****

***Short Project***

**Design and implementation of a face detection and location technique to crop faces from human images DB.**

Authors:

Michal Szabó / [michal.szabo@estudiant.upc.edu](mailto:michal.szabo@estudiant.upc.edu)

Aleix Granero Molinedo / [aleix.granero@estudiant.upc.edu](mailto:aleix.granero@estudiant.upc.edu)

Professor:

Joan Aranda Lopez / [joan.aranda@upc.edu](mailto:joan.aranda@upc.edu)

12. december 2021

**ABSTRACT**

Facial recognition is a way of identifying or confirming an individual’s identity using their face, and can be used to identify people in photos, videos, or in real-time. The main objective of this project is to design and implement a face detection technique to crop faces from human images database. In order to do that, the Viola and Jones algorithm has been used and many of its steps has been followed to obtain our results.

**INDEX**

[1. Introduction 4](#_Toc90147713)

[2. Viola and Jones algorithm 4](#_Toc90147714)

[3.1 Haar-like features 5](#_Toc90147715)

[3.1.1 Specification 5](#_Toc90147716)

[3.1.2 Detection 5](#_Toc90147717)

[3.2 Integral image 8](#_Toc90147718)

[3.3 Applying Haar features 9](#_Toc90147719)

# 1. Introduction

While recognizing a face appears to be a trivial task for human beings, it is very challenging task for computers. The difficulty associated with face detection can be attributed to many variations in scale, location, viewpoint, illumination, occlusions, etc, and there are many different techniques and algorithms to do this procedure. (Dwivedi, 2018)

One of the very well know algorithms is Viola-Jones algorithm, developed in 2001 by Paul Viola and Michael Jones. The Viola-Jones algorithm is an object-recognition framework that allows the detection of image features in real-time. Despite being an outdated framework, Viola-Jones is quite powerful, and its application has proven to be exceptionally notable in real-time face detection. This algorithm has 4 stages: haar-like features, integral image, adaboost training and cascading classifiers. (Juma, 2021)

# 2.1 Haar-like features

Haar-like features are [digital image](https://en.wikipedia.org/wiki/Digital_image) [features](https://en.wikipedia.org/wiki/Feature_(computer_vision)) used in [object recognition](https://en.wikipedia.org/wiki/Object_recognition). In the present project, Haar-features will be used to detect faces using the Viola and Jones algorithm.

### 2.1.1 Specification

With a human face it is a common observation that among all faces the region of the eyes is darker than the region of the cheeks. Therefore, a common Haar feature for face detection is a set of two adjacent rectangles that lie above the eye and the cheek region. Additionally, a second feature has been selected relying on the property that the eyes are darker than the bridge of the nose.

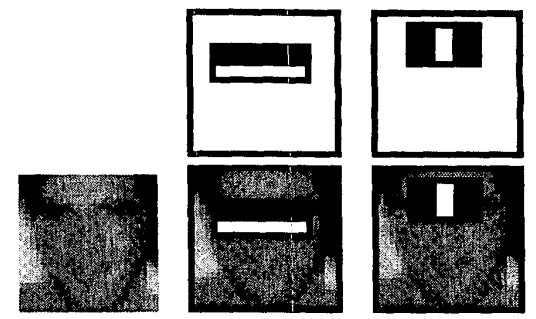


Figure 1 Most common Haar-features in face recognition

### 2.1.2 Detection

Once the Haar-features are defined, we must continue with the computation part, in which we will try to detect them in the images. Logically, it will be practically impossible to find exactly the specified Haar-features, since they are binary and our images are in grayscale, so we will limit ourselves to finding the regions of the images that are most similar.

The detection process consists on applying the Haar-feature in each region of the image and calculate the degree of similarity. This degree of similarity is determined as the difference between the sum of the intensities of the white pixels and the black pixels.

For example, if we want to find the Haar-feature specified in *figure 2* and it is applied in the region of the image specified in *figure 3*, the degree of similarity ∆ can be calculated using the following expression:

Equation. 1: Similarity degree

Knowing that in the ideal case this degree will be 255, we can determine that the region that is closest to this number will be the one that has the closest resemblance to the specified Haar-feature.

|  |  |  |  |
| --- | --- | --- | --- |
| 230 | 200 | 50 | 0 |
| 255 | 230 | 0 | 100 |
| 200 | 230 | 100 | 100 |
| 200 | 230 | 50 | 50 |

|  |  |  |  |
| --- | --- | --- | --- |
| 255 | 255 | 0 | 0 |
| 255 | 255 | 0 | 0 |
| 255 | 255 | 0 | 0 |
| 255 | 255 | 0 | 0 |

Figure. 3: Image region

Figure. 2: Haar-feature specified

In practice we are not going to have only one Haar-feature, so we will search for the region of the image that has the closest resemblance to all the specified Haar-features. This value can be calculated as the product of the ∆ of each of the Haar-features:

Equation 2 Similarity degree for more than one feature

However, if we just specify a Haar-feature with a fixed dimension and look for regions that look like it, our face recognition method is going to get really bad results since the size of faces can change from one image to another. Therefore, it is necessary to change the Haar-feature size and check all the positions in the image for each size. For example, for the previously specified Haar-feature we would follow the following procedure:

Compute all the in each region of 1x4

Resize

Compute all the in each region of 1x2

Initial size

|  |  |  |  |
| --- | --- | --- | --- |
| 255 | 255 | 0 | 0 |

|  |  |
| --- | --- |
| 255 | 0 |

|  |  |
| --- | --- |
| 255 | 0 |
| 255 | 0 |
| 255 | 0 |

Compute all the in each region of 2x2

|  |  |
| --- | --- |
| 255 | 0 |
| 255 | 0 |

Resize

…

Compute all the in each region of 1x6

Resize

Resize

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 255 | 255 | 255 | 0 | 0 | 0 |

Figure. 4: Procedure to compute all Δ

From a theoretical point of view this would be the best method to find the region that most closely resembles the specified Haar-feature, but nevertheless in practice this process is totally unfeasible since computing the ∆ of all possible sizes would require a high computational time. For example, if we are working with a 200x200 image and we want to find the previously specified Haar feature, the number of ∆ that would be calculated would be 201 million.

Another important aspect to highlight is that the Haar-features have different minimum sizes, and grow differently, always under the condition that in each region (whether white or black) there must be the same number of pixels. For example, in the following figure we can see how two different Haar-features would grow.

|  |  |
| --- | --- |
| 255 | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 255 | 255 | 255 | 0 | 0 | 0 |

|  |  |  |  |
| --- | --- | --- | --- |
| 255 | 255 | 0 | 0 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 255 | 255 | 255 | 0 | 0 | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 255 | 255 | 0 | 0 |

|  |  |  |
| --- | --- | --- |
| 0 | 255 | 0 |

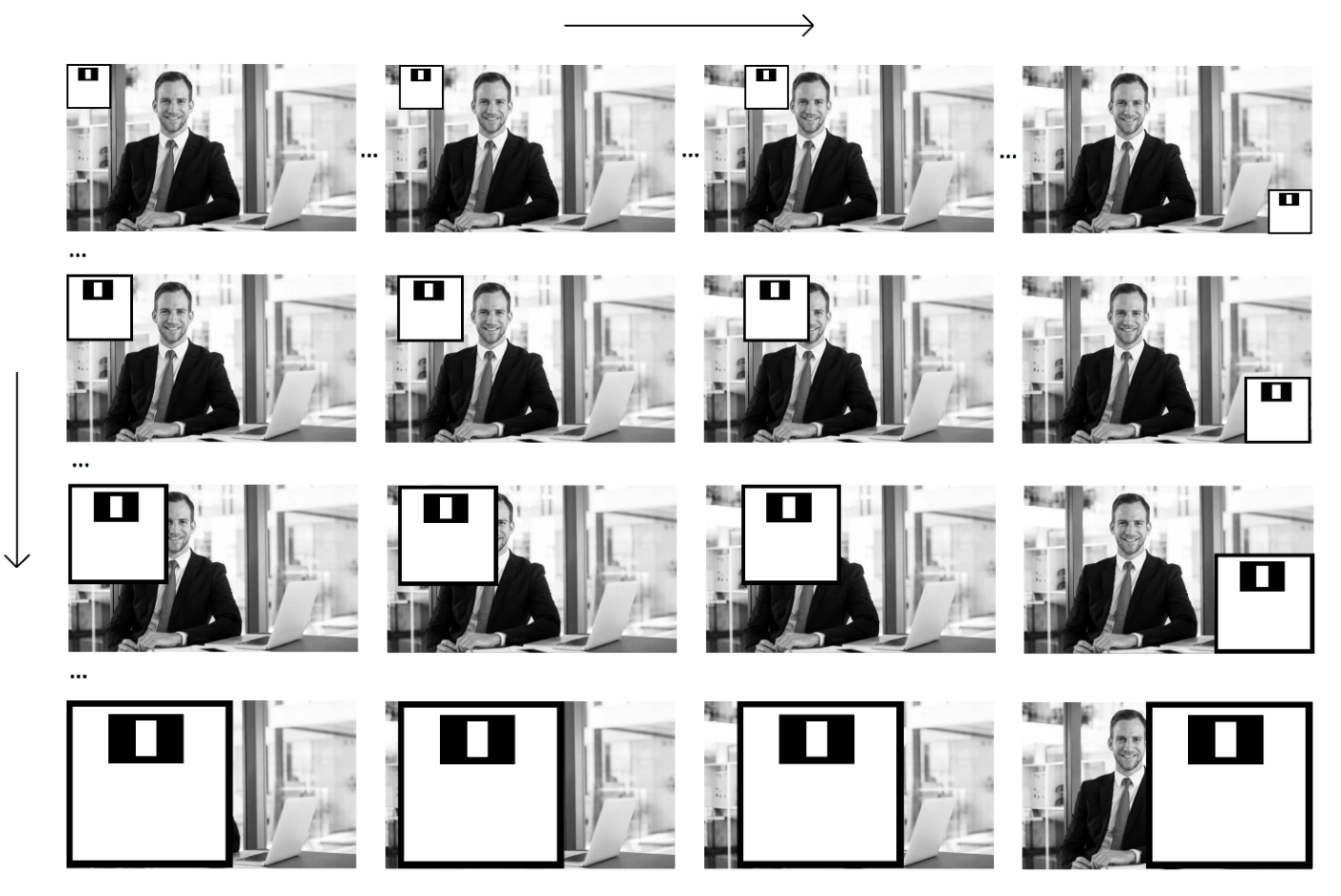
Figure. 5: Increase of two different Haar-features

As can be seen, the second Haar-feature grows faster and therefore the number of ∆ to be calculated would be lower. In the case where we want to find a region of the image where these two Haar-features predominate, this event could become a problem, since we would be working with different sizes.

In order to solve this problem, in the current project a window with initial dimensions 24x24 has been created, which contains inside a Haar-feature with initial dimensions 14x8. This initial window is applied to all the positions of the image to compute the ∆ values, and then it is resized by multiplying its dimensions by 1.25. Then all ∆ values for all positions are recalculated, and so on until the window size exceeds the image dimension.

In the following figure these process can be seen.

Figure 6 Initial size of the window



Changing position (pixel by pixel in x direction)

Changing position (pixel by pixel in y direction)

Figure. 7: Procedure to compute all possible Δ given a specific Haar-feature

# 2.2 Integral image

An Integral image is where each pixel represents the cumulative sum of a corresponding input pixel with all pixels above and left of the input pixel. This algorithm enables rapid calculation of summations over image sub-regions. Any rectangular subset of such sub-region can be evaluated in constant time.

# 3. Applying Haar features from Viola-Jones article

Our first try was to apply the feature that Viola and Jones obtained from the AdaBoost. We designed the Haar features on 24x24 pixel window like they did in the article.

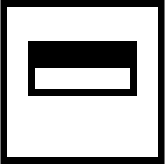
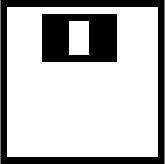
 

Figure. 8: Haar 1 Figure. 9: Haar 2

Then we apply them on picture while changing their sizes. The calculated value was stored in the matrix with the position for the top-left corner pixel. We calculated the intensity value of black and white area using integral image. Then subtracted the black area from the white one and divided it by number of pixels of both areas. We calculated this value for both Haar features and multiplied them so if one is big and another one as well, we get the best position of the face. This is what we obtained.

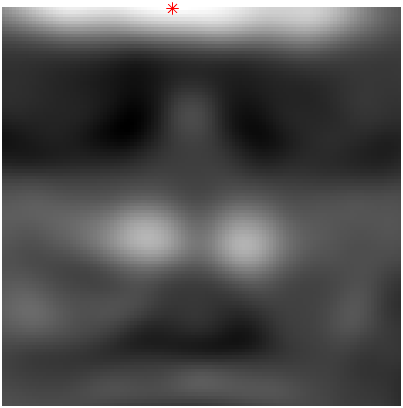
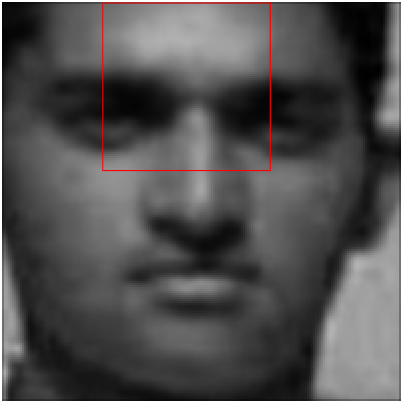
 

Figure. 10: Map Figure. 11: Detection

We figured that something is wrong because these Haar features are calculating the intensity values, so for example the first feature is showing us that the black area should be the dark area of the eyes and the white area should be light area of the cheecks. The darker the area of the eyes is, the smaller the intensity value should be. And the lighter the area of the cheeck is, the larger the intensity value is. But in our results we can see, that his forehead is lighter than the eyebrows so the calculated value should not be that white as we can see in (Fig. 3). What our problem was that, we were multiplying the numbers we obtained from both features. If both of the numbers were negative we got false positive values. So, we changed the code and created a threshold for both features that’s says that everything lower than 0 is 0. This way we obtained these results.

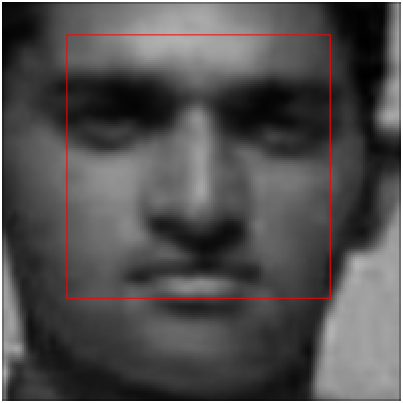
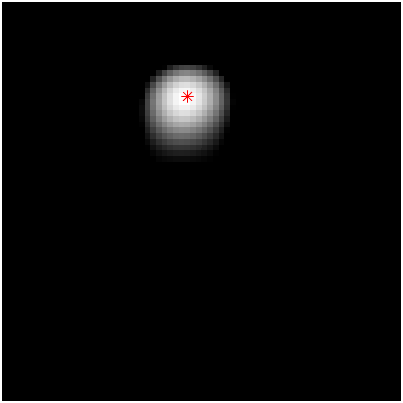


Figure. 12: Map Figure. 13: Detection

These results finally made sense but were not good enough. When we tried to apply this to some other images these were the results we obtained.

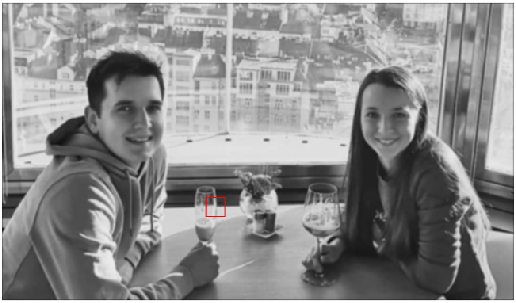
  

Figure. 14: Results with more complicated images

We need to look better at the close images what is happening.

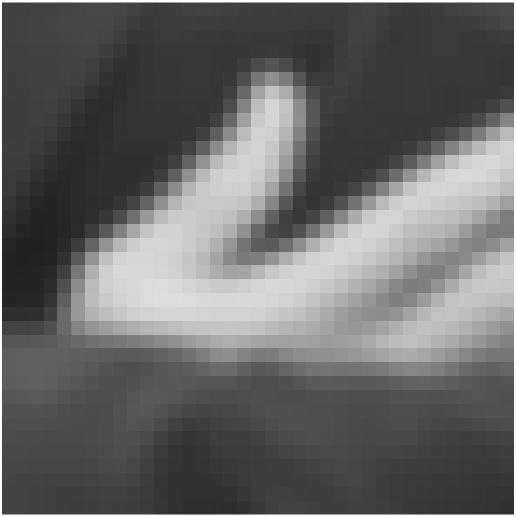
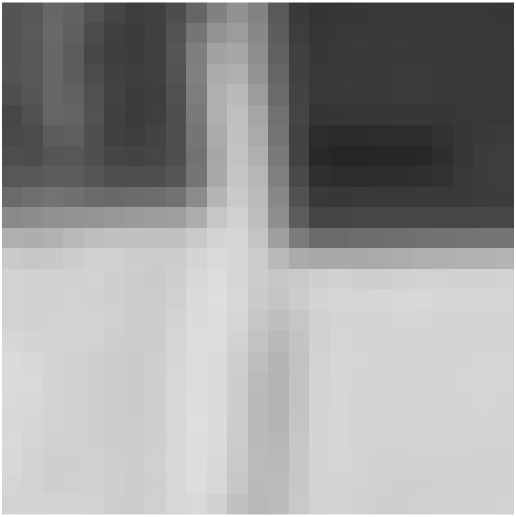


Figure. 15: Cropped images from Fig. 14

When we look at these cropped images, they all fulfil our two haar features. They all have upper part darker then lower at the place of our haar feature and also have a light vertical stripe in the middle. We understood that these two features are not enough. We tried to search for other features that should work, and also tried to see what is actually in Viola and Jones cascade detector but we couldn't find anything. So we decided to train our own features.

## Training Haar features

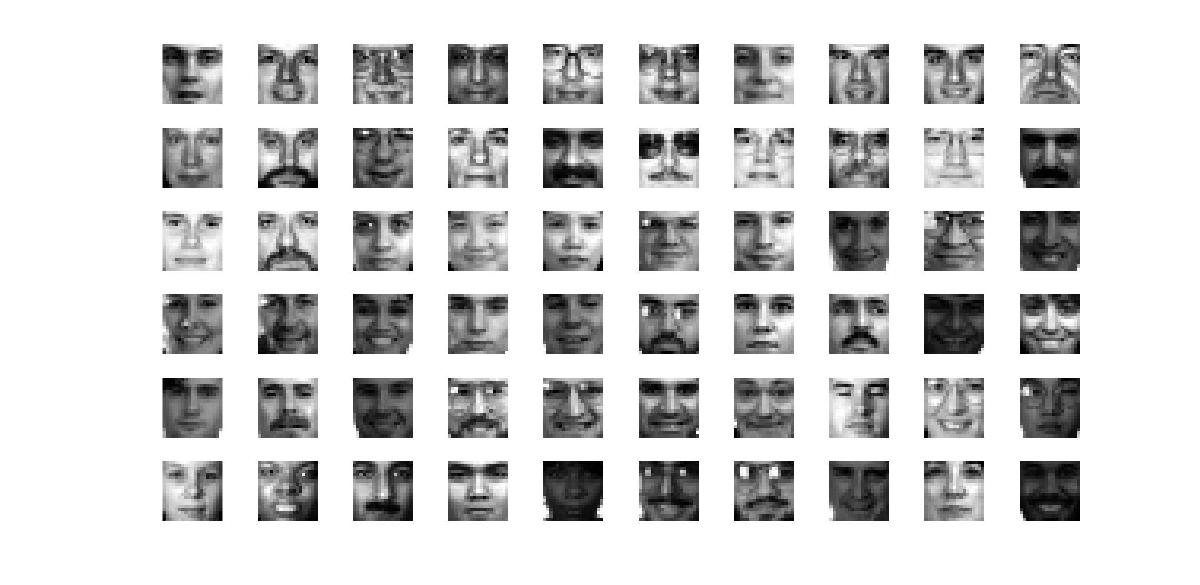
We found database of 19x19 pixel images with more than 2000 examples of face and non-face images. We can show a preview of some pictures from the database.

Figure. 16: Face images database

## Calculating feature values

Figure. 17: Non-face images database

We created an algorithm that computes feature values for all sizes and 5 different types of Haar features. This algorithm is written in Matlab file *trainHaar.m*. We can explain how it works on the following code.

First we define on how many pictures we want to train our Haar features.

% Number of training samples used

faceImages = 300;

nonFaceImages = 300;

Then we load the pictures and save them in separate databases.

fprintf(strcat('\nProcess: \*\*\*Training images inicialization\*\*\*\n'))

% Loading face images

fprintf('Info: Loading face images.\n')

faceIIs = cell(1,faceImages);

for img = 1:faceImages

image\_path = ['TrainingFaces\',int2str(img),'.pgm'];

I = imread(image\_path);

II = integralImage(I);

faceIIs{img} = II;

end

% Loading non-face images

fprintf('Info: Loading non-face images.\n')

nonfaceIIs = cell(1,nonFaceImages);

for img = 1:nonFaceImages

image\_path = ['TrainingNonFaces\',int2str(img),'.pgm'];

I = imread(image\_path);

II = integralImage(I);

nonfaceIIs{img} = II;

end

Now we will initialize our window size which is 19, because our training pictures. For faster but less precise training we can increase *haarSize* which multiplies the size of the Haar features.

% We found database with 19x19 pixel size images, it would be better to use

% 24x24 pixel images because of more compatible feature sizes. In our case

% we dont calculate feature value for last row or column of pixels in with

% certain features

window = 19; % Training Image size

haarSize = 1; % Haar feature size multiplier for faster computation

Then we define haar features which are shown on the pictures below. We can also set our *first\_size* of the haar feature for faster training because we know that the small 2 pixel haar feature is not gonna be very robust.

% 1 2 3 4 5

haars = haarSize\*[1,2; 2,1; 1,3; 3,1; 2,2];

first\_size = [3,2; 2,3; 3,3; 3,3; 4,4]; % Small haars are not robust anyways

Then we define our variable *count* to count how many features we calculated and initialize our database for features and classifiers.

count = 0; % Feature count

featureDatabase = []; % To store calculated features

classifier = []; % To store Haar type and size

And we start calculating our feature values. For each size of the Haar feature we calculate feature value for all face and non-face images so in the end we have number of features for each haar feature size same as the number images we used. We have programed our own function *calcHaarVal* to calculate the value for us. This function takes Integral image, haar type, pixel x and y and size of the haar feature in x and y direction.

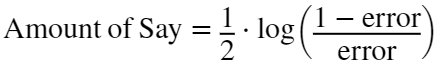
Finally, we have obtained our features and we can start calculating which of the features works the best using AdaBoost.

## AdaBoost

In short words what our AdaBoost does is, that it takes the calculated feature values, for each size and type of the haar feature it sorts the feature values and calculates the threshold using Gini method where we take number between all of the features and see how it divided our values. Gini method looks at how many false positives and false negatives we have for each threshold and calculates Gini value. The threshold with lowest Gini value can separate the features the best. Then we take that threshold and divide our features with it again. We again calculate our Gini value that we now save. We don't have to do it again but it's clearer if we do. From all the Gini values with best threshold calculated for every size and type of feature we take the lowest again to see which type and size of haar feature could separate positive and negative samples. We store 10 best haar features to our topClassifier database. We take the best one and calculate the error it made.



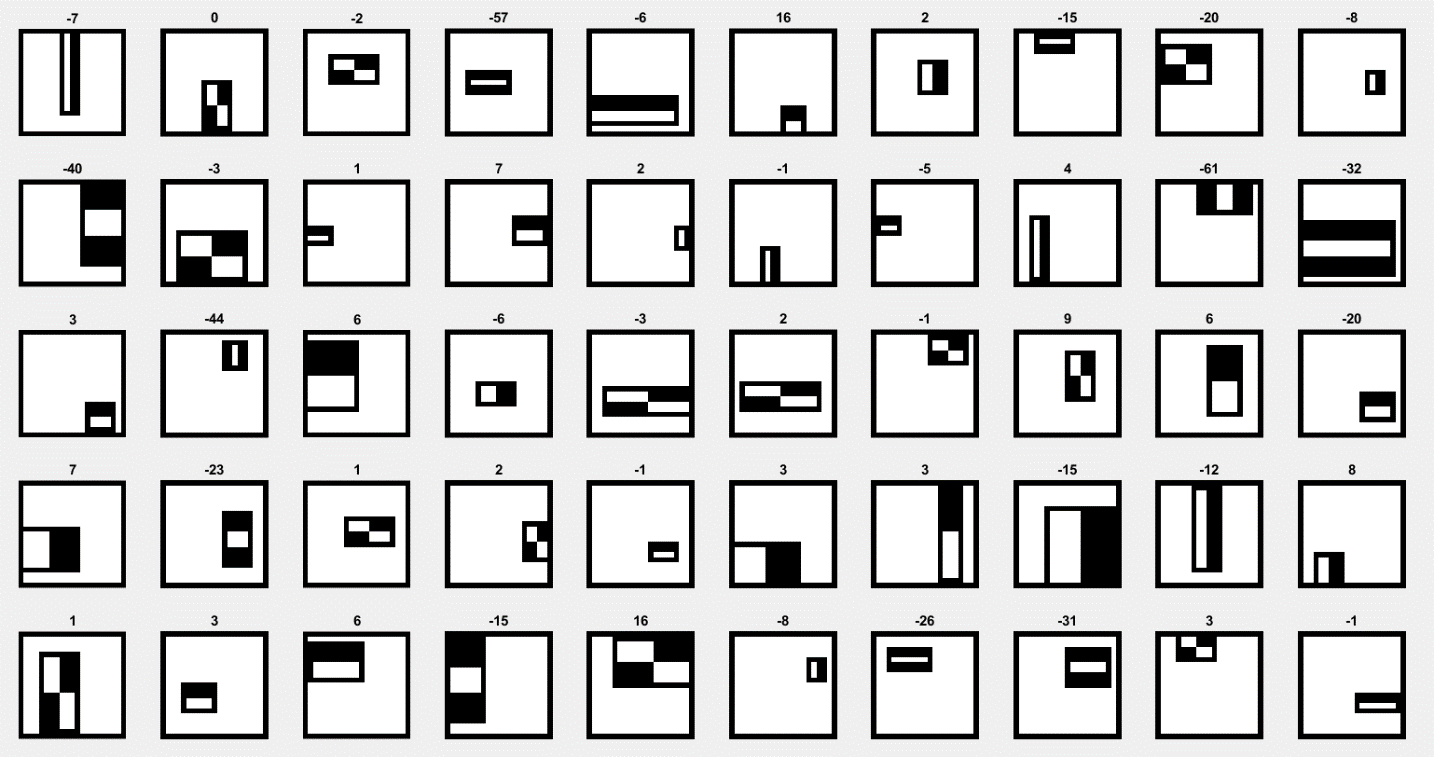
Then using the formula:



Using this Amount of Say we put create new weights where we put more weight to the wrongly classified features. Then we divide them by the sum of new weights so the sum of them would be one. With these new weights we create a new set of feature values where we put more wrongly inclassified features into the feature value database and we repeat the process. We consider that now we have calculated new feature values and we set the same initial weight as before. This is repeated by the number of iterations we set. In the end we save our top classifiers so we can work with them.

We have tried to explain everything in the Matlab file *trainHaar* using comments and everything should be clear.

## Obtained Haar features

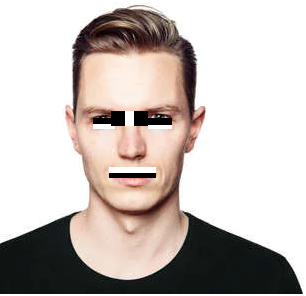
These are the results we obtained when using 300 face images and 300 non-face images with 50 AdaBoost iterations. We display best haar feature from each iteration with their threshold on top.

We can see that some of the features we obtained are similar to the ones Viola and Jones obtained in their article. The results are dependent on the training images as well as how the new features are created in AdaBoost. Now we can try to apply some of the features we got to create our own cascade and detect images.

## Face Detection

To apply face detection, we created new Matlab file called *detectFaces*. At the beginning of this file we can see the parameters that describe the Haar features used in our method.

After many tries, the final Haar-features that we are going to use will be the following:



The method starts with a loop where we pick up an then we compute its size.

for img = 1:images

fprintf(strcat('Info: Detecting faces on image:',int2str(img),'\n'))

I\_features = {}; % We create a new DB for features for every img

[row,col] = size(Is{img}); % Size of the image

maxsize = floor(min([row/24,col/24])); % Max size for haar feature

window = []; % window is our haar feature size

Then, a loop is created in order to resize the window from the minimum (24x24) to the maximum, that will depend on the image size.

for haarSize = 2:0.5:maxsize % Every iteration we make window larger by 0.5

hsize = hsize+1; % Counting window changes

window(hsize) = floor(haarSize\*start\_haar); % Changing window size

fval = zeros(row-window(hsize)+1,col-window(hsize)+1); % Initialize matrix

After, we have to move this window in all the positions of the image an in order to do that the following code is executed.

for i = 1:row-window(hsize)+1 % +1 pixel because we

for j = 1:col-window(hsize)+1 % substracted it

fval(i,j) = ourCascade(IIs{img}, haarSize, i, j,features);

end

end

I\_features{hsize} = fval; % Saving feature matrix for each size of the haar

As it can be seen in the previous code the function *ourCascade* is called, which have the necessary code to compute the values of all the different features defined for a given window size and position. In order to do that, the expressions previously seen and the integral image are used. At the end, as we have seen also in the theoretical part, all the values of the features are multiplied in order to obtain only one value.

val = val1\*val2\*val3\*val4

However, one of these values can be negative if the Haar-feature has no presence with the size and position specified, and that’s why a threshold has been created. If the value is negative but bigger than the threshold, the value will be one and if not, it will be zero.

if val1<=0

if val1>THR

val1 = 1;

else

val1 = 0;

end

end

After obtaining the total value for all the sizes and positions, we have to obtain the best one, that will be the highest value.

bestmatch = []; % Inicialization

% For every size of the feature we find the highest value

for f=1:length(I\_features)

bestmatch(f) = max(max(I\_features{f})); % Save the highest value

if bestmatch(f)==0 % if there is no match

i(f)=0; % We set the coordinates to 0 just to fill matrix

j(f)=0;

else % We save the original coordinates of the highest value

[i(f),j(f)] = find(I\_features{f}==bestmatch(f));

end

end

% Finding the best from all sizes of the window

best = max(bestmatch);

p = find(bestmatch==best);

p = p(1); % if there is multiple best findings

At the end, knowing the position of the window and its size for the best value obtained, the images can be cropped and saved in the directory that we desire.

% Croping the detected window from the image if we find the face

if i(p)==0 && j(p)==0

fprintf(strcat('Info: Could not find a face in image:',int2str(img),'\n'))

else

crop = imcrop(Is{img},[j(p),i(p),window(p),window(p)]);

folder = ['C:\Users\User\Desktop\Project\New\'];

fname = fullfile(folder, ['crop' num2str(img) '.jpg']);

imwrite(crop,fname);

% Plotting all croped images

subplot(3,4,img), imshow(crop)

## Final results

## Conclusions

For our short project which was to design an implementation of a face detection and location technique to crop faces from human images DB, we have done the following:

We were trying to use Viola-Jones algorithm to detect faces in our images. Using only the two features from their article we obtained good results for very simple face images. For comlicated images we found areas that correspond to the haar features but, those areas were not the faces.

In order to find out which another haar features we should use we tried to train our own haar features. We found a database of 19x19 pixels face and non-face images. We created an algorithm to calculate feature values for all the possible sizes and for five different types of haar features. We calculated feature values for 300 face images and 300 non-face images. After that we applied AdaBoost learning algorithm to obtain better haar features. We obtained some features that were similar to the features that Viola and Jones obtained but it really depends our training images and also creation of new feature values in adaboost. The best haar features and their threshold are in Fig. ?????????.

The last step was trying different haar features and creating our cascade. We found out that the haar features that Viola-Jones obtained work well in the first 2 cascades, eliminating a lot of areas without faces. Then we were trying to create smalle features from these two, to detect righ eye, left eye and also the mouth. We tried to apply some haar features that we obtained from AdaBoost. We were also trying to change the threshold but it is very complicated to see the difference. In the end we did not found an optimal solution to find faces in complicated images. For images with just faces its okay. For images that have a complicated background we did not obtain any good results. We tried to change the initial size of the haar feature which worked well, but its complicated to change a size according to the size of the image, when the image can be small with just a face and have a lot of pixels or it can be big with a lot of faces but with less pixels.

What we think that we should do is try to train AdaBoost with more positive and negative images and obtain better results. Also these images should be a very good dataset and with even number of pixels in order to have more possible sizes of haar feature. Then try as many possible options of haar cascades as possible to obtain good results. This is very hard and complicated trial and error task, and maybe its possible to create an algorithm as well, with already good faceDetector to check if the result is positive or negative.