Player Re-identification in a Single Video Feed

1. Introduction

This assignment focuses on solving the problem of player re-identification in a short 15-second football video. The goal was to ensure that each player maintains a consistent identity (ID) across frames—even when they move out of view and reappear. This simulates a real-time tracking scenario where accurate identity preservation is essential.

2. Approach & Methodology

2.1 Object Detection with YOLOv11

We used a fine-tuned YOLOv11 model from Ultralytics (`best.pt`) for detecting players. The model detects four classes—ball, goalkeeper, player, and referee. For tracking purposes, all human figures (player, goalkeeper, referee) were included.

- **Confidence Threshold:** Set to 0.3 for a good balance between missing detections and false positives.
- **Model Path:** `/kaggle/input/object-detection model/pytorch/default/1/best.pt`

2.2 Player Tracking with ByteTrack

Ultralytics provides built-in tracking using ByteTrack (`tracker="bytetrack.yaml"`), which was used directly with `persist=True` to maintain object memory across frames.

- **Why ByteTrack?** It's robust to occlusion and helps maintain consistent IDs for players who temporarily exit and re-enter the frame.
- **Filtering Classes:** We limited tracking to human classes to avoid unnecessary tracking of the ball or false detections.
- **Re-identification Handling:** ByteTrack links detections frame-to-frame, preserving IDs even after short absences.

2.3 Implementation Flow

- Load model and video, extract FPS and resolution.
- Initialize video writer to save the processed output.
- For each frame:
  - Run `model.track()` with `persist=True` and filtered classes.
  - Draw bounding boxes and display `track_id` with class labels.
  - Write the frame to the output video.
- Release video capture and writer after completion.

3. Techniques Tried & Observations

- **Detection-Only:** Failed for re-identification—player IDs reset every frame.
- **YOLO + ByteTrack:** Gave consistent IDs across frames and handled player re-entries well.
- **Filtering Classes:** Helped reduce noise, focused the tracker on relevant figures.

Overall, this combination offered a simple yet effective pipeline for consistent player tracking.

4. Challenges Faced

- **Kaggle File Paths:** Needed verification using `os.path.exists()` to ensure files were correctly accessed.
- **Confidence Tuning:** A lower threshold introduced false positives; 0.3 was the sweet spot.
- **Output Issues:** Required correct codec (`mp4v`) and write permissions in the Kaggle environment.
- **Minor ID Swaps:** ByteTrack handled most cases well, but in high-speed or overlapping movements, occasional ID shifts happened—though minimal due to the short and clear nature of the video.

5. Future Scope

- **More Complex Scenarios:** ByteTrack performs well here, but for crowded or longer videos, a re-ID network (e.g., using a Siamese model) could boost accuracy.
- **Metric-Based Evaluation:** Currently, tracking quality is evaluated visually. Using metrics like MOTA, IDF1, etc., would offer quantitative performance analysis.
- **Comparing Trackers:** DeepSORT, FairMOT, or other SOTA trackers could be tested for improved performance.
- **Parameter Tuning:** ByteTrack allows threshold tuning (e.g., `track_thresh`, `match_thresh`) which may improve results further.