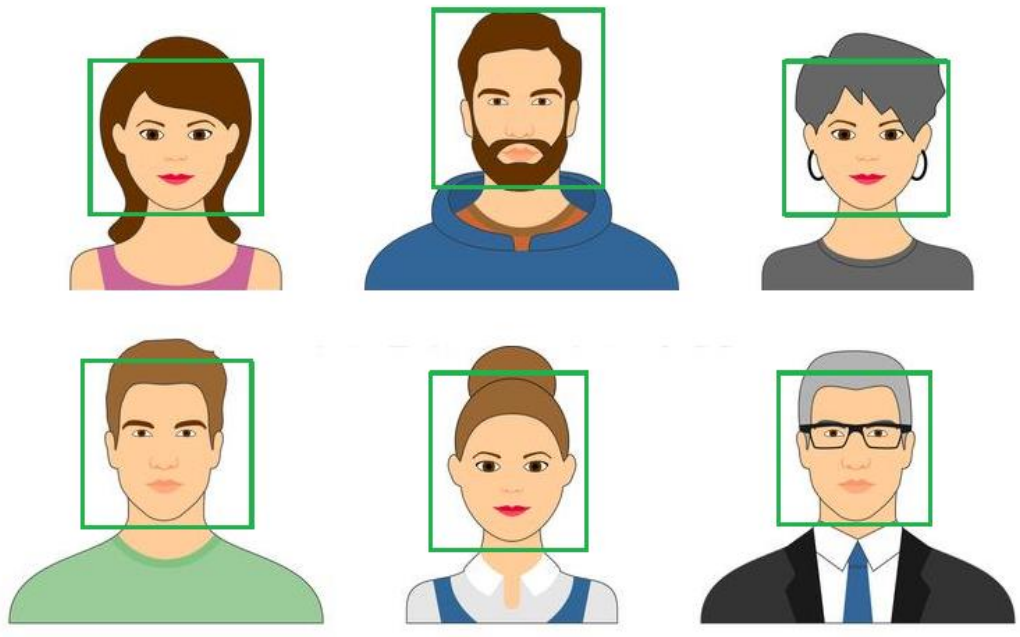
Face Detection Using Haar Cascade



Source: Dreamstime Images

Face detection is a common need nowadays, with many applications and devices using it. Many algorithms have been built to achieve this purpose. One such algorithm is the Haar Cascade algorithm. The algorithm was discussed back in 2001 by *Viola* and *Jones* and was published under the paper named "*Rapid Object Detection using a Boosted Cascade of Simple Features*".Though many years have passed, this algorithm still has broader applications in computer vision and image processing. This technique can be implemented in images as well as in real-time video to detect the faces and eyes.

In this article, we will try to present the paper in a simplified manner, and later, we will implement this technique via a pre-trained model using OpenCV and Python.

Paper can be divided into three parts basically:

> using a new image representation called *integral image*

> constructing a classifier by selecting a small number of important features using *Adaboost*

> combining more complex classifiers successively in cascade structure to improve the speed

We will discuss the above three one by one.

Part-1: Integral image.

The algorithm was trained on 4916 hands labeled images and 9544 non-face images. To train the images, they need to extract features from the images. They could have worked on pixels directly instead of using features. This is due to the fact that feature-based systems are substantially faster than pixel-based systems. And it becomes very useful when there are number of pixels to be scanned. Before diving into integral image part, first let's understand features to identify the faces.

The three main features that were used:

- Edge Features
- Line Features
- Four rectangle features

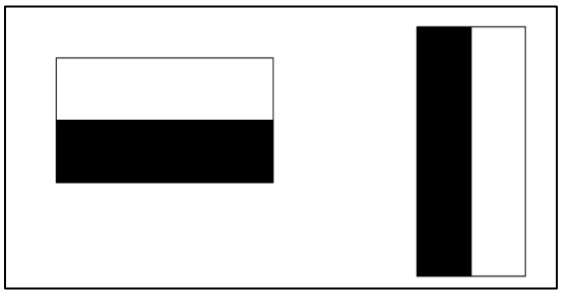


Fig-1) Edge features

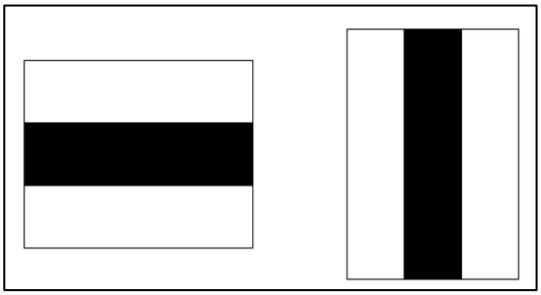


Fig-2) Line features

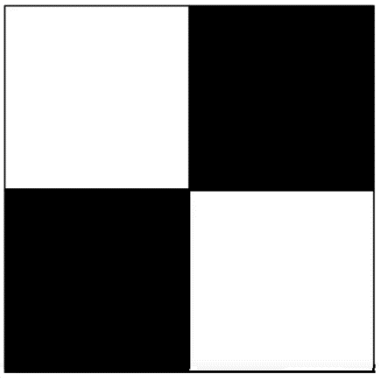


Fig-3) Four rectangle features

Let's discuss it for the edge features. Edge features are responsible for finding the horizontal and vertical lines in the images. Here, the difference between the sum of pixels in within two rectangular regions determines the value of a two-rectangle feature (shown in figure-1). It means to obtain a feature value, first we find the sum of all pixels inside the white rectangle and then find the sum of all pixels inside the black

rectangle and then subtract the mean of the sum of pixles from the white rectangle to the mean of the sum of pixels of the black rectangle.

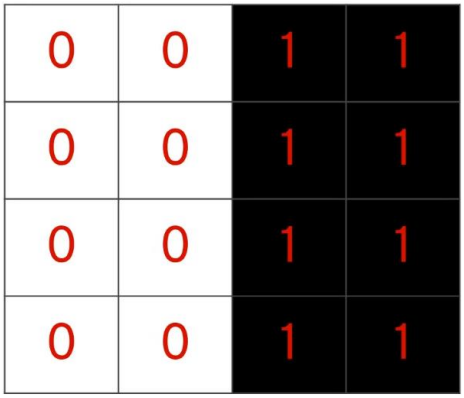


Fig-4a) Haar kernel with all light pixel on left and dark pixels on right

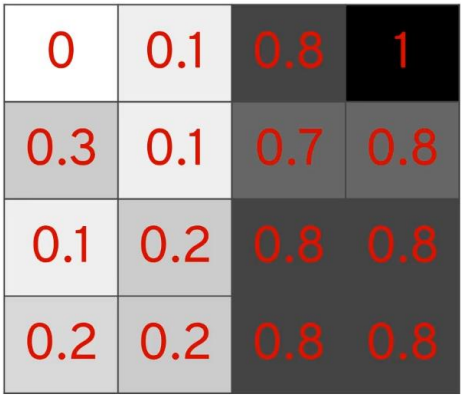


Fig-4b) Sample depiction of image with values between 0 to 1.

Let's compute the sum for figure 4-b) for both lighter region pixels and darker region pixels.

Lighter region → sum([0,0.1,0.3,0.1,0.1,0.2,0.2,0.2]) = 6.5

Darker region → sum([0.8,1,0.7,0.8,0.8,0.8,0.8,0.8]) = 1.2

Mean of lighter region → 1.2/8 = 0.15

Mean of darker region → 6.5/8 = 0.8125

Differences of mean of lighter and darker region = 0.8125 – 0.15 = 0.6625

So, we got 0.6625 as the difference. The closer the result is to 1, better a feature. We can define a threshold to classify whether this feature passed or whether to discard. It could be 0.6 or 0.7 or any value. Suppose the threshold was set for 0.6; hence it passed, and we can say that edge detected.

In the case of *three rectangle features,* the sum of pixels of two outside rectangles is computed and then subtracted from the sum of pixels of the rectangle present inside. Three set of rectangular features, as its structure suggests, describe whether there is a lighter region surrounded by a darker region.

And in the case of *four rectangle features,* as the structure suggests, we find the sum of pixels of the diagonally darker region and then subtract it from the sum of pixels of the lighter region. This set of rectangular features is responsible for detecting pixel intensity changes across diagonals.

Like the convolutional filter of CNN, haar features traverse from top left to bottom right, step by step.

We have shown above that how the feature calculation is done. But doing this calculation for the entire image would lead to a large amount of time and hence is computationally expe. With 24 x 24 resolution of the detector, the overall image contains a set of features of more than 180,000,there needs to have a better approach.

So, the authors come up with the concept of ***Integral Image*.** It is also known by the summed area table. Each pixel in an Integral Image is the sum of all pixels to its left and above in the Original Image.

Mathematcially:

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

where $i(x,y)$ is the value of the pixel at (x,y).

Creating the Integral image of the image is very simple and independent of size. We can do it in raster scan fashion in a single scan of image. The sample of integral image is shown below:

98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Fig -5 a) Image

98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

Fig-5 b) Integral Image.

In the above, notice that 3490 (*fig -5b*) represents the sum of all yellow portioned numbers.

Now consider a situation,if we want to find the brightness of pixel 3490, denoted in yellow below, this is how we will calculate.

98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Fig -6a) Highlighted box sum needs to be computed

98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Fig-6b) Process of computing sum

We will calculate this by subtracting the pixels with value 1249 and 1137 (*denoted by red box in figure 6-a*) and adding 417 (*denoted by green box in figure 6-b*).

98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

For better understanding, let's go deeper into this:

In the image pixel 1137, which was represented using red color in figure 6-b, is represented by a rectangle sky blue, and pixel 1249 is represented by pink color rectangle. Notice that common rectangle containing numbers, (98,110,99,110) has been subtracted twice. So it needs to be added , which leads us to formula:

Fig-7: Process of computing sum shown

In internal image.

Sum = A - B - C + D

Where A = Yellow box in figure 6b

B,C = Red box in figure 6b

D = Green box in figure 6b

Now let's see how does Haars value get calculated in integrated image. Let's visualize it through the image.

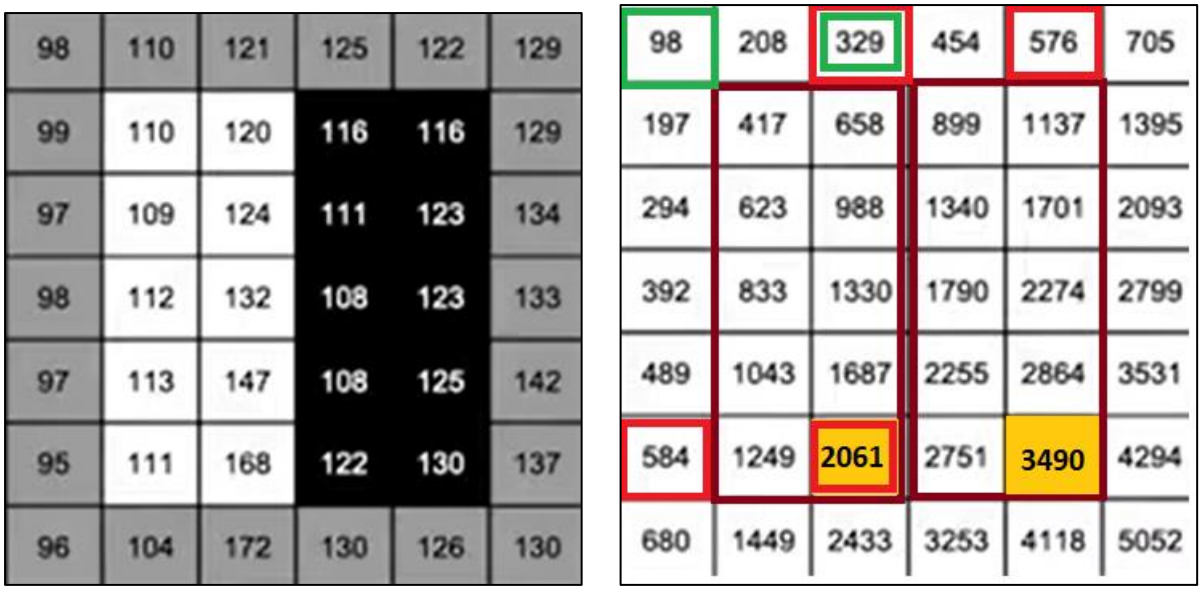


Fig 8- Haar cascade working of integral image

As we know that formula for calculating haar cascade is given by:

= Sum of pixels in white region – sum of pixels in dark region

= (2061-329-584+98) – (3490-576-2061+329)

= 64

In the above picture, logic which we have discussed earlier is followed. To compute white pixels where total sum is 2061, pixles 584 and 329 has to be subtracted, and 98 which has been subtracted twice, needs to be added. Same logic needs to be followed for black region.

Here above all the red color boxes denotes , pixels that needs to be subtracted and green color box, denotes pixels need to be added. In this way complexity reduces by a great amount.

Adaboost

There were over 180,000 rectangles features associated with the each image sub-window, and this number was significantly larger than number of pixels. Despite the fact that each characteristic can be computed quickly, computing the entire set was prohibitively expensive. So, a feature slection technique was needed. And basically this was a classification problem, as we had the features and the training set, any machine learning algorithm could be applied. The authors decided to go with Adaboost, to select a small subset of features and train the classifier.

After experimenting, authors found out that very small number of these features can be combined to form an effective classifier. So, to achieve this goal, weak learning algorithm was designed to single rectangle feature, which best separates the positive and negative examples. The weak learner chooses the best threshold function for each feature so that the least amount of examples are misclassified.

In each round of boosting, Adaboost algorithm selects one feature from the possible features of 180,000.

Initial rectangle features selected by Adaboost were meaningful and could be easily interpreted. The whole idea of feature selection was based on two properties: Region of eyes are often darker than the region of nose and cheeks. So this feature compares the difference in intensity between the region of eyes and region of cheeks. And the second feature was based on property that eyes are darker than bridge of the nose.

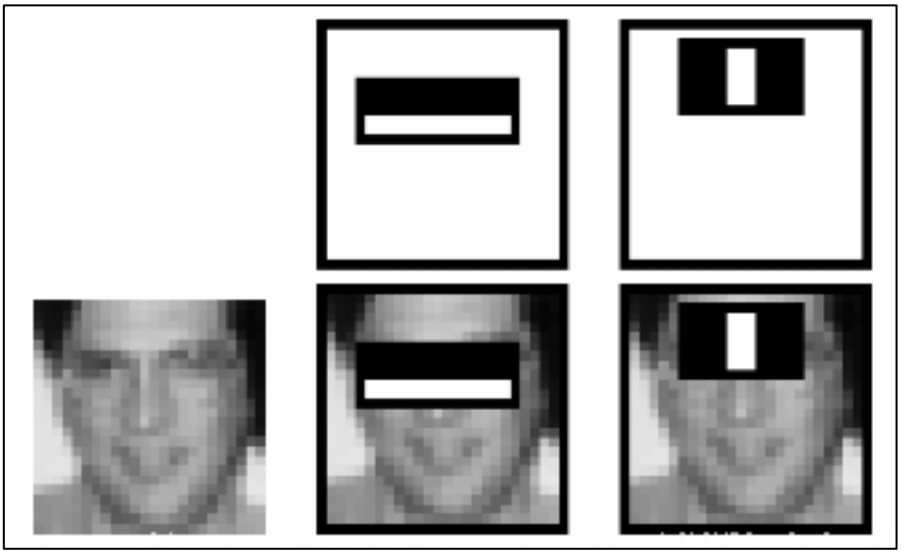


Fig -9: Performing the top two features selected by AdaBoost on an image

The Attentional Cascade

A still a monumental task was to reduce the time of scanning the image, as most of the region in the image would consist of non- image region So, the goal was to create a cascade of classifiers to improve detection performance while decreasing computational time. Idea here was to run the simpler classifier first followed by the complex classifier.

This would work as if simple classifier detects the result as positive, then only complex classifier would run. Hence it would run in sequence first followed by second followed by third and so on. The stage at which negative result is achieved, further processing of subwindows stop. Move on to the next window. The idea is much like of degenerate decision tree, what generally termed as "*cascade*". This achieves in saving a lot of time.

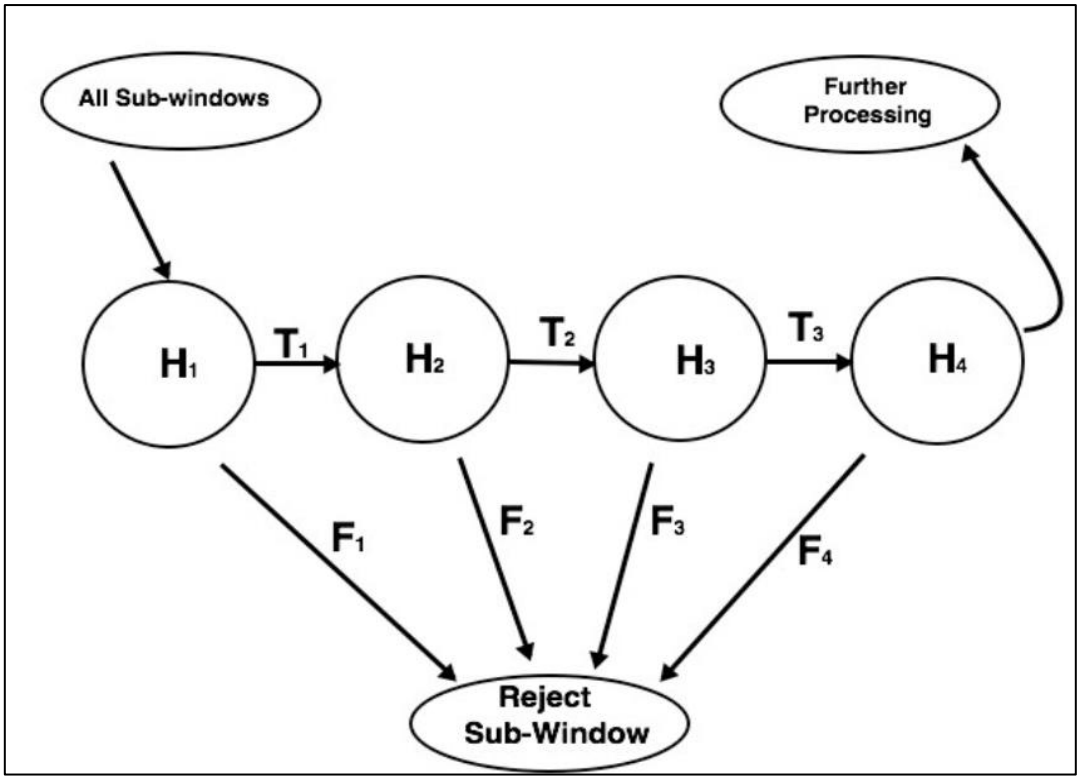


Figure 10 – Processing of degenerate tree. The Attentional Cascade follows the same mechanism.

As described in their paper, the entire face detection process includes 38 steps and approximately 6000 characteristics. Despite this, cascade structure results in fast average detection time. Faces are recognised using an average of 10 feature evaluations per sub-window on a tough dataset with 507 faces and 75 million sub-windows.

The authors' detector has almost 6000 features and 38 stages, with the first five stages having 1, 10, 25, 25, and 50 features. The remaining layers are becoming progressively complex in terms of features. Since in the initial stages, when number of features were small, most of the windows which does not contain any facial features were discarded, so lowering the chance of false negative ratio. And in the later stages, where the features were complex, higher number of features focused on achieving higher detection rate and lower positive rates.

Each stage of the cascade lowers the rate of false positives and increases the rate of detection. A goal is set for the lowest number of false positives and the highest reduction in detection. Each stage is improved by adding more characteristics until the target detection and false positive rates are achieved. Stages are added until the total false positive and detection rate target is satisfied.

This was the explanation for working of face detection using the Haar cascade.

Below given is the example shown in which this classifier was tried. And it detected every face accurately.

Source: Wallpaper flare

References:

Viola, P. and Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. *ACCEPTED CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION*. [online] Available at: https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf.

Integral Image | Face Detection. [online] Available at: https://www.youtube.com/watch?v=5ceT8O3k6os&t=198s [Accessed 24 May 2022].

Python-for-computer-vision-with-opencv-and-deep-learning Udemy Course. [online] [https://www.udemy.com/course/python-for-computer-vision-with-opencv-and-deep-learning] [Accessed 24 May 2022]

Azarcoya-Cabiedes, Willy & Vera-Alfaro, Pablo & Torres-Ruiz, Alfonso & Salas-Rodríguez, Joaquín. (2014). Automatic detection of bumblebees using video analysis. Dyna (Medellin, Colombia). 81. 81-84. 10.15446/dyna.v81n186.40475.