git clone https://github.com/AlexeyAB/darknet

%cd darknet

sed -i 's/OPENCV=0/OPENCV=1/' Makefile

sed -i 's/GPU=0/GPU=1/' Makefile

sed -i 's/CUDNN=0/CUDNN=1/' Makefile

sed -i 's/CUDNN\_HALF=0/CUDNN\_HALF=1/' Makefile

sed -i 's/LIBSO=0/LIBSO=1/' Makefile

import math

# Creating a list of image files of the dataset.

data\_path = './data/obj/train/'

files = os.listdir(data\_path)

img\_arr = []

# Displaying 4 images only.

num = 4

# Appending the array of images to a list.

for fimg in files:

if fimg.endswith('.jpg'):

demo = img.imread(data\_path+fimg)

img\_arr.append(demo)

if len(img\_arr) == num:

break

# Plotting the images using matplotlib.

\_, axs = plt.subplots(math.floor(num/2), math.ceil(num/2), figsize=(50, 28))

axs = axs.flatten()

for cent, ax in zip(img\_arr, axs):

ax.imshow(cent)

plt.show()

wget https://github.com/AlexeyAB/darknet/releases/download/darknet\_yolo\_v3\_optimal/yolov4.conv.137

./darknet detector train data/obj.data cfg/yolov4-obj.cfg yolov4.conv.137 -dont\_show -map

def yolo\_det(frame, config\_file, data\_file, batch\_size, weights, threshold, output, network,

class\_names, class\_colors, save = False, out\_path = ''):

prev\_time = time.time()

# Preprocessing the input image.

width = darknet.network\_width(network)

height = darknet.network\_height(network)

darknet\_image = darknet.make\_image(width, height, 3)

image\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

image\_resized = cv2.resize(image\_rgb, (width, height))

# Passing the image to the detector and store the detections

darknet.copy\_image\_from\_bytes(darknet\_image, image\_resized.tobytes())

detections = darknet.detect\_image(network, class\_names, darknet\_image, thresh=threshold)

darknet.free\_image(darknet\_image)

# Plotting the deetections using darknet in-built functions

image = darknet.draw\_boxes(detections, image\_resized, class\_colors)

print(detections)

if save:

im = cv2.cvtColor(image, cv2.COLOR\_RGB2BGR)

file\_name = out\_path + '-det.jpg'

cv2.imwrite(os.path.join(output, file\_name), im)

# Calculating time taken and FPS for detection

det\_time = time.time() - prev\_time

fps = int(1/(time.time() - prev\_time))

print("Detection time: {}".format(det\_time))

# Resizing predicted bounding box from 416x416 to input image resolution

out\_size = frame.shape[:2]

in\_size = image\_resized.shape[:2]

coord, scores = resize\_bbox(detections, out\_size, in\_size)

return coord, scores, det\_time

pip install paddlepaddle-gpu

pip install "paddleocr>=2.0.1"

from paddleocr import PaddleOCR

ocr = PaddleOCR(lang='en',rec\_algorithm='CRNN')

result = ocr.ocr(cr\_img, cls=False, det=False)

def crop(image, coord):

# Cropping is done by -> image[y1:y2, x1:x2].

cr\_img = image[coord[1]:coord[3], coord[0]:coord[2]]

return cr\_img

# Variables storing colors and fonts.

font = cv2.FONT\_HERSHEY\_SIMPLEX

blue\_color = (255,0,0)

white\_color = (255,255,255)

black\_color = (0,0,0)

green\_color = (0,255,0)

yellow\_color = (178, 247, 218)

def test\_img(input, config\_file, weights, out\_path):

# Loading darknet network and classes along with the bbox colors.

network, class\_names, class\_colors = darknet.load\_network(

config\_file,

data\_file,

weights,

batch\_size= batch\_size

)

# Reading the image and performing YOLOv4 detection.

img = cv2.imread(input)

bboxes, scores, det\_time = yolo\_det(img, config\_file, data\_file, batch\_size, weights, thresh,

out\_path, network, class\_names, class\_colors)

# Extracting or cropping the license plate and applying the OCR.

for bbox in bboxes:

cr\_img = crop(img, bbox)

result = ocr.ocr(cr\_img, cls=False, det=False)

ocr\_res = result[0][0]

rec\_conf = result[0][1]

# Plotting the predictions using OpenCV.

(label\_width,label\_height), baseline = cv2.getTextSize(ocr\_res , font, 2, 3)

top\_left = tuple(map(int,[int(bbox[0]),int(bbox[1])-(label\_height+baseline)]))

top\_right = tuple(map(int,[int(bbox[0])+label\_width,int(bbox[1])]))

org = tuple(map(int,[int(bbox[0]),int(bbox[1])-baseline]))

cv2.rectangle(img, (int(bbox[0]), int(bbox[1])), (int(bbox[2]), int(bbox[3])), blue\_color, 2)

cv2.rectangle(img, top\_left, top\_right, blue\_color,-1)

cv2.putText(img, ocr\_res, org, font, 2, white\_color,3)

# Writing output image.

file\_name = os.path.join(out\_path, 'out\_' + input.split('/')[-1])

cv2.imwrite(file\_name, img)

# Importing libraries and required functionalities.

# DeepSORT imports.

%cd ./deep\_sort

from application\_util import preprocessing

from deep\_sort import nn\_matching

from deep\_sort.detection import Detection

from deep\_sort.tracker import Tracker

from tools\_deepsort import generate\_detections as gdet

import uuid

# Required libraries.

import os

import glob

import random

import time

import cv2

import numpy as np

import darknet

import subprocess

import sys

from PIL import Image

import matplotlib

import matplotlib.pyplot as plt

%matplotlib inline

# Darknet object detector imports.

%cd ./darknet

from darknet\_images import load\_images

from darknet\_images import image\_detection

# Declaring important variables.

# Path of Configuration file of YOLOv4.

config\_file = './darknet/cfg/yolov4-obj.cfg'

# Path of obj.data file.

data\_file = './darknet/data/obj.data'

# Batch size of data passed to the detector.

batch\_size = 1

# Path to trained YOLOv4 weights.

weights = './checkpoint/yolov4-obj\_best.weights'

# Confidence threshold.

thresh = 0.6

# Calling the function.

input\_dir = 'car-img.jpg'

out\_path = '/content/'

test\_img(input\_dir, config\_file, weights,out\_path)

out\_img = cv2.imread('./out\_car-img.jpg')

cv2.imshow(out\_img)

# video test

def test\_vid(vid\_dir, config\_file, weights,out\_path):

# Declaring variables for video processing.

cap = cv2.VideoCapture(vid\_dir)

codec = cv2.VideoWriter\_fourcc(\*'XVID')

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

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

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

file\_name = os.path.join(out\_path, 'out\_' + vid\_dir.split('/')[-1])

out = cv2.VideoWriter(file\_name, codec, fps, (width, height))

# Frame count variable.

ct = 0

# Loading darknet network and classes along with the bbox colors.

network, class\_names, class\_colors = darknet.load\_network(

config\_file,

data\_file,

weights,

batch\_size= batch\_size

)

# Reading video frame by frame.

while(cap.isOpened()):

ret, img = cap.read()

if ret == True:

print(ct)

# Noting time for calculating FPS.

prev\_time = time.time()

# Performing the YOLOv4 detection.

bboxes, scores, det\_time = yolo\_det(img, config\_file, data\_file, batch\_size, weights, thresh,

out\_path, network, class\_names, class\_colors)

# Extracting or cropping the license plate and applying the OCR.

if list(bboxes):

for bbox in bboxes:

cr\_img, cord = crop(img, bbox)

result = ocr.ocr(cr\_img, cls=False, det=False)

ocr\_res = result[0][0]

rec\_conf = result[0][1]

# Plotting the predictions using OpenCV.

txt = ocr\_res

(label\_width,label\_height), baseline = cv2.getTextSize(ocr\_res , font,2,3)

top\_left = tuple(map(int,[int(bbox[0]),int(bbox[1])-(label\_height+baseline)]))

top\_right = tuple(map(int,[int(bbox[0])+label\_width,int(bbox[1])]))

org = tuple(map(int,[int(bbox[0]),int(bbox[1])-baseline]))

cv2.rectangle(img, (int(bbox[0]), int(bbox[1])), (int(bbox[2]), int(bbox[3])), blue\_color, 2)

cv2.rectangle(img, top\_left, top\_right, blue\_color, -1)

cv2.putText(overlay\_img,txt, org, font, 2, white\_color, 3)

#cv2.imwrite('/content/{}.jpg'.format(ct), img)

# Calculating time taken and FPS for the whole process.

tot\_time = time.time() - prev\_time

fps = 1/tot\_time

# Writing information onto the frame and saving it to be processed in a video.

cv2.putText(img, 'frame: %d fps: %s' % (ct, fps),

(0, int(100 \* 1)), cv2.FONT\_HERSHEY\_PLAIN, 5, (0, 0, 255), thickness=2)

out.write(img)

ct = ct + 1

else:

break

# output directly

# Calling the function.

input\_dir = './Pexels Videos 2103099.mp4'

out\_path = '/content/'

test\_vid(input\_dir, config\_file, weights,out\_path)

from IPython.display import HTML

from base64 import b64encode

# Input video path.

save\_path = './out\_Pexels Videos 2103099.mp4'

# Compressed video path.

compressed\_path = "./compressed.mp4"

#compressing the size of video to avoid crashing.

os.system(f"ffmpeg -i {save\_path} -vcodec libx264 {compressed\_path}")

# Show video.

mp4 = open(compressed\_path,'rb').read()

data\_url = "data:video/mp4;base64," + b64encode(mp4).decode()

HTML("""

<video width=400 controls> <source src="%s" type="video/mp4">

</video>

""" % data\_url)