Experiment- 3.2

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:**

YOLO Object Detection:
Utilizes the YOLO model (YOLO from Ultralytics) for object detection.
Detects objects within each frame with bounding box coordinates, confidence scores, and class labels.
Applies a confidence threshold to filter out detections with low confidence.

DeepSort Object Tracking:
Implements the DeepSort tracking algorithm (DeepSort from deep_sort_realtime.deepsort_tracker).
Associates detected objects across frames to create tracks.
Utilizes a specified maximum age for tracks to handle object disappearance and re-appearance.

Process
Frame Processing Loop:
Iterates through each frame of the video stream.
Measures the time taken to process each frame.

Object Detection and Filtering:
Applies YOLO object detection to each frame.
Filters out detections below a specified confidence threshold.

Extracts relevant information, such as bounding box coordinates, confidence scores, and class IDs.

Object Tracking with DeepSort:
Updates the DeepSort tracker with the filtered detection results and the current frame.
Associates and tracks objects over consecutive frames.

Drawing Tracked Objects:
Draws rectangles around tracked objects on the frame.
Displays unique track IDs alongside the objects.

FPS Calculation and Display:
Calculates and displays the Frames Per Second (FPS) on each frame.
Provides a metric for the speed of real-time processing.

User Interaction:
Monitors for the 'q' key press to exit the real-time processing loop

4. Steps:

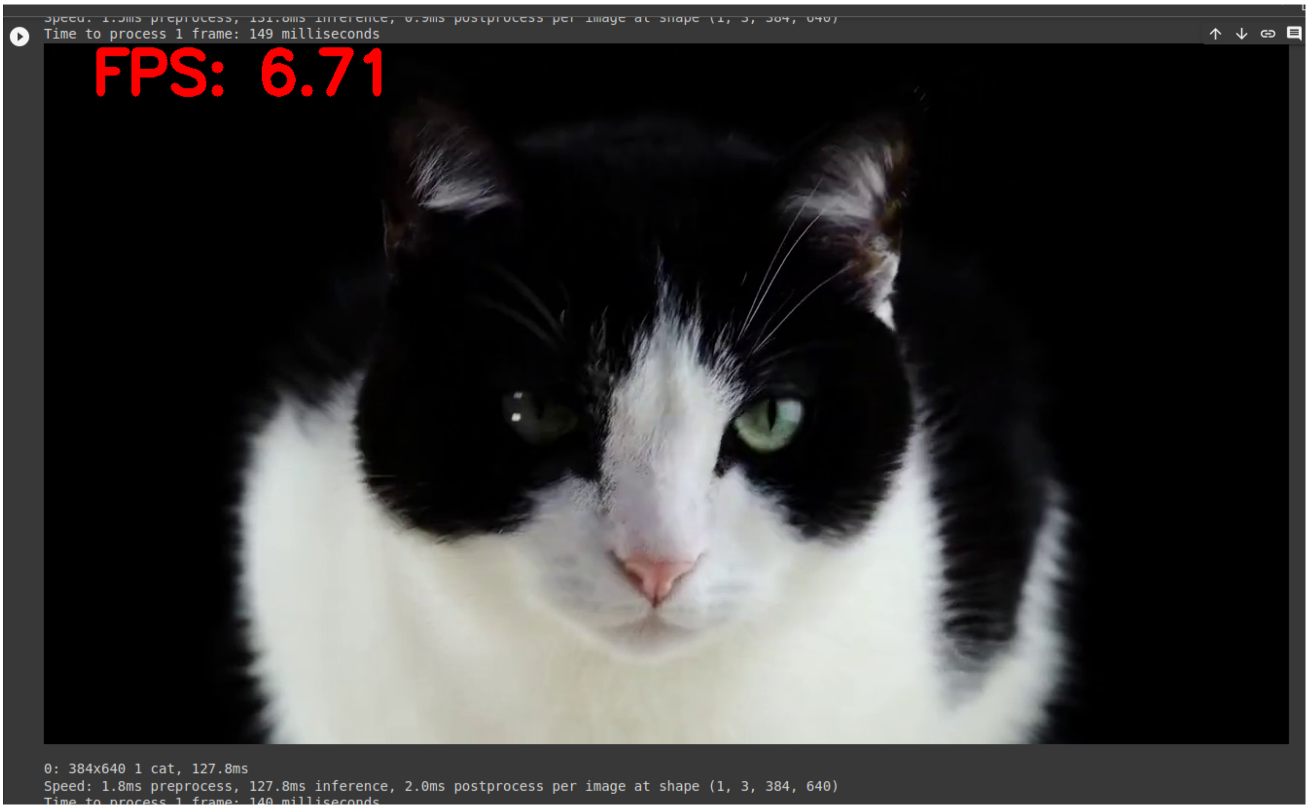
1. Import the necessary libraries, including deep learning frameworks (e.g., TensorFlow, PyTorch) and OpenCV.
2. Load a pretrained deep learning model for object detection and tracking. This can be a model such as YOLO, SSD, or Faster R-CNN.
3. Initialize the video stream or capture a video file for 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("content/drive/MyDrive/Videos/catvideo.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:



```
Speed: 1.5ms preprocess, 131.8ms inference, 0.9ms postprocess per image at shape (1, 3, 384, 640)
Time to process 1 frame: 149 milliseconds
```

```
0: 384x640 1 cat, 127.8ms
Speed: 1.8ms preprocess, 127.8ms inference, 2.0ms postprocess per image at shape (1, 3, 384, 640)
Time to process 1 frame: 140 milliseconds
```

```
0: 384x640 1 cat, 131.8ms Speed: 1.5ms preprocess, 131.8ms inference, 0.9ms postprocess per image at shape (1, 3, 384, 640) Time to process 1 frame: 149 milliseconds
```