|  |
| --- |
| # Copyright 2017 The TensorFlow Authors. All Rights Reserved. |
|  |  | # |
|  |  | # Licensed under the Apache License, Version 2.0 (the "License"); |
|  |  | # you may not use this file except in compliance with the License. |
|  |  | # You may obtain a copy of the License at |
|  |  | # |
|  |  | # http://www.apache.org/licenses/LICENSE-2.0 |
|  |  | # |
|  |  | # Unless required by applicable law or agreed to in writing, software |
|  |  | # distributed under the License is distributed on an "AS IS" BASIS, |
|  |  | # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. |
|  |  | # See the License for the specific language governing permissions and |
|  |  | # limitations under the License. |
|  |  | # ============================================================================== |
|  |  |  |
|  |  | from \_\_future\_\_ import absolute\_import |
|  |  | from \_\_future\_\_ import division |
|  |  | from \_\_future\_\_ import print\_function |
|  |  |  |
|  |  | import argparse |
|  |  | import sys |
|  |  | import time |
|  |  |  |
|  |  | import numpy as np |
|  |  | import tensorflow as tf |
|  |  |  |
|  |  | from led import led |
|  |  | from buzzer import buzzer |
|  |  | import readWrite |
|  |  | from alertEmail import email |
|  |  |  |
|  |  | def load\_graph(model\_file): |
|  |  | graph = tf.Graph() |
|  |  | graph\_def = tf.GraphDef() |
|  |  |  |
|  |  | with open(model\_file, "rb") as f: |
|  |  | graph\_def.ParseFromString(f.read()) |
|  |  | with graph.as\_default(): |
|  |  | tf.import\_graph\_def(graph\_def) |
|  |  |  |
|  |  | return graph |
|  |  |  |
|  |  | def read\_tensor\_from\_image\_file(file\_name, input\_height=299, input\_width=299, |
|  |  | input\_mean=0, input\_std=255): |
|  |  | input\_name = "file\_reader" |
|  |  | output\_name = "normalized" |
|  |  | file\_reader = tf.read\_file(file\_name, input\_name) |
|  |  | if file\_name.endswith(".png"): |
|  |  | image\_reader = tf.image.decode\_png(file\_reader, channels = 3, |
|  |  | name='png\_reader') |
|  |  | elif file\_name.endswith(".gif"): |
|  |  | image\_reader = tf.squeeze(tf.image.decode\_gif(file\_reader, |
|  |  | name='gif\_reader')) |
|  |  | elif file\_name.endswith(".bmp"): |
|  |  | image\_reader = tf.image.decode\_bmp(file\_reader, name='bmp\_reader') |
|  |  | else: |
|  |  | image\_reader = tf.image.decode\_jpeg(file\_reader, channels = 3, |
|  |  | name='jpeg\_reader') |
|  |  | float\_caster = tf.cast(image\_reader, tf.float32) |
|  |  | dims\_expander = tf.expand\_dims(float\_caster, 0); |
|  |  | resized = tf.image.resize\_bilinear(dims\_expander, [input\_height, input\_width]) |
|  |  | normalized = tf.divide(tf.subtract(resized, [input\_mean]), [input\_std]) |
|  |  | sess = tf.Session() |
|  |  | result = sess.run(normalized) |
|  |  |  |
|  |  | return result |
|  |  |  |
|  |  | def load\_labels(label\_file): |
|  |  | label = [] |
|  |  | proto\_as\_ascii\_lines = tf.gfile.GFile(label\_file).readlines() |
|  |  | for l in proto\_as\_ascii\_lines: |
|  |  | label.append(l.rstrip()) |
|  |  | return label |
|  |  |  |
|  |  | if \_\_name\_\_ == "\_\_main\_\_": |
|  |  | file\_name = "tf\_files/flower\_photos/Shantanu/i\_1.jpeg" |
|  |  | model\_file = "tf\_files/retrained\_graph.pb" |
|  |  | label\_file = "tf\_files/retrained\_labels.txt" |
|  |  | input\_height = 224 |
|  |  | input\_width = 224 |
|  |  | input\_mean = 128 |
|  |  | input\_std = 128 |
|  |  | input\_layer = "input" |
|  |  | output\_layer = "final\_result" |
|  |  |  |
|  |  | parser = argparse.ArgumentParser() |
|  |  | parser.add\_argument("--image", help="image to be processed") |
|  |  | parser.add\_argument("--graph", help="graph/model to be executed") |
|  |  | parser.add\_argument("--labels", help="name of file containing labels") |
|  |  | parser.add\_argument("--input\_height", type=int, help="input height") |
|  |  | parser.add\_argument("--input\_width", type=int, help="input width") |
|  |  | parser.add\_argument("--input\_mean", type=int, help="input mean") |
|  |  | parser.add\_argument("--input\_std", type=int, help="input std") |
|  |  | parser.add\_argument("--input\_layer", help="name of input layer") |
|  |  | parser.add\_argument("--output\_layer", help="name of output layer") |
|  |  | args = parser.parse\_args() |
|  |  |  |
|  |  | if args.graph: |
|  |  | model\_file = args.graph |
|  |  | if args.image: |
|  |  | file\_name = args.image |
|  |  | if args.labels: |
|  |  | label\_file = args.labels |
|  |  | if args.input\_height: |
|  |  | input\_height = args.input\_height |
|  |  | if args.input\_width: |
|  |  | input\_width = args.input\_width |
|  |  | if args.input\_mean: |
|  |  | input\_mean = args.input\_mean |
|  |  | if args.input\_std: |
|  |  | input\_std = args.input\_std |
|  |  | if args.input\_layer: |
|  |  | input\_layer = args.input\_layer |
|  |  | if args.output\_layer: |
|  |  | output\_layer = args.output\_layer |
|  |  |  |
|  |  | graph = load\_graph(model\_file) |
|  |  | t = read\_tensor\_from\_image\_file(file\_name, |
|  |  | input\_height=input\_height, |
|  |  | input\_width=input\_width, |
|  |  | input\_mean=input\_mean, |
|  |  | input\_std=input\_std) |
|  |  |  |
|  |  | input\_name = "import/" + input\_layer |
|  |  | output\_name = "import/" + output\_layer |
|  |  | input\_operation = graph.get\_operation\_by\_name(input\_name); |
|  |  | output\_operation = graph.get\_operation\_by\_name(output\_name); |
|  |  |  |
|  |  | with tf.Session(graph=graph) as sess: |
|  |  | start = time.time() |
|  |  | results = sess.run(output\_operation.outputs[0], |
|  |  | {input\_operation.outputs[0]: t}) |
|  |  | end=time.time() |
|  |  | results = np.squeeze(results) |
|  |  |  |
|  |  | top\_k = results.argsort()[-5:][::-1] |
|  |  | labels = load\_labels(label\_file) |
|  |  |  |
|  |  | print('\nEvaluation time (1-image): {:.3f}s\n'.format(end-start)) |
|  |  | template = "{} (score={:0.5f})" |
|  |  |  |
|  |  | led = led() |
|  |  | buzzer = buzzer() |
|  |  | email = email() |
|  |  |  |
|  |  | for i in top\_k: |
|  |  | #print(template.format(labels[i], results[i])) |
|  |  | if (labels[i] =="shantanu" and results[i]>=0.9): |
|  |  | print("Match Found For Shantanu") |
|  |  | led.run(13,1) |
|  |  | buzzer.run(36,0.5) |
|  |  | readWrite.csvWrite("babyMovents.csv",1,0,0,0,0) |
|  |  | break |
|  |  | elif (labels[i] =="lion" and results[i]>=0.9): |
|  |  | print("Match Found For Lion") |
|  |  | readWrite.csvWrite("babyMovents.csv",0,1,0,0,0) |
|  |  | break |
|  |  | elif (labels[i] =="baby" and results[i]>=0.9): |
|  |  | print("Match Found For Baby") |
|  |  | readWrite.csvWrite("babyMovents.csv",0,0,1,0,0) |
|  |  | break |
|  |  | elif (labels[i] =="tiger" and results[i]>=0.9): |
|  |  | print("Match Found For Person") |
|  |  | readWrite.csvWrite("babyMovents.csv",0,0,0,1,0) |
|  |  | break |
|  |  | else: |
|  |  | print("Baby Is Not Sleeping") |
|  |  | readWrite.csvWrite("babyMovents.csv",0,0,0,0,1) |
|  |  | email.emailSend("baby.png","/home/pi/Desktop/project/SC/tf\_files/flower\_photos/pro.png",'temp','life','fire') |
|  |  | break |