|  |  |
| --- | --- |
|  | import numpy as np |
|  | import os |
|  | import six.moves.urllib as urllib |
|  | import sys |
|  | import tarfile |
|  | import tensorflow as tf |
|  | import zipfile |
|  | import time |
|  | from imutils.video import WebcamVideoStream |
|  | from imutils.video import FPS |
|  | import imutils |
|  | from vehicle import vehicle |
|  | import PIL.Image as Image |
|  | from collections import defaultdict |
|  | from io import StringIO |
|  | from PIL import Image |
|  | import time |
|  | from multiprocessing.pool import ThreadPool |
|  | import threading |
|  | import time |
|  | import openalpr\_api |
|  | from openalpr\_api.rest import ApiException |
|  | import numpy as np |
|  | import cv2 |
|  | import tkinter as tk |
|  | from PIL import Image |
|  | from PIL import ImageTk |
|  | import json |
|  | import re |
|  |  |
|  | if tf.\_\_version\_\_ < '1.4.0': |
|  | raise ImportError('Please upgrade your tensorflow installation to v1.4.\* or later!') |
|  | # This is needed to display the images. |
|  |  |
|  |  |
|  | # This is needed since the notebook is stored in the object\_detection folder. |
|  | sys.path.append("..") |
|  |  |
|  |  |
|  | from utils import label\_map\_util |
|  |  |
|  | from utils import visualization\_utils as vis\_util |
|  |  |
|  | # What model to download. |
|  | MODEL\_NAME = 'Cars' |
|  |  |
|  | # Path to frozen detection graph. This is the actual model that is used for the object detection. |
|  | PATH\_TO\_CKPT = MODEL\_NAME + '/output\_inference\_graph.pb' |
|  |  |
|  | # List of the strings that is used to add correct label for each box. |
|  | PATH\_TO\_LABELS = os.path.join('Cars', 'car\_label\_map.pbtxt') |
|  |  |
|  | NUM\_CLASSES = 1 |
|  |  |
|  |  |
|  |  |
|  | api = openalpr\_api.DefaultApi() |
|  | secret\_key = '' |
|  | country = 'us' |
|  | recognize\_vehicle = 0 |
|  | state = '' |
|  | return\_image = 0 |
|  | topn = 1 |
|  | prewarp = '' |
|  |  |
|  | def getLicensePlateNumber(filer): |
|  | try: |
|  | js = api.recognize\_file(filer, secret\_key, country, recognize\_vehicle=recognize\_vehicle, state=state, return\_image=return\_image, topn=topn, prewarp=prewarp) |
|  |  |
|  | js=js.to\_dict() |
|  | #js=list(str(js)) |
|  | X1=js['results'][0]['coordinates'][0]['x'] |
|  | Y1=js['results'][0]['coordinates'][0]['y'] |
|  | X2=js['results'][0]['coordinates'][2]['x'] |
|  | Y2=js['results'][0]['coordinates'][2]['y'] |
|  | img=cv2.imread(filer) |
|  | rimg=img[Y1:Y2,X1:X2] |
|  | frame3=rimg |
|  | img3 = Image.fromarray(frame3) |
|  | w,h=img3.size |
|  | asprto=w/h |
|  | frame3=cv2.resize(frame3,(150,int(150/asprto))) |
|  | cv2image3 = cv2.cvtColor(frame3, cv2.COLOR\_BGR2RGBA) |
|  | img3 = Image.fromarray(cv2image3) |
|  | imgtk3 = ImageTk.PhotoImage(image=img3) |
|  | display4.imgtk = imgtk3 #Shows frame for display 1 |
|  | display4.configure(image=imgtk3) |
|  | display5.configure(text=js['results'][0]['plate']) |
|  | except ApiException as e: |
|  | print("Exception: \n", e) |
|  |  |
|  |  |
|  | def matchVehicles(currentFrameVehicles,im\_width,im\_height,image): |
|  | if len(vehicles)==0: |
|  | for box,color in currentFrameVehicles: |
|  | (y1,x1,y2,x2)=box |
|  | (x,y,w,h)=(x1\*im\_width,y1\*im\_height,x2\*im\_width-x1\*im\_width,y2\*im\_height-y1\*im\_height) |
|  | X=int((x+x+w)/2) |
|  | Y=int((y+y+h)/2) |
|  | if Y>yl5: |
|  | #cv2.circle(image,(X,Y),2,(0,255,0),4) |
|  | #print('Y=',Y,' y1=',yl1) |
|  | vehicles.append(vehicle((x,y,w,h))) |
|  |  |
|  |  |
|  | else: |
|  | for i in range(len(vehicles)): |
|  | vehicles[i].setCurrentFrameMatch(False) |
|  | vehicles[i].predictNext() |
|  | for box,color in currentFrameVehicles: |
|  | (y1,x1,y2,x2)=box |
|  | (x,y,w,h)=(x1\*im\_width,y1\*im\_height,x2\*im\_width-x1\*im\_width,y2\*im\_height-y1\*im\_height) |
|  | #print((x1\*im\_width,y1\*im\_height,x2\*im\_width,y2\*im\_height),'\n',(x,y,w,h)) |
|  | index = 0 |
|  | ldistance = 999999999999999999999999.9 |
|  | X=int((x+x+w)/2) |
|  | Y=int((y+y+h)/2) |
|  | if Y>yl5: |
|  | #print('Y=',Y,' y1=',yl1) |
|  | #cv2.circle(image,(X,Y),4,(0,0,255),8) |
|  | for i in range(len(vehicles)): |
|  | if vehicles[i].getTracking() == True: |
|  | #print(vehicles[i].getNext(),i) |
|  | distance = ((X-vehicles[i].getNext()[0])\*\*2+(Y-vehicles[i].getNext()[1])\*\*2)\*\*0.5 |
|  |  |
|  | if distance<ldistance: |
|  | ldistance = distance |
|  | index = i |
|  |  |
|  |  |
|  | diagonal=vehicles[index].diagonal |
|  |  |
|  | if ldistance < diagonal: |
|  | vehicles[index].updatePosition((x,y,w,h)) |
|  | vehicles[index].setCurrentFrameMatch(True) |
|  | else: |
|  | #blue for last position |
|  | #cv2.circle(image,tuple(vehicles[index].points[-1]),2,(255,0,0),4) |
|  | #red for predicted point |
|  | #cv2.circle(image,tuple(vehicles[index].getNext()),2,(0,0,255),2) |
|  | #green for test point |
|  | #cv2.circle(image,(X,Y),2,(0,255,0),4) |
|  |  |
|  | #cv2.imshow('culprit',image) |
|  | #time.sleep(5) |
|  | #print(diagonal,' ',ldistance) |
|  | vehicles.append(vehicle((x,y,w,h))) |
|  |  |
|  | for i in range(len(vehicles)): |
|  | if vehicles[i].getCurrentFrameMatch() == False: |
|  | vehicles[i].increaseFrameNotFound() |
|  |  |
|  | #print(len(vehicles)) |
|  | pool = ThreadPool(processes=1) |
|  | def checkRedLightCrossed(img): |
|  | global count |
|  | for v in vehicles: |
|  | if v.crossed==False and len(v.points)>=2: |
|  | x1,y1=v.points[0] |
|  | x2,y2=v.points[-1] |
|  | if y1>yl3 and y2<yl3: |
|  | count+=1 |
|  | v.crossed=True |
|  | bimg=img[int(v.rect[1]):int(v.rect[1]+v.rect[3]), int(v.rect[0]):int(v.rect[0]+v.rect[2])] |
|  | frame2=bimg |
|  | img2 = Image.fromarray(frame2) |
|  | w,h=img2.size |
|  | asprto=w/h |
|  | frame2=cv2.resize(frame2,(250,int(250/asprto))) |
|  | cv2image2 = cv2.cvtColor(frame2, cv2.COLOR\_BGR2RGBA) |
|  | img2 = Image.fromarray(cv2image2) |
|  | imgtk2 = ImageTk.PhotoImage(image=img2) |
|  | display2.imgtk = imgtk2 #Shows frame for display 1 |
|  | display2.configure(image=imgtk2) |
|  | #cv2.imshow('BROKE',bimg) |
|  | name='Rule Breakers/culprit'+str(time.time())+'.jpg' |
|  | cv2.imwrite(name,bimg) |
|  |  |
|  |  |
|  | tstop = threading.Event() |
|  | thread = threading.Thread(target=getLicensePlateNumber, args=(name,)) |
|  | thread.daemon = True |
|  | thread.start() |
|  |  |
|  |  |
|  | #cv2.imwrite('culprit.png',bimg) |
|  | #display3.configure(text=count) |
|  |  |
|  | def checkSpeed(ftime,img): |
|  | for v in vehicles: |
|  | if v.speedChecked==False and len(v.points)>=2: |
|  | x1,y1=v.points[0] |
|  | x2,y2=v.points[-1] |
|  | if y2<yl1 and y2>yl3 and v.entered==False: |
|  | v.enterTime=ftime |
|  | v.entered=True |
|  | elif y2<yl3 and y2 > yl5 and v.exited==False: |
|  | v.exitTime=ftime |
|  | v.exited==False |
|  | v.speedChecked=True |
|  | speed=60/(v.exitTime-v.enterTime) |
|  | print(speed) |
|  | bimg=img[int(v.rect[1]):int(v.rect[1]+v.rect[3]), int(v.rect[0]):int(v.rect[0]+v.rect[2])] |
|  | frame2=bimg |
|  | img2 = Image.fromarray(frame2) |
|  | w,h=img2.size |
|  | asprto=w/h |
|  | frame2=cv2.resize(frame2,(250,int(250/asprto))) |
|  | cv2image2 = cv2.cvtColor(frame2, cv2.COLOR\_BGR2RGBA) |
|  | img2 = Image.fromarray(cv2image2) |
|  | imgtk2 = ImageTk.PhotoImage(image=img2) |
|  | display2.imgtk = imgtk2 #Shows frame for display 1 |
|  | display2.configure(image=imgtk2) |
|  | display3.configure(text=str(speed)[:5]+'Km/hr') |
|  | if speed>60: |
|  |  |
|  | #cv2.imshow('BROKE',bimg) |
|  | name='Rule Breakers/culprit'+str(time.time())+'.jpg' |
|  | cv2.imwrite(name,bimg) |
|  | tstop = threading.Event() |
|  | thread = threading.Thread(target=getLicensePlateNumber, args=(name,)) |
|  | thread.daemon = True |
|  | thread.start() |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | detection\_graph = tf.Graph() |
|  | with detection\_graph.as\_default(): |
|  | od\_graph\_def = tf.GraphDef() |
|  | with tf.gfile.GFile(PATH\_TO\_CKPT, 'rb') as fid: |
|  | serialized\_graph = fid.read() |
|  | od\_graph\_def.ParseFromString(serialized\_graph) |
|  | tf.import\_graph\_def(od\_graph\_def, name='') |
|  |  |
|  |  |
|  | label\_map = label\_map\_util.load\_labelmap(PATH\_TO\_LABELS) |
|  | categories = label\_map\_util.convert\_label\_map\_to\_categories(label\_map, max\_num\_classes=NUM\_CLASSES, use\_display\_name=True) |
|  | category\_index = label\_map\_util.create\_category\_index(categories) |
|  |  |
|  | cap=cv2.VideoCapture('video7.avi') # 0 stands for very first webcam attach |
|  | filename="testoutput.avi" |
|  | codec=cv2.VideoWriter\_fourcc('m','p','4','v')#fourcc stands for four character code |
|  | framerate=10 |
|  | resolution=(640,480) |
|  |  |
|  | VideoFileOutput=cv2.VideoWriter(filename,codec,framerate, resolution) |
|  | vs = WebcamVideoStream(src='Set01\_video01.mp4').start() |
|  | ret,imgF=cap.read(0) |
|  | imgF=Image.fromarray(imgF) |
|  | im\_width, im\_height = imgF.size |
|  | xl1=0 |
|  | xl2=im\_width-1 |
|  | yl1=im\_height\*0.5 |
|  | yl2=yl1 |
|  | ml1=(yl2-yl1)/(xl2-xl1) |
|  | intcptl1=yl1-ml1\*xl1 |
|  |  |
|  | count=0 |
|  | xl3=0 |
|  | xl4=im\_width-1 |
|  | yl3=im\_height\*0.25 |
|  | yl4=yl3 |
|  | ml2=(yl4-yl3)/(xl4-xl3) |
|  | intcptl2=yl3-ml2\*xl3 |
|  |  |
|  | xl5=0 |
|  | xl6=im\_width-1 |
|  | yl5=im\_height\*0.1 |
|  | yl6=yl5 |
|  | ml3=(yl6-yl5)/(xl6-xl5) |
|  | intcptl3=yl5-ml3\*xl5 |
|  | ret=True |
|  | start=time.time() |
|  | c=0 |
|  | sesser=tf.Session(graph=detection\_graph) |
|  | image\_tensor = detection\_graph.get\_tensor\_by\_name('image\_tensor:0') |
|  | # Each box represents a part of the image where a particular object was detected. |
|  | detection\_boxes = detection\_graph.get\_tensor\_by\_name('detection\_boxes:0') |
|  | # Each score represent how level of confidence for each of the objects. |
|  | # Score is shown on the result image, together with the class label. |
|  | detection\_scores = detection\_graph.get\_tensor\_by\_name('detection\_scores:0') |
|  | detection\_classes = detection\_graph.get\_tensor\_by\_name('detection\_classes:0') |
|  | num\_detections = detection\_graph.get\_tensor\_by\_name('num\_detections:0') |
|  |  |
|  | window = tk.Tk() #Makes main window |
|  | window.wm\_title("T.M.S") |
|  | window.columnconfigure(0, {'minsize': 1020}) |
|  | window.columnconfigure(1, {'minsize': 335}) |
|  |  |
|  |  |
|  | frame=tk.Frame(window) |
|  | frame.grid(row=0,column=0,rowspan=5,sticky='N',pady=10) |
|  |  |
|  | frame2=tk.Frame(window) |
|  | frame2.grid(row=0,column=1) |
|  |  |
|  | frame3=tk.Frame(window) |
|  | frame3.grid(row=1,column=1) |
|  |  |
|  | frame4=tk.Frame(window) |
|  | frame4.grid(row=2,column=1) |
|  |  |
|  | frame5=tk.Frame(window) |
|  | frame5.grid(row=3,column=1) |
|  |  |
|  | frame2.rowconfigure(1, {'minsize': 250}) |
|  | frame3.rowconfigure(1, {'minsize': 80}) |
|  | frame4.rowconfigure(1, {'minsize': 150}) |
|  | frame5.rowconfigure(1, {'minsize': 80}) |
|  |  |
|  | vehicles=[] |
|  | def main(sess=sesser): |
|  | '''global masterframe |
|  | global started''' |
|  | if True: |
|  | fTime=time.time() |
|  | \_,image\_np=cap.read(0) |
|  | #image\_np = imutils.resize(image\_np, width=400) |
|  |  |
|  | # Definite input and output Tensors for detection\_graph |
|  |  |
|  |  |
|  | # Expand dimensions since the model expects images to have shape: [1, None, None, 3] |
|  | image\_np\_expanded = np.expand\_dims(image\_np, axis=0) |
|  | # Actual detection. |
|  | (boxes, scores, classes, num) = sess.run( |
|  | [detection\_boxes, detection\_scores, detection\_classes, num\_detections], |
|  | feed\_dict={image\_tensor: image\_np\_expanded}) |
|  |  |
|  |  |
|  | # Visualization of the results of a detection. |
|  | img=image\_np |
|  | imgF,coords=vis\_util.visualize\_boxes\_and\_labels\_on\_image\_array( |
|  | image\_np, |
|  | np.squeeze(boxes), |
|  | np.squeeze(classes).astype(np.int32), |
|  | np.squeeze(scores), |
|  | category\_index, |
|  | use\_normalized\_coordinates=True, |
|  | line\_thickness=2) |
|  |  |
|  | matchVehicles(coords,im\_width,im\_height,imgF) |
|  | checkRedLightCrossed(imgF) |
|  | checkSpeed(fTime,img) |
|  | for v in vehicles: |
|  | if v.getTracking()==True: |
|  |  |
|  | for p in v.getPoints(): |
|  | cv2.circle(image\_np,p,3,(200,150,75),6) |
|  |  |
|  | #print(ymin\*im\_height,xmin\*im\_width,ymax\*im\_height,xmax\*im\_width) |
|  | #cv2.rectangle(image\_np,(int(xmin\*im\_width),int(ymin\*im\_height)),(int(xmax\*im\_width),int(ymax\*im\_height)),(255,0,0),2) |
|  | cv2.line(image\_np, (int(xl1),int(yl1)), (int(xl2),int(yl2)), (0,255,0),3) |
|  | cv2.line(image\_np, (int(xl3),int(yl3)), (int(xl4),int(yl4)), (0,0,255),3) |
|  | cv2.line(image\_np, (int(xl5),int(yl5)), (int(xl6),int(yl6)), (255,0,0),3) |
|  | VideoFileOutput.write(image\_np) |
|  | #print('yola') |
|  | frame=cv2.resize(image\_np,(1020,647)) |
|  | cv2image = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGBA) |
|  | img = Image.fromarray(cv2image) |
|  | imgtk = ImageTk.PhotoImage(image=img) |
|  | display1.imgtk = imgtk #Shows frame for display 1 |
|  | display1.configure(image=imgtk) |
|  | window.after(1, main) |
|  |  |
|  |  |
|  | lbl1 = tk.Label(frame,text='Vehicle Detection And Tracking',font = "verdana 12 bold") |
|  | lbl1.pack(side='top') |
|  |  |
|  | lbl2 = tk.Label(frame2,text='Vehicle Breaking Traffic Rule',font = "verdana 10 bold") |
|  | lbl2.grid(row=0,column=0,sticky ='S',pady=10) |
|  |  |
|  | lbl3 = tk.Label(frame3,text='Veicle Speed',font = "verdana 10 bold") |
|  | lbl3.grid(row=0,column=0,sticky ='S',pady=10) |
|  |  |
|  |  |
|  | lbl4 = tk.Label(frame4,text='Detected License Plate',font = "verdana 10 bold") |
|  | lbl4.grid(row=0,column=0) |
|  |  |
|  | lbl5 = tk.Label(frame5,text='Extracted License Plate Number',font = "verdana 10 bold") |
|  | lbl5.grid(row=0,column=0) |
|  |  |
|  | display1 = tk.Label(frame) |
|  | display1.pack(side='bottom') #Display 1 |
|  |  |
|  | display2 = tk.Label(frame2) |
|  | display2.grid(row=1,column=0) #Display 2 |
|  |  |
|  |  |
|  | display3 = tk.Label(frame3,text="",font = "verdana 14 bold",fg='red') |
|  | display3.grid(row=1,column=0) |
|  |  |
|  | display4 = tk.Label(frame4) |
|  | display4.grid(row=1,column=0) |
|  |  |
|  | display5 = tk.Label(frame5,text="",font = "verdana 24 bold",fg='green') |
|  | display5.grid(row=1,column=0) |
|  | masterframe=None |
|  | started= False |
|  | def stream(): |
|  | global masterframe |
|  | global started |
|  | global c |
|  | global tim |
|  | cap=cv2.VideoCapture('vid1.mp4') |
|  | while True: |
|  | started,masterframe=cap.read() |
|  | time.sleep(0.034) |
|  |  |
|  |  |
|  | '''thread = threading.Thread(target=stream) |
|  | thread.daemon = True |
|  | thread.start()''' |
|  | with detection\_graph.as\_default(): |
|  | with tf.Session(graph=detection\_graph) as sess: |
|  | sesser=sess |
|  | main(sess) #Display |
|  | window.mainloop() |