Assignment 1

Face Recognition

Pattern Recognition

Omar Khaled - 5674
Noureen Mahmoud - 5810

1. Load Datasets :

Drive authorisation

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')
```

```
[ ]  path = "/content/drive/MyDrive/ORL/"
     imgs = loadImages(path)
```

ORL datasets were divided into 40 folders “Subject” each containing 10 images, hence the 2 “for” loops, which are saved in LoadedImages Matrix.

```
[ ]  def loadImages(path):
         foldersList = listdir(path)
         loadedImages = []
         for folder in foldersList :
             imagesList = listdir(path+folder)
             for image in imagesList:
                 img = PImage.open(path +folder+'/'+ image)
                 loadedImages.append(img)
         return loadedImages
```

2. Generate Data Matrix and Label Vector :

Reserving DataMatrix with size (40*10= **400**, 70*80= **5600**). Incrementing “j” every 10 images as it determines the label matrix from 1:40

```
[ ]  dataMatrix = np.arange(400*5600).reshape(400,5600)
     j=0
     label = []
     for i in range(0,400) :
         if(i%10 == 0):
             j+=1
         dataMatrix[i] = np.array(imgs[i]).flatten()
         label.append(j)
```

3. Split Dataset:

Dividing the dataset equally (200 -> training and 200 -> testing). Assigning the even images (i%2==0) to be used for testing and adding new labels according.

```
[ ]  trainSet=np.arange(200*5600).reshape(200,5600)
     testSet=np.arange(200*5600).reshape(200,5600)
     trainLabel=[]
     testLabel=[]
     j,k=0,0
     for i in range(0,400):
         if(i%2==0):
             testSet[j]=dataMatrix[i]
             testLabel.append(label[i])
             j+=1
         else:
             trainSet[k]=dataMatrix[i]
             trainLabel.append(label[i])
             k+=1
```

4. PCA:

a. Working on the training dataset hence n = 200

ALGORITHM 7.1. Principal Component Analysis

PCA (D, α):

$\mu = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$ // compute mean

$\mathbf{Z} = \mathbf{D} - \mathbf{1} \cdot \mu^T$ // center the data

$\boldsymbol{\Sigma} = \frac{1}{n}(\mathbf{Z}^T\mathbf{Z})$ // compute covariance matrix

$(\lambda_1, \lambda_2, \ldots, \lambda_d) = \text{eigenvalues}(\boldsymbol{\Sigma})$ // compute eigenvalues

$\mathbf{U} = (\mathbf{u}_1 \quad \mathbf{u}_2 \quad \cdots \quad \mathbf{u}_d) = \text{eigenvectors}(\boldsymbol{\Sigma})$ // compute eigenvectors

$f(r) = \frac{\sum_{i=1}^{r}\lambda_i}{\sum_{i=1}^{d}\lambda_i}$, for all $r = 1, 2, \ldots, d$ // fraction of total variance

Choose smallest r so that $f(r) \geq \alpha$ // choose dimensionality

$\mathbf{U}_r = (\mathbf{u}_1 \quad \mathbf{u}_2 \quad \cdots \quad \mathbf{u}_r)$ // reduced basis

$\mathbf{A} = \{\mathbf{a}_i \mid \mathbf{a}_i = \mathbf{U}_r^T\mathbf{x}_i, \text{for } i = 1, \ldots, n\}$ // reduced dimensionality data

Computing µ -> trainMean, Z -> centeredTrainMatrix, ∑ -> covMatrix

```
[ ]  trainMean=np.mean(trainSet,axis=0)
     centeredTrainMatrix=trainSet-trainMean
     covMatrix = (1/200)*np.dot(np.transpose(centeredTrainMatrix), centeredTrainMatrix)
```

Computing λ -> eigvVal and U -> eigeVect. Then sorting them in descending order as the last few will be relatively negligible

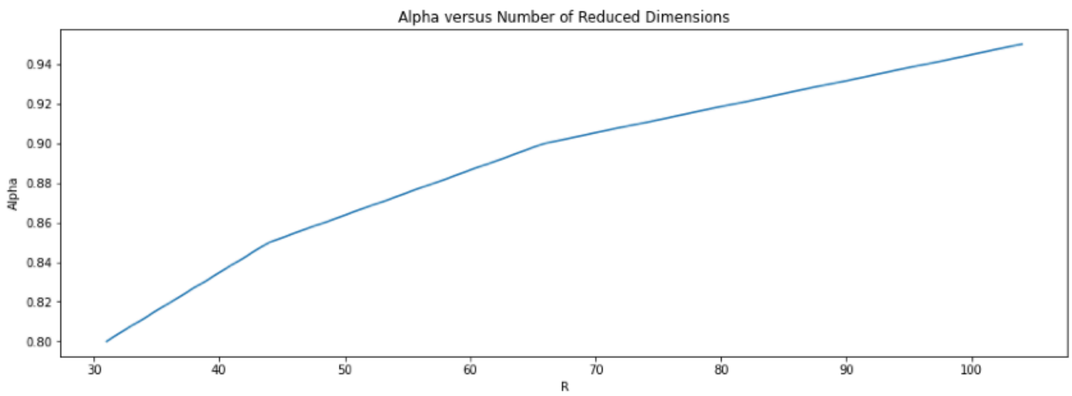
```
[ ] eigVal,eigVect=np.linalg.eigh(covMatrix)
    idx = eigVal.argsort()[::-1]
    sortedEigVal = np.real(eigVal[idx])
    sortedEigVect = np.real(eigVect[:,idx])
```

To choose the right dimensionality (R):
we have to compute $f(r)$ -> fractionOfTotalVariance which is smaller than α-> threshold

```
[ ] def computeDimensionality(eigVal,threshold):
        dataVariance=np.sum(eigVal)
        fractionOfTotalVariance,R,eigValSum=0,1,0
        while(fractionOfTotalVariance<threshold):
            eigValSum+=eigVal[R-1]
            fractionOfTotalVariance=eigValSum/dataVariance
            R+=1
        return R
```

Plotting different alpha {0.8,0.85,0.9,0.95} against R

```
[ ] Alpha1dim=computeDimensionality(sortedEigVal,0.8)
    Alpha2dim=computeDimensionality(sortedEigVal,0.85)
    Alpha3dim=computeDimensionality(sortedEigVal,0.9)
    Alpha4dim=computeDimensionality(sortedEigVal,0.95)
    plt.plot([Alpha1dim,Alpha2dim,Alpha3dim,Alpha4dim], [0.8,0.85,0.9,0.95]);
    plt.title('Alpha versus Number of Reduced Dimensions');
    plt.gcf().set_size_inches(15,5);
    plt.xlabel('R');
    plt.ylabel('Alpha');
```



As alpha increases number of reduced dimensions increases

b. Project the training set, and test sets separately using the same projection matrix.

```
[ ]  projMat1=sortedEigVect[:,0:Alpha1dim]
     projMat2=sortedEigVect[:,0:Alpha2dim]
     projMat3=sortedEigVect[:,0:Alpha3dim]
     projMat4=sortedEigVect[:,0:Alpha4dim]
```

```
[ ]  reducedTrain1= np.dot(trainSet,projMat1)
     reducedTrain2= np.dot(trainSet,projMat2)
     reducedTrain3= np.dot(trainSet,projMat3)
     reducedTrain4= np.dot(trainSet,projMat4)
```

```
[ ]  reducedTest1= np.dot(testSet,projMat1)
     reducedTest2= np.dot(testSet,projMat2)
     reducedTest3= np.dot(testSet,projMat3)
     reducedTest4= np.dot(testSet,projMat4)
```

c. Finding the K-Nearest Neighbour (K = 1 as it's required the simplest)

```
[ ]  def knn(trainingSet,trainingLabel,testSet,testLabel,k):
         knn = KNeighborsClassifier(n_neighbors=k)
         knn.fit(trainingSet,trainingLabel)
         predict=knn.predict(testSet)
         accuracy  = metrics.accuracy_score(testLabel,predict)
         return accuracy
```

```
[ ]  knn(reducedTrain1,trainLabel,reducedTest1,testLabel,1)

     0.895
```

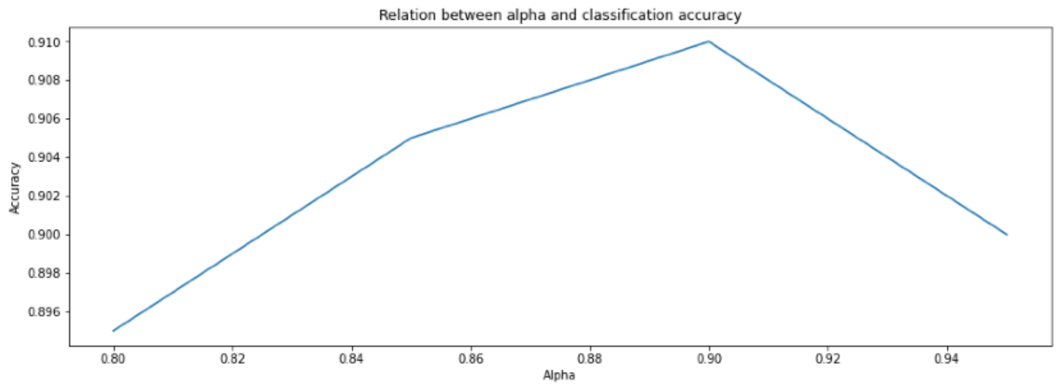
```
[ ]  knn(reducedTrain2,trainLabel,reducedTest2,testLabel,1)

     0.905
```

```
[ ]  knn(reducedTrain3,trainLabel,reducedTest3,testLabel,1)

     0.91
```

```
[ ]  knn(reducedTrain4,trainLabel,reducedTest4,testLabel,1)

     0.9
```

d. Report Accuracy for every value of alpha separately

```
[ ]  plt.plot([0.8,0.85,0.9,0.95],[0.895,0.905,0.91,0.9]);
     plt.title('Relation between alpha and classification accuracy');
     plt.gcf().set_size_inches(15,5);
     plt.xlabel('Alpha');
     plt.ylabel('Accuracy');
```



e. Can you find a relation between alpha and classification accuracy?

as alpha increases, classification accuracy increases until it reaches 0.91 then it start decreasing

5. LDA

ALGORITHM 20.1. Linear Discriminant Analysis

LINEARDISCRIMINANT $(\mathbf{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n)$:

$\mathbf{D}_i \leftarrow \{\mathbf{x}_j \mid y_j = c_i, j = 1, \dots, n\}, i = 1, 2$ `// class-specific subsets`

$\boldsymbol{\mu}_i \leftarrow \text{mean}(\mathbf{D}_i), i = 1, 2$ `// class means`

$\mathbf{B} \leftarrow (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T$ `// between-class scatter matrix`

$\mathbf{Z}_i \leftarrow \mathbf{D}_i - \mathbf{1}_{n_i}\boldsymbol{\mu}_i^T, i = 1, 2$ `// center class matrices`

$\mathbf{S}_i \leftarrow \mathbf{Z}_i^T\mathbf{Z}_i, i = 1, 2$ `// class scatter matrices`

$\mathbf{S} \leftarrow \mathbf{S}_1 + \mathbf{S}_2$ `// within-class scatter matrix`

$\lambda_1, \mathbf{w} \leftarrow \text{eigen}(\mathbf{S}^{-1}\mathbf{B})$ `// compute dominant eigenvector`

a) i) compute mean for each class μ_i -> classesMeans

```
[ ]  classesList=[]
     for i in range(0,201):
         if(i%5==0 and i!=0):
             classesList.append(trainSet[i-5:i,:])
```

```
[ ]  classesMeans = np.mean(classesList, axis=1)
```

a) ii) Replace B-> betweenClassScatterMatrix with S_b

where $n_k = 5$, μ_k -> overallSampleMean, μ -> classesMeans

```
[ ] overallSampleMean = trainSet.mean(axis=0)
```

```
[ ] betweenClassScatterMatrix = 5 * np.dot(np.transpose(classesMeans - overallSampleMean),(classesMeans - overallSampleMean))
```

a) iii) S matrix -> withinClassScatterMatrix

```
[ ] withinClassScatterMatrix = np.zeros((5600,5600))
    for i in range(0,40):
        withinClassScatterMatrix += np.dot(np.transpose(classesList[i] - classesMeans[i]) , (classesList[i] - classesMeans[i]))
```

a) iv) Using 39 dominant eigenvector

calculate s^{-1} then compute dot product with S_b

Sort the eignVectLDA and take the greatest 39 dimensions

```
[ ] withinClassScatterMatrixInv = np.linalg.pinv(withinClassScatterMatrix)
```

```
[ ] sinverse_b = np.dot(withinClassScatterMatrixInv , betweenClassScatterMatrix)
```

```
[ ] eigValLDA,eigVectLDA = np.linalg.eig(sinverse_b)
```

```
[ ] idx = eigValLDA.argsort()[-39:][::-1]
    sortedEigVectLDA = np.real(eigVectLDA[:,idx])
```

b) Project the training set, and test sets separately using the same projection matrix U.

```
[ ] reducedTrainLDA = np.dot(trainSet,sortedEigVectLDA)
    reducedTestLDA = np.dot(testSet,sortedEigVectLDA)
```

c) Use a simple classifier (first Nearest Neighbour to determine the class labels).

d) Report Accuracy for the Multi-class LDA on the face recognition dataset.

```
[ ] knn(reducedTrainLDA,trainLabel,reducedTestLDA,testLabel,1)

0.865
```

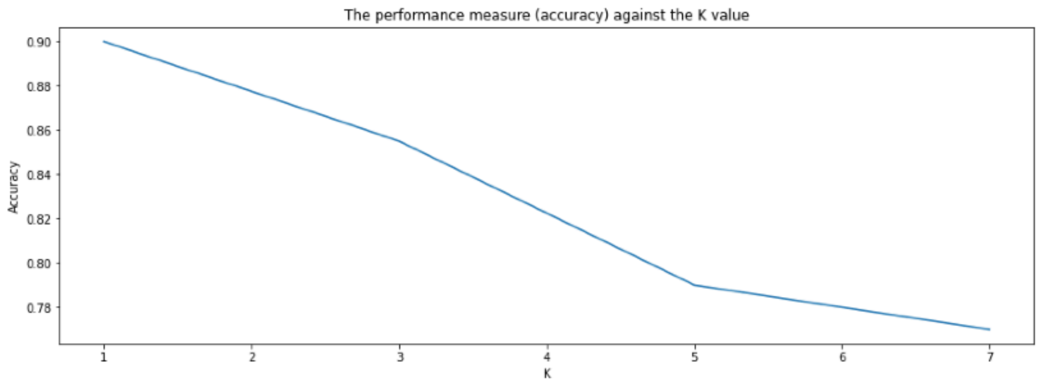
e) Compare the results to PCA results.

PCA is slightly higher than LDA yet still in the same range

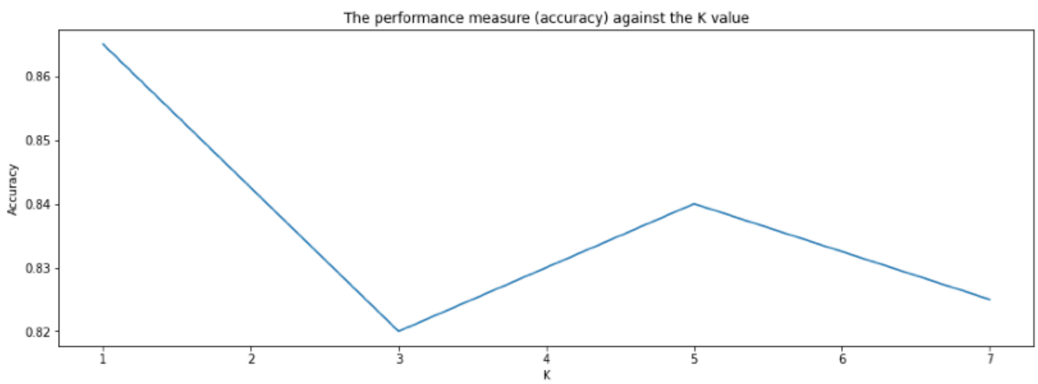
6. Classifier Tuning:

Finding series of k-nearest neighbour [1,3,5,7] instead of the first-nearest neighbour in PCA and LDA

```
[ ] PCAscoreList = []
    for k in range(1,8,2):
        PCAscoreList.append(knn(reducedTrain4,trainLabel,reducedTest4,testLabel,k))
    plt.plot([1,3,5,7],PCAscoreList);
    plt.title('The performance measure (accuracy) against the K value');
    plt.gcf().set_size_inches(15,5);
    plt.xlabel('K');
    plt.ylabel('Accuracy');
```




```
[ ] LDAscoreList = []
    for k in range(1,8,2):
        LDAscoreList.append(knn(reducedTrainLDA,trainLabel,reducedTestLDA,testLabel,k))
    plt.plot([1,3,5,7],LDAscoreList);
    plt.title('The performance measure (accuracy) against the K value');
    plt.gcf().set_size_inches(15,5);
    plt.xlabel('K');
    plt.ylabel('Accuracy');
```



In PCA and LDA, both accuracy decrease when the number of K increase

Compare vs Non-Face Images:

Repeated all steps from 1 to 6 . However instead of creating 40 labels there will be only two (1 -> faces and 2 -> non-face images). Downloading 400 non-face images called “Cifar” and changing their sizes to 70*80 to match the ORL images as well as grey scale

```
[ ]  dataMatrix = np.arange(800*5600).reshape(800,5600)
     label = []
     for i in range(0,400) :
         dataMatrix[i] = np.array(imgs[i]).flatten()
         label.append(1)
     for i in range(400,800) :
         dataMatrix[i] = np.array(cifar[i-400]).flatten()
         label.append(2)
```

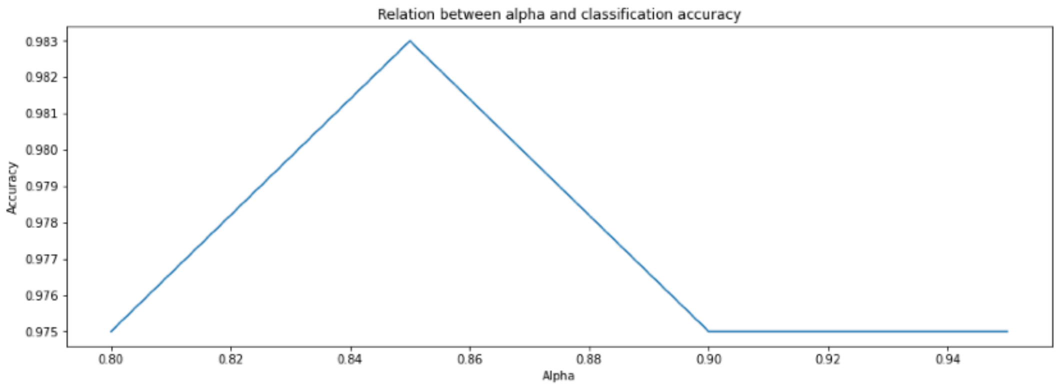
Criticise the accuracy measure for large numbers of non-faces images in the training data?

When the number of non-faces images increases, algorithm’s efficiency in detecting the non-faces images will increase, however this doesn’t change face detection.

Bonus:

Repeated all steps from 1 to 6. The only difference, the dataset is not split equally instead 7:3 (280-> training and 120->testing)

```
[ ]  plt.plot([0.8,0.85,0.9,0.95],[0.975,0.983,0.975,0.975]);
     plt.title('Relation between alpha and classification accuracy');
     plt.gcf().set_size_inches(15,5);
     plt.xlabel('Alpha');
     plt.ylabel('Accuracy');
```




```
[ ]  knn(reducedTrainLDA,trainLabel,reducedTestLDA,testLabel,1)

     0.9583333333333334
```

When the number of training images increased, accuracy increased in both PCA and LDA