**Face feature detection**

**Executive summery**

In this project I implement a face detection program using C++ and OpenCV for stills image.

In each picture the program will find all faces, and on each face it will mark the nose, eyes and lips on top of the original picture.

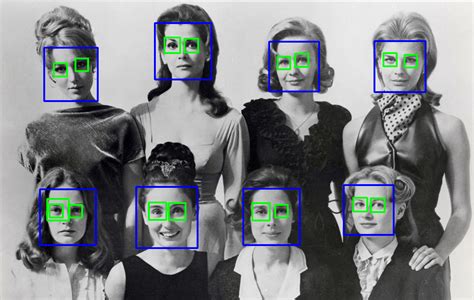
The main engine of this program is based on Haar cascade classifications, where each feature (face, nose etc.) will be detected by a different classification pretrained model that I found online.

**Alternatives overview (among many)**

1. Haar feature-based cascade classifiers *-* an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. An ML based approach where a cascade function is trained from positive and negative images (i.e. with and without the sought feature).

***Advantages*** *- Fast, easy to apply and OpenCV integrated*

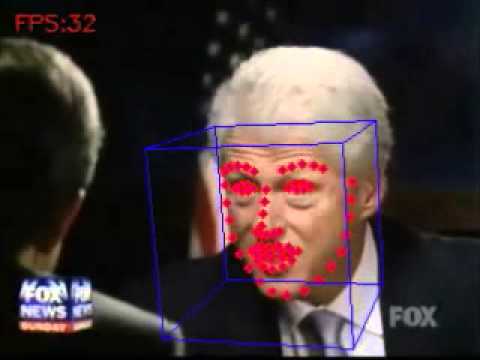
***Disadvantages*** *- less accurate compares to other approaches, can be a pain to tune parameters.*



1. *HOG + Linear SVM: Typically, more accurate than Haar cascades with less false positives. Normally less parameters to tune at test time. Can be slow compared to Haar cascades.*
2. **Deep learning-based detectors -** 
   1. Dlib - A collection of miscellaneous algorithms in Machine Learning, Computer Vision, Image Processing, and Linear Algebra.

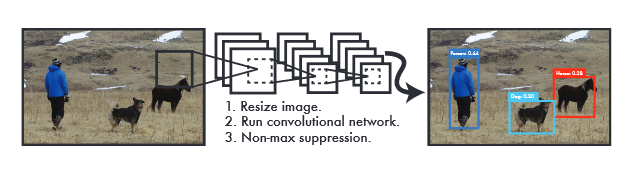


* 1. OpenFace - an open source facial behavior analysis toolkit intended for facial landmark detection, head pose estimation, facial action unit recognition, and eye-gaze estimation.



1. YOLO - real time feature detection base on an CNN, good for applying a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

The model has several advantages over classifier-based systems. It looks at the whole image at test time, so its predictions are informed by global context in the image. It also makes predictions with a single network evaluation which makes it extremely fast.



From the above, in this code I’ll use the Harr cascade classifier, it is an elegant solution and it will give us good enough results.

**Prerequisites *(this code is written with the following preinstall environments)***

OpenCV 3.4.4

Visual studio 2017 community

**Graphical presentation of the algorithm:**

*Block diagram explaining the algorithm*

**Main blocks in the algorithm:**

Define the classifier argument and model path. load the model

CascadeClassifier nose\_cascade;

String nose\_cascade\_name = "C:\\OpenCV3\\opencv\\sources\\data\\haarcascades\\haarcascade\_mcs\_nose.xml";

if (!nose\_cascade.load(nose\_cascade\_name))

{

std::cout << "Error loading nose cascade\n";

return -1;

};

Mat frame = imread("Matan.jpg", IMREAD\_COLOR);

Load the picture

Mat frame\_gray;

cvtColor(frame, frame\_gray, COLOR\_BGR2GRAY);

equalizeHist(frame\_gray, frame\_gray);

Convert the picture to gray scale

Detect all the faces in the picture in return will get Vector of rectangles where each rectangle contains the detected object

std::vector<Rect> faces;

face\_cascade.detectMultiScale(frame\_gray, faces);

Mark with circle all the faces in the picture

for (size\_t i = 0; i < faces.size(); i++)

{

Point center(faces[i].x + faces[i].width / 2, faces[i].y + faces[i].height / 2);

ellipse(frame, center, Size(faces[i].width / 2, faces[i].height / 2), 0, 0, 360, Scalar(0, 0, 255), 4);

In the same “face loop” detect specific features using Haar classifier

std::vector<Rect> nose;

nose\_cascade.detectMultiScale(faceROI, nose, 1.3, 5, 0 | CV\_HAAR\_FIND\_BIGGEST\_OBJECT);

for (size\_t j = 0; j < eyes.size(); j++)

{

Point nose\_center(faces[i].x + nose[j].x + nose[j].width / 2, faces[i].y + nose[j].y + nose[j].height / 2);

int radius = cvRound((nose[j].width + nose[j].height)\*0.025);

circle(frame, nose\_center, radius, Scalar(255, 0, 0), 4);

}

}

Plot and save the picture with the added features

imshow("Face feature detected", frame);

waitKey(0);

imwrite("Face\_detection.jpg", frame);

**Cascade Classifier - *detectMultiScale* Parameters (for tuning purposes)**

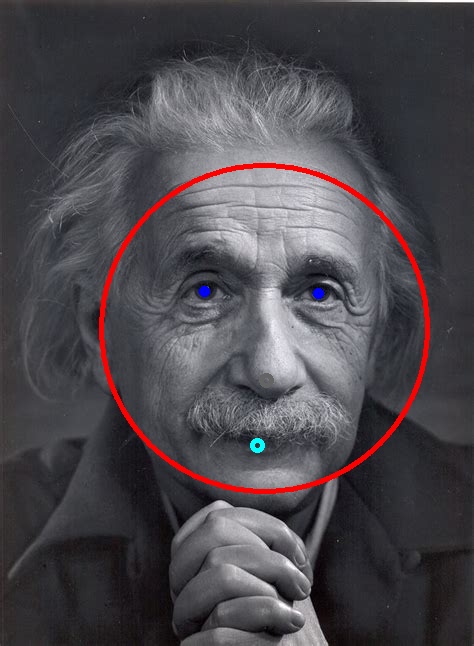
|  |  |
| --- | --- |
| image | Matrix of the type CV\_8U containing an image where objects are detected. |
| objects | Vector of rectangles where each rectangle contains the detected object, the rectangles may be partially outside the original image. |
| numDetections | Vector of detection numbers for the corresponding objects. An object's number of detections is the number of neighboring positively classified rectangles that were joined together to form the object. |
| scaleFactor | Parameter specifying how much the image size is reduced at each image scale. |
| minNeighbors | Parameter specifying how many neighbors each candidate rectangle should have to retain it. |
| minSize | Minimum possible object size. Objects smaller than that are ignored. |
| maxSize | Maximum possible object size. Objects larger than that are ignored. If maxSize == minSize model is evaluated on single scale. |

**Results**

In the picture below, we can see a late-night photo of me with the features marked on it.

Blue dots mark the eyes, gray for the nose and greenish for the mouth.



**Conclusion**

To get better classification results we can use more training pictures, if we have some prior knowledge about the final purpose (e.g. person pose) we can bring more relevant data as well.

preprocessing of the data and the input (e.g. filtering) might improve the classification results.

In aim to speed up the classification we can use GPU for multithread computation (run classification in parallel).

DL, as we discuss in the alternatives review, will probably derive a better classification result for more advanced applications (feature-wise).

**References**

* Haar Cascade Classifier code:
  + <https://docs.opencv.org/3.4.4/d5/d54/group__objdetect.html>
  + <http://alereimondo.no-ip.org/OpenCV/34>
  + <https://github.com/opencv/opencv/tree/master/data/haarcascades>
  + <http://www.willberger.org/cascade-haar-explained/>
  + <https://docs.opencv.org/3.4.4/d7/d8b/tutorial_py_face_detection.html> (OpenCV Python tutorial)
  + <http://comp3204.ecs.soton.ac.uk/cw/viola04ijcv.pdf>
  + <https://www.youtube.com/watch?time_continue=216&v=hPCTwxF0qf4>
* **OpenFace** 
  + <https://github.com/TadasBaltrusaitis/OpenFace>
  + <http://elijah.cs.cmu.edu/DOCS/CMU-CS-16-118.pdf>
* **Dlib** -
  + <http://dlib.net/>
  + <https://en.wikipedia.org/wiki/Dlib>
  + <https://github.com/davisking/dlib>
* **YOLO** 
  + <https://github.com/opencv/opencv/blob/3.4/samples/dnn/object_detection.cpp>
  + <https://pjreddie.com/darknet/yolo/>
  + <https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006>
  + <https://arxiv.org/pdf/1506.02640.pdf>

**Appendices**

Haar feature classifier explained in detail

Haar Cascade is a machine learning object detection algorithm used to identify objects in an image or video and based on the concept of ​​ features proposed by Paul Viola and Michael Jones in their paper "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001.

Initially, the algorithm needs a lot of positive images (i.e. images of faces) and negative images (i.e. images without faces) to train the classifier. From these pictures we can extract the features later sought by the classifier.

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

But among all these features we calculated, most of them are irrelevant. To choose the most relevant ones we are using a concept called Adaboost which selects the best features and trains the classifiers that use them. This algorithm constructs a “strong” classifier as a linear combination of weighted, simple “weak” classifiers.

During the detection phase, a window of the target size is moved over the input image, and for each subsection of the image the Haar features are calculated. This difference is then compared to a learned threshold that separates non-objects from objects. Because each Haar feature is only a "weak classifier”, many Haar features are necessary to describe an object with sufficient accuracy and are therefore organized into cascade classifiers to form a strong classifier.

