Facial Recognition

TNM034 - Advanced Image Processing

Jesper Lund, Tobias Matts, Anton Sterner, Rasmus Ståhl

December 10, 2017

Abstract

This report describes an approach to a facial recognition software, which was developed during five weeks by four students at Linköping University. The methodology and implementation process is described, together with the gathered results and a discussion of improvements and alternative methods. The result of this project is a Matlab program that detects and recognizes faces.

1 Introduction

Facial recognition used for biometric identification is becoming increasingly popular due to its many applications in every-day life; unlocking a smartphone or identifying suspected criminals at an airport are just two of countless others. There are many different techniques that can be applied to images to detect and identify faces. This project will use and evaluate some of the simpler and well proven ways of performing facial detection and recognition.

2 Aim

This project is a part of the course TNM034 *Advanced Image Processing* at Linköping University, and its aim is to develop a facial recognition software. The software should be able to detect and recognize faces in images, by comparing an image with images from a given database. The recognition should work when using images of faces with varying orientation and under different lighting conditions.

3 Methodology

The facial recognition algorithm consists of a few distinct steps, which are described in detail in the following section. The proposed algorithm used in the implemented program presumes a few things, such as that both eyes and the mouth are visible in the image. It also presumes that the faces are not rotated or scaled to large degrees.

3.1 Face Detection

The first step is face detection, where the program first denotes whether there is a face present in the image. It's an important step because there is no need in wasting computing power trying to recognize a face that is not there. This step is also essential for face alignment which is detailed in section 3.1.6.

3.1.1 Color Correction

Recognition of faces under different lighting conditions complicates the process. Color correction is a way to make the lighting conditions in each image more similar. Color correction lays the foundation for eye and mouth detection, which are both based on color (see sections 3.1.4 and 3.1.5). There are different well known color correction algorithms, such as gray world assumption and white point correction. In this project the gray world assumption was used since it yielded the best results. This method works by finding the largest RGB value in the image and using that value to normalize each channel. The results is an image with more neutral lighting. The purpose of using the gray-world assumption method is to establish the same lighting color in each image.

3.1.2 Color Space

The main component in detecting a face in an image is to segment skin regions from non-skin regions. In order to do this correctly, it is important to represent the image in an appropriate color space. The most common color space RGB is not suitable for skin detection, due to the fact that the three components (r, g, b) not only represents color, but also luminance [1]. Due to different lighting conditions, the luminance may vary in a face. Other color spaces, such as YC_bC_r and TSL, are more appropriate color spaces because the luminance can be removed from the color representation. The choice of color space for this implementation is YC_bC_r.

3.1.3 Face Mask

When an appropriate color space for skin detection is used, which is independent of luminance, skin regions can be detected by identifying a cluster that is associated with skin color. There are several approaches to identify this cluster such as thresholding, detecting boundaries and using a pre-set training data set. In this project the threshold method in the YC_bC_r color space was used. This makes it possible to segment skin regions by setting threshold values in the different color space components, where the color values between the threshold values correspond to the color of human skin.

3.1.4 Eye Map

When trying to detect a face, there can be multiple face candidates in a test image. Therefore, there is need for a way to verify these candidates. This can be done by utilizing the human face's characteristic features, e.g. the eyes (which will be covered in this section) and the mouth (covered in section 3.1.5).

The eyes make up for a good distinct feature, since their bright and dark regions differentiate themselves from the, more or less, neutral skin tones. The whites of the eye (as the name implies) stand out with their bright white color, while the pupil contrasts with its dark tone. Light reflexions in the cornea also create bright spots in the eye region.

Another useful feature in the anatomy of the face is the fact that the eyes are located level to each other. In the latter stages of the facial recognition process it is important that the faces to be compared are properly aligned and scaled (see 3.2). If a face is slightly tilted in any direction, the eyes will not be level with each other, which calls for the face to be aligned (more about this in 3.1.6).

After converting from the RGB color space to YC_bC_r the eyes can be found in the image with use of the color components. The eye positions can be found by creating a two-part eye map. The first map, which will from now on be called *EyeMapC*, uses the fact that the red levels C_r are low and that blue levels C_b are high in the eye regions. The map is constructed with the equation 1, where C_b, C_r and $\bar{C_r}$ are the blue, red and the negative of red chroma (255 - C_r) components respectively [2].

$$EyeMapC = \frac{1}{3}((C_b)^2 + (\bar{C_r})^2 + \frac{C_b}{C_r}) \tag{1}$$

The second part of the eye map, *EyeMapL*, utilizes the previously described dark and bright areas of the eye. Since the luma component Y describes the brightness in an image, it is used for constructing the second map. By performing dilation and erosion operations on the luma component, EyeMapL will bring out the dark and bright areas of the image. Equation 2 describes how the map was constructed by dilating the luma component *Y(x,y)* with the structuring element g_σ*(x,y)* and dividing it by the erosion of the luma component [2].

$$EyeMapL = \frac{Y(x,y) \oplus g_\sigma(x,y)}{Y(x,y) \ominus g_\sigma(x,y) + 1} \tag{2}$$

3.1.5 Mouth Map

The implementation of mouth detection utilizes the fact that the lips of a human being is generally significantly more red than any other facial region. The mouth map is constructed using equations 3 and 4 [2]. Here, n is the number of pixels in the full image or the face mask. Matrices C_r^2 and C_r/C_b are normalized to values between 0 and 255.

$$MouthMap = C_r^2 * (C_r^2 - \eta * \frac{C_r}{C_b})^2 \tag{3}$$

$$\eta = 0.95 * \frac{\frac{1}{n}\sum_{(x,y)} C_r(x,y)^2}{\frac{1}{n}\sum_{(x,y)} C_r(x,y)/C_b(x,y)} \tag{4}$$

3.1.6 Face Alignment

Each image is rotated based on the positions of the eyes and mouth. The image is rotated around the left eye position to align the eyes horizontally. First, the angle α between the eyes horizontal positions is calculated by 5, where x and y are image coordinates.

$$\alpha = -\frac{180}{\pi} * atan2^{1}(y_2 - y_1, x_2 - x_1) \tag{5}$$

3.1.7 Face Normalization

The face normalization process removes image features outside the face that are not necessary. The rectangle region that contains the face is extracted by using the positions of the eyes and mouth. This method is used in the paper by Ganhua Li et. al. [3], but in our implementation the scale coefficients differ slightly. The image coordinates of the upper left corner, X_{min} and Y_{min} is computed by 6 and 7, where W is the distance between the eyes and H is the vertical distance between eyes and mouth. The definition of coordinates X_{max} and Y_{max}, the lower rightmost corner of the image, is declared in 8 and 9.

$$X_{min} = \begin{cases} X_C - W & \text{if } X_C - W \text{ is } > 0 \\ 0 & \text{if } X_C - W \text{ is } \leq 0 \end{cases} \tag{6}$$

$$Y_{min} = \begin{cases} Y_C - \frac{3}{5}H & \text{if } Y_C - \frac{3}{5}H \text{ is } > 0 \\ 0 & \text{if } Y_C - \frac{3}{5}H \text{ is } \leq 0 \end{cases} \tag{7}$$

$$X_{max} = X_{min} + 2W \tag{8}$$

$$Y_{max} = Y_{min} + \frac{11}{5}H \tag{9}$$

[1] Four quadrant inverse tangent, https://en.wikipedia.org/wiki/Atan2

3.2 Face Recognition

Storing and comparing potentially large datasets of images requires high computer power and memory capacity. To elude this issue the *Principal Component Analysis* (PCA) algorithm is used, which is an algorithm that uses principal components to represent the data set of face images. Instead of storing and comparing a high-dimensional dataset of images, where each image pixel can be seen as a dimension, this method reduces the dimensions in the dataset drastically by looking at the variance between the images.

A mean image of the images in the database is calculated and subtracted from each of the database images. The result is an image with unique features highlighted. This image is used to calculate a covariance matrix, which subsequently is used to calculate eigenvectors and eigenvalues. The eigenvectors and eigenvalues are used to calculate *eigenfaces* and a weight vector for each eigenface. The weight vector describes how much each eigenface contributes to recreating the image. In other words, this can be thought of as each face in the database can be represented as a linear combination of the eigenfaces, where the weights act as the constants corresponding to the eigenfaces. This is described in 10, where μ is the mean image, w is the weight vectors and u is the eigenfaces.

$$Image_i = \mu + (\sum_{eigenfaces} w_j u_j) \tag{10}$$

When a face candidate is detected and should be recognized, the face image is projected onto the subspace which is spanned by the eigenfaces of the database. The result of the projection is a weight vector which is compared to the weight vectors corresponding to the eigenfaces. The minimum euclidean distance between the candidate weight vector and the subspace weight vectors tells which face image in the database is the best match. A pre-set threshold value at 4.0 determines if the minimum distance is close enough to be able to say that the candidate face exists in the database.

4 Implementation

This section will describe in detail how the program works and how it was implemented. The methods are based on the methodology described in section 3. The program is divided into several modules where each module has a specific functionality. The main module consists of two sections, which is described below.

4.1 Create the database

The first section of the main creates the database of eigenfaces given a directory consisting of images. The aim of this function is to save the matrices corresponding to the eigenfaces, which constitutes the database. To do this, the first step is to detect faces in the given images.

4.1.1 Face Detection

In the face detection step of the process the given images were color corrected using the gray-world assumption method and converted to the YC_bC_r color space. At this point, the next step was to create the face mask. As described in section 3.1.3, a pre-set threshold corresponding to skin color was used. The threshold value was chosen based on experiments on the given data set of images, trying to isolate skin segments. The experiment resulted in threshold values based on the chroma-red channel; $[Cr >= 125, Cr <= 183]$.

The face mask was then used to create a mouth map, isolating only the mouth part of the face image. The face mask was used to eliminate possible disturbances in the background, specifically other areas that contain high levels of red color (C_r). Image (b) in Figure 1 shows the result of applying equations 3 and 4 as described in section 3.1.5 and then normalizing the result. The morphological operation dilation was then applied to more clearly outline the detected areas and connect them into larger objects. Since there may be, in some images, other areas than the mouth inside the facemask that pass the threshold of red, the "correct" mouth area had to be identified. This was implemented by selecting the largest object in the image, i.e. the largest area of connected pixels with values 1 in the binary mouth map. The center point of the largest object was chosen as the center point of the mouth.

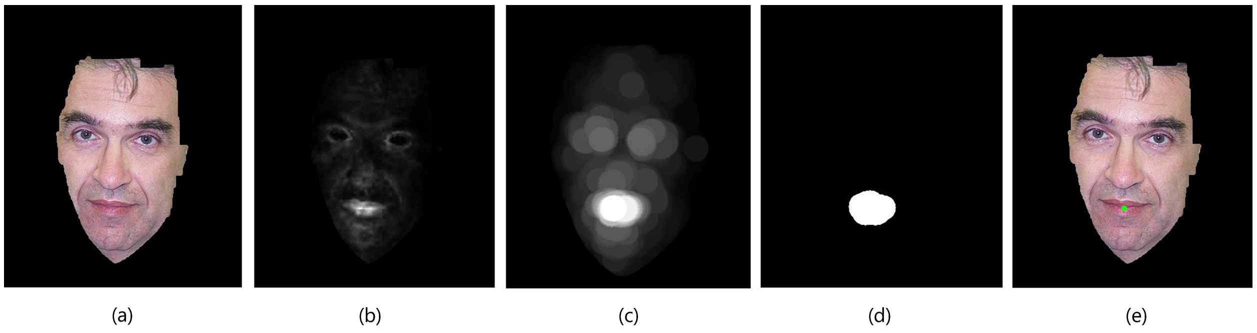

Figure 1: Mouth Map steps: (a) facemask. (b) after normalizing result of applying equation 3. (c) after dilation. (d) singling out mouth area after conversion to binary. (e) original facemask with mouth center highlighted.

The next step was to create the eye map, which process is described in 3.1.4. At this point in the process, some conditions were stated. If the program would not be able to detect two eyes and one mouth, the program should discard the current image and continue with detecting the next one. The position of the eyes and mouth were used to align the face. A custom image rotation algorithm was used to be able to rotate the images around any point, in this case the left eye center. One assumption was that the faces are always rotated less than 90 degrees in any direction from the upright position. The rotation is always centered around the leftmost eye position, which is assumed to be the left eye. Nearest neighbor interpolation was used to estimate the new pixel values after the rotation. The remaining step of the face detection step was to normalize the face, described in 3.1.7.

4.1.2 Calculate subspace of eigenfaces

To create the database, the detected face images were reshaped to a column vector and stored in a matrix. The matrix consisting of the face images were used to perform the *Principal Component Analysis* algorithm described in 3.2. The eigenfaces were calculated by sorting the eigenvalues, which were retrieved from Matlab's `eig` function, in ascending order. The weight vectors were calculated by the dot product of the normalized images and the eigenfaces.

Lastly in the first section of the main function the eigenfaces, mean image and the weight vectors were saved to the database, to be used in the recognition step of the program.

4.2 Candidate face verification

The aim of the second section of the main function is to read a candidate face image from a given directory and check if the face exists in the saved database. This was done by first detecting the candidate face described in 4.1.1. The face image was then projected onto the subspace described in 3.2. If the minimum euclidean distance were lower than the pre-set threshold value, the program should return the index of the matching image in the database. If the retrieved euclidean distance were larger than the threshold value the program should return zero, which means that the candidate face image does not exist in the database. If the candidate face could not get detected, i.e. exactly two eyes and one mouth could not get detected, the program should return negative one.

5 Results

In this section the results of the developed program is given. All images stated in the results are retrieved from `database 1`.

The eigenfaces which constitute the subspace and database are shown in Figure 2.

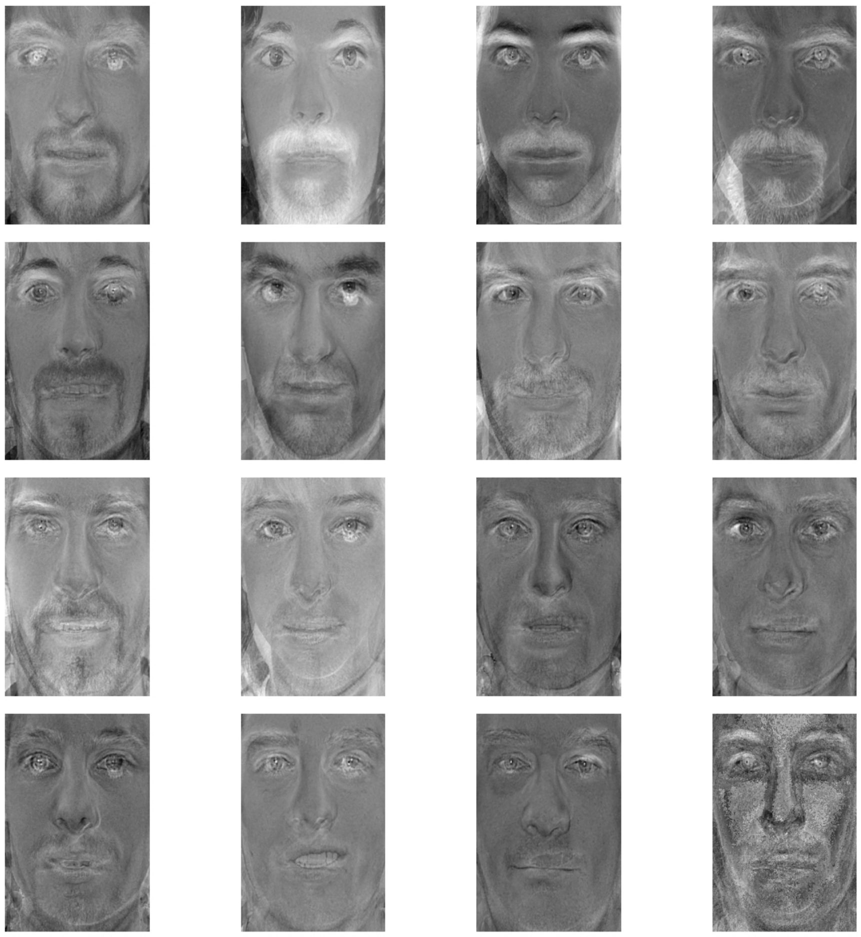

Figure 2: The eigenfaces which constitute the database

In the face detection step of the program, all the candidate face images could be detected. The conditions set for a detected candidate face were that two eyes and one mouth should be detected, with the assumption that the mouth center had to be below the eyes' center. The detected candidate faces, the output of the face detection step, are shown in Figure 3.

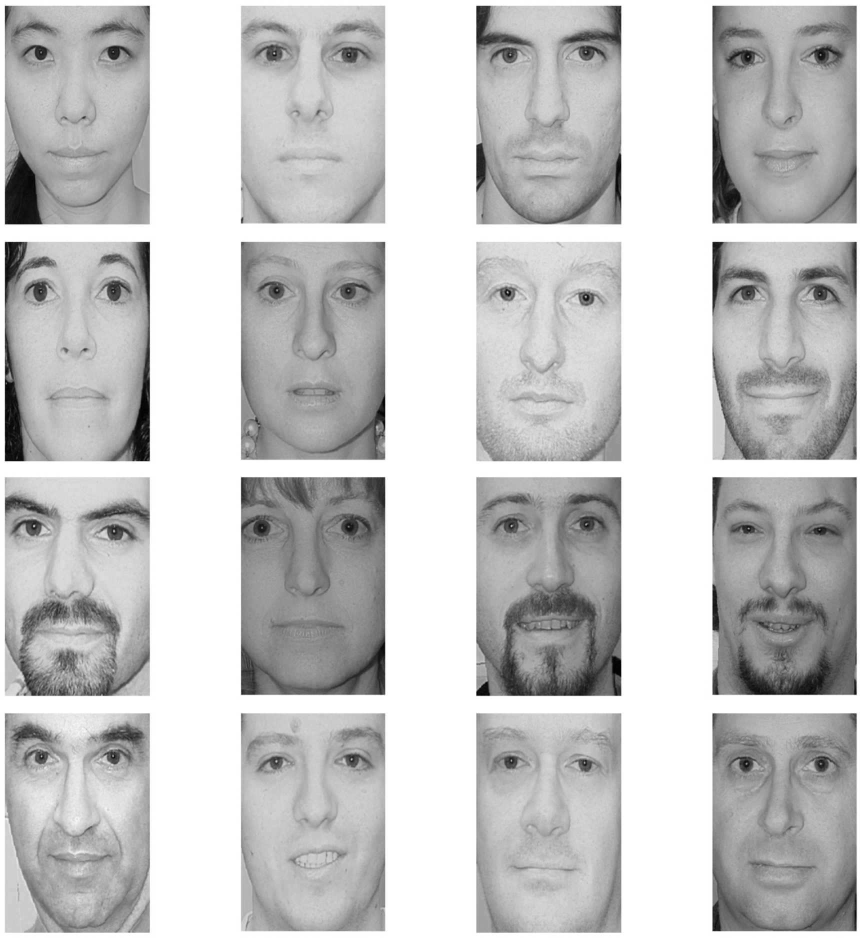

Figure 3: The output images of the face detection step

In the face recognition step, where the candidate faces were projected onto the subspace spanned by the database eigenfaces, all the candidate images were recognized by the program. For each candidate face, the minimum euclidean distance resulted in zero.

6 Discussion

In this section the results are discussed together with difficulties that occured in the implementation. Alternative solutions and further implementations of the methods described in this report is also described.

In `database 1`, the images are taken with no distracting background. The face detection algorithm depends on this condition and therefore the results from the face detection step become poor when images from `database 2` is used, where distracting backgrounds occur. There was one image in `database 1` which resulted in a poor face detection. This occured because the person in the image had bright sparkling areas in the hair, which confused the eye map algorithm. This problem was solved by cropping the binary eye map before extracting the two biggest white areas in the image, i.e. the eyes. The eye map was cropped with the assumption that the eyes are located in a certain area of the face. This resulted in that other areas which did not correspond to the eyes were erased from the eye map.

6.1 Results

In the results, it is stated that all the weight distances of the candidate faces resulted in the value zero. This result occurs because only images in `database 1` are used in the program, which means that the results of the face detection step when creating the database and detecting the candidate face are identical. Therefor, when the candidate face is projected onto the subspace and compared to the database, an identical face is detected. This results in a weight distance equal to zero.

The threshold value, which determines if a matched candidate face is close enough to be able to say that the face exists in the database, was set to the value 4.0. This value was chosen by testing the resulting minimum weight value of images that did not exist in the database. The result of the test was that the images that did not exist in the database had a minimum weight value of approximately 4.0 to 5.0.

6.2 Alternative solutions

There are many other techniques used for face recognition, and it's difficult conclude which methods work the best in every scenario. For one, different developers might often use their own database of face images for recognition, which could affect the results significantly. Even though it was not done in this particular project, it would have been valuable to try some other major face recognition method and compare the results of each method. Canny edge detection for example, which is described in a report written by Bhandari et. al. [4].

Like PCA, edge detection reduces the amount of data that needs to be considered when matching faces, while requiring significantly less computation, and thus time. What differentiates edge detection from the PCA method is instead of keeping a database with eigenfaces, the database is filled with image information in the form of ”nodal points”. Theses points, which are extracted through edge detection, include eye-distance, shape of cheekbones, lenth of jaw line, etc. Gaussian blurring is used to reduce noise in the image that could be mistaken for such points. The results of the study carried out by Bhandari et. al. [4] was described as that the algorithm worked, unless the face to be detected differed drastically from its intended database match in head tilting, lighting conditions, poses etc.

One method that could have been used to define a subspace instead of *Principal Component Analysis* (PCA) is *Linear Discriminant Analysis* (LDA) [5]. Like PCA, the LDA method is used to find the subspace representation of a set of facial images. The resulting basis vectors spanning the subspace are *Fisherfaces.* Unlike the PCA method, LDA uses several facial images of the same individual to group the data set of facial images into classes. This is beneficial when the images have different color and lighting conditions, but requires additional images of each individual. This was not provided in the data set used in this project. If Fisherfaces would have been used in this project, it is likely that the developed program could recognize the faces in `database 2` with a better result. However, the LDA method requires a higher face detection accuracy, which means that the face detection algorithm in this project would have to be further developed.

7 Conclusions

The algorithm described in this report provides a solid basis for face recognition, but it can be concluded that it would need further improvements if it was to be used in a professional setting. The developed program is sensitive to images with different lighting conditions as well as with distracting backgrounds. The main part of the recognition software that could be improved is the face detection step. To retrieve a proper face mask, eye map and mouth map is essential to get a good result of the face detection step.

In general, the face recognition program could be improved by and implementing additional functionality and conditions in the current algorithms, or replacing them with potentially superior alternatives such as canny edge detection or LDA.

References

[1] J.M. Chaves-González, M.A. Vega-Rodríguez, J.A. Gómez-Pulido, J.M. Sánchez-Pérez, *Detecting skin face recognition systems: A color spaces study.* Univ. Extremadura, Dept. Technologies of Computers and Communications, Escuela Politécnica, Campus Universitario s/n, 10071, Cáceres, Spain. (2009)

[2] R-L Hsu, M. Abdel-Mottaleb, A.K. Jain. *Face Detection in Color Images.* IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 24, NO. 5, (2002)

[3] Ganhua Li, Xuanping Cai, Xianshuai Li, Yunhui Liu, *An Efficient Face Normalization Algorithm Based on Eyes Detection.* Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems October 9 - 15, Beijing, China. (2006)

[4] Pankaj Bhandari, Pankaj K Gupta, Karthik U.S Goutham Reddy, Jeeva.B. *Analysis of Face Recognition Based On Edge Detection Algorithm with Hardware Interfacing.* Department of ECE, N.I.T Raichur, India. (2014)

[5] Peter N. Belhumeur, Joao P. Hespanha, and David J. Kriegman. *Eigenfaces vs. Fisherfaces: RecognitionUsing Class Specific Linear Projection.* IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 19, NO. 7. (1997)