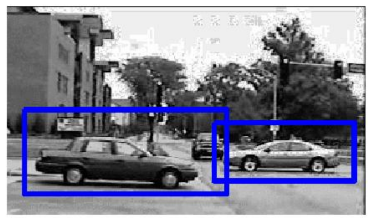
Object Detecion

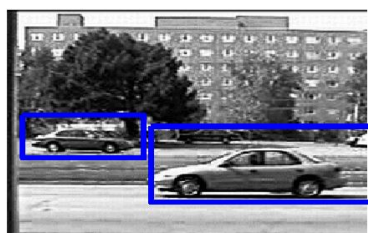
<Vision System>

Department of Robot Engineering
Prof. Younggun Cho

Concept of Detection

- Given a category (ex. face, car, body), localizing objects in images.





Datasets

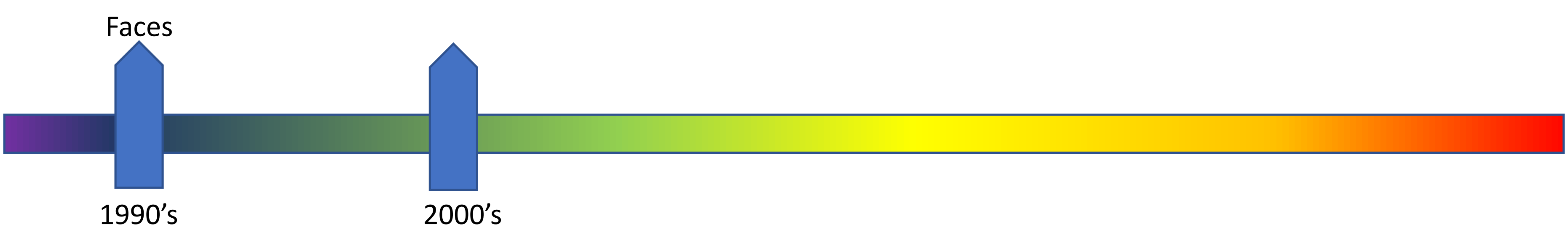
- Face detection
- One category: face
- Frontal faces
- Fairly rigid, unoccluded

1990's

Human Face Detection in Visual Scenes. H. Rowley, S. Baluja, T. Kanade. 1995.

Pedestrians

- One category: pedestrians
- Slight pose variations and small distortions
- Partial occlusions



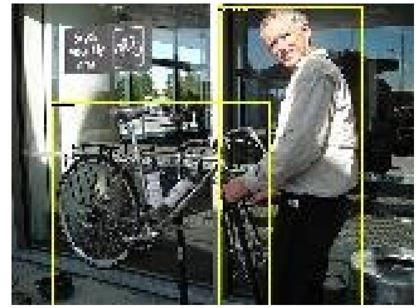
Histograms of Oriented Gradients for Human Detection. N. Dalal and B. Triggs. CVPR 2005

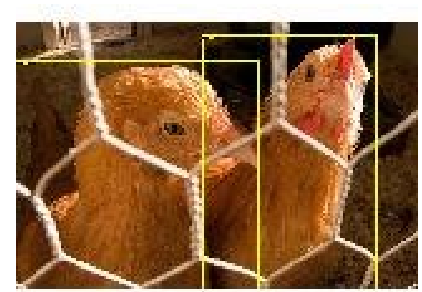
PASCAL VOC

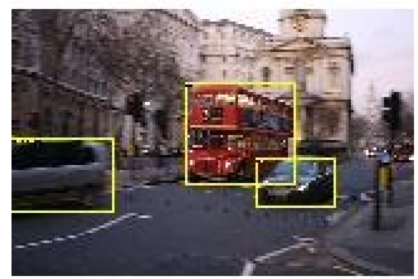
http://host.robots.ox.ac.uk/pascal/VOC/

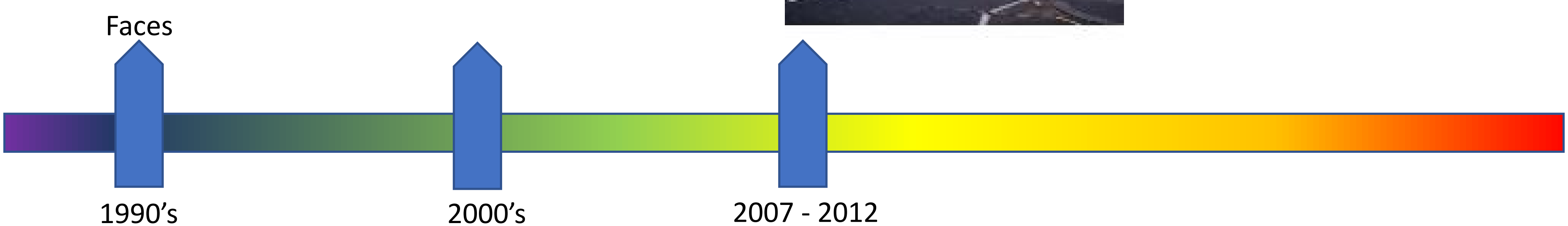
visual object class

- 20 categories
- 10K images
- Large pose variations, heavy occlusions
- Generic scenes
- Cleaned up performance metric









CVDF

Microsoft

facebook

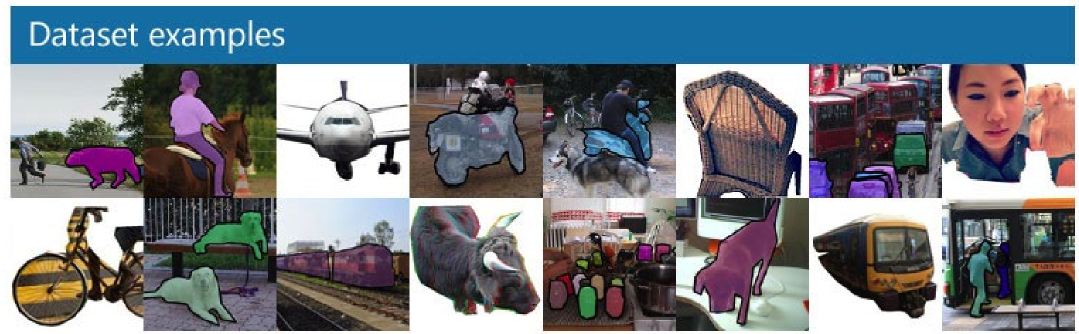
Mighty Ai

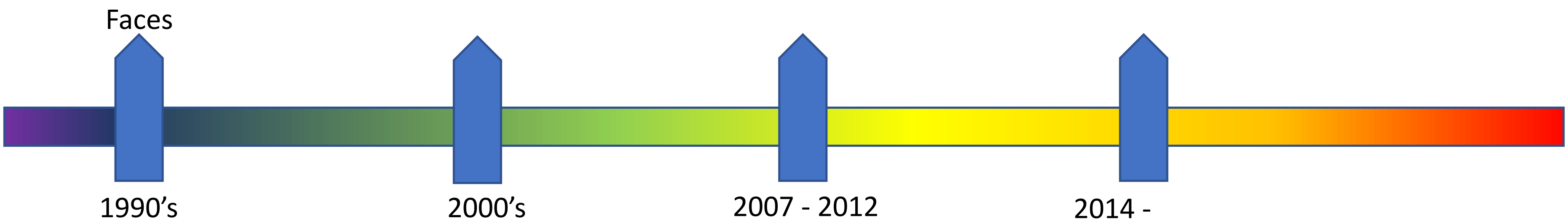
Coco

http://cocodataset.org/#home

Common objects in context

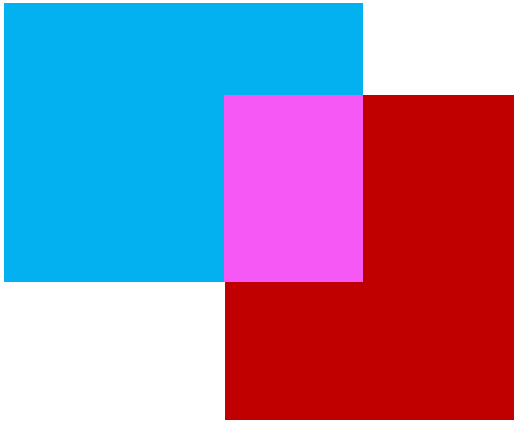
- 80 diverse categories
- 100K images
- Heavy occlusions, many objects per image, large scale variations





Evaluation metric

Matching detections to ground truth



$$IoU(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

intersection over union

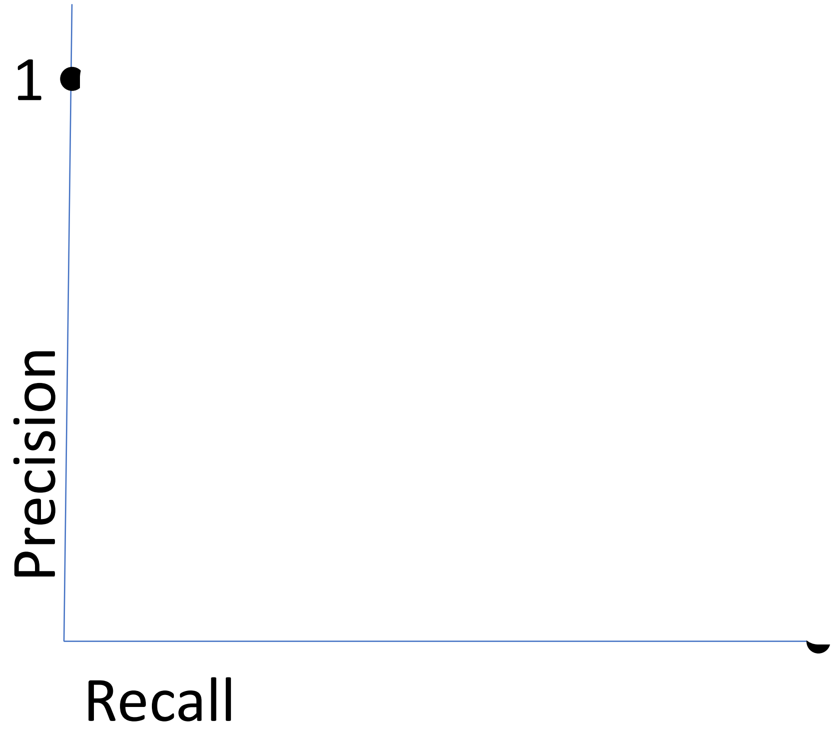
Matching detections to ground truth

- Match detection to most similar ground truth
  - highest IoU
- If IoU > 50%, mark as correct
- If multiple detections map to same ground truth, mark only one as correct
- **Precision** = #correct detections / total detections
- **Recall** = #ground truth with matched detections / total ground truth

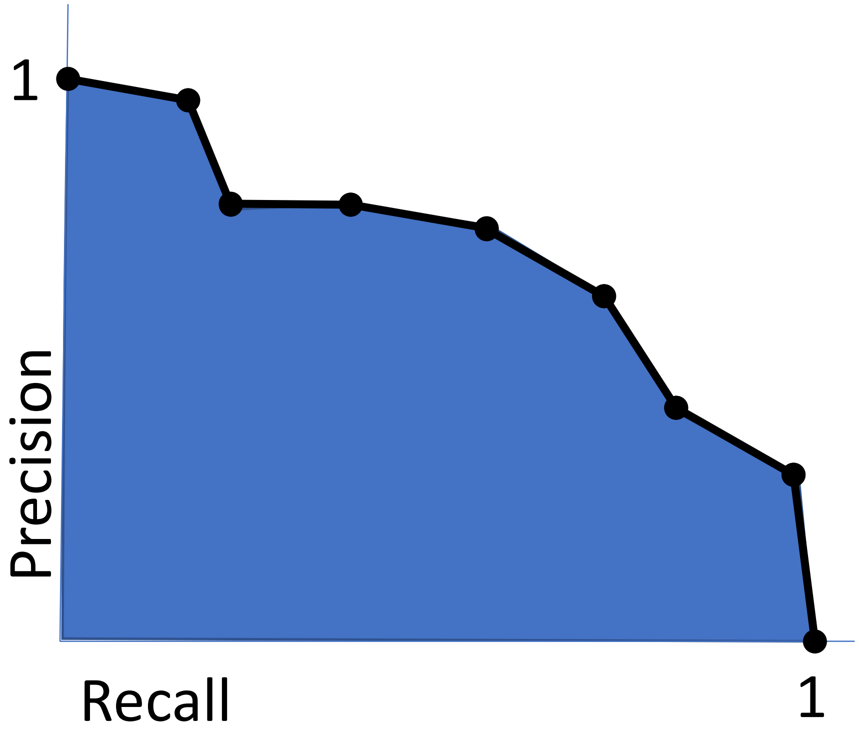
Tradeoff between precision and recall

- ML usually gives scores or probabilities, so threshold
- Too low threshold → too many detections
→ low precision, high recall
- Too high threshold → too few detections
→ high precision, low recall
- Right tradeoff depends on application
  - Detecting cancer cells in tissue: need high recall
  - Detecting edible mushrooms in forest: need high precision

Average precision



Average Precision (AP)



Average average precision

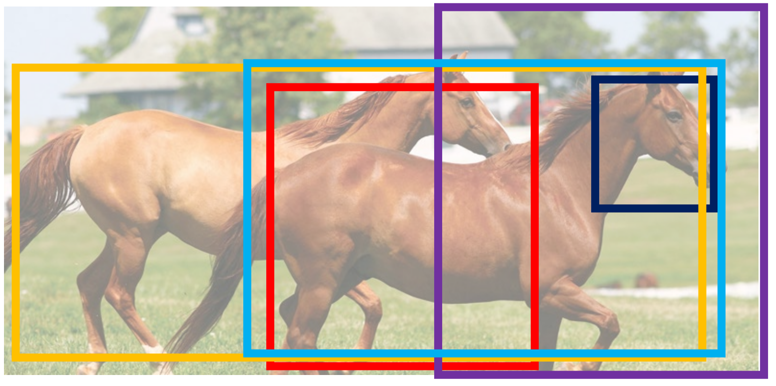
- AP marks detections with overlap > 50% as correct
- But may need better localization
- *Average* AP across multiple overlap thresholds
- Confusingly, still called average precision
- Introduced in COCO

Mean and category-wise AP

- Every category evaluated independently
- Typically report mean AP averaged over all categories
- Confusingly called “mean Average Precision”, or “mAP”

Why is detection hard(er)?

- Precise localization



Why is detection hard(er)?

- Much larger impact of pose

Why is detection hard(er)?

- Occlusion makes localization difficult

Why is detection hard(er)?

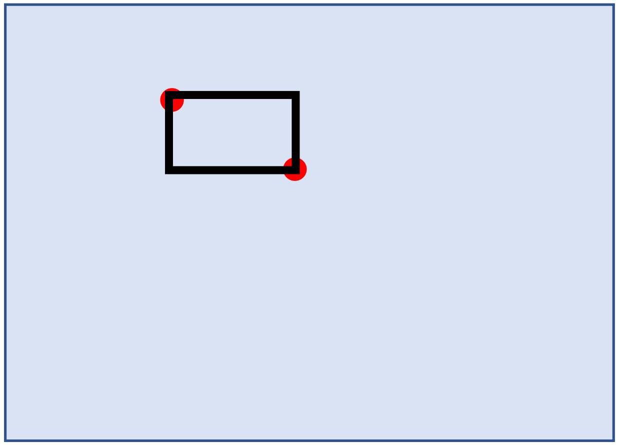
- Counting

Why is detection hard(er)?

- Small objects

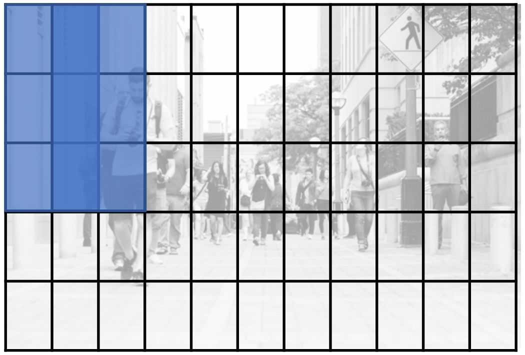
Detection as classification

- Run through every possible box and classify
  - Well-localized object of class k or not?
- How many boxes?
  - Every pair of pixels = 1 box
  - $\binom{N}{2}$ = O(N^2)
  - For 300 x 500 image, N = 150K
  - 2.25 x 10^{10} boxes!
- Related challenge: almost all boxes are negative!



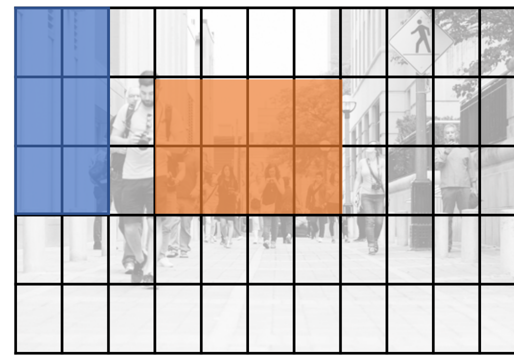
Idea 1: scanning window

- Fix size
- Fix stride
- Crop boxes and classify
- Alternatively
  - Compute collection of feature maps
  - Convolve with filter



Multiple object sizes

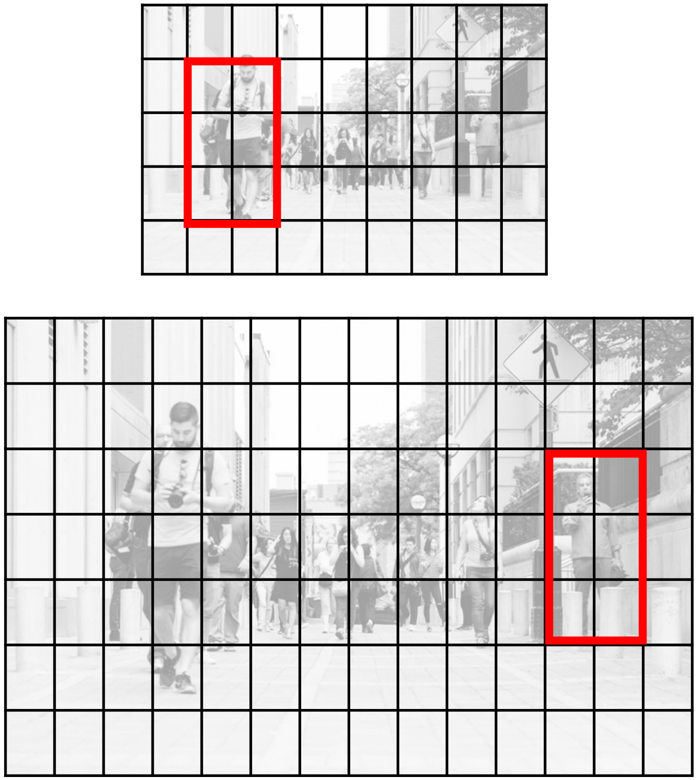
- Objects can appear at any size
- *Discretize* set of sizes into a few different sizes
  - Sometimes called “anchors”
- Train separate classifier for each size



Dealing with large scale changes

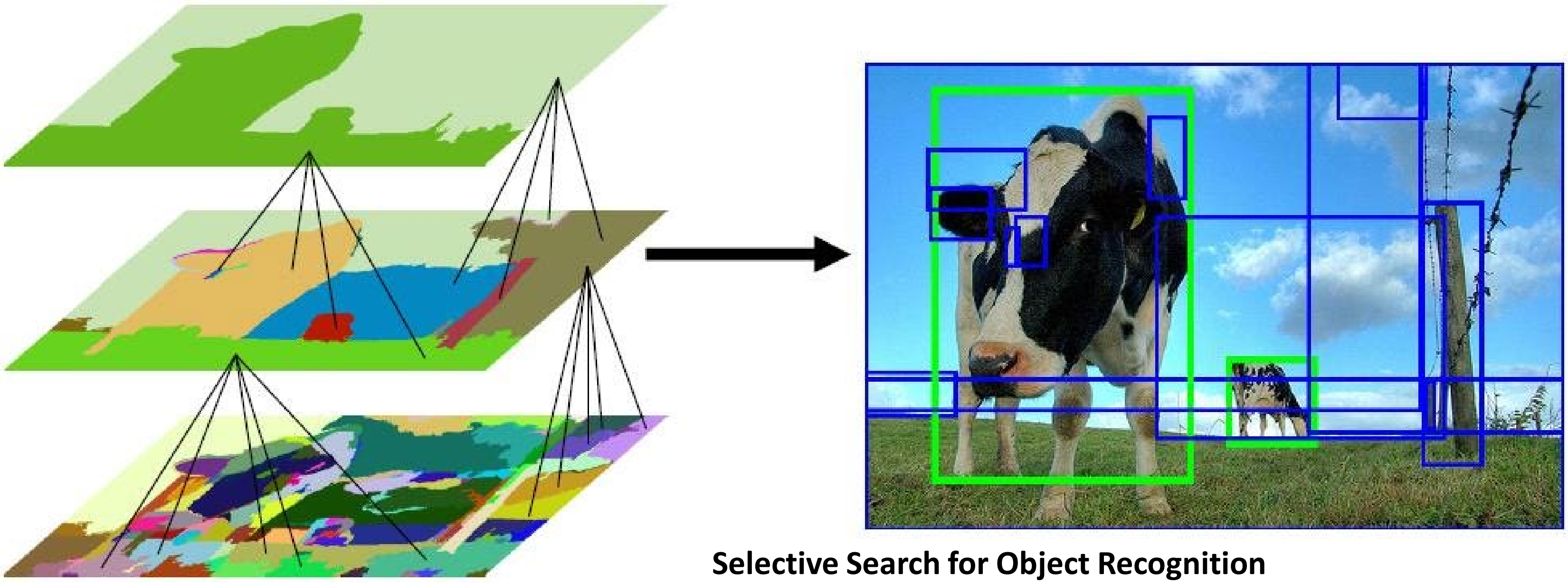
Dealing with large scale changes

- Use an image pyramid
- Run same detector at multiple scales
- Take union of results



Idea 2: Object proposals

- Use segmentation to produce ~5K “candidates”



Selective Search for Object Recognition
J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, A. W. M. Smeulders
In International Journal of Computer Vision 2013.

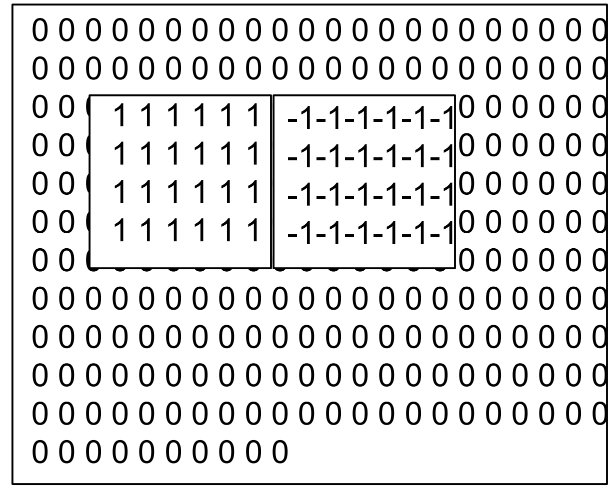
Object proposals

- Basic idea: use grouping cues to identify segments that are likely to be objects
- Multiple versions
  - Do graph cuts with different seeds
  - Oversegment and then combinatorially group nearby objects

Example 1: Face detection

- Slides adapted Grauman & Liebe’s tutorial
  - http://www.vision.ee.ethz.ch/~bleibe/teaching/tutorial-aaai08/
- Also see Paul Viola’s talk (video)
  - http://www.cs.washington.edu/education/courses/577/04sp/contents.html#DM

Viola & Jones Face Detector: Rectangle filters



Why rectangles?

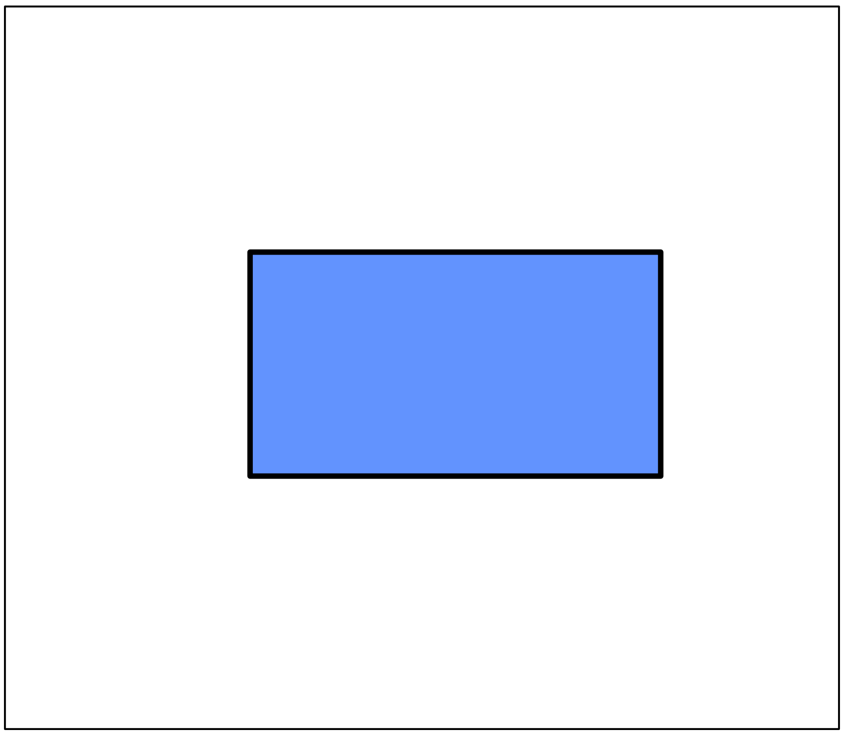
Answer: very very fast to compute

- Trick: *integral images* (aka *summed-area-tables*)

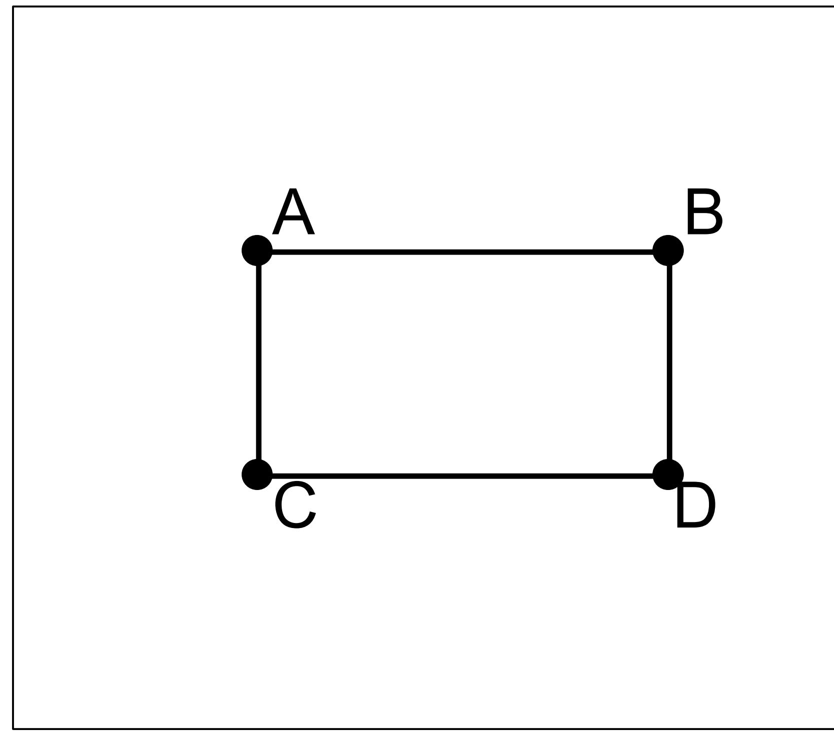
P. Viola and M. Jones. *Rapid object detection using a boosted cascade of simple features.* CVPR 2001.

Integral images

What's the sum of pixels in the blue rectangle?



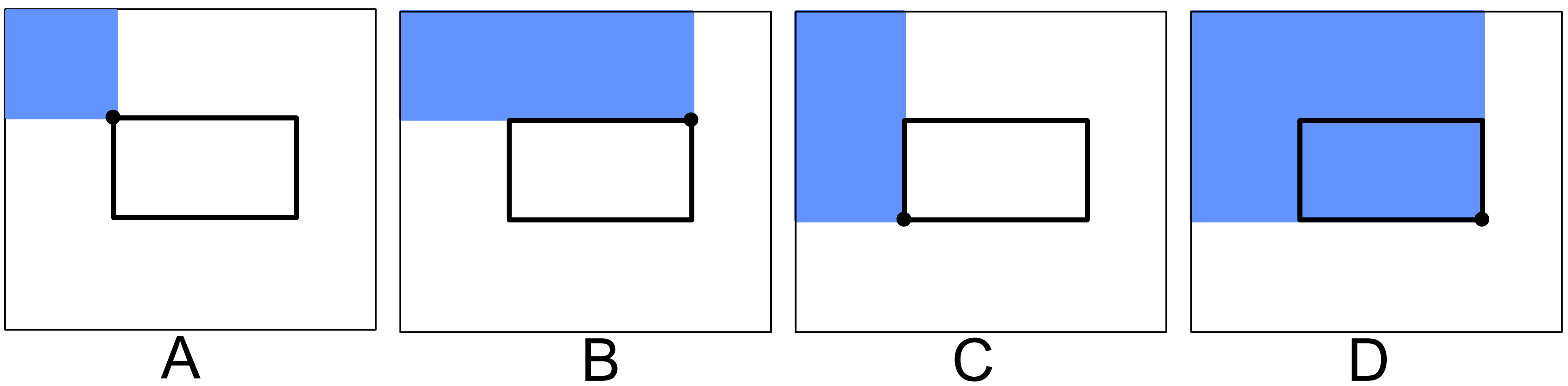
input image



integral image

Integral images

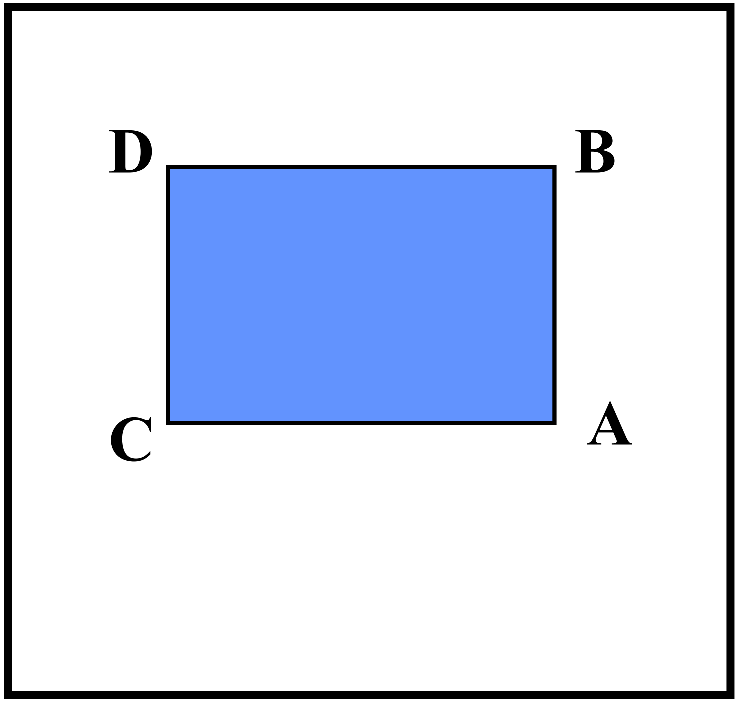
What's the sum of pixels in the rectangle?



Computing sum within a rectangle

- Let A,B,C,D be the values of the integral image at the corners of a rectangle
- Then the sum of original image values within the rectangle can be computed as:

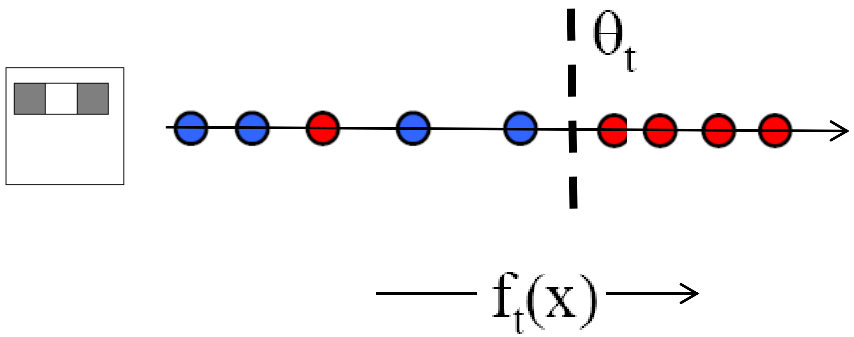
  sum = A – B – C + D
- Only 3 additions are required for any size of rectangle!



Lana Lazebnik

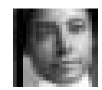
Filter as a classifier

How to convert the filter into a classifier?



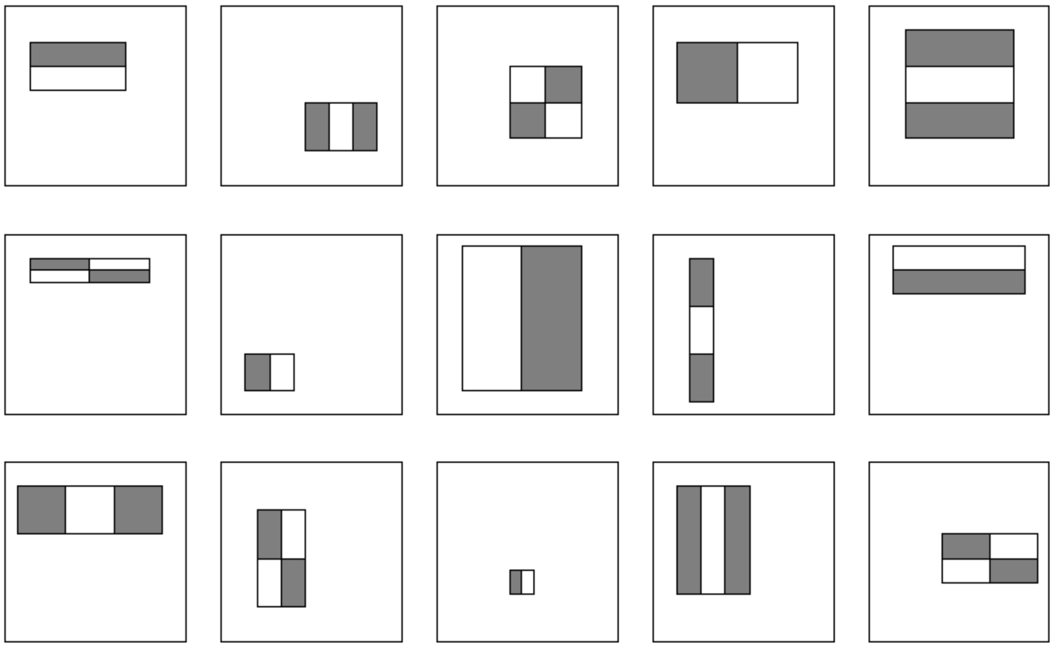
Outputs of a rectangle feature on faces and non-faces.

Resulting weak classifier:



$$h_t(x) = \begin{cases} +1 & \text{if } f_t(x) > \theta_t \\ -1 & \text{otherwise} \end{cases}$$

Finding the best filters...

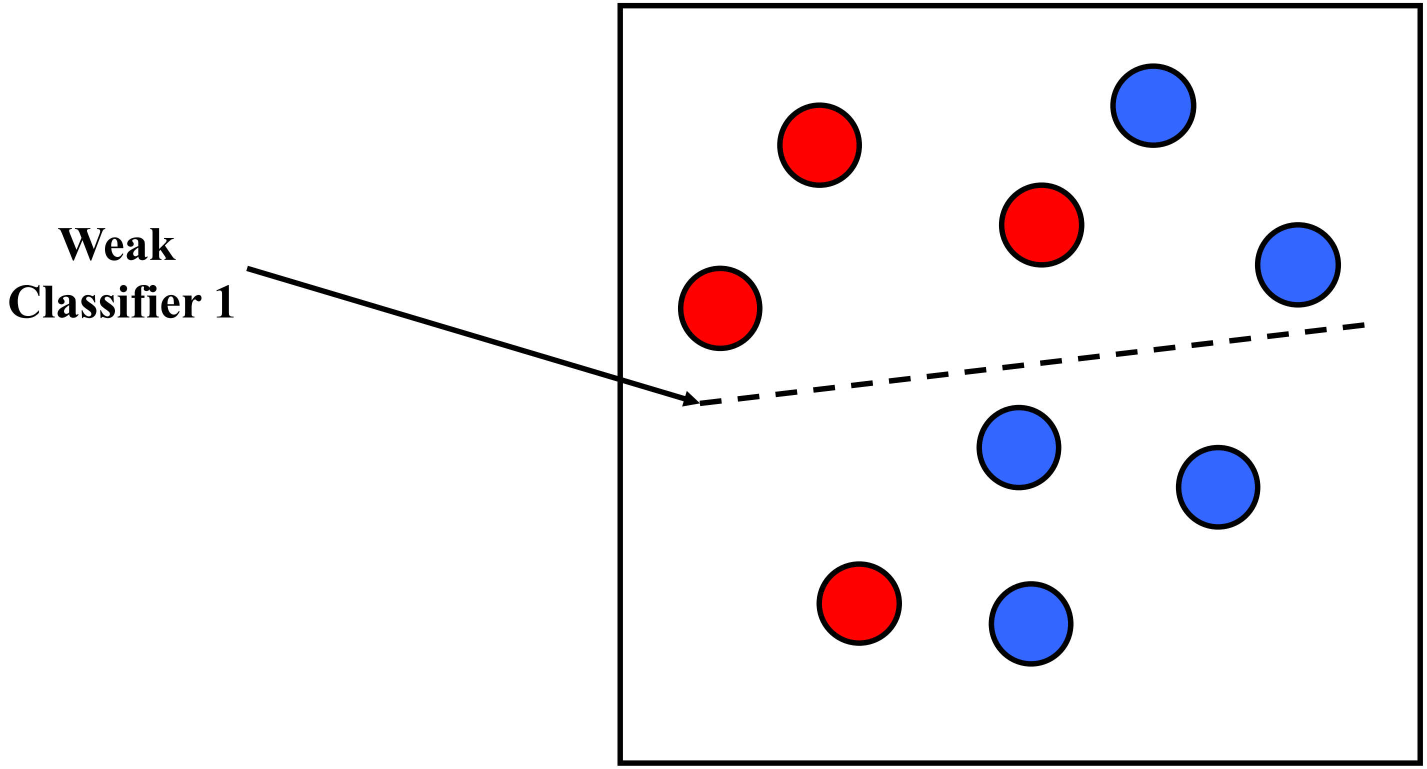


Considering all possible filter parameters: position, scale, and type:

180,000+ possible filters associated with each 24 x 24 window

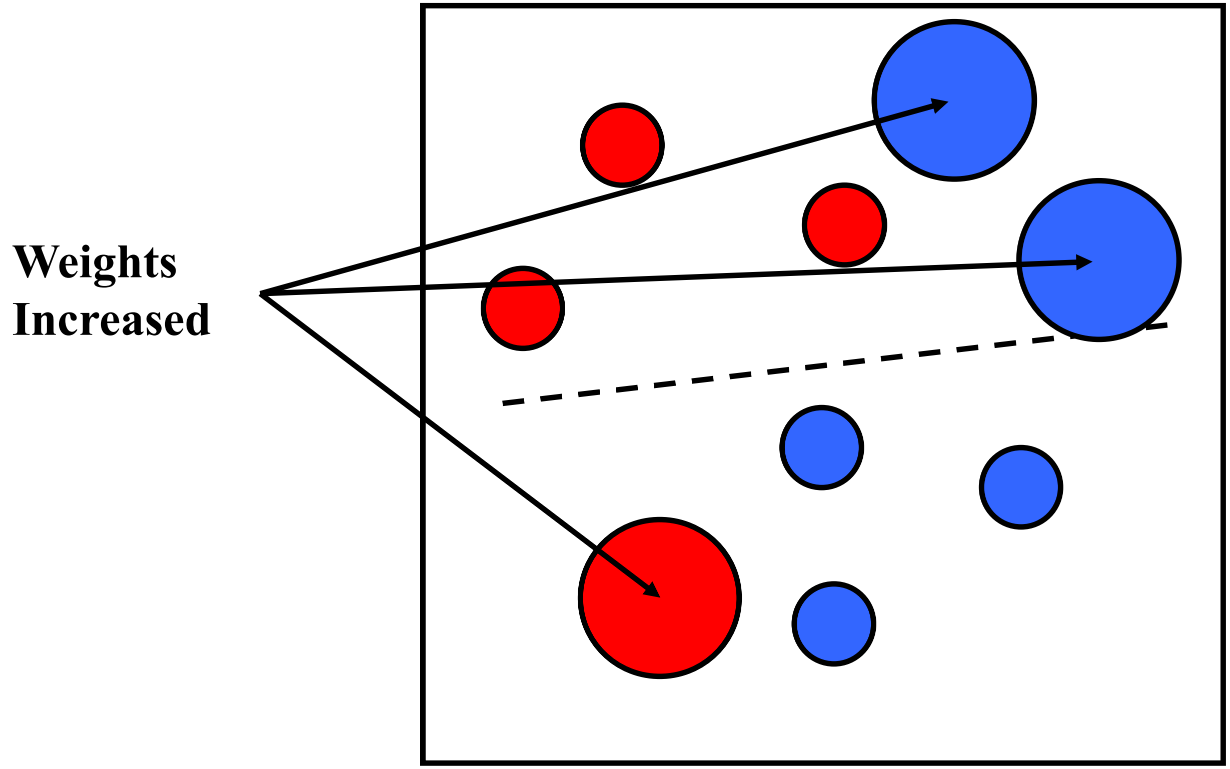
Which of these filters(s) should we use to determine if a window has a face?

Boosting

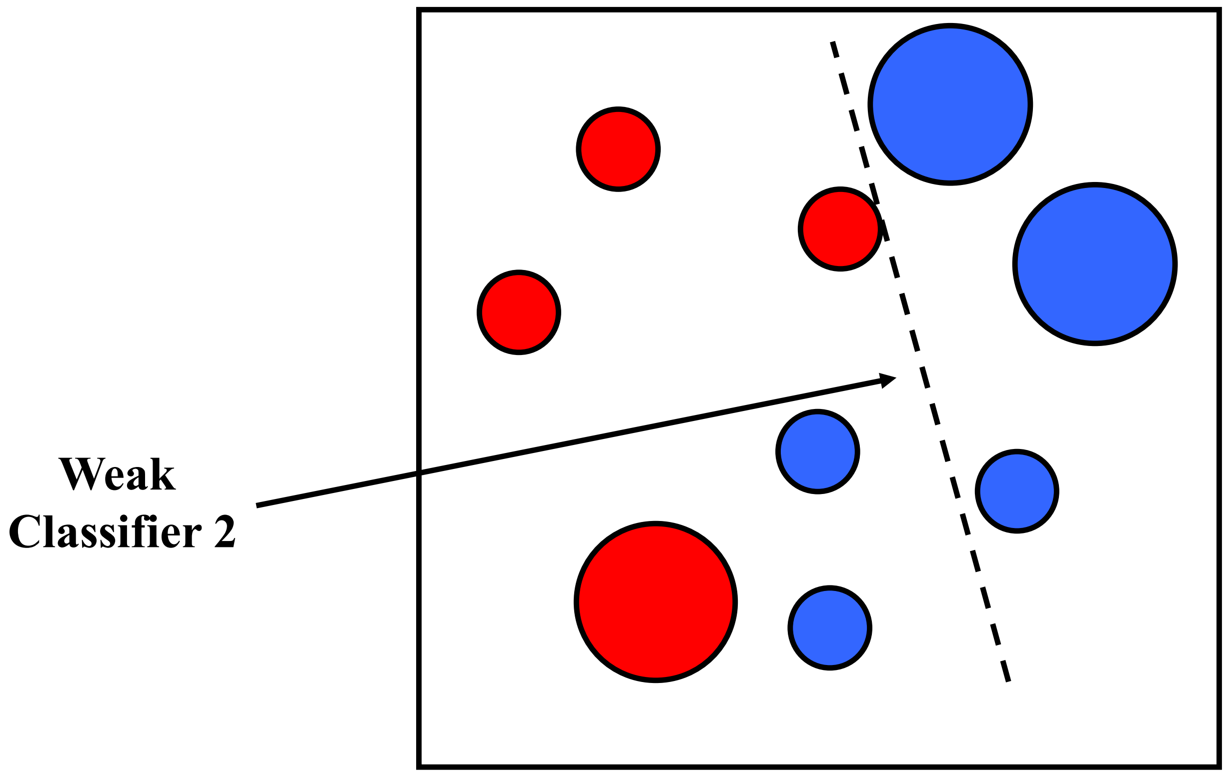


Slide credit: Paul Viola

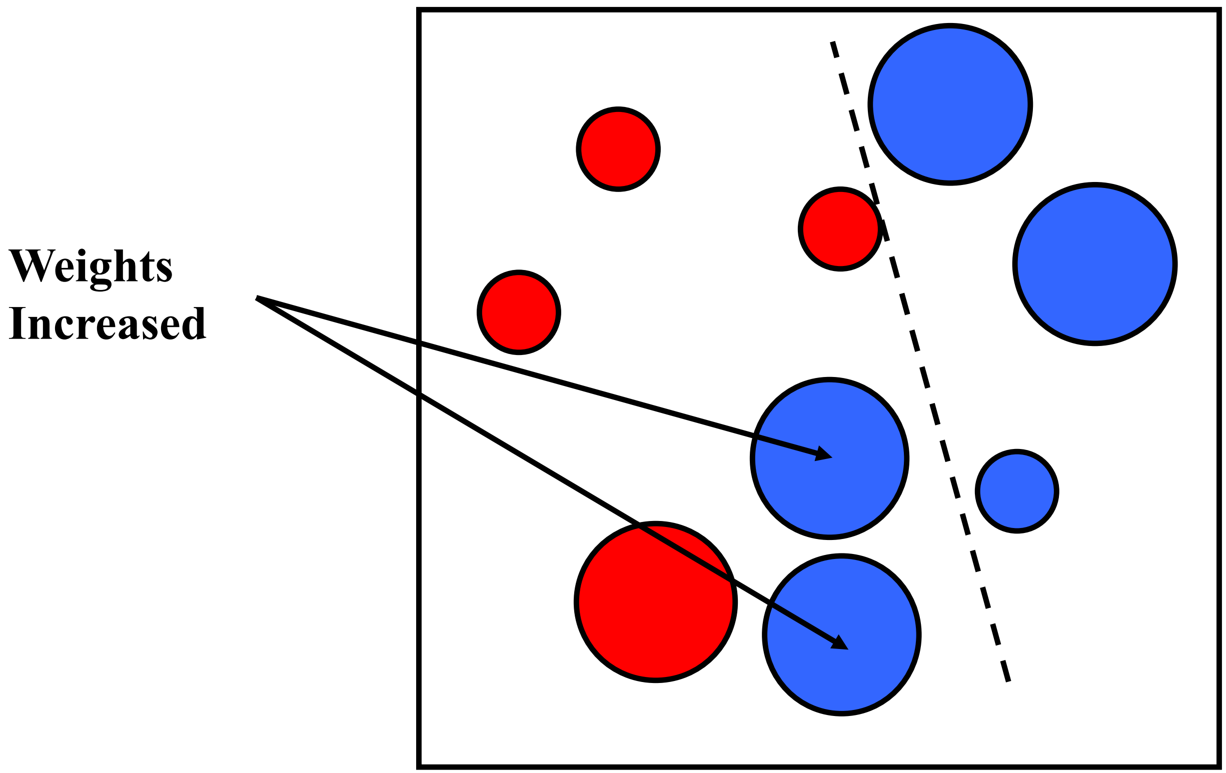
Boosting



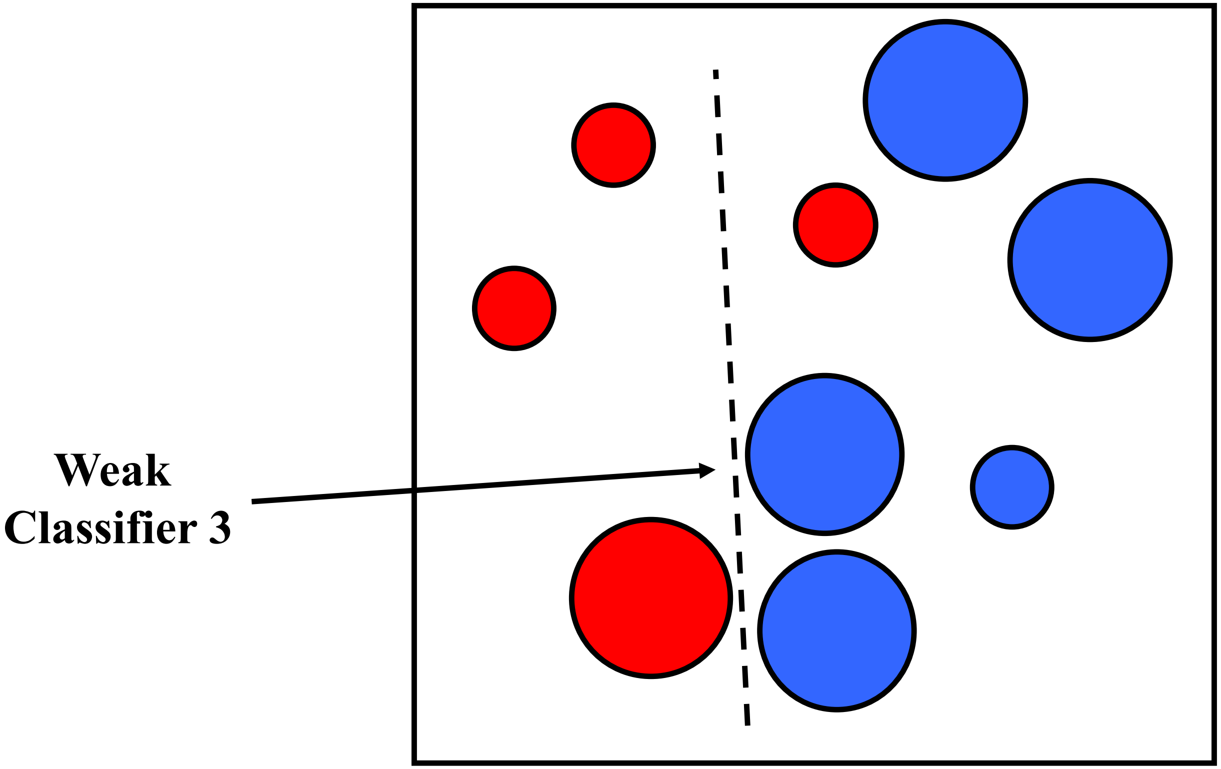
Boosting



Boosting

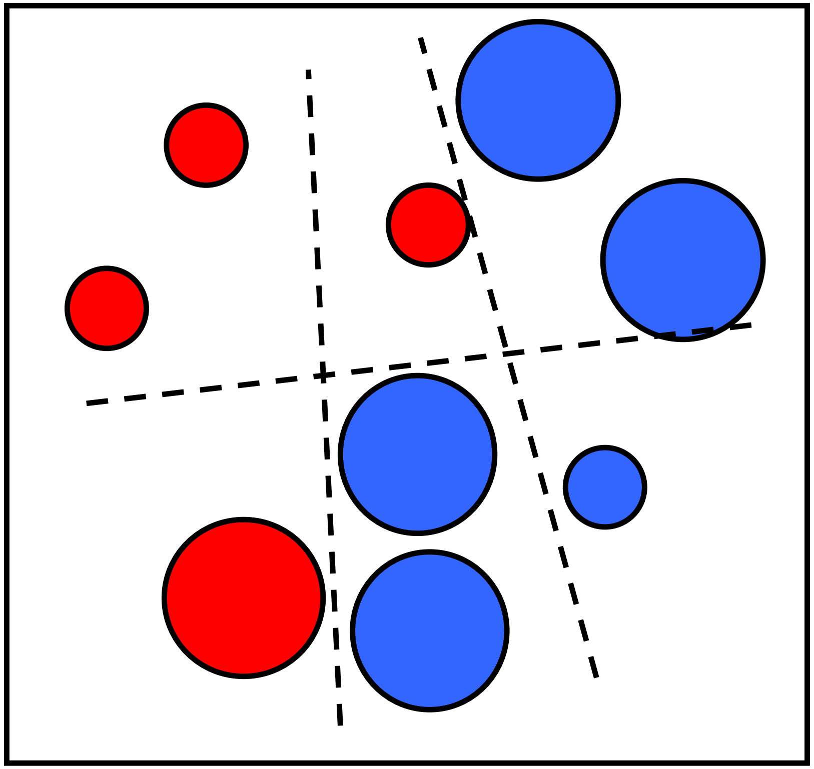


Boosting



Boosting

Final classifier is a combination of weak classifiers



Boosting: training

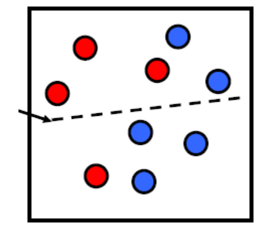
- Initially, weight each training example equally
- In each boosting round:
  - find the weak classifier with lowest *weighted* training error
  - raise weights of training examples misclassified by current weak classifier
- Final classifier is linear combination of all weak classifiers
  - weight of each learner is directly proportional to its accuracy)
- Exact formulas for re-weighting and combining weak classifiers depend on the particular boosting scheme

Slide credit: Lana Lazebnik

AdaBoost Algorithm

- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \ldots, T$:
  1. Normalize the weights,
  $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
  so that w_t is a probability distribution.
  2. For each feature, j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t, $\epsilon_j = \sum_i w_i \, |h_j(x_i) - y_i|$.
  3. Choose the classifier, h_t, with the lowest error ϵ_t.
  4. Update the weights:
  $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$
  where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- The final strong classifier is:
$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$
where $\alpha_t = \log \frac{1}{\beta_t}$

Start with uniform weights on training examples



$\{x_1, \ldots x_n\}$

For T rounds

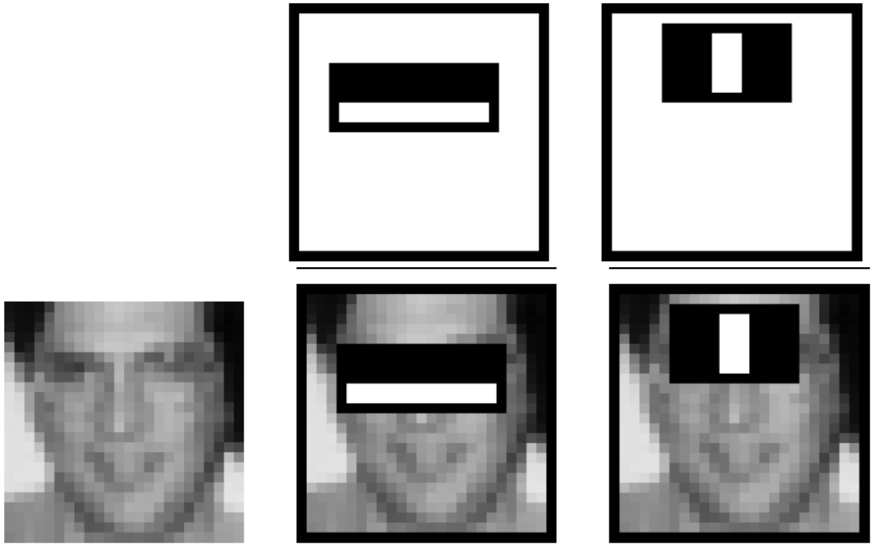
Evaluate *weighted* error for each feature, pick best.

Re-weight the examples:
Incorrectly classified -> more weight
Correctly classified -> less weight

Final classifier is combination of the weak ones, weighted according to error they had.

Freund & Schapire 1995

Viola-Jones Face Detector: Results

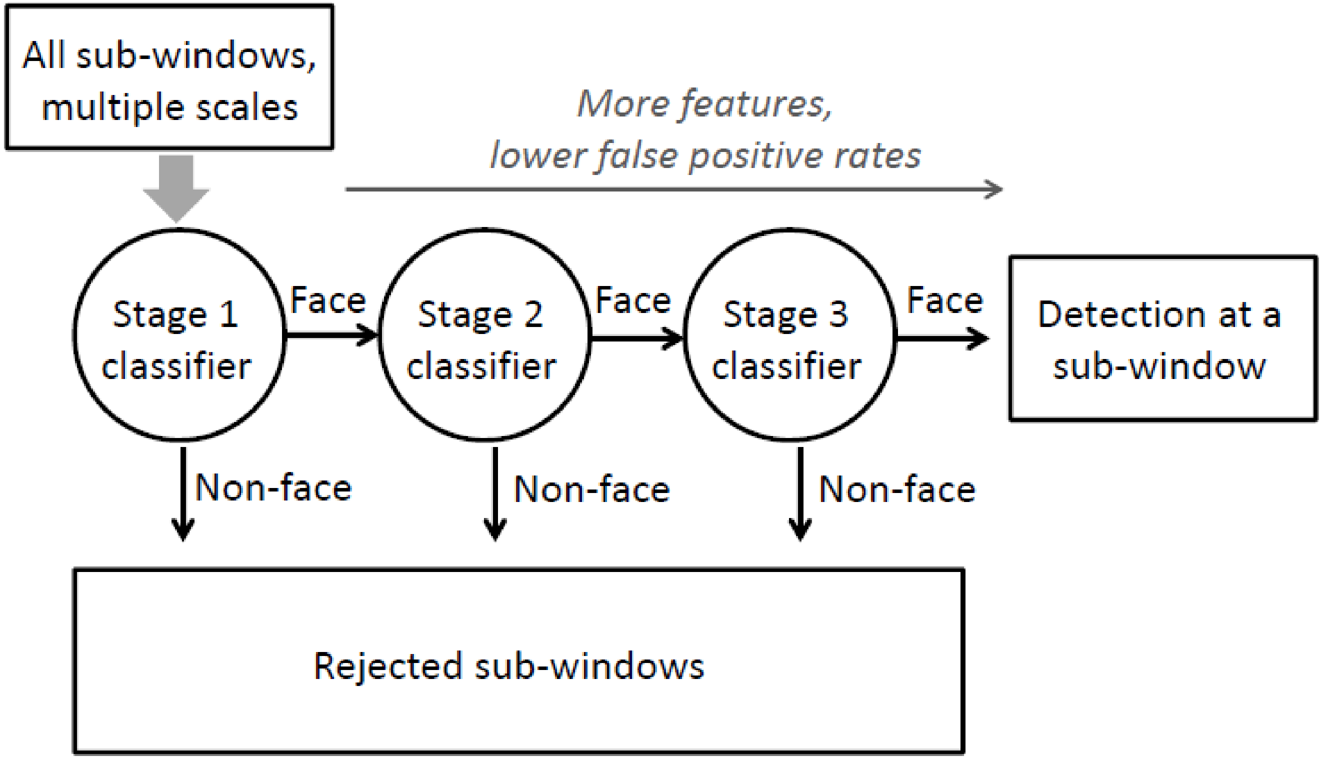


First two features selected

Robust Real-Time Face Detection ,IJCV, 2004

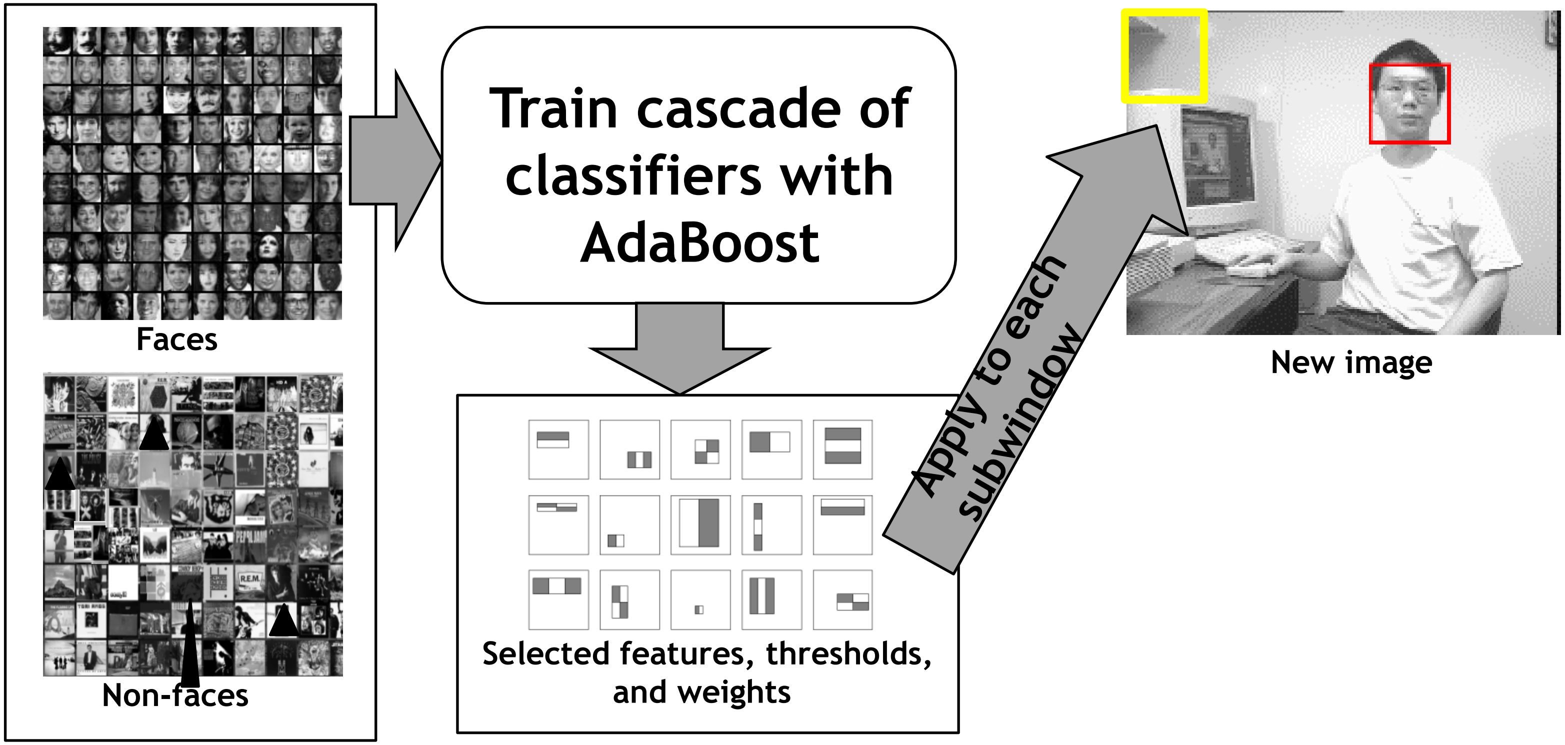
- Even if the filters are fast to compute, each new image has a lot of possible windows to search.
- How to make the detection more efficient?

Cascading classifiers for detection



- Form a *cascade* with low false negative rates early on
- Apply less accurate but faster classifiers first to immediately discard windows that clearly appear to be negative

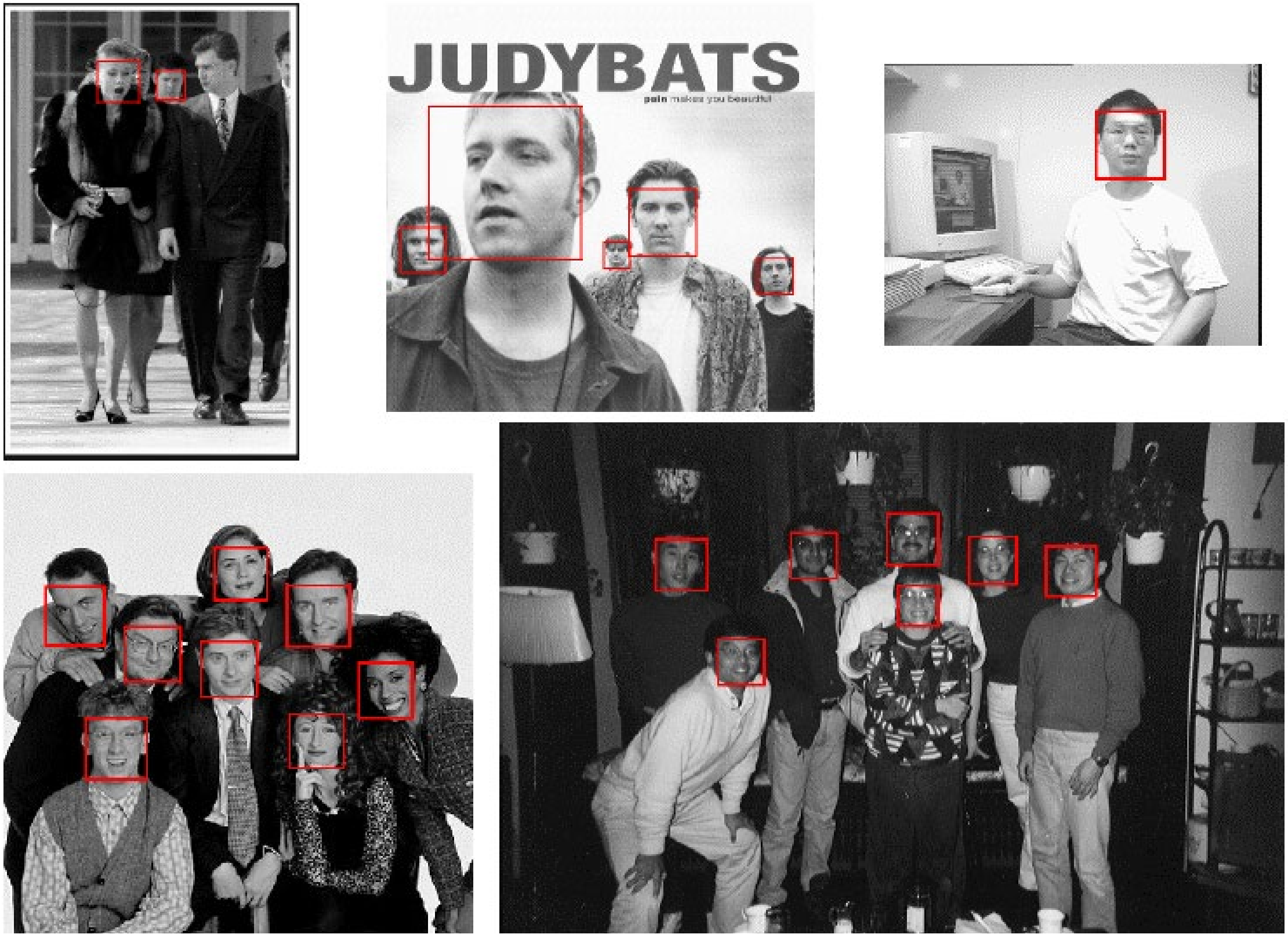
Viola-Jones detector: summary



- Train with 5K positives, 350M negatives
- Real-time detector using 38 layer cascade
- 6061 features in all layers

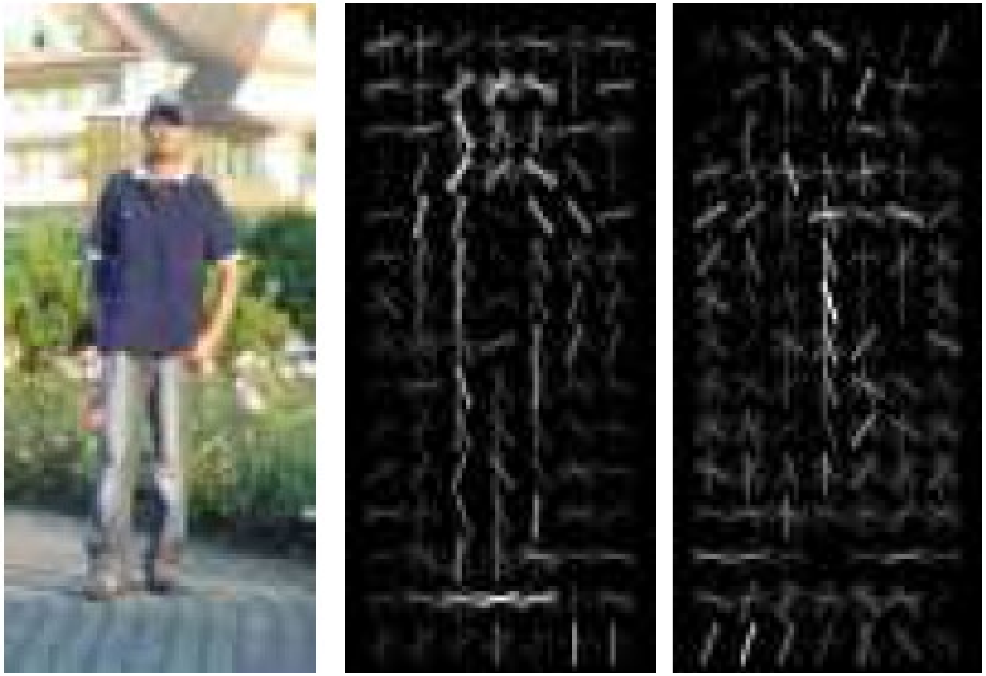
- [Implementation available in OpenCV: http://www.intel.com/technology/computing/opencv/]

Viola-Jones Face Detector: Results



Example 2: Pedestrian detection

- Detecting upright, walking humans also possible using sliding window's appearance/texture; e.g.,

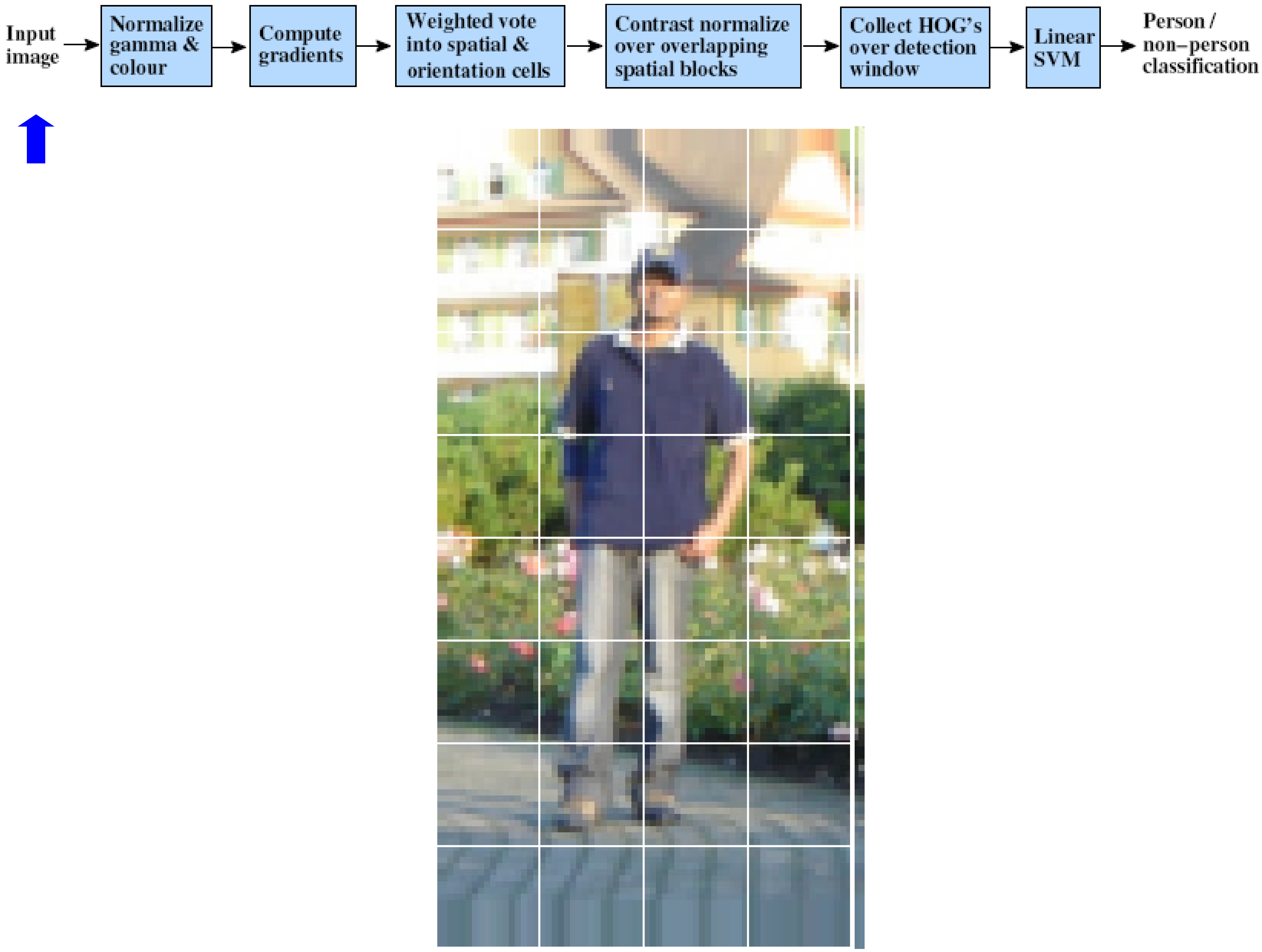


SVM with HoG [Dalal & Triggs, CVPR 2005]

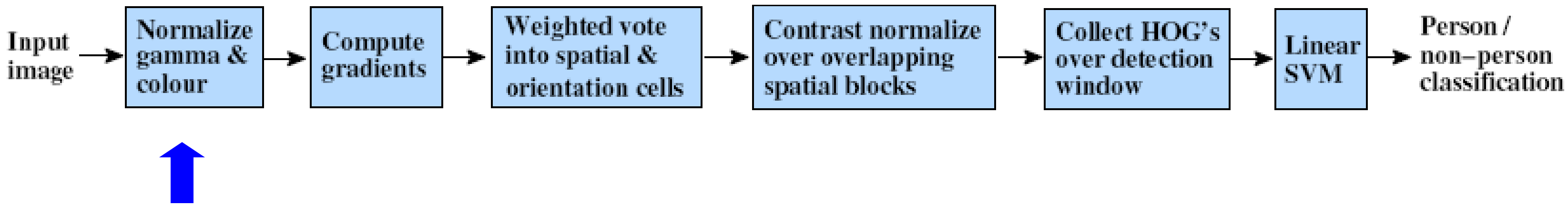
Kristen Grauman

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



Slides by Pete Barnum



- **Tested with**
  - **RGB**
  - **LAB**
  - **Grayscale**

Slightly better performance vs. grayscale

- **Gamma Normalization and Compression**
  - **Square root**
  - **Log**

Very slightly better performance vs. no adjustment

Input image → Normalize gamma & colour → Compute gradients → Weighted vote into spatial & orientation cells → Contrast normalize over overlapping spatial blocks → Collect HOG's over detection window → Linear SVM → Person / non–person classification

Outperforms

-1	0	1

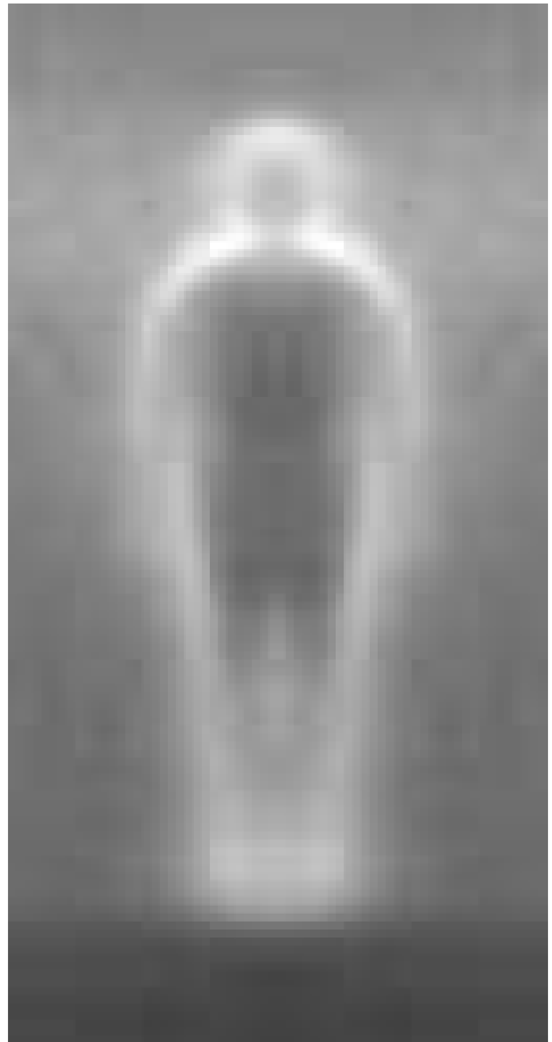
centered

-1	1

uncentered

1	-8	0	8	-1

cubic-corrected

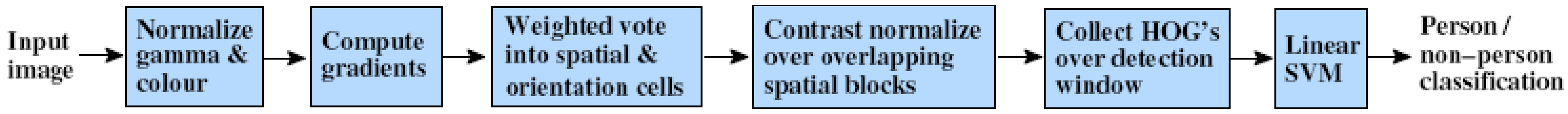


0	1
-1	0

diagonal

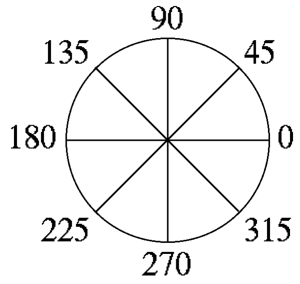
-1	0	1
-2	0	2
-1	0	1

Sobel

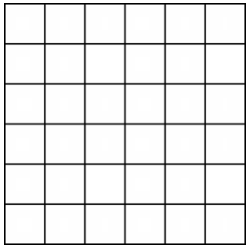


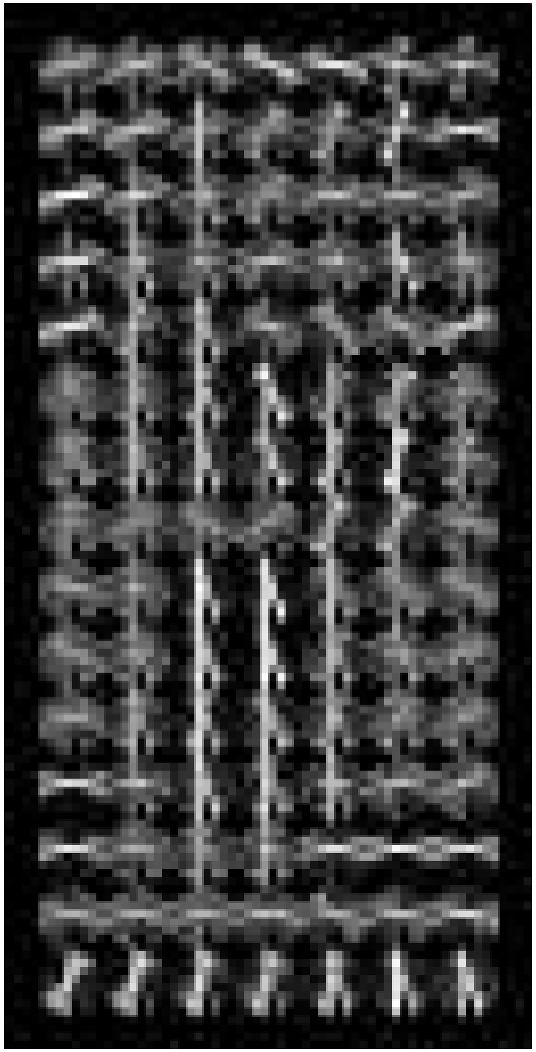
- **Histogram of gradient orientations**

Orientation: 9 bins (for unsigned angles)

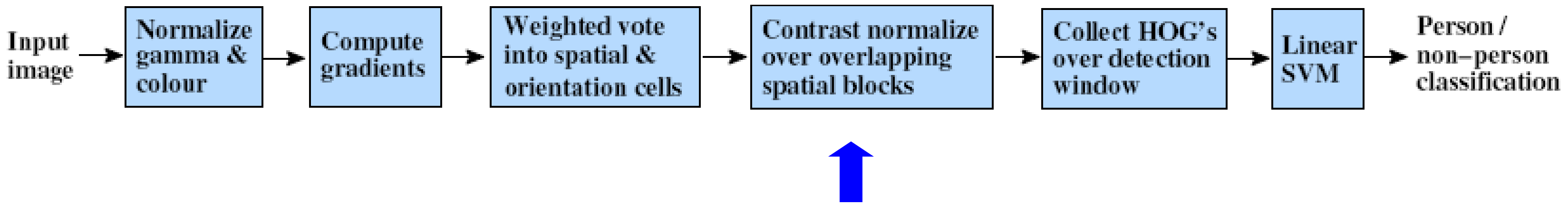


Histograms in k x k pixel cells

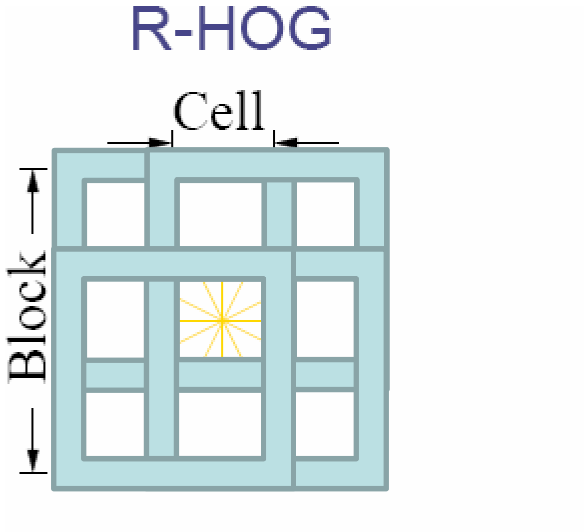




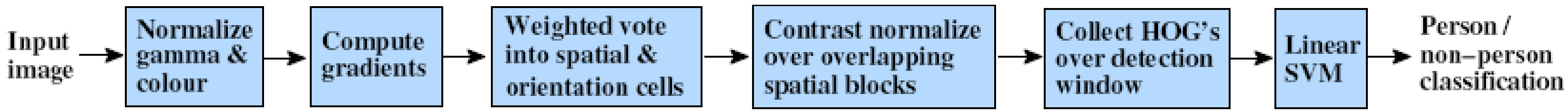
- **Votes weighted by magnitude**
- **Bilinear interpolation between cells**



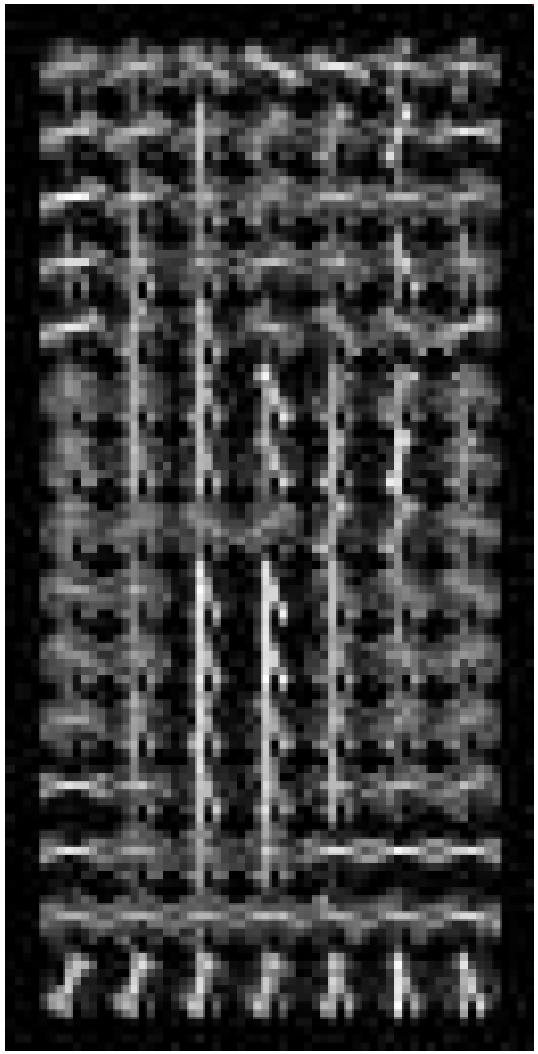
Normalize with respect to surrounding cells



$$L2-norm : v \longrightarrow v/\sqrt{||v||_2^2+\epsilon^2}$$



Original Formulation

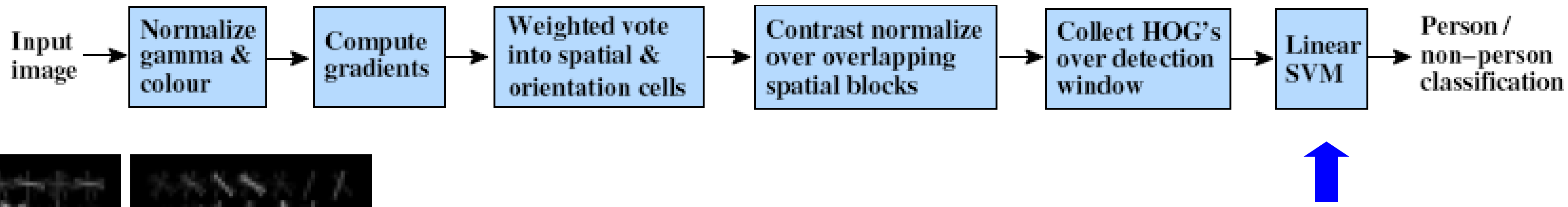
X= 

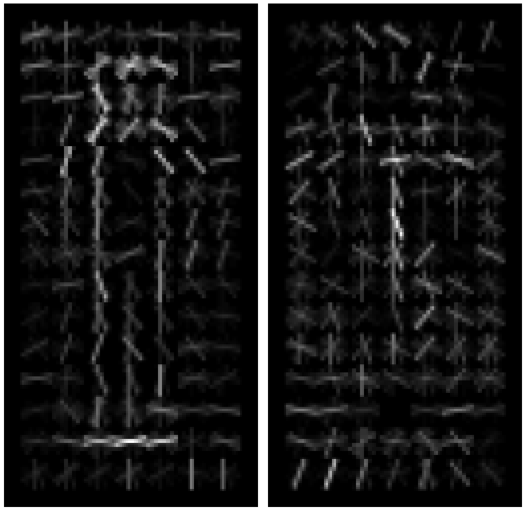
\# orientations

\# features = 15 x 7 x 9 x 4 = 3780

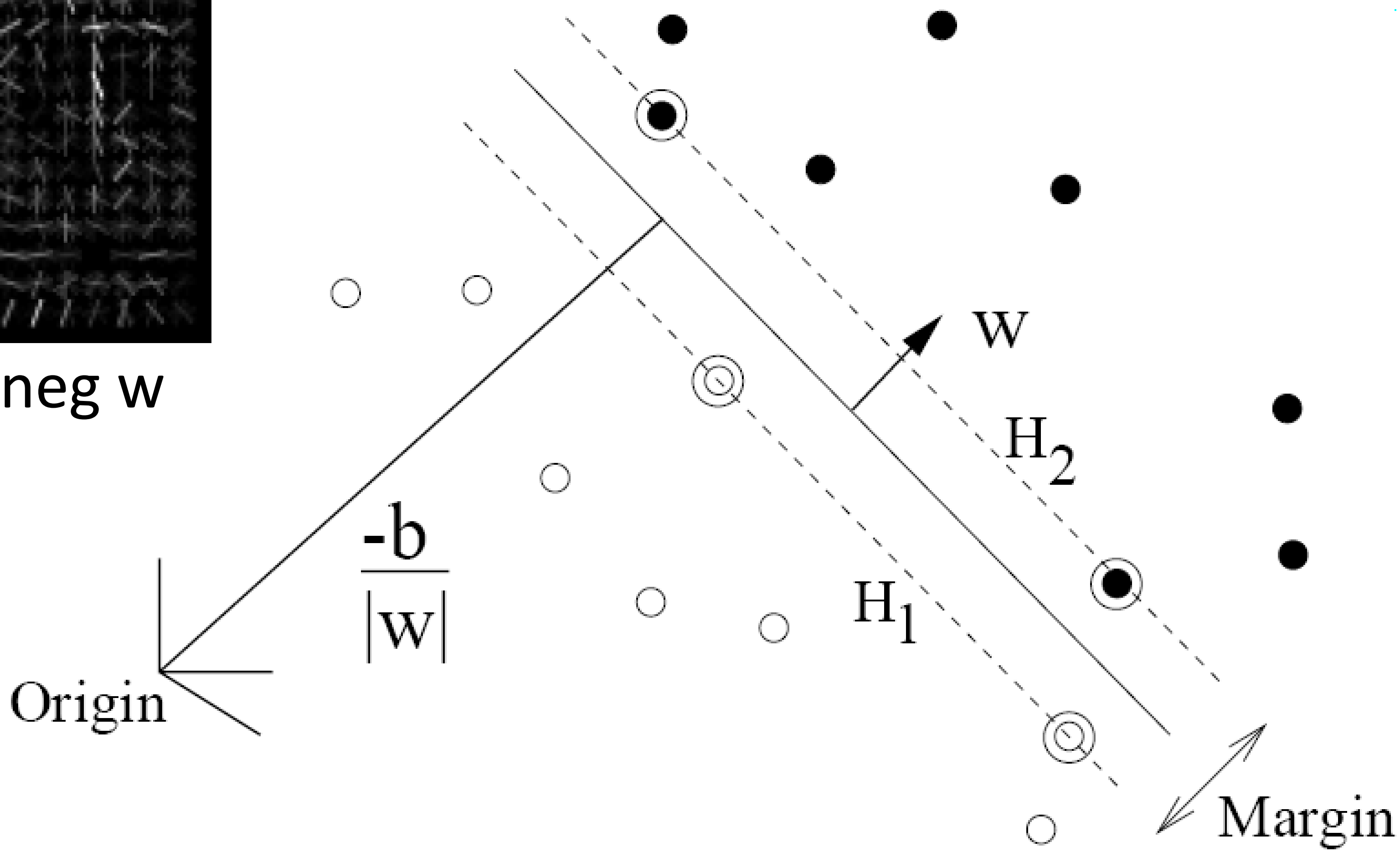
\# cells

\# normalizations by neighboring cells

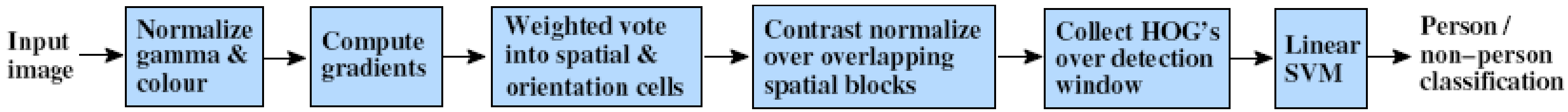


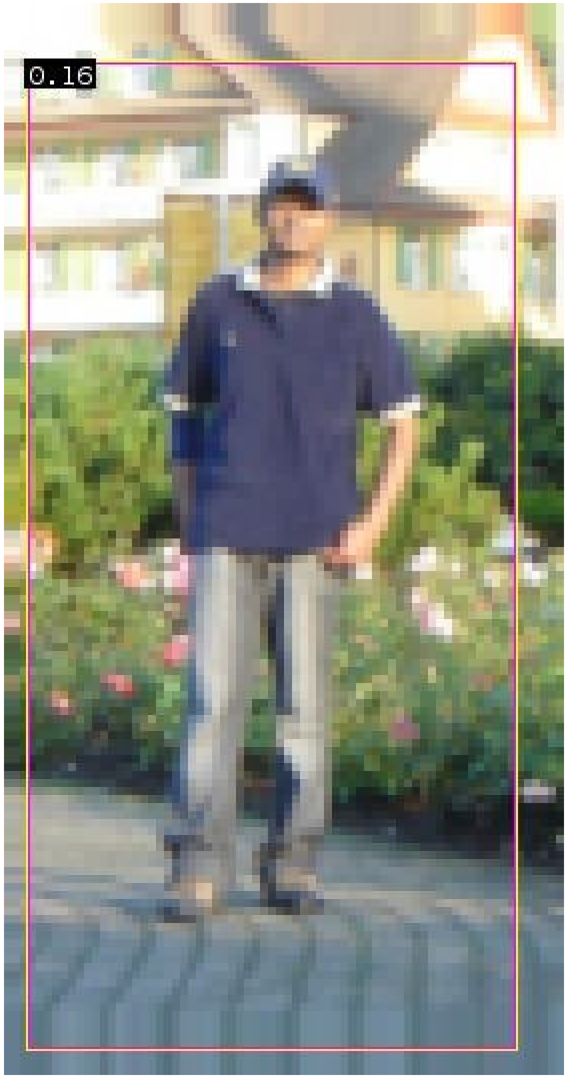


pos w neg w



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





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

$$sign(0.16) = 1$$

$=>$ pedestrian

Detection examples

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 and Weaknesses of Statistical Template Approach

Strengths

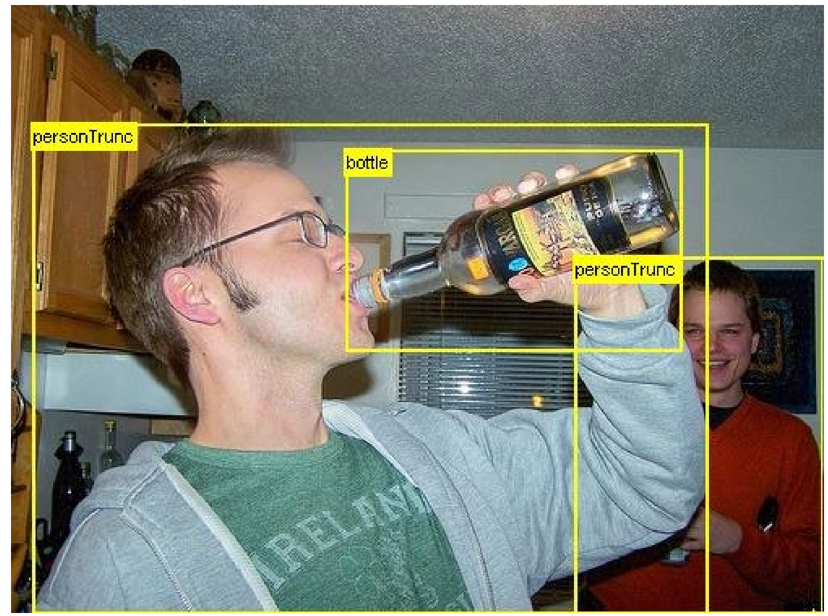
- Works very well for non-deformable objects with canonical orientations: faces, cars, pedestrians
- Fast detection

Weaknesses

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

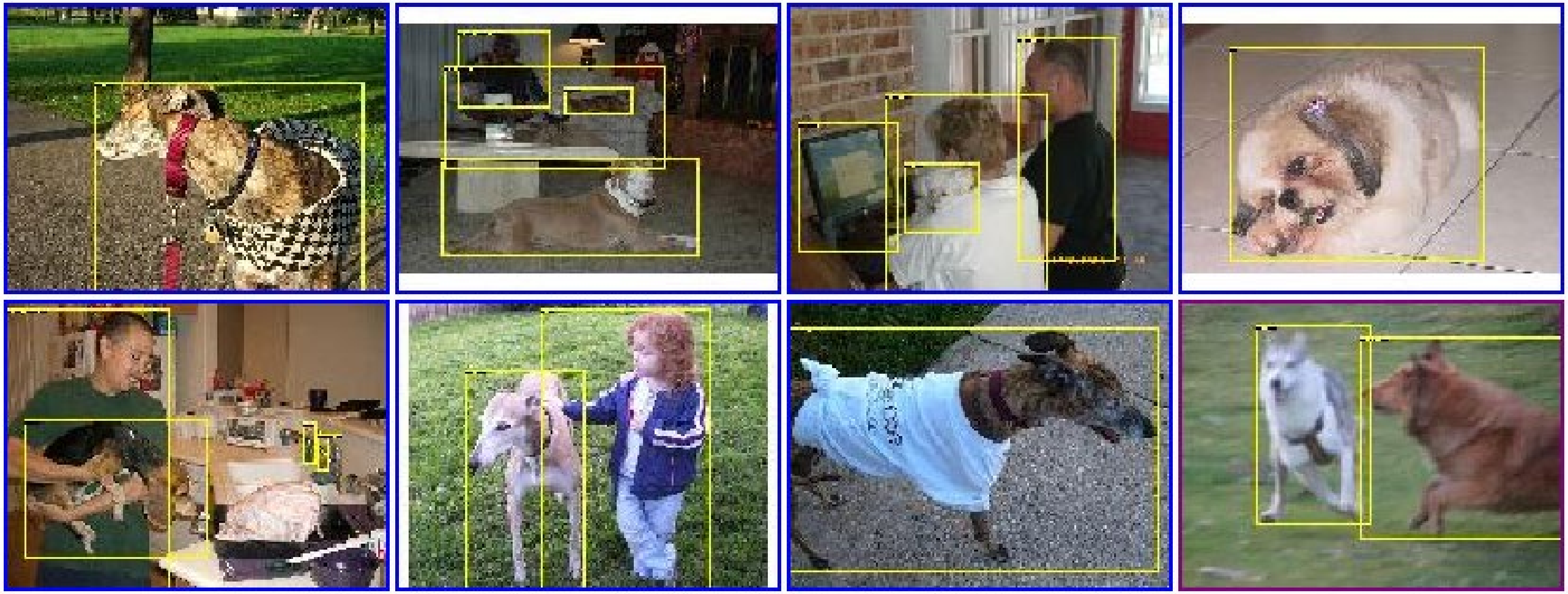
Limitations (continued)

- Not all objects are “box” shaped



Limitations (continued)

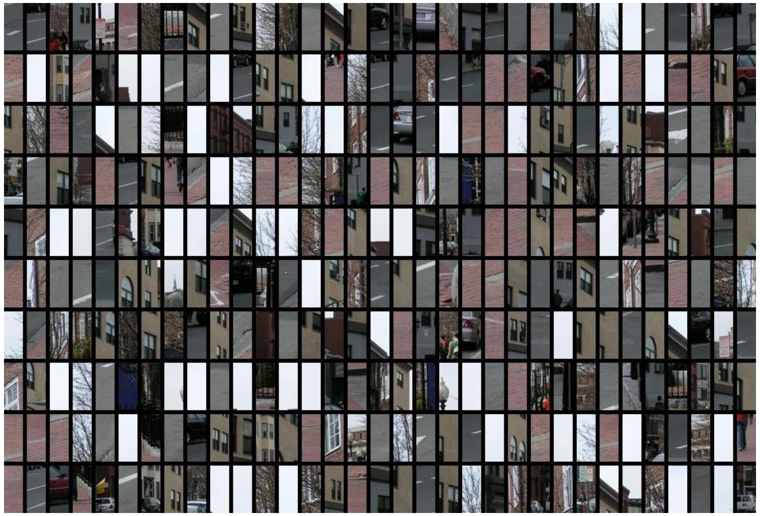
- Non-rigid, deformable objects not captured well with representations assuming a fixed 2d structure; or must assume fixed viewpoint
- Objects with less-regular textures not captured well with holistic appearance-based descriptions



Limitations (continued)

- If considering windows in isolation, context is lost

Sliding window



Detector's view

Limitations (continued)

- In practice, often entails large, cropped training set (expensive)
- Requiring good match to a global appearance description can lead to sensitivity to partial occlusions

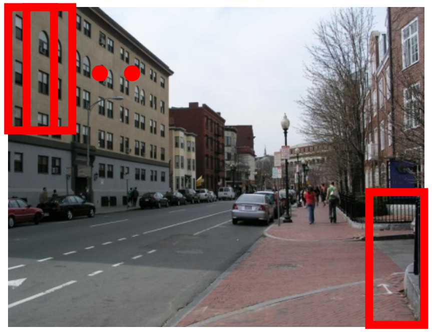
Tricks of the trade

- 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 detectable object
- “Jittering” 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

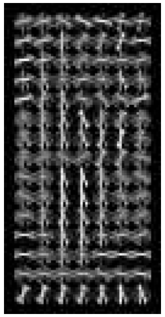
Influential Works in Detection

- Sung-Poggio (1994, 1998) : ~2000 citations
  - Basic idea of statistical template detection (I think), bootstrapping to get "face-like" negative examples, multiple whole-face prototypes (in 1994)
- Rowley-Baluja-Kanade (1996-1998) : ~3600
  - "Parts" at fixed position, non-maxima suppression, simple cascade, rotation, pretty good accuracy, fast
- Schneiderman-Kanade (1998-2000,2004) : ~1700
  - Careful feature engineering, excellent results, cascade
- Viola-Jones (2001, 2004) : ~11,000
  - Haar-like features, Adaboost as feature selection, hyper-cascade, very fast, easy to implement
- Dalal-Triggs (2005) : ~6500
  - Careful feature engineering, excellent results, HOG feature, online code
- Felzenszwalb-Huttenlocher (2000): ~2100
  - Efficient way to solve part-based detectors
- Felzenszwalb-McAllester-Ramanan (2008): ~1300
  - Excellent template/parts-based blend

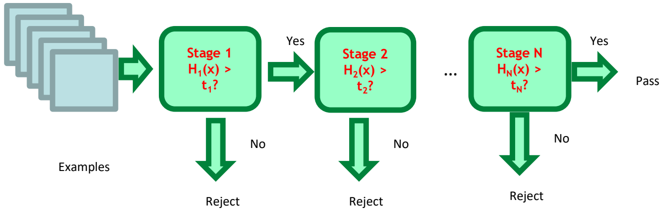
Things to remember



- Sliding window for search
- Features based on differences of intensity (gradient, wavelet, etc.)
  - Excellent results require careful feature design

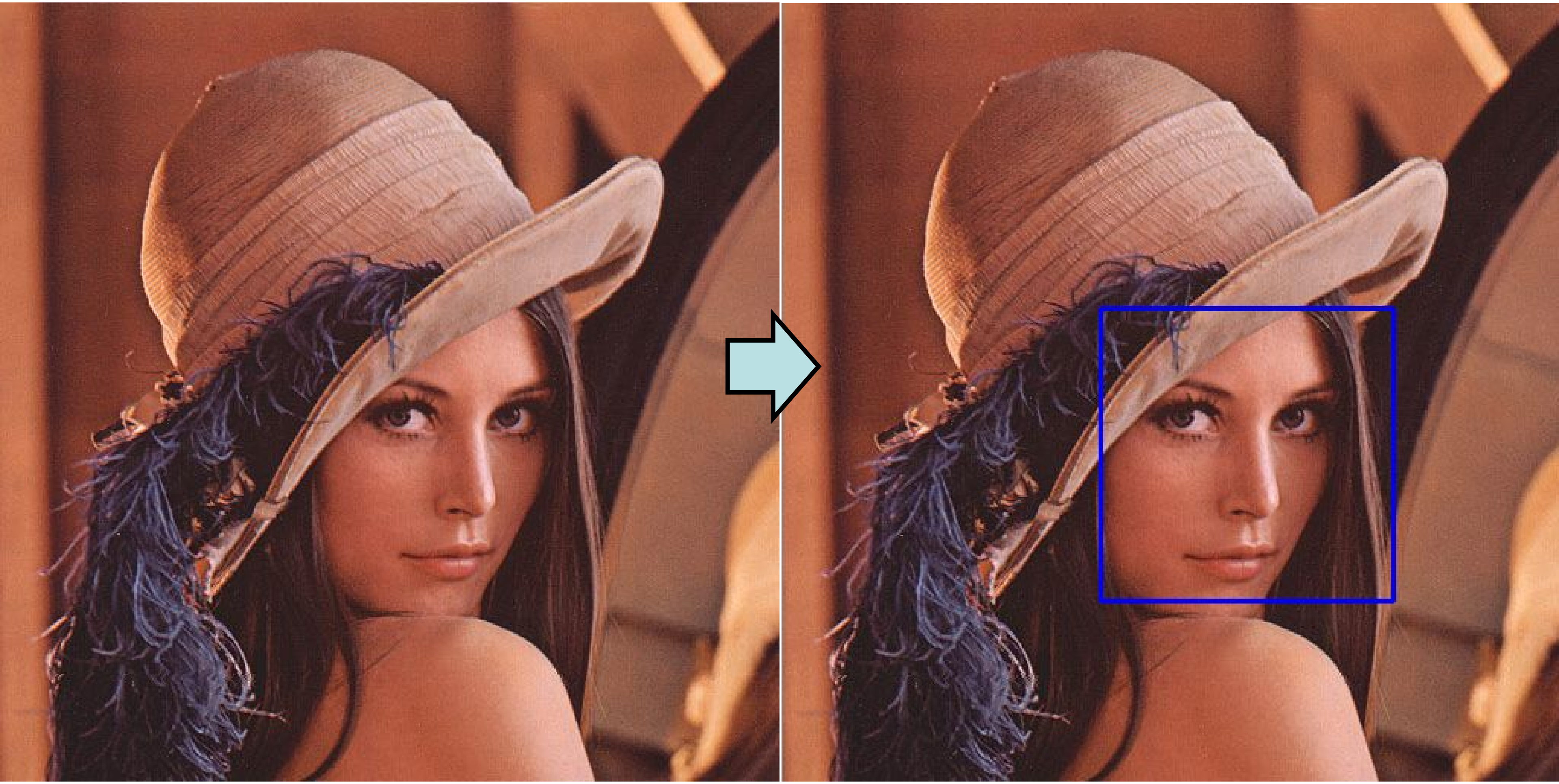


- Boosting for feature selection
- Integral images, cascade for speed



- Bootstrapping to deal with many, many negative examples

Lab. 1 Face Detection



Code

https://github.com/opencv/opencv/tree/master/data/haarcascades

Download here

```
import cv2

# Load the cascade
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
# Read the input image
img = cv2.imread('Lena.jpg')
# Convert into grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Detect faces
faces = face_cascade.detectMultiScale(gray, 1.1, 4)
# Draw rectangle around the faces
for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x+w, y+h), (255, 0, 0), 2)
# Display the output
cv2.imshow('img', img)
cv2.waitKey()
```