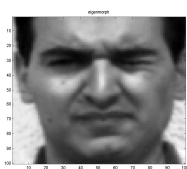


Machine Learning for Signal Processing

Detecting faces (& other objects) in images

Bhiksha Raj

Previously: How to describe a face

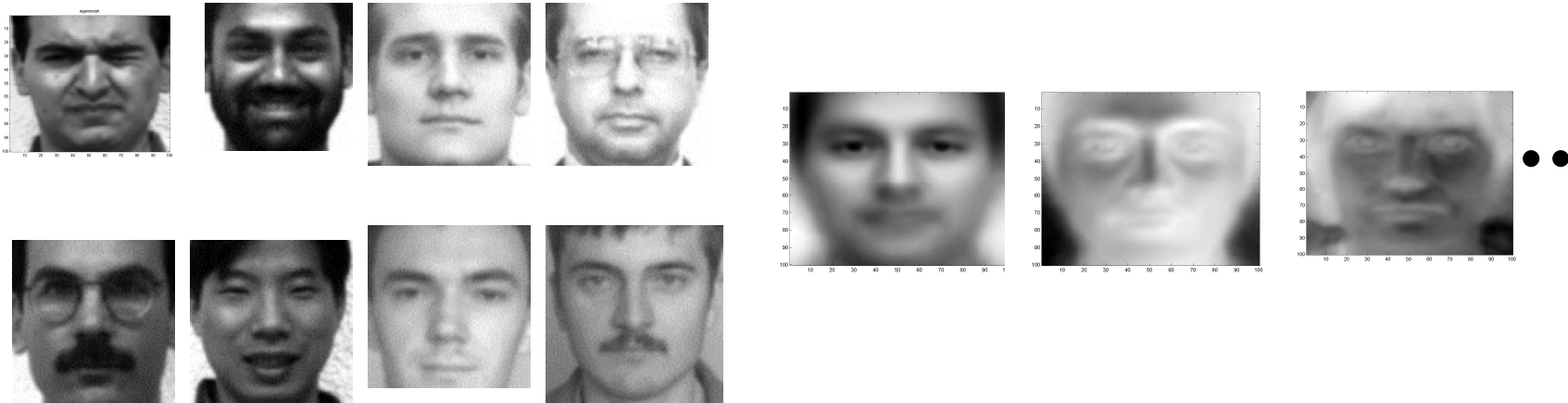


The typical face



- A “typical face” that captures the essence of “facehood”..
- The principal Eigen face..

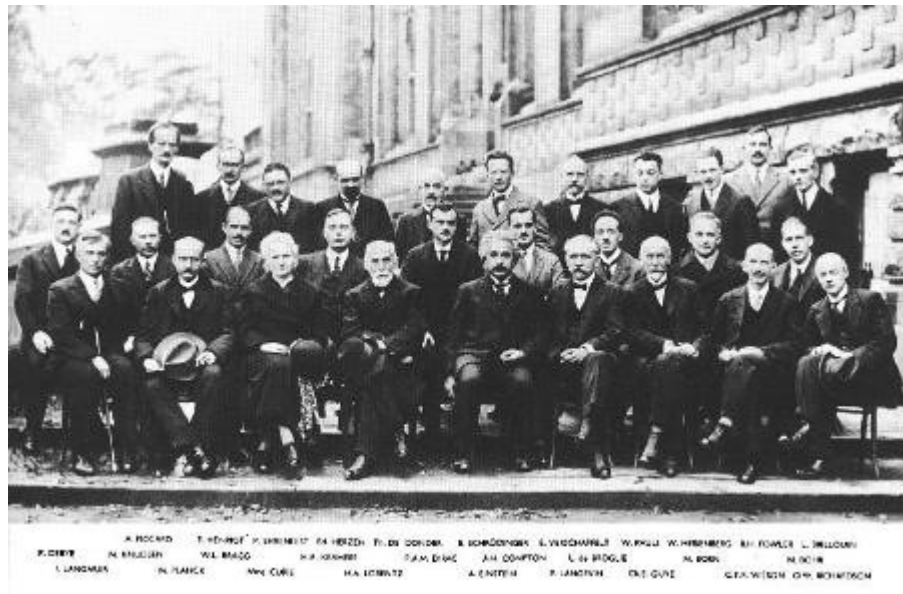
A collection of least squares typical faces



- Extension: Many Eigenfaces
- Approximate **every** face f as $f = w_{f,1} V_1 + w_{f,2} V_2 + \dots + w_{f,k} V_k$
 - V_2 is used to “correct” errors resulting from using only V_1
 - V_3 corrects errors remaining after correction with V_2
 - And so on..
- $V = [V_1 V_2 V_3]$ can be computed through Eigen analysis

Detecting Faces in Images

Detecting Faces in Images



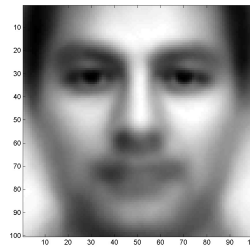
- Finding face like patterns
 - How do we find if a picture has faces in it
 - Where are the faces?
- A simple solution:
 - Define a “typical face”
 - Find the “typical face” in the image

Given an image and a 'typical' face how do I find the faces?



400 × 200
(RGB)

+



100 × 100

+

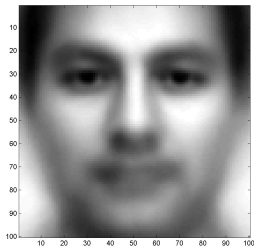


Finding faces in an image



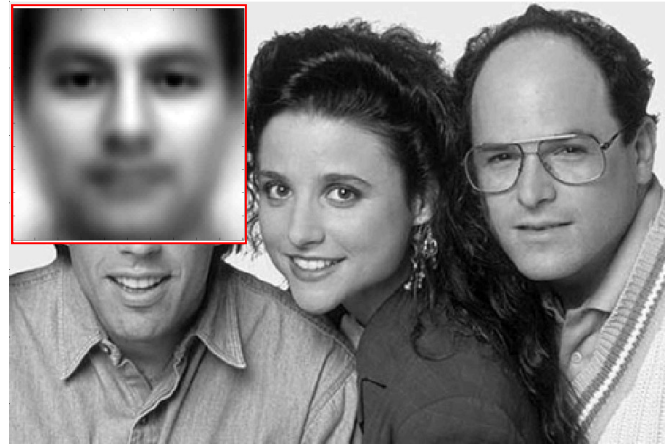
- Picture is larger than the “typical face”
 - E.g. typical face is 100x100, picture is 600x800
- First convert to greyscale
 - $R + G + B$
 - Not very useful to work in color

Finding faces in an image



- Goal .. To find out if and where images that look like the “typical” face occur in the picture

Finding faces in an image



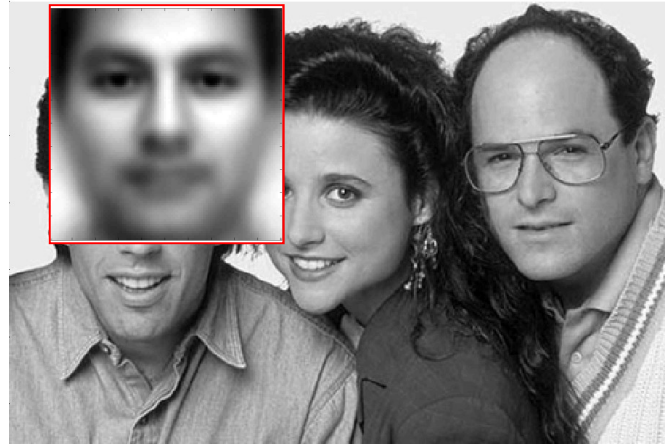
- Try to “match” the typical face to each location in the picture

Finding faces in an image



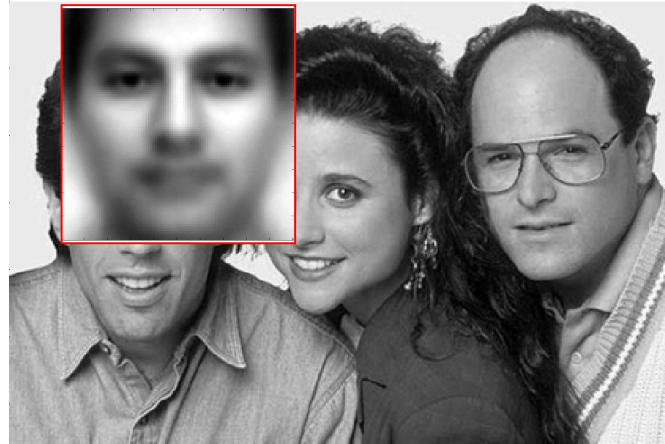
- Try to “match” the typical face to each location in the picture

Finding faces in an image



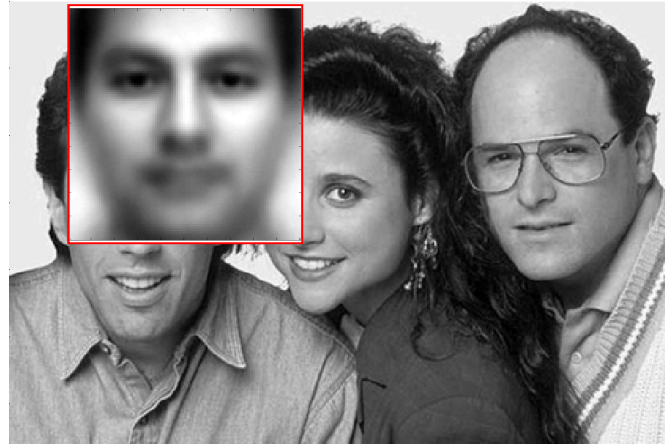
- Try to “match” the typical face to each location in the picture

Finding faces in an image



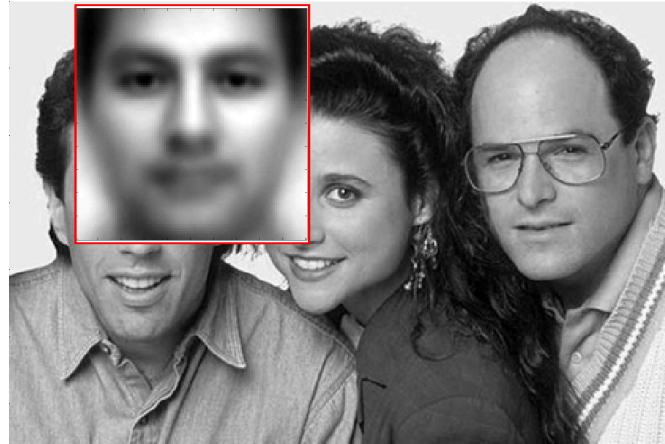
- Try to “match” the typical face to each location in the picture

Finding faces in an image



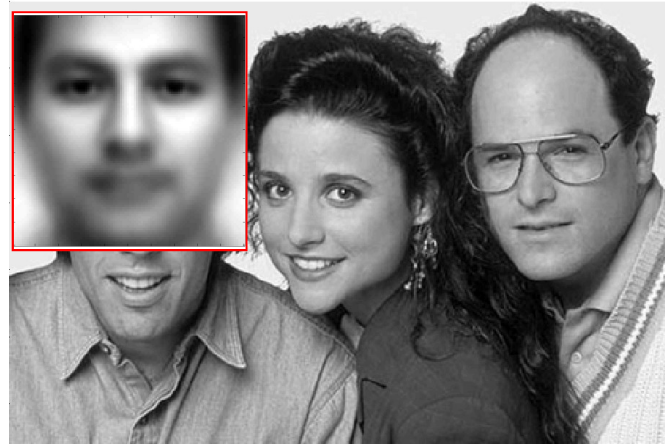
- Try to “match” the typical face to each location in the picture

Finding faces in an image



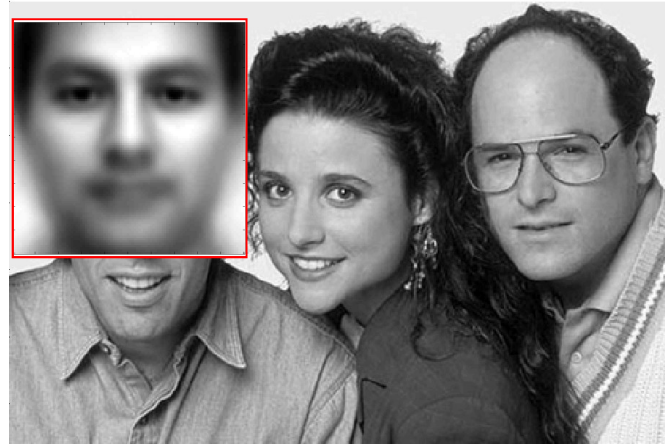
- Try to “match” the typical face to each location in the picture

Finding faces in an image



- Try to “match” the typical face to each location in the picture

Finding faces in an image



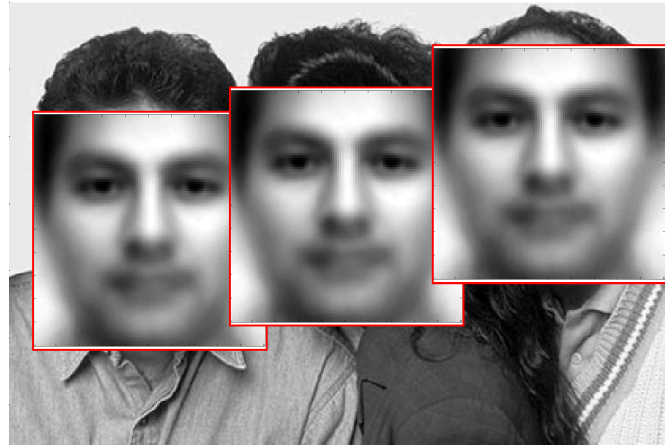
- Try to “match” the typical face to each location in the picture

Finding faces in an image



- Try to “match” the typical face to each location in the picture

Finding faces in an image



- Try to “match” the typical face to each location in the picture
- The “typical face” will explain some spots on the image much better than others
 - These are the spots at which we probably have a face!

How to “match”



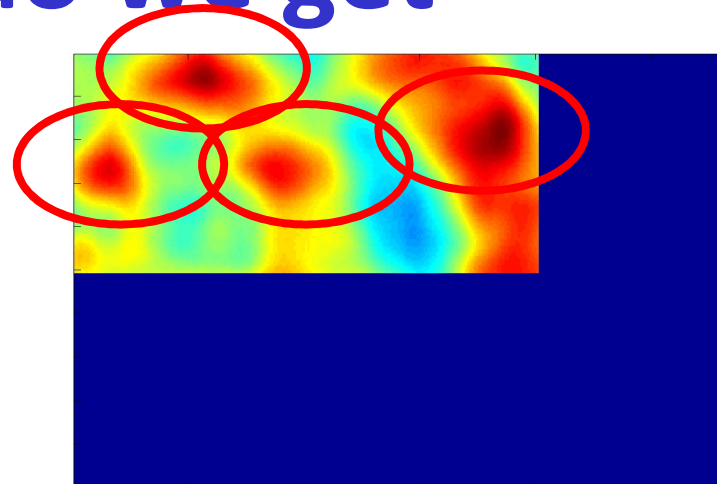
- What exactly is the “match”
 - What is the match “score”

How to “match”



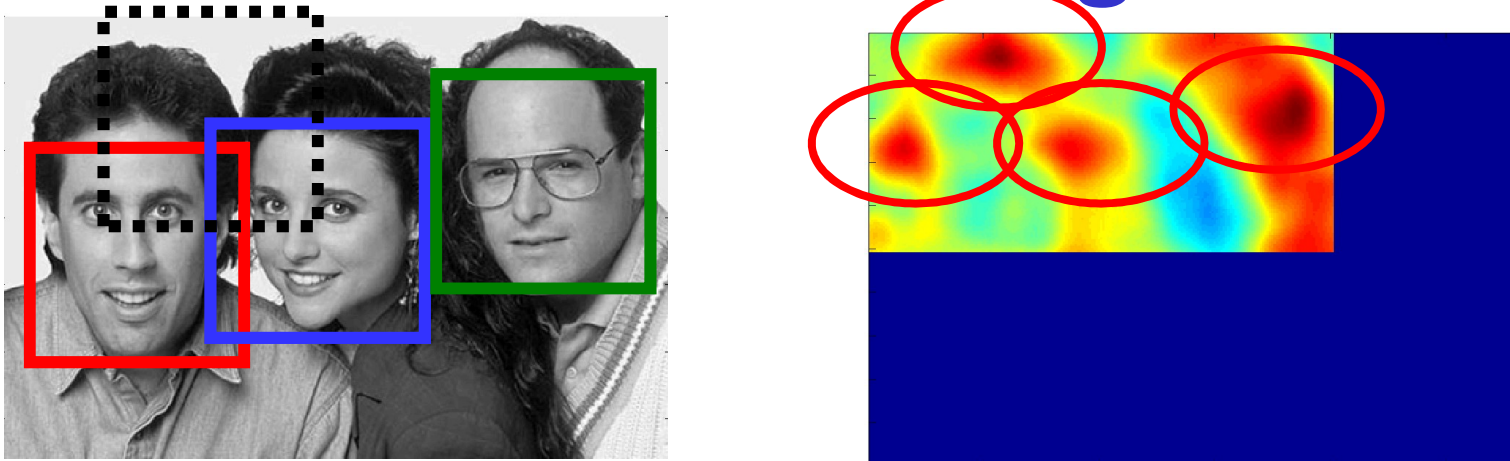
- What exactly is the “match”
 - What is the match “score”
- The DOT Product
 - Express the typical face as a vector
 - Express the region of the image being evaluated as a vector
 - Compute the dot product of the typical face vector and the “region” vector

What do we get



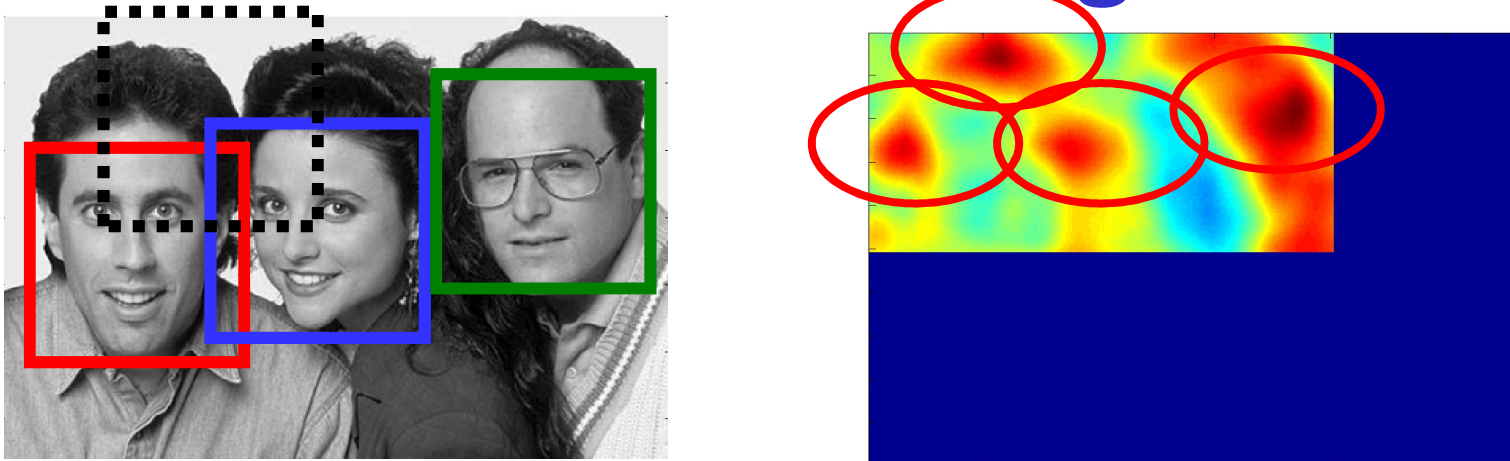
- The right panel shows the dot product at various locations
 - Redder is higher
 - The locations of peaks indicate locations of faces!

What do we get



- The right panel shows the dot product at various locations
 - Redder is higher
 - The locations of peaks indicate locations of faces!
- Correctly detects all three faces
 - Likes George's face most
 - He looks most like the typical face
- Also finds a face where there is none!
 - A false alarm

What do we get



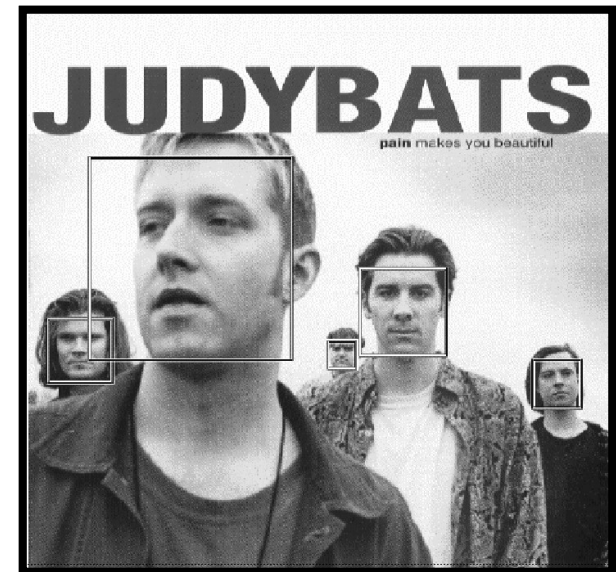
- The right panel shows the dot product at various locations
 - Redder is higher
 - The locations of peaks indicate locations of faces!
- Correctly detects all three faces
 - Likes George's face most
 - He looks most like the typical face
- Also finds a face where there is none!
 - A false alarm

11755/18979



Sliding windows solves only the issue of location – what about scale?

- Not all faces are the same size
- Some people have bigger faces
- The size of the face on the image changes with perspective
- Our “typical face” only represents one of these sizes



Scale-Space Pyramid



Scale the image
(but keep your typical
face template fixed)

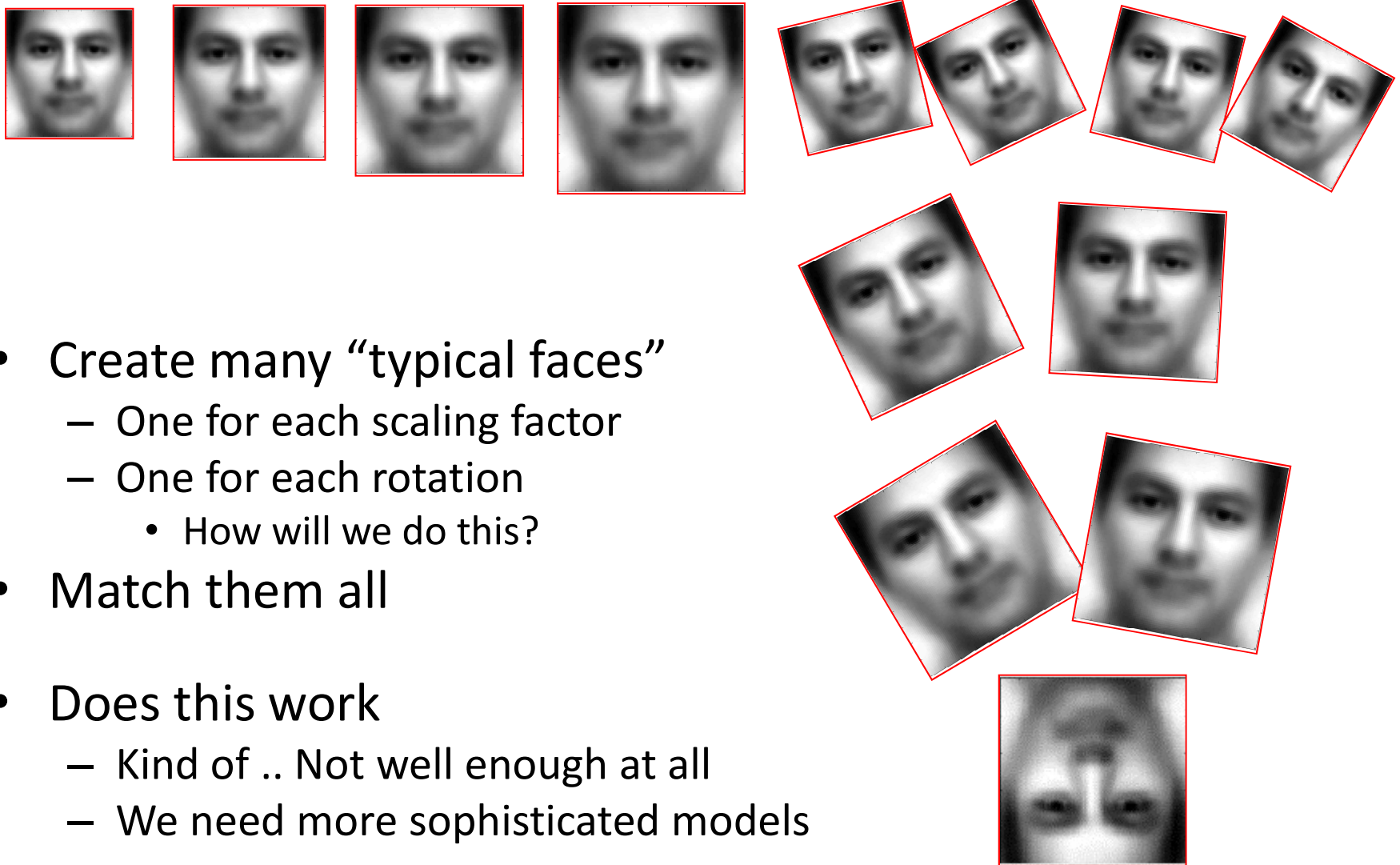
Figure 1.4: The Scale-Space Pyramid. The detector is run using the sliding windows approach over the input image at various scales. When the scale of the person matches the detector scale the classifier will (hopefully) fire yielding an accurate detection.

Location – Scale – What about Rotation?

- The head need not always be upright!
 - Our typical face image was upright



Solution



- Create many “typical faces”
 - One for each scaling factor
 - One for each rotation
 - How will we do this?
- Match them all
- Does this work
 - Kind of .. Not well enough at all
 - We need more sophisticated models

Face Detection: A Quick Historical Perspective

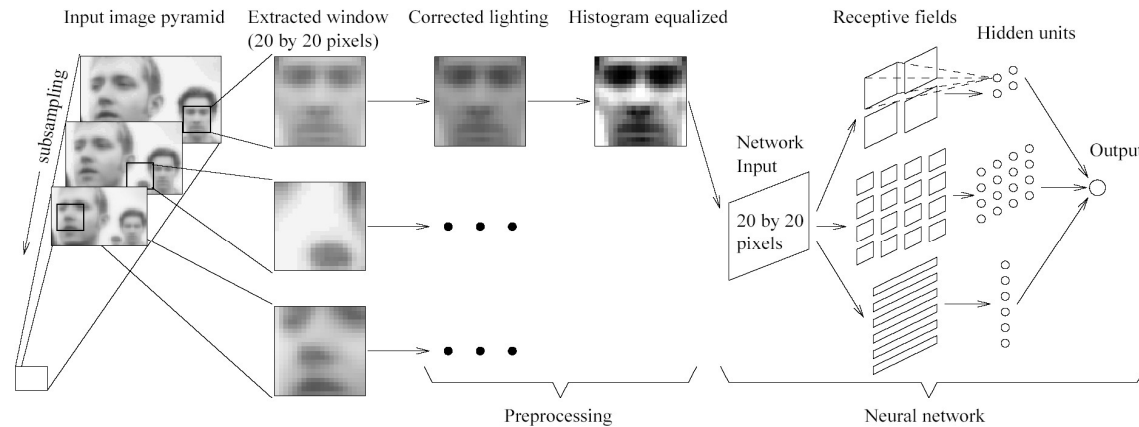


Figure 1: The basic algorithm used for face detection.

- Many more complex methods
 - Use edge detectors and search for face like patterns
 - Find “feature” detectors (noses, ears..) and employ them in complex neural networks..
- The Viola Jones method
 - Boosted cascaded classifiers

Face Detection: A Quick Historical Perspective

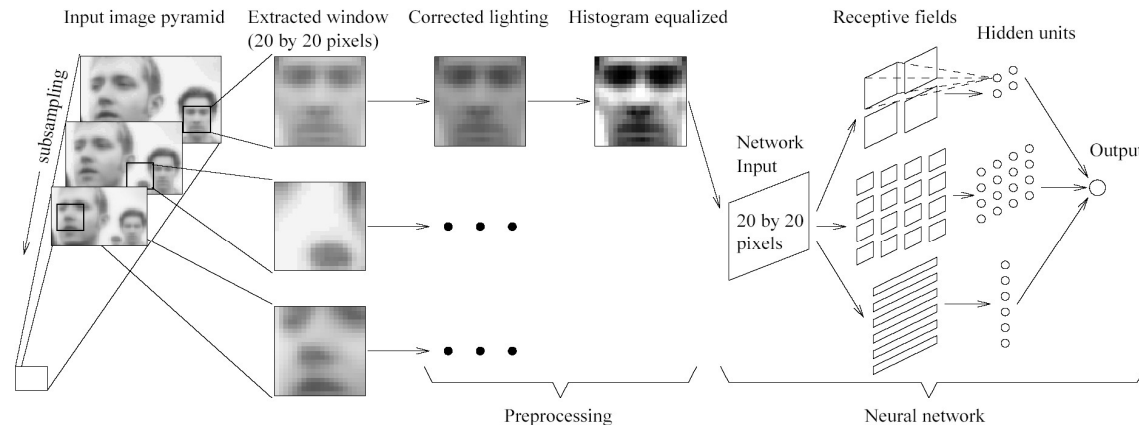


Figure 1: The basic algorithm used for face detection.

- Many more complex methods
 - Use edge detectors and search for face like patterns
 - Find “feature” detectors (noses, ears..) and employ them in complex neural networks..
- **The Viola Jones method (25K+ Citations!)**
 - **Boosted cascaded classifiers**

And even before that – what is classification?

- Given “features” describing an entity, determine the category it belongs to
 - Walks on two legs, has no hair. Is this
 - A Chimpanzee
 - A Human
 - Has long hair, is 5’6” tall, is this
 - A man
 - A woman
 - Matches “eye” pattern with score 0.5, “mouth pattern” with score 0.25, “nose” pattern with score 0.1. Are we looking at
 - A face
 - Not a face?

Classification

- Multi-class classification
 - Many possible categories
 - E.g. Sounds “AH, IY, UW, EY..”
 - E.g. Images “Tree, dog, house, person..”
- Binary classification
 - Only two categories
 - Man vs. Woman
 - Face vs. **not a face...**

Detection vs Classification

- Detection: Find an X
- Classification: Find the correct label X,Y,Z etc.

Detection vs Classification

- Detection: Find an X
- Classification: Find the correct label X,Y,Z etc.
- Binary Classification as Detection: Find the correct label X or *not-X*

Face Detection as Classification



For each square, run a classifier to find out if it is a face or not

- Faces can be many sizes
- They can happen anywhere in the image
- For each face size
 - For each location
 - Classify a rectangular region of the face size, at that location, as a face or not a face
- This is a series of **binary** classification problems
 - We can use the adaboost algorithm

Training Data




ID	E1	E2.	Class
A	0.3	-0.6	+1
B	0.5	-0.5	+1
C	0.7	-0.1	+1
D	0.6	-0.4	+1
E	0.2	0.4	-1
F	-0.8	-0.1	-1
G	0.4	-0.9	-1
H	0.2	0.5	-1

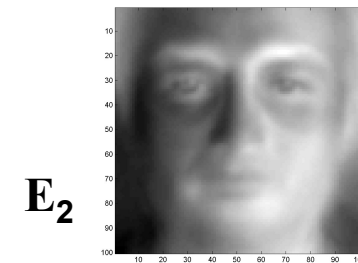
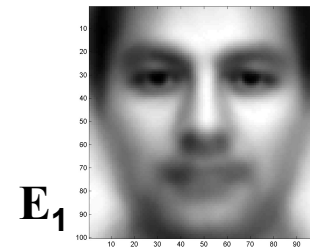
Face = +1
Non-face = -1

The ADABOOST Algorithm

- Initialize $D_1(x_i) = 1/N$
- For $t = 1, \dots, T$
 - Train a weak classifier h_t using distribution D_t
 - Compute total error on training data
 - $\varepsilon_t = \text{Sum} \{D_t(x_i) \frac{1}{2}(1 - y_i h_t(x_i))\}$
 - Set $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$
 - For $i = 1 \dots N$
 - set $D_{t+1}(x_i) = D_t(x_i) \exp(-\alpha_t y_i h_t(x_i))$
 - Normalize D_{t+1} to make it a distribution
- The final classifier is
 - $H(x) = \text{sign}(\sum_t \alpha_t h_t(x))$

ADA Boost


$$= 0.4 E_1 - 0.4 E_2$$



- The final classifier is
 - $H(x) = \text{sign}(\sum_t \alpha_t h_t(x))$
- The output is 1 if the total weight of all weak learners that classify x as 1 is greater than the total weight of all weak learners that classify it as -1

Boosting and Face Detection

- Boosting is the basis of one of the most popular methods for face detection: The Viola-Jones algorithm
 - Current methods use other classifiers like SVMs and neural nets, but Adaboost classifiers remain easy to implement and popular
 - OpenCV implements Viola Jones..

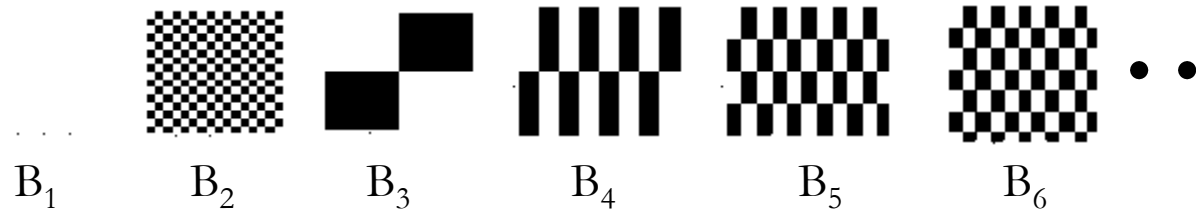
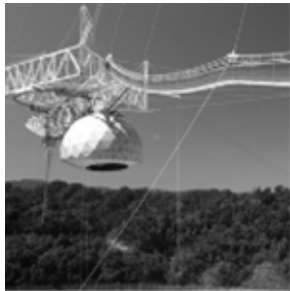
The problem of face detection

- 1. Defining Features
 - Should we be searching for noses, eyes, eyebrows etc.?
 - Nice, but expensive
 - Or something simpler
- 2. Selecting Features
 - Of all the possible features we can think of, which ones make sense
- 3. Classification: Combining evidence
 - How does one combine the evidence from the different features?

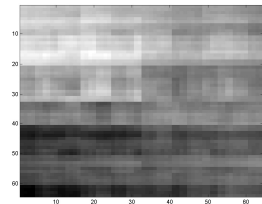
Feature requirements

- Must be sufficiently descriptive
 - So that we achieve good classification
- Must be *extremely* inexpensive to compute
 - Face detection typically required to be performed in realtime on very cheap devices
- Preferably computable using only integer operations
 - Use less power

Features: The Viola Jones Method



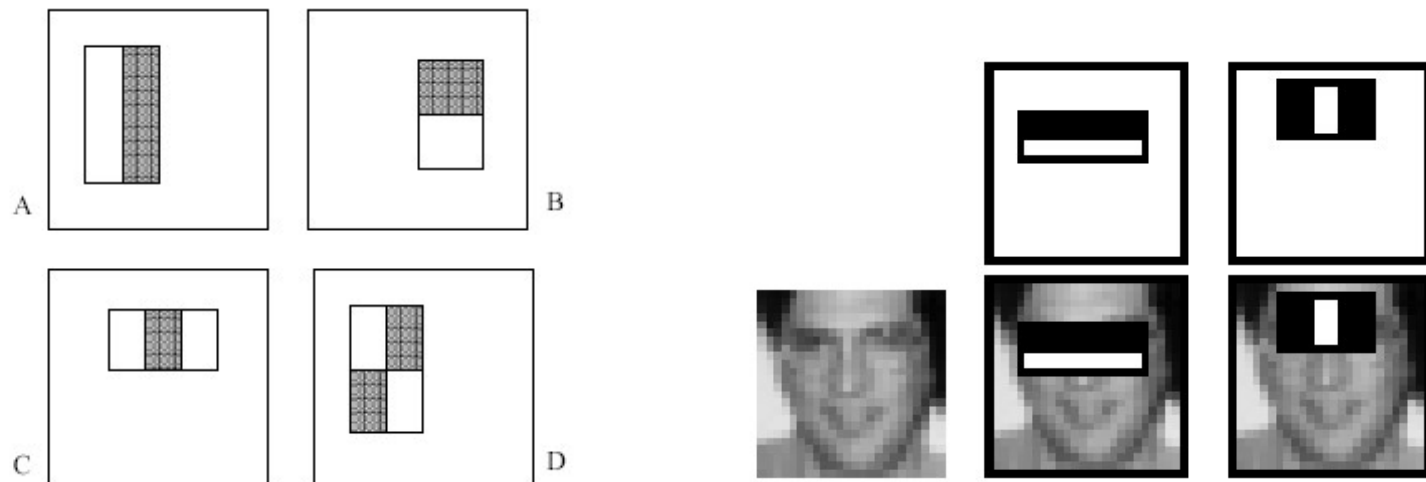
$$\text{Image} \approx w_1 B_1 + w_2 B_2 + w_3 B_3 + \dots$$



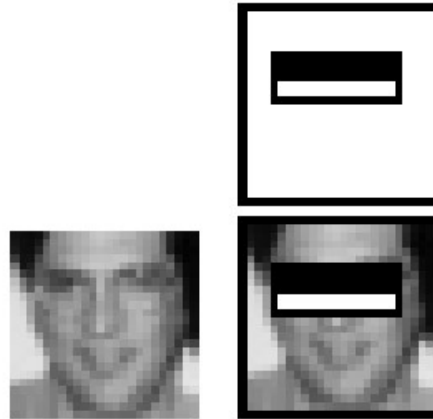
- Integral Features!!
 - Like the Checkerboard
- The same principle as we used to decompose images in terms of checkerboards:
 - The image of any object has changes at various scales
 - These can be represented coarsely by a checkerboard pattern
- The checkerboard patterns must however now be *localized*
 - Stay within the region of the face

Features

- Checkerboard Patterns to represent facial features
 - The white areas are subtracted from the black ones.
 - Each checkerboard explains a *localized* portion of the image
- Four types of checkerboard patterns (only)



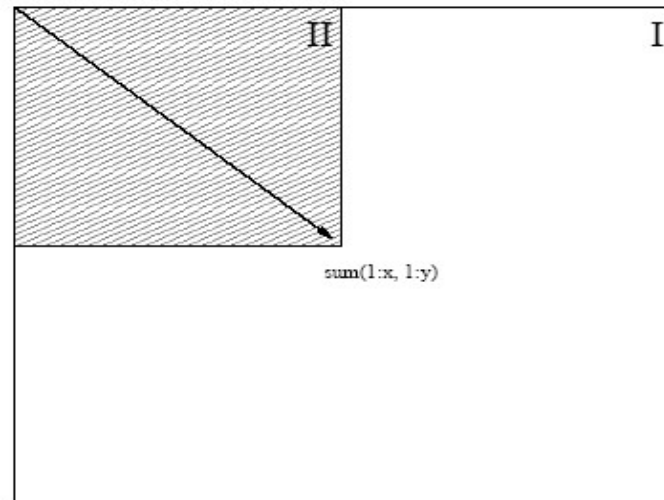
Explaining a portion of the face with a checker..



- How much is the difference in average intensity of the image in the black and white regions
 - $\text{Sum}(\text{pixel values in white region}) - \text{Sum}(\text{pixel values in black region})$
- This is actually the dot product of the region of the face covered by the rectangle and the checkered pattern itself
 - White = 1, Black = -1

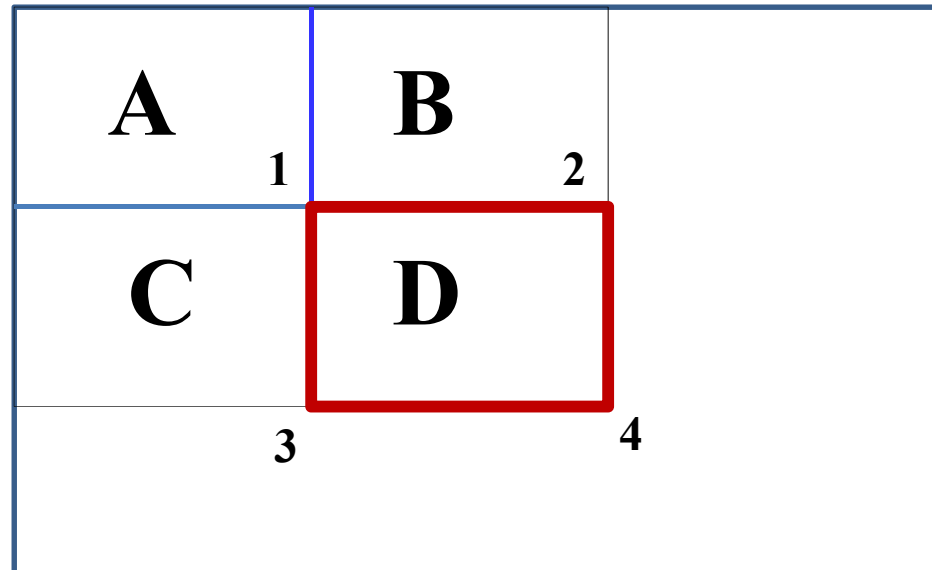
Integral images

- Summed area tables



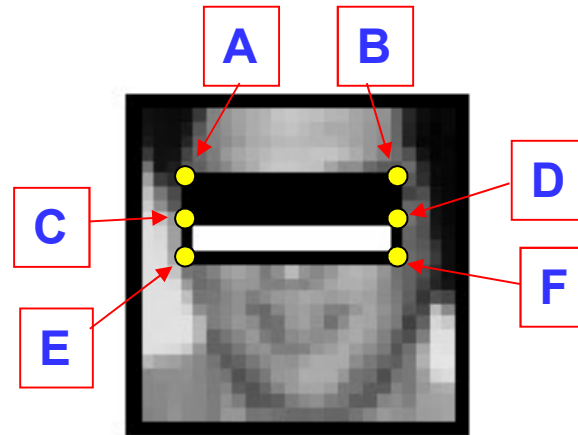
- For each pixel store the sum of ALL pixels to the left of and above it.

Fast Computation of Pixel Sums



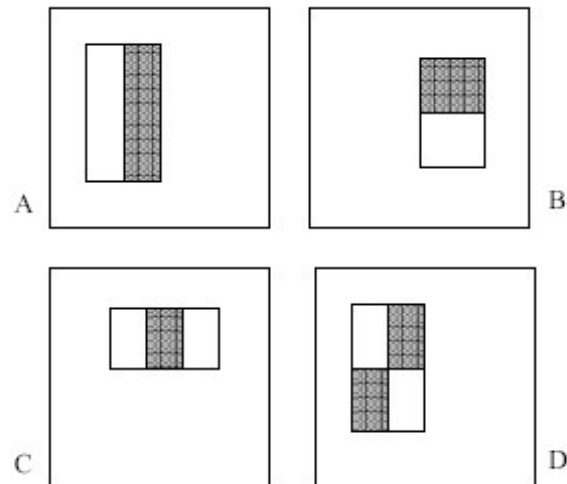
- To compute the sum of the pixels within “D”:
 - $\text{Pixelsum}(1) = \text{Area}(A)$
 - $\text{Pixelsum}(2) = \text{Area}(A) + \text{Area}(B)$
 - $\text{Pixelsum}(3) = \text{Area}(A) + \text{Area}(C)$
 - $\text{Pixelsum}(4) = \text{Area}(A) + \text{Area}(B) + \text{Area}(C) + \text{Area}(D)$
- $\text{Area}(D) = \text{Pixelsum}(4) - \text{Pixelsum}(2) - \text{Pixelsum}(3) + \text{Pixelsum}(1)$

A Fast Way to Compute the Feature



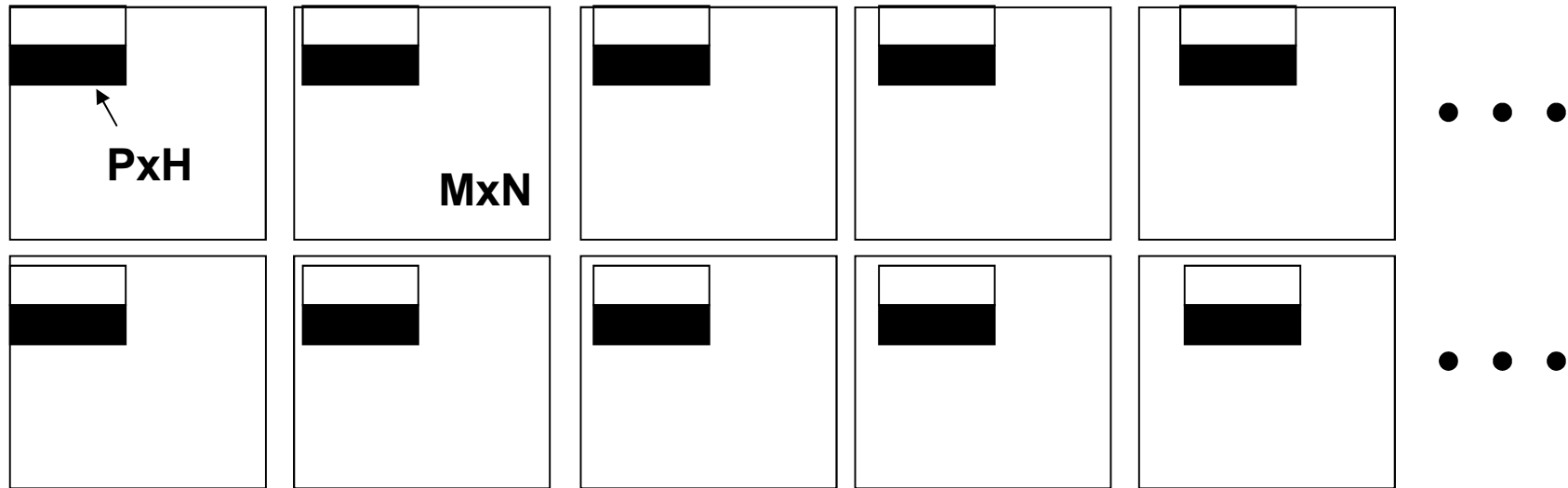
- Store pixel table for every pixel in the image
 - The sum of all pixel values to the left of and above the pixel
- Let A, B, C, D, E, F be the pixel table values at the locations shown
 - Total pixel value of black area = $D + A - B - C$
 - Total pixel value of white area = $F + C - D - E$
 - Feature value = $(F + C - D - E) - (D + A - B - C)$

“Integral” features



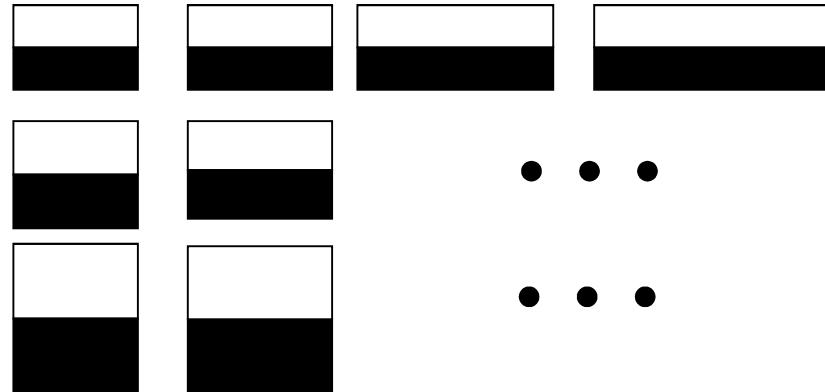
- Each checkerboard has the following characteristics
 - Length
 - Width
 - Type
 - Specifies the number and arrangement of bands
- The four checkerboards above are the four used by Viola and Jones

How many features?



- Each checker board of width P and height H can start at any of $(N-P)(M-H)$ pixels
- $(M-H) * (N-P)$ possible starting locations
 - Each is a unique checker feature
 - E.g. at one location it may measure the forehead, at another the chin

How many features

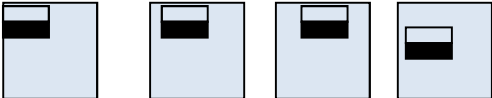





- Each feature can have many sizes
 - Width from (min) to (max) pixels
 - Height from (min ht) to (max ht) pixels
- At each size, there can be many starting locations
 - Total number of possible checkerboards of one type:
No. of possible sizes x No. of possible locations
- There are four types of checkerboards
 - Total no. of possible checkerboards: VERY VERY LARGE!

Learning: No. of features

- Analysis performed on images of 24x24 pixels only
 - Reduces the no. of possible features to about 180000
- Restrict checkerboard size
 - Minimum of 8 pixels wide
 - Minimum of 8 pixels high
 - Other limits, e.g. 4 pixels may be used too
 - Reduces no. of checkerboards to about 50000

No. of features

	F1	F2	F3	F4	F180000
	7	9	2	-1	12
	-11	3	19	17	2

- Each possible checkerboard gives us one feature
- A total of up to 180000 features derived from a 24x24 image!
- Every 24x24 image is now represented by a set of 180000 numbers
 - This is the set of features we will use for classifying if it is a face or not!

The Classifier

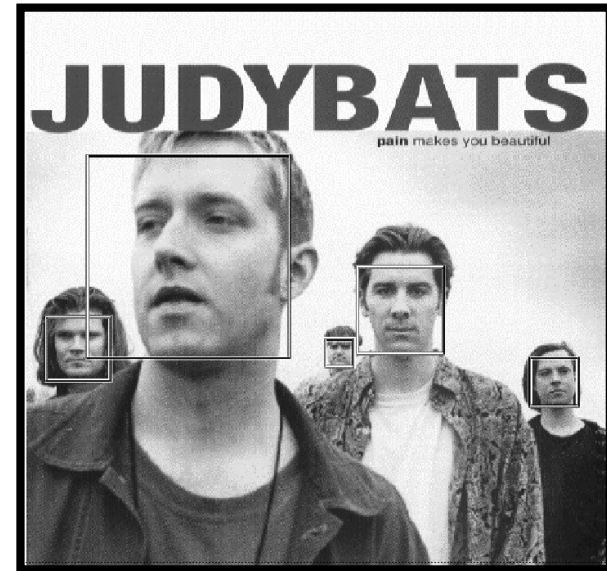
- The Viola-Jones algorithm uses AdaBoost with “stumps”
- At each stage find the best feature to classify the data with
 - I.e the feature that gives us the best classification of all the training data
 - Training data includes many examples of faces and non-face images
 - The classification rule is of the kind
 - If feature $>$ threshold, face (or if feature $<$ threshold, face)
 - The optimal value of “threshold” must also be determined.

To Train

- Collect a large number of facial images
 - Resize all of them to 24x24
 - These are our “face” training set
- Collect a much much much larger set of 24x24 non-face images of all kinds
 - Each of them is
 - These are our “non-face” training set
- Train a boosted classifier

The Viola Jones Classifier

- During tests:
 - Given any new 24x24 image
 - $R = \sum_f \alpha_f (f > p_f \theta(f))$
 - Only a small number of features ($f < 100$) typically used
- **Problems:**
 - Only classifies 24 x 24 images entirely as faces or non-faces
 - Pictures are typically much larger
 - Pictures may contain many faces
 - Faces in pictures can be much larger or smaller
 - Not accurate enough
 - Too slow



Multiple faces in the picture



- Scan the image
 - Classify each 24x24 rectangle from the photo
 - All rectangles that get classified as having a face indicate the location of a face
- For an NxM picture, we will perform $(N-24)*(M-24)$ classifications
- If overlapping 24x24 rectangles are found to have faces, merge them

Multiple faces in the picture



- Scan the image
 - Classify each 24x24 rectangle from the photo
 - All rectangles that get classified as having a face indicate the location of a face
- For an NxM picture, we will perform $(N-24)*(M-24)$ classifications
- If overlapping 24x24 rectangles are found to have faces, merge them

Multiple faces in the picture



- Scan the image
 - Classify each 24x24 rectangle from the photo
 - All rectangles that get classified as having a face indicate the location of a face
- For an NxM picture, we will perform $(N-24)*(M-24)$ classifications
- If overlapping 24x24 rectangles are found to have faces, merge them

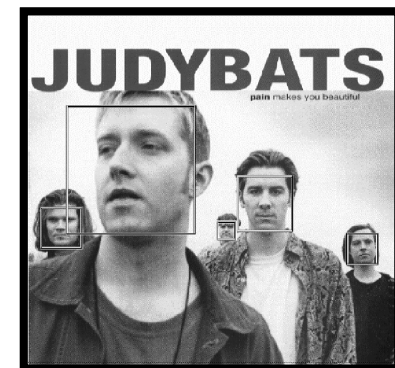
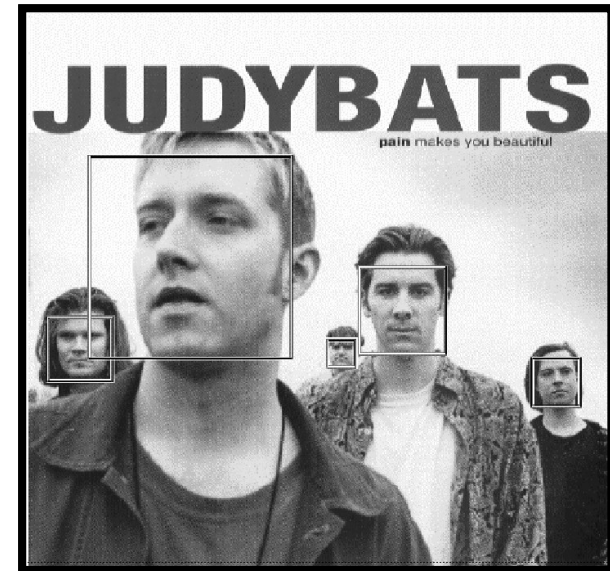
Multiple faces in the picture



- Scan the image
 - Classify each 24x24 rectangle from the photo
 - All rectangles that get classified as having a face indicate the location of a face
- For an NxM picture, we will perform $(N-24)*(M-24)$ classifications
- If overlapping 24x24 rectangles are found to have faces, merge them

Picture size solution

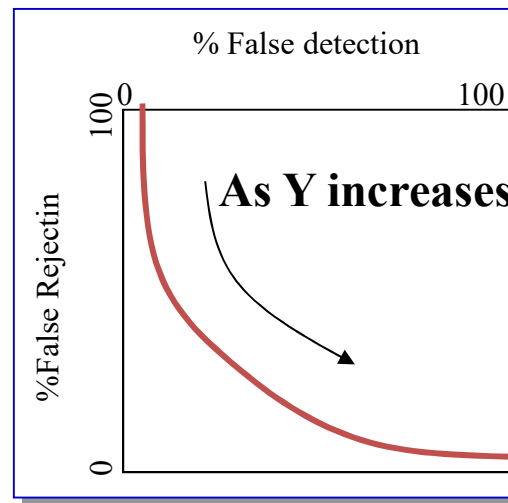
- We already have a classifier
 - That uses weak learners
- *Scale the Picture*
 - Scale the picture down by a factor a
 - Keep decrementing down to a minimum reasonable size



False Rejection vs. False Detection

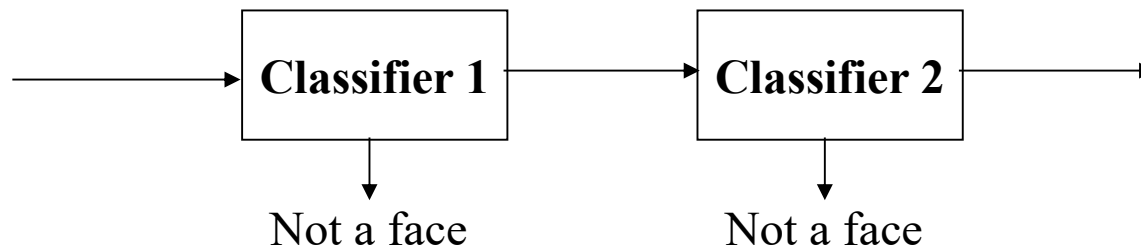
- False Rejection: There's a face in the image, but the classifier misses it
 - Rejects the hypothesis that there's a face
- False detection: Recognizes a face when there is none.
- Classifier:
 - Standard boosted classifier: $H(x) = \text{sign}(\sum_t \alpha_t h_t(x))$
 - Modified classifier $H(x) = \text{sign}(\sum_t \alpha_t h_t(x) + Y)$
 - $\sum_t \alpha_t h_t(x)$ is a measure of certainty
 - The higher it is, the more certain we are that we found a face
 - If Y is large, then we assume the presence of a face even when we are not sure
 - By increasing Y , we can reduce false rejection, while increasing false detection

ROC



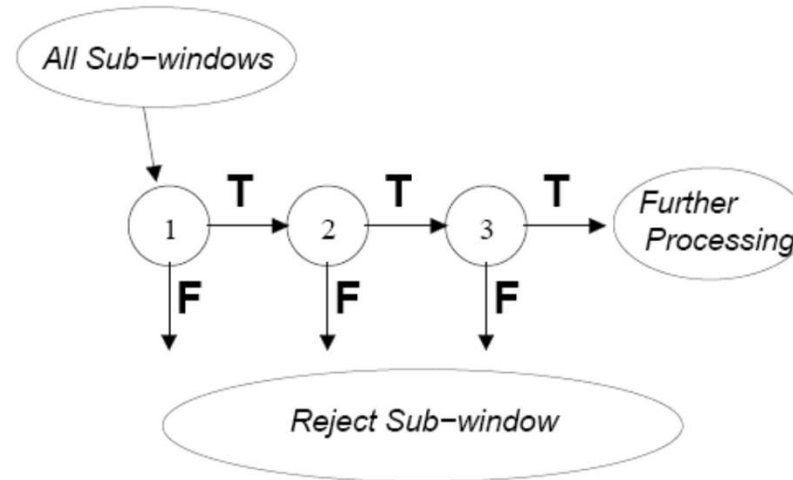
- Ideally false rejection will be 0%, false detection will also be 0%
- As Y increases, we reject faces less and less
 - But accept increasing amounts of garbage as faces
- Can set Y so that we rarely miss a face

Problem: Not accurate enough, too slow



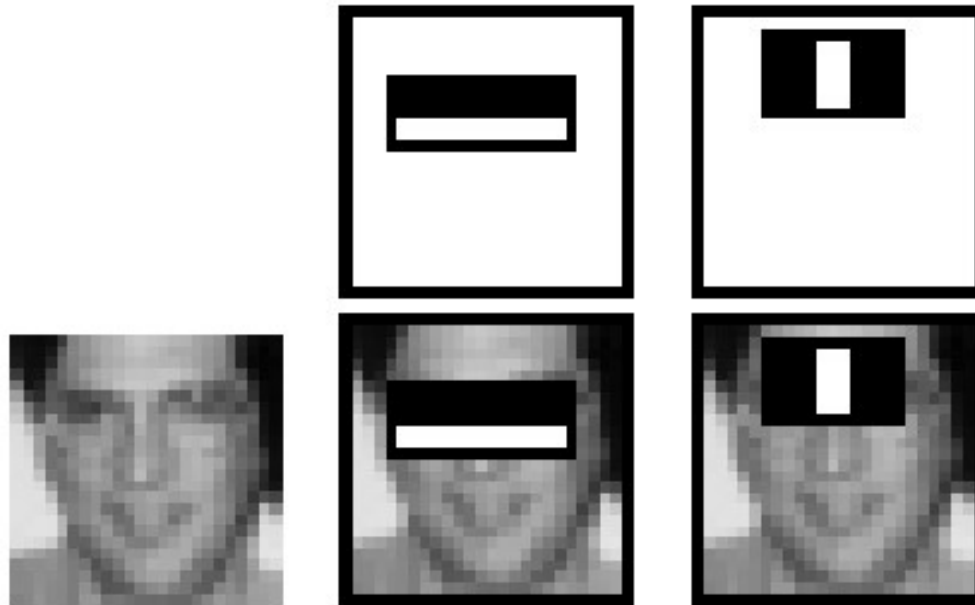
- If we set Y high enough, we will never miss a face
 - But will classify a lot of junk as faces
- Solution: Classify the output of the first classifier with a second classifier
 - And so on.

Problem: Not accurate enough, too slow

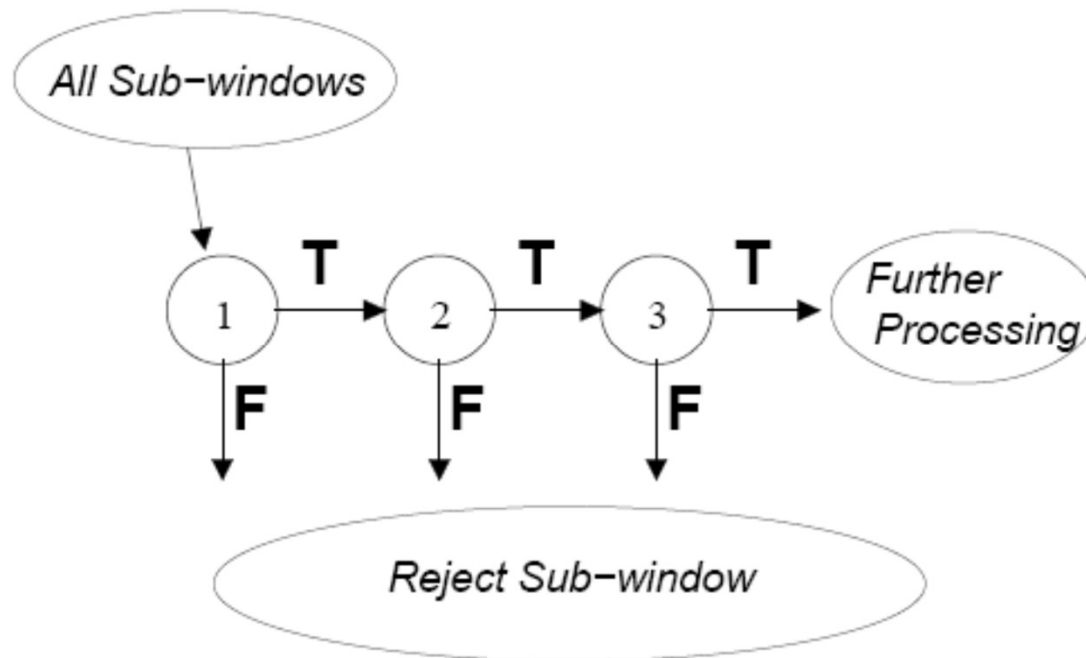


- If we set Y high enough, we will never miss a face
 - But will classify a lot of junk as faces
- Solution: Classify the output of the first classifier with a second classifier
 - And so on.

Useful Features Learned by Boosting



A Cascade of Classifiers



Detection in Real Images

- Basic classifier operates on 24 x 24 subwindows
- Scaling:
 - Scale the detector (rather than the images)
 - Features can easily be evaluated at any scale
 - Scale by factors of 1.25
- Location:
 - Move detector around the image (e.g., 1 pixel increments)
- Final Detections
 - A real face may result in multiple nearby detections
 - Postprocess detected subwindows to combine overlapping detections into a single detection

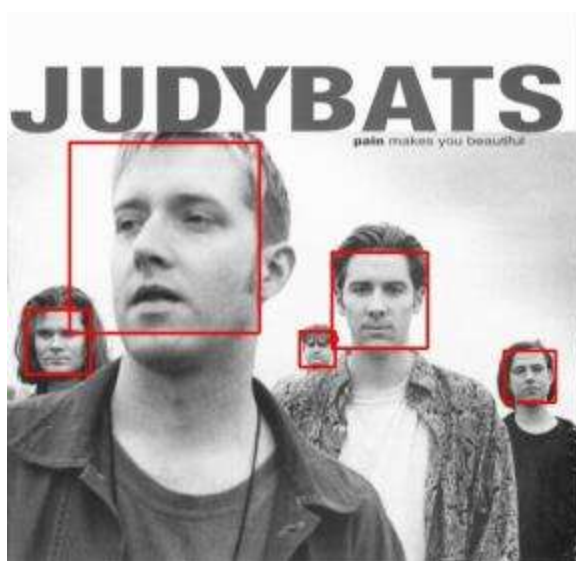
Training

- In paper, 24x24 images of faces and non faces (positive and negative examples).



Sample results using the Viola-Jones Detector

- Notice detection at multiple scales



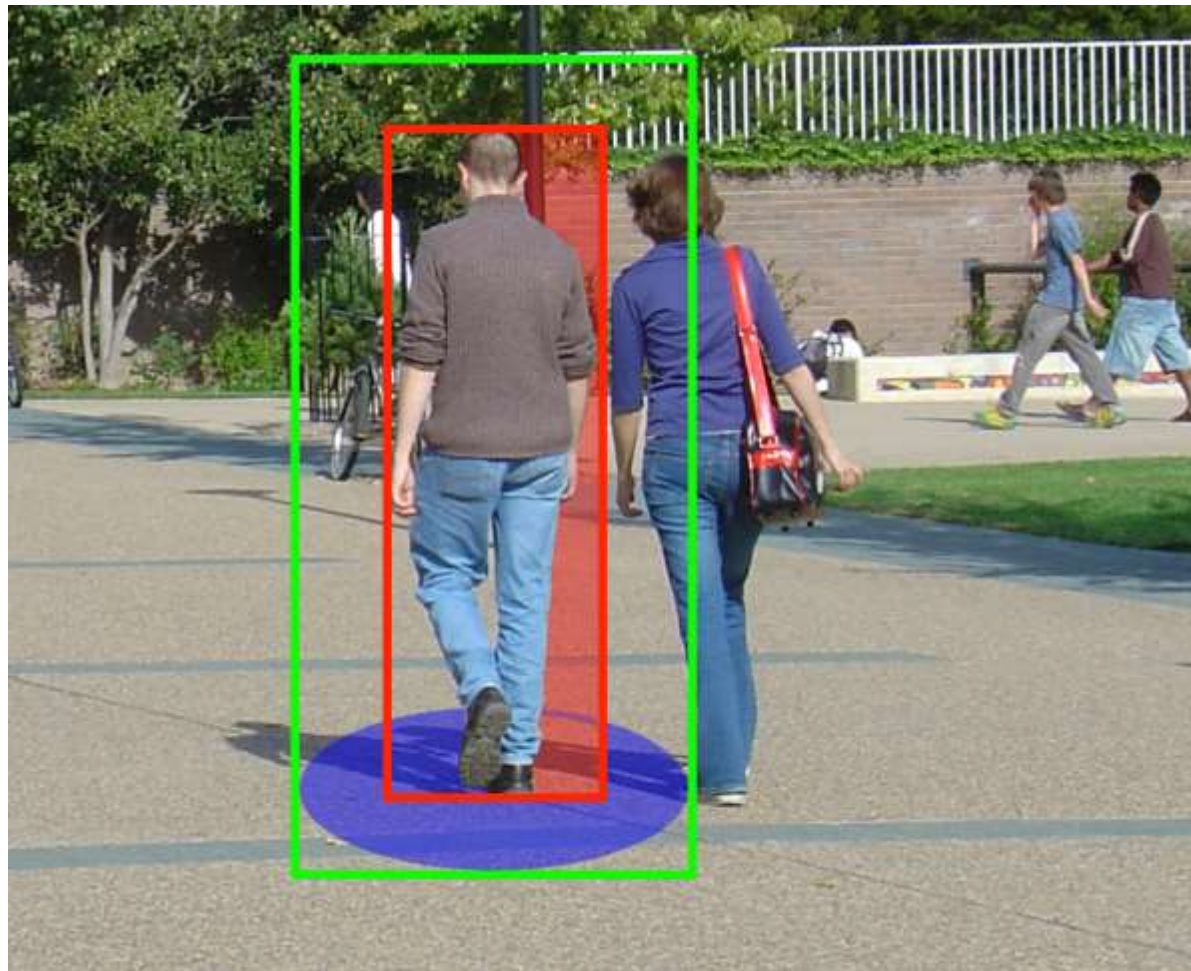
More Detection Examples



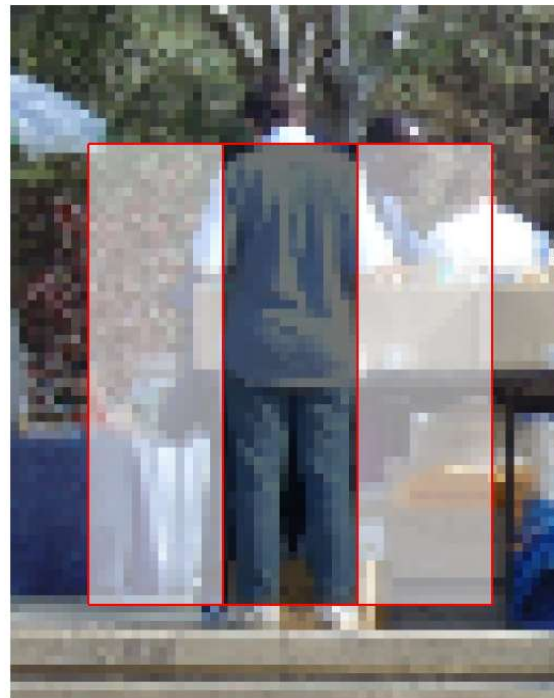
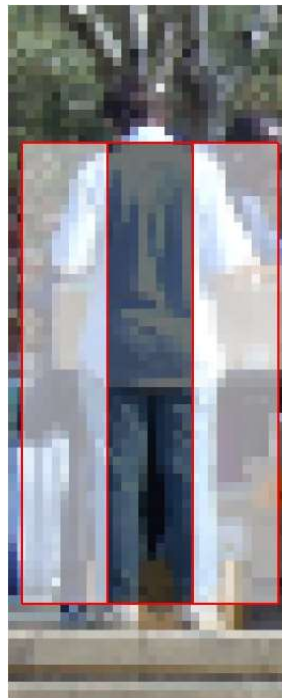
Practical implementation

- Details discussed in Viola-Jones paper
- Training time = weeks (with 5k faces and 9.5k non-faces)
- Final detector has 38 layers in the cascade, 6060 features
- 700 Mhz processor:
 - Can process a 384 x 288 image in 0.067 seconds (in 2003 when paper was written)

Best Window/Background Issues



Best Window/Background Issues



Best Window/Background Issues



Key Ideas

- EigenFace feature
- Sliding windows & scale-space pyramid
- Boosting an ensemble of weak classifiers
- Integral Image / Haar Features
- Cascaded Strong Classifiers