

Classification & Objection Detection

Kuan-Wen Chen

Pattern Recognition

Introduction

- Examples of PR in our daily life:
 - recognize a face
 - read handwritten characters
 - understand spoken words
 - identify car keys in the pocket by feel
 - decide whether a fruit is ripe by its smell
 - etc.

The Goal of PR

- To design and build machines that can recognize pattern. (as an engineering field)
- To gain deeper understanding and appreciation for PR systems in the natural world – particularly in human. (as a science)

Applications

- Face detection, recognition, and verification
- Speech and speaker recognition
- Fingerprint identification
- OCR and document analysis
- Industrial Inspection
- Medical diagnostics
- DNA sequence analysis
- etc.

An Example

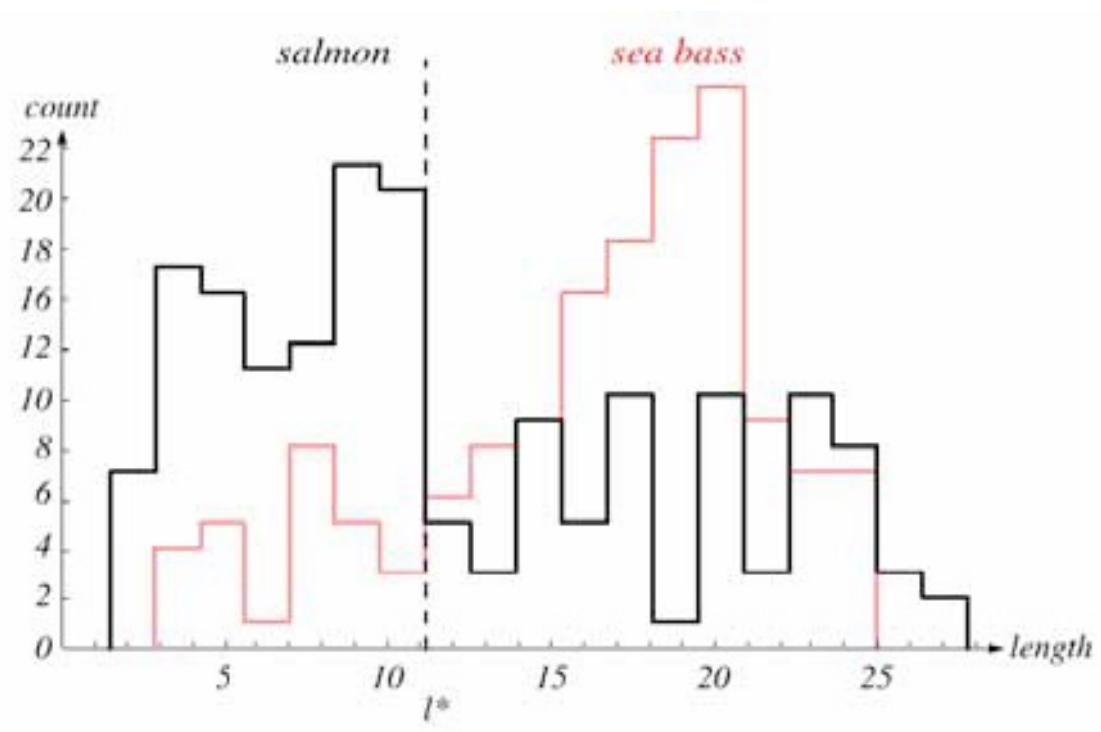
- A pilot project: separate sea bass from salmon.
- Physical differences between the two types of fishes:
length, lightness, width, number and shape of fins,
position of the mouth, etc.



possible **features** to be used in the classifier

An Example

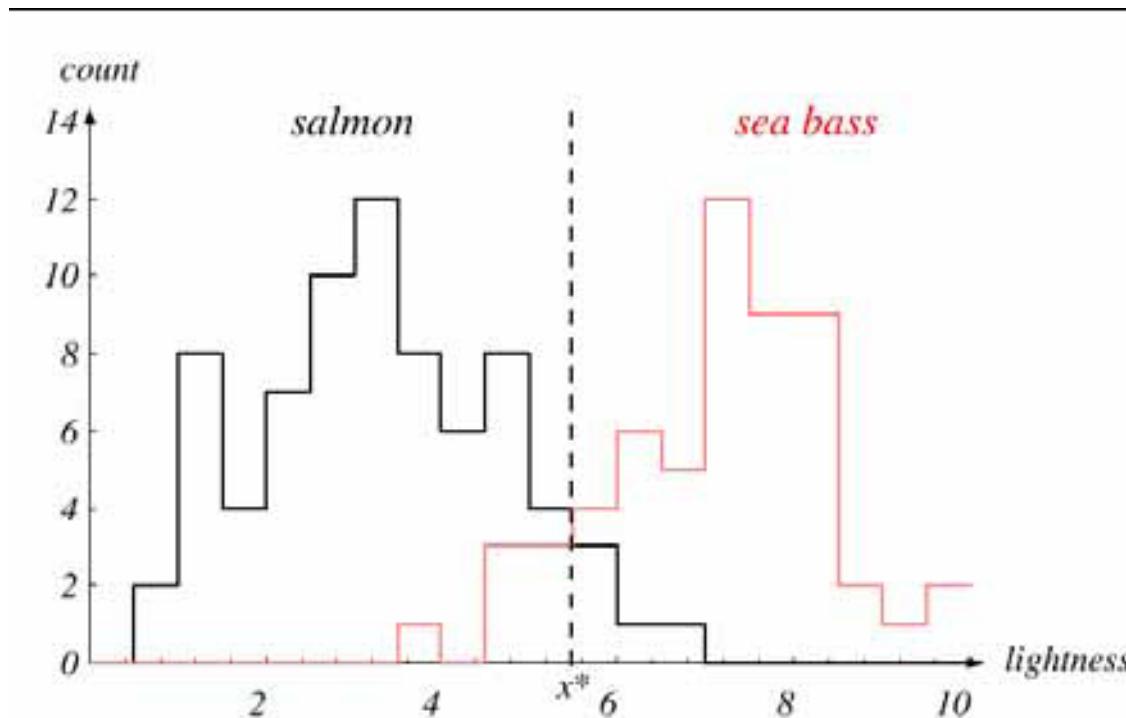
- Histogram for the **length** feature - obtained with some training samples



difficult to choose a good threshold, l^*

An Example

- Histogram for the lightness feature - average **lightness** of the fish scales



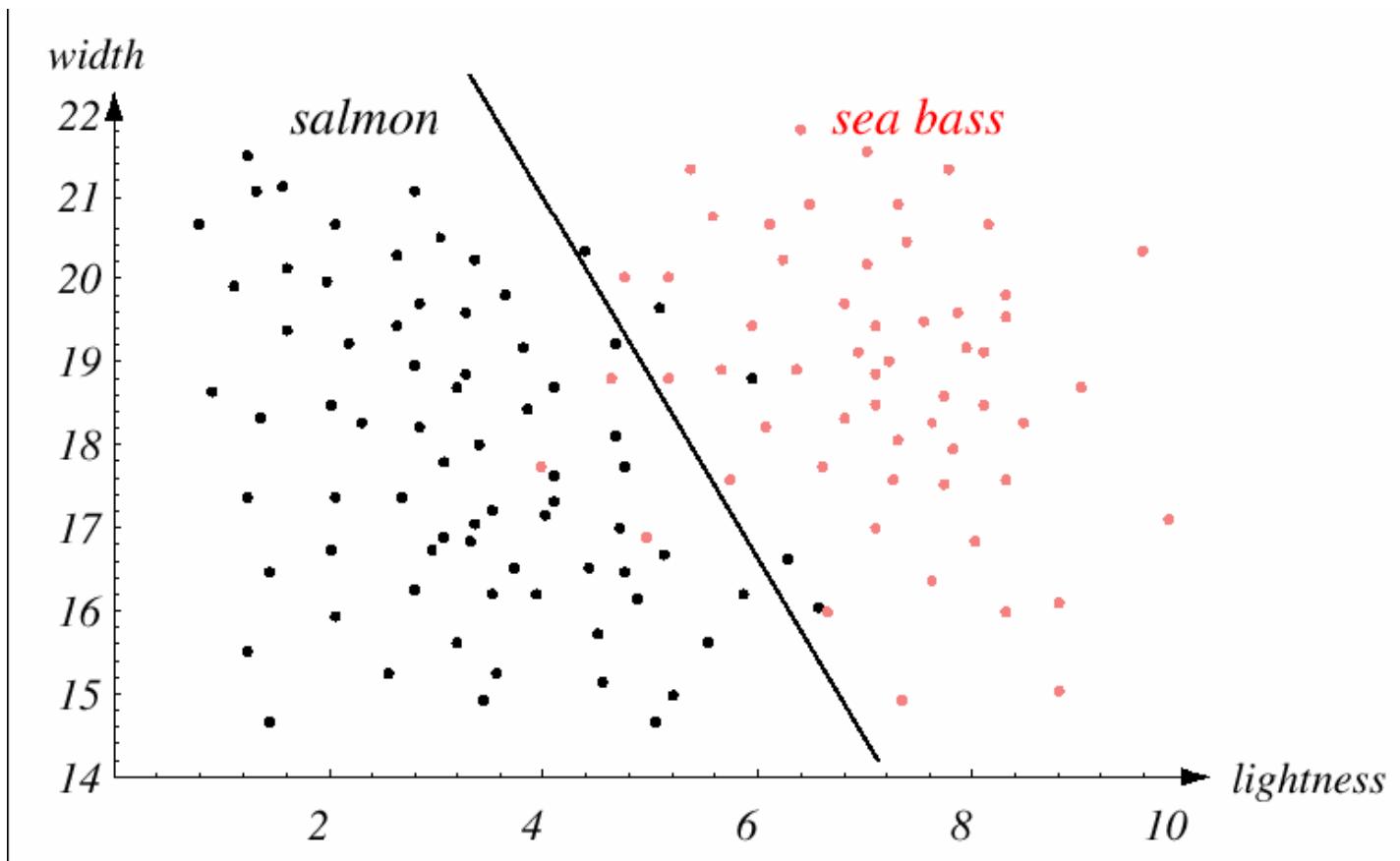
easier to choose a good threshold, x^*

An Example

- Our task is to find the **decision rule** (or to set the **decision boundary**) that can minimize an overall cost.
decision theory
- To improve the performance, try to use more features simultaneously.
 - e.g., sea bass are typically wider than salmon
 - choose lightness and width as features

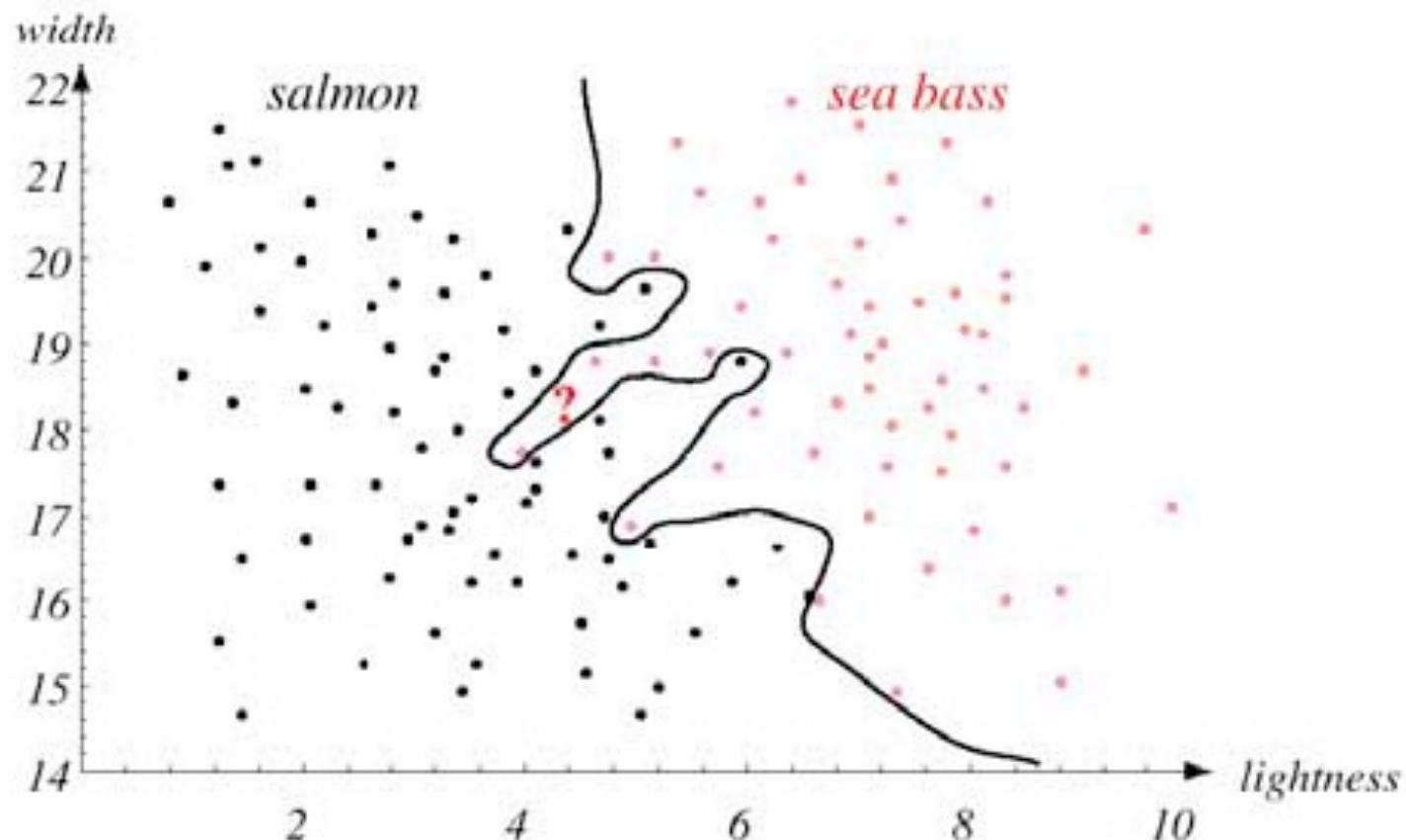
An Example

- Two-dimensional feature space



Generalization

- Our goal is to design a classifier to suggest actions when presented with *novel* patterns, i.e., fish not yet seen.



An Example

- To improve the performance, try to use **more features** simultaneously.
 - e.g., sea bass are typically wider than salmon
 - choose lightness and width as features

Does it more features always lead to better results?

No!!

“Curse” of dimensionality

Curse of dimensionality

Do you know, scientists say we live in 11 dimensional space...



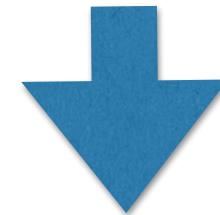
Is that why your brain is almost empty?



aptnk.in



100 x 100



10,000-dimensional vector

Dimension reduction

PR Systems

- A typical PR system consists of the following five components:
 - Sensing (e.g., cameras, microphones.)
 - Preprocessing (Segmentation and Grouping)
 - Feature Extraction
 - Classification
 - Post Processing

The Design Cycle

- Collect Data
- Choose Features
- Choose Model
- Train Classifier
- Evaluate Classifier

How **deep learning** helps for PR:

- Feature
- Model
- Classifier

Bayesian Decision Theory

- Maximum *a posteriori* probability (M.A.P.) classifier is the minimum-error-rate classifier

The Bayes rule:

$$P(\omega_j | x) = P(x | \omega_j) P(\omega_j) / P(x)$$

$$\text{Posterior} = \frac{\text{Likelihood} * \text{Prior}}{\text{Evidence}}$$

Learning

- Supervised Learning
 - Parametric Approach
 - Non-Parametric Approach
 - Parzen Windows, k-NN Estimation
- Unsupervised Learning (*clustering*)
 - Parametric Approach
 - Non-Parametric Approach
 - Clustering
- Reinforcement Learning (learning with a critic)

Foreground Detection



Foreground Detection

- Foreground detection (also known as **background subtraction**) is to detect changes in image sequences.
- All detection techniques are based on modelling the background of the image, i.e. set the background and detect which changes occur.
- Defining the background can be very difficult when it contains shapes, shadows, and moving objects.



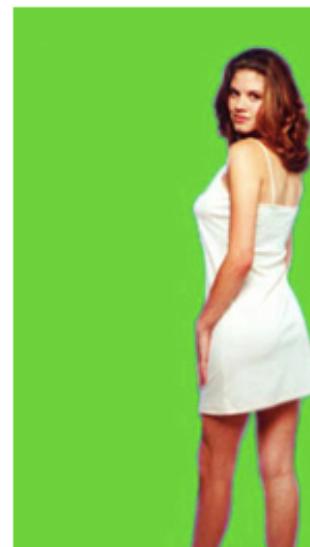
Foreground Detection

- The rationale in the approach is that of detecting the moving objects from the difference between the **current frame** and a **reference frame**, often called “background image”, or “background model”.
- A reliable and robust background subtraction algorithm should handle:
 - Sudden or gradual illumination changes
 - High frequency, repetitive motion in the background (such as tree leaves, flags, waves, . . .)
 - Long-term scene changes (a car is parked for a month).



Purpose of Background Subtraction

- Segment the image into foreground and background
- Reduce problem set for further processing
 - Only process part of picture that contains the relevant information
- Add a virtual background



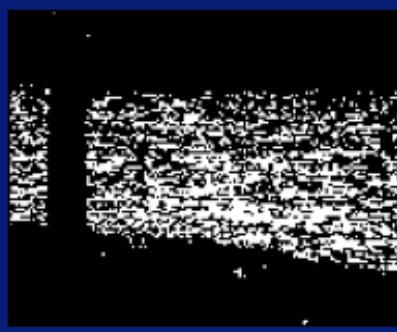
Encountered Problems

- Lighting
 - Shadows
 - Gradual/Sudden illumination changes
- Camouflage
- Moving objects
 - Foreground aperture
- Foreground object becomes motionless



Encountered Problems

- Lighting
 - Shadows
 - Gradual/Sudden illumination changes



**Rippling
Water**

**Water
Surface**

**Camera
Jitter**

**Waving
Trees**

Simple Approach

Image at time t :

$$I(x, y, t)$$



Background at time t :

$$B(x, y, t)$$



|

-

$| > Th$

1. Estimate the background for time t .
2. Subtract the estimated background from the input frame.
3. Apply a threshold, Th , to the absolute difference to get the **foreground mask**.

Problem: how to have a clear background?

Frame Difference

- Background is estimated to be the previous frame.
Background subtraction equation then becomes:

$$\begin{aligned}B(x, y, t) &= I(x, y, t - 1) \\&\Downarrow \\|I(x, y, t) - I(x, y, t - 1)| &> Th\end{aligned}$$

- Depending on the object structure, speed, frame rate and global threshold, this approach may or may **not** be useful (usually **not**).



-



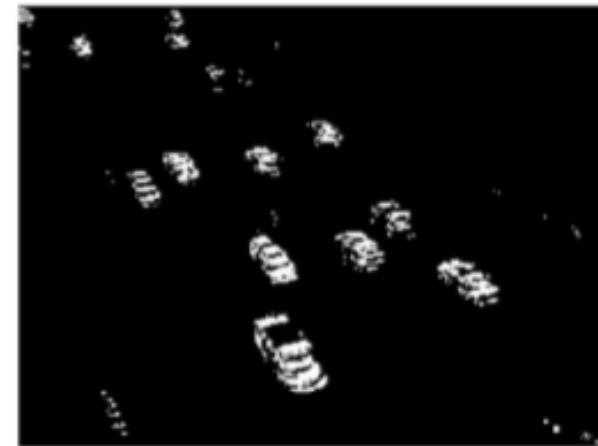
$| > Th$

Frame Difference

$Th = 25$



$Th = 50$



$Th = 100$



$Th = 200$



Mean Filter

- ▶ In this case the background is the mean of the previous n frames:

$$B(x, y, t) = \frac{1}{n} \sum_{i=0}^{n-1} I(x, y, t - i)$$
$$\downarrow$$
$$|I(x, y, t) - \frac{1}{n} \sum_{i=0}^{n-1} I(x, y, t - i)| > Th$$

- ▶ For $n = 10$:

Estimated Background



Foreground Mask



Mean Filter

- ▶ For $n = 20$:

Estimated Background



Foreground Mask



- ▶ For $n = 50$:

Estimated Background



Foreground Mask



Median Filter

- ▶ Assuming that the background is more likely to appear in a scene, we can use the median of the previous n frames as the background model:

$$B(x, y, t) = \text{median}\{I(x, y, t - i)\}$$



$$|I(x, y, t) - \text{median}\{I(x, y, t - i)\}| > Th \text{ where } i \in \{0, \dots, n - 1\}.$$

- ▶ For $n = 10$:

Estimated Background



Foreground Mask



Median Filter

- ▶ For $n = 20$:

Estimated Background



Foreground Mask



- ▶ For $n = 50$:

Estimated Background



Foreground Mask



Advantages & Shortcomings

Advantages:

- Extremely easy to implement and use!
- All pretty fast.
- Corresponding background models are not constant, they change over time.

Disadvantages:

- Accuracy of frame differencing depends on object speed and frame rate!
- Mean and median background models have relatively high memory requirements.
 - In case of the mean background model, this can be handled by a **running average**:

$$B(x, y, t) = \frac{t-1}{t} B(x, y, t-1) + \frac{1}{t} I(x, y, t)$$

or more generally:

$$B(x, y, t) = (1 - \alpha) B(x, y, t-1) + \alpha I(x, y, t)$$

where α is the learning rate.

Advantages & Shortcomings

Disadvantages:

- There is another major problem with these simple approaches:

$$|I(x, y, t) - B(x, y, t)| > Th$$

- There is one global threshold, Th , for all pixels in the image.
- And even a bigger problem: **this threshold is not a function of t .**
- So, these approaches will not give good results in the following conditions:
 - if the background is bimodal,
 - if the scene contains many, slowly moving objects (mean & median),
 - if the objects are fast and frame rate is slow (frame differencing),
 - and if general lighting conditions in the scene change with time!

Advantages & Shortcomings

To represent the background by single value could be unstable especially for non-stationary scenes.

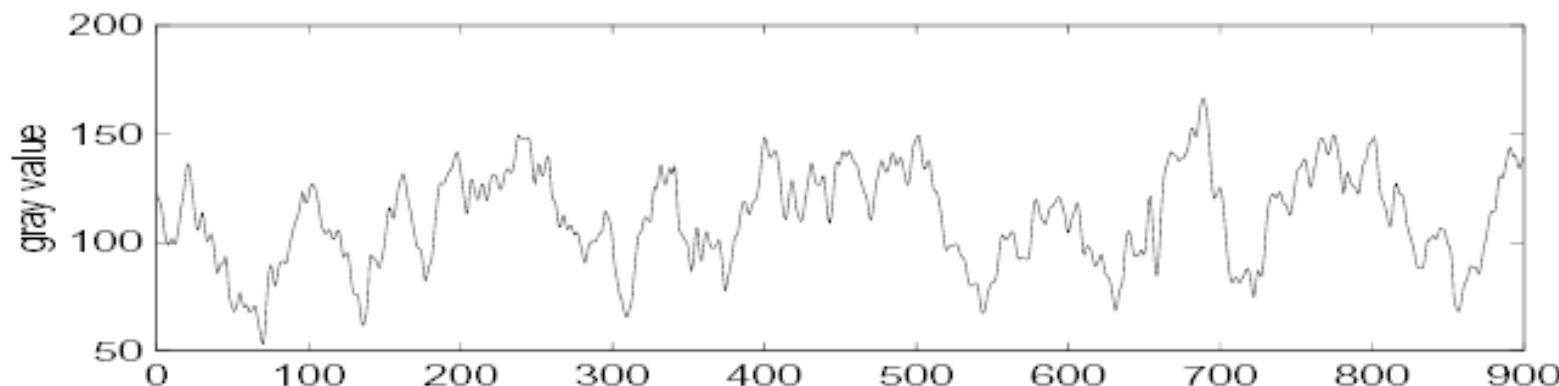


Figure 1: Intensity value overtime



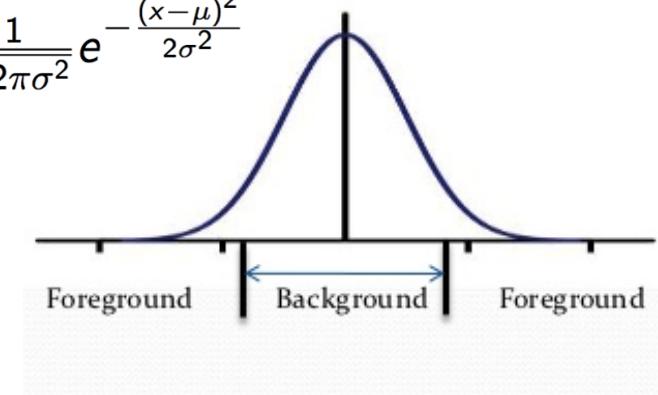
Simple background model

Single Gaussian Model: (parametric model)

- C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfnder: Real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, July 1997.

The background scene is modelled by representing each pixel by a Gaussian model

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



The mean (ϕ) and covariance $U(\phi)$ of each pixel ϕ can be recursively updated as follows

$$\boldsymbol{\mu}^t(\phi) = (1 - \alpha)\boldsymbol{\mu}^{t-1}(\phi) + \alpha I^t(\phi),$$

$$U^t(\phi) = (1 - \alpha)U^{t-1}(\phi) + \alpha \nu(\phi)\nu(\phi)^T$$

α is the learning rate and $\nu(\phi) = (I^t(\phi) - \boldsymbol{\mu}^t(\phi))$

Simple background model

Single Gaussian Model: (parametric model)

- C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: Real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, July 1997.
-

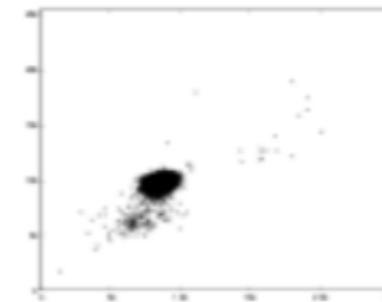
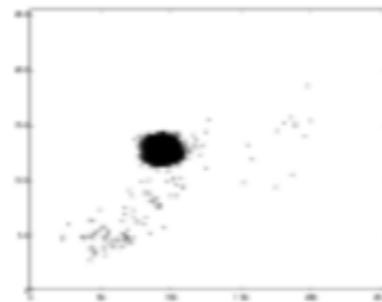
Advantages:

- Easy and fast!

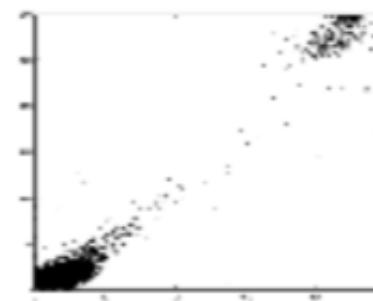
Disadvantages:

- May not be sufficient for representation since some scenes and backgrounds are often dynamic.

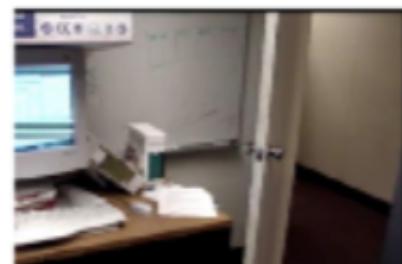
Background modelling



(a)



(b)

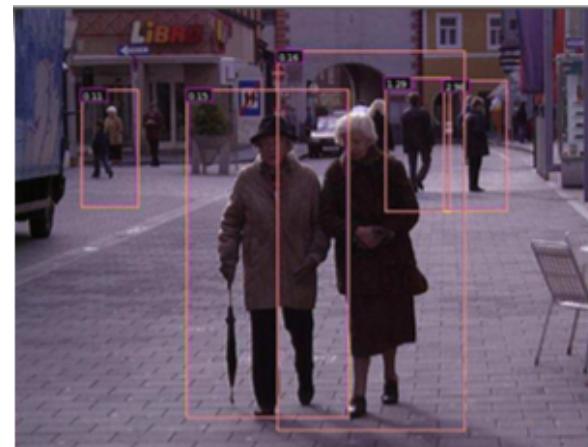
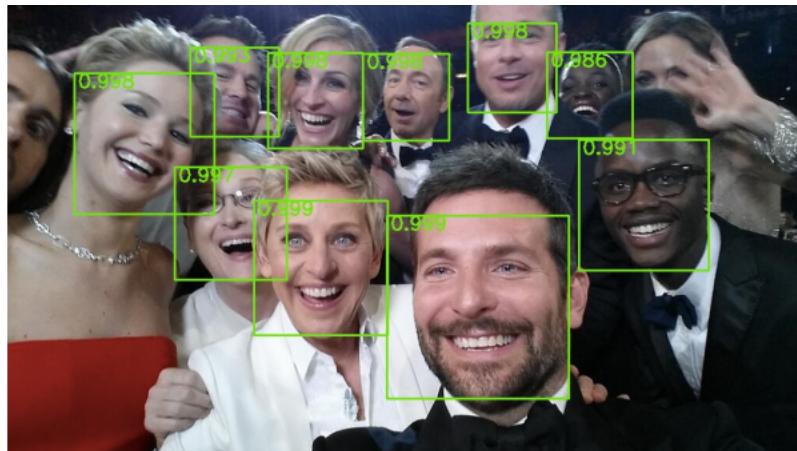


A robust background subtraction algorithm should handle:
lighting changes, repetitive motions from clutter and long-term
scene changes.

OpenCV MoG

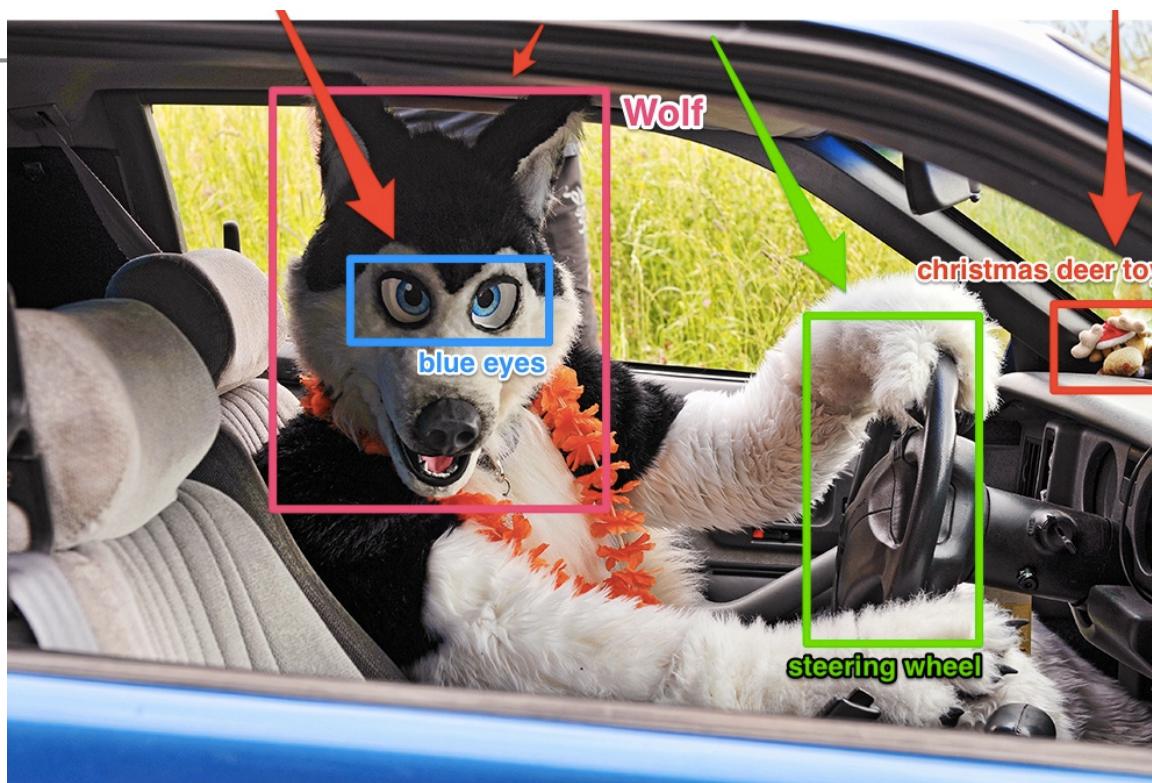


Object Detection



Object Detection

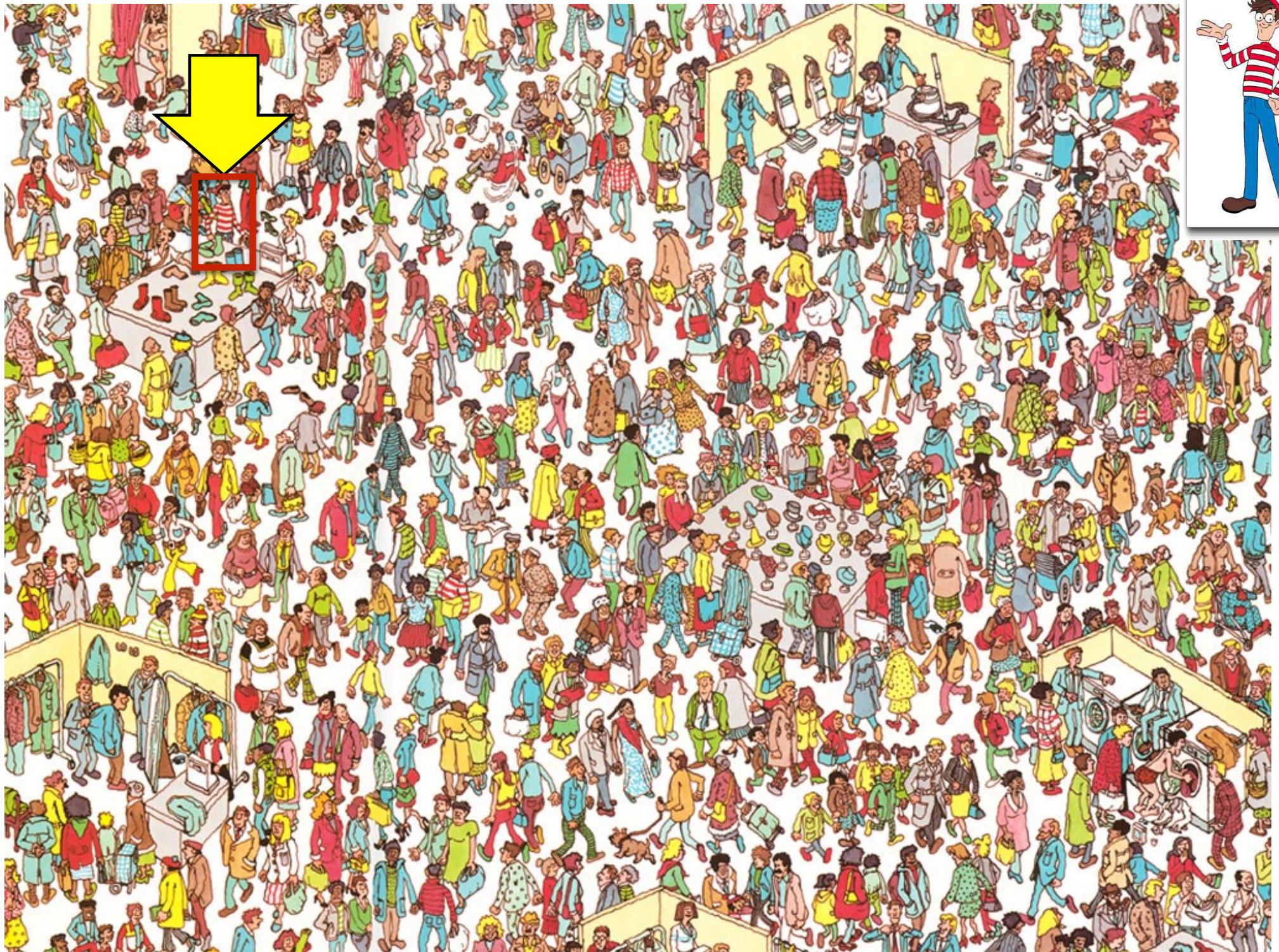
- Deal with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos.
- In ITS, what we are interested in are car, scooter, pedestrian, lane, traffic sign, etc.



Object Detection

- Find the location of an object if it appear in an image
 - Does the object appear?
 - Where is it?

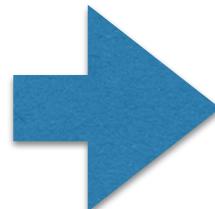




Object Detection

- A standard way is to
 1. save a template or model for what you want to detect

Training images

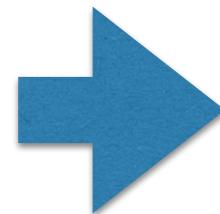
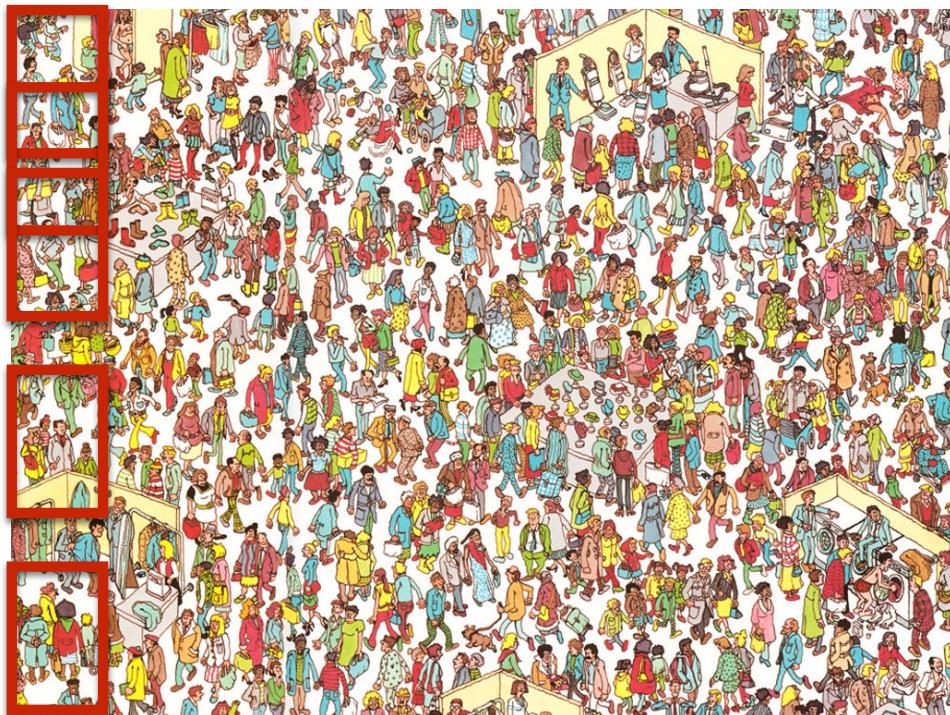


Template or Model

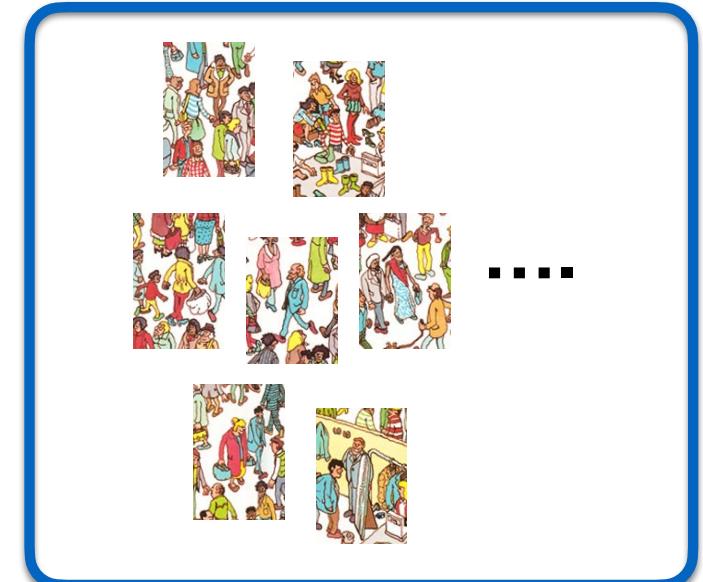


Object Detection

- A standard way is to
 1. save a template or model for what you want to detect
 2. use a sliding window search to generate candidates



Candidates



Object Detection

- A standard way is to
 1. save a template or model for what you want to detect
 2. use a sliding window search to generate candidates
 3. classify the candidates: is it the object or not

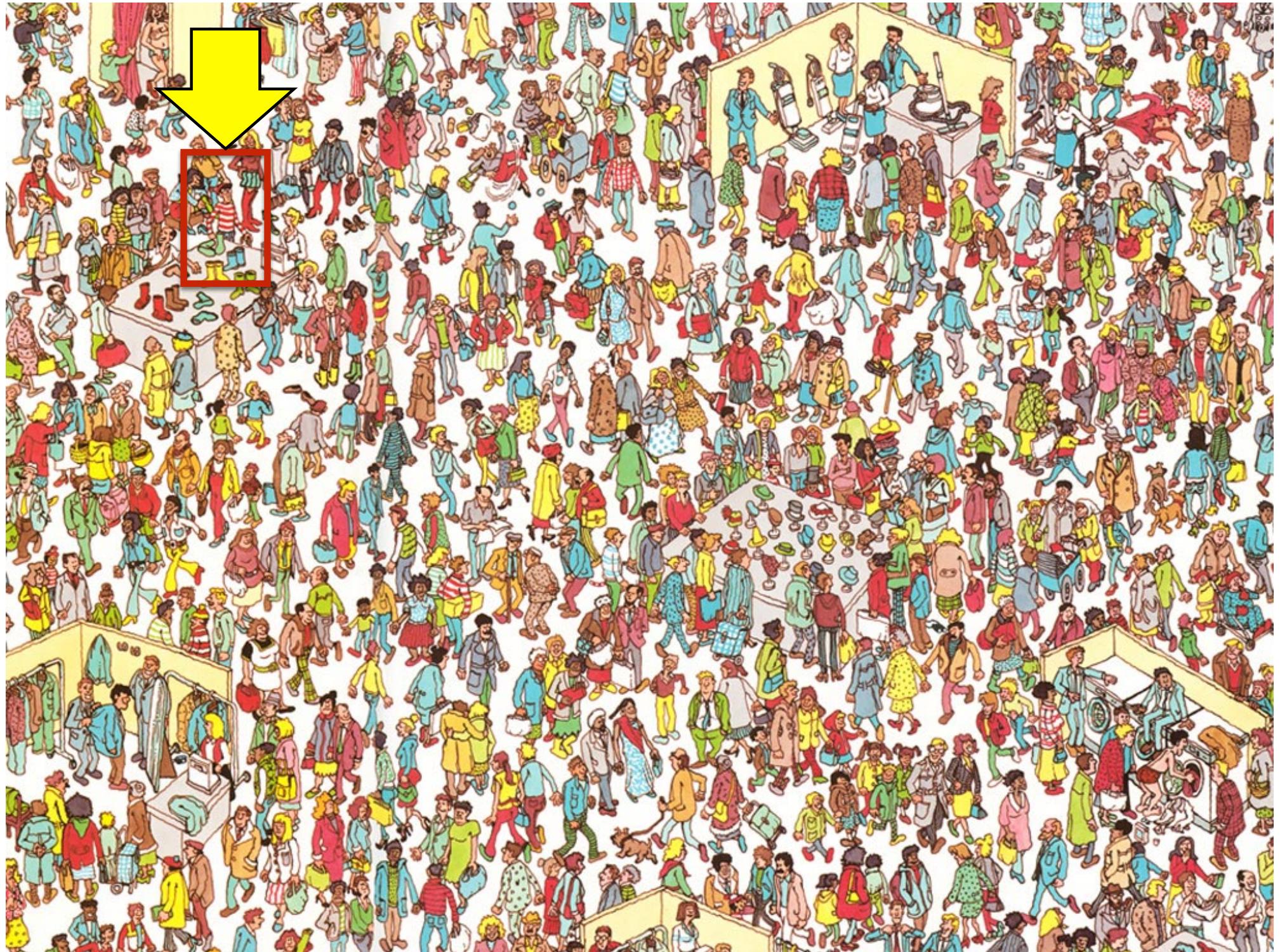
Candidates



Classifier

Yes

No



Object Detection



Challenges

view point variation

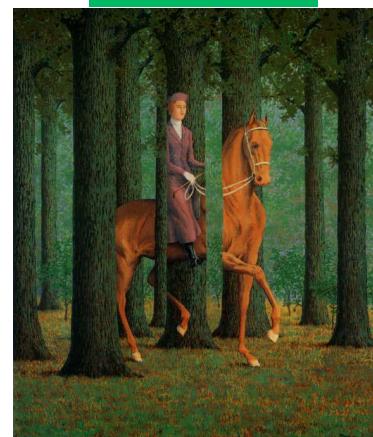


Michelangelo 1475-1564

illumination



occlusion



Magritte, 1957

and small things
from Apple.
(Actual size)

scale



Challenges

deformation



Xu, Beihong 1943

background clutter



Klimt, 1913

Object Detection

- A standard way is to
 1. save a **template** or **model** for what you want to detect
 2. use a sliding window search to generate candidates
 3. **classify** the candidates: is it the object or not



Candidates

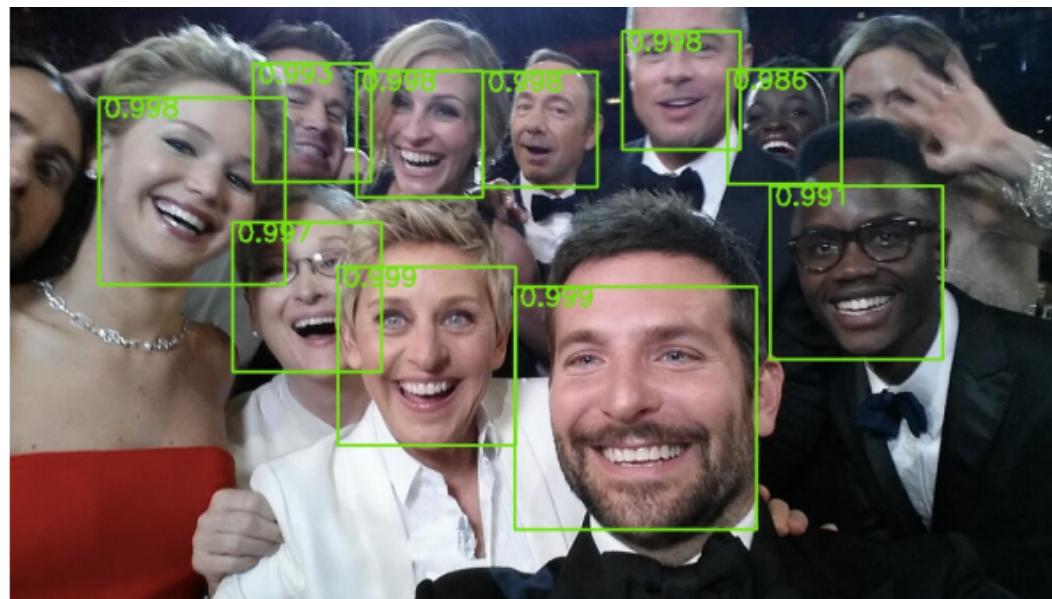


**1. What's the model?
=> features**

**2. What's the classifier?
==> similarity metrics**

**3. Too many candidates
==> speed**

Face Detection & Adaboost

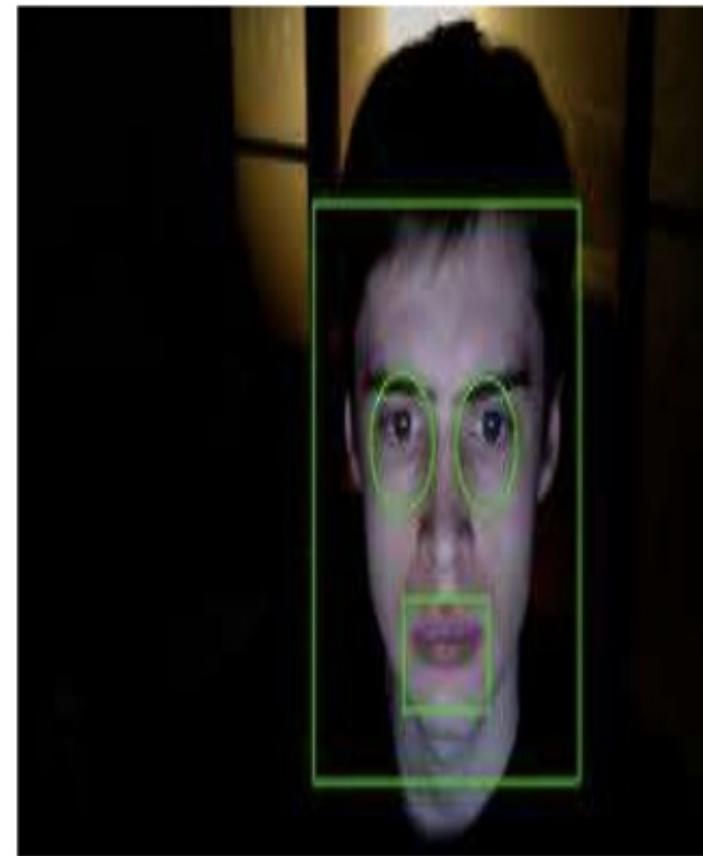


Face detection before 2000

KNOWLEDGE-BASED APPROACH



- It uses human-coded rules to model facial features, such as **two symmetric eyes**, a **nose in the middle** and a **mouth underneath the nose**.



Adaboost

Y. Freund and R. E. Schapire, “A Short Introduction to Boosting,” *Journal of Japanese Society for Artificial Intelligence*, 1999.

Adaboost

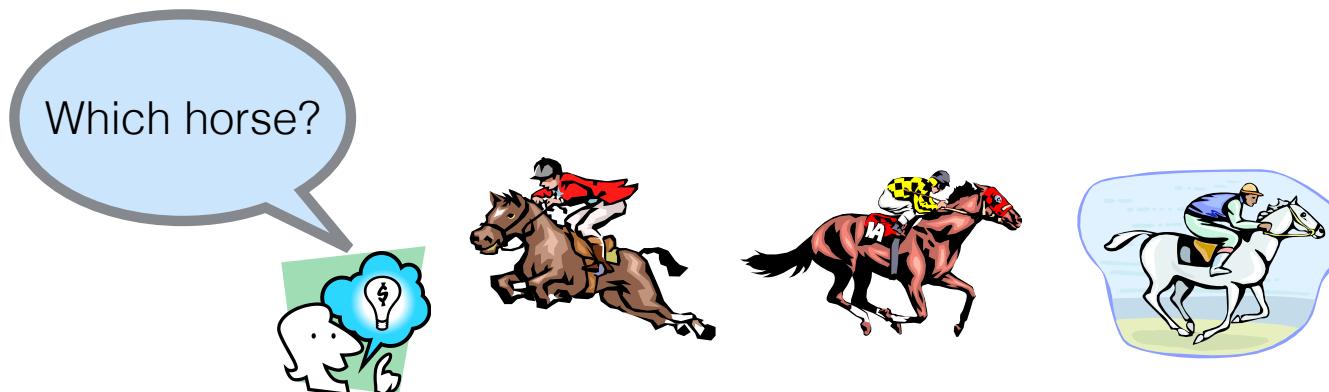


Adaptive Boosting

Y. Freund and R. E. Schapire, “A Short Introduction to Boosting,” *Journal of Japanese Society for Artificial Intelligence*, 1999.

Boosting

- The Horse-Racing Gambler Problem
 - Rules of thumb for a set of races
 - How should we choose the set of races in order to get the best rules of thumb?
 - How should the rules be combined into a single highly accurate prediction rule?

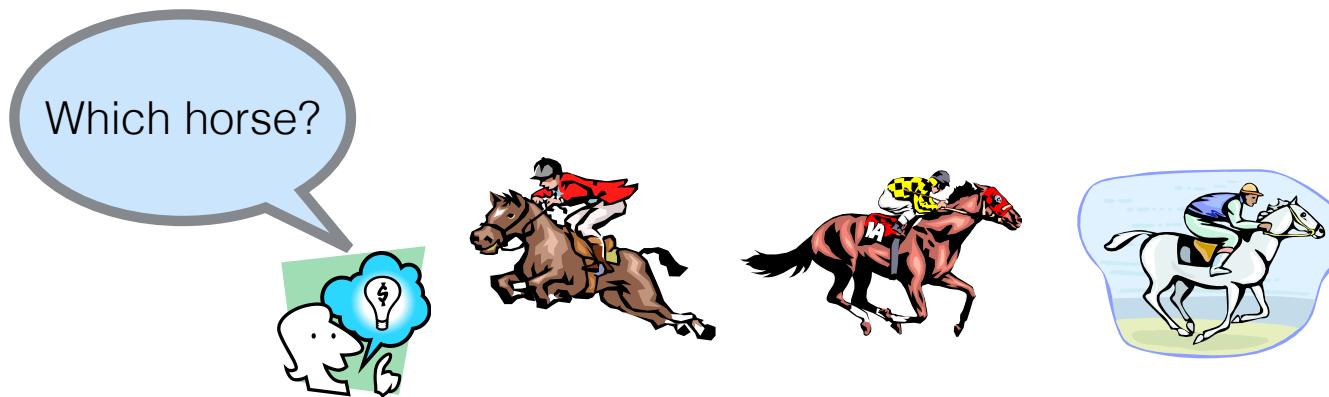


- race record
- current status
- horse owner
- ground type: firm, soft..
- ...

Boosting!

Basic Idea of Boosting

- Hard to find **single highly accurate classification (strong classifier) rule**
- Finding **many rough rules of thumb (weak classifier)** can be a lot easier and more effective than finding a single, highly prediction rule.



- race record
- current status
- horse owner
- ground type: firm, soft..
- ...

Basic Idea of Boosting

- Hard to find **single highly accurate classification rule**
- Finding **many rough rules of thumb** (weak classifier) can be a lot easier and more effective than finding a single, highly prediction rule.

Boosting

- boosting = general method of converting rough rules of thumb into highly accurate prediction rule
- technically:
 - assume given “weak” learning algorithm that can consistently find classifiers (“rules of thumb”) at least slightly better than random, say, accuracy $\geq 55\%$ (in two-class setting)
 - given sufficient data, a boosting algorithm can provably construct single classifier with very high accuracy, say, 99%

Basic Boosting Algorithm

The Boosting Approach

- devise computer program for deriving rough rules of thumb
- apply procedure to subset of examples Q1
- obtain rule of thumb Q2
- apply to 2nd subset of examples
- obtain 2nd rule of thumb
- repeat T times Q3

Basic Boosting Algorithm

Challenges

Q1 how to choose rules of thumb?

- application dependent - face, pedestrian, car...

Q2 how to choose examples on each round?

- concentrate on “hardest” examples (those most often misclassified by previous rules of thumb)

Q3 how to combine rules of thumb in to single classifier?

- take (weighted) majority vote of rules of thumb

Basic Boosting Algorithm

Challenges

Q1 how to choose rules of thumb?

- application dependent - face, pedestrian, car...

Q2 how to choose examples on each round?

- concentrate on “hardest” examples (those most often misclassified by previous rules of thumb)

讓前一輪做錯的盡可能做對



This is the main idea of AdaBoost

Q3 how to combine rules of thumb in to single classifier?

- take (weighted) majority vote of rules of thumb

AdaBoost



Adaptive Boosting

AdaBoost is adaptive in the sense that subsequent classifiers built are tweaked in favor of those instances misclassified by previous classifiers.

AdaBoost Concept

$$\left. \begin{array}{l} h_1(x) \in \{-1, +1\} \\ h_2(x) \in \{-1, +1\} \\ \vdots \\ h_T(x) \in \{-1, +1\} \end{array} \right\} \quad H_T(x) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(x) \right)$$

weak classifiers strong classifier

slightly **better than random**

Weaker Classifiers

$$\left. \begin{array}{l} h_1(x) \in \{-1, +1\} \\ h_2(x) \in \{-1, +1\} \\ \vdots \\ h_T(x) \in \{-1, +1\} \end{array} \right\}$$

weak classifiers

slightly **better than random**

- Each weak classifier learns by considering one simple feature
- T most beneficial features for classification should be selected
- How to
 - define features?
 - select beneficial features?
 - train weak classifiers?
 - manage (weight) training samples?
 - associate weight to each weak classifier?

A Formal Description of Boosting

- given training set $(x_1, y_1), \dots, (x_m, y_m)$
- $y_i \in \{-1, +1\}$ correct label of instance $x_i \in X$
- for $t = 1, \dots, T$:
 - construct distribution D_t on $\{1, \dots, m\}$
 - find weak classifier (“rule of thumb”)
 $h_t : X \rightarrow \{-1, +1\}$
with small error ϵ_t on D_t :
$$\epsilon_t = \Pr_{D_t}[h_t(x_i) \neq y_i]$$
- output final classifier H_{final}

AdaBoost Algorithm

- final classifier:

- $H_{\text{final}}(x) = \text{sign} \left(\sum_t \alpha_t h_t(x) \right)$

- constructing D_t :

- $D_1(i) = 1/m$

$D_t(i)$: probability distribution of x_i 's at time t

- for $t = 1, \dots, T$:

- find weak classifier (“rule of thumb”)

$$h_t : X \rightarrow \{-1, +1\}$$

with small error ϵ_t on D_t : $\epsilon_j = \sum_{i=1}^m D_t(i)[y_i \neq h_j(x_i)]$ minimize weighted error

- given D_t and h_t :

Give error classified patterns more chance for learning.

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases} \\ &= \frac{D_t(i)}{Z_t} \exp(-\alpha_t y_i h_t(x_i)) \end{aligned}$$

一定大於0

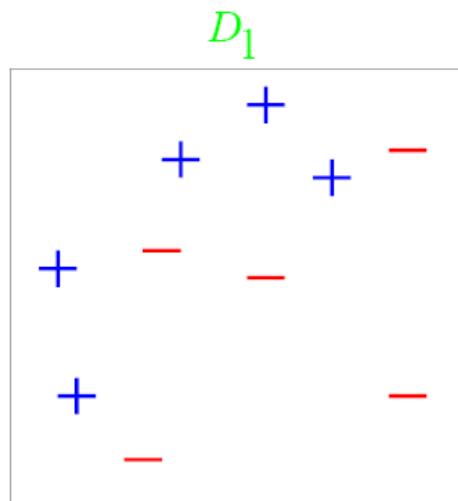
where Z_t = normalization constant

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) > 0 \quad \longrightarrow \text{Why?} \quad \epsilon_t < 0.5$$

Toy Examples

- constructing D_t :

- $D_1(i) = 1/m$

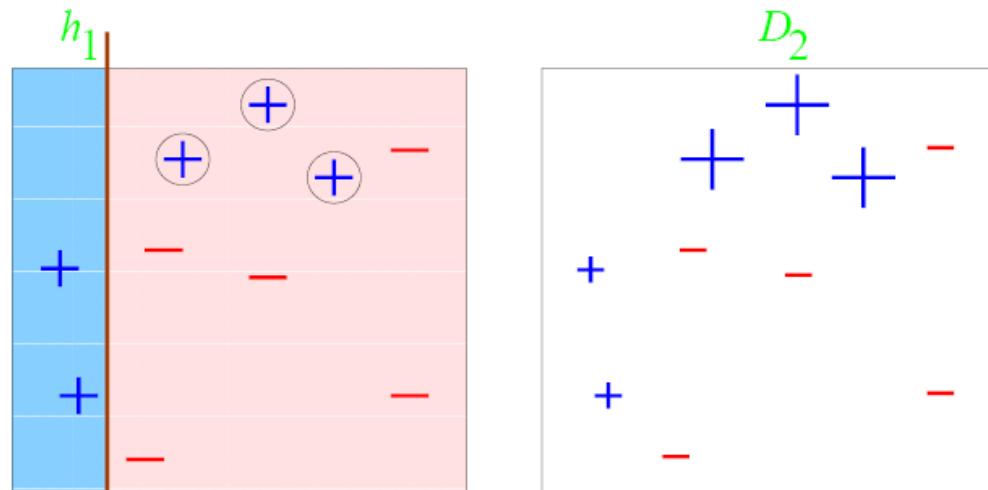


weak classifiers = vertical or horizontal half-planes

Round 1

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases}$$

$$= \frac{D_t(i)}{Z_t} \exp(-\alpha_t y_i h_t(x_i))$$



$$\epsilon_1 = 0.30$$

$$\alpha_1 = 0.42$$

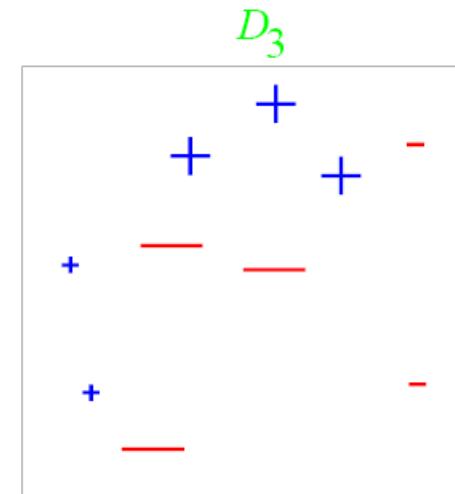
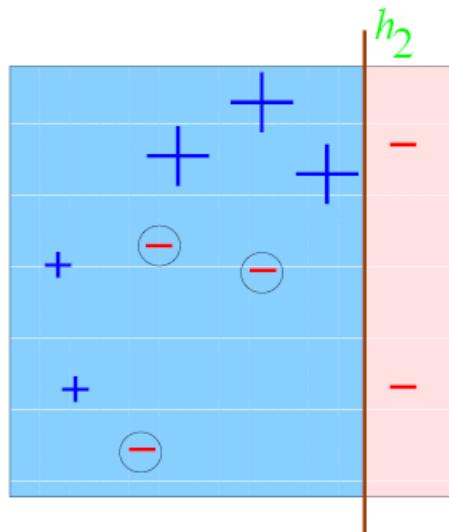
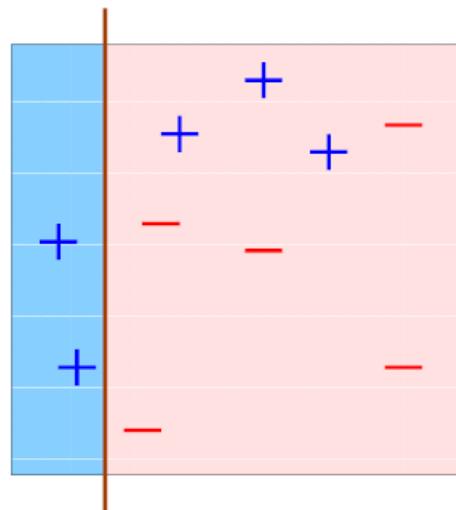
- for $t = 1, \dots, T$:
 - find weak classifier (“rule of thumb”)

$$h_t : X \rightarrow \{-1, +1\}$$

with small error ϵ_t on D_t : $\epsilon_j = \sum_{i=1}^m D_t(i)[y_i \neq h_j(x_i)]$

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) > 0$$

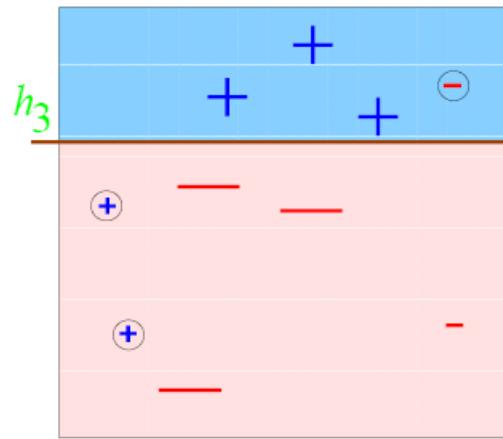
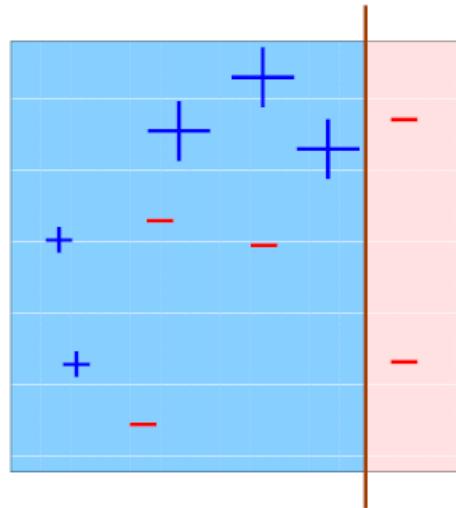
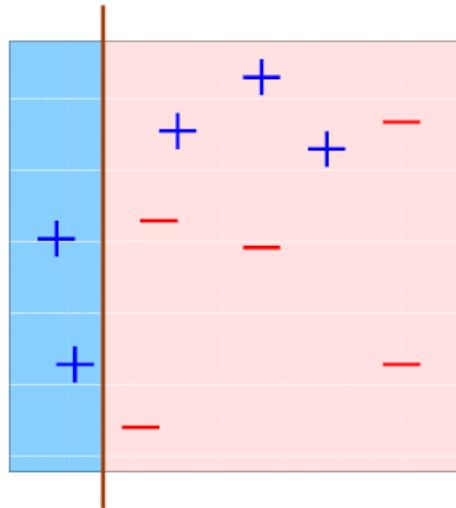
Round 2



$$\varepsilon_2 = 0.21$$

$$\alpha_2 = 0.65$$

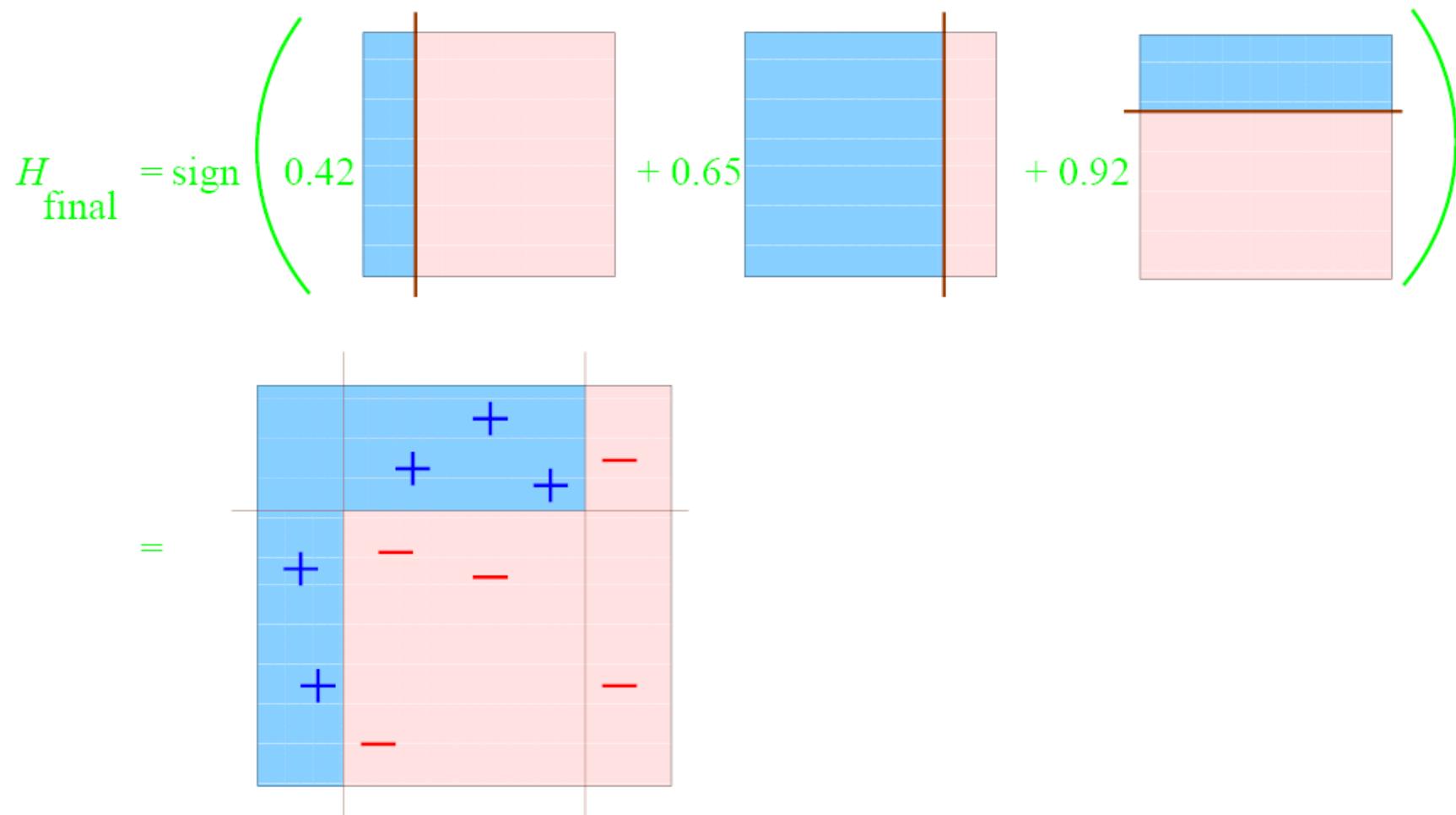
Round 3



$$\varepsilon_3 = 0.14$$

$$\alpha_3 = 0.92$$

Final Classifier



Analysis of training error

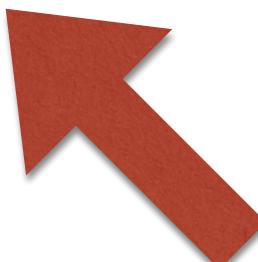
- Theorem:

- write ϵ_t as $1/2 - \gamma_t$
- then

$$\begin{aligned}\text{training error}(H_{\text{final}}) &\leq \prod_t [2\sqrt{\epsilon_t(1-\epsilon_t)}] \\ &= \prod_t \sqrt{1 - 4\gamma_t^2} \\ &\leq \exp\left(-2 \sum_t \gamma_t^2\right)\end{aligned}$$

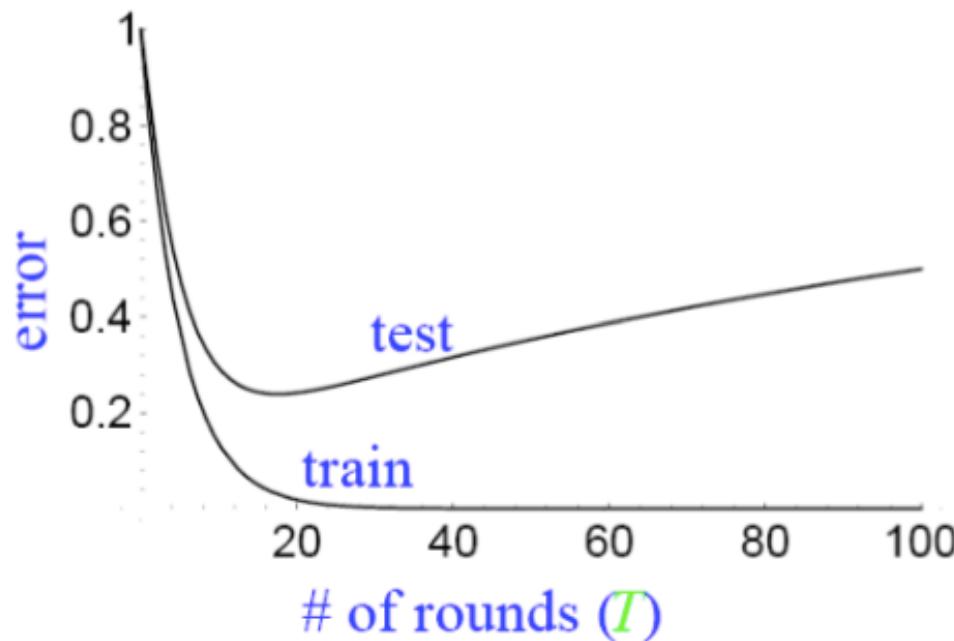
while T increases, the training error
will become smaller and smaller

- so: if $\forall t : \gamma_t \geq \gamma > 0$
then $\text{training error}(H_{\text{final}}) \leq e^{-2\gamma^2 T} < 0$



$\gamma > 0$

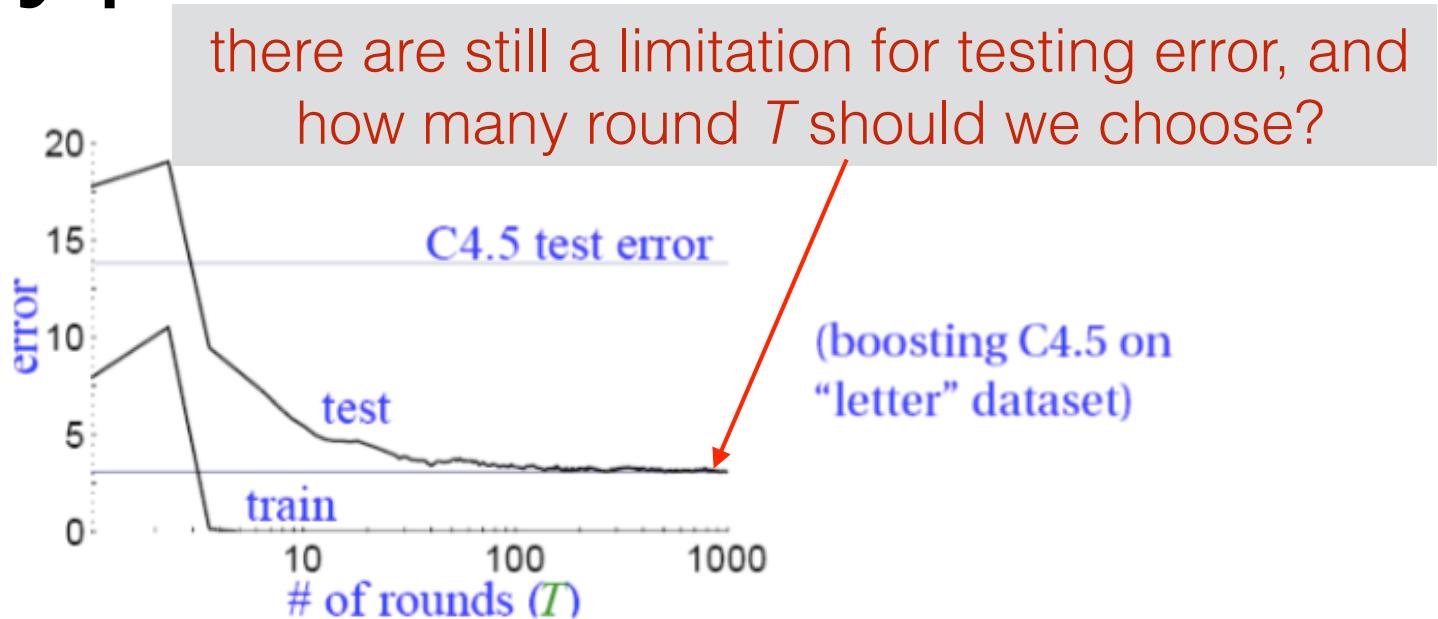
How will test error behave? (A first guess)



expect:

- training error to continue to drop (or reach zero)
- test error to increase when H_{final} becomes “too complex”
 - “Occam’s razor”
 - overfitting
 - hard to know when to stop training

Actual Typical Run



- test error does not increase, even after 1000 rounds
 - (total size > 2,000,000 nodes)
- test error continues to drop even after training error is zero!

	# rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1

$$H_T(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right)$$

- Larger $T \rightarrow$ More computation time
 - The test error will not decrease while T is large enough
- Stop the loop!
- when the training error decrease to a preset value, ex. 5%
 - when the training error decrease slowly
 - ...

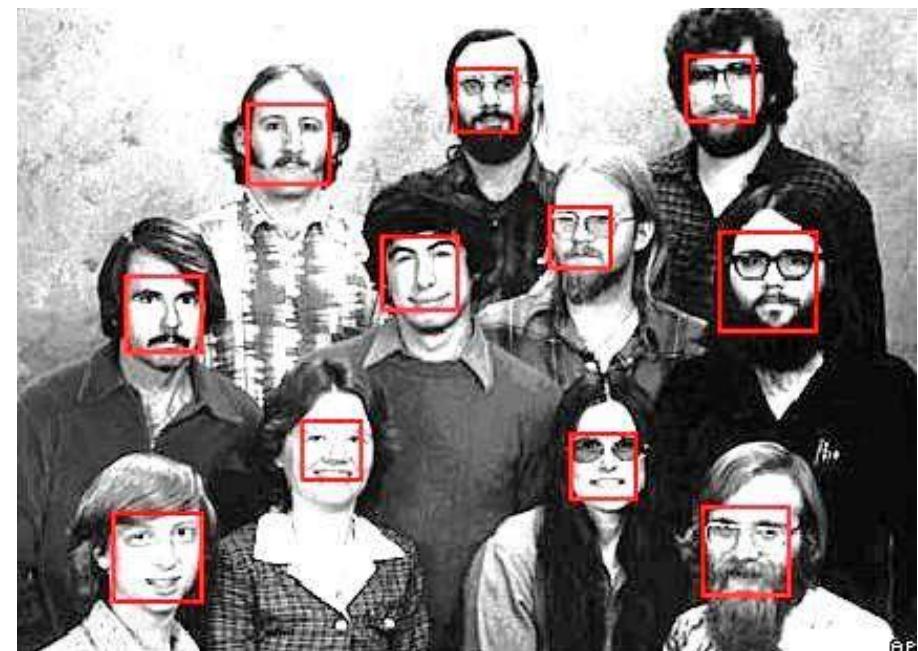
Face Detection

P. Viola and M. Jones, “Rapid Object Detection using a Boosted Cascade of Simple Features,” *CVPR*, 2001.
(cited number: [28308](#) from Google)

P. Viola and M. Jones, “Robust Real-Time Face Detection,”
International Journal of Computer Vision, 2004.
(cited number: [23738](#) from Google)

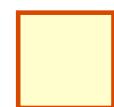
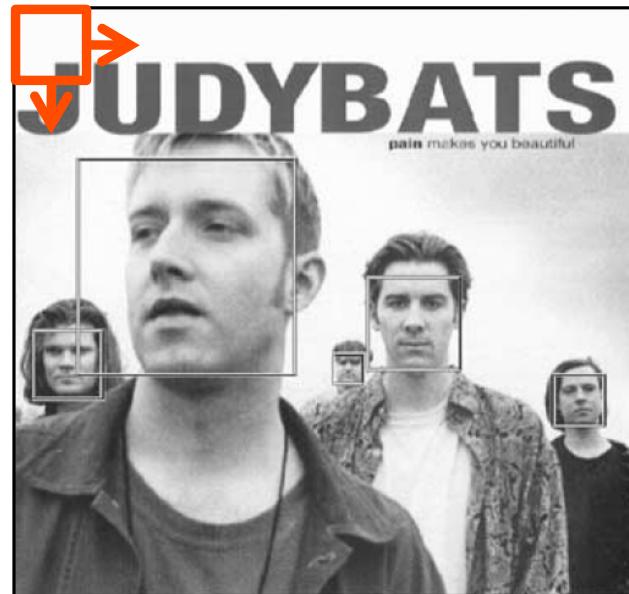
Face Detection

- Detect and localize human faces in images
- Two issues are important
 - Accuracy
 - Efficiency

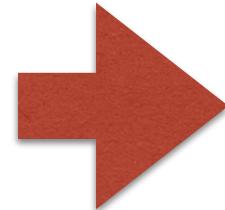


Basic Idea

- Slide a window across image and evaluate a face model at every location.

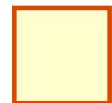


: Location & Scale

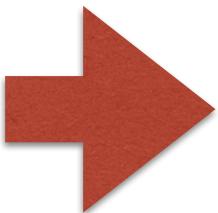


How many windows in
one image?

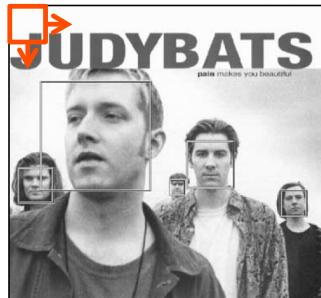
Basic Idea



: Location & Scale



How many windows in
one image?



500 x 500 image

shift window for every 2 pixels: **250 x 250 windows**

window size from 20 x 20, 22 x 22, ..., to 200 x 200:

90 scales

total windows number is: **250 x 250 x 90 = about 5×10^6**

Challenges

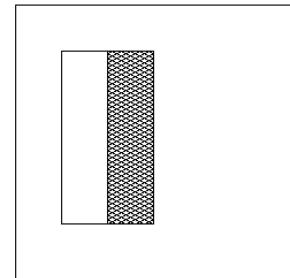
- Slide a window across image and evaluate a face model at every location.
- Sliding window detector must evaluate tens of thousands of location/scale combinations.
- Faces are rare: 0–10 per image
 - For computational efficiency, we should try to spend as little time as possible on the non-face windows
 - A megapixel image has $\sim 10^6$ pixels and a comparable number of candidate face locations
 - To avoid having a false positive in every image, our false positive rate has to be less than 10^{-6}

Viola & Jones Face Detector

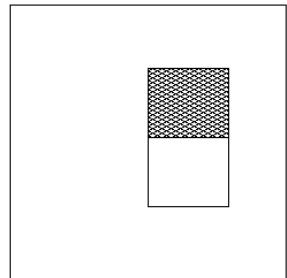
- A seminal approach to **real-time** object detection
- Training is **slow**, but detection is very **fast**
- Key contribution
 - **Integral images** for fast feature evaluation
 - **AdaBoost algorithm** for feature selection and classification
 - **Cascade approach** for fast rejection of non-face windows

Haar Feature

- or **Haar-like features** owe their name to their intuitive similarity with Haar wavelets
- Four types of rectangular features:
 - *two-rectangle feature type* (horizontal/vertical)
 - three-rectangle feature type
 - four-rectangle feature type

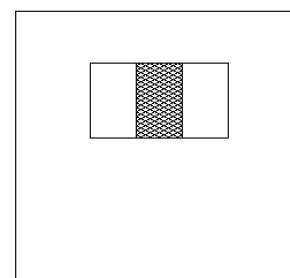


A

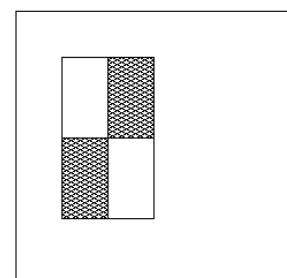


B

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



C



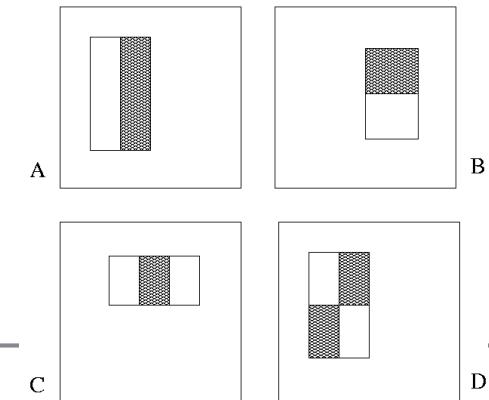
D



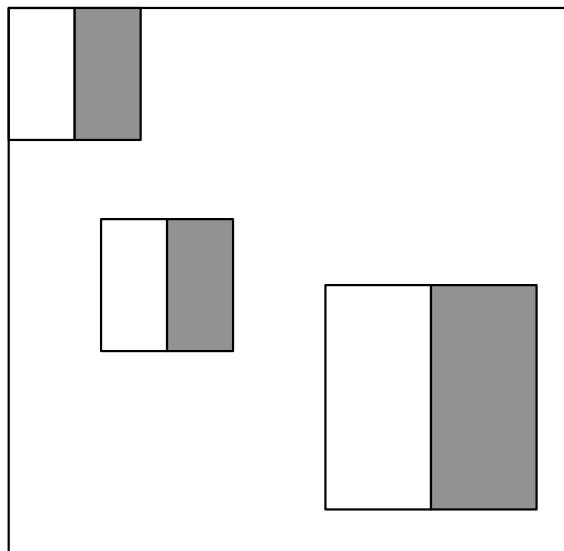
Capture the face symmetry

Haar Feature

- or **Haar-like features** owe their name to their intuitive similarity with Haar wavelets
- Four types of rectangular features:
 - *two-rectangle feature* type (horizontal/vertical)
 - three-rectangle feature type
 - four-rectangle feature type



Type A



A 24x24 detection window

Can be extracted at **any location** with **any scale!**

Haar Feature

why 24x24?

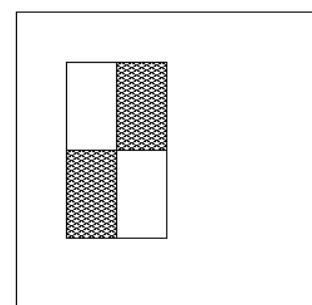
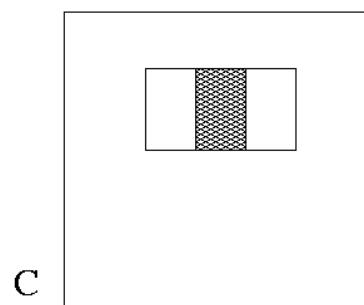
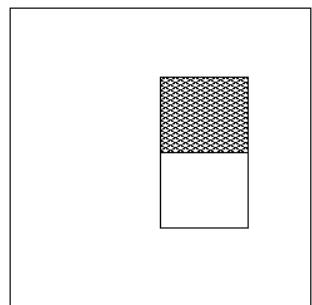
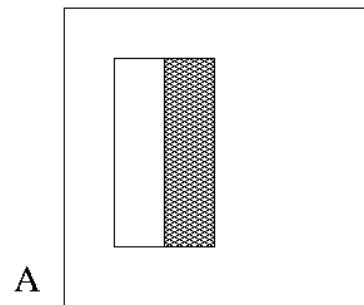
- How many number of possible rectangle features for a 24x24 detection region?

The training data is 24x24



Haar Feature

- How many number of possible rectangle features for a 24×24 detection region?



$$A+B \quad 2 \sum_{w=2}^{12} \sum_{h=1}^{24} (24 - 2w + 1)(24 - h + 1) +$$

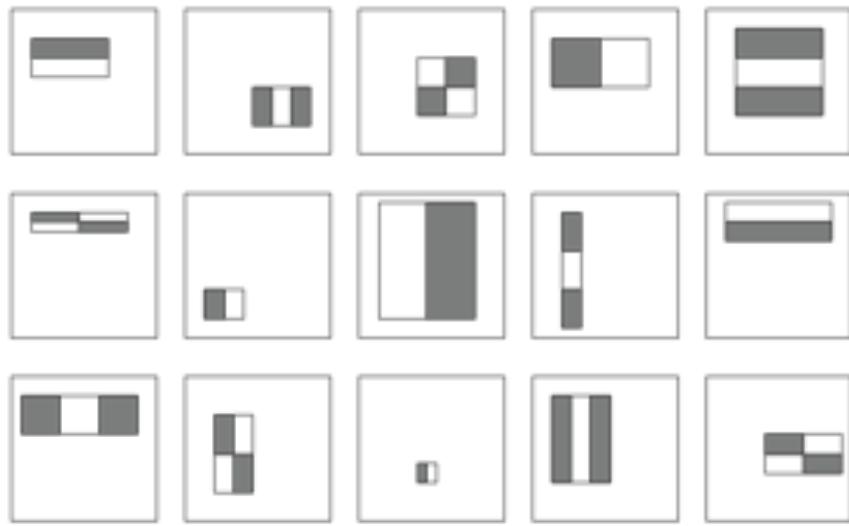
$$C \quad 2 \sum_{w=3}^{8} \sum_{h=1}^{24} (24 - 3w + 1)(24 - h + 1) +$$

$$D \quad \sum_{w=2}^{12} \sum_{h=2}^{12} (24 - 2w + 1)(24 - 2h + 1)$$

: 160,000

Haar Feature

- How many number of possible rectangle features for a 24×24 detection region?



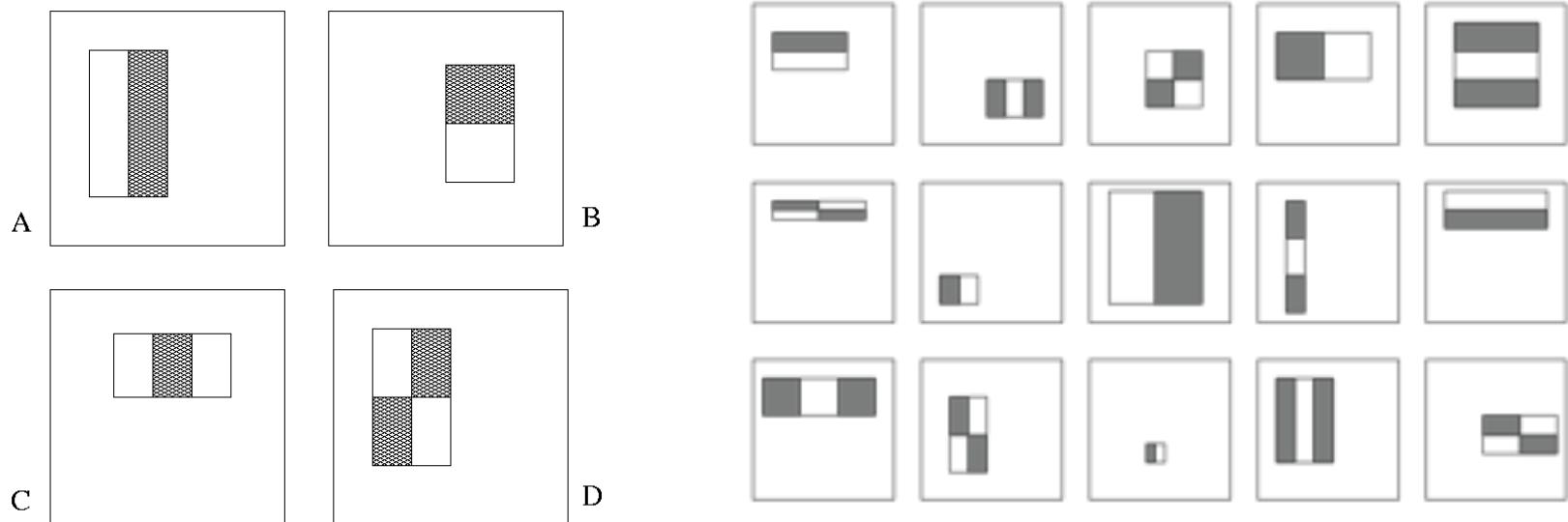
$$\begin{aligned} A+B & 2 \sum_{w=2}^{12} \sum_{h=1}^{24} (24 - 2w + 1)(24 - h + 1) + \\ C & 2 \sum_{w=3}^8 \sum_{h=1}^{24} (24 - 3w + 1)(24 - h + 1) + \\ D & \sum_{w=2}^{12} \sum_{h=2}^{12} (24 - 2w + 1)(24 - 2h + 1) \end{aligned}$$

- What features are good for face detection?
- Can we create a good classifier using just **a small subset** of all possible features?
- **How** to select such a subset?

: 160,000

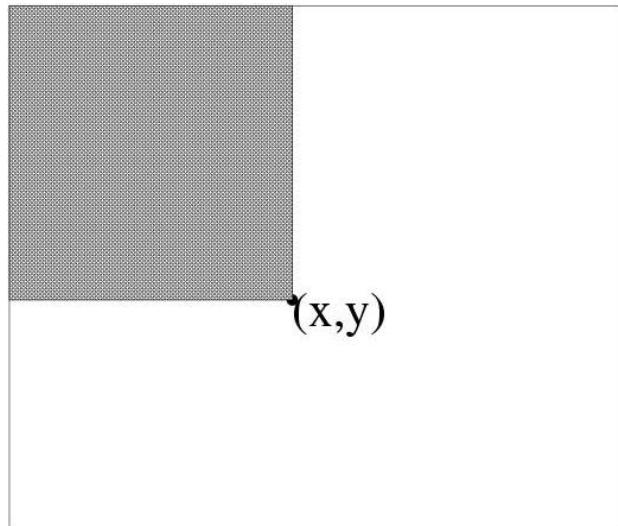
AdaBoost

Haar Feature



The motivation behind using rectangular features, as opposed to more expressive steerable filters is due to their **extreme computational efficiency**

Integral Image



Definition: The *integral image* at location (x, y) , is the sum of the pixel values above and to the left of (x, y) , inclusive.

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

Example:

2	1	2	3	4	3
3	2	1	2	2	3
4	2	1	1	1	2

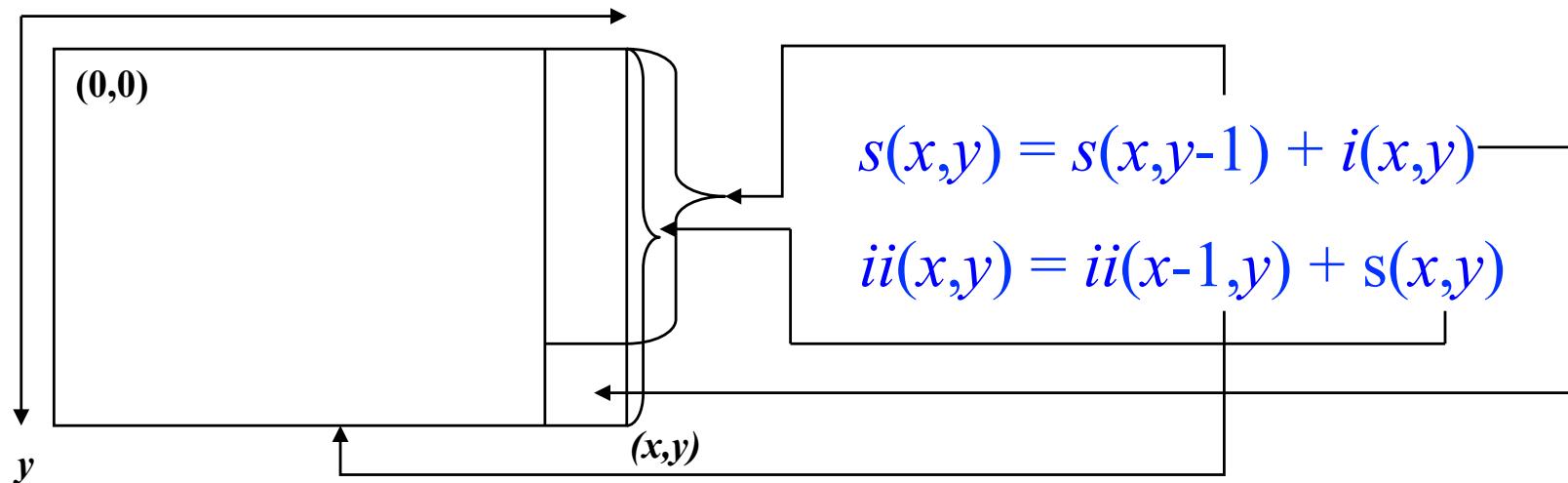
Image

2	3	5	8	12	15
5	8	11	16	22	28
9	14	18	24	31	39

Integral image

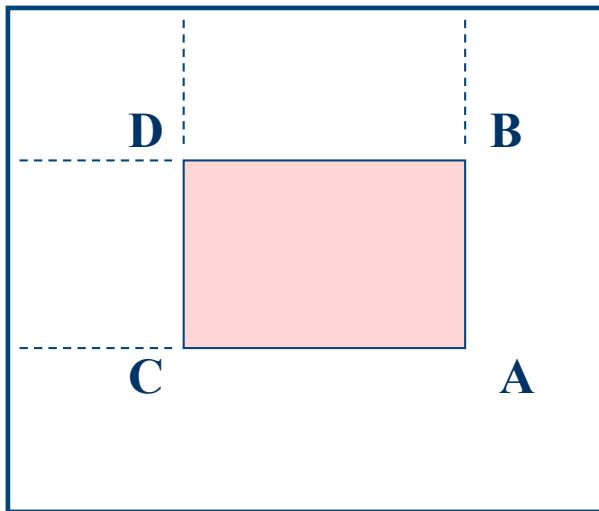
Integral Image

Using the following two recurrences, where $i(x,y)$ is the pixel value of original image at the given location and $s(x,y)$ is the **cumulative column sum**, we can calculate the integral image representation of the image in a single pass.



Rapid evaluation of rectangular features

Using the integral image representation one can compute the value of any rectangular sum in constant time.



$$\text{sum} = ii_A - ii_B - ii_C + ii_D$$

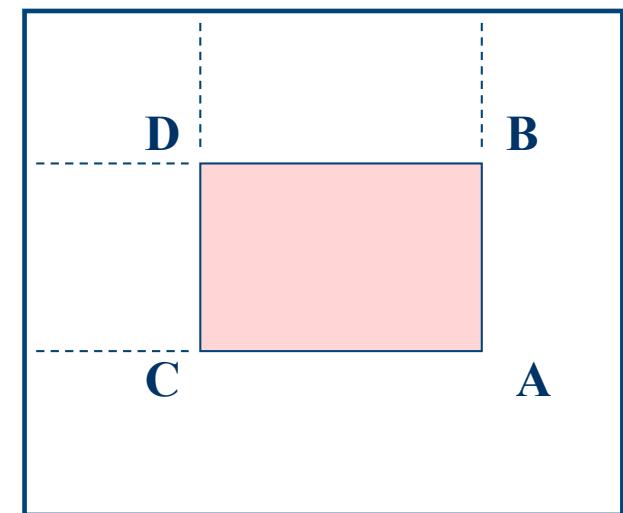
Only 3 additions are required
for any size of rectangle!

As a result two-, three-, and four-rectangular features can be computed with 6, 8 and 9 array references, respectively.

Scaling

Using the integral image representation one can compute the value of any rectangular sum in constant time.

- Integral image enables us to evaluate all rectangle sizes in constant time.
- Therefore, no image scaling is necessary.
- Scale the rectangular features instead!



$$\text{sum} = ii_A - ii_B - ii_C + ii_D$$

Weak Classifiers for Face Detection

$$h_t(x) = \begin{cases} 1 & \text{if } f_t(x) > \theta_t \\ 0 & \text{otherwise} \end{cases}$$

value of rectangle feature

window

threshold

The diagram illustrates the inputs to a weak classifier $h_t(x)$. A vertical arrow labeled "window" points to the input x . Two diagonal arrows point to the features $f_t(x)$ and the threshold θ_t . The text "value of rectangle feature" is positioned above the condition in the if-else statement, and "threshold" is positioned to the right of the threshold variable.

AdaBoost for face detection

- Given example images $(x_1, y_1), \dots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = 1/(2m), 1/(2l)$ for training example i , where m and l are the number of negatives and positives respectively.

For $t = 1 \dots T$

- 1) Normalize weights so that w_t is a distribution
- 2) For each feature j train a classifier h_j and evaluate its error ε_j with respect to w_t
- 3) Choose the classifier h_j with lowest error.
- 4) Update weights according to:

$$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$

where $e_i = 0$ if x_i is classified correctly, 1 otherwise, and

$$\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t} < 1$$

- The final strong classifier is:

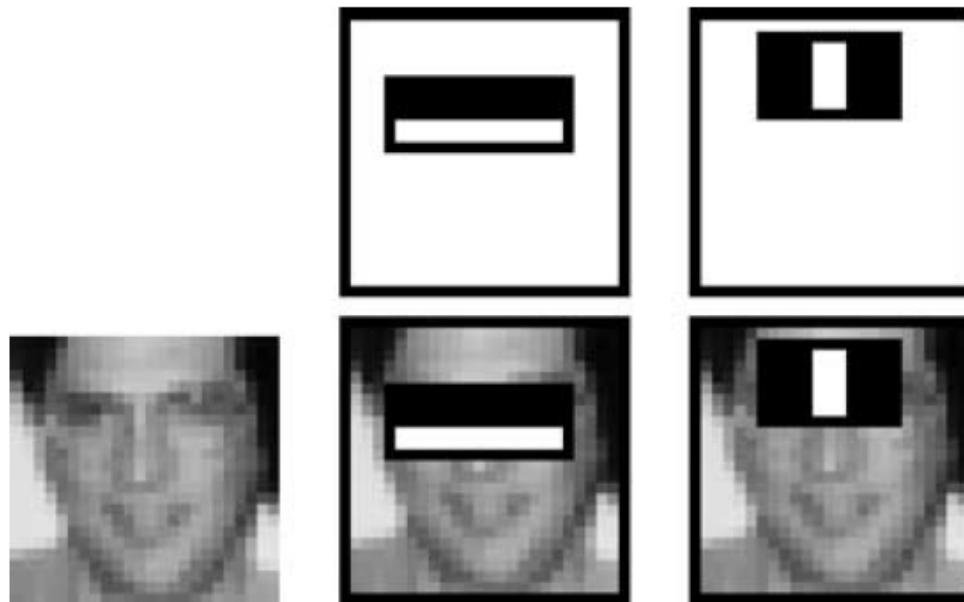
$$h(x) = \begin{cases} 1 & \sum_{t=1}^T \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^T \alpha_t, \\ 0 & \text{otherwise} \end{cases} \quad \text{where} \quad \alpha_t = \log\left(\frac{1}{\beta_t}\right)$$

AdaBoost for face detection

- Training set contains **face** and **non-face** examples
 - Initially, with equal weight
- For each round of boosting:
 - Evaluate each rectangle filter on each example
 - Select best threshold for each filter
 - Select best filter/threshold combination
 - Reweight examples

Features selected

- First two features selected by boosting:



- This feature combination can yield 100% detection rate and 50% false positive rate

AdaBoost for face detection

- Computational complexity of **learning**: $O(TNK)$
 - T rounds, N examples, K features

200 **10,000+** **160,000**
3,000,000+

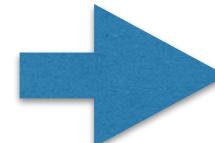
How about **testing**?

$O(TW)$

T rounds (selected features), W window number

200 **10^6+**

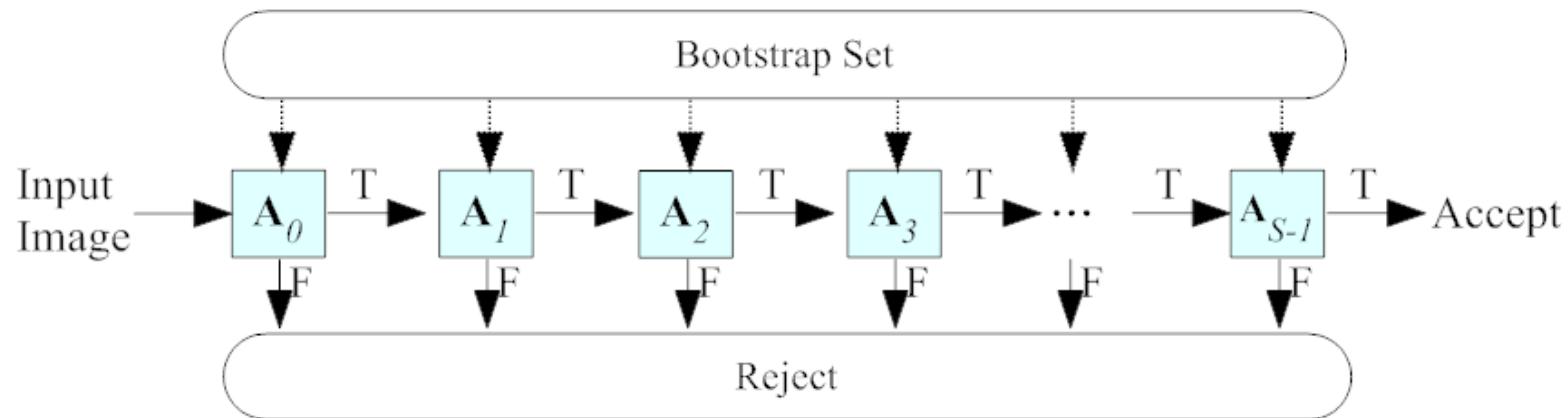
- Faces are rare: 0–10 per image
- Number of windows: negative >> positive
 - Same decision time is paid for accepting or rejecting a window.



Attentional Cascade

Attentional Cascade

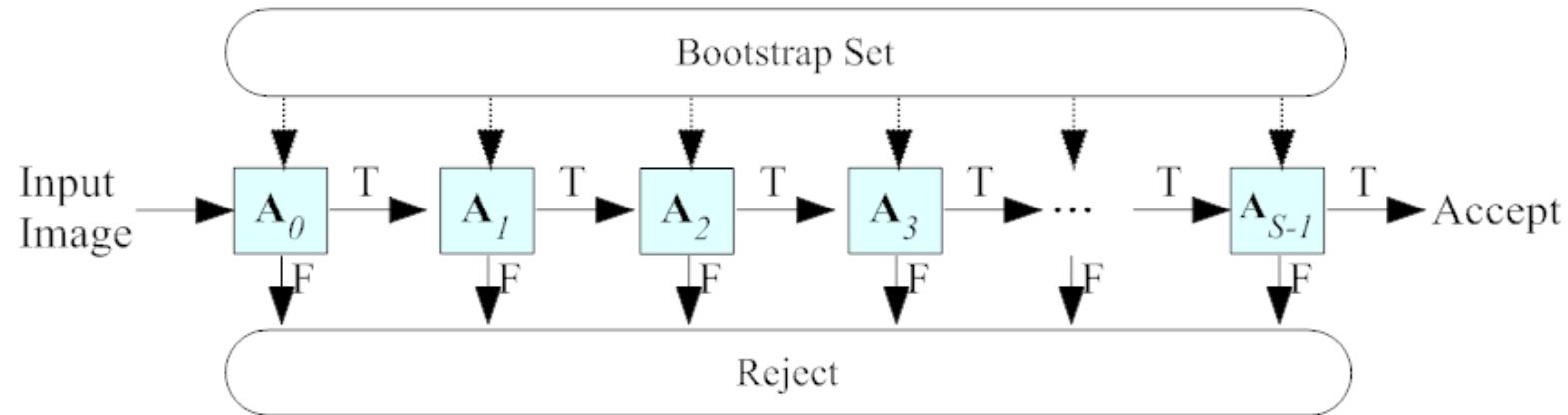
- Basic cascaded structure (S stages)



- Positive response from the first classifier triggers the evaluation of a second (more complex) classifier, and so on.
- A negative outcome at any point leads to the immediate rejection of the sub-window.

Attentional Cascade

- Basic cascaded structure (S stages)



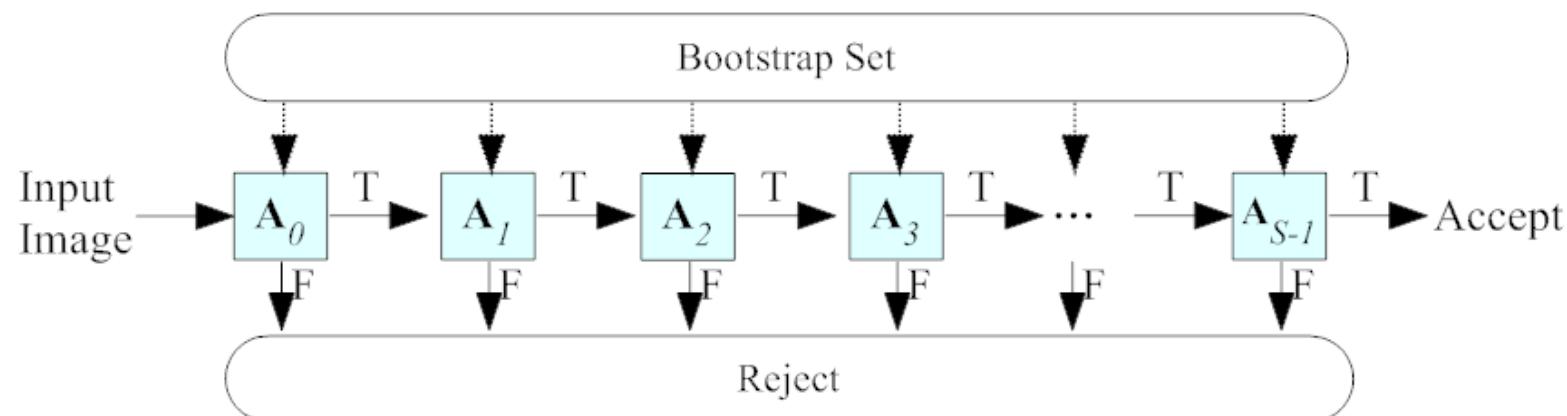
- Each stage has a **very high detection rate** (e.g. 99.95%) and a loose false positive rate (e.g. 50~90%).

Layer number	1	2	3 to 5	6 and 7	8 to 12	13 to 32
Number of features	2	5	20	50	100	200
Detection rate	100%	100%	-	-	-	-
Rejection rate	60%	80%	-	-	-	-

more than 60% windows can be eliminated by using only 2 features

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 AdaBoost 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



Experiments - Dataset for training

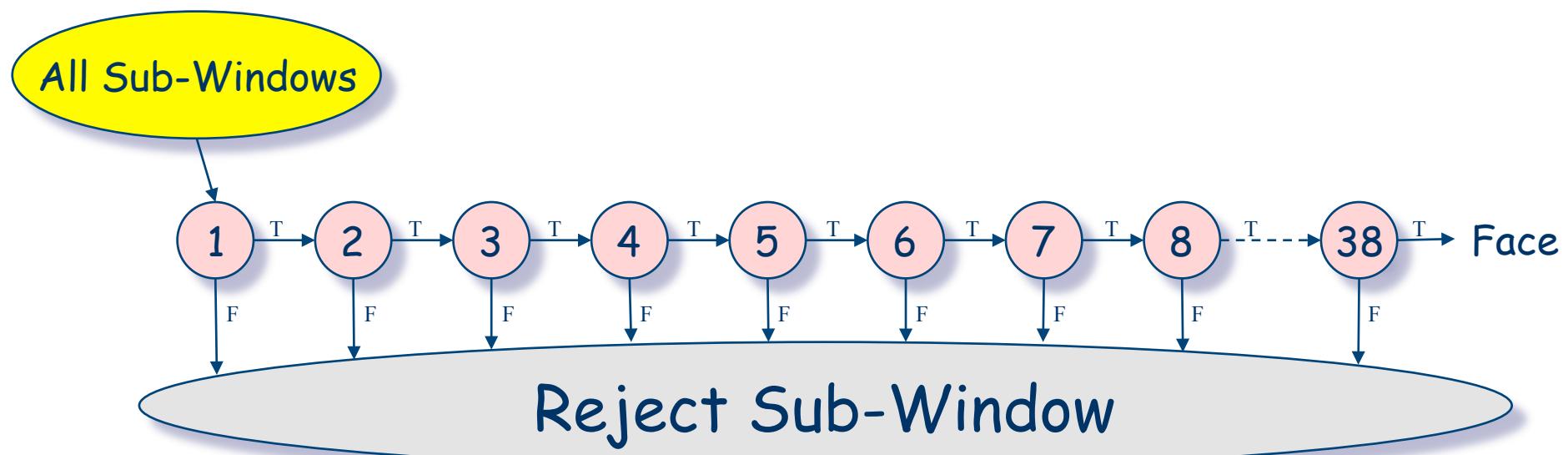
- 4916 positive training example were hand picked **aligned**, **normalized**, and **scaled** to a base resolution of 24x24
- 3,000,000+ negative examples were selected by **randomly picking sub-windows** from 9500 images which did not contain faces



Experiments

- structure of the detector cascade

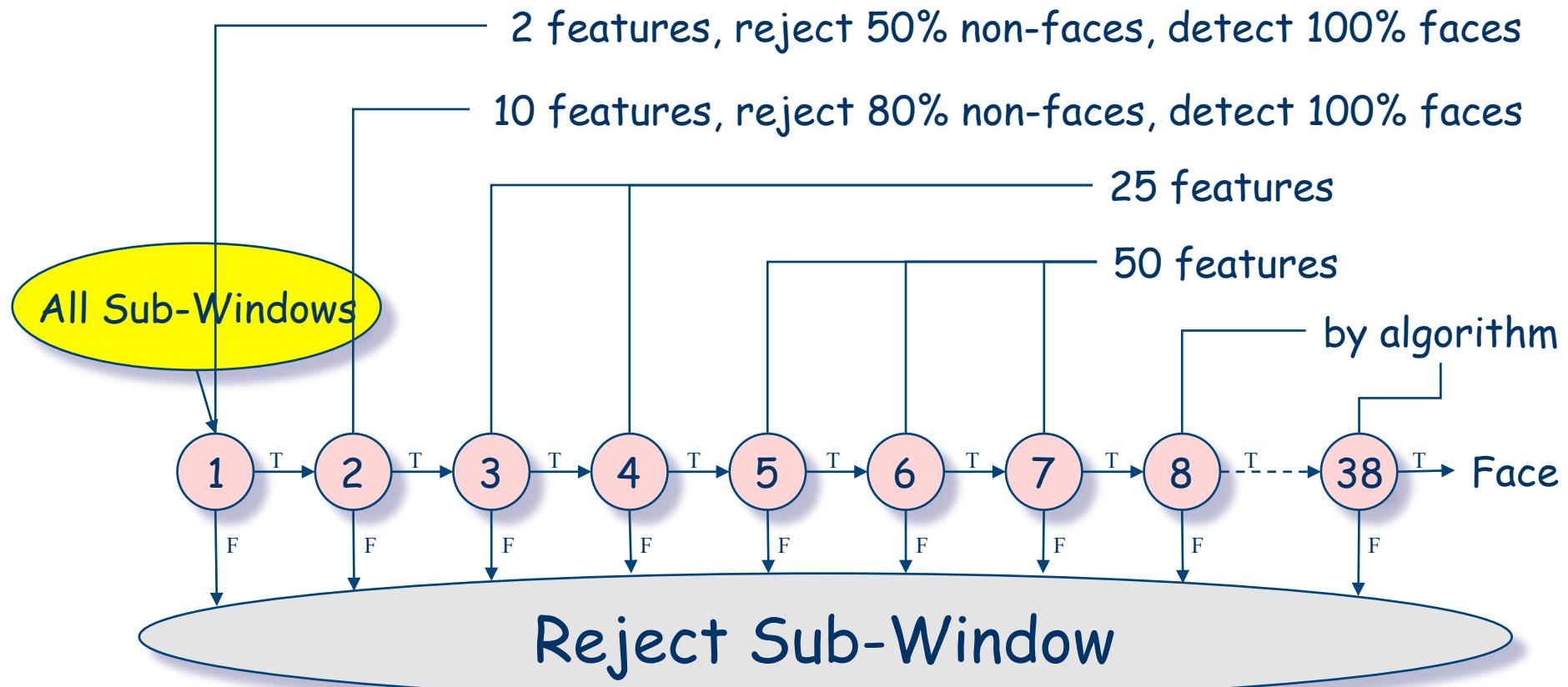
- The final detector had **38** layers and **6061** features in total



Experiments

- structure of the detector cascade

- The final detector had **38** layers and **6061** features in total



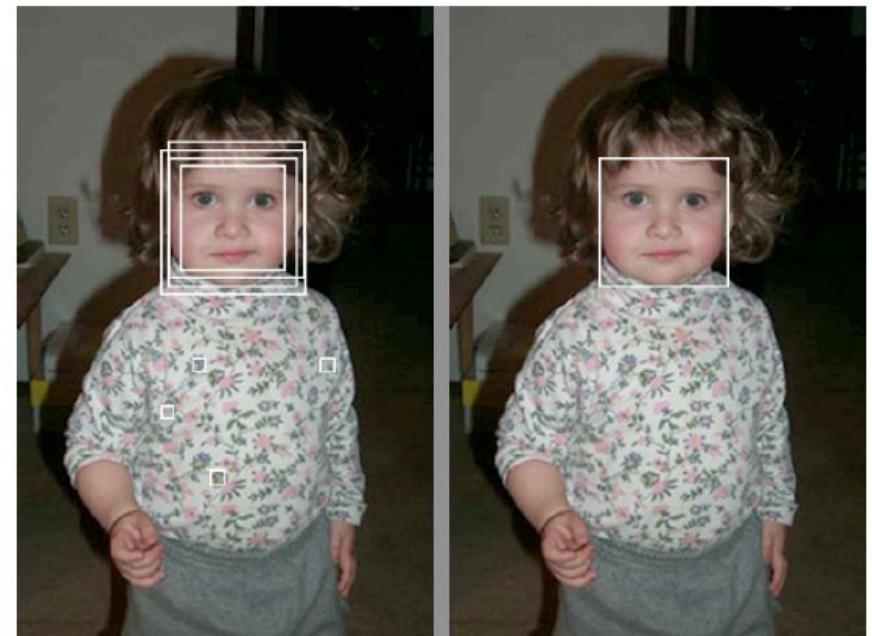
Experiments

- Speed of the Final Detector

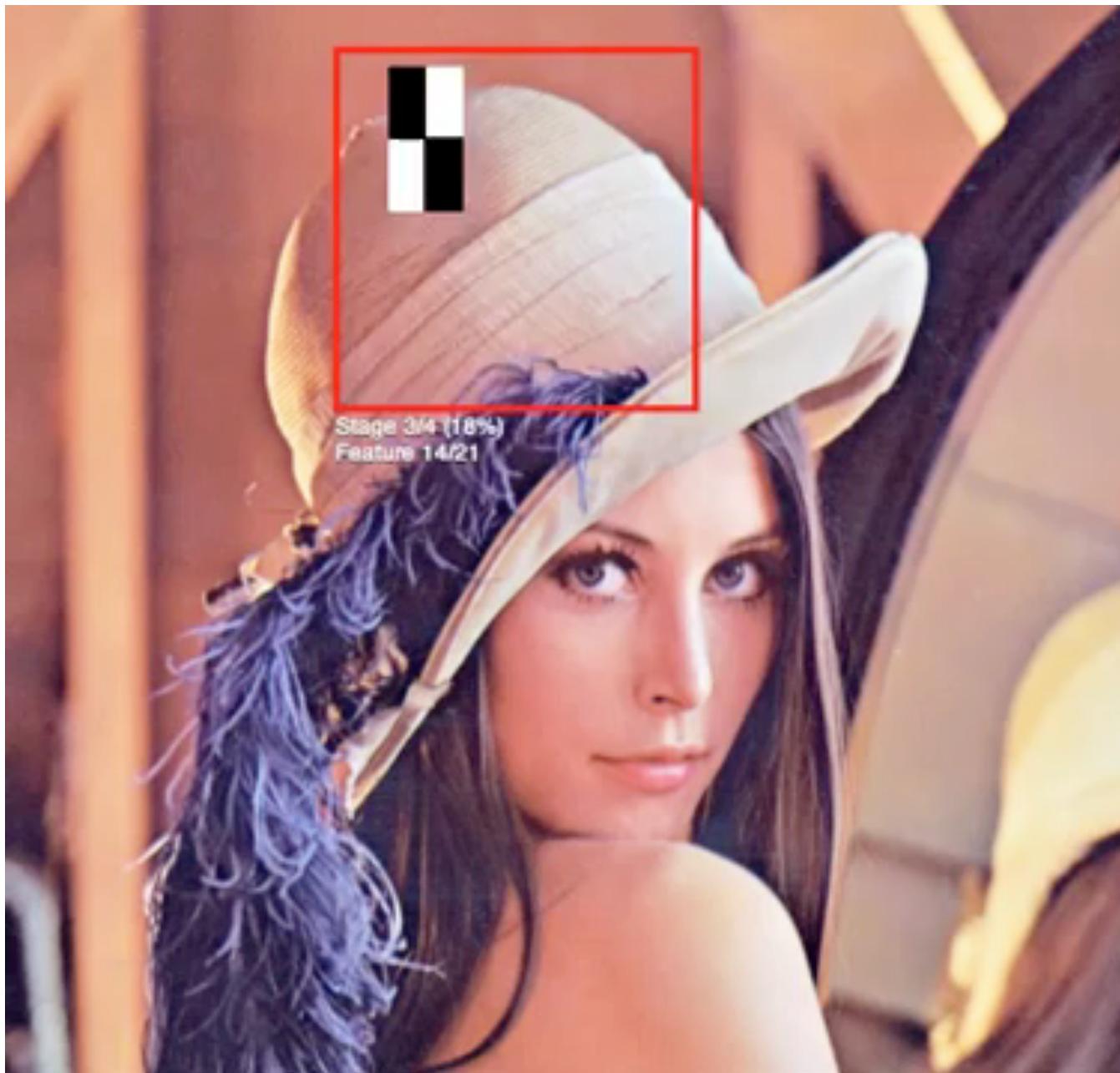
- Speed of the detector ~ total number of features evaluated
- On the MIT-CMU test set the average number of features evaluated is **8** (out of 6061).
- The processing time of a **384 by 288** pixel image on a conventional personal computer (700 MHz Intel P-III) about **0.067 seconds (15 fps)**.
- Processing time should linearly scale with image size, hence processing of a 3.1 mega pixel images taken from a digital camera should approximately take 2 seconds.

Operation of the face detector

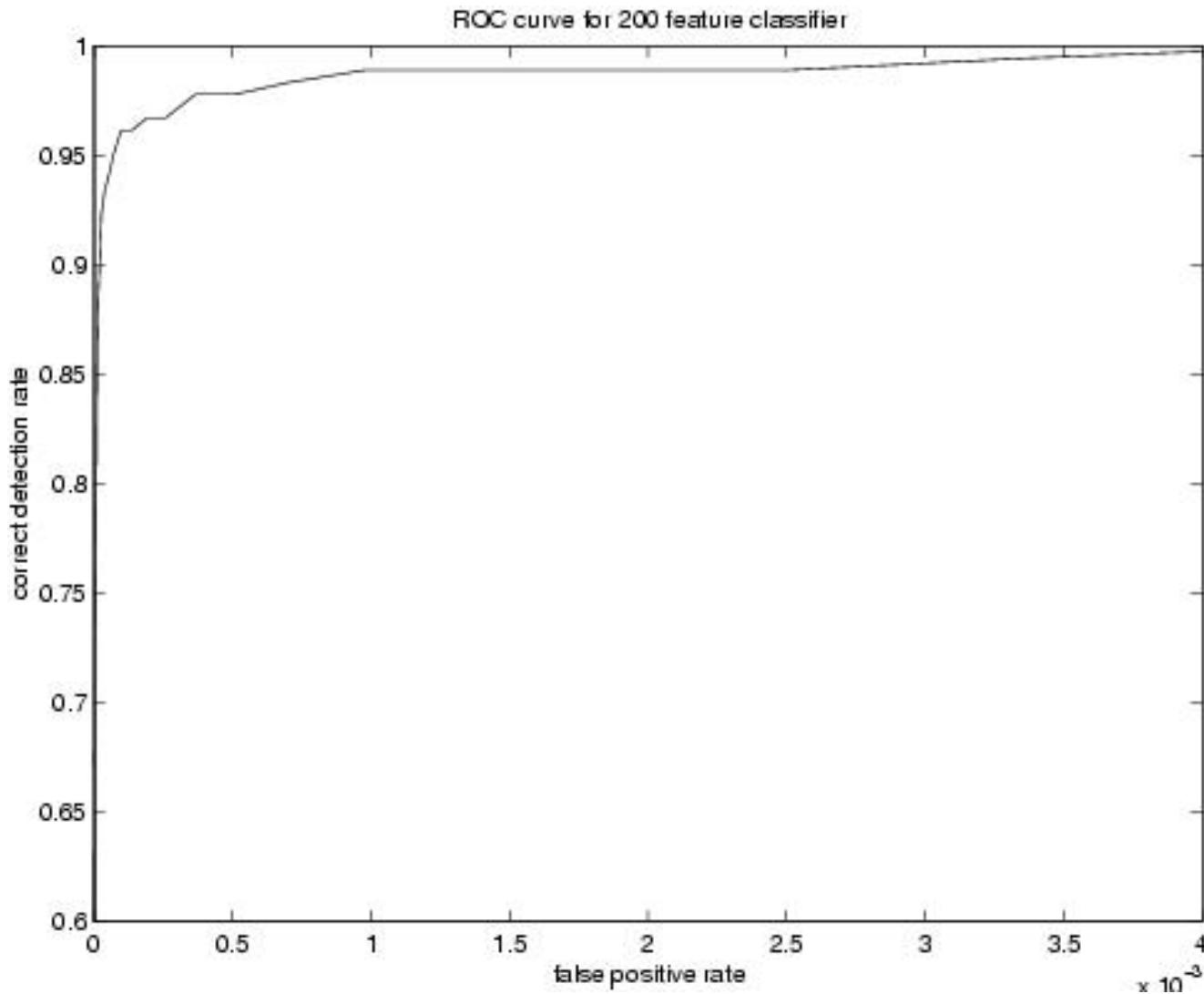
- Since training examples were normalized, image sub-windows needed to be normalized also. This **normalization** of images can be efficiently done using two integral images (regular / squared).
- **Detection at multiple scales** is achieved by scaling the image
- The amount of **shift** between subsequent sub-windows is determined by some constant number of pixels and the current scale.
- **Multiple detections** of a face, due to the insensitivity to small changes in the image of the final detector were, were combined based on overlapping bounding region.



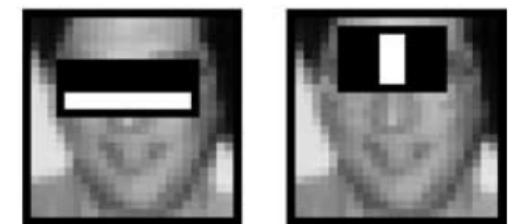
Haar Cascade Visualization



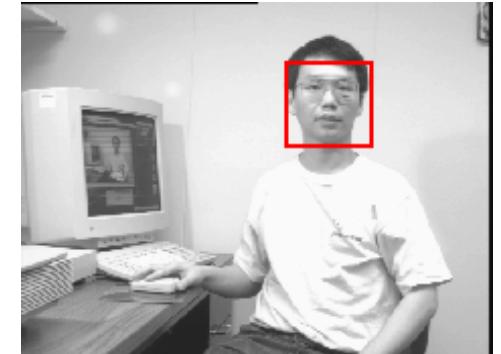
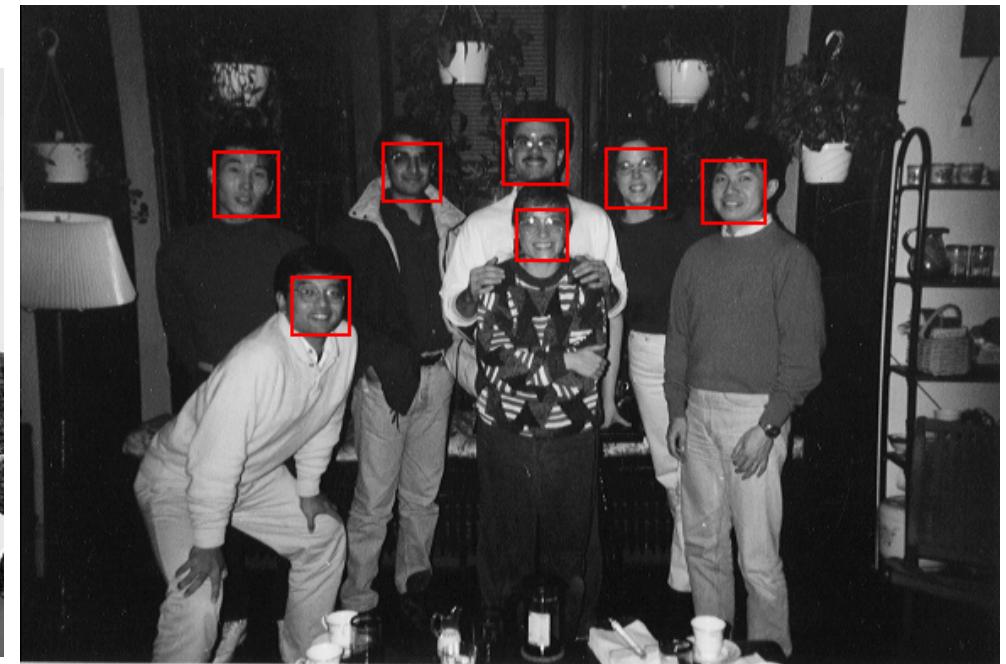
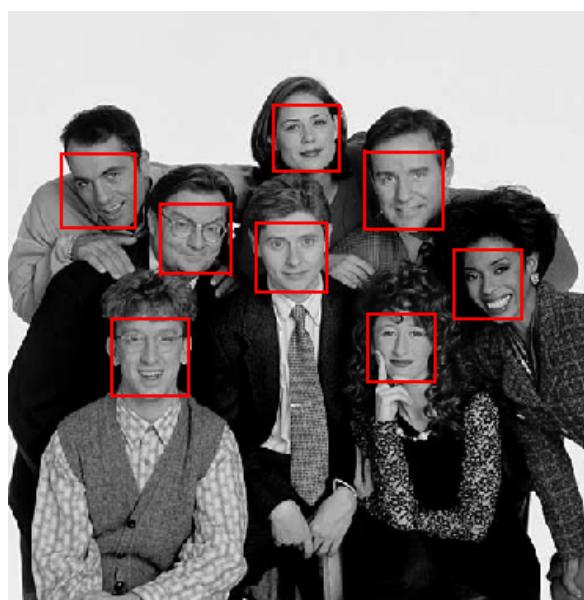
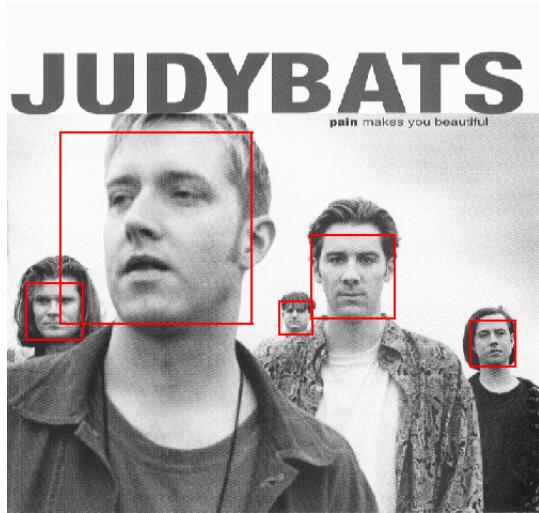
Performance



First two features
selected

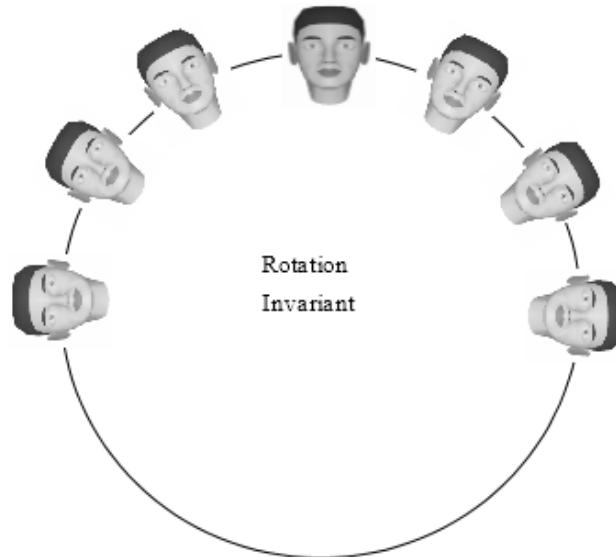


Results

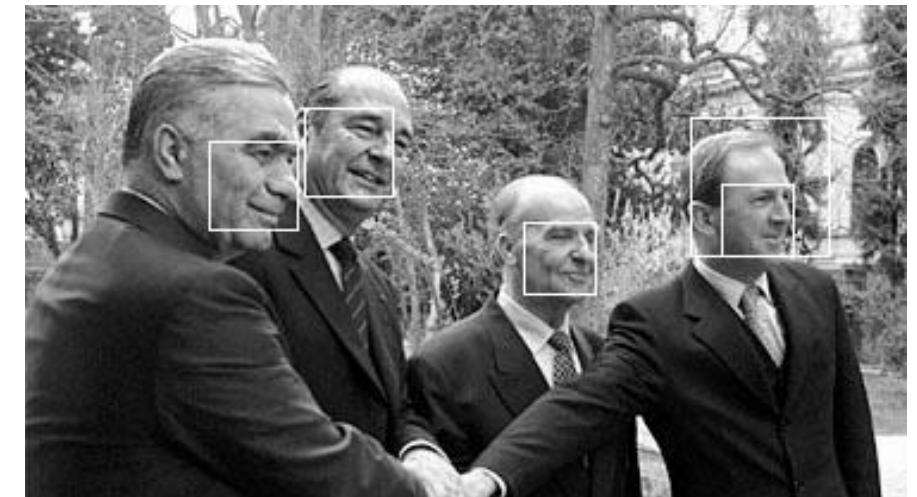
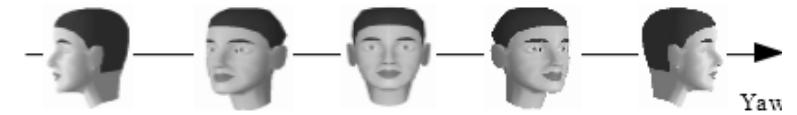


Rotation of face

In-Plane Rotation

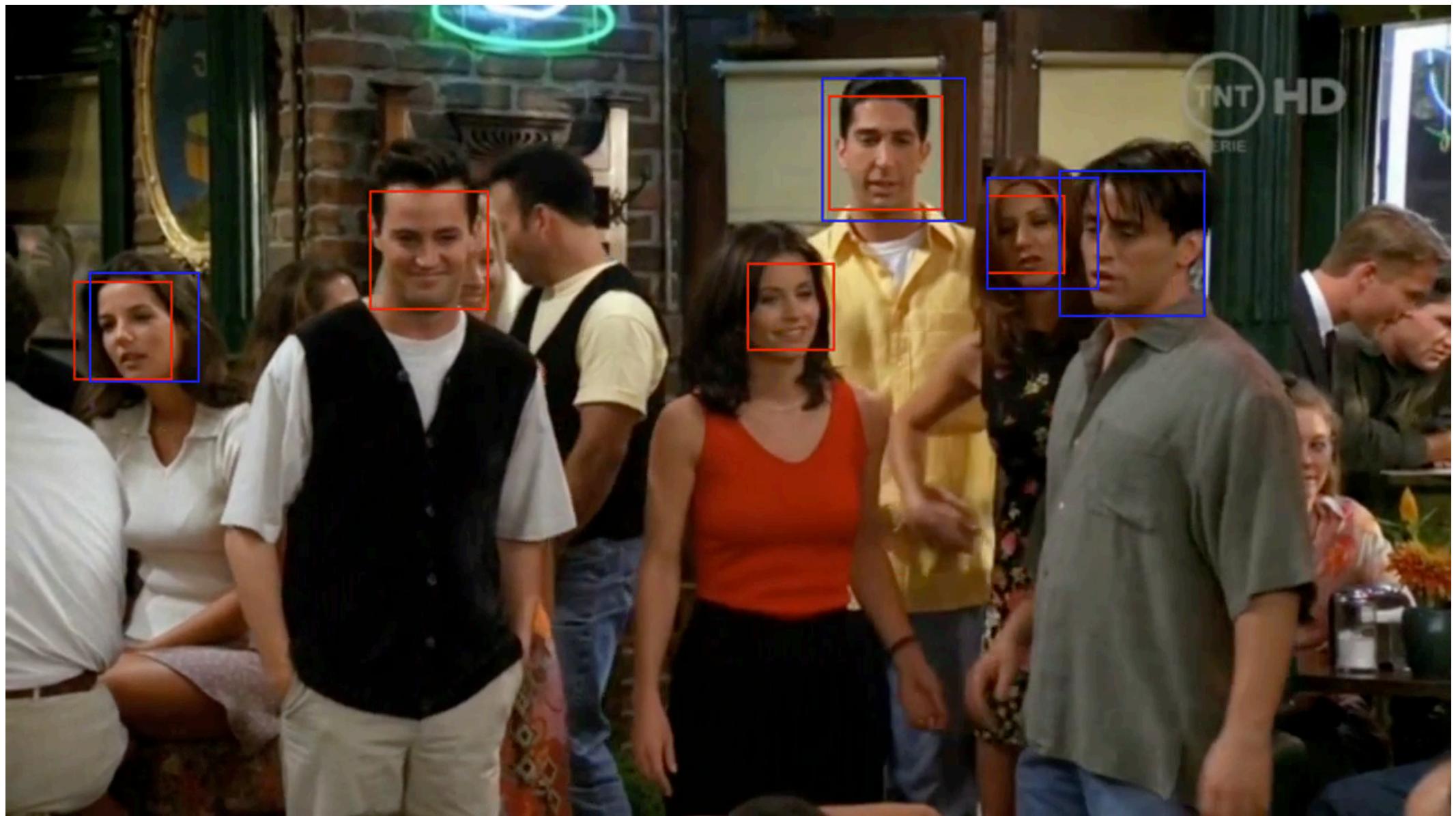


Out-Plane Rotation

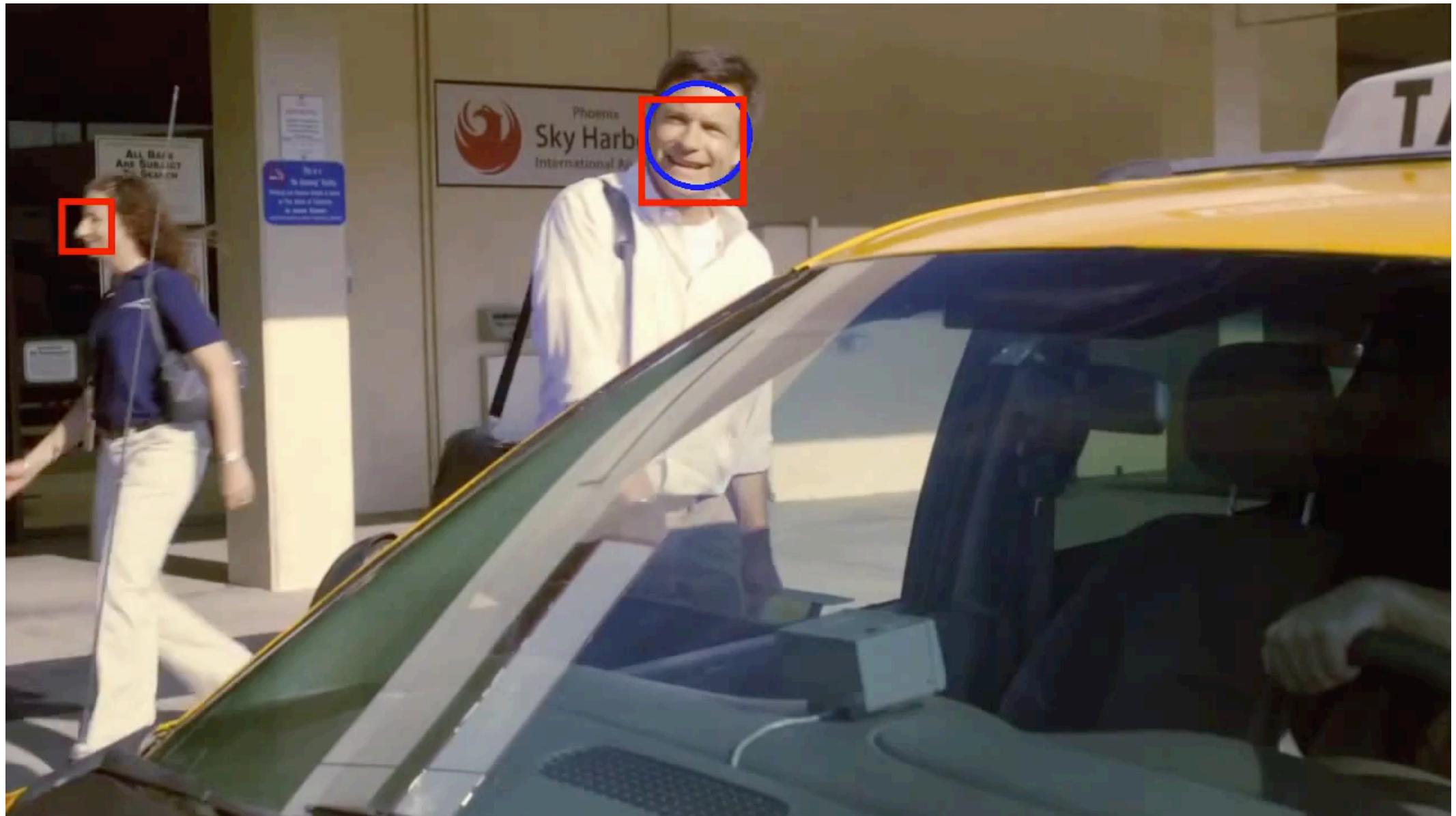


Profile face detection

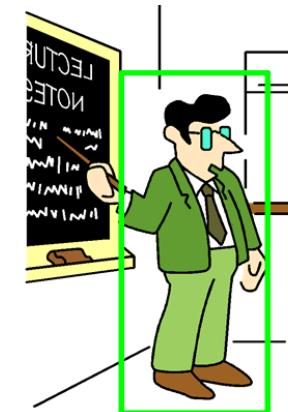
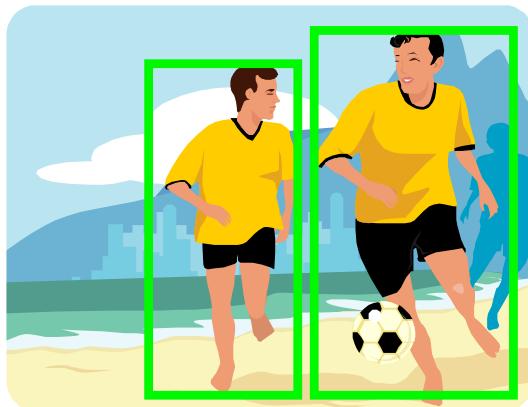
Frontal and Profile Face Detection - OpenCV



Dlib vs OpenCV face detection



Pedestrian Detection



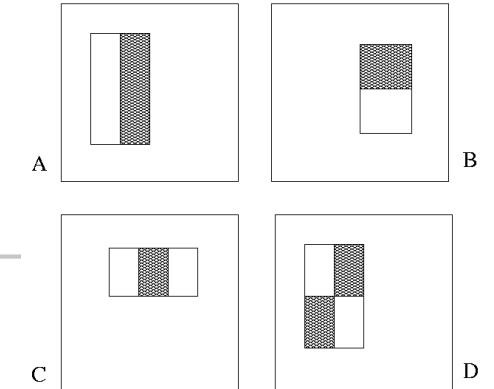
Pedestrian Detection

- Detect and localize human/pedestrian in images
- Similar to face detection, but with more **deformation** and **occlusion** challenges



Pedestrian Detection

- Detect and localize human/pedestrian in images

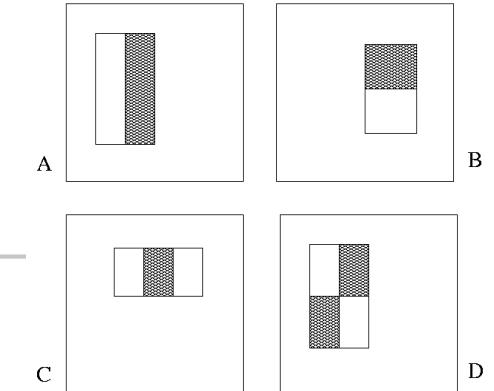


Are Haar features suitable for pedestrian detection?

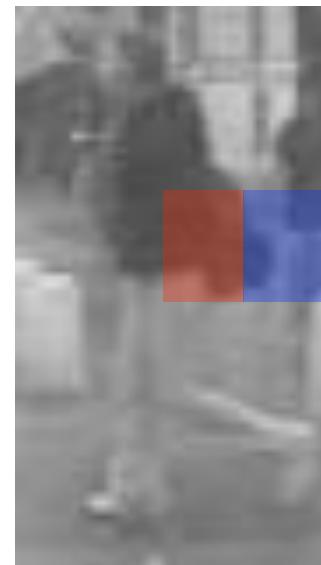
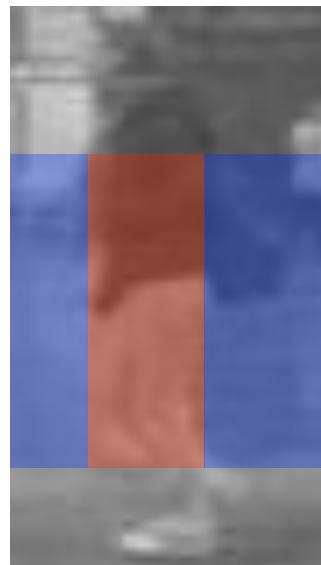


Pedestrian Detection

- Detect and localize human/pedestrian in images



Are Haar features suitable for pedestrian detection?



Pedestrian Detection

- **Challenges:**
 - Wide variety of articulated poses
 - Variable appearance/clothing
 - Complex backgrounds
 - Unconstrained illumination
 - Occlusions
 - Different Scales

Are any other features
more suitable for
pedestrian detection?

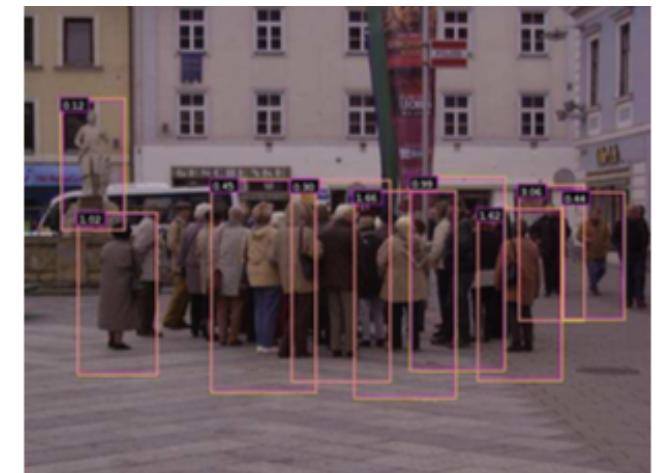
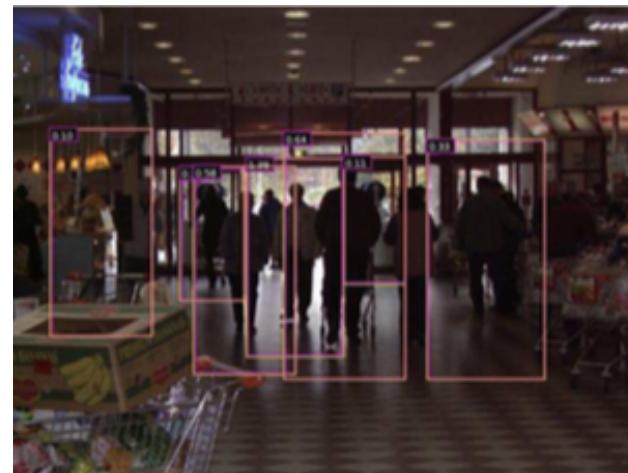
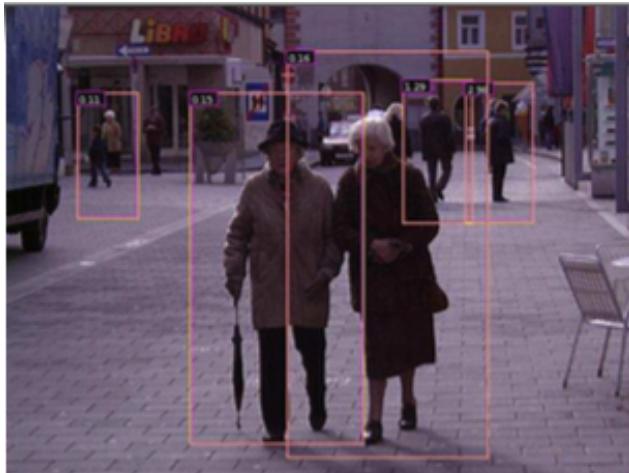


Histograms of Oriented Gradients (HOG)

N. Dalal and B. Triggs, “Histograms of Oriented Gradients for Human Detection,” *CVPR*, 2005.
(cited number: [44941](#) from Google)

Contributions

- Focus on building **robust feature sets (HOG)**
 - Classifier is just linear SVM on normalised image windows
- A human data set (**INRIA Person dataset**)
 - <http://pascal.inrialpes.fr/data/human/>



INRIA Person dataset

- A human data set (**INRIA Person dataset**)
 - <http://pascal.inrialpes.fr/data/human/>

- Well-designed dataset.
- Humans are standing in different positions with different orientations and poses.
- Image resolution is 64 x 128

Data Set

<i>Train</i>	<i>Test</i>
614 positive images	288 positive images
1218 negative images	453 negative images
1208 positive windows	566 positive windows
Overall 1774 human annotations + reflections	



Human



Non-Human

INRIA Person dataset

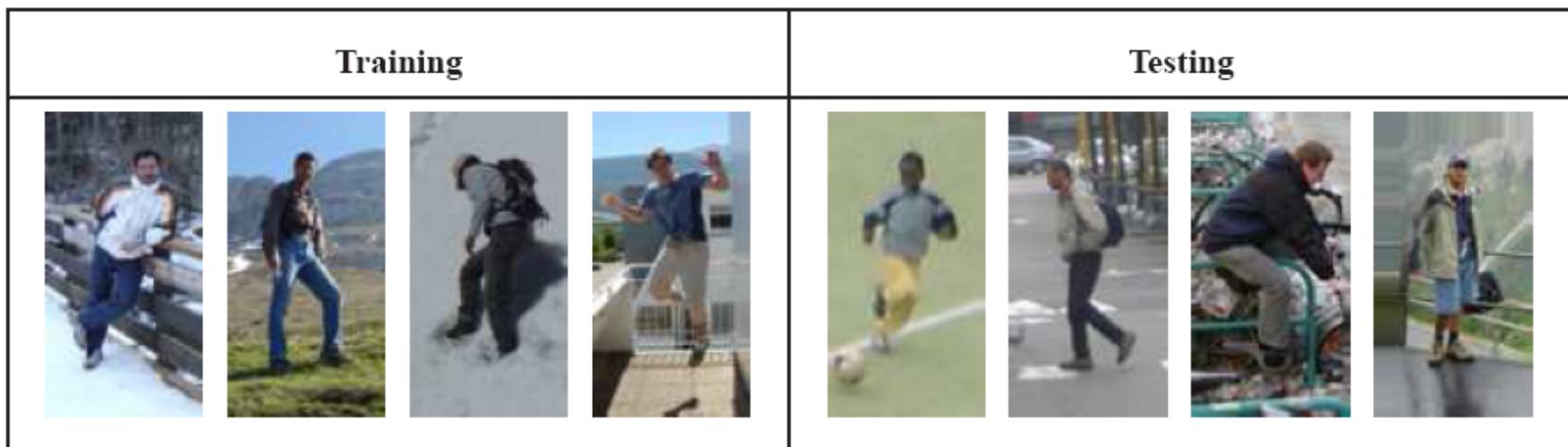
- A human data set (**INRIA Person dataset**)
 - <http://pascal.inrialpes.fr/data/human/>

- Well-designed dataset.
- Humans are standing in different positions with different orientations and poses.
- Image resolution is 64 x 128

Data Set

<i>Train</i>	<i>Test</i>
614 positive images	288 positive images
1218 negative images	453 negative images
1208 positive windows	566 positive windows
Overall 1774 human annotations + reflections	

Positive images:

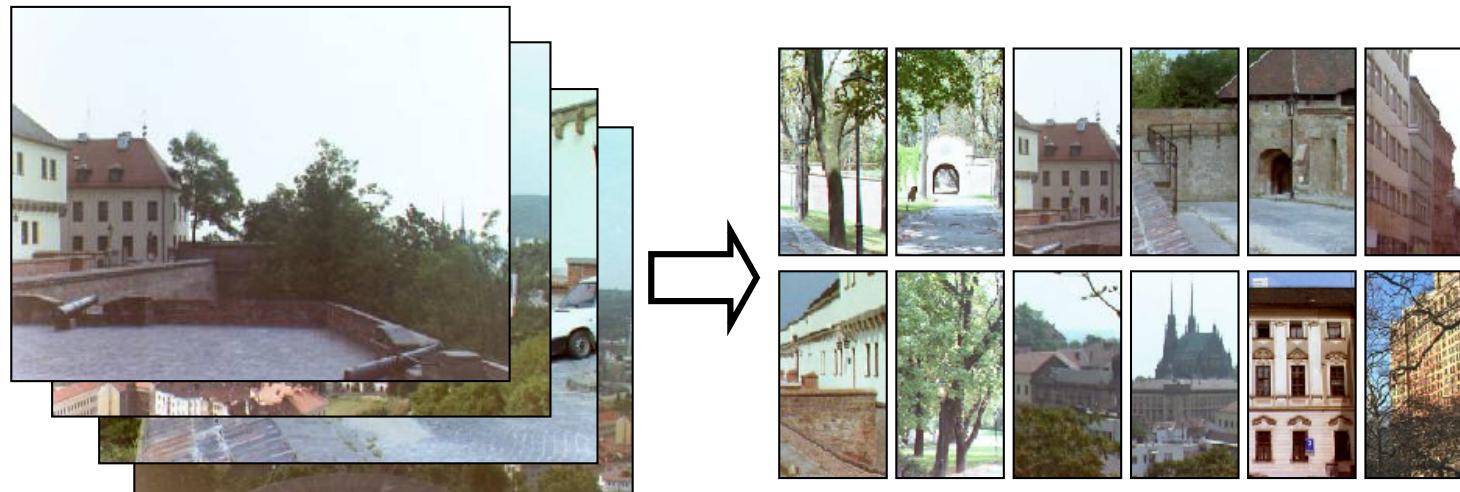


INRIA Person dataset

- A human data set (**INRIA Person dataset**)
 - <http://pascal.inrialpes.fr/data/human/>

- Well-designed dataset.
- Humans are standing in different positions with different orientations and poses.
- Image resolution is 64 x 128

Negative images:

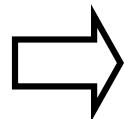
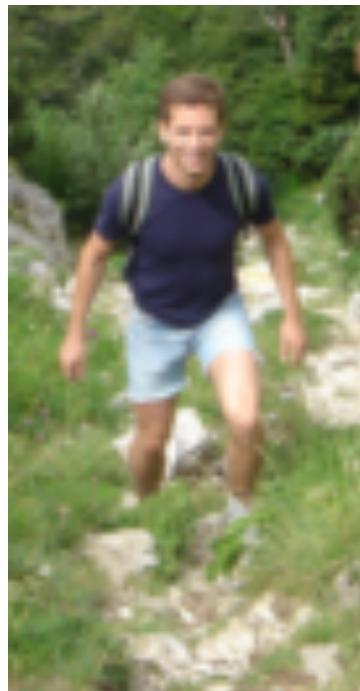


Data Set

<i>Train</i>	<i>Test</i>
614 positive images	288 positive images
1218 negative images	453 negative images
1208 positive windows	566 positive windows
Overall 1774 human annotations + reflections	

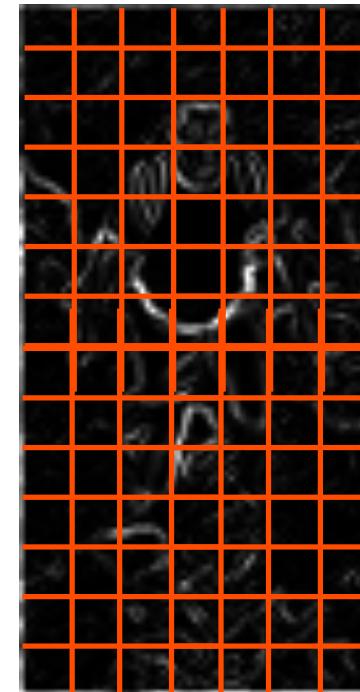
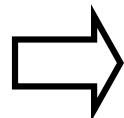
HOG Feature

- HOG (Histogram of Oriented Gradients)
 - For each training image (64 x128), the pixel **gradient magnitude** and **gradient orientation** are calculated



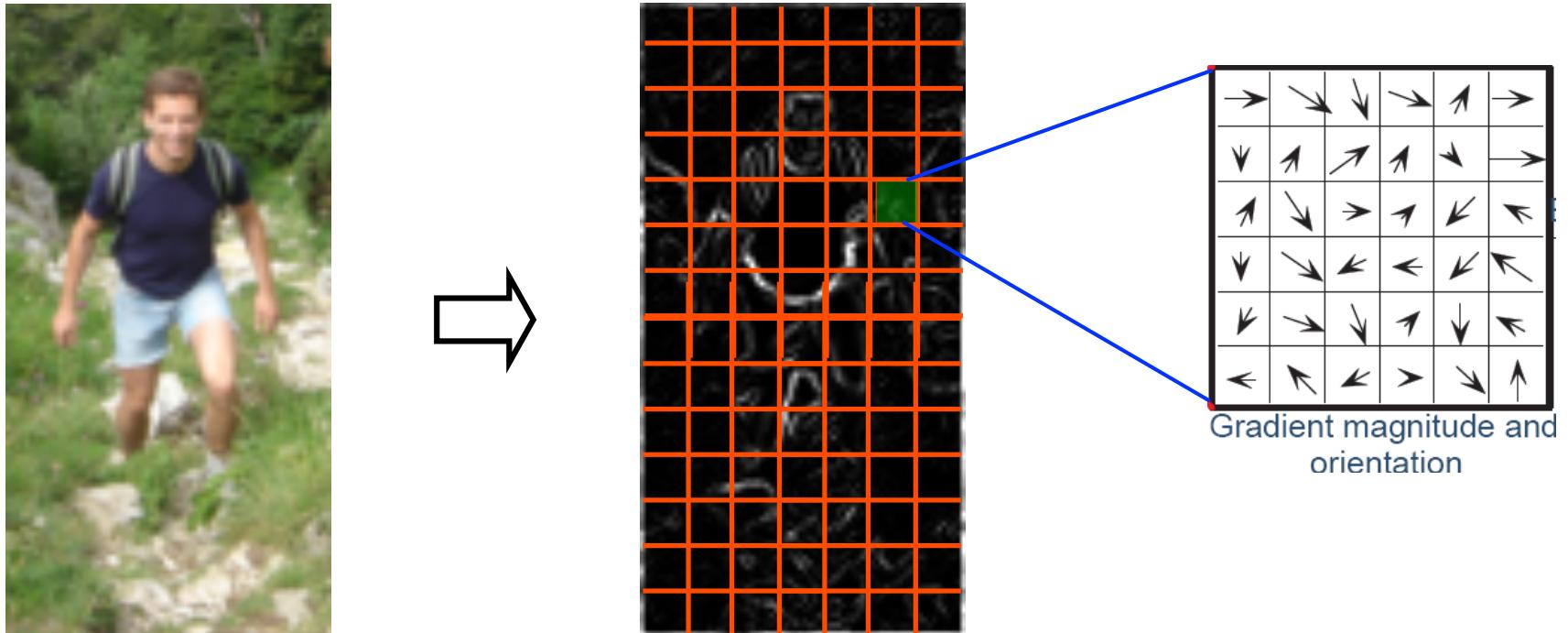
HOG Feature

- HOG (Histogram of Oriented Gradients)
 - For each training image (64×128), the pixel **gradient magnitude** and **gradient orientation** are calculated
 - Each training image (64×128) is divided into non-overlapped **cells** (8×8)
 - There are 8×16 cells.



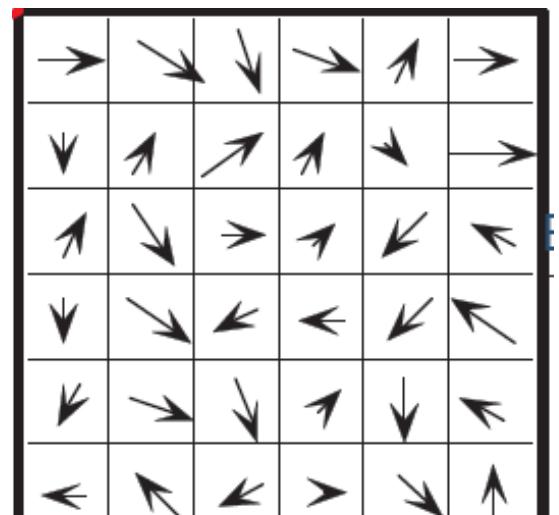
HOG Feature

- HOG (Histogram of Oriented Gradients)
 - For each training image (64×128), the pixel **gradient magnitude** and **gradient orientation** are calculated
 - Each training image (64×128) is divided into non-overlapped **cells** (8×8)
 - There are 8×16 cells.

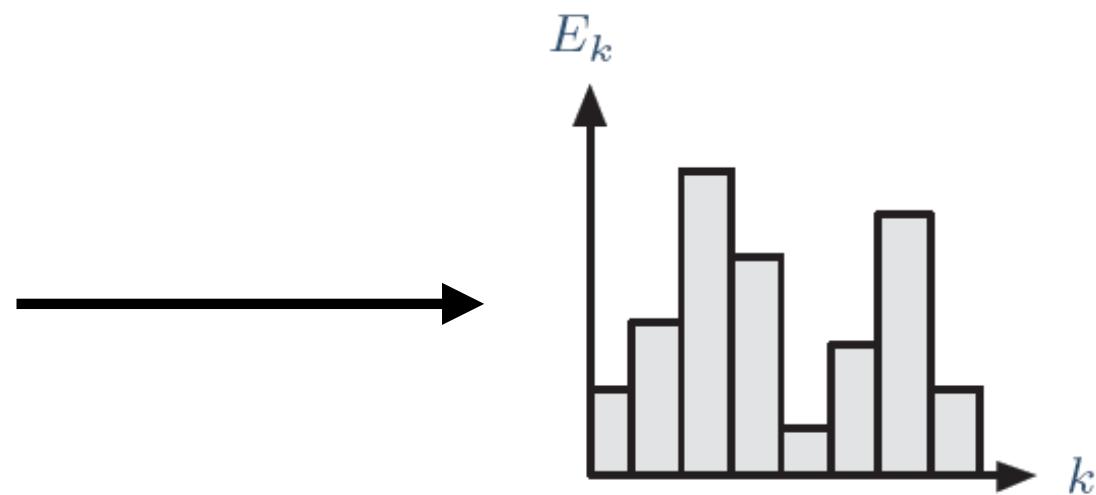


HOG Feature

- HOG (Histogram of Oriented Gradients)
 - For each cell, the gradient orientation histogram E (with K bins over $0^\circ \sim 180^\circ$) is measured.
 - ex. $K = 9$



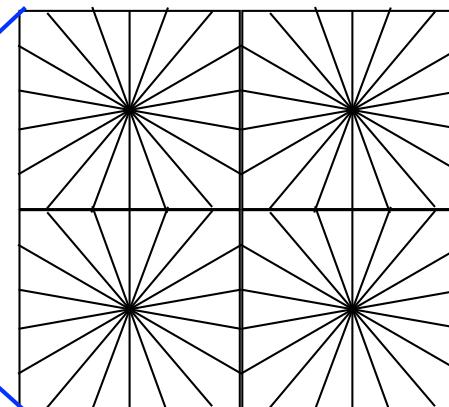
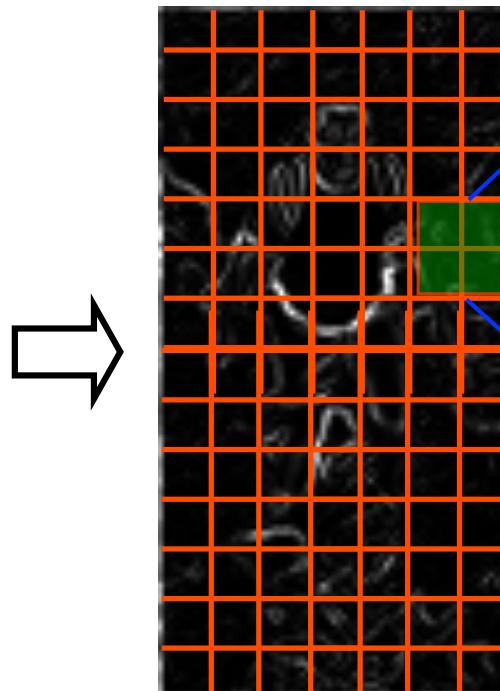
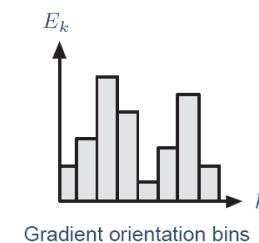
Gradient magnitude and orientation



Gradient orientation bins

HOG Feature

- HOG (Histogram of Oriented Gradients)
 - A **block** (16×16) is composed by four connected cells.
 - A block use 36 dimensional feature vector to capture local gradient orientation.
 - This feature vector is normalized to a unit vector.
 - Different blocks can be overlapped.
 - There are 7×15 blocks.

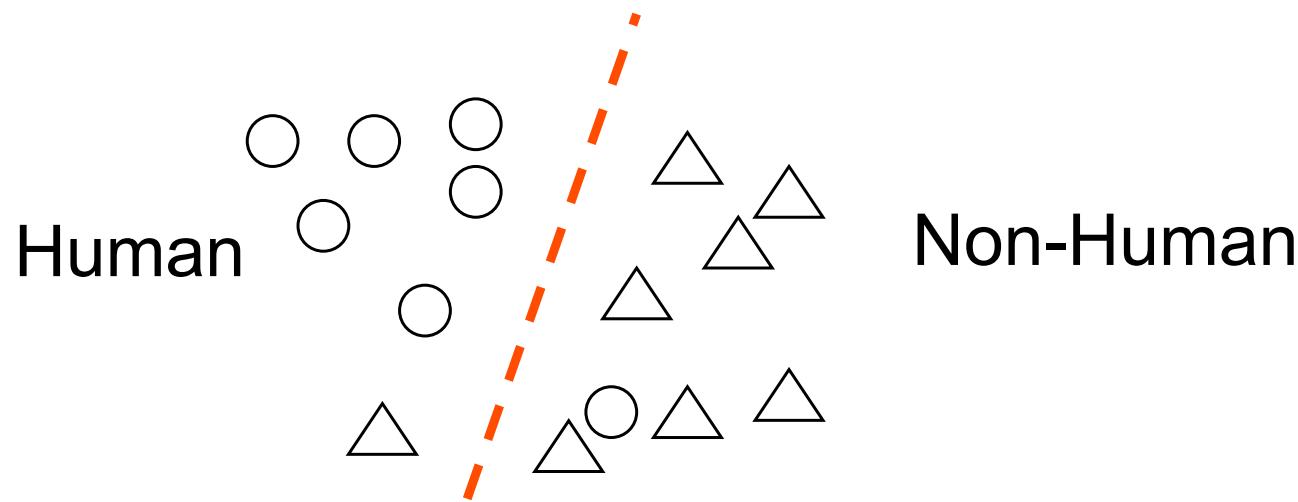
 $\in \mathbb{R}^{36}$ 

HOG Feature

- HOG (Histogram of Oriented Gradients)
 - Feature vectors of all blocks are combined as a super vector to capture global gradient orientation.
 - The super vector is a 3780 ($7 \times 15 \times 36$) dimensional vector.
 - In summary:
 - Each training data (a 64×128 image) is characterized by a 3780 dimensional vector.
 - This vector capture local and global gradient orientation information.
 - Drawback:
 - High dimensional feature vector \Rightarrow High computation Cost!

Classifier

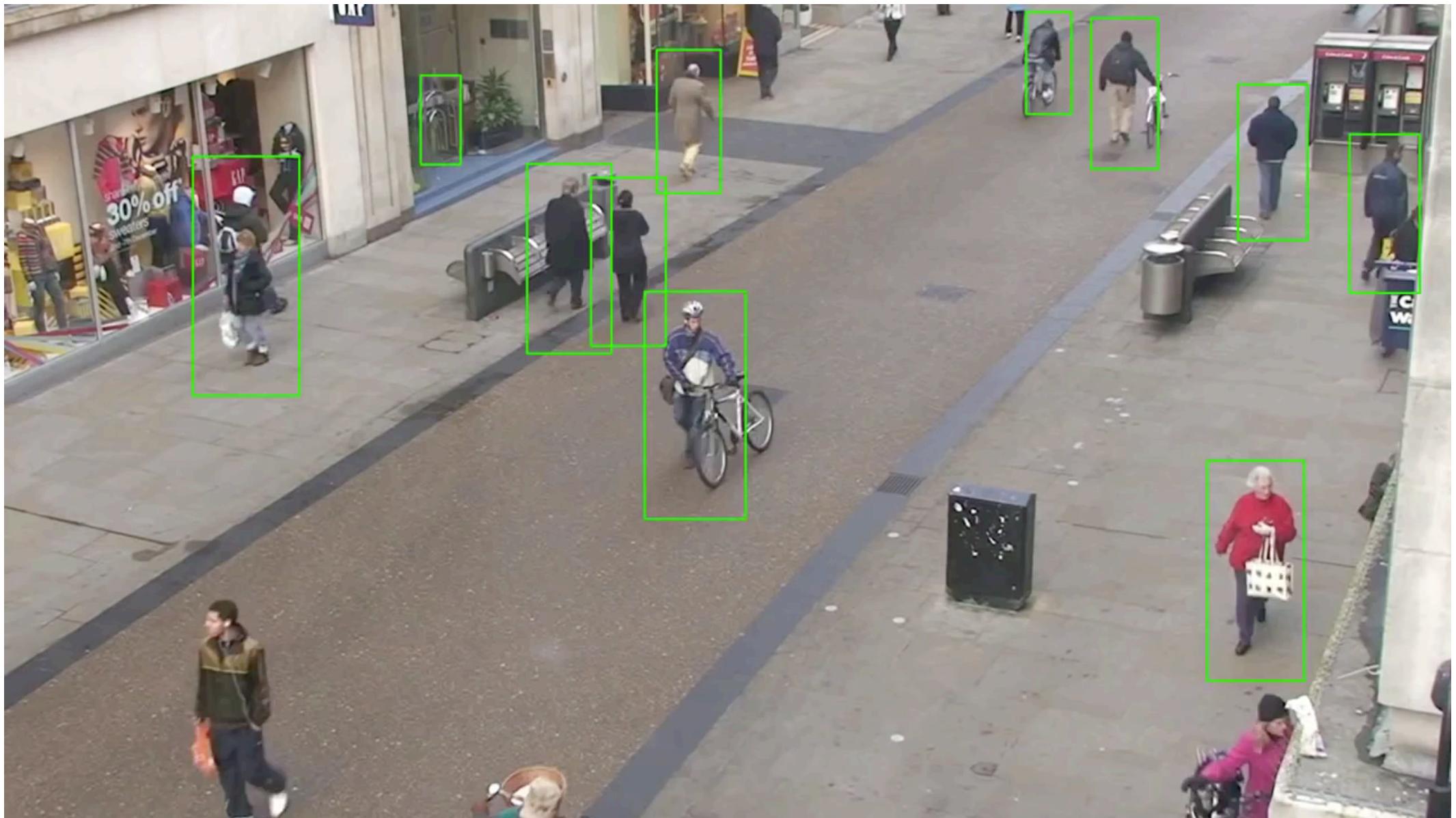
Linear Support Vector Machine (LSVM)



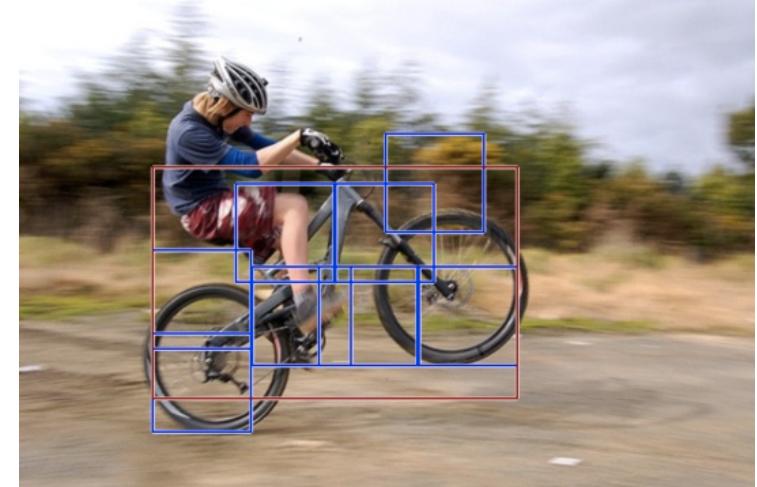
Results



Pedestrian detection using hog+svm (OpenCV)



Deformable Part Models

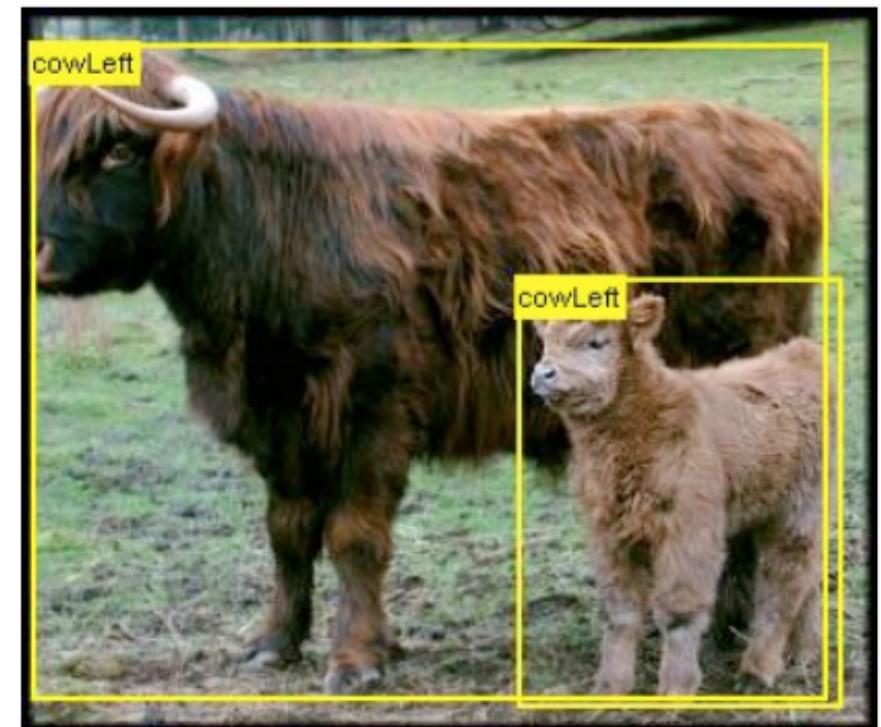


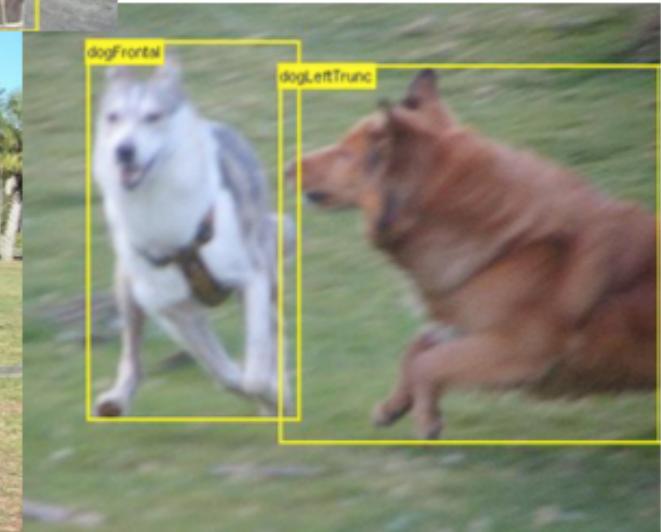
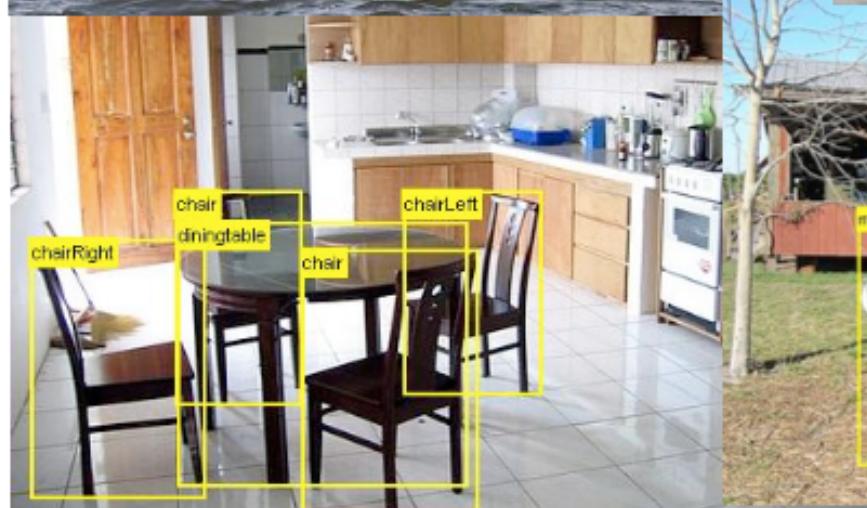
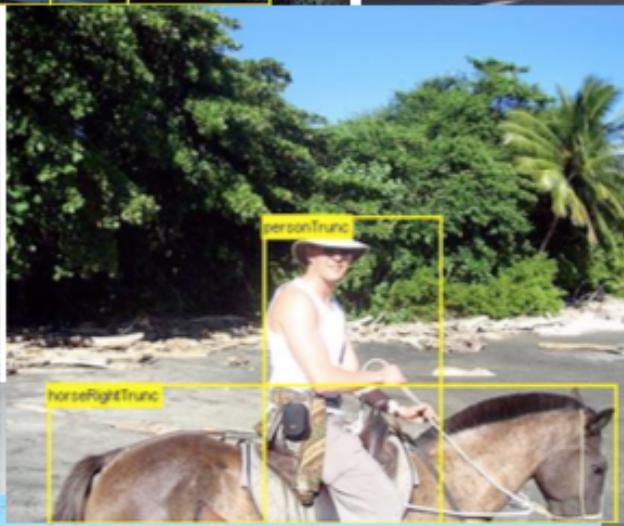
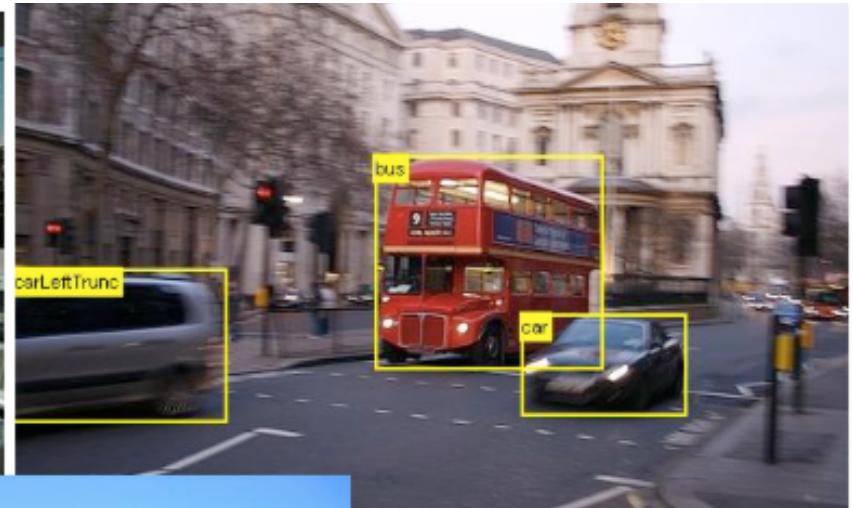
P. Felzenszwalb, D. McAllester, and D. Ramanan, “A Discriminatively Trained, Multiscale, Deformable Part Model,” *CVPR*, 2008.
(cited number: [3895](#) from Google)

P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan, “Object Detection with Discriminatively Trained Part Based Models,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.
(cited number: [12867](#) from Google)

PASCAL Challenge

- ~10,000 images, with ~25,000 target objects
 - Objects from 20 categories (person, car, bicycle, cow, table...)
 - Objects are annotated with labeled bounding boxes





Why is it hard?

- Object in rich categories exhibit significant variability
 - Photometric variation
 - Viewpoint variation
 - Intra-class variability
 - Cars come in a variety of shapes (sedan, minivan, etc)
 - People wear different clothes and take different poses

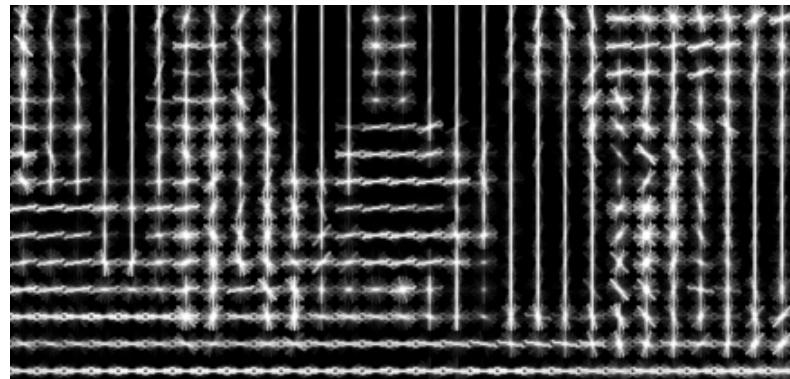
We need rich object models

But this leads to difficult matching and training problems

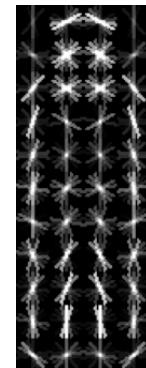
Pedestrian detection with HOG

- Train a pedestrian template using a linear support vector machine
- At test time, convolve feature map with template
- Find local maxima of response
- For multi-scale detection, repeat over multiple levels of a HOG *pyramid*

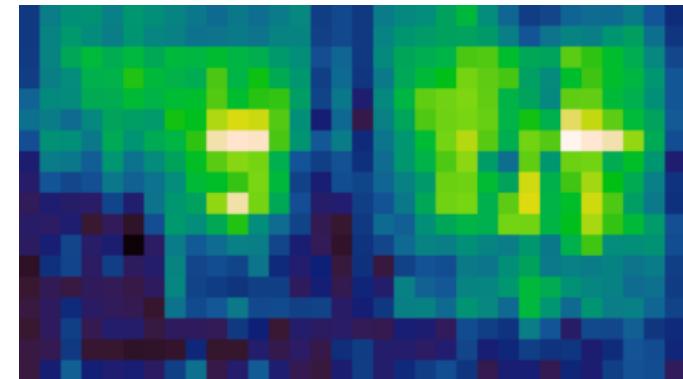
HOG feature map



Template



Detector response map



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

Challenges

- Single rigid template usually not enough to represent a category
 - Many objects (e.g. humans) are articulated, or have parts that can vary in configuration



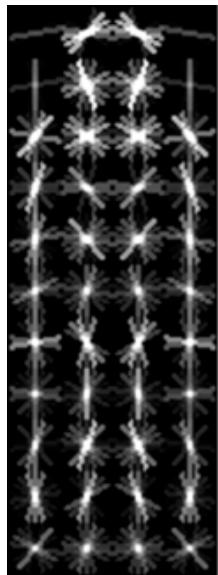
- Many object categories look very different from different viewpoints, or from instance to instance



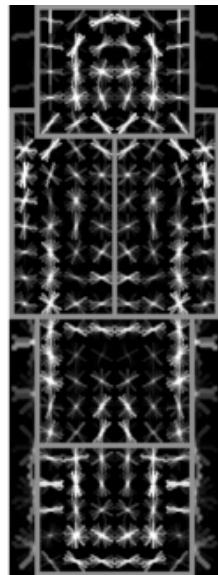
Slide by N. Snavely

Deformable part models

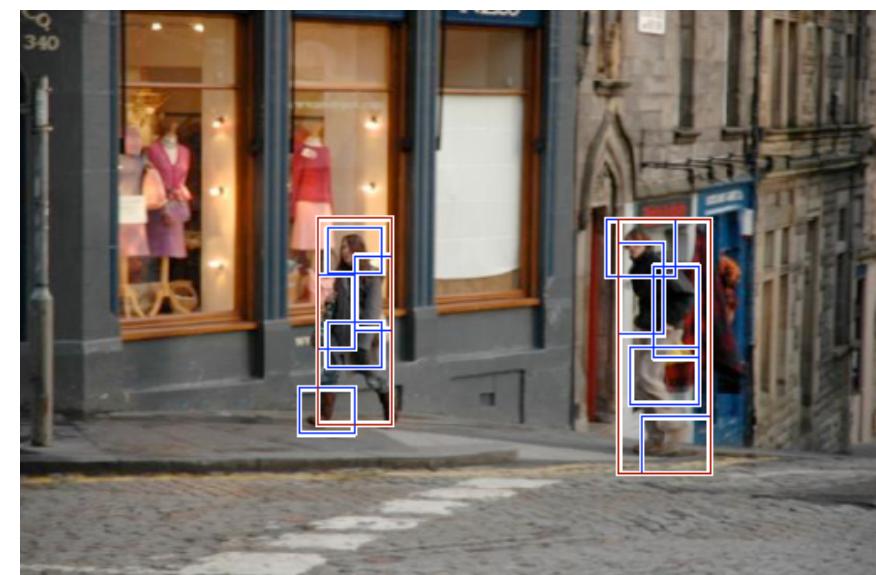
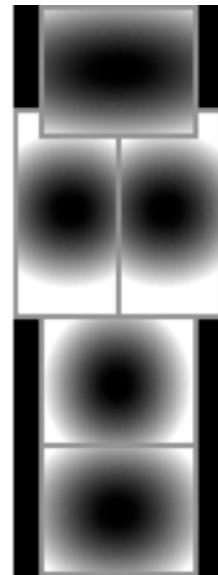
Root
filter



Part
filters



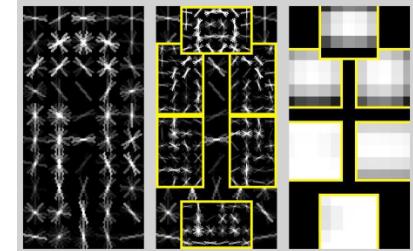
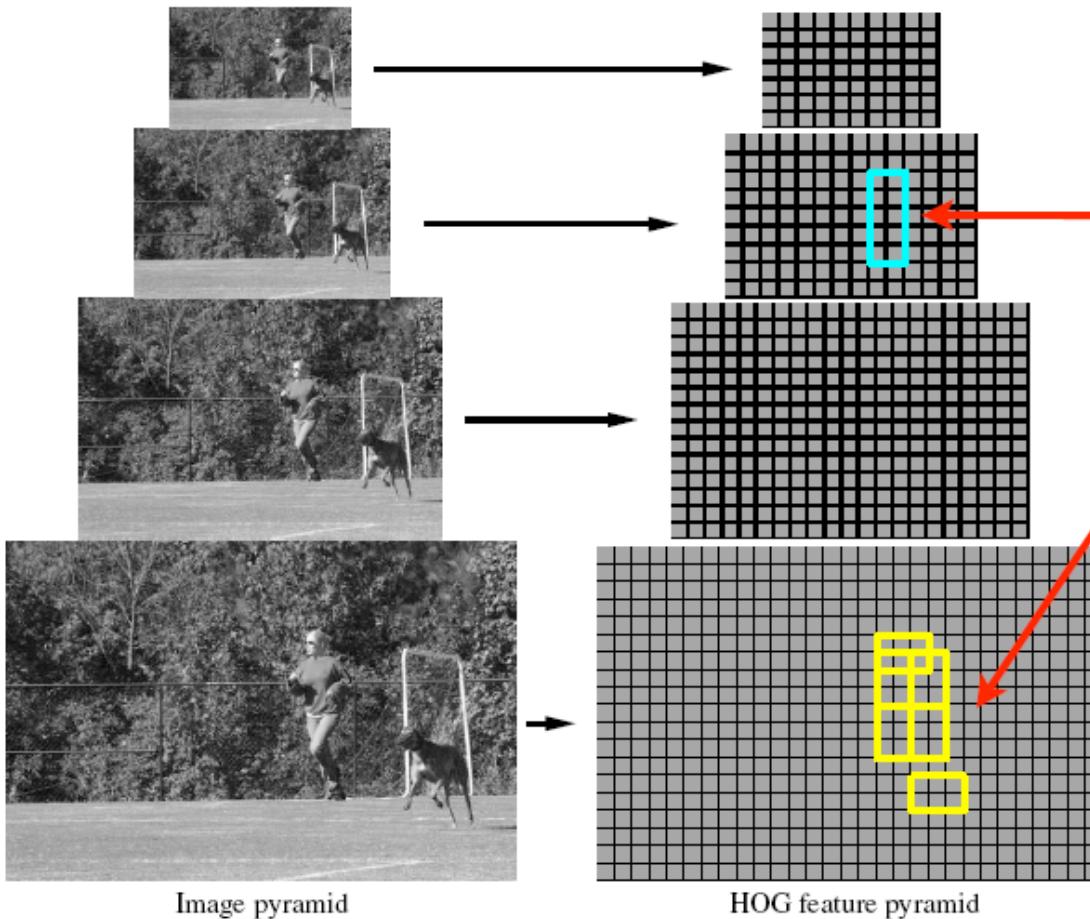
Deformation
weights



- Mixture of deformable part models
- Each component has global template + deformable parts
- Fully trained from bounding boxes alone

Object hypothesis

- Multiscale model: the resolution of part filters is **twice the resolution** of the root



$$z = (p_0, \dots, p_n)$$

p_0 : location of root

p_1, \dots, p_n : location of parts

Score is sum of filter
scores minus
deformation costs

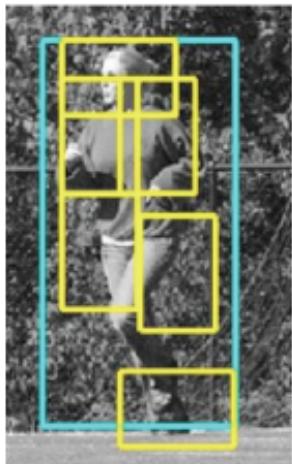
Scoring an object hypothesis

- The score of a hypothesis is the sum of filter scores minus the sum of deformation costs

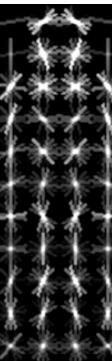
$$score(\mathbf{p}_0, \dots, \mathbf{p}_n) = \sum_{i=0}^n \mathbf{F}_i \cdot \mathbf{H}(\mathbf{p}_i) - \sum_{i=1}^n \mathbf{D}_i \cdot (dx_i, dy_i, dx_i^2, dy_i^2)$$

↑ ↑
Filters Deformation weights
Subwindow features Displacements

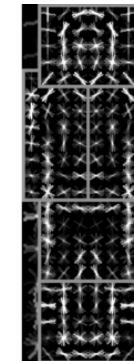
Score = (filter scores) - (deformation costs)



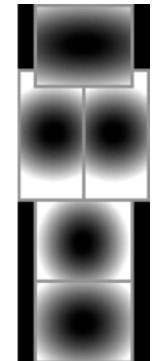
Root filter



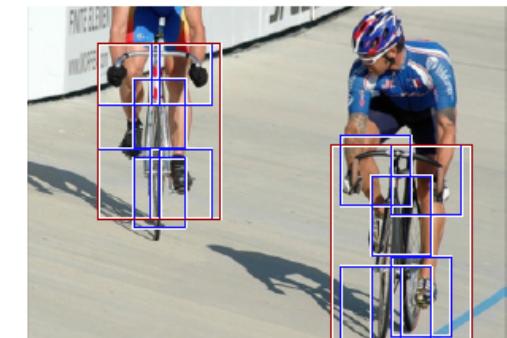
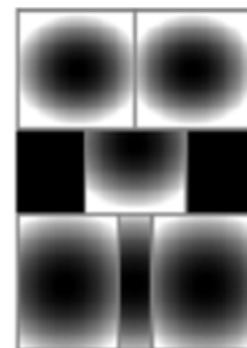
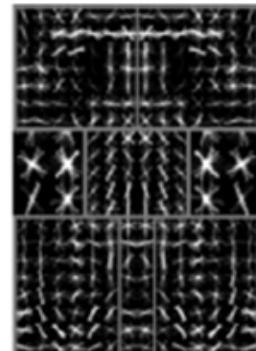
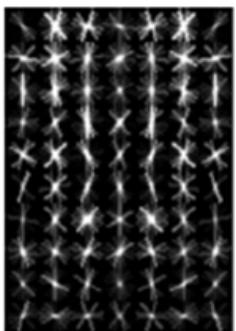
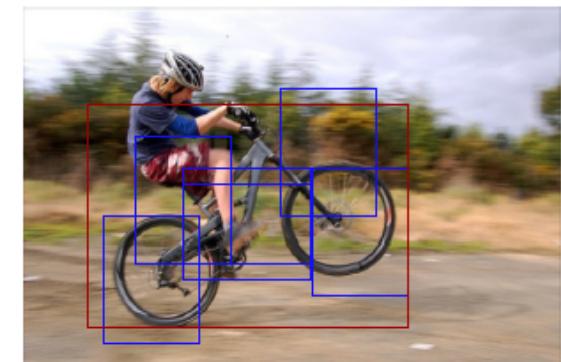
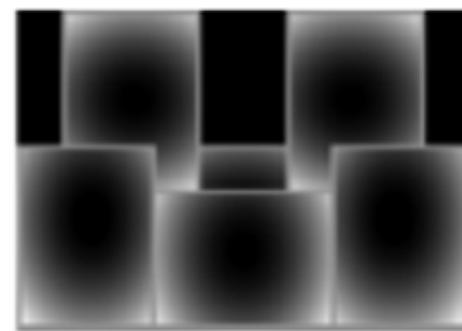
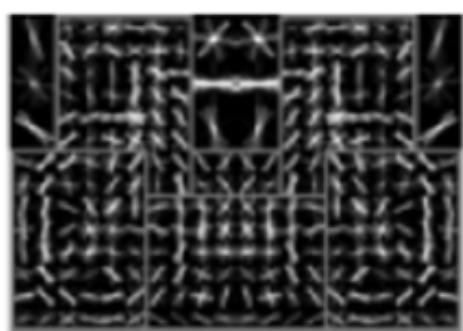
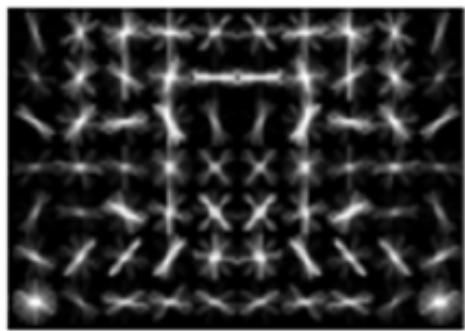
Part filters



Deformation weights

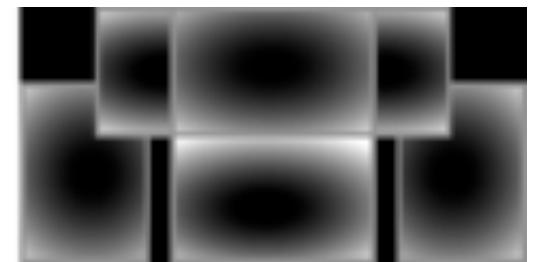
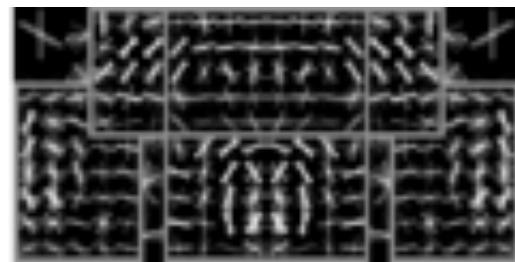
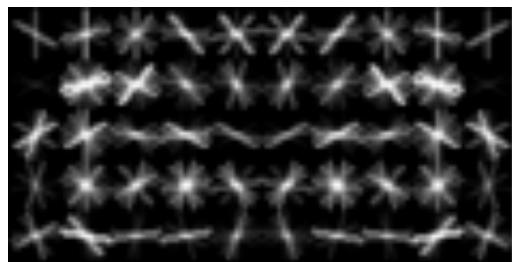


Bicycle model

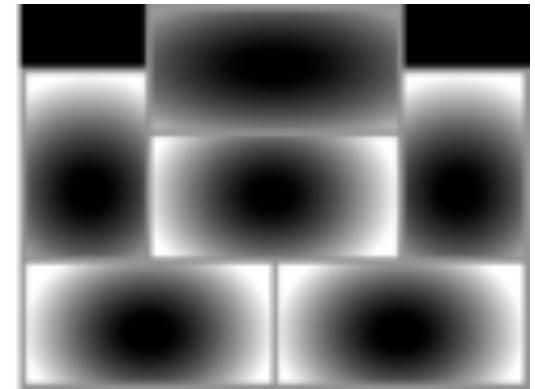
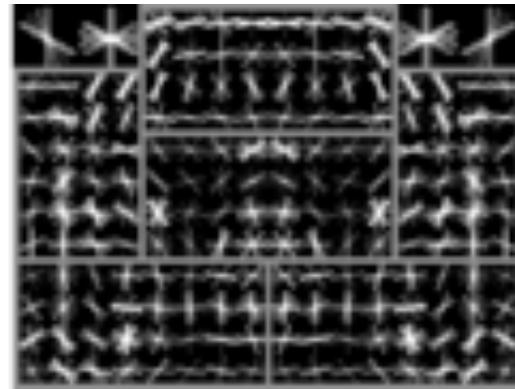
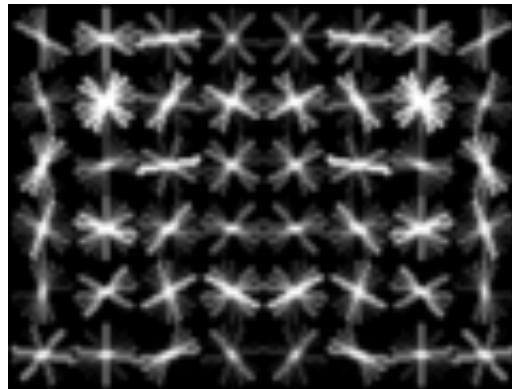


Car model

Component 1

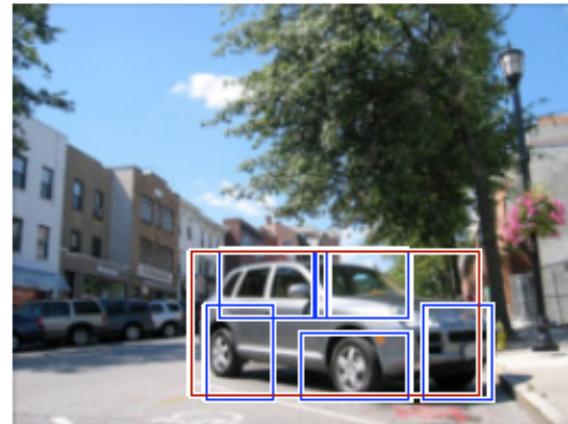
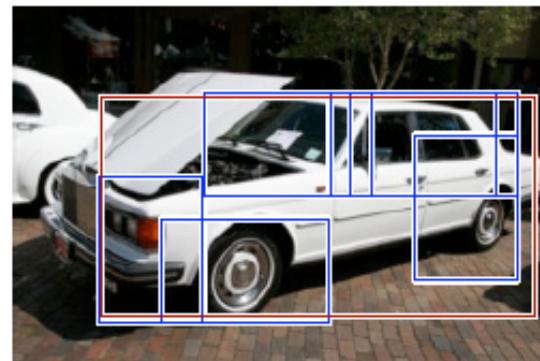
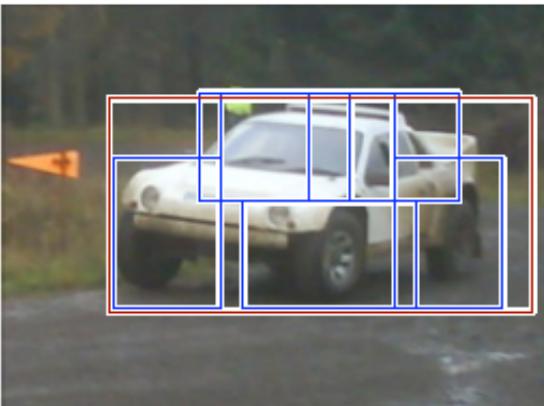


Component 2

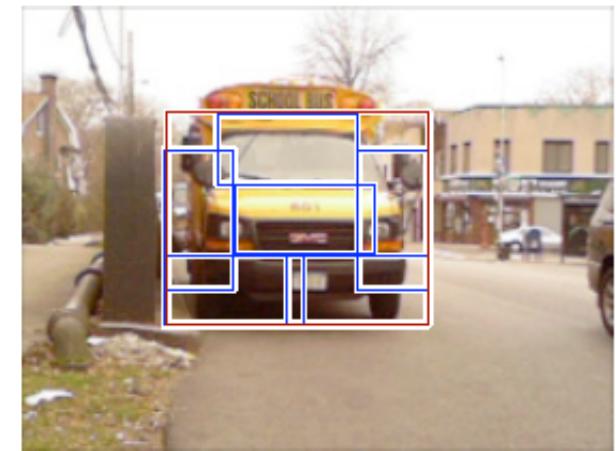
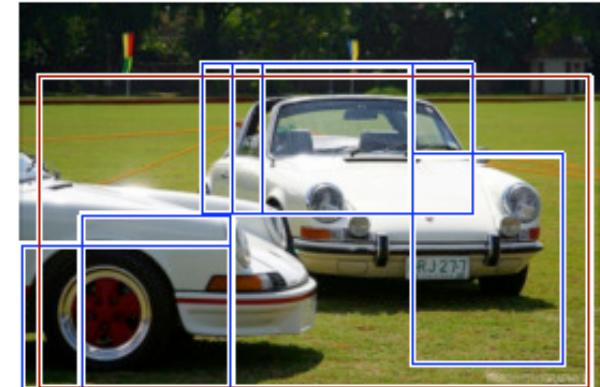


Car detection

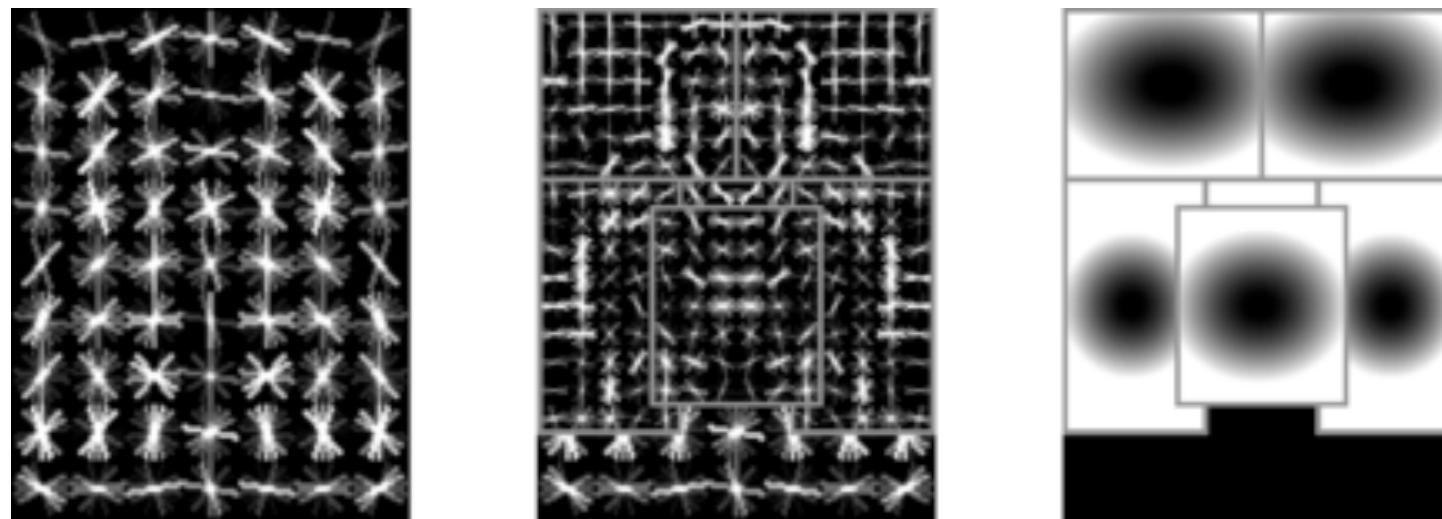
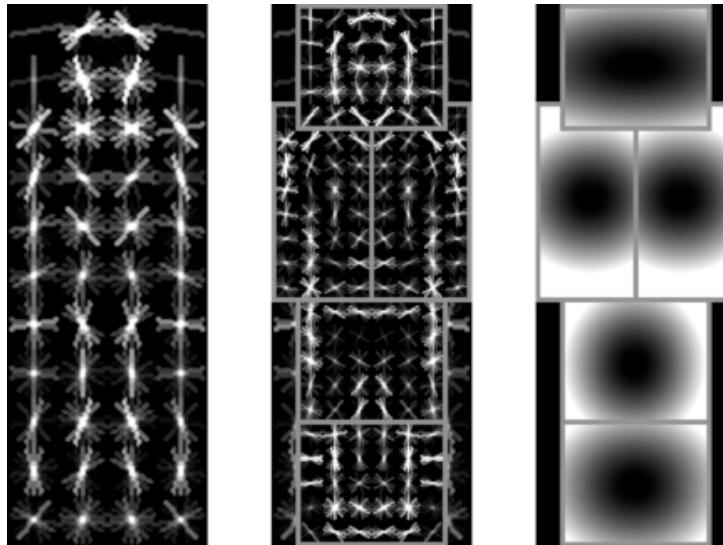
high scoring true positives



high scoring false positives

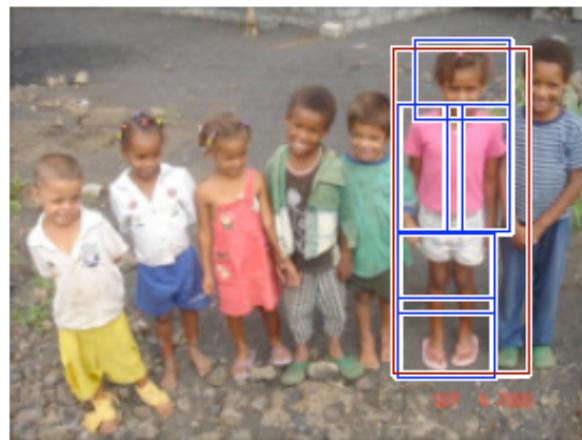
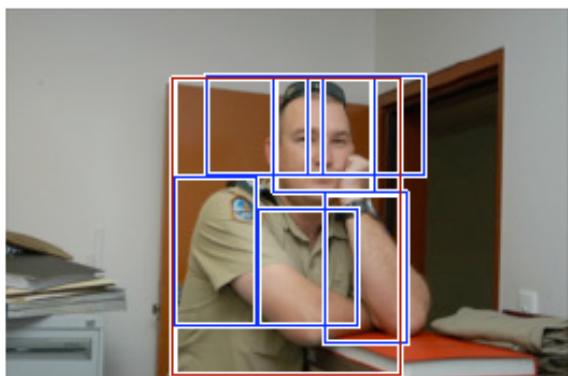
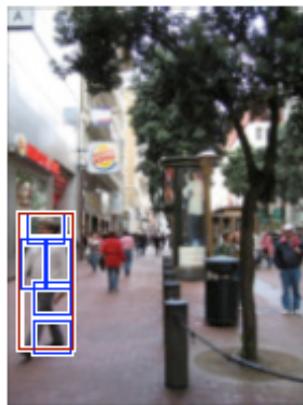
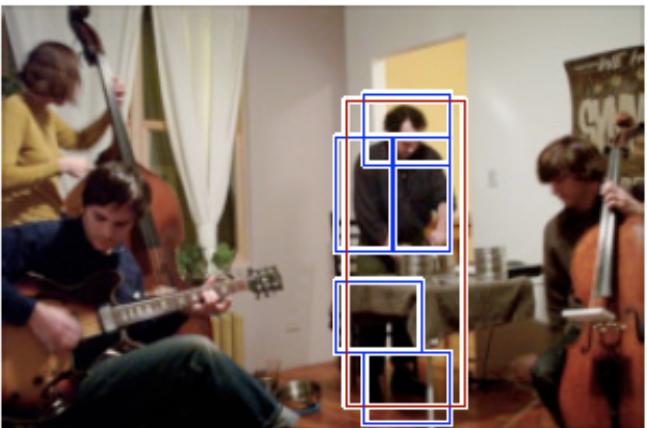


Person model

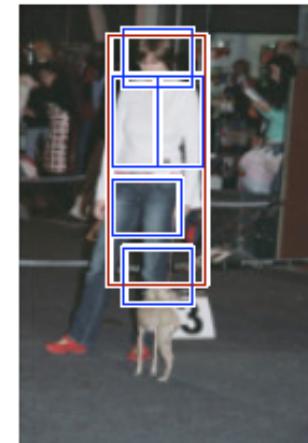


Person detection

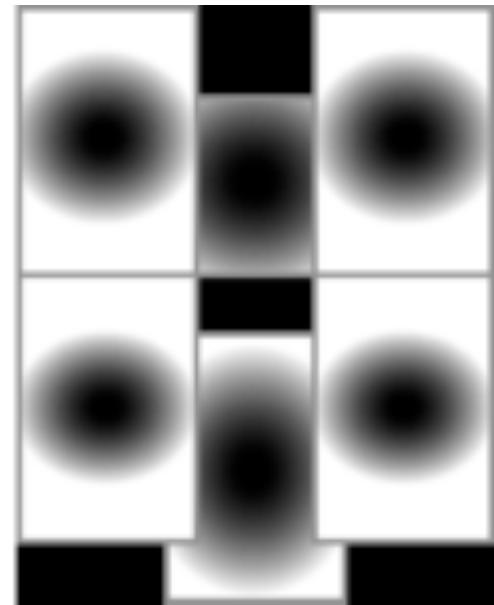
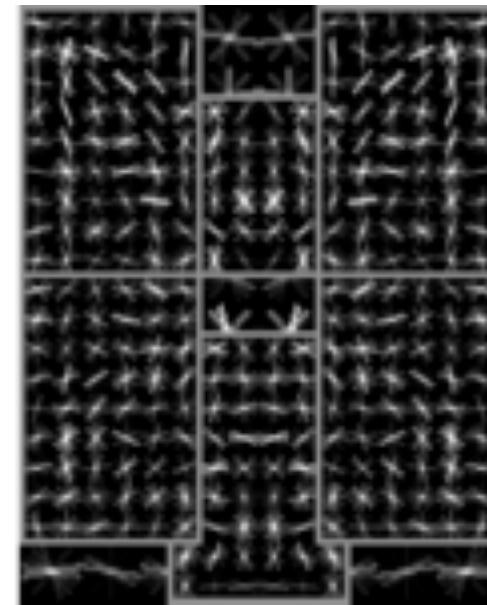
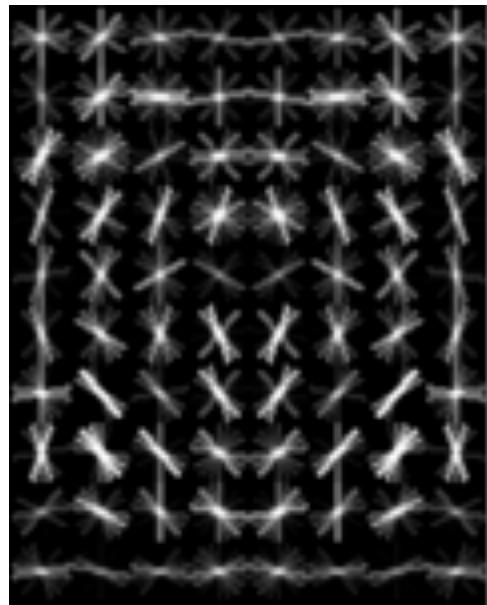
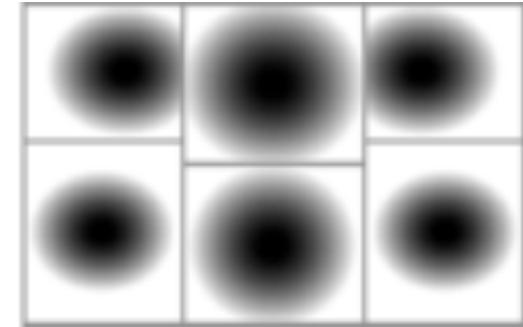
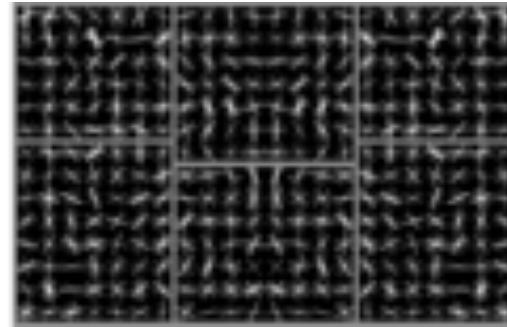
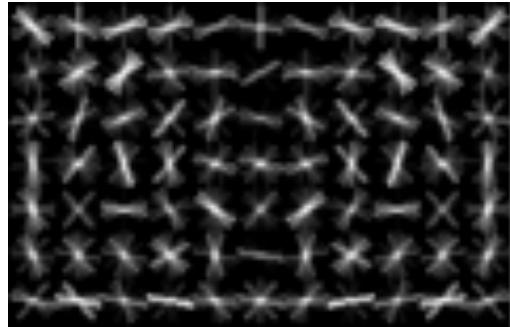
high scoring true positives



high scoring false positives
(not enough overlap)

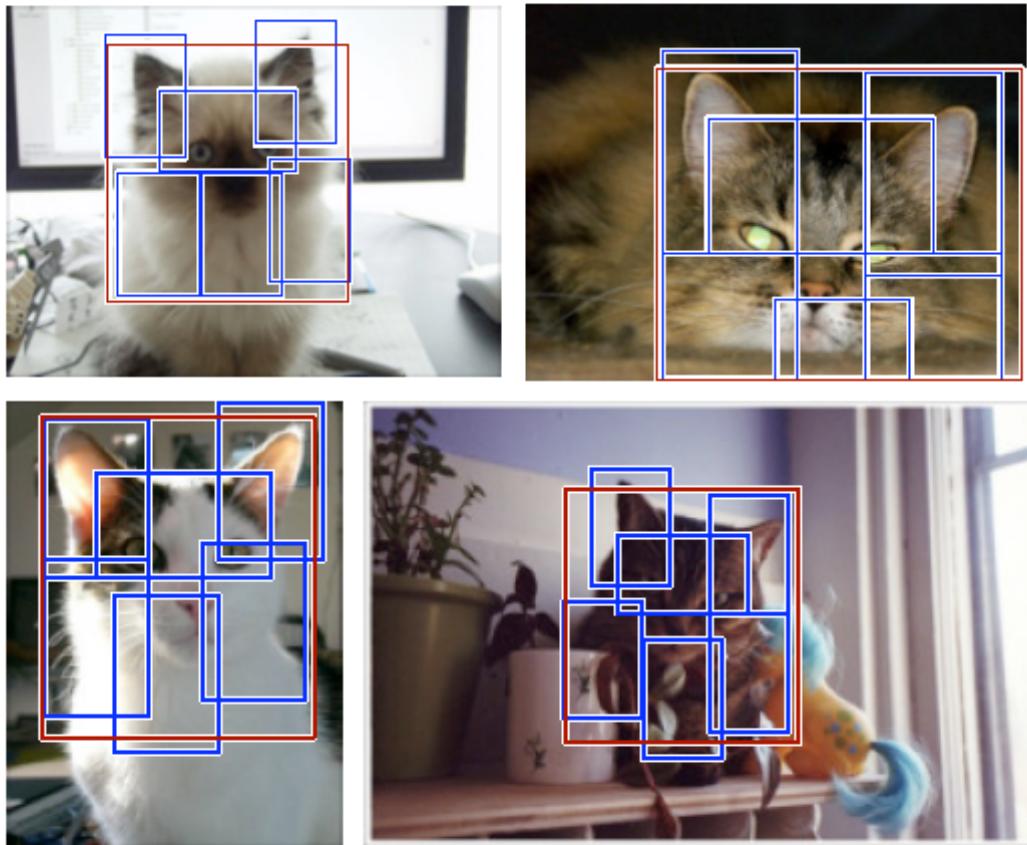


Cat model

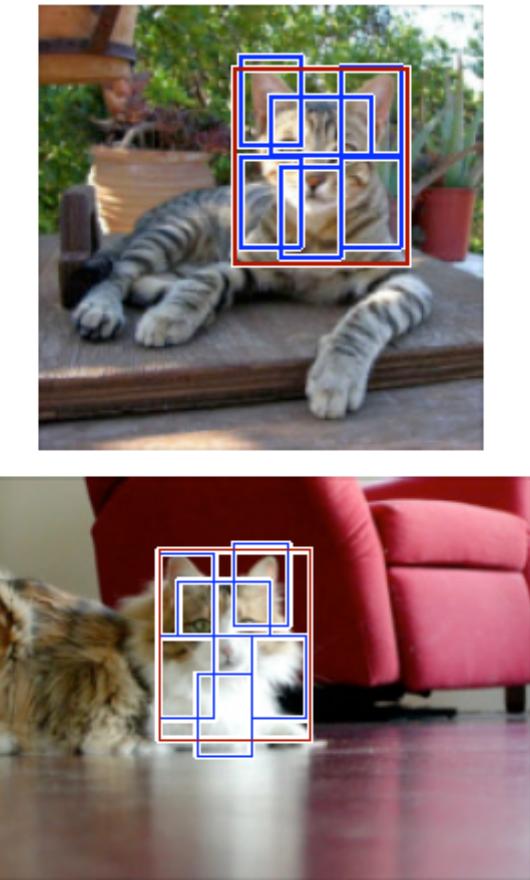


Cat detection

high scoring true positives

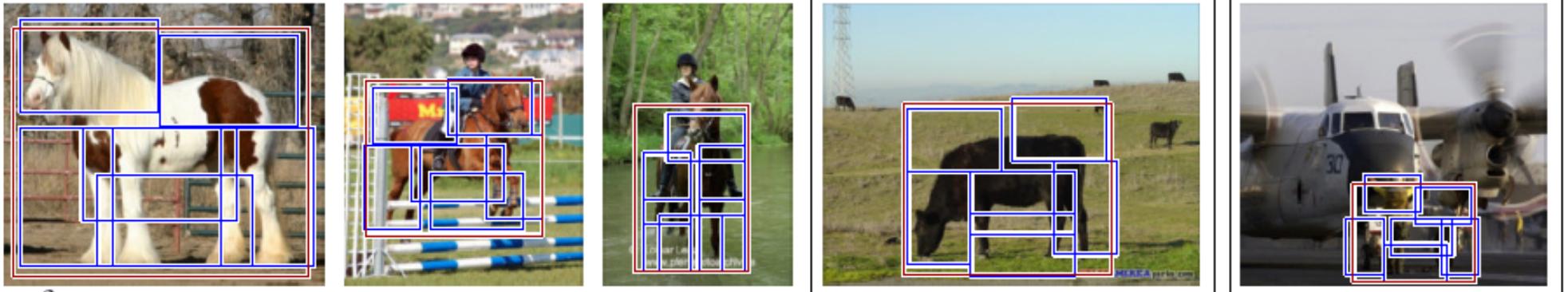


high scoring false positives
(not enough overlap)

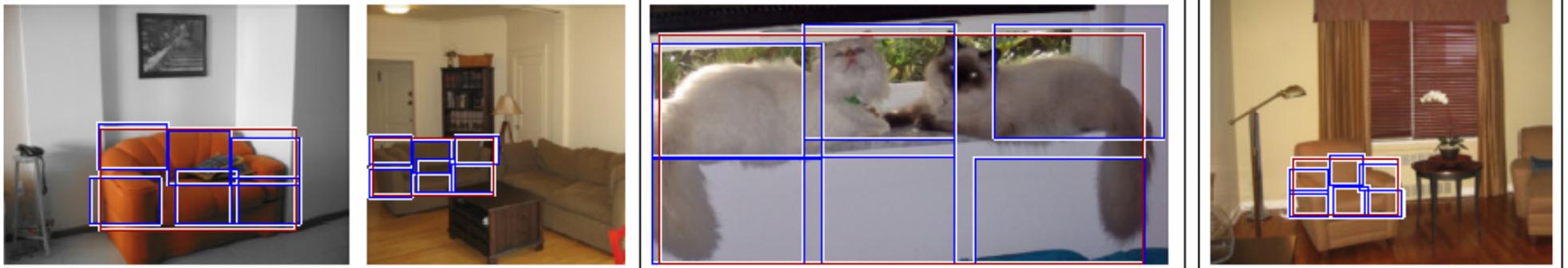


More detections

horse



sofa



bottle



More detections

Deformable Part Models



Classification & Objection Detection

Kuan-Wen Chen