Face Recognition System

*(Using Eigenfaces Method)*

****

# Cluster Innovation Centre

# University of Delhi

**AMIT | ASHUTOSH | TUSHAR**

NOV 2020

Month long project submitted for the paper of:

*Algorithms for Computational Mathematics: Numerical Methods*

# CERTIFICATE OF ORIGINALITY

The work embodied in this report entitled **“Face Recognition System (Using Eigenfaces Method)”** has been carried out by **Amit, Ashutosh and Tushar** for the paper of **“Algorithms for Computational Mathematics: Numerical Methods”**. We declare that the work and language included in this project report is free from any kind of plagiarism.

The work submitted is original and has not been submitted earlier to any institute or university for the award of any degree or diploma.

**Amit**

(11807)

**Ashutosh**

(11811)

**Tushar**

(11840)

# ACKNOWLEDGEMENT

With a deep sense of gratitude, we express our dearest indebtedness to **Prof. Nirmal Yadav** for her support throughout the duration of our project. We would like to thank her for giving us the opportunity to do this wonderful project. Her learned advice and constant encouragement has helped us complete this project. It is a privilege for us to be her students. Without her encouragement and guidance this project would not have materialized.

We are also thankful to our friends and family who helped us with their valuable suggestions and guidance that have been helpful in various phrases of the completion of the project.

# ABSTRACT

This project is designed on the idea that each person has a particular face structure which forms the basis for the *Face recognition system*. Using the facial symmetry, computerized face-matching is possible. The work on face recognition began in the *1960’s*, the results of which are being used for security in various institutions and firms throughout the world. Its market is expected to grow to $7.7 billion by 2022 due to its vast commercial applications.

For computer based face recognition, the images must be processed correctly. The face and its structural properties should be identified carefully, and the resulting image must be converted to *two dimensional digital data*. An efficient algorithm and a database which consists of face images are needed to solve the face recognition problem.

In this project, we have implemented the *Face Recognition System* using *Eigenfaces Method*. In this recognition process, an eigenface is formed for the given face image, and the Euclidean distances between this eigenface and the previously stored eigenfaces are calculated. The eigenface with the smallest Euclidean distance is the one the person resembles the most. The analysis was performed on *105 images* of *15 different people*. An accuracy of *97%* was achieved for our analysis.

# TABLE OF CONTENTS

1. [**Introduction**](#_asvy8tsjlmm8)
2. [**Formulation of problem**](#_2ivgacmuhisb)
   1. [Problem statement](#_893qs4w4re6)
   2. [Theoretical concepts](#_vb8rbobx2tyd)
      1. [Eigenfaces method](#_r21ubm8uyalv)
3. [**Methodology**](#_wjrpmga8dr88)
   1. [Softwares used](#_ghyszprfb3sq)

## [Eigenfaces method for the solution of face recognition problem](#_mz3lb6rd480t)

1. [**Results**](#_cashje7paurz)
2. [**Conclusion**](#_hy4eq6a2h6h8)
3. [**References**](#_gr2w04abc9fk)
4. [**Appendix**](#_ridefnrt9rqq)

# INTRODUCTION

## The face recognition system is similar to other biometric systems. Similar to the fingerprint, the face of an individual has many structures and features unique to that individual. An automatic face recognition system is based on facial symmetry. Face authentication and identification are challenging problems. The fact that in the recent past, there have been more and more commercial, military and institutional applications, making the face recognition systems a very popular subject. To be reliable, such systems have to work with high precision and accuracy.

## In a face recognition system, the database consists of the images of the individuals that the system has to recognize. If possible, several images of the same individual should be included in the database. If the images are selected so that they account for varying facial expressions, lighting conditions, etc., the solution of the problem can be found more easily as compared to the case where only a single image of each individual is stored in the database. A face recognition algorithm processes the captured image and compares it to the images stored in the database. If a match is found, then the individual is identified. If no match is found, then the individual is reported as unidentified.

# FORMULATION OF PROBLEM

## Problem statement:

## The challenges of face recognition are:

## *Shifting and scaling of the image,*

* *Differences in the facial look (different angle, pose, hairstyle, makeup, mustache, beard, etc.)*
* *Lighting,*
* *Aging.*

The algorithm has to work successfully even with the above challenges. In *Table 1*, a comparison of some of the methods used for face recognition based on the number of images in the training set and the resulting success rate is provided.

**Table 1:** Comparison of some methods used for face recognition

| ***Method*** | ***Number of images in the training set*** | ***Success Rate*** |
| --- | --- | --- |
| *Principal Component Analysis* | *400* | *79.65 %* |
| *Principal Component Analysis + Relevant Component Analysis* | *400* | *92.34 %* |
| *Independent Component Analysis* | *40* | *81.35 %* |
| *Hidden Markov Model* | *200* | *84 %* |
| *Active Shape Model* | *100* | *78.12 - 92.05 %* |
| *Wavelet Transform* | *100* | *80-91 %* |
| *Support Vector Machines* | *-* | *85-92.1 %* |
| *Neural Networks* | *-* | *93.7 %* |
| *Eigenfaces Method* | *70* | *92 - 100 %* |

To detect a face using the database provided, we'll be using a method of recognition called *Eigenfaces Method*. This method is an application of the famous *Principal Component Analysis (PCA)*.

Generally speaking, we can assume each image of face as a *30\*30 images*. So we can flatten it to be a *900 - element vector* in the *900 - dimensional space*. We're looking for directions in which these vectors of faces (when projected on those directions) vary most. Those directions will be the eigenfaces. Eigenfaces, in most cases, turn out to look like blurry faces!



**Fig. 1:** Some eigenfaces for example

It actually turns out that *eigenfaces* are the *eigenvectors* of the *covariance matrix* of faces vectors. By assuming that each image of the face is a random vector, we can estimate their covariance matrix.

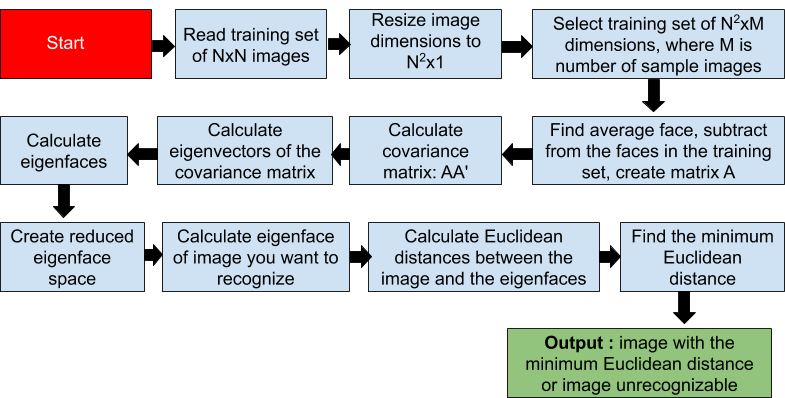
The more the eigenvalue corresponding to an eigenvector, the more variance is there when projecting faces vectors over that eigenvector.

## Theoretical Concepts:

### *Eigenfaces method:* The basis of the eigenfaces method is the *Principal Component Analysis (PCA)*. Eigenfaces and PCA have been used by *Sirovich* and *Kirby* to represent the face images efficiently. They have started with a group of original face images, and calculated the best vector system for image compression. Then *Turk* and *Pentland* applied the Eigenfaces to face recognition problems.

The *Principal Component Analysis (PCA)* is a method of projection to a subspace and is widely used in pattern recognition. An objective of *PCA* is the replacement of correlated vectors of large dimensions with the uncorrelated vectors of smaller dimensions. Main advantages of the *PCA* are its low sensitivity to noise, the reduction of the requirements of the memory and the capacity, and the increase in the efficiency due to the operation in a space of smaller dimensions.

The strategy of the eigenfaces method consists of extracting the characteristic features on the face and representing the face in question as a linear combination of the eigenfaces obtained from the feature extraction process. The principal components of the faces in the training set are calculated. Recognition is achieved using the projection of the face into the space formed by the eigenfaces. A comparison on the basis of the *Euclidean distance* of the eigenvectors of the eigenfaces and the eigenface of the image under question is made. If this distance is small enough, the person is identified. On the other hand, if the distance is too large, the image is regarded as one that belongs to an individual for which the system has to be trained.



**Fig. 2:** Flow chart showing working of eigenfaces method (algorithm)

The reasons for selecting the eigenfaces method for face recognition are:

* *Its independence from the facial geometry,*
* *The simplicity of realization,*
* *Possibility of real-time realization even without special hardware,*
* *The ease and speed of recognition with respect to the other methods,*
* *The higher success rate in comparison to other methods.*

The *challenge* of the eigenfaces face recognition method is the computation time. If the database is large, it may take a while to retrieve the identity of the person under question.

# METHODOLOGY

## Softwares used:

* *System Softwares:*
  + *Ubuntu Desktop 20.04 LTS*
* *Application Softwares:*
  + *Python 3.8.5*
  + *Jupyter Notebook 2.2.6*

## Eigenfaces method for the solution of face recognition problem:

Following steps are involved in the detailed working of this algorithm:

* *Flattening and representing the image as column image vector:* As a starting point, training images of the dimensions *N \* N* are read and then they are converted to *N 2 \*1* dimensions. A training set of *N 2 \*M* dimensions is thus created, where M is the number of sample images.



**Fig. 3:** Representation of image *In* as a vector *rn* where *r = P (Height= number of pixels in an Image) \* N (Width = number of Image)*

* *Compute the average of r image vector:*

The average of the image set is calculated as:

*Ψ = ri..*

where, *Ψ = Average image,*

*M = Number of images,*

*ri = Image vector*

The eigenfaces corresponding to the *highest eigenvalues* are retained. Those eigenfaces define the face space. The eigenspace is created by projecting the image to the face space formed by the eigenfaces. Thus the *weight vectors* are calculated. Dimensions of the image are adjusted to meet the specifications and the image is enhanced in the preprocessing steps of recognition. The weight vector of the image and the weight vectors of the faces in the database are compared.

* *Subtract average face from each face:* Average face is calculated and subtracted from each face in the training set. A *matrix (A)* is formed using the results of the subtraction operation. The difference between each image and the average image is calculated as :

*Фi = ri - Ψ. , i = 1,2….., M*

where, *Фi is the difference between the image and average image*.

* *Computing the covariance matrix:*

The matrix obtained by the s*ubtraction operation (A)* is multiplied by *its transpose* and thus *covariance matrix C* is formed:

*C = ATA.*

where, *A is formed by the difference vectors, i.e.,*

*A = {Ф1 ,Ф2 ,Ф3 …...ФM }*

So, *C = Ф \* ФT >>> (P∗N) ∗ (N∗P) = (P∗P)*

The dimensions of the *matrix C* is *N \* N*. *M images* are used to form *C*. In practice, the dimensions of *C* is *N \* M*.

* *Calculate the eigenvalues and sort them by using a sorting algorithm:*

The eigenvalues of the covariance matrix are calculated and then ordered using the *Quick Sort*, which is a *Divide and Conquer algorithm*. It picks an element as pivot and partitions the given array around the picked pivot.

The key process in quickSort is *partition()*. Target of partitions is given an array and an *element x* of array as pivot, put *x* at its correct position in sorted array and put all → smaller elements *(smaller than x)* before *x*, and put all greater elements *(greater than x)* after *x*. We need the best eigenvalues or important eigenvalues.

To calculate eigenvalues we have two alternatives :

* Using *Power method* to calculate the dominant eigenvalue and then try to obtain the eigenvalues closest to *p* using *inverse power method*.

Let *λk* is the (unknown) *eigenvalue of A* closest to *p*. *λ1, λ2 , . . . , λn* are the eigenvalues of *A*, then *λ1 - p , λ2 - p, . . . , λn - p* are the eigenvalues of the matrix *A - pI*. Then *λk - p* will be the smallest magnitude eigenvalue of *A - pI* but will be the largest magnitude eigenvalue of *(A − pI)-1* . Hence if we apply the power method to *(A − pI)-1* we can obtain *λk.*

* Using *QR Technique* for obtaining all the eigenvalues of a large order matrix. This technique is based on similarity transformations i.e. transformations of the form:

*B = M - 1 A M.*

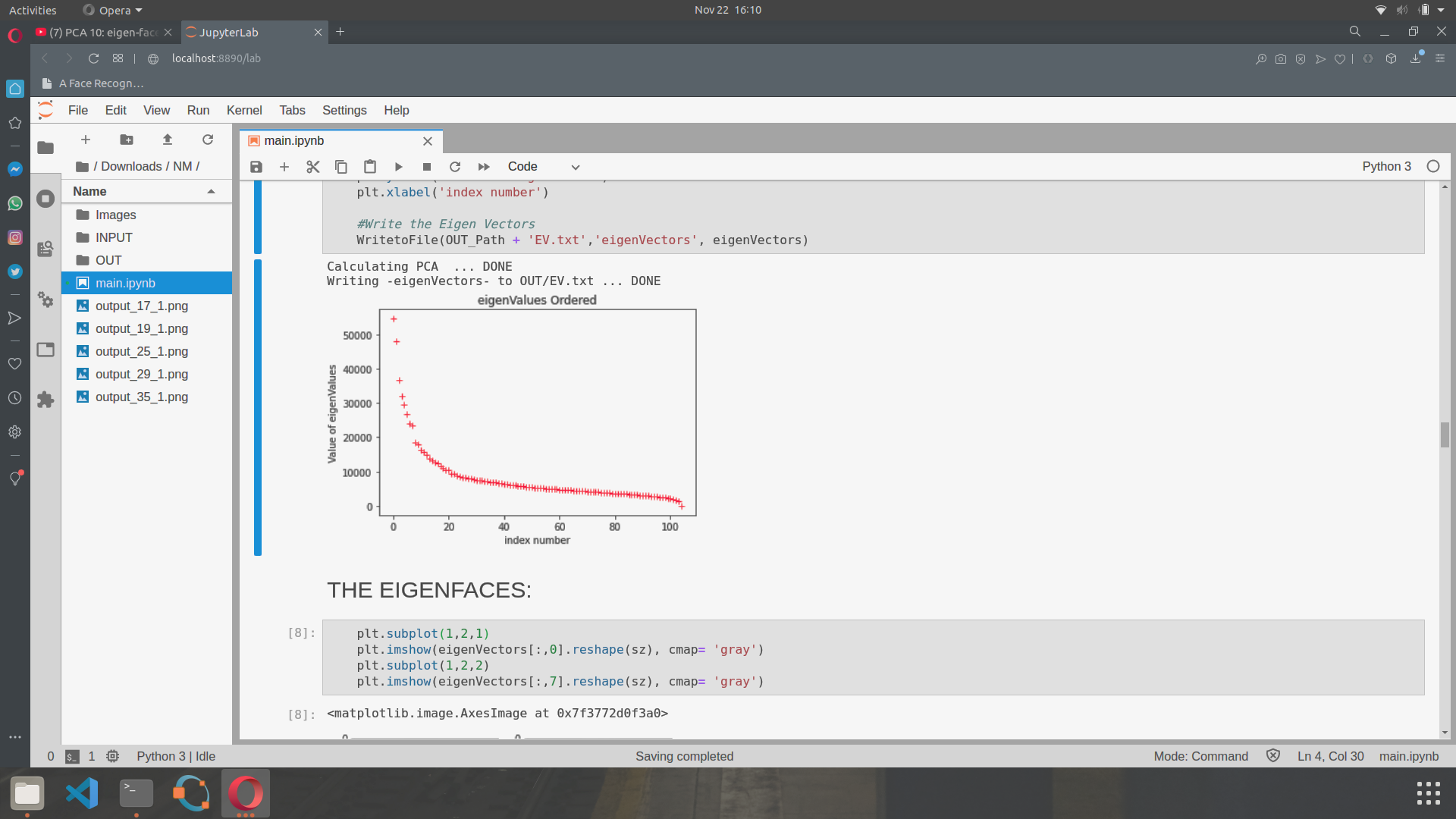
where, *B has the same eigenvalues as A*.

In the QR method *A* is reduced to *upper (or lower) triangular form* and in a triangular matrix, eigenvalues are at the diagonal..

* *Computing the Eigenvectors corresponding to eigenvalues:*

Computing the Eigenvectors of the matrix *C = Ф \* ФT*is very large and is not practical. As the size of the above matrix is huge it's very expensive to compute eigenvectors of such a huge matrix. So, instead we will calculate the eigenvectors of *C1 = Ф \* ФT* as the matrix is small of *N x N* dimension. Then we will pre-multiply the eigenvectors of *C1 = Ф \* ФT* with *Ф* to get eigenvectors of *C1 = Ф \* ФT*.

Since the rank of *A* is *M*, *only M out of N eigenvectors are nonzero*. The eigenvalues of the covariance matrix is calculated. The eigenfaces are created by using the *number of training images minus number of classes* (total number of people) of eigenvectors. The selected set of eigenvectors are multiplied by the A matrix to create a reduced eigenface subspace. The eigenvectors of smaller eigenvalues correspond to smaller variations in the covariance matrix. The discriminating features of the face are retained. The number of eigenvectors depend on the accuracy with which the database is defined and it can be optimized. The group of selected eigenvectors are called the eigenfaces.



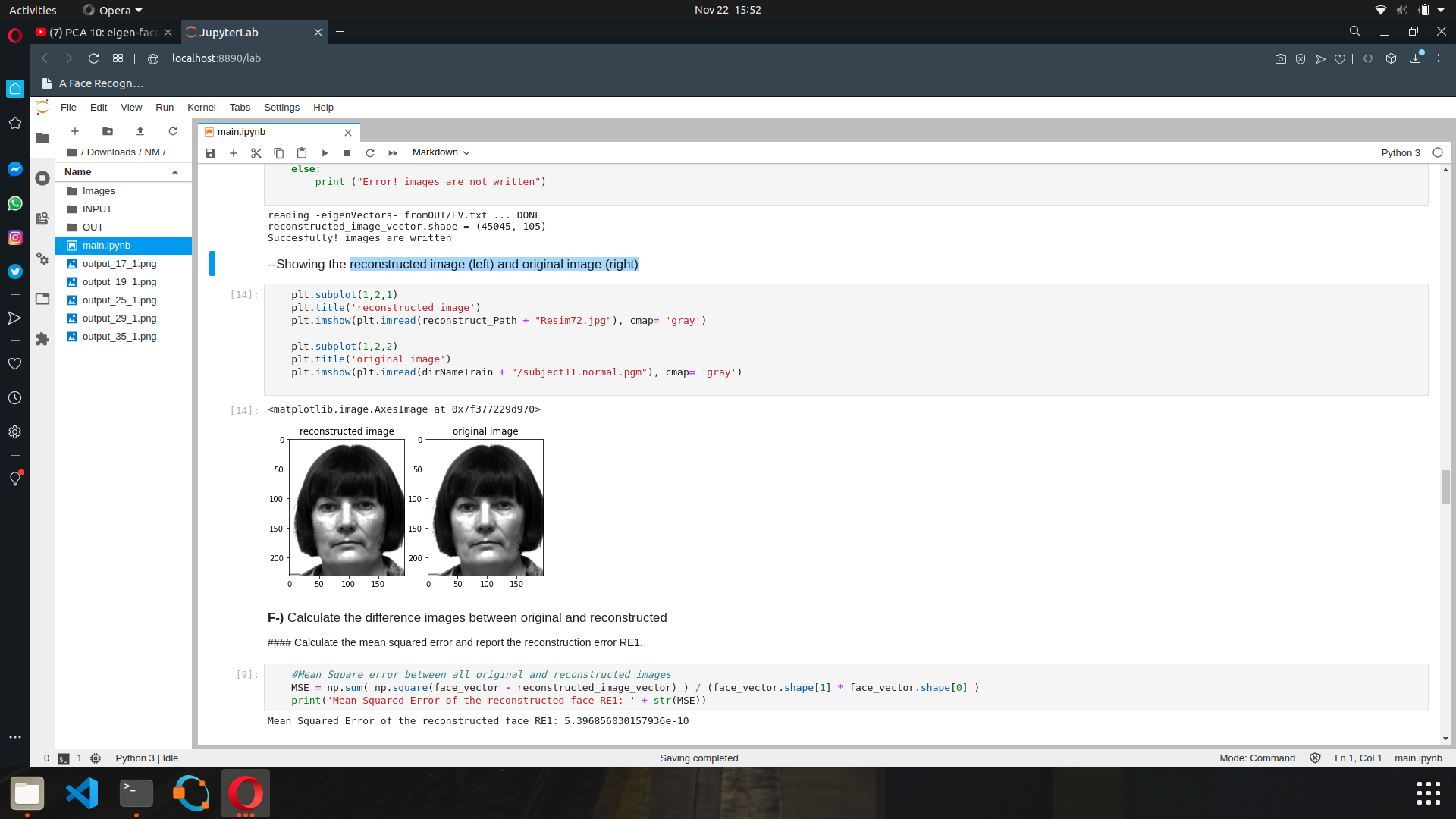
**Fig. 4:** Ordered eigenvalues of Covariance matrix C

* *Choose the best k eigenvectors and find weights of each image and store it:*

Once the eigenfaces have been obtained, the images in the database are projected into the *eigenface space* and the weights of the image in that space are stored.

#### *Reconstruct N face images as a linear combination of bases of the space 'V’:*

We reconstruct the images using the eigenvectors and weights we stored.



**Fig. 5:** Example of reconstructed image (left) and original image (right).

#### *Calculate the difference between original and reconstructed (RE1):*

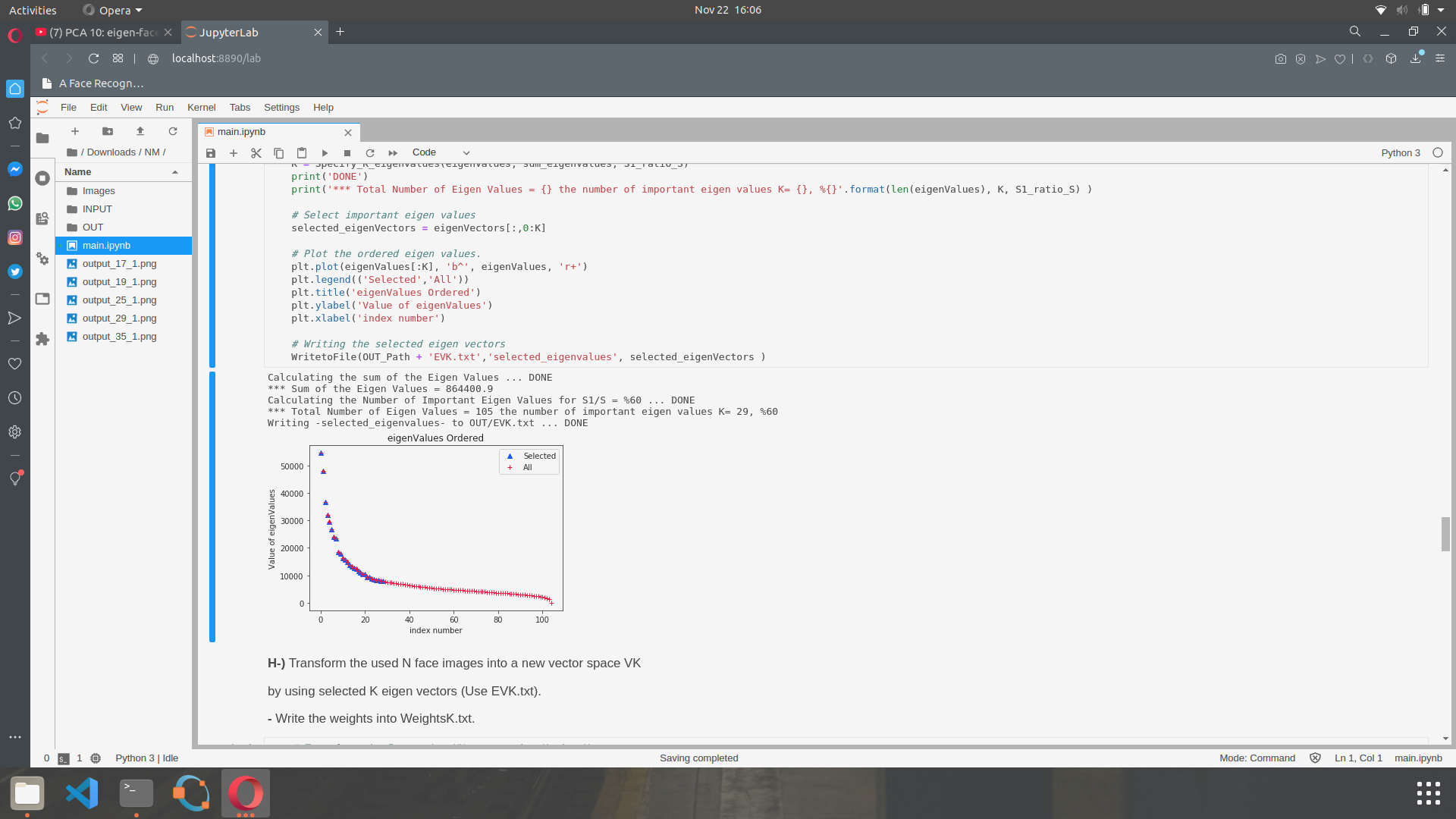
We calculate the *Mean Square Error* between all original and reconstructed images. In this project, the *Mean Squared Error* of the reconstructed face comes out to be *5.396856030157936 e-10.*

#### *Calculate sum of E eigenvalues as S :*

#### Next, we calculate *K, the number of important eigenvalues* by checking the ratio of:

#### *()*

#### where, *S1 denotes sum of the used eigenvalues*.

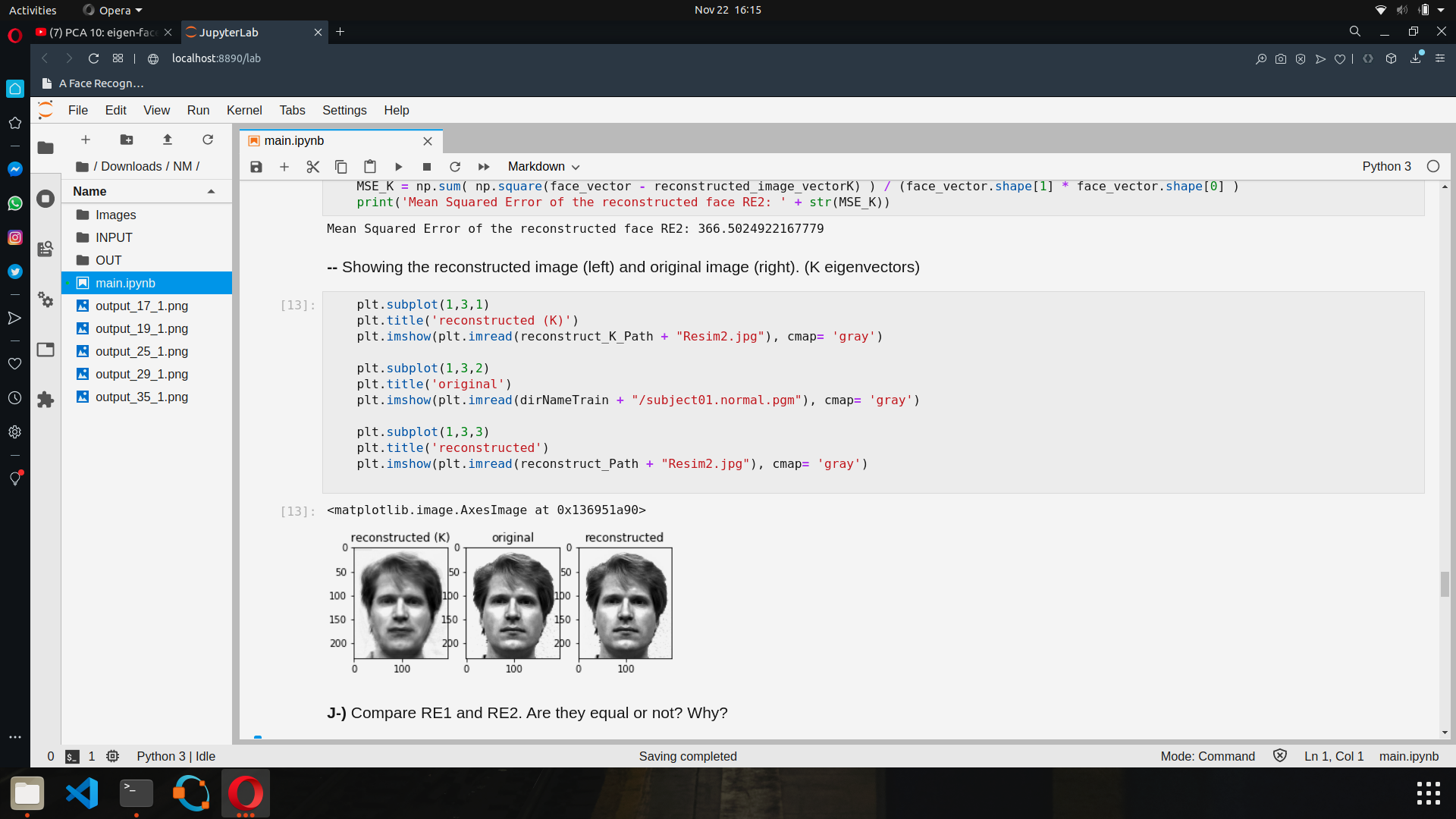


**Fig. 6:** Important eigenvalues

#### *Transform the used N face images into a new vector space VK by using K eigenvectors*

* *Again reconstruct the images and calculate the mean squared error RE2:*

Mean Squared Error of the reconstructed face *RE2* comes out to be *366.5024922167779*. *RE1*and *RE2*are not equal because of their eigenvectors. We choose the best (higher) eigenvalues and we use the eigenvectors corresponding to these eigenvalues but still there is a little bit of information in the unselected eigenvectors.



**Fig. 7:** Reconstructed image (left) and original image (center) for K eigenvectors

#### *Read T face images from the test set and transform them into vector space V*

#### *Calculate the Euclidean distance between each of T test images and N training images:* To determine the identity of an image, the *eigen-coefficients* are compared with the *eigen-coefficients* in the database. The eigenface of the image in question is formed. The *Euclidean distances* between the eigenface of the image and the eigenfaces stored previously are calculated. The person in question is identified as the one whose *Euclidean distance is minimum* below a threshold value in the eigenface database. If all the calculated *Euclidean distances* are larger than the threshold, then the image is unrecognizable.

D =

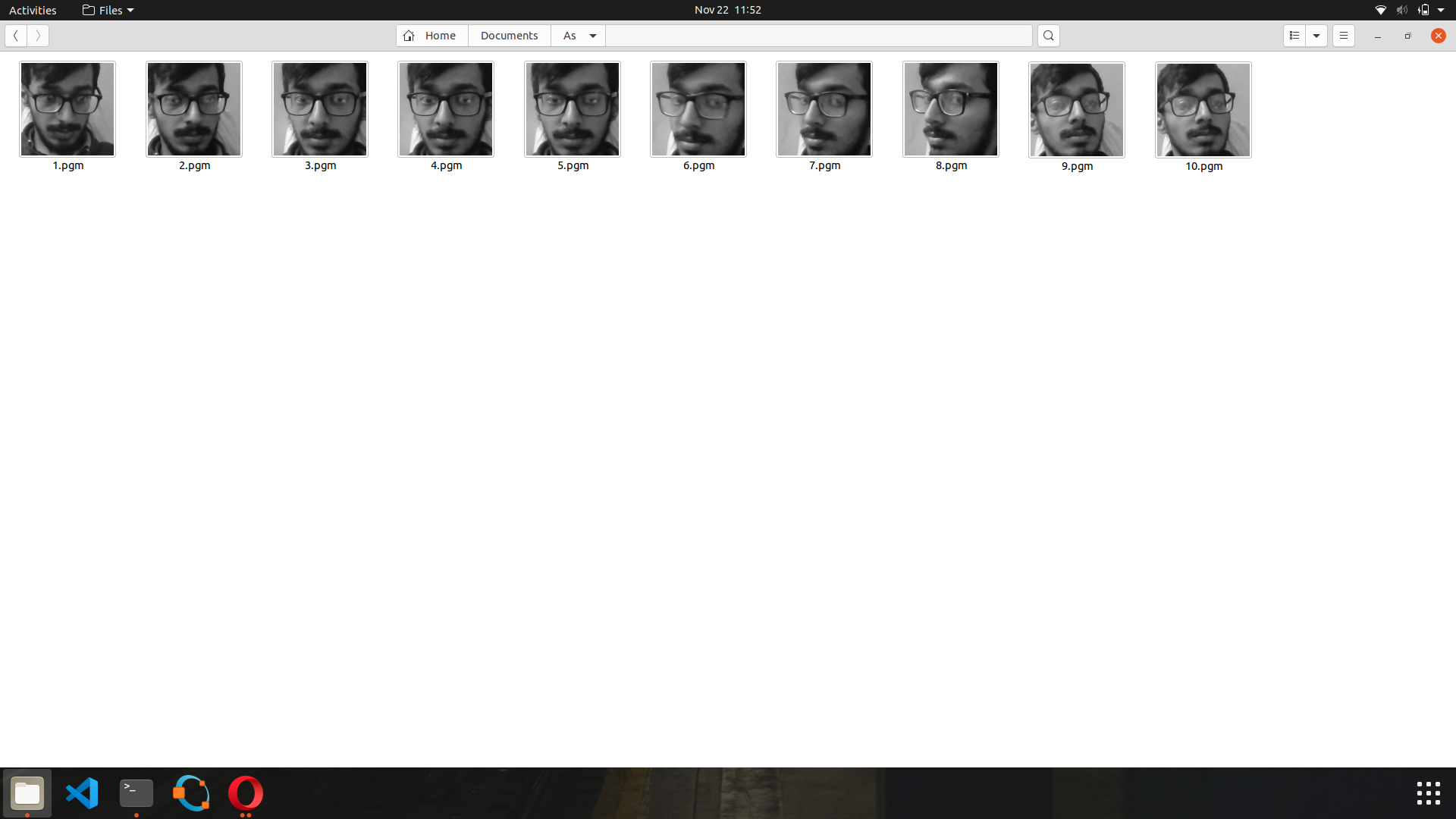
# RESULTS

The database used in this work consists of 105 images of 30 people. A total of 105 images are used. The average face is calculated using the training set. In the below figure, some images of the training set are shown.



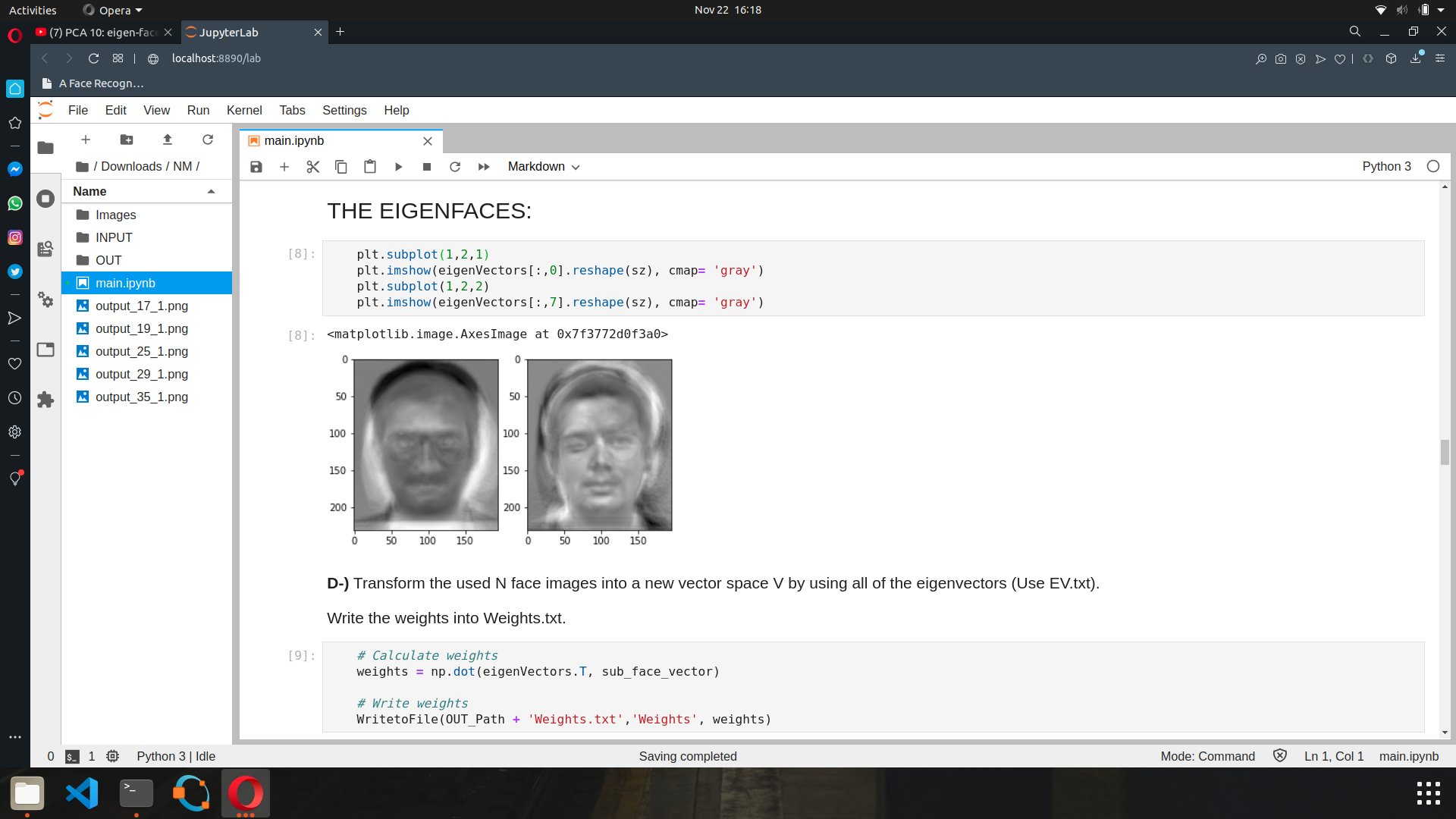
**Fig. 8:** Some Images in Training set

In the figure below, images belonging to the same person are shown. Ten pictures with various facial expressions, distances and lighting conditions are taken. The program processes all of them to recognize the person.



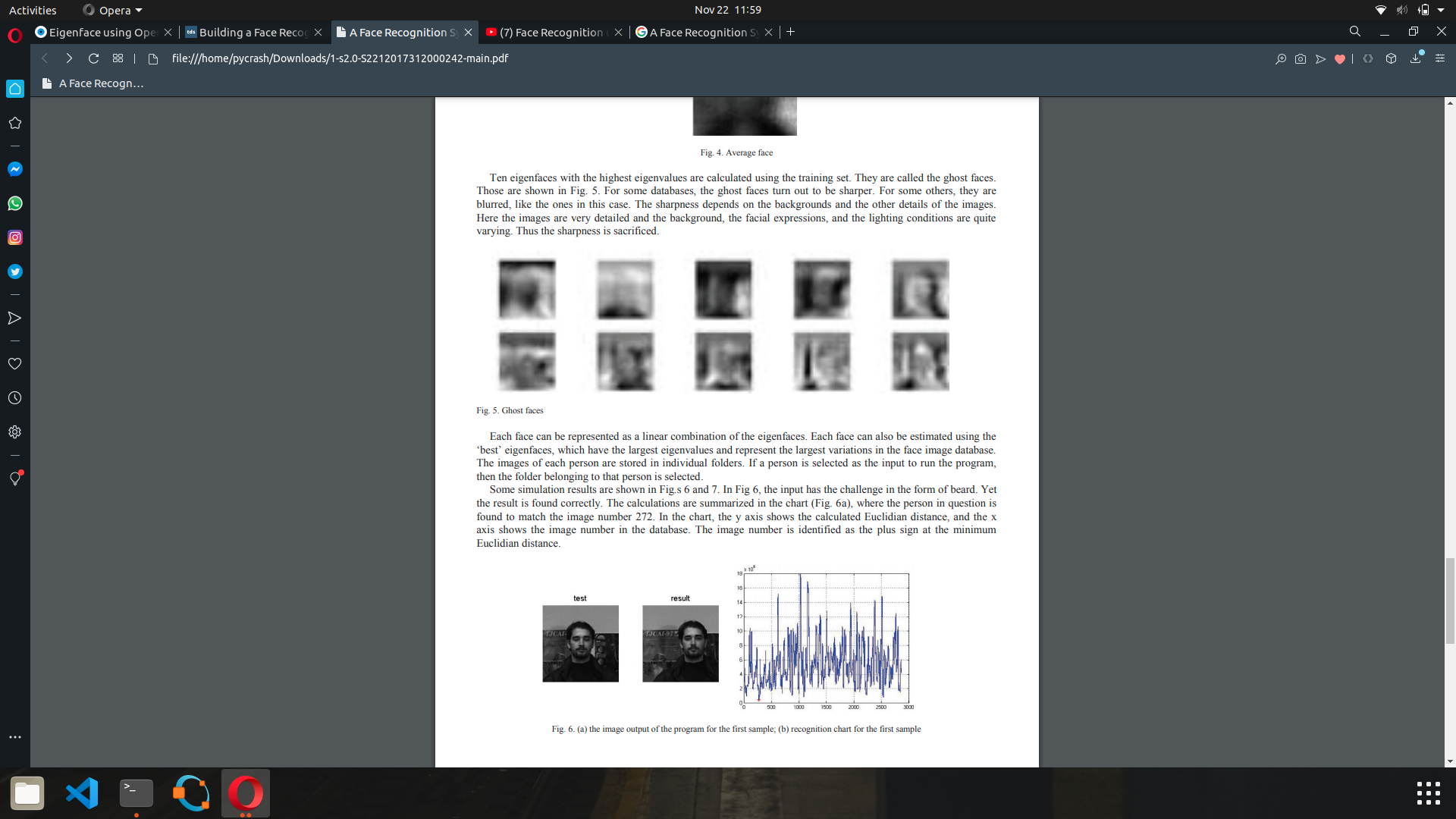
**Fig. 9 :** Images of a single person in the training dataset

Eigenfaces for the given training set comes out to be:



**Fig. 10:** Eigenfaces for the given database

Ten eigenfaces with the highest eigenvalues are calculated using the training set. They are called the *ghost faces*. Those are shown in the figure below. For some databases, the ghost faces turn out to be sharper. For some others, they are blurred, like the ones in this case. The sharpness depends on the backgrounds and the other details of the images. Here the images are very detailed and the background, the facial expressions, and the lighting conditions are quite varying. Thus the sharpness is sacrificed.



# Fig. 11: Ghost Faces

Each face can be represented as a linear combination of the eigenfaces. Each face can also be estimated using the *‘best’* eigenfaces, which have the largest eigenvalues and represent the largest variations in the face image database.

We performed our analysis on the training and test set and the best number of correctly matched faces came out to be *29 out of 30*.

**Table 2:** Results of analysis

| ***TPK - No. of correctly matched Images*** | ***Accuracy*** | ***(No. of important eigenvalues)*** | ***K (Test Weights)*** |
| --- | --- | --- | --- |
| *29 out of 30* | *97%* | *80%* | *57* |
| *29 out of 30* | *97%* | *70%* | *41* |
| *29 out of 30* | *97%* | *60%* | *29 (Selected)* |
| *28 out of 30* | *93%* | *50%* | *19* |
| *24 out of 30* | *80%* | *30%* | *08* |
| ***Total Eigenvalues (S):*** *105* | | | |

For our training set, the accuracy must come out *100%* because the system has already trained the same images and also it used all of the eigenvectors. On the other hand; the system doesn't see the *'test set'*, so the accuracy depends on how much the system learns the face features. We were able to achieve “*97% accuracy”* for the given test set.

# CONCLUSION

The *Eigenfaces method* is applied to a database consisting of *105 images*. The challenging details, such as background, eye-glasses, beard, mustache are dealt with. The success rate is calculated as *100% for the training set* and *97% for the test set*. From the results, it can be concluded that, for recognition, it is sufficient to take about 2*9% eigenfaces* with the highest eigenvalues. It is also clear that the recognition rate increases with the number of training images per person.

It is obvious that if the minimum distance between the test image and other images is zero, the test image entirely matches the image from the training base. If the distance is greater than zero but less than a certain threshold, it is a *known person* with another facial expression, otherwise it is an *unknown person*.

To increase the success rate, the eigenfaces method can be fortified with the use of additional information, such as the *face triangle*. Further work can be done in future focussing on increasing success rate for very large databases.

# REFERENCES

1. Slavković, Marijeta & Jevtić, Dubravka. (2012). Face Recognition Using Eigenface Approach\*. Serbian Journal of electrical engineering. 9. 121-130. 10.2298/SJEE1201121S.
2. Eigenface Method

From: <https://en.wikipedia.org/wiki/Eigenface>

<http://www.scholarpedia.org/article/Eigenfaces>

<https://www.geeksforgeeks.org/ml-face-recognition-using-eigenfaces-pca-algorithm/>

<https://scikit-learn.org/stable/auto_examples/applications/plot_face_recognition.html>

1. Eigenfaces : Recovering Humans from Ghosts

From::<https://towardsdatascience.com/eigenfaces-recovering-humans-from-ghosts-17606c328184>

1. Ngo, D.C.L., Teoh, A.B.J., Goh, A.: Biometric Hash: High-Confidence Face Recognition. IEEE Trans. Circuits and Systems for Video Techn. 16(6), 771–775 (2006)
2. Turk, M., Pentland, A.: Eigenfaces for recognition. Journal Cognitive Neuroscience 3(1), 71–86 (1991)
3. Belhumeur, et al.: Eigenfaces vs. Fisherfaces. Recognition Using Specific Linear Projection PAMI 19, 711–720 (1997)
4. Yang, M.H., Ahuja, N., Kriegman, D.: Face Recognition Using Kernel Eigenfaces. In: IEEE ICIP 2000, pp. 37–40. IEEE, Los Alamitos (2000)
5. B. Karaduman, Relevant Component Analysis, M. S. Thesis, Yıldız Technical University, Turkey, 2008
6. F. S. Samaria and A. C. Harter, Parameterization of a Stochastic Model for Human Face Identification, Proc. of the 2nd IEEE Workshop on Applications of Computer Vision, Sarasota, Florida, 1994
7. M. Turk and A. Pentland, “Eigenfaces for Recognition,” Journal of Cognitive Neuroscience, Vol. 3, No. 1 (1991) 71-86.
8. A. Samal and P. A. Iyengar, "Automatic recognition and analysis of human faces and facial expressions: A survey", *Pattern Recognit.*, vol. 25, pp. 65-77, 1992.
9. Y. Vijaya Lata, Chandra Kiran Bharadwaj Tungathurthi, H. Ram Mohan Rao, A. Govardhan and L.P. Reddy, "Facial Recognition using Eigenfaces by PCA", *International Journal of Recent Trends in Engineering*, vol. 1, no. 1, May 2009.

# APPENDIX

* *Face recognition system code (using Python 3.8.5 & Jupyter Notebook 2.2.6):*

Gdrive folder link: <https://drive.google.com/drive/folders/14oQCKIKNz8gvHemFnfYPY3_L6lm4Sg4R?usp=sharing>

#Setting Up

import os

import sys

import cv2

import numpy as np

import matplotlib

import matplotlib.pyplot as plt

%matplotlib inline

dirNameTrain = 'INPUT/TRAIN'

dirNameTest = 'INPUT/TEST'

OUT\_Path = 'OUT/'

reconstruct\_Path = 'OUT/Reconstruct\_All\_eigen/'

reconstruct\_K\_Path ='OUT/Reconstruct\_K\_eigen/'

# Ratio, Selecting best eigen vectors

S1\_ratio\_S = 60

# Select best K eigenValues

def Specify\_K\_eigenvalues(eigenValues, sum\_eigenValues, percent):

for i in range(0, len(eigenValues)):

if ( ( np.sum(eigenValues[:i]) / sum\_eigenValues ) \* 100 ) > percent :

return i

return None

# Convert the face vector to images and then Write the images

def WriteImages(facevectors, size, path):

for i in range(0, facevectors.shape[1]):

image = facevectors[:,i].reshape(size)

image = image.astype(int)

cv2.imwrite(path + 'Resim' + str(i) + '.jpg', image)

return True

# Read a txt file as matrix

def ReadFromFile(filename, string):

print('reading -' + str(string) + '- from' + str(filename), end=' ... ', flush=True)

file = cv2.FileStorage(filename, cv2.FILE\_STORAGE\_READ)

matrix = file.getNode(string).mat()

file.release()

print('DONE')

return matrix

# Save a matrix as txt

def WritetoFile(filename, string, matrix):

print('Writing -' + str(string) + '- to ' + filename, end=' ... ', flush=True)

file = cv2.FileStorage(filename, cv2.FILE\_STORAGE\_WRITE)

file.write(str(string), matrix)

file.release()

print('DONE')

# Find eigenValues and eigenVectors of a Matrix

def Find\_Eigen(matrix) :

eigenVectors , eigenValues , \_ = np.linalg.svd(matrix, full\_matrices = False )

sort = eigenValues.argsort()[::-1]

eigenValues = eigenValues[sort]

eigenVectors = eigenVectors[:,sort]

return eigenValues,eigenVectors

# Create Data Matrix which keeping the images as a single column

def CreateDataMatrix(images):

print('Creating data matrix',end=' ... ', flush=True)

numImages = len(images)

sz = images[0].shape

data = np.zeros(( sz[0] \* sz[1], numImages), dtype=np.float32)

for i in range(0, numImages):

image = images[i].flatten()

data[:,i] = image

print('DONE')

return data

# Read all of the Images in a folder

def ReadImages(path):

print('Reading images from ' + path, end=' ... ', flush=True)

# Create array of array of images.

images = []

# List all files in the directory and read points from text files one by one

for filePath in sorted(os.listdir(path)):

fileExt = os.path.splitext(filePath)[1]

if fileExt in ['.jpg', '.jpeg', '.pgm']:

# Add to array of images

imagePath = os.path.join(path, filePath)

im = cv2.imread(imagePath)

im = cv2.cvtColor(im, cv2.COLOR\_BGR2GRAY)

if im is None :

print('image:{} not read properly'.format(imagePath))

else :

# Convert image to floating point

im = np.float32(im)

# Add image to list

images.append(im)

# Flip image

#imFlip = cv2.flip(im, 1);

# Append flipped image

#images.append(imFlip)

numImages = len(images) / 2

# Exit if no image found

if numImages == 0 :

print('No images found')

sys.exit(0)

print(str(numImages) + ' files read.')

return images

# Read images

train\_images = ReadImages(dirNameTrain)

# Create face matrix for PCA.

face\_vector = CreateDataMatrix(train\_images)

# Size of images

sz = train\_images[0].shape

number\_train\_images= face\_vector.shape[1]

# Find average\_face\_vector, sum(all image vectors)/number(images).

average\_face\_vector = np.mean(face\_vector, axis=1)

average\_face\_vector.shape = (len(average\_face\_vector), 1)

# Subtract average\_face\_vector from every image vector.

sub\_face\_vector = np.zeros(face\_vector.shape, dtype=np.float32)

sub\_face\_vector = face\_vector - average\_face\_vector

# Calculate covariance matrix of above matrix -> C = A\*transpose(A)

covariance\_matrix = np.dot(sub\_face\_vector.T, sub\_face\_vector)

covariance\_matrix /= number\_train\_images

print("covariance\_matrix.shape =", covariance\_matrix.shape)

# Find eigenvectors and eigenvalues of above covariance matrix.

# eigenvalues arranged to match with associated eigenvector

print('Calculating PCA ', end=' ... ', flush=True)

eigenValues, eigenVectors = Find\_Eigen(sub\_face\_vector)

print ('DONE')

# Plot the ordered eigen values.

plt.plot(eigenValues, 'r+')

plt.title('eigenValues Ordered')

plt.ylabel('Value of eigenValues')

plt.xlabel('index number')

#Write the Eigen Vectors

WritetoFile(OUT\_Path + 'EV.txt','eigenVectors', eigenVectors)

plt.subplot(1,2,1)

plt.imshow(eigenVectors[:,0].reshape(sz), cmap= 'gray')

plt.subplot(1,2,2)

plt.imshow(eigenVectors[:,7].reshape(sz), cmap= 'gray')

# Calculate weights

weights = np.dot(eigenVectors.T, sub\_face\_vector)

# Write weights

WritetoFile(OUT\_Path + 'Weights.txt','Weights', weights)

# Read eigenVectors from file

eigenVectors = ReadFromFile(OUT\_Path + 'EV.txt', 'eigenVectors')

# Reconstruct from 'V' Space

reconstructed\_image\_vector = average\_face\_vector + np.dot(eigenVectors, weights)

print('reconstructed\_image\_vector.shape =', reconstructed\_image\_vector.shape)

# Write Reconstructed Images

Is\_Okey = WriteImages(reconstructed\_image\_vector, sz, reconstruct\_Path)

if Is\_Okey :

print ("Succesfully! images are written")

else:

print ("Error! images are not written")

plt.subplot(1,2,1)

plt.title('reconstructed image')

plt.imshow(plt.imread(reconstruct\_Path + "Resim72.jpg"), cmap= 'gray')

plt.subplot(1,2,2)

plt.title('original image')

plt.imshow(plt.imread(dirNameTrain + "/subject11.normal.pgm"), cmap= 'gray')

#Mean Square error between all original and reconstructed images

MSE = np.sum( np.square(face\_vector - reconstructed\_image\_vector) ) / (face\_vector.shape[1] \* face\_vector.shape[0] )

print('Mean Squared Error of the reconstructed face RE1: ' + str(MSE))

# sum of eigen values

print('Calculating the sum of the Eigen Values', end=' ... ', flush=True)

sum\_eigenValues = np.sum(eigenValues)

print('DONE')

print('\*\*\* Sum of the Eigen Values = ' + str(sum\_eigenValues))

# Calculate the important eigen values

print('Calculating the Number of Important Eigen Values for S1/S = %' + str(S1\_ratio\_S), end=' ... ', flush=True)

K = Specify\_K\_eigenvalues(eigenValues, sum\_eigenValues, S1\_ratio\_S)

print('DONE')

print('\*\*\* Total Number of Eigen Values = {} the number of important eigen values K= {}, %{}'.format(len(eigenValues), K, S1\_ratio\_S) )

# Select important eigen values

selected\_eigenVectors = eigenVectors[:,0:K]

# Plot the ordered eigen values.

plt.plot(eigenValues[:K], 'b^', eigenValues, 'r+')

plt.legend(('Selected','All'))

plt.title('eigenValues Ordered')

plt.ylabel('Value of eigenValues')

plt.xlabel('index number')

# Writing the selected eigen vectors

WritetoFile(OUT\_Path + 'EVK.txt','selected\_eigenvalues', selected\_eigenVectors )

# Transform the Images in VK space using K eigenVectors

weightsK = np.dot(selected\_eigenVectors.T, sub\_face\_vector)

# Writes the weights

WritetoFile(OUT\_Path + 'WeightsK.txt','WeightsK', weightsK)

# Reconstruct from 'VK' Space

reconstructed\_image\_vectorK = average\_face\_vector + np.dot(selected\_eigenVectors, weightsK)

# Write reconstructed images

WriteImages(reconstructed\_image\_vectorK, sz, reconstruct\_K\_Path)

#Calculate Mean Square Error

MSE\_K = np.sum( np.square(face\_vector - reconstructed\_image\_vectorK) ) / (face\_vector.shape[1] \* face\_vector.shape[0] )

print('Mean Squared Error of the reconstructed face RE2: ' + str(MSE\_K))

plt.subplot(1,3,1)

plt.title('reconstructed (K)')

plt.imshow(plt.imread(reconstruct\_K\_Path + "Resim2.jpg"), cmap= 'gray')

plt.subplot(1,3,2)

plt.title('original')

plt.imshow(plt.imread(dirNameTrain + "/subject01.normal.pgm"), cmap= 'gray')

plt.subplot(1,3,3)

plt.title('reconstructed')

plt.imshow(plt.imread(reconstruct\_Path + "Resim2.jpg"), cmap= 'gray')

# Read test images

test\_images = ReadImages(dirNameTest)

# Create face matrix for PCA.

test\_face\_vector = CreateDataMatrix(test\_images)

# Find average\_face\_vector, sum(all image vectors)/number(images).

average\_test\_face\_vector = test\_face\_vector - average\_face\_vector

# Read the eigenVectors from EV.txt

eigenVectors = ReadFromFile(OUT\_Path + 'EV.txt', 'eigenVectors')

#Transform the test images to 'V' space

weights\_test = np.dot(eigenVectors.T, average\_test\_face\_vector)

#Write Test Weights

WritetoFile(OUT\_Path + 'WeightsT.txt','WeightsT', weights\_test)

# Create array to keep matched images

Matched\_Faces = np.zeros((weights\_test.shape[1]))

# Compare all weights\_test among the weights and Calculate Similarity

for i in range(0, weights\_test.shape[1]):

error = np.zeros((weights.shape[1]))

for j in range (0, weights.shape[1]):

error[j] = (np.sum((weights[:, j] - weights\_test[:, i])\*\*2))

# Match the test image with training image

Matched\_Faces[i] = error.argmin() // 7

# Print the indexes of matched images

print(Matched\_Faces)

#Transform the test images to 'VK' space

weights\_test\_K = np.dot(selected\_eigenVectors.T, average\_test\_face\_vector)

#Write Test Weights K

WritetoFile(OUT\_Path + 'WeightsTK.txt','WeightsTK', weights\_test\_K)

# Create array to keep matched images

Matched\_Faces\_K = np.zeros((weights\_test\_K.shape[1]))

# Compare all weights\_test\_K among the weights\_K and Calculate Similarity

for i in range(0, weights\_test\_K.shape[1]):

error\_K = np.zeros((weights.shape[1]))

for j in range (0, weights.shape[1]):

AAA = np.pad(weights\_test\_K[:, i], (0, weights.shape[0] - weights\_test\_K.shape[0]), 'constant')

error\_K[j] = (np.sum((weights[:, j] - AAA)\*\*2))

# Match the test image with training image

Matched\_Faces\_K[i] = error\_K.argmin() // 7

# Print the indexes of matched images

print(Matched\_Faces\_K)

# We already have training weights in 'weights'

# Create array to keep matched images

Matched\_Faces\_training = np.zeros((weights.shape[1]))

for i in range(0, weights.shape[1]):

error\_K = np.zeros((weights.shape[1]))

for j in range (0, weights.shape[1]):

AAA = np.pad(weights[:, i], (0, weights.shape[0] - weights.shape[0]), 'constant')

error\_K[j] = (np.sum((weights[:, j] - AAA)\*\*2))

# Match the test image with training image

Matched\_Faces\_training[i] = error\_K.argmin() // 7

# Print the indexes of matched images

print(Matched\_Faces\_training)