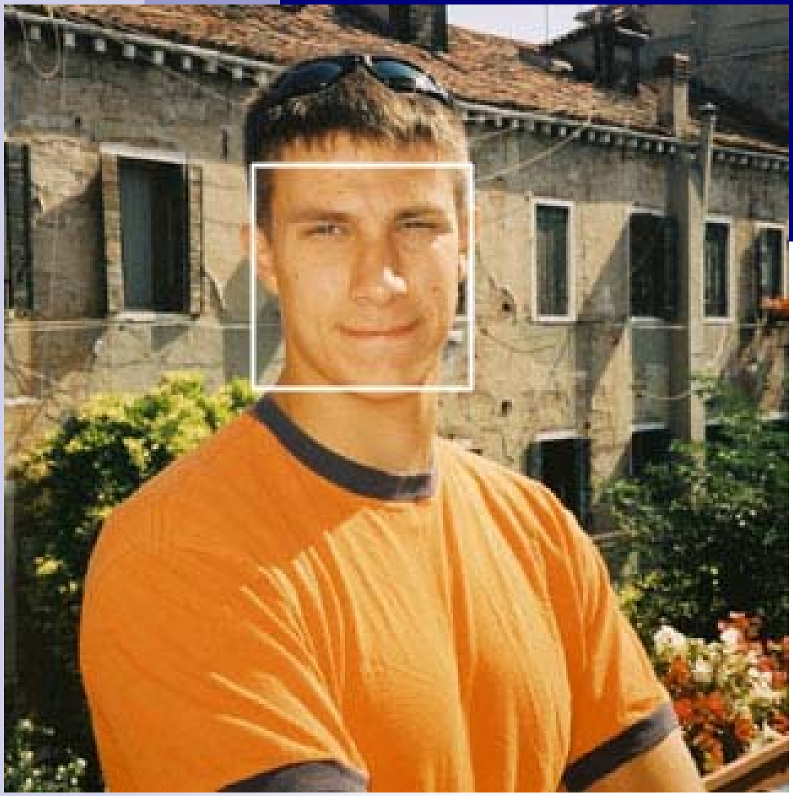
Robust Real-Time Face Detection



International Journal of Computer Vision 57(2), 2004

(first published in CVPR '01)

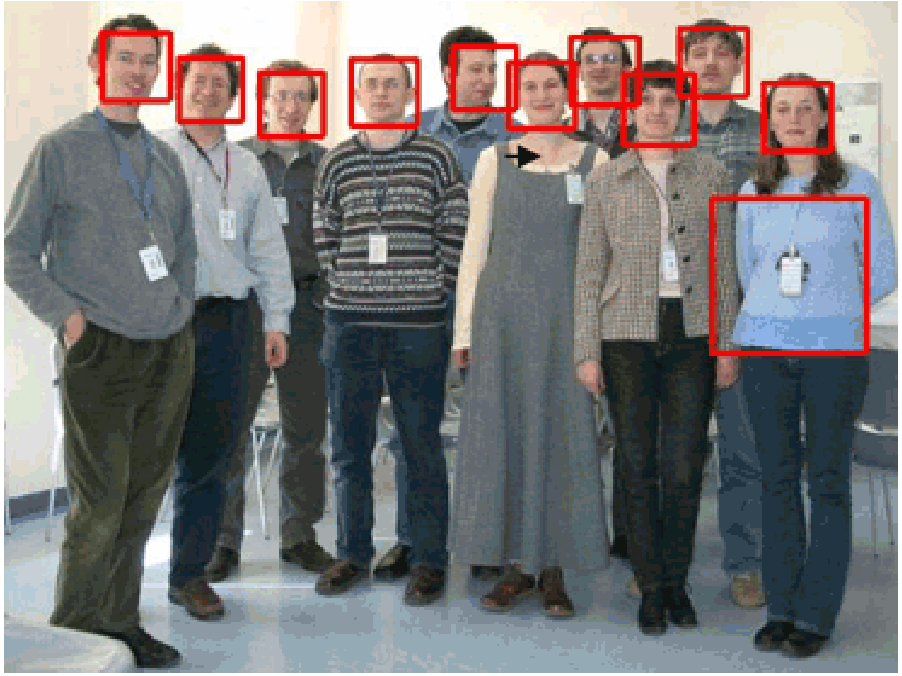
Paul Viola, Microsoft Research

Mike Jones, Mitsubishi Energy Research Lab (MERL)

Presented by Eugene Weinstein

Intro to Face Detection

- Given an image, determine
  - Whether any faces are present, and
  - Where the faces are located
- Many applications
  - Video conferencing
  - Surveillance
  - Biometric Identification
- Techniques relevant to general object recognition problem



Face Detection in Identification

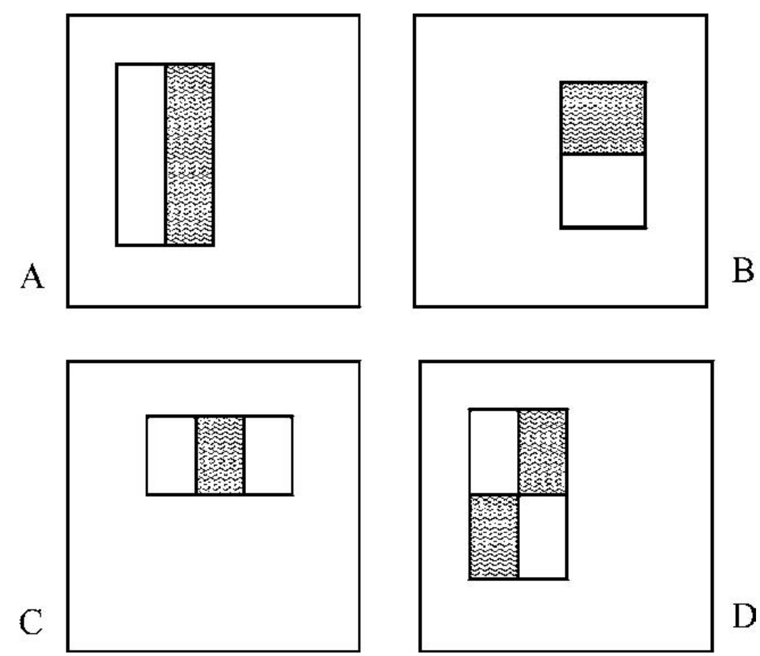
- Face detection is first step in an identification process
- Typical face identification process:

Viola/Jones Detector

- Main focus: speed
  - Achieves detection rates comparable to best systems
  - But, is much faster than most of them
- Main contributions
  1. “Integral Image” representation allows fast feature computation
  2. AdaBoost-based classifier training procedure
  3. Classifier cascade allows fast rejection of non-face images

Rectangular Features

- Use rectangle features instead of pixels
  - Features model face better with limited data
  - Feature-based classifier much faster
- Compute sum of pixels within a box, features are combinations of box sums:
  - B, W: Black, white regions
  - Two rectangles: W-B
  - Three: W1+W2-B
  - Four: W1+W2-(B1+B2)



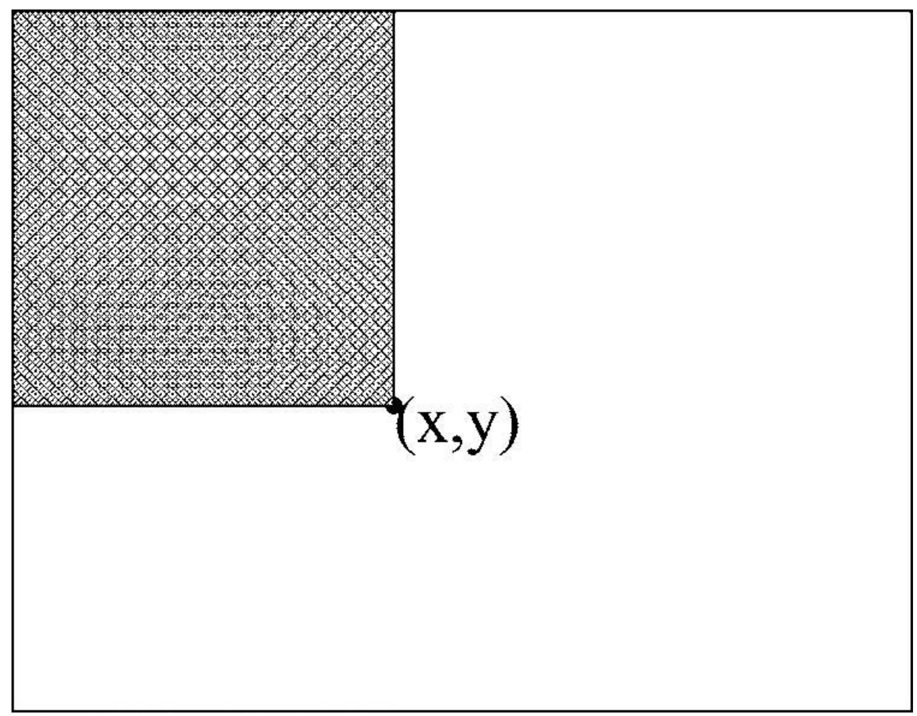
Integral Image

- Detector resolution: 24x24 → 160,000 possible rectangle features
- Fast way to compute: integral image
  - Integral image is the sum of pixels above and to the left

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

  - Can compute in one pass using the recurrences

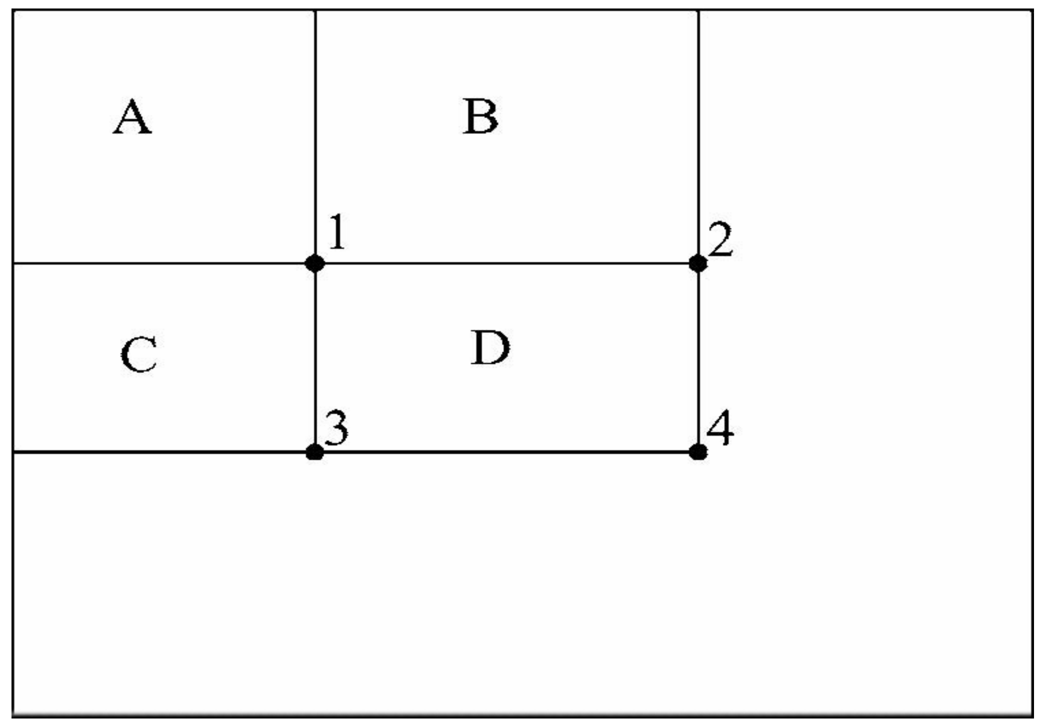
$$\begin{aligned} s(x,y) &= s(x,y-1) + i(x,y) \\ ii(x,y) &= ii(x-1,y) + s(x,y) \end{aligned}$$



Using the Integral Image

- Rectangular sums can be computed with four array references:

$$
\begin{aligned}
ii(1) &= A \\
ii(2) &= A + B \\
ii(3) &= A + C \\
ii(4) &= A + B + C + D \\
D &= 4 - (2 + 3) + 1
\end{aligned}
$$



Learning the Classifier

- Each 24x24 region has 160,000 rectangle features >> # pixels!
- Impractical to compute complete feature set
- Idea: can make an effective classifier from a small number of features
- But which features?

AdaBoost for Feature Selection

- Standard AdaBoost scenario: boost classification performance of a "weak" classifier, e.g., perceptron
  - Apply to successively harder problems
  - Tweak parameters at each classification stage
- This work: use box sum features as weak classifiers
  - AdaBoost finds sequence of best features
- Training is more efficient than other algorithms
  - Linear in number of training examples: $O(MNK)=10^{11}$
    - K: # features (160,000)
    - N: # examples (20,000)
    - M: # iterations of AdaBoost (200)

AdaBoost Formal Guarantees

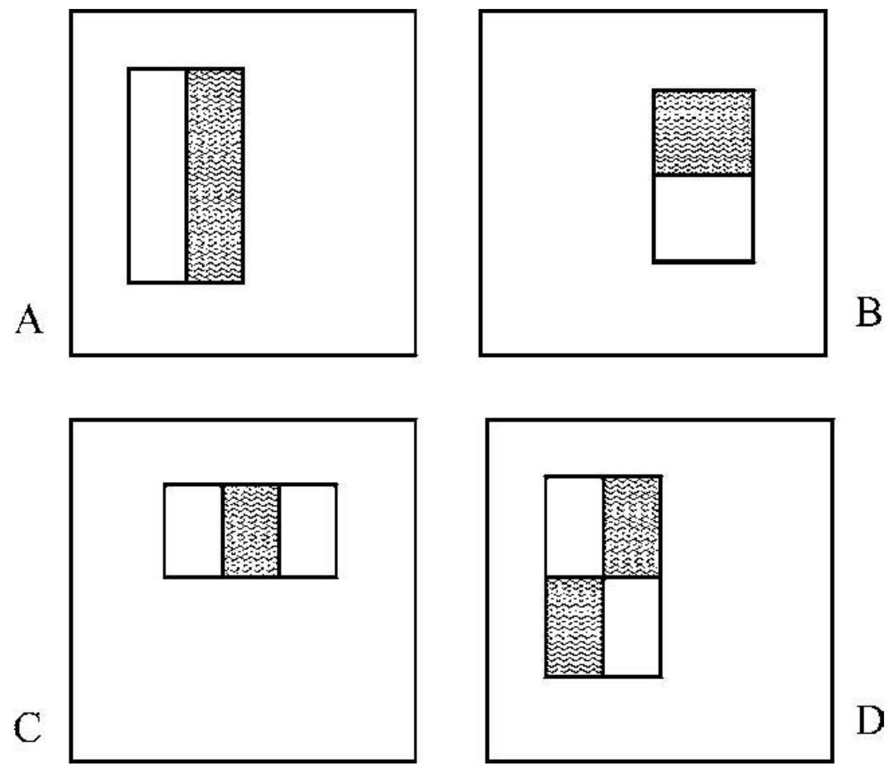
- Training error approaches zero exponentially
- Large margins are rapidly achieved
  - Large margins → good generalization error

Features as Weak Classifiers

- Take one feature, decide how to use it for classification

$$h(x, f, p, \theta) = \begin{cases} 1 & \text{if } pf(x) < p\theta \\ 0 & otherwise \end{cases}$$

  - f = feature
  - p = polarity {+1,-1}
  - θ = threshold



AdaBoost for Feature Selection

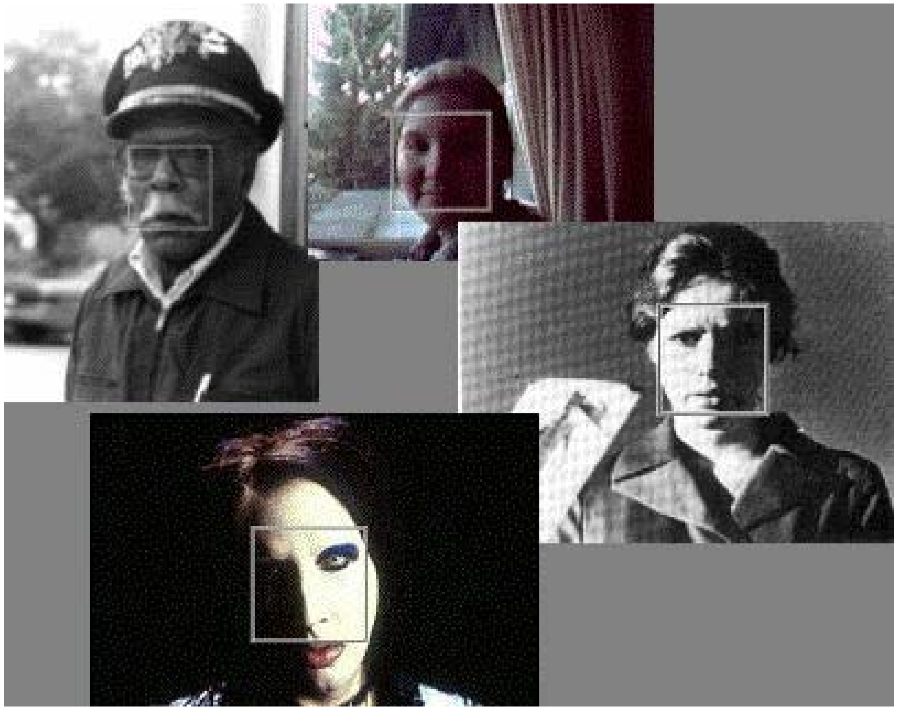
- Given: example images labeled +/-
- Repeat T times
  1. Select classifier with lowest weighted error over all
     - Features
     - Thresholds
     - Polarities
  2. Selected classifier is the hypothesis of this iteration
  3. Update the weights to emphasize examples on which this step's classifier is wrong
- Final (strong) classifier is a weighted combination of the weak classifiers
  - Weighted according to their accuracy

AdaBoost Initialization

- Given: example images x_i and labels y_i={0,1}
- Initialize weights:

$$w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$$

  - m , l : # positive, negative examples



AdaBoost Training Loop

- For *t=1,…,T*
  1. Normalize the weights: $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$
  2. Select min-error classifier h_t :
     $\epsilon_t = \text{min}_{f,p,\theta} \sum_i w_i |h(x_i, f, p, \theta) - y_i|$
  3. If x_i classified incorrectly, don't change its weight. Otherwise, adjust its weight down:
     $w_{t+1,i} = w_{t,i} \, \frac{\epsilon_t}{1-\epsilon_t}$

Final (Strong) Classifier

- Linear combination of weak classifiers
- Weighted by performance of each classifier

$$C(x) = \text{sign}\left[\sum_{t=1}^{T}\left(\log\frac{1-\epsilon_t}{\epsilon_t}\right)\left(h_t(x)-\frac{1}{2}\right)\right]$$

- Note, if $\epsilon_t = 0.5$, classifier t does not contribute to combination

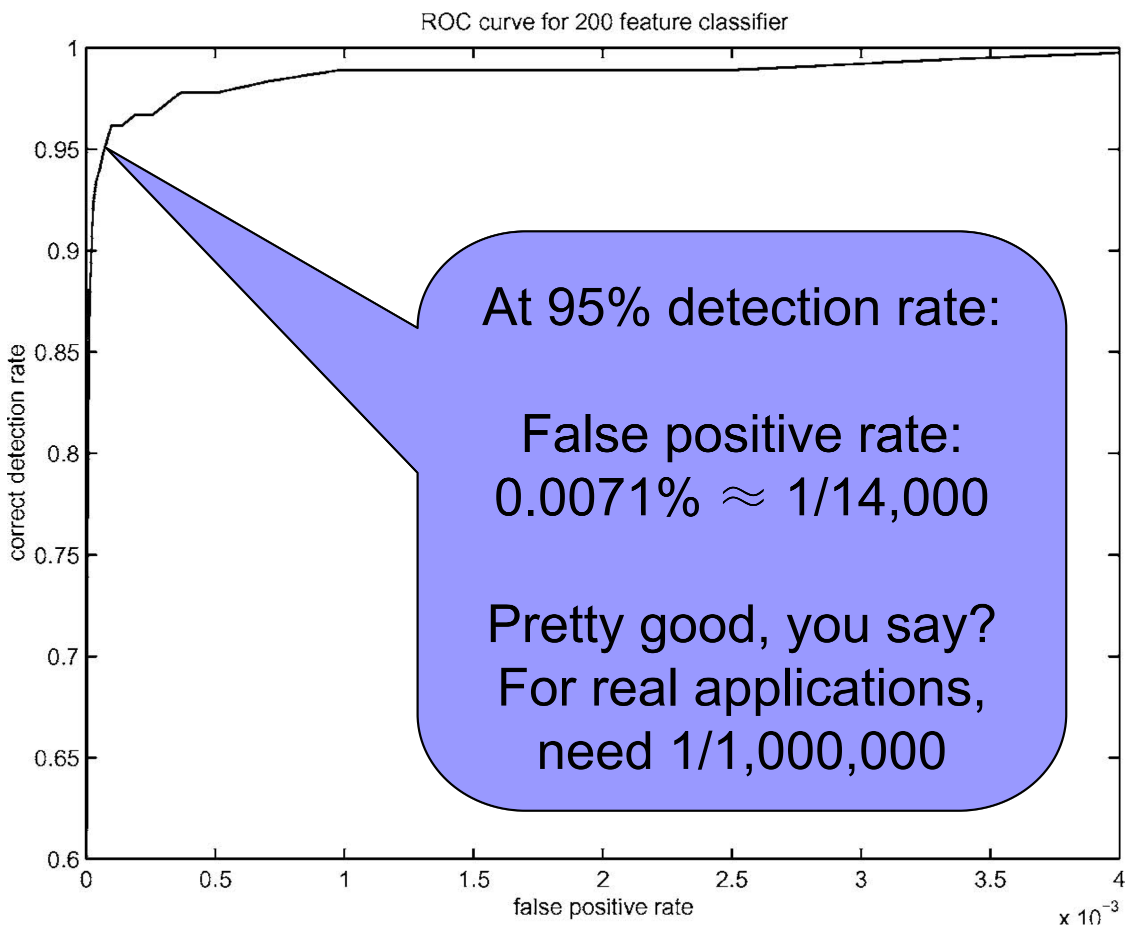
Classifier Characteristics

- First two features selected are quite intuitive

- Accurate, but not enough for real tasks
- Fast: 0.7 seconds for 384x288 image
  - But, adding more features increases computation time linearly
  - So, how to improve accuracy and keep the speed?

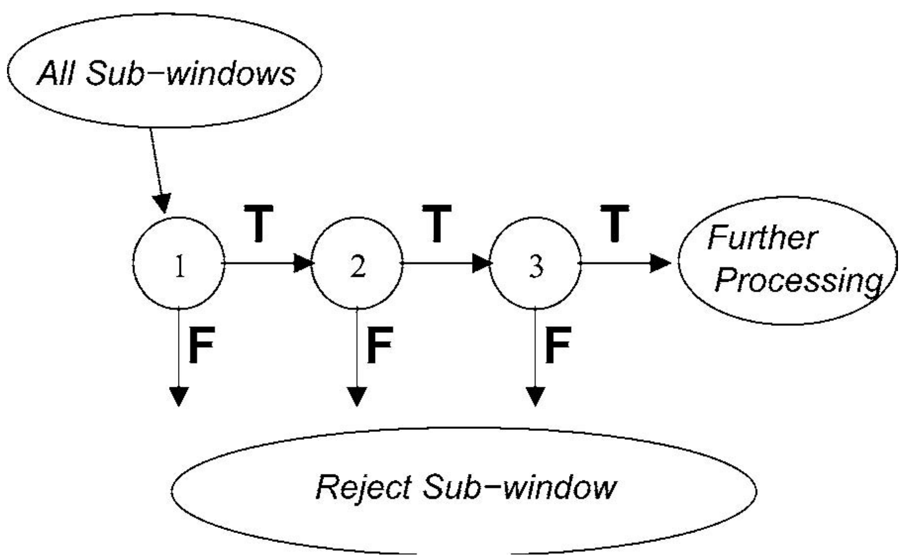
Accuracy of Resulting Classifier

- Detection tasks: to get more true detections, give up more false positives – ROC curve



The Attentional Cascade

- Use degenerate decision tree of classifiers
- A negative result from <u>any</u> classifier leads to immediate rejection
- Idea: Vast majority of sub-windows are rejected very quickly



Cascade Training Methodology

- Each classifier trained on false positives of previous stages
- Second classifier gets harder task than first, and etc.
- To train, first decide on accuracy and speed goals
  - Past systems get 85-95% detection rates at 10^{-5}-10^{-6} false positive rate
  - Goal is to match this with max performance

Cascade Performance Goals

- Cascade of K classifiers
  - f_i: false positive rate of ith classifier
  - d_i: detection rate of ith classifier
- Total false positive, detection rates are

$$F = \prod_{i=1}^{K} f_i \qquad D = \prod_{i=1}^{K} d_i$$

Setting Performance Goals

$$F = \prod_{i=1}^{K} f_i \qquad D = \prod_{i=1}^{K} d_i$$

- Can set goals for FP/det rate
  - E.g., to get 0.9 det rate from 10-stage cascade, need 0.99 det rate at each stage ($0.99^{10} \approx 0.9$)
  - But, only need FP rate of 30% ($0.3^{10} \approx 6 \times 10^{-6}$)
- Want classifiers with high detection rate, and can accept large FP rates

AdaBoost Classifier Again

- Linear combination of weak classifiers
- Weighted by accuracy of each classifier

$$C(x) = \text{sign}\left[\sum_{t=1}^{T}\left(\log\frac{1-\epsilon_t}{\epsilon_t}\right)\left(h_t(x)-\frac{1}{2}\right)\right]$$

Tweaking the Thresholds

- Remember the term $h_t(x)-1/2$?
- 1/2 is the default AdaBoost threshold
  - But what if we try to tweak it?
- Say we only care about detection rate
  - Can achieve 100% with only two features
  - But… with 50% false positive rate
  - And it’s fast! ≈60 CPU instructions

A Very Big But…

- We can tweak the AdaBoost thresholds to give us desired detection/FP rates
- But, effect on training and generalization guarantees of AdaBoost currently unclear!
- Ideally, want to globally optimize
  - Number of classifier stages
  - Number of features in each stage
  - Threshold of each stage
- But, not currently feasible

And Now for the Real Algorithm

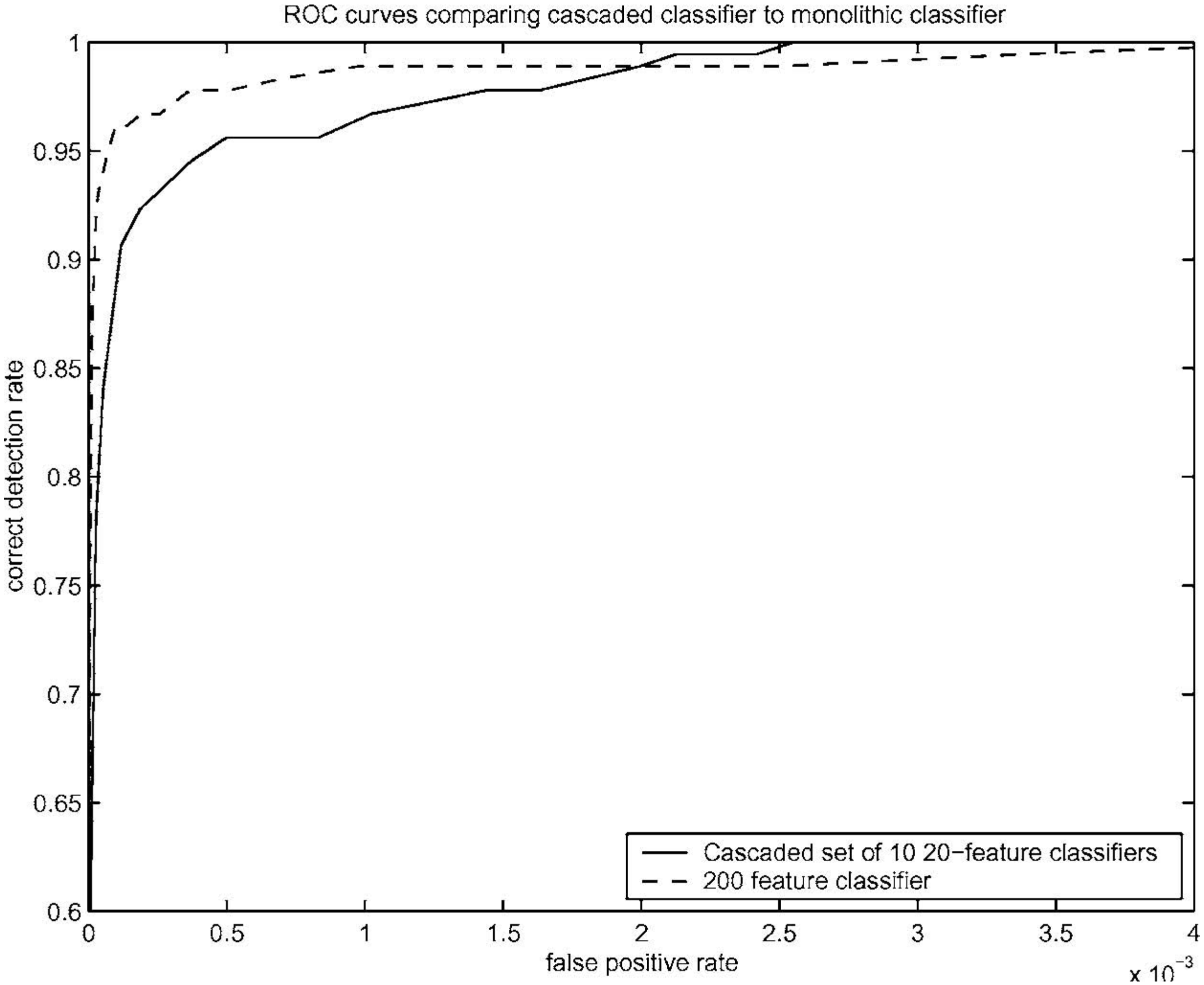
- Inputs
  - f, maximum acceptable FP rate per layer
  - d, minimum acceptable det rate per layer
  - F_{target}, target overall FP rate
- Idea: keep adding classifiers until you meet the performance targets

Training Algorithm for Cascade

- While global FP rate not met
  - n ←0
  - Repeat
    - n←n+1
    - Train a classifier from n features with AdaBoost
    - Evaluate it on validation set
    - Decrease threshold of classifier until its detection rate is at least d
  - Until we find a classifier with FP rate <f
  - Add classifier to cascade
    - Train future classifiers on false positives

An Experiment

- To evaluate cascade approach, train
  - Single 200-feature classifier
  - Cascade of ten 20-feature classifiers



Final System – Training

- 4,916 faces cropped by hand, scaled to 24x24
- 9,500 non-face images

Final Cascade

- Features for initial classifiers chosen by hand (“trial and error”):
  - 2, 10, 25, 25, 50, 50, 50 features
  - Chosen manually “to reduce training time”
- Then, use training algorithm – sort of…
  - Add 25 features at a time instead of one
- Final result
  - 38 classifier layers
  - 6,060 total features

Training and Detection Speed

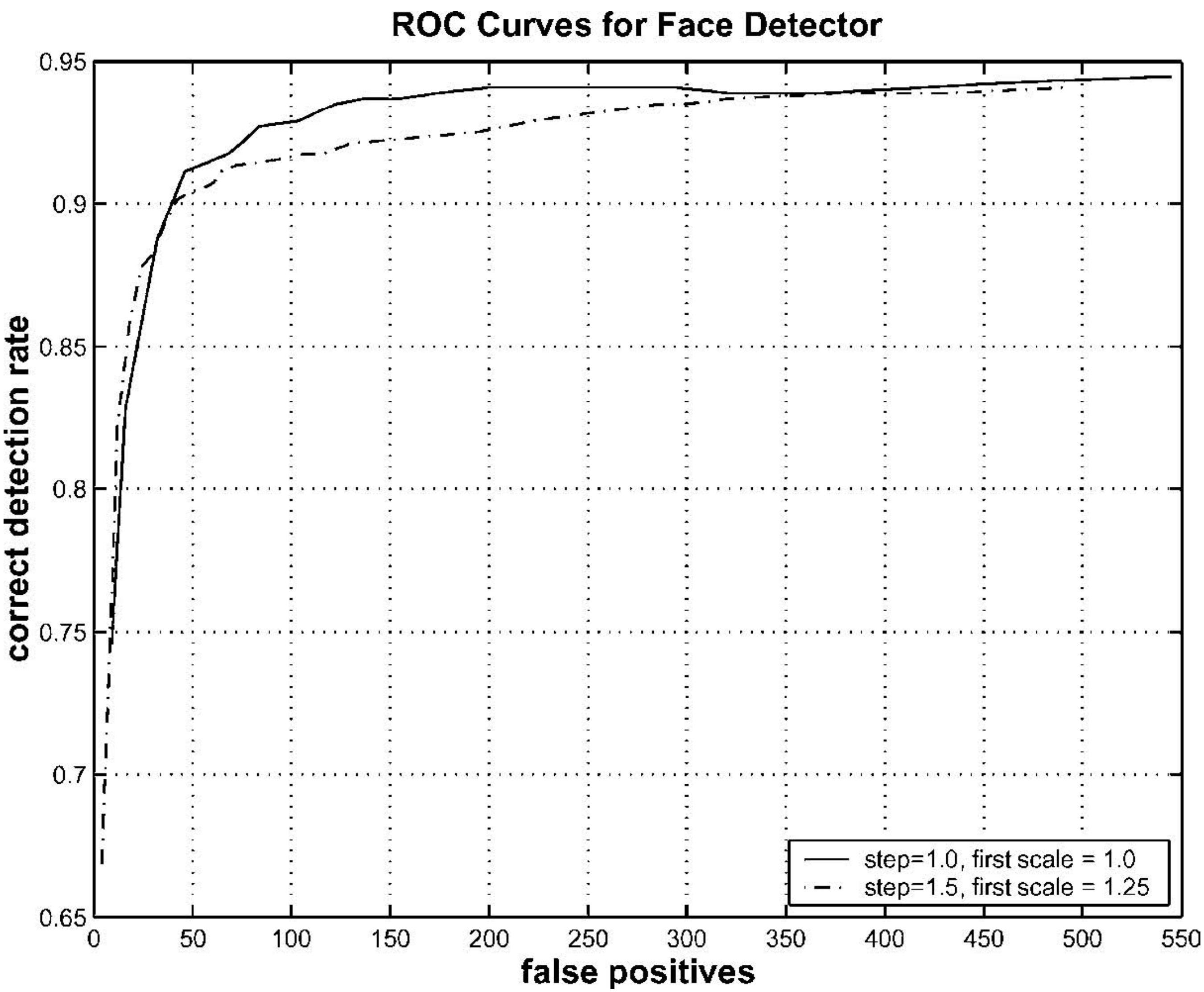
- Training: “weeks” on 466 MHz Sun machine
  - Now can run in parallel in about a day
- On average, eight features are evaluated
- 384x288 image takes .067 seconds
  - That’s 15Hz!
  - 15 times faster than previous detector of comparable accuracy (Rowley et al., 1998)

Practical Issues

- Faces can occur at multiple scales
- Scale the detector, not the image
  - 1.25 scale step works well: 1.0x, 2.25x, etc.
- Sweep detector over all possible regions
- Multiple detections can occur for one face
  - Combine overlapping detections into one
  - Then, take the average to get final position

Final System – Testing

- MIT+CMU frontal face set: 130 images, 507 faces



Comparing with Previous Work

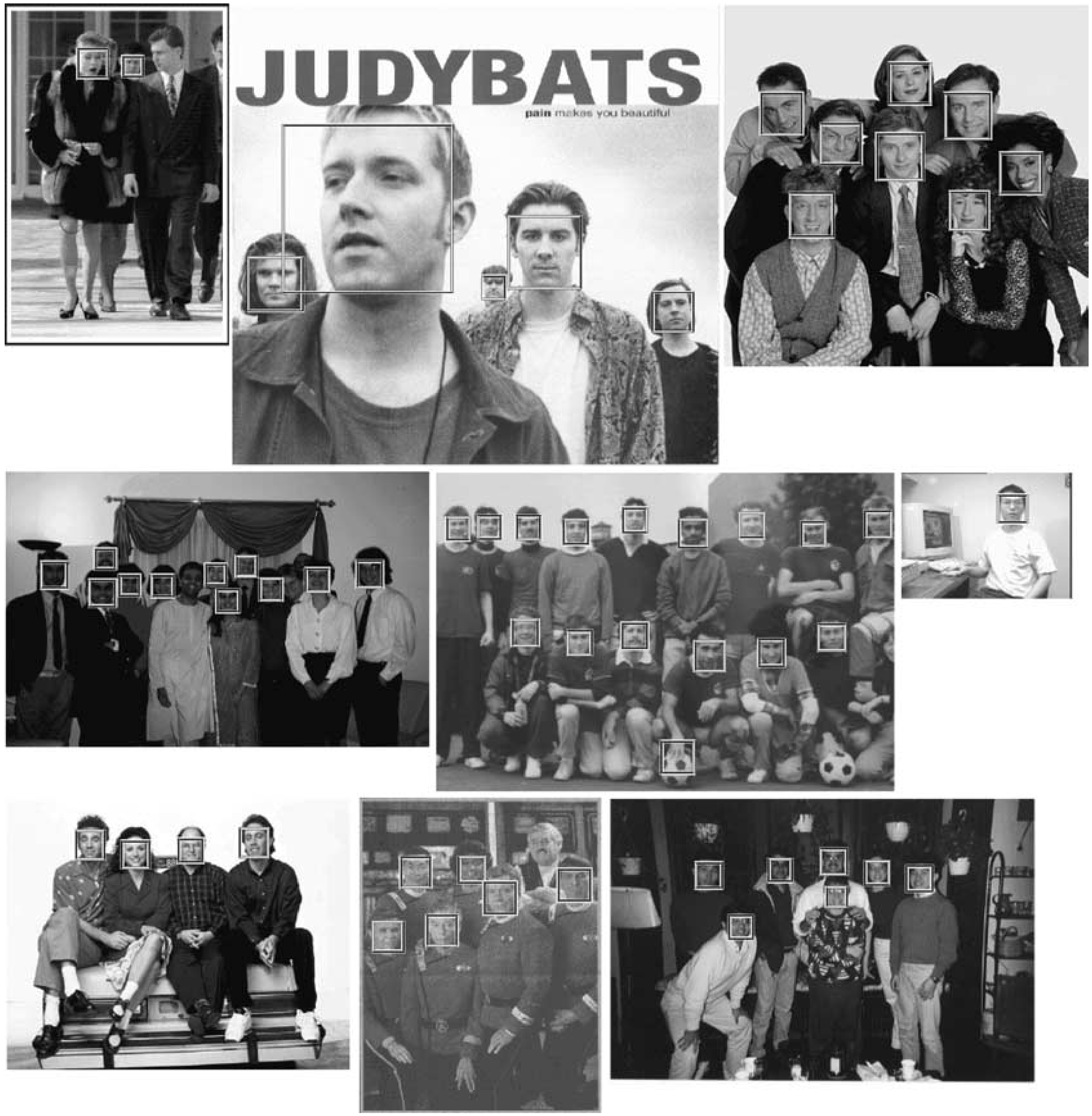
- MIT+CMU fontal face set: 130 images, 507 faces
- Near state-of-the-art accuracy
- State-of-the-art speed

	False detections							
Detector	10	31	50	65	78	95	167	422
Viola-Jones	76.1%	88.4%	91.4%	92.0%	92.1%	92.9%	93.9%	94.1%
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%	89.9%
Schneiderman-Kanade	–	–	–	94.4%	–	–	–	–
Roth-Yang-Ahuja	–	–	–	–	(94.8%)	–	–	–

Comments

- Viola-Jones is by far the most widely used face detector
- Works well, but some issues
  - AdaBoost theoretical guarantees not necessarily preserved
  - A lot of hand-tweaking
  - Not totally rotation invariant
    - ±15° in plane, ±45° out of plane
    - Future work addresses this

Gratuitous Example Images



Thank You!

- Any questions?