FACE RECOGNITION APPLICATION BASED ON EMBEDDED SYSTEM

by

WEIHAO GAO

Submitted in partial fulfillment of the requirements
For the degree of Master of Science

Thesis Advisor: Professor Christos Papachristou

Department of Electrical Engineering and Computer Science

CASE WESTERN RESERVE UNIVERSITY

January, 2014
We hereby approve the thesis/dissertation of

Weihao Gao

candidate for the___Master____ degree *

Christos Papachristou

Frank Merat

Daniel Saab

(date) ____November 11, 2013____

*We also certify that written approval has been obtained for any proprietary material contained therein.
# Table of Contents

1. Introduction………………………………………………………………9

1.1 Background and Significance………………………………………9

1.2 The main contents and structure of this paper…………………11

2. Research Plan…………………………………………………………13

2.1 Research Contents…………………………………………………13

2.1.1 Basic Process. .................................................................13

2.1.2 Key Issues .................................................................13

2.2 Research plan………………………………………………………14

2.2.1 Comparison of face recognition algorithms………………14

2.2.2 Functional Analysis of the Application…………………..16

2.3 Knowledge Chart…………………………………………………….19

3. Obtain the Face Image…………………………………………………..20

3.1 Android Program Platform…………………………………………20

3.1.1 Build up the Android development platform………………20

3.1.2 Brief introduction of Android……………………………….20

3.2 Face Image Capture…………………………………………………..22

3.2.1 Activity in Android………………………………………………22

3.2.2 Android Program Structure…………………………………….24

3.2.3 Control the Camera ..........................................................26

3.3 Face Detect ……………………………………………………………27

3.4 Face Detect Algorithm ..........................................................29
3.4.1 Rectangle features and integral image ................................. 30

3.4.2 AdaBoost training process ............................................. 33

4. Image preprocessing .......................................................... 38

  4.1 Unify the Size ............................................................. 39

  4.2 Grayscale ................................................................. 42

  4.3 Histogram equalization .................................................. 43

  4.4 Image Smoothing ......................................................... 45

5. PCA Algorithm ............................................................... 47

  5.1 Training ................................................................. 48

  5.2 Recognition ............................................................. 49

  5.3 Using Open CV in PCA algorithm ................................... 50

  5.4 PCA algorithm principle ............................................... 51

6. Results and Analysis ...............................................................

  6.1 Test 1 .................................................................... 55

  6.2 Test 2 .................................................................... 56

  6.3 Strategies ................................................................. 56

  6.4 Dimension number and recognition rate ............................ 61

7. Conclusion and Future Work ..................................................

  7.1 Conclusion ................................................................. 63

  7.2 Future Work .............................................................. 64

Appendix A: Notebook Application Screenshots ............................. 65

Appendix B: Notebook Application Introduction ........................... 68
References

69
List of Tables

Table 3-1: Haar features.................................................................30
Table 3-2: Calculation of Haar features ........................................33
Table 3-3: Threshold calculation of a weak classifier .......................36
Table 6-1: Different number of feature dimensions and their corresponding result..61
List of Figures

Figure 2-1: Basic application procedure .................................................. 18
Figure 2-2: Knowledge chart ................................................................. 19
Figure 3-1: Android system architecture ................................................. 21
Figure 3-2: Activity lifecycle ............................................................... 23
Figure 3-3: A sample of Android application structure ........................... 25
Figure 3-4: Eye detection on Android ................................................... 29
Figure 3-5: Haar features applied in face location ................................. 31
Figure 3-6: m × m sub-window ............................................................ 32
Figure 4-1: Judgments by the end of face detection ............................... 40
Figure 4-2: Dimensional display of gray value in bilinear interpolation .... 41
Figure 4-3: Color image gray-scale transformation ............................... 42
Figure 4-4: Histogram equalization ...................................................... 44
Figure 4-5: Example of Histogram equalization ................................. 44
Figure 5-1: Schematic diagram of a face recognizer ............................ 50
Figure 5-2: PCA algorithm in Android ................................................ 51
Figure 5-3: Five samples ................................................................. 52
Figure 5-4: Two projections ............................................................... 52
Figure 5-5: Map to a less-dimensional plane ....................................... 53
Figure 6-1: Failed sample ................................................................. 56
Figure 6-2: Test result ................................................................. 58
Figure 6-3: Training images ............................................................... 58
Figure 6-4: Person A(left)  Person C(right)  

Figure 6-5: Person B in the training group  

Figure 6-6: Person D in the training group  

Figure 6-7: Shortest distance results  

Figure 6-8: Eigenfaces and mean image  

Figure 7-1: Recent free download times from the internet
Face Recognition Application Based On Embedded System

Abstract

by

WEIHAO GAO

The purpose of this application is to develop an embedded system application which is able to collect face image and recognize the face by comparing with the database inside the system.

Face recognition as a type of biometric methods has the features of non-contact, safety and convenience. It is widely used in human-computer interaction, transaction authentication, security and other fields.

Recent years, with the development of mobile internet and embedded computer, it becomes possible to run face recognition on embedded system. This type of application has huge potential in remote payment and personal information security. This application is running on Android operating system which is an operating system based on the Linux kernel, and designed primarily for touchscreen mobile devices such as smartphones and tablet computers.

The procedure of face recognition includes face detection, face normalization and recognition. This paper studies these key issues and successfully developed an application with nice recognition rate.

The main contents and results are as follows:
1) Discusses the face detection method. It used Adaboost algorithm and Haar features to detect human faces.

2) Studies image pre-processing methods. Standardize the images so as to minimize the storage space and speed up the computation speed.

3) Summarize a variety of face recognition algorithms especially principle component analysis which is used in this application. Discuss the theoretical foundation of PCA algorithm.

4) Fulfilled all the features from face detection to recognition in Android platform. Using ORL face image database for testing and got a correct identification rate of over 85%. Fully verify the effectiveness of the program. Discuss the results and identification strategies.

**Key words:** Android; OpenCV; Adaboost; Face recognition; PCA
1. Introduction

1.1 Background and Significance

Face recognition is one of the most important areas in image analysis. The research of face recognition can be traced back to 1970s [1][2][3]. With decades of study, face recognition has improved a lot. There are famous international conferences such as Conference on Computer Vision and Pattern Recognition, International Conference on Image Processing and Automatic Face and Gesture Recognition. Face recognition becomes a hot research area because it has great potential and wide social need. An application of face recognition can be used in the following areas:

1. Identification of identity: Passports, identity cards and other automatic identification.
2. License: Confidential departments access, ATM machines.
3. Intelligent monitoring: suspects tracking and alarm in unattended situation.

Let’s make an example, if we need to get money out of an ATM machine, we might have to provide the PIN (personal ID number) and password. However, we might forget the password or somebody can find out the password. If we are using human face recognition as an additional protection, we might not need to worry too much if we lose our credit card.
Face recognition also has the following features:

1. Compared to other biometrics, such as fingerprint and DNA, face recognition does not need the tester to cooperate.
2. Image sample collection can be done in a natural and easy way which does not require complex equipment.
3. A lot of neuroscience research has shown that face recognition is controlled by a certain part of human brain and is very important to us. Research on face recognition is important for us to understand human brains and also important to artificial intelligence and machine learning.

Moreover with the development of computer science especially computing capabilities, it comes possible to run very complex programs on embedded system. There are many embedded systems focus on safety monitoring and personal digital assistant. For the reasons above, I decide to develop a face recognition application in embedded system.

Android is a Linux-based operating system designed primarily for touchscreen mobile devices such as smartphones and tablet computers.

Android has opened the source code which allows the software to be freely modified and distributed by device manufacturers, wireless carriers and enthusiast developers.

Additionally, Android has a large community of developers writing applications that extend the functionality of devices, written primarily in a customized version of
the Java programming language. In October 2012, there were approximately 700,000 apps available for Android, and the estimated number of applications downloaded from Google Play, Android's primary app store, was 25 billion.

These factors have allowed Android to become the world's most widely used smartphone platform, and also the hottest embedded environment in recent years.

My application is working on Android environment mainly because it is free, easy and convenient. There are many useful face recognition applications based on personal computer but we can hardly find any of them based on embedded system. Even though there are some very professional recognition products on specialized embedded system, there are too big and expensive. With the development of computer technology, it becomes possible to do face recognition on smaller devices like a smartphone or other mobile devices [4][5].

1.2 The main contents and structure of this paper

The application I develop here is to do face recognition in android system. It is going to help to protect the phone owner’s information and property security. Moreover, this application might also be used as a type of Implantation code in other Android application which helps to protect their application. In this thesis, the overall structure is divided into five chapters.

Chapter 1 Introduction This chapter describes the background of Android and face recognition, significance.

Chapter 2 research programs and technology roadmap, this chapter introduces the various face recognition algorithms, programs and implementation technology
Chapter 3 image captures, this chapter introduces the basics of Android programming platform, and by controlling the mobile phone camera for face capture and face detection. Introduce AdaBoost face detection algorithm.

Chapter 4 image preprocessing, this chapter describes the relevant image processing algorithms and implementation, including grayscale, gray image histogram equalization, bilinear interpolation and image smoothing.

Chapter 5 Principal component analysis algorithm and implementation, this chapter describes the usage of PCA algorithm, and Open CV programming platform with Android programming library sharing. This chapter also introduces the theoretical foundation and training practice of PCA on embedded system.

Chapter 6 Result and analysis, I am going to discuss the testing results of the face recognition application on embedded system.

Chapter 7 Conclusions a summary of the full text and the prospect of this topic.
2. **Research Plan**

2.1 **Research Contents**

2.1.1 **Basic Process**

Firstly use the camera of the embedded system to get the face image of the target person. After that do basic face image pre-processing algorithm to the image such as color grayscale, histogram equalization and image smoothing filter. Then we are going to catch the face image using face detect algorithm and change it into a standardize size. Connect the Open CV class libraries with Android environment and use the principal component analysis (PCA) algorithm for face image feature extraction [6].

2.1.2 **Key Issues**

My topic focuses on solving the following issues:

1. Preprocessing of the face image

   In this application I tried different ways of pre-processing the original images. Image preprocessing is going to minimize the size of the image and also enlarge image contrast.

2. Face detection and standardize the face image

   In Android system there is already algorithms and function which help us to locate the face image. There are also functions in Open CV which helps us to locate the face image.

3. Face recognition
Different algorithms are developed in doing face recognition. However some of them are not useful in running on an embedded system. In this application I choose principal component analysis in doing face recognition.

In this face recognition application, several types of decision can be made. What we call face recognition is a broad term which may be further specified to one of following tasks:

1. Identify if the target person is one of the person in the database.
2. Recognize if there is more than one person on the picture and to find out if they are in the database.
3. Categorize the target face to a certain class.

These different targets should be reflected in the application.

2.2 Research plan

2.2.1 Comparison of face recognition algorithms

There are different face recognition methods:

1. Face recognition based on geometric characteristics

Firstly use a geometric feature vector to represent the human face. Using a hierarchical clustering based pattern recognition to do human face recognition\cite{7}\cite{8}. Facial feature vector is a geometric shape and geometry of organs based on feature vectors, Whose components usually include a custom face the Euclidean distance between two points, curvature or angle\cite{9}.

2. Face recognition using template matching
Template matching method is mostly used normalized cross-correlation, direct
calculation of the degree of matching between two images [10]. Since this method
requires the target of two images have the same scale, orientation and illumination
conditions[11], the pre-normalization become very important of the work. There are
some typical matching functions that we can use to do comparation with the images in
the database.

3. Face recognition using hidden Markov model

Hidden Markov Model is a signaling system used to describe the characteristics
of a common statistical model which is widely used in voice recognition. Facial
features are in accordance with the distribution of a natural sequence, i.e. from the top
to the small, left to right, even if the human face in the plane and the vertical direction
is rotated, this order is not going to be change, However it becomes difficult to
implement[12][13].

4. Face recognition using neural network

This is a very hot research direction in recent years, which has a very good
ability to deal with damaged facial images and different illumination conditions.
However, it needs to implement a great number of neurons and a long training
time.These features make this hardly to fulfill in embed system[14][15].

5. Principle component analysis method

Principle component analysis is the algorithm I am using in this application.
Compared to the algorithm above, it is more simple and it has no parameter limits.
Principle component analysis is also called eigenface method[16]. Eigenfaces are a set
of eigenvectors used in the computer vision problem of human face recognition. The approach of using eigenfaces for recognition was developed by Sirovich and Kirby (1987) and used by Matthew Turk and Alex Pentland in face classification. It is considered the first successful example of facial recognition technology. These eigenvectors are derived from the covariance matrix of the probability distribution of the high-dimensional vector space of human faces.

2.2.2 Functional Analysis of the Application

To do face recognition, firstly use the camera of the embedded system to get the face image of the target person. After that do basic face image preprocessing algorithm to the image such as color grayscale and histogram equalization. Then we are going to catch the face image and changing it to a standardize size. Connect the Open CV class libraries with Android environment and use the principal component analysis (PCA) algorithm for face image feature extraction.

Let’s consider face identification first. Each face in the training set is transformed into the face space and its components are stored in memory. The face space has to be populated with these known faces. An input face is given to the system, and then it is projected onto the face space. The system computes its distance from all the stored faces.

However, two issues should be carefully considered:

1. What if the image presented to the system is not a face?
2. What if the face presented to the system has not already learned, i.e., not stored as a known face?
The first defect is easily avoided since I am using preprocessing algorithm to standardize the face image. The images with a low correlation can be rejected in the comparison with the face database. Or these two issues are altogether addressed by categorizing following two different regions:

1. Near face space and near stored face => known faces
2. Near face space but not near a known face => unknown faces

The flow chart of the very basic application is shown Figure 2-1:

![Figure 2-1: Basic application procedure.](image-url)
If we are looking for one person among a great amount of people, how many images of that person we should have in our database so as to balance the relationship between computation speed, hardware space and correct rate? Moreover, we might face some other problems such as difference of environment, difference of the person’s facial angle and distance of the person’s face to the camera.

In a certain application, the threshold and index changes if the database changed.

Different situations might make difference to the application. The recognition will become more difficult and perhaps we might need somebody to check the result when the similarity is greater than a threshold. In this paper I have done some testing and derived some strategies to solve different type of recognition problems.

2.3 Knowledge Chart

From the above analysis, the technical route is shown in Figure 2-2:
Figure 2-2: Knowledge Chart.
3. **Obtain the Face Image**

3.1 **Android Program Platform**

3.1.1 **Build up the Android development platform**

Android is the most popular mobile phone platform. The development environment can be built in both Windows and Linux environment[17].

To build up the android environment:

1. Firstly, we have to download the latest JDK application from http://www.oracle.com/technetwork/java/index.html. JDK is the Java environment we need for the Android environment. After all Android application is run under Java code.

2. Secondly, we have to download the latest SDK and install in our computer. SDK is the development environment of Android. We can download this from the official website of Android.

3. To write a android program in our computer we need a software to write and debug. Eclipse is the most common Java and Android development software. We can download the free version from: http://developer.android.com

After we have all these three parts downloaded, we need to put them together, this process is widely introduced on the internet.

3.1.2 **Brief introduction of Android**

Android system architecture has four-layer structures, from the upper to the lower layer are application layers, application framework layer, system libraries layer and Linux kernel layer. Figure 3-1 shows the architecture of Android system.
Figure 3-1: Android system architecture

They have different functions as follow:

1) Linux kernel layer

Android is based on Linux2.6 kernel, the core system services such as security, memory management, process management, network protocols, and driver model relies on the Linux kernel.

2) System libraries layer

Android provide a library code in C/C++. It include different functions such as:

SQLite, a lightweight database engine open to all functions.

SGL: Skia graphics library, the underlying 2D graphics engine.

FreeType: bitmap and vector font.

Libc: inherited from the standard C system library of BSD, Specifically for embedded linux device.

OpenGL: 3D graphics library based on OpenGL ES 1.0API standard
3) Application Framework

Android provides a set of application development framework. Such as: Activity Manager, Window Manager, Content Providers, View Providers, Package Manager, Telephony Manager, Resource Manager, Location Manager, Notification Manager. Developers can use the API provided by the framework to achieve their functions. It greatly simplify the code which make the code more reusable.

4) Application layer

Android platform is not just the operating system, Also contains a number of applications, Such as SMS messaging client program, Phone Dialer, Picture Viewer, Web browser and other applications.

These applications are written in the Java language, And these applications can all be replaced by other applications, This differs from the other mobile operating systems, more flexible and personalized.

3.2 Face Image Capture

Face image capture is the beginning of this application. It is not only about using camera of the Android embedded system but also include face detection.

3.2.1 Activity in Android

Unlike other programming paradigms in which apps are launched with a main() method, the Android system initiates code in an Activity instance by invoking specific callback methods that correspond to specific stages of its lifecycle. There is a sequence of callback methods that start up an activity and a sequence of callback
methods that tear down an activity. Figure 3-2 shows the activity lifecycle expressed as a step pyramid.

![Activity Lifecycle Diagram](image)

Figure 3-2: Activity lifecycle

This shows how, for every callback used to take the activity a step toward the Resumed state at the top, there's a callback method that takes the activity a step down. The activity can also return to the resumed state from the Paused and Stopped state.

There are several situations in which an activity transitions between different states that are illustrated in figure 3-2. However, only three of these states can be static. That is, the activity can exist in one of only three states for an extended period of time:

- **Resumed**
  
  In this state, the activity is in the foreground and the user can interact with it (Also sometimes referred to as the "running" state).

- **Paused**
In this state, the activity is partially obscured by another activity—the other activity that's in the foreground is semi-transparent or doesn't cover the entire screen. The paused activity does not receive user input and cannot execute any code.

Stopped

In this state, the activity is completely hidden and not visible to the user; it is considered to be in the background. While stopped, the activity instance and all its state information such as member variables is retained, but it cannot execute any code.

The other states (Created and Started) are transient and the system quickly moves from them to the next state by calling the next lifecycle callback method. That is, after the system calls onCreate(), it quickly calls onStart(), which is quickly followed by onResume(). That's it for the basic activity lifecycle.

3.2.2 Android Program Structure

Figure 3-3 shows a sample of Android application structure:
Figure 3-3: A sample of Android application structure

Src is the folder which we use to save the source code.

Gen is the folder which has all the files created by the application itself.

Bin is the folder where we can place pictures and layout. Layout is a xml file which help us to arrange the layout of the application.

AndroidManifest.xml is a file we use to arrange the basic information of the application. For example, we have to add the following code in AndroidManifest.xml file so as to have the right to visit the sd card:

```xml
<uses-permission
    android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
```
Whenever we add a new activity to the application, we need to add the information of this activity to AndroidManifest.xml.

### 3.2.3 Control the Camera

To use the camera of the android device, we have to use intent function in Android. Intent is a special and important function in android embedded system design. It not only helps to guide the application jump from one activity to another but also help to pass data and start function.

The key code of take picture in Android is shown below:

```java
try {
    PHOTO_DIR.mkdirs();//

    Intent imageCaptureIntent = new Intent(
        MediaStore.ACTION_IMAGE_CAPTURE);

    picName = getPhotoFileName();
    picName = picName.replace("-", "");
    picName = picName.replace(":", "");

    File out = new File(PHOTO_DIR, picName);

    Uri uri = Uri.fromFile(out);

    imageCaptureIntent.putExtra(MediaStore.EXTRA_OUTPUT, uri);
    imageCaptureIntent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY, 1);
}
```
startActivityForResult(imageCaptureIntent, CAMERA_WITH_DATA);

} catch (ActivityNotFoundException e) {
    Toast.makeText(this, "doTakePhoto: e=" + e, Toast.LENGTH_LONG)
        .show();
}

Moreover, before we take a picture in android, we have to verify that there is a
SD card installed in our system. The key code is shown below:

    String status = Environment.getExternalStorageState();

    if (status.equals(Environment.MEDIA_MOUNTED)) {
        doTakePhoto();
    } else {
        Toast.makeText(Takephoto.this, "There is no sd card", Toast.LENGTH_LONG).show();
    }

3.3 Face Detection

Before we do face recognition, we have to standardize the face image. To
standardize the face image, it is necessary to find out if the picture do has a human
face on it, and cut the face image into standard size. The face judgment procedure
should be done before the preprocessing of the face image. There are four different
circumstances in this part.

1) There is no face image on the picture
2) There are several face images on the picture

3) There is only one face image on the picture but the face is incomplete.

4) There is one face image and it is complete.

Only in situation 4 we can go to the next step.

The key code of this face judgment is shown below:

```java
int width = myBitmap1.getWidth();

    int height = myBitmap1.getHeight();

    Bitmap myBitmap = Bitmap.createBitmap(myBitmap1, 0, 0, width,

        height, matrix, true);

    imageWidth = myBitmap.getWidth();

    imageHeight = myBitmap.getHeight();

    myFace = new FaceDetector.Face[numberOfFace];

    myFaceDetect = new FaceDetector(imageWidth, imageHeight,

        numberOfFace);

    numberOfFaceDetected = myFaceDetect.findFaces(myBitmap,

        myFace);
```

Another important situation worth noting is that: what if the user takes a flipped picture. There is no way the embedded system can find a face image in upside down picture. We can solve this problem by using the gyro device of the system to recode the shooting angle however this process is going to greatly slow down the speed of
the program what’s more it will not totally avoid this problem. In that case, I set all the testing into a vertical shooting environment. Camera angle is not changing.

There are already face detect API in android. We only have to import android.media.FaceDetector and android.media.FaceDetector.Face to our program. However, this is not going to give out the face on an image but give out the eyes of the person. Moreover, we can also use Open CV to load classifications. It is also based on the detection of human eyes. Figure 3-4 shows the eye detection on android:

![Eye detection on android](image)

**Figure 3-4: Eye detection on android**

Since we are aiming to do face recognition in embedded system, it is very important to have a clear face image at first instead of a blurred out image.

**3.4 Face Detect Algorithm**

Face detection algorithms in both Android and Open CV are to detect the eyes on human face. They are based on AdaBoost algorithm. AdaBoost is the short for
adaptive boosting which is a machine learning algorithm formulated by Yoav Freund and Robert Schapire. AdaBoost algorithm is very sensitive to noisy data and outliers [18]. Basic idea of Adaboost algorithm is to use a large number of weak classifiers to add up together to form a very strong classification through a certain method [19]. Each weak classifier may perform very weak, just a little better than random. Combination of large number of weak classifiers can greatly improve the performance [20][21]. Using AdaBoost algorithm into eye detection, we need large number of training samples with human eyes and other images. Each training sample is assigned a weight, and it demonstrates a probability of some weak classifiers which can be selected into training set. If a sample is accurately classified by the current weak classifier, its weight will be reduced. On the contrary, if a sample is not properly classified, the weight is to be raised accordingly [22][23].

3.4.1 Rectangle features and integral image

Since AdaBoost is a statistical method, it is important to choose effective feature for classification. Haar features are introduced into face detection [24]. The five basic Haar features are shown Table 3-1, s and t are the ratio of horizontal and vertical pixel:

<table>
<thead>
<tr>
<th>Haar</th>
<th>1.</th>
<th>2.</th>
<th>3.</th>
<th>4.</th>
<th>5.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s, t)</td>
<td>(1, 2)</td>
<td>(2, 1)</td>
<td>(1, 3)</td>
<td>(3, 1)</td>
<td>(2, 2)</td>
</tr>
</tbody>
</table>

Table 3-1: Haar features
Haar feature value is the total gray value in the white region minus total gray value in the black region. Feature-based detect system is much faster than pixel-based system which is more sensitive to simple graphical structure. Facial features can easily described by a rectangle features, for example, usually the eyes are darker than the cheeks; nose on both sides of the color to be deeper than the nose; mouth deeper than the surrounding colors [25].

![Haar features applied in face location](image)

**Figure 3-5: Haar features applied in face location**

In a $m \times m$ sub-window, we only need to determine the top-left corner $A(x_1, y_1)$ and lower right rectangle $B(x_2, y_2)$ to find out a feature rectangle

1) $x_2 - x_1$ can be divided into $s$ segments.

2) $y_2 - y_1$ can be divided into $t$ segments.

The minimum size of this rectangle is $s \times t$ or $t \times s$, the maximum size is $[m/s] \cdot s \times [m/t] \cdot t$ or $[m/t] \cdot t \times [m/s] \cdot s$. 

31
The number of rectangle features in the m × m sub-window is:

\[ A(x_1, y_1) : \quad x_1 \in \{1, 2, \ldots, m-s, m-s+1\}, \quad y_1 \in \{1, 2, \ldots, m-t, m-t+1\}; \]

\[ x_2 \in X = \{x_1 + s - 1, x_1 + 2 \cdot s - 1, \ldots, x_1 + (p - 1) \cdot s - 1, x_1 + p \cdot s - 1\}, \]

\[ y_2 \in Y = \{y_1 + t - 1, y_1 + 2 \cdot t - 1, \ldots, y_1 + (q - 1) \cdot t - 1, y_1 + q \cdot t - 1\}, \]

\[ \Omega_{(s,t)} = \sum_{x_1=1}^{m-s+1-1} \sum_{y_1=1}^{m-t+1-1} p \cdot q \]

\[ = \sum_{x_1=1}^{m-s+1-1} \sum_{y_1=1}^{m-t+1-1} \left[ \frac{m-x_1+1}{s} \right] \left[ \frac{m-y_1+1}{t} \right] \]

\[ = \sum_{x_1=1}^{m-s+1-1} \left[ \frac{m-x_1+1}{s} \right] \cdot \sum_{y_1=1}^{m-t+1-1} \left[ \frac{m-y_1+1}{t} \right] \]

\[ = \left( \left[ \frac{m}{s} \right] + \left[ \frac{m-1}{s} \right] + \cdots + \left[ \frac{s+1}{s} \right] + 1 \right) \cdot \left( \left[ \frac{m}{t} \right] + \left[ \frac{m-1}{t} \right] + \cdots + \left[ \frac{t+1}{t} \right] + 1 \right) \]
Table 3-2: Calculation of Haar features

As the chart shown above there are over 160,000 square features on an image with 24*24 pixel resolution [26].

In order to calculate Haar feature value quickly, we are using integral image.

Let $i(x, y)$ represent gray intensity of the original image at $(x, y)$, and $ii(x, y)$ represent gray intensity of the integral image at $(x, y)$, satisfy

$$ii(x, y) = \sum_{x'\leq x, y'\leq y} i(x', y')$$

Using integral image, we only need to scan the original image once and we are able to calculate the gray value in any rectangle of the image with constant cost of time.

### 3.4.2 AdaBoost Training Process
AdaBoost algorithm is using decision tree to cascade large number of weak classifiers. A weak classifier in this application is to see if a Haar feature value satisfies the condition of human eyes. Obviously, one weak classifier is not going to come out with a good enough result. If we combine a large number of weak classifiers, it might lead to a good result [27].

Every Haar feature can be considered as a weak classifier, however not all of them are useful in determine a human’s face [28]. The key work of AdaBoost algorithm is to find out more useful Haar features.

The training process of AdaBoost algorithm is:

Given example images \((x_i, y_i)\), \(\ldots\), \((x_n, y_n)\) where \(y_i = 0, 1\) for negative and positive examples respectively.

Initialize weights \(w_{i,1} = \frac{1}{2m}, \frac{1}{2l}\) for \(y_i = 0, 1\) respectively, where \(m\) and \(l\) are the number of negatives and positives.

\[\text{Fort}=1, \ldots, T\]

1. Normalize the weight

\[w_{i,j} \leftarrow \frac{w_{i,j}}{\sum_{j=1}^{n} w_{i,j}}\]

so that \(w_i\) is a probability distribution.

2. For each feature, \(j\), train a classifier \(h_j\) which is restricted to using a single feature. The error is evaluated with respect to \(w_i\).

\[e_j = \sum_{i} w_{i,j} |h_j(x_i) - y_i|\]

3. Choose the classifier, with the lowest error.
4. Update the weight

\[ w_{i+1} = w_i \beta_i^{1 - e_i} \]

Where \( e_i = 0 \) if example \( x_i \) is classified correctly, \( e_i = 1 \) otherwise and

\[ \beta_i = \frac{e_i}{1 - e_i} \]

The final strong classifier is

\[ h(x) = \left\{ \begin{array}{ll} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{array} \right. \]

Where \( \alpha_i = \log \frac{1}{\beta_i} \)

Each weak classifier is expressed as:

\[ h_j(x) = \left\{ \begin{array}{ll} 1 & \text{if } p_j f_j(x) \leq p_j \beta_j \\ 0 & \text{otherwise} \end{array} \right. \]

\( x \) is a sample image. \( f(x) \) represents the feature value. \( p \) controls the inequality.

All the samples have to find out the best threshold. Firstly, we have to calculate its feature value. Sort the feature value of each sample in ascending order. Find the feature value has the smallest classification error:

\[ \varepsilon = \min(S^+ + (T^- - S^-), S^- + (T^+ - S^+)) \]

Sum of all eye sample weights: \( T^+ \)

Sum of all no eye sample weights: \( T^- \)

Sum of all eye sample weights before feature value: \( S^+ \)

Sum of all no eye sample weights before feature value: \( S^- \)
Suppose we have five samples and in the first loop, the calculation of best weak classifier is showed in the following table:

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>f</th>
<th>W</th>
<th>T+</th>
<th>T-</th>
<th>S+</th>
<th>S-</th>
<th>A</th>
<th>B</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>-1</td>
<td>2</td>
<td>1/5</td>
<td>3/5</td>
<td>2/5</td>
<td>0</td>
<td>0</td>
<td>2/5</td>
<td>3/5</td>
<td>2/5</td>
</tr>
<tr>
<td>X2</td>
<td>-1</td>
<td>3</td>
<td>1/5</td>
<td>3/5</td>
<td>2/5</td>
<td>0</td>
<td>1/5</td>
<td>1/5</td>
<td>4/5</td>
<td>1/5</td>
</tr>
<tr>
<td>X3</td>
<td>1</td>
<td>5</td>
<td>1/5</td>
<td>3/5</td>
<td>2/5</td>
<td>0</td>
<td>2/5</td>
<td>0</td>
<td>5/5</td>
<td>0</td>
</tr>
<tr>
<td>X4</td>
<td>1</td>
<td>7</td>
<td>1/5</td>
<td>3/5</td>
<td>2/5</td>
<td>1/5</td>
<td>2/5</td>
<td>1/5</td>
<td>4/5</td>
<td>1/5</td>
</tr>
<tr>
<td>X5</td>
<td>1</td>
<td>8</td>
<td>1/5</td>
<td>3/5</td>
<td>2/5</td>
<td>2/5</td>
<td>2/5</td>
<td>2/5</td>
<td>3/5</td>
<td>2/5</td>
</tr>
</tbody>
</table>

*Table 3-3: Threshold calculation of a weak classifier*

\[
A = S^+ + (T^- - S^-) \\
B = S^- + (T^- - S^-) \\
E = \min(A, B)
\]

Threshold feature value is 5 on this Haar feature.

The strong classifier is to let all the weak classifiers to vote on an image. And get the weighted summation on error rate of each weak classifier.

In a 24 * 24 image, there are over 160,000 Haar features, the training time and space are going to be particularly large. To increase the training speed, there are many ways for example:

1. Decrease Haar features, we might not need features on the side of the image or with only several pixels.

2. In the training process dynamically reduce the rectangle features that are not working well in making decision.
Open Cv comes with an eye detection training result which can be used in human face detection. I have compared Open CV and android API, it comes out that android API runs faster in eye detection.
4. Image Preprocessing

After we use face detection algorithm and find out the human face on the target image, we have to do image preprocessing before we do face recognition. Image preprocessing is a very important part of work. It is able to minimize the size of our image, decrease noise and enlarge image contrast.

Before we do image processing, implement Open CV (Open Source Computer Vision Library) to our application. Open CV is a library of programming functions mainly aimed at real-time computer vision, developed by Intel, and now supported by Willow Garage and Itseez. It is free for use under the open source BSD license. The library is cross-platform. It not only supports Android but also support IOS, Windows and Linux.

Open CV has three important properties [29][30]:

1. Open CV has included more than 300 cross-platform C functions. They are high-level APIs. It does not have to depend on other external libraries - although you can use some external libraries.

2. Open CV is free for all non-commercial and commercial applications.

3. Open CV can be used not only in face recognition but also human-computer interaction, object recognition, image segmentation, motion recognition, motion tracking, and robotics.

We can download the latest version of Open CV from the official site http://sourceforge.net/projects/opencvlibrary/files/. After we have the Open CV downloaded, we just need to add it to Android dependencies [31][32].
The Open CV version we use here is OpenCV-2.3.1.

In the generation and transmission of human face images, due to variety of reasons, that the image becomes deterioration or degradation, which makes the image different from the original scene. Therefore, it is necessary to do some face image processing so as to improve the visual effect of the image.

4.1 Unify the Size

Using AdaBoost eye detection, we are able to find out two things:

1. Is there are human face on the target image?

2. How many human faces are on the target image?

When there is more than one human face on the image, the application will request the user to take another picture. The outputs of face detection are face number, eye distance on each face and middle coordinate of two eyes. This process is combined with the results of AdaBoost detection which is shown on the picture below:
There are two outputs in a detection process: eye distance and middle coordinate of two eyes. Use these outputs to cut the face image from the original image. In order to minimize the image and focus on the detail of human face, I unified all the testing images into 92*112 which is the size of images in ORL face database. The resize process is using bilinear interpolation. Bilinear interpolation has the ability of anti-aliasing. It is one of the most simple and common image scaling methods [33][34].
Figure 4-2: Dimensional display of gray value in bilinear interpolation

An example of bilinear interpolation is shown above, the gray values on P can be calculated from its nearest four points A, B, C and D.

\[ A = (x_1, y_2) \]
\[ B = (x_2, y_2) \]
\[ C = (x_1, y_1) \]
\[ D = (x_2, y_1) \]

Linear interpolation in the x-direction:

\[ g(Q) \approx \frac{x_2-x}{x_2-x_1} g(A) + \frac{x-x_1}{x_2-x_1} g(B) \]
\[ g(R) \approx \frac{x_2-x}{x_2-x_1} g(C) + \frac{x-x_1}{x_2-x_1} g(D) \]

Linear interpolation in the y-direction:

\[ g(P) \approx \frac{y_2-y}{y_2-y_1} g(Q) + \frac{y-y_1}{y_2-y_1} g(R) \]

Bilinear interpolation is more complex than nearest neighborhood interpolation and linear interpolation but it does not need too much computing power \[35]\[36].
4.2 Grayscale

Face recognition technology generally take grayscale images as the research object. Color image contains too much useless information [37].

To change color image to gray, usually use the following formula:

\[ Gray = 0.39R + 0.50G + 0.11B \]

![Figure 4-3: Color image gray-scale transformation](image)

There are two ways to transform the color image to grayscale. The first one is to do it using Java by ourselves. The code is shown below:

```java
int color = pix_result[width * i + j];
int red = Color.red(color);
int green = Color.green(color);
int blue = Color.blue(color);

color = (int)((float)red*0.3 + (float)green*0.59 + (float)blue*0.11) & 0xFF;
```

However, we can also use the Open CV function, which is easier:

```java
Bitmap result = forProcess.copy(Bitmap.Config.ARGB_8888, true);
Mat gray = new Mat(),
rgba = new Mat();
```
43

4.3 Histogram Equalization

Histogram equalization is a method in image processing of contrast adjustment using the image’s histogram [38].

This method usually increases the global contrast of many images, especially when the usable data of the image is represented by close contrast values. Through this adjustment, the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values.

Histogram equalization not only enhances the image contrast but also reduce the interference of the image pipeline. It makes the image feature more easily extracted. We are running histogram equalization on grayscale images so that there is only one index on each pixel.

Histogram equalization is an effective method for image enhancement.

Use a variable R to represent the gray level of the image. Suppose R is normalized to the interval [0, 1] R = 0 represents black, R = 1 represents white.

We build transformation:

\[ S = T(R) \quad 0 \leq R \leq 1 \]

In the input image, the value of R for each pixel to produce a gray scale value S. \( T(R) \) has to satisfy the following requirements:
(1) $T(R)$ Monotonic function within $0 \leq R \leq 1$

(2) $0 \leq R \leq 1 \quad 0 \leq T(R) \leq 1$

Inverse transform can be expressed as:

$$R = T^{-1}(S) \quad 0 \leq S \leq 1$$

![Figure 4-4: Histogram equalization](image)

The figure below is an example of histogram equalization on a face image:

![Figure 4-5: Example of Histogram equalization](image)

The main code in Java is shown below:

```java
Gray=PixelsGray[i];

FerqueceGray[Gray]++;

SumGray[i]=SumGray[i-1]+FerqueceGray[i];

SumGray[i]=(int)(SumGray[i]*255/length);

PixelsGray[k] = SumGray[PixelsGray[k]];

return PixelsGray;
```

However, since we have imported Open CV in our application, we can simply using Open CV code to do histogram equalization, this code is shown below:
Mat mGrayhis=new Mat();

Imgproc.equalizeHist(gray, mGrayhis);

Imgproc.cvtColor(mGrayhis,rgba,Imgproc.COLOR_GRAY2RGBA,4);

Utils.matToBitmap(rgba, result);

### 4.4 Image Smoothing

If we take a picture by the android device, it is inevitable that there is noise on our face image. In order to decrease noise, there are basically two things we can do:

The first one is to do global process including the Wiener filter, Kal-man filter and some other methods[39].

The second one is to use local operators, such as mean filter, median filter, gradient inverse weighted filter and other classic algorithms. In this application, we use mean filter to smooth the face image.

Mean filter is to use the mean value of the local neighborhood to represent every pixel.

\[
 h(i, j) = \frac{1}{m} \sum_{(k,l) \in N} f(k,l)
\]

M is the number of pixel in the local neighborhood. For example in the 3 times 3 neighborhood of pixel point \((i,j)\), we have the formula:

\[
 h(i, j) = \frac{1}{9} \sum_{k=i-1}^{i+1} \sum_{l=j-1}^{j+1} f(k,l)
\]
The size of the neighborhood is going to control the filter level. We select the following filter weights which has one peak and symmetric in horizontal and vertical directions.

\[
\begin{pmatrix}
1 & 1 & 1 \\
16 & 8 & 16 \\
1 & 1 & 1 \\
8 & 4 & 8 \\
16 & 8 & 16 \\
\end{pmatrix}
\]

Core code is as follow:

\[
\text{average}=(1/4*\text{data}[i*width+j]+ \\
1/8*\text{data}[i*width+j-1]+ 1/8*\text{data}[i*width+j+1]+ \\
1/8*\text{data}[(i-1)*width+j]+ 1/16*\text{data}[(i-1)*width+j-1]+ \\
1/16*\text{data}[(i-1)*width+j+1]+ 1/8*\text{data}[(i+1)*width+j]+ \\
1/16*\text{data}[(i+1)*width+j-1]+ 1/16*\text{data}[(i+1)*width+j+1])/9;
\]

\[
\text{filterData}[i*width+j]=(\text{int})(\text{average})
\]

\[
\text{filterData}[i]=(\text{filterData}[i]-\text{min})*255/(\text{max}-\text{min});
\]

The process above helps to reduce the size of an ordinary image from 25 kb into less than 10 kb. Especially the image unified part, it cuts the original image obtained from the camera from over 70 kb into 10 kb. The face image size compress from 720*960 to 92*120.
5. **PCA Algorithm**

The Principal Component Analysis (PCA) is one of the most successful techniques that have been used in image recognition and compression. PCA is a statistical method under the broad title of factor analysis. The purpose of PCA is to reduce the large dimensionality of the data space (observed variables) to the smaller intrinsic dimensionality of feature space (independent variables), which are needed to describe the data economically. This is the case when there is a strong correlation between observed variables.

The jobs that PCA can do are prediction, redundancy removal, feature extraction, data compression, etc. [40]. PCA is a classical technique in the linear domain which is working on signal processing, image processing, system and control theory, communications, etc.

Face recognition has many applicable areas. Moreover, it can be categorized into face identification, face classification, or sex determination. The most useful applications contain crowd surveillance, video content indexing, personal identification, mug shots matching, entrance security, etc. The main idea of using PCA for face recognition is to express the large 1-D vector of pixels constructed from 2-D facial image into the compact principal components of the feature space. This can be called eigenspace projection. Eigenspace is calculated by identifying the eigenvectors of the covariance matrix derived from a set of facial images [41][42].
5.1 Training

In the training phase, suppose we have $M$ training images of size $N$ (rows of image times columns of image). Convert $M$ images into a 1-D vector by concatenating each row into a long thin vector. $p_j$ represents the pixel value $[43]$.

$$x_i = [p_1 \ldots p_N]^T, i = 1, \ldots, M$$

All the training images are in the same size, we can then calculate the mean image from each image vector. Let $m$ represent the mean image.

$$m = \frac{1}{M} \sum_{i=1}^{M} x_i$$

And let $w_i$ defined as subtract the mean image from the original images

$$w_i = x_i - m$$

Next step, we have to construct the covariance matrix of $w_i$

$$C = \frac{1}{M} \sum_{i=1}^{M} w_i w_i^T = \frac{1}{M} WW^T \quad C \in R^{N \times N}$$

After that, we have to calculate the eigenvalues and eigenvectors of the covariance matrix. $N$ is the size of each image, it is usually a very large number.

According to Singular Value Decomposition we can calculate the eigenvalues and eigenvectors of $W^T W$ instead of $WW^T$.

$$W^T W \in R^{M \times M}$$

We will obtain all non-zero eigenvalues $[\lambda_1, \lambda_2, \ldots, \lambda_r]$ $\quad (1 \leq r \leq M)$. These eigenvalues are sorted in descending order. The corresponding eigenvectors constitute a feature space $U = [u_0, u_1, \ldots, u_{r-1}] \in R^{N \times r}$. Map each of the images in training group to the feature space:
\[ Y = U^T \ast X \in R^{r \times 1} \]

\( Y \) is the coordinate index in space \( U \).

**5.2. Recognition**

In the recognition phase (or, testing phase), take the picture of the target person and convert into a 1-D vector by concatenating each row into a long thin vector. Map this image into the feature space. The coordinate index is also called feature vector which represents the coordinate of the sample image on the feature space. Feature space is also known as a less dimensional space which expressed the most signal of the training group.

Once an image is projected into a feature space, we have to determine the similarity of these images. What I am using here is Euclidean distance. \( x \) and \( y \) are the coordinates of two face image on the less-dimensional space [44].

\[
\text{Eu}(x, y) = \sum_{i=1}^{k} (x_i - y_i)^2
\]

We can find out if the target person is in the database by compare the Euclidean distances. Moreover, different strategies can be used to deal with false detection.

Schematic diagram of the face recognition system that will be implemented is shown in Figure 5-1.
5.3 Using Open CV in PCA algorithm

We have already used Open CV in image preprocessing, now we are using Open CV in PCA algorithm.

As we discussed before, PCA algorithm has some important factors to implement: image data, eigenvectors, covariance matrix, eigenspace projection[45][46]

The core codes are shown below:

\[
\text{org.opencv.core.Core.PCACompute(TrainImg, avg, eigenvectors);}
\]

\[
\text{org.opencv.core.Core.PCAPrjECT(TrainImg, avg, eigenvectors, result);}\]

\[
\text{org.opencv.core.Core.PCAPrjECT( TestImg, avg, eigenvectors,result2);}\]
The figure 5-2 shows how we use Open CV to do PCA algorithm in Android:

![Diagram](image)

**Figure 5-2: PCA algorithm in Android**

### 5.4 PCA Algorithm Principle

The covariance matrix of the training images represents the variance between dimensions.

The eigenvectors corresponding to nonzero eigenvalues of the covariance matrix produce an orthonormal basis for the subspace within which most image data can be represented with a small amount of error. The eigenvectors are sorted from high to
low according to their corresponding eigenvalues. The eigenvector associated with the largest eigenvalue is one that reflects the greatest variance in the image. That is, the smallest eigenvalue is associated with the eigenvector that finds the least variance.

The dimensions of a feature vector consistent with the original image, it can be seen as an image. Therefore, these vectors are called eigenface. They represent the difference between the image and the mean image in different directions. Select the larger eigenvalues and use the corresponding eigenvectors to constitute a feature space. The coordinate index in feature space represents the distance of each image to the less-dimensional space.

The greater the variance, the more information it contains. We can explain the theory in the following way:

In signal processing theory, signal has large variance and noise has smaller variance.

The figure 5-3 below has five samples and their mean value on each dimension is 0.
Figure 5-3: Five samples

Now we would like to map these five samples into one plane on figure 5-4:

Figure 5-4: Two projections

According to the signal processing theory, the projection on the left plane has maximum variance between sample points which makes it a better projection.

If we have \( x_1, x_2, \ldots, x_n \) samples and we would like to map to a less-dimensional plane, the coordinates on plane \( u \) are \( x^{(i)} u \) or \( u^T x^{(i)} \). It shows on figure 5-5:

Figure 5-5: map to a less-dimensional plane
Since the projected mean is 0, the variance is:

\[
\frac{1}{m} \sum_{i=1}^{m} (x^{(i)}^T u)^2 = \frac{1}{m} \sum_{i=1}^{m} u^T x^{(i)} x^{(i)^T} u = u^T (\frac{1}{m} \sum_{i=1}^{m} x^{(i)} x^{(i)^T}) u
\]

\[
\Sigma = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} x^{(i)^T})
\]

is the covariance matrix of the original samples:

\[
\lambda = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} u)^2
\]

\[
\lambda = u^T \Sigma u
\]

\(u\) is a unit vector which makes it \(u \lambda = uu^T \Sigma u = \Sigma u\).

\(\lambda\) is the eigenvalue of covariance matrix. \(u\) is the eigenvector of covariance matrix. The derivation above shows that the corresponding eigenvector of the largest eigenvalue of covariance matrix is the plane that contains the most information of the original samples. Therefore, we can find out the plane which has more information by using PCA algorithm.

Using 200 face images in ORL face database, we find out that the eigenvalues decrease exponentially. The sum of top 71 eigenvalues is over 90% of the total sum. In that case, we can decrease the dimensions to 71 and still keep 90% of the total information. The original dimensions of each image are 10304 [47].

Since the application is working in mobile device, we might not need too many images in the training group. When the subspace dimension is under a certain number, it has a positive correlation with the recognition rate. On the other hand, if the training group is too big, smaller eigenvalue might provide useless information. In this application I have tested the relationship between the recognition rate and subspace dimensions.
6. Results and Analysis

I am using the ORL face database for the testing of the application. ORL face database was made by Olivetti Laboratory in Cambridge, UK from April 1992 to April 1994. It has face images of 40 people of different age and genders and races. There are 10 grayscale images of each person and different facial expressions were made in these images. The rotation is up to 20 degrees and face size up to 10% variation. The same person in these images can take off his glasses or even close their eyes. These images are all in grayscale and in the size 92*112 and they all been normalized. We don’t have to run the standardize code on these. The version of these images is BMP, which can be used directly by android [48].

I have taken different tests. These tests are done in Samsung I9308 mobile phone. The CPU model is Samsung Exynos 4412 and Quad-Core with 1433MHz frequency. The RAM capacity is 1 GB and ROM capacity is 16GB.

6.1 Test 1

I am taking 10 images of 10 different people in the training database [49]. The training matrix is 10*10304. The subspace dimension number is 10. The embedded system has to identify 10 images at once and it takes less than 3 seconds. There are totally 100 images to recognize. 85 images correctly identified. 15 images failed. The recognition rate is 85%.

There are two reasons for the 15% error:

1) glasses

2) rotation of the face
One very obvious example is shown figure 6-1:

![Figure 6-1: Failed sample](image)

There are 10 images of the same person, if we set image 1 in the database, it will only recognize image 5 and 6. However if we take image 3 in the database, the application will only not been able to recognize image 1 and 6. It seems that glasses does matter.

### 6.2 Test 2

I am taking 20 images of 20 different people in the training database. The training matrix is 20*10304. The subspace dimension number is 20. The embedded system has to identify 10 images at once and it still takes less than 3 seconds. There are totally 200 images to recognize. 165 images correctly identified. 35 images failed. The recognition rate is 82.5%. Some testing images correctly identified in test 1 but failed in test 2. Human faces with glasses or big rotation are the main reasons that failed. I decrease the less-dimension space number from 20 to 15 and the recognition rate decrease to 80%. It shows that the increase of subspace dimension from 10 to 20 is not the reason that the recognition rate decrease.

### 6.3 Strategies

Assuming that we are using this application to protect the information in our mobile phone, it is important to increase the images in the training set. However, this
is only going to help decrease the rate that someone instead of the owner unlocked the information. There is still large chance that the owner unable to unlock the information using his face. There are two tests might be useful.

Firstly, we take 10 different people in the training group, each person has three different images which make it 30 images. Then the program will calculate the distance between the target image and the eigenface. In this way, we have one target person and three different face images of this person in the training group.

1) If more than two out of the closest three pictures point to the same and correct person, then the user can unlock the system.

2) If the average distance between the testing image and three different images of the person is the closest then unlock the system.

In the first test, we have a training group with 30 images of 10 people, each of them has three pictures. There are 100 testing images and 7 of them came out wrong in the first strategy. The recognition rate is 93%.

In the second test, we have a training group with 30 images of 10 people, each of them has three different pictures. There are 100 testing images to test. For each images, there are 30 results represent the distance difference between every images of the training group. Since every three images represent a potential person, we calculate the distance between the testing person and every people in the database and the closest distance represent the result. In 100 testing images, only 1 of them came out wrong. The recognition rate is 99%.
The figure 6-2 shows how the application finds out the target person within 30 images database.

![Figure 6-2: Test result](image)

Here we comes a question, in some situation, we are not sure if the target person is in our database. If we are using the second strategy, there must be a result no matter the target person is in our database or not. To avoid this type of mistake, we come out a new strategy.

3) If we combine the first and second strategy together, we can greatly avoid all type of mistakes.
Figure 6-3: Training images

Figure 6-4: Person A (left)  Person C (right)

Figure 6-5: Person B in the training group

Figure 6-6: Person D in the training group
We are taking person A’s image as the testing image. By using the second method, the shortest distance points to the person B in the training group. We assume that person A is person B. By using the first method, we get to know the three shortest distances. If they all point to person B, we are sure that person A is person B. If two out of three data point to person B, the system will going to ask more information. If less or equal to one data point to person B, it is almost sure that person B is not person A.

We are taking person C’s image as testing image. By using the second method, the shortest distance points to the person D. However, person C is not in our training group. By using the first method, we find out that the two out of three of his most likely results are not pointing to person D. We can almost sure that he is not in our database.

Moreover, since the images in ORL face database have already been preprocessed, if we are doing the preprocessing by the application, the face detect part is going to filtered some poor quality face images. It will also help to increase the recognition accuracy.
6.4 Dimension Number and Recognition Rate

In order to find out the relationship between subspace dimension number and the recognition rate, I have done some testing using ORL face image database on embedded system[50].

Choose 5 images of each person in ORL database and compose a training group with 200 face images and use other 80 images as testing sample. A target image is correctly recognized only if the average Euclidean distance is the smallest. Different number of feature dimensions and their corresponding results are shown table 6-1:

(d represent to the subspace dimension and r represent to the recognition rate)

<table>
<thead>
<tr>
<th>d</th>
<th>1</th>
<th>10</th>
<th>30</th>
<th>50</th>
<th>70</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>0.35</td>
<td>0.75</td>
<td>0.78</td>
<td>0.88</td>
<td>0.95</td>
<td>0.95</td>
</tr>
</tbody>
</table>

*Table 6-1: Different number of feature dimensions and their corresponding result*

As we have learned before, the eigenvalue decrease exponentially which means that the information in each eigenvector decrease exponentially. This test shows that the recognition rate does not increase obviously when the subspace dimension is over 50. The image below shows the first, second, tenth, fiftieth, seventieth eigenface and the mean image.
This image shows that the information contained in each eigenface decrease with the decrease of eigenvalue. It is hardly to see the difference between the fiftieth and seventieth eigenface.

According to Analysis of PCA-based Face Recognition Algorithms: Eigenvector Selection and Distance Measures from Computer Science Department of Colorado State University not include the top three eigenvalue will increase the recognition rate. I tried to exclude the top three eigenvectors and found out the recognition rate decreased in a 30 images training group. According to this paper, Wendy S. Yambor believes that the first eigenvector represents the light from right to left and the second and third eigenvector represent the light from top to the bottom. Since the images in ORL database are all done in room and already been standardized, it is unnecessary to exclude the top eigenvectors in this application [51].
7. Conclusion and Future Work

7.1 Conclusion

What I have been doing is to build up an application which is able to do face recognition on embedded system especially on Android environment. I have chosen the combination of AdaBoost and PCA algorithm.

During the development, I have been working on face detection algorithm, preprocessing methods and face recognition. I have done testing on face detection speed and ability of preprocessing. There are also testing on the selection of subspace dimensions. It has been proved that the recognition rate is over 90% using proper strategy. The recognition speed is under 2 seconds once the target image is taken. The process can be named as face verification. This is used mostly in security verification.

This application can be used as an individual program but can also be used as a part of other application. It does not need to be limited into a security type of program.

I have developed a notebook program with face unlock feature. It has been downloaded more than 100 times in the last month. Face unlock feature becomes the most interesting feature of this application.

The figure 7-1 shows the resent download times within the last month. It shows that there is huge market potential for this type of application.
Figure 7-1: Recent free download times from the internet

7.2 Future Work

Future work of this application might be focus on the speed and recognition rate. If we would like to develop an embedded system application for example a one-to-many matching program. It can be used as registration of a class or punched-card machine. We might focus on different other ways of preprocessing so as to speed up the process. The training image right now provides a covariance matrix of $10304 \times 10304$. It is possible to reduce the size of each image using different method of image scaling. Optimize the Android application structure and reduce the use of memory and objects. Create face recognition database. Create different processes and procedures based on different situation. There should be more testing on face recognition under different environments such as light and face offset.
Appendix A:

Notebook Application Screenshots

Note edition page

Main menu
Face login and set up menu

Face capture

1. Take a picture
2. Rotate the image
3. Save into database
Preprocessing result and save menu

1. Take a picture
2. Plot the image
3. Save into database

set up
quit
Appendix B:

Notebook Application Introduction

It is a notebook application on Android that each note can be protected by both password and user face. On the main menu, it gives options to go to password management, face management, about, refresh, new note and exit. Once a new note is created, the application goes to the note edit interface. On the note edit interface, user can press the home bottom and choose to save it or protect it with user face or password. Once a note is protected with password or face, the user has to unlock it with correct password or face. All the notes are saved in SQLite database. User face is firstly saved into the SD card, once a preprocessing is finished; the standardized face image is saved into the training group and SD card information is going to be deleted. In order to protect the information with user face, he has to firstly set up the user face. The training images are from ORL database and stored in the application. It is available on http://as.baidu.com/a/item?docid=3308372&pre=web_am_se.
References


http://blog.csdn.net/abcmx/article/details/8632636


http://docs.opencv.org/platforms/android/refman.html
[31] Android development with OpenCV [Online]
http://docs.opencv.org/doc/tutorials/introduction/android_binary_package/dev_with_OCV_on_Android.html


[33] Bilinear interpolation Wikipedia [Online]
http://en.wikipedia.org/wiki/Bilinear_interpolation


[38] Histogram equalization Wikipedia [Online]
http://en.wikipedia.org/wiki/Histogram_equalization


[45] Face Recognition using OpenCV [Online]
http://whbzju.github.io/blog/2013/07/28/renlianshibiezongjie/

[46] PCA in OpenCV [Online]
http://blog.csdn.net/jiangnanmeiyi/article/details/4870007

[47] Significance of eigenvectors [Online]
http://blog.sina.com.cn/s/blog_49a1f42e0100fvdu.html

[48] ORL face database [Online]
http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

