**Object Tracking with DeepSORT | By Shoeb Ahmad**

Object tracking is a computer vision technique that involves automatically tracking the movement and position of an object in a video sequence over time. The goal of object tracking is to maintain the identity of the object, even as it moves and changes appearance, and to accurately locate it in each frame of the video.

Deep SORT (Deep Simple Online Realtime Tracking) is a state-of-the-art object tracking algorithm that combines a deep learning-based object detector with a tracking algorithm to achieve high accuracy and robustness in crowded and complex environments.

Deep SORT is based on the SORT (Simple Online and Realtime Tracking) algorithm, which uses the Kalman filter and Hungarian algorithm to associate object detections across frames. Deep SORT extends SORT by incorporating a deep neural network-based object detector, typically a convolutional neural network (CNN), to improve the accuracy of object detection.

**How DeepSort works?**

Diagram

Description automatically generated

Fig 1.0 Image Ref :<https://www.researchgate.net/figure/Architecture-of-Deep-SORT-Simple-online-and-real-time-tracking-with-deep-association_fig2_353256407>

DeepSORT (Deep Learning-based SORT) is an extension of the popular object tracking algorithm called SORT (Simple Online and Realtime Tracking). DeepSORT adds deep appearance feature extraction and a matching process to the original SORT algorithm to improve its tracking accuracy and robustness.

The architecture of DeepSORT can be broken down into the following steps:

1. Detection: Use an object detection algorithm (such as YOLO) to detect objects in each frame of a video.
2. Feature extraction: Extract a deep appearance feature vector for each detected object using a CNN-based feature extractor (such as ResNet).
3. Data association: Associate the detected objects across frames using a matching algorithm (such as the Hungarian algorithm) that takes into account both the location and appearance of the objects.
4. Track management: Manage the tracks by updating the state of each track (i.e., position and velocity) based on the associated objects and their appearance features.
5. Track pruning: Remove tracks that have not been associated with any objects for a certain number of frames or that have low confidence scores.

The deep appearance features used in DeepSORT are learned during training from a large dataset of object images. By incorporating appearance features in addition to the location and motion information used by the original SORT algorithm, DeepSORT is able to handle situations where objects may temporarily disappear or occlude each other, leading to more accurate and robust object tracking.

**Code level Implementation of DeepSort-**

here is an example of code-level implementation of object detection with DeepSORT in Python using the OpenCV, NumPy, and TensorFlow libraries:

import cv2  
import numpy as np  
import tensorflow as tf  
from deep\_sort import preprocessing  
from deep\_sort import nn\_matching  
from deep\_sort.detection import Detection  
from deep\_sort.tracker import Tracker  
from deep\_sort import generate\_detections  
from tensorflow.compat.v1 import ConfigProto  
from tensorflow.compat.v1 import InteractiveSession  
  
# Set up TensorFlow session  
config = ConfigProto()  
config.gpu\_options.allow\_growth = True  
session = InteractiveSession(config=config)  
  
# Load object detector  
model = tf.saved\_model.load("path/to/saved\_model")  
  
# Set up tracker  
metric = nn\_matching.NearestNeighborDistanceMetric("cosine", 0.2)  
tracker = Tracker(metric)  
  
# Open video file  
cap = cv2.VideoCapture("path/to/video\_file")  
  
while True:  
 # Read frame from video file  
 ret, frame = cap.read()  
 if not ret:  
 break  
  
 # Object detection  
 detections = []  
 boxes, scores = model(frame)  
 for i in range(boxes.shape[0]):  
 if scores[i] > 0.5:  
 detection = Detection(boxes[i], scores[i], preprocessing(frame, boxes[i]))  
 detections.append(detection)  
  
 # Object tracking  
 tracker.predict()  
 tracker.update(detections)  
  
 # Visualize results  
 for track in tracker.tracks:  
 if not track.is\_confirmed() or track.time\_since\_update > 1:  
 continue  
 bbox = track.to\_tlbr()  
 cv2.rectangle(frame, (int(bbox[0]), int(bbox[1])), (int(bbox[2]), int(bbox[3])), (255, 255, 0), 2)  
 cv2.putText(frame, "ID: " + str(track.track\_id), (int(bbox[0]), int(bbox[1]) - 5), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 0), 2)  
  
 # Display frame  
 cv2.imshow("Object Detection and Tracking", frame)  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
  
# Release video file and close windows  
cap.release()  
cv2.destroyAllWindows()

In this example, we first set up a TensorFlow session and load the object detection model from a saved model file. We then set up a DeepSORT tracker with a cosine distance metric and a threshold of 0.2. We then read frames from a video file and perform object detection using the loaded model. We create detection objects using the detection results and pass them to the tracker for data association and track management. Finally, we visualize the tracking results by drawing bounding boxes and track IDs on the frames and display them in a window. Note that this example assumes that the deep\_sort library and its dependencies are installed and accessible in the Python environment.

**Lets talk about why DeepSort Algorithm is popular…**

DeepSORT (Deep Learning-based SORT) has several advantages that make it better than other object tracking algorithms:



Fig 1.1

1. Improved tracking accuracy: By incorporating deep appearance features in addition to the location and motion information used by the original SORT algorithm, DeepSORT is able to handle situations where objects may temporarily disappear or occlude each other, leading to more accurate and robust object tracking.
2. Handling multiple objects: DeepSORT can track multiple objects simultaneously and maintain their identity over time, even in crowded scenes.
3. Real-time performance: DeepSORT is designed to operate in real-time, making it suitable for many applications such as surveillance, robotics, and self-driving cars.
4. Low computational cost: The use of a deep feature extractor in DeepSORT is computationally efficient and does not significantly increase the overall computational cost of the algorithm.
5. Easy to integrate with existing object detectors: DeepSORT can be easily integrated with existing object detectors, such as YOLO, to provide an end-to-end object detection and tracking pipeline.

**Limitations In DeepSort …..**

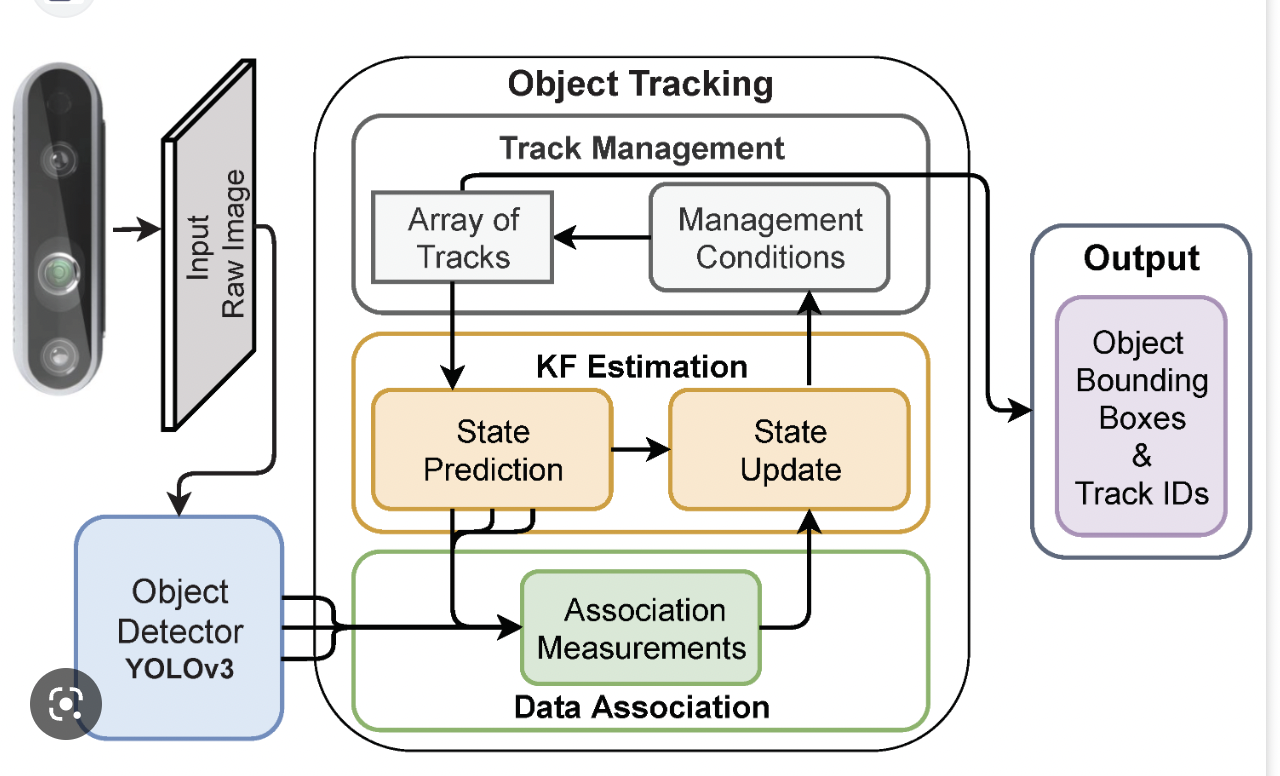


Fig 1.2 Image Ref -<https://www.mdpi.com/2076-3417/12/3/1319>

While DeepSORT is a powerful object tracking algorithm, there are still some limitations to its performance:

1. Dependence on object detection: DeepSORT relies on the accuracy of the object detection algorithm to identify and track objects. If the object detector produces inaccurate or inconsistent detections, DeepSORT’s tracking performance can be compromised.
2. High memory usage: The use of deep appearance features in DeepSORT can lead to high memory usage, which can become a limitation when tracking a large number of objects.
3. Sensitivity to parameter tuning: The performance of DeepSORT is sensitive to the choice of various parameters, such as the distance metric used for matching, the threshold for association, and the minimum track length. Fine-tuning these parameters for specific applications can be challenging and time-consuming.
4. Limited ability to handle appearance changes: While DeepSORT’s deep appearance features allow it to track objects with some degree of appearance changes, it may struggle to track objects that undergo drastic changes in appearance, such as changes in shape or texture.
5. Limited ability to handle occlusions: Like most object tracking algorithms, DeepSORT may struggle to track objects that are occluded by other objects, especially when those objects are of similar appearance.

Overall, while DeepSORT is a highly effective object tracking algorithm, it is important to consider its limitations and evaluate its performance in the context of specific use cases.