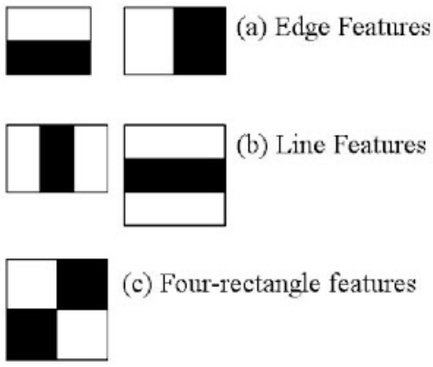
Neelkuamr Mistry
ID 214371306

Face detection using Haar Cascade:

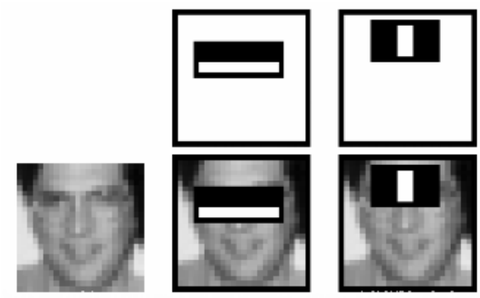
Haar cascade is a machine learning based approach which we can use to detect faces and other objects.

To start off, Haar algorithm needs a lot of images of faces and lots of images which are NOT faces to train the classifier. Then we need to extract features from it which we can use to detect faces. Luckily, pre-trained models are readily available on the internet for us to use.

Haar face detection is dependent on feature calculation. Examples of features are shown in the image below. Each feature is a value obtained by subtracting sum of pixels under the white rectangle from the sum of pixels under the black rectangles.



How do we use these features for our purpose? Well, look at the example in the picture below. In the first column after the image of face, we are using a feature (horizontal black and white stripes) to identify eyebrows and eyes in a facial image. This works on the fact that eyebrows and eyes will have more darker shade compare to the facial portion below it. In the second column of pictures, we are using a vertical feature to qualify eyebrows and eyes. This works on the fact that eyes and eyebrows have darker shades than nose line.



But, calculating these feature values for a large image is a lot of calculations. To solve this, it uses integral image which reduces the calculations for a given pixel to an operation involving just four pixels.

Also, we do not necessarily have to use all the available features because some of them are simply irrelevant. For example, applying the eyes and eyebrows qualifiers (discussed earlier) to cheek or any other areas will be irrelevant and useless. Feature selection optimization is done by Adaboost.

For this, each and every feature is applied to all training images and then we find the best threshold which will classify the faces to positive and negative. We then look at the errors and select the best features to classify the faces. This process is repeated until a low enough error rate is achieved. The final classifier includes the best suited features.

In addition, to make things even faster, we use cascade of classifiers. Instead of applying all the features from classifier, the features are grouped in different stages. In stage 1, we select the portion of image and apply the most

conclusive features of classifier. If the results of those features are positive, only then we progress to stage 2 and apply more features. If it passes all stages, then we conclude that portion of the image is a face. If it fails in any stage, then that portion of image does not have face and we head onto the next portion in image.

Face detection using Histogram of Oriented Gradients (HOG) method:

To detect faces in a given image, we first convert it to a grayscale image. Then we start looking at every single pixel of image, one at a time. For each pixel, we then look at its immediately surrounding pixels and compare their shading with the middle pixel. Then we draw an arrow in the direction the image is getting darker. If we follow the same process for all pixels in our image, one by one, we will end up with an image with lots of arrows. These arrows are gradients which point towards darkness from the light.

Having an arrow for every Single pixel of the image will add too much of details and we do not want that for our purpose. To overcome this, we divide image into small squares (e.g., 16 x 16) of pixels. In each square, we count how many pixels points towards which direction and replace the square with an arrow pointing towards the maximum count direction. At the end, we will have a HOG image which represent the simple facial structure. Below is the representation of a before and after image for this transformation.

To detect faces in this HOG image, we just must find parts in this image which looks like a known HOG face pattern. This known HOG pattern can be obtained from training faces.

Why do we use arrows(gradients) instead of directly comparing the color values in each pixel?
Different images of same person have different lighting on face. So, the pixel color values for different images would be far apart. By using HOG algorithm, we are basically comparing the similarity of those pixel color value differences in each image – which serves our purpose.

Face recognition:

One simple approach to recognize a face in image is to compare its face encoding with encodings of all the faces we know. (using face_recognition.compare_faces() method) However, this approach would not be efficient when we have a large dataset of know faces because comparing it with each face one by one takes a lot of time.

The solution to this is to only compare the most important landmarks of the faces. Deep Convolutional Neural Network (DNN) could be trained to take 128 most important measurements of the faces. Also, deep learning automatically figures out which measurements are important.

The training process requires 3 face images at a time.

1. Load a known face image.
2. Load another image of the same known person.
3. Load image of a totally different person.

The algorithm analyzes all 3 images and generates the measurements for the features so that image 1 and 2 have slightly similar measurements but image 2 and 3 has measurements slightly apart. This training is done on millions of faces. The more we train this, the better it gets. Thankfully, pre-trained models are already available over the internet for us to use. So, we can run our face images through and get 128 measurements for them.

Finally, to find the name of unknown face, we just have to compare our 128 face measurements with known faces measurement and find out which one is the closest match.

References:

https://docs.opencv.org/3.4/db/d28/tutorial$_c$*ascade*$_c$*lassifier.html*
https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78