Katz School of Science and Health

AI 5003 Numerical Methods

Real-time Face Recognition using the SVD

Sana Omar & Radek Holik

Fall 2022

Table of Contents

1 Theoretical Introduction

Dimension reduction is the process of reducing the number of variables under observation. This is because we have multiple univariate data points in high dimensions. We are treating the image either as a matrix of data points, or a vector of data points. This is where the concept of face space or (i.e., subspace) of much lower dimensionality, embedded in a higher dimensional image space originates. The main issue is how to properly define and determine a low-dimensional subspace of a facial appearance.

Dimensionality reduction is defined as $R^N \rightarrow R^M$ ($M<N$) and can be divided into feature selection and feature extraction.

Feature selection is choosing a subset of all the features

$$[x_1\ x_2\ \dots\ x_n] \text{ Feature selection } [\ x_{i1}\ x_{i2}\ \dots\ x_{im}\ ]$$

Feature extraction is creating new features from existing ones

$$[x_1\ x_2\ \dots\ x_n] \text{ Feature extraction } [\ y_1\ y_2\ \dots\ y_m\ ]$$

Dimensional reduction brings several benefits. It reduces the demands on the computer's CPU and operating memory. Reduction is especially crucial in real-time systems such as surveillance cameras at airports, computer-human interaction on cell phones, system key access based on face recognition, etc.

1.1 Singular Value Decomposition (SVD)

Singular value decomposition (SVD) is a factorization and linear dimension reduction technique of a rectangular real or complex matrix, it has several applications such as signal processing and statistics. We intend to apply it to facial recognition because it follows the approach of PCA which is used to extract the holistic global features of the training set, by taking the mean-square error sense.

It uses the concept of a covariance matrix; which makes it a second-order method. SVD reduces the dimension of high-dimensional and highly variable data into a lower dimension by looking for the largest variance. This process is done by finding the orthogonal linear combinations of the original variables.

When variances are sorted, we can ignore variances up to a point of choice. This is indeed the essence of the reduction concept. The main relationships of interest are preserved while transforming correlated variables into uncorrelated variables.

SVD identifies the sorted dimensions with the highest information value. Mathematically it means factoring matrices into a series of linear approximations to see the structure of the matrix. This is done by finding the eigenvalues and eigenvectors of AA^T and A^TA. This gives us "three matrices *U*, *V* & *S* where the eigenvectors of A^TA make up the columns of *V,* the eigenvectors of AA^T make up the columns of *U*. and the singular values in *S* are square roots of eigenvalues from AA^T or A^TA. The singular values are the diagonal entries of the *S* matrix and are arranged in descending order. The singular values are always real numbers. If matrix *A* is a real matrix, then *U* and *V* are also real."

1.2. General notation and mathematical model

U : m x n matrix of the orthonormal eigenvectors of AA^T
V^T : transpose of a n x n matrix containing the orthonormal eigenvectors of A^TA.
$\sum$: a n x n diagonal matrix of the singular values which are the square roots of the eigenvalues of A^TA.

1. S_1 is the first principal component with the highest variance
2. The second PC is the linear combination with the second largest variance and orthogonal to the first PC, and so on
3. These PCs are the ones being disregarded under a specific threshold
4. To have the same dimension, we need to scale the variables by standardizing them into mean zero and standard deviation one.

The mathematical model formulated and explained by (S. Sharath et al., 2011) is given below:
"Let *A* is m' X n' real matrix and $N = A^TA$

Figure 1

Range and Null space of the matrix

R denotes the range space and N denotes the null space of a matrix. The rank of *A*, A^T, A^TA, AA^T is equal and is denoted by ρ orthonormal basis v_i $1 \le i \le \rho$ are sought for $R_A{}^T$ where ρ is the rank of $R_A{}^T$& u_i $1 \le i \le \rho$ for R_A such that,

$$AV_j = S_j U \tag{1}$$

$$A^T U_j = S_j V_j, \qquad 1 \leq j \leq \rho \tag{2}$$

The advantages of having such a basis are that geometry becomes simplified and gives a decomposition of *A* into *ρ* one-ranked matrices. Combining the equations (Eq. 1) & (Eq. 2).

$$A = \Sigma S_j V_j U_j^T, \qquad 1 \leq j \leq \rho \tag{3}$$

If V_j is known then, $Uj = (1/Sj)AVj$ $|Sj| = \| AVj \|$ therefore, $s_j \neq 0$, choosing $s_j > 0$,

$$AV_j = S_j U_j \tag{4}$$

$$A^T AV_j = S_j A^T U_j \tag{5}$$

$$A^T AV_j = S_j^2 V_j \tag{6}$$

Let $S_j{}^2 = \mu_i$, $N\ V_j = \mu_i V_i$ has required U_i's as orthonormal eigenvectors of $N = A^T A$ are found and $S_j = \sqrt{\mu_i}$. Where $\mu_i > 0$ are eigenvalues corresponding to V_j. The resulting U_i span the Eigen subspace. When SVD is applied to the sample set below in figure 2, the corresponding eigenfaces obtained are shown in figure 3. The figure highlights the holistic features of the given sample set.

Figure 2

Training set example faces

Figure 3

Eigenfaces from SVD
Basis selection from SVD

If A is the face Space, then x vectors are drawn from $[X_1 \ldots X_x] = \Pi_{<1..x>}(A^{-1}UD)$, where U & D are the unitary and diagonal matrices of SVD of A.

2. Practical case – Project

2.1 Plan of Activities

1. Loading Data
2. Data Set Manipulation
3. Setting data into two sets
4. Face Recognition in the Original Image Space
5. Recognition of each Image
6. Singular Value Decomposition (SVD)
7. Selection of the Dimension p
8. Projection on the Reduced Face Space

"The goal of this project is to develop a real-time system for face recognition using dimensionality reduction techniques. The cost (in terms of time and memory) to identify a face is greatly reduced by representing face images in a reduced subspace (called the "face space"), instead of using the original "image space" of m × n pixels."

2.2 Coding

2.2.1 Loading Data

First, we begin by loading the images from the GitHub library, scraping them using beautiful soup (python library), and preparing the links and file names using regex – we have 165 images. Each of these has a size of 243 x 320.

A sample of our images' dataset as the following - 12 images:

Figure 4

These images are called: subject 1, 2, 3 ... etc. with expressions of wink, sleepy, angry, etc.

2.2.2 Data Set Manipulation

We transformed these images into vectors by defining a function called img2vec using the reshaping method. We also did the opposite by defining the vector to image function and returning the image in the new dimension.

We proceeded by constructing the matrix by defining the function img2mat, enumerating through faces, vectorizing each one of them, and concatenating to construct the matrix called S.

After converting faces to vectors, we get dataSet: (77760, 165).

We get the Mean Face $\bar{f}$ by summing all the vectors and dividing this sum by N numbers of the original images (165), this is done for later normalization of the images.

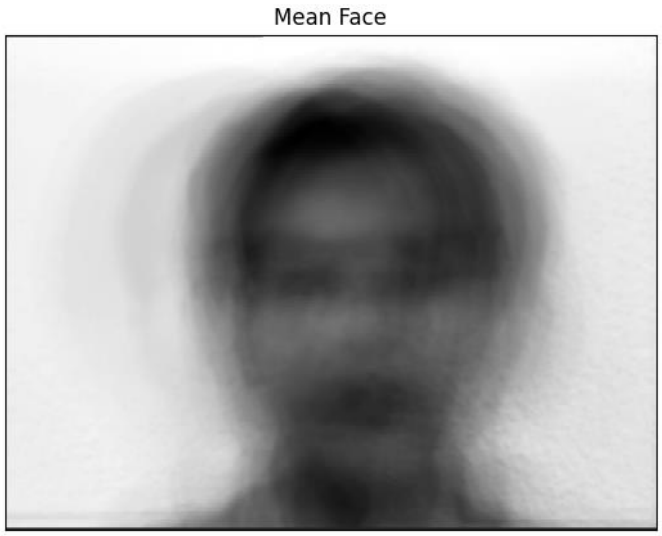


Figure 5

2.2.3 Setting Data into Two Sets

Before we divided our data set into two sets, we shuffled this data set to make sure there is not one item of data repeated and its subjects are evenly distributed over the whole data set.

By dividing the data in the ratio of 75:25, this means out of 165 we use 123 pictures for Database *F* and 42 pictures for Query Set *Q*.

We express this concept in two sets with the following dimensions:

F : (77760, 123)

Q : (77760, 42)

Here comes the face recognition part using the 1-nearest neighbor classifier.

2.2.4 Theoretical Into for the 1-Nearest Neighbor Classifier

It classifies the new data point (query data point) based on its distance in space to all of the data points from F which are labeled with the name of the subjects (classes).

Then, it assigns a label or a class to the new data point based on the nearest data point from F.

It is the simplest classifier; it uses either Euclidean spaces in the case of continuous variables or a hamming distance in the case of discrete variables to calculate the spatial distance.

2.2.5 Methodology of Prediction

Using a function code called query, a prediction is done for one image, this function is called by another function called recognition, to recognize a group of images.

We start with one face recognition; we call the same previous functions query and recognition to recognize only one image.

The results we get:

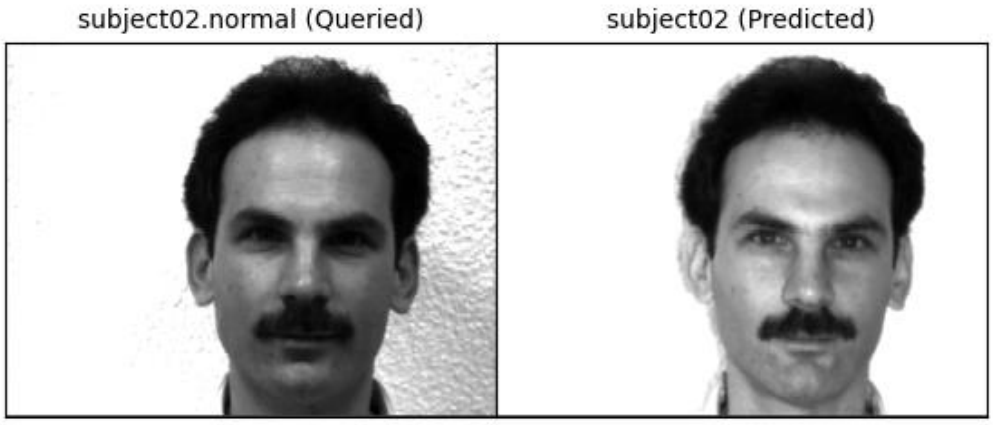
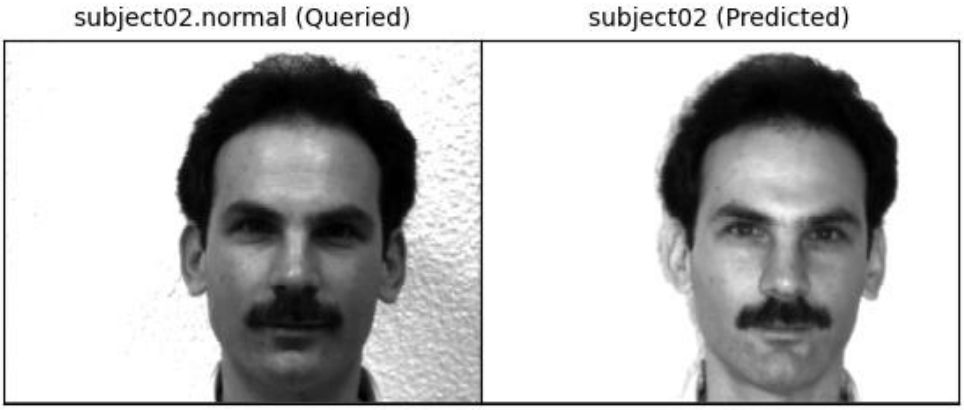


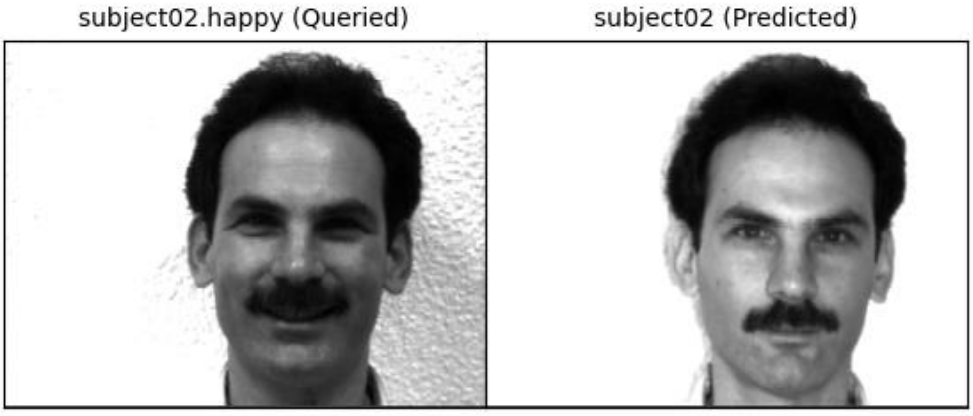
Figure 6

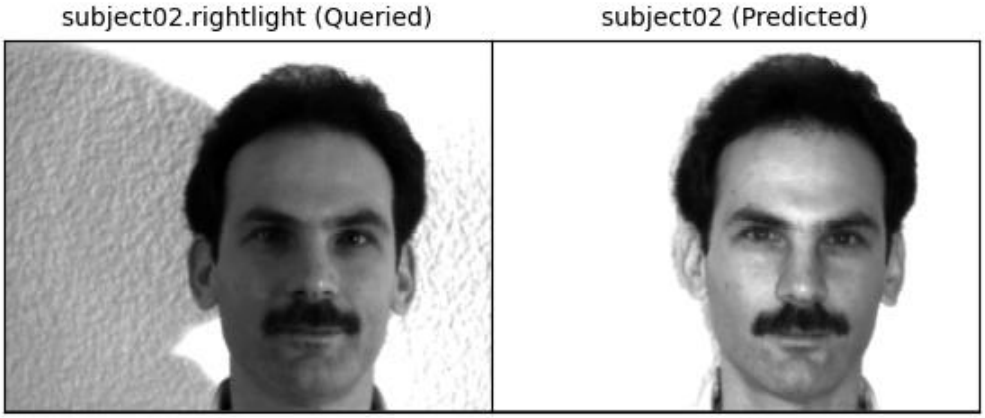
```
Number of Queried Pictures : 1
Number of Matches : 1
Subject Accuracy : 100.0%
The closest match in the database F is subject02.

Maximum Memory Usage (RAM): 146.01921558380127 MB
Execution time: 0:00:00.175288 [hh:mm:ss]
```

The next case is using 5 images at once, with the same approach:







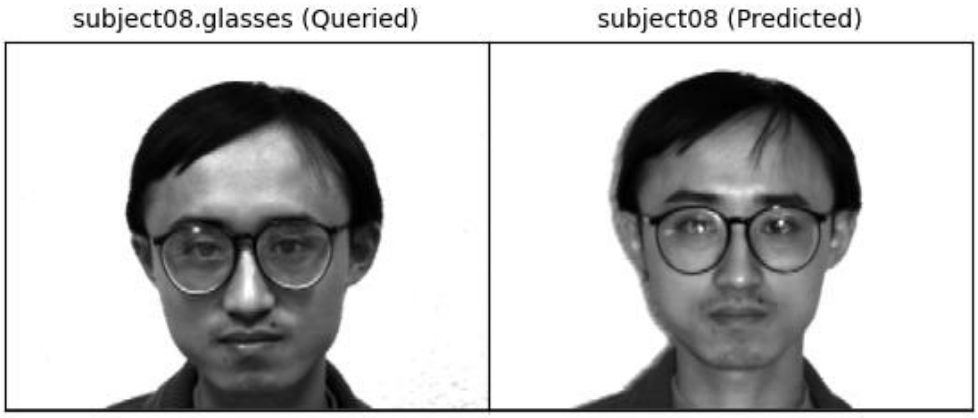


Figure 7

```
Number of Queried Pictures : 5
Number of Matches : 5
Subject Accuracy : 100.0%

Maximum Memory Usage (RAM): 148.44360637664795 MB
Execution time: 0:00:01.331530 [hh:mm:ss]
```

2.2.6 Recognition of all the images in Q

Next, we queried the whole set Q (42 images). We get the following result.

```
Number of Queried Pictures : 42
Number of Matches : 35
Subject Accuracy : 83.33%

Maximum Memory Usage (RAM): 146.02006149291992 MB
Execution time: 0:00:03.449265 [hh:mm:ss]
```

Again, we repeat the same previous case but with a ratio of 80:20.

```
Number of Queried Pictures : 33
Number of Matches : 28
Subject Accuracy : 84.85%

Maximum Memory Usage (RAM): 156.69980239868164 MB
Execution time: 0:00:02.843214 [hh:mm:ss]
```

More attempts, with a ratio of 60:40.

```
Number of Queried Pictures : 66
Number of Matches : 52
Subject Accuracy : 78.79%

Maximum Memory Usage (RAM): 117.54290962219238 MB
Execution time: 0:00:04.332699 [hh:mm:ss]
```

For individual ratios, it can be observed that with a larger ratio for set F, the prediction is more accurate and vice versa. The prediction accuracy for individual ratios is as follows: 78.79% (60:40), 83.33% (75:25), and 84.85% (80:20).

2.2.7 The coding for SVD Part

Even before proceeding to the SVD itself, we normalized our entire data set by subtracting the average shape from each shape. Then we shuffled our data set and divided it into set F and set Q in the ratio 75:25.

We have decided to calculate the SVD of the F database and not the whole data set including the query set Q. The goal of developing this algorithm is to be deployed for use in the real world in real-time. The query set Q is our testing set where we want to get to know the accuracy of new examples that are not occupied in the database F.

We initially used the full NumPy.linalg.svd for SVD calculation. After calling this function, this function immediately started to fill the computer's operating memory (32GB), then the SSD started to work at 100% and after about 5 minutes of calculation, the computer stopped responding to any commands. After researching the scientific article "Facial Recognition with Singular Value Decomposition" (The Golub-Reinsch Algorithm was used), we found that we can use the reduced SVD which does not require a lot of operational memory.

There is an alternative library called the Dask library. This library has been developed to handle larger matrices with more than 10,000 parameters. This library is very versatile with the support of computer parallelism. For this reason, we chose SVD from the Dask library, specifically its NumPy equivalent called Array. This library computes the reduced SVD of the matrix F approximately in 47 milliseconds compared to NumPy reduced SVD which can compute the same in 935 milliseconds (approx. 20 times slower).

2.2.8 Quick Mathematical Explanation

$$F = U\Sigma V^T \tag{7}$$

The matrix *F* is decomposed into the above three matrices, *U* and V^T are square unitary orthogonal matrices, and ∑ is diagonal. U has the same shape as *F* and *U*'s columns are the eigenfaces/eigenvectors of our system.

$$U^T U = UU^T = I \tag{8}$$

$$V^T V = VV^T = I \tag{9}$$

$\sum$ has $\sigma_1 >= \sigma_2 >= \sigma_3 \ldots >= 0$, so each column is more important than the columns after it, they are all singular values.

The interpretation of each vector of *U* is eigenvectors. This indicates that most of the information is stacked in the *p* first σ's.

2.2.9 Coding Part

```
U, Sigma, Vt = np.linalg.svd(F, full_matrices=False)
```

Here, we specify the full matrix as False because the full matrix means the full SVD of F which would overflow the operating memory of the computer. It is called Nympy reduced and this line of coding has taken:

```
Maximum Memory Usage (RAM): 73.1014757156372 MB
Execution time: 0:00:01.263223 [hh:mm:ss]
```

Next, we found a more efficient way of computing SVD in the Dask library. The reduced SVD, by computing the three matrixes using it:

```
u, sigma, vt = da.linalg.svd_compressed(a, F.shape[1])
```

We obtain:

```
Maximum Memory Usage (RAM): 0.6948003768920898 MB
Execution time: 0:00:00.053894 [hh:mm:ss]
```

It is obvious that it consumes much less memory and an excellent time compared to the previous one.

If the NumPy array is used for numerical calculations in the entire program, then using the Dask Library loses all the advantages of speed and efficiency. This is caused by converting the resulting structures into NumPy arrays, which is time-consuming (3.7 seconds in our case).

By checking the shapes of all matrixes involved in computation:

```
F: (77760, 123)
U: (77760, 123)
Σ: (123,)
Vᵀ: (123, 123)
UΣVᵀ: (77760, 123)
```

This check shows that the mastics from SVD have the correct dimensions.

2.2.10 Eigen Faces

Now, we check the eigen faces by potting them, this is done by looping through the *U* matrix:

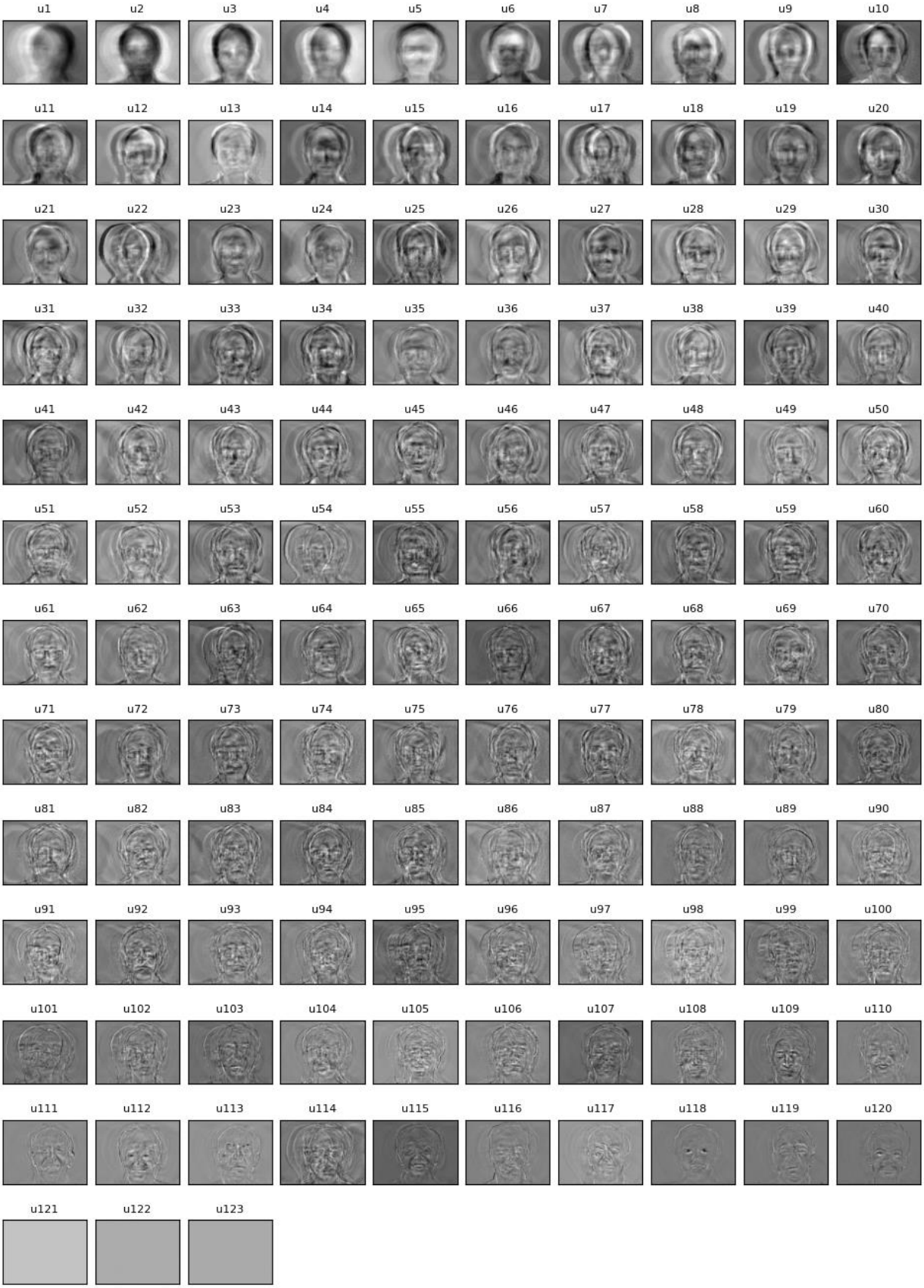


Figure 8

2.2.11 Selection of the Dimensionality *p*

Now, we plot the singular values distribution and their cumulative sum or energy, this will aid us to determine where we want to cut down space or reduce dimension:

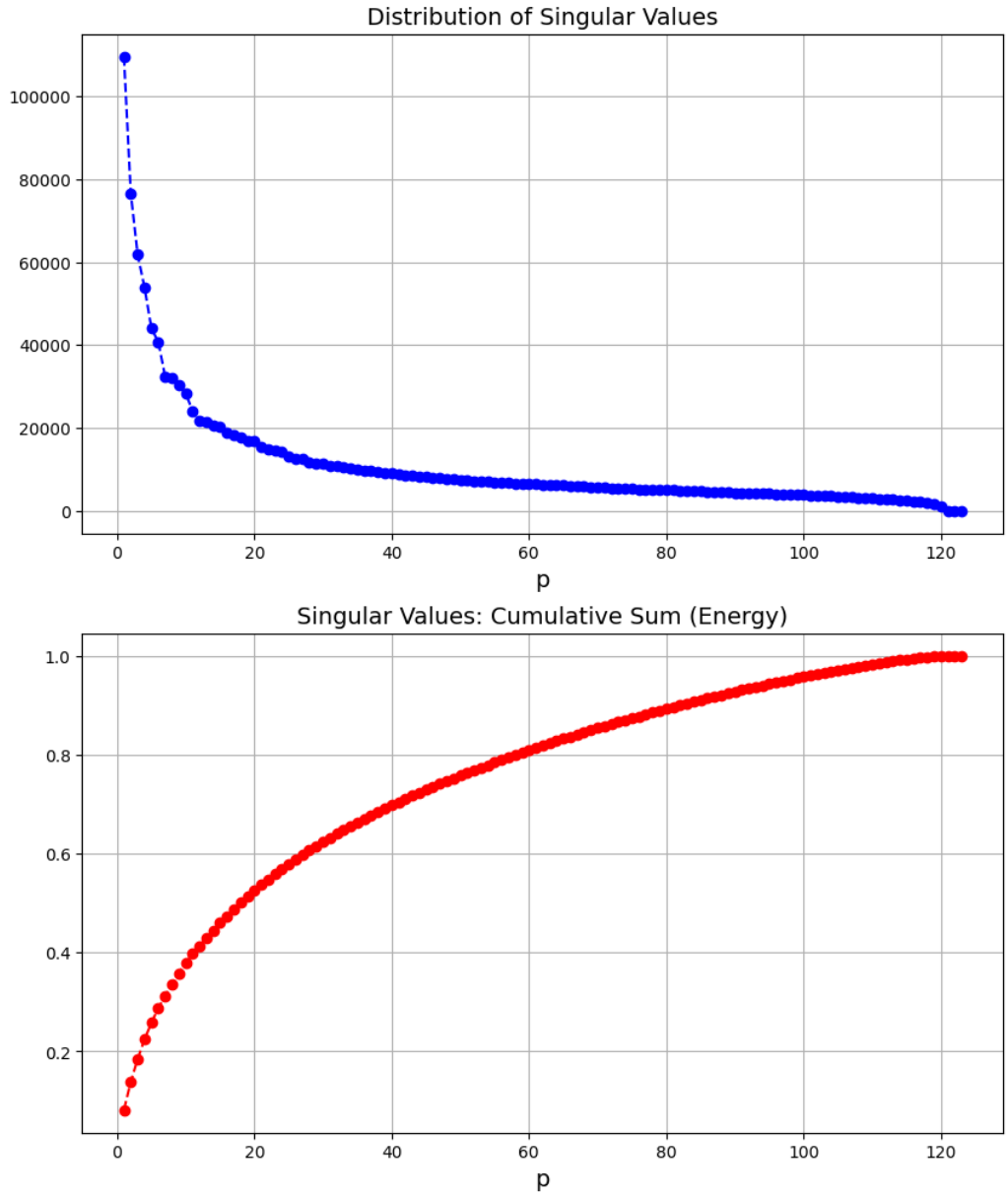


Figure 9

By choosing the first half 50% of the cumulative sum, we reduce the whole dimension to only p = 18, the information is mostly in the very first columns.

2.2.12 Projection on the Reduced Face Space

Now, we reduce the face space *U* to 18, and we obtain Base-Face $\widetilde{U}$. With this projection we have our matrices reduced to the following:

- The Base-Face $\widetilde{U}$ is (77760, 18)
- Reduced Database $\tilde{F}$ (18, 123)
- Reduced Query Set $\tilde{Q}$ (18, 42)

Base-faces:

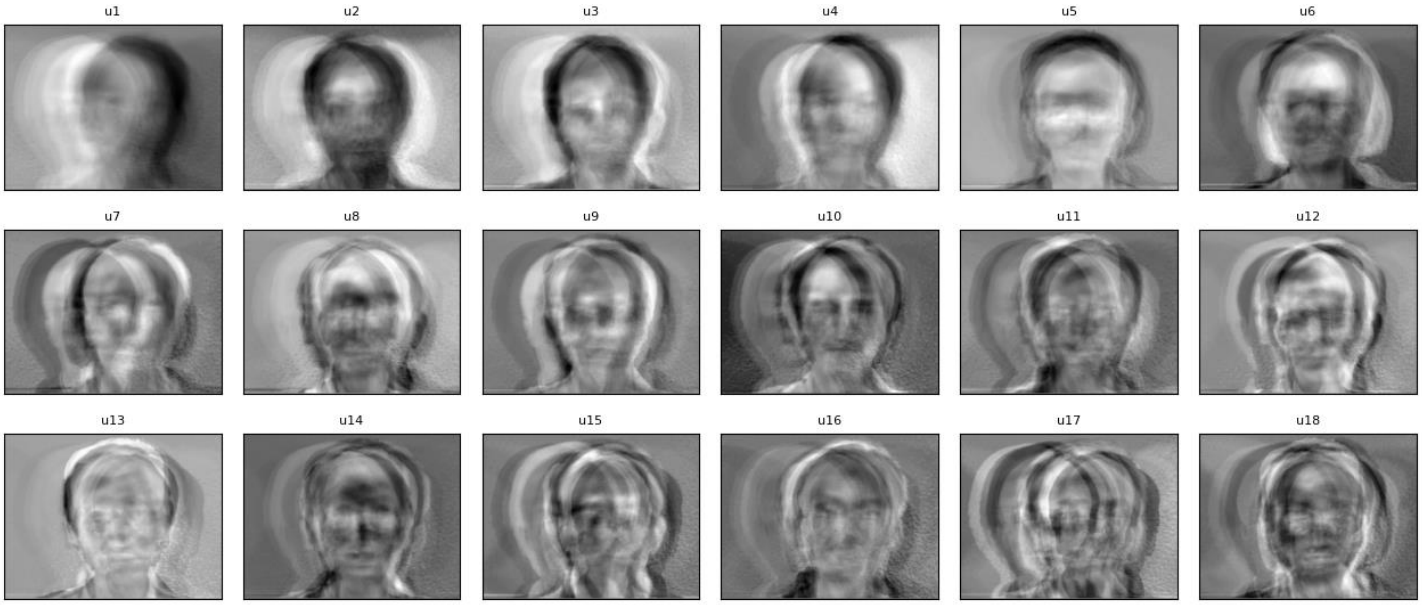


Figure 10

The step of applying the recognition function we used previously using the reduced matrices:

```
recognition(Q_reduced, F_reduced, pictures = False)
```

We obtain:

```
Number of Queried Pictures : 42
Number of Matches : 30
Subject Accuracy : 71.43%

Maximum Memory Usage (RAM): 0.06630802154541016 MB
Execution time: 0:00:00.058737 [hh:mm:ss]
```

The new accuracy in the reduced $\tilde{Q}$ set is 71.43%, which represents a decrease in accuracy compared to the unreduced *Q* set by approximately 12% from 83.33%.

We recognize one reduced face:

Number of Queried Pictures : 1
Number of Matches : 1
Subject Accuracy : 100.0%

Maximum Memory Usage (RAM): 0.06424617767333984 MB
Execution time: 0:00:00.000998 [hh:mm:ss]

2.2.13 Accuracy Based on p

Now we test the accuracy based on p, and p is the 50% of *F* we chosen previously, and *F* is the whole matrix and apply the recognition function for each face:

```
P = list(range(1, F.shape[1] + 1))[::-1]
accuracy = []

for p in P:
     U_base = U[:,:p]
     F_reduced = U_base.T @ F
     Q_reduced = U_base.T @ Q
     accuracy.append(recognition(Q_reduced, F_reduced, pictures = False,
print_report = False))
```

We plot the outcome.

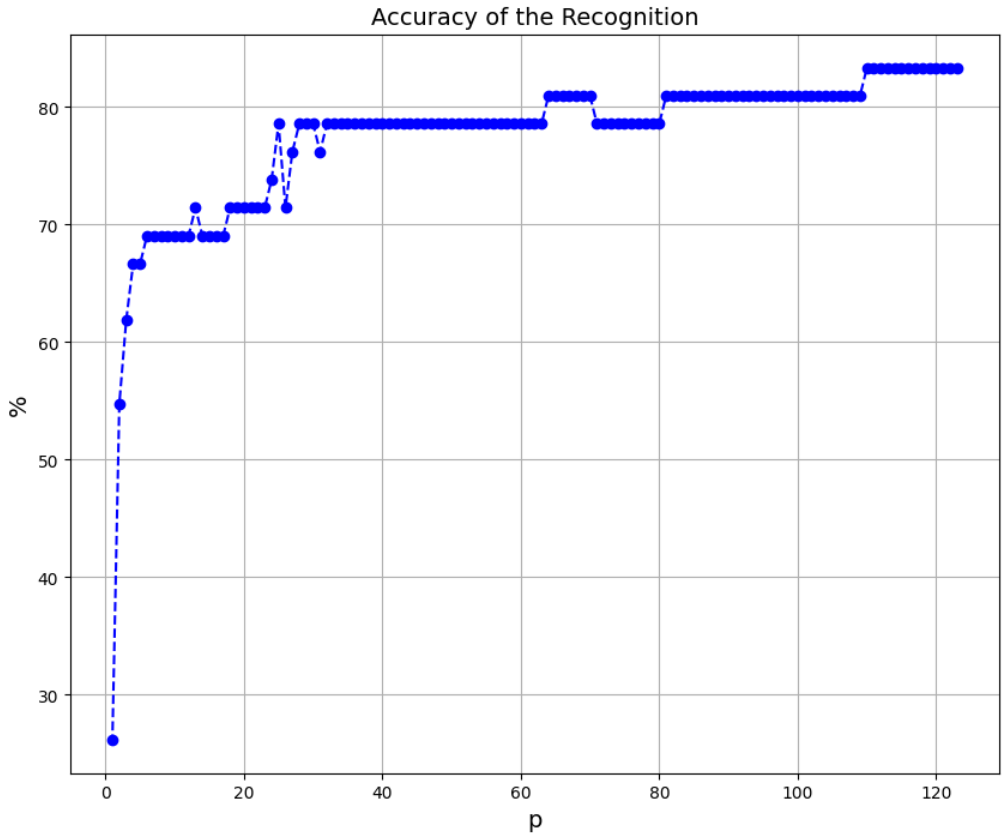


Figure 11

The given graph is based on the change in p from the value 1 to the value 123 (unreduced *F*). It can be observed that as *p* changes, so does the accuracy, and in general, it can be said that the higher the *p*, the higher the accuracy. There are some exceptions to this statement that can be seen in the thesis graph. A feasible compromise between accuracy and dimension reduction is to choose p around 65.

2.2.14 Extra Activities

In this part, we programmed our SVD function.

This function looks like this::

```
def SVD(A):
    '''
       SVD : Singular Value Decomposition
    '''

    # AᵀA
    AtA = A.T @ A

    # Eigen values and eigenvectors of AᵀA
    S2, V = np.linalg.eig(AtA)

    # Vᵀ
    Vt = V.T

    # Singular values
    S = np.sqrt(S2)

    # U = A V
    U = A @ V

    # Normalization of U
    U = U/ np.linalg.norm(U, 2, axis=0)

    return U, S, Vt
```

This is an almost straightforward computation, when calling u, s, vt = SVD(F).

This requires the following computational resources:

```
Maximum Memory Usage (RAM): 438.1795129776001 MB
Execution time: 0:00:00.308770 [hh:mm:ss]
```

Compared to the SVD of standard libraries such as Numpy or Dask, our SVD implementation with 309 milliseconds is in the middle between the SVD of these libraries. In terms of RAM consumption, it is the worst with 438 MB. From the point of view of the consumption of operating memory, it would be necessary to think about the optimization of the mentioned implementation. One candidate for this implementation can be Randomized Singular Value Decomposition by N. Halko, P. Martinsson, and J. A. Tropp published in 2011.

In the next part, we focused on the average face of each individual.

Without normalizing, we took original database F, we list names of subjects or faces, we find the average face for each of them – in case he/she has more than one face, by plotting these averages we get:

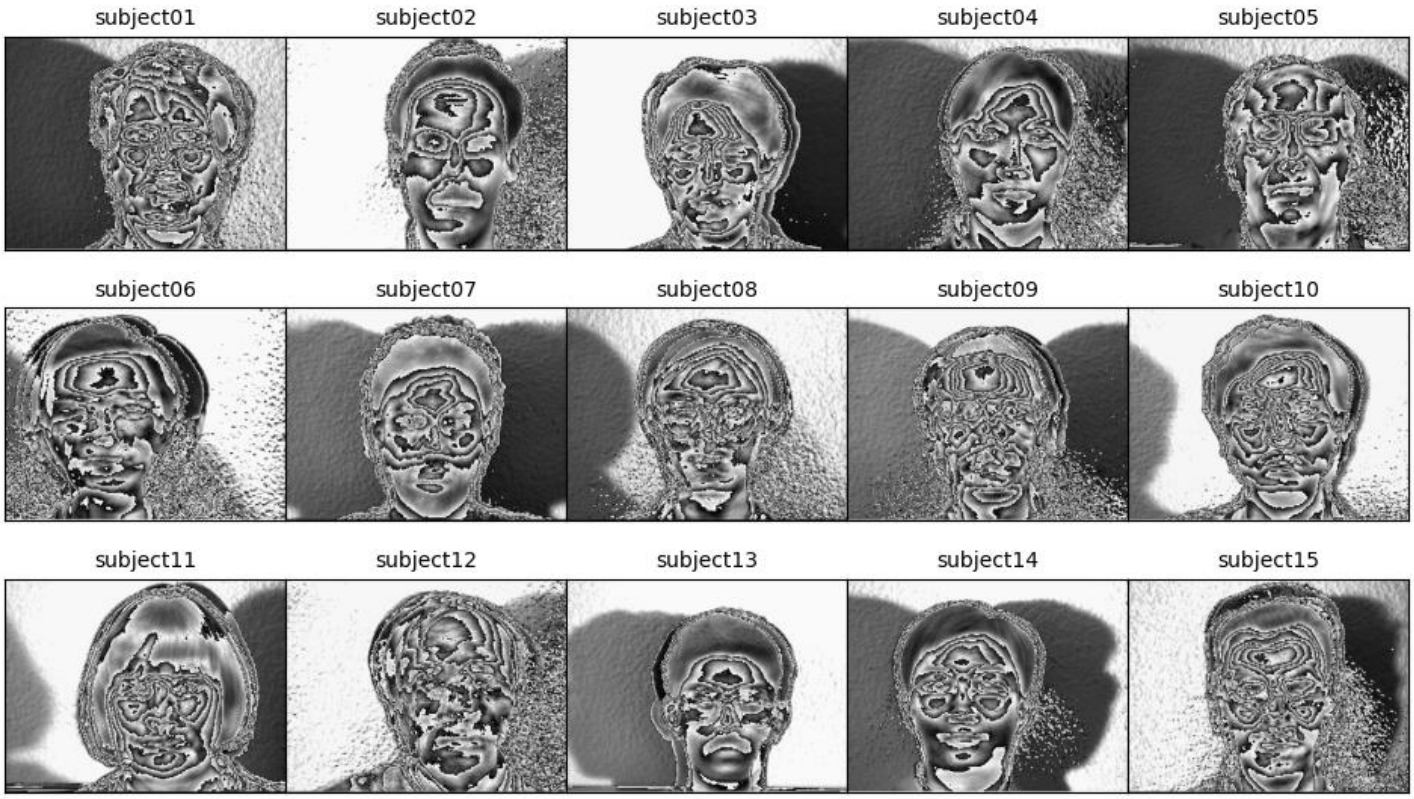


Figure 12

This requires the following computational resources:

```
Number of Queried Pictures : 42
Number of Matches : 5
Subject Accuracy : 11.9%

Maximum Memory Usage (RAM): 17.87422275543213 MB
Execution time: 0:00:00.411459 [hh:mm:ss]
```

We calculated the average face for each individual from database F. Then we performed recognition. The memory requirement decreased, but the accuracy decreased to 11.9%.

3 Conclusion

After investigating multiple methods and cases for SVD computation and recognition of faces in real-time we discovered several findings.

If we only focus on face recognition without SVD, then this is not very suitable for real-time applications. This is because the mentioned 1NN algorithm needs the entire database F for its functionality. Although our database F is not very large, it contains 123 faces, so recognizing faces in set Q takes 3.2 seconds and 146 MB. These time and memory requirements are closely related to the sizes of the sets F and Q. If we increase the ratio between Q and F in favor of F, then the accuracy increases.

In the case when we included SVD, the time and memory requirements of face recognition in the Q set dropped dramatically. Specifically, the memory requirement dropped from 146 MB to 0.07 MB (2386 times less - better) and the time requirement dropped from 3.2 seconds to 0.015 seconds (215 times faster). It is clear from these data that if computing resources are limited for any reason or real-time face recognition is required, then SVD should be involved.

The inclusion of SVD has one major weakness, and that is reduced recognition accuracy. This can be partially compensated by a suitable selection of the parameter p when there is not such a large reduction in accuracy or another recognition algorithm. This algorithm can be a more sophisticated classifier or a deep neural network.

It should not be overlooked that there is a whole range of SVD algorithms out there. We can name, for example, Hestenes' Algorithm, Golub-Kahan Algorithm, Tridiagonalization and Symmetric QR iteration, Tridiagonalization and Divide and Conquer Algorithm, and Bisection and Inverse Iteration. Among the mentioned SVD variants, the Divide and Conquer Method for real-time face recognition shows great promise (Pradhan, 2013).

SVD plays an important role in image recognition and can fundamentally reduce computational requirements thus speeding up the recognition process. This has a crucial role, especially in real-time systems.

References

- S. Sharath, S., Murthy K N, B. & S, N. (2011). Dimensionality Reduction Techniques for Face Recognition | IntechOpen. Reviews, Refinements and New Ideas in Face Recognition, 3. 10.5772/18251, Retrieved from: www.intechopen.com/chapters/17174

- Large SVDs: https://blog.dask.org/2020/05/13/large-svds

- NVIDIA. (n.d.). What is Dask? NVIDIA Data Science Glossary. Retrieved December 22, 2022, from https://www.nvidia.com/en-us/glossary/data-science/dask/

- Measuring the allocated memory with trace malloc: https://stackoverflow.com/questions/70525623/measuring-the-allocated-memory-with-tracemalloc

- Randomized Singular Value Decomposition; "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions", Nathan Halko, Per-Gunnar Martinsson, Joel A. Tropp, 2011: https://arxiv.org/abs/0909.4061

- Pradhan, T., Routray, A., & Kabi, B. (2013). Comparative Evaluation of Symmetric SVD Algorithms for Real-Time Face and Eye Tracking. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-30232-9_13