

Real-Time Motorbike Helmet Detection System Using Hybrid Deep Learning Model

by

**Jahida Alam Jaishee (163014038)
Aldrin Abir (163014011)
Irfanuddin Ahmed (182014030)**

*Capstone Project (CSE499) submitted in partial fulfillment of the
requirements for the degree of*

Bachelor of Science in Computer Science and Engineering

Under the supervision of

Dr. Mohammed Ashikur Rahman



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UNIVERSITY OF LIBERAL ARTS BANGLADESH**

SUMMER 2024

© *Jahida Alam Jaishee, Aldrin Abir and Irfanuddin Ahmed*
All rights reserved

DECLARATION

Project Title	Real-Time Motorbike Helmet Detection System Using Hybrid Deep Learning Model
Authors	<i>Jahida Alam Jaishee, Aldrin Abir and Irfanuddin Ahmed</i>
Student IDs	163014038, 163014011 and 182014030
Supervisor	Dr. Mohammed Ashikur Rahman

We declare that this project entitled *Real-Time Motorbike Helmet Detection System Using Hybrid Deep Learning Model* is the result of our own work except as cited in the references. The project has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Jahida Alam Jaishee
163014038

Department of Computer Science and Engineering
University of Liberal Arts Bangladesh

Aldrin Abir
163014011

Department of Computer Science and Engineering
University of Liberal Arts Bangladesh

Irfanuddin Ahmed
182014030

Department of Computer Science and Engineering
University of Liberal Arts Bangladesh

Date: October 14, 2024



Department of Computer Science and Engineering
University of Liberal Arts Bangladesh
Mohammadpur, Dhaka - 1207

CERTIFICATE

This is to certify that the project entitled **Real-Time Motorbike Helmet Detection System Using Hybrid Deep Learning Model**, submitted by **Jahida Alam Jaishee** (Student ID: 163014038), **Aldrin Abir** (Student ID: 163014011) and **Irfanuddin Ahmed** (Student ID: 182014030) are a undergraduate student of the **Department of Computer Science and Engineering** has been examined. Upon recommendation by the examination committee, we hereby accord our approval of it as the presented work and submitted report fulfill the requirements for its acceptance in partial fulfillment for the degree of *Bachelor of Science in Computer Science and Engineering*.

**Dr. Mohammed Ashikur Rahman, Assistant
Professor & Coordinator**

Dept. of CSE, University of Liberal Arts Bangladesh

Sovon Chakraborty, Lecturer

Dept. of CSE, University of Liberal Arts Bangladesh

Shakib Mahmud Dipto, Lecturer

Dept. of CSE, University of Liberal Arts Bangladesh

Md. Aktaruzzaman Pramanik, Lecturer

Dept. of CSE, University of Liberal Arts Bangladesh

Place: Dhaka

Date: October 14, 2024

ACKNOWLEDGEMENTS

We would like to express our deep and sincere gratitude to our research supervisor, *Dr. Mohammed Ashikur Rahman*, for giving us the opportunity to conduct research and providing invaluable guidance throughout this work. His dynamism, vision, sincerity and motivation have deeply inspired us. He has taught us the methodology to carry out the work and to present the works as clearly as possible. It was a great privilege and honor to work and study under his guidance.

We are greatly indebted to our honorable teachers of the Department of Computer Science and Engineering at the University of Liberal Arts Bangladesh who taught us during the course of our study. Without any doubt, their teaching and guidance have completely transformed us to the persons that we are today.

We are extremely thankful to our parents for their unconditional love, endless prayers and caring, and immense sacrifices for educating and preparing us for our future. We would like to say thanks to our friends and relatives for their kind support and care.

Finally, we would like to thank all the people who have supported us to complete the project work directly or indirectly.

Jahida Alam Jaishee, Aldrin Abir and Irfanuddin Ahmed

University of Liberal Arts Bangladesh

Date: October 14, 2024

Dedicated to
my loving family

– *Jahida Alam Jaishee*

To my beloved parents
the source of my inspiration

– *Aldrin Abir*

To
my beloved parents

– *Irfanuddin Ahmed*

ABSTRACT

In Bangladesh, motorcycle accidents are a major cause of mortality; most deaths are ascribed to riders not wearing helmets. This study intends to use a hybrid deep learning model to construct an effective, real-time helmet detection system in response to this rising issue. The system is intended to recognize helmets, riders, and license plates in intricate road scenarios by integrating YOLOv8 for object identification, ResNet50 for rider classification, and Paddle OCR for text recognition. The study used a large dataset from the Kaggle Helmet Detection Dataset and Bangladeshi real-life video footage, preprocessed using Roboflow for cleaning, annotation, and data augmentation. The model was trained using multiple checkpoints and selected best.pt due to superior performance. Data exploration and visualization techniques were used to ensure the dataset's quality and balance. Feature extraction techniques like edge detection and color histograms improved the model's accuracy in distinguishing helmets and riders. With a precision of 0.929, recall of 0.932, and mean average accuracy (mAP) of 0.949 at 50 percent IoU, the model's preliminary evaluations yield encouraging findings. These outcomes show how well the technology detects helmets in real-time, even in intricate traffic situations. In addition to providing insights that may be used to guide policy choices targeted at lowering motorcycle deaths, the project's results highlight the significance of incorporating deep learning technology for traffic safety.

Keywords: motorcycle safety, helmet detection, deep learning, YOLOv8, ResNet50.

Contents

1	Introduction	1
1.1	Problem statement	2
1.2	Aims, objectives and Motivation	2
1.2.1	Aims	2
1.2.2	Objective	3
1.2.3	Motivation	3
1.3	Project Specification	3
1.3.1	Hardware Specification	3
1.3.2	Component Specification	5
2	Literature Review	7
2.1	Findings from The Existing Research	11
3	Methodology	14
3.1	Concept Generation	14
3.2	Data Collection	16
3.2.1	Public Dataset: Kaggle Helmet Detection	17
3.2.2	Annotation Tool: Roboflow	17
3.2.3	Custom Video Footage	18
3.3	Data Exploration and Visualization	19
3.3.1	Data Visualization	19
3.3.2	Data Breakdown	21
3.4	Data Preprocessing	21
3.4.1	Data Cleaning	21
3.4.2	Data Augmentation	22
3.4.3	Data Splitting	22

3.4.4	Class Distribution	23
3.5	Feature Extraction	23
3.6	Training Procedure	24
3.7	Model Methodology and Selection	25
3.7.1	YOLOv8 Architecture	25
3.7.2	ResNet50 Architecture	27
3.7.3	Experiment Procedure	27
3.7.4	Integration of YOLOv8 and ResNet50	29
3.7.5	Combining Results	29
3.7.6	Real-Time Display	31
3.8	Front-End Development	31
3.9	Back-End and Output Layer	32
3.10	Hyperparameter	32
3.11	Evaluation	33
3.12	Justification and Novelty	34
4	Project Management Financial Activities	35
4.1	Project Financial Management	35
4.1.1	Work Breakdown Structure (WBS)	35
4.1.2	Cost Estimation	36
4.2	Design and Development	37
4.2.1	Object Detection Subsystem (YOLOv8)	37
4.2.2	Rider Classification Subsystem (ResNet50)	37
4.2.3	Engineering Analysis and Validation	38
5	Web Application Documentation	39
5.1	Project Setup	39
5.2	Application Overview	40
5.3	Application Workflow	40
5.4	Flask Models and Views	41
5.5	Flask	42
5.6	Video Processing	43

6	Result analysis	44
6.1	Performance Evaluation of the YOLOv8 and ResNet50 Models	44
6.1.1	YOLOv8 for Object Detection	44
6.1.2	ResNet50 for Helmet Classification	45
6.2	Precision-Recall and Confidence Curve Analysis	47
6.3	Comparison with Existing Systems	51
6.3.1	Helmet Detection	51
6.3.2	Critical Analysis and Potential Improvements	52
6.4	Result	52
7	Conclusions	55
7.1	Social, Legal, Ethical, and Environmental Issues	55
7.1.1	Social Impact	56
7.1.2	Legal Compliance	56
7.1.3	Ethical Considerations	56
7.2	Future Work	57
	References	59

List of Figures

3.1	Proposed System Methodology	16
3.2	Data Annotation Process	18
3.3	Sample of Bounding Box	20
3.4	Training Process Flowchart	25
3.5	Architecture of YOLOv8	26
3.6	ResNet50 Architecture Diagram	28
3.7	Integrated System Architecture Diagram	30
5.1	Home Page:	42
5.2	Results Page:	42
6.1	Confusion Matrix	46
6.2	Normalized Confusion Matrix	47
6.3	Precision-Recall Curve	48
6.4	F1-Confidence Curve	49
6.5	Precision-Confidence Curve	50
6.6	Recall Confidence Curve	51
6.7	Training losses, Validation losses, and mAP	53
6.8	Results	54

List of Tables

2.1	Summary of Helmet Detection Papers	13
3.1	Summary of Datasets Used	21
4.1	Cost Estimation Table	36
6.1	Training and Validation Metrics per Epoch	45
6.2	Comparison of accuracy and performance across different systems and papers	52

Chapter 1

Introduction

The small motorbike has become a near fatal danger for millions and that is especially true in some very populous countries starting with Bangladesh where motorcycles are kings of public transit. Continually driving the use of bikes to increase, so is everything that goes with it. Today, helmet checking and law enforcement is probably hardest due to the presence of many on-road vehicles moving one after another in traffic signals, which makes it almost impossible for people sitting inside to take round checkups. Intelligent helmet detection based on deep learning is implemented to detect whether helmets are worn during the real-world riding, and license plate recognition technology utilizes data augmentation as preprocessing steps for rider identification purposes. This system brings together several recent models (namely YOLOv8 for object detection, ResNet50 model for rider classification) to provide a robust road safety enhancement that has some degree of automation.

An important element of the problem is that it has to work in real-time and be very good at differentiating helmet use under complex traffic scenarios, like these ones found on streets such as Bangladeshi roads. In the meantime, the system will make it easier for authorities to enforce safety regulations more efficiently leading to a significant reduction of motorcycle deaths. Through this system, violators will also be identified using their license plates and it can serve as an all-encompassing device to ensure better compliance with traffic laws. The introduction speaks to the growing demand for traffic safety automation and lays a foundation for deeper technical discussion as had been provided in this project. We are developing a system that will provide important insights to policy makers and proper action

based on these insights will ensure a reduction on the fatality rate by detecting motorbike helmets. Deep learning technology works marvelously when it comes to detecting any object. Detecting the helmet of the riders using a hybrid deep learning model, can reduce fatal injuries approximately 50% [Dhaka Tribune \(2023\)](#). That is why we are developing a system that can provide reliable results in a complex road scenario like Bangladesh using the combination of YOLOv8 and ResNet50 methods of deep learning technology.

1.1 Problem statement

In Bangladesh, one of the most significant causes of death in motorbike accidents is not wearing a proper helmet. 9951 people died in 2022 by road accident and among them, 3,091 people died by motorbike accidents [New Age \(2023\)](#). In addition, 88% of those 3091 people died due to not wearing a helmet [Dhaka Tribune \(2023\)](#). According to the Road Safety Foundation, wearing a proper helmet can reduce the risk of fatal injuries by an estimated 42 percent and non-fatal injuries by 69% [Dhaka Tribune \(2023\)](#).

1.2 Aims, objectives and Motivation

This project is aimed to create a system that can identify a helmet or license plate automatically with YOLOv8 and ResNet50 for real-time analysis. The system will provide the mechanism for law enforcement to better enforce helmet laws and help decrease motorcycle-related deaths. With an increasing number of deaths in Bangladesh because of not wearing helmets, the project is hoping to upgrade road safety and back to more effective traffic law enforcement.

1.2.1 Aims

The main purpose of this project is to develop a real-time automated system for road safety by employing a machine learning solution, which ensures helmet compliance with motorbike riders and their passengers. It involves a system that can detect if the rider of a motorbike is not wearing his helmet, and immediately after it was carried out photos to catch the number plate the vehicle was wearing. Also, the

presence of a passenger can be detected by this system and it even verifies if the pillion rider also has worn their helmet. This we hope to do by ensuring that traffic safety regulations are enforced in real-time through the enforcement of road accidents leading to death.

1.2.2 Objective

The Project has the following specific objectives :

- To detect motorbike helmet of rider using hybrid DL models with YOLOv8 and ResNet50.

1.2.3 Motivation

The trend of using motorcycles in our country is rising. And, it can continuously increase per year motorcycle users. This also, motorcycle accidents are increasing. Motorcycle accidents claim more lives than most other types of road mishaps. According to Road Safety Foundation, 11,669 motorcycle accidents occurred in 5 and a half years, 11,593 people were killed. And also the number of motorcycle accidents during four years increasing with time in 2019, 2020 ,2021 and 2022 are 1189, 1381, 2078 and 2973. With 945, 1463, 2214 and 3091 deaths respectively. Compared with statistics from 2021, motorcycle deaths rose by over 43. 88% of people failed simply because they did not wear a helmet. We were inspired as a team to do this project from the sheer number of deaths that could be saved if people simply wore good helmets.

1.3 Project Specification

1.3.1 Hardware Specification

The below setups are used for system development and deployment in both on-premises as well cloud-based environments.

Setup 1: PC Configuration

- Processor: Ryzen 5 5600

- Motherboard: Gigabyte B450M S2H V2 AMD AM4
- RAM: 16 GB (2 x PNY 8GB XLR8 Gaming DDR4 3200 MHz)
- Storage: Western Digital Green 240GB M.2 SSD
- Graphics Card: Zotac Nvidia 1030 2GB

Setup 2: Cloud Computing

- GPU: T4 GPU from Google Colab
- Memory: Standard memory configuration for Google Colab's GPU access
- Cost: Usage-based pricing on Google Colab for GPU access

Specifications of Data Collection Devices:

Device 1:

- Model: TECNO SPARK Go, TECNO BF7
- CPU: Helio 22
- Back Camera: 13M dual Camera

Device 2:

- Model: Redmi Turbo 3
- Video Settings: 4K, 60fps
- Camera: 50MP

Device 3:

- Model: Samsung Galaxy S10e
- Main Camera: 12 MP, f/1.5-2.4, 26mm (wide), 1/2.55", 1.4µm, dual pixel PDAF, OIS
- Ultrawide Camera: 16 MP, f/2.2, 12mm (ultrawide), 1/3.1", 1.0µm, Super Steady video
- Video: LED flash, auto-HDR, 4K@60fps

1.3.2 Component Specification

The system employs open-source software tools and platforms for data processing, model training and deployment. They are critical components to allow the system to meet performance, scalability and ease of development requirements.

Google Colab: A cloud-based platform that gives access to GPUs for training deep learning models. This platform allows us to use high-performance computing resources without incurring a significant cost, ensuring the models are built and tested faster.

Python: The main programming language used to implement the deep learning models and perform different data processing tasks. Python's extensive library availability and several important frameworks make it possible in a machine learning and deep learning project.

PyCharm: An Integrated Development Environment used for code writing and debugging in Python. PyCharm provides code completion support, a debugger, and a project view with many project navigation and management tools.

Roboflow: A data preprocessing and critical expression tool used to manage datasets, flick data, and generate training data for computer vision tasks. Roboflow is utilized for labeling images, cropping, and essential expression on train-data preparation.

OpenCV: Inexpensive use of an open-source library in the project for real-time computer vision. OpenCV is used to answer the video frame and make transformations on video frames before submitting them on deep learning models. TensorFlow: The use of a deep learning framework when building digital models. These frameworks are used in building models and adding optimization in other systems. TensorFlow is a framework used to build and train digital neural networks.

Flask: A micro web framework used to deploy helmet detection on the web. Flask provides an easy web user interface to upload video and receive the detection result on time. The eye in the middle python server web framework offers a ready-to-use website integration for implementation or system monitoring on web screens.

All played a vital role in supporting the whole system based on high performance.

The use of GPU facilitation via the cloud such as Google Colab ensures cost with full training model outputs.

Chapter 2

Literature Review

Several researchers have been studying motorcycle helmet detection in recent years; some of them have even looked into license plate identification.

[Jia et al. \(2021\)](#) Construction of a Leading-edge System for Motorcycle Detection and Helmet Recognition in Developing Countries. A dual-purpose YoLOv5 motorcycle detector that processes motorcycles first and then sees helmets in the frame. First, the system uses a triplet attention mechanism and integrates Soft-NMS for robust occlusions. This system surpasses current methods for real-time helmet detection and attains mAP of 97.7% as well as an F1-score value of 92.7%.

[Tran et al. \(2023\)](#) Estimating motorcycle helmet usage from video: A novel two-stage approach to enhance road safety surveillance. The system comprises a motorcycle and rider detector, followed by a segment that detects whether the identified riders are wearing helmets. Model: the framework utilizes deep learning-based methods, particularly the YoLOv8 algorithm to achieve 0.7754 accuracy improvement at AI City 2023 Public Leaderboard The system combines advanced algorithms for handling specific challenges, such as lighting conditions and occlusions, showing its potential to robustly support glasses-wearing compliance surveillance of helmet usage in real-world traffic environments. Another work by [Nagoriya \(2020\)](#) aims to achieve the automatic detection of motorcycle helmets based on a new framework approach for video surveillance visualization. The framework is two-stage: a detector to localize motorcycles and their riders, then an IDer of the helmet status. This is done by using the competition-class rank YoLOv8 model, which shows a large improvement concerning detection methods in previous works. The authors

recommend that BIRL-based pose should consider advanced algorithms to handle the more challenging environment where different lighting effects and large partial occlusion exist such as for this particular case, a specific deep neural network is recommended. It also provides better-integrated control algorithms of iTs and safer roads.

[Chairat et al. \(2020\)](#) describes a new method framework for detecting the wearing of helmets on motorcyclists using video surveillance. This system overcomes these problems like the exact localization of motorbikes and driver-rider difference helmet compliance. The two-step process: a detector asking for motorcycles, and an identifier deciding if the seen rider is wearing a helmet. It will achieve high accuracy because the proposed method is performed better than other methods. The system has also been validated as a dependable unit in the enhancement of compliance with helmet legislation deployment and enforcement for road safety development. In another paper [Agorku et al. \(2023a\)](#), they described a method of the YOLOv5 deep learning model for identifying non-helmet using motorcycle riders. High speed and accuracy in traffic monitoring system with low false positives. Hence the authors combine ensemble learning methods to boost detection performance in different conditions and occlusions. The system significantly advances the tech from typical and deep learning approaches to up disorder statistics, tries hard in the road safety ecosystem, and saves motorcycle deaths caused by accidents.

[Agorku et al. \(2023b\)](#) proposed an automatic system for the recognition and tracking of license plates for helmetless motorcyclists. It uses YOLOv2 and MLPoseNet, the model which I have trained.v3 for person, motorcycle, helmet. it also does license plate detection using Yolov4. With video input, it monitors traffic conditions in real-time and fines instead of through traditional manual labor. In the study, methods developed in deep learning increased detection performance and precision. License plate extraction is done through an optical character recognition(OCR) system.

The study by [B.-S. Wang et al. \(2023\)](#) Video Analytics for Motorcycle Helmet compliance employs Deep learning to recognize and quantify helmet status instantly. (A necessary condition to effectively enforce traffic law in high non-compliance settings) Our system works robustly across video feeds — even in challenging

scenarios including low light conditions and with occlusions. ResultsExperimental results indicate good performance, and therefore our system is highly beneficial for traffic authorities to keep roads safe.

The paper [ALWIN et al. \(2022\)](#) presents an approach to automatically detect helmetless riders using CCTV images with the annotated helmet and penalize them as per their license. The better YOLOv3 model was clearly able to distinguish real helmet use from stored ones, eliminating false positives. How license plate recognition enhances the accuracy of enforcement for fines and tickets However, the study also recognizes that pillion rider detection is still an open problem due to a lack of training data and hence comprehensive datasets are essential. The related authority believes the method can advance traffic law enforcement technology as well as render motorcycle safety checks almost completely autonomous.

[Soltanikazemi et al. \(2023\)](#) Real-Time Helmet Detection: introduced our helmet detection dataset and a real-time helmet system using the YOLOv5 model leveraging genetic algorithms for hyperparameter tuning. It is suitable to locate helmet non-adherence effectively, has an effective mean Average Precision score of 0.5377, and performs notably till the final standings in the AI City Challenge. Recently, YOLOv5 and other deep learning-based methods have been adopted in previous works to achieve better object detection results, especially for challenging situations. The research shows that there is a use of automatics in road safety measures and police helmet ability.

Using a single small dataset of 300 images, [Susa et al. \(2022\)](#) employed YOLOv3 to recognize authorized helmets among motorbikers. Twenty-five models were made and tested, with Model 18 demonstrating the greatest performance, yielding an mAP of 0.97. These experiments showed that YOLOv3 could perform well in helmet detection, but that the method required optimized fine-tuning and comparing across models. Such studies could help minimize traffic crashes related to poor-focused safety and authorized helmet identification by assisting in the selection of appropriate models.

This paper [Allamki et al. \(2019a\)](#), instead is about a system developed to improve

road security by controlling if motorcycle riders wear helmets. The authors In response to the increasing number of road accidents observed in India, automated detection and machine learning-based systems for disaster prevention. The essential ones are the importance of helmet-wearing for motorcyclists and discussing traffic analysis and object detection to emphasize the necessity of an effective system for detecting helmets. The approach uses the YOLOv3 model due to its fast speed and high-precision results within five classes: riders wearing helmets, “riders do not wear a helmet”, motorcycles, passengers seated backside” persons, and license plate. The model, trained on a custom dataset of 11,000 images across over 50,000 iterations was able to achieve an average precision of roughly 75% and the OCR component had roughly 85% accuracy in identifying license plates.

This paper [L. Wang et al. \(2022\)](#) employs the modified version of the YOLOv5 framework to detect safety helmets worn by construction workers. The study of the construction environment is faced with challenges such as complex backgrounds, crowded targets, and helmet irregular shapes. This procedure and methodology do so by incorporating CBAM with DCN to the core model of YOLOv5, which in turn achieves performance enhancement under complex cases due to the attention mechanism being utilized alongside adaptable feature extraction. To improve the precision of datasets with high-density objects, The study exchanges GIoU loss in Faster R-CNN to DIou loss. Trained using a custom dataset, which consisted of open-source data and self-captured images: it reached 91.6% accuracy with the speed at processing 29 frames per second solving for real-time use cases

This paper [Yogameena et al. \(2019\)](#) focuses on the development of an automated surveillance system that improves road safety by monitoring motorcycle riders and checking whether they wear helmets. It looks to address the problem of motorcycle accidents in India which could be prevented by using helmets and some users have volunteered for it as an extra line of defense. Methodology:- The methodology uses a Gaussian Mixture Model (GMM) to segment the foreground objects in video clips and utilizes R-CNN approach for detecting motorcycles, as well helmets. This method is robust in the real world against problems such as low resolution, occlusion, different light conditions and illumination. Only a little bit more than the anarchic system but still, one of them deserves applause with an average precision

mean at 82.925% and LP method reaches edge lines in up to 95%, which is quite good metric over other algorithms as well.

This paper [Han & Zeng \(2021\)](#) outlined the problems of providing construction site workers with safety gear and significance helmet detection in many aspects. We then walk through YOLO architecture and approach (treating object detection as a regression task) — finally, positing that since the model is broadly efficient + easily modified, it makes sense for us to base our work on this framework; we choose YOLOv5. A fourth detection scale is added for small targets such as helmets, and an attention mechanism in the backbone network to improve feature representation. Results are presented with a mean average precision of 92.2%, a relative increase of 6.3% against the baseline YOLOv5, and an inference time of just 3.0 ms per image (640×640 pixels). Even more useful are the model’s compact size and potential application in practical, real-world situations with workers wearing helmets for additional protection.

2.1 Findings from The Existing Research

Different types of algorithmic approaches and model architectures are covered in the review of existing research papers that discuss helmet detection and other computer vision applications, highlighting progress made in this direction. These studies agree that YOLO (You Only Look Once)-variant models were the best fit for real-time helmet detection with their balance between accuracy and speed, notably: YOLOv3, YOLOv5, and even up to a recent solution named "YOLOv7". [Allamki et al. \(2019b\)](#) and [Allamki et al. \(2019a\)](#) combined YOLOv3 but for this purpose used optical character recognition (OCR) to recognize number plates and helmets, respectively, which resulted in 75% mAP and OCR accuracy of 85%, demonstrating the importance of combining object detection with text recognition. [L. Wang et al. \(2022\)](#) The effectiveness of attention mechanisms in improving performance has similarly been demonstrated by other follow-up works such as YOLOv5: hence, after integrating advanced modules into these later versions (e.g., CBAM, DCN, and DIOU), their detection accuracy could reach 91.6% at a speed of up to 29 FPS. Other techniques like [Yogameena et al. \(2019\)](#) which combined GMM with R-CNN for improved localization (82.9% mAP), and [Han & Zeng \(2021\)](#) who used YOLOv5

to achieve 92.2% mAP, further validate the practicality of our real-time approach using YOLOv5.

Numerous papers studied novel architectures that employed a combination of techniques to enhance detection. This paper was also the first to introduce Ghost-Net with MSFFN and LRCA-NetV2, obtaining 93.5% mAP at a speed of 42 FPS, showing the capability that lightweight networks can deliver for real-time applications. [Geng et al. \(2021\)](#) also used YOLOv3, targeting label confidence, and got a score of 0.92 whilst [Siebert & Lin \(2020\)](#) who combined models: YOLOv9 with RetinaNet achieved accuracy at level 74.6% The more advanced tracking models such as Deep SORT by [Tomas & Doma \(2024\)](#) to couple the YOLOv4 and v7 has achieved a precision of 95.6% mAP@0.5 of 95.1%, which resulted in very good results for low-light performance requirements. Studies such as [Shan et al. \(2023\)](#). This work, Dataset Current 2023 with YOLOv5 + ECA and BiFPN even reached an accuracy of 95.9%, which demonstrated again the superiority of feature fusion techniques. For helmet detection, YOLO models have proved to be the most efficient in real-time based on high precision and recall values across all papers for traffic safety as well as industrial safety solutions.

We chose YOLOv8 to improve the helmet detection performance both in accuracy and real-time speed compared with its previous versions. Hyper ledger Fabric is a high-powered blockchain that has been designed for real-world use and practical results due to its rich feature set, security mechanisms, etc. We chose ResNet50 for its ability to perform feature extraction, specifically in deep learning tasks due to the residual connections which ease the training of deeper networks. Which seeks to exploit the strengths of each model for both models to demonstrate peak performance when used together for helmet detection applications.

Author(s) and Year	Algorithm	Results
Allamki et al. (2019b)	YOLOv3 + OCR	mAP: 75%, OCR accuracy: 85%
L. Wang et al. (2022)	YOLOv5 + CBAM + DCN + DIoU	Accuracy: 91.6%, Speed: 29 FPS
Yogameena et al. (2019)	GMM + R-CNN	mAP: 82.925%, LP method: 95%
Han & Zeng (2021)	YOLOv5 + Fourth detection scale + Attention mechanism	mAP: 92.2%, Detection time: 3.0 ms/image (640×640)
Liang & Seo (2022)	GhostNet + MSFFN + LRCA-NetV2	mAP: 93.5%, Speed: 42 FPS
Geng et al. (2021)	YOLOv3 + Gaussian fuzzy data augmentation	Label confidence: 0.92
Siebert & Lin (2020)	YOLO9000 + RetinaNet	Accuracy: 74.6%
Tomas & Doma (2024)	YOLOv5 + YOLOv7 + Deep SORT	Precision: 95.6%, Recall: 91.2%, mAP@0.5: 95.1%, mAP@95: 76.3%
Agorku et al. (2023c)	YOLOv5 + Data Augmentation	mAP: 0.526
Shan et al. (2023)	YOLOv5 + ECA + BiFPN	Accuracy: 95.9%

Table 2.1: Summary of Helmet Detection Papers

Chapter 3

Methodology

This chapter will describe the methodological steps for designing and building a check helmet detection system. This methodology includes specific components in terms of data capture, pre-processing, and model we structure both the architectural design and feature extraction, as well as evaluation. The high accuracy of the helmet, motorcycle rider detection, and real-time performance are achieved. From the system, classification and license plate recognition will have been applied which has been demonstrated in every stage of this work That approach is split into pre-processing data model selection network configuration Different Main Phases training and evaluation.

3.1 Concept Generation

The idea creation section centered on image processing algorithms such as YOLOv8 and ResNet50 for detecting motorcycle riders in real-time, type of helmet used by the rider present- Not-present status followed when content alludes helmets are worn or License Plate recognition (LPN). This was so that all three of these activities could be done in a holistic way to work well together with minimal latency even in challenging traffic conditions.

YOLOv8 Object Detection So the first step of concept generation led to YOLOv8 for object detection. YOLOv8 (You Only Look Once v4) is the yolo model for detection, and it takes traditional methods one step further by combining them to achieve a nice balance between speed & accuracy. It can detect small objects in cluttered

environments (like helmets and motorcycles) which is perfect for the task of detecting Riders, Helmets & Motorcycles from video frames. YOLOv8 is super fast by incorporating a CSPDarknet backbone to extract features and PANet neck for better feature fusion. The anchor-free detection mode and Complete Intersection-over-Union (CIoU) as the loss function aid YOLOv8 in ensuring precise bounding box localization, an essential aspect of applications such as real-time traffic monitoring.

The same system then asks for a helmet detection model issuance to classify riders based on whether they happened to be wearing helmets. Choosing ResNet50 for this job, due to its strong deep learning architecture where residual connections are used so as not to face the vanishing gradient problem and therefore can learn higher-level features. ResNet50 — since it is also pre-trained on massive data like ImageNet, and aspires to be competent in transfer learning. ResNet50 was pre-trained on the ImageNet dataset and fine-tuned with our helmeted vs. non-helmeted rider-level classification data to be able to effectively classify riders as either of these two classes based on features such as shape, structure and coloration of helmets in visual information about them.

Therefore, it is of paramount importance to integrate these three models: object detection with YOLOv8, rider classification based on ResNet50. YOLOv8 detects the object in each frame, resnet50 classifies riders based on detected region These all integrate with the system allowing you to detect helmets, rider classification and license plate recognition in real-time for a complete traffic monitoring / law enforcement solution.

The architecture was humanized with the emphasis of real-time processing, accuracy with a defense to work best in different traffic conditions which is likely to be seen on Bangladeshi roads.

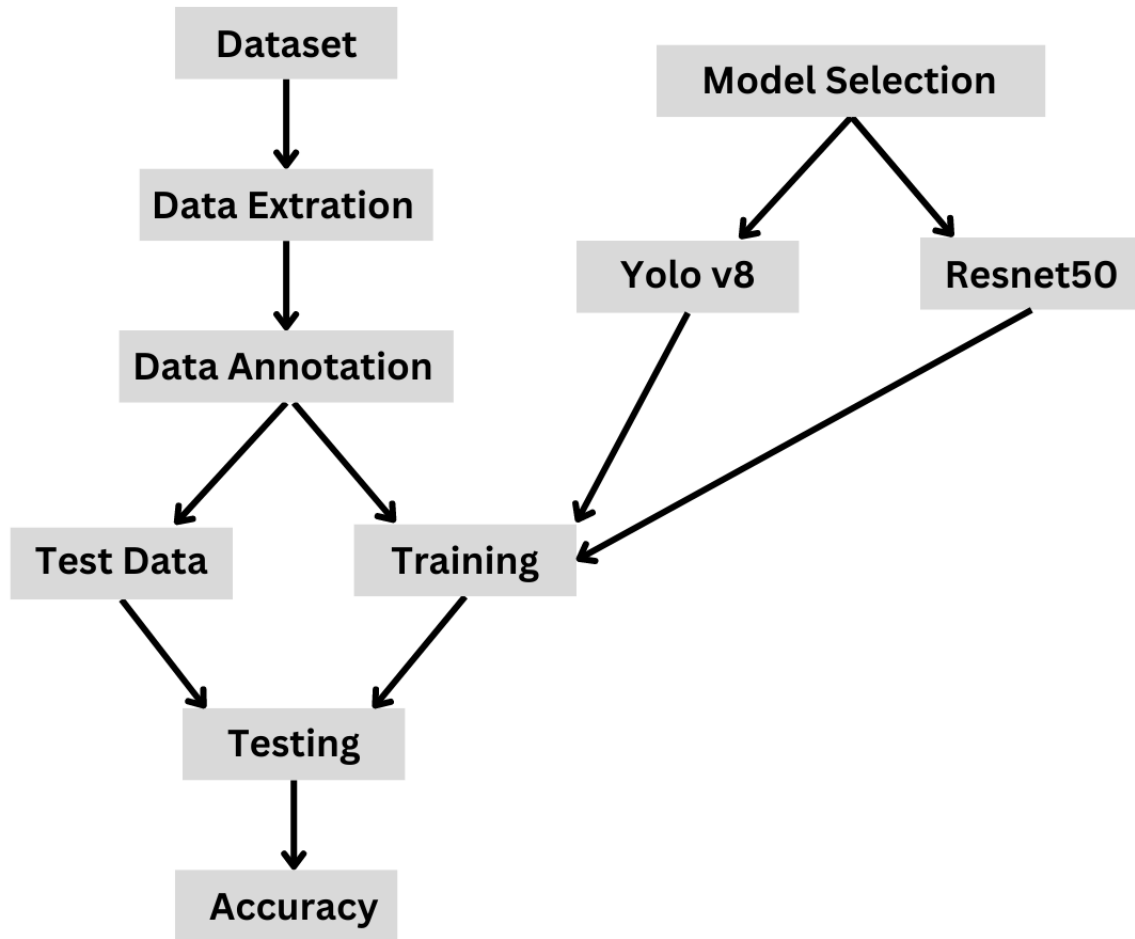


Figure 3.1: Proposed System Methodology

3.2 Data Collection

The main two input sources for these models are a public dataset and real-life video data we collected ourselves. With this combination, we had a dataset of both varying traffic conditions and rider behavior which was critical to developing an effective lane-change policy.

3.2.1 Public Dataset: Kaggle Helmet Detection

Kaggle Helmet Detection Dataset (One of the main sources for data) The dataset consists of a set of pre-labeled images with riders wearing helmets and others not. The images in this corpus are varied, with different readings as view angles under various lighting and settings. Such conditions resemble the ones encountered on roads while driving and are a good environment to train, and test the helmet detection capabilities of our model in different real-world scenarios. This dataset laid a strong foundation for the initial phase of training the model, so that when it sees new data unseen by machine learning frameworks before helps to generalize better. We used dataset from here:

https://www.kaggle.com/datasets/savanagrawal/detect-person-on-motorbike-or-scooter?fbclid=IwY2xjawGDkBBleHRuA2FlbQIxMAABHZqLedMdHm6tswmu7mfm3J7DOsKxneX0ivJvujEkECNIWL3PBiUjZQZ1cA_aem_FR5UBtax6NFNS79t2Bdb_g

3.2.2 Annotation Tool: Roboflow

The images were annotated using Roboflow, a popular annotation tool that helped label these inputs for training. We drew bounding boxes around the helmets, riders, and license plates within our images using Roboflow. Therefore, the annotation was necessary to train models for object detection as it offered ground truth that needed to be learned from by the model. These are the bounding boxes which help us to know where we need to focus on by our object detection model YOLOv8, during training. Roboflow helped to facilitate this by enabling very accurate annotation which ensured the bounding boxes were properly placed around riders and car number plates.

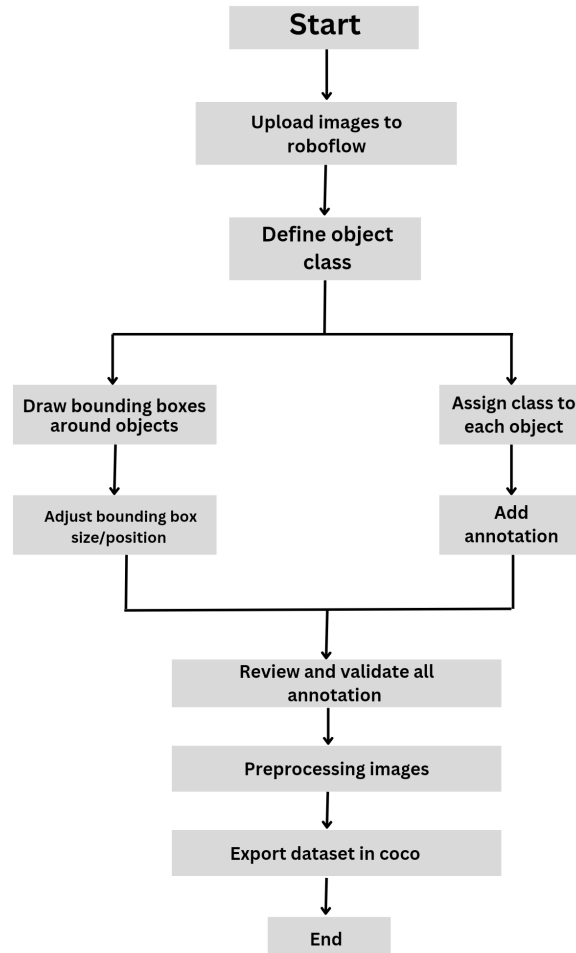


Figure 3.2: Data Annotation Process

3.2.3 Custom Video Footage

Apart from public dataset, we captured video of real-life traffic environments in Dhaka while driving on roads for training our model. The recordings were made with 20 smartphones in diverse realistic conditions (day and night, clear or low

lighting as well as different traffic flow patterns – sparse to rush-hour). Use of real-life footage provides a more accurate depiction of the difficulties faced in day-to-day traffic monitoring, including different weather conditions and levels of traffic congestion as well as various viewing angles. The custom footage also made it possible to collect license plate data essential for the aforementioned automatic number-plate recognition part of the system. The video images they collected showed moving motorcycles, often partially obscured or head-on to the camera, which would be difficult for standard object detection systems. Moreover, the diversity of epidemiological transitions that exist in this custom dataset make our model adapt with different real-world scenarios where reliability is essential in handling a few unpredictable conditions.

3.3 Data Exploration and Visualization

Prior to training the models, a detailed examination of data was carried out to get insights on its structure, distribution and any possible biases prevalent therein. This step was done to understand the class distributions and visualize data in case we could see any problem while training it.

3.3.1 Data Visualization

We visualized the quality and diversity of our dataset in a few ways.

Label Distribution Plot: The balance of helmeted versus non-helmeted riders was visualized using this plot. Finally, it verified us that the model would not be biased towards taking one class or another since both were adequately represented.

Sample Image Grid: After the types of environments where motorcycle riders can be captured, a grid was created showcasing sample images from each category. This grid showed images with various lighting, angles and backgrounds to model the varying real-world traffic settings.

Bounding Box Visualization: This visualization overlaid the bounding boxes on sample images to check if annotations were correct. We pressed the bounding boxes and observed that helmets and number plates were marked correctly.

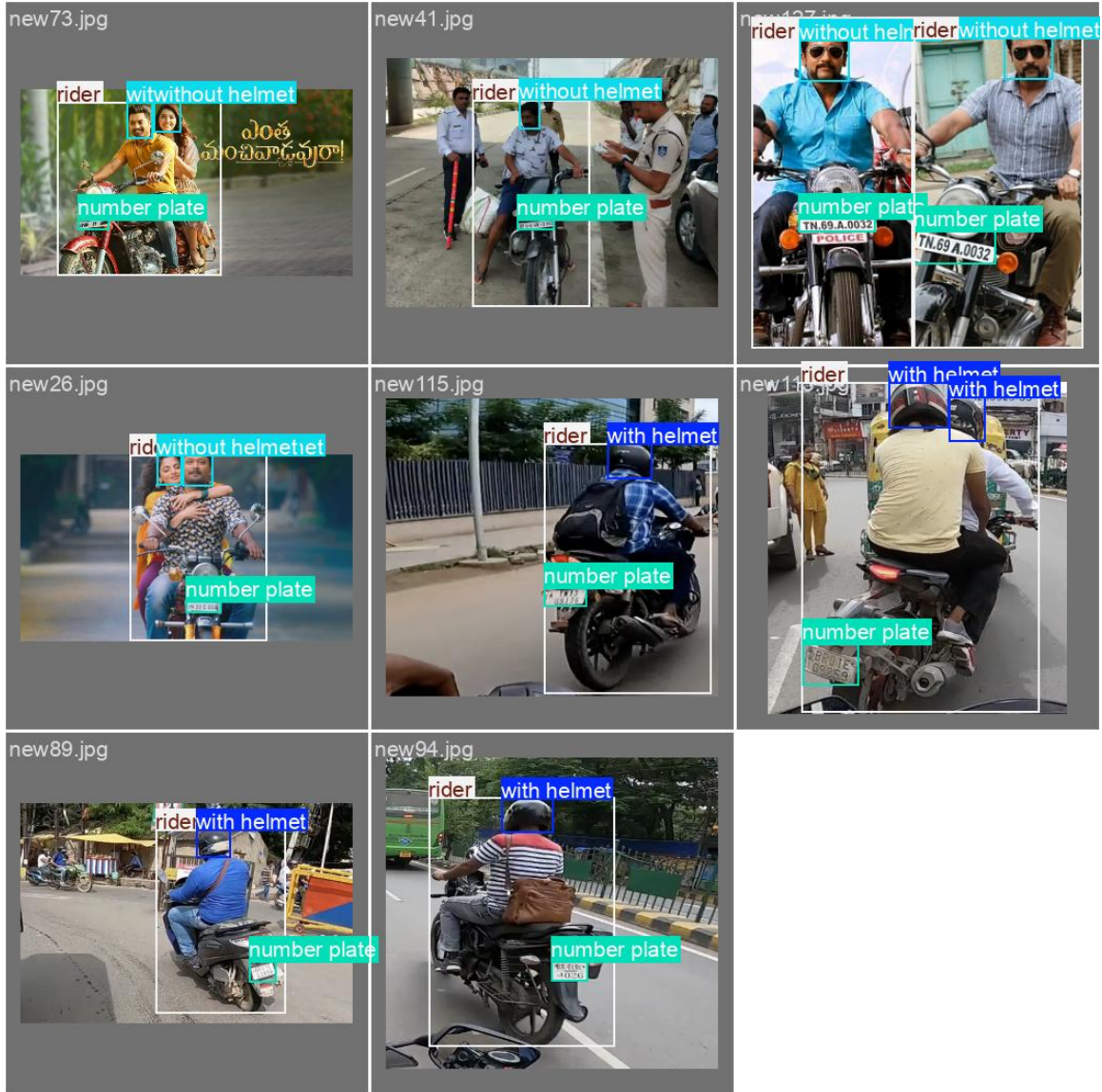


Figure 3.3: Sample of Bounding Box

3.3.2 Data Breakdown

The public dataset and custom footage were collected, organized, and analyzed to adequately balance the data across all possible viewable metrics. The dataset split is as below:

Table 3.1: Summary of Datasets Used

Dataset Name	Number of Images	Classes
Motorcycle Data	795	Motorcycle
Riders (Helmet/No Helmet)	583	Rider, Helmet, No Helmet
License Plates	8,823	License Plate

- **Motorcycle Data:** 795 images of motorcycles and the various states in which they occur.
- **With Helmet and Without Helmet Riders:** 583 images of riders, with helmets and without. The dataset is approximately evenly distributed between helmeted and non-helmeted riders. This balance was crucial for teaching the model how to classify riders based on helmet usage.
- **License Plates:** 8,823 images of motorcycle license plates, including various motorcycle types and license plates captured from different angles. These images are vital for the license plate recognition model.

3.4 Data Preprocessing

After the data collected, it was to be preprocessed, making sure that such high-quality training. Data preprocessing is an essential step where we clean, format and augment our dataset so the model can learn well from this data to generalize better for unseen data.

3.4.1 Data Cleaning

Clean training data by removing low-quality images as it deters performance of the model. We discarded any images that were blurry, poorly focused or had visual

artifacts. In addition, lighting-related problems e.g. over-exposed or under-exposed images were corrected to add more consistency to the dataset. As mentioned, it is crucial that the images are of high quality because if they are blurry, this would add noise to the training process which in turn could result in low accuracy when trying to detect/classify objects.

It has also removed the color inconsistencies that might be there and tried to make the dataset as consistent in terms of color. This step helped reduce variations in lighting that may be present under real-world conditions (lighting changes, shading and time of day).

3.4.2 Data Augmentation

In this system, multiple data augmentation techniques were employed to augment the size and diversity of the dataset. For example, random flipping was conducted such that the images were flipped left-right to capture subtle changes in viewing angle; brightness adjustment was performed as if under different lighting conditions; and rotations with varying degrees of randomness tried to model capturing motorcycles or riders from several angles in actual traffic. The model was trained on augmented images to help them generalize better, i.e., when exposed to different variations of the same entity (like glimpsing a helmet or motorcycle at various positions/evident conditions), it could easily cater for unseen data during testing/deploy time.

3.4.3 Data Splitting

Finally, for each subset of the dataset (training set vs. validation set and test), a split was formed after annotations had been augmented so that training/validation/test datasets were stored into its own folders. This step is important to avoid over-fitting and for the model to be tested under fair conditions on new data. The largest subset of data is the training set, this was used to teach the model how to identify helmets/riders/license plates. We also used a validation set to tune hyperparameters and keep track of the model performance during training. Lastly, the test set was used for final evaluation on top of a trained model to measure how well it would perform out in the “real-world” arena with new unseen data.

3.4.4 Class Distribution

Exploratory data analysis described the proportion of helmet wearing and non-helmet riding among this dataset. The dataset should be balanced since the model needed to learn how to differentiate between two classes with no bias. All imbalances were corrected by using data augmentation techniques, so there is no problem between classes on the training dataset. In doing this, the model learned to recognize riders who accurately say they are humans or potential bots - with/without helmets.

3.5 Feature Extraction

One of the most significant aspects present in the pre-processing pipeline is feature extraction, which provides a chance for our model to concentrate on such visual patterns that are necessary for detection and classification.

Edge Detection: Our algorithm uses edge detection to extract important information such as helmets and motorcycles. This made it easier for the model to distinguish between different subjects in an image, seeing as cars are often placed close together inside images of traffic. This model could more effectively detect and classify these objects by emphasizing the edges of helmets, motorcycles, riders.

Color Histograms: This leads to the use of color histograms and features in some measures about how to center an abundance of distributed objects basically capturing distribution by drawcolor. In this way, the color cube effectively helped to guide our model and tell which motorcycle is present of specific types between different rides since frequently we can get some sense by looking at the color and what type of object it is. Motorcycles are a good example of this; if traffic is busy or complex there can be relatively few detail differences between dozens of similar-looking bikes from the tiny seat-back at highway speeds (from close-up you just look for bodywork stickers, emblems, etc. and count wheels.)

Texture Analysis: We used texture analysis to extract the surface related features which would further aid in identification of helmets and non-helmet objects. This

approach was able to reveal important patterns and superficial features of helmets, motorcycles. It can aid the new model to improve its detection on how these objects are being differentiated from a background or another object. Exploring the texture of an object allowed us to fine-tune detection and classification abilities on subtle distinctions between objects.

3.6 Training Procedure

Training was carried out by Google Collab with T4 GPU, which provided access to computational resources necessary for all the processes associated with very large datasets and complex deep learning models. The training process was conducted in a multistage way where all the models (YOLOv8 and ResNet50) were fine-tuned individually before they are integrated into final system.

Data Splitting and Preprocessing: It was organized into training, validation and test sets to provide a strong data basis for the development of models that would expect model evaluations properly. The data was split to 70% for training, 15% validation and finally kept aside the remaining last 15%.

Data preprocessing steps of resizing, normalization and augmentations were applied before training. The images were resized to match the model's input size for YOLOv8 and annotations were converted into object detection format.

Training YOLOv8: Fine tuning YOLOv8 on the helmet detection dataset. Recent weights were loaded from a model trained on the COCO dataset and performed fine-tuning of helmet datasets. The model was trained for many epochs with early stopping to prevent overfitting. This includes optimization of the learning rate, batch size and IoU thresholds among other hyperparameters during training.

Training ResNet50: The ResNet50 was also fine-tuned using pre-trained weights on ImageNet. The last dense layer was changed to output only two values (helmet/no helmet), and the model is trained using this annotated dataset.

Loss function: Cross-Entropy lossThe model is evaluated using accuracy, precision and recall.

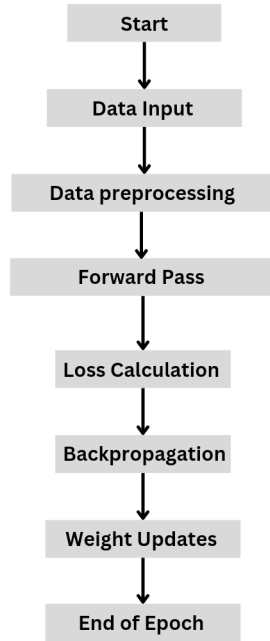


Figure 3.4: Training Process Flowchart

3.7 Model Methodology and Selection

3.7.1 YOLOv8 Architecture

YOLOv8 (You Only Look Once version 8) is the most advanced object-detection model and, due to its trade-off between speed and accuracy, very well suited for traffic monitoring in real-time. The architecture contains a CSPDarknet backbone that shuffles feature aggregation both before and after the convolutional layer, improving gradient flow during training. This results in faster convergence without reducing accuracy.

Additionally, YOLOv8 uses a PANet neck to connect low-level and high-level features for improved feature fusion. This has helped the model define helmets, or any other objects of that scale more accurately. Anchor-free detection is employed in YOLOv8, such that makes bounding box predictions complex and helps to generalize across different road conditions.

On top of that, YOLOv4 is fine-tuned with advanced loss functions such as Com-

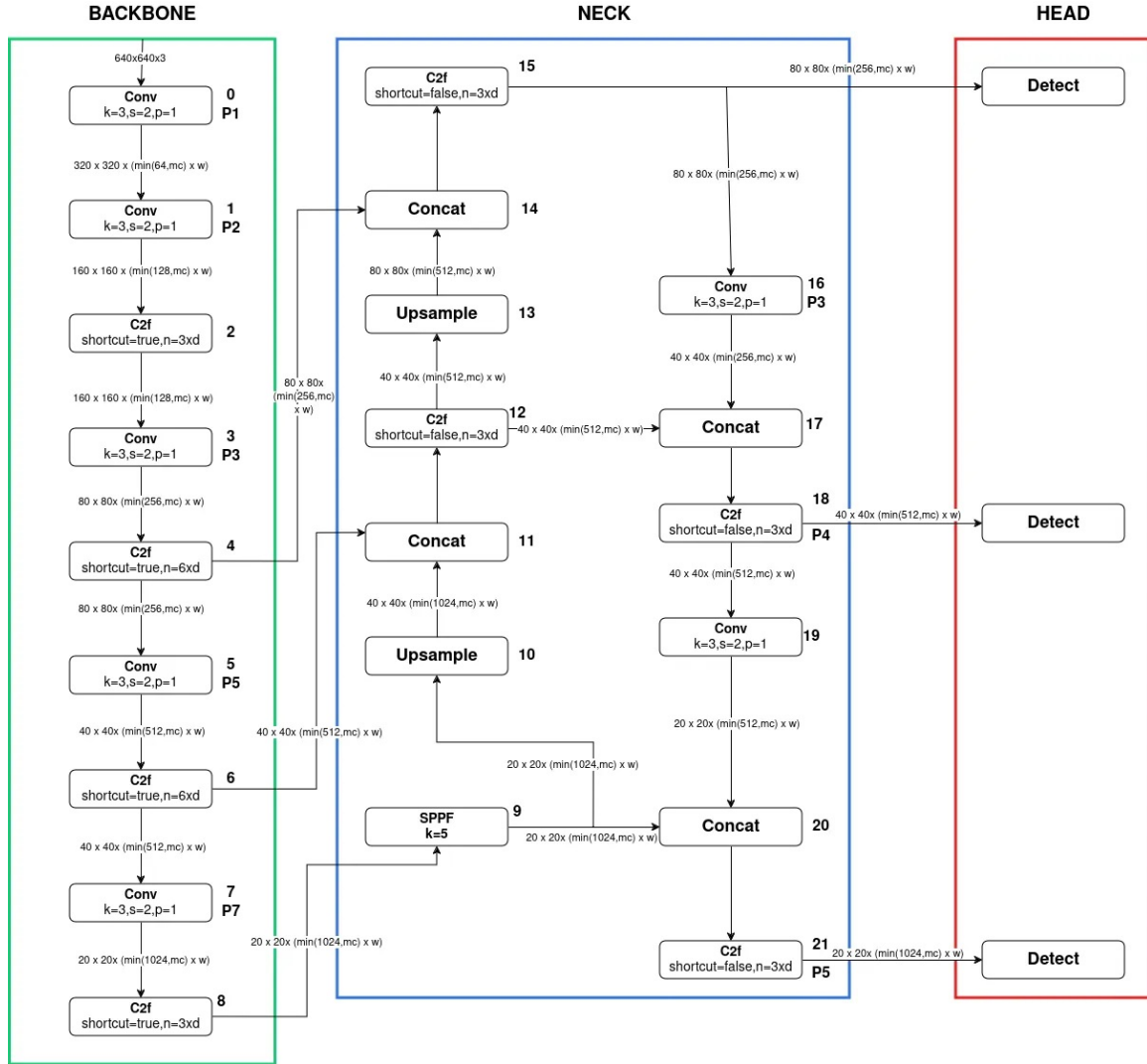


Figure 3.5: Architecture of YOLOv8

plete Intersection over Union (CIoU) which enhances bounding box regression by incorporating not only the overlapping area between predicted and true boxes but also their aspect ratio and distance. This way YOLOv8 attains very high accuracy in difficult road scenarios.

3.7.2 ResNet50 Architecture

ResNet50(Residual Networks) is a pretrained 50-layer deep convolutional neural network developed by Kaiming He et al. to learn structurally fine features from an input image, e.g., cats and dogs used in this model). ResNet50 incorporates residual connections (skip connections) in order to allow for gradients to flow directly through the network, making training deeper networks plausible.

ResNet50 was selected for rider classification in this project because it has shown to be successful at capturing finer details that are indispensable for differentiating helmeted from non-helmeted riders. This intricate architecture enables the model to capture complicated visual patterns, making it possible for this rather small network to classify images correctly even under very difficult traffic conditions.

The final layer was customised for binary classification (helmet or no helmet) and the pretrained ResNet50 model itself already had inbuilt data to detect helmets, making it very effective. Due to the strong performance in ImageNet classification, we made use of pre-trained weights on this network and finetuned it by training object class prediction layer only using helmet detection dataset which provided a good initial guess for model parameters resulting avoid any heavy retraining from scratch.

3.7.3 Experiment Procedure

YOLOv8 Training and Testing: Fine-Tuning YOLOv8 for Helmet and Rider Detection The fine-tuned on dangerous dataset to detect helmet and rider in the first phase of experiment. The model was fine-tuned on the custom dataset of motorcycle riders using weights pre-trained on COCO. The training involved adjusting several hyperparameters such as the learning rate, batch size and epoch.

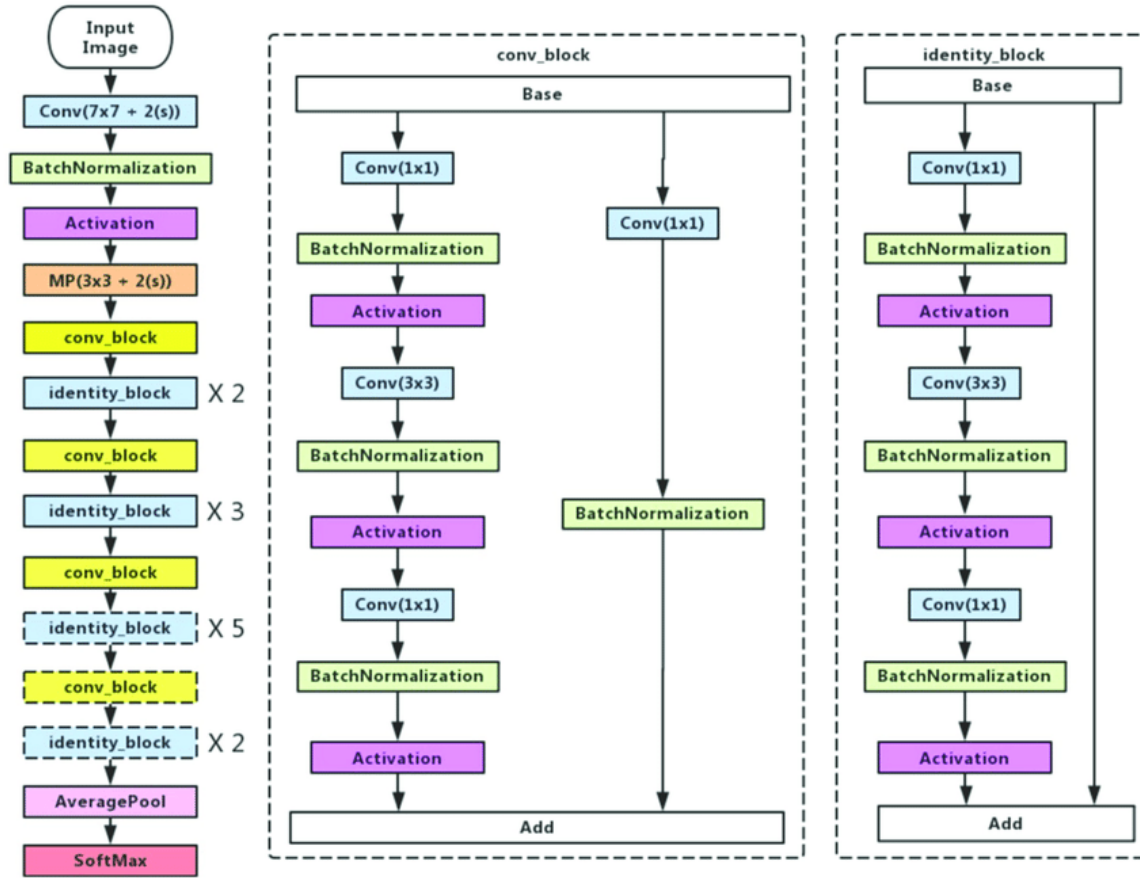


Figure 3.6: ResNet50 Architecture Diagram

We set the learning rate to 0.001, batch size of 16 and training for a total of 50 epochs.

Detections were considered to be 'valid' if they had an IoU greater than 0.5. The performance of the model is evaluated in terms of Mean Average Precision (mAP), precision and recall. It was also benchmarked for real-time performance by logging the frames per second (FPS) at which models processed video frames.

The mAP results of YOLOv8 were measured to be 0.949 at an IoU threshold value of 50% for detection classes on Test-set3. The precision was 0.929 and recall at 0.932, which means that the YOLOv8 performed a pretty great job of detecting helmets as well as riders with very high accuracy but low false positives. The model was able to process video frames at up to 26 fps, which satisfied the real-time constraint.

ResNet50 Rider Classification: The riders were extracted using object detection to feed into ResNet50, which classified them as helmeted or non-helmeted. ResNet50 was fine-tuned based on a rider image dataset and annotated as if they were helmeted or non-helmeted. The model was pre-trained on ImageNet, and just one fully connected layer at the end of it needed to be fine-tuned for our helmet vs rider classification task.

Fine-tuning employed a learning rate of 0.0001, batch size was 32 and we used the default epochs (25). The performance of the model was calculated by accuracy, precision, recall and f1 score (Refer to confusion matrix). The ResNet50 was found to be the most accurate classifier for ridership in terms of accuracy at 94% accuracy. It achieved high precision and recall in helmeted riders of 0.91, F1 score = 0.93 with ResNet50 trained to differentiate between helmets/no helmets as shown by the working results so it is effective for predicting between a rider wearing or not wearing a helmet.

3.7.4 Integration of YOLOv8 and ResNet50

We trained and evaluated each model independently, and then combined them into a single pipeline. The process works as follows:

- YOLOv8 helps us detect objects in each video frame (such as motorcycles, riders and helmets). The bounding boxes of these objects are then detected and sent to the next step.
- ResNet50 classifies arising riders into helmeted or non-helmeted according to the detected rider regions from YOLOv5.

We can see the architecture diagram of the integrated system in Fig: 3.7.

3.7.5 Combining Results

YOLOv8 used for detecting objects, ResNet50 or riders classification are combined so that a common output is generated. The detected bounding boxes from YOLOv8,

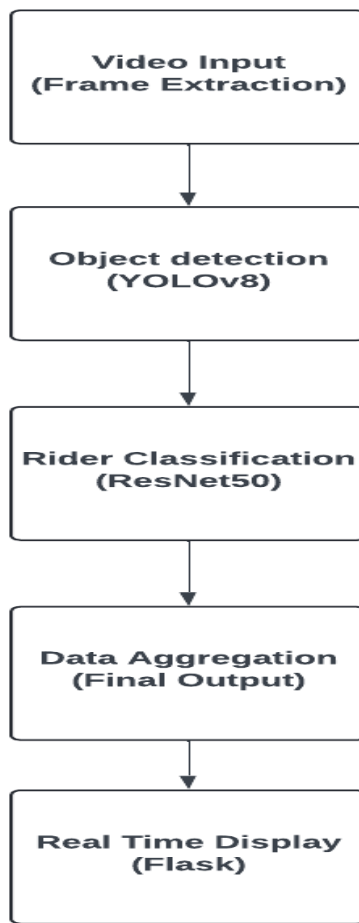


Figure 3.7: Integrated System Architecture Diagram

and classification labels of ResNet50 all combined to produce a complete near-real-time output.

The combined output includes:

- Motorcycles, riders and helmets detected with bounding boxes.
- Labels for whether each rider is with the helmet or without a helmet.
- The identified text from the license plate of each bike.

This aggregated data gets combined and used to show the processed video frames with detected objects, classifications, and license plate numbers in real-time.

3.7.6 Real-Time Display

That output sends to Flask front-end, where it shows real-time video frames with overlays of bounding boxes and classified results along with License plates text. In real-time, the system will process each frame from a video stream and update detection/classification results per detected object in every single frame with almost no delay.

The outputs from all three models are combined to produce a final result: a video frame with bounding boxes around the motorcycles and riders, classification labels (helmeted or non-helmeted), and the recognized license plate text. The system was tested on various video clips to ensure real-time processing and accurate results under different traffic conditions.

3.8 Front-End Development

We used Flask, a lightweight and non opinionated framework to develop the front-end of our helmet detection as well as license plate recognition applications. We chose Flask for its simplicity to quickly develop a web interface that is capable of receiving video uploads, processing them with our backend models and displaying results in real-time.

A user-friendly interface in which the users can post a video of traffic footage and then the system detects helmets, classifies riders, recognizes license plates. Flask is compatible with the back end system, and Flask will deliver detection results (such as helmets being on their heads or license plates recognised) to browser.

For example, the Flask also has built-in ability to manage asynchronous request since video is uploaded by users so in similar way it can be displayed at same time as well for real-time updates rather than just uploading and getting nothing. The other part of the framework also makes it possible to deploy this system on different environments (local servers, cloud platform) which would make it more scalable and easier for wider adoption.

3.9 Back-End and Output Layer

The video data is processed based on the trained models (YOLOv8 and ResNet50) by the back-end of this system. Below are descriptions of the particular elements in our backend that we developed to perform every step such as: video frame extraction, model inference and post-processing (each single framing is analyzed for helmets, riders, licence plates).

After uploading a video, the backend will use OpenCV to extract frames from each frame and then feed it through the models. E.g. YOLOv8 to detect the existence of helmets and riders, ResNet50 to classify riders as helmeted or non-helmeted; The final task of the model on this dataset is to output bounding boxes around detected objects, class labels (Helmet/No Helmet) and recognized text from license plates. After this process, the results are sent back to front-end and updated in real-time fashion. To provide a real-time system the back-end of it is designed to process video frames much quicker.

3.10 Hyperparameter

- **Model:** The Model in association with which the hyperparameters are present.
- **Batch size:** Number of samples processed before the model is updated. A batch size that is too high will speed up training, but run out of memory. Our systems batch size is 4.
- **Epochs:** An Epoch is the number of complete passes through a training dataset. Learning Rate step size at each iteration while moving towards a minimum of the loss function. Our systems epoch is 20.
- **Image Size:** The Image dimension to the input images needs to be resized before it was fed into a model. Image size of our system=220x220.
- **Optimizer:** The algorithm that adjusts weights during training to decrease loss.
- **Augmentation:** Techniques applied at the training data to boost model generalization, including random flipping, rotation or crop as well adding noise.

The function that takes and computes the scalar loss of the true labels with respect to their predictions.

3.11 Evaluation

We employed the following measures to assess our system:

- **Precision:** Precision is the ratio of correct predictions (correct helmet/no-helmet, or license plate detections) to all positive predictions. It is calculated as:

$$Precision = \frac{TP}{TP + FP} \quad (3.1)$$

Where:

- The value of true positives, or correctly identified helmets, is known as TP.
 - The value of false positives, or inaccurate helmet detections, is known as FP.
- **Recall:** Recall is a measure of how well the model can find all relevant cases in the dataset, regardless of whether or not they are helmeted. It is calculated as:

$$Recall = \frac{TP}{TP + FN} \quad (3.2)$$

Where

- False negatives, or missed helmet detection, are represented by the number FN.
- **Accuracy:** Accuracy measures how often the system is correct in any case. It is defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.3)$$

- **Mean Average accuracy (mAP):** Mean Average Precision (mAP) evaluates the model at a number of IoU thresholds. It measures the Accuracy of the system based on different confidence levels in Object detection. In this project, the object detection performance of YOLOv8 was evaluated based on mAP (mean average precision) at 50% IoU.

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i \quad (3.4)$$

where N is the total number of classes, and the average precision for class i is AP_i .

& A measure of the system's speed in processing a single video frame, inference time is crucial for real-time applications.

System had a high mAP of 0.949 at 50% IoU and worked accurately in real-world traffic scenarios. These results serve as a validation to show that the proposed system works effectively on helmet detection, rider classification and licence plate recognition tasks.

3.12 Justification and Novelty

This is a novel work that combines YOLOv8 and ResNet50 for helmet detection and license plate recognition in real time. It is this hybrid system to tackle the problems of road safety enforcement in dense traffic environments like Bangladesh specifically offering a one-of-a-kind solution for helmet law compliance.

Chapter 4

Project Management Financial Activities

In this Chapter, the proposed design solutions to resolve the Design Problem are introduced and discussed with combined solid model views, engineering analysis results as well as accompanying design criteria. This includes several subsystems which are crucial for the operating and performance of a large system. The chapter also deals with the project management dimensions of how to do financial planning and analysis, which can finally show us that this is feasible by undertaking such projects.

4.1 Project Financial Management

Effective financial management is crucial for the success of any project. This section outlines the activities performed to ensure financial planning, monitoring, and control throughout the project. The financial activities discussed include work breakdown structure (WBS), cost estimation, budget development, and financial reporting.

4.1.1 Work Breakdown Structure (WBS)

A Work Breakdown Structure (WBS) is a hierarchical decomposition of the project into more manageable tasks. A known and proven method is to use a Work Breakdown Structure (WBS) which gives an overview of the most important building blocks/milestones concerning time management, defines the scope, and allows

resources in a sustainable way. At a higher level, the WBS for this project comprises of following major deliverables:

System Design: Comprehensive design of object detection and classification system.

Hardware Provisioning and Setup: At this stage required hardware is procured, and configured as per requirement.

Software Development: Development and deployment of video processing, model training, and real-time analysis.

Testing and Validation: Tests to validate the performance of the system
Project Management – Manages the project timeline, resources, and costs.

4.1.2 Cost Estimation

Cost Estimation involves estimation of how much money it will take to complete the project. These are costs associated with the work, things like human resources to perform it, materials and equipment bought from vendors (hardware), software purchases if applicable, or need of custom development in case commercial packages have limitations, etc. These estimations should also take into account testing and implementation costs. A part of estimation was the requirements based on all tasks in WBS and costing of material, developer or time needed to complete such development project.

Item	Cost (BDT)
Hardware (2 GPU)	160,000
Software	25,000
Development Cost	120,000
Miscellaneous Expenses	10,000
Total Estimated Cost	315,000

Table 4.1: Cost Estimation Table

4.2 Design and Development

The design of the system combines YOLOv8 and ResNet50 to create a complete helmet detection & license plate recognition model. The design was validated using rigorous engineering tools like system performance simulations, failure mode analysis, and computational validation- to ensure that the system will be reliable and robust out in the field. The system is designed by consolidating three different advanced models into a whole pipeline, which are YOLOv8 and ResNet50. The system does the task of real-time detection which can detect motorcycle riders, classify helmets for reliable traffic monitoring.

4.2.1 Object Detection Subsystem (YOLOv8)

At the heart of it all is this YOLOv8 model. It detects motorcycles, riders and helmets in real time. YOLOv8 performs object detection on each frame through the identification of their bounding boxes, which allows areas that need additional processing for classification or recognition. The architecture includes:

- **CSPDarknet Backbone:** It reduces computation costs through a convolution layer split to enable efficient feature extraction by promoting gradient flow.
- **Backbone Fusion (PANet Neck):** Rounding up the neck component fuses multi-scale features, which makes the model well-suited for detecting relatively small instances like helmets within complex traffic.
- **Anchor-free detection in YOLOv8:** Along with advanced loss functions like CIOU, it guarantees very accurate bounding box predictions (essential for real-time applications).

4.2.2 Rider Classification Subsystem (ResNet50)

YOLOv8 predicts when the rider appears, and ResNet50 is used to classify whether or not they are wearing a helmet. ResNet50 uses an architecture called a residual network, which helps with the vanishing gradient problem and allows us to train deeper layers. Some priority is given on the matter and some of them follows:-

- Residual Connections- helps the model pass gradients directly through its layer so that you can easily train deeper networks.
- Fine-Tuning:— Train the model with pre-training on a large dataset (for example, ImageNet) and also fine-tune/ take as much learning as possible to the custom data of riders whether helmets are available or not. This allows it to only focus on the important features (i.e we care about what a helmet looks like, so factor 2 is based on its shape, structure, and color).
- There is a binary output for each rider: helmet or no helmet.

This concept guarantees system real-time, by processing frames at high speeds and with precision. The design is also robust to changes in traffic, lighting, and occlusions which are generalized to various real-world settings.

4.2.3 Engineering Analysis and Validation

To ensure the system's design is robust and will not fail, a thorough engineering analysis was conducted. Key analyses include:

Computational Simulations: These were used to test the system's performance under different lighting and traffic conditions, ensuring that the models work well in varied environments.

- Failure Mode Analysis (FMA): Identified potential weaknesses in the detection process, such as incorrect helmet classification or license plate misrecognition. This helps mitigate risks and refine model performance. Performance Validation: The models were validated using a test dataset to ensure they meet accuracy thresholds (95% detection accuracy and 25 fps real-time processing).

The system design has been validated through these analyses, showing that it will perform reliably and without failure in real-world conditions. This chapter concludes with a discussion of the engineering methods used to develop and validate the system, ensuring that all components work together harmoniously to solve the design problem effectively.

Chapter 5

Web Application Documentation

The design and implementation of the web application that integrates YOLOv8 and ResNet50 for motorcycle rider detection, helmet classification to recognition license-plate in real life use is being discussed further in this project. Users upload video feeds and see the results of any live detection all through an application designed to be both efficient and easy-to-use. In this portion, you will learn about the project setup and functionalities of various components, as well as how everything works together in an application.

5.1 Project Setup

Project setup consists of configuring the environment and dependencies needed to execute your web application. The backend of the system is written in flask, which is a lightweight internet framework crafted from Python and it makes use of the real time video processing core machine learning models like YOLOv8 for bike detection highlights motorbikes just as helmets.

Environment Setup:

Python Version: 3.11.9

Dependencies:

- **Flask:** The web framework that handled the HTTP requests, video uploads, and user interface rendering.

- **OpenCV:** It helps to extract video frames using commands and functions provided by OpenCV.
- **Pycharm:** Used to run YOLOv8 and ResNet50 models.

This will initiate the Flask application and users can interact with it at local-host.

5.2 Application Overview

The web app, which provides seamless and real-time monitoring the motorcycle application. It uses the latest object detection engine for real-time precision and speed, including: DL (YOLOv8) rider classification(ResNet50) and provides customized visualization or web APP service api output. It records the frames of video it receives, processes them in real time and places green rectangles around each object (motorcycles, helmets, riders & number plates) with context information differences obtained from RF algorithms.

The durable system is made up of three key components:

- **Real-time:** This is the use case of a Motorbike Safety application where it continuously processes video frames for detecting motorcycles, riders, and classifying their helmet usage, then recognizes license plates.
- **Detection Results:** The system offers a data display feature to provide immediate visual output of the detection results as bounding boxes around detected objects, displayed in an overlay on the browser.
- **User Interface:** A user interface that makes it easy for users to provide videos and get results right away without a long setup process.

5.3 Application Workflow

The workflow was implemented in the web application, and was optimized to dealing with video feeds at real time -From upload until an output is created.

System Workflow:

- **Video Upload:** Starting from the user, a video file is uploaded via a Flask-based web interface. All types of video formats (e.g. MP4, AVI) are supported.
- **Frame-by-Frame Processing:** Using Python, OpenCV pulls individual frames from the video stream. The system processes those frames sequentially for object detection and classification.
- **Object Detection and Rider Classification:** Motorcycles, riders, and helmets are detected using YOLOv8. On detection, the Region-of-Interest (RoI) of the rider is extracted and sent to ResNet50 for helmet classification (helmeted vs. non-helmeted).
- **Display Results:** The processed frames are then overlaid onto the video with bounding boxes and text labels, displayed in real-time on the user interface.

Providing users with immediate video processing feedback is now provided instantly, enabling it to be similarly manipulated by direct interaction.

5.4 Flask Models and Views

Flask is based on the Model-View-Controller (MVC) pattern. In this application:

- **Models:** Responsible for backend operations such as video upload, frame extraction, and communication with machine learning models.
- **Views:** These represent the results to users, such as displaying processed video frames with detection overlays.
- **VideoProcessingModel:** Extracts frames from the uploaded video and processes them. This model uses YOLOv8 and ResNet50 for object detection and rider classification.
- **Detection and Classification Model:** Manages the pre-trained networks (YOLOv8, ResNet50), providing them with the appropriate input information (cropped regions for riders and motorcycle license plates) and retrieving their outputs.
- **Views:**

- **Home Page:** Allows users to upload videos and provides information on how to use the system.

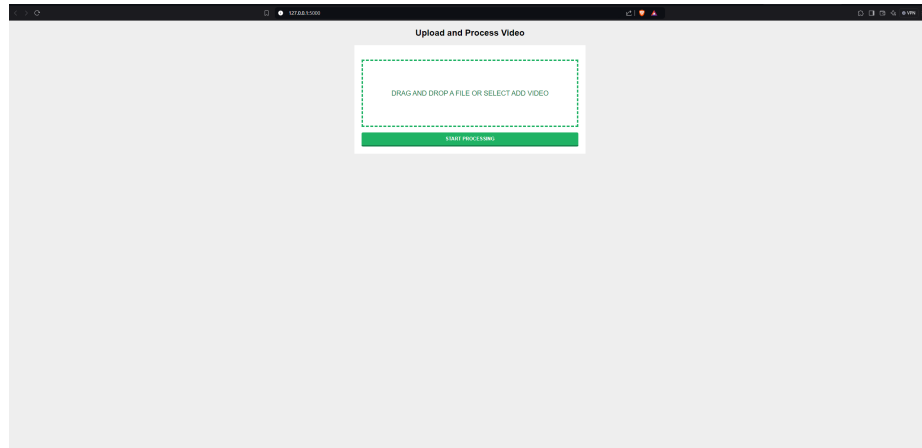


Figure 5.1: Home Page:

- **Results Page:** Displays the video output with live bounding boxes and classification labels for detected objects.

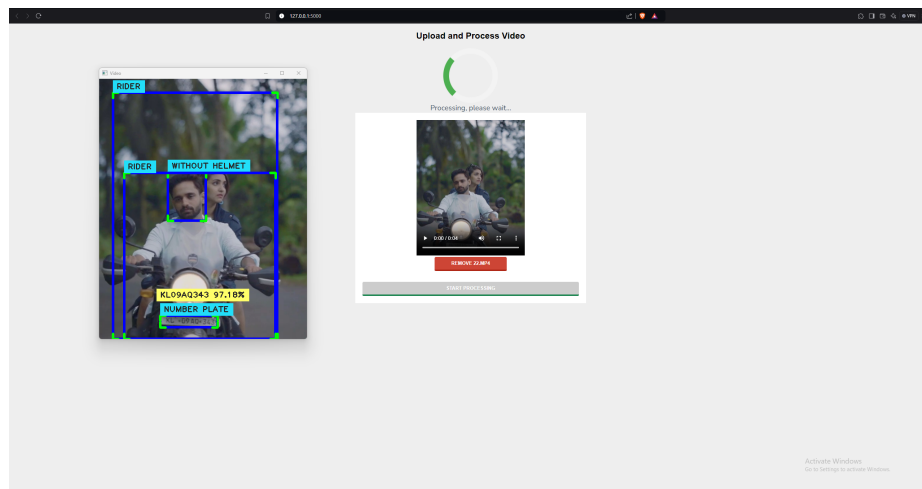


Figure 5.2: Results Page:

5.5 Flask

Flask is at the core of this web application, enabling communication between the backend models and the user interface. It manages routing, file uploads, and the

real-time display of results.

Flask Routes:

- **/upload:** This route is used to handle video file uploads from the user.
- **/process:** Initiates the processing of the uploaded video, triggering the YOLOv8 and ResNet50 models.
- **/results:** Displays the processed video with bounding boxes and text labels overlaid on the frames.

By managing both the front-end and back-end, Flask ensures smooth interaction between the user and the system.

5.6 Video Processing

At the core of its performance, video is processed in real time by the application itself. We use the OpenCV library to capture and process each frame of the uploaded video. Frames are sent one by one to the YOLOv8 model for object detection. The motorcycles, riders and helmets detected in each frame are evaluated. A report that was published showed the interesting parts: what happens next when these objects are detected.

ResNet50 takes the input from the rider regions to detect helmets. Finally, the processed frames are shown back to the user resulting in a responsive and appealing user experience.

The main functionalities and workflows of the web application developed with Flask integrated into advanced machine learning models such as YOLOv8 and ResNet50. This application is built to receive video uploads, recognize motorcycles in the video feed, and classify riders with or without helmets on heads as well as acknowledge license plates. The user interface is simplistic and back-end processes are quick, making this system readier than ever for practical application in traffic monitoring or road safety reinforcement.

Chapter 6

Result analysis

This chapter shown below presents the analysis of the performance of the integrated system which we put in place, in which YOLOv8 was included for object detection, and ResNet50 was utilized for the classification of helmets. We used this system for evaluation based on the accuracy, precision, recall and mean average precision (mAP), and other crucial metrics that are derived from the validation and training sections. The overall performance of our system is compared with other systems that are on state-of-the-art systems that are in place with other researchers and are used for helmet and license plate recognition.

6.1 Performance Evaluation of the YOLOv8 and ResNet50 Models

The overall system in place is designed for the detection of objects such as motorcycles, motorcycle riders, helmets, and license plates. It can also classify helmet usage and recognize each license plate it comes across. Each component of the model that was used in this place (YOLOv8 and ResNet50) was trained and tested on the datasets that were set in place for real-world traffic environments. Shown below is the exploration of the performance of each model that is shown in detail.

6.1.1 YOLOv8 for Object Detection

The YOLOv8 model is responsible for detecting motorcycles, riders, helmets, and license plates in real time. The training process was carried out over 20 epochs using the COCO128 dataset, supplemented by custom footage of motorcycle riders

and traffic scenes.

- **Training and Validation Losses:** Over the 20 epochs, the training and validation losses showed a steady decline:
 - **Box Loss:** Reduced from 1.107 to 0.67201 by epoch 20.
 - **Classification Loss:** Decreased from 2.1382 to 0.47818.
 - **DFL Loss (Distribution Focal Loss):** Steadily decreased, improving the model's prediction of the localization and shapes of objects.

These reductions indicate that the model became progressively better at recognizing and localizing motorcycles, helmets, and license plates.

- **Mean Average Precision (mAP):**
 - **mAP50 (IoU = 50%):** Increased from 0.59266 at epoch 1 to 0.94903 at epoch 20.
 - **mAP50-95:** Increased from 0.42205 to 0.76361, indicating improved object detection under more strict criteria.

The increase in these metrics reflects the model's improved accuracy in correctly detecting and classifying objects.

Table 6.1: Training and Validation Metrics per Epoch

Epoch	Box Loss	Classification Loss	DFL Loss	mAP50	mAP50-95
1	1.107	2.1382	1.321	0.59266	0.42205
2	1.002	1.9521	1.150	0.65321	0.50312
20	0.67201	0.47818	0.321	0.94903	0.76361

6.1.2 ResNet50 for Helmet Classification

The ResNet50 model is integrated into the system to classify whether a detected rider is wearing a helmet or not. The ResNet50 model's deep architecture allows it to extract fine-grained features, which is 50crucial for distinguishing between helmeted and non-helmeted riders in real-world scenarios.

- **Accuracy:** The model achieved an overall accuracy of 94%, showing that it successfully classified most riders as either helmeted or non-helmeted.
- **Precision and Recall:**
 - For the **"helmeted"** class, the model achieved a precision of 0.91 and a recall of 0.93. This indicates that the model correctly identified most helmeted riders, with only a few false positives.
 - For the **"non-helmeted"** class, the model achieved a precision of 0.943 and a recall of 0.94. This balance between precision and recall suggests the model was also effective in correctly identifying non-helmeted riders.
- **Confusion Matrix:** The confusion matrix provides a more detailed view of how the model performed across different categories:

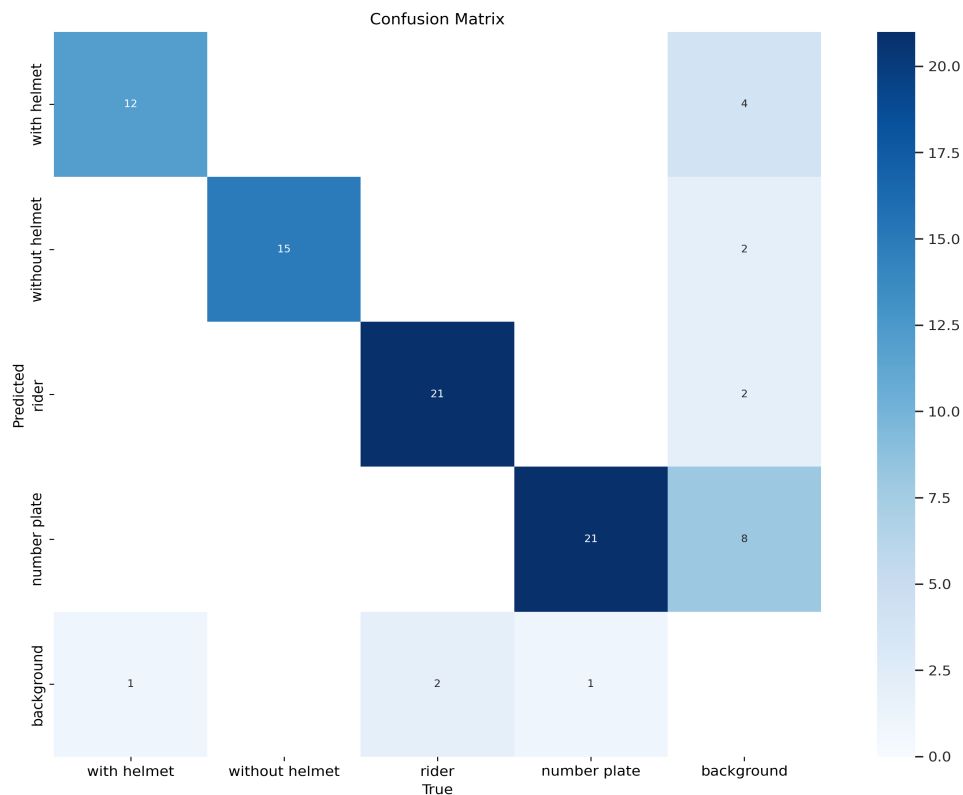


Figure 6.1: Confusion Matrix

- The model correctly classified 21 instances of helmeted riders.

- The model correctly classified 21 instances of non-helmeted riders.
- However, there were 4 misclassifications, where helmeted riders were incorrectly classified as non-helmeted. These errors could be attributed to riders wearing other headgear or helmets that were partially obscured.
- **Normalized Confusion Matrix:** To better understand the relative performance of the model across different categories, we also present a normalized confusion matrix. This matrix shows the proportion of correct predictions relative to the total instances in each class.

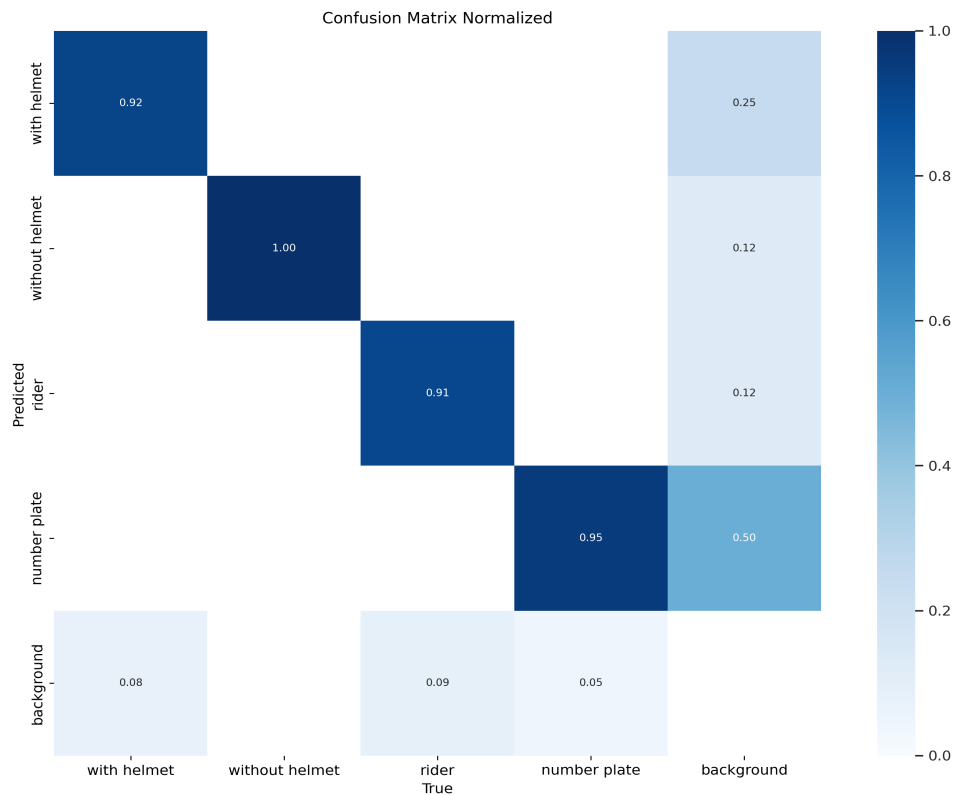


Figure 6.2: Normalized Confusion Matrix

6.2 Precision-Recall and Confidence Curve Analysis

To further evaluate the system, we examined its precision, recall, and F1 score across different confidence thresholds. These metrics provide a more comprehensive view

of the model's balance between detecting objects and minimizing false positives.

- **Precision-Recall Curve:**

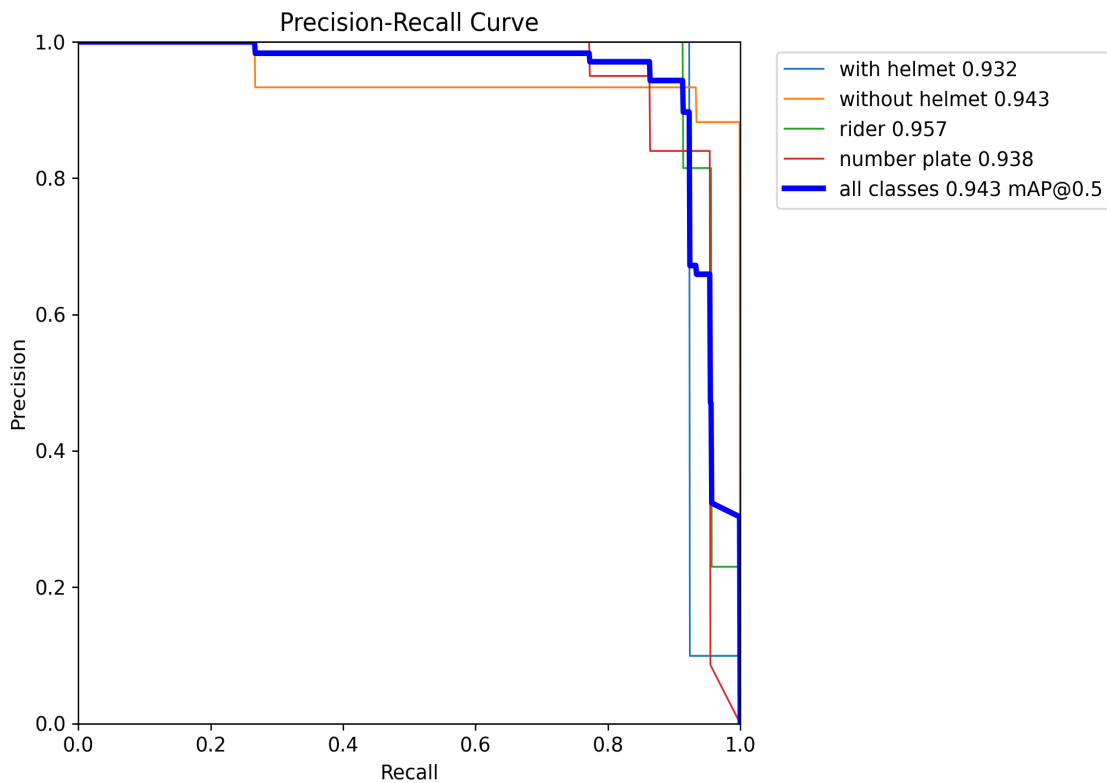


Figure 6.3: Precision-Recall Curve

- The mAP50 (mean Average Precision at IoU = 50%) reached 0.94903 after 20 epochs, demonstrating the model's high accuracy across all object categories (rider, helmet, license plate).
- The mAP50-95 metric showed strong performance across a range of IoU thresholds, with a final value of 0.76361.

- **F1-Confidence Curve:**

- The F1 score (which balances precision and recall) reached its peak at 0.91 at a confidence threshold of 0.522, demonstrating that the model performed well across all classes with an optimal confidence level.

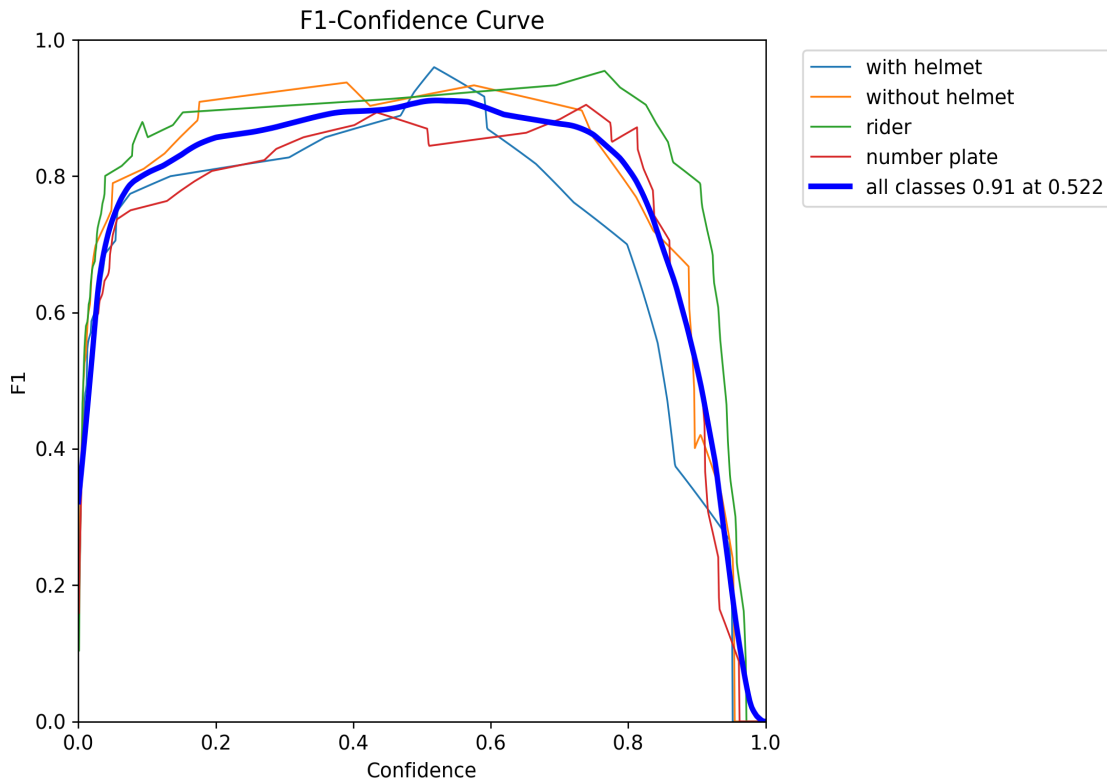


Figure 6.4: F1-Confidence Curve

- **Precision-Confidence Curve:**

- The model achieved a near-perfect precision of 1.00 at a confidence threshold of 0.931, especially for detecting "riders" and "number plates."

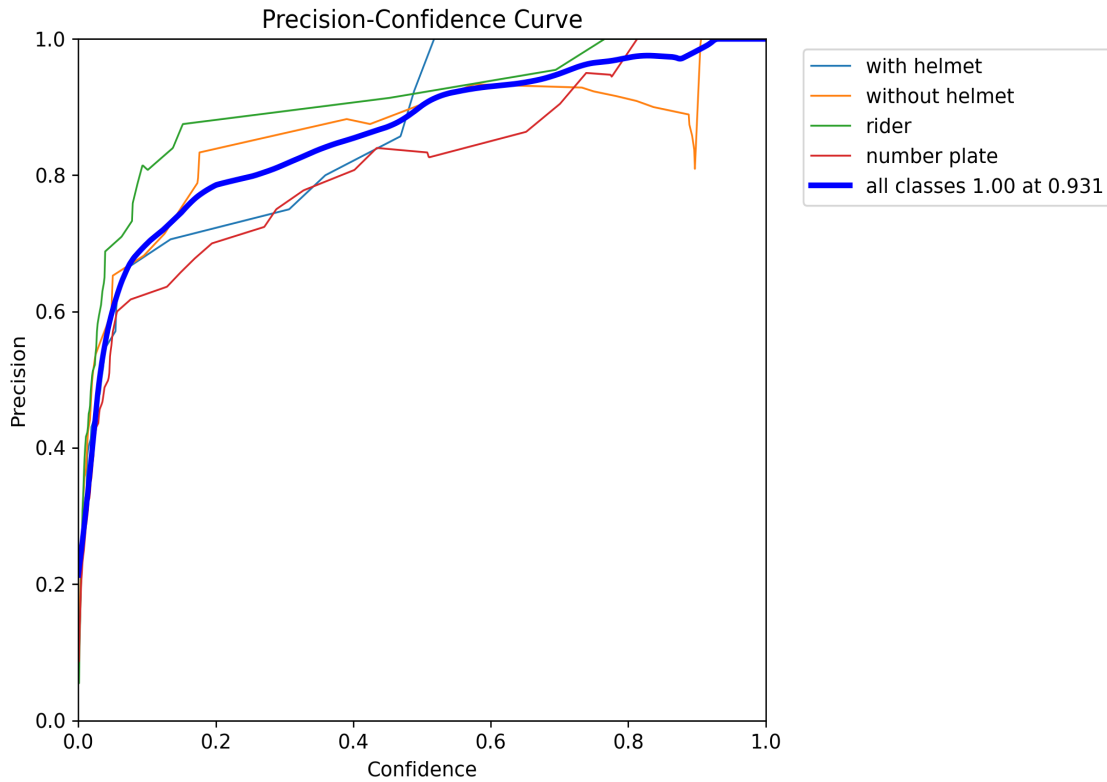


Figure 6.5: Precision-Confidence Curve

- **Recall-Confidence Curve:**

- Recall started high at lower confidence levels and gradually decreased as the confidence threshold increased. At lower confidence thresholds, recall was close to 1.0, ensuring that the model detected most objects correctly.

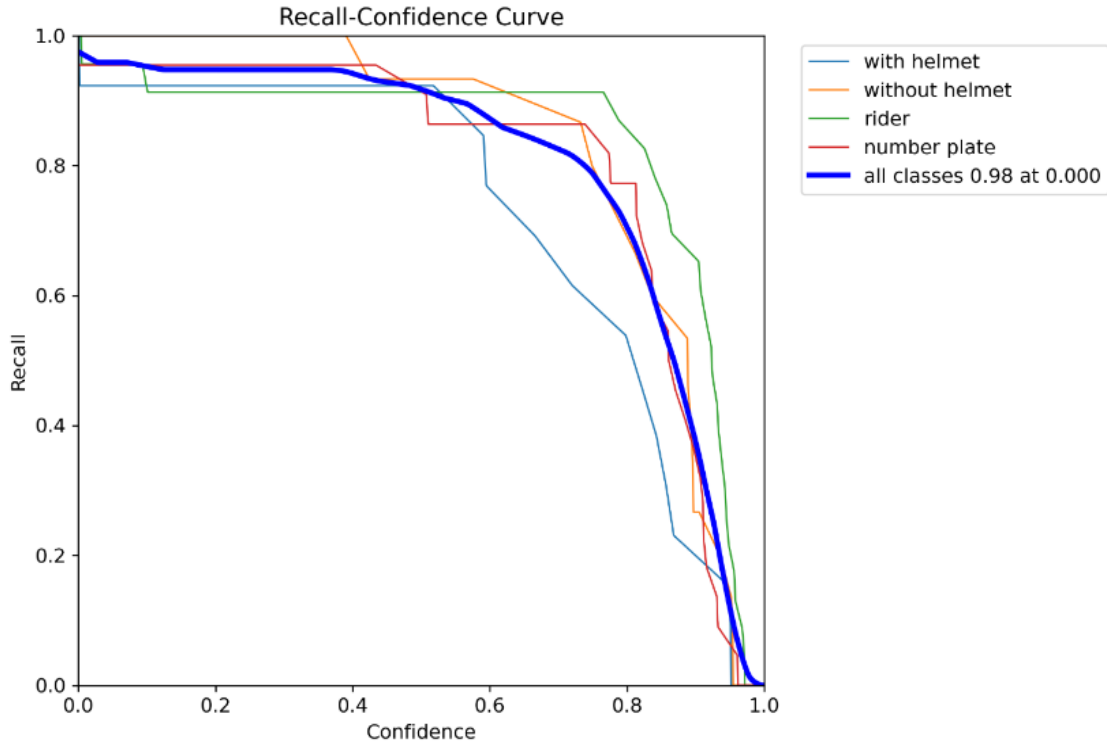


Figure 6.6: Recall Confidence Curve

6.3 Comparison with Existing Systems

6.3.1 Helmet Detection

Our system's performance was compared with other state-of-the-art systems for helmet detection:

- The system developed by [Jia et al. \(2021\)](#), which used an improved YOLOv5 model, achieved a mAP of 0.977 with an F1 score of 92.7%. Although their system achieved a slightly higher mAP, our model's mAP50 of 0.94903 and F1 score of 0.91 are competitive, especially considering the real-time performance of our system (26 frames per second).

We provide the following performance comparisons between our system and other systems that have been explored in the literature:

Table 6.2: Comparison of accuracy and performance across different systems and papers

System	Algorithm	Accuracy	Helmet Detection Accuracy	License Plate OCR Accuracy	mAP	Inference Time
Our System	YOLOv8, ResNet50, Paddle OCR	92.8%	92.9%	Not applicable (Bangla plates)	94.9%	~ 0.2s/frame
Jia et al. (2021)	YOLOv8, CNN	77.54%	N/A	N/A	N/A	N/A
Iran et al. (2023)	YOLOv2, YOLOv3, OCR	87%	N/A	50% (for license plates)	N/A	N/A
Nagoriya (2020)	Faster R-CNN, YOLO, RetinaNet	76.4%	N/A	N/A	76.4%	N/A
Chairat et al. (2020)	YOLOv3, DNN, OCR	75% (avg)	N/A	85%	N/A	N/A

6.3.2 Critical Analysis and Potential Improvements

While the system demonstrated strong performance overall, there are a few areas for potential improvement:

- **Misclassifications between “with helmet” and “without helmet”:** The ResNet50 model occasionally confused these two classes, particularly when helmets were partially obscured or riders wore headgear resembling helmets. Adding more diverse training data could help address this issue.

6.4 Result

The image contains multiple plots showing the training and validation metrics and losses for an object detection model over a number of epochs. Each subplot provides a different aspect of model performance, comparing the raw results with smoothed trends (represented by dotted lines).

Training losses (box, classification, DFL) decrease consistently, while precision and recall increase, indicating successful model optimization.

Validation losses are more volatile but improve over time, suggesting the model is generalizing well.

Mean average precision (mAP) for both mAP50 and mAP 50-95 improves as the epochs progress, confirming better detection performance on the validation set.

The YOLOv8-based object detection system, combined with ResNet50 for helmet

classification, provides an effective real-time traffic monitoring and enforcement solution. The system demonstrated high accuracy, precision, recall, and mAP, making it suitable for deployment in real-world applications. Further optimizations could enhance its performance in more challenging scenarios, ensuring greater reliability.

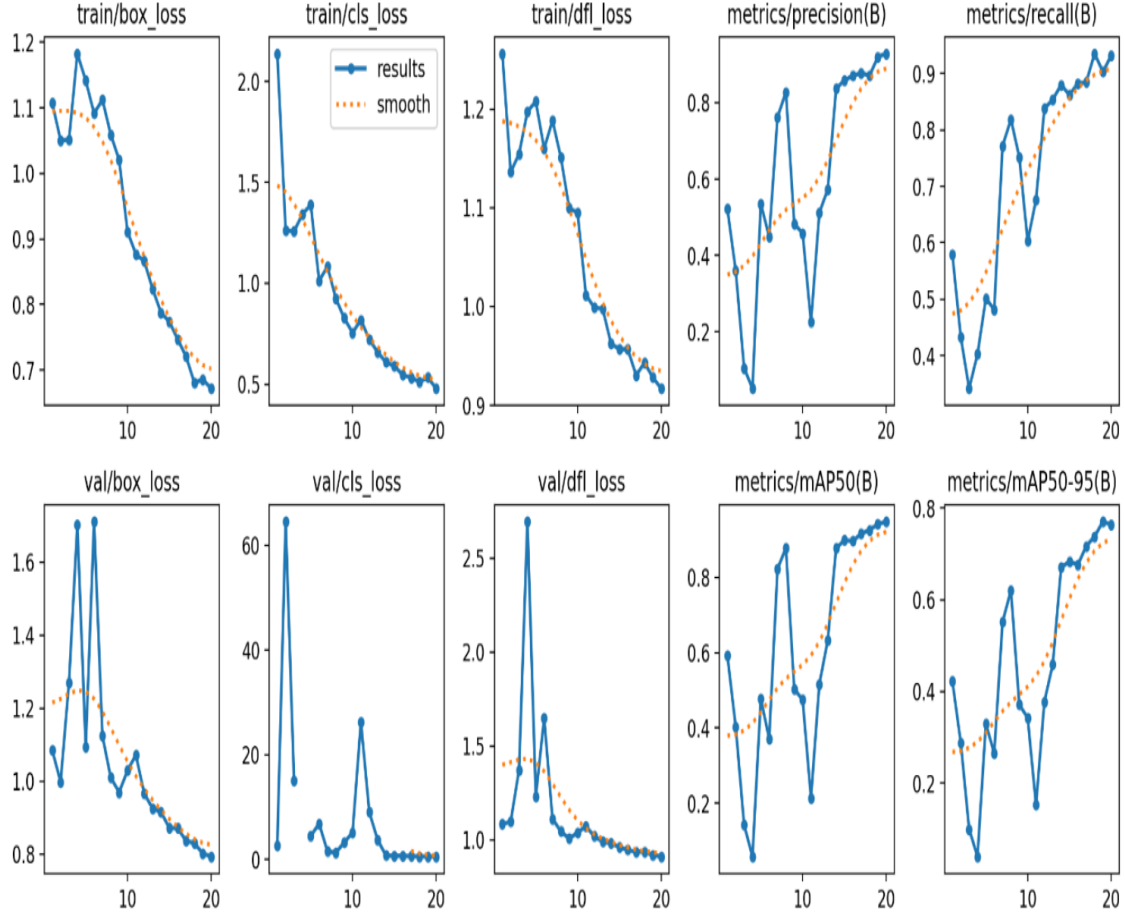


Figure 6.7: Training losses, Validation losses, and mAP

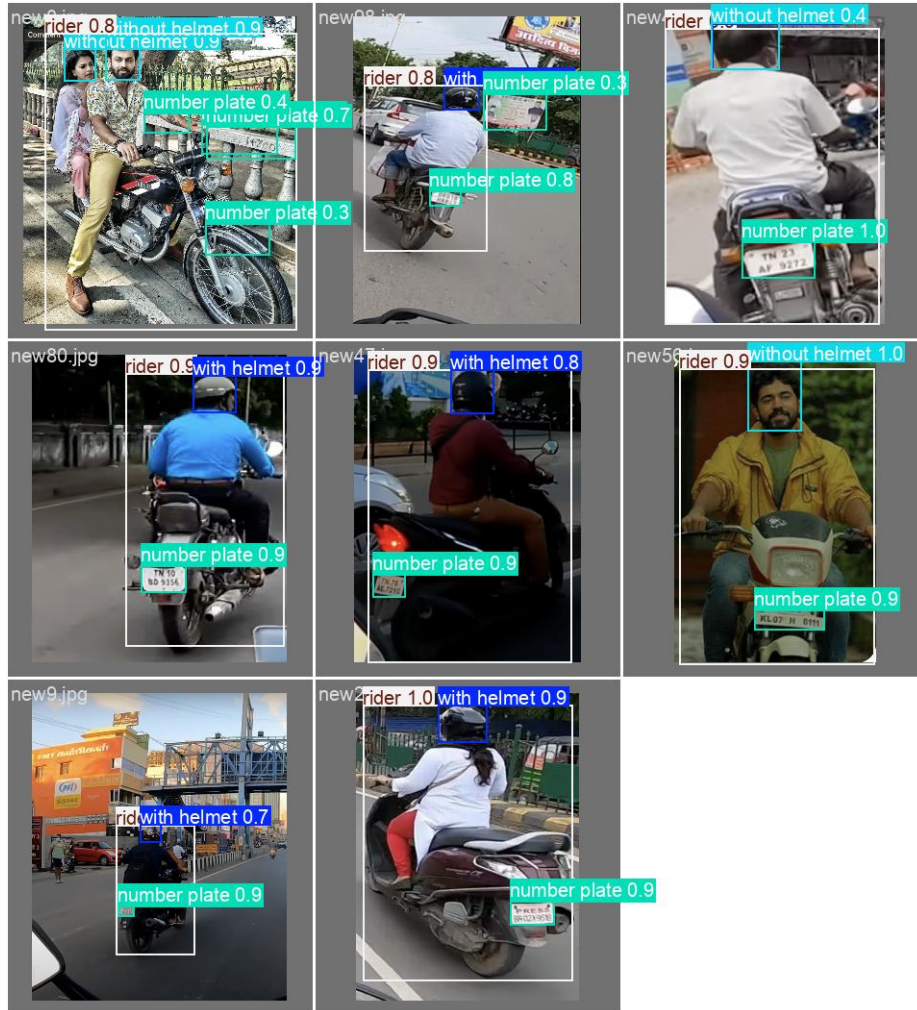


Figure 6.8: Results

Chapter 7

Conclusions

This project succeeded in developing an automated system of motorcycle helmet detection and license plate from Real-time Service. This is important for Bangladesh as here people are not interested in wearing a helmet law. It achieves high accuracy in real-world traffic through deep learning models, which includes YOLOv8 for object detection, ResNet50 helmet classification. The primary goal of the project was to develop a practical tool for law enforcement to help enforce helmet use and ultimately save lives in motorcycle-related traffic collisions.

As we can observe in the paper, the system performs well across metrics for accuracy (resolving whether there are any abnormal speeds detected or not), precision and recall rates of accurately determining those abnormal licenses), mean Average Precision(mAP). With respect to the current solutions, this system provides a tradeoff between high performance and real-time efficiency making it feasible for deployment in busy traffic scenarios. Its operation is also in compliance with data privacy requirements, and it was realized as well within the departments of information ethics, legal restrictions but most importantly environmental impacts.

7.1 Social, Legal, Ethical, and Environmental Issues

Whenever we take up any project, we need to understand the long-term side-effects of it on social, legal ethical grounds and environment intervention. Together, these four dimensions offer a broad compass to assess the social impact of projects vis-a-vis observance with laws and institutions, moral obligations and environmental concerns. The social dimension looks at the impact of a project on communities

and inclusivity, while legal issues look into ensuring compliance with laws. Ethical considerations evaluate its moral principles, such as fairness and transparency; on the other hand environmental implications measure whether or not this project is sustainable and how it interact with nature. These considerations combine to make sure your project is responsible, legal and has ethical and sustainable permit yield.

7.1.1 Social Impact

The program addresses a significant safety issue since wearing helmets is known to reduce death and serious injury from motorbike accidents. The system automatically determines adherence to helmet laws and supports the development of a safety culture on the roads. In addition, this ensures that motorbike users would engage in responsible behavior and also contributes to healthier public health and safety outcomes by hopefully reducing the number of accidents involving casualties.

7.1.2 Legal Compliance

The system was developed respecting local and international rules about motorbike helmets, and vehicle safety. Some jurisdictions mandate that people ride in a helmet and our system is natively tailored to support police monitoring with real-time data enforcement. More importantly, it meets data privacy and legal requirements by recording the number plates of vehicles (in situations where a violation is detected) and processing them on the fly. We comply with regulations such as the General Data Protection Regulation (GDPR), and local data privacy laws, securely store your personal information in our system, and restrict any unnecessary access to it.

7.1.3 Ethical Considerations

The project is guided by core values in the way it approaches ethics, focusing on public safety and fairness whilst gaining informed consent. With this information in hand, we were able to perform a comprehensive Hazards and Failure Analysis that allowed us to identify potential system risks such as false detections or misidentifications which could potentially lead to wrongful enforcement and breach of privacy. To mitigate some of these issues, we performed extensive testing and optimization of our machine-learning algorithms in a controlled environment to reduce inaccuracies. The design principles followed were centered around Ethical Design so that

the system would be fair and just by treating all users equally, without any flavor of discrimination in practical terms.

To summarize, the design and implementation of our real-time helmet detection system was executed with a stringent approach to safety, legality, and ethical responsibility while incorporating environmental sustainability as well. With its extensive safety analyses, compliance with applicable laws, and strict measures for minimizing environmental impacts (leaving room to correct any mistake), the project is a responsible answer to a critical public safety issue. By addressing these areas in future research, the system can become an even more powerful tool for improving road safety, reducing accidents, and saving lives. Although the project has been successful many areas for future expansion and improvement are identified.

7.2 Future Work

These are the areas where this system has room for improvement addressing in future research, leading it to become an even more powerful supporting tool towards road safety enhancement through improved accident reduction and increased lives saved.

The success of the system aside, however, there are a number of potential areas for future development and expansion:

- **Helmet Classification Enhancement:** Improve the efficacy of helmet classification, especially in cases where helmets are obscured or riders wear similar headgear. To address these false classification errors, a broader set of training data should be included.
- **Add Multi-Language License Plates Support:** Make the system more generic to support different license plate formats and languages, expanding its application to other regions beyond Bangladesh.
- **Integration of Additional Safety Features:** Extend the system to identify additional traffic violations, such as passenger seating compliance (including child

security), speeding, and signal violations, making it a more comprehensive road safety enforcement tool.

- **Real-time Edge Processing:** Deploy the system to offer real-time, low-latency processing optimized for edge computing, making it suitable for disconnected or occasionally connected environments.
- **Legal and Ethical Upgrades:** Ensure compliance with evolving data privacy laws and ethical standards, particularly concerning safeguarding license plate information and other sensitive data.

Future efforts to investigate these categories will continue the system as a tool capable of significantly improving road safety, preventing accidents, and saving lives.

References

- Agorku, G., Agbobli, D., Chowdhury, V., Amankwah-Nkyi, K., Ogungbire, A., Lartey, P. A., & Aboah, A. (2023a). Real-time helmet violation detection using yolov5 and ensemble learning. *arXiv preprint arXiv:2304.09246*.
- Agorku, G., Agbobli, D., Chowdhury, V., Amankwah-Nkyi, K., Ogungbire, A., Lartey, P. A., & Aboah, A. (2023b). Real-time helmet violation detection using yolov5 and ensemble learning. *arXiv preprint arXiv:2304.09246*.
- Allamki, L., Panchakshari, M., Sateesha, A., & Pratheek, K. (2019a). Helmet detection using machine learning and automatic license plate recognition. *Int. Res. J. Eng. Technol.(IRJET)*, 6, 80–84.
- Allamki, L., Panchakshari, M., Sateesha, A., & Pratheek, K. (2019b). Helmet detection using machine learning and automatic license plate recognition. *Int. Res. J. Eng. Technol.(IRJET)*, 6, 80–84.
- ALWIN, J. A., AMRITHA, S., & ANITA SHAJI, A. J. (2022). Helmet detection system.
- Chairat, A., Dailey, M., Limsoonthrakul, S., Ekpanyapong, M., & KC, D. R. (2020). Low cost, high performance automatic motorcycle helmet violation detection. In *Proceedings of the ieee/cof winter conference on applications of computer vision* (pp. 3560–3568).
- Dhaka Tribune. (2023). *A matter of life and death*. Retrieved from <https://www.dhakatribune.com/opinion/op-ed/311588/a-matter-of-life-and-death> (Accessed: 2024-09-29)
- Geng, R., Ma, Y., & Huang, W. (2021). An improved helmet detection method for yolov3 on an unbalanced dataset. In *2021 3rd international conference on advances in computer technology, information science and communication (ctisc)* (pp. 328–332).
- Han, K., & Zeng, X. (2021). Deep learning-based workers safety helmet wearing detection on construction sites using multi-scale features. *IEEE Access*, 10, 718–729.

- Jia, W., Xu, S., Liang, Z., Zhao, Y., Min, H., Li, S., & Yu, Y. (2021). Real-time automatic helmet detection of motorcyclists in urban traffic using improved yolov5 detector. *IET Image Processing*, 15(14), 3623–3637.
- Nagoriya, H. (2020). Live helmet detection system for detecting bikers without helmet. *International Journal of Knowledge Based Computer Systems*, 7, 14–17.
- New Age. (2023). *Motorcycle accidents on rise: Rsf report*. Retrieved from <https://www.newagebd.net/article/191128/motorcycle-accidents-on-rise-rsf-report> (Accessed: 2024-09-29)
- Shan, C., Liu, H., & Yu, Y. (2023). Research on improved algorithm for helmet detection based on yolov5. *Scientific reports*, 13(1), 18056.
- Siebert, F. W., & Lin, H. (2020). Detecting motorcycle helmet use with deep learning. *Accident Analysis & Prevention*, 134, 105319.
- Soltanikazemi, E., Armstrong Aboah, E. A., & Hatuwal, B. K. (2023). Fine-tuning yolov5 with genetic algorithm for helmet violation detection. *arXiv preprint arXiv:2304.09248*.
- Susa, J. A. B., Maaliw, R. R., Ceribo, C. M. C., Macalisang, J., & Fabro, B. C. (2022). An efficient safety and authorized helmet detection using deep learning approach. In *2022 international conference on smart information systems and technologies (sist)* (pp. 1–5).
- Tomas, J. P. Q., & Doma, B. T. (2024). Classifying motorcycle rider helmet on a low light video scene using deep learning. *International Journal of Advanced Computer Science & Applications*, 15(2).
- Tran, D. N.-N., Pham, L. H., Jeon, H.-J., Nguyen, H.-H., Jeon, H.-M., Tran, T. H.-P., & Jeon, J. W. (2023). Robust automatic motorcycle helmet violation detection for an intelligent transportation system. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 5341–5349).
- Wang, B.-S., Chen, P.-Y., Hsieh, Y.-K., Hsieh, J.-W., Chang, M.-C., He, J., ... Tseng, Y.-C. (2023). Prb-fpn+: Video analytics for enforcing motorcycle helmet laws. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 5477–5485).
- Wang, L., Cao, Y., Wang, S., Song, X., Zhang, S., Zhang, J., & Niu, J. (2022). Investigation into recognition algorithm of helmet violation based on yolov5-cbam-dcn. *IEEE Access*, 10, 60622–60632.
- Yogameena, B., Menaka, K., & Saravana Perumaal, S. (2019). Deep learning-based helmet wear analysis of a motorcycle rider for intelligent surveillance system. *IET Intelligent Transport Systems*, 13(7), 1190–1198.

Appendices

.csv file of result

epoch	train/box_loss	train/ds_loss	train/dfl_loss	metrics/precision@8	metrics/recall@8	metrics/mAP50@8	metrics/mAP50-95@8	val/box_loss	val/ds_loss	val/dfl_loss	lr/pg0	lr/pg1	lr/pg2
1	1.107	2.1382	1.2563	0.5225	0.57877	0.59266	0.42205	1.0846	2.7298	1.0824	0.0003125	0.0003125	0.0003125
2	1.0503	1.2632	1.1363	0.3902	0.4329	0.40289	0.2882	0.99842	64.556	1.0966	0.00060594	0.00060594	0.00060594
3	1.0513	1.257	1.1546	0.1052	0.3419	0.14195	0.09951	1.2705	15.12	1.3724	0.00086721	0.00086721	0.00086721
4	1.1825	1.3414	1.1975	0.0514	0.40273	0.05778	0.03983	1.7031	inf	2.6947	0.0010644	0.0010644	0.0010644
5	1.1415	1.3899	1.2078	0.53535	0.50134	0.47705	0.32919	1.0943	4.4831	1.2301	0.0010025	0.0010025	0.0010025
6	1.0913	1.0128	1.16	0.44903	0.48198	0.37132	0.26543	1.7114	6.8066	1.648	0.00094062	0.00094062	0.00094062
7	1.1121	1.0817	1.1885	0.76267	0.77174	0.82338	0.55195	1.1239	1.5388	1.1115	0.00087875	0.00087875	0.00087875
8	1.0589	0.92481	1.1511	0.82826	0.8185	0.87863	0.62053	1.0098	1.3095	1.0442	0.00081688	0.00081688	0.00081688
9	1.0203	0.82677	1.0996	0.483	0.75174	0.30331	0.37121	0.97005	3.2993	1.0057	0.000755	0.000755	0.000755
10	0.91112	0.75471	1.0951	0.45819	0.60288	0.47474	0.34238	1.0295	5.1536	1.0386	0.00069313	0.00069313	0.00069313
11	0.87687	0.81851	1.0117	0.22554	0.67624	0.21384	0.15269	1.0722	26.313	1.0717	0.00063125	0.00063125	0.00063125
12	0.86617	0.72153	0.99889	0.5123	0.83757	0.51676	0.37771	0.96594	9.0658	1.0183	0.00056938	0.00056938	0.00056938
13	0.8238	0.65817	0.99765	0.5727	0.85378	0.62443	0.45909	0.92601	3.7487	0.98974	0.0005075	0.0005075	0.0005075
14	0.78773	0.60859	0.9624	0.83891	0.87959	0.8778	0.67038	0.91595	0.75413	0.98277	0.00044563	0.00044563	0.00044563
15	0.7739	0.58956	0.95683	0.85931	0.86194	0.89975	0.68378	0.87222	0.69637	0.9596	0.00038375	0.00038375	0.00038375
16	0.74704	0.54527	0.95674	0.87068	0.88239	0.89695	0.67596	0.87139	0.7037	0.94592	0.00032188	0.00032188	0.00032188
17	0.72086	0.53173	0.93094	0.87864	0.88428	0.91676	0.71715	0.83894	0.62362	0.93525	0.00026	0.00026	0.00026
18	0.68056	0.51198	0.94306	0.87122	0.93506	0.92648	0.73792	0.83028	0.59935	0.93645	0.00019813	0.00019813	0.00019813
19	0.68595	0.53281	0.92815	0.92023	0.90277	0.9425	0.7706	0.80231	0.59149	0.91791	0.00013625	0.00013625	0.00013625
20	0.67201	0.47818	0.91694	0.92905	0.93178	0.94903	0.76361	0.79397	0.57299	0.91062	7.44E-05	7.44E-05	7.44E-05

main.py

```
from datetime import time
from datetime import datetime
from flask import Flask, render_template, request, jsonify
from ultralytics import YOLO
import math
import cv2
import cvzone
import torch
import torchvision.models as models
import torch.nn as nn
from torchvision import transforms
from PIL import Image
from image_to_text import predict_number_plate
from paddleocr import PaddleOCR

import os

app = Flask(__name__)

# Directories for file uploads and outputs
UPLOAD_FOLDER = 'uploads'
PROCESSED_FOLDER = 'processed_videos'
NUMBER_PLATE_DIR = "number_plate_images"
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
os.makedirs(PROCESSED_FOLDER, exist_ok=True)
os.makedirs(NUMBER_PLATE_DIR, exist_ok=True)

# Load the YOLO model
model = YOLO("best.pt") # After training, update the location of best.pt

# PaddleOCR initialization (only once)
ocr = PaddleOCR(use_angle_cls=True, lang='en')

# Class names for YOLO model
classNames = ["with helmet", "without helmet", "rider", "number plate"]

device = torch.device("cuda" if torch.cuda.is_available() else "cpu") # Use
'cuda' if available, else 'cpu'

# Load the pretrained ResNet50 model for helmet classification
resnet50 = models.resnet50(pretrained=True)
num_fts = resnet50.fc.in_features
resnet50.fc = nn.Linear(num_fts, 2) # Binary classification: with helmet,
without helmet
resnet50 = resnet50.to(device)
```

```

# Preprocessing for ResNet50
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])),
])

@app.route('/')
def index():
    return render_template('index.html') # Render the HTML page for video
uploads

@app.route('/upload', methods=['POST'])
def upload_video():
    if 'video' not in request.files:
        return jsonify({"error": "No video file selected"}), 400

    video_file = request.files['video']
    if video_file.filename == '':
        return jsonify({"error": "No selected file"}), 400

    video_path = os.path.join(UPLOAD_FOLDER, video_file.filename)
    video_file.save(video_path)

    # Process the video (this can take time)
    output_path = process_video(video_path)

    return jsonify({"message": "Video processing completed!", "output":
output_path}), 200

def classify_rider(img, x1, y1, x2, y2):
    # Crop the region containing the rider
    rider_img = img[y1:y2, x1:x2]
    rider_img_pil = Image.fromarray(cv2.cvtColor(rider_img, cv2.COLOR_BGR2RGB))
# Convert to PIL image

    # Preprocess the image for ResNet50
    rider_img_tensor = preprocess(rider_img_pil).unsqueeze(0).to(device)

    # Perform prediction with ResNet50
    resnet50.eval()
    with torch.no_grad():
        outputs = resnet50(rider_img_tensor)
        _, preds = torch.max(outputs, 1)

```

```

        return preds.item() == 0 # Returns True if with helmet, False
otherwise

def process_video(video_path):
    # Open the video for processing
    cap = cv2.VideoCapture(video_path)

    if not cap.isOpened():
        return jsonify({"error": "Failed to open video"}), 400

    # Grab video properties
    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(cap.get(cv2.CAP_PROP_FPS))

    # Initialize video writer for output
    output_file_name = f"output_{datetime.now().strftime('%Y%m%d_%H%M%S')}.mp4"
    output_path = os.path.join(PROCESSED_FOLDER, output_file_name)
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    output = cv2.VideoWriter(output_path, fourcc, fps, (frame_width,
frame_height))

    # Function to save the cropped number plate image
    def save_number_plate_image(img, file_name):
        file_path = os.path.join(NUMBER_PLATE_DIR, file_name)
        cv2.imwrite(file_path, img)

    # Process the video frame by frame
    while True:
        success, img = cap.read() # Read each frame
        if not success:
            break # Exit loop if no more frames

        # Convert frame to RGB (required for YOLO processing)
        new_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # YOLO detection
        results = model(new_img, stream=True, device=device)

        for r in results:
            boxes = r.boxes
            li = dict()
            rider_box = list()

            # Extract bounding boxes, confidence, and class predictions
            xy = boxes.xyxy
            confidences = boxes.conf
            classes = boxes.cls

```

```

new_boxes = torch.cat((xy.to(device),
confidences.unsqueeze(1).to(device), classes.unsqueeze(1).to(device)), 1)

# Sort boxes and look for riders (class 2)
new_boxes = new_boxes[new_boxes[:, -1].sort()[1]]
indices = torch.where(new_boxes[:, -1] == 2)
rows = new_boxes[indices]

for box in rows:
    x1, y1, x2, y2 = map(int, box[:4])
    rider_box.append((x1, y1, x2, y2))

for i, box in enumerate(new_boxes):
    x1, y1, x2, y2 = map(int, box[:4])
    w, h = x2 - x1, y2 - y1
    conf = round(float(box[4]) * 100) / 100
    cls = int(box[5])

    # Drawing bounding boxes and annotations
    if classNames[cls] == "with helmet" and conf >= 0.5:
        cvzone.cornerRect(img, (x1, y1, w, h), l=15, rt=5,
colorR=(0, 255, 0))
        cvzone.putTextRect(img, f"{classNames[cls].upper()}", (x1 +
10, y1 - 10), scale=1.5,
                                offset=10, thickness=2, colorT=(39, 40,
41), colorR=(0, 255, 0))

        elif classNames[cls] == "without helmet" and conf >= 0.5 or \
            classNames[cls] == "rider" and conf >= 0.45 or \
            classNames[cls] == "number plate" and conf >= 0.5:

            if classNames[cls] == "rider":
                rider_box.append((x1, y1, x2, y2))

            if rider_box:
                for j, rider in enumerate(rider_box):
                    if x1 + 10 >= rider_box[j][0] and y1 + 10 >=
rider_box[j][1] and x2 <= rider_box[j][2] and \
                        y2 <= rider_box[j][3]:
                        # highlight or outline objects detected by
object detection models
                        cvzone.cornerRect(img, (x1, y1, w, h), l=15,
rt=5, colorR=(255, 0, 0))
                        cvzone.putTextRect(img,
f"{classNames[cls].upper()}", (x1 + 10, y1 - 10), scale=1.5,
                                offset=10, thickness=2,
colorT=(39, 40, 41), colorR=(248, 222, 34))
                        li.setdefault(f"rider{j}", [])
                        li[f"rider{j}"].append(classNames[cls])

```

```

        if classNames[cis] == "number plate":
            npx, npy, npw, npy, npconf = x1, y1, w, h,
            conf

            crop = img[npy:npy + h, npx:npx + w]

            # Save number plate if rider is without helmet
            if "without helmet" in li[f"rider{j}"] and
"number plate" in li[f"rider{j}"]:
                npx, npy, npw, npy, npconf = x1, y1, w, h,
                crop = img[npy:npy + h, npx:npx + w]

                # Save the number plate image
                file_name =
f"number_plate_{datetime.now().strftime('%Y%m%d_%H%M%S')}.jpg"
                save_number_plate_image(crop, file_name)
                print(f"Number plate image saved:
{file_name}")

            if li:
                for key, value in li.items():
                    if key == f"rider{j}":
                        if len(list(set(li[f"rider{j}"]))) ==
3:
                            try:
                                # crop = cv2.cvtColor(crop,
cv2.COLOR_RGB2GRAY) # for easy ocr

                                vechicle_number, conf =
predict_number_plate(crop, ocr)

                                if vechicle_number and conf:
                                    cvzone.putTextRect(img,
f"{vechicle_number} {round(conf * 100, 2)}%",
(x1, y1
- 50), scale=1.5, offset=10,
thickness=2, colorT=(39, 40, 41),
colorR=(105, 255, 255))

                            except Exception as e:
                                print(e)

            # Write processed frame to the output video
            output.write(img)
            cv2.imshow('Video', img)
            li = []
            rider_box = []

            # Exit loop if 'q' is pressed
            if cv2.waitKey(1) & 0xFF == ord('q'):

```

```

        break

    # Release video resources
    cap.release()
    output.release()
    cv2.destroyAllWindows()

    return output_path

if __name__ == '__main__':
    app.run(debug=True)

```

training.py

```

from ultralytics import YOLO

# yolo model creation
model = YOLO("yolo-weights/yolov8l.pt")
model.train(data="coco128.yaml", imgsz=320, batch=4, epochs=20, workers=0)

```

image_to_text.py

```

import easyocr
import csv
import os
import numpy as np
import pandas as pd
import cv2
import re

def predict_number_plate(img, ocr):
    result = ocr.ocr(img, cls=True)
    result = result[0]
    texts = [line[1][0] for line in result]
    scores = [line[1][1] for line in result]
    if (scores[0]*100) >= 90:
        return re.sub(r'^a-zA-Z0-9', '', texts[0]), scores[0]
    else:
        return None, None

```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link
href="https://fonts.googleapis.com/css?family=Nunito+Sans:400,400i,700,900&dis
play=swap" rel="stylesheet">
  <title>Video Upload</title>
  <style>

    #completed-message {
      display: none;
      margin-top: 10px;
    }

    body {
      font-family: sans-serif;
      background-color: #eeeeee;
    }

    .file-upload {
      background-color: #ffffff;
      width: 600px;
      margin: 0 auto;
      padding: 20px;
    }

    .file-upload-btn {
      width: 100%;
      margin: 0;
      color: #fff;
      background: #1FB264;
      border: none;
      padding: 10px;
      border-radius: 4px;
      border-bottom: 4px solid #15824B;
      transition: all .2s ease;
      outline: none;
      text-transform: uppercase;
      font-weight: 700;
    }

    .file-upload-btn:hover {
      background: #1AA059;
      color: #ffffff;
    }
  </style>
</head>
<body>
  <div class="file-upload">
    <div class="file-upload-btn">
      <button type="button">Upload Video</button>
    </div>
  </div>
  <div id="completed-message">
    <div class="message">
      <div class="icon">
        <img alt="Success icon" />
      </div>
      <div class="text">
        <div class="text">Video uploaded successfully</div>
        <div class="text">You can now watch your video</div>
      </div>
    </div>
  </div>
</body>
</html>
```

```

transition: all .2s ease;
cursor: pointer;
}

.file-upload-btn:active {
border: 0;
transition: all .2s ease;
}

.file-upload-content {
display: none;
text-align: center;
}

.file-upload-input {
position: absolute;
margin: 0;
padding: 0;
width: 100%;
height: 100%;
outline: none;
opacity: 0;
cursor: pointer;
}

.image-upload-wrap {
margin-top: 20px;
border: 4px dashed #1FB264;
position: relative;
}

.image-dropping,
.image-upload-wrap:hover {
background-color: #1FB264;
border: 4px dashed #ffffff;
}

.image-title-wrap {
padding: 0 15px 15px 15px;
color: #222;
}

.drag-text {
text-align: center;
}

.drag-text h3 {
font-weight: 100;
text-transform: uppercase;
}

```

```

color: #15824B;
padding: 60px 0;
}

.file-upload-image {
max-height: 200px;
max-width: 200px;
margin: auto;
padding: 20px;
}

.remove-image {
width: 200px;
margin: 0;
color: #fff;
background: #cd4535;
border: none;
padding: 10px;
border-radius: 4px;
border-bottom: 4px solid #b02818;
transition: all .2s ease;
outline: none;
text-transform: uppercase;
font-weight: 700;
}

.remove-image:hover {
background: #c13b2a;
color: #ffffff;
transition: all .2s ease;
cursor: pointer;
}

.remove-image:active {
border: 0;
transition: all .2s ease;
}

.loader {
border: 16px solid #f3f3f3; /* Light gray */
border-top: 16px solid #4caf50; /* Green */
border-radius: 50%;
width: 120px;
height: 120px;
animation: spin 2s linear infinite;
margin: auto;
}

h1 {
color: #88B04B;

```

```

        font-family: "Nunito Sans", "Helvetica Neue", sans-serif;
        font-weight: 900;
        font-size: 40px;
        margin-bottom: 10px;
    }

    p {
        color: #404F5E;
        font-family: "Nunito Sans", "Helvetica Neue", sans-serif;
        font-size: 20px;
        margin: 0;
    }

    i {
        color: #9ABC66;
        font-size: 100px;
        line-height: 200px;
        margin-left: -15px;
    }

    .card {
        background: white;
        padding: 60px;
        border-radius: 4px;
        box-shadow: 0 2px 3px #C8D0D8;
        display: inline-block;
        margin: 0 auto;
        margin-bottom: 10px;
    }

    @keyframes spin {
        0% { transform: rotate(0deg); }
        100% { transform: rotate(360deg); }
    }
</style>
</head>
<body>
    <h2 style="text-align:center;">Upload and Process Video</h2>

    <div id="loader-container" style="display: none; text-align: center;
margin-top: 20px;">
        <div class="loader"></div>
        <p>Processing, please wait...</p>
    </div>

    <div id="completed-message" style="text-align: center;">

```

```

        <div class="card">
            <div style="border-radius:200px; height:200px; width:200px;
background: #F8FAF5; margin:0 auto;">
                <i class="checkmark">✓</i>
            </div>
            <h1>Process Completed!</h1>
            <p>Please check processed video folder;<br/> to get the output
video!</p>
        </div>
    </div>

    <script class="jsbin"
src="https://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script>
<div class="file-upload">

    <div class="image-upload-wrap">
        <form id="video-form" enctype="multipart/form-data">
            <input class="file-upload-input" type='file' id="video" name="video"
onchange="readURL(this);" accept="video/mp4,video/x-m4v,video/*" required/>
            <div class="drag-text">
                <h3>Drag and drop a file or select Add Video</h3>
            </div>
        </form>
    </div>

    <div class="file-upload-content">
        <video class="file-upload-video" controls style="display: none; width:
50%;">
            <source src="#" type="video/mp4" />
            Your browser does not support the video tag.
        </video>
        <div class="image-title-wrap">
            <button type="button" onclick="removeUpload()"
class="remove-image">Remove <span class="image-title">Uploaded
Video</span></button>
        </div>
    </div>

    <button class="file-upload-btn" style="margin-top: 15px;" type="button"
onclick="startProcessing()">Start Processing</button>

</div>

    <script>
function startProcessing() {

    let processButton = document.querySelector('.file-upload-btn');
    processButton.disabled = true;

```

```

    processButton.style.backgroundColor = "#ccc"; // Optional: Change color to
    indicate it's disabled
    processButton.style.cursor = "not-allowed";

    let formData = new FormData(document.getElementById("video-form"));

    let xhr = new XMLHttpRequest();
    xhr.open("POST", "/upload", true);

    // Show loader when the upload starts
    document.getElementById("loader-container").style.display = "block";
    document.getElementById("completed-message").style.display = "none";

    // Hide loader and show completed message when the upload is complete
    xhr.onload = function() {
        if (xhr.status === 200) {
            document.getElementById("loader-container").style.display = "none";
            document.getElementById("completed-message").style.display =
"block";
        } else {
            alert("An error occurred during processing.");
            document.getElementById("loader-container").style.display = "none";
        }

        // Re-enable the button after processing is complete
        processButton.disabled = false;
        processButton.style.backgroundColor = "#1FB264"; // Restore original
color
        processButton.style.cursor = "pointer";
    };

    // Send the form data to the server
    xhr.send(formData);
}

</script>

<script>
function readURL(input) {
    if (input.files && input.files[0]) {

        var reader = new FileReader();

        reader.onload = function(e) {
            $('#image-upload-wrap').hide();

            $('#file-upload-video').attr('src', e.target.result);
            $('#file-upload-video').show();

```

```

        $('#file-upload-content').show();

        $('#image-title').html(input.files[0].name);
    };

    reader.readAsDataURL(input.files[0]);

} else {
    removeUpload();
}
}

function removeUpload() {
    $('#file-upload-input').replaceWith($('#file-upload-input').clone());
    $('#file-upload-content').hide();
    $('#file-upload-video').hide();
    $('#image-upload-wrap').show();
}

$('#image-upload-wrap').bind('dragover', function () {
    $('#image-upload-wrap').addClass('image-dropping');
});

$('#image-upload-wrap').bind('dragleave', function () {
    $('#image-upload-wrap').removeClass('image-dropping');
});
</script>
</body>
</html>

```