#import lib files

from tracking.centroidtracker import CentroidTracker

from tracking.trackableobject import TrackableObject

import tensornets as nets

import cv2

import numpy as np

import time

import dlib

import tensorflow.compat.v1 as tf

import os

import threading

def countVehicles(param):

# param -> path of the video

# list -> number of vehicles will be written in the list

# index ->Index at which data has to be written

tf.disable\_v2\_behavior()

# Image size must be '416x416' as YoloV3 network expects that specific image size as input

img\_size = 416

    inputs = tf.placeholder(tf.float32, [None, img\_size, img\_size, 3])

    model = nets.YOLOv3COCO(inputs, nets.Darknet19)

ct = CentroidTracker(maxDisappeared=5, maxDistance=50) # Look into 'CentroidTracker' for further info about parameters

    trackers = [] # List of all dlib trackers

    trackableObjects = {} # Dictionary of trackable objects containing object's ID and its' corresponding centroid/s

    skip\_frames = 10 # Numbers of frames to skip from detecting

    confidence\_level = 0.40 # The confidence level of a detection

    total = 0 # Total number of detected objects from classes of interest

    use\_original\_video\_size\_as\_output\_size = True # Shows original video as output and not the 416x416 image that is used as yolov3 input (NOTE: Detection still happens with 416x416 img size but the output is displayed in original video size if this parameter is True)

    video\_path = os.getcwd() + param # "/videos/4.mp4"

    video\_name = os.path.basename(video\_path)

# print("Loading video {video\_path}...".format(video\_path=video\_path))

if not os.path.exists(video\_path):

        print("File does not exist. Exited.")

        exit()

# YoloV3 detects 80 classes represented below

all\_classes = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus", "train", "truck", \

                  "boat", "traffic light", "fire hydrant", "stop sign", "parking meter", "bench", \

                  "bird", "cat", "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra", "giraffe", \

                  "backpack", "umbrella", "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard", \

                  "sports ball", "kite", "baseball bat", "baseball glove", "skateboard", "surfboard", \

                  "tennis racket", "bottle", "wine glass", "cup", "fork", "knife", "spoon", "bowl", "banana", \

                  "apple", "sandwich", "orange", "broccoli", "carrot", "hot dog", "pizza", "donut", "cake", \

                  "chair", "sofa", "pottedplant", "bed", "diningtable", "toilet", "tvmonitor", "laptop", "mouse", \

                  "remote", "keyboard", "cell phone", "microwave", "oven", "toaster", "sink", "refrigerator", \

                  "book", "clock", "vase", "scissors", "teddy bear", "hair drier", "toothbrush"]

# Classes of interest (with their corresponding indexes for easier looping)

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

    with tf.Session() as sess:

        sess.run(model.pretrained())

        cap = cv2.VideoCapture(video\_path)

# Get video size (just for log purposes)

width =  int(cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

        height = int(cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

# Scale used for output window size and net size

width\_scale = 1

        height\_scale = 1

        if use\_original\_video\_size\_as\_output\_size:

            width\_scale = width / img\_size

            height\_scale = height / img\_size

        def drawRectangleCV2(img, pt1, pt2, color, thickness, width\_scale=width\_scale, height\_scale=height\_scale):

            point1 = (int(pt1[0] \* width\_scale), int(pt1[1] \* height\_scale))

            point2 = (int(pt2[0] \* width\_scale), int(pt2[1] \* height\_scale))

            return cv2.rectangle(img, point1, point2, color, thickness)

        def drawTextCV2(img, text, pt, font, font\_scale, color, lineType, width\_scale=width\_scale, height\_scale=height\_scale):

            pt = (int(pt[0] \* width\_scale), int(pt[1] \* height\_scale))

            cv2.putText(img, text, pt, font, font\_scale, color, lineType)

        def drawCircleCV2(img, center, radius, color, thickness, width\_scale=width\_scale, height\_scale=height\_scale):

            center = (int(center[0] \* width\_scale), int(center[1] \* height\_scale))

            cv2.circle(img, center, radius, color, thickness)

# Python 3.5.6 does not support f-strings (next line will generate syntax error)

#print(f"Loaded {video\_path}. Width: {width}, Height: {height}")

# print("Loaded {video\_path}. Width: {width}, Height: {height}".format(video\_path=video\_path, width=width, height=height))

skipped\_frames\_counter = 0

        while(cap.isOpened()):

            try :

                ret, frame = cap.read()

                img = cv2.resize(frame, (img\_size, img\_size))

            except:

                print(total\_str)

            output\_img = frame if use\_original\_video\_size\_as\_output\_size else img

            tracker\_rects = []

            if skipped\_frames\_counter == skip\_frames:

# Detecting happens after number of frames have passes specified by 'skip\_frames' variable value

# print("[DETECTING]")

trackers = []

                skipped\_frames\_counter = 0 # reset counter

                np\_img = np.array(img).reshape(-1, img\_size, img\_size, 3)

                start\_time=time.time()

                predictions = sess.run(model.preds, {inputs: model.preprocess(np\_img)})

# print("Detection took %s seconds" % (time.time() - start\_time))

# model.get\_boxes returns a 80 element array containing information about detected classes

# each element contains a list of detected boxes, confidence level ...

detections = model.get\_boxes(predictions, np\_img.shape[1:3])

                np\_detections = np.array(detections)

# Loop only through classes we are interested in

for class\_index in classes.keys():

                    local\_count = 0

                    class\_name = classes[class\_index]

# Loop through detected infos of a class we are interested in

for i in range(len(np\_detections[class\_index])):

                        box = np\_detections[class\_index][i]

                        if np\_detections[class\_index][i][4] >= confidence\_level:

# print("Detected ", class\_name, " with confidence of ", np\_detections[class\_index][i][4])

local\_count += 1

                            startX, startY, endX, endY = box[0], box[1], box[2], box[3]

                            drawRectangleCV2(output\_img, (startX, startY), (endX, endY), (0, 255, 0), 1)

                            drawTextCV2(output\_img, class\_name, (startX, startY), cv2.FONT\_HERSHEY\_SIMPLEX, .5, (0, 0, 255), 1)

# Construct a dlib rectangle object from the bounding box coordinates and then start the dlib correlation

tracker = dlib.correlation\_tracker()

                            rect = dlib.rectangle(int(startX), int(startY), int(endX), int(endY))

                            tracker.start\_track(img, rect)

# Add the tracker to our list of trackers so we can utilize it during skip frames

# Write the total number of detected objects for a given class on this frame

# print(class\_name," : ", local\_count)

else:

# If detection is not happening then track previously detected objects (if any)

# print("[TRACKING]")

skipped\_frames\_counter += 1

# Increase the number frames for which we did not use detection

# Loop through tracker, update each of them and display their rectangle

for tracker in trackers:

                    tracker.update(img)

                    pos = tracker.get\_position()

# Unpack the position object

startX = int(pos.left())

                    startY = int(pos.top())

                    endX = int(pos.right())

                    endY = int(pos.bottom())

# Add the bounding box coordinates to the tracking rectangles list

                    tracker\_rects.append((startX, startY, endX, endY))

# Draw tracking rectangles

                    drawRectangleCV2(output\_img, (startX, startY), (endX, endY), (255, 0, 0), 1)

# Use the centroid tracker to associate the (1) old object centroids with (2) the newly computed object centroids

            objects = ct.update(tracker\_rects)

# Loop over the tracked objects

            for (objectID, centroid) in objects.items():

# Check to see if a trackable object exists for the current object ID

to = trackableObjects.get(objectID, None)

                if to is None:

# If there is no existing trackable object, create one

to = TrackableObject(objectID, centroid)

                else:

                    to.centroids.append(centroid)

# If the object has not been counted, count it and mark it as counted

if not to.counted:

                        total += 1

                        to.counted = True

# Store the trackable object in our dictionary

                trackableObjects[objectID] = to

# Draw both the ID of the object and the centroid of the object on the output frame

object\_id = "ID {}".format(objectID)

                drawTextCV2(output\_img, object\_id, (centroid[0] - 10, centroid[1] - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 1)

                drawCircleCV2(output\_img, (centroid[0], centroid[1]), 2, (0, 255, 0), -1)

# Display the total count so far

total\_str = str(total)

                drawTextCV2(output\_img, total\_str, (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (0, 0, 255), 2)

# Display the current frame (with all annotations drawn up to this point)

cv2.imshow(video\_name, output\_img)

            key = cv2.waitKey(1) & 0xFF

            if key  == ord('q'): # QUIT (exits)

                break

            elif key == ord('p'):

                cv2.waitKey(0) # PAUSE (Enter any key to continue)

    cap.release()

    cv2.destroyAllWindows()

    print("Exited")

    """

    function which will run our code

    will write the number of veicles in the list provided

    """

if \_\_name\_\_ == "\_\_main\_\_":

    countVehicles("/videos/test.mp4")

# Logic for setting the time for each signal