

The Use of Curvelets in Face Recognition

Andrew S. Barr

June 10, 2011

Introduction

The Problem and Historical Approaches

The problem my project addresses is; ‘Given an image of a face, determine if an image of the same person is stored in a database’. If the given image is exactly the same, the verification is trivial since one could just check if corresponding pixel values match. In all applications, the image will vary with respect to parameters such as head angle, lighting, facial expression, and hair style. These parameters are latent, and in general, images cannot be segmented in a way such that these parameters are isolated. The most common approach to the classification problem is to create a linear space that contains all the faces in the database and regards them as linear combinations of generated basis faces. Images can then be compared numerically by analyzing their projections onto a common basis. Let a vector that stores the projection coefficients be called a feature vector. Assuming that there are less face bases than pixels in each face image¹, this simplifies greatly the task of comparing images. The reduction of the comparison from $N * N$ pixels to a vector of size ‘length(feature_vector)’ is called dimension reduction. This idea assumes that a large number of faces can be accurately (perhaps not perfectly) reconstructed from a smaller set of basis faces.

Building Entry

My project was originally conceived to be applied to a building entry system. The idea is that face recognition would provide an additional level of security for those who use it. Limiting my image processing to face images taken by a building entry system greatly simplified my the classification task. This allowed me to assume that my images were of the same size, properly registered (generally oriented the same), had similar lighting, free of noise, and with the face taking up the vast majority of the image. It also forced me to consider computational efficiency because the recognition in such a system would ideally be both quick and accurate or if they covaried, properly balanced to achieve an acceptable mix of the two. At the onset of my project, I decided that a reasonable amount of time for a classification should be one

¹I am making the assumption that the images are all the same size

second. While the generation of the database is time consuming, the classification is quick and is close to that time when the EigenFaces or FisherFaces techniques are used. With the CurveFaces, the time is a little greater since a curvelet transform, thresholding, and inverse transform are taken before the comparison of the feature vector is even made. As of now, my classification scheme takes about that time. In spite of it failing to produce correct results, the correction to the error is likely not going to make a large runtime difference, as any modifications to make my code correct are likely to be small.

EigenFaces

The creation of a basis is not trivial because it must capture all of principal directions of variance within the data with a minimal sized set of vectors. Pentland and Turk, in 1991, pioneered the use of the Singular Value Decomposition (SVD) to obtain a face basis. They create a matrix where each image from a given set of images has been reshaped and stored as a column. The SVD on this matrix produces a basis that matches our criteria and can be shown to be optimal in doing so. That insures us that we will be able to use a relatively small amount of bases to store our images without substantial loss. Since the bases produced by this method look like an averaged face image, and the SVD involves the solution of an eigenvalue problem, this technique is called EigenFaces. The faces are called the principal modes of the decomposition.

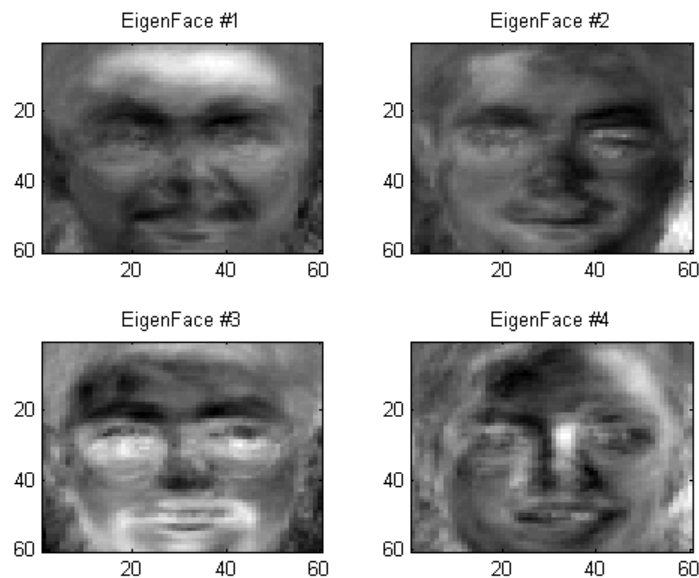


Figure 1: The first four principal Eigenfaces generated by my code.

FisherFaces

Another popular technique that is designed to handle databases with multiple images per person is called FisherFaces. FisherFaces were introduced in 1997 and were designed to maximize variance within a class and minimize variance from one class to another. It makes use of Fisher's Linear Discriminant, sometimes referred to as LDA (Linear Discriminant Analysis) for this. LDA, in the most general form, looks for an ideal projection of data onto a lower dimensional hyperplane. The 2d dimensional case can be seen as projecting a set of data points onto a line. In the ideal case, the projection of one class of points will fall onto a continuous segment of the line. Classification is then achieved by matching a data point to a segment of the line that corresponds to a class. When this is applied to images, to aim is to make the classification more robust to variations within different images of an individual. One such variation could be a lighting condition or a pose. Unless an environment is carefully controlled, it is possible that two images of the same person taken at different times will look substantially different. LDA takes these differences to account and allows for accurate region of classification. Since the calculation of the Euclidean distance between two feature vectors can serve as a metric for the closeness of two images, we can partition the space of the feature vectors, where each vector corresponds to a point in a space of dimension 'length(feature_vector)' into regions corresponding to a class. We can use either a Voronoi decomposition or K-means clustering to determine the exact boundaries of these regions. As the FisherFaces are not the focus of my project, I referenced and appropriated portions of code for the FisherFace create from A Brooks' personal website [?].

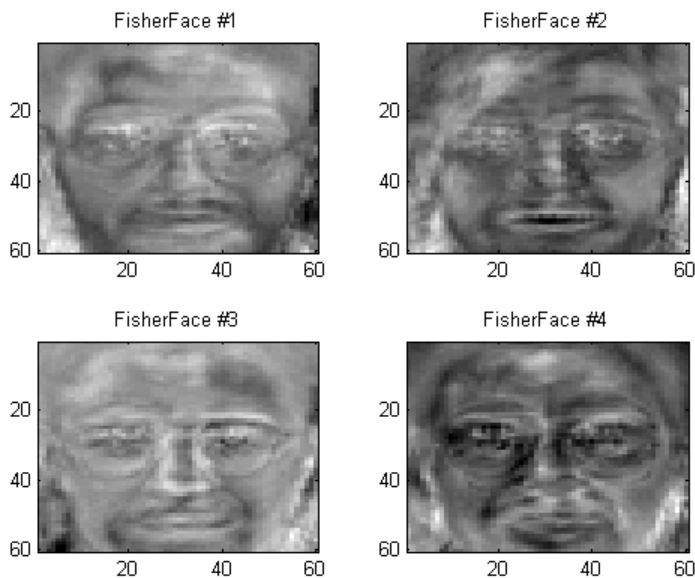


Figure 2: Example FisherFaces.

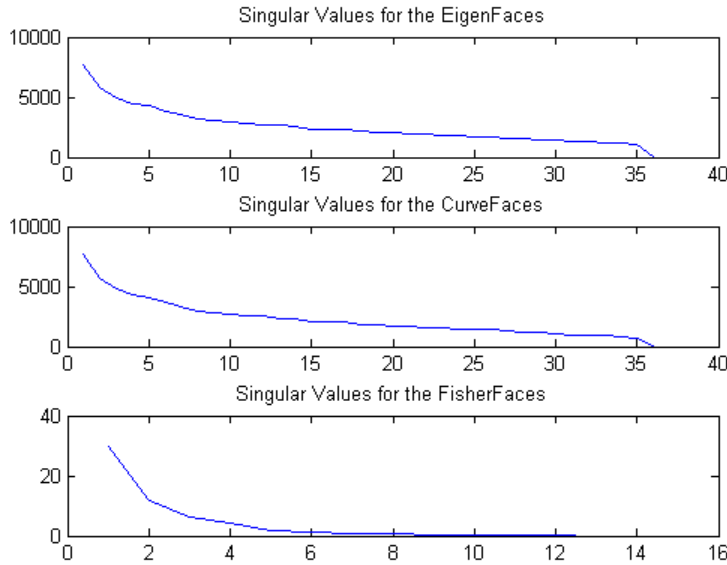


Figure 3: An plot of the magnitude of the eigenvalues.

My Approach using Curvelets Combined with other Techniques

My goal in my project is to effectively filter out unimportant² features of the face images. My hope is that I will be able to create an Eigen-basis is more a function of variance in the face so they principal modes decay quicker in their power. I propose to do this do this by using the curvelet decomposition, which was developed in its modern form by Candes and Donoho [Lem04] around the year 2000. The curvelet decomposition can be used to easily extract contiguous and curved edge features from an image. It does so at a number of different scales and is robust against small discontinuities. Individual curvelets are essentially rotated and translated lines. Since they are close to lines, the projection is like a line integral over a region. If a curvelet is being projected on top of the line, it will have a high coefficient. Since the transform does this at a few scales, the curvelet transform will have have much larger coefficients on edge and line areas. A simple thresholding of the coefficients leaves one with an image where these are isolated.

My hypothesis is that an image that has been decomposed into a small amount of curvelets will have a more sparse decomposition when the SVD is performed. I posit that the edge features play the largest role in determining a face, so the variance should be maximized in a way that reflects that. I experiment with the curvelet transformation on different sized images with different threshold levels. The recognition results obtained are compared to results that were found using EigenFaces and FisherFaces. A shortcoming of

²What makes a feature important or unimportant in an image is highly subjective. In this case, it should be part of the face, so there is no interference from something more transient, like clothing or hairstyle

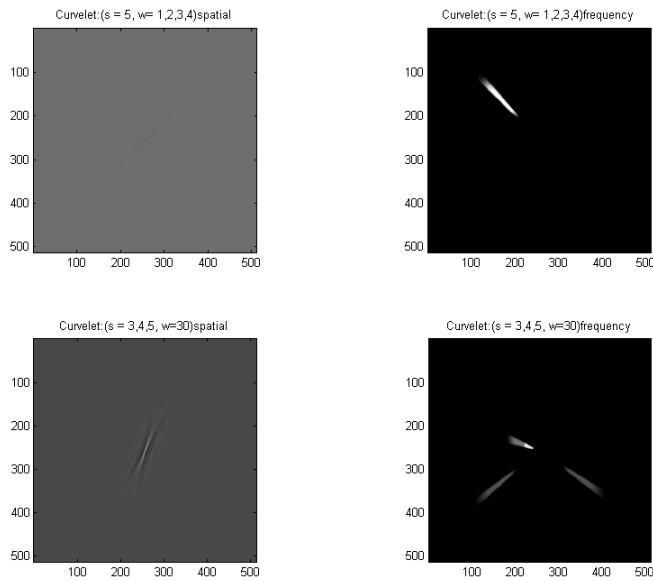


Figure 4: An example of curvelets.

my proposed procedure is that it may actually make the classification more difficult and I could not find a way to prove that it would not. Using the curvelet transform and then the SVD might result in a basis space that has larger reconstruction errors. By reconstruction errors, I refer to the inability of the generated basis, when limited to a few principal modes, to perfectly reconstruct the original images. To clarify, the test image would have to be first curvelet transformed and thresholded in the same manner as the database images before the projection onto the CurveFaces (the name for the faces generated by this procedure) is done.

If I am just interested in finding edges, then why do I not use a simpler technique to isolate them such as by wavelets or convolution masks? I saw this idea proposed in a paper³ where the author look at the finest scale wavelets. Not seeing the use of the Sobel operators for edge detection in face recognition worried me, but I felt that curvelets would work better because there is more control over parameters such as scales or angles. This could be useful if it were possible to match a certain scale to features such as eyes or mouth. Because of the digital implementation of the transform, only a limited amount of scales are possible, and I only use 3 scales for images of size $N = 45$ or $N = 60$. The guarantee of sparsity from the mathematical development is what makes curvelets the subject of much recent research in face recognition. Sparsity implies that variance is stored in only a relatively small amount of coefficients, so I should be able to obtain a low-dimensional comparison from the curvelet coefficients. The exact details of implementation and data structure of the Discrete Fast Curvelet Transform (DFCT) are out of the scope of this paper, but a curvelet transform

³'Human Face Recognition using PCA on a Wavelet Subband' by GC Feng.

generates an amount of coefficients that are at least an order of magnitude greater than the number of pixels. However, from the sparsity, we know that the vast majority of these are relatively small and can be turned to zero without a significant loss of representation accuracy. The amount of coefficients varies between the different curvelet parameters of scales and angles.

Other Work with Curvelets in Face Recognition

One paper I looked at, from Wei, Sun, and Yin, uses a ‘Common Vector’ obtained by looking at the variance and covariance of the curvelet coefficients at different scales. The classification method is completely different than the Euclidean distance based method that I used for my implementation. It is based on the error from reconstruction of the test image. I attempted to implement this but had trouble succeeding since it uses a technique called Discriminate Power Analysis (DPA), which was recently developed and for which I had little resources. It also delved a bit more into the techniques using feature vectors that I only just started looking into.

Another possible approach, proposed in a paper by Mandal, Majumdar, and Wu [MW07], investigates curvelet transforms of 2,4, and 8-bit images for face recognition. His approach uses Support Vector Machines (SVMs), which he describes as being related to Neural Network classifiers. I just checked out a book that describes Neural Networks and was not able to implement anything to try to replicate the code. The paper claims success but does not give much detail about the classifier other than that it was a One-Against-All (OAA) SVM.

Contemporary Methods

A common thread in all of these classifications is the need for dimension reduction. For almost any given basis, one can find a paper about its use in face recognition. The important thing is that the structure is geared for the image. For instance, a bit plane decomposition or a quad-tree representation holds structure about an image, but that structure may not be relevant or easily usable for face images. There are no shortage of papers regarding curvelets and face recognition and the interest is still there. Another modern technique of interest is called LaplacianFaces and it uses LLE (Locally Linear Embedding). They claim increased accuracy over other techniques and the avoidance of some common numerical issues in the FisherFace generation. I did an undergraduate research project on LLE when I arrived at UW so I am interested in pursuing this further.

Since Chris Ding published his 2005 paper on the ‘2dSVD’, there have been numerous investigations into the extension of SVD, LDA, and PCA into a purely ‘2d’ form. The claim is that the reshaping of the images into column vectors removes important column-column information from the SVD. Because Ding’s original paper included face and mentioned face recognition applications in it, I felt that it could be an important part of my research. I anticipated replacing the standard SVD with it, because I wanted to use anything that provided a claim of higher accuracy. I spent a significant amount of time on my project

implementing 2dSVD and it ultimately was never used. I spent time trying to work with 2dLDA and 2dPCA but did not have the chance to get it finished to be used for this project.

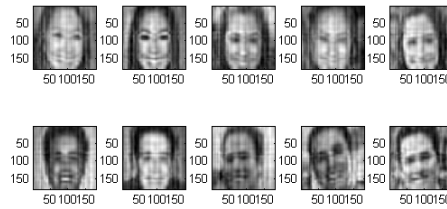


Figure 5: A 2dSVD reconstruction using just 15 eigenvalues.

Programming Issues

Data Reading

One of the most time consuming tasks of the project was to input a database of images into my functions. I used the Georgia Tech Face Database, created by scientist Ara Nefian of Carnegie Mellon University. The database includes, for 50 people, 15 images each. The images include quite a large amount of variation in facial expression, viewing angle, and lighting. Since these were things that I considered out of the scope of the building entry software, I had to manually select shots that were close to straight on from the Even though I used the precropped images, I still had to crop and resize them. Since I was resizing the images uniformly, I assumed that it would not impact the classification. I also had to learn how to use the UNIX ‘find’ command in order to sort them and select out a subset of them for me to use.

The next step in the uploading process was to convert the images from RGB to grayscale. I decided to include an option to equalize the histogram of the images using the built-in ‘histeq’ function. In some of the papers, and in the course notes from Professor Kutz of UW, they advise subtracting the mean of the set of images from each individual image. I do that in the calculation of the SVD, and from my understanding, it is done for linearity reasons. I was originally hesitant to do it because I knew that having negative values in an image normally leads to images being clipped. It is always possible to rescale them using the ‘imagesc’ command, but I was worried about information loss, so I saved the mean to be used in reconstruction.

Database

At first I experimented with the Georgia Tech Database which is hosted at the website of Ara Nefian. For each subject, there are 15 images with various angles. I wanted to implement

Fisher's discriminant so I needed multiple images per individual. This fits with a building entry system in that it is not unreasonable for building security to ask someone to take a few pictures for their security clearance. The database is freely available for download at http://www.anefian.com/research/GTdb_crop.zip this website. I considered making the switch to ORL or YALE. I was interested in the Yale Database because it is large and the poses are clearly marked within the filename. The background is not entirely removed in some pictures, so I test image would have to include the same background. This is not an issue for building security because the training image and test image would be taken under the same controlled environment. The Yale images also have 10 different levels of luminance and so it would have been ideal for testing FisherFaces. The ORL database is widely used, but it seems to suffer from the same perceived shortcomings in the Georgia Tech Database. There are many variations in some images and they are not consistent with respect to the index. For a human, the differences between the images is negligible, but for my current classification program there are already issues, even when I select close test images. When, I have a working procedure, I will verify it on the ORL.

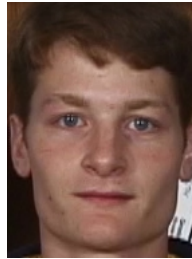


Figure 6: A sample face from the database.

Memory and Speed

I originally intended to use 180×180 images for all of my work. I did this because I felt that the resolution was reasonably high and it would allow me to have flexibility in choosing a high number of different scales of curvelets⁴. I had no computational errors computing the SVD of large databases. The implementation of the curvelet transform that I used, part of a package made by Gabriel Peyre⁵ [Pey06], is fairly computationally intensive. For an $N \times N$ image, it is approximately equivalent to 8 FFTs. I have had no trouble doing this two times⁶ for each of 50 images in a reasonable amount of time. My first computational hurdle came when I was implementing FisherFaces. The S_b and S_w matrices are of dimension $N^2 \times N^2$ and so I would have memory errors in MATLAB when I would try to create an $180^2 \times 180^2 = 32400 \times 32400$ matrix. No increase in my Windows paging file would allow

⁴For an $N \times N$ image, we can only generate curvelets up to $\lceil \log_2(N) - 3 \rceil$

⁵<http://www.ceremade.dauphine.fr/~peyre/numerical-tour/links/>

⁶The forward and inverse curvelet transforms are the same

me to create matrices with over a billion entries of type double. I tried reducing the image size by half, but I still had memory errors with size $90^2 \times 90^2$. I eventually recreated my ‘gen_database()’ method to be able to resize the database images to either 45×45 or 60×60 . While this succeeded in creating the FisherFaces, I feel that it diminished the power of the curvelet transform. I had this thought because the curvelets looked more blocky and discontinuous than I had ever seen. At the smallest scale, unless specified otherwise, the standard option in the ‘perform_curvelet_transform’ method is to use wavelets. I intend to compare the CurveFaces results with a waveletFaces result so I changed the option to use curvelets at the smallest scale. From looking at the image, it is hard to notice a difference from wavelets since everything appears to come in square blocks. This could also be a relic of the downsizing of the test image to an image that is about the fourth of the original size. A benefit that the smaller sized images brought was a substantial speed up in generation and identification time that took me closer to my goal of identification within one second.

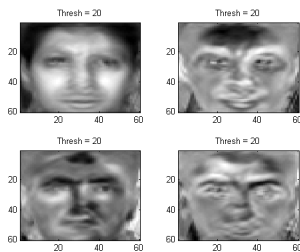


Figure 7: An example of CurveFaces with a 60×60 image.

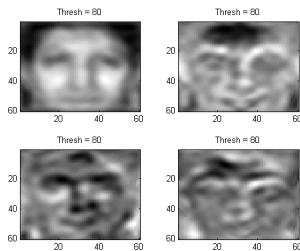


Figure 8: This image shows the limitations of a small image with curvelets

Indexing, Databases, Program Flow

Deciding how to store and operate on my images was a non-trivial part of my project. Since every image is associated with a class, corresponding to which person the image belongs to, I created the vector ‘class_vec’ which maps indices to the associated class. I had to learn a number of MATLAB commands such as ‘find’ to help with my array processing, but it

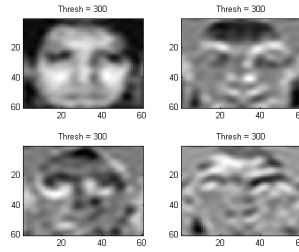


Figure 9: I'm not sure if this produces better or worse discrimination.

substantially helped me to clean up my code. I decided to use matrices rather than cell array to store my images since my manipulations all treated them as vectors and it was simple to operate on them using the `:` operator. I had little prior experience with MATLAB so my original idea was to store the images into cell arrays. This complicated my implementation and at one point, I had to redo my code entirely. For my testing of the curvelet thresholding values, I created a cell array that held a few 'A' arrays, which hold the images in the columns.

There are three different m-files that make up my program. Initial creation of the database with the GT face images is taken care of by `'gen_database_fisher()'`. It generates databases of a few pre-defined sizes, and once the database is created, it can be passed into the testing method, `'im_test_cr()'`, and reused without regeneration. `'im_test_cr'` takes an image, processes it, and returns the result of the tests with EigenFaces, FisherFaces, and CurveFaces. `'test_presentation'` shows a way that I have automated my testing.

Mathematical

While the mathematics behind the classification techniques was fairly straightforward, there were some areas that caused me trouble throughout my code and they are discussed here.

Subtracting the Mean

As one who examines my code will size, one of the most contentious issues I had during the procedure was deciding what to do with the mean of the image. As with any image subtraction, my immediate concern that is that subtraction will result in clipping which ultimately is a loss in data. Some papers made a mention of doing this and others did not. In Professor Kutz's notes for his Amath 582 class, he presents the subtraction as part of the SVD calculation process and gives a one-line way to do that with the image-column matrix. It centers the data at 0, and since we are looking for a linear subspace, it seemed to make sense that we put our original image into a certain range. While I never got a rigorous mathematical answer, my classifier worked better in the case with the subtract mean so I kept that within my code. Another question I never answered was where to subtract the mean, and what mean should I use. I investigated doing this upon upload and using the mean on

of the training set on the test image, and looked at quite a few combinations fo doing this at various to make sure that I was not comprimising the results of the rest of my code. In the end, I decided to subtract the mean right before taking the SVD, and I subtract the mean of the training set from my test image.

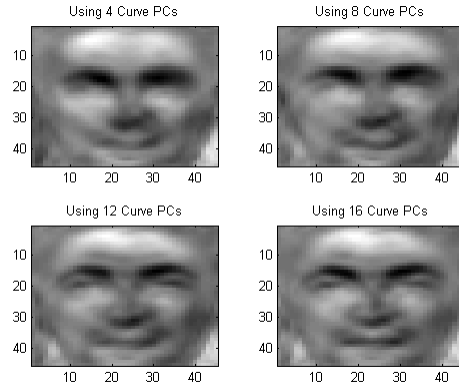


Figure 10: The projection of the test image onto CurveFaces.

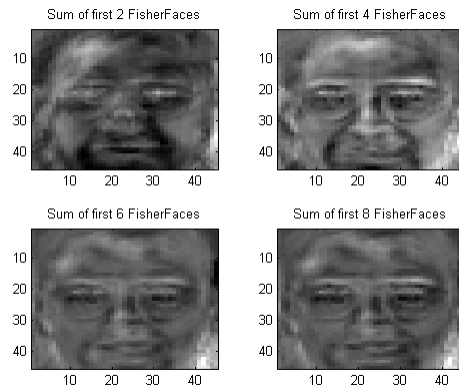


Figure 11: The projection of the test image onto FisherFaces.

Classification

Statistical classification and everything related to feature vectors were new to me this quarter. I had to spend quite a bit of time learning about these ideas on my own. While the closest distance and k-means classifiers are easy to understand, I had a lot of trouble understanding multi-class LDA and some of the other methods that I encountered. I have yet to take a single statistics course so when I did any of my reading, I needed to look up a lot terms along

the way. In the future, I would love to find a class at UW where I can continue to learn about this. It seems that classification is a very basic level problem in statistics and most papers I found only mention the classification technique briefly, and only elaborate in the case of more specialized feature vectors.. For now, I have checked out the book, ‘Statistical Pattern Recognition’ by Andrew Webb, and I have started to read about perceptrons. The explanations so far are clear and I looking forward to being about to know a little about Neural Networks and SVMs since I see them mentioned often in the literature.

Conceptual

Before starting this project, I already understood the concepts of SVD, PCA, and LDA fairly well. I was curious about affect of reshaping images using the SVD and started looking for work by others on this. Mainly, since I was working with 2 dimensional bases, I wanted to find an analog to the inner product and the vector bases. I already knew about the using tensor products of 1 dimensional bases, but with curvelets, they are unable to be expressed in such a way so I needed to look for a better way to understand. The papers that I found about the 2dSVD, 2dPCA, and 2dLDA were originally complex to me, but I eventually was able to understand them. While these were interesting papers, and I experiment with the results, at the end, they did not contribute anything to my classification program and ended up significantly detracting from the time I could have spent fixing my classifier algorithm. I know this class is not a math class, but I spent a large amount of time reading papers and really trying to understand the way others achieved recognition or specialized decompositions rather than implementing them. Since the clustering and classification are barely mentioned in the papers I read, I looked online for tools to help me visualize my data and find I am continuing to look for a clustering toolkit to help me investigate other ways of segmenting my data.

Code Overview

Generation of the database: `gen_database_fisher(n)`

This code creates the image database that is used when testing an image for set membership. From the beginning, one first specifies the number of classes they want to include in the upload and also the amount of pictures per class. Currently, I have 21 classes available with 3 images each. In the future, I would like to make it possible to use all 15 images per class. For now, the field ‘pics_per_class’ is a constant set to 3. The other parameters are described in detail with in-line comments. Because I am working with so many parameters, I have attempted to automate my testing as much as I can. For this reason, I include a number of diagnostic switches that trigger plotting and other variations of the main process. The first half of the code places the images and the curvelet transformed images into matrices of size $(N * N) \times \text{total_pics}$, so each column is a reshaped picture vector. The SVD is taken and the principal modes are extracted and stored in a database as specified in my code. FisherFaces

are also generated, following the process described at ScholarPedia and by the original paper describing FisherFaces.

Testing of an image: `im_test_cr(test_im, db)`

This portion of my code takes in a test image, computes its projections onto the principal values, and then uses the Euclidean distance as a method of classification. For the curvelet test, the test image must be decomposed and thresholded at the same level as the original database pictures were. There are many plots that are generated when some of the diagnostic switches are turned on, but the actual classification is fairly quick. I did not profile the individual tests, but I think that the check for the EigenFace images is close to the 1 second that was my goal. I am having an issue using the tic/toc commands (I am returned a number in the range 30000 seconds) so I do not have reliable figures.

Automated Testing: `test_presentation()`

I created this section to help me with my testing. There are so many possible parameters in my programs that I wanted to be able to minimize the set up time for each test. This divides the testing into two sections, corresponding to the cases $N = 45$ and $N = 60$ where N is the number of pixels. I made a series of booleans to determine the tests that are run. At the top of each set is the corresponding database generation command. It must be set to ‘true’ if one wants to use any of the tests under it. Results specific to each test are contained within the comments. The way my code was written makes it easy to add more tests. I intend to make the `im_test_cr` have an output so that I can automate the collection of information. This kind of information includes timing, accuracy rate, the class, etc.

Results

At this stage, my face detection programs are very imperfect and it is hard to definitively give any results. For my test of an exact image contained in the database, my EigenFace and FisherFace classifier correctly determined the class. This failed for the CurveFaces which is alarming. My second test was for an image that was very close to its corresponding images in the database. Only EigenFaces achieved a positive result. For test3, I wanted to confirm that at least my EigenFace classifier was working, and that I was not just getting a positive result from a magic number in the code or just coincidence. Fortunately, my test on a different image was a success for Eigenfaces. What is interesting is that my CurveFaces classifier also made the correct detection. So I decided to check to see if there were other cases in which my CurveFaces made a correct classification. My next test, test4, was to show me the effects of hair on detection. Everything about the face, including the mouth, eyes, nose, lighting, and angle matched, but there was a minor difference in hair style. While some of the papers claimed the ability to detect an image correctly in spite of such variations, my implementations all failed. My test5 shows how large of an impact registration can have on

the detection. I found an image that was very close to an image in the database, except very slightly and uniformly translated. Every single test again failed to produce a correct classification. Test6 was to see how much glasses affected the classification. For each person in the Georgia Tech database, some pictures include glasses and others do not. For this reason, the faint presence of glasses can be seen in the EigenFaces. I was presently surprised to see that my EigenFaces and CurveFaces correctly classified the test image of a man who was only wearing glasses in the test image. The results for the $N = 60$ portion indicate some serious shortcomings in my generalization I attempted in my code for EigenFaces and CurveFaces. However, I did get two positive identifications from FisherFaces so I feel that the code I made for that portion was not completely useless.

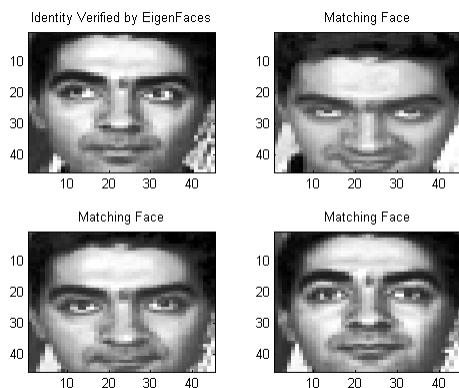


Figure 12: An Example of EigenFace Verification.

Conclusions

At this point, I cannot say if my hypothesis that the curvelets decomposition before EigenFace generation would help achieve better results was verified. The magnitude of the similar values are similar to the CurveFaces which indicates to me that my hypothesis is wrong. I still would like to see this graph as a function of different levels of curvelet thresholds to see if there is any difference. Until I implement a better classification scheme and automate my testing, it is hard to know the exact cause of the failure.

I also understood why I do not see Curvelets and FisherFaces together in the literature, in spite of always seeing EigenFaces and FisherFaces together. Because of the eigenvalue computation for the $N^2 \times N^2$ image, one quickly runs into memory issues with fairly small FisherFaces. Curvelets, on the other hand, need images of a certain size, or else they are very close to the EigenFaces or WaveletFaces result, depending on the threshold.

Another thing I became aware of was the difficulty of the detection problem. I assumed that small deviations in angle, hair, and lighting would have only a minor effect on the detection. This was from what I read in papers. I came to see how important it is to extract an appropriate feature set and remove all the extraneous data possible.

Ideas for Improvement

In spite my attempts at the careful application of the curvelet transform, I still found that much of my images contain a lot of extra information about clothing and background. While doing a google search on FisherFaces, I came across an image that showed a possible way that I could preprocess my image to remove this. I could select an oval region that isolates the face, and then zero the background. Ideally, as much of the hair as possible would be cropped from the picture. The region could also be a box that encompasses the eyes, nose and mouth, our main face features. While I am sure this could be done algorithmically, using vectors generated for face detection, in a building entry system, it is feasible for this to be done manually, at least the database creation. Software could be made so that the user could select out the region and then that could be the database image. The only downsides to this are that it assumes that new entries are being added to the database one at the time, and it is subjective and could introduce errors. Test images would have to be similarly decomposed or a more sophisticated classification method that takes this into account developed.



Figure 13: An Example of better feature extraction.

I am also interested in emailing some of the paper authors about obtaining their code. I believe strongly in reproducible research so it bothers me that I found so little to help one get started with this project. For how many papers there are and researchers with this on their webpages, there is a dearth of public code for this. I only was able to find protected p files and someone's pseudocode. It may be because uploading the images requires setting paths, but one should be able to just create self contained archive and share this. The subset of the George Tech database that I used was only about 4 megabytes so there is no reason that this is not feasible. I intend to continue improving my code for this project and eventually putting it on my own personal website.

Something I was unable to implement was a suitable reject criteria. For now, I just looked at the closest match in the database, but in a building entry system, it is just as important to know if someone is not in the database. Since my threshold values are hand-determined, it is a matter of examining the data. To obtain my current ones, I looked at the average magnitudes of my feature vector and tried to make a reasonable number. However, there is no mathematically derived reason for my choice, and I tried to make it a function of input

image size. I think that the solution to this is more straightforward when other classifiers are used.

Code

gen_database_fisher.m

The following is my main code.

```

1
2     function [ db ] = gen_database_fisher( in_n ) % Make a global n
3 %GEN_DATABASE Generates the database for interactive classification.
4 % db object is specified below:
5 % db has the following structure:
6 % db{1} = A           - This is the set of images, stored as columns.
7 % db{2} = A_cr       - Curvelet threshed images as columns
8 % db{3} = P           - These are the EigenFaces
9 % db{4} = P_cr       - These are the CurveFaces
10 % db{5} = w           - These are the EigenWeights
11 % db{6} = w_cr       - These are the CurveWeights
12 % db{7} = class_vec  - This is the vector that stores class data.
13 % db{8} = m_A        - This is the mean of A (
14 % db{9} = m_A_cr    - This is the mean of A_cr.
15 % db{10}= options    - These are the curvelet options in a struct.
16
17 %Make sure to clear variables before using!
18 clc;
19
20 %% Set the path for the toolboxes I am using.
21 % This assumes my archive is unzipped entirely into one folder.
22 getd = @(p)path(p,path);
23 getd('toolbox_signal/');
24 getd('toolbox_general/');
25 getd('CurveLab-2.1.2/');
26 getd('CurveLab-2.1.2/fdct_usfft_matlab');
27 getd('CurveLab-2.1.2/fdct_wrapping_matlab');
28 getd('GTA');
29
30 %% My Upload Parameters.
31 num_class      = 12;           % The number of different people in the db (up to 21)
32 pics_per_class = 3;           % For now, this is a constant. (only 3)
33 total_pics     = num_class*pics_per_class; % Total pictures in db.
34 class_vec      = zeros(total_pics,1);    % A vector that relates class to index
35 I              = cell( (total_pics), 1); % A cell array that holds all the images.
36
37 % UNCOMMENT IF THERE ARE SIZE ERRORS
38 % disp(['Input n =', num2str(in_n)])
39
40 % Hand-tuned Parameters (Specific to my image-set)
41 n          = in_n;           % Dimensions of image. I pre-determined this and crop to this size.
42 np         = n * n;          % Number of pixels.
43 npc        = 16;             % Number of Principal Components (Must be less than total pics)
44 bw_thresh  = 0.4;           % This is thresholding if we want to make images binary.
45
46 %% Runtime Parameters
47 plot_modes = true;          % This is a method that shows the curve+eigen-faces.
48 check      = false;        % Shows images throughout the code. Makes it substantially slower.
49 hist_eq    = true;         % Equalizes the histogram of each image.
50 sub_mean   = true;         % Subtracts the mean from each image. (Before subbing the mean of them all.)
51                                     % True subs the mean from the SVD. False doesn't
52 plot_Xfaces= true;         % Includes plots for the first 4 Eigen,Curve, and FisherFaces.

```

```

53 eig_val_plot=true; % This is the plot for the three Singular values. It is expensive.
54 % Extra Experimentation
55 wavelet_dc = false; % Done for comparison.
56 SVD2D      = false; % Generates the 2dSVD of the data with npc modes. Symmetric in rows/columns.
57 binary_img = false; % Use this to create a database of binary images.
58
59 %% Curvelet Parameters
60 options.thresh = 30; % A higher number does more thresholding. (do 55 for 180. 20 for 60)
61 options.null=0;
62 options.finest=1; % Use curvelets(1) or wavelets(2) at finest scales.
63 options.nbscales=3; % default is ceil(log2(min(M,N)) - 3) (3 for n==60, 3 for n==45)
64 options.nbangles_coarse=8; % Multiple of 4. Minimum 8
65 options.is_real =1; % real(1), complex(0)
66 options.n = n; % The size. This implies only square images.
67
68 %% Cell Object Creation (These may be replaceable by below)
69 I = cell( (total_pics), 1); % A cell array that holds all the images.
70 I_cr = cell(total_pics, 1); % Store the transformed images.
71 I_cr_th = cell(total_pics, 1); % Store the thresholded images
72 I_cr_inv = cell(total_pics, 1); % Store the thresholded images
73
74 %% Create the large matrices for SVD.
75 A = zeros((n*n), total_pics);
76 A_cr = zeros((n*n), total_pics);
77
78 %% Preprocessing of the images.
79
80 % This works with n==45, n==60.
81 % Load, resize, and convert to grayscale.
82 tic;
83 count = 1;
84 for k = 1:num_class
85     for j =1:pics_per_class
86         % String Processing
87         % For class less than 10
88         if (k<10);
89             tmp_str = ['0' num2str(k)];
90             conc = ['s' tmp_str];
91         else
92             conc = ['s' num2str(k)];
93         end
94         % For each image
95         if (j<10);
96             tmp_str = ['0' num2str(j)];
97         else
98             tmp_str = num2str(j);
99         end
100        fin_str = [conc '_' tmp_str '.jpg' ];
101
102        % Store the class.
103        class_vec(count,1) = k;
104
105        % Read in the image.
106        I_tmp = imread(fin_str);
107
108        % To implement FisherFaces, I NEED smaller images.
109
110        if (n == 60)
111            I_tmp1 = imresize(I_tmp, [80,60]);
112            I_tmp1g = rgb2gray( imcrop(I_tmp1,[1 10 60 59 ] ) ); % [xmin ymin width height]
113        else % I am only giving n == 60 or n == 45
114            I_tmp1 = imresize(I_tmp, [60,45]);
115            I_tmp1g = rgb2gray( imcrop(I_tmp1,[1 7 45 44 ] ) ); % 60 x 60 and 45 x 45 images.
116        end
117

```

```

118     % Equalize the histogram (Decide if this is good for FisherFaces)
119     if hist_eq
120         I_tmp1g = histeq(I_tmp1g);
121     end
122     I{(count)} = double(I_tmp1g);
123
124     % More involved processing
125
126     % Subtract the mean. (Causes clipping)
127     %if (sub_mean)
128     %     I{count} = I{count} - sum(mean(I{count},2));
129     %end
130
131     % Binarization
132     if (binary_img)
133         I{count} = im2bw(I{count}, gray(256), bw_thresh );
134     end
135
136     % Apply the appropriate transforms
137
138     I_tmp1           = I{count}(:,:,);
139     I_cr{count}      = perform_curvelet_transform(I_tmp1, options);
140     I_cr_th{count}   = perform_thresholding(I_cr{count}, options.thresh, 'hard');
141     I_cr_inv{count}  = perform_curvelet_transform(I_cr_th{count},options);
142
143     % Store imagesets into columns for SVD
144     A(:, (count))    = reshape(I{count}, (n*n), 1);
145     A_cr(:, (count)) = reshape(I_cr_inv{count}, (n*n), 1);
146
147     % Visual Check of uploading.
148     if (check)
149         figure(1)
150             subplot(2,1,1)
151                 colormap (gray)
152                 imagesc(I{count});
153                 title('Original Image')
154                 axis image
155             subplot(2,1,2)
156                 colormap (pink)
157                 imagesc(reshape(A_cr(:,count), n,n))
158                 title('Curvelet Decomposition')
159                 axis image
160             drawnow;
161             pause;
162     end
163     count = count + 1;
164 end
165
166 disp('Database Creation Complete:')
167 disp('Image uploading, basic processing, and transformation took ')
168 toc;
169
170 % I can clear all of the I cells by here
171 clear I
172 clear I_cr
173 clear I_cr_th
174 clear I_cr_inv
175
176 %% Calculate the basic EigenFaces and Curvefaces.
177 m_A           = mean(A,2); % This is a column mean, so an image mean.
178 m_A_cr        = mean(A_cr,2);
179 A_m           = A - repmat(m_A, 1,total_pics);
180 A_m_cr        = A_cr - repmat(m_A_cr, 1,total_pics);
181
182 % Only change here. And, mean is passed on now.

```

```

183 if (sub_mean)
184     [U, S, V] = svd(A - repmat(m_A, 1,total_pics), 0);
185     [U_cr, S_cr, V_cr] = svd(A_cr - repmat(m_A_cr, 1,total_pics ), 0);
186 else
187     [U, S, V] = svd(A, 0);
188     [U_cr, S_cr, V_cr] = svd(A_cr, 0);
189 end
190 % Extract the principal Eigenvalues.
191 % These are useful to find the rate of decay in the modes.
192 sigma = diag(S);
193 sigma_cr = diag(S_cr);
194
195 % Take out the first npc EigenFaces (npc = Number of Principal Components).
196 % These are used for projections.
197 P = U(:, 1:npc);
198 P_cr = U_cr(:, 1:npc);
199
200 %% FisherFace generation
201 % NOTE: I RECEIVED HELP AND USED CODE ON THIS SECTION FROM THE REPORT POSTED ON
202 % DAILYBURRITO.COM.
203 % Please see my references for the exact link. I intended to spend the
204 % most of my time looking at curvelet classification so this
205 % section was extra.
206
207 % Create the S_b matrix
208 % This code depends on the BAD assumption that each class has an
209 % equal number of members.
210
211 % Sb is formed iteratively, by adding together the covariance of
212 % each class.
213
214
215 % Use subtracted mean, image matrix A_m, A_m_cr
216
217 Sb = zeros(np); % Dim = [np np]
218 class_means = zeros( np, num_class); % Dim = [np num_class]
219 for i = 1:num_class
220     cl_ind = find(class_vec == i); % Indices for every member of the class.
221     class_mat = [A_m(:,cl_ind(1)), A_m(:, cl_ind(2)), A_m(:,cl_ind(3))]; % This is hard coded f
222     class_mean = mean(class_mat, 2);
223     class_means(:,i) = class_mean;
224     Sb = Sb + ( (class_mean - m_A)*(class_mean - m_A)' );
225 end
226
227 Sw = zeros(np);
228 for i = 1:total_pics
229     % This uses class vector to reference the corresponding class_mean
230     % to the class_vec
231     tmp = A(:,i) - class_means(:,class_vec(i));
232     Sw = Sw + ( tmp*tmp' );
233 % Sw = Sw + ( ( A(:,i) - class_means(:, class_vec(i)) )*( A(:,i) - class_means(:, class_vec(i))' ) )
234
235 end
236
237 % Solve the eigenvalue problem (SbV = SwVLambda)
238 % P comes from the EigenFaces
239
240 Sbb = P.'*Sb*P;
241 Ssw = P.'*Sw*P;
242
243 % Find Eigenvectors and Eigenvalues. (D_f stores them on the diagonals)
244 [V_f, D_f] = eig(Sbb,Ssw); %eig finds generalized eigs for Sbb and Ssw (both are square)
245 Ds = diag(D_f);
246
247 % Print out D_f, make sure I am getting the eigenvalues

```

```

248     % I don't use these for anything
249     [tmp,index]= sort(abs(Ds),'descend');
250     % index is the order of eigenvalues.
251     % They are from largest abs to smallest abs
252     eigVals = Ds(index);
253     eigVecs = P*V_f(:,index);
254
255     % My eigVecs are my principal components
256     w_f2      = eigVals(1:npc);
257     P_f       = eigVecs(:,1:npc);
258
259     % Normalize to calculate weights.
260     [nrowsP, ncolsP] = size(P);
261     [nrowsP_cr, ncolsP_cr] = size(P_cr);
262     [nrowsP_f, ncolsP_f] = size(P_f);
263
264     for j = 1: ncolsP
265         P(:,j) = P(:,j) ./ sqrt( sum( P(:,j).^2 ));
266     end
267
268     for j = 1: ncolsP_cr
269         P_cr(:,j) = P_cr(:,j) ./ sqrt( sum( P_cr(:,j).^2 ));
270     end
271
272     for j = 1: ncolsP_f
273         P_f(:,j) = P_f(:,j) ./ sqrt( sum( P_f(:,j).^2 ));
274     end
275
276
277     % Calculate the weights for each A in the classifier.
278     % This is just the dot product of each mode with each image.
279     w      = P' * A;
280     w_cr   = P_cr' * A_cr;
281     w_f    = P_f' * A;
282
283
284     %% DISPLAYS/DIAGNOSTICS
285
286     if (plot_Xfaces)
287         % Show the EigenFaces
288         if (plot_modes)
289             figure(6)
290             for i = 1:4
291                 colormap (gray)
292                 subplot(2,2,i)
293                 imagesc(reshape(P(:,i),n,n))
294                 title(['EigenFace #', num2str(i)])
295             end
296         % Show the CurveFaces
297         figure(7)
298         for i = 1:4
299             colormap (gray)
300             subplot(2,2,i)
301             imagesc(reshape(P_cr(:,i),n,n))
302             title(['CurveFace #', num2str(i)])
303         end
304         % Show the FisherFaces
305         figure(8)
306         for i = 1:4
307             colormap (gray)
308             subplot(2,2,i)
309             imagesc(reshape(P_f(:,i),n,n))
310             title(['FisherFace #', num2str(i)])
311         end
312     end

```

```

313 end
314
315
316 %% Show all of the eigenvalues/weights
317
318 if (eig_val_plot)
319
320     range = 1:length(sigma);
321     figure(9)
322     subplot(3,1,1)
323         plot(range, sigma)
324         title('Singular Values for the EigenFaces')
325     subplot(3,1,2)
326         plot(range, sigma_cr)
327         title('Singular Values for the CurveFaces')
328     subplot(3,1,3)
329         % I only selected out for npc amount of Eigenvalues
330         plot(1:length(eigVals), eigVals)
331         title('Singular Values for the FisherFaces')
332 end
333
334     % Look at this in a second
335 % Store the items in the database to be passed to the classifier.
336 db = cell(7,1);
337 db{1} = A;
338 db{2} = A_cr;
339 db{3} = P;
340 db{4} = P_cr;
341 db{5} = w;
342 db{6} = w_cr;
343 db{7} = class_vec;
344 db{8} = m_A;
345 db{9} = m_A_cr;
346 db{10}= options;
347 db{11}= w_f;
348 db{12}= P_f;
349 db{13}= w_f2;
350
351 disp('Database Creation Complete')
352 end

```

im_test_cr.m

The following is my test method.

```

1
2 function [ results ] = im_test_cr( test_im, db)
3 %im_test_cr
4 % Classify the image using Euclidean distance classifier.
5 % My feature vector is my test_image projected onto the principal
6 % modes of the training set.
7
8 % db has the following structure:
9 % db{1} = A - This is the set of images, stored as columns.
10 % db{2} = A_cr - Curvelet threshed images as columns
11 % db{3} = P - These are the EigenFaces
12 % db{4} = P_cr - These are the CurveFaces
13 % db{5} = w - These are the EigenWeights
14 % db{6} = w_cr - These are the CurveWeights
15 % db{7} = class_vec - This is the vector that stores class data.
16 % db{8} = m_A - This is the mean of A.
17 % db{9} = m_A_cr - This is the mean of A_cr.
18 % db{10}= options - These are the curvelet options in a struct.
19 % db{11}= w_f; - These are the FisherWeights

```

```

20 % db{12}= P_f;          - These are the FisherFaces
21 % db{13}= w_f;          - These are just eigVecs of Fishers
22
23 %% Notes
24 % Dimensions
25 % proj_onto_eigs The projection of test onto each pc. [npc x 1 ]
26 % eig_weights The npc components of each picture. [npc x total_pics]
27 % These pictures are ordered just like in A.
28 %% Runtime Params
29 sub_mean = true;      % True subtracts from the class. False, from the image.
30 make_unit = true;     % Makes all the P matrices have unit columns for projection.
31 plot_all_modes = false;
32 %% Code
33
34 % Echo the input pic
35 figure(1)
36 colormap (gray)
37 imagesc(test_im)
38 title('Test Image')
39 pause;
40
41
42 % Load from db:
43 % From class vec:
44 class_vec = db{7};
45 total_pics = length(class_vec);
46 tmp_v = unique(class_vec);
47 num_classes = length(tmp_v); % How many different classes.
48 eig_weights = db{5};
49 curve_weights = db{6};
50
51 % From the SVD:
52 P = db{3};
53 P_cr = db{4};
54
55 % From our image matrices
56 A = db{1};
57 A_cr = db{2};
58 m_A = db{8};
59 m_A_cr = db{9};
60 options = db{10};
61
62 %% EigenFaces Test
63
64 % Detection/runtime parameters
65 n = options.n;
66 [np,npc] = size(P);
67 threshFace = 45 * n; % The average magnitude of wt vectors are ~1000 empirically for 45.
68 found = false;
69 found_index = 0;
70
71 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
72 % Second thing to test
73
74 % Reshape the test image so it is a vector
75 test_v = reshape(test_im, (n*n), 1);
76
77 % Subtract the mean
78 if (sub_mean) % True subtracts from the set A. False, from the image.
79 test_v = test_v - m_A;
80 else
81 test_v = test_v - mean(test_v);
82 end
83
84 if (make_unit)
85 [nrows_P, ncols_P] = size(P);

```

```

85     for j = 1: ncols_P
86         P(:,j) = P(:,j) ./ sqrt( sum( P(:,j).^2 ));
87     end
88 end
89 % Project the test image onto all of the principal modes/eigenFaces (eigenFaces are the rows of P')
90 proj_onto_eigs = P' * test_v;
91
92
93 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
94 %%%%%%%%%% CLASSIFICATION
95 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
96 % I am determining a nearest neighbor rule for classification.
97 % I project my faces into the space of the Principal Components.
98 % For each component, I have an associated weight.
99 % That leaves me with a vector that stores these weights in each
100 % row. The distance between two of these vectors tells me how
101 % similar two images are.
102
103 % Calculate the distance (Euclidean) from the projected face to the
104 % other faces. This is an [npc x 1] dimensional vector for each.
105 distances = zeros(total_pics,1);
106 % The distances are calculated correctly, eig_weights stored correct.
107 for i = 1:total_pics
108     distances(i) = sqrt( sum( (proj_onto_eigs(:) - eig_weights(:,i)).^2 ));
109 end
110
111
112 %%% INVESTIGATE USING A DIFFERENT METHOD OF CLASSIFICATION.
113 % K-means. Voronoi.
114
115 % Nearest Neighbor Classification
116 % Find and order the closest face.
117 % The 'closest' face is the one with the most similar projection.
118
119 % Find the minimum distance
120 min_dist = bitmax;
121 min_index = -1;
122 for i = 1:total_pics
123     if (distances(i) < min_dist)
124         min_index = i;
125         min_dist = distances(i); % THIS IS ESSENTIAL
126     end
127 end
128
129 % At this point, min_index should be where the minimum is.
130 % Examine the closest match and make a decision
131 %if (distance(min_index) < threshFace )
132 if ( min_dist < threshFace )
133     found = true;
134     found_index = min_index;
135     found_class = class_vec(found_index);
136     disp('Found a face using EigenFaces')
137 else
138     disp('Image not recognized.')
139 end
140
141 % THIS CODE ALL WORKS
142
143 % Display All the Images in the Found Class
144 if found
145     % Find the other images in the found class
146     % I assume that there are only 3 images per class.
147     [class_indices] = find(class_vec == found_class);
148     %disp('This is class_indices')
149     %class_indices

```

```

150
151     % Retrieve these images
152     % Make a plot with all the pictures from the individual (class)
153     subplot(2,2,1)
154         colormap (gray)
155         imagesc(test_im)
156         title('Identity Verified by EigenFaces')
157
158     % Make a matrix for the found images
159     found_mat = zeros(np,3);
160     for i = 1:3
161         found_mat(:,i) = A(:,class_indices(i));
162     end
163
164     for i = 1:3
165         subplot(2,2,i+1)
166             tmp_im = reshape( found_mat(:,i), n, n);
167             colormap (gray)
168             imagesc( tmp_im );
169             title('Matching Face')
170     end
171 end
172
173 %%%%%%%%%%% DIAGNOSTIC %%%%%%%%%%%
174
175 if (plot_all_modes)
176
177     % Plot the eigenfaces of the current on to make sure the
178     % decomposition is ok.
179
180     figure(2)
181         subplot(2,2,1)
182             colormap (gray)
183             imagesc(test_im)
184             title('Test_im')
185         for i=1:3
186             % Reconstruct the image using more Eigenfaces
187             eig_img = proj_onto_eigs(i) * P(:,i);
188             subplot(2,2,i+1)
189                 colormap (gray)
190                 imagesc(reshape(eig_img,n,n))
191                 title(['Projections onto mode #', num2str(i)])
192         end
193
194     %%%%%%%%%%% DIAGNOSTIC 2 %%%%%%%%%%%
195
196     % Make a summed image using 2,4,6, and 8 principal components
197     figure(3)\begin{figure}[h]
198         \begin{center}
199         \includegraphics[scale=0.75]{ringing.png}
200         \caption{\scriptsize The filtered bread.}
201         \end{center}
202     \end{figure}
203     eig_test = zeros(np,1);
204     for i = 1:8
205         eig_test = eig_test + ( proj_onto_eigs(i) * P(:,i) ) ;
206         if (mod(i,2)==0)
207             subplot(2,2,i/2)
208                 colormap (gray)
209                 imagesc(reshape(eig_test,n,n))
210                 title(['Using ', num2str(i), ' Eig PCs'])
211         end
212     end
213
214     figure(4)

```

```

215     eig_test = zeros(np,1);
216     for i = 1:16
217         eig_test = eig_test + ( proj_onto_eigs(i) * P(:,i) );
218         if (mod(i,4)==0)
219             subplot(2,2,i/4)
220                 colormap (gray)
221                 imagesc(reshape(eig_test,n,n))
222                 title(['Using ', num2str(i), ' Eig PCs'])
223         end
224     end
225 end
226
227
228     %% CURVELET TEST
229     % Default parameters
230     threshCurveFace = 1000;
231     found_cr = false;
232     found_index_cr = 0;
233     options = db{10};
234
235 % Assumptions
236 %n = options.n This is passed in and defined above.
237
238 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% FOR PAPER
239 %%% Determine a good thresh level for curvelets
240
241 % I am going to have to curvelet decompose my testimage, test_im.
242
243 % Take the curvelet transform. Use the same thresh/options for your
244 % original stuff.
245 test_cr = perform_curvelet_transform(test_im, options);
246 test_cr_th = perform_thresholding(test_cr, options.thresh, 'hard');
247 test_cr_inv = perform_curvelet_transform(test_cr_th, options);
248
249 test_v = reshape(test_cr_inv, (n*n), 1);
250 % MEAN SUBTRACTION
251 % Subtract the mean
252 if (sub_mean) % True subtracts from the set A. False, from the image.
253     test_v = test_v - m_A_cr;
254 else
255     test_v = test_v - mean(test_v);
256 end
257
258 if (make_unit)
259     [nrows_P_cr, ncols_P_cr] = size(P_cr);
260     for j = 1: ncols_P_cr
261         P_cr(:,j) = P_cr(:,j) ./ sqrt( sum( P_cr(:,j).^2 ));
262     end
263 end
264 proj_onto_curves = P_cr' * test_v;
265
266 % Calculate distances between projections
267 distances_cr = zeros(total_pics,1);
268 for i = 1:total_pics
269     distances_cr(i) = sqrt( sum( proj_onto_curves(:) - curve_weights(:,i)).^2 );
270 end
271
272 % Nearest Neighbor Classification
273 % Find and order the closest faces, based on distance
274 % Find the minimum distance
275 min_dist_cr = bitmax;
276 min_index_cr = -1;
277 for i = 1:total_pics
278     if (distances_cr(i) < min_dist_cr)
279         min_index_cr = i;

```

```

280         min_dist_cr = distances_cr(i); % Update the minimum
281     end
282 end
283
284 % Examine the closest point and make a decision
285 % I have the distance and class data from above.
286     if (min_dist_cr < threshCurveFace ) % Gives the index to the closest one.
287         % Store the index of the face. And then make the plot with
288         % all of them.
289         found_cr = true;
290         found_index_cr = min_index_cr;
291         found_class_cr = class_vec(found_index_cr);
292
293         disp('Found a match using CurveFaces')
294
295     else
296         % Not a face.
297         disp('Image not recognized.')
298     end
299
300
301 % Display this for the found
302 if found_cr
303     % Find the other images in the found class
304     [class_indices_cr] = find(class_vec == found_class_cr);
305
306     %disp('This is class_indices')
307     %class_indices_cr
308
309     % Retrieve these images
310     % They are already stored in A. So it would be redundant to
311     % take them out and re-store them.
312 % Make a plot with all the pictures from the individual (class)
313 figure(5)
314 subplot(2,2,1)
315     colormap (gray)
316     imagesc(test_im)
317     title('Identity Verified by Curvelets')
318 for i = 1:3
319     subplot(2,2,i+1)
320     colormap (gray)
321     imagesc(reshape( A(:,class_indices_cr(i)), n,n) )
322     title('Matching Face')
323 end
324 end
325
326 figure(6)
327 subplot(2,2,1)
328     colormap (gray)
329     imagesc(test_im)
330     title('Projections onto curvefaces.')
331 for i=1:3
332 % Reconstruct the image using more Curvefaces
333     curve_img = proj_onto_curves(i) * P_cr(:,i);
334     subplot(2,2,i+1)
335         colormap (gray)
336         imagesc(reshape(curve_img,n,n))
337         title(['Projection onto CurveFace # ', num2str(i)])
338     end
339
340 %% THIS SHOULD ALL BE ONTO P_cr
341
342 % Make a summed image using 2,4,6, and 8 principal components
343 figure(7)
344 curve_test = zeros(np,1);

```

```

345     for i = 1:8
346         curve_test = curve_test + ( proj_onto_curves(i) * P_cr(:,i) ) ;
347         if (mod(i,2)==0)
348             subplot(2,2,i/2)
349                 colormap (gray)
350                 imagesc(reshape(curve_test,n,n))
351                 title(['Sum of first ', num2str(i), ' CurveFaces'])
352         end
353     end
354
355     figure(8)
356     curve_test = zeros(np,1);
357     for i = 1:16
358         curve_test = curve_test + ( proj_onto_curves(i) * P_cr(:,i) ) ;
359         if (mod(i,4)==0)
360             subplot(2,2,i/4)
361                 colormap (gray)
362                 imagesc(reshape(curve_test,n,n))
363                 title(['Using ', num2str(i), ' Curve PCs'])
364         end
365     end
366
367     %% FisherFace Test
368     test_v = test_im;
369     fisher_weights = db{11};           % - These are the FisherWeights, w_f
370     P_f            = db{12};           % - These are the FisherFaces
371     fisherThresh   = 1000;            % - This is also arbitrary
372
373
374     test_v = reshape(test_v,np,1);
375     if (sub_mean) % True subtracts from the set A. False, from the image.
376         test_v = test_v - m_A;
377     else
378         test_v = test_v - mean(test_v);
379     end
380
381     % I know Fisher's are already returned as unit.
382     if (make_unit)
383         [nrows_P_f, ncols_P_f] = size(P_f);
384         for j = 1:ncols_P_f
385             P_f(:,j) = P_f(:,j) ./ sqrt( sum( P_f(:,j).^2 ));
386         end
387     end
388     proj_onto_fisher = P_f' * test_v;
389
390     % Calculate the distances
391     distances_f = zeros(total_pics,1);
392     for i = 1:total_pics
393         distances_f(i) = sqrt( sum( proj_onto_fisher(:) - fisher_weights(:,i)).^2 );
394     end
395
396     % Nearest Neighbor Classification
397     % Find and order the closest faces, based on distance
398     % Find the minimum distance
399     min_dist_f = bitmax;
400     min_index_f = -1;
401     for i = 1:total_pics
402         if (distances_f(i) < min_dist_f)
403             min_index_f = i;
404             min_dist_f = distances_f(i);
405         end
406     end
407
408     % Examine the closest point and make a decision
409     %for i = 1:2

```

```

410         % I have the distance and class data from above.
411         if (min_dist_f < fisherThresh ) % Gives the index to the closest one.
412             % Store the index of the face. And then make the plot with
413             % all of them.
414             found_f = true;
415             found_index_f = min_index_f;
416             found_class_f = class_vec(found_index_f);
417             disp('Found a match using FisherFaces')
418         else
419             % Not a face.
420             disp('Image not recognized.')
421         end
422
423         if found_f
424             % Find the other images in the found class
425             [class_indices_f] = find(class_vec == found_class_f);
426             figure(9)
427             subplot(2,2,1)
428                 colormap (gray)
429                 imagesc(test_im)
430                 title('Identity Verified by FisherFaces')
431             for i = 1:3
432                 subplot(2,2,i+1)
433                     colormap (gray)
434                     imagesc(reshape( A(:,class_indices_f(i)), n,n) )
435                     title('Matching Face')
436             end
437         end
438
439         %%% FISHERFACE DIAGNOSTIC
440         % Make a summed image using 2,4,6, and 8 FisherFaces
441         figure(10)
442         fisher_test = zeros(np,1);
443         for i = 1:8
444             fisher_test = fisher_test + ( proj_onto_fisher(i) * P_f(:,i) );
445             if (mod(i,2)==0)
446                 subplot(2,2,i/2)
447                     colormap (gray)
448                     imagesc(reshape(fisher_test,n,n))
449                     title(['Sum of first ', num2str(i), ' FisherFaces'])
450             end
451         end
452     end
453 end

```

test_presentation.m

These are the tests that I showed for my presentation.

```

1
2 % REDO THE TEST AUTOMATION
3
4
5 % Clear Everything/Start Fresh
6 close all;
7 clear all;
8 clc;
9
10 disp('Beginning TEST')
11 disp('*****')
12
13
14
15 % The outer level is important for DB creation

```

```

16 test45 = true;
17     test1      = true;
18     test2      = true;
19     test3      = true;
20     test4      = true;
21     test5      = true;
22     test6      = true;
23 test60 = false;
24     test7      = false;
25     test8      = false;
26     test9      = false;
27     test10     = false;
28
29
30
31 if test45
32     % Generate the database for n=45
33     % Params:
34     hist_eq     = true; % Perform a histogram equalization on each of the images.
35     check       = true; % Look at images as they are uploaded
36     n           = 45 ;
37     %Perform tests 1-7
38     % Create the database
39     Bf = gen_database_fisher(n); % This runs default with n = 45
40
41     if (test1)
42         %%%%%%%%%%% TEST 1 %%%%%%%%%%%
43         %% Exact Match in the Database %%
44         %%%%%%%%%%% WORKS FOR EIGENFACES %%%%%%%%%%% CLASS 7
45         %%%%%%%%%%% FAILS FOR CURVEFACES %%%%%%%%%%% CLASS 8
46         %%%%%%%%%%% WORKS FOR FISHERFACES %%%%%%%%%%% CLASS 7
47
48     close all;
49     disp('Beginning test1')
50     % Upload the test image. Process it.
51     % Test7
52     % Out of all images, I expect this one to be classified the
53     % easiest.
54     I_tmp = imread('test7.jpg');
55     if (n == 45)
56         % Resize the image
57         I_tmp2 = imresize(I_tmp, [60,45]);
58         % Conversion to Grayscale
59         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 7 45 44 ] ) ); % 60 x 60 and 45 x 45 images.
60     elseif (n == 60)
61         I_tmp2 = imresize(I_tmp, [80,60]);
62         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 10 60 59 ] ) );
63     elseif (n == 180)
64         I_tmp2 = imresize(I_tmp, [240 180]);
65         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 30 180 179 ] ) ); % [xmin ymin width height]
66     end
67     if hist_eq
68         I_tmp3 = histeq(I_tmp3);
69     end
70     test7 = double(I_tmp3);
71     if check
72         figure()
73         colormap (gray)
74         imagesc(test7)
75         title('Test Image: test7.jpg')
76         drawnow;
77         pause;
78     end
79     % Test the exact image
80     im_test_cr(test7, Bf)

```

```

81         pause;
82     end
83
84
85     if (test2)
86         %%%%%%%%%%% TEST 2 %%%%%%%%%%%
87         %% Very close to an image in the database
88         %%%%%%%%%%% WORKS FOR EIGENFACES %%%%%%%%%%% CLASS 7
89         %%%%%%%%%%% FAILS FOR CURVEFACES %%%%%%%%%%% CLASS 9
90         %%%%%%%%%%% FAILS FOR FISHERFACES %%%%%%%%%%% CLASS 10
91
92     close all;
93     disp('Beginning test2')
94     I_tmp = imread('test7_other.jpg');
95     if (n == 45)
96         % Resize the image
97         I_tmp2 = imresize(I_tmp, [60,45]);
98         % Conversion to Grayscale
99         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 7 45 44 ] ) ); % 60 x 60 and 45 x 45 images.
100    elseif (n == 60)
101        I_tmp2 = imresize(I_tmp, [80,60]);
102        I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 10 60 59 ] ) );
103    elseif (n == 180)
104        I_tmp2 = imresize(I_tmp, [240 180]);
105        I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 30 180 179 ] ) ); % [xmin ymin width height]
106    end
107
108    if hist_eq
109        I_tmp3 = histeq(I_tmp3);
110    end
111
112    test7_other = double(I_tmp3);
113    if check
114        figure()
115        colormap (gray)
116        imagesc(test7_other)
117        title('Test Image: test7_other.jpg')
118        drawnow;
119        pause;
120    end
121
122    close all;
123    im_test_cr(test7_other, Bf)
124    % The found_class should be class 7
125    pause;
126 end
127
128 if (test3)
129     %%%%%%%%%%% TEST 3 %%%%%%%%%%%
130     %%% Confirm EigenFace Result %%%
131     %%%%%%%%%%%
132
133     %%%%%%%%%%% WORKS FOR EIGENFACES %%%%%%%%%%% CLASS 2
134     %%%%%%%%%%% WORKS FOR CURVEFACES %%%%%%%%%%% CLASS 2
135     %%%%%%%%%%% FAILS FOR FISHERFACES %%%%%%%%%%% CLASS 7
136
137 close all;
138 disp('Beginning test3')
139 I_tmp = imread('test2_1.jpg');
140 if (n == 45)
141     % Resize the image
142     I_tmp2 = imresize(I_tmp, [60,45]);
143     % Conversion to Grayscale
144     I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 7 45 44 ] ) ); % 60 x 60 and 45 x 45 images.
145 elseif (n == 60)

```

```

146         I_tmp2 = imresize(I_tmp, [80,60]);
147         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 10 60 59 ] ) );
148     elseif (n == 180)
149         I_tmp2 = imresize(I_tmp, [240 180]);
150         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 30 180 179 ] ) ); % [xmin ymin width height]
151     end
152
153     if hist_eq
154         I_tmp3 = histeq(I_tmp3);
155     end
156
157     test2_1 = double(I_tmp3);
158     if check
159         figure()
160         colormap (gray)
161         imagesc(test2_1)
162         title('Test Image: test2_1.jpg')
163         drawnow;
164         pause;
165     end
166
167     close all;
168     im_test_cr(test2_1, Bf)
169     % The found_class should be class 2
170     pause;
171 end
172
173 if (test4)
174     %%%%%%%%%%% TEST 4 %%%%%%%%%%%
175     %%% TRIPLE CONFIRM EIGENFACE RESULT
176     %%%%%%%%%%%
177     %%% FAILS FOR EIGENFACES %%% X
178     %%% FAILS FOR CURVEFACES %%% CLASS 2
179     %%% FAILS FOR FISHERFACES %%% CLASS 2
180
181     %%% This one would not surprise me if not detected. The bangs on the
182     %%% man's hair are a little different from all of the training pictures
183
184     close all;
185     disp('Beginning test4')
186     I_tmp = imread('test9.jpg');
187     if (n == 45)
188         % Resize the image
189         I_tmp2 = imresize(I_tmp, [60,45]);
190         % Conversion to Grayscale
191         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 7 45 44 ] ) ); % 60 x 60 and 45 x 45 images.
192     elseif (n == 60)
193         I_tmp2 = imresize(I_tmp, [80,60]);
194         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 10 60 59 ] ) );
195     elseif (n == 180)
196         I_tmp2 = imresize(I_tmp, [240 180]);
197         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 30 180 179 ] ) ); % [xmin ymin width height]
198     end
199
200     if hist_eq
201         I_tmp3 = histeq(I_tmp3);
202     end
203
204     test9 = double(I_tmp3);
205     if check
206         figure()
207         colormap (gray)
208         imagesc(test9)
209         title('Test Image: test9.jpg')
210         drawnow;

```

```

211         pause;
212     end
213
214     close all;
215     im_test_cr(test9, Bf)
216     % The found_class should be class 2
217     pause
218 end
219
220 if (test5)
221     %%%%%%%%%%% TEST 5 %%%%%%%%%%%
222     %% TRY TO RECONFIRM EIGENFACE RESULT
223     %%%%%%%%%%%
224     %%%%%%%%%%% FAILS FOR EIGENFACES %%%%%%%%%%% X
225     %%%%%%%%%%% FAILS FOR CURVEFACES %%%%%%%%%%% CLASS 1
226     %%%%%%%%%%% FAILS FOR FISHERFACES %%%%%%%%%%% CLASS 4
227
228 close all;
229     disp('Beginning test5')
230
231 % This image is slightly translated relative to all of the training
232 % images. I expect that that was the cause of the failure.
233
234     I_tmp = imread('test15.jpg');
235     if (n == 45)
236         % Resize the image
237         I_tmp2 = imresize(I_tmp, [60,45]);
238         % Conversion to Grayscale
239         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 7 45 44 ] ) ); % 60 x 60 and 45 x 45 images.
240     elseif (n == 60)
241         I_tmp2 = imresize(I_tmp, [80,60]);
242         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 10 60 59 ] ) );
243     elseif (n == 180)
244         I_tmp2 = imresize(I_tmp, [240 180]);
245         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 30 180 179 ] ) ); % [xmin ymin width height]
246     end
247
248     if hist_eq
249         I_tmp3 = histeq(I_tmp3);
250     end
251
252     test15 = double(I_tmp3);
253     if check
254         figure()
255         colormap (gray)
256         imagesc(test15)
257         title('Test Image: test15.jpg')
258         drawnow;
259         pause;
260     end
261
262     close all;
263     im_test_cr(test15, Bf)
264     % The found_class should be class 15
265     pause;
266
267 end
268
269
270 if (test6)
271     %%%%%%%%%%% TEST 6 %%%%%%%%%%%
272     %% TRY TO RECONFIRM EIGENFACE RESULT
273     %%%%%%%%%%%
274     %%%%%%%%%%% WORKS FOR EIGENFACES %%%%%%%%%%% CLASS 3
275     %%%%%%%%%%% WORKS FOR CURVEFACES %%%%%%%%%%% CLASS 3

```

```

276 %%%%%%%%%% FAILS FOR FISHERFACES %%%%%%%%%%%%%% CLASS 9
277
278 % This image is close to two of the training faces but has glasses on.
279
280 close all;
281 disp('Beginning test6')
282 I_tmp = imread('glasses.jpg');
283 if (n == 45)
284     % Resize the image
285     I_tmp2 = imresize(I_tmp, [60,45]);
286     % Conversion to Grayscale
287     I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 7 45 44 ] ) ); % 60 x 60 and 45 x 45 images.
288 elseif (n == 60)
289     I_tmp2 = imresize(I_tmp, [80,60]);
290     I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 10 60 59 ] ) );
291 elseif (n == 180)
292     I_tmp2 = imresize(I_tmp, [240 180]);
293     I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 30 180 179 ] ) ); % [xmin ymin width height]
294 end
295
296 if hist_eq
297     I_tmp3 = histeq(I_tmp3);
298 end
299
300 glasses = double(I_tmp3);
301 if check
302     figure()
303     colormap (gray)
304     imagesc(glasses)
305     title('Test Image: glasses.jpg')
306     drawnow;
307     pause;
308 end
309
310 close all;
311 im_test_cr(glasses, Bf)
312 % The found_class should be class 3
313 pause;
314 end
315
316
317 %% CONCLUSION
318
319 % At this point, I believe that I can conclusively say that my
320 % EigenFaces and CurveFaces method have limited success in identifying
321 % subjects.
322
323 % My FisherFaces is failing so I am going to check my classifier
324 % carefully (It seems to be calculating the Euclidean Distance perfectly)
325 % I will investigate the literature, because I think it used a
326 % different classifier than the EigenFaces.
327
328 % I still have to experiment with different sizes and thresholds for
329 % the CurveFaces
330
331
332
333 end
334 if test60
335     % Generate the database for n=60
336     % Params:
337     hist_eq = true;
338     check = true; % Look at images
339     n = 60;
340     % Create the database

```

```

341         Bf = gen_database_fisher(n); % This runs default with n = 45
342
343     % Tests 8-10
344     if (test8)
345         %%%%%%%%%%% TEST 8 %%%%%%%%%%%
346         %% Look at the original exact match but with the change in size
347         %%%%%%%%%%%
348         %%%%%%%%%% FAILS FOR EIGENFACES %%%%%%%%%% X
349         %%%%%%%%%% FAILS FOR CURVEFACES %%%%%%%%%% CLASS 10
350         %%%%%%%%%% WORKS FOR FISHERFACES %%%%%%%%%% CLASS 7
351
352     % This image is close to two of the training faces but has glasses on.
353
354     close all;
355     disp('Beginning test8')
356     I_tmp = imread('test7.jpg');
357     if (n == 45)
358         % Resize the image
359         I_tmp2 = imresize(I_tmp, [60,45]);
360         % Conversion to Grayscale
361         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 7 45 44 ] ) );
362     elseif (n == 60)
363         I_tmp2 = imresize(I_tmp, [80,60]);
364         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 10 60 59 ] ) );
365     elseif (n == 180)
366         I_tmp2 = imresize(I_tmp, [240 180]);
367         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 30 180 179 ] ) ); % [xmin ymin width height]
368     end
369
370     if hist_eq
371         I_tmp3 = histeq(I_tmp3);
372     end
373
374     test7 = double(I_tmp3);
375     if check
376         figure()
377         colormap (gray)
378         imagesc(test7)
379         title('Test Image: test7.jpg')
380         drawnow;
381         pause;
382     end
383
384     close all;
385     im_test_cr(test7, Bf)
386     % The found_class should be class 3
387     pause;
388 end
389
390 % CONCLUSION:
391 % I have big a scaling issue! I thought EigenFaces was working so I
392 % will check to see if I have any magic numbers within the code
393 % that tie it to n = 45. My results were identical so see if I
394 % inadvertently built something in that is causing the failure.
395
396 if (test9)
397     %%%%%%%%%%% TEST 9 %%%%%%%%%%%
398     %% Look at the original close match but with the change in size
399     %% (test7_other)
400     %%%%%%%%%%%
401     %%%%%%%%%% FAILS FOR EIGENFACES %%%%%%%%%% X
402     %%%%%%%%%% FAILS FOR CURVEFACES %%%%%%%%%% CLASS 10
403     %%%%%%%%%% WORKS FOR FISHERFACES %%%%%%%%%% CLASS 7
404
405     % This image is close to two of the training faces but has glasses on.

```

```

406
407     close all;
408     disp('Beginning test9')
409     I_tmp = imread('test7_other.jpg');
410     if (n == 45)
411         % Resize the image
412         I_tmp2 = imresize(I_tmp, [60,45]);
413         % Conversion to Grayscale
414         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 7 45 44 ] ) );
415     elseif (n == 60)
416         I_tmp2 = imresize(I_tmp, [80,60]);
417         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 10 60 59 ] ) );
418     elseif (n == 180)
419         I_tmp2 = imresize(I_tmp, [240 180]);
420         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 30 180 179 ] ) ); % [xmin ymin width height]
421     end
422
423     if hist_eq
424         I_tmp3 = histeq(I_tmp3);
425     end
426
427     test7_other = double(I_tmp3);
428     if check
429         figure()
430         colormap (gray)
431         imagesc(test7_other)
432         title('Test Image: test7_other.jpg')
433         drawnow;
434         pause;
435     end
436
437     close all;
438     im_test_cr(test7_other, Bf)
439     % The found_class should be class 3
440     pause;
441 end
442 % Conclusion:
443     % I have a serious issue with EigenFaces when n = 60
444     % CurveFaces is always going to class = 7
445     % FisherFaces is working without a hitch.
446
447
448 if (test10)
449     %%%%%%%%%%% TEST 10 %%%%%%%%%%%
450     %% Look at test2 which worked for EigenFaces at n = 45
451     %%%%%%%%%%%
452     %%%%%%%%%%% FAILS FOR EIGENFACES %%%%%%%%%%% X
453     %%%%%%%%%%% FAILS FOR CURVEFACES %%%%%%%%%%% CLASS 10
454     %%%%%%%%%%% FAILS FOR FISHERFACES %%%%%%%%%%% CLASS 8
455     close all;
456     disp('Beginning test10')
457     I_tmp = imread('test2.jpg');
458     if (n == 45)
459         % Resize the image
460         I_tmp2 = imresize(I_tmp, [60,45]);
461         % Conversion to Grayscale
462         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 7 45 44 ] ) );
463     elseif (n == 60)
464         I_tmp2 = imresize(I_tmp, [80,60]);
465         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 10 60 59 ] ) );
466     elseif (n == 180)
467         I_tmp2 = imresize(I_tmp, [240 180]);
468         I_tmp3 = rgb2gray( imcrop(I_tmp2,[1 30 180 179 ] ) ); % [xmin ymin width height]
469     end
470

```

```
471         if hist_eq
472             I_tmp3 = histeq(I_tmp3);
473         end
474
475         test2 = double(I_tmp3);
476         if check
477             figure()
478             colormap (gray)
479             imagesc(test2)
480             title('Test Image: test2.jpg')
481             drawnow;
482             pause;
483         end
484
485         close all;
486         im_test_cr(test2, Bf)
487         % The found_class should be class 3
488         pause;
489     end
490
491
492
493     end
```

References

References

- [Lem04] Donoho Ying Candes Lemanet. The md5 message-digest algorithm. <http://www.curvelet.org/>, 2004.
- [MW07] Tanaya Mandal Angshul Majumdar and Q.M. Jonathan Wu. Face Recognition by Curvelet Based Feature Extraction. *IEE*, 48:806–817, 2007.
- [Pey06] Gabriel Peyre. A numerical tour of signal processing. <http://www.ceremade.dauphine.fr/~peyre/numerical-tour/>, 2006.