**Practical Assignment**

**Objective: - Vehicle Counting and Classification**

You can use the computer vision techniques to classify vehicles on the road, HMV(heavy motor vehicle) or LMV( light motor vehicle) and also count the number of vehicles that travel through a road.

The data can be stored to analyze the different vehicles that travel from a road.

**Dataset Link: -**

Use anyone of your choice.

**Task: -** Create a Desktop App for videos where we can count and classify the vehicles.

**Deployment: -** Any Free Platform(Try to look out for free options.)

**Assignment Submission: -** Only submit the hosted app link. OR GitHub Link

**Step #1 – Import Relevant Packages and Initialize the Network**

import cv2  
  
import csv  
  
import collections  
  
import numpy as np  
  
from tracker import \*

***Initialize Tracker***

tracker = EuclideanDistTracker()

***Detection Confidence Threshold***

confThreshold =0.1  
  
nmsThreshold= 0.2

First, you import all of the project’s required packages.

Then, from the *tracker*program, you initialize the *EuclideanDistTracker()* object and set the object to “*tracker*.”

*confThreshold* and *nmsThreshold*are the detection and suppression minimal confidence score thresholds, respectively.

# Middle cross line position  
  
middle\_line\_position = 225  
  
up\_line\_position = middle\_line\_position - 15  
  
down\_line\_position = middle\_line\_position + 15

You need to modify the middle\_line\_position according to your need.

***Store Coco Names in a list***

classesFile = "coco.names"  
  
classNames = open(classesFile).read().strip().split('\n')  
  
print(classNames)  
  
print(len(classNames))  
  
The Output

***Model Files***

modelConfiguration = 'yolov3-320.cfg'  
  
modelWeights = 'yolov3-320.weights'  
  
***Configure the Network Model***  
  
net = cv2.dnn.readNetFromDarknet(modelConfiguration, modelWeights)

***Configure the Network Backend***

net.setPreferableBackend(cv2.dnn.DNN\_BACKEND\_CUDA)  
  
net.setPreferableTarget(cv2.dnn.DNN\_TARGET\_CUDA)  
  
***Define Random Colors for Each Class***  
  
np.random.seed(42)  
  
colors = np.random.randint(0, 255, size=(len(classNames), 3), dtype='uint8')

**Step #2 – Read the Frames from the Video Files**

* *Cap.read()* reads each frame from the capture object after reading the video file using the video capture object.
* You cut your frame in half by using *cv2.reshape()*.
* The crossing lines are drawn in the frame using the *cv2.line()*function.
* Finally, you display the generated image using the *cv2.imshow()*function.

***Initialize the VideoCapture Object***

 cap = cv2.VideoCapture('video.mp4')  
  
  def realTime():  
  
while True:  
  
success, img = cap.read()  
  
img = cv2.resize(img,(0,0),None,0.5,0.5)  
  
ih, iw, channels = img.shape  
  
# Draw the crossing lines  
  
cv2.line(img, (0, middle\_line\_position), (iw, middle\_line\_position),  
  
(255, 0, 255), 1)  
  
cv2.line(img, (0, up\_line\_position), (iw, up\_line\_position), (0, 0, 255), 1)  
  
cv2.line(img, (0, down\_line\_position), (iw, down\_line\_position), (0, 0, 255), 1)  
  
# Show the frames  
  
cv2.imshow('Output', img)  
  
if name == 'main':  
  
realTime()

**Step #3 – Pre-Processing Frames and Running Detection**

* This YOLO version accepts 320×320 image objects as input. The network’s input is a blob object. The function *dnn.blobFromImage()* accepts an image as input and returns a blob object that has been shrunk and normalized.
* The image is fed onto the network using *net.forward(),* and it produces a result.
* Finally, you invoke the custom *postProcess()*function to post-process the output.

input\_size = 320  
  
blob = cv2.dnn.blobFromImage(img, 1 / 255, (input\_size, input\_size), [0, 0, 0], 1, crop=False)  
  
# Set the input of the network  
  
net.setInput(blob)  
  
layersNames = net.getLayerNames()  
  
outputNames = [(layersNames[i[0] - 1]) for i in net.getUnconnectedOutLayers()]  
  
# Feed data to the network  
  
outputs = net.forward(outputNames)  
  
# Find the objects from the network output  
  
postProcess(outputs,img)

**Step #4 – Post-Processing Output**

The forward network output has three outputs. Each output object is an 85-length vector.

* Four times the bounding box (centerX, centerY, width, height)
* One confidence box
* 80x class assurance

Let’s start by defining the post-processing function.

detected\_classNames = []  
  
def postProcess(outputs,img):  
  
global detected\_classNames  
  
height, width = img.shape[:2]  
  
boxes = []  
  
classIds = []  
  
confidence\_scores = []  
  
detection = []  
  
for output in outputs:  
  
for det in output:  
  
scores = det[5:]  
  
classId = np.argmax(scores)  
  
confidence = scores[classId]  
  
if classId in required\_class\_index:  
  
if confidence > confThreshold:  
  
# print(classId)  
  
w,h = int(det[2]*width) , int(det[3]*height)  
  
x,y = int((det[0]*width)-w/2) , int((det[1]*height)-h/2)  
  
boxes.append([x,y,w,h])  
  
classIds.append(classId)  
  
confidence\_scores.append(float(confidence))

**Step #5 – Counting All the Tracked Vehicles on the Road**

* After receiving all detections, you use the *tracker*object to keep track of those things. The *tracker. update()* function maintains track of all identified objects and updates their positions.
* The custom function *count\_vehicle* counts the number of vehicles that have passed through the road.

***The count\_vehicle Feature***

#Update the tracker for each object  
  
boxes\_ids = tracker.update(detection)  
  
for box\_id in boxes\_ids:  
  
    count\_vehicle(box\_id)

***Empty Lists***

* Create two empty lists (temporary ones) for storing the Vehicle IDs entering the entry crossing line.

List for store vehicle count information

temp\_up\_list = []  
  
temp\_down\_list = []

* To count four Vehicle classes in the up and down routes are *Up\_list* and *down\_list.*

up\_list = [0, 0, 0, 0]  
  
down\_list = [0, 0, 0, 0]

***Function for Counting Vehicle***

def count\_vehicle(box\_id):  
  
x, y, w, h, id, index = box\_id  
  
# Find the center of the rectangle for detection  
  
center = find\_center(x, y, w, h)  
  
ix, iy = center

***The find\_center Function***

* The center point of a rectangle box is returned through the *find\_center* function.

#Find the current position of the vehicle  
  
if (iy > up\_line\_position) and (iy < middle\_line\_position):  
  
    if id not in temp\_up\_list:  
  
        temp\_up\_list.append(id)  
  
elif iy < down\_line\_position and iy > middle\_line\_position:  
  
    if id not in temp\_down\_list:  
  
        temp\_down\_list.append(id)  
  
elif iy < up\_line\_position:  
  
    if id in temp\_down\_list:  
  
        temp\_down\_list.remove(id)  
  
        up\_list[index] = up\_list[index]+1  
  
elif iy > down\_line\_position:  
  
    if id in temp\_up\_list:  
  
        temp\_up\_list.remove(id)  
  
        down\_list[index] = down\_list[index] + 1

**Step #6 – Saving the Final Data**

* You open a new file, data.csv, with write permission only using the *open*function.
* Then, you write three rows: the first with class names and directions, the second with up and down route counts, and the third with both.
* The *writerow()* function saves a row of data to a file.

#Write the Vehicle counting information in a file and save it  
  
with open("data.csv", 'w') as f1:  
  
cwriter = csv.writer(f1)  
  
cwriter.writerow(['Direction', 'Vehicle', 'motorbike', 'bus', 'truck'])  
  
up\_list.insert(0, "Up")  
  
down\_list.insert(0, "Down")  
  
cwriter.writerow(up\_list)  
  
cwriter.writerow(down\_list)  
  
f1.close()