COMP4026 Computer Vision and Pattern Recognition

Lab 6: Face Recognition with Eigenface

After completion of this lab, students will have a more experience in building a complete computer vision and pattern recognition prototype via implementing a classic Eigenface (PCA) method in face recognition.

Table of Contents

Prepared by Mr. R Shao, Ms. F Lyu, Prof. PC Yuen,

1. Face recognition with Eigenface

First, you need to install the opencv-contrib-python package via **pip install opencv-contrib-python.**

As a toy example, we just recognize two people with Eigenface representation. It consists of two phases, namely training and testing.

Training Phase

We use 12 images for each person to train our model. All training data is inside 'training-data' folder. One folder contains images for each person which is named with format 'sLabel (e.g. s1, s2)'. Thus label (e.g 1, 2) is the integer label assigned to that person. For example, folder named s1 means that this folder contains images for person 1. The directory structure tree for training data is as follows:

```
training-data
|-------------- s1
|               |-- 1.jpg
|               |-- ...
|               |-- 12.jpg
|-------------- s2
|               |-- 1.jpg
|               |-- ...
|               |-- 12.jpg
```

Fig. 1. Directory structure tree for training data

Name of two people:

```python
import cv2
import os
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
#
# So our training data consists of total 2 persons with 12 images of each person. All training data is inside _`training-data`_ fold
#
# ```
# 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 name
#
# **Note:** As we have not assigned `label 0` to any person so **the mapping for label 0 is empty**.

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

Prepare the dataset using **prepare_training_data** function, which detects faces and

assigns the corresponding labels accordingly.

```
#this function will read all persons' training images, detect face from each image
#and will return two lists of exactly same size, one list
# of faces and another list of labels for each face
def prepare_training_data(data_folder_path):

    #------STEP-1--------
    #get the directories (one directory for each subject) in data folder
    dirs = os.listdir(data_folder_path)

    #list to hold all subject faces
    faces = []
    #list to hold labels for all subjects
    labels = []

    #let's go through each directory and read images within it
    for dir_name in dirs:

        #our subject directories start with letter 's' so
        #ignore any non-relevant directories if any
        if not dir_name.startswith("s"):
            continue;

        #------STEP-2--------
        #extract label number of subject from dir_name
        #format of dir name = slabel
        #, so removing letter 's' from dir_name will give us label
        label = int(dir_name.replace("s", ""))

        #build path of directory containin images for current subject subject
        #sample subject_dir_path = "training-data/s1"
        subject_dir_path = data_folder_path + "/" + dir_name

        #get the images names that are inside the given subject directory
        subject_images_names = os.listdir(subject_dir_path)
```

```
        #------STEP-3--------
        #go through each image name, read image,
        #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)

            #detect face
            face, rect = detect_face(image)

            face = cv2.resize(face, (200, 300))

            #------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)


    return faces, labels
```

The **detect_face** function detects the faces using the pre-trained face detector in OpenCV.

```
#function to detect face using OpenCV
def detect_face(img):
    #convert the test image to gray image as opencv face detector expects gray images
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    #load OpenCV face detector, I am using LBP which is fast
    #there is also a more accurate but slow Haar classifier
    face_cascade = cv2.CascadeClassifier('C:/Users/ruishao/Anaconda3/Lib/site-packages/cv2/data/haarcascade_frontalface_default.xml')

    #let's detect multiscale (some images may be closer to camera than others) images
    #result is a list of faces
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.2, minNeighbors=5);

    #if no faces are detected then return original img
    if (len(faces) == 0):
        return None, None

    #under the assumption that there will be only one face,
    #extract the face area
    (x, y, w, h) = faces[0]

    #return only the face part of the image
    return gray[y:y+w, x:x+h], faces[0]
```

A face classifier is then developed based on the EigenFace representation using the function **cv2.face.EigenFaceRecognizer_create().** We train the face classifier with prepared training data as follows.

```
#let's first prepare our training data
#data will be in two lists of same size
#one list will contain all the faces
#and other list will contain respective labels for each face
print("Preparing data...")
faces, labels = prepare_training_data("../lab6 data/training-data")
print("Data prepared")

#print total faces and labels
print("Total faces: ", len(faces))
print("Total labels: ", len(labels))


face_recognizer = cv2.face.EigenFaceRecognizer_create()


#train our face recognizer of our training faces
face_recognizer.train(faces, np.array(labels))
```

We design the prediction function to predict the labels of testing face images and plot the predicted label (person name) on the testing images.

```
#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)


#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)

    face = cv2.resize(face, (200, 300))

    #predict the image using our face recognizer
    label, confidence = face_recognizer.predict(face)
    #get name of respective label returned by face recognizer
    label_text = subjects[label]

    #draw a rectangle around face detected
    draw_rectangle(img, rect)
    #draw name of predicted person
    draw_text(img, label_text, rect[0], rect[1]-5)

    return img
```

Testing Phase

Let's do the testing as follows.

```
print("Predicting images...")

#load test images
test_img1 = cv2.imread("../lab6 data/test-data/test1.jpg")
test_img2 = cv2.imread("../lab6 data/test-data/test2.jpg")

#perform a prediction
predicted_img1 = predict(test_img1)
predicted_img2 = predict(test_img2)
print("Prediction complete")

#display both images
cv2.imshow(subjects[1], cv2.resize(predicted_img1, (400, 500)))
cv2.imshow(subjects[2], cv2.resize(predicted_img2, (400, 500)))
cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.waitKey(1)
cv2.destroyAllWindows()
```

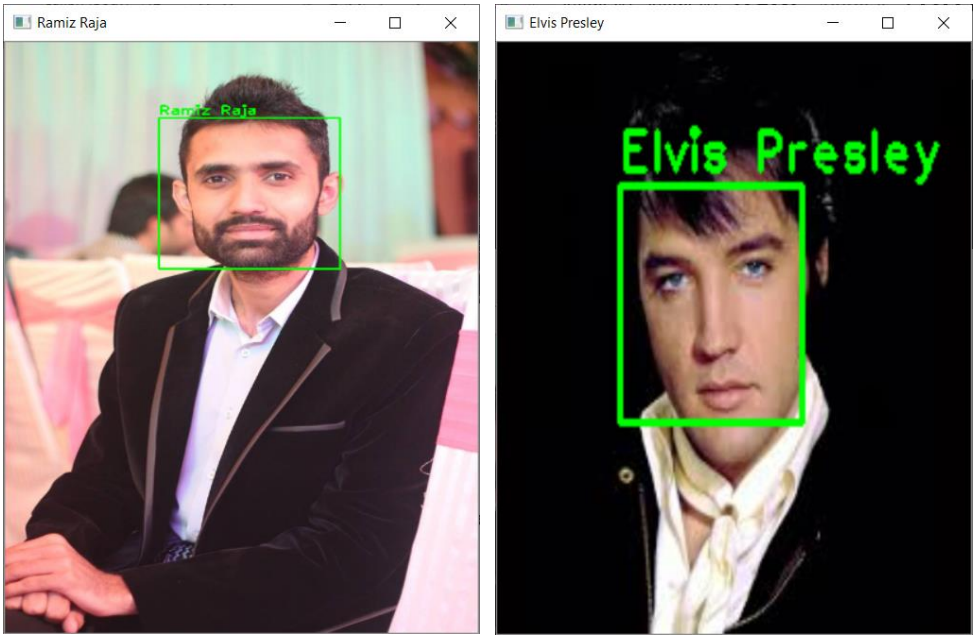

Fig.2. Predicted Labels

2. Lab Assignment 2

Train the model again with 3 individuals (classes) with different numbers of training data/individual, (10, 20 and 30) which is called training set. You may use images from any individuals. Using another 20 images of each individual outside training set for testing. That is, there is no overlapping between training and testing images. Record the results.

What to submit: You are required to submit a 2-page report to Mr. Feng PAN by email (and cc PC Yuen) on or before 1 Nov 2023. Name your file as 12345678_Lab_Assignement2.pdf where 12345678 is your student ID. The report should contain the followings, but not limited to,

1. Give a short description of your work
2. Include a link to show all images for training
3. Include a link to show all images for testing
4. Report the recognition results
5. Discussion on your results
6. Conclusion

Late Submission: 10% deduction per day.

Reference

[1] **Face Recognition with OpenCV**:
https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html
https://github.com/informramiz/opencv-face-recognition-python

[2] M A Turk and A Pentland, “Face recognition using eigenfaces”, CVPR 1991.