AI-Based Traffic Monitoring System

Course Title: Artificial Intelligence &ExpertSystem

Course Code: CT-361

GROUP MEMBERS:

ABDULLAH KHALID (CR-22027)

SHEHERYAR AMIR (CR-22008)

HAMZA RIAZ KHAN (CR-22031)

About Project:

Our "AI Based Traffic Monitoring System" is an innovative solution designed to enhance urban traffic management using advanced artificial intelligence techniques. We are leveraging YOLOv8, a state-of-the-art object detection model, to detect and track vehicles such as cars, motorcycles, buses, trucks, and bicycles in both images and videos, providing real-time visual insights into traffic conditions. Additionally, we utilize YAMNet, a pre-trained audio classification model from TensorFlow Hub, integrated with libraries like librosa and moviepy, to analyze audio extracted from videos and identify critical events such as ambulance sirens, traffic jams, accidents, and normal traffic conditions. By combining these technologies in a single system, implemented through Python scripts like detect_yolo_image.py, detect_yolo_video.py, and sound_analysis.py, we aim to assist traffic authorities and emergency services with accurate, automated monitoring, improving safety and efficiency in urban environments.

Problem statement:

Urban traffic management poses a significant challenge due to the high volume of vehicles, unpredictable congestion, and the need for rapid emergency response. The complexity increases with the difficulty of processing real-time visual and auditory data to detect incidents like accidents or ambulances amidst normal traffic noise. Traditional methods rely on manual monitoring, which is inefficient and error-prone, necessitating an automated, AI-driven solution to enhance traffic flow and safety in densely populated cities like Karachi.

Justification, Complexity, and Solution

- **Why the Problem Is Complex**:
  - **Urban Scale**: Managing traffic in cities involves thousands of vehicles, diverse road conditions, and real-time changes.
  - **Multi-Sensory Data**: Combining visual (vehicle detection) and auditory (sound analysis) data requires sophisticated AI to handle noise, occlusion, and variability.
  - **Emergency Response**: Detecting ambulances or accidents instantly amidst normal traffic is challenging due to overlapping sounds and fast-moving objects.
- **How Your Project Solves It**:
  - **Vehicle Detection**: YOLOv8 provides fast, accurate identification of vehicles (cars, buses, etc.) in images and videos, enabling traffic volume monitoring.
  - **Sound Analysis**: YAMNet classifies critical events (e.g., ambulance sirens, crashes) from audio, alerting authorities to emergencies or congestion.
  - **Integration**: The final_project.py combines both, offering a streamlined system to monitor and respond to traffic conditions, improving safety and efficiency.

- **Justification**:
  - Your system automates traffic monitoring, reducing human error and response time. It's practical for cities like Karachi, where traffic chaos is common, and can aid emergency services with real-time insights.

Image and Video Detection:

I. Description

VEHICLE DETECTION IN IMAGE USING YOLO V8:

- **YOLO (You Only Look Once)**: A state-of-the-art, real-time object detection algorithm that predicts bounding boxes and class probabilities directly from an image in a single evaluation.
- **YOLOv8:** The latest iteration of the YOLO family, providing higher accuracy and performance.
- **confidence:** A measure of the model's certainty about a detected object's classification.
- **model.names**: Maps the class IDs to their corresponding names (e.g., car, truck).
- **cv2.putText()**: Adds text labels to the image, such as object class and confidence score.

VEHICLE DETECTION IN VIDEO USING YOLO V8:

- **distance.euclidean**: Computes the Euclidean distance between two points for tracking vehicles across frames.
- **defaultdict**: A dictionary-like structure that initializes missing keys with a default value, used here for tracking vehicle counts by type.
- **distance_threshold**: A maximum distance to associate a detected object with an existing tracked vehicle.
- **Video Tracking:** Assigning unique IDs to detected vehicles and persisting their information across video frames**.**

II. Code & Output

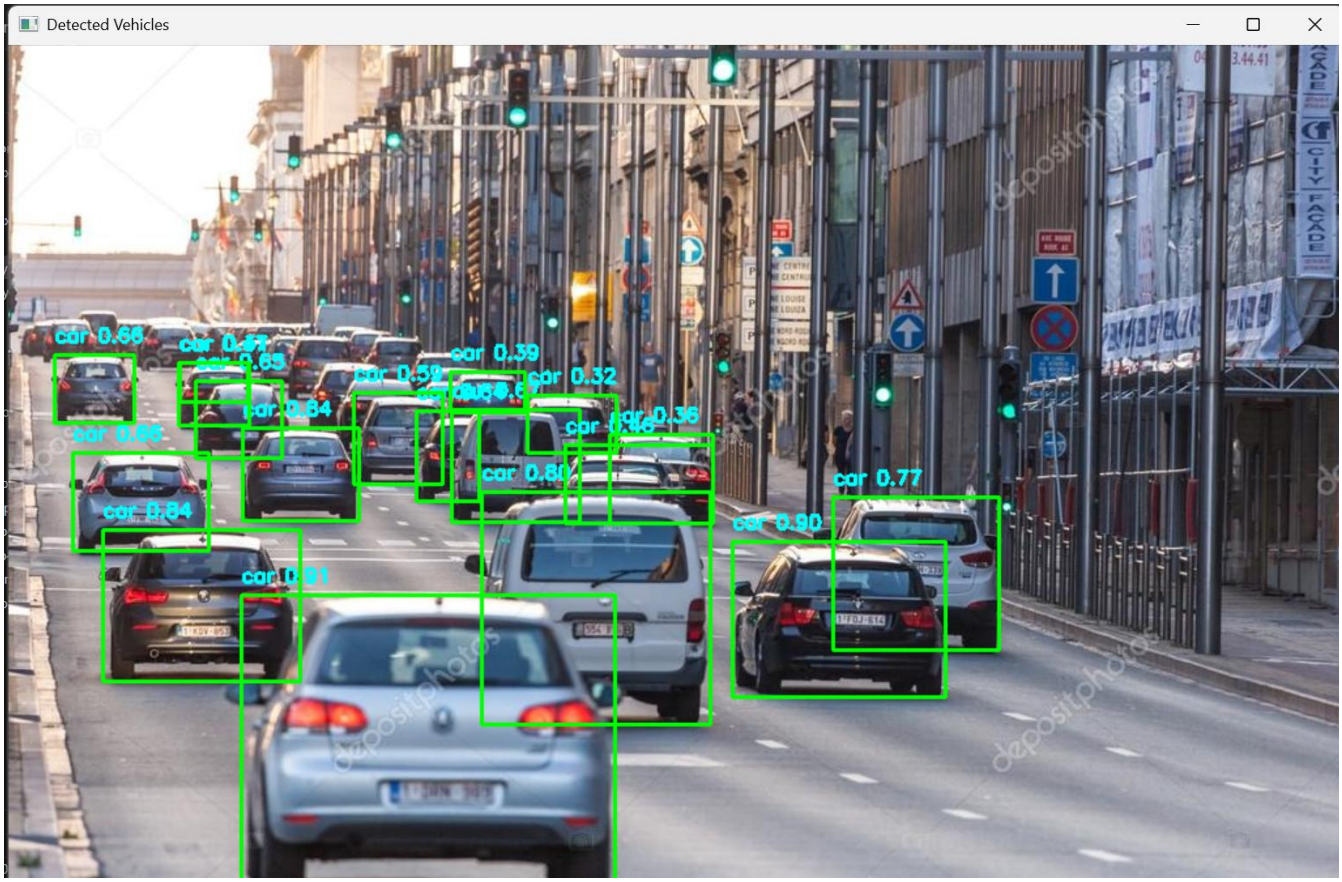
VEHICLE DETECTION IN IMAGE USING YOLO V8:

CODE:

```
import cv2
from ultralytics import YOLO

# Load the YOLOv8 model
model = YOLO('yolov8n.pt')

# Load the image
image_path = 'morecars.JPG'
image = cv2.imread(image_path)

# Detect objects
results = model(image)

# List of vehicle classes YOLOv8 can detect (based on COCO dataset)
vehicle_classes = ['car', 'motorcycle', 'bus', 'truck', 'bicycle']

# Draw boxes and labels for all vehicles
for result in results[0].boxes.data:
    x1, y1, x2, y2, confidence, class_id = result.tolist()
    if confidence > 0:  # Only show confident detections
        class_name = model.names[int(class_id)]
        if class_name in vehicle_classes:  # Check if it's a vehicle
            cv2.rectangle(image, (int(x1), int(y1)), (int(x2), int(y2)), (0, 255, 0),
2)
            cv2.putText(image, f'{class_name} {confidence:.2f}', (int(x1), int(y1) -
10),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 0), 2)


# Optional: Count vehicle types
vehicle_counts = {}
for result in results[0].boxes.data:
    x1, y1, x2, y2, confidence, class_id = result.tolist()
    if confidence > 0:
        class_name = model.names[int(class_id)]
        if class_name in vehicle_classes:
```

```
            vehicle_counts[class_name] = vehicle_counts.get(class_name, 0) + 1

# Print counts
print("-----------------Vehicle Counts--------------------")
for vehicle, count in vehicle_counts.items():
    print(f"{vehicle}: {count}")

# Save and show the result
cv2.imwrite('output_img.JPG', image)
print("Saved result to output_image1.JPG")
cv2.imshow('Detected Vehicles', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



VEHICLE DETECTION IN VIDEO USING YOLO V8:

CODE:

```
import cv2
from ultralytics import YOLO

# Load the YOLOv8 model
model = YOLO('yolov8n.pt')

# Open the video file
video_path = 'video8-lahore.mp4'
cap = cv2.VideoCapture(video_path)

# Get video properties
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))

# Define the output video
out = cv2.VideoWriter('output_video8.mp4', cv2.VideoWriter_fourcc(*'mp4v'), fps,
(frame_width, frame_height))

# List of vehicle classes
vehicle_classes = ['car', 'motorcycle', 'bus', 'truck', 'bicycle']

# Process each frame
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Detect objects in the frame
    results = model(frame)

    # Draw boxes and labels for vehicles
    for result in results[0].boxes.data:
        x1, y1, x2, y2, confidence, class_id = result.tolist()
        if confidence > 0:
            class_name = model.names[int(class_id)]
            if class_name in vehicle_classes:
```

```
                cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)), (0, 255,
0), 2)
                cv2.putText(frame, f'{class_name} {confidence:.2f}', (int(x1), int(y1)
- 10),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 0), 2)

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

# Release resources
cap.release()
out.release()
cv2.destroyAllWindows()
print("Saved result to output_video8.mp4")
```

OUTPUT: *Saved to file*

III. Justification

VEHICLE DETECTION IN IMAGE USING YOLO V8:

The YOLOv8 model enhances object detection by leveraging deep learning for greater accuracy and real-time performance. This code applies the model to detect vehicles in an image, filters results by confidence, and visually annotates detections with bounding boxes and labels. YOLO's efficiency and precision make it an advanced alternative to Haar Cascade.

VEHICLE DETECTION IN VIDEO USING YOLO V8:

This code enhances vehicle detection by implementing object tracking, allowing persistent monitoring and analysis of vehicle flow over time. It combines YOLOv8's detection capabilities with tracking logic to identify unique vehicles, count their types, and provide insights like total vehicle count. Tracking also minimizes redundant detections and aids in real-time video-based traffic monitoring.

Sound Analysis:

I. Description

- **librosa**: Python library for analyzing audio, used here to preprocess audio files into waveforms.
- **moviepy.editor**: Python library for editing and processing video files, used to extract audio from videos.
- **YAMNet**: A pre-trained audio classification model from TensorFlow Hub that identifies sounds from a predefined set of classes.
- **IPython.display**: Provides tools for displaying rich media (e.g., video) inline in environments like Jupyter notebooks or Colab.
- **Relevant Class Indices**: Specific class IDs from YAMNet's output corresponding to "Traffic Jam," "Ambulance," and "Accident."

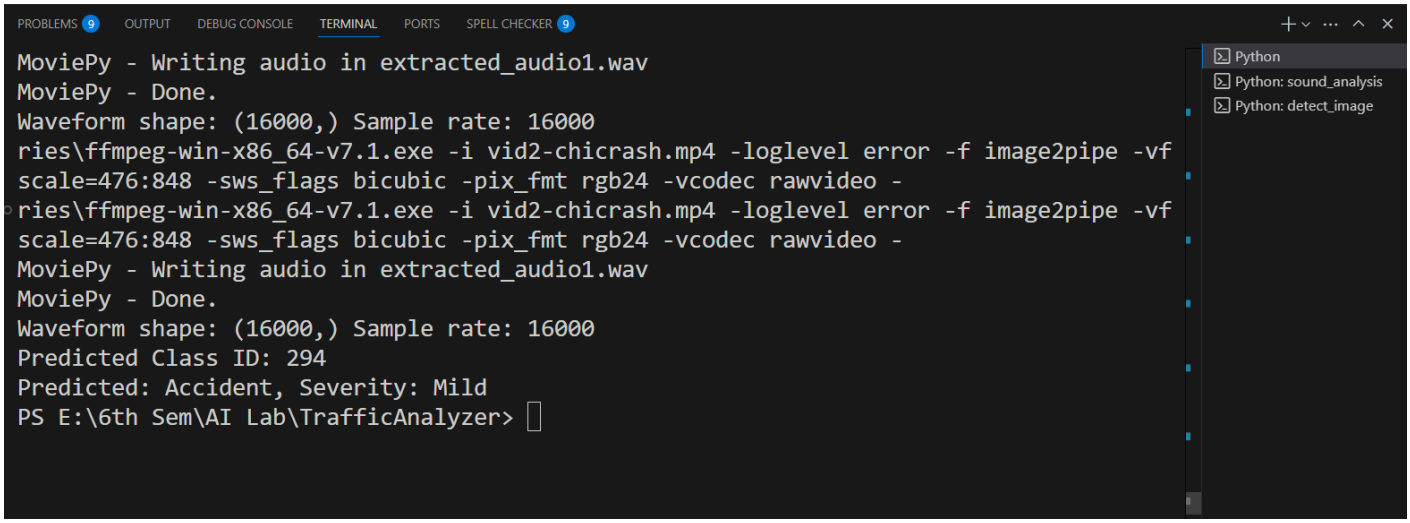
II. Code & Output

CODE:

```
from moviepy.video.io.VideoFileClip import VideoFileClip
import librosa
import numpy as np
import tensorflow_hub as hub

# Load YAMNet model
model = hub.load('https://tfhub.dev/google/yamnet/1')
print("Model loaded:", model)

# Preprocess audio
def preprocess_audio(file_path):
    waveform, sr = librosa.load(file_path, sr=16000, mono=True)
    waveform = librosa.util.fix_length(waveform, size=16000)
    print("Waveform shape:", waveform.shape, "Sample rate:", sr)
    return waveform

# Classify audio
def classify_audio(file_path):
```

```
    waveform = preprocess_audio(file_path)
    scores, _, _ = model(waveform)
    predicted_idx = np.argmax(scores, axis=-1)[0]
    intensity = np.mean(np.abs(waveform))
    print("Predicted Class ID:", predicted_idx)
    if predicted_idx == 321:  # Traffic noise (rush hour)
        level = "Low" if intensity < 0.1 else "Moderate" if intensity < 0.5 else
"Severe"
        print(f"Predicted: Traffic Jam, Blockage: {level}")
    elif predicted_idx == 316:  # Emergency (ambulance sound)
        level = "Low" if intensity < 0.1 else "Medium" if intensity < 0.5 else "High"
        print(f"Predicted: Ambulance, Urgency: {level}")
    elif predicted_idx == 294:  # Vehicle sound (vehicles crashing)
        level = "Mild" if intensity < 0.3 else "Severe"
        print(f"Predicted: Accident, Severity: {level}")
    elif predicted_idx == 494:  # Silence (normal traffic)
        print("Predicted: Normal Traffic (Silence)")
    else:
        print("No relevant sound detected.")

# Extract audio from video and classify
video = VideoFileClip('vid2-chicrash.mp4')
audio = video.audio
audio.write_audiofile('extracted_audio1.wav')
classify_audio('extracted_audio1.wav')
```

OUTPUT:



III. Justification

This code processes video to extract audio, then classifies it into specific categories using the YAMNet model. It ensures seamless integration by automating steps like audio extraction, video playback, and classification. This setup aids in sound-based event detection for traffic

Assumptions & Limitations:

- The system assumes reliable internet for loading the YAMNet model.
- Video inputs are assumed to have clear audio and visual data for accurate detection.
- YOLOv8 and YAMNet are pre-trained on relevant datasets (e.g., COCO, AudioSet) applicable to urban traffic.
- Graph-based traffic optimization (route planning) is not implemented due to time constraints."
- "YAMNet may misclassify sounds in noisy environments or if audio quality is poor."
- "YOLOv8 detection accuracy depends on lighting and camera angles, potentially missing obscured vehicles."
- "The system is tested on limited datasets (e.g., specific videos) and may need adjustment for broader use.

END OF REPORT