# Design and implementation of a boosting technique for object identification

## Abstract

Object identification is used in a widely number of fields for daily tasks, e.g. face, pedestrian or car recognition. One of the most used object identification algorithm is the one proposed by Viola and Jones that combines simple features extraction from integral images, and AdaBoost algorithm as a way to train and create a strong classifier using a cascade of weak classifiers. This project aims to replicate the Viola-Jones algorithm using their paper (1) and to use it for face identification... In order to do so, the tools being used in this project are Matlab and a set of random images.

1. Introduction

Face Detection had a big development in the last century, and nowadays is used in all kind of devices and media, like cameras, cell phones or websites[[1]](#footnote-1). Viola and Jones (1) algorithm has been specially popular due its low false positive rate, high performance - they were able to detect faces within a 380x280 image in less than 0.7 seconds in now obsolete computers, with a 85% prediction rate and 1 in 14084 false positive rate (2) - and its versatility - it can be easily trained to detect all kind of objects. Nowadays, some of the object recognition algorithms were inspired[[2]](#footnote-2) by or improved versions of the Viola and Jones algorithm. Also, its relative simplicity makes it a very interesting subject of research in Computer Vision.

* 1. Overview

This report aims to explain the workings of the AdaBoost Algorithm: the importance of using Haar-like Features with Integral Images, how the AdaBoost variation used works, the initial considerations used to implement the algorithm and what were the main challenges and problems that were faced.

Although the algorithm can be trained to recognise any type of object, in this project it has been used exclusively to recognise faces, so this report will use the notation of “faces” and “non faces” to describe the objects that are aimed to be classified.

1. State of the art

Facial recognition systems enable a computer to identify or verify a person that appears in an image or a video from a video source. Face recognition is a technology that has evolved in a fast rate the last 20 years, and nowadays is used for several applications:

* **Identification**: for police to identify a person in an image or video from a database
* **Security**: to enable the use of systems, like computers or phones to a certain person
* **Camera photos**: so it can focus properly an image,
* **commercial identification and marketing tool**: so a system can automatically detect a human and show an advertisement that fits their interests
* etc.

Also, the technology that was developed to enable face recognition can be trained and used to recognize other things (animal, objects, etc.). This gives these algorithms a lot of applications in different fields.

* 1. Characteristics of the object recognition algorithms

Some of the Face recognition programs that were studied in this project arePrincipal Component Analysis, Hidden Markov model, Neuronal motivated dynamic link matching, Elastic Bunch Graph Matching, 3d face recognition… all of them have in common that their objective must solve 2 problems:

* It should select a feature or set of features that gives enough information for identification.
* It should use a classifier that is able to label an image as face or non-face. The selection of these classifiers tends to depend of the type of features we decided to use.

That is, face recognition combine two different fields: computer vision to extract features and pattern recognition to learn to classify it. All the different techniques are defined and named after the choice of different approaches in these two fields.

1. Viola and Jones Object Recognition algorithm

Originally designed for face recognition, The Viola–Jones AdaBoost face detection, proposed in 2001 by Paul Viola and Michael Jones, was used to create the Viola–Jones general object detection framework, the first competitive object detection framework in real-time. Its efficiency is based in 3 key concepts that were different to prior object detection paradigms:

* **Haar-like** rectangular **figures** were used to create the features that will be used to classify an image.
* Instead of working with the images luminance pixel by pixel, the images were pre-processed to create what is called an **Integral Image**.
* A **boosting algorithm**, variation of AdaBoost, was used in order to select several features used in a number of weak classifiers that, in combination, create a strong classifier.

In the following sections we will explain all three concepts.

* 1. Haar-like features

|  |
| --- |
| C:\Users\Ancode\Desktop\tmp35b0287_thumb.png |
| Figure 1: Examples of rectangles to be used to make features. (Haar-like features) |

Viola and Jones algorithm uses Haar-like features extracted in the images that will be used (to classify or to train the classifier)

These features are created by obtaining the sum of different areas of the image, and then adding or subtracting those resultant values. This is represented by black and white squares, being the white squares the region that will be added, and the black ones the areas that will be substracted from the white ones. This type of features is called Haar-like, by their similarity to Haar wavelets (3).

In Figure 1 there are examples of the representation of four possible types of Haar-like squares that can be used in the features. One must note that their proportions are not fixed: there are different variations of each Haar-like feature, using a different number of pixels (Figure 2).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |
| Figure 2: Examples of several versions of the same type of Haar-like type rectangle figure |

In order to create a Haar-like feature, a detection window of a specific pixel size is used, and considers adjacent rectangular regions at a specific location. To apply it to an image, we sum up the pixel intensities in the detection window's squares regions and calculates the difference between sums, and compare it with an specific value associated to the feature (created in the training using the sample image set).

In Viola and Jones algorithm[[3]](#footnote-3), the Haar-like features are represented by a grey image with black and white squares. The black part pixels luminance sum is subtracted from the pixels luminance sum that fall in the white part (Figure 3). The result of applying a feature to an image is a single number.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 2 | 5 | 6 | 4 | 4 | 1 | 7 | 2 | 5 | 6 | | 4 | 6 | 4 | 2 | 5 | 8 | 5 | 6 | 4 | 2 | | 2 | 5 | 1 | 5 | 1 | 5 | 3 | 5 | 1 | 5 | | 2 | 1 | 6 | 4 | 2 | 5 | 8 | 6 | 4 | 2 | | 6 | 4 | 5 | 1 | 5 | 1 | 5 | 5 | 1 | 5 | | 5 | 1 | 1 | 2 | 5 | 6 | 4 | 1 | 2 | 5 | | 1 | 2 | 9 | 4 | 6 | 5 | 5 | 9 | 4 | 6 |   Black part: 2+5+5+1+4+2+1+5+2+5=32 White part: 8+5+5+3+5+8+1+5+6+4=50  Haar-like associated value: 50-32=18 |
| Figure 3: Example of the calculation of the Haar-like associated value to a feature inside a 10x7 window |

Using Haar-like features we can have simple value associated to an image (or a part of an image) that is relatively easy to compare with others in order to create a classifier. Yet this method has two weak points:

In one image **there are a lot of Haar-like features combinations**. As an important part of the algorithm is to choose the best feature to be used in the final classifier, even if we limit the Haar-like features types to a small number, the exhaustive set that has too been taking in consideration can be computationally expensive, especially for big images. In order to optimize it, we can do the following:

* limit the number of feature types
* limit the feature size to small windows - for example, 24x24 pixels[[4]](#footnote-4)
* limit the Haar-like rectangles to a minimum rectangle size of , for example, 8x8 pixels[[5]](#footnote-5)

Adding those limitations the number of possible features to use can be reduced. In this project case, the number of features used is 58140.

* 1. Integral Images

|  |
| --- |
| C:\Users\Ancode\Desktop\references and images\integral-image-example.png |
| Figure 4: Original image and integral image[[6]](#footnote-6). In an integral image the value of the luminance of a pixel is the sum of the luminance from the pixels located above and below it. |

During the search for the suitable features, the each Haar-like value **must be calculated for every image** in order to have its associated number. That means to make several accesses to the image (20 accesses in Figure 3 case). That can be expensive in computational time, and is particularly important in the boosting part of the algorithm where it should be computed for a large pool of images during the training. This problem is solved using what we call "**Integral Images**".

An Integral Image, or II, is a matrix created based on an image, in where every new pixel is the sum of the intensity of the pixels above and to the left of it (Figure 4).

Where ii stands for integral image, and i(x,y) stands for the value of the intensity of the pixel located in the coordinates x, y.

Using the integral image, it is easy to calculate the sum of all pixels in one rectangle inside it only with the localization of the four corner points. In Figure 5 can be seen that, the value can be obtained with only 4 accesses to the intensity value from the ii, when we had to use the total number of the pixels that form the rectangle in the original image (6 in Figure 5's case).

|  |
| --- |
| C:\Users\Ancode\Desktop\references and images\integral-image-example.png |
| Figure 5: Calculation of the sum of all pixels intensity in one rectangle from an integral image[[7]](#footnote-7). |
|  |

We can write it using the following equation:

It should be noted that the computation of the Haar-like feature assigned value computes several square's intensity sum, and doing it with an integral image using this method reduces the computational complexity from O(n) to O(1): it will require only two subtractions and one addition to retrieve a sum, independently of the size of the region.

So, in Viola and Jones method, the Haar-like features are computed from an integral image, boosting the computational speed of the algorithm for a big set of images.

* 1. AdaBoost

|  |
| --- |
| C:\Users\Ancode\Desktop\references and images\cascade.jpg |
| Figure 6: Boosting algorithm diagram |

The combination of II with Haar like feature describe an efficient way to represent and compare features in large sets of images, but there is a big number of features to compare in each image (58.140 in this project’s case), so a method is needed to find the best features that discriminate faces from non-faces. Viola and Jones face detection algorithm uses boosting in order to find the best features.

Boosting, in this context, means that the algorithm have a training stage in where we select a set of “weak classifiers”, marginally better than luck, that, working together creates a “boosted” strong classifier.

In the stage before being able to be used to classify, the program should find the best features by testing them against positive (images that are faces) and negative (images that are not faces) examples. In this case, an adapted version of AdaBoost was used.

* + 1. Weak classifier

AdaBoost training uses a series of weak classifiers. In practice, any random weak classifier will have a marginally better chance of a 50% shot to classify faces from non faces. The idea is to use a set of weak classifiers to test one image; in every stage of the identification, one weak classifier from the set is used to check if an image can be identified it as a face. This process is repeated with all the classifiers that form the composite classifier. At the end, the results of all the classifiers are compared and classification will be return taking in account the “votes” of all the classifiers. Using weak classifiers in sequence can statistically improve the chances enough to create a strong classifier.

So, in order to create a weak classifier the following is needed:

* A set of positive (faces) and negative (non faces) sample images.
* A Haar-like feature

For each feature fj, we create a weak classifier; hj(x) is defined by applying the feature to the training set that will produce:

* a weighted histogram
* an optimal **threshold** ϴj that best separates the faces from non-faces
* a **parity** pj that describes whether the face outputs are above or below the threshold

With all this data, the weak classifier j becomes defined by the feature fj,, the optimal threshold ϴj and the parity pj assigned to it. The output of the classifier outputs 1 if the image is classified as a face and -1 if it is classified as non-face. Mathematically we can describe it as follows[[8]](#footnote-8):

* + 1. Training

|  |
| --- |
|  |
| Figure 7: The AdaBoost algorithm as shown in the Viola and Jones paper (1) |

The next step is to find the best set of weak classifiers. An intuitive but naïve way to do it would be to rank all the features in order of smallest error of classification. But this method won't work because it tends to select a set of features that are very similar but slightly scaled or shifted, being basically the same feature, so using it in sequence will not boost the strong classifier, being it too similar to a weak classifier (2).

The method Viola and Jones used is AdaBoost, though other boosting algorithms may be used depending the implementation. AdaBoost fix the problem by changing the weights of the image every time the training checks a new feature, making our second best feature not similar to our first best feature, and forcing to look for a different one. We have to repeat this process for all the features in order to select suitable weak classifiers. We can see the algorithm in Figure 7.

Due to the large number of features, the training is a process that can take a lot of time. Yet, once the best features are detected, we can use the saved results to create a fast face detector[[9]](#footnote-9).

1. Evaluation
   1. Results

In this implementation of Viola and Jones AdaBoost, a training set of 235 positive images (faces, Figure 8) and 241 negative images (no faces, Figure 9) and a test set of 251 positive (Figure 10) and 259 negative images (Figure 11) have been used. Each was a gray scale 24x24 pixels image. The training program was run 7 times, with an average training time of 8 hours and 30 minutes (in the first training trials, the training set was 135 positive and 135 negative images, and the training time was around 5 hours).

When the training was finished a list of all the weak classifiers with its corresponding weight was created, in where the weight represents the importance of that classifiers “vote” inside of a strong classifier.

In this stage there was a detail that was not expected: some weak classifiers had negative weight value, and others had positive weight value. This was due the creation of the classifiers threshold and the stochastic nature of most of the training values: some classifiers had a lower than 50% rate of success in the training, so in this case, AdaBoost considers that is more useful to “do the opposite of what the classifier suggest”.

|  |  |
| --- | --- |
| C:\Users\ancode\AppData\Local\Microsoft\Windows\INetCache\Content.Word\faces-test.jpg | C:\Users\ancode\AppData\Local\Microsoft\Windows\INetCache\Content.Word\no-faces-test.jpg |
| Figure 8: Faces, training set (235 images) | Figure 9: no faces, training set (241 images) |

|  |  |
| --- | --- |
|  |  |
| Figure 10: Faces, test set (251 images) | Figure 11: no faces, test set (259 images) |
| Due this special case, two different approaches were used to select weak classifiers:   * Select the best classifiers taking in account the biggest absolute weight * Select the best classifiers taking in account the biggest weight | |

|  |  |
| --- | --- |
|  |  |
| Figure 12: total results by image for a strong classifier composed of 1000 weak classifiers (red: strong classifier mistake, blue: strong classifier correct guess ) |  |

* + 1. By biggest absolute weight

A negative weight means that a classifier is worse than luck. For that, AdaBoost considers it can be a good weak classifier if taken the opposite of its recommendation. So, in this case, a boosting algorithm should take in account the “absolute value” in order to select the best weak classifiers. In the current case, the strong classifier has been tested against a testing set of 251 positive and 259 negative images, and a number of 1000 weak classifiers have been selected.

A detail that was observed in Figure 12 is that this strong classifier tends to take all images as “faces”.

|  |
| --- |
|  |
| Figure 13: Rate of success depending of the number of weak classifiers (the graph is shown until 100, due not showing any particular change) |

Figure 13, shows that the percentage of success drops depending of the number of classifiers used, and this is not the expected result. The max number of classifiers used in this case was 1000. The drop has been more or less constant until reaching 70 classifiers, after that, the results are similar, gravitating around 50% of success, that is, no different than luck.

In this approach, the best classifier is the best of the weak classifiers by themselves, without taking in account the boosting. So it can be considerate a failure.

* + 1. By biggest weight

|  |
| --- |
|  |
| Figure 14: total results by image for a strong classifier composed of 1000 weak classifiers  (red: strong classifier mistake, blue: strong classifier correct guess ) |

The second approach is to take the negative values as defective classifiers and ignore them: when choosing the best threshold, its features got values too similar in both faces and non faces, so it was hard to create a real threshold, thus the result being “too bad” by luck. In the other hand, the ones who had a better positive result was because they were easier to classify, thus are better candidate to be weak classifiers.

In Figure 14 the results of this strong classifier in the same conditions of the last case can be seen (testing set of 251 positive and 259 negative images, and 1000 weak classifiers). In this case, the classifier clearly works better than luck. For what can be seen, the strong classifier is especially good in recognize faces, but it finds difficult to classify a non-face.

|  |
| --- |
|  |
| Figure 15: Rate of success depending of the number of weak classifiers |

In Figure 15 one can be the evolution of the strong classifier depending of the number of weak classifiers it has. In this case, the success of the classifier is better than in the case in where the negative weighted weak classifiers were choosen. The strong classifier starts in arround 70% for a small number of classifiers, and drops until the number of weak classifiers reach 60. Then the rate of success grows with each new weak classifier added until 600 classifiers are reach. After that it stays stable around 68-69%.

This trend can mean that, while the firsts 12 weak classifiers are very good, the following ones aren’t suitable to be in the strong classifier and work against the overall boosting until the 60th classifier. Then all the following classifiers work properly together.

1. Conclusions
   1. About the results

After the tests in boths situations (by absolute weight and by weight) is clear that to get by absolute weight is not a suitable approach, at least in this case, because the boosting in the strong classifier doesn’t work – and actually makes it worse. In the other hand, using only weak classifiers with positive weight proved to be more succesful. So **following only the strong classifier made from positive weight weak classifiers** will be considered.

* + 1. Learning process

The strong classifier had a success rate of 70%. This is clearly better than luck, but is less of the results Viola and Jones (and others) have report (with success rates around 80%). When looking to Figure 15, one can see that, the algorithm gave high weights to some weak classifiers that had around the same success rate than the overall strong classifier, but also gave high weights to others that work against the classifier, so is clear that the algorithm can be improved.

Some of the improvements could be:

* The success of a weak classifier depends directly of the method of selecting a proper threshold. If a better method to select the threshold can be found, it will boost the project’s results
* Adaboost is a learning method that highly depends on the order of the features we are using. In viola and jones we have an extensive set of features, and a lot of them are very similar between each others (belong to the same family and are in similar positions). This can make that one feature get a certain weight, and it will penalize the ones from the same “family”. This risks to give a higher punctuation to a feature that is worse than other from its own family. It can be interesting to improve AdaBoost to take this in account. Another option can be to repeat the learning process several times with different orders. However, both approaches would increase the process time of an already long (8 hours) training.
* As seen in the results, the strong classifier finds easy to identify a face, but has problems identifiying a non face. A method to improve this, would be to increase the non faces in the training set.
* Other option that could improve the algorithm is to use a bigger training set overall. However, this will make the learning process to take even more time

|  |
| --- |
| *C:\Users\ancode\AppData\Local\Microsoft\Windows\INetCache\Content.Word\4-examples.jpg* |
| Figure 16: Haar-like features from the weak classifiers with strongest weight |

* + 1. Weak classifier

In Figure 16 an example of the best haar-like features that are used in the best classifiers are shown. Intution says that they show the difference between the background and face, or between hair and face, thus making them specially potent for a set of images sharing this caracteristic (dark background and face or people with hair).

Another thing that was found in the project is that most of the haar-like features with bigger weight belongs the area from the top until the antidiagonal. Those preferences in features could be due the traning set used.

* 1. Conclusions about the mini project

Although this project was interesting, it was especially troublesome for several factors, specially the training. In this some of those problems and the methods used to overcome they will be enounced.

* + 1. Programming issues

As the project was created in Matlab, it was likely to not to be as fast as it could be using other programming languages (for example, C++).

The training process takes too long. At the beginning a training set of 300 images was used, and it took around 5 hours to finish, and the results weren’t satisfactory. After that a bigger training set was used (510 images) that time extended to 8 hours and half. This proved to be quite troublesome for several reasons:

* Matlab sometimes got unstable and stopped the project, so all the training done until that time was lost. In order to avoid this kind of problems, auto-save and load feature were added, so the training saves automatically when is closed, or when it has reached a % of the total result.
* In several occasions Matlab froze in middle of the training, not giving any sign that the training was progressing or not. In order to ensure that the program is still running, a progress bar and time estimation was added. The time estimation proved to be quite accurate.
* Due the extensive time of the training, to debug or to look to the state proved to be quite challenging (print the information in Matlab console was useless and slowed the training considerably). For that reason, an option was added to show a graph with several status information. This proved to be quite useful to check the training behavior and to detect certain bugs. In order to it to not delay the program too much, the graph is updated every 20 cycles.

Also, the project was bigger than expected, thus the complexity of the programming increased exponentially. That made it hard to keep track of the different files and options. In order simplify all the processes (training, testing, checking the weak classifier values or showing the Haar like features from a particular weak classifier), a visual GUI was created.

|  |  |
| --- | --- |
|  | *C:\Users\ancode\AppData\Local\Microsoft\Windows\INetCache\Content.Word\stadistics and estimate time.png* |
| Figure 17: program’s visual GUI | Figure 18: real time statistics and progress bar with time estimation |

* + 1. Theory issues

Viola and Jones paper extended their explanation the Integral Image and Haar-like features concepts a great deal. However, they were not very specific in their implementation of the training (they created their own modification of AdaBoost designed to their project). It is most likely due the method was patented, so they didn’t want to reveal too much detail from the training process, and it was the task of the reader to figure out its own version of AdaBoost. For this reason a closer implementation of the original AdaBoost was used.

Another detail they don’t explain in much detail not in Viola and Jones nor in AdaBoost paper is the process to detect the optimal threshold (they just assume that “the best threshold is selected”), so a custom one was designed. The first version of the threshold had some naïve assumptions: it took for granted that most of the time there would be a real direct separation between the feature values, when actually it was common to have a mixed separation between both of them. Also, it didn’t take in account outliers. For that reason a second version of the best threshold selection was designed, using the standard deviation and the mean value so outliers are ignored[[10]](#footnote-10)

## Index of Figures

[*Figure 1: Examples of rectangles to be used to make features. (Haar-like features)* 2](#_Toc453540636)

[*Figure 2: Examples of several versions of the same type of Haar-like type rectangle figure* 2](#_Toc453540637)

[*Figure 3: Example of the calculation of the Haar-like associated value to a feature inside a 10x7 window* 3](#_Toc453540638)

[*Figure 4: Original image and integral image. In an integral image the value of the luminance of a pixel is the sum of the luminance from the pixels located above and below it.* 3](#_Toc453540639)

[*Figure 5: Calculation of the sum of all pixels intensity in one rectangle from an integral image.* 4](#_Toc453540640)

[*Figure 6: Boosting algorithm diagram* 4](#_Toc453540641)

[*Figure 7: The AdaBoost algorithm as shown in the Viola and Jones paper* (1) 5](#_Toc453540642)

[*Figure 8: Faces, training set (235 images)* 5](#_Toc453540643)

[*Figure 9: no faces, training set (241 images)* 5](#_Toc453540644)

[*Figure 10: Faces, test set (251 images)* 6](#_Toc453540645)

[*Figure 11: no faces, test set (259 images)* 6](#_Toc453540646)

[*Figure 12: total results by image for a strong classifier composed of 1000 weak classifiers (red: strong classifier mistake, blue: strong classifier correct guess )* 6](#_Toc453540647)

[*Figure 13: Rate of success depending of the number of weak classifiers (the graph is shown until 100, due not showing any particular change)* 6](#_Toc453540648)

[*Figure 14: total results by image for a strong classifier composed of 1000 weak classifiers (red: strong classifier mistake, blue: strong classifier correct guess )* 7](#_Toc453540649)

[*Figure 15: Rate of success depending of the number of weak classifiers* 7](#_Toc453540650)

[*Figure 16: Haar-like features from the weak classifiers with strongest weight* 8](#_Toc453540651)

[*Figure 17: program’s visual GUI* 9](#_Toc453540652)

[*Figure 18: real time statistics and progress bar with time estimation* 9](#_Toc453540653)

## Quotations

1. *Robust Real Time Object Detection.* **Jones and Viola.** 2001, 2nd International Workshop on Statistical and Computational Theories of Vision.

2. *AdaBoost Face Detection.* **Masnadi-Shinrazi, Hamed.** s.l. : Department of Electrical and Computer Engineering University of California, San Diego, 2004.

3. **Wikipedia.** Haar-like features. [Online] en.wikipedia.org/wiki/Haar-like\_features.

4. **Tarhini, Ali.** www.codeproject.com. [Online] May 2011. www.codeproject.com/Articles/85113/Efficient-Face-Detection-Algorithm-using-Viola-Jon.

5. **Souza, César de.** www.codeproject.com. [Online] December 2014. www.codeproject.com/Articles/441226/Haar-feature-Object-Detection-in-Csharp.

6. **Hu, Sirui.** *ECE661 Computer Vision HW 9: Object Recognition using Viola-Jones Ada Boost Algorithm.* 2012.

7. **Wytyczak-Partyka, Andrzej.** code.google.com subversion. [Online] Google Summer of Code 2007., Summer 2007. https://code.google.com/p/facedetect-f-spot/source/browse/trunk/MatlabVJDetector/#MatlabVJDetector%253Fstate%253Dclosed.

8. **Mahdi Yektaii and Imad Khoury.** Matlab-based Fast Face Detection. [Online] Computer Vision (COMP358B) Project Winter 2006, 2006. sites.google.com/site/khouryimad/computervision(comp358b)project.

9. **Kroon, Dirk-Jan.** *Implementation of the Detection in the Viola-Jones framework.* 2010.

10. Facebook’s facial recognition software is now as accurate as the human brain, but what now? [Online] ExtremeTech, March 19, 2014. www.extremetech.com/extreme/178777-facebooks-facial-recognition-software-is-now-as-accurate-as-the-human-brain-but-what-now .

1. Social networks like Facebook have always been very interested in improvements in face recognition for tagging (10). [↑](#footnote-ref-1)
2. It should be noted that Viola and Jones patented their face recognition algorithm to commercial use. Some people have created variations that uses the same concepts but applied in different or different algorithms, in order to differ their implementation enough to not to have to pay. [↑](#footnote-ref-2)
3. There is not a real reason for using the black part for subtraction, it could be the opposite, but we will use this notation in this document. [↑](#footnote-ref-3)
4. For an object like a face, that window have enough information to create a proper feature, though for other type of object detection it might be wise to use another window size [↑](#footnote-ref-4)
5. Some sizes for the rectangles don't really work well for being too small so we can use this limitation without being scared [↑](#footnote-ref-5)
6. source of the image: (7) [↑](#footnote-ref-6)
7. source of the image: (7) [↑](#footnote-ref-7)
8. h(x) means to apply the weak classifier to the image x. f(x) is the value of applying the Haar-Like feature to the image x. We must note that the image x is always an integral image. [↑](#footnote-ref-8)
9. During the mini project, we have found some places who offer the detection code with the already saved strong classifier, but not the training code, f.e. (8) [↑](#footnote-ref-9)
10. If well is true that in using the standard deviation as an outlier discriminator is risky (it will make the system ignore the some of the most extreme values, even if they are not outlier), as we had a relatively big set of numbers, the results should not vary too much. However, an improvement of the outlier and the threshold selection can be useful to improve the training procedure. [↑](#footnote-ref-10)