Traffic Law Enforcement System Using YOLOv8 Documentation

By: Mayssa Gritli

Supervisor: Arafet Zouari

Special thanks to Becher Mejri for the metrics dashboard

Overview:

This project aims to develop a real-time traffic law enforcement system using computer vision and deep learning capable of detecting traffic violations in real-time using a camera mounted on a vehicle. The system will utilize a YOLOv8 model for object detection, specifically targeting traffic signs and signals (Speed Limits, Red/Green Light and Stop Sign). Upon detecting a violation, the system will save the violations details and record a video.

System Architecture

The system consists of several key components:

YOLOv8 Model: A custom-trained object detection model for identifying traffic signs and signals.

Violation Detection Algorithm: Logic to interpret detected objects and identify traffic violations.

Real-time Processing System: Integration of the model and algorithms with live camera feed.

Database: MySQL database for storing violation records and model performance metrics.

Frontend Application: Built with Outsystems for visualizing data.

Project Structure

- **main.py**: Main script for detecting traffic signs and processing video inputs.
- **checker.py**: Handles traffic control logic, including tracking vehicle behaviors, checking for violations and interfacing with a database for logging purposes.

- **YoloDetector.py:** For detecting objects in images or video frames using the YOLO object detection model from the ultralytics library using OpenCV for image processing, drawing bounding boxes around detected objects, and displaying their class labels and confidence scores.
- **YoloTraining**: Jupyter Notebook for setting up the environment, preparing the custom dataset, ,training the YOLO model with different parameters to find the most accuracy, plotting useful curves and plots and saving the details in a MySQL database.
- **Dashboard**: Built using OutSystems, this dashboard visualizes model training metrics and traffic violations data.

Components

1. YOLOv8 Model Training (YoloTraining.ipynb)

- **Dataset**: Custom traffic sign dataset from Kaggle
- **Training Process**:
    - Experimented with different hyperparameters to optimize accuracy
    - Logged training results in MySQL database for comparison
- **Best Model**: Saved as best.pt

2. CNN Model (for comparison)

- Implemented using TenserFlow and Keras:

```python
import tensorflow as tf
from tensorflow.keras import layers, models
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, average_precision_score
import matplotlib.pyplot as plt

def load_and_preprocess_data(csv_file, image_dir, batch_size=16):
    df = pd.read_csv(csv_file)

    def process_image(filename, xmin, ymin, xmax, ymax, label):
        image_path = tf.strings.join([image_dir, filename], separator='/')
        image = tf.io.read_file(image_path)
        image = tf.image.decode_jpeg(image, channels=3)
        image = tf.image.resize(image, [224, 224])
        image = tf.keras.applications.resnet50.preprocess_input(image)

        bbox = tf.convert_to_tensor([xmin, ymin, xmax, ymax], dtype=tf.float32)
        return image, (bbox, label)

    filenames = df['filename'].values
    labels = df['class'].values
    bboxes = df[['xmin', 'ymin', 'xmax', 'ymax']].values

    dataset = tf.data.Dataset.from_tensor_slices((filenames, bboxes[:, 0], bboxes[:, 1], bboxes[:, 2], bboxes[:, 3], labels))
    dataset = dataset.map(process_image, num_parallel_calls=tf.data.AUTOTUNE)
    dataset = dataset.batch(batch_size).prefetch(tf.data.AUTOTUNE)

    return dataset
```

```python
def create_model(num_classes, input_shape=(224, 224, 3)):
    base_model = tf.keras.applications.ResNet50(weights='imagenet', include_top=False, input_shape=input_shape)

    for layer in base_model.layers:
        layer.trainable = False

    x = base_model.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)

    # Dense layers for class prediction
    x1 = tf.keras.layers.Dense(1024, activation='relu')(x)
    x1 = tf.keras.layers.BatchNormalization()(x1)
    x1 = tf.keras.layers.Dropout(0.5)(x1)
    x1 = tf.keras.layers.Dense(512, activation='relu')(x1)
    x1 = tf.keras.layers.BatchNormalization()(x1)
    x1 = tf.keras.layers.Dropout(0.5)(x1)
    class_output = tf.keras.layers.Dense(num_classes, activation='softmax', name='class_output')(x1)

    # Dense layers for bounding box regression
    x2 = tf.keras.layers.Dense(1024, activation='relu')(x)
    x2 = tf.keras.layers.BatchNormalization()(x2)
    x2 = tf.keras.layers.Dropout(0.5)(x2)
    x2 = tf.keras.layers.Dense(512, activation='relu')(x2)
    x2 = tf.keras.layers.BatchNormalization()(x2)
    x2 = tf.keras.layers.Dropout(0.5)(x2)
    bbox_output = tf.keras.layers.Dense(4, name='bbox_output')(x2)

    model = tf.keras.Model(inputs=base_model.input, outputs=[bbox_output, class_output])
    return model
```

```python
def train_model(epochs=50, batch_size=16, initial_learning_rate=0.001):
    train_dataset = load_and_preprocess_data('/content/data/train_output.csv', '/content/data/car/train/images', batch_size)
    valid_dataset = load_and_preprocess_data('/content/data/valid_output.csv', '/content/data/car/valid/images', batch_size)

    train_df = pd.read_csv('/content/data/train_output.csv')
    num_classes = train_df['class'].nunique()

    model = create_model(num_classes)

    lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
        initial_learning_rate,
        decay_steps=10000,
        decay_rate=0.9,
        staircase=True)

    optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)

    model.compile(optimizer=optimizer,
                  loss={'class_output': 'sparse_categorical_crossentropy', 'bbox_output': 'mean_squared_error'},
                  loss_weights={'class_output': 1.0, 'bbox_output': 1.0},
                  metrics={'class_output': 'accuracy'})

    early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

    history = model.fit(train_dataset,
                        validation_data=valid_dataset,
                        epochs=epochs,
                        callbacks=[early_stopping],
                        verbose=1)
```

```python
                        epochs=epochs,
                        callbacks=[early_stopping],
                        verbose=1)

    model.save('/content/data/trained_model.h5')
    return history, model
# Run the training
history, trained_model = train_model(epochs=50, batch_size=16, initial_learning_rate=0.001)

# Evaluate on test set
test_dataset = load_and_preprocess_data('/content/data/test_output.csv', '/content/data/car/test/images', batch_size=32)
test_results = trained_model.evaluate(test_dataset)

print("Test results:")
for name, value in zip(trained_model.metrics_names, test_results):
    print(f"{name}: {value}")

# Plotting
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['class_output_accuracy'], label='Training Accuracy')
plt.plot(history.history['val_class_output_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
```

- Converting labels from Darknet to CSV format

```python
import os
import csv
from glob import glob
from PIL import Image
def yolo_to_tf_csv(base_dir, output_csv, dataset_type):
    labels_dir = os.path.join(base_dir, dataset_type, 'labels')
    images_dir = os.path.join(base_dir, dataset_type, 'images')
    #lhne bch njibou l labels lkol puisque kolhom text files
    yolo_files = glob(os.path.join(labels_dir, '*.txt'))
    #lhne bch naabiw l csv mteena bl content eli fl text files
    with open(output_csv, 'w', newline='') as csvfile:
        csv_writer = csv.writer(csvfile)
        csv_writer.writerow(['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax'])
        for yolo_file in yolo_files:
            base_name = os.path.splitext(os.path.basename(yolo_file))[0]
            image_file = os.path.join(images_dir, f"{base_name}.jpg")
            if not os.path.exists(image_file):
                print(f"Warning: Image file not found for {yolo_file}")
                continue
            with Image.open(image_file) as img:
                img_width, img_height = img.size

            with open(yolo_file, 'r') as f:
                lines = f.readlines()
            for line in lines:
                line = line.strip()
                if line and not line.startswith('#'):  # Skip empty lines and comments
                    try:
                        class_id, x_center, y_center, width, height = map(float, line.split())
                        x_min = int((x_center - width/2) * img_width)
                        y_min = int((y_center - height/2) * img_height)
```

```python
                    try:
                        class_id, x_center, y_center, width, height = map(float, line.split())
                        x_min = int((x_center - width/2) * img_width)
                        y_min = int((y_center - height/2) * img_height)
                        x_max = int((x_center + width/2) * img_width)
                        y_max = int((y_center + height/2) * img_height)
                        csv_writer.writerow([
                            os.path.basename(image_file),
                            img_width,
                            img_height,
                            int(class_id),
                            x_min,
                            y_min,
                            x_max,
                            y_max
                        ])
                    except ValueError as e:
                        print(f"Error processing line in {yolo_file}: {line}")
                        print(f"Error message: {str(e)}")
    print(f"Conversion complete for {dataset_type} set. Output saved to {output_csv}")
def process_all_sets(base_dir, output_dir):
    for dataset_type in ['train', 'valid', 'test']:
        output_csv = os.path.join(output_dir, f'{dataset_type}_output.csv')
        yolo_to_tf_csv(base_dir, output_csv, dataset_type)
base_dir = '/content/data/car'
output_dir = '/content/data'
process_all_sets(base_dir, output_dir)
```

```
Conversion complete for train set. Output saved to /content/data/train_output.csv
Conversion complete for valid set. Output saved to /content/data/valid_output.csv
Conversion complete for test set. Output saved to /content/data/test_output.csv
```

- Implementing code to measure model metrics
- Saving metrics to a MySQL database

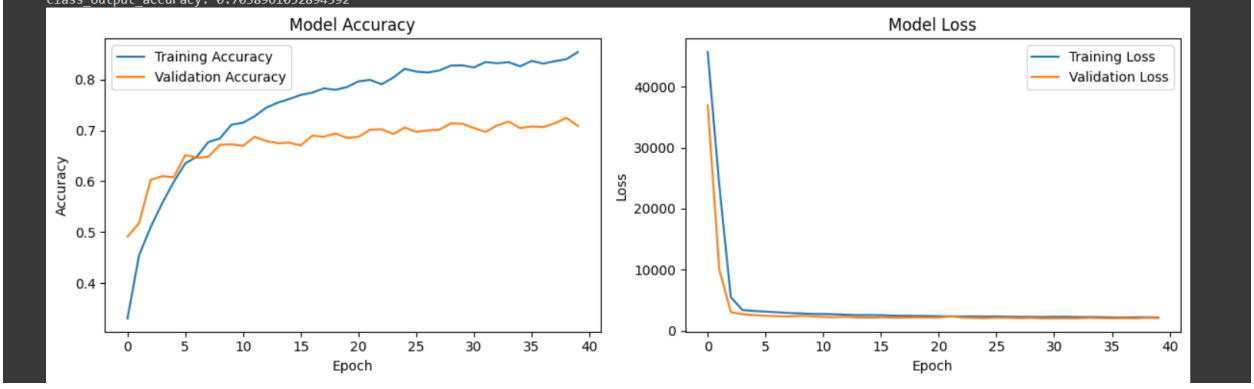
```
1 •  SELECT * FROM sql7720134.cnn_result;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

epochs	training_accuracy	validation_accuracy
39	0.843648	0.71822
40	0.845044	0.725636
41	0.843416	0.722458
42	0.840624	0.70339
43	0.852722	0.708686
44	0.84202	0.70339
45	0.851792	0.698093
46	0.85947	0.71822
47	0.859702	0.721398
48	0.85947	0.71822
49	0.855514	0.706568
50	0.852722	0.713983

- Plotting accuracy curves comparing validation and training performance

```
Epoch 40/50
269/269 [==============================] - 21s 79ms/step - loss: 2152.0396 - bbox_output_loss: 2151.6633 - class_output_loss: 0.3752 - class_output_accuracy: 0.853
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file for
  saving_api.save_model(
25/25 [==============================] - 3s 127ms/step - loss: 2640.3059 - bbox_output_loss: 2639.1169 - class_output_loss: 1.1890 - class_output_accuracy: 0.7039
Test results:
loss: 2640.305908203125
bbox_output_loss: 2639.116943359375
class_output_loss: 1.1889731884002686
class_output_accuracy: 0.7038961052894592
```



- Trained on the same dataset as YOLOv8
- Used for performance comparison with YOLOv8

3. YOLO Detector (yolo_detector.py)

- YOLODetector class:
  - Initializes the YOLO model
  - Provides methods for object detection and drawing bounding boxes
  - Integrated with OpenCV for video processing and frame analysis.

yolo_detector.py > YOLODetector > draw_detections

```python
from ultralytics import YOLO
import cv2
class YOLODetector:
    def __init__(self, model_path='best.pt'):
        self.model = YOLO(model_path)
        self.class_names = self.model.names
    def detect(self, frame, conf=0.5):
        results = self.model(frame, conf=conf)
        detections = []
        for r in results:
            boxes = r.boxes
            for box in boxes:
                x1, y1, x2, y2 = map(int, box.xyxy[0])
                cls = int(box.cls[0])
                conf = float(box.conf[0])
                class_name = self.class_names[cls]
                detections.append({
                    'class_name': class_name,
                    'bounding_box': (x1, y1, x2, y2),
                    'confidence': conf
                })
        return detections
    def draw_detections(self, frame, detections):
        for det in detections:
            x1, y1, x2, y2 = det['bounding_box']
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
            label = f"{det['class_name']} {det['confidence']:.2f}"
            cv2.putText(frame, label, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
        return frame
```

4. Violation Checker (checker.py)

checker.py > CarControl > __init_

```python
import mysql.connector
import cv2
import os
from datetime import datetime
'''
class hedhi tekhou 5 frames t9ayadhom each time , l threshold howa 9adeh lezm l ratio mtaa l sign fl whole image yfout
bch tthseb lkarhba taht lblaka '''
class CarControl:
    def __init__(self,fps,width,height, frames_to_check=5, threshold=0.003, db_config=None,rec_size=100):
        self.frames_to_check = frames_to_check
        self.fps = fps
        self.width = width
        self.height = height
        self.threshold = threshold
#history fih l frames l 5 eli n9aydou fehom esm lclasse taa sign si famech none , l ratio eli lezm akber mel threshold w l confidence
        self.history = []
        for i in range(frames_to_check):
            self.history.append(["none", 0, 0])
        #last 10 secs Window
        self.record = []
        self.max_rec_size = rec_size
        self.id = 0

        self.recent_red_conf=0
        self.in_red = False
        self.passed = True

        self.in_stop = False
        self.curr_speed_lim = 50
```

```python
        #details l db eli bch n9aydou feha l violations
        #Database connection
        self.db_config= {
                        'host': 'sql7.freemysqlhosting.net',
                        'user': 'sql7720134',
                        'database': 'sql7720134',
                        'password': '3b6uDHWTa3',
                        'port' : '3306'
                    }
        self.db_connection = None
        self.db_cursor = None
        self.connect_to_db()

        if self.db_config:
            self.connect_to_db()
#log nhebou nchoufou y9ayed wale (just testing)
    def log_viol(self, violID, type):
        time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        log_entry = f"{violID}_{type}_{time}\n"
        log_file = open("log.txt", "a")
        log_file.write(log_entry)
        log_file.close()

    def update_rec(self,frame):
        self.record.append(frame)
        if(len(self.record) > self.max_rec_size):
            self.record.pop(0)
```

checker.py > CarControl > __init__

```
 8  class CarControl:
53      def update_rec(self,frame):
56              self.record.pop(0)
57  #db error and exception handling
58      def connect_to_db(self):
59          try:
60              self.db_connection = mysql.connector.connect(**self.db_config)
61              self.db_cursor = self.db_connection.cursor()
62              print("Successfully connected to the database")
63          except mysql.connector.Error as err:
64              print(f"Error connecting to MySQL database: {err}")
65  #lhne naabiw l list taa l history chykoun feha list mtaa 5 frames kol frame aandou [calssname,ratio,confidence]
66      def update(self, sign_class_name, ratio,conf):
67          self.history.append([sign_class_name, ratio, conf])
68          if len(self.history) > self.frames_to_check:
69              self.history.pop(0)
70  #lhne check ken l sign red wale
71      def check_red(self):
72          for i in self.history:
73              if i[0] == "Red Light" and i[1] >= self.threshold:
74                  self.recent_red_conf =i[2]
75                  self.in_red = True
76   #check ken l sign green detected
77      def check_green(self):
78          for i in self.history:
79              if i[0] == "Green Light" and i[1] >= self.threshold:
80                  self.in_red = False
81  #check ken l sign Stop
82      def check_stop(self):
83          for i in self.history:
```

checker.py > CarControl > __init__

```
 8  class CarControl:
82      def check_stop(self):
83          for i in self.history:
84              if i[0] == "Stop" and i[1] >= self.threshold:
85                  self.in_stop = True
86   #check for which speed limit detected
87      def update_speed(self):
88          speed_limit_class_names = ["Speed Limit 10", "Speed Limit 20", "Speed Limit 30", "Speed Limit 40",
89                                     "Speed Limit 50", "Speed Limit 60", "Speed Limit 70", "Speed Limit 80",
90                                     "Speed Limit 90", "Speed Limit 100", "Speed Limit 110", "Speed Limit 120"]
91          for i in self.history:
92              if i[0] in speed_limit_class_names and i[1] >= self.threshold:
93                  self.curr_speed_lim = int(i[0].split()[-1])
94
95
96      def record_write(self, path):
97          os.makedirs(os.path.dirname(path), exist_ok=True)
98          fourcc = cv2.VideoWriter_fourcc(*'XVID')
99          out = cv2.VideoWriter(path, fourcc, self.fps, (self.width, self.height))
100         if not out.isOpened():
101             print(f"Error: Could not open video writer for {path}")
102             return
103         try:
104             for f in self.record:
105                 if f.shape != (self.height, self.width, 3):
106                     f = cv2.resize(f, (self.width, self.height))
107                 if len(f.shape) == 2:
108                     f = cv2.cvtColor(f, cv2.COLOR_GRAY2BGR)
109                 out.write(f)
```

```
109                 out.write(f)
110             print(f"Video successfully written to {path}")
111         except Exception as e:
112             print(f"Error while writing video: {str(e)}")
113         finally:
114             out.release()
115 
116         if os.path.exists(path) and os.path.getsize(path) > 0:
117             print(f"Video file created successfully: {path}")
118         else:
119             print(f"Error: Video file was not created or is empty: {path}")
120 
121         out.release()
122 #lhne lkhedma taa violations
123     def check_violation(self, speed):
124         violation_type = None
125         confidence_score = 0
126 
127         if self.in_stop and speed > 5:
128             violation_type = "STOP VIOLATION"
129             confidence_score = max([i[2] for i in self.history if i[0] == "Stop"])
130 
131         if speed > self.curr_speed_lim:
132             violation_type = "SPEED LIMIT EXCEEDED"
133             confidence_score = 0.8
134 
135         a = 0
136         b = 0
```

checker.py > CarControl > __init__

```
8   class CarControl:
123     def check_violation(self, speed):
136         b = 0
137         for i in self.history:
138             if  i[1] < self.threshold or i[0] != "Stop":
139                 b += 1
140             if  i[1] < self.threshold or i[0] != "Red Light":
141                 a += 1
142         if b == self.frames_to_check:
143             self.in_stop = False
144         if a == self.frames_to_check:
145             if self.in_red:
146                 confidence_score = self.recent_red_conf
147                 violation_type = "RED LIGHT VIOLATION"
148             self.in_red = False
149 
150         if violation_type:
151             print(violation_type)
152             if violation_type == "RED LIGHT VIOLATION":
153                 type = 'RD'
154             elif violation_type == "SPEED LIMIT EXCEEDED":
155                 type = "SP"
156             else:
157                 type = 'ST'
158             cursor = self.db_connection.cursor()
159             cursor.execute("SELECT MAX(DetectionID) FROM Results")
160             newID = cursor.fetchone()[0]
161             if newID is None:
162                 newID = 1
163             else:
```

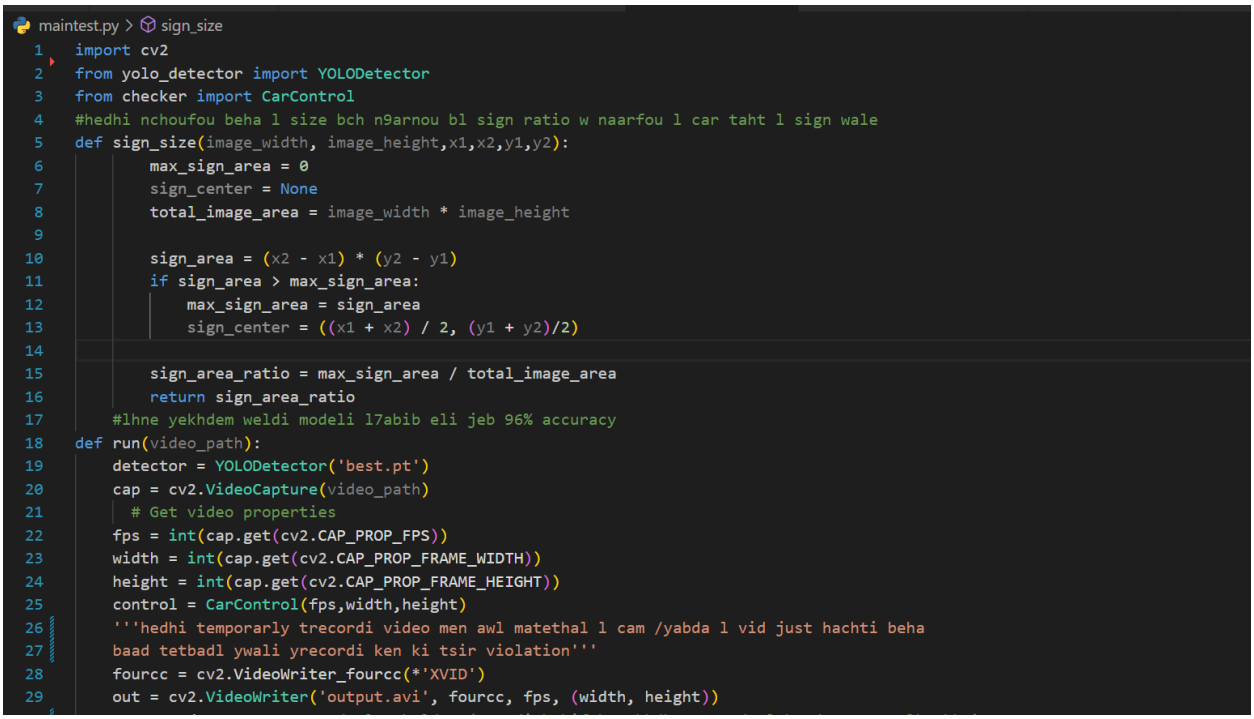
```
162                 newID = 1
163             else:
164                 newID = newID + 1
165             script_dir = os.path.dirname(os.path.abspath(__file__))
166             violations_dir = os.path.join(script_dir, "violations")
167             os.makedirs(violations_dir, exist_ok=True)
168             p = os.path.join(violations_dir, f"VIOLATION#{type}{newID}.avi")
169             print(f"Video will be saved to: {p}")
170             self.record_write(p)
171             cursor.close()
172             self.save_violation(newID, violation_type, speed, p, confidence_score)
173             self.log_viol(newID, violation_type)
174     #lhne nsavi l results fl db
175       def save_violation(self, DetectionID, violation_type, speed, video_path, confidence_score):
176           if not self.db_connection:
177               print("Database connection not established. Cannot save violation.")
178               return
179
180           timestamp = datetime.now()
181
182           insert_query = """
183           INSERT INTO Results
184           (DetectionID, Timestamp, ViolationType, Speed, Speed_Limit, Video_Path, Confidence_Score)
185           VALUES (%s, %s, %s, %s, %s, %s, %s)
186           """
187           violation_data = (
188               DetectionID,
189               timestamp,
```

checker.py > CarControl > __init__

```
 8   class CarControl:
175       def save_violation(self, DetectionID, violation_type, speed, video_path, confidence_score):
182           insert_query = """
183           INSERT INTO Results
184           (DetectionID, Timestamp, ViolationType, Speed, Speed_Limit, Video_Path, Confidence_Score)
185           VALUES (%s, %s, %s, %s, %s, %s, %s)
186           """
187           violation_data = (
188               DetectionID,
189               timestamp,
190               violation_type,
191               speed,
192               self.curr_speed_lim,
193               video_path,
194               confidence_score
195           )
196
197           try:
198               self.db_cursor.execute(insert_query, violation_data)
199               self.db_connection.commit()
200               print(f"Violation saved: {violation_type}")
201           except mysql.connector.Error as err:
202               print(f"Error saving violation to database: {err}")
203
204       def __del__(self):
205           if self.db_connection:
206               self.db_cursor.close()
207               self.db_connection.close()
```

- CarControl class:
  - Manages the state of detected signs and potential violations
  - Implements logic for various violation types (red light, speed limit, stop sign)
  - Logs events to a MySQL database for record-keeping and analysis
  - Outputs a video of the violation.

5. Main Application (maintest.py)



```python
import cv2
from yolo_detector import YOLODetector
from checker import CarControl
#hedhi nchoufou beha l size bch n9arnou bl sign ratio w naarfou l car taht l sign wale
def sign_size(image_width, image_height,x1,x2,y1,y2):
        max_sign_area = 0
        sign_center = None
        total_image_area = image_width * image_height

        sign_area = (x2 - x1) * (y2 - y1)
        if sign_area > max_sign_area:
            max_sign_area = sign_area
            sign_center = ((x1 + x2) / 2, (y1 + y2)/2)

        sign_area_ratio = max_sign_area / total_image_area
        return sign_area_ratio
    #lhne yekhdem weldi modeli l7abib eli jeb 96% accuracy
def run(video_path):
    detector = YOLODetector('best.pt')
    cap = cv2.VideoCapture(video_path)
      # Get video properties
    fps = int(cap.get(cv2.CAP_PROP_FPS))
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    control = CarControl(fps,width,height)
    '''hedhi temporarly trecordi video men awl matethal l cam /yabda l vid just hachti beha
    baad tetbadl ywali yrecordi ken ki tsir violation'''
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    out = cv2.VideoWriter('output.avi', fourcc, fps, (width, height))
```

maintest.py > run

```python
def run(video_path):
    out = cv2.VideoWriter('output.avi', fourcc, fps, (width, height))
    car_speed = 0  #Car Speed Placeholder (maandich kifeh nekhdhou speed ml hardware taa lkarhba)
#lhne nekhdhou l frames w nhezouhom lel carcontrol nental9ou f khedmet l checker
    while True:
        ret, frame = cap.read()
        if not ret:
            break

        image_height, image_width = frame.shape[:2]
        detections = detector.detect(frame)
        speed_limit_class_names = ["Speed Limit 10", "Speed Limit 20", "Speed Limit 30", "Speed Limit 40",
                                   "Speed Limit 50", "Speed Limit 60", "Speed Limit 70", "Speed Limit 80",
                                   "Speed Limit 90", "Speed Limit 100", "Speed Limit 110", "Speed Limit 120"]
        for_check=False

        for i, det in enumerate(detections, 1):
            class_name = det['class_name']
            x1, y1, x2, y2 = det['bounding_box']
            confidence = det['confidence']
            print(f"  Detection {i}:")
            print(f"  Class Name: {class_name}")
            print(f"  Bounding Box: (x1={x1}, y1={y1}, x2={x2}, y2={y2})")
            print(f"  Confidence: {confidence:.2f}")
            print()
            print(control.history)
            if confidence >= 0.5:
                for_check=True
                control.update(class_name,sign_size(image_width,image_height,x1,x2,y1,y2),confidence)
                print(control.history)
```

maintest.py > run

```python
def run(video_path):
                print(control.history)
                if class_name in speed_limit_class_names:
                    control.update_speed()
                if class_name == "Red Light":
                    control.check_red()
                if class_name == "Green Light":
                    control.check_green()
                if class_name == "Stop":
                    control.check_stop()
        if not for_check:
            control.update("none",0,0)

        control.check_violation(car_speed)

        frame = detector.draw_detections(frame, detections)
        control.update_rec(frame)
        out.write(frame)

        cv2.imshow("Traffic Sign Detection", frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
    out.release()
    cv2.destroyAllWindows()
if __name__ == "__main__":
    video_path = "testvid.mp4"
    run(video_path)
```

- Integrates the YOLODetector and CarControl classes
- Processes Live Camera feed and detects violations in real-time or from a video.

6. Dashboard using OutSystems

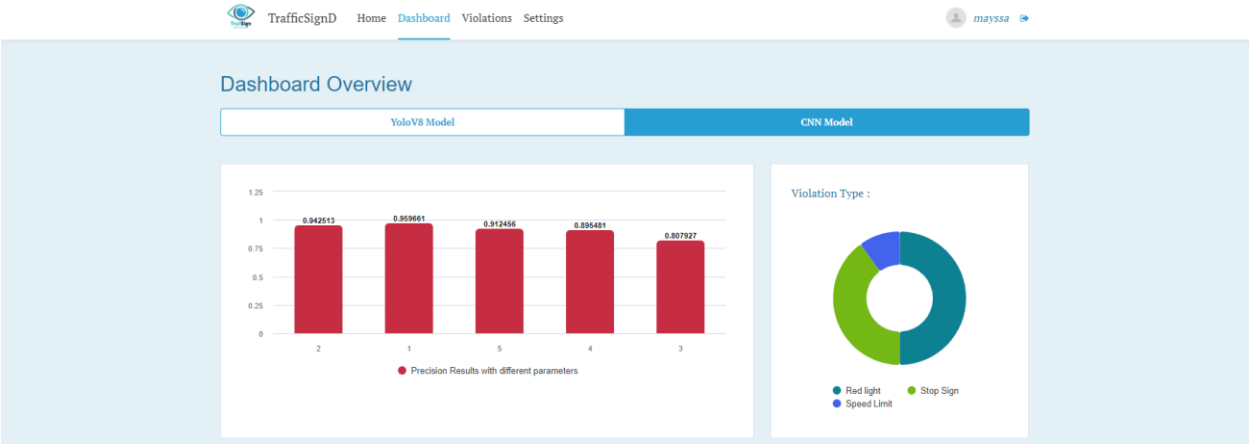
A dashboard created with OutSystems provides a user-friendly interface to visualize model performance and traffic violations. It consists of two main sections:

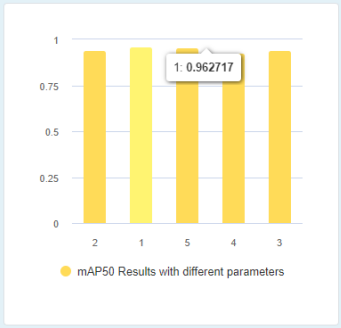
- **Model Training Metrics**:
  - **Purpose**: Displays various training metrics to evaluate the performance of the YOLO model and a custom Convolutional Neural Network (CNN) created for comparison.
  - **Metrics Displayed**:
    - **Training, Validation, and Testing Metrics**: Shows different parameters and metrics, such as accuracy, loss and precision for each phase of the model training process.
    - **Model Comparison**: Provides a comparative analysis between the YOLO model and the custom CNN based on their performance metrics. This helps in understanding which model performs better under different conditions.
  - **Data Source**: Metrics and performance data are generated during model training in Jupyter Notebook and stored in a MySQL database. The

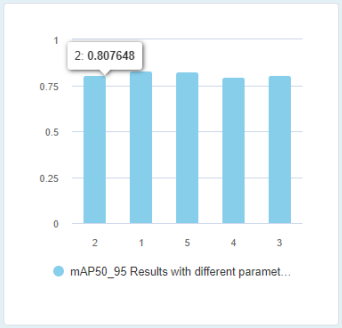
data is then imported into OutSystems using Integration Studio.

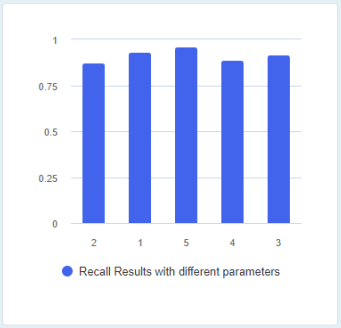
- **Traffic Violations**:
  - **Purpose**: Displays detailed information about detected traffic violations.
  - **Details Shown**:
    - **Violation Type**: Type of violation detected (e.g., running a red light, speeding).
    - **Timestamp**: The exact date and time when the violation occurred.
    - **Other Details**: Additional data such as speed and detection ID.

- **Data Source**: Violation data is inserted into a MySQL database from the detection system in maintest.py. This data is then imported into OutSystems using Integration Studio, allowing real-time monitoring and reporting.

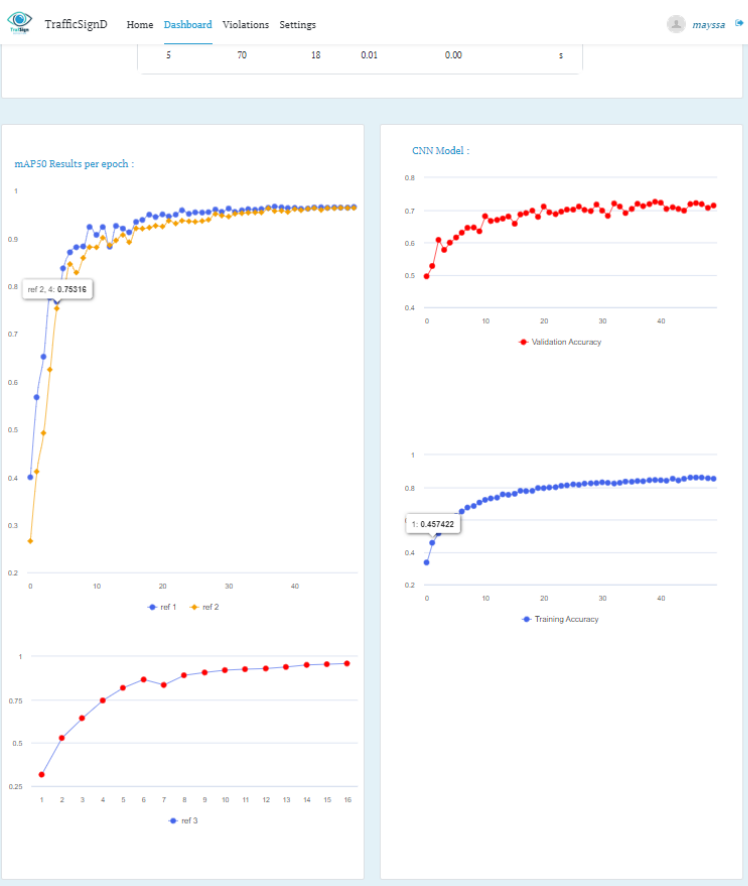


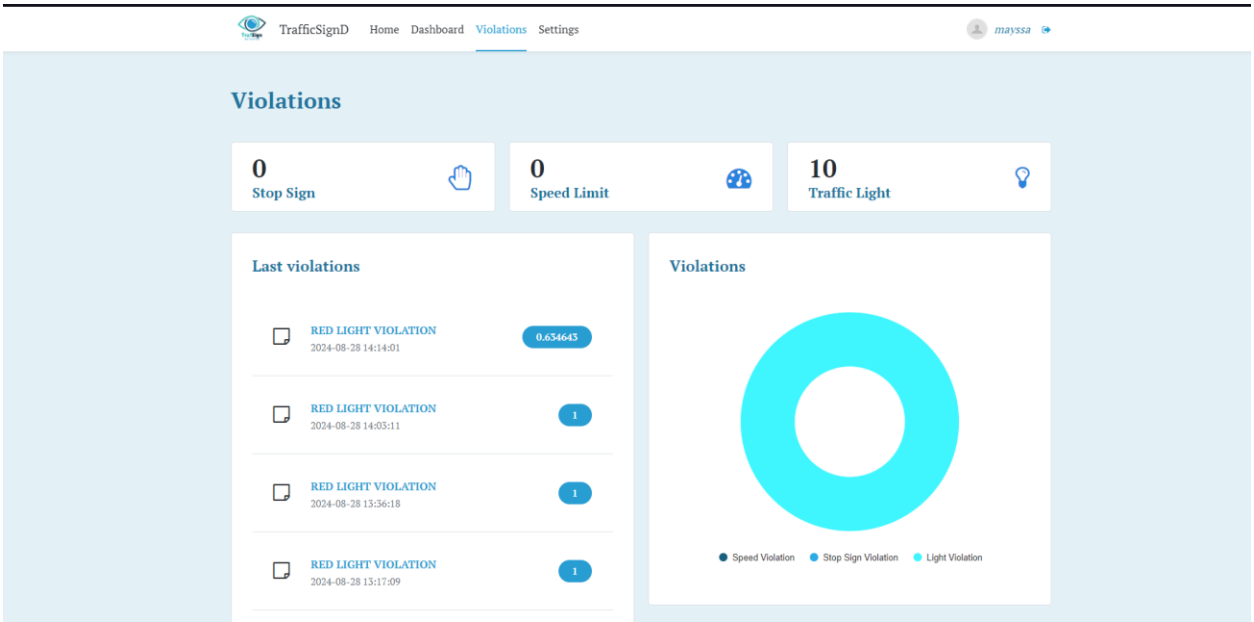






Ref	epoch	batch	lr	dropout	Model_Type
2	50	16	0.01	0.00	n
4	100	64	0.00	0.15	n
1	50	16	0.01	0.00	s
3	16	8	0.01	0.00	s
5	70	18	0.01	0.00	s





Detailed Components:

1. maintest.py

The main entry point for running the traffic detection system.

Key Functions:

- **sign_size(image_width, image_height, x1, x2, y1, y2)**:
  - Calculates the area and center of a detected sign relative to the image.
  - Parameters:
    - image_width, image_height: Dimensions of the input image.
    - x1, x2, y1, y2: Bounding box coordinates of the detected sign.
  - Returns:

        - The ratio of the sign area to the total image area.
- **run(video_path)**:
    - Initializes the YOLO detector and processes video frames to detect traffic signs and control vehicle behavior.
    - Parameters:
        - video_path: Path to the input video file (when I want to use a video instead of live feed)
    - Execution Flow:
        - Initializes the YOLO detector with a pre-trained model (best.pt).
        - Captures video frames using OpenCV and extracts properties like FPS, width, and height.
        - Initializes CarControl to handle detected signs and traffic logic.
        - Processes each video frame to detect traffic signs and identify any violations.

2. checker.py

Contains the CarControl class, which manages traffic control logic and database interactions.

Key Class:

- **CarControl**:
    - Handles the detection and management of traffic violations based on detected signs.
    - **Constructor Parameters**:

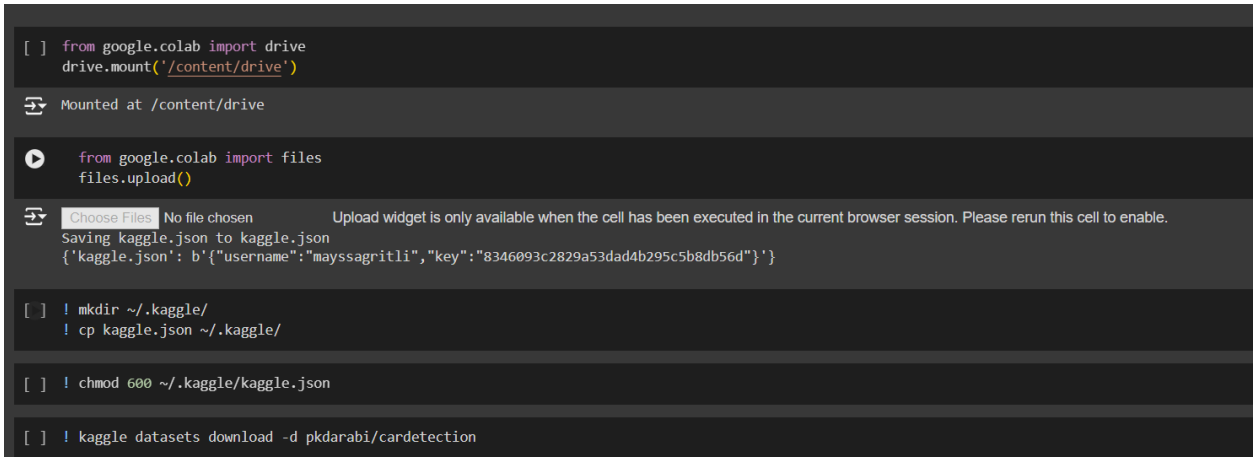
        - fps, width, height: Video properties to understand the context of detections.
        - frames_to_check: Number of frames to evaluate for a decision-making process.
        - threshold: minimum sign ratio for the car to be considered positioned under the traffic sign.
        - rec_size: Size of the violation output video (frames).
    - **Methods**:
        - **Initialization and Configuration**:
            - Sets up internal state variables for tracking and decision-making.
            - Connects to the MySQL database using the provided db_config.
        - **Violation Detection**:
            - Implements logic to detect traffic violations such as speeding, running red lights, and ignoring stop signs.
            - Uses flags like in_red, passed, and in_stop to track vehicle states.
        - **Database Interaction**:
            - Logs detected violations into the MySQL database.
            - Closes the database connection upon completion.

3. YoloTraining.ipynb

A Jupyter Notebook used for preparing the environment, datasets, and training the YOLO model.

Key Sections:

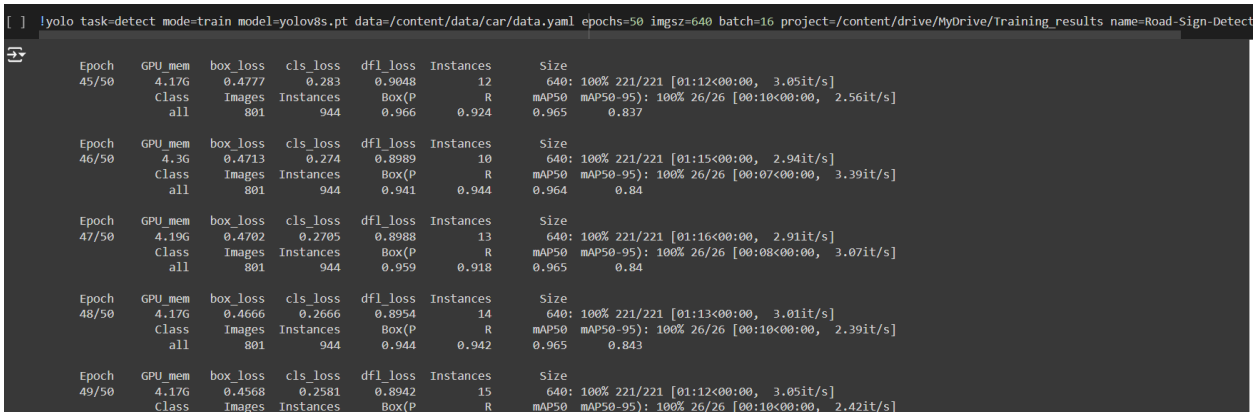
- **Environment Setup**:
  - Mounts Google Drive to access stored files and datasets.
  - Uses the Kaggle API to download datasets required for model training.



```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive

    from google.colab import files
    files.upload()

    Choose Files  No file chosen   Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
    Saving kaggle.json to kaggle.json
    {'kaggle.json': b'{"username":"mayssagritli","key":"8346093c2829a53dad4b295c5b8db56d"}'}

[ ] ! mkdir ~/.kaggle/
    ! cp kaggle.json ~/.kaggle/

[ ] ! chmod 600 ~/.kaggle/kaggle.json

[ ] ! kaggle datasets download -d pkdarabi/cardetection
```

- **Model Training**:
  - Trains the YOLOv8 model using the prepared dataset.

Training



```
[ ] !yolo task=detect mode=train model=yolov8s.pt data=/content/data/car/data.yaml epochs=50 imgsz=640 batch=16 project=/content/drive/MyDrive/Training_results name=Road-Sign-Detect

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      45/50      4.17G     0.4777      0.283     0.9048         12        640: 100% 221/221 [01:12<00:00,  3.05it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 26/26 [00:10<00:00,  2.56it/s]
                   all        801        944      0.966      0.924      0.965      0.837

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      46/50       4.3G     0.4713      0.274     0.8989         10        640: 100% 221/221 [01:15<00:00,  2.94it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 26/26 [00:07<00:00,  3.39it/s]
                   all        801        944      0.941      0.944      0.964       0.84

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      47/50      4.19G     0.4702     0.2705     0.8988         13        640: 100% 221/221 [01:16<00:00,  2.91it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 26/26 [00:08<00:00,  3.07it/s]
                   all        801        944      0.959      0.918      0.965       0.84

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      48/50      4.17G     0.4666     0.2666     0.8954         14        640: 100% 221/221 [01:13<00:00,  3.01it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 26/26 [00:10<00:00,  2.39it/s]
                   all        801        944      0.944      0.942      0.965      0.843

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      49/50      4.17G     0.4568     0.2581     0.8942         15        640: 100% 221/221 [01:12<00:00,  3.05it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 26/26 [00:10<00:00,  2.42it/s]
```

```
50 epochs completed in 1.238 hours.
Optimizer stripped from /content/drive/MyDrive/Training_results/Road-Sign-Detection2/weights/last.pt, 22.5MB
Optimizer stripped from /content/drive/MyDrive/Training_results/Road-Sign-Detection2/weights/best.pt, 22.5MB

Validating /content/drive/MyDrive/Training_results/Road-Sign-Detection2/weights/best.pt...
Ultralytics YOLOv8.2.48 🚀 Python-3.10.12 torch-2.3.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 11131389 parameters, 0 gradients, 28.5 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 26/26 [00:14<00:00,  1.80it/s]
                   all        801        944      0.939      0.944      0.966      0.844
           Green Light         87        122      0.841      0.825      0.847      0.527
             Red Light         74        108       0.83      0.806      0.829      0.525
       Speed Limit 100         52         52      0.945      0.992      0.994      0.904
       Speed Limit 110         17         17      0.971      0.941      0.976      0.916
       Speed Limit 120         60         60      0.964          1      0.995      0.933
        Speed Limit 20         56         56      0.967      0.982      0.987      0.866
        Speed Limit 30         71         74      0.933      0.973      0.992      0.928
        Speed Limit 40         53         55      0.964      0.978      0.985      0.896
        Speed Limit 50         68         71      0.942      0.901      0.986      0.871
        Speed Limit 60         76         76      0.964      0.947      0.982      0.909
        Speed Limit 70         78         78      0.959      0.974      0.992       0.91
        Speed Limit 80         56         56      0.941          1      0.995      0.891
        Speed Limit 90         38         38      0.945       0.91      0.973      0.814
                  Stop         81         81      0.985      0.988       0.99       0.92
Speed: 0.4ms preprocess, 4.5ms inference, 0.0ms loss, 3.8ms postprocess per image
Results saved to /content/drive/MyDrive/Training_results/Road-Sign-Detection2
```

Validation

```
from ultralytics import YOLO

# Load a trained model
model = YOLO('/content/drive/MyDrive/Training_results/Road-Sign-Detection1n/weights/best.pt')

# Validate the model
metrics = model.val(data='/content/data/car/data.yaml')
```

```
Ultralytics YOLOv8.2.61 🚀 Python-3.10.12 torch-2.3.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3,008,573 parameters, 0 gradients, 8.1 GFLOPs
val: Scanning /content/data/car/valid/labels.cache... 801 images, 0 backgrounds, 0 corrupt: 100%|██████████| 801/801 [00:00<?, ?it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 51/51 [00:12<00:00,  4.08it/s]
                   all        801        944      0.951      0.908      0.965      0.839
           Green Light         87        122      0.914       0.73      0.866      0.522
             Red Light         74        108      0.894      0.676      0.834      0.507
       Speed Limit 100         52         52      0.998      0.981      0.994      0.904
       Speed Limit 110         17         17      0.742      0.941      0.959      0.894
       Speed Limit 120         60         60          1      0.914       0.99      0.932
        Speed Limit 20         56         56       0.98      0.982      0.987      0.876
        Speed Limit 30         71         74      0.979      0.959      0.991      0.929
        Speed Limit 40         53         55      0.897      0.982      0.991      0.895
        Speed Limit 50         68         71          1       0.91      0.982      0.876
        Speed Limit 60         76         76      0.974      0.934      0.971      0.893
        Speed Limit 70         78         78      0.974      0.962      0.989      0.914
        Speed Limit 80         56         56      0.981      0.939      0.989       0.88
        Speed Limit 90         38         38          1      0.808       0.98      0.789
                  Stop         81         81      0.981      0.988      0.995      0.935
Speed: 1.6ms preprocess, 4.4ms inference, 0.0ms loss, 1.4ms postprocess per image
Results saved to runs/detect/val3
```

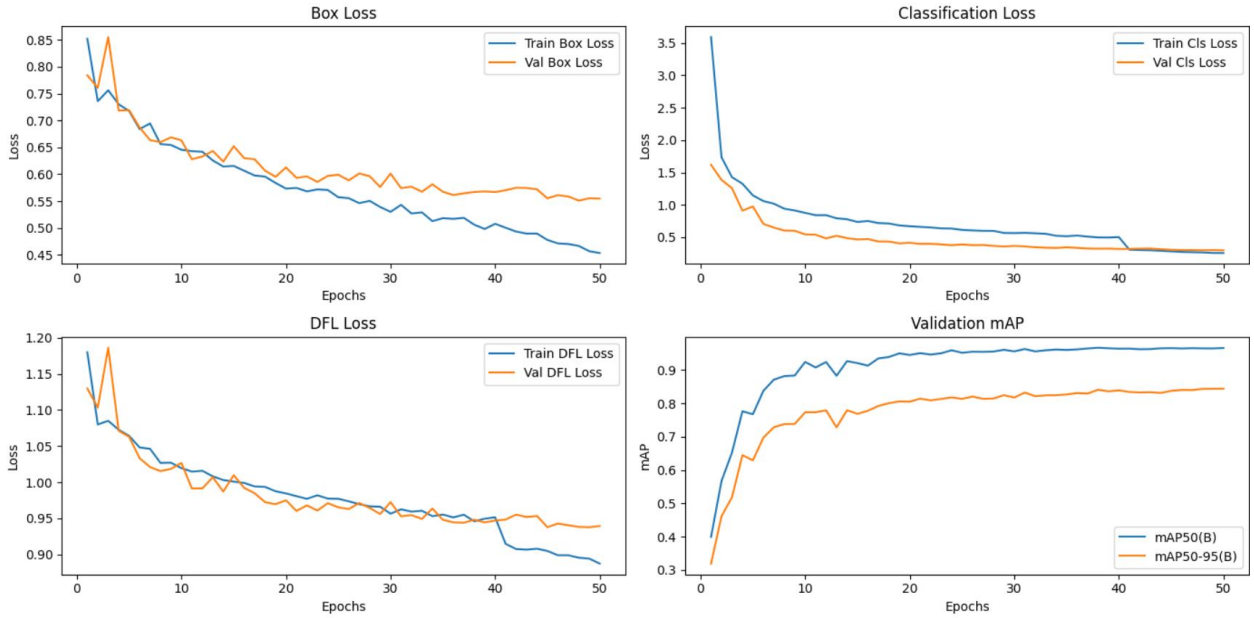
Testing

```python
# Testing the model i'm happy with(weldi)
from ultralytics import YOLO
Valid_model = YOLO('/content/drive/MyDrive/Training_results/Road-Sign-Detection2/weights/best.pt')

# Evaluating the model on the testset
metrics = Valid_model.val(split = 'test', data='/content/data/car/data.yaml')
# final results

print("precision(B): ", metrics.results_dict["metrics/precision(B)"])
print("metrics/recall(B): ", metrics.results_dict["metrics/recall(B)"])
print("metrics/mAP50(B): ", metrics.results_dict["metrics/mAP50(B)"])
print("metrics/mAP50-95(B): ", metrics.results_dict["metrics/mAP50-95(B)"])
```

```
Ultralytics YOLOv8.2.59 🚀 Python-3.10.12 torch-2.3.1+cu121 CPU (Intel Xeon 2.20GHz)
Model summary (fused): 168 layers, 11,131,389 parameters, 0 gradients, 28.5 GFLOPs
val: Scanning /content/data/car/test/labels.cache... 638 images, 1 backgrounds, 0 corrupt: 100%| | 638/638 [00:00<?, ?it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%| | 40/40 [10:14<00:00, 15.36s/it]
                   all        638        770      0.96       0.93      0.963       0.83
           Green Light         77        110     0.929      0.827      0.923      0.578
             Red Light         71         94     0.894      0.691        0.8      0.519
       Speed Limit 10          2          3         1      0.997      0.995      0.765
       Speed Limit 100        45         46     0.936          1      0.993      0.892
       Speed Limit 110        21         21         1      0.877      0.921      0.853
       Speed Limit 120        40         44     0.942      0.977      0.979      0.892
        Speed Limit 20        46         46      0.96      0.957      0.978      0.883
        Speed Limit 30        60         60     0.978       0.95      0.993      0.928
        Speed Limit 40        51         53     0.986      0.962      0.982      0.891
        Speed Limit 50        47         50     0.901       0.94      0.971      0.889
        Speed Limit 60        45         45     0.975      0.933      0.964      0.852
        Speed Limit 70        52         53     0.942      0.925      0.987      0.866
        Speed Limit 80        60         61     0.971          1      0.994      0.902
        Speed Limit 90        33         34     0.989      0.912      0.965      0.817
```

Plotting

```python
plt.ylabel('Loss')
plt.title('Box Loss')
plt.legend()

# Classification Loss
plt.subplot(2, 2, 2)
plt.plot(epochs, train_cls_loss, label='Train Cls Loss')
plt.plot(epochs, val_cls_loss, label='Val Cls Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Classification Loss')
plt.legend()

# DFL Loss
plt.subplot(2, 2, 3)
plt.plot(epochs, train_dfl_loss, label='Train DFL Loss')
plt.plot(epochs, val_dfl_loss, label='Val DFL Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('DFL Loss')
plt.legend()

# mAP50 and mAP50-95
plt.subplot(2, 2, 4)
plt.plot(epochs, map50, label='mAP50(B)')
plt.plot(epochs, map50_95, label='mAP50-95(B)')
plt.xlabel('Epochs')
plt.ylabel('mAP')
plt.title('Validation mAP')
plt.legend()
```



- Saves the trained model to Google Drive or the local environment.

- **Database Insertion**:
  - Provides code snippets to insert model training results or metadata into the MySQL database.

```
        # Connect to the database
        connection = mysql.connector.connect(
            host='sql7.freemysqlhosting.net',
            database='sql7720134',
            user='sql7720134',
            password='3b6uDHWTa3'
        )
        if connection.is_connected():
            cursor = connection.cursor()
            insert_query = """
            INSERT INTO TestMetrics (Precision_p, Recall_R, map50, map50_95,ref)
            VALUES (%s, %s, %s, %s,3)
            """
            cursor.execute(insert_query, data)
            connection.commit()
            print("Data inserted successfully into TestMetrics table")
    except Error as e:
        print(f"Error: {e}")

    finally:
        if connection.is_connected():
            cursor.close()
            connection.close()
            print("MySQL connection is closed")

if __name__ == "__main__":
    data = (precision0, recall0, map50_0, map50_95_0)
    insert_metrics_to_db(data)
```

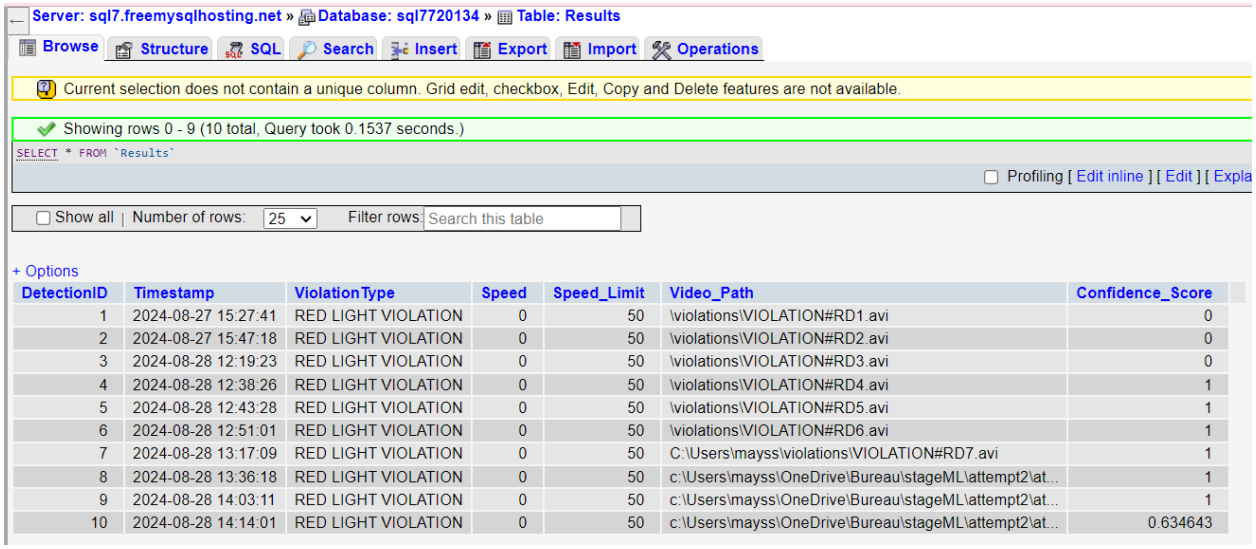
```
Data inserted successfully into TestMetrics table
MySQL connection is closed
```

Database Schema

- The system uses a MySQL database with the following tables:

```
CREATE TABLE Results (
    DetectionID INT PRIMARY KEY,
    Timestamp DATETIME,
    ViolationType VARCHAR(50),
    Speed FLOAT,
```

```
    Speed_Limit INT,
    Video_Path VARCHAR(255),
    Confidence_Score FLOAT
);
```



Server: sql7.freemysqlhosting.net » Database: sql7720134 » Table: Results

Browse | Structure | SQL | Search | Insert | Export | Import | Operations

Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available.

Showing rows 0 - 9 (10 total, Query took 0.1537 seconds.)

```
SELECT * FROM `Results`
```

Profiling [Edit inline] [Edit] [Expla

Show all | Number of rows: 25 Filter rows: Search this table

+ Options

DetectionID	Timestamp	ViolationType	Speed	Speed_Limit	Video_Path	Confidence_Score
1	2024-08-27 15:27:41	RED LIGHT VIOLATION	0	50	\violations\VIOLATION#RD1.avi	0
2	2024-08-27 15:47:18	RED LIGHT VIOLATION	0	50	\violations\VIOLATION#RD2.avi	0
3	2024-08-28 12:19:23	RED LIGHT VIOLATION	0	50	\violations\VIOLATION#RD3.avi	0
4	2024-08-28 12:38:26	RED LIGHT VIOLATION	0	50	\violations\VIOLATION#RD4.avi	1
5	2024-08-28 12:43:28	RED LIGHT VIOLATION	0	50	\violations\VIOLATION#RD5.avi	1
6	2024-08-28 12:51:01	RED LIGHT VIOLATION	0	50	\violations\VIOLATION#RD6.avi	1
7	2024-08-28 13:17:09	RED LIGHT VIOLATION	0	50	C:\Users\mayss\violations\VIOLATION#RD7.avi	1
8	2024-08-28 13:36:18	RED LIGHT VIOLATION	0	50	c:\Users\mayss\OneDrive\Bureau\stageML\attempt2\at...	1
9	2024-08-28 14:03:11	RED LIGHT VIOLATION	0	50	c:\Users\mayss\OneDrive\Bureau\stageML\attempt2\at...	1
10	2024-08-28 14:14:01	RED LIGHT VIOLATION	0	50	c:\Users\mayss\OneDrive\Bureau\stageML\attempt2\at...	0.634643

Usage

1. Ensure all dependencies are installed:

   pip install ultralytics opencv-python mysql-connector-python

2. Set up the MySQL database using the provided schema.
3. Run the main application:

   python maintest.py

Future Improvements

1. Implement real-time speed detection
2. Enhance the violation detection algorithms for higher accuracy
3. Train with another dataset to detect cars, other signs

Conclusion

This Traffic Law Enforcement System demonstrates the application of advanced computer vision techniques to improve road safety. By leveraging YOLOv8 for object detection and custom algorithms for violation detection, the system provides a robust solution for real-time traffic monitoring and driver behavior analysis.