***Face and Eye Recognition***

* *Rus Rares Tudor*
* *Group 30432*

*Contents:*

1. *Introduction . . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .*
   1. *Context. . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .*
   2. *Specifications. . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .*
   3. *Objectives. . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .*
2. *Bibliographic study . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .*
3. *Analysis. . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .*
4. *Design . . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .*
5. *Implementation . . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .*
6. *Testing and validation . . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .*
7. *Conclusions . . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .*
8. *Bibliography. . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .*

1. Introduction

1.1 Context

Face detection is a type of application classified under “computer vision” technology. It is the process in which algorithms are developed and trained to properly locate faces or objects (in object detection, a related system), in images. These can be in real time from a video camera or from photographs. An example where this technology is used are in airport security systems. In order to recognize a face, the camera software must first detect it and identify the features before making an identification.

The goal of this project is to design and implement a system for locating faces in digital images. These are in JPEG format only. Before we continue, we must differentiate between face recognition and face detection. They are not the same, but one depends on the other. In this case face recognition needs face detection for making an identification to “recognize” a face. I will only cover face detection.

1.2 Specifications

Face detection uses classifiers, which are algorithms that detects what is either a face (1) or not a face (0) in an image. Classifiers have been trained to detect faces using thousands to millions of images in order to get more accuracy. OpenCV uses two types of classifiers, LBP (Local Binary Pattern) and Haar Cascades. OpenCV is the most popular library for computer vision. Originally written in C/C++, it now provides bindings for Python.

OpenCV uses machine learning algorithms to search for faces within a picture. Because faces are so complicated, there isn’t one simple test that will tell you if it found a face or not. Instead, there are thousands of small patterns and features that must be matched. The algorithms break the task of identifying the face into thousands of smaller, bite-sized tasks, each of which is easy to solve. These tasks are also called classifiers. For something like a face, you might have 6,000 or more classifiers, all of which must match for a face to be detected (within error limits, of course). But therein lies the problem: for face detection, the algorithm starts at the top left of a picture and moves down across small blocks of data, looking at each block, constantly asking, “Is this a face? … Is this a face? … Is this a face?” Since there are 6,000 or more tests per block, you might have millions of calculations to do, which will grind your computer to a halt.

1.3 Objectives

The purpose of this project is to design a face recognition algorithm that can recognize faces in manipulated images. Also is to develop a very efficient algorithm, in terms of low computational complexity, with the maximum number of face detection and the minimum number of false alarms. The final step is to determine real faces from the face candidates.

2. Bibliographic study

Haar Cascade Algorithm, which it will be used in this project , is based on the Haar Wavelet technique to analyze pixels in the image into squares by function. This uses machine learning techniques to get a high degree of accuracy from what is called “training data”. This uses “integral image” concepts to compute the “features” detected. Haar Cascades use the Adaboost learning algorithm which selects a small number of important features from a large set to give an efficient result of classifiers.

The algorithm has four stages:

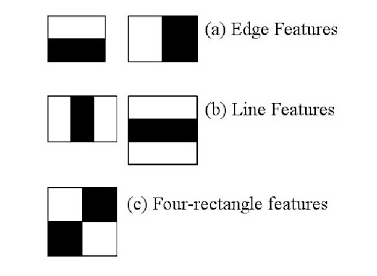
1. Haar Feature Selection

2. Creating Integral Images

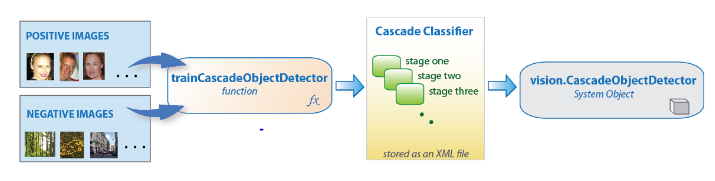
3. Adaboost Training

4. Cascading Classifiers

Initially, the algorithm needs a lot of positive images of faces and negative images without faces to train the classifier.   Then we need to extract features from it.

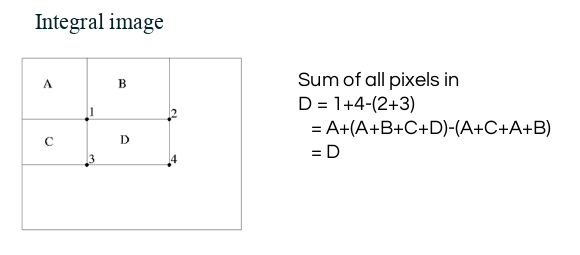
First step is to collect the Haar Features.  A Haar​ feature considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums.

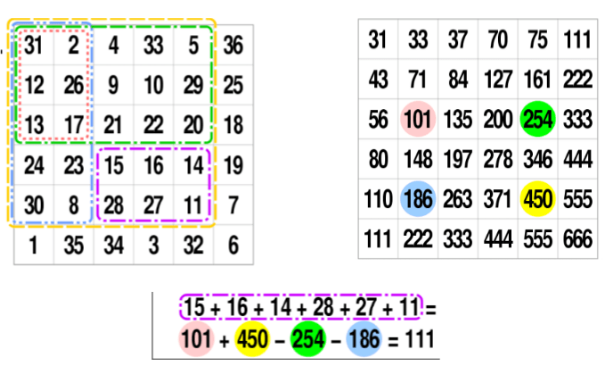
After that, to select the best features, Adaboost concept is used for it. This algorithm constructs a “strong” classifier as a linear combination of weighted simple “weak” classifiers. Because each Haar feature is only a "weak classifier" (its detection quality is slightly better than random guessing) a large number of Haar features are necessary to describe an object with sufficient accuracy and are therefore organized into cascade classifiers to form a strong classifier.

The cascade classifier consists of a collection of stages, where each stage is an ensemble of weak learners. The weak learners are simple classifiers called *decision stumps*. Each stage is trained using a technique called boosting. *Boosting* provides the ability to train a highly accurate classifier by taking a weighted average of the decisions made by the weak learners. 

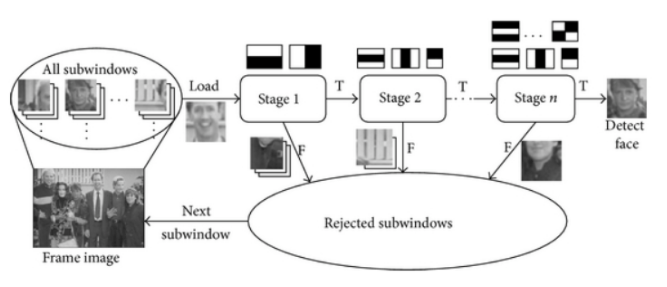
3. Analysis

By analyzing the project resolution requirements of face recognition, we see that in order to design our goal we need to follow some steps in order to “train” our face detector.

We need to understand the Integral Images concept. The algorithm proposed by Viola Jones uses a 24X24 base window size, and that would result in more than 180,000 features being calculated in this window. Imagine calculating the pixel difference for all the features? The solution devised for this computationally intensive process is to go for the Integral Image concept. The integral image means that to find the sum of all pixels under any rectangle, we simply need the four corner values. 

This means, to calculate the sum of pixels in any feature window, we do not need to sum them up individually. All we need is to calculate the integral image using the 4 corner values. The example below will make the process transparent. 

We, also need to improve classifier accuracy using Adaboost. As pointed out above, more than 180,000 features values result within a 24X24 window. However, not all features are useful for identifying a face. To only select the best feature out of the entire chunk, a machine learning algorithm called Adaboost is used. What it essentially does is that it selects only those features that help to improve the classifier accuracy. It does so by constructing a strong classifier which is a linear combination of a number of weak classifiers. This reduces the amount of features drastically to around 6000 from around 180,000.



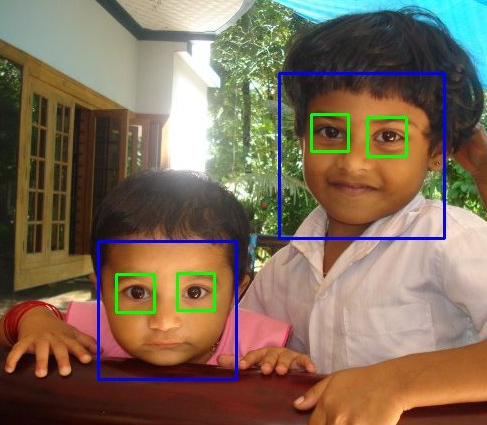
4. Design

We will use OpenCV library in order to detect faces in image, because OpenCV loads an image in BGR format, we will need to convert it into RBG format to be able to display its true colors, for that we will use a predefined function called cvtColor, where the last parameter is cv2.COLOR\_BGR2RGB.

OpenCV comes with a lot of pre-trained classifiers. For instance, there are classifiers for smile, eyes, face, etc. These come in the form of xml files and are located in the OpenCV/ data/ haarcascades/ folder. So we will need to add an trained classifier, if we need one whom we train we can do this too.

After, the classifiers are loaded, then we will need to add out input image. After the image is loaded and a face is founded, it returns the position of detected faces as Rect(x,y,w,h). Once we get these locations we can create a ROI for the face and apply eye detection on this ROI(because eyes are always on the face).

Results should look like below:



5. Implementation

Before implementation I installed python, and OpenCV.

The project contains the following files:

- face\_detect.py

In this file, the algorism is coded.



The imported modules are cv2, which will make all the functionalities of the OpenCV library and sys which handles the system functionalities.



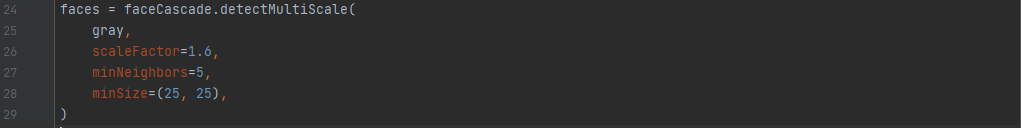
In the “imagePath” variable, is saved the path to the image that will be used by the program to detect faces and eyes, given as a command parameter. “cascPath” and “eyePath” variables are used to save the path to the Haar Cascade training data.



“faceCascade” and “eyeCascade” are the cascade classifiers are saved as already trained classifiers(presented in the .xml files).



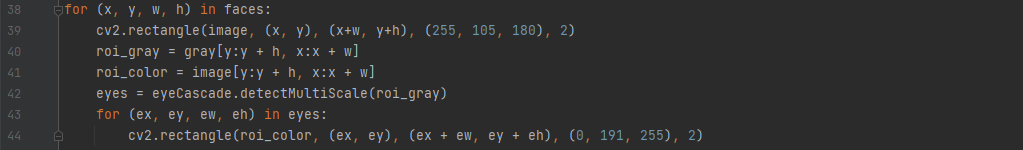
In the “image” variable is saved the image given as an input, and “grey” convert it to the grayscale, because Haar Cascade algorithm only works on grayscale pictures.



The variable “faces” is used to detect faces in the image given as input. In order to do that, I used the function “detectMultiScale()” function, in which 4 parameters are given “gray, the grayscale image, “scaleFactor”, the scale ratio for the faces detected, “minNeighbors”, who defines how many objects are detecte3d near the current one before it declares the face found and “minSize” the size of each moving objects.



After that we print in the command prompt how many faces were detected.



In the first “for” the program draws purple rectangles around the faces, using OpenCV’s functions and also in the second “for” it draws rectangles yellow around eyes.

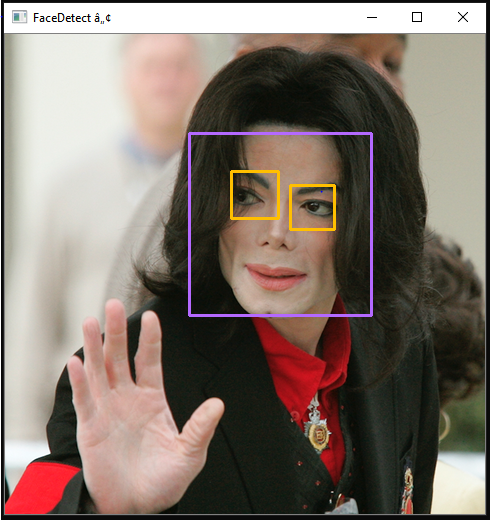
- haarcascade\_frontalface\_default.xml & haarcascade\_eye.xml

Harr Cascade training data.

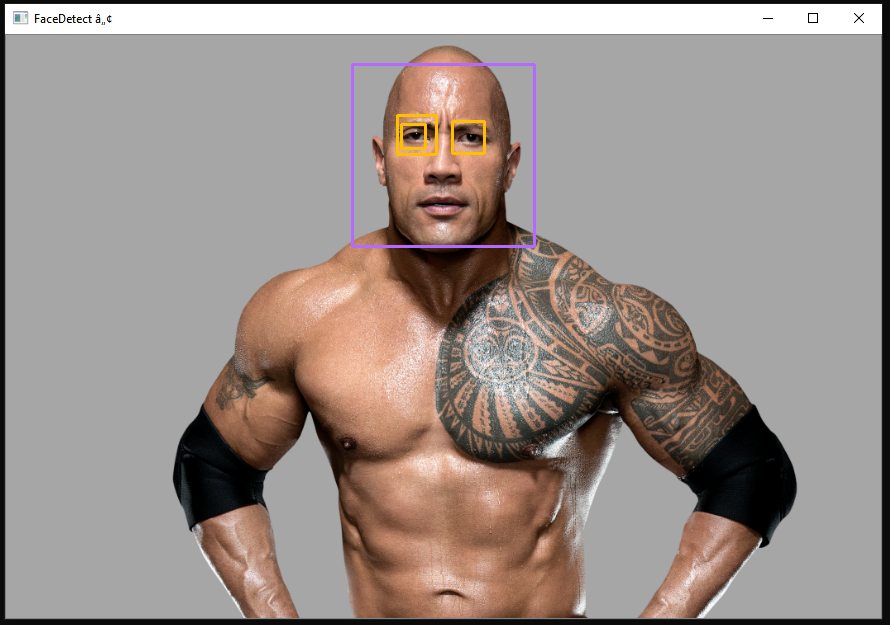
- photos

6. Testing and Validation

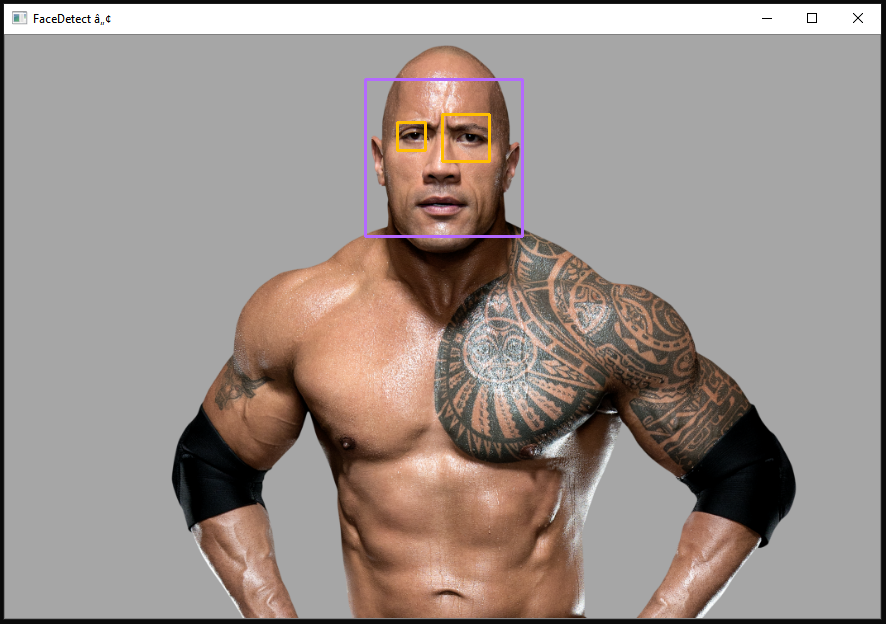
In order to test and to get results, we have to open command prompt in the file where the project is and type “python face\_detect.py X.png”, where face\_detect.py is the name of our project and X is the name of the photo we want to check.

Our first example is this python face\_detect.py mj1.png and the result is: 

Another test would be python face\_detect.py mj1.png and the result is:



 Let’s try again. I changed the parameters and found that setting the scaleFactor to 1.6 got rid of the wrong eye square.



7. Conclusions

Since this project is based on machine learning, the results will never be 100% accurate. We will get good enough results in most cases, but occasionally the algorithm will identify incorrect objects as faces.

In any case, the project was one that aroused my curiosity about how the computer works and understands digital data. I am satisfied because I managed to better understand the material taught in the laboratories.

8. Bibliography

* <http://www.willberger.org/cascade-haar-explained/>
* <https://opencv.org/>
* <https://pythonprogramming.net/haar-cascade-face-eye-detection-python-opencv-tutorial/>
* <https://realpython.com/face-recognition-with-python/>
* <https://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/>