Realtime face detection

Samuel Englund

March 17, 2003

Summary

This Master thesis is done in collaboration with Cecil AB. The goal is to evaluate visual face detection methods. The techniques evaluated are all based on the "window transform", which is a sort of band pass transform. Genetic programming is introduced to improve the window transform. The algorithms AdaBoost and Support Vector Machine (SVM) are both tested as classifiers for the face detector. A successful improvement of AdaBoost that enables a reduced number of training examples has been done. Thereafter follows a discussion about the possibilities for the CNN chip to use the tested algorithms. The CNN chip is under development by Cecil AB.

Sammanfattning

Detta examensarbete görs i samarbete med Cecil AB. Målet är att undersöka metoder för visuell ansiksdetektering. Teknikerna som prövas baseras alla på "the window transform", som är en slags bandpass transform. Genetisk programmering introduceras för att förbättra fönstertransformen. Klassificeringsalgoritmerna AdaBoost och Support Vector Machine (SVM) testas som klassificerare till detektorn. En förbättring av AdaBoost som gör att man kan använda ett reducerat antal träningsbilder har utförts. Därefter följer en diskussion om hur CNN chipet kan tillgodogöra sig de prövade algoritmerna. CNN chipet är under utveckling hos Cecil AB.

Aknowledgements

I would like to thank Professor Mats Rudemo, Torbjörn Ahlqvist, Christian Edberg and Johan Pettersson for all their suggestions and support. I also wish to explain my gratitude to the staff of Cecil for being such great colleges and for making me feel at home there.

Contents

1 Introduction **5**
1.1 Multi-class image classification 5
1.1.1 Training 6
1.2 Detection 7
1.3 A guide through this thesis 7

2 A market survey **8**
2.1 Freeware 8
2.1.1 Carnegie Mellon 8
2.1.2 Fraunhofer Institut 8
2.1.3 National Research Council of Canada 9
2.1.4 University of Alberta 9
2.1.5 Sluggish Software 10
2.2 Commercial systems 10
2.2.1 Visionsphere 10
2.2.2 Imagis Technologies inc. 11
2.2.3 Identix 11

3 Window-AdaBoost **12**
3.1 Preface 12
3.2 Introduction 12
3.3 Why the window transform? 13
3.4 Classifiers 14
3.5 The training sets 15
3.6 The AdaBoost training 16
3.6.1 A flow chart over window-AdaBoost 17
3.7 Test 18
3.7.1 Test 1 19
3.7.2 Test 2 19
3.7.3 Test 3 19
3.7.4 Test 4 20
3.7.5 Test 5 21
3.7.6 Test 6 22
3.8 Further discussion about the window-AdaBoost technique 25

4 Support Vector Machine, SVM **26**
4.1 Preface . . . 26
4.2 Introduction . . . 26
4.3 Linear SVM and separable data . . . 27
4.4 Linear SVM and *non*-separable data . . . 29
4.5 *Non*-linear SVM . . . 30
4.6 Setting up a SVM test . . . 31
4.6.1 Single detection or detection in stages? . . . 31
4.6.2 Choosing dimensions . . . 32
4.6.3 Known problems with SVM . . . 32
4.6.4 Setting parameters . . . 32
4.6.5 Software . . . 32
4.7 Test . . . 33
4.7.1 SVM + using previous results . . . 33
4.7.2 Highest mean . . . 34
4.7.3 Ransac . . . 35
4.8 Conclusions on SVM . . . 36

5 Genetic programming **38**
5.1 Preface . . . 38
5.2 Introduction to GP . . . 38
5.3 This specific GP . . . 39
5.3.1 Test AdaBoost 1 . . . 40
5.3.2 Test nearest neighbor . . . 40
5.3.3 Test AdaBoost 2 . . . 41

6 CNN-Cellular Neural Network **43**
6.1 Preface . . . 43
6.2 Motivation . . . 43
6.3 Basics about CNN . . . 43
6.4 CNN friendly operations . . . 44
6.4.1 Filtering . . . 44
6.4.2 Morphology . . . 45
6.4.3 Diffusion . . . 45

7 Future work **46**
7.1 Better feature extraction . . . 46
7.2 Better decision surface . . . 46

8 Appendix **48**
8.1 Images for training . . . 48

Chapter 1

Introduction

The goal for this master thesis is to evaluate visual detection methods, which is of interest for Cecil. I chosed the area of face detection both because there is a potential market for Cecil and because there has been a lot of work prior in this area. The first real time detector was claimed to be done by Viola & Jones [1] and their work is the base for all methods that have been tried here.
CNN is a short for *Cellular Neural Network* and is a subset of neural networks. Cecil is currently developing a CNN chip designed for fast close-to-sensor image processing. Chapter 6 is about how the investigated algorithms apply to the CNN chip.

1.1 Multi-class image classification

Classification is to take an image of a given size and to label it to one of the predetermined classes. Any multi-class classifier consist of two parts: Feature extraction and decision surface.

Feature extraction

To classify a sub image based on its pixel values is usually not the fastest way around. The dimension of a grey scale image is the same as its pixel number, and for color images it is the pixel number multiplied with three. That takes too much time to analyze for a real time system. Feature extraction is to look at some specific things for the classes in the picture. Let us imagine that we want a classifier for farm animals and there are three classes of animals: pigs, hens and rabbits. The animals in each class share some features. Pigs are pink and clean shaved, hens are white and feathery and rabbits are grey and furry. If we got a color picture of one of the animals on the farm, what would be a feature to look for in order to determine its specie and class with a minimum of effort? The most obvious is to see what the mean color is, that is a transform to the 3D RGB color space. One could also look at the texture of the animal and determine if it was a clean-shaved, feathery or furry, but that would mean a transform into much higher dimensioned transform space.

$$\begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & & \\ \cdot & & \cdot & \\ \cdot & & & \cdot \end{pmatrix} \rightarrow \begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{pmatrix}$$

Image space Transform space

Decision surface

Once the image is in the transform space we need one or more decision surfaces. A decision surface is a surface that separates one or more classes. If the transformation of the unknown animal appears in within the decision surface/surfaces that encapsulate the pink(pig) area we can label the image as a pig and so on. Figure 1.1 shows a decision surface for two classes in a two dimensional space.

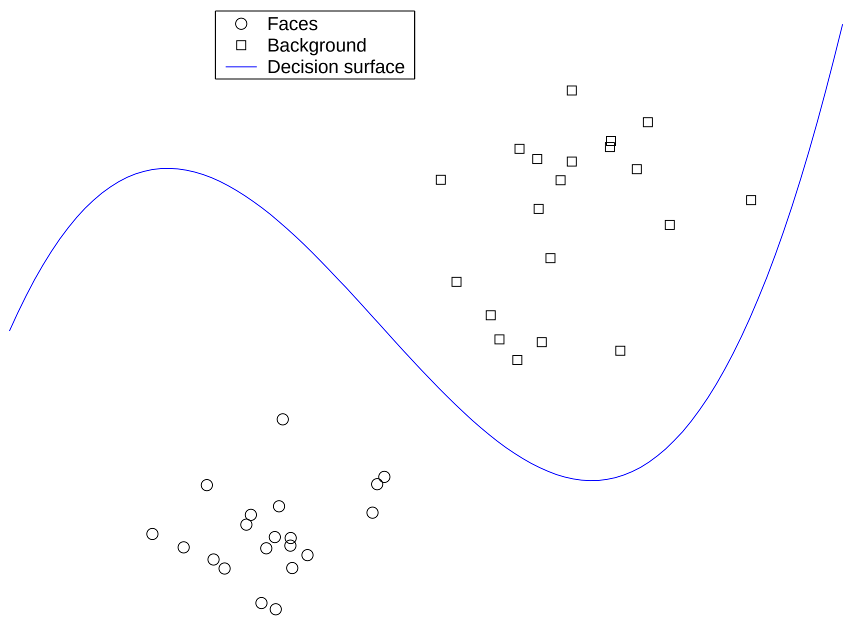

Figure 1.1: An example of what the transform space can look like in two dimensions

The operator has to choose possible transform methods and algorithms to construct decision surfaces. When these theories are planned, some hard work still remains. The detector has to be trained.

1.1.1 Training

Just like a secret agent, the detector can not rely only on theoretical studies, but it has to do some field work training as well to get prepared. The training is necessary to set parameters for the algorithms and to find the decision surface(s).

1.2 Detection

Detection is a special case of general multi-class image classification. Only the positive class and the background class exist in the detection case. In this master thesis some things will remain unchanged. The positive class will be faces. The size of images to classify are 24 by 24 pixels wide. One type of transform will be used.

1.3 A guide through this thesis

We start by examining the technique I call window-AdaBoost developed by Viola & Jones [1] in chapter 3. The image representation is the window transform and the decision surface is constructed using AdaBoost. The types of windows for the transform are done manually by educated guessing.
Problems with huge training sets lead to a (probably) new way to handle the negative training set. After that follows a trial to improve their work by using SVM as constructor of the decision surface is found in chapter 4, but without much luck. A rather successful trial to let the computer automatically generate transform dimensions is done in chapter 5. Genetic programming is used for selecting and trying the transforms.
The next part of the work is about how well the window transform applies to CNN technique and that is found in chapter 6.
In the last part of the thesis, a couple of suggestions are made for future work concerning this subject.

Chapter 2

A market survey

Here is a short survey about what is available on the market. There are both freeware and commercial products available today. This survey does not claim to be complete, but the hope is that it can give an idea on what has been done and what is currently possible.

2.1 Freeware

2.1.1 Carnegie Mellon

The method is described in
http://amp.ece.cmu.edu/Publication/Jie/icme2000_jie.pdf.
The basic idea is to have a priori knowledge of what colors are the most common in a face. Each pixel is statistically evaluated to get the likelihood of what class it belongs to, face or background. The pixels are clustered together and a search algorithm is applied to get a cluster of pixels that are the most likely to be a face. When the most likely cluster is found, the locations of the pixels in that cluster point out the face. I have not found any data on detection rate, but images on the web site
http://amp.ece.cmu.edu/projects/FaceTracking/Default.htm
show what the result could look like.
Carnegie Mellon offer C++ code to be downloaded for free at
http://amp.ece.cmu.edu/projects/FaceTracking/download.htm
but they do not have an immediately executable program.

2.1.2 Fraunhofer Institut

Unfortunately, it is not described what method that are used, but an excellent 60-days free demo is offered to download at
http://www.iis.fraunhofer.de/bv/biometrie/download/index.html.
The demo is easy to install and tracks faces in the web camera that has to be connected to the computer. The web camera I had on my computer is of very low quality, but the program was robust enough to track my face anyway. Impressive!

2.1.3 National Research Council of Canada

Dr. Dmitry O. Gorodnichy has constructed a program to track the nose, called Nouse. Nouse locates the nose and enables you to steer programs with our nose as a mouse or joystick. Nouse seems to use intensity maxima around the nose, but the whole truth is a business secret. Download Nouse at
http://www.cv.iit.nrc.ca/research/Nouse/.
The program is very easy to install and the only additional ware you need is a web camera. Once the nose is aligned to the tracker, Nouse tracks the tip of the nose very smoothly. Sometimes the program could not keep up the tracking if I moved the head too fast. I believe that Nouse is working with some kind of minimization because if it loses track of the nose, it follows an other part of the face like chin, cheek or forehead. It seems to look for any bright local maximum in the face to track. Once it finds a bright spot on the face it follows it as if it was the tip of the nose. Nouse keeps up good in real time and as I just mentioned before, I did not have a very good web camera. Very impressive! Picture proof of me in nouse is found in Figure 2.1.

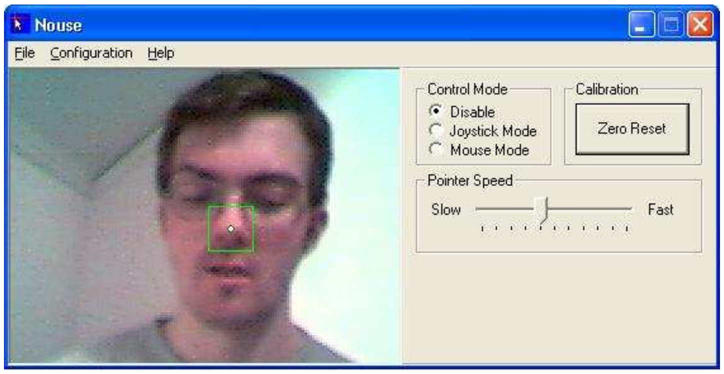

Figure 2.1: Screen dump of Nouse using my webcam.

2.1.4 University of Alberta

They provide Matlab Code for Principal Component Analysis (PCA), a method using eigenvectors as a basis for a linear vector space. A SVM (Support Vector Machine) sets up a decision surface that divides the vector space into two parts, faces and background. PCA+SVM gives high detection rate, but it is too slow to use in any real time system operated by a PC because the method is computationally demanding. Initiated colleges at Cecil say that the CNN chip can operate PCA+SVM in real time. At
http://vasc.ri.cmu.edu/cgi-bin/demos/findface.cgi
you can write the URL to an image you want to analyze. This is an example of

what it can do (Figure 2.2). Many more examples can be found on that site.

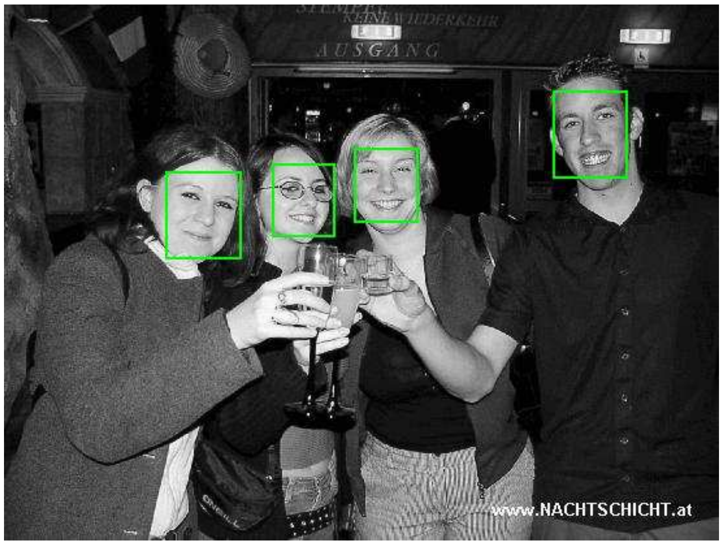

Figure 2.2: All faces in the image are successfully detected

2.1.5 Sluggish Software

On the following website you can read about their strategy.
http://www.fuzzgun.btinternet.co.uk/FaceLocation.htm
The idea is to take out pixels that may be a part of a face based on what color the pixel has. The areas with a lot of potential pixels will be examined further using PCA as described in the previous method. There is software to download on
http://www.fuzzgun.btinternet.co.uk/Downloads.htm.
Unfortunately, I did not succeed to run any of the software on that site.

2.2 Commercial systems

Commercial system include a recognition part as well as the detection. Few people would ever invest in a system that just detects and locates faces. To make a system for recognition you generally need to detect the face and from there pick out the face and compare it to a database. It is generally difficult to get any information on how good the detection part is, so I just present some system available today.

2.2.1 Visionsphere

Visionsphere has developed systems for verifying users of computers and for entrance. A detection part that isolates the face is incorporated, but further

information about how it works is a trade secret. Both software and hardware are available. Visit Visionsphere at
http://www.visionspheretech.com/.

2.2.2 Imagis Technologies inc.

Imagis has one system for entrance and one system specially made for the police. Imagis has announced that the police in Orange County, CA in the US use their system currently for booking people they arrest. No information what so ever on how it works. Visit Imagis at
http://www.imagistechnologies.com/.

2.2.3 Identix

In 1997 the cases of robbery had began to rise dramatically in London, UK. The police in Newham, a London borough, asked themselves what could be done to stop the alarming trend in crime. An American company, Visionics (later merged with Identix), proposed a system where surveillance cameras were put up in the streets. A detection system located faces and another system compared the faces with a data base of known criminals. The goal was to get a system that could alarm before the suspect had left the camera. The system was successfully put into operation in 1997 and an immediate drop in street crime of 34% was attained. While crime continued to rise in other parts of London, Newham stayed on a lower crime level. To put up surveillance systems in the streets and in other public places is controversial, since many people do not like the feeling of being controlled by any government, but I leave that discussion out of this thesis. Everything above is according to
http://www.sourceuk.net.
Visit Identix at
http://identix.com

Chapter 3

Window-AdaBoost

3.1 Preface

I call the method in [1] for window-AdaBoost. This chapter fully explains window-AdaBoost and several tests using this technique are presented.

3.2 Introduction

The technique using the window transform and AdaBoost was developed by Paul Viola and Michael J.Jones [1]. This method was presented at the ECCV convent in 2001 and was claimed to be the first method for real time face detection. The most critical factor in detection is speed. Viola & Jones present numbers that show that their system can scan a 384 by 288 pixels large image in 11 scales in $\frac{1}{15}$second. They used an ordinary desk top computer with a Pentium 700MHz processor for that test.
Using window-AdaBoost as the base, Stan Z. Li et al. further developed the detection system to incorporate faces rotated in all directions [3], which Viola & Jones did not. There are more examples of visual detectors using AdaBoost of which one is found in [2].
For feature extraction a bandpass transform is used that is referred to as the window transform. The window transform is similar to Haar wavelets, but with the fundamental differences that it is left-right symmetrical, over-complete and has no inverse transform. But frontal faces are left-right symmetrical (in the ideal case) so such a picture can be transformed without loss of information. Each dimension in the transform space is defined by a window, which is a set of rectangles. In Figure 3.1 you can see which types of rectangle combination that I have used. The transform value is the sum of the pixels in the black area minus the sum of the pixels in the white area(s). The windows are chosen to emphasize the natural differences in intensity within a face. Eyes and mouth are typically low in intensity while forehead, nose and chin are higher. The rectangles are specified by width, height and origin in the image.
The two areas contain the same number of pixels, which removes the low frequency part of the image, i.e how bright the image is. The smaller the window used are the higher are the frequencies caught. The deterministic part of this method is which windows to pick.

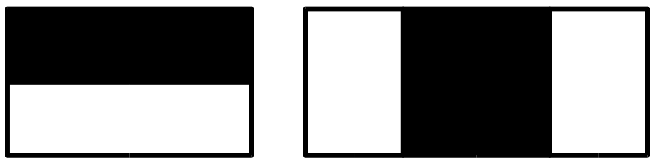

Figure 3.1: The two types of rectangle combination for the window transform. The width and height are variable, but with the restrictions that the rectangles must be next to each other and that the areas of the black and white rectangles are the same.

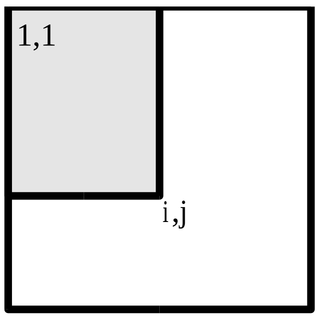

Figure 3.2: $P_{i,j}$ is the pixel sum of the shaded area

3.3 Why the window transform?

These kinds of rectangular windows are used not only because of their ability to produce distinct transform values for faces, but they are also efficient to compute. The secret is to take the integral representation of the image. The integral image has the same number of pixels as the original image, but the pixel values ($P_{i,j}$) are the sum of all pixels ($p_{m,n}$) from that point (i,j) and up and to the left (Figure 3.2).

$$P_{i,j} = \sum_{m=1}^{i} \sum_{n=1}^{j} p_{m,n}$$

When we apply a window to an image we calculate the sum of all the pixels within some rectangles. The number of operations used when summing the pixels in an area is the same as the number of pixels within the same. The computational finesse with integral representation and rectangular windows is that the pixel sum of any rectangles is calculated by one addition and two subtractions. As there are a whole lot of rectangles to be computed, the integral image is the key to the quick transforming.

$$\text{Rectangle sum} = \sum_{i=i_1}^{i_2} \sum_{j=j_1}^{j_2} p_{i,j} = P_{i_1-1,j_1-1} + P_{i_2,j_2} - P_{i_2,j_1-1} - P_{i_1-1,j_2} \quad (3.1)$$

3.4 Classifiers

First we have to list some denotations.

Class In detection problems there exist two classes, positive and negative. The detector labels all subimages (24 by 24 pixels) it searches to be either positive or negative.

Positive In this paper a positive image is an image of a face. If the detector labels a subimage positive it means that the subimage should be a face.

Negative A negative subimage is a subimage that is anything but a face.

False positive, FP An image is called *false positive* if the image is not a face, but the detector labels it positives. A *true positive* image is an image of a face that the detector correctly labels positive.

False negative, FN An image is called *false negative* if the image is a face, but the detector labels it negative, which means it does not find that face. A *true negative* is a negative image correctly labelled negative.

Classifier The detector in this chapter consists of several classifiers. A classifier suggests to which class an image belongs. There are two kinds of classifiers. The first one is the *weak classifier*, written WC in text and h in equations. The second one is the *strong classifier* SC in text and H in equations. The classifiers are further described below.

When the final detector scans a big image on site it has to go through all 24 by 24 subimages and has to rescale the big image to find faces in all kinds of scales. Any big image always contain many more negative than positive subimages, so the wish is to quickly classify the negative subimages correctly to get rid of them without spending too much time on each negative. The positive subimages are much fewer, so we don't bother if it takes more computing until the detector establishes that a subimage is positive. To solve this problem the detector is built in a cascade of so called strong classifiers, SC. Each subimage goes through all SC but if one SC on the way classifies it as negative, the following SC doesn't have to process it. That is because it is only images that pass as positives through all SC that are labelled positive by the detector. Supposed that the detector has no false positive or false negatives, then all negative sub-images get filtered out somewhere on the way and only the few positive has to go through all SC.
The SC make their decisions based on suggestions from a couple of weak classifiers, WC. The object of training is to pick out the right WC and to assemble them into a SC.
Let x be a 24 by 24 image, represented as the pixel values in one 576 dimensional vector. A WC consists of a window transform $t_j(x)$, a threshold θ_j and a parity $p_j \in \{-1, 1\}$. The suggestion the WC makes is either 1 for x being positive or 0 for x being negative. Remember that WC is called h in equations.

$$h_j(x) = \begin{cases} 1 & \text{if } t_j(x)p_j < p_j\theta_j \\ 0 & \text{else} \end{cases} \tag{3.2}$$

A weight α is assigned to each WC which corresponds to how well it did in the

Figure 3.3: Examples of background images from which the negative training examples are taken

training compared to the other weak classifiers. Thus, a big α-value means that it is a good classifier so that the SC can rely more on it.
Finally, the SC (H in equations) with the threshold Z and T weak classifiers is determined by:

$$H(x) = \begin{cases} 1 & \text{if } \sum_{j=1}^{T} h_j(x)\alpha_j \geq Z \\ 0 & \text{else} \end{cases} \tag{3.3}$$

If $H(x) = 1$, the SC labels the image positive. The parameters of the WC as well as Z are determined during the training.

3.5 The training sets

All statistical training methods require training examples. All a priori knowledge we have is to which class the training images belong. The negative training examples are easily found on the web and for starters 100 images were collected. Two examples are seen in Figure 3.3. From these images a program randomly picked out 24 by 24 sub images in different scales.
Examples of negative images are found in Figure 3.4. There are ready-to-use

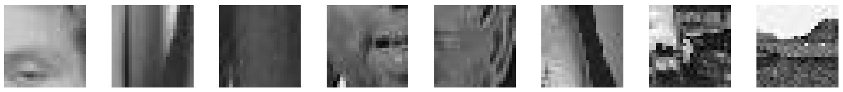

Figure 3.4: Examples of negative images resized to 24 by 24 pixels to use in the training

databases for faces, but only the Olivetti Face Database was appropriate for this case. All other databases were either non free, stored images in impossible formats or were just not suitable for this task. In the Olivetti Face Database there are ten pictures per person and 40 persons in total. The pictures are taken for different head poses but are considered to be frontal enough. Examples are seen in Figure. 3.5.
Thank you AT&T Laboratories Cambridge for sharing these images without any profit.

Figure 3.5: Examples of persons from the Olivetti Face Database

There are more examples of training images in chapter8.

3.6 The AdaBoost training

In this section the training of the classifiers will be considered, first the positive and negative examples. Denote the number of negatives training examples by N_n and the number of positive training examples by N_p. Each image x_i is associated with a weight, w_i, which initially is set to

$$w_i = \begin{cases} \frac{1}{2N_n} & \text{if image } x_i \text{ is positive} \\ \frac{1}{2N_p} & \text{if image } x_i \text{ is negative} \end{cases}, \text{ where } \sum_{1}^{N_n+N_p} w_i = 1$$

An image is also associated with its class, y_i, which is zero or one, depending on if it is negative or positive.
Now comes the tough part in which the best weak classifier is picked out. The word "best" is in the AdaBoost meaning and is based on the error e_j. e_j is the sum of the weights of the not correctly classified training examples which can be written as

$$e_j = \min_{\theta_j, p_j} \sum_{\forall i} w_i |h_j(x_i, \theta_j, p_j) - y_i| \tag{3.4}$$

The WC with the the smallest $e_j = e$ is chosen.
Let us now make a note that one WC might be good at finding, say perfectly frontal faces and another good at finding faces that are slightly rotated in some direction. A good SC need contribution from WC that can point out all kinds of faces and therefor the weights are updated so that the training examples that are wrongly classified by previous WC's are counted as more important. Put

$$w_i \leftarrow \begin{cases} w_i & \text{if image } x_i \text{ was } \underline{\text{not}} \text{ classified correctly} \\ w_i \frac{e}{1-e} & \text{if image } x_i \text{ was classified correctly} \end{cases} \tag{3.5}$$

and normalize the weights so that

$$\sum_{\forall i} w_i = 1$$

Start over and continue until you have T WC's.
Z, the threshold in the SC can now be set and the user has to decide an acceptable number of false negatives, FN. The threshold Z is found by running

the SC over the positive training examples and set the value for Z for which the number of false negatives equals the chosen level. In Figure 3.6 it is illustrated how that Z-value is set so that four FN are allowed for that training set.

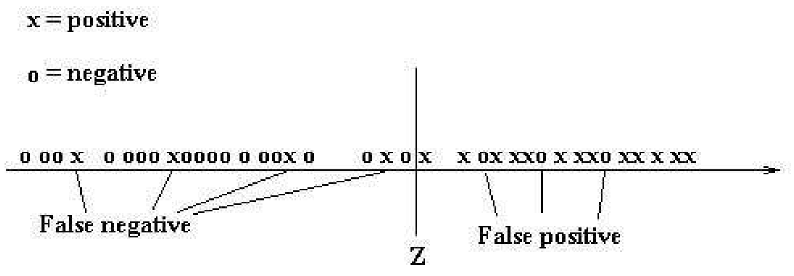

Figure 3.6:

One of the good features of the boosting is that the negative training examples labelled negative by the SC are dropped from the training set. The previous SC are able to find them so the next SC can concentrate on the false positives instead. This will speed up the training because training time is linearly dependent on the number of training images.
The algorithm starts over again to find the next H with the reduced set of negative examples.

3.6.1 A flow chart over window-AdaBoost

1. Prepare Chose the rectangle combinations $t_1 \dots t_n$, get the images to train on $x_1 \dots x_{N_p+N_n}$ and calculate the window transform values $t_i(x_j)$ on all the images. The number of SC and WC have to be decided as well as how many FN that are allowed for each SC.

2. Train

2a, main boosting loop.

- while the number of SC $< N_s$
- go to **2b**
- remove all true negative that this SC finds in the training set
- increase the number of SC
- end while

2b, loop to train one SC.

- Initiate weights for training images.

$$w_i = \begin{cases} \frac{1}{2N_n} & \text{if image } x_i \text{ is positive} \\ \frac{1}{2N_p} & \text{if image } x_i \text{ is negative} \end{cases}$$

- for j=1:T
  Calculate error values for all n_t transform dimensions.

$$e_k = \min_{\theta_k, p_k} \sum_{i=1}^{N_n+N_p} w_i |h_k(x_i, \theta_k, p_k) - y_i| \qquad k = 1 \cdots n_t \tag{3.6}$$

  The minimization problem give both e_k, θ_k and p_k for all dimensions.
  h_j is chosen by which of the n_t dimension that has the lowest error $e_k, k = 1 \cdots n_t$.
  The α-value for the WC is computed by:

$$\alpha_j = \log\left(\frac{1-e_j}{e_j}\right) \tag{3.7}$$

  Update the weights for the training images

$$w_i \leftarrow \begin{cases} w_i \left(\frac{e}{1-e}\right) & \text{if image } x_i \text{ was classified correctly} \\ w_i & \text{if image } x_i \text{ was not classified correctly} \end{cases} \tag{3.8}$$

  and normalize the weights so that $\sum_{i=1}^{N_p+N_n} w_i = 1$ apply.

- end for
- Set threshold Z for the SC based on how many FN that are allowed
- return H

3.7 Test

A number of training programs were made with a little improvement every time. When the training images are collected and the code is written it is time to set the parameters for the training.

$\{t_i\}$ the set of transform windows.

N_n the number of negative training examples.

N_p the number of positive training examples.

T the number of WC in one SC.

N_s the number of SC.

Fn the number of FN allowed for each SC.

Which parameters that have to be set and which don't depends on the design of the program. Here is a little about the impact the parameters above make.
The types of transform windows available is important and an expansion to more than the two types that is used here is probably a good idea.
N_n and N_p have to be large enough so that the samples resembles reality. But N_n and N_p can't be too big because that will take to much time to train and will at a certain number eventually crash the computer.
Large T and N_s will make the final classifier more accurate, but slower. Fn controls the rate $\frac{\text{FP}}{\text{FN}}$.

3.7.1 Test 1

$N_p = 400$, $N_n = 3000$, $T = 10$, $N_s = 20$, $Fn = 2$ and there are 2075 transform dimensions. All positive training examples are assumed to be equally hard to find (= iid). The aim was set for a 90% detection rate for the positive training examples: $(0.90 = (1 - Fn/N_p)^{N_s}$ if $Fn = 2)$. The training took about 3 hours and the result for this first classifier was 2.2% FN and 4.5% FP, when the detector was tested on the training set. 2.2% FN is better than expected (and probably ruining the assumption of iid), but 4.5% FP makes the detector totally useless. A manual investigation of which faces the detector could not find was performed. There was only one black person in the training set which contained 40 persons and the images of him were more difficult to detect. I believe that is because the window transform compare intensities between different parts of the face. The window transform values probably get quite different for white and black persons. This flaw can be remedied by adding more black people to the training set. There were several men with beard, but they were not difficult to detect even though one suspect the transform values to differ from the standard person of the training set which is a white male who is beardless and who is not wearing glasses. My explanation to this is that the training examples contain many enough beard-wearing men, but not enough examples of black persons.
The next test on the schedule was to see how changing T and N_s affect the result.

3.7.2 Test 2

T was changed to 20 and N_s to 10 so that there were 200 weak classifiers in total for this test as well. Fn was set to 4 this time to aim at the same true positive detection rate. All negative training examples were found and removed after eight out of ten SC. The eight SC gave 4.6% false negatives and about 2% FP when tested on images from without the training set. Keeping the same total number of weak classifier, but grouping them differently did have an effect on the training.
Since two strong classifiers were missing the number of negative training examples had to be increased.

3.7.3 Test 3

N_n was increased to 20,000 and this time it was many enough. The result though, is not okay since there still were to many FP. It took 13 hours to complete the calculation.

To see how the number of false positives decreased for every strong classifier the program got to continue to run and find more SC. The detector was meant to use ten strong classifiers, so the additional ten were trained only made for an investigative purpose and the results are seen in Figure 3.7. The number of false positives seem to decrease logarithmical to the number of SC.
The question is why some of the negative training examples are so difficult to classify?

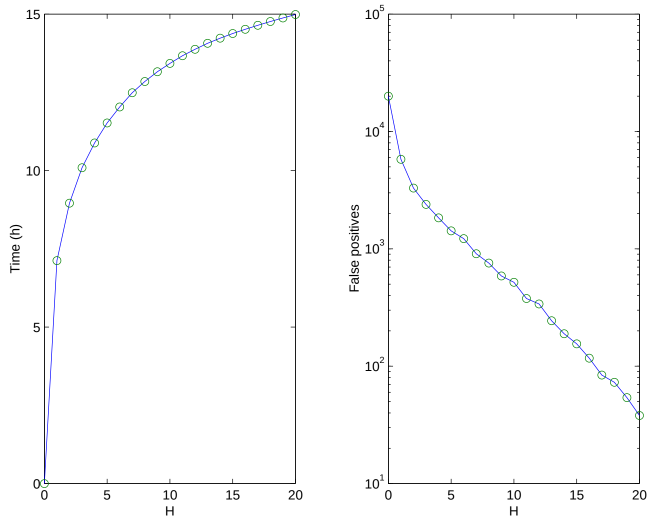

Figure 3.7: A plot of the time passed against the number of negative training examples left in training (the FP) as a function of how many SC that yet have been trained

3.7.4 Test 4

If the positive training examples are more alike, they will span a smaller volume in the transform space. All faces were cropped so that hair and background vanished. Up until now, the faces had not been really aligned, which cropping them manually like in Figure 3.8 fixed.
The program was started with the parameters set to the same as before.Due to the fact that the positive training examples were more alike after the cropping, the negative training examples were too few again. Around 20,000 negative training examples was the maximum number the computer could handle and even if that could be fixed, it kind of felt like this was not the best way to go. The first strong classifiers get so many more negative examples than the last, since the number of negative training examples decrease logarithmical in number for each SC. It would be more fair if all SC had the same number of examples to train on.

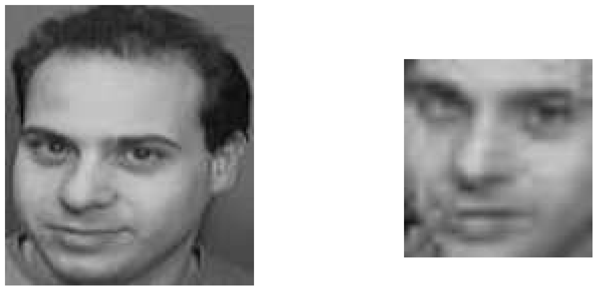

Figure 3.8: The image to the right is rescaled to 24 by 24 pixels, but due to a smoothing that occurred when it was exported to the editor the pixels are not directly visible

3.7.5 Test 5

If the program was to be operating in the original way, it would need a big load of negative training examples when more WC and SC were to be added. That cost training time as well as it is difficult to make the computer to handle such big matrices.
But few are the problems that have no solution. Start with a feasible number of negative training examples randomized from some images, just like before. After the first SC all the FP are removed from that set, as usual. Now, let a program use the SC as a detector and scan some images that do not contain any faces, and save all positive detections. These FP get transformed and are added to the set of negative training examples. This solution is not solely computationally efficient, but is also correct from the original theory's point of view. All that is done is simply to add some more negative training examples. The same procedure is repeated after each SC.
The parameters were set to $N_n = 10000$, $T = 20$, $N_s = 10$ and $Fn = 4$. The search for new negative examples is what took the most time. Finding classifiers was done in minutes but updating the negative training set was a matter of hours.
Figure 3.9 show training time and false positives as a function of the number of strong classifiers. The result of the detector can be seen in Figure 3.10. There are a lot of multiple detections as one can see, but they can be used to improve the detection rate. If it is stated that it takes, for example three multiple detections, for being classifying a face, all the single false positives will be removed. I did not implement this improvement because it is more meaningful to use this to tune a system on site, for example a detector at a door.
Counted clockwise and starting at the upper left, the size of the images are 200 by 100, 300 by 222, 57 by 57 and 90 by 290. The detector was set to rescale the images by 0.8, which means that we merge the pixels so that 20% of them vanish. Counted in the same order as before, the number of sub images to be scanned is 29,537, 134,668, 1,931 and 38,634, which equals to 204,771 sub images. If one assume that the images come from a tuned detector then all faces but two in the lower right image would have been detected. I did an arbitrary effort to try to count the FP and defined that to be detections which are not a

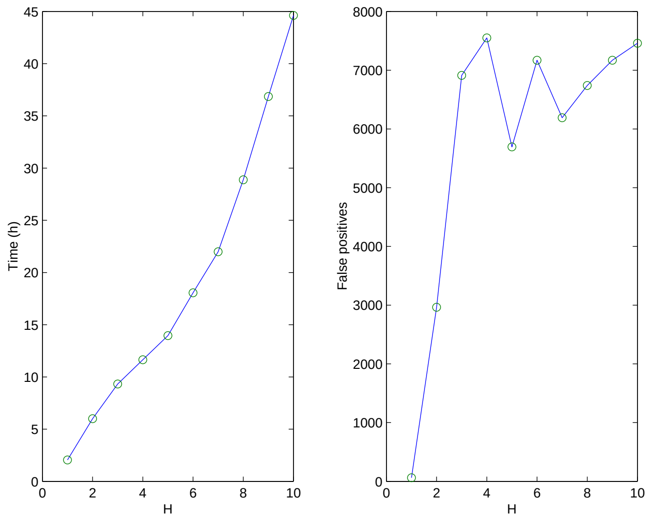

Figure 3.9: A plot of the time passed the number of negative training examples left in training (the false positive) as a function of how many strong classifiers that yet have been trained

part of a multiple (true positive) detection. I got a total of 23 FP detections.
A ROC curve (Receiver Operating Characteristic) is a popular measure for detectors, see for example [4]. A ROC curve plots the rate of true positives, TP, against the rate of FP. The perfect detector reach the point $(1, 0)$.
The four images that were scanned with this detector reach the point $(0.78, 1.1 \cdot 10^{-5})$ in ROC space. The ROC is usually used to illustrate how a variable affects a detector. In this case we only have one point in ROC space and that is neither fun to look at nor that informative. Viola & Jones have an ROC curve that has a TP rate of over 0.98 in a span of FP rate between $5 \cdot 10^{-4}$ and $4 \cdot 10^{-3}$, which rather outclass the current result. One strong reason for that is because they have more training images and that they have different T in each strong classifier which they have manually tried out. It would be a fun experiment to make an optimization program to find the perfect T_j:s for a given number of strong classifiers.
Viola & Jones first SC consists of only two WC and is claimed to reject 60% of the negative training examples. The following 33 SC contain more and more WC, even up to 200 in the last few.

3.7.6 Test 6

This is the final approach and we are going to see how altering the number of WC will affect the final detector. Because of the cascade principle, the detector will benefit from few WC in the first SC and more WC in the last. The number of WC have previously been kept constant.
I tried two methods. In the first one I copied Viola & Jones result that was published in their paper. That test did not turn out very well.
The theory behind the second method is just to add more WC until the number

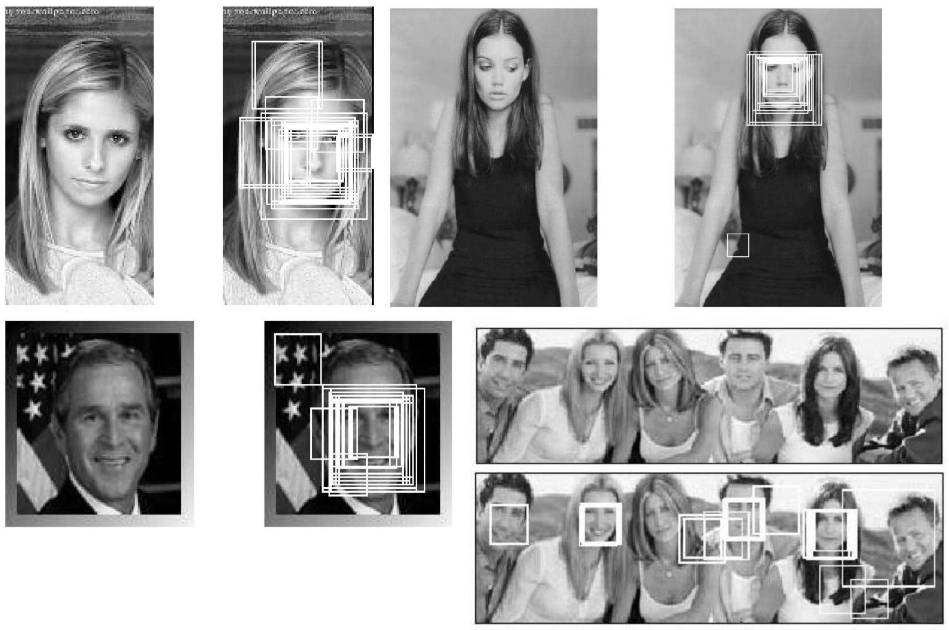

Figure 3.10: Four images scanned by the detector in **Test 5**. Clockwise and starting from the upper left: Buffy, Katie, Friends and Bush

of FP are few enough. Three tests were conducted before the result of the detector was satisfying. Each of the tests used between 300 and 450 images from which negative examples were taken. 200 more positive examples had to be added for the last tests because the prior two tests had too high rate of FN. These faces came from images found on the web. Here is the new, altered, flow chart of my version of windowed-AdaBoost. → are new items in the algorithm.

2a, main boosting loop.

- while the number of number of SC < N_s
- go to **2b**
- remove all true positive that this H finds in the training set
- → Search for new negative training examples using the SC that currently exist as a detector.
- increase the number of SC
- end while

2b, loop to train one SC.

- →while the number of FP > chosen limit
- find the best WC in the same way as before
- →assemble current WC into one SC

- →calculate the number of FP on the training set for this SC
- end while

Below are the parameters with comparicy to earlier tests
$\{t_i\} = 2112$, increased
$N_n = 10,000$
N_p, increased
T, not set. The limit of FP in the training set was instead set to 5500.
$N_s = 19$
$Fn = 3$

Setting N_n, N_s and the limit of FP to their values above is theoretically the same as stating with $10,000 * 1.45^{17} = 5.510^6$ negative examples. That number would produce a matrix that is impossible to handle for most computers.

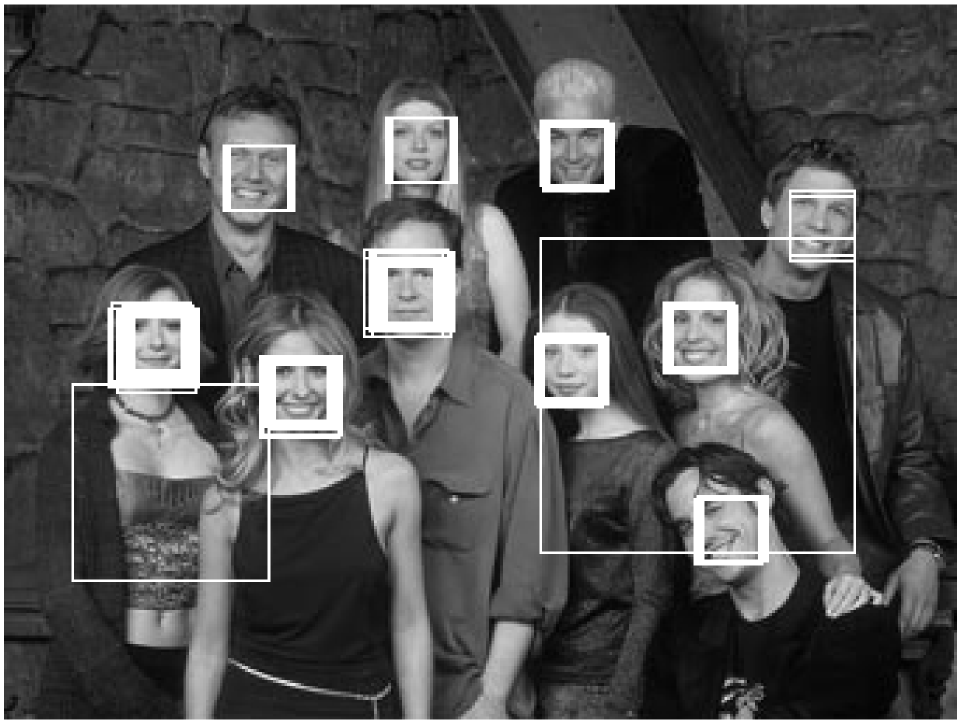

Figure 3.11: The cast of Buffy the vampire slayer

All faces in both Figure 3.11 and Figure 3.12 get multiple detection, so the rate of TP is very close to 1. When tested on five images without any faces in them the result was 108 FP out of 2.1 millon, which is a FP rate of $5.1 \cdot 10^{-5}\%$. The difficulty to measure ROC values lies in that the FP are not evenly distributed between the images. One image got 89 FP while one one got

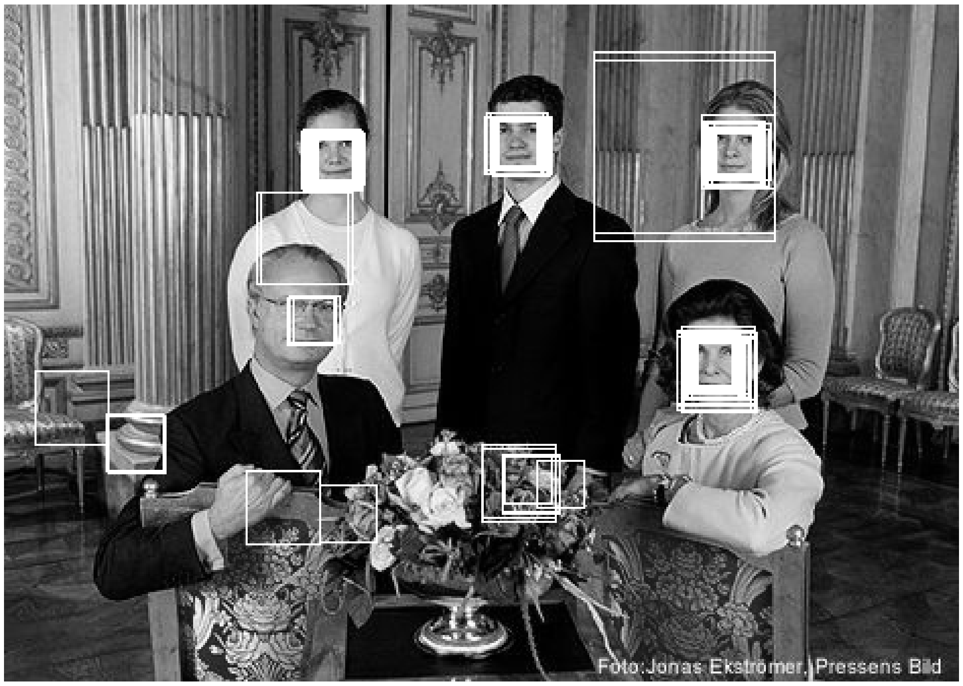

Figure 3.12: The royal family

zero FP. The two images were about the same size. Anyway, measured in ROC space, this result is very close to what Viola & Jones got.

3.8 Further discussion about the window-AdaBoost technique

I believe that this is almost as far as one can come using window-AdaBoost. Updating the negative training set was quite a success which made it possible to train the detector in about one week on an ordinary PC, but who will settle for less then a detector that reach (100,0)% in ROC space?
There are two things I am eager to try. I want to see if the decision surface can be improved and I want to see if I can make the computer to automatically choose the rectangle combinations better than I have done.

Chapter 4

Support Vector Machine, SVM

4.1 Preface

This chapter covers the classification method called SVM. SVM is well known, mathematically well founded and has a history of producing great results as a classifier. Both AdaBoost and SVM produce decision surfaces, but the decision surfaces are usually not the same for the two methods.
In this chapter the most important theory behind SVM is presented and a couple of tests are performed to investigate a SVM based detector. Only the two class case is covered and all objects are images, since this master thesis focus on visual detection.

4.2 Introduction

SVM is a method to find a decision surface that separates training data of two (or more) classes. Once the decision surface is established, the classification of unknown objects is based simply and solely on which side of the decision surface, in a transform space, the object is located. To reformulate in mathematical terms one can say that SVM finds a projection of the image $\mathbf{x}$ into $\mathbf{R}$.

$$C(\mathbf{x}) \rightarrow \mathbf{R}$$

The class of $\mathbf{x}$ is decided by $\text{sign}(C(\mathbf{x}))$.
The image can be represented in any form, as long as it is a vector like pixel values or, as in previous chapter, transform values.
SVM is a well know technique for classification and relies on mathematical statistics, which is appreciated in our field of work. My main source of information was from Christopher J.C. Burges fine work [5], which I found both comprehensive and well written. Some of the mathematics were left out here, because I did not find it necessary nor well founded to completely prove this well known technique once again. However, all the theory needed to train a two class SVM is included here. There is a vast array of papers and books on the subject, so readers who are interested can easily grab deeper into the subject.

4.3 Linear SVM and separable data

For training we have a set of images $\mathbf{x_i} \in \mathbf{R}^n$ with corresponding class label $y_i \in \{-1, 1\}$. There are l number of images in training. Assume that the two classes of training data can be separated by a hyperplane P. Any point $\mathbf{x}$ on P satisfy $\mathbf{x} \cdot \mathbf{w} + b = 0$, where $\mathbf{w}$ is the normal to P. Continue to define two parallel planes to P which lie as close to the nearest point (or points) on each side of P as possible. Name the parallel planes P_+ and P_- and the distances from them to P d_+ and d_-. Figure 4.1 shows what it could look like when $n = 2$.

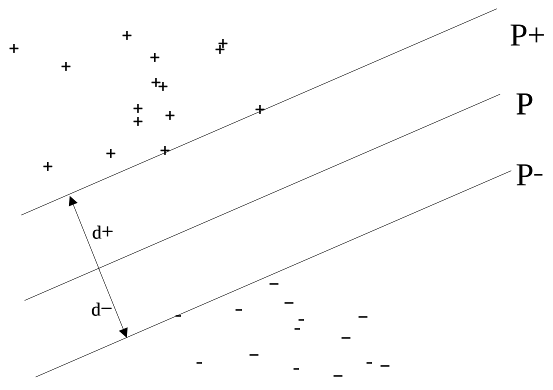

Figure 4.1: In this 2D space the decision surface is a straight line and it separates the two classes without any object being on the wrong side of it. That is called a separable case.

If the training data is linearly separable, the points also satisfy the conditions as follows.

$$\mathbf{x_i} \cdot \mathbf{w} + b \geq 1 \quad \text{for } y_i = 1 \tag{4.1}$$

$$\mathbf{x_i} \cdot \mathbf{w} + b \leq -1 \quad \text{for } y_i = -1 \tag{4.2}$$

The two equations above are combined into the following expression.

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \quad \forall i \tag{4.3}$$

There is a bunch of candidates for P, so a definition of which one is the best is required. It can be shown that the risk for misclassification is lowest if d_+ and d_- is maximized.
The perpendicular distance between the origin and P is $|b|/\|\mathbf{w}\|$. The distance between origin and the other two hyper planes is thus $|b+1|/\|\mathbf{w}\|$ and $|b-1|/\|\mathbf{w}\|$. Subtraction yields $d_+ + d_- = |2|/\|\mathbf{w}\|$.
The very essence of what just have been shown is that maximizing d_+ and d_- is the same as minimizing $\|\mathbf{w}\|$. Here follows the definition of the first optimization problem.

$$\textbf{Primal} \begin{cases} \text{min} & \frac{\|\mathbf{w}\|^2}{2} \\ \text{subj. to} & y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \qquad i = 1 \dots l \end{cases} \tag{4.4}$$

This is a convex quadratic minimization problem. The benefit of this fact is that it guarantees convergence and that every minimum that is found is a global minimum. The problem can be rewritten into a Lagrange formulation. Minimizing the objective function above is the same as minimizing the Lagrangian L. Positive constrains gives positive Lagrange multipliers α_i.

$$L = \frac{\|\mathbf{w}\|^2}{2} - \sum_{i=1}^{l} \alpha_i (y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1) \tag{4.5}$$

Further constrains for the Lagrangian is that derivatives with respect to α_i will be zero.

$$\textbf{Lagrangian} \begin{cases} \text{min} & L \\ \text{subj. to} & \frac{\partial L}{\partial \alpha_i} = 0 \qquad i = 1 \dots l \\ & \alpha_i \geq 0 \\ & y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \end{cases} \tag{4.6}$$

This is also a convex quadratic minimization program because the objective function is quadratic and the constrains form a convex set as they are all linear constrains. This fact makes it possible to define the dual to the problem.
The dual objective function is to maximize L while requiring that derivatives with respect to $\mathbf{w}$ and b are zero and that the Lagrange multipliers are positive. This dual formulation is caller the Wolfe dual.

$$\textbf{WolfeDual} \begin{cases} \text{max} & L \\ \text{subj. to} & \frac{\partial L}{\partial b} = 0 \\ & \nabla_{\mathbf{w}} L = \mathbf{0} \\ & \alpha_i \geq 0 \qquad i \dots l \end{cases} \tag{4.7}$$

The two equality constrains can be inserted into the objective function. They are

$$\mathbf{w} = \sum_{i=1}^{l} \alpha_i y_i \mathbf{x}_i \tag{4.8}$$

and

$$\sum_{i=1}^{l} \alpha_i y_i = 0 \tag{4.9}$$

When(4.8) and (4.9) are inserted into the lagrangian L, it becomes the dual Lagrangian, L_D. The maximization problem comes down to.

$$\begin{array}{c}\textbf{Final}\\\textbf{Dual}\end{array} \begin{cases} \text{max} & L_D = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{l,l} \alpha_i \alpha_j (\mathbf{x}_i \cdot \mathbf{x_j}) \\ \text{subj. to} & \alpha_i \geq 0 \qquad i \dots l \\ & \sum_{i=1}^{l} \alpha_i y_i = 0 \end{cases} \tag{4.10}$$

This program is more preferable from a computational point of view. $\mathbf{w}$ can be computed directly from (4.8), but to retrieve b the KKT conditions will have to be included, which for this kind of problem are booth *necessary and sufficient* conditions. These are the KKT conditions for formulation in (4.6).

$$\nabla_{\mathbf{w}} L = \mathbf{0} \quad (4.11)$$

$$\frac{\partial L}{\partial b} = 0 \quad (4.12)$$

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \quad (4.13)$$

$$\alpha_i(y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1) = 0 \quad (4.14)$$

$$\alpha_i \geq 0 \quad (4.15)$$

KKT condition (4.13) is used to compute b.

But where are the support vectors then? When (4.10) is solved, there are some $\alpha_i \neq 0$. Corresponding x_i are located on $P+$ or P_-, and they are called *support vectors.*

4.4 Linear SVM and *non*-separable data

Non-separable means that no hyperplane can be placed so that it has all the training objects from class 1 on one side of the hyper plane and all from class 2 on the other side. This fact makes the solution in previous section infeasible, since it does not fulfill the constrains in (4.3). This problem is solved using slack variables in the constraint. There are l slack variables ε_i in total.

$$\mathbf{x}_i \cdot \mathbf{w} + b \geq 1 - \varepsilon_i \quad \text{for } y_i = 1 \quad (4.16)$$

$$\mathbf{x}_i \cdot \mathbf{w} + b \leq -1 + \varepsilon_i \quad \text{for } y_i = 1 \quad (4.17)$$

$$\varepsilon_i \geq 0 \quad (4.18)$$

If any training object want to trespass into the wrong side of the hyperplane, the corresponding ε will be greater than 0. The goal is to have as few training objects as possible on the wrong side of the hyperplane, so a penalty is added for every trespass in the objective function $\frac{\|\mathbf{w}\|^2}{2} + C(\sum_{i=1}^{l} \varepsilon_i)^k$. Together with the three constrains above it is a convex minimization problem, if k is a positive integer. For $k = 1$ or 2 it is also a quadratic problem. For k=1, the slack variables will not show up in the (Wolfe) dual so that is the most common value to choose for k.
C is set by the user. A big C will punish a lot for trespassing. For the case when the decision surface can be other functions than a hyperplane (next section) a big C will "bend" the surface more.

This time the Lagrangian is:

$$L = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l} \varepsilon_i - \sum_{i=1}^{l} \alpha_i(y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \varepsilon_i) - \sum_{i=1}^{l} \gamma_i \varepsilon_i \quad (4.19)$$

γ_i are Lagrange multipliers for the constrain (4.18).
The KKT conditions turns out to be like this.

$$\nabla_{\mathbf{w}} L = \mathbf{w} - \sum_{i=1}^{l} \alpha_i y_i \mathbf{x}_i = \mathbf{0} \tag{4.20}$$

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \varepsilon_i \geq 0 \tag{4.21}$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{l} \alpha_i y_i = 0 \tag{4.22}$$

$$\nabla_{\varepsilon} L = \mathbf{C} - \alpha - \gamma = 0 \tag{4.23}$$

$$\varepsilon \geq 0 \tag{4.24}$$

$$\alpha_i \geq 0 \tag{4.25}$$

$$\gamma_i \geq 0 \tag{4.26}$$

$$\alpha_i(y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \varepsilon_i) = 0 \qquad \text{for } l = 1 \ldots l \tag{4.27}$$

$$\gamma_i \varepsilon_i = 0 \qquad \text{for } l = 1 \ldots l \tag{4.28}$$

Again, the same dual is used to compute the problem. Remember that C is a parameter that the user sets.

$$\begin{array}{l} \textbf{Nonsep.} \\ \textbf{Dual} \end{array} \left\{ \begin{array}{ll} \text{max} & L_D = \sum_{i=1}^{l} \alpha_i - \sum_{i=1,j=1}^{l,l} \frac{1}{2} \alpha_i \alpha_j \mathbf{x}_i \mathbf{x_j} \\ \text{subj. to} & 0 \leq \alpha_i \leq C \qquad \text{for } i = 1 \ldots l \\ & \sum_{i=1}^{l} \alpha_i y_i = 0 \end{array} \right. \tag{4.29}$$

The solution to the hyperplane is retrieved from KKT condition (4.20) and (4.27).

4.5 *Non*-linear SVM

In many cases it is not enough to use a hyperplane to separate the classes. It definitely brings more possibilities to find a better decision surface if a curved surface can be used. Instead of defining a function surface in the original space, the data is mapped into another (multi dimensional) space Ω where a linear SVM training can be done. Recall the dual maximization problem (4.29) in last section. The training data appear only as scalar products. Define the mapping function as Φ, then the training data will appear as $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$. We define a *kernel function* $K(\mathbf{x}_i, \mathbf{x}_j) \equiv \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$. The reason for using a kernel function is because it is faster to use the kernel as a functional than taking the long way via the transform and scalar product. The optimization problem becomes as follows.

$$\begin{array}{l} \textbf{Nonlinear} \\ \textbf{Dual} \end{array} \left\{ \begin{array}{ll} \text{max} & L_D = \sum_{i=1}^{l} \alpha_i - \sum_{i=1,j=1}^{l,l} \frac{1}{2} \alpha_i \alpha_j K(x_i, x_j) \\ \text{subj. to} & 0 \leq \alpha_i \leq C \qquad \text{for } i = 1 \ldots l \\ & \sum_{i=1}^{l} \alpha_i y_i = 0 \end{array} \right. \tag{4.30}$$

The decision surface can now be retrieved to us a classifier. b is to be found in the modified equation (4.27):

$$\alpha_i(y_i(K(\mathbf{x}_i, \mathbf{w}) + b) - 1 + \varepsilon_i) = 0 \qquad \text{for } i = 1 \ldots l \tag{4.31}$$

The $\mathbf{w}$ is not needed because the classifier is used directly via the kernel function. If $\mathbf{s_i}$ are the support vectors from the solution and there exist n_s of them, the final classifier becomes as follows:

$$sign\,(C(\mathbf{x})) = sign\left(\sum_{i=1}^{n_s} \alpha_i y_i K(\mathbf{s}_i, \mathbf{x}) + b\right) \tag{4.32}$$

What really has changed from the linear case is the space in which the hyperplane is calculated. That was done by manipulating the scalar product which was replaced by the kernel function K. But can any space and kernel function be used, or are there any restrictions to them for the equations to hold? Yes, there are restrictions and they are summarized in *Mercer's Condition*.

Mercer's condition

Let $g(\mathbf{x})$ be a function with finite L_2 norm i.e $\int (g(\mathbf{x}))^2 d\mathbf{x} < \infty$. For *any* such g, the following must hold:

$$\int K(\mathbf{x}, \mathbf{y}) g(\mathbf{x}) g(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0$$

That did not seem to be such a harsh condition? Well, bear in mind that there are a fair number of functions with finite L_2 norm. Scientists have found three possible kernel functions.

$K(\mathbf{x}, \mathbf{y}) = (a\mathbf{x} \cdot \mathbf{y} + b)^p$	for $p \in Z$ and $ab \in R$	Polynomial
$K(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^2/2\sigma^2}$	for $\sigma \in R$	Gaussian
$K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa\mathbf{x} \cdot \mathbf{y} - \delta)$	for $\kappa\delta \in R$	Sigmoidal

The sigmoidal type is not valid for all values on the parameters and will therefor not be used in this work.

4.6 Setting up a SVM test

The idea is to use the window transform from last chapter for feature extraction and to see if the decision surface constructed by a SVM will make an improved detector. However, there are a number of decisions to make and problems to solve on the way when comparing SVM to AdaBoost.

4.6.1 Single detection or detection in stages?

A single detection means that all the transform dimensions of the features are searched at one time. Detection in stages was used in last chapter and has the advantage that it is way faster because of the cascade effect. A detector in stages is on the other hand more difficult to train.

4.6.2 Choosing dimensions

The size of the sub images that the detector checks are 24 by 24 pixels. That gives the image a dimension of 576. The two window types used in last chapter comes in 2075 different shapes, or dimensions. It is quite obvious that only a few of the 2075 dimensions can be used in order to gain any speed at all. In last chapter some of the detector used up to 200 dimensions (transforms) and a specific search strategy was provided. A strategy has to been found to select which dimensions to use here.

4.6.3 Known problems with SVM

The SVM theory guarantees that the optimization will converge and that every optimum is a global optimum, as shown before. That is until the computer gets its hands on the problem anyway. Computing precision and bad ranked matrixes can cause problems, but it can be controlled if the following is kept in mind.

- Choosing a too big penalty parameter C can cause problems with convergence.
- SVM:s can get in trouble if any dimension has a lot of scatter. To avoid that we need to pick the right dimensions.
- Precision problem in training can occur if transform values between dimensions differ too much.

4.6.4 Setting parameters

These are the parameters we can set.

- C controls the penalty for misclassified objects in training. This parameter has great influence on what the decision surface will look like. A big C generally makes the decision surface to be more bent. C is a critical parameter to set!
- a_i, is a weight for each class. This parameter is multiplied with C so that we can set different penalties for misclassifying training objects from different classes.
- K, kernel function. Polynomial or gaussian.
- N_p, number of positive training examples.
- N_n, number of negative training examples.
- d, the dimension of the SVM, i.e. how many transforms we use.

4.6.5 Software

There are a lot of program packages for SVM training available for free on the market. Since all code so far is done in Matlab, I downloaded the **OSU SVM Classifier Matlab Toolbox (ver 3.00)** developed by Junshui Ma, Yi Zhao, and Stanley Ahalt. It is fast thanks to C++ made mex-files. No additional optimization software is needed, just the training class data.

4.7 Test

4.7.1 SVM + using previous results

This test uses the dimensions selected in **Test 5** (3.7.5). The test will find out which one of AdaBoost and SVM that construct the best classifier from these dimensions.
Parameters were set as follows. $C = 1$, $a_i = 1$, for all i, K was set as a linear kernel (polynomial, degree 0), $N_p = 400$, $N_n = 4000$, $d = 20$ and there are ten stages in the final classifier. The result is shown in Figure 4.2. It is easy to

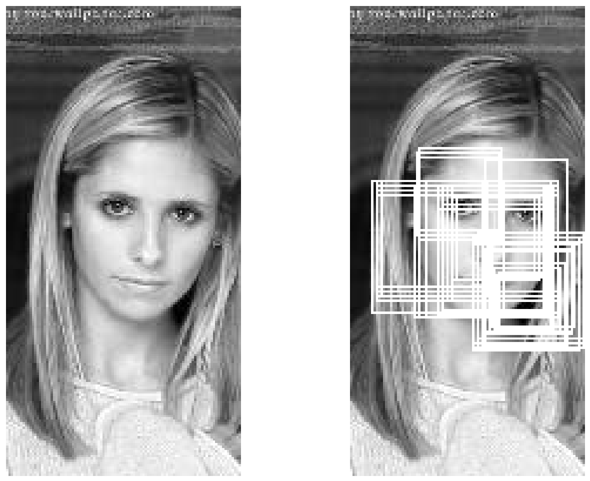

Figure 4.2: The result of the SVM classifier when the class threshold is set to the default value 0

reduce the number of detections here, booth real and false detections. What the SVM classifier really returns when it search an image is not a binary yes or no (given by $sign(C)$), but the whole value C. In Figure 4.2 zero is chosen as threshold. The more an image resembles a face the higher the output value is. If the bar for positive detection is raised to 1, the result is fewer detections as seen in figure 4.3. The fewer detections do not improve the quality, in measure of true positives per false positives.
In order to see if the SVM could beat AdaBoost, the training had to be manipulated. The OSU SVM trainer can be given an input value for the penalty for misclassifying each class. When training AdaBoost, condition were given that there would be at most one percent false negatives in the training set. The first strong classifier from the AdaBoost test had 61 false positives when trained on 10 000 negative images.
The task now is to simultaneously adjust C and the penalty a_i for the two classes until one percent false negatives is reached. If the rate of FP become less than 0.61%, the SVM wins!
As stated earlier, the SVM trainer can be somewhat difficult to get to converge, and it sure was not easier when the class penalty was added. To make things easier for the SVM the dimensions were normalized so that the largest norm, in each dimension of the positives, was equal to one. Normalizing during training

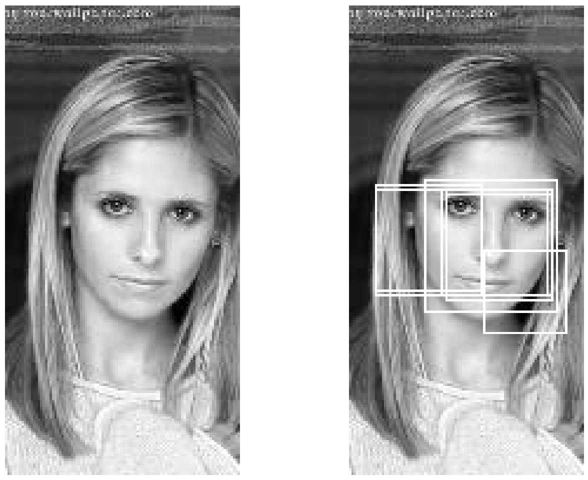

Figure 4.3: The result of the SVM classifier when the class threshold is set to 1

only requires rescaling of the support vectors after the training is complete.
For polynomials with degree greater than two, it was easy to get zero false detections on the training set if penalty C was high enough. Therefor two sets were required, one for training and one for testing. The training set contains 1000 negatives and 300 positives and for testing 1000 negative plus 100 positive. The test result was not that uplifting and the conclusion is that SVM produced very poor results on the test set compared to AdaBoost.
As comparison the corresponding strong classifier from AdaBoost was tested on 5000 negative images and found 47 false positives.

kernel	C	class weight	false pos./1000	false neg./100
linear	10	1:2	36	1
quadratic	10	1:10	31	1
cubic	10	1:10	108	1
gaussian	3	1:2	19	1
AdaBoost			9.4	1

4.7.2 Highest mean

Well, it seems like the SVM surfaces did not apply very well for the dimensions chosen by AdaBoost. But, might there be another set of dimensions that a SVM surface would be perfect for? Let us see what the maximum, minimum and mean for the dimensions look like in Figure 4.4. Remember that every dimension has either maximum one or minimum minus one because of the normalizing. The norm of the highest mean is almost 0.5 for the positive examples which is very high considering that all dimensions are normalized so that the maximum norm of every dimension is one. The 20 dimensions with the highest mean in norm were tried. The question is once again if this set of dimension could beat AdaBoost with a lower rate of false positive under the constrain that at most one percent false negatives would be given in the output. The training was performed under the same conditions as last time.

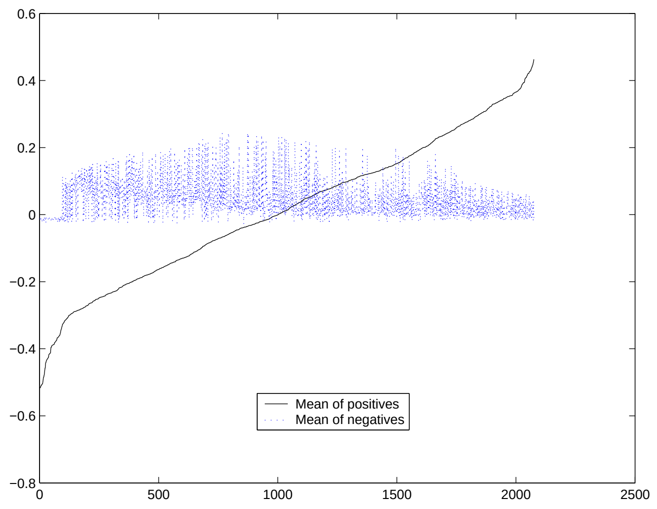

Figure 4.4: Plot of the mean transform values of the training images. The dimensions are sorted by rising mean in the dimensions of the positive training images. The solid line is the dimension mean of positive images and the blur is the corresponding mean from the negative training images

kernel	C	class weight	false pos./1000	false neg./100
linear	1200	1:9	132	1
quadratic	30	1:20	750	1
cubic	26	1:10	652	1
gaussian	0.15	1:5	165	1
original **update**			9.4	1

4.7.3 Ransac

Ransac is a simple method for trying dimensions and train a SVM. However, there is one problem that has to be taken into consideration. To find a 20D classifier takes a lot of manual work to adjust parameters so that the condition to allow at most one percent false negatives comply. All sets of dimensions will require different values for the parameters. It is possible to make a constrained optimization problem out of it, but there is a great risk that the optimization problems are unfeasible, especially for larger C:s. Unfeasability is meant in the sense that the OSU SVM classifier has no interrupt of any kind and can get stuck in a forever ongoing loop. I know this by experience.
There is another way around the problem though. A classifier in 100-D with a quadratic kernel function could quite easily be done and in this case no constraints are needed. The best set of dimensions can be picked out directly from the **ransac**-training. The reason why this works is that 100 dimensions is powerful enough to separate the classes with only a few misclassifications in either direction. A short investigation of the parameters gave that $C = 1$ and the class weights set to 1:1 seemed to be okay for a couple of tested dimension sets. This is how **ransac** works.

```
while forever
    Randomly pick out 100 dimensions
    Train and test a SVM with the 100 dimensions
    If the set of dimensions was the best one yet, then save the set
end
```

Every test round took approximately 16s. Assume that there is one set among all the possible that is superior as classifier, then how long time would it take to find that set? There are 2075 dimensions in total and 100 are taken out at one time. The probability that the first dimension belongs to the superior set is 100/2075. Under the assumption that the first dimension was picked out right, the probability that the second dimension belongs to the optimum set will be 99/2074 and so on and so forth.

$$P(\text{The best set is picked}) = \frac{100\ldots 1}{2075\ldots 1875} = \binom{2075}{100}^{-1} \approx 2\cdot 10^{-173}$$

Since every set in **ransac** is equally likely, the mean number of trials that has to be gone through is $1/2P$, or $2.5\cdot 10^{172}$. If every trial takes 16s, the mean time until the best set is found is $16\cdot 2.5\cdot 10^{172} = 4\cdot 10^{173}$ s, which equals to 10^{157} billion years. I don't really expect to find the optimum set of features during my twenty weeks.
Let's instead hope that a good enough set is much easier to find. The computer got to run over the weekend producing 4500 trials. The training was done on 4000 negatives plus 350 negatives. Testing was done 571 negatives and 50 positives in an attempt to get an even rate between training and testing. The total number of misclassified test images was used as a measure. There were six out of 4500 sets that produced one misclassification in total. It is quite clear that to further continue this training more images have to be used in testing, and/or a later selection has to be added for the best runners up. The results of the best single classifier can be seen in Figure 4.5.

4.8 Conclusions on SVM

The task was to get a set of window transforms and to train a SVM as classifier on that set. The operator has to aim for a single detector or a detector in stages. The latter alternative is better because it is much faster since most of the negative sub-images are rejected using only a few windows (cascade effect). A detector in stages is on the other hand more difficult to train because of constrains on the false negative rate.
Unlike AdaBoost, SVM does not provide a strategy for which dimensions to use. A SVM was trained using the dimensions chosen by a comparable detector trained with AdaBoost. The result was not as good as what was obtained using AdaBoost as classifier. The dimensions with highest mean was also tried for the SVM, but it did not improve results for the classifier.
A single detector in 100 dimensions was created using ransac. It was not a success, but there is a possibility for improvement and the ransac algorithm will definitely keep converging against a global minimum and within an infinity or so, the best 100 dimensions are sure to be found!

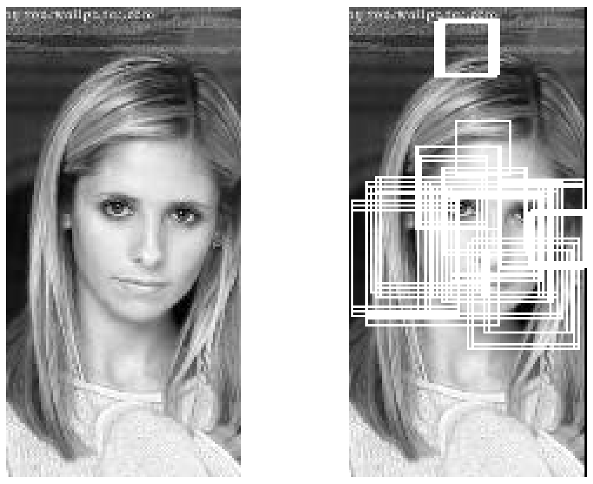

Figure 4.5: The result of the single 100 dimensional SVM classifier trained with ransac

Chapter 5

Genetic programming

5.1 Preface

In chapter 3 I suggested that we should not be limited to use only the two types of rectangle combinations for the window transform as we previously have been using. Genetic programming (GP) is easy to implement as a search algorithm to find better rectangle combinations. Three tests using GP to find windows for a detector are performed in this chapter. AdaBoost and nearest neighbor is used as classifiers. The second AdaBoost test gave better training results than using the rectangle combinations that were done manually in chapter 3!

5.2 Introduction to GP

Genetic programming is a kind of optimization method used in special cases like when gradients are not available or the function is not smooth enough to use conventional optimization programming. The idea is taken directly from nature via Darwin's famous work on the evolution theory. The first idea of evolutionary computing was introduced in the 1960s by I. Rechenberg in his work "Evolution strategies". His thoughts have been developed by many scientists and we can now use this strategy to solve numerous problems thanks to the great computing power we have today. The thing that struck me the most is the theoretical simplicity, even though it is a bit more complicated to tune the program in reality.
Most of the ideas and stages in GP have got parallels in nature. The goal is to find, or evolve, the best possible individual. Being best refers to some quantifiable test that has got to be able to do on each individual and that is called the *fit*. Each individual has got a *DNA* that decides all its features and also what fit it will get. The DNA is not surprisingly built up by *genes*, that are considered to be the fundamentals of each individual. A bunch of individuals together form a *population*. There can be several populations, separated from each other, but there is always a chance that some individuals *migrate* to other populations. Two individuals within a population can *mate* and produce offspring. The children have got some genes from both parents. The parents DNA is often of the same length and the copying starts from parent 1 until the first *crossover*, where copying genes from parent 2 takes over. The crossovers can

occur exponentially distributed, set to a fixed length or maybe occur just once at a random place in the DNA. It is a trial and error decision the user has to make, or even better, implement as another genetic function! During mitosis there can also occur *mutation* that alter one or more genes in a random fashion. One last way of altering the DNA in the offspring is to swap places for the genes the new individual. The location of a gene in the DNA is often important, just like in nature, so swapping places of genes must be done with care and preferably with some knowledge of the consequences or else the program might not converge as desired.
After such a *generation*, the individuals within a population are compared with respect to their fit and some of them (not necessarily the "weakest") are killed and removed from the program.
Almost everything about GP can be found on the website
http://cs.felk.cvut.cz/~xobitko/ga/

5.3 This specific GP

The SVM was supposed to find better decision surfaces, but without much success. Another suggested improvement from chapter 3 was to find better rectangle combinations for the window transform. The original windows are rectangles lying right next to each other. An improvement and a step against more arbitrary windows is if the rectangles were not fixed in size and did not have to lay next to one and other. For a starter only two rectangles per window transform was thought of, but this set of possible combination really blows up (8.1 billion combinations for 24 by 24 size sub images), and that is why GP is introduced. It is very time consuming to make all the windows by hand, not to mention the time it would take to evaluate everything in the AdaBoost. Computation time using AdaBoost is quite proportional to the number of windows that are defined so the attempt is to use GP to find better windows and three test have been conducted on this subject.
There are a lot of parameters to set, but I tried to make things as simple as possible just to see if there was any chance to improve the original Window-AdaBoost. The possibility of multiple populations was excluded. That limitation increase the risk of getting stuck on a local minimum, which may explain why the first test was not successful. This is the list of parameters to set.

- L The numbers of genes per individual.
- N The Size of the population.
- mr_1 Likelihood of type one mutation.
- mr_2 Likelihood of type two mutation.
- cr If crossovers (in mitosis) are assumed to occur exponentially, the mean length before a crossover is cr.
- N_p Number of positive images in training.
- N_n Number of negative images in training.

5.3.1 Test AdaBoost 1

In this test one gene defined a *pair* of rectangles. Each individual had 20 genes to make it comparable to previous tests. Starting out with ten individuals with randomly built rectangles, six of them mated and produced three children in each generation. The mitosis used exponentially occurring crossovers with $cr = 10$. There were two kinds of mutations of the genes. Mutation type 1 changed one side of a rectangle one step in either (possible) direction. Mutation type 2 was more drastic and remade the two rectangles randomly. Parents were chosen to be the six ones with the smallest value τ_i.

$$\tau_i = fit_i\,(1 + |randn|)$$

Here, "$randn$" is a normal distributed random number. This tactic makes it more likely to get a parent with a good fit to become a parent but does not exclude individuals with bad fit value, just as in nature. The new children were sent to an AdaBoost training, where they got to train on 5000 negative and 400 positive images. The threshold for the strong classifier was set to find 99% of the positive images, once again to make it comparable to previous tests. A suiting fit value is of course the number of false positives in the training set, which is also one of the fastest ways to make an evaluation. Due to the AdaBoost training the genes could switch places. The hope was that the switching of places would eventually end while converging because some windows are better at the beginning while other windows are better at the end of the DNA. This assumption is not mathematically founded in any way. The parameters are very arbitrarily chosen and are set as follows.

$$\begin{aligned} L &= 20 \\ N &= 10 \\ mr_1 &= 0.1 \\ mr_2 &= 0.01 \\ cr &= 10 \\ N_p &= 400 \\ N_n &= 5000 \end{aligned}$$

In the beginning the program seemed to converge promising, but it did not last long until a local minimum seemed to have been reached.

5.3.2 Test nearest neighbor

My colleges at Cecil suggested some changes in the program, among others to use *nearest neighbor* instead of AdaBoost.
Nearest neighbor calculates the mean of each class. The mean refers to the mean in each window transform dimension. To classify a 24 by 24 image, the image first get window-transformed. The labelling is based on which class center is the nearest.
Their second proposal was to let one gene be one rectangle instead of being a pair of rectangles. The idea is that it would be more arbitrary and more dynamical to train.
Using nearest neighbor is much faster than using AdaBoost when training a classifier and that means that more generations can be evolved in the same amount of time which let nearest neighbor test more combinations. Nearest neighbor

has on the other hand a much "weaker" decision surface than what AdaBoost produce. The parameters were set to this.

L	$=$	40
N	$=$	10
mr_1	$=$	0.1
mr_2	$=$	0.1
cr		cr was not used, but one randomly placed crossover allways occured
N_p	$=$	400
N_n	$=$	5000

The fact that the decision surface is weaker for nearest neighbor was probably more important than that more generations got tested and the best fit after 23 000 generations (17 hours) was 65, i.e. 65 false positives out of 5000. The original method had 61 false positives out of 10 000 so nearest neighbor does not seem to be a method that can improve the results.

5.3.3 Test AdaBoost 2

This test uses AdaBoost, but in this case each gene incorporate only one rectangle. Using AdaBoost had been so successful before, but training had to be faster because every generation took too much time. What took the most time was to get the pixel sum in all images after a mutation of a rectangle. Rectangles that were not mutated did not have to be recalculated. The AdaBoost was by then well optimized and very fast. There were 40 genes and if they formed all possible rectangle combinations there would be 780 rectangle combinations to try out in total. It sure was worth trying and I set the parameters to this.

L	$=$	20
N	$=$	10
mr_1	$=$	0.1
mr_2	$=$	0.05
cr		cr was not used, but one randomly placed crossover allways occured
N_p	$=$	400
N_n	$=$	5000

3866 generations (14 hours) gave a minimum fit of 26 false positives out of 5000. If that value is comparable to the original window-AdaBoost training this GP program produce better windows than I did myself. The original method had 61 false positives out of 10 000 which is 30.5 false positives out of 5000. The windows evolved for the 20 weak classifiers are found in Figure 5.1

One might be a little suspicious, because the windows are not left-right symmetrical as one might expect them to be. They do in fact look a little "strange". A program that constraint the rectangles to be left-right symmetrical is easily constructed. It may be possible that the left-right symmetry constrain makes the program converge better, but there was no time for more testing on this subject. GP is only included to see if it might work in training and I state that Window-AdaBoost with rectangles selected by GP does indeed work.

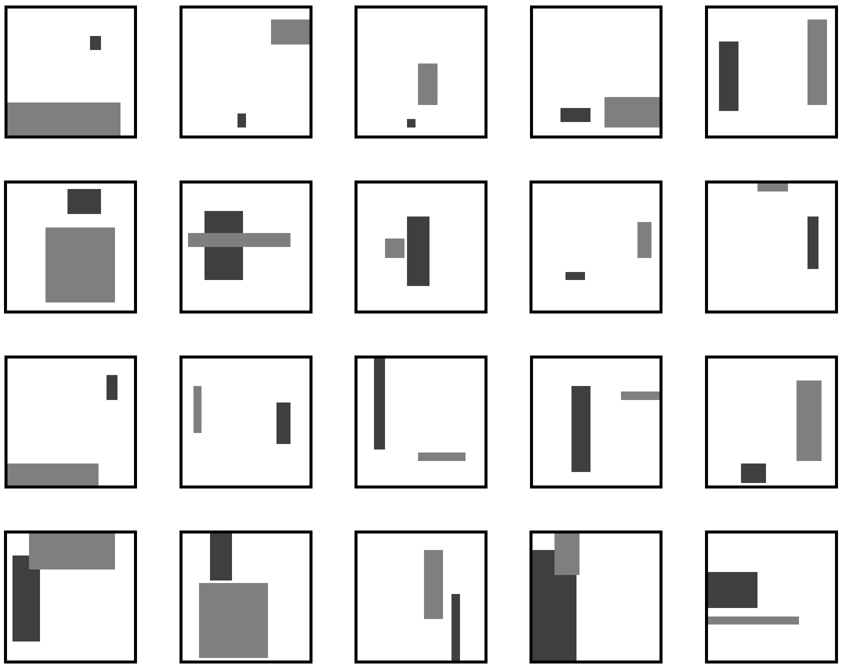
Figure 5.1: The rectangles evolved for the window transform

Chapter 6

CNN-Cellular Neural Network

6.1 Preface

The CNN chip in this text is made for image processing. The chip is briefly presented as well as what operations that are best suited for it. A calculation is included on how fast we can perform the window transform with the CNN chip compared to an ordinary computer.

6.2 Motivation

Every person who has ever dealt with image processing, in the mathematical sense, knows how computationally demanding it is. When we compute an image it is most often represented by its pixel values, so that we can apply our mathematical operations in a familiar fashion. The ordinary desktop computer can only operate on one pixel at one time, simply because that is how they work.
Let us relate to object detection, which is the subject of this thesis. If we recall the methods mentioned in earlier chapters, they search in sub-images that are 24 by 24 pixels which equals to 576 pixels. The ordinary computer thus needs to process all 576 pixels for every sub-image. The detector search a bigger image, say 128 by 128 pixels and do that in all scales. For objects in the smallest scale, 24 by 24, there are 10.816 sub-images in such an image. That sums up to taking 6.23 million pixel values into account for that single scale.
The CNN chip is specifically made for parallel image processing, which means that it process all pixels at one time. Parallel processing obviously requires fewer operations compared to sequentially process each pixel one by one. However, there are some limitations to what operations are available for the CNN chip.

6.3 Basics about CNN

The last generation of CNN works with image of size 128 by 128. Each pixel is connected to its eight nearest neighbors and the input is five matrixes, a one variable function and a constant term. A more arbitrary approach is to let each pixel be connected to neighbors beyond the eight closest, but increasing that radius complicates the chip design immensely for each step. We can describe the process in the current chip with this differential equation. Index i and j refer to location in space.

$$\frac{dx_{ij}}{dt} = -x_{ij} + \sum_{k=i-1,l=j-1}^{i+1,j+1} \mathbf{A}_{k-i+2,l-j+2} f(x_{kl}) + \sum_{k=i-1,l=j-1}^{i+1,j+1} \mathbf{B}_{k-i+2,l-j+2} u_{kl} + Z + z_{ij} \tag{6.1}$$

The time derivative on the left side vanish after an instant and a steady state solution arise. The solution in every pixel is then obtained if $x_{i,j}$ is taken to the left side of (6.1).
The following list explain the variables in the equation.

- **A** is a 3 by 3 matrix for weights from a function f of the neighbors.
- $f(x_{ij})$ is often a step, ramp or sigmoidal function.
- **B** is also a 3 by 3 matrix for weights from another input image **u**.
- u_{ij} is a pixel value from the input image.
- Z is a bias for the whole image i.e. same for all pixels.
- z_{ij} is a bias specific for each pixel.

There are two more matrixes to use. The first one is a mask that can force chosen pixels not to change their values during operation on the image. The second matrix is another mask that allow chosen pixels to be stored in the analogue memory. Pixels in the analogue memory that are not chosen by the writing mask do not change their value.

6.4 CNN friendly operations

6.4.1 Filtering

One can get both high pass and low pass filtering with CNN. Setting the matrixe A and B like this will produce a high pass filtering (edge detection).

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} B = \begin{pmatrix} 0 & -0.5 & 0 \\ -0.5 & 2 & -0.5 \\ 0 & -0.5 & 0 \end{pmatrix} \tag{6.2}$$

A low pass filtering (smoothing)is achieved if we set A and B like this.

$$A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} B = \frac{1}{9} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \tag{6.3}$$

f can be a ramp function.

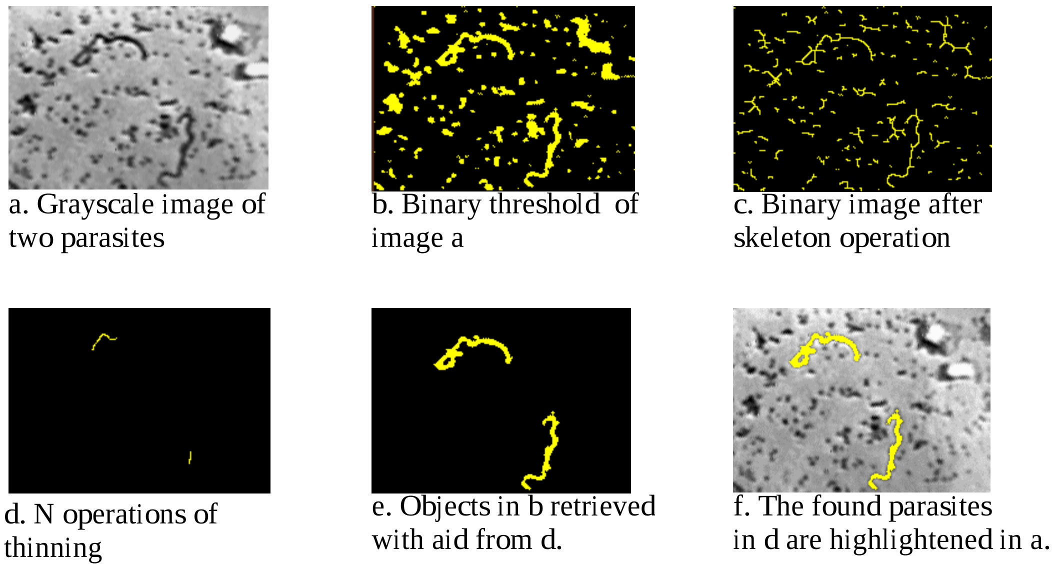
a. Grayscale image of two parasites
b. Binary threshold of image a
c. Binary image after skeleton operation
d. N operations of thinning
e. Objects in b retrieved with aid from d.
f. The found parasites in d are highlightened in a.

Figure 6.1: An example of what is attainable with basic and morphologic operation

6.4.2 Morphology

There are many applications in which morphological operations are used. These operation often demand an immense computational effort. The parallelism of CNN makes it run in just a few operations instead. Let us look at an example where we want to detect parasites in a gray scale image.

6.4.3 Diffusion

The equation 6.1 can be set to work as the heat equation.

$$\nabla_{\mathbf{xx}} u(x,t) = \frac{\partial u}{\partial t} \tag{6.4}$$

A steady state solution requires space boundary condition whereas the transient solution also needs the initial values. The steady state is of less interest in image processing than what transient solution are. Running the heat equation on the initial image for a while will make the image diffuse. The diffusion is made in one single operation and the degree of diffusion is decided by the time you let the diffusion run. Parameters are set like this.

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} B = \emptyset \tag{6.5}$$

f is once again a ramp function.

Chapter 7

Future work

I would like to make some suggestion on how future work in this subject could proceed. This sums up the questions and problems that I have not studied due to lack of time.

7.1 Better feature extraction

Under the assumption that this technique is to be applied to the CNN chip there are two things I expect can be tested for improvement.

1. The size of the sub images have been 24 by 24 pixels throughout the whole thesis. That size does not have to be the best one, especially since faces by nature are higher than they are wide.

2. Start using other shapes than rectangles. Rectangles were used in the window transform solely because they are fast to calculate thanks to integral representation of the images.
   I suggest we look for a mask of two non overlapping and connected areas using genetic programming. That would probably produce a better class discriminating transform. While building this program I suggest a restriction that the mask should be left-right symmetrical, because frontal faces are just that. Genetic programming has shown good result so that is a technique which probably is a good idea to use when looking for the two non overlapping areas.

7.2 Better decision surface

There are two questions here left to answer. What is the optimal number of weak classifiers in each strong classifier and how many strong classifiers should we use? These are both question we must answer using trial and error. Test 6 in chapter 3 show a way to solve the problem of how many weak classifiers to use.

Bibliography

[1] Paul Viola & Michael J. Jones.: "Robust Real-time Object Detection". http://www.merl.com/people/viola/research/publications/CRL-TR-2001-01.pdf

[2] Yoav Freund and Robert E. Schapire.: "A short introduction to boosting". *Journal of Japanese Society for Artificial Intelligence, 14(5):771-780, September, 1999.*

[3] Stan Z. Li, Long Zhu, ZhenQiu Zhang, Andrew Blake, Hongjiang Zhang & Harry Shum.: "Statistical Learning of Multi-View Face Detection". http://research.microsoft.com/∼szli/papers/FaceDet-ECCV2002.zip

[4] N.V. Chavla, K.W. Bowyer, L.O. Hall and W.P.Kegelmeyer.(2002): "SMOTE: Synthetic Minority Over-sampling Technique" *Journal of Artificial Intellegence Research 16* 321-357

[5] Christopher J.C. Burges editor Usama Fayyad.: "A Tutorial on Support Vector Machines for Pattern Recognition"
Data Mining and Knowledge Discovery, 2, 121-167(1998) Kluwer Academic Publishers, Boston

[6] Bernd Heisele, Tomas Poggio and Massimiliano Pontil.: "Face Detection in Still Gray Images"
ftp://publications.ai.mit.edu/ai-publications/1500-1999/AIM-1687.pdf

Chapter 8

Appendix

8.1 Images for training

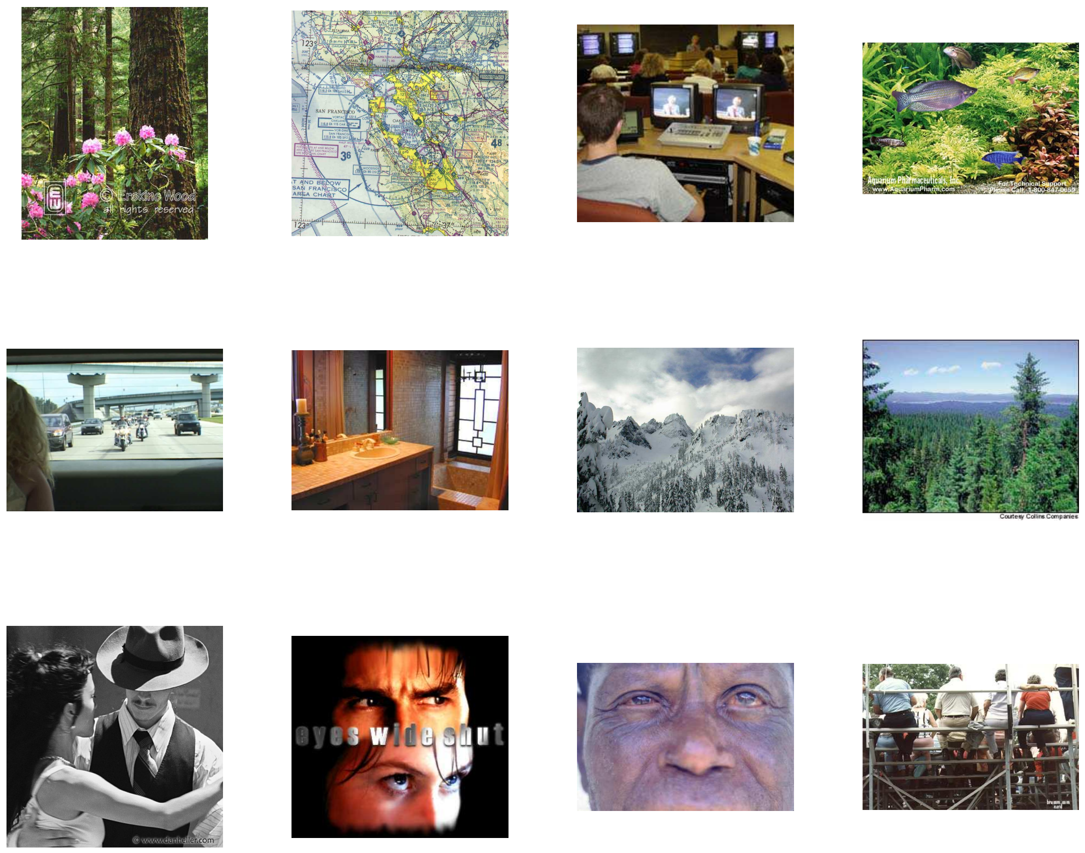

Figure 8.1: Negative training examples were taken from images like this. Up to 450 such images had to be used.

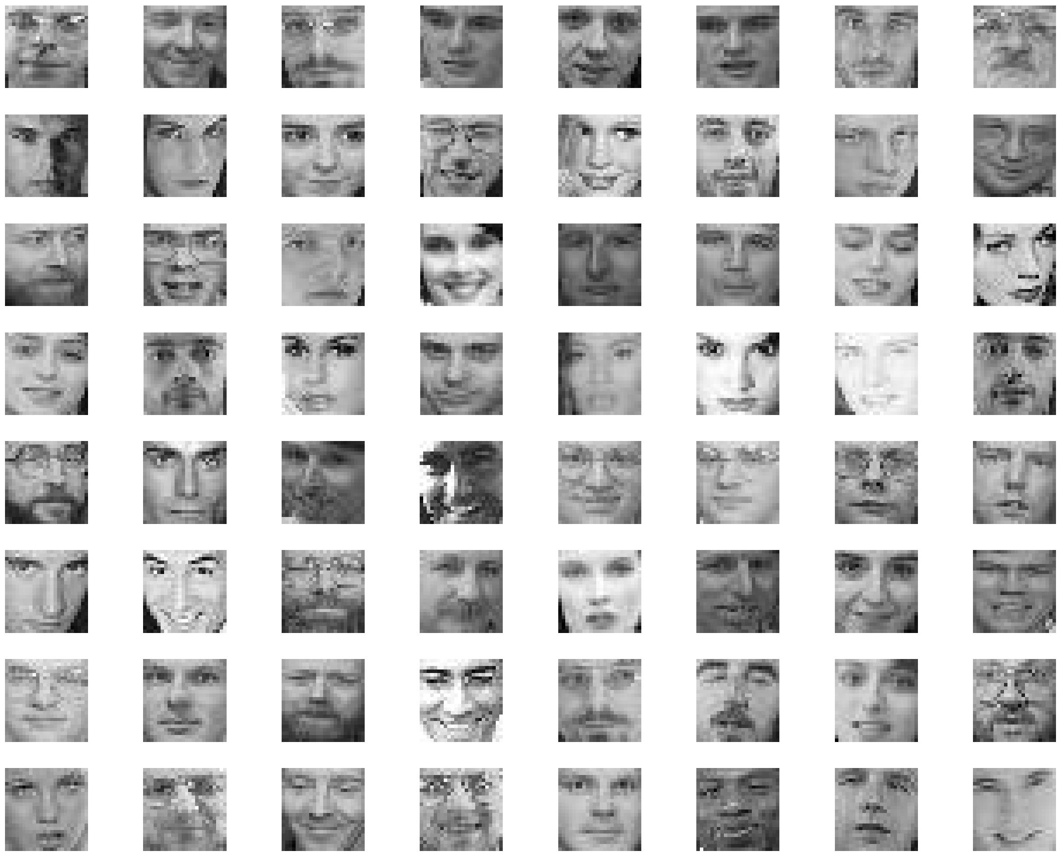

Figure 8.2: Here are 64 of the 600 positive training examples..