University of Colorado at Boulder

APPM 3310

Matrix Methods

Facial Recognition Through Singular Value Decomposition

Author:	*Student ID & Lecture:*
Cooper Simpson	106067886 & 001
Jake Schroeder	105870933 & 001
Cole Sturza	107465166 & 002
Tashi Wischmeyer	108185908 & 001

December 12, 2018

Contents

I. Abstract

Singular value decomposition (SVD) is an extremely powerful and versatile matrix factorization. We present an application of SVD in a rudimentary facial recognition program. Using SVD, we are able to break down images and create a means for efficiently storing data that can be used in the processing of images for recognition. We describe the technique of numerically analyzing images to create the facial recognition system. We discuss the results of this application, its effectiveness, its drawbacks, and how it might be further extended beyond the scope of this project.

II. Attribution

Cole Sturza, Cooper Simpson, Jake Schroeder, and Tashi Wichmeyer all wrote sections of the Matlab code, helped gather data, and aided in the preparation of this report.

III. Introduction

This project explores the applications of singular value decomposition in facial recognition. We seek to explore the capabilities of this matrix factorization in identifying images as being a member of a database. Facial recognition has many uses in various fields; the question is: How does one recognize a face? Furthermore, how does one do this efficiently and effectively? While facial recognition might be something that humans do amazingly easily many times a day, it is not something that is immediately transferable to an algorithmic program. Techniques in linear algebra provide a means for approaching this problem. Expressing an image as a matrix and using SVD to break it down into principal components one can then develop a solution to the facial recognition issue. In this report, we outline the mathematical ideas and tools behind our process of facial recognition. We will describe the structure of SVD and how it provides the necessary information needed for facial recognition. Furthermore, we will discuss our algorithms for analytic facial recognition and the results of our numerical studies. We will explore the capabilities of our end result program as well as its faults. Altogether, we will expound upon the process and implementation of facial recognition as we have accomplished.

IV. Mathematical Formulation

A. Singular Value Decomposition

The crux of the material presented here lies with singular value decomposition and its components. The formulation is given below for an $m \times n$ matrix A, where U and V^T are orthogonal, and Σ is diagonal [4]. The discussion is for matrices that are real.

$$A = U\Sigma V^T \tag{1}$$

U is $m \times m$, and V^T is $n \times n$ with Σ $m \times n$. If one rearranges the SVD equation then one may observe an important fact regarding the nature of U and V^T [2].

$$AA^T U = U\Sigma\Sigma^T \tag{2}$$

$$A^T AV = V\Sigma^T\Sigma \tag{3}$$

From this one can see that the columns of U are the eigenvectors of AA^T, likewise the columns of V are eigenvectors for $A^T A$. Furthermore, both of these matrices have the same eigenvalues given by the diagonal entries of Σ^2. This will have more meaning in later discussion, but has been introduced here. The diagonal entries of Σ are the singular values of A ordered in a non-increasing fashion. The columns of U and V may also be referred to as the left and right hand singular vectors of A [4]. In SVD, the number of singular values contained in Σ depends on the rank of A. For a matrix of rank r, there will be r singular values [4]. Because of this, zeros are inserted into the diagonal of Σ if the matrix is not full rank. It is also important to note that the first r columns of U form a basis for $Range(A)$ – its column space. In fact, the SVD of a matrix can be constructed in an alternate fashion, via decomposing the four fundamental sub-spaces of the matrix A [4] [2].

$$\left(U = \begin{bmatrix} \vdots & \vdots \\ R(A) & N(A^T) \\ \vdots & \vdots \end{bmatrix} \right) \left(V = \begin{bmatrix} \vdots & \vdots \\ R(A^T) & N(A) \\ \vdots & \vdots \end{bmatrix} \right) \tag{4}$$

In 4, one can see how U and V are formed which follows with SVD in that they are

orhtogonal matrices. The fact the U holds a basis for the column space of A will be important in our process for facial recognition [2].

One further topic related to SVD will be pivotal in understanding the methods we present later, the distance to lower rank matrices. For a matrix $A_{m\times n}$ with singular values $\sigma_1 ... \sigma_r$, the distance from A to the closest matrix of rank $k < r$ is given below [4].

$$\sigma_{k+1} = \min_{rank(B)=k} \|A - B\|_2 \tag{5}$$

Where the minimizer B_k is given by

$$B_k = U \begin{pmatrix} D_k & 0 \\ 0 & 0 \end{pmatrix} V^T \tag{6}$$

In 6, one can see that the matrix that minimizes the distance (B) is the matrix formed from the SVD of A by removing everything but the first k singular values from the diagonal matrix Σ. The distance obtained from this minimizer is the first singular value that was left out $(k+1)$ as seen in 5 [4].

This theorem tells us two important things. The first is that for a Σ with zeroes on the diagonal, it is perfectly safe to ignore them. The distance to the closest matrix with no zero singular values is zero, as seen from 5. This ties in with the second item of importance from the above equations. If one wants to approximate the matrix A with a matrix of rank $k < r$, and if this is to be done judiciously, then the singular values will tell one the error, in a sense, of removing values. The question to "How much information am I losing?" is answered by the magnitude of the distance to the approximation matrix given by the first excluded singular value in 5. This will allow one to reduce the size of stored information while still maintaining the output integrity – a concept vital to this project in facial recognition.

To further understand the information that SVD provides, it is useful to look at the covariance matrix. The covariance matrix is defined as $C = AA^T$ [2]. This matrix is symmetric and gives information about the variance in its column vectors, which are images in our application. One may note that this is exactly the matrix whose eigenvectors are the columns of U and whose eigenvalues are the squares of the singular values from the SVD decomposition. By computing SVD. one obtains the same information about the

variation of the data contained in the matrix obtained with the covariance. We do not use the covariance matrix explicitly in our process, but its features are implicit in SVD. It is noted here simply to describe that the problem may be approached from multiple angles while still reaching the same conclusions.

B. Application to Facial Recognition

Having discussed SVD and its structure in depth, how does it apply to facial recognition? Its purpose is best revealed in a discussion of the steps to process images for facial recognition. To use SVD in the recognition of faces, the first step is to build a training set of photos so as to be able to apply SVD. One can think of a photo as an $m \times m$ matrix, where each index contains numerical information about the pixels at that location. Different photos have different resolutions and thus different pixel densities. In order to apply SVD to a set of many images one must alter the structure so as to build a matrix capable of this decomposition in a meaningful way. To do this one stacks all of the columns of a particular image matrix on top of each other to create a $m^2 \times 1$ vector [3]. This effectively maps the image to a lower dimensional space. One can do this for each individual photo in the training set and then insert all of these vectors into a matrix. This matrix is a representation of all of the face images in the training set. It is important to note that the mean face vector should be removed from each individual vector as this represents information that all of the images contain and will not aid in identification [2]. Additionally, it is advisable to use a large and diverse data set because fundamentally, this captures all possible faces that could be created. After the matrix has been constructed, one may then compute the SVD of said matrix. Doing so provides one with the matrix U. As discussed previously the columns of U are the eigenvectors of the covariance matrix, and if the rank of the face matrix is r, then the first r columns of U form a basis for the range of A. Interpreting this for a matrix of faces, one observes that a basis for this matrix is a basis for all possible faces that can be created from the training set. SVD also provides one with the matrix Σ containing the singular values of A. By looking at the magnitudes of the singular values, one can get an idea of how much information from U needs to be retained. This allows one to discard any information other than the first k columns of U that correspond to an error σ_{k+1} that is acceptable for the situation. This is the desired subspace that can then be used for facial recognition. By projecting a known image of person onto this space, one

can get the eigenface representation – simply the projection onto this reduced subspace [2]. Taking an unknown photo one may project this onto the same space, and then compare the result with the known representations to determine if it is a face in the database. The projection is accomplished through equations below [2].

$$f - avg(f) = U_k y \tag{7}$$

Where f is a face vector, k is the number of columns kept from U, and y is the projection. Rearranging 7 gives us the equation for projecting – 8.

$$y = U_k^T (f - avg(f)) \tag{8}$$

y is the projection onto the subspace and is the relevant information for recognizing images [2]. If one takes a new image not in the database and computes its projection in the same manner, it can be compared to the known projections.

$$error = \|x - y_i\|_2 \tag{9}$$

Where x is the unknown projection, and y_i is the known projection. Applying the 2-norm to the difference of the two vectors gives an error [3]. Recognition comes if that error lies within an acceptable tolerance determined by the situation.

V. Examples and Numerical Results

A. Algorithms and Data Collection

We present our process for implementing facial recognition using SVD to a real life situation of identifying faces in images. To collect data for this project, we attended a Global Engineering RAP event and asked for volunteers to take pictures of. When collecting the data, consistency among all the photos was a high priority as explained earlier. It is necessary to get a well normalized and representative data set. We used a high quality DSLR camera and tripod to take all the photos, and had everyone sit in front of a white background, thus ensuring the same lighting, angles, and distance to subject were consistent among all photos. There were three types of photos we took of each individual: a plain faced photo, a smile photo, and a tilted face photo. Some people we had take a fourth

photo with glasses if they had them, and others took another with some sort of headgear (i.e. a hat or bandana). Examples of the three main types of photos can be seen below.

(a) Plain faced photo **(b) Smile photo** **(c) Tilted face photo**

Fig. 1 Three types of pictures we took of every person.

In order to begin the data analysis, each photo in what is defined as the training set database is converted to a 100×100 grayscale image in our Matlab function `addPerson`. While this is a sizeable reduction in image quality, it significantly improves computation time and overall data size. Each image is then converted to a column vector and concatenated to form a $10000 \times n$ matrix, where $n =$ the number of files/faces in the folder. To be able to more effectively capture the relevant information of each face vector, the function `separate` subtracts out the average value of the vector, which represents a uniform bias of all the faces.

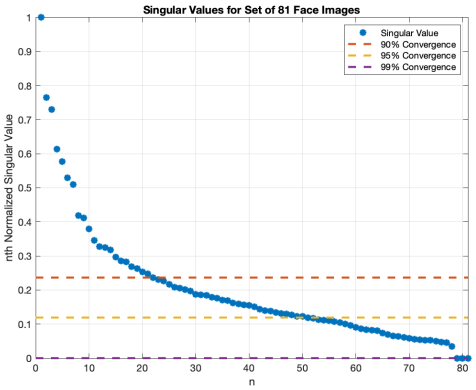
The next step in the process is to calculate the singular value decomposition of the face matrix and project each of the normalized faces, which is a fairly simple task with Matlab's built in tools. Our function (`eigStuff`) is called and is passed the normalized face matrix. It then calculates the SVD of the matrix and iterates over each column calculating the eigenface representation for each face. The "eigenface" representation for each face is found by projecting the individual face vectors that compose the face matrix onto the column space of U, which we obtained from calculating the SVD of the normalized face matrix.

In order to determine if an input image contains a face within the database, it is normalized by subtracting the mean face and converted to a vector just as the training database images were. Then it is projected onto the column space of A contained in the first k columns of U_k in the function `projector`. This generates a vector that is compared to the projections of each of the training set images. The closest values of each of the

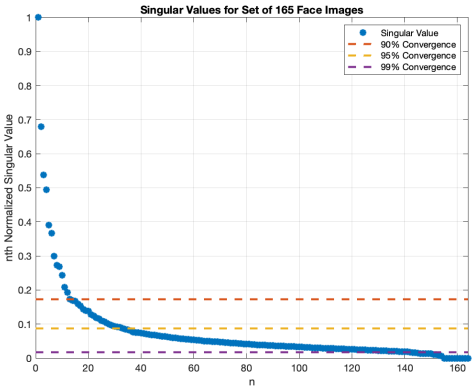
difference vectors with respect to the 2-norm represent a match, and is output from the function.

B. Results

1. Singular Values



(a) Our Set of 81 Pictures (27 Repeated Faces)



(b) Sample Set of 165 Pictures (15 Repeated Faces)

The above figures plot singular values for three different sets of data. Figure [2a] plots the singular values for the combined data set of all pictures our group took, meaning there are 27 different faces represented by 3 different poses: plain, smiling, and tilted. Figure [2b] is a similar plot of singular values, except it uses a completely different set of face images from the Yale Face Database [1]. While not as high quality as our images, the Yale Face Database provides about twice the number of images. Furthermore, these images contain 11 different poses of each subject rather than just 3. While the database images are not used in our facial recognition analysis, the plot of the singular values helps to illustrate the convergence of singular values in data sets with more similarity. The convergence of the singular values becomes more rapid as more facial images are added to the data set. This means that the larger singular values, which occur in decreasing order along the diagonal of the Σ matrix, contain more information about the data set. This is important to consider in the reconstruction of faces, where the eigenface training set projections can effectively be used as a way to compress an image of a face by deleting singular values after they've reached a desired level of convergence. However, this requires a significantly larger and more diverse training set than was available to us, so facial recognition is focused on rather

than image compression using eigenfaces.

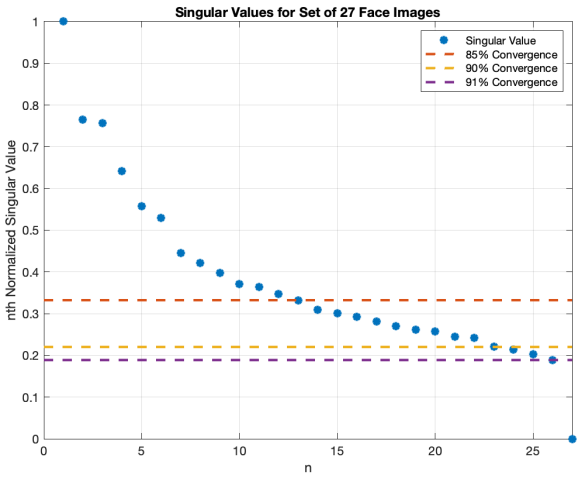


Fig. 3 Our Set of 27 Pictures (No Repeated Faces)

Figure [3] represents the singular values for the training set used in the facial recognition algorithm. Here, the "plain" pose for each of the 27 faces is used. In comparison with the larger data sets, the convergence of the singular values is much slower, largely because every face used is different and therefore provides significant information to the small training set. Extrapolating on this concept, if the training set were significantly larger, each new face added to the set would contribute less information to the set as a whole, which is illustrated as a more rapidly converging set of singular values.

2. Recognition

In order to recognize a face that corresponds to a face in the training set, the photo in question is projected onto U_k, an orthonormal basis of the training set derived from the singular value decomposition. k represents how many columns from U are kept according to the information found in Σ. All the photos in the training set have also been projected onto the same subspace. The error between the photo in question and each of the training set photos is then found. Therefore, it follows that the resulting error is significantly smaller for images that are more similar, which implies that images containing the same face will result in a much smaller error value. Where error is a measure of the 2-norm of the difference vector.

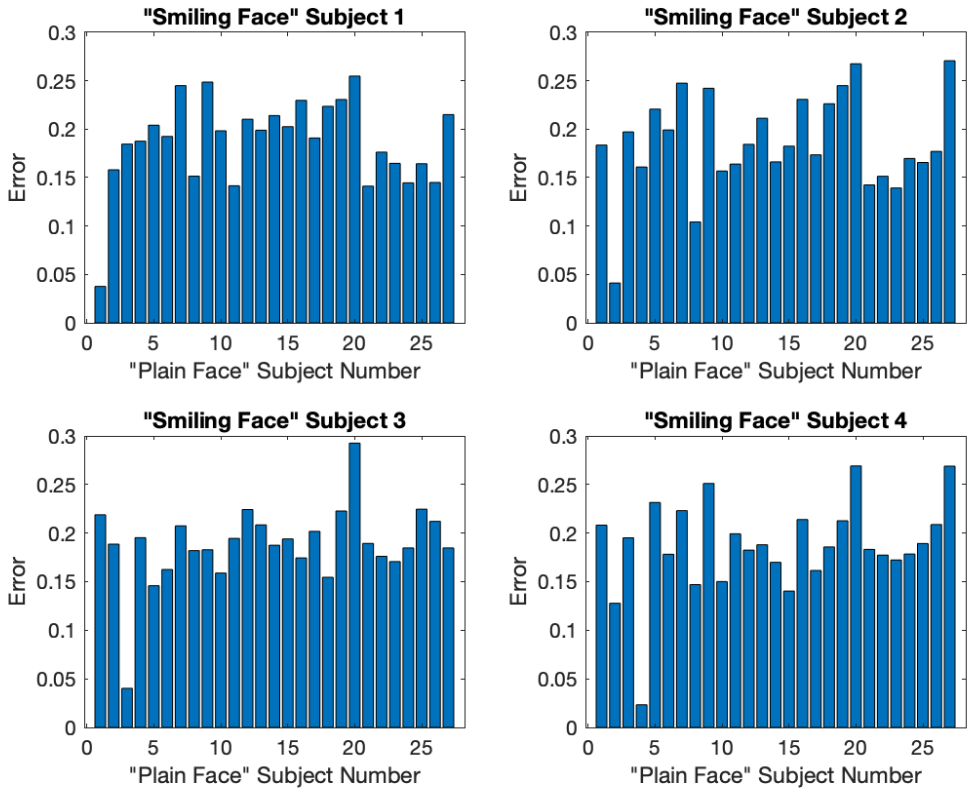


Fig. 4 Comparison of Smiling Subject Faces to Plain Subject Faces

Figure [4] shows the resulting error values for four input images that contain a subject who is also in the training set, but is smiling instead. Looking at the bar graph whose input photo is of subject 1 smiling, the error associated with the "plain face" is much lower corresponding to subject 1 than with any other subject. This effectively identifies the input image of the person smiling as subject 1, which is known to be true. The same test is done in the following bar graphs for the first four subjects. The bar that is significantly lower than the rest corresponds to the subject whose face is most similar to the face in the input image. It can also be reasoned that if the input image were also used in the training set, there would be an exact match, and the error for the corresponding subject would be 0.

Figure [5] is a representative plot of what would happen if an input image is not of any person represented in the training set. Clearly, there is no significant decrease in error for any individual subject, meaning the facial recognition is inconclusive. In this specific case, subject 27 is removed from the training set. Then, the image of subject 27 smiling is used as the input. It can be concluded that the error corresponding to a face outside the training

set is essentially the same as the error for two faces that do not match up.

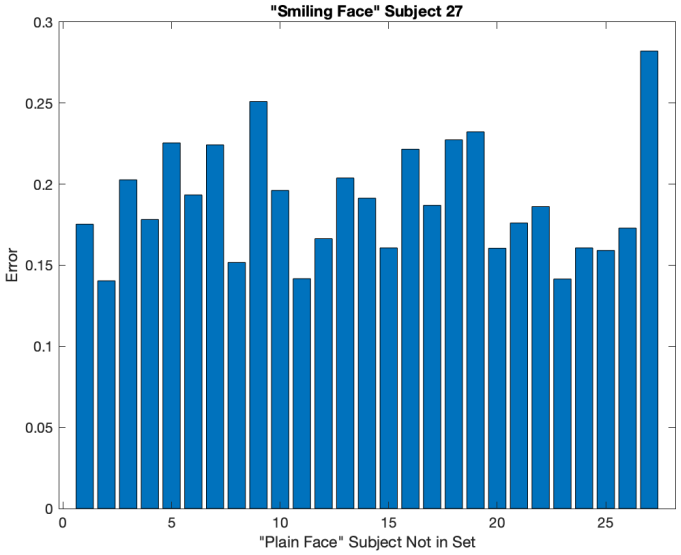


Fig. 5 Comparison of Smiling Subject Face Not in Set to Plain Subject Faces

To quantify whether or not a new face can be matched to a preexisting subject in the database, each new photograph is projected onto U_k, as before. The resulting vectors are then compared to the training set, producing an error, which is subsequently normalized and stored as the columns of a matrix. The diagonal elements of this matrix, represented as black points below in Figure [6], are the normalized error that results from the comparison of the new photo and the training photo of the same subject. The maximum of these values is the largest error between two photos of the same subject. The off-diagonal elements of this matrix are the errors produced from comparing the training set photo from one subject to the new photo of a different subject. These errors are shown as red points below in Figure [6]. The minimum of these values is the smallest error between subjects in the set.

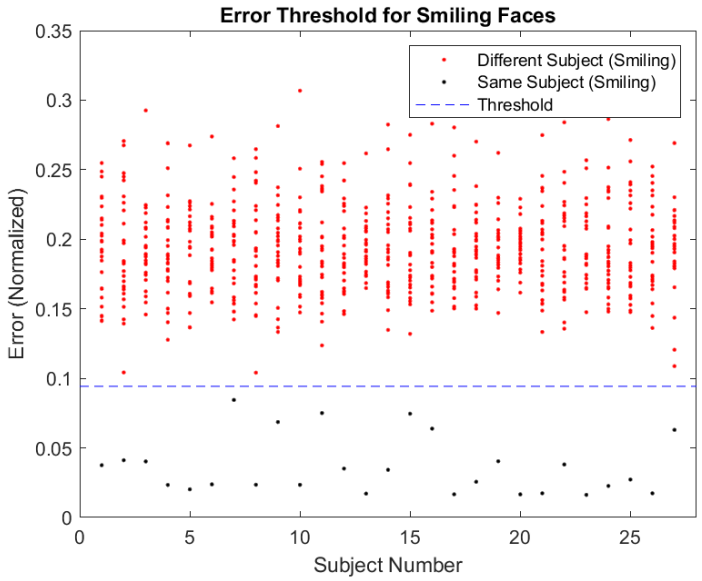


Fig. 6 Comparison of Smiling Faces with Faces in Training Set

A threshold for recognition can then be defined, rudimentarily, as the value that is equidistant to the maximum same-subject comparison and the minimum different-subject comparison. Any new photo comparison that generates an error above the threshold for a given subject will not be recognized as that subject. The threshold value for this data set, pictured above as the blue line in Figure [6], is 0.0943, allowing an error margin of 9.43% for the different photos of a given subject. By inspection, this threshold distinguishes a face that quantitatively matches a subject in the training set from a face that is closer to one subject than any of the others.

Identifying a facial image that is relatively normalized in comparison with the training images provides for more accurate results. Problems arise when one tires to recognize a photo that has more variation in its structure (i.e. size, lighting, angle, etc.).

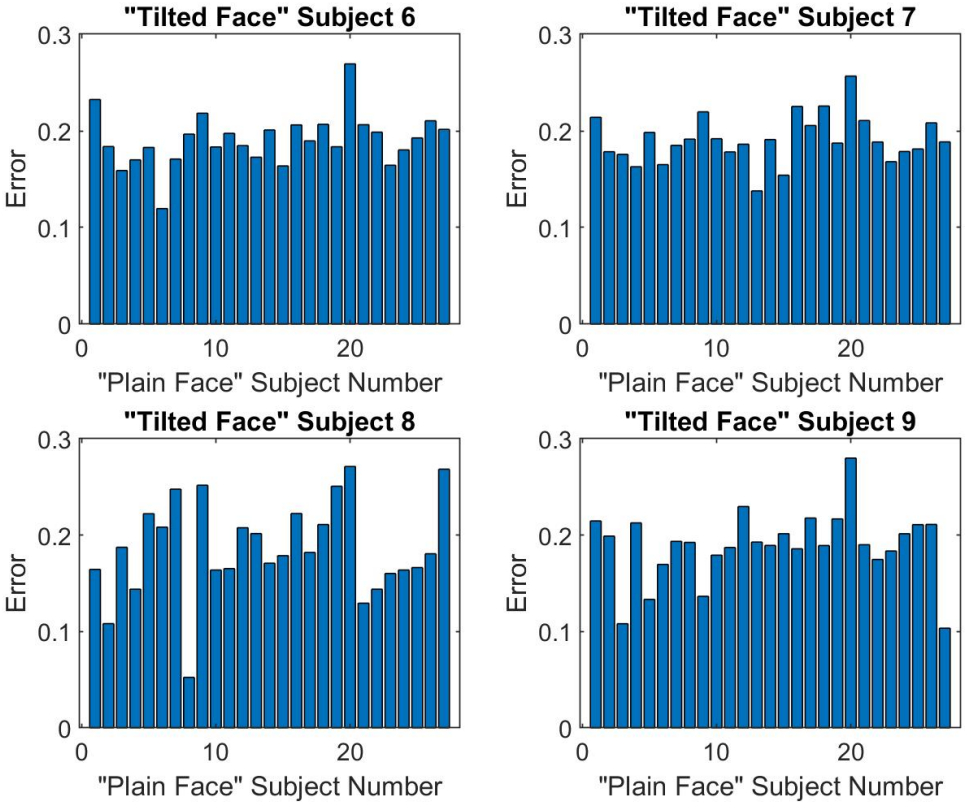


Fig. 7 Comparison of Tilted Subject Faces to Plain Subject Faces

In Figure[7] one can see the errors when comparing tilted face images with the training data for four subjects. When the tilted photos of our 27 subjects were processed in our program there were five incorrect identifications. Subjects 7 and 9 in Figure [7] are among these. It can be seen that for incorrect identifications there is no obvious minimum error. Even in subjects 6 and 8, which were correctly identified, the minimum error is still above the error threshold shown in 6. This shows how images that are not normalized with respect to the training set are much harder to identify.

(a) Tilted Faced Photo Subject 9 **(b) Misidentified as Subject 27** **(c) Misidentified as Subject 3**

Fig. 8 Subject 9 Misidentified as Subject 27 or Subject 3

In 8 one can see the tilted photo for subject 9 and two photos that were misidentified as subject 9. Our program misidentified subject 9 as subject 27, and in 7 one can see that the second closest error was subject 3.

3. Removing Singular Values

After obtaining all of our results, we decided to try and remove singular values. This is beneficial because we can still have a working facial recognition program, while massively decreasing the computation time. This can be particularly useful when the data size increases to very large numbers. For example, a police database likely contains millions of photos of people, and if they needed to identify a person using facial recognition, they would need to search through their entire database. If no singular values are removed, their matrix of singular values would be restrictively large. Without a supercomputer, the time required to do all the computation would exponentially increase, since matrix multiplication has a time complexity of $O(n^3)$. Therefore, to cut down on the time of matrix operations, it is possible to remove he least significant singular values. This will decrease the size of Σ from the SVD factorization and significantly decrease the computation time. To test the number of singular values we could eliminate we began removing singular values and testing if we could still identify everyone in the database. We found that we could remove 23 singular values, only keeping 4, before the facial recognition process failed.

VI. Discussion and Conclusions

In this report, we have presented an analysis of the technique and results from applying concepts in Linear Algebra to the problem of identifying faces in images. Specifically, we have explained the use of singular value decomposition (SVD) to aid in providing a solution to this problem. SVD allows one to compute a basis for the column space of a matrix. In this application, the matrix was made up of column vectors representing facial images. The range of this matrix is all faces that can be represented as a linear combination of this basis. Furthermore, SVD also provides information about the singular values of said matrix. This allows one to ignore some of the information in the basis in return for some error dependent on the maximum size of the singular values left out. Essentially, one is able to efficiently represent faces in a database using a smaller amount of information obtained from the original images. Using this mathematical information, we sought to study the execution of this in a real life scenario. We obtained a rather small, but usable, set of images with varying facial expressions, orientations, or accessories. Processing this data we were able use its SVD to accomplish facial recognition of the photos in our data set. We found that we could accurately identify photos, still somewhat similar to the training photos, as being a member or non-member of our database. By looking at the singular values and their magnitudes we were able to reduce the size of the data we retained for the facial recognition and still accurately identify images. Although, our sample size was already very small. The real application of this reduction comes with much larger data sets where a larger fraction of information can be removed. To further this work, it would be advisable to test on a much larger sample size of images. This would allow one to see exactly how little information is necessary for recognition. Additionally, it would be informative to test this process with images that are less uniform than ours. Having more variation in the images complicates the situation and may result in a breakdown of the process. Altogether, we have accomplished our goal of using SVD to recognize facial images. We discovered that a large amount of data can be removed while still preserving the process' integrity. Furthermore, we have found that accurate recognition, in our case, is dependent on well normalized photos – which is a major drawback of the system. These results, while capable of creating a facial recognition system, have more of an application in informing further research and development on the topic.

VII. References

[1] Georghiades, A.S. and Belhumeur, P.N. and Kriegman, D.J. "From Few to Many: Illumination Cone Models for Face Recognition under Variable Lighting and Pose", IEEE Trans. Pattern Anal. Mach. Intelligence, vol. 23, num. 6, pp. 643-660, 2001

[2] Singular Value Decomposition, Eigenfaces, and 3D Reconstructions, by Muller, Magaia, and Herbst, SIAM Review (2004).

[3] M. Turk and A. Pentland, "Eigenfaces for Recognition", Journal of Cognitive Neuroscience, vol. 3, no. 1, pp. 71-86, 1991, hard copy

[4] Meyer, Carl Dean. Matrix Analysis and Applied Linear Algebra. Nachdr. ed., Philadelphia, SIAM, Society for Industrial and Applied Mathematics, 2004.

VIII. Appendix

A. Code for `eigStuff`

```
%   Function - eigStuff
%   Parameters:
%       faceMatNorm - A m x n normalized matrix of faces
%                     represented as column vectors.
%   Returns:
%       eigVec      - The U component in SVD for faceMatnorm.
%       sigma       - A matrix of singular values for faceMatnorm.
%       faceDist    - A matrix of the projections of each face vector onto
%                     the k columns of U that are kept.
function [eigVec, sigma, faceDist] = eigStuff(faceMatNorm)
    % k number of columns being kept.
    global numEig;

    % Calculate SVD of a matrix of faces.
    % S is the matrix of singular values, and U
    % and V are orthonormal matrices.
    % V was never used in the program and is replaced by the ~ symbol.
    [U, S, ~] = svd(faceMatNorm);
```

```
    eigVec = U;
    sigma = S;

    rc = size(faceMatNorm); % Row vector for size of faceDist matrix.
    faceDist = zeros(numEig, rc(2));

    % Loop over each column of faceDist.
    for i = 1:rc(2)
        % Calculate eigenface representation for each face.
        faceDist(:,i) = U(:,1:numEig)'*faceMatNorm(:,i);
    end
end
```

B. Code for `projector`

```
function [y] = projector(U,face)
    % k number of columns being kept.
    global numEig;
    % Generate a vector to compare to the projections
    % of each of the training set images.
    y = U(:,1:numEig)'*face;
end
```