Business Assistant: Facial Recognition on Android Phones

Ervin Teng, Brian Jungman
Department of Electrical Engineering
Stanford University
Stanford, USA
{eteng, bjungman}@stanford.edu

***Abstract*—Facial recognition is an application of image processing that has begun to see more widespread use in the past few years. In this paper, we discuss the implementation of facial recognition techniques on an Android mobile phone, as well as the design and implementation of an application using facial recognition. We also analyze several tweaks to the algorithm, and their effects on both recognition performance and speed (frame rate).**

Keywords-face detection; face recognition; mobile phones, Android

I. INTRODUCTION

With the continuation of Moore's Law and the increasing pervasiveness of electronics in the everyday lives of people throughout the industrialized world, society is constantly being presented with numerous ways for the improvement of daily life, through matters big and small. With the drastic rise in the past few years of smartphones, and cellular phones in general, people are able to do more and more with the computers that they carry around in their pockets. In this paper, we discuss an application of image processing designed to make common occurrences in a person's life easier and potentially less embarrassing. Specifically, we discuss the options available to one looking to use technology to assist them with remembering the names of people they have met before.

II. APPLICATION OVERVIEW

In our approach to simplifying people's lives, we have decided to create an Android application that will implement our vision of this "Business Assistant." The app has two main functions: to add a new "user" to the database (a person that the "operator" would like to recognize in the future), and to recognize a user currently in the phone's database. After recognizing an existing user, the phone would then potentially perform various functions, including displaying their name on the screen and allowing the operator to view a photo of the user's business card.

A. Adding a New User

The first step in the process of adding a new user is to add photos of the new user's face to the phone's database. To do so, the operator points the phone at a user. The app will then detect whether there is a face in existence, using the Haar-like features designed to detect a face (Fig. 6). This detection is implemented via the OpenCV method detectMultiScale. The operator can then hit a button to add a user. If a face is detected, the phone will then take and save five photos of the user.

Each of these five images is preprocessed before being saved to the phone. Specifically, they each undergo: cropping of the photo to only contain the face, conversion to gray scale, photo resizing to 150 pixels by 150 pixels, and histogram equalization. These operations are done to minimize the noise in the training images, caused by factors such as things being behind the user, faces being either closer to or farther away from the phone's camera or uneven lighting conditions. We then retrain our eigenfaces algorithm, including the new user. The resulting eigenfaces and mean are saved to RAM.

The next step in the application is to enter the name of the user (Fig. 7). Once this is completed, the operator has the opportunity to add a photo of a business card. If the operator chooses to do this, he then saves the business card photo to the phone.

B. Recognizing a User

If the "Recognize" switch is turned on, the app will attempt to recognize any faces found on-screen. The face detection mentioned in the previous section is always active; however, recognition is only performed with the switch turned on. Assuming at least one face is present, the app takes the largest face found and assumes that face corresponds to the desired user. The next step is to perform the same preprocessing on the detected face. The grayscale image patch is cropped to consist of only the user's face, resized to be 150 pixels by 150 pixels, and has histogram equalization performed on it.

Next, the test image is projected onto the space defined by the stored mean and eigenvectors. The resulting image is thus projected onto the same lower dimensional space as when adding a new user. We then calculate the Euclidean norm of the difference between this projection and the projection of all the training images, and rank the matches based on this norm. This user's most likely name is then displayed above the detected face on the screen in real-time (Figs. 8, 10), and will update accordingly if the original user leaves the frame and another user enters. At any time that a user is found within the frame, the operator has the option to view the user's business card, which simply displays the business card image taken when adding that user (Fig. 9).

III. IMPLEMENTATION

In implementing our application, the first decision we had to make was whether we wanted to make an application that ran in real-time, or whether we wanted to take a photo and perform operations on it, either on the device or on a remote server. We immediately decided it was preferable to keep all of the operations on the device and not outsource to an outside server. This would allow for the application to be used even in situations where the operator does not have any network connectivity. One example where this could easily occur is if the application is installed on a tablet without cellular connectivity that is not in range of a familiar Wi-Fi network. After comparing the advantages and disadvantages of each real-time and one-time face recognition, we decided that the advantages of a real-time implementation outweighed the disadvantages.

The advantages we identified to a real-time implementation were:

- If the frame being processed does not have a recognized user, it is easier and faster to try another frame, reducing the impact of false rejections by the face recognizer.
- If the application has difficulty identifying a user, cues on the live screen (such as the box we display around the detected face) make it easier for the operator to make changes (such as rotating the phone to match the tilt of the user's head) that will allow the app to recognize the user's face.
- The interface feels more natural – taking a photo instills the feel of asking one's phone "please identify the person in front of me," while analyzing the frames in real-time has the feel of asking the much more natural question "who's that?"

We identified the following disadvantages to a real-time implementation:

- If the processing is not performed quickly enough the application will feel very slow and be a pain to use.
- We would not have the ability to explicitly focus the camera before taking the picture, making the quality of our test images to process be dependent on how well the camera auto-focuses when not taking a picture.
- If the user's face appears to be close to multiple different users in the database, the information about the user, such as the printout of his/her name on the screen, can change very quickly (flicker back and forth between different users), making it hard for the operator to determine who the user is.

After deciding to implement the face detection and recognition in real-time, the discussion of the algorithm to use in order to recognize the faces began. After researching options and looking at examples from previous years, we decided to implement the eigenfaces algorithm, due to the various features in the Java port of OpenCV that allow for assisting with this implementation. While the Fisherfaces algorithm generally sees better results than those seen with the eigenfaces algorithm, the projection matrices are much more complex to find than they are in eigenfaces. While this additional complexity would not be prohibitive if we were to implement our facial recognition application using MATLAB, we realized that computing these matrices in real-time on the Android phone was going to be prohibitively slow for our goals. Previous implementations of Fisherface on mobile phones, such as that seen in [1], have relied on pre-computation of these matrices via MATLAB and storing them on the phone. However, using this technique limits the usage of the app to only recognize users that existed in the training set when the matrices were created in MATLAB. In addition, the Fisherfaces algorithm relies on differing images (e.g. different lighting conditions) of every person in the database. One of our main goals with our application is that we want to be able to add new users within the app, and short of having the user pose for the operator, the images obtained would be similar, making the Fisherfaces algorithm not a huge advantage for our situation.

In an attempt to improve the performance of the relatively primitive eigenfaces algorithm, we attempted several tweaks to the post processing.

- Changing the number of eigenvalues/vectors from the default five.
- Turning on and off the histogram equalization.
- Cropping the square frame on the sides to remove more of the background.
- Performing eye detection and straightening the image respectively.

The implementation of most of these features is pretty self-explanatory. The straightening was performed by using a Haar Cascade similar to that used for face detection; we detected the eyes, calculated the slope between them, and rotated the image by that amount. Figure 1 shows an image processed in this way. Note the small eye detection circles and the replication artifacts on the top right and bottom left edges.

The results of these tweaks will be mentioned in section V.

Figure 1. Image after straightening

IV. EIGENIMAGES AND EIGENFACES

The eigenface algorithm for facial recognition uses the Karhunen-Loeve transform. For a vector (in the case of images, the images are reshaped into 1-dimensional vectors), unitary transforms conserve the total amount of energy in the matrix, but move the energy to different parts of the matrix. The KL transform is a type of unitary transform, with the transformation matrix $A = \Phi^H$. Φ is the eigenmatrix of the vector's autocorrelation matrix, R_{ff}. Out of all unitary transforms, the KL transform compresses the most energy into

the first entries of the matrix. By concentrating most of the energy of an image in the first columns and eigenvalues of the autocorrelation matrix, we can create a low rank matrix that provides a good approximation of the original image.

To apply the Karhunen-Loeve transform to images, each training image is first reshaped into a vector Γ_p of length MN. After calculating μ, the mean of all of the image vectors, we calculate the training matrix $T = [\Gamma_1 - \mu, \Gamma_2 - \mu, \dots \Gamma_P - \mu]$. We then compute the eigenvectors u_i and eigenvalues of the matrix $S = T^H T$. Each of these eigenvectors is of length L, where L is the number of training images in the database. We then decide the value to choose for M, which is the number of desired features we wish to use in our face recognition, and save the M eigenvectors u_i associated with the M largest eigenvalues.

When conducting the recognition of a test image via the eigenfaces algorithm, the first steps are to reshape the test image into a vector and subtract the mean, μ, found during the training phase. We then project the test vector onto the same space as in the training phase, by left-multiplying it by T^H. The final step is to compare it to the stored eigenvectors $u_i \; for \; i = 1, \dots, M$, each of size L, by looking at the Euclidean distance between the two vectors to determine the best match, and return the appropriate result.

V. Results

After adding many different users to our database, both real and on-computer, throughout the final class poster session, we performed tests of our application on a single user, and 20 different images of that user of varying lighting conditions and facial expressions. Fig. 2 shows the image of this user in the training database, and Fig. 3 shows the test images. To most closely replicate real-world application, we did the test through the phone's camera. Because we wanted to use the same training databases for all tests, we applied the modifications mentioned to the database images during the training step rather than the capture step.

Fig. 4 shows the effects of removing the histogram equalization operation, performing an additional crop on the photo to reduce the width of the photo, and adding in the ability to detect the user's eyes and rotate the image accordingly. As we expected, removing the histogram equalization resulted in less accurate results, due to the variance in lighting conditions. We expected that performing additional cropping of the photo to remove pixels from the left and right edges of the photo would improve the results (because it removes background pixels due to the typical ovality of a face), but contrary to our expectations this cropping performed worse than the default algorithm. This is likely because the extremities of a face still contain significant information, such as ears or hairstyle, which even humans use to differentiate between people. We were also surprised to see that detecting the position of a user's eyes and rotating the image so that their eyes were horizontal performed worse than using the image in its original unrotated position. We believe that this may be because the images in the database were taken with the images were taken before the rotation algorithm was implemented, and expect to see better performance on new users that are added after this addition is implemented.

Figure 2. Training image of chosen user

Figure 3. Testing images of chosen user, scaled to fit in a grid

Our application initially used 5 eigenvalues and eigenvectors to compare the test image to the images in the database. However, after discussions during the poster session we conducted tests to see the effects of changing the number of eigenvalues used. The results of these tests are shown in Fig. 5. As we expected, using more eigenvalues resulted in generally better performance. We were surprised to see that using 5 eigenvalues resulted in better performance for recognizing the correct user than 7, 12, or 15 eigenvalues, but when we looked at which method had the correct user in the top 2, 3, 4, or 5 users choosing more eigenvalues generally performed the best. The exception to this is that choosing 10 eigenvalues consistently outperformed all of the other options, even 12 and 15 eigenvalues.

An additional figure of merit particular to our problem is the issue of frame rate. Thankfully, it turns out that as long as the training is performed beforehand, the bottleneck is actually the face detection, not the recognition, since the recognition images are fairly small. The basic algorithm, with or without live recognition, runs at around 12 fps on a 1.2 GHz dual core tablet running Android 3.2. Adding histogram equalization and cropping do not significantly affect the frame rate. However, the additional processing required to detect eyes and rotate the image halve the frame rate. While still usable, this greatly diminishes the augmented reality feel of the application.

VI. Conclusions

In this project, we implemented facial detection and recognition to allow an operator to find the name of and other information about a user in the phone's database. There are several limitations to our algorithm that affect the accuracy of the application. If the lighting conditions are significantly different than they were in the training images, then the algorithm saw significantly lower accuracy than in similar lighting conditions. Additionally, our algorithm had performance issues when attempting to identify users of Asian descent, possibly due to the large percentage of our database consisting of people of Asian descent.

As part of future work, we would like to implement Fisherfaces on the Android device, while simultaneously

adding a longer interval between captures. It might also be possible to use the camera's flash to alter the lighting conditions between captures of the same user, thus improving the performance of Fisherfaces. We believe that this would cause significant delays when adding new users to the database, but the potential performance increase may be worth the additional delay. We would also like to explore better algorithms for face detection and face segmentation, to reduce the number of false negatives we received in our application.

VII. References

[1] Guillame Davo; Xing Chao; Kishore Sriadibhatla, "Face Recognition in Mobile Phones," Stanford University EE 368 Spring 2010

[2] Bangpeng Yao; Haizhou Ai; Shihong Lao, "Matching Texture Units for Face Recognition." Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on, vol., no., pp.1920-1923, 12-15 Oct. 2008

[3] Han Yanbin; Yin Jianqin; Li Jinping, "Human Face Feature Extraction and Recognition Based on SIFT," Computer Science and Computational Technology. 2008. ISCST '08. International Symposium

[4] Fei-Fei Li, "OpenCV and Face Detection," Princeton University COS 429 Fall 2008

[5] Enami, Shervin, "Introduction to Face Detection and Recognition." http://www.shervinemami.info/faceRecognition.html. 04 Feb. 2012

VIII. Work Breakdown

Ervin Teng: Algorithm design, implementation, verification

Brian Jungman: Algorithm design, debugging, testing

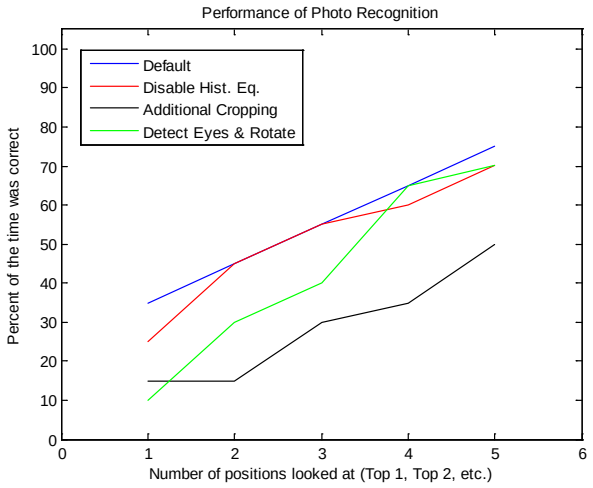

Figure 4. Performance of face recognition with the default conditions, histogram equalization disabled, and additional face cropping

Figure 5. Performance of face recognition with varying numbers of eigenvalues

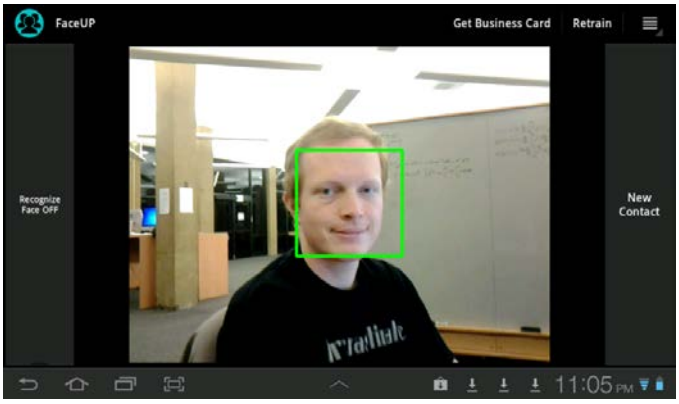

Figure 6. The screen before adding a user

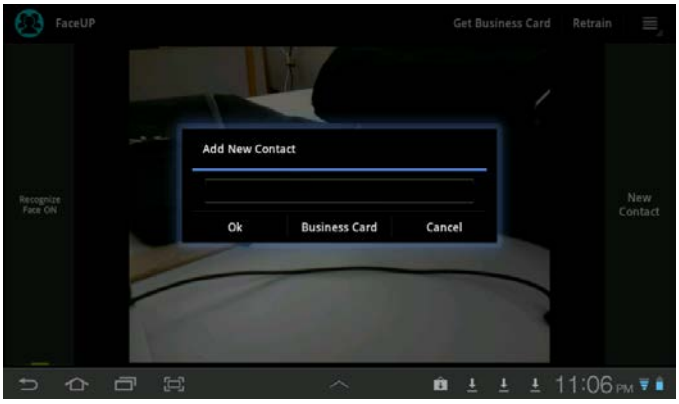

Figure 7. The screen to add a new user

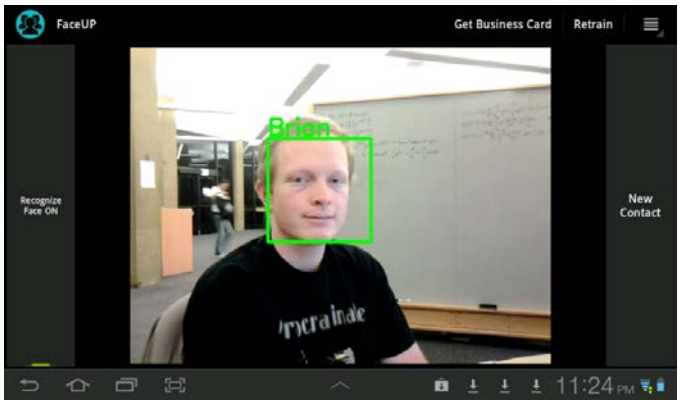

Figure 8. Recognizing an existing user

Figure 9. Displaying an existing user's business card

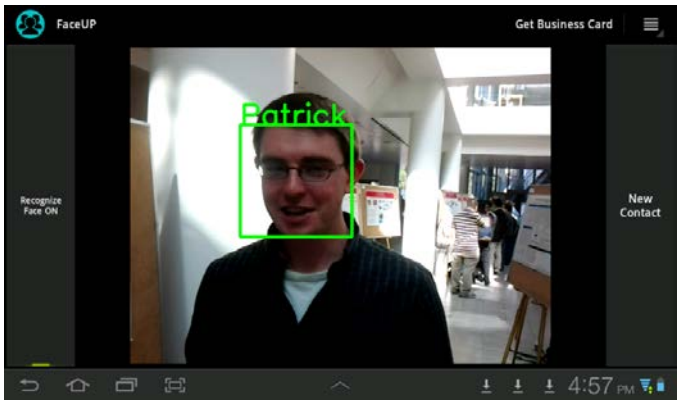

Figure 10. Recognizing an existing user