Viola-Jones Algorithm Face Detection Intuition

06.10.2019

Computer Vision

Viola - Jones Algorithm Face Detection Intuition

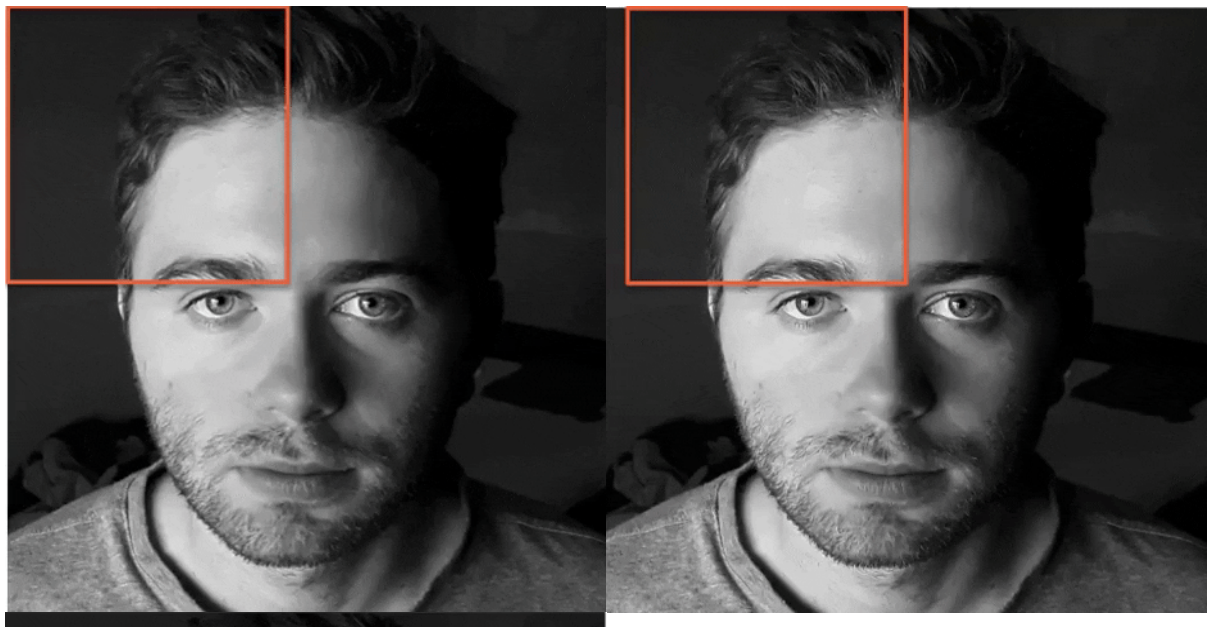
- It is one of the most powerful to date algorithms for computer vision.
- It’s slowly being surpassed by deep learning but nevertheless its simplicity is so powerful that it is still being used.
- Like other classifiers it has 2 stages: Training and Detection. We will first talk about the detection and then about training.

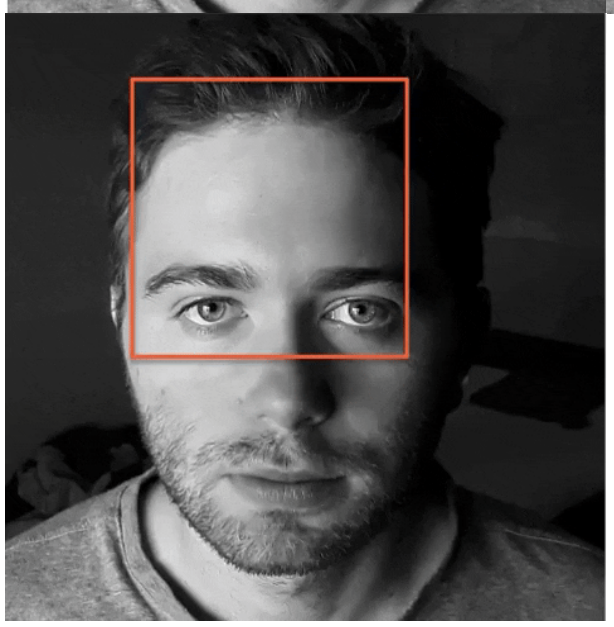
- So let’s try to understand how a trained model detects a face:

    - Let’s say I have a frontal face image like below. That is exactly what algorithm is needed. Most of the time the Viola-Jones algorithm performs the best with frontal faces.

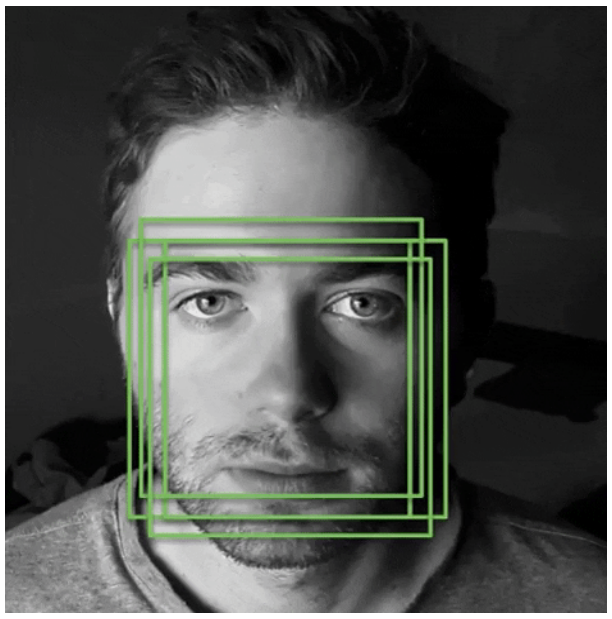
- First algorithm turns image into gray scale under the hood.It means less data to process

- So algorithm starts looking for some features of the face. It outlines a little window starts from the top left corner and moves to the right. Algorithm looks for the features inside this window. They are called Haar-like features but we'll talk about them in the next section so for now let's assume that the algorithm is looking for some specific features like eyes, eyebrows, mouth, nose, forehead, cheek and so on…





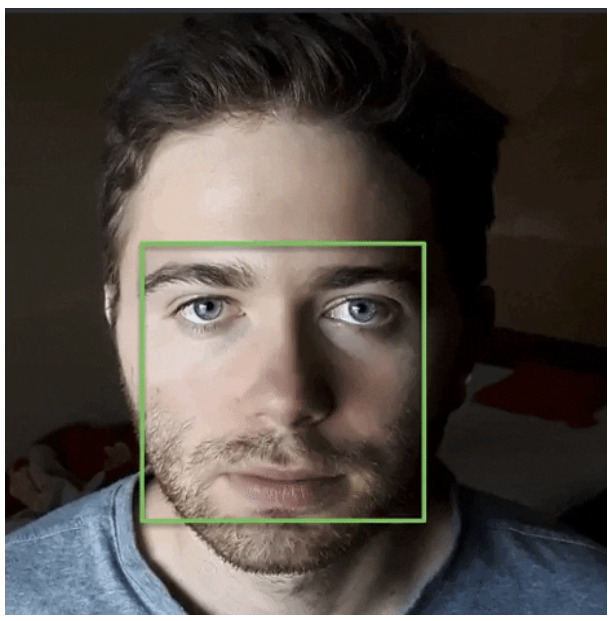
- If it detecs a single eye and a eyebrow, these features are not enough to detect a face. So the algorithm continues to seek.
- When all the features needed are detected algorithm marks the current window and says here I found a face.



- Still algorithm will look for all the windows, if it detects a face in another window again it will mark this new window as well.

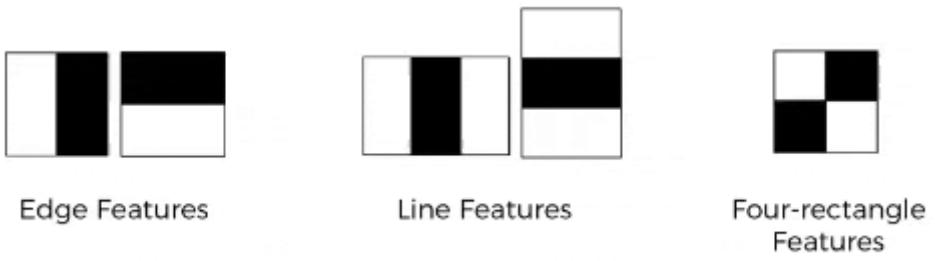
- Note that the **size of this window varies** because faces can be small or large. This **whole process will be repeated many times with smaller and bigger windowss**. Also the horizontal and vertical window steps will be smaller in reality.

- After the detection of the face, algorithm will place the window of the same position on the colored image too.



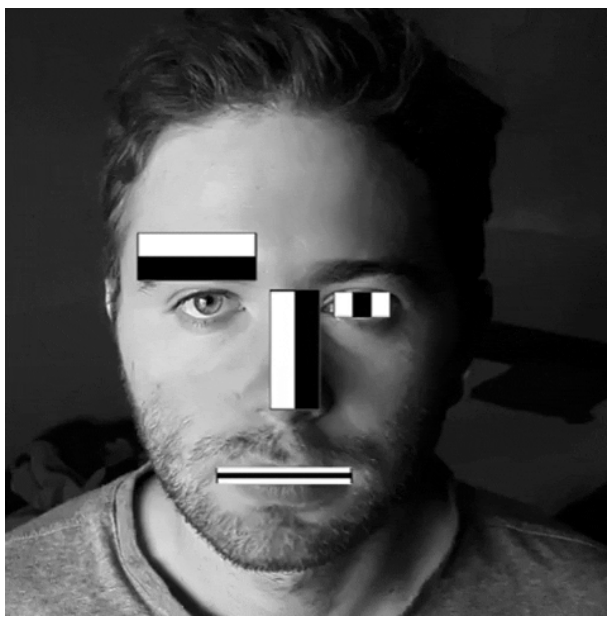
Haar-like Features

- There are 3 basic Haar-like features that Viola and Jones identified in their paper.



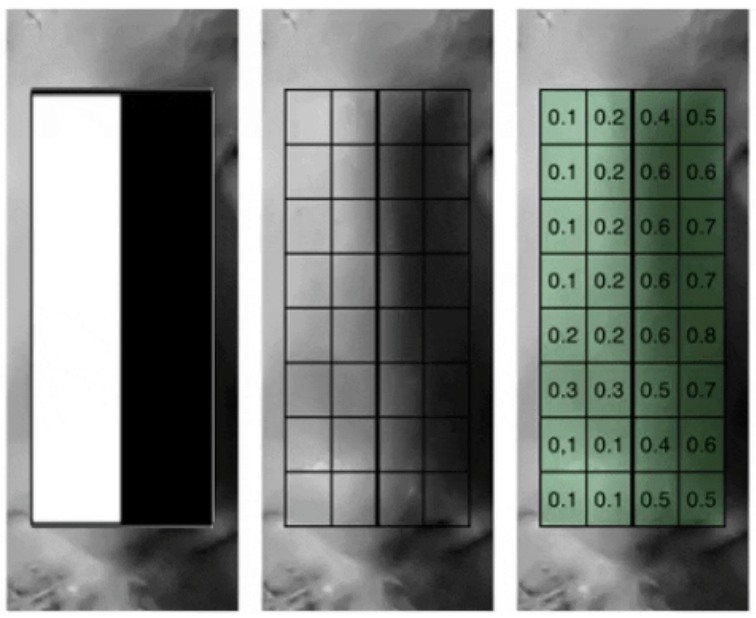
- These are all scalable. Different scaled ones are accepted as a separate feature.
- Let's have a look at how these features work out in terms of a face

    - If we take a look at the mouth we can see that there is a tiny black line between the lips. So that can be accepted as a Haar-like line feature.
    - Or an eyebrow can be seen as an edge feature
    - Nose can be seen as an edge feature as well
    - Also an eye can be seen as a line feature as well

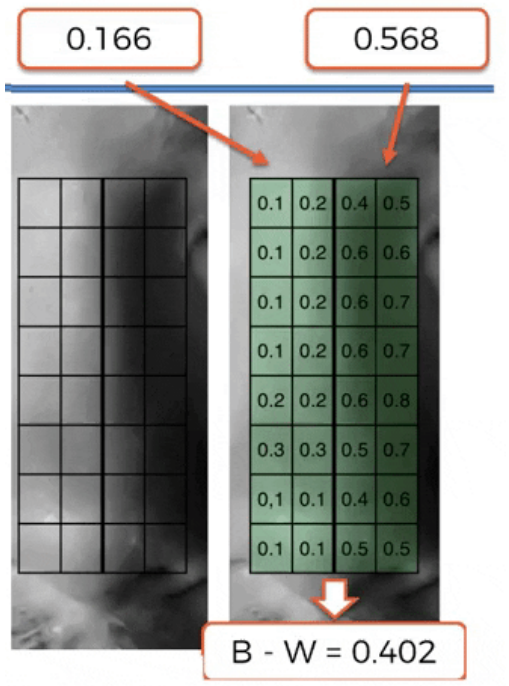


- Thus the detection algorithm is actually looking for these Haar-like features window by window.
- If the algorithm detects enough features for a window it decides that: yes there is a face inside this window.
- But there is a question how does it know which features to look for? That is for training section. For now let’s try to understand **how the algorithm decides if a specific haar-like feature exists in a window**?

- To gain a better understanding about how the algorithm decides if a specific feature exists let's zoom in to the edge feature on the nose:



    - For simplicity let’s say there 8x4 = 32 pixels for this specific feature.
    - Each pixel has its pixel intensity value. 0 means 100% white and 1 means 100% black.
    - For an ideal case, and edge feature has one side 100% white and the other side 100% black. But in application it is grayscale.
    - So what the algorithm will do is, compare how close this 32 pixels nose area to the ideal edge feature.
    - It calculates the average pixel intensity for left and right 16 pixels regions. Then it subtracts one from the another.



- For our case the result is 0.568 – 0.166 = 0.402. For the ideal case the result would be 1 – 0 = 1
- So the closer the result is to 1 the closer that this feature exists.
- But for what value of result, we can say that this feature is present?
    - The training algorithm sets the features and the tresholds but it is for another topic.
    - Let’s say the treshold is set as 0.3 for this specific edge feature. Then the detecting algorithm would assume that this feature is present in this specific window we are looking for.
    - So algorithm looks for everywhere in the image and looks for the feature when it detects a 32 pixels area that gives a result bigger than the treshold then it says: this feature exists in this specific window.
    - Note that there might be thousands of features and algorithm does the same thing for all of them.

Integral Image

- So in order to decide that if a specific feature exists in a window we need to make certain calculations over and over again.
- For the current window we need to look for a specific feature by making additions, subtractions, divisions again and again.
- As you can imagine this process can be quite a costly exercise in terms of computations.

- Fortunately there is a hack for making these calculations quickly. Integral images.
- Integral image that allows for very fast feature evaluation
- The integral image can be computed from an image using a few operations per pixel. Once computed, any one of these Harr-like features can be computed at any scale or location in constant time.

- Let's say we have an image with a certain number of pixels as shown below and for simplicity's sake we will represent pixel intensities between 0 and 10:

1	2	5	7	2	8	0	6	4	6
9	8	0	4	9	5	10	7	10	3
7	6	10	2	0	10	4	9	10	8
3	8	1	5	4	8	0	9	5	8
9	5	0	1	3	4	1	9	6	1
1	2	5	6	9	9	0	2	4	0
1	2	4	1	6	6	10	4	2	5
5	6	2	10	5	3	9	10	10	2

- Let's say we are calculating a Haar-like feature and in order to calculate that feature on this image we need to calculate the sum of all the intensities in the red box.
- Adding each pixels one by one would be computationally expensive especially when the size of the red box become larger.
- So in order to solve this problem, integral images will be used. Let's see how integral images are constructed:
- Integral image is exactly the same size as the original image.

Image

1	2	5	7	2	8	0	6	4	6
9	8	0	4	9	5	10	7	10	3
7	6	10	2	0	10	4	9	10	8
3	8	1	5	4	8	0	9	5	8
9	5	0	1	3	4	1	9	6	1
1	2	5	6	9	9	0	2	4	0
1	2	4	1	6	6	10	4	2	5
5	6	2	10	5	3	9	10	10	2

Integral image

		25							

- So the specific pixel value on the integral image calculated by adding all the specified pixels in the original image.

Image

1	2	5	7	2	8	0	6	4	6
9	8	0	4	9	5	10	7	10	3
7	6	10	2	0	10	4	9	10	8
3	8	1	5	4	8	0	9	5	8
9	5	0	1	3	4	1	9	6	1
1	2	5	6	9	9	0	2	4	0
1	2	4	1	6	6	10	4	2	5
5	6	2	10	5	3	9	10	10	2

Integral image

		25							
				134					

- The integral image can be obtained as below by repeating the same process for every single pixel:

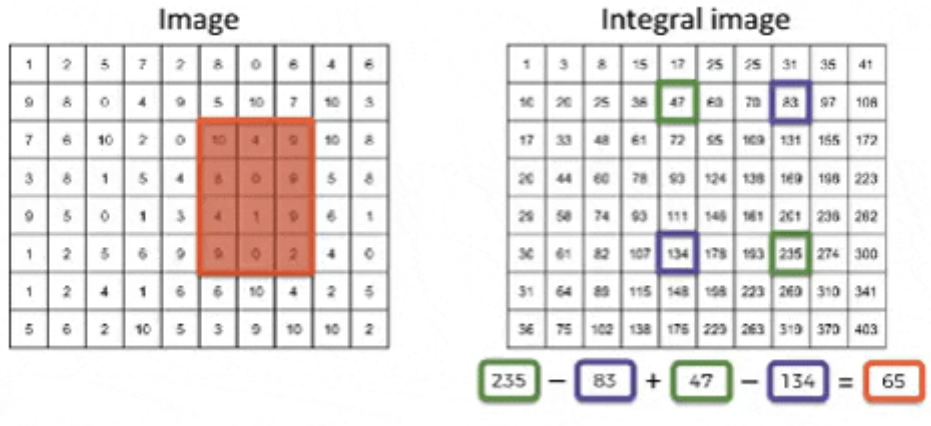
Image

1	2	5	7	2	8	0	6	4	6
9	8	0	4	9	5	10	7	10	3
7	6	10	2	0	10	4	9	10	8
3	8	1	5	4	8	0	9	5	8
9	5	0	1	3	4	1	9	6	1
1	2	5	6	9	9	0	2	4	0
1	2	4	1	6	6	10	4	2	5
5	6	2	10	5	3	9	10	10	2

Integral image

1	3	8	15	17	25	25	31	35	41
10	20	25	36	47	60	70	83	97	108
17	33	48	61	72	95	109	131	155	172
20	44	60	78	93	124	138	169	198	223
29	58	74	93	111	146	161	201	236	262
30	61	82	107	134	178	193	235	274	300
31	64	89	115	148	198	223	269	310	341
36	75	102	136	176	229	263	319	370	403

- Then the integral image can be used to evaluate a feature very quickly. Let's see how:



- Just by performing 4 operations on the integral image, we are able to calculate the sum of the values inside any size of a rectangle on the original image.
- So if we are trying to calculate the sum of a 1000 pixels to 1000 pixels area we are able to find the results just by performing 4 operations instead of a 1000000 operations
- Remember all Haar-like features are rectangles thus the concept of integral image allow us to evaluate features very quickly.

Training Classifiers

- I have explained how a trained model can detect a face.
- Basically the algorithm scans the image window by window and for each window it is looking for the presence of some specific Haar-like features.
- And based on how many of these features (there are also weights for each feature) are exist, the classifier decides if there is a face in the current window.
- Let's say the algorithm is looking for a 8px to 4px feature. (Note that the size of the feature can be scaled based on the window size. Because there might be smaller or bigger faces on the image.)
- For every window it should check every possible 8px to 4px areas and calculate a result based on the pixel intensity for each 8x4 areas and if the result is bigger then a specified treshold algorithm can decide that the feature is present in this specific window.
- And If I want to check the absence of a smaller faces on the picture. Then smaller windows will be used for the scanning and the features will be scaled based on the new window size.
- Also we've seen that the concept of integral image is very helpful during the feature evaluation.

- Now we will discuss that **what features and what tresholds should the algorithm look for ?**
- For a 24px to 24px image there are 180 000+ possible features. We need to identify which ones of them are acutally descriptive for a face.
- Also for each feature we need to specify a treshold.
- Both the descriptive features and tresholds for each feature will be determined by a Adaptive Boosting or AdaBoost algorithm.

- For now let's try to gain a general intuiton about the learning process and then in the next section I will be explaining the AdaBoost in a more detailed form.

    - For the training purpose a lot of face and non-face labeled images are given to the algorithm.

Face Images

Non-Face Images

- First the algorithm shrinks the images into 24px to 24px formats. (Smaller the images less possible features there are)
- And then algorithm looks for the descriptive features and their optimal treshold values. Basically it decides which features are common in human faces and which features are irrelevant.
- Note that, when the detection algorihm is working, instead of a shrinking the image, the features are scaled up.

Adaptive Boosting (AdaBoost)

- Recall that there are over 180,000 rectangle features associated with each image sub-window (24px to 24px).
- Even though each feature can be computed very efficiently, computing the complete set is prohibitively expensive.
- Very small number of these features can be combined to form an effective classifier. The **main challenge is to find these features**.

- In support of this goal, the weak learning algorithm is designed to select the single rectangle feature which best separates the positive and negative examples
- For each feature, the weak learner determines the optimal threshold classification function, such that the minimum number of examples are misclassified.

A weak classifier $h_j(x)$ thus consists of a feature f_j, a threshold θ_j and a parity p_j indicating the direction of the inequality sign:

$$h_j(x) = \begin{cases} 1 & \text{if } p_j f_j(x) < p_j \theta_j \\ 0 & \text{otherwise} \end{cases}$$

- Here x is a 24x24 pixel sub-window of an image. See Table 1 for a summary of the boosting process.
- In practice no single feature can perform the classification task with low error. Features which are selected in early rounds of the boosting process had error rates between 0.1 and 0.3. Features selected in later rounds, as the task becomes more difficult, yield error rates between 0.4 and 0.5.

- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \ldots, T$:
  1. Normalize the weights,
  $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
  so that w_t is a probability distribution.
  2. For each feature, j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t, $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$.
  3. Choose the classifier, h_t, with the lowest error ϵ_t.
  4. Update the weights:
  $$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$$
  where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- The final strong classifier is:
$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$
where $\alpha_t = \log \frac{1}{\beta_t}$

Table 1: The AdaBoost algorithm for classifier learning.

- Each round of boosting the algorithm selects one feature from the 180,000 potential features.

Let's try to understand what is going on for the AdaBoost algorithm in a more intuitive way:

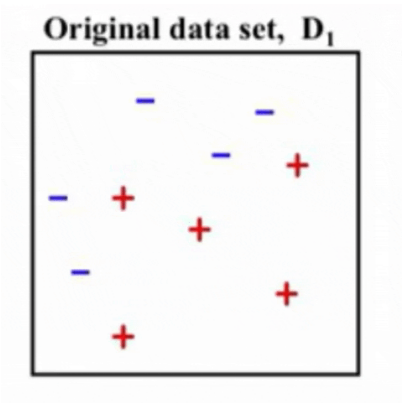
Here is the general algorithm :

- Assume I want to train a classifier which uses T features instead of 180 000 where T<180 000
- For t = 1,...,T
  - For each feature (180 000+) train a classifier h(x) which is restricted to use a single feature.
  - Choose the classifier with the lowest error.
  - Error is evaluated like :

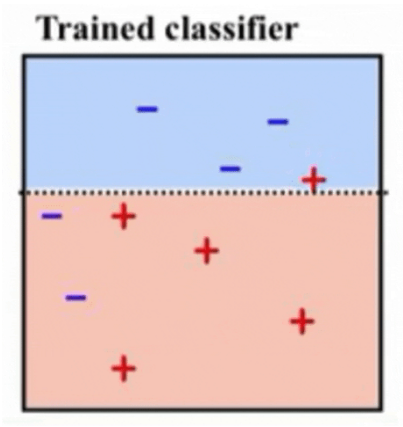
Weighted average loss:
$$J(\theta) = \sum_i w_i J_i(\theta, x^{(i)})$$

  - Where **wi** is the weight of the ith example and **Ji = h(x)-y** where **h(x)** is the guess for the specific example (0 or 1) and **y** is the actual result for a specific example (0 or 1).
  - Update the weights **w** of each training example. Increase the weight if it is not classified correctly.
- Construct the final strong classifier by using selected T features and assigning required weights to them: **hfinal(x) = a1 * h1(x) + a2 * h2(x) + a3 * h3(x) + ...+ aT * hT(x)**

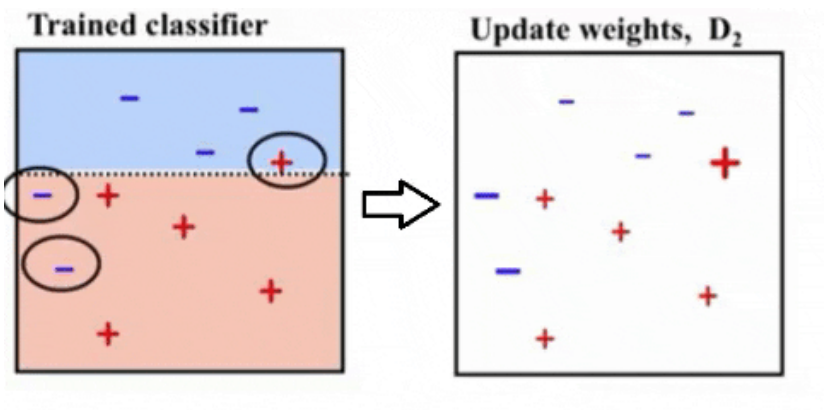
- Let's say we have a data set as below. Assume positive signs represent face data and negatives represent non-face data.



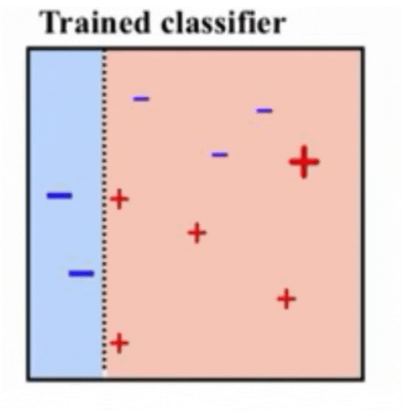
- Now by using a single feature (best performance among 180 000 features), algorithm has trained a classifier h(x).
- So, by trial and error method, algorithm has determined the strongest performance feature and its treshold.
- If the feature exists in an example classifier guesses face otherwise non-face.



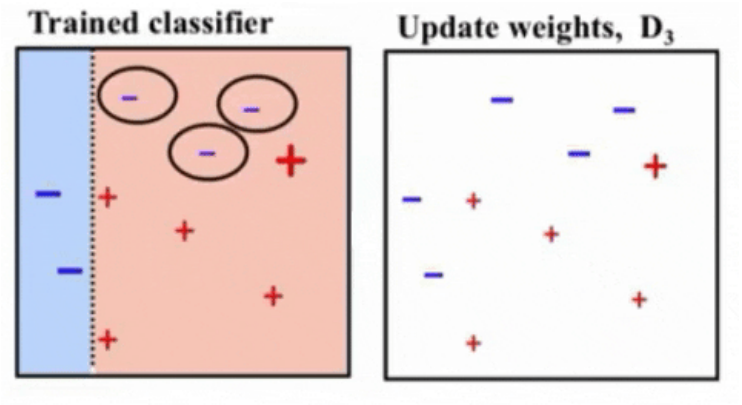
- For the next step, the weights of the misclassified examples are increased and the weights of the correctly clasified examples are reduced:



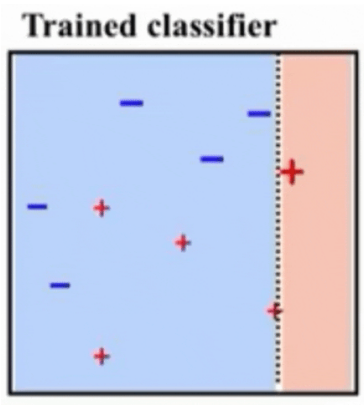
- Thereby we are focusing on the misclassified examples for the selection of the next feature.
- Now we are training another single feature classifier h(x), but this time some examples has more weight.



- Note that the boundary line on the figure represents a single feature classifier with a specified treshold. So it can correctly classify most of the examples.
- For the next step, again the weights for each examples are updated. Increased for misclassified ones and decreased for the correctly classified ones.

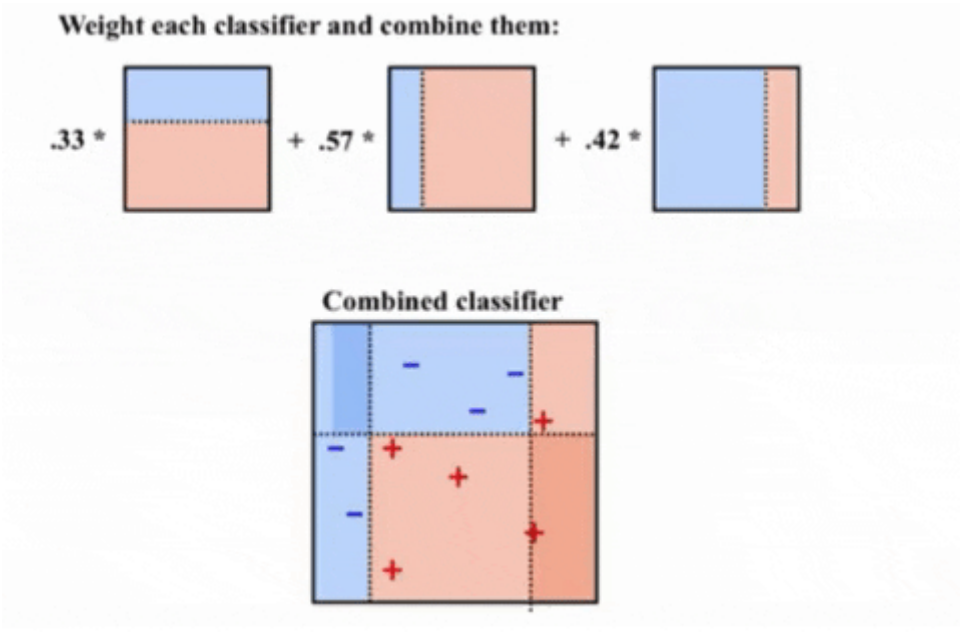


- And again another single feature classifier is trained by primarily focusing on getting the larger data points right.



- This feature selection process can continue like this until the Tth classifier. I will stay here for the sake of simplicity.

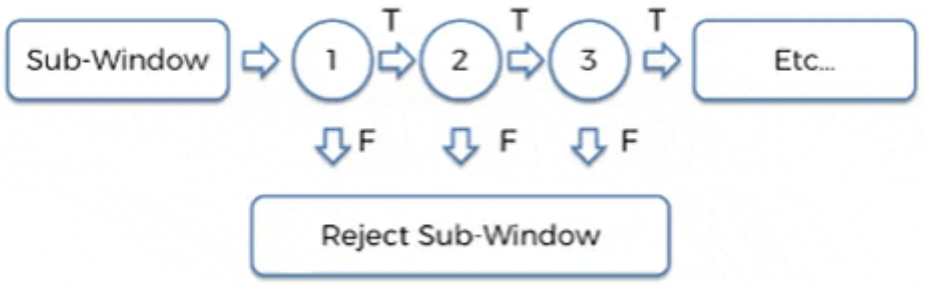
- **In the end each of these classifiers is assigned a weight and their weighted combination becomes the overall predictor.**



- **So a more complex and higher performance classifier is obtained by combining single feature classifiers.**

Cascading

- Cascading is another hack that the Viola-Jones algorithm applies in order to speed up the process.
- Here how it works, we take a sub-window or a window and we look for the first feature (most important feature).



- If the first feature is not present in the current sub-window then we reject the current window and we don't look for the rest of the features.
- It can be tought like: If there is no nose in the current window then no need to look for the eyes or eyebrows it can not be a face.
- If the first feature is present then we can look for the second feature.
- If the second feature is not present then reject the window move to the next one.
- If the second feature is present then we can look for the third one. It goes like this...

- In reality it is a little bit more complex. At the first step instead of checking for a top feature it checks for the top 5 or more features and it decides based on them.
- At the other stages it can check for more features because they are getting less and less important.

References:

- Rapid Object Detection using a Boosted Cascade of Simple Features
- SuperDataScience Team
- https://www.youtube.com/watch?v=ix6IvwbVpw0

More blogs