Face Recognition using Eigenfaces

Aneri Dalwadi AU1940153
Malav Doshi AU1940017
Parth Shah AU1940065
Samkit Kundalia AU1940021

Abstract

This project focuses on identifying the recognised face from the provided training set. This project uses the concepts of Singular Value decomposition and Principal Component Analysis for dimensionality reduction, feature extraction, precise data visualisation and finding a correlation between the same set of different images. Our group has tried to develop a system to recognise a set of images of the same person and find correlation between the images to identify a person.

Introduction

The human face is an extremely complex structure with unique features. Face recognition is nowadays widely used in smartphones and more efficient algorithms are still under development for recognising the face more precisely. Here the most common method is via Eigenface . Eigenface is an appearance-based to face recognition where different features and variations are captured from a collection of face images.

Now, as face is such a complex structure, it becomes very difficult to capture all features of faces. Hence we use Principal component analysis (PCA) to overcome this problem. PCA is a method to find a set of projection vectors such that those vectors contain most of the information about original data. Hence reduces the dimensionality by projecting a M-dimensional space to a P-dimensional space, where P≤M.

The main motto behind our project is not only to understand how eigenfaces and PCA play a role in facial recognition but also to implement the same in form of a python code.

The need for Facial Recognition

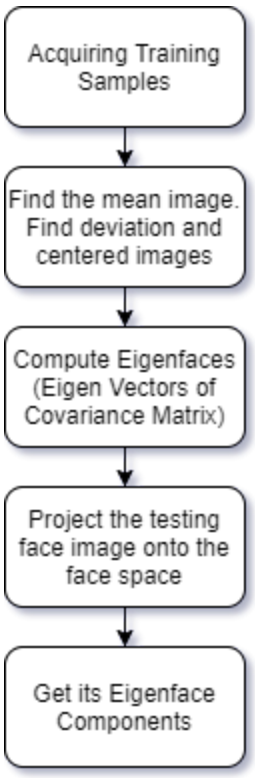
Nowadays, there is a significant increase in Electronics transactions. There is a greater demand for fast and accurate user identification and authentication. Other forms of passcodes can be stolen such as PINs. But this problem can be solved by facial recognition as it is undeniably connected to the owner.

This technology can also be used by law enforcements to find missing people or criminals.It has also found some really interesting applications in the field of medicine such as detection of genetic disorders.

It is also better than other forms of biometrics as it requires no physical interaction, can use existing hardware, allows passive identification in any environment and also it requires no expert to interpret the results.

Nowadays, facial recognition is also used for attendance purposes in School and Universities to make sure students aren’t skipping classes.

Approach



Every face is a data matrix. Consider a grayscale image, here every pixel is a number which can be one if it's white , zero if black and anything in between it is grey. Hence our primary focus to input a dataset of images of a person whose face we need to recognise.

Now the most common approach as discussed above is recognition with eigenfaces.

Eigenfaces are the Eigenvectors of the covariance matrix and are often referred to as ghostly images. The prime reason to approach this methodology is that it helps us represent the input data more efficiently. Here, each individual face is represented as a linear combination of eigen faces.

Now let's consider a dataset which contains *k* different persons, and there are multiple images of the same person and dimension for each image is *NxN*. Here every image from every sub-dataset can be categorized as different as it contains images at face from different angles or face.

First we convert these images into vector of *N^2* Which will be in turn a column vector with dimension *N^2x1*.

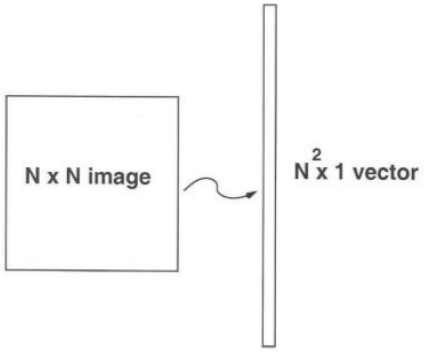


Figure 1

Now we find a mean faces from image of every person from the images given in dataset ,which is given by

$$\psi = \frac{1}{m} \sum_{i=1}^{m} x_i$$

Now this mean image would also be a column vector of dimension N^2x1, as we will have k such mean faces. We get a matrix with k columns and N^2 rows which consists of column vectors which are nothing but the mean faces.

Our next task is Eigenvalues and vectors of the covariance matrix. But before we do that, we need to understand why SVD here.

Singular Value Decomposition (SVD)

Now before we apply Principal Component Analysis (PCA), we need to find the value of principal components and we do that by using SVD. A singular value decomposition of a matrix is expressed as: $A = U\Sigma V^T;$

Where U and V are orthogonal matrices and matrix Σ is a diagonal matrix, which consists of non-negative singular values σ_i

Where, The singular values are placed in Σ in descending order such as

$\sigma 1 \geq \sigma 2 \geq \cdots \geq \sigma p \geq 0 where p = min(n,m)$.

Covariance-Matrix

Now that we know SVD, we should understand that PCA is done by decomposition of the Covariance matrix.

Typically covariance matrix is,

$$C_1 = A^T A$$

Now Covariance matrix can be $C_1 = AA^T$ but if we consider the latter, clearly our Covariance matrix would be very large as it will have dimension of $N^2 x N^2$. But if we consider our covariance matrix as $A^T A$, we see that the matrix only depends on the number of people in our dataset i.e k. Hence we get a matrix of dimensionality *kxk*.

Eigenfaces

Now that we have got the covariance matrix, we need to find the Eigenfaces which are nothing but the eigenvectors of the covariance matrix. The main motivation behind this Eigenfaces is to

- Extract relevant facial information.
- It also helps to represent the face images more efficiently.
- It also helps reduce time and space complexity i.e each face can be represented with a small number of parameters.

Thus each image is approximated using a subset of EigenValues.

Principal Component Analysis (PCA)

PCA seeks to find a principal axes which can be defined as an orthonormal coordinate system which can capture most of the variance in the data. In PCA, every image is represented as the linear combination of weighted eigenvectors and eigenvalues which we obtained from the covariance matrix. The advantage of PCA is that it can find patterns between the variables in the data and compress the data, without much loss of information by reducing the number of dimensions .

The first principal component is the linear combination with maximum variance and for the n^{th} principal component, it is a linear combination with highest variance and is orthogonal to the first n-1 principal components.

Libraries used in the code

We have used the least number of libraries and only used them until really necessary.

We have used the following libraries:

- NumPy: We have used the Numpy library for array declarations and reshaping.
- MatPlot: We have used the Matplot library to plot a matrix as an image.
- Pillow: We have used this library to open an image and convert it to grayscale.
- Math: This library is used to create basic mathematical operations such as trigonometric functions, square root functions and to find absolute values.

Approach of the Code

Firstly, we use the Image module from Pillow library to open the image. Clearly, the information stored per pixel would be dramatically reduced once we convert the image to grayscale. We do so by using the ImageOps module. We convert it to a b column matrix of size N^2x1, from NxN.
The columns of the b matrix contain the images represented in their pixel form. Now, given the matrix 'b', we find the mean matrix and label it bMean. Then, we find covariance matrix with the formula $b^T * b$.
We then find the eigenValues and eigenVectors using Jacobi's method
This matrix of eigenVectors of the covariance matrix is known as eigenFace matrix. Using this eigenFace matrix, we find projection of each image into the eigenFace space. This matrix of projection is compared with the projection matrix of the input image.

Input / Output

Input Sample 1:

Output Sample 1:

Input Sample 2:

Output Sample 2:

Input Sample 3:

Output Sample 3:

Input Sample 4:

Output Sample 4:

Input Sample 5:

Output Sample 5:

Storing and Grayscale

```python
#Storing the training set
for i in range(0,5):
   for j in range(0,4):
      img_path = "sampleimg/img"+ str(i+1)+ "." + str(j+1) + ".jpg"
      img[i][j] = Image.open(img_path)
      img[i][j] = img[i][j].resize((n,n))

#Converting to Grayscale
img_grey = [[0 for x in range(similar_faces)] for y in range(unique_people)]
for i in range(0,5):
   for j in range(0,4):
      img_grey[i][j] = ImageOps.grayscale(img[i][j])
```

Image to NxN and then NxN to N^2x1

```python
# Converting the image to NxN matrix
b_1 = [[0 for x in range(similar_faces)] for y in range(unique_people)]
b = [[0 for x in range(similar_faces)] for y in range(unique_people)]
for i in range(0,unique_people):
   for j in range(0,similar_faces):
      b_1[i][j] = np.array(img_grey[i][j])
      plt.imshow(b_1[i][j], interpolation='nearest')
      #plt.show()

# Converting the NxN matrix to (N^2)x1 matrix
for i in range(0,unique_people):
   for j in range(0,similar_faces):
      b[i][j]=b_1[i][j].reshape((n*n,1))
```

Calculating the mean matrix

```
# Calculating the Mean Matrix
bMean=[[0 for x in range(n*n)] for y in range(unique_people)]
for i in range(0,unique_people):
   bMean[i]=0
   for j in range(0,similar_faces):
      bMean[i]+=(b[i][j])/similar_faces
```

Calculating the mean matrix

```
#Calculating the transpose of aMatrix
p = nosofmatrices
q = n*n
aMatrix_T = np.zeros((p,q))
for i in range(0,p):
    for j in range(0,q):
        aMatrix_T[i][j] = aMatrix[j][i]
print("ORIGINAL MATRIX IS: \n",aMatrix)
print("TRANSPOSE MATRIX IS: \n",aMatrix_T)

#Covariance Matrix=A_t*A
p = nosofmatrices
q = n*n
covarianceMat = np.zeros((nosofmatrices,nosofmatrices))
sum=0
for i in range(0,p):
   for j in range(0,p):
      for k in range (0,q):
         sum = sum + aMatrix_T[i][k] * aMatrix[k][j]
      covarianceMat[i][j] = sum
      sum=0
```

Transpose and Covariance

```
#Decomposing the Covariance using SVD
AtA = covarianceMat
n= nosofmatrices
d = np.zeros((nosofmatrices,nosofmatrices))
s = np.zeros((nosofmatrices,nosofmatrices))
s1 = np.zeros((nosofmatrices,nosofmatrices))
s1t = np.zeros((nosofmatrices,nosofmatrices))
temp = np.zeros((nosofmatrices,nosofmatrices))
zero= 1e-4
pi = 3.141592654
for i in range (0,n):
 for j  in range(0,n):
   d[i][j]=AtA[i][j]
   s[i][j]=0
#Converting s to an identity matrix
for i in range(0,n):
   s[i][i]=1
flag=0
i=0
j=1
max=math.fabs(d[0][1])
for p in range(0,n):
   for q in range(0,n):
      if(p!=q):
         if(max < math.fabs(d[p][q])):
            max = math.fabs(d[p][q])
            i=p
            j=q
```

Rotational Angle and Diagonal Matrix

```
#Finding rotational angle
if(d[i][i]==d[j][j]):
   if(d[i][j] > 0):
      theta=pi/4
   else:
      theta=-pi/4
else:
   #formula to be used for the the diagonal elements that are found equal
   theta=0.5*math.atan(2*d[i][j]/(d[i][i]-d[j][j]))
#Computing S1 matrix (Transformation matrix)
for p in range(0,n):
   for q in range(0,n):
      s1[p][q]=0
      s1t[p][q]=0
for p in range(0,n):
   s1[p][p]=1
   s1t[p][p]=1
s1[i][i]=math.cos(theta)
s1[j][j]=s1[i][i]
s1[j][i]=math.sin(theta)
s1[i][j]=-s1[j][i]
s1t[i][i]=s1[i][i]
s1t[j][j]=s1[j][j]
s1t[i][j]=s1[j][i]
s1t[j][i]=s1[i][j]

for i in range(0,n):
   for j in range(0,n):
      temp[i][j]=0
      for p in range(0,n):
         temp[i][j]+=s1t[i][p]*d[p][j]

#Getting diagonal matrix
for i in range(0,n):
   for j in range(0,n):
      d[i][j]=0
      for p in range(0,n):
         d[i][j]+=temp[i][p]*s1[p][j]

for i in range(0,n):
   for j in range(0,n):
      temp[i][j]=0
      for p in range(0,n):
         temp[i][j]+=s[i][p]*s1[p][j]

for i in range(0,n):
   for j in range(0,n):
      s[i][j]=temp[i][j]
for i in range(0,n):
   for j in range(0,n):
      if(i!=j):
         if(math.fabs(d[i][j] > zero)):
            flag=1
```

References:

1. "ML: Face Recognition Using Eigenfaces (PCA Algorithm)," *GeeksforGeeks*, 26-Mar-2020. [Online]. Available: https://www.geeksforgeeks.org/ml-face-recognition-using-eigenfaces-pca-algorithm/.
2. D. Gargaro, "The pros and cons of facial recognition technology," *IT PRO*, 25-Aug-2020. [Online]. Available: https://www.itpro.com/security/privacy/356882/the-pros-and-cons-of-facial-recognition-technology.
3. "PCA Theory Examples," .https://www.projectrhea.org/rhea/index.php/PCA_Theory_Examples.
4. S. Zhang and M. Turk, "Eigenfaces," *Scholarpedia*. http://www.scholarpedia.org/article/Eigenfaces.
5. Paul, Liton & Suman, Abdulla. (2012). Face recognition using principal component analysis method. International Journal of Advanced Research in Computer Engineering & Technology (IJARCET). 1. 135-139.
6. N. Acar, "Eigenfaces: Recovering Humans from Ghosts," *Medium*, 02-Sep-2018. [Online]. Available: https://towardsdatascience.com/eigenfaces-recovering-humans-from-ghosts-17606c328184.
7. J. VanderPlas, "In Depth: Principal Component Analysis," *In Depth: Principal Component Analysis | Python Data Science Handbook*. [Online]. Available: https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html.
8. "Implementing a Principal Component Analysis (PCA)," Dr. Sebastian Raschka, 13-Apr-2014. [Online]. Available: https://sebastianraschka.com/Articles/2014_pca_step_by_step.html.
9. S. Dey, "EigenFaces and A Simple Face Detector with PCA/SVD in Python," *sandipanweb*, 08-Jan-2018. [Online]. Available: https://sandipanweb.wordpress.com/2018/01/06/eigenfaces-and-a-simple-face-detector-with-pca-svd-in-python/.
10. j2kun, "Eigenfaces, for Facial Recognition," Math ∩ Programming, 16-Aug-2013. [Online]. Available: https://jeremykun.com/2011/07/27/eigenfaces/.
11. "Faces recognition example using eigenfaces and SVMs¶," *scikit*. [Online]. Available: https://scikit-learn.org/stable/auto_examples/applications/plot_face_recognition.html.
12. ThomIves, "ThomIves/BasicLinearAlgebraToolsPurePy," GitHub. [Online]. Available: https://github.com/ThomIves/BasicLinearAlgebraToolsPurePy.