Pattern Recognition CC484N,

Computer and communication Engineering (SSP),

Alexandria University

Face Recognition using ORL Dataset

Hania Ahmed 5369

Aly Feteha 5894

Mohamed Gamal 5464

Seif Eldin Ahmed 5366

Abstract

Face recognition from images is a sub-area of the general object recognition problem. It is of particular interest in a wide variety of applications. Here, the face recognition is based on the PCA algorithm and modified LDA algorithm. The aim is to show that LDA is better than PCA in face recognition. Face and facial feature detection plays an important role in various applications such as human computer interaction, video surveillance, face tracking, and face recognition. Face recognition not only makes hackers virtually impossible to steal one's " password" but also increases the user-friendliness in human-computer interaction. Apparently the face is the most visible part of human anatomy and serves as the first distinguishing factor of a human being

FACE RECOGNITION

Libraries used:

```
import os

import cv2

from PIL import Image

import numpy as np

from os import listdir

from os.path import isfile, join

from matplotlib import pyplot, pyplot as plt

from sklearn.neighbors import KNeighborsClassifier

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

Functions :

Face recognition with ORL dataset

```
def read_file(dir_name):

   matrix_input = np.zeros(shape=(1, 10304))

   imgs_counter = 0

   files_counter = 0

   for subdir, dirs, files in sorted(os.walk(dir_name)):

       files_counter += 1

       for filename in sorted(files, key=len):

           filepath = subdir + os.sep + filename

           if filepath.endswith(".pgm"):

               imgs_counter += 1

               new_row = convert_img(filepath)

               matrix_input = np.append(matrix_input, np.matrix(new_row),
axis=0)
```

```
            if filepath.endswith(".jpg"):
                imgs_counter += 1
                img = Image.open(filepath)
                resized_img = img.resize((92, 112))
                img = resized_img.convert('L')
                ready_image = np.array(img).reshape((1, 10304))
                matrix_input = np.append(matrix_input, ready_image, axis=0)
                # print(ready_image)
    files_counter -= 1
    matrix_input = np.delete(matrix_input, 0, 0)
    return matrix_input, imgs_counter, files_counter
def convert_img(img_path):
    img = cv2.imread(img_path, -1).flatten()
    return img
```

In this function we read the file and place the images into a matrix_input after converting them using cv2.imread() which is in opencv library, if the image extension is ".pgm", otherwise if the image extension is ".jpg" then we resize the image, convert it and reshape it (this is used only with the non-faces dataset). Then the matrix_input, and the counters are returned.

```
def data_split(imgs, imgs_counter, files_counter, faces_flag):
    training = np.zeros(shape=(1, 10304))
    testing = np.zeros(shape=(1, 10304))
    split_labels_vector = np.arange(imgs_counter / 2)
    k = 0
    for i in range(int(imgs_counter / 2)):
        if (faces_flag):  # faces dataset
```

```
            if (i % int((imgs_counter / files_counter) / 2) != 0):
                split_labels_vector[i] = k
            else:
                k = k + 1
                split_labels_vector[i] = k
        else:  # faces and non faces dataset
            if (i < int(imgs_counter / 4)):
                split_labels_vector[i] = 0  # face
            else:
                split_labels_vector[i] = 1  # noface
    for i in range(int(imgs_counter)):
        if (i % 2 == 0):
            testing = np.append(testing, np.matrix(imgs[i]), axis=0)
        else:
            training = np.append(training, np.matrix(imgs[i]), axis=0)
    testing = np.delete(testing, 0, 0)
    training = np.delete(training, 0, 0)
    return testing, training, split_labels_vector
```

In data_split function, the dataset is splitted into testing and training, and the labels are created for each class in training(in faces dataset 40 classes: label for each person, in non-faces dataset 2 classes: label for faces and another for non-faces) by using the counters returned from read_file function and checking the faces flag, if it is set then the first if condition is satisfied and each person takes the same label for their photos in the training dataset, otherwise the non-faces training dataset gets labeled 0 for faces and 1 for non-faces.

FACE RECOGNITION

The PCA algorithm

The PCA algorithm is used to reduce the dimensionality of the data given based on the spreading/variance of this data. It calculates the eigenvectors and the eigenvalues and project the data based on the highest set of eigenvalues which is calculated from the expected variance and compared to the alpha given, and it is done to remove the noise and uncorrelated data and project only the important and correlated data.

```
def PCA(training, testing, alpha, split_labels_vectors):
   print("------------------PCA------------------")
   mean = np.mean(training, axis=0)
   centralized_matrix = training - mean
   cov = np.cov(centralized_matrix, bias=True, rowvar=False)
   eigenvalues, eigenvectors = np.linalg.eigh(cov)
   evalues_mat = np.diag(eigenvalues)
   index = np.argsort(eigenvalues)[::-1]
   sorted_evalues = eigenvalues[index]
   sorted_evectors = eigenvectors[:, index]
   r = 0
   while (np.sum(sorted_evalues[0:r]) / np.sum(sorted_evalues) - alpha <=
1e-6):
       r += 1
   projected_matrix_training = np.dot(training, sorted_evectors[:, :r])
   projected_matrix_testing = np.dot(testing, sorted_evectors[:, :r])
   knn = [1, 3, 5, 7]
   score_calc(knn, projected_matrix_training, projected_matrix_testing,
split_labels_vectors, "PCA")
```

FACE RECOGNITION

The algorithm:

- Calculate the mean of the training matrix
- Calculate Z matrix (Z = training - mean)
- Get the covariance matrix and compute the eigenvectors and eigenvalues
- Sort the eigenvalues
- Loop to get the expected variance <= the given alpha
- Calculate the reduced dimensions matrix and project it

The PCA aims to get the highest accuracy using the expected variance that is almost equivalent to the given alpha (0.8,0.85,0.9,0.95) that is passed one by one to the function. Then it compares them to the results of KNN.

The LDA algorithm

Linear Discriminant Analysis is a “classical” technique in pattern recognition, where it is used to find a linear combination of features which characterize or separate two or more classes of objects or events. The resulting combination is used for dimensionality reduction before it can be classified.

```
lda(training, split_labels_vectors, testing):
   print("------------------------------LDA------------------------------")
   mean_training = np.mean(training, axis=0)
   mean = np.zeros(shape=(1, 10304))
   S = np.zeros(shape=(1, 10304))
   Sb = np.zeros(shape=(1, 10304))
   for i in range(5, len(training) + 5, 5):
```

```
        mean = np.append(mean, np.matrix(np.mean(training[i - 5:i], axis=0)),
axis=0)
    mean = np.delete(mean, 0, 0)
    for i in range(40):
        Sb = Sb + np.dot(5 * ((mean[i] - mean_training).T), mean[i] -
mean_training)
    k = 0
    Z = np.zeros(shape=(1, 10304))
    for i in range(5, len(training) + 5, 5):
        Z = np.append(Z, np.matrix(training[i - 5:i]) - mean[k], axis=0)
        k = k + 1
    Z = np.delete(Z, 0, 0)
    for i in range(200):
        S = S + (np.dot(Z[i].T, Z[i]))
    eigenvalues, eigenvectors = np.linalg.eigh(np.dot(np.linalg.inv(S), Sb))
    knn = [1, 3, 5, 7]
    index = np.argsort(eigenvalues)[::-1]
    sorted_evectors = eigenvectors[:, index]
    dims = sorted_evectors[:, 0:39]
    projected_matrix_training = np.dot(training, dims)
    projected_matrix_testing = np.dot(testing, dims)
    score_calc(knn, projected_matrix_training, projected_matrix_testing,
split_labels_vectors, "LDA")
```

The algorithm:

- Calculate the mean of the training matrix

FACE RECOGNITION

- Calculate the mean of each class in the training matrix
- Calculate the B matrix ($B = \sum nk\ (\mu k - \mu)(\mu k - \mu)T$)
- Calculate Z matrix (class matrices) ($Zi = Di - \mu i$)
- Calculate S matrix ($S = \sum Zi.T \cdot Zi$)
- Calculate eigenvalues and vectors using eig(S^-1 . B)
- Calculate the reduced dimensions matrix (of 39 eigenvectors)and project it

Then the results of LDA is passed to score_calc to be compared to the results of KNN classification

KNN

```
def score_calc(knn, projected_matrix_training, projected_matrix_testing,
split_labels_vectors, title):
   print("------------ score ", title, " -----------")
   scores = [0, 0, 0, 0]
   for i in range(len(knn)):
       neigh = KNeighborsClassifier(n_neighbors=knn[i], weights='distance')
       neigh.fit(projected_matrix_training, split_labels_vectors)
       scores[i] = neigh.score(projected_matrix_testing, split_labels_vectors)
   plt.scatter(knn, scores)
   plt.title(title)
   plt.show()
```

The KNN algorithm deals with supervised data to produce appropriate results if an unlabeled (unsupervised) data is entered by checking the class that the new data is closest to and adding it to that class. This is done according to the number of neighbors given (K) and the distances between the query point and these neighbours, the optimal value of neighbours is according to the datasets but K = 1 gives the highest accuracy as the Data tests have high similarity with the training data, also K is preferred to be an odd number for tie breaking.

In our case, the main class is the training matrix and the testing matrix is the checked data, if the distance between the testing matrix elements and one of the classes of the training matrix is smaller than the rest then these elements belong to that class and a high score is produced (high accuracy) otherwise a low score (low accuracy). And since the testing and the training matrices both have the same people but with different photos then the accuracy here is high.

The KNN function is called in both PCA and LDA to measure the accuracy of each algorithm and LDA shows a higher score when K = 1 and K = 3, and the same score in both when K = 5 and K = 7. Therefore, LDA is more accurate in face recognition than PCA in most K values

PCA scores: [0.93, 0.91, 0.875, 0.865] -> ALPHA = 0.8

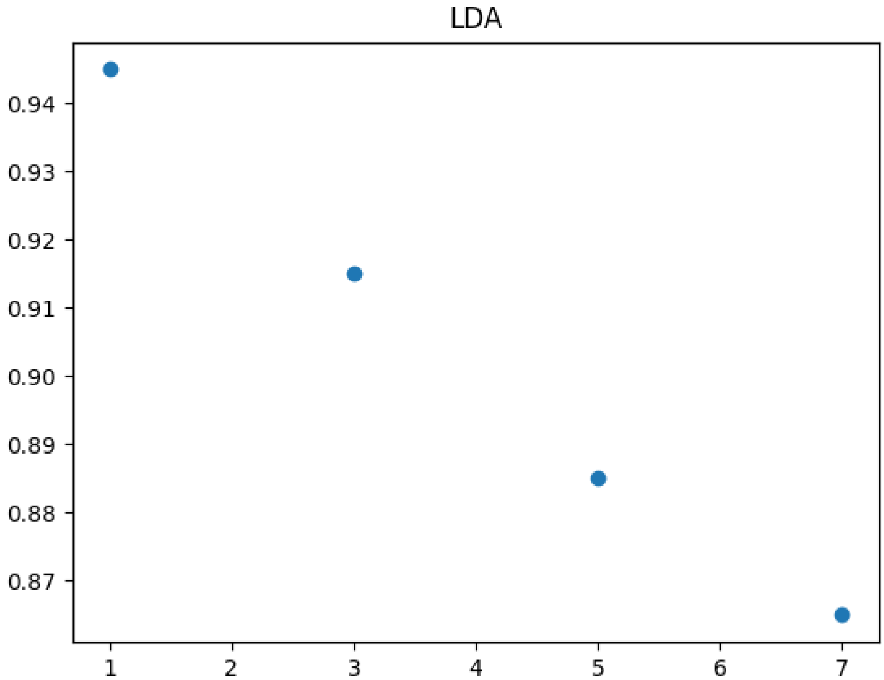
PCA scores: [0.94, 0.905, 0.89, 0.86] -> ALPHA = 0.85

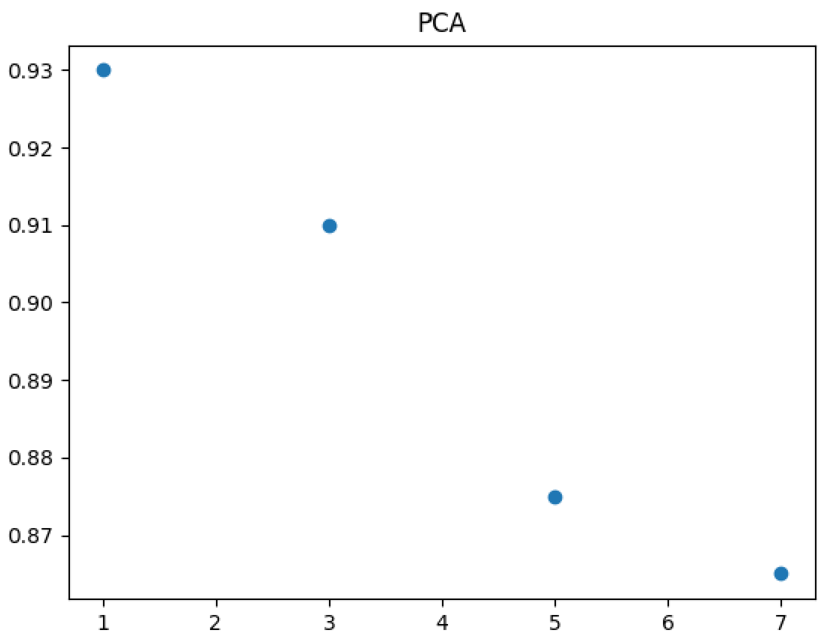
PCA scores: [0.945, 0.905, 0.895, 0.855] -> ALPHA = 0.9

PCA scores: [0.935, 0.895, 0.885, 0.865] -> ALPHA = 0.95

LDA scores: [0.945, 0.915, 0.885, 0.865]

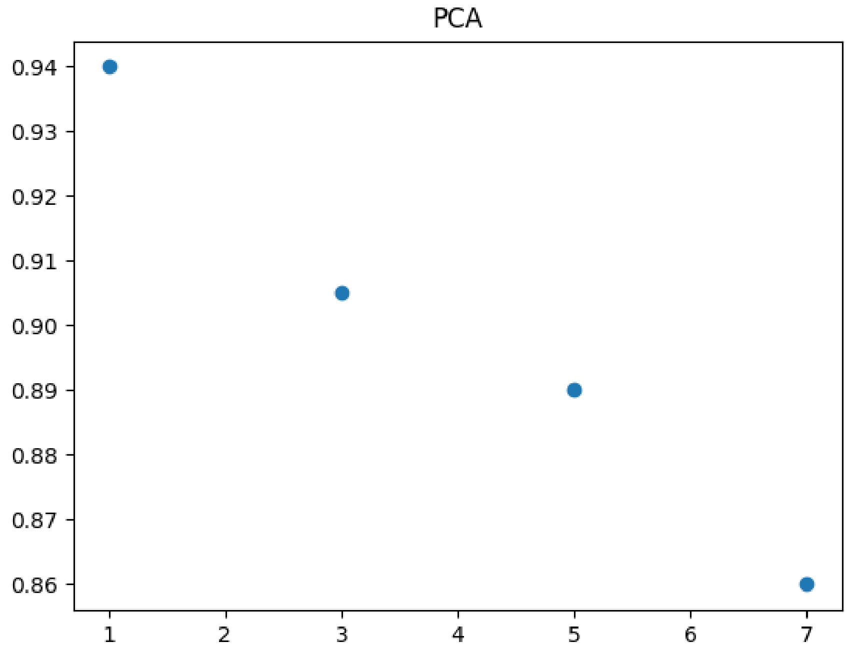
FACE RECOGNITION



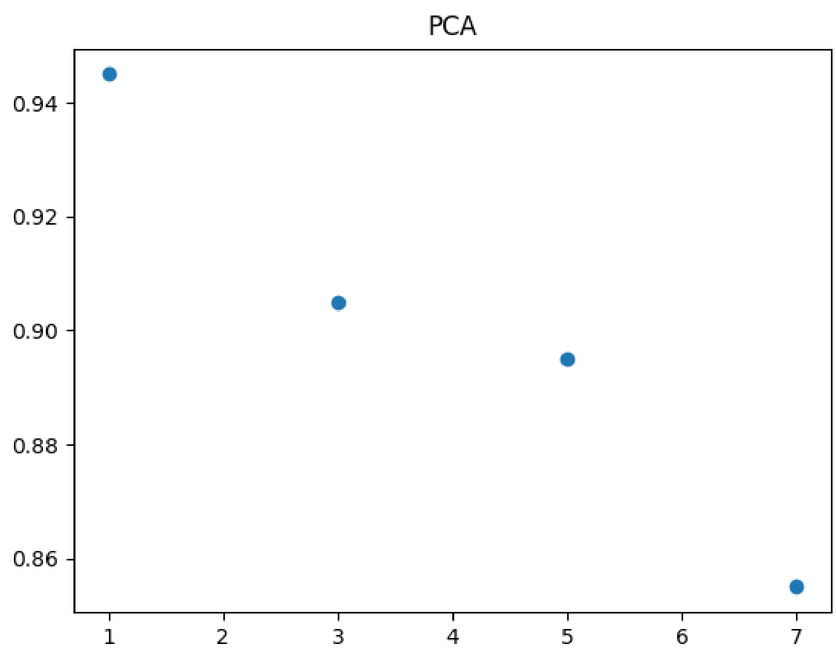


ALPHA = 0.8

FACE RECOGNITION

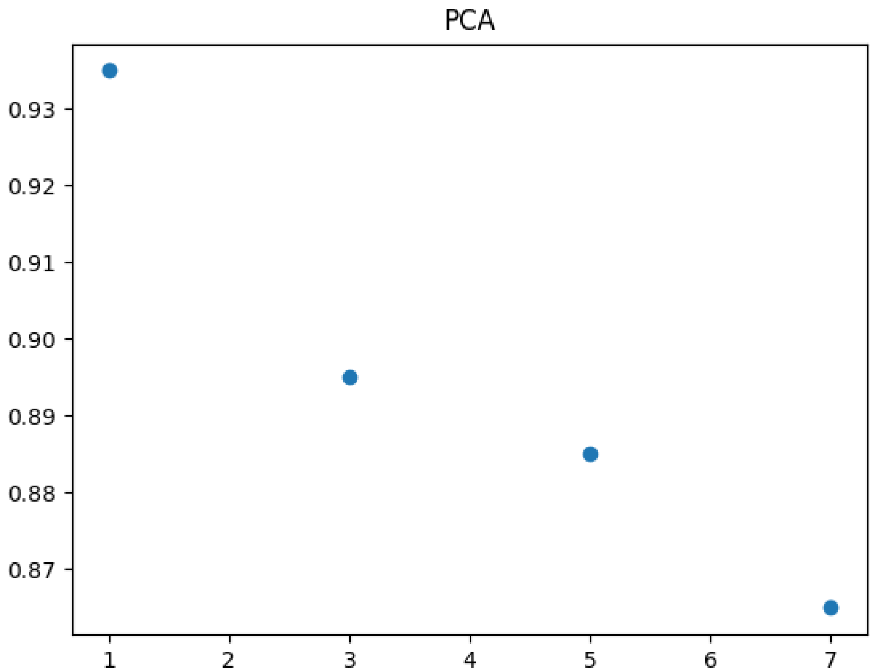


ALPHA = 0.85



ALPHA = 0.9

FACE RECOGNITION



ALPHA = 0.95

Faces VS non-faces

For the faces dataset we used the same ORL dataset and for the non-faces cats dataset is used.

The datasets are read the same way in read_file() then passed through data_split() to get the labels using the else part in this for loop

```
for i in range(int(imgs_counter / 2)):
      if (faces_flag):  # faces dataset
          if (i % int((imgs_counter / files_counter) / 2) != 0):
              split_labels_vector[i] = k
          else:
              k = k + 1
              split_labels_vector[i] = k
      else:  # faces and non faces dataset
          if (i < int(imgs_counter / 4)):
```

```
                split_labels_vector[i] = 0  # face
            else:
                split_labels_vector[i] = 1  # noface
```

Then the LDA algorithm is called but this time lda_no_face() is called which performs the same as lda function but with different lengths for the different distribution of labels.

```
def lda_no_face(training_no_face, split_labels_vectors, testing_no_face):
   mean_training = np.mean(training_no_face, axis=0)
   mean = np.zeros(shape=(1, 10304))
   S = np.zeros(shape=(1, 10304))
   Sb = np.zeros(shape=(1, 10304))
   for i in range(200, len(training_no_face) + 200, 200):
       mean = np.append(mean, np.matrix(np.mean(training_no_face[i - 200:i],
axis=0)), axis=0)
   mean = np.delete(mean, 0, 0)
   print(mean)
   print(mean.shape)
   for i in range(2):
       Sb = Sb + np.dot(200 * ((mean[i] - mean_training).T), mean[i] -
mean_training)
   k = 0
   Z = np.zeros(shape=(1, 10304))
   for i in range(200, len(training_no_face) + 200, 200):
       Z = np.append(Z, np.matrix(training_no_face[i - 200:i]) - mean[k],
axis=0)
       k = k + 1
   Z = np.delete(Z, 0, 0)
```

```
for i in range(10):
    S = S + (np.dot(Z[i].T, Z[i]))
eigenvalues, eigenvectors = np.linalg.eigh(np.dot(np.linalg.inv(S), Sb))
index = np.argsort(eigenvalues)[::-1]
sorted_evectors = eigenvectors[:, index]
dims = sorted_evectors[:, 0:39]
projected_matrix_training = np.dot(training_no_face, dims)
projected_matrix_testing = np.dot(testing_no_face, dims)
score_calc_lda2(projected_matrix_training, projected_matrix_testing,
split_labels_vectors, "LDA NON-FACES")
```

Then in order to calculate the accuracy with KNN classification, score_calc_lda2() function is called which also performs KNN algorithm but with K = 1 and dividing the training set into 4 subsets and calculating the accuracy score for each subset.

non face scores: [0.915, 0.93, 0.935, 0.9475]