Face Detection and Recognition

Linus Mossberg

Linköpings Universitet

1 INTRODUCTION

Facial recognition has become a widespread technology in recent years that successfully has been integrated into applications such as biometric device unlock systems and biometric attendance systems. Facial recognition has also started to see use in more controversial applications such as police surveillance systems.

The facial recognition problem consists of several chained sub-problems that all must be solved sufficiently well for the facial-recognition system to work consistently. These sub-problems can be divided into color correction, face detection, face normalization and face recognition. Each of these sub-problems can also be broken down into further sub-problems, such as eye detection, mouth detection, feature extraction and so on.

This report aims to present and motivate a face detection and recognition system that solves each of these sub-problems, as well as presenting and comparing the chosen methods against alternative methods. Some of these methods are based entirely on methods presented in other research papers, while others are more novel and was developed specifically for this implementation.

The implementation is implemented as a MATLAB program and it utilizes the Caltech Faces 1999 dataset for training and testing the program. This dataset contains 450 images of 27 people with various facial expressions, taken in different environments and under various lighting conditions. 16 of these people have been assigned ID-numbers, while the remaining 11 people are unknown. The program should be able to detect and recognize the face in an input image and return either the ID-number corresponding to the person or 0 if the person is unknown.

2 Background and Methods

The following sections describes all of the methods used to solve the facial recognition problem in detail, as well as alternative methods for many of the sub-problems and some implementation details.

2.1 White Balance

Throughout the entire detection and recognition pipeline, it is assumed that the input images have a neutral white scene illuminant. This is not the case for images in the Caltech database however, and methods to perform automatic white balancing is therefore required. This is done by finding the scene illuminant in the image, i.e. the color of the illumination in the scene. All pixels in the image is then transformed using the color transformation that transforms the scene illuminant to the white illuminant. Three different methods of finding the scene illuminant is used: Principal Component Analysis (PCA), Gray World (GW) and Gray Contribution (GC).

2.1.1 Principal Component Analysis

The PCA method is implemented using the built-in *illumpca* method, and it works by performing principal component analysis on a subset of RGB-vectors that has been projected on the average RGB-vector of the image [1]. The subset of projected RGB-vectors consists of the darkest and brightest RGB-vectors after projection. The first principal component vector represents the direction in RGB space where the variance of the subset of RGB-vectors is the greatest. This principal component is itself an RGB-vector, and it is the resulting scene illuminant of the method. The principal component analysis method is also used later to create a face recognition model called eigenfaces, and it is described in more detail there.

2.1.2 Gray World

The GW method is implemented using the built-in *illumgray* method, and it works by simply assuming that the scene illuminant is the average color of the image. Or from another point of view, it assumes that the average color of a white balanced image should be gray.

2.1.3 Gray Contribution

The GC method works similarly to the GW method. It computes the illuminant by taking average color of the brightest pixels whose gray intensity contribution accounts for 50% of the total gray intensity in the image, but with any overexposed pixels excluded. The gray intensity contribution of pixels is evaluated by using the Y-component of the YIQ color space, and the pixels that contributes 50% of the gray intensity is found using a histogram.

2.1.4 Skin Illuminant

If the returned scene illuminant from any of the above methods also fits the skin model, then it is likely that a bright face takes up much of the image frame which has influenced the scene illuminant approximation. If this scene illuminant were to be used to white balance the image, then the skin tones in the image would likely be pushed towards a bluer hue. This would in turn make it impossible to detect the face later. To mitigate this, the skin-likelihood of the scene illuminant is evaluated using the skin model. If the scene illuminant fits the skin model, then no white balancing is performed on the image.

2.2 Skin Detection

The first step in the implemented face detection method is to detect skin pixels in the image. This is done by creating a skin model that defines regions in a color space that corresponds to skin pixels, which then can be evaluated to find likely skin pixels in the images. There are many color spaces and methods to define skin regions within them to choose from when creating a skin model.

2.2.1 Transformed YCbCr Skin Model

The first method that was considered and implemented is described in *Face Detection in Color Images* [2], which uses a transformed YCbCr color space with an elliptical region in the CbCr subspace to define the skin model. Both the color space transformation and ellipse properties are defined by constants that the authors derived from observations of skin pixel samples. This method works well for some skin tones in the dataset but worse for others. One possible explanation for this is that the constants seems to be derived from skin pixel samples taken from scanned film-photographs, while the images in the dataset were taken with an early digital camera. These camera types have different response functions which could render skin tones differently. In order to properly detect skin tones in the provided dataset, a new skin model therefore had to be created based on skin pixel samples taken from the dataset.

2.2.2 Collecting Skin Pixel Samples

The skin pixel samples were collected by selecting a few representative faces from the dataset and then manually masking out skin pixels from them. These images had to be white balanced, which posed a problem since the white balance method itself depends on the skin model. To mitigate this issue, the selected images were either well balanced to begin with or had a blue bias rather than red. The selected images also had large bright areas that were brighter than the faces, and where those areas looked to be white reflecting materials. By then performing the white balance on the entire image and not just the cropped face (which would remove contextual information), the skin model evaluation could be removed from the white balance step with descent results. The PCA white balancing method was used as this produced the most consistent and accurate results, judged both visually and by looking at the skin tone spread of the samples in HS-subspace. Ideally the skin model should probably be created with unprocessed images taken under perfect white light, however.

2.2.3 Probability-based HSV Skin Model

Once a good distribution of skin pixel samples had been collected, a new skin model could be created. Since the model no longer had to be tied to any existing model with pre-defined constants, a new probability-based model was developed.

2.2.3.1 Creating the Model

The model is similar to the one described in *Skin Detection and Segmentation of Human Face in Color Images* [3], but instead of using a covariance matrix to implicitly describe an ellipse-shaped distribution in the chosen color-subspace, an explicitly described distribution is created on a 2D-grid using a bivariate histogram. The grid is created by filling the histogram with the collected skin pixel samples, and the resulting histogram represents the density of skin pixels at different coordinates in the chosen color-subspace. This grid is then normalized by scaling the bin counts to the range $[0,1]$ and then flattened by taking the cube root of each element to reduce variation. The result represents a skin probability distribution[1] which can be used to find the probability of a color being a skin color.

The benefit of using this method compared to using a covariance matrix is that the skin regions can have any shape in the chosen color-subspace. Any color-subspace can be used, and the histogram will automatically scale non-uniformly to fit the data and maximize the effective resolution of the grid. The HS-subspace from the HSV color space with hue shifted 180° to center skin tones was chosen because it produced the best results among the color subspaces that were considered, the remaining of which were: Y[CbCr], Y[CgCr] and [HS]V transformed to radial coordinates. The HS-subspace is also the most suitable color space for skin detection according to *Detecting skin in face recognition systems: A colour spaces study* [4]. The grid resolution is set to be $512x512$ and the resulting probability distribution can be seen in Figure 1.

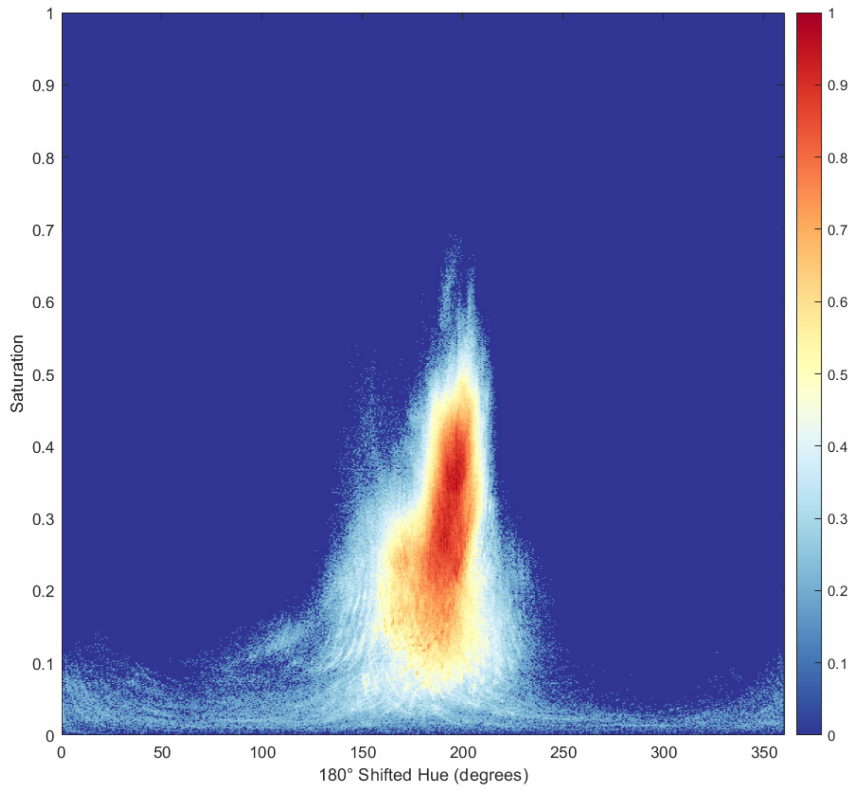

Figure 1 - Visualization of the skin probability distribution in hue-saturation subspace

[1] The integral of a proper probability distribution should be 1, but this is not necessary in this case and scaling the distribution with a constant makes no difference in the following steps.

2.2.3.2 Evaluating the Model

The skin probability of pixels in an image is evaluated by looking up the probability values of the corresponding color coordinates in the probability distribution grid. Bilinear interpolation is used to find the interpolated probability within the $2x2$ grid-cell where the color coordinate of a pixel is located, which results in four grid-lookups for each pixel. The result is a skin probability image and an example of this is shown in Figure 2 (b).

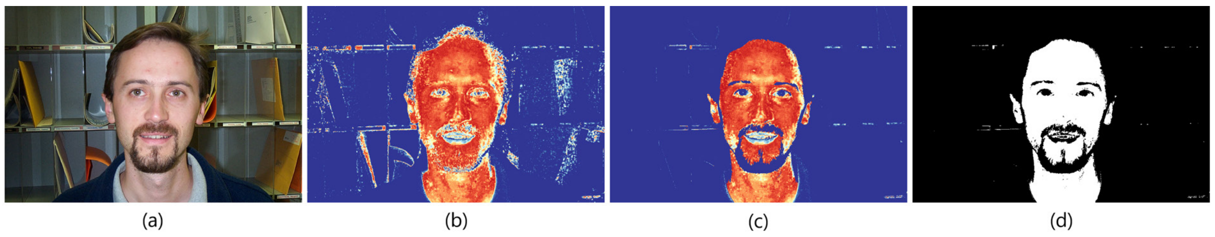

Figure 2 - Skin model evaluation: a) Original image, b) Skin probability image, c) Value-limited skin probability image, d) Resulting skin binary image. Color legend is the same as in Figure 1.

The skin probability image is used to find the most probable skin pixels in the image. This is done by finding a probability threshold value that divides the image pixels into skin pixels and non-skin pixels. The threshold value could be set to a constant such as 0.5, but this would not work well in all situations given that some skin tones might be underrepresented in the skin model and the backgrounds might have different amounts of skin probable regions. A better solution is to use an automatic thresholding algorithm, and the one chosen for this purpose is *Otsu's method* via the built-in function *graythresh*. Otsu's method finds the threshold value that divides the probability pixels into two groups with minimum within-group variance and maximum between-group variance [3].

The resulting thresholded skin binary image works, but it is improved by finding a lower limit value/brightness threshold of the resulting skin pixels and by setting any pixel that falls under this threshold to 0. This threshold is found by using Otsu's method on the value-component of the masked skin pixels. This is very effective at removing hair, iris, background clutter and such that fits the skin model in hue and saturation but are less bright than what skin pixels tend to be in the dataset. The difference between before and after this step is show in image (b) and (c) in Figure 2. The assumption that skin pixels generally are brighter is most often true for the Caltech dataset, but the non-value-limited binary skin image is returned as well and is used when the value-limited binary skin image fails to produce a good face mask.

Automatic thresholding can't be used when a single color value is evaluated, such as when the white balance method evaluates the scene illuminant. In these cases, the evaluation method uses a special threshold value which is computed when the skin model is built. This threshold is defined to be the threshold that discards 68.5% of the lower probability contribution in the probability distribution grid.

2.3 FACE MASK

The skin binary image returned from the skin model evaluation is used to create the face mask. The face mask creation algorithm has been developed to keep the face region as intact as possible and no morphological operations are applied to it directly. Despite how extensive the algorithm is, the only processing done directly on the resulting face mask is small hole fill operations and removal of connected binary regions. Besides finding the most likely face region in the binary image, the algorithm also assigns a quality score to the returned face mask.

2.3.1 Face Candidates

The first step in the algorithm once any small holes have been filled is to find possible face candidates based on connected component properties of regions in the binary image. The binary regions must meet all of the following conditions to be considered a valid face candidate:

- More than zero holes (Euler number less than 1).
- Angle between major axis and y-axis less than 45°. Ignore this if ratio between major and minor axis length is less than 1.25.
- Area of region with holes filled larger than 15 000 pixels.
- Eccentricity of region in range $[0.4, 0.92]$.
- Ratio between area of bounding box and area of region in range $[0.3, 0.88]$.
- Ratio between bounding box height and width less than 2.25.
- Ratio between area of region with holes filled and area of region larger than 1.01.

These conditions have been found experimentally and they usually only leave one valid face candidate, but there can be more. If no face candidates remain, the binary skin image is returned, and the face mask quality is set to 0.

If there are remaining face candidates, the next step is to pair some of the discarded regions with the face candidates. This is done because parts of the face are sometimes separated from the larger face region, a common example of this is the chin and mouth region which sometimes is separated from the face by a beard. An example of this is shown in the rightmost image in Figure 3. Regions are paired with the face candidates if their bounding box overlaps the face candidate bounding box by more than 45%, provided that the area of the region is less than 25% of the face candidate area.

Finally, a few more connected component properties are generated from the face candidates with pairs included. This is done by first creating a new binary image for each face candidate which includes the paired regions, and then performing morphological close and hole filling operations on the result to create a perimeter region. The properties generated from these perimeter regions are circularity and the perimeter ratio, which is the ratio between the perimeter of the ellipse encompassing the region and the perimeter of the actual region. Any face candidate with a perimeter ratio less than 0.617 is then discarded. These properties are also used later to determine the face mask quality.

The remaining face candidates are considered to be possible faces. Before a quality score is determined for the remaining candidates, any overexposed clipped pixels within the face perimeter region is added to the face mask. This is done because the skin model can't determine if clipped pixels belongs to the skin, but at this point they are likely to belong.

2.3.2 Face Mask Quality

The face mask quality of the remaining candidate regions is determined using different connected component properties of the regions. The following properties influence the quality score:

- Perimeter ratio.
- Circularity.
- Ratio between area and bounding box area.
- Distance to center of frame.
- Ratio between area of holes in region and area of region.
- Likelihood that the triangle that runs through the three largest holes in the region could constitute a face triangle (mouth and eyes).

These properties and the ways in which they influence the quality score was determined experimentally. The quality score is initially used to select the single best face candidate if there are multiple, because the dataset only contains one face per image. This could however easily be extended to detect multiple faces by using a quality threshold. The results of all these steps for a few extreme cases is shown in Figure 3.

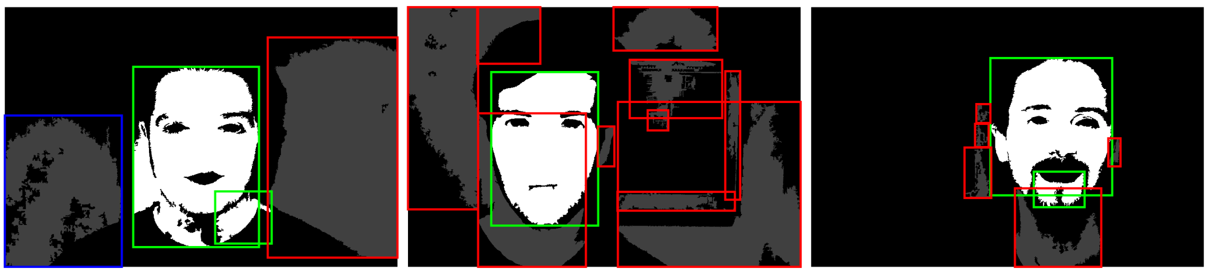

Figure 3 – Extreme case face mask region examples with colored bounding boxes. Green: winner regions. Blue: runner up regions. Red: Discarded regions.

The main use of the quality score is however to select the best face mask from a collection of face masks generated using images with different white balance and skin model settings. Four images are generated with different white balance methods: none, PCA, GW and GC. The skin model is then evaluated for each image, which results in two skin binary images for each of the four images, one with value limitation and one without. The result is a total of eight skin binary images, each of which can be used to find a face mask with a corresponding quality score. The non-value-limited skin binary images are only used if the best face mask score of the value-limited ones falls below a specified threshold, however.

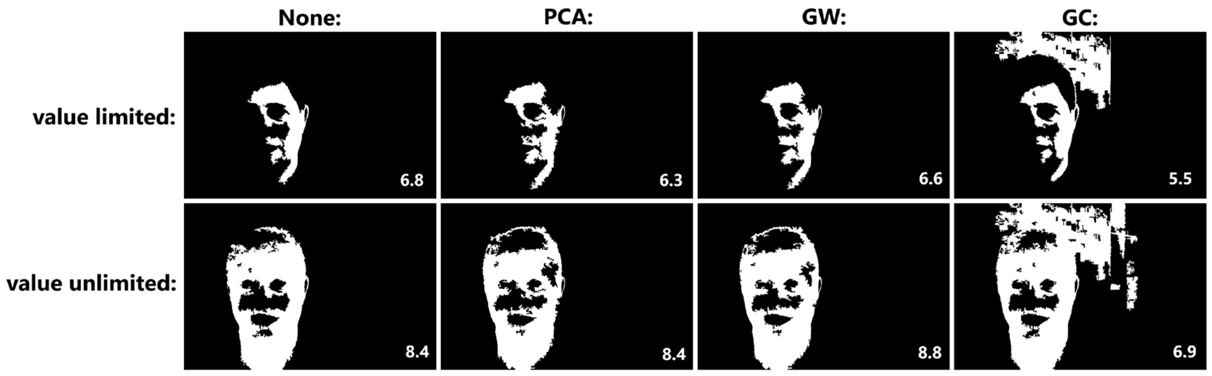

Figure 4 - Face masks generated using different methods for a difficult case. The quality score for each face mask is shown in the lower right corner.

Face masks for a particularly difficult case which would fail without this method is shown in Figure 4. There's no white balance method or skin model setting that works in all cases and removing any one of these would reduce the detection rate of the program.

2.4 Eye Detection

The eye detection method relies on the face mask a lot because it only searches for eyes located in holes in the upper part of the face mask. This would not work with a less accurate skin model, and it's the reason why the face mask algorithm tries to keep the face region as intact as possible. The upper part of the face mask is masked out using the upper part of the ellipse that encompasses the face region, which wouldn't work without a reliable face mask either.

The eyes are located by first generating the eye map described in *Face Detection in Color Images* [2] and then applying the eye mask (consisting of holes from the upper part of the face mask) to it. Even though this region is very limited, there can still be several possible eye regions within it. In order to select the best possible pair of eyes and to locate the center of these eyes, an iterative Otsu thresholding algorithm was developed. The basic steps of one iteration are as follows:

1. Generate a new eye mask by thresholding the eye map with the eye mask generated in the previous iteration applied.
2. Find possible eye pairs in the new eye mask.
3. Select eye mask to use in next iteration:
    a. If there are more than one possible eye pair, select the best pair using an eye property space and generate a new eye mask consisting of this eye pair.
    b. If there are one possible eye pair, do nothing and assign the new eye mask generated in (1) to be the eye mask. This will refine the eye mask.
    c. If this is the first iteration and only one eye is found, start flattening the eye map to recover an eye that might be dimmer. Assign the old eye mask that haven't been thresholded as the eye mask.

d. If this is not the first iteration and only one eye is found, refinement is done, and the eye mask generated in the previous iteration contains the final refined eye mask with one eye pair. Break the loop and locate the eye positions in the eye mask.

There are several more special cases and ways in which the loop can break, but this is the basic idea of the algorithm. Valid eye pairs are defined to be any region pairs with a pixel distance in the range $[30, 300]$ that forms a line with a maximum angle of 30° to the x-axis.

The eye property space was defined experimentally based on connected component properties of the eye regions such that increasing values in each dimension means the region is more eye-like in that dimension. This means that each possible eye region can be represented by an eye property space vector, and the length of this vector determines how eye-like the region is. The distance between two eyes in eye property space also determines how similar the eyes are. To determine the best eye pair, a combination of the sum of the eye-likeness of each eye in a pair and the similarity of these eyes is used.

2.5 Mouth Detection

The mouth detection uses a similar method as the eye detection. It relies on the eye detection and only searches for mouths located in an ellipse positioned, rotated and scaled relative to the eyes. This ellipse is defined based on expected value and standard deviation for a few face samples of the ratio between the distance from mouth to the eyeline and the distance between the eyes.

The mouth is found within this ellipse region by generating the mouth map described in *Face Detection in Color Images* [2] and then using an iterative Otsu thresholding method similar to the one used for the eye detection, albeit a bit simpler since the search region is more limited and there are no mouth pairs.

2.6 Face Normalization

Face normalization is performed to normalize the appearance of faces to only leave information that is relevant for the recognition step. This is done by applying an affine transformation to the image, which is defined to be the transformation that transforms the detected face triangle in the image to a pre-defined template triangle. This applies the correct scaling, translation and rotation to the image to align detected faces with each other. The image is then cropped to a pre-defined size and converted to grayscale. The resulting image gray levels are also stretched to the range $[0, 1]$ to normalize the lighting slightly. Histogram equalization is commonly used to better normalize the lighting, but this can be handled better by the recognition models by removing the first three major principal components from the eigenfaces model or by simply using the fisherfaces model which handles lighting better.

The affine transformation differs depending on which recognition model will be used. With the eigenfaces model, the transformation is allowed to scale the image non-uniformly to properly align all three triangle points with the template triangle. With the fisherfaces model, only the eye points are aligned to the template triangle and the mouth point is ignored. This results in uniform scaling only which doesn't warp the face. A subtle example of this can be seen in Figure 5. The difference becomes more pronounced if the face proportions of the detected face deviates more from the average face, which the template face triangle is based on.

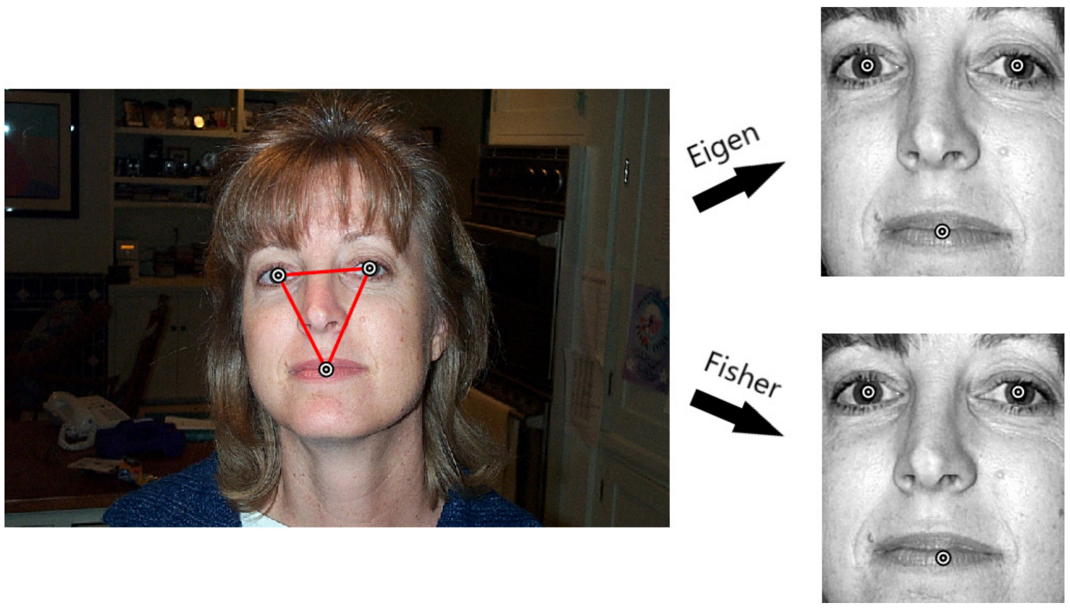

Figure 5 - Difference in face transformation when using eigenfaces and fisherfaces.

2.7 Face Recognition

Face recognition is performed by defining a face space which consists of one dimension for each pixel in the normalized face image, where the gray value of each pixel defines the position along the corresponding dimension. Euclidian distances between face coordinates in this space defines the similarity of faces, and this can be used to create a face recognition model. The problem with this representation however is that the normalized face images consists of $256 \times 284 = 72704$ pixels, which results in as many spatial dimensions and basis vectors in the face space. Luckily, the most important facial features aren't complex enough to warrant this many dimensions, and faces can instead be described by a few carefully selected basis vectors in the face space that represents the most important features.

There are different dimensionality reduction methods that can be used to find basis vectors like these. Principal Component Analysis (PCA) can be performed on a collection of training faces to form a set of basis vectors along which the variance in the training faces is the largest. These basis vectors are then called Eigenfaces [5].

Linear Discriminant Analysis (LDA) is another dimensionality reduction method that can be performed on a collection of training faces that are labelled with the person that they belong to. This instead forms a set of basis vectors along which the variance between persons in the training data is maximized, while the variance between faces of the same person in the training data is minimized. These basis vectors are then called Fisherfaces [6].

2.7.1 Eigenfaces

The eigenfaces of a collection of N training faces, which forms the face space vectors $x_1, \ldots, x_N$, are the eigenvectors of the covariance matrix C defined by:

$$A = [(x_1 - \mu), \ldots, (x_N - \mu)]$$

$$C = AA^T$$

where μ is the mean face space vector of the training faces. The problem with this covariance matrix is however that it has the dimensions $M \times M$, where M is the number of dimensions in the face space. As mentioned previously, the number of dimensions is the number of pixels in the normalized face images, which in this case results in $(256 \cdot 284)^2$ elements in the covariance matrix. Each element is represented by a double precision floating point value of 8 bytes, which results in a total size of $8 * (256 \cdot 284)^2 \approx 42 \cdot 10^9$ bytes for the covariance matrix, i.e. 42 gigabytes. Most systems today are not able to handle this, and another solution is therefore required.

Since the number of dimensions is much larger than the number of faces in the training set, i.e. $M > N$, one can instead find the eigenvectors u_i of the matrix $A^T A$ with a more manageable size of $N \times N$, and then compute the eigenfaces v_i using $v_i = Au_i$. [5]

In practice, however, the built-in economy-sized Singular Value Decomposition (SVD) function is used to find the eigenfaces and no covariance matrix is needed. It can be shown that if $X = [x_1, \ldots, x_N]$ and its singular value decomposition is $X = U\Sigma V$, then the first columns of U is the first eigenfaces of the training set. [7] The corresponding eigenvalues are the squared singular values (located in the diagonal of Σ) divided by N.

The first three major eigenfaces are removed since the largest variation in the training data tends to be caused by 3-dimensional illumination changes. [6] Of the remaining $N - 3$ eigenfaces, only the first major eigenvectors that contributes 88% of the total variance are kept. The variance contribution of each eigenface is determined by its corresponding eigenvalue. This removes minor eigenfaces that barely contribute without affecting the accuracy much.

A face space vector x can now be projected to the eigenface subspace using:

$$w_i = v_i^T \cdot (x - \mu)$$

Where $[w_1, \ldots, w_E]$ is the eigenface subspace coordinate, also known as the weights, and E is the number of remaining eigenfaces. The eight first eigenfaces for a set of 72 training faces generated by the program is shown in Figure 6.

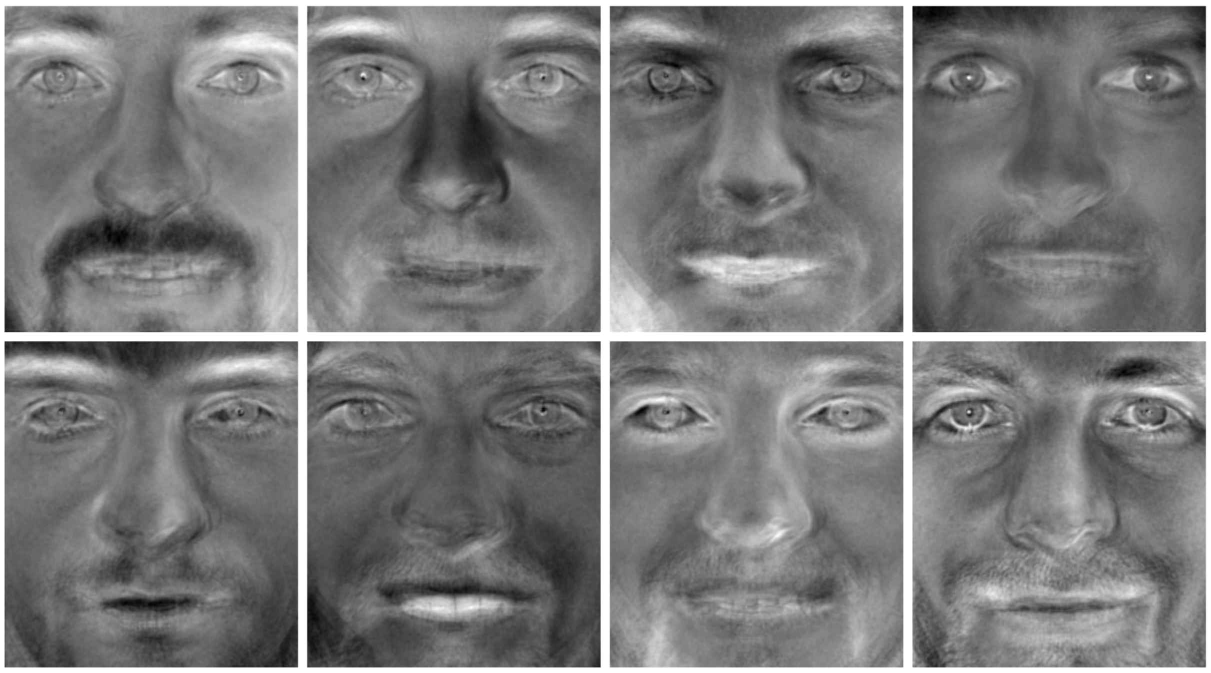

Figure 6 - Eight first eigenfaces generated by the program for a set of 72 training faces.

2.7.2 Fisherfaces

Fisherfaces for a collection of N training faces with c different labelled persons (classes) are found by first finding the $N - c$ major eigenfaces $v_1, \dots, v_{N-c}$ of the training faces and then projecting the training faces to this eigenface subspace [6]. This drastically reduces the number dimensions of the training face vectors, from $M (= 72704)$ to $N - c$, and it is required in practice to reduce the size of the scatter matrices.

The between class scatter matrix S_b is then defined as:

$$S_b = \sum_{i=1}^{c} N_i \cdot (\mu - \mu_i) \cdot (\mu - \mu_i)^T$$

where N_i is the number of training faces of person i and μ_i is the mean projected face vector of person i, while μ is the mean projected face vector of all training faces. The within class scatter matrix S_w is defined as:

$$A_i = [(x_{i,1} - \mu_i), \dots, (x_{i,N_i} - \mu_i)]$$

$$S_w = \sum_{i=1}^{c} A_i A_i^T$$

where $x_{i,1}, \dots, x_{i,N_i}$ is the projected faces of person i. The fisherfaces is then formed using:

$$V = [v_1, \dots, v_{N-c}]$$

$$U = [u_1, \dots, u_{c-1}]$$

$$F = V \cdot U$$

where the column vectors $u_1, \dots, u_{c-1}$ of U is the $c-1$ first major generalized eigenvectors of the scatter matrices S_b and S_w, and the column vectors $f_1, \dots, f_{c-1}$ of F is the resulting fisherfaces [6]. The generalized eigenvectors of scatter matrices S_b and S_w are computed using the built-in function $eig(S_b, S_w)$.

A face space vector x can now be projected to the fisherface subspace using:

$$w_i = f_i^T \cdot x$$

where $[w_1, \dots, w_{c-1}]$ is the fisherface subspace coordinate, also known as the weights. The eight first fisherfaces for a set of 72 training faces generated by the program is shown in Figure 7.

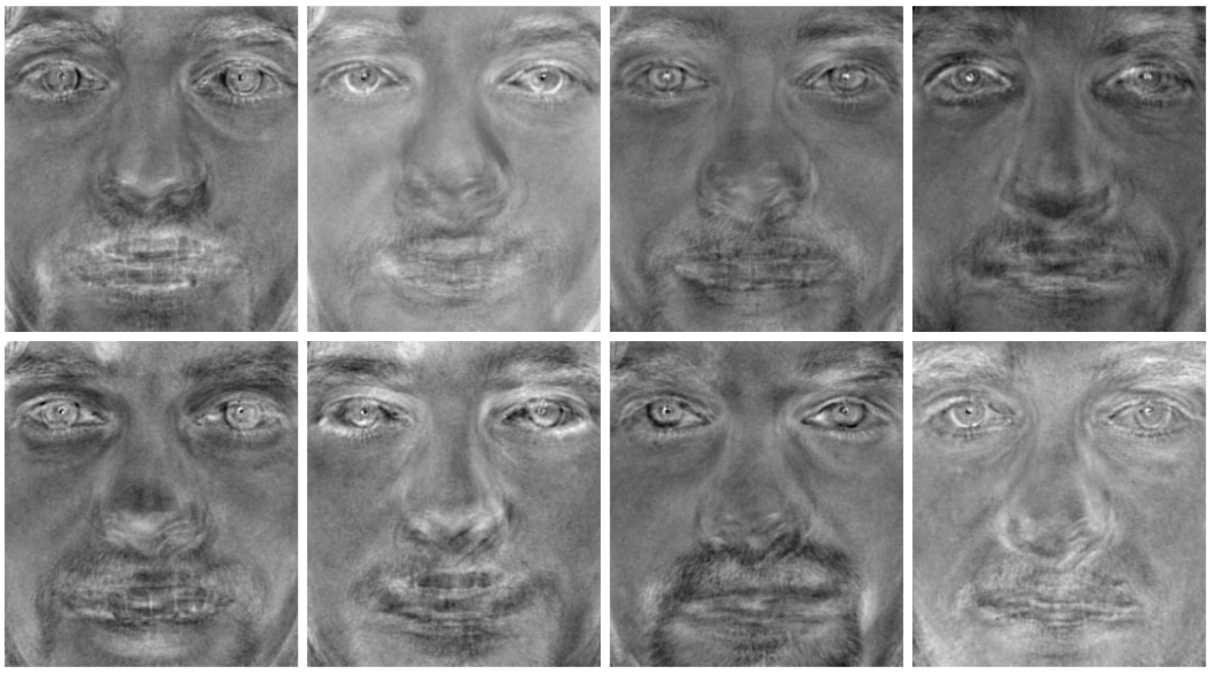
Figure 7 - Eight first fisherfaces generated by the program for a set of 72 training faces.

Note that the training faces of these are not aligned in the same way as the training faces of the eigenfaces, which creates more of a misaligned ghosting effect. The model performs better with this type of alignment despite this, however, likely because it preserves face proportions which are useful features.

2.7.3 Using Eigenfaces and Fisherfaces Models for Recognition

Euclidian distances between dimensionality reduced face coordinates in these new Eigenfaces and Fisherfaces subspaces defines the similarity of faces, and this can be used to create a face recognition model. This is done by projecting the training faces to the subspace to find the subspace coordinates of them, and then storing these coordinates paired with the ID-number that identifies the person that the training face belongs to.

A previously unseen image of a face can then be identified by projecting it to the subspace. By then comparing the distances to each of the training face coordinates in this subspace, the face is identified as being the person that belongs to the closest training face coordinate.

Stopping here would cause a problem however since faces of unknown persons would be identified as the most similar-looking known person. The program must be able to differentiate between known and unknown persons, which is achieved by using a distance threshold value. If the closest face is further away than this distance threshold, then the face is identified as belonging to an unknown person.

2.7.4 Finding Distance Thresholds

The distance threshold should be chosen differently depending on the purpose of the program. In security biometric applications, accepting the wrong person is unacceptable while rejecting the correct person based on a bad image is more acceptable. The distance threshold therefore must be low for this type of application.

In a biometric attendance system however, where unknown people are not expected to show up, accepting the wrong person is more acceptable while rejecting the correct person is not as acceptable. The distance threshold therefore must be high for this type of application.

The third option is to strive for maximum total accuracy, in which case the sum of false acceptances and rejections are minimized, which might be desirable in a more general application. The distance threshold for this type of application lies somewhere in between the thresholds of the previous two applications.

These thresholds must be found experimentally by testing the system on a set of images. The threshold chosen for the program is the ones that maximizes the total accuracy, and a few graphs that was generated to aid in finding thresholds are shown in Figure 8.

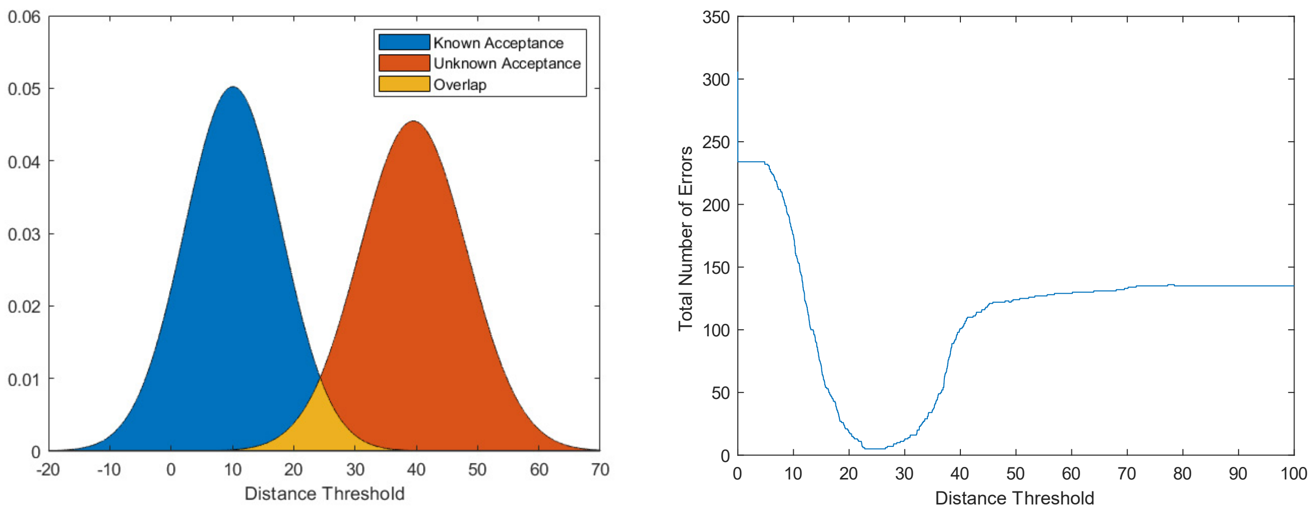

Figure 8 - Left: Normal distributions of known person acceptance and unknown person acceptance. Right: Total number of errors, or sum of false acceptances and rejections. Both are generated using a fisherfaces model with 72 training images. The test data is the entire Caltech dataset.

3 RESULTS AND DISCUSSION

3.1 TRAINING THE MODELS

The models are trained with 72 images of 16 people that have been assigned different ID-numbers. The remaining people are unknown, and the program should recognize them as such and return an ID-number of 0. The training images were chosen to try and cover a wide range of facial expressions and illumination conditions. The program can correctly detect all known faces in the entire Caltech database except for three, which means that these three images could not be chosen as training images.

The number of training images is limited to 72 because otherwise there would be less images left to test the program with. This is because dimensionality-reduced subspace coordinates (weights) are stored for each training image, which means that training images are more or less guaranteed to be recognized correctly, regardless of how well the program actually performs. The test images have never been seen by the program prior to testing and no weight information about them are stored.

The training resulted in 15 base vectors in the fisherfaces model and 25 base vectors in the eigenfaces model. The fisherfaces distance threshold was found to be 24, while the eigenfaces distance threshold was found to be 16.9. These distance thresholds maximize the total accuracy when the models are tested on the entire Caltech dataset.

3.2 ACCURACY

Of the remaining 378 images that the program hasn't seen before, the program is able to correctly detect and recognize 98.68% of the faces using the fisherfaces model, either with the ID corresponding to the person or 0 if the person is unknown. This could be increased to about 99.2% with more training images, but no more since the detection fails in three extremely underexposed images. The eigenfaces model only correctly recognizes 91.53% of the faces, despite using more base vectors than the fisherfaces model. These results with corresponding false acceptance and rejection rates are listed in Table 1.

Table 1 - Final results.

Model	**Accuracy**	**FRR**	**FAR**
Fisherfaces	98.68%	1.32%	0.00%
Eigenfaces	91.53%	7.67%	0.79%

The training images are not included in these numbers as these would skew the results, given that their exact subspace coordinates already are known by the model. False rejection rate (FRR) refers occurrences where the program doesn't recognize a known face, while false acceptance rate (FAR) refers to occurrences where the program thinks that it recognizes face that is either unknown or belongs to a different known person.

The 5 images that accounts for the 1.32% false rejection rate of the fisherfaces model are shown in Figure 9.

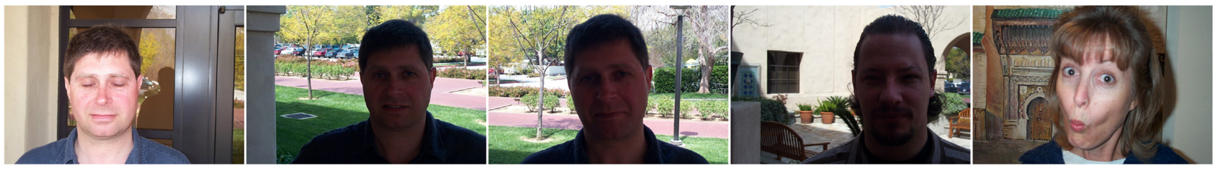

Figure 9 - All images that were falsely rejected by the fisherfaces model

The first and last image are detected correctly, but the model hasn't been taught the extreme facial expressions. This places them at a distance further away than the threshold distance to any known face. They are however still placed closest to the correct person, but increasing the threshold distance to include them would instead introduce false acceptances. Since the eyes are closed in the first image, the program isn't quite able to locate the exact center of the eyes, but it still finds them with descent accuracy using the eyelashes. This aligns the face slightly wrong however which also affects the distance. The remaining three images could simply not be detected because there are barely any color left in the faces due to the extreme underexposure and poor dynamic range.

3.3 Speed

The detection phase of the program is quite slow and takes a few seconds to complete. Surprisingly, the function that accounts for most of this time (by far) is the built-in *chromadapt* function, which applies the white balance correction on the image using the computed scene illuminant. Changing the method that this function uses to the '*simple*' method, which just divides each pixel with the scene illuminant, speeds this up by several orders of magnitude. This reduces accuracy slightly however, which is probably caused by the fact that the program has been configured using another method throughout the development process more so than the method itself. I can't imagine that the slow speed of the default method is anything other than a bug however, which might be fixed in a future release.

Another contributor to the slow speed is the fact that the entire detection pipeline runs 4 or 8 times per input image. This is however completely parallelizable which could improve the speed 4 or 8 times if as many concurrent threads are available on the system.

4 References

[1] D. Cheng, D. K. Prasad och M. S. Brown, ”Illuminant Estimation for Color Constancy: Why spatial-domain methods work and the role of the color distribution.,” 2014.

[2] R.-L. Hsu, M. Abdel-Mottaleb och A. K. Jain, ”Face Detection in Color Images,” 2002.

[3] B. Wang, X. Chang och C. Liu, ”Skin Detection and Segmentation of Human Face in Color Images,” 2011.

[4] J. M. Chaves-González, M. A. Vega-Rodríguez, J. A. Gómez-Pulido och J. M. Sánchez-Pérez, ”Detecting skin in face recognition systems: A colour spaces study,” 2009.

[5] M. Turk och A. Pentland, ”Eigenfaces for Recognition,” 1991.

[6] P. N. Belhumeur, J. P. Hespanha och D. J. Kriegman, ”Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection,” 1997.

[7] N. Muller, L. Magaia och B. M. Herbst, ”Singular Value Decomposition, Eigenfaces, and 3D Reconstructions,” 2004.