Project No. 3
"Real-time face Recognition"

Dimension Reduction
(An application of Singular Value Decomposition)

December 18, 2022

Submitted by: Group 9
Liteshwar Rao

Numerical Methods, AI-5003
Topics in Scientific Computing, MAT-5402
Fall 2022
Yeshiva University

Contents

1 Abstract

This project explores one of the applications of Singular Value Decomposition (SVD) of a matrix in image recognition. The main idea is to reduce the dimension of an image in such a manner that SVD preserves important features of an image and helps to reduce cost in terms of time and memory of computer. We compare the accuracy and time involved in identification of an image between dimension reduction using SVD and original image space. We intend to explore the method to find the number of dimension in SVD or columns in left eigenvectors that we need to recognize an image on reduced face space. Furthermore, we also compare results between our code for SVD and *Python* function for SVD. Moreover, we see that how our result changes when we define some threshold level to identify an image.

2 Introduction[1]

This project is drawn from a paper written by **Guoliang Zeng** on **Facial Recognition with Singular Value Decomposition**. We are given a set of images , and we are tasked with developing an algorithm to identify a new image with one of the images in given set which closely resembles its features. In mathematics or computer science, an image is decomposed in terms of matrix of m x n columns where m x $n = M$ represent the pixels of an image. Using the application of SVD we aim to represent all images in reduced dimension while preserving their important characteristics.

3 Methodology

We divide the given set of images into two different sets namely training set and testing set followed by converting all the images in these two sets into a column vector. Finally, for each set we append these column vectors into a matrix of M x N dimension where M represents the number of pixels in the image and N represents the number of images in the set. We select columns of testing set and compare it with the columns of training set one by one in terms of taking the 2-norm of difference between one column of testing and all columns of training set. The one, which gives the least 2-norm we define it to be closely related match of the image from testing set.

4 Approach

We compare two different approaches in this project. First method is to use the full M x N matrix of images and try to identify the new image with one of the images in training set. Second method involves the reduction of dimensions of a M x N matrix into P x N matrix where $P < M$ using **SVD** and then attempt to identify the new image. We explore how to select P in such a way that it helps us identify an image , and it reduces the dimension also. We compare accuracy of these two approaches in terms of right match of an image and cost in terms of time it takes to execute an algorithm.

[1]Paper: Facial Recognition with Singular Value Decomposition, by: Guoliang Zeng, Arizona State University Polytechnic Campus

5 Activity 1

We load all 165 images with **Pillow** library in Python and use **os** to access these images in the folder named 'data' in the directory. We use **matplotlib** to plot some of these images. The plot of few faces is shown below.

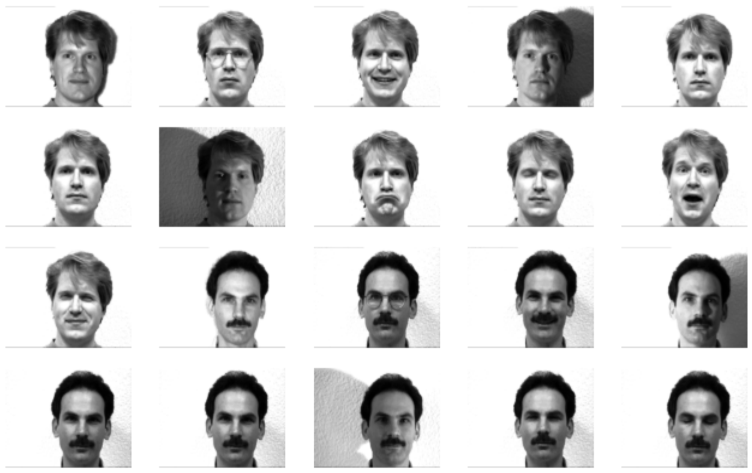


Figure 1 Plot of first 20 images

6 Activity 2

Given the set of 165 images where each image has 243 x 320 pixels, we convert each image into a 1-D array of size $M = 77760 = 243$ x 320 of real values. Append all converted arrays into a matrix of M x N dimension where $M = 77760$ represents real values of pixels of an image and $N = 165$ represents the number of images. We compute the mean face and plot it as shown below:

$$\boxed{\bar{f} = \frac{1}{N}\sum_{i=1}^{N} f_i} \tag{1}$$

In matrix form we can write the converted images into 1-D array as shown below. The $\bar{f}$ has dimension of 77760 x 1.

$$\textbf{Images converted into 1-D array} = \begin{bmatrix} f_{11} & f_{12} & \dots & f_{1N} \\ f_{21} & f_{22} & \dots & f_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ f_{M1} & f_{M2} & \dots & f_{MN} \end{bmatrix}$$

$$\bar{f} = \tfrac{1}{N} \begin{bmatrix} \sum_{i=1}^{N} f_{1i} \\ \sum_{i=1}^{N} f_{2i} \\ \vdots \\ \sum_{i=1}^{N} f_{Mi} \end{bmatrix}$$

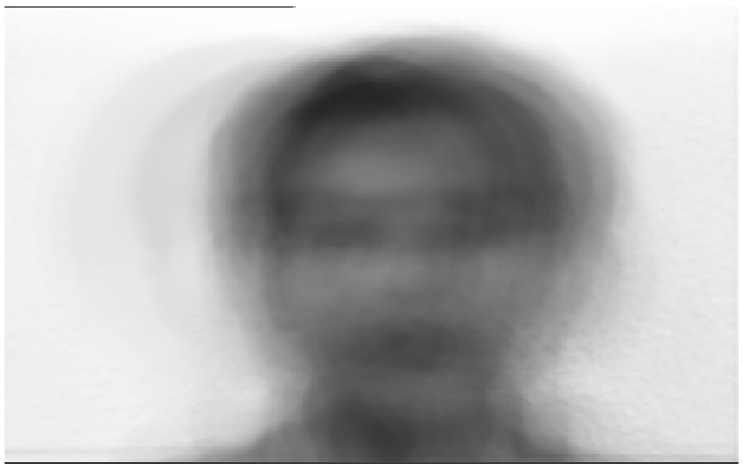


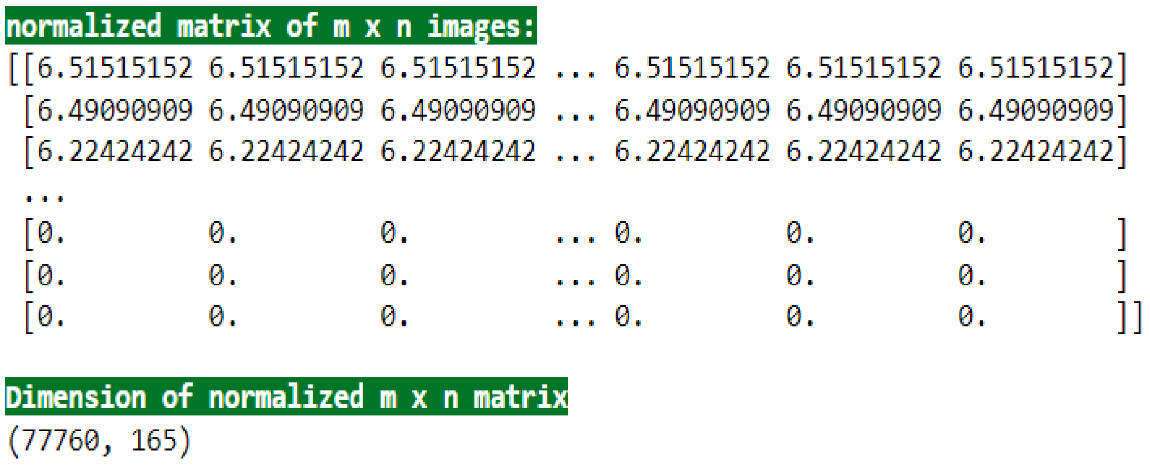
Figure 2 Mean face $\boldsymbol{\bar{f}}$

We normalize each column (image) by subtracting the mean face from its column. It is defined below.

$$\boxed{a_j = f_{ij} - \bar{f}} \tag{2}$$

$$\text{where } i = 1, 2, \ldots, M \text{ and } j = 1, 2, \ldots, N$$

A few elements of normalized matrix is shown below.



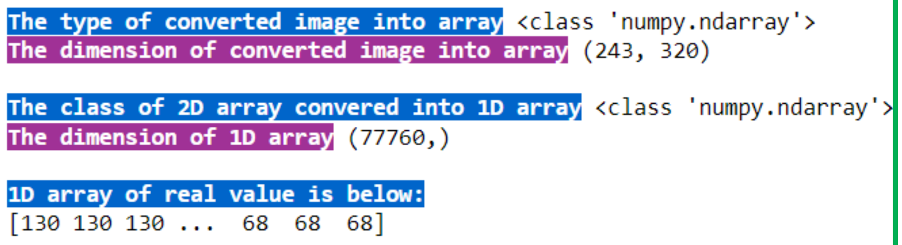
```
normalized matrix of m x n images:
[[6.51515152 6.51515152 6.51515152 ... 6.51515152 6.51515152 6.51515152]
 [6.49090909 6.49090909 6.49090909 ... 6.49090909 6.49090909 6.49090909]
 [6.22424242 6.22424242 6.22424242 ... 6.22424242 6.22424242 6.22424242]
 ...
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]]

Dimension of normalized m x n matrix
(77760, 165)
```

Figure 3 Normalize data

We write following functions. Their sample output is also shown below.

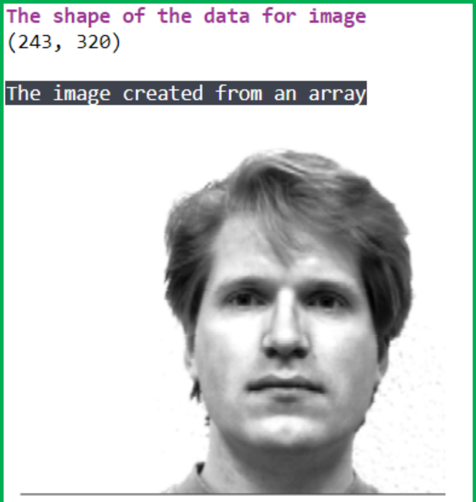
(a) **img2vec** that converts an image of 243 x 320 pixels into 1-D array of $M = 77760$ column vector. The class, dimension of an image and its converted elements into 1-D array are shown below in figure 4.

(b) **vec2img** that converts an array of $M = 77760$ into an image. We first transform back this 1-D array into $m = 243$ x $n = 320$ pixels to plot it as an image. The sample is shown below in figure 5.



```
The type of converted image into array <class 'numpy.ndarray'>
The dimension of converted image into array (243, 320)

The class of 2D array convered into 1D array <class 'numpy.ndarray'>
The dimension of 1D array (77760,)

1D array of real value is below:
[130 130 130 ...  68  68  68]
```

Figure 4 img2vec output



```
The shape of the data for image
(243, 320)

The image created from an array
```

Figure 5 vec2img output

7 Activity 3

We divide the total set of 165 images (11 images per individual for 15 individuals) into two disjoint sets. We call one data set as training set denoted by *known* folder and other data set as testing set denoted by *unknown* folder. We decide to keep 5 images per individual in ‘unknown’ folder and other 6 images in ‘known’ folder. So, total images in ‘unknown’ folder is 5 x 15 = 75 and in ’known’ folder it is 6 x 15 = 90. The images which are in ‘unknown’ folder end with rightlight, leftlight, normal, happy, and noglasses. Conversely, the images which are in ‘known’ folder end with centerlight, glasses, sad, sleepy, surprised, and wink.

8 Activity 4

We write a function **query(face, database)** that receives a face from 'unknown' folder and compare it with all the images in 'known' folder, which we call it database and returns the identity of the closest match. Mathematically, it is defined as below.

(a) Use the results of activity 3 for division of images into two disjoint sets.

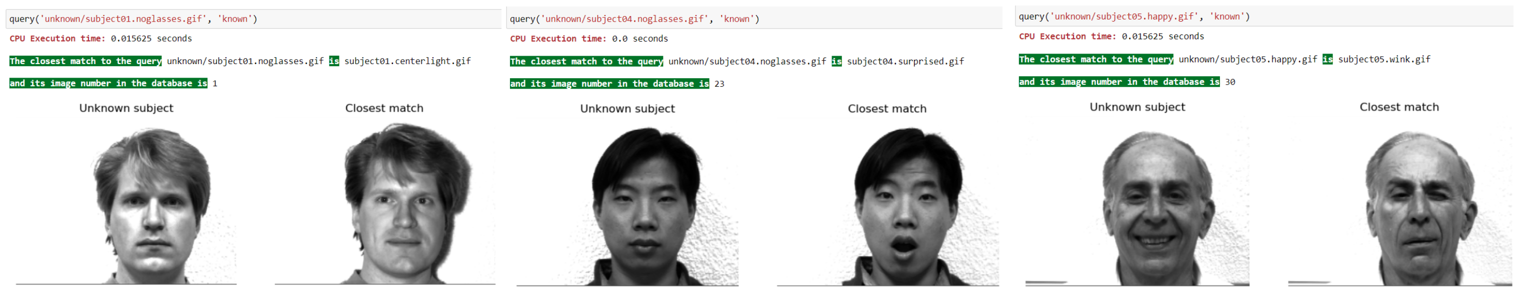
(b) Use the results of activity 2 to convert all images in both 'unknown' and 'known' folders into 1-D array of column vectors. The columns in 'unknown' and 'known' folders are represented by f_{ui} and f_{kj} where u and k denotes 'unknown' and 'known' folder respectively and $i = 1, 2, \dots, 75$ and $j = 1, 2, \dots, 90$.

(c) Use one of the columns, say f_{ui}, from 'unknown' folder and take the 2-norm of $\|f_{ui} - f_{kj}\|_2$ for $j = 1, 2, \dots, 90$ and choose which gives the least value.

(d) The least value of 2-norm gives the closest identity in 'known' folder with respect to the image in 'unknown' folder.

We checked our algorithm with 6 columns in 'unknown' folder and find that 3 are correctly identified and 3 are not correctly identified by the algorithm. The results are shown below. The algorithm gives us the identity of the closest image, image number in the 'known' folder, and execution time to find the closest match.

We can say that, not generally though, but with our careful selection of images in 'unknown' folder that end with noglasses are identified correctly. However, images that end either with rightlight or leftlight are incorrectly identified.

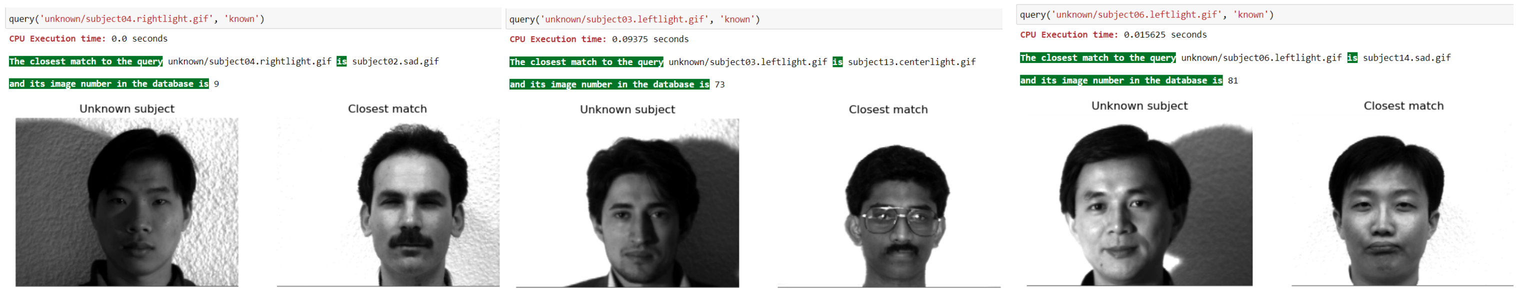


```
query('unknown/subject01.noglasses.gif', 'known')
CPU Execution time: 0.015625 seconds
The closest match to the query unknown/subject01.noglasses.gif is subject01.centerlight.gif
and its image number in the database is 1
```

```
query('unknown/subject04.noglasses.gif', 'known')
CPU Execution time: 0.0 seconds
The closest match to the query unknown/subject04.noglasses.gif is subject04.surprised.gif
and its image number in the database is 23
```

```
query('unknown/subject05.happy.gif', 'known')
CPU Execution time: 0.015625 seconds
The closest match to the query unknown/subject05.happy.gif is subject05.wink.gif
and its image number in the database is 30
```

Figure 6: Correctly identified Faces



```
query('unknown/subject04.rightlight.gif', 'known')
CPU Execution time: 0.0 seconds
The closest match to the query unknown/subject04.rightlight.gif is subject02.sad.gif
and its image number in the database is 9
```

```
query('unknown/subject03.leftlight.gif', 'known')
CPU Execution time: 0.09375 seconds
The closest match to the query unknown/subject03.leftlight.gif is subject13.centerlight.gif
and its image number in the database is 73
```

```
query('unknown/subject06.leftlight.gif', 'known')
CPU Execution time: 0.015625 seconds
The closest match to the query unknown/subject06.leftlight.gif is subject14.sad.gif
and its image number in the database is 81
```

Figure 7: Incorrectly identified Faces

9 Activity 5

Using the algorithm of activity 4, we compare all the columns in 'unknown' folder with 'known' folder and check the accuracy of algorithm in terms of percentage of images correctly identified by it. To count how many images are correctly identified we see the output. We know that accuracy depends on how we divide the total images into two disjoint sets. So, algorithm gives different results on different disjoint sets. For the sets we have now, we can say that the accuracy is approximately 21.33% because we got 16 images out of 75 images correctly identified. The execution time is either 0 or 0.015625 seconds except for few images. Moreover, we can say that images that are correctly identified end with normal, happy, and noglasses. This is not true in opposite direction. We don't intend to say that images that end with these three features are correctly identified always. But images that end with either rightlight or leftlight are never identified correctly.

Now, we create new disjoint sets to check the algorithm. This time we are keeping images that end with centerlight, leftlight, sleepy, surprised, and wink in 'unknown1' folder and images that end with rightlight, normal, happy, noglasses, glasses, and sad in 'known1' folder. For the sets we have now, we can say that the accuracy is approximately 29.33% because we got 22 images out of 75 images correctly identified. The execution time is either 0 or 0.015625 or 0.09375 seconds except for few images. Moreover, we can say that images that are correctly identified end with sleepy, surprised, and wink. But images that end with either centerlight or leftlight are never identified correctly.

Further, we create new disjoint sets to check the algorithm. This time we are keeping images that end with noglasses, normal, rightlight, sad, and sleepy in 'unknown2' folder and images that end with leftlight, happy, centerlight, glasses, surprised, and wink in 'known2' folder. For the sets we have now, we can say that the accuracy is approximately 21.33% because we got 16 images out of 75 images correctly identified. The execution time is either 0 or 0.015625 seconds except for few images. Moreover, we can say that images that are correctly identified end with sleepy, sad, normal, and wink. But images that end with rightlight never identified correctly.

Lastly, we create new disjoint sets to check the algorithm. This time we are keeping images that end with happy, normal, sad, sleepy, and wink in 'unknown3' folder and images that end with leftlight, centerlight, rightlight, glasses, noglasses, and surprised in 'known3' folder. For the sets we have now, we can say that the accuracy is approximately 46.66% because we got 35 images out of 75 images correctly identified. The execution time is either 0 or 0.015625 or 0.109375 seconds except for few images. Moreover, we can say that images that are correctly identified end with happy, normal, sad, sleepy, and wink. In this division, we have correct identification for each type of image in 'unknown3' folder.

10 Activity 6

For this activity we work with the images/columns in 'known3' folder, because with this division we get higher accuracy in activity 5, and we like to compare the accuracy of algorithm of activity 5 with that of SVD algorithm we plan to do in subsequent sections. We compute Singular Value Decomposition (SVD) of the matrix that has images of 'known3' folder as its columns. Let's call this matrix A and its dimension is $M = 77760$ x $N = 90$. The reason why we compute SVD of 'known3' folder only, and why we do not compute SVD of its corresponding folder 'unknown3' is that we want to find the reduced face space or in other words we want to reduce the dimension of matrix A, and we like to compare the images in 'unknown3' folder with this reduced face space to check if we can perform face recognition with reduced dimension or we can reduce the cost in terms of computation because we have smaller matrix now without compromising the accuracy. Mathematically speaking we want to project the faces or images or vectors of folder 'unknown3' on this reduced face subspace with the objective to reduce the distance between the plane defined by this reduced face subspace which has vectors of 'known3' folder and vectors of 'unknown3' folder. If we compute SVD of 'unknown3' folder also and then perform face recognition, our purpose of face recognition defeats naturally because the vectors of 'unknown3' are already present in the reduced face subspace. There is nothing left to project on this subspace. All the images in 'unknown3' folder will then be linear combination of its own images and images of 'known3' folder, and we are sure to get 100% accuracy. Then we are left with no data for testing. The left eigenvectors of U matrix serve as basis vectors of set of images in training set. It means all images in training set can be obtained as a linear combination of these basis vectors. If SVD is computed on testing set too, then we have a different basis vectors and all images in both training and testing set can be obtained as a linear combination of these basis vectors. In other words, now our testing set is already contained in training set, and then our projection leads to 100 % matching. Or in other words, we are working with one set of training images only. In brief, we are not conducting face recognition of unknown images. Therefore, SVD is computed only on training set.

It is similar to the problem of least square method wherein we attempt to find the orthogonal projection of a vector on the plane, which does contain this particular vector. If this vector is already present in the plane, then this must be the linear combination of the basis vectors of the plane.

We execute following steps to compute SVD.

(a) Use activity 3 to divide the images into two disjoint sets namely 'known3' and 'unknown3' folder.

(b) Use activity 2 convert the images of 'known3' folder into 1-D array and append them into matrix A.

(c) Use further activity 2 to compute the mean face of these columns.

(d) Again use activity 2 to normalize the columns of matrix A by subtracting from them the mean face. Let's call this normalized matrix B. Now, our task is to compute SVD of B.

$$\text{Using SVD, we can write } \boxed{B = U\Sigma V^T} \tag{3}$$

The dimension of B is 77760 x 90, of U is 77760 x 77760, of Σ is 77760 x 90, and of V^T is 90 x 90. The matrix U and V are orthonormal matrices; the dot product of their column is zero if column numbers are different, otherwise it is 1. The norm of the vectors is 1. The Σ matrix is diagonal matrix with singular values in its diagonal. We know that for an orthonormal matrix, the transpose of an orthonormal matrix is the inverse of matrix.

$$\text{consider } B^T.B$$

$$\boxed{\begin{aligned} B^T.B &= V.\Sigma^T.U^T.U.\Sigma.V^T \\ &= V.\Sigma^T.\Sigma.V^T \end{aligned}} \qquad \begin{matrix}(4)\\(5)\end{matrix}$$

By using the concept of Diagonalization we can say that, the matrix V has orthonormal columns which are eigenvector of $B^T.B$ and the matrix $\Sigma^T.\Sigma$ has eigenvalues (λ_i) of $B^T.B$. So, the singular values in the matrix Σ are $\sqrt{\lambda_i}$. So, we compute $B^T.B$, its eigenvectors and its eigenvalues which give us V and Σ. Then, using the relation between orthonormal matrix and its inverse we compute matrix U by the following.

$$\text{Multiply both side by } V \text{ in equation 3}$$

$$\boxed{\begin{aligned} B.V &= U.\Sigma \\ B.v_i &= \sigma_i.u_i \\ u_i &= \begin{cases} \frac{B.v_i}{\sigma_i}, & \text{for } i = 1, 2, \ldots, N \\ 0 & \text{for } i = N+1, N+2, \ldots, M \end{cases} \end{aligned}} \qquad \begin{matrix}(6)\\(7)\\(8)\end{matrix}$$

The base faces are defined by the first N columns of U. Since these columns are orthonormal, their norm is 1. They do not represent the pixels anymore. We scale them back to 0:255 and then plot the base faces. The first 20 base faces are shown below.

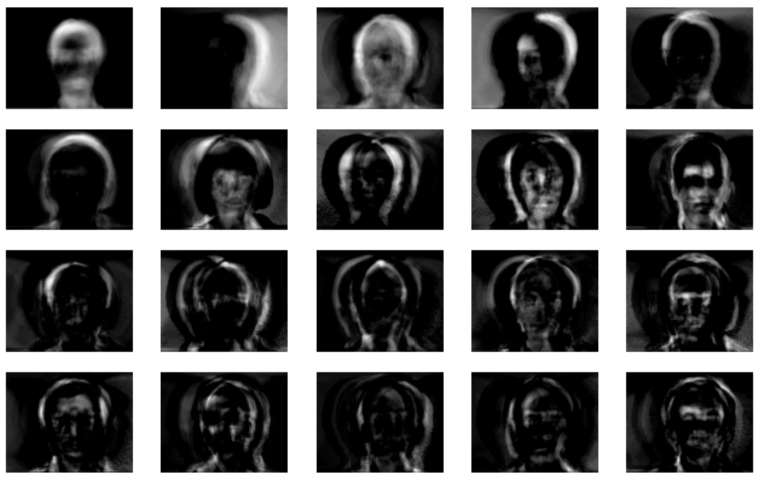


Figure 8 Base faces $\boldsymbol{u_i}$

11 Activity 7

Our task is to select appropriate number of columns, say p, of U such that it reduces the dimension of base faces and retains maximum information contained in matrix B. The procedure to select p is described below.

Initially we have

$$\boxed{B_{mXn} = U_{mXm}.\Sigma_{mXn}.V_{nXn}^T} \tag{9}$$

By using equations 6, 7, and 8, the dimensions has reduced as shown below.

$$\boxed{B_{mXn} = U_{mXn}.\Sigma_{nXn}.V_{nXn}^T} \tag{10}$$

because singular values are zero for $n+1, n+2, \ldots, m$
and the corresponding columns of U are also zero

Now, matrix B can be written as:

$$\boxed{B = \sigma_1.u_1.v_1^T + \sigma_2.u_2.v_2^T + \ldots + \sigma_n.u_n.v_n^T} \tag{11}$$

such that $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_n \geq 0$

We need to select appropriate p number of $\sigma_i's$ as follows.

$$\boxed{B_p = \sigma_1.u_1.v_1^T + \sigma_2.u_2.v_2^T + \ldots + \sigma_p.u_p.v_p^T} \tag{12}$$

such that

$$\boxed{\|B - B_p\|_2 = \sigma_{p+1} \text{ is at a minimum}} \tag{13}$$

is provided by retaining the first p terms in equation 11

We plot the singular values $\sigma_1, \sigma_2, \ldots, \sigma_n$ as a function of p and look at their decay. We choose appropriate p where 'elbow' of the graph approximately starts. The plot is shown below. We see that elbow of the graph approximately starts at $p = 20$. So, we select 20 columns of U.

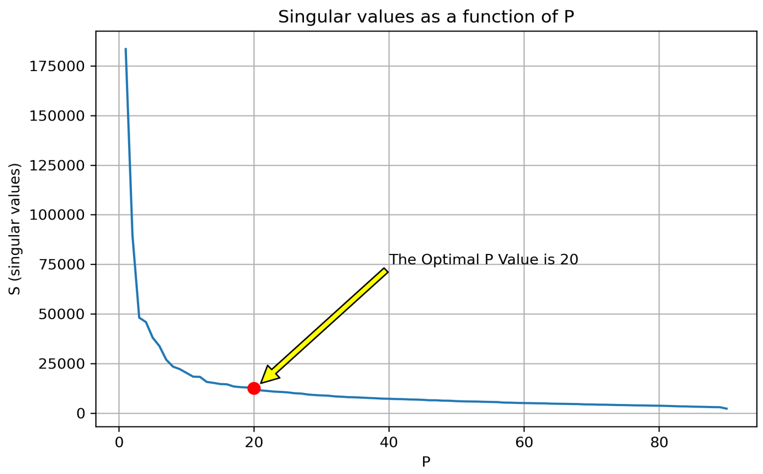


Figure 9 Singular values $\sigma_i's$ as function of p

12 Activity 8

Now, we project images contained in training and testing sets on $p = 20$ columns of U we found in activity 7. The projection works as follows. For an image i represented by its coordinate vector x_i has a dimension of $M = 77760$ x $N = 1$.

$$\boxed{x_i = \begin{bmatrix} u_1 & u_2 & \dots & u_{20} \end{bmatrix}^T .(f_i - \bar{f})} \tag{14}$$

where f_i is a 1-D array using img2vec function and f is mean face using activity 2

Note that we are normalizing both our training and testing sets by subtracting mean face from an image as shown by equation 14 in $(f_i - \bar{f})$.

We project all 90 and 75 images in training and testing sets respectively using the division rule of activity 5 for 'unknown/known' folders. The matrix of projected training set has dimension of 20 x 90 and the matrix of testing set has dimension of 20 x 75. Notations are shown below.

$$\boxed{\begin{aligned} R &= \begin{bmatrix} x_{r1} & x_{r1} & \dots & x_{r90} \end{bmatrix} \qquad (15) \\ S &= \begin{bmatrix} x_{s1} & x_{s1} & \dots & x_{s75} \end{bmatrix} \qquad (16) \end{aligned}}$$

where R and S denotes training and testing set respectively

Now, we compare each column of testing set with each column of training set and select the one which gives the least 2-norm of difference. It is shown below.

$$\boxed{\|x_{si} - x_{rj}\|_2} \tag{17}$$

$$\text{for } i = 1, 2, \dots, 75$$
$$\text{for } j = 1, 2, \dots, 90$$

The least value of 2-norm gives the closest identity in training set with respect to the image in testing set. Similar to activity 5, we check our algorithm for 4 different training and testing sets, namely we compare 'unknown' folder with 'known' folder, 'unknown1' folder with 'known1' folder, 'unknown2' folder with 'known2' folder, and 'unknown3' folder with 'known3' folder. The comparison between SVD based face recognition on reduced face space versus face recognition on full space is summarized in the table below.

Table 1: Comparison of Face Recognition methods

Parameter	Folder	SVD	Full Space
Accuracy	known/unknown	74.66 %	21.33 %
	known1/unknown1	72 %	29.33%
	known2/unknown2	82.66 %	21.33 %
	known3/unknown3	90.66 %	46.66 %
Execution Time	known/unknown (0 seconds)	61	54
	known/unknown (0.015625 seconds)	8	15
	known1/unknown1 (0 seconds)	61	45
	known1/unknown1 (0.015625 seconds)	10	26
	known2/unknown2 (0 seconds)	64	41
	known2/unknown2 (0.015625 seconds)	8	31
	known3/unknown3 (0 seconds)	59	43
	known3/unknown3 (0.015625 seconds)	12	28

For execution time, we are counting for how many queries executed in 0 and 0.015625 seconds since these two execution times appear more frequently. Other execution times are different and appear for just once or twice and they are close to 0.015625.

We can see that accuracy has significantly improved with SVD algorithm. The accuracy with SVD algorithm is almost 2.5 times that of full space algorithm. For instance, for known1/unknown1 folder case, accuracy with SVD is 72 % while with full algorithm it is 29.33 %.

In terms of execution time, we can see that SVD algorithm runs faster since number of queries executed in 0 seconds are higher for each case of SVD algorithm. For instance, for known1/unknown1 folder, we got 61 queries matched in 0 seconds, while with full algorithm we got 45 queries matched in 0 seconds. Similarly, we got 10 queries matched in 0.015625 seconds with SVD while it is 26 with full algorithm. So, full algorithm is taking more time in query matching.

The parameter p is critical here. If we keep on increasing p from $p = 20$ till $p = 71$ we don't see any change in accuracy. It means each new singular value σ_i ,for $i = 21, 22, \ldots, 71$, is not contributing much in terms of the information contained in matrix B in equation 10. The change in accuracy by just 1% is taken place when $p = 72$. Moreover, if we increase p from $p = 1$ to $p = 2$ or $p = 3$ the incremental change in accuracy is higher than compared to increase in p. After increasing the p from $p = 3$ to $p = 20$ the incremental change in accuracy is small as compared to change in p. We can conclude that we can get maximum accuracy with the least possible execution time when $p = 20$, after which there is no change or a little change in accuracy.

It makes sense to use SVD for this problem since we get higher accuracy in lesser time than with full space algorithm.

13 Extra credit 1

Now, we compare our code for SVD with *Python* library function SVD in terms of accuracy and execution time. We use only **fullmatrices=False** in SVD argument of Python library. For this comparison, we use our combination of 'known3/unknown3' folders. We see that we get exactly the same results with Python library as with our code of SVD in terms of accuracy and number of queries matched in 0 seconds. We find changes only in terms of number of images identified in 0.015625 seconds. Accuracy is still at 90.66%. No. of images identified in 0 and 0.015625 seconds are 59 and 5 respectively.

Now, we run the same experiment on 'unknown2/known2' folders. This time also we get same results in terms of accuracy and number of queries matched in 0 seconds. We find changes only in terms of number of images identified in 0.015625 seconds. Accuracy is still at 82.66%. No. of images identified in 0 and 0.015625 seconds are 64 and 5 respectively.

If we consider number of images identified in 0.015625 seconds by this Python SVD function we can say that Python function slightly runs faster.

14 Extra credit 3

In addition to executing an algorithm for face recognition, which works by choosing the face in terms of the least value of the 2-norm of the difference between a testing image and all images in training set, we define an arbitrary threshold of $\epsilon_0 = 3000$. If we get 2-norm of a testing image greater than 3000, then we consider that image from testing set as a new identity and we add this image in the training set. However, if we get 2-norm less than 3000, the algorithm chooses the face which has the least 2-norm among all the faces which have 2-norm less than 3000. To see how it works, we use our 'unknown3/known3' folders. We see that 23 images from 'unknown3' folder has failed to pass the 3000 tolerance level of 2-norm and consequently has added to the 'known3' folder as new identities. Other 52 images have been identified correctly.

To see how the algorithm works if we change our tolerance level. We set new tolerance level $\epsilon_0 = 1000$. We see that 61 images from 'unknown3' folder has failed to pass the 1000 tolerance level of 2-norm and consequently has added to the 'known3' folder as new identities. Other 14 images have been identified correctly.

Now, we repeat the same process with $\epsilon_0 = 10000$. We see that none of the images from 'unknown3' folder has failed to pass the 10000 tolerance level of 2-norm because it is set at higher level and consequently none has added to the 'known3' folder as new identities. Out of all images passed the tolerance level, 7 of them were identified incorrectly. Other 68 images have been identified correctly.

Generally, we can see that lower the tolerance level, higher the accuracy and addition of more images as new identity. Conversely, higher the tolerance level, lower the accuracy and addition of fewer images as new identity.

References

[1] Plot multiple images using subplot function
https://www.youtube.com/watch?v=hvmZHNUqcqY&list=PLYB63YniHqMkDzbhEWxfA411c2N3bOAPT
https://www.youtube.com/watch?v=uGPqnPwr1JY&t=255s

[2] Load images in Jupyter using Pillow
https://machinelearningmastery.com/how-to-load-and-manipulate-images-for-deep-learning-in-python-with-pil-pillow/

[3] Converting the image to numpy array and back operation
https://www.pluralsight.com/guides/importing-image-data-into-numpy-arrays

[4] Reading images in Python
https://www.youtube.com/watch?v=52pMFnkDU-4

[5] Save images to a folder
https://www.youtube.com/watch?v=6Qs3wObeWwc&t=475s

[6] Execution Time
https://pynative.com/python-get-execution-time-of-program/