MOT Take Home Assignment

Name: Razan Siraj
Repository Link: https://github.com/Razan1412/Multi-Object-Tracking.git

Notes: The **main** branch contains the full repository, and should be cloned to test my code. A pull request containing the **key changes** I made can be found for review here: https://github.com/Razan1412/Multi-Object-Tracking/pull/1

Task 1: Conceptual Understanding of ByteTrack

Multi Object Tracking

MOT is a computer vision task that involves detecting **objects** in a video and assigning a consistent **identity** to each object **across frames**. The primary challenges include handling frequent occlusions (where objects are hidden by others), managing object entries and exits from the scene, and maintaining identities through the video.

The 3 main components are:

1. An **Object Detector** that detects all objects in a frame.
2. A **Feature Extractor** that assigns separate identities to similar objects. Two pedestrians (same class) can thus be assigned different identities.
3. A **Track Manager** to handle creation, updating and deletion of tracks (as objects enter, move and leave the frame)

How ByteTrack Works

Most MOT systems discard detections with confidence scores below a certain threshold. This means that occluded objects stop being tracked prematurely, despite still existing in the scene.

ByteTrack attempts to avoid this by introducing a **two-step strategy**:

1. Match high confidence detections to existing object tracks using similarity measures (usually based on IoU or on features from the **Feature**

Extractor).

2. Try to match any **unmatched tracks** to the low-confidence detections, using **only** IoU (since appearance features are unreliable under occlusion).

This approach allows the tracker to avoid losing occluded objects and **maintain object identities** even in difficult conditions. The result is a significant reduction in ID switches and missed objects.

Task 2: Reproducing Baseline Results (YOLOX + FastReID)

Results:

- Highest HOTA Score: **69.093 ≈ 69.1**

Challenges

1. **Development Environment:** I do not have a local GPU, so I set up a GCP VM using a T4 GPU. I SSHed in using VSCode, and set up permanent keys to maintain ease of access. This was the setup I used throughout.

2. **Dependency Conflicts:** The TrackTrack repository lacks any requirements.txt or environment.yml file for the whole pipeline, only individual requirements in each repo. I had to guess and check library versions for compatibility, which made for a frustrating dev experience.

3. **Randomness:** While the repository provided mot17_val_0.95.pickle detections, it did not provide mot17_val_0.80.pickle detections. As such, I decided to run the entire pipeline from scratch. While every execution resulted in a HOTA score above 69.05 (≈ 69.1), the results varied randomly due to inherent variations in detections, and I wanted a higher score.

Task 3: Integrating a New Detector (RF-DETR)

Results

- Highest HOTA Score: **24.817 ≈ 24.8**

Challenges

1. **Architectural Difference:** YOLOX is CNN based, and outputs multiple overlapping bboxes as its detections. NMS is required to filter out the duplicates and keep the max confidence bounding box.

   Meanwhile, RF-DETR is Transformer based, and inherently outputs non-overlapping bboxes, making NMS inapplicable. Since the Tracker expected 2 pickle files (.80 and .95 NMS), I chose to run RF-DETR once, and then produce 2 pickle files by filtering detections based on confidence scores (above 95 vs above 80).

2. **Empty Detections:** Since the RF-DETR model was not fine tuned for this dataset, it sometimes found 0 objects in frame (especially for the above 95 confidence threshold). My code initially did not include these frames in the output .pickle file.

   However, this led to errors downstream with the Tracker, which assumed that if a frame existed in the 0.80 detection file, it must also exist in the 0.95 file, leading to a KeyError when it couldn't find the corresponding entry. I solved this by including entries for every frame, even if empty.

Task 4: Integrating a New Re-ID Model (OSNet)

Results

- Highest HOTA Score: **23.815 ≈ 23.8**

Challenges

1. **Type Error**: My script was reading images with OpenCV and performing cropping operations, which resulted in NumPy arrays. However, the torchreid library's image transformation pipeline was designed to work with **PIL Image** objects.

   I converted the NumPy array crop into a PIL Image right before passing it to the transformation pipeline.

Summary of Results

Task	Detector	Feature Extractor	HOTA
Task 2	YOLOX	FastReID	69.1
Task 3	RF-DETR	FastReID	24.8
Task 4	RF-DETR	OSNet	23.8