The smallest element of an image is called a **pixel**, or a picture element. It is basically a dot in the picture. An image contains multiple pixels arranged in rows and columns.

You will often see the number of rows and columns expressed as the image **resolution**. For example, an Ultra HD TV has the resolution of 3840x2160, meaning it is 3840 pixels wide and 2160 pixels high.

But a computer does not understand pixels as dots of color. It only understands numbers. To convert colors to numbers, the computer uses various color models.

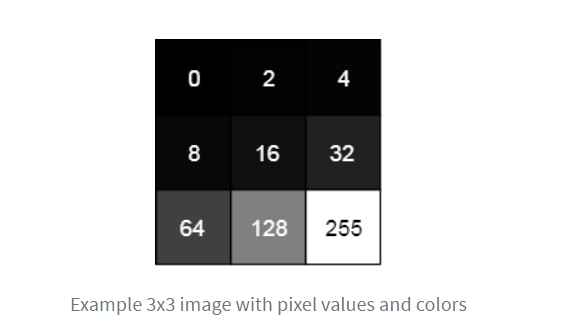
In color images, pixels are often represented in the RGB color model. RGB stands for **R**ed **G**reen **B**lue. Each pixel is a mix of those three colors. RGB is great at modeling all the colors humans perceive by combining various amounts of red, green, and blue.

Since a computer only understand numbers, every pixel is represented by three numbers

corresponding to the amounts of red, green, and blue present in that pixel

1. Grayscale(black and white) images

* Each pixel is represented by three numbers, corresponding to the amounts of red, green and blue present in that pixel.
* In many applications, the range of intensities is from 0 (black) to 255 (white). Everything between 0 and 255 is various shades of gray
* If each grayscale pixel is a number, an image is nothing more than a matrix (or table) of numbers:



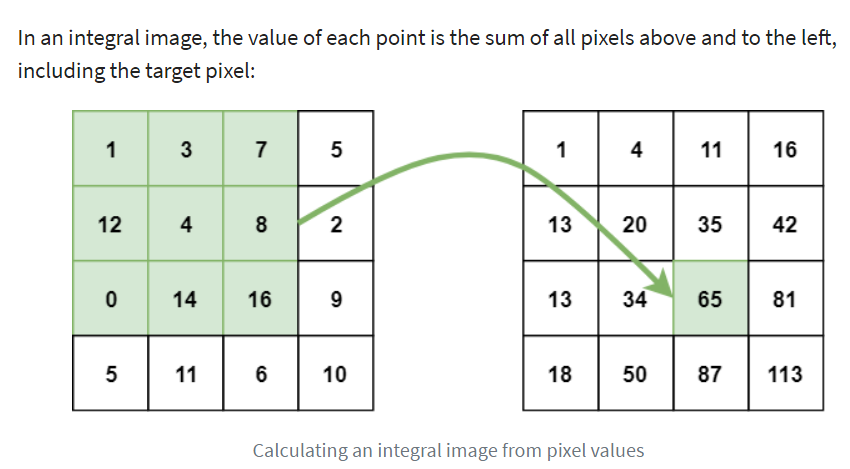
* BigNote: in color image, there are three such matrices representing the red, green and blue channels.

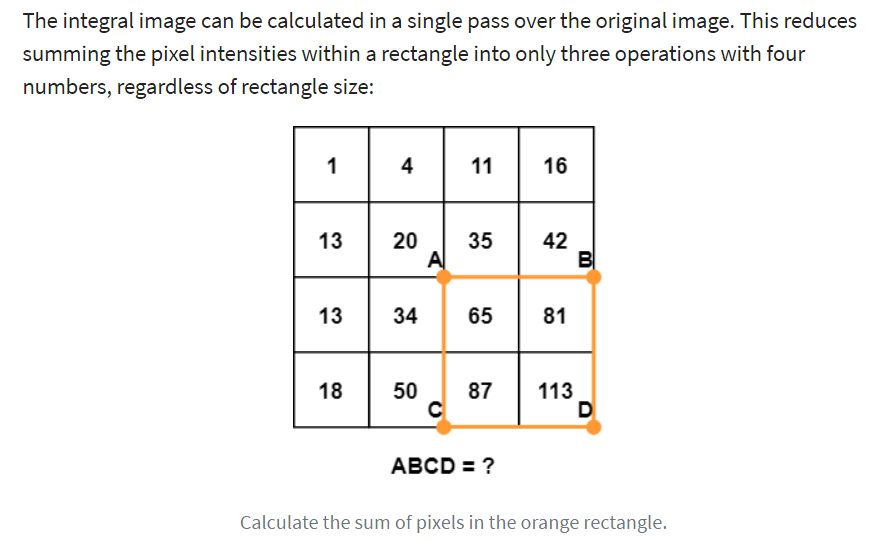
1. What are feature

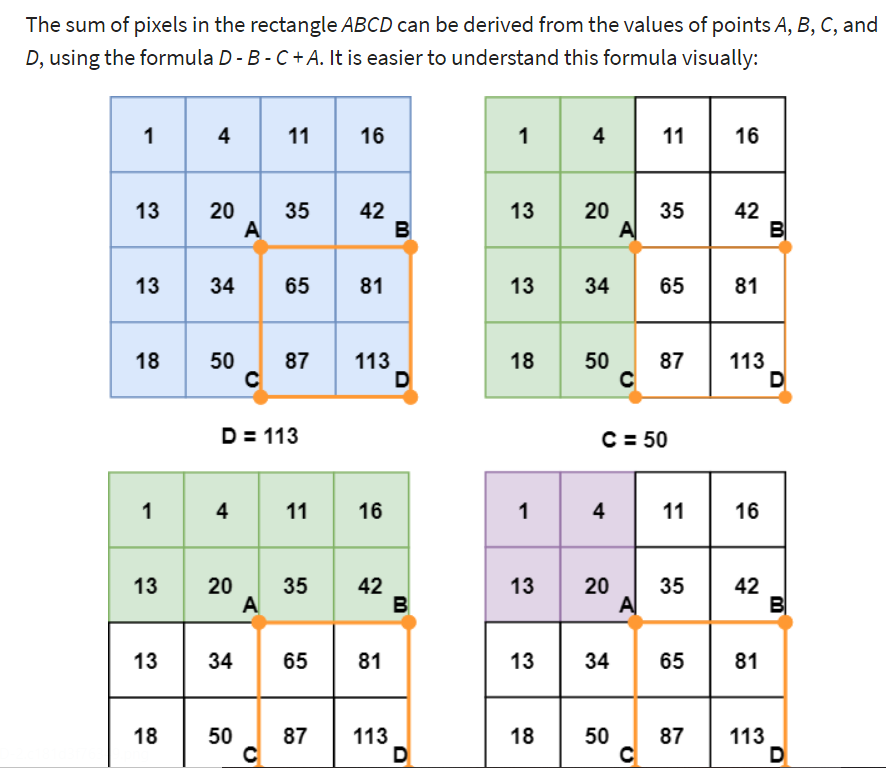
* A feature is a piece of information in an image that is relevant to solving a certain problem.
* Could be simple like a single pixel value, or more complex like edges, corners, and shapes. You can combine multiple simple features into a complex feature.

1. Harlike feature

* All human faces share some similarities. If you look at a photograph showing a person’s face, you will see, for example, that the eye region is darker than the bridge of the nose. The cheeks are also brighter than the eye region. We can use these properties to help us understand if an image contains a human face.
* 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. For uniform areas like a wall, this number would be close to zero and won’t give you any meaningful information.
* <https://realpython.com/traditional-face-detection-python/>
* Simple solution: plus all value pixels in black area and white area.
* But this takes a lot of time using the limited resources of a computer.
* To tackle this problem, Viola and Jones used integral images.



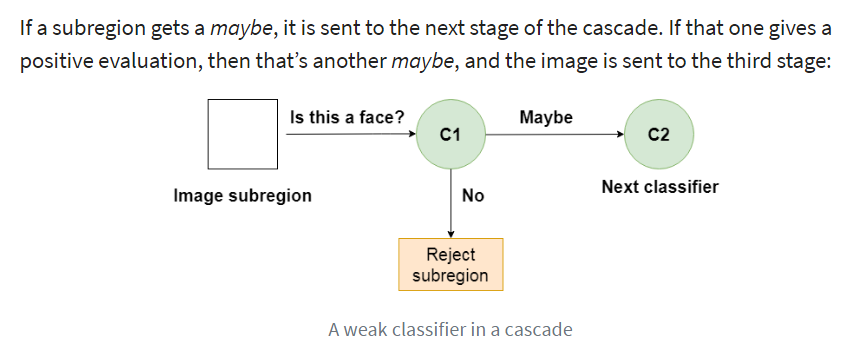


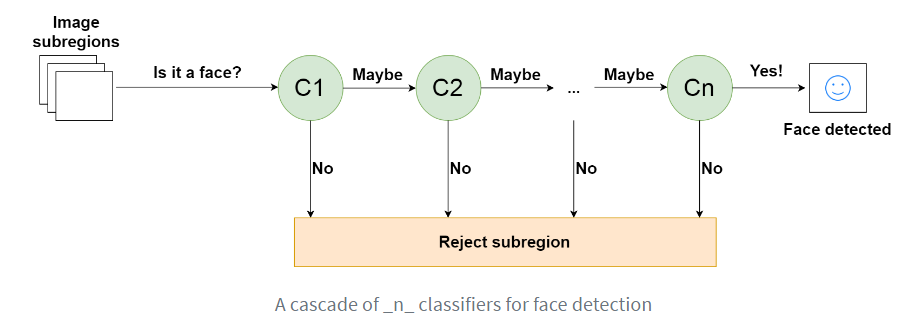


Now you have a simple way to calculate the difference between the sums of pixel values of two rectangles. This is perfect for Haar-like features!

1. AdaBoost

* 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.
* To solve it, Viola and Jones turned their strong classifier (consisting of thousands of weak classifiers) into a cascade where each weak classifier represents one stage. The job of the cascade is to quickly discard non-faces and avoid wasting precious time and computations.

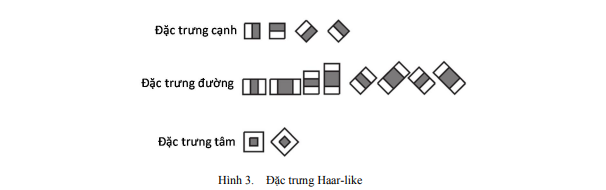




* This is designed so that non-faces get discarded very quickly, which saves a lot of time and computational resources.  Since every classifier represents a feature of a human face, a positive detection basically says, “Yes, this subregion contains all the features of a human face.” But as soon as one feature is missing, it rejects the whole subregion.
* To accomplish this effectively, it is important to put your best performing classifiers early in the cascade. In the Viola-Jones algorithm, the eyes and nose bridge classifiers are examples of best performing weak classifiers.

1. Using a Viola-Jone Classifier

* Training a Viola-Jones classifier from scratch can take a long time. Fortunately, a pre-trained Viola-Jones classifier comes out-of-the-box with OpenCV! You will use that one to see the algorithm in action.



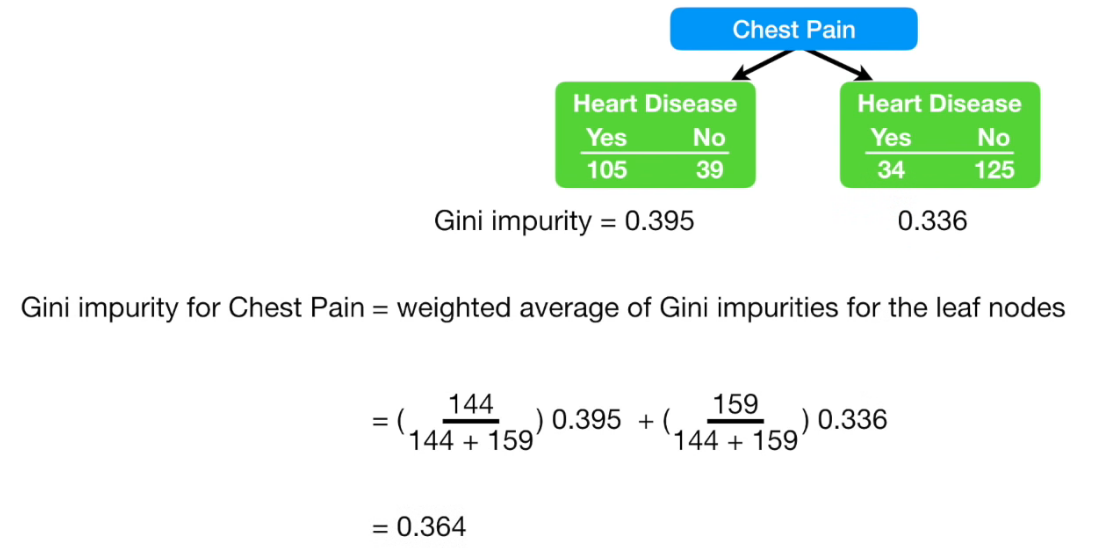
* They trained a classifier like this to find the best features and the best order to apply those feature.
* Ví dụ:
  + Đặc trưng cạnh: Divide a group of pixels into two.
  + Đặc trưng đường: three-retangle feature
* Delta = White\_region – Black\_region

Q2: How they are useful?

* FirstNote:
  + Dark pixel: lower values
  + Bright pixel: higher values

1. Decision Tree

* Gini impurity = 1 – (the probability of “yes”)^2 – (the probability of “no”)^2



1. Adaboost

* Adaboost has three main concepts.

1. Adaboost first concept

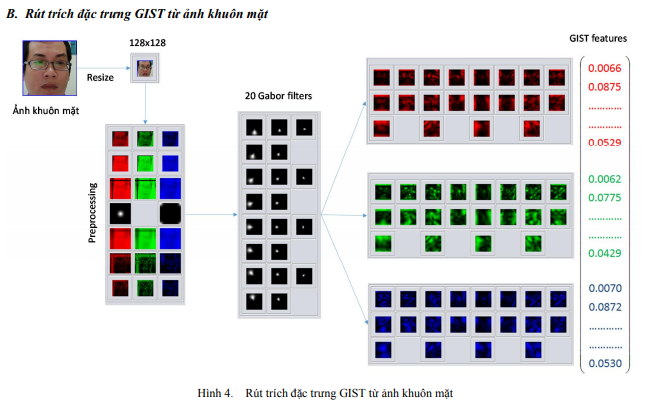
* Adaboost has a tree called stump that has 1 node, and two leaves
* Stumps are technically “Weak learners”.
* Adaboost combines a lot of “Weak learners” to make classifications. The weak leaners are almost aways Stumps.
* Mỗi mẫu có một trọng số

1. Some Stumps get more say in the classification than others.
2. Each stump is made by taking the previous stump’s mistakes into account.
3. Lọc trung bình (Normalize Box Filter)

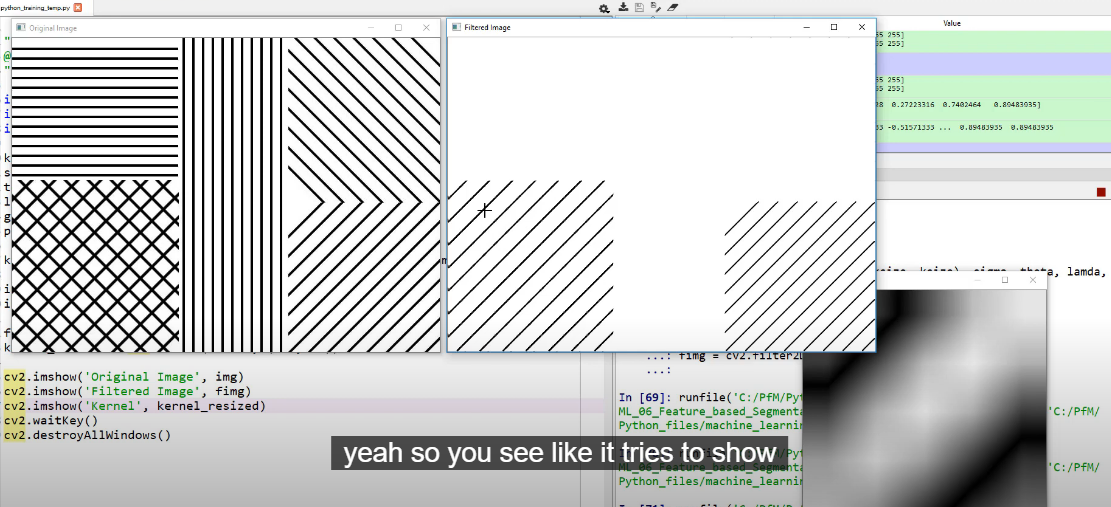
* Normalize Box Filter là bộ lọc làm mịn ảnh.

1. Gabor Filter

* Gabor is a convolution filter representing a combination of gaussian and a sinusoidal term.
* Chuyển ảnh về hệ GreyScale



* Gabor can be used to generate features that represent texture and edges.
* This is the band, this only passing the things that let them through and anything that in other action that the other direction you know completely blocking. The horizontal is gone and the vertical is gone.



* Gabor được cấu thành từ tích của hàm Gaussian và hàm cos (phần thực), hàm sin (phần ảo). Tham khảo: [Gaussian function](https://en.wikipedia.org/wiki/Gaussian_function)
* Công dụng của Gabor filter là phát hiện cạnh (trích cạnh) của ảnh.

1. Correlation / Convolution

* Correlation là toán tử tìm **sự tương quan** của cửa sổ trên ảnh gốc. Bạn để ý rằng, vùng pixel trên ảnh gốc có mẫu (pattern) **càng giống** với cửa sổ (kernel) thì giá trị tại điểm tương ứng của ảnh đầu ra **càng lớn**. Bạn hãy xem lại ví dụ correlation 1D sẽ thấy rõ điều này, vùng 3 pixel có phân bố càng giống kernel (-1 1 2 –> giá trị lớn dần và pixel cuối lớn nhất gấp đôi) thì giá trị đầu ra càng lớn, đây chính là [ý nghĩa của tích vô hướng](https://minhng.info/toan-hoc/y-nghia-tich-vo-huong.html).
* Điểm khác biệt lớn nhất của convolution với correlation đó là convolution có tính chất kết hợp: \(K2\*(K1\*I) = (K2\*K1)\*I\). Điều này thể hiện rằng thay vì ta lấy ảnh gốc I convolve với kernel K1, sau đó lấy ảnh kết quả convolve với K2 THÌ ta có thể thực hiện lấy kernel K1 convole với K2 thành 1 kernel nào đó, sau đó lấy kernel kết quả này áp dụng cho ảnh gốc I. Chính nhờ tính chất này mà khi thiết kế kernel, thay vì thiết kế nhiều phép convolve tuần tự ta có thể kết hợp chúng lại thành 1 kernel duy nhất. Chi phí tính toán của convolution / correlation là tương đối lớn.

1. Gabor Filter Extends

* Texture is an important feature of images.
* Gabor filters are spatial sinusoids localized by a Gaussian window, and they are orientation and frequency sensitive band pass filters.

1. LBPH Algorithm

* Divide the examined window into cells (e.g. 16x16 pixels for each cell).
* For each pixel in a cell, compare the pixel to each of its 8 neighbors (on its left-top, left-middle, left-bottom, right-top, etc.). Follow the pixels along a circle, i.e. clockwise or counter-clockwise.
* Where the center pixel's value is greater than the neighbor's value, write "0". Otherwise, write "1". This gives an 8-digit binary number (which is usually converted to decimal for convenience).
* Compute the histogram, over the cell, of the frequency of each "number" occurring (i.e., each combination of which pixels are smaller and which are greater than the center). This histogram can be seen as a 256-dimensional feature vector.
* Optionally normalize the histogram.
* Concatenate (normalized) histograms of all cells. This gives a feature vector for the entire window.
  1. Face & the Local Binary Pattern – Computerphile
* LBP looks at nine pixels at a time
* LBP looks at a little block of three by three pixels.
* It’s particularly interested at the Central pixel
  + Step1:
  + Step2: String of numbers which we then turn into a decimal number.
  + Note: The nice thing about these local binary patterns is that it is illumination Invariant -> If you change the lighting on the scene all these pixel values will go up but the relative difference between the pixels will remain the same