AI VIET NAM – COURSE 2024

Object Detection Part II - Exercise

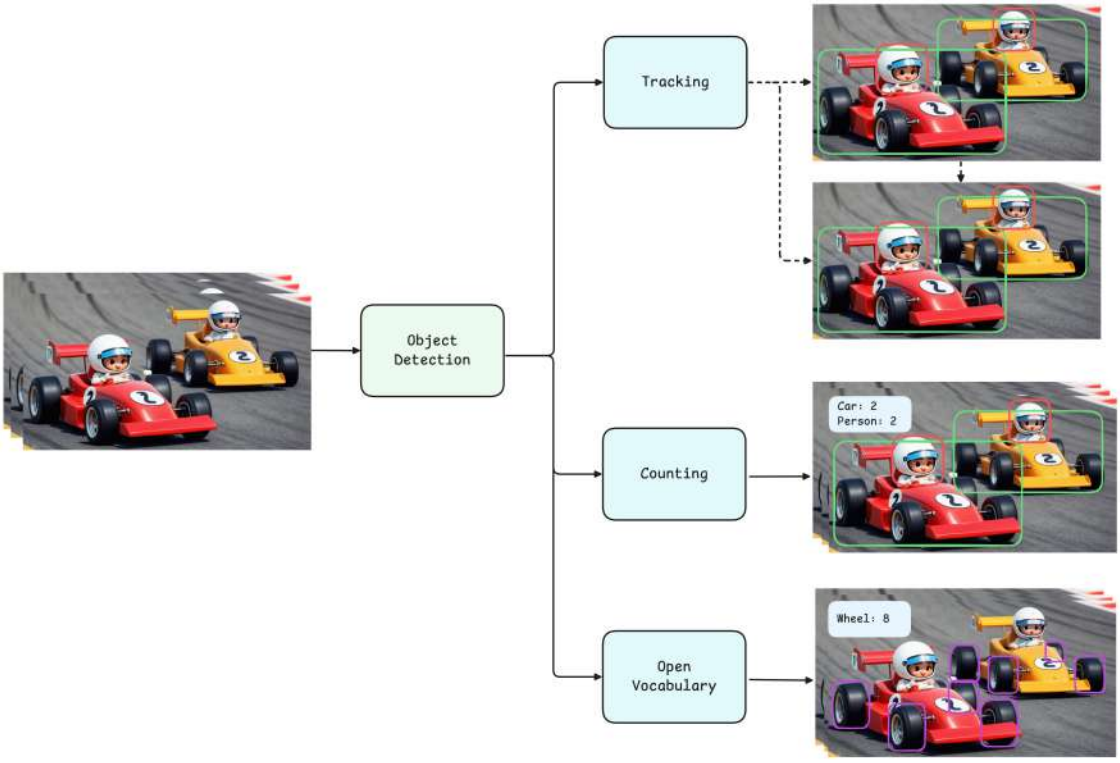
Minh-Duc Bui

Ngày 18 tháng 1 năm 2025

Phần I: Giới thiệu

Trong buổi hôm nay, chúng ta sẽ cùng nhau tìm hiểu 3 bài toán phổ biến ứng dụng Object Detection: Object Tracking, Object Counting, và Open Vocab Detection.

- **Tracking:** Theo dõi vị trí các đối tượng qua các khung hình liên tiếp.
- **Counting:** Đếm số lượng các đối tượng trong ảnh.
- **Open Vocabulary:** Nhận diện các đối tượng không nằm trong danh sách label cố định.



Hình 1: Hình ảnh minh họa cho 3 bài toán Tracking, Counting và Open Vocab Detection.

Phần II: Nội dung chính

Bài toán 1: Object Tracking

Version đơn giản

Trong bài toán này, chúng ta sẽ triển khai một hệ thống **Object Tracking** đơn giản sử dụng mô hình YOLO để theo dõi các đối tượng trong video. Dưới đây là các bước thực hiện:

Import các thư viện cần thiết Đầu tiên, chúng ta cần import các thư viện để xử lý video và thực hiện việc theo dõi đối tượng.

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

```

Tải mô hình YOLO và mở video Chúng ta sẽ tải mô hình YOLO và mở file video cần xử lý.

```
# Load the YOLO11 model
model = YOLO("yolo11l.pt")

# Open the video file
video_path = "samples/vietnam.mp4"
cap = cv2.VideoCapture(video_path)

```

Thiết lập VideoWriter để lưu video kết quả Tiếp theo, chúng ta thiết lập đối tượng `VideoWriter` để lưu video đã được xử lý với các đối tượng được theo dõi.

```
# Get video properties
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))

# Create VideoWriter object
video_name = video_path.split("/")[-1]
output_path = f"run/{video_name.split('.')[0]}_tracked.mp4"
fourcc = cv2.VideoWriter_fourcc(*"mp4v")
out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

```

Khởi tạo lịch sử theo dõi và vòng lặp xử lý frames Chúng ta sẽ sử dụng một `defaultdict` để lưu trữ lịch sử theo dõi các đối tượng và thực hiện vòng lặp qua từng frame của video để thực hiện việc theo dõi.

```python
# Store the track history
track_history = defaultdict(lambda: [])

# Loop through the video frames
while cap.isOpened():
    # Read a frame from the video
    success, frame = cap.read()

    if success:
        # Run YOLO11 tracking on the frame, persisting tracks between frames
        results = model.track(frame, persist=True, show=False)

        # Get the boxes and track IDs (with error handling)
        boxes = results[0].boxes.xywh.cpu()
        try:
            track_ids = results[0].boxes.id
            if track_ids is not None:
                track_ids = track_ids.int().cpu().tolist()
            else:
                track_ids = []  # No tracks found in this frame
        except AttributeError:
            track_ids = []  # Handle case where tracking fails

        # Visualize the results on the frame
        annotated_frame = results[0].plot()

        # Plot the tracks only if we have valid tracking data
        if track_ids:
            for box, track_id in zip(boxes, track_ids):
                x, y, w, h = box
                track = track_history[track_id]
                track.append((float(x), float(y)))  # x, y center point

                if len(track) > 120:  # retain 30 tracks for 30 frames
                    track.pop(0)

                # Draw the tracking lines
                points = np.hstack(track).astype(np.int32).reshape((-1, 1, 2))
                cv2.polylines(
                    annotated_frame,
                    [points],
                    isClosed=False,
                    color=(230, 230, 230),
                    thickness=4,
                )
        # Write the frame to output video
        out.write(annotated_frame)
    else:
        # Break the loop if the end of the video is reached
        break

# Release everything
cap.release()
out.release()
print(f"Video has been saved to {output_path}")

```

Kết quả sau khi xử lý Sau khi chạy chương trình, video kết quả sẽ được lưu tại đường dẫn đã chỉ định.

Version optimized

Phiên bản này được tối ưu hóa bằng cách sử dụng batching và cải thiện hiệu suất xử lý. Các bước thực hiện như sau:

Import các thư viện cần thiết Chúng ta sẽ import thêm các thư viện hỗ trợ như `argparse`, `tqdm` và thay đổi cách import `logger`.

```
import argparse
from collections import defaultdict
import cv2
import numpy as np
from tqdm import tqdm
from ultralytics import YOLO
from loguru import logger

```

Định nghĩa cấu hình và khởi tạo video Chúng ta sẽ định nghĩa một hàm để tải cấu hình và một hàm để khởi tạo các đối tượng `VideoCapture` và `VideoWriter`.

```
def load_config():
    """Load and return configuration settings"""
    return {
        "model_path": "yolo11x.pt",
        "track_history_length": 120,
        "batch_size": 64,
        "line_thickness": 4,
        "track_color": (230, 230, 230),
    }

def initialize_video(video_path):
    """Initialize video capture and writer objects"""
    cap = cv2.VideoCapture(video_path)
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(cap.get(cv2.CAP_PROP_FPS))

    video_name = video_path.split("/")[-1]
    output_path = f"run/{video_name.split('.')[0]}_tracked.mp4"
    fourcc = cv2.VideoWriter_fourcc(*"mp4v")
    out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

    return cap, out, output_path

```

Cập nhật lịch sử theo dõi Tiếp theo, chúng ta sẽ cập nhật lịch sử theo dõi và loại bỏ các track cũ.

```python
def update_track_history(
    track_history,
    last_seen,
    track_ids,
    frame_count,
    batch_size,
    frame_idx,
    history_length,
):
    """Update tracking history and remove old tracks"""
    current_tracks = set(track_ids)
    for track_id in list(track_history.keys()):
        if track_id in current_tracks:
            last_seen[track_id] = frame_count - (batch_size - frame_idx - 1)
        elif frame_count - last_seen[track_id] > history_length:
            del track_history[track_id]
            del last_seen[track_id]

```

Vẽ các đường theo dõi trên frame Chúng ta sẽ thêm các đường theo dõi các đối tượng lên từng frame video.

```python
def draw_tracks(frame, boxes, track_ids, track_history, config):
    """Draw tracking lines on frame"""
    if not track_ids:
        return frame

    for box, track_id in zip(boxes, track_ids):
        x, y, w, h = box
        track = track_history[track_id]
        track.append((float(x), float(y)))
        if len(track) > config["track_history_length"]:
            track.pop(0)

        points = np.hstack(track).astype(np.int32).reshape((-1, 1, 2))
        cv2.polylines(
            frame,
            [points],
            isClosed=False,
            color=config["track_color"],
            thickness=config["line_thickness"],
        )
    return frame

```

Xử lý một batch các frames Để tối ưu hóa hiệu suất, chúng ta sẽ xử lý nhiều frames cùng một lúc.

```python
def process_batch(model, batch_frames, track_history, last_seen, frame_count,
    config):
    """Process a batch of frames through YOLO model"""
    results = model.track(
        batch_frames,
```

```
        persist=True,
        tracker="botsort.yaml",
        show=False,
        verbose=False,
        iou=0.5,
    )

    processed_frames = []
    for frame_idx, result in enumerate(results):
        boxes = result.boxes.xywh.cpu()
        track_ids = (
            result.boxes.id.int().cpu().tolist() if result.boxes.id is not None
    else []
        )

        update_track_history(
            track_history,
            last_seen,
            track_ids,
            frame_count,
            len(batch_frames),
            frame_idx,
            config["track_history_length"],
        )

        annotated_frame = result.plot(font_size=4, line_width=2)
        annotated_frame = draw_tracks(
            annotated_frame, boxes, track_ids, track_history, config
        )
        processed_frames.append(annotated_frame)

    return processed_frames

```

Hàm chính để xử lý video Cuối cùng, chúng ta sẽ kết hợp tất cả các bước trên vào một hàm chính để xử lý toàn bộ video.

```
def main(video_path):
    """Main function to process video"""
    CONFIG = load_config()
    model = YOLO(CONFIG.get("model_path", "yolo11x.pt"))

    cap, out, output_path = initialize_video(video_path)
    track_history = defaultdict(lambda: [])
    last_seen = defaultdict(int)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    with tqdm(
        total=total_frames,
        desc="Processing frames",
        colour="green",
    ) as pbar:
        frame_count = 0
        batch_frames = []

        while cap.isOpened():
```

```
            success, frame = cap.read()
            if not success:
                break

            frame_count += 1
            batch_frames.append(frame)

            if len(batch_frames) == CONFIG["batch_size"] or frame_count == total_frames:
                try:
                    processed_frames = process_batch(
                        model,
                        batch_frames,
                        track_history,
                        last_seen,
                        frame_count,
                        CONFIG,
                    )
                    for frame in processed_frames:
                        out.write(frame)
                        pbar.update(1)
                    batch_frames = []

                except Exception as e:
                    logger.error(
                        f"Error when handling frames {frame_count - len(batch_frames) + 1} to {frame_count}: {str(e)}"
                    )
                    batch_frames = []
                    continue

    try:
        cap.release()
        out.release()
        cv2.destroyAllWindows()
        logger.info(f"{output_path}")
    except Exception as e:
        logger.error(f"{str(e)}")

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--video-path", type=str, default="samples/vietnam-2.mp4")
    args = parser.parse_args()

    main(args.video_path)

```

Giải thích các thay đổi và tối ưu hóa

- **Sử dụng batching**: Thay vì xử lý từng frame một, chúng ta xử lý nhiều frames cùng lúc để tăng hiệu suất.
- **Cấu hình linh hoạt**: Định nghĩa các tham số cấu hình trong hàm `load_config` để dễ dàng điều chỉnh.
- **Xử lý lỗi**: Sử dụng `try-except` để xử lý các lỗi có thể xảy ra trong quá trình xử lý frames.

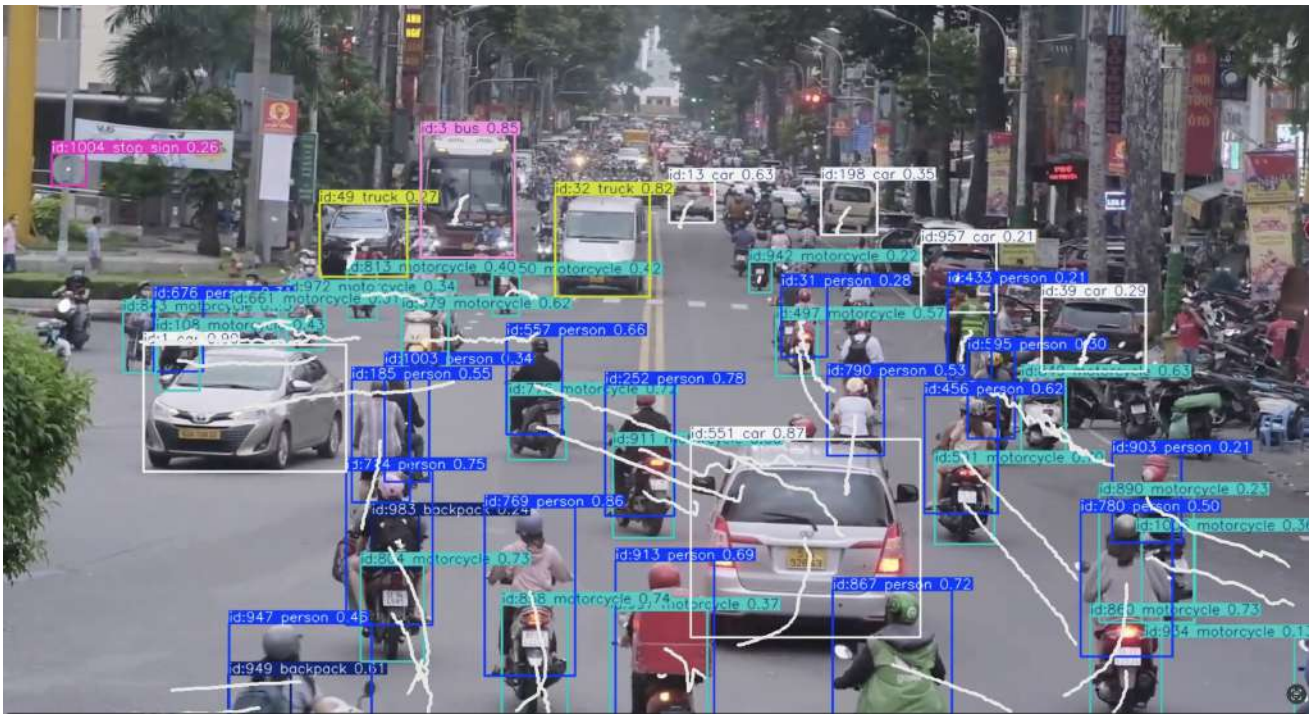
- **Thay thế logger**: Sử dụng `loguru` thay cho `logger` từ `src.utils` để đơn giản hóa việc logging.

Chạy chương trình Để chạy chương trình, bạn có thể sử dụng lệnh sau trong terminal:

```
python your_script.py --video-path "samples/vietnam-2.mp4"

```

Kết quả sau khi xử lý Sau khi chạy phiên bản tối ưu hóa, video kết quả sẽ được lưu tại đường dẫn đã chỉ định với hiệu suất xử lý cao hơn và các đối tượng được theo dõi mượt mà hơn.
Hình dưới đây minh họa output:



Hình 2: Output bài toán tracking (1).

Bài toán 2: Object Counting

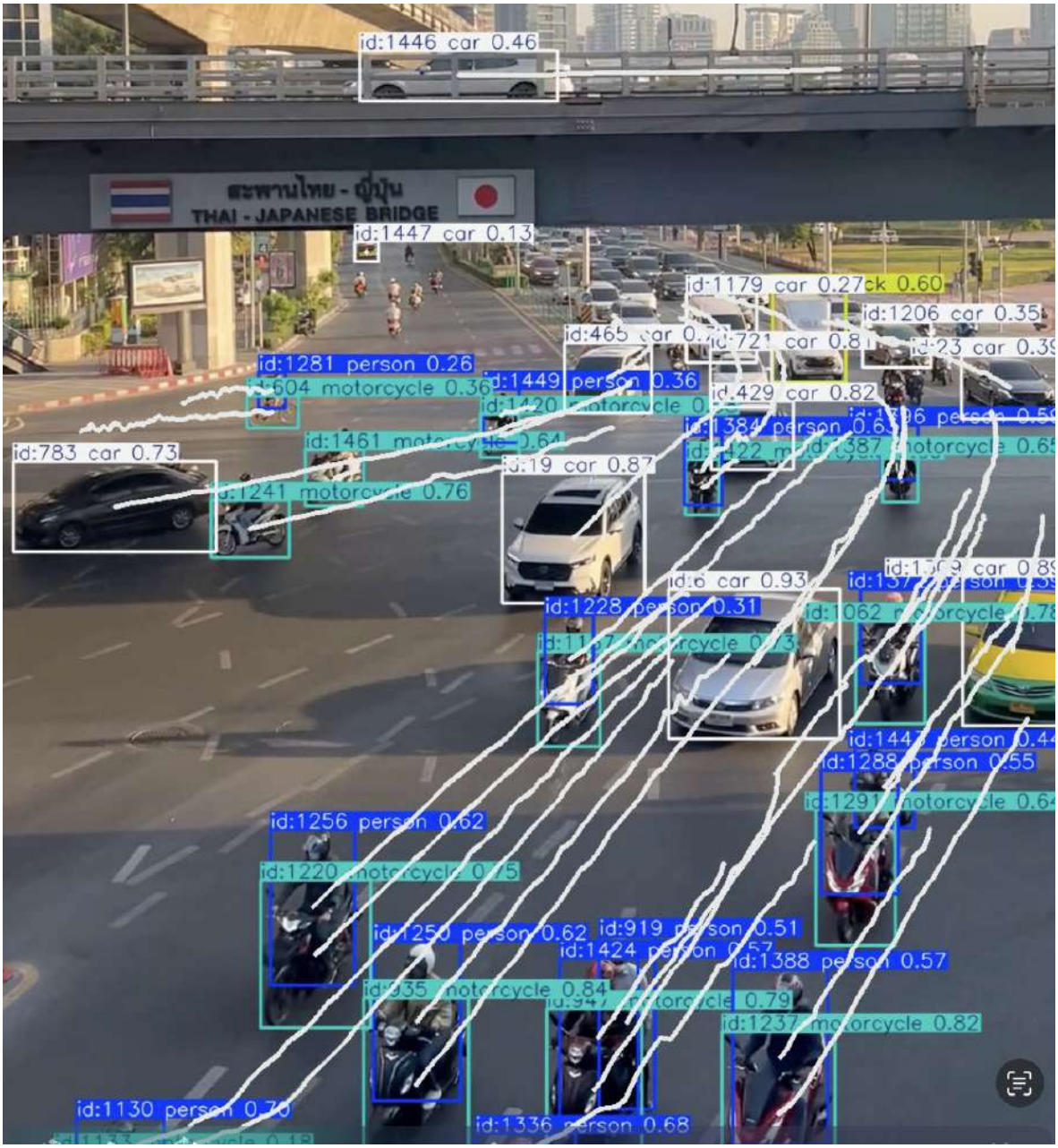
Trong bài toán này, chúng ta sẽ triển khai một hệ thống **Object Counting** để đếm số lượng đối tượng xuất hiện trong một khu vực cụ thể của video. Sử dụng mô hình YOLO, chúng ta sẽ xác định và đếm các đối tượng trong vùng được định nghĩa trước. Dưới đây là các bước thực hiện:

Import các thư viện cần thiết

Đầu tiên, chúng ta cần import các thư viện cần thiết để xử lý video và thực hiện việc đếm đối tượng.

```
import cv2
from ultralytics import solutions

```



Hình 3: Output bài toán tracking (2).

Khởi tạo video và định nghĩa vùng đếm

Chúng ta sẽ mở file video cần xử lý và định nghĩa các điểm để tạo thành vùng đếm đối tượng.

```
cap = cv2.VideoCapture("samples/highway.mp4")
assert cap.isOpened(), "Error reading video file"
w, h, fps = (
    int(cap.get(x))
    for x in (cv2.CAP_PROP_FRAME_WIDTH, cv2.CAP_PROP_FRAME_HEIGHT, cv2.
    CAP_PROP_FPS)
)
```

```
7
8 # Define region points
9 # region_points = [(20, 400), (1080, 400)]  # For line counting
10 region_points = [
11     (430, 700),
12     (1600, 700),
13     (1600, 1080),
14     (430, 1080),
15 ]  # For rectangle region counting: top left, top right, bottom right, bottom
       left
16
```

Giải thích:

- `cv2.VideoCapture`: Mở file video để xử lý.
- `region_points`: Định nghĩa các điểm để tạo thành vùng đếm đối tượng. Trong trường hợp này, chúng ta sử dụng một hình chữ nhật.

Thiết lập VideoWriter để lưu video kết quả

Chúng ta cần thiết lập đối tượng `VideoWriter` để lưu video đã được xử lý với các đối tượng được đếm.

```
1 # Video writer
2 video_writer = cv2.VideoWriter(
3     "./run/highway_counted.mp4", cv2.VideoWriter_fourcc(*"mp4v"), fps, (w, h)
4 )
5
```

Khởi tạo ObjectCounter

Chúng ta sẽ sử dụng `ObjectCounter` từ thư viện `ultralytics.solutions` để đếm các đối tượng trong vùng đã định nghĩa.

```
1 # Init ObjectCounter
2 counter = solutions.ObjectCounter(
3     show=False,  # Display the output
4     region=region_points,  # Pass region points
5     model="yolo11x.pt",  # model="yolo11n-obb.pt" for object counting using
      YOLO11 OBB model.
6 )
7
```

Giải thích:

- `show=False`: Không hiển thị cửa sổ kết quả.
- `region`: Các điểm định nghĩa vùng đếm đối tượng.
- `model`: Đường dẫn tới mô hình YOLO được sử dụng để phát hiện đối tượng.

Xử lý video và đếm đối tượng

Chúng ta sẽ đọc từng frame của video, áp dụng `ObjectCounter` để đếm các đối tượng và ghi lại kết quả vào video đầu ra.

```
# Process video
while cap.isOpened():
    success, im0 = cap.read()
    if not success:
        print(
            "Video frame is empty or video processing has been successfully completed."
        )
        break
    im0 = counter.count(im0)
    video_writer.write(im0)

cap.release()
video_writer.release()
cv2.destroyAllWindows()

```

Giải thích:

- Vòng lặp `while`: Đọc từng frame của video cho đến khi hết.
- `counter.count(im0)`: Áp dụng `ObjectCounter` để đếm đối tượng trong frame hiện tại.
- `video_writer.write(im0)`: Ghi frame đã được đếm đối tượng vào video đầu ra.

Chạy chương trình Để chạy chương trình, bạn có thể sử dụng lệnh sau trong terminal:

```
python your_counting_script.py

```

Kết quả sau khi xử lý

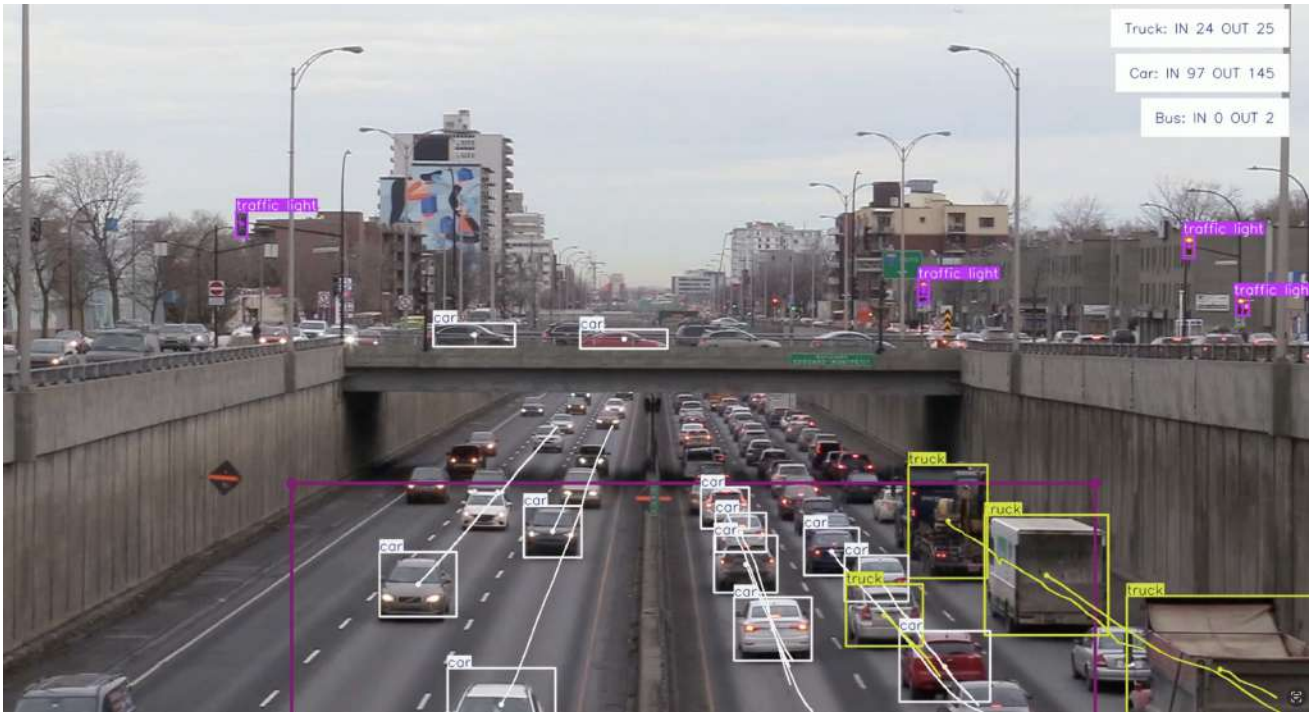
Sau khi chạy chương trình, video kết quả sẽ được lưu tại đường dẫn đã chỉ định với các đối tượng được đếm và đánh dấu trong vùng đã định nghĩa.
Hình dưới đây minh họa kết quả của mô hình sau khi đếm các đối tượng trong video.

Bài toán 3: Open Vocab Detection

Trong bài toán này, chúng ta sẽ triển khai một hệ thống **Open Vocab Detection** để phát hiện các đối tượng trong hình ảnh dựa trên các lớp tùy chỉnh. Sử dụng mô hình YOLO-World, chúng ta có thể dễ dàng định nghĩa và phát hiện các lớp đối tượng theo nhu cầu cụ thể. Dưới đây là các bước thực hiện:

Import các thư viện cần thiết

Đầu tiên, chúng ta cần import các thư viện cần thiết để thực hiện việc phát hiện đối tượng.



Hình 4: Kết quả của mô hình Object Counting sau khi xử lý video.

```
from ultralytics import YOLOWorld
from ultralytics.engine.results import Boxes

from src.utils import save_detection_results

```

Khởi tạo mô hình YOLO-World

Chúng ta sẽ khởi tạo mô hình YOLO-World với trọng số được huấn luyện trước.

```
# Initialize a YOLO-World model
model = YOLOWorld("yolov8s-world.pt")
```

Giải thích:

- `YOLOWorld`: Một phiên bản mở rộng của mô hình YOLO, hỗ trợ phát hiện đối tượng với các lớp tùy chỉnh.
- `"yolov8s-world.pt"`: Đường dẫn tới file trọng số của mô hình đã được huấn luyện trước.

Định nghĩa các lớp tùy chỉnh

Chúng ta có thể định nghĩa các lớp đối tượng mà chúng ta muốn mô hình phát hiện. Trong ví dụ này, chúng ta sẽ chỉ định lớp **bus**.

```
# Define custom classes
model.set_classes(["bus"])  # <--------- Change this to the class you want to
    detect

```

Giải thích:

- `set_classes`: Hàm này cho phép chúng ta định nghĩa các lớp đối tượng mà mô hình sẽ phát hiện. Bạn có thể thay đổi danh sách này để phù hợp với nhu cầu của mình.

Thực hiện dự đoán trên một hình ảnh

Chúng ta sẽ sử dụng mô hình đã được cấu hình để thực hiện dự đoán trên một hình ảnh cụ thể.

```
# Execute prediction on an image
results: Boxes = model.predict("samples/bus.jpg")

```

Giải thích:

- `model.predict`: Hàm này thực hiện việc dự đoán các đối tượng trong hình ảnh đầu vào.
- `"samples/bus.jpg"`: Đường dẫn tới hình ảnh mà chúng ta muốn phát hiện đối tượng.
- `results`: Kết quả trả về bao gồm các bounding boxes và các thông tin liên quan đến các đối tượng được phát hiện.

Lưu kết quả phát hiện đối tượng

Chúng ta sẽ lưu kết quả phát hiện đối tượng dưới dạng hình ảnh với các bounding boxes được vẽ lên.

```
# Save detection results as images
save_detection_results(results)

```

Giải thích:

- `save_detection_results`: Hàm này chịu trách nhiệm lưu kết quả phát hiện đối tượng vào các file hình ảnh. Hàm này có thể được định nghĩa trong module `src.utils`.

Chạy chương trình Để chạy chương trình, bạn có thể sử dụng lệnh sau trong terminal:

```
python your_open_vocab_detection_script.py

```

Kết quả sau khi phát hiện

Sau khi chạy chương trình, hình ảnh kết quả sẽ được lưu với các bounding boxes được vẽ lên các đối tượng được phát hiện.

Hình dưới đây minh họa kết quả của mô hình sau khi phát hiện đối tượng **bus** trong hình ảnh.

Hình 5: Kết quả của mô hình Open Vocab Detection sau khi phát hiện đối tượng **phone, glasses, mask** trong hình ảnh.

Giải thích thêm:

- Đảm bảo rằng mô hình YOLO-World đã được tải xuống và đường dẫn `model` trong `YOLOWorld` là chính xác.
- Các lớp đối tượng có thể được thay đổi bằng cách chỉnh sửa danh sách trong `set_classes`.
- Kết quả phát hiện đối tượng sẽ được lưu trong thư mục được chỉ định bởi hàm `save_detection_results`.

Phần III: Câu hỏi trắc nghiệm

1. Trong quá trình **Object Tracking** (Bài toán 1), thông tin `track_id` được sử dụng với mục đích gì?
   (*a*) Hiển thị màu sắc của bounding box.
   (*b*) Tự động thay đổi kích thước khung hình.
   (*c*) Phân biệt các đối tượng khác nhau khi di chuyển qua nhiều khung hình.
   (*d*) Tăng tốc độ xử lý video.

2. **Bài toán 2 (Object Counting)** thường được ứng dụng để làm gì trong thực tế?
   (*a*) Làm nổi bật các vật thể có kích thước lớn trong ảnh.
   (*b*) Phân loại toàn bộ hình ảnh.
   (*c*) Đếm số lượng các đối tượng trong một vùng hoặc khung hình.
   (*d*) Tăng độ phân giải của ảnh.

3. Trong **Bài toán 1 (Object Tracking)**, hàm `draw_tracks` trong mã nguồn có tác dụng gì?
   (*a*) Vẽ thêm chữ watermark lên khung hình.
   (*b*) Tô màu toàn bộ đối tượng được phát hiện.
   (*c*) Vẽ đường nối theo quỹ đạo di chuyển của đối tượng qua các khung hình.
   (*d*) Giảm nhiễu cho video.

4. Khi sử dụng **batching** (Bài toán 1 phiên bản *optimized*) trong **Object Tracking**, lợi ích chính là gì?
   (*a*) Giảm chất lượng đầu ra.
   (*b*) Tăng tốc độ và hiệu suất xử lý.
   (*c*) Tạo ra nhiều bounding box bị trùng lặp.
   (*d*) Làm méo khung hình video.

5. Trong **Bài toán 2 (Object Counting)**, biến `region_points` được sử dụng để:
   (*a*) Xác định khu vực hoặc đường kẻ dùng để đếm đối tượng.
   (*b*) Xóa nền của hình ảnh.
   (*c*) Thay đổi kích thước của từng đối tượng.
   (*d*) Gán nhãn cho các đối tượng hiếm.

6. **Bài toán 3 (Open Vocab Detection)** cho phép chúng ta:
   (*a*) Chỉ nhận diện các lớp đã được huấn luyện cố định.
   (*b*) Phát hiện các đối tượng không có trong danh sách nhãn cố định.
   (*c*) Giảm số lượng nhãn cần huấn luyện.
   (*d*) Bỏ qua các bounding box có kích thước nhỏ hơn ngưỡng.

7. Trong **Bài toán 2 (Object Counting)**, thư viện `ultralytics.solutions` được sử dụng để:
   (*a*) Hiển thị tất cả các mô hình YOLO có sẵn.
   (*b*) Giảm độ phân giải của video xuống mức tối thiểu.

(*c*) Huấn luyện mô hình từ đầu (scratch).

(*d*) Triển khai `ObjectCounter` giúp đếm số lượng đối tượng trong một vùng.

8. **Đường dẫn** đến video đầu vào và video kết quả (xuất ra) trong các **bài toán xử lý video** (như Bài toán 1 và Bài toán 2) được thiết lập chủ yếu thông qua:

   (*a*) `cv2.VideoCapture` và `cv2.VideoWriter`.

   (*b*) `numpy.load` và `numpy.save`.

   (*c*) `matplotlib.imread` và `matplotlib.imsave`.

   (*d*) `cv2.imshow` và `cv2.imwrite`.

9. Trong **hàm** `process_batch` (Bài toán 1 phiên bản *optimized*), biến `results` được trả về dùng để:

   (*a*) Lưu video đã chỉnh sửa.

   (*b*) Ghi lại lịch sử các đường tracking.

   (*c*) Hiển thị trực tiếp video ra màn hình.

   (*d*) Chứa thông tin về `boxes` và `track_ids` sau khi chạy mô hình.

10. Một yếu tố quan trọng giúp nâng cao **hiệu suất tính toán** trong **Bài toán 1 (Object Tracking), phiên bản optimized** là:

    (*a*) Dừng theo dõi đối tượng ngay sau khi phát hiện lần đầu.

    (*b*) Sử dụng cùng lúc nhiều mô hình YOLO khác nhau.

    (*c*) Áp dụng **batching** để xử lý nhiều khung hình trong một lần suy luận.

    (*d*) Tăng độ phân giải của video lên gấp đôi.

1 Phụ lục

1. **Code và data:** Code và data sample các bạn có thể tìm ở github tại đây.
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại đây (Lưu ý: Sáng thứ 3 khi hết deadline phần bài tập, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Rubric:**

Object Detection Exercise - Rubric		
Bài toán	**Kiến Thức**	**Đánh Giá**
1	**Object Tracking:** - Kiến thức về cách sử dụng mô hình YOLO (phiên bản track) để phát hiện và theo dõi đối tượng trong video. - Hiểu cách thức lưu trữ lịch sử đối tượng, `track_id`, và vẽ đường di chuyển (`polylines`). - Biết áp dụng kỹ thuật **batching** để tối ưu tốc độ xử lý video.	- Nắm vững quá trình xử lý từng khung hình, gắn `track_id` cho đối tượng. - Triển khai thành công vòng lặp đọc video, gán ID và vẽ quỹ đạo di chuyển. - Sử dụng hiệu quả batching để tăng hiệu suất.
2	**Object Counting:** - Kiến thức về xác định khu vực đếm (định nghĩa `region_points`) và cách tích hợp với mô hình phát hiện đối tượng (YOLO, `ultralytics.solutions`). - Sử dụng `ObjectCounter` để đếm số lượng đối tượng trong một vùng/video.	- Thiết lập được `region_points` hoặc đường line để xác định khu vực đếm. - Biết cách đọc/ghi video và ghi nhận số lượng đối tượng đếm được theo thời gian. - Áp dụng mô hình YOLO kết hợp `ObjectCounter` để hiển thị kết quả chính xác.
3	**Open Vocab Detection:** - Kiến thức về mô hình `YOLOWorld` và cách `set_classes` để phát hiện những lớp chưa có trong danh sách nhãn cố định. - Hiểu cơ chế nhận diện nhiều đối tượng mới mà không cần huấn luyện lại hoàn toàn.	- Sử dụng được `YOLOWorld` với các lớp tùy chỉnh. - Thiết lập mô hình dự đoán, phân tích kết quả phát hiện, và lưu kết quả (bounding boxes) thành file. - Đảm bảo tính mở rộng cho các lớp đối tượng mới.

- Hết -