WorldsOkayestAirman Face Recognizer create function updated. eb10096 on Oct 20, 2017

2 contributors

344 lines (256 sloc) 16.2 KB

```
1
2  # coding: utf-8
3
4  # Face Recognition with OpenCV
5
6  # To detect faces, I will use the code from my previous article on [face detection](https://www.superdatascience.com/opencv-face-detection/
7
8  # ### Import Required Modules
9
10 # Before starting the actual coding we need to import the required modules for coding. So let's import them first.
11 #
12 # - **cv2:** is _OpenCV_ module for Python which we will use for face detection and face recognition.
13 # - **os:** We will use this Python module to read our training directories and file names.
14 # - **numpy:** We will use this module to convert Python lists to numpy arrays as OpenCV face recognizers accept numpy arrays.
15
16 # In[1]:
17
18 #import OpenCV module
19 import cv2
20 #import os module for reading training data directories and paths
21 import os
22 #import numpy to convert python lists to numpy arrays as
23 #it is needed by OpenCV face recognizers
```

```
import numpy as np


# ### Training Data

# The more images used in training the better. Normally a lot of images are used for training a face recognizer so that it can learn differ
#
# So our training data consists of total 2 persons with 12 images of each person. All training data is inside _`training-data`_ folder. _`t
#
# ```
# training-data
# |-------------- s1
# |               |-- 1.jpg
# |               |-- ...
# |               |-- 12.jpg
# |-------------- s2
# |               |-- 1.jpg
# |               |-- ...
# |               |-- 12.jpg
# ```
#
# The _`test-data`_ folder contains images that we will use to test our face recognizer after it has been successfully trained.

# As OpenCV face recognizer accepts labels as integers so we need to define a mapping between integer labels and persons actual names so be
#
# **Note:** As we have not assigned `label 0` to any person so **the mapping for label 0 is empty**.

# In[2]:

#there is no label 0 in our training data so subject name for index/label 0 is empty
subjects = ["", "Ramiz Raja", "Elvis Presley"]


# ### Prepare training data

```

```
# You may be wondering why data preparation, right? Well, OpenCV face recognizer accepts data in a specific format. It accepts two vectors,
#
# For example, if we had 2 persons and 2 images for each person.
#
# ```
# PERSON-1    PERSON-2
#
# img1        img1
# img2        img2
# ```
#
# Then the prepare data step will produce following face and label vectors.
#
# ```
# FACES                        LABELS
#
# person1_img1_face              1
# person1_img2_face              1
# person2_img1_face              2
# person2_img2_face              2
# ```
#
#
# Preparing data step can be further divided into following sub-steps.
#
# 1. Read all the folder names of subjects/persons provided in training data folder. So for example, in this tutorial we have folder names:
# 2. For each subject, extract label number. **Do you remember that our folders have a special naming convention?** Folder names follow the
# 3. Read all the images of the subject, detect face from each image.
# 4. Add each face to faces vector with corresponding subject label (extracted in above step) added to labels vector.
#
# **[There should be a visualization for above steps here]**

# Did you read my last article on [face detection](https://www.superdatascience.com/opencv-face-detection/)? No? Then you better do so righ

# In[3]:
```

```
94
95  #function to detect face using OpenCV
96  def detect_face(img):
97      #convert the test image to gray image as opencv face detector expects gray images
98      gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
99
100     #load OpenCV face detector, I am using LBP which is fast
101     #there is also a more accurate but slow Haar classifier
102     face_cascade = cv2.CascadeClassifier('opencv-files/lbpcascade_frontalface.xml')
103
104     #let's detect multiscale (some images may be closer to camera than others) images
105     #result is a list of faces
106     faces = face_cascade.detectMultiScale(gray, scaleFactor=1.2, minNeighbors=5);
107
108     #if no faces are detected then return original img
109     if (len(faces) == 0):
110         return None, None
111
112     #under the assumption that there will be only one face,
113     #extract the face area
114     (x, y, w, h) = faces[0]
115
116     #return only the face part of the image
117     return gray[y:y+w, x:x+h], faces[0]
118
119
120 # I am using OpenCV's **LBP face detector**. On _line 4_, I convert the image to grayscale because most operations in OpenCV are performed
121 #
122 # Now you have got a face detector and you know the 4 steps to prepare the data, so are you ready to code the prepare data step? Yes? So le
123
124 # In[4]:
125
126 #this function will read all persons' training images, detect face from each image
127 #and will return two lists of exactly same size, one list
128 # of faces and another list of labels for each face
```

```
129 def prepare_training_data(data_folder_path):
130 
131     #------STEP-1--------
132     #get the directories (one directory for each subject) in data folder
133     dirs = os.listdir(data_folder_path)
134 
135     #list to hold all subject faces
136     faces = []
137     #list to hold labels for all subjects
138     labels = []
139 
140     #let's go through each directory and read images within it
141     for dir_name in dirs:
142 
143         #our subject directories start with letter 's' so
144         #ignore any non-relevant directories if any
145         if not dir_name.startswith("s"):
146             continue;
147 
148         #------STEP-2--------
149         #extract label number of subject from dir_name
150         #format of dir name = slabel
151         #, so removing letter 's' from dir_name will give us label
152         label = int(dir_name.replace("s", ""))
153 
154         #build path of directory containin images for current subject subject
155         #sample subject_dir_path = "training-data/s1"
156         subject_dir_path = data_folder_path + "/" + dir_name
157 
158         #get the images names that are inside the given subject directory
159         subject_images_names = os.listdir(subject_dir_path)
160 
161         #------STEP-3--------
162         #go through each image name, read image,
163         #detect face and add face to list of faces
```

```
        for image_name in subject_images_names:

            #ignore system files like .DS_Store
            if image_name.startswith("."):
                continue;

            #build image path
            #sample image path = training-data/s1/1.pgm
            image_path = subject_dir_path + "/" + image_name

            #read image
            image = cv2.imread(image_path)

            #display an image window to show the image
            cv2.imshow("Training on image...", cv2.resize(image, (400, 500)))
            cv2.waitKey(100)

            #detect face
            face, rect = detect_face(image)

            #------STEP-4--------
            #for the purpose of this tutorial
            #we will ignore faces that are not detected
            if face is not None:
                #add face to list of faces
                faces.append(face)
                #add label for this face
                labels.append(label)

    cv2.destroyAllWindows()
    cv2.waitKey(1)
    cv2.destroyAllWindows()

    return faces, labels

```

```
199
200 # I have defined a function that takes the path, where training subjects' folders are stored, as parameter. This function follows the same
201 #
202 # **(step-1)** On _line 8_ I am using `os.listdir` method to read names of all folders stored on path passed to function as parameter. On _
203 #
204 # **(step-2)** After that I traverse through all subjects' folder names and from each subject's folder name on _line 27_ I am extracting th
205 #
206 # **(step-3)** On _line 34_, I read all the images names of of the current subject being traversed and on _line 39-66_ I traverse those ima
207 #
208 # **(step-4)** On _line 62-66_, I add the detected face and label to their respective vectors.
209
210 # But a function can't do anything unless we call it on some data that it has to prepare, right? Don't worry, I have got data of two beauti
211 #
212 # ![training-data](visualization/tom-shahrukh.png)
213 #
214 # Let's call this function on images of these beautiful celebrities to prepare data for training of our Face Recognizer. Below is a simple
215
216 # In[5]:
217
218 #let's first prepare our training data
219 #data will be in two lists of same size
220 #one list will contain all the faces
221 #and other list will contain respective labels for each face
222 print("Preparing data...")
223 faces, labels = prepare_training_data("training-data")
224 print("Data prepared")
225
226 #print total faces and labels
227 print("Total faces: ", len(faces))
228 print("Total labels: ", len(labels))
229
230
231 # This was probably the boring part, right? Don't worry, the fun stuff is coming up next. It's time to train our own face recognizer so tha
232
233 # ### Train Face Recognizer
```

```
234
235 # As we know, OpenCV comes equipped with three face recognizers.
236 #
237 # 1. EigenFace Recognizer: This can be created with `cv2.face.createEigenFaceRecognizer()`
238 # 2. FisherFace Recognizer: This can be created with `cv2.face.createFisherFaceRecognizer()`
239 # 3. Local Binary Patterns Histogram (LBPH): This can be created with `cv2.face.LBPHFisherFaceRecognizer()`
240 #
241 # I am going to use LBPH face recognizer but you can use any face recognizer of your choice. No matter which of the OpenCV's face recognize
242
243 # In[6]:
244
245 #create our LBPH face recognizer
246 face_recognizer = cv2.face.LBPHFaceRecognizer_create()
247
248 #or use EigenFaceRecognizer by replacing above line with
249 #face_recognizer = cv2.face.EigenFaceRecognizer_create()
250
251 #or use FisherFaceRecognizer by replacing above line with
252 #face_recognizer = cv2.face.FisherFaceRecognizer_create()
253
254
255 # Now that we have initialized our face recognizer and we also have prepared our training data, it's time to train the face recognizer. We
256
257 # In[7]:
258
259 #train our face recognizer of our training faces
260 face_recognizer.train(faces, np.array(labels))
261
262
263 # **Did you notice** that instead of passing `labels` vector directly to face recognizer I am first converting it to **numpy** array? This
264 #
265 # Still not satisfied? Want to see some action? Next step is the real action, I promise!
266
267 # ### Prediction
268
```

```
# Now comes my favorite part, the prediction part. This is where we actually get to see if our algorithm is actually recognizing our traine
#
# Below are some utility functions that we will use for drawing bounding box (rectangle) around face and putting celeberity name near the f

# In[8]:

#function to draw rectangle on image
#according to given (x, y) coordinates and
#given width and heigh
def draw_rectangle(img, rect):
    (x, y, w, h) = rect
    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)

#function to draw text on give image starting from
#passed (x, y) coordinates.
def draw_text(img, text, x, y):
    cv2.putText(img, text, (x, y), cv2.FONT_HERSHEY_PLAIN, 1.5, (0, 255, 0), 2)


# First function `draw_rectangle` draws a rectangle on image based on passed rectangle coordinates. It uses OpenCV's built in function `cv2
#
# Second function `draw_text` uses OpenCV's built in function `cv2.putText(img, text, startPoint, font, fontSize, rgbColor, lineWidth)` to
#
# Now that we have the drawing functions, we just need to call the face recognizer's `predict(face)` method to test our face recognizer on

# In[9]:

#this function recognizes the person in image passed
#and draws a rectangle around detected face with name of the
#subject
def predict(test_img):
    #make a copy of the image as we don't want to chang original image
    img = test_img.copy()
    #detect face from the image
    face, rect = detect_face(img)
```

```
304
305     #predict the image using our face recognizer
306     label, confidence = face_recognizer.predict(face)
307     #get name of respective label returned by face recognizer
308     label_text = subjects[label]
309
310     #draw a rectangle around face detected
311     draw_rectangle(img, rect)
312     #draw name of predicted person
313     draw_text(img, label_text, rect[0], rect[1]-5)
314
315     return img
316
317 # Now that we have the prediction function well defined, next step is to actually call this function on our test images and display those t
318
319 # In[10]:
320
321 print("Predicting images...")
322
323 #load test images
324 test_img1 = cv2.imread("test-data/test1.jpg")
325 test_img2 = cv2.imread("test-data/test2.jpg")
326
327 #perform a prediction
328 predicted_img1 = predict(test_img1)
329 predicted_img2 = predict(test_img2)
330 print("Prediction complete")
331
332 #display both images
333 cv2.imshow(subjects[1], cv2.resize(predicted_img1, (400, 500)))
334 cv2.imshow(subjects[2], cv2.resize(predicted_img2, (400, 500)))
335 cv2.waitKey(0)
336 cv2.destroyAllWindows()
337 cv2.waitKey(1)
338 cv2.destroyAllWindows()
```

```
339
340
341
342
343
```