Face Recognition

Gaurav Yogeshwar

ABSTRACT

Face recognition presents a challenging problem in the field of image analysis and computer vision. The security of information is becoming very significant and difficult. Security cameras are presently common in airports, Offices, University, ATM, Bank and in any locations with a security system. Face recognition is a biometric system used to identify or verify a person from a digital image. Face Recognition system is used in security. A face recognition system should be able to automatically detect a face in an image. This involves extracting its features and then recognizing it, regardless of lighting, expression, illumination, aging, transformations (translate, rotate and scale image) and pose, which is a difficult task.

Facial recognition system

A facial recognition system is a technology capable of matching a human face from a digital image or a video frame against a database of faces. Such a system is typically employed to authenticate users through ID verification services, and works by pinpointing and measuring facial features from a given image.

PROBLEM STATEMENT

Our job is to create a facial recognition system that can take the attendance of the child and upload it on the sheet.

Roadmap

Face recognition is really a series of several related problems:

1. First, look at a picture and find all the faces in it
2. Second, focus on each face and be able to understand that even if a face is turned in a weird direction or in bad lighting, it is still the same person.
3. Third, be able to pick out unique features of the face that you can use to tell it apart from other people— like how big the eyes are, how long the face is, etc.
4. Finally, compare the unique features of that face to all the people you already know to determine the person's name.

Face Recognition — Step by Step

Step 1: Finding all the Faces

Before recognizing the face it is necessary to detect it, so the first step in our pipeline is face detection. Obviously we need to locate the faces in a photograph before we can try to tell them apart!

As a human, your brain is wired to do all of this automatically and instantly. In fact, humans are too good at recognizing faces and end up seeing faces in everyday objects:

Computers are not capable of this kind of high-level generalization (at least not yet…), so we have to teach them how to do each step in this process separately.

There are several methods for face detection like Haar cascades, dlib frontal face detector, MTCNN, and a Caffe model using OpenCV's DNN module.

Out of these methods Haar Cascade classifier gave the worst results.

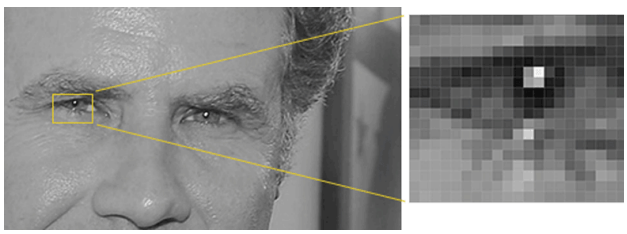
Dlib frontal face detector & MTCNN is best in detecting faces with high accuracy and OpenCV's Caffe model of the DNN is also good.

Dlib frontal face detector

HOG method invented in 2005 called Histogram of Oriented Gradients — or just HOG for short. Gradient is directional change in pixel intensity.

To find faces in an image, we'll start by making our image black and white because we don't need color data to find faces:

Our goal is to figure out how dark the current pixel is compared to the pixels directly surrounding it. Then we want to draw an arrow showing in which direction the image is getting darker:



Then we want to draw an arrow showing in which direction the image is getting darker:

The gradient for each pixel consists of magnitude and direction, calculated using the following formulae:

$$g = \sqrt{g_x^2 + g_y^2}$$

$$\theta = \arctan \frac{g_y}{g_x}$$

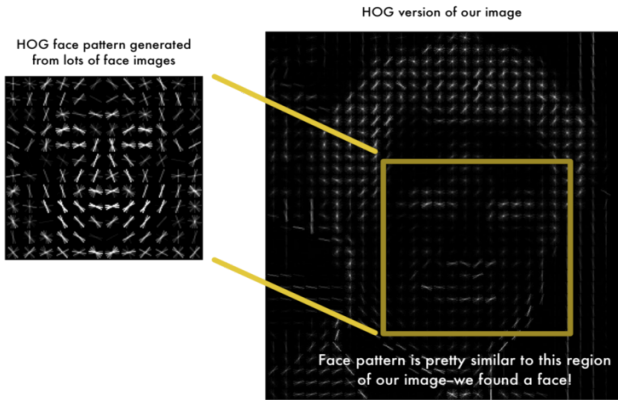
g_x and g_y are respectively the horizontal and vertical components of the change in the pixel intensity

If you repeat that process for every single pixel in the image, you end up with every pixel being replaced by an arrow. These arrows are called gradients and they show the flow from light to dark across the entire image:

But saving the gradient for every single pixel gives us way too much detail. We end up missing the forest for the trees. It would be better if we could just see the basic flow of lightness/darkness at a higher level so we could see the basic pattern of the image.

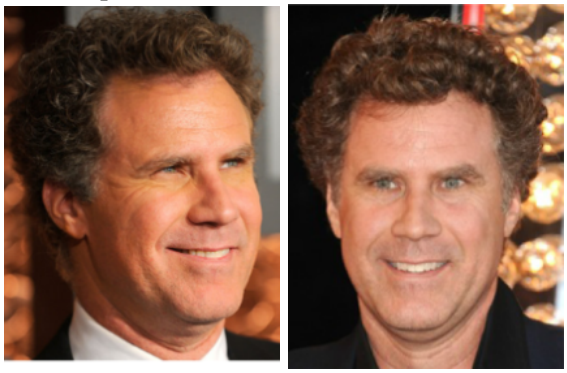
To do this, we'll break up the image into small squares of 16x16 pixels each. In each square, we'll count up how many gradients point in each major direction (how many point up, point up-right, point right, etc…). Then we'll replace that square in the image with the arrow directions that were the strongest.

To find faces in this HOG image, all we have to do is find the part of our image that looks the most similar to a known HOG pattern that was extracted from a bunch of other training faces:



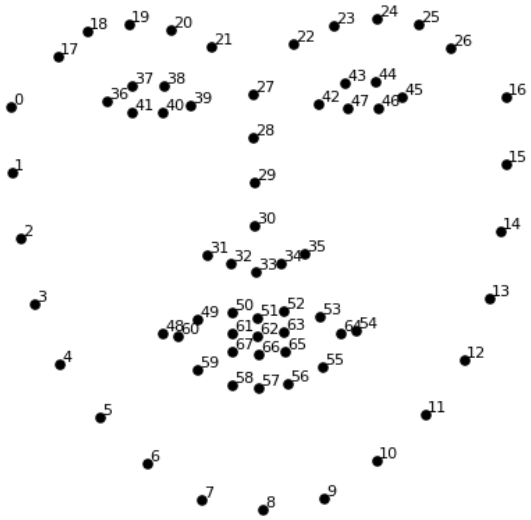
Step 2: Posing and Projecting Faces

Whew, we isolated the faces in our image. But now we have to deal with the problem that faces turned different directions look totally different to a computer:



To account for this, we will try to warp each picture so that the eyes and lips are always in the sample place in the image. We are going to use an algorithm called face landmark estimation.

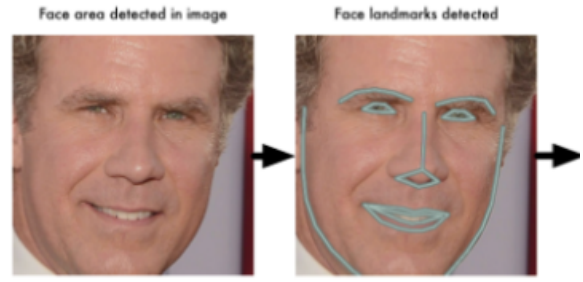
The basic idea is we will come up with 68 specific points (called **landmarks**) that exist on every face.

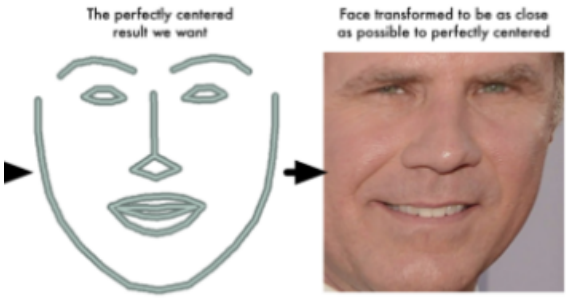


Face Feature	Points to Represent
Jaw	01-16
Right Eyebrow	17-21
Left Eyebrow	22-26
Nose	27-34
Right Eye	36-41
Left Eye	42-47
Mouth	48-67

learning algorithm to be able to find these 68 specific points on any face:

We are only going to use basic image transformations like rotation and scale that preserve parallel lines (called **affine transformations**)



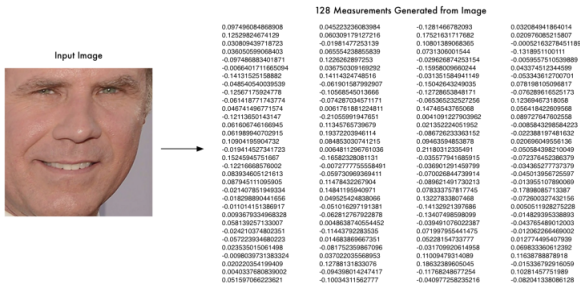


Now no matter how the face is turned, we are able to center the eyes and mouth in roughly the same position in the image.

Step 3: Encoding Faces

```
[-0.23, -0.54, ..., 0.27]
```

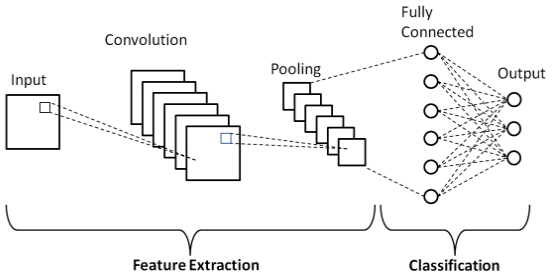
Before we can recognize faces in images and videos, we first need to quantify the faces in our training set. Keep in mind that we are not actually training a Deep Convolutional Neural Network here — the network has already been trained to create 128-d embeddings on a dataset of ~3 million images.



So which measurements should we collect from each face to build our known face database? Ear size? Nose length? Eye color? Something else?

It turns out that the measurements that seem obvious to us humans (like eye color) don't really make sense to a computer looking at individual pixels in an image. Researchers have discovered that the most accurate approach is to let the computer figure out the measurements to collect itself. Deep learning does a better job than humans at figuring out which parts of a face are important to measure.

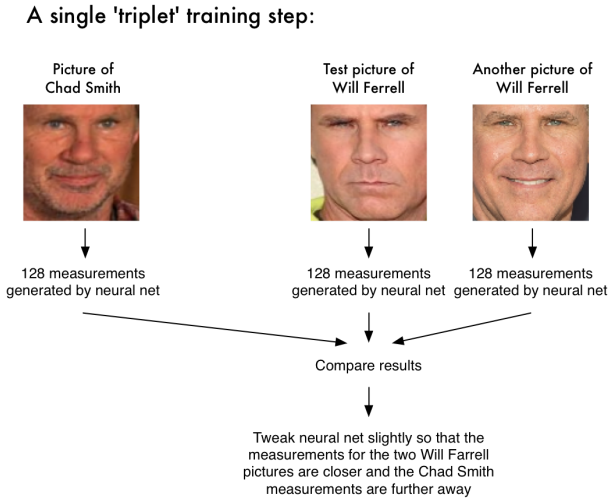
Deep Convolutional Neural Networks



- Input
- Convolution layers
- Pooling layers
- Densely connected layers
- Output

Step 4: Finding the person's name from the encoding

All we have to do is find the person in our database of known people who has the closest measurements to our test image.

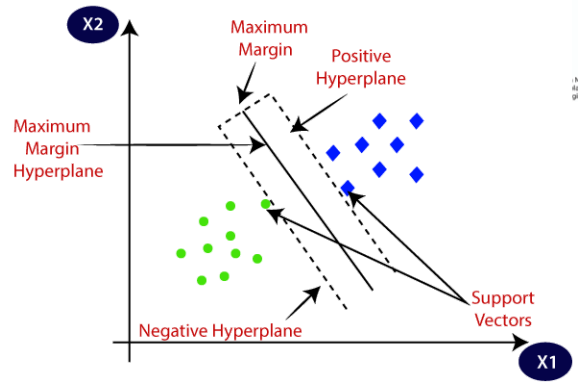


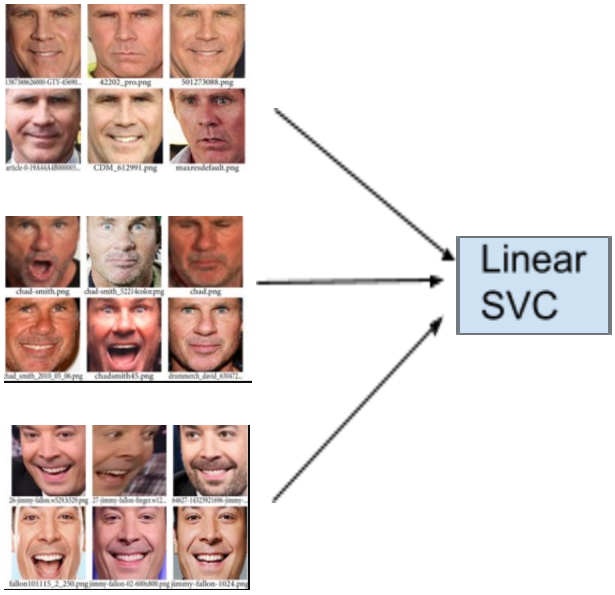
We can do that by using any basic machine learning classification algorithm. No fancy deep learning tricks are needed. We'll use a simple linear SVM classifier, but lots of classification algorithms could work.

All we need to do is train a classifier that can take in the measurements from a new test image and tell which known person is the closest match. Running this classifier takes milliseconds. The result of the classifier is the name of the person!

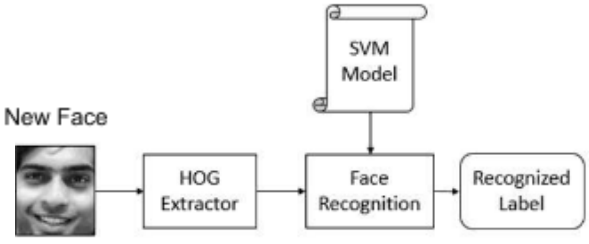
SVM

Support Vector Machine (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, the goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.





Block diagram of face training

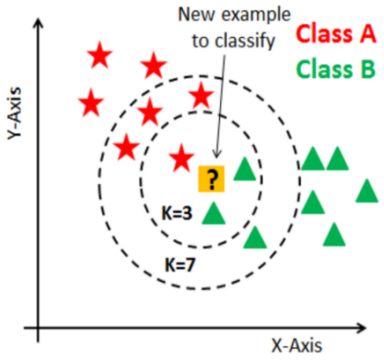


Block diagram of the face recognition process

k-NN

During classification, we can use a simple k-NN model + votes to make the final face classification.

K is the number of nearest neighbors to use. For classification, a majority vote is used to determine which class a new observation should fall into. Larger values of K are often more robust to outliers and produce more stable decision boundaries than very small values (K=3 would be better than K=1, which might produce undesirable results.



Pros

- Faster while using CPU, very light weight.
- Work under small occlusion.

Cons

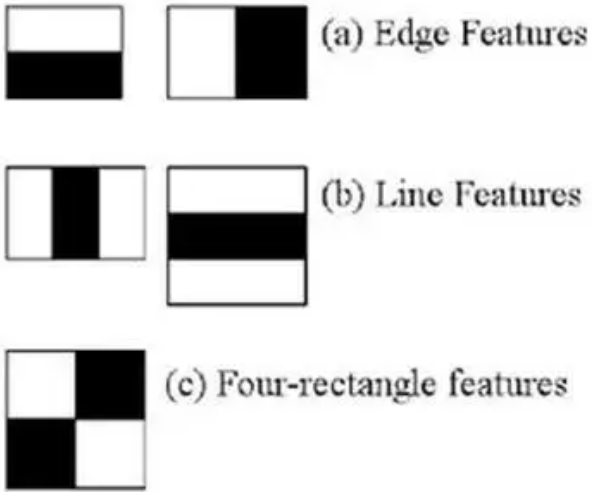
- Min size of face should be 80X80.
- Does not work for side faces or high extremes of non-frontal faces, like looking down or up.
- Doent work under heavy occlusion.

As I have said there are several methods for face detection.

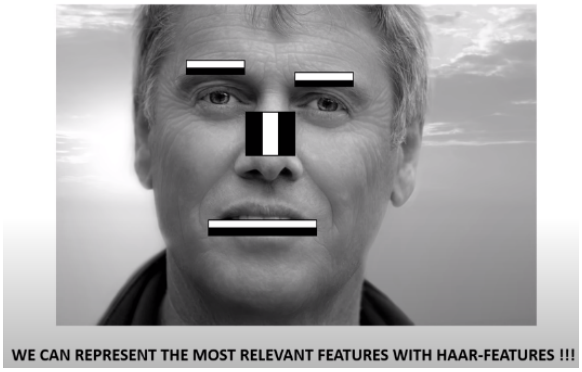
How does a Haar Cascade Classifier work?

The first step is to collect the Haar features. A Haar feature is essentially calculations that are performed on adjacent rectangular regions at a specific location in a detection window. The calculation involves summing the pixel intensities in each region and calculating the differences between the sums. Here are some examples of Haar features below.

These features can be difficult to determine for a large image. This is where integral images come into play because the number of operations is reduced using the integral image.

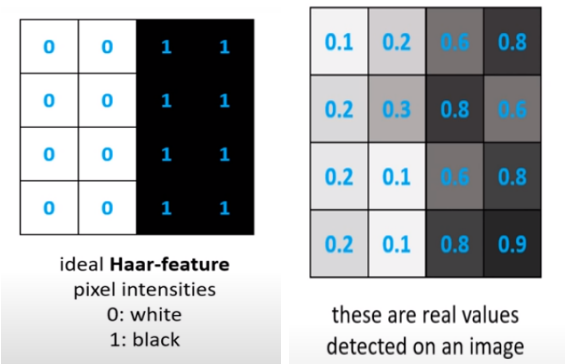


In the example below, images are used as convolutional kernels to extract features, where each feature is a value obtained by subtracting the summation of pixels under the black rectangle from the summation of pixels under the white rectangle.



Viola-Jones algorithm will compare how close the real scenario is to the ideal case.

1. Let's sum up the white pixel intensities
2. Calculate the sum of the black pixel intensities

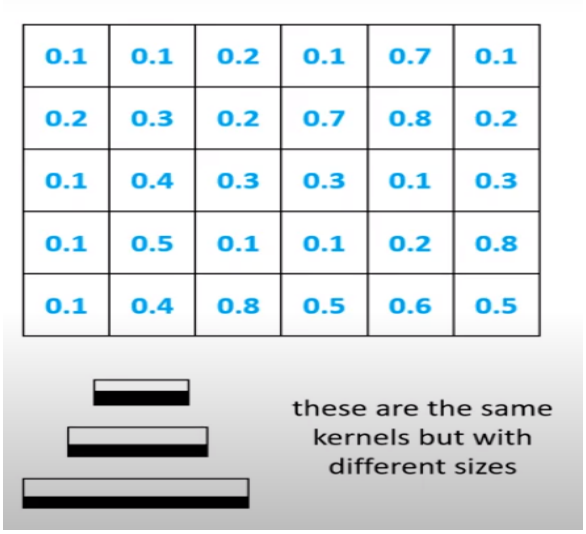


$$\Delta = \text{dark} - \text{white} = \frac{1}{n}\sum_{\text{dark}}^{n} I(x) - \frac{1}{n}\sum_{\text{white}}^{n} I(x)$$

Δ for ideal Haar-feature is 1.
Δ for the real image : 0.74- 0.18 = 0.56

The closer the value to 1, the more likely to have found a Haar - feature.

Creating Integral Images



The problem is that we have to calculate the average of a given region several times
~ the time complexity of these operations are O(N^2) .

we can use integral approach to achieve O(1)

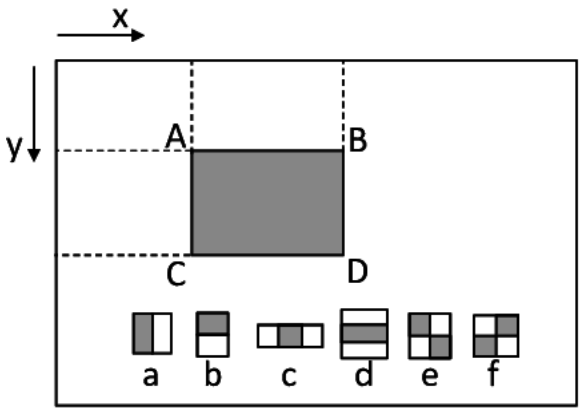
We have to use the Haar- feature with all possible size and locations ~ 200k features to calulate, every time we have to use a quadratic algorithm so there is a need for a better approch.

Integral images essentially speed up the calculation of these Haar features. Instead of computing at every pixel, it instead creates sub-rectangles and creates array references for each of those sub-rectangles. These are then used to compute the Haar features.

The idea is transforming an input images into a summed-area table, where the value at any point (x, y) in that table is the sum of all the pixels above and to the left of (x, y), inclusive:

$$I(x,y) = \sum_{\substack{x' \le x \\ y' \le y}} i(x', y')$$

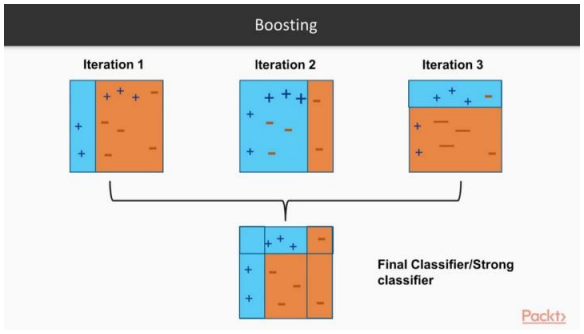
Where I(x,y) is the value of the integral image pixel in the position (x,y), while i(x,y) is the corresponding intensity in the original image. It is a recursive formula, hence, if we start from one corner of the input image, we will have the same result in the integral image. To make it clearer, let's see an example:



It's important to note that nearly all of the Haar features will be irrelevant when doing object detection, because the only features that are important are those of the object. However, how do we determine the best features that represent an object from the hundreds of thousands of Haar features? This is where Adaboost comes into play.

Adaboost essentially chooses the best features and trains the classifiers to use them. It uses a combination of "weak classifiers" to create a "strong classifier" that the algorithm can use to detect objects.

Weak learners are created by moving a window over the input image, and computing Haar features for each subsection of the image. This difference is compared to a learned threshold that separates non-objects from objects. Because these are "weak classifiers," a large number of Haar features is needed for accuracy to form a strong classifier.



The last step combines these weak learners into a strong learner using cascading classifiers.

$$F_T(x) = \sum_{t=1}^{T} f_t(x) \qquad f_t(x) = \alpha_t h(x)$$

$F_T(x)$ is Strong Learner

α_t weight calculated by considering the last iteration's error

$h(x)$ is weak Learner

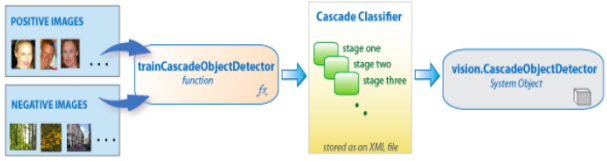
Implementing Cascading Classifiers

The cascade classifier is made up of a series of stages, where each stage is a collection of weak learners. Weak learners are trained using boosting, which allows for a highly accurate classifier from the mean prediction of all weak learners.

Based on this prediction, the classifier either decides to indicate an object was found (positive) or move on to the next region (negative). Stages are designed to reject negative samples as fast as possible, because a majority of the windows do not contain anything of interest.

It's important to maximize a low false negative rate, because classifying an object as a non-object will severely impair your object detection algorithm. A video below

shows Haar cascades in action. The red boxes denote “positives” from the weak learners.



Haar cascades are one of many algorithms that are currently being used for object detection. One thing to note about Haar cascades is that it is very important to reduce the false negative rate, so make sure to tune hyperparameters accordingly when training your model.

Pros

One of the primary benefits of Haar cascades is that they are just so fast — it’s hard to beat their speed,

Cons

The downside to Haar cascades is that they tend to be prone to false-positive detections, require parameter tuning when being applied for inference/detection, and just, in general, are not as accurate as the more “modern” algorithms we have today.

There is another face detection technique that performs the best of all which is

1. MTCNN (Multi-task Cascaded Convolutional Neural Network).
2. Dlib’s CNN model.
3. OpenCV’s Caffe model of the DNN.

Neural network based methods give high accuracy but they are slower than non-neural network models.
Should we use the CNN model for face recognition ?
The answer depends on the capability of your system? If you have a GPU then you can go with CNN methods which I have mentioned above.

Reference links

- Face detection with OpenCV and deep learning
- Face recognition with OpenCV, Python, and deep learning
- Face Detection Models: Which to Use and Why?
- How Does A Face Detection Program Work? (Using Neural Networks)
- Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning
- Face Detection with Haar Cascade