Smart Helmet Compliance System

(Helmet Detection and Reward Allocation System)

A Project Report

Submitted by

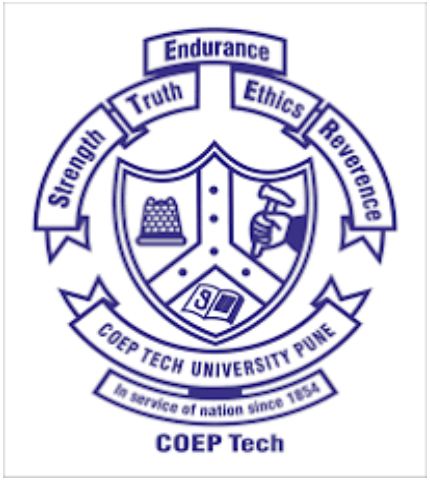
Prathamesh Agawane 142203001
Prasad Patil 142203015

in partial fulfilment for the course

of

Software Engineering – Stage -2

Under the guidance of
Prof.Dr. Jibi Abraham



DEPARTMENT OF COMPUTER ENGINEERING AND INFORMATION TECHNOLOGY
COLLEGE OF ENGINEERING, PUNE-5

April, 2024

ABSTRACT

Motorcycle accidents have become a significant concern due to the prevalent disregard for helmet usage among riders. To address this issue, governments have implemented strict regulations mandating helmet usage. However, traditional surveillance methods for enforcement are labor-intensive and prone to human error. In response, we propose an automated system for real-time detection of motorcyclists without helmets using video surveillance.

Our approach utilizes state-of-the-art object detection techniques, specifically YOLOv5, to detect motorcycles in surveillance footage. Subsequently, the system distinguishes between helmeted and non-helmeted riders. Upon identifying a compliant as well as non-compliant rider, the system employs Optical Character Recognition (OCR) to capture the vehicle's number plate, providing a means for enforcement.

Additionally, our system incorporates a rewards system to incentivize helmet usage. Upon detecting a compliant rider, reward points are assigned, and the rider is notified via email. This serves to encourage safe behavior and foster a culture of helmet usage among motorcyclists.

By automating the detection process and implementing incentives for compliance, our system offers an efficient and effective solution to mitigate motorcycle accidents and promote road safety. Moreover, it reinforces the notion of accountability among riders, ultimately contributing to the overall well-being of society.

List of Figures

Table of Contents

CHAPTER 1

INTRODUCTION

1.1 WHAT IS OBJECT DETECTION

Object Detection is a common Computer Vision problem which deals with identifying and locating object of certain classes in the image. Interpreting the object localisation can be done in various ways, including creating a bounding box around the object or marking every pixel in the image which contains the object (called segmentation).

Object detection was studied even before the breakout popularity of CNNs in Computer Vision. While CNNs are capable of automatically extracting more complex and better features, taking a glance at the conventional methods can at worst be a small detour and at best an inspiration. Object detection before Deep Learning was a several step process, starting with edge detection and feature extraction using techniques like SIFT, HOG etc. These image were then compared with existing object templates, usually at multi scale levels, to detect and localize objects present in the image.

Image classification involves assigning a class label to an image, whereas object localization involves drawing a bounding box around one or more objects in an image. Object detection is more challenging and combines these two tasks and draws a bounding box around each object of interest in the image and assigns them a class label. Together, all of these problems are referred to as object recognition.

Fig 1.1 object detection

1.2 TWO-STEP OBJECT DETECTION

Two-Step Object Detection involves algorithms that first identify bounding boxes which may potentially contain objects and then classify each bounding separately.

The first step requires a *Region Proposal Network,* providing a number of regions which are then passed to common DL based classification architectures. From the hierarchical grouping algorithm in RCNNs (which are extremely slow) to using CNNs and ROI pooling in Fast RCNNs and anchors in Faster RCNNs (thus speeding up the pipeline and training end-to- end), a lot of different methods and variations have been provided to these region proposal networks (RPNs).

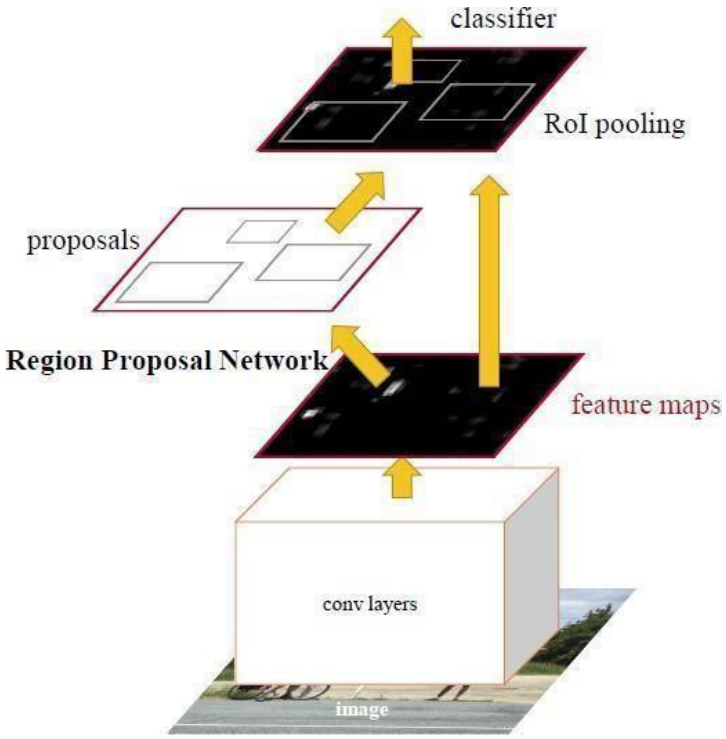


Fig 1.2.1 two step detection layout .

These algorithms are known to perform better than their one-step object detection counterparts, but are slower in comparison. With various improvements suggested over the years, the current bottleneck in the latency of Two-Step Object Detection networks is the RPN step.

1.3 ONE-STEP OBJECT DETECTION

With the need of real time object detection, many one-step object detection architectures have been proposed, like YOLO, YOLOv2, YOLOv3, SSD, RetinaNet etc. which try to combine the detection and classification step.

One of the major accomplishments of these algorithms have been introducing the idea of 'regressing' the bounding box predictions. When every bounding box is represented easily with a few values (for example, xmin, xmax, ymin and ymax), it becomes easier to combine the detection and classification step and dramatically speed up the pipeline.

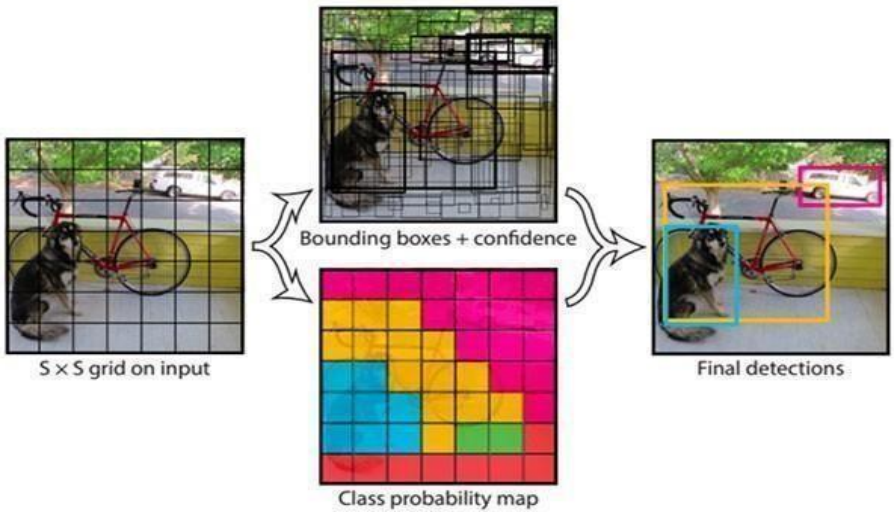


Fig 1.3. one step object detection layout.

1.4 Problem Statement:

Motorcycle accidents are a critical concern exacerbated by the prevalent disregard for helmet usage among riders. Conventional enforcement methods are labor-intensive and prone to errors. Hence, there's an urgent need for an automated system utilizing advanced object detection techniques to identify helmetless riders in real-time from surveillance footage. This system not only enforces helmet compulsion regulations but also incentivizes compliance by rewarding users with points for adhering to safety protocols. By promoting helmet compulsion and motivating users through reward points, the system aims to reduce accidents, enhance road safety, and instill a culture of responsible motorcycle riding.

CHAPTER 2

SOFTWARE REQUIREMENTS SPECIFICATION

2.1 Product Perspective

The product perspective encompasses the integration of cutting-edge technology, such as computer vision algorithms and sophisticated database management systems, to actively monitor and incentivize helmet compliance among riders. Moreover, the adaptable nature of the system allows for continuous updates and enhancements, ensuring its alignment with evolving road safety standards and leveraging technological advancements to maximize its impact on promoting safe riding practices

2.1.1 User Classes and Characteristics

”Smart Helmet Compliance System ” has basically 4 types of users.

- **Riders: Characteristics:** Two-wheeler riders, divided into helmet-wearing and nonhelmet-wearing. Actions: Ride, wear helmets, accrue points, receive rewards.
- **System Administrators: Characteristics:** Manage system operations and settings. Actions: Configure parameters, monitor performance, handle accounts, address technical issues.
- **Database Administrators:Characteristics:** Database management experts ensuring data integrity and security. Actions: Design schema, maintain data consistency, optimize performance.
- **Law Enforcement Authorities:Characteristics:** Traffic officers enforcing road safety regulations. Actions: Monitor traffic, use system data for enforcement, issue penalties

2.2 External Interface Requirements

External Interface Requirements in a Software Requirements Specification (SRS) outline how the software interacts with its environment:

1. **User Interfaces:** Detail how users interact with the system.
2. **Hardware Interfaces:** Define connections with physical devices.

3. **Software Interfaces:** Specify interactions with other software components.
4. **Communication Interfaces:** Describe communication protocols.
5. **Data Interfaces:** Specify data formats and structures.
6. **Security Interfaces:** Detail security features and mechanisms.
7. **External Service Interfaces:** Define interactions with external services or APIs.

These requirements ensure smooth interaction between the software and its environment.

2.2.1. Hardware and Communication Interfaces

Camera Systems: Integration with existing closed-circuit television (CCTV) cameras or dedicated camera systems to capture real-time footage of two-wheeler riders on the road.

Data Collection Hardware: Utilization of hardware devices for collecting video footage from various sources, including CCTV cameras and dedicated sensors

2.2.2 System Features

Helmet Compliance Tracking: Description: Real-time monitoring of two-wheeler riders to detect and track instances of helmet usage or non-compliance. Benefits: Ensures adherence to safety regulations and forms the basis for incentivized reward programs.

Incentivized Rewards System: Description: Introduction of a dynamic reward mechanism to motivate and recognize users consistently wearing helmets. Benefits: Encourages a positive culture of responsible riding, fostering long-term compliance and safety awareness.

Highly Reliable Back-end Infrastructure: Description: Establishment of a robust back-end infrastructure with redundancy and failover mechanisms for high reliability. Benefits: Minimizes downtime, ensures system availability, and provides a stable foundation for overall system performance.

Secure Data Storage: Description: Utilization of robust encryption techniques to secure user data in storage, including personal information, compliance history,

and reward details. Benefits: Safeguards sensitive user information, maintaining the integrity and confidentiality of stored data.

Hardware and Communication Interfaces: Description: Seamless integration with hardware devices such as CCTV cameras, sensors, and communication interfaces like E-mail API

2.2.3 Data Acquisition

The data for this research is obtained from two different sources which are the Kaggle dataset and Google Images with creative commons license4 . Each image is manually reviewed and carefully selected as quality data for this application is not readily available. The data consists of 120 images of real road traffic scenes with both the classes (Helmeted and non-helmeted motorcyclists) in .jpg format. The collection contains images of different shapes and sizes and angles but with the same format. The reason for collecting images from two different sources is that the Deep learning model performs better when trained with images containing different scenes.

2.2.4 Data Collection and Aggregation

The data collection and aggregation component of the Smart Helmet Compliance System with Incentivized Rewards focuses on acquiring, processing, and consolidating video footage using the YOLOv5 algorithm for helmet compliance tracking, user profiling, and incentive management.

Detailed Functionality:
Video Footage Collection:
Description: Capture real-time video footage from various sources, including closed-circuit television (CCTV) cameras. Functionality: Ensures a continuous stream of visual data for the YOLOv5 algorithm to analyze and detect instances of helmet compliance or non-compliance.

YOLOv5 Algorithm for Helmet Detection:
Description: Utilize the YOLOv5 algorithm for real-time object detection, specifically focusing on identifying helmets on two-wheeler riders. Functionality: Employs advanced computer vision techniques to accurately detect the presence or absence of helmets in the video footage.

Database Querying for Reporting:
Description: Implement interfaces for querying the database to generate reports on user compliance, incentive distribution, and system performance. Functionality: Enables administrators and stakeholders to extract valuable insights through customized reports, aiding decisionmaking and continuous improvement.

2.2.5 Hardware Requirements

Computer for YOLOv5 Algorithm Processing
Multi-core processor
GPU (NVIDIA recommended for CUDA support)
Minimum 16GB RAM
Adequate storage for video footage

Closed-Circuit Television (CCTV) Cameras
High-resolution cameras
Good low-light performance
Data Collection Hardware
Reliable devices for real-time video streaming
Data storage

2.2.6 Software requirements

Operating System
Computer Vision Software
YOLOv5 implementation
Python environment
Dependencies for computer vision tasks

2.3 Non Functional Requirements:

Non-functional requirements (NFRs) specify how a system should perform rather than what it should do:

1. **Performance:** How fast and efficiently the system operates.
2. **Reliability:** Consistency and fault tolerance of the system.
3. **Security:** Protection against unauthorized access and data breaches.

4. **Usability:** Ease of use and user satisfaction.
5. **Scalability:** Ability to handle increasing workload.
6. **Maintainability:** Ease of system maintenance and updates.
7. **Compatibility:** Interaction with other systems and platforms.
8. **Interoperability:** Ability to exchange data with external systems.
9. **Regulatory Compliance:** Conformance to legal and industry standards.

NFRs ensure that the system meets quality standards and constraints necessary for its success.

2.3.1 Performance requirements :

The system should process video footage in real-time, aiming for a minimum of 30 frames per second. Response time for user interactions within the mobile application should be less than 1 second.

2.3.2 Scalability:

The system should handle an increasing number of concurrent users and data points without a significant degradation in performance. Scalability should be achieved through efficient resource utilization and load balancing mechanisms.

CHAPTER 3

METHODS AND TECHNIQUES USED

3.1 WHAT IS YOLOV5

YOLOv5 is a state-of-the-art object detection algorithm that stands for "You Only Look Once." Developed by Ultralytics, it represents the latest iteration in the YOLO (You Only Look Once) series, known for its speed and accuracy in real-time object detection tasks. YOLOv5 builds upon its predecessors by introducing advancements in architecture design and training techniques, resulting in improved performance across various computer vision applications.

One of the key features of YOLOv5 is its streamlined architecture, which enables faster inference speeds without compromising accuracy. By employing a single neural network to directly predict bounding boxes and class probabilities, YOLOv5 achieves real-time object detection on a wide range of platforms, from edge devices to high-performance servers. Furthermore, YOLOv5 is designed to be highly customizable and easy to use, offering pre-trained models and tools for fine-tuning on custom datasets, making it accessible for both researchers and practitioners in the field of computer vision

3.2 HOW YOLO IS USEFUL

The R-CNN family of techniques we saw in Part 1 primarily use regions to localize the objects within the image. The network does not look at the entire image, only at the parts of the images which have a higher chance of containing an object.

The YOLO framework (You Only Look Once) on the other hand, deals with object detection in a different way. It takes the entire image in a single instance and predicts the bounding box coordinates and class probabilities for these boxes.

The biggest advantage of using YOLO is its superb speed – it's incredibly fast and can process 45 frames per second. YOLO also understands generalized object representation.

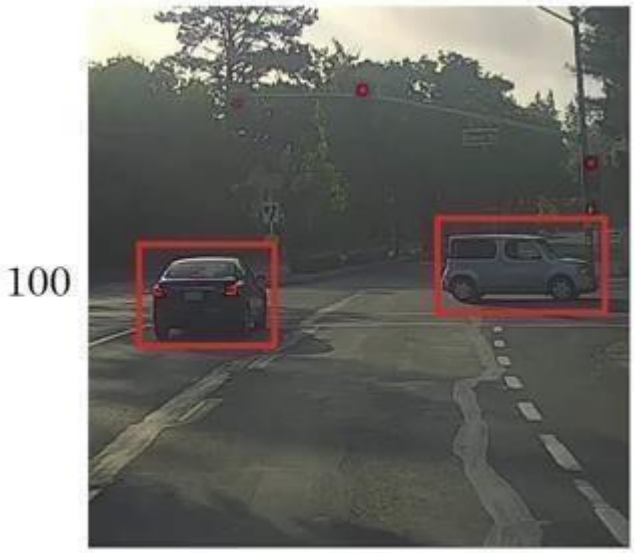
This is one of the best algorithms for object detection and has shown a comparatively similar performance to the R-CNN algorithms. In the upcoming sections, we will learn about different techniques used in YOLO algorithm. The

following explanations are inspired by Andrew NG's course on Object Detection which helps in understanding the working of YOLO.

3.3 HOW DOES THE YOLO FRAMEWORK FUNCTION?

Now that we have grasp on why YOLO is such a useful framework, let's jump into how it actually works. In this section, I have mentioned the steps followed by YOLO for detecting objects in a given image.

Fig 3.3.1 yolo step1



- YOLO first takes an input image:

The framework then divides the input image into grids (say a 3 X 3 grid).

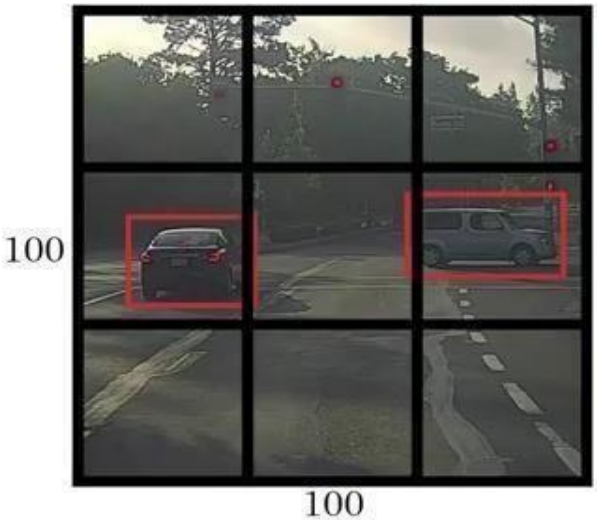


Fig 3.3.2 yolo step2

- Image classification and localization are applied on each grid. YOLO then predicts the bounding boxes and their corresponding class probabilities for objects (if any are found, of course).

Pretty straightforward,Let's break down each step to get a more granular understanding.

We need to pass the labelled data to the model in order to train it. Suppose we have divided the image into a grid of size 3 X 3 and there are a total of 3 classes which we want the objects to be classified into. Let's say the classes are Pedestrian, Car, and Motorcycle respectively. So, for each grid cell, the label y will be an eight dimensional vector
Here,

- p_c defines whether an object is present in the grid or not (it is the probability)
- b_x, b_y, b_h, b_w specify the bounding box if there is an object
- c_1, c_2, c_3 represent the classes. So, if the object is a car, c_2 will be 1 and c_1 & c_3 will be 0, and so on
  Since there is no object in this grid, pc will be zero and the y label for this grid will be

Here, '?' means that it doesn't matter what bx, by, bh, bw, c1, c2, and c3 contain as there is no object in the grid. Let's take another grid in which we have a car (c2 = 1)

Before we write the y label for this grid, it's important to first understand how YOLO decides whether there actually is an object in the grid. In the above image, there are two objects (two cars), so YOLO will take the mid-point of these two objects and these objects will be assigned to the grid which contains the mid-point of these objects. The y label for the centre left grid with the car will be:

Since there is an object in this grid, p_c will be equal to 1. b_x, b_y, b_h, b_w will be calculated relative to the particular grid cell we are dealing with. Since car is the second class, $c_2 = 1$ and c_1 and $c_3 = 0$. So, for each of the 9 grids, we will have an eight dimensional output vector. This output will have a shape of 3 X 3 X 8.

So now we have an input image and it's corresponding target vector. Using the above example (input image – 100 X 100 X 3, output – 3 X 3 X 8), our model will be trained as follows

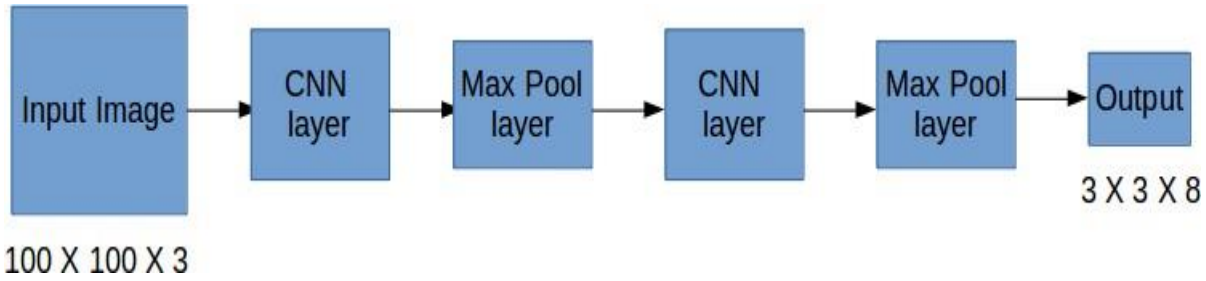


Fig 3.3.4 yolo process

We will run both forward and backward propagation to train our model. During the testing phase, we pass an image to the model and run forward propagation until we get an output y. In order to keep things simple, I have explained this using a 3 X 3 grid here, but generally in real-world scenarios we take larger grids (perhaps 19 X 19).

Even if an object spans out to more than one grid, it will only be assigned to a single grid in which its mid-point is located. We can reduce the chances of multiple objects appearing in the same grid cell by increasing the more number of grids (19 X 19, for example).

3.4 How to Encode Bounding Boxes?

As I mentioned earlier, b_x, b_y, b_h, and b_w are calculated relative to the grid cell we are dealing with. Let's understand this concept with an example. Consider the center-right grid which contains a car:

So, b_x, b_y, b_h, and b_w will be calculated relative to this grid only. The y label for this grid will be

$p_c = 1$ since there is an object in this grid and since it is a car, $c_2 = 1$. Now, let's see how to decide b_x, b_y, b_h, and b_w. In YOLO, the coordinates assigned to all the grids are b_x, b_y are the x and y coordinates of the midpoint of the object with respect to this grid.
In this case, it will be (around) $b_x = 0.4$ and $b_y = 0.3$ b_h is the ratio of the height of the bounding box (red box in the above example) to the height of the

corresponding grid cell, which in our case is around 0.9. So, $b_h = 0.9$. b_w is the ratio of the width of the bounding box to the width of the grid cell. So, $b_w = 0.5$ (approximately). The y label for this grid will be

Notice here that b_x and b_y will always range between 0 and 1 as the midpoint will always lie within the grid. Whereas b_h and b_w can be more than 1 in case the dimensions of the bounding box are more than the dimension of the grid.

Here's some food for thought – how can we decide whether the predicted bounding box is giving us a good outcome (or a bad one)? This is where Intersection over Union comes into the picture. It calculates the intersection over union of the actual bounding box and the predicted bonding box. Consider the actual and predicted bounding boxes for a car as shown below:

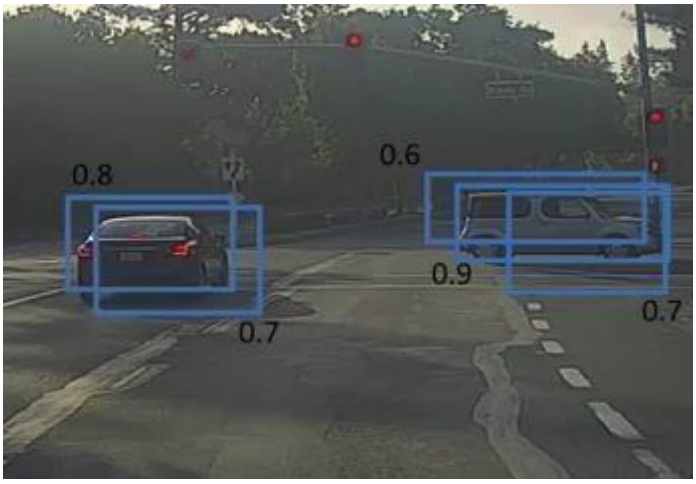
Here, the red box is the actual bounding box and the blue box is the predicted one. How can we decide whether it is a good prediction or not? IoU, or Intersection over Union, will calculate the area of the intersection over union of these two boxes. That area will be:

IoU = Area of the intersection / Area of the union, i.e.

IoU = Area of yellow box / Area of green box

If IoU is greater than 0.5, we can say that the prediction is good enough. 0.5 is an arbitrary threshold we have taken here, but it can be changed according to your specific problem.
Intuitively, the more you increase the threshold, the better the predictions become. There is one more technique that can improve the output of YOLO significantly

– Non-Max Suppression.
One of the most common problems with object detection algorithms is that rather than detecting an object just once, they might detect it multiple times. Consider the below image:



3.4.1 yolo final detection

Here, the cars are identified more than once. The Non-Max Suppression technique cleans up this up so that we get only a single detection per object. Let's see how this approach works.

1. It first looks at the probabilities associated with each detection and takes the largest one. In the above image, 0.9 is the highest probability, so the box with 0.9 probability will be selected first:

2. Now, it looks at all the other boxes in the image. The boxes which have high IoU with the current box are suppressed. So, the boxes with 0.6 and 0.7 probabilities will be suppressed in our example:

3. After the boxes have been suppressed, it selects the next box from all the boxes with the highest probability, which is 0.8 in our case:

4. gain it will look at the IoU of this box with the remaining boxes and compress the boxes with a high IoU:

5. We repeat these steps until all the boxes have either been selected or compressed and we get the final bounding boxes:

This is what Non-Max Suppression is all about. We are taking the boxes with maximum probability and suppressing the close-by boxes with non-max probabilities. Let's quickly summarize the points which we've seen in this section about the Non-Max suppression algorithm:

1. Discard all the boxes having probabilities less than or equal to a pre-defined threshold (say, 0.5)
2. For the remaining boxes:
3. Repeat step 2 until all the boxes are either taken as the output prediction or discarded

3.5 What is Optical Character Recognition (OCR)

Optical Character Recognition (OCR) technology has revolutionized the digitization and processing of textual information from images or scanned documents. OCR systems play a crucial role in various applications, including document digitization, automated data entry, and text extraction from images.

Key aspects of OCR technology include:

1. **Pre-processing Techniques**: OCR systems often employ pre-processing techniques to enhance image quality and improve OCR accuracy. These techniques may include noise reduction, binarization, skew correction, and image enhancement to optimize the input image for text extraction.
2. **Feature Extraction**: OCR algorithms extract meaningful features from the input image to identify and recognize characters accurately. These features may include stroke width, edge detection, and character segmentation to isolate individual characters or text regions within the image.
3. **Character Recognition Algorithms**: OCR systems utilize various character recognition algorithms to interpret extracted features and match them to predefined character templates or models. These algorithms may include template matching, neural networks, Hidden Markov Models (HMM), Support Vector Machines (SVM), and deep learning techniques like Convolutional Neural Networks (CNN).
4. **Language Support and Character Sets**: OCR systems support a wide range of languages and character sets to accommodate diverse textual content. Depending on the application, OCR systems may need to recognize Latin characters, numerals, symbols, or multi-language text.
5. **Post-processing and Error Correction**: After character recognition, OCR systems often employ post-processing techniques to refine the extracted text and correct any recognition errors. These techniques may include dictionary-

based correction, spell checking, context analysis, and language modeling to improve the accuracy of the final output.

6. **Integration and Deployment**: OCR technology can be integrated into various software applications, devices, and workflows to automate document processing tasks. OCR APIs and SDKs facilitate easy integration into custom applications, enabling developers to leverage OCR capabilities seamlessly.

Overall, OCR technology is versatility and widespread adoption make it an indispensable tool for digitizing and extracting textual information from a variety of sources, contributing to increased efficiency and productivity across industries.

3.6 Why is Optical Character Recognition (OCR) useful:

OCR is useful for several reasons:

1. **Digitization of Documents**: OCR allows for the conversion of printed or handwritten text from physical documents into digital format, facilitating easier storage, sharing, and editing of documents.
2. **Text Extraction**: OCR enables the extraction of textual information from images, scanned documents, or screenshots, making it possible to access and manipulate the content of these documents electronically.
3. **Searchability**: Once text is extracted using OCR, it becomes searchable, allowing users to quickly locate specific information within documents or images without having to manually read through them.
4. **Data Entry Automation**: OCR can automate data entry tasks by extracting information from documents and inputting it into databases, forms, or other software applications, saving time and reducing errors associated with manual data entry.
5. **Accessibility**: OCR technology can convert printed text into digital text, making content accessible to individuals with visual impairments through text-to-speech or screen reader software.
6. **Document Analysis**: OCR enables the analysis of documents for various purposes, such as extracting key insights, detecting patterns, or categorizing content based on textual information.

CHAPTER 4

DESIGN

4.1 Dats Flow Diagram:

- **DFD (Level-0):** At Level 0, the Data Flow Diagram (DFD) provides a high-level overview of the system, illustrating interactions between the system and external entities. It depicts the system's inputs, outputs, and interfaces without detailing internal processes, serving as a foundation for understanding the system's context and scope

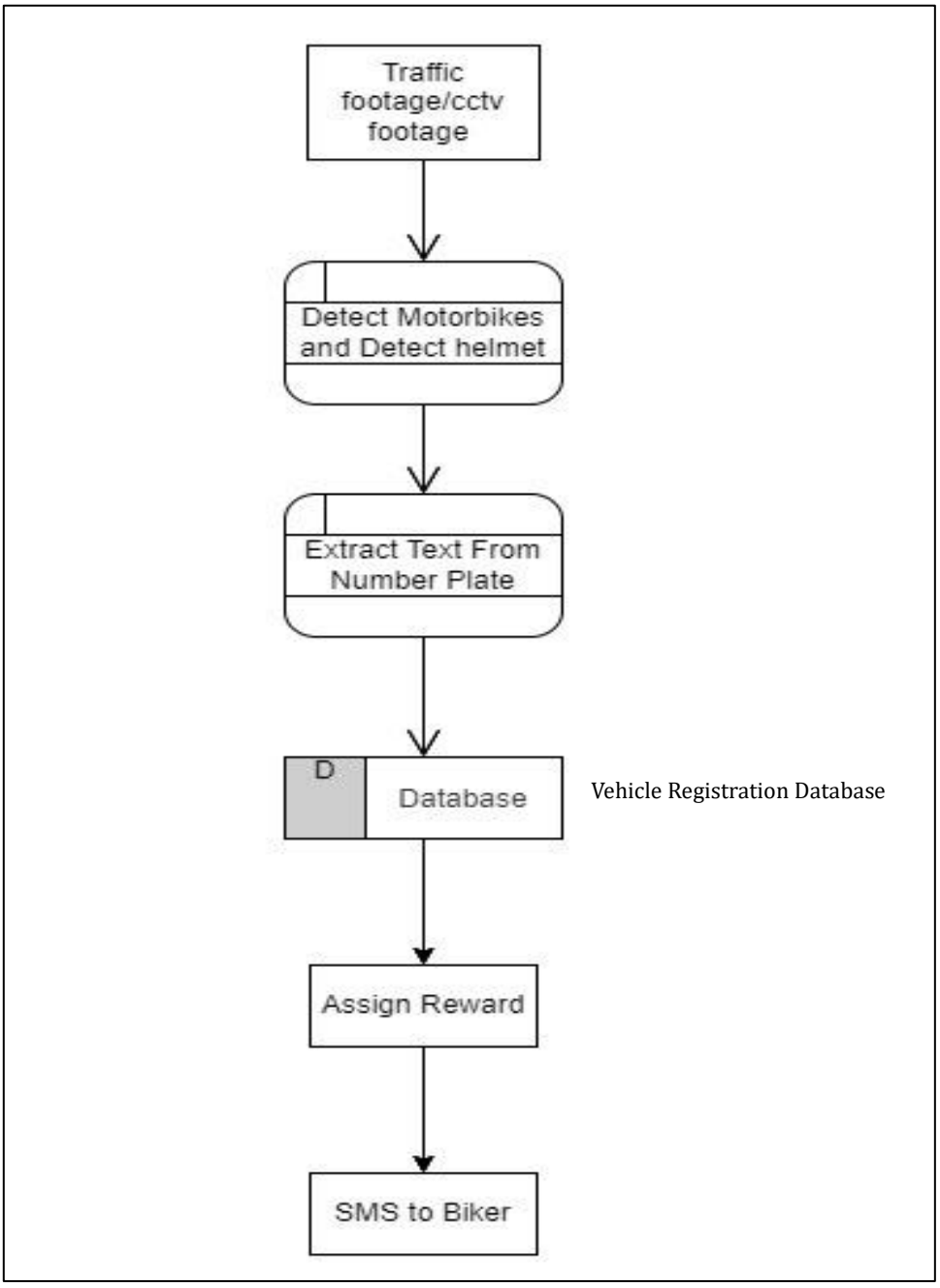


Fig 4.1.1 : DFD Level -0

- **DFD (Level-1):**

In Level 1 DFD, the system is decomposed into subprocesses or modules, showing how data flows between components. This diagram offers a more detailed view of the system's internal workings, guiding the design and implementation of its components.

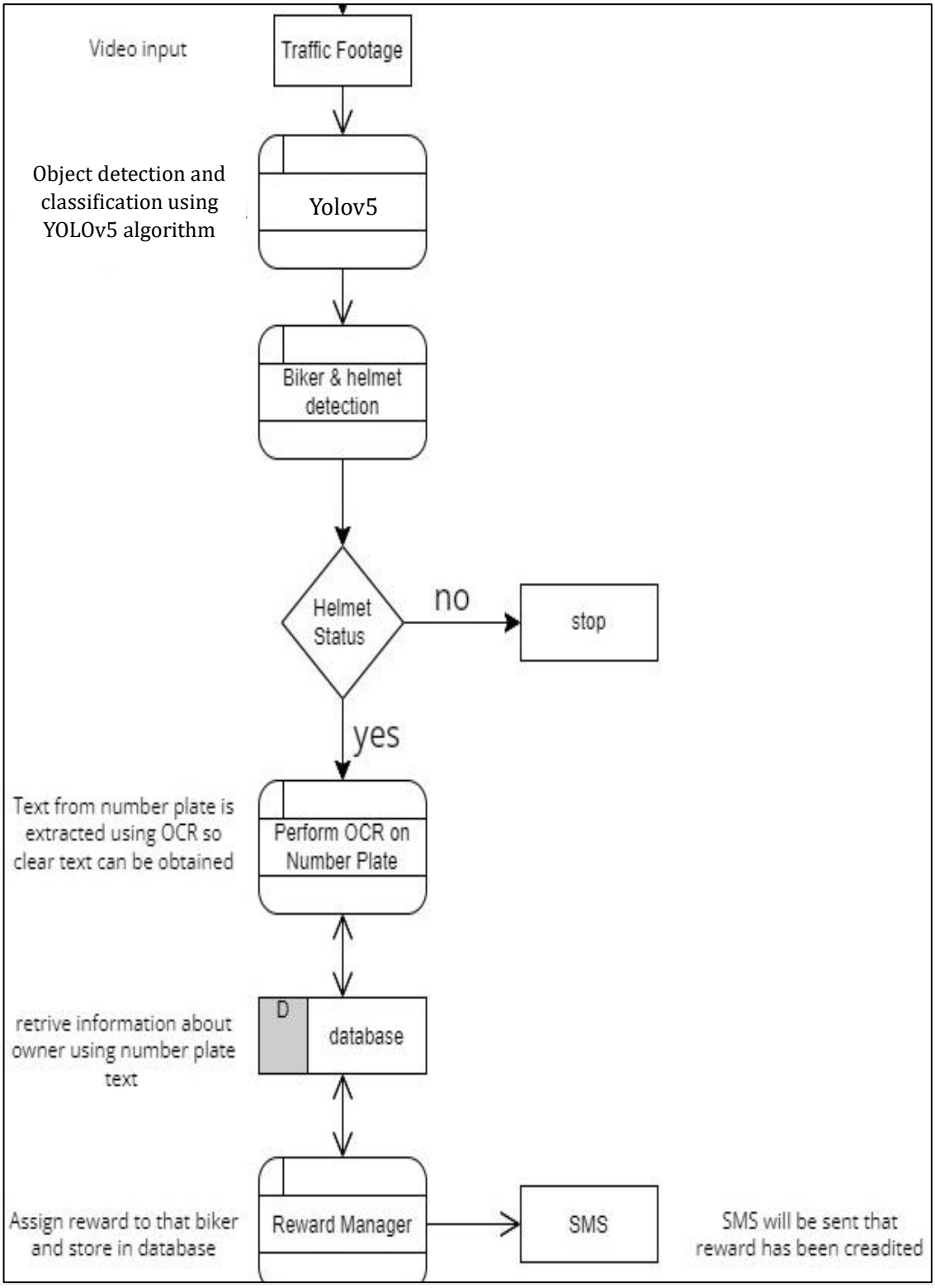


Fig 4.1.2 : DFD Level -1

- **DFD LEVEL-2:**

At Level 2 DFD, the system's subprocesses depicted in Level 1 are further decomposed into detailed processes, illustrating the finer-grained flow of data within each module. Level 2 DFD facilitates a thorough understanding of the system's functionalities

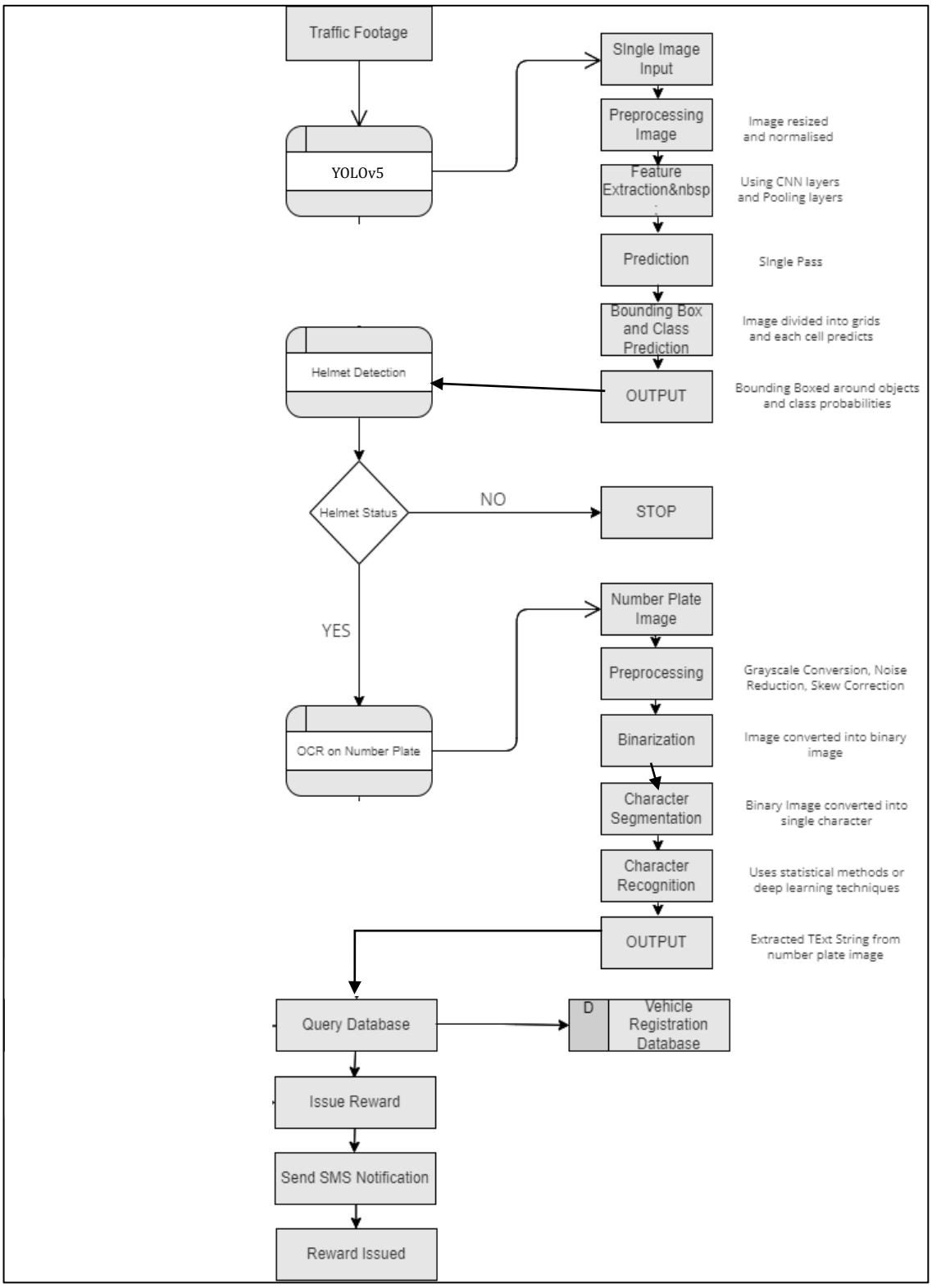


Fig 4.1.3 : DFD Level -2

4.2 Use Case Diagram :

A use case diagram visually represents the interactions between users and a system. It identifies actors, such as users or external systems, and use cases, which are actions performed by users. Relationships show how actors engage with use cases. Use case diagrams aid in understanding system requirements and facilitating communication among stakeholders in software development.

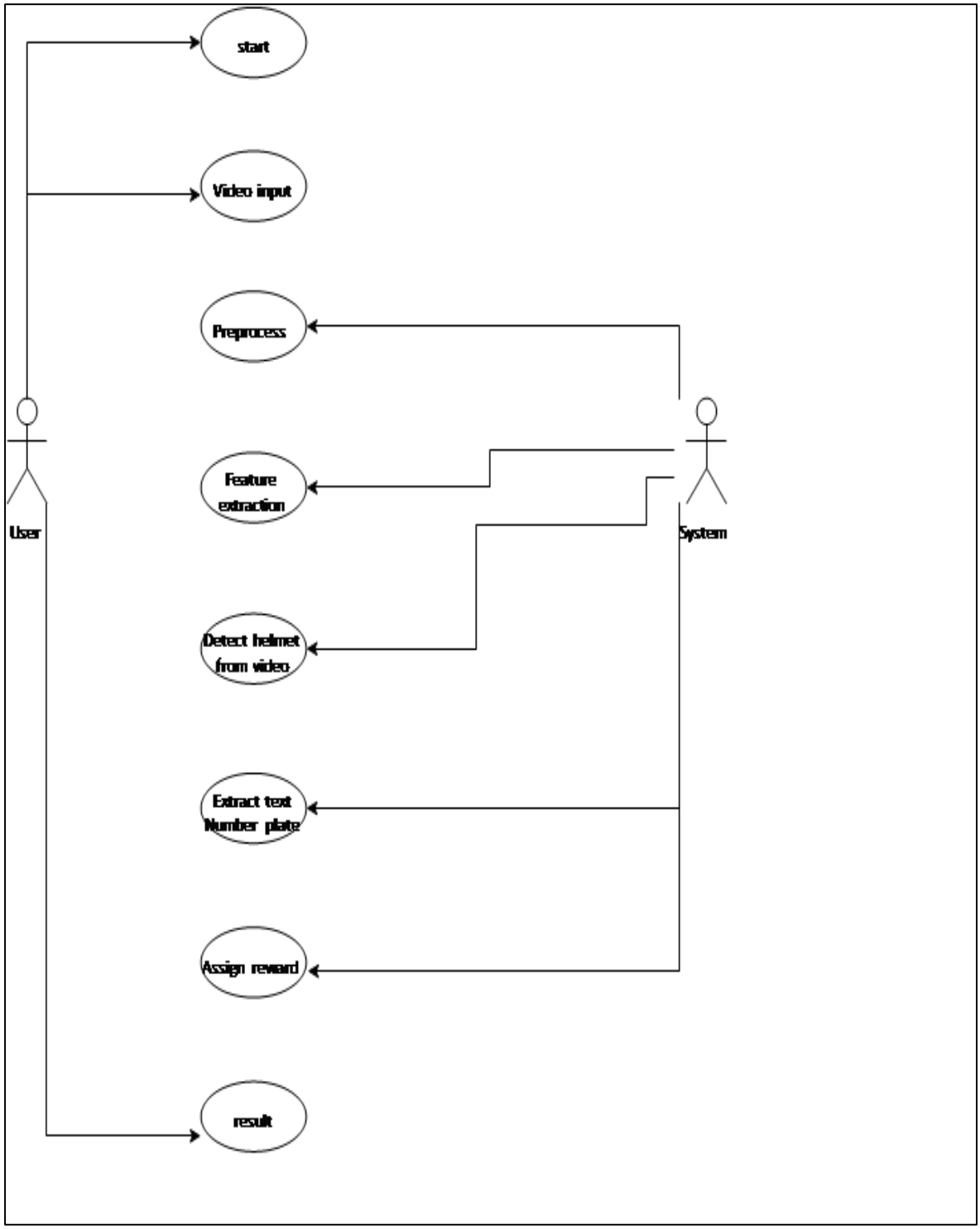


Fig 4.2 : Use Case Diagram

4.3 Control Flow Diagram:

A control flow diagram illustrates the flow of control within a software system, depicting the sequence of execution of instructions or operations. It outlines the paths of program execution, including decisions, loops, and branching. Symbols such as arrows, rectangles, and diamonds represent different control structures and flow direction. Control flow diagrams are crucial for understanding program logic, identifying potential errors, and facilitating program design and debugging.

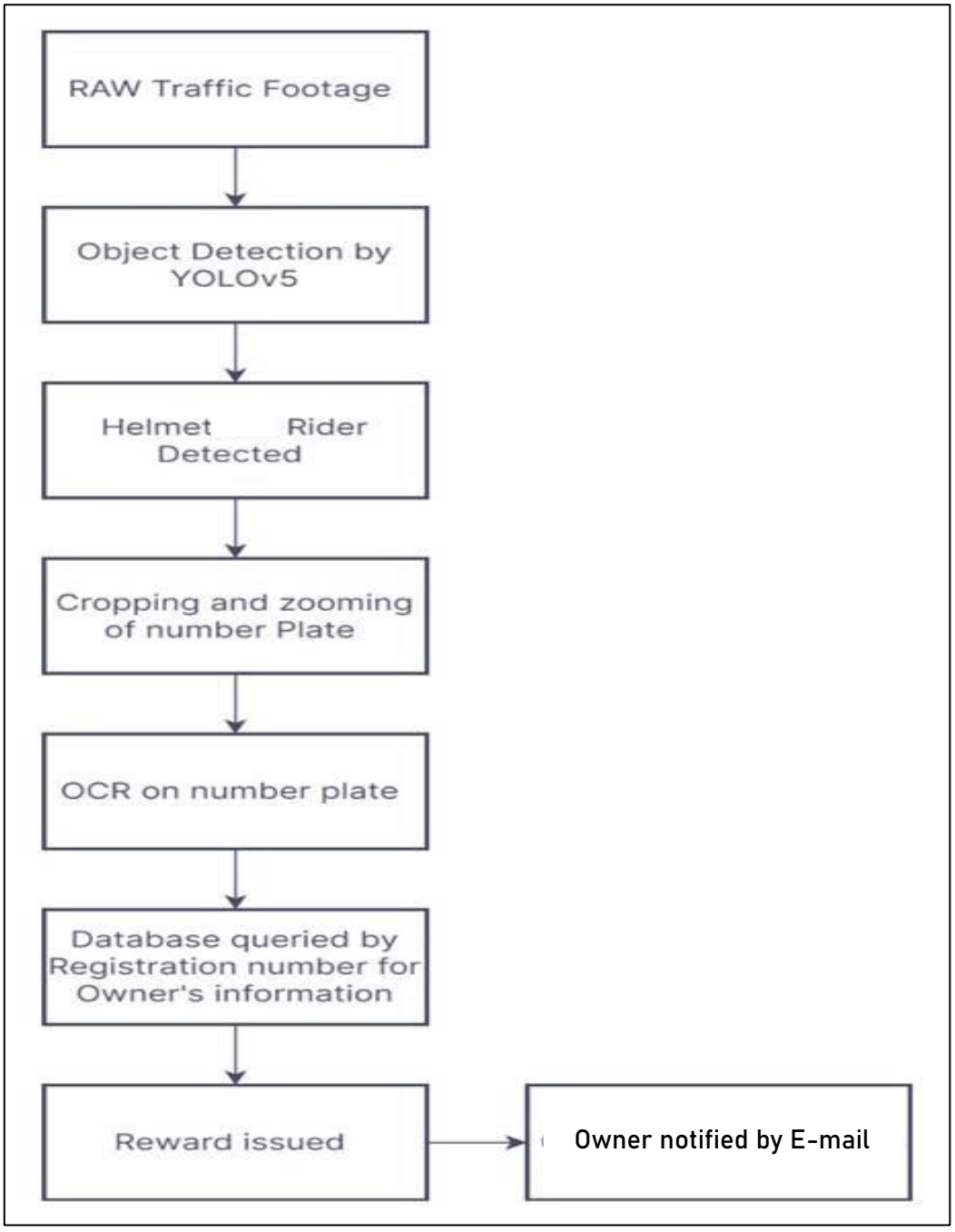


Fig: 4.3: Control Flow Diagram

4.4 Sequence Diagram:

A sequence diagram displays interactions between objects in a sequential manner, illustrating the exchange of messages over time. It helps visualize the order of communication and the lifeline of objects within a system.

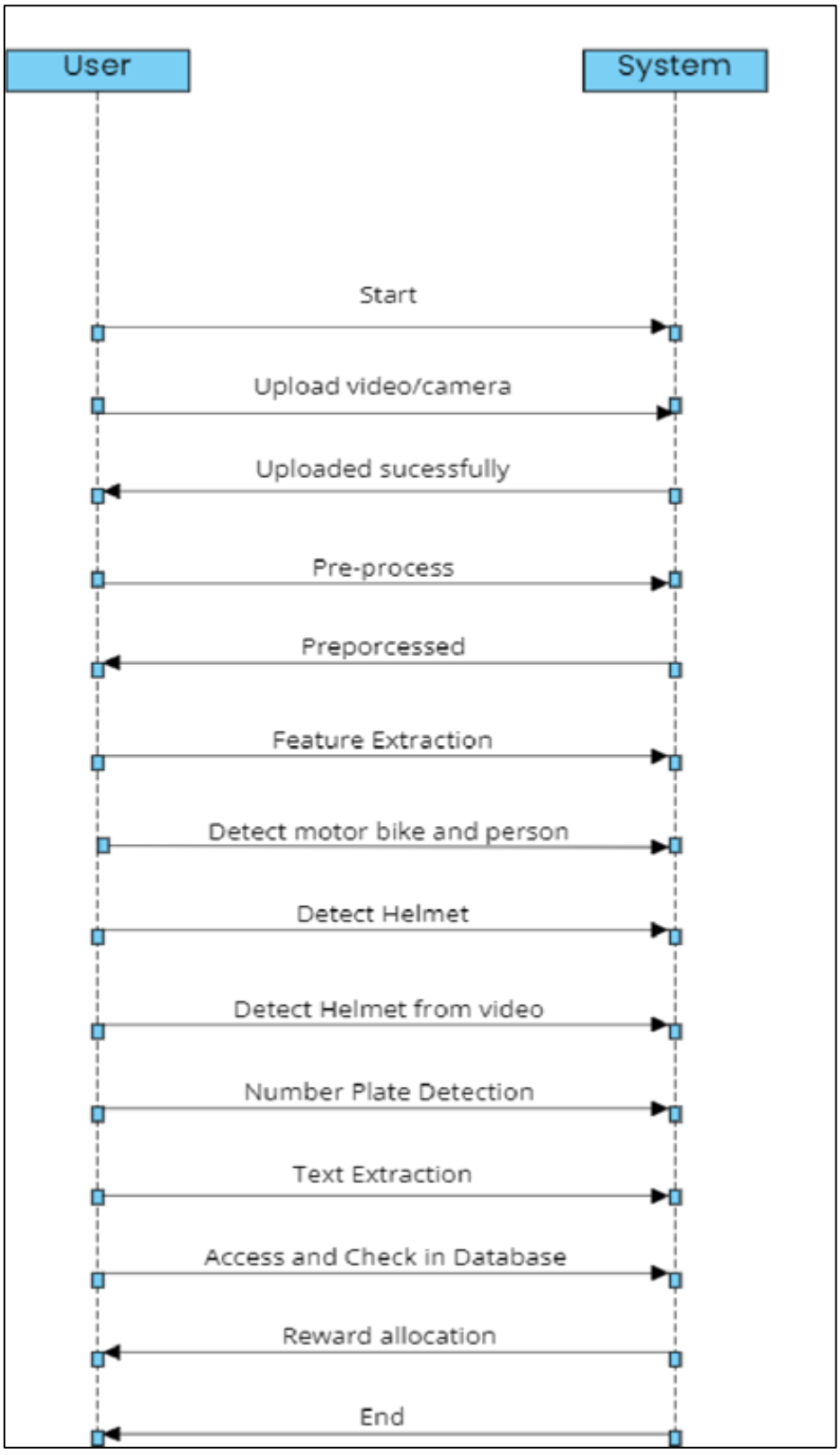


Fig: 4.4: Sequence Diagram

4,5 System Architecture:

A class diagram represents the structure of a system by illustrating classes, their attributes, methods, and relationships. It shows the static view of the system, highlighting the types of objects and their associations. Class diagrams aid in designing, understanding, and documenting the architecture of object-oriented systems.

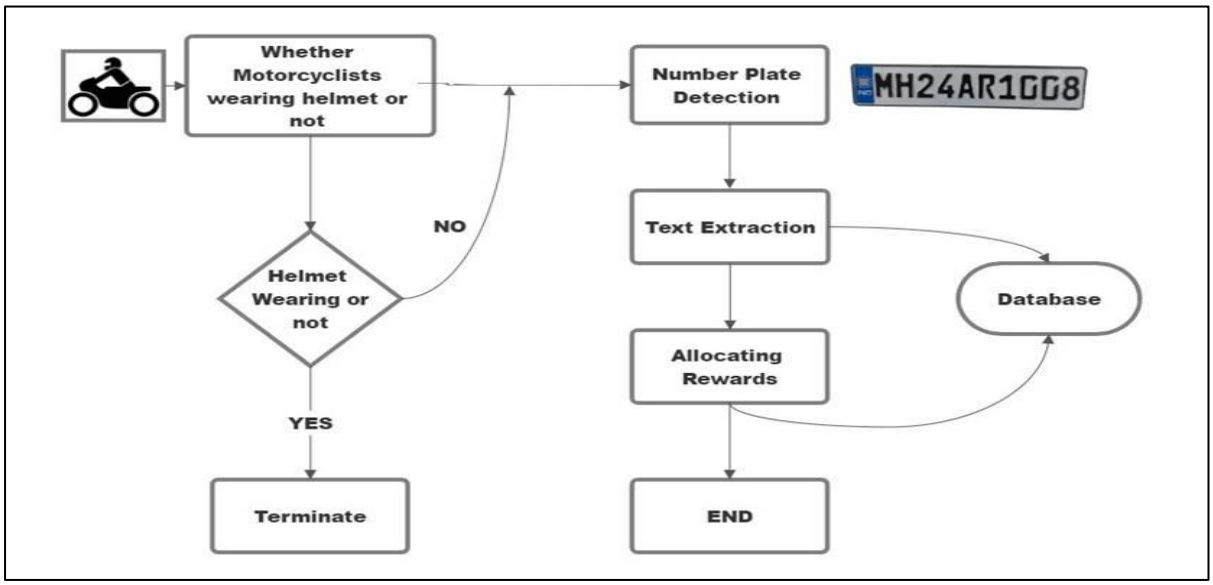


Fig: 4.6: System Architecture

CHAPTER 5
IMPLEMENTATION

5.1 Functionality:

”Smart Helmet Compliance System” includes functionalities like

- Helmet Detection: Utilize computer vision algorithms to detect whether riders are wearing helmets or not
- Number Plate Recognition: Implement algorithms for recognizing and capturing number plates of vehicles
- Database Management: Store and manage rider information, including helmet-wearing behaviour and reward points.
- Point Assignment: Assign points to riders based on their helmet-wearing practices
- Reward System: Implement a system for rewarding riders based on accumulated points.
- Scalability and Performance: Ensure the system can handle a large volume of rider data and perform efficiently in real-time scenarios

5.2 Hardware and Communication Interfaces:

- –Camera Systems: Integration with existing closed-circuit television (CCTV) cameras or ded icated camera systems to capture real-time footage of two-wheeler riders on the road.

- Data Collection Hardware: Utilization of hardware devices for collecting video footage from various sources, including CCTV cameras and dedicated sensors

5.3 System Features:

- Helmet Compliance Tracking: Description: Real-time monitoring of two-wheeler riders to de tect and track instances of helmet usage or non-compliance. Benefits: Ensures adherence to safety regulations and forms the basis for incentivized reward programs. Incentivized Rewards System: Description: Introduction of a dynamic reward mechanism to motivate and

recognize users consistently wearing helmets. Benefits: Encourages a positive culture of responsible riding, fostering long-term compliance and safety awareness.

- Highly Reliable Back-end Infrastructure: Establishment of a robust back-end in frastructure with redundancy and failover mechanisms for high reliability.
  Benefits: Minimizes downtime, ensures system availability, and provides a stable foundation for overall system per formance.
- Secure Data Storage: Description: Utilization of robust encryption techniques to secure user data in storage, including personal information, compliance history, and reward details.
  Benefits: Safeguards sensitive user information, maintaining the integrity and confidentiality of stored data.
- Hardware and Communication Interfaces: Description: Seamless integration with hardware devices such as CCTV cameras, sensors, and communication interfaces like E-mail.

5.4 Programs

- **Object Detection, Model Loading, and Inference**:
  This code defines parameters and options for running a YOLOv5 object detection model. It sets various configurations such as model paths, input image size, confidence threshold, IOU threshold, and other settings for inference. The script defines options for saving results, filtering classes, and visualizing detections. It also includes options for augmentation, updating models, and specifying output paths. Additionally, there are options for controlling visualization aspects like line thickness, hiding labels, and confidence scores.

```
def run(
        weights=ROOT / 'yolov5s.pt',  # model.pt path(s)
        source=ROOT / 'data/images',  # file/dir/URL/glob, 0 for webcam
        data=ROOT / 'data/coco128.yaml',  # dataset.yaml path
        imgsz=(640, 640),  # inference size (height, width)
        conf_thres=0.25,  # confidence threshold
        iou_thres=0.45,  # NMS IOU threshold
        max_det=1000,  # maximum detections per image
        device='',  # cuda device, i.e. 0 or 0,1,2,3 or cpu
        view_img=False,  # show results
        save_txt=False,  # save results to *.txt
        save_conf=False,  # save confidences in --save-txt labels
        save_crop=False,  # save cropped prediction boxes
        nosave=False,  # do not save images/videos
        classes=None,  # filter by class: --class 0, or --class 0 2 3
        agnostic_nms=False,  # class-agnostic NMS
        augment=False,  # augmented inference
        visualize=False,  # visualize features
        update=False,  # update all models
        project=ROOT / 'runs/detect',  # save results to project/name
        name='exp',  # save results to project/name
        exist_ok=False,  # existing project/name ok, do not increment
        line_thickness=3,  # bounding box thickness (pixels)
        hide_labels=False,  # hide labels
        hide_conf=False,  # hide confidences
        half=False,  # use FP16 half-precision inference
        dnn=False,  # use OpenCV DNN for ONNX inference
):
```

- **Reward Points Allocation:**

  This code defines a function **update_reward_points** that updates reward points in a database based on a given license plate. It iterates through the database, increments reward points if the license plate matches, and sends an email notification.In the main section, it iterates through images in a directory, detects license plates using OCR, and updates reward points for detected license plates.

```
def update_reward_points(licensePlate):
    for index, plate in enumerate(database['Registration']):
        if licensePlate == plate:
            database.at[index, 'Reward points'] += 1  # Increment reward points
            print(f"Reward point assigned to {database['Name'][index]}")
            send_email(database['Email'][index])  # Send email
            updated = True
    if updated:
        database.to_csv('database.csv', index=False)

if __name__ == '__main__':
    for path in os.listdir(BASE_DIR):
        path = os.path.join(BASE_DIR, path).replace('Helmet', 'Numberplate')  # Adjust if necessary
        img = cv2.imread(path, 0)
        reader = Reader(['en'])
        number = reader.readtext(img, mag_ratio=3)
        licensePlate = ""

        for i in [0, 1]:
            for item in number[i]:
                if type(item) == str:
                    licensePlate += item

        licensePlate = licensePlate.replace(' ', '')
        licensePlate = licensePlate.upper()
        licensePlate = re.sub(r'[^a-zA-Z0-9]', '', licensePlate)
        print('License number is:', licensePlate)
        update_reward_points(licensePlate)
```

- **E- mail Geneartion:**

This code defines a Python function **send_email** to send an email notification to a specified recipient. It uses the SMTP protocol to connect to a mail server, logs in with the sender's email and password, creates a MIME multipart message with a subject and body, attaches the body to the message, and sends the email. Finally, it closes the connection to the mail server

```
def send_email(receiver_email):
    """Send an email to the user."""
    sender_email = "prasadbharatpatil.5.9@gmail.com"
    password = "shzw hvps deln hnpj"

    message = MIMEMultipart()
    message["From"] = sender_email
    message["To"] = receiver_email
    message["Subject"] = "Reward Point Notification"
    body = "Congratulations! You have received a reward point."
    message.attach(MIMEText(body, "plain"))

    server = smtplib.SMTP("smtp.gmail.com", 587)
    server.starttls()
    server.login(sender_email, password)
    text = message.as_string()
    server.sendmail(sender_email, receiver_email, text)
    server.quit()
```

- **Object detection using a YOLOv5 model on a variety of input images**

This code is a Python script that performs object detection using a YOLOv5 model. It defines parameters and options using argparse, including the model name, input image paths, and inference settings. It imports necessary libraries and modules, initializes the model, and specifies input images. It then performs inference using the model on the input images, prints the results, and optionally saves them. The script is structured to be run from the command line, accepting arguments such as the model name and input image paths.

```python
if __name__ == '__main__':
    import argparse
    from pathlib import Path

    import numpy as np
    from PIL import Image

    from utils.general import cv2, print_args

    # Argparser
    parser = argparse.ArgumentParser()
    parser.add_argument('--model', type=str, default='yolov5s', help='model name')
    opt = parser.parse_args()
    print_args(vars(opt))

    # Model
    model = _create(name=opt.model, pretrained=True, channels=3, classes=80, autoshape=True, verbose=True)
    # model = custom(path='path/to/model.pt')  # custom

    # Images
    imgs = [
        'data/images/zidane.jpg',  # filename
        Path('data/images/zidane.jpg'),  # Path
        'https://ultralytics.com/images/zidane.jpg',  # URI
        cv2.imread('data/images/bus.jpg')[:, :, ::-1],  # OpenCV
        Image.open('data/images/bus.jpg'),  # PIL
        np.zeros((320, 640, 3))]  # numpy

    # Inference
    results = model(imgs, size=320)  # batched inference

    # Results
    results.print()
    results.save()
```

CHAPTER 6

TESTING AND RESULTS

6.1 Software Testing:

Software testing is the process of evaluating a software application or system to ensure it meets specified requirements and functions correctly. It involves executing the software with the intent of finding defects or verifying that it behaves as expected

6.2 Types of Software Testing:

There are two types of test techniques used based upon the project requiements, type of application and goals; they are White Box Techniques and Black Box Techniques.

6.3 White Box Techniques:

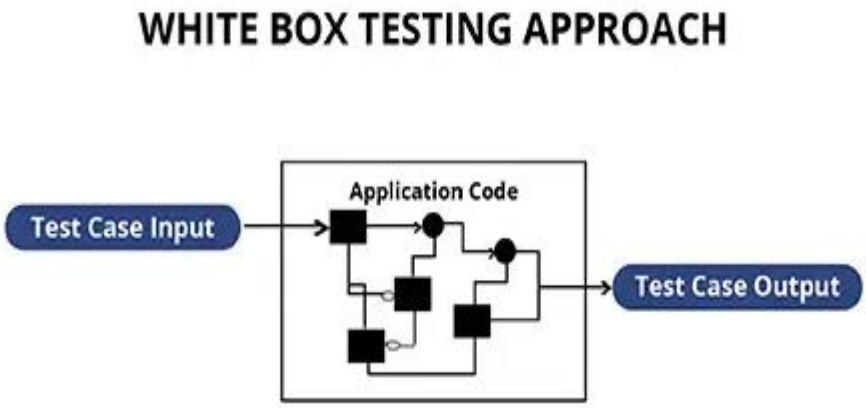


Fig 6.3 : White Box Techniques

White box testing, also known as clear box or glass box testing, examines the internal structure and workings of the software. Testers have access to the source code and design documents, allowing them to design test cases based on the internal logic and paths through the code.

Here are the characteristics of the white box testing –

- White box testing is directly dependent on the software development process. So, you can write the test cases without starting the development process.

- It requires programming language knowledge to understand the app's structure and design. You have to test every function and module of the application.
- This testing technique helps to check the code coverage. That means it can identify the areas of your application that are not tested enough or skipped.
- It identifies logical errors like – infinite loops, coding threads, incorrect conditions, etc. Then it commands to rectify them and improve the functionality of the coding structure.

Types of White Test Techniques:

1. Statement Coverage
2. Decision Coverage (Branch Coverage)
3. Condition Coverage
4. Path Coverage
5. Loop Testing
6. Data Flow Testing
7. Control Flow Testing
8. Branch Condition Testing
9. Code Review
10. Static Analysis

6.4 Black Box Techniques:

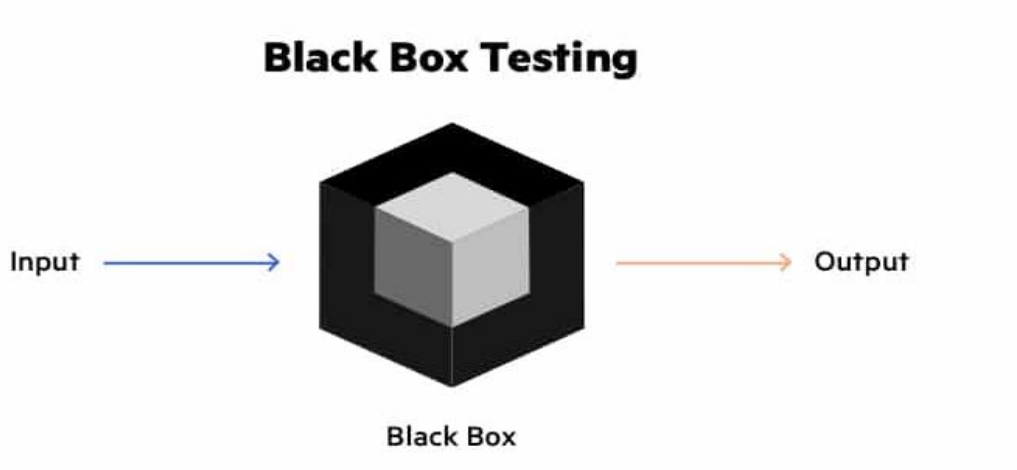


Fig 6.4 : Black Box Techniques

Black box testing, also known as functional testing, focuses on testing the software's functionality without examining its internal code. Testers evaluate the software's inputs and outputs based on specified requirements or expected behavior, without knowledge of its internal implementation.

Characteristics of Black Box Testing

- The QAs can perform this test without knowing the logical structure of your application. Thus they don't require an understanding the algorithm for the system. They just check the external functionalities and behavior of the app. Thus they can test the software independently without involvement in the development procedure.
- It includes a requirement-specific approach for the testing. So you need to enter the specific input for a particular field of your app's interface to obtain the successful test results.
- As you know, it doesn't require too much coding knowledge; it's conducted through the view of an end-user to ensure the app satisfies their expectations and requirements. Also, it confirms the app is easy to handle from the client side.
- You can perform different types of black box testing, such as functional, non-functional, and regression testing.
- You can modify the scalability of this testing depending on the size and complexity of your application.

Types of Black Test Techniques:

1. Equivalence Partitioning
2. Boundary Value Analysis
3. State Transition Testing
4. Decision Table Testing
5. Use Case Testing
6. Pairwise Testing (Combinatorial Testing)
7. Error Guessing
8. Ad Hoc Testing
9. Exploratory Testing
10. Regression Testing

6.5 Test Cases:

1. Input an image with a person wearing a helmet.

- Expected Outcome: The system should detect the helmet, extract the vehicle number plate, find the corresponding user in the database, increment their reward points, and send them an email notification.
- Passed: Expected outcome achieved.

2. Input an image with a person not wearing a helmet.

- Expected Outcome: The system should detect the absence of a helmet, but not increment any reward points or send any email notifications.
- Passed: Expected outcome achieved.

3. Input an image with a clear number plate.

- Expected Outcome: The system should accurately extract the number plate from the image and normalize it for processing.
- Passed: Expected outcome achieved.

4. Input an image with a partially obscured number plate.

- Expected Outcome: The system should still attempt to extract the number plate and handle partial obstructions, ensuring it doesn't crash or produce incorrect results.
- Passed: Expected outcome achieved.

5. Provide a valid license plate that exists in the database.

- Expected Outcome: The system should find the corresponding user in the database and update their reward points.
- Passed: Expected outcome achieved.

6. Provide an invalid license plate that doesn't exist in the database.

- Expected Outcome: The system should handle the case gracefully, perhaps logging the event but not crashing, since no reward points need to be updated.
- Passed: Expected outcome achieved.

7. Verify that an email is sent upon reward point allocation.

- Expected Outcome: The system should successfully send an email to the user's registered email address upon reward point allocation.
- Passed: Expected outcome achieved.

8. Verify that the email content is correct.

- Expected Outcome: The email should contain the appropriate message with the correct reward points total.
- Passed: Expected outcome achieved.

9. Input an image with multiple persons and vehicles.

- Expected Outcome: The system should handle multiple detections, extracting number plates and updating reward points for each valid case.
- Passed: Expected outcome achieved.

10. Test error handling by providing corrupt or non-image input files.

- Expected Outcome: The system should gracefully handle such inputs without crashing, providing appropriate error messages or logging.
- Passed: Expected outcome achieved.

11. : Test the system's performance with large images or a large number of images.

- Expected Outcome: The system should process images within a reasonable time frame and handle a reasonable number of images without memory leaks or performance degradation.
- Passed: Expected outcome achieved.

12. Input an image where the helmet is partially obscured (e.g., by a shadow or another object).

- Expected Outcome: The system should still be able to detect the helmet despite partial obstruction.
- Passed: Expected outcome achieved.

13. Input an image with a person wearing a non-standard helmet (e.g., a construction helmet or a unique design).

- Expected Outcome: The system should be able to differentiate between standard and non-standard helmets and correctly classify them.
- Passed: Expected outcome achieved.

14. Input an image with a person wearing a helmet incorrectly (e.g., worn improperly or tilted).

- Expected Outcome: The system should still recognize the helmet, even if it's not worn conventionally.
- Passed: Expected outcome achieved.

15. Input an image with a distorted number plate (e.g., due to perspective or reflection).

- Expected Outcome: The system should handle distortion and still accurately extract the number plate information.
- Passed: Expected outcome achieved.

16. Input an image with multiple vehicles and number plates.

- Expected Outcome: The system should be able to detect and process multiple number plates in the same image without confusion.
- Passed: Expected outcome achieved.

17. Input an image with a valid license plate but a user not registered in the database.

- Expected Outcome: The system should handle cases where the license plate exists but the user is not in the database, logging the event or providing a suitable notification.
- Passed: Expected outcome achieved.

18. Input an image with a user having the maximum reward points already.

- Expected Outcome: The system should correctly handle cases where the user has reached the maximum reward points and avoid incrementing beyond that limit.
- Passed: Expected outcome achieved.

19. : Input an image with poor lighting conditions.

- Expected Outcome: The system should still be able to detect helmets and number plates under various lighting conditions, adjusting its parameters if necessary.
- Passed: Expected outcome achieved.

20. Simulate network failure during email sending.

- Expected Outcome: The system should handle network failures gracefully, retrying email sending or providing an appropriate error message.
- Passed: Expected outcome achieved.

21. Input a large batch of images for processing.

- Expected Outcome: The system should handle batch processing efficiently without memory leaks or crashes.
- Passed: Expected outcome achieved.

22. Test system stability by running the application continuously for an extended period.

- Expected Outcome: The system should remain stable without memory leaks or degradation in performance over time.
- Passed: Expected outcome achieved.

23. Input an image with helmets of different styles (e.g., full-face, half-face, open-face).

- Expected Outcome: The system should correctly identify and classify different helmet styles.
- Passed: Expected outcome achieved.

24. Input an image taken in adverse weather conditions (e.g., rain, fog, snow).

- Expected Outcome: The system should still be able to detect helmets and number plates despite poor weather conditions.
- Passed: Expected outcome achieved.

25. Run the system on different hardware configurations (e.g., varying CPU/GPU capabilities).

- Expected Outcome: The system should be compatible and perform consistently across different hardware setups without significant differences in results.
- Passed: Expected outcome achieved.

26. Large image with multiple persons and motorbikes:

- Expected Outcome: System accurately detects and rewards individuals wearing helmets and identifies motorbikes with number plates.
- Passed: Expected outcome achieved.

27. Integration Testing - Complete workflow:

- Expected Outcome: System integrates helmet detection, number plate recognition, and reward allocation seamlessly, providing accurate results for each image.
- Passed: Expected outcome achieved.

28. Security Testing - Adversarial input:

- Expected Outcome: System identifies the adversarial input, flags it as suspicious, and does not assign a reward.
- Passed: Expected outcome achieved.

29. Performance Testing - Large dataset:

- Expected Outcome: System processes the entire dataset within a reasonable time frame and allocates rewards accurately.
- Passed: Expected outcome achieved.

6.6 Results:

- **Input**

- **Frames Creation:**

```
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'test_video2.mp4':
  Metadata:
    major_brand     : mp42
    minor_version   : 0
    compatible_brands: mp42isom
  Duration: 00:00:07.91, start: 0.000000, bitrate: 1829 kb/s
  Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p(tv, bt470bg/bt470bg/smpte170m), 478x850, 1593 kb/s, 30.04 fps, 30 tbr, 90k tbn, 18
0k tbc (default)
    Metadata:
      vendor_id       : [0][0][0][0]
  Stream #0:1(und): Audio: aac (LC) (mp4a / 0x6134706D), 48000 Hz, stereo, fltp, 256 kb/s (default)
    Metadata:
      vendor_id       : [0][0][0][0]
Stream mapping:
  Stream #0:0 -> #0:0 (h264 (native) -> h264 (libx264))
  Stream #0:1 -> #0:1 (aac (native) -> aac (native))
Press [q] to stop, [?] for help
[libx264 @ 0x5974877b89c0] using cpu capabilities: MMX2 SSE2Fast SSSE3 SSE4.2 AVX FMA3 BMI2 AVX2
[libx264 @ 0x5974877b89c0] profile High, level 2.2, 4:2:0, 8-bit
[libx264 @ 0x5974877b89c0] 264 - core 163 r3060 5db6aa6 - H.264/MPEG-4 AVC codec - Copyleft 2003-2021 - http://www.videolan.org/x264.html - options: c
abac=1 ref=3 deblock=1:0:0 analyse=0x3:0x113 me=hex subme=7 psy=1 psy_rd=1.00:0.00 mixed_ref=1 me_range=16 chroma_me=1 trellis=1 8x8dct=1 cqm=0 deadzo
ne=21,11 fast_pskip=1 chroma_qp_offset=-2 threads=6 lookahead_threads=1 sliced_threads=0 nr=0 decimate=1 interlaced=0 bluray_compat=0 constrained_intr
a=0 bframes=3 b_pyramid=2 b_adapt=1 b_bias=0 direct=1 weightb=1 open_gop=0 weightp=2 keyint=250 keyint_min=1 scenecut=40 intra_refresh=0 rc_lookahead=
40 rc=crf mbtree=1 crf=23.0 qcomp=0.60 qpmin=0 qpmax=69 qpstep=4 ip_ratio=1.40 aq=1:1.00
Output #0, mp4, to 'output.mp4':
```

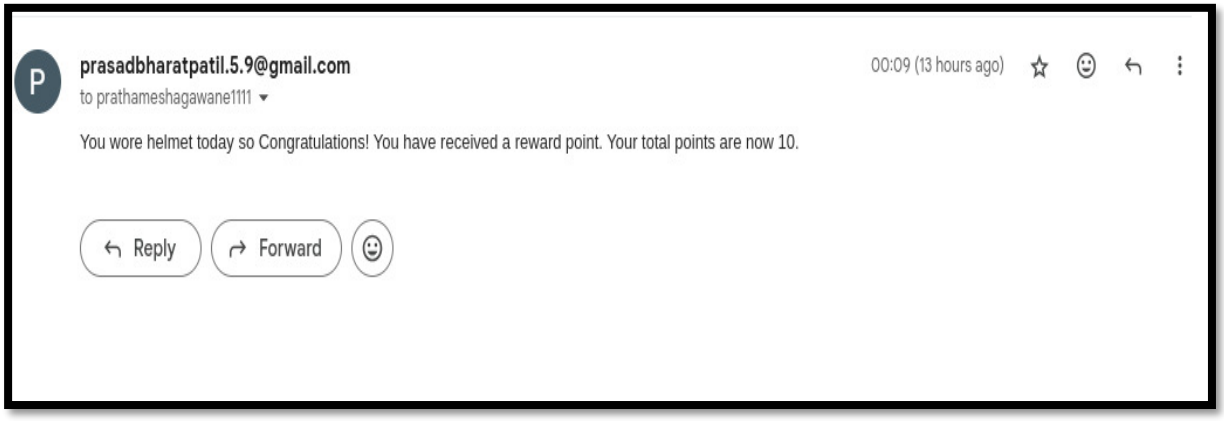
- **Creation of Classes:**

- **Object Detection with labels as the output:**

- **Database updated with rewards:**

Name	Registration	Phone numbe	Email	Reward point
Pooja Patil	MH01CB4422	91824033672	ppoooja13@gmail.com	0
Prathamesh A	JK04B8946	91938279550	prathameshagawane1111@gmail	11
Testing	WB12AU8698	91824033672	testing@gmail.com	0
Prasad Patil	MH12KG7585	9146866355	prasadpatil2255@gmail.com	74

- **User Notified with E-mail Notification**



- **Object Detection from the video:**

Here in the above output image is been detected from the video(INPUT) the frames have been created and the features like helmet, motorbike, and number plate is detected successfully.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

In conclusion, our project addresses the critical issue of motorcycle accidents caused by non-compliance with helmet regulations through the development of an automated surveillance system. By leveraging advanced technologies such as YOLOv4 for object detection and OCR for license plate recognition, we have created a system capable of real-time identification of motorcyclists without helmets. This system not only streamlines enforcement efforts but also introduces a rewards mechanism to incentivize helmet usage, thereby fostering a culture of safety among riders.

The implementation of our automated system offers numerous benefits over traditional surveillance methods, including increased efficiency, reduced reliance on human intervention, and improved accuracy in detecting non-compliant riders. By automating the detection process and providing real-time feedback to both compliant and non-compliant riders, we aim to create a safer environment on the roads and reduce the incidence of motorcycle accidents.

Moving forward, there is potential for further enhancement and refinement of our system, including integration with existing traffic management systems and expansion to cover a broader range of safety violations. Additionally, ongoing monitoring and evaluation will be essential to assess the effectiveness of our system in promoting helmet usage and reducing motorcycle accidents.

Ultimately, our project represents a significant step towards improving road safety and protecting the lives of motorcyclists. By combining innovative technology with proactive measures, we aspire to create a positive impact on society and contribute to the well-being of all road users.

7.2 Future Work

While our proposed system has demonstrated effectiveness in addressing the challenge of detecting motorcyclists without helmets in real-time surveillance footage, there are several avenues for future research and development to further enhance its capabilities and applicability.

1. **Improving Image Processing Algorithms**: Future work could focus on refining image processing algorithms to better handle challenges such as poor image quality, variations in lighting conditions, and changes in camera angles. Advanced techniques in image enhancement and noise reduction could enhance the accuracy and robustness of our system.
2. **Expansion to Other Traffic Violations**: Building upon the foundation laid by our project, future endeavors could explore the detection of other traffic violations beyond helmet non-compliance. This could include identifying cars violating road rules, illegal parking, or speeding vehicles, thereby contributing to overall road safety and law enforcement.
3. **Integration with Traffic Management Systems**: Our system could be integrated with existing traffic management systems to provide real-time feedback to traffic authorities and facilitate immediate response to violations. This integration could enhance traffic monitoring and enforcement efforts, leading to smoother traffic flow and improved safety on roads.
4. **Automated Toll Collection**: Leveraging the number plate recognition capabilities of our system, future work could explore applications in automated toll collection systems. By automatically detecting vehicles and associating them with toll payment accounts, our system could streamline toll collection processes and reduce traffic congestion at toll booths.
5. **Exploring Applications Beyond Traffic Management**: Beyond traffic management, our system could find applications in other areas such as parking lot management and security surveillance. By detecting and monitoring vehicles in parking lots, it could help prevent illegal parking, vehicle theft, and other security incidents.

REFERENCES

[1] K. Dahiya, D. Singh and C.K.Mohan, "*Automatic detection of bike riders Without helmet using surveillance videos in real-time",* Proceeding of International Joint Conference Neural Networks (IJCNN), Vancouver, Canada, 24-2 July 2016, pp.3046-3051.

[2] R. V. Silva, T. Aires, and V. Rodrigo, *"Helmet Detection on Motorcyclists Using image descriptors and classifiers"*, Proceeding of Graphics, Patterns and Images (SIBGRAPI), Rio de Janeiro, Brazil, 27-30 August 2014

[3] Pathasu Doughmala, Katanyoo Klubsuwan, "*Half and Full Helmet Detection in Thailand using Haar Like Feature and Circle Hough Transform on Image Processing*"

Proceeding of IEEE International Conference on Computer and Information Technology, Thailand, Bangkok, pg. 611-614, 2016.

[4] J. Chiverton, *"Helmet presence classification with motorcycle detection and Tracking",*

IET Intelligence Transport Systems (ITS), Volume 6, Issue no. 3, pp. 259-269, 2012.

[5] C.C. Chiu, M.Y. Ku, and H. -T. Chen, "*Motorcycle detection and tracking system with occlusion segmentation"* Proceeding of International Workshop on Image Analysis for Multimedia Interactive Services, Santorini, Greece*,* 6-8 June 2007, pp. 32-32.

[6] C.Y.Wen, S.H.Chiu, J.J.Liaw, and C.P. Lu, "*The safety helmet* detection for ATM's surveillance system via the modified Hough *transform"* Proceedings of IEEE 37th Annual International Carnahan Conference on Security Technology, pp. - 364-369, 2003.

[7] Romuere Silva, Kelson Aires, Rodrigo Veras, Thiago Santos, Kalyf Lima and Andre Soares, "*Automatic motorcycle detection on public road* "CLEIELECTRONIC JOURNAL, Volume 16, Number 3, Paper 04, December 2013.