TWO-WHEELER TRAFFIC RULE VIOLATION DETECTION

Team Members

Pratham Jaiswal 20BCE1795

Arnav Rawat 20BCE1395

Shubham Sharma 20BCE1631

Ayush Jain 20BCE1073

P. Satyanarayana Reddy 20BRS1263

Guide

Geetha S.

School name: School of Computing Science and Engineering

Faculty name (ERP No): S.Geetha, (50587)

Academic Year: 2023-2024 (Winter Semester 23-24)

Title of the project: Two-Wheeler Traffic Rule Violation Detection

Number of students involved in this project: 5

Pratham Jaiswal
20BCE1795

Arnav Rawat
20BCE1395

Shubham Sharma
20BCE1631

Ayush Jain
20BCE1073

P.Satyanarayana Reddy
20BRS1263

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

DECLARATION

We hereby declare that the thesis entitled **"Two-Wheeler Traffic Rule Violation Detection"** submitted by **Pratham Jaiswal (20BCE1795)**, **Arnav Rawat (20BCE1395)**, **Shubham Sharma (20BCE1631)**, **Ayush Jain (20BCE1073)**, and **Satyanarayana Reddy (20BRS1263)** for the completion of the course, **Technical Answers For Real World Problems (CSE1901)** is a record of bonafide work carried out by me under the supervision of **Dr. Geetha S**, our course instructor. We further declare that the work reported in this document has not been submitted and will not be submitted, either in part or in full, for any other courses in this institute or any other institute or university.

Place: Chennai

Pratham Jaiswal 20BCE1795

Arnav Rawat 20BCE1395

Shubham Sharma 20BCE1631

Ayush Jain 20BCE1073

Satyanarayana Reddy 20BRS1263

Date:

Signature of Candidates:

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School Of Computer Science And Engineering

CERTIFICATE

This is to certify that the report entitled **"Two-Wheeler Traffic Rule Violation Detection"** is prepared and submitted by **Pratham Jaiswal (20BCE1795), Arnav Rawat (20BCE1395), Shubham Sharma (20BCE1631), Ayush Jain (20BCE1073), and Satyanarayana Reddy (20BRS1263)** to **Vellore Institute of Technology, Chennai**, in partial fulfillment of the requirement for the course, **Technical Answers For Real World Problems (CSE1901),** is a bonafide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for any other course and the same is certified.

Dr. Geetha S

Guide

ACKNOWLEDGEMENT

We hereby want to acknowledge the support and guidance of all those who helped me completing this project. We want to extend my special thanks to Dr. Nithyanandam P, Head of the Department (HoD), B.Tech Computer Science and engineering CSE, VIT Chennai and Dr. Ganesan R, Dean of the School of Computer Science & engineering, VIT Chennai for giving me the opportunity to complete this project. We thank Dr. Parvathi R, Associate Dean (Academics) of the School of Computer Science & engineering, VIT Chennai and Dr. Geetha S, Associate Dean (Research) of the School of Computer Science & engineering, VIT Chennai for helping me out in this project. We also acknowledge with a deep sense of reverence, my gratitude towards my parents and my friends for their valuable moral support and suggestions.

Table of Contents

List of Figures

Chapter 1: Introduction

1.1 Abstract

Ensuring road safety is paramount in modern society, and the increasing prevalence of two-wheelers on roadways demands a robust approach to enforce traffic rules effectively. This project addresses the pressing need for improved road safety by focusing on the detection and identification of traffic rule violations committed by two-wheeler riders.

The significance of road safety cannot be overstated, as it directly correlates with reducing accidents, injuries, and fatalities. Two-wheeler riders, vulnerable to road hazards, contribute significantly to traffic violations, necessitating targeted measures to enhance compliance with traffic regulations. This project aims to leverage advanced technology to detect violations committed by two-wheeler riders, focusing on wrong lane usage, lack of helmet adherence, and instances of triple riding.

The choice of this project is underscored by the escalating challenges posed by two-wheeler traffic violations. Traditional methods of monitoring and enforcement are often insufficient in addressing the nuanced and dynamic nature of these violations. By integrating cutting-edge technologies, such as the Roboflow Detection Model APIs and OCR.Space API, this project seeks to enhance the efficiency and accuracy of traffic rule enforcement.

The project's focus on detecting violations such as wrong lane usage, absence of helmets, and instances of triple riding aligns with the broader goal of creating a safer road environment. The systematic process flow, encompassing motorcycle detection, orientation checks, face and helmet detection, and license plate recognition, ensures a comprehensive approach to identifying and documenting violations.

This project represents a crucial step towards mitigating the risks associated with two-wheeler traffic violations, contributing to overall road safety. Through the integration of advanced technologies and a meticulous process flow, the system aims to not only identify violations but also provide a basis for informed and targeted enforcement efforts, ultimately fostering a safer and more secure road environment for all.

1.2 Problem Statement

The current traffic management systems often lack robustness in addressing two-wheeler-specific rule violations, posing a significant challenge to road safety. Two-wheeler riders frequently flout traffic regulations, such as helmet non-compliance, unauthorized lane usage, triple riding, and obscured license plates. Existing surveillance mechanisms often fall short in effectively identifying and documenting these violations, leading to inadequate enforcement. This project aims to tackle these challenges by implementing a comprehensive Two Wheeler Traffic Rule Violation Detection system. The goal is to develop a solution that utilizes advanced machine learning models for precise detection of specific infractions, providing a more accurate and automated approach to enforce traffic rules for two-wheelers, ultimately contributing to enhanced road safety and compliance.

1.3 Motivation

The motivation behind this project stems from the pressing need to address the escalating challenges posed by non-compliance with traffic regulations among two-wheeler riders. Recognizing the inherent vulnerability of two-wheeler users and the heightened risks associated with rule violations, there is a critical imperative to deploy advanced technological solutions for more effective monitoring and enforcement. By leveraging state-of-the-art machine learning models, this project seeks to revolutionize the conventional approaches to traffic management, offering a proactive and automated system capable of identifying specific violations such as helmet negligence, improper lane usage, and obscured license plates. The ultimate aim is to contribute to a safer road environment by harnessing the power of technology to encourage responsible behavior and ensure the adherence of two-wheeler riders to traffic rules.

1.4 Introduction

In our ever-evolving urban landscapes, road safety stands as a cornerstone for fostering secure and efficient transportation systems. The surge in two-wheeler usage demands innovative approaches to ensure adherence to traffic regulations, ultimately minimizing accidents and safeguarding lives. This project takes a significant stride towards bolstering road safety by concentrating on the detection and identification of traffic rule violations committed by two-wheeler riders.

The overarching objective of this project is to develop a sophisticated system capable of accurately detecting violations such as wrong lane usage, absence of helmets, and instances of triple riding. However, beyond mere detection, the system's focus extends to evidence capture, recognizing the paramount importance of documenting and recording violations for effective enforcement and subsequent legal proceedings.

The critical need for such a system arises from the unique challenges posed by two-wheeler traffic violations. Conventional methods of monitoring and enforcement often fall short in addressing the dynamic nature of these violations. By leveraging cutting-edge technologies, such as the Roboflow Detection Model APIs and OCR.Space API, the project aims to enhance the precision and efficiency of traffic rule enforcement.

In a departure from traditional approaches, this system not only identifies violations but also goes a step further by capturing comprehensive evidence. Beyond merely flagging instances of non-compliance, it records visual evidence, including images of the violated motorcycle, faces of the riders, and their license plates. This holistic approach not only facilitates immediate enforcement actions but also provides a valuable repository of evidence for future reference, analysis, and legal proceedings.

As we delve into the intricacies of this project, it becomes evident that it is not just about technological innovation; it is about crafting a safer, more accountable, and responsible road ecosystem. By focusing not only on detection but also on robust evidence capture, this system aims to contribute significantly to the overarching goal of creating roadways that prioritize the safety and well-being of all commuters.

Chapter 2: Related Works

The paper addresses the escalating issue of traffic rule violations in developing countries, stemming from population growth and an increasing number of vehicles. Existing automated technologies face challenges in handling nonuniform illumination and diverse license plate formats. To overcome these obstacles, the authors propose a system integrating traffic signal detection and speed estimation. This comprehensive approach aims to identify and manage violations effectively. The incorporation of traffic signal detection enhances the system's capability to pinpoint violations at intersections, while speed estimation provides insights into over-speeding incidents. The system's output, including information on violations and speed data, is stored in a database for authorities to take necessary actions. By combining these parameters and leveraging a centralized database, the proposed system offers a promising solution to enhance traffic management and road safety in the face of increasing challenges in developing countries. [1]

The surge in new vehicles on roads has precipitated congested traffic conditions, prompting a concerning uptick in traffic rule violations and subsequent road accidents. To combat this issue, the paper advocates for the implementation of a traffic violation detection system utilizing computer vision, with a focus on optimizing accuracy. Employing YOLOV3 object detection, the system targets key violations such as signal jumps, vehicle speed infractions, and vehicle count. The paper's approach involves sophisticated techniques, including region of interest selection and tracking the location of vehicles within frames, resulting in an impressive accuracy of 97.67% for vehicle count detection and 89.24% for speed violation detection. The literature survey contextualizes this work within the broader landscape of computer vision applications in traffic management. It reveals a growing trend in leveraging YOLO-based models for real-time object detection in dynamic traffic scenarios. Previous research in traffic signal violation detection aligns with the paper's approach, employing deep learning techniques to identify signal jumps. The inclusion of speed violation detection aligns with broader efforts to mitigate speed-related accidents, with various studies exploring computer vision and machine learning algorithms for accurate speed estimation. The optimization strategies employed in the proposed system resonate with a wider body of literature focused on enhancing the precision and efficiency of object detection systems. In summary, the paper contributes to a burgeoning field

by providing a comprehensive and optimized solution to traffic rule violations, leveraging state-of-the-art computer vision techniques for enhanced road safety and traffic regulation in the face of escalating vehicular density. [2]

The escalating global population and the heightened demand for automobiles, particularly in urban areas, have precipitated severe traffic congestion, escalating the peril of traffic violations on a worldwide scale. This surge in vehicular density not only imperils road safety but also fosters a decline in public awareness, resulting in an upswing of accidents and tragic fatalities. To address these pressing challenges, there is an imperative need for automated traffic violation detection systems capable of enforcing regulations and mitigating human complacency. The proposed system, as outlined in the abstract, endeavors to swiftly identify signal violations, promptly notifying individuals of the impending consequences for breaching traffic laws. Its purported efficiency surpasses human capabilities, especially in comparison to traditional methods where traffic police manually capture images of violators, limiting their ability to address multiple violations simultaneously. The system leverages cutting-edge computer vision techniques to provide real-time detection of various traffic violations, presenting a promising avenue to enhance road safety globally. The reported high accuracy positions the system as a reliable tool for law enforcement agencies seeking to improve traffic management and reduce accidents. In conclusion, the paper contributes to the ongoing discourse surrounding the use of technology, particularly computer vision, to address the complex and global issue of traffic violations. The proposed system, emphasizing real-time detection, efficiency, and accuracy, offers a robust solution to enhance road safety and regulate traffic in the face of increasing urbanization and vehicle ownership. [3]

The paper addresses the crucial issue of ensuring safety measures on Indian roads through the identification of traffic rule violators. The complexities associated with this task, such as occlusion and illumination challenges, underscore the need for advanced technologies. The proposed end-to-end framework integrates various deep learning techniques to detect violations, notify violators, and store relevant data for analysis and statistical generation, facilitating informed decision-making for traffic rules policies. The initial step involves vehicle detection using the YOLO (You Only Look Once) object detection method. Subsequently, each detected vehicle undergoes scrutiny for specific violations, such as helmet non-compliance and crosswalk

infringements. Helmet violation is identified through a Convolutional Neural Network (CNN)-based classifier, while crosswalk violation detection employs Instance Segmentation through the Mask R-CNN architecture. Following violation detection, Optical Character Recognition (OCR) is employed to extract vehicle numbers for the respective violators, facilitating subsequent notification. This proposed autonomous system presents a comprehensive solution to enforce stringent regulation of traffic rules, showcasing the integration of state-of-the-art deep learning techniques for effective and automated violation detection. The framework's holistic approach, encompassing detection, notification, and data storage, contributes to the broader discourse on leveraging deep learning for enhancing traffic management and safety. The paper underscores the significance of technological advancements in addressing road safety challenges, particularly in the context of developing countries like India, and highlights the potential of end-to-end autonomous systems in fostering robust enforcement of traffic regulations. [4]

The paper addresses the increasing challenges within existing traffic management systems, particularly in developing countries, where a surge in agonizing accidents has been observed. The primary culprits identified are over-speeding and violations such as unnecessary lane changes, both of which significantly contribute to the escalating number of accidents and unexpected fatalities. The paper underscores the urgent need to address this issue, presenting a solution implemented through Raspberry Pi and OpenCV contour detection technology. The authors describe the development of a prototype device intended for installation in traffic surveillance cameras near traffic signal positions, with connectivity to metropolitan traffic servers. The system operates by detecting vehicles violating specified traffic rules, promptly sending the collected data to the server for further action. The proposed intelligent traffic monitoring system aims to streamline data collection by minimizing manual intervention, thus reducing time wastage and associated costs. Additionally, the system aims to enhance the enforcement of traffic regulations by identifying and documenting rule violations, aiding traffic management departments in applying laws more rigorously. The reported accuracy of the proposed model stands at 78.83%, signifying its potential to contribute significantly to the reduction of daily accidents. In the broader literature, the use of intelligent systems for traffic management has garnered attention as a means to address the complexities of modern traffic scenarios. Various studies have explored the integration of technologies such as Raspberry Pi and

computer vision for real-time monitoring and violation detection. The proposed system aligns with this trend, showcasing the practical application of emerging computing technologies to enhance traffic safety. The focus on reducing manual data collection resonates with a broader movement toward automation in traffic management, emphasizing the importance of efficiency and accuracy. The reported accuracy of 78.83% positions the system as a promising tool for mitigating accidents caused by lane-based rule violations. This paper contributes to the discourse on leveraging intelligent systems to address traffic management challenges, particularly in developing regions, offering a practical solution to enhance rule enforcement and road safety. The integration of Raspberry Pi and OpenCV in the proposed system underscores the versatility and adaptability of emerging computing technologies in tackling real-world problems, showcasing their potential to revolutionize conventional traffic management approaches and contribute to safer road environments. [5]

The paper focuses on real-time traffic rule infringement detection using video processing techniques, specifically targeting wrong-way and clearway violations. Highlighting the broader applications of video processing in traffic research, such as traffic density measurement and crash detection, the authors emphasize the critical need for early detection of violations to enhance traffic flow and safety. The developed system incorporates features for vehicle detection, tracking, security lane monitoring, and the crucial detection of wrong-way violations, which are identified as major contributors to accidents resulting in significant loss of life and property. The paper underscores the importance of real-time detection to promptly address security lane violations, hindering the use of specific lanes by emergency units, and wrong-way incidents that pose substantial risks. Overall, the proposed system aligns with the broader trend of leveraging artificial intelligence and data processing for traffic management, offering a technologically advanced solution for immediate rule violation identification and intervention.[6]

The paper delves into the realm of traffic violation detection using machine vision, with a specific focus on swerving and blocking pedestrian lanes. The proposed algorithm leverages the background difference method, emphasizing the integration of a genetic algorithm to enhance the system's efficacy in detecting these specific violations. The process involves subtracting a captured image by a reference image, followed by the execution of a genetic algorithm to identify violators. The results indicate an average convergence of approximately 8 iterations and

fewer than 300 generations, showcasing the algorithm's suitability for real-time implementation in traffic detection systems. The choice of utilizing captured photos from a CCTV camera as system inputs underscores the practicality of the proposed solution. The outputs, comprising cropped pictures of the identified vehicles with the mentioned violations, contribute to the specificity and clarity of the system's results. In the broader literature, the integration of machine vision and genetic algorithms for traffic violation detection reflects a sophisticated approach to addressing road safety challenges. Various studies have explored the use of machine vision for real-time traffic monitoring, with an emphasis on enhancing detection accuracy and efficiency. Genetic algorithms, known for their optimization capabilities, have been employed in diverse applications, including traffic management. The specific focus on swerving and blocking pedestrian lanes aligns with the broader effort to address nuanced traffic violations, contributing to a comprehensive approach to road safety. The paper's findings affirm the viability of the proposed algorithm for real-time implementation, offering insights into its convergence efficiency and potential applications in traffic surveillance. This work contributes to the ongoing discourse on the integration of advanced technologies in traffic management, emphasizing the role of machine vision and genetic algorithms in creating effective and responsive systems for detecting and addressing specific traffic violations. [7]

In this paper, a novel helmet identification methodology is introduced, leveraging a dual-pronged approach to enhance detection rates. The first method employs a Haar-like feature-based face detection technique, aimed at effectively distinguishing individuals donning complete helmets from those without. Complementing this, the second method integrates the circle Hough transform, offering a sophisticated means of discerning between individuals not wearing helmets and those sporting half-helmets. By synergizing these two methodologies, the paper aims to significantly improve the overall accuracy and efficiency of helmet detection systems, showcasing a promising stride in the realm of computer vision and safety technologies. [8]

The paper introduces a method capitalizing on the distinctive geometric shapes of helmets and exploiting lighting variations within different helmet components. Notably, the strategy harnessed the Hough transform to implement a circular arc detection mechanism, resulting in remarkable accuracy in helmet detection. Despite these promising outcomes, the paper underscores a potential limitation—due to the relatively low number of test photos employed for

evaluation, there arises a need for more extensive validation. This highlights a crucial aspect, suggesting that further testing and refinement are essential before these techniques can be deemed reliable for practical implementation in real-world scenarios. [9]

The authors deployed adaptive background removal techniques on video frames to discern dynamic entities, with a primary focus on bikers. Leveraging the power of Convolutional Neural Networks (CNNs), the investigators sought to precisely differentiate motorcyclists from the identified moving objects, honing in particularly on individuals lacking proper helmet adherence. This innovative methodology showcased the integration of cutting-edge computer vision and deep learning technologies to enhance the accuracy and efficiency of helmet detection within a dynamic video environment. The study's findings illuminate a promising stride in the realm of object identification, shedding light on the potential applications of advanced neural network architectures in addressing crucial safety concerns related to helmet usage among motorcyclists.[10]

The authors devised a comprehensive system employing image processing techniques and Convolutional Neural Networks (CNNs) to identify helmet infractions. This intricate system encompassed three pivotal components: motorbike detection, categorization of helmet versus no-helmet instances, and the identification of motorbike license plates. Despite the success in accurately detecting helmet violations, the computational complexity of these techniques posed a significant drawback, especially in terms of real-time applications. The system's limitation to helmet detection alone highlighted the need for streamlined, more efficient methodologies to cater to the demands of practical, real-world implementation. [11]

This study introduces a vigilant system designed to scrutinize diverse scenarios and pinpoint instances of helmet violations. However, it operates exclusively with static photos and is tailored to address a specific type of infringement. Notably, recent advancements in the YOLO (You Only Look Once) algorithm have substantially enhanced the efficiency of techniques, enabling them to meet real-time requirements more expeditiously. The evolution in algorithmic approaches underscores a pivotal shift toward faster and more dynamic solutions for identifying and addressing helmet violations, showcasing the continual progress in the field of computer vision and surveillance technologies. [12]

In a significant stride towards enhancing road safety, the study showcased a deep learning system for the identification of triple riding and speed infractions on two-wheelers. Leveraging a combination of Convolutional Neural Networks (CNNs) and the YOLO (You Only Look Once) algorithm, the system demonstrated swift detection capabilities. However, it was discerned that the framework primarily concentrated on a singular type of infringement, potentially limiting its applicability to a broader range of traffic violations. Moreover, the observed accuracy of the system was notably low, suggesting room for improvement and refinement to meet the stringent demands of precise and reliable traffic violation detection. [13]

Tthe paper suggests an automated system designed to detect and address various two-wheeler offenses, encompassing violations such as failure to wear a helmet, riding with a phone, triple riding, wheeling, and illegal parking. To achieve precise violation identification and tracking, the authors advocate for the implementation of a custom-trained YOLO-v4 + DeepSORT algorithm. Additionally, they propose the integration of Tesseract for efficient number plate detection and extraction, enhancing the system's capabilities for comprehensive enforcement. This holistic framework signifies a robust attempt to tackle a spectrum of traffic infractions, showcasing the synergy of advanced algorithms to ensure a more effective and accurate monitoring of road safety regulations. [14]

Presenting an automated solution, the paper introduces a method for monitoring traffic restrictions related to the number of passengers on a two-wheeler. Utilizing the YOLO-v3 model for object identification and implementing a depth estimate approach, the system assesses the distance between individuals on the two-wheeler to determine compliance with safety regulations. Despite the effectiveness of these approaches, the study highlights a significant drawback—their accuracy diminishes considerably in low-light conditions. This limitation underscores the challenges associated with real-world implementation and necessitates further exploration to enhance the system's robustness under varied lighting scenarios for a more reliable traffic monitoring solution. [15]

Chapter 3: Methodology

3.1 Basic Method

The methodology for Two-Wheeler Traffic Rule Violation Detection, which includes detecting if all riders are wearing helmets or not, whether the rider is driving in the correct direction or not, and whether there are more than two riders or not. Then it extract license plate numbers of the two-wheelers violating these rules, may be broken down into five crucial steps:

1. **Data Acquisition:** There are three datasets. First one for detecting two-wheelers, helmets, and license plates, with a total of 961 images; second one for detecting human faces, with a total of 3283 images; and third one for detecting front and rear of two-wheeler, with a total of 193 images.

2. **Model Training:** There are three models, one for each dataset, and each model is trained on Roboflow cloud.

3. **Model Implementation:** The trained models are called using the Roboflow API.

4. **Violation Detection:** In the violation detection process, the Roboflow Model APIs are initially initialized, followed by the setup of video processing. For each frame in the video, motorcycles are detected, and individual assessments are made for potential violations. Each motorcycle undergoes a series of checks, determining rear-facing orientation, assessing face-helmet overlap, and calculating face count. If violations such as Wrong Lane, No Helmet, or Triple Riding are detected, corresponding flags are set.

5. **License Plate Number Extraction:** If any violation flag is True, and a license plate is identified, the OCR.Space API is used to extract the license plate number, saving both the plate image and number.

3.2 Process Flow

I. **Initialize each Roboflow Model with API**

II. **Initialize Video Processing**

III. **Scan Every 180th frame**

IV. **Motorcycle Detection:**

A. Detects all two-wheelers/motorcycles in a frame.

V. **Bounding Box Extraction:**

A. For each detected motorcycle, it extracts its bounding box.

VI. **Orientation Check:**

A. Determines if the motorcycle is front-facing or rear-facing.

B. Flags a "Wrong Lane Violation" if the motorcycle is rear-facing.

VII. **Face and Helmet Detection:**

A. Detects faces and helmets within the cropped image.

B. Counts the number of faces.

C. Reduces the face count if the detected face and helmet areas overlap by more than 60%.

VIII. **No Helmet Violation:**

A. Detects helmets again and counts them.

B. Flags a "No Helmet Violation" if no helmets are detected or if the number of faces is greater than 1.

IX. **Triple Riding Violation:**

A. Sums up the final counts of helmets and faces.

B. Flags a "Triple Riding Violation" if the sum is greater than 2.

X. **License Plate Detection**

A. If any violation is detected, it captures the license plate using the OCR.Space API.

XI. **Saving Violation Data:**

A. Saves the violated motorcycle image along with its license plate image and text.

B. Records the list of violations for each images

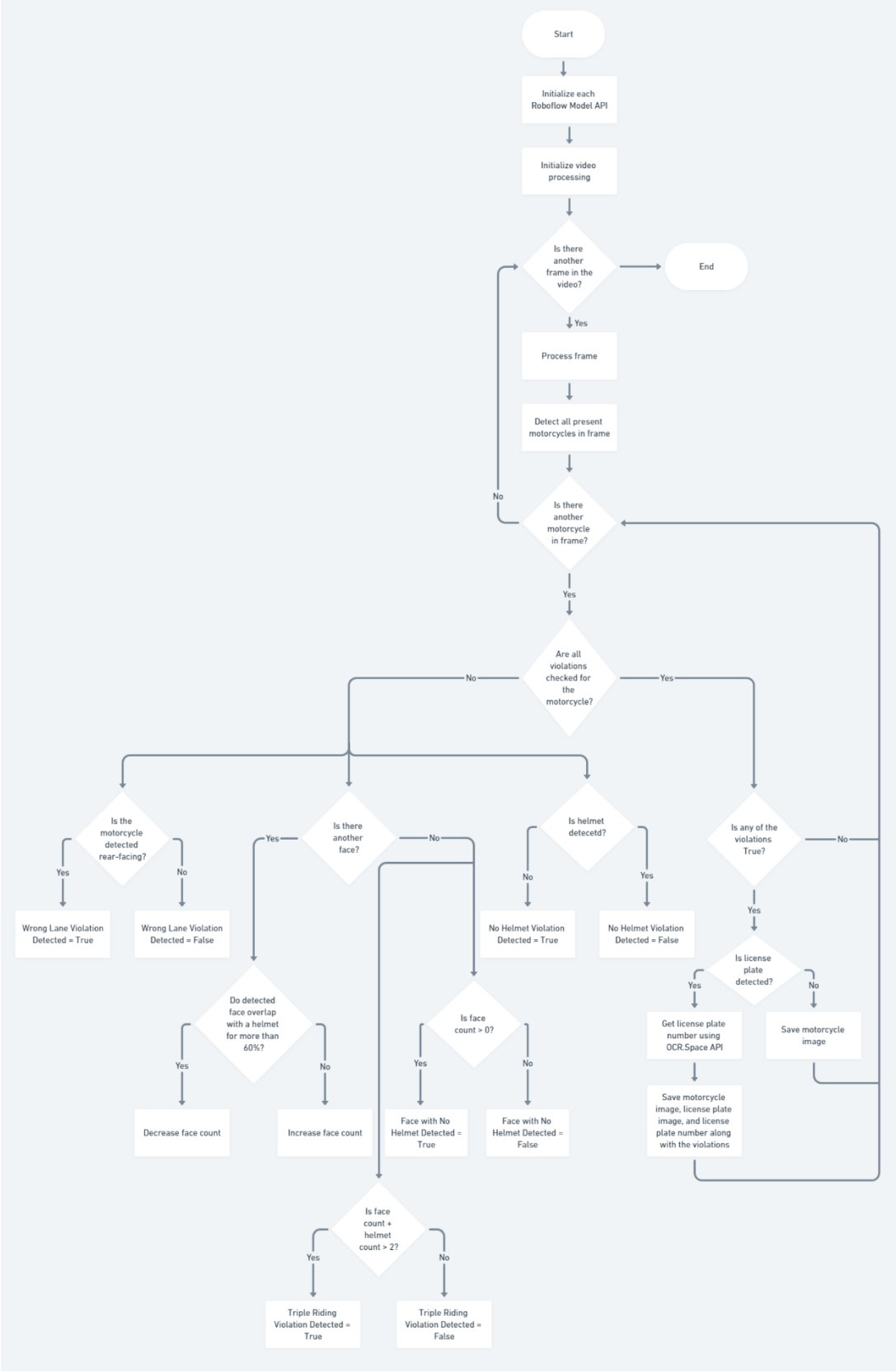
Start
Initialize each Roboflow Model API
Initialize video processing
Is there another frame in the video?
End
Yes
Process frame
Detect all present motorcycles in frame
No
Is there another motorcycle in frame?
Yes
Are all violations checked for the motorcycle?
No
Yes
Is the motorcycle detected rear-facing?
Yes
No
Wrong Lane Violation Detected = True
Wrong Lane Violation Detected = False
Is there another face?
Yes
No
Do detected face overlap with a helmet for more than 60%?
Yes
No
Decrease face count
Increase face count
Is face count > 0?
Yes
No
Face with No Helmet Detected = True
Face with No Helmet Detected = False
Is helmet detecetd?
No
Yes
No Helmet Violation Detected = True
No Helmet Violation Detected = False
Is any of the violations True?
No
Yes
Is license plate detected?
Yes
No
Get license plate number using OCR.Space API
Save motorcycle image
Save motorcycle image, license plate image, and license plate number along with the violations
Is face count + helmet count > 2?
Yes
No
Triple Riding Violation Detected = True
Triple Riding Violation Detected = False

Figure 3.1.1. Process Flow

Chapter 4: Implementation and Results

4.1 Code Breakdown

1. **Import Necessary Libraries**

```
import os

from PIL import Image, ImageDraw

import cv2

from roboflow import Roboflow

from datetime import datetime, timezone, timedelta

import requests

import json

import re

from tqdm import tqdm

from decouple import config
```

2. **Define OCR function**

```
def ocr_space_file(filename, overlay, api_key, language):

    payload = {

                'isOverlayRequired': overlay,

                'apikey': api_key,

                'language': language,

                'OCREngine': 2,

            }

    with open(filename, 'rb') as f:
```

```
        r = requests.post('https://api.ocr.space/parse/image',
                          files={filename: f},
                          data=payload,
                          )
    data = json.loads(r.content.decode())
    lines = data["ParsedResults"][0]["TextOverlay"]["Lines"]
    lpnum = "".join(line["LineText"] for line in lines)
    lpnum = re.sub(r'[^a-zA-Z0-9]', '', lpnum)
    return lpnum
```

3. **Define bounding box drawing function**

```
def draw_detections(p1, p2, p3, img):
    class_colors = {
        'helmet': 'blue',
        'motorcyclist': 'green',
        'license_plate': 'red',
        'face': 'darkmagenta',
        'front': 'darkgoldenrod',
        'rear': 'darkorchid'
    }
    draw = ImageDraw.Draw(img)
    preds = {'predictions': p1['predictions'] + p2['predictions'] +
p3['predictions']}
    for prediction in preds['predictions']:
        x, y, width, height = (
```

```
            prediction['x'],
            prediction['y'],
            prediction['width'],
            prediction['height']
        )
        x1 = x - width / 2
        y1 = y - height / 2
        x2 = x + width / 2
        y2 = y + height / 2
        class_name = prediction['class']
        confidence = prediction['confidence']
        label_color = class_colors.get(class_name, 'black')
        if class_name=='motorcyclist':
            draw.rectangle([x1, y1, x2, y1+14], fill=label_color)
            label_position = (x1 + 5, y1 + 2)
        else:
            draw.rectangle([x1, y1-14, x2, y1], fill=label_color)
            label_position = (x1 + 5, y1-12)
             draw.rectangle([x1,  y1,  x2,  y2],  outline=label_color,
width=2)
        label = f"{class_name} ({confidence:.2f})"
        draw.text(label_position,label, fill='white')
    return img
```

4. **Initialize Roboflow API and Models**

```
# Roboflow API keys

roboflow_api_key = config("ROBOFLOW_API_KEY")

# Initialize Roboflow instances

rf = Roboflow(api_key=roboflow_api_key)

p1 = rf.workspace().project("helmet-detection-project")

m1 = p1.version(13).model

p2 = rf.workspace().project("face-detection-mik1i")

m2 = p2.version(21).model

p3 = rf.workspace().project("two-wheeler-lane-detection")

m3 = p3.version(3).model
```

5. **Initialize Video Processing**

```
video_path = 'input.mp4'

cap = cv2.VideoCapture(video_path)

# Get video details

frame_width = int(cap.get(3))

frame_height = int(cap.get(4))

fps = cap.get(5)

total_frames = int(cap.get(7))

# Violate Date folder

current_date                                                    =
datetime.now(timezone.utc).astimezone(timezone(timedelta(hours=5,
minutes=30))).strftime("%d-%m-%Y")

folder_path                =                os.path.join(os.getcwd(),
f"Violations/{current_date}")
```

```
os.makedirs(folder_path, exist_ok=True)
```

6. **Perform detection on every 180th frame**

```
for frame_number in tqdm(range(0, total_frames, 180), desc="Processing frames", unit="frames"):
    ret, frame = cap.read()
    if not ret:
        break
            pil_frame = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    image_path = "temp_frame.jpg"
    pil_frame.save(image_path)
    r1 = m1.predict(image_path, confidence=40, overlap=40)
    pred1 = r1.json()['predictions']
    for pr1 in pred1:
        helmet_detected = False
        face_detected = False
        rear_detected = False
        more_than_two_detected = False
        num_faces_detected = 0
        num_helmets_detected = 0
        if pr1['class'] == 'motorcyclist':
            motorcyclist_x, motorcyclist_y, motorcyclist_width, motorcyclist_height = pr1['x'], pr1['y'], pr1['width'], pr1['height']
```

```
            motorcyclist_x1, motorcyclist_y1 = int(motorcyclist_x -
motorcyclist_width / 2), int(motorcyclist_y - motorcyclist_height /
2)

            motorcyclist_x2, motorcyclist_y2 = int(motorcyclist_x +
motorcyclist_width / 2), int(motorcyclist_y + motorcyclist_height /
2)

              motorcyclist_image = pil_frame.crop((motorcyclist_x1,
motorcyclist_y1, motorcyclist_x2, motorcyclist_y2))

            motorcyclist_image.save("temp_motorcyclist_image.jpg")

            # Lane check

                    r3 = m3.predict("temp_motorcyclist_image.jpg",
confidence=0, overlap=10)

            lane = r3.json()

            if lane['predictions']:

                  max_conf = max(lane['predictions'], key=lambda x:
x['confidence'])

                lane['predictions'] = [max_conf]

            pred3 = lane['predictions']

            for lane_prediction in pred3:

                if lane_prediction['class'] == 'rear':

                          rear_x, rear_y, rear_width, rear_height =
lane_prediction['x'],                          lane_prediction['y'],
lane_prediction['width'], lane_prediction['height']



                     if motorcyclist_x1 < rear_x < motorcyclist_x2
and motorcyclist_y1 < rear_y < motorcyclist_y2:

                      rear_detected = True
```

```
                        break

            # Face detected

                    r2  =  m2.predict("temp_motorcyclist_image.jpg",
confidence=40, overlap=30)

            pred2 = r2.json()['predictions']

            for face_prediction in pred2:

                if face_prediction['class'] == 'face':

                        face_x, face_y, face_width, face_height =
face_prediction['x'],                          face_prediction['y'],
face_prediction['width'], face_prediction['height']



                    if motorcyclist_x1 < face_x < motorcyclist_x2
and motorcyclist_y1 < face_y < motorcyclist_y2:

                        num_faces_detected += 1

                            # Avoid detecting helmet and face in same
area and calculating number of people incorrectly

                        for helmet_prediction in pred1:

                                    if helmet_prediction['class'] ==
'helmet':

                                    helmet_x, helmet_y, helmet_width,
helmet_height   =   helmet_prediction['x'],   helmet_prediction['y'],
helmet_prediction['width'], helmet_prediction['height']



                                face_x1 = face_x - face_width / 2

                                face_y1 = face_y - face_height / 2

                                face_x2 = face_x + face_width / 2
```

```
                                face_y2 = face_y + face_height / 2

                                 helmet_x1 = helmet_x - helmet_width
/ 2

                                helmet_y1 = helmet_y - helmet_height
/ 2

                                 helmet_x2 = helmet_x + helmet_width
/ 2

                                helmet_y2 = helmet_y + helmet_height
/ 2

                                overlap_x1 = max(face_x, helmet_x)

                                overlap_y1 = max(face_y, helmet_y)

                                          overlap_x2 = min(face_x +
face_width, helmet_x + helmet_width)

                                          overlap_y2 = min(face_y +
face_height, helmet_y + helmet_height)



                                 overlap_width = max(0, overlap_x2 -
overlap_x1)

                                overlap_height = max(0, overlap_y2 -
overlap_y1)



                                     overlap_area = overlap_width *
overlap_height



                                face_area = face_width * face_height



                                if overlap_area / face_area > 0.6:
```

```
                                    num_faces_detected -= 1
                                    break
            if num_faces_detected > 0:
                face_detected = True
            # Helmet check
            for helmet_prediction in pred1:
                if helmet_prediction['class'] == 'helmet':
                    helmet_x, helmet_y, helmet_width, helmet_height
=           helmet_prediction['x'],          helmet_prediction['y'],
helmet_prediction['width'], helmet_prediction['height']


                    if motorcyclist_x1 < helmet_x < motorcyclist_x2
and motorcyclist_y1 < helmet_y < motorcyclist_y2:
                        helmet_detected = True
                        num_helmets_detected += 1
            # More than two riding
            if num_faces_detected + num_helmets_detected > 2:
                more_than_two_detected = True
                    r4  =  m1.predict("temp_motorcyclist_image.jpg",
confidence=60, overlap=40)
                   colored_motorcycle  =  draw_detections(r4.json(),
r2.json(), lane, motorcyclist_image)
            # Violated license plate
            if not helmet_detected or face_detected or rear_detected
or more_than_two_detected:
```

```
                violation_names = []

                if not helmet_detected or face_detected:

                    violation_names.append('no_helmet')

                if rear_detected:

                    violation_names.append('wrong_lane')

                if more_than_two_detected:

                    violation_names.append('triple_riding')

                                                        timestamp    =
datetime.now(timezone.utc).astimezone(timezone(timedelta(hours=5,
minutes=30))).strftime("%d-%m-%Y %H %M %S")

                     image_name = ", ".join(violation_names) + f" -
{timestamp}"

                lp_detected = False

                for pr11 in pred1:

                    if pr11['class'] == 'license_plate':

                                    license_plate_x, license_plate_y,
license_plate_width, license_plate_height = pr11['x'], pr11['y'],
pr11['width'], pr11['height']

                                if motorcyclist_x1 < license_plate_x <
motorcyclist_x2    and    motorcyclist_y1    <    license_plate_y    <
motorcyclist_y2:

                                    license_plate_x1, license_plate_y1 =
int(license_plate_x - license_plate_width / 2), int(license_plate_y
- license_plate_height / 2)

                                    license_plate_x2, license_plate_y2 =
int(license_plate_x + license_plate_width / 2), int(license_plate_y
+ license_plate_height / 2)
```

```
                                                    license_plate_image =
pil_frame.crop((license_plate_x1,                        license_plate_y1,
license_plate_x2, license_plate_y2))

                                license_plate_image.save("temp_lp.jpg")

                                                                    lpnum  =
ocr_space_file(filename="temp_lp.jpg",                    overlay=False,
api_key=config("OCR_SPACE_API"), language='eng')

                                if lpnum.strip():

                                            image_name = lpnum + " - " +
image_name

                                else:

                                    image_name = image_name

                                                        image_folder_path  =
os.path.join(folder_path, image_name)

                                           os.makedirs(image_folder_path,
exist_ok=True)

                                        violated_motorcycle_image_path =
os.path.join(image_folder_path, f"{lpnum} - motorcyclist.jpg")


colored_motorcycle.save(violated_motorcycle_image_path)

                                    violated_motorcycle_lp_image_path =
os.path.join(image_folder_path, f"{lpnum} - license_plate.jpg")


license_plate_image.save(violated_motorcycle_lp_image_path)

                                                               lp_text_path  =
os.path.join(image_folder_path,                 f"{lpnum}                  -
license_plate_number.txt")
```

```
                            with open(lp_text_path, 'w') as file:
                                file.write(f"Violated License Plate
Number - {lpnum}")
                            lp_detected = True

                            if os.path.exists("temp_lp.jpg"):
                                os.remove("temp_lp.jpg")
                            break

                if not lp_detected:
                    image_folder_path = os.path.join(folder_path,
image_name)
                    os.makedirs(image_folder_path, exist_ok=True)
                                violated_motorcycle_image_path =
os.path.join(image_folder_path,                f"motorcyclist.jpg")
colored_motorcycle.save(violated_motorcycle_image_path)
if os.path.exists("temp_motorcyclist_image.jpg"):
    os.remove("temp_motorcyclist_image.jpg")
cap.release()
print("Video processing completed.")
```

4.2 Results

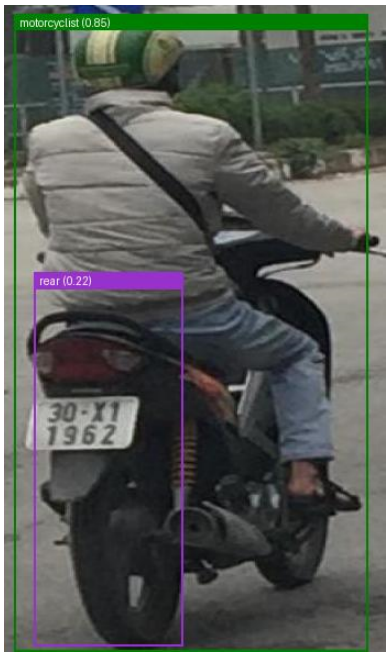

Figure 4.2.1. Saved Violated Two Wheeler Image

Figure 4.2.2. Saved Violated Two Wheeler License Plate Image

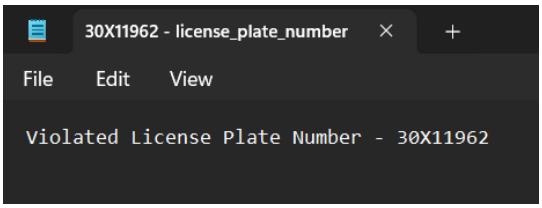

Figure 4.2.3. Saved Violated Two Wheeler License Plate Number

Chapter 4: Conclusion

The Two Wheeler Traffic Rule Violation Detection project presented here demonstrates a comprehensive and efficient approach to monitoring and identifying traffic violations related to two-wheelers. The integration of multiple computer vision models, each specialized in detecting specific elements such as two-wheelers, helmets, license plates. faces, and front/rear side of two-wheeler, showcases a sophisticated system for analyzing video frames. The project efficiently processes video input, leveraging the capabilities of Roboflow's models, and provides detailed information on detected violations.

Furthermore, the project goes beyond mere detection by implementing additional checks, such as verifying the correct usage of lanes, identifying instances of triple riding, and ensuring accurate license plate recognition. The utilization of OCR (Optical Character Recognition) for license plate extraction and subsequent validation enhances the code's versatility and applicability. The generated output includes not only visual evidence of violations but also text files containing the violated license plate numbers, providing a comprehensive record for further analysis.

Overall, this project represents a commendable effort in leveraging machine learning for traffic rule enforcement, contributing to improved road safety and compliance. The modular and well-structured design, along with the integration of external APIs and libraries, underscores its potential for adaptation and expansion in real-world traffic monitoring systems.

References

[1] Pattanashetty, Vishal B., et al. "Traffic rules violation detection system." Information and Communication Technology for Competitive Strategies (ICTCS 2020) ICT: Applications and Social Interfaces. Springer Singapore, 2022.

[2] Franklin, Ruben J. "Traffic signal violation detection using artificial intelligence and deep learning." 2020 5th international conference on communication and electronics systems (ICCES). IEEE, 2020.

[3] Reddy, P. Srinivas, et al. "Traffic rules violation detection using machine learning techniques." 2021 6th International conference on communication and electronics systems (ICCES). IEEE, 2021.

[4] Tonge, Aniruddha, et al. "Traffic rules violation detection using deep learning." 2020 4th international conference on electronics, communication and aerospace technology (ICECA). IEEE, 2020.

[5] Arnob, Faed Ahmed, et al. "An intelligent traffic system for detecting lane based rule violation." 2019 International Conference on Advances in the Emerging Computing Technologies (AECT). IEEE, 2020.

[6] Şentaş, Ali, Seda Kul, and Ahmet Sayar. "Real-time traffic rules infringing determination over the video stream: wrong way and clearway violation detection." 2019 International Artificial Intelligence and Data Processing Symposium (IDAP). IEEE, 2019.

[7] Uy, Aaron Christian P., et al. "Machine vision for traffic violation detection system through genetic algorithm." 2015 International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM). IEEE, 2015.

[8] P. Doungmala and K. Klubsuwan, ‘‘Helmet wearing detection in Thailand using Haar like feature and circle Hough transform on image processing,’’ in Proc. IEEE Int. Conf. Comput. Inf. Technol. (CIT), Dec. 2016, pp. 611–614.

[9] J. Chiverton, ‘‘Helmet presence classification with motorcycle detection and tracking,’’ IET Intell. Transport Syst., vol. 6, no. 3, pp. 259–269, Sep. 2012.

[10] C. Vishnu, D. Singh, C. K. Mohan, and S. Babu, ‘‘Detection of motorcyclists without helmet in videos using convolutional neural network,’’ in Proc. Int. Joint Conf. Neural Netw. (IJCNN), May 2017, pp. 3036–3041.

[11] K. C. D. Raj, A. Chairat, V. Timtong, M. N. Dailey, and M. Ekpanyapong, ‘‘Helmet violation processing using deep learning,’’ in Proc. Int. Workshop Adv. Image Technol. (IWAIT), Jan. 2018, pp. 1–4.

[12] A. Soni and A. P. Singh, ‘‘Automatic motorcyclist helmet rule violation detection using tensorflow & Keras in OpenCV,’’ in Proc. IEEE Int. Students’ Conf. Electr. Electron. Comput. Sci. (SCEECS), Feb. 2020, pp. 1–5.

[13] N. C. Mallela and R. Volety, ‘‘Detection of the triple riding and speed violation on two-wheelers using deep learning algorithms,’’ Multimedia Tools Appl., vol. 80, no. 6, pp. 8175–8187, Mar. 2021.

[14] R. S. Charran and R. K. Dubey, ‘‘Two-wheeler vehicle traffic violations detection and automated ticketing for Indian road scenario,’’ IEEE Trans. Intell. Transp. Syst., vol. 23, no. 11, pp. 22002–22007, Nov. 2022.

[15] K. Lavingia, M. Vaja, P. Chaturvedi, and A. Lavingia, ‘‘Machine learning based approach for traffic rule violation detection,’’ in Proc. IEEE 7th Int. Conf. Recent Adv. Innov. Eng. (ICRAIE), vol. 7, Dec. 2022, pp. 244–249.

Appendix

```
import os
from PIL import Image, ImageDraw
import cv2
from roboflow import Roboflow
from datetime import datetime, timezone, timedelta
import requests
import json
import re
from tqdm import tqdm
from decouple import config


def ocr_space_file(filename, overlay, api_key, language):
    payload = {
                'isOverlayRequired': overlay,
                'apikey': api_key,
                'language': language,
                'OCREngine': 2,
            }
    with open(filename, 'rb') as f:
        r = requests.post('https://api.ocr.space/parse/image',
                          files={filename: f},
                          data=payload,
```

```
                    )
    data = json.loads(r.content.decode())

    lines = data["ParsedResults"][0]["TextOverlay"]["Lines"]

    lpnum = "".join(line["LineText"] for line in lines)
    lpnum = re.sub(r'[^a-zA-Z0-9]', '', lpnum)

    return lpnum

def draw_detections(p1, p2, p3, img):
    class_colors = {
        'helmet': 'blue',
        'motorcyclist': 'green',
        'license_plate': 'red',
        'face': 'darkmagenta',
        'front': 'darkgoldenrod',
        'rear': 'darkorchid'
    }

    draw = ImageDraw.Draw(img)

    preds = {'predictions': p1['predictions'] + p2['predictions'] + p3['predictions']}
```

```
for prediction in preds['predictions']:
    x, y, width, height = (
        prediction['x'],
        prediction['y'],
        prediction['width'],
        prediction['height']
    )

    x1 = x - width / 2
    y1 = y - height / 2
    x2 = x + width / 2
    y2 = y + height / 2

    class_name = prediction['class']
    confidence = prediction['confidence']

    label_color = class_colors.get(class_name, 'black')

    if class_name=='motorcyclist':
        draw.rectangle([x1, y1, x2, y1+14], fill=label_color)
        label_position = (x1 + 5, y1 + 2)
    else:
```

```
            draw.rectangle([x1, y1-14, x2, y1], fill=label_color)
            label_position = (x1 + 5, y1-12)

        draw.rectangle([x1, y1, x2, y2], outline=label_color, width=2)

        label = f"{class_name} ({confidence:.2f})"
        draw.text(label_position,label, fill='white')

    return img

# Roboflow API keys
roboflow_api_key = config("ROBOFLOW_API_KEY")

# Initialize Roboflow instances
rf = Roboflow(api_key=roboflow_api_key)

p1 = rf.workspace().project("helmet-detection-project")
m1 = p1.version(13).model

p2 = rf.workspace().project("face-detection-mik1i")
m2 = p2.version(21).model

p3 = rf.workspace().project("two-wheeler-lane-detection")
```

```
m3 = p3.version(3).model


video_path = 'input.mp4'
cap = cv2.VideoCapture(video_path)


# Get video details
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))
fps = cap.get(5)
total_frames = int(cap.get(7))


# Violate Date folder
current_date = datetime.now(timezone.utc).astimezone(timezone(timedelta(hours=5, minutes=30))).strftime("%d-%m-%Y")
folder_path = os.path.join(os.getcwd(), f"Violations/{current_date}")
os.makedirs(folder_path, exist_ok=True)


# Process every 60th frame
for frame_number in tqdm(range(0, total_frames, 180), desc="Processing frames", unit="frames"):
    ret, frame = cap.read()
    if not ret:
        break


    pil_frame = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
```

```
    image_path = "temp_frame.jpg"
    pil_frame.save(image_path)

    r1 = m1.predict(image_path, confidence=40, overlap=40)
    pred1 = r1.json()['predictions']

    for pr1 in pred1:
        helmet_detected = False
        face_detected = False
        rear_detected = False
        more_than_two_detected = False
        num_faces_detected = 0
        num_helmets_detected = 0

        if pr1['class'] == 'motorcyclist':
                motorcyclist_x, motorcyclist_y, motorcyclist_width, motorcyclist_height = pr1['x'], pr1['y'], pr1['width'], pr1['height']

                motorcyclist_x1, motorcyclist_y1 = int(motorcyclist_x - motorcyclist_width / 2), int(motorcyclist_y - motorcyclist_height / 2)
                motorcyclist_x2, motorcyclist_y2 = int(motorcyclist_x + motorcyclist_width / 2), int(motorcyclist_y + motorcyclist_height / 2)
```

```
            motorcyclist_image = pil_frame.crop((motorcyclist_x1, motorcyclist_y1, motorcyclist_x2, motorcyclist_y2))
        motorcyclist_image.save("temp_motorcyclist_image.jpg")


        # Lane check
        r3 = m3.predict("temp_motorcyclist_image.jpg", confidence=10, overlap=10)
        lane = r3.json()


        if lane['predictions']:
            max_conf = max(lane['predictions'], key=lambda x: x['confidence'])
            lane['predictions'] = [max_conf]


        pred3 = lane['predictions']


        for lane_prediction in pred3:
            if lane_prediction['class'] == 'rear':
                            rear_x, rear_y, rear_width, rear_height = lane_prediction['x'], lane_prediction['y'], lane_prediction['width'], lane_prediction['height']


                            if motorcyclist_x1 < rear_x < motorcyclist_x2 and motorcyclist_y1 < rear_y < motorcyclist_y2:
                    rear_detected = True
                    break
```

```
        # Face detected
        r2 = m2.predict("temp_motorcyclist_image.jpg", confidence=40, overlap=30)
        pred2 = r2.json()['predictions']

        for face_prediction in pred2:
            if face_prediction['class'] == 'face':
                face_x, face_y, face_width, face_height = face_prediction['x'], face_prediction['y'], face_prediction['width'], face_prediction['height']

                if motorcyclist_x1 < face_x < motorcyclist_x2 and motorcyclist_y1 < face_y < motorcyclist_y2:
                    num_faces_detected += 1

                    # Avoid detecting helmet and face in same area and calculating number of people incorrectly
                    for helmet_prediction in pred1:
                        if helmet_prediction['class'] == 'helmet':
                            helmet_x, helmet_y, helmet_width, helmet_height = helmet_prediction['x'], helmet_prediction['y'], helmet_prediction['width'], helmet_prediction['height']

                            face_x1 = face_x - face_width / 2
                            face_y1 = face_y - face_height / 2
                            face_x2 = face_x + face_width / 2
                            face_y2 = face_y + face_height / 2
```

```
                helmet_x1 = helmet_x - helmet_width / 2
                helmet_y1 = helmet_y - helmet_height / 2
                helmet_x2 = helmet_x + helmet_width / 2
                helmet_y2 = helmet_y + helmet_height / 2

                overlap_x1 = max(face_x, helmet_x)
                overlap_y1 = max(face_y, helmet_y)
                overlap_x2 = min(face_x + face_width, helmet_x + helmet_width)
                overlap_y2 = min(face_y + face_height, helmet_y + helmet_height)

                overlap_width = max(0, overlap_x2 - overlap_x1)
                overlap_height = max(0, overlap_y2 - overlap_y1)

                overlap_area = overlap_width * overlap_height

                face_area = face_width * face_height

                if overlap_area / face_area > 0.6:
                    num_faces_detected -= 1
                    break

if num_faces_detected > 0:
```

```
            face_detected = True


        # Helmet check
        for helmet_prediction in pred1:
            if helmet_prediction['class'] == 'helmet':
                helmet_x, helmet_y, helmet_width, helmet_height = helmet_prediction['x'], helmet_prediction['y'], helmet_prediction['width'], helmet_prediction['height']


                if motorcyclist_x1 < helmet_x < motorcyclist_x2 and motorcyclist_y1 < helmet_y < motorcyclist_y2:
                    helmet_detected = True
                    num_helmets_detected += 1


        # More than two riding
        if num_faces_detected + num_helmets_detected > 2:
            more_than_two_detected = True


        r4 = m1.predict("temp_motorcyclist_image.jpg", confidence=60, overlap=40)
        colored_motorcycle = draw_detections(r4.json(), r2.json(), lane, motorcyclist_image)


        # Violated license plate
        if not helmet_detected or face_detected or rear_detected or more_than_two_detected:


            violation_names = []
```

```
        if not helmet_detected or face_detected:
            violation_names.append('no_helmet')
        if rear_detected:
            violation_names.append('wrong_lane')
        if more_than_two_detected:
            violation_names.append('triple_riding')


                    timestamp = datetime.now(timezone.utc).astimezone(timezone(timedelta(hours=5, minutes=30))).strftime("%d-%m-%Y %H %M %S")
        image_name = ", ".join(violation_names) + f" - {timestamp}"
        lp_detected = False


        for pr11 in pred1:
            if pr11['class'] == 'license_plate':
                    license_plate_x, license_plate_y, license_plate_width, license_plate_height = pr11['x'], pr11['y'], pr11['width'], pr11['height']


                        if motorcyclist_x1 < license_plate_x < motorcyclist_x2 and motorcyclist_y1 < license_plate_y < motorcyclist_y2:
                        license_plate_x1, license_plate_y1 = int(license_plate_x - license_plate_width / 2), int(license_plate_y - license_plate_height / 2)
                        license_plate_x2, license_plate_y2 = int(license_plate_x + license_plate_width / 2), int(license_plate_y + license_plate_height / 2)
```

```
                    license_plate_image = pil_frame.crop((license_plate_x1, license_plate_y1, license_plate_x2, license_plate_y2))

                    license_plate_image.save("temp_lp.jpg")

                    lpnum = ocr_space_file(filename="temp_lp.jpg", overlay=False, api_key=config("OCR_SPACE_API"), language='eng')

                    if lpnum.strip():
                        image_name = lpnum + " - " + image_name
                    else:
                        image_name = image_name
                    image_folder_path = os.path.join(folder_path, image_name)
                    os.makedirs(image_folder_path, exist_ok=True)

                        violated_motorcycle_image_path = os.path.join(image_folder_path, f"{lpnum} - motorcyclist.jpg")
                    colored_motorcycle.save(violated_motorcycle_image_path)

                        violated_motorcycle_lp_image_path = os.path.join(image_folder_path, f"{lpnum} - license_plate.jpg")
                    license_plate_image.save(violated_motorcycle_lp_image_path)

                    lp_text_path = os.path.join(image_folder_path, f"{lpnum} - license_plate_number.txt")
                    with open(lp_text_path, 'w') as file:
```

```
                    file.write(f"Violated License Plate Number - {lpnum}")

                lp_detected = True

                if os.path.exists("temp_lp.jpg"):
                    os.remove("temp_lp.jpg")
                break

        if not lp_detected:
            image_folder_path = os.path.join(folder_path, image_name)
            os.makedirs(image_folder_path, exist_ok=True)
            violated_motorcycle_image_path = os.path.join(image_folder_path, f"motorcyclist.jpg")

            colored_motorcycle.save(violated_motorcycle_image_path)

if os.path.exists("temp_motorcyclist_image.jpg"):
    os.remove("temp_motorcyclist_image.jpg")
cap.release()

print("Video processing completed.")
```