DWDM PROJECT

FACIAL RECOGNITION USING PYTHON AND OPEN CV2

SEMESTER – 6
IT B

SHUBHAM SORTE
130911228

SABARISH NARAYANAN
130911190

ABSTRACT

The main aim of the project is to develop a system which can be trained to recognize a face with any facial expression. This means that the person can be smiling, winking, surprised, etc. while the facial recognition software is working on the his/her face.

The project uses Python 3 as source language and Open CV 2 for the facial recognition.

The images of the person whose face needs to be recognized should be in the database for training purposes.

PROBLEM STATEMENT

To develop a system for facial recognition for any expression of the person. The system should be able to recognize faces with any expression possible and it should do so with the help of the model which is pre-trained with images of the person concerned.

OBJECTIVES

- To develop a system which can train a data model to detect different features of a face of a person.
- To create different clusters of same person face with different facial expressions using Local Binary Pattern Histograms
- To use data from these clusters to identify the face of the person when input next time.
- The program shows confidence of how much it is sure of the person’s face.

LITERATURE SURVEYS

- Face Detection and Tracking using OpenCV by S.V. Viraktamath , Mukund Katti, Aditya Khatawkar & Pavan Kulkarni.
  Abstract - An application for automatic face detection and tracking on video streams from surveillance cameras in public or commercial places is discussed in this paper. In many situations it is useful to detect where the people are looking for, e.g. in exhibits, commercial malls, and public places in buildings. This system can be used for security purpose to record the visitor face as well as to detect and track the face. A program is developed using OpenCV that can detect people's face and also track from the web camera.

- Facial Recognition using OpenCV by Shervin EMAMI , Valentin Petruț SUCIU

  Abstract - Because of general curiosity and interest in the matter, the author has proposed to create an application that would allow user access to a particular machine based on an in-depth analysis of a person’s facial features. This application will be developed using Intel’s open source computer vision project, OpenCV and Microsoft’s .NET framework.

METHODOLOGY

The whole process can be divided in three major steps -

1. The first step is to find a good database of faces with multiple images for each individual.
2. The next step is to detect faces in the database images and use them to train the face recognizer.
3. The last step is to test the face recognizer to recognize faces it was trained for.

The actual code is less than 40 lines of python code, thanks to the terse syntax of python.

IMPLEMENTATION

DATABASE

We will use the Yale Face Database that contains 165 grayscale images of 15 individuals in gif format, There are 11 images for each individual. In each image, the individual has a different facial expression like happy, sad, normal, surprised, sleepy etc. (See Appendix for some example images)

We will use this database by using 10 images of the total 11 images of each individual in training our face recognizer and the remaining single image of each individual to test our face recognition algorithm.

The images corresponding to each individual are named like subject<number>.<facial_expression> where number ranges from 01, 02, 03…, 14, 15 and facial_expression is the expression that the individual has in the image. For individual numbered 01 -

subject01.gif
subject01.glasses
subject01.glasses.gif
subject01.happy
subject01.leftlight
subject01.noglasses
subject01.normal
subject01.rightlight
subject01.sad
subject01.sleepy
subject01.surprised

subject01.wink

We will not train the image with the .sad extension. We will use these images to test the face recognizer.

IMPLEMENTATION

REQUIRED MODULES -

1. cv2 - This is the OpenCV module and contains the functions for face detection and recognition.
2. os - This module will be used to maneuver with image and directory names. First, we will use this module to extract the image names in the database directory and then from these names we will extract the individual number, which will be used as a label for the face in that image.
3. Image - Since, the dataset images are in gif format and as of now, OpenCV does not support gif format, we will use Image module from PIL to read the image in grayscale format.
4. numpy - Our images will be stored in numpy arrays.

LOADING THE FACE DETECTION CASCADE

For the purpose of face detection, we use the Haar Cascade provided by OpenCV. The haar cascades that come with OpenCV are located in the /data/haarcascades> directory of your OpenCV installation. We use haarcascade_frontalface_default.xml for detecting the face. So, we load the cascade using the cv2.CascadeClassifier function which takes the path to the cascade xml file.

CREATE THE FACE RECOGNIZER OBJECT

The next step is creating the face recognizer object. The face recognizer object has functions like FaceRecognizer.train to train the recognizer and FaceRecognizer.predict to recognize a face.

For the project we used Local Binary Patterns Histograms Face Recognizer. We used LBPH since they are easy to understand when graphically represented.

CREATE THE FUNCTION TO PREPARE THE TRAINING SET

We defined a function get_images_and_labels that takes the absolute path to the image database as input argument and returns tuple of 2 list, one containing the detected faces and the other containing the corresponding label for that face.

We load the current image in a 2D numpy array image. We cannot read the images directly using cv2 because as of now, OpenCV doesn't support gif format images and unfortunately, our database images are in this format. So, we use the Image module from PIL to read the images in grayscale format and convert them into numpy arrays which are compatible with OpenCV.

From the image name, we extract the individual number. This number will be the label for that face. We use CascadeClassifier.detectMultiScale to detect faces in the image. For each face it returns a rectangle in the format (Top-Left x pixel value, Top-Left y pixel value, Width of rectangle, Height of rectangle.).We slice the ROI from the image and append it to the list images and the corresponding label in the list labels. Once, we are done with this loop, we return the 2 lists in the form of a tuple.

PERFORM THE TRAINING

We perform the training using the FaceRecognizer.train function. It requires 2 arguments, the features which in this case are the images of faces and the corresponding labels assigned to these faces which in this case are the individual number that we extracted from the image names.

TESTING THE FACE RECOGNIZER

We will test the results of the recognizer by using the images with .sad extension which we had not used earlier. As done in the get_images_and_labels function, we append all the image names with the .sad extension in a image_paths list. Then for each image in the list, we read it in grayscale format and detect faces in it. We pass this to the FaceRecognizer.predict function which will assign it a label and it will also tell us how confident it is about the recognition. The label is an integer that is one of the individual numbers we had assigned to the faces earler. This label is stored in nbr_predicted. The more the value of confidence variable is, the less the recognizer has confidence in the recognition.
We check if the recognition was correct by comparing the predicted label nbr_predicted with the actual label nbr_actual. The label nbr_actual is extracted using the os

module and the string operations from the name of the image. We also display the confidence score for each recognition.

A confidence value of 0.0 is a perfect recognition.

(Please have a look at the appendix for the screenshots of the terminal with the program working)

CONCLUSION

We performed the task of face recognition using OpenCV in less than 40 lines of python codes. The code assigns a label to each image that is to recognized. But, what if the face to be recognized is not even in the database. In that case, the confidence score comes to our rescue. When, the face is not known by the face recognizer, the value of confidence score will be very high and you can use a threshold to ascertain that the face was not recognized.

REFERENCES

- S.V. Viraktamath, Mukund Katti, Aditya Khatawkar & Pavan Kulkarni (2013),"Face Detection and Tracking using OpenCV,SIJ",SDM College of Engineering & Technology, Dharwad, Karnataka,INDIA, Vol. 1, No. 3
- Shervin Emami (2010),"Introduction to Face Detection and Face Recognition",Article,http://shervinemami.info/faceRecognition.html
- Bikramjot Singh Hanzra (2015),"Face Recognition using Python and OpenCV", Article, http://hanzratech.in/2015/02/03/face-recognition-using-opencv.html

FUTURE WORK

This project covers the very basics of facial recognition. There is a lot of potential on this subject and many applications are developed everyday which uses facial recognition in one way or another.

This project can be taken forward by some of the following work -

- A live training model , where a camera can be used to take live pictures and train the model, instead of sending an array of images. This will be more practical since sending images to the program is a tedious job.
- Porting the program to work on a mobile application. Everything is going mobile and if we could port this to mobile, there are lot of applications possible. For eg. - face to unlock phone, pay online authenticating with your face, different phone modes depending on who is using the phone, etc.

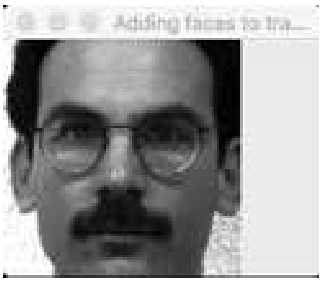
APPENDIX

CODE

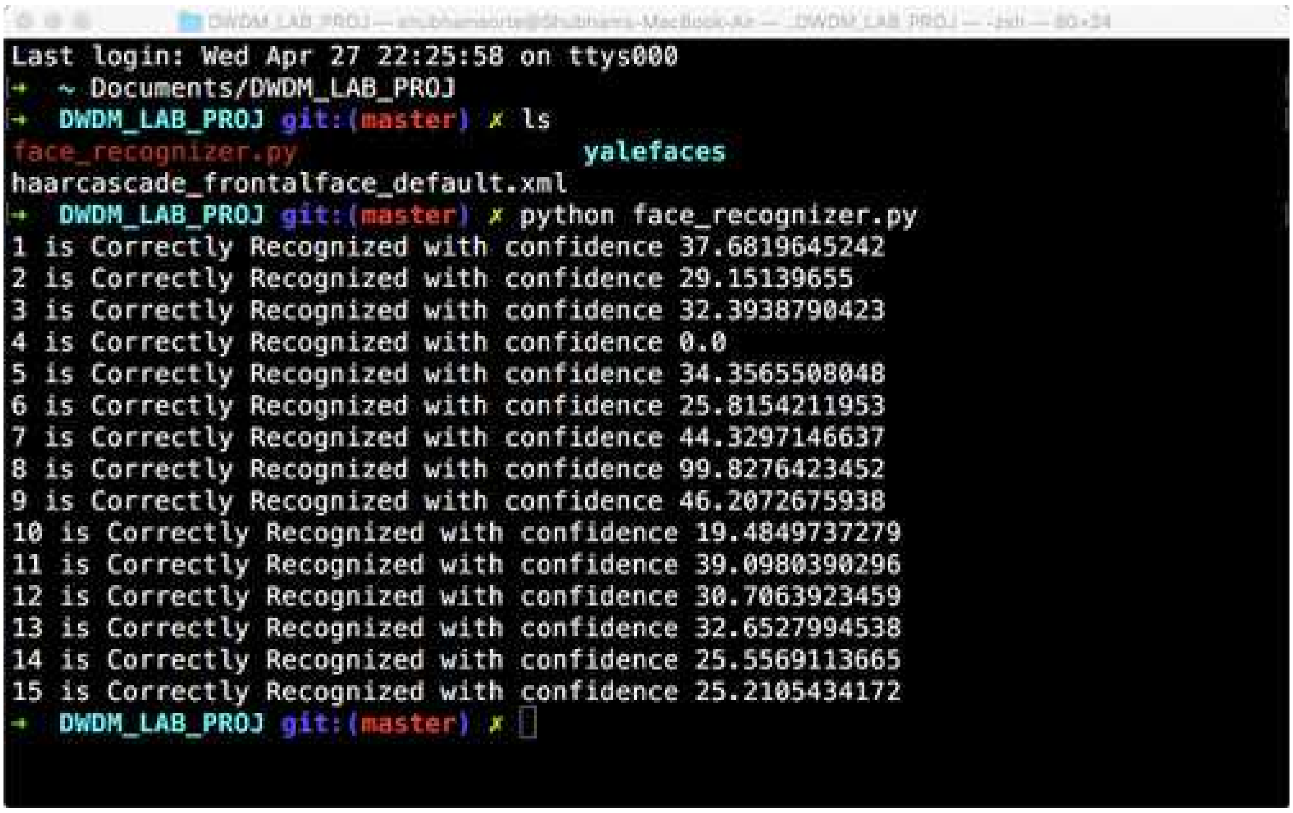
```
#!/usr/bin/python

# Import the required modules
import cv2, os
import numpy as np
from PIL import Image

# For face detection we will use the Haar Cascade provided by OpenCV.
cascadePath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath)

# For face recognition we will the the LBPH Face Recognizer
recognizer = cv2.createLBPHFaceRecognizer()

def get_images_and_labels(path):
    # Append all the absolute image paths in a list image_paths
    # We will not read the image with the .sad extension in the training set
    # Rather, we will use them to test our accuracy of the training
    image_paths = [os.path.join(path, f) for f in os.listdir(path) if not f.endswith('.sad')]
    # images will contains face images
    images = []
    # labels will contains the label that is assigned to the image
    labels = []
    for image_path in image_paths:
        # Read the image and convert to grayscale
        image_pil = Image.open(image_path).convert('L')
        # Convert the image format into numpy array
        image = np.array(image_pil, 'uint8')
        # Get the label of the image
        nbr = int(os.path.split(image_path)[1].split(".")[0].replace("subject", ""))
        # Detect the face in the image
        faces = faceCascade.detectMultiScale(image)
        # If face is detected, append the face to images and the label to labels
        for (x, y, w, h) in faces:
            images.append(image[y: y + h, x: x + w])
            labels.append(nbr)
            cv2.imshow("Adding faces to traning set...", image[y: y + h, x: x + w])
            cv2.waitKey(50)
    # return the images list and labels list
    return images, labels

# Path to the Yale Dataset
path = './yalefaces'
# Call the get_images_and_labels function and get the face images and the
# corresponding labels
images, labels = get_images_and_labels(path)
cv2.destroyAllWindows()

# Perform the tranining
recognizer.train(images, np.array(labels))


image_paths = [os.path.join(path, f) for f in os.listdir(path) if f.endswith('.sad')]
for image_path in image_paths:
    predict_image_pil = Image.open(image_path).convert('L')
    predict_image = np.array(predict_image_pil, 'uint8')
    faces = faceCascade.detectMultiScale(predict_image)
    for (x, y, w, h) in faces:
        nbr_predicted, conf = recognizer.predict(predict_image[y: y + h, x: x + w])
        nbr_actual = int(os.path.split(image_path)[1].split(".")[0].replace("subject", ""))
        if nbr_actual == nbr_predicted:
            print "{} is Correctly Recognized with confidence
{}".format(nbr_actual,conf)
        else:
            print "{} is Incorrect Recognized as {}".format(nbr_actual, nbr_predicted)
        cv2.imshow("Recognizing Face", predict_image[y: y + h, x: x + w])
        cv2.waitKey(1000)
```

IMAGE SET EXAMPLE IMAGES

TRAINING WINDOW



TERMINAL WITH RESULT



```
Last login: Wed Apr 27 22:25:58 on ttys000
→  ~ Documents/DWDM_LAB_PROJ
→  DWDM_LAB_PROJ git:(master) ✗ ls
face_recognizer.py                    yalefaces
haarcascade_frontalface_default.xml
→  DWDM_LAB_PROJ git:(master) ✗ python face_recognizer.py
1 is Correctly Recognized with confidence 37.6819645242
2 is Correctly Recognized with confidence 29.15139655
3 is Correctly Recognized with confidence 32.3938790423
4 is Correctly Recognized with confidence 0.0
5 is Correctly Recognized with confidence 34.3565508048
6 is Correctly Recognized with confidence 25.8154211953
7 is Correctly Recognized with confidence 44.3297146637
8 is Correctly Recognized with confidence 99.8276423452
9 is Correctly Recognized with confidence 46.2072675938
10 is Correctly Recognized with confidence 19.4849737279
11 is Correctly Recognized with confidence 39.0980390296
12 is Correctly Recognized with confidence 30.7063923459
13 is Correctly Recognized with confidence 32.6527994538
14 is Correctly Recognized with confidence 25.5569113665
15 is Correctly Recognized with confidence 25.2105434172
→  DWDM_LAB_PROJ git:(master) ✗ 
```

In the image above note that, individual number 4 is recognized with a perfect score because subject04.sad and subject04.normal are the same images. From the image above, we can see that our Face Recognizer was able to recognize all the faces correctly.