**Innovative Project Progress Report**

**Project Title:** The Analysis of Deep Learning Based Vehicle Classification, Tracking and Speed Estimation Systems

**Date:** 18.04.2024

**Author(s):** Chaitanya Anand, Arkadeepto Majumder

**Project Mentor:** Prof. Ardhendu Kundu

**1. Introduction**

Traffic surges with booming tech and cities, leading to congestion and accidents. Deep learning algorithms like YOLO and Faster R-CNN offer a solution for vehicle classification, tracking, and speed detection in complex environments. Our project uses these techniques to attempt to detect vehicles causing congestion.

**2. Methodology**

We have chosen YOLO object detection model for our project. The model was trained on COCO Dataset of images of annotated vehicles. YOLO offers the fastest real-time detection compared to other models. We don’t want a larger degree of precision as we need only approximate bounding boxes to calculate the speed and identification of vehicles, so YOLO model is idea for our project.

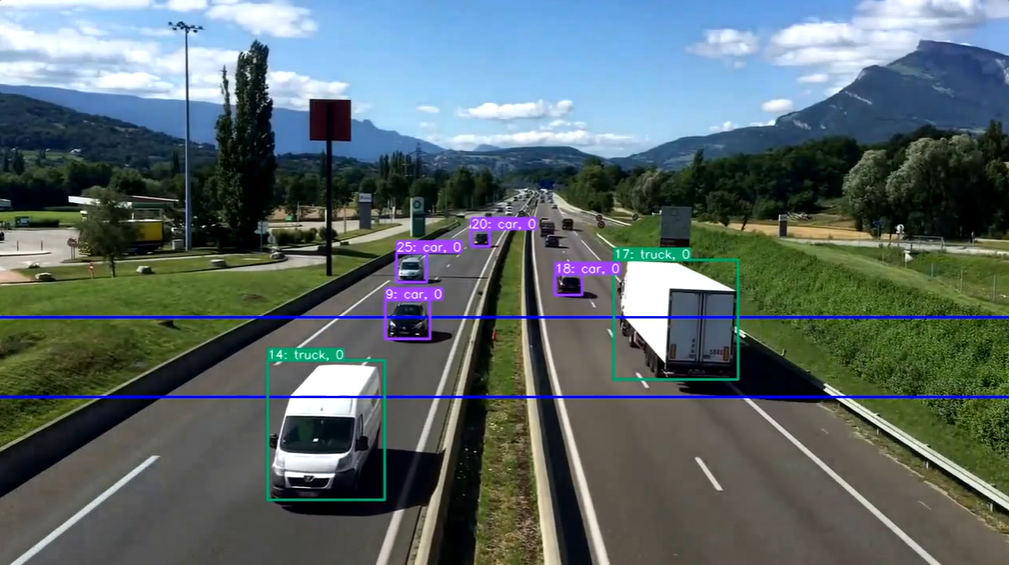
To keep a track of detected vehicles we use DeepSORT tracking algorithm which allows us to tag vehicles with unique ids and keep a record of their speeds for further processing.

**3. Milestones:**

* 5th Semester: We researched different object detection methods and their accuracy metrics and decided upon YOLO for object detection. We began writing a review paper on this subject and added our method of attempt at detecting vehicle speeds. We implemented our project to detect and classify vehicles.
* 6th Semester: Our previous implementation of the project involved detecting the same object multiple times in each frame leading to multiple bounding boxes over the same vehicle. We improved our project by including an object tracking method( DeepSORT) along with our previous project. This streamlined the detection format and allows us to tag individual vehicles with their recorded speeds, as long as they are in the camera feed. We researched this object tracking method and included our findings in our paper.
* Future objectives: We plan to detect whether a slow-moving vehicle is causing congestion or not by determining:
  1. Velocity of Buses: Determine the speed of buses by analyzing their movement across frames. A significant drop in speed compared to vehicles in earlier time frames may indicated congestion.
  2. Surrounding Vehicle Density: Analyzing the density of traffic near the bus. High vehicle density with slow movement can signal a traffic jam.

**4. Results**

A continuous box over the vehicles is established and we can track each individual vehicles and their speeds in the back-end thanks to DeepSORT.



**5. Conclusion**

Over two semesters, we developed a deep learning project for vehicle detection and speed estimation. We started by researching object detection methods and implementing YOLO for vehicle classification. In the second semester, we tackled duplicate detections by integrating DeepSORT, an object tracking algorithm. This allows us to track individual vehicles and calculate their speeds within the camera's view, significantly improving our project's accuracy.

**Appendix**

import cv2

import time

# Global dictionaries to store the vehicle positions and crossing times across frames

previous\_positions = {}

line\_crossing\_times = {}

def draw\_boxes\_vehicles\_counting\_el(img, bbox, identities=None, categories=None, names=None, offset=(0,0), fps=30):

    # Line coordinates and thickness

    line1\_y = 400

    line2\_y = 500

    line\_thickness = 2

    # Draw the lines on the image

    cv2.line(img, (0, line1\_y), (img.shape[1], line1\_y), (255, 0, 0), line\_thickness)

    cv2.line(img, (0, line2\_y), (img.shape[1], line2\_y), (255, 0, 0), line\_thickness)

    for i, box in enumerate(bbox):

        x1, y1, x2, y2 = [int(i) + offset[j % 2] for j, i in enumerate(box)]

        cat = int(categories[i]) if categories is not None else 0

        id = int(identities[i]) if identities is not None else 0

        cx, cy = int((x1 + x2) / 2), int((y1 + y2) / 2)

        speed\_text  = 0

        # Record the crossing time when crossing each line

        frame\_time = time.time()

        if cy > line1\_y and 'line1' not in line\_crossing\_times.get(id, {}):

            line\_crossing\_times[id] = {'line1': frame\_time}

        if cy > line2\_y and 'line2' not in line\_crossing\_times.get(id, {}):

            line\_crossing\_times[id]['line2'] = frame\_time

            # Calculate speed if both line crossing times are recorded

            if 'line1' in line\_crossing\_times[id]:

                time\_difference = line\_crossing\_times[id]['line2'] - line\_crossing\_times[id]['line1']

                ## ENTER DISTANCE BETWEEN ACTUAL LINE( VIRTUAL DISTANCE IS TAKEN AS PLACEHOLDER)

                distance = line2\_y - line1\_y  # Assuming both lines are horizontal

                speed = 0

                if(time\_difference!=0):

                    speed = (distance / time\_difference) \* (fps / 10)  # Convert speed into a more relatable unit

                speed\_text = f"{speed:.2f} units/s"  # Units depends on the actual scale of the video

        # Other drawing operations

        cv2.rectangle(img, (x1, y1), (x2, y2), color=compute\_color\_for\_labels(cat), thickness=2, lineType=cv2.LINE\_AA)

        label = f"{id}: {classNames[cat]}, {speed\_text}"

        (w, h), \_ = cv2.getTextSize(label, cv2.FONT\_HERSHEY\_SIMPLEX, fontScale=1/2, thickness=1)

        cv2.rectangle(img, (x1, y1), (x1 + w, y1 - h - 3), color=compute\_color\_for\_labels(cat), thickness=-1, lineType=cv2.LINE\_AA)

        cv2.putText(img, label, (x1, y1 - 2), cv2.FONT\_HERSHEY\_SIMPLEX, 1/2, [255, 255, 255], thickness=1, lineType=cv2.LINE\_AA)

    return img

def compute\_color\_for\_labels(cat):

    # Simple function to assign a color based on the category

    return (int(cat) \* 125 % 256, int(cat) \* 25 % 256, int(cat) \* 75 % 256)

classNames = {0: "car", 1: "truck", 2: "bus"}  # Example categories

classNames = cococlassNames()

output = cv2.VideoWriter('Output1.avi', cv2.VideoWriter\_fourcc('M', 'J', 'P', 'G'), 10, (frame\_width, frame\_height))

totalcountup=[]

totalcountdown=[]

fps = cap.get(cv2.CAP\_PROP\_FPS)

while True:

    xywh\_bboxs = []

    confs = []

    oids = []

    outputs = []

    ret, frame = cap.read()

    if ret:

      result = list(model.predict(frame, conf=0.5))[0]

      bbox\_xyxys = result.prediction.bboxes\_xyxy.tolist()

      confidences = result.prediction.confidence

      labels = result.prediction.labels.tolist()

      for (bbox\_xyxy, confidence, cls) in zip(bbox\_xyxys, confidences, labels):

          bbox = np.array(bbox\_xyxy)

          x1, y1, x2, y2 = bbox[0], bbox[1], bbox[2], bbox[3]

          x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)

          conf = math.ceil((confidence\*100))/100

          cx, cy = int((x1+x2)/2), int((y1+y2)/2)

          bbox\_width = abs(x1-x2)

          bbox\_height = abs(y1-y2)

          xcycwh = [cx, cy, bbox\_width, bbox\_height]

          xywh\_bboxs.append(xcycwh)

          confs.append(conf)

          oids.append(int(cls))

      xywhs = torch.tensor(xywh\_bboxs)

      confss= torch.tensor(confs)

      outputs = deepsort.update(xywhs, confss, oids, frame)

      if len(outputs)>0:

          bbox\_xyxy = outputs[:,:4]

          identities = outputs[:, -2]

          object\_id = outputs[:, -1]

          draw\_boxes\_vehicles\_counting\_el(frame, bbox\_xyxy, identities, object\_id, fps=fps)

      output.write(frame)

    else:

        break

output.release()

cap.release()