Player Re-Identification

Author: Priyam Kumari
Date: July 2025

1. Approach and Methodology

The objective of this project was to **consistently identify players across frames in a single sports video**, even when players exit and re-enter the field of view.

Pipeline Overview:

- **Detection:**
  Implemented using a pre-trained **YOLOv11** model to detect players (class ID 0) in each frame.
- **Tracking:**
  Initially used **SORT** for real-time object tracking. Later replaced with **DeepSORT** to improve identity consistency using appearance features.
- **Re-Identification:**
  Extracted **color, texture, and spatial features** from player crops to aid in distinguishing similar-looking players. Used **cosine similarity** to match re-entered players to previous identities.
- **Modular Design:**
  Code is structured into modular components: `detector.py`, `feature_extractor.py`, `tracker.py`, `visualizer.py`, ensuring easy debugging and extension.

2. Techniques Tried and Outcomes

Technique	Outcome / Observation
SORT Tracker	Lightweight and fast, but failed to maintain consistent IDs when players reappeared after occlusion.
DeepSORT Tracker	More robust; used appearance embeddings to maintain identity consistency.
Color Histogram Features	Useful for basic differentiation; worked well in scenes with distinct jersey colors.
Texture (LBP) + Spatial Features	Enhanced matching when players had similar colors but different body shapes/positions.
YOLOv11 with NMS tuning	Improved detection accuracy by reducing duplicate boxes and refining confidence thresholds.

3. Challenges Encountered

- **Frequent Occlusions:** Players overlap or exit/enter frequently, making ID consistency a challenge.
- **Similar Jerseys:** Differentiating between players of the same team was non-trivial, especially under motion blur.
- **Short Video Length:** With only 15 seconds of footage, evaluating long-term ID consistency was limited.
- **Resource Limitations:** Couldn't train a custom re-ID network or fine-tune embeddings due to time/GPU constraints.

Project Building Process

The project was developed in a modular and incremental fashion, with each component built, tested, and improved over time. Below is the step-by-step breakdown:

1. Environment Setup

- Created a Python virtual environment using `venv` or `conda`.
- Installed necessary dependencies including:
  - `torch`, `opencv-python`, `numpy`, `matplotlib`
  - `ultralytics` (for YOLOv11)
  - `deep_sort_realtime` (for DeepSORT tracking)
- Verified GPU compatibility and tested sample YOLO detections.

2. Player Detection with YOLOv11

- Loaded the **YOLOv11 model** using the Ultralytics interface.
- Wrote a script to:
  - Read video frame-by-frame using OpenCV.
  - Detect **only players (class 0)**.
  - Draw bounding boxes and save the output video.
- Filtered detections based on confidence score and class ID.

Output: Annotated video with raw player detections.

3. Object Tracking Integration

a. Initial Tracking with SORT

- Integrated the SORT algorithm for fast, lightweight tracking.
- IDs were assigned to each player, but **got reset frequently** due to occlusions.

b. Switched to DeepSORT

- Integrated `deep_sort_realtime` for better **identity preservation**.
- This used **appearance features (embeddings)** to match players even after temporary disappearances.

Output: Video with consistent player IDs across frames.

4. Modularization

Split the code into the following clean, reusable modules inside a `src/` directory:

Module	Description
`detector.py`	Runs YOLOv11 detection and returns filtered bounding boxes.
`feature_extractor.py`	Extracts color, texture, and spatial features from player crops.
`tracker.py`	Manages tracking using DeepSORT and associates IDs.
`visualizer.py`	Draws boxes, IDs, and debug info on frames.
`main.py` / `run_pipeline.py`	Runs the full end-to-end pipeline on a video.

5. Feature-Based Re-Identification

To handle **player re-entry or identity correction**, a re-identification system was added:

- Extracted visual features from each player crop:
  - **Color histograms** (RGB or HSV)
  - **Texture descriptors** (e.g., Local Binary Patterns)
  - **Spatial features** (bounding box size, position)
- Used **cosine similarity** to compare current player crops with previous identities.
- If a player disappears and reappears, similarity matching is used to **reassign their original ID**.

6. Output and Testing

- Processed a 15-second sports video through the complete pipeline.
- Verified that:
  - Detection was accurate.
  - Tracking maintained consistent IDs for most players.
  - Re-identification helped match returning players correctly.
- Saved the output to `outputs/output_with_tracking.mp4`.

7. Final Steps

- Added logging and command-line arguments to improve usability.
- Wrote tests using synthetic frames to validate each module individually.
- Prepared documentation and final report for submission.

Here's a **step-by-step guide on how to run your Player Re-Identification project**, formatted cleanly for your report or `README.md`:

How to Run the Project (Step-by-Step)

Step 1: Clone the Repository

```
git clone https://github.com/your-username/player-reid-project.git
cd player-reid-project
```

Step 2: Set Up the Virtual Environment

Create and activate a virtual environment:

```
# Create
python -m venv venv

# Activate
# On Windows
venv\Scripts\activate

# On macOS/Linux
source venv/bin/activate
```

Step 3: Install Dependencies

```
pip install -r requirements.txt
```

Make sure the following are included in `requirements.txt`:

- `torch`
- `opencv-python`
- `ultralytics` (for YOLOv11)
- `deep_sort_realtime`
- `numpy`, `matplotlib`, `scikit-image`

Step 4: Place Your Input Video

Put your input video (e.g., `input.mp4`) inside a folder named `inputs/`.

```
mkdir inputs
# Copy your video here, e.g.:
# inputs/input.mp4
```

Step 5: Run the Full Pipeline

```
python run_pipeline.py --input inputs/input.mp4 --output
outputs/output_with_tracking.mp4
```

This script will:

- Detect players in each frame using YOLOv11
- Track them across frames using DeepSORT
- Optionally re-identify returning players
- Draw bounding boxes and IDs
- Save the final video in `outputs/`

Step 6: Check the Output

After the script runs, your processed video will be saved as:

```
outputs/output_with_tracking.mp4
```

Open it to verify detection + tracking consistency.

Optional: Run Component-wise Tests

You can test individual components using the test script:

```
python test_pipeline.py
```

This runs basic tests for:

- Detection
- Feature extraction
- Tracking
- Visualization

Optional CLI Arguments

You can customize execution via command-line flags:

```
python run_pipeline.py \
    --input inputs/your_video.mp4 \
```

```
--output outputs/tracked_video.mp4 \
--confidence 0.3 \
--visualize True
```