Player Re-Identification Report

---

1. Approach and Methodology

We implemented a solution for re-identifying football players in a 15-second video using the following methodology:

- Object Detection: Used the provided YOLOv11 model to detect players in each frame.
- Player ID Assignment: Assigned IDs to detected players using DeepSORT tracking.
- Re-identification: DeepSORT maintained consistent IDs when players reappeared in the frame.
- Visualisation: Used OpenCV to draw bounding boxes and IDs on each frame. Output was optionally saved as a video.

2. Techniques Tried and Their Outcomes

- YOLOv11 + DeepSORT: Successfully tracked players and re-identified returning players.
- OpenCV: Used for rendering tracked output and saving video.
- Google Colab: Enabled seamless GPU-based model execution and testing.

3. Challenges Encountered

- Re-Identification Delay: Some ID mismatches occurred with occlusion or appearance changes.
- Model Limitations: The YOLO model occasionally misclassified overlapping players.
- Colab Upload Limitations: Video uploads and processing took time, depending on network speed

4. What Remains / Future Improvements

- Enhance Re-ID Accuracy: Integrate appearance-based models like OSNet.
- Handle Long-Term Occlusion: Improve long-gap tracking continuity.
- Train Custom Detection Model: Fine-tune YOLO for better player-specific detection.
- Deploy Real-Time App: Use OpenCV/Streamlit for live video tracking.

Player Re-Identification Report

Code Walkthrough (Cell-by-Cell Explanation)

Cell 1: Install Dependencies

```
!pip install ultralytics
!pip install opencv-python-headless
!pip install deep_sort_realtime
```

Explanation:

This cell installs all required Python packages:

- ultralytics for YOLOv11 model inference.
- opencv-python-headless for frame processing in Google Colab.
- deep_sort_realtime for player tracking and re-identification.

Cell 2: Upload Model and Video

```
from google.colab import files

print("Upload YOLOv11 model (.pt file):")
model_upload = files.upload()

print("Upload your 15-second video (15sec_input_720p.mp4):")
video_upload = files.upload()
```

Explanation:

This cell prompts the user to upload:

- YOLOv11 model file (best (2).pt)
- Input video (15sec_input_720p.mp4)

Cell 3: Load YOLO Model

```
from ultralytics import YOLO

model = YOLO("/content/best (2).pt")
print(f"✅ Model '{'/content/best (2).pt'}' loaded successfully!")
```

Explanation:

- Loads the uploaded YOLOv11 model. This model detects players and the ball frame-by-frame.

Cell 4: Initialise DeepSORT Tracker

```
from deep_sort_realtime.deepsort_tracker import DeepSort
tracker = DeepSort(max_age=30)
```

Player Re-Identification Report

Explanation:

- DeepSORT keeps track of player identities using motion and appearance. The max_age=30 allows a temporary loss of tracking.

Cell 5: Run Detection + Tracking and Display Results

```
import cv2
from IPython.display import display, clear_output
from google.colab.patches import cv2_imshow

video_path = "/content/15sec_input_720p.mp4"
cap = cv2.VideoCapture(video_path)

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

   results = model(frame)[0]

   detections = []
   for box in results.boxes.data.tolist():
       x1, y1, x2, y2, score, cls = box
       if int(cls) == 0:  # Assuming 'player' is class 0
           detections.append(([x1, y1, x2 - x1, y2 - y1], score, 'player'))

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

   for track in tracks:
       if not track.is_confirmed():
           continue
       track_id = track.track_id
       ltrb = track.to_ltrb()
       x1, y1, x2, y2 = map(int, ltrb)
       cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 0), 2)
       cv2.putText(frame, f"ID: {track_id}", (x1, y1 - 10),
                   cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

   clear_output(wait=True)
   cv2_imshow(frame)

cap.release()
cv2.destroyAllWindows()
```

Explanation:

Then, loop through each frame:

- Detect players using YOLO
- Track them using DeepSORT
- Draw bounding boxes and IDs

Cell 6: Save the Output Video

```
cap = cv2.VideoCapture(video_path)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))

out = cv2.VideoWriter('tracked_output.mp4', cv2.VideoWriter_fourcc(*'mp4v'),
fps, (width, height))

cap.set(cv2.CAP_PROP_POS_FRAMES, 0)  # Restart from beginning

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

   results = model(frame)[0]
   detections = []
   for box in results.boxes.data.tolist():
       x1, y1, x2, y2, score, cls = box
       if int(cls) == 0:
           detections.append(([x1, y1, x2 - x1, y2 - y1], score, 'player'))

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

   for track in tracks:
       if not track.is_confirmed():
           continue
       track_id = track.track_id
       ltrb = track.to_ltrb()
       x1, y1, x2, y2 = map(int, ltrb)
       cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 0), 2)
       cv2.putText(frame, f"ID: {track_id}", (x1, y1 - 10),
                   cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

   out.write(frame)
```

```
cap.release()
out.release()
print("Tracked video saved as 'tracked_output.mp4'")
```

Explanation:

- This saves the final output with player IDs as a new video (tracked_output.mp4).