

# Computer Vision - Lecture 9

## Sliding-Window based Object Detection II

23.11.2016

Bastian Leibe

RWTH Aachen

<http://www.vision.rwth-aachen.de>

[leibe@vision.rwth-aachen.de](mailto:leibe@vision.rwth-aachen.de)

# Course Outline

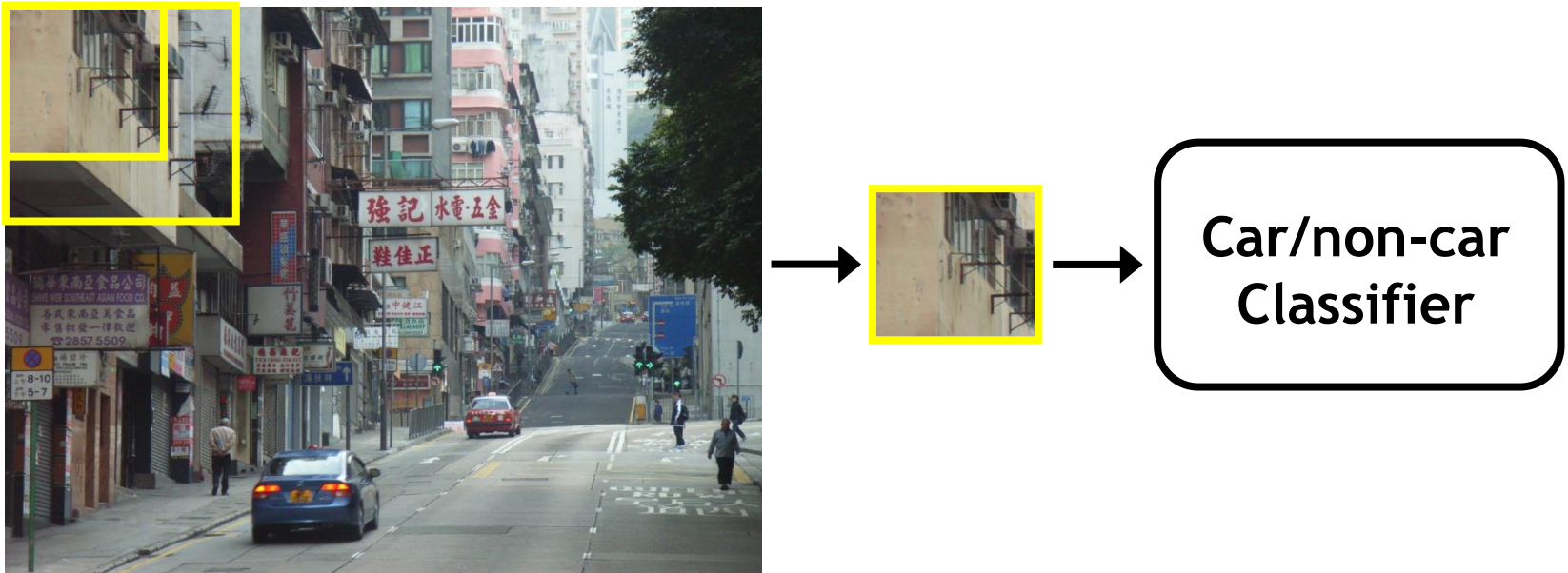
- **Image Processing Basics**
- **Segmentation**
  - Segmentation and Grouping
  - Segmentation as Energy Minimization
- **Recognition & Categorization**
  - **Sliding-Window Object Detection**
  - Image Classification
- **Local Features & Matching**
- **3D Reconstruction**
- **Motion and Tracking**

# Topics of This Lecture

- **Recap: Classification with SVMs**
  - Support Vector Machines
  - HOG Detector
- **Classification with Boosting**
  - AdaBoost
  - Viola-Jones Face Detection
- **Discussion**

# Recap: Sliding-Window Object Detection

- If the object may be in a cluttered scene, slide a window around looking for it.



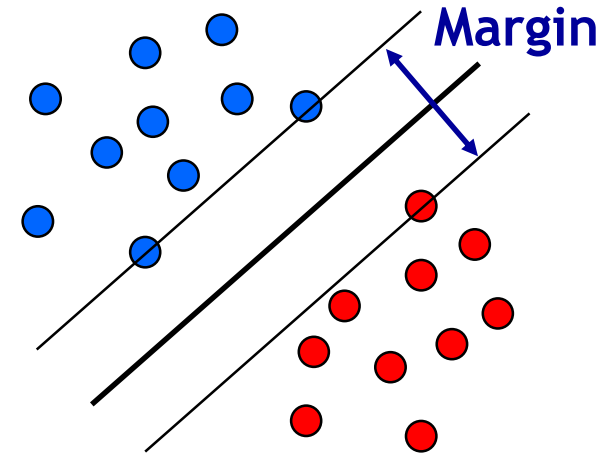
- Essentially, this is a brute-force approach with many local decisions.

# Recap: Support Vector Machine (SVM)

- **Basic idea**

- The SVM tries to find a classifier which maximizes the **margin** between pos. and neg. data points.
- Up to now: consider linear classifiers

$$\mathbf{w}^T \mathbf{x} + b = 0$$



- Formulation as a **convex optimization problem**

- Find the hyperplane satisfying

$$\arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$$

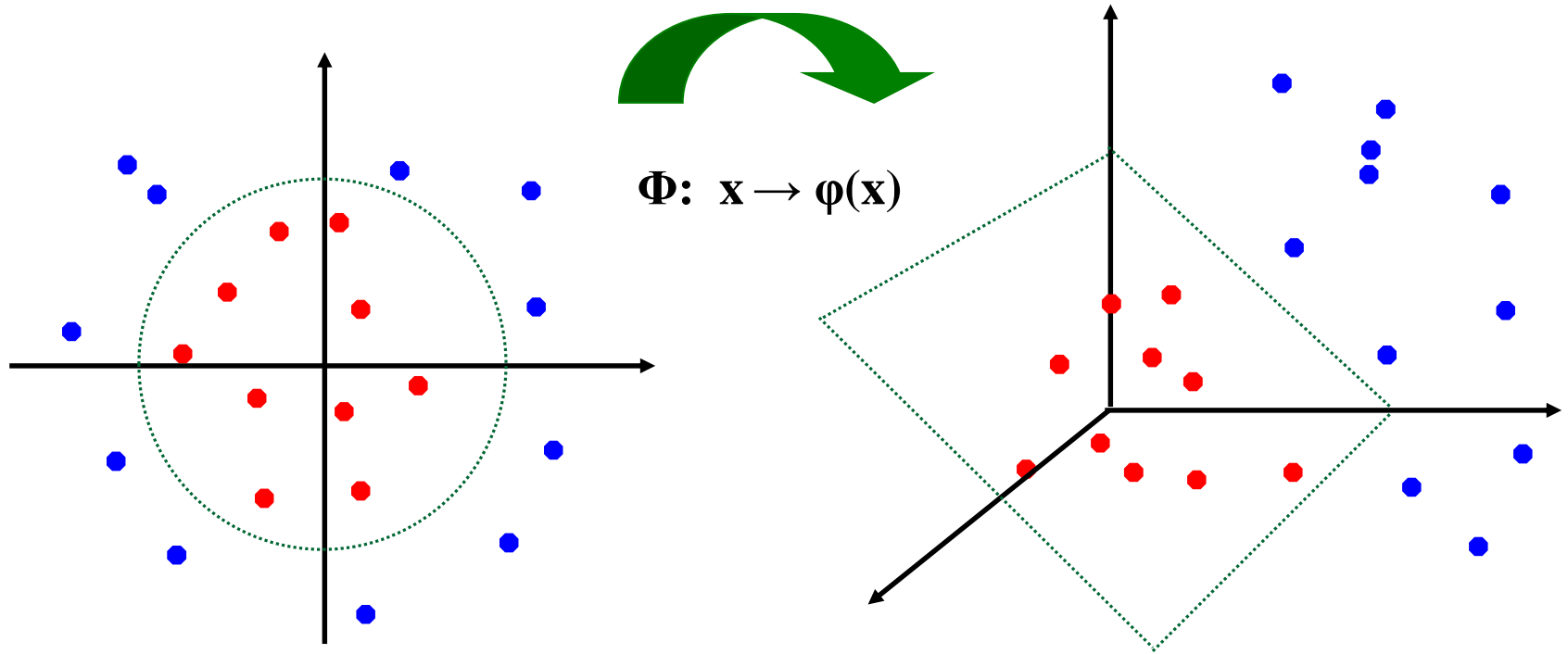
under the constraints

$$t_n (\mathbf{w}^T \mathbf{x}_n + b) \geq 1 \quad \forall n$$

based on training data points  $\mathbf{x}_n$  and target values  $t_n \in \{-1, 1\}$ .

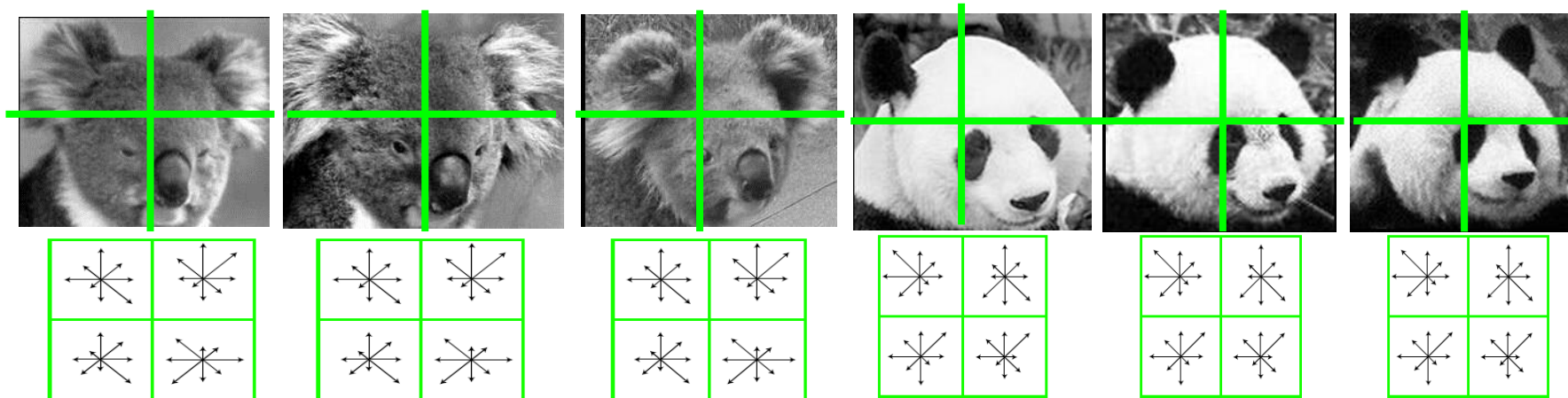
# Recap: Non-Linear SVMs

- General idea: The original input space can be mapped to some higher-dimensional feature space where the training set is separable:



# Recap: Gradient-based Representations

- Consider edges, contours, and (oriented) intensity gradients



- Summarize local distribution of gradients with histogram
  - Locally orderless: offers invariance to small shifts and rotations
  - Contrast-normalization: try to correct for variable illumination

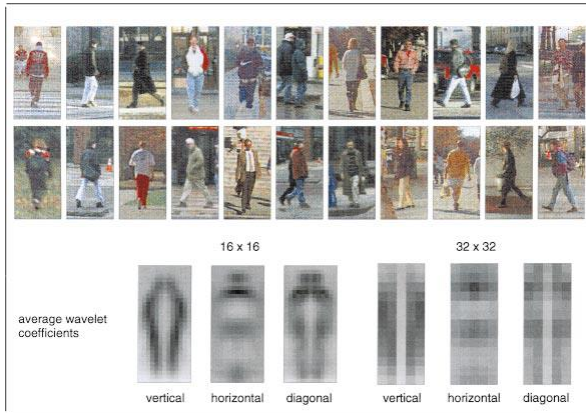
# Topics of This Lecture

- **Recap: Classification with SVMs**
  - Support Vector Machines
  - HOG Detector
- **Classification with Boosting**
  - AdaBoost
  - Viola-Jones Face Detection
- **Discussion**



# Pedestrian Detection

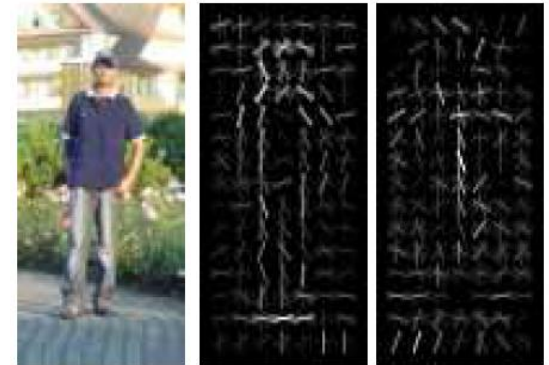
- Detecting upright, walking humans using sliding window's appearance/texture; e.g.,



**SVM with Haar wavelets**  
[Papageorgiou & Poggio, IJCV 2000]



**Space-time rectangle features** [Viola, Jones & Snow, ICCV 2003]



**SVM with HoGs** [Dalal & Triggs, CVPR 2005]

# HOG Descriptor Processing Chain

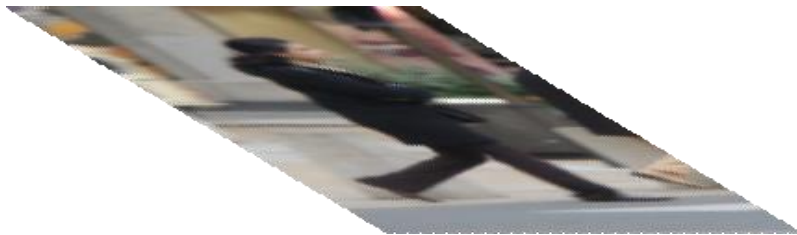


Image Window

# HOG Descriptor Processing Chain

- **Optional: Gamma compression**
  - Goal: Reduce effect of overly strong gradients
  - Replace each pixel color/intensity by its square-root

$$x \mapsto \sqrt{x}$$

⇒ Small performance improvement



Gamma compression

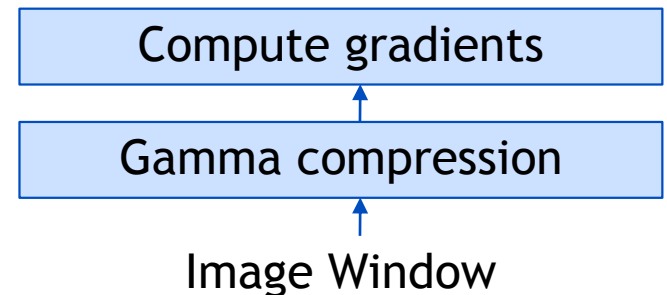
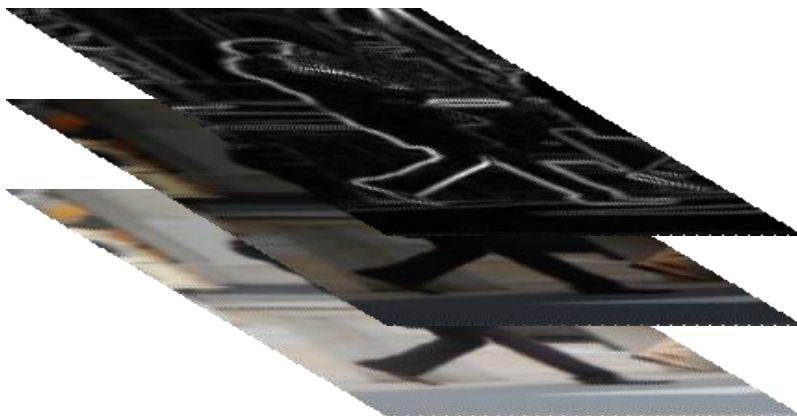
Image Window

# HOG Descriptor Processing Chain

- Gradient computation

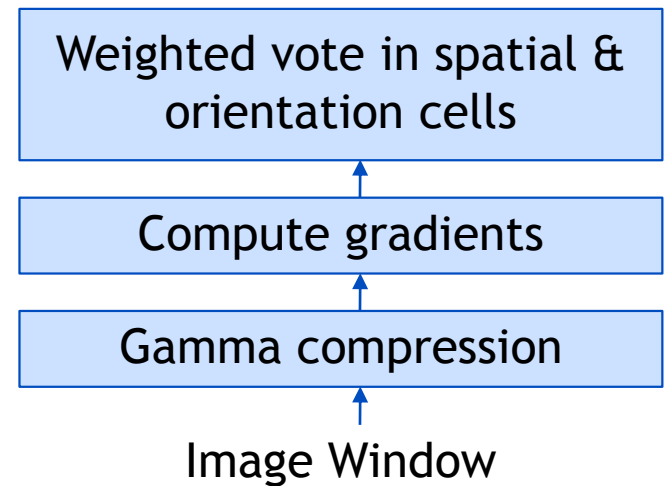
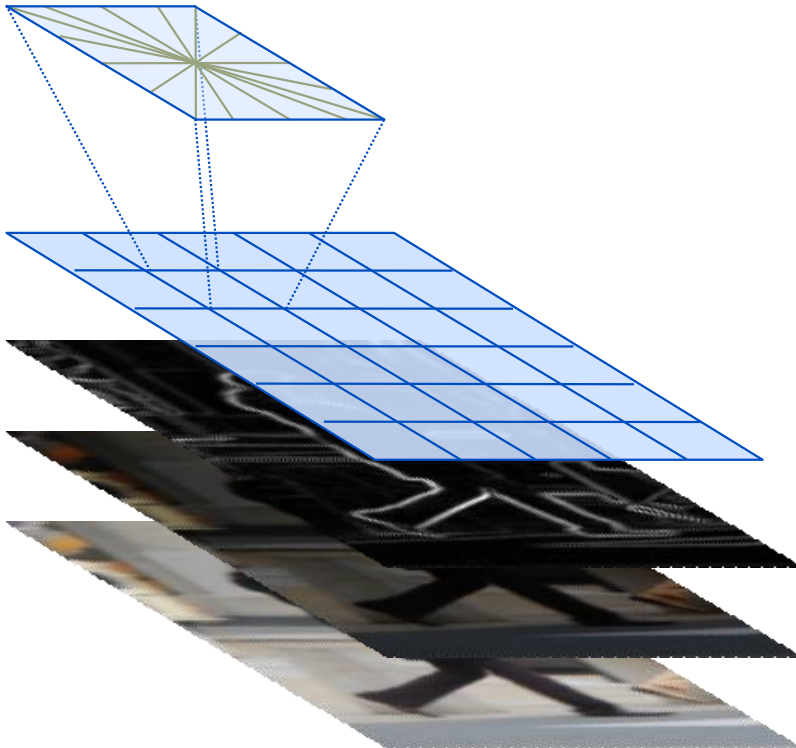
- Compute gradients on all color channels and take strongest one
- Simple finite difference filters work best (no Gaussian smoothing)

$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$



# HOG Descriptor Processing Chain

- **Spatial/Orientation binning**
  - Compute localized histograms of oriented gradients
  - Typical subdivision:  
 $8 \times 8$  cells with 8 or 9 orientation bins



# HOG Cell Computation Details

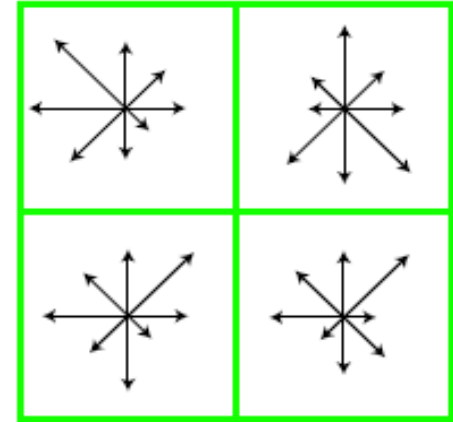
- Gradient orientation voting

- Each pixel contributes to localized gradient orientation histogram(s)
- Vote is weighted by the pixel's gradient magnitude



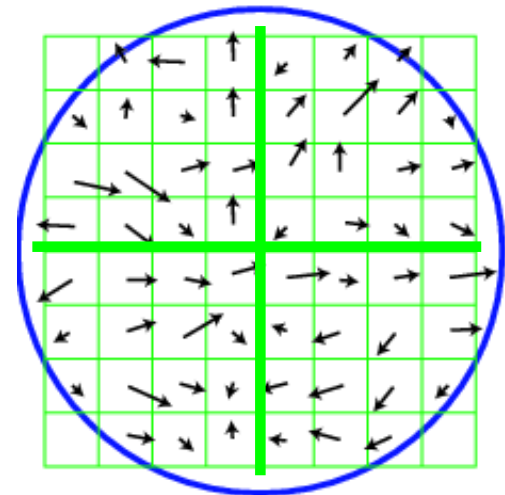
$$\theta = \tan^{-1} \left( \frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right)$$

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$



- Block-level Gaussian weighting

- An additional Gaussian weight is applied to each  $2 \times 2$  block of cells
- Each cell is part of 4 such blocks, resulting in 4 versions of the histogram.



# HOG Cell Computation Details (2)

- Important for robustness: **Tri-linear interpolation**

- Each pixel contributes to (up to) 4 neighboring cell histograms
- Weights are obtained by **bilinear interpolation in image space**:

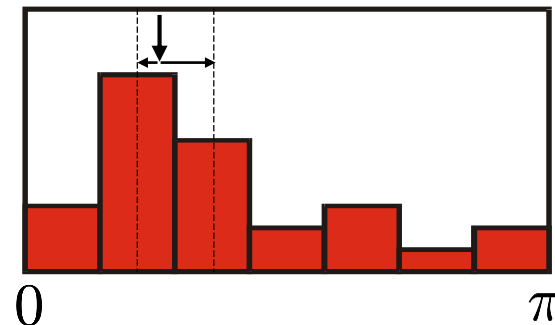
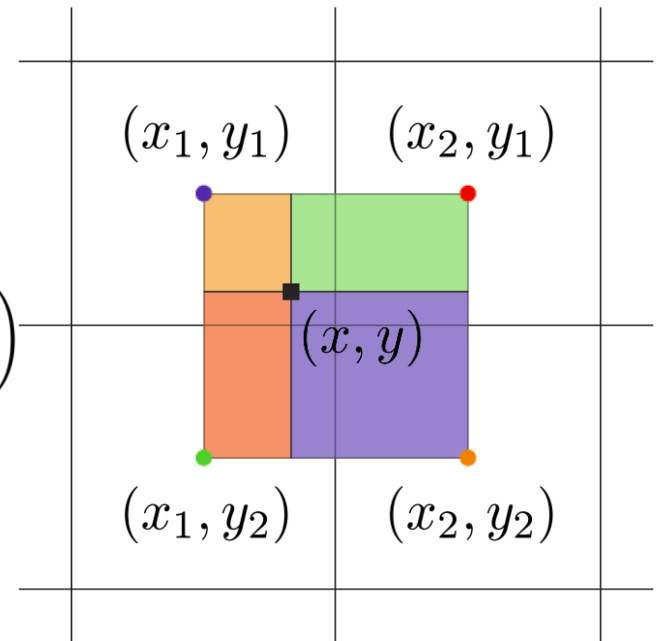
$$h(x_1, y_1) \leftarrow w \cdot \left(1 - \frac{x - x_1}{x_2 - x_1}\right) \left(1 - \frac{y - y_1}{y_2 - y_1}\right)$$

$$h(x_1, y_2) \leftarrow w \cdot \left(1 - \frac{x - x_1}{x_2 - x_1}\right) \left(\frac{y - y_1}{y_2 - y_1}\right)$$

$$h(x_2, y_1) \leftarrow w \cdot \left(\frac{x - x_1}{x_2 - x_1}\right) \left(1 - \frac{y - y_1}{y_2 - y_1}\right)$$

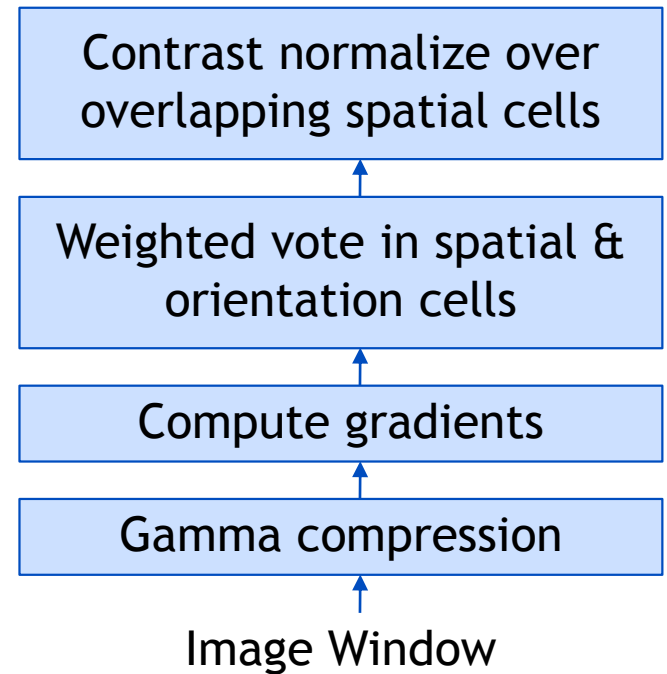
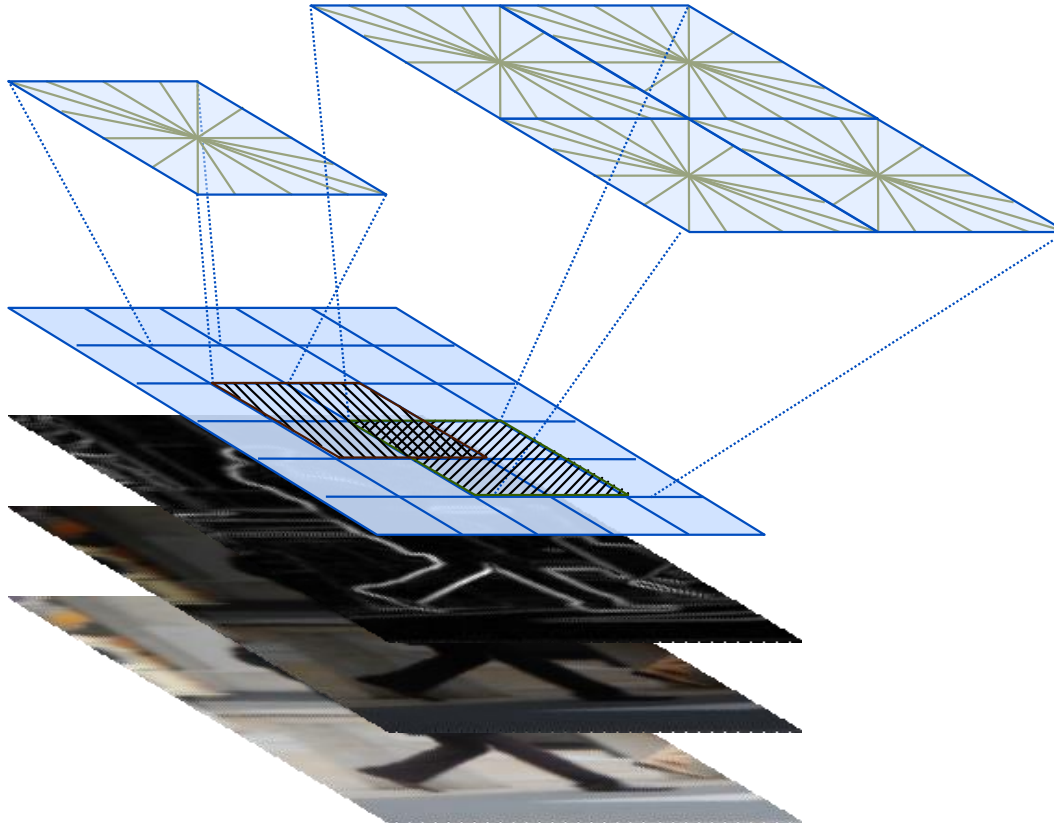
$$h(x_2, y_2) \leftarrow w \cdot \left(\frac{x - x_1}{x_2 - x_1}\right) \left(\frac{y - y_1}{y_2 - y_1}\right)$$

- Contribution is further split over (up to) 2 neighboring orientation bins via **linear interpolation over angles**.



# HOG Descriptor Processing Chain

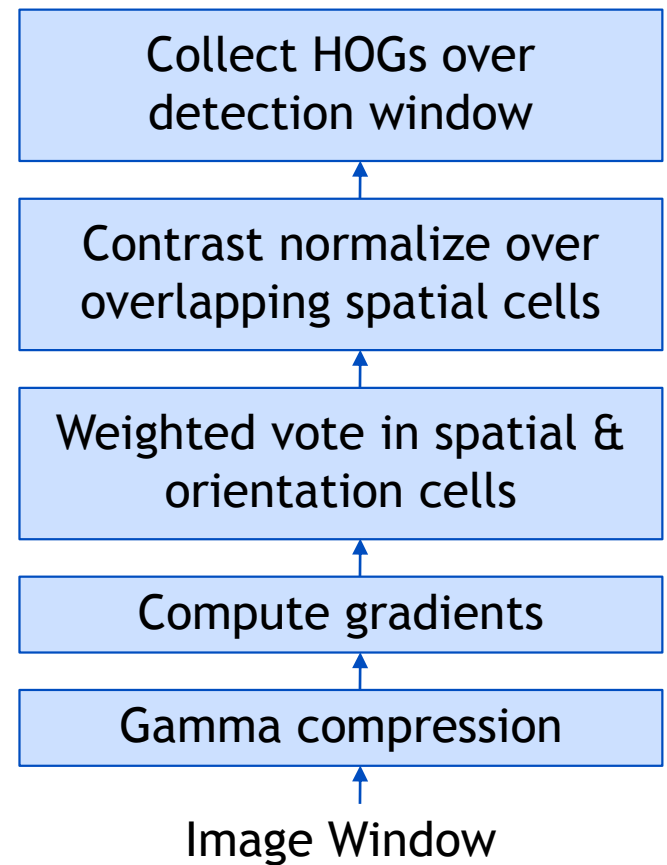
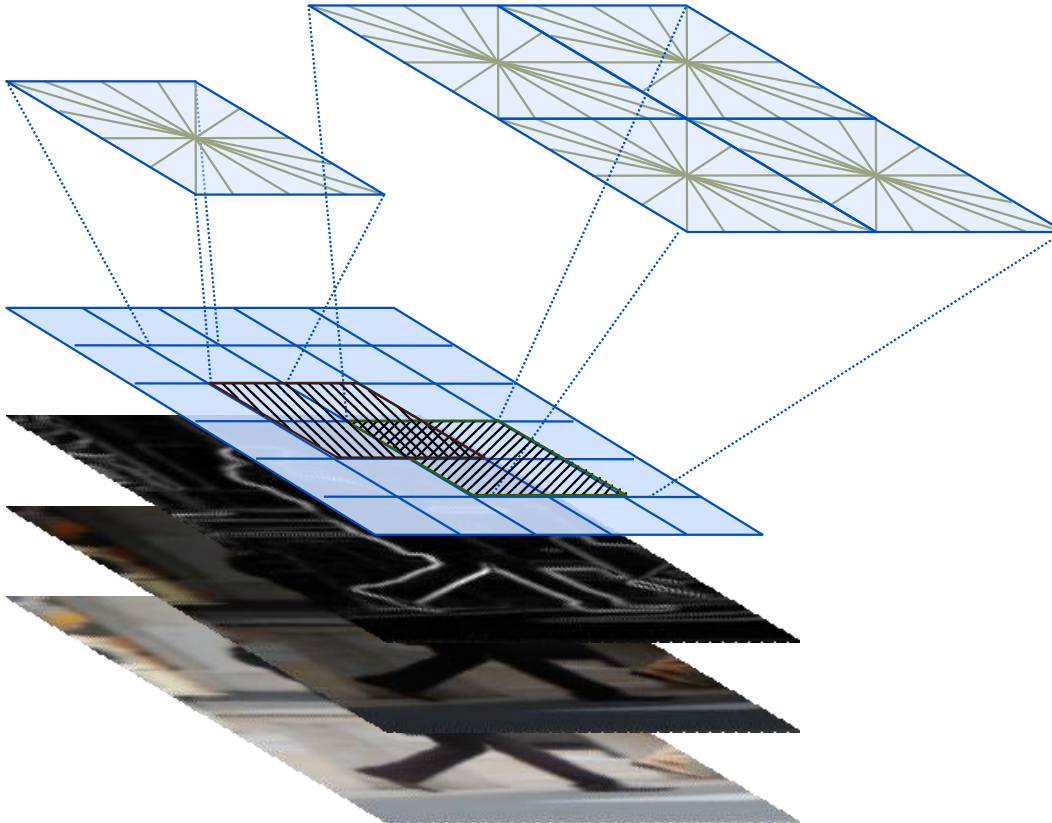
- 2-Stage contrast normalization
  - L2 normalization, clipping, L2 normalization





# HOG Descriptor Processing Chain

- Feature vector construction
  - Collect HOG blocks into vector  
[ ..., ..., ..., ... ]

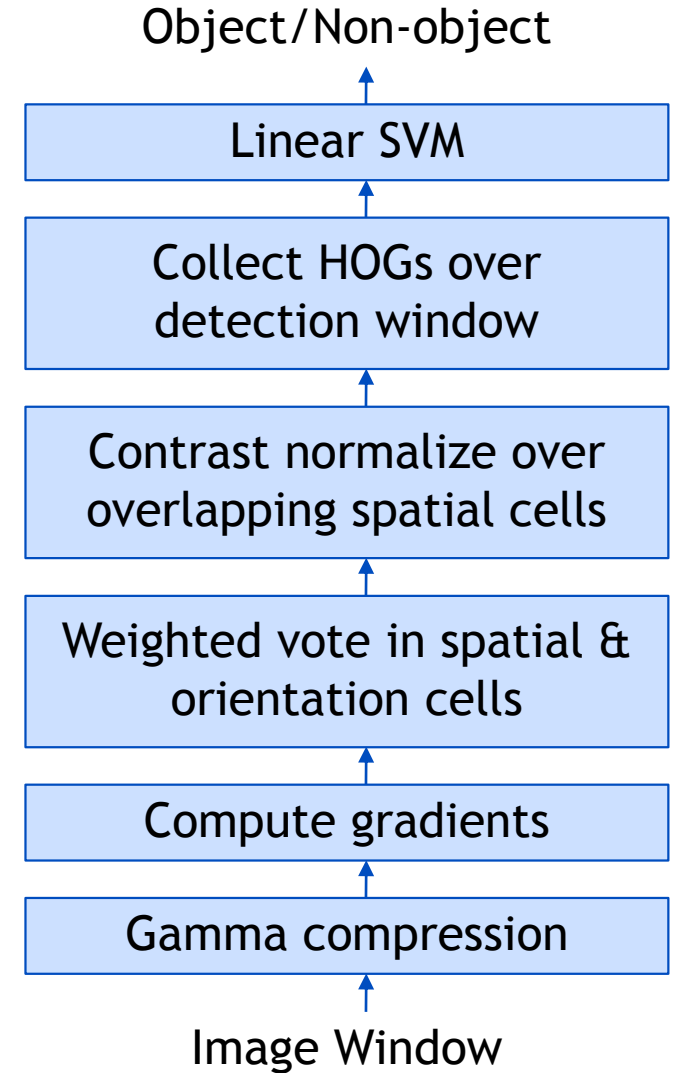
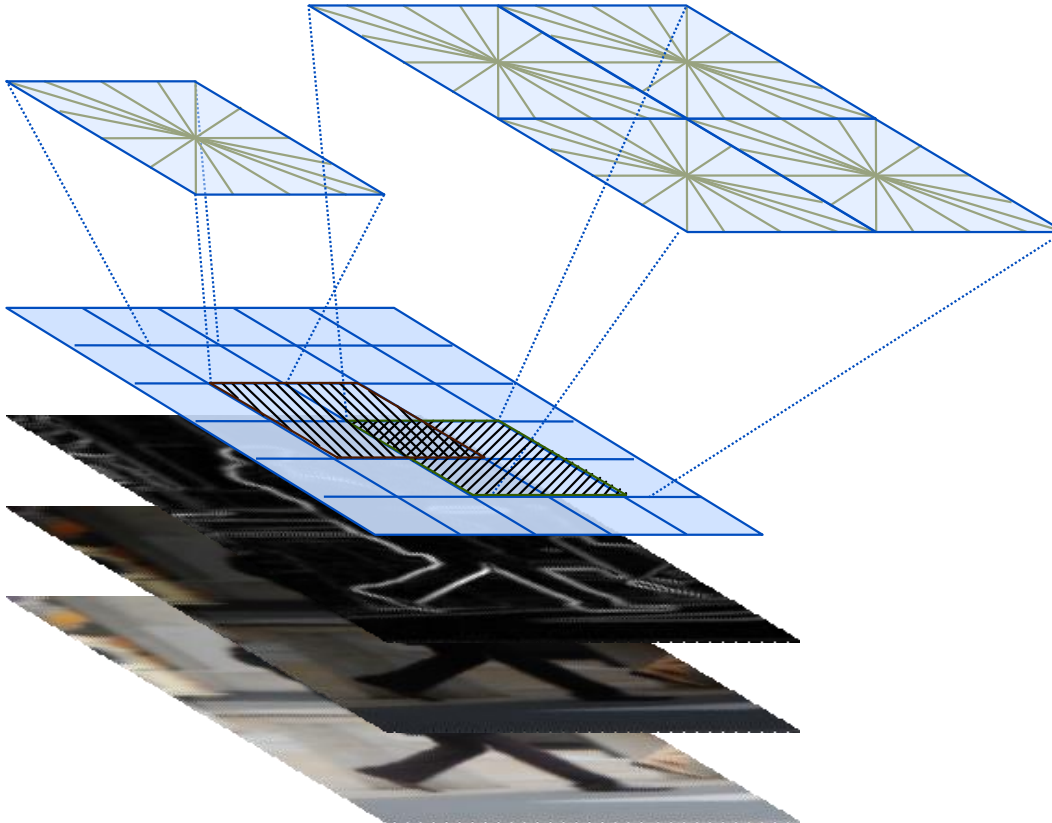


# HOG Descriptor Processing Chain

- SVM Classification

- Typically using a linear SVM

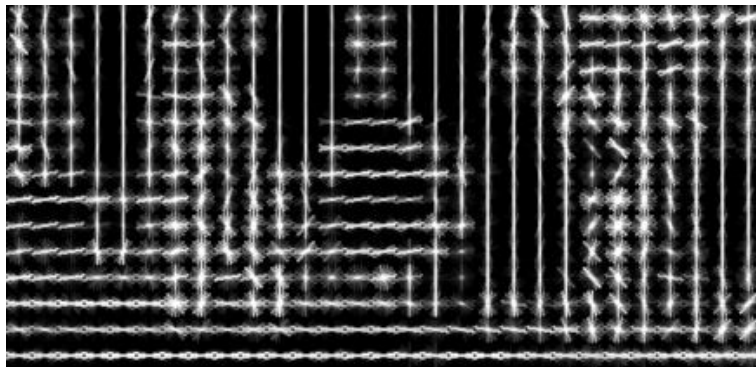
[ ..., ..., ..., ... ]



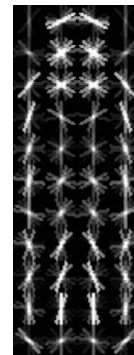
# Pedestrian Detection with HOG

- Train a pedestrian template using a linear SVM
- At test time, convolve feature map with template

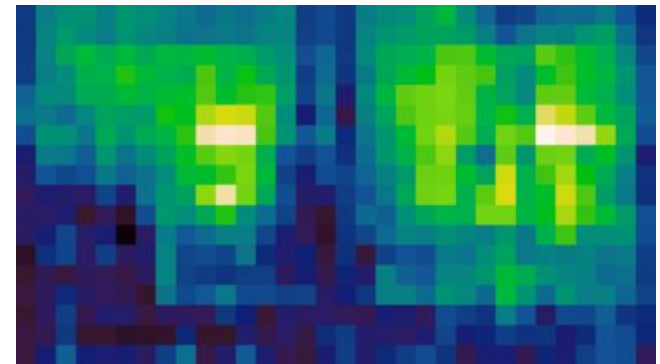
HOG feature map



Template

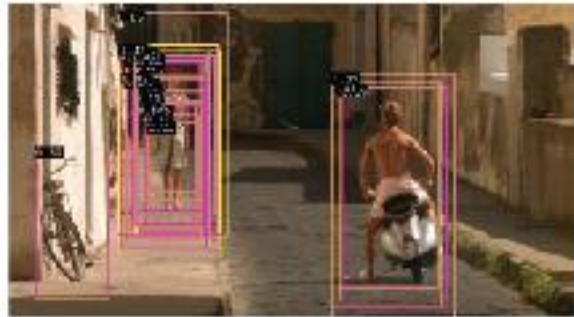


Detector response map



N. Dalal and B. Triggs, [Histograms of Oriented Gradients for Human Detection](#),  
CVPR 2005

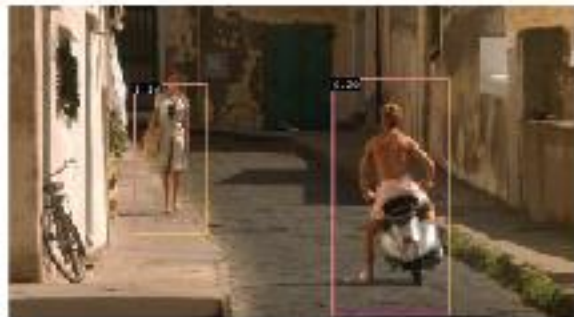
# Non-Maximum Suppression



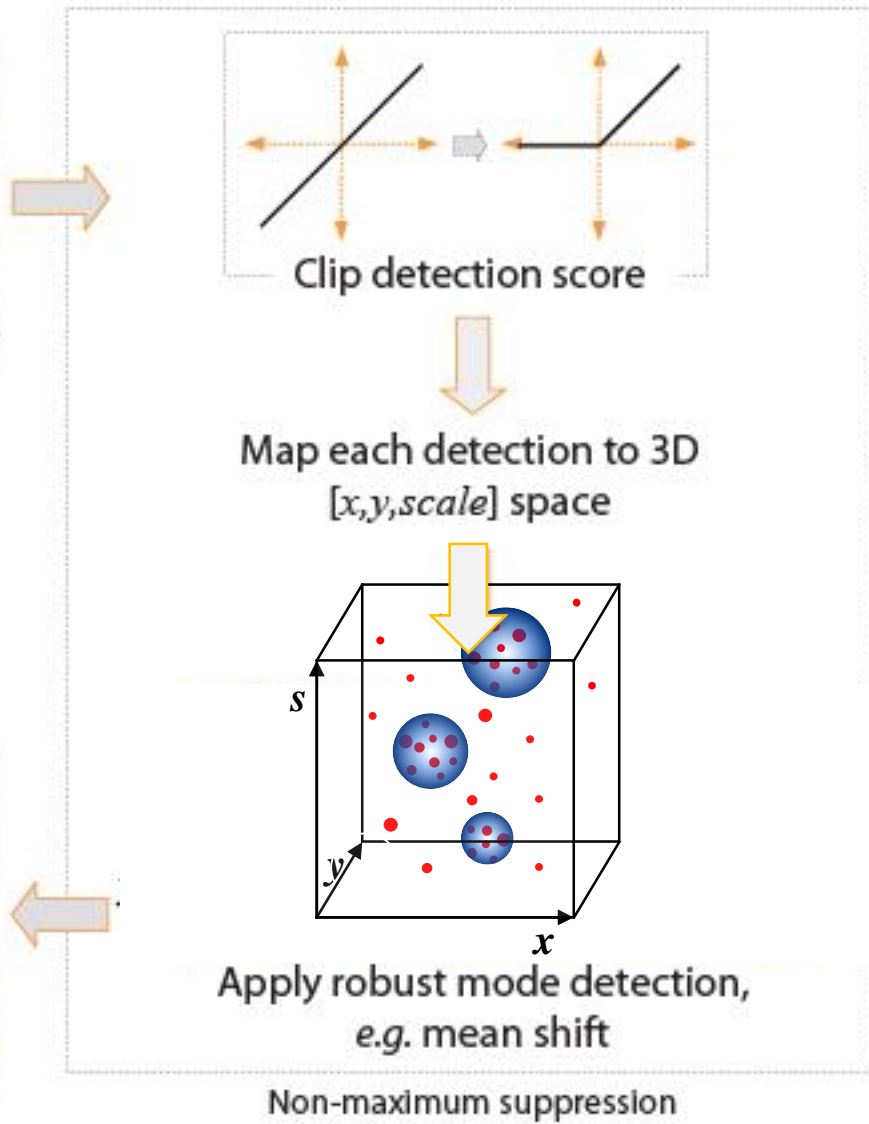
After multi-scale dense scan



Goal



Fusion of multiple detections

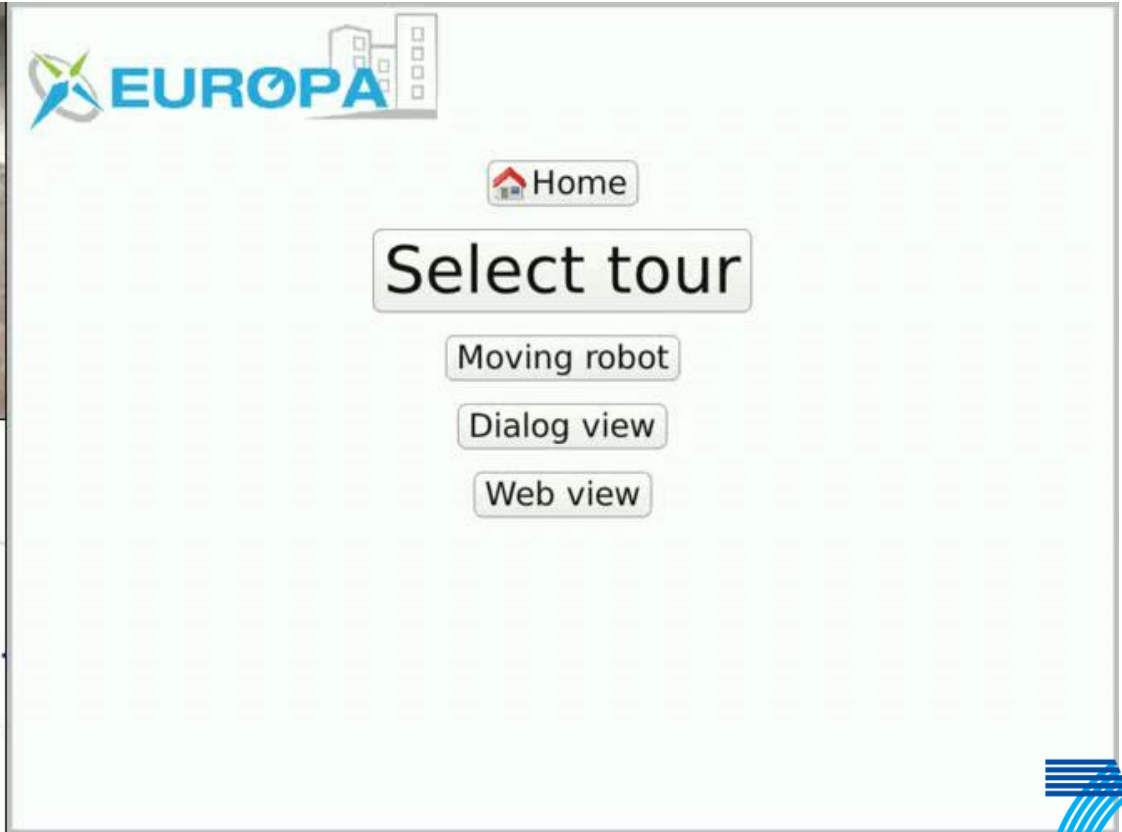


# Pedestrian detection with HoGs & SVMs



- [Navneet Dalal](#), [Bill Triggs](#), [Histograms of Oriented Gradients for Human Detection](#), CVPR 2005

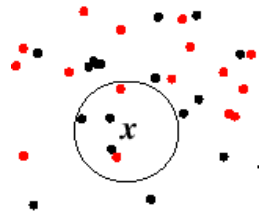
# Applications: Mobile Robot Navigation



[link to the video](#)

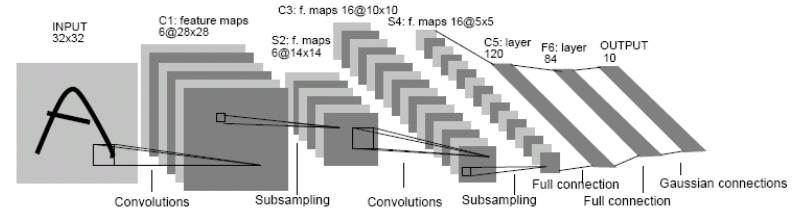
# Classifier Construction: Many Choices...

## Nearest Neighbor



Shakhnarovich, Viola, Darrell 2003  
Berg, Berg, Malik 2005,  
Boiman, Shechtman, Irani 2008, ...

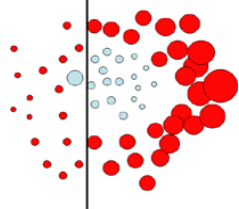
## Neural networks



LeCun, Bottou, Bengio, Haffner 1998  
Rowley, Baluja, Kanade 1998

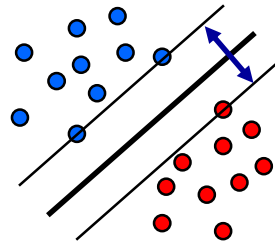
...

## Boosting



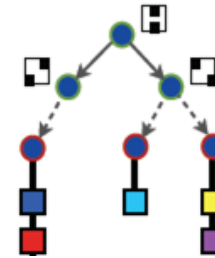
Viola, Jones 2001,  
Torralba et al. 2004,  
Opelt et al. 2006,  
Benenson 2012, ...

## Support Vector Machines



Vapnik, Schölkopf 1995,  
Papageorgiou, Poggio '01,  
Dalal, Triggs 2005,  
Vedaldi, Zisserman 2012

## Randomized Forests



Amit, Geman 1997,  
Breiman 2001,  
Lepetit, Fua 2006,  
Gall, Lempitsky 2009,...

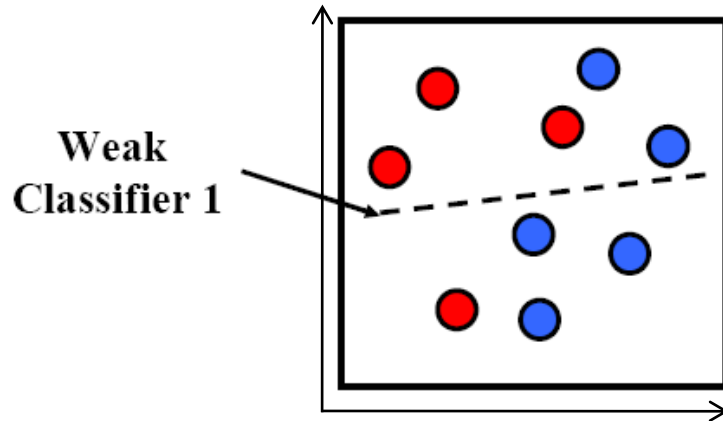
# Boosting

- Build a strong classifier  $H$  by combining a number of “weak classifiers”  $h_1, \dots, h_M$ , which need only be better than chance.
- Sequential learning process: at each iteration, add a weak classifier
- Flexible to choice of weak learner
  - including fast simple classifiers that alone may be inaccurate
- We’ll look at Freund & Schapire’s AdaBoost algorithm
  - Easy to implement
  - Base learning algorithm for Viola-Jones face detector

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



# AdaBoost: Intuition



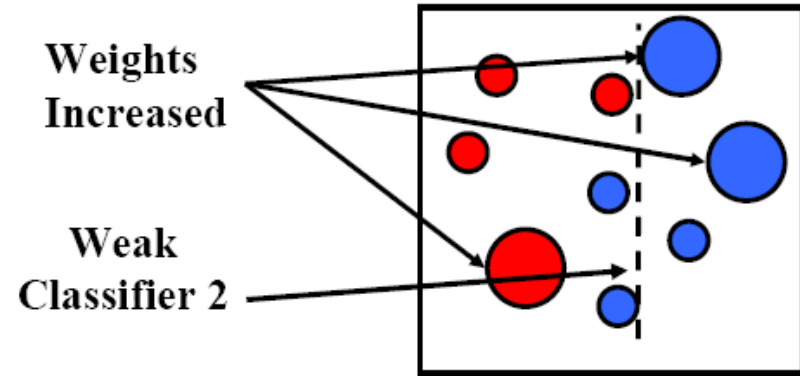
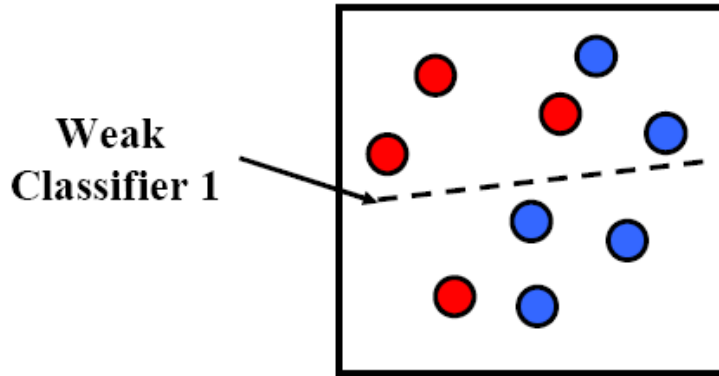
Consider a 2D feature space with **positive** and **negative** examples.

Each weak classifier splits the training examples with at least 50% accuracy.

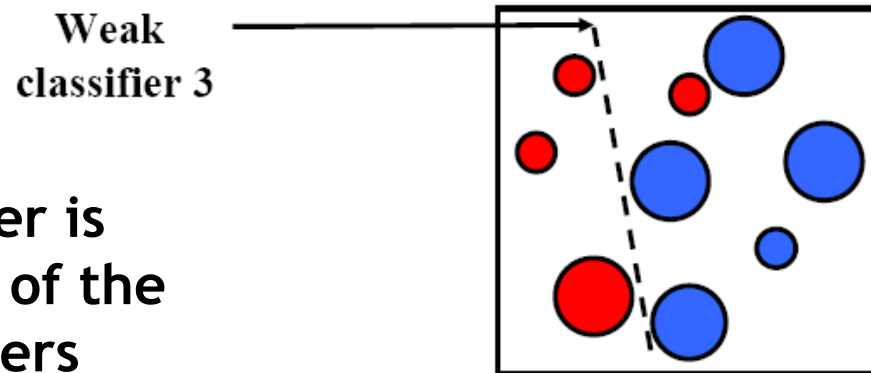
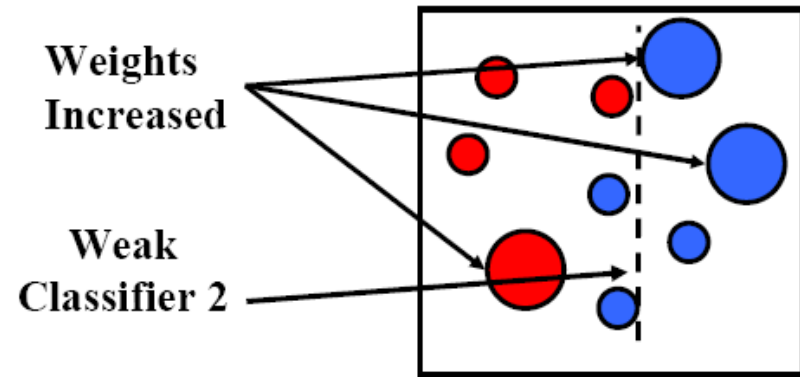
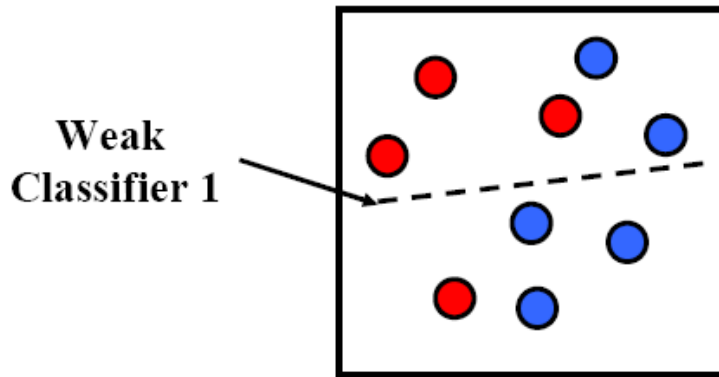
Examples misclassified by a previous weak learner are given more emphasis at future rounds.

Figure adapted from Freund and Schapire

# AdaBoost: Intuition



# AdaBoost: Intuition

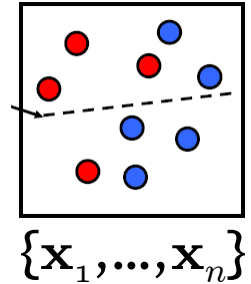


Final classifier is combination of the weak classifiers

# AdaBoost - Formalization

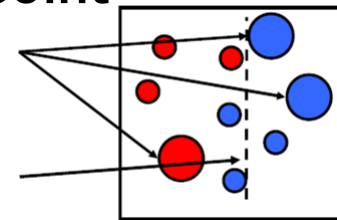
- 2-class classification problem

- Given: training set  $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$  with target values  $\mathbf{T} = \{t_1, \dots, t_N\}$ ,  $t_n \in \{-1, 1\}$ .
- Associated weights  $\mathbf{W} = \{w_1, \dots, w_N\}$  for each training point.



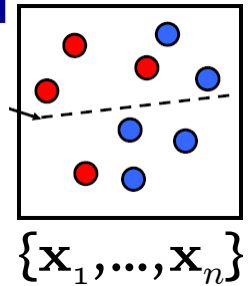
- Basic steps

- In each iteration, AdaBoost trains a new weak classifier  $h_m(\mathbf{x})$  based on the current weighting coefficients  $\mathbf{W}^{(m)}$ .
- We then adapt the weighting coefficients for each point
  - Increase  $w_n$  if  $\mathbf{x}_n$  was misclassified by  $h_m(\mathbf{x})$ .
  - Decrease  $w_n$  if  $\mathbf{x}_n$  was classified correctly by  $h_m(\mathbf{x})$ .
- Make predictions using the final combined model



$$H(\mathbf{x}) = \text{sign} \left( \sum_{m=1}^M \alpha_m h_m(\mathbf{x}) \right)$$

# AdaBoost: Detailed Training Algorithm



1. Initialization: Set  $w_n^{(1)} = \frac{1}{N}$  for  $n = 1, \dots, N$ .

2. For  $m = 1, \dots, M$  iterations

a) Train a new weak classifier  $h_m(\mathbf{x})$  using the current weighting coefficients  $\mathbf{W}^{(m)}$  by minimizing the weighted error function

$$J_m = \sum_{n=1}^N w_n^{(m)} I(h_m(\mathbf{x}_n) \neq t_n) \quad I(A) = \begin{cases} 1, & \text{if } A \text{ is true} \\ 0, & \text{else} \end{cases}$$

b) Estimate the weighted error of this classifier on  $\mathbf{X}$ :

$$\epsilon_m = \frac{\sum_{n=1}^N w_n^{(m)} I(h_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^N w_n^{(m)}}$$

c) Calculate a weighting coefficient for  $h_m(\mathbf{x})$ :

$$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}$$

d) Update the weighting coefficients:

$$w_n^{(m+1)} = w_n^{(m)} \exp \{ \alpha_m I(h_m(\mathbf{x}_n) \neq t_n) \}$$

# AdaBoost: Recognition

- Evaluate all selected weak classifiers on test data.

$$h_1(\mathbf{x}), \dots, h_m(\mathbf{x})$$

- Final classifier is weighted combination of selected weak classifiers:

$$H(\mathbf{x}) = \text{sign} \left( \sum_{m=1}^M \alpha_m h_m(\mathbf{x}) \right)$$

- **Very simple procedure!**
  - Less than 10 lines in Matlab!
  - But works extremely well in practice...

# Example: Face Detection

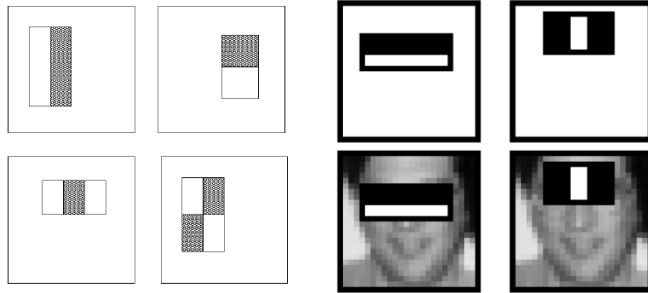
- Frontal faces are a good example of a class where global appearance models + a sliding window detection approach fit well:
  - Regular 2D structure
  - Center of face almost shaped like a “patch”/window



- Now we'll take AdaBoost and see how the Viola-Jones face detector works

# Feature extraction

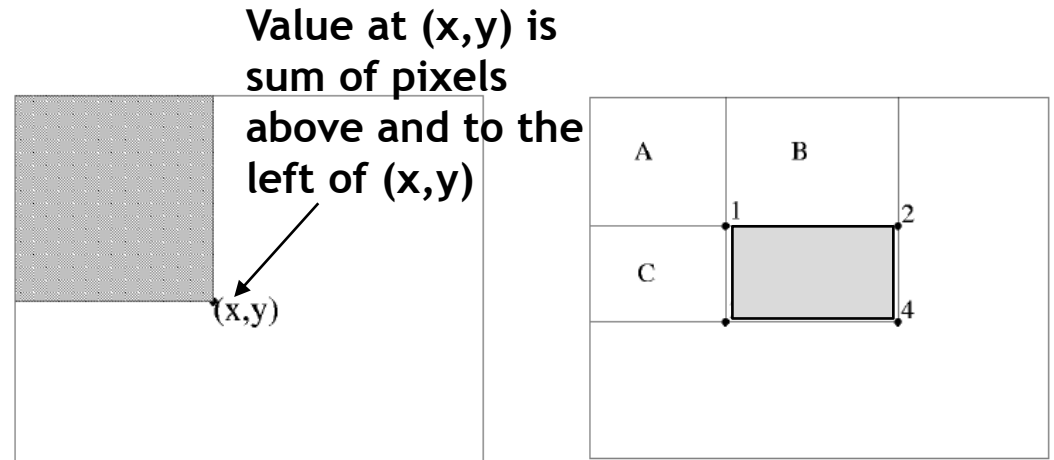
## “Rectangular” filters



Feature output is difference between adjacent regions

Efficiently computable with integral image: any sum can be computed in constant time

Avoid scaling images → scale features directly for same cost

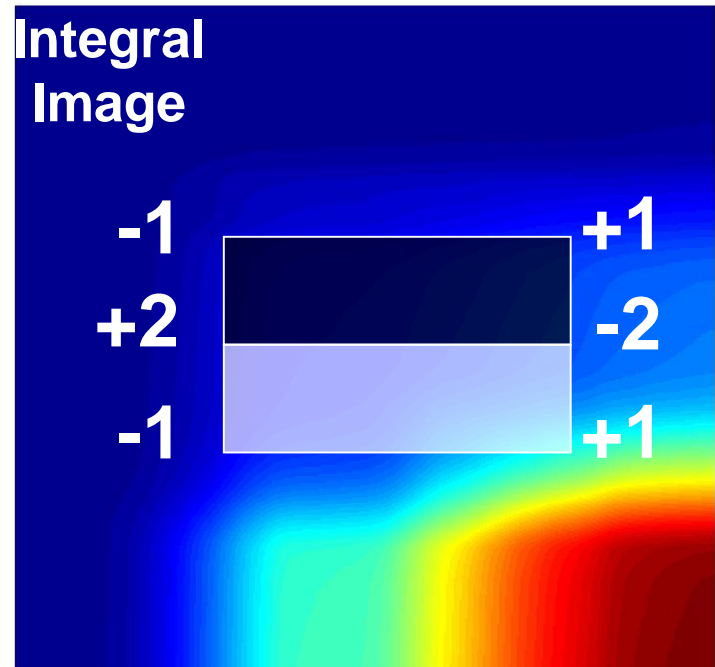
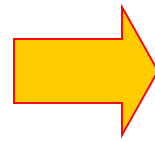
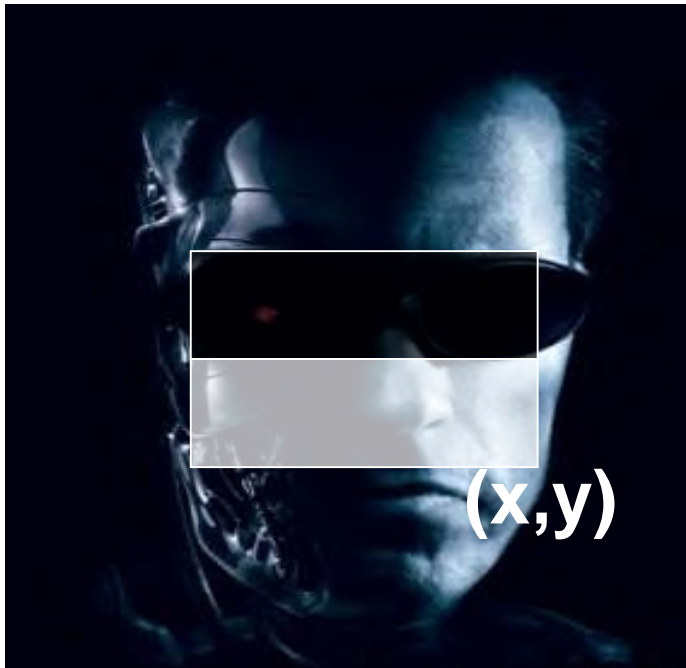


Integral image

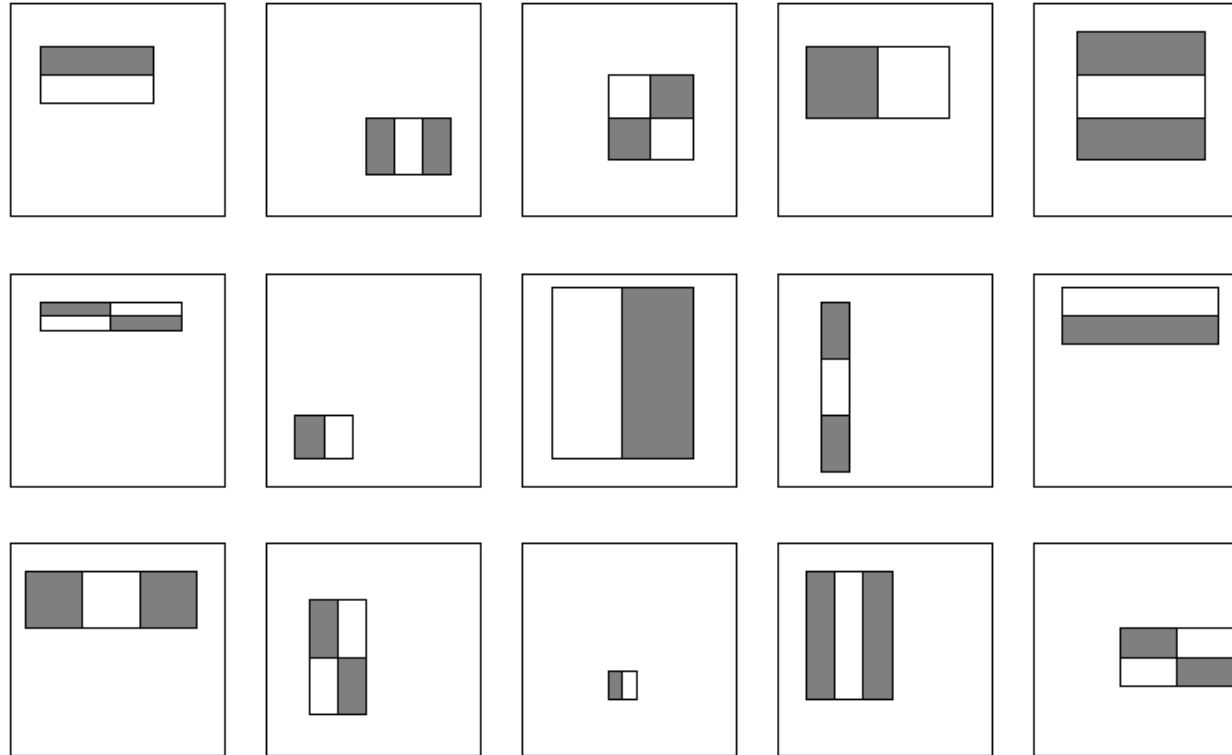
$$\begin{aligned}
 D &= 1 + 4 - (2 + 3) \\
 &= A + (A + B + C + D) - (A + C + A + B) \\
 &= D
 \end{aligned}$$



# Example



# Large Library of Filters



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

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

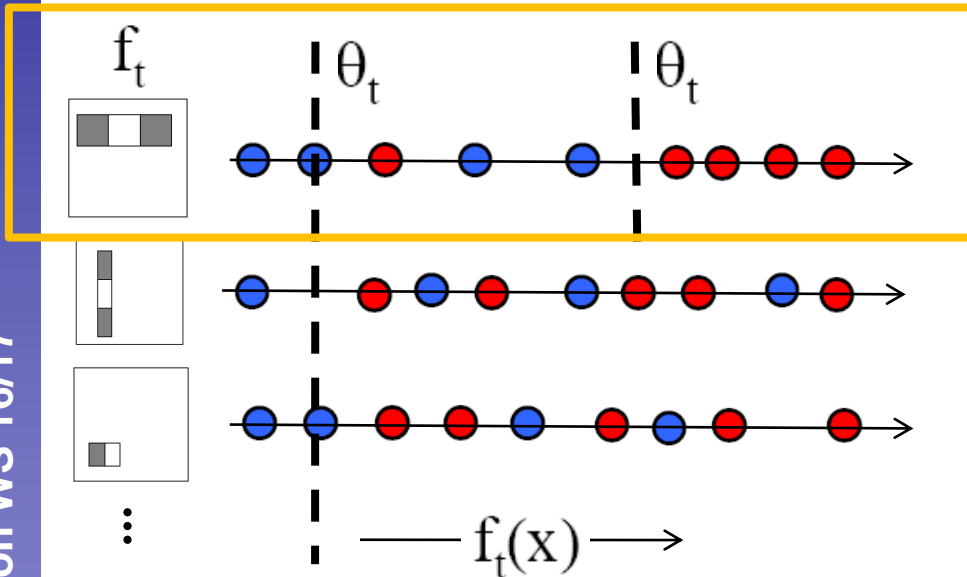
Use AdaBoost both to select the informative features and to form the classifier

Weak classifier:

filter output  $> \theta$ ?

# AdaBoost for Feature+Classifier Selection

- Want to select the single rectangle feature and threshold that best separates **positive** (faces) and **negative** (non-faces) training examples, in terms of *weighted* error.



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

Resulting weak classifier:

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

For next round, reweight the examples according to errors, choose another filter/threshold combo.

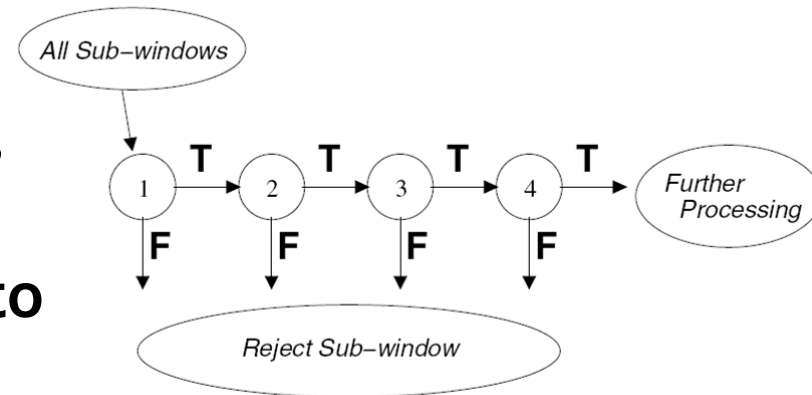
# AdaBoost for Efficient Feature Selection

- Image features = weak classifiers
- For each round of boosting:
  - Evaluate each rectangle filter on each example
  - Sort examples by filter values
  - Select best threshold for each filter (min error)
    - Sorted list can be quickly scanned for the optimal threshold
  - Select best filter/threshold combination
  - Weight on this features is a simple function of error rate
  - Reweight examples

P. Viola, M. Jones, [Robust Real-Time Face Detection](#), IJCV, Vol. 57(2), 2004.  
(first version appeared at CVPR 2001)

# Cascading Classifiers for Detection

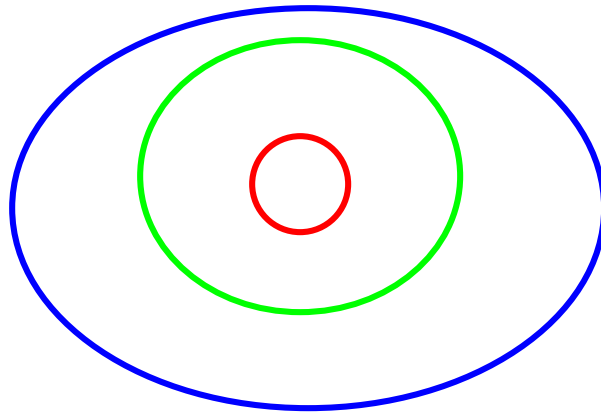
- Even if the filters are fast to compute, each new image has a lot of possible windows to search.
- For efficiency, apply less accurate but faster classifiers first to immediately discard windows that clearly appear to be negative; e.g.,
  - Filter for promising regions with an initial inexpensive classifier
  - Build a chain of classifiers, choosing cheap ones with low false negative rates early in the chain



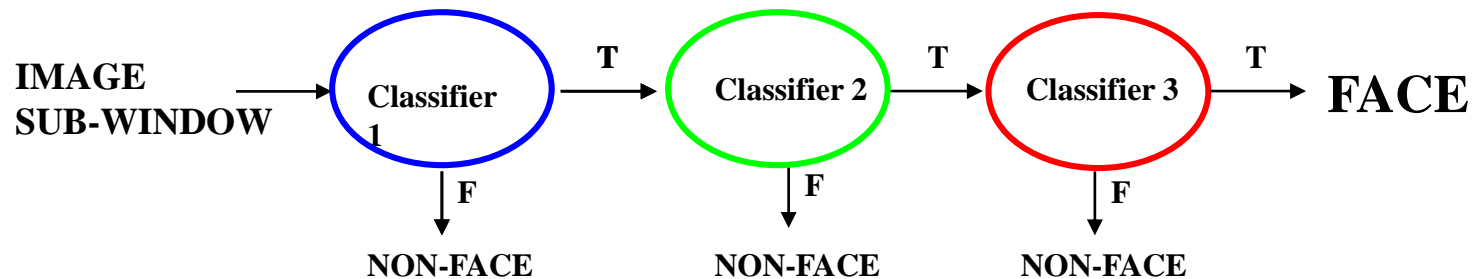
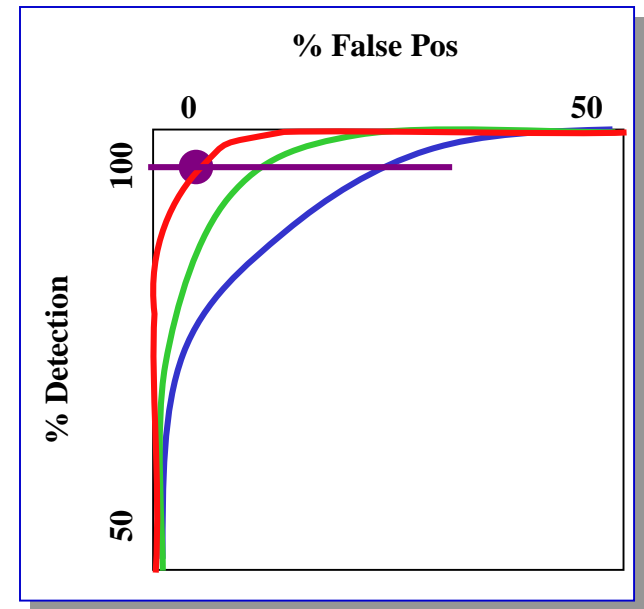
[Fleuret & Geman, IJCV 2001]  
[Rowley et al., PAMI 1998]  
[Viola & Jones, CVPR 2001]

# Cascading Classifiers

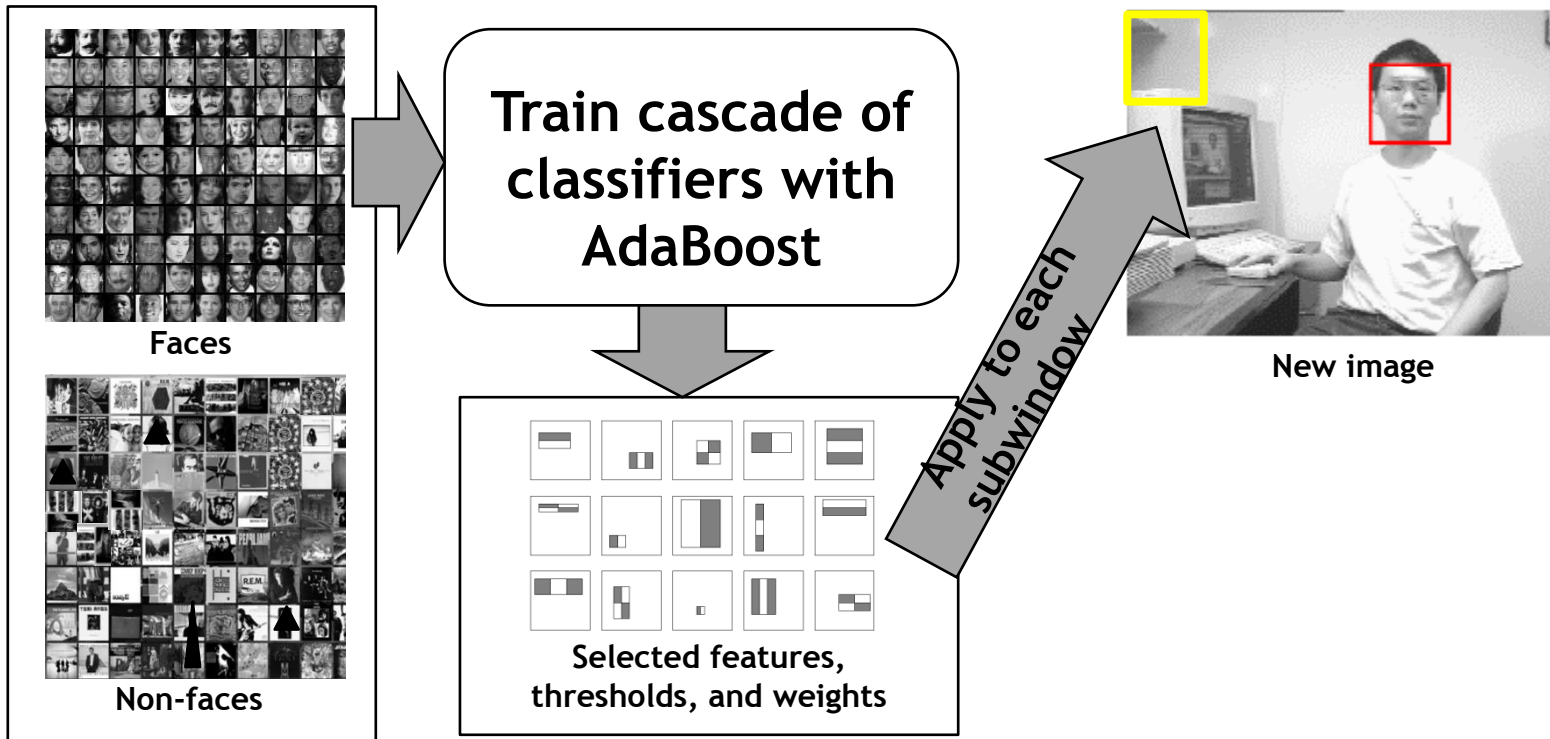
- Chain classifiers that are progressively more complex and have lower false positive rates:



Receiver operating characteristic

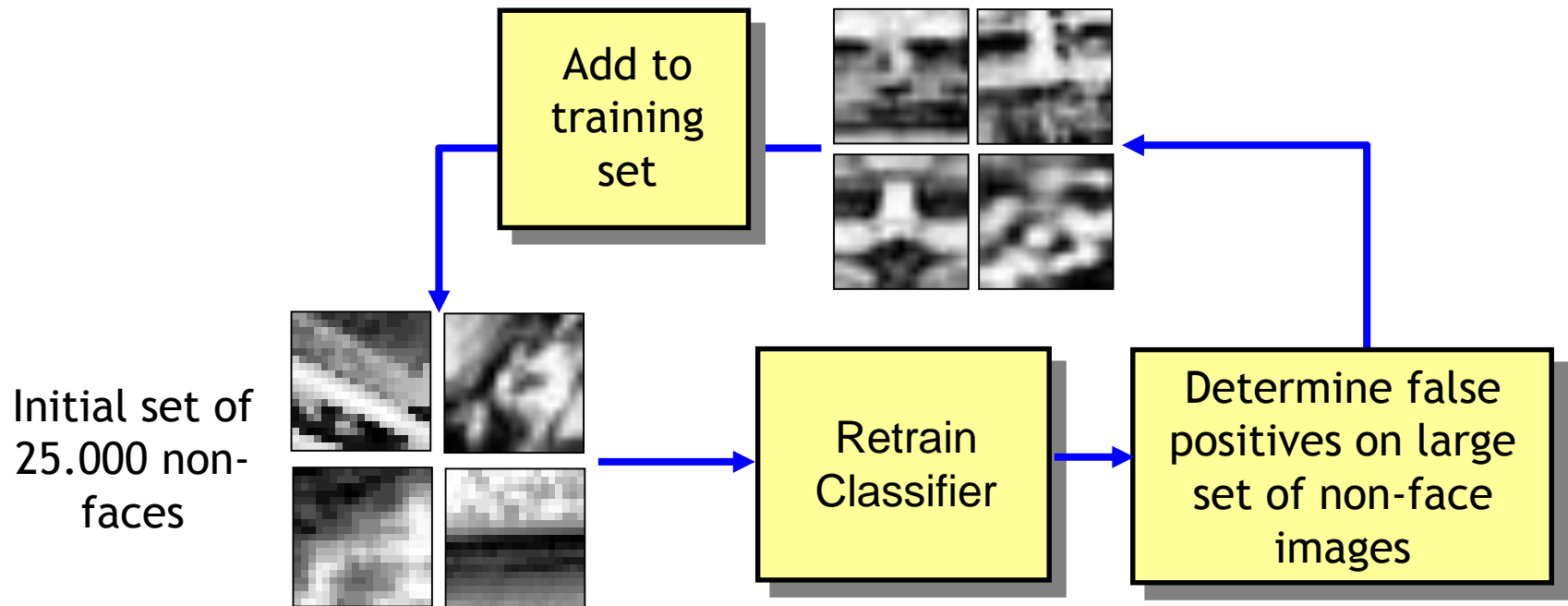


# Viola-Jones Face Detector: Summary



- Train with 5K positives, 350M negatives
- Real-time detector using 38 layer cascade
- 6061 features in final layer
- [Implementation available in OpenCV:  
<http://sourceforge.net/projects/opencvlibrary/>]

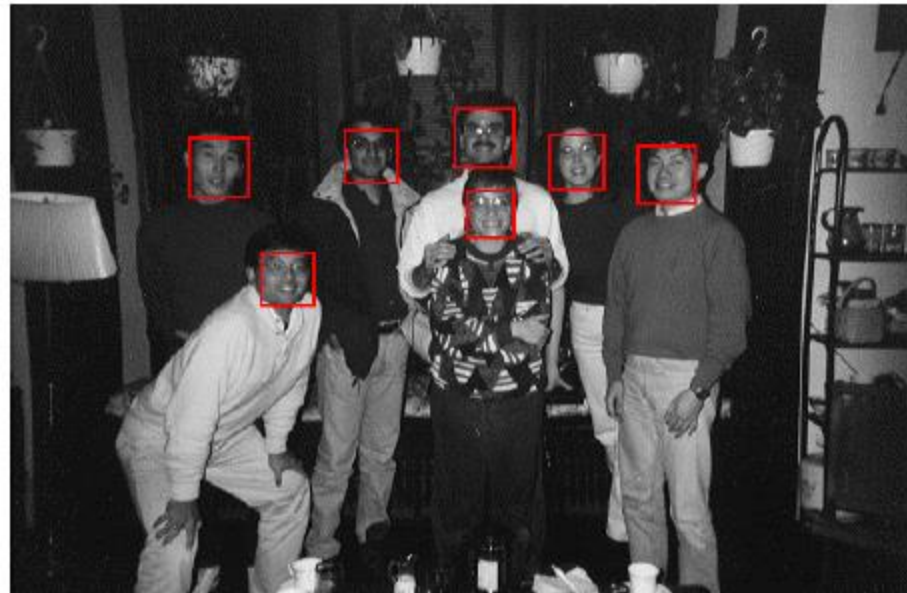
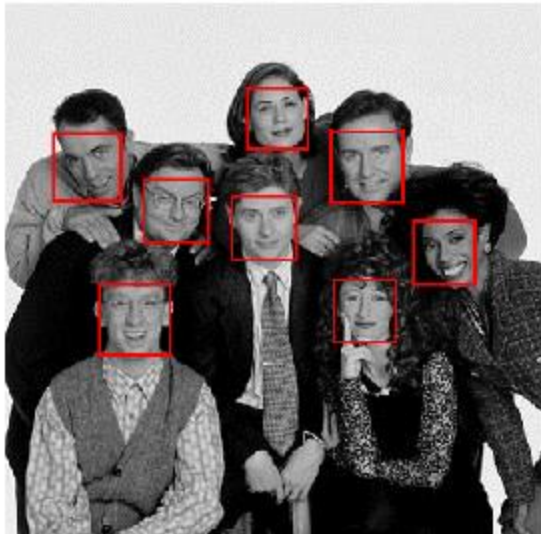
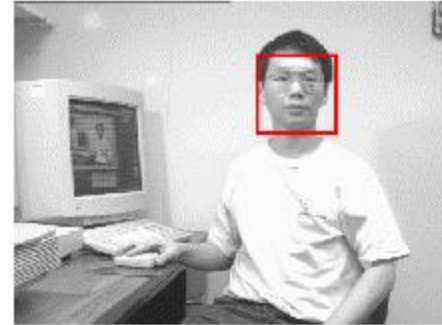
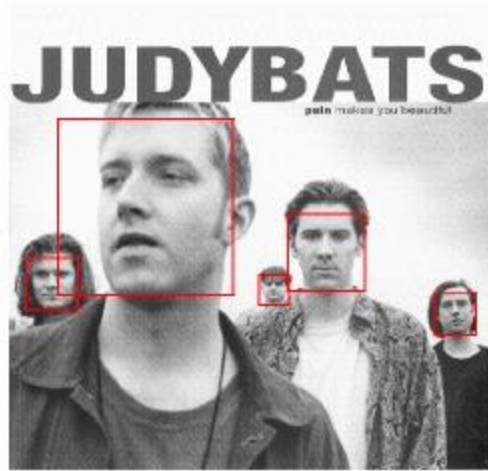
# Practical Issue: Bootstrapping



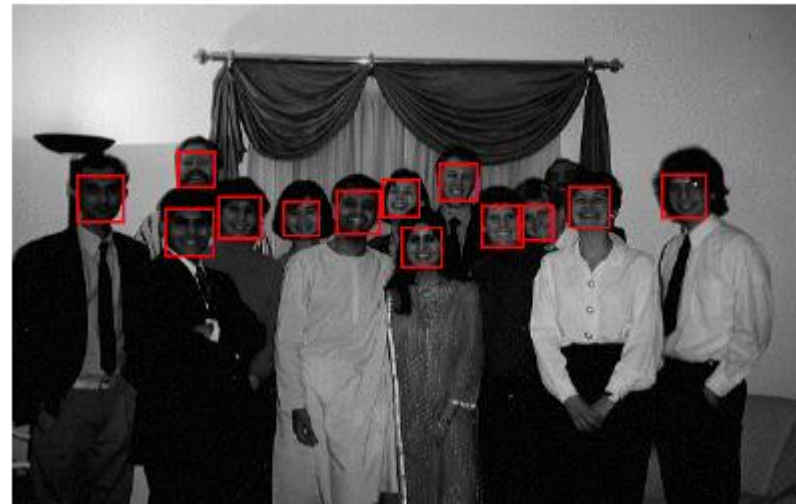
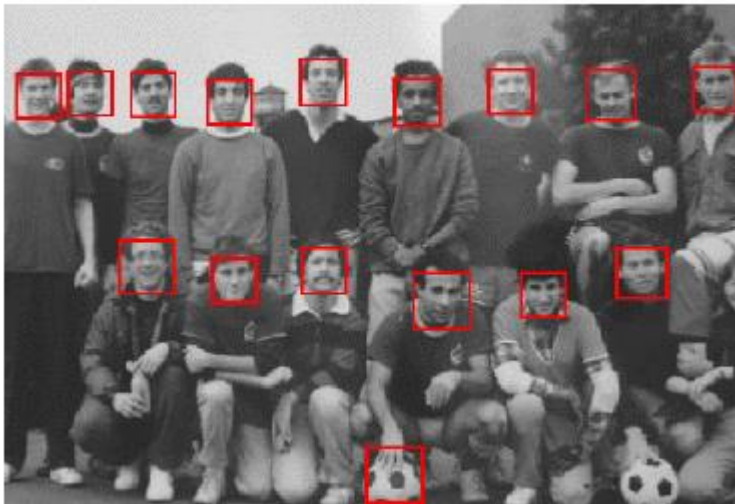
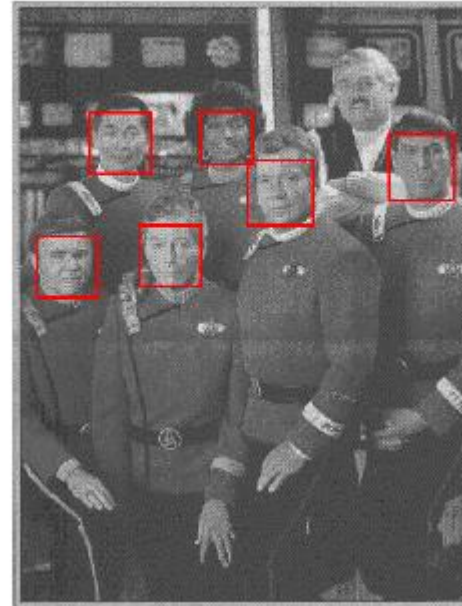
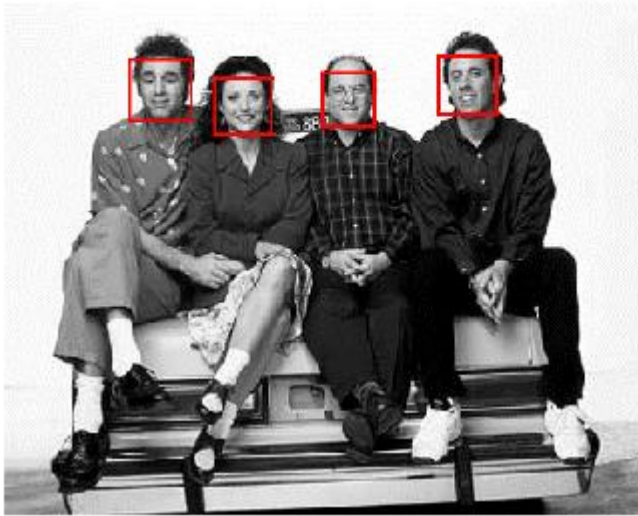
- **Problem: 1 face in 116'440 examined windows**
  - Can easily find negative examples, but which ones are useful?
  - Apply iterative training approach
  - False positives on negative validation images are included in training set as “hard negatives”



# Viola-Jones Face Detector: Results



# Viola-Jones Face Detector: Results



# Viola-Jones Face Detector: Results

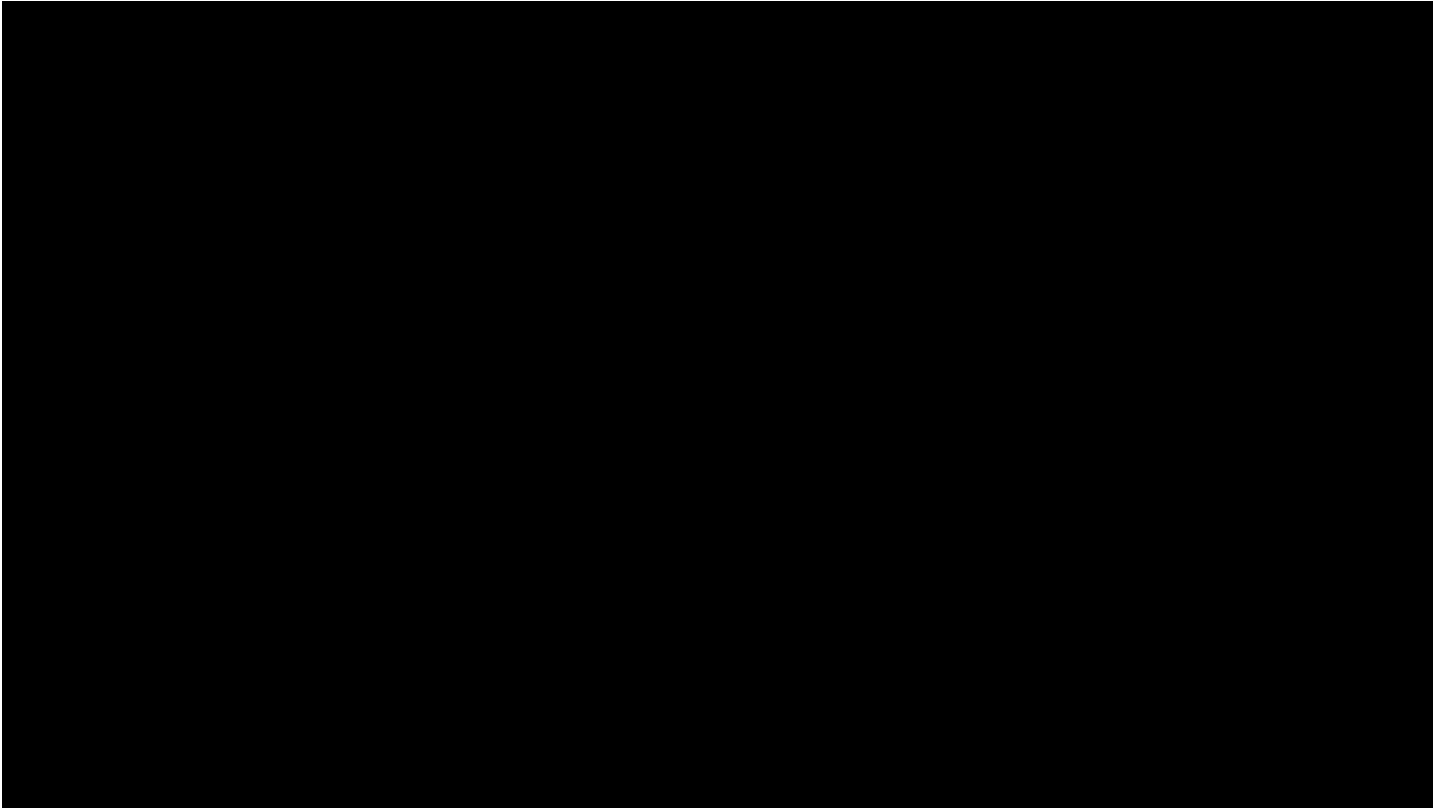


# You Can Try It At Home...

- The Viola & Jones detector was a huge success
  - First real-time face detector available
  - Many derivative works and improvements
- C++ implementation available in OpenCV [Lienhart, 2002]
  - <http://sourceforge.net/projects/opencvlibrary/>
- Matlab wrappers for OpenCV code available, e.g. here
  - <http://www.mathworks.com/matlabcentral/fileexchange/19912>

P. Viola, M. Jones, [Robust Real-Time Face Detection](#), IJCV, Vol. 57(2), 2004

# Example Application



Frontal faces detected and then tracked, character names inferred with alignment of script and subtitles.

Everingham, M., Sivic, J. and Zisserman, A.

"Hello! My name is... Buffy" - Automatic naming of characters in TV video, BMVC 2006.

<http://www.robots.ox.ac.uk/~vgg/research/nface/index.html>

# Summary: Sliding-Windows

- Pros

- Simple detection protocol to implement
- Good feature choices critical
- Past successes for certain classes
- Good detectors available (Viola & Jones, HOG, etc.)

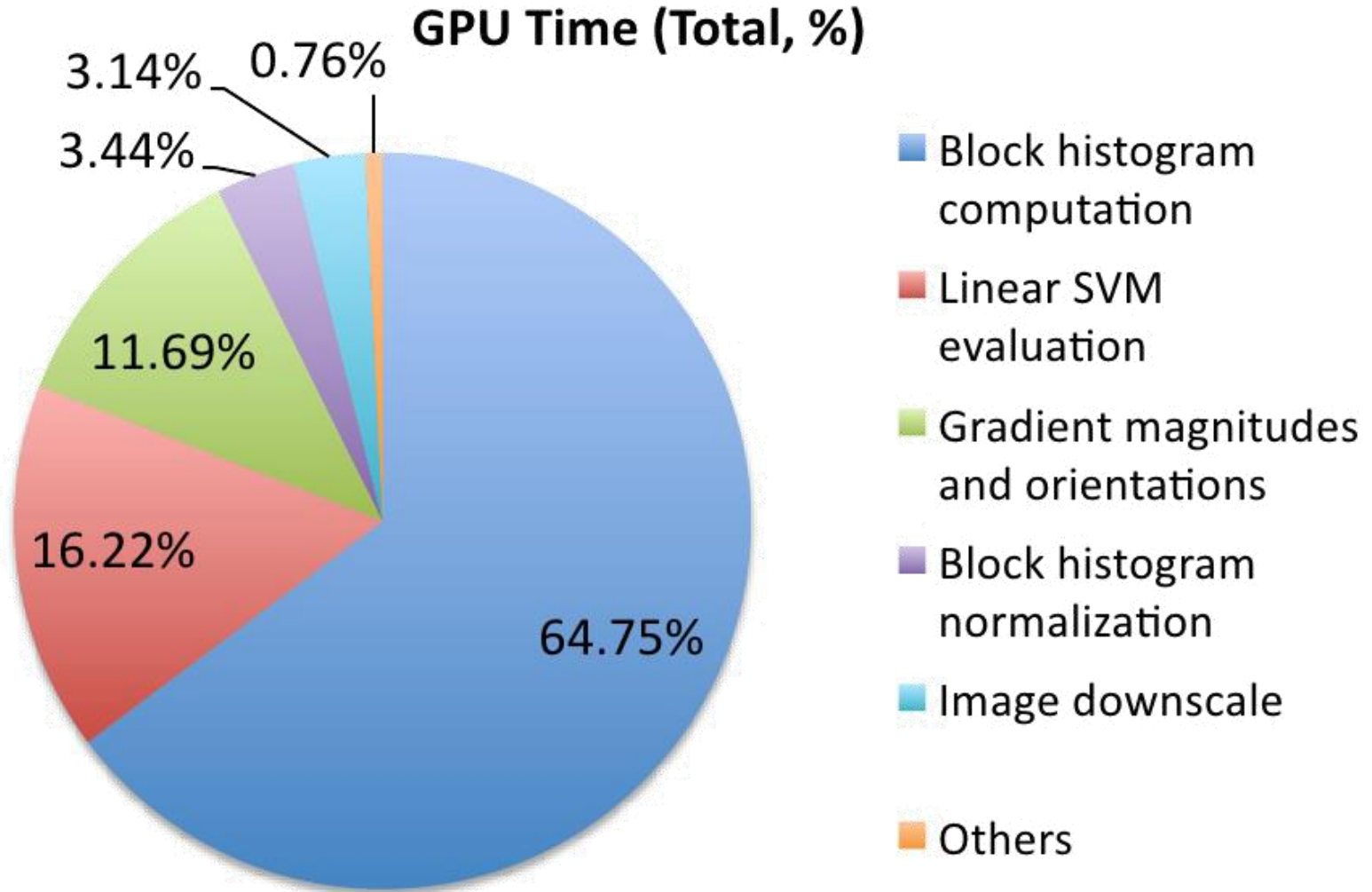
- Cons/Limitations

- High computational complexity
  - For example: 250,000 locations x 30 orientations x 4 scales = 30,000,000 evaluations!
  - This puts tight constraints on the classifiers we can use.
  - If training binary detectors independently, this means cost increases linearly with number of classes.
- With so many windows, false positive rate better be low

# Feature Computation Trade-Off

- **Linear SVM Detectors**
  - Same computations performed for each image window
  - It pays off to precompute the features once
  - Complex features can be used
- **AdaBoost Cascaded Detectors**
  - Potentially different computations for each window location
  - May be more efficient to evaluate the features on-the-fly for each image window
  - If cascading shall be used, simple features are preferable

# What Slows Down HOG (CUDA Implem.)



- Results from fastHOG (10fps) [Prisacariu & Reid 2009]



# Limitations: Low Training Resolutions

- Many (older) S/W detectors operate on tiny images
  - Viola&Jones: 24×24 pixels
  - Torralba et al.: 32×32 pixels
  - Dalal&Triggs: 64×96 pixels (notable exception)
- Main reasons
  - Training efficiency (exhaustive feature selection in AdaBoost)
  - Evaluation speed
  - Want to recognize objects at small scales
- But...
  - Limited information content available at those resolutions
  - Not enough support to compensate for occlusions!

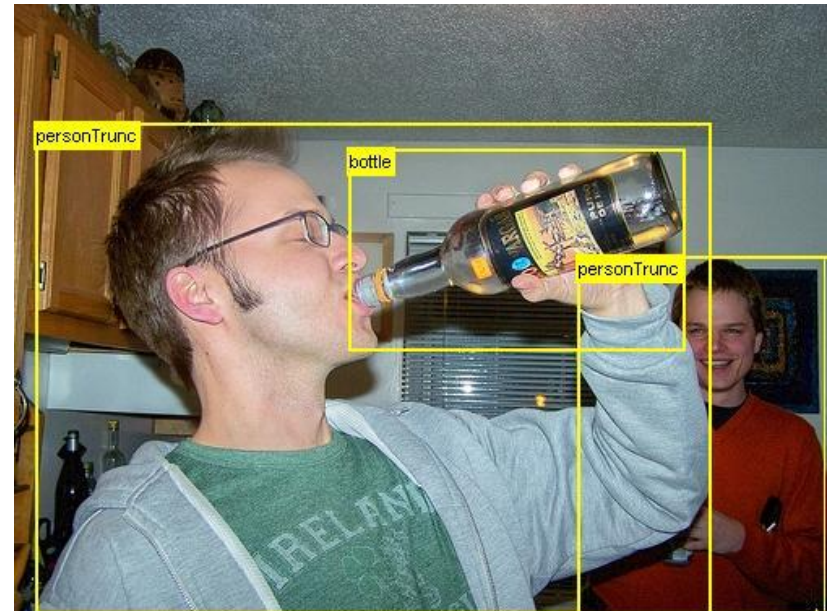
# Limitations: Changing Aspect Ratios

- Sliding window requires fixed window size
  - Basis for learning efficient cascade classifier
- How to deal with changing aspect ratios?
  - Fixed window size
    - ⇒ Wastes training dimensions
  - Adapted window size
    - ⇒ Difficult to share features
  - “Squashed” views [Dalal&Triggs]
    - ⇒ Need to squash test image, too



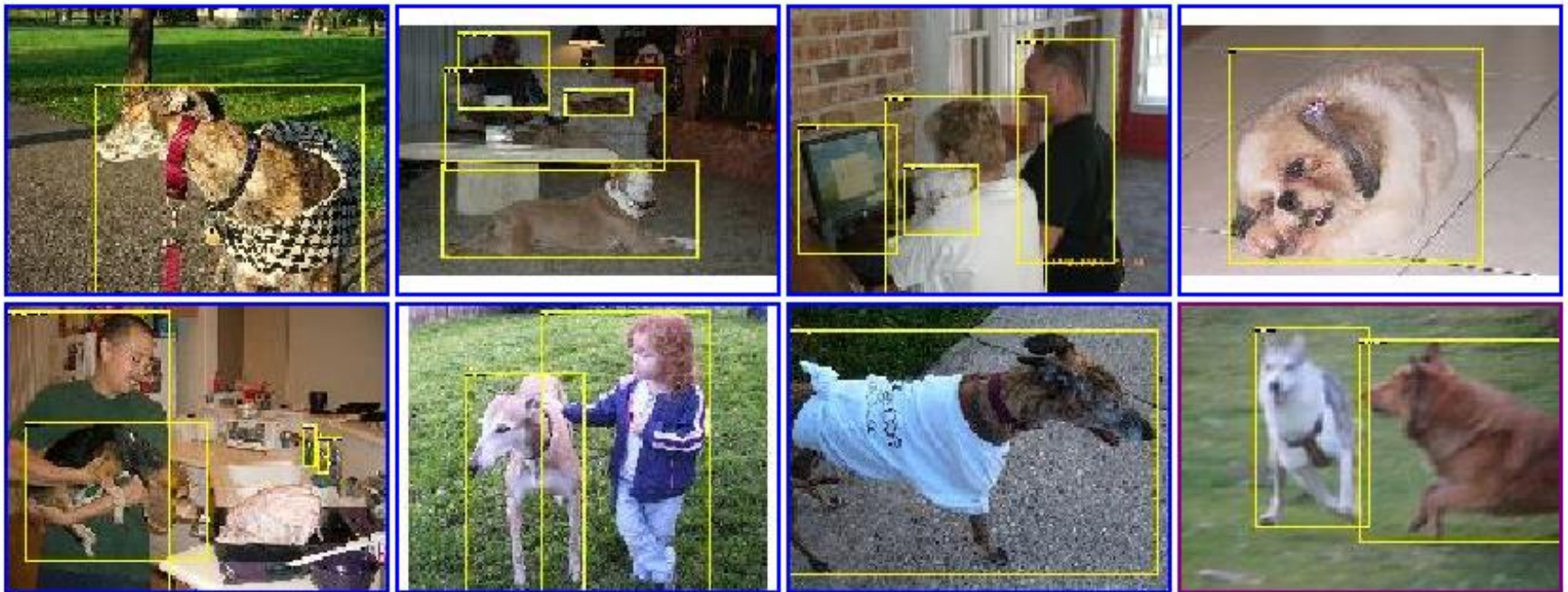
# Limitations (continued)

- 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



# References and Further Reading

- Read the Viola-Jones paper
  - P. Viola, M. Jones,  
[Robust Real-Time Face Detection](#),  
IJCV, Vol. 57(2), 2004.  
(first version appeared at CVPR 2001)
- Viola-Jones Face Detector
  - C++ implementation available in OpenCV [Lienhart, 2002]
    - <http://sourceforge.net/projects/opencvlibrary/>
  - Matlab wrappers for OpenCV code available, e.g. here
    - <http://www.mathworks.com/matlabcentral/fileexchange/19912>
- HOG Detector
  - Code available: <http://pascal.inrialpes.fr/soft/olt/>