Player Re-Identification Using DeepSORT and BoT-SORT

Documentation of Models

Author: Gogada Harsha Vardhan

harshavardhangogada@gmail.com

As part of Option 2 from the assignment, I undertook the challenge of developing two cutting-edge models for player re-identification in soccer match videos: BoT-SORT and DeepSORT. This task was both challenging and engaging. I had a great time working on it.

The beauty of working with advanced tracking models is that we don't always need to build everything from scratch. With the availability of powerful pre-trained frameworks, our primary focus shifts to fine-tuning and integration, which still demands a deep understanding of the underlying mechanics and data.

So now let's dive into the details of the models I have developed.

Model 1: BoT-SORT:

BoT-SORT is an advanced multi-object tracking algorithm built on top of ByteTrack, incorporating optional appearance-based ReID features.

I utilized the official Ultralytics BoT-SORT tracker with YOLOv8. The botsort.yaml file is the configuration file for BoT-SORT used by Ultralytics' YOLOv8 and above. It controls how the tracker behaves—how it links detections across frames to maintain object identities.

I customized the botsort.yaml configuration file to optimize performance for our dataset.

Methodology:

I made the following key adjustments in botsort.yaml:

- *track_high_thresh*: Minimum confidence to update or start tracks using strong detections.

Set to match your YOLO conf (e.g., 0.5).

- *track_low_thresh*: Allows using low-confidence detections to update existing tracks.
  Important for handling occlusions; too low may track noise.

- *new_track_thresh*: Minimum score required to start a new track.
  Raise this to reduce false-positive tracks.

- *match_thresh*: IoU threshold for matching detections to existing tracks.
  Higher = stricter; lower = more tolerant of movement.

- *with_reid*: Enables appearance-based ReID.
  True improves ID consistency but is slower. False relies on motion only.

- *gmc_method*: Stabilizes camera motion (none, sift, orb).
  Use 'none' for static videos.

- *max_age*: How long to keep a track without detection before removing it.
  Increase for crowded scenes with occlusion.

- *n_init*: Frames needed to confirm a new track.
  Helps filter out short-lived false tracks.

- *min_box_area*: Ignore detections smaller than this area.
  Useful to filter out noise or background clutter.

I then saved the output video with bounding boxes and unique track IDs for each player.

Output Results:

- The BotSort maintains remarkably good short-term id consistencies, and the compatibility with pre-trained YOLO models is great.
- It is lightweight and fast. I ran these models using an NVIDIA TESLA P100 GPU on *Kaggle.*
- The whole runtime time wraps around 117 seconds.
- It had Fewer ID Switches because it combines motion and (optional) appearance features for more consistent object IDs.

Limitations:

BoT-SORT algorithms struggle with long-term consistent IDs because of regions such as “player-overlaps” and “fast-panning scenes”.

Requires scenario-specific fine-tuning to avoid ID switches or detection failures

Model 2: DeepSORT (with Re-Identification):

DeepSORT (Deep Simple Online and Realtime Tracking) is an object tracking algorithm that combines:

- SORT– IoU + Kalman Filtering.

- Deep Re-Identification model for appearance-based consistency.

Methodology:

I have worked on cloning an git hub repo (link), which is an adaptation of the prominent 'Deep Sort' Algorithm introduced in 2017.

It has all the standard tracking and filtering algorithms, along with re-ID implementations.

I customized some of the core files of this repo manually to adapt to our settings since the repo is designed for various YOLO models and versatile goals.

Files such as tracker.py, DeepSortReid.py, detect_objects.py, extract_features.py, etc., need to be edited and executed to work on our model.

(Note: In the output video of this algorithm, I haven't trained for different classes such as player and goalkeeper. I took the liberty to mainly focus on the re-ID part, although with only a few more modifications, we can achieve multi-class tracking also, just like we did in the BotSort algorithm.)

Output Results:

- Although it uses re-ID as core part of its algorithm is still lighter when compared to other algorithms such as Strong-Sort and ByteTrack when using re-ID.
- Unlike basic SORT, it doesn't rely solely on IoU, and motion appearance cues help prevent frequent ID switching, especially in crowded frames.
- You can inject custom track logic, memory, cache, or visualization directly inside your tracking loop, which is harder in out-of-the-box frameworks.

Limitations:

- In the case of our video, my model struggled in the first 2 seconds and got much better later on.

- I noticed that it is sensitive than BotSort when it comes to ID switching
- Integrating out YOLO model deeply might give better results, but it is too complex and hard to tune.

So these are the two models I have worked on both BoT-SORT and DeepSORT offer unique advantages, and their application to sports analytics is incredibly promising.

Difficulties I have faced:

- Even though “best.pt” is well trained on players, referees, and ball, video being fast, only 720p, and just 15 secs caused my models to struggle.
- I have also tried the Strong-sort implementation, but it is somehow very specific about the YOLO model we feed in.

What would I do with more time/resources :

- I would work on improving the re-ID accuracies of my models by advanced fine-tuning or writing my own tracking and re-ID algorithm for the task.
- I would spend time on learning the techniques of Strong-SORT since it is supposed to work better than most of the algorithms.

Thank you for the opportunity to work on this.

Gogada Harsha Vardhan