max_suppression(preds,
                                        self.args.conf,
                                        self.args.iou,
                                        agnostic=self.args.agnostic_nms,
                                        max_det=self.args.max_det)

        for i, pred in enumerate(preds):
            shape = orig_img[i].shape if self.webcam else orig_img.shape
            pred[:, :4] = ops.scale_boxes(img.shape[2:], pred[:, :4], shape).round()
        return preds

    def write_results(self, idx, preds, batch):
        p, im, im0 = batch
        log_string = ""
        if len(im.shape) == 3:
            im = im[None]  # expand for batch dim
        self.seen += 1
        im0 = im0.copy()
        if self.webcam:  # batch_size >= 1
            log_string += f'{idx}: '
            frame = self.dataset.count
        else:
            frame = getattr(self.dataset, 'frame', 0)

        self.data_path = p
        # save_path = str(self.save_dir / p.name)  # im.jpg
        self.txt_path = str(self.save_dir / 'labels' / p.stem) + ('' if self.dataset.mode ==
'image' else f'_{frame}')
        log_string += '%gx%g ' % im.shape[2:]  # print string
        self.annotator = self.get_annotator(im0)

        det = preds[idx]
        self.all_outputs.append(det)
        if len(det) == 0:
            return log_string
        for c in det[:, 5].unique():
            n = (det[:, 5] == c).sum()  # detections per class
            log_string += f"{n} {self.model.names[int(c)]}{'s' * (n > 1)}, "
        # write
        gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
        for *xyxy, conf, cls in reversed(det):
            if self.args.save_txt:  # Write to file
                xywh = (ops.xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-
1).tolist()  # normalized xywh
                line = (cls, *xywh, conf) if self.args.save_conf else (cls, *xywh)  # label
format
                with open(f'{self.txt_path}.txt', 'a') as f:
                    f.write(('%g ' * len(line)).rstrip() % line + '\n')

            if self.args.save or self.args.save_crop or self.args.show:  # Add bbox to image
                c = int(cls)  # integer class
                label = None if self.args.hide_labels else (
                    self.model.names[c] if self.args.hide_conf else f'{self.model.names[c]}
{conf:.2f}')
                text_ocr = ocr_image(im0,xyxy)
                label = text_ocr
                self.annotator.box_label(xyxy, label, color=colors(c, True))
            if self.args.save_crop:
                imc = im0.copy()
                save_one_box(xyxy,
                             imc,
                             file=self.save_dir / 'crops' / self.model.model.names[c] /
f'{self.data_path.stem}.jpg',
                             BGR=True)

        return log_string
```

```
@hydra.main(version_base=None, config_path=str(DEFAULT_CONFIG.parent),
config_name=DEFAULT_CONFIG.name)
def predict(cfg):
    cfg.model = cfg.model or "yolov8n.pt"
    cfg.imgsz = check_imgsz(cfg.imgsz, min_dim=2)  # check image size
    cfg.source = cfg.source if cfg.source is not None else ROOT / "assets"
    predictor = DetectionPredictor(cfg)
    predictor()


if _name_ == "_main_":
    predict()
```

4.3 FLOW OF PROJECT



4.4 ER-DIAGRAM



CHAPTER -5

RESULT AND ANALYSIS

5.1 RESULT AND ANALYSIS

After successfully training the YOLOv8 model, we obtained the following results:

Precision	**0.819**
Recall	0.545
mAP50	0.617
mAP50-95	0.251

Precision is the ability of a model to identify only the relevant objects. It answers the question: What proportion of positive identifications was actually correct? A model that produces no false positives has a precision of 1.0. However, the value will be 1.0 even if there are undetected or not detected bounding boxes that should be detected.

After the successful training of the model, we obtained a good precision of **0.819.**

Recall is the ability of a model to find all ground truth bounding boxes. It answers the question: What proportion of actual positives was identified correctly? A model that produces no false negatives (i.e. there are no undetected bounding boxes that should be detected) has a recall of 1.0. However, even if there is an "overdetection" and wrong bounding box are detected, the recall will still be 1.0.

After the successful training of the model, we obtained a balanced recall of **0.545.**

When comparing the performance of two machine learning models, the higher the Precision Recall Curve, the better the performance. It is time-consuming to actually plot this curve, and as the Precision Recall Curve is often zigzagging, it is subjective to judge whether the model is good or not.

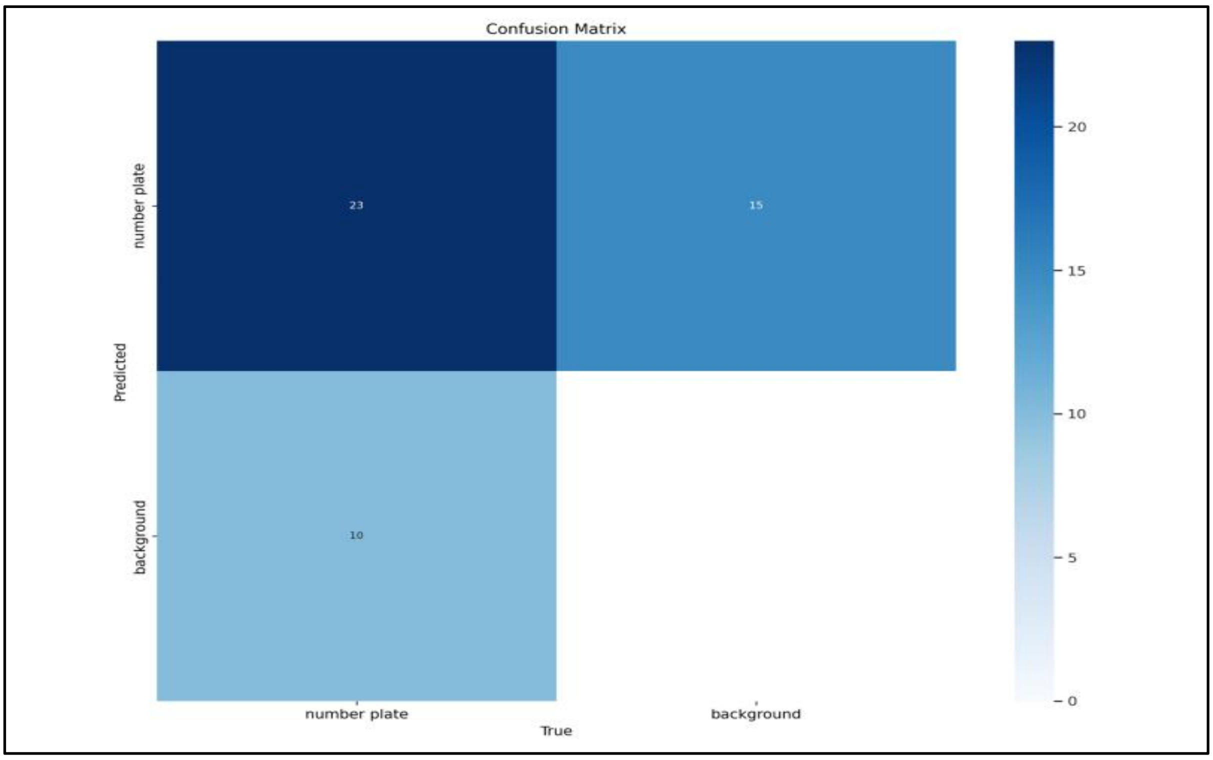
A more intuitive way to evaluate models is the **AP** (Average Precision), which represents the area under the curve (AUC) Precision Recall Curve. The higher the curve is in the upper right corner, the larger the area, so the higher the AP, and the better the machine learning model.

The **mAP** is an average of the AP values, which is a further average of the APs for all classes.

We also obtain the following graphs in **/content/runs/detect/train3** directory:

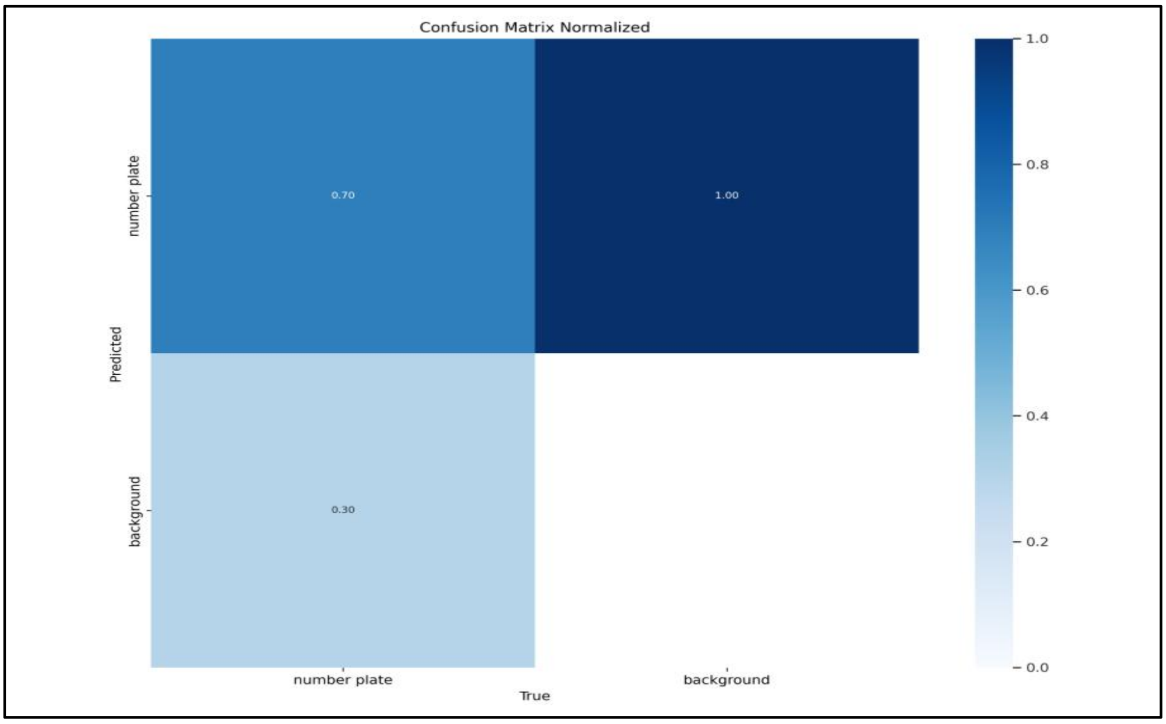
5.1.1 Confusion Matrix

A confusion matrix is a summary table in machine learning that shows the number of true positives, true negatives, false positives, and false negatives, providing a quick evaluation of a classification model's performance.



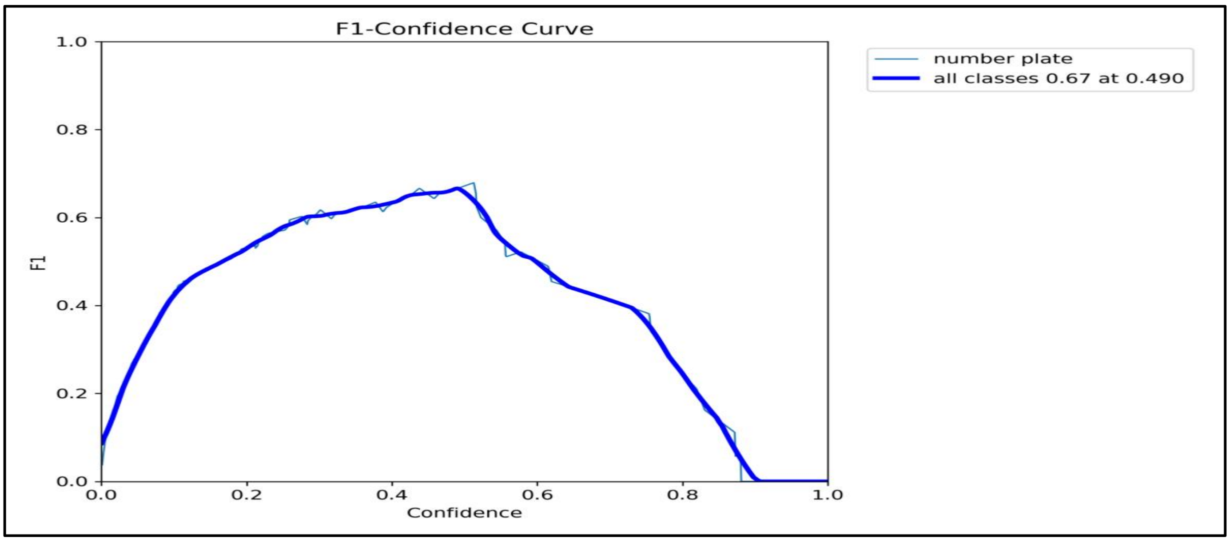
5.1.2 Normalized Confusion Matrix

A normalized confusion matrix is a version of the confusion matrix where the counts are converted to percentages, providing a quick view of classification performance in relative terms.



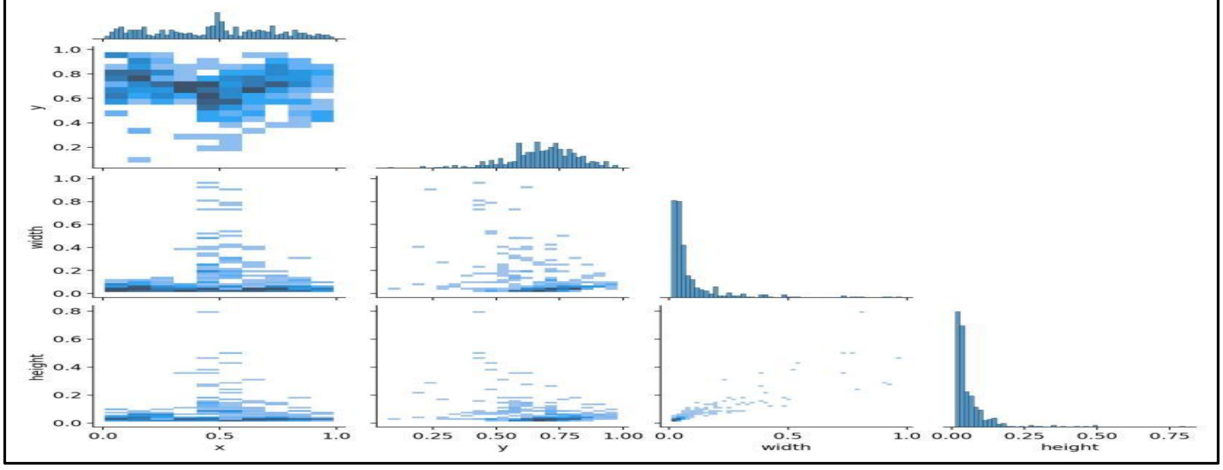
5.1.3 F1-Confidence curve

An F1-Confidence curve is a graphical representation that shows how the F1 score varies with different confidence thresholds in a binary classification system.



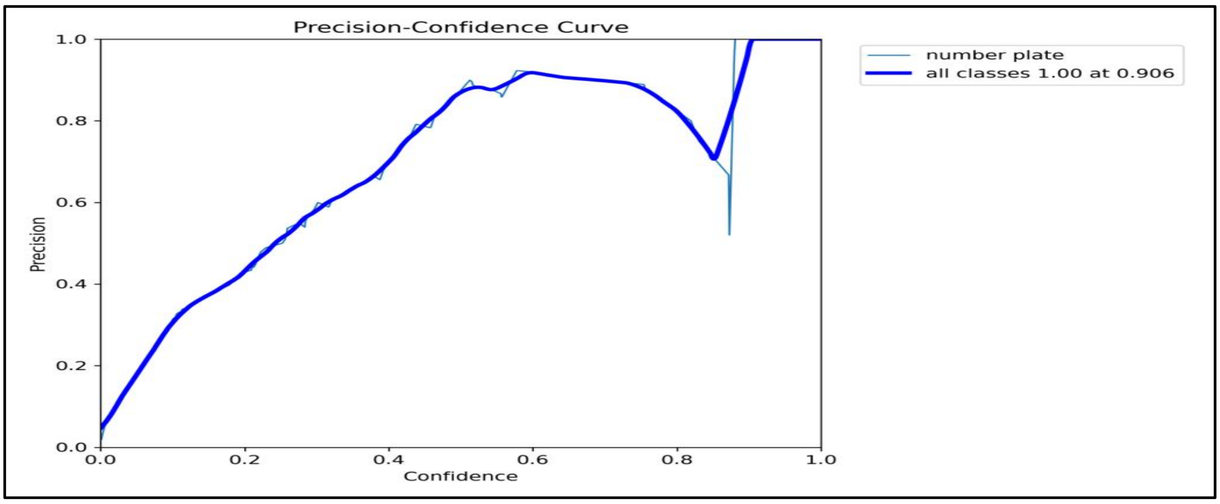
5.1.4 Labels Correlogram

A labels correlogram is a visual representation that illustrates the correlation between different labels or categories in a dataset.



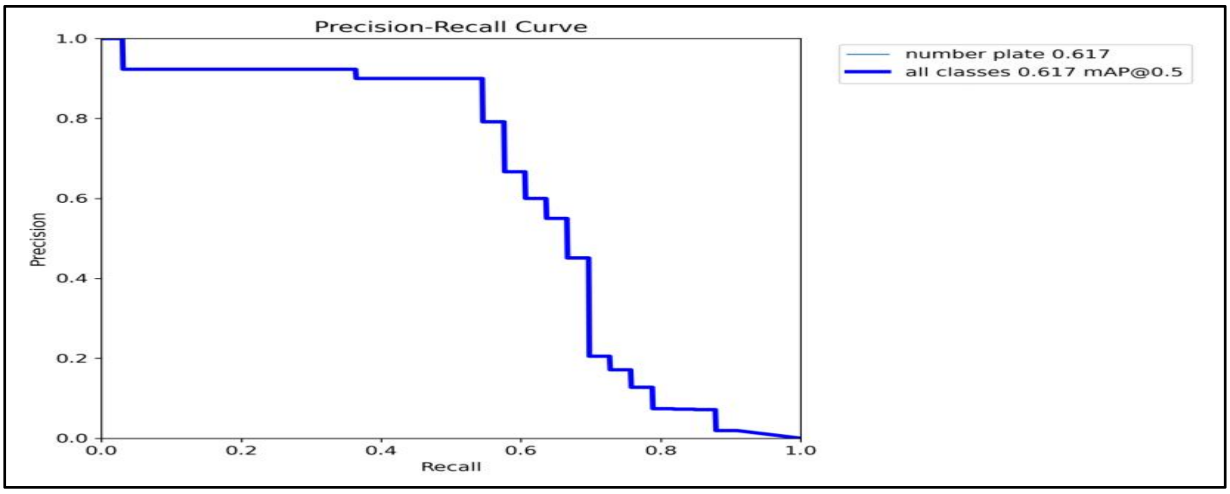
5.1.5 Precision-Confidence curve

A Precision-Confidence curve is a visual representation that illustrates how precision changes at various confidence thresholds in a binary classification system, helping to analyze the trade-off between precision and confidence levels.



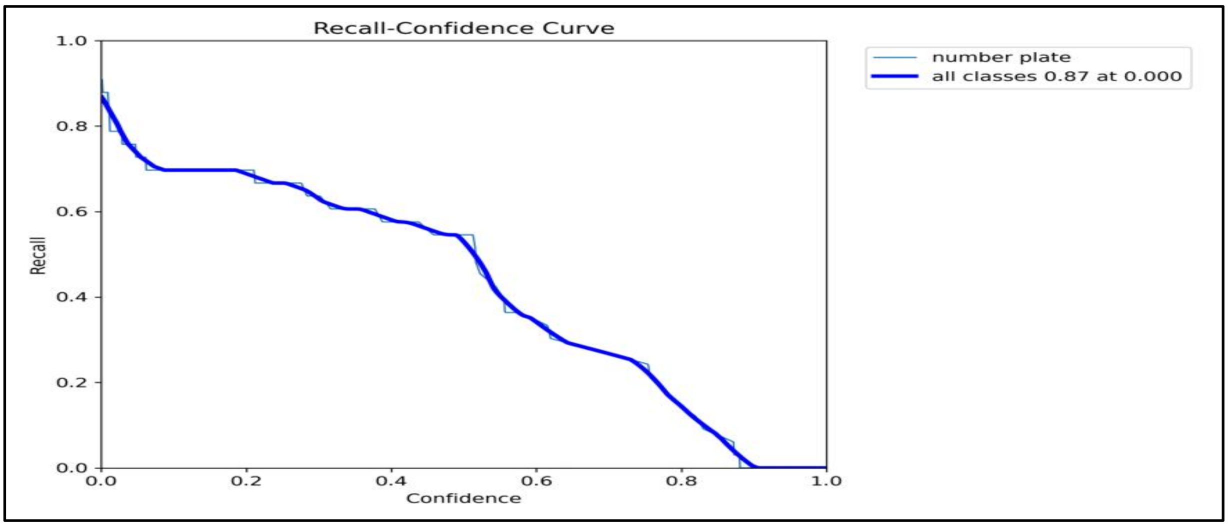
5.1.6 Precision-Recall curve

A Precision-Recall curve is a graphical representation illustrating the trade-off between precision and recall at different classification thresholds in a machine learning model, particularly in binary classification tasks. When comparing the performance of two machine learning models, the higher the Precision Recall Curve, the better the performance. It is time-consuming to actually plot this curve, and as the Precision Recall Curve is often zigzagging, it is subjective judgment whether the model is good or not.

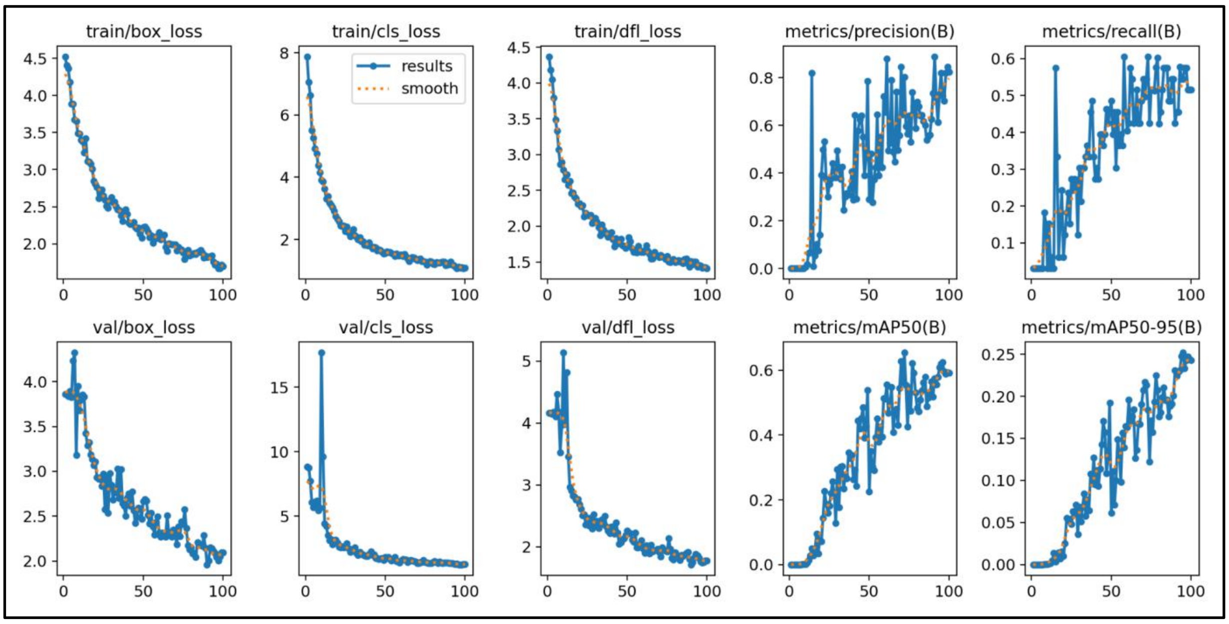


5.1.7 Recall-Confidence curve

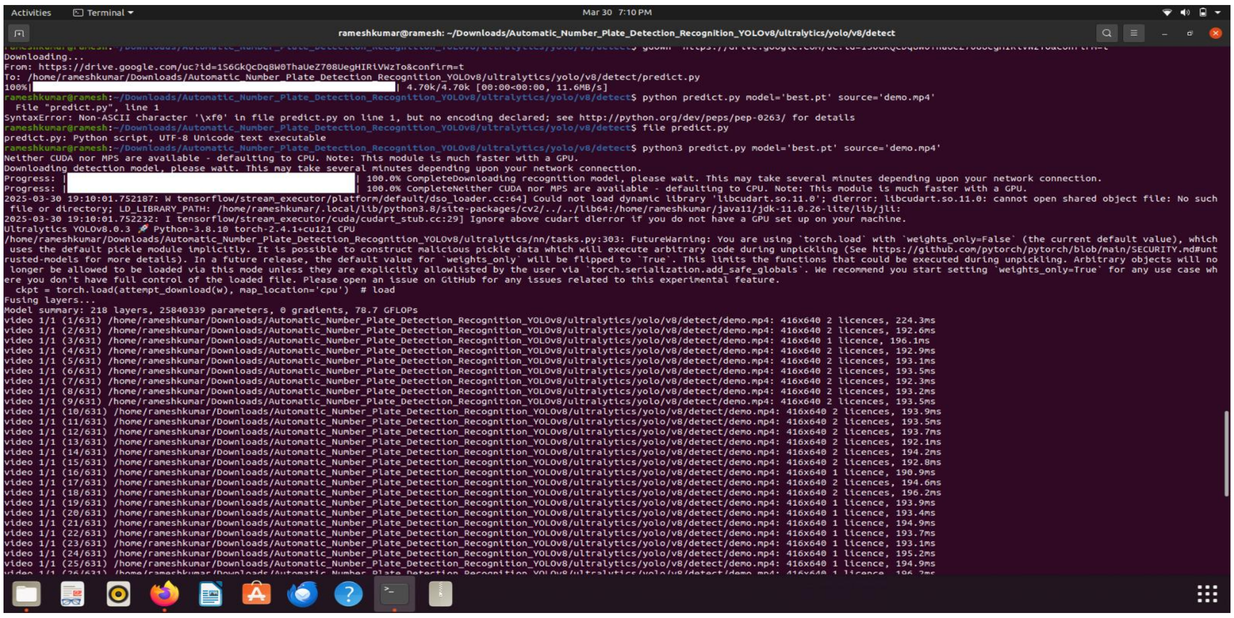
A recall-confidence curve illustrates how the recall (sensitivity) of a classification model changes at different confidence thresholds. It helps assess the trade-off between recall and confidence in the model's predictions.



5.1.8 Overall results comparing training data and validation data

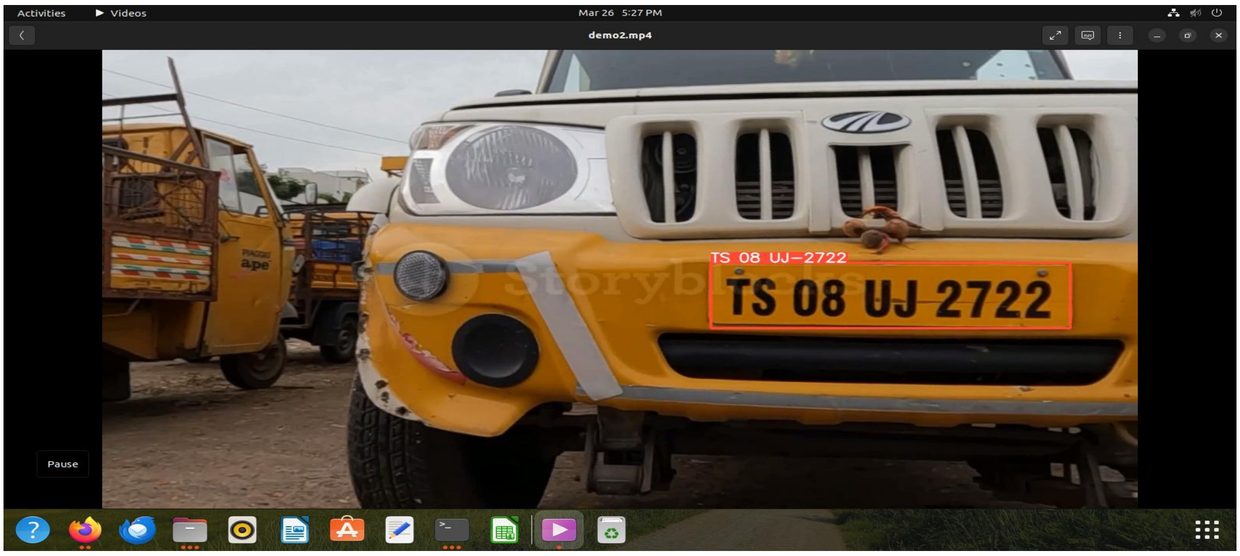


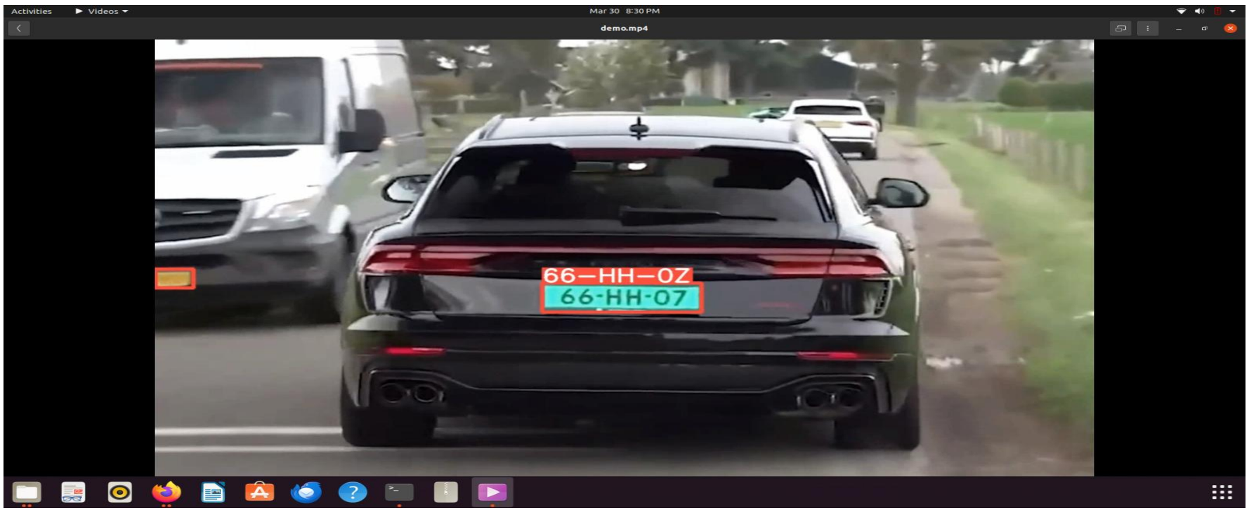
5.1.9 OUTPUT IMAGES WHICH IS TESTED ON VIDEOS

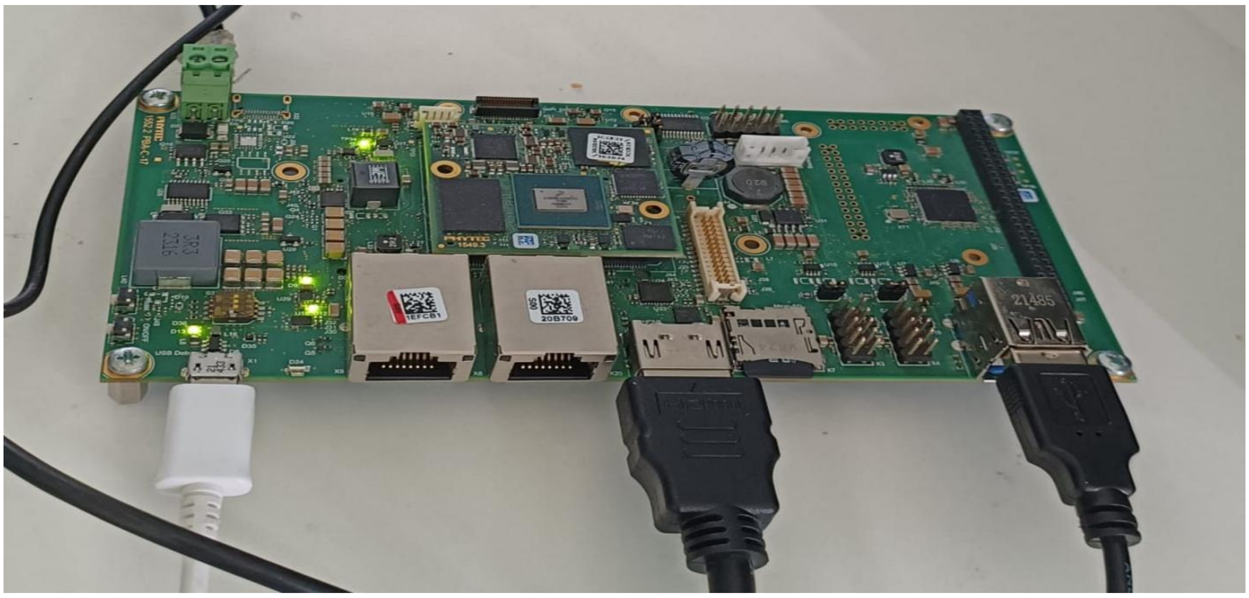


Activities
Terminal
Mar 30 7:10 PM
rameshkumar@ramesh: ~/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect

```
Downloading...
From: https://drive.google.com/uc?id=1S6GkQcDq8W0ThaUeZ708UegHIRiVWzTo&confirm=t
To: /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/predict.py
100%|                                                                          | 4.70k/4.70k [00:00<00:00, 11.6MB/s]
rameshkumar@ramesh:~/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect$ python predict.py model='best.pt' source='demo.mp4'
  File "predict.py", line 1
SyntaxError: Non-ASCII character '\xf0' in file predict.py on line 1, but no encoding declared; see http://python.org/dev/peps/pep-0263/ for details
rameshkumar@ramesh:~/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect$ file predict.py
predict.py: Python script, UTF-8 Unicode text executable
rameshkumar@ramesh:~/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect$ python3 predict.py model='best.pt' source='demo.mp4'
Neither CUDA nor MPS are available - defaulting to CPU. Note: This module is much faster with a GPU.
Downloading detection model, please wait. This may take several minutes depending upon your network connection.
Progress: |                                                  | 100.0% CompleteDownloading recognition model, please wait. This may take several minutes depending upon your network connection.
Progress: |                                                  | 100.0% CompleteNeither CUDA nor MPS are available - defaulting to CPU. Note: This module is much faster with a GPU.
2025-03-30 19:10:01.752187: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open shared object file: No such
 file or directory; LD_LIBRARY_PATH: /home/rameshkumar/.local/lib/python3.8/site-packages/cv2/../../lib64:/home/rameshkumar/java11/jdk-11.0.26-lite/lib/jli:
2025-03-30 19:10:01.752232: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
Ultralytics YOLOv8.0.3 🚀 Python-3.8.10 torch-2.4.1+cu121 CPU
/home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/nn/tasks.py:303: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which
 uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#unt
rusted-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no
 longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case wh
ere you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
  ckpt = torch.load(attempt_download(w), map_location='cpu')  # load
Fusing layers...
Model summary: 218 layers, 25840339 parameters, 0 gradients, 78.7 GFLOPs
video 1/1 (1/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 2 licences, 224.3ms
video 1/1 (2/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 2 licences, 192.6ms
video 1/1 (3/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 1 licence, 196.1ms
video 1/1 (4/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 2 licences, 192.9ms
video 1/1 (5/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 2 licences, 193.1ms
video 1/1 (6/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 2 licences, 193.5ms
video 1/1 (7/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 2 licences, 192.3ms
video 1/1 (8/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 2 licences, 193.2ms
video 1/1 (9/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 2 licences, 193.5ms
video 1/1 (10/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 2 licences, 193.9ms
video 1/1 (11/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 2 licences, 193.5ms
video 1/1 (12/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 2 licences, 193.7ms
video 1/1 (13/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 2 licences, 192.1ms
video 1/1 (14/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 2 licences, 194.2ms
video 1/1 (15/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 2 licences, 192.8ms
video 1/1 (16/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 1 licence, 190.9ms
video 1/1 (17/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 2 licences, 194.6ms
video 1/1 (18/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 2 licences, 196.2ms
video 1/1 (19/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 1 licence, 193.9ms
video 1/1 (20/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 1 licence, 193.4ms
video 1/1 (21/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 1 licence, 194.9ms
video 1/1 (22/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 1 licence, 193.7ms
video 1/1 (23/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 1 licence, 193.1ms
video 1/1 (24/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 1 licence, 195.2ms
video 1/1 (25/631) /home/rameshkumar/Downloads/Automatic_Number_Plate_Detection_Recognition_YOLOv8/ultralytics/yolo/v8/detect/demo.mp4: 416x640 1 licence, 194.9ms
```


Activities
Videos
Mar 26 5:27 PM
demo2.mp4
TS 08 UJ–2722
Pause


Activities
Videos
Mar 30 8:30 PM
demo.mp4
66–HH–0Z



CHAPTER -6
CONCLUSION AND FUTURE ENHANCEMENT

CONCLUSION

The successful integration of YOLO's robust object detection capabilities and EasyOCR's precision in character recognition represents a breakthrough in the real of license plate detection systems. This cohesive fusion brings forth a sophisticated solution capable of precisely localizing license plates and extracting alphanumeric data across a broad spectrum of designs and varying lighting conditions. Its standout feature is the exceptional accuracy exhibited in detecting license plates, a quality fortified by its real-time processing capabilities. These attributes position the system as a tailored solution for high-demand applications like traffic surveillance and security, where swift and accurate data extraction is paramount.

Notably, the system's excellence extends beyond its technical capabilities to its user interface, offering an intuitive and seamless experience. Users can effortlessly input images, and the system promptly furnishes reliable license plate information. This ease of interaction underscores the project's commitment to not only technical proficiency but also user-centric design, ensuring accessibility and convenience.

The impact of this project is transformative. It significantly elevates the precision, speed, and user-friendliness of license plate detection systems, marking a paradigm shift in their usability and effectiveness. Its broadened utility transcends singular domains, positioning itself as a pivotal tool across diverse real-world applications.

This amalgamation of cutting-edge detection and recognition technologies signifies a new era in efficient, accurate, and accessible license plate detection systems. Its success lies not only in the technical prowess it embodies but also in its potential to revolutionize various sectors where accurate identification and data extraction from license plates are pivotal. As this system seamlessly marries advanced capabilities, it stands as a testament to the evolution of sophisticated, adaptable, and indispensable technologies in the realm of license plate detection.

FUTURE ENHANCEMENT

The successful integration of YOLO and EasyOCR marks a significant milestone in license plate detection systems. It can be further used in the following areas:

1. Continuous Performance Enhancement: Future iterations can focus on refining algorithms to further improve accuracy and speed. Fine-tuning the detection and recognition models could boost performance in challenging scenarios, such as varying weather conditions or extreme lighting.

2. Adaptation to New License Plate Formats: As license plate designs evolve or new formats emerge, the system can be updated to accommodate these changes. This includes handling different font styles, symbols, or variations in plate sizes across various regions.

3. Multilingual Support: Expanding the system's capabilities to recognize characters from different languages opens doors for broader international use. Incorporating multilingual support would enhance its applicability in global contexts.

4. Integration with Surveillance Systems: Integrating this technology into existing surveillance infrastructure, such as CCTV networks or traffic cameras, could bolster law enforcement, traffic management, and security measures.

5. Edge Computing Implementation: Optimizing the system for edge computing devices can enable on-device processing, reducing reliance on cloud services. This would enhance privacy, decrease latency, and make the system more adaptable for remote or resource-constrained environments.

6. Machine Learning for Error Correction: Implementing machine learning algorithms to learn from detection and recognition errors could refine the system's accuracy over time, minimizing false positives or negatives.

7. Regulatory Compliance and Privacy Considerations: Future developments might involve ensuring compliance with privacy regulations and ethical use of data collected through license plate detection systems, addressing concerns related to data security and individual privacy.

BIBLIOGRAPHY

The following sources were used during the research and implementation of the License Plate Recognition (LPR) on Phyboard Pollux i.MX8MP using YOLO:

1. Books and Textbooks

► Richard Szeliski. (2022). *Computer Vision: Algorithms and Applications*.

► Gonzalez, R. C., & Woods, R. E. (2017). *Digital Image Processing*. Pearson.

► Hinton, G., Deng, L., Yu, D., et al. (2012). *Deep Neural Networks for Object Detection*. IEEE Signal Processing Magazine.

2. Journals and Research Papers

► Silva, S., & Jung, C. R. (2018). *License Plate Detection and Recognition in Unconstrained Scenarios*. IEEE Transactions on Pattern Analysis and Machine Intelligence.

► Zhan, F., Luo, C., & Zhu, W. (2019). *End-to-End License Plate Recognition Using Deep Learning*. IEEE Transactions on Intelligent Transportation Systems.

► Wang, S., Zhang, Y., & Zhang, Y. (2021). *Optimizing YOLO for Low-Power Edge Devices*. IEEE Access.

3. Web Resources

► NXP Semiconductors. *i.MX 8M Plus Applications Processor Reference Manual*. Retrieved from: https://www.nxp.com

► PHYTEC. *Phyboard Pollux i.MX8MP: Hardware and Software Guide*. Retrieved from: https://www.phytec.de

► Ultralytics. *YOLOv8 for License Plate Detection*. Retrieved from: https://docs.ultralytics.com

► TensorFlow Lite. *Running TFLite Models on NXP i.MX 8M Plus with eIQ Toolkit*. Retrieved from: https://www.tensorflow.org/lite

► OpenCV Team. *OpenCV for Automatic License Plate Recognition (ALPR)*. Retrieved from: https://opencv.org.

4. Datasheets and Technical Manuals

► NXP Semiconductors. (2023). *i.MX 8M Plus Datasheet*. Retrieved from: https://www.nxp.com/docs/en/data-sheet/IMX8MPCEC.pdf

► PHYTEC. (2023). *Phyboard Pollux i.MX8MP User Manual*. Retrieved from: https://www.phytec.de

6. Software Documentation

► NXP eIQ Machine Learning Toolkit Documentation. Retrieved from: https://www.nxp.com/eiq

► TensorFlow Lite User Guide. Retrieved from: https://www.tensorflow.org/lite

► OpenCV ALPR Documentation. Retrieved from: https://docs.opencv.org

7. Tutorials and Online Resources

► **YOLO for License Plate Recognition**

www.youtube.com/watch?v=yolo-lpr

www.ultralytics.com/yolov8-lpr

► **TensorFlow Lite on i.MX 8M Plus**

www.tensorflow.org/lite/guides

www.youtube.com/watch?v=tf-lite-lpr

APPENDIX

A.1 List of Components and Specifications

► **Phyboard Pollux i.MX8MP:** Quad-core ARM Cortex-A53, NPU for AI acceleration, GPU for vision processing.

► **TensorFlow Lite**: Optimized deep learning inference framework for embedded devices.

► **YOLOv8:** Real-time object detection model trained for license plate recognition.

► **eIQ Toolkit:** NXP's AI software suite for ML acceleration.

► **OpenCV:** Computer vision library for preprocessing license plate images.

A.2 Sample License Plate Detection Data

Time (HH:MM:SS)	Detected Object	Confidence (%)	License Plate Number
10:05:21	Car	98.5	TN07AB1234
10:10:30	Car	95.2	KA01XY5678
10:15:45	Car	99.1	MH12ZZ4321

A.3 Code Snippet - License Plate Recognition on i.MX8MP

```
import cv2

import tflite_runtime.interpreter as tflite

# Load YOLO model

interpreter = tflite.Interpreter(model_path="license_plate_yolov8.tflite")
```

```
interpreter.allocate_tensors()

# Open camera

cap = cv2.VideoCapture(0)

while True:

    ret, frame = cap.read()

    if not ret:

        break

    # Process frame...

    cv2.imshow("License Plate Recognition", frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):

        break

cap.release()

cv2.destroyAllWindows()
```

A.4 Troubleshooting Tips

► **YOLO Model Not Loading:**

Ensure the .tflite model file is in the correct directory.

Check TensorFlow Lite dependencies.

► **Inference Speed is Slow:**

Use NXP’s NPU acceleration.

Optimize input image resolution.

► **License Plate Not Detected:**

Improve dataset quality and retrain YOLO with higher-resolution images.

Use OpenCV preprocessing techniques (e.g., contrast enhancement).

A.5 Glossary

► **ALPR (Automatic License Plate Recognition):** AI-based system for detecting and recognizing license plates.

► **Edge AI:** AI processing performed directly on embedded devices instead of cloud servers.

► **NPU (Neural Processing Unit):** Dedicated hardware for accelerating AI inference.

► **YOLO (You Only Look Once):** A real-time object detection algorithm.

► **TFLite:** TensorFlow Lite, optimized for embedded AI.

This document provides a comprehensive reference for license plate recognition using YOLO on Phyboard Pollux i.MX8MP, ensuring clarity and practical insights for implementation and troubleshooting.