1. Introduction

In the discipline of computer vision, object tracking is the act of identifying and tracking one or more objects through a series of frames in a video. In this report, my objective is to demonstrate the use of advanced object detection models to accurately count the total number of vehicles in a short video clip and provide an output.

To address this problem, I will adopt a modern approach that combines the strengths of YOLO v8 (You Only Look Once version 8) as the core detection model for enhanced control and flexibility in object detection. Additionally, ByteTrack, a robust object-tracking method, aids in accurately following objects across frames, ensuring continuity in video analysis. also, incorporating Roboflow Supervision for video processing which enhances the overall workflow, offering efficient management of video data for improved object detection and tracking outcomes. By integrating these advanced technologies, I aim to achieve a reliable and efficient solution for accurate vehicle counting and contribute to advancement in computer vision applications.

There are various models available to address the issues related to object detection and tracking, including Faster R-CNN, SDD, YOLO, Mask R-CNN, and SORT. For this report, the **YOLO v8 (You Only Look Once version 8)** model in Python mode was adopted due to its accuracy, speed and ease of implementation in object detection, with the integration of ByteTrack, a powerfully built object-tracking method, to accurately follow objects across frames and ensure continuity in video analysis. Furthermore, I incorporated Roboflow Supervision for video processing to enhance the overall workflow, which offers efficient management of video data for enhanced object detection and tracking outcomes.

2. Implementation

This section outlines a detailed step-by-step process to achieving our desired result.

(1) First, ascertain the availability of GPU access by running the "**!nvidia-smi"** command line. This will display information such as the GPU utilisation, memory usage, temperature and other vital details about the NVIDIA GPUs on the system as shown below.

```
[1]  !nvidia-smi
```

```
Sun Nov 26 22:12:34 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   55C    P8     9W /  70W |      0MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

(2) This section installs Ultralytics YOLO v8 and checks the environment setup and dependencies, including GPU availability using the code below:

```
[2]  !pip install ultralytics

     from IPython import display
     display.clear_output()

     import ultralytics
     ultralytics.checks()
```

```
Ultralytics YOLOv8.0.218 🚀 Python-3.10.12 torch-2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB)
Setup complete ✅ (2 CPUs, 12.7 GB RAM, 26.9/78.2 GB disk)
```

(3) The following code prints the current working directory or location from which our task is being executed.

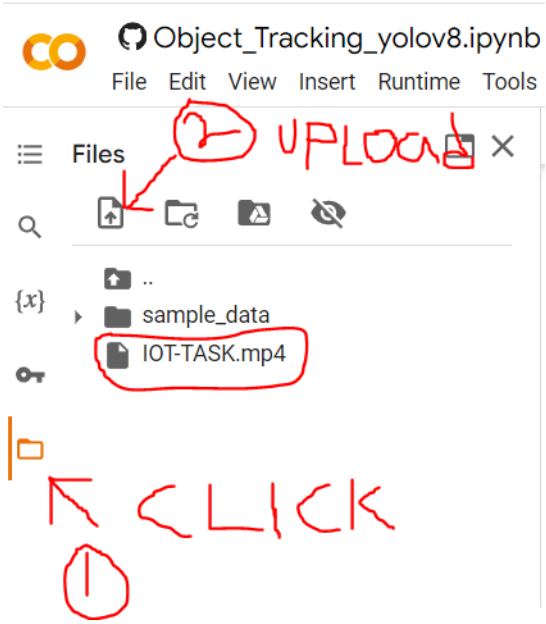
```
[3]  import os
     HOME = os.getcwd()
     print(HOME)
```

```
/content
```

(4) This segment sets the current working directory to the home directory and creates a file path for the uploaded video named “IOT-TASK.mp4” located in the home directory using the code below:

```
[4]  %cd {HOME}

     SOURCE_VIDEO_PATH = f"{HOME}/IOT-TASK.mp4"

     /content
```

NB: *Before proceeding with step 4, it is important to upload your file or video of interest directly into Google Colab from your local drive as shown in the image below.*



(5) For tracking objects in a video, the **YOLO v8 in Python mode** with **ByteTrack** integration was installed for better control, continuity and accuracy in object detection in video analysis. To execute this setup, refer to the code below:

```
[5]  %cd {HOME}
     !git clone https://github.com/ifzhang/ByteTrack.git
     %cd {HOME}/ByteTrack
     !sed -i 's/onnx==1.8.1/onnx==1.9.0/g' requirements.txt
     !pip3 install -q -r requirements.txt
     !python3 setup.py -q develop
     !pip install -q cython_bbox
     !pip install -q onemetric
     !pip install -q loguru lap thop
     from IPython import display
     display.clear_output()
     import sys
     sys.path.append(f"{HOME}/ByteTrack")


     import yolox
     print("yolox.__version__:", yolox.__version__)

     yolox.__version__: 0.1.0
```

```
[6]  from yolox.tracker.byte_tracker import BYTETracker, STrack
     from onemetric.cv.utils.iou import box_iou_batch
     from dataclasses import dataclass

     @dataclass(frozen=True)
     class BYTETrackerArgs:
         track_thresh: float = 0.25
         track_buffer: int = 30
         match_thresh: float = 0.8
         aspect_ratio_thresh: float = 3.0
         min_box_area: float = 1.0
         mot20: bool = False
```

(6) In addition, **Roboflow Supervision** was incorporated to improve object detection and tracking results through better video processing and effective video data management using the code below:

```
[7]  !pip install supervision==0.1.0


     from IPython import display
     display.clear_output()


     import supervision
     print("supervision.__version__:", supervision.__version__)

     supervision.__version__: 0.1.0
```

```
[8]  from supervision.draw.color import ColorPalette
     from supervision.geometry.dataclasses import Point
     from supervision.video.dataclasses import VideoInfo
     from supervision.video.source import get_video_frames_generator
     from supervision.video.sink import VideoSink
     from supervision.notebook.utils import show_frame_in_notebook
     from supervision.tools.detections import Detections, BoxAnnotator
     from supervision.tools.line_counter import LineCounter, LineCounterAnnotator
```

(7) This section manually matched the bounding boxes from the model with those created by the tracker using the code below:

```
[9]  from typing import List

     import numpy as np


     # converts Detections into format that can be consumed by match_detections_with_tracks function
     def detections2boxes(detections: Detections) -> np.ndarray:
         return np.hstack((
             detections.xyxy,
             detections.confidence[:, np.newaxis]
         ))


     # converts List[STrack] into format that can be consumed by match_detections_with_tracks function
     def tracks2boxes(tracks: List[STrack]) -> np.ndarray:
         return np.array([
             track.tlbr
             for track
             in tracks
         ], dtype=float)


     # matches our bounding boxes with predictions
     def match_detections_with_tracks(
         detections: Detections,
         tracks: List[STrack]
     ) -> Detections:
         if not np.any(detections.xyxy) or len(tracks) == 0:
             return np.empty((0,))

         tracks_boxes = tracks2boxes(tracks=tracks)
         iou = box_iou_batch(tracks_boxes, detections.xyxy)
         track2detection = np.argmax(iou, axis=1)

         tracker_ids = [None] * len(detections)

         for tracker_index, detection_index in enumerate(track2detection):
             if iou[tracker_index, detection_index] != 0:
                 tracker_ids[detection_index] = tracks[tracker_index].track_id

         return tracker_ids
```

(8) Loading the pre-trained model for YOLO v8 was done using the following code below:

```
[10] # settings
     MODEL = "yolov8x.pt"
```

```
[11] from ultralytics import YOLO

     model = YOLO(MODEL)
     model.fuse()
```

```
Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8x.pt to 'yolov8x.pt'...
100%|██████████| 131M/131M [00:00<00:00, 249MB/s]
YOLOv8x summary (fused): 268 layers, 68200608 parameters, 0 gradients, 257.8 GFLOPs
```

(9) This section entails a list of predefined objects and associated codes that are used in defining the **Class ID of interest** such as {0: ‘person’, 2: ‘car’, 7: ‘truck’, 14: ‘bird’, 19: ‘cow’} and many more from the list below:

Predefined objects and the associated codes are as follow: {0: 'person', 1: 'bicycle', 2: 'car', 3: 'motorcycle', 4: 'airplane', 5: 'bus', 6: 'train', 7: 'truck', 8: 'boat', 9: 'traffic light', 10: 'fire hydrant', 11: 'stop sign', 12: 'parking meter', 13: 'bench', 14: 'bird', 15: 'cat', 16: 'dog', 17: 'horse', 18: 'sheep', 19: 'cow', 20: 'elephant', 21: 'bear', 22: 'zebra', 23: 'giraffe', 24: 'backpack', 25: 'umbrella', 26: 'handbag', 27: 'tie', 28: 'suitcase', 29: 'frisbee', 30: 'skis', 31: 'snowboard', 32: 'sports ball', 33: 'kite', 34: 'baseball bat', 35: 'baseball glove', 36: 'skateboard', 37: 'surfboard', 38: 'tennis racket', 39: 'bottle', 40: 'wine glass', 41: 'cup', 42: 'fork', 43: 'knife', 44: 'spoon', 45: 'bowl', 46: 'banana', 47: 'apple', 48: 'sandwich', 49: 'orange', 50: 'broccoli', 51: 'carrot', 52: 'hot dog', 53: 'pizza', 54: 'donut', 55: 'cake', 56: 'chair', 57: 'couch', 58: 'potted plant', 59: 'bed', 60: 'dining table', 61: 'toilet', 62: 'tv', 63: 'laptop', 64: 'mouse', 65: 'remote', 66: 'keyboard', 67: 'cell phone', 68: 'microwave', 69: 'oven', 70: 'toaster', 71: 'sink', 72: 'refrigerator', 73: 'book', 74: 'clock', 75: 'vase', 76: 'scissors', 77: 'teddy bear', 78: 'hair drier', 79: 'toothbrush'}

(10) This section clarifies the object that needs to be detected and tracked in the **Class ID of interest,** by defining it as 'car and truck' with the associated code in **CLASS_ID = [2,7]** as shown below:

```
[12] CLASS_NAMES_DICT = model.model.names
     # class_id of interest - car, truck
     CLASS_ID = [2,7]
```

(11) This segment predicts and annotates the frames in the video by displaying the number of cars or trucks detected, their speed and post-processing per image. To accomplish this, run the following code:

```
[13] # create frame generator
     generator = get_video_frames_generator(SOURCE_VIDEO_PATH)
     # create instance of BoxAnnotator
     box_annotator = BoxAnnotator(color=ColorPalette(), thickness=4, text_thickness=4, text_scale=2)
     # acquire first video frame
     iterator = iter(generator)
     frame = next(iterator)
     # model prediction on single frame and conversion to supervision Detections
     results = model(frame)
     detections = Detections(
         xyxy=results[0].boxes.xyxy.cpu().numpy(),
         confidence=results[0].boxes.conf.cpu().numpy(),
         class_id=results[0].boxes.cls.cpu().numpy().astype(int)
     )
     # format custom labels
     labels = [
         f"{CLASS_NAMES_DICT[class_id]} {confidence:0.2f}"
         for _, confidence, class_id, tracker_id
         in detections
     ]
     # annotate and display frame
     frame = box_annotator.annotate(frame=frame, detections=detections, labels=labels)

     %matplotlib inline
     show_frame_in_notebook(frame, (16, 16))
```

(12) After executing the code in step 11. the expected outcome is shown below.

It is evident that the Yolo v8 model effectively distinguished between trucks and cars, while simultaneously calculating the number of vehicles passing and their speed.

(13) To determine the number of vehicles entering and exiting the video, a line segment's 'START' and 'END' points were defined to count the points where the vehicles come IN and OUT. Additionally, a file path was created and assigned to the `f"{HOME}"` directory, which represents the video result's source path. To accomplish this, run the code below:

```
# settings
LINE_START = Point(50, 1700)
LINE_END = Point(3500, 1700)

TARGET_VIDEO_PATH = f"{HOME}/IOT-TASK-result.mp4"
```

It is important to use a different name for the output video file from the name of the uploaded video file. For instance, if the uploaded video is named **`IOT-TASK.mp4`**, when generating the output video, you should change the name to **`IOT-TASK-result.mp4`**. This will allow the model to generate the output video efficiently.

(14) To gain information about the video file, run the code below:

```
[15] VideoInfo.from_video_path(SOURCE_VIDEO_PATH)

VideoInfo(width=3840, height=2160, fps=25, total_frames=538)
```

This displays information such as the width, height, frames per second, and total number of frames in the video.

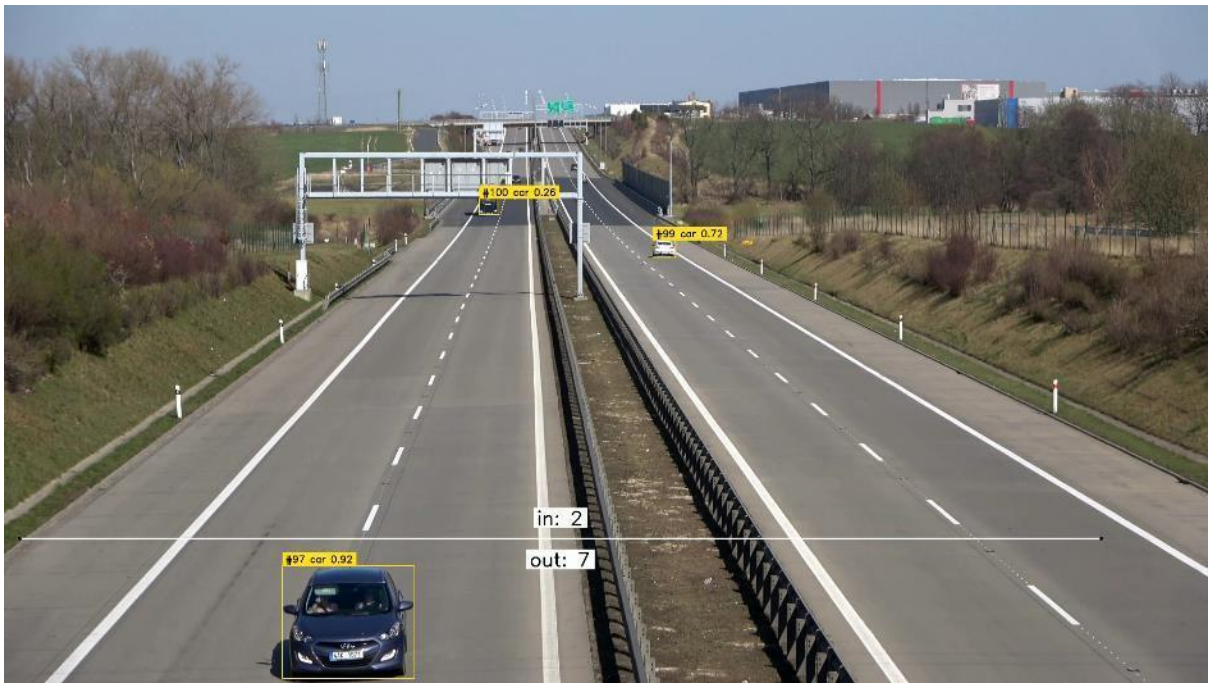
(15) This section is where the all dots connect for performing the object detection, tracking, and annotation on the video frames using the YOLO v8 model and provide the annotated frames in a new video file at the colab upload tab. Additionally, the `tqdm` module was imported and used to display the progress bars during iteration. To accomplish this, execute the code provided below:

```
[16] from tqdm.notebook import tqdm


# create BYTETracker instance
byte_tracker = BYTETracker(BYTETrackerArgs())
# create VideoInfo instance
video_info = VideoInfo.from_video_path(SOURCE_VIDEO_PATH)
# create frame generator
generator = get_video_frames_generator(SOURCE_VIDEO_PATH)
# create LineCounter instance
line_counter = LineCounter(start=LINE_START, end=LINE_END)
# create instance of BoxAnnotator and LineCounterAnnotator
box_annotator = BoxAnnotator(color=ColorPalette(), thickness=2, text_thickness=2, text_scale=1)
line_annotator = LineCounterAnnotator(thickness=4, text_thickness=4, text_scale=2)

# open target video file
with VideoSink(TARGET_VIDEO_PATH, video_info) as sink:
    # loop over video frames
    for frame in tqdm(generator, total=video_info.total_frames):
        # model prediction on single frame and conversion to supervision Detections
        results = model(frame)
        detections = Detections(
            xyxy=results[0].boxes.xyxy.cpu().numpy(),
            confidence=results[0].boxes.conf.cpu().numpy(),
            class_id=results[0].boxes.cls.cpu().numpy().astype(int)
        )
```

```
[16]
# filtering out detections with unwanted classes
mask = np.array([class_id in CLASS_ID for class_id in detections.class_id], dtype=bool)
detections.filter(mask=mask, inplace=True)
# tracking detections
tracks = byte_tracker.update(
    output_results=detections2boxes(detections=detections),
    img_info=frame.shape,
    img_size=frame.shape
)
tracker_id = match_detections_with_tracks(detections=detections, tracks=tracks)
detections.tracker_id = np.array(tracker_id)
# filtering out detections without trackers
mask = np.array([tracker_id is not None for tracker_id in detections.tracker_id], dtype=bool)
detections.filter(mask=mask, inplace=True)
# format custom labels
labels = [
    f"#{tracker_id} {CLASS_NAMES_DICT[class_id]} {confidence:0.2f}"
    for _, confidence, class_id, tracker_id
    in detections
]
# updating line counter
line_counter.update(detections=detections)
# annotate and display frame
frame = box_annotator.annotate(frame=frame, detections=detections, labels=labels)
line_annotator.annotate(frame=frame, line_counter=line_counter)
sink.write_frame(frame)
```

(16) After completing step 15, the resulting annotated frames will be displayed as a new video in the file upload section and can be downloaded for further analysis as shown below:

(17) Find below a preview image of the outcome:



3. Conclusion

After careful analysis, it can be concluded that the integration of ByteTrack and Roboflow supervision with the YOLO v8 model has successfully met all the task requirements. The model was able to accurately detect and track each passing vehicle in the video clip while assigning the correct class ID to them. Additionally, the model was capable of counting and displaying the total number of vehicles moving IN and OUT of the video frames. In a critical evaluation, the solidly built object detection capacity of YOLO v8, the enhanced tracking precision provided by ByteTrack, and the streamlined workflow enabled by Roboflow are the strengths of this solution. However, certain weaknesses need to be addressed, such as difficulties in fine-tuning parameters and optimising for specific scenarios.

Several ideas can be implemented to improve this solution further. Firstly, it would be beneficial to optimise the hyperparameters of YOLO v8 and fine-tune the model on a more diverse dataset. This will improve the accuracy and generalisability of the model across different scenarios, lighting conditions, and vehicle types.

Furthermore, improving the tracking abilities of ByteTrack by using more advanced algorithms to handle occlusions and object interactions will help overcome potential challenges in complex traffic situations. Additionally, continuously monitoring and

updating the model based on feedback from real-world performance will ensure its adaptability to evolving scenarios and emerging challenges.