ECE 420: Embedded DSP Laboratory
Final Project Report
Near-Real-Time Facial Recognition on an Android Tablet Using Fisherfaces

Jonathan Wang & Kristian Lauszus
NetID: JYWANG6 & LAUSZUS2

December 14, 2016

Abstract

In the previous assigned lab project, our group explored facial detection and recognition using eigenfaces, as described in the paper *Eigenfaces for recognition*[3]. The eigenfaces approach of facial recognition has a high success rate in a well-defined static environment. However, we find that its success rate deteriorates with variations in lighting and facial expression. With that in mind, we wish to explore a variant of the Eigenfaces approach: the Fisherfaces approach.

1 Motivation

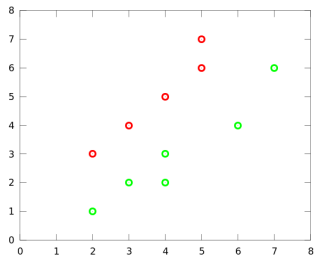
We derived our idea from the research paper *Eigenfaces vs. Fisherfaces: Recognition Using Class Specified Linear Projection* by Belhumeur et al [2]. The problem statement is as follows: "Given a set of face images labeled with the person's identity (*the learning set*) and an unlabeled set of face images from the same group of people (*the test set*), identify each person in the test images. Our goal is to implement the algorithm described in the paper as an Android application.

2 Research Paper Review

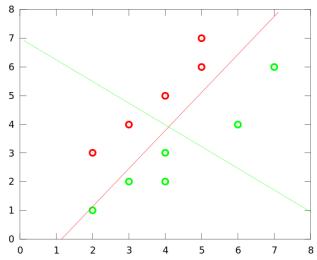
2.1 Eigenfaces Method for Facial Recognition

The Eigenfaces approach is derived in a way that uses Principal Component Analysis (PCA) to maximize variation between the training image sets of faces. However, through PCA we not only maximize scatter between different classes, but the scatter within the same class. As stated by Moses et al., "the variation between the images of the same face due to illumination and viewing direction are almost always larger than the image variation due to change in face identity" [1]. Therefore, while PCA projections may be helpful for facial reconstruction from a low dimensional basis, they may not be as useful when the end goal is facial discrimination. While a popular method of overcoming this disadvantage is to discard the three most significant principal components, it is likely that this leads to a loss of information that is helpful for facial discrimination.

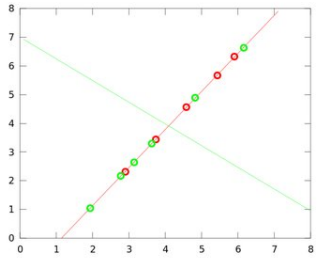
We can visualize this concept with a small 2D example [4], as seen in Figure 1. Figure 1a shows a scatter plot of a set of two-dimensional data. We can imagine each data point as a face image with two features that varies between faces. Furthermore, we label each color as a distinct face (or class). Since the original data is 2D, we can find a maximum of two principal components, shown in Figure 1b. The first principal component is one that maximizes the variance between every data point in the original set, and every subsequent component is found by subtracting all the previous components from the original set, then finding the component that maximizes the variance of the resulting set. Intuitively, if we projected the original data to the first principal component (red), we should expect a resulting plot that can be easier to classify. However, Figure 1c depicts that this is not the case. While this lower dimension projection did indeed maximize the variance between each face image, the two faces are no longer linearly separable. Projecting on the second component Figure 1d shows a much better representation for classification.



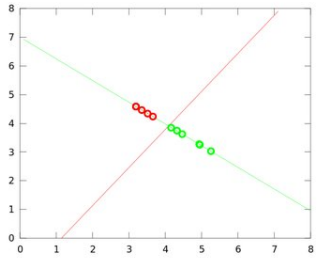
(a) Simple 2D Data



(b) First two Principal Components



(c) Projection onto the first principal component



(d) Projection onto the second principal component

Figure 1: Simple PCA Example[4]

In a perfect world where each face image only contains information about the face, PCA may be a perfectly acceptable method for facial recognition. Unfortunately, in real life there's no algorithmic way to pick the correct number of "unwanted" principal components to discard, especially since each datapoint is essentially its own "class" (PCA is unlabeled), therein lies the limitations with this method.

2.2 Fisherfaces Method for Facial Recognition

Since Eigenfaces' maximization of both in-class and between-class scatter is undesirable, we wish to implement a method that maximizes the scatter between classes, while minimizing it within a class. The Fisherfaces method mentioned in Belhumeur et al.[2] is a prominent approach for this purpose.

The Fisherfaces approach is based on linear subspaces. If we can train a specific face using different images of the same face under arbitrary lighting conditions, we can extract the underlying static face, and represent it as a 3D linear subspace, indifferent to lighting directions, in a Lambertian surface. This highlights one of the most significant differences between the Eigenfaces method and the Fisherfaces method, specifically that the latter uses a labeled training set in order to gain a more accurate classification.

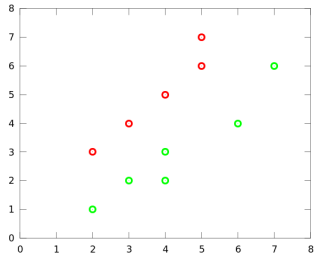
Since the training set is labeled, we can use class specific linear methods to reduce the dimensionality of the feature space. In Fisherspaces, a specific method of LDA (linear discriminant analysis) - Fisher's Linear Discriminant (FLD) is one of said *class specific method*, in that it reforms the scatter in order to make it more viable for classification.

We depict this method again using the same simple dataset from before. Due to the labeled nature FLD, we see that the lower dimension component found using FLD provides a much better discrimination (Figure 2b).

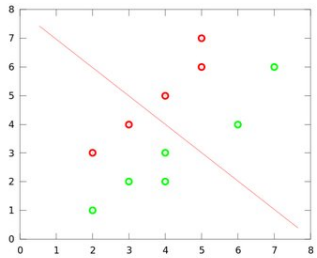
It's important to note that these are simplified, yet non-trivial examples. Here we are reducing a 2D space to a 1D space, but in the case of face recognition, the original data space could be as large as $N^2 * c$, where N^2 is the number of pixels, and c is the number of classes. Using the FLD method we would be able to reduce that down to simply $c - 1$ dimensions. This indicates a huge reduction in dimensionality, without which classification between different faces would be practically impossible.

2.3 Algorithmic Description

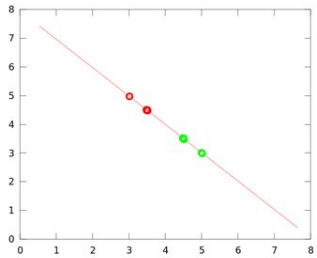
The Fisherfaces algorithm is as follows:



(a) Simple 2D Data



(b) LDA Principal Component



(c) Projection onto the LDA Principal Component

Figure 2: Simple PCA Example[4]

1. Construct the image matrix $\mathbf{X}$ similarly to the eigenfaces method, but assign each image to a class (relating to its original identity) in the corresponding class vector c. There are c of such $\mathbf{X}$ vectors, each with n different samples.

$$\boldsymbol{X} = \{\boldsymbol{X_1}, \boldsymbol{X_2}, ..., \boldsymbol{X_c}\}$$

$$\boldsymbol{X_i} = \{x_1, x_2, ..., x_n\}$$

2. Project $\mathbf{X}$ into the $(N-c)$-dimensional subspace via PCA, with the rotational matrix W_{PCA}. N is the number of samples in $\mathbf{X}$, and c is the number of unique classes/faces.

$$W_{PCA} = \arg\max_W |W^T S_T W|$$

3. Calculate the between-class scatter of the projection,

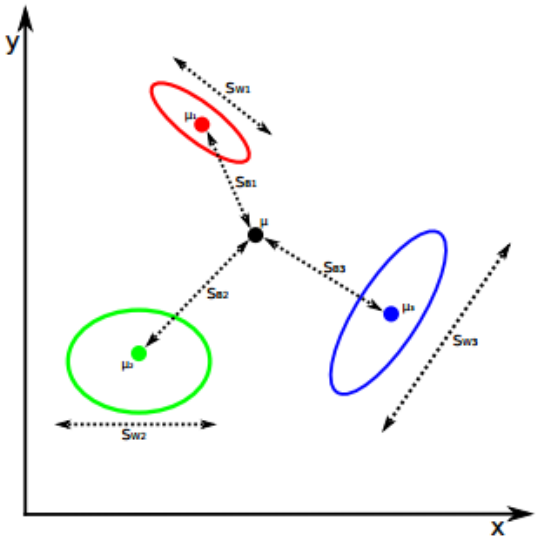


Figure 3: 3 Classes Example of FLD

$$S_B = \sum_{i=1}^{c} N_i(\boldsymbol{\mu}_i - \mu)(\boldsymbol{\mu}_i - \mu)^T$$

and the within-class scatter of the projection,

$$S_W = \sum_{i=1}^{c} \sum_{\boldsymbol{x} \in X_i} (\boldsymbol{x}_k - \boldsymbol{\mu}_i)(\boldsymbol{x}_k - \boldsymbol{\mu}_i)^T$$

Where μ is the total mean:

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i$$

and $\boldsymbol{\mu}_i$ is the mean of each class $i \in \{1, ..., c\}$:

$$\mu_i = \frac{1}{|X_i|} \sum_{x_j \in X_i} x_j$$

4. Apply Fisher's Linear Discriminant and maximize the ratio of the determinant of between-class scatter and within-class scatter. The solution is given by the set of generalized eigenvectors W_{FLD} of S_B and S_W. This gives us $c-1$ non-zero eigenvalues.

$$W_{FLD} = \arg\max_W \frac{|W^T W_{PCA}^T S_B W_{PCA} W|}{|W^T W_{PCA}^T S_W W_{PCA} W|}$$

5. Obtain the Fisherfaces:

$$W = W_{FLD}^T W_{PCA}^T$$

The transformation matrix W can now be used to project samples into the $(c-1)$-dimensional

space. A spacial diagram of the FLD classification can be seen in Figure 3.

Once we have the set of Fisherfaces, we can treat it similarly to the Eigenfaces vector and project new faces into the Fisherfaces face space to perform analysis.

3 C++ Implementation

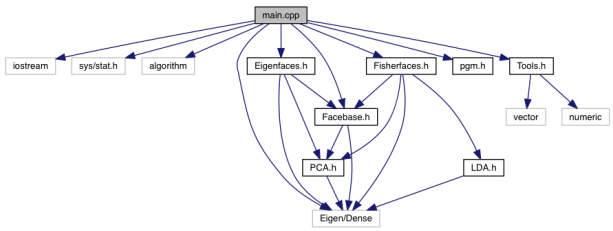
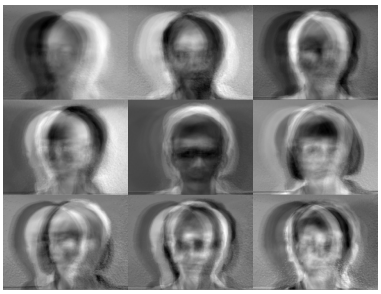


Figure 4: C++ Class Diagram

In our outline in Figure 2.3, we use both PCA and LDA. Since PCA was already completed in the assigned project lab, we took the essential parts of our C++ Eigenfaces implementation, encapsulated PCA as a base class, and Facebase as its derived class. The Facebase class, as the name implies, includes methods that calculate the various distances in the face base, as well as methods that project and reconstruct the face used in the Eigenfaces method. The procedures that train the database according to the two different approaches (Eigenfaces and Fisherfaces) are class-specific, so an Eigenfaces class and a Fisherfaces class are created, both inheriting the Facebase class. To satisfy the LDA requirements, an LDA class is created, which is also inherited by the Fisherfaces class. Figure 4 shows the class structure as described.
The source code of the C++ part of the project can be found at `https://github.com/Lauszus/FaceRecognitionLib`.

4 Testing Results

We confirmed some of the results mentioned in *Eigenfaces vs Fisherfaces...*[2] with our C++ implementation of the two algorithms. We used the Yale Database, which contains 160 frontal face images covering 16 individuals taken under 10 different conditions. Within this database, Fisherfaces perfectly finds all the faces of the test subject as seen in Figure 6, whereas Eigenfaces finds all the subjects under similar lighting with the largeset principal component (Figure 5). This is to be expected by observing the first 9 eigenfaces extracted (Figure 5a). Here, it's obvious that the two largest features are actually just the variation of lighting in the background, whereas the first two fisherfaces extracted (Figure 6a) are, like the rest of the fisherfaces, actual features of the faces.



(a) Eigenfaces

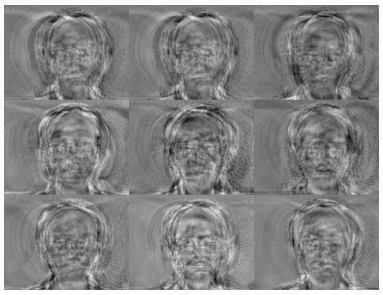
(b) Matches

Figure 5: Eigenfaces & Matches on the Yale Database

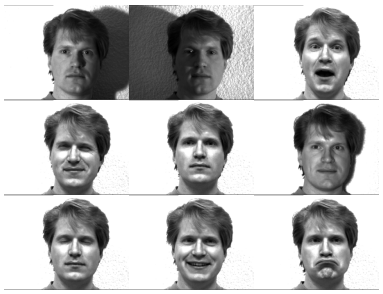
5 Android Integration

The Android application we developed can use either Eigenfaces or Fisherfaces to perform facial recognition on images taken using the camera. The main application is written in Java using the Android SDK. The algorithm choice, along with the two threshold values, can be selected in the navigation menu (seen in Figure 8a). The operation of the app can be seen in Figure 7. After the user choses an algorithm and takes a picture, the desired recognition algorithm runs, and returns one of four possible notifications as a toast:

1. **"Face detected..."**: the image is near the face space and also near a face class
2. **"Unknown face..."**: the image is near the face space but not near any known face class
3. **"False recognition..."**: the image is distant from the face space, but is near a face class
4. **"Image is not a face..."**: the image is distance from both the face space and all known face classes



(a) Eigenfaces



(b) Matches

Figure 6: Fisherfaces & Matches on the Yale Database

An example of this is seen in Figure 8b. In each case, the important distances are given to enable the user to choose better threshold values depending on the background.

Upon exiting the app, the database is saved and reloaded automatically on next launch. This database can be cleared via the "CLEAR TRAINING SET" button in the navigation menu. The taken images are turned into greyscale images, then scaled to a 200x150 resolution via OpenCV before given to the algorithm. This is done to both save storage space and decrease the CPU load on calculation.
The source code to the Android App can be found at: `https://github.com/Lauszus/FaceRecognitionApp`.

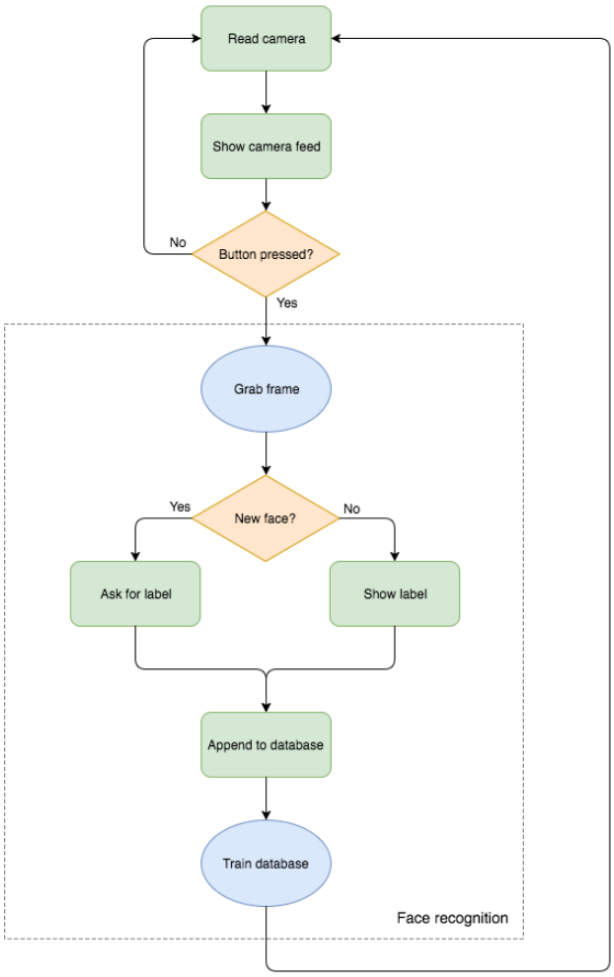


Figure 7: Project block diagram

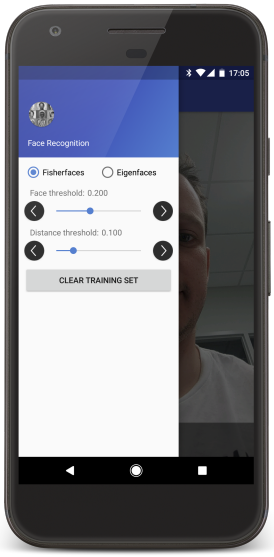
References

[1] Y. Adini, Y. Moses, and S. Ullman. Face recognition: The problem of compensating for changes in illumination direction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):721–732, 07 1997. ISSN 0162-8828. doi: 10.1109/34.598229.

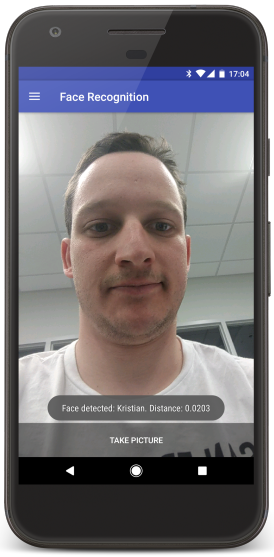
[2] P.N. Belhumeur, J.P. Hespanha, and D.J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):711–720, 07 1997. ISSN 0162-8828. doi: 10.1109/34.598228.

[3] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 01 1991. ISSN 0898-929X. doi: 10.1162/jocn.1991.3.1.71.

[4] Philipp Wegner. Principal component analysis and linear discriminant analysis with gnu octave, 10 2011. URL `http://www.bytefish.de/blog/pca_lda_with_gnu_octave/`.

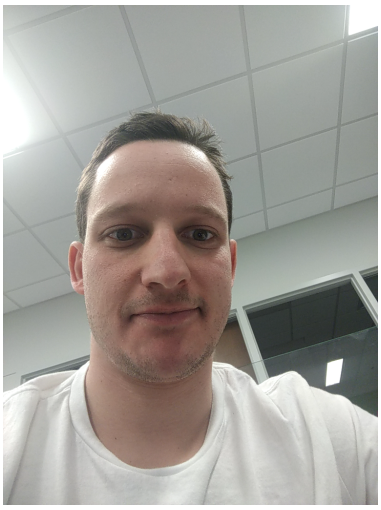


(a) Android App UI

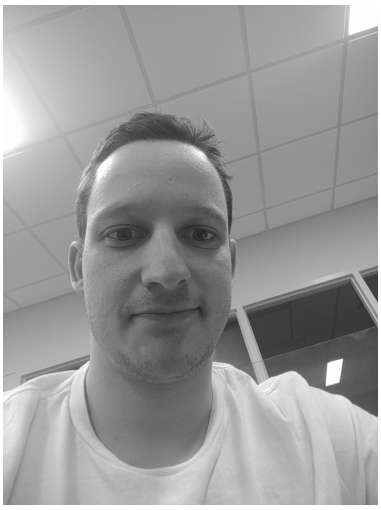


(b) Face Detected on UI

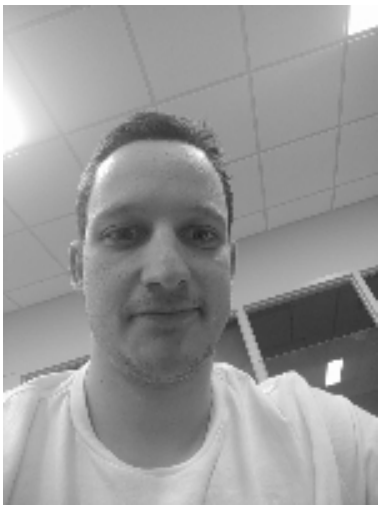
Figure 8: Android UI Demo



(a) Original Image



(b) Greyscale Image



(c) 200x150 Scaled Image

Figure 9: Input Images from Camera