Object Tracking Using YOLOv8

May 31, 2025

Abstract

This report details the implementation of an object tracking system using the YOLOv8 model from Ultralytics. Two tracking approaches are covered: single object tracking that locks onto and follows one person in a video, and multiple object tracking (MOT) using the ByteTrack algorithm integrated with YOLOv8 to track multiple persons simultaneously. The code was implemented in Python and run in a Google Colab environment, leveraging OpenCV for video processing.

1 Introduction

Object tracking is a fundamental task in computer vision where the goal is to locate and follow objects of interest across video frames. In many applications like surveillance, autonomous driving, and human-computer interaction, accurate and real-time tracking is crucial. This project utilizes the YOLOv8 (You Only Look Once version 8) model for object detection combined with tracking logic to monitor human movement in videos.

2 YOLOv8 Model

YOLOv8 is the latest evolution of the YOLO object detection family, developed by Ultralytics. It is designed to be both fast and accurate, suitable for real-time applications. The `yolov8n.pt` model used here is the "nano" variant, optimized for speed and efficiency with lightweight architecture.

YOLOv8 outputs bounding boxes with class predictions and confidence scores. For tracking, we focus on the ‘person‘ class (class ID 0).

3 Single Object Tracking Implementation

3.1 Overview

The single object tracking algorithm works by:

- Detecting all persons in each frame.
- Locking onto the first detected person in the video.
- Tracking that person across frames by selecting the detected bounding box closest to the last tracked position.
- Drawing a path that represents the tracked person's trajectory.

3.2 Code Explanation

Listing 1: Single Object Tracking Core Logic

```
from ultralytics import YOLO
import cv2
import numpy as np

# Load YOLOv8 nano model
model = YOLO('yolov8n.pt')

# Open video file
cap = cv2.VideoCapture('/content/drive/MyDrive/
    Traking_video/input_video1.mp4')

# Get video properties
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))
fps = int(cap.get(5))

# Prepare output video writer
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('/content/drive/MyDrive/
    output_video.mp4', fourcc, fps, (frame_width,
    frame_height))

tracked_points = []
selected_person_center = None

```

```
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    results = model(frame)

    closest_center = None
    closest_distance = float('inf')
    person_detected = False

    # Loop through detections in the frame
    for result in results:
        boxes = result.boxes.xyxy.cpu().numpy()
        classes = result.boxes.cls.cpu().numpy()
        confidences = result.boxes.conf.cpu().numpy()

        for box, cls, conf in zip(boxes, classes,
           confidences):
            if cls == 0 and conf > 0.3:  # Person class
               with confidence threshold
                x1, y1, x2, y2 = map(int, box)
                center = ((x1 + x2) // 2, (y2 + y1) //
                   2)
                person_detected = True

                if selected_person_center is None:
                    selected_person_center = center
                    tracked_points.append(center)
                else:
                    # Calculate distance to previously
                       tracked point
                    distance = np.linalg.norm(np.array(
                       center) - np.array(tracked_points
                       [-1]))
                    if distance < closest_distance:
                        closest_distance = distance
                        closest_center = center

    if closest_center and closest_distance < 100:
        tracked_points.append(closest_center)

    # Draw tracking path in red lines
```

```
    if person_detected and len(tracked_points) > 1:
        for i in range(1, len(tracked_points)):
            cv2.line(frame, tracked_points[i-1],
               tracked_points[i], (0, 0, 255), 2)

    out.write(frame)

cap.release()
out.release()
```

3.3 Key Points

- The detection threshold is set at 0.3 to reduce false positives.
- Euclidean distance is used to associate the detected person closest to the last tracked point.
- The tracked path is visualized by connecting tracked points using red lines.

4 Multiple Object Tracking (MOT) Implementation

4.1 Overview

Multiple object tracking is more complex as it involves detecting and maintaining identities for multiple persons simultaneously. Here, YOLOv8 is combined with the `ByteTrack` algorithm, which efficiently handles data association and ID assignment.

4.2 Code Explanation

Listing 2: Multiple Object Tracking Core Logic

```
model = YOLO('yolov8n.pt')

cap = cv2.VideoCapture('/content/drive/MyDrive/
   Traking_video/input_video1.mp4')

frame_width = int(cap.get(3))
frame_height = int(cap.get(4))
```

```
fps = int(cap.get(5))

fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('/content/drive/MyDrive/Traking_video/output_video.mp4', fourcc, fps, (frame_width, frame_height))

colors = [...]  # List of distinct colors

track_history = {}

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    results = model.track(frame, persist=True, classes=[0], tracker='bytetrack.yaml')

    for result in results:
        boxes = result.boxes.xyxy.cpu().numpy()
        ids = result.boxes.id
        if ids is None:
            continue
        ids = ids.cpu().numpy()

        for box, id in zip(boxes, ids):
            id = int(id)
            x1, y1, x2, y2 = map(int, box)
            center = ((x1 + x2) // 2, (y2 + y1) // 2)

            if id not in track_history:
                color_index = len(track_history) % len(colors)
                track_history[id] = {'points': [center], 'color': colors[color_index]}
            else:
                track_history[id]['points'].append(center)

            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
```

```
                cv2.putText(frame, f'Person {id}', (x1, y1
                    -10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,
                    255, 0), 2)

                if len(track_history[id]['points']) > 1:
                    color = track_history[id]['color']
                    for i in range(1, len(track_history[id][
                        'points'])):
                        cv2.line(frame, track_history[id]['
                            points'][i-1], track_history[id][
                            'points'][i], color, 2)

    out.write(frame)

cap.release()
out.release()
```

4.3 Key Points

- ByteTrack assigns persistent IDs to detected persons enabling consistent tracking.
- A color-coded path is drawn for each person for easy visual differentiation.
- Track histories are stored in a dictionary keyed by person IDs.

5 Video Processing and Output

Both tracking approaches extract the video's width, height, and frame rate to configure the `VideoWriter`, ensuring output video properties match the input.

The output videos are saved back to Google Drive for easy access.

6 Conclusion

This project demonstrates practical application of YOLOv8 for real-time object tracking in videos. Single object tracking serves scenarios where following a specific target is required, while multiple object tracking supports more complex scenes with several moving persons. The integration with

Google Drive and Google Colab provides a scalable environment for testing on various videos.

7 References

- Jocher, G., et al. Ultralytics YOLOv8 GitHub repository. `https://github.com/ultralytics/ultralytics`
- Zhang, Y., et al. ByteTrack: Multi-Object Tracking by Associating Every Detection Box. `https://arxiv.org/abs/2110.06864`
- OpenCV Documentation. `https://opencv.org/`