Fast Face Detection Using AdaBoost

Julien Meynet

16th July 2003

Acknowledgments

Neuf mois après mon arrivée ici, tout se termine: le projet et les examens en même temps. Une année passe très vite, on a à peine le temps d'apprécier à leur juste valeur les bienfaits de la vie à l'EPFL et en dehors.

Je tiens à remercier tout particulièrement Jean-Philippe pour son soutient et sa bonne humeur quotidienne et bien sur Vlad sans qui je me serai parfois vite découragé et grâce à qui j'ai pu travailler avec motivation et enthousiasme.

Ensuite je ne veux oublier personne au LTS, toutes ces cultures mélangées qui font un joli cocktail tous les jours. Ceux qui sont déjà partis mais que personne n'a oublié : Irene, Naara, Alvaro, François, Issa, Luca, Marc, Yin et les rescapés qui pourront encore profiter du labo quelques temps: May, Vannessa, Ale, Chris, David, Emilio.

Quelques mots à toutes les autres personnes que j'ai pu côtoyer en suisse sans qui mon adaptation ici aurait été difficile et tout ceux que j'ai laissé à Grenoble et qui m'ont beaucoup manqués.

Enfin, je souhaite dire merci à toute ma famille, soutient permanent et beaucoup plus que ça. Mes parents, Sonia et Jérôme, les grands parents et bien d'autres.

Abstract

In this report, a face detection method is presented. Face detection is a difficult task in image analysis which has each day more and more applications. The existing methods for face detection can be divided into image based methods and feature based methods. We have developed an intermediate system, using a boosting algorithm to train a classifier which is capable of processing images rapidly while having high detection rates. The main idea in the building of the detector is a learning algorithm based on boosting: AdaBoost. AdaBoost is an aggressive learning algorithm which produces a strong classifier by choosing visual features in a family of simple classifiers and combining them linearly. The family of simple classifiers contains simple rectangular wavelets which are reminiscent of the Haar basis. Their simplicity and a new image representation called Integral Image allow a very quick computing of these Haar-like features. Then a structure in cascade is introduced in order to reject quickly the easy to classify background regions and focus on the harder to classify windows . For this, classifiers with an increasingly complexity are combined sequentially. This improves both, the detection speed and the detection efficiency. The detection of faces in input images is proceeded using a scanning window at different scales which permits to detect faces of every size without re-sampling the original image. On the other hand, the structure of the final classifier allows a real-time implementation of the detector.

Some results on real world examples are presented. Our detector yields good detection rates with frontal faces and the method can be easily adapted to other object detection tasks by changing the contents of the training dataset.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this report, a face detection approach is presented. Face detection is an essential application of visual object detection and it is one of the main components of face analysis and understanding with face localization and face recognition. It becomes a more and more complete domain used in a large number of applications, among which we find security, new communication interfaces, biometrics and many others.

The goal of face detection is to detect human faces in still images or videos, in different situations.

In the past several years, lots of methods have been developed with different goals and for different contexts. We will make a global overview of the main of them and then focus on a detector which processes images very quickly while achieving high detection rates. This detection is based on a boosting algorithm called AdaBoost and the response of simple Haar-based features used by Viola and Jones[1].The motivation for using Viola's face detection framework is to gain experience with boosting and to explore issues and obstacles concerning the application of machine learning to object detection.

1.1 General idea

Automatic face detection is a complex problem which consists in detecting one or many faces in an image or video sequence. The difficulty resides in the fact that faces are non rigid objects. Face appearance may vary between two different persons but also between two photographs of the same person, depending on the lightning conditions, the emotional state of the subject and pose. That is why so many methods have been developed during last years. Each method is developed in a particular context and we can cluster

these numerous methods into two main approaches: image based methods and feature-based methods. The first one use classifiers trained statically with a given example set. Then the classifier is scanned through the whole image. The other approach consists in detecting particular face features as eyes, nose etc...

1.2 The chosen approach

We have chosen to work in a common context. The goal of this project is to detect very quickly low resolution faces in cluttered background. This situation can be found in many applications as surveillance of public places. The method used is both image based and feature based. It is image based in the sense that it uses a learning algorithm to train the classifier with some well chosen train positive and negative examples. It is also feature based because the features chosen by the learning algorithm are for lots of them directly related to the particular features of faces (eyes positions, contrast of the nose bridge). The boosting techniques improve the performances of base classifiers by re-weighting the training examples. The learning using Boosting is the main contribution of this face detection.

On the other hand, the simple classifiers used for the boosting are simples Haar-like features which permits a fast computation while good detection rates.

1.3 Overview of the report

In the next chapter, an overview of the main existing approaches is given. We first define precisely what the face detection task is and then detail the image based and feature based methods. Chapter 3 explains the developed algorithm. The main theory of Boosting is given as well as the use of the Haar-like masks, a new image representation and an implementation in cascade. Finally the last chapter will focus on the experiments and results of our face detector.

Some implementation issues and detection examples are given in the Annexes.

Chapter 2

Overview of Face Detection

2.1 Introduction

In the following we will present different aspects of the face processing domain while reviewing the main existing methods.

First of all, we need to define what face detection is, why it is an interesting objective and how it can be approached with various methods.

We can define the face detection problem as a computer vision task which consists in detecting one or several human faces in an image. It is one of the first and the most important steps of Face analysis. Usually, the methods for face recognition or expression recognition assume that the human faces have been extracted from the images, but while the human visual system permit us to find instantaneously faces in our purview indifferently of the external conditions, doing the same automatically with a computer is a quite difficult task.

2.1.1 A brief history

Along face detection, many other parts of Face analysis present useful applications and the number of these applications is increasing considerably nowadays with the evolution of the automatic systems in the life of every one of us. Face Recognition, Face localization, Face Tracking, Facial expression recognition are the main of these research domains.

- Face recognition consists in identifying the people present in images, in other words, we want to assign one name to one detected face. It is used in security systems for example.
- Face localization is the problem of finding precisely the position of one face, whose presence is already known in a single image.

- Face tracking has for goal to follow a detected face in a sequence of images in a real world context in most of the cases.

- Facial expression recognition will try to estimate the affective state of detected people (happiness sadness etc...).

It is clear that the first step for all these problems is to find faces in images. For that various approaches have been developed and that is what will be detailed in this section.

The first face detection systems have been developed during the 1970's but the computation limitations restricted the approaches to anthropometric techniques which could be efficient in only few applications as passport photograph identification for instance. It is only since the beginning of th 1990's that more elaborated techniques have been built with the progress in video coding and the necessity of face recognition. In the past years, lots of different techniques have been developed, in such a proportion that today we can count not less than 150 different methods.

2.1.2 Face detection difficulties

If automatic face detection has not been developed before, it is because it is particularly hard to build robust classifiers which are able to detect faces in different image situations and face conditions even if it seems really easy to do this with our human visual system. In fact, the object "face" is hard to define because of its large variability, depending on the identity of the person, the lightning conditions, the psychological context of the person etc...

The main challenge for detecting faces is to find a classifier which can discriminate faces from all other possible images. The first problem is to find a model which can englobe all the possible states of faces. Let's define the main variable points of the faces:

- The face global attributes. We can extract some common attributes from every face. A face is globally an object which can be estimated by a kind of ellipse but there are thin faces, rounder faces... The skin color can also be really different from one person to one another.

- The pose of the face. The position of the person in front of the camera which has been used to acquire the image can totally change the view of the face: the frontal view, the profile view and all the intermediate positions, upside down...

- The facial expression. Face appearance depends highly on the affective state of the people. The face features of a smiling face can be far from those of an indifferent temperament or a sad one. Faces are non-rigid objects and that will limit considerably the number of detection methods.

- Presence of added objects. Face detection included objects that we can usually find on a face: glasses which change one of the main characteristics of the faces: the darkness of the eyes. Natural facial features such as mustache beards or hair which can occult one part of the face.

- Image Condition. The face appearance vary a lot in function of the lightning conditions, the type of illumination and intensity and the characteristics of the acquisition system need to be taken in account.

The next figure shows some typical face examples extracted from the CMU test dataset [23].

Figure 2.1: Typical faces extracted from the CMU Database [23]. We can notice the great variability of the non-rigid object "Face".

The background composition is one of the main factors for explaining the difficulties of face detection. even if it is quite easy to build systems which can detect faces on uniform backgrounds, most of the applications need to detect faces in any background condition, meaning that the background can

be textured and with a great variability. So our two class classification task is to assign an image to the face class or the Non faces class. Given a set of examples, we can extract some properties of faces for representing the face class but it is impossible to find properties which can represent all the non face class. Some images with complex background are presented in the figure 2.2.

Figure 2.2: Examples of faces in a complex background. These images are taken from the CMU Database [23].

In this context, various approaches have been taken to detect faces in images. But as the face detection task is quite complex, each method is build in a precise context and we will now review the main existing methods. The next sections detail the two main face detection approaches:

- **Image-Based methods** which are built given a set of examples and uses a sliding window to perform the detection.
- **Geometrical-Based** methods which take in account geometric particularity of face structures.

2.1.3 Definition of some general notions needed to understand face detection problem

First of all, we have to define some basic criteria that will determine the performances of the detectors. The first notion that we need to introduce is the detection rate. The detection rate d is the percentage of faces in the image that have been correctly detected by the detector. In lots of applications, it is the rate that we want to maximize. On the other hand, we have to define the false rates. The false negative f_n rate is the opposite of the detection rate in the sense that it is the rate of faces that have been forgotten by the detector: $f_n = 1 - d$. The false positive rate is the second essential rate considered in face detection: let f_p be the rate of non faces windows that are classified as faces by the detector. Due to the large number of windows evaluated in an usual image, this false positive rate is usually 10^{-5}or 10^{-6} but this low value is not really significant.

Once this definitions are given, it is easy to understand that the objective of the face detection is to maximize the detection rate d while minimizing the false positive rate f_p. However, as in lots of applications in the real life, it is hard to have both low false positive rate and high detection rate, and that is why we have to look for a trade off between the two parameters. All the methods described in the following sections will try with different approaches to find the better compromise between false positive rate and detection rate. Finally, we will see that it is hard to compare the methods because of the problem of detection evaluations and of the different contexts. How can we measure the goodness of a detector?

2.2 Image-Based Detection

2.2.1 Introduction

The Image-Based methods have been doubtless the more used until today. We qualify them of "Image-Based" because they are built using example images in opposition to some "template-methods" which need an *a-priori* knowledge about faces. In order to extract the features from some training examples, we will need to follow a statistical learning approach or other machine learning algorithms. The principle is to learn a face and a non-face distribution, given a set of positive and negative examples. For this, we will naturally be placed in a probabilistic context : An image or any input data is considered as an random variable x and the two classes *face* and *Non face* are characterized by their conditional density functions: $p(x|face)$

and $p(x|non-face)$ (see [22]). It is obvious that these density functions are unknown and one of the main goals is to approximate them in order to discriminate faces and non faces. Then there are several methods to find discriminant functions with permit to classify a given example in the face class or the non face class. In this probabilistic approach, many different methods exist, among which Eigenfaces, Fisher's Linear Discriminant and Neural network or support vector machines etc...

The main difficulty in this approach is that the example dimension, i.e. the dimension of x is often high so an important step will be to reduce this example space in order to find a discriminant function which separates positive and negative examples.

2.2.2 Eigenfaces

2.2.2.1 Definition

The first Image-Based method that we will describe in this section is called Eigenfaces. The principle of face detection using Eigenfaces is to extract these features from a set of images by Principal Components Analysis (PCA) and estimate if the extracted Eigenfaces correspond to typical face pattern. In fact all input images can be represented by a weighted vector of Eigenfaces in the eigen space and the challenge is to determine if this linear combination is closer to one class or to the other. A global overview of face recognition using Eigenfaces can be found in [25] .

2.2.2.2 Principal Components Analysis

The first step for this Eigenfaces classification is to extract the Eigenfaces from the original images. For this, the Principal Components Analysis (PCA) is used. PCA which is also known as the Karhunen-Loeve method reduces the input space dimensionality by applying a linear projection that maximize the scatter of all projected samples. This subsection presents the main steps of such an analysis.

Let $\{x_1, x_2, ..., x_N\}$ be a set of N images which are values from a *n-dimensional* feature space. The orthonormal matrix W define a linear transformation from the *n-dimensional* space to a *m-dimensional* feature space where $m < n$ (dimensionality reduction). Noticing that $W \in R^{n \times m}$ the new feature vectors $y_k \in R^m$ are defined by the linear transformation:

$$y_k = W^T x_k, \quad k = 1, 2, ..., N. \tag{2.1}$$

Then the total scatter matrix S_T is defined as

$$S_T = \sum_{k=1}^{N} (x_k - \mu)(x_k - \mu)^T \tag{2.2}$$

where μ is the mean of all the examples : $\mu = \frac{\sum_{k=1}^{N} x_k}{N}$.

By applying the linear transformation, the new scatter matrix in the *m-dimensional* subspace is given by $W^T SW$. The PCA theory shows that the optimal linear projection W_{opt} is the one which minimizes the determinant of the projected scatter matrix (for the samples $\{y_1, y_2, ..., y_N\}$), i.e.

$$W_{opt}^T = \arg\max_w |W^T S_T W| = [w_1, w_2, ..., w_m]. \tag{2.3}$$

The set $\{w_i | i = 1, ..., m\}$ are the *n-dimensional* eigenvectors of S_T, corresponding to the $\{\lambda_i | i = 1, ..., m\}$ eigenvalues ordered decreasingly.

This projection in the feature space using W_{opt}^T permits to decompose the distance between an example and the face space into two components: the distance in feature space $DIFS$ (projection on the m dimensional space) and the distance from feature space $DFFS$. For more details about PCA, see [26], [27] and [28].

One serious point is that the main variance cause in an object class is the lightning variations as shown in [29] . The optimal linear transformation W_{opt} given by PCA has the drawback to focus on components representing the illumination changes. One of the correction methods is to let out the first principal Eigenfaces considering that they contain almost all the variations due to lightning.

2.2.3 Fisher's Linear Discriminant

Even if the Eigenfaces method seems to be quite efficient on non noisy images, one of the drawbacks is that it does not minimize the intra-class variance. A good classifier is a classifier in which the model of each class has a small variance while a large variance between different classes. Fisher's Linear Discriminant (FLD) is one method to find the optimal projection. The projection determined by $z = W_{FLD}^T x$ minimize the quantity $\frac{S_{BC}}{S_{WC}}$, which is the ratio between the between class variance S_{BC} and the within class S_{WC}, see [24]. If we consider the general case of a c-class problem, then we can define the between class covariance matrix by :

$$S_{BC} = \sum_{i=1}^{c} N_i (\mu_i - \mu)(\mu_i - \mu)^T \tag{2.4}$$

and the within class covariance matrix by :

$$S_{WC} = \sum_{i=1}^{c} \sum_{x_k \in X_i} (x_k - \mu_i)(x_k - \mu_i)^T \tag{2.5}$$

where μ is the mean of all the samples, μ_i is the mean of the class X_i and N_i the number of samples in the class X_i.

The optimal projection is obtained if we choose the projection matrix W_{FLD}^T as follow:

$$W_{FLD}^T = arg\, max_\omega \frac{|W^T S_{BC} W|}{|W^T S_{WC} W|} = [w_1, w_2, ..., w_m], \tag{2.6}$$

where $\{w_i | i = 1, ..., m\}$is the set of generalized eigenvectors of S_{BC} and S_W, which are associated to the eigenvalues $\{\lambda_i | i = 1, ..., m\}$.

In [24] is it shown that the upper bound for the projection space dimension is $c-1$where c is the number of classes. In our binary class case, the projected space is a line.

An example in the next figure shows the comparison between the two methods: PCA and FLD.

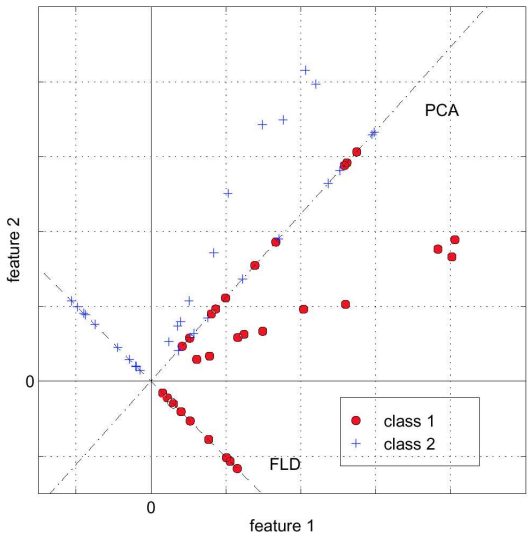


Figure 2.3: A comparison of principal component analysis (PCA) and Fisher's linear discriminant (FDL) for a two class problem where data for each class lies a linear subspace. (taken from [24]).

2.2.4 Other methods in Eigen-space

Others methods which use dimensionality reduction in the image space have been developed. One of the most efficient is the distribution based model developed by Sung and Poggio (see [4]). The method consists in modeling both the distribution of face patterns and non face patterns. The face distribution is modeled using 6 face pattern prototypes clustered by a modified version of the *k-means* clustering algorithm. This algorithm computes the 6 centroids and covariance matrix of the 6 multi-dimensional Gaussian. In order to decrease the number of misclassified examples, 6 other Gaussian clusters representing the non face class are built using some critical non face pattern which are face-like patterns in the sense that their prototypes are close to the face models. These face-like non faces are chosen using a Bootstrap method which mean collecting the false positive patterns detected on a large set of images. Given these 12 clusters, a candidate window pattern has to be classified as face or non face. For this, each the distance between the tested pattern and the 12 clusters centroids are computed using 2 metrics. The first component is normalized Mahalanobis distance between the tested pattern's projection and the cluster centroid in a subspace spanned by the cluster's 75 largest eigenvectors. The second is the Euclidean distance between the test pattern and its projection in the subspace. So the entire set of 12 distance measurements is a vector of 24 values. Then a multi-layer perceptron (MLP) is used to separate the positive and negative examples. This approach is quite powerful but the limit is that the choice of all the parameters is not clear : what is the optimal number of clusters, how many examples do we have to use to train the classifier?

One other interesting method is a Bayesian based model.

2.2.5 Neural Network, SVM, HMM, Winnow

Other machine learning tools can be used to train good classifiers. Among these learning approaches, we can find neural network oriented systems and support vector ones. These are the more popular tools in machine learning and the most common used nowadays. The next two subsections expose them and make an overview of the different existing systems using them.

2.2.5.1 Neural Network

One of the best face detection system in term of false positive rate and detection rate is a Neural Network-Based face detection developed by Rowley [11]. It uses a retinally connected neural network which decides if a scanned

windows is a face or not. The face detection system can be divided in two main steps :

- **A neural network-based filter.**

  The input of this first stage is a pre-processed square image (20x20 pixels in [11]) and the output of the neural network is an real value between -1 and +1. The preprocessing and neural network steps are presented in the next figure.

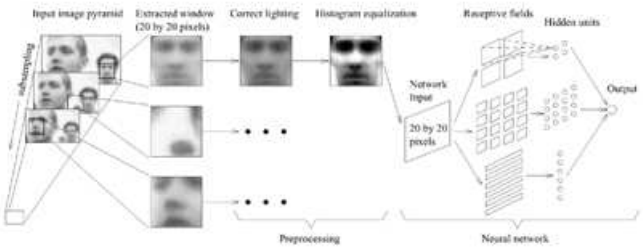


Figure 2.4: Neural Network-based face detection proposed by [11]

  The original image is decomposed in a pyramid of images (by simple sub-sampling) in order to detect faces larger than the basic detector size. The Receptive fields and Hidden units are shown in figure. There are three types of hidden units to represent local features that represent well faces. This first stage yields good detection rates (if the training set is particularly well chosen) but it remains still an insufficient false positive rate.

- **Arbitration and merging overlapping detections.**

  In order to improve this high false positive rate, two neural networks are trained with various initializations (in term of non face training set, weight initialization and order presentation). These two networks are built by the methods of the first step.

  Even if the two networks have individually bad false positive rates, the false alarms may differ from one network to the other. Hence, an integration of the result using a simple arbitration strategy improve significantly the detection results. The most common of these strategy is called ANDing. A window if definitively classified as face only if the two neural networks have detected it.

This method using neural networks have good results in term of false positive rate and detection rate, but one limitation is that the quality of the detection

depends highly on the coherence of the training sets and on the tuning of the neural networks which has lots of parameters.

2.2.5.2 Support Vector Machine

Support Vector Machine is a learning technique introduced by Vapnik [19]. It seems to be efficient when the data sets become larger than few thousands. It the case of face detection if we want to describe precisely all the faces (because of the variability of faces.) The principle is to find the decision surfaces by solving a linearly constrained quadratic programming problem.

The hyperplan decision is the one that maximize the margin between the face and the non faces classes. One of the simple margin that can be used is the distance between the closest points of the two classes. The points that are kept in the hyperplan are not numerous. They are called supper vectors but they are the most important because they define the boundary between the two classes. Osuna and al. have developed such a face detection system using Support Vector Machine in [49].

2.2.5.3 Hidden Markov Model

These Hidden Markov Models have been used by Samaria and Young (see [41] and [42]) for face localization and recognition. The principle is to divide a face pattern into several regions such as forehead, eyes, nose, mouth and chin. A face pattern is then recognize if these features are recognize in an appropriate order. In other words, a face pattern is a sequence of observation vectors where each vector is a strip of pixels. A image is scanned in a precise order and an observation is taken by block of pixels. The boundaries between strips of pixels are represented by probabilistic transitions between states and the image data within a region is modeled by a multivariate Gaussian distribution. The output states correspond to the class to which the observation belong. Other methods using HMM have been developed by Rajagopolan [44], and Sung [43].

2.2.5.4 Sparse Network of Winnows (SNoW)

SNoW is a sparse network of linear functions that uses the Winnow update rule defined in [45]. We define two linear units called target nodes: one as representation for the face pattern and another one for the non-face pattern. Given a set of relations that may be of interest in the input image, each input image is mapped into a set of features which are present in it. This representation is given to the SNoW procedure and propagates to the target nodes. Let $A_t = \{i_1, ..., i_m\}$ be the set of features that are present in an

example and are linked to the target node t. Then the linear unit is active if and only if $\sum_{i \in A_t} w_i^t > \theta_t$, where w_i^t is the weight on the edge connecting the $i-th$ feature to the target node t and θ_t is its threshold. The Winnow update consists in a threshold θ_t at the target t, two update parameters: a promotion parameter $\alpha > 1$ and a demotion parameter $0 < \beta < 1$.

2.3 Geometrical-Based Detection

2.3.1 Introduction

The previous statistical methods are based on a learning to obtain a face model from one positive and one negative data set. They are not directly correlated to the particular geometrical features of a typical face. Some other methods are in such a point of view. They are called Geometrical-based or Feature-based. Many approaches have been taken is this large area of feature-based detection and we can distinguish:

- **the top-down approach** : One model is computed for one scale. This was used by Yang and Huang [30], and Lanitis [31] .
- **the bottom-up approach :** The faces are searched in an image by the presence of facial features. See Leung [32] and Sumi [33].

The main advantage of this geometric approach is that the face detection is not restricted to frontal faces. In fact the main face features (eyes nose, skin color etc...) are present independently of the pose and the lighting conditions.

2.3.2 Top-down methods

This category includes all the methods that used a multi scale approach. The great majority of them use the skin color to find faces in images. The existing system use several segmentation algorithms to extract faces from the images. The more classical ones are region growing, Gibbs Random Field Filtering and more...

The skin color is maybe one of the features the first noticed by the human visual system. Many methods use different color spaces. The main advantage of this approach is that the face detection is very fast. However, there is one important issue: lots of problems appears if the background contains faces of the skin color.

Yang and Ahuja [36] have build their system in this sense. Although the human skin color seems to change from one example to one other, the

effective variation is more luminance than the color itself. The distribution is modeled by a Gaussian distribution. All the pixel are tested and we attribute them the skin color if their corresponding probability is greater than a given threshold. Finally, a region is declared as face if more than 70 percents of its pixels have the skin color.

Another method proposed by Saber and Tekalp [47] uses Gibbs Random Field filtering as segmentation algorithm.After the segmentation, each region is approximated by an ellipse. the distance between the ellipse and the region shape is computed using the Hausdorff distance measure. If this last measure is greater than a predefined threshold, the region is rejected. Then a procedure of finding the facial features is applied.

Wei and Sethi uses a quite different approach in [**?**, **?**]. They use a partitioning of the human skin region to detect faces. The binary image of the segmented skin is obtained by performing skin color classification at each pixel location. The a morphological closing is performed followed by an opening to remove small regions. Then the remaining regions are another time approximated by ellipses.

2.3.3 Bottom-up methods

The principle is to find invariant features of faces. By invariant, we mean invariant by scaling, poses, lighting conditions and other variations. The common and natural features that are usually extracted are the eyes, the nose the mouth and the hair line. Any edge detector might be used to extract them. A bottom up method try to find this features in an original image and then they are grouped according to their geometrical relationships.

The difference between the methods in this bottom-up approach resides in the way to choose the features and how to establish the links between them.

One of the early methods was proposed by Govindaraju in [46]. In this method, the facial features are characterized by curves and structural relationships which link them. Two successive stages are applied: First, curves of the faces are extracted from an input image to find the face candidates. The features detected are then grouped using a matching process (a cost function and one threshold).

Leung [32] uses a random graph matching by apply a set of Gaussian filters which is compared to a template graph representing a face. (The comparison between the computed graph and the template is usually a simple threshold).

In another method used by Yow and Cipolla [34], a set of derivative filter is apply in order to select edge features like the corner of the eyes for

example. Then only the points that have particular properties are kept: those which have parallel edges for example. The remaining points are then linked together and they are used to build a face model.

Cai and Goshtasby [35] used the color information but in a different approach than [36]. A face is recognized by the presence of particular features which do not have the same color than the skin) in a skinned color region.

2.4 Evaluation difficulties

As the definition of detecting faces in images is really simple: determine whether or not there are any faces in images and, if present, return the image location and extend for each face, we can think that it is easy to evaluate the performances of a face detector. However many parameters have to be taken in account to do this. How can we measure the goodness of a detection? How do we have to integrate the false alarms (how do we have to consider the false positive rate?) What about the detection speed? Several such questions make hard the face detection evaluation.

It would be interesting to compare the existing methods in face detection but the major problem is that every method is made in a particular context and today there are still no standards for face detection evaluation that will make easier the future research work about face detection.

The first step in detection evaluation is to use a common testing set which contains a large variety of situations. The most common used set is probably the CMU testing set which contains many faces manually labeled (see Fig. 2.2 for examples). Then we usually use the detection rate over false positive rate ratio to characterize the performances even if the number of false alarms is directly related to the way how the images are scanned (more precisely the number of sub-windows scanned). A summary of the main results and method comparison can be found in [22] and [37]. Nevertheless, we can give general observations about the different approaches. The image based techniques are quite efficient regarding the frontal face detection. The detection rate reaches more than 90% with at most several tens of false alarms in a typical sized image but the main limitation of the image based methods is that the faces detected will slightly match with the training examples. Thus it is difficult for example to include in the training set faces at many different poses, with both rotations in and out of plane. The geometrical approaches are more robust in term of face pose, i.e. the face orientation in front of the acquisition system but they generally give worse detection results. Both the segmentation part and the feature extraction are critical points. The use of the color information needs is not really representative of faces because lots

of objects in the background may have the human skin color.

The speed of the detector is without any doubt the parameter the most difficult to take into account. Each method has its own speed and it is difficult to determine the speed performances: it depends on the scanning method and on the way it is implemented.

So it has been shown that a lot of different approaches are available but face detection is still an open task. Many solutions are possible taking into account the results of the existing methods. The main promising approach seems to be combined approaches of image based and feature based methods. We will see one of them which uses a Boosting learning algorithm in the next chapters.

Chapter 3

Fast Face Detection Using AdaBoost

3.1 Introduction

3.1.1 Choice of the method

In this third chapter we discuss about a face detection based on a boosting algorithm which yields good detection rates. This detector is highly inspired by the Robust Real-time Object Detection of Viola and Jones [1]. We have chosen to build a model using a statistical learning given some positive and negative examples. A learning algorithm trains a classifier by selecting visual features, so we will discuss why this chosen algorithm is appropriate for face detection and explain how it works. We will also emphasis on some other essentials key contributions like a new image representation, the choice of these visual features and finally the introduction of a detection in cascade.

3.1.2 Context of the frontal face detection

Before going into details, we just remark that every face detection method is designed in a particular context, that is why it is not always easy to compare the results between them. Some detectors have for only goal to have a detection rate as near as possible from 100% but our project is a little bit in a different context: even if we naturally want reach good detection rates, we want to build a real-time oriented detector. So the goal is to detect all the faces (or almost all of them) even if this means we have to accept a higher false positive rate (non-face images labeled as face by the detector). This choice is only in order to respect most of applications which

need for example to detect all the people in front of a video camera. (Video surveillance for instance).

On the other hand, if for example, a camera is placed in a airport hall, the faces are often low resolution faces, at different scales and the background seems to be quite textured and complicated. In this way, we have to built a robust detector with respect to illumination, face variation and face size. On the other side, if we keep in mind that we want to detect faces for a further face recognition or comprehension, it would be good to select only faces which can be considered as frontal faces, this will explain the choice of the training set used to learn the final classifier (see 4.1). To summarize, even if we could choose other face detection contexts, this one seems to be the most used in the real-world applications. We will particularly pay attention to the fact that it will be interesting to build a simple and unbiased representation that can represent faces. (And objects by generalization).

3.1.3 Why Boosting and Haar features?

The chapter 2 presented the main approaches available to build a face detection system. Now the context of our face detection is given, we can explain why we choose this approach using a boosting learning algorithm and simple Haar features. As we want to detect faces in various background and principally low resolution faces, it would be improper to use purely geometrical methods. In fact the main advantage to these geometrical methods is the geometric invariant properties. We are not interested by them because we have chosen to stay in a frontal face detection context. So it is quite naturally that we have oriented our choice towards learning algorithms. Boosting is a powerful iterative procedure that builds efficient classifiers by selecting and combining very simple classifiers. Good theoretical results have been demonstrated, so we have some theoretical guarantees for achieving good detection rates. This idea is interesting in the sense that a combination of simple classifiers may intuitively give a rapid detection without deteriorating the detection rates. So it seems to be one of the best compromise between efficiency in term of detection and speed.

3.1.4 Overview of the detection

This new method given by Viola[1] is a combined method of more traditional ones like geometrical and image based detection. It is a geometrical in the sense that it uses general features of human faces: position of particular features among which the eyes the nose and the mouth. We will not try to extract particular face features: It is only an *a-posteriori* observation in the

sense that the selected Haar-like masks are effectively representing particular facial features but it is not our decision. See section 4.2 for details about the selected features. On the other hand, it is also image based because we use a statistical learning with the use of a consequent data set needed to build the face model.

Viola has developed this face detector in July 2001 and he was inspired by the work of Papageorgiou[2]. It seemed to be the fastest and the most robust and it is still today. The speed of the detection is notably given by the simplicity of the features chosen and the good detection rates are obtained by the use of the fundamental boosting algorithm AdaBoost which selects the most representative feature in a large set.

To have a concrete idea of the performances of the detection, imagine that Viola's detector can process 15 frames of 384x288 pixel images per second on a conventional 700 MHz Intel Pentium.

Let us look at the main steps of the fast face detector that will be explored in the next sections.

The detector consists in scanning an image by a shifting window at different scales. Each sub-window is tested by a classifier made of several stages (notion of cascade). If the sub-window is clearly not a face, it will be rejected by one of the first steps in the cascade while more specific classifier (later in the cascade) will classify it if it is more difficult to discriminate.

The first contribution is the choice of the features that describe the faces. The principle of the detection is to apply successively simple classifiers to combine them in a final strong classifier. The choice of these features is fundamental for the performances of the detection. The difficulty is to find masks simple enough to permit a fast classification but characteristic enough to discriminate faces and non faces. A good compromise for that is obtained by the use of reminiscent of Haar Basis functions. In fact the feature response is nothing more than the difference of two, three or four rectangular regions at different scales and shapes. To improve the computation time of these features, we introduce a new image representation called Integral Image which permits to compute a rectangle area with only 4 elementary operations, i.e additions and subtractions.

Then, as we have a large set of features at disposition, AdaBoost is used to select a small set of them to construct a strong final classifier. We want to keep only the features which separates the best positive and negative examples. At each selection step, a weak classifier (one feature) is selected so AdaBoost provides an effective learning algorithm and strong bounds on generalization performance. Finally, the third important contribution is the cascade implementation which focuses the detection on critical regions of interest. Thus, it first eliminates quickly regions where there are no positive

examples and then, the more we go down in the cascade process, the more specific the classifiers are and so almost only faces are detected.

For example, the first stage of Viola's detector is a combination of only two features which rejects 60% of negative examples and it provides a false negative rate of 1% with only 20 simple operations.

Section 2 will describe the particularity of the features and the computation with the integral image. Then, section 3 will focus on the learning algorithm and the method in which the weak classifiers are combined to ensure a strong final classifier. Finally, section 4 will expose the cascade structure.

3.2 Features and Integral Image

This section presents the features used in our statistical face detection. Human faces are objects particularly hard to model because of there significant variety in color and texture and there are no constraints on the background. In fact, if we want to build a model which is able to take in account this face variability without identifying cluttered backgrounds, it will not work to use such as maximum likelihood methods for example. The next few subsections expose different methods used to model the faces.

3.2.1 Overview of the existing face models.

Due to the context of our face detection, methods like maximum likelihood are particularly not efficient. Will will thus focus on example based face models to train a significant classifier.Many descriptive features could be used to train a classifier by Boosting. The next subsections explain some of them that have been used recently. We can distinguish to main methods that seems to be the more efficient :

- Pixel-based models
- Haar-Like features models

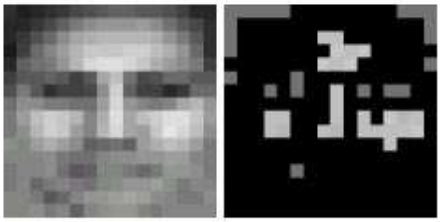
3.2.1.1 A pixel-based method

A possible way of modeling faces is to use a pixel representation as presented in Pavlovic's detection [5].

In order to train a boosted classifier as we will discuss later, Pavlovic uses a combination of weak classifiers based on the pixel values to boost the model. Let

$$h_k \in \left\{ t\left(X \mid \theta, l\right) = \text{sign}\ \left(X^{(l)} - \theta\right) \right\} \tag{3.1}$$

be a weak classifier where X denotes a vectorized image of gray-scale pixel values and $X^{(l)}$ is its l-th pixel. The weak classifier has an image as input and a decision face or non face as output, in comparison with a threshold θ. The used learning algorithm is AdaBoost which selects from the training dataset the pixels which represent the best a face structure. As you can see on the following figure, the geometrical basic features of faces are recognized: the eye region, the nose and the mouth.



(a) Average face (b) "Boosted face"

Figure 2: Points on the "boosted face" align with most descriptive features on the average face. "Boosted face" was obtained by sampling from the learned boosted threshold model.

Figure 3.1: Points on a "boosted face" align with most descriptive features on the average face. "Boosted face" was obtained by sampling from the learned boosted threshold model.

Figure 3.1 (a) shows an example of an average face obtained from the training dataset. Figure 3.1(b) shows a typical "face" image sampled from a function learned using boosting. Each non white location corresponds to a pixel selected by the boosting algorithms.

This method seems to be quite efficient because the boosting learning theoretically gives good training results but imagine that in a 19 x19 pixel image, there are some 361 pixels, we have to apply at each scanning window 361 weak classifications and combine them to obtain a final strong classifier. We will try to improve the computation time by using other face models.

3.2.2 Haar-like features

Comparing these face modeling methods and taking into account the specific needs of our application, we arrived to conclusion that a feature based method would be more appropriate rather than pixel based. There are many motivations for using features (some reminiscent of Haar Basis functions) than pixels directly as Pavlovic [5]. The most common reason is that features can act to encode ad-hoc domain knowledge that is difficult to learn using a finite quantity of training data. And as we will see, these features can operates much faster than pixel-based system.

These features are the same as those used by Papageorgiou[2]. The Haar wavelets are a natural set basis functions which computes the difference of intensity in neighbor regions. The next subsection recalls basic theory about wavelet representation.

3.2.2.1 Rectangular Haar Features

In our face detection system, very simple features are used. We use some reminiscent of Haar Basis. Recall that the wavelet function corresponding to Haar wavelet is:

$$\psi(x) = \begin{cases} 1 \;\; if\; 0 \leq x < \frac{1}{2} \\ -1 \;\; if\; \frac{1}{2} \leq x < 1 \\ 0 \;\; otherwise \end{cases} \tag{3.2}$$

There are three kinds of Haar-like features. The value of a two-rectangle feature is the difference between the sum of the pixels within two rectangular regions. The regions have the same size and shape and are horizontally or vertically adjacent.(see figure 3.2). A three-rectangle feature computes the sum within two outside rectangles subtracted from the sum in a center rectangle. Finally, a four-rectangle feature computes the difference between diagonal pairs of rectangles.

Given that the basic resolution of the detector is 15x20, the exhaustive set of rectangle features is quite large: 37525. Note that unlike the Haar basis, the set of rectangle features is over-complete.

Figure 3.3 shows the different two- three- and four- rectangles prototypes used by our detector.

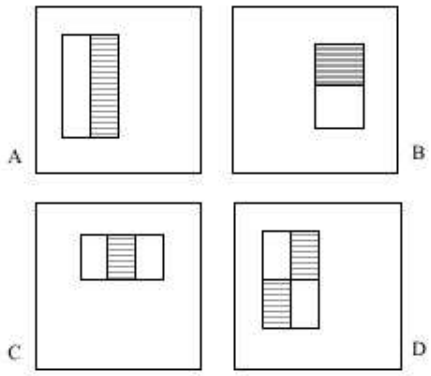


Figure 3.2: Example rectangle features shown relative to the enclosing detection window. The sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles. Two-rectangle features are shown in (A) and (B). Figure (C) shows a three-rectangle feature, and (D) a four-rectangle feature.

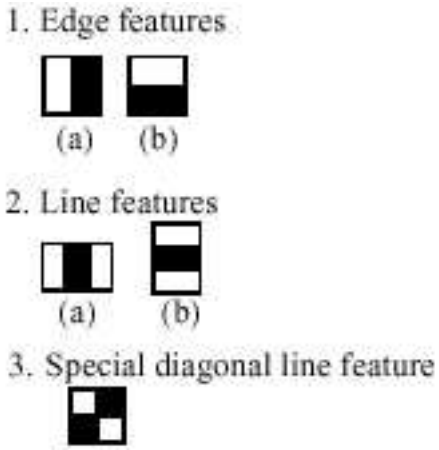


Figure 3.3: Feature prototypes of simple Haar-like . Black areas have negative and white areas positive weights.

Number of features:

The number of features derived from each prototype is quite large and differs from prototype to prototype and can be calculated as follows. Let H and W be the size of a $H \times W$ pixels window and let w and h be the size of one prototype inside the window as shown on figure 3.4.

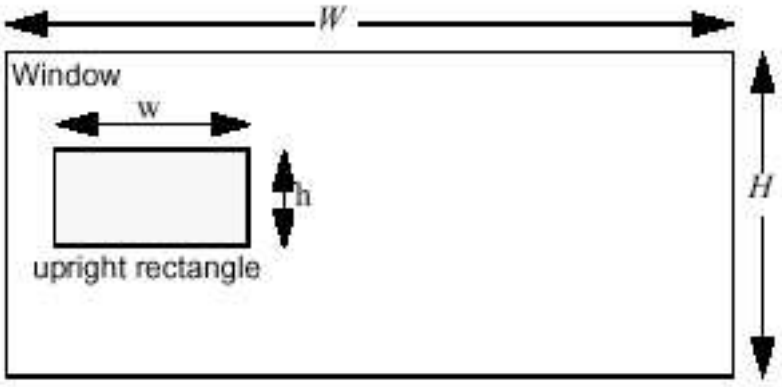


Figure 3.4: Upright rectangle in a window.

Let $X = \left\lfloor \frac{W}{w} \right\rfloor$ and $Y = \left\lfloor \frac{H}{h} \right\rfloor$ be the maximum scaling factors in x and y direction. An upright feature of size wxh then generates features for an image of size $W \times H$:

$$X \cdot Y \left(W + 1 - w\frac{X+1}{2} \right) \left(H + 1 - h\frac{Y+1}{2} \right) \tag{3.3}$$

Results with the notations of Figure 3.3:

Feature type	w/h	X/Y	count
(1a) (1b)	2/1;1/2	7/20	23520
(2a) (2b)	3/1;1/3	5/20	8400
(3)	2/2	7/10	5600
Total			37520

Table 3.2: Number of features in a 15×20 window for each prototype.

As detailed in Table 3.2 and given that the base resolution of the detector is 15×20, the exhaustive set of rectangle features is quite large: 37520. Note that unlike the Haar basis, the set of rectangle features is over-complete.

Even if our detector only uses these four types of features, we could use other types: for instance we could introduce the same rectangular features but rotated by 45 degrees as made by [7] as shown in Figure 3.5.

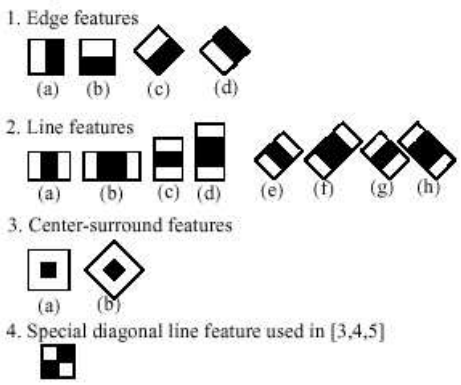


Figure 3.5: Feature prototypes of simple Haar-like and center-surround features, in line and rotated by 45 degrees. Black areas have negative and white areas positive weights.

With these other rotated features and center-surround features, the new set of features has 117,941 components in a 20x24 window.

On another side, Papageorgiou [2] introduce another kind of Haar-feature called *quadruple density transform* . This one permits to achieve the spatial resolution necessary for detection and to increase the expressive power of the model. It is nothing more than an extension of the 2D Haar wavelet as shown in Figure 3.6.

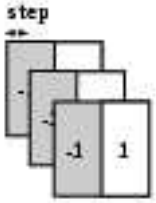


Figure 3.6: Quadruple density 2D Haar basis.

We have decided to limit our set to the simple Haar-like wavelets of Figure 3.3 because it seems to be complete enough to obtain good detection results. The choice of the feature is important but not crucial in order to train the classifiers because as explained in the next section, the training is a combination of weak classifiers. It does not really matter if the features are not optimal, and it seems that the horizontally and vertically oriented features represent better faces that rotated ones which would represent non

symmetries of faces. It is not a lack or a great loss to limit our set to basics features. We leave other types of features for a future work.

3.2.2.2 Discussion

The chosen rectangular features seem to be primitive if we compare them to other alternatives such as steerable filters [10]. Steerable filters are really well adapted to boundaries detection, image compression and texture analysis whereas the rectangle features are more sensitive to bars, the presence of edges and quite simple image structures. All the dilemma of choosing the representation resides in the compromise between the simplicity which provides fast computing and more representative filters but slower computation.

In the next subsection a new image representation will be introduced in order to improve the computing speed of these Haar-like masks responses.

3.2.3 Integral Image

We now know that we need Haar-like features to train the classifiers. The goal of this part is to introduce a new image representation called *Integral Image* which yields a fast feature computation.

This representation is in close relation with "sum area tables" as used in graphics [8].

The value of the *Integral Image* at the coordinates (x, y) is the sum of all the pixels above and to the left of (x, y), including this last point as shown in Figure 3.7.

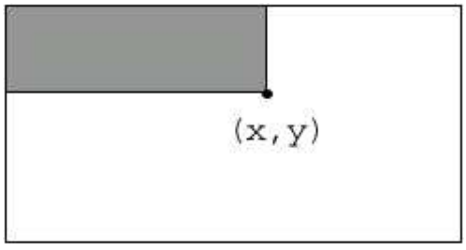


Figure 3.7: The Integral Image representation. The Integral image value at the point (x,y) is the sum of all the pixels above and to the left of (x,y).

Let ii be the integral image of the initial image i and $ii(x, y)$ the value of the integral image at the point (x, y).

We can define the integral image ii by :

$$ii(x,y) = \sum_{\substack{x' \leq x, \\ y' \leq y}} i(x',y'). \tag{3.4}$$

As we use this new representation to improve the computation time, let us explain its advantages.

First it can be computed in a efficient way using the following pair of recurrences:

$$\begin{cases} s(x,y) = s(x,y-1) + i(x,y) \\ ii(x,y) = ii(x-1,y) + s(x,y), \end{cases}$$

where $s(x,y)$ is the cumulative row sum, $\forall x,\ s(x,-1) = 0$, and $\forall y,\ ii(-1,y) = 0$. The integral image can thus be computed in one pass at the beginning of the detection over the original image i.

The main advantage using such a representation is that any rectangular sum in the original image can be computed in four array references (see Figure 3.8) in the integral image. The difference between two rectangular sums can be computed in eight references. Therefore computing a feature is only a difference of two, three or four rectangular sums.

The two rectangle features are computed with six references because the two rectangles are adjacent. The three rectangle features need eight references and the four rectangle array only nine.

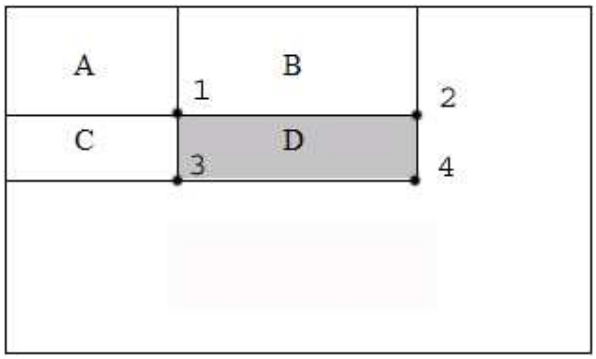


Figure 3.8: Figure 2: The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the of the pixels in rectangle A. The value at location 2 is A+B, at location 3 is A+C, and at location A+B+C+D. The sum within D can be computed as 4+1-(2+3).

There are some other reasons which made us choose the integral image representation. One of them is given by the boxlet work of Simard, et al.

[9]. It is based on a fundamental property of linear operations (e.g. $f \cdot g$ or $f \star g$). Any invertible linear operation can be applied to f or g if its inverse is applied to the result. For instance, assuming that f and g have finite support and that f^n denotes the n-th integral of f (or the n-th derivative if n is negative), we can write the following convolution identity:

$$(f * g)^n = f^n * g = f * g^n \tag{3.5}$$

where $\star$ denotes the convolution operator. They also show that the convolution can be significantly accelerated if the derivatives of f and g are sparse. From this property we can extract that for example:

$$(f'') * \left(\int \int g \right) = f * g. \tag{3.6}$$

We can apply this last formula to the rectangle sum computation: let r be the rectangle (with value 1 inside and 0 outside) and i the image, the sum in the rectangle is $i \cdot r$ and it can be computed as follow:

$$i.r = \left(\int \int i \right) \cdot r''. \tag{3.7}$$

The integral image is in fact the double integral of the image (that is why it is called integral image) and the second derivative of the rectangle yields four delta functions at the corners of a rectangle. The evaluation of the second dot product is accomplished with four array accesses.

One of the consequences of the use of such a representation is the way to scan the images.

The conventional systems computes a pyramid of images to process the detection at several scales. By using the integral image, we only need to re-scale the 20x15 pixels detector and apply it on the first integral image. No re-sampling and no image rescaling are needed that is why it provides a significant gain of time and it becomes easier to implement than using the pyramid approach.

This approach permits to compute a single feature at every location and at every scale in few operations. The power of all these independent feature is still quite weak, so the challenge of the next section is to find how the best features are selected and how we can combine them to produce a strong final classifier.

3.3 Learning with AdaBoost

3.3.1 Introduction

Considering a mono-stage classifier and given a set of features, we can build a face detector by applying all the masks at each image location (each shift and each scale). For this many different learning methods could be used .

Moreover, we have a complete set of 37520 features which is far larger than the number of pixels, so even if the features responses are very simple to compute (notably with the integral image representation), applying the all set of features would be two expensive in time. The next stage in the building of the face detector is thus to use a learning function which selects a small set of these features: the ones which separates the best positive and negative examples. The resulting final classifier would be a simple linear combination of these few Haar-like features.

For this, we will discuss in this section about an algorithm called AdaBoost (Adaptive Boosting) (see Figure 3.9) which has two main goals:

- Selecting a few set of features which represents as well as possible faces.
- Train a strong final classifier with a linear combination of these best features.

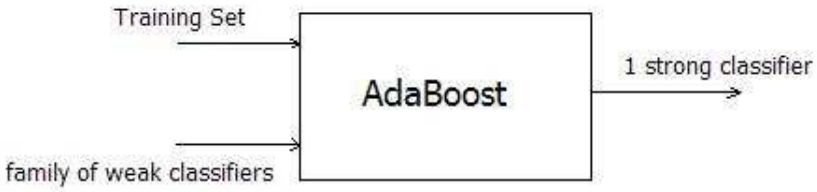


Figure 3.9: Basic scheme of AdaBoost.

In the following subsections, it is explained why we have chosen this algorithm instead of more classical ones and then we some theory is explained to show why AdaBoost is efficient and how it can be adapted to face detection.

Possible algorithms Given a set a features and a training set of positive and negative examples (see section 4.1 on page 65 for details about the training dataset), any machine learning approach could be used to learn a classification function. Here is a summary of the main possible approaches which could be used to train a classifier:

- Mixture of Gaussian model
- Simple image features and neural network
- Support Vector Machine
- Winnow learning procedure.

Why AdaBoost AdaBoost is an efficient boosting algorithm which combine simple statistical learners while reducing significantly not only the training error but also the more elusive generalization error. As all the learning functions, it presents advantages and drawbacks which are exposed here:

Advantages:

- **No a prior knowledge.** As shown in Figure 3.9, AdaBoost is an algorithm which only needs two inputs: a training dataset and a set of features (classification functions). There is no need to have any *a priori* knowledge about face structure. The most representative features will automatically be selected during the learning.
- **Adaptive algorithm**. At each stage of the learning, the positive and negative examples are tested by the current classifier. If an example x_i is misclassified, that means that it is hard to classify i.e it cannot clearly be assign in the good class. In order to increase the discriminant power of the classifier these misclassified examples are up-weighted for the next algorithm iterations. So the easily classified examples are detected in the first iterations and will have less weight in the learning of the next stages to focus on the harder examples.
- **The training error theoretically converge exponentially towards 0**. As proved by Freund and Schapire in [12], given a finite set of positive and negative examples, the training error reaches 0 in a finite number of iterations.

Drawbacks :

- **The result depends on the data and weak classifiers.** The quality of the final detection depends highly on the consistence of the training set. Both the size of the sets and the interclass variability are important factors to take in account. Other way, the types of basic classifiers which are combined have some influence on the result. The only need for all the basic functions is to be better than random selection but if we

want to achieve good detection rates in a cogent number of iterations, they have to be as well chosen as possible.

- **Quite slow training**. At each iteration step, the algorithm tests all the features on all the examples which requires a computation time directly proportional to the size of the features and examples sets. Imagine that the training set has many thousands of positive and negative examples and a complete set of 37520 features. However, the computation time is increased linearly with the size of the both sets.

3.3.2 The weak classifiers

This subsection shows how the Haar-like features can be used to build simple classifiers which need AdaBoost. The principle of the Boosting is to combine simple classifiers which are called weak learners. These weak learners are called weak because we do not expect even the best classification function to classify the data well, they only need to classify correctly the examples in more of 50% of the cases.

One easy way to link the weak learner and the Haar features is to assign one weak learner to one feature. So the AdaBoost algorithm will select at each round the feature that provides the best separation between positive and negative examples.

3.3.2.1 From features to weak classifiers

This subsection shows how to build the weak classifiers with the rectangle features. A feature response is a difference of the sum of pixels in neighbor regions. We hope that these responses then permit to distinguish positive and negative examples. For each feature and at each iteration of AdaBoost (because all the examples are re-weighted at each iterations, so the response to one feature of one example will not necessary be the same at each stage).

In other terms, one weak classifier is a feature evaluation followed by an optimal thresholding. This threshold is optimal in the sense that the minimum number of examples are misclassified.

We can summary this by the following formula:

A weak classifier $h_j(x)$ consists of a feature f_j, a threshold θ_j and a parity p_j indicating the direction of the inequality sign:

$$h_j(x) = \begin{cases} 1 & if \ \ p_j f_j(x) < p_j \theta_j \\ 0 & otherwise \end{cases} , \tag{3.8}$$

where x is an weighted example, as well positive as negative. It is weighted in the sense that all the examples are re-weighted at each stage of the algorithm.

The next subsection shows how to find the optimal threshold for each feature.

3.3.2.2 The optimal threshold

Given one feature f_j and all the examples responses $f_j(x_i)$, $i \in \textit{training set}$ to this feature, we want the threshold θ_j that separates the best positive and negative examples. One easy method would be to approximate the positive and negative distributions by two Gaussian, with only two parameters for each Gaussian. This approach would work in theory in the sense that we only want classifiers which achieve more than 50% of detection rate. But in practice the distributions have for many features a great standard deviation such that lots of examples are not characterized by the appropriate Gaussian (see Figure 3.10).

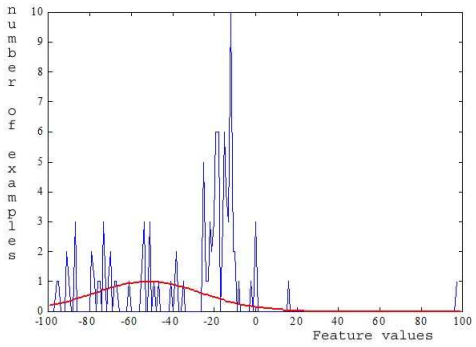


Figure 3.10: Examples distribution for a single feature for the face class and its Gaussian approximation.

A more appropriate approach is to find the threshold from the cumulative histograms.

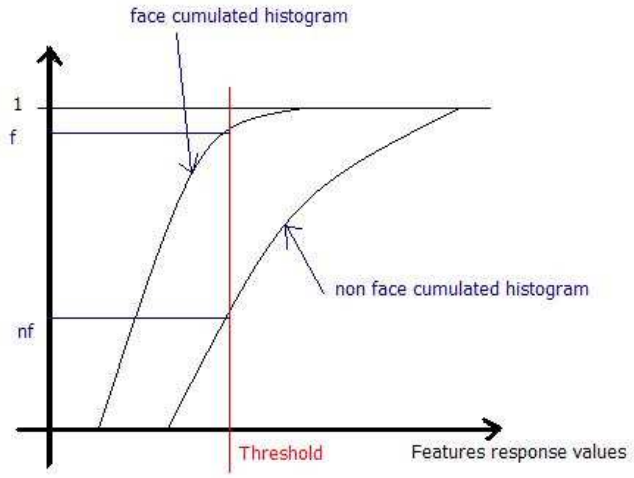


Figure 3.11: The optimal Threshold is the one that minimize the number of positive and negative misclassified examples.

Let θ be a threshold which yields f misclassified faces and nf misclassified non faces. The optimal threshold is the one that minimize $(1 - f + nf)$ in the configuration of Figure 3.11.

3.3.3 AdaBoost

3.3.3.1 Introduction

History of Boosting and AdaBoost methods The chosen learning algorithm AdaBoost is a Boosting algorithm so before explaining the use of AdaBoost in the context of face detection, basic theory about boosting will be introduced.

The Boosting theory takes its roots in the PAC learning [12]. They proved that a combination of simple learners, only better than random could yield a good final hypothesis. That is the main idea of what is called Boosting. AdaBoost (**Ada**ptive **Boost**ing) was introduced as a practical algorithm of the Boosting theory.

Let $h_1, h_2, ..., h_T$ be a set of simple hypothesis and consider the composite ensemble of hypothesis :

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x). \tag{3.9}$$

where α_t denotes the coefficient with which the ensemble member h_t is combined. Both α_t and h_t have to be learned during the boosting process.

In the beginning, Boosting algorithms were appreciated for their performances with low noise data. However, the first algorithms provided too bad results with noisy patterns due to overfitting so the applications of Boosting were limited.

On the other hand, AdaBoost can be viewed as a constraint gradient descent in an error function with respect to the margin. AdaBoost asymptotically achieves a large margin classification, that means that it concentrate its resources on a few hard-to-learn patterns that are interestingly very similar to support vectors. [13].

Trying to improve the robustness of Boosting, it was interesting to clarify the relations between Optimization Theory and Boosting procedures. From here, it became possible to define Boosting algorithms for regressions [14], multi class problems, unsupervised learning and to establish convergence proofs for boosting algorithms by using results from the Theory of Optimization.

For details about Boosting applications, publications, softwares and demonstrations, see [15].

Introduction to Boosting and Ensemble Methods In this whole section, we focus on the problem of binary classification to stay in the context of face detection with the face class and the non-face class.

The task of the binary classification is to find a rule, which, given a set of patterns, assigns an object to one of the two classes.

Let $\mathbf{X}$ be the input space which contains the objects and we denote the set of possible classes by $\mathbf{Y}$ (In our case, $\mathbf{Y}= \{-1, +1\}$). The task of learning can be summarized as follow: Estimate a function $f : \mathbf{X} \rightarrow \mathbf{Y}$, using input, output training data pairs generated independently at random from an unknown probability distribution $P(x, y)$,

$$(x_1, y_1), \cdots (x_n, y_n) \in R^d \times \{-1, +1\} \tag{3.10}$$

such that f will correctly predict unseen examples (x,y). In the case where $\mathbf{Y}=\{-1, +1\}$ we have a so-called *hard classifier* and the label assigned to an input $\mathbf{x}$ is given by $y = f(x)$.

The true performance of the classifier f is assessed by

$$L(f) = \int \lambda(f(x), y) dP(x, y), \tag{3.11}$$

where λ is a chosen loss function. The risk $L(f)$ is often called the *generalization error* in the sense that it measures the loss with respect to the example

not observes in the training set. For binary classification, we usually use the loss function $\lambda(f(x), y) = \mathbf{I}(y \cdot f(x) \leq 0)$, where $\mathbf{I}$(E)=1 if the event E occurs and 0 otherwise. In other words,

$$\lambda(f(x_i), y_i) = \begin{cases} 1, & \text{if } x_i \text{ is misclassified} \\ 0, & \text{otherwise} \end{cases} .$$

Since the probability distribution $P(x, y)$ is unknown, this risk $L(f)$ cannot be directly minimized. So we have to estimate a function as close as possible from $f_{optimal}$ based on the available information, i.e. the training examples and the properties of the function class $\mathbf{F}$ from which f is chosen. One classical solution is to approximate the generalization error by the empirical risk defined as follow:

$$\widehat{L}(f) = \frac{1}{N} \sum_{n=1}^{N} \lambda(f(x_n), y_n), \tag{3.12}$$

It is the case if the examples are uniformly distributed. If the training set is large enough, we expect that:

$$\lim_{N \to \infty} \widehat{L}(f) = L(f).$$

However, one stronger condition is required to validate the last formula: The risk error $\widehat{L}(f)$ has to converge *uniformly* over the class of functions $\mathbf{F}$ to $L(f)$.

While this condition is possible for large size training sets, for small samples size large deviations are possible and overfitting might occur. If it is the case, the generalization cannot be obtained by minimizing the training error $\widehat{L}(f)$.

As Boosting algorithms generate a complex hypothesis, one may think that the complexity of the resulting function class would increase dramatically when using an ensemble of many learners. It is the case under some conditions .

Now discuss about a strong and weak model called PAC for learning binary classifiers.

Let S be a sample consisting of N data points $\{(x_n, y_n)\}_{n=1}^{N}$,where $x_n$are generated independently at random from some distribution $P(x)$ and $y_n = f(x_n)$, f belongs to some known class F of binary functions. A strong PAC (Probably Approximately Correct) learning algorithm has the property that for every distribution P, every $f \in \mathbf{F}$ and every $\epsilon \geq 0$, $\delta \leq \frac{1}{2}$ with the probability larger than $1 - \delta$, the algorithm outputs a hypothesis h such that $Pr[h(x) \neq f(x)] \leq \epsilon$. The running time of the algorithm should be

polynomial in $1/\epsilon$, $1/\delta$, n, d, where d is the dimension (appropriately defined) of the input space. A weak PAC learning algorithm is defined without any constraints, except that it is only required to satisfy the conditions for particular ϵ and δ rather than all pairs.

Consider a combination of hypothesis as shown in 3.9. There are many approaches for selecting both the coefficients α_t and the base hypothesis h_t. In a Bagging approach, the hypothesis $\{h_t\}_{t=1}^{T}$ are chosen based on a set of T bootstrap samples, and the coefficients α_t are set to $\alpha_t = 1/T$ (see [16] for detailed Bagging approach) . The advantage of this simple method is that it tends to reduce the variance of the overall estimate $f(x)$.

The AdaBoost algorithm is a more sophisticated algorithm for Boosting the combination of the hypotheses.

It is called Adaptive in the sense that examples that are misclassified get higher weights in the next iteration, for instance the examples near the decision boundary are harder to classify and therefor get high weights in the input set after the first iterations.

The next figure illustrates AdaBoost learning on a 2-D data set.

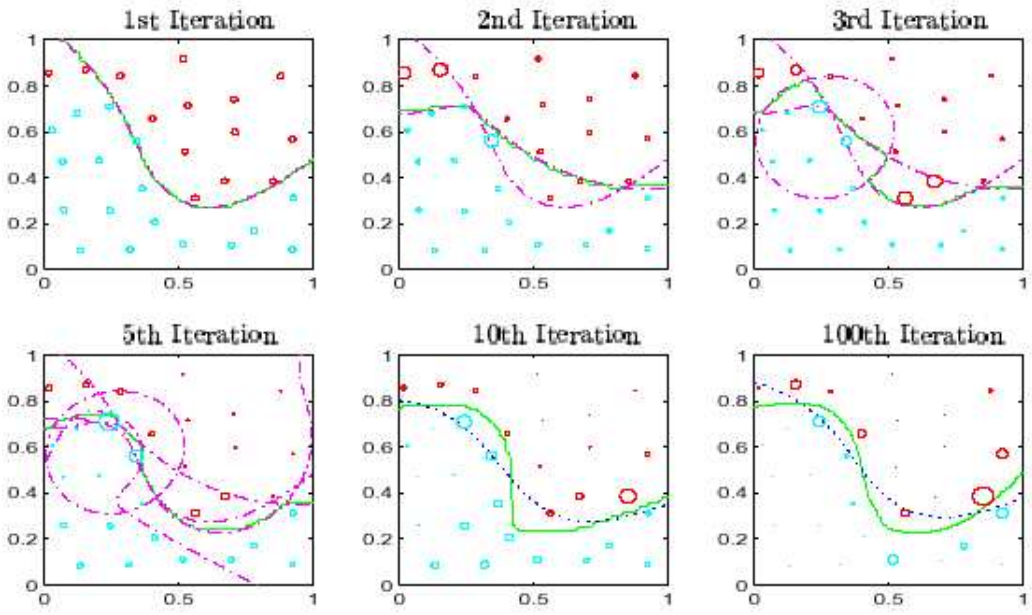


Figure 3.12: Illustration of AdaBoost on a 2D toy data set: The color indicates the label and the diameter is proportional to the weight of the examples in the first, second third, 5th, 10th and 100th iteration. The dashed lines show the decision boundaries of the single classifiers (up to the 5th iteration). The solid line shows the decision line of the combined classifier. In the last two plots the decision line of Bagging is plotted for a comparison.

AdaBoost is explained here, it will be discussed after in detail.

Algorithm 1 The AdaBoost algorithm. [17]

1. **Input:** $S = \{(\mathbf{x_1}, y_1), ..., (\mathbf{x_N}, y_N)\}$ Number of iterations T.
2. **Initialize:** $d_n^{(1)} = 1/N$ for all $n = 1, ..., N$
3. **Do for** $t = 1, ..., T$,
   (a) Train classifier with respect to the weighted sample set $\{S, \mathbf{d^{(t)}}\}$and obtain hypothesis $h_t : \mathbf{x} \longmapsto \{-1, +1\}$, i.e. $h_t = \mathrm{L}(S, \mathbf{d^{(t)}})$.
   (b) Calculate the weighted training error ε_t of h_t:
   $$\varepsilon_t = \sum_{n=1}^{N} d_n^{(t)} \mathbf{I}(y_n \neq h_t(x)_n),$$
   (c) Set :
   $$\alpha_t = \frac{1}{2} log \frac{1 - \varepsilon_t}{\varepsilon_t}.$$
   (d) Update the weights:
   $$d_n^{(t+1)} = d_n^{(t)} exp\{-\alpha_t y_n h_t(x_n)\}/Z_t,$$
   where Z_t is a normalization constant, such that $\sum_{n=1}^{N} d_n^{(t+1)} = 1$.
4. **Break if** $\varepsilon_t = 0$ or $\varepsilon_t \geq \frac{1}{2}$ and set T=t-1.
5. **Output:** $f_T(x) = \sum_{t=1}^{T} \frac{\alpha_t}{\sum_{r=1}^{T} \alpha_r} h_t(x)$

To understand this fundamental algorithm, the main steps are detailed in the following paragraphs.

3.3.3.2 AdaBoost step by step

AdaBoost is an aggressive algorithm which selects one weak classifier at each step.

A weight $\mathbf{d^{(t)}} = (d_1^{(t)}, ..., d_N^{(t)})$ is assigned to the data at step t and a weak learner h_t is constructed based on $\mathbf{d^{(t)}}$. This weight is updated at each iteration. The weight is increased for the examples which have been misclassified in the last iteration.

The weights are initialized uniformly: $d_n^{(1)} = 1/N$ for the general version of AdaBoost but we will see in subsection 3.3.3.7 on page 55 how it is modified to adapt AdaBoost to our face detection problem.

To estimate if an example is correctly or badly classified, the weak learner produces a weighted empirical error defined by:

$$\varepsilon_t(h_t, \mathbf{d^{(t)}}) = \sum_{n=1}^{N} d_n^{(t)} \mathbf{I}(y_n \neq h_t(x)_n). \tag{3.13}$$

Once the algorithm has selected the best hypothesis h_t, its weight $\alpha_t = \frac{1}{2} \log \frac{1-\varepsilon_t}{\varepsilon_t}$ is computed such that it minimizes a loss function. One of the possible loss function considered in AdaBoost is:

$$G^{AB}(\alpha) = \sum_{n=1}^{N} exp\{-y_n(\alpha h_t(x_n) + f_{t-1}(x_n))\}, \tag{3.14}$$

where f_{t-1} is the combined hypothesis of the previous iteration given by:

$$f_{t-1}(x_n) = \sum_{r=1}^{t-1} \alpha_r h_r(x_n). \tag{3.15}$$

The iteration loop is stopped if the empirical error ε_t equals 0 or $\varepsilon_t \geq \frac{1}{2}$. If $\varepsilon = 0$, the classification is optimal at this stage and so it is not necessary to add other classifiers. If $\varepsilon_t \geq \frac{1}{2}$, the classifiers does not respect the weak condition anymore. They are not better than random selection so AdaBoost cannot be efficient at all. (see 3.3.3.3)

Finally, all the weak hypotheses selected at each stage h_t are linearly combined as follow:

$$f_T(x) = \sum_{t=1}^{T} \frac{\alpha_t}{\sum_{r=1}^{T} \alpha_r} h_t(x). \tag{3.16}$$

The final classification is a simple threshold which determines if an example x_i is classified as positive or negative.

Other similar algorithms such as LogitBoost or Arcing algorithms use different loss functions.

3.3.3.3 Leverage of the weak learners

At each iteration, AdaBoost, constructs weak learners based on weighted examples. We will now discuss the performances of these weak learners based on re-weighted examples. To see the leverage of these weak learners, we need to define some basic tools.

First of all, define what is a baseline learner :

Definition 1: *Let* $\mathbf{d} = (d_1, ..., d_n)$ *be a probability weighing of the data points* S*. Let* S_+ *be the subset of the positively labeled points, and similarly for* S_-*. Set* $D_+ = \sum_{n:y_n=+1} d_n$ *and similarly for* D_-*. The baseline classifier* f_{BL} *is defined as :*

$$f_{BL}(x) = sign(D_+ - D_-), \tag{3.17}$$

for all x*. So it predicts +1 if* $D_+ \geq D_-$ *and -1 otherwise. It is immediately obvious that for any weighting* $\boldsymbol{d}$*, the error of the baseline is at most 1/2.*

With this definition, we can define more properly what is a weak learner: A learner is a weak learner for example S if given any weighting $\mathbf{d}$ on S, it is able to achieve a weighted classification error which is strictly smaller than $1/2$.

A key ingredient of Boosting algorithms is a weak learner which is required to exit in order for the overall algorithm to perform well. We demand that the weighted empirical error of each weak learner is strictly smaller than $\frac{1}{2} - \frac{1}{2}\gamma$, where γ is an edge parameter quantifying the deviation of the performance of the weak learner from the baseline classifier introduced before. Consider a weak learner that outputs a binary classifier h based on a data set $S = \{(x_n, y_n)\}_{n=1}^N$ each pair (x_n, y_n) of which is weighted by a non-negative weight d_n. We then demand that

$$\varepsilon_t(h_t, \mathbf{d}) = \sum_{n=1}^{N} d_n \mathbf{I}(y_n \neq h(x)_n) \leq \frac{1}{2} - \frac{1}{2}\gamma, \qquad (\gamma > 0). \tag{3.18}$$

For some simple weak learners, it may not exist $\gamma > 0$ which respects the previous condition without imposing some conditions about the data.

Now consider one situation for which it is possible to find a positive γ which respects the empirical error condition in 3.18. Consider a mapping f from the binary cube $\mathbf{X} = \{+1, -1\}^d$ to $\mathbf{Y} = \{+1, -1\}$, and assume the true labels y_n are given by $y_n = f(x_n)$. We wish to approximate f by combinations of binary hypotheses h_t belonging to $\mathbf{H}$. Let H be a class of binary hypotheses, and let D be a distribution over $\mathbf{X}$. The correlation between f and H, with respect to D, is given by

$C_{H,D}(f) = sup_{h \in H} \mathbf{E}_D\{f(x)h(x)\}$. The distribution-free correlation between f and H is given by $C_H(f) = inf_D C_{H,D}(f)$. It can be shown that if $T > 2log(2)dC_H(f)^{-2}$ then f can be represented exactly as

$$f(x) = sign(\sum_{t=1}^{T} h_t(x)). \tag{3.19}$$

In other words, if H is highly correlated with the target function f, then f can be exactly represented as a combination of a small number of functions from H. Hence after a sufficiently large number of iterations, the empirical error can be expected to approach zero.

This example shows a binary classification with to separate classes. In general situations, the data may be strongly overlapping, so more advanced tools need to be found to establish such rules about the weak learners.

3.3.3.4 Convergence of the Training Error to Zero

We have seen just before that under some appropriate conditions, that the weighted empirical error could be smaller than $\frac{1}{2} - \frac{1}{2}\gamma$, $(\gamma > 0)$. We will now explain how this condition can imply a strong and fundamental result for AdaBoost (it can be generalized to most of Boosting algorithms): The condition $\varepsilon_t(h_t, \mathbf{d}) \leq \frac{1}{2} - \frac{1}{2}\gamma$, $(\gamma > 0)$ is sufficient to ensure that the empirical error of the strong final hypothesis approaches zero as the number of iterations increases. The proof of this important property of AdaBoost is given in this paragraph.

Let f be a real-valued classification function. The classification is performed using $sign(f)$ but we will work with the actual value of f.

Let $y \in \{-1, +1\}$ be the labels of the binary classification and $f \in R$, we define the margin of f at the example (x_n, y_n) as :

$$\rho_n(f) = y_n f(x_n). \tag{3.20}$$

Consider the following function defined for $0 \leq \theta \leq 1/2$,

$$\varphi_\theta(z) = \begin{cases} 1 & \text{if } z \leq 0 \\ 1 - z/\theta & \text{if } 0 < z \leq \theta \\ 0 & \text{otherwise.} \end{cases} \tag{3.21}$$

Let f be a real-valued function taken values in $[-1, +1]$. The *empirical margin error* is defined as :

$$\widehat{L}^\theta(f) = \frac{1}{N} \sum_{n=1}^{N} \varphi_\theta(f(x_n), y_n). \tag{3.22}$$

If it is obvious from the definition that the classification error, namely the fraction of misclassified examples, is given by $\theta = 0$, i.e. $\widehat{L}(f) = \widehat{L}^0(f)$. In addition, $\widehat{L}^\theta(f)$ is monotonically increasing in θ. We not that we often use the so-called 0/1-margin error defined by:

$$\widetilde{L}^\theta(f) = \frac{1}{N}\sum_{n=1}^{N} \mathbf{I}(f(x_n), y_n) \leq \theta.$$

Noting that $\varphi_\theta(yf(x)) \leq \mathbf{I}(yf(x) \leq \theta)$, it follows that $\widehat{L}^\theta(f) \leq \widetilde{L}^\theta(f)$.

Theorem 1. *Consider AdaBoost as described in Algorithm 1. Assume that at each round, the weighted empirical error satisfies* $\varepsilon_t(h_t, \mathbf{d}) \leq \frac{1}{2} - \frac{1}{2}\gamma_t,\ (\gamma_t > 0)$. *Then the empirical margin error of the composite hypothesis* f_T *obeys*

$$\widehat{L}^\theta(f_T) \leq \prod_{t=1}^{T}(1-\gamma_t)^{\frac{1-\theta}{2}}(1+\gamma_t)^{\frac{1+\theta}{2}}. \tag{3.23}$$

Proof. We present a proof from [18] for the case where $h_t \in \{-1, +1\}$. We begin by showing that for *every*$\{\alpha_t\}$

$$\widetilde{L}^\theta(f_T) \leq exp(\theta\sum_{t=1}^{T}\alpha_t)(\prod_{t=1}^{T} Z_t). \tag{3.24}$$

Where by definition,

$$\begin{aligned} Z_t &= \sum_{n=1}^{N} d_n^{(t)} e^{-y_n\alpha_t h_t(x_n)} \\ &= \sum_{n:y_n=h_t(x_n)} d_n^{(t)} e^{-\alpha_t} + \sum_{n:y_n\neq h_t(x_n)} d_n^{(t)} e^{+\alpha_t} \\ &= (1-\varepsilon_t)e^{-\alpha_t} + \varepsilon_t e^{\alpha_t}. \end{aligned}$$

From the definition of f_T it follows that

$$yf_T(x) \leq \theta \ \Rightarrow\ exp\,(-y\sum_{t=1}^{T}\alpha_t h_t(x)\theta\sum_{t=1}^{T}\alpha_t) \geq 1,$$

which we rewrite as

$$\mathbf{I}[Y\,f_T(X) \leq \theta] \leq exp\,(-y\sum_{t=1}^{T}\alpha_t h_t(x)\theta\sum_{t=1}^{T}\alpha_t). \tag{3.25}$$

Note that

$$\begin{aligned} d_n^{(T+1)} &= \frac{d_n^{(t)} exp(-\alpha_T y_n h_T(x_n))}{Z_T} \\ &= \frac{exp(-\sum_{t=1}^{T} \alpha_T y_n h_t(x_n))}{N \prod_{t=1}^{T} Z_t} \ (by\, induction) \end{aligned} \tag{3.26}$$

Using 3.25 and 3.26 we find that

$$\begin{aligned} \tilde{L}^{\theta}(f) &= \frac{1}{N} \sum_{n=1}^{N} I[y_n f_T(x_n) \leq \theta] \\ &\leq \frac{1}{N} \sum_{n=1}^{N} [\exp(-y_n \sum_{t=1}^{T} \alpha_t h_t(x_n) + \theta \sum_{t=1}^{T} \alpha_t)] \\ &= \frac{1}{N} \exp(\theta \sum_{t=1}^{T} \alpha_t) \sum_{n=1}^{N} \exp(-y_n \sum_{t=1}^{T} \alpha_t h_t(x_n)) \\ &= \exp(\theta \sum_{t=1}^{T} \alpha_t)(\prod_{t=1}^{T} Z_t) \sum_{n=1}^{N} d_n^{(T+1)} \\ &= \exp(\theta \sum_{t=1}^{T} \alpha_t)(\prod_{t=1}^{T} Z_t). \end{aligned}$$

Next, set $\alpha_t = (\frac{1}{2}) \log((1-\varepsilon_t)/\varepsilon_t)$ as in algorithm 1, which easily implies that

$$Z_t = 2\sqrt{(1-\varepsilon_t)\varepsilon_t}.$$

Substituting this result into 3.24 we find that

$$\tilde{L}^{\theta}(f) \leq \prod_{t=1}^{T} \sqrt{4\varepsilon_t^{1-\theta}(1-\varepsilon_t)^{1+\theta}}$$

which yields the desired result result upon recalling that $\varepsilon_t = 1/2 - \gamma_t/2$, and noting that $\hat{L}^{\theta}(f) \leq \tilde{L}^{\theta}(f)$. In the case that $\theta = 0$, i.e. one considers the training error, we find that

$$\hat{L}(f_T) \leq e^{-\sum_{t=1}^{T} \gamma_t^2/2}, \tag{3.27}$$

from which we infer that the condition $\sum_{t=1}^{T} \gamma_t^2 \to \infty$ suffices to guarantee that $\hat{L}(f_T) \to 0$. For example, the condition $\gamma_t \geq c/\sqrt{t}$. Clearly this holds if $\gamma_t \geq \gamma_0 > 0$ for some positive constant γ_0. In fact, in this case $\hat{L}^{\theta}(f_T) \to 0$ for any $\theta \leq \gamma_0/2$.

If each weak classifier is slightly better than random so that γ_t is bounded away from zero, then the training error drops exponentially fast. This bound, combined with the bounds on generalization error given below prove that AdaBoost is indeed a boosting algorithm in the sense that it can efficiently convert a weak learning algorithm into a strong learning algorithm.

We have establish the proof of the convergence to zero of the training error. This property is essential for the performances of AdaBoost but not enough. What is even more important is the evolution of the generalization error.

3.3.3.5 Generalization Error Bounds

We know that the training error produced by AdaBoost approaches exponentially zero as the number of iterations increases. However, as the examples of the training set are manually labeled, it is not really interesting to know that these examples are well classified. It will be wiser to see how efficient the final model is on other dataset which haven't been used for the training. The error committed on this new dataset (with positive and negative examples) is called the generalization error and it will be shown in this paragraph how it can be bounded.

Recalling that AdaBoost, such as all learning algorithms can be viewed as a procedure for mapping any data set $S = \{(x_n, y_n)\}_{n=1}^{N}$to some hypothesis h belonging to a hypothesis class H consisting to functions from $\mathbf{X}$ to $\{-1, +1\}$. We want to test the performance of the hypothesis $\hat{f}$ on future data, considering that $\hat{f}$ and S are random variables. Let $\lambda(y, f(x))$ be a loss function which measures the loss caused by using the hypothesis f to classify input x, the true label of which is y. The loss expected is given by $L(h) = \mathbf{E}\{\lambda(y, f(x))\}$, where the expectation is taken with respect to the unknown probability distribution generating the pairs (x_n, y_n).We will use the following loss function :

$$\lambda(y, f(x)) = \mathbf{I}[y \neq f(x)], \tag{3.28}$$

where $\mathbf{I}[\varepsilon] = \begin{cases} 1 & \text{if event } \varepsilon \text{ occurs} \\ 0 & \text{otherwise} \end{cases}$. In this case: $L(f) = Pr[y \neq h(x)]$, the probability of misclassification.

Vapnik [19] proved a classical result about the empirical classification of binary hypothesis f, to the probability of error $Pr[y \neq f(x)]$. Let $P(y \neq f(x)$ be the classification error probability and $\hat{P}(y \neq f(x))$ the empirical classification error.

First of all, we need to define the VC-dimension of a class of binary hypotheses F.

Definition : *Consider a set of N points* $X = (x_1, ..., x_N)$ *. Each binary function* f *defines a dichotomy* $(f(x_1), ..., f(x_N)) \in \{-1, +1\}^N$ *on these points. Allowing* f *to run over all elements of* f*, we generate a subset of the binary N-dimensional cube* $\{-1, +1\}^N$*, denoted by* F_X*, where* $F_X = \{(f(x_1), ..., f(x_N)) : f \in F\}$ *. The VC-dimension of F, noted VC-dim(F), is defined as the maximal cardinality of the set of points* $X = (x_1, ..., x_N)$ *for which* $|F_X| = 2^N$*.*

Good estimates of the VC dimensional are available for many classes of hypotheses.

We can now use a theorem from [21] to introduce a generalization error bound:

Theorem : *Let F be a class of* $\{-1, +1\}$*-valued functions defined over a set* $\boldsymbol{X}$*. Let P be a probability distribution on* $\boldsymbol{X} \times \{-1, +1\}$*,and suppose that N-samples* $S = \{(x_1, y_1), ..., (x_N, y_N)\}$*are generated independently at random according to P. Then there is an absolute constant c, such that for any integer N, with probability at least* $1 - \delta$ *over samples of length N, every* $f \in F$ *satisfies*

$$P(y \neq f(x)) \leq \hat{P}(y \neq f(x)) + c \cdot \sqrt{\frac{VCdim(F) + log(1/\delta)}{N}}. \tag{3.29}$$

While structural risk minimization is a mathematically sound method, the upper bounds on $L(f)$ that are generated in this way might be larger the actual value and so the chosen number of iterations T might be much smaller than the optimal value, leading to inferior performance. A simple alternative is to use cross validation in which a fraction of the training set is left outside the set used to generate $f_T(x)$. The value of T is chosen to be one for which the error of the final hypothesis on the validation is minimized.

3.3.3.6 A few comments

It would be interesting to notice that the final hypothesis $f_T(x)$ is closely related to the Bayesian theory.

Let (x, y) be the examples generated according to the distribution P on $\mathbf{X} \times \{\mathbf{0}, \mathbf{1}\}$. Suppose we are given a set of $\{0, 1\}$-valued hypotheses $h_1 ... h_T$ and that our goal is to combine the predictions of these hypotheses as good as possible. Then, given an example x and the hypothesis prediction $h_t(x)$, the Bayes optimal decision rule says that we should predict the label with the highest likelihood, given the hypothesis values, i.E. we should predict :

$$\begin{cases} 1 & \text{if } Pr[y = 1|h_1(x), ..., h_T(x)] > Pr[y = 0|h_1(x), ...h_T(x)] \\ & 0 \quad \text{otherwise} \end{cases}.$$

Now assume that the errors of the all hypotheses are independent of one another and of the target concept, in other words, we assume that $h_t(x) \neq y$ is conditionally independent of y and the predictions of all other hypotheses $h_1, ..., h_{t-1}, h_{t+1}, ..., h_T$. With these assumptions, we can rewrite the Bayesian optimal decision as the prediction of :

$$\begin{cases} 1 & \text{if } Pr[y=1] \prod_{t:h_t(x)=0} \epsilon_t \prod_{t:h_t(x)=1} (1-\epsilon_t) > Pr[y=0] \prod_{t:h_t(x)=1} \epsilon_t \prod_{t:h_t(x)=0} (1-\epsilon_t) \\ 0 & \text{otherwise} \end{cases}, \tag{3.30}$$

where $\epsilon_t = Pr[h_t(x) \neq y]$.

Assuming that h_0 always predicts 1, $Pr[y=0] = \epsilon_0$. Taking the logarithm on both sides in 3.30 and rearranging the terms, we find that the Bayes optimal decision is the same as the final rule generated by AdaBoost.

3.3.3.7 Adaptation to face detection

The algorithm presented in the previous paragraph is not specific to face detection. This new subsection will explain how the algorithm can be adapted to our face detection context, particularly with the introduction of an asymmetric classification.

AdaBoost, as described in 1, is a an algorithm which minimize the classification error (or generalization error) but it does not minimize the number of false negative as explained in 3.1.2.

There are several methods to modify AdaBoost in order to obtain an asymmetric algorithm, asymmetric in the sense that we want to increase the influence of the positive examples which have been misclassified earlier in the precess in order to minimize the false negative rate, i.e. the rate of the faces which are missed.

One first simple mean would be to unbalance the initial distribution of the positive and negative examples as in [20]. If we want to minimize the false negatives, we can increase the weight on positive examples so that the minimum error criteria will also have very low false negatives.

This idea can be introduced by changing the loss function in a non symmetric loss function.

Recall that the classical AdaBoost minimizes

$$\prod_t Z_t = \sum_i \exp(-y_i \sum_t h_t(x_i)). \tag{3.31}$$

Each term in the summation is bounded above by the loss function from 3.28:

$$\exp(-y_i \sum_t h_t(x_i)) \geq \lambda_i(y_i, f(x_i)) = \mathbf{I}[y_i \neq f(x_i)], \tag{3.32}$$

where λ is the loss function. It follows that minimizing $\prod_t Z_t$ minimizes an upper bound on simple loss.

So we can introduce the asymmetric loss defined by:

$$A\lambda_i = \begin{cases} \sqrt{k} & \text{if } y_i = 1 \text{ and } f(x_i) = -1 \\ \frac{1}{\sqrt{k}} & \text{if } y_i = -1 \text{ and } f(x_i) = 1, \\ 0 & \text{otherwise} \end{cases} \tag{3.33}$$

where false negative cost k times more than false positives. Note that $A\lambda_i = \exp(y_i \log \sqrt{k})\lambda_i$.We can easily obtain the bounds of the asymmetric loss by combining 3.32 and 3.33. If take 3.32 and multiply both sides by $\exp(y_i log\sqrt{k})$ we find :

$$\exp(-y_i \sum_t h_t(x_i)) \cdot \exp(y_i \log \sqrt{k}) \geq A\lambda_i \tag{3.34}$$

In order to minimize this bound, we can use a non-uniform weight initialization:

Modify 2 in AdaBoost algorithm 1 by

$$d_n^{(1)} = \exp(y_i \log(\sqrt{k}))/N.$$

Updating the weights will become:

$$D_{t+1}(i) = \frac{\exp(-y_i \sum_t h_t(x_i)) \cdot \exp(y_i \log(\sqrt{k}))}{\prod_t Z_t} \tag{3.35}$$

The modification of the pre-weighting is transmitted through the second term of the numerator.

This new weighting process permits to reduce efficiently the false negative rate as shown in Figure 3.13.

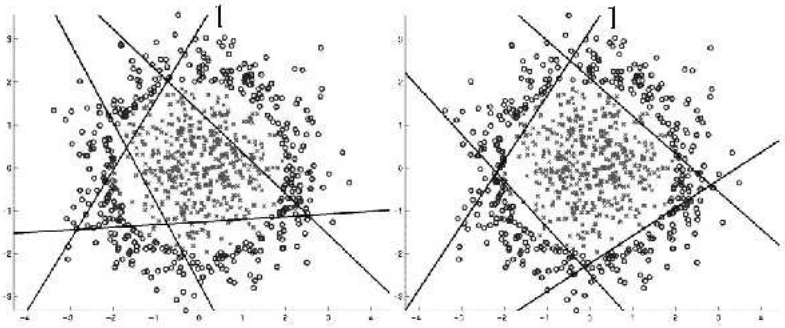


Figure 3.13: Two simple examples: positive examples are x, negative o and weak classifiers are linear separators. On the left is the naive asymmetric result. The first feature selected is labeled 1. Subsequent features attempt to balance positive and negative errors. Notice that no linear combination of the 4 weak classifiers can achieve a low false positive and low false negative rate. On the right is the asymmetric boosting result. After learning 4 weak classifier the positives are well modeled and most of the negative are rejected.

However, the effects of the unbalanced weights are lost after the first iteration. In fact, the AdaBoost algorithm seems to be too greedy. The first classifier absorbs the effects of the asymmetric weights.

3.3.4 discussion

AdaBoost selects thus a small set of features and as detailed in section 4.2, good results can be obtained with some 200 features. The first features selected in the process can be quite easily interpreted: they emphasize on particular features of faces. The eye region is often darken than the front and the nose bridge is brighter than the eyes.

However, this is not enough to reach the fixed goal of our project. The computation time for a 200 features classifier is to large to satisfy us. We will introduce a way to combine classifiers in cascade in order to focus quickly on the regions of interest.

3.4 Classification in Cascade

We know how to select a small number of critical features and to combine them into a strong classifier. However, we need to introduce a new main contribution in our face detection system in order to reduce significantly the computation time. This contribution is an attentional cascade which can

furthermore achieve better detection performances. We have seen that it is possible to minimize the number of false negatives instead of classical training error, that is precisely the main idea that will be used to build this cascade classifier.

3.4.1 Why is it so efficient?

The principle is to reject quickly the majority of negative windows while keeping almost all positive examples and then focus on more sensitive sub-windows with more complete classifiers. To do that, the first stages in the cascade will contain only few features, which achieve very high detection rates (about 100 %) but will have a false positive rate of roughly 40 %. It is clear that it is not acceptable for a face detection task but combining succecively many of these stages which are more and more discriminant will permit to reach the goal of fast face detection.

We can just compare this cascade structure with a degenerated decision tree. If a sub-window is classified as positive at one stage, it proceeds down in the cascade and will be evaluated by the next stage. It will be like this until this sub-window is found negative by one stage or if all the stages classify it as positive. In this last case, it will finally be considered as a positive example. The Figure 3.14 shows this cascade process.

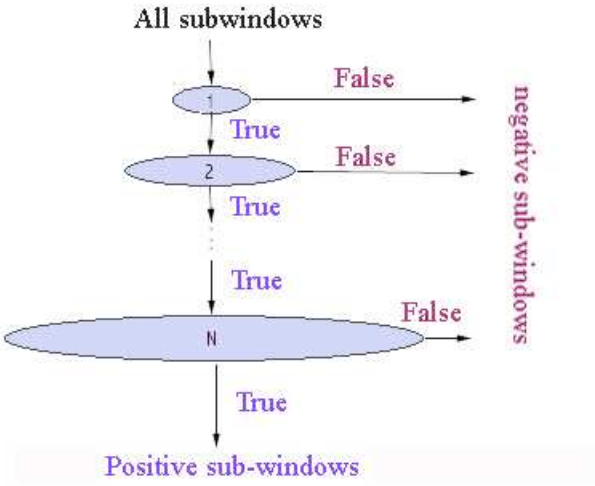


Figure 3.14: Schematic description of cascade detection. A series of classifiers is applied to every sub-window. The initial classifier eliminates a large number of negative examples with very little processing. Subsequent layers eliminate additional negatives but require additional computation. After several stages of processing, the number of sub-windows have been reduced radically. The examples classified as positive by the last stage are definitively considered as positive.

As explained in a context of the face detection in 3.1.2, the goal of the project is to detect faces in images which contain few faces. Noticing that there are about 20.000.000 windows in a 100×100 image for only a few faces, the great majority of windows are negative ones. So it is a real gain of time to reduce quickly this number. Even if the last stages of the cascade are based on many thousand features, they will be called only for few sub-windows.

3.4.2 Building more consistent classifiers

We have defined the cascade as a succession of classifiers. The first ones are quite simple but as we progress in the cascade, the classifiers have to be more consistent. This paragraph describes how such more consistent classifiers can be built at each stage.

First of all, the last stages of the cascade have more features than the first ones. The AdaBoost algorithm generates a training error which decreases theoretically exponentially with the number of iteration (see 3.3.3.4). If there are more features (i.e. AdaBoost has been ran with more iterations) the final classifier is more discriminant between positive and negative examples,

in other words, we can say that such classifier are "stronger" than classifiers with few features (i.e. few iterations).

The second but not less important reason for using a cascade classification is the way chosen to select the training set. At each learning step, the classifier or the i-th stage, so-called i-th classifier is tested on a test set of negative examples. All the misclassified examples are kept for the $(i+1)$-th classifier such that the $(i+1)$-th classifier will focus on harder examples than ordinary ones. By this mean, we force the further classifiers to have better false positive rate.

3.4.3 Training a cascade of classifiers

The goal of the cascade detection is to achieve given both false positive rate and detection rate. The choice of these goals is arbitrary. Typically, past systems have achieved detection rates between 85 and 95 percent and false positive rate on the order 10^{-5}. The number of features in each stage and the total number of stages will depend of these constraints.

Let F be the false positive rate of the cascaded classifiers, K the number of classifiers and f_i the false positive rate of the *i-th* classifier on examples that get through to it. For a given trained cascade of classifiers, F is given by:

$$F = \prod_{i=1}^{K} f_i, \tag{3.36}$$

Then the detection rate can be computed as:

$$D = \prod_{i=1}^{K} d_i, \tag{3.37}$$

where d_i is the detection rate of the *i-th* classifier on the examples that get through to it.

To fix the ideas, a examples is given here. If we want to achieve a detection rate of 90 percent, we can build a 10 stage classifier in which each stage has a detection rate of 0.99. Indeed, $0.9 \simeq 0.99^{10}$. If each of these stage rejects 70 percent of negatives (i.e. a false positive rate of 30 percent), the total false positive rate is $0.30^{10} \simeq 6.10^{-6}$.

The number of features evaluated when scanning real images is necessary a probabilistic process. Any window will progress down through the cascade, one classifier at a time, until it is decided that the window is negative or, in really rare cases, the window succeeds in each test and is labeled positive. The behavior of this process is determined by the distribution of the images

of the test set. The main tool which can measure the performance of a given classifier is its positive rate, which is the proportion of windows which are labeled as potentially containing the object of interest. Given a number of stages in the cascade K, the positive rate p_i of the i-th classifier. Let n_i be the number of features in the i-th stage. The expected number of features which are evaluated is given by:

$$N = n_0 + \sum_{i=1}^{K}(n_i \prod_{j<i} p_i) \tag{3.38}$$

We can notice only few examples are objects, that is why the positive rate is almost equal to the false positive rate.

As it was explained in section 3.3.3.7, the original AdaBoost algorithm has to be modify to ensure the minimization of the false negative rate instead of the training error. One simple way to impose that is to adjust the final threshold. Increasing this threshold will affect badly the detection rate and improve the false positive rate, while the opposite will yield lower detection rate with higher false positive rate. The main problem is that is has never been proved that modifying the AdaBoost theoretical training threshold preserve the guarantees in term of generalization error.

The cascade structure has three main parameters that we have to determine :

- The total number of classifiers : K
- The number of features n_i of each stage i
- The threshold θ_i of each stage i.

Finding the three optimal parameters is quite complicated if we keep in mind that we want to minimize the computation time of the total classification. The principle is to increase the number of features and of stages until the given detection objective are reached.

Given the minimum acceptable rates f_i (false positive rate for the $i-th$ stage) and d_i (detection rate for the $i-th$ stage), the detection rate d_i is reached by decreasing the AdaBoost threshold $\theta_i$and this also directly affects f_i. We increase the number of features n_i in the $i-th$ stage until f_i is obtained. The general principle of the cascade learning is given in the algorithm 2 :

One major factor for the efficiency of the cascade learning is the management of the sets during the training. Usual training sets are used for the first stage, and then, at each iteration, the current stage classifier is evaluated on

a validation set in order to minimize coherent false positive and false negative rates. It would obviously unskilled to evaluate F_i and D_i on the training set which was used to obtain the model because these values would be evaluated much better than with other examples. Then, at each stage we reinitialize the negative training set. Once the objectives F_i and D_i are reached for the stage i, the current model is tested on a large negative set chosen randomly and many false positive alarms are introduced into the negative training set for the stage $i-1$. As the new negative training set is made with examples which have been misclassified by the stage i, the stage $i+1$ will be build with examples which can be considered as "hard examples". So, the more we go down into the cascade, the more critical the examples are, and the last stages in the cascade are more robust models which discriminate better positives and negatives than the first stages. Since the large majority of negative windows have been rejected by the first stages, even if the further stages need more computation time to classify the windows, only few windows have passed earlier stages and the global computation time is not too much affected by these last discriminant stages.

3.5 Scanning

As explained with the definition of the integral image in 3.2.3, the scanning of an input image is quite simple and efficient with the integral image representation. To detect faces of different sizes and places in a image, we will apply scaled and shifted detectors all over the image. Our basic detector is a 20x15. All the images used to train the model, as well faces as non faces are of this size, and accordingly, all the selected rectangular features that we have to apply in the windows are defined in this 20x15 basic window.

Although the scanning process seems to be simple at the first sight we need to take some care while rescaling the detector if we want to preserve the efficiency of the model. See A.1 for details about the implementation issues about the scanning window.

Once all the possible windows have been scanned all over the image, we have to integrate a process which clusters the multi-detections of a single face in order to have finally one bounding box around one face. It is clear that the shifting of the window at different scales using a small shift step and a small scaling steps permits to detect all the faces of any size and position. However, one face may be detected several times during the scanning process (by neighbor windows or very close scales at the same position). The chosen method to cluster these multiple detections is to cluster all the positive windows which are close enough. Then the center of the resulting bounding

box is simply the gravity center of all the centers in the cluster and the size is the mean of all the sizes.

Let $\{c_1, c_2, ..., c_n\}$ be the centers of the windows in a cluster containing n windows and $\{w_1, w_2, ..., w_n\}$ their respective width (The height of the window is directly given and preserved by the constant ratio $r = \frac{h}{w}$). The center of the cluster is given by: $c = \frac{1}{n} \sum_{i=1}^{n} c_i$ and the final width is simply $w = \frac{1}{n} \sum_{i=1}^{n} w_i$.

Algorithm 2 Learning in cascade.

1. **Input:** Definition of the targets of the learning
   - f the maximum acceptable false positive rate per stage.
   - d the minimum acceptable detection rate per layer
   - F_{target} the false positive rate desired at the end of the process.
   - P the set of positive examples
   - N the set of negative examples
2. **Initialization:**
   - $F_0 = D_0 = 1$
   - $i = 0$ the number of the current layer.
3. **Main loop:**
   - While $F_i > F_{target}$
     - $i \leftarrow i + 1$
     - $n_i = 0$
     - $F_i = F_{i-1}$
     - while $F_i > f \times F_{i-1}$
       - $n_i \leftarrow n_i + 1$
       - Train a classifier using AdaBoost (Algorithm 1 at page 47) with P and N as training set.
       - Compute F_i and D_i for the current classifier with the validation set.
       - Decrease the threshold of the classifier until the detection rate for the i-th classifier is at least $d \times D_{i-1}$: $D_i \geq D_{i-1}.d$
     - Empty the negative training set.
     - If $F_i > F_{target}$, evaluate the current cascade classifier on a set of negative examples and put any false detections into the set N.

Chapter 4

Experiments and Results

This Section exposes the different results obtained by the face detector that has been developed. We will discuss our choices, try to interpret some results such as the power of the selected features and finally estimate the global performances of the system. Of course, all these results are obtained in specific conditions and we will particularly pay attention to explain these testing conditions.

The first step we will focus on is the choice of the datasets because it influences a lot the quality of the learning (and the evaluation of the results). Then, an important step is the result of the learning algorithm, how does AdaBoost perform? what kind of features are selected? what are the different training rates which can fix the quality of the learning? Then in which way the cascade implementation improves the face detection will be explained. The final section relates about the performances of the final detector tested on a particular Testing Set.

4.1 Datasets

The Datasets represent all the images that we use for our face detection task. It is really important to notice that the choice of the datasets is crucial for the learning and the tests on the detectors. We can separate the Datasets as follows:

- Learning Data
- Testing Data

Learning Data : It clusters all the examples that have been used to train and test the different classifiers. There are some positive and negative ex-

amples. On one side there are positive examples which are faces extracted from different sources: Banca database (see [39]), the BioID images (can be found in [38]) and XM2VTS [40]. The faces are thus pictures from different acquisition conditions and lightning conditions. Concerning the Negative examples (non faces), they have to represent the best the backgrounds that can be found in real situations. Thus we just extract them randomly from the web in images without faces. It is hard to know a priori which images are the most representative of the non face class and the number of non faces that we need to train the classifier. However a bootstrap method will select non face images that are the hardest to classify and so to find more precisely the boundary between the face and non face classes.

The images from the various Databases are of different sizes and the faces are more or less cropped while our learning set has to be homogeneous in term of size and face repartition . As most of the faces are higher than larger, we have chosen a rectangular window of 20x15 pixels. All the faces have been cropped and rescaled if necessary in order to respect this basic detector size. (We notice that the examples are effectively low resolution ones as imposed in the context of the detection.)

- **Training Set** : It is the input of AdaBoost for the mono-stage classifier (to train some 500 features) and for the first sage of a learning in cascade. Recall that one of the limitations of AdaBoost is the large influence of the input data on the boosting results, the choice of the training set needs particular careful. The use of diverse databases is well adapted because images from a single database are often taken from from similar conditions. For example, faces of the BANCA database have a lot of variety in the sense that people do not always look exactly at the camera but the lightning conditions are quite troublesome because the light often comes from one side of the face. Regrouping the different sources, we have 8257 faces and more than 300.000 non faces.

- **Testing Set :** Once a classifier has been trained using AdaBoost, we have to train it on another set of images (both positive and negative images).Thus we can obtain the test error of the classifier.

  In the case of the cascade, this set is used during the learning to test the current cascade. The misclassified examples become the train examples of the next stage. Thus each stage is directly adapted to the efficiency of the previous ones, in the sense that it is trained from the examples that have been badly classified by the previous stages.

  For the mono-stage classifier, we have 60.000 non faces built by an intermediate detector and 8257 faces. The examples used by the cascade

are the same as the ones used in the training set.

- **Validation Set:** This set is used to test the performances of the cascade: Some images are presented to the final cascade detector and it permits to evaluate the global quality of the detector.

Testing Data :

- Images from the CMU: They represent many real situations with several faces and unconstrained background. (See Figure 2.2 p 15 for examples.) They are used to test the final classifier using a scanning window. They are the most used Test images because they englobe a large radius of real situations and the exact position of the faces in these images has been manually labeled in a groundtrouth file containing the position of the eyes. This set differs from the learning set because the images are not 20x15 ones. The size of the images vary (roughly from 80x80 to 750x750). Testing these images allows to evaluate the detection rates, the speed of the detector (and the scanning window) and the behavior of the scaling system .

4.2 Learning results

This section explains the performances of AdaBoost and all the learning process in general by giving the results of different classifiers trained with different parameters such as the dataset used, number of features, the number of stages in the cascade, etc...

4.2.1 Weakness of the weak classifiers

It has been shown in theory that the weak classifiers used to train with AdaBoost need to satisfy one condition. They have to be better than random selection. That means that they have to classify correctly the examples in at least 50% of cases. Let us look how evolutes the error rate of the selected features. The model used is trained with 3000 faces and some 30.000 non faces. The results are shown in Figure 4.1.

We can notice that the best feature (the first selected by AdaBoost) misclassifies roughly 13% of the examples (faces and non faces are treated indifferently), while this error rate increases quickly until more than 40% for the last selected features. That shows clearly why the feature responses followed

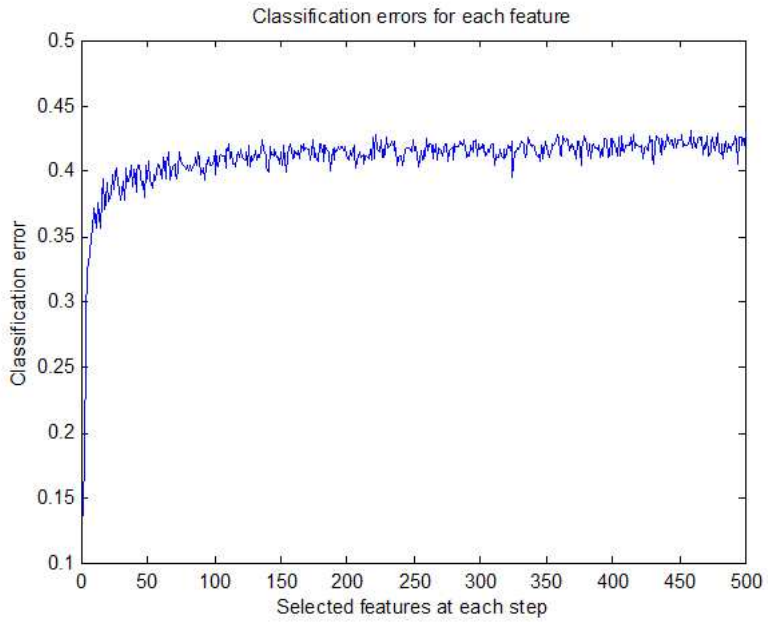


Figure 4.1: Classification errors of the selected features. The first selected feature classify well the examples in 13% of the cases while the 500th features only 42.5%.

by a threshold are qualified of weak. It proves that the challenge of boosting is to organize many of these weak classifiers into a linear combination followed by another final threshold.

4.3 Test results

4.3.1 Mono-stage classifier

Let us see what are the performances of a single stage classifier trained with about 37.520 features, 3000 faces and 30.000 non faces chosen by bootstrapping (false alarms from a previous simple classifier.). To evaluate the learning performances, i.e. the classification rates after the Boosting process, we have to recall the definition of two of the main evaluation errors:

1. **The Training error** noted ϵ_{tr} which is the error rate made on the training set:

$$\epsilon_{tr} = \frac{1}{N_{tr}} \sum_{i=1}^{N_{tr}} |H(x_i^{(tr)}) - y^{(tr)}(x_i^{(tr)})|,$$

where N_{tr} denotes the total number of training examples (positive + negative), $x_i^{(tr)}$ the i-th example and $y_i^{(tr)}$the label of the i-th example. The theory about Boosting shows that this training error tends to 0 when the number of iterations increases.

2. **The Testing error** noted ϵ_{te} which is the error rate made on the testing set:

$$\epsilon_{te} = \frac{1}{N_{te}} \sum_{i=1}^{N_{te}} |H(x_i^{(te)}) - y^{(te)}(x_i^{(te)})|,$$

where N_{te} denotes the total number of training examples (positive + negative), $x_i^{(te)}$ the i-th example and $y_i^{(te)}$the label of the i-th example.

In this first experiment, the testing set is made of 6.000 faces taken from the Banca Database and a part of the BioID database, while we use 30.000 randomly selected non faces. The Figure 4.2 shows the obtained results.

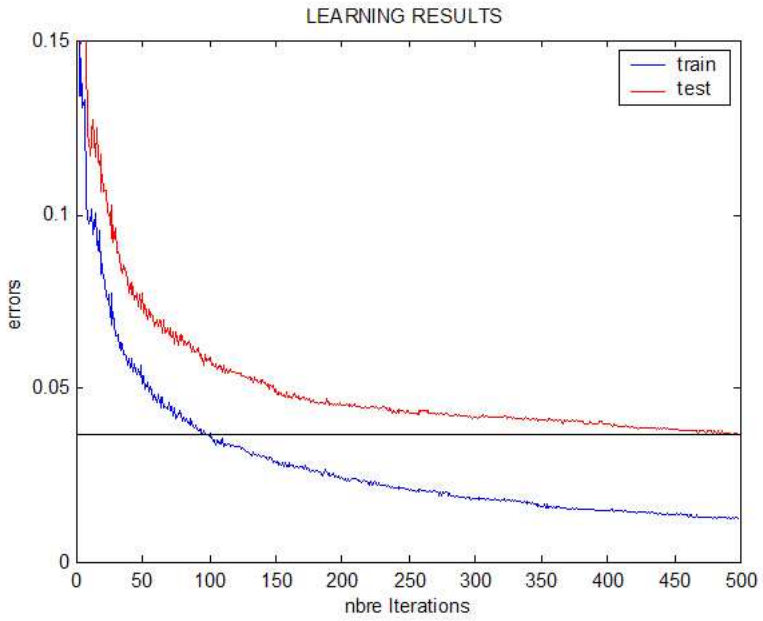


Figure 4.2: Training and Testing errors for a model trained on 3.000 faces and 30.000 non faces.

4.3.2 Results

Figure 4.3 shows some examples tested on the CMU Data set.

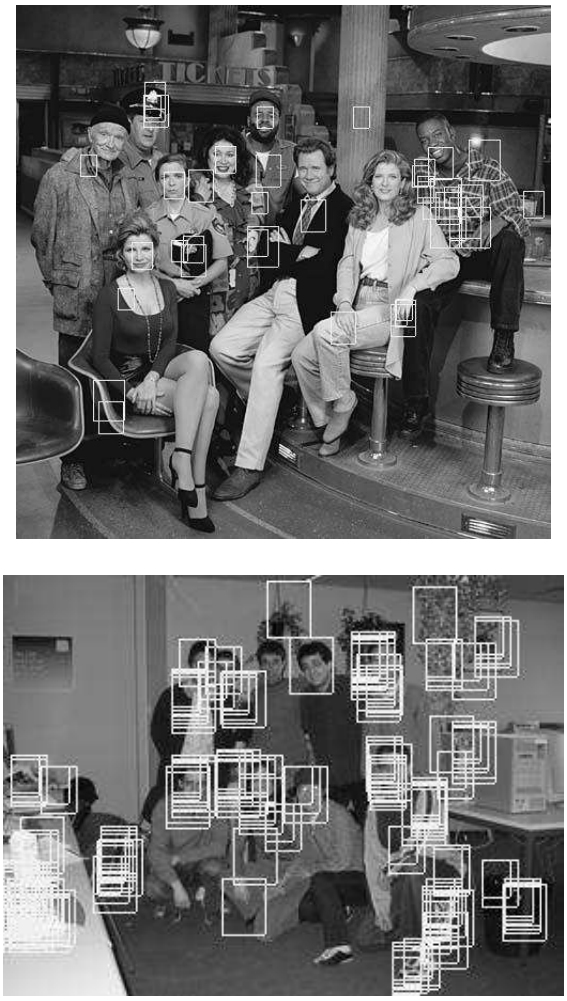


Figure 4.3: Examples of CMU images tested with a mono-stage classifier.

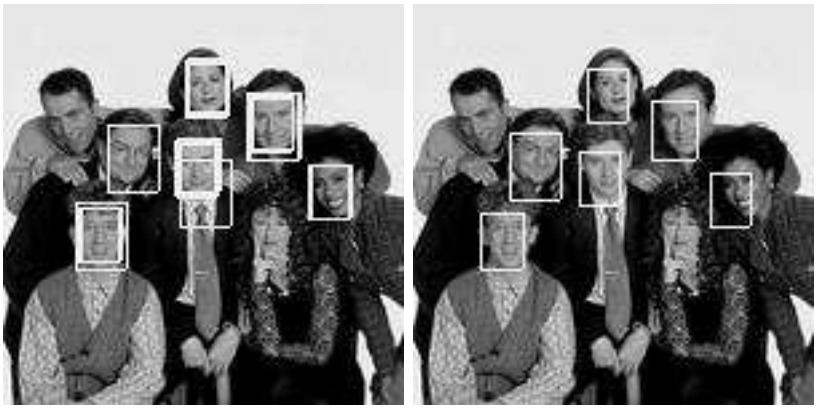
You can see that some of the faces are missed and there are still lots of false alarms. Some faces are missed because for this mono-stage classifier only the training error has been minimized during the training. That means that we have not specified to keep as many positive examples as possible while having higher false positive rate.

For this, as explained in the theory, to minimize the false negative rate, the final threshold in AdaBoost is decreased until all the positive examples are well classified. On the other hand, the false positive rate increases highly

when decreasing this threshold. To keep a reasonable false positive rate at each iteration step, we admit to miss a few faces.

4.4 The multiple detections

As it is shown in the last figures, it is difficult to see and evaluate clearly which are the regions of the images that contain faces. Indeed, many bounding boxes often frame a single face and the arbitration is made by the integration of multiple detections. Here are some pictures before and after the multi-detection algorithm.



(a) (b)

Figure 4.4: Integration of the multiple detections. (a) multiple detections: 17 positive windows. (b) after arbitrating: 6 windows.

The integration of these multiple detection is quite intuitive. All the detected windows in the image are clustered. Two windows are clustered together if their recovering area is higher than a predefined threshold. Then for each cluster, the final window is computed as the mean of the windows in the cluster. Thus, the center of the final window is the gravity center of all the centers and the definitive size is the mean of the size. One problem may arise when two faces are very close. In fact, when neighbor windows containing two close faces are detected, they are clustered together and so the resulting window can be between the two faces. Thus both of the faces may be missed.

An example is given in figure 4.5. To solve this problem, we can introduce another condition to cluster two windows together. The difference of their sizes has to be larger than a predefined threshold.

Another implementation which may be more appropriate would be the use of median windows instead of simple mean window. This is let to a future work.

Figure 4.5: Example of bad multiple detection integration. The black bounding box contains two faces.

4.5 The Cascade classifier

4.5.1 Training the cascade

4.5.1.1 Choice of the parameters

In this project, a cascade of classifiers has been developed. The final version of the cascade was built as follows.

First of all, we had to choose the training examples and the cascade parameters which determine the number of stages and the number of features in each stage.

We use an initial set of about 300.000 negative examples. This set is built by bootstrapping and then, each example is symmetrized in order to ensure the invariability to face illumination orientation. We have a total of 8500 faces.

The goal of the cascade is to apply classifiers more and more specialized when we go through the process. To reject quickly the great majority of negative windows while keeping a high detection rate, we have taken:

- $f = 60\%$: the false positive rate needed to add a new stage. That means that at each stage 60% of the negative windows that have passed the previous stages are rejected.

- $d = 99.9\%$: the detection rate needed at each stage. 99.9% of the positive examples kept by the previous stages have to pass through the current stage.

During the learning process, we start with roughly 300.000 negative examples. Then, at each stage, only the examples that are considered as positive are kept for the next training set. Thus, the stages in the process are directly trained to classify the examples that have been misclassified by the previous ones. The examples that are hardest to classify are left to the last stages of the cascade. Thus we have trained a cascade of 20 stages.

Table 4.1 shows the training results.

Stage Number	Size of the Negative Training set	Feature Number
1	298.900	5
2	163.149	7
3	97.687	10
4	55.088	13
5	32.124	13
6	19.245	14
7	11.330	17
8	6.418	21
9	3.740	24
10	2.173	28
11	1.271	33
12	745	43
13	417	49
14	249	40
15	88	23
16	49	22
17	28	10
18	12	09
19	5	2
20	1	2

Table 4.1: Training Results. Number of negative examples in the train set and the corresponding number of features used.

One of the limitations of this cascade is when the negative example number becomes too small. As shown in Table 4.1 and in Figure 4.6, the number of features per stages decreases from the $14-th$ stage. In fact, AdaBoost uses only 249 negative examples to train the model. It seems to be not consistent

enough and it is possible to classify these examples with less features. Thus, we will only use the 14 first stages in our final face detector.

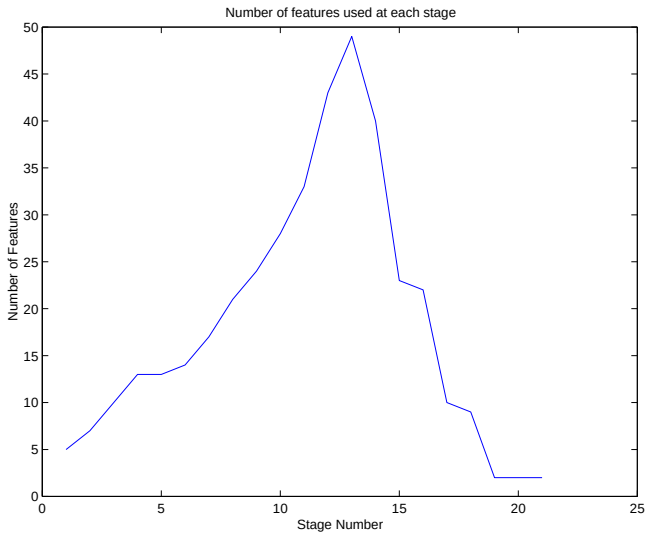


Figure 4.6: Number of features per stage.

4.5.1.2 Discussion

Let us see what are the main characteristics of building a cascade. Many variants could be used to train a cascade of classifiers and is is not easy to find the most appropriate one. For example, we have chosen to determine the number of stages and the number of features in each stage given the detection rates targets. Let us wonder if it would be better to build a cascade with more stages and fewer features in each stage or less stages with more features.

Adding a stage in the cascade produces a reduction of the number of false alarms which is one of our main goals. However, the stages have to be more and more specialized while the number of stages increases so more features are needed to obtain the targets specified and it is clear that the detection rate is more affected by the last stages than the first ones which are few discriminant. This notice has been verified during the tests of the detector by adding more or less stages.

Another implementation could be used to train the cascade. Instead of fixing the target parameters f and d, we could decide a priori to build a cascade structure as follow: Fix for each cascade stage the number of features (for example 2 features in the first stage, 5 in the second, 10 for the two following stages, etc..) and then add stages while the global rates targets are not reached. This method does not guarantee to have the same efficiency

for all stages but the global performances of the cascade are similar to our implementation.

4.5.2 Results of the cascade

This section gives the main results that have been obtained using a cascade of 14 classifiers. The first stage has five features while the second one has seven. The number of features increases until the last stage which has 40 features. Figure 4.7 shows the rate of negative examples rejected at each stage in the cascade. It shows clearly the principle of the cascade. The first stages reject a great majority of negative examples, those which are easy to distinguish from faces and then as the number of stages increases, the remaining examples are harder to reject. The last stage focus only on few examples hard to classify which we could call face-like examples.

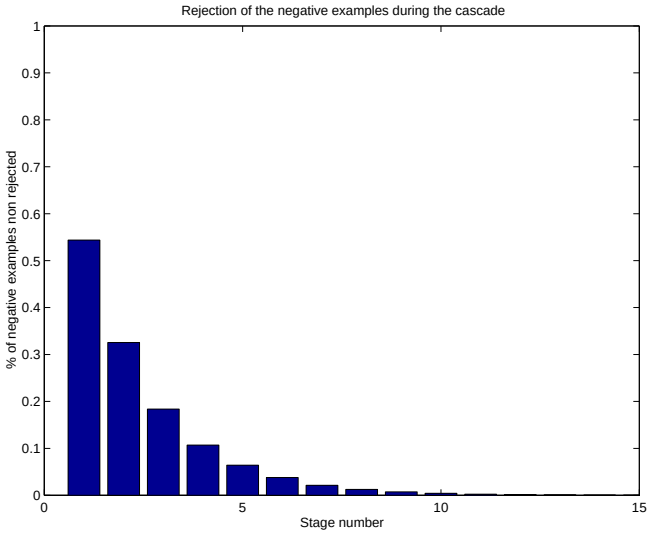


Figure 4.7: Evolution of the false positive rate during the cascade of 14 stages.

The evaluation of the performances of the classifier can be divided into two groups: the test on the test set made of 8500 faces and about 900.000 non faces and the test on the CMU Dataset.

4.5.2.1 Experiments on a Real-World test Set

The detector has been tested on the MIT-CMU frontal face test set [23].We have 132 images with a total of 507 frontal faces. The performances of the detector can be placed at every fonctionning point. By changing the

final threshold of the detection, we can modify the detection rate and the false positive rate. Decreasing the threshold will yield a better detection rate (100% if the threshold equals 0) but the false positive rate will increase slightly. On the other hand, increasing the threshold will decrease the number of false alarms and also the detection rate. We can choose the threshold depends on the goals of the detector. In our case, we want a high detection rate so the threshold will be quite low.

In these tests, we use a shifting step of 1 pixel for the scanning window and a scale factor of 1.25.

The next figures show some results using this detector.

Figure 4.8 shows well detected faces in the set of CMU while Figure 4.10 shows some poor detections.

Figure 4.8: Results of the detector using a cascade of 14 classifiers. Good Detections

Figure 4.9: Results of the detector using a cascade of 14 classifiers. Example of good detections with some false alarms.

Figure 4.10: Results of the detector using a cascade of 14 classifiers. Example of poor detections. Some faces are missed and false alarms are detected.

The detector using these 14 stages is robust and quite efficient on the CMU Dataset. We obtain 85% of detection rate considering the 507 faces in the test set and 55 false alarms. 85% of detector rate may appear insufficient but we have to take into account the contain of the CMU set. Some of the 507 faces in the 132 images are not well adapted to be detected by our cascade. In fact some of the faces are not really frontal faces but more profile views, other faces are manually drawn or photographs of cards and these kinds of examples were not in our training set. So they were not supposed to be classified as faces. We have also used these examples and taken them into account in our test because they already were used in testing previous detectors. Thus we can compare our work with existing methods.

Our detector is quite efficient on this set even if Viola's detector is more efficient. We can notice that Viola' s detector contains 32 stages with more

features which yields lower false positive rate. For more examples, see Annexes A.3.

4.5.2.2 Experiments on a Video sequence

The detector has also been tested on video sequences. We have applied our detector on 220 images successively. The situation is a classical real time application: a video camera was placed in a corridor and some people crossed the scene. The detector is quite efficient and it permits us to evaluate the power of the features. In fact, faces were quite well detected during the sequence but not in every image. A well detected face in the image number i was not necessarily detected in the image $i + 1$ even the visual difference between the two images is very low. That shows the detection is really precise. A single pixel may have lot of weight in the detection.

We have to notice that in this test, no time integration was used. Figure 4.11 shows the general scene of this video sequence test.

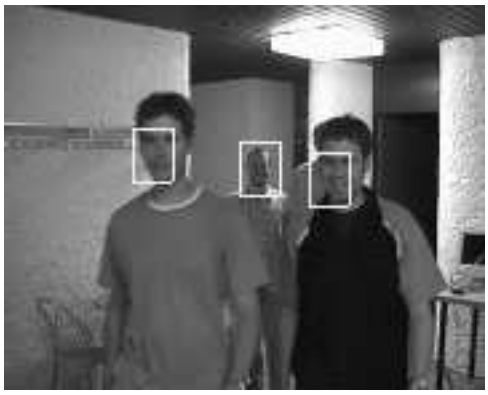


Figure 4.11: One image of the video sequence used to test the detector.

4.6 Speed of the detector

The method used is particularly well adapted to real time applications. In fact, the computed model needs few operations to be applied on an image. The scanning windows is clearly the slowest part of the decision but we have seen that the integral image representation and the use of the rectangular Haar-like features authorize an efficient way to implement the scaling of the window. No down-sampling is needed because it is the detector itself that change of size.

However, the implementation of our final detector is not well adapted and the code is not optimized enough to obtain good speed performances. It will be the first step of the future work.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

We have seen in this report the complexity of the face detection task. Many methods can be used existing a precise context for each of those methods. We have chosen an intermediate method between the image based and the feature based detection method. A face detection system has been developed using a Boosting algorithm and simple rectangular Haar-like features. This method presents many advantages in comparison with other methods for detecting faces.

- The frontal face detector yields very good detection performances (in term of ratio over detection rate and false positive rate).
- The performances can be highly increased in specific applications. In fact, if the face detector is placed in a fixed scene, the training set can be adapted to this scene to build a robust detector.
- The computation of the classifier is very fast because of the use of simple rectangular features which are easily computed with the integral image.
- The method can be easily adapted to other kind of application such as pedestrian detection for example. The principle is to change the training sets.

However, we have to recall that this method, as every face detection method has its own limitations:

- The detector has been trained with only frontal faces with uniform pose.

- It is difficult to predict the optimal values of some parameters. It is the case, for example, of the number of training examples. The efficiency of the final detector depends directly on this dataset. The main problem remains the generalization power of the trained model. If the model is efficient on the train data set, we do not know a priori how it acts on the testing data. Particularly, if the train set is too complete, the classifier will be too specialized on the train faces and some faces may be missed on real set images.

- This detection of low resolution images gives the position of eventual faces and an idea of their size. However we do not have a precise position of the faces. We do not know precisely the position of the eyes for examples such that a further face analysis may be applied in some applications.

Our face detection system present good results. Face detection using AdaBoost is an interesting compromise between image based and feature based methods. We use a learning procedure to extract feature which are linked to the geometrical characteristics of faces. We can distinguish three main contributions in this face detection system: The use of rectangular Haar-features computed efficiently with a new image representation called image integral, the learning algorithm AdaBoost which selects the best set of these Haar-like threshold and finally an implementation in cascade which permits to decrease the detection time while increasing the detection rates.

The result is that the detector is efficient in terms of detection rate in spite of a non negligible number of false alarms.

5.2 Future Work

The results of this face detection system are really promising but it is clear that many improvements are possible. The main improvements could target the detection speed and the detection performances.

- The first improvement possible is certainly to modify the implementation of our scanning processing in order to take profit of the power of the cascade. The classification of a window needs only few processor instructions.

- Then the training of the cascade may be improved using more appropriated training sets. The problem is to find an optimal set with a large generalization power. The Bootstrapping principle seems to be

particularly well chosen and it could be more used during the learning process to improve the robustness of the classifier.

- Our final cascade has finally 14 stages with a total of some 100 features. We could modify a little the cascade learning to obtain more stages and features. This would reduce the number of false detections while keeping good detection rates. One of the limitation of Boosting is the learning time so more time would certainly give more consistent stages.

- Another interesting work would be to try other kind of features. For this, many solution can be tried. For example, we could introduce rotated Haar-like features with the corresponding equivalent of integral image representation. Some other features could be tried.

- The learning process could be improved by introducing the advantages of decision trees into the learning via AdaBoost.

- The case of detected rotated faces has to be taken into account. Many methods are possible. The first method would be to introduce directly rotated images into the face training set such that a single cascade would detect the faces at each given angle. A second method consists in using the vertical detector and applying it with a rotation of the scanning window. The problem would be the approximations of the rotated features which would decrease the global performances. The last method seems to be the most appropriate. It would be to train a cascade for each given angle such that each cascade would be specialized for one angle. All the cascades would then be applied to the scanned windows.

Appendix A

Annexes

A.1 Scanning implementation

Like the most of the Image-based methods, we use a sliding window to scan the images. The implementation of this sliding window needs some care for different reasons if we want to obtain good results in term of detection rate and speed.

Recall that our basic detector is a 20×15 pixels and that the training of the classifiers using AdaBoost has been made with examples of the same size. We want to detect faces from different scales and positions so we have to re-scale this basic detector along the scanning process. We have seen that the Integral representation permits to re-scale the detector instead of using a traditional image pyramid, i.e. no down-sampling is needed to scan the image at different scales. There are three main issues about this window scanning:

- How can we change the detector size (and the size of th features) without changing the properties of the selected features?
- How can we choose the shifting and scaling steps to find a good compromise between speed and precision?
- How can we integrate the multiple detections?

A.1.1 Rescaling the window

Let us see how to re-scale the basic window of $h \times w$ where h and w are respectively the height and the width of the initial window. Suppose that we want to re-scale this window by a factor of $\Delta = 1.25$. It is clear that the h and w values are integers so we have to make an approximation to obtain

a window at the next scale. We choose to preserve the ratio $r = h/w$ to keep a window of the same shape as the basic detector. Thus, the size of the window rescaled by Δ is :

$$(\lfloor \Delta \cdot h \rfloor) \times \left(\lfloor \frac{r}{\lfloor \Delta \cdot h \rfloor} \rfloor \right)$$

where $\lfloor \cdot \rfloor$ represents the rounding operation towards the largest integer smaller than the argument $\cdot$.

The problem is now to re-scale the Haar-like features which have been selected by AdaBoost. For example , consider a 2-rectangle feature as defined in Figure :

Denote x and y the coordinates of the up left corner of the feature, h_f and w_f the height and width of the feature. We choose another time to keep the ration between height and width such that $r = \frac{h}{w} = \frac{x}{y} = \frac{h_f}{w_f}$. So the feature is rescaled and moved proportionally in the window. The problem is that rounding the values of the coordinates of the rectangle's corners change a few the properties of the mask. The rescaled mask is one mask which is maybe not so good than the basic one, and it would maybe not be selected by AdaBoost. However, there is no other possibility than making an approximation.

The experiments made on several set of images shows that this issue has not too bad consequences on the final results. In fact choosing sufficiently small shift step and scale step ensure to detect faces in different scales so even if a face is missed because the imprecision of one scale, it is in most of the cases detected at other scales.

A.2 AdaBoost implementation

We have seen that AdaBoost is a powerful learning algorithm which selects the best weak classifiers given a set of training images. One of the main drawbacks is that the result depends highly on the size and consistence of the datasets. Our final choice has been to choose a training set containing some 300.000 negative examples and more than 8000 positive ones. Recalling that there are 37520 features, the computing time during the learning is quite long as well as the memory used is big.

For example, if we want to build a simple mono-stage classifier using 200 features. The first step is to write the integral images of all the positive and negative examples. Considering our 308.000 images 15×20 (which means 300 integers per image) written into a binary format, the integral image file takes 352 Mb on the disk. Then, given these integral images, we have to compute

the total set of features responses. Indeed, as the same feature responses are used at each learning step, it would be two heavy to compute them at each iteration. A feature response is an integer (sums and differences of integers) and we have a total of 37520 features which means that a single file containing all the features responses would have a size of 43 Gb. Assuming that this file would be created, we have then to read it completely at each iteration step (It can not be loaded into the memory in one time, of course). It would take many days to build a classifier with these data.

In order to improve that, we have chosen to work using a parallel implementation to distribute the work on several processors. For this we have used the MPI (Message Passing Interface) library. The cluster on which we have provided the training has 5 machines and we just launch 2 processes on each machine so we have a total of 10 process than can work in parallel.

The parallel method chosen in a traditional master slave implementation. The master process (Process 0) send the respective data to each of the 10 slaves processes and then clusters all the independent results. In the practice, the repartition of the processes is made as follow: As we have to evaluate and compare the features responses for 308.000 images and 37520 features, each slave process will treat 3752 features for all the images.

The process 0 sends the image weights to each slave process, then each process finds, independently of the others, the best feature into the set of 3752 and send the results back to the process 0 (the index of the best feature and the corresponding classification error). Finally, the master process compares the results of the 10 processes and extract the best of the 10. Then the weights are reevaluated and a new iteration begins with the new weights.

Thus the total computing time is reduced by a factor of 10 (or a few less if we take into account that all the processes have access to the same hard-disk in the same time). Note that each feature file written in a binary format still need 4 Gb and they have to be read completely at each iteration step. Thus, building a complete classifier of 200 features needs roughly one week to compute.

A.3 Examples

This annexes relates complementary examples from the CMU Database.

Figure A.1: Examples of the CMU Database.

Figure A.2: Examples of the CMU Database.

Bibliography

[1] Paul Viola & Micheal Jones. Robust real-time object detection. Second International Workshop on Statistical Learning and Computational Theories of Vision Modeling, Learning, Computing and Sampling, July 2001.

[2] C.Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In *International Conference on Computer vision* , 1998.

[3] M. Betke and N. Makris. Fast object recognition in noisy images simulated annealing. In *Proceedings of the Fifth International Conference on Computer Vision,* pages 523-20, 1995.

[4] 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, December 1994.

[5] Pavlovic V. and Garg A. *Efficient Detection of Objects and Attributes using Boosting.*IEEE Conf. Computer Vision and Pattern Recognition. 2001.

[6] Mallat S. *A theory for the multi-resolution signal decomposition: the Wavelet representation.* IEEE Pattern Analysis and Machine Intelligence, Vol.11, n° 7, pages 676-693, 1989.

[7] Rainer Lienhart and Jochen Maydt, An Extended Set of Haar-like Features for Rapid Object Detection, Intel Labs, Intel Corporation, Santa Clara, CA 95052, USA

[8] F. Crow. Summed-area tables for texture mapping. In Proceedings of SIGGRAPH, volume 18(3), pages 207212, 1984.

[9] Patrice Y. Simard, Lon Bottou, Patrick Haffner, and Yann Le Cun. Boxlets: a fast convolution algorithm for signal processing and neural networks. In M. Kearns, S. Solla, and D. Cohn, editors, Advances in Neural Information Processing Sys-tems, volume 11, pages 571577, 1999.

[10] William T Freeman and Edward H. Adelson. The design and use of steerable filters. IEEE Transactions on Pattern Analysis and Machine Intelligence, 13(9): 891-906, 1991.

[11] H. Rowley, S. Baluja and T. Kanade. Neural network-based face detection. In *IEEE Patt. Anal. Mach. Intell.,* volume 20, pages 22-38, 1998.

[12] Yoav Freund and Robert and E. Schapine. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory: Eurocolt '95*, pages 23-37. Springer-Verlag, 1995.

[12] L.G Valiant. A theory of the learnable. *Communications of the ACM,* 27(11):1134-1142, November 1984.

[13] G. Rätsch, T. Onoda K-R. Müller. Soft Margins for AdaBoost, Machine Learning, 1-35, August 1998.

[14] Duffy and D.P Helmbold. Boosting methods for regression. Technical report, Department of Computer Science, University of Santa Cruz, 2000.

[15] http://www.boosting.org

[16] L. Breiman. Bagging predictors. *Machine Learning, 26(2):123-140,* 1996.

[17] Y. Freund and R.E Schapire. Game theory, on-line prediction and boosting In *Proc. COLT, pages 325-332,* New York, NY, 1996. ACM Press.

[18] R.E. Schapire. A brief introduction to boosting. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence,* 1999

[19] V.N. Vapnik. *Estimation of Dependencies Based on Empirical Data.* Springer-Verlag, 1982

[20] P. Viola and M. Jones, *Fast and Robust Classification using Asymmetric AdaBoost and a Detector Cascade*. Mitsubishi Electric Research Lab, Cambridge, MA. 2001

[21] R. Meir and G. Rätsch. *An Introduction To Boosting and Leveraging.* Department of Electrical Engineering, Technion, Haifa 32000, Israel.

[22] M.H. Yang, D. Kriegman, N. Ahuja, *Detecting Faces in Images: A survey.* IEEE Transactions on pattern analysis and machine intelligence, vol. 24 n° 1, January 2002.

[23] CMU Dataset: ftp://whitechapel.media.mit.edu/pub/images/.

[24] P. Belhummer, J. Hespanha, and D. Kriegman. *Eigenfaces versus fisherfaces: Recognition using class specific linear projection.* IEEE Transactions on Pattern Analisis and machine Intelligence, 19(7):711-720, 1997.

[25] D. Pissarenko , *Eigenface-based facial recognition*, December 1, 2002

[26] P. Hallinan. *A low* dimmensional *representation of human faces for arbitrary lighting Conditions.* In Proc. IEEE Conf. on Computer Vision and Pattern Recognition. pages 995-999, 1994.

[27] L. Sirovitch and M. Kirby, *low dimensional procedure for the caracterization of human faces.* J. Optical Soc. of America A, 2:519-524, 1987

[28] M. Turk and A. Pentland. *Eigenfaces for recognition.* J. of Cognitive Neuroscience, 3(1), 1991.

[29] Y. Moses, Y. Adini and S Ullman. *Face Recognition: the problem of the compensating for changing in illumination direction.* In Europeean conference on Computer Vision, pages 286-296, 1994.

[30] G. Yang and T. S. Huang. *Human face detection in a complex background.* Pattern Recognition, 27(1):53-63, 1994

[31] A. Lanitis, C. J. Taylor, and T. F. Cootes. *An automatic face identification system using flexible appearance models.* Image and Vision Computing, 13(5):393-401, 1995

[32] T. Leung, M Burl and P. Perona. *Finding Faces in cluttered scenes using labeled random graph matching.* In Proc. 5th Int. Conf. on Computer Vision, pages 637-644, MIT, Boston 1995.

[33] Y. Sumi and Y. Ohta. *Detection of face orientation and facial components using distributed appearance modeling.* In Proc. Int. Workshop on Auto. Face and Gesture Recogn., pages 254-259, Zurich, 1995.

[34] K. C. Yow and R. Cipolla. *Scale and orientation invariance in human face detection.* In Proc. British Machine Vision Conference, pages 745-754, 1996.

[35] J. Cai, A. Goshasby, *Detecting human faces in color images.* Image and Vision Computing, 18:63-74, 1999.

[36] M. H. Yang and N. Ahuja. *Detecting human Faces in Color Images.* Beckman Institute and Department of Electrical and Computer Engineering University of Illinois at Urbana-Champaign., Ubrana, IL 61801.

[37] T. Pham, M. Worring. *Face Detection Methods, A Critical Evaluation.* Intelligent Sensory Information Systems, Department of computer science., Amsterdam 2002.

[38] BioID Face Database. *www.humanscan.de/support/downloads/facedb.php*

[39] E. Bailly-Baillire, S. Bengio, F. Bimbot, M. Hamouz, J. Kittler, J. Mariethoz, J. Matas, K. Messer, V. Popovici, F. Poree, B. Ruiz, J.P. Thiran. *The BANCA Database and Evaluation Protocol.* IDIAP, Martigny, Switzerland, IRISA, Rennes, France, ITS-EPFL, Lausanne Suisse, University Carlos, Madrid Spain, University of Surrey, Surrey UK. 2003.

[40] XM2VTS Database. xm2vtsdb.ee.surrey.ac.uk

[41] F. Samaria and S. Young, *HMM Based Architecture for Face Identification*, Image and Vision Computing, vol 12, p 537-583 1994

[42] F. Samaria. *Face Recognition Using Hidden Markov Models.* PhD Thesis, University of Cambridge, 1994.

[43] K-K Sung, *Learning and Example Selection for Object and Pattern Detection,* PhD Thesis, Massachusetts Institute of Technology 1996.

[44] A. Rajagopolan, K. Kumar, J. Karlekar, R. Manivasakan, M. Patil, U. Desai, P. Poonacha and S. Chaudhuri. *Finding Faces in Photographs,* Proc Sixth IEEE Int'l Conf. Computer Vision, 1998

[45] N. Littlestone. *Learning quickly when irrelevant attributes abound : a new linear-threshold algorithm.* Machine learning vol 2 pp 285-318 1998

[46] V. Govindaraju, S.N. Srihari, D.B Sher. *A computational model for face location.* In Proc. of the Third International Conference on Computer Vision, pages 718-721, 1990.

[47] E. Saber and A.M. Tekalp. *Frontal-view face detection and facial feature extraction using color, shape and symmetry based cost functions.* Pattern Recognition Letters, 19(8), pages 669-680, 1998.

[48] G. Wei, I.K Sethi. *Face detection for image annotation.* Pattern Recognition letters, 20(11), pages 1313-1321, 1999.

[49] Osuna E., Freund E. Girosi F.: *Training support vector machines: an application to face detection.* In Proceedings of Computer Vision and pattern recognition. 1998 45-51.