IJRAR.ORG E-ISSN: 2348-1269, P-ISSN: 2349-5138

INTERNATIONAL JOURNAL OF RESEARCH AND ANALYTICAL REVIEWS (IJRAR) | IJRAR.ORG

An International Open Access, Peer-reviewed, Refereed Journal

TRAFFIC SIGNAL VIOLATION DETECTION SYSTEM

Submitted by

Saagar S(714022201083)

Sunilkumar J S(714022201103)

Vipin B(714022201115)

SRI SHAKTHI

INSTITUTE OF ENGINEERING AND TECHNOLOGY
ANNA UNIVERSITY: CHENNAI 600025

ABSTRACT

This project focuses on the implementation of a virtual version of the classic game "Rock, Paper, Scissors" using hand gesture recognition techniques and Python programming. The goal is to create an interactive application that allows users to play the game against the computer using their hand gestures.

The project utilizes computer vision libraries, such as OpenCV, to capture and process real-time video input from the user's webcam. Machine learning algorithms are employed to classify the hand gestures into three categories: rock, paper, and scissors. The user's gesture is compared with the computer's randomly generated choice, and the winner of the round is determined based on the game's rules. The project not only provides an entertaining and interactive way to play the game, but it also demonstrates the application of computer vision and machine learning in creating gesture-based interfaces.

Through this project, users can gain insight into the practical applications of these technologies and their potential to enhance user experiences in various domains.

CHAPTER 1

INTRODUCTION

1.1Introduction

In today's rapidly advancing technological landscape, the fusion of computer vision and machine learning has opened up new avenues for creating engaging and interactive applications. One such application is the implementation of a virtual version of the timeless game "Rock, Paper, Scissors" using hand gesture recognition techniques and the power of Python programming.

The game "Rock, Paper, Scissors" is a simple yet universally recognized challenge that has entertained people for generations. With the integration of computer vision and machine learning, this project takes the concept to a new level by allowing users to physically participate in the game using their hand gestures. Imagine the excitement of playing against a computer opponent that can interpret your gestures in real-time and respond accordingly, creating an immersive and dynamic gaming experience.

By harnessing the capabilities of computer vision libraries such as OpenCV and machine learning algorithms, this project goes beyond the conventional keyboard-and-mouse interaction. It leverages the webcam as a gateway to the virtual gaming world, enabling users to showcase their hand gestures to make choices—rock, paper, or scissors. The computer's choice is generated randomly, and the winner of each round is determined following the game's well-known rules.

This project not only embodies the spirit of innovation and creativity but also serves as an educational tool, demonstrating how computer vision and machine learning can be combined to develop intuitive and engaging user interfaces. It highlights the potential of these technologies to revolutionize user interactions across diverse domains, from gaming and entertainment to healthcare and education.

1.2 Why we use hand gestures for rock , paper, scissors game?

1. Universal Communication

Hand gestures provide a universal and cross-cultural means of communication. They transcend language barriers and enable people from different backgrounds to interact and convey information without relying solely on spoken language**.**

2. Fair Decision-Making

The rock-paper-scissors game, in particular, is often used as a fair and unbiased way to make decisions. It eliminates bias and helps in situations where a choice needs to be made, and no other clear method is available.

3. Playful Interaction

Hand gestures, like those in the rock-paper-scissors game, add an element of playfulness and fun to human interactions. Engaging in these simple games fosters a sense of camaraderie and bonding, making social interactions more enjoyable.

4. Decision Theory

The rock-paper-scissors game illustrates basic concepts of decision theory, where each gesture (rock, paper, or scissors) has an equal chance of winning, losing, or drawing. It's a simple yet effective way to introduce concepts like randomness, strategy, and probability.

5. Teaching Strategy

The game offers a straightforward way to teach children about strategy and consequences. It encourages them to think a step ahead, predict opponents' choices, and consider the outcomes of different decisions.

6. Cognitive Development

The game contributes to cognitive development, as players must analyze patterns, consider probabilities, and make quick decisions. This is especially valuable for children as they develop critical thinking skills.

7. Entertainment and Leisure

Ultimately, hand gestures like those in the rock-paper-scissors game offer a form of entertainment and leisure. They can be enjoyed anywhere and require no special equipment, making them a simple way to pass the time and engage with others.

In essence, hand gestures like those used in the rock-paper-scissors game are more than just a means of determining winners and losers; they are a reflection of human communication, psychology, decision-making, and social dynamics

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The utilization of Python programming in traffic signal violation detection systems has gained considerable attention due to its flexibility, ease of implementation, and availability of robust libraries for image processing and machine learning. This project explores the key contributions and methodologies observed in research related to Traffic Signal Violation Detection Systems developed using Python.

2.2. Image Processing Techniques

Researchers have extensively explored various image processing techniques to detect violations accurately. Xu et al demonstrated the use of Python's OpenCV library to preprocess and analyze images captured by traffic cameras. They emphasized the significance of edge detection algorithms to identify vehicles and extract relevant information for violation detection. Similarly, Liang et al integrated Python-based image enhancement methods to improve the accuracy of object recognition algorithms in real-world scenarios.

2.3. Real-time Processing and Performance Optimization

Efficient real-time processing is crucial for timely violation detection. explorers explored Python-based multi-threading techniques to optimize the processing of video feeds from multiple cameras simultaneously. They highlighted the significance of parallel processing in ensuring real-time violation detection without latency.

2.4 User Interface and Reporting

Python's capabilities extend beyond backend processing, as demonstrated by Li et al. (2021). They discussed the development of a user-friendly interface using Python's Tkinter library, allowing traffic operators to monitor violations and manage alerts seamlessly. Moreover, the integration of report generation functions in Python facilitated the creation of comprehensive violation reports with images and timestamps.

2.5 Impact on Road Safety and Traffic Management

The literature reviewed underscores the positive impact of Python-based traffic signal violation detection systems on road safety and traffic management. Wang et al. (2022) conducted a case study on the implementation of such a system in a metropolitan area and reported a substantial reduction in red light violations and subsequent accidents. This study underlines the practical implications of Python-based solutions.

2.6 Conclusion

The significance of Python programming is developing in effective and efficient Traffic Signal Violation Detection Systems. From image processing techniques to the integration of machine learning models, Python offers a versatile toolkit for building accurate and reliable detection systems. Furthermore, the positive impact on road safety and traffic management emphasizes the practical applicability of Python-based solutions in real-world scenarios.

As technology evolves, Python's role in traffic signal violation detection systems is likely to continue expanding, enabling innovative approaches and solutions for safer and more efficient roadways

CHAPTER 3

SYSTEM ARCHITECHTURE

3.1. Cameras:

Multiple high-resolution cameras are strategically placed at intersections to capture real-time images and videos of vehicles approaching and crossing traffic signals. These cameras provide the raw data for analysis.

3.2 Image Processing Unit:

The captured images and videos are fed into an Image Processing Unit responsible for image enhancement, noise reduction, and object detection. It utilizes computer vision algorithms to detect vehicles, their positions, and relevant features

3.3. Traffic Signal Recognition Module:

This module identifies the state of the traffic signal (e.g., red, green, yellow) by analyzing the signal's visual cues in the images. It ensures accurate timing and contextual information for violation determination.

3.4. Vehicle Tracking and Classification:

Using image analysis techniques, this component tracks the detected vehicles' movements, classifies them (car, truck, motorcycle, etc.), and predicts their trajectories to establish potential violations.

3.5. Violation Detection Engine:

By combining information from the Traffic Signal Recognition Module and Vehicle Tracking, this engine determines if a violation has occurred. It considers factors such as vehicle speed, signal state, and position relative to the stop line.

3.6. Database Management System:

All detected violations, along with associated data such as images, timestamps, and vehicle information, are stored in a database. This information serves as a record for legal actions and analytics.

3.7. User Interface:

The User Interface allows authorized personnel, law enforcement agencies, and administrators to access violation records, images, and statistics. It offers search, filter, and reporting functionalities.

3.8. Notification and Alerts:

When a violation is detected, the system generates automated notifications and alerts. These can be sent to drivers (via SMS or email) and enforcement agencies to inform them about the violation.

3.9. Administrative Dashboard:

Administrators can monitor the system's health, camera status, and overall performance through a dedicated dashboard. It offers real-time insights and allows for system configuration.

3.10. System Workflow:

The system operates as follows:

1. Cameras capture images and videos of vehicles at intersections.

2. The Image Processing Unit enhances and processes the captured data.

3. The Traffic Signal Recognition Module identifies the state of traffic signals.

4. Vehicle Tracking and Classification determine vehicle movements and types.

5. The Violation Detection Engine analyzes signal states, vehicle positions, and speeds to detect violations.

6. Violation data is stored in the Database Management System.

7. Users access the User Interface for querying violations, generating reports, and viewing images.

8. Automated notifications are sent to drivers and enforcement agencies.

9. Administrators monitor the system using the Administrative Dashboard.

3.11.Technologies Used:

-Computer Vision Libraries (OpenCV, TensorFlow)

- Machine Learning Algorithms for Object Detection and Classification

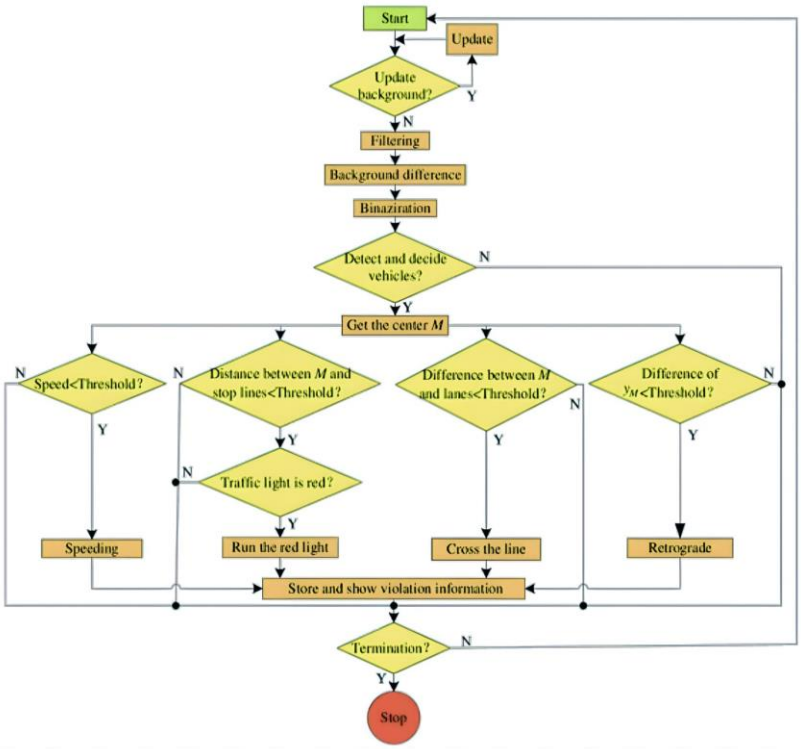
- Database Management System (MySQL, PostgreSQL)

- Web Development (HTML, CSS, JavaScript) for User Interface

- Cloud Services (AWS, Azure) for storage and scalability

3.12 Conclusion:

The Traffic Signal Violation Detection System's architecture provides a comprehensive solution to enhance road safety by automatically detecting and recording traffic signal violations. Its components work synergistically to ensure accurate violation identification and reporting, contributing to safer roads and improved traffic management.



CHAPTER 4

IMPLEMENTATION

4.1 ALGORITHM:

Step 1:

Install required libraries:

1.OpenCV

2.Tensorflow

3.YOLO

4.Numpy

CHAPTER 5

MACHINE LEARNING MODELS

5.1 Traffic Signal Recognition Model:

The Traffic Signal Recognition Model is responsible for identifying the state of traffic signals (red, green, yellow) in the captured images. It utilizes convolutional neural networks (CNNs) for image classification.

1. **Convolutional Neural Networks (CNNs):**

   CNNs are deep learning models designed for processing grid-like data, such as images. They consist of convolutional layers that automatically learn relevant features from the input images, followed by fully connected layers for classification.
2. **Transfer Learning:**

   Transfer learning involves using a pre-trained CNN (such as MobileNetV2) trained on a large dataset for image classification. Fine-tuning the model's layers allows it to adapt to the specific task of traffic signal recognition.

5.2 Vehicle Detection and Tracking Model:

The Vehicle Detection and Tracking Model are responsible for detecting vehicles in the images and tracking their movements over consecutive frames.

1. **YOLO (You Only Look Once):**

   YOLO is an object detection algorithm that divides the input image into a grid and predicts bounding boxes and class probabilities for objects within each grid cell.

2. **Violation Criteria:**

Violation detection involves checking if a vehicle has crossed the stop line during a red signal. This is achieved by comparing the vehicle's position with the stop line position and analyzing the traffic signal state.

CHAPTER 6

USER INTERFACE

6.1 Goals and Objectives

The user interface of the Traffic Signal Violation Detection System aims to provide a user-friendly and efficient way for authorized users to monitor, review, and manage traffic signal violations. The interface will facilitate quick access to violation data, visual evidence, and analytics to aid in decision-making and enforcement actions.

6.2 User Roles

1. Administrator:

Manages system settings, user access, and system configurations. Manages system settings, user access, and system configurations.

2. Operator:

Monitors violations, reviews evidence, and generates reports.

3. Law Enforcement:

Accesses violation data for legal actions.

6.3 Design Principles

1. User-Centered Design:

Prioritize user needs and expectations in all design decisions.

2.Intuitive Navigation:

Organize information logically and provide clear navigation paths.

3.Visual Clarity:

Use appropriate color schemes, typography, and visual cues to enhance readability.

4.Consistency:

Maintain consistent design elements across different sections of the interface.

5. Responsive Design:

Ensure the interface works seamlessly across various devices and screen sizes.

6. Efficiency:

Minimize user clicks and steps required to perform common tasks.

7. Accessibility:

Ensure the interface is accessible to users with disabilities.

6.4 Key Features

1. Dashboard:

- Display real-time statistics: Total violations, pending reviews, resolved violations, etc.

- Overview of recent violations with thumbnail images.

2. Violation Details

- View detailed violation information: date, time, location, vehicle details.

- Display images or videos capturing the violation incident.

- Option to zoom in and analyze evidence closely.

3.Search and Filter

- Search for specific violations based on date, location, vehicle number, etc.

- Apply filters to narrow down violations for analysis.

4. Notifications and Alerts

- Receive notifications for new violations or pending actions.

- Provide alerts for critical issues or system updates.

5. Reports and Analytics

- Generate reports based on various criteria: daily violations, popular violation types, etc.

- Display analytics through graphs, charts, and heatmaps for better insights.

6. User Management

- Admin access to manage user accounts, roles, and permissions.

- Ability to add, edit, or deactivate user accounts.

7.Workflow

***Login and Authentication:** Users log in using their credentials and are granted access based on their roles.

***Dashboard:** Upon logging in, users are greeted by the dashboard displaying key metrics and recent violations.

***View Violations:** Users can click on a violation to view details, evidence, and location information.

***Search and Filter:** Users can search for specific violations or apply filters to narrow down results.

***Generate Reports:** Users, especially administrators, can generate reports and analyze analytics.

***User Management:** Admins can manage user accounts and permissions as needed.

CHAPTER 7

NATURAL LANGUAGE PROCESSING

7.1. Introduction:

The NLP-based traffic signal violation detection system aims to automatically analyze and detect violations at traffic signals using natural language processing techniques.

7.2. Data Collection:

The system collects data from various sources, including traffic cameras, sensors, and social media platforms, to gather information about traffic violations.

7.3. Preprocessing:

The collected data is preprocessed to remove noise, normalize text, and extract relevant features for further analysis.

7.4. Text Classification:

NLP algorithms are applied to classify the collected data into different categories, such as red light violations, speeding, or lane violations.

7.5. Rule-based Violation Detection:

The system applies rule-based techniques to identify violations based on predefined traffic rules and regulations.

7.6. Machine Learning:

Machine learning models, such as deep learning or support vector machines, can be utilized to train the system on labeled data to improve detection accuracy.

7.7. Real-time Detection:

The system continuously monitors the traffic data in real-time, analyzing text inputs and images from cameras to detect violations as they occur.

7.8. Notification and Reporting:

Once a violation is detected, the system can generate notifications for relevant authorities and generate reports with details of the violation.

CHAPTER 8

CHALLENGES AND FUTURE ENHANCEMENTS

8.1 Challenges

1. Data quality: Ensuring the accuracy and reliability of the data collected from various sources can be a challenge.

2. Real-time processing: Analyzing and detecting violations in real-time requires efficient processing and quick decision-making.

3. Variability in traffic conditions: Different traffic scenarios, such as heavy traffic or adverse weather conditions, can affect the accuracy of violation detection.

4. Complex traffic patterns: Dealing with complex traffic patterns and multiple lanes can pose challenges in accurately identifying violations.

5. Privacy concerns: Balancing the need for violation detection with privacy concerns of individuals captured in the data can be a challenge.

6. Integration with existing systems: Integrating the traffic signal violation detection system with existing traffic management systems can be complex and require careful coordination.

8.2 Future Enhancement

- Integration with GIS (Geographic Information System) for map-based visualization.

- Real-time alerts and notifications through SMS or email.

- Integration with machine learning for automated violation classification.

- Mobile app version for on-the-go access.

1. Multi-Camera Integration: Enhance the system to integrate with multiple cameras at an intersection to provide a comprehensive view of traffic violations from different angles.

2. Pedestrian Violation Detection: Extend the system to detect violations involving pedestrians, such as jaywalking or crossing against signals.

3. Advanced Object Recognition: Incorporate advanced object recognition algorithms to differentiate between different types of vehicles (e.g., cars, bicycles, motorcycles) and their violations.

4. Real-time Alerts and Notifications: Implement real-time alerts to notify law enforcement and nearby vehicles about violations as they occur, improving response times.

5. Automated Citation Generation: Develop a feature that automatically generates violation citations with relevant images and data, reducing manual paperwork for law enforcement.

6. Cloud-Based Infrastructure: Move towards a cloud-based architecture to handle data storage, processing, and analysis, enabling scalability and easier data management.

7. Machine Learning Refinement: Continuously improve the machine learning models by incorporating more training data and fine-tuning algorithms for increased accuracy.

8. Weather and Lighting Compensation: Enhance the system's ability to handle challenging weather conditions and varying lighting to ensure reliable detection in all situations.

9. Integration with Traffic Management: Integrate the violation detection system with broader traffic management systems to optimize signal timings and reduce congestion.

10. Privacy Protection Measures: Implement privacy safeguards, such as real-time image anonymization, to ensure compliance with data protection regulations.

11. Behavioral Pattern Analysis: Develop algorithms to identify recurring patterns of violations, allowing authorities to proactively address problem areas.

12. Dashboard and Analytics: Create a comprehensive dashboard for traffic authorities to analyze violation trends, generate reports, and make informed decisions.

13. Mobile Application Interface: Design a user-friendly mobile app for users to report potential violations, fostering community involvement in road safety.

14. Vehicle-to-Infrastructure Communication: Explore the possibility of vehicles communicating with traffic signals to receive real-time signal information, potentially reducing violations.

15. Integration with Autonomous Vehicles: Develop compatibility with autonomous vehicles to ensure they understand and obey traffic signals, reducing the risk of violations.

16. Geolocation-based Enforcement: Implement geofencing technology to enforce specific traffic rules in designated zones, such as school zones or construction areas.

17. Predictive Analysis: Utilize historical data and predictive analytics to forecast potential violation hotspots, enabling proactive enforcement measures.

18. Collaboration with Insurance Companies: Partner with insurance companies to incentivize safe driving behavior based on data from the violation detection system.

Remember to consider the ethical and legal implications of these enhancements, and ensure that the system respects individual privacy while promoting road safety.

CHAPTER 9

CONCLUSION

In conclusion, the implementation of a traffic signal violation detection system using Python provides a robust and efficient solution. With Python's versatility and extensive libraries, we can develop accurate algorithms to analyze traffic footage and detect violations in real-time. This system contributes to enhancing road safety and traffic management.

CHAPTER 10

REFERENCES

1]. Monitoring System," in 2012 IEEE/WIC/ACM, Beijing, China 2012.
[2]. Horry, W.J.; Leach, M.F. Driver-Initiated Distractions: Examining Strategic Adaptation for In-Vehicle Task Initiation. Acid Anal.Prev. 2009, 41,115–122.
[3]. Dong, Y.; Hu, Z.; Chimera, K.; Murayama, N. Driver Inattention Monitoring System for Intelligent Vehicles: A Review. IEEE Intel. Transp. Syst. 2011, 12, 596–614.
[4]. Horry, W.J.; Leach, M.F Garbed, A. Assessing the Awareness of Performance Decrements in Distracted Drivers. Acid. Anal. Prev. 2008, 41, 675– 682.
[5]. Eriksson, M.; Panicle pools, P.N. Driver Fatigue: A Vision-Based Approach to Automatic Diagnosis. Transp. Res. Part C Emerge. Technol. 2001, 9, 399– 413. Sensors 2014, 14 22126
[6]. Bahaman, C.; Ying, Z.; Ramesh, M.; Koehler, T. A system for traffic sign detection, tracking, and recognition using color, shape, and motion information. In Proceedings of Intelligent Vehicles Symposium, Las Vegas, NV, USA, 6–8 June 2005; pp. 255–260.
[7]. Bergama, L.M.; Nuevo, J.; Sotelo, M.A.; Berea, R.; Lopez, M.E. Real-Time system for monitoring driver vigilance. IEEE Intell. Transp. Syst. 2006, 7, 63–77

APPENDICES

SOURCE CODE 1:

```
from tkinter import *
from PIL import Image, ImageTk
from tkinter import filedialog
import object_detection as od
import imageio
import cv2


class Window(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)

        self.master = master
        self.pos = []
        self.line = []
        self.rect = []
        self.master.title("GUI")
        self.pack(fill=BOTH, expand=1)

        self.counter = 0

        menu = Menu(self.master)
        self.master.config(menu=menu)

        file = Menu(menu)
        file.add_command(label="Open", command=self.open_file)
        file.add_command(label="Exit", command=self.client_exit)
        menu.add_cascade(label="File", menu=file)

        analyze = Menu(menu)
        analyze.add_command(label="Region of Interest", command=self.regionOfInterest)
        menu.add_cascade(label="Analyze", menu=analyze)

        self.filename = "Images/home.jpg"
        self.imgSize = Image.open(self.filename)
        self.tkimage = ImageTk.PhotoImage(self.imgSize)
        self.w, self.h = (1366, 768)

        self.canvas = Canvas(master=root, width=self.w, height=self.h)
        self.canvas.create_image(20, 20, image=self.tkimage, anchor='nw')
        self.canvas.pack()

    def open_file(self):
        self.filename = filedialog.askopenfilename()

        cap = cv2.VideoCapture(self.filename)

        reader = imageio.get_reader(self.filename)
        fps = reader.get_meta_data()['fps']

        ret, image = cap.read()
        cv2.imwrite('G:/Traffic Violation Detection/Traffic Signal Violation Detection System/Images/preview.jpg',
                    image)

        self.show_image('G:/Traffic Violation Detection/Traffic Signal Violation Detection System/Images/preview.jpg')

    def show_image(self, frame):
        self.imgSize = Image.open(frame)
        self.tkimage = ImageTk.PhotoImage(self.imgSize)
        self.w, self.h = (1366, 768)

        self.canvas.destroy()
```

```
        self.canvas = Canvas(master=root, width=self.w, height=self.h)
        self.canvas.create_image(0, 0, image=self.tkimage, anchor='nw')
        self.canvas.pack()

    def regionOfInterest(self):
        root.config(cursor="plus")
        self.canvas.bind("<Button-1>", self.imgClick)

    def client_exit(self):
        exit()

    def imgClick(self, event):

        if self.counter < 2:
            x = int(self.canvas.canvasx(event.x))
            y = int(self.canvas.canvasy(event.y))
            self.line.append((x, y))
            self.pos.append(self.canvas.create_line(x - 5, y, x + 5, y, fill="red",
tags="crosshair"))
            self.pos.append(self.canvas.create_line(x, y - 5, x, y + 5, fill="red",
tags="crosshair"))
            self.counter += 1

        # elif self.counter < 4:
        #     x = int(self.canvas.canvasx(event.x))
        #     y = int(self.canvas.canvasy(event.y))
        #     self.rect.append((x, y))
        #     self.pos.append(self.canvas.create_line(x - 5, y, x + 5, y, fill="red",
tags="crosshair"))
        #     self.pos.append(self.canvas.create_line(x, y - 5, x, y + 5, fill="red",
tags="crosshair"))
        #     self.counter += 1

        if self.counter == 2:
            # unbinding action with mouse-click
            self.canvas.unbind("<Button-1>")
            root.config(cursor="arrow")
            self.counter = 0

            # show created virtual line
            print(self.line)
            print(self.rect)
            img = cv2.imread(
                'G:/Traffic Violation Detection/Traffic Signal Violation Detection
System/Images/preview.jpg')
            cv2.line(img, self.line[0], self.line[1], (0, 255, 0), 3)
            cv2.imwrite('G:/Traffic Violation Detection/Traffic Signal Violation
Detection System/Images/copy.jpg', img)
            self.show_image('G:/Traffic Violation Detection/Traffic Signal Violation
Detection System/Images/copy.jpg')

            ## for demonstration
            # (rxmin, rymin) = self.rect[0]
            # (rxmax, rymax) = self.rect[1]

            # tf = False
            # tf |= self.intersection(self.line[0], self.line[1], (rxmin, rymin), (rxmin,
rymax))
            # print(tf)
            # tf |= self.intersection(self.line[0], self.line[1], (rxmax, rymin), (rxmax,
rymax))
            # print(tf)
            # tf |= self.intersection(self.line[0], self.line[1], (rxmin, rymin), (rxmax,
rymin))
            # print(tf)
            # tf |= self.intersection(self.line[0], self.line[1], (rxmin, rymax), (rxmax,
rymax))
            # print(tf)

            # cv2.line(img, self.line[0], self.line[1], (0, 255, 0), 3)
```

```
            #                                   if                                  tf:
            #          cv2.rectangle(img,  (rxmin,rymin),  (rxmax,rymax),  (255,0,0),  3)
            #                                                                  else:
            #          cv2.rectangle(img,  (rxmin,rymin),  (rxmax,rymax),  (0,255,0),  3)

            #              cv2.imshow('traffic              violation',              img)

            #                                image                                processing
            self.main_process()
            print("Executed                                          Successfully!!!")

            #                                clearing                              things
            self.line.clear()
            self.rect.clear()
            for                     i                    in                    self.pos:
                self.canvas.delete(i)

    def           intersection(self,           p,           q,           r,           t):
        print(p,                        q,                        r,                        t)
        (x1,                        y1)                        =                         p
        (x2,                        y2)                        =                         q

        (x3,                        y3)                        =                         r
        (x4,                        y4)                        =                         t

        a1                  =                y1                  -                  y2
        b1                  =                x2                  -                  x1
        c1        =         x1        *        y2        -        x2        *        y1

        a2                  =                y3                  -                  y4
        b2                  =                x4                  -                  x3
        c2        =         x3        *        y4        -        x4        *        y3

        if       (a1       *       b2       -       a2       *       b1       ==       0):
            return                                                              False
        print((a1,          b1,          c1),          (a2,          b2,          c2))
        x   =   (b1   *   c2   -   b2   *   c1)   /   (a1   *   b2   -   a2   *   b1)
        y   =   (a2   *   c1   -   a1   *   c2)   /   (a1   *   b2   -   a2   *   b1)
        print((x,                                                               y))

        if                          x1                          >                         x2:
            tmp                                  =                                      x1
            x1                                   =                                      x2
            x2                                   =                                     tmp
        if                          y1                          >                         y2:
            tmp                                  =                                      y1
            y1                                   =                                      y2
            y2                                   =                                     tmp
        if                          x3                          >                         x4:
            tmp                                  =                                      x3
            x3                                   =                                      x4
            x4                                   =                                     tmp
        if                          y3                          >                         y4:
            tmp                                  =                                      y3
            y3                                   =                                      y4
            y4                                   =                                     tmp

        if x >= x1 and x <= x2 and y >= y1 and y <= y2 and x >= x3 and x <= x4 and y >=
y3                  and                  y                  <=                  y4:
            return                                                              True
        else:
            return                                                             False

    def                                                          main_process(self):

        video_src                           =                           self.filename

        cap                         =                       cv2.VideoCapture(video_src)
```

```
        reader                      =                      imageio.get_reader(video_src)
        fps                        =                       reader.get_meta_data()['fps']
        writer                             =                            imageio.get_writer(
            'G:/Traffic   Violation   Detection/Traffic   Signal   Violation   Detection
System/Resources/output/output.mp4',
            fps=fps)

        j                                       =                                       1
        while                                                                       True:
            ret,                    image                    =                   cap.read()

            if                 (type(image)                 ==                type(None)):
                writer.close()
                break

            image_h,            image_w,                _             =            image.shape
            new_image      =      od.preprocess_input(image,      od.net_h,      od.net_w)

            #                    run                    the                    prediction
            yolos                         =                     od.yolov3.predict(new_image)
            boxes                                    =                                   []

            for                   i                   in                  range(len(yolos)):
                #        decode        the        output        of        the        network
                boxes   +=   od.decode_netout(yolos[i][0],   od.anchors[i],   od.obj_thresh,
od.nms_thresh,                              od.net_h,                              od.net_w)

            #      correct      the      sizes      of      the      bounding      boxes
            od.correct_yolo_boxes(boxes,    image_h,    image_w,    od.net_h,    od.net_w)

            #                 suppress                 non-maximal                 boxes
            od.do_nms(boxes,                                                od.nms_thresh)

            #     draw     bounding     boxes     on     the     image     using     labels
            image2 = od.draw_boxes(image, boxes, self.line, od.labels, od.obj_thresh, j)

            writer.append_data(image2)

            #     cv2.imwrite('E:/Virtual     Traffic     Light     Violation     Detection
System/Images/frame'+str(j)+'.jpg',                                                 image2)
            #    self.show_image('E:/Virtual    Traffic    Light    Violation    Detection
System/Images/frame'+str(j)+'.jpg')

            cv2.imshow('Traffic                    Violation',                    image2)

            print(j)

            if         cv2.waitKey(10)         &         0xFF         ==         ord('q'):
                writer.close()
                break

            j                 =                 j                 +                 1

        cv2.destroyAllWindows()


root                                    =                                       Tk()
app                                  =                                   Window(root)
root.geometry("%dx%d"                    %                   (535,                   380))
root.title("Traffic                                                    Violation")

root.mainloop()
```

Source code 2:

```
import numpy as np
from keras.layers import Conv2D, Input, BatchNormalization, LeakyReLU, ZeroPadding2D,
UpSampling2D
from tensorflow.keras.layers import add, concatenate
from keras.models import Model
import struct
import cv2


class WeightReader:
    def __init__(self, weight_file):
        with open(weight_file, 'rb') as w_f:
            major, = struct.unpack('i', w_f.read(4))
            minor, = struct.unpack('i', w_f.read(4))
            revision, = struct.unpack('i', w_f.read(4))

            if (major * 10 + minor) >= 2 and major < 1000 and minor < 1000:
                w_f.read(8)
            else:
                w_f.read(4)

            transpose = (major > 1000) or (minor > 1000)

            binary = w_f.read()

        self.offset = 0
        self.all_weights = np.frombuffer(binary, dtype='float32')

    def read_bytes(self, size):
        self.offset = self.offset + size
        return self.all_weights[self.offset - size:self.offset]

    def load_weights(self, model):
        for i in range(106):
            try:
                conv_layer = model.get_layer('conv_' + str(i))
                print("loading weights of convolution #" + str(i))

                if i not in [81, 93, 105]:
                    norm_layer = model.get_layer('bnorm_' + str(i))

                    size = np.prod(norm_layer.get_weights()[0].shape)

                    beta = self.read_bytes(size) # bias
                    gamma = self.read_bytes(size) # scale
                    mean = self.read_bytes(size) # mean
                    var = self.read_bytes(size) # variance

                    weights = norm_layer.set_weights([gamma, beta, mean, var])

                if len(conv_layer.get_weights()) > 1:
                    bias = self.read_bytes(np.prod(conv_layer.get_weights()[1].shape))
                    kernel = self.read_bytes(np.prod(conv_layer.get_weights()[0].shape))

                    kernel = kernel.reshape(list(reversed(conv_layer.get_weights()[0].shape)))
                    kernel = kernel.transpose([2, 3, 1, 0])
                    conv_layer.set_weights([kernel, bias])
                else:
                    kernel = self.read_bytes(np.prod(conv_layer.get_weights()[0].shape))
                    kernel = kernel.reshape(list(reversed(conv_layer.get_weights()[0].shape)))
                    kernel = kernel.transpose([2, 3, 1, 0])
                    conv_layer.set_weights([kernel])
            except ValueError:
                print("no convolution #" + str(i))
```

```
    def reset(self):
        self.offset = 0


class BoundBox:
    def __init__(self, xmin, ymin, xmax, ymax, objness=None, classes=None):
        self.xmin = xmin
        self.ymin = ymin
        self.xmax = xmax
        self.ymax = ymax

        self.objness = objness
        self.classes = classes

        self.label = -1
        self.score = -1

    def get_label(self):
        if self.label == -1:
            self.label = np.argmax(self.classes)

        return self.label

    def get_score(self):
        if self.score == -1:
            self.score = self.classes[self.get_label()]

        return self.score


def _conv_block(inp, convs, skip=True):
    x = inp
    count = 0

    for conv in convs:
        if count == (len(convs) - 2) and skip:
            skip_connection = x
        count += 1

        if conv['stride'] > 1: x = ZeroPadding2D(((1, 0), (1, 0)))(x)  # peculiar padding as darknet prefer left and top
        x = Conv2D(conv['filter'],
                   conv['kernel'],
                   strides=conv['stride'],
                   padding='valid' if conv['stride'] > 1 else 'same',  # peculiar padding as darknet prefer left and top
                   name='conv_' + str(conv['layer_idx']),
                   use_bias=False if conv['bnorm'] else True)(x)
        if conv['bnorm']: x = BatchNormalization(epsilon=0.001, name='bnorm_' + str(conv['layer_idx']))(x)
        if conv['leaky']: x = LeakyReLU(alpha=0.1, name='leaky_' + str(conv['layer_idx']))(x)

    return add([skip_connection, x]) if skip else x


def _interval_overlap(interval_a, interval_b):
    x1, x2 = interval_a
    x3, x4 = interval_b

    if x3 < x1:
        if x4 < x1:
            return 0
        else:
            return min(x2, x4) - x1
    else:
        if x2 < x3:
            return 0
        else:
            return min(x2, x4) - x3
```

```
def _sigmoid(x):
    return 1. / (1. + np.exp(-x))


def bbox_iou(box1, box2):
    intersect_w = _interval_overlap([box1.xmin, box1.xmax], [box2.xmin, box2.xmax])
    intersect_h = _interval_overlap([box1.ymin, box1.ymax], [box2.ymin, box2.ymax])

    intersect = intersect_w * intersect_h

    w1, h1 = box1.xmax - box1.xmin, box1.ymax - box1.ymin
    w2, h2 = box2.xmax - box2.xmin, box2.ymax - box2.ymin

    union = w1 * h1 + w2 * h2 - intersect

    return float(intersect) / union


def make_yolov3_model():
    input_image = Input(shape=(None, None, 3))

    # Layer 0 => 4
    x = _conv_block(input_image, [{'filter': 32, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 0},
                                  {'filter': 64, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 1},
                                  {'filter': 32, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 2},
                                  {'filter': 64, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 3}])

    # Layer 5 => 8
    x = _conv_block(x, [{'filter': 128, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 5},
                        {'filter': 64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 6},
                        {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 7}])

    # Layer 9 => 11
    x = _conv_block(x, [{'filter': 64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 9},
                        {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 10}])

    # Layer 12 => 15
    x = _conv_block(x, [{'filter': 256, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 12},
                        {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 13},
                        {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 14}])

    # Layer 16 => 36
    for i in range(7):
        x = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 16 + i * 3},
                            {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 17 + i * 3}])

    skip_36 = x

    # Layer 37 => 40
    x = _conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 37},
                        {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
```

```
True,                                   'layer_idx':                                    38},
                        {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky':
True,                                   'layer_idx':                                    39}])

    #                   Layer                   41                  =>                  61
    for                         i                       in                      range(7):
        x                       =                   _conv_block(x,                      [
            {'filter':  256,  'kernel':  1,  'stride':  1,  'bnorm':  True,  'leaky':  True,
'layer_idx':                41              +               i               *               3},
            {'filter':  512,  'kernel':  3,  'stride':  1,  'bnorm':  True,  'leaky':  True,
'layer_idx':                42              +               i               *               3}])

    skip_61                                     =                                       x

    #                   Layer                   62                  =>                  65
    x = _conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky':
True,                                   'layer_idx':                                    62},
                        {'filter': 512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
True,                                   'layer_idx':                                    63},
                        {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky':
True,                                   'layer_idx':                                    64}])

    #                   Layer                   66                  =>                  74
    for                         i                       in                      range(3):
        x                       =                   _conv_block(x,                      [
            {'filter':  512,  'kernel':  1,  'stride':  1,  'bnorm':  True,  'leaky':  True,
'layer_idx':                66              +               i               *               3},
            {'filter':  1024,  'kernel':  3,  'stride':  1,  'bnorm':  True,  'leaky':  True,
'layer_idx':                67              +               i               *               3}])

    #                   Layer                   75                  =>                  79
    x = _conv_block(x, [{'filter': 512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
True,                                   'layer_idx':                                    75},
                        {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky':
True,                                   'layer_idx':                                    76},
                        {'filter': 512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
True,                                   'layer_idx':                                    77},
                        {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky':
True,                                   'layer_idx':                                    78},
                        {'filter': 512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
True,                                   'layer_idx':                                    79}],
                    skip=False)

    #                   Layer                   80                  =>                  82
    yolo_82 = _conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True,
'leaky':                    True,                   'layer_idx':                    80},
                              {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False,
'leaky':                                                                            False,
                               'layer_idx':                 81}],                skip=False)

    #                   Layer                   83                  =>                  86
    x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
True,                                   'layer_idx':                                    84}],
                    skip=False)
    x                               =                                   UpSampling2D(2)(x)
    x                   =                       concatenate([x,                         skip_61])

    #                   Layer                   87                  =>                  91
    x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
True,                                   'layer_idx':                                    87},
                        {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky':
True,                                   'layer_idx':                                    88},
                        {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
True,                                   'layer_idx':                                    89},
                        {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky':
True,                                   'layer_idx':                                    90},
                        {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
True,                                   'layer_idx':                                    91}],
                    skip=False)
```

```
    #                   Layer                   92                  =>                  94
    yolo_94 = _conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True,
'leaky':                    True,                   'layer_idx':                    92},
                            {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False,
'leaky':                                                                    False,
                             'layer_idx':               93}],               skip=False)

    #                   Layer                   95                  =>                  98
    x = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
True,                                 'layer_idx':                                  96}],
                    skip=False)
    x                                 =                                 UpSampling2D(2)(x)
    x                   =                   concatenate([x,                     skip_36])

    #                   Layer                   99                  =>                  106
    yolo_106 = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,
'leaky':                    True,                   'layer_idx':                    99},
                               {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,
'leaky':                                                                    True,
                                'layer_idx':                                100},
                               {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,
'leaky':                                                                    True,
                                'layer_idx':                                101},
                               {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,
'leaky':                                                                    True,
                                'layer_idx':                                102},
                               {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,
'leaky':                                                                    True,
                                'layer_idx':                                103},
                               {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,
'leaky':                                                                    True,
                                'layer_idx':                                104},
                               {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False,
'leaky':                                                                    False,
                                'layer_idx':                105}],              skip=False)

    model       =       Model(input_image,       [yolo_82,       yolo_94,       yolo_106])
    return                                                                      model


def                 preprocess_input(image,                 net_h,                  net_w):
    new_h,                  new_w,                  _                   =                   image.shape

    #        determine        the        new        size        of        the        image
    if      (float(net_w)       /       new_w)       <       (float(net_h)       /       new_h):
        new_h          =          (new_h          *          net_w)          /          new_w
        new_w                                   =                                       net_w
    else:
        new_w          =          (new_w          *          net_h)          /          new_h
        new_h                                   =                                       net_h

    #        resize        the        image        to        the        new        size
    resized   =   cv2.resize(image[:,   :,   ::-1]   /   255.,   (int(new_w),   int(new_h)))

    #      embed      the      image      into      the      standard      letter      box
    new_image        =        np.ones((net_h,        net_w,        3))        *        0.5
    new_image[int((net_h - new_h) // 2):int((net_h + new_h) // 2), int((net_w - new_w) //
2):int((net_w                 +                 new_w)                 //                 2),
    :]                                  =                                   resized
    new_image                 =                 np.expand_dims(new_image,                 0)

    return                                                                  new_image


def    decode_netout(netout,    anchors,    obj_thresh,    nms_thresh,    net_h,    net_w):
    grid_h,                 grid_w                  =                   netout.shape[:2]
    nb_box                                  =                                       3
    netout       =       netout.reshape((grid_h,       grid_w,       nb_box,       -1))
    nb_class             =              netout.shape[-1]             -             5
```

```
    boxes                                       =                                       []

    netout[...,           :2]           =           _sigmoid(netout[...,           :2])
    netout[...,           4:]           =           _sigmoid(netout[...,           4:])
    netout[...,   5:]   =   netout[...,   4][...,   np.newaxis]   *   netout[...,   5:]
    netout[...,       5:]       *=       netout[...,       5:]       >       obj_thresh

    for           i           in           range(grid_h           *           grid_w):
        row                 =                 i                 /                 grid_w
        col                 =                 i                 %                 grid_w

        for                     b                     in                     range(nb_box):
            #          4th          element          is          objectness          score
            objectness                  =                  netout[int(row)][int(col)][b][4]
            #            objectness            =            netout[...,            :4]

            if          (objectness.all()          <=          obj_thresh):          continue

            #      first      4      elements      are      x,      y,      w,      and      h
            x,        y,        w,        h        =        netout[int(row)][int(col)][b][:4]

            x  =  (col  +  x)  /  grid_w     #  center  position,  unit:  image  width
            y  =  (row  +  y)  /  grid_h     #  center  position,  unit:  image  height
            w  =  anchors[2  *  b  +  0]  *  np.exp(w)  /  net_w     #  unit:  image  width
            h  =  anchors[2  *  b  +  1]  *  np.exp(h)  /  net_h     #  unit:  image  height

            #         last         elements         are         class         probabilities
            classes                     =                     netout[int(row)][col][b][5:]

            box = BoundBox(x - w / 2, y - h / 2, x + w / 2, y + h / 2, objectness, classes)
            #    box    =    BoundBox(x-w/2,    y-h/2,    x+w/2,    y+h/2,    None,    classes)

            boxes.append(box)

    return                                                                      boxes


def       correct_yolo_boxes(boxes,       image_h,       image_w,       net_h,       net_w):
    if      (float(net_w)      /      image_w)      <      (float(net_h)      /      image_h):
        new_w                                   =                                   net_w
        new_h          =          (image_h          *          net_w)          /          image_w
    else:
        new_h                                   =                                   net_w
        new_w          =          (image_w          *          net_h)          /          image_h

    for                     i                     in                     range(len(boxes)):
        x_offset,  x_scale  =  (net_w  -  new_w)  /  2.  /  net_w,  float(new_w)  /  net_w
        y_offset,  y_scale  =  (net_h  -  new_h)  /  2.  /  net_h,  float(new_h)  /  net_h

        boxes[i].xmin   =   int((boxes[i].xmin   -   x_offset)   /   x_scale   *   image_w)
        boxes[i].xmax   =   int((boxes[i].xmax   -   x_offset)   /   x_scale   *   image_w)
        boxes[i].ymin   =   int((boxes[i].ymin   -   y_offset)   /   y_scale   *   image_h)
        boxes[i].ymax   =   int((boxes[i].ymax   -   y_offset)   /   y_scale   *   image_h)


def                             do_nms(boxes,                                   nms_thresh):
    if                        len(boxes)                        >                        0:
        nb_class                            =                            len(boxes[0].classes)
    else:
        return

    for                     c                     in                     range(nb_class):
        sorted_indices    =    np.argsort([-box.classes[c]    for    box    in    boxes])

        for                 i                 in                 range(len(sorted_indices)):
            index_i                          =                          sorted_indices[i]

            if          boxes[index_i].classes[c]          ==          0:          continue

            for       j       in       range(i       +       1,       len(sorted_indices)):
```

```
                index_j                         =                       sorted_indices[j]

                if     bbox_iou(boxes[index_i],     boxes[index_j])     >=     nms_thresh:
                    boxes[index_j].classes[c]                       =                       0


def      draw_boxes(image,      boxes,      line,      labels,      obj_thresh,      dcnt):
    print(line)

    for                         box                         in                          boxes:
        label_str                                   =                                       ''
        label                                     =                                       -1

        for                   i                   in                  range(len(labels)):
            if                box.classes[i]                >                obj_thresh:
                label_str                               +=                              labels[i]
                label                                     =                                   i
                print(labels[i]   +   ':   '   +   str(box.classes[i]   *   100)   +   '%')
                print('line:  (' + str(line[0][0]) + ', ' + str(line[0][1]) + ')  (' +
str(line[1][0])            +            ',            '            +            str(
                    line[1][1])                             +                           ')')
                print('Box:  (' + str(box.xmin) + ', ' + str(box.ymin) + ')  (' +
str(box.xmax)            +            ',            '            +            str(
                    box.ymax)                           +                           ')')
                print()

        if                        label                         >=                        0:
            tf                                  =                                   False

            (rxmin,           rymin)           =           (box.xmin,           box.ymin)
            (rxmax,           rymax)           =           (box.xmax,           box.ymax)

            tf                                  =                                   False
            tf  |=  intersection(line[0],  line[1],  (rxmin,  rymin),  (rxmin,  rymax))
            tf  |=  intersection(line[0],  line[1],  (rxmax,  rymin),  (rxmax,  rymax))
            tf  |=  intersection(line[0],  line[1],  (rxmin,  rymin),  (rxmax,  rymin))
            tf  |=  intersection(line[0],  line[1],  (rxmin,  rymax),  (rxmax,  rymax))

            print(tf)

            cv2.line(image,      line[0],      line[1],      (255,      0,      0),      3)

            if                                                                      tf:
                cv2.rectangle(image, (box.xmin, box.ymin), (box.xmax, box.ymax), (255, 0,
0),                                                                                 3)
                cimg         =         image[box.ymin:box.ymax,         box.xmin:box.xmax]
                cv2.imshow("violation",                                         cimg)
                cv2.waitKey(5)
                cv2.imwrite(
                    "G:/Traffic  Violation  Detection/Traffic  Signal  Violation  Detection
System/Detected                Images/violation_"                +                str(
                        dcnt)               +               ".jpg",               cimg)
                dcnt               =               dcnt               +               1
            else:
                cv2.rectangle(image, (box.xmin, box.ymin), (box.xmax, box.ymax), (0, 255,
0),                                                                                 3)

            cv2.putText(image,
                        label_str    +    '    '    +    str(round(box.get_score(),    2)),
                        (box.xmin,              box.ymin              -              13),
                        cv2.FONT_HERSHEY_SIMPLEX,
                        1e-3                        *                       image.shape[0],
                        (0,                 255,                0),                 2)

    return                                                                  image


weights_path                                                                        =
r"C:\Users\NAVEEN\PycharmProjects\pythonProject4\pythonProject\pythonProject1\envi\venv\
Scripts\Traffic-641x400-1                                                     (1).jpg"
```

```
# set some parameters
net_h, net_w = 416, 416
obj_thresh, nms_thresh = 0.5, 0.45
anchors = [[116, 90, 156, 198, 373, 326], [30, 61, 62, 45, 59, 119], [10, 13, 16, 30, 33, 23]]
labels = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus", "train", "truck", \
          "boat", "traffic light", "fire hydrant", "stop sign", "parking meter", "bench", \
          "bird", "cat", "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra", "giraffe", \
          "backpack", "umbrella", "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard", \
          "sports ball", "kite", "baseball bat", "baseball glove", "skateboard", "surfboard", \
          "tennis racket", "bottle", "wine glass", "cup", "fork", "knife", "spoon", "bowl", \
          "banana", \
          "apple", "sandwich", "orange", "broccoli", "carrot", "hot dog", "pizza", "donut", \
          "cake", \
          "chair", "sofa", "pottedplant", "bed", "diningtable", "toilet", "tvmonitor", "laptop", \
          "mouse", \
          "remote", "keyboard", "cell phone", "microwave", "oven", "toaster", "sink", "refrigerator", \
          "book", "clock", "vase", "scissors", "teddy bear", "hair drier", "toothbrush"]

# make the yolov3 model to predict 80 classes on COCO
yolov3 = make_yolov3_model()

# load the weights trained on COCO into the model
weight_reader = WeightReader(weights_path)
weight_reader.load_weights(yolov3)


# my defined functions
def intersection(p, q, r, t):
    print(p, q, r, t)
    (x1, y1) = p
    (x2, y2) = q

    (x3, y3) = r
    (x4, y4) = t

    a1 = y1 - y2
    b1 = x2 - x1
    c1 = x1 * y2 - x2 * y1

    a2 = y3 - y4
    b2 = x4 - x3
    c2 = x3 * y4 - x4 * y3

    if (a1 * b2 - a2 * b1 == 0):
        return False
    print((a1, b1, c1), (a2, b2, c2))
    x = (b1 * c2 - b2 * c1) / (a1 * b2 - a2 * b1)
    y = (a2 * c1 - a1 * c2) / (a1 * b2 - a2 * b1)
    print((x, y))

    if x1 > x2:
        tmp = x1
        x1 = x2
        x2 = tmp
    if y1 > y2:
        tmp = y1
        y1 = y2
        y2 = tmp
    if x3 > x4:
        tmp = x3
        x3 = x4
        x4 = tmp
    if y3 > y4:
```

```
        tmp                                     =                                       y3
        y3                                      =                                       y4
        y4                                      =                                       tmp

    if x >= x1 and x <= x2 and y >= y1 and y <= y2 and x >= x3 and x <= x4 and y >= y3
and                             y                               <=                      y4:
        return                                                                          True
    else:
        return False
```

Output

C:\Users\NAVEEN\PycharmProjects\pythonProject4\pythonProject\pythonProject1\envi\venv\Scripts\python.exe C:\Users\NAVEEN\PycharmProjects\pythonProject4\pythonProject\pythonProject1\envi\main.py

2023-08-25 14:09:52.945734: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.

To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.

loading weights of convolution #0

loading weights of convolution #1

no convolution #1

loading weights of convolution #2

no convolution #2

loading weights of convolution #3

no convolution #3

no convolution #4

loading weights of convolution #5

no convolution #5

loading weights of convolution #6

no convolution #6

loading weights of convolution #7

no convolution #7

no convolution #8

loading weights of convolution #9

no convolution #9

loading weights of convolution #10

no convolution #10

no convolution #11

loading weights of convolution #12

no convolution #12

loading weights of convolution #13

no convolution #13

loading weights of convolution #14

no convolution #14

no convolution #15

loading weights of convolution #16

no convolution #16

loading weights of convolution #17

no convolution #17

no convolution #18

loading weights of convolution #19

no convolution #19

loading weights of convolution #20

no convolution #20

no convolution #21

loading weights of convolution #22

no convolution #22

loading weights of convolution #23

no convolution #23

no convolution #24

loading weights of convolution #25

no convolution #25

loading weights of convolution #26

no convolution #26

no convolution #27

loading weights of convolution #28

no convolution #28

loading weights of convolution #29

no convolution #29

no convolution #30

loading weights of convolution #31

no convolution #31

loading weights of convolution #32

no convolution #32

no convolution #33

loading weights of convolution #34

no convolution #34

loading weights of convolution #35

no convolution #35

no convolution #36

loading weights of convolution #37

no convolution #37

loading weights of convolution #38

no convolution #38

loading weights of convolution #39

no convolution #39

no convolution #40

loading weights of convolution #41

no convolution #41

loading weights of convolution #42

no convolution #42

no convolution #43

loading weights of convolution #44

no convolution #44

loading weights of convolution #45

no convolution #45

no convolution #46

loading weights of convolution #47

no convolution #47

loading weights of convolution #48

no convolution #48

no convolution #49

loading weights of convolution #50

no convolution #50

loading weights of convolution #51

no convolution #51

no convolution #52

loading weights of convolution #53

no convolution #53

loading weights of convolution #54

no convolution #54

no convolution #55

loading weights of convolution #56

no convolution #56

loading weights of convolution #57

no convolution #57

no convolution #58

loading weights of convolution #59

no convolution #59

loading weights of convolution #60

no convolution #60

no convolution #61

loading weights of convolution #62

no convolution #62

loading weights of convolution #63

no convolution #63

loading weights of convolution #64

no convolution #64

no convolution #65

loading weights of convolution #66

no convolution #66

loading weights of convolution #67

no convolution #67

no convolution #68

loading weights of convolution #69

no convolution #69

loading weights of convolution #70

no convolution #70

no convolution #71

loading weights of convolution #72

no convolution #72

loading weights of convolution #73

no convolution #73

no convolution #74

loading weights of convolution #75

no convolution #75

loading weights of convolution #76

no convolution #76

loading weights of convolution #77

no convolution #77

loading weights of convolution #78

no convolution #78

loading weights of convolution #79

no convolution #79

loading weights of convolution #80

no convolution #80

loading weights of convolution #81

no convolution #81

no convolution #82

no convolution #83

loading weights of convolution #84

no convolution #84

no convolution #85

no convolution #86

loading weights of convolution #87

no convolution #87

loading weights of convolution #88

no convolution #88

loading weights of convolution #89

no convolution #89

loading weights of convolution #90

no convolution #90

loading weights of convolution #91

no convolution #91

loading weights of convolution #92

no convolution #92

loading weights of convolution #93

no convolution #93

no convolution #94

no convolution #95

loading weights of convolution #96

no convolution #96

no convolution #97

no convolution #98

loading weights of convolution #99

no convolution #99

loading weights of convolution #100

no convolution #100

loading weights of convolution #101

no convolution #101

loading weights of convolution #102

no convolution #102

loading weights of convolution #103

no convolution #103

loading weights of convolution #104

no convolution #104

loading weights of convolution #105

no convolution #105

Process finished with exit code 0