Computer Vision
CSC9010/5930

Instructor: Dr. Edward Kim

Histograms of Oriented Gradients for Human Detection (HOG)

N. Dalal and B. Triggs

CVPR 2005

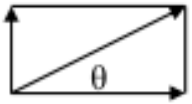
Computing Gradients

- Centered: $f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h}$
- Filter masks in x and y directions
  - Centered:

-1	0	1

-1
0
1

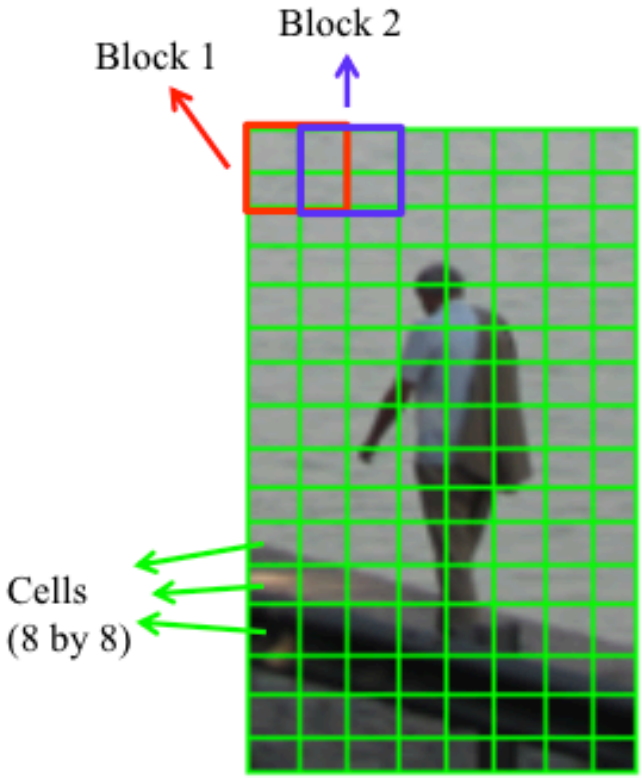
- Gradient
  - Magnitude: $s = \sqrt{s_x^2 + s_y^2}$



  - Orientation: $\theta = \arctan(\frac{s_y}{s_x})$

Blocks, Cells

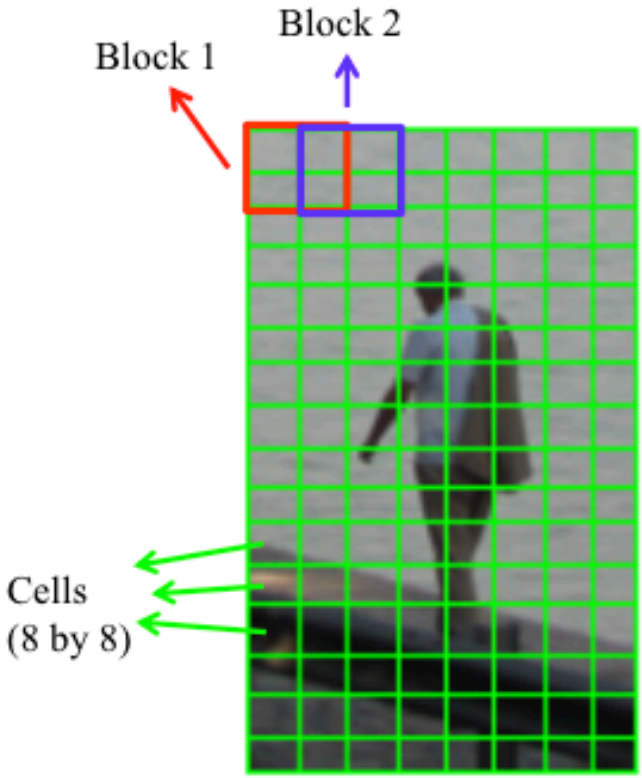
- 16x16 blocks of 50% overlap.
  - 7x15=105 blocks in total
- Each block should consist of 2x2 cells with size 8x8.



- HOG feature extraction
  - Compute centered horizontal and vertical gradients with no smoothing
  - Compute gradient orientation and magnitudes
    - For color image, pick the color channel with the highest gradient magnitude for each pixel.
  - For a 64x128 image,
  - Divide the image into 16x16 blocks of 50% overlap.
    - 7x15=105 blocks in total
  - Each block should consist of 2x2 cells with size 8x8.
  - Quantize the gradient orientation into 9 bins
    - The vote is the gradient magnitude
    - Interpolate votes between neighboring bin center.
    - The vote can also be weighted with Gaussian to downweight the pixels near the edges of the block.

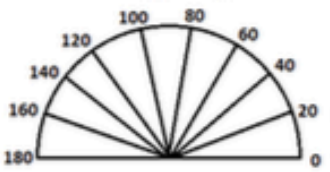
Blocks, Cells

- 16x16 blocks of 50% overlap.
  - 7x15=105 blocks in total
- Each block should consist of 2x2 cells with size 8x8.

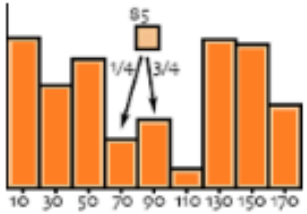


Votes

- Each block consists of 2x2 cells with size 8x8
- Quantize the gradient orientation into 9 bins (0-180)



9 Bins



Bin centers

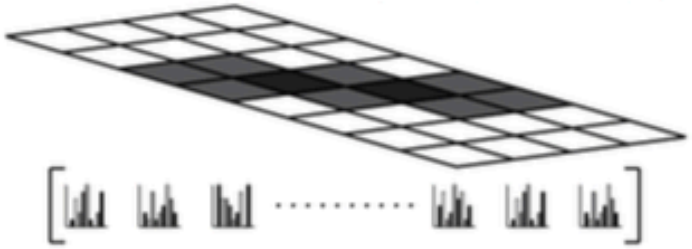
  - The vote is the gradient magnitude

  - Interpolate votes linearly between neighboring bin centers.
    - Example: if θ=85 degrees.
    - Distance to the bin center Bin 70 and Bin 90 are 15 and 5 degrees, respectively.
    - Hence, ratios are 5/20=1/4, 15/20=3/4.

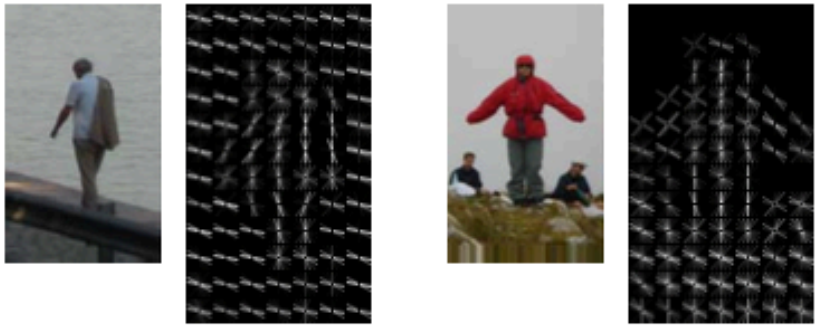
  - The vote can also be weighted with Gaussian to down weight the pixels near the edges of the block.

Final Feature Vector

- Concatenate histograms
  - Make it a 1D vector of length 3780.



- Visualization



Results

Navneet Dalal and Bill Triggs "Histograms of Oriented Gradients for Human Detection" CVPR05

SIFT Vs HOG

SIFT

- 128 dimensional vector
- 16 by 16 window
- 4x4 sub-window (16 total)
- 8 bin histogram

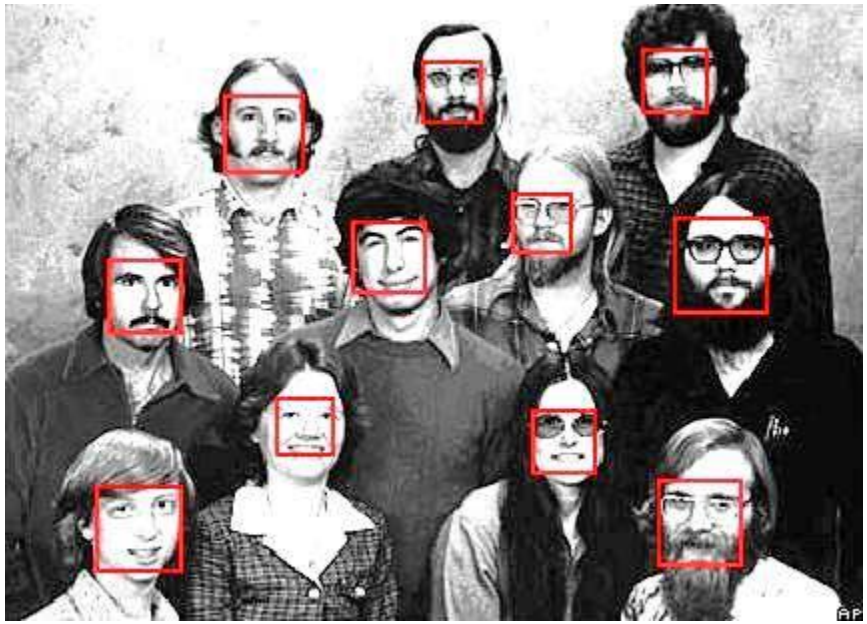
HOG

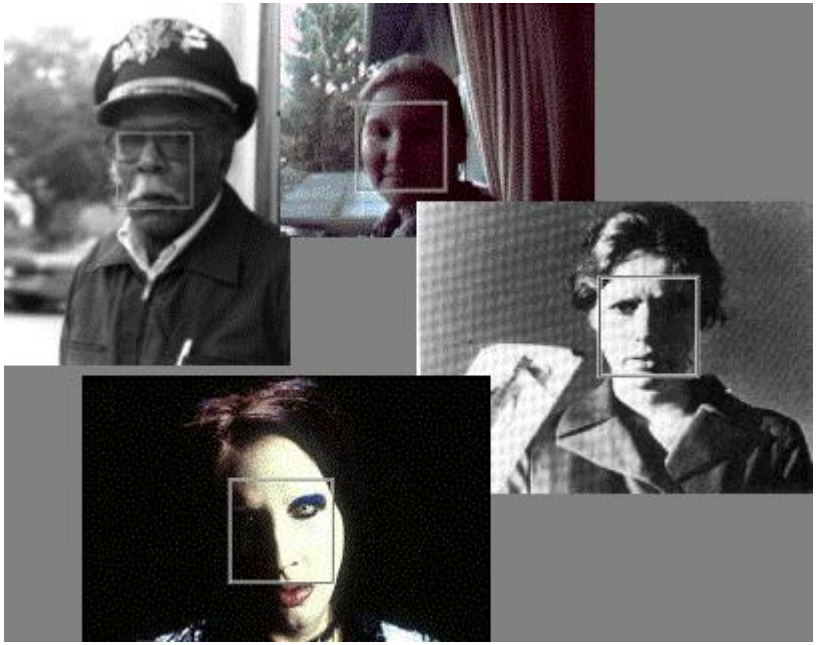
- 3,780 dimensional vector
- 64 by 128 window
- 16 by 16 blocks with overlap
- Each block consists of 2 by 2 cells each of 8 by 8
- Overlapping
- 9 bin histogram

Adaboost for Face Detection

Slides adapted from P. Viola and Tai-Wan Yue

The Task of Face Detection





Basic Idea

Slide a window across image and evaluate a face model at every location; no. of locations where face is present is very very small

Challenges

- Slide a window across image and evaluate a face model at every location
- Sliding window detector must evaluate tens of thousands of location/scale combinations
- Faces are rare: 0–10 per image
  - For computational efficiency, we should try to spend as little time as possible on the non-face windows
  - A megapixel image has $\sim 10^6$ pixels and a comparable number of candidate face locations
  - To avoid having a false positive in every image, the false positive rate has to be less than 10^{-6}

The Viola/Jones Face Detector

- A seminal approach to real-time object detection
- Key ideas
  - *Integral images* for fast feature evaluation
  - *Boosting* for feature selection
  - *Attentional cascade* for fast rejection of non-face windows

P. Viola and M. Jones. *Rapid object detection using a boosted cascade of simple features.* CVPR 2001.

P. Viola and M. Jones. *Robust real-time face detection.* IJCV 57(2), 2004.

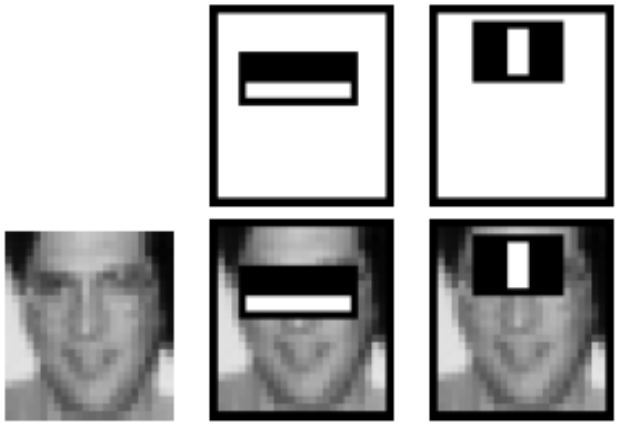
Features

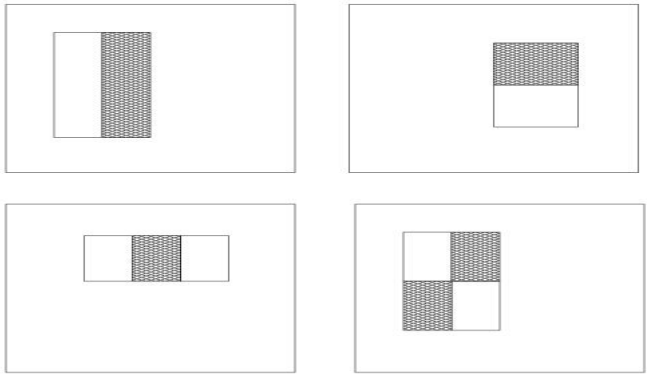
- Can a simple feature (i.e. a value) indicate the existence of a face?
- All faces share some similar properties
  - The eyes region is darker than the upper-cheeks.
  - The nose bridge region is brighter than the eyes.
  - **That is useful domain knowledge**

- Need for encoding of Domain Knowledge:
  - ***Location - Size:*** eyes & nose bridge region
  - ***Value:*** darker / brighter

Rectangle features

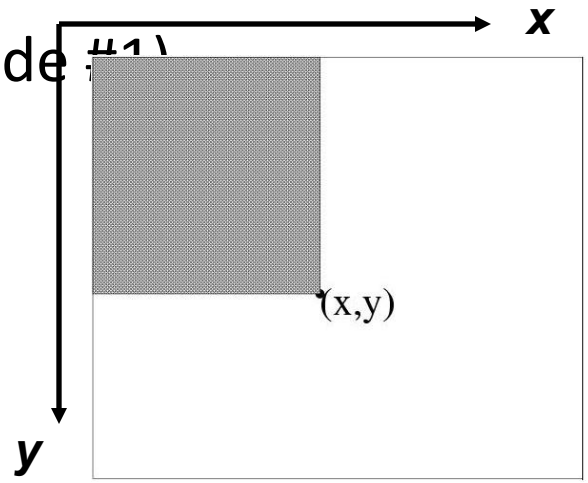
- Rectangle features:
  - Value = ∑ (pixels in black area) - ∑ (pixels in white area)
  - Three types: two-, three-, four-rectangles, Viola&Jones used two-rectangle features
  - For example: the difference in brightness between the white &black rectangles over a specific area
- Each feature is related to a special location ir the sub-window
- Each feature may have any size
- Why not pixels instead of features?
  - Features encode domain knowledge
  - Feature based systems operate faster





Integral Image Representation

(also check back-up slide #1)



- Given a detection resolution of 24x24 (smallest sub-window), the set of different rectangle features is ~160,000 !
- Need for speed
- Introducing Integral Image Representation
  - Definition: The integral image at location (x,y), is the sum of the pixels above and to the left of (x,y), inclusive
- The Integral image can be computed in a single pass and only once for each sub-window!

formal definition:

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x',y')$$

Recursive definition:

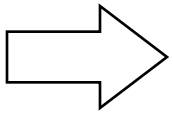
$$s(x,y) = s(x,y-1) + i(x,y)$$

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

back-up slide #1

IMAGE

0	1	1	1
1	2	2	3
1	2	1	1
1	3	1	0

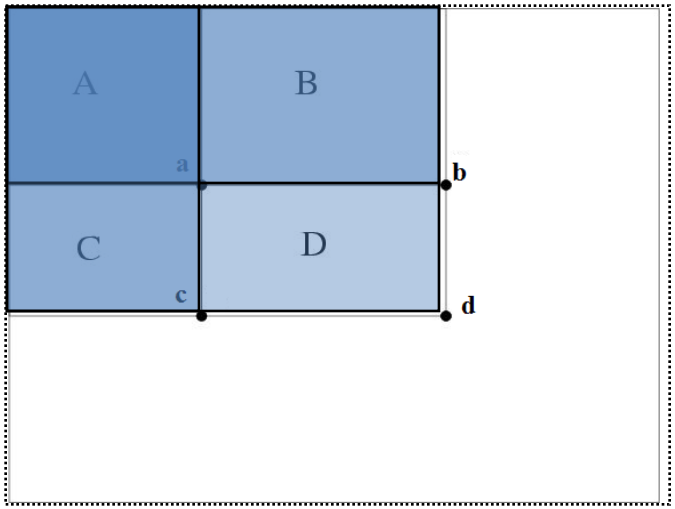


INTEGRAL IMAGE

0	1	2	3
1	4	7	11
2	7	11	16
3	11	16	21

Rapid computation of rectangular features

- Back to feature evaluation . . .
- Using the integral image representation we can compute the value of any rectangular sum (part of features) in **constant time**
  - For example the integral sum inside rectangle D can be computed as:
    $ii(d) + ii(a) - ii(b) - ii(c)$
- two-, three-, and four-rectangular features can be computed with 6, 8 and 9 **array references** respectively.
- As a result: feature computation takes less time



ii(a) = A

ii(b) = A+B

ii(c) = A+C

ii(d) = A+B+C+D

D = ii(d)+ii(a)-
ii(b)-ii(c)

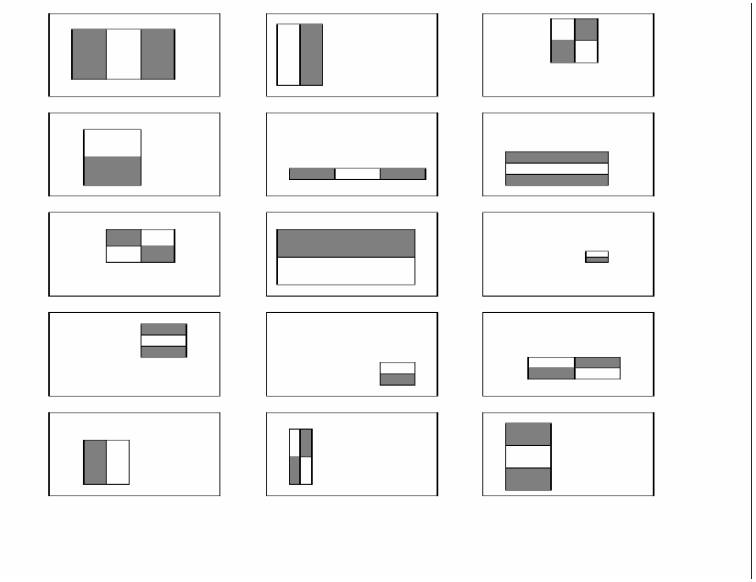
Three goals

1. *Feature Computation*: features must be computed as quickly as possible
2. *Feature Selection*: select the most discriminating features
3. *Real-timeliness*: must focus on potentially positive image areas (that contain faces)

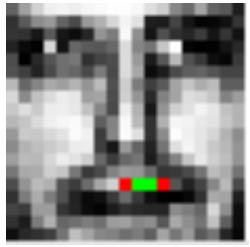
How did Viola & Jones deal with these challenges?

Feature selection

- Problem: Too many features
  - In a sub-window (24x24) there are ~160,000 features (all possible combinations of orientation, location and scale of these feature types)
  - impractical to compute all of them (computationally expensive)
- We have to select a subset of relevant features – which are informative - to model a face
  - **Hypothesis**: "A very small subset of features can be combined to form an effective classifier"
  - How?
    - AdaBoost algorithm



Relevant feature



Irrelevant feature

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

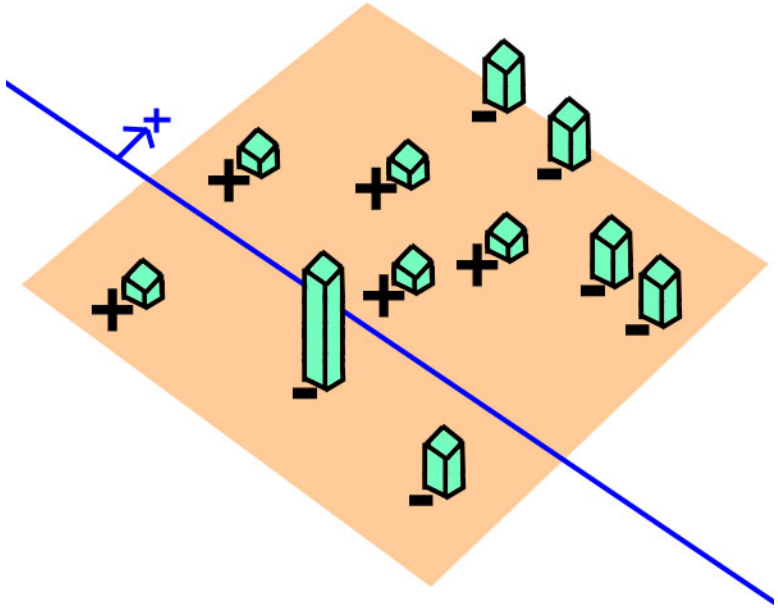
Strong classifier

Image

Weight

AdaBoost 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.



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

AdaBoost - *Getting the idea…*

(pseudo-code at back-up slide #2)

- Given: example images labeled +/-
  - Initially, all weights set equally
- Repeat T times
  - Step 1: choose the most efficient weak classifier that will be a component of the final strong classifier (Problem! Remember the huge number of features…)
  - Step 2: Update the weights to emphasize the examples which were incorrectly classified
    - This makes the next weak classifier to focus on “harder” examples
- Final (strong) classifier is a weighted combination of the T “weak” classifiers
  - Weighted according to their accuracy

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

AdaBoost – *Feature Selection*

Problem

- On each round, large set of possible weak classifiers (each simple classifier consists of a single **feature**) – Which one to choose?
  - choose the most efficient (the one that best separates the examples – the lowest error)
  - choice of a classifier corresponds to choice of a feature
- At the end, the 'strong' classifier consists of T features

Conclusion

- AdaBoost searches for a small number of good classifiers – features (feature selection)
- adaptively constructs a final strong classifier taking into account the failures of each one of the chosen weak classifiers (weight appliance)
- AdaBoost is used to **both** select a small set of features and train a strong classifier

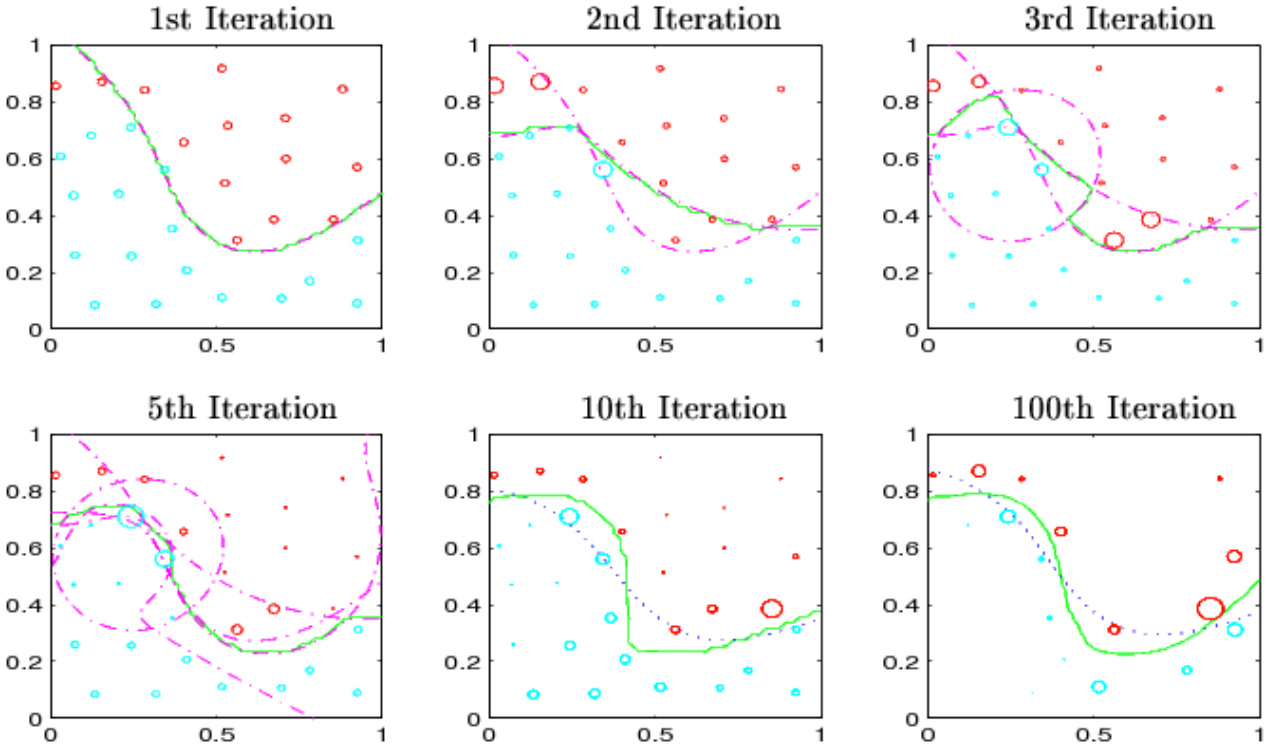
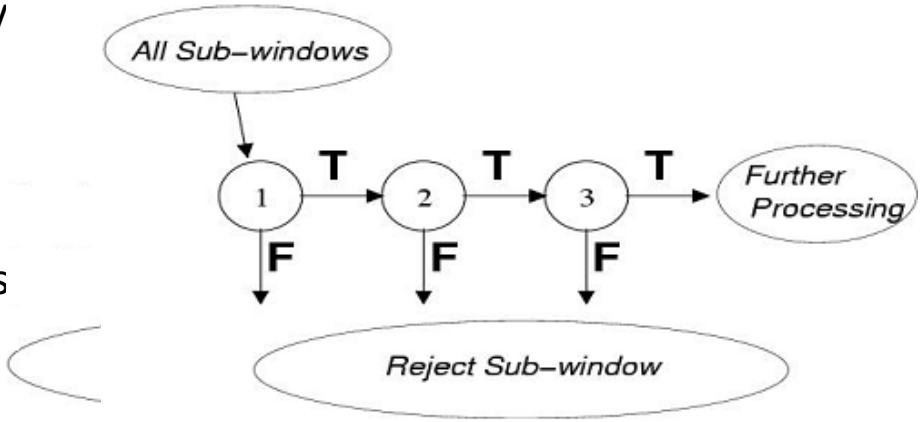


Fig. 1. Illustration of AdaBoost on a 2D toy data set: The color indicates the label and the diameter is proportional to the weight of the examples in the first, second, third, 5th, 10th and 100th iteration. The dashed lines show the decision boundaries of the single classifiers (up to the 5th iteration). The solid line shows the decision line of the combined classifier. In the last two plots the decision line of Bagging is plotted for a comparison. (Figure taken from [153].)

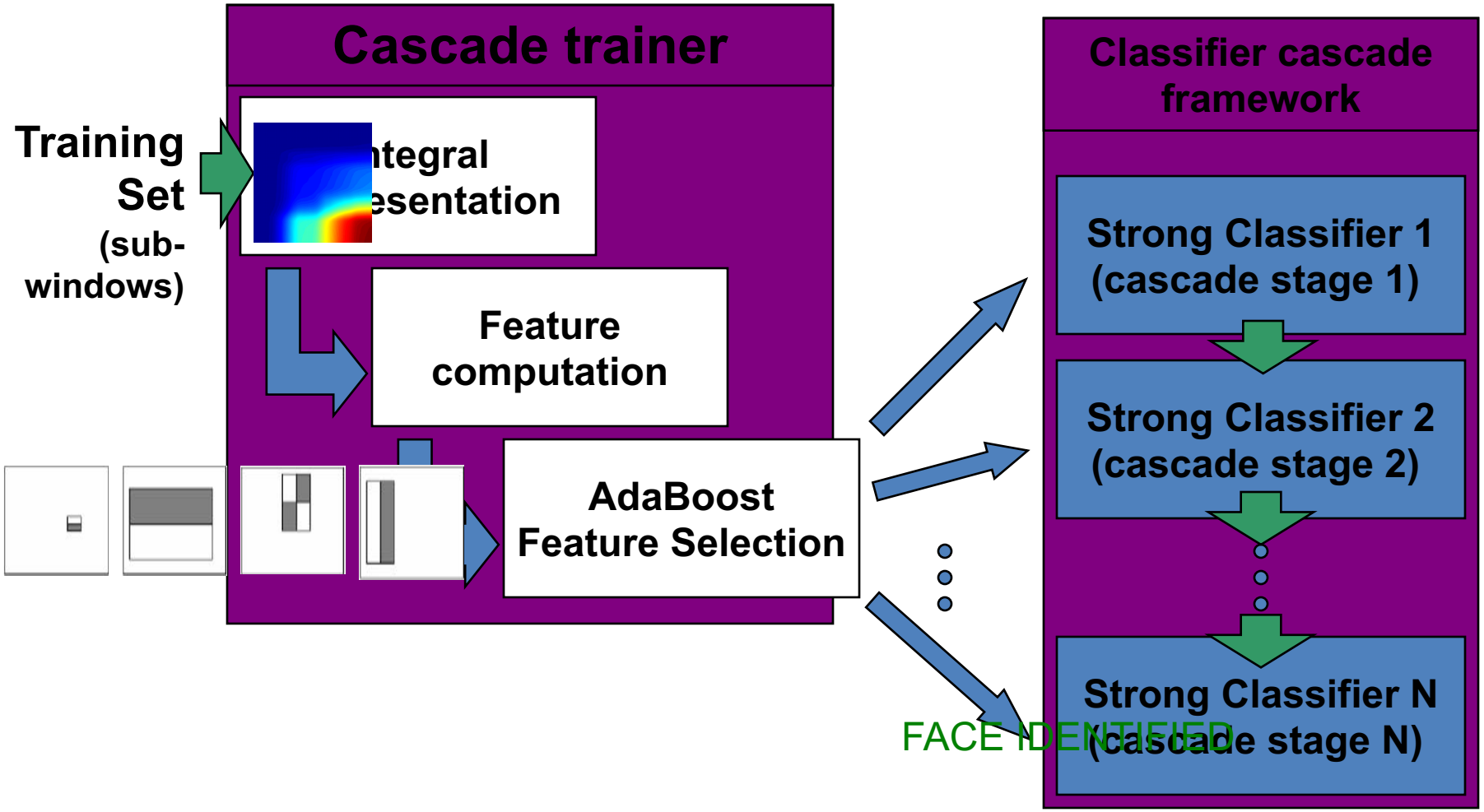
The attentional cascade

- On average only 0.01% of all sub-windows are positive (are faces)
- Status Quo: equal computation time is spent on all sub-windows
- Must spend most time only on potentially positive sub-windows.
- A simple 2-feature classifier can achieve almost 100% detection rate with 50% FP rate.
- That classifier can act as a 1st layer of a series to filter out most negative windows
- 2nd layer with 10 features can tackle "harder" negative-windows which survived the 1st layer, and so on…
- A **cascade** of gradually more complex classifiers achieves even better detection rates.



On average, much fewer features are computed per sub-window (i.e. speed x 10)

Testing phase Training phase



Speed of the Final Detector

- On a 700 Mhz Pentium III processor, the face detector can process a 384 ×288 pixel image in about .067 seconds"
  - 15 times faster than previous detector of comparable accuracy (Rowley et al., 1998)

- Average of 8 features evaluated per window on test set

Output of Face Detector on Test Images