**CODE**

**DATA TRAINING**

from \_\_future\_\_ import print\_function

import keras

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten

from keras.layers import Conv2D, MaxPooling2D

from keras import backend as K

batch\_size = 128

num\_classes = 10

epochs = 12

img\_rows, img\_cols = 28, 28

# the data, split between train and test sets

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

if K.image\_data\_format() == 'channels\_first':

    x\_train = x\_train.reshape(x\_train.shape[0], 1, img\_rows, img\_cols)

    x\_test = x\_test.reshape(x\_test.shape[0], 1, img\_rows, img\_cols)

    input\_shape = (1, img\_rows, img\_cols)

else:

    x\_train = x\_train.reshape(x\_train.shape[0], img\_rows, img\_cols, 1)

    x\_test = x\_test.reshape(x\_test.shape[0], img\_rows, img\_cols, 1)

    input\_shape = (img\_rows, img\_cols, 1)

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

x\_train /= 255

x\_test /= 255

print('x\_train shape:', x\_train.shape)

print(x\_train.shape[0], 'train samples')

print(x\_test.shape[0], 'test samples')

# 6. Preprocess class labels

# convert class vectors to binary class matrices

y\_train = keras.utils.to\_categorical(y\_train, num\_classes)

y\_test = keras.utils.to\_categorical(y\_test, num\_classes)

# 7. Define model architecture

model = Sequential()

model.add(Conv2D(32, kernel\_size=(3, 3),

                 activation='relu',

                 input\_shape=input\_shape))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(num\_classes, activation='softmax'))

# 8. Compile model

model.compile(loss=keras.losses.categorical\_crossentropy,

              optimizer=keras.optimizers.Adadelta(),

              metrics=['accuracy'])

# 9. Fit model on training data

model.fit(x\_train, y\_train,

          batch\_size=batch\_size,

          epochs=epochs,

          verbose=1,

          validation\_data=(x\_test, y\_test))

# 10. Evaluate model on test data

score = model.evaluate(x\_test, y\_test, verbose=0)

print('Test loss:', score[0])

print('Test accuracy:', score[1])

# 11. Save model

model.save('mnist\_trained\_model.h5')  # creates a HDF5 file

### 

### Pi-cam

import numpy as np

#from skimage import data, io

#from matplotlib import pyplot as plt

from skimage import img\_as\_ubyte        #convert float to uint8

from skimage.color import rgb2gray

import cv2

import datetime

import argparse

import imutils

import time

from time import sleep

from imutils.video import VideoStream

from keras.models import load\_model

model=load\_model('mnist\_trained\_model.h5')

# construct the argument parse and parse the arguments

ap = argparse.ArgumentParser()

ap.add\_argument("-p", "--picamera", type=int, default=-1,

    help="whether or not the Raspberry Pi camera should be used")

args = vars(ap.parse\_args())

# initialize the video stream and allow the cammera sensor to warmup

vs = VideoStream(usePiCamera=args["picamera"] > 0).start()

time.sleep(2.0)

def ImagePreProcess(im\_orig):

    im\_gray = rgb2gray(im\_orig)

    #io.imshow(im\_gray)

    #plt.show()

    img\_gray\_u8 = img\_as\_ubyte(im\_gray)

    #cv2.imshow("Window", img\_gray\_u8)

    #io.imshow(img\_gray\_u8)

    #plt.show()

    #Convert grayscale image to binary

    (thresh, im\_bw) = cv2.threshold(img\_gray\_u8, 128, 255, cv2.THRESH\_BINARY | cv2.THRESH\_OTSU)

    #cv2.imshow("Window", im\_bw)

    #resize using opencv

    img\_resized = cv2.resize(im\_bw,(28,28))

    #cv2.imshow("Window", img\_resized)

    ############################################################

    #resize using sciikit

    #im\_resize = resize(im,(28,28), mode='constant')

    #io.imshow(im\_resize)

    #plt.show()

    #cv2.imshow("Window", im\_resize)

    ##########################################################

    #invert image

    im\_gray\_invert = 255 - img\_resized

    #cv2.imshow("Window", im\_gray\_invert)

    ####################################

    im\_final = im\_gray\_invert.reshape(1,28,28,1)

    # the below output is a array of possibility of respective digit

    ans = model.predict(im\_final)

    print(ans)

    # choose the digit with greatest possibility as predicted dight

    ans = ans[0].tolist().index(max(ans[0].tolist()))

    print('DNN predicted digit is: ',ans)

def main():

    # loop over the frames from the video stream

    while True:

        try:

            # grab the frame from the threaded video stream and resize it

            # to have a maximum width of 400 pixels

            frame = vs.read()

            frame = imutils.resize(frame, width=400)

            # draw the timestamp on the frame

            timestamp = datetime.datetime.now()

            ts = timestamp.strftime("%A %d %B %Y %I:%M:%S%p")

            cv2.putText(frame, ts, (10, frame.shape[0] - 10), cv2.FONT\_HERSHEY\_SIMPLEX,

                0.35, (0, 0, 255), 1)

            # show the frame

            cv2.imshow("Frame", frame)

            key = cv2.waitKey(1) & 0xFF

            # if the `q` key was pressed, break from the loop

            if key == ord("q"):

                break

                # do a bit of cleanup

                cv2.destroyAllWindows()

                vs.stop()

            elif key == ord("t"):

                cv2.imwrite("num.jpg", frame)

                im\_orig = cv2.imread("num.jpg")

                ImagePreProcess(im\_orig)

            else:

                pass

        except KeyboardInterrupt:

            # do a bit of cleanup

            cv2.destroyAllWindows()

            vs.stop()

if \_\_name\_\_=="\_\_main\_\_":

    main()

from picamera.array import PiRGBArray

from picamera import PiCamera

import numpy as np

import argparse

import os

import argparse

import cv2

import numpy as np

import sys

import glob

import importlib.util

import time

import RPi.GPIO as GPIO

import time

import os

panServo = 27

tiltServo = 17

x=0

y=0

GPIO.setmode(GPIO.BCM)

camera = PiCamera()

camera.resolution = (1296,972)

camera.framerate = 10

rawCapture = PiRGBArray(camera, size=(camera.resolution))

cnt=0

# 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=0.3)

parser.add\_argument('--image', help='Name of the single image to perform detection on. To run detection on multiple images, use --imagedir',

default=None)

parser.add\_argument('--imagedir', help='Name of the folder containing images to perform detection on. Folder must contain only images.',

default=None)

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)

use\_TPU = args.edgetpu

# Parse input image name and directory.

IM\_NAME = args.imageS

IM\_DIR = args.imagedir

# If both an image AND a folder are specified, throw an error

if (IM\_NAME and IM\_DIR):

print('Error! Please only use the --image argument or the --imagedir argument, not both. Issue "python TFLite\_detection\_image.py -h" for help.')

sys.exit()

# If neither an image or a folder are specified, default to using 'test1.jpg' for image name

if (not IM\_NAME and not IM\_DIR):

IM\_NAME = 'b1.jpg'

# Import TensorFlow libraries

# If tflite\_runtime is 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('tflite\_runtime')

if pkg:

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()

# Define path to images and grab all image filenames

if IM\_DIR:

PATH\_TO\_IMAGES = os.path.join(CWD\_PATH,IM\_DIR)

images = glob.glob(PATH\_TO\_IMAGES + '/\*')

elif IM\_NAME:

PATH\_TO\_IMAGES = os.path.join(CWD\_PATH,IM\_NAME)

images = glob.glob(PATH\_TO\_IMAGES)

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

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

# Path to label map file

PATH\_TO\_LABELS = os.path.join(CWD\_PATH,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])

# 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)

409

input\_mean = 127.5

input\_std = 127.5

# image\_path="/home/sajid/Pictures/Webcam/2020-07-23-171414.jpg"

# print(image\_path)

# print(images)

# Loop over every image and perform detection

# video = cv2.VideoCapture(0)

#ret = video.set(3,1280)

print(width)

print(height)

# rawCapture.truncate(0)

a=1

global chck

chck=0

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

image = frame.array

# image =cv2.imread('20191109\_142048.jpg')

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

imH, imW, \_ = image.shape

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

input\_data = np.expand\_dims(image\_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)))

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

# Draw label

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

label = '%s: %d%%' % (object\_name, int(scores[i]\*100)) # Example: 'person: 72%'

labelSize, baseLine = cv2.getTextSize(label, cv2.FONT\_HERSHEY\_SIMPLEX, 0.2, 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(image, (xmin, label\_ymin-labelSize[1]-10), (xmin+labelSize[0], label\_ymin+baseLine-10), (255, 255, 255), cv2.FILLED)

# cv2.putText(image, label, (xmin, label\_ymin), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 255, 0), 2) # Draw label text

print(object\_name)

if(object\_name=='1' or object\_name=='2'or object\_name=='3'or object\_name=='4'):

chck=1

cv2.rectangle(image, (xmin, label\_ymin-labelSize[1]-10), (xmin+labelSize[0], label\_ymin+baseLine-10), (255, 255, 255), cv2.FILLED)

cv2.putText(image, label, (xmin, label\_ymin), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 255, 0), 2) # Draw label text

object\_name=''

if(object\_name=='cotton' and chck==1):

x\_center=(xmax+xmin)/2

y\_center=(ymax+ymin)/2

x\_angle=100-(70/1296)\*int(x\_center)

y\_angle=80-(60/975)\*int(y\_center)

#

print(x\_center)

print(y\_center)

chck=0

object\_name=''

if((x\_center>650 and x\_center<850) and (y\_center>400 and y\_center<500)):

print('i am in center ')

#

else:

mapServoPosition (x\_center, y\_center)

cv2.rectangle(image, (xmin, label\_ymin-labelSize[1]-10), (xmin+labelSize[0], label\_ymin+baseLine-10), (255, 255, 255), cv2.FILLED)

cv2.putText(image, label, (xmin, label\_ymin), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 255, 0), 2) # Draw label text

# All the results have been drawn on the image, now display the image

cv2.imshow('Object detector', image)

cv2.imwrite('c2.png',image)

print("Total Number of Student Found: ",cnt)

# Press any key to continue to next image, or press 'q' to quit

if cv2.waitKey(1) & 0xFF == ord('q'):

cv2.destroyAllWindows()

break

rawCapture.truncate(0)

# Clean up

cv2.destroyAllWindows()