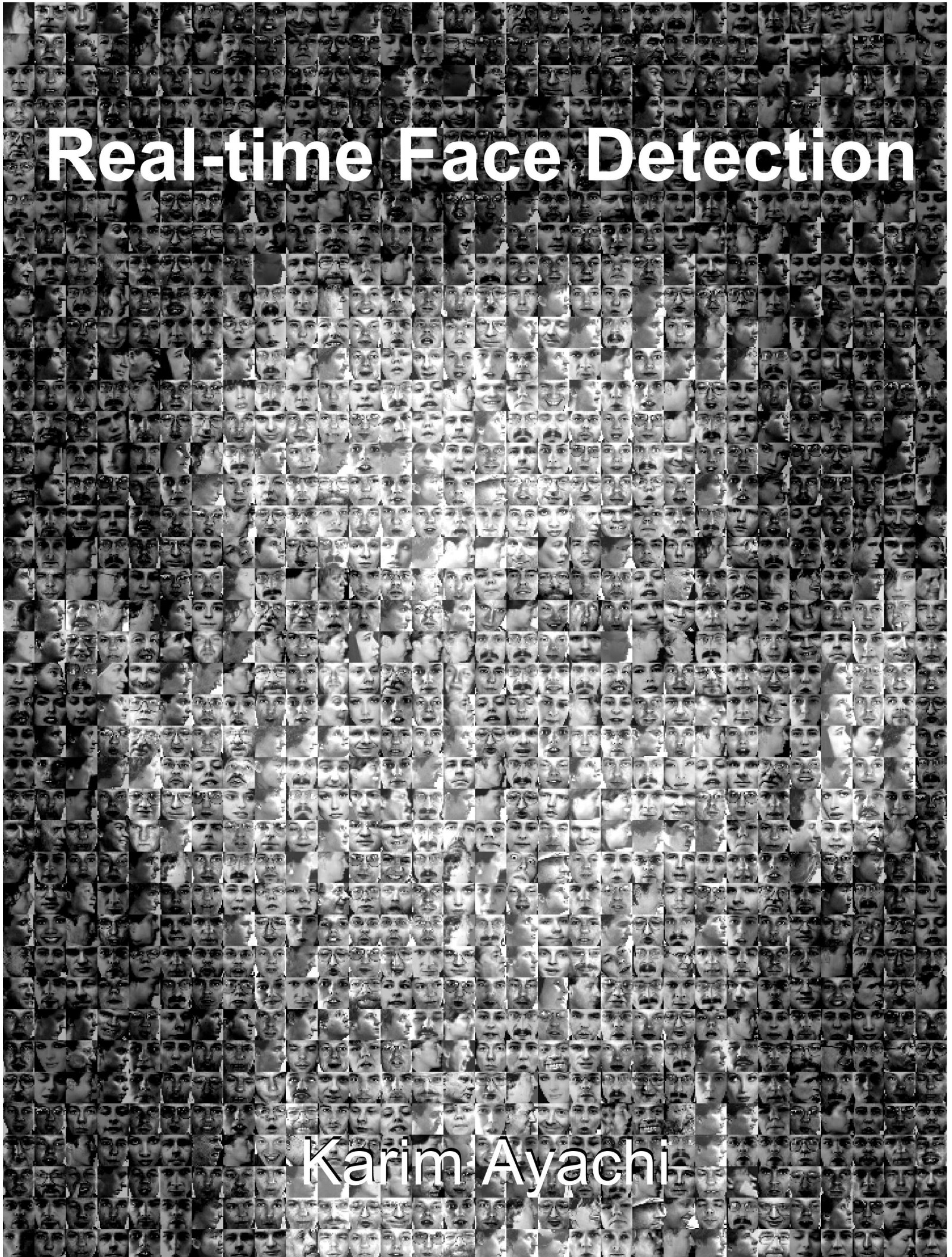
Real-time Face Detection
Karim Ayachi

University of Amsterdam
August 4th 2004
M.Sc. Thesis

Real-time Face Detection

Karim Ayachi

Supervised by: B.J.A. Kröse

Contents

Chapter 1

Introduction

Face detection plays an important role in the development of many autonomous cognitive systems. Both at a low and a high abstraction level. It is an indispensable tool in creating user interfaces for intelligent systems that interact with humans, but it also is an essential framework for more complex systems. Face detection can be a first step for, among others, autonomous face recognition or surveillance systems.

Face detection has been a field of research since the early nineties as a sub field of pattern recognition or object detection and many schemes have since been proposed. All the difficulties that exist in the domain of object detection exist in face detection. Among these problems are illumination differences and scaling and rotation of the object. In addition to these common problems there are also difficulties unique to the domain of face detection. These problems include differences in facial expressions and the great variety in facial features such as glasses or facial hair. Another problem in most current systems is the detection of non-frontal or profile faces.

Our objective in this research is to examine the different approaches to the problem and implement a basic face detection system that can serve as a reference point or a toolkit for future research. Also we will examine the details of such a system in order to possibly extend the existing schemes to incorporate non-frontal faces or to increase the performance. Furthermore we will examine the effects of the size of the training set, since most existing methods are based on learning algorithms that use datasets for training.

In **Chapter 2** we will give an overview of some of the current methods of face detection. **Chapters 3** till **5** describe more closely the method we chose to follow. **Chapter 6** gives an in-depth view of our implementation and the extensions to the system. Finally in **Chapter 7** we present results to our experiments and an evaluation of the face detection method.

Chapter 2

Face Detection Overview

In face detection, input pictures are searched for the occurrence of faces. Existing methods can be grouped into model-based and template-based methods. The model-based approach uses facial features, mostly relying on a priori knowledge. These features can be lines and curves describing the contours of the face or skin color. In template-based methods a window frame or detector is run across the image, trying to match the current sub window to a template. This detector has a fixed size and represents the current portion of the image to be evaluated. At each location the detector has to label the sub window as a face or non-face. To do so, the detector includes one or more classifiers. The classifiers can be neural networks, simple linear classifiers, statistical methods using Principal Component Analysis, etc. They are usually trained using a set of faces and a set of non-faces of which certain features are used as input for the classifier. Features can be, among others, wavelets[1] or raw pixel data. During detection these values are also used to classify the sub window.

Once an image has been scanned, the scale has to be altered in order to detect faces that are differently sized. Usually this is done by using a small, fixed size detector and scaling the input image down after it has been scanned, then repeating the whole process until the input image is small enough for the detector to detect even the largest faces. (see figure 2.1)

2.1 Methods

While our method, as described in the next chapters, is template-based, I will give a modest overview of 3 of the most widely used template-based methods: neural networks, support vector machines and the real-time approach of Viola and Jones.

[1] The simplest of wavelets, the Haar wavelet is the basis for the Haar-like filters explained in chapter 3. Many other types of wavelets exist, of which some, such as Gabor wavelets are also frequently used in the field of object detection.

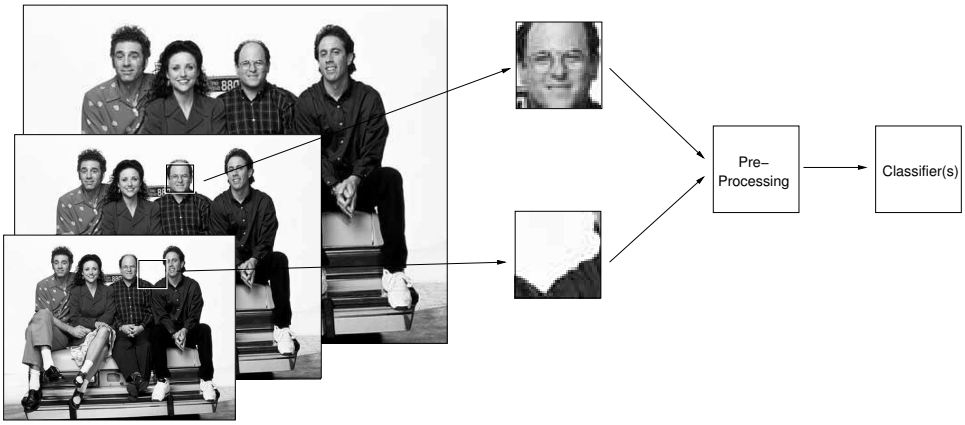

Figure 2.1: Schematic representation of face-detection. The input image is scaled to different sizes. This is called a pyramid. The detector or window frame is run across the input pictures. Pre-processing is usually applied to the sub-window and then it is evaluated by the classifier(s).

2.1.1 Neural Networks

Rowley et al. [6] propose a template-based method based on training a neural network. They use the framework of figure 2.1 with a detector window frame of 20×20 pixels. Each of the sub windows marked by the detector window is pre-processed to correct extreme lighting conditions and histogram equalized to correct for different camera gains and to improve contrast. This sub window is than fed to the neural network. The network has 3 different types of hidden units. One type of 4 units that look at the 4 10×10 sub regions. One that has 16 units that look at the 16 5×5 sub regions finally 6 hidden units that look at 6 overlapping 20×5 pixel horizontal stripes. (see figure 2.2) These hidden units are combined to give an output -1 for non-face and 1 for face windows.

A dataset with approximately 1000 faces is used to generate 15 samples of each original image by randomly scaling, translating and rotating by small amounts. The domain of the non-face images is far more complex than that of the face images. Features of these images are almost random, the only thing they have in common is the absence of facial features. So for the non-face images, 1000 random images are generated out of scenery pictures not containing any faces. The network is trained using this collection of samples.

An adapted version of Sung and Poggio's bootstrapping [9] learning algorithm is used to further train the network: the system is run on a number of images not containing any faces. Sub images that are incorrectly classified as a face are collected and added to the training set as negative examples.

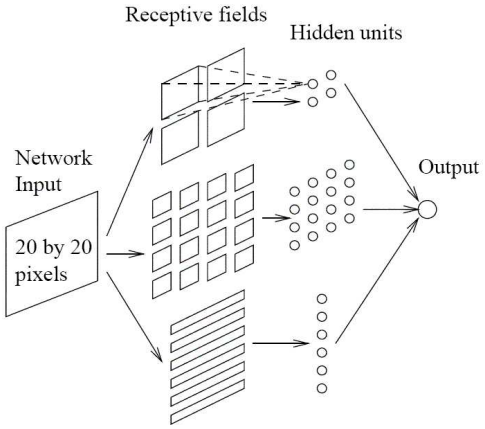

Figure 2.2: The 3 types of hidden units in the neural network. Each looking at different sub regions of the window frame.

2.1.2 Support Vector Machines

Osuna et al. [8] use the same template-based framework as Rowley et al., but with a Support Vector Machine (SVM) instead of a neural network. SVMs are a new pattern classification algorithm. While other techniques try to minimize the training error, SVMs minimize an upper bound on the generalization error. In the domain of face detection, the input sub window is mapped onto a high-dimensional vector. Osuna et al. use a detector frame of 19×19 pixels. They use the same pre-processing as Rowley et al. and an extra binary pixel mask that filters out some of the pixels close to the border of the frame, so reducing the dimensionality of the input space. The SVM is trained with a 2nd degree polynomial classifier, relying on Sung and Poggio's bootstrapping algorithm [9].

2.1.3 Viola and Jones

The final template-based method we will describe here is the one proposed by Viola and Jones [1].Their detector is based on simple linear classifiers that work not on raw pixel data, but on so called features. They are simple local binary features, that are specifically good at detecting high contrast areas. In face detection these areas occur at the bridge of the nose, around the eyes, etc. Viola and Jones introduce an image representation based on sums of pixels they call the 'integral image', allowing the features to be calculated very fast. An adapted version of AdaBoost is used to select and combine the best classifiers out of a huge selection. Finally the 'attentional cascade' is introduced. This cascade divides the set of classifiers in multiple stages of a tree-like structure. Again the 'Bootstrapping' algorithm is used to train

the subsequent stages. This layering of the classifiers makes it possible to do detection in real-time while still achieving a performance comparable to the other main face detection systems.

2.2 Conclusion

As said, we have chosen to use a template-based method. These methods don't rely on prior knowledge of the domain and thus are suitable for unsupervised learning. The main problem with template-based methods is that they are very sensitive to rotation and scaling. Scaling can be done by scaling the input picture, as done in [6] and [8] or by scaling the detector window frame, as done by [1]. Rotation however is much harder to compensate for. An extension for the neural network approach was proposed by Rowley et al. [11] where an extra neural network is used in the detector, that 'guesses' the angle of the sub window and de-rotates accordingly. This only works for in-plane rotations and does not provide a solution for out-of-plane rotations (profile faces). Viola and Jones extend their method by training different detectors for different poses and use a decision tree to call the appropriate detector [10].

The method of Viola and Jones is the only one able to achieve real time performance and yield results comparable to the best non real time systems. Also when using different cascades for different poses, the system could deliver robust multi-view detection, while still remaining real time. As we are interested in real time applications of the face detection problem, we have implemented the Viola and Jones method. In the next chapters we will describe it in more detail and give results of our experiments with the basic system and our extensions.

Chapter 3

Features

The main ingredient of Viola and Jones' work [1] is the use of a simple feature detection system. A 'detector window' is run across the input picture, searching for faces. At each location the current window frame is analyzed by examining a set of local features. The actual local features used are very similar to Haar Basis functions [4]. Combined with the use of a so called 'integral image' calculating these features can be done extremely fast. Equally important, the time it takes to evaluate a classifier based on these features is constant and not related to size or position.

3.1 Fast Haar-like Features

Haar features consist of 2 or more adjacent rectangles in the detector window frame. The sums of the pixels within the rectangles are subtracted from each other. This way it is possible to detect contrast differences. A dark area next to a light area will give a distinct value, whereas areas that area are equally bright will give values close to 0. Viola and Jones show that this works well in the domain of face recognition where there are many of these contrast-rich areas. E.g. the bridge of the nose or the eyebrows. Viola and Jones extend the Haar feature set to 5 different categories (figure 3.1).

Features can be represented in a way described by Lienhart.[2] The whole area of the feature represents the 'white' area in which the 'gray' areas reside. The white area is given a weight of -1 and the gray areas are given a positive weight according to the percentage of the total area they occupy. The feature value is computed by summing the weighted pixel values:

$$f = \sum_{r=1}^{R} \left(w_r \sum_{x \in F_r} x \right), \tag{3.1}$$

where R is the number of rectangles in the feature, w is the weight of the rectangle, F_i is the i-th rectangle in the feature and x is a pixel value. For

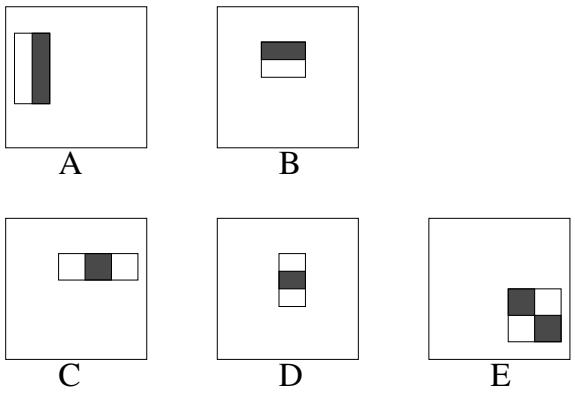

Figure 3.1: The 5 basic feature rectangles. The sum of the pixels in the gray areas are subtracted from the sum of the pixels in the white areas. C and D are Viola and Jones' extensions to the Haar functions

example, the value of a feature of type A (figure 3.1) with one white and one gray rectangle, both equal in size is $-1 \sum_{x \in F_{total}} x + 2 \sum_{x \in F_{gray}} x$.

The standard detector window frame has a base resolution of 24×24 pixels, but can be scaled to any size and translated anywhere in a picture. The total number of features of all 5 types scaled and translated across all possible sizes and locations within the 24×24 pixels is 162,336. As in the OpenCV implementation[15] there is a minimum size of 32 pixels for the features. This speeds up learning, because with this restriction the total set only has 108,490 features. Very small features can never be very good classifiers, but can accidentally perform well in the AdaBoost algorithm (see chapter 4). So by eliminating these small features, noise is reduced.

3.2 Classifiers

As will be described in chapter 4, the face detector is constructed by combining simple classifiers using a Boosting algorithm to form one robust classifier. Each of these simple classifiers operate on a single feature. These classifiers are trained using a nearest mean method.

The positive training set consists of a collection of 24×24 pixels pictures of faces. The negative set is randomly generated by taking random subwindows from a collection of pictures not containing any faces and scaling them to 24×24 pixels (see 7.1).

Per feature the images in the positive and negative sets are evaluated and an average value of both the positives and the negatives is calculated (the means μ_A and μ_B). A threshold θ is then set to the average of these positive and negative means. If the positive mean is smaller than the threshold,

the classifier must return 1 for values smaller than the threshold and 0 otherwise. If the positive mean is larger, 0 is returned for values smaller then the threshold and 1 otherwise.

$$h(x) = \begin{cases} l & f(x) < \theta \\ r & f(x) > \theta \end{cases}, \tag{3.2}$$

where h is the classifier on input sample x and $f(x)$ is the feature-value of sample x.
$l = 1$ if $\mu_A < \theta < \mu_B$, 0 otherwise
$r = 1$ if $\mu_A > \theta > \mu_B$, 0 otherwise

3.3 Integral Image

In order to achieve real time detection, these classifiers must be evaluated very fast. Viola and Jones introduce the so-called 'integral image' [1]. This image representation allows for the Haar features to be calculated extremely fast. The integral image is similar to the Summed Area Table used in computer graphics and is in many publications referred to as SAT. The integral image is constructed by taking for (x, y) the value of (x, y) of the original picture and adding the sum of all the pixels above and to the left.

$$(x, y)_{integral} = \sum_{x' \leq x, y' \leq y} (x', y')_{original} \tag{3.3}$$

When this integral image is constructed, the feature rectangles can be calculated very easily because the image already contains information about summed areas. Calculation the pixel sum of any rectangle requires only 4 array references (see figure 3.2). The features consist of 2 or 3 of such rectangles, so on average about 10 array look-ups are needed per feature, no matter the size or place of the rectangles.

3.4 Variance Normalization

To make detection more robust, the samples are normalized to compensate for different lighting conditions. For each training sample the variance of the pixel-values in the detector window is calculated. $\sigma^2 = \frac{1}{n}\sum x^2 - \mu^2$. σ^2 is the variance, n is the number of pixels in the detector window, x is a pixel-value and μ is the mean of the pixel values. Then each pixel is normalized:

$$x' = \frac{(x - \mu)}{2\sigma}, \tag{3.4}$$

where σ is the standard deviation, the square root of the variance. Each pixel now has a value roughly between -1 and 1. This is multiplied by 128 and 128 is added to get them back in the range of 0-255. Values are clipped

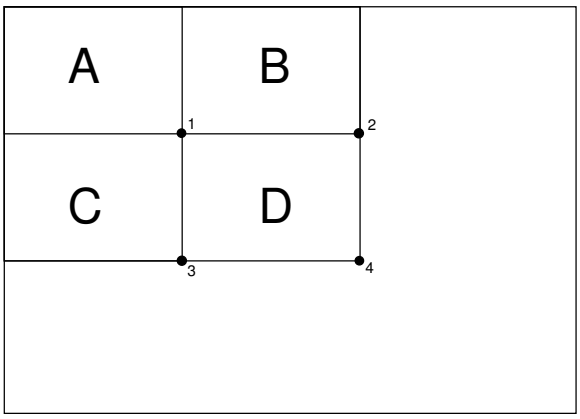

Figure 3.2: Calculating the rectangle sums using the integral image. The sum of D can be obtained by subtracting 2 and 3 (representing the sums of A+B and A+C) from 4 and 1 (representing the sums of A+B+C+D and A). D=4+1-(2+3).

at 0 and 255. The constant 2 is chosen in correspondence with the work of Lienhart and gives good results with bringing the values between -1 and 1. Higher or lower values result in a higher or lower contrast.

3.5 Conclusion

Using these simple features in combination with the integral image, Viola and Jones have created a framework for detectors that can operate fast enough to make real time detection possible. On the downside, classifiers based on these features are far too simple to yield an acceptable performance. In order to achieve reasonable detection rates, a great number of these classifiers are necessary. In the next chapter we will explain how Viola and Jones use an adapted version of the AdaBoost boosting algorithm in order to select and combine the best classifiers out of the total set.

Chapter 4

Boosting

Using the simple feature classifiers of chapter 3 and achieving robust detection would require a large amount of these classifiers, because the individual classifiers are too simple to perform well by themselves. As we discussed in the previous chapter, even in a 24×24 window there is a huge collection of unique features. Calculating such a large number of functions everywhere in a picture would, although they are fast, still take up too much time for real-time applications.

The number of features that are evaluated at each location has to be reduced. After training the classifiers a subset has to be selected. There are a couple of algorithms that select and combine the best classifiers such as bagging or boosting algorithms. Viola and Jones use an adapted version of the AdaBoost algorithm to combine classifiers based on the best features into a single powerful classifier.

4.1 Boosting

Simple classifiers, such as the local feature-based classifiers of chapter 3, are not capable of yielding good results by themselves. They are called 'weak' classifiers. Boosting is a method of combining a set of weak classifiers to form a 'strong' classifier. The standard boosting method is AdaBoost (Adaptive Boosting). (Ada)Boosting is a deterministic procedure and selects classifiers from the total set sequentially. Selecting the classifiers is based on the results of the previous iteration: more emphasis in the

next iteration is placed on areas that have high error rates in the current. This is done by assigning all training samples a normalized weight. During each iteration of the algorithm, a classifier $\eta_t(x)$ is constructed using these weights. Then the weights are updated, increasing the values of misclassified samples and decreasing the values of correctly classified samples. This way, in the next iteration the algorithm focuses more on the samples that are misclassified. These samples seem 'harder to learn' or, more correctly, are

nearest to the decision boundary.

The error ϵ_t is calculated, based on the sum of the weights of the misclassified samples. These errors are 'boosted' by a factor $(1-\epsilon_t)/\epsilon_t$ to put more emphasis on the misclassified samples. The constructed classifiers are combined using a linear weighting. Table 4.1 show the AdaBoost algorithm as it appears in [12].

- Initialize the weights $w_i = 1/n, i = 1, \ldots, n$.
- For $t = 1, \ldots, T$,
  - construct a classifier $\eta_t(x)$ from the training data with weights $w_i, I = 1, \ldots, n$;
  - calculate ϵ_t as the sum of the weights w_i corresponding to misclassified patterns;
  - if $\epsilon_t > 0.5$ or $\epsilon_t = 0$ then terminate the procedure, otherwise set $w_i = w_i(1-\epsilon_t)/\epsilon_t$ for the misclassified patterns and re-normalize the weights so that they sum to unity.
- For a two-class classifier, in which $\eta_t(x) = 1$ implies $x \in \omega_1$ and $\eta_t(x) = -1$ implies $x \in \omega_2$, form a weighted sum of the classifiers η_t,

$$\hat{\eta} = \sum_{t=1}^{T} log(\frac{1-\epsilon_t}{\epsilon_t})\eta_t(x)$$

and assign x to ω_1 if $\hat{\eta} > 0$

Table 4.1: The standard AdaBoost algorithm

4.2 Using AdaBoost for feature-selection

Viola and Jones use an adapted version of the AdaBoost algorithm to select and combine the best of the 108,490 trained weak classifiers. As with the standard AdaBoost algorithm each of the training samples is assigned a normalized weight. The training set consists of a positive set of images with hand-labeled faces and a negative set, generated using random portions of images out of a negative set of pictures not containing any faces. With n faces and m non-faces, the positive samples get assigned the weight $1/2n$ and the negative samples get assigned the weight $1/2m$. Then all trained classifiers are evaluated and their individual errors are calculated based on

the weights of the misclassified samples. The classifier with the lowest error is chosen.

The weights of all samples are adjusted according to the error of the chosen classifier, so that when selecting the next classifier, more emphasis is placed on the misclassified samples. These steps are then repeated until enough classifiers are collected (table 4.2).

- Given n positive and m negative samples $x1, \ldots, xn, xn{+}1, \ldots, xn{+}m$.
- The weights of the images are initialized to $\frac{1}{2n}$ for the positive set and $\frac{1}{2m}$ for the negative set. This is a probability distribution.
- For $t = 1, \ldots, T$:
  - The weights are normalized so that they sum to unity:

$$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n+m} w_{t,j}}$$

  - Calculate the error of all the weak classifiers by evaluating each one against all images in the training set and adding the weight of misclassified images to the overall error ϵ_j of the classifier h_j.
  - Choose the classifier h_t with the lowest error.
  - Update the weights that are classified correctly inversely proportional to the error of the chosen classifier:

$$w_{t+1,i} = w_{t,i}\beta_t^{1-\epsilon_i},$$

    where $\epsilon_i = 0$ if sample x_i is classified correctly and 1 otherwise. $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- The final strong classifier is:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

  where $\alpha_t = \log \frac{1}{\beta_t}$ and h_t is the t-th classifier chosen. The threshold $(\frac{1}{2} \sum_{t=1}^{T} \alpha_t)$ is the standard threshold, but better results are gotten when it is adjusted.

Table 4.2: The AdaBoost algorithm for selecting the classifiers based on the best features

4.3 Conclusion

By using boosting to select and combine the best weak classifiers a strong classifier can be created. Viola and Jones show that classifiers can be constructed that are 'strong' enough to make a robust face detector. In this way we can use the classifiers that are based on the simple local features described in chapter 3. Recall that these features can be calculated with few operations. So by combining a subset of the total number of 108,490 features into a strong classifier, we have a robust detector that can be evaluated very fast. However, to get results that are comparable to other widely used face detection methods, still a large number of classifiers are needed, although the number is greatly reduced by the AdaBoost algorithm. Even with these fast features it would take too much time to calculate each one on every location. To limit the number of classifiers that have to be evaluated every time, Viola and Jones introduce the so called 'Attentional Cascade'.

Chapter 5

Cascade

As said in the previous chapter, even when Boosting is used to select the best features, in order to achieve robust detection still a great amount of classifiers is needed. The idea is that every sub window (the portion of the image currently marked by the detector window frame) has to be classified as fast as possible. To be sure that a window contains a face, all the classifiers have to be evaluated, but it is possible to classify a sub window as non-face after evaluating only a few classifiers. Viola and Jones show that a boosted classifier of 2 weak classifiers can be adjusted to detect almost all faces and reject 60% of the non-faces. The rejected sub windows need no further processing so this can be used to dramatically reduce the number of sub windows that have to be further evaluated. They have implemented the idea in the so called 'Attentional Cascade'. The Cascade consists of layers of classifiers - the stages. A negative outcome of any stage results in the dismissal of the current sub window. A positive outcome triggers the next stages. A sub window is classified as containing a face when all stages return a positive outcome.

5.1 Performance

The error consists of missed detections (false negatives) and sub windows incorrectly classified as a face (false positives). The detection rate is the percentage of faces correctly detected (true positives). In order to have a robust detector, the detection rate must be very high (low false negative rate). Around 0.9 (90%) is considered acceptable. More important, the false positive rate must be very low. Considering that, using the standard settings that will be described in the next chapter, an average input picture of 384×286 pixels contains approximately 150,000 sub windows to be evaluated, a seemingly reasonable false positive rate of 0.1% still results in 150 false detections per picture. False positive rates have to be in the order of 10^{-6} to drop below 1 false detection on average per image.

Viola and Jones define the false positive rate and the detection rate of a trained cascade as:

$$F_{total} = \prod_{i=1}^{S} F_i, \tag{5.1}$$

where F_{total} is the false positive rate of the cascaded detector, S is the number of stages in the cascade and F_i is the false positive rate of the ith stage on the samples that get through to it. The same is defined for the detection rate:

$$D_{total} = \prod_{i=1}^{S} D_i, \tag{5.2}$$

where D_{total} is detection rate of the cascaded detector, S is the number of stages in the cascade and D_i is the detection rate of the ith stage on the samples that get through to it. Recall that each stage is a boosted 'strong' classifier and thus has a detection and error rate just like a conventional classifier. If we set a target detection rate and a target false positive rate for the cascade, we will have to consider what this means for the individual stages. To achieve a detection rate of 0.9 with a 10 stage cascade, each stage will have to have a detection rate of 99% ($0.99^{10} \approx 0.9$). This may seem hard to achieve, but the false positive rate of each stage only has to be 30 % ($0.33^{10} \approx 6 \times 10^{-6}$).

5.2 Creating the Cascade using AdaBoost

The AdaBoost algorithm described in table 4.2 can be used to train the layers of a cascade. The standard AdaBoost threshold ($\frac{1}{2}\sum_{t=1}^{T}\alpha_t$) however will not give the best results here, as this threshold gives an overall low error rate and we are more interested in minimizing the false positive rate, while still achieving an acceptable detection rate. Since we're aiming at real time application, speed is one of the biggest constraints. Every sub window has to be qualified as soon as possible. Considering that almost 100% of the sub windows is non-face this can be translated to: every sub window has to be rejected as soon as possible. Also considering that the detector has several tries to detect a face (different scales and locations around a face), we can conclude that minimizing the false positive rate is more important than maximizing the detection rate. However, to achieve acceptable performance, still almost all faces have to be detected.

Better than using a fixed threshold per stage, we let the algorithm determine the optimal value. To do this, we set a target detection rate and a target false positive rate for every stage. AdaBoost then selects the first classifier(s) for a stage. The threshold of the stage is initially set to its maximum ($\sum_{t=1}^{T}\alpha_t$, the sum of the return values of all classifiers. I.e. all classifiers must qualify the sub window as a face). Then slowly the threshold

is lowered until the target detection rate for that stage is met. If at that time the false positive rate is still greater than the target false positive rate, AdaBoost is called again and an extra classifier is added to the stage.

Reducing the threshold only makes sense if the step size is very small. This slows down learning considerably because the stage has to be evaluated every time the threshold changes, which can be many 10,000s of times. We use a line search method that changes the threshold on average 25 times before finding the optimal value (see table 5.1). This way time is only spend on the AdaBoost algorithm and is linear to the number of classifiers.

Once a stage has the right detection rate and false positive rate, a new negative set is created by again randomly selecting sub windows from a collection of non-face pictures, as during boosting explained in chapter 4, evaluating them with the unfinished cascade and using only the false positives. This 'Bootstrapping' algorithm is used in many of the face detection schemes such as the neural network approach of Rowley et al.[6] and the support vectors of Osuna et al.[8]. The next stage is constructed using this new set of negative images in the AdaBoost algorithm. An independent validation set with non-face images is used to determine the false positive rate.

- Set Detection Rate D and False Positive Rate F for this stage.
- While False Positive Rate $\leq F$
  - Add classifier to the stage using AdaBoost
  - Set $lowerBound = 0$, $upperBound = \sum_{t=1}^{T} \alpha_t$.
  - Set $threshold = (lowerBound + upperBound)/2$
  - Until $upperBound$ - $lowerBound \approx 0$
    - If Detection Rate $\leq D$ then $upperBound = threshold$, $threshold = (lowerBound + threshold)/2$
    - If Detection Rate $\geq D$ then $lowerBound = threshold$, $threshold = (upperBound + threshold)/2$
- Generate a new negative set, using false positives from the unfinished cascade

Table 5.1: The algorithm for creating a stage in the Cascade

5.3 Conclusion

With the addition of the Attentional Cascade to the rest of their work, Viola and Jones have tied together all aspects of a real time face detector. Where the integral image helps calculate the local features very fast, the boosting helps to make it robust and finally the cascade is an elegant way of reducing the number of classifiers generated by the AdaBoost algorithm.

Chapter 6

Our Implementation and Extensions

Based on the reported performance of Viola and Jones method we have decided to examine this system more closely. The first step was to implement the basic face detector and run some experiments to test the performance. The next step was to extend the system. In [2] and [10] extensions are proposed. Both introduce new features, aiming at detecting not only frontal faces, but also non-frontal and in-plane rotated faces. [10] also combines the current system with a tracking algorithm that would make the system robust under occlusion.

In the next sections we will describe our implementations of the basic system and our extensions. I.e. the training algorithm for the classifiers, our grouping and post-processing algorithm and our work on non-frontal faces.

6.1 Training

We have trained the 108,490 classifiers, each based on a single local feature and one threshold. Training is done by using the simple nearest mean algorithm described in section 3.2. When the Gaussian distributions of both classes have an equal variance, the nearest mean approach will return an optimal threshold. However in the case our face detection problem, the variance of the positive dataset that are the faces probably will not be same as the variance of the random non-faces. We expect that face data is low-variant and non-face data is high-variant (as will be described in section 7.3.1, where we will also present results for a different type of classifier). The threshold is slowly moved towards the positive mean. At every step the classifier is reevaluated and the threshold is set when the performance stops increasing.

6.2 Detection

During detection the input image is searched for the occurrence of faces. The input image is converted to an integral image and a detector window frame is run across the image. At each location the detector classifies the current sub window using the trained cascade. Each stage of the cascade is defined by a number of weak classifiers with an individual threshold and a weight (see chapter 4). There also is a threshold for the whole stage (see chapter 5).

The image is scanned n times, each time the detector frame is a factor s larger. The detector window frame of $s^n \times (24 \times 24)$ pixels is translated $d \times s$ pixels across the image. Common values for these variables are $n = 11$, $s = 1.25$, $d = 1.5$ and a starting scale of 1.25. With these values a typical image of 384×286 pixels has 149,239 sub windows that have to be evaluated.

At every one of these locations the stages of the cascade are evaluated. If a stage rejects the detection window, the detector moves on to the next location and/or scale, else the next stage is evaluated. If none of the stages reject the window, the location is marked as containing a face. Windows are rejected when the sum of return values based on their weight of the individual single-feature classifiers is smaller than the threshold generated by (5.1). To achieve high speed recognition and to exploit the idea behind the cascade to the fullest, it is important that windows are rejected as soon as possible. Therefore the first stages should have as high thresholds and as little classifiers as possible, while still having an almost 100% detection rate.

6.3 Variance Normalization

While the system was trained using normalized samples, during detection the sub windows would also have to be normalized. Following the scheme of equation 3.4 during detection would take too much time. Recalculating every pixel in every sub window can not be done in real-time. However, the mean μ of a sub window can be calculated easily using the integral image. The total pixel sum of a sub window can be obtained directly from the integral image. Divided by the number of pixels we get μ. To do the same for the sum of squared pixels, Viola and Jones introduce a squared integral image. Before detection a second integral image is made, containing the sum of the pixels squared. Recall that variance σ^2 is defined as: $\sigma^2 = \frac{1}{n}\sum x^2 - \mu^2$ Using this new squared integral image we can calculate $\frac{1}{n}\sum x^2$ with only 4 additional array references.

Each feature rectangle contains the sum of the pixels within and each pixel should have the normalized value:

$$x' = \frac{(x - \mu)}{2\sigma} \times 128 + 128, \tag{6.1}$$

as defined in equation 3.4. Likewise the sum of these normalized pixel values is:

$$\sum x' = \sum \left(\frac{(x - \mu)}{2\sigma} \times 128 + 128 \right) \tag{6.2}$$

The sum in this equation is the sum of the pixels in the rectangle. Since the mean and the standard deviation are the same for all pixels within the sub window and the rest are constants except for x, this can be written as

$$\sum x' = \frac{(128 \sum x) - (128 n \mu)}{2\sigma} + 128n \tag{6.3}$$

x is a pixel value, so $\sum x$ is the feature value of the rectangle, n is the number of pixels in the rectangle. x' and $\sum x'$ are the normalized pixel- and feature- values. This way, normalizing can be done in constant time.

6.4 Grouping and Post-processing

After the detection algorithm has marked the location of the faces, usually the same faces are detected multiple times. This happens because the detector is robust under small changes in scaling and translation. These detections have to be combined to form one or more distinct detections. (see table 6.4).

- Create an empty set of unique detections $\mathbf{S} = \{\}$
- For $j = 1, \ldots, J$, where J is the number of detections:
  - $\forall i \in \mathbf{S}$:
    if Euclidian distance between the center of j and the center of $i < \theta$ then:
    'add j to i': making i a new mean of the previous value(s) of i and j
  - if j matches to no i, then add j to $\mathbf{S}$, creating a new unique detection

θ is based on the size of j multiplied with a constant.

Table 6.1: The algorithm for grouping detections

When we have this new set of unique detections we can use this information to do some post-processing. If we know how many original detections are responsible for creating one unique detection and we know that faces are

likely to generate multiple detections, we can filter out some more false positives. False positives are not likely to trigger the detector many times, unless the sub window really resembles a face. We do this by ignoring detections that have triggered the detector proportionally few times:

$$\text{accept } i \text{ if } \frac{x_i}{J} \geq \frac{1}{I}, \tag{6.4}$$

where i is the current unique detection, x_i is the number of detections responsible for unique detection i, J is the total number of detections and I is the number of unique detections.

This scheme works very well when there are one or two faces present in the scene, but could have unwanted results when there are a lot of faces present. Then the faces that are easily detected generate more detections and faces that are harder to detect could be filtered out.

6.5 Non-profile faces

The extensions mentioned in the introduction to chapter 6, [2] and [10] are based on an extended feature set. For in-plane rotations such a set would be indispensable, since rotated faces lack the simple horizontal and vertical features that upright faces have, such as the nose or the eyes. However we decided to test the basic feature set on the domain of profile faces. Although profile faces have less clearly defined features than frontal faces (and are therefore more difficult to detect), the facial features could quite possibly be covered by the basic feature set.

To test this, we have trained our detector with a set of profile faces. For the negative faces we have used the same bootstrapping method with the same non-face pictures as for the frontal faces. Results will be presented in chapter 7.

For the sake of the experiment, we have created a single profile face detector next to our frontal face detector. However in a practical situation, all detectors would have to be combined to form one robust all-round face detector. This means that each window frame in a picture has to be evaluated by all detectors. In our case these would be three different detectors: one for frontal detection, one for profile faces facing right and one for profile faces facing left. To run different detectors sequentially imposes a great strain on speed, especially when even more detectors are used for in-plane rotated faces. Alternatively a top level classifier could determine the detector that has to be called for each window frame. Although this would be faster, it also would be less accurate. Viola and Jones show in [10] that using such a top level classifier gives very good results and accuracy-loss is negligible.

6.6 Conclusion

Bringing it all together, we have succeeded in creating an extendible framework for real-time face detection purposes, based on the method of Viola and Jones. In the next chapter we will present results of our detector and discuss the weak and strong points.

Chapter 7

Experiments

7.1 Objective

The objective of these experiments is to examine more closely the advantages and disadvantages of the Viola and Jones detector. As discussed in chapter 5, false positive rates have to be extremely low in order to produce a usable face detector. We present Receiver Operating Characteristic curves for our implementation that show that false positive rates are achievable that are acceptable while still having a high enough detection rate. We have used a dataset containing 2021 faces. In order to find an optimal size of the training set, we studied the effect of altering the size of the set on the performance. In section 7.3.3 we present our findings. An important constraint on the system is execution time, since we are aiming at real time detection. In section 7.3.4 we show that our system is fast enough for real time detection and compare it to the speed reported by Viola and Jones and the OpenCV[15] implementation available on the internet.

7.2 Set up

Our frontal faces detector is a 12 stage cascade. The first layer contains 2 feature classifiers (see figure 7.1). This layer has a detection rate of 99% and rejects 74% of the non-faces. The total number of classifiers in the cascade is 1062. On the face dataset, the detector has a detection rate of 0.89 and a false positive rate in the order of 10^{-5}. The classifiers are trained using our dataset of 2021 faces and 3000 randomly generated non-faces. The same positive set is used during boosting. For each stage a new negative set is created, following the common bootstrapping method of Sung and Poggio. Because the negative sets used for training the cascade are made out of false positives of the other stages than the current, it could only be used to calculate the false positive rate of the current stage. Therefore a separate (constant) negative set is used for evaluating the entire cascade and

to calculate the false positive rate.

All results presented in this chapter are on the positive training set and an independent negative set. Section 7.3.3 shows that there is no big difference in results when using a separate test set.

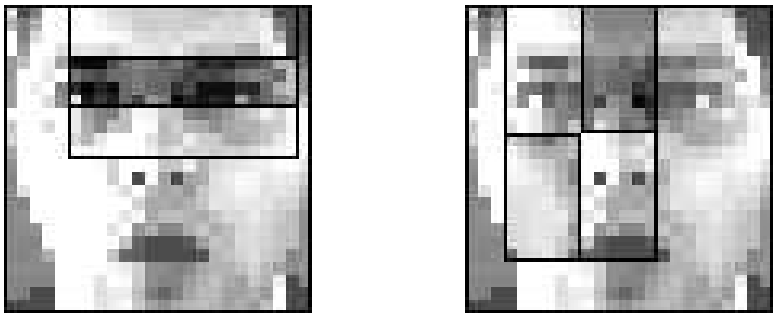

Figure 7.1: The first two features selected by AdaBoost.

7.2.1 Learningset creation

The negative samples are created randomly during training, as explained in section 4.2 but the positive samples are pre-created. We have labeled a collection of images containing faces by hand. The regions of interest are then cropped from the pictures and scaled to 24×24 pixels using a standard linear interpolation algorithm[3]: Let $I(x, y)$ be a pixel in the new image, then $I'(x', y')$ is a pixel in the old image, where $x' = x \times (oldRows/newRows)$ and $y' = y \times (oldColumns/newColumns)$. $I'(x', y')$ is the pixel value of the integer mapping of (x',y') in the old image. Further let $a = x' -$ integer of x' and $b = y' -$ integer of y'.

$$\begin{aligned} I(x,y) &= (1-a)(1-b)I'(x',y') + a(1-b)I'(x'+1,y') + \\ &\quad (1-a)bI'(x',y'+1) + abI'(x'+1,y'+1), \end{aligned} \tag{7.1}$$

After the sub windows are scaled, they are variance normalized (3.4) and converted to integral image (3.3). 2021 of these labeled faces are combined to form our dataset.

7.3 Results

7.3.1 Quadratic Classifiers

We have experimented with a quadratic classifier instead of the standard single threshold classification used by Viola and Jones. Because of the complete random nature of the negative trainingset (the domain of the non-faces

is extremely large), one can expect the variance of the feature values to be very high for non-faces. The best features, those selected by AdaBoost, can be expected to have a very small variance on face data. When training a quadratic classifier, not only a mean for positive and negative samples, but also the variance of both classes is calculated. This way 2 thresholds are constructed, based on the mean and the variance of both classes and the sharp defined faces can be 'isolated' from the random non-faces (see figure 7.2).

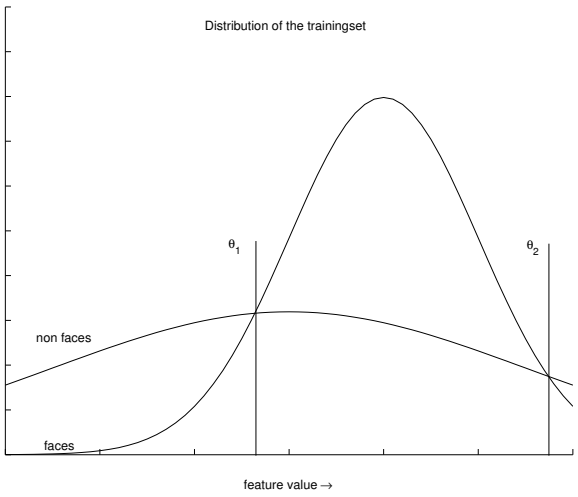

Figure 7.2: Demonstration the use of a quadratic classifier with two thresholds θ_1 and θ_2. The thresholds can 'isolate' the low-variant faces from the high-variant non-faces.

However the results are very disappointing. There seems to be no improvement over the classic nearest mean approach. Table 7.1 present the averaged results of a nearest mean detector with 2 classifiers (the first stage of our cascade) and averaged results of a quadratic detector also with 2 classifiers, both on the positive training set and an independent negative set.

	detection rate	false positive rate
nearest mean	98.98% ±0.04%	26,37% ±0.60%
quadratic	99.18% ±0.41%	31.59% ±0.62%

Table 7.1: Quadratic versus Nearest Mean classifiers. Both methods use 2 classifiers combined by AdaBoost.

7.3.2 ROC Curves

Here we present the Receiver Operating Characteristic (ROC curves) for our cascade. The ROC curve for the first stage (figure 7.3) is created by decreasing the threshold from $\alpha_1+\alpha_2$ (see table 4.2. There are two classifiers each with an α value. Let α_1 be the smallest and α_2 the largest) to 0. With only 2 classifiers it is clear that there are 3 breakpoints: When the threshold $\theta \geq (\alpha_1+\alpha_2)$ all samples are rejected. When $\theta \leq 0$, all samples are classified as a face. When $0 < \theta \leq \alpha_1$, a detection is made when either one of the classifiers returns positive. When $\alpha_1 < \theta \leq \alpha_2$, a detection is made only when classifier 2 (the one with the largest α) returns positive. Finally when $\alpha_2 < \theta \leq (\alpha_1 + \alpha_2)$, the detector returns positive only if both classifiers return positive.

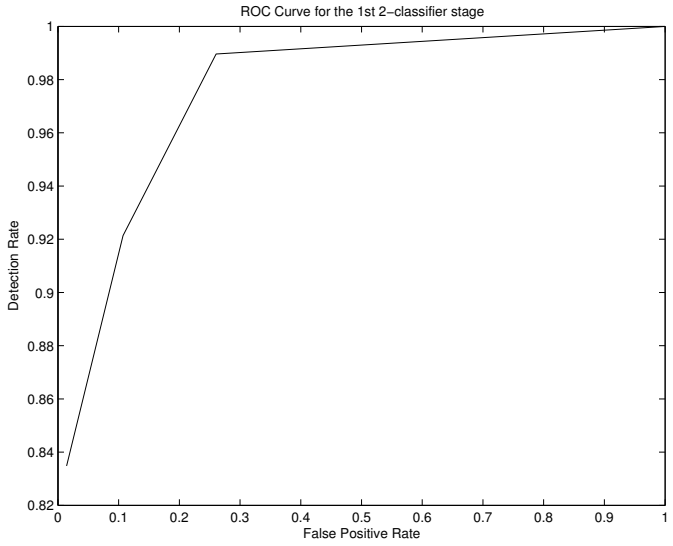

Figure 7.3: ROC Curve for a 1 stage detector with 2 classifiers on the dataset containing 2,021 faces and 6,000 non-faces.

The ROC curve of the 12 stage cascade (figure 7.4) is constructed by decreasing the threshold of the final stage from $\sum_{t=1}^{T} \alpha_t$ to 0. The detection rate upperbound of approximately 0.9 is the detection rate of stages 1 till 9 combined.

Discussion

The ROC curve in figure 7.4 shows that false positive rates in the order of 10^{-5} are achievable with detection rates of approximately 0.9. As discussed in section 5.1 these values are acceptable when building a robust detector.

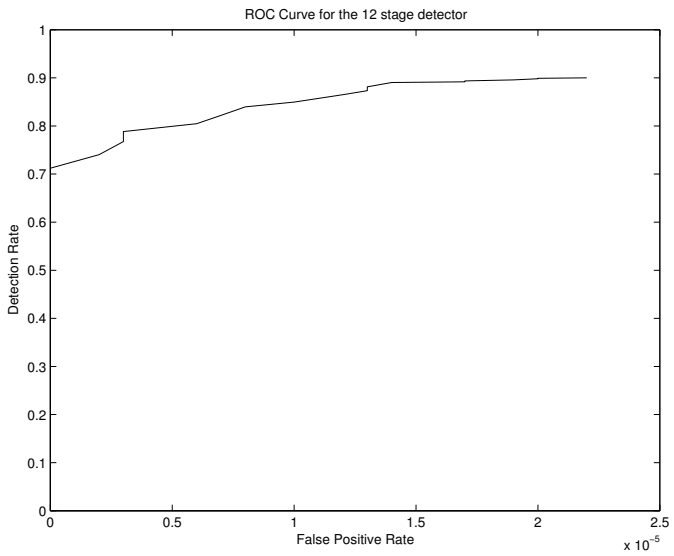

Figure 7.4: ROC Curve for a 12 stage detector with in total 1062 classifiers on the dataset with 2,021 faces and 1,000,000 non-faces. Note that the scale of the x-axis is different from figure 7.3

7.3.3 Trainingset size

We have studied the effects of varying the size of the dataset on the detection rate. To do so, we have split the dataset in a training- and a test-set. We have trained the classifiers with the trainingset and evaluated them with both the trainingset and the testset. The results as shown in figure 7.5 are based on a 3 classifier detector with a false positive rate of approximately

10%. Since detectors with few classifiers have a very rough ROC curve (as in figure 7.3) it is very hard to get all the sample points to have roughly the same false positive rate. Therefore we will need a measure that incorporates the false positives in the detection rate.

Blackwell[14] describes a method to estimate the chance of success of a system. He describes the chances at the event of a signal S (which maps to a face in our domain) or no signal N (which maps to a non-face). $P(S|s)$ is a Hit, $P(S|n)$ is a False alarm, $P(N|s)$ is a Miss and $P(N|n)$ is a Correct reject. Hits and false alarms map to our detection and false positive rates. So we have ROC curves of Hits and False alarms. This leads to a measurement for the chance succes:

$$P^*(S|s) = \{P(S|s) - P(S|n)\}/\{1 - P(S|n)\} \tag{7.2}$$

Even with this measure, it still are shaky lines, but it is clear that the lines are convergent and reach an optimal value at around 1100.

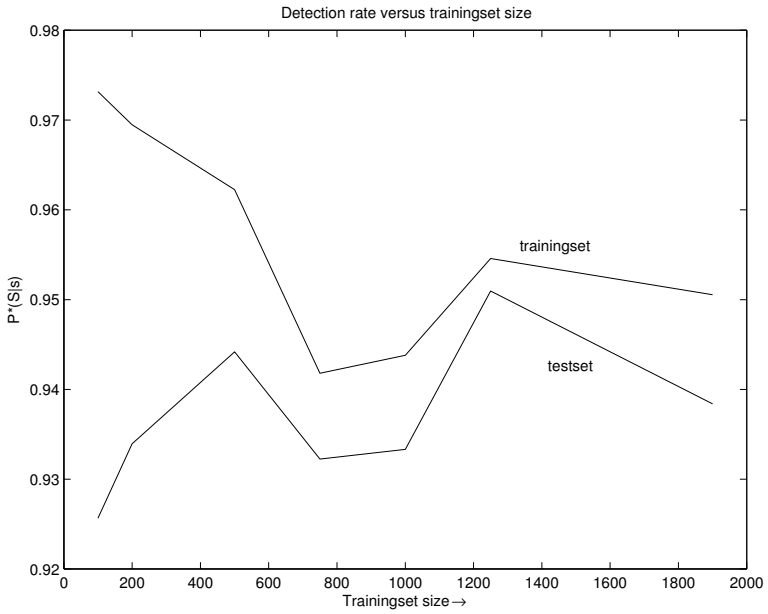

Figure 7.5: Results for different dataset sizes.

7.3.4 Speed

Viola and Jones report a execution time of 0.067 seconds per image, which is around 15 frames per second, on a 700 Mhz Pentium III. They use images of 384×288 pixels. Their detector uses a starting scale of 1.25, a step size of 1.5 and a scale factor of 1.25 (see section 6.2). With the same settings and variance normalization enabled our detector runs at 4 frames per second on a 2.4 Ghz Pentium 4 and at 8 frames per second when normalization is disabled. This considerable difference in speed can only be explained by a non-optimized implementation of our detection algorithm, since our cascade contains less classifiers than that of Viola and Jones and the first layers of our cascade perform equally well if not better than Viola and Jones'.

With a starting scale of 1.3, a step size of 1.6 and a scale factor of 1.25, our detector runs at 8-10 frames per second with normalization and at 15-20 frames per second without normalization. A test application bundled with the OpenCV implementation that exists on the internet runs at approximately 15 frames per second on our computer. This application uses a scale factor of 1.1 and a variable step size. It also uses an edge detection routine that speeds up detection by eliminating regions of the picture that are likely not to contain faces.

7.3.5 Non-frontal detection

As said in chapter 6, we have trained a second cascade using a profile face dataset. Training was done with 500 faces and again 3000 non-faces. The same bootstrapping algorithm during boosting was used as with the frontal detector. The final cascade has 12 stages with a detection rate of 0.894 and a false positive rate of 4.0×10^{-6}. In total there are 1099 classifiers. Although these values on the dataset are very similar to the values of the frontal detector on the dataset, performance on real life pictures is not nearly as good as with frontal faces. When tested on the MIT/CMU profile test set, many faces were missed and the false positive rate was higher than would be acceptable for a robust system. Profile faces lack the clearly defined features that are characteristic to frontal faces, so detection of these faces relies on features that are very sensitive to rotation or occlusion. E.g. hair covering half the face and the ear. Extended feature sets could be the answer, but detection will always be harder than the detection of frontal faces.

7.4 Conclusion

With the error rates presented in 7.3.2 and the speed presented in section 7.3.4 we show that the scheme of Viola and Jones is capable of achieving robust real time face detection using simple classifiers. Using better classifiers does not improve performance of the boosted system. Also our experiments with non-frontal detection show that the method is highly extendible. Our experiment in section 7.3.3 shows that a dataset of at least 1000 samples is required to reach optimal learning. This means that our non-frontal detector could benefit from a larger dataset than the one we have used containing 500 faces.

Chapter 8

Conclusion

Viola and Jones provide an all-round detection framework that is worth examining closely, where future research could focus on video sequences to combine tracking algorithms with real time detection and rotation invariant detection. Also profile face detection still is not nearly as good as frontal face detection, so the effect of extending the feature set on profile face detection could be examined. Certainly creating a larger training set would improve the performance of our profile detector.

There still is a lot of work to be done, but real-time face detection is at a point where it certainly performs well enough to be used in practical applications.

Bibliography

[1] P. Viola and M. Jones. Robust Real-time Object Detection. *International Journal of Computer Vision 57*, pages 137-154, 2004

[2] R. Lienhart and J. Maydt. An Extended Set of Haar-like Features for Rapid Object Detection. *IEEE ICIP*, vol. 1, pp. 900-903.

[3] B. Carlson and C. Looney. A New Method of Image Interpolation. Technical Report Department of Computer Science, College of Engineering, University of Nevada, Reno.

[4] C.P. Papageorgiou, M. Oren and T. Poggio. A General Framework for Object Detection. In *Proceedings of International Conference on Computer Vision*, pages 555-562 1998.

[5] R. Bunschoten. Face Detection using Neural Networks. *M.Sc. Thesis.* University of Amsterdam, 1998

[6] H.A. Rowley, S. Baluja and T. Kanade. Human Face Detection in Visual Scenes. Technical Report CMU-CS-95-158, School of Computer Science, Carnegie Mellon University, 1995.

[7] T.V. Pham and M. Worring. Face Detection Methods, a Critical Evaluation. Technical Report 11, Intelligent Sensory Information Systems Group, University of Amsterdam, 2000.

[8] E. Osuna, R. Freund and F. Girosi. Training Support Vector Machines: an Application to Face Detection. In *Proceedings of CVPR'97*, pages 130-136 1997.

[9] K.K. Sung and T. Poggio. Example Based Learning for View-Based Human Face Detection. A.I. Memo 1521, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1994

[10] P. Viola and M. Jones, Fast Multi-view Face Detection. Technical Report TR2003-96, Mitsubishi Electric Research Laboratories. 2003.

[11] H.A. Rowley, S. Baluja and T. Kanade. Rotation Invariant Neural Network-Based Face Detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 38-44 1998

[12] A.R. Webb. Statistical Pattern Recognition. p. 294-296. 2002.

[13] B.J.A. Kröse. A Description of Visual Structure, *Phd. Thesis*, Delft 1986.

[14] H.R. Blackwell. Psychological Thresholds: Experimental Studies of Measurements. Bull. Eng. Res. Inst. U. Mich.36. 1969.

[15] OpenCV Computer Vision Library. http://www.intel.com/research/mrl/research/opencv/, 2004.