import os

import cv2

import numpy as np

import tensorflow as tf

import tensorflow.compat.v1 as tf

import sys

# This is needed since the notebook is stored in the object\_detection folder.

sys.path.append("..")

# Import utilites

from object\_detection.utils import label\_map\_util

from object\_detection.utils import visualization\_utils as vis\_util

# Name of the directory containing the object detection module we're using

MODEL\_NAME = 'inference\_graph'

VIDEO\_NAME = 'testvideo.mp4'

# Grab path to current working directory

CWD\_PATH = os.getcwd()

# Path to frozen detection graph .pb file, which contains the model that is used

# for object detection.

PATH\_TO\_CKPT = os.path.join(CWD\_PATH,MODEL\_NAME,'frozen\_inference\_graph.pb')

# Path to label map file

PATH\_TO\_LABELS = os.path.join(CWD\_PATH,'training','labelmap.pbtxt')

# Path to video

PATH\_TO\_VIDEO = os.path.join(CWD\_PATH,VIDEO\_NAME)

# Number of classes the object detector can identify

NUM\_CLASSES = 6

# Load the label map.

# Label maps map indices to category names, so that when our convolution

# network predicts `5`, we know that this corresponds to `king`.

# Here we use internal utility functions, but anything that returns a

# dictionary mapping integers to appropriate string labels would be fine

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)

# Load the Tensorflow model into memory.

detection\_graph = tf.Graph()

with detection\_graph.as\_default():

od\_graph\_def = tf.GraphDef()

with tf.io.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='')

sess = tf.Session(graph=detection\_graph)

# Define input and output tensors (i.e. data) for the object detection classifier

# Input tensor is the image

image\_tensor = detection\_graph.get\_tensor\_by\_name('image\_tensor:0')

# Output tensors are the detection boxes, scores, and classes

# 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 represents level of confidence for each of the objects.

# The 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')

# Number of objects detected

num\_detections = detection\_graph.get\_tensor\_by\_name('num\_detections:0')

# Open video file

video = cv2.VideoCapture(PATH\_TO\_VIDEO)

while(video.isOpened()):

# Acquire frame and expand frame dimensions to have shape: [1, None, None, 3]

# i.e. a single-column array, where each item in the column has the pixel RGB value

ret, frame = video.read()

frame\_expanded = np.expand\_dims(frame, axis=0)

# Perform the actual detection by running the model with the image as input

(boxes, scores, classes, num) = sess.run(

[detection\_boxes, detection\_scores, detection\_classes, num\_detections],

feed\_dict={image\_tensor: frame\_expanded})

# Draw the results of the detection (aka 'visulaize the results')

vis\_util.visualize\_boxes\_and\_labels\_on\_image\_array(

frame,

np.squeeze(boxes),

np.squeeze(classes).astype(np.int32),

np.squeeze(scores),

category\_index,

use\_normalized\_coordinates=True,

line\_thickness=8,

min\_score\_thresh=0.80)

# All the results have been drawn on the frame, so it's time to display it.

cv2.imshow('Object detector', frame)

# Press 'q' to quit

if cv2.waitKey(1) == ord('q'):

break

# Clean up

video.release()

cv2.destroyAllWindows()