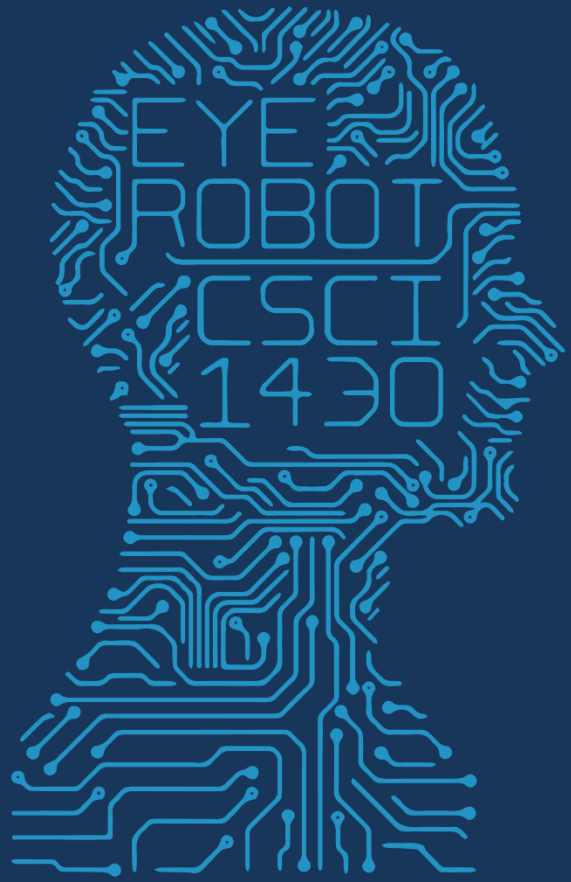
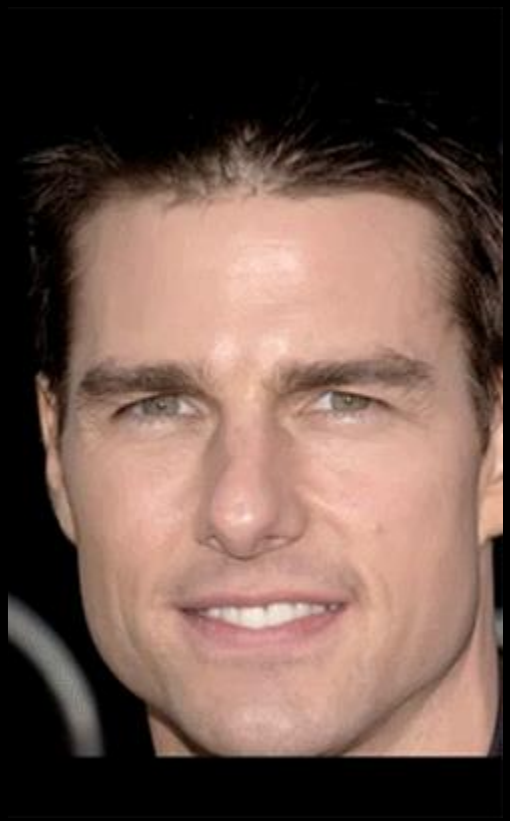
1950

Future Vision



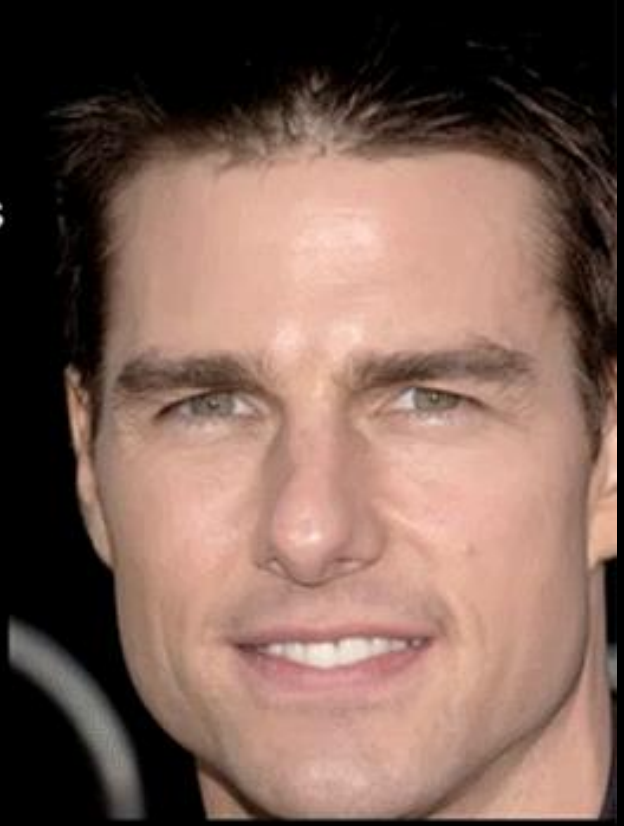
2020

Computer Vision



“Flashed Face Distortion”
2nd Place in the 8th Annual Best Illusion of the Year Contest , VSS 2012

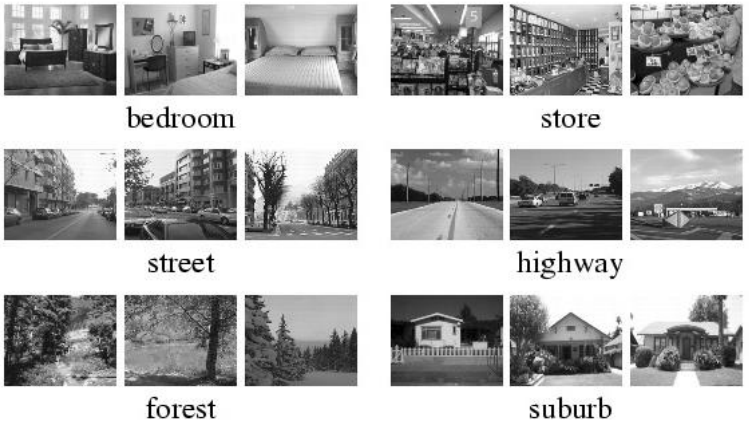
Keep your eyes
on the cross



Recognition so far

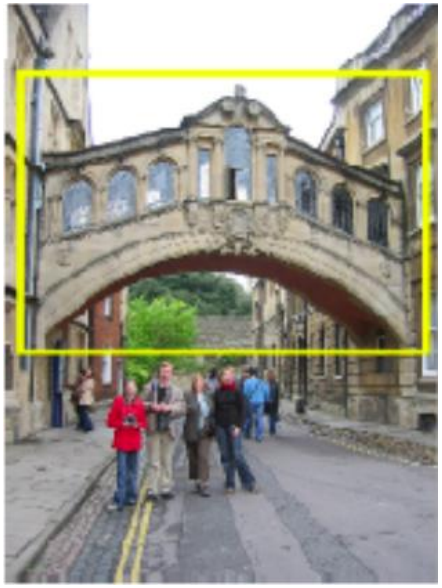
Category:

- Is this a bedroom?
- What class of scene is this?
- Holistic features/quantization



Instance:

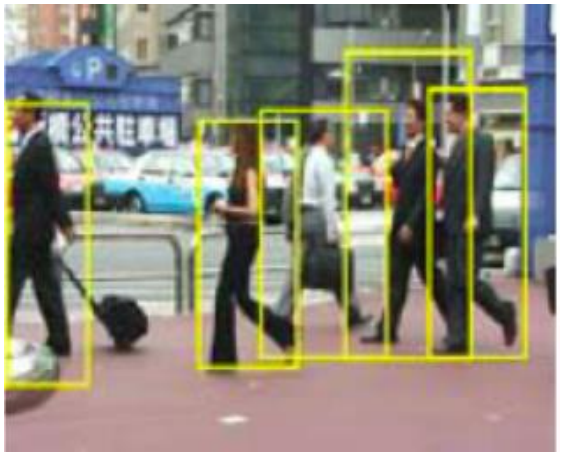
- Find this specific famous building.
- Find this person.
- Local features/precise correspondence
- Often within a database of images



Recognition so far

Object (category) detection:

- Find all the people
- Find all the faces
- Often within a single image
- Often ‘sliding window’



Scenes have “stuff” – distribution of materials and surfaces with arbitrary shape.

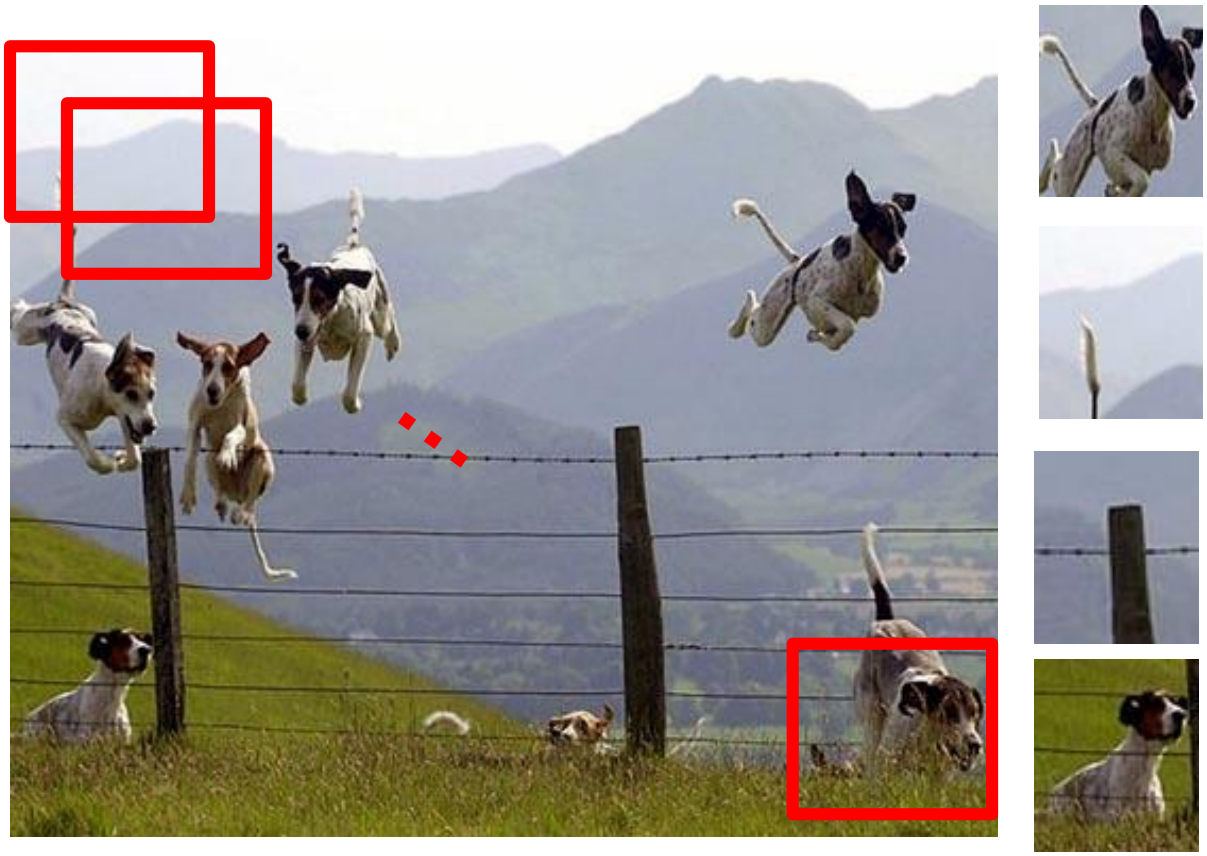
- Bag of Words ok!

Objects are “things” with shape, boundaries.

- Bag of Words less ok as spatial layout is lost!

Object Category Detection

- Focus on object search: “Where is it?”
- Build templates that quickly differentiate object patch from background patch



Object or **Non-Object?**

Challenges in modeling the object class

Illumination

Object pose

'Clutter'

Occlusions

Intra-class appearance

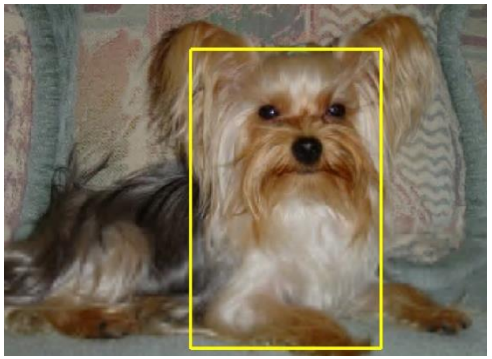
Viewpoint

[K. Grauman, B. Leibe]

Challenges in modeling the non-object class

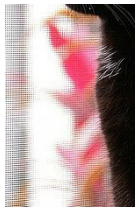
True Detections

Bad Localization



Confused with Similar Object

Misc. Background



Confused with Dissimilar Objects

[Hays]

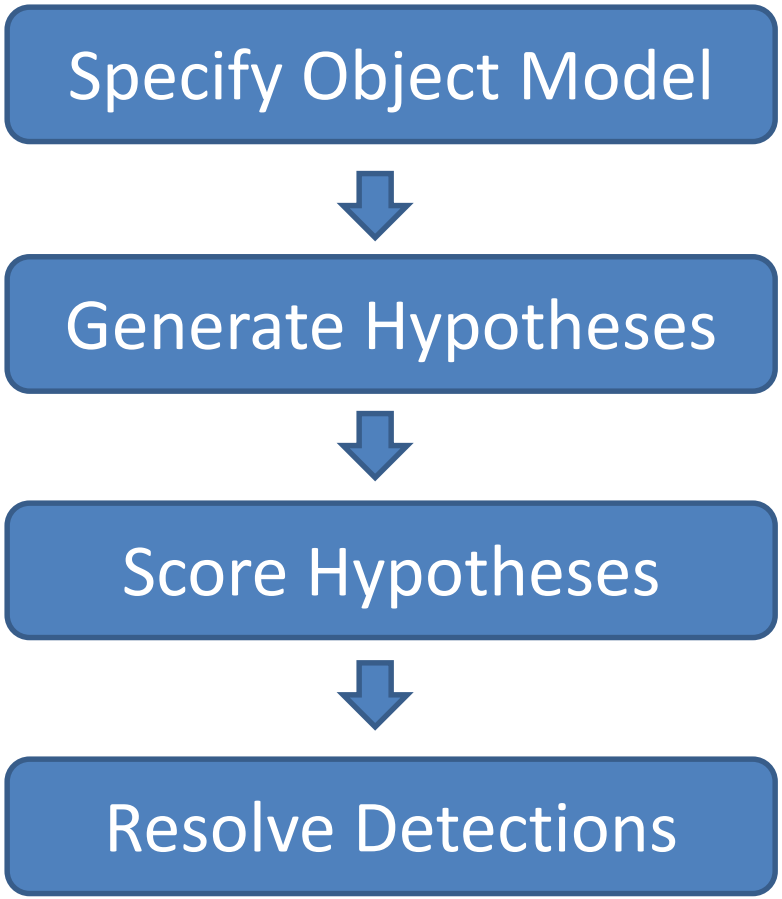
Object Detection Design challenges

- How to efficiently search for likely objects
  - Even simple models require searching hundreds of thousands of positions and scales.

- Feature design and scoring
  - How should appearance be modeled?
    What features correspond to the object?

- How to deal with different viewpoints?
  - Often train different models for a few different viewpoints

General Process of Object Recognition

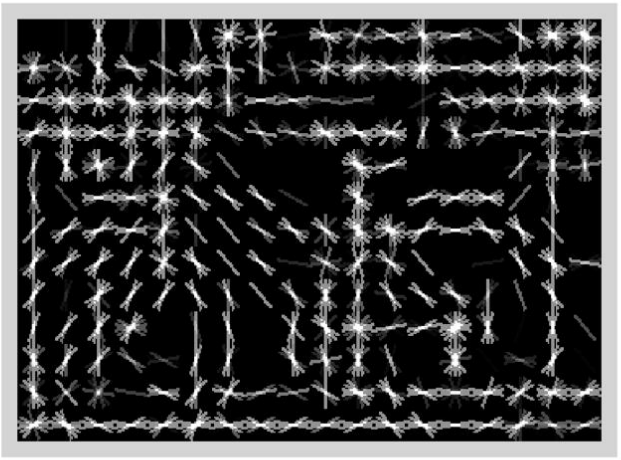


What are the object parameters?

Specifying an object model

1. Statistical Template in Bounding Box
   - Object is some (x,y,w,h) in image
   - Features defined wrt bounding box coordinates

Image

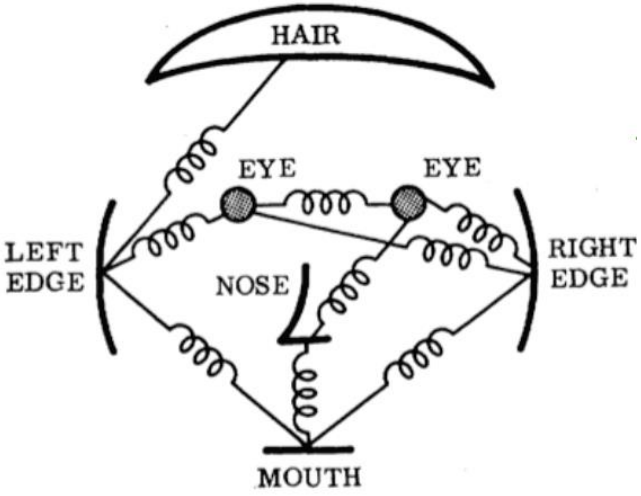


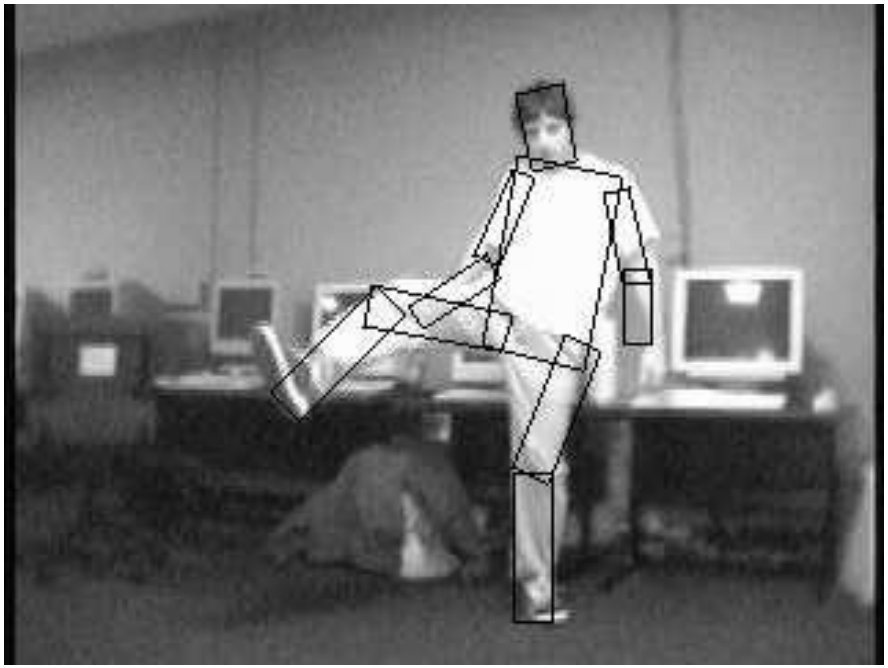
Template Visualization

Images from Felzenszwalb

Specifying an object model

2. Articulated parts model
   - Object is configuration of parts
   - Each part is detectable



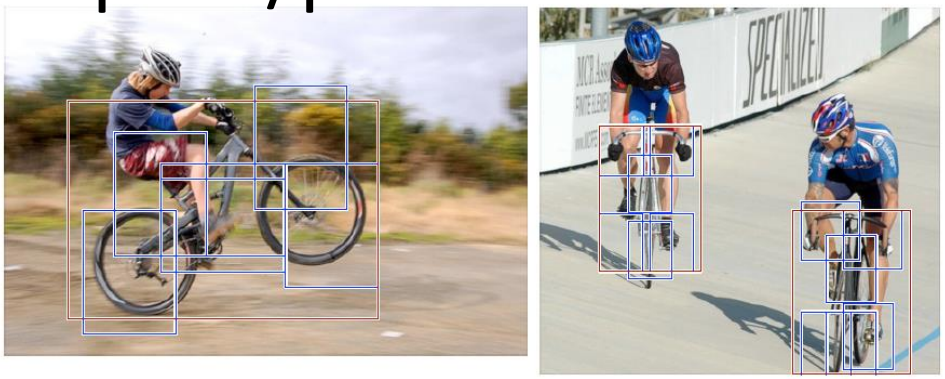


Images from Felzenszwalb

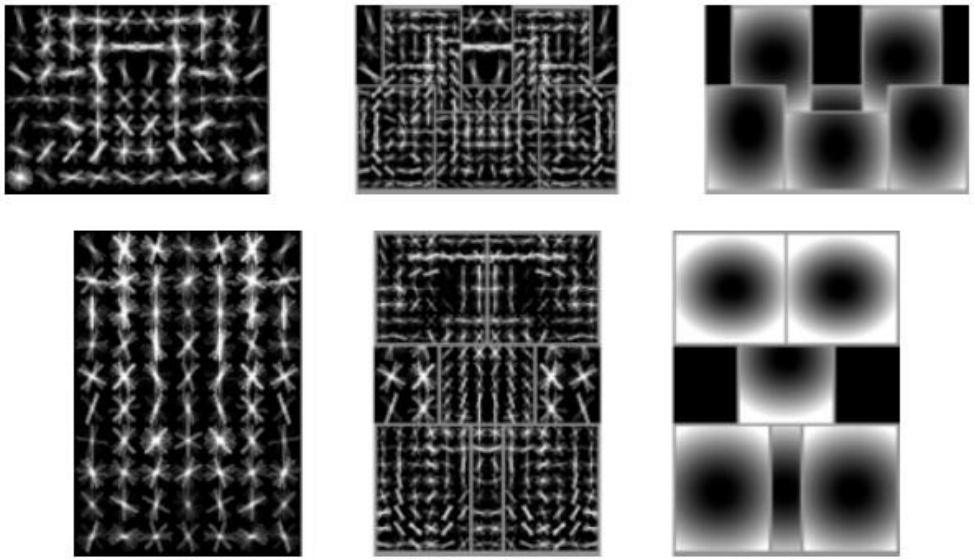
Specifying an object model

3. Hybrid template/parts model

Detections



Template Visualization



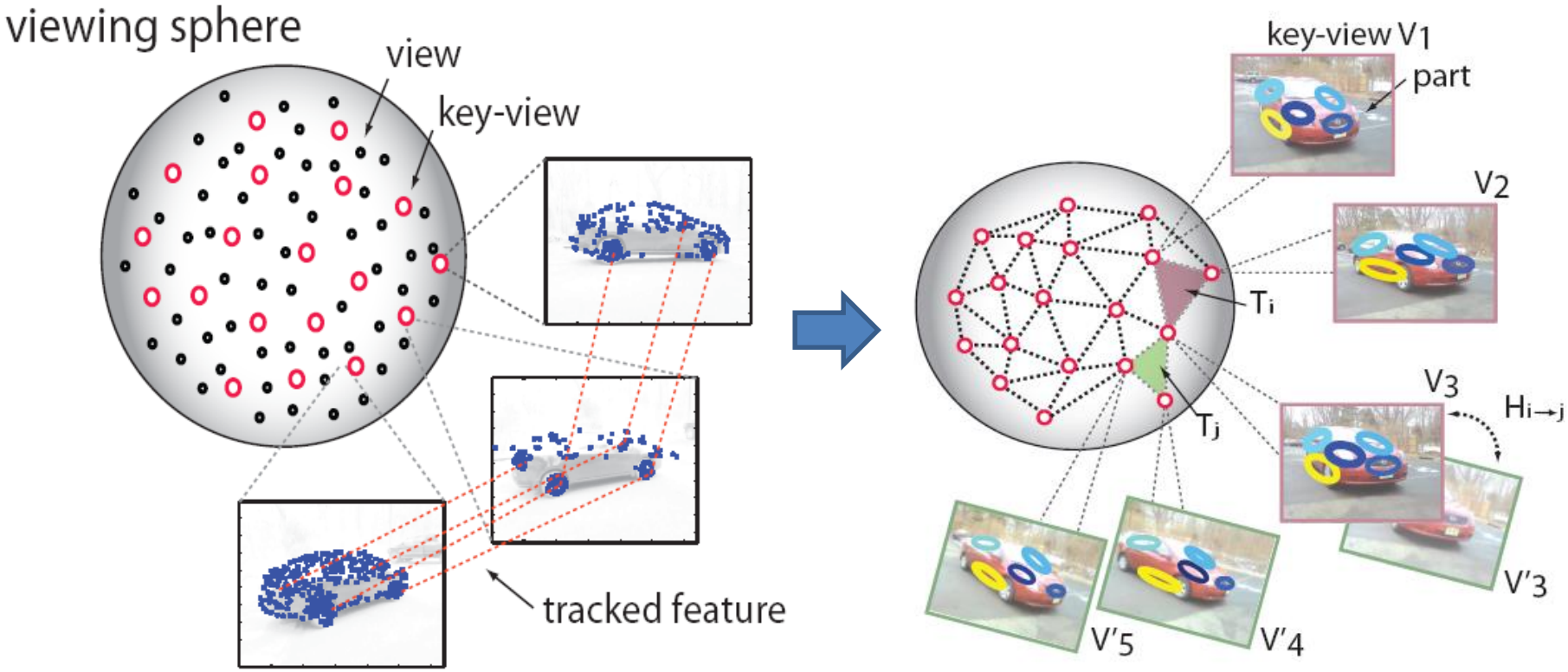
root filters coarse resolution | part filters finer resolution | deformation models

Felzenszwalb et al. 2008

Specifying an object model

4. 3D-ish model

- Object is collection of 3D planar patches under affine transformation

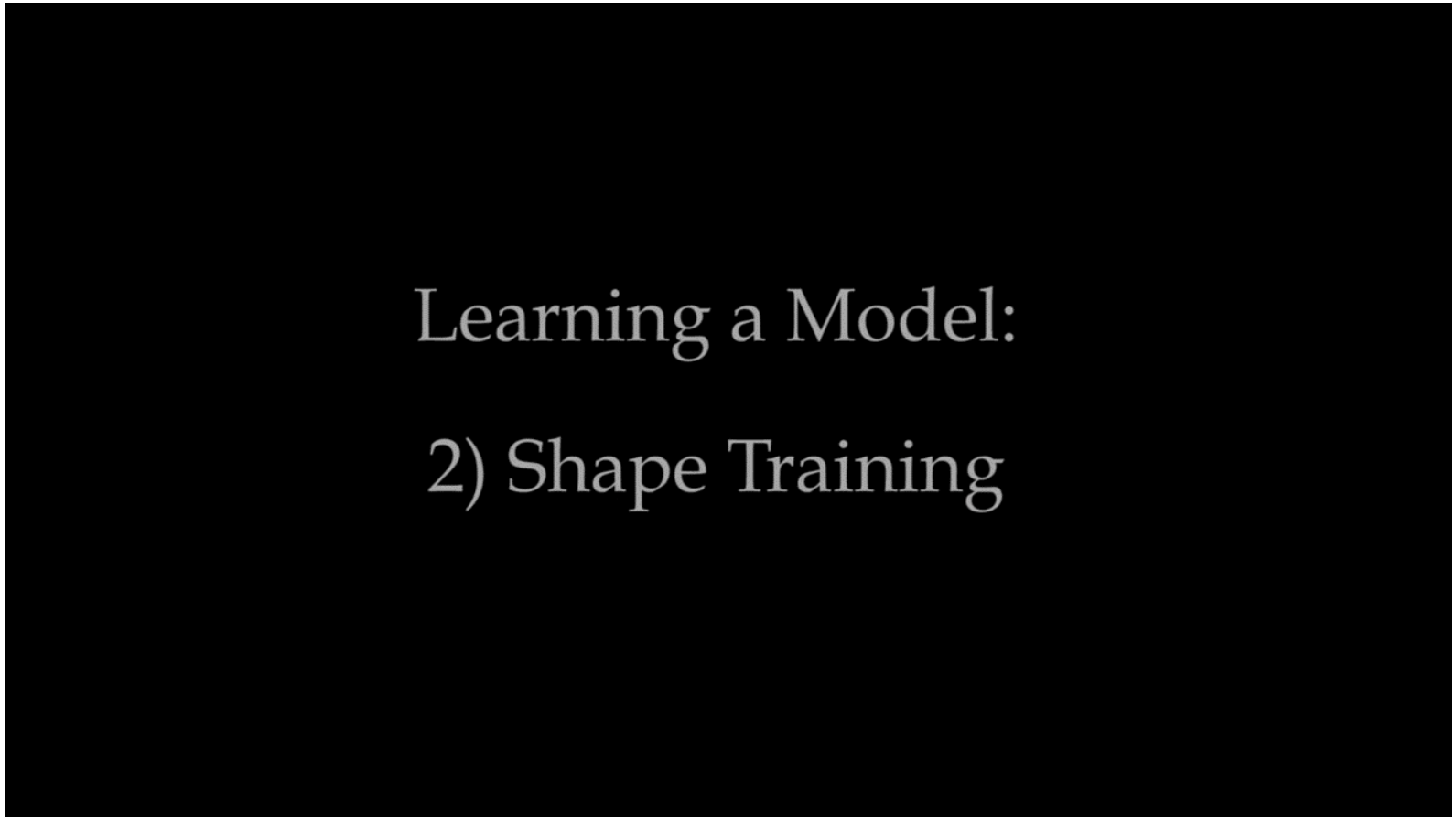


Loper et al. 2015

Specifying an object model

5. Deformable 3D model

- Object is a parameterized space of shape/pose/deformation of class of 3D object

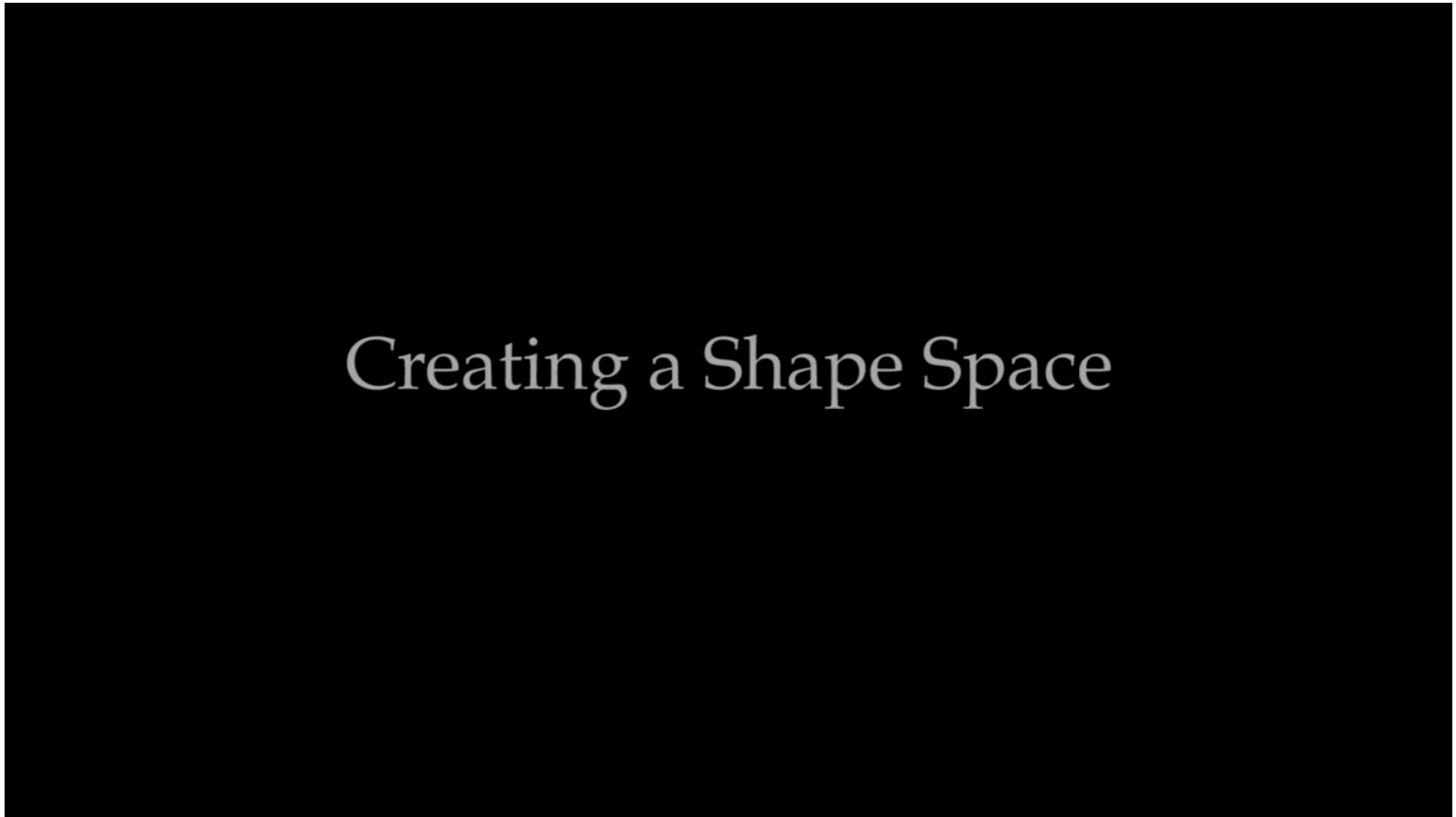


Loper et al. 2015

Specifying an object model

5. Deformable 3D model

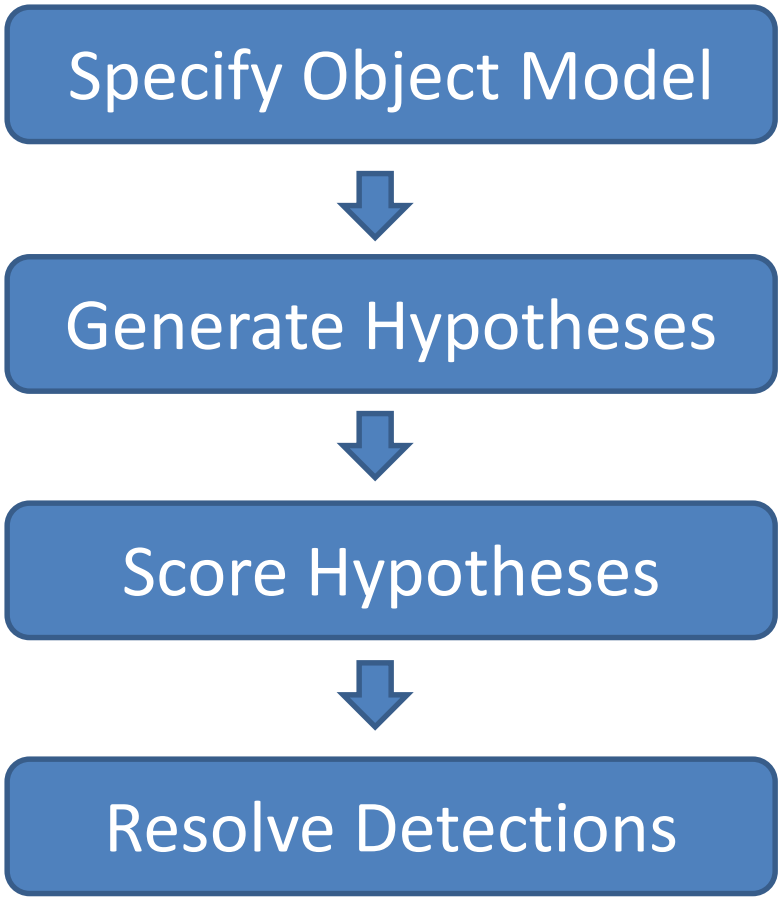
- Object is a parameterized space of shape/pose/deformation of class of 3D object



Why not just pick the most complex model?

- Inference is harder
  - More parameters
  - Harder to ‘fit’ (infer / optimize fit)
  - Longer computation

General Process of Object Recognition



Propose an alignment of the model to the image

Generating hypotheses

1. 2D template model / sliding window
   - Test patch at each location and scale

Generating hypotheses

1. 2D template model / sliding window
   - Test patch at each location and scale

Note – Template did not change size

Each window is separately classified

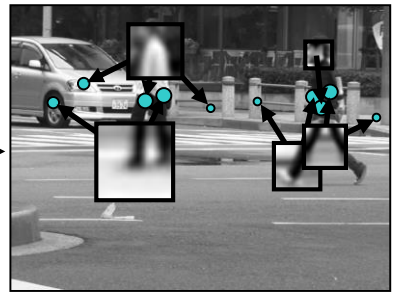
Generating hypotheses

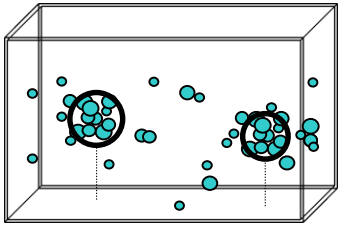
2. Voting from patches/keypoints

Interest Points

Matched Codebook Entries

Probabilistic Voting





3D Voting Space (continuous)

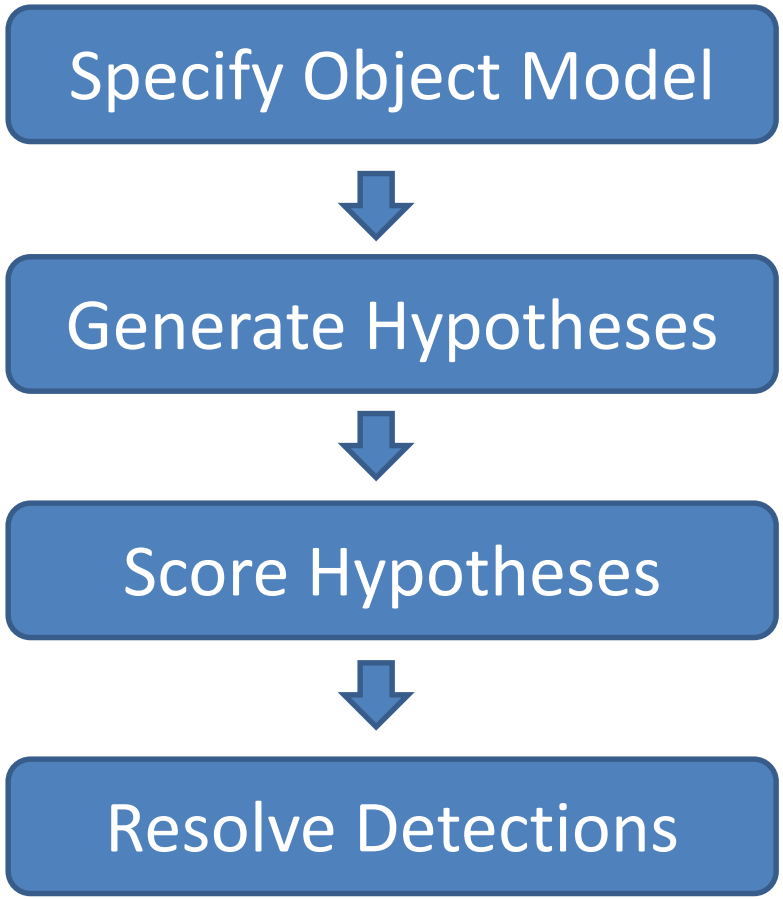
Implicit Shape Model by Leibe et al.

Generating hypotheses

3. Region-based proposal
   - Arbitrary bounding box + image ‘cut’ segmentation

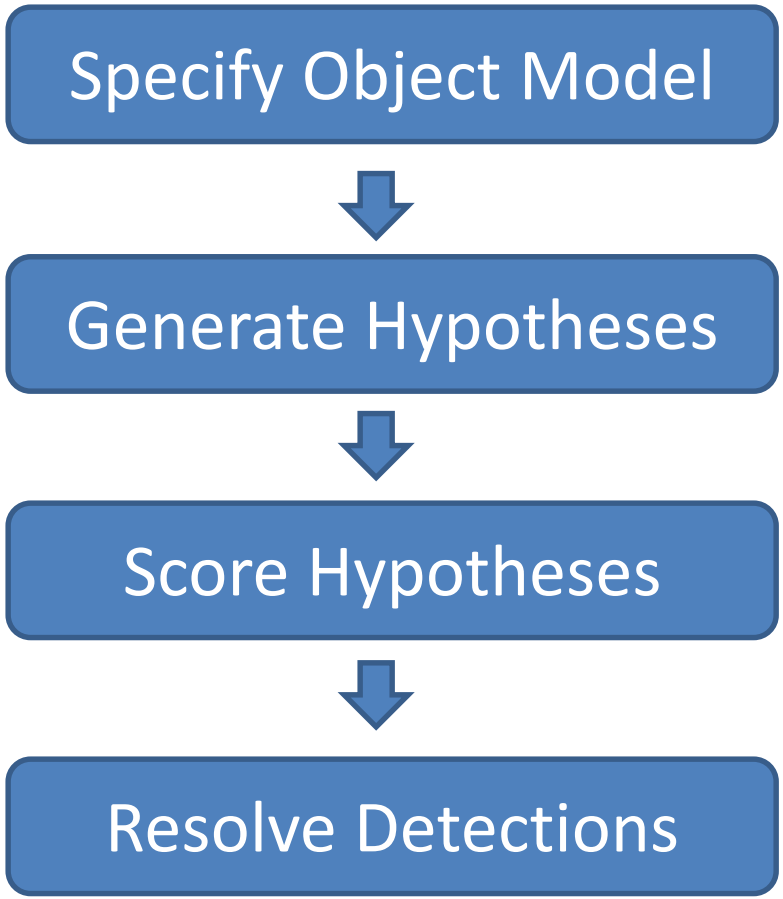
Endres Hoiem 2010

General Process of Object Recognition



Mainly gradient-based features, usually based on summary representation, many classifiers.

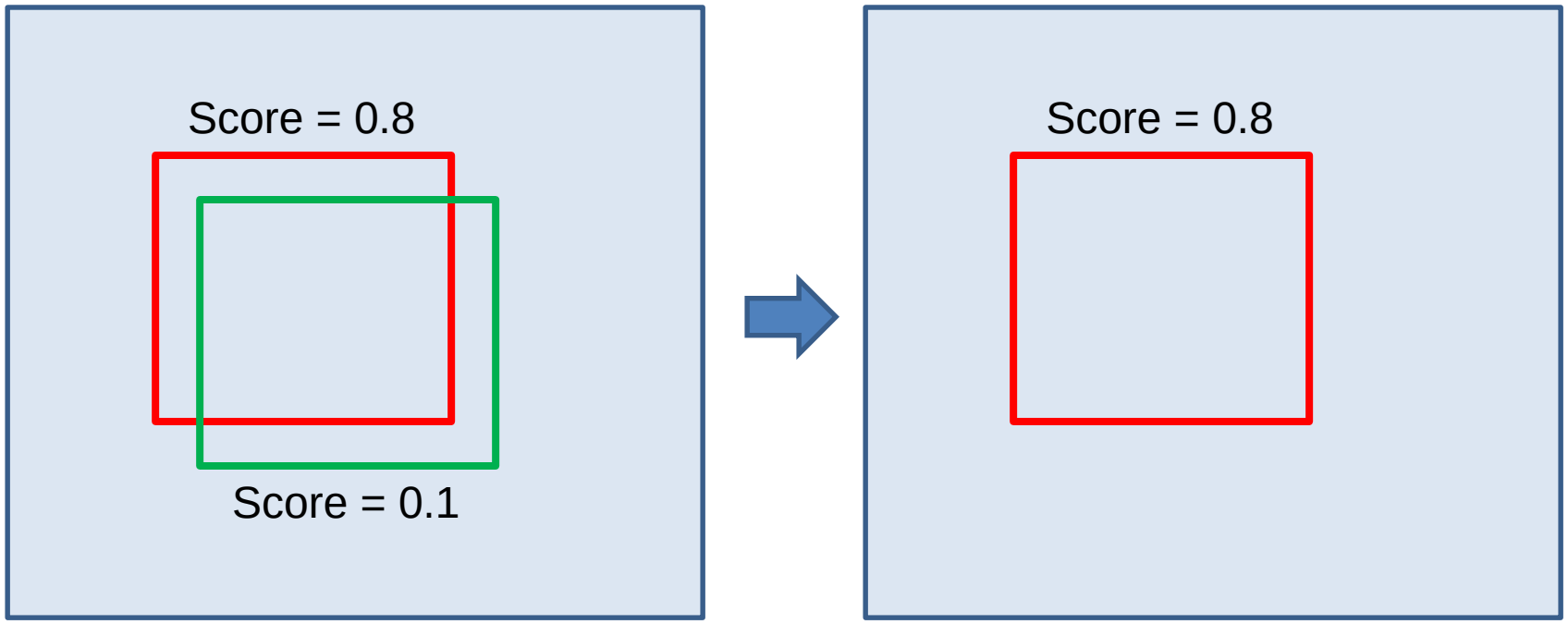
General Process of Object Recognition



Rescore each proposed object based on whole set

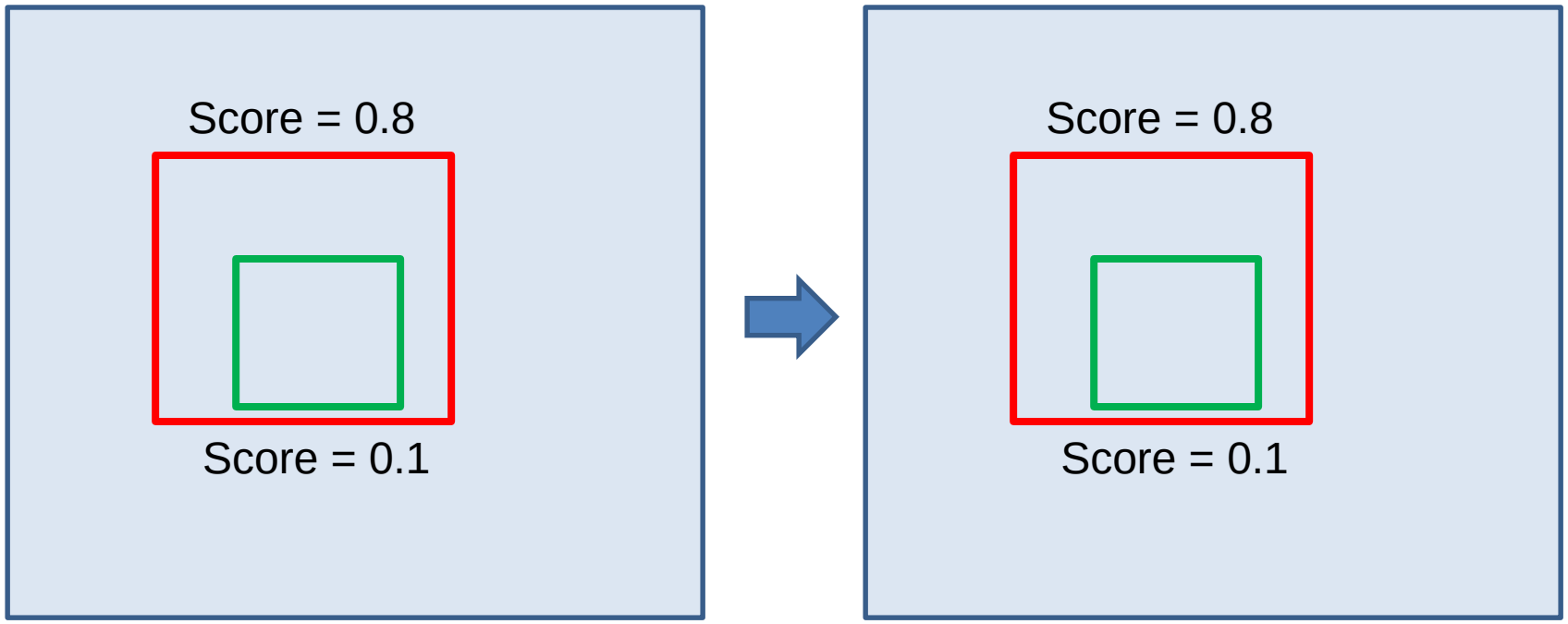
Resolving detection scores

1. Non-max suppression



Resolving detection scores

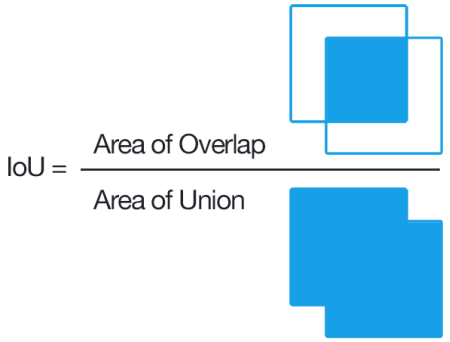
1. Non-max suppression



“Overlap” score is below some threshold

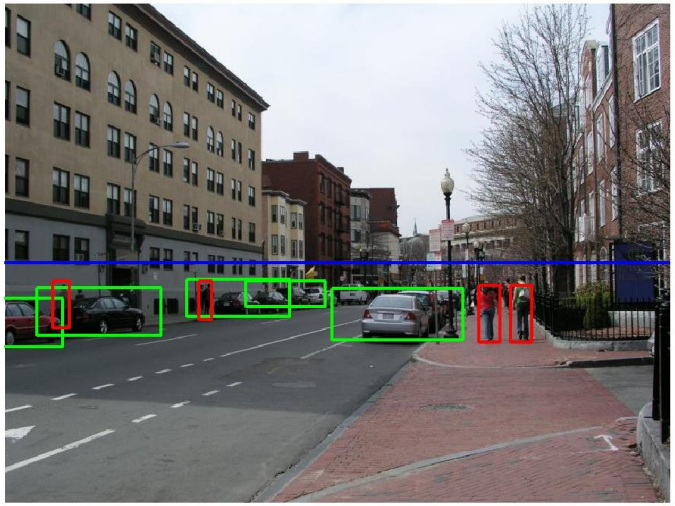
Where overlap = intersection over union (IoU)
often called *Jaccard index* or *similarity*

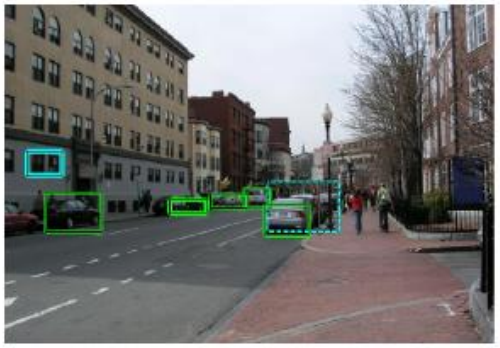
IoU close to 1.0 is good



Resolving detection scores

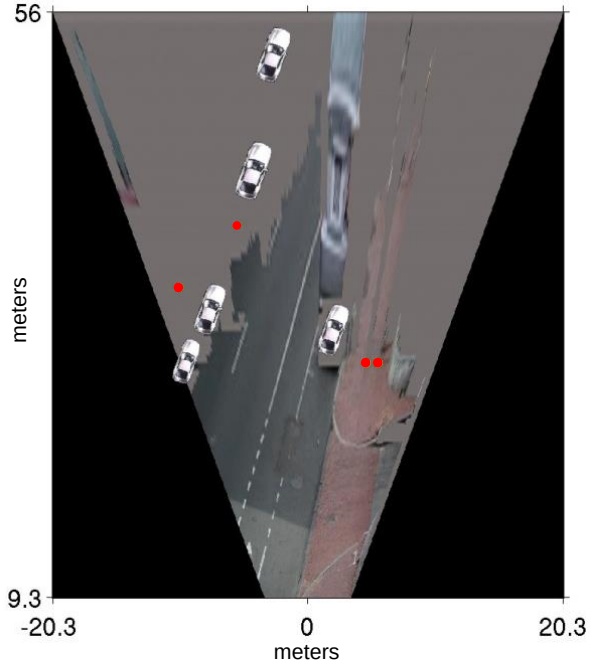
2. Context/reasoning
   - Via geometry
   - Via known information or prior distributions





(g) Car Detections: Local

(h) Ped Detections: Local



Hoiem et al. 2006

Dalal Triggs: Person detection with HOG & linear SVM

- Histograms of Oriented Gradients for Human Detection, Navneet Dalal, Bill Triggs, International Conference on Computer Vision & Pattern Recognition - June 2005
- http://lear.inrialpes.fr/pubs/2005/DT05/

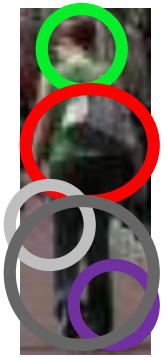
Statistical Template

Object model =

sum of scores of features *at fixed positions!*

+3 +2 -2 -1 -2.5 = -0.5 ? > 7.5

Non-object



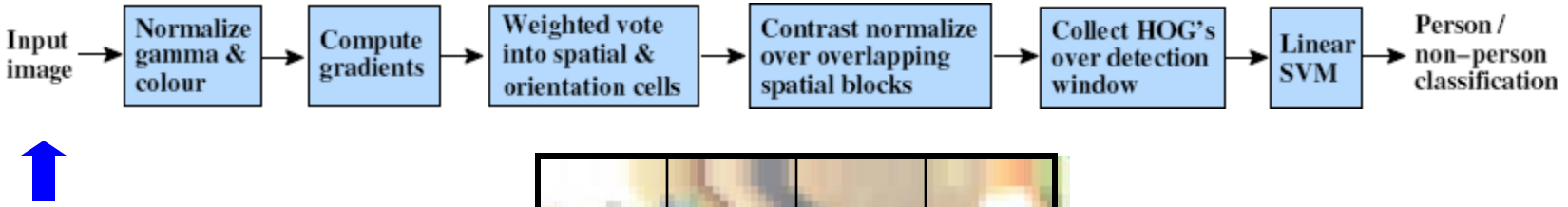
+4 +1 +0.5 +3 +0.5 = 10.5 ? > 7.5

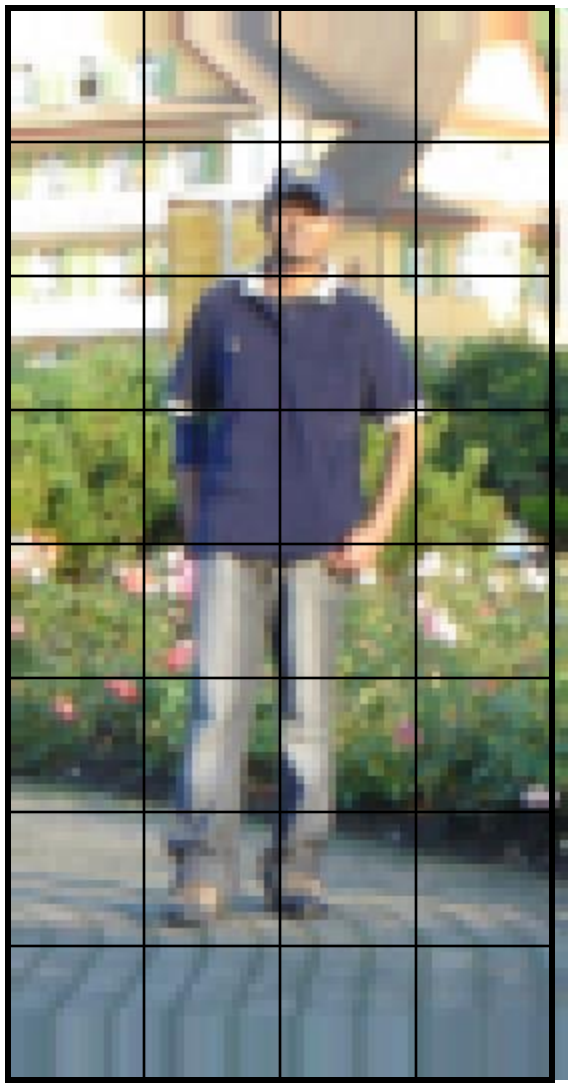
Object

Example: Dalal-Triggs pedestrian detector

1. Extract fixed-sized (64x128 pixel) window at each position and scale
2. Compute HOG (histogram of gradient) features within each window
3. Score the window with a linear SVM classifier
4. Perform non-maxima suppression to remove overlapping detections with lower scores

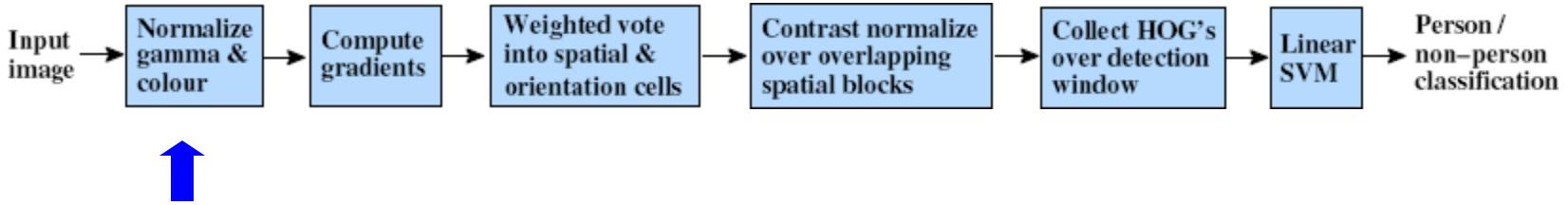
Navneet Dalal and Bill Triggs, Histograms of Oriented Gradients for Human Detection, CVPR05





Slides by Pete Barnum

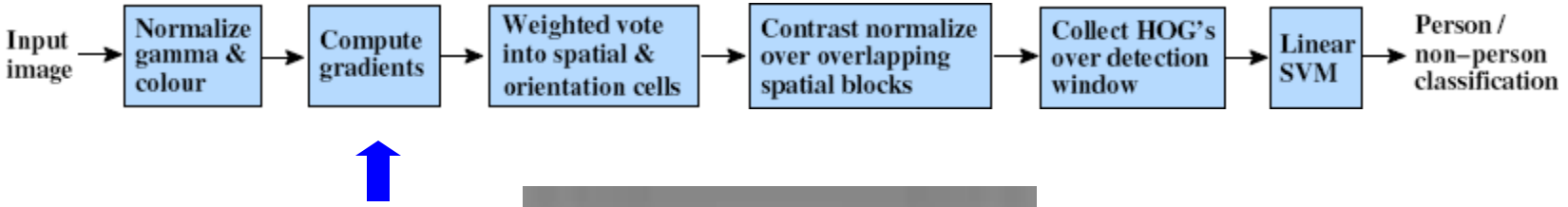
Navneet Dalal and Bill Triggs, Histograms of Oriented Gradients for Human Detection, CVPR05



- Tested with
  - RGB
  - LAB
  - Grayscale

  } Slightly better performance vs. grayscale
- Gamma Normalization and Compression
  - Square root
  - Log

  } Very slightly better performance vs. no adjustment



Outperforms

-1	0	1

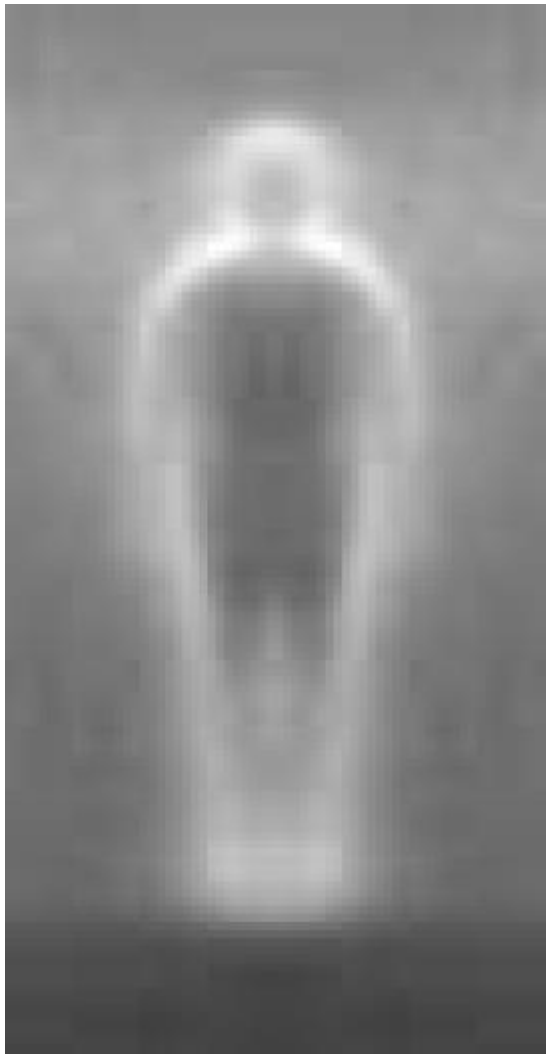
centered

-1	1

uncentered

1	-8	0	8	-1

cubic-corrected



0	1
-1	0

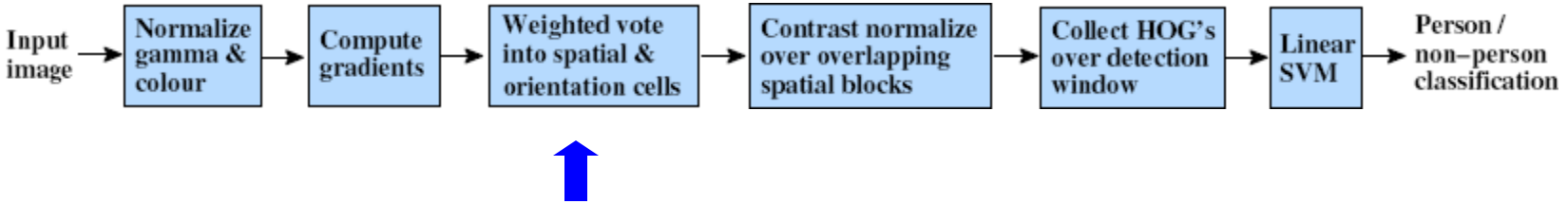
diagonal

-1	0	1
-2	0	2
-1	0	1

Sobel

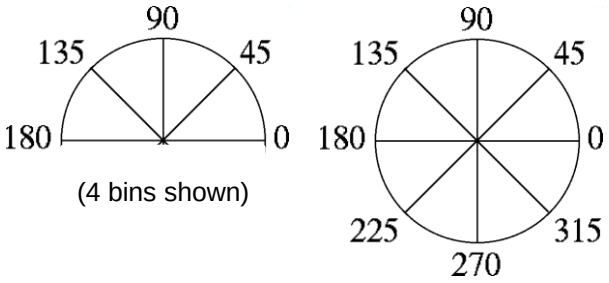
Slides by Pete Barnum

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

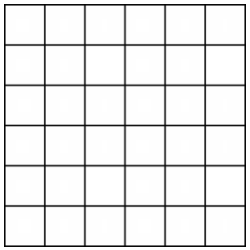


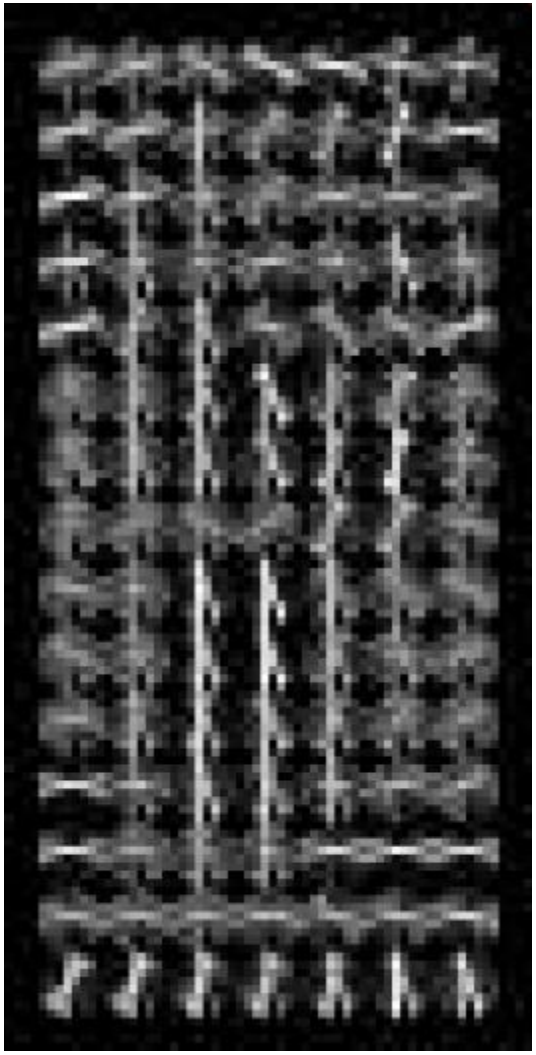
Histogram of Oriented Gradients

Orientation: 9 bins (for unsigned angles 0 -180)



Histograms over k x k pixel cells

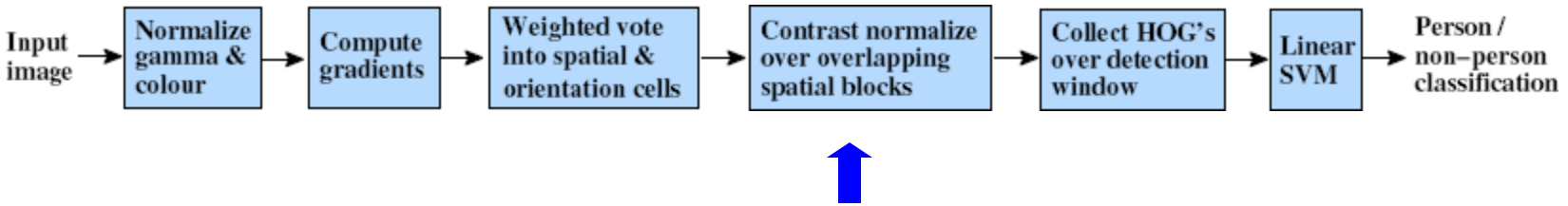




- Votes weighted by magnitude
- Bilinear interpolation between cells

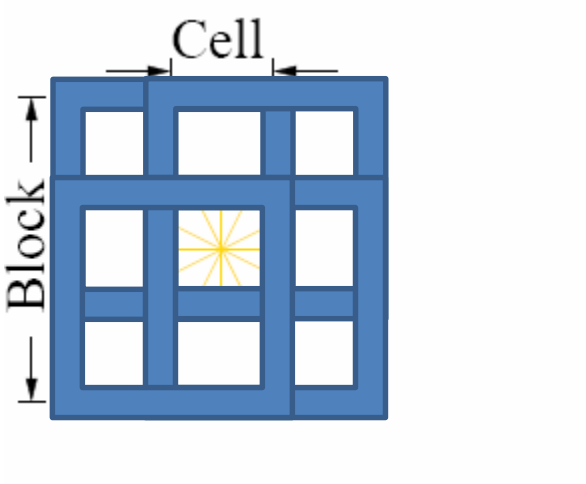
Slides by Pete Barnum

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



Normalize with respect to surrounding cells

Rectangular HOG (R-HOG)



How to normalize?

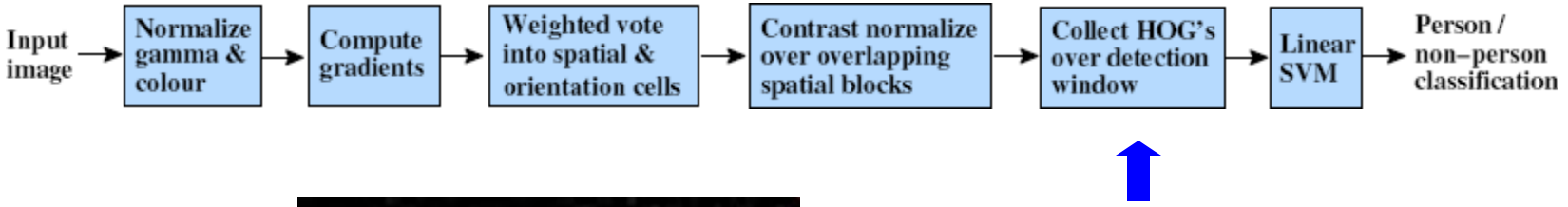
- Concatenate all cell responses from neighboring blocks into vector.
- Normalize vector.
- Extract responses from cell of interest.
- Do this 4x for each neighbor set in 2x2.

$$f = \frac{v}{\sqrt{\|v\|_2^2 + e^2}}$$

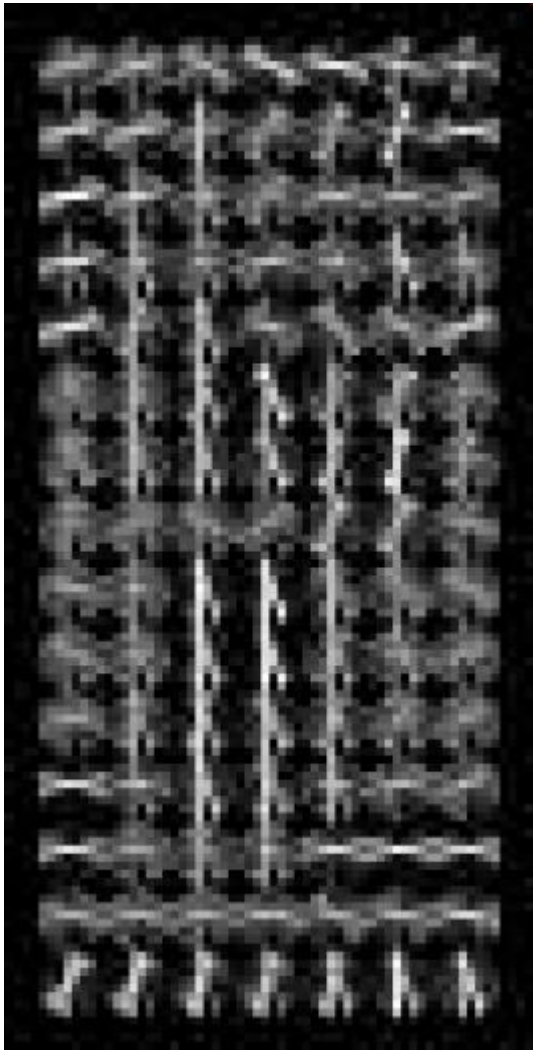
e is a small constant
(to remove div. by zero on empty bins)

Slides by Pete Barnum

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



X=



orientations

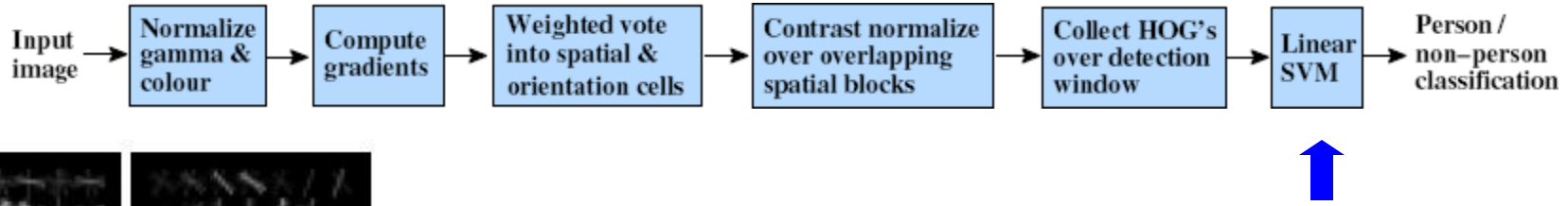
features = 15 x 7 x 9 x 4 = 3780

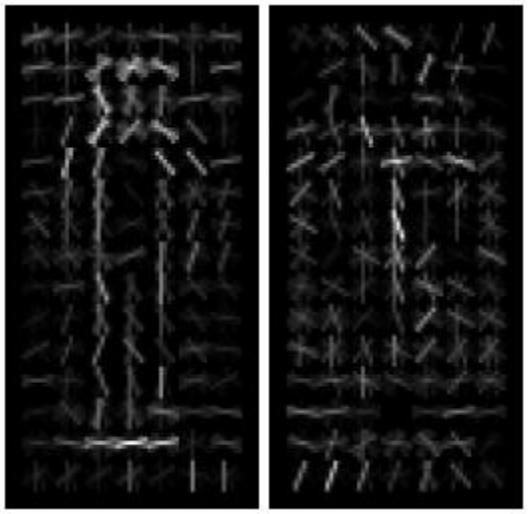
cells

normalizations by neighboring cells

Slides by Pete Barnum

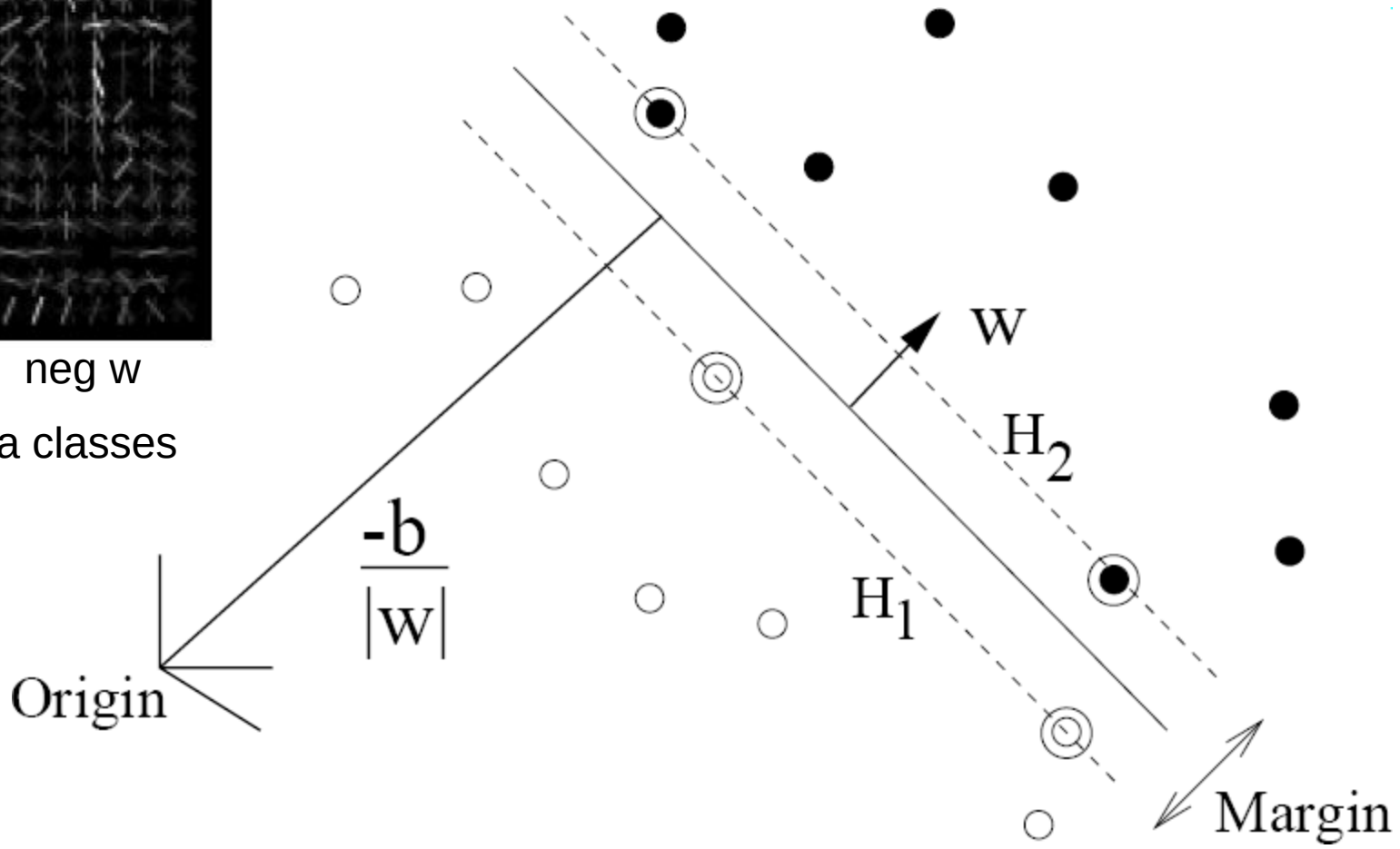
Navneet Dalal and Bill Triggs, Histograms of Oriented Gradients for Human Detection, CVPR05





pos w neg w

Training data classes



Slides by Pete Barnum

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

$$0.16 = w^T x - b$$

$$sign(0.16) = 1$$

$$=>$$ pedestrian

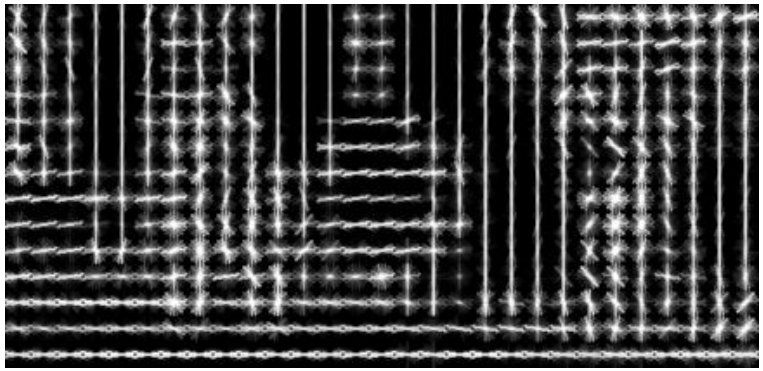
Slides by Pete Barnum

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

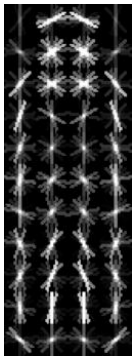
Pedestrian detection with HOG

- Learn a pedestrian template using a support vector machine
- At test time, compare feature map with template over sliding windows.
- Find local maxima of response
- *Multi-scale*: repeat over multiple levels of a HOG pyramid

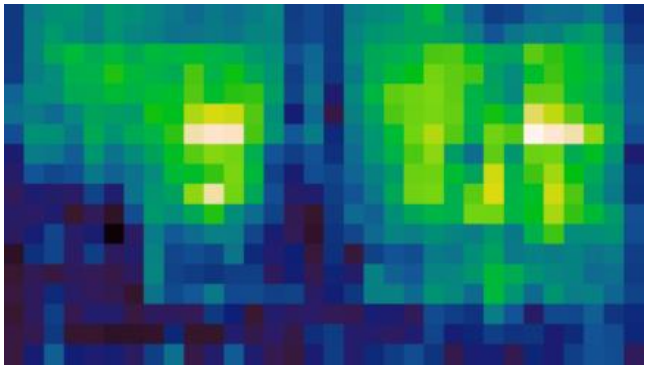
HOG feature map



Template



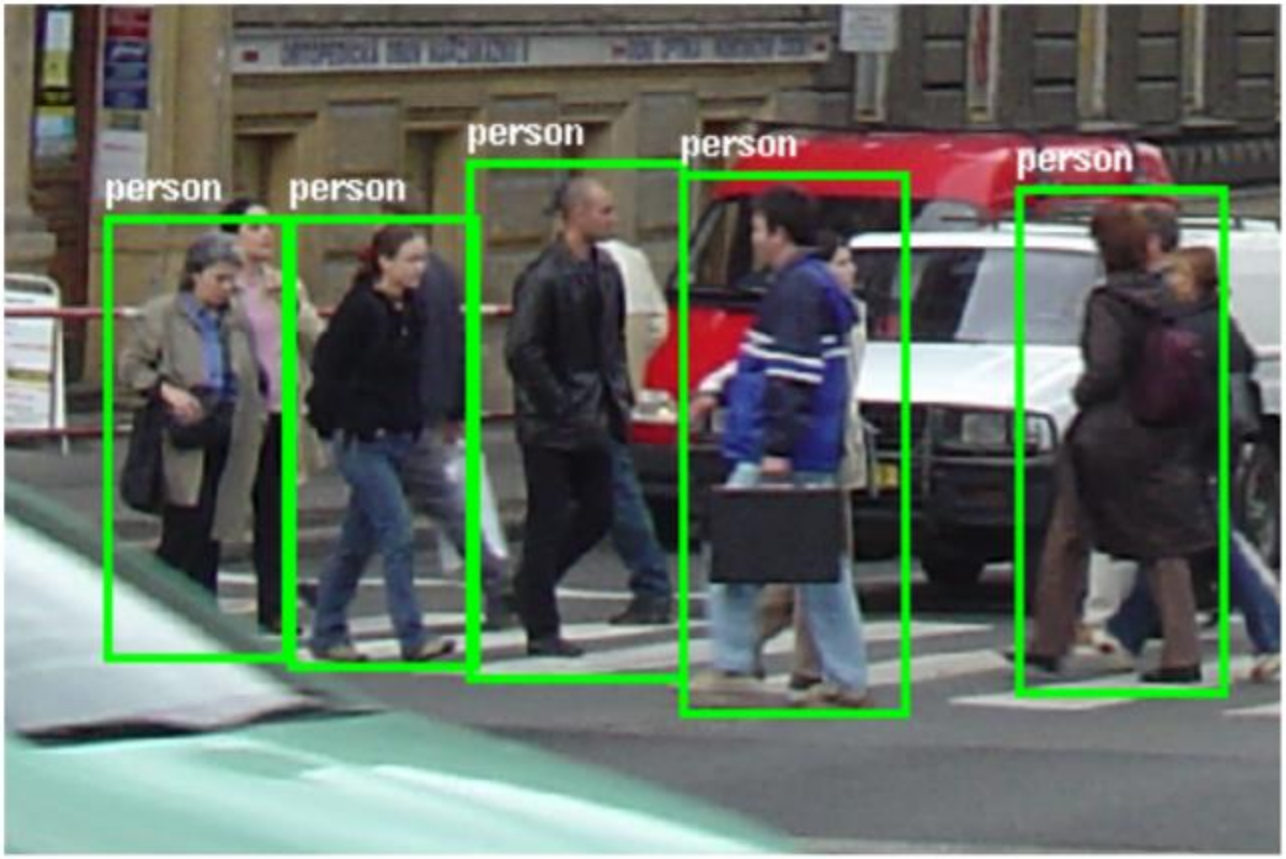
Detector response map



Can be ‘continuous’ (subpixel) for more sophisticated maxima finding

N. Dalal and B. Triggs, Histograms of Oriented Gradients for Human Detection, CVPR 2005

INRIA pedestrian database



INRIA pedestrian database issues

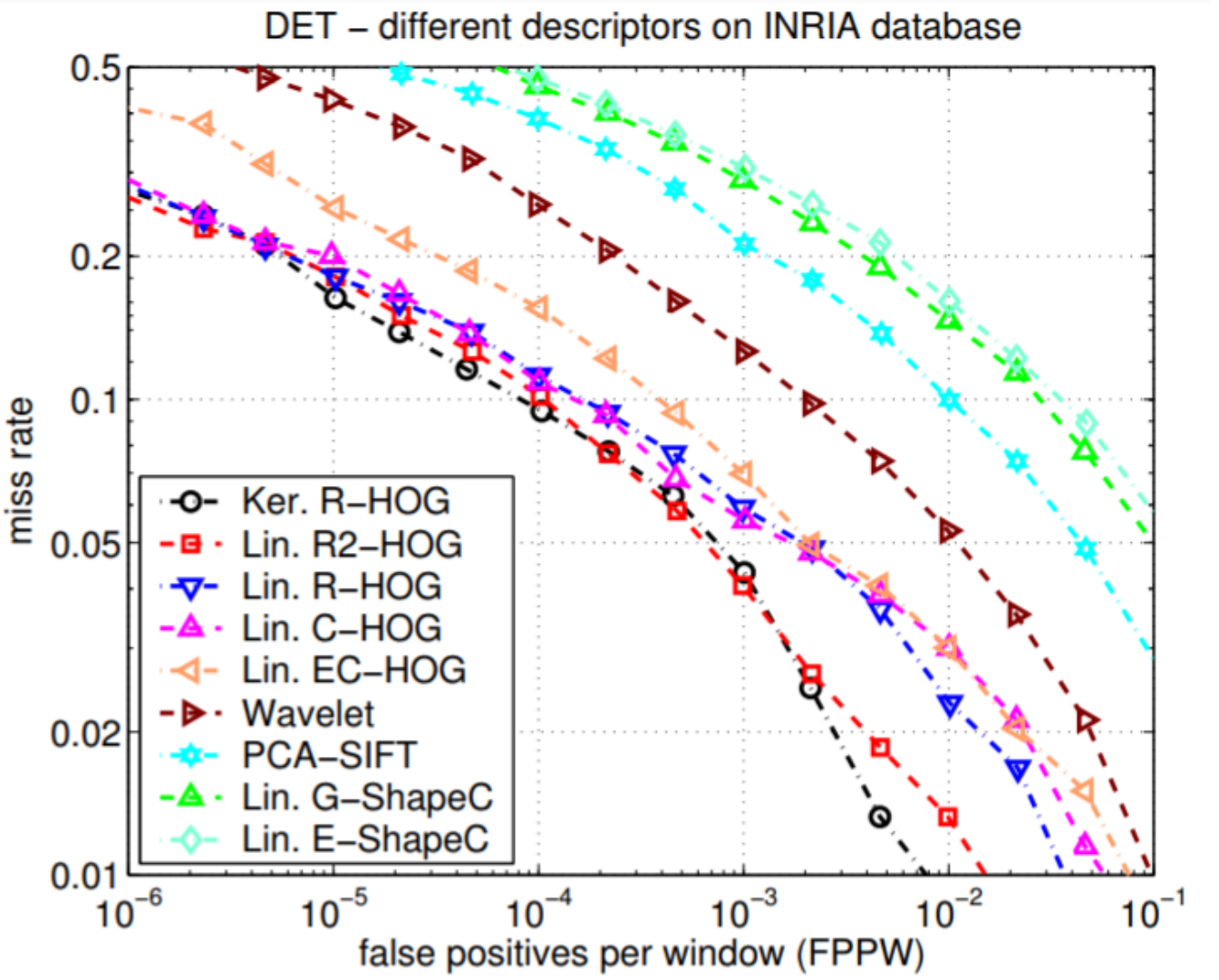
(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 1. Details from the INRIA test set highlighting some limitations. (a–d) Unlabelled persons. (e–h) Ambiguous cases. (e) Reflections of persons on a shop window, not labelled. (f) Some persons drawn on a wall, only one of them is labelled. (g) Some mannequins, all labelled. (h) A poster depicting a man, not labelled.

How good is HOG at person detection?

Miss rate =
1 - recall



Something to think about…

- Sliding window detectors work
  - *very well* for faces
  - *fairly well* for cars and pedestrians
  - *badly* for cats and dogs
- Why are some classes easier than others?

Strengths/Weaknesses of Statistical Template Approach

Strengths

- Works very well for non-deformable objects with canonical orientations: faces, cars, pedestrians
- Fast detection

Weaknesses

- Not so well for highly deformable objects or “stuff”
- Not robust to occlusion
- Requires lots of training data

Tricks of the trade

- Details in feature computation really matter
  - E.g., normalization in Dalal-Triggs improves detection rate by 27% at fixed false positive rate

- Template size
  - Typical choice is size of smallest expected detectable object

- “Jittering” or “augmenting” to create synthetic positive examples
  - Create slightly rotated, translated, scaled, mirrored versions as extra positive examples.

- Bootstrapping to get hard negative examples
  1. Randomly sample negative examples
  2. Train detector
  3. Sample negative examples that score > -1
  4. Repeat until all high-scoring negative examples fit in memory

Live demo

Dalal-Triggs uses a template with a rigid form – humans are boxed shaped.

But…is there a way to learn the spatial layout more fluidly?

- Might help us capture more appearance variation…

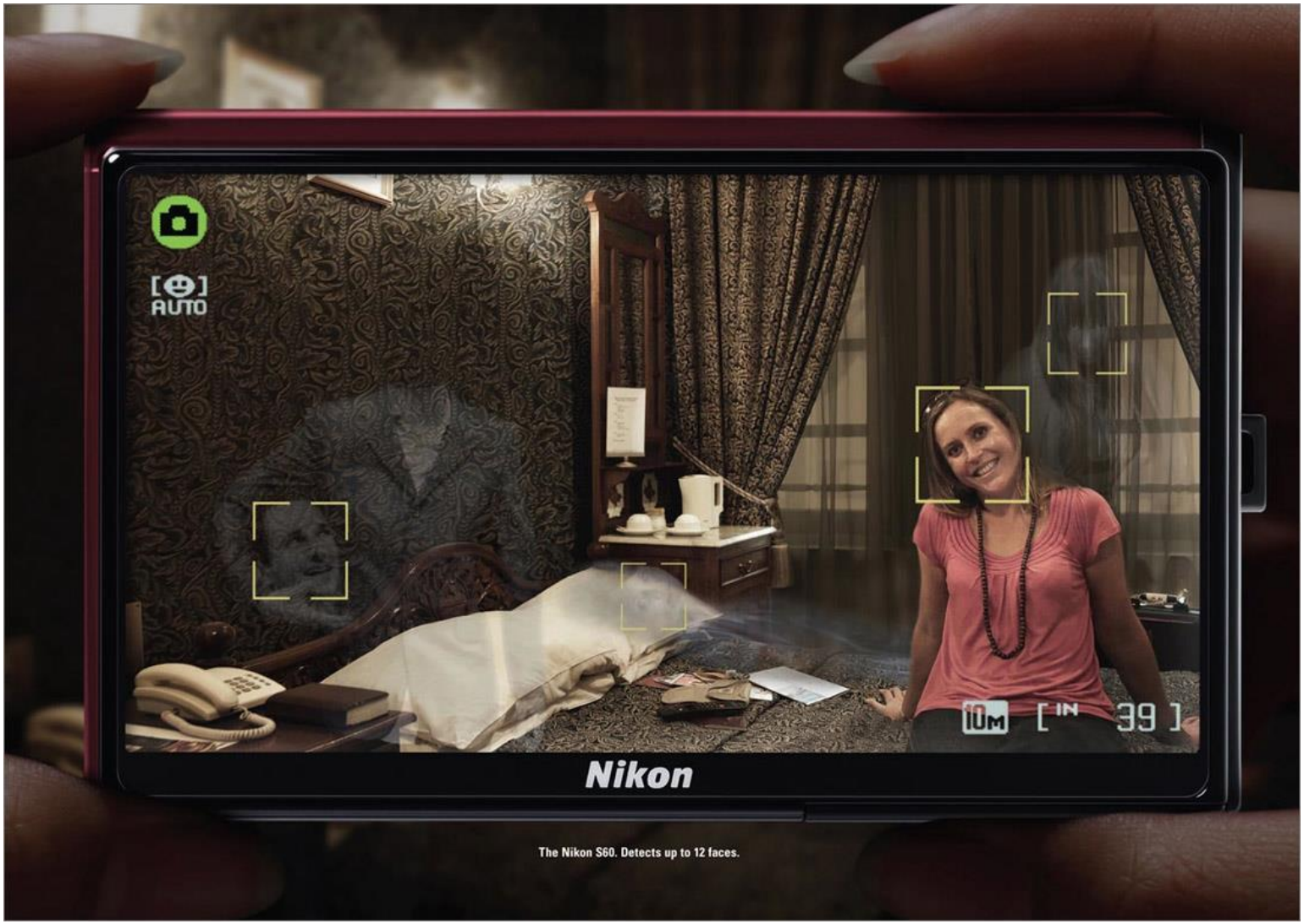
What about faster, too?

FACE DETECTION

Consumer application: Apple iPhoto

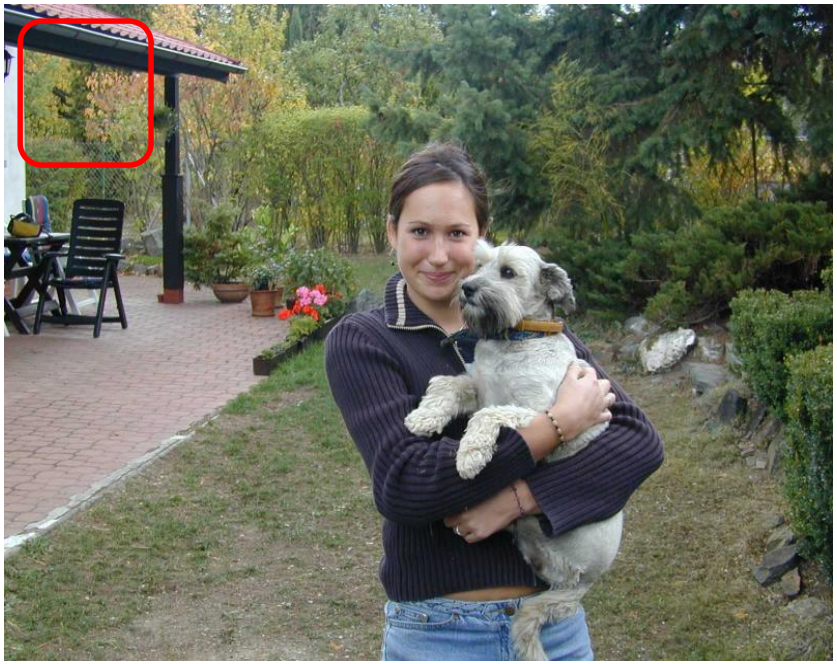
Things iPhoto thinks are faces

"The Nikon S60 detects up to 12 faces."




This person tried to unlock your Phone
2016/07/23 15:51

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.

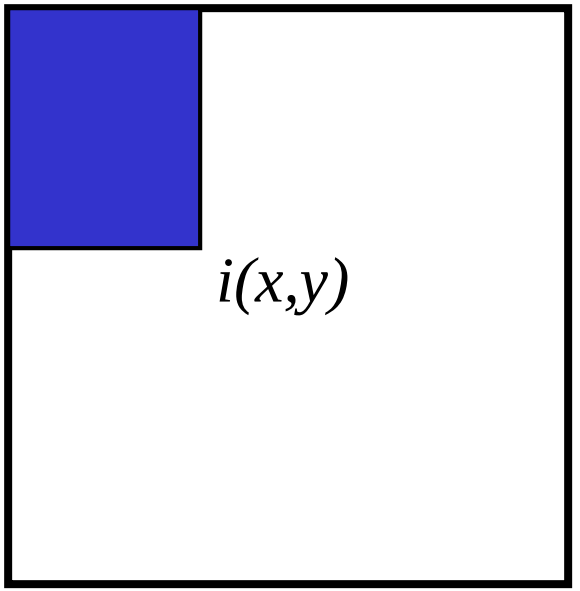
Training is slow, but detection is very fast

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.

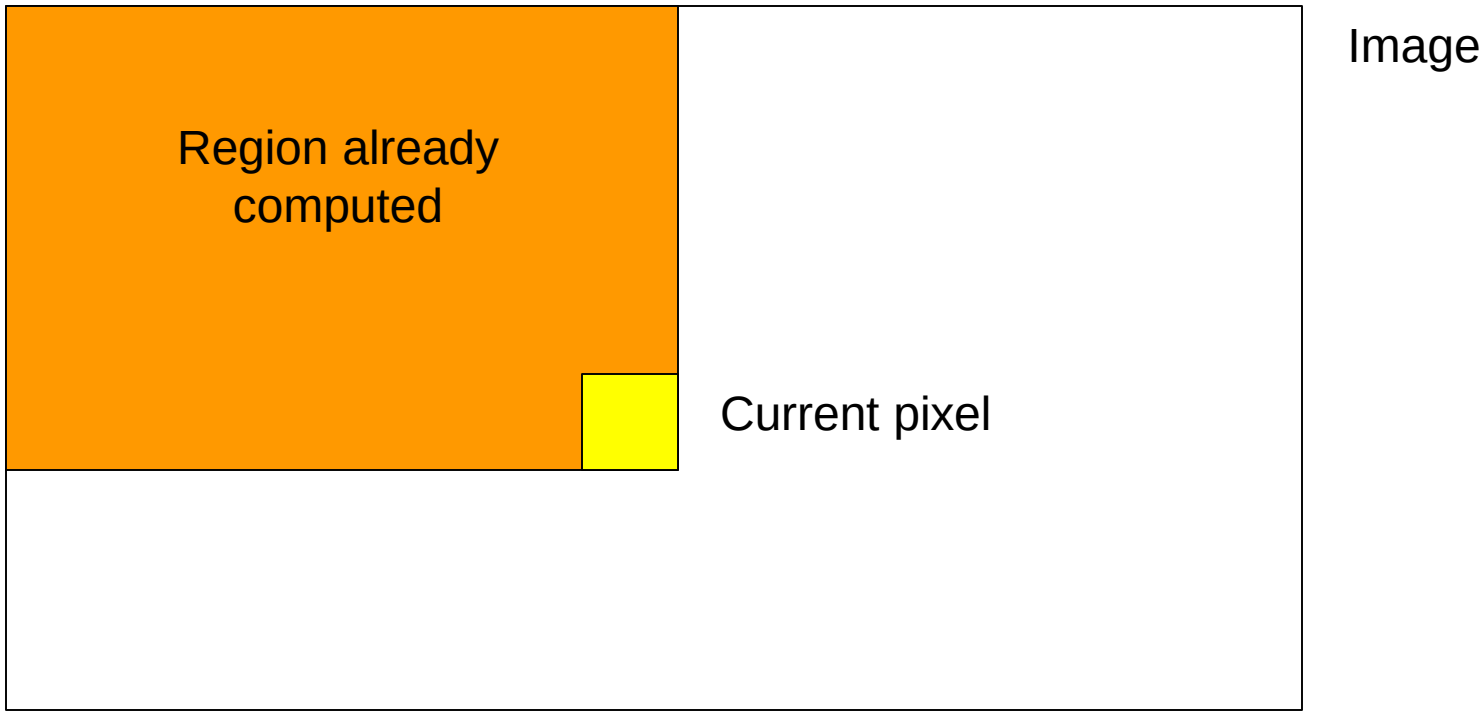


- This can quickly be computed in one pass through the image.

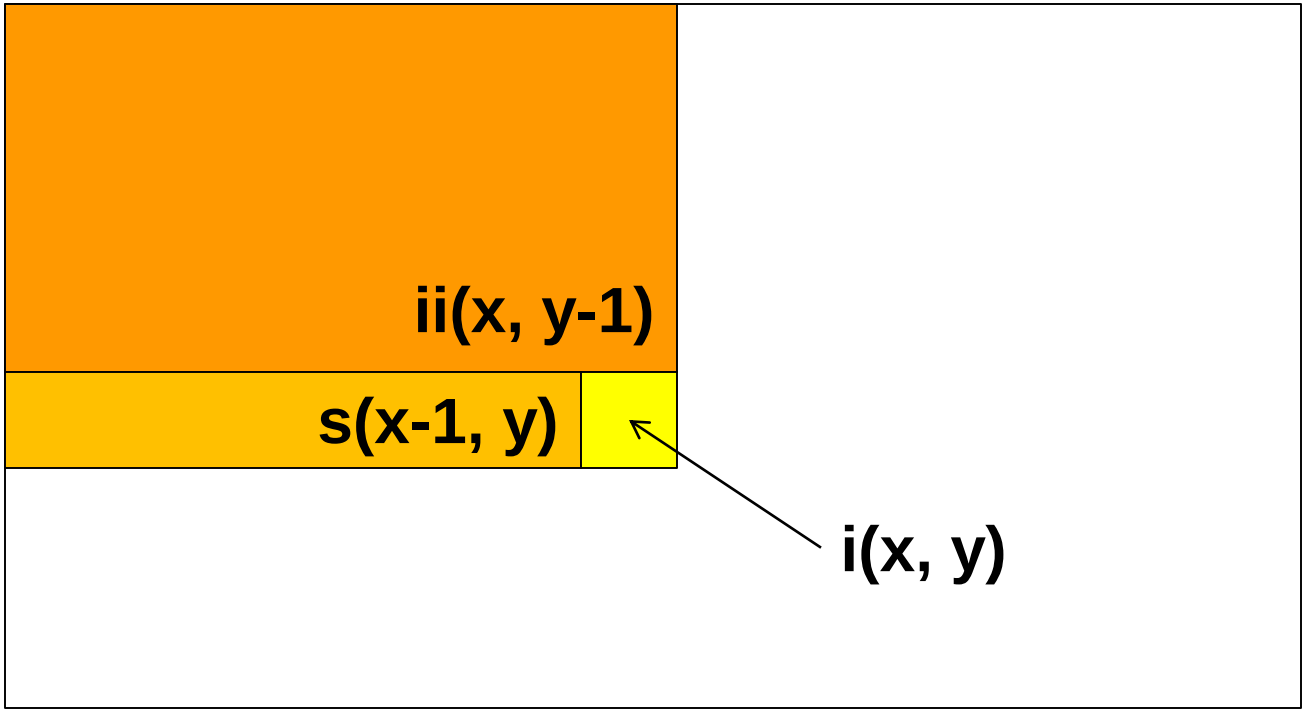
$$I_{\sum}(x,y) = \sum_{\substack{x' \le x \\ y' \le y}} i(x',y')$$

- ‘Summed area table’

Computing the integral image



Computing the integral image



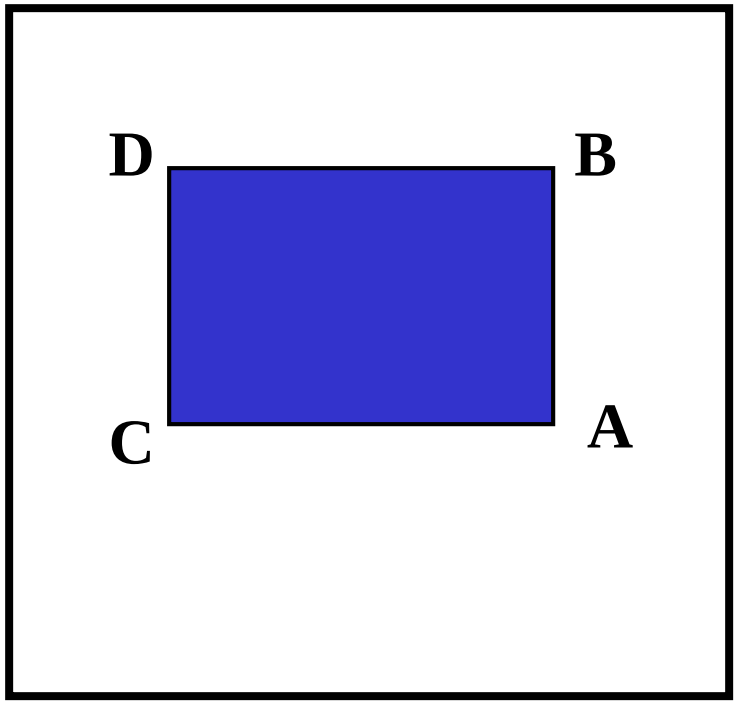
Cumulative row sum: $s(x, y) = s(x–1, y) + i(x, y)$

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

Python: ii = np.cumsum(i)

Computing sum within a rectangle

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



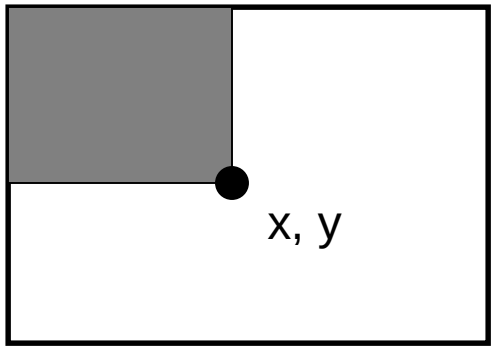
- The sum of original image values within the rectangle can be computed as:

  sum = A – B – C + D

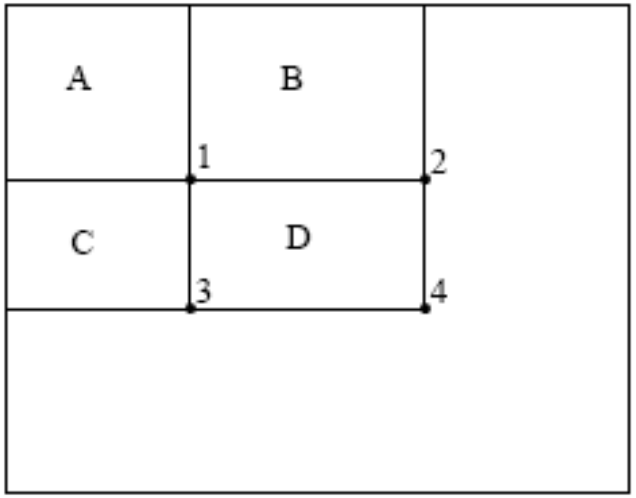
Only 3 additions are required for any size of rectangle!

Integral Images

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



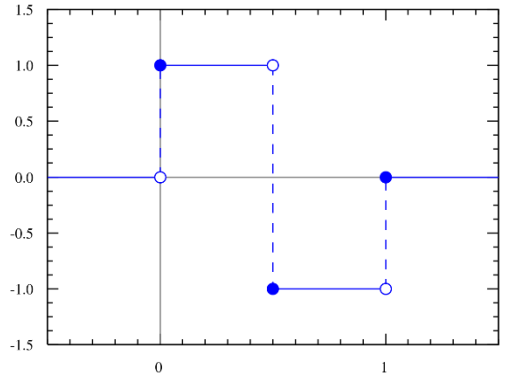
ii(x,y) = Sum of the values in the grey region



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

Features that are fast to compute

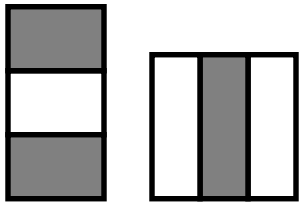
Haar wavelet



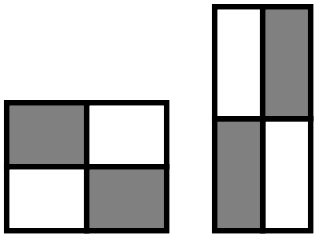
"Haar-like features"

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

Two-rectangle features



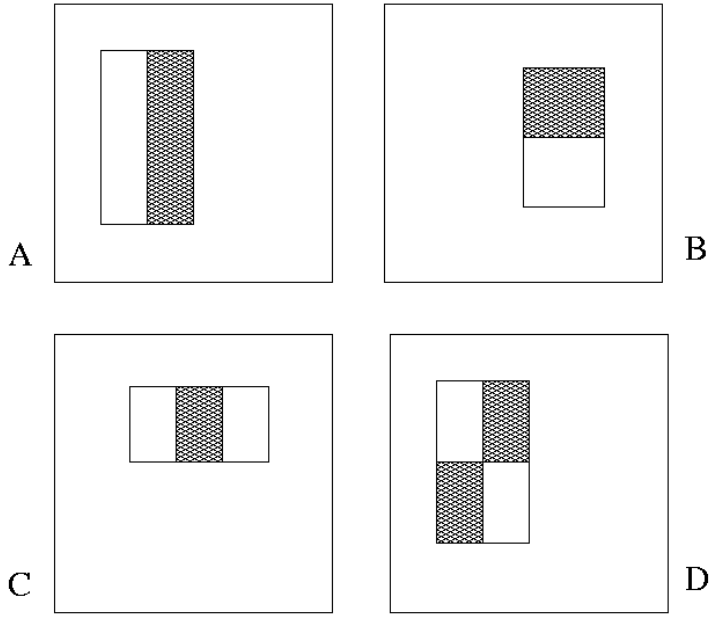
Three-rectangle features



Etc.

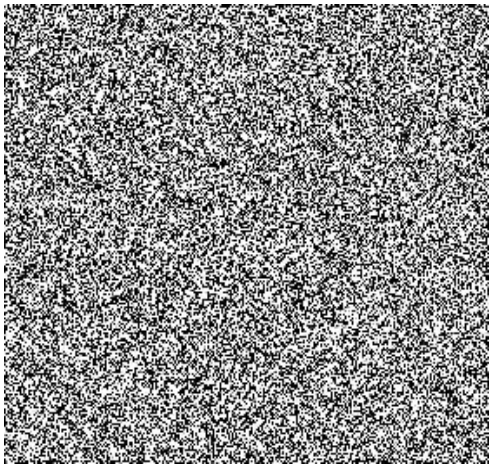
Image Features

"Rectangle filters"



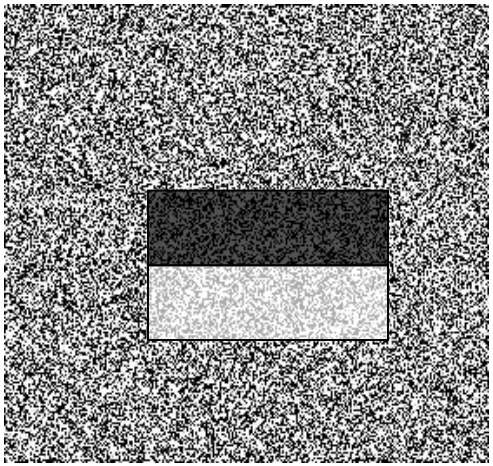
$$Value = \sum(\text{pixels in white area}) - \sum(\text{pixels in black area})$$

Example

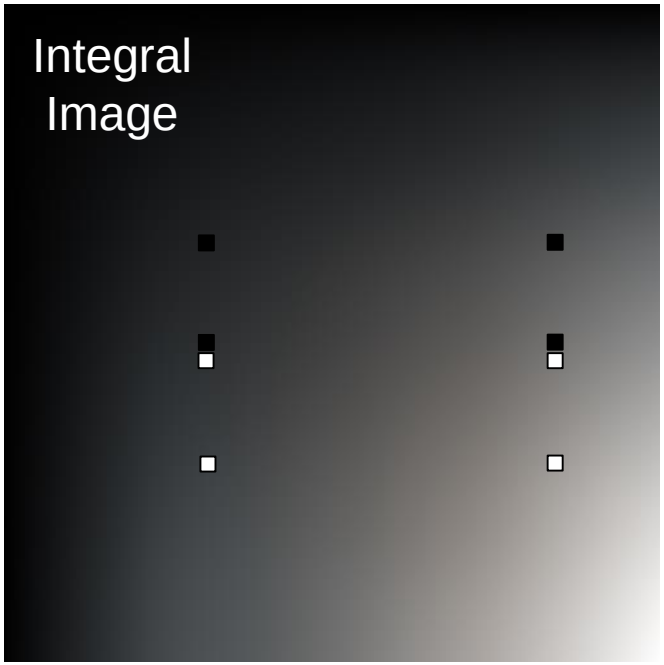


Source

Result



Computing a rectangle feature

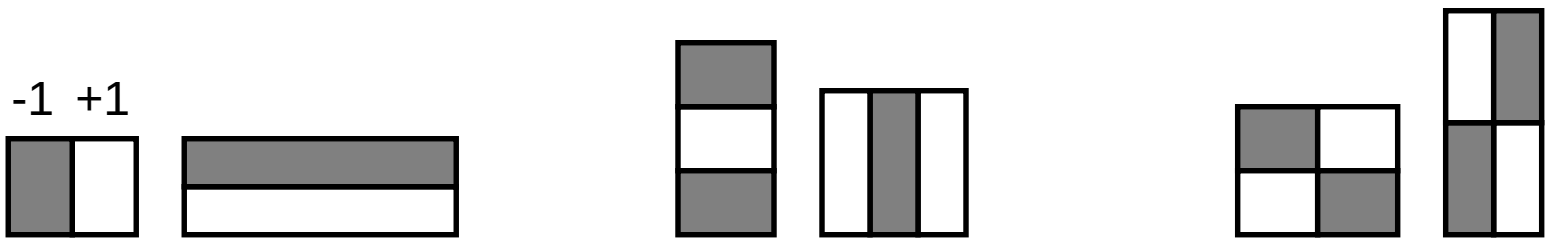


But these features are rubbish…!

Yes, individually they are ‘weak classifiers’

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

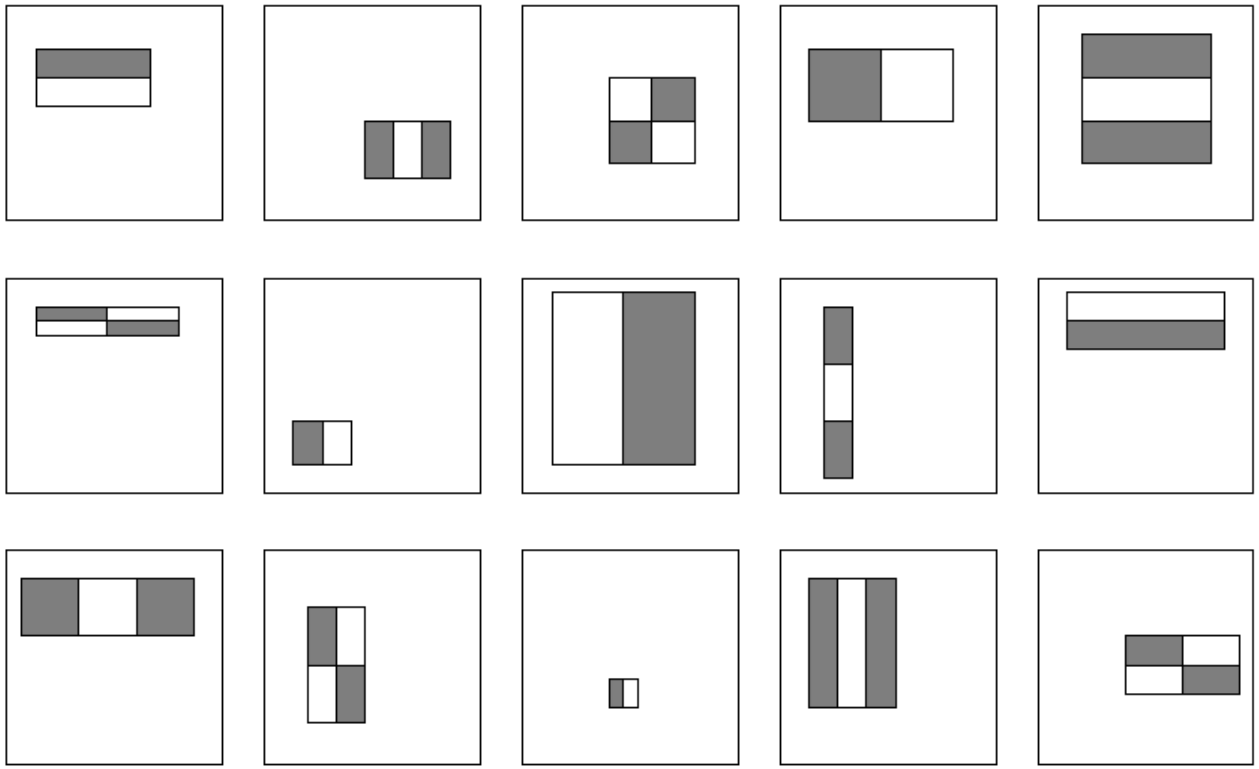
But, what if we combine *thousands* of them…



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!



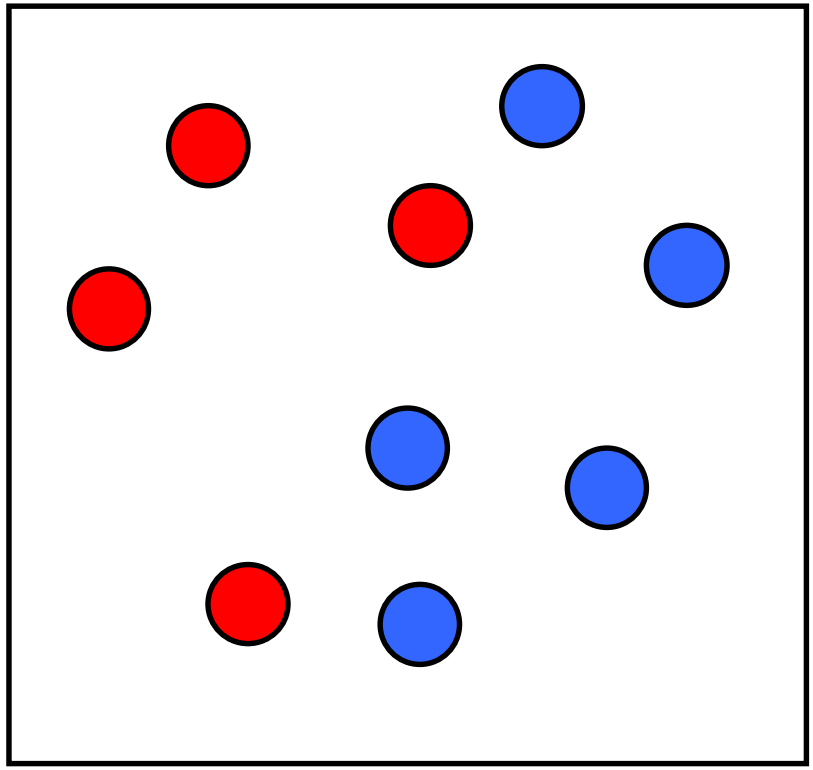
How many features are there?

- 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



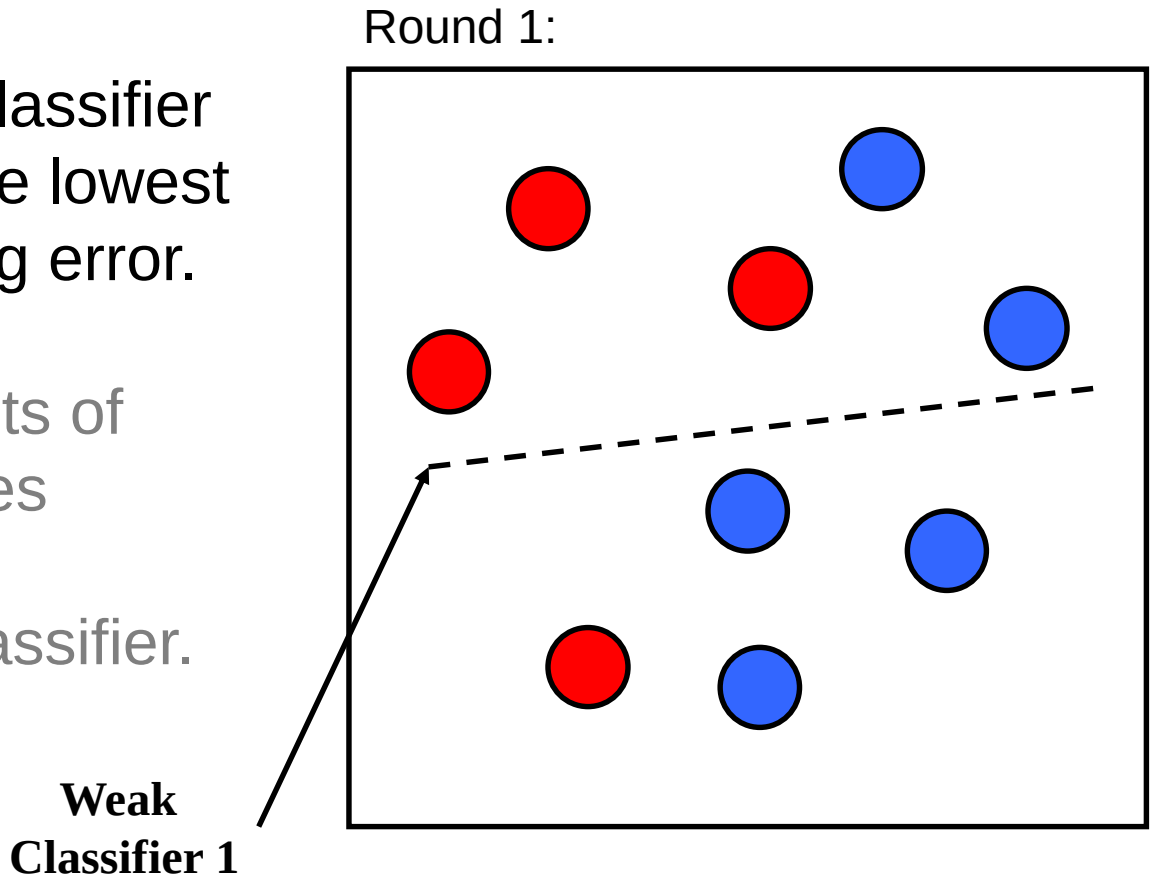
Slide credit: Paul Viola

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.



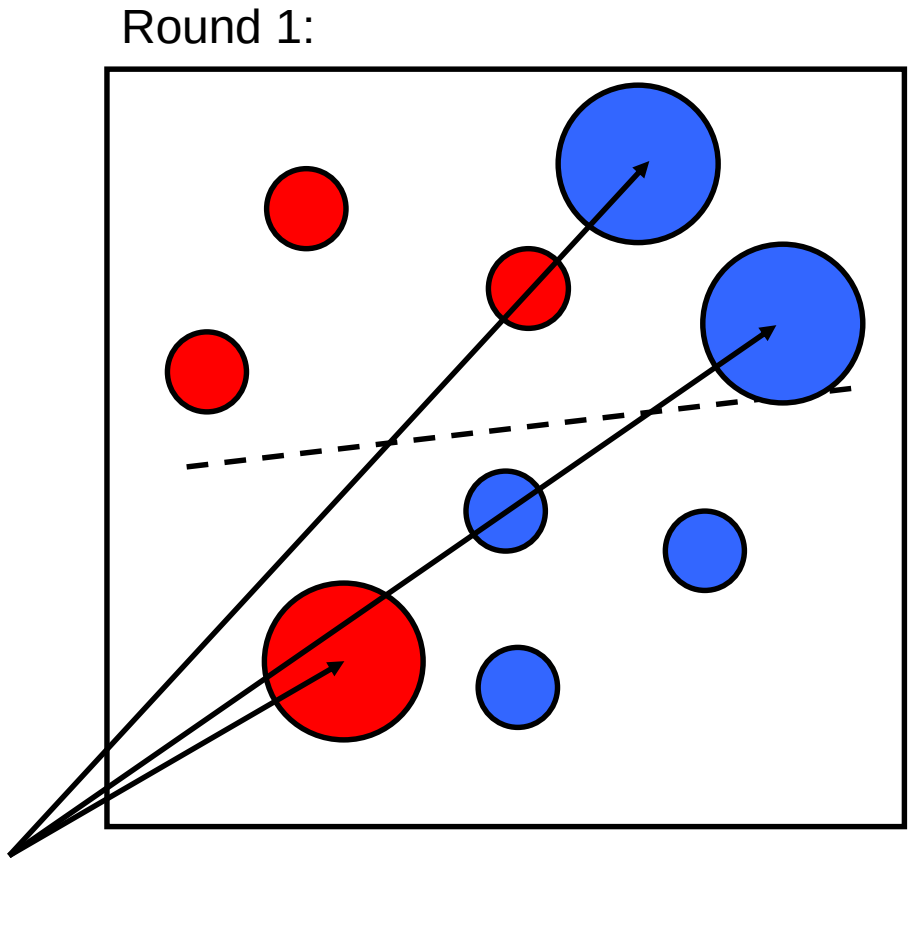
Slide credit: Paul Viola

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.

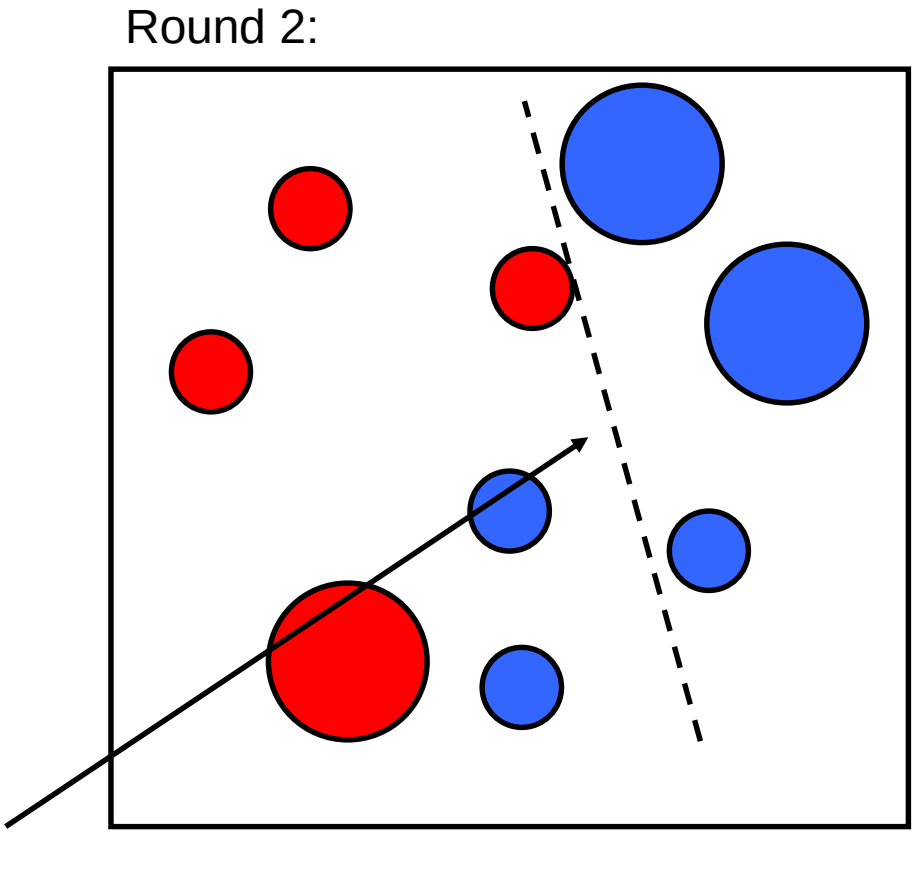


Boosting illustration

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.

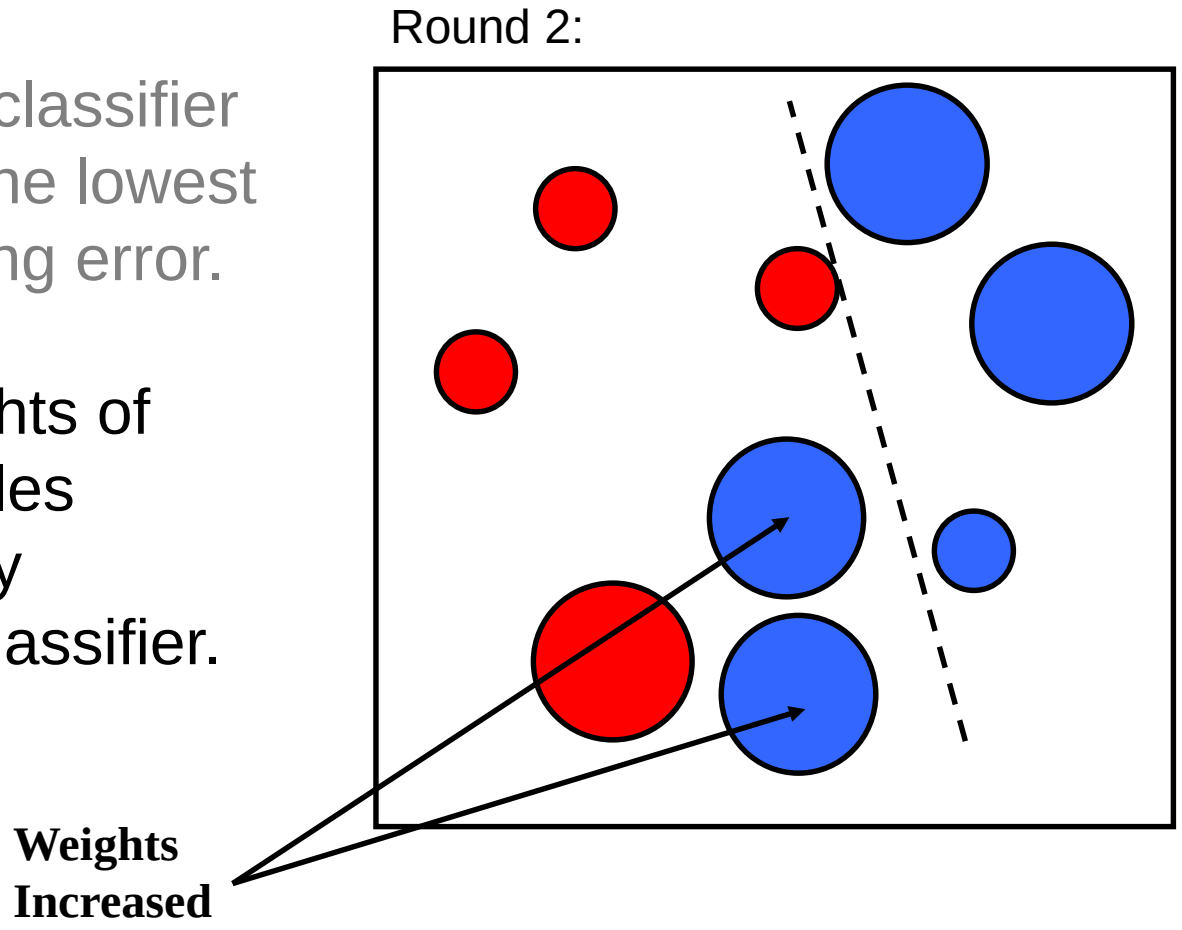


Boosting illustration

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.

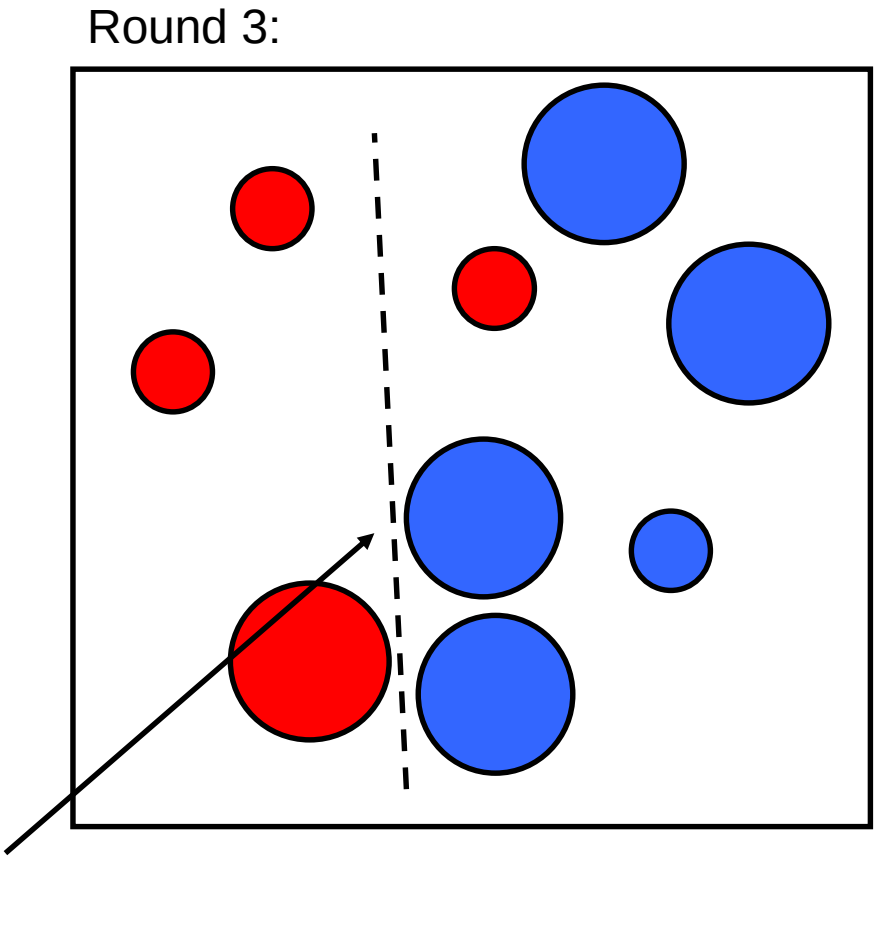


Boosting illustration

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.

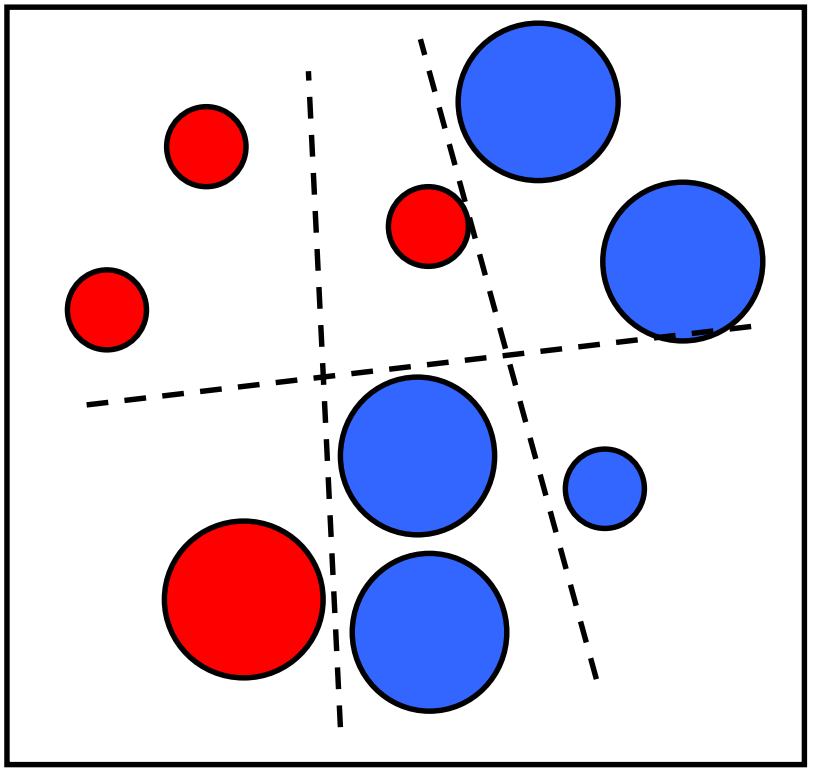


Boosting illustration

Compute final classifier as linear combination of all weak classifier.

Weight of each classifier is directly proportional to its accuracy.

Round 3:

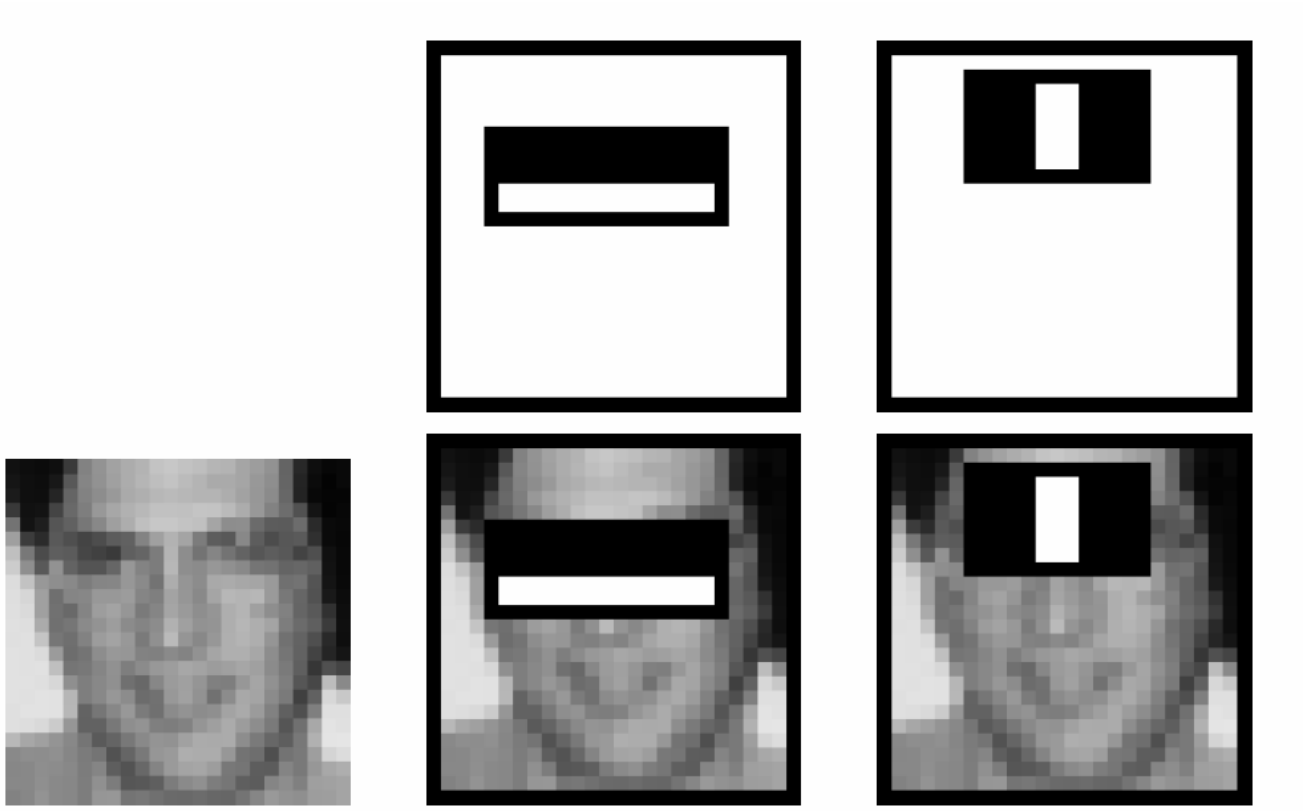


Exact formulas for re-weighting and combining weak learners depend on the particular boosting scheme (e.g., AdaBoost).

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 (180K)
- Select discriminative features that work well together

Final strong learner

$$h(\mathbf{x}) = \mathbf{sign}\left(\sum_{j=1}^{M} \alpha_j h_j(\mathbf{x})\right)$$

window

Weak learner

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

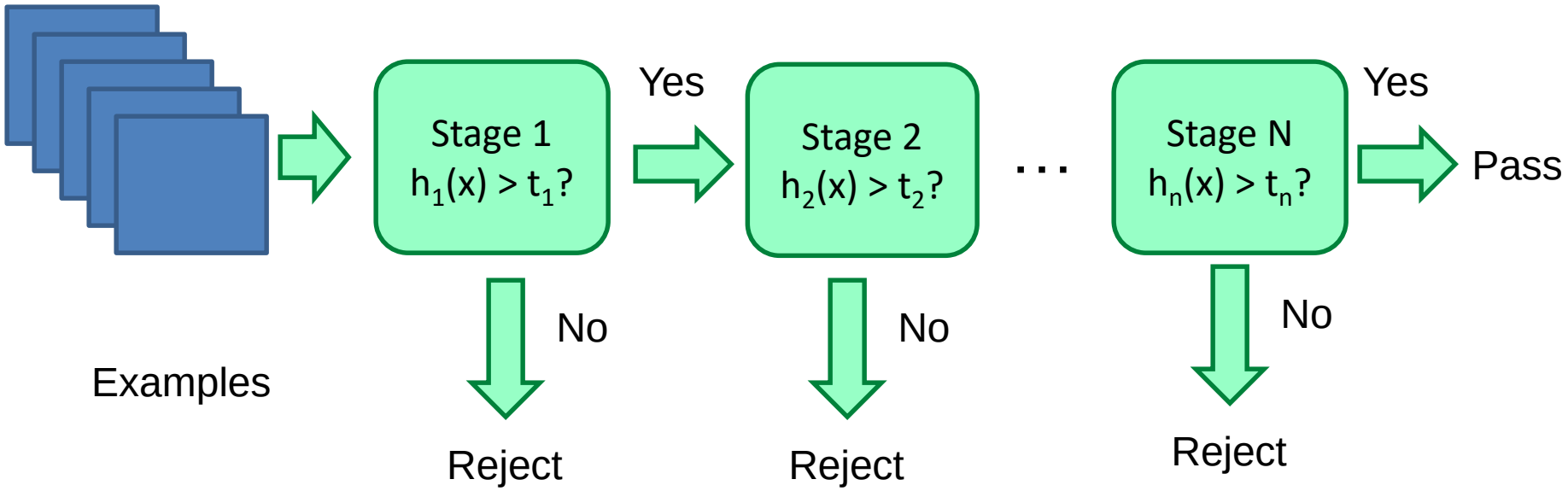
$$h(\boldsymbol{x}) = \text{sign}\left[\sum_{j=0}^{m-1} \alpha_j h_j(\boldsymbol{x})\right]. \tag{14.7}$$

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



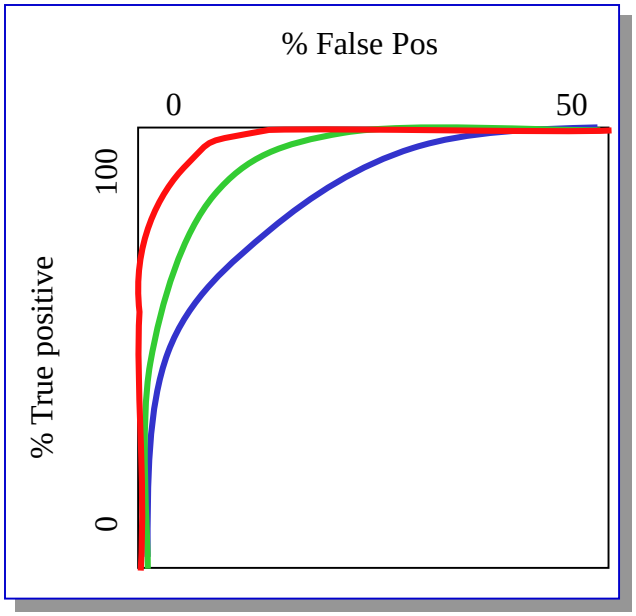
Fast classifiers early in cascade which reject many negative examples but detect almost all positive examples.

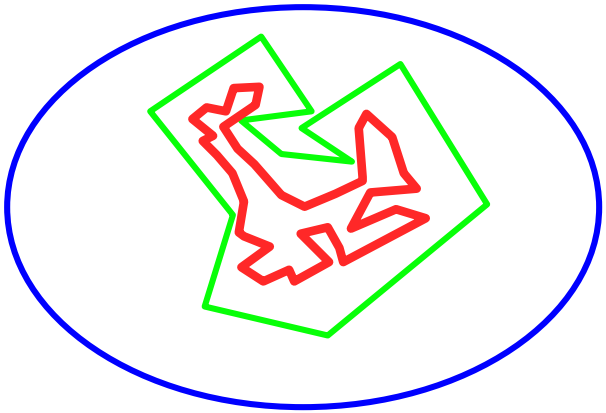
Slow classifiers later, but most examples don't get there.

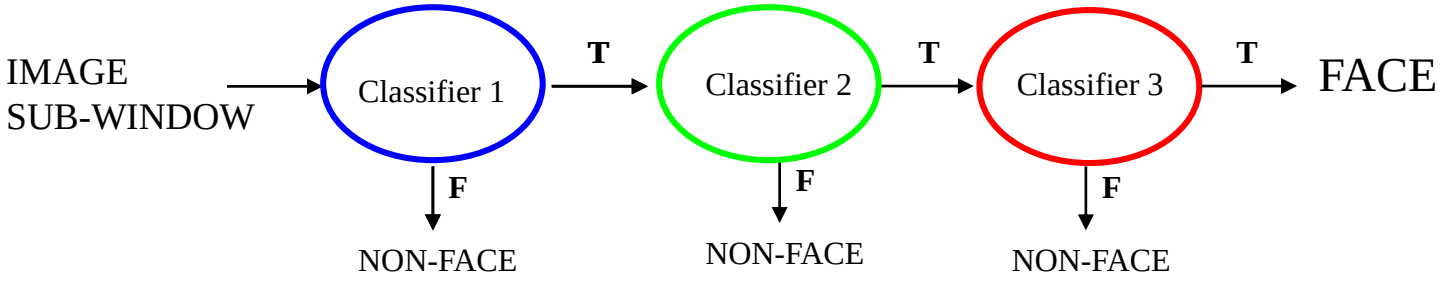
Attentional cascade

Chain classifiers that are progressively more complex and have lower false positive rates:

Receiver operating characteristic







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

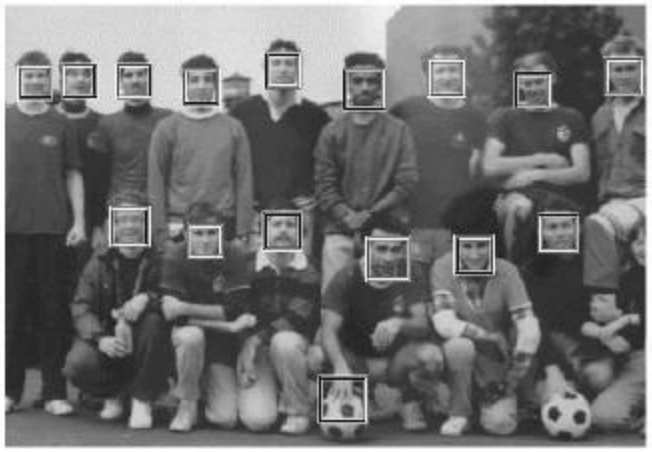
- 38 stages with 1, 10, 25, 50 … features
  - 6061 total used out of 180K candidates
  - 10 features evaluated on average
- Training Examples
  - 4916 positive examples
  - 10000 negative examples collected after each stage
- Scanning
  - Scale detector rather than image
  - Scale steps = 1.25 (factor 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)



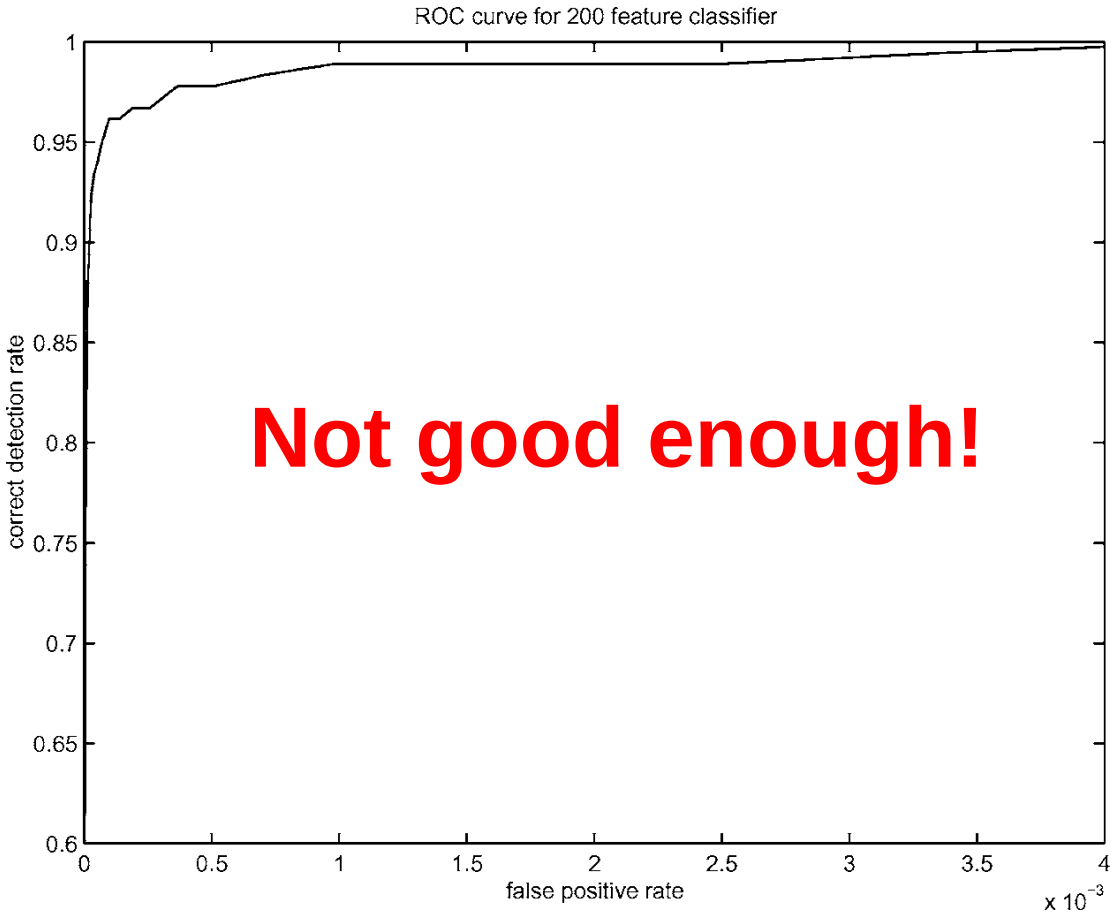
Detector \ False detections	10	31	50	65	78	95	167
Viola-Jones	76.1%	88.4%	91.4%	92.0%	92.1%	92.9%	93.9%
Viola-Jones (voting)	81.1%	89.7%	92.1%	93.1%	93.1%	93.2 %	93.7%
Rowley-Baluja-Kanade	83.2%	86.0%	-	-	-	89.2%	90.1%
Schneiderman-Kanade	-	-	-	94.4%	-	-	-
Roth-Yang-Ahuja	-	-	-	-	(94.8%)	-	-

MIT + CMU face dataset

Boosting for face detection

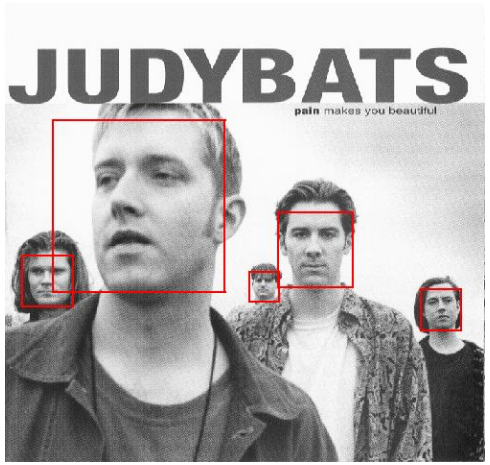
- A 200-feature classifier can yield 95% detection rate and a false positive rate of 1 in 14084

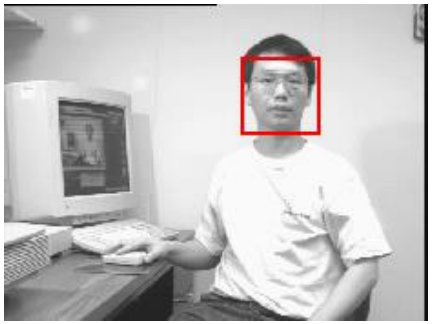
True positive rate

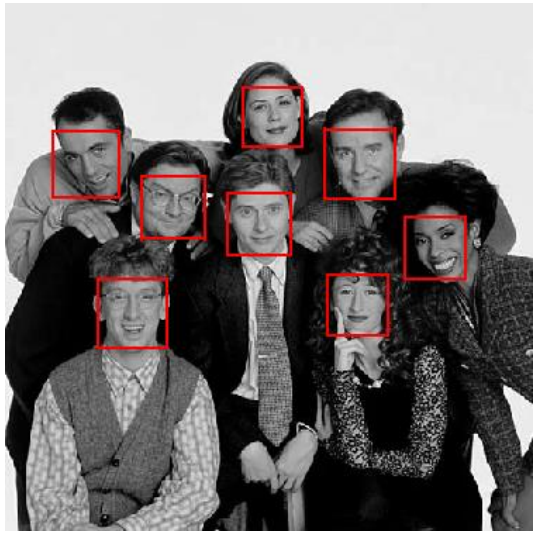


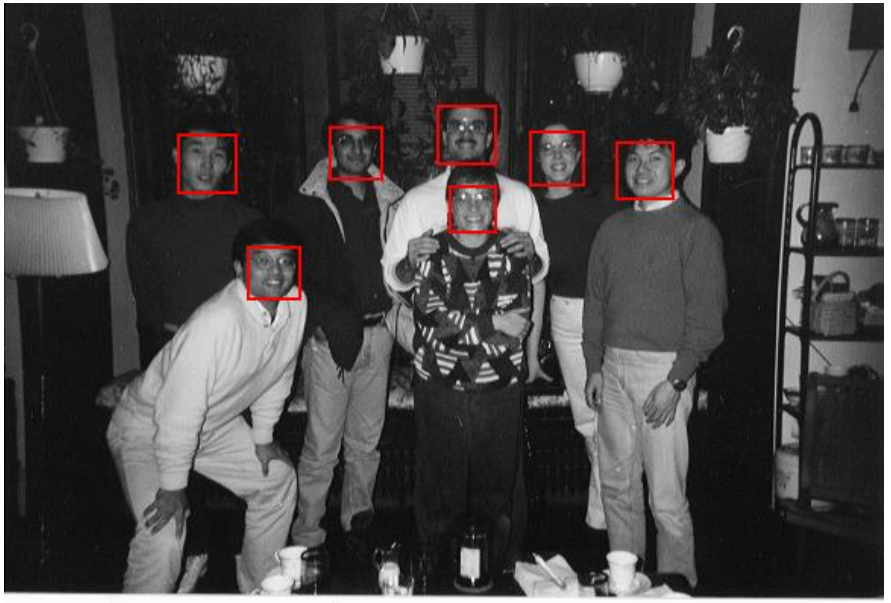
Receiver operating characteristic (ROC) curve

Output of Face Detector on Test Images



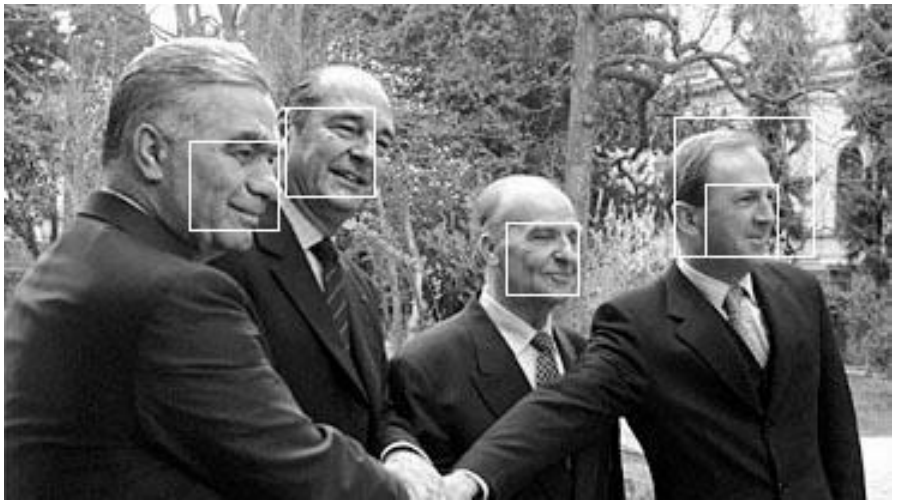






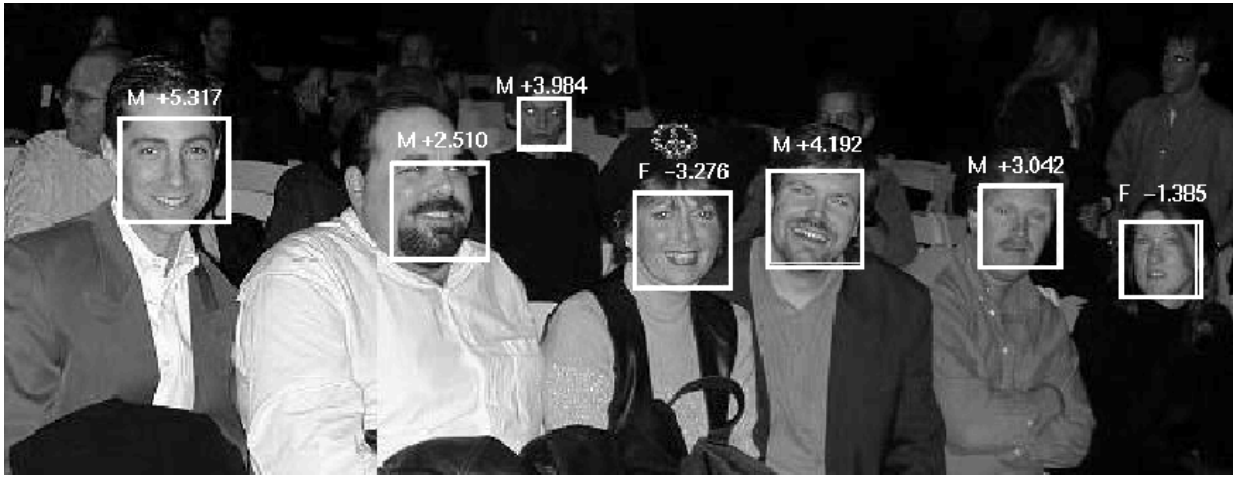
Other detection tasks

Facial Feature Localization

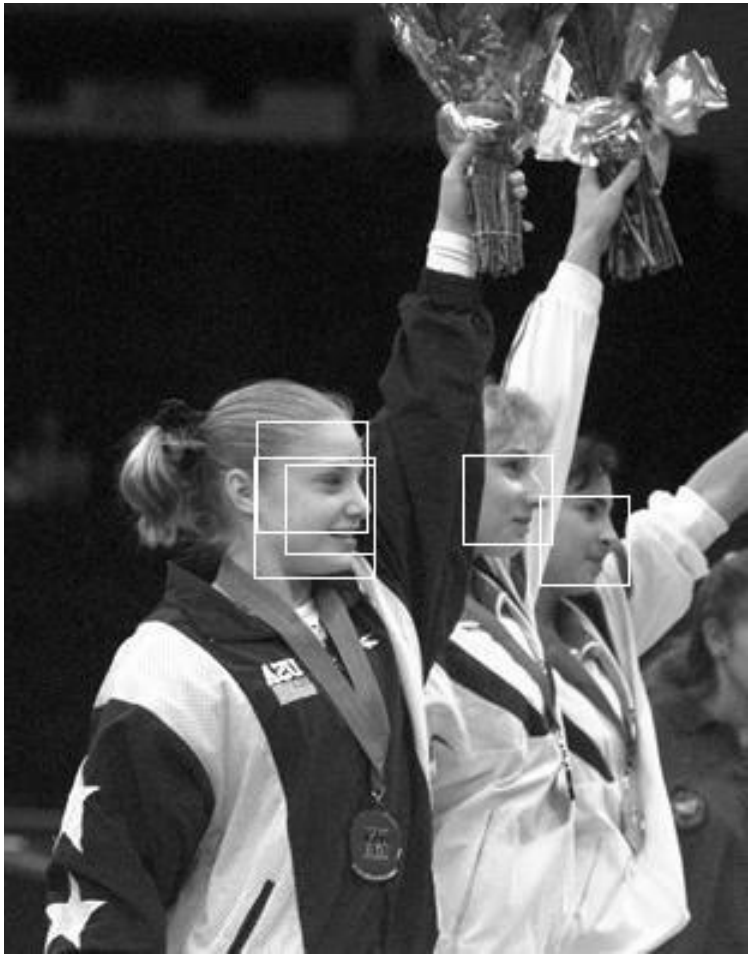


Profile Detection

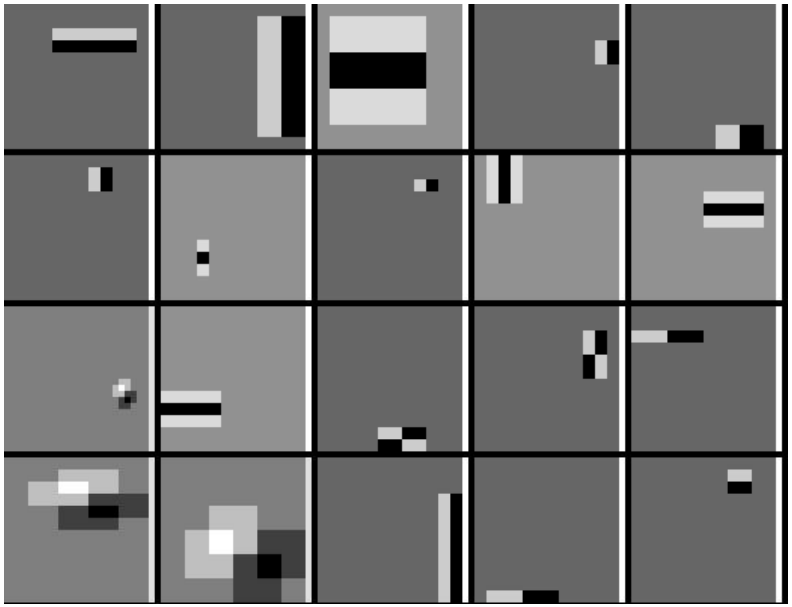
Male vs. female



Profile Detection



Profile Features



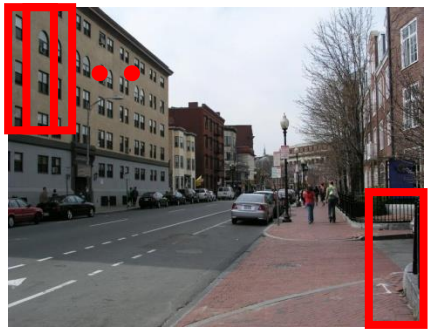
Live demo

Summary: Viola/Jones detector

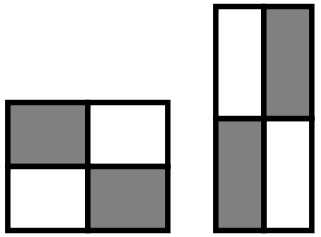
- Rectangle features
- Integral images for fast computation
- Boosting for feature selection
- Attentional cascade for fast rejection of negative windows

Things to remember

- Sliding window for search

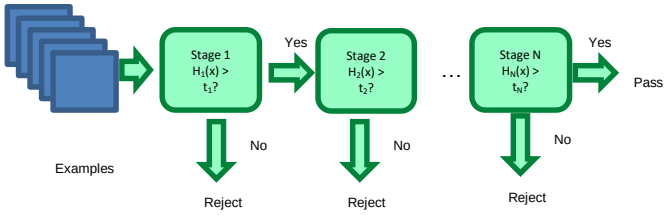


- Features based on differences of intensity (gradient, wavelet, etc.)
  - Excellent results = careful feature design



- Boosting for feature selection

- Integral images, cascade for speed



- Bootstrapping to deal with many, many negative examples