!pip install opencv-python-headless

#for machine learning models

!pip install tensorflow

#micro web framework for setting up a simple web server.

!pip install flask

!pip install pandas

import cv2

import numpy as np

import matplotlib.pyplot as plt

import os

from pathlib import Path

from google.colab import drive

drive.mount('/content/drive')

project\_folder = '/content/drive/MyDrive/KPDL/image'

import os

# Đường dẫn tuyệt đối của thư mục dự án mới

# Thay đổi thư mục làm việc

os.chdir(project\_folder)

# Xác nhận thư mục làm việc mới

print("Thư mục làm việc hiện tại:", os.getcwd())

# Liệt kê tất cả các tập tin và thư mục trong thư mục làm việc mới

print("Danh sách các tập tin và thư mục:")

print(os.listdir('.'))

image\_paths = ('2.png', '4.png', '3.png', '5.png', '1.png', '6.png', '18.png', '13.png', '14.png', '12.png', '15.png', '10.png', '17.png', '8.png', '9.png', '16.png', '11.png', '7.png', '22.png', '31.png', '25.png', '23.png', '24.png', '28.png', '20.png', '30.png', '29.png', '19.png', '27.png', '21.png', '26.png', '32.png')

#set up 3 images per row

num\_rows = len(image\_paths) // 3 + (1 if len(image\_paths) % 3 else 0)

plt.figure(figsize=(15, 5 \* num\_rows))

#print images

for index, path in enumerate(image\_paths):

print(path)

image = cv2.imread(path)

#check if the image was loaded successfully

if image is not None:

plt.subplot(num\_rows, 3, index + 1)

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

#print name of image

plt.title(path.split('/')[-1])

plt.axis('off')

else:

print(f"Failed to load image at path: {path}")

#set layout

plt.tight\_layout()

plt.show()

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from skimage.filters import sobel

!pip install tensorflow

import cv2

import numpy as np

import matplotlib.pyplot as plt

def detect\_traffic\_light\_color(image, rect):

# Unpack the rectangle coordinates

x, y, w, h = rect

# Crop the region of interest (ROI) from the image

roi = image[y:y+h, x:x+w]

# Convert the ROI to HSV color space

hsv = cv2.cvtColor(roi, cv2.COLOR\_BGR2HSV)

# Define ranges for red, green, and yellow colors in HSV

# Note: These ranges might need adjustment depending on your specific images

lower\_red = np.array([100, 130, 130])

upper\_red = np.array([200, 350, 350])

lower\_green = np.array([240, 150, 150])

upper\_green = np.array([300, 350, 350])

# Create masks for colors

mask\_red = cv2.inRange(hsv, lower\_red, upper\_red)

mask\_green = cv2.inRange(hsv, lower\_green, upper\_green)

# Calculate the percentage of pixel within the mask for each color

red\_ratio = np.sum(mask\_red) / (w \* h)

green\_ratio = np.sum(mask\_green) / (w \* h)

# Determine the most dominant color in the ROI

if red\_ratio > green\_ratio:

detected\_color = "Red"

elif green\_ratio > red\_ratio:

detected\_color = "Green"

else:

detected\_color = "Unknown"

# For visualization, draw a rectangle around the ROI in the original image

cv2.rectangle(image, (x, y), (x+w, y+h), (255, 100, 100), 2)

return image, detected\_color

# Set up 3 images per row

num\_rows = len(image\_paths) // 3 + (1 if len(image\_paths) % 3 else 0)

plt.figure(figsize=(15, 5 \* num\_rows))

# Iterate through image paths

for index, path in enumerate(image\_paths):

image = cv2.imread(path)

if image is None:

print("Không thể tải hình ảnh từ", path)

else:

rect = (1720, 120, 130, 350) # Example rectangle coordinates

processed\_image, detected\_color = detect\_traffic\_light\_color(image, rect)

plt.subplot(num\_rows, 3, index + 1)

plt.imshow(cv2.cvtColor(processed\_image, cv2.COLOR\_BGR2RGB))

plt.title(f"Detected Color: {detected\_color}")

plt.axis('off')

# Set layout

plt.tight\_layout()

plt.show()

\*\*YOLOV9\*\*

!nvidia-smi

import os

HOME = os.getcwd()

print(HOME)

!git clone https://github.com/SkalskiP/yolov9.git

%cd yolov9

!pip install -r requirements.txt -q

!pip install -q roboflow

!wget -P {HOME}/weights -q https://github.com/WongKinYiu/yolov9/releases/download/v0.1/yolov9-c.pt

!wget -P {HOME}/weights -q https://github.com/WongKinYiu/yolov9/releases/download/v0.1/yolov9-e.pt

!wget -P {HOME}/weights -q https://github.com/WongKinYiu/yolov9/releases/download/v0.1/gelan-c.pt

!wget -P {HOME}/weights -q https://github.com/WongKinYiu/yolov9/releases/download/v0.1/gelan-e.pt

!python detect.py --weights {HOME}/weights/gelan-c.pt --conf 0.1 --source image\_paths --device 0

\*\*END YOLOV9\*\*

\*\*YOLOV3 detect traffic light\*\*

pip install requests

import requests

def download\_file(url, filename):

# Gửi một HTTP GET request đến URL

response = requests.get(url)

# Mở một tệp nhị phân mới để ghi

with open(filename, 'wb') as f:

f.write(response.content)

# URLs của tệp weights, cfg và names

url\_weights = 'https://pjreddie.com/media/files/yolov3.weights'

url\_cfg = 'https://raw.githubusercontent.com/pjreddie/darknet/master/cfg/yolov3.cfg'

url\_names = 'https://raw.githubusercontent.com/pjreddie/darknet/master/data/coco.names'

# Tải các tệp xuống

download\_file(url\_weights, 'yolov3.weights')

download\_file(url\_cfg, 'yolov3.cfg')

download\_file(url\_names, 'coco.names')

pip install opencv-python-headless

import cv2

import numpy as np

# Định nghĩa các hàm cần thiết

def load\_yolo():

# Đường dẫn tới tệp cấu hình và trọng số YOLO

weightsPath = "yolov3.weights"

configPath = "yolov3.cfg"

labelsPath = "coco.names"

# Tải trọng số và cấu hình để tạo mô hình

net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)

# Tải tên các lớp

with open(labelsPath, "r") as f:

classes = [line.strip() for line in f.readlines()]

return net, classes

def detect\_objects(img, net, output\_layers):

blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

net.setInput(blob)

outputs = net.forward(output\_layers)

return outputs

def get\_boxes(outputs, height, width, conf\_threshold=0.5):

boxes = []

confidences = []

class\_ids = []

for output in outputs:

for detection in output:

scores = detection[5:]

class\_id = np.argmax(scores)

confidence = scores[class\_id]

if confidence > conf\_threshold:

center\_x = int(detection[0] \* width)

center\_y = int(detection[1] \* height)

w = int(detection[2] \* width)

h = int(detection[3] \* height)

x = int(center\_x - w / 2)

y = int(center\_y - h / 2)

boxes.append([x, y, w, h])

confidences.append(float(confidence))

class\_ids.append(class\_id)

return boxes, confidences, class\_ids

def analyze\_traffic\_light\_color(img, box):

x, y, w, h = box

traffic\_light\_img = img[y:y+h, x:x+w]

hsv\_img = cv2.cvtColor(traffic\_light\_img, cv2.COLOR\_BGR2HSV)

# Định nghĩa các khoảng màu HSV

red1\_lower = np.array([0, 70, 50])

red1\_upper = np.array([10, 255, 255])

red2\_lower = np.array([170, 70, 50])

red2\_upper = np.array([180, 255, 255])

green\_lower = np.array([50, 70, 50])

green\_upper = np.array([90, 255, 255])

yellow\_lower = np.array([20, 100, 100])

yellow\_upper = np.array([30, 255, 255])

# Tạo mask cho mỗi màu

red\_mask1 = cv2.inRange(hsv\_img, red1\_lower, red1\_upper)

red\_mask2 = cv2.inRange(hsv\_img, red2\_lower, red2\_upper)

red\_mask = cv2.bitwise\_or(red\_mask1, red\_mask2)

green\_mask = cv2.inRange(hsv\_img, green\_lower, green\_upper)

yellow\_mask = cv2.inRange(hsv\_img, yellow\_lower, yellow\_upper)

# Tính tổng diện tích phát hiện màu sắc

if np.sum(red\_mask) > np.sum(green\_mask) and np.sum(red\_mask) > np.sum(yellow\_mask):

return "Red"

elif np.sum(green\_mask) > np.sum(red\_mask) and np.sum(green\_mask) > np.sum(yellow\_mask):

return "Green"

elif np.sum(yellow\_mask) > np.sum(red\_mask) and np.sum(yellow\_mask) > np.sum(green\_mask):

return "Yellow"

else:

return "Unknown"

def process\_image(img\_path):

model, classes = load\_yolo()

image = cv2.imread(img\_path)

if image is None:

print(f"Failed to load image: {img\_path}")

return None

height, width, channels = image.shape

layer\_names = model.getLayerNames()

output\_layers = [layer\_names[i - 1] for i in model.getUnconnectedOutLayers().flatten()]

outputs = detect\_objects(image, model, output\_layers)

boxes, confidences, class\_ids = get\_boxes(outputs, height, width)

color\_detected = draw\_labels(boxes, confidences, class\_ids, classes, image)

return color\_detected

def draw\_labels(boxes, confidences, class\_ids, classes, img):

traffic\_light\_color = None

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

for i in indexes.flatten():

x, y, w, h = boxes[i]

label = str(classes[class\_ids[i]])

if label == "traffic light":

color = analyze\_traffic\_light\_color(img, (x, y, w, h))

if color in ["Red", "Green", "Yellow"]:

traffic\_light\_color = color

return traffic\_light\_color

# Chạy phân tích cho mỗi ảnh

for img\_path in image\_paths:

color = process\_image(img\_path)

if color:

traffic\_light\_colors[img\_path] = color

else:

traffic\_light\_colors[img\_path] = "No traffic light detected"

# In ra kết quả

for img\_path, color in traffic\_light\_colors.items():

print(f"{img\_path}: {color}")

\*\*Creat stopline\*\*

pip install labelImg

import cv2

import matplotlib.pyplot as plt

def draw\_line(image\_path, start\_point, end\_point, color=(255, 0, 0), thickness=3):

# Đọc ảnh

image = cv2.imread(image\_path)

if image is None:

print("Failed to load image at", image\_path)

return

# Vẽ đường thẳng

cv2.line(image, start\_point, end\_point, color, thickness)

# Hiển thị ảnh

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.title('Image with Line')

plt.show()

# Lưu ảnh nếu muốn

# cv2.imwrite('path\_to\_save\_image.jpg', image)

# Các tham số mẫu cho hàm draw\_line

start\_point = (0, 820) # Điểm bắt đầu của đường thẳng

end\_point = (1610, 900) # Điểm kết thúc của đường thẳng

for image\_path in image\_paths:

draw\_line(image\_path, start\_point, end\_point, color=(0, 255, 0), thickness=6)

\*\*YOLOV3 detect car\*\*

pip install opencv-python-headless matplotlib

from google.colab.patches import cv2\_imshow

confidence\_threshold = 0.3 # Lower it from 0.5 to 0.3

nms\_threshold = 0.4 # Adjust this based on your specific needs

import cv2

import numpy as np

from google.colab.patches import cv2\_imshow # Import cv2\_imshow for Google Colab

def load\_yolo():

net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")

classes = []

with open("coco.names", "r") as f:

classes = [line.strip() for line in f.readlines()]

layer\_names = net.getLayerNames()

output\_layers = [layer\_names[i - 1] for i in net.getUnconnectedOutLayers().flatten()]

return net, classes, output\_layers

def detect\_objects(img, net, output\_layers):

height, width, \_ = img.shape

blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), swapRB=True, crop=False)

net.setInput(blob)

outputs = net.forward(output\_layers)

boxes = []

confidences = []

class\_ids = []

for output in outputs:

for detection in output:

scores = detection[5:]

class\_id = np.argmax(scores)

confidence = scores[class\_id]

if confidence > 0.5:

center\_x = int(detection[0] \* width)

center\_y = int(detection[1] \* height)

w = int(detection[2] \* width)

h = int(detection[3] \* height)

x = int(center\_x - w / 2)

y = int(center\_y - h / 2)

boxes.append([x, y, w, h])

confidences.append(float(confidence))

class\_ids.append(class\_id)

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

return boxes, indexes, class\_ids, confidences

def draw\_labels(boxes, indexes, class\_ids, classes, img):

if len(indexes) > 0:

indexes = indexes.flatten() # Ensure 'indexes' is a flat array

for i in indexes:

box = boxes[i]

x, y, w, h = box

label = str(classes[class\_ids[i]])

color = (0, 255, 0) # Color for the bounding box

cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)

cv2.putText(img, label, (x, y - 5), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, color, 2)

return img

# Load YOLO

net, classes, output\_layers = load\_yolo()

# Load and process each image

for image\_path in image\_paths:

image = cv2.imread(image\_path)

if image is None:

print("Image not found:", image\_path)

continue

height, width, \_ = image.shape

outputs = detect\_objects(image, net, output\_layers)

boxes, indexes, class\_ids, confidences = get\_box\_dimensions(outputs, height, width)

image = draw\_labels(boxes, indexes, class\_ids, classes, image)

cv2\_imshow(image) # Display the image with detections

# Print detection details

print("Detections:", len(indexes))

for i in indexes:

box = boxes[i]

print(f"Box {i}: {box}, Class: {classes[class\_ids[i]]}, Confidence: {confidences[i]:.2f}")

def get\_box\_dimensions(outputs, height, width, conf\_threshold=0.3): # Adjusted confidence threshold

boxes = []

confs = []

class\_ids = []

for output in outputs:

for detect in output:

scores = detect[5:]

class\_id = np.argmax(scores)

confidence = scores[class\_id]

if confidence > conf\_threshold:

center\_x = int(detect[0] \* width)

center\_y = int(detect[1] \* height)

w = int(detect[2] \* width)

h = int(detect[3] \* height)

x = int(center\_x - w / 2)

y = int(center\_y - h / 2)

boxes.append([x, y, w, h])

confs.append(float(confidence))

class\_ids.append(class\_id)

return boxes, confs, class\_ids

# Now adjust the NMS threshold in the draw\_labels function

def draw\_labels(boxes, confs, colors, class\_ids, classes, image, nms\_threshold=0.4): # Adjusted NMS threshold

indexes = cv2.dnn.NMSBoxes(boxes, confs, conf\_threshold, nms\_threshold)

for i in indexes:

i = i[0]

box = boxes[i]

x, y, w, h = box

label = str(classes[class\_ids[i]])

color = colors[class\_ids[i]]

cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)

cv2.putText(image, label, (x, y + 30), cv2.FONT\_HERSHEY\_PLAIN, 1, color, 2)

return image

import cv2

import numpy as np

from google.colab.patches import cv2\_imshow # Import cv2\_imshow

def load\_yolo():

net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")

classes = []

with open("coco.names", "r") as f:

classes = [line.strip() for line in f.readlines()]

layer\_names = net.getLayerNames()

# Flatten the layer indices to handle them as a simple list

output\_layers = [layer\_names[i - 1] for i in net.getUnconnectedOutLayers().flatten()]

return net, classes, output\_layers

def detect\_objects(image, net, output\_layers):

blob = cv2.dnn.blobFromImage(image, scalefactor=0.00392, size=(416, 416), mean=(0, 0, 0), swapRB=True, crop=False)

net.setInput(blob)

outputs = net.forward(output\_layers)

return outputs

def get\_box\_dimensions(outputs, height, width):

boxes = []

confs = []

class\_ids = []

for output in outputs:

for detect in output:

scores = detect[5:]

class\_id = np.argmax(scores)

confidence = scores[class\_id]

if confidence > 0.5:

center\_x = int(detect[0] \* width)

center\_y = int(detect[1] \* height)

w = int(detect[2] \* width)

h = int(detect[3] \* height)

x = int(center\_x - w / 2)

y = int(center\_y - h / 2)

boxes.append([x, y, w, h])

confs.append(float(confidence))

class\_ids.append(class\_id)

return boxes, confs, class\_ids

def draw\_labels(boxes, confs, colors, class\_ids, classes, image):

indexes = cv2.dnn.NMSBoxes(boxes, confs, 0.5, 0.4)

for i in indexes:

i = i[0]

box = boxes[i]

x, y, w, h = box

label = str(classes[class\_ids[i]])

color = colors[class\_ids[i]]

cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)

cv2.putText(image, label, (x, y + 30), cv2.FONT\_HERSHEY\_PLAIN, 1, color, 2)

return image

# Load YOLO

net, classes, output\_layers = load\_yolo()

colors = np.random.uniform(0, 255, size=(len(classes), 3))

for image\_path in image\_paths:

image = cv2.imread(image\_path)

if image is None:

print("Image not found:", image\_path)

continue

height, width, \_ = image.shape

outputs = detect\_objects(image, net, output\_layers)

boxes, confs, class\_ids = get\_box\_dimensions(outputs, height, width)

image = draw\_labels(boxes, confs, colors, class\_ids, classes, image)

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)) # Convert BGR to RGB

plt.axis('off') # Hide axis

plt.show()

def extract\_license\_plate(frame, mask\_line):

# Convert the image to grayscale (Haar cascades are typically trained on grayscale images)

gray = cv2.cvtColor(mask\_line, cv2.COLOR\_BGR2GRAY)

# Apply CLAHE to equalize the histogram

clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))

gray = clahe.apply(gray)

# Erode the image using a 2x2 kernel to remove noise

kernel = np.ones((2, 2), np.uint8)

gray = cv2.erode(gray, kernel, iterations=1)

# Find the bounding box of non-black pixels

non\_black\_points = cv2.findNonZero(gray)

x, y, w, h = cv2.boundingRect(non\_black\_points)

# Calculate the new width of the bounding box, excluding 30% on the right side

w = int(w \* 0.7)

# Crop the image to the bounding box

cropped\_gray = gray[y:y+h, x:x+w]

# Detect license plates in the image (this returns a list of rectangles)

license\_plates = license\_plate\_cascade.detectMultiScale(cropped\_gray, scaleFactor=1.07, minNeighbors=15, minSize=(20, 20))

# List to hold cropped license plate images

license\_plate\_images = []

# Loop over the license plates

for (x\_plate, y\_plate, w\_plate, h\_plate) in license\_plates:

# Draw a rectangle around the license plate in the original frame (here you need the original coordinates)

cv2.rectangle(frame, (x\_plate + x, y\_plate + y), (x\_plate + x + w\_plate, y\_plate + y + h\_plate), (0, 255, 0), 3)

# Crop the license plate and append it to the list (here x\_plate and y\_plate are relative to cropped\_gray)

license\_plate\_image = cropped\_gray[y\_plate:y\_plate+h\_plate, x\_plate:x\_plate+w\_plate]

license\_plate\_images.append(license\_plate\_image)

return frame, license\_plate\_images

def apply\_ocr\_to\_image(license\_plate\_image):

# Threshold the image

\_, img = cv2.threshold(license\_plate\_image, 120, 255, cv2.THRESH\_BINARY)

# Convert OpenCV image format to PIL Image format for pytesseract

pil\_img = Image.fromarray(img)

# Use pytesseract to extract text from the image

full\_text = pytesseract.image\_to\_string(pil\_img, config='--psm 6')

return full\_text.strip() # Removing any extra white spaces from the ends

def draw\_penalized\_text(frame):

# Set font, scale, thickness, and color

font = cv2.FONT\_HERSHEY\_TRIPLEX

font\_scale = 1

font\_thickness = 2

color = (255, 255, 255) # White color

# Initial position for Y-coordinate

y\_pos = 180

# Put title on the frame

cv2.putText(frame, 'Fined license plates:', (25, y\_pos), font, font\_scale, color, font\_thickness)

# Update Y-coordinate position

y\_pos += 80

# Loop through all fined license plates

for text in penalized\_texts:

# Add fined license plate text on the frame

cv2.putText(frame, '-> '+text, (40, y\_pos), font, font\_scale, color, font\_thickness)

# Update Y-coordinate for next license plate

y\_pos += 60

def create\_database\_and\_table(host, user, password, database):

try:

# Create a connection

connection = mysql.connector.connect(

host = host,

user = user,

password = password

)

if connection.is\_connected():

# Create a new database cursor

cursor = connection.cursor()

# Create a new database using the provided name

cursor.execute(f"CREATE DATABASE IF NOT EXISTS {database}")

print(f"Database {database} created successfully!")

# Use the newly created database

cursor.execute(f"USE {database}")

# Create a new table

cursor.execute("""

CREATE TABLE IF NOT EXISTS license\_plates (

id INT AUTO\_INCREMENT PRIMARY KEY,

plate\_number VARCHAR(255) NOT NULL UNIQUE,

violation\_count INT DEFAULT 1

)

""")

print("Table created successfully!")

cursor.close()

except Error as e:

print("Error while connecting to MySQL", e)

finally:

if connection.is\_connected():

connection.close()

def update\_database\_with\_violation(plate\_number, host, user, password, database):

try:

connection = mysql.connector.connect(

host = host,

user = user,

password = password,

database = database

)

if connection.is\_connected():

cursor = connection.cursor()

# Check if the license plate already exists in the table

cursor.execute(f"SELECT violation\_count FROM license\_plates WHERE plate\_number='{plate\_number}'")

result = cursor.fetchone()

if result:

# Increment violation\_count by 1 if plate\_number already exists

cursor.execute(f"UPDATE license\_plates SET violation\_count=violation\_count+1 WHERE plate\_number='{plate\_number}'")

else:

# Insert a new record if plate\_number doesn't exist

cursor.execute(f"INSERT INTO license\_plates (plate\_number) VALUES ('{plate\_number}')")

connection.commit()

cursor.close()

except Error as e:

print("Error while connecting to MySQL", e)

finally:

if connection.is\_connected():

connection.close()

def print\_all\_violations(host, user, password, database):

try:

connection = mysql.connector.connect(

host = host,

user = user,

password = password,

database = database

)

if connection.is\_connected():

cursor = connection.cursor()

# Fetch all violations from the database

cursor.execute("SELECT plate\_number, violation\_count FROM license\_plates ORDER BY violation\_count DESC")

result = cursor.fetchall()

print("\n")

print("-"\*66)

print("\nAll Registered Traffic Violations in the Database:\n")

for record in result:

print(f"Plate Number: {record[0]}, Violations: {record[1]}")

cursor.close()

except Error as e:

print("Error while connecting to MySQL", e)

finally:

if connection.is\_connected():

connection.close()

def clear\_license\_plates(host, user, password, database):

try:

connection = mysql.connector.connect(

host = host,

user = user,

password = password,

database = database

)

if connection.is\_connected():

cursor = connection.cursor()

# Delete all records from the table

cursor.execute("DELETE FROM license\_plates")

connection.commit()

cursor.close()

except Error as e:

print("Error while connecting to MySQL", e)

finally:

if connection.is\_connected():

connection.close()

def main():

# Ensure the database and table exist

# create\_database\_and\_table(DB\_HOST, DB\_USER, DB\_PASSWORD, DB\_NAME)

# Clear the license plates from the previous run. (Comment out this line if desired!)

# clear\_license\_plates(DB\_HOST, DB\_USER, DB\_PASSWORD, DB\_NAME)

# Open the video file

vid = cv2.VideoCapture('/kaggle/input/license-plate-recognition-for-red-light-violation/traffic\_video\_modified.mp4')

# Create detector object

detector = LineDetector()

# Loop through each frame in the video

while True:

# Read frame

ret, frame = vid.read()

# Break if frame is not returned

if not ret:

break

# Assuming rect is the rectangle where the traffic light is located

rect = (1700, 40, 100, 250)

# Detect traffic light color

frame, color = detect\_traffic\_light\_color(frame, rect)

# Detect white line

frame, mask\_line = detector.detect\_white\_line(frame, color)

# Process the frame if the light is red

if color == 'red':

# Extract license plate

frame, license\_plate\_images = extract\_license\_plate(frame, mask\_line)

# Process each detected license plate

for license\_plate\_image in license\_plate\_images:

# Apply OCR to the license plate image

text = apply\_ocr\_to\_image(license\_plate\_image)

# Add the detected license plate to the list if it matches the pattern and is not already in the list

if text is not None and re.match("^[A-Z]{2}\s[0-9]{3,4}$", text) and text not in penalized\_texts:

penalized\_texts.append(text)

print(f"\nFined license plate: {text}")

# Plot the license plate image

plt.figure()

plt.imshow(license\_plate\_image, cmap='gray')

plt.axis('off')

plt.show()

# Update the database with the license plate violation

# update\_database\_with\_violation(text, DB\_HOST, DB\_USER, DB\_PASSWORD, DB\_NAME)

# Draw the penalized text onto the frame if there is any

if penalized\_texts:

draw\_penalized\_text(frame)

# Display the frame

# cv2\_imshow('frame', frame)

cv2\_imshow(frame)

# Break if ESC key is pressed (uncomment the following line when running on a local system with GUI support)

if cv2.waitKey(1) == 27:

break

# Release the video

vid.release()

# Close all OpenCV windows (uncomment the following line when running on a local system with GUI support)

cv2.destroyAllWindows()

# Print all the violations from the database

# print\_all\_violations(DB\_HOST, DB\_USER, DB\_PASSWORD, DB\_NAME)

%%capture

# Download the trained Haar Cascade from the GitHub repository

url = "https://raw.githubusercontent.com/FarzadNekouee/Traffic-Violation-Detection/master/haarcascade\_russian\_plate\_number.xml"

response = requests.get(url)

with open('haarcascade\_russian\_plate\_number.xml', 'wb') as file:

file.write(response.content)

# Load the trained Haar Cascade

license\_plate\_cascade = cv2.CascadeClassifier('haarcascade\_russian\_plate\_number.xml')

# Create a list to store unique penalized license plate texts

penalized\_texts = []

\*\*EXTRACT TIME\*\*

from google.colab.patches import cv2\_imshow

!pip install pytesseract

!sudo apt install tesseract-ocr

try:

from PIL import Image

except ImportError:

import Image

import cv2

import pytesseract

!which tesseract

pytesseract.pytesseract.tesseract\_cmd = (

r'/usr/bin/tesseract'

)

import os

os.chdir(project\_folder)

image\_paths = ('2.png', '4.png', '3.png', '5.png', '1.png', '6.png', '18.png', '13.png', '14.png', '12.png', '15.png', '10.png', '17.png', '8.png', '9.png', '16.png', '11.png', '7.png', '22.png', '31.png', '25.png', '23.png', '24.png', '28.png', '20.png', '30.png', '29.png', '19.png', '27.png', '21.png', '26.png', '32.png')

img\_cv = cv2.imread(image\_paths[0])

cv2\_imshow(img\_cv)

img\_cv = cv2.imread(image\_paths[0])

# Cropping an image

cropped\_image = img\_cv[15:65, 0:530]

# Display cropped image

cv2\_imshow(cropped\_image)

import re

pattern=r'(\d{4}\-\d{2}\-\d{2}\s\d{2}\:\d{2}\:\d{2})'

# Example of adding any additional options

custom\_oem\_psm\_config = r'--oem 3 --psm 6'

def extract\_text(file\_path):

img\_cv = cv2.imread(file\_path)

# Cropping an image

cropped\_image = img\_cv[15:65, 0:530]

text=pytesseract.image\_to\_string(cropped\_image, config=custom\_oem\_psm\_config, lang = 'eng')

print(text)

if match := re.search(pattern, text, re.IGNORECASE):

time\_str = match.group(1)

else:

time\_str = ''

return time\_str

import pandas as pd

pd.DataFrame({'FILE': image\_paths})\

.assign(TIME=lambda df: df['FILE'].apply(extract\_text))