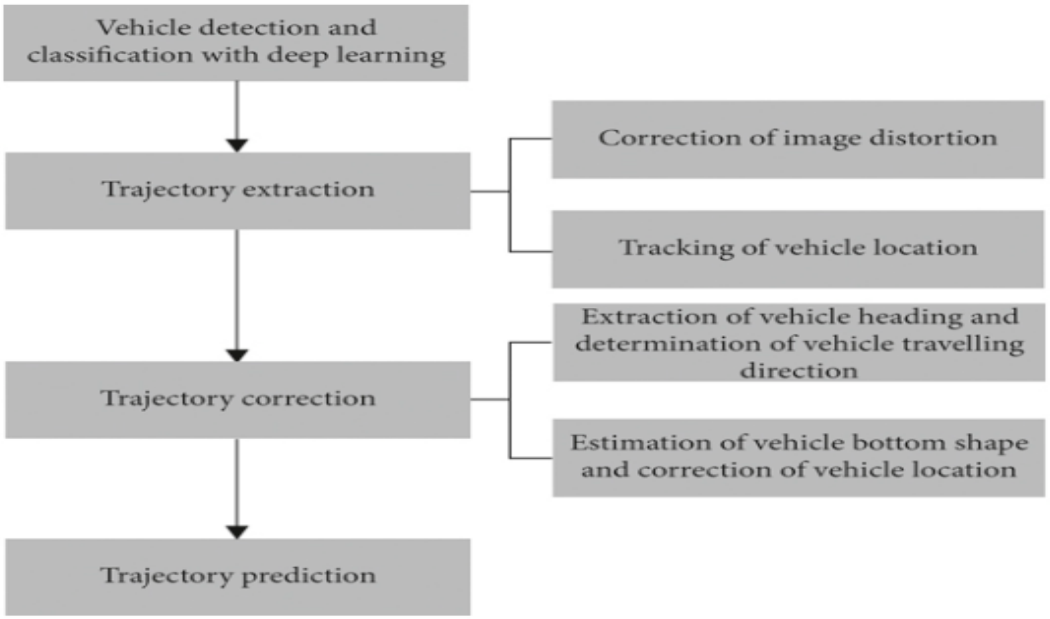
Vehicle Tracking and Number Plate Recognition System using (Drone/CCTV) Footage

- By Pramod Kshirsagar 21116073

● Intro to the Project

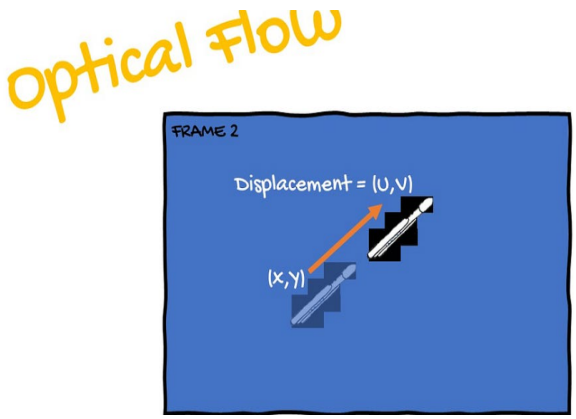
The main task of the project is to track the vehicles present in the video using **bounding boxes** and naming the class of the vehicle, and also a side task will be to extract the number plate text on the vehicle present in a particular frame. It is an emerging technology that aims to improve traffic management, enhance law enforcement and provide security for vehicles.
The system involves using videos from cameras of passing vehicles and using computer algorithms to extract and analyse the **position change of the vehicles**.
The main steps for executing this system are: 1) Detection
2) Bounding
3) Prediction
4) Tracking

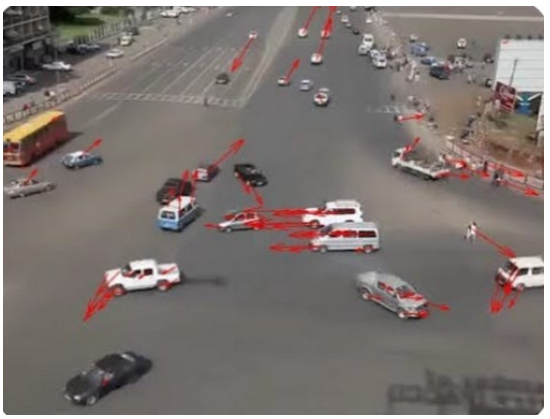


The code gives a useful and finalised video output.

- # Literature Review

This blog: [DeepSORT — Deep Learning applied to Object Tracking | by Ritesh Kanjee | Augmented Startups | Medium](#) by Augmented Startups provides an overview of DeepSORT, a deep learning-based object tracking algorithm. The author begins by explaining the importance of object tracking in computer vision applications, such as **self-driving cars**, surveillance systems, and robotics. It then introduces DeepSORT and nicely describes how it differs from traditional object tracking methods. The original SORT algorithm uses a combination of motion and appearance features to track objects. The author then cites the work of Wojke, who extended SORT by incorporating deep appearance features using a deep neural network.





It tells the advantages of DeepSORT, such as its ability to handle occlusions and track objects over long periods of time, and its superior performance compared to other state-of-the-art object tracking methods like Detectron,etc. As it uses detections from yolo algorithm, uses **Kalman Filtering** and includes **optical flow** for smooth Tracking.
Also, we get to know about the performance metrics, IOU score, and non max suppression used in it, and how Kalman Filter works.
They also mention the limitations of DeepSORT, such as its reliance on accurate detections and its sensitivity to changes in lighting and camera angles.

Another Blog: https://www.visobyte.com/2023/03/object-tracking-using-bytetrack.html tells about ByteTrack which is a deep learning-based object tracking algorithm that combines convolutional neural networks (**CNNs**) and recurrent neural networks (**RNNs**) to track objects in real-time. It uses motion prediction, Kalman filtering, and online model adaptation to handle occlusions, changes in lighting and viewpoint, and other challenges that can arise in real-world tracking scenarios.
It tells its advantages like robustness, efficiency, flexibility, and accuracy. ByteTrack is designed to be computationally efficient and can track objects in real-time video streams with high frame rates, making it well-suited for applications such as

surveillance, robotics, and autonomous driving, where real-time performance is critical. Its **flexibility** makes it a versatile tool for a variety of use cases. Its advanced feature detection and tracking techniques lead to highly accurate tracking results, particularly in applications where tracking accuracy is critical.
It also tells its disadvantages like its sensitivity to changes in lighting and occlusions. The algorithm may struggle to track objects under varying lighting conditions, particularly if the lighting changes abruptly or dramatically. It may also have difficulty tracking an object if it becomes partially or completely occluded by another object or if the object itself changes shape or appearance. Additionally, the algorithm may require significant training data and manual annotation to achieve optimal performance.

This paper: http://docs.neu.edu.tr/library/6812355026.pdf helps in understanding basics steps of the system, without much coding part, as it just explains the functionality of the system.

• Methodology

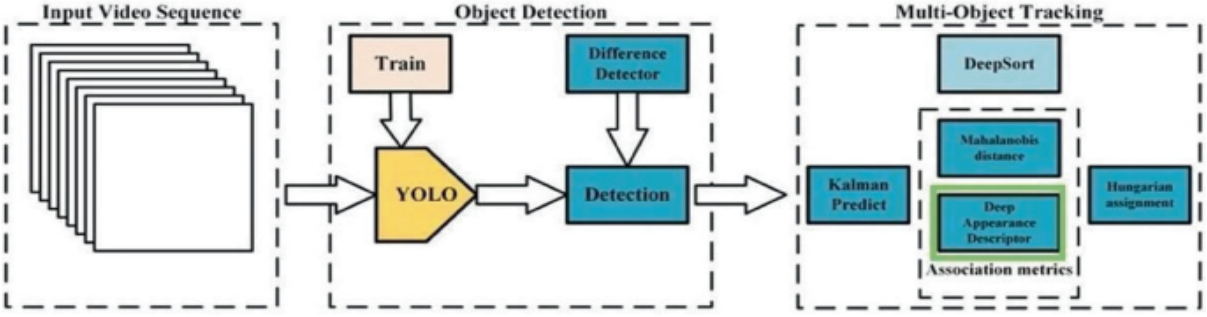
The process I followed in this Project is :
For Tracking: I followed two different approaches (Both are included in the code)

➢ 1st: DeepSORT Approach:

Importing necessary libraries like NumPy, OpenCV, tensorflow, etc.

Loading the YOLOv3 model: The model is trained on a large dataset of images and is capable of detecting many different object categories.

Defining the DeepSORT tracker: The DeepSORT algorithm is implemented using the deep_sort library. A tracker object is created using the configuration file provided by the library. The tracker is defined by creating an **instance** of the 'DeepSort' class, which takes in a configuration file that specifies the hyperparameters and settings for the tracker. The configuration file can be customised to tune the tracker's performance.



Processing each frame: The main loop of the code iterates over each frame of the video. The loop starts by reading the next frame from the video file.

Object detection using YOLOv3: For each frame, YOLOv3 is used to detect objects and their bounding boxes. The input frame is preprocessed by resizing it to the required input size of the model and normalising its pixel values. Then, the model is used to perform object detection on the preprocessed frame, which returns a list of detected objects and their corresponding **bounding boxes.**

Object tracking using DeepSORT: The detected objects are passed to the DeepSORT tracker which assigns unique IDs to them and tracks their movement across frames. The tracker uses a combination of appearance and motion features to associate objects across frames and estimate their position and velocity.

```
def draw_bbox(image, bboxes, CLASSES, show_label=True, show_confidence = True, Text_colors=(255,255,0), rectangle_colors=''
```

Drawing bounding boxes and IDs: Finally, the bounding boxes and IDs of the tracked objects are drawn onto the frame using OpenCV. The bounding boxes are drawn around the detected objects, and the IDs are displayed next to the bounding boxes.

Outputting the result: The output video is saved using OpenCV's VideoWriter function. This can be downloaded from content in colab.

In summary, the code performs object detection and tracking by first detecting objects in each frame using YOLOv3, then using the DeepSORT algorithm to track the objects across frames.

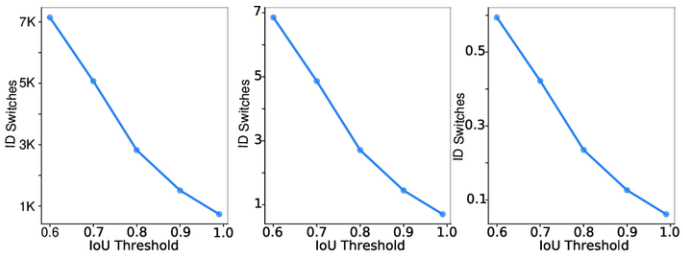
- ➢ 2nd: ByteTrack Approach:

Importing necessary libraries including tqdm, ByteTracker, VideoInfo, and others.

Creating **ByteTracker** instance: This instance contains all the necessary parameters for the ByteTracker algorithm.

A VideoInfo object is created by passing the source video file path. This object contains the video metadata like frame rate, width, height, etc.
A frame generator is created using the get_video_frames_generator() function which generates frames from the source video.

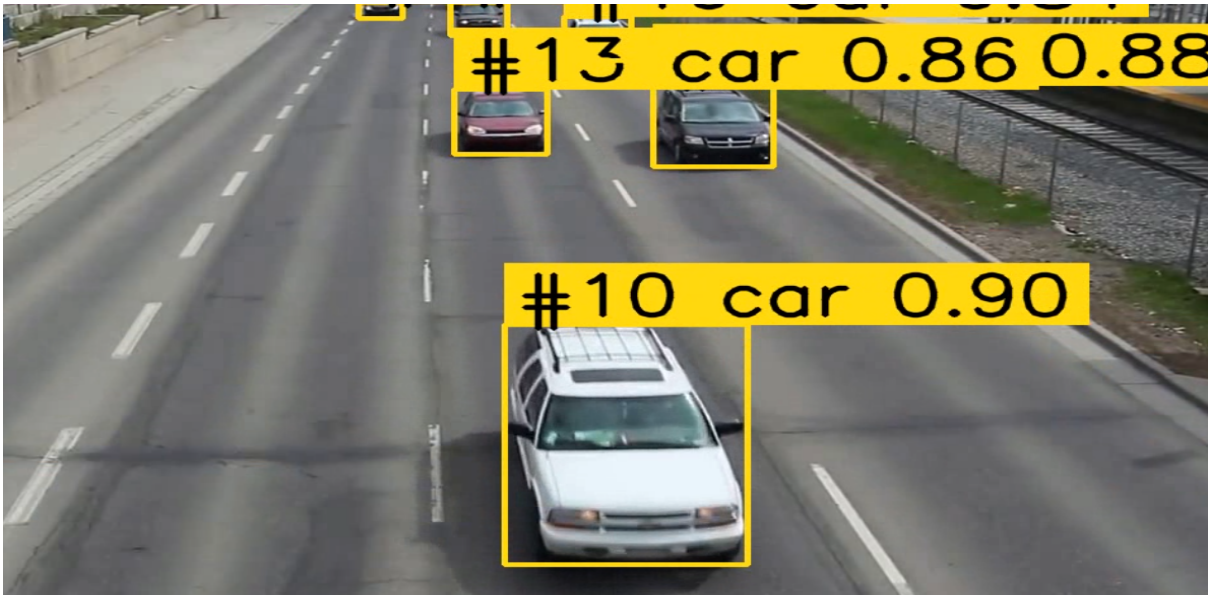
A LineCounter object is created by passing the start and end points of a line on the frame that represents the boundary of a region where vehicles are counted.



BoxAnnotator and LineCounterAnnotator objects are created to draw bounding boxes around the detected vehicles and display the vehicle count respectively.

A target video file is opened using the VideoSink context manager which writes the annotated frames to the target video file. The main loop of the code iterates over each frame of the video using tqdm to show progress.

Model prediction: For each frame, a pre-trained object detection model is used to predict the bounding boxes of vehicles present in the frame. The predictions are converted into a Detections object. Detections with unwanted classes are filtered out using a boolean mask.



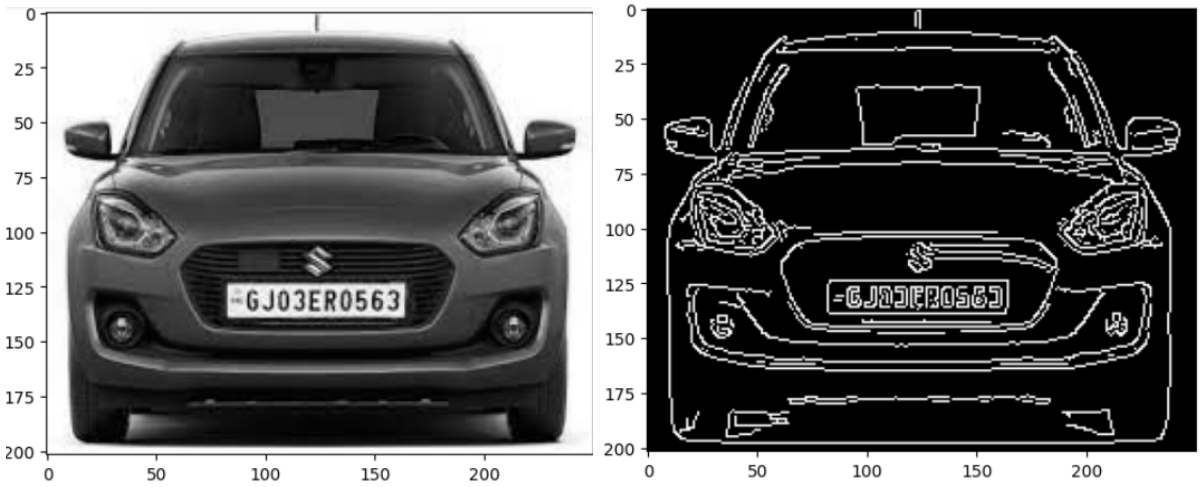
Tracking detections: The filtered detections are passed to the ByteTracker along with the frame dimensions, and the tracker returns a list of tracks and assigns a unique ID to each vehicle track, which is used to match detections with tracks.
Custom labels are created for each detection by combining the tracker ID, class name, and **confidence score.**
Bounding boxes are drawn and the annotated frame is written to the target video file using the VideoSink object.

Comparison: Both approaches give tracking outputs, but after testing, I found out that ByteTrack method gave better results, with accurate detections, and less crashing.

<u>For Number-plate Extraction:</u>

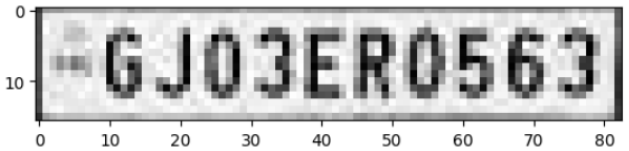
Importing necessary libraries including **EasyOCR** , MatPlotlib, and Imutils.

Converting the Image in the frame to grayscale using opencv to simplify the work.



Edge Detection using bilateral filter and canny functions. This is done to create contours.

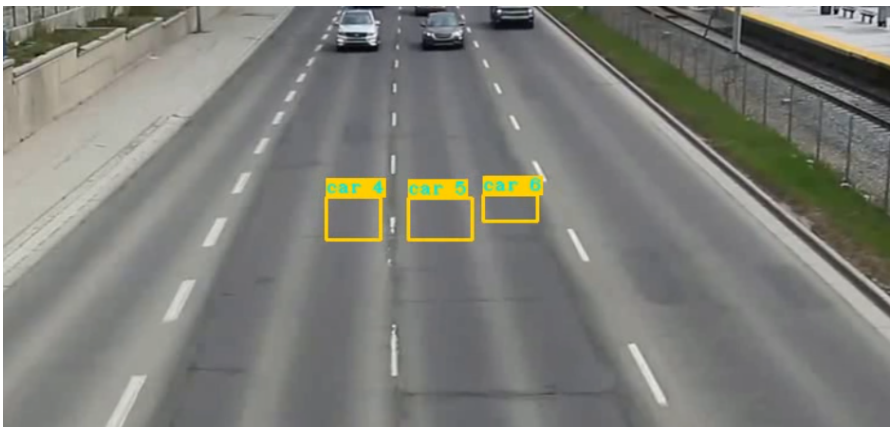
All the contours are extracted, and compiled, out of which contours having 4 edges are selected and the location of the number plate is detected.



A mask of the number plate is created and EasyOCR module is applied on the mask, which will give us the number plate text as output.

- <u>Experiments/Simulations</u> (also includes failures)

I tried changing the **class_threshold** (which decides which detections to select) to 0.7 rather than 0.8, which gave incorrect detections. For Ex: The first frame output was:



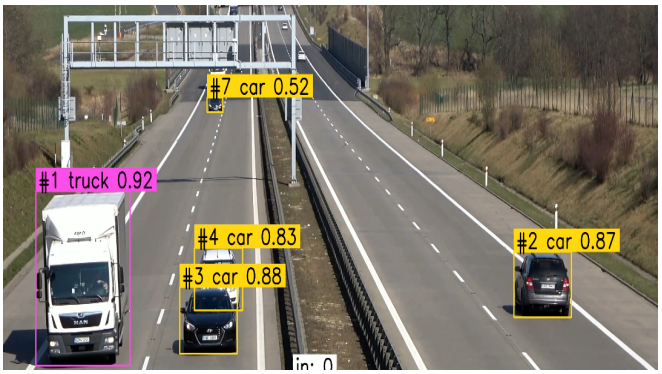
I tried reducing the classes in the yolo model which helped in better and faster output in DeepSORT also:

```
NUM_CLASS = {
    1: 'bicycle',
    2: 'car',
    3: 'motorbike',
    5: 'bus',
    7: 'truck'
}
```

I also tried an alternate yolo architecture which was compatible for tensorflow 2.3.0rc and colab had deprecated versions below 2.8 hence the code ran successfully but no detections were visible and the output was of 1 second only as the code crashed early.

- # Results

Finally, after trying many configurations, I came up with a Final code for Vehicle Tracking and NumberPlate Recognition which led to satisfactory and useful results:



The number plate extraction had high accuracy and tracking was done corrc\ect bounding boxes and classes.

- # References

https://learnopencv.com/understanding-multiple-object-tracking-usingdeepsort/#:~:text=DeepSORT%20can%20be%20defined%20as,offline%20just%20before%20implementing%20tracking.

https://www.v7labs.com/blog/yolo-object-detection

https://github.com/kcg2015/Vehicle-Detection-and-Tracking

https://towardsdatascience.com/guide-to-car-detection-using-yolo-48caac8e4ded#:~:text=YOLO%20is%20a%20stateof,contains%20information%20about%205%20boxes.

https://www.mdpi.com/1424-8220/22/10/3813