chrisheimbuch /
Traffic_Vehicle_Real_Time_Detection

Code Issues Pull requests Actions Projects Wiki Security In

Traffic_Vehicle_Real_Time_Detection / source / app.py

chrisheimbuch clean up of code e753caa · yesterday

175 lines (132 loc) · 7.08 KB

Code Blame Raw

```python
#Imports to work on the backend with Flask.
from flask import Flask, render_template, request, redirect, url_for, Response, make_response
import cv2
import numpy as np
from ultralytics import YOLO
import time
import os

#Initialize flask
app = Flask(__name__)

#Note: Update this path to where ever the best.pt file is saved on your computer directory.
model_path = r"C:\Users\chris\Desktop\capstone project\Traffic_Vehicle_Real_Time_Detection\sou

#YOLO model instance
yolo_model = YOLO(model_path)

#List of class names corresponding to my YOLO model
class_names = ['bus', 'car', 'motorbike', 'threewheel', 'truck', 'van']

#Function to process frames and return them with YOLO detection results
def gen_frames():
    #Open the webcam
    cap = cv2.VideoCapture(0)

    while True:
        success, frame = cap.read()
        if not success:
            break

        #Process the frame with YOLO model
        results = yolo_model.predict(source=frame, save=False)

        for result in results:
            boxes = result.boxes.xyxy  #This is bounding box coordinates
            labels = result.boxes.cls  #This is for class labels
            confidences = result.boxes.conf  #This will display Confidence scores

            for box, label, confidence in zip(boxes, labels, confidences):
                #Get the class name based on the label index
                class_name = class_names[int(label)] if int(label) < len(class_names) else f"

                x1, y1, x2, y2 = map(int, box)

                #Draw bounding box with thicker lines
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)

                #Draw class name and confidence score
                cv2.putText(frame, f"{class_name} ({confidence:.2f})", (x1, y1 - 10),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)

        #Encode the frame in JPEG format and yield the result
        ret, buffer = cv2.imencode('.jpg', frame)
        frame = buffer.tobytes()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

    cap.release()

#This is a route for live detection.
@app.route('/webcam')
def webcam_feed():
    """Route to start the webcam feed and display it."""
    return Response(gen_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')

#Route for videos
@app.route('/video_detection', methods=['POST'])
def video_detection():
    file = request.files['video']  # Get the uploaded video
    video_path = 'static/uploaded_video.mp4'
    processed_video_path = 'static/processed_video.mp4'

    #Check if the processed video exists and delete it
    if os.path.exists(processed_video_path):
        os.remove(processed_video_path)

    #Save the uploaded video to disk
    file.save(video_path)

    #Open the video with OpenCV
    cap = cv2.VideoCapture(video_path)

    #Define the codec and create a VideoWriter object to save the output video as MP4
    fourcc = cv2.VideoWriter_fourcc(*'H264')
    out = cv2.VideoWriter(processed_video_path, fourcc, 20.0, (int(cap.get(3)), int(cap.get(4

    while cap.isOpened():
        success, frame = cap.read()
        if not success:
            break

        #Process the frame with YOLO model
        results = yolo_model.predict(source=frame, save=False, device='cuda')

        for result in results:
            boxes = result.boxes.xyxy  #Bounding box coordinates
            labels = result.boxes.cls  #Class labels (indices)
            confidences = result.boxes.conf  #Confidence scores

            for box, label, confidence in zip(boxes, labels, confidences):
                class_name = class_names[int(label)] if int(label) < len(class_names) else f"
                x1, y1, x2, y2 = map(int, box)

                #Draw bounding box
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
                cv2.putText(frame, f"{class_name} ({confidence:.2f})",
                            (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)

        #Write the processed frame to the output video
        out.write(frame)

    cap.release()
    out.release()

    #Pass only the filename to the template, not the full path
    return render_template('index.html', video_path='processed_video.mp4')

#upload and process methods for images
@app.route('/', methods=['GET', 'POST'])
def upload_and_process():
    if request.method == 'POST':
        file = request.files['image']
        image = cv2.imdecode(np.frombuffer(file.read(), np.uint8), cv2.IMREAD_COLOR)

        #Process the image with YOLO model
        results = yolo_model.predict(source=image, save=False)

        #Initialize a list to store class names and confidence scores
        classifications = []

        #Draw the bounding boxes and labels on the image
        for result in results:
            boxes = result.boxes.xyxy  #Bounding box coordinates
            labels = result.boxes.cls  #Class labels (indices)
            confidences = result.boxes.conf  #Confidence scores

            for box, label, confidence in zip(boxes, labels, confidences):
                #Get the class name based on the label index
                class_name = class_names[int(label)] if int(label) < len(class_names) else f"

                #Append both the class name and confidence score to the list
                classifications.append({
                    'class': class_name.title(),
                    'confidence': round(float(confidence) * 100)  #Convert to percentage and
                })

                x1, y1, x2, y2 = map(int, box)

                #Draw bounding box with thicker lines (thickness = 3)
                cv2.rectangle(image, (x1, y1), (x2, y2), (0, 255, 0), 10)

                #Draw larger class name and confidence score (font scale = 1.2, thickness = 3
                cv2.putText(image, f"{class_name} ({confidence:.2f})",
                            (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 2.0, (0, 0, 255), 3)

        #Resize the image to make it larger (e.g., 1.5x the original size)
        image = cv2.resize(image, (int(image.shape[1] * 1.5), int(image.shape[0] * 1.5)))

        #Save the processed image
        processed_image_path = 'static/processed_image.jpg'
        cv2.imwrite(processed_image_path, image)

        #Return the page with the processed image and classification details, including the d
        return render_template('index.html', image_path=processed_image_path, classifications

    #Default GET request just renders the page for upload
    return render_template('index.html')

#Add a route to handle re-uploading
@app.route('/reupload')
def reupload():
    return redirect(url_for('upload_and_process'))

if __name__ == '__main__':
    app.run(debug=True)
```