OBJECT TRACKING PROJECT

1. Objective

The goal of this task is to develop an object tracking-based application. This application will:

1. Track objects in a video
2. Calculate and display specific metrics
3. Visualise tracking results
4. Export and display the output in an HTML window.

2. Approach

- **Detection**: Use YOLOv8 for object detection due to its speed and accuracy.
- **Tracking**: Match object centroids across frames to maintain consistent object IDs.
- **Visualization**:
  - Draw bounding boxes around detected objects.
  - Mark centroids for tracking.
  - Visualize object movement with trails.

3. Tech Stack

- **Programming Language**: Python
- **Object Detection Model** : YOLO (yolov8)
- **Libraries**:
  - **OpenCV**: For video processing and visualization.
  - **NumPy**: For mathematical operations.
  - **OS**: For handling file paths and system operations.
- **Output Formats**:
  - MP4 for processed video.
  - HTML for embedding and displaying results in a browser.

4. Implementation Details

1. YOLO Model Selection:

YOLO (You Only Look Once) is a real-time object detection algorithm developed by Joseph Redmon and Ali Farhadi in 2015. It is a single-stage object detector that uses a convolutional neural network (CNN) to predict the bounding boxes and class probabilities of objects in input images.

YOLOv8 is the latest iteration in the YOLO series of real-time object detectors, offering cutting-edge performance in terms of accuracy and speed.

In Yolov8 , we use PreTrained Model **Yolov8n.pt** to detect object in real time.

2. **Video Processing**:

- Read frames from the input video using OpenCV.
- Extract video properties like FPS, width, and height.

3. **Object Tracking**:

- Use a centroid-based approach to track objects across frames.
- Match objects based on proximity to the last known centroid.

4. **Visualization**:

- Draw bounding boxes, centroids, and trails for tracked objects.
- Label each object with its class name (e.g., "person").

5. **Metrics Calculation**:

- Count unique objects detected.
- Compute time spent in the video for each object.

6. **Output Generation**:

- Save the processed video.
- Create an HTML file with the embedded video for browser viewing.

5. CODING STEPS:

1. Import Packages:

```
import cv2
import numpy as np
from ultralytics import YOLO
from collections import defaultdict
import os
```

2. Import Model:

```
model = YOLO("yolov8n.pt")
```

3. Video Capture & its Properties:

```
cap = cv2.VideoCapture(input_video_path)
fps = int(cap.get(cv2.CAP_PROP_FPS))
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
```

```
input_video_path = "input_video.mp4"
output_video_path = "output_video.mp4
```

Output Video

```
fourcc = cv2.VideoWriter_fourcc(*"mp4v")
out = cv2.VideoWriter(output_video_path, fourcc, fps, (frame_width, frame_height))
```

4. Object tracking data

```
object_data = defaultdict(lambda: {"frames": [], "centroids": []})
next_object_id = 1
Object_id_mapping = {}
```

5. Function to calculate centroid of a bounding box

```
def calculate_centroid(box):
    x_min, y_min, x_max, y_max = box
    return int((x_min + x_max) / 2), int((y_min + y_max) / 2)
```

6. Collect Bounding boxes , Class IDs , Class Names

```
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
```

```
        break
results = model(frame)
   detections = results[0].boxes.xyxy.cpu().numpy()
    class_ids = results[0].boxes.cls.cpu().numpy().astype(int)
    class_names = [model.names[class_id] for class_id in class_ids]


        frame_data = []
            for det, class_name in zip(detections, class_names):
                x_min, y_min, x_max, y_max = map(int, det)
                centroid = calculate_centroid((x_min, y_min, x_max,
        y_max))
```

Match with existing objects

```
    matched = False
         for obj_id, data in object_data.items():
             if data["frames"] and np.linalg.norm(np.array(data["centroids"][-
    1]) - np.array(centroid)) < 50:  # Match threshold

    object_data[obj_id]["frames"].append(len(object_data[obj_id]["frames"])
    + 1)
                object_data[obj_id]["centroids"].append(centroid)
                frame_data.append((obj_id, (x_min, y_min, x_max, y_max),
    centroid, class_name))
                matched = True
```

```
            break
if not matched:
        # Assign new object ID
        global next_object_id
        object_data[next_object_id]["frames"].append(1)
        object_data[next_object_id]["centroids"].append(centroid)
        frame_data.append((next_object_id, (x_min, y_min, x_max, y_max), centroid, class_name))
        next_object_id += 1
```

7. Draw bounding boxes, centroids, trails, and labels

```
  for obj_id, box, centroid, class_name in frame_data:
    x_min, y_min, x_max, y_max = box
    cv2.rectangle(frame, (x_min, y_min), (x_max, y_max), (0, 255, 0), 2)
    cv2.circle(frame, centroid, 5, (0, 0, 255), -1)
    cv2.putText(frame, class_name, (x_min, y_min - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 255), 2)
```

```
    # Draw trail
    for i in range(1, len(object_data[obj_id]["centroids"])):
        cv2.line(frame, object_data[obj_id]["centroids"][i - 1], object_data[obj_id]["centroids"][i], (255, 0, 0), 2)
```

output video

```
    out.write(frame)
```

8. Export metrics

```
metrics = {
    "unique_object_ids": len(object_data),
    "time_spent_per_object": {
        obj_id: len(data["frames"]) / fps for obj_id, data in
object_data.items()
    },
}
```

9. Print metrics

```
print("Metrics:")
print(f"Total Unique Objects Detected:
{metrics['unique_object_ids']}")
for obj_id, time_spent in metrics["time_spent_per_object"].items():
    print(f"Object {obj_id}: {time_spent:.2f} seconds")
```

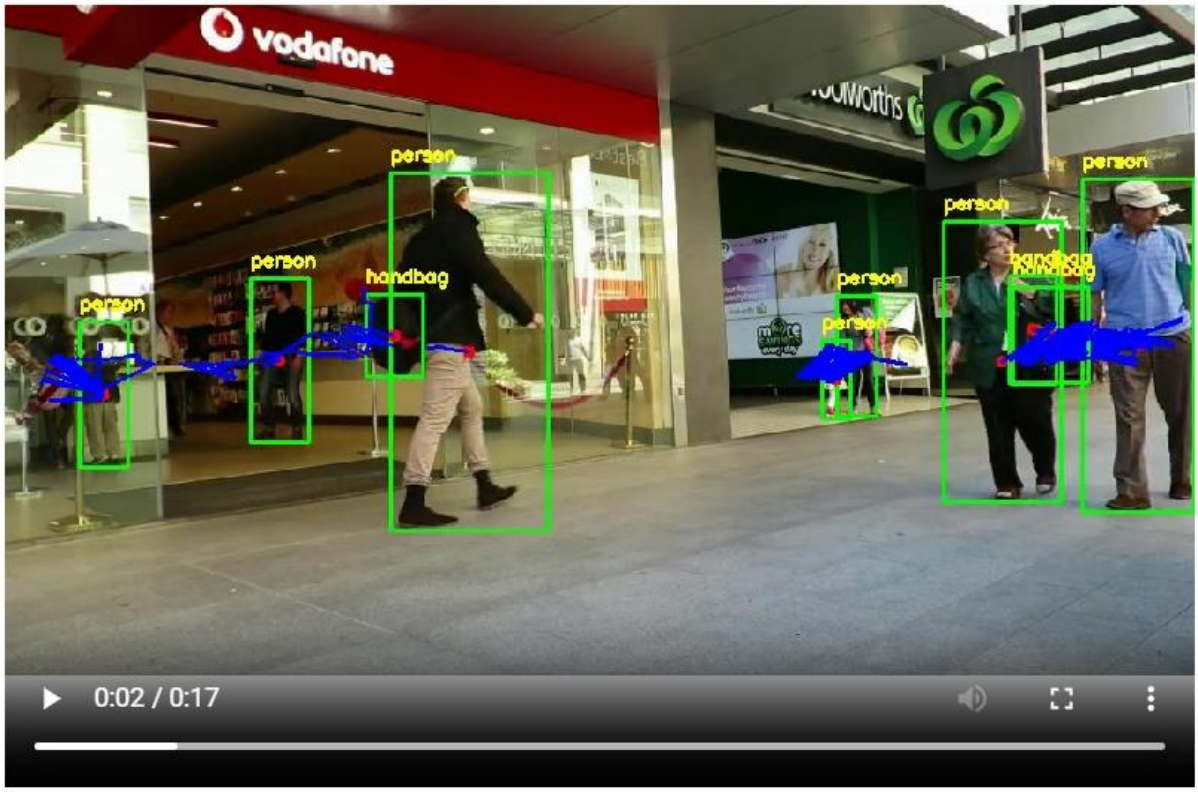
10. HTML for output

```
<!DOCTYPE html>
<html>
<head>
    <title>Object Tracking Output</title>
</head>
<body>
    <h1>Object Tracking Results</h1>
    <video controls width="640" height="480">
        <source src=" outputfile.mp4" type="video/mp4">
        Your browser does not support the video tag.
    </video>
</body>
</html>
```

HOSTED LINK ->**https://brilliant-gelato-2db36d.netlify.app/**

11. Output Screenshot

Object Tracking Results



```
0: 384x640 9 persons, 9.0ms
Speed: 4.4ms preprocess, 9.0ms inference, 1.2ms postprocess per image at shape (1, 3, 384, 640)
Metrics:
Total Unique Objects Detected: 45
Object 1: 10.60 seconds
Object 2: 34.50 seconds
Object 3: 20.50 seconds
Object 4: 9.20 seconds
Object 5: 14.93 seconds
Object 6: 7.27 seconds
Object 7: 11.73 seconds
Object 8: 13.17 seconds
Object 9: 5.40 seconds
Object 10: 14.13 seconds
Object 11: 14.60 seconds
Object 12: 4.47 seconds
```

12. Conculsion.

Thus, The Object Tracking Project is completed with the help of YOLO Object Detection Model.