Q3d) Object tracker: SORT + DEEP SORT

- Dataset: Traffic junction videos you have collected already
- For the detections, use Faster RCNN and YOLO V2
- Implement a car counter using object tracking (SORT and DeepSORT)
- Document the results and observations
- Document the difference between SORT and DeepSORT
- Create ground truths if needed for evaluation
- Not needed to implement Faster RCNN/YOLO V2/SORT/Deep SORT from scratch

Faster RCNN:

- Faster R-CNN is a two-stage object detector used for object detection, introduced by Ross Girshick, Shaoqing Ren, Kaiming He, and Jian Sun in 2015.
- It consists of three main parts: **Convolution layers for feature extraction, Region Proposal Network (RPN) for predicting objects and bounding boxes, and a network for predicting object classes and bounding boxes based on the proposed regions.**

A brief description on Faster RCNN:

1. Region Proposal Network (RPN):

- Faster R-CNN introduces a Region Proposal Network (RPN) to generate region proposals on the given frame.
- The RPN takes an image as input and generates a set of regions (bounding boxes) that potentially contain objects.
- It slides a small window (typically 3x3 kernel) called an anchor box across the image at various locations and scales.
- For each anchor box, the RPN predicts the likelihood of it containing an object and refines its coordinates.
- These predicted regions serve as proposals for potential objects.

2. Feature Extraction:

- The input image is passed through a convolutional neural network (CNN) to extract a feature map.
- This feature map is fed into both the Region Proposal Network (RPN) and the subsequent stages of the network.

3. Region of Interest (RoI) Pooling:

- The proposed regions (bounding boxes) generated by the RPN are mapped back to the original feature map.
- RoI pooling is then applied to these regions to extract fixed-size feature maps.

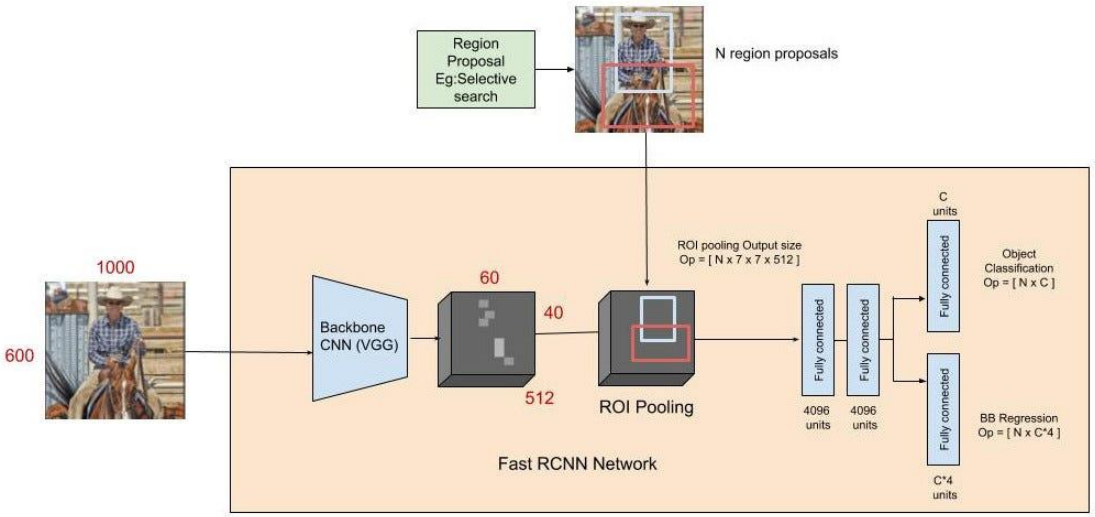
4. Object Classification and Bounding Box Regression:

- The fixed-size feature maps from the RoI pooling are passed through fully connected layers.
- These layers are responsible for two tasks:
    - **Object classification**: Determining the presence of objects within the proposed regions and classifying them into predefined categories.
    - **Bounding box regression**: Adjusting the coordinates of the bounding boxes to more accurately fit the objects.

5. Loss Function:

- The network is trained using a multi-task loss function, which combines the losses from the RPN and the object detection network.
- The RPN loss includes terms for both classification and bounding box regression.
- Similarly, the object detection network loss includes terms for classification and bounding box regression.

We are using Faster RCNN to detect objects in each frame of the video we are processing and then using the frames we recreate the video with the bounding boxes around the objects.



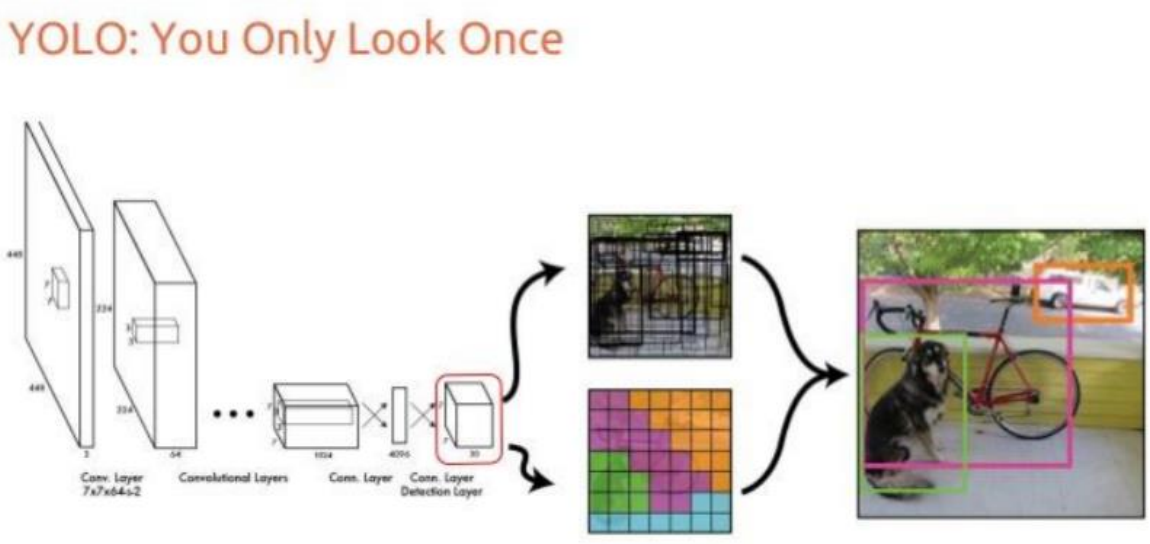
YOLO v8:

- YOLO (You Only Look Once) is a renowned family of real-time object detection models for computer vision, known for its speed and accuracy in detecting objects in images or video frames in a single forward pass through the neural network
- YOLO divides an image into a grid, predicting bounding boxes and class probabilities for each grid cell, enabling precise and rapid detections even when objects are partially obscured or small
- YOLOv8, the latest iteration of the YOLO models released in 2021, builds upon the success of its predecessors by incorporating advancements in computer vision, offering improved accuracy and speed compared to previous versions
- YOLOv8 achieves enhanced performance through a larger and deeper neural network architecture trained on extensive datasets, utilizing "swish activation" to aid network convergence, and implementing a novel backbone architecture called CSPDarknet to enhance information flow within the network
- This version also integrates features like data augmentation, learning rate schedules, and enhanced training strategies to boost overall performance, making it a cutting-edge object detection algorithm suitable for various computer vision applications
- Problems YOLOv8 can solve are Image Classification, Detection, Segmentation. It works with enhances efficiency and speed.

A brief description on YOLO:

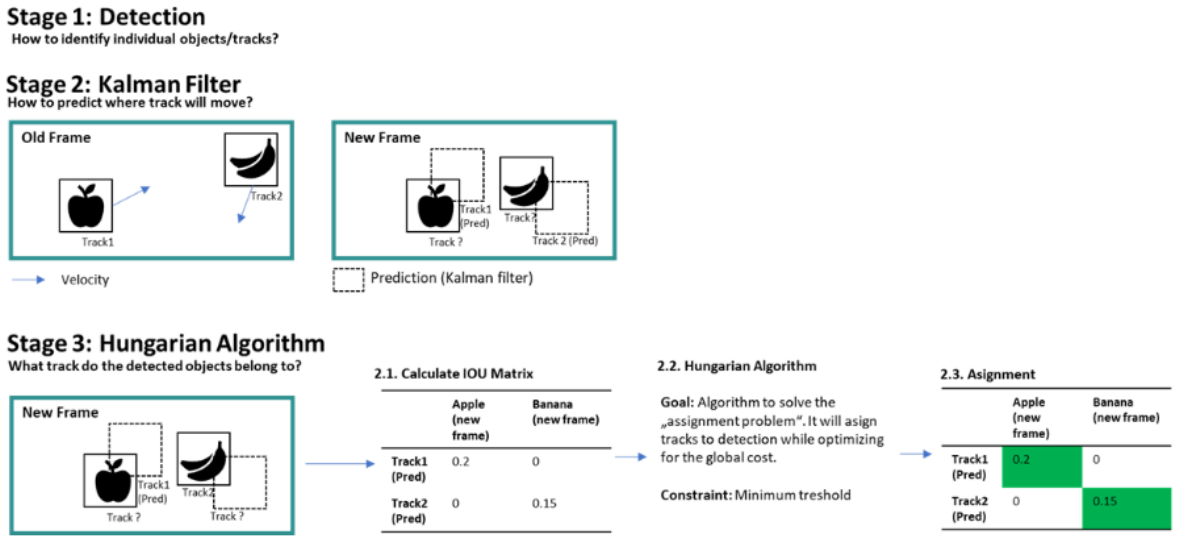
1. **Single Neural Network**:
    - YOLO approaches object detection as a single regression problem, where a single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation.
    - Unlike traditional object detection methods which use separate steps for object localization and classification, YOLO does both tasks simultaneously.
2. **Grid Cell Division**:
    - YOLO divides the input image into a grid of cells. Each cell is responsible for predicting bounding boxes and their corresponding class probabilities.
    - The dimensions of the grid can vary based on the network architecture and the desired output resolution.
3. **Bounding Box Prediction**:
    - Each grid cell predicts a fixed number of bounding boxes (typically 2 or 3).
    - For each bounding box, the network predicts:
        - Coordinates (x, y, width, height) relative to the boundaries of the grid cell.
        - Confidence score indicating the confidence that the box contains an object.
        - Class probabilities for each class.

4. **Anchor Boxes**:
    - YOLO employs anchor boxes to improve bounding box predictions.
    - Anchor boxes are predefined shapes with specific aspect ratios and scales.
    - Each bounding box prediction is offset from an anchor box, enabling the model to handle objects of different sizes and aspect ratios more effectively.
5. **Class Prediction**:
    - For each grid cell and bounding box, YOLO predicts class probabilities.
    - The class probabilities are conditioned on the presence of an object in the bounding box.
    - YOLO uses softmax activation to predict class probabilities independently for each class.
6. **Non-Maximum Suppression (NMS)**:
    - After predicting bounding boxes and class probabilities, YOLO applies non-maximum suppression to filter out redundant and overlapping bounding boxes.
    - This ensures that each object is detected only once with the highest confidence.
7. **Loss Function**:
    - YOLO optimizes a custom loss function that combines localization loss (measuring the accuracy of bounding box coordinates) and confidence loss (measuring the objectness score).
    - Additionally, YOLO incorporates a classification loss component to penalize incorrect class predictions.



SORT:

Simple Online and Realtime Tracking (SORT) was published in 2017 by Bewley et al. and is one of the first algorithms to handle object tracking in real-time.
The workflow is described as below:



1) Estimation Model (Kalman Filter)

In the next step, the movement of the objects is predicted based on a Kalman filter. A Kalman filter estimates the new position of an object by extrapolating the motion of the object. After observing the new position, the Kalman filter updates its belief. This video provides an introduction: here.

2) Data Association

Ultimately the detections must be associated with tracks. We use the Kalman filter predictions to measure the overlap (IOU) between predictions (based on last frame) and actual detections. If the detection and the Kalman prediction overlap, the detection is assigned to the track. In practice, this is more complicated than in the chart since multiple objects are involved. Finally, a matrix holds the IOUs and a Hungarian algorithm associates tracks and detections.

3) Creation and Deletion of Track Identities

When a new detection has an IOU below a specific threshold it is not assigned to an existing track but classified as a new object. Tracks are removed when no detection is assigned for a certain number of frames. That helps to avoid unbound growth.

DeepSort:

DeepSort was published in 2019 by Wojke, Bewley and Paulus and is a successor to SORT. The authors argue that SORT creates too many "identity switches" when the sight of objects is blocked. Therefore, they propose to enrich the motion model (Kalman filter) with a deep learning component that incorporates the visual features of an object. The workflow for deepsort is given as follows:

1) Calculating the cost matrix:

The cost matrix serves a similar purpose to the IOU matrix in SORT. It assigns scores on how likely a detection belongs to a track. As stated earlier, it works slightly different, because DeepSort also includes visual features of the object. The cost matrix is a combination of the motion model (Kalman filter) and visual similarity (Deep Neural Network Embeddings) (stage 3). The Kalman filter (stage 1) is almost like the one in SORT. The difference is that the IOU overlap calculation changes to a Mahalanobis distance. It considers the uncertainty of the prediction (by using the standard deviation) and measures distance even if there is no overlap.

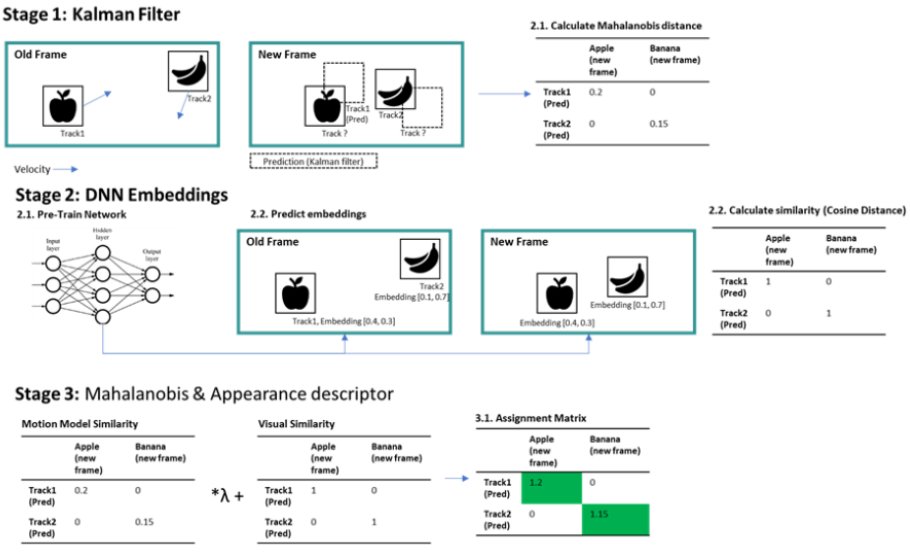
For the visual similarity (stage 3), DeepSort uses a pre-trained neural network (CNN). The authors use this network to extract embeddings, a lower-level representation of that image. They compare the embeddings of the detections and tracks using a cosine distance.

Additionally, the authors introduce a "gate matrix" that guarantees that only associations within certain gating regions are admissible.

2) Assignment Problem

For each track that scores below a certain age/frame threshold, the algorithm assigns new detections. The detection that has the least cost is associated with that track.

For all unmatched tracks, the algorithm runs an IOU matching like for SORT.



Differences between SORT and DeepSORT are as follows:

1. **Methodology**:
    - **SORT**: it is based on a combination of Kalman filtering and Hungarian algorithm for data association. It mainly relies on motion prediction and appearance matching for tracking objects.
    - **DeepSORT**: it extends SORT by incorporating deep learning features. It integrates a deep neural network for appearance embedding, which enables more robust and accurate data association even in crowded and occluded scenes.
2. **Data Association**:
    - **SORT**: it primarily uses motion prediction and Euclidean distance metrics for data association. It assigns detections to existing tracks based on their predicted locations and appearance similarities.
    - **DeepSORT**: it enhances data association by utilizing appearance features extracted from a deep neural network. It embeds the appearance of each detection and track into a high-dimensional space, enabling more accurate matching using techniques like cosine similarity.
3. **Appearance Embedding**:
    - **SORT**: it does not include a dedicated appearance embedding mechanism. It relies solely on motion prediction and simple features for data association.
    - **DeepSORT**: it integrates appearance embedding into the tracking process. It uses a deep neural network, often pretrained on large-scale datasets like ImageNet, to extract discriminative appearance features. These features help in distinguishing between objects with similar motion patterns.
4. **Robustness**:
    - **SORT**: it may struggle in scenarios with heavy occlusions or appearance changes due to its reliance on simple appearance features.
    - **DeepSORT**: it is more robust to occlusions and appearance variations due to its ability to learn discriminative appearance features. This makes it more suitable for complex tracking scenarios encountered in crowded environments or surveillance systems.
5. **Performance**:
    - **SORT**: it is computationally efficient and can achieve real-time performance on modest hardware.
    - **DeepSORT**: it is more computationally intensive compared to SORT due to the involvement of deep neural networks. However, with optimizations and hardware acceleration, it can still achieve real-time or near-real-time performance on modern hardware.

Results and Observations:

Drive Link to all the videos:

https://drive.google.com/drive/folders/1CGRblvBSVrFE4PIdoErdFfvLgztle55I?usp=sharing

FasterRCNN (DeepSort and Sort):

- **Actual Count of the no. of cars in the video = 8**
- **Calculated count of the no. of cars = [9 (DeepSORT) and 17 (SORT)]**

Faster RCNN + DeepSort

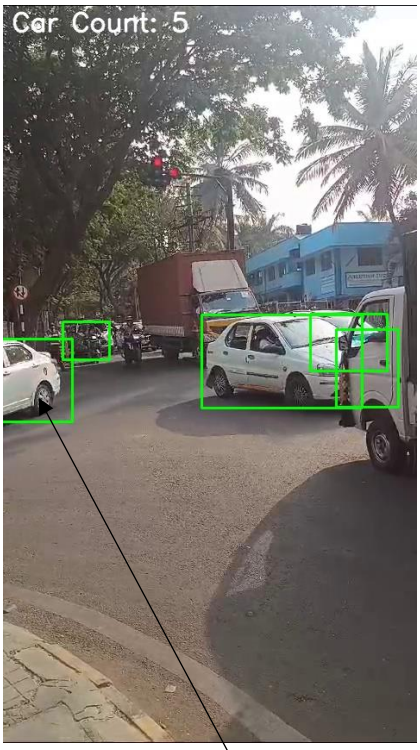
Faster RCNN + Sort

We can see that Sort suffers from occlusion(**the blue car** behind the white car is not detected whereas DeepSort is able to perfectly detect and track the blue car).

The count is nearly same for DeepSORT whereas it is a lot higher for SORT tracking algorithm, this is happening because:

- For SORT, the unique ids returned by the tracker object is not proper as the algorithm is assigning new id to an object which went out of the frame for sometime and then again re-enters(even if the object gets occluded for sometime behind another object).
- For DeepSORT more or less we are getting similar count in comparison to the ground truth mainly because of the pretrained CNN which helps in visual similarity in addition to the SORT technique already used.

- If we see the videos as well, the tracking of an object is much better in DeepSORT compared to SORT algorithm.
- The association of same objects or different objects are also better in DeepSORT compared to SORT mainly because of the usage of the pretrained CNN which does visual similarity by comparing the embeddings of the detections and tracks using a cosine distance.



Faster RCNN + DeepSort

Faster RCNN + Sort

- As we can see the **white car** in the left side of the image is getting tracked by the DeepSORT tracker whereas SORT is not able to track it once a part of the object moves out of the frame.
- Also as we can see the count of DeepSORT is steady whereas the count of SORT is increasing rapidly because of assigning new unique ids to objects which went out of the view in one frame and re-entered in another frame.
- The DeepSORT tracker is consistently tracking the cars behind the white car on the right whereas the SORT tracker is not able to do that.

The time taken to re-generate the video with counter and tracker for DeepSORT was ~33minutes.

The time taken to re-generate the video with counter and tracker for SORT was ~41minutes.

The time is totally dependent on the system configuration of the person running the code.

YOLOv8 (DeepSort and Sort):

- **Actual Count of the no. of cars in the video = 8**
- **Calculated count of the no. of cars = [8 (DeepSORT) and 13 (SORT)]**

YOLOv8 + DeepSort

YOLOv8 + Sort

First of all YOLOv8 is a very fast model architecture which is able to classify, detect and segment objects in an image/frame in just 1 pass hence the name YOLO(You Only Look Once). We can see the difference in time, Faster RCNN + DeepSORT took almost 33minutes compared to YOLOv8 which took almost 16minutes(mentioned below).

Similar to FasterRCNN, in YOLOv8 as well we can see that Sort suffers from occlusion(the **white car behind the white truck** on the right is not detected whereas DeepSort is able to perfectly detect and track that white car.

The count in this case is exactly equal to the Ground truth for DeepSORT(most of the time the count is equal, only with a very minimum probability will it be unequal) whereas it is a lot higher for SORT tracking algorithm, this is happening because:

- For SORT, the unique ids returned by the tracker object is not proper as the algorithm is assigning new id to an object which went out of the frame for sometime and then again re-enters(even if the object gets occluded for sometime behind another object).

- For DeepSORT we are getting almost equal count in comparison to the ground truth mainly because of the pretrained CNN phase to generate the embeddings which helps in visual similarity in addition to the SORT technique already used.
- If we see the videos as well, the tracking of an object is much better in DeepSORT compared to SORT algorithm.
- The association of same objects or different objects are also better in DeepSORT compared to SORT mainly because of the usage of the pretrained CNN which does visual similarity by comparing the embeddings of the detections and tracks using a cosine distance. Also YOLOv8 detects objects much better compared to Faster RCNN.

YOLOv8+ DeepSort

YOLOv8 + Sort

- As we can see the **white car** in the left side of the image is getting tracked by the DeepSORT tracker whereas SORT is not able to track it once a part of the object moves out of the frame.
- Also as we can see the count of DeepSORT is steady whereas the count of SORT is increasing rapidly because of assigning new unique ids to objects which went out of the view in one frame and re-entered in another frame.
- The DeepSORT tracker is consistently tracking the cars behind the white car on the right whereas the SORT tracker is not able to do that.

The time taken to re-generate the video with counter and tracker for DeepSORT was ~16minutes.

The time taken to re-generate the video with counter and tracker for SORT was ~24minutes.

NB: The time is totally dependent on the system configuration of the person running the code.(But YOLOv8 + DeepSORT will be the fastest).

YOLOv8 vs FasterRCNN:

- The object detection precision and accuracy is much better in YOLOv8 compared to FasterRCNN.
- The bounding boxes are much more tightly bound in YOLOv8 compared to Faster RCNN.
- Overlapping of bounding boxes in YOLOv8 is much lesser compared to Faster RCNN.
- The time taken to process the video and regenerate it with counter and object tracker in YOLOv8 (~16mins) compared to Faster RCNN(~33mins), thus making YOLOv8 being more real-time compared to Faster RCNN.
- Among all the workflows implemented, YOLOv8 + DeepSort is working with the best accuracy and efficiency.
- Refer the videos in the drive link to understand the differences.

****