Eigenface Recognition With Non-Frontal Looking Faces

Austin Stone

May 18, 2014

Project objectives:

The goals of this project are (1) given a new, unknown image, determine if that image is a face, (2) if that image is a face, determine which direction it is looking, and (3) if that image is a face that has been seen before (perhaps in a different orientation), report that the face has been seen before. The final goal (4) is to do the previously listed goals extremely fast.

To achieve this, I built off an idea for facial recognition known as the "Eigenface Method" [1].This method is particularly notable for its speed of processing and for its dramatic application of the mathematical method of principal components analysis [2]. One of its caveats is that it is known to be somewhat lacking in ability to capture faces at odd orientations. I am hoping to build on the method in a way that will allow it to successfully recognize faces at different angles.

High Level Overview of Eigenface Method:

Naturally, images can be thought of as matrices where each element of the image matrix codes for the pixel color value. To avoid some unnecessary processing complications, all images used for my algorithm were converted to greyscale where each pixel is represented by a value between 0 to 255 where 0 is as light as possible, and 255 is as dark as possible. In this format, it is easy to apply linear algebra techniques to compute a wide range of important statistics about the pictures. For the purposes of this project, I am concerned with image similarity.

Computing similarity naively between image matrices can be very expensive. For example, if you have a database of 50 images against which to compare a new image, and each image matrix is 1000 by 1000, this involves $1000 * 1000 * 50 = 50000000$ arithmetic operations to compute the euclidian distance between two image matrices. This type of operation can be untenable for practical purposes, especially when images are large and/or there are many images in the test set and storing all of them uncompressed becomes very space intensive. The Eigenface method can speed up this distance calculation by several orders of magnitude and avoid the requirement of storing the entire database of training images in memory.

Imagine again that our training set has image matrices that are 1000 by 1000, and there are 100 of them. Each image matrix can be represented as a $1000 * 1000 = 1000000$ by 1-dimensional vector (created by appending all the rows of the image matrix together). You can imagine plotting this vector as a point in a 1000000-dimensional space. Taking every image in the database, converting it to a vector, and plotting it this way, the principal components are the axes in which most of the variance of the points lie (i.e., the direction in which most of the spread of the points is). We can take the top 20 of these axes of greatest variance (which are all orthogonal to each other, with the first axis explaining most of the variance and the second axis explaining most of the variance not explained by the first and so

on). Since these principal components are axes (lines) in a 1000000-dimensional space, each principal component is a 1000000-dimensional vector and can be viewed as an image by collapsing it down to a 1000 by 1000-dimensional matrix. This "matricized" version of the principal components is what is known as an Eigenface. Each image in the database can then be projected onto the first twenty of these Eigenfaces, and instead of storing each of the 100 full 1000 by 1000 image matrices in the database, we can instead just store the first twenty eigenvectors and the weights of each image's projection onto the first twenty Eigenfaces, which will be a 1 by 20-dimensional vector. Images in the database can be reconstructed fairly well by linear combination of the Eigenfaces weighted by the projection weightings as is shown in Figure 8. Furthermore, comparisons can now be achieved by simply comparing the projection of the new, unknown image onto the Eigenfaces to the projections of the test images onto the Eigenfaces. Without any further optimizations, this now only requires $25 * 25 * 100 = 62500$ operations, making each comparison 1600 times faster than the naive euclidian distance calculation.

Overview of Methods

My approach is similar to the method outlined in the previous section except for a few additions. The database I used had images looking in 5 angle increments from - 90 degrees (left profile) to + 90 degrees (right profile). A few sample images from the database are shown in Figure 1.

Example Images from Database

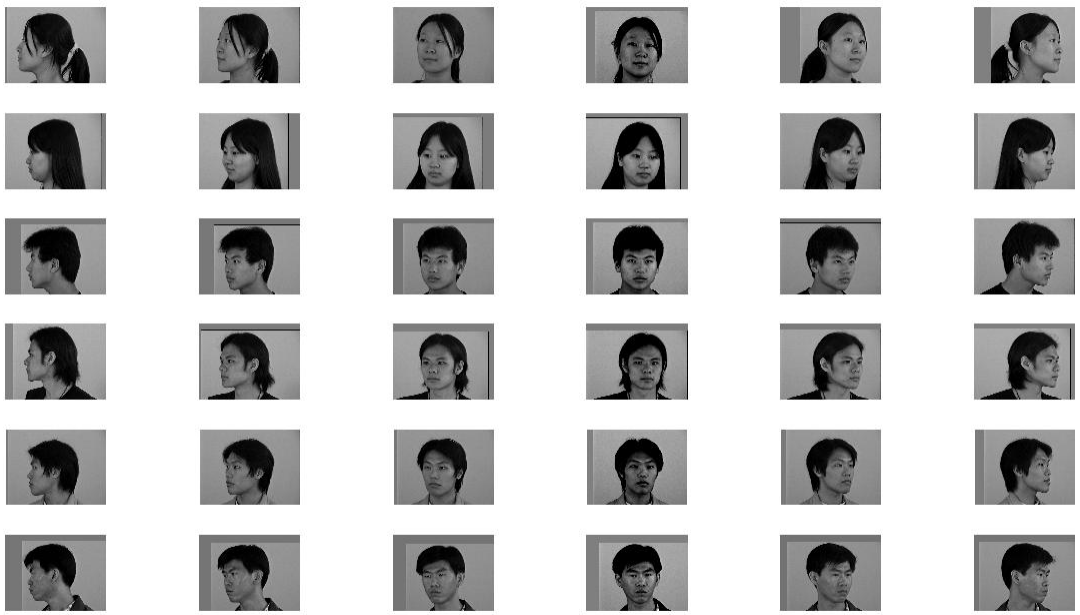

Figure 1: These are sample images from my database. My database contains roughly 7400 images of 200 different people photographed looking at each 5 angle increment from left (-90) to right (90). This picture does not show all angle increments.

I created a database structure that had fields for each direction increment. In each field I stored a matrix where each column of the matrix is a unique image in vector format. That is:

$$face\_mat = [\tau_1, \tau_2, \tau_3, ...\tau_n]$$

where τ_i is image number i in vector format. From this, I was able to obtain the following mean image by averaging all the columns of the data matrix together. This "raw mean" image is displayed in Figure 2.

Mean Composite before Centering Algorithm

Figure 2: This is the mean image of the database of uncentered facial images that was used to center all other images. You can see that the image is very blurry and wide, representing the lack of centering in the raw images.

I used this mean image to center all other images by exploring shifts of the normalized images up and down and finding the shifted location that maximized the cosine similarity with the mean image. The cosine similarity was calculated by converting each image to a vector representation, and then calculating the dot product of these two images. Figures 3 and 4 show an example image before and after the shifting algorithm, and Figure 5 shows the new mean image taken from all the centered images.

Image Before Shifting with Centering Algorithm

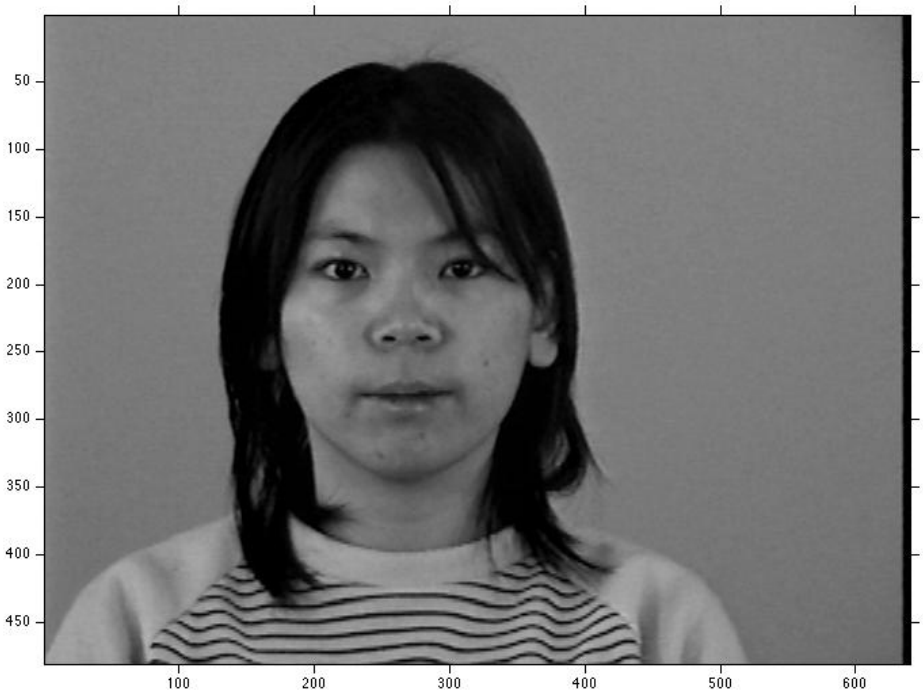

Figure 3: This is the image before running my centering algorithm. As you can see, it is off center to the left and up.

Image After Shifting with Centering Algorithm

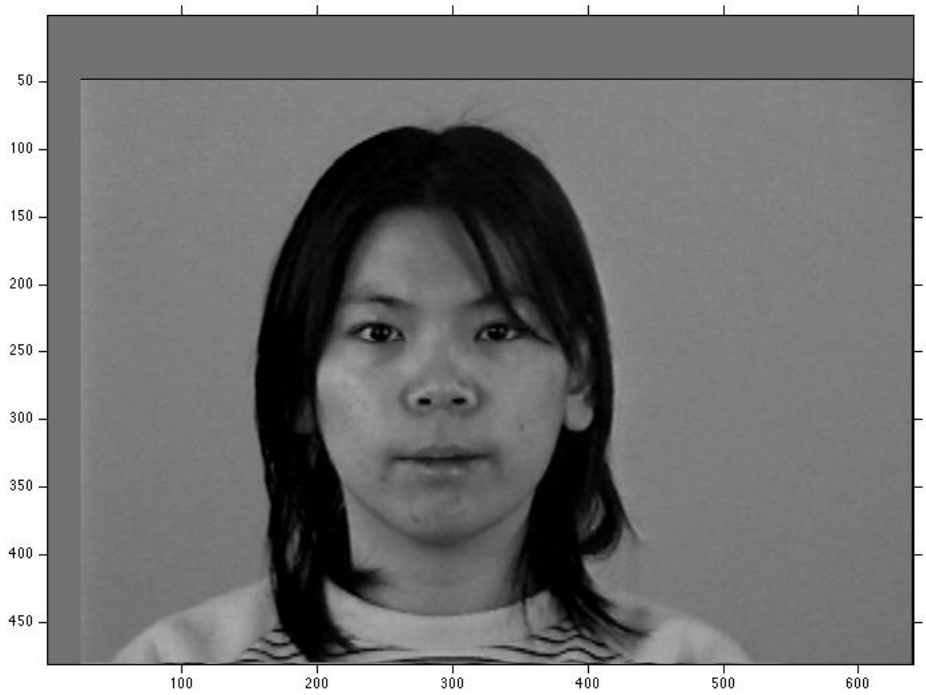

Figure 4: This is the image after running my centering algorithm. As you can see, the face is now middle aligned by shifting to the left and down.

Mean Composite after Centering Algorithm

Figure 5: This is the mean image of the database of facial images that have been passed through my centering algorithm. As you can see, this mean is much clearer and more resolute than the initial mean of the uncentered images in Figure 2.

Centering the images was important so that the eigenfaces will capture the different features of the faces instead of the variance due to the fact that the images are off center. Centering new images will also help with recognition of the images.

After centering the images, the next step was to generate the eigenfaces (principal components) for each angle class of face (i.e., -90 degree looking images, -85 degree looking images, ... 90 degree looking images). One of the most computationally efficient methods of computing principal components is to use Matlab's built-in singular value decomposition. Given a data matrix, X, the principal components of the data matrix are the eigenvectors of the covariance matrix, $X^T X$. Since singular value decomposition decomposes X into USV^T, where U and V are orthonormal matrices and S is a diagonal matrix, the following is true:

$$\begin{gathered} X = USV^T \\ XVS^TU^T = USV^TVS^TU^T \\ XX^T = US^2U^T \\ XX^TU = US^2 \end{gathered}$$

Thus, I do singular value decomposition of the mean centered face image matrix $face\_mat$. This produces a new matrix V of the same size as $face\_mat$ where the columns are the principal components.

Collapsing these principal component column vectors back down into 640 by 480 image matrices, I can view them as images. These images are what is known as eigenfaces. Figure 6 displays an image matrix of the first 20 eigenfaces for forward looking faces.

First 20 "Eigenfaces" for Forward Facing Images

Figure 6: These can be thought of as capturing different fundamental features present in the database of faces. These most significant features are measured for each face, and each face is then assigned a "score" in terms of how much of these features it contains.

The final step before validation is to use a series of sample images from each class (non-facial images, facial images that have not been seen before, and facial images that have been seen before but under different orientations) to compute a slew of statistics that can be used to classify future unknown images. Unfortunately, when I first downloaded my database, I had the understanding that it provided multiple different images of the same subject for all orientations. However, I later discovered that these duplicate images were only flipped left to right versions of other images in the database which behave differently than would a new image of a subject. Since I don't currently have a database with duplicate images, I am unable to compute statistics for this class of image, although I do have code to compute the statistics for this class. The remainder of my analysis only concerns distinguishing images that are not faces from images that are faces.

To compute the statistics needed for classification, for all fields in the database (i.e., all angle increments from -90 to 90), I projected all images in these fields images onto the first 30 eigenfaces for that field. These projection weightings and the first 30 eigenfaces are all that needs to be stored for the proceeding steps.

The projection of face number i in field number f is denoted as $eig_{i,f}$. For all values of$i$ and f, $eig_{i,f}$ is a vector of length 30 by 1, with element g corresponding to the length of the projection of image i onto eigenface g. Each image in the database can be reconstructed from its vector $eig_{i,f}$ and the first 30 eigenfaces. Below is an example of this.

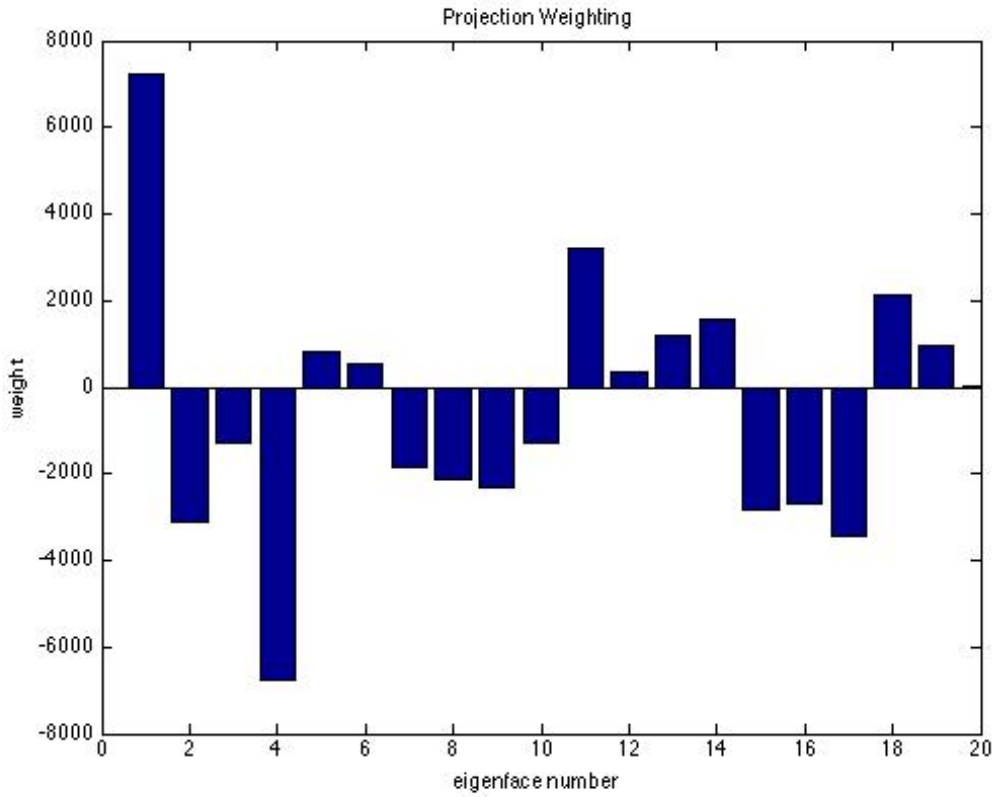

Figure 7: This is a bar graph shows the vector $eig_{i,0}$ for some image i in the 0 degree (front facing) field. You can think of the value of each bar as representing how much that eigenface contributes to the image.

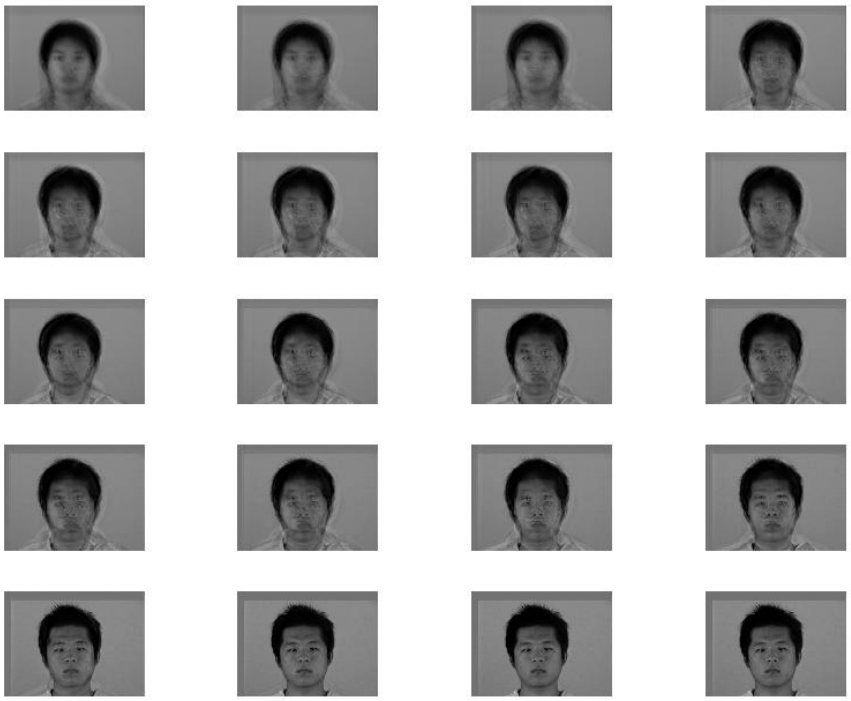

Figure 8: This image tiling shows the reconstruction process of the image whose projection vector was displayed in the previous figure. The upper left corner is the first eigenface for this image times its projection weighting, and moving left to right (and down to the proceeding row when the end of a row is reached) is "overlaying" (linearly combining) the next eigenface for this image weighted by that eigenface's projection weighting.

To compute my classification statistics, I used 50 new images from the database at each orientation (50 times 37 images = 1850 images total), and I used 50 new images not of faces which I randomly selected from Google images. Given a new image that is of a face looking a known orientation f, I projected this image onto the principal components previously calculated and stored for that orientation f and computed its distance from all $eig_{i,f}$ for that f in the database. Thus I have 50 trials for each orientation for new faces. I also ran 50 trials at each orientation by projecting my 50 non-facial images onto the principal components for that orientation, and computed the distance from each $eig_{i,f}$ for all images i in that orientation.[1]

From this procedure, the statistics that I saved for use in classification are as follows:

- Average distance
- Standard deviation of distance

[1]For the figures above, I used eigenfaces computed from non-normalized images, and I did not normalize images before projecting them onto the eigenvectors. To make classification performance better, I normalized all images before doing PCA, and I normalized all test images before projecting them onto the principal components. This helps account for variance in projection weightings of very dark or very light (i.e., low valued) images.

- Minimum distance
- Standard deviation of minimum distance
- Maximum distance
- Standard deviation of maximum distance

The bar graphs in Figures 9 and 10 display some of these statistics and indicate a clear difference between non-facial images and facial images.

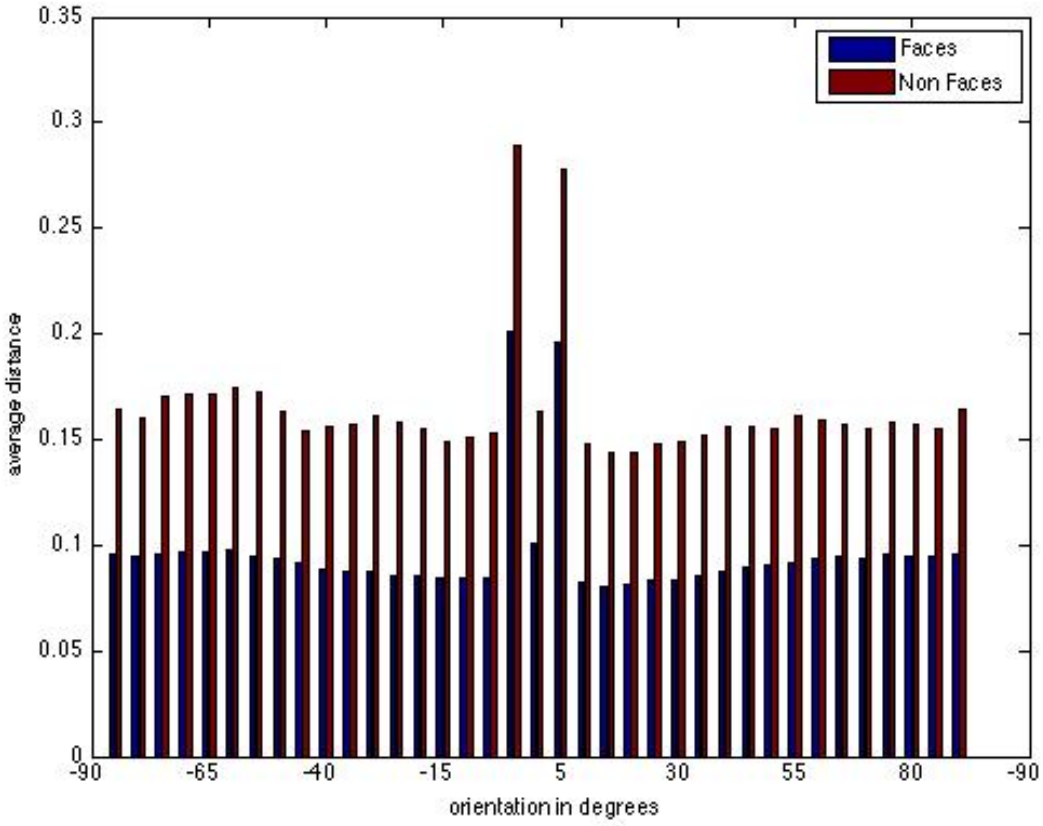

Figure 9: New, unseen facial images tend to be almost twice as close to images in the database as non-facial images.

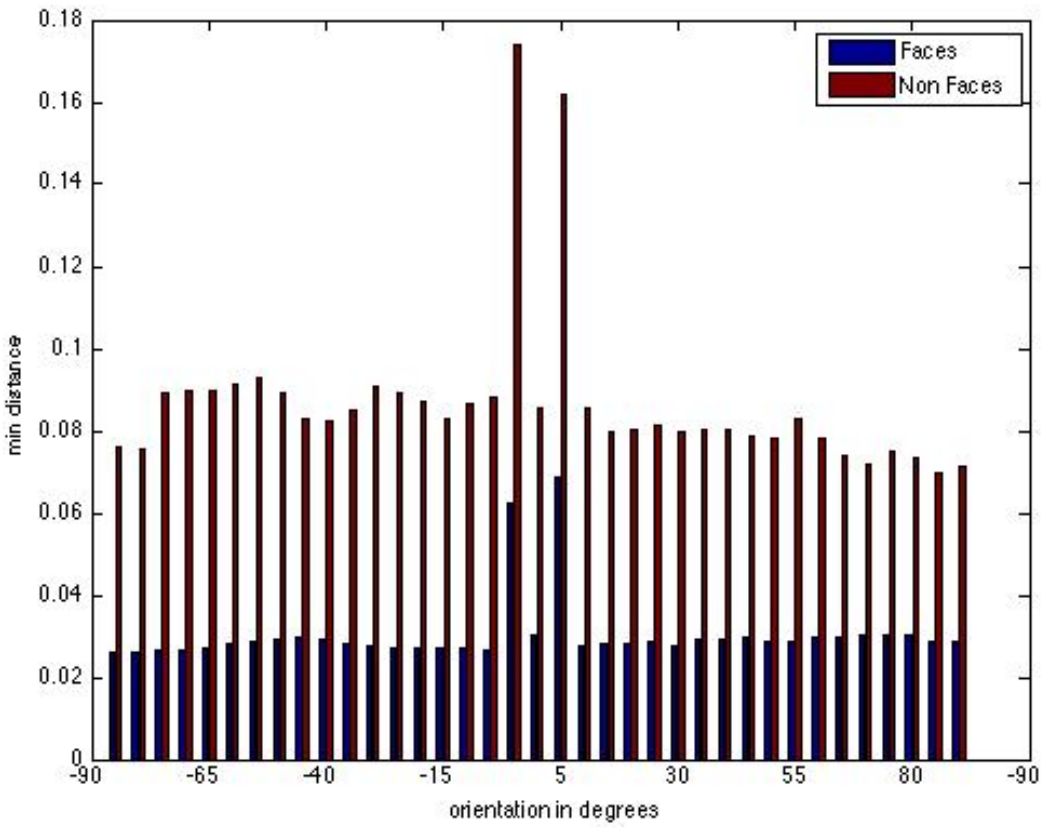

Figure 10: The minimum distance (i.e., distance of new images to nearest neighbor) is even more pronounced than the average distance.

With these statistics in hand, I can now perform classification. I tried several different approaches to classification, with none of them able to recapture the angle of the face with any tenable accuracy. However, a simple iteration through the orientations and maximum odds selection was able to differentiate between facial images and non-facial images with a true positive rate of .9995, a true negative rate of 1, a false positive rate of 0, and a false negative rate of .0005 (the only error was classifying one image that was a face incorrectly). These numbers are derived from running classification with 50 new face images at each orientation (a total of 1850 facial images) and 50 non-facial images. This is an improvement of the standard model of the eigenface (in which there is only one eigenface basis consisting of the principal components of forward facing images); running the algorithm only using the forward face basis with the same classification method yielded a true positive rate of 0.9422, a false negative rate of .05, true negative rate of 1, and a false positive rate of 0. Thus, my method classifies non forward looking faces with a higher accuracy than the standard eigenface method.

The odds were calculated by assuming the maximum, minimum, and average distances were normally distributed, and calculating normal probability density of each with the mean and standard deviations calculated previously for facial images and non-facial images. Simply adding the values of the normal probability density functions for the minimum, maximum, and average distances for facial images and

dividing that by the sum of the minimum, maximum and average distance normal probability density functions for non-facial images yields the odds of the image being a face (not seen before) at that orientation. [2] Perhaps a better approach would be to calculate the fisher information contained by the mean, maximum and minimum distance relative to the true classification of the image, and to weight the mean, minimum and maximum appropriately based on the fisher information carried by each variable.

Discussion

From this project, I have developed a method that can reduce the dimensionality of facial images and use this low dimensional representation of the images to quickly determine the similarity of a new, unknown image to facial images in my database. My dimensionality reduction method also lends itself to image compression in that most of the variance in a large database of images can be captured by storing just a few of the principal components and a vector for each image containing the weight of that image projected onto these principal components. As an auxiliary method, I also developed a simplistic image centering algorithm based on cosine similarity. While my classification method is able to very successfully classify facial images at different orientations, the false negative rate will likely greatly increase when facial images are under different lightings, different distances, or have different facial expressions. Obvious extensions of my project would be to attempt to correctly perform classification in some of these situations. Furthermore, I was unable to test the ability of my algorithm to classify faces that have been seen before due to a lack of duplicate images and time constraints on this project. In the future, I hope to finish this aspect of my project, and extend my project functionality to be able to successfully classify facial images under a broader range of circumstances.

References

[1] Turk, Matthew A and Pentland, Alex P. *Face recognition using eigenfaces.Computer Vision and Pattern Recognition* 1991. Proceedings IEEE Computer Society Conference on 1991

[2] Ruiz-del-Solar, J and Navarrete, P. *Eigenspace-based face recognition: a comparative study of different approaches* 2005

[2]This method performed better than simple thresholding. It might seem odd that a min distance value of zero might not get classified as a face because it will be several standard deviations away from the E[min]. Keep in mind that for the purposes of this classification, I am only interested in whether the image is a face that hasn't been seen before or not a face at all. Extremely small mean or minimum distance values would likely indicate that the image is an image seen before, which would be a different classification.