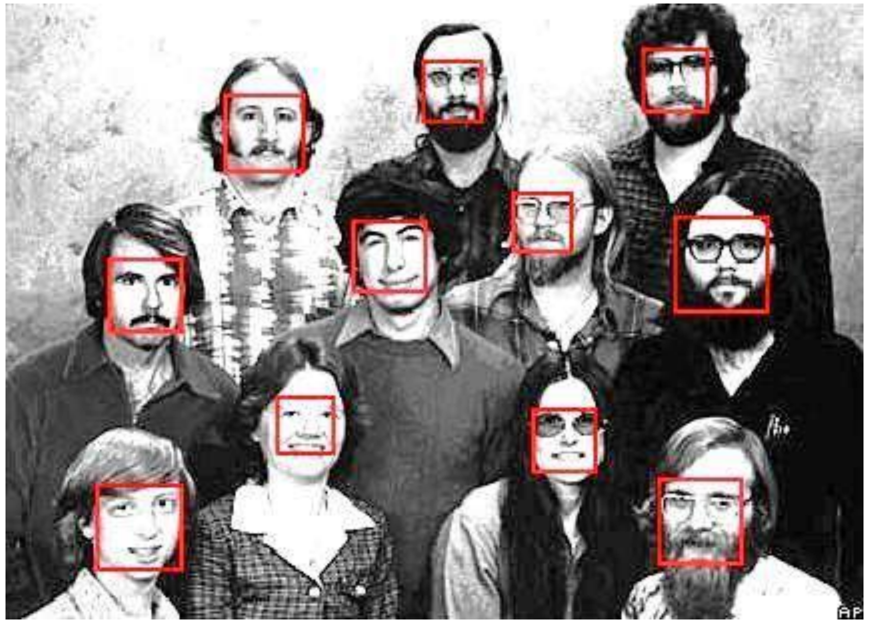
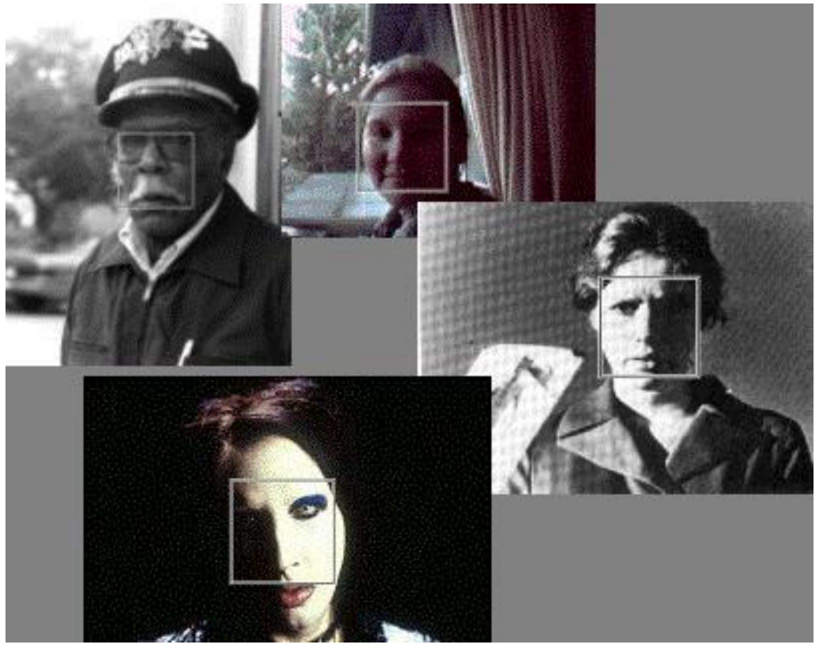
Viola Jones Face Detection

Shahid Nabi
Hiader Raiz
Muhammad Murtaz

Face Detection





Train The Classifier

Use facial and non facial images

Train the classifier

Find the threshold value

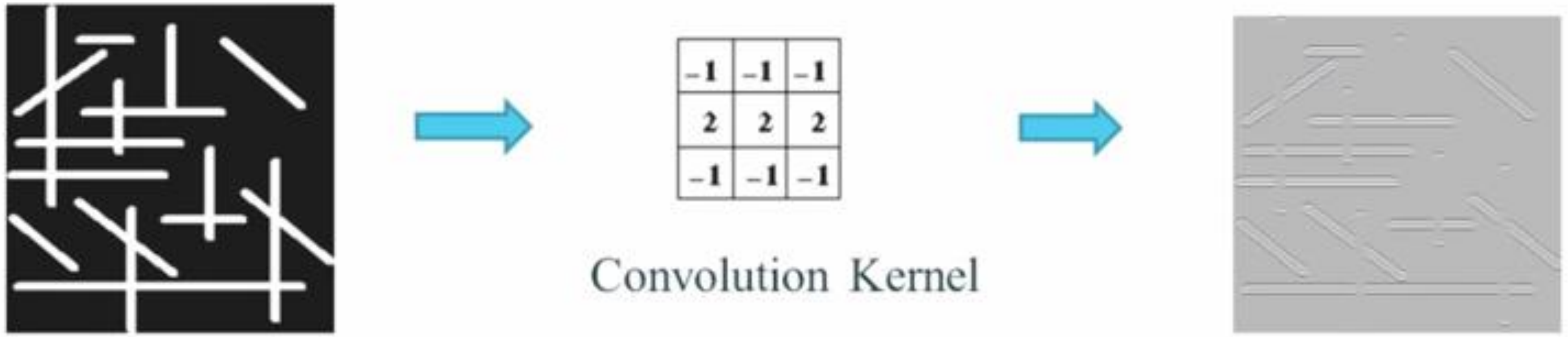
Test the classifier

Topics to be covered

- Haar Features
- Integral Images
- Adaboost
- Cascading

Horizontal Edge Detection

Basic introduction to edge detection



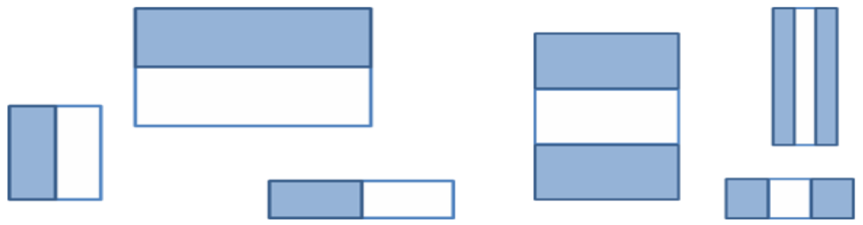
–1	–1	–1
2	2	2
–1	–1	–1

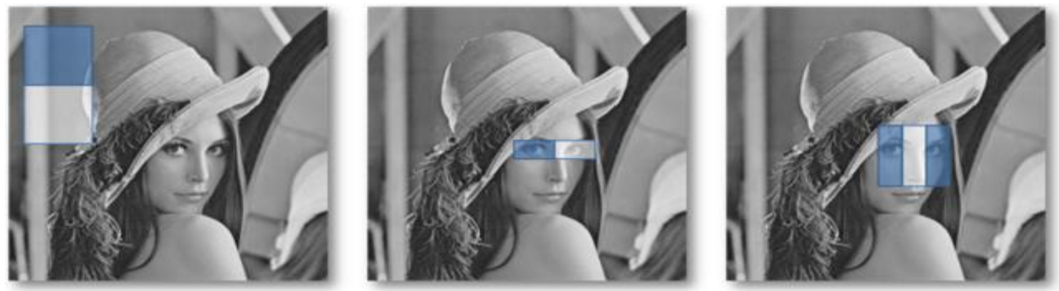
Convolution Kernel

Output image(right) has high intensity at pixels where the convolution kernel pixel pattern matches perfectly with the input image.

Haar Features

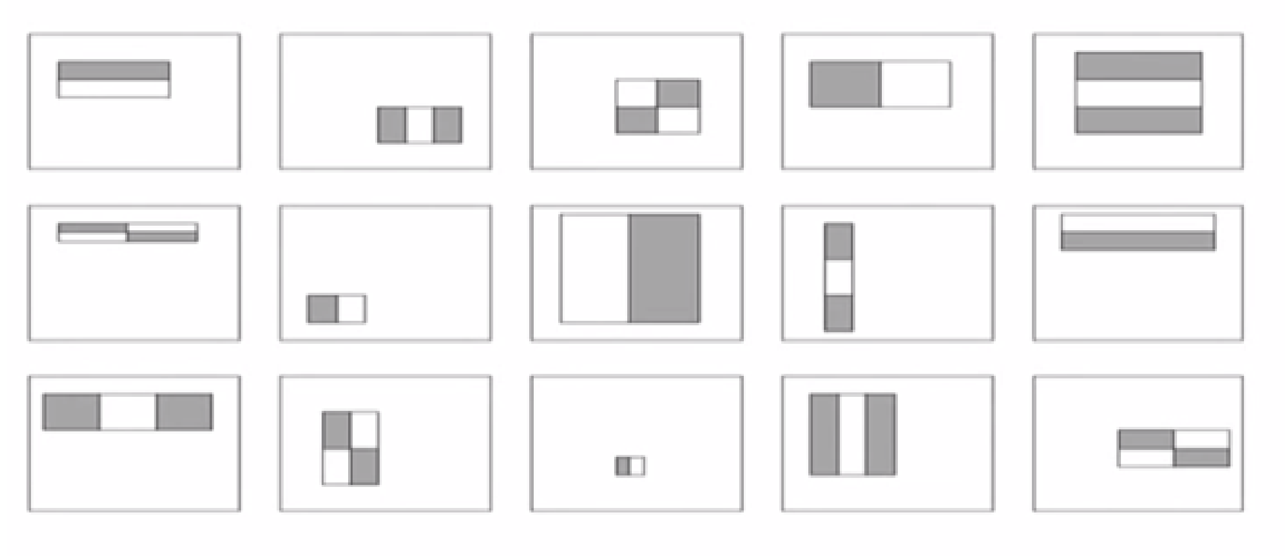
- Haar Features are to these convolution kernels which are used to detect presence of a feature in the given image.
- Each features results in a single value which is calculated by subtracting sum of the pixels under white rectangle from the sum of pixels under black rectangle.


Haar features used in Viola Johns


Applied on given image

Haar Features

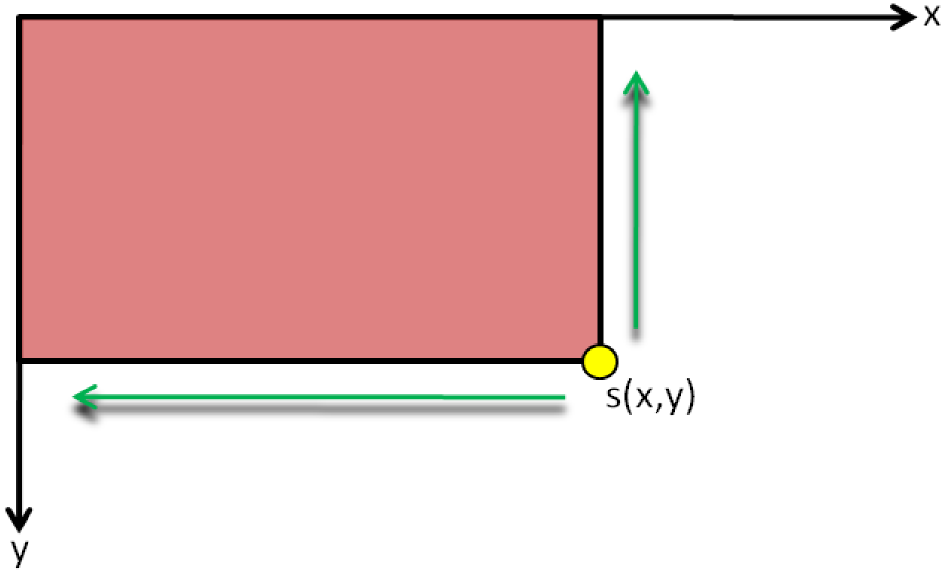
- Viola Johns algorithm uses 24x24 window as a base window size to start evaluating these features in a given image.
- Each If we consider all the possible parameter of Haar features like position, scale and type we end up calculating about 160,000+ in this window.

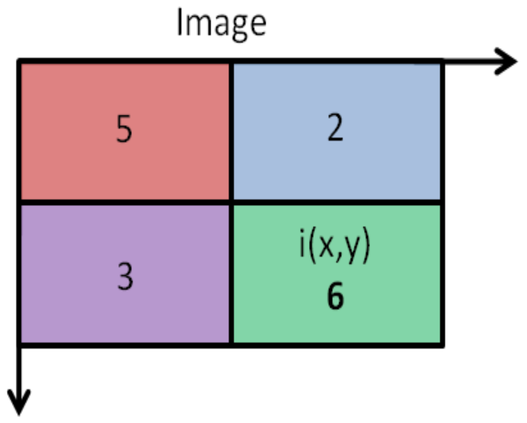


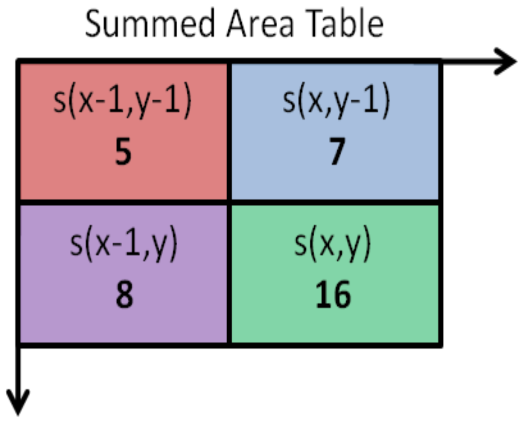
Integral Image

- In an Integral Image or Summed Area table the value at pixel (x,y) is the sum of pixels above and to the left of (x,y) including the original pixel value of (x,y) itself.

$$s(x,y) = i(x,y) + s(x-1,y) + s(x,y-1) - s(x-1,y-1)$$

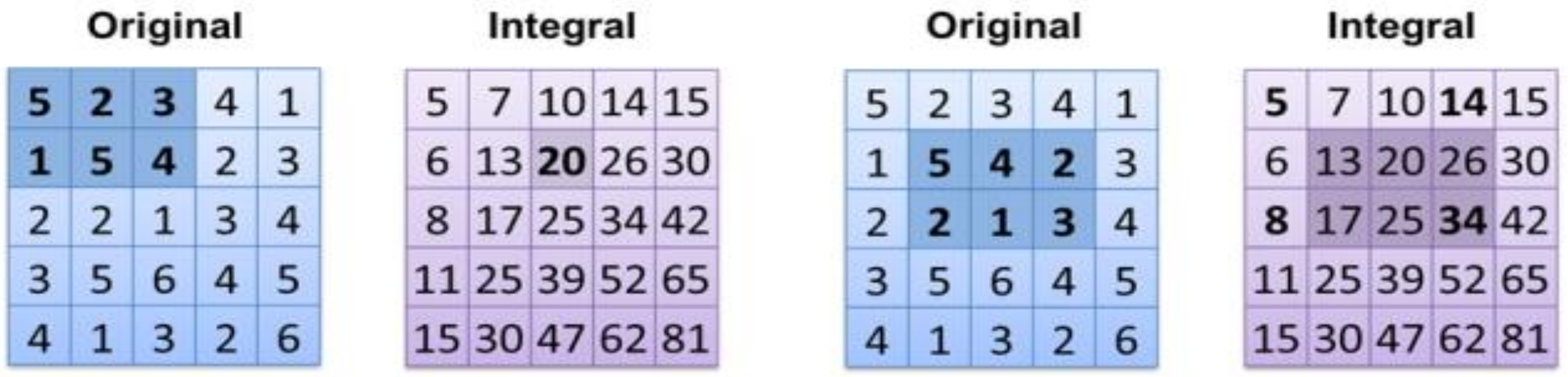






Integral Image

- Integral Image allows for the calculation of sum of all pixels inside any given rectangle using only four values at the corners of the rectangle.



Original

5	**2**	**3**	4	1
1	**5**	**4**	2	3
2	2	1	3	4
3	5	6	4	5
4	1	3	2	6

Integral

5	7	10	14	15
6	13	**20**	26	30
8	17	25	34	42
11	25	39	52	65
15	30	47	62	81

5 + 2 + 3 + 1 + 5 + 4 = 20

Original

5	2	3	4	1
1	**5**	**4**	**2**	3
2	**2**	**1**	**3**	4
3	5	6	4	5
4	1	3	2	6

Integral

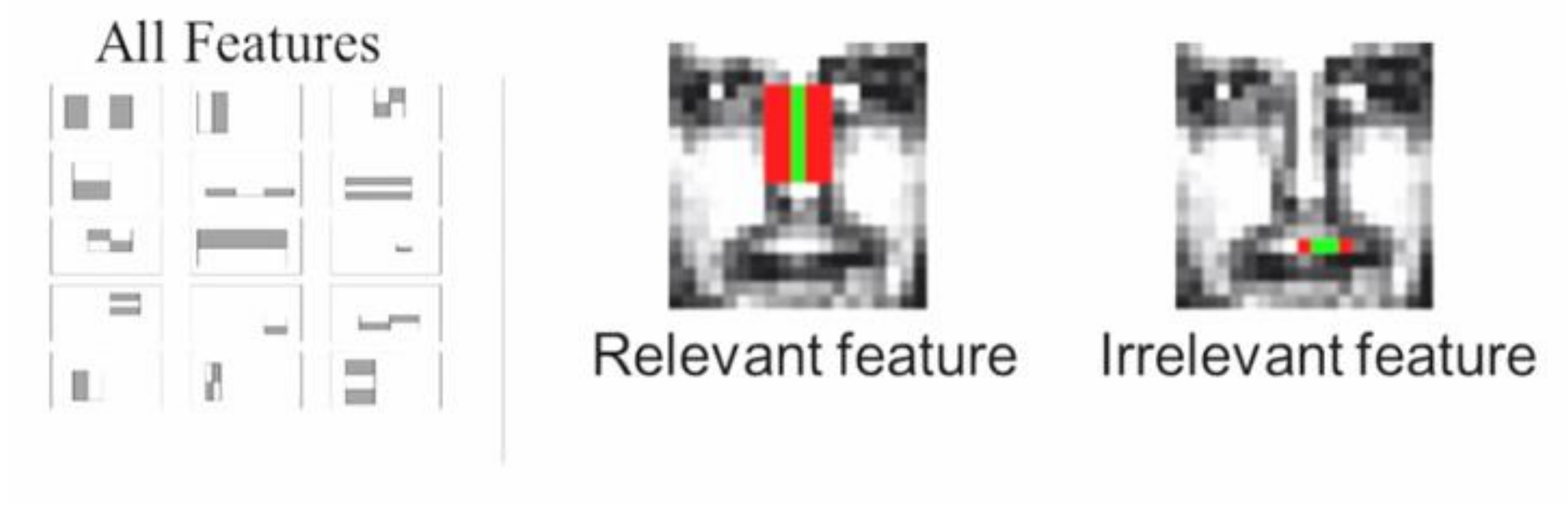
5	7	10	**14**	15
6	13	20	26	30
8	17	25	**34**	42
11	25	39	52	65
15	30	47	62	81

5 + 4 + 2 +
2 + 1 + 3 = 17

34 - 14 - 8 + 5 = 17

Adaboost

- As stated previously there are approximately 160,000+ feature value with in a decetor 24x24 base resolution which need to be calculated. But only few of the features are useful for face detection.



Adaboost

- Adaboost is a machine learing algorithm which helps in finding only the best features among all these 160,000+ features. After these features are found a weighted combination of all these features is used in evaluating and deciding any given face have a face or not. Each of the selected features are considered okay to be included if they can at least perform better than random guessing (detects more than half the cases).
- These features are also called weak classifier. Adaboost constructs a strong classifier as a linear combination of weak classifier.

$$\underset{\text{Strong classifier}}{\underset{\blacktriangle}{F(x)}} = \alpha_1 \underset{\text{Weak classifier}}{\underset{\blacktriangle}{f_1(x)}} + \alpha_2 f_2(x) + \alpha_3 f_3(x) + \ldots$$

ADABOOST

Stands for “**Ada**ptive” **boost**

Constructs a “strong” classifier as a linear combination of weighted simple “weak” classifiers

$$F(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \alpha_3 f_3(x) + \ldots$$

Weak classifier

Strong classifier

Image

Weight

ADABOOST - *CHARACTERISTICS*

Features as weak classifiers

- Each single rectangle feature may be regarded as a simple weak classifier

An iterative algorithm

- AdaBoost performs a series of trials, each time selecting a new weak classifier

Weights are being applied over the set of the example images

- During each iteration, each example/image receives a weight determining its importance

Adaboost Learning With Many Features

We have 160,000 features – how can we learn a classifier with only a few hundred training examples without over fitting?

Idea:

- Learn a single very simple classifier (a “weak classifier”)
- Classify the data
- Look at where it makes errors
- Reweight the data so that the inputs where we made errors get higher weight in the learning process
- Now learn a 2nd simple classifier on the weighted data
- Combine the 1st and 2nd classifier and weight the data according to where they make errors
- Learn a 3rd classifier on the weighted data
- … and so on until we learn T simple classifiers
- Final classifier is the combination of all T classifiers
- This procedure is called “Boosting” – works very well in practice.

Overview | Integral Image | AdaBoost | Cascade

- Given example images (x_1,y_1) , … , (x_n,y_n) where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = 1/(2m), 1/(2l)$ for training example i, where m and l are the number of negatives and positives respectively.

For $t = 1 \dots T$

1) Normalize weights so that w_t is a distribution
2) For each feature j train a classifier h_j and evaluate its error ε_j with respect to w_t.
3) Chose the classifier h_j with lowest error.
4) Update weights according to:

$$w_{t+1,i} = w_{t,i} \beta_t^{1-\varepsilon_i}$$

where $e_i = 0$ is x_i is classified correctly, 1 otherwise, and

$$\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$$

- The final strong classifier is:

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & otherwise \end{cases}, \qquad \text{where} \qquad \alpha_t = \log(\frac{1}{\beta_t})$$

- 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, $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$
  2. Select the best weak classifier with respect to the weighted error

     $$\epsilon_t = \min_{f,p,\theta} \sum_i w_i \,|\, h(x_i, f, p, \theta) - y_i \,| .$$

     See Section 3.1 for a discussion of an efficient implementation.
  3. Define $h_t(x) = h(x, f_t, p_t, \theta_t)$ where f_t, p_t, and θ_t are the minimizers of ϵ_t.
  4. Update the weights:

     $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$

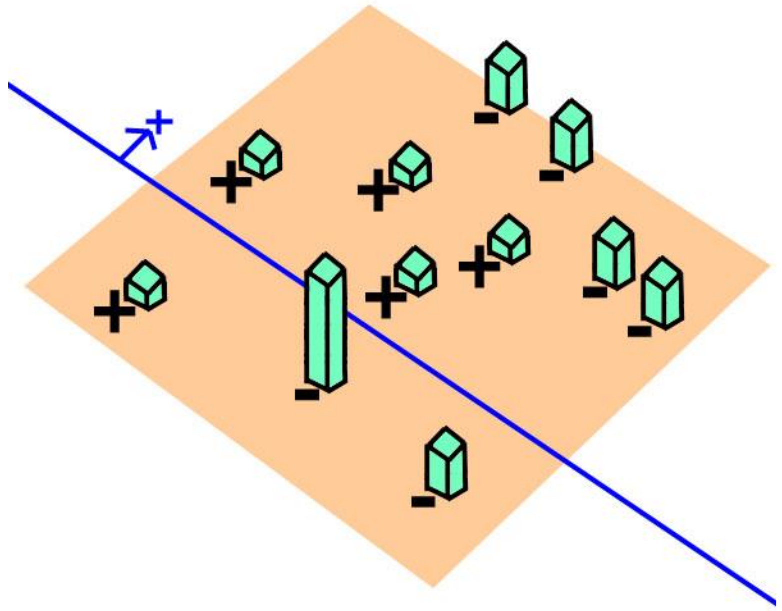
     where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- The final strong classifier is:

$$C(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha_t = \log \frac{1}{\beta_t}$

ADABOOST EXAMPLE

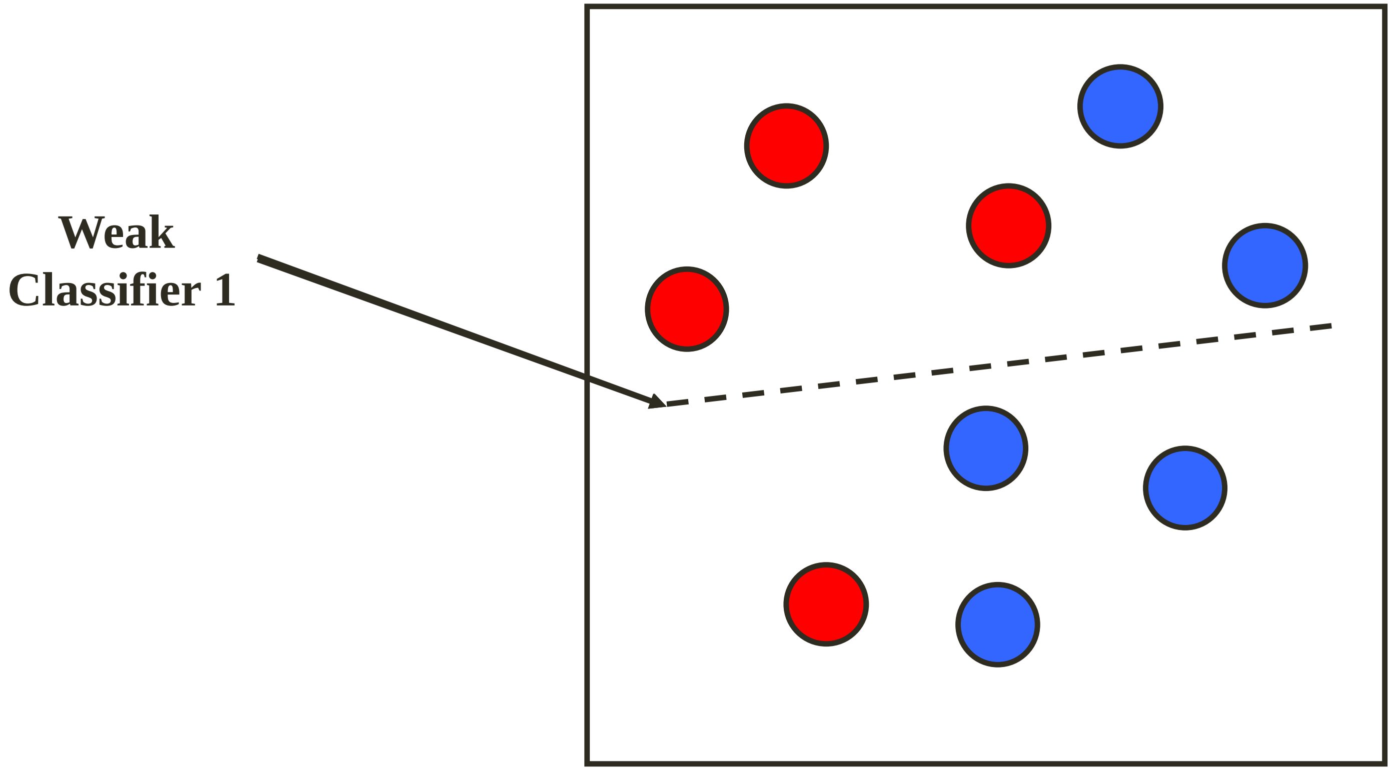
- AdaBoost starts with a uniform distribution of "weights" over training examples.
- Select the classifier with the lowest weighted error (i.e. a "weak" classifier)
- Increase the weights on the training examples that were misclassified.
- (Repeat)
- At the end, carefully make a linear combination of the weak classifiers obtained at all iterations.



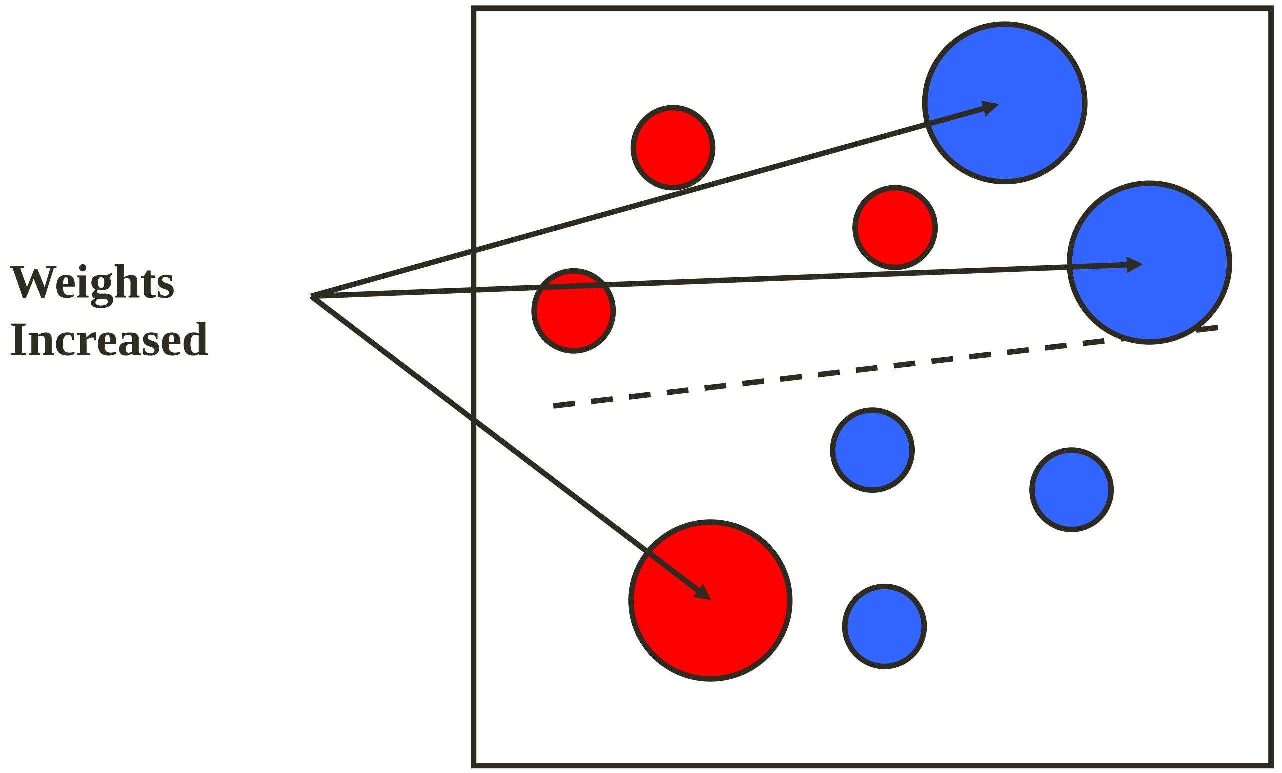
$$h_{\text{strong}}(\mathbf{x}) = \begin{cases} 1 & \alpha_1 h_1(\mathbf{x}) + \ldots + \alpha_n h_n(\mathbf{x}) \geq \frac{1}{2}(\alpha_1 + \ldots + \alpha_n) \\ 0 & \text{otherwise} \end{cases}$$

Slide taken from a presentation by Qing Chen, Discover Lab, University of Ottawa

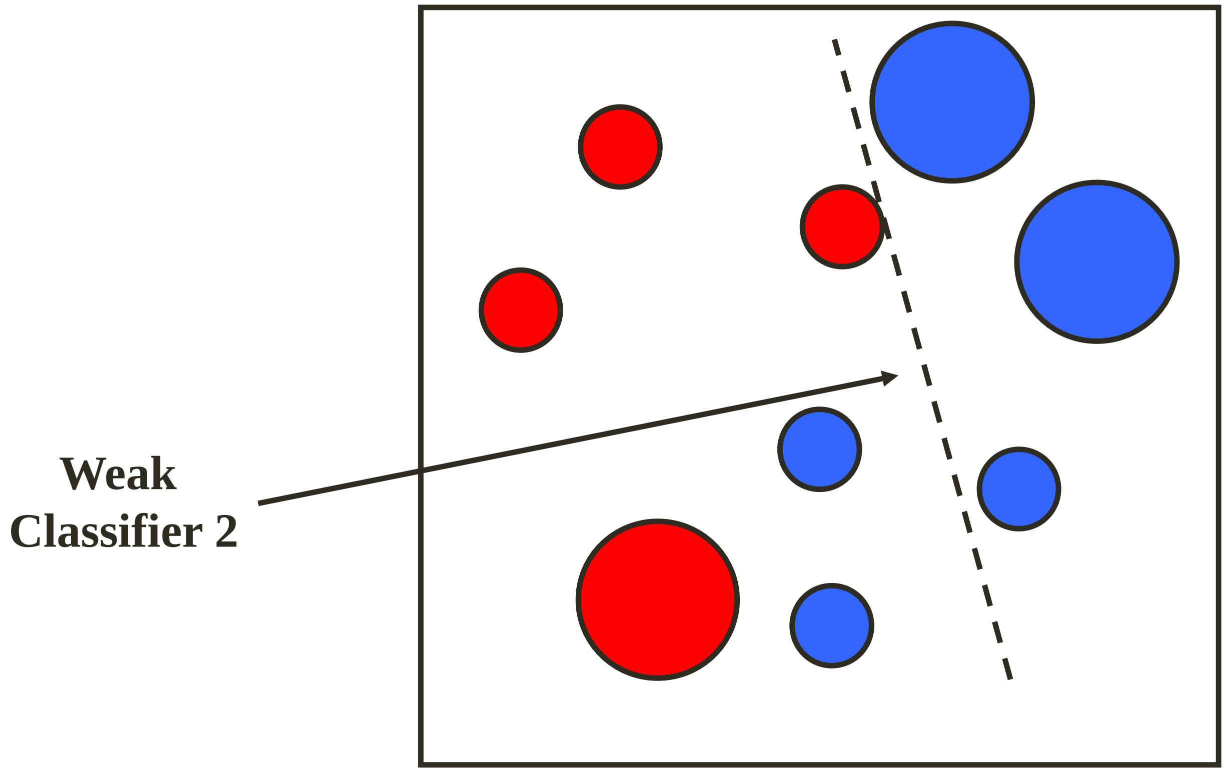
BOOSTING ILLUSTRATION



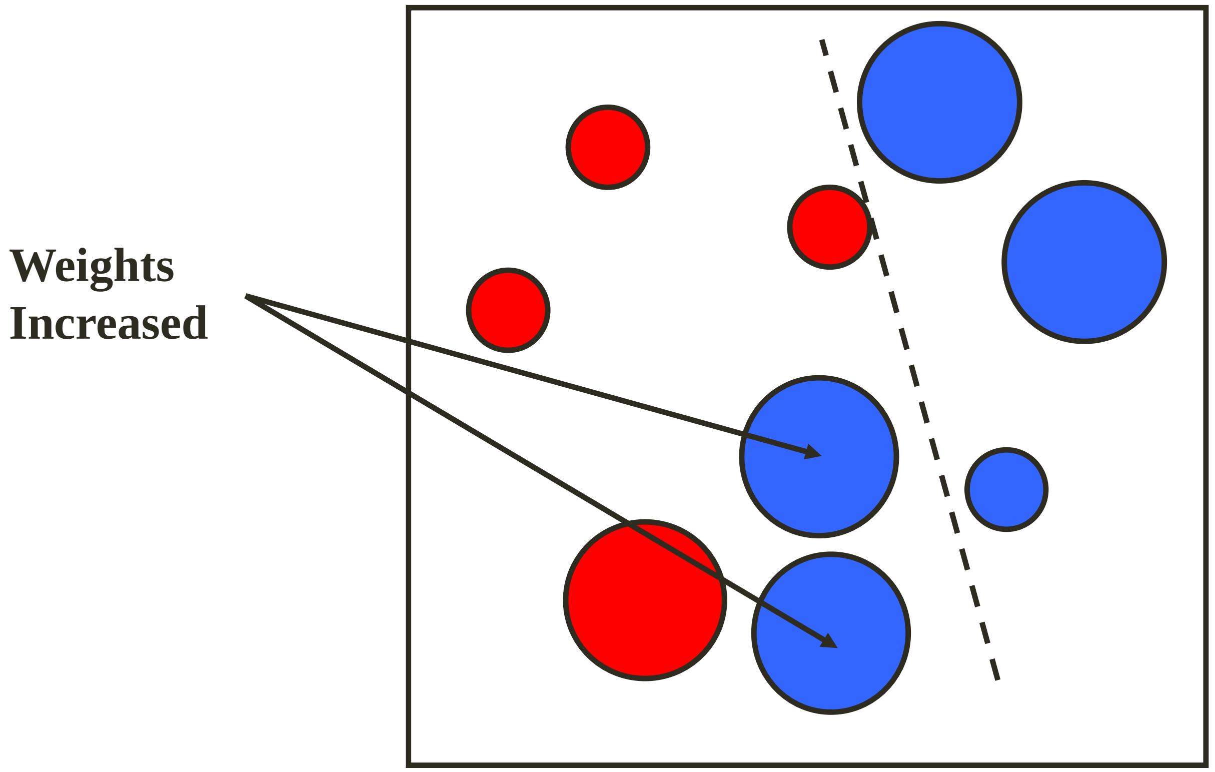
BOOSTING ILLUSTRATION



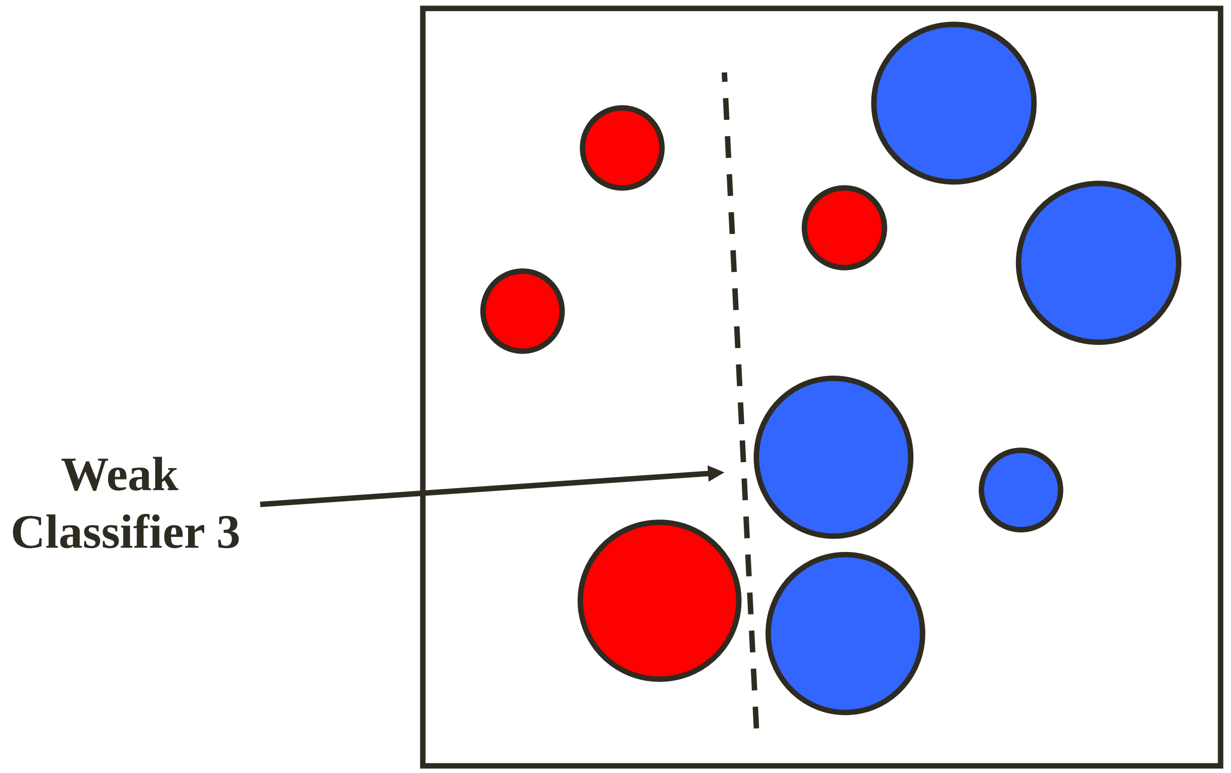
BOOSTING ILLUSTRATION



BOOSTING ILLUSTRATION

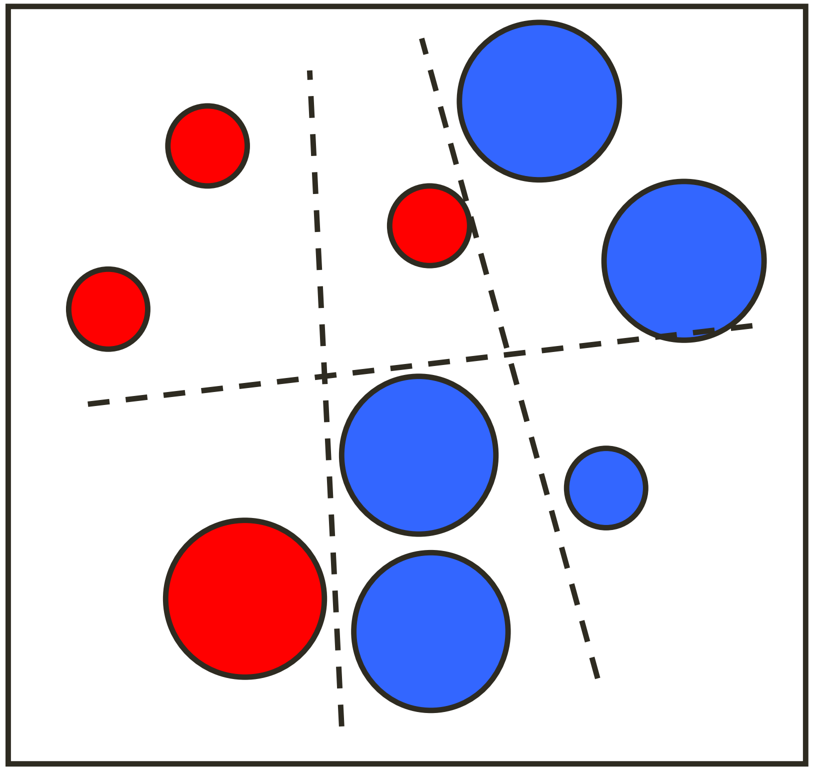


BOOSTING ILLUSTRATION



BOOSTING ILLUSTRATION

Final classifier is a combination of weak classifiers



NOW WE HAVE A GOOD FACE DETECTOR

We can build a 200-feature classifier!

Experiments showed that a 200-feature classifier achieves:

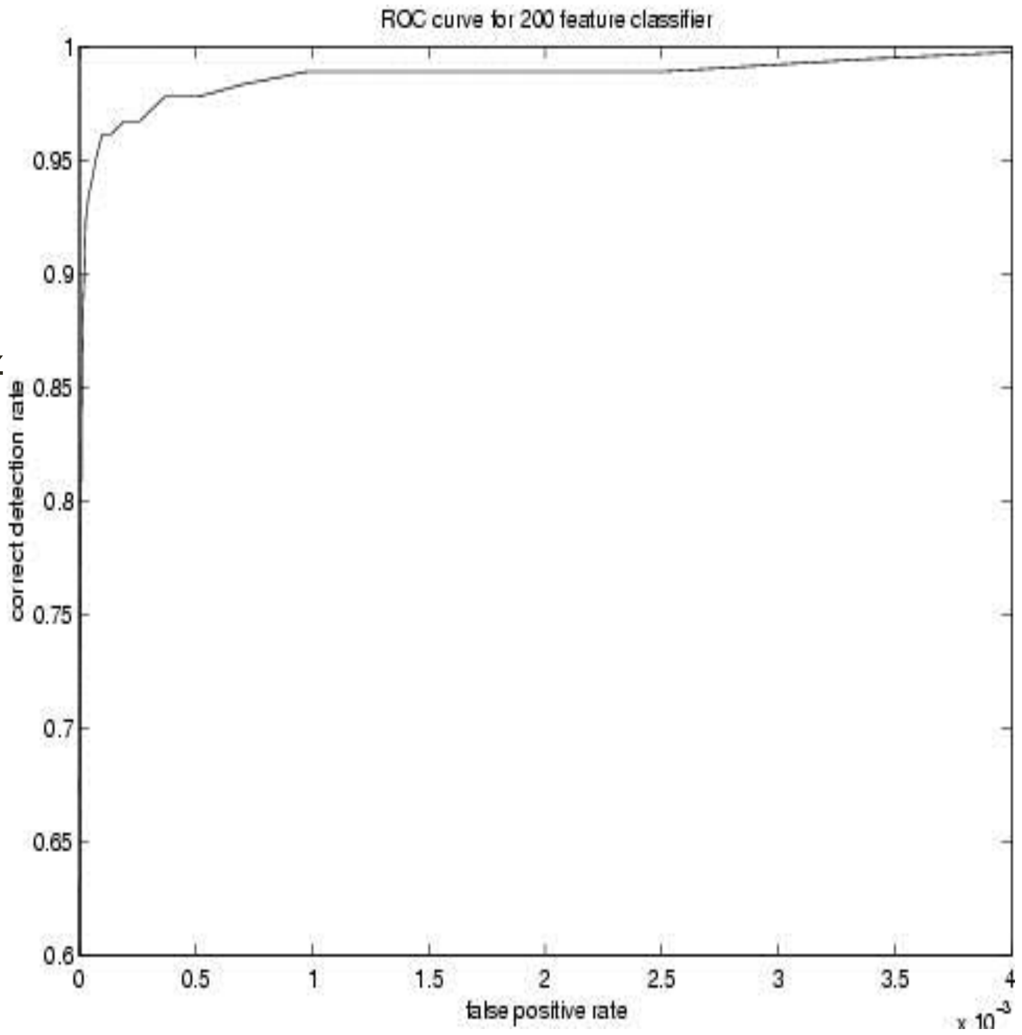
- 95% detection rate
- Scans all sub-windows of a 384x288 pixel image in 0.7 seconds (on Intel PIII 700MHz

The more the better (?)

- Gain in classifier performance
- Lose in CPU time

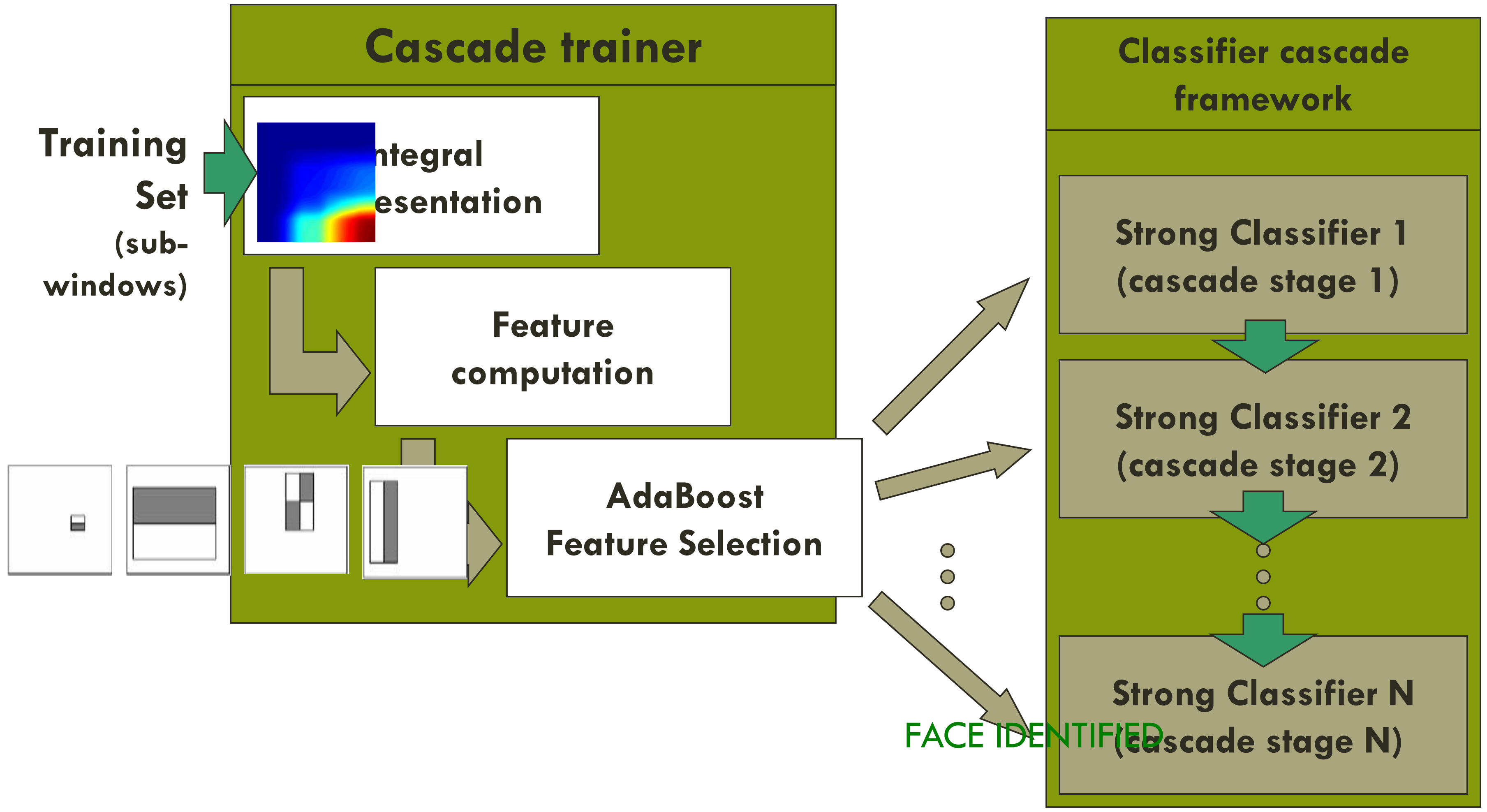
Verdict: good & fast, but not enough

- 0.7 sec / frame **IS NOT** real-time.



TRAINING PHASE

Testing phase

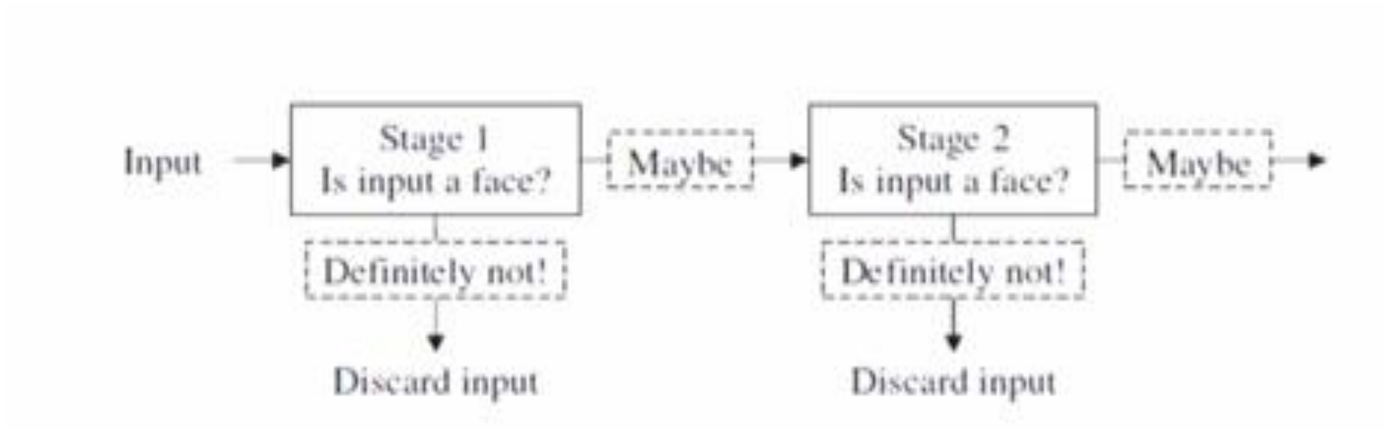

Cascade trainer
Training Set (sub-windows)
Integral Representation
Feature computation
AdaBoost Feature Selection
Classifier cascade framework
Strong Classifier 1 (cascade stage 1)
Strong Classifier 2 (cascade stage 2)
Strong Classifier N (cascade stage N)
FACE IDENTIFIED

Cascading

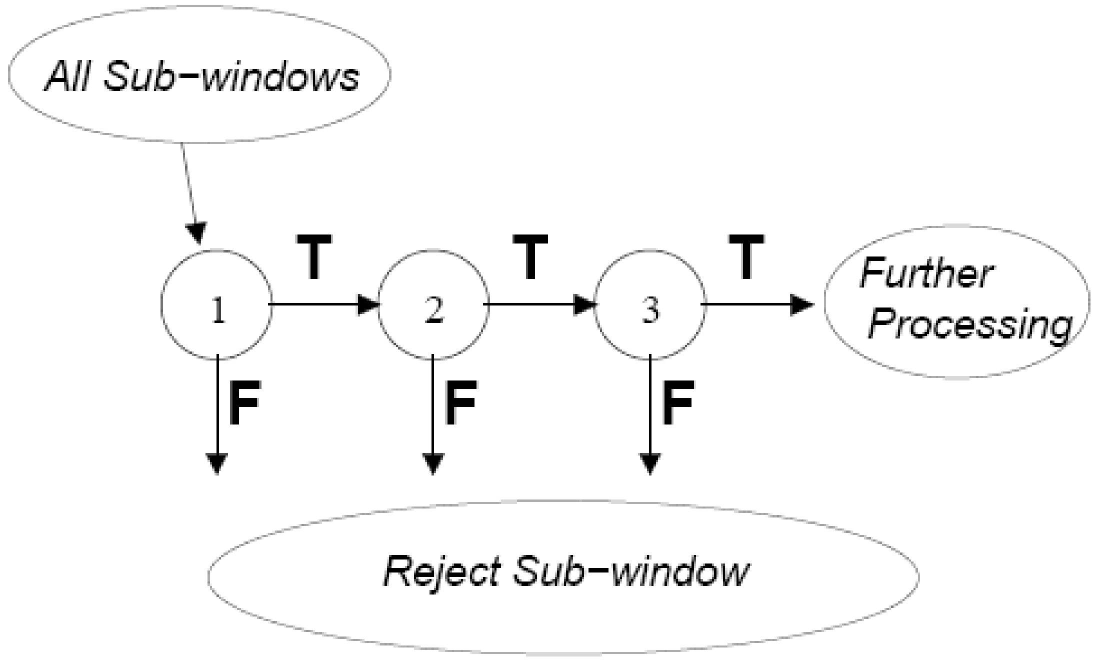
- The basic principle of the Viola-Jones face detection algorithm is to scan the detector many times through the same image – each time with a new size.
- Even if an image should contain one or more face it is obvious that an excessive large amount of the evaluated sub-windows would still be negatives (non-faces).
- So the algorithm should concentrate on discarding non-faces quickly and spend more on time on probable face regions.
- Hence a single strong classifier formed out the linear combination of all best features is not a good to evaluate on each window because of computation cost.
- Therefor a cascade classifier is used which is composed of stages each containing a strong classifier. So all features are grouped into several stages where each stage has certain number of features.

Cascading

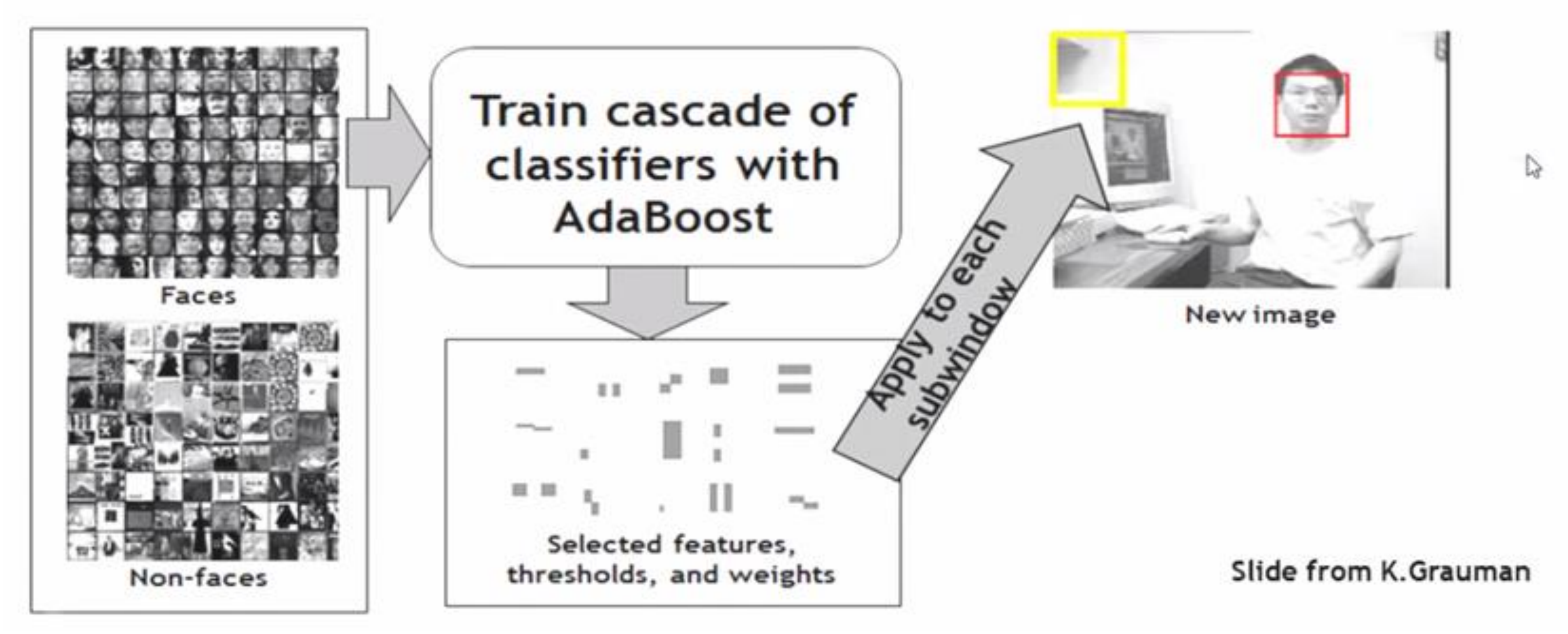
- The Job of each stage is used to determine whether a given sub window is definitely not a face or may be a face. A given window is immediately discarded as not a face if it fails in any of the stage.



A Cascade of Classifiers



Summary


Faces
Non-faces
Train cascade of classifiers with AdaBoost
Selected features, thresholds, and weights
Apply to each subwindow
New image
Slide from K.Grauman

PROS …

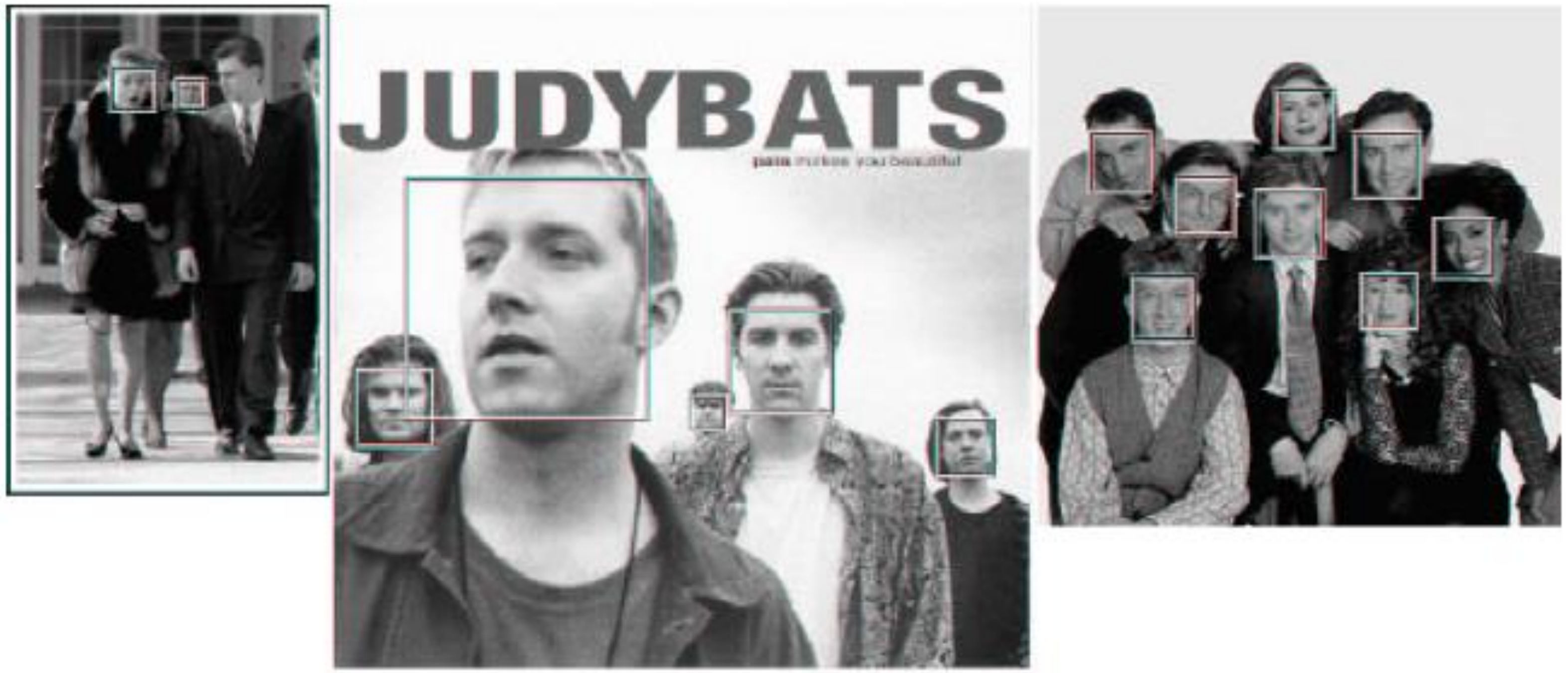
- Extremely fast feature computation
- Efficient feature selection
- Scale and location invariant detector
  - Instead of scaling the image itself (e.g. pyramid-filters), we scale the features.
- Such a generic detection scheme can be trained for detection of other types of objects (e.g. cars, hands)

… and cons

- Detector is most effective only on frontal images of faces
  - can hardly cope with 45^o face rotation
- Sensitive to lighting conditions

RESULTS

(detailed results at back-up slide #4)



RESULTS (CONT.)