Numerical Methods Project 3 - Facial Recognition and Single Value Decomposition

Siddanta KC Juju Ren Rivka Shapiro
Yeshiva University: Katz School of Science and Health

December 23, 2022

1 Introduction

Facial recognition software has been a goal of many. With various possible applications, it is widely sought after and studied. However, a main issue with facial recognition is processing the data. What may appear to be a simple face is actually an image with hundreds of thousands of pixels. Computationally, this is far too much data to produce a fast and effective recognition algorithm. One solution to this problem is dimensionality reduction; a method of compressing and reducing the data for a given image. The algebraic technique of Singular Value Decomposition (SVD) can be used to accomplish dimensionality reduction. SVD can be described in the matrix equation: $A = U\Sigma V^T$, Where A is the original matrix of dimension $m \times n$, U is a matrix of dimension $m \times m$, containing the orthonormal eigenvectors of AA^T, Σ is an $m \times n$ diagonal matrix containing the square root of the eigenvalues of $A^T A$, and V is a matrix of dimension $n \times n$ containing the orthonormal eigenvectors of $A^T A$. The matrix U is considered the reduced form of the matrix A, and can be used more easily in facial recognition algorithms.
In the project we will use SVD to write a facial recognition algorithm. We will assess the accuracy and efficiency of our algorithm by comparing it to the results obtained from non-reduced images.

2 Results and methodologies

2.1 Activity 1

This activity can be thought of as a preparation for the project. Here, we process the given database and reorganize it, separating the images into folders based on the subject to which they belong. The database contains 165 images; with fifteen subjects having eleven images each. Our organization will make the images easier to reference in future activities.

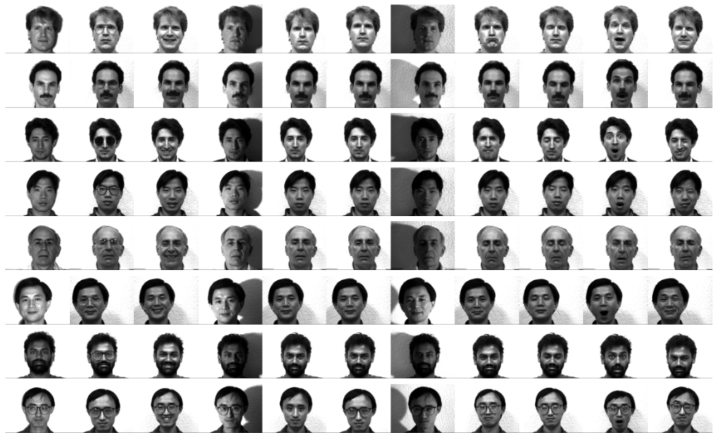

Figure 1: Facial Recognition With SVD

2.2 Activity 2

In this activity we write two opposite functions to convert images to vectors and vice versa. With given image with a dimension of 243×320 pixels, function Img2Vec we built in this step can convert all images to a flatten vector which is one dimentional array of size $m = 77760 = 243 \times 320$. The reverse function, Vec2Img, reshapes the vector back from an array to an image of $m \times n$ which can then be displayed as an $m \times n$ image.
with the database of 165 faces, each face has $m \times n = M$ pixels. There are $N = 165$ individual face images. After testing these functions on a single image, we then converted all images in the database into vectors, and compiled all the vectors into a matrix with final dimensions 165×77760. We then found the mean of the data set matrix, and displayed it as figure 2. The mean image is calculated

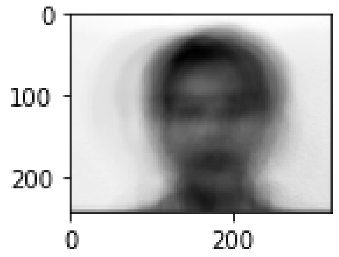

Figure 2: mean face of 165 sample images

by the function $\bar{f} = \frac{1}{N}\sum_{i=1}^{N} f_i$ which formed the matrix of images with the computed $\bar{f}$. The vector form of the mean was then subtracted from each column of the dataset matrix to normalize it.

2.3 Activity 3

In this activity we will begin developing the recognition algorithm. To start, we break up our original data into a training set and a testing set. Our training set includes 6 images from each subject, and the testing set is populated with the remaining 5. The images are then compressed into arrays and stored in two matrices.
With the help with Numpy library function Random.choice. The image data from each subject is randomly spited into two groups: Train and test. The training group contains 6 faces of each subject, which is $6 \times 15 = 90$ faces. The testing group contains 5 faces of each subject, which is $5 \times 15 = 75$ faces. The testing set of data can also be referred to as the query set, as it is filled with "unknown" faces which must be queried against the known.

2.4 Activity 4

The recognition algorithm is very simple. For the unknown, or query, image, we find the known image with the smallest difference. We apply the *np.linalg.norm*() function to calculate the vector norms of the image with each image vector in the database. The minimum distance is being selected as the closest face's distance. The subject ID number of the closest face is bring selected as the id number of the input image. This is accomplished by subtracting the vector of the query image from every column of the training matrix (which contains all the known faces). We run the recognition algorithm on these training and testing sets as they are, existing in the original face space.
In a test of 20 random images, the algorithm is between 75-90% accurate. We will later compare these results to those of testing on a reduced space

2.5 Activity 5

In this activity, we will determine if the distribution of images between testing and training sets affects the accuracy of the algorithm. We built the function *accuracytest* looping with all data in multiple rounds, every round, the testing data and the training data will be randomly split. The function is showing with the percentage of the true face recognition (accuracy) and monitoring the time of each loop.

We tested 30 random distributions, and queried every testing image in each round of distribution. However, the accuracy of the rounds remained between 70% and 90%, with an average of 79.5%. We recorded the time taken to complete each round, which is 0.59 seconds on average. It can be concluded that the size and distribution of the training and querying sets has a small, if not insignificant effect on the accuracy of recognition.

Table 1: Accuracy & Efficiency

Round	**Accuracy**	**Time**
0	0.7466666666666667	0.5585098266601562
1	0.8666666666666667	0.5935287475585938
2	0.8	0.5617804527282715
3	0.8133333333333334	0.5587940216064453
4	0.8133333333333334	0.5816190242767334
5	0.7733333333333333	0.5848855972290039
...	...	...

2.6 Activity 6

Now we will apply dimensionality reduction. This is accomplished by using SVD, and taking the $B = U_p^T \times A$ matrix as the new "reduced face space".[1] As previously discussed, the U matrix is a square matrix of dimension m×m, containing the orthonormal eigenvectors of the original matrix multiplied by its transpose U_p^T. is the transpose of the first p columns of the U matrix. We will use the matrix of all vectorized images produced in Activity 2, as our original matrix, A. However, as images can only be plotted on a value scale of [0 : 255], the new reduced matrix must be normalized so that all the entries lie within this range. We can then convert the vectors to images which we will call "base faces", as they are from basis vectors of the face space.

2.7 Activity 7

Now we will work on actually reducing the dimension of the face space. To do so, we will utilize the the matrix. Recall that this matrix is rectangular, and its diagonal entries correlate to the root of the eigenvalues of the V matrix. These diagonal entries are also referred to as Singular Values. In plotting the singular values, we can assess to which dimension we will reduce the face space. Our plot shows an elbow in rank at $s_{40} = 9833$, so we will set our p value to 40. We will then take the first 40 columns of U and transpose them. Then we will dot this new matrix with our original matrix A. This will result in a reduced facespace.

2.8 Activity 8

Using our new subspace of images, a matrix of dimension (40,165), we will randomly split the set into testing and training groups as done previously in Activity 4. After 30 rounds of random set distribution and query, the new subspace was able to achieve an average accuracy of 78.4% and an average time of 0.0282 seconds.
We then investigated the effects of the size of p on our results. We ran accuracy tests on matrices of dimension (100, 165) and (10,165). The matrix with p=100 had an average recognition accuracy of 79.1% and the matrix with p=10 an average accuracy of 71.9%. Both ran with an average time of 0.03 seconds. .

3 Conclusion

The goal of this project was to investigate the application of singular value decomposition to facial recognition algorithms. The results of our experiments show that while using an original database produces an average accuracy of 79.5%, it takes 0.5 seconds, a considerable amount of computational time. Reducing the data using SVD only reduces the accuracy by about 1-3%, but reduces the

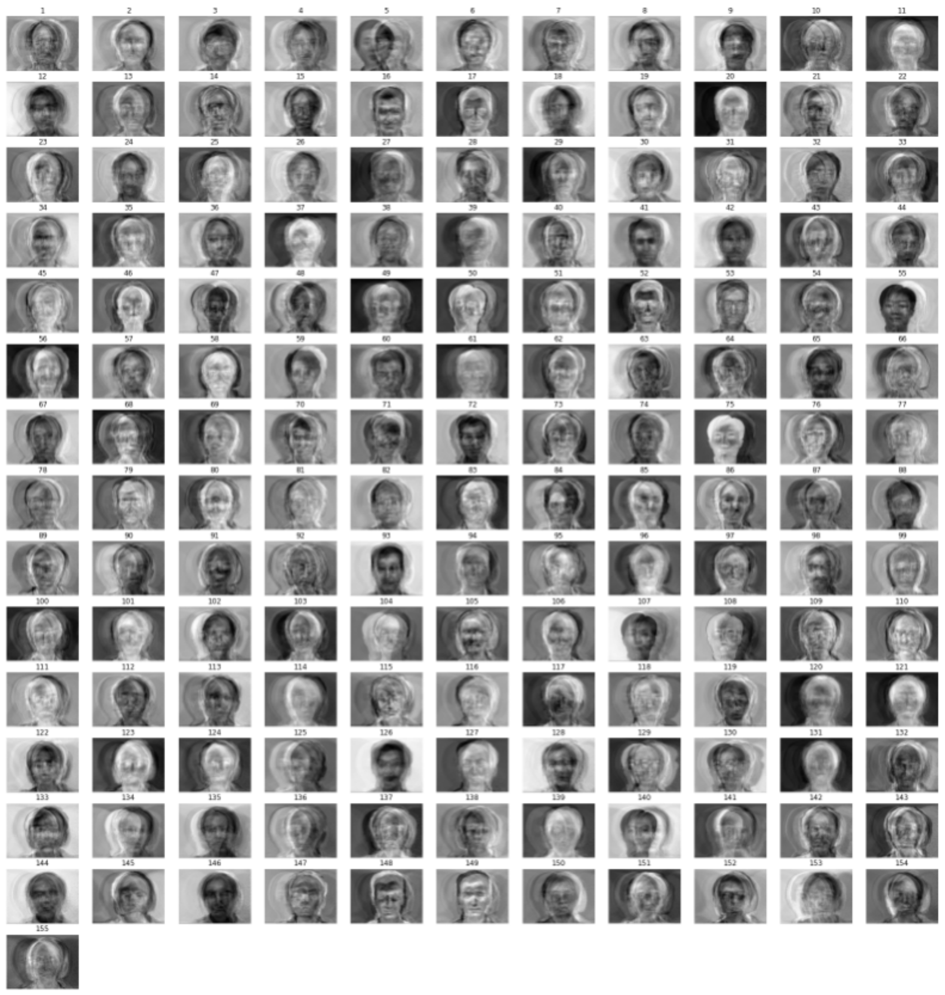

Figure 3: Base faces

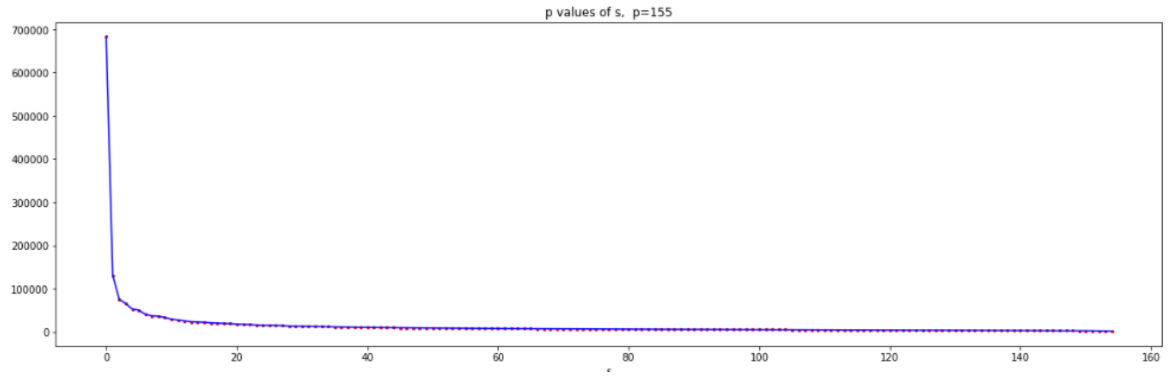

Figure 4: Plot of Singular Values

computational time to about 0.03 seconds, a 94% decrease compared to the original (normalized) data. The size of p affects the results minimally. A change in p does not have an effect on the computation time. However, an increase in p does increase the accuracy, and a decrease in p decreases the accuracy, but only slightly. It can therefore be concluded that reduction by SVD is worthwhile, as even using a larger p value in the reduction still significantly decreases the computational time, while maintaining nearly equivalent accuracy.

Table 2: Accuracy & Efficiency

Round	**Accuracy**	**Time**
0	0.8	0.03799891471862793
1	0.7466666666666667	0.030079126358032227
2	0.8133333333333334	0.026935577392578125
3	0.7333333333333333	0.015041112899780273
4	0.84	0.03133511543273926
5	0.68	0.031246185302734375
6	0.8266666666666667	0.03180980682373047
7	0.84	0.01571202278137207
8	0.7333333333333333	0.029800891876220703
9	0.7466666666666667	0.02920842170715332
10	0.72	0.014106273651123047
...	...	...
25	0.8	0.015268087387084961
26	0.8533333333333334	0.01617884635925293
27	0.7733333333333333	0.03133988380432129
28	0.8266666666666667	0.0312502384185791
29	0.8133333333333334	0.03180503845214844

References

[1] Guoliang Zeng. Facial recognition with singular value decomposition. In *Advances and innovations in systems, computing sciences and software engineering*, pages 145–148. Springer, 2007.