Experiment- 9

Student Name: Kush Bhasin **UID:** 20BCS7677
Branch: BE(CSE) **Section/Group:** 20BCS-1/B
Semester: 7th
Subject Name: Computer Vision Lab
Subject Code: 20CSP- 422

1. **Aim:** Write a program to examine the performance of various pretrained deep learning models for real-time object tracking tasks.

2. **System Requirements:**
   - Python 3.9
   - Visual Studio Code

3. **Description:**
   Real-time object tracking is a crucial task in computer vision and has numerous applications such as surveillance systems, autonomous vehicles, and robotics. Some pre-trained deep learning models that have shown promising performance in real-time object tracking are:

   1. MobileNetV2: MobileNetV2 is a lightweight and efficient deep neural network architecture that is well-suited for real-time applications. It achieves a good balance between accuracy and speed, making it popular for object tracking on resource-constrained devices.
   2. YOLO (You Only Look Once) v8: YOLO v8 is a fast and accurate object detection model that can be used for object tracking. By processing the entire image in a single pass, YOLO v8 achieves real-time performance. Tracking can be achieved by associating detected objects across consecutive frames.
   3. EfficientNet: EfficientNet is a family of deep neural network architectures that are known for their excellent trade-off between accuracy and computational efficiency. These models, such as EfficientNet-B0 to EfficientNet-B7, can be used as feature extractors for object tracking tasks.
   4. Faster R-CNN: Faster R-CNN is a widely used object detection model that can be adapted for object tracking. By extracting features from the pretrained backbone network and combining them with a tracking algorithm, real-time object tracking can be achieved.
   5. SiamRPN/SiamMask: SiamRPN (Siamese Region Proposal Network) and SiamMask are deep learning-based tracking algorithms that can track objects in real-time. They employ Siamese networks and template matching techniques to estimate the object's position and perform online adaptation to handle appearance changes.

6. DeepSORT: DeepSORT (Deep Learning-based SORT) is a combination of the SORT (Simple Online and Real-time Tracking) algorithm with deep appearance features. It utilizes a deep neural network, such as a CNN, to extract appearance features and combines them with motion information for robust and real-time object tracking.

7. MDNet: MDNet (Multi-Domain Network) is a deep learning-based tracking algorithm that learns a discriminative model for the target object online. It leverages a convolutional neural network to extract features and adaptively updates the model to handle appearance variations and occlusions.

4. Steps:

1. Import the necessary libraries, including deep learning frameworks (e.g., TensorFlow, PyTorch) and OpenCV.
2. Load a pretrained deep learning model for object detection and tracking. This can be a model such as YOLO, SSD, or Faster R-CNN.
3. Initialize the video stream or capture a video file for real-time processing.
4. Read each frame from the video stream and preprocess it if required.
5. Pass the preprocessed frame through the deep learning model to detect and track objects.
6. Display the output frame with bounding boxes or other visual indicators representing the tracked objects.
7. Repeat steps 4-6 for subsequent frames until the video stream ends or the video file is fully processed.
8. Calculate and display the performance metrics, such as tracking accuracy, processing time, and frame rate.
9. Analyze the results and compare the performance of different pretrained deep learning models for object tracking.

5. Code:

```
import datetime
from ultralytics import YOLO
import cv2
from deep_sort_realtime.deepsort_tracker import DeepSort
from google.colab.patches import cv2_imshow

CONFIDENCE_THRESHOLD = 0.8
GREEN = (0, 255, 0)
WHITE = (255, 255, 255)

video_cap = cv2.VideoCapture("park.mp4")
```

```
model = YOLO("yolov8n.pt")
tracker = DeepSort(max_age=50)

while True:
    start = datetime.datetime.now()
    ret, frame = video_cap.read()

    if not ret:
        break

    detections = model(frame)[0]
    results = []

    for data in detections.boxes.data.tolist():
        confidence = data[4]

        if float(confidence) < CONFIDENCE_THRESHOLD:
            continue

        xmin, ymin, xmax, ymax = int(data[0]), int(data[1]), int(data[2]),
int(data[3])
        class_id = int(data[5])
        results.append([[xmin, ymin, xmax - xmin, ymax - ymin], confidence,
class_id])

    tracks = tracker.update_tracks(results, frame=frame)

    for track in tracks:
        if not track.is_confirmed():
            continue

        track_id = track.track_id
        ltrb = track.to_ltrb()
        xmin, ymin, xmax, ymax = int(ltrb[0]), int(ltrb[1]), int(ltrb[2]),
int(ltrb[3])

        cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), GREEN, 2)
        cv2.rectangle(frame, (xmin, ymin - 20), (xmin + 20, ymin), GREEN, -1)
        cv2.putText(frame, str(track_id), (xmin + 5, ymin - 8),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, WHITE, 2)

    end = datetime.datetime.now()
    print(f"Time to process 1 frame: {(end - start).total_seconds() * 1000:.0f}
milliseconds")
```

```
        fps = f"FPS: {1 / (end - start).total_seconds():.2f}"
        cv2.putText(frame, fps, (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 255), 8)

        cv2_imshow(frame)

        if cv2.waitKey(1) == ord("q"):
            break

    video_cap.release()
    cv2.destroyAllWindows()
```

6. Output:

```
0: 384x640 2 persons, 1 bench, 187.7ms
Speed: 5.3ms preprocess, 187.7ms inference, 1.4ms postprocess per image at shape (1, 3, 384, 640)
Time to process 1 frame: 341 milliseconds
```