Detector.py

# Import packages

import os

import argparse

import cv2

import numpy as np

import sys

import time

from threading import Thread

import importlib.util

# Define VideoStream class to handle streaming of video from webcam in separate processing thread

class VideoStream:

"""Camera object that controls video streaming from the Picamera"""

def \_\_init\_\_(self,resolution=(640,480),framerate=30):

# Initialize the PiCamera and the camera image stream

self.stream = cv2.VideoCapture(0)

ret = self.stream.set(cv2.CAP\_PROP\_FOURCC, cv2.VideoWriter\_fourcc(\*'MJPG'))

ret = self.stream.set(3,resolution[0])

ret = self.stream.set(4,resolution[1])

# Read first frame from the stream

(self.grabbed, self.frame) = self.stream.read()

# Variable to control when the camera is stopped

self.stopped = False

def start(self):

# Start the thread that reads frames from the video stream

Thread(target=self.update,args=()).start()

return self

def update(self):

# Keep looping indefinitely until the thread is stopped

while True:

# If the camera is stopped, stop the thread

if self.stopped:

# Close camera resources

self.stream.release()

return

# Otherwise, grab the next frame from the stream

(self.grabbed, self.frame) = self.stream.read()

def read(self):

# Return the most recent frame

return self.frame

def stop(self):

# Indicate that the camera and thread should be stopped

self.stopped = True

# Define and parse input arguments

parser = argparse.ArgumentParser()

parser.add\_argument('--modeldir', help='Folder the .tflite file is located in',

required=True)

parser.add\_argument('--graph', help='Name of the .tflite file, if different than detect.tflite',

default='detect.tflite')

parser.add\_argument('--labels', help='Name of the labelmap file, if different than labelmap.txt',

default='labelmap.txt')

parser.add\_argument('--threshold', help='Minimum confidence threshold for displaying detected objects',

default=0.5)

parser.add\_argument('--resolution', help='Desired webcam resolution in WxH. If the webcam does not support the resolution entered, errors may occur.',

default='800x640')

parser.add\_argument('--edgetpu', help='Use Coral Edge TPU Accelerator to speed up detection',

action='store\_true')

args = parser.parse\_args()

MODEL\_NAME = args.modeldir

GRAPH\_NAME = args.graph

LABELMAP\_NAME = args.labels

min\_conf\_threshold = float(args.threshold)

resW, resH = args.resolution.split('x')

imW, imH = int(resW), int(resH)

use\_TPU = args.edgetpu

# Import TensorFlow libraries

# If tensorflow is not installed, import interpreter from tflite\_runtime, else import from regular tensorflow

# If using Coral Edge TPU, import the load\_delegate library

pkg = importlib.util.find\_spec('tensorflow')

if pkg is None:

from tflite\_runtime.interpreter import Interpreter

if use\_TPU:

from tflite\_runtime.interpreter import load\_delegate

else:

from tensorflow.lite.python.interpreter import Interpreter

if use\_TPU:

from tensorflow.lite.python.interpreter import load\_delegate

# If using Edge TPU, assign filename for Edge TPU model

if use\_TPU:

# If user has specified the name of the .tflite file, use that name, otherwise use default 'edgetpu.tflite'

if (GRAPH\_NAME == 'detect.tflite'):

GRAPH\_NAME = 'edgetpu.tflite'

# Get path to current working directory

CWD\_PATH = os.getcwd()

# Path to .tflite file, which contains the model that is used for object detection

PATH\_TO\_CKPT = os.path.join(CWD\_PATH,MODEL\_NAME,GRAPH\_NAME)

# Path to label map file

PATH\_TO\_LABELS = os.path.join(CWD\_PATH,MODEL\_NAME,LABELMAP\_NAME)

# Load the label map

with open(PATH\_TO\_LABELS, 'r') as f:

labels = [line.strip() for line in f.readlines()]

# Have to do a weird fix for label map if using the COCO "starter model" from

# https://www.tensorflow.org/lite/models/object\_detection/overview

# First label is '???', which has to be removed.

if labels[0] == '???':

del(labels[0])

IGNORE = set(['person','bicycle','car','motorcycle','airplane','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','sandwich','broccoli','carrot','hot dog','pizza','donut','cake','chair','couch','potted plant','bed','dining table','toilet',

'tv','laptop','mouse','remote','keyboard','cell phone','microwave','oven','toaster','sink','refrigerator','book','clock','vase','scissors','teddy bear','hair drier',

'toothbrush'])

# Load the Tensorflow Lite model.

# If using Edge TPU, use special load\_delegate argument

if use\_TPU:

interpreter = Interpreter(model\_path=PATH\_TO\_CKPT,

experimental\_delegates=[load\_delegate('libedgetpu.so.1.0')])

print(PATH\_TO\_CKPT)

else:

interpreter = Interpreter(model\_path=PATH\_TO\_CKPT)

interpreter.allocate\_tensors()

# Get model details

input\_details = interpreter.get\_input\_details()

output\_details = interpreter.get\_output\_details()

height = input\_details[0]['shape'][1]

width = input\_details[0]['shape'][2]

floating\_model = (input\_details[0]['dtype'] == np.float32)

input\_mean = 127.5

input\_std = 127.5

# Initialize frame rate calculation

frame\_rate\_calc = 1

freq = cv2.getTickFrequency()

# Initialize video stream

videostream = VideoStream(resolution=(imW,imH),framerate=30).start()

time.sleep(1)

#for frame1 in camera.capture\_continuous(rawCapture, format="bgr",use\_video\_port=True):

while True:

# Start timer (for calculating frame rate)

t1 = cv2.getTickCount()

# Grab frame from video stream

frame1 = videostream.read()

# Acquire frame and resize to expected shape [1xHxWx3]

frame = frame1.copy()

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

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

input\_data = np.expand\_dims(frame\_resized, axis=0)

# Normalize pixel values if using a floating model (i.e. if model is non-quantized)

if floating\_model:

input\_data = (np.float32(input\_data) - input\_mean) / input\_std

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

interpreter.set\_tensor(input\_details[0]['index'],input\_data)

interpreter.invoke()

# Retrieve detection results

boxes = interpreter.get\_tensor(output\_details[0]['index'])[0] # Bounding box coordinates of detected objects

classes = interpreter.get\_tensor(output\_details[1]['index'])[0] # Class index of detected objects

scores = interpreter.get\_tensor(output\_details[2]['index'])[0] # Confidence of detected objects

#num = interpreter.get\_tensor(output\_details[3]['index'])[0] # Total number of detected objects (inaccurate and not needed)

# Loop over all detections and draw detection box if confidence is above minimum threshold

for i in range(len(scores)):

if ((scores[i] > min\_conf\_threshold) and (scores[i] <= 1.0)):

# Get bounding box coordinates and draw box

# Interpreter can return coordinates that are outside of image dimensions, need to force them to be within image using max() and min()

ymin = int(max(1,(boxes[i][0] \* imH)))

xmin = int(max(1,(boxes[i][1] \* imW)))

ymax = int(min(imH,(boxes[i][2] \* imH)))

xmax = int(min(imW,(boxes[i][3] \* imW)))

object\_name = labels[int(classes[i])] # Look up object name from "labels" array using class index

if object\_name in IGNORE:

continue

cv2.rectangle(frame, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)

# Draw label

label = '%s ' % (object\_name) # Example: 'orange P10 w=135g'

labelSize, baseLine = cv2.getTextSize(label, cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, 2) # Get font size

label\_ymin = max(ymin, labelSize[1] + 10) # Make sure not to draw label too close to top of window

cv2.rectangle(frame, (xmin, label\_ymin-labelSize[1]-10), (xmin+labelSize[0], label\_ymin+baseLine-10), (255, 255, 255), cv2.FILLED) # Draw white box to put label text in

cv2.putText(frame, label, (xmin, label\_ymin-7), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 0), 2) # Draw label text

# Draw framerate in corner of frame

#cv2.putText(frame,'FPS: {0:.2f}'.format(frame\_rate\_calc),(30,50),cv2.FONT\_HERSHEY\_SIMPLEX,1,(255,255,0),2,cv2.LINE\_AA)

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

cv2.imshow('Object detector', frame)

# Calculate framerate

t2 = cv2.getTickCount()

time1 = (t2-t1)/freq

frame\_rate\_calc= 1/time1

# Press 'q' to quit

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

break

# Clean up

cv2.destroyAllWindows()

videostream.stop()