

# Object Tracking - Project

Dinh-Thang Duong

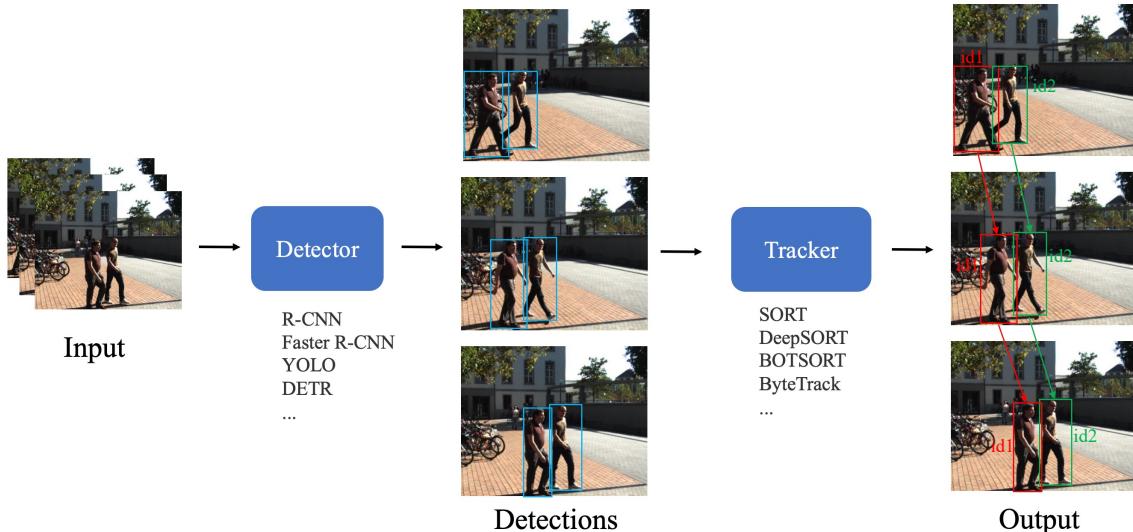
Minh-Duc Bui

Quang-Vinh Dinh

Ngày 15 tháng 1 năm 2024

## Phần I: Giới thiệu

**Object Tracking** là một bài toán quan trọng trong lĩnh vực Computer Vision, đòi hỏi khả năng liên tục theo dõi và xác định vị trí của các object trong các khung hình video liên tục. Một trong những phương pháp phổ biến để giải quyết bài toán này là sử dụng kết hợp giữa model YOLOv8 và thuật toán DeepSORT.



YOLO (You Only Look Once) là một model object detection nhanh chóng và hiệu quả, cho phép xác định các object trong một hình ảnh hoặc video bằng cách chia hình ảnh thành grid, việc predict bounding box và class label diễn ra trên từng grid cell. YOLOv8 là phiên bản mới nhất với độ chính xác cao và tốc độ nhanh hơn trong môi trường thời tiết và ánh sáng khác nhau.

DeepSORT (Simple Online and Realtime Tracking with a Deep Association Metric) là một thuật toán object tracking dựa trên sự kết hợp giữa deep learning và kỹ thuật theo dõi dựa trên đặc trưng. Nó sử dụng mạng siamese để tính toán sự tương đồng giữa các bounding box liên tiếp trong các khung hình và tracking các object dựa trên sự thay đổi trong sự tương đồng này.

Khi kết hợp YOLOv8 và DeepSORT, ta có thể thực hiện việc phát hiện và theo dõi object trong video một cách hiệu quả. YOLOv8 sẽ xác định object trong từng frame ảnh, sau đó DeepSORT sẽ sử dụng thông tin này để xác định và tracking các object theo thời gian. Kết quả là một hệ thống hoàn chỉnh cho việc tracking các object trong video, có thể được ứng dụng trong nhiều lĩnh vực như giám sát an ninh, theo dõi giao thông, và nhiều ứng dụng khác đòi hỏi sự hiểu biết và quản lý vị trí của các object trong thời gian thực.

Trong project này, ta sẽ xây dựng một chương trình Pedestrian Tracking (vật thể tracking ở đây là người đi bộ) sử dụng YOLOv8 và DeepSORT. Input và output của chương trình như sau:

- **Input:** Một video có chứa người đi bộ.
- **Output:** Đường di chuyển (trajectory) của những người đi bộ trong video.

## Phần II: Cài đặt chương trình

Trong pipeline của project này, ta sẽ xây dựng hai module chính gồm module Detector và module Tracker. Chương trình được cài đặt và chạy trên Google Colab. Cách triển khai từng module sẽ được trình bày ở phần bên dưới như sau:

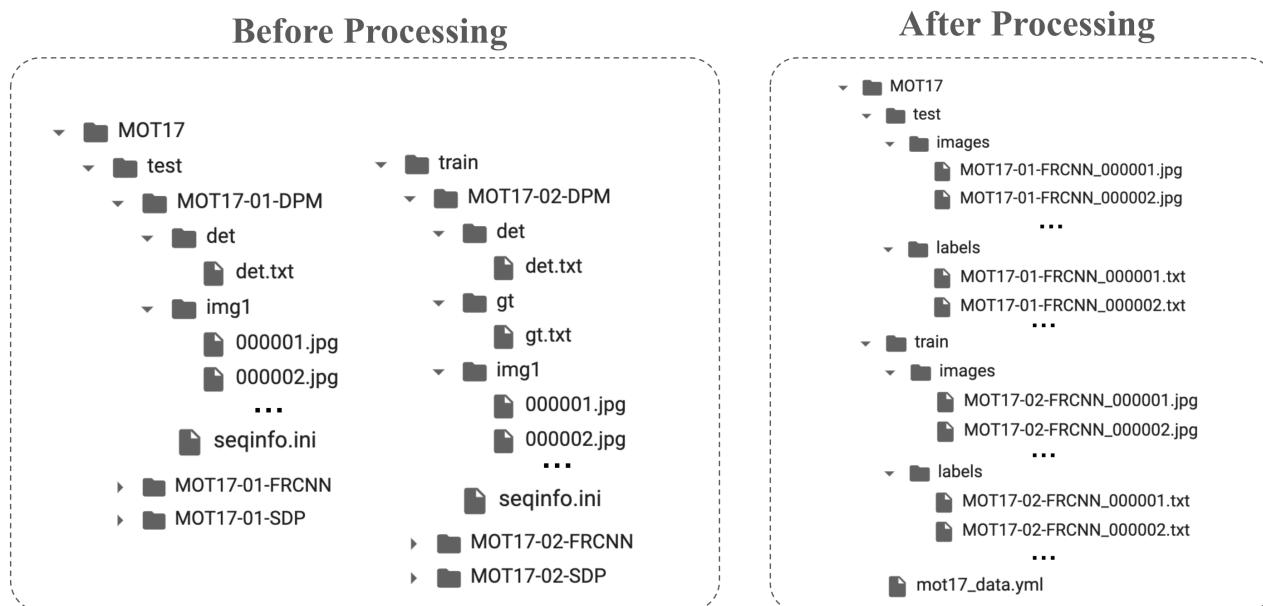
- **Tải bộ dữ liệu Object Tracking (MOT17):** Trong project này, ta sẽ sử dụng bộ dữ liệu **MOT17** (Multiple Object Tracking 17), các bạn tải bộ dữ liệu này tại [đây](https://motchallenge.net/data/MOT17.zip). Trong môi trường code notebook, ta có thể tải bộ dữ liệu này một cách tự động bằng lệnh sau:

```
1 !wget https://motchallenge.net/data/MOT17.zip
```

Sau khi download data xong ta sẽ unzip data thông qua lệnh sau:

```
1 !unzip -qq MOT17.zip
```

Sau khi đã unzip data, ta sẽ thu được folder MOT17 như hình bên trái phía dưới:



Hình 1: Hình bên trái là cấu trúc data MOT17 khi đã unzip. Hình phải là cấu trúc MOT17 sau khi đã process để phù hợp với cấu trúc của YOLO.

Ta có 2 folder chính là train và test, trong đó bao gồm các subfolder, mỗi subfolder tương ứng với 1 video tracking đã được chuyển thành các frame ảnh. Mỗi subfolder sẽ có 3 version tương với kết quả từ 3 model khác nhau là DPM, FRCNN và SDP (nội dung các frame trong các subfolder này là giống nhau, chỉ thông tin khác nhau ở file det.txt). Sau đây là thông tin có trong 1 folder:

- det: chứa file det.txt chứa thông tin chỉ class person trong từng frame.
- gt: chứa file gt.txt bao gồm thông tin của từng object trong từng frame (không có đối với folder test).
- img1: chứa các image được sắp xếp theo thứ tự frame.
- seqinfo.ini: chứa các metadata về image như hình sau:

```
1 [Sequence]
2 name=MOT17-02-DPM
```

```

3 imDir=img1
4 frameRate=30
5 seqLength=600
6 imWidth=1920
7 imHeight=1080
8 imExt=.jpg

```

Các file det.txt và gt.txt có format như sau:

```

1 frame id bb_left bb_top bb_width bb_height confident class visibility

```

Lưu ý: Trong project này ta sẽ chỉ sử dụng file det.txt (chỉ bao gồm person class), do file gt.txt bao gồm rất nhiều các class khác ngoài person sẽ tăng độ phức tạp của project.

Ta sẽ loop qua từng subfolder trong 2 folder chính train và test để thực hiện process data, trước đó, ta hãy cùng viết một số hàm phụ trước:

1. **Import các thư viện cần thiết:** Đầu tiên, ta khai báo một vài thư viện mà ta sẽ sử dụng trong phần process data này:

```

1 import pandas as pd
2 import os
3 import yaml
4 import shutil
5 import configparser
6
7 from tqdm import tqdm

```

2. **Xây dựng hàm normalize bounding box:** Hàm đầu tiên ta dùng để chuyển đổi bounding box từ (x\_min, y\_min, width, height) trong file det.txt thành (bb\_left, bb\_top, width, height) và normalize về [0; 1].

```

1 def convert_to_yolo_format(bb, img_width, img_height):
2     x_center = bb['bb_left'] + (bb['bb_width'] / 2)
3     y_center = bb['bb_top'] + (bb['bb_height'] / 2)
4
5     # Normalize the coordinates by the dimensions of the image
6     x_center /= img_width
7     y_center /= img_height
8     bb_width_normalized = bb['bb_width'] / img_width
9     bb_height_normalized = bb['bb_height'] / img_height
10
11    # Clip the values to make sure they are between 0 and 1
12    x_center = max(min(x_center, 1), 0)
13    y_center = max(min(y_center, 1), 0)
14    bb_width_normalized = max(min(bb_width_normalized, 1), 0)
15    bb_height_normalized = max(min(bb_height_normalized, 1), 0)
16
17    return (x_center, y_center, bb_width_normalized, bb_height_normalized)

```

3. **Xây dựng hàm đọc thông tin file seqinfo.ini:** Tiếp theo là hàm để đọc thông tin từ file seqinfo.ini để lấy thông tin về width và height của các image trong subfolder đó, và file det.txt để lấy các thông tin về bounding box. Cuối cùng là tạo 1 file label mới để lưu vào (vì format của YOLO yêu cầu mỗi file ảnh sẽ có tương ứng một file label).

```

1 def process_folder(folder_path):
2     # Read image dimensions from seqinfo.ini
3     config = configparser.ConfigParser()

```

```

4     config.read(os.path.join(folder_path, 'seqinfo.ini'))
5     img_width = int(config['Sequence']['imWidth'])
6     img_height = int(config['Sequence']['imHeight'])
7
8     # Load ground truth data
9     gt_path = os.path.join(folder_path, 'det/det.txt')
10    gt_data = pd.read_csv(gt_path, header=None, names=['frame', 'id', 'bb_left', 'bb_top', 'bb_width', 'bb_height', 'conf', 'class', 'visibility'])
11
12    labels_folder = os.path.join(folder_path, 'labels')
13    os.makedirs(labels_folder, exist_ok=True)
14
15    for frame_number in gt_data['frame'].unique():
16        frame_data = gt_data[gt_data['frame'] == frame_number]
17        label_file = os.path.join(labels_folder, f'{frame_number:06d}.txt')
18
19        with open(label_file, 'w') as file:
20            for _, row in frame_data.iterrows():
21                yolo_bb = convert_to_yolo_format(row, img_width, img_height)
22                file.write(f'0 {yolo_bb[0]} {yolo_bb[1]} {yolo_bb[2]} {yolo_bb[3]}\n')

```

**4. Xây dựng hàm process toàn bộ dữ liệu:** Cuối cùng, ta sẽ viết hàm để process toàn bộ folder trong 1 folder chính là train/test. Ta sẽ chỉ sử dụng 1 trong 3 version (FRCNN), các folder khác ta sẽ xóa bỏ.

```

1 def process_all_folders(base_directory):
2     # List all subdirectories in the base directory
3     for folder_name in tqdm(os.listdir(base_directory)):
4         folder_path = os.path.join(base_directory, folder_name)
5
6         # Delete folder not contain 'FRCNN' in name
7         if 'FRCNN' not in folder_name:
8             os.system(f'rm -rf {folder_path}')
9             continue
10
11        if os.path.isdir(folder_path):
12            process_folder(folder_path)

```

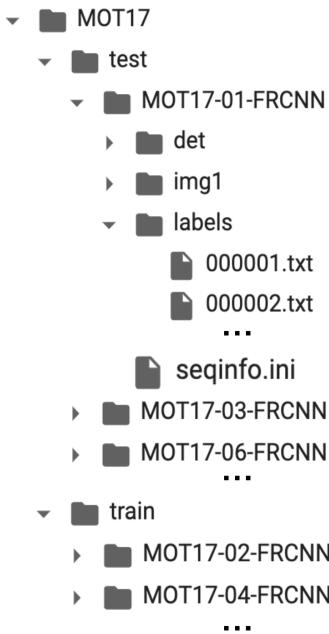
Sau khi đã chuẩn bị các hàm trên, ta chỉ cần sử dụng 2 dòng code sau để tiến hành process folder train và test.

```

1 process_all_folders('MOT17/train')
2 process_all_folders('MOT17/test')

```

Sau khi đã process labels cho các folder, kết quả sẽ như hình sau:



Tiếp theo, ta sẽ move các file image và label trong từng subfolder của train và test vào cùng 1 folder duy nhất và xóa các subfolder còn lại. Để làm được bước này, ta cần định nghĩa một số hàm sau:

1. **Xây dựng hàm move file:** Vì các file name trong từ subfolder đều giống nhau, đều bắt đầu từ 000001.jpg. Để có thể move vào cùng 1 folder chung ta cần thêm phần `folder_name` vào đầu tên file để các file không bị override.

```

1 def rename_and_move_files(src_folder, dst_folder,
                           folder_name, file_extension):
2     for filename in os.listdir(src_folder):
3         if filename.endswith(file_extension):
4             # Include folder name in the new filename
5             new_filename = f'{folder_name}_{filename}'
6             shutil.move(os.path.join(src_folder, filename), os.path.join(
7                 dst_folder, new_filename))
    
```

2. **Xây dựng hàm move tất cả files:** Tiếp theo ta sẽ viết hàm để move tất cả các file từ các subfolders trong `base_directory` vào hai thư mục chính là 'images' và 'labels'. Dối với mỗi subfolder, các file ảnh (.jpg) trong thư mục 'img1' và các file nhãn (.txt) trong thư mục 'labels' được di chuyển.

```

1 def move_files_all_folders(base_directory):
2     images_dir = os.path.join(base_directory, 'images')
3     labels_dir = os.path.join(base_directory, 'labels')
4     os.makedirs(images_dir, exist_ok=True)
5     os.makedirs(labels_dir, exist_ok=True)
6
7     for folder_name in tqdm(os.listdir(base_directory)):
8         if folder_name in ['images', 'labels']: # Skip these folders
9             continue
10
11         folder_path = os.path.join(base_directory, folder_name)
12         if os.path.isdir(folder_path):
13             rename_and_move_files(os.path.join(folder_path, 'img1'),
14                                   images_dir, folder_name, '.jpg')
    
```

```

15     rename_and_move_files(os.path.join(folder_path, 'labels'),
16                           labels_dir, folder_name, '.txt')

```

3. **Xây dựng hàm xóa các subfolders:** Tiếp theo ta sẽ xóa tất cả các subfolders trong base\_directory, trừ hai thư mục 'images' và 'labels'. Đối với mỗi subfolder không phải là 'images' hoặc 'labels', hàm shutil.rmtree() được sử dụng để xóa toàn bộ thư mục đó, bao gồm cả nội dung bên trong.

```

1 def delete_subfolders(base_directory):
2     for folder_name in os.listdir(base_directory):
3         folder_path = os.path.join(base_directory, folder_name)
4         if os.path.isdir(folder_path) and folder_name not in ['images', 'labels']:
5             shutil.rmtree(folder_path)
6             print(f"Deleted folder: {folder_name}")

```

Cuối cùng, ta sẽ dùng 2 dòng code sau để thực hiện xóa các subfolder cho folder train và test.

```

1 delete_subfolders('MOT17/train')
2 delete_subfolders('MOT17/test')

```

Cuối cùng ta sẽ setup file yaml chứa thông tin về các folder train/test và số lượng class cho project này:

```

1 class_labels = [
2     'objects',
3 ]
4 dataset_root_dir = os.path.join(
5     os.getcwd(),
6     'MOT17'
7 )
8 yolo_yaml_path = os.path.join(
9     dataset_root_dir,
10    'mot17_data.yml'
11 )
12
13 data_yaml = {
14     'path': dataset_root_dir,
15     'train': 'train/images',
16     'val': 'test/images',
17     'nc': len(class_labels),
18     'names': class_labels
19 }
20
21 with open(yolo_yaml_path, 'w') as f:
22     yaml.dump(data_yaml, f, default_flow_style=False)

```

Đây là kết quả cuối cùng mà ta có được:



- **Module Detector: YOLOv8** Đối với module Detector, ta sẽ sử dụng model YOLOv8 để phát hiện vị trí của các vật thể cần tracking. Các bước cài đặt cho module này như sau:

1. **Tải thư viện ultralytics:** ta cài đặt thư viện cho YOLO bằng lệnh sau:

```

1 !pip install ultralytics -q
2
3 import ultralytics
4 ultralytics.checks()
    
```

Sau khi tải xong, ta dùng hàm `ultralytics.checks()` để kiểm tra việc cài đặt đã ổn hay chưa.

2. **Import các thư viện cần thiết:**

```

1 from ultralytics import YOLO
    
```

3. **Training model:** Sau khi đã chuẩn bị data ta chỉ cần khai báo một số hyper-parameter và tiến hành train model YOLOv8:

```

1 from ultralytics import YOLO
2
3 # Load the YOLOv8 model
4 model = YOLO('yolov8s.pt')
5
6 # Config
7 epochs = 30
8 batch_size = -1 # Auto scale based on GPU memory
9 img_size = 640
10 project_name = 'models/yolo'
11 name = 'yolov8s_mot17_det'
12
13 # Train the model
14 results = model.train(
15     data=yolo_yaml_path,
16     epochs=epochs,
17     batch=batch_size,
18     imgsz=img_size,
19     project=project_name,
20     name=name
21 )
    
```

```

Epoch 26/30 GPU_mem box_loss cls_loss dfl_loss Instances Size
14.1G 0.5529 0.3136 0.8212 293 640: 100% [██████] 85/85 [00:32<00:00, 2.65it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████] 47/47 [00:21<00:00, 2.17it/s]
all 5919 110141 0.899 0.821 0.912 0.647

Epoch 27/30 GPU_mem box_loss cls_loss dfl_loss Instances Size
14.2G 0.5403 0.306 0.8192 309 640: 100% [██████] 85/85 [00:32<00:00, 2.65it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████] 47/47 [00:21<00:00, 2.15it/s]
all 5919 110141 0.898 0.816 0.908 0.648

Epoch 28/30 GPU_mem box_loss cls_loss dfl_loss Instances Size
14.2G 0.5277 0.2974 0.8173 268 640: 100% [██████] 85/85 [00:31<00:00, 2.66it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████] 47/47 [00:22<00:00, 2.11it/s]
all 5919 110141 0.901 0.809 0.904 0.648

Epoch 29/30 GPU_mem box_loss cls_loss dfl_loss Instances Size
14.1G 0.5213 0.294 0.8162 254 640: 100% [██████] 85/85 [00:31<00:00, 2.67it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████] 47/47 [00:22<00:00, 2.09it/s]
all 5919 110141 0.9 0.812 0.907 0.656

Epoch 30/30 GPU_mem box_loss cls_loss dfl_loss Instances Size
14.3G 0.508 0.287 0.8132 371 640: 100% [██████] 85/85 [00:32<00:00, 2.64it/s]
Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████] 47/47 [00:24<00:00, 1.91it/s]
all 5919 110141 0.901 0.809 0.905 0.652

30 epochs completed in 0.476 hours.
Optimizer stripped from runs/detect/yolov8s_det/weights/last.pt, 22.5MB
Optimizer stripped from runs/detect/yolov8s_det/weights/best.pt, 22.5MB

Validating runs/detect/yolov8s_det/weights/best.pt...
Ultralytics YOLOv8.0.237 Python-3.10.13 torch-2.1.1+cu118 CUDA:0 (NVIDIA RTX A5000, 24247MiB)
Model summary (fused): 168 layers, 11125971 parameters, 0 gradients, 28.4 GFLOPs
    Class Images Instances Box(P R mAP50 mAP50-95): 100% [██████] 47/47 [00:39<00:00, 1.18it/s]
    all 5919 110141 0.9 0.812 0.907 0.656
Speed: 0.1ms preprocess, 0.9ms inference, 0.0ms loss, 1.1ms postprocess per image
Results saved to runs/detect/yolov8s_det

```

Lưu ý, sau khi traning hoàn tất, các bạn cần lưu lại file weights (best.pt hoặc last.pt) để load mô hình đã train được trong file code tracking.

4. **Xây dựng class Detector:** Với model YOLO vừa huấn luyện xong, ta sẽ xây dựng một class Detector để tiện trong việc sử dụng. Class này sẽ nhận tham số đầu vào là đường dẫn file model, có phương thức `detect()` để thực hiện detection cho một ảnh bất kỳ. Cách cài đặt như sau:

```

1 from ultralytics import YOLO
2
3 class YOLOv8:
4     def __init__(self, model_path):
5         self.model = YOLO(model_path)
6
7     def detect(self, source_img):
8         results = self.model.predict(source_img, verbose=False)[0]
9         bboxes = results.boxes.xywh.cpu().numpy()
10        bboxes[:, :2] = bboxes[:, :2] - (bboxes[:, 2:] / 2)
11        scores = results.boxes.conf.cpu().numpy()
12        class_ids = results.boxes.cls.cpu().numpy()
13
14    return bboxes, scores, class_ids
15
16
17

```

Trong phương thức `detect()`, ta sẽ cài đặt để trả về ba thông tin gồm:

- **bboxes:** Danh sách các bounding box. Lưu ý, bounding box trả về phải có format (x\_min, y\_min, width, height).
- **scores:** Danh sách confidence score tương ứng với các bounding box.
- **class\_ids:** Danh sách các class id tương ứng với các bounding box.

## • Module Tracker: DeepSORT

1. **Cài đặt thư viện:** Để sử dụng được DeepSORT, các thư viện sau cũng cần phải được cài đặt:

```

1 !pip install scikit-learn numpy opencv-python tensorflow spacy -q
2 !pip install gdown==4.6.0 -q

```

Ta tải gdown phiên bản 4.6.0 để có thể tải tự động folder checkpoint của DeepSORT trong ở bước sau.

2. **Cài đặt source code DeepSORT:** Để cài đặt thuật toán này, đầu tiên ta cần tải source code từ github về máy như sau:

```

1 !git clone https://github.com/wjnwn59/deep_sort.git

```

Trong đoạn code trên, ta clone source code từ github vào thư mục /content của colab.

3. **Tải checkpoint CNN của DeepSORT:** Để chạy được DeepSORT, ta cần tải các file tại [đây](#). Các bạn có thể tải một cách tự động đoạn code này như sau:

```

1 !gdown --no-check-certificate --folder https://drive.google.com/open?id=18
      fKzfqnqhW3s9zwsCbnVJ5XF2JFeqMp

```

#### 4. Import các thư viện cần thiết:

```

1 import os
2 import cv2
3 import numpy as np

```

5. **Xây dựng class Tracker:** Tương tự như phần Detector, ta cũng sẽ xây dựng một class cho Tracker để thuận tiện trong việc sử dụng. Code cài đặt như sau: Đầu tiên, ta import các module từ source code DeepSORT cần thiết để chạy tracking. Việc import module từ file code cũng có cú pháp tương tự như import thư viện thông thường:

```

1 from deep_sort.deep_sort import nn_matching
2 from deep_sort.deep_sort.detection import Detection
3 from deep_sort.deep_sort.tracker import Tracker
4 from deep_sort.tools import generate_detections as gdet

```

Sau đó, ta định nghĩa class DeepSORT như sau:

```

1 class DeepSORT:
2     def __init__(
3         self,
4             model_path='resources/networks/mars-small128.pb',
5             max_cosine_distance = 0.7,
6             nn_budget = None,
7             classes=['objects']
8     ):
9
10         self.encoder = gdet.create_box_encoder(model_path, batch_size=1)
11         self.metric = nn_matching.NearestNeighborDistanceMetric('cosine',
12             max_cosine_distance, nn_budget)
13         self.tracker = Tracker(self.metric)
14
15         key_list = []
16         val_list = []
17         for ID, class_name in enumerate(classes):
18             key_list.append(ID)
19             val_list.append(class_name)
20         self.key_list = key_list
21         self.val_list = val_list
22
23
24

```

```

25     def tracking(self, origin_frame, bboxes, scores, class_ids):
26         features = self.encoder(origin_frame, bboxes)
27
28         detections = [Detection(bbox, score, class_id, feature)
29                         for bbox, score, class_id, feature in zip(bboxes,
30                                                         scores,
31                                                         class_ids,
32                                                         features)]
33
34         self.tracker.predict()
35         self.tracker.update(detections)
36
37         tracked_bboxes = []
38         for track in self.tracker.tracks:
39             if not track.is_confirmed() or track.time_since_update > 5:
40                 continue
41             bbox = track.to_tlbr()
42             class_id = track.get_class()
43             conf_score = track.get_conf_score()
44             tracking_id = track.track_id
45             tracked_bboxes.append(
46                 bbox.tolist() + [class_id, conf_score, tracking_id]
47             )
48
49         tracked_bboxes = np.array(tracked_bboxes)
50
51     return tracked_bboxes

```

Trong class này, ta sẽ nhận tham số đầu vào là file checkpoint của DeepSORT cũng như các thông tin cần thiết của model. Ngoài ra, ta cũng sẽ xây dựng phương thức `tracking()` để thực hiện tracking với danh sách kết quả detections từ detector (DeepSORT nhận kết quả bounding box theo format (`x_min, y_min, width, height`)).

**6. Xây dựng hàm vẽ kết quả tracking:** Với kết quả detection từ YOLOv8 và tracking từ DeepSORT, ta sẽ xây dựng một hàm dùng để vẽ các thông tin dự đoán của hai model, dưới dạng bounding box và mã ID, lên hình đầu vào. Cách làm như sau:

```

1 def draw_detection(img, bboxes, scores, class_ids, ids,
2                     classes=['objects'], mask_alpha=0.3):
3     height, width = img.shape[:2]
4     np.random.seed(0)
5     rng = np.random.default_rng(3)
6     colors = rng.uniform(0, 255, size=(len(classes), 3))
7
8     mask_img = img.copy()
9     det_img = img.copy()
10
11    size = min([height, width]) * 0.0006
12    text_thickness = int(min([height, width]) * 0.001)
13
14    # Draw bounding boxes and labels of detections
15    for bbox, score, class_id, id_ in zip(bboxes, scores, class_ids, ids):
16        color = colors[class_id]
17
18        x1, y1, x2, y2 = bbox.astype(int)
19
20        # Draw rectangle
21        cv2.rectangle(det_img, (x1, y1), (x2, y2), color, 2)
22
23

```

```

24     # Draw fill rectangle in mask image
25     cv2.rectangle(mask_img, (x1, y1), (x2, y2), color, -1)
26
27     label = classes[class_id]
28     caption = f'{label} {int(score * 100)}% ID: {id_}'
29     (tw, th), _ = cv2.getTextSize(text=caption,
30                                         fontFace=cv2.FONT_HERSHEY_SIMPLEX,
31                                         fontScale=size,
32                                         thickness=text_thickness)
33     th = int(th * 1.2)
34
35     cv2.rectangle(det_img, (x1, y1), (x1 + tw, y1 - th), color, -1)
36     cv2.rectangle(mask_img, (x1, y1), (x1 + tw, y1 - th), color, -1)
37     cv2.putText(det_img, caption, (x1, y1),
38                 cv2.FONT_HERSHEY_SIMPLEX, size,
39                 (255, 255, 255),
40                 text_thickness, cv2.LINE_AA)
41
42     cv2.putText(mask_img, caption, (x1, y1),
43                 cv2.FONT_HERSHEY_SIMPLEX, size,
44                 (255, 255, 255),
45                 text_thickness, cv2.LINE_AA)
46
47     return cv2.addWeighted(mask_img, mask_alpha, det_img, 1 - mask_alpha, 0)

```

**7. Xây dựng hàm chạy tracking với một video:** Tracking thường được ứng dụng với dữ liệu video. Vì vậy, ta sẽ xây dựng một hàm nhận vào là một video và trả về kết quả tracking, kèm theo video được vẽ kết quả tracking lên. Cách cài đặt như sau:

```

1 def video_tracking(video_path, detector, tracker,
2                     is_save_result=False, save_dir='tracking_results'):
3     cap = cv2.VideoCapture(video_path)
4     width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
5     height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
6
7     if is_save_result:
8         os.makedirs(save_dir, exist_ok=True)
9         # Get the video properties
10        fps = int(cap.get(cv2.CAP_PROP_FPS))
11
12        # Define the codec and create the video writer
13        fourcc = cv2.VideoWriter_fourcc(*'MJPG')
14
15        save_result_name = 'output_video.avi'
16        save_result_path = os.path.join(save_dir, save_result_name)
17        out = cv2.VideoWriter(save_result_path, fourcc, fps, (width, height))
18
19
20    all_tracking_results = []
21    tracked_ids = np.array([], dtype=np.int32)
22    while True:
23        ret, frame = cap.read()
24
25        if not ret:
26            break
27
28        detector_results = detector.detect(frame)
29        bboxes, scores, class_ids = detector_results
30
31        tracker_pred = tracker.inference(
32            origin_frame=frame,

```

```

33         bboxes=bboxes ,
34         scores=scores ,
35         class_ids=class_ids
36     )
37     if tracker_pred.size > 0:
38         bboxes = tracker_pred[:, :4]
39         class_ids = tracker_pred[:, 4].astype(int)
40         conf_scores = tracker_pred[:, 5]
41         tracking_ids = tracker_pred[:, 6].astype(int)
42
43         # Get new tracking IDs
44         new_ids = np.setdiff1d(tracking_ids, tracked_ids)
45
46         # Store new tracking IDs
47         tracked_ids = np.concatenate((tracked_ids, new_ids))
48
49         result_img = draw_detection(
50             img=frame,
51             bboxes=bboxes,
52             scores=conf_scores,
53             class_ids=class_ids,
54             ids=tracking_ids
55         )
56     else:
57         result_img=frame
58
59     all_tracking_results.append(tracker_pred)
60
61     if is_save_result == 1:
62         out.write(result_img)
63
64     # Break the loop if 'q' is pressed
65     if cv2.waitKey(25) & 0xFF == ord('q'):
66         break
67
68     # Release video capture
69     cap.release()
70     if is_save_result:
71         out.release()
72     cv2.destroyAllWindows()
73
74     return all_tracking_results

```

Theo đó, hàm trên sẽ thực hiện duyệt qua từng frame trong video đọc được, sau đó áp dụng detection và tracking. Kết quả tracking của frame đó sẽ được lưu lại vào một list. Đồng thời, ta thực hiện vẽ kết quả tracking lên ảnh frame, từ đó có thể tổng hợp thành một video với kết quả tracking.

8. **Thực hiện tracking:** Tổng hợp các thành phần đã cài đặt, ta sẽ ứng dụng để chạy với một video mẫu (các bạn có thể tải video này tại [đây](#)). Code triển khai như sau:

```

1 yolo_model_path = 'yolov8_mot_det.pt'
2
3 detector = YOLOv8(yolo_model_path)
4 tracker = DeepSORT()
5
6 video_path = '/content/CityRoam.mp4'
7 all_tracking_results = video_tracking(
8     video_path,
9     detector,
10    tracker,

```

```

11         is_save_result=True
12     )

```

Khi thực thi xong đoạn code trên, chúng ta có thể lấy file video có kết quả tracking theo đường dẫn '/content/tracking\_results/output\_video.avi', các bạn cũng có thể sử dụng đoạn code này để chạy video trực tiếp trên colab cho thuận tiện theo dõi: Đầu tiên, chuyển đổi file video từ đuôi .avi sang đuôi .mp4:

```

1 from IPython.display import HTML
2 from base64 import b64encode
3 import os
4
5 # Input video path
6 output_video_path = 'tracking_results/output_video.avi'
7
8 # Compressed video path
9 compressed_path = 'tracking_results/result_compressed.mp4'
10
11 os.system(f"ffmpeg -i {output_video_path} -vcodec libx264 {compressed_path}")

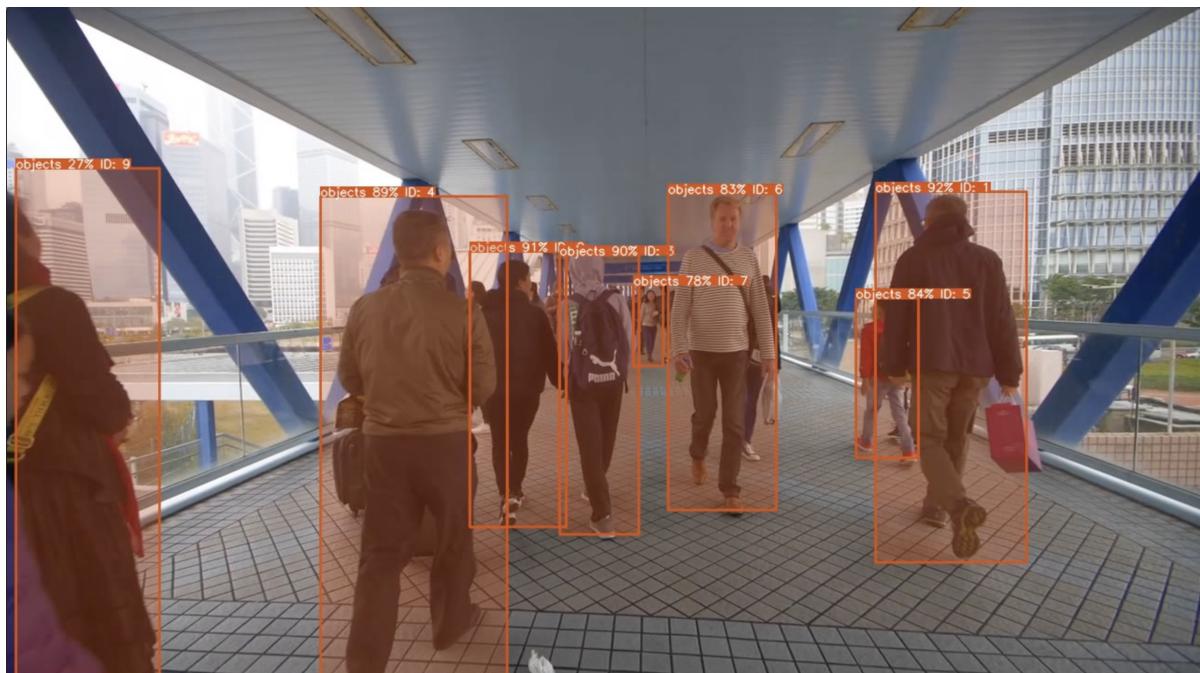
```

Sau đó, chạy code hiển thị video:

```

1 # Show video
2 mp4 = open(compressed_path, 'rb').read()
3 data_url = "data:video/mp4;base64," + b64encode(mp4).decode()
4 HTML("""
5 <video width=600 controls>
6     <source src=\"%s\" type=\"video/mp4\">
7 </video>
8 """ % data_url)

```



Hình 2: Ảnh cắt từ video đã qua tracking từ chương trình chúng ta xây dựng. Các bạn có thể coi full video tại [đây](#).

## Phần III: Câu hỏi trắc nghiệm

1. Bài toán Object Tracking là gì?
  - (a) Theo dõi các đối tượng trong một chuỗi hình ảnh.
  - (b) Phân loại các đối tượng trong một hình ảnh.
  - (c) Phát hiện các đối tượng động trong môi trường 3D.
  - (d) Dự đoán hướng di chuyển của đối tượng.
2. Input và Output của một hệ thống object tracking gồm có những gì?
  - (a) Nhận vào dữ liệu thời gian thực, trả về kết quả dự đoán vị trí của vật thể trong tương lai.
  - (b) Nhận vào ảnh đã được tiền xử lý, trả về kết quả phân loại.
  - (c) Nhận vào các frame ảnh của video, trả về các vật thể track được (trajectories).
  - (d) Nhận vào dữ liệu sensor, trả về ảnh 3D của vật thể track được.
3. Trong Single Object Tracking, mục tiêu chính là gì?
  - (a) Theo dõi nhiều đối tượng cùng một lúc.
  - (b) Theo dõi một đối tượng duy nhất qua các khung hình.
  - (c) Nhận dạng tất cả các đối tượng trong hình ảnh.
  - (d) Phân loại các đối tượng dựa vào hình dạng.
4. Multi Object Tracking khác Single Object Tracking như thế nào?
  - (a) Theo dõi đồng thời nhiều đối tượng.
  - (b) Chỉ tập trung vào một đối tượng.
  - (c) Sử dụng mạng nơ-ron sâu cho việc theo dõi.
  - (d) Phân loại các đối tượng theo màu sắc.
5. Trong YOLOv8, "non-maximum suppression" (NMS) được sử dụng để làm gì?
  - (a) Tăng tốc độ xử lý hình ảnh.
  - (b) Giảm kích thước của model.
  - (c) Loại bỏ các bounding box trùng lặp.
  - (d) Cải thiện độ chính xác của việc phân loại đối tượng.
6. Điểm mạnh của YOLOv8 so với các phiên bản trước là gì?
  - (a) Tốc độ xử lý chậm hơn.
  - (b) Độ chính xác thấp hơn trong phát hiện đối tượng.
  - (c) Tốc độ xử lý nhanh hơn và độ chính xác cao hơn.
  - (d) Sử dụng ít dữ liệu hơn cho việc huấn luyện.
7. Khi nào nên sử dụng Single Object Tracking thay vì Multi Object Tracking?
  - (a) Khi cần theo dõi nhiều đối tượng cùng lúc.
  - (b) Khi chỉ cần tập trung vào một đối tượng duy nhất.
  - (c) Khi không cần độ chính xác cao.

- (d) Khi dữ liệu đầu vào là hình ảnh tĩnh.
8. DeepSORT là gì?
- (a) Một thuật toán phân loại ảnh.
  - (b) Một thuật toán theo dõi đối tượng trong thời gian thực.
  - (c) Một mô hình học sâu dành cho dự đoán thời tiết.
  - (d) Một khung làm việc cho mạng xã hội.
9. Điểm nào sau đây là cải tiến quan trọng của thuật toán DeepSORT so với SORT?
- (a) Độ chính xác trong việc tracking thấp.
  - (b) Yêu cầu nhiều phần cứng.
  - (c) Tăng cường độ ổn định trong tracking với việc sử dụng đặc trưng từ model deep learning.
  - (d) Giảm độ phức tạp trong việc tính toán.
10. Đâu là khó khăn chính của bài toán object tracking?
- (a) Độ chính xác cao trong môi trường phức tạp.
  - (b) Cần nguồn resource lớn.
  - (c) Duy trì sự ổn định khi tracking nhiều object với nhiều kích cỡ khác nhau.
  - (d) Đơn giản hóa độ phức tạp tính toán.

- *Hết* -