Karachi Institute of Economics and Technology

Collaboration with Pakistan Air Force

NC PROJECT

FACE RECOGNITION USING PCA

This project mainly addresses the building of face recognition system by using Principal Component Analysis (PCA). PCA is a statistical approach which used the Eigenvalues and vectors and reduces the number of variables in face recognition.

Aug 15, 2022

By

Haris Ahmed

INTRODUCTION:

Face recognition is a technique for identifying or verifying a person's identity using their face. Face recognition technology can identify people in real time as well as in still photos and videos. A facial recognition system uses biometrics to map facial features from a photograph or video. It compares the information with a database of recognized faces to see if there is a match. There are lots of algorithms effective at performing face recognition but their precision may vary. I'll discuss how our project applies PCA to face recognition. PCA is a useful statistical technique
that has found application in fields such as face recognition and image compression. The PCA face recognition algorithm as proposed by Turk and Pentland. This method attempts to provide the features that determine how the face images differ from one another. These features are used to identify the face from a database of images.

PROJECT METHODOLOGY:

In this project ATT face databases have been employed. ATT database contains images with very small changes in orientation of images for each subject involved. These ATT database provide a comprehensive dataset for testing the performance of the algorithms chosen.

Figure 1: Images of a subject from the ATT Database

Language and Libraries Used:

- The implementation has been carried out using Python using the matplotlib, numpy Libraries.

ALGORITHM:

1.-Flat the black and white images of the training set (from matrices to vectors)
2.-Calculate the mean.
3.-Normalize the training set: for each image, subtract the mean.
4.-Calculate the covariance: multiply all images by themselves.
5.-Extract eigenvectors from the covariance.
6.-Calculate eigenfaces: eigenvectors x normalized pictures.
7.-Choose the most significant eigenfaces.
8.-Calculate weights: chosen eigenfaces x normalized pictures.

Imagine a picture of a face as a [m x n] image matrix. We convert each matrix into [(m x n) x 1] image vector. Thus, each face can be represented as a vector and the whole set of faces will represent a vector space of dimension m x n x N, where N is the images of all faces. This space is large. And it needs to be described with a training set of faces with Train Images. The training sample needs to be projected into this space, where each dimension will give a certain contribution to the overall presentation. The principal component method (PCA) allows us to find the basis space in such a way that the data in it are located, in a sense, optimally. Thus we can construct a new basis of smaller dimensions so that its components will carry maximum information. Leaving the dimensions with only useful information, we obtain a feature space in which each image of the original sample is presented in a generalized form. Next, we take an image from the Test set, we can map it to a pre-created space and determine which image of the training sample our example is closest to. If it is at a relatively large distance from all the data, then this image is more likely to not belong to our database at all. After applying the following algorithm, we see that the first 40 components carry almost 100% of the information. Select them. Next we will display each training image in the space of the main components and find the weights. After the vector w is known, it is necessary to determine which of the existing objects it

is closest to. Is the minimum distance and index of the object to which this image is located closest. If we input an image that is not in the source database, the vector w will still be calculated and the minimum distance will be found. Therefore the criterion t0 is introduced, if the minimum distance < t0 this means that the image belongs to the recognizable class if the minimum distance is > t0, then there is no such image in the database. The value of this criterion is chosen empirically.

CONCLUSION:

In this project face recognition using PCA algorithm were studied. The PCA using Python Programming and the performance was determined in terms of the recognition accuracy and the execution time taken. Experiments were performed under different conditions; by varying the input face image dataset and also by varying the parameters of the individual algorithms. The study PCA in terms of the accuracy of recognition is Okay. The computational PCA is Normal or Considered as Okay. Under the effects of blur in the face image not very good. For noisy images result was considered not very accurate. This algorithm fail to perform well when the input face images consist of varying orientation of the faces during image acquisition.

CODE:

In [10]:
```
from matplotlib import pyplot as plt
from matplotlib.image import imread
import numpy as np
from numpy import linalg as LA
```

Setting Protocol

40 subjects each subject have 9 images so, Training Images per subject would be 6 and Testing would be 3

In [11]:
```
path = "img"
width  = 92
height = 112
totalPerson=40 # s1- s40 or 40 subjects each subject 9 images
noOfTrainImage= 6*totalPerson #6 pimages for training
noOfTestImageImage= 3*totalPerson #3 pimages for Testing
```

In [12]:
```
training = np.empty(0)
testing = np.empty(0)
for i in range(40):
    for j in range(9):
        if j < 6:
            training = np.append(training, f"{path}\\s{i+1}\\{j+1}.pgm")
        else:
            testing = np.append(testing, f"{path}\\s{i+1}\\{j+1}.pgm")
```

1)Every Pic Quantized(0-255) Value Store in 1 Dimension

In [13]:
```
vector=np.ndarray(shape=(noOfTrainImage, height*width), dtype=np.float64) # Empty Array of 10304x240
for i in range(len(training)):
    img = plt.imread("" + training[i]) # Read image in 2d (112*92) ()
    vector[i,:] = np.array(img, dtype='float64').flatten() # Make it 1 Dimension


testingVector=np.ndarray(shape=(noOfTrainImage, height*width), dtype=np.float64)
for i in range(len(testing)):
    img = plt.imread("" + testing[i])
    testingVector[i,:] = np.array(img, dtype='float64').flatten()

print("Train Images:")
images = [plt.imread(training[i]) for i in range(1,len(training),6)]
fig, axes = plt.subplots(5, 6, sharex=True, sharey=True)
for img, ax in zip(images, axes.flat):
    ax.imshow(img, cmap='gray')
    ax.axis('off')
```

```
Train Images:
```

2) Column Wise Mean (10304x1)

In [14]:
```
mean = np.zeros((1,height*width))
for i in vector:
    mean = np.add(mean,i)
mean = np.divide(mean,float(noOfTrainImage)).flatten()
```

3) Find Mean Difference(10304x240)

In [15]:
```
meanDiff = np.ndarray(shape=(noOfTrainImage, height*width))
for i in range((noOfTrainImage)):
    meanDiff[i] = np.subtract(vector[i],mean)
```

In [16]:
```
print("Normalized Images:")
images = [(meanDiff[i].reshape(height,width)) for i in range(1,len(training),6)]
fig, axes = plt.subplots(5, 6, sharex=True, sharey=True)
for img, ax in zip(images, axes.flat):
    ax.imshow(img, cmap='gray')
    ax.axis('off')
```

```
Normalized Images:
```

4) Coveriance Calculate(280x280)

Relationship b/w two Random Variables

In [17]:
```
covMatrix=np.cov(meanDiff)
covMatrix = np.divide(covMatrix,240)
print('Covariance Matrix Shape:', covMatrix.shape)
```

```
Covariance Matrix Shape: (240, 240)
```

5) Eigen Values and Vector

koi features repeat horhe hon unki khtm krta hai

In [18]:
```
eigenValues, eigenVectors = LA.eig(covMatrix)
print('eigenvalues.shape: {} eigenvectors.shape: {}'.format(eigenValues.shape, eigenVectors.shape))
```

```
eigenvalues.shape: (240,) eigenvectors.shape: (240, 240)
```

6) EigenSpace

In [19]:
```
eigenSpace = np.dot(vector.transpose(),np.transpose(eigenVectors))
eigenSpace = eigenSpace.transpose()
eigenSpace.shape
```

Out[19]: (240, 10304)

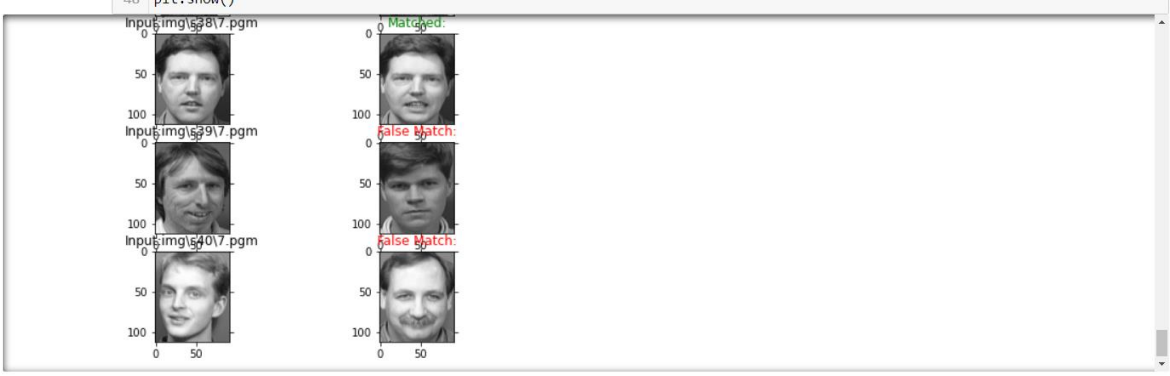
In [20]:
```
images = [(eigenSpace[i].reshape(height,width)) for i in range(0,eigenSpace.shape[0],6)]
fig, axes = plt.subplots(5, 6, sharex=True, sharey=True)
for img, ax in zip(images, axes.flat):
    ax.imshow(img, cmap='gray')
    ax.axis('off')
```

7) Mapping our Images Into Eigen Space

In [21]:
```
w = np.array([np.dot(eigenSpace,i) for i in meanDiff])
print(w.shape)
```

```
(240, 240)
```

In [23]:
```
count=0
numOfimages=0
matchImages=0
def Visualization(testImg, trainingImages,proj_data,w, highestValueinMap):
    global count,numOfimages,matchImages
    testFace= plt.imread(testImg)
    numOfimages += 1
    testFaceVector = np.array(testFace, dtype='float64').flatten()
    testFaceMeanDiff = np.subtract(testFaceVector,mean)

    plt.subplot(120,2,1+count)
    plt.imshow(testFace, cmap='gray')
    plt.title('Input:'+'.'.join(testImg.split('//')[:1]))
    plt.tick_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')
    count+=1

    mappingTest = np.dot(eigenSpace, testFaceMeanDiff)
    diff  = w - mappingTest
    ecludeanDistance = LA.norm(diff, axis=1)
    index = np.argmin(ecludeanDistance)




    plt.subplot(120,2,1+count)
    if ecludeanDistance[index] <= highestValueinMap: # It's a face
        match = testImg.split('\\' )[1] == trainingImages[index].split('\\')[1]
        if match:
            plt.title('Matched:', color='g')
            plt.imshow(imread(trainingImages[index]), cmap='gray')
            matchImages += 1
        else:
            plt.title('False Match:', color='r')
            plt.imshow(imread(trainingImages[index]), cmap='gray')


    plt.tick_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')
    plt.subplots_adjust(right=1.2, top=5.8)

    count+=1

fig = plt.figure(figsize=(5, 30))


for i in range(0,len(testing),3):
    Visualization(testing[i], training,eigenSpace,w, highestValueinMap=174784509.00590986)

plt.show()
```



In [24]:
```
print(f"Accuracy {int((matchImages/numOfimages)*100)}%")
```

```
Accuracy 75%
```

In [ ]:
```

```