PLAYER RE-IDENTIFICATION USING YOLOv11 & DeepSORT

Objective

In this project, I developed a robust player tracking system that leverages the power of YOLOv11 (Ultralytics) for real-time object detection and DeepSORT for multi-object tracking and re-identification. The primary goal was to assign unique and consistent IDs to each football player across video frames, even when players are briefly occluded or move across the field.

Tools and Technologies Used

- **YOLOv11 (Ultralytics)**: A state-of-the-art real-time object detection algorithm, used here for detecting players.
- **DeepSORT**: An advanced tracking algorithm that maintains identities across frames using appearance and motion cues.
- **OpenCV**: For video reading, writing, and drawing visual annotations.
- **NumPy**: Used for handling numerical operations and array-based computations.

Step-by-Step Implementation

1. Import Required Libraries

```
import cv2
import numpy as np
from ultralytics import YOLO
from deep_sort_realtime.deepsort_tracker import DeepSort
```

This step imports the necessary libraries that facilitate object detection, tracking, and video processing.

2. Load YOLOv11 Model and Input Video

```
model = YOLO("C:/Users/ASUS/Downloads/STEALTH MODE INTERNSHIP/best.pt")
cap = cv2.VideoCapture("C:/Users/ASUS/Downloads/STEALTH MODE INTERNSHIP/SAMPLE VIDEOS/15sec_input_720p.mp4")
```

Here, I load a custom-trained YOLOv11 model optimized for detecting football players. The input video is opened using OpenCV to prepare it for frame-by-frame processing.

3. Initialize the DeepSORT Tracker

```
tracker = DeepSort(max_age=30, n_init=3, max_cosine_distance=0.4)
```

I configure DeepSORT with hyperparameters that balance between responsiveness and stability. This helps to keep consistent IDs for players even when they temporarily disappear due to occlusion or camera movement.

- `max_age`: Frames to wait before a lost track is deleted.
- `n_init`: Minimum number of frames a new object must appear in before it's considered confirmed.
- `max_cosine_distance`: Appearance similarity threshold for tracking.

4. Set Up the Output Video Writer

```
fps = cap.get(cv2.CAP_PROP_FPS)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('output_reid_v2.mp4', fourcc, fps, (width, height))
```

The output video writer is configured to match the input video's frame rate and resolution. The result is saved in MP4 format with bounding box annotations and player IDs.

---

5. Main Loop for Detection and Tracking

```
frame_idx = 0
track_history = {}

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

    results = model(frame)
    detections = []

    for box in results[0].boxes:
        cls = int(box.cls[0])
        conf = float(box.conf[0])
        x1, y1, x2, y2 = map(int, box.xyxy[0])

        if cls == 0:  # Class 0 corresponds to players
            detections.append(([x1, y1, x2, y2], conf, "player"))

    tracks = tracker.update_tracks(detections, frame=frame)

    for track in tracks:
        if not track.is_confirmed():
```

```
            continue

        track_id = track.track_id
        l, t, r, b = map(int, track.to_ltrb())

        if track_id not in track_history:
            track_history[track_id] = []
        track_history[track_id].append((frame_idx, (l, t, r, b)))

        cv2.rectangle(frame, (l, t), (r, b), (0, 255, 0), 2)
        cv2.putText(frame, f'ID: {track_id}', (l, t - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)

    cv2.putText(frame, f'Frame: {frame_idx}', (10, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0, 0, 255), 2)

    out.write(frame)
    frame_idx += 1
```

This loop processes each frame from the video. The YOLOv11 model is used to detect all players, and only those detections with class `0` (players) are passed to the DeepSORT tracker.

The tracker assigns a persistent ID to each detected player. Bounding boxes and IDs are drawn on the frame, and the processed frame is written to the output video file.

A history of player positions is also stored, which could be used later for analysis such as heatmaps or player paths.

---

6. Release Resources and Save Output

```
cap.release()
out.release()
print("Processing complete. Output saved as 'output_reid_v2.mp4'")
```

Finally, I release the video resources and complete the export process.

[18]:
```
import cv2
import numpy as np
from ultralytics import YOLO
from deep_sort_realtime.deepsort_tracker import DeepSort
```

[19]:
```
model = YOLO("C:/Users/ASUS/Downloads/STEALTH MODE INTERNSHIP/best.pt")
cap = cv2.VideoCapture("C:/Users/ASUS/Downloads/STEALTH MODE INTERNSHIP/SAMPLE VIDEOS/15sec_input_720p.mp4")
```

[20]:
```
tracker = DeepSort(max_age=30, n_init=3, max_cosine_distance=0.4)
```

[21]:
```
fps = cap.get(cv2.CAP_PROP_FPS)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('output_reid_v2.mp4', fourcc, fps, (width, height))
```

[22]:
```
frame_idx = 0
track_history = {}

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

    results = model(frame)
    detections = []

    for box in results[0].boxes:
        cls = int(box.cls[0])
        conf = float(box.conf[0])
        x1, y1, x2, y2 = map(int, box.xyxy[0])

        if cls == 0:  # Class 0 corresponds to players
            detections.append(([x1, y1, x2, y2], conf, "player"))

    tracks = tracker.update_tracks(detections, frame=frame)

    for track in tracks:
        if not track.is_confirmed():
            continue

        track_id = track.track_id
        l, t, r, b = map(int, track.to_ltrb())

        if track_id not in track_history:
            track_history[track_id] = []
        track_history[track_id].append((frame_idx, (l, t, r, b)))

        cv2.rectangle(frame, (l, t), (r, b), (0, 255, 0), 2)
        cv2.putText(frame, f'ID: {track_id}', (l, t - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 0), 2)

    cv2.putText(frame, f'Frame: {frame_idx}', (10, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0, 0, 255), 2)

    out.write(frame)
    frame_idx += 1
```

[23]:
```
cap.release()
out.release()
print("Processing complete. Output saved as 'output_reid_v2.mp4'")
```

```
Processing complete. Output saved as 'output_reid_v2.mp4'
```

Output

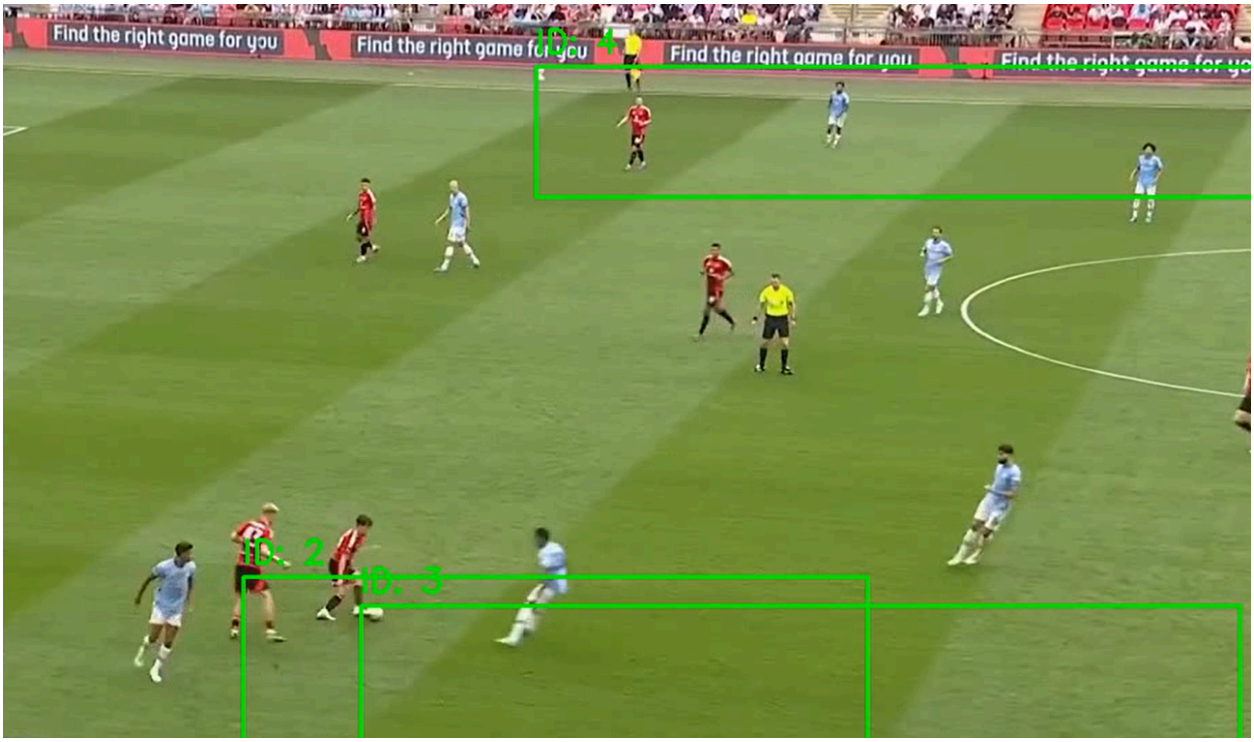
- **Output Video File:** `output_reid_v2.mp4`

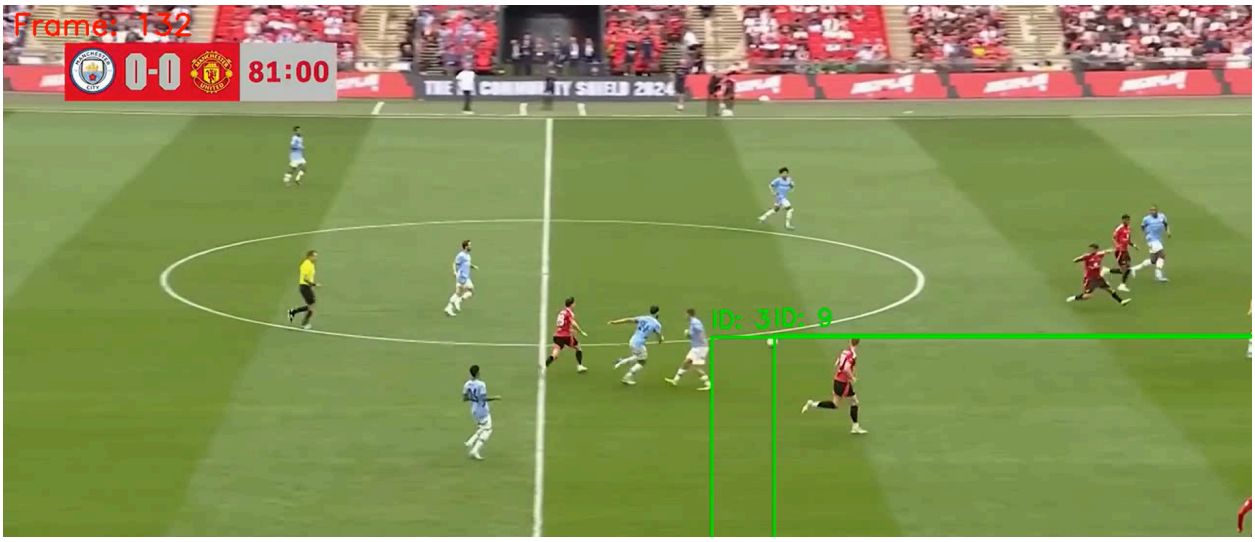
- Each football player is accurately detected and tracked across frames.
- IDs are consistently maintained even with brief occlusions or rapid movement.
- Frame numbers are displayed for debugging and progress tracking.

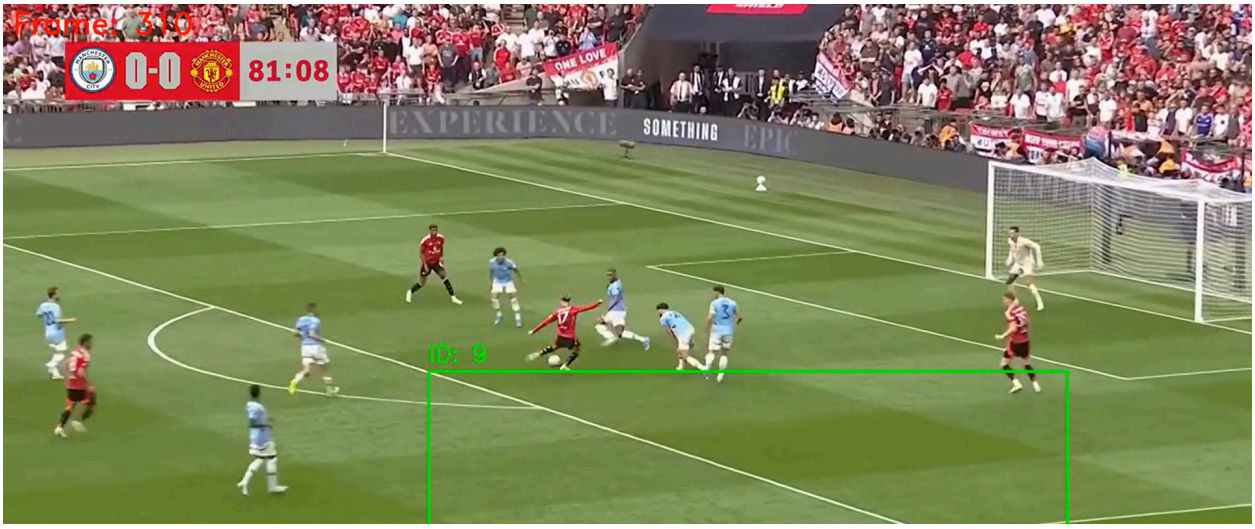
Conclusion

This project demonstrates how to combine a modern object detection model (YOLOv11) with a reliable tracking system (DeepSORT) to solve a real-world sports analytics task. The system performs accurate and consistent player re-identification across frames, forming the basis for further developments like pass detection, tactical pattern analysis, and automatic event tagging in football matches.

SCREENSHOTS OF THE RESULT OUTPUT -




Frame: 132
0-0
81:00
ID: 3 ID: 9


Frame: 319
0-0
81:08
ID: 9