Albert Pumarola Peris

Master's degree in Automatic Control and Robotics

June 2016

End of Bachelor Degree Project

Master's degree in Automatic Control and Robotics

Optimization of common computer vision algorithms

Beating OpenCV Face Detector

MEMORY

Author:	Albert Pumarola Peris
Director/s:	Cecilio Angulo Bahón
School:	School of Industrial Engineering of Barcelona
Call:	June 2016

ETSEIB

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

UPC

Abstract

In this Master Thesis one of the most common problems related to face detection is presented: fast and accurate unconstrained face detection. To deal with this problem a new general learning method is presented.

The proposed method introduces a set of upgrades and modifications on key concepts and ideas of Decision Trees, AdaBoost and Soft Cascade learning techniques. Firstly, a new variation of Decision Trees with quadratic thresholds able to maximize the margin distance between classes is introduced. Considering a training set independent of face orientation and viewpoints information, the proposed algorithm is able to learn a combination of features to cluster faces under unconstrained face position and orientation. Next, a new definition of the Soft Cascade thresholds training principles is provided. Hence, this modification leads to a better formulation of the loss function associated to the AdaBoost algorithm.

The trained face detector has been tested over the Face Detection Data Set and Benchmark (FDDB) and compared against the current state of the art classifiers. The obtained results show that the proposed face detector (i) is able to detect faces with unconstrained position, and (ii) it works faster than the current state of the art methods.

Contents

ETSEIB

ETSEIB

List of Figures

List of Tables

ETSEIB

List of Algorithms

Abbreviations

CMU	**C**arnegie **M**ellon **U**niversity
RAM	**R**andom **A**ccess **M**emory
GPU	**G**raphics **P**rocessor **U**nit
CPU	**C**entral **P**rocessing **U**nit
II	**I**ntegral **I**mage
NPD	**N**ormalized **P**ixel **D**ifference
HOG	**H**istogram of **O**riented **G**radients
SVM	**S**upport **V**ector **M**achine

Mathematical notation

I	Image
$H(x)$	Strong Classifier
$h(x)$	Weak Classifier
f	Feature
w_s	Weight of a sample s
α_t	Weight of a Cascade Weak Classifiers $h_t(x)$
θ_t	Threshold of a Cascade Weak Classifiers $h_t(x)$
S_j	Node j of a Decision Tree
S_j^L,S_j^R	Left and right sons of a Decision Tree node S_j
Θ_{j1},Θ_{j2}	Thresholds of a Decision Tree node S_j

Chapter 1

Introduction

1.1 Information, context and project description

It is said that the face is one of the most powerful channels of non-verbal communication. Data extracted from a person's facial behaviour can be used as a predictor of his/her internal states, social behaviour, biometrics and psychopathology. In other words, just by analyzing a person's facial behaviour we could estimate if that person suffers from depression, the level of interest of an audience attending a talk, car drivers fatigued, even detecting security issues. These are just some of the applications of the face analysis field, but its possibilities are limitless. Face analysis pipelines are really complex an diverse, but they all have one common component, a *face detection* step. This step is critical as no analysis can be performed without first detecting the person face. Hence, the main goal of this project is to develop a fast, accurate and unconstrained face detector.

Face detection is a well studied problem in the Computer Vision research area. Modern face detectors can easily detect frontal faces near to the camera and it is a well solved problem. Current research focuses on detection of faces in unconstrained positions. In this case, there are two main difficulties: large visual variations of the same face due to illumination, occlusions, expression variations and image resolution; and the huge search space of possible face positions and sizes for a given image. Current state of the art algorithms obtain very good detection results, but with a very high computational complexity.

The new proposed learning method introduces a set of upgrades and modifications of the key concepts and ideas of Decision Trees, AdaBoost and Soft Cascade. First, a new variation of Decision Trees is presented. Upgraded with quadratic thresholds and able of maximizing the margin distance it is capable of learning the optimal combination of features to cluster faces under unconstrained face position and orientation. Moreover, the proposed solution does not require face orientation and viewpoints training labels. It is capable of automatically clustering the different viewpoints at their lower levels. Hence, the learned face detector is more robust to unconstrained faces than the state-of-the-art methods based on learning one model for each of the possible views, and then, for a given sample, trying to guess which ones to run.

Second, a new definition of the Soft Cascade thresholds learning method is proposed. Instead of first training the cascade and then calibrating the thresholds, it is proposed to learn them during the training stage. Furthermore, instead of learning them with the current naïve based approach used in the current state of the art Soft Cascade, thresholds are learnt using a data-based method relying on support vector machines.

Third, by joining the learning and calibration stages of the Soft Cascade learning, it is opened the possibility to define a better formulation of the AdaBoost loss function.

The trained face detector has been tested over the new Face Detection Data Set and Benchmark (FDDB) and proved to achieve state-of-the-art performance and beat in terms of memory, speed and accuracy the Viola-Jones method and most of the current state-of-the-art methods.

The associated research work leading to this Master Thesis has been developed at the Carnegie Mellon Robotics Institute, in the Human Sensing Lab. The mission in this research laboratory is to develop advanced computational tools to model and understand human behavior from sensory data. Their main research topics are human sensing, component analysis and face analysis.

The implemented face detector will be used as the face detection module of their face analysis pipeline. It represents a high honor and responsibility to design and implement the face detection module because this laboratory is considered to be one of the best ones in face analysis research. They are currently using the Viola-Jones OpenCV implementation. Thus, the main goal is to beat this detector.

1.2 Scope

As a main goal in this Master Thesis, a new learning method for face detection will be designed and implemented from scratch, without the use of any particular library, but image data manipulation and *Support Vector Machines.* Its main purpose will be to detect faces which will after be analyzed to extract behavioral states of a person in real time. Neither data extraction nor analysis can be performed if the face is not well detected, hence the detector module must work in real time with high detection accuracy. It is also desired that the detector works with unconstrained positions. It is out of the scope of this project to detect small-blurry faces or faces with high level of occlusion because even if detected, no good posterior analysis could be performed.

Work in this thesis is more than a theoretical research, but a real world product release project. The designed detector will work in a constrained laboratory environment, but also must be robust to real world problems as well as memory and computation complexity. The implemented detection algorithm will not only run on computers, but also on devices with less computational power such as smartphones and tablets. As a consequence, its computational complexity must be very low, and cannot rely on a GPU for better reason. Its memory complexity must also be low because cellular users are not prompted to install apps asking for high memory resources. For the scope of this project, it is also supposed that the device has at least 1GB of RAM, a dual-core processor and a camera with a minimum resolution of $500x480$ pixels.

1.3 Objectives

The main objective of this project is to implement a face detector working on popular cellulars with the following features:

- Beat the current face detector module used in the laboratory pipeline, the Viola-Jones OpenCV implementation.
- All the designed training algorithms and features must be implemented from zero (except the Support Vector Machine algorithm).

- Low computational complexity: the module should run at a minimum speed of 100 Hz on a dual-core processor, in GPU-denied devices.
- Low memory complexity: it should be able to run with a maximum memory consumption of 500 MB and with a model lower that 15 MB.
- State-of-the-art accuracy for fast algorithms should be obtained.
- Detect faces with unconstrained position in cluttered environments.
- Highly modular: It should be easily adapted to new features and classifiers.
- Modern code: all algorithms and features must be implemented with highly efficient, CPU optimized modern code.
- Transparent: develop a set of scripts which can display and plot the behaviour of the detector and the learning process to make them as much transparent as possible.

Personal objectives:

- Learn
- Learn
- Learn... not only theoretical knowledge but also the way of working and understanding the passion and almost an obsession of Carnegie Mellon University researchers.
- Enjoy the experience of being on one of the best computer vision university and research groups.

1.4 Resources

To develop this project a set of hardware, software and human resources have been used.

Hardware

- Server where the proposed training algorithm is executed. 20 cores Xeon E7 v2/Xeon E5 v2/Core i7 DMI2, 180 GB of memory, no graphic cards.
- Sony Vaio computer where the learned model is tested in real time.

Software

- Ubuntu 12.04
- Vim text edtior.
- ShareLaTeX, LaTeXonline text editor.
- BitBucket, a github web-based hosting service.
- Mendeley, a PDF manager and reader.

Humans The author of this project and the members of the *Human Sensing Lab* experts in the field of computer vision. These experts will bring feedback during the development of the project.

Chapter 2

State of the art

Face detection is a deeply studied problem and a lot of different approaches and methods have been developed. The most cited work in this field is the famous Viola and Jones paper [1] which introduced in 2013 the concepts of Integral Image based Haar features, AdaBoost based learning model and the combination of the classifiers in a Cascade structure. Since then, a lot of new approaches mainly focused on using different types of features and classifiers have been proposed. Currently we can cluster all methods in either, traditional or modern approaches. Traditional approaches are based on traditional classifiers such as Support Vector Machines (SVM) [2], AdaBoost with Cascade [1] and random Decision Trees [3]. In the other hand, modern approaches are based on using highly complex non-linear models generated with Deep Learning [4–6] or by using a 3D model of the face [7].

Main differences between both approaches are the size of the training data and time needed to train, the computational and memory complexity, and accuracy. Traditional approaches require much less number of samples and time to train, and they also have a much lower computational and memory complexity, making them much faster than modern approaches. However, modern approaches lead to higher accuracy. They claim to obtain even a higher accuracy than a human. Highly difficult benchmarks like FDDB [8] show how modern approaches have an accuracy of almost 90% vs 50% of traditional approaches. However, this high accuracy leads to a high computational complexity (15s for a full-HD image) v's 10ms in traditional approaches. There are multiple works which try to merge these two approaches, trying to get the best of both approaches. A common way is by having a Cascade of Convolutional Neural Networks (CNNs) [6, 9].

This project will focus on traditional approaches because of the real time constraint. Most traditional face detection methods are appearance based.Many different features has been proposed such as Haar [1], Local Binary Pattern (LBP) [10], Histogram of Gradients (HoG) [11], SURF [12], Local Binary Pattern histogram [13] and Normalized Pixel Difference (NPD) [14]. Detectors with manually defined features are usually faster in speed but more simple [15]. The most famous ones are based on Haar kernels which are the most common used features for frontal face detection. Texture based methods like LBP have been proven to be robust in front of monotonic light changes. HoG is able to better generalize non-face images, but has a higher computational cost. SURF features improve face detection performance by due to its high dimensionality it critically slows down the training process. Comparison of pixel intensity features [14, 16, 17] does not contain as much information as conventional features, but they are much faster as they do not need to process anything by using a look-up-table.

Not only different features have been introduced in the literature, but also modifications in the AdaBoost algorithm, like Vector Boost [18] and MBHBoost [19], as well as the election of the boosting weak classifiers. Multiple studies [14, 20] have demonstrated that using Decision Trees (CART) as the boost weak classifier tends to give much better results than just a simple Stump.

All state-of-the-art traditional methods have troubles when trying to detect multiview faces. Most common approaches consist of training a set of cascade detectors for each possible view, such as Parallel Cascade [21] and Pyramid architecture [22]. To avoid having to execute all models for each one of the evaluated regions of interest (ROI) they propose a previous step in which an approximation of the face orientation is estimated and only the classifiers near to that configuration are evaluated. Training of multiple classifiers for all the possible face position configurations increase the computational and memory complexity as well as the amount of labeling required. Moreover, a bad estimation of the face position would lead to face not to being detected. It is proposed in [14] a method based on Deep Quadratic Trees to automatically learn a general unconstrained face position model without the need of a labeling stage.

Chapter 3

Theoretical background

The implemented face detector is based in a Soft Cascade of weak classifiers trained with Adaboost. In this chapter all the theoretical background related to the framework is presented.

3.1 AdaBoost

Boosting is an approach to machine learning based on the idea of creating a highly accurate classifier by combining many simple and easy to find rules / subclassifiers. Freund and Schapire [23] proposed AdaBoost, the first practical boosting algorithm. The idea is to ensemble a set of T weighted really simple classifiers with an error rate just below random, called weak classifiers, to create a higher order classifier named strong classifier. Boosting methods are expressed as:

$$H(x) = \sum_{t=1}^{T} sign(\alpha_t h_t(x)) \tag{3.1}$$

where $H(x)$ is the strong classifier which is composed by T weak classifiers $h_t(x)$,

$$h_t(x) = \begin{cases} +1, & \text{if } x \text{ classified as a positive sample} \\ -1, & \text{if } x \text{ classified as a negative sample} \end{cases} \tag{3.2}$$

weighted by α_t, and x being the sample to be classified.

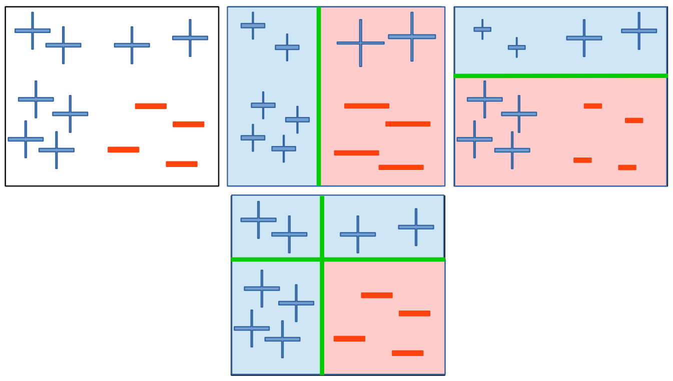


FIGURE 3.1: Dummy example of AdaBoost.

A dummy example can be seen at Figure 3.1, where the size of the samples represent their weights and the green lines the weak classifiers. The final classifier (bottom line) is a weighted combination of booth learned weak classifiers (top line).

AdaBoost proposes to learn iteratively the weak classifiers and their corresponding weights. At a given step t, the weak classifier $h_t(x)$ with lowest error rate, in a constrained time window, is found; then, you find its corresponding weight α_t; finally, samples weight is updated increasing those which were misclassified in order to emphasize them when learning the next weak classifier $h_{t+1}(x)$. The algorithm is stopped when either, the desired classification rate is reached or no more weak classifier can be learned because the defined maximum number has been attained.

At the first step, all samples have the same importance, so they all have the same weight:

$$w_s^1 = \frac{1}{n} \tag{3.3}$$

where s is the index of a sample and n the number of samples. Hence, the error rate for a particular step can be computed as the sum of the weights for the samples that were misclassified by the new learned weak classifier:

$$e_t = \sum_{i=1}^{N} w_i \left| h_t(x_i) - y_i \right|, \text{ where } N \text{ is the number of samples} \tag{3.4}$$

Once computed the error e_t of the current weak classifier $h_t(x)$, Freund and Schapire set α_t formula to ensure that the overall error for $H(x)$ will stay under an exponential bound, and eventually go to zero:

$$\alpha_t = -\frac{1}{2}\log\left(\frac{e_t}{1-e_t}\right) \tag{3.5}$$

If we analyze the formula for α_t, it can be easily seen that the weight given to a weak classifier is the inverse of the classification error. In other words, higher classification rate has a weak classifier, higher weight is assigned to. Weak classifiers with higher classification rate will have a higher contribution on the final strong classifiers

The last reaming step is to update the weights assigned to samples based on the new weak classifier $h_t(x)$. To do so there are two common approaches. The first one is to update the weights based on the classification rate of the new weak classifier,

$$w_s^{t+1} = \begin{cases} \frac{w_s^t}{N_t}e^{-\alpha_t}, & \text{if s correctly classified} \\ \frac{w_s^t}{N_t}e^{+\alpha_t}, & \text{if s misclassified} \end{cases} \tag{3.6}$$

The other approach is to understand Boost as a gradient descent method where the loss function to be minimized is $\sum_i e^{-y_i \alpha_t h_t(x_i)}$. The idea is to maximize the score of the positives samples and minimize the score of the negatives samples. With this approach the weights are updated as,

$$w_s^t = w_s^{t-1} e^{-y_s \alpha_t h_t(x_s)} \tag{3.7}$$

As a summary, at each iteration, choose $h_t(x)$ which minimizes $e_t = \sum_{i=1}^{N} w_i \left|h_t(x_i - y_i)\right|$. Then, set the weight of $h_t(x)$ to be $\alpha_t = -\frac{1}{2}\log\left(\frac{e_t}{1-e_t}\right)$ and update $H_t(x) = H_{t-1}(x) + \alpha_t h_t(x)$.

3.2 Weak classifiers. Decision Trees

Decision Trees [24] classifiers are tree-like graph models of decisions where nodes represent features and branches represent splits of the data based on the feature result. The most common weak classifiers used in face detection with AdaBoost Cascade are Decision Stumps. Decision Stumps are 1-rule decision models which predict a classification result based on one single feature. They can be seen as 1-level Decision Trees (see Figure 3.2).

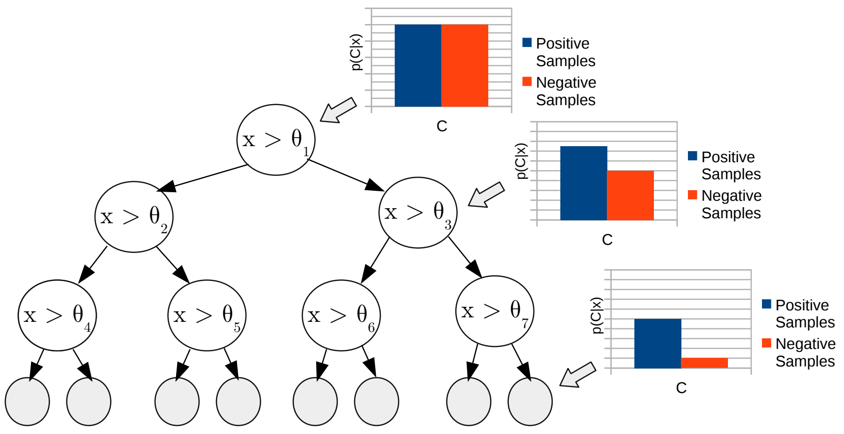


FIGURE 3.2: Example of a Decision Tree.

A given sample to be classified is inserted to the Decision Tree from the root node where it is tested with one feature. Based on the evaluation result, the sample goes down through the left or right branch. The node connected to the followed branch is then evaluated and the sample is again redirected left or right. This process is repeated until reaching a leaf with and assigned label to it. The sample is label with the same label as the one assigned to the leaf where it has fallen.

The construction of the tree consist on partitioning the training data in each leaf in a way that the classification error at each leave decreases. The feature that best splits the samples into the purest possible children nodes is chosen. Measuring how pure a subset of samples is the same as measuring the amount of information that would be needed to correctly label a given sample in that subset. This can be calculated as the difference of probabilities of the labels in the subset. However, the common approach to decide the feature to assign to a node is by calculating the opposite. Instead of calculate the purity, the impurity of the subsets is measured with the *entropy*. The entropy of a random subset is measured as:

$$H = -\sum_{y \in \mathbf{Y}} P(y) \log P(y) \tag{3.8}$$

where $\mathbf{Y}$ are all the possible class labels. The entropy for a binary set of samples containing p positives samples and n negative ones is:

$$H\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n} \log \frac{p}{p+n} - \frac{n}{p+n} \log \frac{n}{p+n} \tag{3.9}$$

The binary entropy function is shown in Figure 3.3. Its maximum value 1 is reached

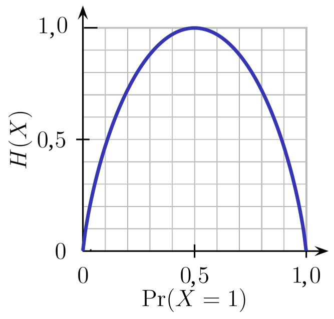


FIGURE 3.3: Binary entropy function

when there is the same probability of being of any of booth classes ($p(X = a) = 0.5$ or $p(X = b) = 0.5$). Opposite to that, its minimum value 0 is reached when the probability of being of one class is 1 ($p(X = a) = 1$ or $p(X = a) = 0$). Therefore, to minimize the classification rate at each level of the tree it is necessary that its reducing uncertainty of the prediction at each tree levels keeps decreasing. In other words, the entropy comparison between the parent node and its sons gives a measure of the information gained by doing the split using that particular feature. This can be expressed as:

$$I_j = H(S_j) - \sum_{i \in \{L,R\}} \frac{|S_j^i|}{|S_j|} H\left(S_j^i\right) \tag{3.10}$$

where S_j^L, S_j^R are the two sons of node S_j. At each node of the tree, the feature with highest information gain is selected. This process is applied recursively to its sons until one of the stop criterion is satisfied. There are multiple stop criterion such us reaching the maximum level of the tree or reaching a sub-state with null entropy.

To avoid overfeeding, once the tree has been constructed a trim step is performed. In this step, the nodes that do not improve the information gained or its contribution is very low are removed.

At each leaf of the tree there are set of p positives samples and n negatives samples which can be represented with a histogram (see Figure 3.2). Hence, the probability of a new sample that has fallen into that leaf of being positive is $P(x = pos) = \frac{p}{p+n}$ and being negative $P(x = neg) = \frac{n}{p+n}$.

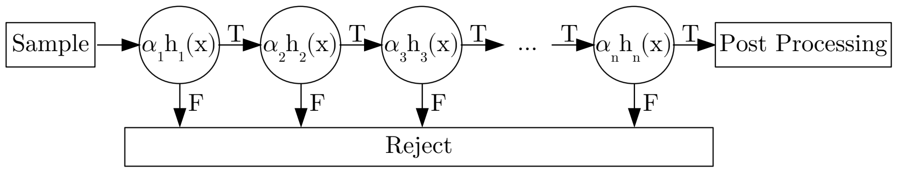


FIGURE 3.4: Cascade structure.

3.3 Soft Cascade

One of the principal problems when training classifiers is the trade-off between accuracy and speed. This problem becomes even bigger when applying boost techniques. Evaluating all the weak classifiers for a given sample produces, normally, higher results of precision but at a higher computational cost. A common way to solve this problem is by applying a cascade structure.

Cascade structures (see Figure 3.4) decomposes a strong classifier into a linear sequence of weak classifiers. For a given stage of the cascade, all the samples classified as positive for the current weak classifier are sent to the next stage. Samples classified as negatives are rejected. Only the samples which pass all the stages of the cascade are finally classified as positive.

The intuition behind this strategy is that there is a significant statistical dependency among the features. Hence, a lot of samples can clearly be rejected by the first weak classifiers without the need of evaluating all the rest.

There are two main problems with cascade structures. First, information obtained in one of the stages of the cascade is discarded as it passes to the next stage. Second, all true positive samples must be classified as positive by all the weak classifiers to be classified as positive.

These two problems are solved by Soft Cascade. Instead of having a sequence of binary stages, it proposes to have a sequence of stages each giving a score on how well a given sample passed that stage. At each stage of the cascade the cumulative sum of all the previous stages plus the current one is evaluated. If this value is higher than a training threshold the sample is sent to the next stage, otherwise it is discarded. The pseudo code of this rejection function is shown in Algorithm 1.

Algorithm 1 Soft Cascade rejection function

```
 1: procedure Soft Cascade rejection algorithm
 2:     $cum\_score \leftarrow 0$
 3:     for i=1 in T do
 4:         $cum\_score \leftarrow cums\_score + \alpha_t h_t(x)$
 5:         if $cum\_score < threshold_t$ then
 6:             $return - 1$
 7:         end if
 8:     end for
 9:     $return 1$
10: end procedure
```

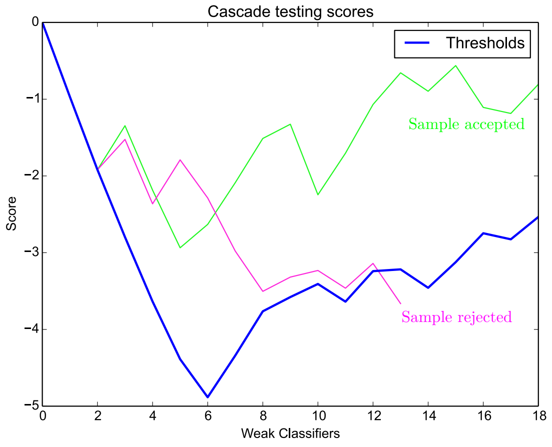


FIGURE 3.5: Cascade score trace

In this way samples are not prematurely rejected. Samples which did good in several weak classifiers but failed in one are not discarded because they accumulate a score high enough to keep them alive. Moreover, each weak classifier has a weight α assigned to it. Hence, samples with high score in the important classifiers are more likely to be finally classified as positive samples. A score trace is presented in Figure 3.5. As it can be seen, scores that at some point are lower than the rejection trace are rejected and automatically classified as negatives. Those that are not rejected by the cascade are considered positive.

Some modifications of the original AdaBoost algorithm must be performed to train the cascade. Now, training samples can be rejected by a training rejection function defined as:

$$\sum_{j=1}^{t} \alpha_j h_j(x) \geq \frac{1}{2} \sum_{j=1}^{t} \alpha_j \tag{3.11}$$

At a given stage t only samples with a cumulative score higher or equal than half of the sum of previous classifiers weights α_j are selected. By eliminating training samples,

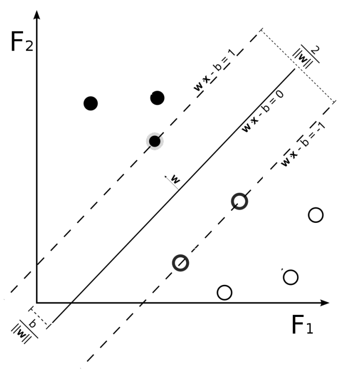


FIGURE 3.6: A Support Vector Machine.

more precise and specialized weak classifiers are learned as the cascade advances as they are only required to learn the subset of samples that the previous classifiers were not able to reject.

When deleting training samples, at each stage of the cascade classifiers have less training samples to manage. To solve this issue, bootstrap is introduced. Each time a new weak classifier must be trained K new negative samples are introduced. These negative samples must satisfy the condition that they are miss-classified by the current strong classifier $H(x)$. The number of samples K to be added is critic. If K is too large, AdaBoost will not be able to catch up and the error rate over the negative training set will be high. If K is too small, to achieve a good false positive to many weak classifiers would be required.

The last missing parameter to tune is how to decide the thresholds of the rejection function. To do so, a tuning stage is performed after all the cascade has been learned using new samples not previously seen by the classifier. Firstly, a maximum of positive samples that can be rejected at each stage is set. Then, for each stage all the new samples are evaluated and the threshold is set to be the maximum possible value which satisfy the number of positive samples to be rejected.

3.4 Support Vector Machines

Support vector machines [25] are one of the most popular supervised learning models for classification and regression analysis. SVM tries to find the optimal maximum margin hyperplane that best separate the sample classes (see Figure 3.6).

The hyperplane can be we written as the set of points $\mathbf{x}$ which satisfy the condition:

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{3.12}$$

where $\mathbf{w}$ is a vector perpendicular to the hyperplane. Therefore, given a sample expressed as the vector $\mathbf{x}$ it can be determined in which side of the hyperplane it is located by projecting $\mathbf{x}$ to $\mathbf{w}$. The decision rule is expressed as:

$$\widehat{y} = \begin{cases} +1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b \geq 0 \\ -1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b < 0 \end{cases} \tag{3.13}$$

The maximum-margin hyperplane is situated between the two parallel hyperplanes that separates the two classes (as it can be seen in figure 3.6). This two hyperplanes can be expressed as:

$$\begin{cases} \mathbf{w} \cdot \mathbf{x}_{+} + b \geq +1 \\ \mathbf{w} \cdot \mathbf{x}_{-} + b \geq -1 \end{cases} \tag{3.14}$$

Introducing the term y_i

$$\widehat{y} = \begin{cases} +1, & \text{if } i \text{ is a positive sample} \\ -1, & \text{if } i \text{ is a positive sample} \end{cases} \tag{3.15}$$

it results in:

$$\begin{cases} \mathbf{w} \cdot \mathbf{x}_{+} + b \geq +1 \Rightarrow y_i(\mathbf{w} \cdot \mathbf{x}_{+} + b) - 1 \geq 0 \\ \mathbf{w} \cdot \mathbf{x}_{-} + b \leq -1 \Rightarrow y_i(\mathbf{w} \cdot \mathbf{x}_{-} + b) - 1 \geq 0 \end{cases} \tag{3.16}$$

Now booth equations have the same structure. Hyperplanes fall into the support vectors, taht is those with constraint equal to zero,

$$y_i(\mathbf{w} \cdot \mathbf{x_i} + b) - 1 = 0 \tag{3.17}$$

The distance d to be maximized can be represented as

$$d = (\mathbf{x}_{+} - \mathbf{x}_{-}) \cdot \frac{\mathbf{w}}{\|\mathbf{w}\|} \tag{3.18}$$

where $\mathbf{x}_+, \mathbf{x}_-$ are two support vectors and $\mathbf{w}$ a vector perpendicular to the hyperplane. Then, by using Equation 3.15, Equation 3.18 can be transformed as,

$$(1 - b + 1 + b) \cdot \frac{\mathbf{w}}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|} \tag{3.19}$$

The SVM must achieve:

$$min\frac{1}{2}\left\|\mathbf{w}^2\right\| \tag{3.20}$$

To minimize Equation 3.20, Lagrange multipliers can be applied and obtain that:

$$maxL = \frac{1}{2}\sum \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \tag{3.21}$$

It is important to notice that because L defines a convex space SVM will never fall into local maxima, but a global one exist. This formulation works for linearly separable data. If data is not linearly separable, samples are transformed into another space which there can be separable.

Our samples are not linearly separable hence, a kernel transformation could be applied. However, only linear SVMs will be used in this woirk to be able to achieve the real time constraint.

Chapter 4

Proposed learning method

In this Chapter modifications for a Cascade of Weak Classifiers with AdaBoost are proposed, which will improve the classification results in the literature as well as to speed up the training and classification computational time.

4.1 Weak classifiers. Decision Tree

As previously seen, most face detectors use *boosted stumps* as weak classifiers. They present two main limitations. First, they are not capable of representing interaction between different feature dimensions. Second, the simple one-threshold splitting strategy can express a low amount of information. This section introduces a set of proposed modifications to solves this problems.

Depth of the Decison Trees

Most approaches in the literature achieve multiple-view detection by training one classifier for each view. Notice that a window with a face in it will only be correctly classified as a face if the classifier corresponding to the face view is used. To classify a given window there are three possible strategies: use all the classifiers, use only one or use a subset. Thus, the main problem with this approach is the trade-off between speed and classification rate during testing. In the literature, a pre-classification stage is added to chose which subset of classifiers should be used for each given windows. The problem with this strategy is that if the pre-classification stage fails to return a set containing the correct classifier the windows will be miss-classified.

Our proposed method does not need to have one classifier for each view, instead only one classifier for all the views is trained thanks to the tree structure. A tree structure allows to organize the evaluated windows as they go down the tree. Then, by increasing the depth of the tree its complexity also increases and, as a consequence, its able to better cluster the evaluated windows in its leafs. In the literature for face detection classifiers the common used height is only 1 (Decision Stumps). Instead, in this project the proposed Decision Trees are of height 5, that is Decision Trees having 5 levels of depth. This value has been tuned using cross-validation experiments. In this way we can express much more complex relations between features which allow a better the views than Decision Stumps.

Notice that the cross-validated height of the tree is much higher than the one used in the literature for face detection which are only of height 1 (decision stumps). Having Decision Trees with big depth makes them complex enough to be able to cluster the different views in its leafs. In other words, our Decision Trees are able to identify a face no matter to its orientation view thanks to their big depth. In this way, there is no need to have one classifier for each possible view, only with one classifier we are able to detect faces with unconstrained face orientation. Thus we do not need the pre-classification stage which could fail and we are also speeding up the face detector speed.

It is important to remark that the weak classifiers do not return the view of the face, only the measure of the certainty of the windows being a face. However we need the Decision Trees to be able to cluster the faces views. In this way when a windows to be classified goes down the tree it will be moved to the branch with the features that better evaluates its type of view.

Having Decision Trees with big depth also allows them to optimally combine several features together to represent a more robust face structure. For example, in the case of occlusions, the local features corresponding to the occluded part of the face will return that it is not a face. But, thanks to its tree structure the evaluated windows will be moved to other branches of the tree with local features situated in the parts of the face that are not covered. In this way, the windows will be correctly classified.

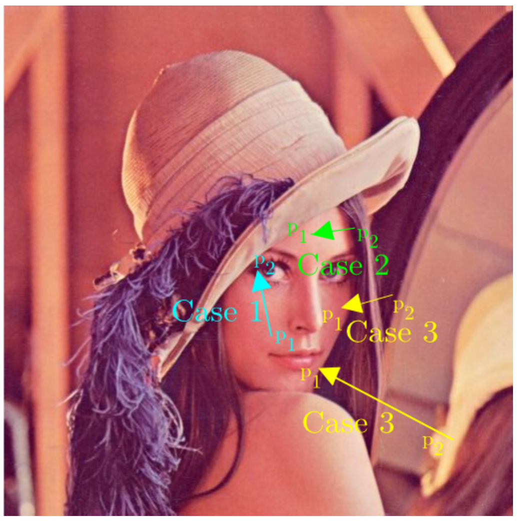


FIGURE 4.1: Three possible NPD cases to be represented.

Thresholds: Introducing quadratic thresholds

Common decision trees have only one threshold as the splitting decision strategy ($\Theta < f$). Therefore, only first order information contained in the features is taken into account. For the cases of features *NPD* and *Haar* only two structures can be learned:

$$f_{min} \leq f \leq \Theta < 0 \leq f_{max} \tag{4.1}$$

$$f_{min} \leq 0 < \Theta \leq f \leq f_{max} \tag{4.2}$$

Where f is the evaluated feature and f_{min} and f_{max} are the feature minimum and maximum possible values. For NPD $f = \frac{I(\mathbf{p_1})-I(\mathbf{p_2})}{I(\mathbf{p_1})+I(\mathbf{p_2})}$ where $I(\mathbf{p_1})$, $I(\mathbf{p_2})$ are two pixels intensity values. For Haar $f = K(\mathbf{p})$ where $K(\mathbf{p})$ is a Haar kernel evaluated with its left top corner at the pixel $\mathbf{p}$ of the sample.

To exemplify the formulas lets analyze the NPD feature $f = \frac{I(\mathbf{p_1})-I(\mathbf{p_2})}{I(\mathbf{p_1})+(\mathbf{p_2})}$. Equation 4.1 is translated as pixel $\mathbf{p_1}$ is notably darker than pixel $\mathbf{p_2}$. Equation 4.2 is the opposite, $\mathbf{p_2}$ notably darker than $\mathbf{p_1}$. The problem with this representation is that is not possible to express “high contrast with uncertain polarity”. It is very important that we represent this case because it is desired to be able to represent an edge between face and background no matter if the background is brighter or darker than the face.

The only way to represent all three cases (see Figure 4.1) is to use a quadratic splitting strategy:

$$af^2 + bf + c < \Theta, \text{ where f is the feature} \tag{4.3}$$

In this way, the feature space is divided by a parabola instead of a line. In a practical way, instead of learning a,b,c the two second order threshold $f \in [\Theta_1, \Theta_2]$ are learned:

$$f_{min} \leq \Theta_1 \leq f \leq \Theta_2 \leq f_{max} \tag{4.4}$$

This can also be applied to Haar features.

To find the optimal Θ_1 and Θ_2, for each possible feature dimension we compute a histogram of N bins representing the feature value and the weight of the training sample as voting value. Then, over the histograms we perform an exhaustive search optimizing the splitting criterion between the training samples falling inside vs outside of the regions defined by the thresholds.

Classification Result: Measure of the certainty of the windows label

At the end of the face training process, all leafs of the Decision Tree have a set of positive and negative training samples that have fallen into them. It can be seen as a set of histograms of booth positive and negative training samples each representing one of the leafs of each tree. Commonly, for a given windows to be classified, a Decision Tree returns as the classification result the label with more frequency at the node where it has fallen. Another common return value is the probability of the positives training samples at the fallen node. In our implementation the returned value is a measure of the certainty of the windows label, where $h(x) \approx 1$ represents being very sure that the windows is a face, $h(x) \approx -1$ that it is not a face and $h(x) \approx 0$ random guess. Which is expressed as the difference between the weighted probability of the windows of being positive and being negative.

$$h(R) = \frac{w_+ - w_-}{w_+ + w_-} \in [-1, 1] \tag{4.5}$$

where R is the region of interest (roi, windows) evaluated and w_+, w_- are the weights of the positive and negative training samples that fall during training into the leaf where the windows to be evaluated has fallen.

Classification Error as Splitting Criterion

To train the threshold of the Decision Tree (splitting criterion) *entropy* is normally used. This function is evaluated millions of times: for each possible configuration $\{\Theta_1, \Theta_2\}$ of each possible feature dimension of each node of each tree. Hence this function must be very fast and highly optimized.

To improve the training velocity, the criterion function was changed from *entropy* to *MSE* (Mean Square Error). Where the MSE of a node S_j is the sum of the MSE of its left son S_j^L and right son S_j^R:

$$MSE(S_j) = \sum_{i \in \{L,R\}} MSE(S_j^i) \tag{4.6}$$

where the MSE of a son node S_j^i is computed as:

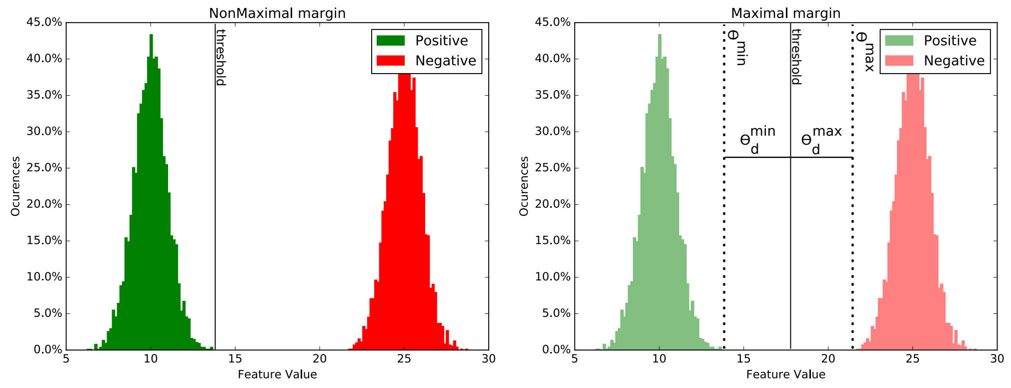
$$MSE(S_j^i) = w_+(h(S_j^i) - 1)^2 + w_-(h(S_j^i) + 1)^2 \tag{4.7}$$

where w_+,w_- are the positive and negative weights of the training samples falling to the son node S_j^i.

Computing this splitting criterion is 6 times faster than computing the *Information Gain*. With this change, the total training time was reduced almost 30% and the classification rate over the validation set only decreased $\sim -2.3\%$. This is a very important improvement because training time is a critical factor when cross-validating.

Approximate Maximum Margin

The common Decision Tree splitting criterion is given as the result of an optimization problem to find just the maximum *information gain*, equivalently, in our proposed method, finding the minimum *mean square error*. This strategy behaves poorly for flat regions of global minima. It is illustrated with the simple example in Figure 4.2. This behaviour is not desired. By setting the threshold close to one of the training samples cluster, the model is becoming very sensible to noise. Small changes produced by noise of that cluster will easily cross the threshold and as a consequence be misclassified.



(A) Without maximal margin. (B) With maximal margin.

Figure 4.2: Approximate maximal margin.

To solve this problem, and being inspired by an SVM, we propose to change the decision tree splitting criterion so that it tries to maximize the margin. To do so, we introduce to the optimization problem a constraint to maximize the distance between connected global minimums,

$$\begin{aligned} \underset{\Theta_1,\Theta_2}{\text{minimize}} \quad & e(\Theta_1,\Theta_2) = \sum_{i\in\{L,R\}} MSE(S_j^i) \\ \text{subject to} \quad & \Theta_{1d}^{max} = \Theta_{1d}^{min}, \Theta_{2d}^{max} = \Theta_{2d}^{min} \end{aligned} \tag{4.8}$$

where S_j^L, S_j^R are the two sons of node S_j and Θ_d^{max}, Θ_d^{min} are the distances from the threshold to the maximum and minimum connected global minimums positions in the 1-dimensional solution space (see Figure 4.2).

4.2 Soft Cascade

As seen in Section 3.3, traditional Soft Cascade is designed in two stages. First, train the model with AdaBoost. Second, calibration of the model. In the tuning step a new set of calibration samples not previously seen by the classifier is used to find each weak classifier threshold. This approach has two problems. First, it needs a big dataset (training+calibration+validation+test). Second, the training samples are not being rejected by the learned cascade thresholds which causes that the n-th weak classifier is being trained with samples that the weak classifiers $[1, n-1]$ were able to reject. Thus, the learned weak classifiers are not specialized but general. In this section is presented our proposed solution to solve these problems and new extensions to make it more robust.

Tuning Thresholds in the Training Stage

It is complicated to generate a big and rich dataset of multiple-view faces for training. It is really difficult and time consuming to get a large number of samples of some of the views. It is not desirable to require an extra set of instances just to calibrate the cascade thresholds. Thus, we propose that the thresholds are learned during the training phase, after each weak classifier is trained, so just using the training set.

This change would normally decrease the classifier classification rate. However, it is compensated by improving the algorithm to chose the thresholds using SVM (presented in the next subsection). Moreover, by peaking the thresholds on line, the training samples moving to the next weak classifier to be trained are better selected. Those whose cumulative score is lower than the learned threshold are not used to train the new one. In this way, the new weak classifier is only trained with the kind of training samples that the previous weak classifiers were not able to properly classify. Thus, the classifiers are more specialized and can better filter specific regions of the feature space of faces vs not faces that its predecessors did not explore/specialize on.

Choose cascade thresholds with SVM

As previously explained in Section 3.3, the common strategy to select the thresholds is to set the maximum accepted false negatives training samples to be rejected at each step and set the threshold as the maximum possible value that satisfies the constraint. The decision of the threshold values have a direct critical implication to the classification rate. Thus, this standard greedy algorithm is too simple and we propose to set it using machine learning.

Machine learning is very good to make simple decisions based on a lot of rich data. Hence, instead of choosing the threshold values with a greedy algorithm we use a one-dimensional linear SVM. To find the threshold θ_t of the current stage t, the current positive and negative sample scores are introduced to the SVM as the one-dimensional feature training samples of booth labels positive and negative. Then the SVM is trained and from the learned model we extract the cascade threshold θ as:

$$\theta = -\frac{b}{w} \tag{4.9}$$

Notice that w is a number and not a vector because it is a one dimensional SVM, b is the bias. The idea with a cascade of classifiers at a given stage t is to try not to reject any windows being a face and hope that the false positive will be discarded by the following stages $[t + 1, T]$ (T being the total number of classifiers). During testing the threshold should let pass almost all positive samples. To achieve this behaviour the SVM is learned imposing a much higher weight value to the positive training samples.

During training do not reject samples close to the threshold

Changes of position and illumination of a sample may change the score of that sample. Thus, at a given stage, a testing samples close to the threshold may sometimes be discarded and sometimes not because of small changes of position, orientation and illumination. To make the cascade more robust to this case, during training the 5% of the training samples under the threshold are not rejected. In this way, the next learned weak classifiers will be more robust to these training samples. Then during testing, if testing samples located in the same region of the feature space pass the current stage due to noise, the next weak classifier will have been trained to deal with this kind of samples.

4.3 AdaBoost

Due to the proposed changes of the weak classifiers and the cascade, new optimizations can be introduced to the AdaBoost algorithm to improve the classification rate.

Not Thresholded Weak Classifiers

As seen in Section 3.1, common AdaBoost approaches use weak classifiers in the from $h(x) \in \{\pm 1\}$,

$$h_t(R) = \begin{cases} +1, & \text{if } f \geq \theta \\ -1, & \text{otherwise} \end{cases} \tag{4.10}$$

where R is the region of interest (roi, windows). In this form the classification score is given by α_t. Our weak classifiers are of the form $h(R) \in [-1, +1]$ where its value represent the level certainty of the classified label. By thresholding the weak classifier value we would be losing valuable information. Therefore, our weak classifiers maintain the range $h(R) \in [-1, +1]$.

Weights Depend on the Threshold

By having the thresholds computed online during the learning phase also helps to improve the training samples weight update at each boost iteration. Let us first analyze the current state-of-the-art AdaBoost loss function:

$$L(H) = \sum_{s=1}^{N} e^{-m_s} \tag{4.11}$$

with

$$m_s = y_s \sum_{t=1}^{T} \alpha_t h_t(X_s) \tag{4.12}$$

where N is the number of training samples, T the number of weak classifiers, m_s the voting margin and X_s the sample. What it does is to maximize positive scores while minimizing at the same time the negatives scores. By having this score behaviour, negative windows will fall under the threshold and as a consequence, rejected.

However, on simple example it can be seen that miss-performs in simple scenarios (see Figure 4.3). Imagine two samples of the training set. During the training process, the negative sample has an absolute value score lower than the positive sample at all the weak classifiers. Thus, given the equation to update the weights derived from the traditional loss function is $w_s^t = w_s^{t-1} e^{-y_s \alpha_t h_t(x_s)}$ the negative sample had a higher weight at each weak classifier. The problem appears when in the calibration stage the chosen threshold happens to be closer to the positive sample rather than to the negative sample due to the rest of the dataset. The weight of the positive sample should have been higher than the negative one on all the weak classifiers, but instead the opposite was done. In other words, during the training stage all weak classifiers have been focusing on the wrong training sample.

To solve this problem we propose to use the fact that the proposed method learns the thresholds on line during the training phase. Instead of just trying to maximizing/minimizing the scores, add the constraint that the maximization/minimization of the scores

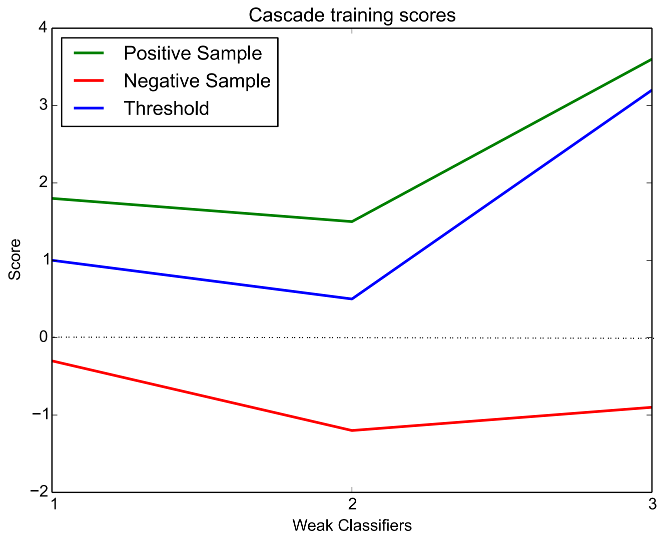


FIGURE 4.3: Problem with traditional AdaBoost loss function.

must be relative to the threshold. To express this the voting margin is updated to:

$$m_s = y_s(\sum_{t=1}^{T} \alpha_t h_t(x_s) - \theta_t) \tag{4.13}$$

and as a consequence the equation to update the training samples wights its also updated to

$$w_s^t = e^{-y_s(\sum_{i=1}^{t} \alpha_i h_i(x_s) - \theta_t)} \tag{4.14}$$

which can be rewritten as

$$w_s^t = \frac{w_s^{t-1} e^{-y_s(\alpha_t h_t(x_s) - \Delta\theta)}}{N} \tag{4.15}$$

Where $\Delta\theta = \theta_t - \theta_{t-1}$ and N is a normalization factor to ensure $\sum_s^S w_s^t = 1$.

4.4 Strong Classifier as a Final Filter

Inspired by the *Joint Cascade* [16] method, a post-processed classifier to filter the most challenging windows has been added as the post-processing stage of the cascade structure (see Figure 3.4). This posterior strong classifier is a learned SVM for each one of the face viewpoints with our own vectorized version of HoG (histogram of oriented gradients) with SIMD instructions (see Section 4.5). The learned SVM is linear to be able to maintain the real time factor.

This final filter is just *temporal*, we are currently working in finding a better one. The problem with this filter is that it requires labeling of the clusters viewpoints and this is not desired for our learning method. However, in experimentation (se Section 7.6) it has been shown that only 1% of the windows require the execution of this filter.

4.5 Modern and Highly Optimized Code with SIMD Instructions

Modern CPU include SIMD instructions to accelerate 3D graphics and audio/video processing in multimedia applications. These instructions allow parallelism at a data level by allowing to perform the same operation to multiple data in a single machine instruction. Notice that this is not concurrency. It is parallel because multiple computations are being done in parallel but only in a single process.

For example, let's suppose a piece of code for a sequence of array calculations:

```
float a[8], b[8], c[8];
//... put values into arrays
for (int i = 0; i < 10; i++) {
   c[i] = a[i] + b[i]*1.5f;
}
```

This code can be vectorized with SIMD instructions:

```
Vec8f avec, bvec, cvec;
//... put values into vectors
cvec = avec + bvec * 1.5f;
```

`Vec8f` vectors are of type `__m256` which represents a 256-bit vector register holding 8 floating point numbers of 32 bits each. The first code will generate approximately 44 instructions depending on the compiler. However, the vectorized code are only 2 machine instructions with AVX instructions and 4 if the set instructions are SSE2. The difference is caused because the the maximum vector register size of SSE2 instructions is half as big of AVX.

With this simple code optimization we achieved a speedup of 22 for AVX instruction and of 11 for SSE2 instructions. This optimization plus applying other modern C++ code techniques where one of the main reason that made the detector run at 2ms during testing with images of 1080p.

Chapter 5

Features

In this chapter a theoretical and computational description of the implemented features is presented.

5.1 Haar-like features

Haar-like features are the most common used features for face detection. They are a set of local appearance kernel each representing a pair of two-dimensional Haar functions. The intuition is to have a set of rectangles-composed kernel describing the difference of pixels intensity between the rectangles in a given position and scale of the region of interest.

Many different kernels have been proposed in literature [1, 26, 27]. The kernel configurations are not random at all. Each configuration is defined with the purpose of describing different properties of a region of interest such us edges, lines and surroundings. An example of these descriptions is presented in Figure 5.1. For rotated face, we have also introduced 45 degrees rotated kernels [27].

Each kernel is composed by K rectangular shaped areas with an assigned weight each. Its value is computed as:

$$f = \sum_{k=1}^{K} w_k \sum_{i=1}^{N} \sum_{j=1}^{M} I(i,j) \tag{5.1}$$

where w_k is the weight of the k^{th} rectangle of dimensions $N \times M$ and $I(i,j)$ the intensity value of the (i,j) pixel. Traditionally weight must satisfy $\sum_{k=1}^{K} w_k = 0$.

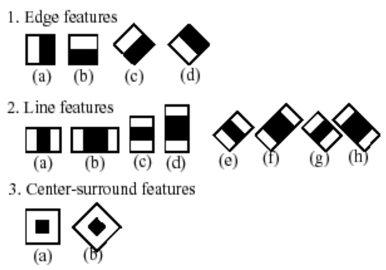


FIGURE 5.1: Haar-like kernels feature description.

Algorithm 2 Integral Image.

```
Require: Original Image of $N \times M$ size
Ensure: Integral Image of $N \times M$ size
1: procedure GENERATEINTEGRALIMAGE(I)
2:     $II(1,1) := I(1,1)$;
3:     for i=1 in $N$ do
4:         for j=1 in $M$ do
5:             $II(i,j) := I(i,j) + II(i,j-1) + II(i-1,j) - II(i-1,j-1)$;
6:         end for
7:     end for
8: end procedure
```

To evaluate f applying directly the presented Equation 5.1 has a high computational cost because the intensity sum of overlapping areas is recomputed multiple times by different kernel. In practice, instead of applying Equation 5.1 an *Integral Image* is calculated and used to rapidly evaluate f at any scale and position with $O(1)$ complexity. It is computed as,

$$II(i,j) = \begin{cases} \sum_{s=1}^{i} \sum_{t=1}^{j} I(s,t), & \text{if } 1 \leq i \leq N \text{ and } 1 \leq j \leq M \\ 0, & \text{otherwise} \end{cases} \tag{5.2}$$

where N and M are the number of rows and columns of the original image I respectively. The *Integral Image* has been computed with the Algorithm 2 with linear time $O(N \times M)$.

By using Integral Image, querying the sum of intensity pixels in a rectangular area can be performed with $O(1)$ complexity as,

$$I(p_1, p_2) = II(p_1.y, p_1.x) - II(p_1.y, p_2.x) - II(p_2.x, p_1.y) + II(p_2.y, p_2.x) \tag{5.3}$$

Where $P1$ represent the top left point and $P2$ the bottom right point of the rectangular region. For a fast computation of the rotated kernels, a rotated *Integral Image* is also generated.

In this project it was NOT used a predefined set of kernels. Instead, a total number of 78900 kernels were generated and then accepted or rejected by applying *boost*. To generate these kernels a set of 14 templates were deformed by width and height and set in all possible positions that feet in the region of interest.

5.2 Normalized Pixel Difference

The Normalized Pixel Difference (NPD) is a very simple but yet elegant local feature based on comparison of pixel intensities. It is value is computed as the normalized difference of pixel intensity between two points (see Figure 5.2),

$$f(p_1, p_2) = \frac{I(p_1) - I(p_2)}{I(p_1) + I(p_2)} \tag{5.4}$$

where I is the intensity value and p_1 and p_2 two given pixels. For grey-scale images of $I \in [0, 255]$ NPD is a bounded nonlinear function $f(p_1, p_2) \in [-1, 1]$. If we analyze the formula that what this feature is returning a vector which its module is the percentage of joint intensity $I(p_1) - I(p_2)$ and its sense indicates the ordinal relationship between booth pixels p_1, p_2.

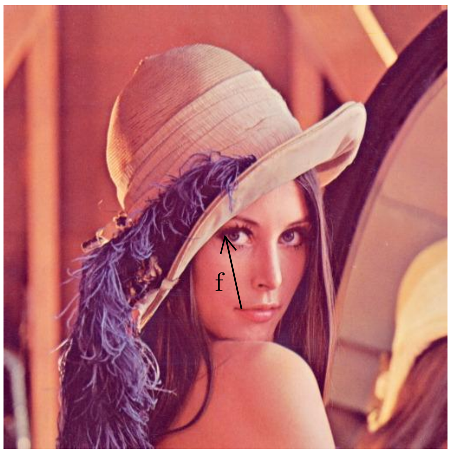


FIGURE 5.2: Normalized pixel difference representation.

NPD is a very elegant feature because despite of its apparent simplicity it has a lot of desirable properties. First, the vector module is symmetric, for booth $I(p_1) - I(p_2)$ and $I(p_2) - I(p_1)$ its value is the same, only its sense changes. Hence, the future space is reduced to $\binom{N \times M}{2}$ dimension in comparison with most other difference of pixels features which have a feature space of $\frac{(N\times M)!}{(N\times M-2)!}$ where $N \times M$ are the number of pixels of the region of interest.

Second, the sign of the NPD (its vector sense) indicates the ordinal relationship between booth evaluated pixels. In literature [28, 29] it has been shown that ordinal relationship is a good indicator for object detection because it describes the intrinsic structure, in other words, it describes a given part of an object with respect to the others. However, this property is highly sensitive to noise for pixels of similar intensity. For example, being $p_1 = 6, p_2 = 7$ the feature value is easy to change between positive and negative signs due to small changes in illumination. This issue has been solved by introducing second order thresholds in the decision tree. This is explained with further detail in Section 4.1.

Third, invariant to scale changes of pixels intensity. For features with this simplicity it Is practically impossible to find a feature which is invariant to any light variation in the region of interest. However, by being scale invariant to pixels intensity it is expected to have some level of robustness in front of changes of illumination.

Fourth, its previously explained $[-1, 1]$ bound makes it perfectly fitted for threshold and histogram based learning methods such us decision trees which are the selected weak classifier representation.

For a fast computation, $T_{255x255}$ look-up table is precomputed, where rows represent all possible values of $I(p_1)$ and columns all possible values of $I(p_2)$. Hence, given a couple of pixels, its NPD value can be queried with complexity $O(1)$ as $T(p_1, p_2)$. Although having the same computational complexity as a Haar feature, only one query to a matrix is required vs 4 queries for HAAR.

5.3 Histogram of Oriented Gradients

Histogram of oriented gradients (HoG) is a commonly used feature for object detection. It describes the shape of an object as a distribution of its intensity gradients. The image is divided into small spatial regions called cells each containing $n \times n$ pixels. For each cell a histogram of its pixels gradient directions is computed and normalized to make it more robust to illumination and shadowing. The final descriptor is the concatenation of the normalized histograms of each block (group of cells).

First, the horizontal $\partial_x(x, y)$ and vertical $\partial_x(x, y)$ image gradients are computed. To do so, the *derivative masks* is applied over the image in the vertical and horizontal direction.

$$[-1, 0, 1] \text{ and } [-1, 0, 1]^\top \tag{5.5}$$

Second, for each pixel in the image its gradient magnitude and orientation are computed.

$$m(x_i, y_i) = \sqrt{\partial_x(x_i, y_i)^2 + \partial_y(x_i, y_i)^2} \tag{5.6}$$

$$\Theta(x_i, y_i) = arctan\left(\frac{\partial_y(x_i, y_i)}{\partial_x(x_i, y_i)}\right) \tag{5.7}$$

Then each cell histogram is generated. Histograms x coordinate represent the angle of the gradients which can be unsigned ($[0°, 180°]$) or signed ($[0°, 360°]$)s). Each pixel within the cell falls into a bin of the histogram based on its gradient orientation and casts a weighted vote for that orientation equal to its gradient magnitude.

To improve robustness against illumination and shadow changes a normalization factor is applied to large spatial regions. In other words, normalization is applied over groups of cells named blocks which overlap so each cell contributes to multiple blocks. There are different possible block normalization techniques, the one used consists on computing the L2-norm (Equation 5.8) and then clipping the values and re-normalize,

$$\frac{h}{\sqrt{\|h\|_2^2 + c^2}} \tag{5.8}$$

where v represents the histogram and c a constant. The final descriptor is the concatenation of the normalized histograms.

Chapter 6

Temporary planning

In this Chapter the definition of tasks and their temporal scheduling plan are defined.

6.1 Tasks identification

The development of this project will be performed using *Agile* methodologies. Hence, the project is divided into a set of tasks (iterations) each one divided in analysis, implementation, testing, integration and documentation. The application of agile methodologies will allow a more close up feedback relationship with the project supervisors as well as a faster response against obstacles and incidents.

6.1.1 State of the art and learning

In this first task an evaluation of the current state of the art will be analyzed. The objective of this task is to identify the most promising state of the art training algorithms, features and classifiers which do/could fulfill the established detector constraints. Once identified, study all the necessary theory in order to understand and be able to implement the method.

6.1.2 Debugging dataset generation

Search and filter an on line frontal face dataset to be used for initial debugging of the classifier, features and learning algorithm.

6.1.3 Learning classifier

Once decided in the *State of the art* this task states the classifier to be used. Its implementation must be highly modular and reusable because different features and learning algorithm would be tested on it. Also, it is not desired to implement it with low level optimizations as multiple version and changes could be made in future iterations. The classifier must be as transparent as possible. To achieve this transparency a set of scripts to plot the current state of the classifier will be implemented.

6.1.4 Features

Implement, test and evaluate all identified features that could be used. All features must have the same interface for modular and reusable purposes. In this way, changing between features does not affect any of the other modules.

6.1.5 Learning algorithm

Implementation and testing of the learning algorithm to train the classifier. Learning algorithms tend to be a black box. It is desired to make this step as transparent as possible. To do so, a set of scripts to plot the state of each step of the learning process must be implemented.

6.1.6 Final dataset generation

Once the classifier is properly being trained, a rich, well filtered dataset will be generated. It is intended to implement a script to scratch the web, downloading face pictures and then implementing a program to fast label each sample. The labeling will consist of defining the face bounding box and position configuration.

6.1.7 Classifier training and validation

In this step the classifier will have to be implemented by cross-validating all the critical parameters.

6.1.8 Classifier implementation optimization

Once the classifier and the features to be used have been decided, highly optimize the code. Run a code profiler to identify the most critical parts of the code an apply modern low-level compiler optimizations.

6.1.9 Final documentation

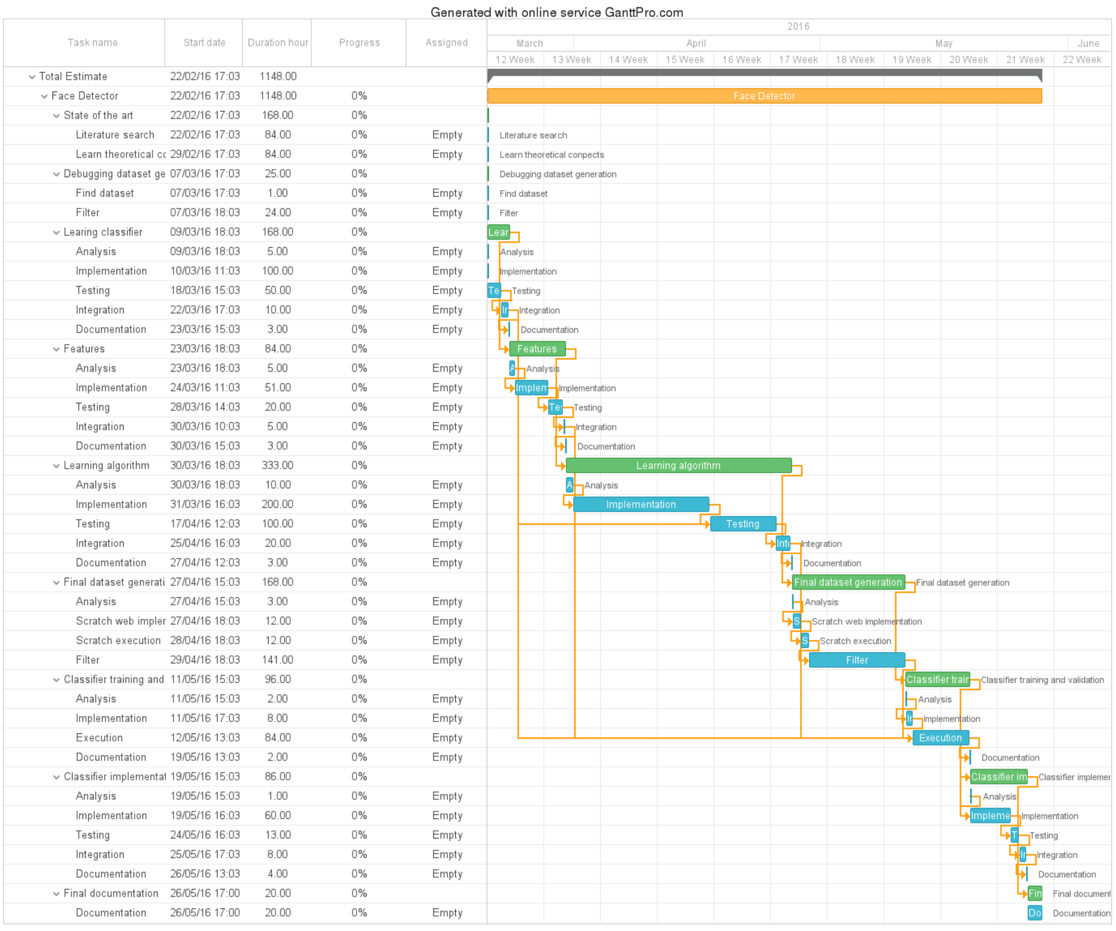
At each iteration the documentation of that task will be generated. Hence, in this final iteration only final polishes over the documentation will be made.

6.2 Time estimation

In Figure 6.1 a time estimation of each task/iteration is presented.

Task Time Estimation	
Task/Iteration	Time [h]
State of the art and learning	168
Debugging dataset generation	25
Learning classifier	168
Features	168
Learning algorithm	420
Final dataset generation	24
Classifier training and validation	96
Classifier implementation optimization	213
Final documentation	20
TOTAL	**1289**

TABLE 6.1: Time estimation table of the initial planning.


Generated with online service GanttPro.com

Task name	Start date	Duration hour	Progress	Assigned
Total Estimate	22/02/16 17:03	1148.00		
Face Detector	22/02/16 17:03	1148.00	0%	
State of the art	22/02/16 17:03	168.00	0%	
Literature search	22/02/16 17:03	84.00	0%	Empty
Learn theoretical cc	29/02/16 17:03	84.00	0%	Empty
Debugging dataset ge	07/03/16 17:03	25.00	0%	
Find dataset	07/03/16 17:03	1.00	0%	Empty
Filter	07/03/16 18:03	24.00	0%	Empty
Learing classifier	09/03/16 18:03	168.00	0%	
Analysis	09/03/16 18:03	5.00	0%	Empty
Implementation	10/03/16 11:03	100.00	0%	Empty
Testing	18/03/16 15:03	50.00	0%	Empty
Integration	22/03/16 17:03	10.00	0%	Empty
Documentation	23/03/16 15:03	3.00	0%	Empty
Features	23/03/16 18:03	84.00	0%	
Analysis	23/03/16 18:03	5.00	0%	Empty
Implementation	24/03/16 11:03	51.00	0%	Empty
Testing	28/03/16 14:03	20.00	0%	Empty
Integration	30/03/16 10:03	5.00	0%	Empty
Documentation	30/03/16 15:03	3.00	0%	Empty
Learning algorithm	30/03/16 18:03	333.00	0%	
Analysis	30/03/16 18:03	10.00	0%	Empty
Implementation	31/03/16 16:03	200.00	0%	Empty
Testing	17/04/16 12:03	100.00	0%	Empty
Integration	25/04/16 16:03	20.00	0%	Empty
Documentation	27/04/16 12:03	3.00	0%	Empty
Final dataset generati	27/04/16 15:03	168.00	0%	
Analysis	27/04/16 15:03	3.00	0%	Empty
Scratch web impler	27/04/16 18:03	12.00	0%	Empty
Scratch execution	28/04/16 18:03	12.00	0%	Empty
Filter	29/04/16 18:03	141.00	0%	Empty
Classifier training and	11/05/16 15:03	96.00	0%	
Analysis	11/05/16 15:03	2.00	0%	Empty
Implementation	11/05/16 17:03	8.00	0%	Empty
Execution	12/05/16 13:03	84.00	0%	Empty
Documentation	19/05/16 13:03	2.00	0%	Empty
Classifier implementat	19/05/16 15:03	86.00	0%	
Analysis	19/05/16 15:03	1.00	0%	Empty
Implementation	19/05/16 16:03	60.00	0%	Empty
Testing	24/05/16 16:03	13.00	0%	Empty
Integration	25/05/16 17:03	8.00	0%	Empty
Documentation	26/05/16 13:03	4.00	0%	Empty
Final documentation	26/05/16 17:00	20.00	0%	
Documentation	26/05/16 17:00	20.00	0%	Empty

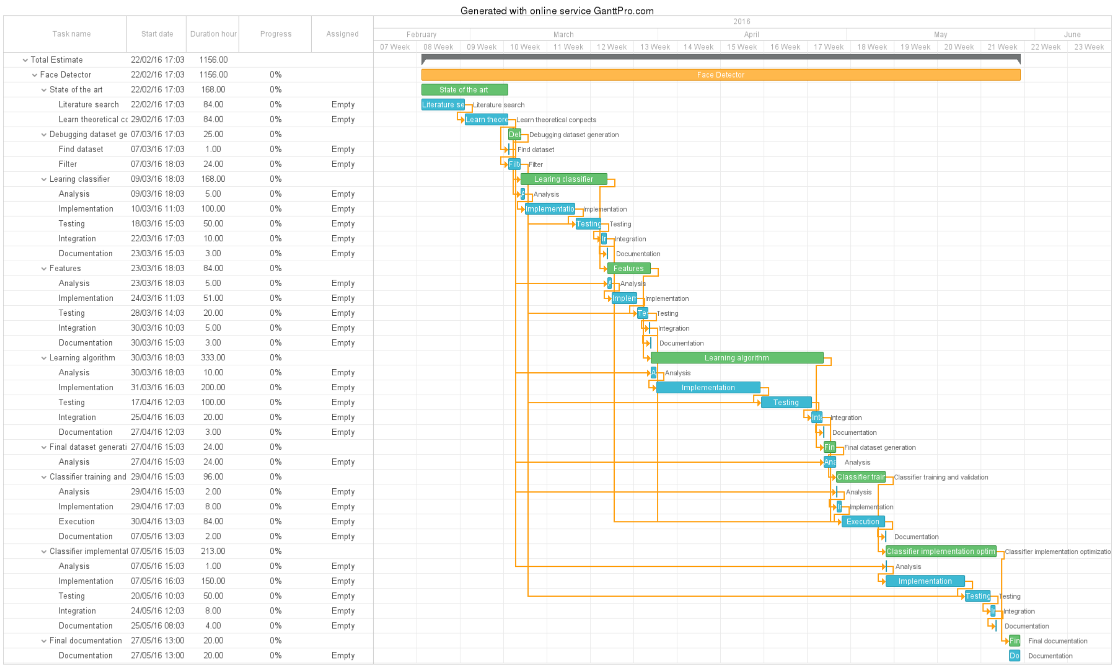
FIGURE 6.1: Gannt diagram initial planning

6.3 Initial planing

At CMU a researcher works 12 hours and 7 days a week. As a Visiting Scholar this timetable is also applied to me. Hence, all the planning has been calculated with an estimated 84 hours of full dedication to this project every week. The visiting period is 22 weeks. A Gantt planning diagram can be seen in Figure 6.1.

6.4 Final planning

In the initial planning it was defined that only 48 hours would be necessary to generate the dataset of samples. This estimation was based on the supposition that the laboratory would proportionate it. Then, it would only be necessary to clean it and adapt it to our necessities. However, the laboratory did not have any dataset that fitted our needs. It was necessary to hand-label 200K faces of multiple viewpoints which took one week.


Generated with online service GanttPro.com

Task name	Start date	Duration hour	Progress	Assigned
Total Estimate	22/02/16 17:03	1156.00		
Face Detector	22/02/16 17:03	1156.00	0%	
State of the art	22/02/16 17:03	168.00	0%	
Literature search	22/02/16 17:03	84.00	0%	Empty
Learn theoretical cc	29/02/16 17:03	84.00	0%	Empty
Debugging dataset ge	07/03/16 17:03	25.00	0%	
Find dataset	07/03/16 17:03	1.00	0%	Empty
Filter	07/03/16 18:03	24.00	0%	Empty
Learing classifier	09/03/16 18:03	168.00	0%	
Analysis	09/03/16 18:03	5.00	0%	Empty
Implementation	10/03/16 11:03	100.00	0%	Empty
Testing	18/03/16 15:03	50.00	0%	Empty
Integration	22/03/16 17:03	10.00	0%	Empty
Documentation	23/03/16 15:03	3.00	0%	Empty
Features	23/03/16 18:03	84.00	0%	
Analysis	23/03/16 18:03	5.00	0%	Empty
Implementation	24/03/16 11:03	51.00	0%	Empty
Testing	28/03/16 14:03	20.00	0%	Empty
Integration	30/03/16 10:03	5.00	0%	Empty
Documentation	30/03/16 15:03	3.00	0%	Empty
Learning algorithm	30/03/16 18:03	333.00	0%	
Analysis	30/03/16 18:03	10.00	0%	Empty
Implementation	31/03/16 16:03	200.00	0%	Empty
Testing	17/04/16 12:03	100.00	0%	Empty
Integration	25/04/16 16:03	20.00	0%	Empty
Documentation	27/04/16 12:03	3.00	0%	Empty
Final dataset generati	27/04/16 15:03	24.00	0%	
Analysis	27/04/16 15:03	24.00	0%	Empty
Classifier training and	29/04/16 15:03	96.00	0%	
Analysis	29/04/16 15:03	2.00	0%	Empty
Implementation	29/04/16 17:03	8.00	0%	Empty
Execution	30/04/16 13:03	84.00	0%	Empty
Documentation	07/05/16 13:03	2.00	0%	Empty
Classifier implementat	07/05/16 15:03	213.00	0%	
Analysis	07/05/16 15:03	1.00	0%	Empty
Implementation	07/05/16 16:03	150.00	0%	Empty
Testing	20/05/16 10:03	50.00	0%	Empty
Integration	24/05/16 12:03	8.00	0%	Empty
Documentation	25/05/16 08:03	4.00	0%	Empty
Final documentation	27/05/16 13:00	20.00	0%	
Documentation	27/05/16 13:00	20.00	0%	Empty

FIGURE 6.2: Gannt diagram final planning.

As previously explained, this project has been managed with agile methodologies which allow to check and adapt dynamically the initial planning. Thus, a modification of the initial planning was made. Hours of optimization where moved to generate the dataset. With this new planning the number of code optimizations and tricks decreased but all the initial objectives where achieved. The new gantt diagram is presented in Figure 6.2.

Task Time Estimation	
Task/Iteration	Time [h]
State of the art and learning	168
Debugging dataset generation	25
Learning classifier	168
Features	168
Learning algorithm	420
Final dataset generation	168
Classifier training and validation	84
Classifier implementation optimization	84
Final documentation	20
TOTAL	**1285**

TABLE 6.2: Time estimation table final planning.

Chapter 7

Experimentation and results

Every time a new model is learned with new updates and upgrades a battery of tests are executed to validate the model. Moreover, this unit tests generate a sequence of plots to try to transform the black box classifier into a white box classifier. In this chapter, some of these plots are presented.

All these experiments have been performed with the same model configuration which used NPD features. The one with better classification rate found so far. The learning algorithms take 20 minutes to learn the model in a 20 cores Intel Corporation Xeon E7 v2/Xeon E5 v2/Core 3GHz sever without GPU.

The model has been trained with a dataset of 101241 positive samples and 305203 negative ones. 80% of the samples have been used to cross-validate and 20% to test. There are more negative samples than positives because of bootstrap.

7.1 Proof of concept - basic tests

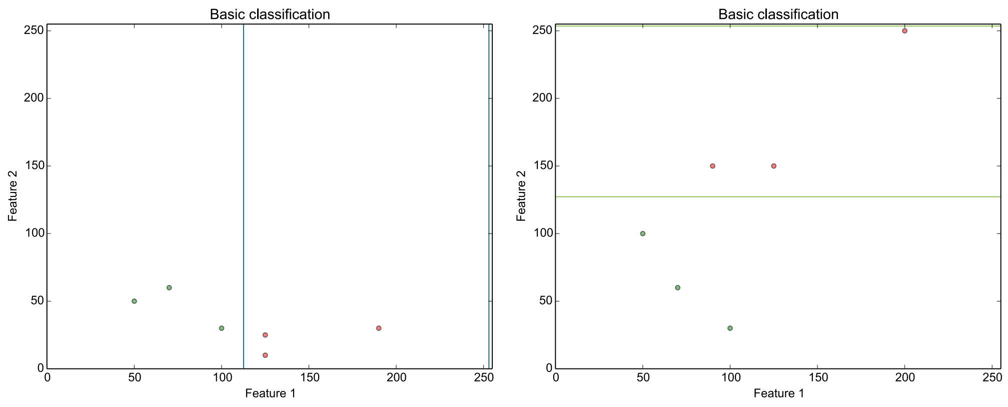


FIGURE 7.1: Two linearly separable basic test with two feature $f \in [0, 255]$. The radius of each sample represents its weight.

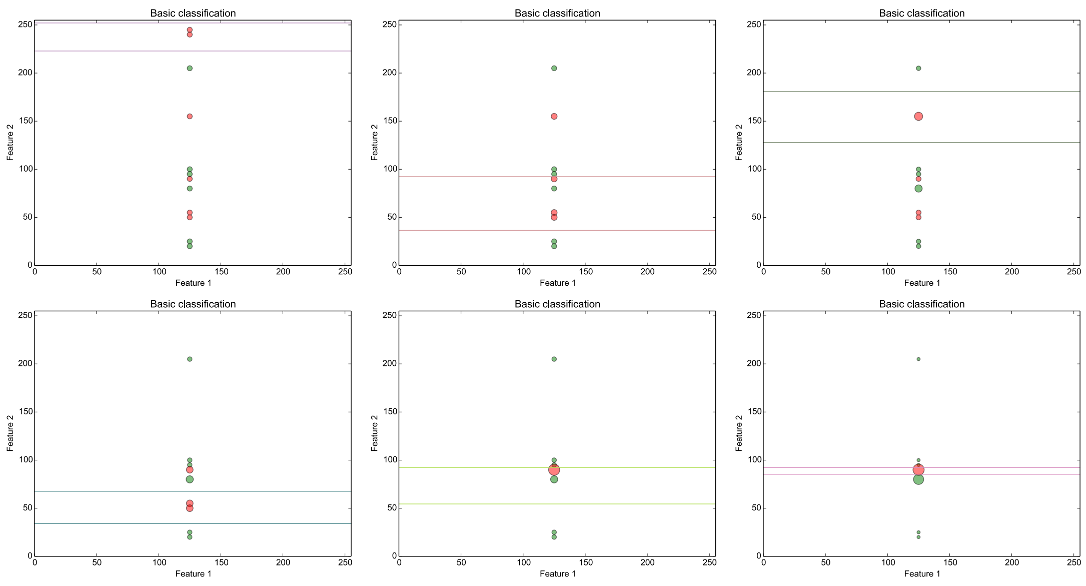


FIGURE 7.2: First nonliear basic test with two feature $f \in [0, 255]$. The radius of each sample represents its weight.

The proposed classifier introduces a lot of changes over the traditional methods on which it is based on. Thus, it has been tested over a set of simple linearly and nonlinear separable datasets in a 2-dimensional space as an initial proof of concept.

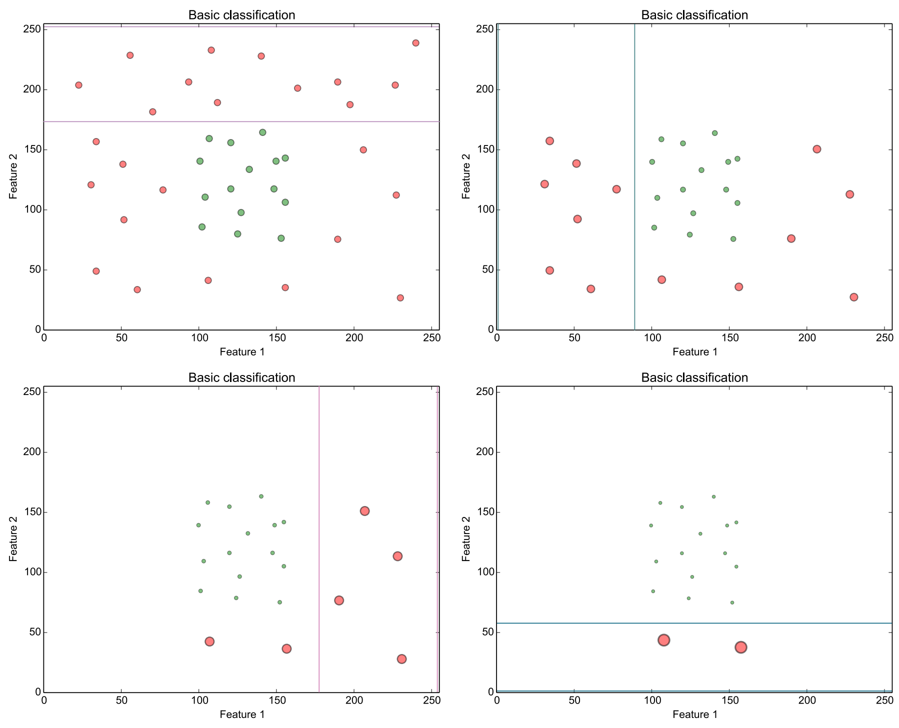


FIGURE 7.3: Second nonliear basic test with two feature $f \in [0, 255]$. The radius of each sample represents its weight.

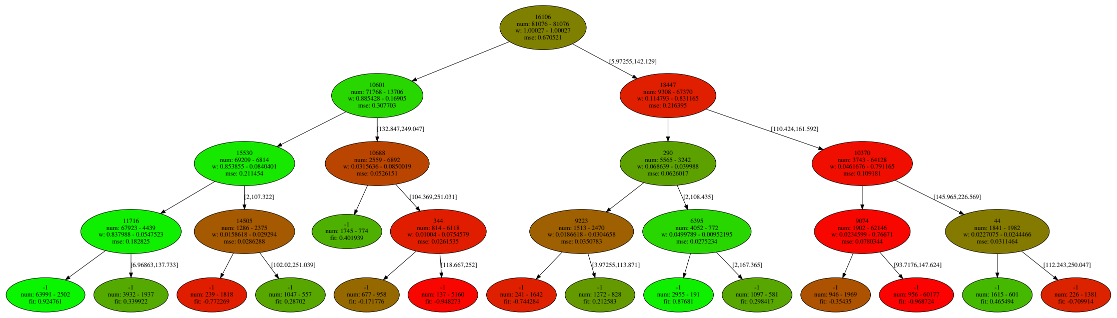
First, it was tested with two simple linearly separable datasetes (see Figure 7.1). Notice how the thresholds are able to maximize the margin between clusters thanks to the proposed modification of the decision trees splitting criterion.

The proposed classifier was also tested with more difficult nonlinear samples (see Figure 7.2 and Figure 7.3). To make the plots more understandable, all presented examples have been trained with the classifier constrained to learn one level decision trees. The radius of the samples (circles) represent the weight of each sample.

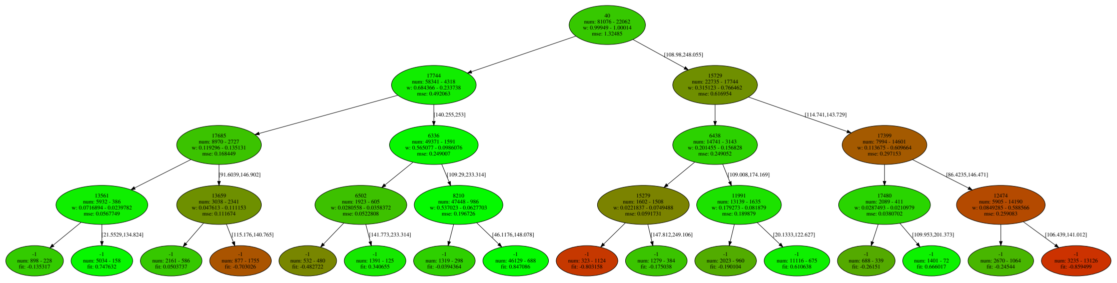
As it can be seen, the classifier is able to learn a nonlinear model to correctly classify nonlinear separable samples. Notice how at each step the weight associated to misclassified instances increase so that the following classifier take them more into consideration.

7.2 Learned cascade decision trees

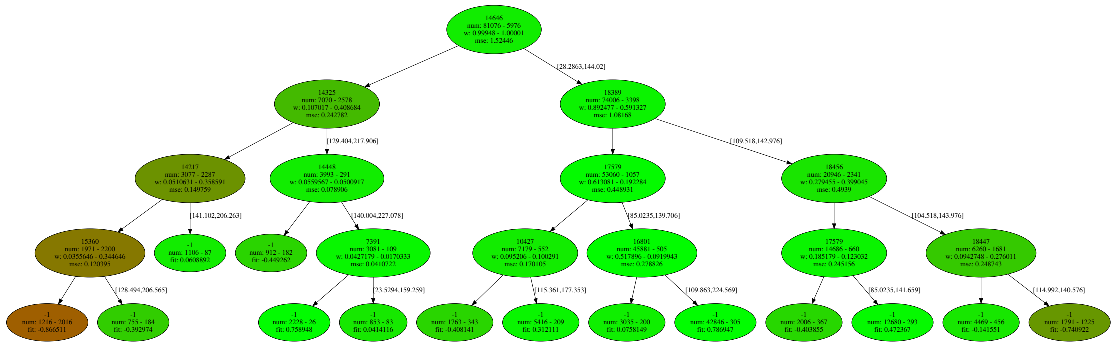
With the objective of making the classifier as transparent as possible we developed a script to plot a representation of the weak classifiers (see Figure 7.4). Each node of the tree contains the following information: ID of the selected feature, number and weight



(A) First weak classifier.



(B) Second weak classifier.



(C) Third weak classifier

FIGURE 7.4: Learned cascade decision trees.

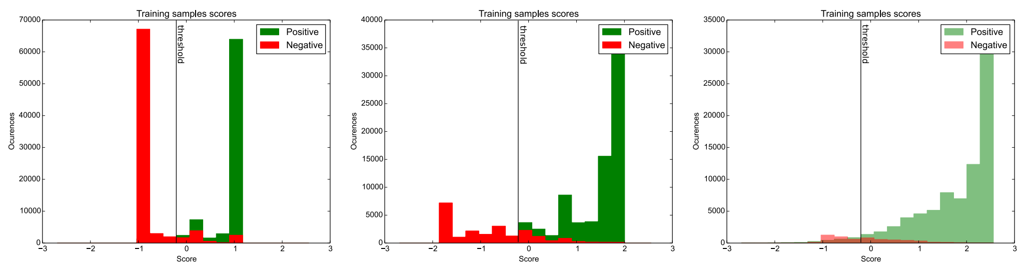
of the positive and negatives samples to train, and the classification error of the selected feature. The leafs contain the number of positive and negatives samples that have fallen into that node and the learned score of that leaf. The rgb color of each node represents the proportion positive (green) vs negative (red) samples. Hence, the greener a node the more training positive samples fall into that node.

It is important to know that there is not constraint informing to which direction all the nodes should send each label. Hence, going right or left it is not relevant and some nodes send the positives to left and some to right. The only important think is to split data, no matter the direction.

The resulting trees have the expected behaviour. Each node tries to split positive from negative samples. Also in the first nodes, the samples are not well clustered, then, as the samples go down the tree they are better classified. In the first and third trees it can also be seen that the implemented decision trees are not necessary full. This can be due to the trimming algorithm or because that node only received samples of one class.

The evolution of the weak classifiers is also the expected one. The more advanced a weak classifier is located in the cascade the higher the proportion of positive samples in each node. Although bootstrap is trying to add more negative samples, the classifier is capable of rejecting almost all of this new samples with the previously trained weak classifiers.

7.3 Evolution of the training samples scores and thresholds



(A) First weak classifier. (B) Second weak classifier. (C) Third weak classifier.

FIGURE 7.5: Evolution of the training samples scores and thresholds.

The next experiments presents the evolution of the training samples scores and weak classifiers thresholds as the classifier is learned (see Figure 7.5). Once the first decision tree is learned, each samples receives its score according to the node on which it had fallen. Surprisingly, the first weak classifier, by just combining 4 simple NPD features, it is already capable of correctly clustering most of the samples. It can also be seen that the thresholds being learned using a linear one-dimensional SVM are correctly prioritizing the idea of letting pass almost all positive samples.

7.4 Evolution of the training samples quantity

The correctly classified true negatives and the samples with low contribution (low weight) are discarded. This causes that as the number of weak classifiers increases, the number

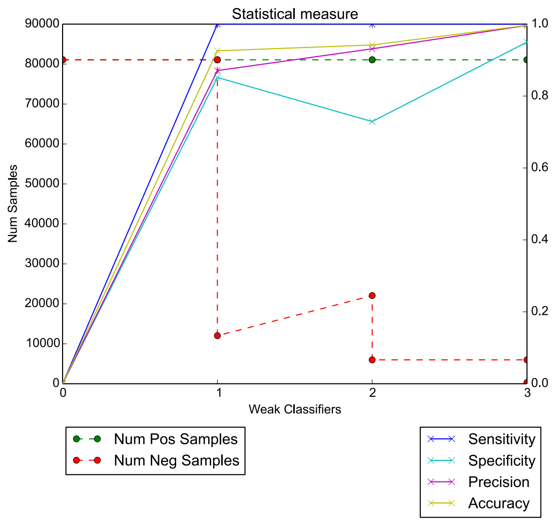


FIGURE 7.6: Evolution of the training samples quantity.

of samples decreases. To counteract this effect bootstrap was added. The goal of this experiment is to determine the variations over the quantity of training samples caused by bootstrap, trimming and the weak classifiers (see Figure 7.6).

As expected, the number of positives samples decrease slower than the negative samples because the only way they can be discarded is by the trimming process. The negatives samples decrease faster however, it can be seen that bootstrap tries to compensate this effect by adding more negative samples. As anticipated, as the classification rate over the training samples increases, bootstrap is less capable of finding new negative samples to add. When the classification rate is close to 1, bootstrap is not able to add enough negative samples and the training process stops due to a lack of negative samples. For example, in Figure 7.6, at the third weak classifier bootstrap is already not able to add more negative samples. This experiments shows the importance of having a much bigger dataset of rich negative samples with respect to the positive dataset.

7.5 Evolution of the testing samples scores

The plots illustrating this experiment were used a lot during the whole development of the project. All updates and upgrades done to the detector or its learning process were only added if they proved to improve the classification rate over the validation samples and shown a significant change in their evolution of the score.

		Prediction outcome		
		$\widehat{p}$	$\widehat{n}$	**total**
Actual value	**p**	TP 17979	FN 21	P=17800
	n	FP 13	TP 17987	N=17800
	total	$\widehat{P}$=17992	$\widehat{N}$=18008	

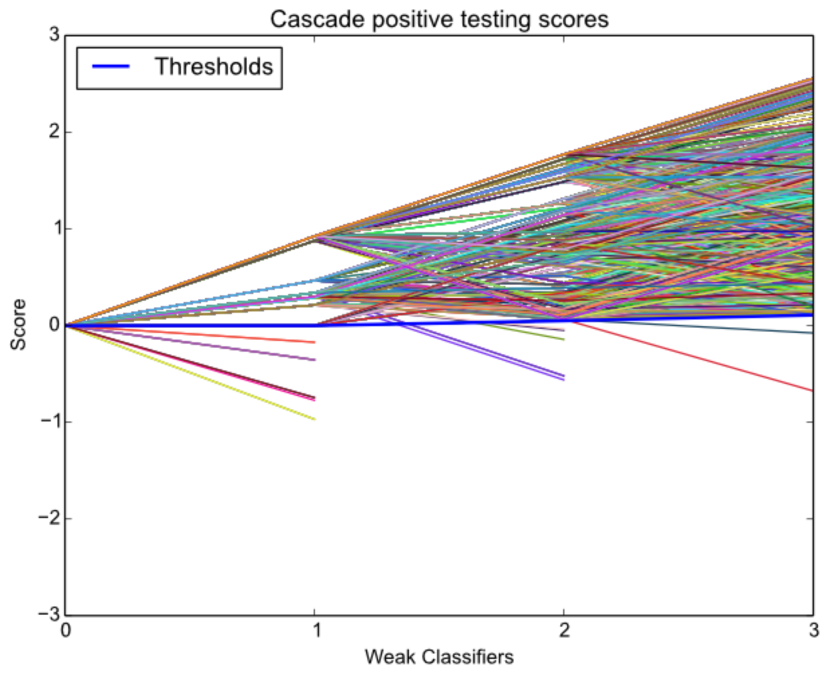
Table 7.1: Confusion matrix.

This experiment introduces the evolution of the scores, not over the validation set (set of instances in the training set considered when cross-validating) but over the test set (20% of the dataset). As it can be seen in Figure 7.7, in general the positives samples keep increasing its score while the negatives do the opposite until they are discarded when crossing the threshold. However, it can be seen that it is not always the case. Some positives samples do cross the cascade threshold (false negatives) and some negatives samples never end up crossing the threshold (false positives). A explanation of the scores trace has been previously explained in Section 3.3.

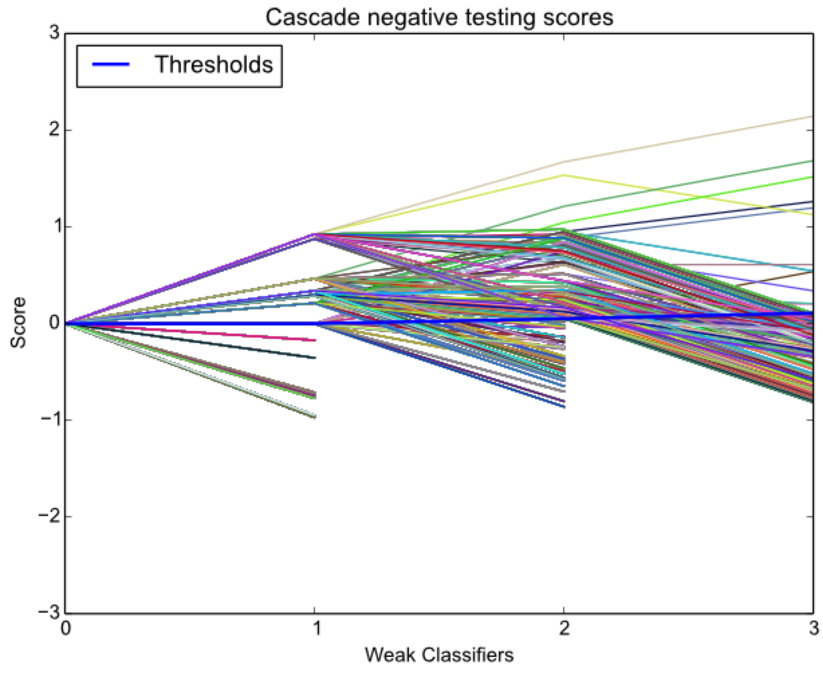
The confusion matrix is presented in Figure 7.1. As it can be seen the percentage of true positives is $TP = 99.9\%$ and true negatives reach also $TN = 99.9\%$. These results seem perfect but they are actually not. For an image of 1080p around 2400 windows are evaluated. Hence, around 24 windows could be false positive and false negatives. To filter these possible misclassified samples non-maximum suppression is used (see Section 7.7).

7.6 Rejection evolution of the testing samples in the cascade

The proposed face detector is composed by two clear stages. First, a stage composed by a cascade of weak classifiers to discard almost all negative samples. Second, a final strong classifier only used to classify the most challenging samples. This experiment presents the rejection evolution of booth stages over the testing set (see Figure 7.8).



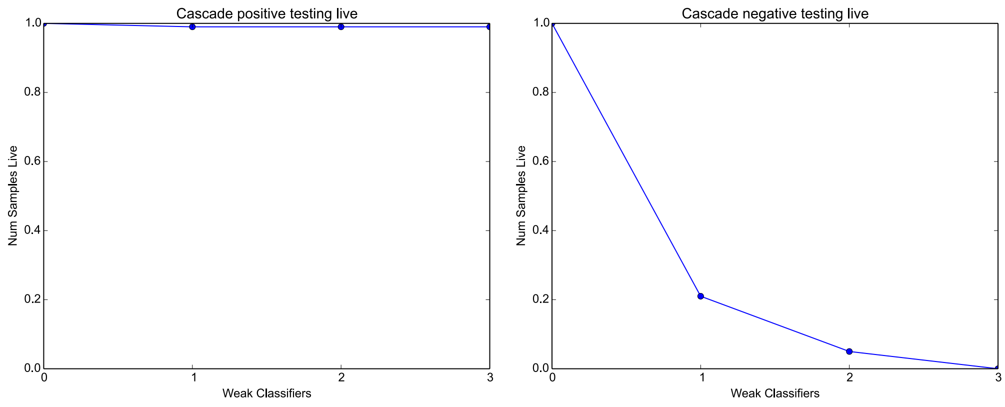
(A) Positive samples.



(B) Negative samples.

FIGURE 7.7: Evolution of the testing samples scores.

79% of the samples are rejected with only the first three weak classifiers. These results are really promising. The proposed decision trees are capable of correctly classifying 99% of the negative samples only by applying 15 difference of normalized pixels values. Only 1% of the samples require the use of the strong classifier as a filter.



(A) Positive samples. (B) Negative samples.

FIGURE 7.8: Evolution of the testing samples live.

7.7 Non-maximum suppression as a filter

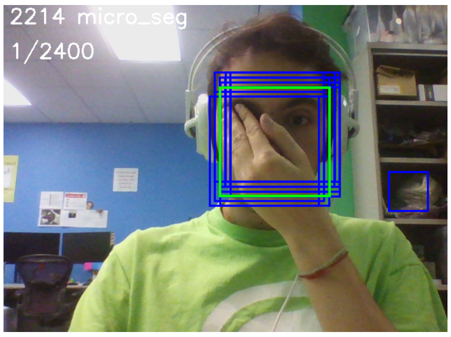


FIGURE 7.9: Real time video frame showing non-maximum suppression. Blue bounding boxes are all the face detections, the green one is the final detection.

During testing, for a single face, multiple windows are classified as face. To combine all these multiple overlapped detections into a single one the well-known non-maximum suppression algorithm is applied (see Figure 7.9).

False positive are normally not as overlapped as true positive. Thus, we also use the non-maximum suppression to filter false positives by setting a threshold of the minimum overlap required to be considered a true positive.

7.8 Real time

As mentioned in the objectives, the face detector was required to run at real time (see Figure 7.9). To validate this objective, the learned face detector was run over a real-time

	Our Method (full speed)	Our Method (for FDDB)	NPD	Yan-DPM [30]	JCascade [16]
CPU	i5@1.6GHz	i5@1.6GHz	i5@3.1GHz	X5650@2.66GHz	@2.93GHz
Cores	4	4	4	6	n/a
Threads	1	1	4	12	1
Speed [fps]	500	70	70.06	25	34.97

Table 7.2: Unconstrained face detection speed comparison.

	Our Method (unconstrained)	Viola-Jones OpenCV (frontal)
CPU	i5@1.6GHz	i5@1.6GHz
Cores	4	4
Threads	1	4
Speed [fps]	500	333

Table 7.3: Speed of our method vs the once currently being used.

	Our Method (unconstrained)	Viola-Jones OpenCV (frontal)
Space	20KB	15MB
Memory	43MB	123MB

Table 7.4: Space and memory of our method vs the once currently being used.

input video stream of 1080p. On a i5@1.6GHz without GPU it runs at 0.002ms (500 Hz) with multi-scale multi-face detection. It is important to notice that it is running at 500Hz on a single thread. Multi-thread was not used because mobile phones do not ensure multiple process execution for developer defined tasks.

A table comparing speed performance is presented in Table 7.2 (data has been obtained from [14]). As it can be seen our proposed method is 7.14 times faster than the fastest current state of the art method for unconstrained face detection.

In Table 7.3 it is also presented a comparison of the proposed method versus the Viola-Jones OpenCV face detection currently being used in the Human Sensing Laboratory pipeline. Notice that our method works with unconstrained face conditions while the OpenCV face detector only works for frontal views.

When using detectors in a mobile phone it is also very important to take into account the model size and memory at runtime. No user wants to install an application which consumes a lot of space or memory (high battery loss). Thus a comparison of the currently being used face detector in the laboratory (OpenCV) versus our proposed method is done in Table 7.4. As it can be seen, booth the model size and the required runtime memory are lower in our proposed method.

7.9 Benchmark

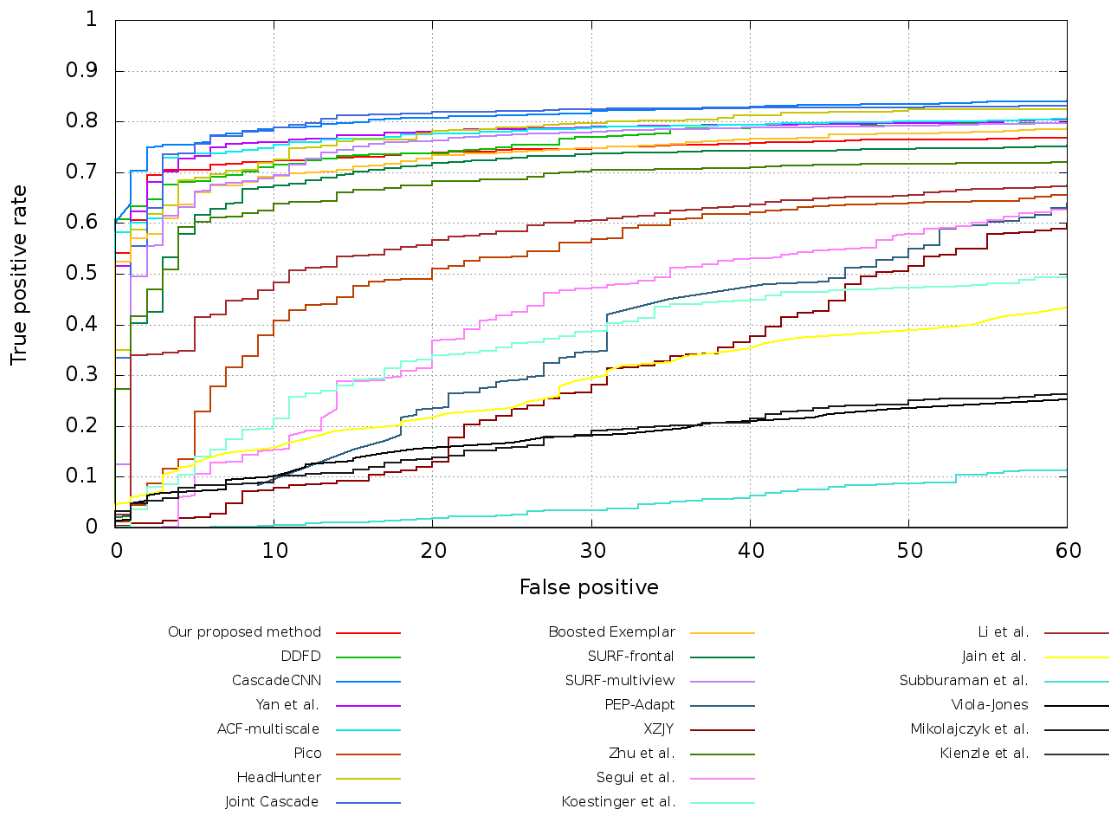
The proposed learning method have been evaluated with the challenging FDDB benchmark [8]. It evaluates 2845 images containing 5171 faces. The discrete and continuous ROC curves are presented in Figure 7.10a and 7.10b respectively. As it can be seen our proposed method is among the best state of the art methods. It is also important to remark that our face detector is only capable of returning squared-shape face detections and the ground truth face annotations provided by the benchmark are ellipses. Thus, the classification performance is negatively affected, specially the continuous metrics.

All methods with better accuracy are very slow as specified in [31], [32]. The method most similar to ours in terms of accuracy and speed is the *Joint Cascade* [16]. Their method have a better accuracy but at a cost of lower speed, 34.97 fps. Our method can run at top speed at 500 fps but to evaluate the FDDB benchmark it has been decreased to 70 fps to achieve better accuracy. The *Joint Cascade* method uses a post-processed classifier to filter the most challenging samples to improve its results. Our method can still be updated with costly computational processes thanks to its current high speed. Thus, as a future work and inspired by the *Joint Cascade* method it is proposed to also add a classifier to filter samples.

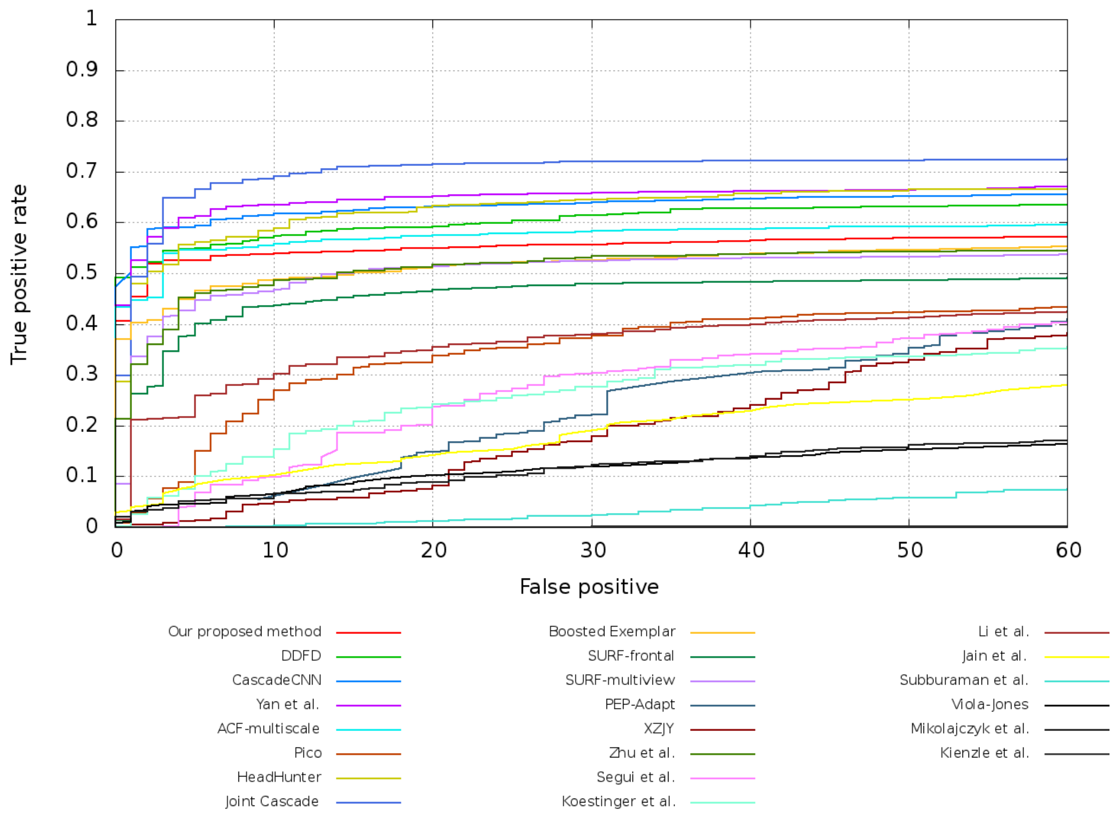
In our case, the most important information to extract from this benchmark that the proposed method has a much higher accuracy than the Viola-Jones method which was the one currently used in the Human Sensing Labortory pipeline.

Figure 7.11 and Figure 7.12 present some of the evaluated images of the FDDB. As it can be seen this benchmark has faces with challenging expressions, position, lighting, blur and occlusions. We have put a lot of effort to solve one the the most common problems in face detectors which is to correctly classify African-Americans but still we have not been able to solve it. This problem appears because it is more difficult to distinguish the common features of a face on a black skin when there is not enough illuminated light. We have also put a lot of effort to detect people faces wearing sunglasses, beards and with atypical illumination. In most cases the learned detector is able to detect them.

Even though we did not make any effort to make the learned detector robust against occlusions it works surprisingly well with occluded faces. This is mainly thanks to the tree structure combined with the NPD features. Features analyzing occluded part of the



(A) Roc - Discrete score metrics ROC.



(B) Roc - Continuous score metrics ROC.

FIGURE 7.10: ROC curves of the FDDB benchmark.

face will return that its not a face. But, thanks to the deep tree structure the sample will be moved to other branches of the tree with local features situate in the parts of the face that are not covered. In this way, the sample will be correctly classified. Moreover, the NPD features are not depend on a continuous area like Haar. As long as the two points defining the feature vector are not occluded it does not matter whats on the middle.

Currently the greatest weaknesses of the detector are very tiny faces, or small blurred faces. This is caused because we have not focused our attention on this type of images. Even if we are able to detect them, no good posterior analysis could be performed.

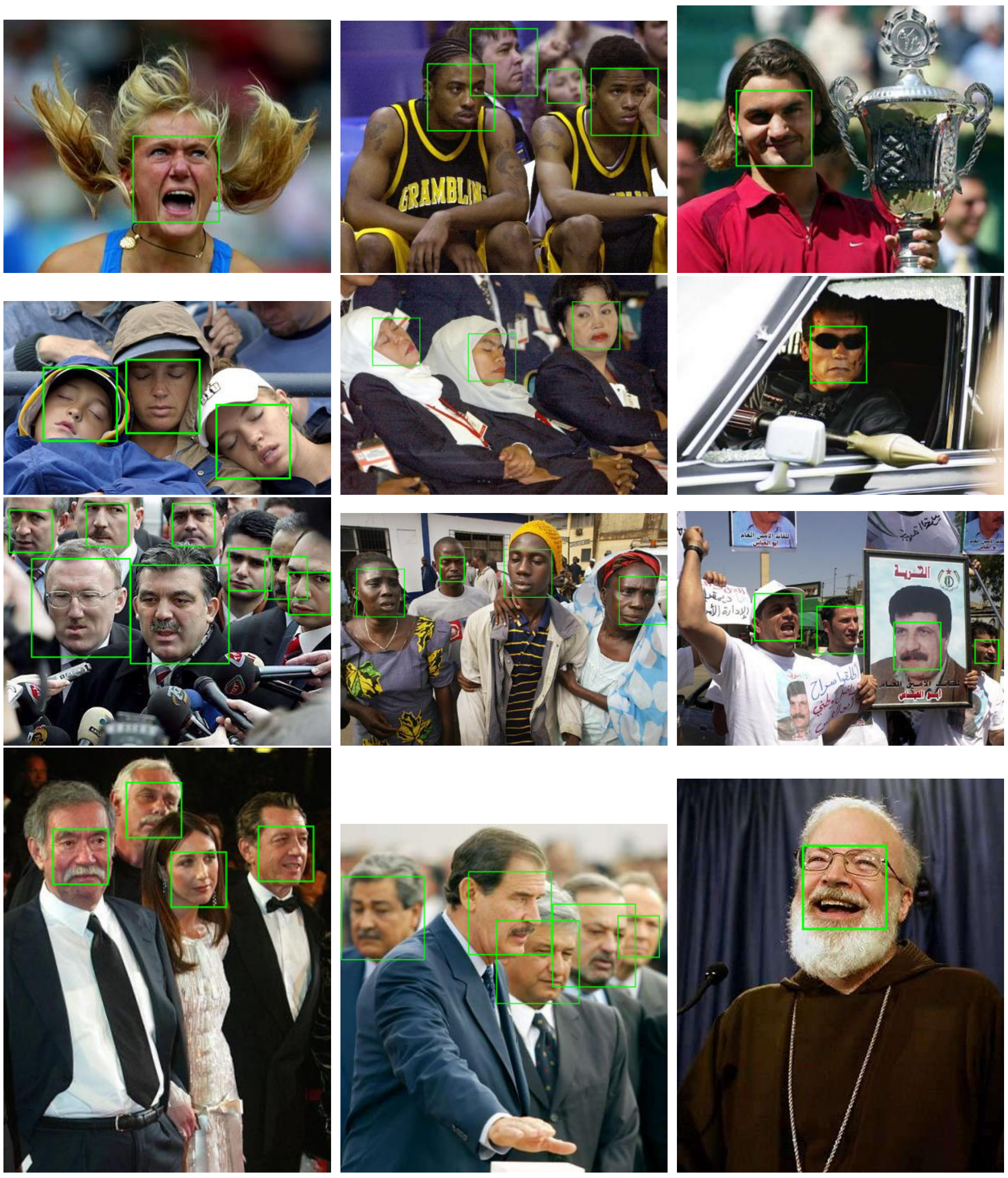


FIGURE 7.11: Correctly classified samples of the FDDB benchmark.

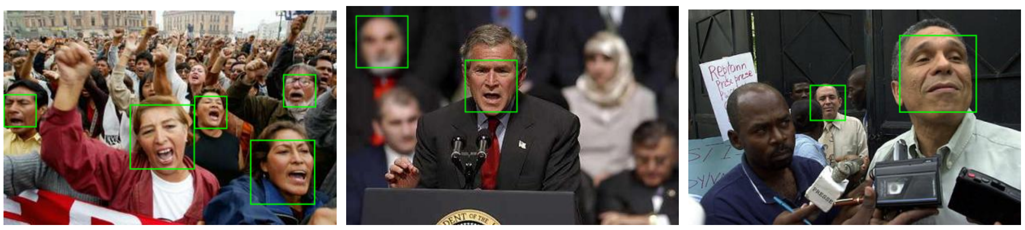


FIGURE 7.12: Misclassified samples of the FDDB benchmark.

Chapter 8

Sustainability and social commitment

This project introduces a new learning method for face detection. There is still a lot of work to be done, but with the proposed extensions the introduced method could have a great impact on real-time face frameworks system for low power computational systems. Therefore it could have an important social, economical and environmental impact.

The face is one of the most powerful channels of non-verbal communication. Thus, if we could track a persons face thought out the day we could estimate his/her internal states, social behaviour, biometrics and psychopathology. This could be achieved by enhancing all kind of products with the ability to monitor a persons face at a low cost.

The face detection module of the pipeline of the tracking system could be implemented with the proposed infrastructure because of its real-time and low computation complexity cost. It could run on a low cost processor and camera sensor with minimal energy consumption (only one thread and 43MB of memory to run).

8.1 Economical impact

By monitoring a persons face we could determine not only his/her internal state but also what is he/she focusing at and having interest for. Which would have a very important economical impact. Brands would know what people are interest for, allowing them to conceive better strategic decisions.

Manufacturers and designers would be able to open a hole new world of endless possibilities. For example, cars stopping when they detect that the person is not focused on the road. Online stores recommending products and improving its website design based on what you are eyes are focusing on while interacting with their website. Or even domotic houses changing their illumination and playlists based on their owners mood.

8.2 Social impact

Being able to monitor a person state in a daily basis would help to improve the welfare state, thus, making a big social impact. Improvements in security by detecting atypical internal state of individuals in crowded spaces; improvements in health treatment by detecting diseases symptoms such us depression in an early stage; or even improvements of the public infrastructures based on how the citizens interact with them.

However, it could also cause a negative social impact due to privacy rights. A big proportion of population would not tolerate a system to track their internal state through out the day. Even if it is improving their welfare state. This negative impact could be reduced by introducing this kind of system in a slow and graduate way and making social awareness campaigns.

8.3 Environmental impact

As previously explained, this technology could rise a new tendency to install low cost cameras in all our appliances and infrastructures. The augment of small camera sensors would cause a negative environmental impact due to all the generated technological wastes.

Despite the possible indirect negative impact, the proposed framework is camera agnostic. In other words, is not dependant of the camera hardware and lens as it can be retrained for a given hardware specifications. This property would allow the production of new appliances and infrastructures with recycled or leftover components from previous productions. Thus, this property of the framework would reduce the ecological footprint of possible future mass production of low cost camera.

Chapter 9

Conclusions and future work

In this project a new learning method for unconstrained face detection has been presented. The proposed method introduces a set of upgrades and modifications of the key concepts and ideas of Decision Trees, AdaBoost and Soft Cascade.

In this project we have presented a new variation of Decision Trees able to learn the optimal combination of features to cluster faces under unconstrained face position and orientation thanks to its deep property despite being trained without face orientation and viewpoints information. It has also been redefined its splitting criterion, now there are second order thresholds which allow them to express second order information contained in the features. They have also been enhanced with the ability to approximately maximize the margin distance similar to an SVM.

It has also been proposed a new definition of the Soft Cascade thresholds training principles which are now learned with one-dimensional linear SVMs during the training phase. This new strategy is not a nïve based approach anymore but instead a decision based on the data.

It has also been demonstrated with a simple example case how the traditional AdaBoost loss function can end up miss-leading the trained weak classifiers. To solve this problem a better formulation of the AdaBoost loss function which takes into account the cascade thresholds has been proposed.

The unsupervised property of the proposed method allows to spend much less time labeling. It is only necessary to crop faces. Moreover, by not having to label each sample to its corresponding view the human-labeling error is eliminated.

The learning algorithm takes 20 minutes to learn the model in a 20 cores Intel Corporation Xeon E7 v2/Xeon E5 v2/Core 3GHz sever without GPU. In the experimentation phase it has been proven with the FDDB benchmark that the trained detector achieves state-of-the-art performance in detecting unconstrained faces. Moreover, it runs at 0.002ms (500 fps) on a i5@1.6GHz without GPU which is higher than the current state of the art unconstrained face detectors. It has also been shown that the dimension of the learned model is 20KB and only requires 43MB of memory to run. Which makes it perfect for hardly constrained low space and computational complexity devices like mobile phones. It has also been proven that the learn detector is able to detect faces under unconstrained face position and orientation and even with occlusions, different lightning conditions and resolutions.

One of the main reasons of the detector speed-up with respect to the current state of the art methods is that we have taken as much as effort to develop the math as to develop the code. Booth learning process and final detector have been carefully implemented with modern and highly optimized code with SIMD instructions which allow to vectorize code at a hardware level.

The strengths and weaknesses of the detector have also been analyzed and justified during the study of the FDDB benchmark classification results. The strengths of the learn detector are: unconstrained position and orientation, atypical illumination, occlusions, sunglasses and beards. The weaknesses are African-Americans (working on that) and very tiny faces or small blurred faces. We are not interested in solving the last two weaknesses because even if detected, no good posterior analysis could be performed.

All the established technical objectives have been proven to be achieved in the experimentation section. All personal set of goals related to this project have also been achieved. Having the opportunity to develop this project at the Carnegie Mellon University's Robotics Institute, in the Human Sensing Laboratory has been my best professional and personal experience so far. There I have not only learn a lot of theory related to machine learning and computer vision but also what it means to be working at a top laboratory in all the personal and labour aspects.

There is still a lot of work to be done to improve the learned detector. First, a better training dataset must be generated. The current one has not enough samples for some of the face viewpoints. Second, the learned face detector can still be upgraded with a new costly computational processes thanks to its current high speed. Inspired by the *Joint Cascade* [16] method, a post-processed classifier to filter the most challenging samples to improve its results should be added. Currently it is being tested with linear SVMs for each one of the viewpoints but we consider that it is not a good solution because then the method needs further training. Third, more code optimization processes could be made with template meta-programing at a cost of code readability. Fourth, keep working in trying to improve African-American people face detections. Fifth, learn a CNN for face detection and extract from a cross-validated layer its kernels which then would be used as features in our learning method.

Chapter 10

Costs

To fulfill this project a set of hardware, software and human resources are needed. In this chapter a cost analysis is presented. Hardware and software costs have been calculated based on its amortization. To calculate human resources costs a workload of 84 hours/week (defined in the temporal planing).

10.1 Hardware costs

HARDWARE COSTS				
Product	Price	Units	Service life	Amortization
Server*	8000 €	1	4 years	564 €
Sony vaio	999 €	1	4 years	70.55 €
TOTAL	**999 €**	-	-	**70.55 €**

Table 10.1: Hardware Costs Table.

10.2 Software costs

SOFTWARE COSTS				
Product	Price	Units	Service life	Amortitzation
Ubuntu 14.04	0 €	1	1 year	0 €
Vim	0 €	1	1 year	0 €
OpenCV	0 €	1	1 year	0 €
ShareLaTeX	0 €	1	1 year	0 €
BitBucket	0 €	1	1 year	0 €
Mendeley	0 €	1	1 year	0 €
TOTAL	**0 €**	-	-	**0 €**

TABLE 10.2: Software Costs Table.

10.3 Human resource costs

This project have been only developed by one person. Hence, this person did take the role of all the persons which would be normally involved in this kind of project. It's been assumed that this work would involve: *Project Manager* with a salary of 20 €/ hour; *Programmer* with a salary of 12 €/ hour; *Tester*, with a salary of 10 €/ hour.

HUMAN RESOURCES COSTS (1/2)			
Task	Rol	Time [h]	Price
State of the art	Project Manager	60	1200 €
	Programmer	108	1296 €
	Tester	0	0 €
	Total	**168**	**1496 €**
Debugging dataset generation	Project Manager	0	0 €
	Programmer	0	0 €
	Tester	25	250 €
	Total	**25**	**250 €**
Learning classifier	Project Manager	5	100 €
	Programmer	113	1356 €
	Tester	50	500 €
	Total	**168**	**1956 €**
Features	Project Manager	5	100 €
	Programmer	59	708 €
	Tester	20	200 €
	Total	**84**	**1008 €**
Learning algorithm	Project Manager	10	200 €
	Programmer	223	2676 €
	Tester	100	1000 €
	Total	**333**	**3876 €**
Classifier training and validation	Project Manager	2	40 €
	Programmer	8	96 €
	Tester	0	0 €
	Total	**10**	**1036 €**

TABLE 10.3: Human resources costs table 1/2.

HUMAN RESOURCES COSTS (2/2)			
Task	Rol	Time [h]	Price
Classifier implementation optimization	Project Manager	1	10 €
	Programmer	72	864 €
	Tester	13	130 €
	Total	**86**	**1004€**
Final documentation	Project Manager	10	200 €
	Programmer	5	60 €
	Tester	5	50 €
	Total	**100**	**310 €**
TOTAL		**1285**	**10936 €**

TABLE 10.4: Human resources costs table 2/2.

10.4 Overhead costs

Overhead costs			
Concept of the spending	Price	Time	Total
Rent and utilities	864 €/month	22 weeks	4752 €
ADSL	45 €/month	22 weeks	990 €
Transport	35 €/month	22 weeks	770 €
TOTAL			**6512 €**

TABLE 10.5: Overhead costs table.

10.5 Total costs

Finally, a total sum of all costs is presented with "tipologia general" taxes of 21%.

TOTAL COSTS	
Concept of the spending	Cost
Hardware	70.55 €
Software	0 €
Human resources	10936 €
Overhead costs	6512 €
Subtotal	17518.55 €
I.V.A (21%)	3678.9 €
TOTAL	**21197.45 €**

Table 10.6: Total costs table.

10.6 Economic viability

The *Human Sensing Laboratory* plans to have **major** economic benefits in a future by selling the complete face analysis pipeline as a library. Therefore, this project would be economically viable for the laboratory because it is one indispensable module of the pipeline.

Bibliography

[1] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[2] Edgar Osuna, Robert Freund, and Federico Girosi. Training support vector machines: an application to face detection. In *Computer vision and pattern recognition, 1997. Proceedings., 1997 IEEE computer society conference on*, pages 130–136. IEEE, 1997.

[3] Tin Kam Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE, 1995.

[4] Shuo Yang, Ping Luo, Chen-Change Loy, and Xiaoou Tang. From facial parts responses to face detection: A deep learning approach. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3676–3684, 2015.

[5] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *Neural Networks, IEEE Transactions on*, 8(1):98–113, 1997.

[6] Ilya Kalinowski and Vladimir Spitsyn. Compact convolutional neural network cascade for face detection. *arXiv preprint arXiv:1508.01292*, 2015.

[7] Adrian Barbu, Nathan Lay, and Gary Gramajo. Face detection with a 3d model. *arXiv preprint arXiv:1404.3596*, 2014.

ETSEIB

[8] Vidit Jain and Erik Learned-Miller. Fddb: A benchmark for face detection in unconstrained settings. Technical Report UM-CS-2010-009, University of Massachusetts, Amherst, 2010.

[9] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep convolutional network cascade for facial point detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3476–3483, 2013.

[10] Hongliang Jin, Qingshan Liu, Hanqing Lu, and Xiaofeng Tong. Face detection using improved lbp under bayesian framework. In *Image and Graphics (ICIG'04), Third International Conference on*, pages 306–309. IEEE, 2004.

[11] Qiang Zhu, Mei-Chen Yeh, Kwang-Ting Cheng, and Shai Avidan. Fast human detection using a cascade of histograms of oriented gradients. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1491–1498. IEEE, 2006.

[12] Jianguo Li, Tao Wang, and Yimin Zhang. Face detection using surf cascade. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 2183–2190. IEEE, 2011.

[13] Hongming Zhang, Wen Gao, Xilin Chen, and Debin Zhao. Object detection using spatial histogram features. *Image and Vision Computing*, 24(4):327–341, 2006.

[14] S. Liao, A. K. Jain, and S. Z. Li. A fast and accurate unconstrained face detector. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):211–223, Feb 2016. ISSN 0162-8828. doi: 10.1109/TPAMI.2015.2448075.

[15] Nuno Vasconcelos and Mohammad J Saberian. Boosting classifier cascades. In *Advances in Neural Information Processing Systems*, pages 2047–2055, 2010.

[16] Dong Chen, Shaoqing Ren, Yichen Wei, Xudong Cao, and Jian Sun. Joint cascade face detection and alignment. In *Computer Vision–ECCV 2014*, pages 109–122. Springer, 2014.

[17] Nenad Markus, Miroslav Frljak, Igor S Pandzic, Jörgen Ahlberg, and Robert Forchheimer. Object detection with pixel intensity comparisons organized in decision trees. *arXiv preprint arXiv:1305.4537*, 2013.

[18] Chang Huang, Haizhou Ai, Yuan Li, and Shihong Lao. High-performance rotation invariant multiview face detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(4):671–686, 2007.

[19] Yen-Yu Lin and Tyng-Luh Liu. Robust face detection with multi-class boosting. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 680–687. IEEE, 2005.

[20] Wei-Yin Loh. Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1):14–23, 2011.

[21] Bo Wu, Haizhou Ai, Chang Huang, and Shihong Lao. Fast rotation invariant multi-view face detection based on real adaboost. In *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, pages 79–84. IEEE, 2004.

[22] Stan Z Li and ZhenQiu Zhang. Floatboost learning and statistical face detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(9):1112–1123, 2004.

[23] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[24] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[25] Marti A. Hearst, Susan T Dumais, Edgar Osman, John Platt, and Bernhard Scholkopf. Support vector machines. *Intelligent Systems and their Applications, IEEE*, 13(4):18–28, 1998.

[26] Stan Z Li, Long Zhu, ZhenQiu Zhang, Andrew Blake, HongJiang Zhang, and Harry Shum. Statistical learning of multi-view face detection. In *Computer VisionECCV 2002*, pages 67–81. Springer, 2002.

[27] Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 1, pages I–900. IEEE, 2002.

[28] Pawan Sinha. Qualitative representations for recognition. In *Biologically motivated computer vision*, pages 249–262. Springer, 2002.

[29] ShengCai Liao, Zhen Lei, XiangXin Zhu, Zhenan Sun, Stan Z Li, and Tieniu Tan. Face recognition using ordinal features. In *Advances in Biometrics*, pages 40–46. Springer, 2006.

[30] Bin Yang, Junjie Yan, Zhen Lei, and Stan Z Li. Aggregate channel features for multi-view face detection. In *Biometrics (IJCB), 2014 IEEE International Joint Conference on*, pages 1–8. IEEE, 2014.

[31] Haoxiang Li, Zhe Lin, Xiaohui Shen, Jonathan Brandt, and Gang Hua. A convolutional neural network cascade for face detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5325–5334, 2015.

[32] Haoxiang Li, Gang Hua, Zhe Lin, Jonathan Brandt, and Jianchao Yang. Probabilistic elastic part model for unsupervised face detector adaptation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 793–800, 2013.

ETSEIB