Chapter - 1

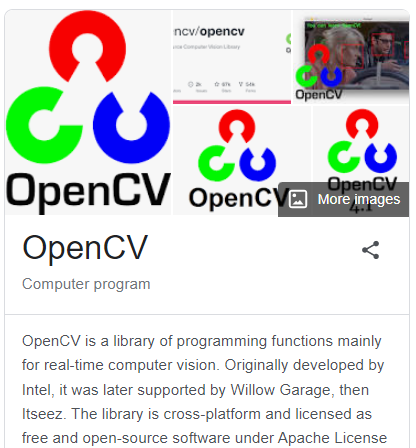
**Computer Vision**

**OpenCV: face detection**

Viola-Jones Algorithm, Haar-like Features, Integral Image,

Training Classifiers, Adaptive Boosting (Adaboost), Cascading

* 1. **Introduction**
* First we will learn all about the Viola-Jones functions, the intuition behind it and then we'll get the practical application in face detection.
* Then we're going to learn how to use deep learning in computer vision for object detection.
* Finally we'll learn about GANN and we'll use it to generate images.



**1.2 OpenCV objectives**

1. Viola-Jones Algorithm: What is it, how it was created, why it is still powerful.
2. Haar-Like features :We'll talk about *Haar-Like features* which are foundations of *Viola-Jones Algorithm* and is actually used in *OpenCV*.
3. Integral-image: Then we'll discuss *Integral-image* which is a "*HACK*" used in the *Viola-Jones Algorithm* as part of the success of the algorithm can be attributed to this Integral Image Hack.
4. Training Classifiers: How in the training process actually happens.
5. Adaptive Boosting (Adaboost): Which is another hack that helps the *Viola-Jones Algorithm* in its success.
6. Cascading: Also another hack which really speeds up the *Viola-Jones Algorithm* and that's why we have things

* like OpenCV and visual recognition which is so powerful.

**1.3 Viola-Jones Algorithm**

|  |  |
| --- | --- |
| This is the algorithm that lies the foundation of the OpenCV library. This is one of the most powerful algorithms for computer vision. This algorithm was developed by *Paul Viola* and *Michael* *Jones* and it was developed in *2001*.   * It's slowly being surpassed by Deep Learning. But nevertheless it is in its simplicity it's so powerful that it is still being used for computers. It's not just detecting ***faces*** on ***images*** but also for real time computer vision detecting ***faces*** on ***videos*** as well. |  |

* The Viola-Jones algorithm consists of two stages: The first stage is the *training*. The second stage is the *detection*.
* Detection (how a trained model works): Assume that our model is trained up and we're ***gonna*** see how it works in action.



* Notice the image is a *frontal face* and that's exactly what the Viola-Jones algorithm looks for. It's designed to look for frontal faces not on somebody looking to the side or up or down.
* Most of the time it's performs the best with **frontal face**.

|  |  |
| --- | --- |
| * The first step is to turn this image into a Gray-scale. * Note that the results are still astonishing. And there's just *less data* to *process*. * Once the face is detected (i.e algorithm finds that actual location or face on the color image), you won't even notice that it's working in grayscale, but *under the hood this Voila-Jones algorithm is working with a grey-scale version* of your image. |  |

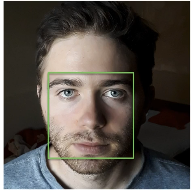
* How the image is scanned: (It is similar to CNN in Convolution operation and feature detection process). At the second step, algorithm starts with a *Window* (little box). This little *box* starts from *top* *left* *corner* and moves to the right.
* Algorithm looking for the face outlines.
* For each step this box moved from *left* to *right*, and then after reaching the right corner it moves *down* and starts again from the *left*.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

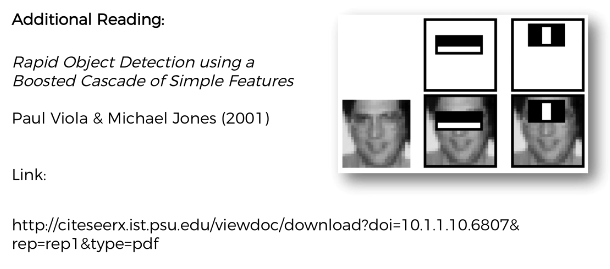
* It's looking for certain features of the face: like *eyebrows*, *eyes*, the *nose*, the *lips*, the *chin* the *forehead* the *cheeks* and so on. Note that for simplicity we consider those as a feature, but in reality these feature can be something else in algorithm.
* So this algorithm is *looking through the pixels* in this box, and when it can detect an eyebrow ( or a feature) and then it also looking for the other features (such as eyes, nose, mouth etc) it doesn't detect an eye (or other related features to be a face) then it decides that this is not a face.
* Then the box move on. Once again it detects tow eyebrow but same thing doesn't detect any eyes. So it's not a face.
* Finally when it got all the features, it can see that it's got both eyebrows, both eyes and nose and a mouth. So then it highlights this as a very high potential to be a face.
* Then this box moves one step forward and can see another face it can see two eyebrows, two eyes, and nose and mouth. So it highlights that again it keeps going on.
* So after this box moves to the end, where it can no longer see the full eyebrow or the full eyes so this is no longer a face.
* And then finally it stops.

|  |  |  |
| --- | --- | --- |
|  |  |  |

* And then it just find the same box on the color image too.

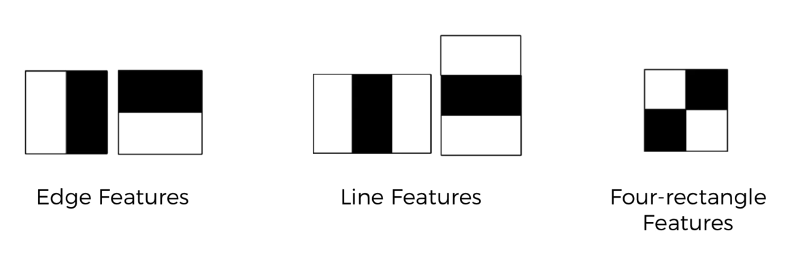


* That's how Viola-Jones algorithm scans this whole image.
* The size of this box varies because faces can be small or large.
* For *intuition* we consider the *big steps*, but in reality the *steps* to *moving* the box is *very small*. So in reality the steps are smaller.
* And the *face might* have been *detected* *many times*, because in reality the *box went through many times*, and the steps of the box were much smaller.
* Each box says that there's a high likelihood of being a face there.
* And when a lot of boxes overlap, that means that it is most likely a face and then it will detect the face there.
* And that's how you see that Green Square in your Facebook or in other kind of phase detection application.
* Additional Reading: Now if you would like to do some additional reading then the best place to start is the original paper by *Paul Viola* and *Michael* *Jones*. It's called Rapid Object dictation using a boosted cascade of simple features.



**1.4 Haar-like Features**

* Haar Wavelet: The name Haar comes from Alfred Haar who was a Hungarian mathematician in the 19th century and he developed a mathematical concept called the Haar Wavelet which is a wave functions, act similar to the Fourier transformation.
* Haar like features: The Haar like features derived from the Haar Wavelet. These features are the **EDGE** features, the **LINE** features and the **FOUR** **RECTANGLE** features.

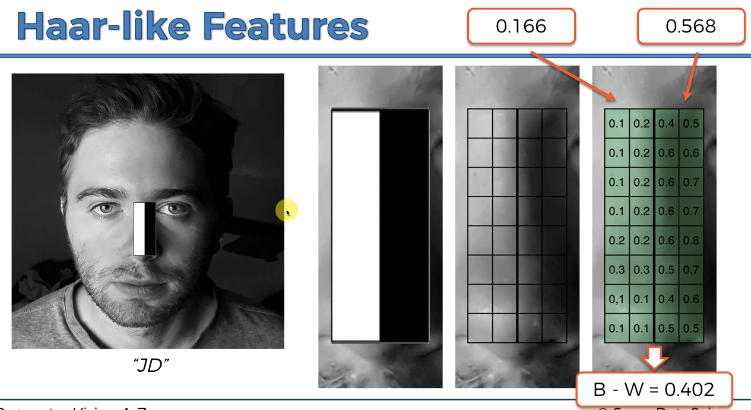


* Basically here we have is pixels. On an image you might have a *edge like pixels* with light and shadow (EG: nose of a face), we can also find a *line feature* (like eyebrow of a face or lips) and generally *Four rectangles* like light-shadow mix.

* These features are also scalable, can be longer or wider.

|  |  |
| --- | --- |
| * Lets see an example that how these features works on a face detection. * **Mouth:** ***Line feature***, because there is a dark shadow between the lips. * **Eyebrow:** The eyebrow is darker than the forehead so let's put a ***Edge-feature*** there. * So ***Edege-feature*** can identify where the ***break division*** like dark to light transition happens. * **Nose:** Part of the nose is brighter than the other part. The *bridge* of the *nose* is brighter than right-part. Basically any photo in grayscale you will often find that the bridge of the nose is brighter than either just one side or even both sides. It depends on the lighting. * So, sometimes you might find here like ***a Line-feature*** and some-times an ***Edge-feature***. In our case it’s an Edge-feature. * **Eye:** Line feature. Because middle of the eye is darker. |  |

* **Common features:** In general they're going to be a voting system that the *more features the more likely it is a face*. We just keep finding these features and some of them would be dependent on the lighting or the person. But overall we'll find a set of features that are commonly present on most human faces. Like at least 80% or 90% features will be present on a face.
* How this works in terms of the pixels: Let's talk about the *nose bridge* area that we've identified *Edge feature* over here. We're going to have the pixels. For example let's say there are pixels.
* We've given them values *(Gradings like a fuzzy set)* between ***0*** and ***1*** there is 0 = white and 10 = Black. (grayscale is from 0 to 255 but for simplicity we've normalized this value.)
* Most *white parts of the nose are closer to 0*, and *darker-part is closer to 1*.
* Process of calculation: Now the Viola-Jones algorithm will compare these Gradings to the real Harr-Like feature how close they are.
* It takes average of the white pixels (average of all 16 pixels = 0.168), and black pixels (al 16 black pixels 0.568).
* Then it subtract the average values, .
* Now for the ideal Haar-Like feature Difference is 1, i.e for Haar-Like feature (100% black and 100% white).
* So in the real world case, the difference is more closer to 1, then it is more closer to indentifiable Haar-Like feature.
* Threshold value: Of course you're never going to have an exact 1 here or an exact 0. This is always going to be some sort of gradients some sort of different colors shades and so on.
* That's why the Viola-Jones algorithm has some thresholds.
* Through the training process this Viola-Jones algorithm will find this threshold value.



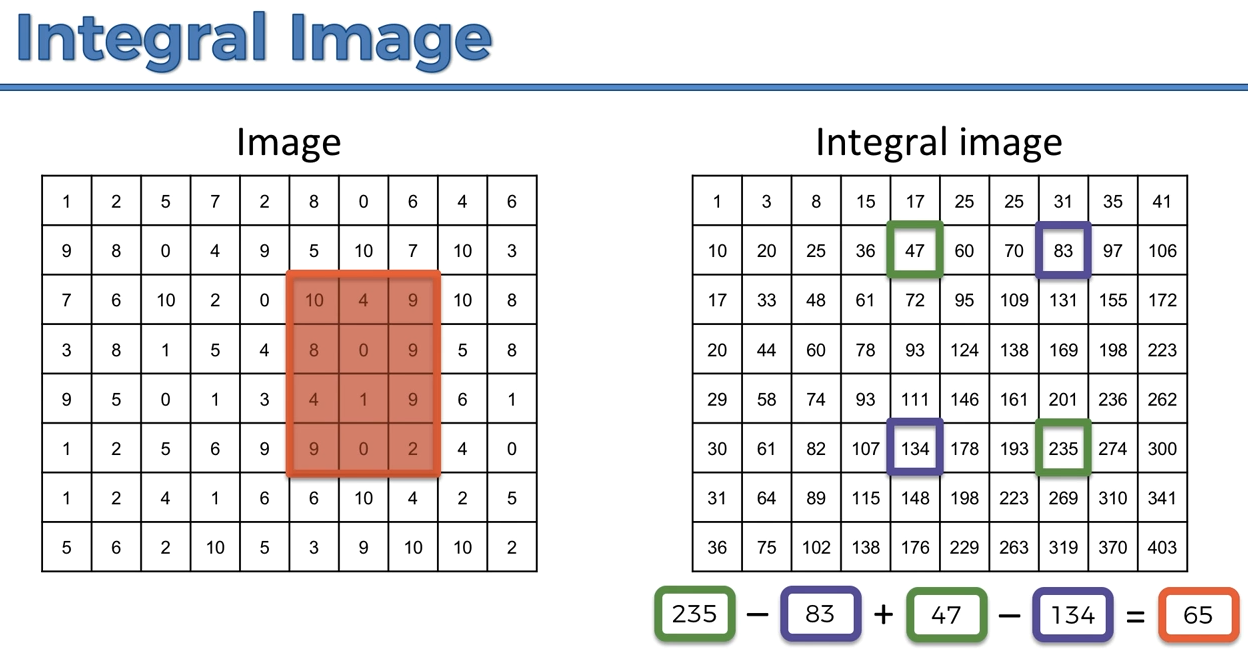
* For example, for the specific feature in the position of Nose it's identified that the minimum threshold might be 0.3.
* Whenever you get a result a calculated value here that is greater than 0.3 that means that indeed this is somewhat similar to this Haar-Like feature (represents a Nose) which we're looking for.
* However, the algorithms looks for this this Haar-Like feature (represents a Nose) all over the image and calculates the highest probability.
* If the featue not found then it wont recognize the face.

Once those features that were identified in the face the like for eyebrows and mouth and so on. For algorithm, these features forms the basis to understand what elements roughly a face is constructed from and then it will be able to use those to detect the face.

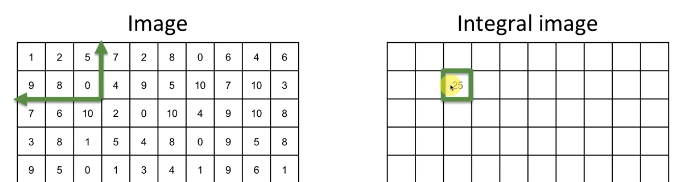
**1.5 INTEGRAL IMAGE**

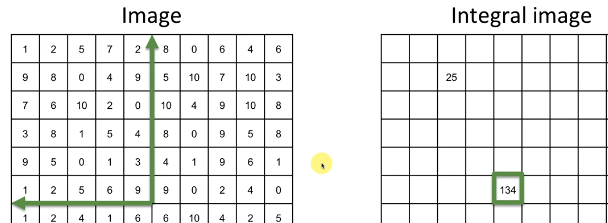
In previous section we decided the Haar-Like-Features but it can be quite a costly in terms of computations. If you have lots of these features at evaluating and especially if these features are quite large.

* Integral image is a hack to doing this very quickly.

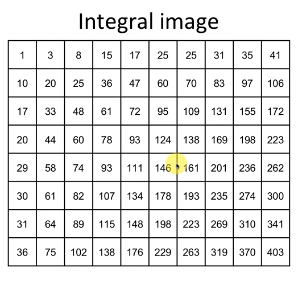


* Let's say we're calculating a Haar Like Feature and we need to calculate the *sum* of all of the *intensities* in this specific area that we've *outlined* to evaluate *feature* on this image.
* We need to perform 12 operation to get the sum.
* Now for Real-life this calculation may be too large, for example 1000 of operation to perform.
* This makes our algorithm too slow.
* Now the hack is, we can compute this kind of sum in special way.
* Forming an Integral image: First we calculate the sum for each pixels, and we sum all the values in the rectangle formed by this pixel (i.e from left start to that pixel).

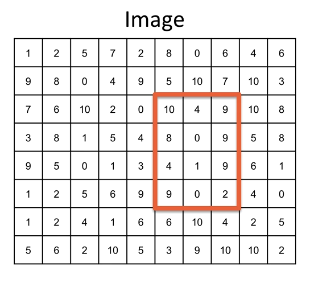




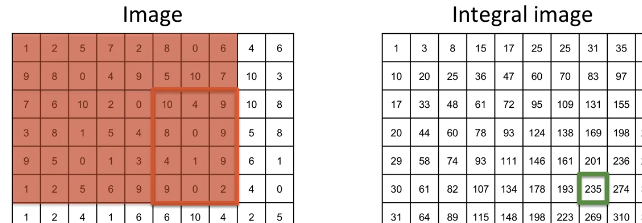
* Then finally we got following Integral Image:



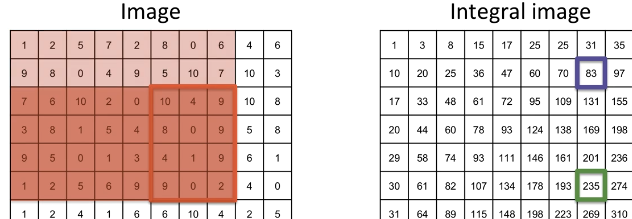
* How to calculate the sum: Suppose we want to calculate the sum of following rectangle from Integral image.



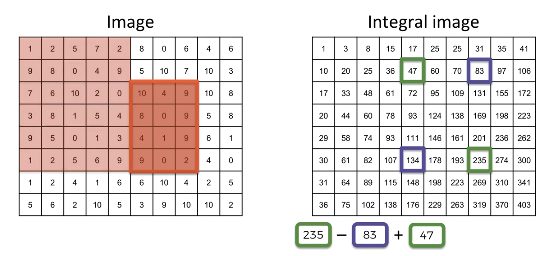
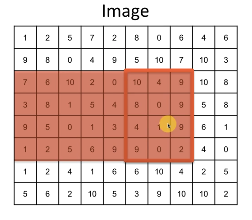
* Select the Bottom-Right corner from Integral Image.



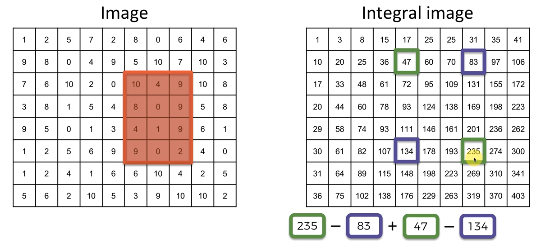
* Subtract Top-Right from Bottom-Right corner.



* Add Top-Left corner to the Result.



* Finally Subtract Bottom-Left corner from the Result.



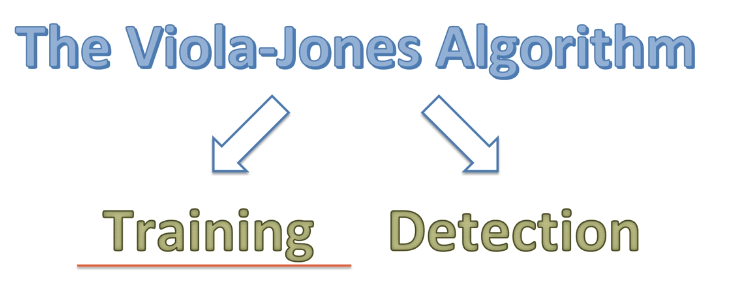
* Notice we are performed only 4-operations. It is always 4-operatios for any size of rectangle.

1. Find the Bottom Right corner
2. Subtract Top-Right corner from Bottom Right corner,
3. Add Top-Left corner.
4. Finally Subtract Bottom-Left corner.

That’s how we can reduce the operations to calculate Haar-Like features by Forming an Integral image.

**1.6 TRAINING Classifiers**

Above we discussed the Detection process. Now let's discuss the Training process.



|  |  |
| --- | --- |
| * During training process, we're training our model with these features: * The trick is that they're scalable, they can be shorter or longer, wider or narrower. And we need to identify which ones of them are actually **descriptive** for a **face** or are **common features** of a face. * The algorithm is trained about two things: * To identify the features that are related to detect a face: There can be multiple similar features present in a face. We need to help the algorithm understand which features are important to identify a face and also identify which features are not important. * Learn about the threshold to detect a feature: We know that , to detect a **Haar-like feature** from an **image** we need a *threshold* because a grayscale image is not 100% dark or white. * The training will also help the algorithm to understand those thresholds. Should they be set up **0.3**. or **0.7** or something else. * It needs to understand where to set those thresholds for the different features. |  |

* Procedure:
* During training we shrink all the image to  **pixels** and then it looks for these different features on that image.
* We shrink these image to reduce the features.
* Since these features are scalable, for a very large image, say pixels, then there will be lots of different variations of these features.
* There will be lots and lots of different combinations of these features (with different lengths and widths and heights of these features).
* And as a result the training will take much longer time.
* So it is easier to shrink all the image and find these features.
* ***After*** finding these ***features*** by ***training*** ***process*** we just ***re-scale*** them to apply ***bigger size image*** to detect the face.

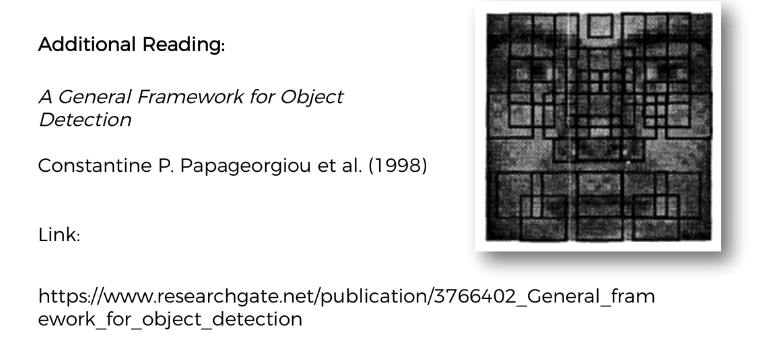
|  |  |
| --- | --- |
| * How to feed data: Representation fo data to the algorithm. * One image is not enough: * Algorithm might pick up the wrong things from this one image. * That's why you need to supply lots of images of *frontal faces* of *people* to the Algorithm. So that it can understand which of those *features* that it's looking for are actually *common* in any human face. * It can also *specify* the *non-important* *features* that are *actually not present* on most of the *other* *images*.   Here we've only got a couple of faces but in the real algorithm, in the actual original paper by Voila-Jones, they supplied their algorithm 4916 manually labeled faces. |  |

|  |  |
| --- | --- |
| * There are other tricks needs to apply: * Make RANDOM MIRROR-IMAGE from the available image. So from 4916 images we can get 9,832 images. * Supply Non Face Images: We need to make sure that those features are common just for faces and they're not common for something else. And that's why you need to supply also non face images. * We need to train our model that the identified features for a face are also not common in any other images. For this reason we need to also provide some other random images that are not representing any human face. * These images are some random photos objects everything. * But make sure that none of those images have faces in them. * Viola Jones supplied their algorithm 9544 of these images. |  |

* Note that these ones don't have to be **pixels** they're going to be any size (but there is also a limit, say under  **pixels** ).
* Divide a random non-face image: For a big image you can just take some windows from that image and treat each one as a separate image for training purposes.

Voila-Jones used about 350,000,000 sub windows that they had for training.

* Additional readings:

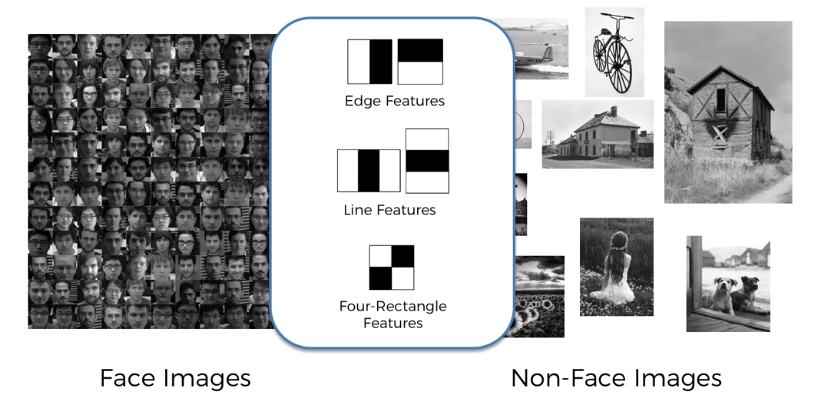


* It was written in 1998 before the paper by Voila-Jones and that's where the Haar-like features were introduced in the first place.
* On the other hand the original Voila-Jones paper has everything you need. It has all the details about the Haa-Like features and how everything works. That's a full enough resource.

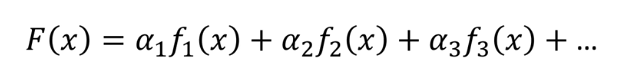
**1.7 Adaptive Boosting or ADABOOST**

In previous section, we discussed how the *algorithm* uses *input data* or *images* to find which features are important to recognizing faces. It also uses the **non-facial images** to understand the features that give it high rates of false positives.

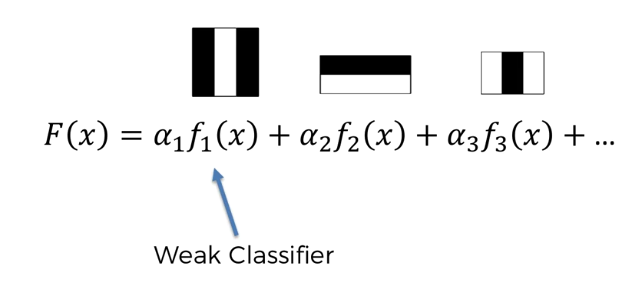
* And then it discards those features and makes them less important and focus on the features that are present in faces and faces only. And that's basically how the algorithm is trained.
* False Positive (Type I error): predicted as "it will happen", but in reality this "actually not happened". This is kind of "Warning". ***Predicted*** as ***Positive*** but in ***reality*** they are ***Negative***, hence False-Positive.
* False Negative (Type II error): predicted as "it won't happen", but in reality this "actually happened". This is kind of "fatal Error". ***Predicted*** as ***Negative*** but in ***reality*** they are ***Positive***, hence False-Negative.
* False negative is a bit worse.



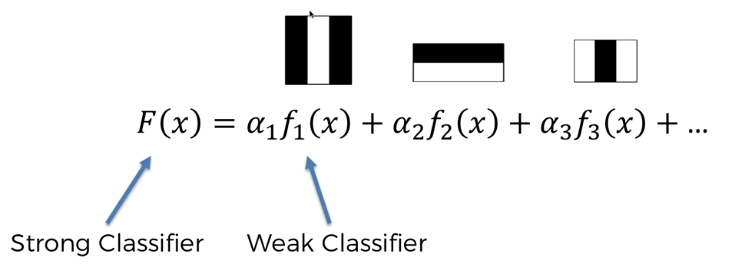
* Then we have these different features and also thresholds *(at which they are considered to be present in an image).*
* Too many Features: Now the problem is, there are too many features that detected in **px** images, nearly **180,000+**.
* The reason for that is that these features are scalable. They are not fixed at some specific pixels.
* So not only are we looking at these features on all their different positions in this image but also all of the size-variations of each one of these features.
* Hence, they end up **180,000+** possible options that we would have to explore.
* Now it poses two concerns:
* Hard to train: You need to check **180,000+** features for all 9832 faces in the **training data**. And also in ***10,000*** of non-facial images. Hence the training becomes quite long.
* Hard to test: Let's say somehow we managed to train our algorithm with all of those features, now when we're detecting the faces in test image, we have to check **180,000+** possible features every single time. So testing become slow.
* Adaboost: Adaboost solves this problem. We're going to take our *features* and we're going to put them together into a *classifier*.



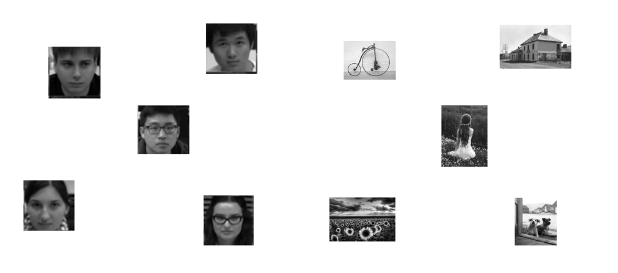
* Here is the classifier and , , are the features.



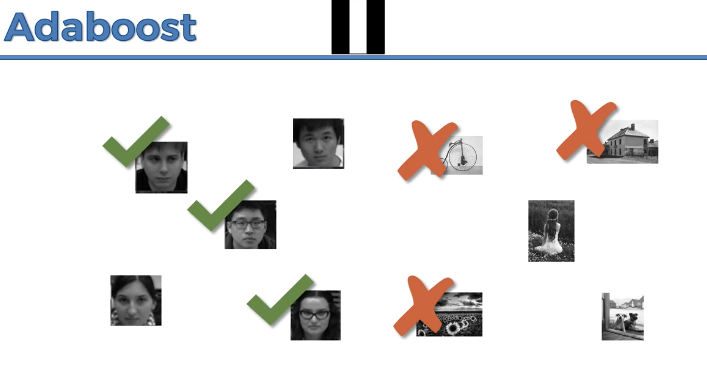
* Weak Classifier : So we've got our features aligned and it keeps going like that. And *each* one of these *features* is called a *weak* *classifier* on its own. It doesn't get a very *high rate of success* (as long as it gets over 50% that's already good).
* Strong Classifier (Ensemble Method): When you have one weak-classifier by itself it's not as good (it maybe has a 60% success rate). But when you have two week-classifiers together (even though 60% or 55%) they're much stronger.
* And as you keep *adding* these *weak-classifers*, they get stronger.
* The point is, you don't need all **180,000+** of them. You might just need a like 2000 or 3000 from these weak-features to get a really good result, a very strong classifier.
* And this is called an Ensemble Method because you are leveraging the power of the crowd. It's more like relating those separate **weak-classifier** together to make a **strong-classifier**.



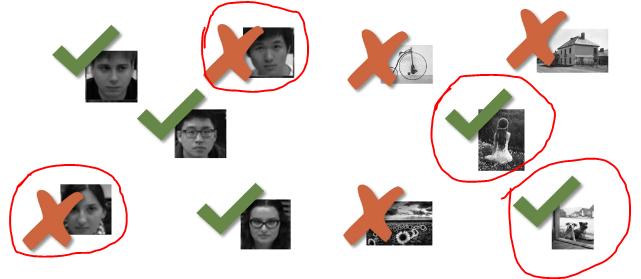
* Finding features:
* How to find right features (best weak classifiers): This is where adaptive boosting comes into play. Let's say you have 10 pictures you have *five faces* and *five non-faces*.



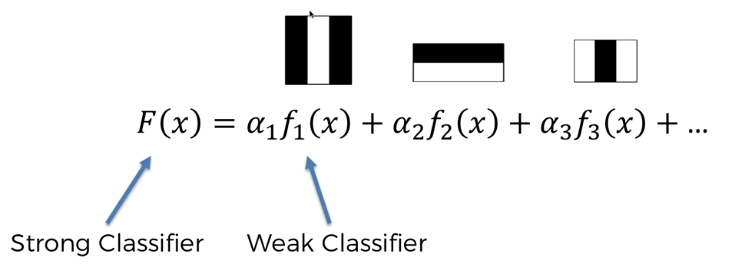
* Now we want to find the beast features to build that strong classifier . Let's say for a certain feature the model identify the following images: It detects *3-faces(feature is present)* and *3-non-faces(feature is absent)*.

****

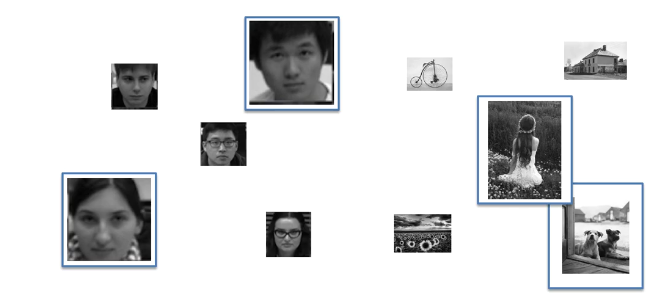
* Also we have FALSE-NEGATIVES and FALSE-POSITIVES errors: Two false-ve(Type II) on the left and two false+ve(Type I) on right.



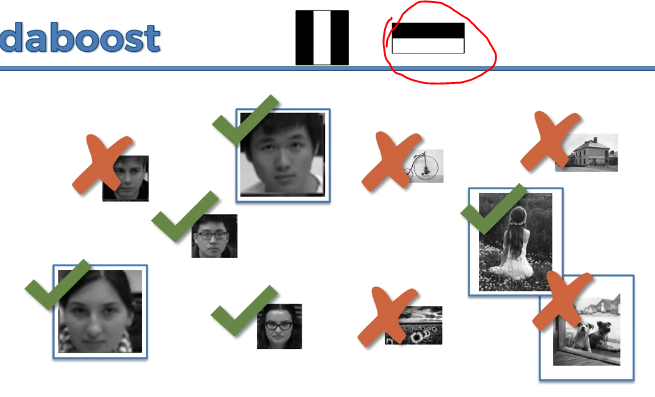
* Complement Features: Now what the AdaBost does, it just not only finds the important features, it also finds the complement-feature.
* It don't just combine the strongest features (because they might be leveraging off very similar things)
* It also finds the complement of previously extracted features which will help it to fix where extracted feature is strong and also where it's weak (where it's making mistakes).
* This will help complement that area and will help improve the performance in those areas which are falling behind in this feature by only-itself.
* For example in following, is a complement feature of and is complement of and . And so on.



* We're not just taking the strongest features we can find, we're staying a strong feature (or the strongest) and then we take in the one that best complements this one and then we're taking the next one which best *complements of those previous two* and the next on and so on.
* So basically instead of just taking the strongest-standalone-feature all the time we're constructing the strongest classifier.
* By leveraging the strengths of and then fixing up its weaknesseswith and then fixing up and weaknesses with and so on.
* So in the next round the AdaBoost will look for something that complements previous features that we found already.
* How it'll do that: It will give more weight to where the errors were made.
* Makes the *error* so *important* that the next *Haar-like feature* will found from these mistaken-images (kinda learn from the mistakes).



* It makes the *importance (weight)* of these images *higher* and it's going to find a *feature* that treats them the best that helps I *classify* them properly.
* That’s the way to fix the problems of the first feature that we had so.
* Now it applies to classify these five images correctly. And finds the corresponding feature (classifier).



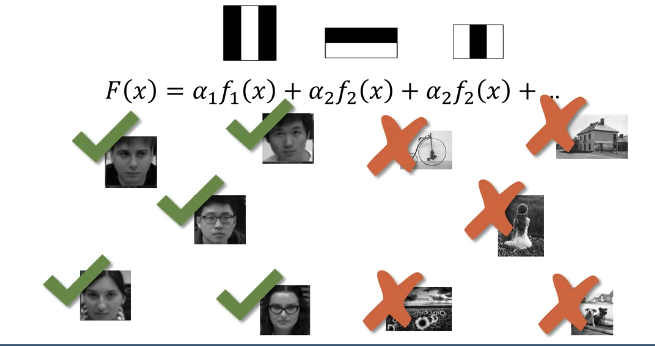
* And in this step, we got also some errors, **false-ve** and **false+ve**. Then the same processes are followed to found next complement-feature.



* So next we get another feature:



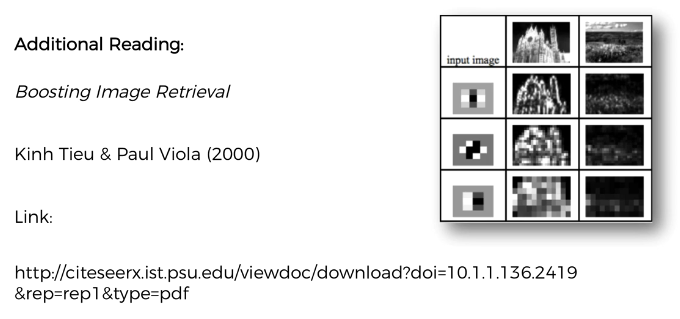
* Our final *goal* is to find an *overall (strong) classifier* **,** that classify all the images correctly.



* The point is to keep building this strong classifier through these weak classifiers and minimizing the error rate until you get to a satisfactory level, until you get to a very high percentage of correct classifications like 99%.

And at that point you don't need the 180,000 features, you just need that first couple of thousand (like 2000 or 4000 features) to get a satisfactory level.

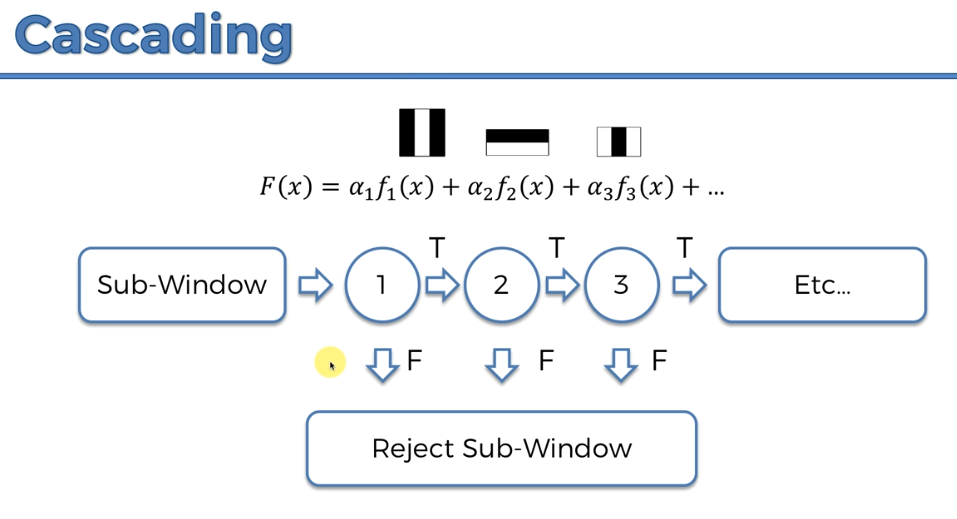
* CASCADING: That's the first step of AdaBoost what we discussed in this section. And then the next step is going to be the CASCADING.
* And once you combine **AdaBoost** and **CASCADING** together you'll see that it actually improves the efficiency substantially.
* Additional Reading: If you'd like to read a bit more about AdaBoost there is a paper by *Kinh Tieu* and *Paul Viola*. It's called ***Boosting Image Retrieval***. And this paper inspired some of the work in the Voila-Jones paper which we were talking about at the start.



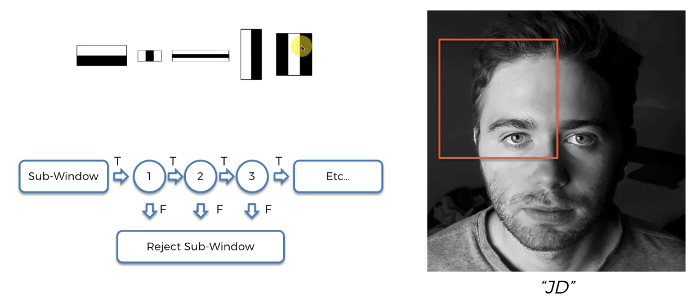
**1.8 CASCADING**

Above we discussed about the *Strong Classifiers* and how they're constructed from *Weak classifiers*.

CASCADING is another hack that the Voila-Jones algorithm applies in order to speed up the process.



* Cascading: Don’t need to find all selected **weak features** in a **sub-window**.
* We take a sub-window (window that is sliding across our image) that we're evaluating the features.
* In that window we look for the first feature (high-weight) in all or list of weak- features, it's present on that same window we go to the second weak-feature (second highest weight),
* If it doesn't satisfies the first feature, we reject the sub-window and take another sub-window, and start looking for the features all over again.
* The point of *cascading* is, we don’t need to evaluate all the following (*low weighted*) *features* if the *high-weighted features* are not present.
* In reality we do not take single features for cascading, actually we take a group/batch of features, and check if most of them are present in the window, and then we move to the *next group of features* to evaluate on that window. We reject the window, if any group of the features are ***not present* *in that* *window, don’t need to evaluate following features in that window***.
* For example: we consider first 12 features, if these are *present* in the *sub-window*, we move to the next 25 features. If they are *not* *present*, we *reject* the *sub-window*, and we *don’t need to evaluate other 1967 or 2967 features on this window*.



* That’s how CASCADING speeds up the process.