Advanced Computer Vision

Detection with Sliding Windows- Part2

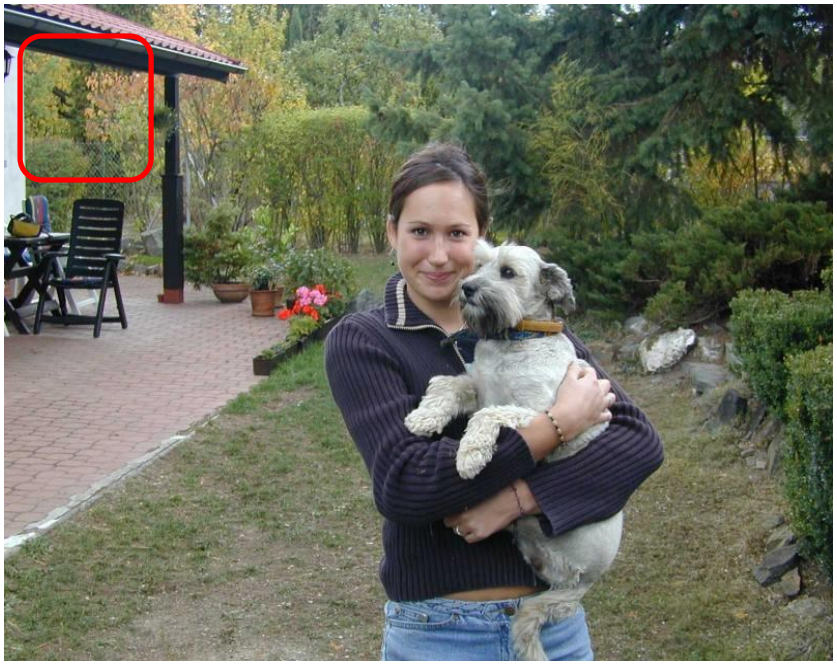
FACE DETECTION

Consumer application: Apple iPhoto

Things iPhoto thinks are faces

"The Nikon S60 detects up to 12 faces."

Face detection and recognition



Detection

Recognition

"Sally"

Challenges of face detection

Sliding window = tens of thousands of location/scale evaluations

- One megapixel image has ~10^6 pixels, and a comparable number of candidate face locations

Faces are rare: 0–10 per image

- For computational efficiency, spend as little time as possible on the non-face windows.

- For 1 Mpix, to avoid having a false positive in every image, our false positive rate has to be less than 10^{-6}

Sliding Window Face Detection with Viola-Jones

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.

Many Slides from Lana Lazebnik

The Viola/Jones Face Detector

A seminal approach to real-time object detection.

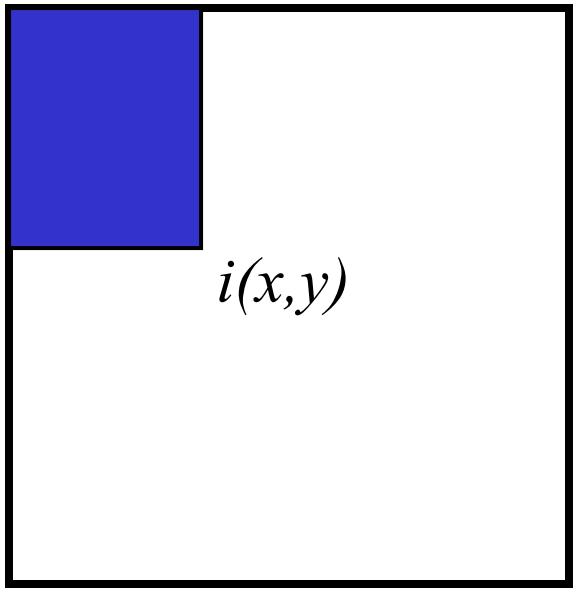
Training is slow, but detection is very fast

Key ideas:

1. *Integral images* for fast feature evaluation
2. *Boosting* for feature selection
3. *Attentional cascade* for fast non-face window rejection

1. Integral images for fast feature evaluation

- The *integral image* computes a value at each pixel (*x*,*y*) that is the sum of *all* pixel values above and to the left of (*x*,*y*), inclusive.

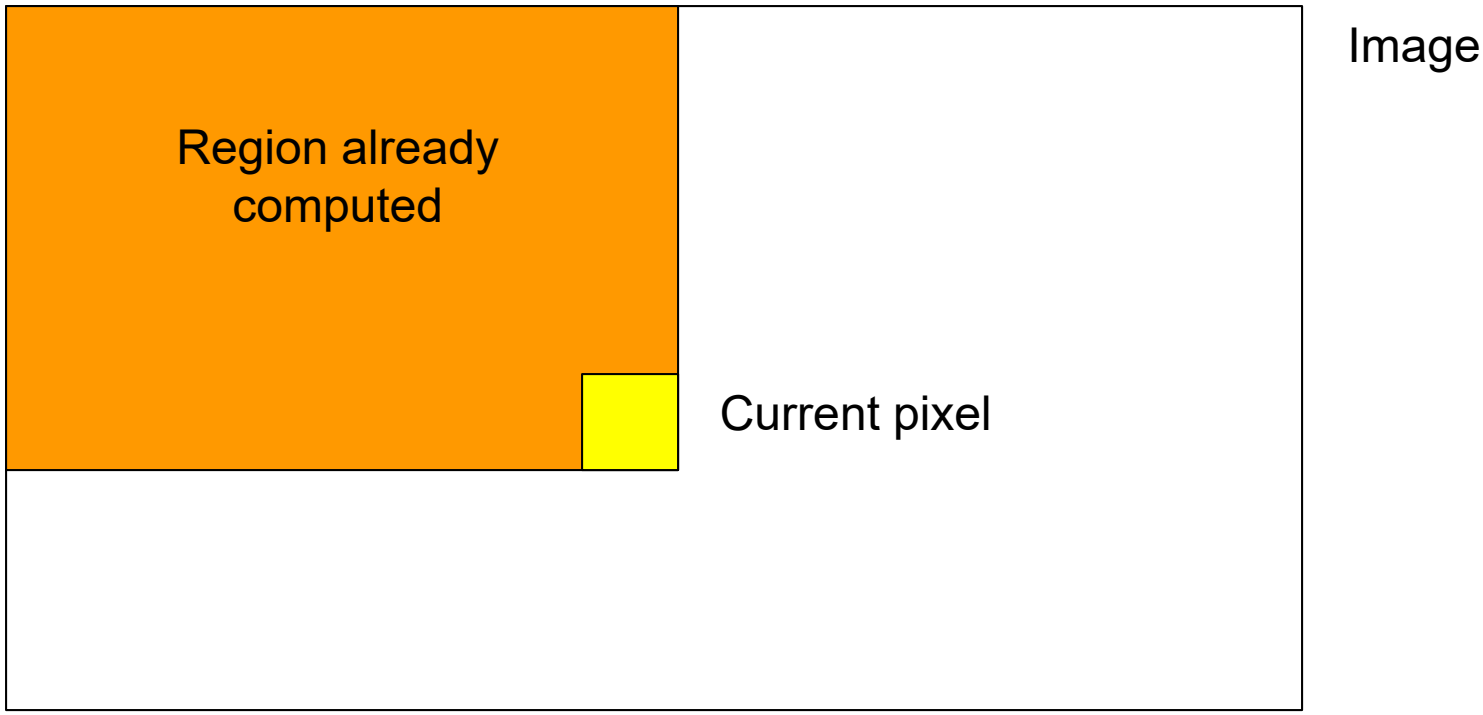


- This can quickly be computed in one pass through the image.

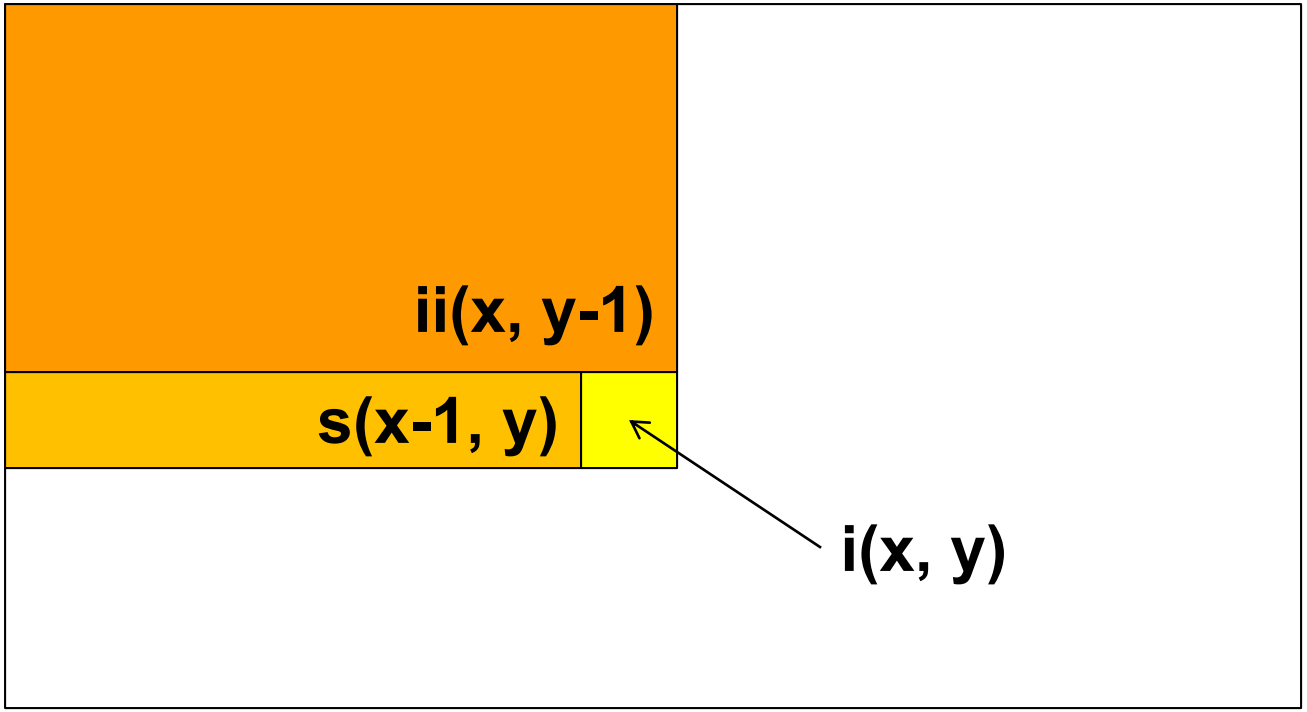
$$I_{\sum}(x,y) = \sum_{\substack{x' \le x \\ y' \le y}} i(x',y')$$

- 'Summed area table'

Computing the integral image



Computing the integral image



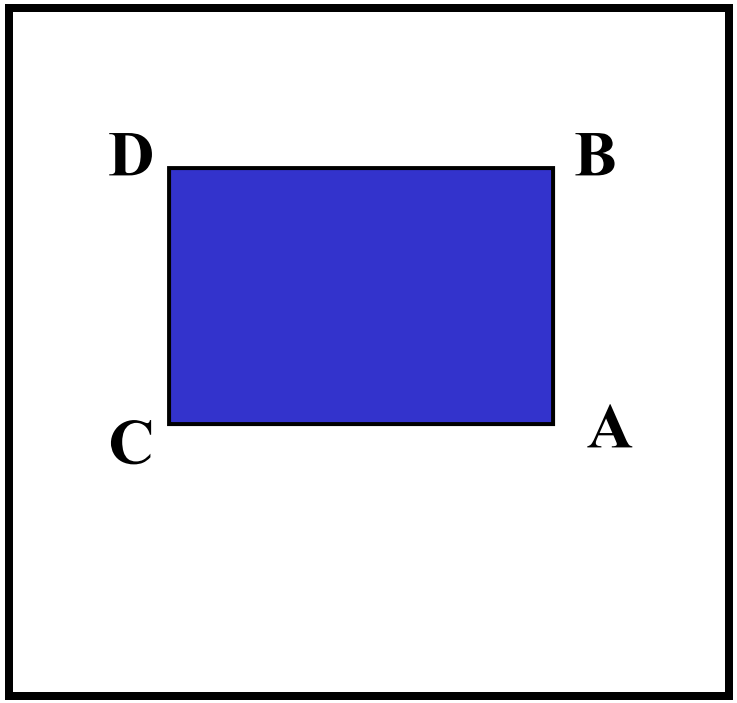
Cumulative row sum: $s(x, y) = s(x–1, y) + i(x, y)$

Integral image: $ii(x, y) = ii(x, y−1) + s(x, y)$

MATLAB: ii = cumsum(cumsum(double(i)), 2);

Computing sum within a rectangle

- Let A,B,C,D be the values of the integral image at the corners of a rectangle



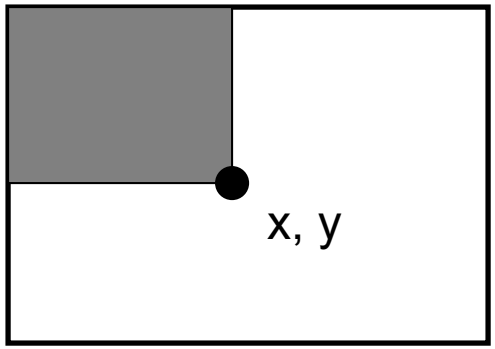
- The sum of original image values within the rectangle can be computed as:

  sum = A – B – C + D

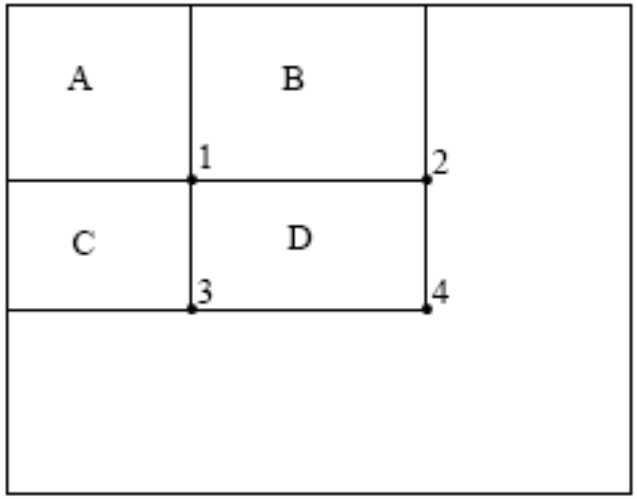
Only 3 additions are required for any size of rectangle!

Integral Images

- `ii = cumsum(cumsum(im, 1), 2)`



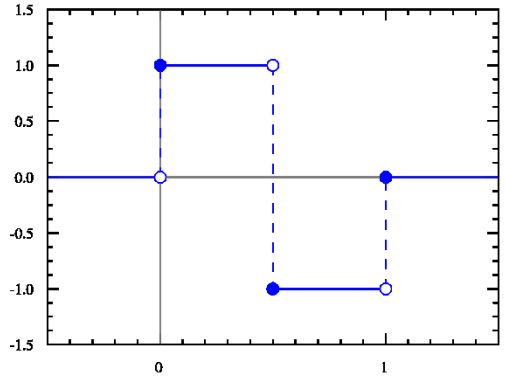
ii(x,y) = Sum of the values in the grey region



SUM within Rectangle D is
ii(4) - ii(2) - ii(3) + ii(1)

Features that are fast to compute

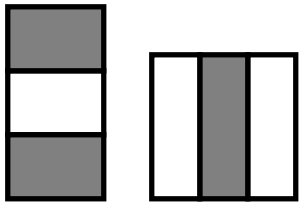
Haar wavelet



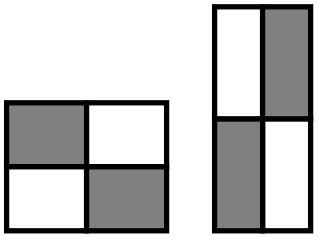
"Haar-like features"

- Differences of sums of intensity
- Computed at different positions and scales within sliding window

Two-rectangle features



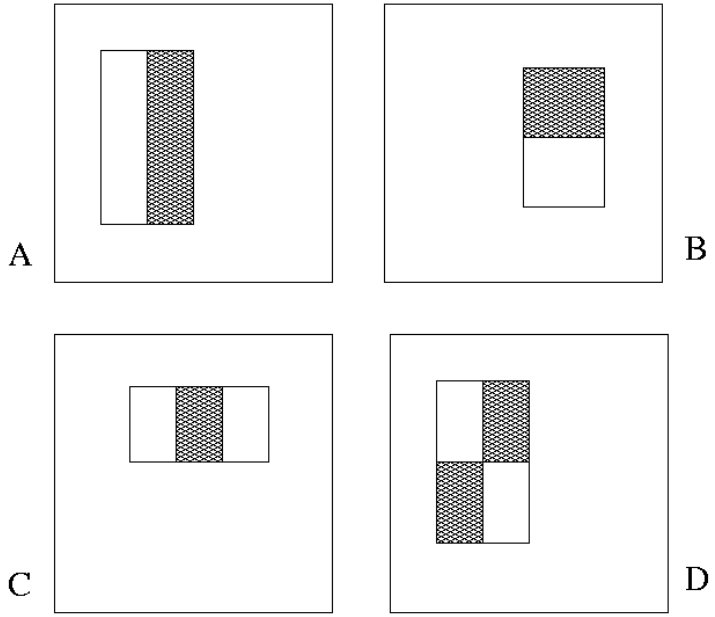
Three-rectangle features



Etc.

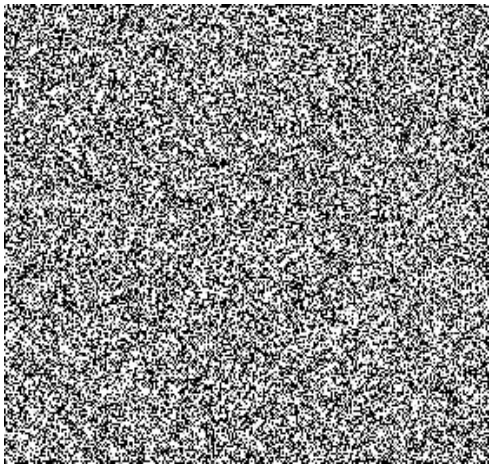
Image Features

"Rectangle filters"

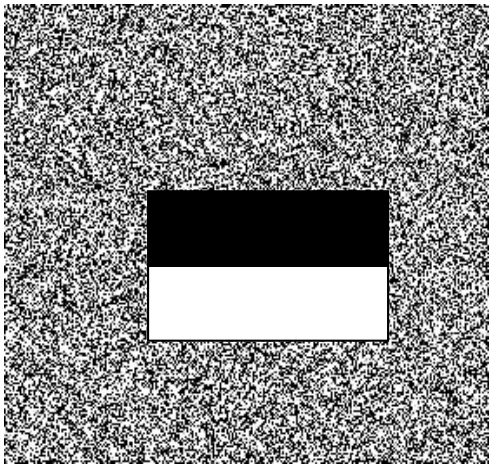


$$Value = \sum(\text{pixels in white area}) - \sum(\text{pixels in black area})$$

Example

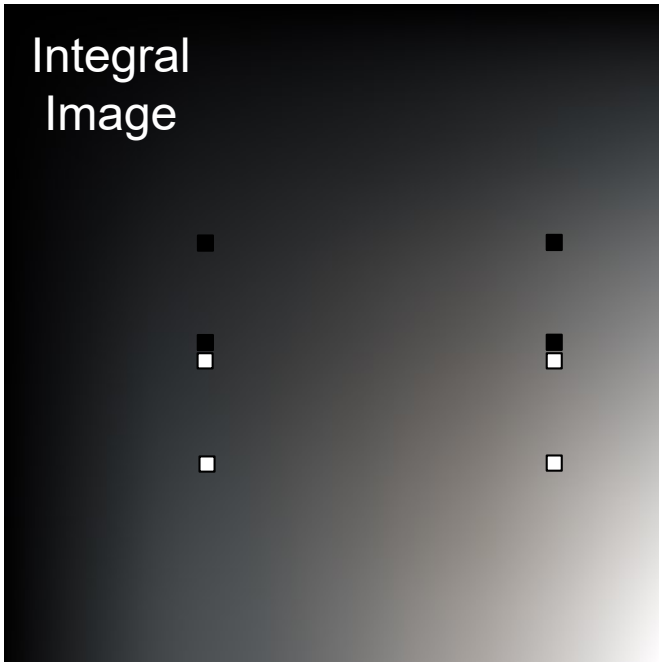


Source



Result

Computing a rectangle feature

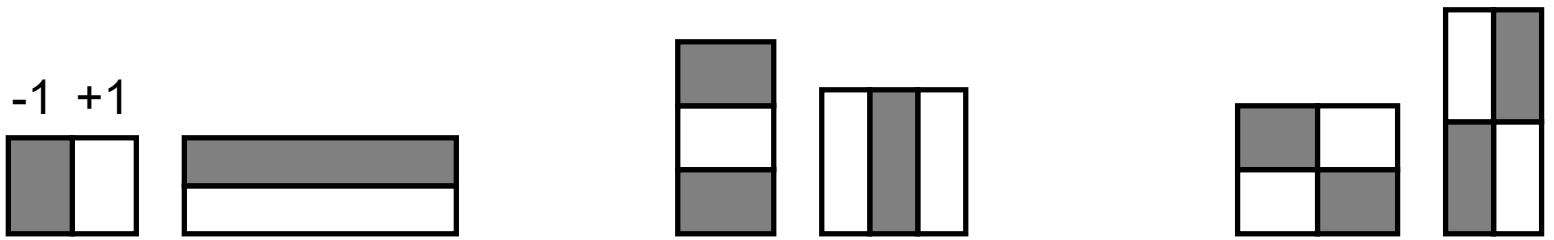


But these features are rubbish…!

Yes, individually they are ‘weak classifiers’

Jargon: ‘feature’ and ‘classifier’ are used interchangeably here. Also with ‘learner’.

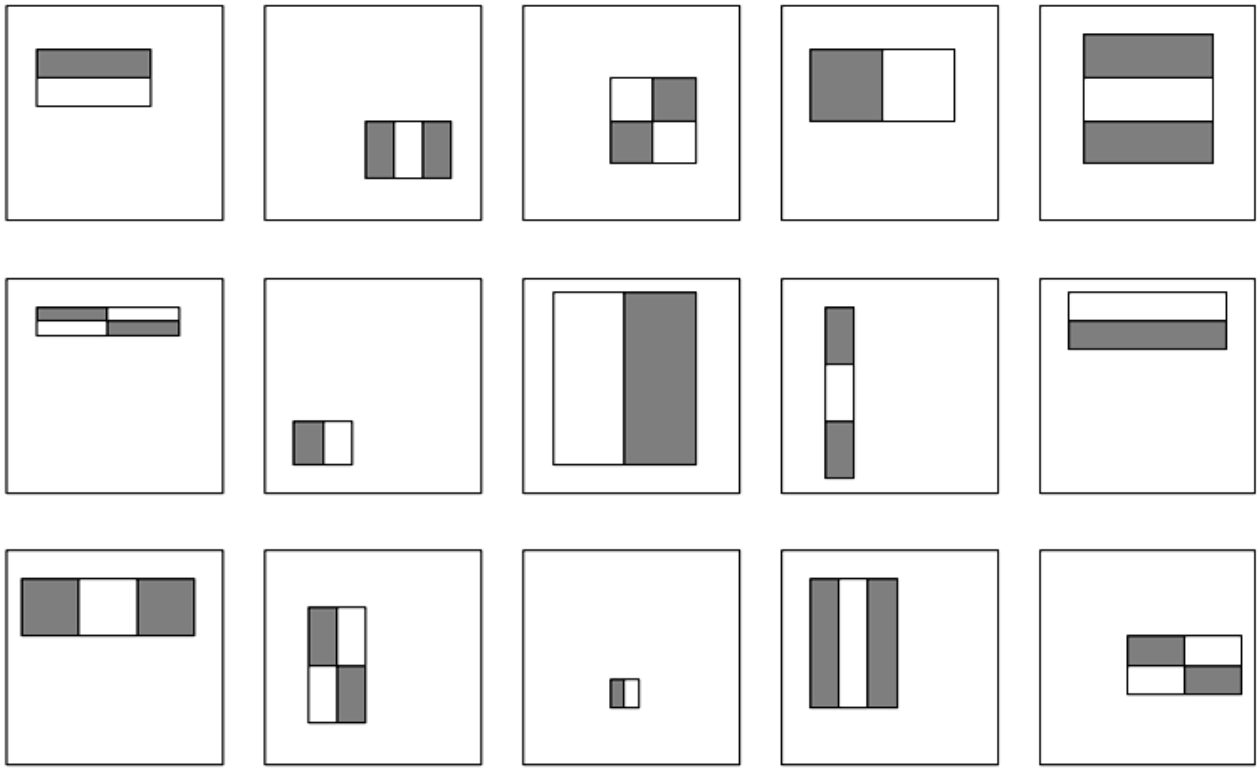
But, what if we combine *thousands* of them…



Two-rectangle features Three-rectangle features Etc.

How many features are there?

For a 24x24 detection region, the number of possible rectangle features is ~160,000!



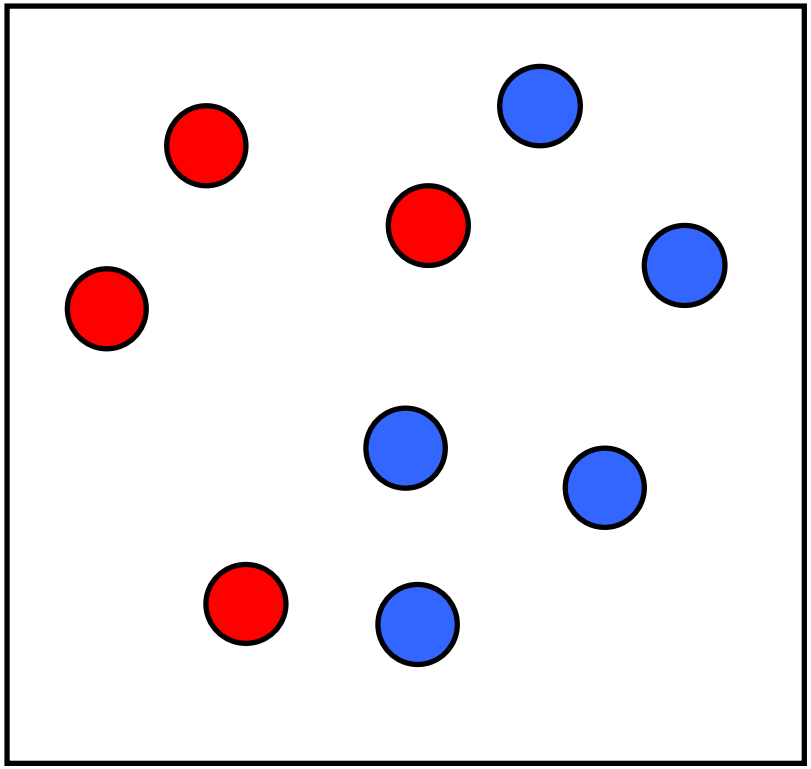
How many features are there?

- For a 24x24 detection region, the number of possible rectangle features is ~160,000!
- At test time, it is impractical to evaluate the entire feature set.
- Can we learn a ‘strong classifier’ using just a small subset of all possible features?

2. *Boosting* for feature selection

Initially, weight each training example equally.

Weight = size of point



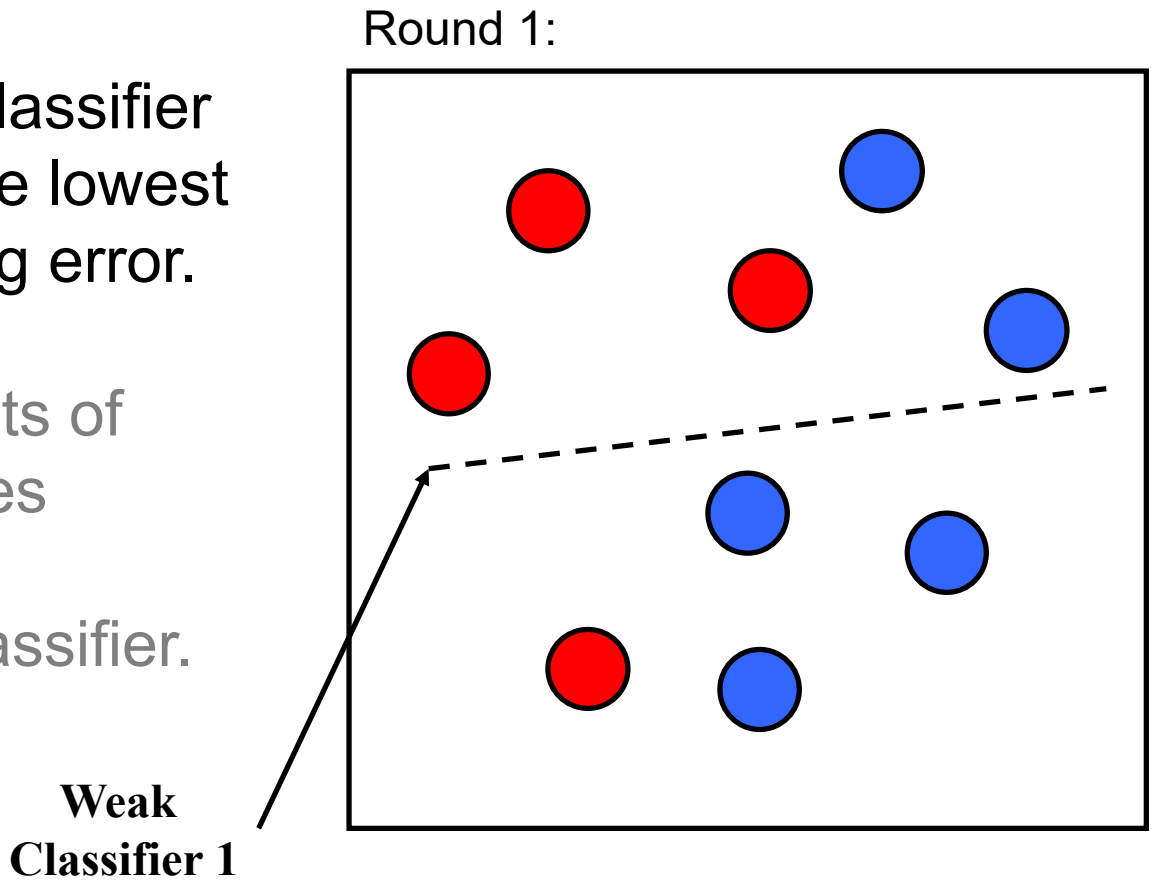
Slide credit: Paul Viola

2. *Boosting* for feature selection

In each boosting round:

Find the weak classifier that achieves the lowest *weighted* training error.

Raise the weights of training examples misclassified by current weak classifier.



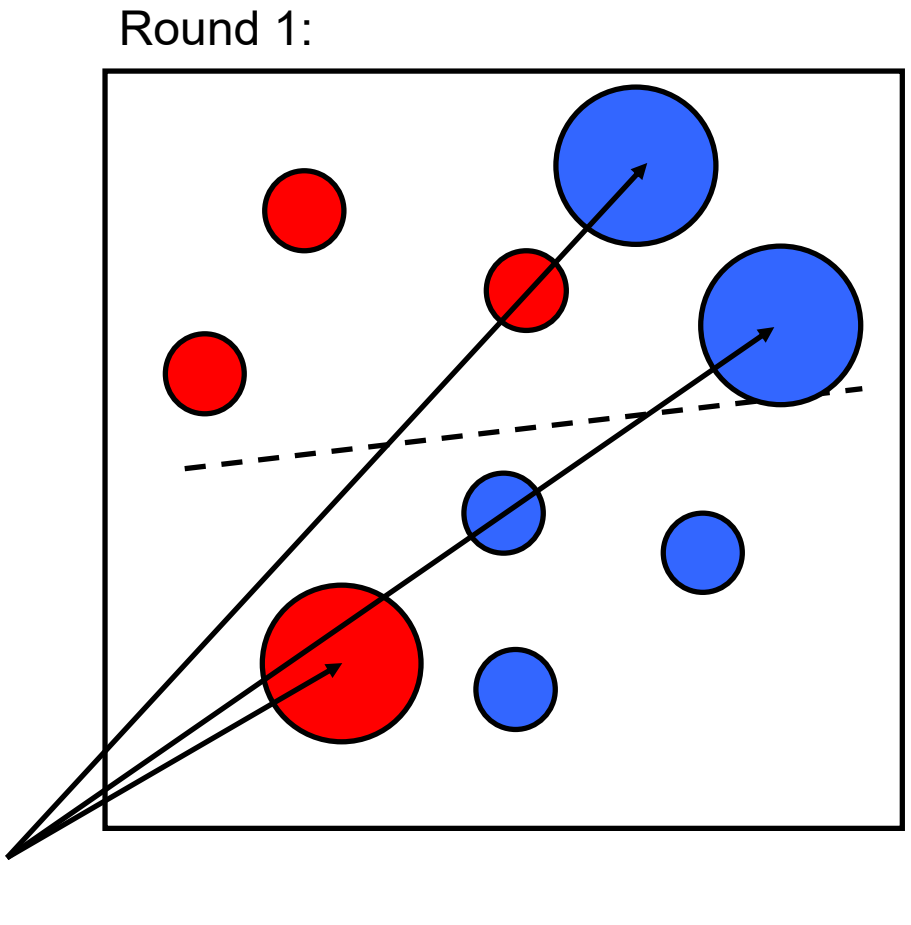
Slide credit: Paul Viola

Boosting illustration

In each boosting round:

Find the weak classifier that achieves the lowest *weighted* training error.

Raise the weights of training examples misclassified by current weak classifier.

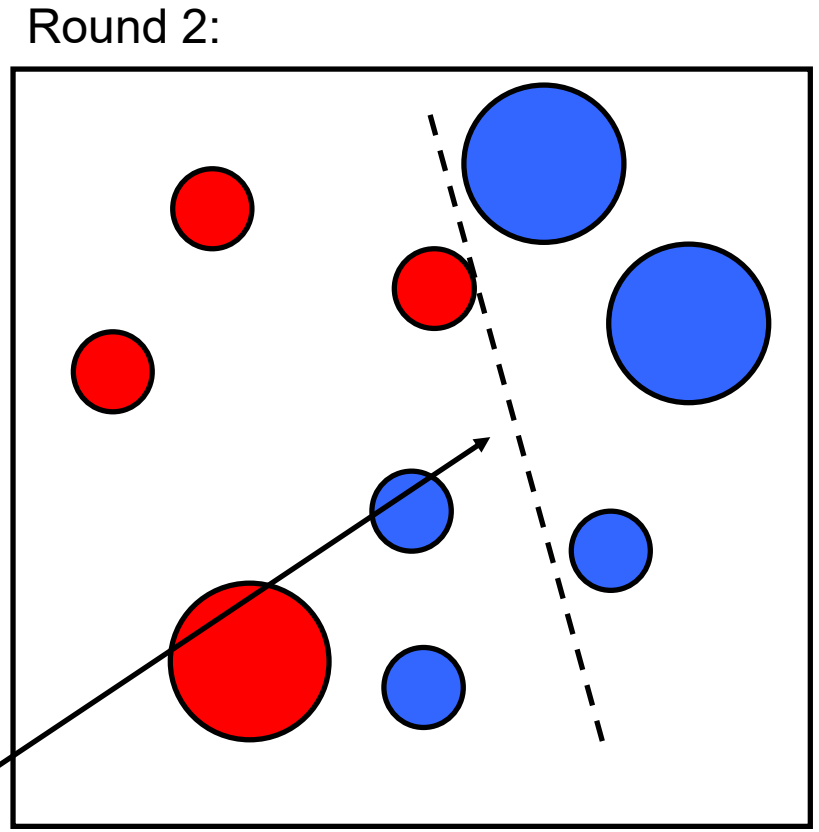


Boosting illustration

In each boosting round:

Find the weak classifier that achieves the lowest *weighted* training error.

Raise the weights of training examples misclassified by current weak classifier.

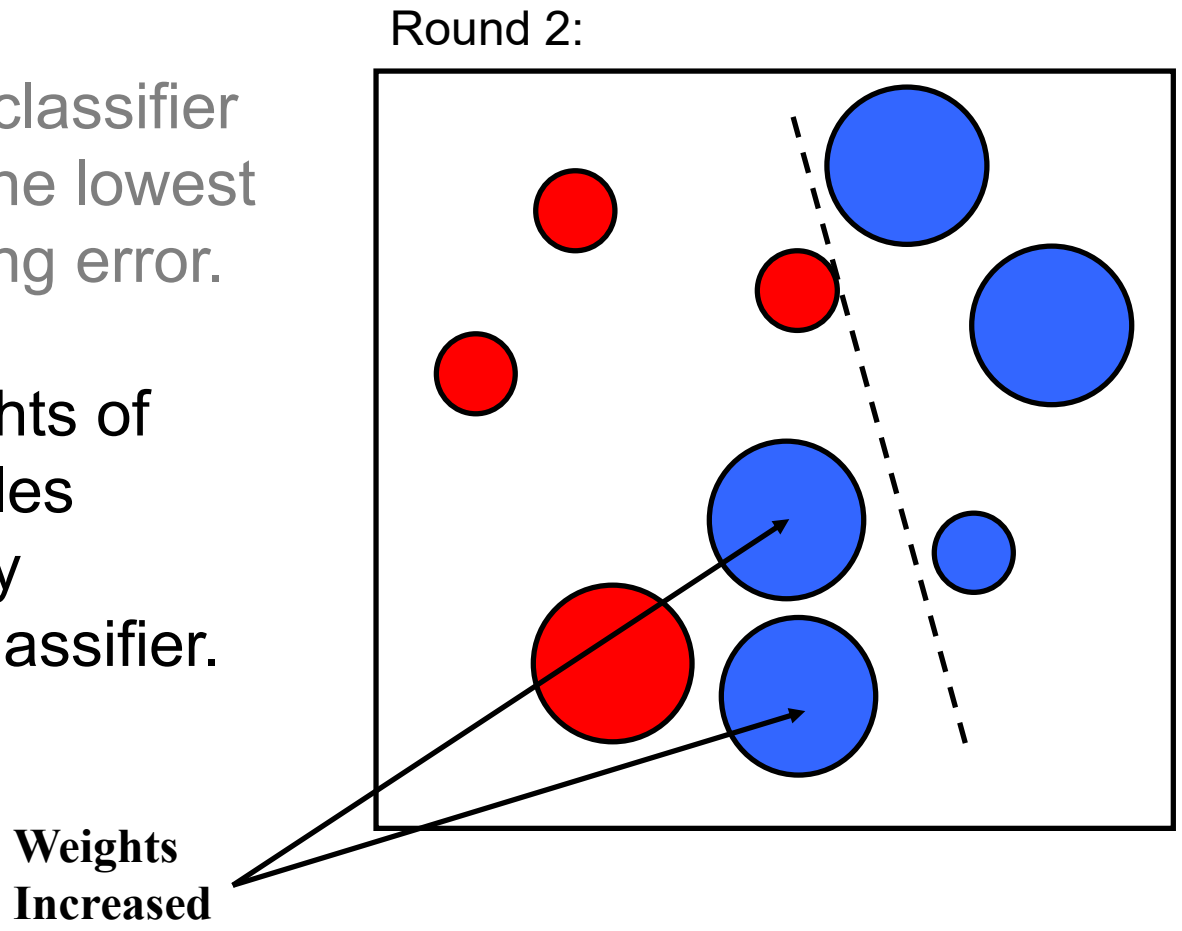


Boosting illustration

In each boosting round:

Find the weak classifier that achieves the lowest *weighted* training error.

Raise the weights of training examples misclassified by current weak classifier.

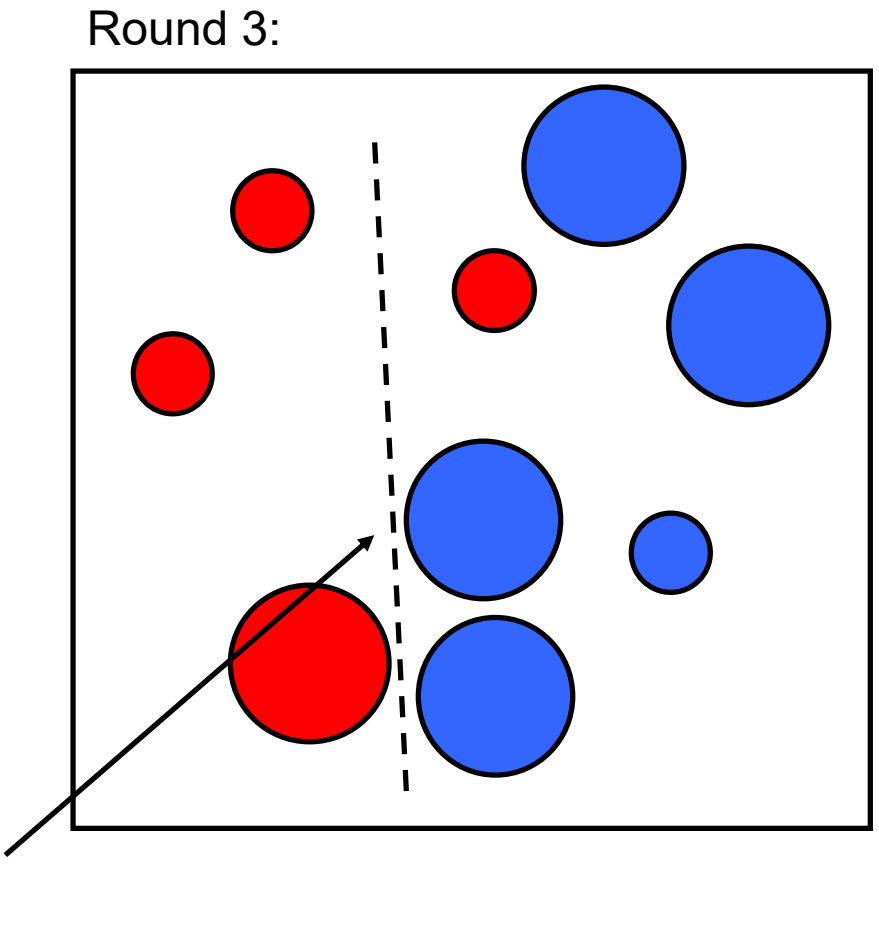


Boosting illustration

In each boosting round:

Find the weak classifier that achieves the lowest *weighted* training error.

Raise the weights of training examples misclassified by current weak classifier.

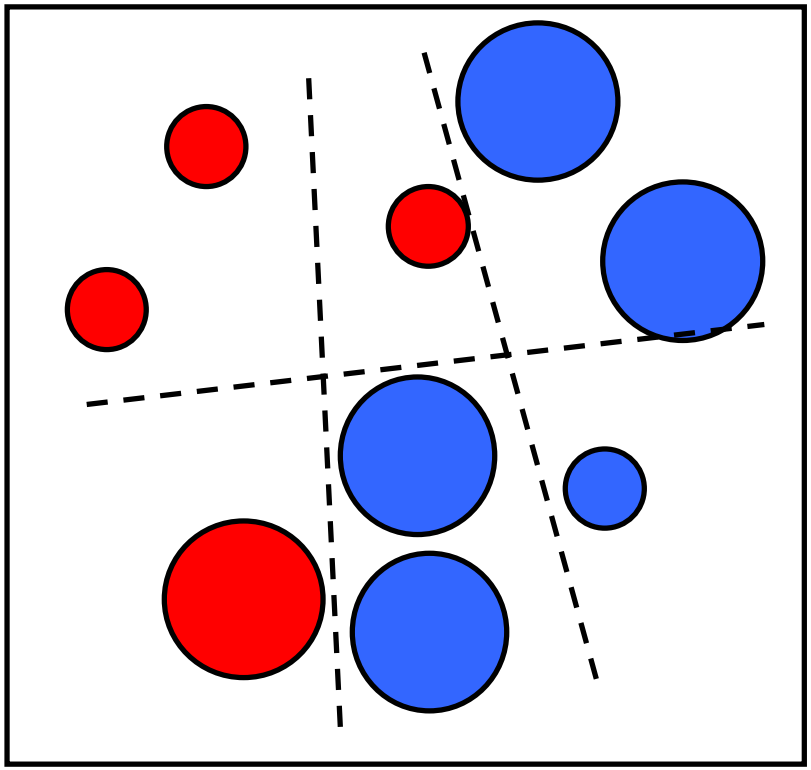


Boosting illustration

Compute final classifier as linear combination of all weak classifier.

Weight of each classifier is directly proportional to its accuracy.

Round 3:

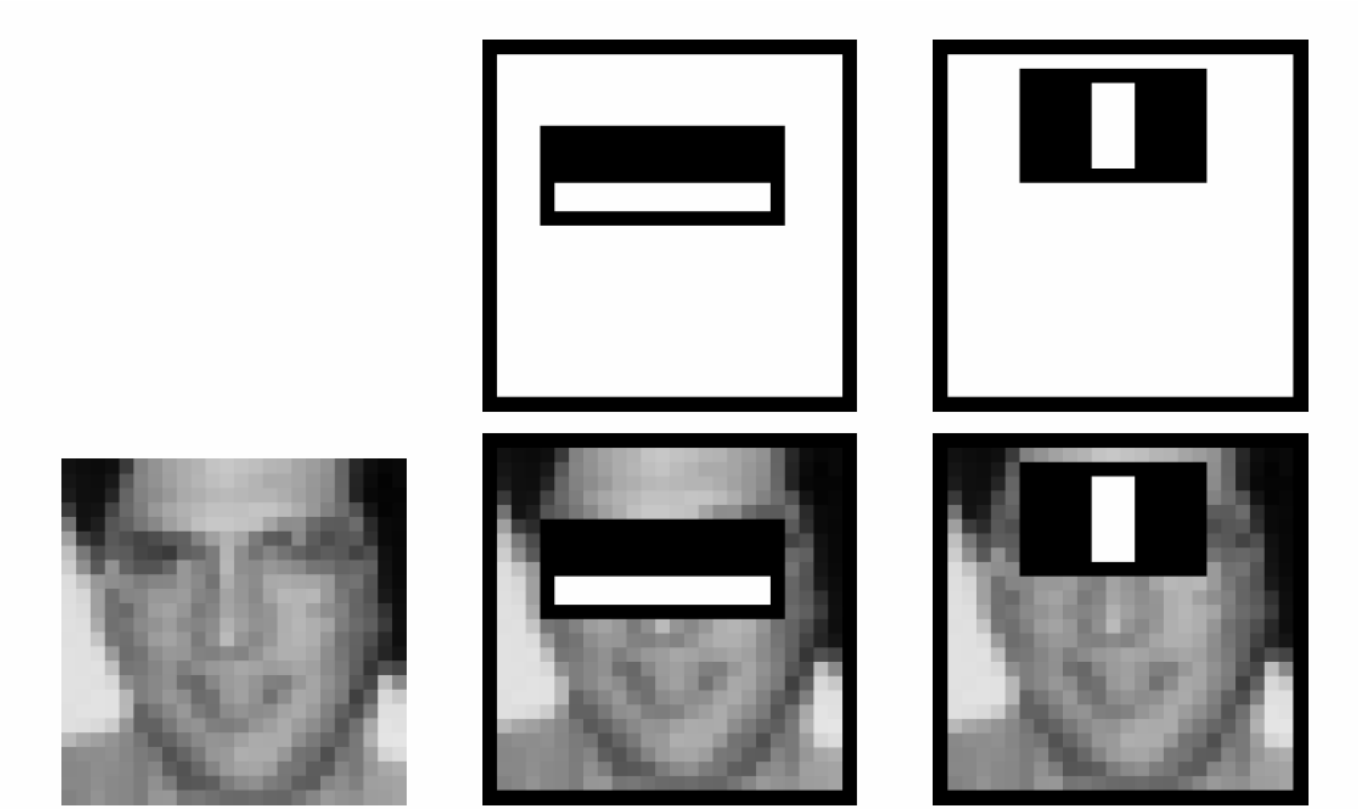


Exact formulas for re-weighting and combining weak learners depend on the particular boosting scheme (e.g., AdaBoost).

Y. Freund and R. Schapire, A short introduction to boosting,
Journal of Japanese Society for Artificial Intelligence, 14(5):771-780, September, 1999.

Boosting for face detection

- First two features selected by boosting:



This feature combination can yield 100% recall and 50% false positive rate

Feature selection with boosting

- Create a large pool of features (160K)
- Select discriminative features that work well together

Final strong learner → $h(\mathbf{x}) = \text{sign}\left(\sum_{j=1}^{M} \alpha_j h_j(\mathbf{x})\right)$ ← Weak learner

window ← Learner weight

  - "Weak learner" = feature + threshold + 'polarity'

value of rectangle feature → $h_j(\mathbf{x}) = \begin{cases} -s_j & \text{if } f_j < \theta_j \\ s_j & \text{otherwise} \end{cases}$ ← threshold

'polarity' = black or white region flip → $s_j \in \pm 1$

  - Choose weak learner that minimizes error on the weighted training set, then reweight

Adaboost pseudocode Szeliski p665

1. Input the positive and negative training examples along with their labels $\{(\boldsymbol{x}_i, y_i)\}$, where $y_i = 1$ for positive (face) examples and $y_i = -1$ for negative examples.
2. Initialize all the weights to $w_{i,1} \leftarrow \frac{1}{N}$, where N is the number of training examples. (Viola and Jones (2004) use a separate N_1 and N_2 for positive and negative examples.)
3. For each training stage $j = 1 \ldots M$:
   (a) Renormalize the weights so that they sum up to 1 (divide them by their sum).
   (b) Select the best classifier $h_j(\boldsymbol{x}; f_j, \theta_j, s_j)$ by finding the one that minimizes the weighted classification error

$$e_j = \sum_{i=0}^{N-1} w_{i,j} e_{i,j}, \tag{14.3}$$

$$e_{i,j} = 1 - \delta(y_i, h_j(\boldsymbol{x}_i; f_j, \theta_j, s_j)). \tag{14.4}$$

   For any given f_j function, the optimal values of (θ_j, s_j) can be found in linear time using a variant of weighted median computation (Exercise 14.2).
   (c) Compute the modified error rate β_j and classifier weight α_j,

$$\beta_j = \frac{e_j}{1 - e_j} \quad \text{and} \quad \alpha_j = -\log \beta_j. \tag{14.5}$$

   (d) Update the weights according to the classification errors $e_{i,j}$

$$w_{i,j+1} \leftarrow w_{i,j} \beta_j^{1-e_{i,j}}, \tag{14.6}$$

   i.e., downweight the training samples that were correctly classified in proportion to the overall classification error.
4. Set the final classifier to

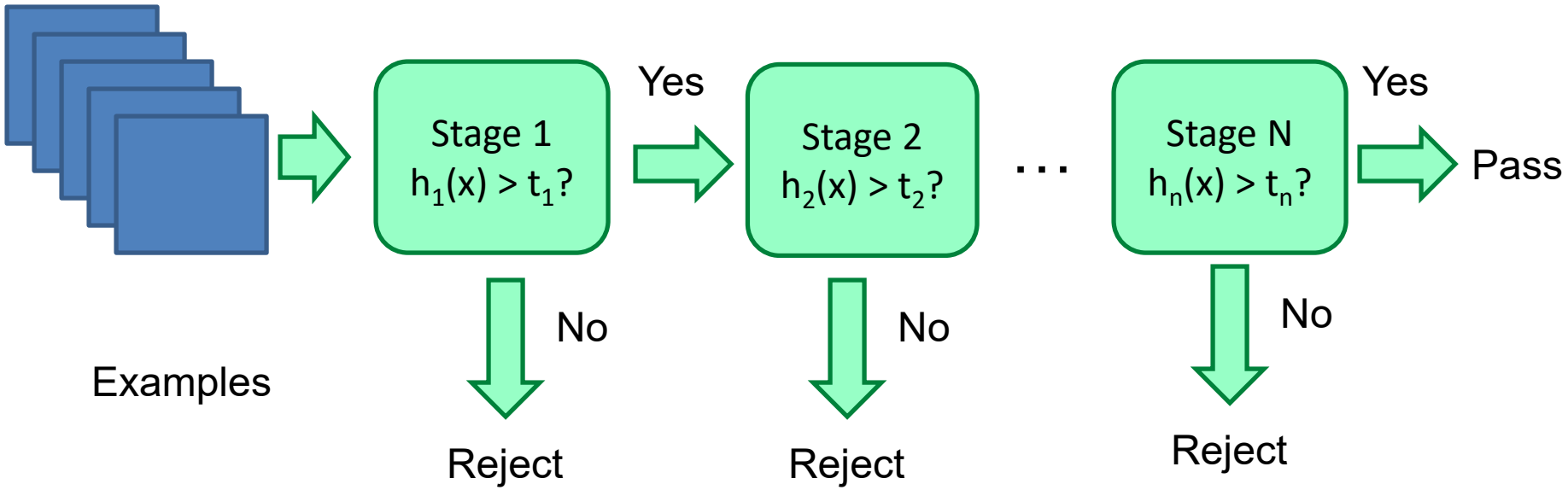
$$h(\boldsymbol{x}) = \text{sign}\left[\sum_{j=0}^{m-1} \alpha_j h_j(\boldsymbol{x})\right]. \tag{14.7}$$

Boosting vs. SVM

- Advantages of boosting
  - Integrates classifier training with feature selection
  - Complexity of training is linear instead of quadratic in the number of training examples
  - Flexibility in the choice of weak learners, boosting scheme
  - Testing is fast

- Disadvantages
  - Needs many training examples
  - Training is slow
  - Often doesn't work as well as SVM (especially for many-class problems)

Cascade for Fast Detection



Fast classifiers early in cascade which reject many negative examples but detect almost all positive examples.

Slow classifiers later, but most examples don't get there.

Training the cascade

- Set target detection and false positive rates for each stage
- Keep adding features to the current stage until its target rates have been met
  - Need to lower boosting threshold to maximize detection (as opposed to minimizing total classification error)
  - Test on a *validation set*
- If the overall false positive rate is not low enough, then add another stage
- Use false positives from current stage as the negative training examples for the next stage

The implemented system

- Training Data
  - 5000 faces
    - All frontal, rescaled to 24x24 pixels
  - 300 million non-faces
    - 9500 non-face images
  - Faces are normalized
    - Scale, translation
- Many variations
  - Across individuals
  - Illumination
  - Pose

Viola-Jones details

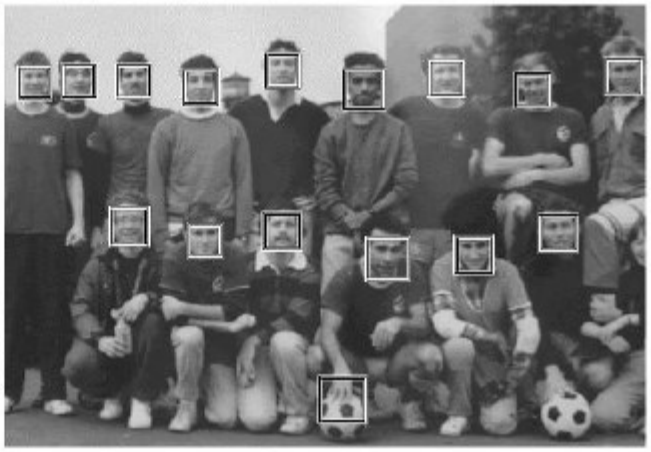
- 38 stages with 1, 10, 25, 50 … features
  - 6061 total used out of 180K candidates
  - 10 features evaluated on average
- Training Examples
  - 4916 positive examples
  - 10000 negative examples collected after each stage
- Scanning
  - Scale detector rather than image
  - Scale steps = 1.25 (factor between two consecutive scales)
  - Translation 1*scale (# pixels between two consecutive windows)
- Non-max suppression: average coordinates of overlapping boxes
- Train 3 classifiers and take vote

System performance

- Training time: “weeks” on 466 MHz Sun workstation
- 38 cascade layers, total of 6061 features
- Average of 10 features evaluated per window on test set
- “On a 700 Mhz Pentium III processor, the face detector can process a 384 by 288 pixel image in about .067 seconds”
  - 15 Hz
  - 15 times faster than previous detector of comparable accuracy (Rowley et al., 1998)

Viola Jones Results

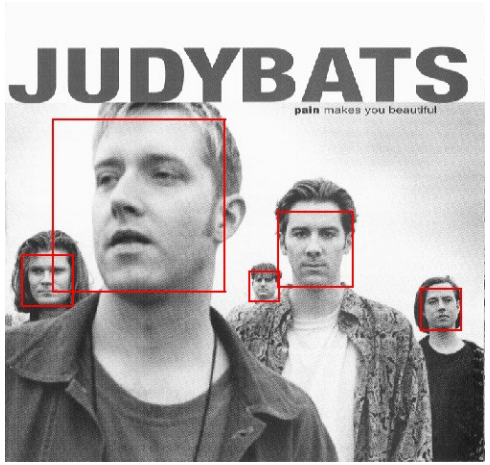
Speed = 15 FPS (in 2001)

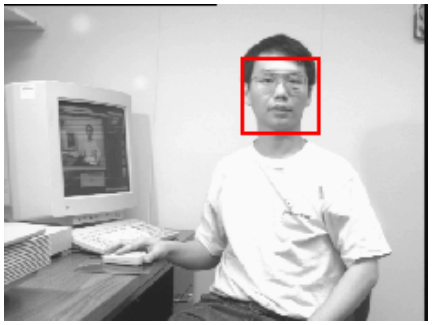



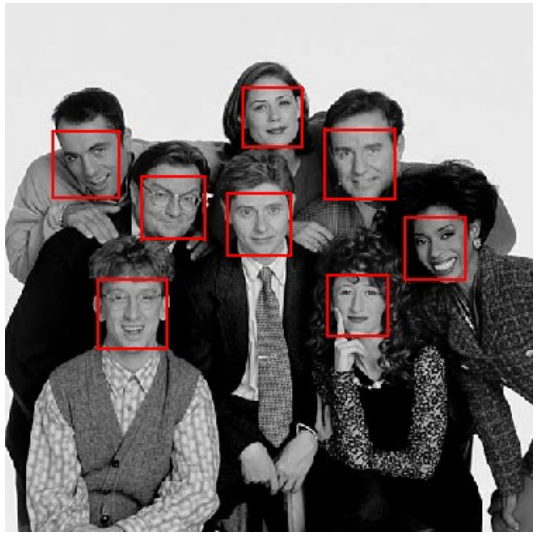

Detector \ False detections	10	31	50	65	78	95	167
Viola-Jones	76.1%	88.4%	91.4%	92.0%	92.1%	92.9%	93.9%
Viola-Jones (voting)	81.1%	89.7%	92.1%	93.1%	93.1%	93.2 %	93.7%
Rowley-Baluja-Kanade	83.2%	86.0%	-	-	-	89.2%	90.1%
Schneiderman-Kanade	-	-	-	94.4%	-	-	-
Roth-Yang-Ahuja	-	-	-	-	(94.8%)	-	-

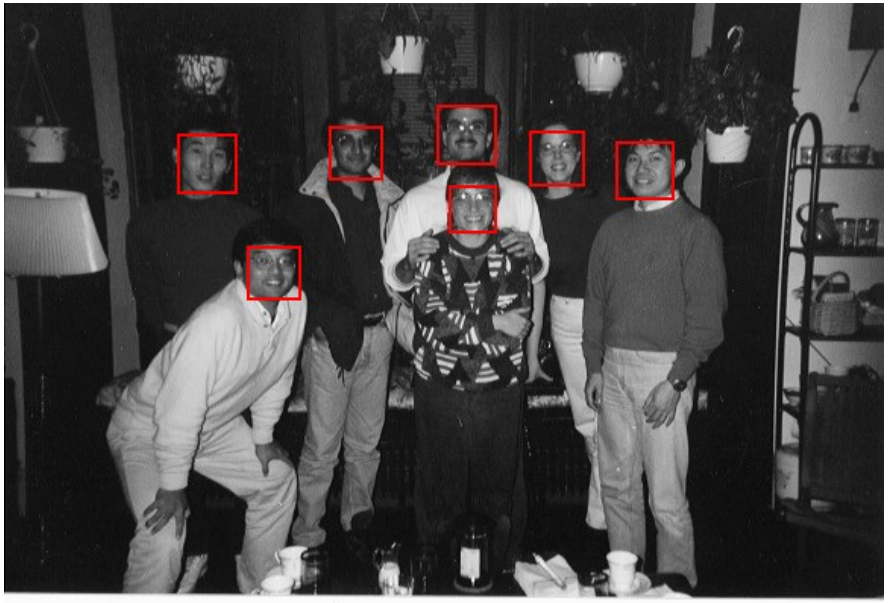
MIT + CMU face dataset

Output of Face Detector on Test Images







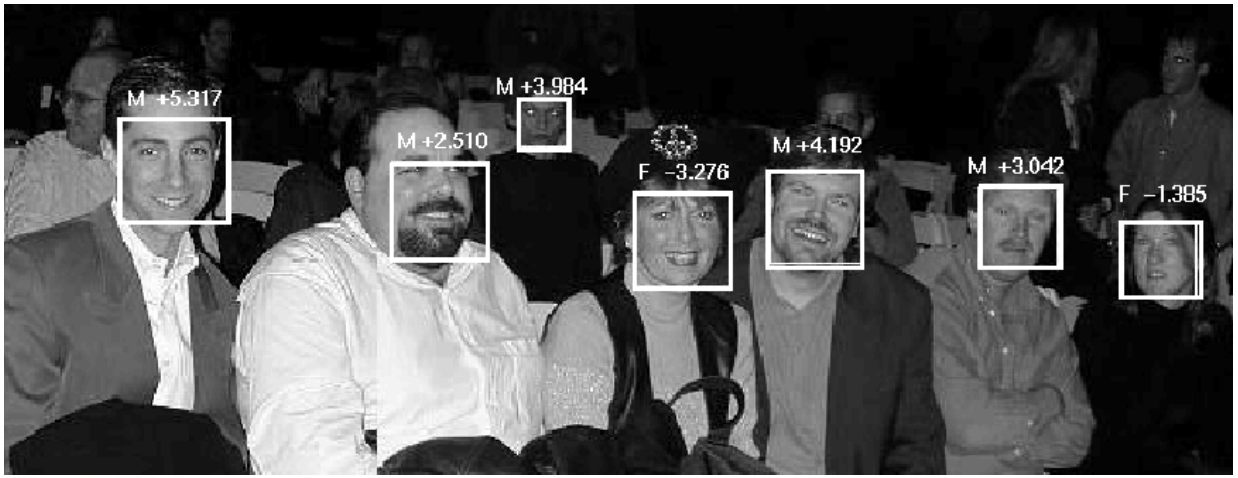


Other detection tasks

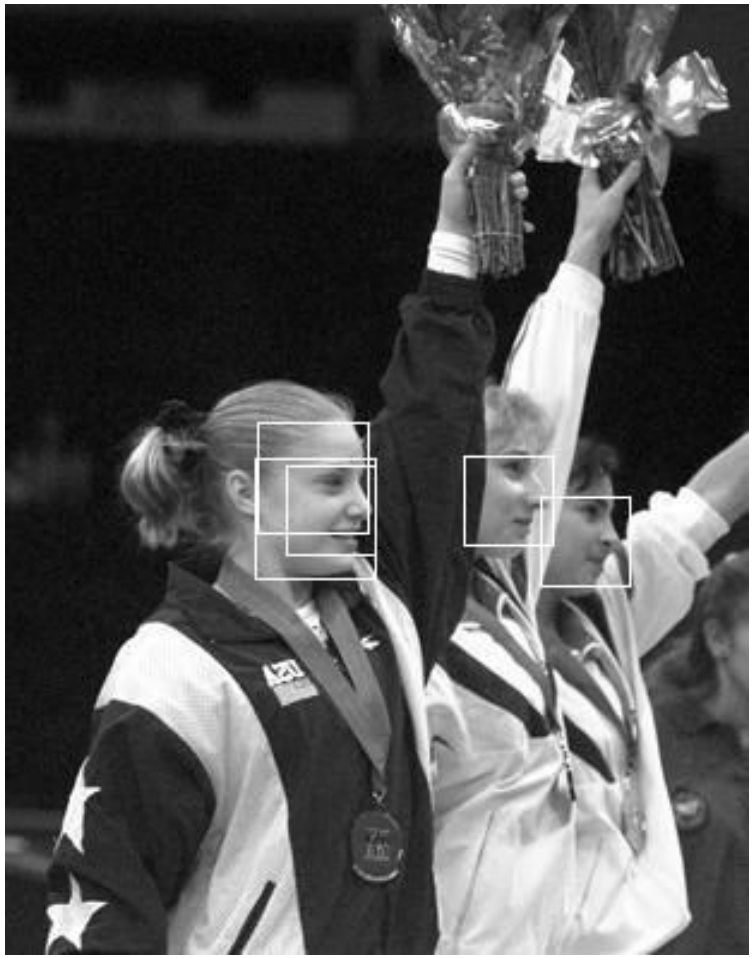
Facial Feature Localization

Profile Detection

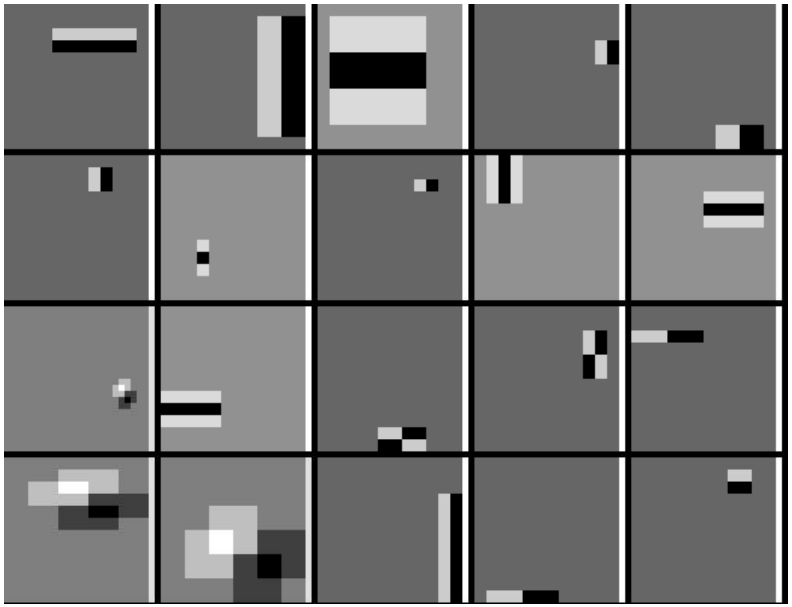
Male vs. female



Profile Detection



Profile Features

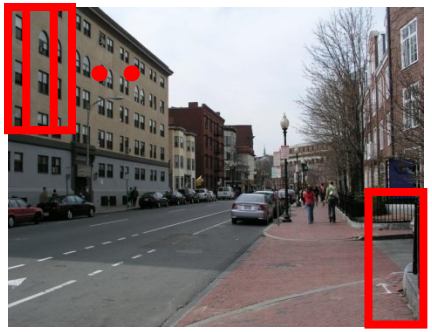


Summary: Viola/Jones detector

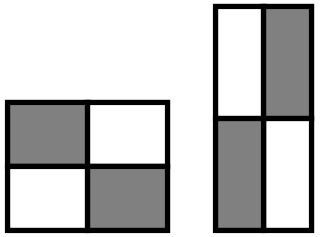
- Rectangle features
- Integral images for fast computation
- Boosting for feature selection
- Attentional cascade for fast rejection of negative windows

Things to remember

- Sliding window for search

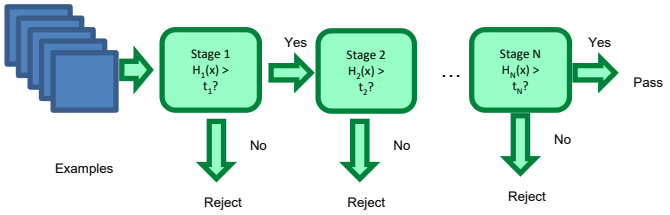


- Features based on differences of intensity (gradient, wavelet, etc.)
  - Excellent results = careful feature design



- Boosting for feature selection

- Integral images, cascade for speed



- Bootstrapping to deal with many, many negative examples