5-24-2019

Facial Image Characterization and Reconstruction Using Singular Value Decomposition

Jessica R. Robinson
Portland State University

1. Introduction

From the lens of a microscope to the aerial view of a satellite, images give us access to a plethora of visual information that human eyes could never process in its entirety. The use of algorithms and computation to process, manipulate, and even produce images has become exceedingly popular in the field of mathematics and computer science due to the wide range of practical applications to which it can be used. This can be seen in advancements of facial recognition, environmental monitoring, and medical imaging technology, to name a few.

Image processing through numerical techniques often requires a large amount of computational resources and memory storage. Due to this fact, one of the preliminary issues addressed in many image processing algorithms is that of low-dimensional characterization of the image collection in question. Once this is complete, the effort to carry out processes like principal component analysis or image reconstruction is done much easier and in more reasonable time.

My goal for this academic project is to investigate the implementation and effectiveness of an image characterization and reconstruction algorithm formulated by Lawrence Sirovich and Robert M. Kirby of Brown University (3) by applying it to a collection of facial images. In the process, I will demonstrate a theoretical framework of Singular Value Decomposition- a fundamental tool used in the algorithm.

2. Experimental Setup

2.1. **The Images.** The experimental database for this project consists of 750 facial images collected from "Labeled Faces in the Wild" (2), a database of facial images conglomerated at the University of Massachusetts, Amherst in the Computer Science department. The images are in grayscale and depict the faces of men and women in fairly consistent position, as shown in Figure 1. Each person featured in the database can be found in up to 14 photos. To facilitate the data analysis in this project, each image has been appropriately cropped and is numerically represented as a matrix of dimension 200×150. Each entry in a given matrix is representative of a pixel in the image and is indicative of that pixel's deviation from the mean grayscale of the entire image.

Figure 1. Sample photos from database

2.2. **Notation.** Let M denote the total number of images in the ensemble ($M = 750$). Additionally, let n and m represent the row and column dimension of a given image matrix, respectively. Lastly, for ease later, let $\mathcal{N}$ denote the total number of elements in a given image matrix. Note: $\mathcal{N} = 30,000$ for a given 200 x 150 image.

Let an image be denoted by $\varphi \in \mathbb{R}^{n \times m}$ where $\varphi = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & x_{2m} \\ x_{31} & x_{32} & \dots & x_{3m} \\ \vdots & \vdots & \ddots & \vdots \\ \dots & \dots & \dots & \dots \end{bmatrix}$

2.3. **The Average Image.** The first step to characterizing this ensemble of images is to acquire an average image. This average will be used for later computation in the algorithm. Denoted $\overline{\varphi}$, the average image for the ensemble is defined as

$$\overline{\varphi} = \frac{1}{M} \sum_{i=1}^{M} \varphi_i.$$

As Python is the language used in implementing the pieces of the algorithm, the construction of this average facial image would be computed by the following:

```
14  def get_mean_face():
15      mean_face = np.zeros((200, 150))
16      for i in range(0, M):
17          face = all_faces[i]
18          mean_face += face
19      return (1/M)*mean_face
```

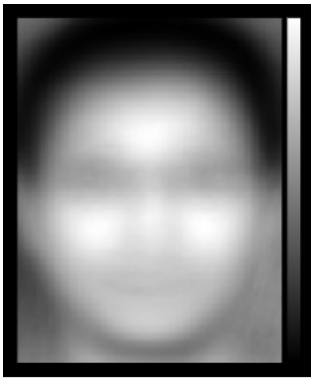


FIGURE 2. Average image displayed

Note: For the purposes of computation in later steps, from this point forward, image matrices will be concatenated into stacked $\mathcal{N}$ x 1 vectors via a column-major

flattening procedure.

2.4. **The Perturbation Matrix.** From this average image, the perturbation matrix can be constructed. This matrix will provide a value for each individual image's deviation from this database average. Denoted P, the perturbation matrix is

$$P = [\phi_1, \phi_2, \phi_3, \ldots, \phi_M] \; \epsilon \; \mathbb{R}^{\mathcal{N} \times M} \text{ where } \phi_i = \varphi_i - \overline{\varphi}$$

And is constructed in Python as follows:

```
for i in range (0, M):
    face = all_faces[i]
    face_v = np.matrix.flatten(face, order='F')
    P_Matrix[:,i] = face_v - mean_face_v
```

It is worth noting at this point that the dimension of the perturbation matrix is already significantly larger than that of the individual images due to the fact that a characterization of all 750 images is represented here.

2.5. **The Covariance Matrix.** The next necessary components for the representation and reconstruction procedure are the eigenvectors of the covariance matrix for the image ensemble. The covariance matrix can be found by taking the dyadic product of each element of the perturbation with respect to one another. This is also equivalent to multiplying the matrix by itself in transpose. Denoted C, the covariance matrix is created by

$$C = \left(\frac{1}{M-1}\right) PP^T = \left(\frac{1}{M-1}\right) \sum_{i=1}^{M} \phi_i \phi_i^T$$

A covariance matrix holds information about elements' relationship to all other elements within it as well as the elements' variance treated as a random variable on the diagonal. The operation carried out above is that of a matrix with $30,000 \times 750$ dimension multiplied by one of $750 \times 30,000$ dimension. Based on this, the covariance matrix in question would be of $30,000 \times 30,000$ dimension. It is worth noting that the storage of a matrix this large in double precision arithmetic would require seven gigabytes of memory.

Because the storage and resources for this project are not of the capacity necessary to compute, store, or manipulate this matrix, a computationally efficient method will be taken to obtain the eigenvectors needed for the next step in the reconstructive algorithm. This method relies heavily on the mathematical theory of singular value decomposition.

3. Singular Value Decomposition

3.1. **The Mathematical Framework.** Singular Value Decomposition, or SVD, is the representation of any $n \times m$ matrix by three distinct matrices representing a three step transformation. It is a method used frequently in situations where data to be manipulated is large and complex yet only certain components or behavior need to be known to carry out the process in question.

To demonstrate the transformation that SVD represents when used on a given $n \times m$ matrix, consider the following geometrical interpretation borrowed from a set of lectures by L. Trefethen and D. Bau (4).

Let A be a 2×2 matrix with arbitrary elements. Let S be a unit circle in $\mathbb{R}^2$. The transformation of A to the unit circle S results in a hyper-ellipse, which will be denoted AS.

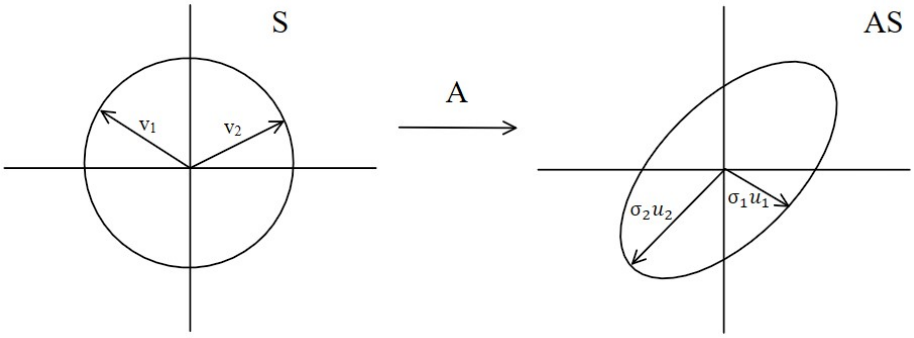


FIGURE 3. Unit circle S transformed by 2 x 2 matrix A

With this transformation in mind, the framework for singular value decomposition is very nicely demonstrated.

Consider the scenario where the unit circle S were expressed via a set of two vectors (matching the number of columns in A). Note: these vectors are denoted v_1 and v_2 in Figure 3 above. Now, consider the computations denoted $\{Av_i : i \in [1, m]\}$ where m is the number of columns of A (in this case, $m = 2$). The vectors which result from this will be denoted $\{\sigma_i u_i : i \in [1, m]\}$. Therefore, for a given vector v_i,

$$Av_i = \sigma_i u_i$$

By this, it can be seen that the right hand term in this equivalence represents the vectors which express the hyper-ellipse as a result of the transformation A. Now, consider the following:

$$Av_i v_i^T = \sigma_i u_i v_i^T \implies A = \sum_{i=1}^{m} \sigma_i u_i v_i^T$$

At this point, the singular value decomposition of the matrix A has been defined. Let U, Σ, and V denote the matrices containing the u_i, σ_i, and v_i vectors, respectively (where $i = 1, 2, ..., n$).

$$U = \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1m} \\ u_{21} & u_{22} & \dots & u_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \dots & \dots & \dots & \dots \end{bmatrix} \in \mathbb{R}^{nxm}, \quad V = \begin{bmatrix} v_{11} & v_{12} & \dots & v_{1m} \\ v_{21} & v_{22} & \dots & v_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \dots & \dots & \dots & \dots \end{bmatrix} \in \mathbb{R}^{mxm},$$

$$\Sigma = \begin{bmatrix} \sigma_{11} & 0 & 0 & \dots \\ 0 & \sigma_{22} & 0 & \dots \\ 0 & 0 & \sigma_{33} & \vdots \\ \dots & \dots & \dots & \ddots \end{bmatrix} \in \mathbb{R}^{mxm}$$

To reiterate the above demonstration, performing a singular value decomposition on a matrix A is the representation of this matrix via three multiplied matrices (U, Σ, and V in transpose).

$$A = U\Sigma V^T$$

U and V are orthonormal matrices while Σ contains the singular values in an ordered fashion such that

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_m \geq 0$$

Note that the term $\sigma_i u_i v_i^T$ is switched around a bit to $U\Sigma V^T$ in matrix form. This is due to a property regarding the multiplication of a diagonal matrix (in this case, Σ).

Property: Let A and D be matrices where A is of n x m dimension, and D is a square diagonal matrix of m x m dimension with zeros in the non-diagonal entries.

$$A = [(:,1),(:,2),\dots,(:,m)], \quad D = \begin{bmatrix} d_{11} & 0 & 0 & \dots \\ 0 & d_{22} & 0 & \dots \\ 0 & 0 & d_{33} & \vdots \\ \dots & \dots & \dots & \ddots \end{bmatrix}$$

Then the product of AD, by result of matrix multiplication properties, gives the following:

$$AD = [d_1 A(:,1), d_2 A(:,2), \dots, d_m A(:,m)]$$

Each matrix, U, Σ, and V, and their respective vectors have significance in relation to the geometric transformation outlined previously. They also each hold important properties and details about the matrix A which they represent.

The V matrix is the collection of unit vectors which create the axes of the unit circle transformed by A- classified as the right singular vectors of A.

The U matrix is the collection of directional unit vectors which represent the orientation of the hyper-ellipse as a result of the A transformation- classified as the left singular vectors of A.

The Σ matrix is a diagonal matrix holding the values which attach to the left singular vectors as magnitudes and represent the stretching of the hyper-ellipse. Following this logic, a given unit vector v_i, under A, will be transformed to be $\sigma_i u_i$, a new vector with stretched magnitude and direction.

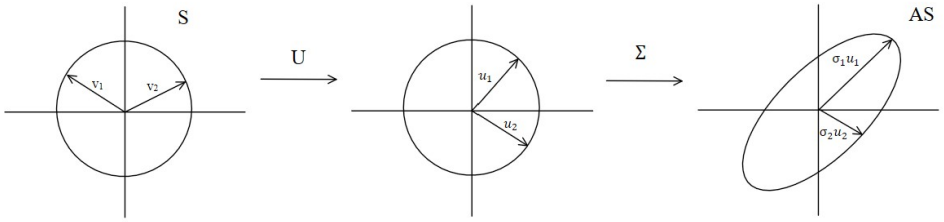


FIGURE 4. Unit circle S transformed by 2 x 2 matrix A

An interesting note about some of the functionality of SVD, the values that the singular value decomposition of a given matrix are made up of are actually providing useful information about patterns and structures that were found in the original matrix. For example, in the context of this project, consider the result when the left singular vectors are reshaped into the correct sized matrices (200×150) and displayed as images themselves (Figure 5).

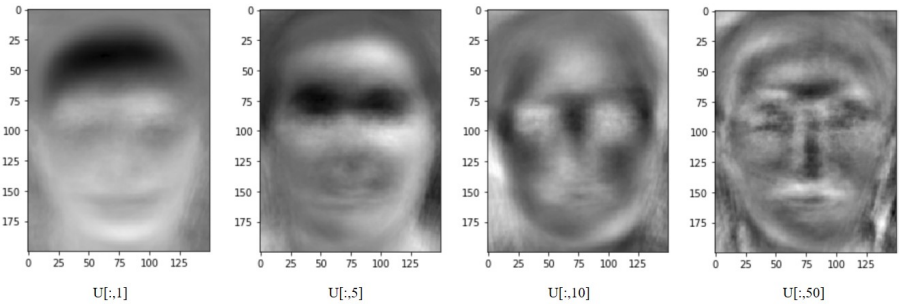


FIGURE 5. Left singular vectors displayed as images

What can be seen here is that the left singular vectors, as they provide an orthonormal basis for the matrix A (to which SVD was performed), are providing some of the principal components of that matrix. SVD is useful in this way as it relies on the fact that the data being manipulated is not truly random; there are natural patterns and common structures lying within the data.

In this case, see that the left singular vectors are preserving the overall structure of the faces in the database as principal components. It is also worth noting that primary singular vectors (i.e. u_1, u_2, u_3, etc.) will preserve this structure much better than that of the latter vectors (u_{m-1}, u_m) due to the accumulation of skewing features from the database as it transcends through the images. These primary vectors provide very good low-dimensional representations of the matrix which has been decomposed by this process.

3.2. **Full vs. Reduced or Economy SVD.** In the framework drawn up so far, the dimensions of the U, Σ, and V matrices are as follows: $U \in \mathbb{R}^{nxm}$, $\Sigma \in \mathbb{R}^{mxm}$, $V \in \mathbb{R}^{mxm}$. This is considered a thin, economy, or reduced singular value decomposition due to the fact that in many cases with an $n \times m$ matrix, U is not

a square matrix. In the case of a full singular value decomposition of A, $n-m$ orthogonal columns (so as to hold necessary properties) would be appended to the U matrix. Additionally, to hold the multiplication stable, $n-m$ additional rows would be appended to the Σ matrix. In most scenarios, these rows of Σ consist of all zeros so as to induce cancellation when multiplied with the corresponding columns of U.

In this case, a full singular value decomposition would not provide any additional useful information to the efforts of the project and would cause a much slower and lengthier computation. Due to this, a reduced or thin singular value decomposition will be used.

3.3. **Useful Properties.** It is a proven property that given any $n \times m$ matrix, there exists a singular value decomposition for it. So, for any A, there exists a $U\Sigma V^T$ set of multiplied matrices to represent it. Due to this fact, SVD and its related properties have many practical applications. The most applicable theorem in the context of this project in image processing is stated as follows:

Theorem 3.1. *Let A be an $n \times m$ matrix. Then the left singular vectors of A, within its singular value decomposition, are the eigenvectors of AA^T.*

Proof: Consider the calculation

$$AA^T = (U\Sigma V^T)(U\Sigma V^T)^T = (U\Sigma)(V^TV)(\Sigma^TU^T) = U(\Sigma\Sigma^T)U^T$$

Note that V is orthonormal implying that $V^{-1} = V^T$. Hence $V^TV = I$. Because U is orthonormal, this holds for UU^T as well. Multiplying by U on the right hand side gives:

$$AA^TU = U\Sigma\Sigma^T$$

Considering AA^T multiplied by a single vector of U gives:

$$AA^Tu_i = \sigma_i^2 u_i$$

Note that the diagonal entries of Σ multiplied in this way are what result in the σ_i^2 term.

The above line, which depicts a matrix multiplied by a vector equivalent to a scalar multiple of a vector, emulates the definition of an eigenvector of AA^T. Therefore, the left singular vectors, or columns of U, are the eigenvectors of AA^T $\square$.

Another important theorem related to the singular value decomposition of a matrix that is useful for this project is in regards to low-rank approximations of a given matrix (or image, in this case.) But first, a definition is necessary.

Definition: Let the Frobenius norm of an n x m matrix be defined by

$$||A||_F = \left(\sum_{i=1}^{n}\sum_{j=1}^{m}|a_{ij}|^2\right)^{1/2}$$

Now, this definition can be used in the following proposition regarding approximations of a matrix A.

Theorem 3.2. *Define $A_v = \sum_{j=1}^{v}\sigma_j u_j v_j^T$. Then*

$$||A - A_v||_F = \sqrt{\sigma_{v+1}^2 + \cdots + \sigma_m^2}$$

This theorem shows that a given matrix, when represented by a sum of v rank one singular value decompositions, has an error determined by the remaining $m-v$ singular values.

4. Implementation and Results

4.1. **The Algorithm.** By performing singular value decomposition on the matrix P (recall that PP^T gives the covariance matrix for the image database), and utilizing Theorem 3.1 from above, the last necessary values for the algorithm can be found- the eigenvectors of, what would be, the covariance matrix.

Let a reconstructed/characterized image be denoted $\hat{\varphi}$ and built in the following way:

$$\hat{\varphi} = \overline{\varphi} + \sum_{i=1}^{M} a_i u_i$$

where the coefficients a_i are evaluated as

$$a_i = (u_i, \varphi_i - \overline{\varphi})$$

Note: a_n is the Euclidian product of the corresponding two vectors.

In order to test this algorithm in its ability to reconstruct a photo, a test image will be removed from the database ($M = 749$ now) and a new average image, perturbation matrix, and singular value decomposition will be found. By means of these elements, the algorithm will build a reconstructed image. Some results are shown in Figures 6 and 7 below:

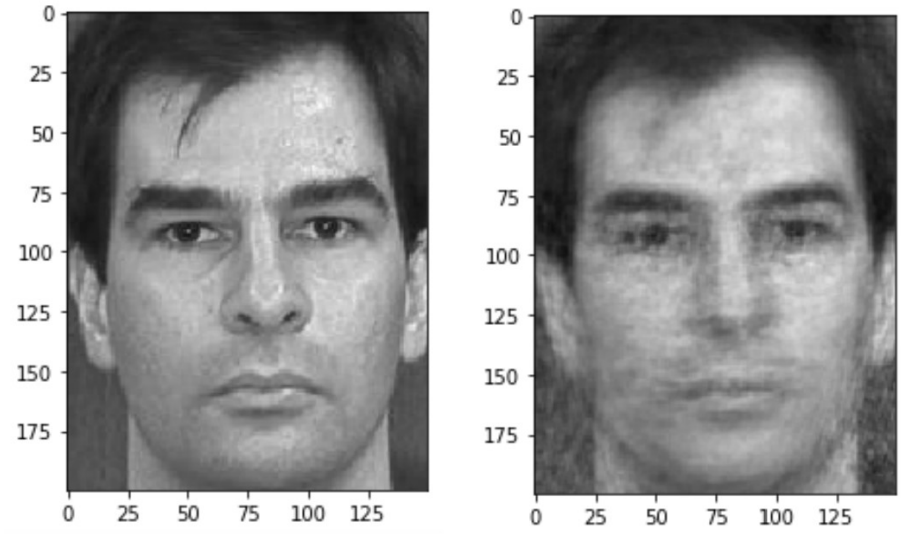


FIGURE 6. Left: Original image. Right: Reconstruction based on the remaining images in the database.

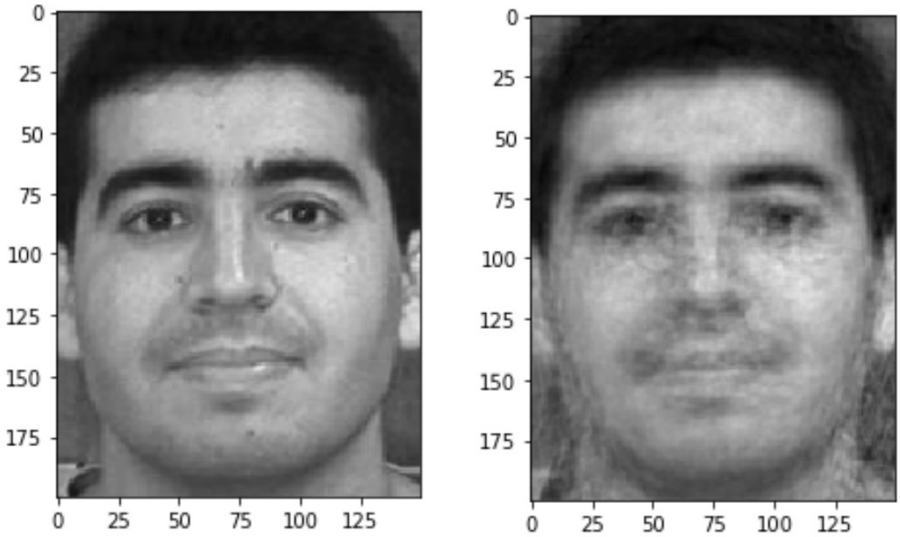


FIGURE 7. Left: Original image. Right: Reconstruction based on the remaining images in the database.

4.2. **SVD Applied.** In creating these reconstructions, the algorithm is using the entire 749 image database (the given image removed). This is a relatively small and easily computable process given the resources at hand. Though, consider a

scenario where the database used to reconstruct an image is of thousands, possibly millions of images. How many processed images are necessary to produce a decent reconstruction of an image? This evaluation of error is something that can be shown visually via a guess-and-check procedure given the figures above, but it can also be represented numerically. Because our algorithm is representing an image based on a linear combination of singular left singular values, this can be done by means of Theorem 3.2 stated in the previous section. By that, the following holds:

$$||\varphi - \hat{\varphi}_i||_F = \sqrt{\sigma_{i+1}^2 + \cdots + \sigma_m^2}$$

By this, if a given image is reconstructed by means of i database images, the error of this image in comparison to the original (due to the use of singular value decomposition) is given by the square rooted sum of the singular values of the images in the range from $i+1$ to m.

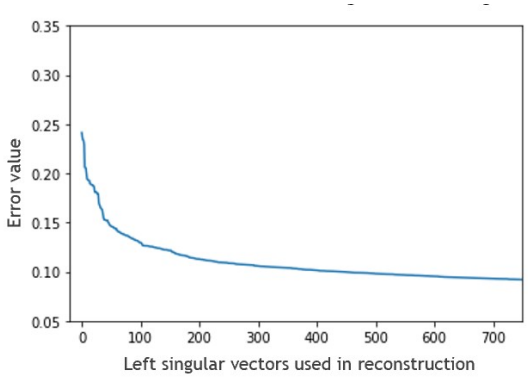


FIGURE 8. Error values for reconstruction depicted in Figure 5

The figure above features the behavior of the relative error of a reconstructed image as more images are used in its creation. When very little images are used, the error is extremely high and results in a poor image. As more images are used, the error gets closer to zero. This is because the algorithm has more information about the faces in the database to use in its construction of the new image.

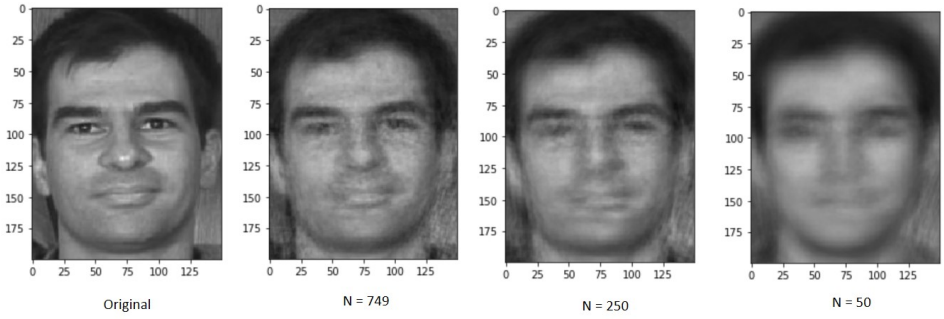


FIGURE 9. As less images are used, reconstruction is worse in quality

Seen above, as the algorithm uses less information in the database to create the reconstructed photo, the product is of a lesser quality. It is also noticeable that the reconstruction resembles the average image accrued earlier on as fewer images are used.

5. Future Work

Based on the work done in this project, it is evident that there are a plethora of routes to take with this image processing algorithm.

An interesting scenario to test and implement would be to, instead of feeding the algorithm a whole and unaltered photo (which are hard to come by in practice), feed it an image which is damaged or masked in some way and see how well the reconstruction fares. This would be a useful tool in many applications of image processing (medical imaging, environmental monitoring, facial recognition, etc.).

For example, a masked face in the case of the database for this project would look something like the photo in Figure 10.

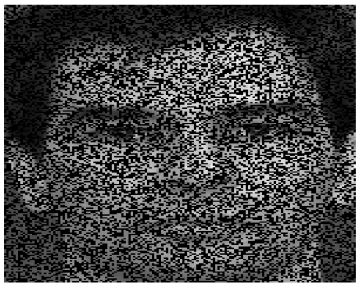


Figure 10. Example of a masked face

In addition, the database used in implementation was relatively small in the field of image processing. It would be interesting to see how this algorithm fares with a much larger and varying database. The increase in faces could increase the algorithm's ability to reconstruct smaller details in faces but would also require more storage and computing power to implement.

6. References

1. Everson, R., and L. Sirovich. "Karhunen–Loève Procedure for Gappy Data." Journal of the Optical Society of America A, 12.8, (Aug. 1995), pgs. 1657–1664.

Everson and Sirovich, in this article, discuss a least-squares procedure for restoring a faulty image with missing data. A main problem addressed in the article is determining the minimal amount of image data necessary to accomplish a reasonable or complete restoration. Based on its diction and location, article lies in the niche of numerical optimization within the field of mathematics and is directed towards an audience who is at least at the undergraduate level of mathematics if not at a graduate or doctorate level. In the context of this project, this source has aspects which I would like to aim for in the furthering of my research in image restoration. The techniques used here will be helpful to reference as I take on more difficult reconstructions using the algorithm I have developed.

2. Huang, Gary B, et al. "Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments." University of Massachusetts, Amherst, Oct. 2007, people.cs.umass.edu/ elm/papers/lfw.pdf.

This source is a collection of over 13,000 images for the furthering of research in facial image optimization. It is designated by the contributors that the photos have been harvested from various parts of the internet and the name of the individuals are included with the photos. It is safe to assume that the intension for this source is deliberate as only those with familiarity with optimization of image data are going to be able to use this database for its intended use. As a result, the audience for the attached articles are going to be those in the mathematical optimization field. This source is where my test images have been taken from. Because I need images of similar size, dimension, color, and scale in order for my algorithm to work properly, this database is the optimal source to pull from.

3. Sirovich, L., and M. Kirby. "Low-Dimensional Procedure for the Characterization of Human Faces." Journal of the Optical Society of America A, 4.3, (Mar. 1987), pgs. 519–524.

Sirovich and Kirby, in this article, present a method for the representation of facial images in a low-dimensional space. They use principal component analysis that allows for the characterization of a large ensemble of images with a relatively low amount of computation power and time. I would say that this article falls in the category of an undergraduate or graduate level understanding of numerical optimization and linear algebra based on the diction and notation. This source is the one I have referenced the most in the beginning of my project research. It has functioned very well as a base for furthering my algorithm to restore more difficult images.

4. Trefethen, Lloyd N., and David Bau. Numerical Linear Algebra. Society for Industrial and Applied Mathematics, 1997.

This source is a set of lectures/textbook chapters covering a foundation of linear algebra as well as a theoretical framework of singular value decomposition. This source is aimed towards students, most likely those of a graduate level or those in upper-level linear algebra courses.

Honors Project Implementation

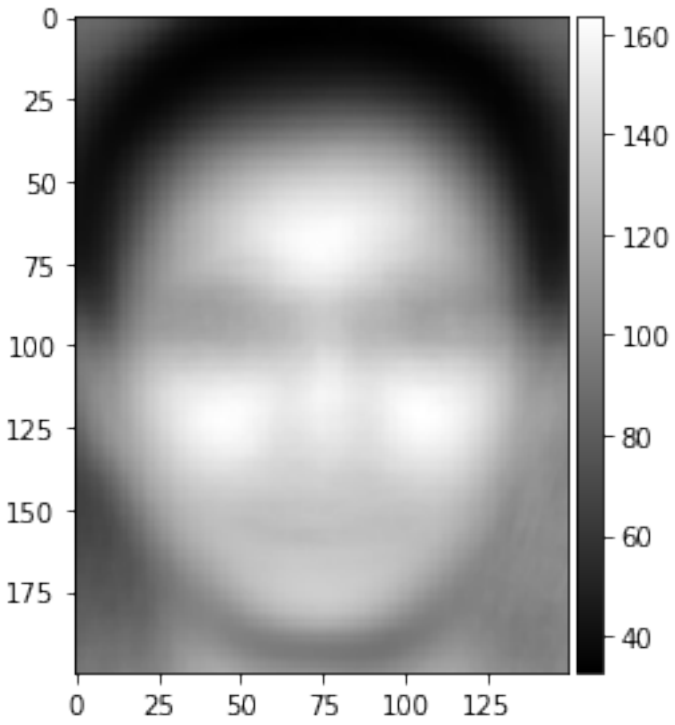
June 13, 2019

In [1]:
```
import numpy as np
import skimage
from skimage.io import imread_collection
import matplotlib
import matplotlib.pyplot as plt
from skimage import img_as_ubyte, img_as_float

# number of images in database
M = 749
# read in database
all_faces = imread_collection("DATABASE/*.jpg", conserve_memory=True)
# initialize face matrix
face = np.zeros((200, 150))
error_vals = np.zeros(M)

# loop through collection of faces, add to mean face
def get_mean_face():
    mean_face = np.zeros((200, 150))
    for i in range(0, M):
        face = all_faces[i]
        mean_face += face
    return (1/M)*mean_face
```

In [2]:
```
skimage.io.imshow(get_mean_face(), cmap='gray')
```

```
C:\Users\jessi\Anaconda3\lib\site-packages\skimage\io\_plugins\matplotlib_plugin.py:80: UserWa
  warn("Float image out of standard range; displaying "
```

Out[2]:
```
<matplotlib.image.AxesImage at 0x1b74c1adef0>
```



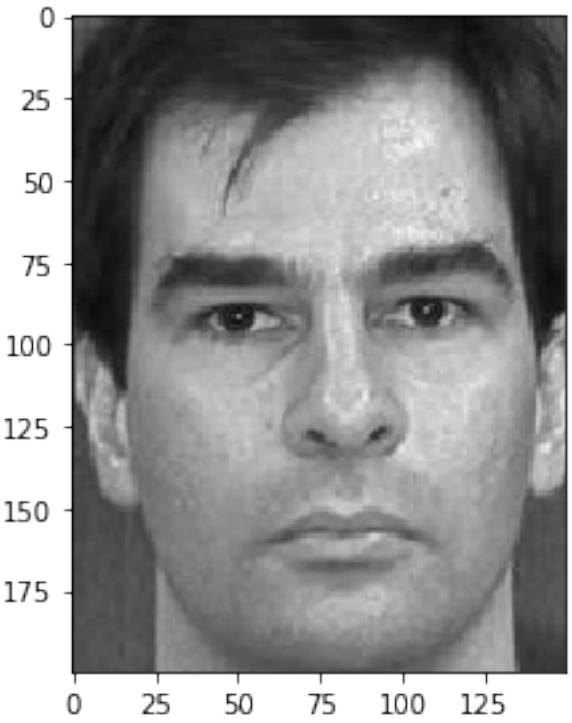
In [3]:
```
# Perturbation Matrix

# concatenate average face
mean_face_v = np.matrix.flatten(get_mean_face(), order='F')
P_Matrix = np.zeros((30000, M))

for i in range (0, M):
    face = all_faces[i]
    face_v = np.matrix.flatten(face, order='F')
    P_Matrix[:,i] = face_v - mean_face_v

# thin SVD on perturbation marix
u, s, v = np.linalg.svd(P_Matrix, full_matrices=False)
```

In [4]:
```
def reconstruct(test_face):
    '''
    Function which builds a reconstructed image matrix by means of the algorithm using
    the columns of u and the average face
    '''
    #concatenation
    test_face_v = np.matrix.flatten(test_face, order='F')

    #initialize reconstructed image as average image to be added to
    reconstructed = mean_face_v
```

```
            for i in range(0, M):
                reconstructed += ((np.inner(u[:,i], (test_face_v - mean_face_v)))*u[:,i])
                error_vals[i] = np.linalg.norm(test_face_v - reconstructed)/np.linalg.norm(tes

            # scaling values for displaying purposes
            reconstructed = np.reshape(reconstructed, (200, 150), order='F')
            reconstructed = np.round(reconstructed, decimals=0)
            reconstructed = reconstructed.astype(int)
            reconstructed = img_as_ubyte(reconstructed)

            print("Displaying reconstructed image: ")
            skimage.io.imshow(reconstructed, cmap='gray')

            return

        print("Displaying true image: ")
        skimage.io.imshow(skimage.io.imread("s01_01.jpg"))
```

```
Displaying true image:
```

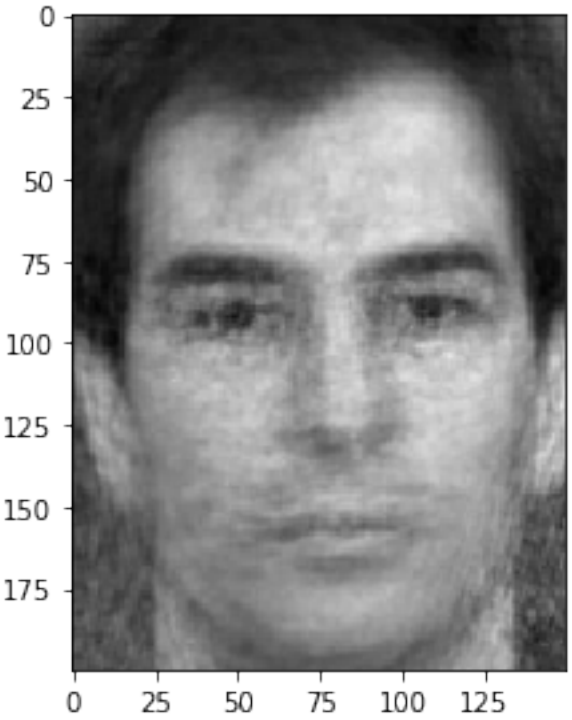
Out[4]: <matplotlib.image.AxesImage at 0x1b74c2ba198>



```
In [5]: reconstruct(skimage.io.imread("s01_01.jpg"))
```

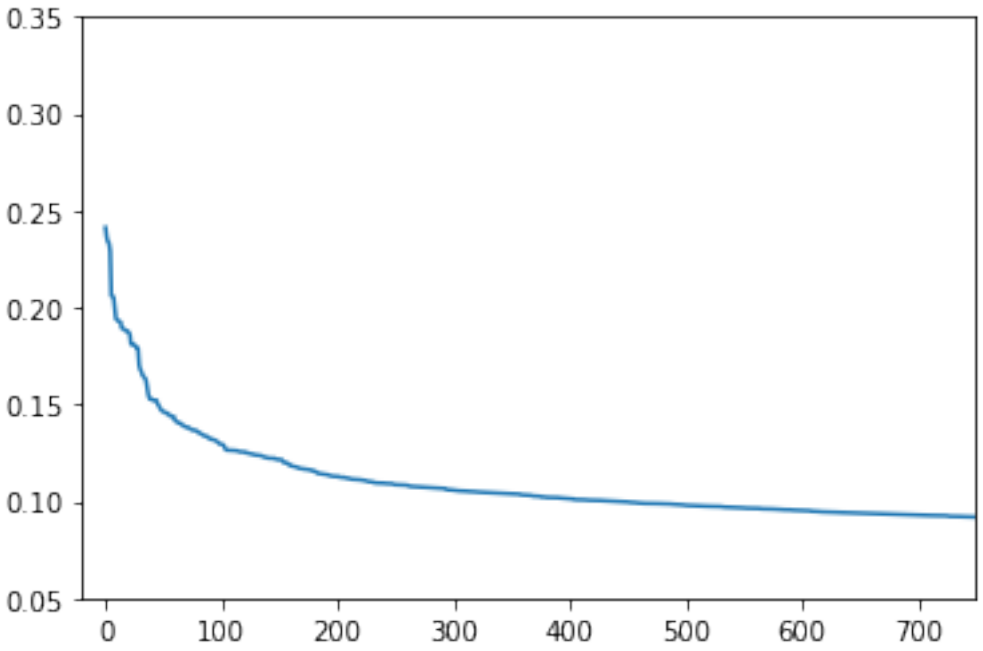
```
C:\Users\jessi\Anaconda3\lib\site-packages\skimage\util\dtype.py:126: UserWarning: Possible sig
  .format(dtypeobj_in, dtypeobj_out))
C:\Users\jessi\Anaconda3\lib\site-packages\skimage\util\dtype.py:179: UserWarning: Downcasting
  "value {} fits in {}".format(a.dtype, dtype, a.max(), dtype))
```

```
Displaying reconstructed image:
```




```
In [6]: # Displaying error values
        plt.axis([-20, 750, .05, .350])
        plt.plot(error_vals, "-")
```

```
Out[6]: [<matplotlib.lines.Line2D at 0x1b761e88b70>]
```



In [ ]:

In [ ]: