CS685 Image Understanding

Final Project

Fast Face Detection based on Eigenfaces with small training data set

Xin Ye

P001248758

Abstract

This project designs a fast face detection method based on Eigenfaces with small training data set. The project focuses on fast face detection with the assumption that the images to be detected has good brightness condition and the human faces are all frontal faces if there are faces in the images, and the images' size are neither too big nor too small. The project detects faces in 2-D images.

The project firstly trains a small data set of training faces images to obtain an eigenfaces subspace, and then finds the potential area of faces by searching ROI (Region Of Interest) in the binary image obtained from the input color image, that is firstly converts the color image to binary image and finds the human skin area by a HSV threshold, and then groups the image to find useful objects, and then detects faces in successive small ROI windows based on the Euclidean distance between the ROI and the trained eigenfaces subspace.

.

Introduction

The face detection is always one of the most important issues in computer vision. Face detection is also the first step in face recognition. This project focuses on finding a fast face detection method in some given conditions that could be used in detecting human faces from a 2-D image with just a small training data set. The given conditions might include good brightness condition, suitable size of image, all frontal faces, well dressing people with normal facial expression, and so on. Because in some situation we don’t need to find all the faces in all kinds of images, we might just need to find a specific group of people or even a specific person from some pictures taken in good conditions, thus this project focuses on finding such a method for fast training and fast detecting with a small training data set from pictures taken in good conditions.

Background and Motivation

Fast face detection is useful in face recognition. If we could detect a face fast, then we may recognize a face fast. And it is better if we could recognize a face at the same time when we are detecting the face. Thus we might consider using the same method in doing face detection and face recognition. Then the eigenfaces method is a

good choice for this purpose.

The eigenfaces method is the first method applied in face detection and face recognition using PCA approach [1], [2], after PCA was firstly used in representing human face features [3], [4]. Even though the eigenfaces method is old, it is still an important and effective method in face detection and recognition [5]. It is a linear projection method as Fisherface [6], and it is also reported to outperform the LDA method when the number of training samples is small [7]. And if the training data set is small, then the training will be fast. As well the detection will be fast. Thus the eigenfaces method is a good choice in small training data set. And it also could be used for future recognition purpose.

For the fast detection purpose, we could assume some good conditions as said at the beginning of this chapter. Thus we could apply some specific methods to improve the efficiency of the detection. The approach this project use is: firstly converts the color image to a binary image by a HSV range threshold, and then dilates and groups the binary image to further find the potential facial area, and then uses successive small ROI (region of interest) windows to detect faces. This method could avoid searching the whole image with different size of windows, thus saves times and improves the efficiency of the detection.

The Technical Approach

This project designs a fast face detection method based on the eigenfaces approach, the project primarily contains two parts that are the training part and the detecting part as shown in Figure 1.

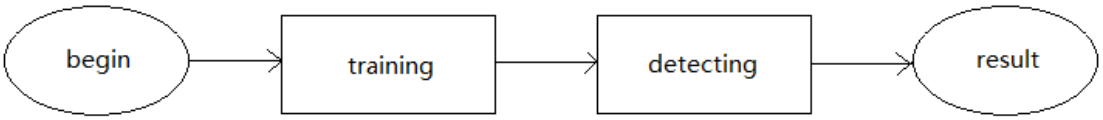


Figure 1

As the figure 1 shows, this project firstly trains the training faces, and then detects faces of the input image based on the trained eigenfaces subspace. The major structure and diagram of the training part and the detecting part is shown in Figure 2.

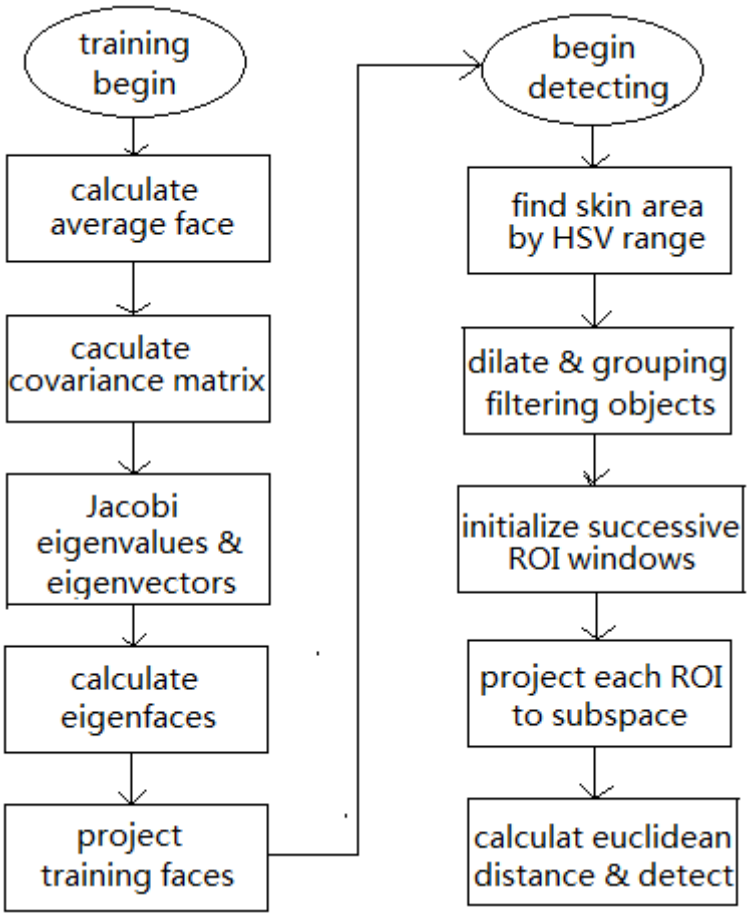


Figure 2

1. Training Approach

In the training part, twenty-three of face images are used in the testing of this project, each training face image is a grey level image with the width 92 and the height 112 and 8bit depth. The face images used as the training set are shown in the Figure 3. The reason of choosing these face images is that these images contain different types of frontal faces among which there men and woman, persons smile and persons don't smile. And eight of these face images are even taken from the same person that help improve the accuracy of the detection on this specific person, thus could help future recognition.

Figure 3

The summary procedure of the training part is:

1. Smooth each training image by a Gaussian filter;
2. Calculate the average face of all the training face images;
3. Calculate the covariance matrix of the training data set;
4. Calculate the eigenvalues and eigenvectors of the covariance matrix by Jacobi rotation method;
5. Calculate the eigenfaces subspace by the eigenvectors and normalize the eigenfaces by the eigenvalues;
6. Project each training face image onto the subspace to obtain the weights of each training image for the next step of detection.

1.1 Gaussian template

Rather than design a Gaussian filter, this project just designs a smoothing function gaussianfilter() using a 3 by 3 Gaussian template directly, with the standard deviation of 0.849.

1.2 Calculation of the average image

The calculation of the average face is by the formula shown in Figure 4. This project designs a subroutine function whose name is calculate_average_image() to calculate the average face.

$$\Psi = \frac{1}{M}\sum_{n=1}^{M} \Gamma_n$$

Figure 4

Where Ψ is the average face and Γn stands for a training face image. Each training face image could be seen as a vector. For

example, a training face image's size in this project is 92 by 112, and then this image could be seen as a vector of 10304. The 'M' in Figure 4 stands for the total number of training images, and in this project this number is 23 as said at previous part. The average face image of all the training face images shown in the Figure 3 is shown in the Figure 5.

Figure 5

1.3 Calculation of the covariance matrix

The calculation of covariance matrix is shown is Figure 6.

$$\Phi_i = \Gamma_i - \Psi$$

$$C = \frac{1}{M}\sum_{n=1}^{M} \Phi_n \Phi_n^T = AA^T$$

$$A = [\Phi_1\ \Phi_2\ \ldots\ \Phi_M]$$

Figure 6

Where Γi is one of the training face images seen as a vector, and **Ψ** is the average face image also seen as a vector, and **Φ**i is the difference between a train image and the average face, and A is called a difference matrix in this project. From Figure 6 we have seen that the covariance matrix **C** is a 10304 by 10304 matrix in

this project, and it is very uneconomic to calculate the eigenvectors and the eigenvalues of such a large matrix. Thus we could calculate the eigenvectors and the eigenvalues of **L** where **L** is shown in the Figure 7.

$$A^T A\mathbf{v}_i = \mu_i \mathbf{v}_i$$
$$AA^T A\mathbf{v}_i = \mu_i A\mathbf{v}_i$$
$$L = A^T A,$$

Figure 7

As shown in Figure 7, if we calculate the eigenvectors of **L**, then we could calculate the eigenvectors of **C**, and L is just a 23 by 23 matrix that is much easier to be handled. Thus this project designs a subroutine function calculate_covariance_matrix() to calculate **L**. But if we calculate the eigenvectors of L, we could only obtain 23 eigenfaces, and by calculating eigenvectors of C we could have 10304 eigenfaces. This means that much information will be lost. However, we just need the eigenfaces with the most eigenvalues to help detecting faces. Thus this method is effective and efficient.

1.4 Calculation of eigenvectors and eigenvalues of L (Jacobi)

The calculation of eigenvectors and eigenvalues of the symmetric matrix L is based on Jacobi rotation method. This project uses the Jacobi function from OpenCV. And this project rewrites the Jacobi function from OpenCV with the purpose of improving the efficiency and accuracy even though this project has not found anything to be

improved. However, let us have a look at the Jacobi rotation [8].

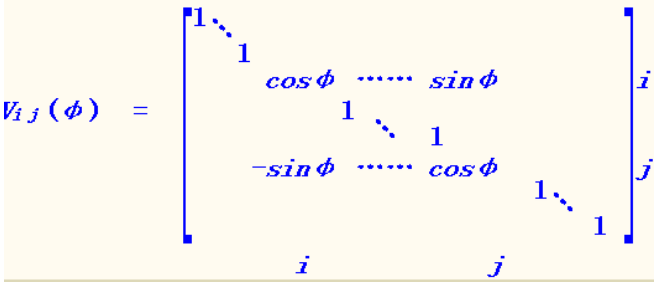


Figure 8

The matrix shown is Figure 8 is a basic Jacobi rotation. The basic rotation and the matrix after being applied by such a basic rotation is also could be seen in Figure 9.

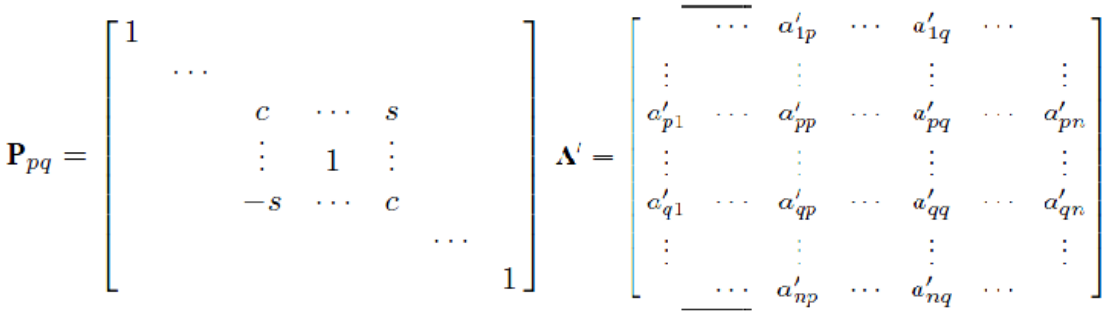


Figure 9

$$
\begin{aligned}
a'_{rp} &= ca_{rp} - sa_{rq} \\
a'_{rq} &= ca_{rq} + sa_{rp}
\end{aligned}
\qquad r \neq p,\ r \neq q
$$

$$a'_{pp} = c^2 a_{pp} + s^2 a_{qq} - 2sca_{pq}$$

$$a'_{qq} = s^2 a_{pp} + c^2 a_{qq} + 2sca_{pq}$$

$$a'_{pq} = (c^2 - s^2)a_{pq} + sc(a_{pp} - a_{qq})$$

$$\theta \equiv \cot 2\phi \equiv \frac{c^2 - s^2}{2sc} = \frac{a_{qq} - a_{pp}}{2a_{pq}}$$

$$a'_{pq} = 0$$

$$S' = S - 2|a_{pq}|^2$$

Figure 10

From Figure 10 we could see that choose the right θ we could let

a’pq and a’qp to 0, and the sum of the squares of the off-diagonal elements is convergent by Jacobi rotations. Thus by doing many times of Jacobi rotations, we could let the sum of the squares of the off-diagonal elements to become very small (tends to 0), then the diagonal elements will be the eigenvalues of the original matrix, and the eigenvectors of the original matrix is the columns of V where V = P1·P2·P3 ⋯ and Pi is a basic Jacobi rotation matrix.

This project rewrites the Jacobi subroutine from OpenCV that using successive threshold to apply Jacobi rotation. The output of the Jacobi subroutine is shown as Figure 11.

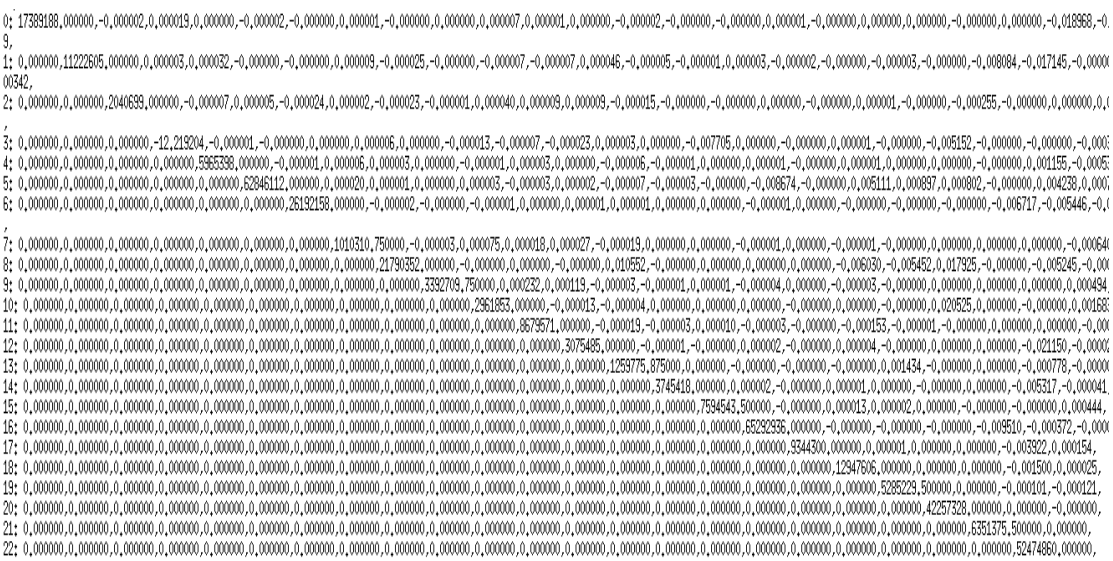


Figure 11

1.5 Calculation of Eigenfaces

As shown in Figure 7, if v is the eigenvector of L, then Av is the eigenvector of C. And the corresponding eigenvalue λ is the square of ||Av||. This project designs a subroutine calculate_eigenfaces() to calculate and normalize the eigenvectors of C, and these

eigenvectors of C are the eigenfaces we need. The eigenfaces of the training face images shown in Figure 3 are shown below in the Figure 12.

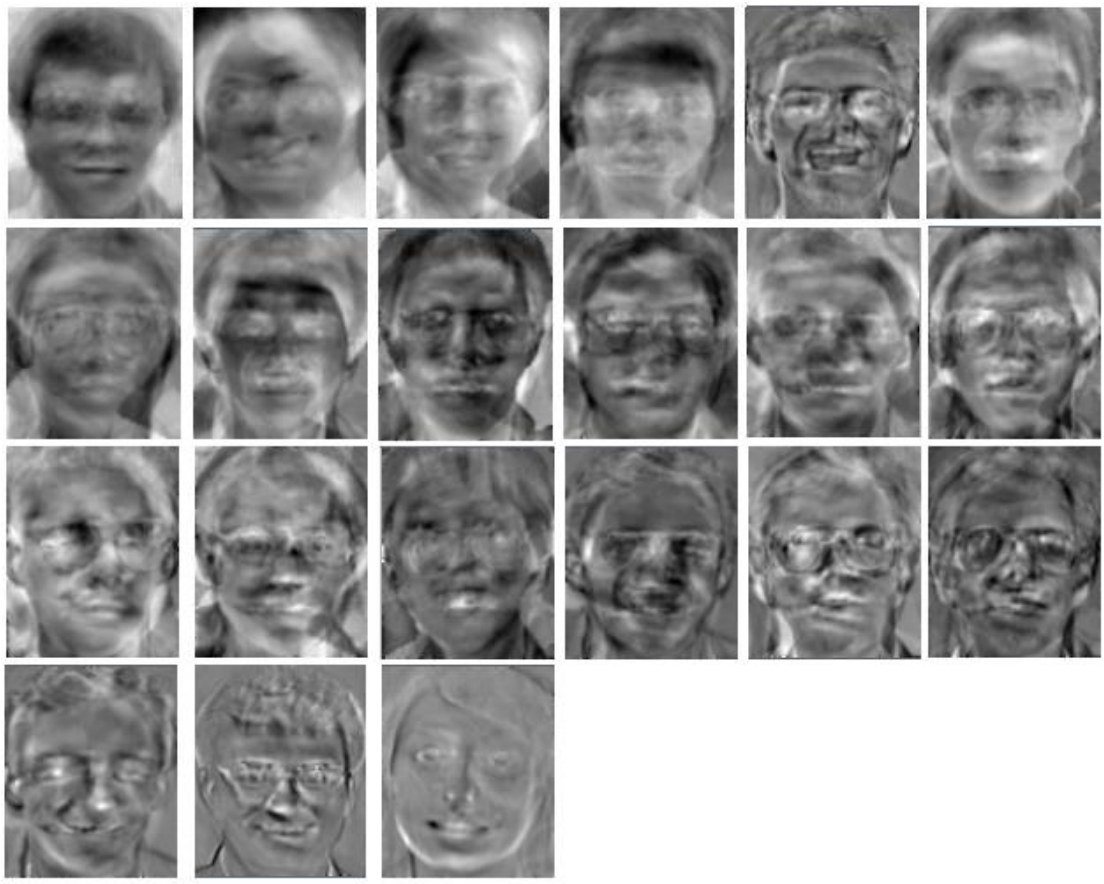


Figure 12

1.6 Project Training Faces

The projection of each training face to the eigenfaces subspace is based on the formula shown in Figure 13.

$$\omega_k = \mathbf{u}_k^T(\mathbf{\Gamma} - \mathbf{\Psi})$$

Figure 13

Where T is a training face image, ψ is the average face image, u_k is an eigenface, and ω_k is the projected weight of face image T on eigenface u_k.

This project designs a subroutine project_to_subspace() to project the training face images onto the eigenfaces subspace to obtain all the weights of each train image to all the eigenfaces.

1.7 What I have done VS what is in OpenCV about eigenfaces

1. There is a function cvCalcEigenObjects() in OpenCV that calculates eigenvectors of covariance matrix of input samples. However this function is a general purpose function based on PCA approach and is not a specific function in computer vision, thus the efficiency of this function is influenced by a lot of useless procedures, variables, and calculation steps. And using this function has to include another useless shared libraries cvaux.so which is large and will influence the compile efficiency. And the most important is that this function gives only the eigenvectors and eigenvalues as the output without other important things such as the difference matrix ϕ. Thus, I write every subroutine of the eigenfaces approach except the Jacobi function because the Jacobi function in OpenCV is efficient.

2. I write subroutines calculating the average face, covariance matrix, the difference matrix ϕ, and eigenfaces. Even though I rewrite the Jacobi function the same as the Jacobi function in OpenCV, I do not need to include cvaux.so which is a large and useless library, thus improve the efficiency of compiling.

3. The subroutines I wrote in this project give more efficiency than the single function in OpenCV. For example, this project set the difference matrix ϕ as an output when calculating the covariance matrix, thus ϕ could be used as an input in the calculation of eigenfaces, thus there is no need to calculate ϕ many times, and then improve the efficiency.

4. I designed subroutine calculating not only the Euclidean distances between an image and the training face images, but also the Euclidean distance between an image and the whole eigenfaces subspace. This helps greatly in detecting faces. And the calculation is done in the same procedure when calculating the projected weights, thus improves efficiency again.

2. Detecting Approach

The detecting part begins after the training part was done. As soon as the training procedure is finished, the output eigenfaces, the average face and the projected weights of training images to eigenfaces will be treated as the input variables in the detection subroutine.

The summary procedure of the detecting part is:

1. Detects the image's size and the number of channels. If the input image is with the same size as the training images, then projects the input image to eigenfaces subspace directly. Else if the input

image is a general size three channel color image, then applies further procedures and subroutines to find the potential face areas to be projected and detected.

2. In the case of dealing with a general size three channel color image, this project firstly converts the color image to a binary image for further processing, this project choose a HSV range as the threshold for the conversion.

3. Erodes the binary image after it is converted from a color image to separate objects which are possibly from different targets in the image. This project applies two times of erode operation with different types of structure elements.

4. Groups the binary image by 6-connectedness after doing the dilation. And then calculates the complexity of each object.

5. Filters out useless objects among which the area and complexity are very small.

6. Groups the processed binary image again and sets up successive small ROI windows for all the remaining objects by the size and the shape of the rectangle boundaries of these objects.

7. Gets a small grey level image from the original color image that this grey image is corresponding to the ROI window in the binary image, and then resize the small grey image as the training images.

8. Projects the grey ROI image to the eigenfaces subspaces to

obtain the weights, and then calculate the Euclidean distances between this ROI image and all the training images, and the Euclidean distance between this ROI image and the eigenfaces subspace.

9. Calculates the possibility of the existence of a face in this ROI image by the minimum and maximum Euclidean distance between this ROI image and the training images, and by the Euclidean distance between this ROI image and the eigenfaces subspace, and by the average of all the Euclidean distances between the ROI image and all the training images.

10. The final step is detecting a face by the final possibility of the existence of a face, and then output the result.

2.1 Threshold (HSV)

The threshold used in converting the color image to a binary image is very important. If we have a specific purpose of the detection, then we would have a specific range of objects, and then we would have specific assumptions, and then we would apply specific thresholds.

This project choose a HSV range as the threshold to convert the color image to a binary image at the assumption that we focus on detecting human faces and not animal or cartoon faces or faces with masks, and the brightness of the picture is well, and the picture

is taken in good condition that the faces are all frontal faces and people dress well with normal facial expression. Then we could assume a range of HSV values to be the potential area of human skins. This project assume the minimum hue is 0°, and the maximum hue is 40°, this range of hue implies write to yellow of human skins. And this project also assumes the minimum saturation is 30, and the maximum saturation is 150, and the minimum value is 80, and the maximum value is 255. However, if the input image could not satisfies the assumed conditions, then the output would be meaningless in further processing, an example could be seen as below in Figure 14.

Figure 14

A good and meaningful example of the input and the output of the conversion could be seen as below in Figure 15.

Figure 15

2.2 Erosion

Erosion is very useful in this project because it could help separate objects from the same connected region. Thus could help find the face region without the dealing with connected but useless components such as the clothes and hands. This project designs a subroutine erode() doing erosion operation with 6 different types of structure elements. And this project chooses two types of these structures elements to do two times of erosions. The type1 structure element could help separate objects greatly and the type6 structure element could further eliminate some noise without erode too much. The structure elements could be seen in Figure 16.

type1	type6
1, 1, 1	0, 1, 0
1, 1, 1	0, 1, 1
1, 1, 1	0, 0, 0

Figure 16

An example of the erosion could be seen as below in Figure 17. From the example we could see that after doing the erosion operation, the output binary image is more meaningful for the

further processing and detecting.

Figure 17

2.3 Panning

The panning procedure begins after the erosion operation. The purpose of this panning procedure is to filter out useless objects to improve the efficiency of further detection. This project designs three subroutines in doing the panning procedure. The first subroutine group_by_connectedness() groups the binary image by 6-connectedness. The subroutine calculate_complexity() calculates the complexity of each object in this binary image. And then the subroutine panning() filters out useless objects that the size and the complexity of these objects are very small. That is, the tiny size

objects and very simple objects are seen as useless and are filtered out. An example of the output of this procedure could be seen in Figure 18.

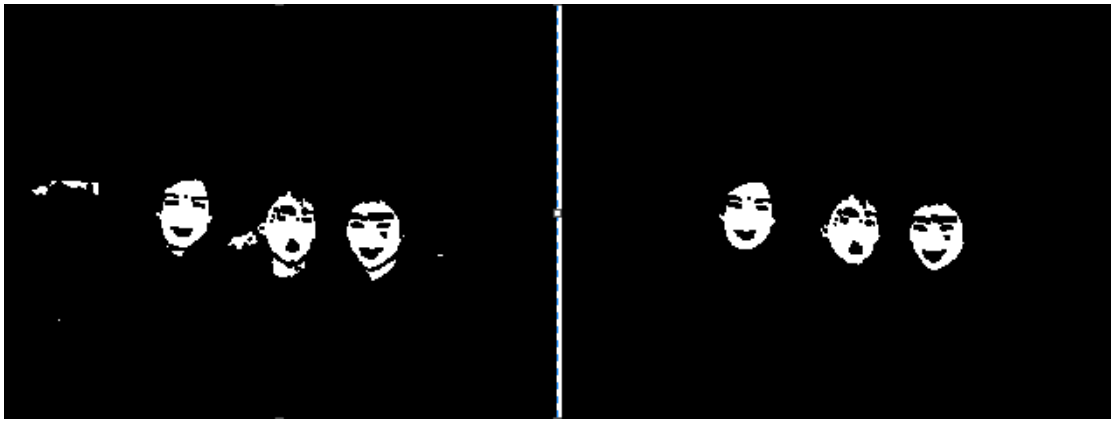


Figure 18

2.4 Get ROI images by successive ROI windows

After doing all the previous procedures, the remaining binary image might contains only meaningful objects to be detected as shown above in Figure 18. However, there might be the situation shown below in Figure 19.

Figure 19

Thus we could apply successive small windows to get different small ROI (Region of Interest) on each object region in the binary

image, and then get the corresponding ROI grey image from the original input color image. Of course this project converts the color image to a grey level image at the beginning that there is no need to convert the color image to grey image at each time we want to get a ROI grey image.

The major steps of getting ROI grey images by successive small ROI windows are:

1. Finds the coordinates of the rectangle boundary of each remaining object in the binary image. This project designs a subroutine get_roi_coordinate() to do this. A rectangle boundary of one object in the binary image shown in Figure 19 could be seen as below in Figure 20.

Figure 20

2. Initializes a small ROI window, for example, we could initialize a small ROI window with the same height as the rectangle boundary shown above in Figure 20, and the width of this small ROI window could be calculated by width=height*ratio1 where height is the height of this ROI window and ratio1= (the height of a training image) / (the width of a training image). Here, we could also initialize a ROI window with the width the same as the width of a

rectangle boundary in the situation that the shape of this rectangle boundary is too slim rather than too fat. Then the height of such a ROI window could be calculated as height=width*ratio2 where ratio2= (the width of a training image) / (the height of a training image). And we could also choose the height or width of the ROI window to other values at different situations and then calculate the corresponding width or height. After we initialize the size of the ROI window, then we could initialize the x and y coordinates of this ROI window, in this project, the ROI windows are always begins from the top northwest side in the object rectangle boundary.

3. Gets the ROI grey image after setting up the ROI window. This project designs a subroutine get_roi_image() to obtain a small ROI grey image based on the size and the coordinates of the ROI window in the binary image. An example of a ROI grey image corresponding to an initialized ROI window in the binary object rectangle boundary shown in Figure 20 could be seen as below in Figure 21.

Figure 21

4. Apply successive ROI windows to get successive ROI grey image for detection. The ROI window could moves from the west to

the east in the rectangle boundary, or moves from the north to the south, or moves from the northwest to the southeast. This project just designs algorithms to make successive ROI windows moving from the west to the east in the situation that the rectangle boundary of an object in the binary image is too fat, and algorithms to make successive ROI windows moving from the north to the south in the situation that the rectangle boundary of an object in the binary image is too slim. If the rectangle boundary of an object is neither too fat nor too slim, then resize the corresponding grey image and projects it to the eigenfaces subspace for detection directly, at the assumption that a face is in one of the three situations in an object rectangle boundary as shown below in Figure 22.

Figure 22

The moving of the ROI window or the making of successive windows is done by setting step width and threshold for the moving

of window. If we want successive windows moving from the west to the east, then we could set up x step width for the moving and a threshold to stop the moving. The idea is the same for the moving of windows from the north to the south. The x step width could be made as x step width = (the width of the rectangle boundary – the width of the initialized ROI window) / 20, and 20 means that we will have at most 20 successive windows for detection. The threshold to stop the moving of windows could be made as threshold = the width of the rectangle boundary minus the width of the initialized ROI window.

5. Resize the ROI grey image to the same size as the training image. This step of resizing is done by a function from OpenCV cvResize() using the bilinear interpolation method.

2.5 Projects each ROI image

Project each ROI grey image to the eigenfaces subspace to obtain the weights of this ROI image. Firstly smooth the ROI grey image, and then projects it onto the eigenfaces subspace. The projection is done based on the formula shown in Figure 13 in the previous chapter of this report. This project designs a subroutine project_to_subspace() to project the image to the eigenfaces subspace to obtain all the projected weights, and this subroutine calculates the Euclidean distance between an image and the whole

eigenfaces subspace at the same time.

2.6 Calculates the Euclidean distance

Calculate the Euclidean distances between the ROI image and all the eigenfaces. This is done by the formula shown as below in Figure 23.

$$\epsilon_k^2 = \|(\mathbf{\Omega} - \mathbf{\Omega}_k)\|^2$$
$$\epsilon^2 = \|\mathbf{\Phi} - \mathbf{\Phi}_f\|^2$$

Figure 23

The Figure 23 shows the calculation of Euclidean distance between an image and a specific eigenface, and the Euclidean distance between an image and the whole eigenfaces subspace.

After calculating the Euclidean distances, find the maximum Euclidean distance and the minimum Euclidean distance from the above step, and also calculates the average of all the Euclidean distances between the ROI image and all the eigenfaces.

2.7 Calculates the possibility

Calculate the possibility of the existence of a face in the ROI image by considering the maximum Euclidean distance, the minimum Euclidean distance, the average Euclidean distance, and the Euclidean distance between the ROI image and the whole eigenfaces subspace. This project designs a subroutine calculate_possibility() to calculate the possibility by some predefined thresholds. If the minimum Euclidean distance is below

a threshold as well the Euclidean distance to the subspace is also below a threshold, then there might possibly be a face in the image. But if only the minimum Euclidean is below a threshold and the Euclidean distance to the subspace is much larger than a threshold, then we could not say that there is a face because this situation only implies that the image is coding similarly to one of the training face images. And if the minimum Euclidean distance is a little larger than a threshold, but the Euclidean distance to the subspace is below a threshold as well the average Euclidean and the maximum Euclidean are below some thresholds, then we could still make a decision that there is an unknown human face in the image.

The thresholds for calculating the final possibility are made by many times of testing on many test images in this project. However, if we want to improve the accuracy of detection, then we have to set higher thresholds which means that more faces will be missed. And if we set lower thresholds then more faces would be detected, at the cost of more false detection. Thus there is a compromise in choosing the thresholds for calculation of possibility. An example of image with false detection could be seen as below in Figure 24.

Figure 24

2.8 Detect faces

The last step of the detecting part as well as the last step of the whole project is detecting faces based on the final possibility of the existence of a face. An example of the output in an image with both the right detection and false detection could be seen below in Figure 25.

Figure 25

Once we detect a face in a ROI image, then we could stop the moving of the ROI window in the current rectangle boundary area and continue to search for another rectangle boundary of another object and do the detection.

Experiments and testing

In order to test the efficiency and access the viability of this approach, this project builds experimental system whose diagram could be seen in Figure 2 to shows outputs of each step of the whole program (the major diagram could be seen above in Figure 2, and the major steps and procedures of this project could be seen above in the technical approach chapter). And for testing in different conditions, this project chooses pictures taken in different conditions and with different people, and these people are with different facial expressions.

Experiments of training

The experimental system of eigenfaces is shown in Figure 2, and the training data set is shown in Figure 3. The testing plans and test results of eigenfaces are:

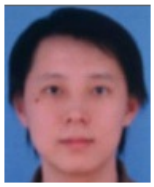
1. Test the calculation of the average face. This subroutine works well, and the result could be seen in Figure 5.

2. Test the calculation of the covariance matrix. It is hard to access the accuracy of each element in the covariance matrix. However, Figure 11 shows that there seems to be no error in calculation of the covariance matrix because the output covariance matrix indeed helps calculating the useful eigenfaces.

3. Test the Jacobi rotation. The output of the Jacobi rotation could be seen in Figure 11 in which is a covariance matrix after being calculated by Jacobi rotations.

4. Test the eigenfaces. The output eigenfaces could be seen above in Figure 12. The accuracy has to be tested in further steps.

5. Test of the whole training procedure and eigenfaces approach. After calculating the previous procedures and obtain the eigenfaces subspace, this project projects each training face onto the eigenfaces subspace, and detect images with the same size as the training images based on the eigenfaces and the projected weights of training images to eigenfaces. Through the testing of detection of these images, this project finds that the eigenfaces approach works in recognition procedure, that is, an image with a human face indeed has smaller Euclidean distance to subspace and has a smaller minimum Euclidean distance to eigenfaces. And if we project and calculates the Euclidean distance of a known face image, we could find that the Euclidean distance tends to zero which means that the face is recognized. However, this project also finds that the brightness of an image influence greatly of the result. To illustrate the testing result of eigenfaces better, let us see an example shown below in Figure 26.



```
euclidean distance to facespace 7950137.000000    euclidean distance to facespace 15838220.000000
euclidean distance is 0.000000, 42721388.000000   euclidean distance is 0.000000, 82236360.000000
euclidean distance is 0.000000, 38696092.000000   euclidean distance is 0.000000, 98950184.000000
euclidean distance is 0.000000, 22287248.000000   euclidean distance is 0.000000, 67824208.000000
euclidean distance is 0.000000, 17184666.000000   euclidean distance is 0.000000, 56389332.000000
euclidean distance is 0.000000, 25402882.000000   euclidean distance is 0.000000, 60028372.000000
euclidean distance is 0.000000, 14286589.000000   euclidean distance is 0.000000, 32292710.000000
euclidean distance is 0.000000, 21825788.000000   euclidean distance is 0.000000, 76394080.000000
euclidean distance is 0.000000, 30936534.000000   euclidean distance is 0.000000, 82168864.000000
euclidean distance is 0.000000, 52682872.000000   euclidean distance is 0.000000, 68838616.000000
euclidean distance is 0.000000, 38065052.000000   euclidean distance is 0.000000, 97535240.000000
euclidean distance is 0.000000, 21586888.000000   euclidean distance is 0.000000, 64972520.000000
euclidean distance is 0.000000, 22367680.000000   euclidean distance is 0.000000, 61319832.000000
euclidean distance is 0.000000, 25660758.000000   euclidean distance is 0.000000, 59942800.000000
euclidean distance is 0.000000, 13163183.000000   euclidean distance is 0.000000, 33709392.000000
euclidean distance is 0.000000, 21754262.000000   euclidean distance is 0.000000, 81440232.000000
euclidean distance is 0.000000, 31933340.000000   euclidean distance is 0.000000, 87861792.000000
euclidean distance is 0.000000, 66840388.000000   euclidean distance is 0.000000, 104231528.000000
euclidean distance is 0.000000, 45850752.000000   euclidean distance is 0.000000, 106917832.000000
euclidean distance is 0.000000, 36775064.000000   euclidean distance is 0.000000, 117505952.000000
euclidean distance is 0.000000, 44433760.000000   euclidean distance is 0.000000, 112846360.000000
euclidean distance is 0.000000, 25267064.000000   euclidean distance is 0.000000, 36786284.000000
euclidean distance is 0.000000, 52006816.000000   euclidean distance is 0.000000, 84277112.000000
euclidean distance is 0.000000, 46232452.000000   euclidean distance is 0.000000, 68393112.000000
```

Figure 26

As shown above in Figure 26, the person in the left testing image is the same person in the right testing image, and even the facial expressions are the same. However, the result Euclidean distances are very different. The reason is because the brightness and background of the right image are different with the training images while the left image is close to the training images. Thus, the brightness and the background of an image could influence the result of recognition by eigenfaces. And the choosing of training images could of course influence the viability of the training procedure based on eigenfaces approach. To deal with this problem, we could apply some preprocessing procedure such as normalizing the images.

Experiments of Detection

The experimental system's diagram of detection could be seen in Figure 2, and each step could be seen in the technical approach chapter. The testing plan is:

1. Tests of converting the image to binary image. This project applies a HSV range as the threshold to convert to image to a binary image at the assumption that we are dealing with images taken in good light conditions and we just need to detect human faces with normal facial expressions. Some good examples could be seen above in Figure 15 and Figure 17. And some examples could be seen below in Figure 27 that it is hard to find potential area for further detection of faces, and even the conversion makes the detection more difficult.

Figure 27

To handle with this problem, we could choose a larger range of threshold, or apply other algorithms based on the edge or color

histogram to calculate a threshold rather than use a HSV range as the threshold.

2. Tests of the erosion and connectedness for panning. The erosion function works well in this project, and this project has tested 6 types of structure elements, and has tried different combinations of structure elements in more than one time of erode operation. The more erode, the more information we might lost. Thus this project uses two times of erode operations, and the result could be seen above in Figure 17. The 6-connectedness method is used in this project in grouping region in the binary image, and then calculates the complexity, and then filtering out useless objects whose areas and complexities are very small. The result could be seen in Figure 18. And through the experiments and testing, it seems that erosion, grouping and panning all work well. This is because these functions work on the output of the firstly converting of binary image. If we obtain a useful binary image, then we could get useful information by erosion and grouping and panning. However, if we work on a meaningless binary image, the erosion and grouping could not help in the detection. Thus the viability of this part relies on the quality of the binary image.

3. Tests of successive ROI windows. As shown in Figure 22, this project assumes that the face in a rectangle boundary of an object

in the binary image is in one of the three situations. Thus this project just applies successive ROI windows either from the north to the south or from the west to the east. But through experiments, we could see another situation as shown below in Figure 28.

Figure 28

As shown in Figure23, this project could not detect the face by the successive ROI windows used in this project. To detect this face, this project has to apply a more complex successive window searching from the northwest to the southeast. But this will decrease the efficiency and might cause more false detections. Another method is to erode one more time or erode with a stronger structure element.

4. Tests of projection and detection. This project chose different pictures taken in different conditions with different human faces to project to eigenfaces subspace and detect faces by the Euclidean distance. The project firstly detects faces just by the minimum Euclidean distance between the input image and the eigenfaces. But this causes many false detections and many missed faces. An

example could be seen below in Figure 29.

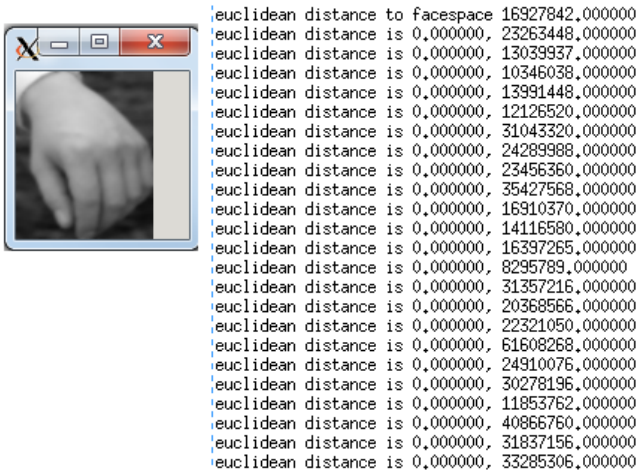


Figure 29

As shown in Figure 29, the minimum Euclidean distance is below a threshold used in this project, but this is a hand rather than a face. Thus this project designs an algorithm calculating the possibility of a face by considering the combination of the minimum Euclidean distance, the maximum Euclidean distance, the average Euclidean distance and the Euclidean distance to the subspace. And then some unknown faces used to be missed can be detected too.

However, even though this algorithm help improve the accuracy and detecting rate, it is still hard to compromise the accuracy of detection and the detecting rate. The smaller threshold causes the more accurate detection but with more missed faces. A potential solution to improve the detection might be applying self-adaptive thresholds to calculate the possibility. That is if the detection system could not detect faces then it could adjust the thresholds to lower values, and if the system find many faces in a short time and a

small number of images then it could adjust the thresholds to higher values.

5. Test the whole project. The whole project is tested on many different pictures. The detection works well on images taken in good conditions as assumed in the beginning of this report. But if the project is working on images taken outside the previous assumptions, then it could not detect anything. An example containing both good detection and the reason of a missed face is shown below in Figure 30.

Figure 30

As Figure 30 shown, if the image is in good condition as the previous assumptions, then the faces could be detected. In the first picture, two faces are detected, and the reason of a missed face is because this person's face is different with the training faces, thus the final possibility calculated is a little smaller than the predefined threshold. And the reason of another picture is that this picture is too small, and then the face area is treated as useless object and be filtered out. One solution is that we could find a larger image.

Future Works

In order to improve the viability and efficiency of this fast face detection approach, some improvements and method could be applied in the future works.

1. Apply preprocessing techniques for eigenfaces training and eigenfaces detection. Such preprocessing technique could be normalizing the training images and test images.

2. Apply more complex algorithms rather than just choose a HSV range to calculate the threshold to convert the image to binary image. We could apply feed –back algorithms in calculating the self-adaptive thresholds. The color range could be adjusted through the color histogram of the input image.

3. Apply Edge Detection and Hough Transform in finding the potential ROI of face area, and filter out useless objects by the edge features and shape features.

4. Apply a more adaptive successive window to search in a specific region from the northwest to the southeast.

5. Apply self-adaptive thresholds rather than fixed thresholds in calculation of the final possibility. Thus the adaptive threshold could compromise itself that increase the accuracy in some situations and decrease the missing rate in other situations.

Conclusions

This project designs a fast face detection method to detect human faces based on eigenfaces approach with a small training data set. Through the design and experiments of this project, we could see that the PCA idea and eigenfaces approach could be useful at some specific assumptions. With a small training data set, the eigenfaces approach could train fast and detect fast with a good performance and good accuracy on images taken in good conditions as the previous assumptions. And this project designs an approach to fast find the potential area to be detected by doing processing in binary image. By combining the binary techniques and PCA idea, we could design any specific fast detection system in detecting and recognizing specific objects in specific conditions. In future works, if we have different objects, then we could have different assumptions. And then we could choose different training data set or even different detection method such as LDA or HMM rather than PCA. And we could apply different algorithms to calculate thresholds to convert the image to binary image. Or we could directly find the ROI by other method without converting to binary. And we could also apply different algorithms to calculate the final possibility to obtain the result.

References

1. M. Turk, A. Pentland, Eigenfaces for recognition, Journal of Cognitive Neuroscience 3 (1) (1991)

2. M. Turk, A. Pentland, Face Recognition Using Eigenfaces, Proc. IEEE Conference on Computer Vision and Pattern Recognition 1991

3. Sirovich, L., & Kirby, M. Low-dimensional procedure for the characterization of human faces (1987)

4. Kirby, M., & Sirovich, L. Application of the Karhunen-Loeve procedure for the characterization of human faces, IEEE Transaction PAMI (1990)

5. Yang MH, Kriegman D, Ahuja N, Detecting faces in images: A survey. IEEE Trans Pattern Analysis and Machine Intelligence, 2002, 24 (1)

6. Peter N. Belhumeur, Joao P. Hespanha, and David J. Kriegman, Eigenface s vs. Fisherfaces: Recognition Using Class Specific Linear Projection, IEEE Transaction PAMI 1997

7. Aleix M. Martinez, PCA versus LDA, IEEE transaction PAMI 3(2) (2001)

8. William H. Press, S. Teukolsky, W. Vetterling, B. Flannery, Numerical Recipes in C, Cambridge University Press, 1988-1992

9. Pearson,Karl, On Lines and Planes of Closest Fit to Systems of Points in Spaces, 1901

10. http://www.facedetection.com/

11. http://groups.csail.mit.edu/vision/publications/index.php

12. http://opencv.willowgarage.com/documentation/cpp/index.html

13. http://www.cognotics.com/opencv/index.html

14. http://www.cs.cmu.edu/~cil/vision.html

15. http://www.fizyka.umk.pl/nrbook/bookcpdf.html