**Face feature detection**

**Executive summery**

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

In etch picture the code will find all the faces and in each face, It will mark the nose eyes and lips and draw them on the screen.

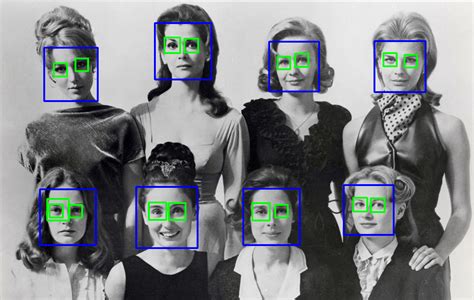
The main engine of this program is based on Haar cascade classifications, each feature (face, nose etc.) will detect by a different classification pretrained model that I found online.

**Options review**

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. ML based approach where a cascade function is trained from a positive and negative images.

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

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



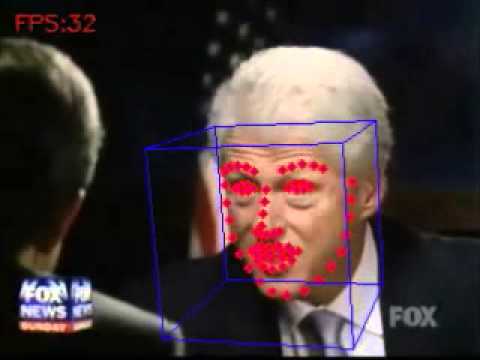
*HOG + Linear SVM: Typically, more accurate than Haar cascades with less false positives. Normally less parameters to tune at test time. Can be slow compared to Haar cascades.*

***Deep learning-based detectors -***

***Dlib*** *-* A collection of miscellaneous algorithms in Machine Learning, Computer Vision, Image Processing, and Linear Algebra.



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

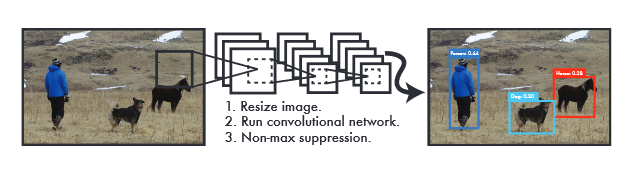


***YOLO*** *- real time feature detection base on an CNN, good for*

Appling 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

This makes it extremely fast, more than 1000x faster than R-CNN and 100x faster than [Fast R-CNN](https://github.com/rbgirshick/fast-rcnn).



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

OpenCV 3.4.4

Visual studio 2017 community

**Code - in graph form**

*Block diagram explaining the algorithm*

**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. |

**Haar feature classifier explain 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 (e.g. images of faces) and negative images (e.g. images without faces) to train the classifier. From these pictures we can extract the features.

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.

But among all these features we calculated, most of them are irrelevant. To choose the most relevant 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 and 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” 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.*

Each stage of the classifier labels the region defined by the current location of the sliding window as either positive or negative. Positive indicates that an object was found and negative indicates no objects were found. If the label is negative, the classification of this region is complete, and the detector slides the window to the next location. If the label is positive, the classifier passes the region to the next stage. The detector reports an object found at the current window location when the final stage classifies the region as positive.

To work well, each stage in the cascade must have a low false negative rate. If a stage incorrectly labels an object as negative, the classification stops, and you cannot correct the mistake. However, each stage can have a high false positive rate. Even if the detector incorrectly labels a nonobject as positive, you can correct the mistake in subsequent stages. Adding more stages reduces the overall false positive rate, but it also reduces the overall true positive rate.



**Results**

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

Blue dot marks the eyes, gray for the nose and greenish for the mouth.



**Conclusion**

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

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

*How to speed up the classification we can use GPU for multithreds computation.*

*DL as we discose in the Options review probably bring a better classification result for more advense applications.*

**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>