**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 etch face It will mark the nose eyes and lips and draw them on the screen.

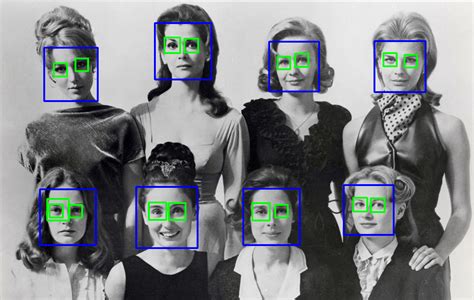
The main engine of this program is based on Haar cascade classifications, each feature (face, nose etc.) will detect by a different classification pretrained model that I found online.

**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.*



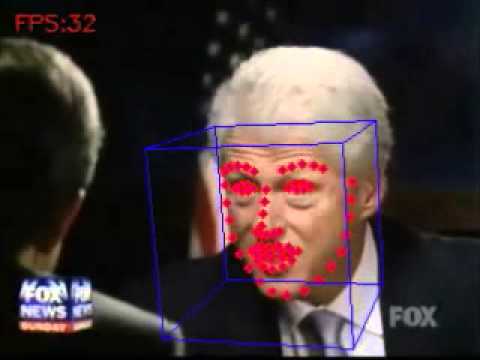
*HOG + Linear SVM: Typically, more accurate than Haar cascades with less false positives. Normally less parameters to tune at test time. Can be slow compared to Haar cascades.*

***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.

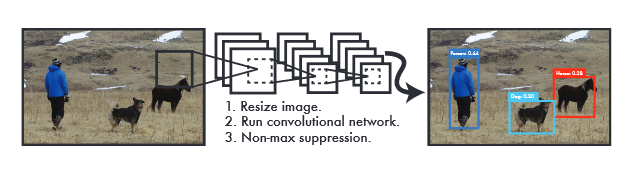


***YOLO*** *- real time feature detection base on an CNN, good for*

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**

*Block diagram explaining the algorithm*

**Cascade Classifier - *detectMultiScale* Parameters**

|  |  |
| --- | --- |
| 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. |
| flags | Parameter with the same meaning for an old cascade as in the function cvHaarDetectObjects. It is not used for a new cascade. |
| 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 details**

Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, Haar features shown in the below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.



image

Now, all possible sizes and locations of each kernel are used to calculate lots of features. For each feature calculation, we need to find the sum of the pixels under white and black rectangles. To solve this, they introduced the integral image. However large your image, it reduces the calculations for a given pixel to an operation involving just four pixels. Nice, isn't it? It makes things super-fast.

But among all these features we calculated, most of them are irrelevant. For example, consider the image below. The top row shows two good features. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks. The second feature selected relies on the property that the eyes are darker than the bridge of the nose. But the same windows applied to cheeks or any other place is irrelevant. So how do we select the best features out of 160000+ features? It is achieved by Adaboost.



image

For this, we apply each and every feature on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. Obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are the features that most accurately classify the face and non-face images. (The process is not as simple as this. Each image is given an equal weight in the beginning. After each classification, weights of misclassified images are increased. Then the same process is done. New error rates are calculated. Also new weights. The process is continued until the required accuracy or error rate is achieved or the required number of features are found).

The final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others forms a strong classifier. The paper says even 200 features provide detection with 95% accuracy. Their final setup had around 6000 features. (Imagine a reduction from 160000+ features to 6000 features. That is a big gain).

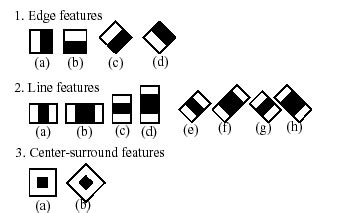
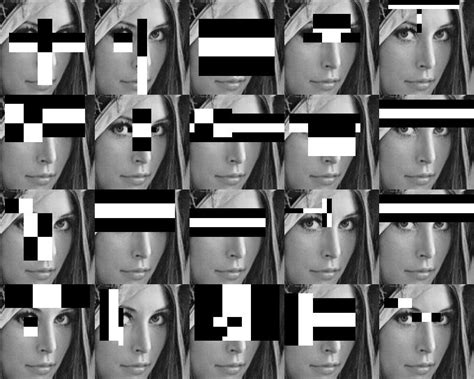
So now you take an image. Take each 24x24 window. Apply 6000 features to it. Check if it is face or not. Wow.. Isn't it a little inefficient and time consuming? Yes, it is. The authors have a good solution for that.

In an image, most of the image is non-face region. So it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot, and don't process it again. Instead, focus on regions where there can be a face. This way, we spend more time checking possible face regions.

For this they introduced the concept of Cascade of Classifiers. Instead of applying all 6000 features on a window, the features are grouped into different stages of classifiers and applied one-by-one. (Normally the first few stages will contain very many fewer features). If a window fails the first stage, discard it. We don't consider the remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region. How is that plan!

The authors' detector had 6000+ features with 38 stages with 1, 10, 25, 25 and 50 features in the first five stages. (The two features in the above image are actually obtained as the best two features from Adaboost). According to the authors, on average 10 features out of 6000+ are evaluated per sub-window.

So this is a simple intuitive explanation of how Viola-Jones face detection works. Read the paper for more details or check out the references in the Additional Resources section.

**Results**

Couple of pictures showing the results

Time estimation

**Conclusion**

*How to speed up the classification - GPU*

*How to get better classification results – more training picture, preprocessing data, resizing*

**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>
* Open face option –
  + <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>