FACE RECOGNITION REPORT

Steps for Face detection in images:

1) Load your training images:

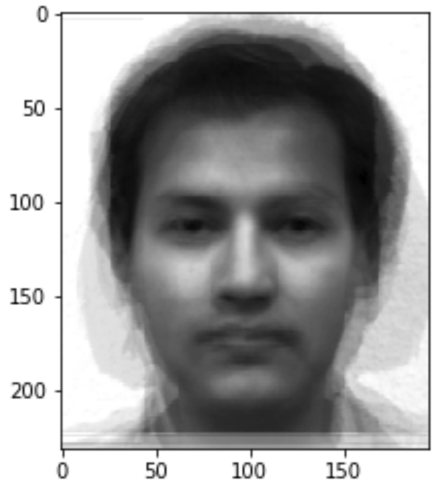
You can load training images using glob library in python using glob.glob function.
Training images are :

2) Convert your images into column vectors.

In our program we have loaded 8 training images for face recognition. For every image put your image pixels in an array and put it in a column of matrix. If you have 8 faces, you will have 8 columns. Each image size in our case is 195 x 231 (width x height) pixels. Stack each row of an image into a single column vector. Therefore in our case the size of column vector is (45,045 * 1).
The whole column matrix size is (45,045 * 1)

3) Calculate the mean.

In this step we have to add all column into one column. Thus, all 'images columns' row by row and then divide this by total number of images, in our case 8. The single column matrix is the Average matrix.



4) Reduce your matrix created in step2 using mean from step3.

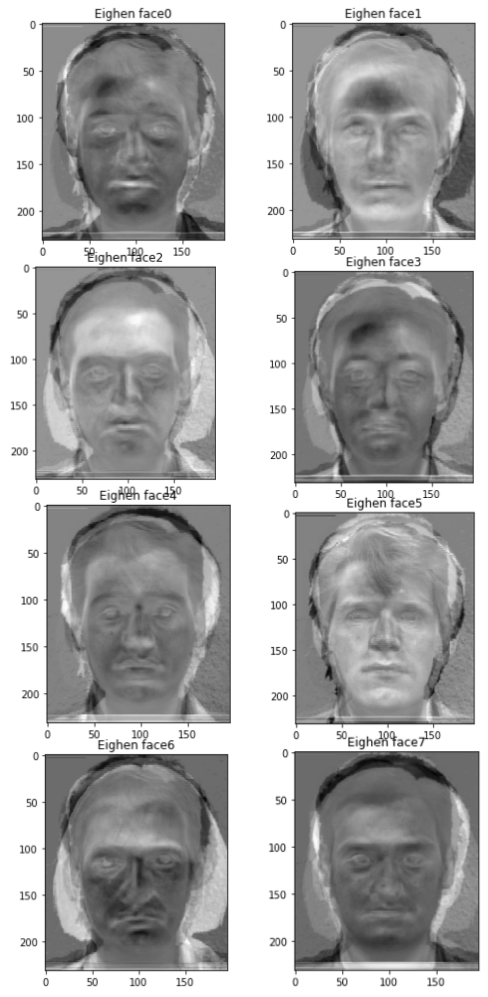
Now, subtract the Column matrix created in step 2 from Average Matrix created in step3. This matrix is saved as Processing Matrix in our case. We need to have the data centered at origin. After that, we can perform matrix rotation and scaling. Matrix rotation and scaling are performed at center of the coordinate system, thus, we need to move our dataset to the center.

5) Calculate Covariance Matrix C.

Covariance Matrix = Processing Matrix_Transpose * Processing Matrix.
The dimension of this covariance matrix is M*M. Where, M is the number of training images.
In practical, we have to calculate the covariance matrix using formula :
Covariance Matrix = Processing Matrix * Processing Matrix_Transpose.
But, this matrix we will consume very much space as the size of the matrix will be (45,045 * 45,045) in our case.
Hence to reduce the computation space needed we perform step 5.

6) Calculate the Eigen Vectors and Eigen Values of Covariance Matrix.

Calculate the eigen vectors of matrix using the below method.
eigenValue, eigenVectors = LA.eig((covarianceMatrix))……… Step a.
But to calculate the actual eigen vector's we have to perform some extra computation.
Such as, eigen vectors_real = Processing Matrix * eigen vectors from step a.
The eigen vectors_real matrix is of dimension (45,045 * 8).
eigen vectors_real can then be projected onto the face space.
face_space = (eigen vectors_real)_transpose * column matrix.



```
PCA Coefficients for training image 1 :
[  1.53443135e+08  -4.12463935e+07  -4.29998601e-08  -5.17636090e+07
   1.14888738e+07   2.98320729e+07   4.28493981e+07  -1.73505791e+05]
PCA Coefficients for training image 2 :
[ -3.09965508e+08  -1.01569225e+08   1.14002507e-07  -7.23282904e+06
  -3.43527729e+05   6.69097274e+05   4.01893283e+06   1.38164759e+07]
PCA Coefficients for training image 3 :
[ -2.57010807e+07   5.31036358e+07  -4.20901895e-08   7.67395071e+07
  -1.13657321e+07  -8.39149947e+06   4.79155988e+07  -2.13665227e+07]
PCA Coefficients for training image 4 :
[  6.79094869e+07   2.86096157e+07  -2.81697964e-08  -2.11976451e+07
   9.00930120e+06  -4.32998677e+07   3.38368115e+06   6.14469992e+07]
PCA Coefficients for training image 5 :
[ -7.58393182e+07   9.86591897e+07   6.21237512e-09  -7.35308451e+07
  -2.03390545e+07   4.95663886e+06  -1.34689494e+07  -2.09059360e+07]
PCA Coefficients for training image 6 :
[ -5.48035678e+07   7.56177219e+07  -1.90045919e-08   4.07897743e+07
   2.52009615e+07   2.68289508e+07  -2.99136079e+07   9.97235825e+06]
PCA Coefficients for training image 7 :
```

```
[  8.55914694e+07  -5.32769258e+07   3.40638428e-08  -4.35819945e+04
   1.02912808e+07  -2.56112293e+07  -2.36191334e+07  -7.27564771e+07]
PCA Coefficients for training image 8 :
[  1.59365384e+08  -5.98976188e+07  -2.20142873e-08   3.62392288e+07
  -2.39421030e+07   1.50158366e+07  -3.11659201e+07   2.99666082e+07]
```

Here we finish our training part.

FACE RECOGNITION

1) For each image I, subtract

Average face matrix from the Input face.
Column matrix_test = Input face – Average Matrix

2) Compute its projection onto face space.
testFaceSpace = np.dot(eigenSpaceTranspose, columnMatrixTest)

3) Reconstructed face image
reconstructedInputFace = np.dot(eigenSpace, testFaceSpace)

4)Now calculate the distance between input face image and its construction
d0 = dist(column matrix_test – input face)

5) Compute the distance between input face and training images in the face space
di = dist(testFaceSpace – faceSpace)

6) In this step we choose the threshold value to identify whether our image is face image or not.
In our case we have kept the threshold value to be T0 = 7000000000000
T1 is used to identify whether the face is present in our training set or not.
T1 = 85000000
If d0 > t0, then the input face image is not a face image. Otherwise,
min(di) = dj is the face image it corresponds to.

If dj < T1, then face is recognized as face j.

Code:

```
import numpy as np
import cv2
import glob
from PIL import Image
from numpy import linalg as LA
image_list = []
columnMatrix = np.zeros((45045, 0))
# Calculating Column matrix
print "The images are input in the following order."
for filename in glob.glob('/home/surbhi/Documents/ComputerVision/Project2/Face dataset/*.jpg'): #assuming
gif
    im = Image.open(filename)
    image = cv2.imread(filename,0)
    image_list.append(im)
    print(filename)
    # print image.shape
    elementsArray = np.array(())
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            elementsArray = np.append(elementsArray, image[i][j])
    columnVector = np.zeros((len(elementsArray), 1))
    index = 0
    for item in elementsArray:
        columnVector[index, 0] = int(item)
        index += 1
    # print columnVector.shape
    columnMatrix = np.hstack((columnMatrix, columnVector))
#Calculating Mean Matrix
avgMatrix = np.zeros((columnMatrix.shape[0],1))
for i in range(columnMatrix.shape[0]):
    sum = 0
    for j in range(columnMatrix.shape[1]):
        sum += columnMatrix[i][j]
    avg = sum / columnMatrix.shape[1]
    avgMatrix[i][0] += avg
processingMatrix = np.zeros((columnMatrix.shape[0], columnMatrix.shape[1]))
for i in range(columnMatrix.shape[0]):
    for j in range(columnMatrix.shape[1]):
        processingMatrix[i][j] = columnMatrix[i][j] - avgMatrix[i][0]
# Calculating Co-variance matrix
processingMatrixTranspose = processingMatrix.transpose()
covarianceMatrix = np.array(())
covarianceMatrix = np.dot(processingMatrixTranspose, processingMatrix)
# computing the eigen values, eigen vectors for covariance matrix
eigenValue, eigenVectors = LA.eig((covarianceMatrix))
eigenValueMatrix = np.zeros((len(eigenValue), 1))
index = 0
for item in eigenValue:
    eigenValueMatrix[index,0] = item
    index += 1
eigenSpace = np.array(())
realEigenVectors = np.array((eigenVectors))
eigenSpace = np.dot(processingMatrix, realEigenVectors)
#
# for k in range(eigenSpace.shape[1]):
#     plt.title('Eighen face' + str(k+1))
#     plt.imshow((eigenSpace[:, k].reshape(231, 195)), cmap='gray')
#     plt.savefig('EigenFace'+str(k+1)+'.png')
# Projecting training face to face space
eigenSpaceTranspose = np.transpose(eigenSpace)
faceSpace = np.array(())
faceSpace = np.dot(eigenSpaceTranspose, processingMatrix)
# eigenfaces recognition
# Take the testing image as input
Test_image =
cv2.imread('/home/surbhi/Documents/ComputerVision/Project2/TestingDataset/subject01.happy.jpg',0)
# cv2.imshow('image2', Test_image)
cv2.waitKey(0)
```

```
# Subtracting mean face from test_image
columnMatrixTest = np.zeros((45045, 0))
elementsArrayTest = np.array(())
for i in range(Test_image.shape[0]):
    for j in range(Test_image.shape[1]):
        elementsArrayTest = np.append(elementsArrayTest, Test_image[i][j])
columnVectorTest = np.zeros((len(elementsArrayTest), 1))
index = 0
for item in elementsArrayTest:
    columnVectorTest[index, 0] = int(item)
    index += 1
columnMatrixTest = np.hstack((columnMatrixTest, columnVectorTest))
columnMatrixTest = columnMatrixTest - avgMatrix                    #Creating a test image column matrix
                                                                  # and subtracting it from mean face
# Compute projection onto face space
testFaceSpace = np.array(())
testFaceSpace = np.dot(eigenSpaceTranspose, columnMatrixTest)
# Reconstructing face image
reconstructedInputFace = np.array(())
reconstructedInputFace = np.dot(eigenSpace, testFaceSpace)
# print reconstructedInputFace.shape
# Compute distance between input face and its reconstruction
distanceMatrix = np.array(())
distanceMatrix = LA.norm(np.subtract(reconstructedInputFace,columnMatrixTest))
print "D0 = ",distanceMatrix
# Compute distance between input face and training images in the face space
distance = []
for i in range(len(testFaceSpace)):
    d0 = LA.norm(np.subtract(np.transpose(testFaceSpace), np.transpose(faceSpace)[i]))
    distance.append(d0)
print "distance(i) = "
print distance
print ""
#Choosing the threshold
#T0 is used to identify whether the image is face or non-face
T0 = 7000000000000
#T1 is used to identify whether the face is present in the dataset or not
T1 = 89000000
if(distanceMatrix > T0):
    print ""
    print "Classification: non-face"
else:
    minimum_val = min(distance)
    print "Distance D of Test Image to Train Image is",min(distance)
    if (minimum_val > T1):
        print ""
        print "Classification: unknown face"
    else:
        position = np.argmin(distance)
        print ""
        print "Classification: The image displayed at position = ",position, "is the image recognized."
```