Mobile Face Recognization

CS4670 Final Project

Cooper Bills and Jason Yosinski
{csb88,jy495}@cornell.edu

December 12, 2010

Abstract

We created a mobile based system for detecting faces within a picture and matching these faces to a precompiled database. Face detection is done using a Haar cascade, and face recognization is done using a combination of Fisherfaces and PCA analysis, also known as Eigenfaces. The system shows promising performance and processes images quickly even on a mobile platform.

1 Introduction

Our original proposal was to create a mobile application that will identify people in an photo using pictures from facebook, and then poll their information and status and display it in the photo. This was to create an augmented reality interface on the mobile device. However, we quickly found that to create a quality application with this ability would require more time and resources than we had.

As an alternative, we created a simpler application that utilized a local database with the goal to find and identify faces in an mobile image. This database is created on a desktop computer and copied to the phone before the application is used. Our testing utilized two databases: The Yale Face Database[1] [5] and our own custom database from facebook images.

2 Related Work

Face detection and identification is an active field in computer vision, and many methods have been developed. For face detection, one of the most notable is the "Viola-Jones" method, published by Paul Viola and Michael Jones [9]. Their work is based on AdaBoost[6],

[1]Available here: `http://cvc.yale.edu/projects/yalefaces/yalefaces.html`

improved by cascading classifiers by complexity - improving speed and accuracy. The algorithm this project utilizes for facial detection was developed by Rainer Lienhart and Jochen Maydt at Intel labs.[7] Their algorithm adds rotated Harr-like features to improve accuracy.

Face identification (recognition) methods have existed for decades, and is continually improving. One of the oldest methods that has stood the test of time is Eigenfaces[8], developed by Matthew Turk and Alex Pentland, which utilizes Principal Component Analysis (PCA) for recognition. This method has been expanded using Linear Discriminant Analysis (LDA) to improve accuracy, known as Fisherfaces [4]. Newer facial recognition systems use 3D models[1], and were considered for this project, but were ruled out because the phone has limited computational ability.

3 Approach

Similar projects to ours have been conducted, such as the Introduction to Face Detection and Face Recognition by Shervin Emami [3]. Our project was based on this outline in the beginning, and has grown to include other methods and abilities. Our project consisted of many components, which are described in this section.

3.1 Detection

The first objective of this project is to identify human faces in an image. Our approach was to utilize Lienhart's implementation [7] of Haar Cascades, which has already been implemented in Intel's OpenCV library. [2] We utilized a pretrained cascade in the OpenCV library, which obtained acceptable accuracy and was used in this project.

3.2 Preprocessing

Facial recognition, especially Eigenfaces, is very dependent on positioning and lighting within an image. We created a simple program to preprocess our images for us. It utilizes our detection algorithm to crop the image, then it resizes the face to 100x100, then finally converts it to grayscale and equalizes the image. Utilizing the detection algorithm to crop aligns the faces nearly perfectly, and equalizing the grayscale image creates an even brightness and contrast across all faces. These two steps are shown in Figure 1.

This preprocess is also applied to every face found while running on the phone.

3.2.1 Database Generation

We utilized two databases in this project: The Yale Face Database [5] (15 subjects, 165 images) and a database we created from facebook of our friends (6 subjects, 70 images).

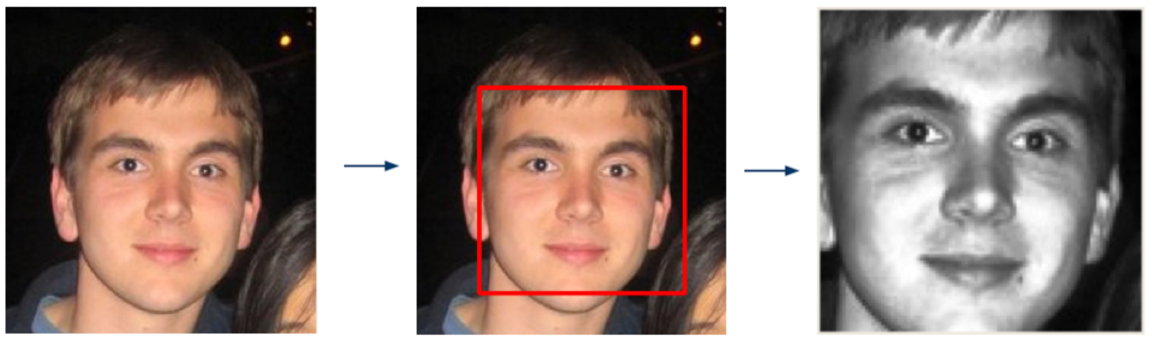

Figure 1: Steps in our preprocessing process.

All faces are preprocessed using the process described in 3.2. The Yale database was used mostly for testing, and our database was used in application.

Our application uses these images at run-time to generate the Fisher Linear Discriminant models.

3.2.2 Eigenvector Generation

One of the two databases described above is loaded, and the eigenvectors of the training data are generated using PCA, similar to [8]. This is done beforehand, and the eigenvectors are stored for use at application run-time.

Also, all images in the database are decomposed into the eigenvector space, and their vectors saved to disk for nearest neighbor calculation.

3.3 Eigenvector Nearest Neighbor

After a face has been preprocessed, we can identify it using the Eigenfaces method. This is done by projecting the face into the eigenvector space, or from a 100x100 matrix to a vector whose length is the number of eigenvectors. Nearest neighbors are then conducted on this new vector, comparing it to all images in the original database (their eigenvectors are pre-calculated for speed).

However, during testing of the Yale database, we decided to see what would happen if one of our faces were compared (we were not in the database). We assumed that it would pick out the neighbors that looked similar to us, but that's not what happened — instead, the 10 closest neighbors were all of different subjects with similar lighting. This was a red flag for us, and we decided to improve our program by using the Fisher faces method instead.

3.4 Fisher Linear Discriminant

Because using nearest neighbors seemed unable to directly classify faces correctly, we decided to use a more discriminative approach, namely, the Fisher Linear Discriminant (FLD) [10]. FLD attempts to find the single dimension along which data can be projected to produce maximum *separation*. The separation s can be defined as [10]:

$$s = \frac{\sigma^2_{between}}{\sigma^2_{within}} = \frac{(\vec{w} \cdot \vec{\mu}_{y=1} - \vec{w} \cdot \vec{\mu}_{y=0})^2}{\vec{w}^T \Sigma_{y=1} \vec{w} + \vec{w}^T \Sigma_{y=0} \vec{w}} = \frac{(\vec{w} \cdot (\vec{\mu}_{y=1} - \vec{\mu}_{y=0}))^2}{\vec{w}^T (\Sigma_{y=0} + \Sigma_{y=1}) \vec{w}} \tag{1}$$

Intuitively, we project all the data onto a vector $\vec{w}$, and then choose $\vec{w}$ to maximize the distance between the projected means divided by the sum of the widths of the projected covariances. Fortunately, the $\vec{w}$ that maximizes s can be calculated in closed form.

$$\vec{w} = (\Sigma_{y=0} + \Sigma_{y=1})^{-1} (\vec{\mu}_{y=1} - \vec{\mu}_{y=0}) \tag{2}$$

In reality, we add a normalizing term to ensure that the sum of covariance matrices is invertible, so we calculate $\vec{w}$ as:

$$\vec{w} = (\Sigma_{y=0} + \Sigma_{y=1} + \epsilon I)^{-1} (\vec{\mu}_{y=1} - \vec{\mu}_{y=0}) \tag{3}$$

Finally, Fisher's Linear Discriminant is designed to maximally separate only two classes from each other. To work around this limitation, we train k separate one-vs-all classifiers for each of the k subjects in our database. To classify a new test image using the database, we compute a score for each class and then choose the class with the highest score. The score we chose is the following ratio:

$$r^{(i)} = \frac{s^{(i)}_{test} - s^{(i)}_{\mu_0}}{s^{(i)}_{\mu_1} - s^{(i)}_{\mu_0}} \tag{4}$$

where $s^{(i)}_{\vec{x}}$ is the projection of $\vec{x}$ onto $\vec{w}$:

$$s^{(i)}_{\vec{x}} = \vec{w} \cdot \vec{x} \tag{5}$$

We could compute the FLD on the raw images, but given our 100 x 100 images we would have to invert a 10,000 by 10,000 dimensional covariance matrix from Equation 3 above, which would be both prohibitively expensive and unnecessary. Alternately, we can use PCA to reduce the dimensionality first, and compute $\vec{w}$ in this space instead. This is the approach we ended up using.

To tune the number of eigenvectors used, we compared the performance using different numbers of eigenvectors on classifying all the "happy" faces of the Yale data set. Results are shown in Figure 2. From these results, we selected 24 eigenvectors for all further work.

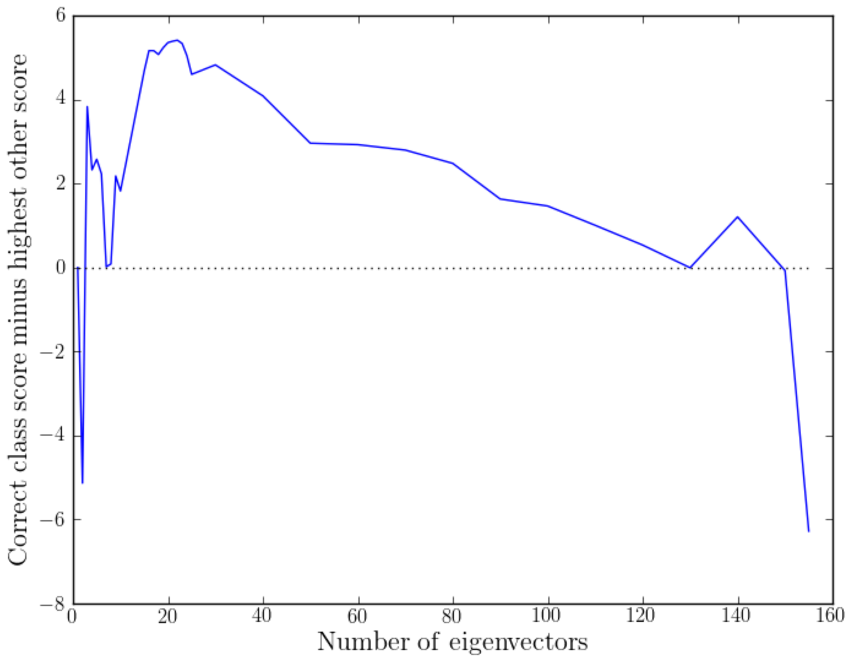

Figure 2: Discriminative power versus number of eigenvectors. Discriminative power is measured as the ratio score for the correct class minus the highest score for any other class. If this quantity is above zero, then the classification was performed correctly. This analysis helped us settle on using 24 eigenvectors for the remaining work.

3.5 Computation Considerations for Mobiles Devices

Mobile platforms are limited in both computational power and memory space. Throughout this project, this was a consideration in what algorithms to use, and how our code is implemented. To keep our application on the phone as responsive as possible, we broke op our application into three parts:

3.5.1 Preprocessing

We did as much computation as we could off the phone, such as preprocessing images, generating the database, and calculating eigenvectors. These results were saved to files and transferred to the phone with the application, which were loaded at run-time.

3.5.2 Run-Time

When the application is run it loads the database and other data files we have already created. The application then uses this to calculate the Fisher discriminative features. This causes our application to take about 5 seconds to load, which is a reasonable amount of time.

3.5.3 Picture

Finally, more calculation is done on-board the phone when a picture is taken. First, it applies the Haar cascades to find the faces, then it applies the preprocessing as described before. It then applies our Fisherfaces algorithm to label the incoming face. Finally, it paints a square and the label onto the image, displays it on the phone, and saves it to the phone. All this takes about 3 seconds, which is completely reasonable for phone-based calculations.

The Haar cascades take the longest, and originally took around 10 seconds. The time was cut by considering less scales while applying the cascade.

4 Results

4.1 Face Detection

Some results using our Haar wavelet detector are shown in Figure 3 through Figure 5.

4.2 Eigenvector creation

Eigenvectors were then created from the whole data set, as shown in Figure 6. These were used initially for nearest neighbor classification, and then as further input for the Fisher's linear discriminant model.

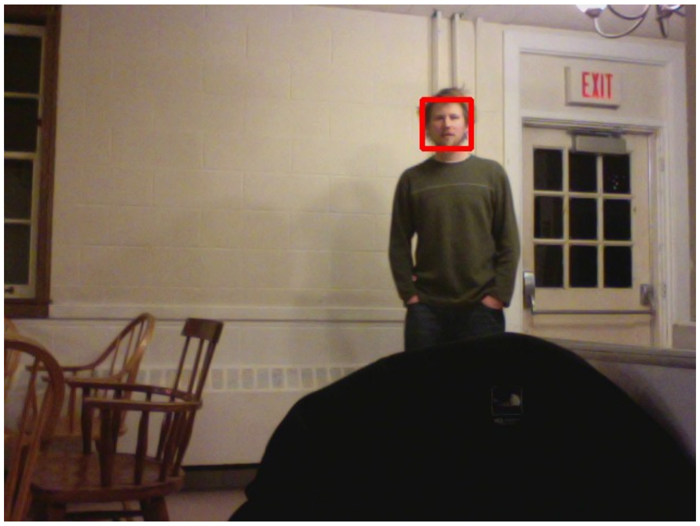

Figure 3: Haar wavelet detection of a face taking up a small fraction of the image.

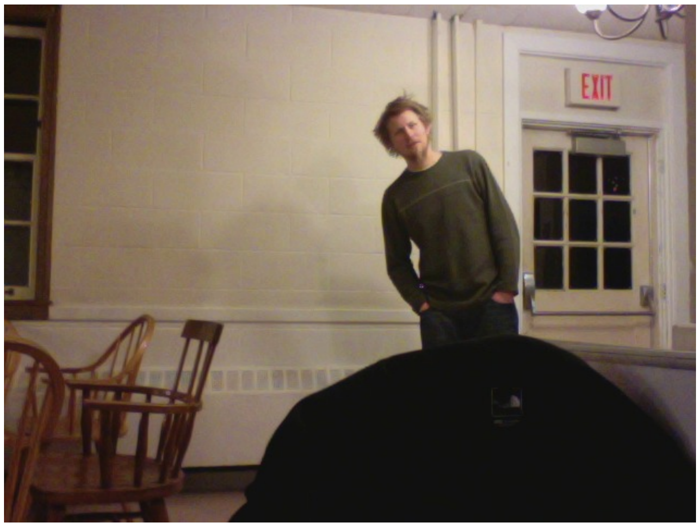

Figure 4: Our Haar wavelet detector tends not to work if the faces are rotated more than 30 degrees.

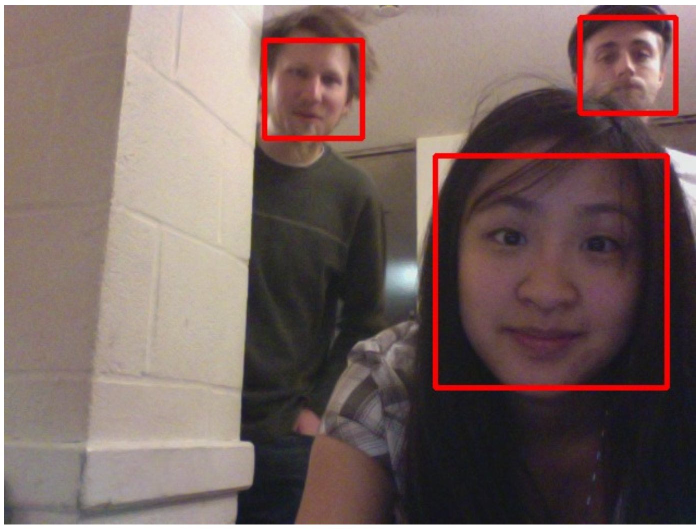

Figure 5: Detection of multiple faces at different scales in a single image works well.

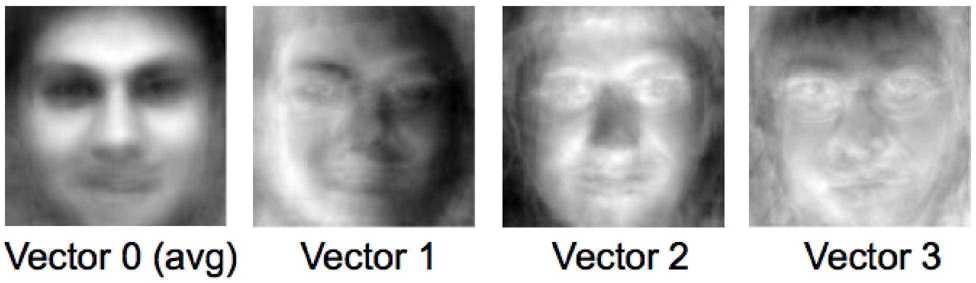

Figure 6: The first few computed eigenvectors of the images.

4.3 Fisher results on Yale Data set

On the yale data set [5], we achieved 100% classification results for the two sets of faces we tested on: the “normal” faces as well as the “happy faces”. Figure 7 shows a few of the subjects, along with the out-of-class mean ($\mu + 0$), the in-class mean (μ_1), and the computed *Fisher face*. The Fisher face is the projection onto visible units of the dimension of maximum separability $\vec{w}$.

4.4 Fisher results on phone/computer images

In addition to the Yale data set, we also classified correctly a high percentage of the pictures taken with the phone. Figure **??** and Figure **??** show a few examples of correctly classified images taken with the phone’s camera.

5 Discussion and Future Work

We found that the Fisher face method worked much better than the nearest neighbor eigenvector method, even when trained with a relatively small data set (10-15 faces per subject). Of course, this method is linear, and as such improvements may be made. One natural extension is to allow nonlinear separation between classes, perhaps via a support vector machine (SVM) method or similar.

We would also like to add the ability to link with Facebook, so that face database may be populated directly by the user’s friends and friends of friends.

6 Conclusion

We successfully created a working application that was able to find and identify faces on a mobile platform. We have showed high detection performance using a Haar cascade, and face recognition using a combination of Fisher faces and PCA analysis (Eigenfaces). This recognition showed very accurate performance on a standard benchmark, the Yale face database, as well as on a custom database assembled from users’ Facebook images.

References

[1] V. Blanz and T. Vetter. Face recognition based on fitting a 3D morphable model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1063–1074, 2003.

[2] G. Bradski. The OpenCV Library. *Dr. Dobb’s Journal of Software Tools*, 2000.

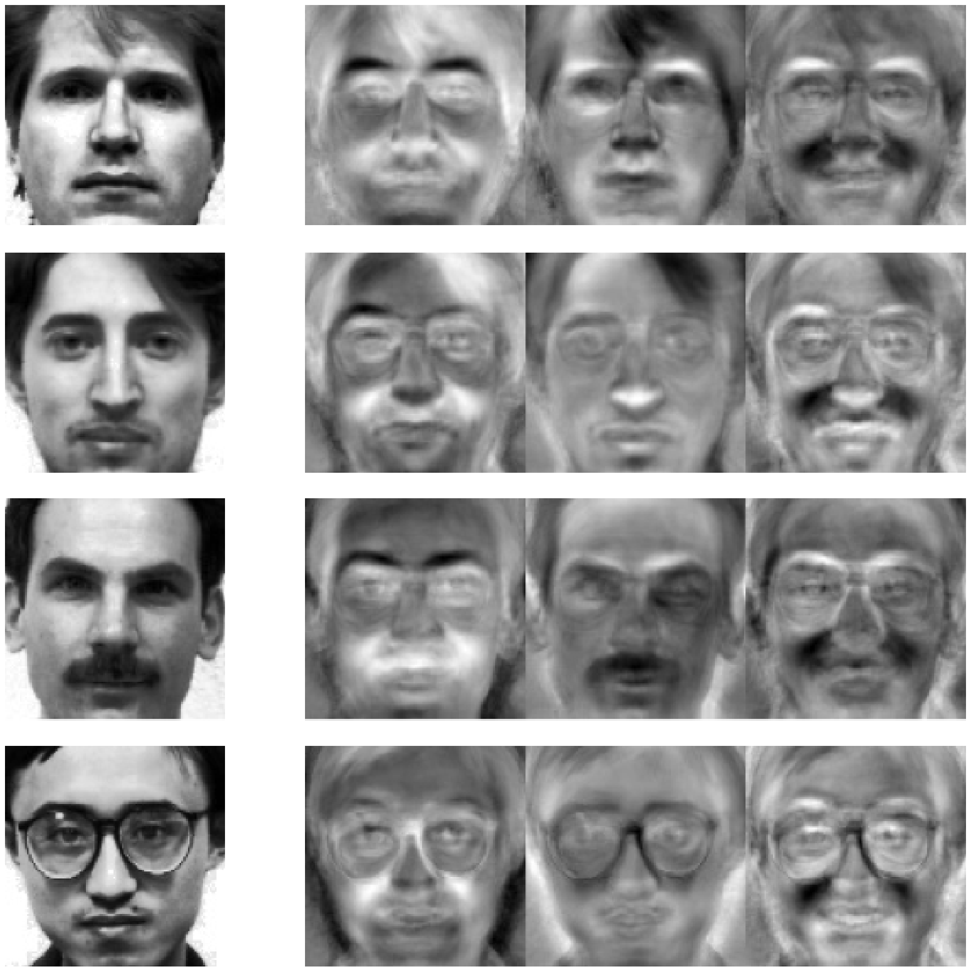

Figure 7: On the left are the “normal” faces in several of the classes. Next (horizontally) is the mean of the out-of-class samples, the mean of the in-class samples, and the Fisher face. The Fisher face is the dimension along which maximum linear separability occurs.

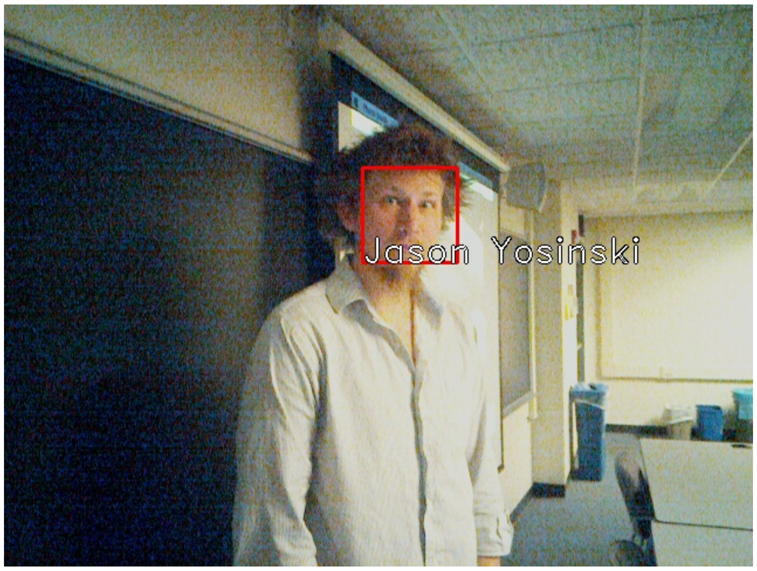

Figure 8: An example of a correctly classified image, taken during our presentation demo.

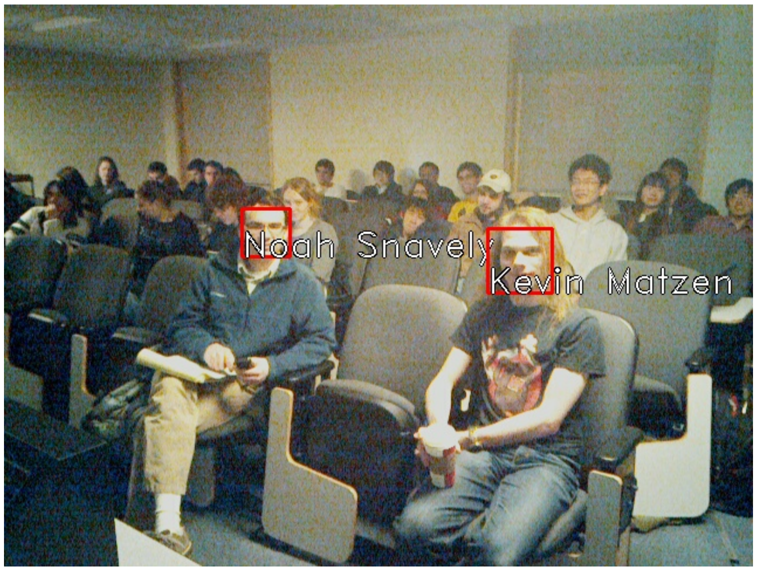

Figure 9: Another example of correctly recognized faces, in this case, with multiple faces in view. The many people in the background are not recognized as faces because they fall below a size threshold.

[3] Shervin Emami. Introduction to face detection and face recognition. `http://www.shervinemami.co.cc/faceRecognition.html`.

[4] K. Etemad and R. Chellappa. Discriminant analysis for recognition of human face images. In *Audio-and Video-based Biometric Person Authentication*, pages 125–142. Springer, 1997.

[5] Yale Center for Computational Vision and Control. Yale face database. `http://cvc.yale.edu/projects/yalefaces/yalefaces.html`.

[6] Y. Freund and R. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, pages 23–37. Springer, 1995.

[7] R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In *IEEE ICIP*, volume 1, pages 900–903. Citeseer, 2002.

[8] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991.

[9] P. Viola and M. Jones. Robust real-time object detection. *International Journal of Computer Vision*, 57(2):137–154, 2002.

[10] Wikipedia. Linear discriminant analysis. `http://en.wikipedia.org/wiki/Linear_discriminant_analysis#Fisher.27s_linear_discriminant`, 2010.