Viola-Jones detected 87% of the total faces whereas Kanade-Lucas-Tomasi detected only 84%. An interesting thing to note is that out of all the images that were run through both the algorithms, Viola-Jones detected faces in a few images that weren’t detected by the Kanade-Lucas-Tomasi algorithm but there were no images whose faces were detected by Kanade-Lucas-Tomasi but not by Viola-Jones.

**Viola – Jones**

There are four stages: Haar selection of characteristics, creation of images that are integral, training of adaboost, classifiers that cascade.

Viola-Jones has several advantages like feature selection that is sophisticated and an invariant detector that locates scales. We can scale the features instead of scaling the image itself. Since it’s a general scheme of detection, it can be trained for detecting other things like cars.[6]

But Viola-Jones comes with disadvantages too. It is not as effective detecting tilted or turned faces. It is sensitive to lighting conditions and there could possibly be different detections of the exact face due to sub windows overlapping.[6]

**Implementation**

The test face images are tested through both of the algorithms. In some instances, Viola-Jones detected the face, but Kanade-Lucas-Tomasi didn’t. In some cases, neither detected the faces. In majority of the cases, both detected the faces. But there was no instance where Kanade-Lucas-Tomasi detected the faces but Viola-Jones didn’t.

**Table 1**

|  | **Viola-Jones** | **Kanade-Lucas-Tomasi** |
| --- | --- | --- |
| **Looking front** | 97% | 90% |
| **Looking left** | 90% | 85% |
| **Looking right** | 88% | 83% |
| **Looking up** | 80% | 80% |
| **Looking down** | 80% | 80% |
| **Total** | 87% | 84% |

**Conclusions**

Viola-Jones has a great detection rate in every scenario and is better than the Kanade-Lucas-Tomasi in every scenario.

**VIOLA-JONES ALGORITHM**

Developed in 2001 by Paul Viola and Michael Jones, the Viola-Jones algorithm is an object-recognition framework that allows the detection of image features in real-time.

Viola-Jones is quite powerful and its application has proven to be exceptionally notable in real-time face detection. Although this algorithm is painfully slow to train, it can detect faces with impressive speed.

Given an image(this algorithm works on grayscale image), the algorithm looks at many smaller subregions and tries to find a face by looking for specific features in each subregion. It needs to check many different positions and scales because an image can contain many faces of various sizes. Viola and Jones used Haar-like features to detect faces in this algorithm.

**The characteristics of Viola–Jones algorithm which make it a good detection algorithm are:**

* Robust – very high detection rate (true-positive rate) & very low false-positive rate always.
* Real time – For practical applications at least 2 frames per second must be processed.
* Face detection only (not recognition) - The goal is to distinguish faces from non-faces (detection is the first step in the recognition process).

**The Viola Jones algorithm has four main steps:**

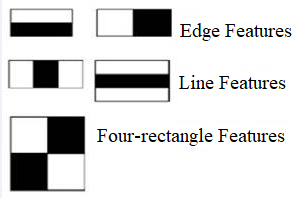
* Selecting Haar-like features
* Creating an integral image
* Running AdaBoost training
* Creating classifier cascades

**What are Haar-Like features?**

In the 19th century a Hungarian mathematician, Alfred Haar gave the concepts of Haar wavelets, which are a sequence of rescaled “square-shaped” functions which together form a wavelet family or basis. Voila and Jones adapted the idea of using Haar wavelets and developed the so-called Haar-like features.

Haar-like features are digital image features used in object recognition. All human faces share some universal properties of the human face like the eyes region is darker than its neighbour pixels, and the nose region is brighter than the eye region.

A simple way to find out which region is lighter or darker is to sum up the pixel values of both regions and compare them. The sum of pixel values in the darker region will be smaller than the sum of pixels in the lighter region. If one side is lighter than the other, it may be an edge of an eyebrow or sometimes the middle portion may be shinier than the surrounding boxes, which can be interpreted as a nose This can be accomplished using Haar-like features and with the help of them, we can interpret theifferent parts of a face.



There are 3 types of Haar-like features that Viola and Jones identified in their research:

* Edge features
* Line-features
* Four-sided features

Edge features and Line features are useful for detecting edges and lines respectively. The four-sided features are used for finding diagonal features.

The value of the feature is calculated as a single number: the sum of pixel values in the black area minus the sum of pixel values in the white area. The value is zero for a plain surface in which all the pixels have the same value, and thus, provide no useful information.

Since our faces are of complex shapes with darker and brighter spots, a Haar-like feature gives you a large number when the areas in the black and white rectangles are very different. Using this value, we get a piece of valid information out of the image.

To be useful, a Haar-like feature needs to give you a large number, meaning that the areas in the black and white rectangles are very different. There are known features that perform very well to detect human faces:



For example, when we apply this specific haar-like feature to the bridge of the nose, we get a good response. Similarly, we combine many of these features to understand if an image region contains a human face.

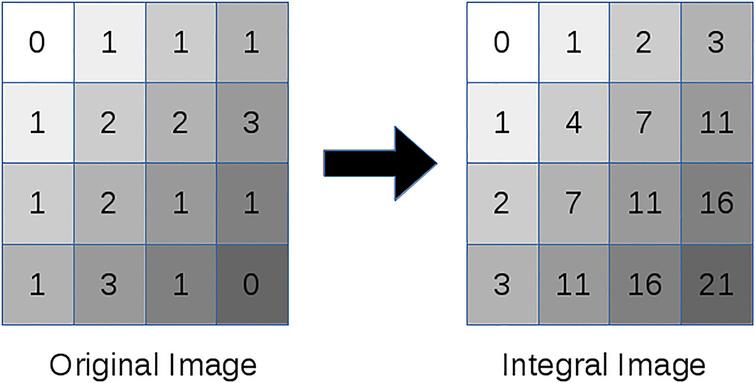
What are Integral Images?

In the previous section, we have seen that to calculate a value for each feature, we need to perform computations on all the pixels inside that particular feature. In reality, these calculations can be very intensive since the number of pixels would be much greater when we are dealing with a large feature.

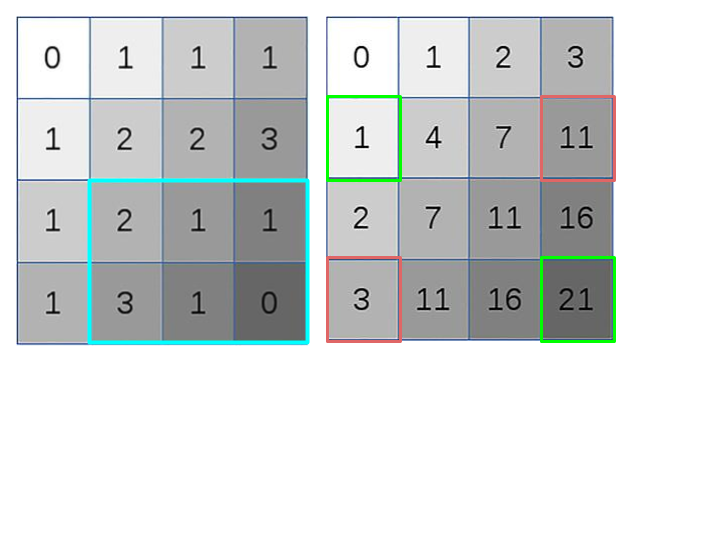
The integral image plays its part in allowing us to perform these intensive calculations quickly so we can understand whether a feature of several features fit the criteria.

An integral image (also known as a summed-area table) is the name of both a data structure and an algorithm used to obtain this data structure. It is used as a quick and efficient way to calculate the sum of pixel values in an image or rectangular part of an image.

In an integral image, the value of each point is the sum of all pixels above and to the left, including the target pixel:



Using these integral images, we save a lot of time calculating the summation of all the pixels in a rectangle as we only have to perform calculations on four edges of the rectangle. See the example below to understand.



When we add the pixels in the blue box, we get 8 as the sum of all pixels and here we had six elements involved in your calculation. Now to calculate the sum of these same pixels using the integral image, you just need to find the corners of the rectangle and then add the vertices which are green and subtract the vertices in the red boxes. Now doing that here

21+1 - 11 -3 =8

We get the same answer and only four numbers are involved in calculations. No matter how many pixels are in the rectangle box, we will just need to compute on these 4 vertices.

Now to calculate the value of any haar-like feature, you have a simple way to calculate the difference between the sums of pixel values of two rectangles.

How is AdaBoost used in viola jones algorithm?

Next, we use a Machine Learning algorithm known as AdaBoost. But why do we even want an algorithm?

The number of features that are present in the 24×24 detector window is nearly 160,000, but only a few of these features are important to identify a face. So we use the AdaBoost algorithm to identify the best features in the 160,000 features.

In the Viola-Jones algorithm, each Haar-like feature represents a weak learner. To decide the type and size of a feature that goes into the final classifier, AdaBoost checks the performance of all classifiers that you supply to it.

To calculate the performance of a classifier, you evaluate it on all subregions of all the images used for training. Some subregions will produce a strong response in the classifier. Those will be classified as positives, meaning the classifier thinks it contains a human face. Subregions that don’t provide a strong response don’t contain a human face, in the classifiers opinion. They will be classified as negatives.

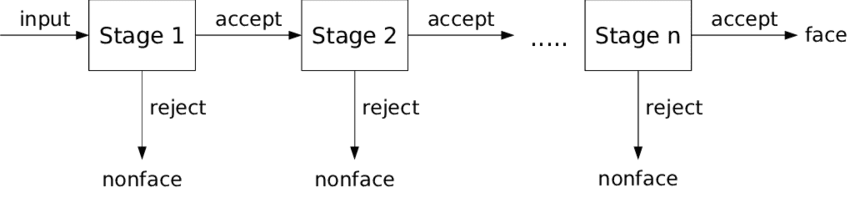
The classifiers that performed well are given higher importance or weight. The final result is a strong classifier, also called a boosted classifier, that contains the best performing weak classifiers.

So when we’re training the AdaBoost to identify important features, we’re feeding it information in the form of training data and subsequently training it to learn from the information to predict. So ultimately, the algorithm is setting a minimum threshold to determine whether something can be classified as a useful feature or not.

What are Cascading Classifiers?

Maybe the AdaBoost will finally select the best features around say 2500, but it is still a time-consuming process to calculate these features for each region. We have a 24×24 window which we slide over the input image, and we need to find if any of those regions contain the face. The job of the cascade is to quickly discard non-faces, and avoid wasting precious time and computations. Thus, achieving the speed necessary for real-time face detection.

We set up a cascaded system in which we divide the process of identifying a face into multiple stages. In the first stage, we have a classifier which is made up of our best features, in other words, in the first stage, the subregion passes through the best features such as the feature which identifies the nose bridge or the one that identifies the eyes. In the next stages, we have all the remaining features.



When an image subregion enters the cascade, it is evaluated by the first stage. If that stage evaluates the subregion as positive, meaning that it thinks it’s a face, the output of the stage is maybe.

When a subregion gets a maybe, it is sent to the next stage of the cascade and the process continues as such till we reach the last stage.

If all classifiers approve the image, it is finally classified as a human face and is presented to the user as a detection.

Now how does it help us to increase our speed? Basically, If the first stage gives a negative evaluation, then the image is immediately discarded as not containing a human face. If it passes the first stage but fails the second stage, it is discarded as well. Basically, the image can get discarded at any stage of the classifier