Final Lab: Object Detection

Name: Sumna Imran

Seat number: B18101105

Section: A

Year: 4th

Semester: 7th

Date: 31/May/2022

Course instructor:
Miss Farzeen Ashfaq

Final Lab

Question #01:

Use the object detection algorithms (YOLO, Mask RCNN, Fast RCNN, Faster RCNN, Zero-shot learning) n any vision-based (video/image) dataset for counting the vehicles on the road and predict traffic congestion.

Deep neural networks have gained fame for their capability to process visual information. And in the past few years, they have become a key component of many computer vision applications.

Among the key problems neural networks can solve is detecting and localizing objects in images. Object detection is used in many different domains, including autonomous driving, video surveillance, and healthcare.

Object Detection:

Object detection is a computer vision technology that localizes and identifies objects in an image. Due to object detection's versatility, object detection has emerged in the last few years as the most commonly used computer vision technology. To explore the concept of object detection it's useful, to begin with, image classification. Image classification goes through levels of incremental complexity.

1. **Image classification** aims at assigning an image to one of several different categories (e.g. car, dog, cat, human, etc.), essentially answering the question *"What is in this picture?"*. One image has only one category assigned to it.
2. **Object localization** allows us to locate our object in the image, so our question changes to *"What is it and where it is?"*.
3. **Object detection provides** the tools for doing just that – finding all the objects in an image and drawing the so-called **bounding boxes** around them.

In a real real-life scenario, we need to go beyond locating just one object but rather multiple objects in one image. For example, a **self-driving car** has to find the location of other cars, traffic lights, signs, and humans and take appropriate action based on this information.

YOLO algorithm gives much better performance on all the parameters we discussed along with a high fps for real-time usage. YOLO algorithm is an algorithm based on regression, instead of selecting the interesting part of an Image, it predicts classes and bounding boxes for the whole image in **one run of the Algorithm.**

HOW THE YOLO ALGORITHM WORKS

YOLO ALGORITHM WORKS USING THE FOLLOWING THREE TECHNIQUES:

- Residual blocks
- Bounding box regression
- Intersection Over Union (IOU)

-File

- yoloV3.weights
- yoloV3.cfg
- Vehicle Video

Library

- numpy
- OpenCVCV

Platform:

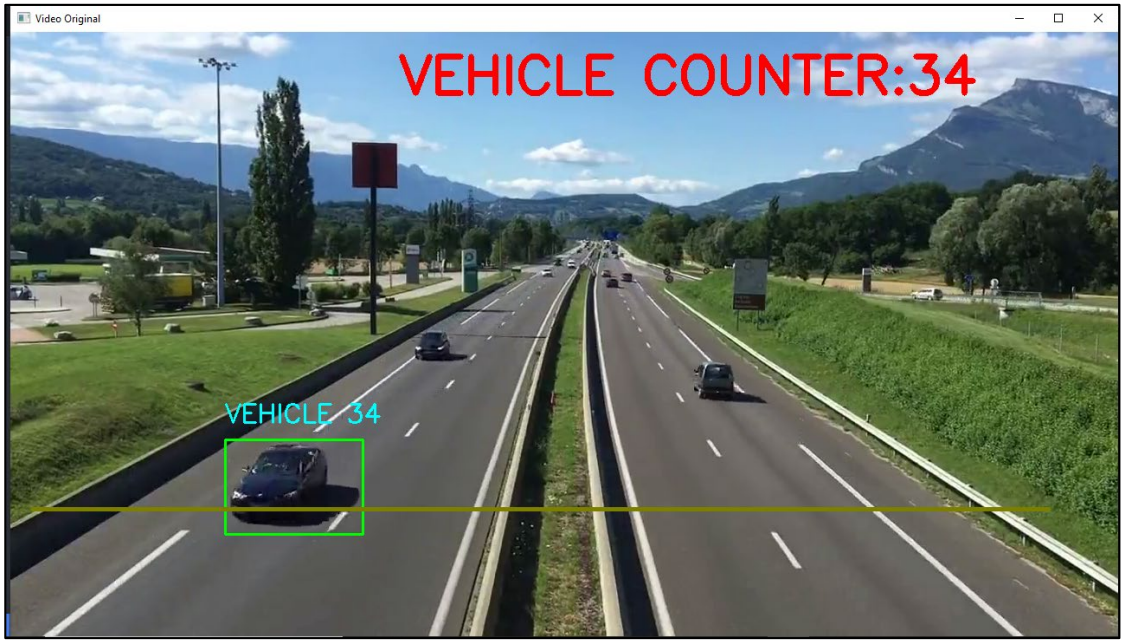
The platform I used for this task was **visual studio code** (VS code). And the language I used for implementing this task was python as you know python is becoming famous for deep learning and machine learning implementation.

I have to install some dependencies for this which were ***Numpy*** (is an open-source Python library that makes it easy to complex numerical operations with large datasets) and

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products.

DataSet:

For this task, I am taking a video of vehicles as my input for the program can detect the vehicles and count the number of vehicles passing the line. I have taken the video from the following link: *https://drive.google.com/file/d/1aUI1k47cUnpk8SxYqNwnUmOMQvIX6chO/view*



Main Components for Vehicle Counter:

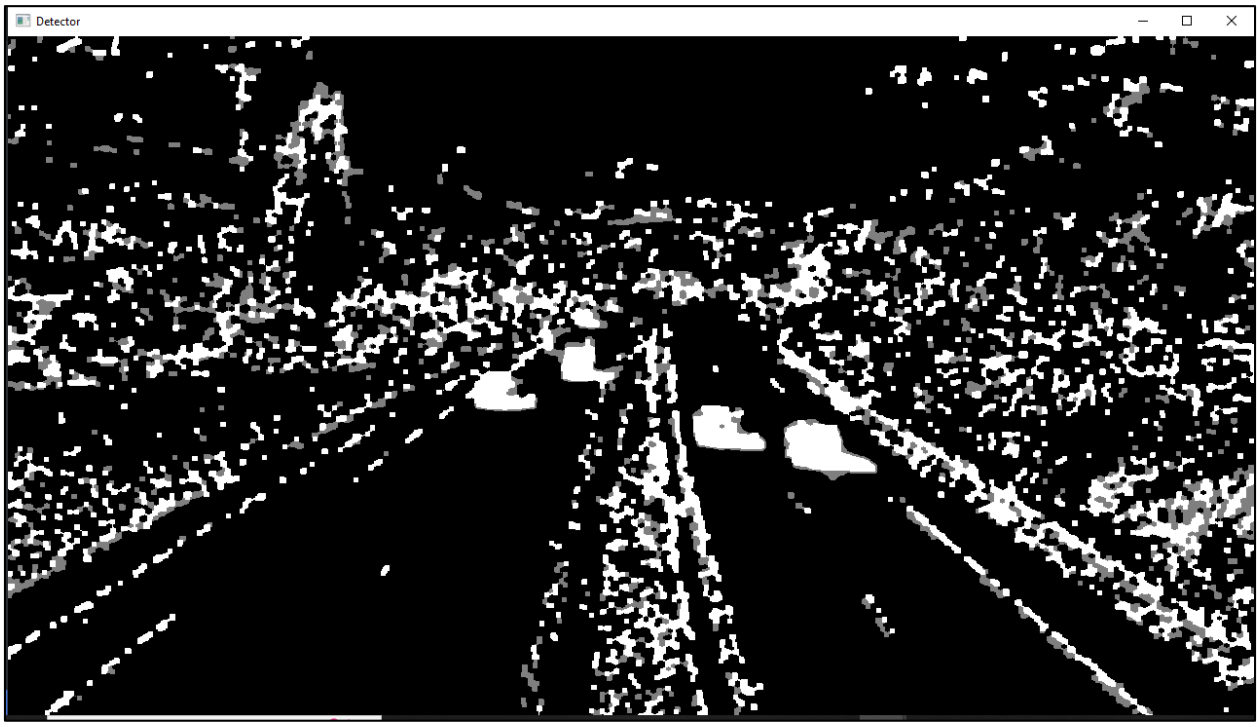
1. Detector
2. Tracker
3. Counter

Code:

```
from tkinter import CENTER
import cv2
from cv2 import dilate
import numpy as np

#web camera
cap = cv2.VideoCapture('video.mp4')

#declaring min width and height of rectangle
min_width_rectangle = 80
min_height_rectangle = 80

count_line_position = 550

#Innitialize Subtractor
algo = cv2.createBackgroundSubtractorMOG2()

def center_handle(x,y,w,h):
    x1 = int(w/2)
    y1 = int(y/2)
    cx = x + x1
    cy = y + y1
    return cx, cy

detect = []
offset = 6  #allowable error between pixels
counter = 0


while True:
    ret,frame1 = cap.read()
    grey = cv2.cvtColor(frame1,cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(grey,(3,3),5)
    #applying on each frame
    img_sub = algo.apply(blur)
    dilat = cv2.dilate(img_sub,np.ones((5,5)))
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
    dilatada = cv2.morphologyEx(dilat,cv2.MORPH_CLOSE,kernel)
    dilatada = cv2.morphologyEx(dilatada,cv2.MORPH_CLOSE,kernel)
    countershape,h =
cv2.findContours(dilatada,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

    cv2.line(frame1,(25,count_line_position),(1200,count_line_position),(255,127,
0),3)
```

```
    #creating rectangle for objects/vehicle
    for(i,c) in enumerate(countershape):
        (x,y,w,h) = cv2.boundingRect(c)
        validate_counter = (w >= min_width_rectangle) and (h>=
min_height_rectangle)
        if not validate_counter:
            continue

        cv2.rectangle(frame1,(x,y),(x+w,y+h),(0,255,0),2)
        cv2.putText(frame1,"VEHICLE "+str(counter),(x,y-
20),cv2.FONT_HERSHEY_SIMPLEX,1,(255,254,0),2)

        center = center_handle(x,y,w,h)
        detect.append(center)
        cv2.circle(frame1,center,4,(0,0,255),-1)

        for(x,y) in detect:
            if y < (count_line_position +offset) and y >(count_line_position-
offset):
                counter+=1
            cv2.line(frame1,(25,count_line_position),(1200,count_line_position),(
0,127,125),3)
            detect.remove((x,y))
            print("Vehicle Counter:"+str(counter))

    cv2.putText(frame1,"VEHICLE
COUNTER:"+str(counter),(450,70),cv2.FONT_HERSHEY_SIMPLEX,2,(0,0,255),5)

    #cv2.imshow('Detector', dilatada)
    cv2.imshow('Video Original', frame1)

    if cv2.waitKey(1)==13:
        break

cv2.destroyWindow()
cap.release()
```

Backend working of Detector:



Output: