CAP 4453
Robot Vision

Dr. Gonzalo Vaca-Castaño

gonzalo.vacacastano@ucf.edu

Administrative details

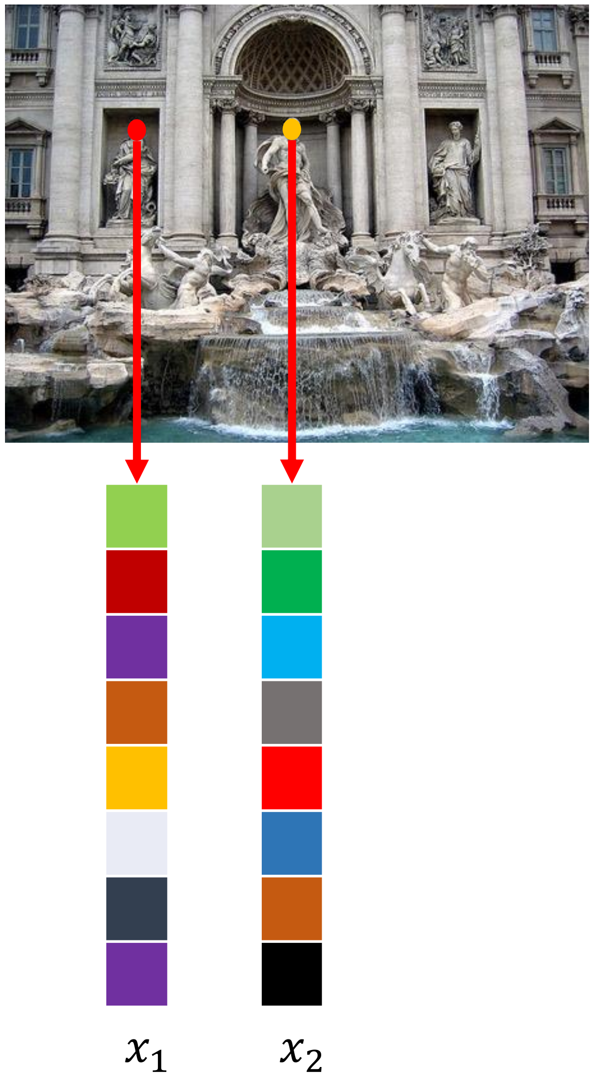
- Issues submitting homework

Credits

- Some slides comes directly from:
  - Kristen Grauman
  - A. Zisserman
  - Ross B. Girshick

Short Review from last class

Feature Descriptor

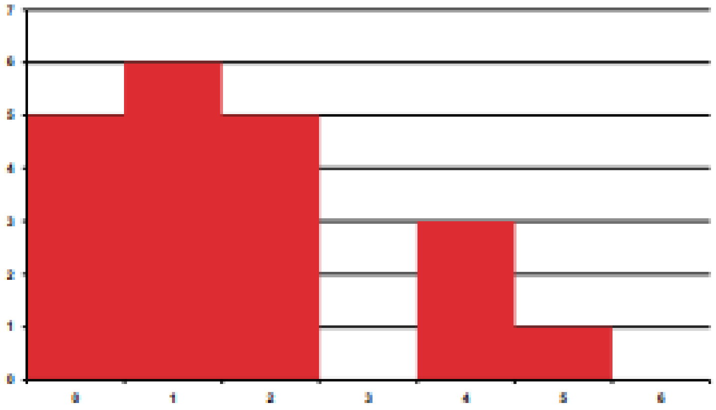


Histogram of Oriented Gradients (HOG)

- Revisiting histogram

0	1	1	2	4
2	1	0	0	2
5	2	0	0	4
1	1	2	4	1

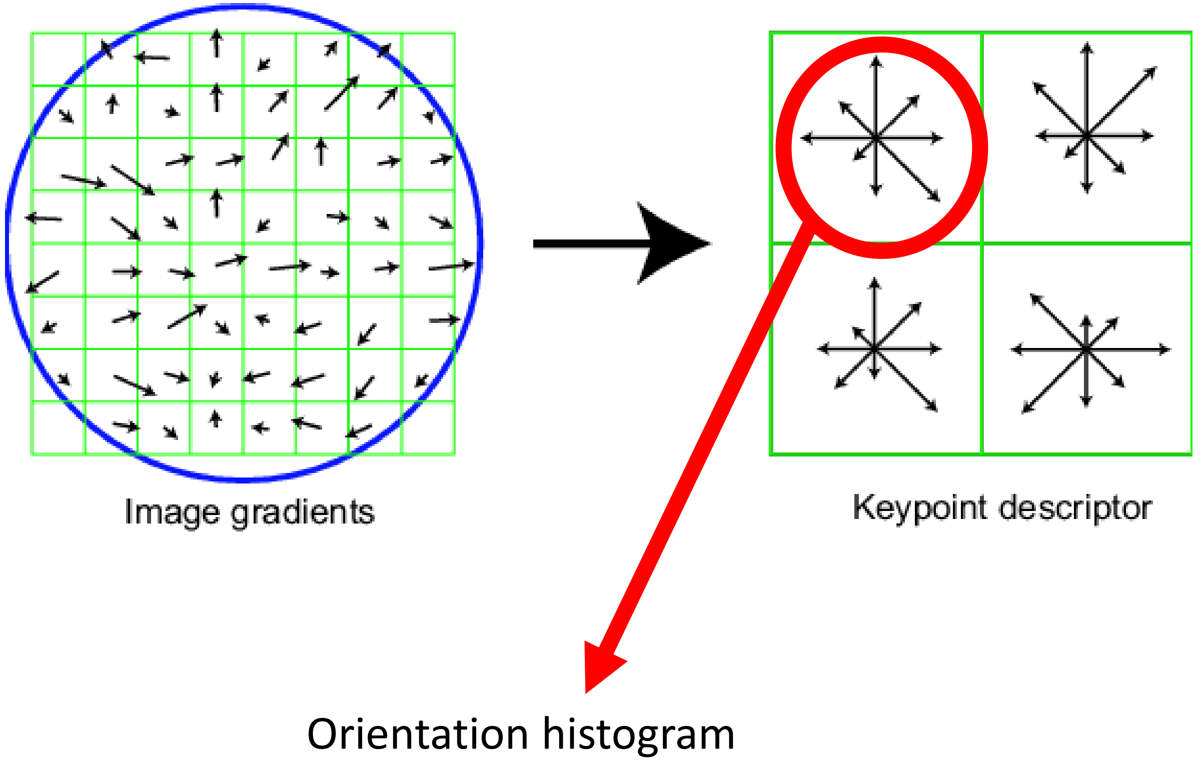
image



histogram

Invariance to deformation

- Deformation can also move pixels around
- Again, instead of precise location of each pixel, only want to record rough location
- Divide patch into a grid of *cells*
- Record *counts* of each orientation in each cell: *orientation histograms*



Feature detection and description

- Harris corner detection gives:
  - Location of each detected corner
  - Orientation of the corner (given by $\mathbf{x}_{max}$)
  - Scale of the corner (the image scale which gives the maximum response at this location)
- Want feature descriptor that is
  - Invariant to photometric transformations, translation, rotation, scaling
  - Discriminative

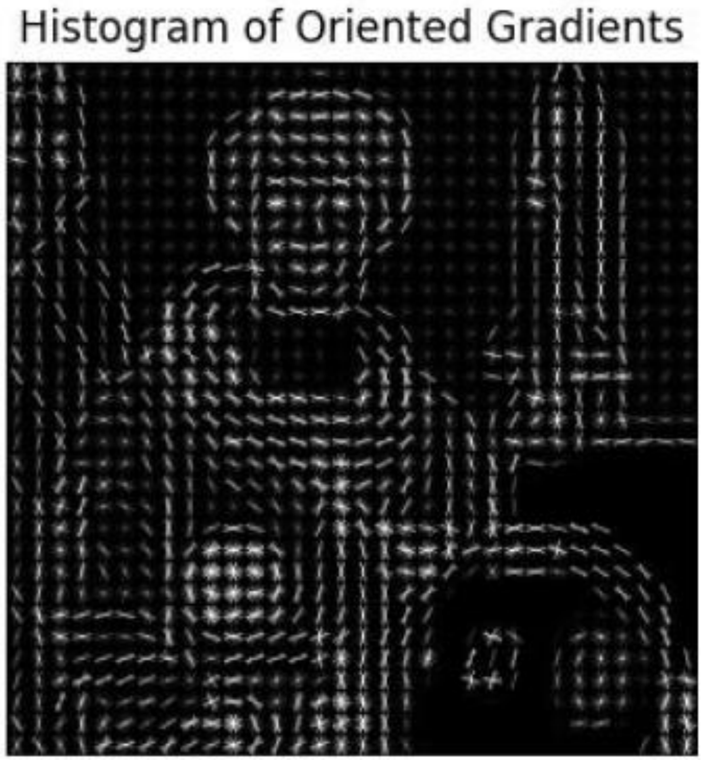
Summary of HOG computation

- Step 1: Extract a square window (called "block") of some size around the pixel location of interest.
- Step 2: Divide block into a square grid of sub-blocks (called "cells") (2x2 grid in our example, resulting in four cells).
- Step 3: Compute orientation histogram of each cell.
- Step 4: Concatenate the four histograms.
- Step 5: normalize v using one of the three options:
  - Option 1: Divide v by its Euclidean norm.
  - Option 2: Divide v by its L1 norm (the L1 norm is the sum of all absolute values of v).
  - Option 3:
    - Divide v by its Euclidean norm.
    - In the resulting vector, clip any value over 0.2
    - Then, renormalize the resulting vector by dividing again by its Euclidean norm

Histogram of Oriented Gradients (HOG)

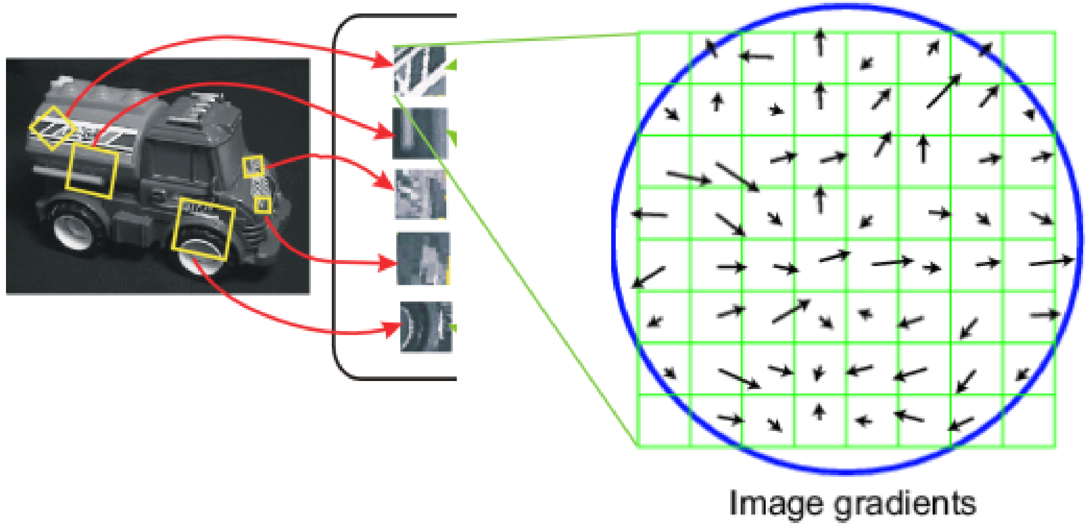
- Parameters and design options:
- Angles range from 0 to 180 or from 0 to 360 degrees?
  - In the Dalal & Triggs paper, a range of 0 to 180 degrees is used
- Number of orientation bins.
  - Usually 9 bins, each bin covering 20 degrees.
- Cell size.
  - Cells of size 8x8 pixels are often used.
- Block size.
  - Blocks of size 2x2 cells (16x16 pixels) are often used.
- Usually a HOG feature has 36 dimensions.
  - 4 cells * 9 orientation bins.

Histogram of Oriented Gradients (HOG)



SIFT descriptor

- Compute on local 16 x 16 window around detection.
- Rotate and scale window according to discovered orientation ϴ and scale σ (gain invariance).
- Compute gradients weighted by a Gaussian of variance half the window (for smooth falloff).

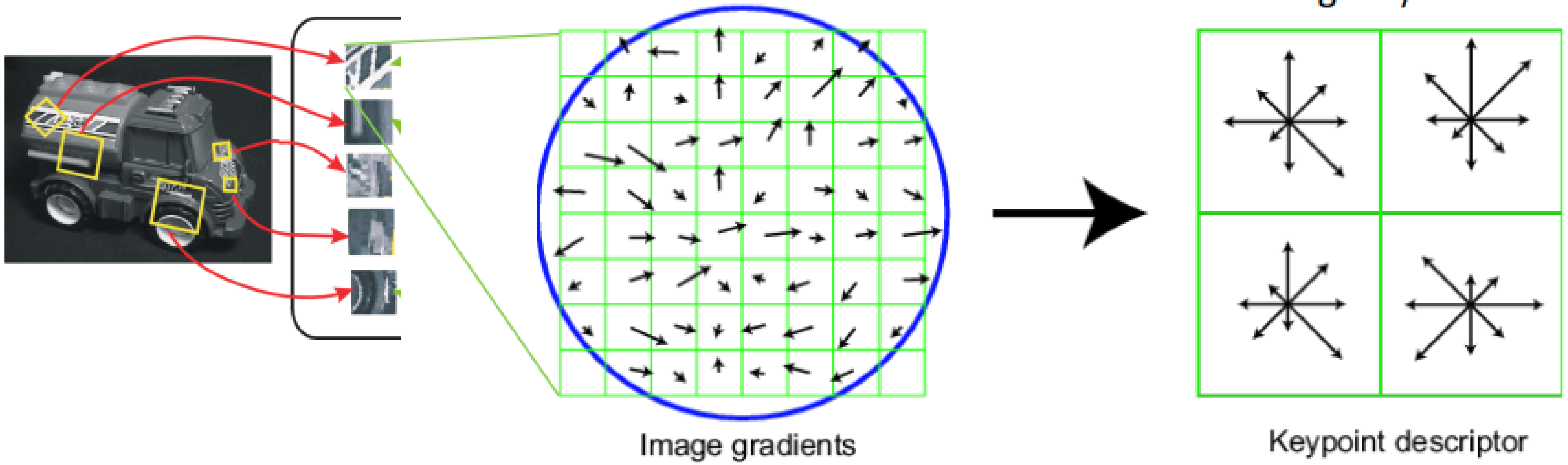


Actually 16x16, only showing 8x8

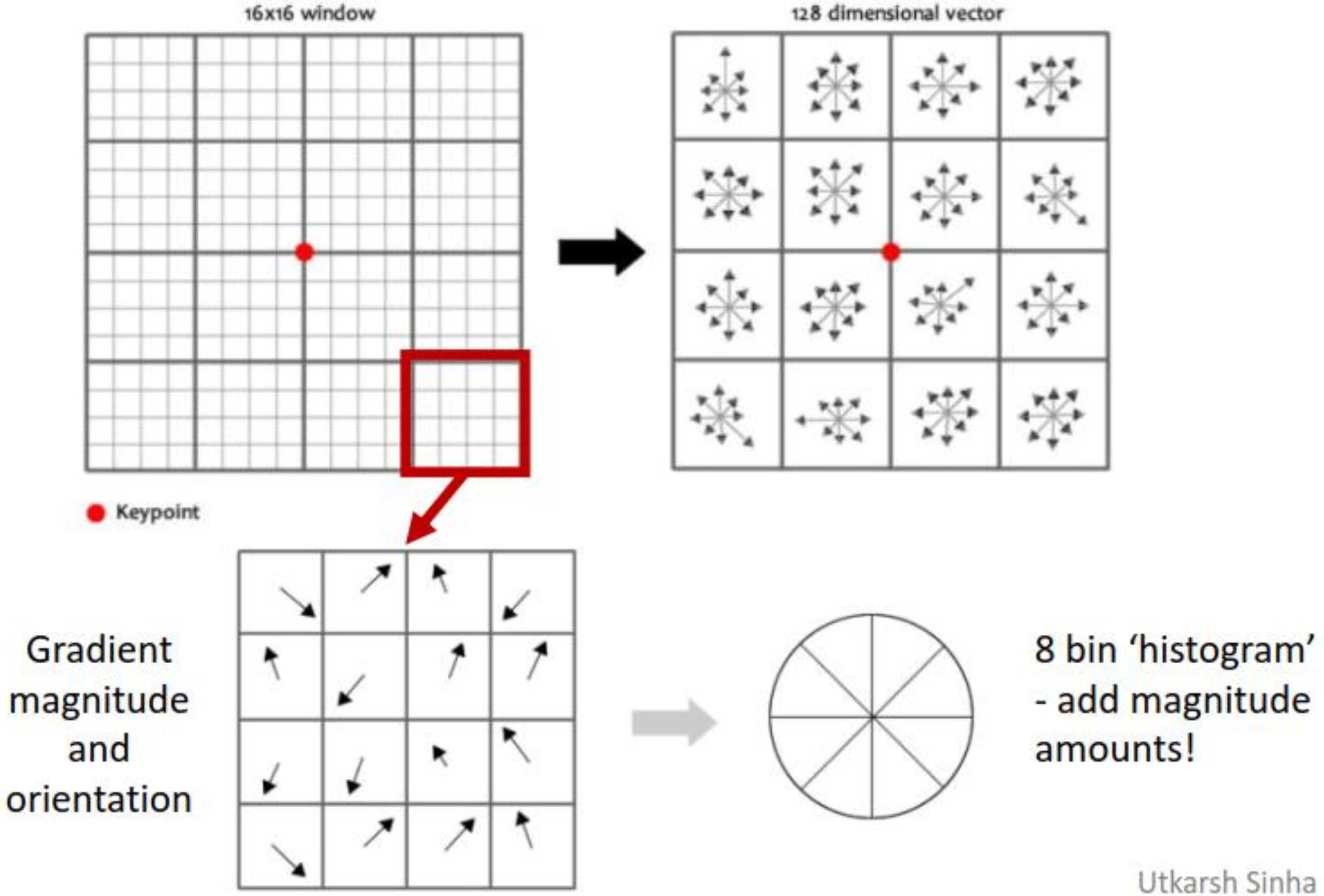
SIFT descriptor

- 4x4 array of gradient orientation histograms weighted by gradient magnitude.
- Bin into 8 orientations x 4x4 array = 128 dimensions.

Showing only 2x2 here but is 4x4

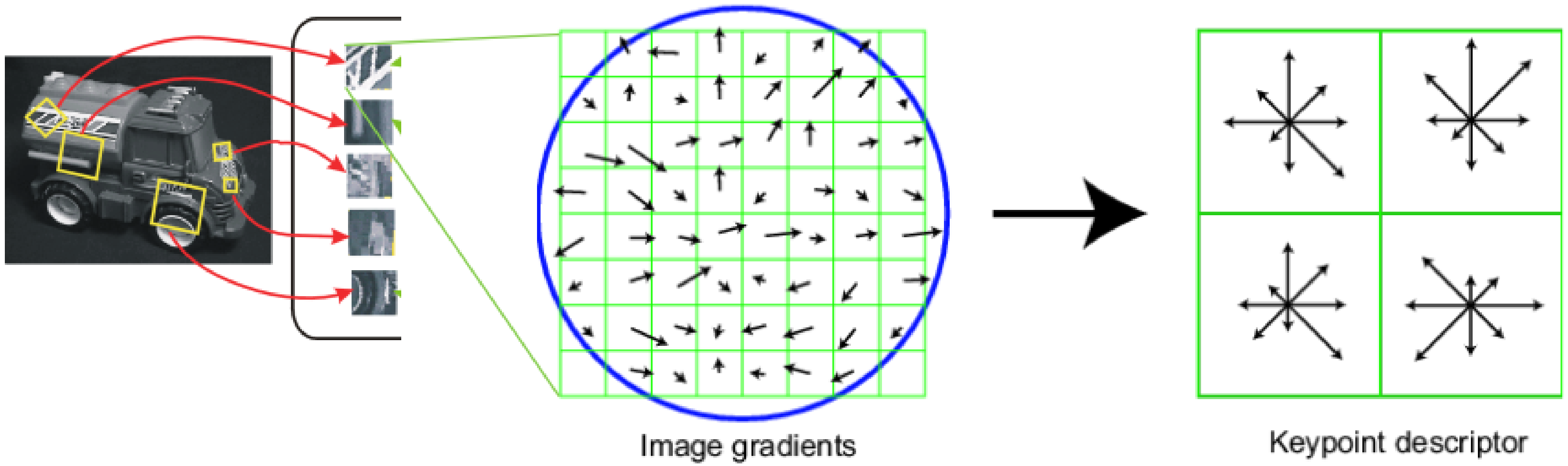


SIFT Descriptor Extraction



Reduce effect of illumination

- 128-dim vector normalized to 1
- Threshold gradient magnitudes to avoid excessive influence of high gradients
  - After normalization, clamp gradients > 0.2
  - Renormalize



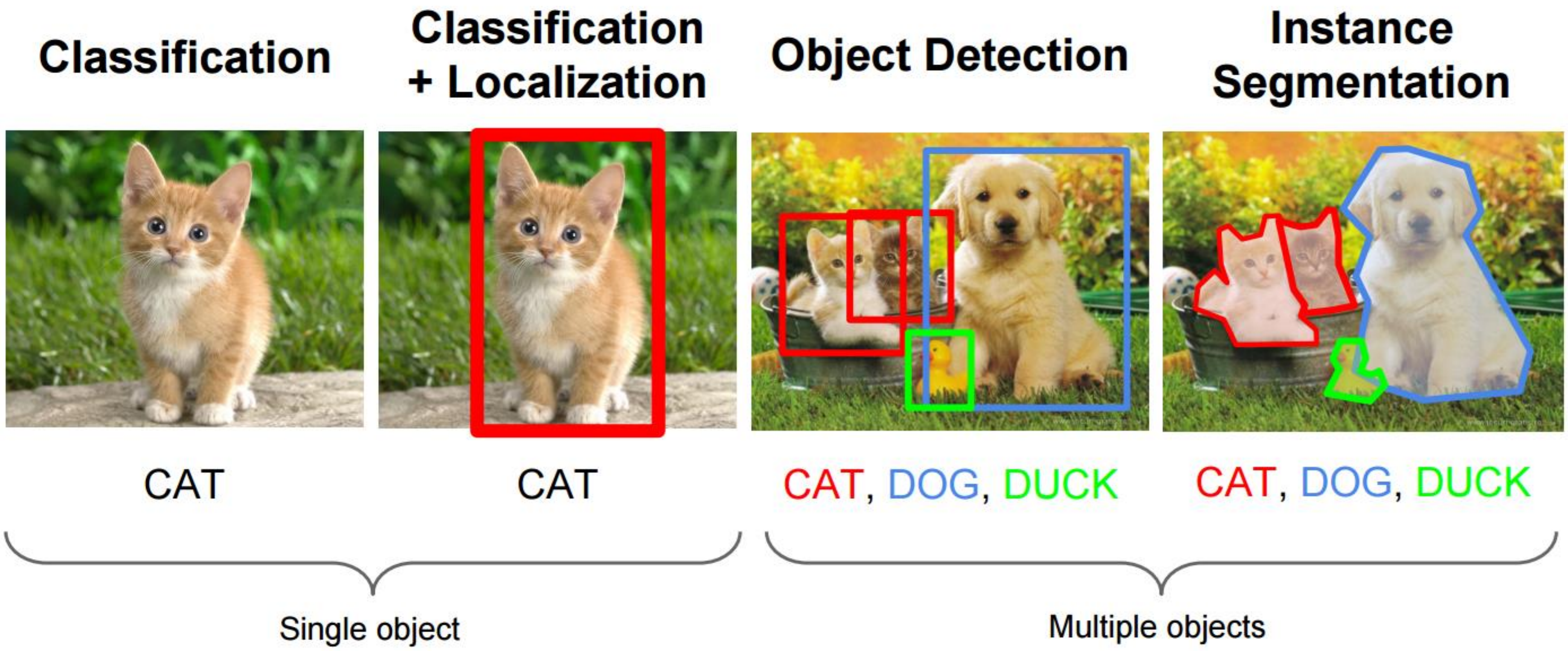
Robot Vision

13. Object detection I

Outline

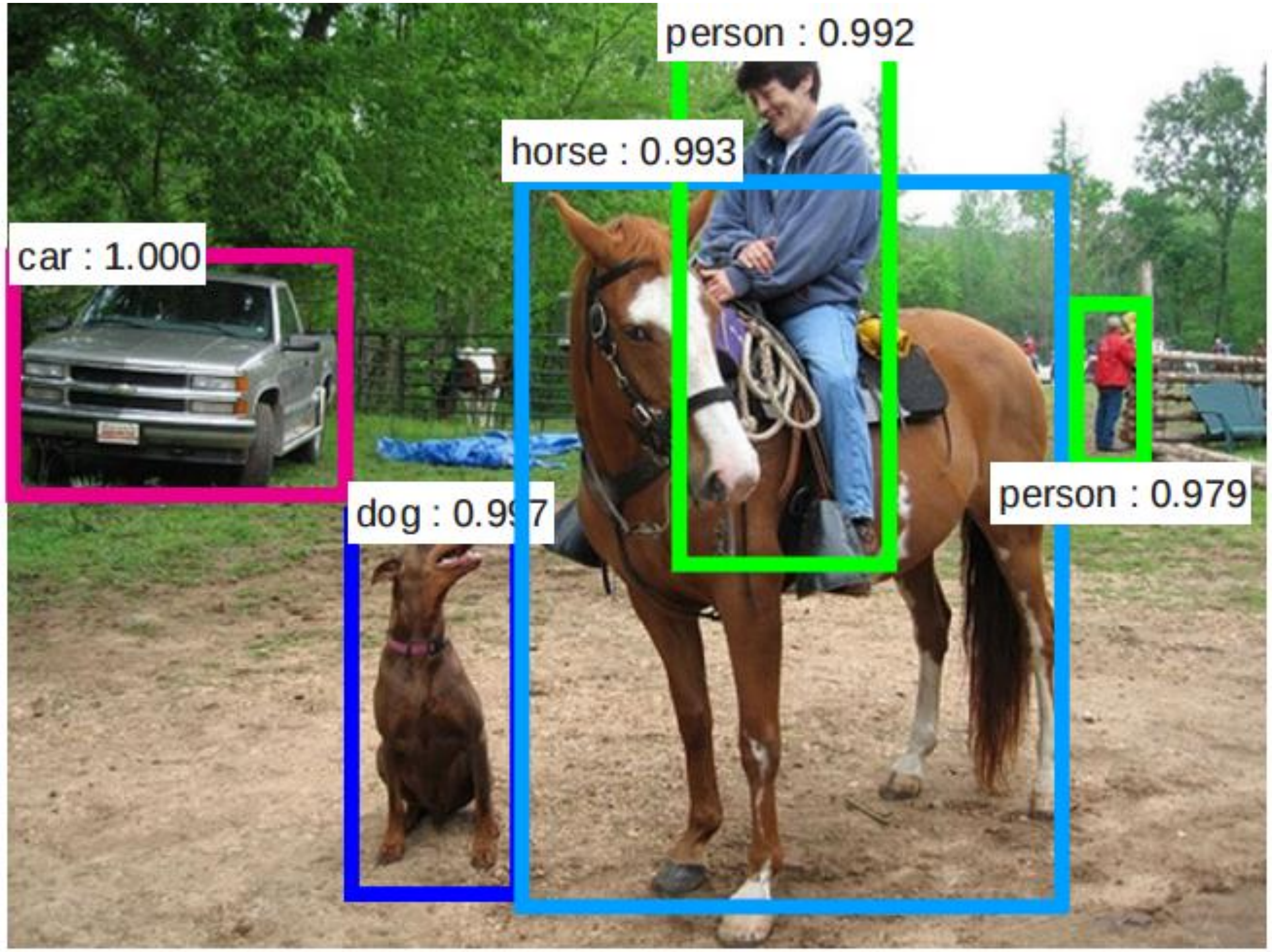
- **Overview: What is Object detection?**
- Top methods for object detection
- Object detection with Sliding Window and Feature Extraction(HoG)
  - Sliding Window technique
  - HOG: Gradient based Features
  - Machine Learning
    - Support Vector Machine (SVM)
  - Non-Maxima Suppression (NMS)
- Implementation examples
- Deformable Part-based Model (DPM)

What is object detection



http://cs231n.stanford.edu/slides/winter1516_lecture8.pdf

Object detection



- Multiple outputs
  - Bounding box
  - Label
  - Score

Detection Competitions

Pascal VOC
COCO
ImageNet ILSVRC

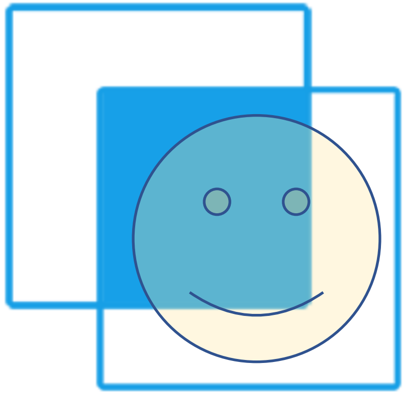
VOC: 20 classes

COCO: 200 classes

http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html#introduction

Valid detection

- Groundtruth:
  - *Bounding box*
  - *Label*
- Possible detection
  - *Bounding box*
  - *Label*
  - score



$$score_{iou} = \frac{Intersected\ Area}{Union\ BB\ area}$$

Different criteria to declare detections:

Pascal criteria

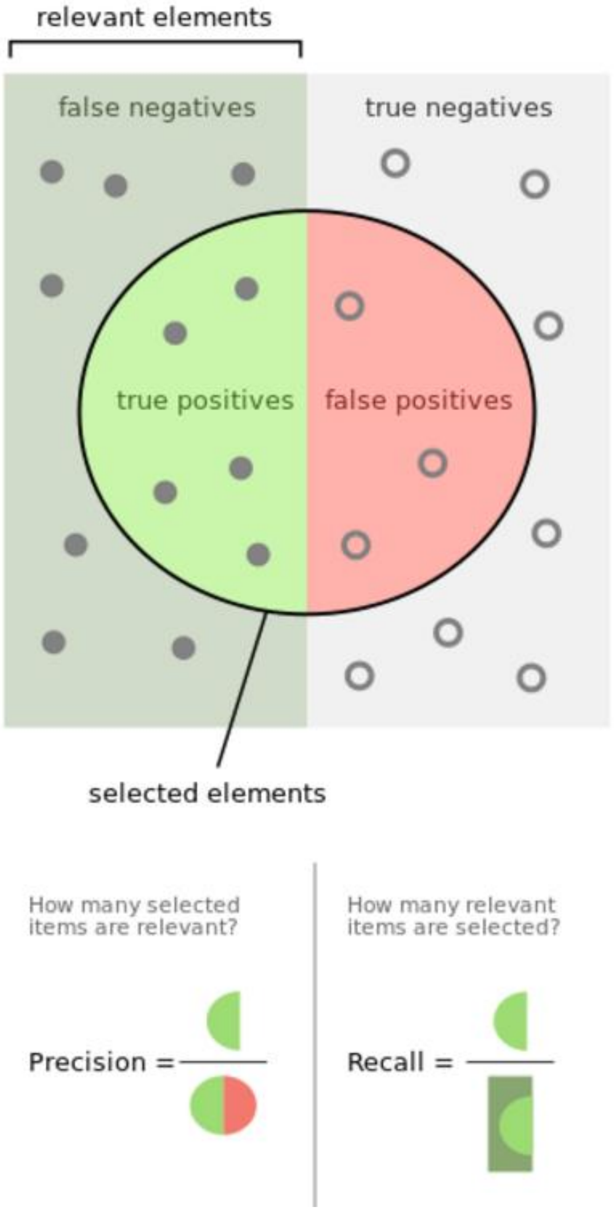
$$score_{iou} > 0.5$$

All of these:

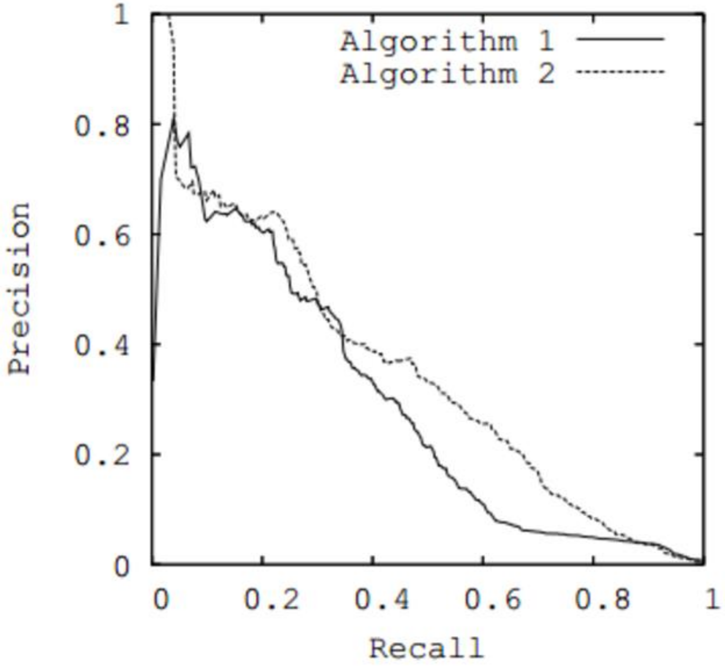
$score_{iou} > 0.5$
$score_{iou} > 0.55$
$score_{iou} > 0.6$
$score_{iou} > 0.65$
$score_{iou} > 0.7$
$score_{iou} > 0.75$
$score_{iou} > 0.8$
$score_{iou} > 0.9$
$score_{iou} > 0.95$

Terms

Recall
Precision
mAP
IoU

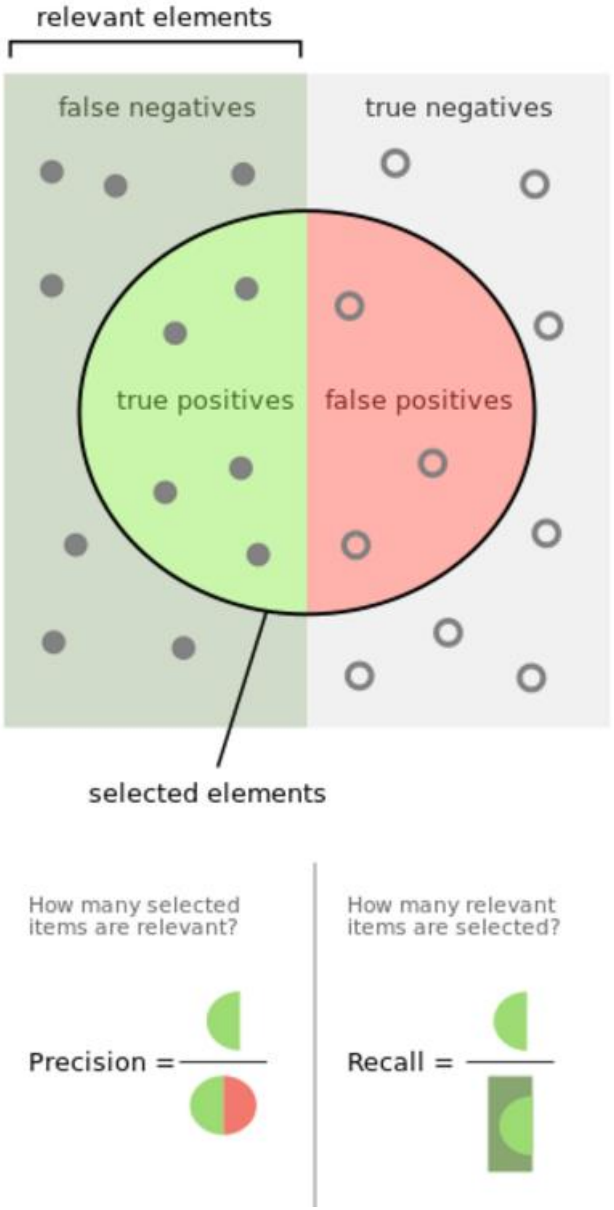


Possible detection
- Bounding box
- Label
- ***score***

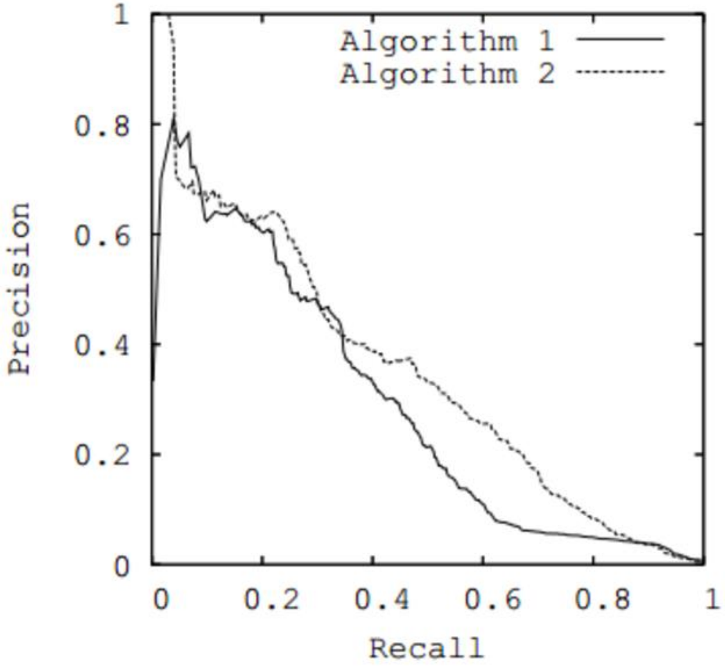


Terms

Recall
Precision
mAP
IoU



Possible detection
- Bounding box
- Label
- ***score***



Average precision (AP): Area under curve

Terms

Possible detection

Bounding box

Label

score

Recall

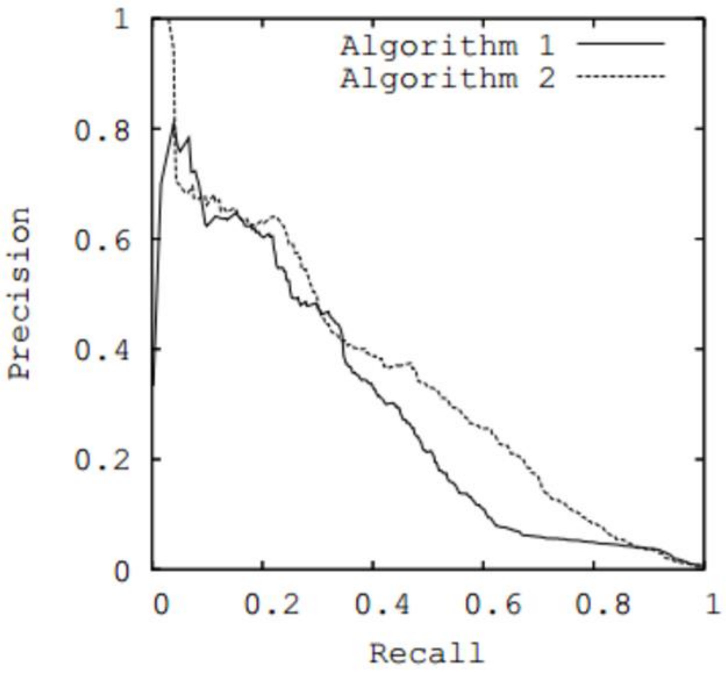
Precision

mAP

IoU

mAP is simply all the AP values averaged over different classes/categories

Box Average Precision (AP@[0.5:0.95]): sums IOUs
between 0.5 and 0.95 and divides the sum by the number of the IOU values



Average precision (AP): Area under curve

Outline

- Overview: What is Object detection?
- **Top methods for object detection**
- Object detection with Sliding Window and Feature Extraction(HoG)
  - Sliding Window technique
  - HOG: Gradient based Features
  - Machine Learning
    - Support Vector Machine (SVM)
  - Non-Maxima Suppression (NMS)
- Implementation examples
- Deformable Part-based Model (DPM)

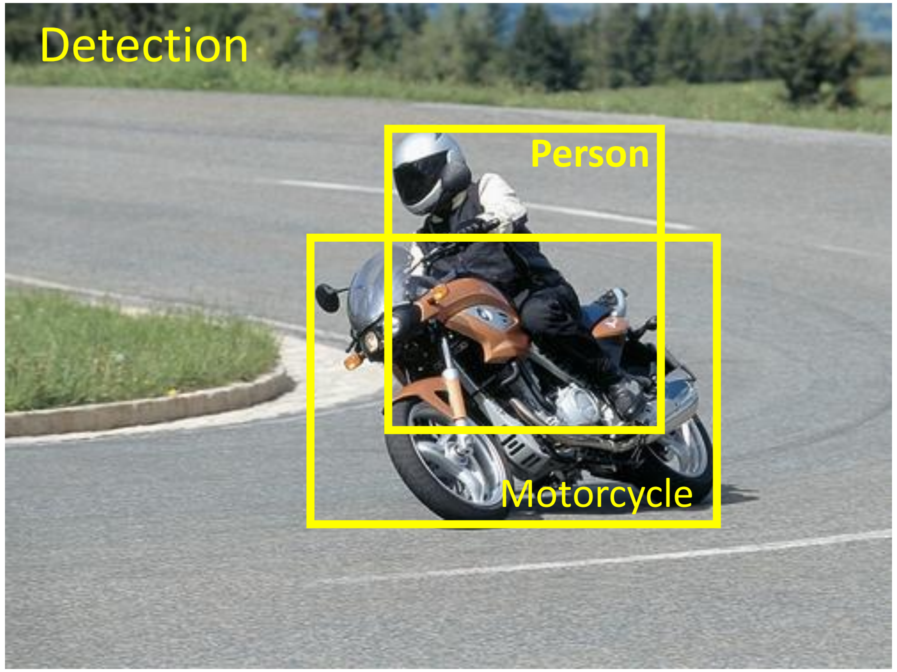
Popular algorithms for object detection

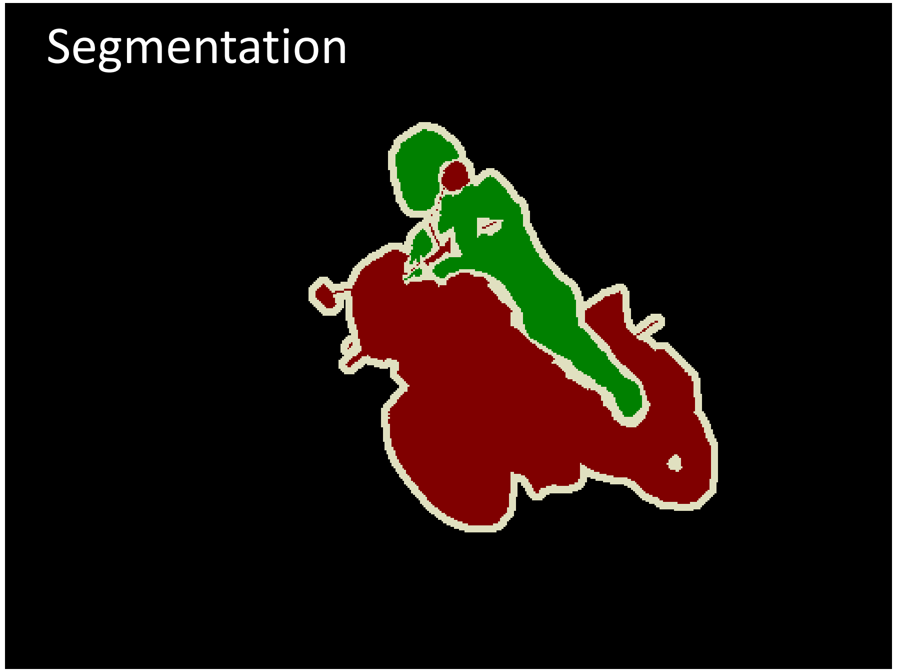
- Pre-DeepLearning
  - HOG + SVM (Dalal, Triggs)
  - Deformable Part-based Model (DPM)
- Deep learning
  - Fast R-CNN
  - Faster R-CNN
  - Region-based Convolutional Neural Networks (R-CNN)
  - Region-based Fully Convolutional Network
  - Single Shot Detector (SSD)
  - YOLO (You Only Look Once)

PASCAL VOC 2005-2012

20 object classes **22,591 images**

Classification: person, motorcycle



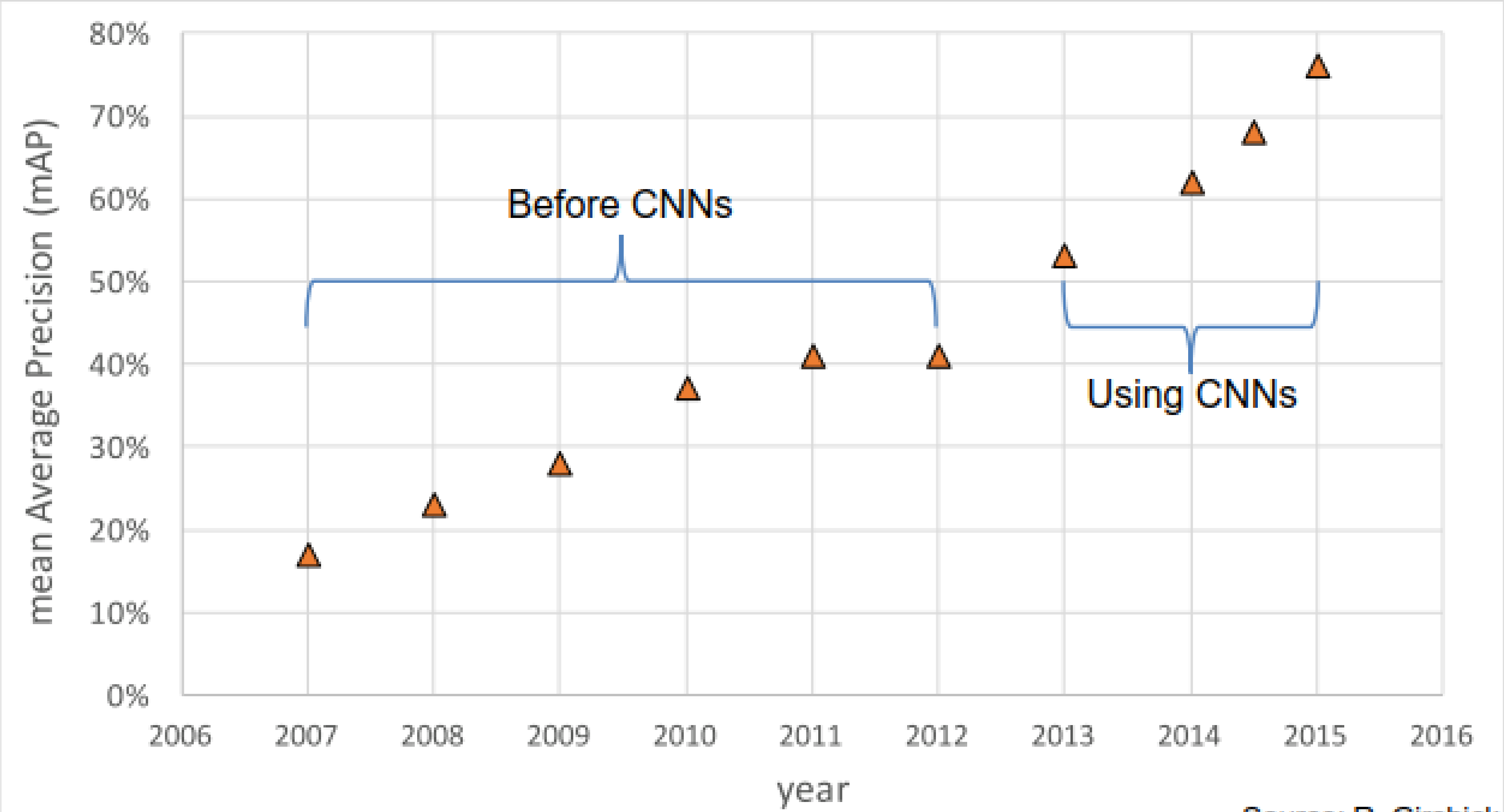


Action: riding bicycle

Everingham, Van Gool, Williams, Winn and Zisserman.
The PASCAL Visual Object Classes (VOC) Challenge. IJCV 2010.

Object detection progress

PASCAL VOC

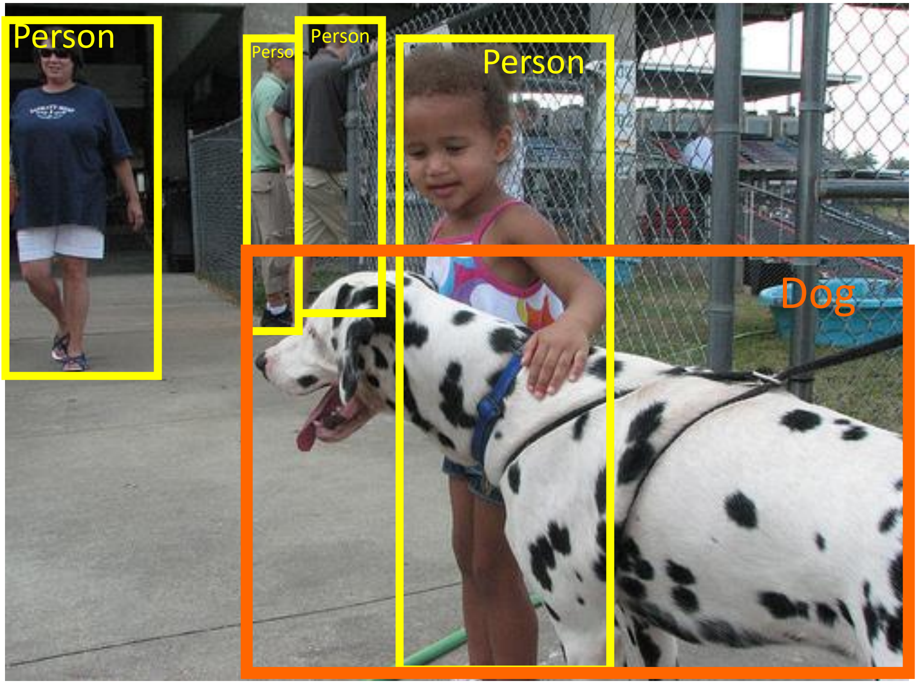


Source: R. Girshick

IMAGENET Large Scale Visual Recognition Challenge (ILSVRC) 2010-2014

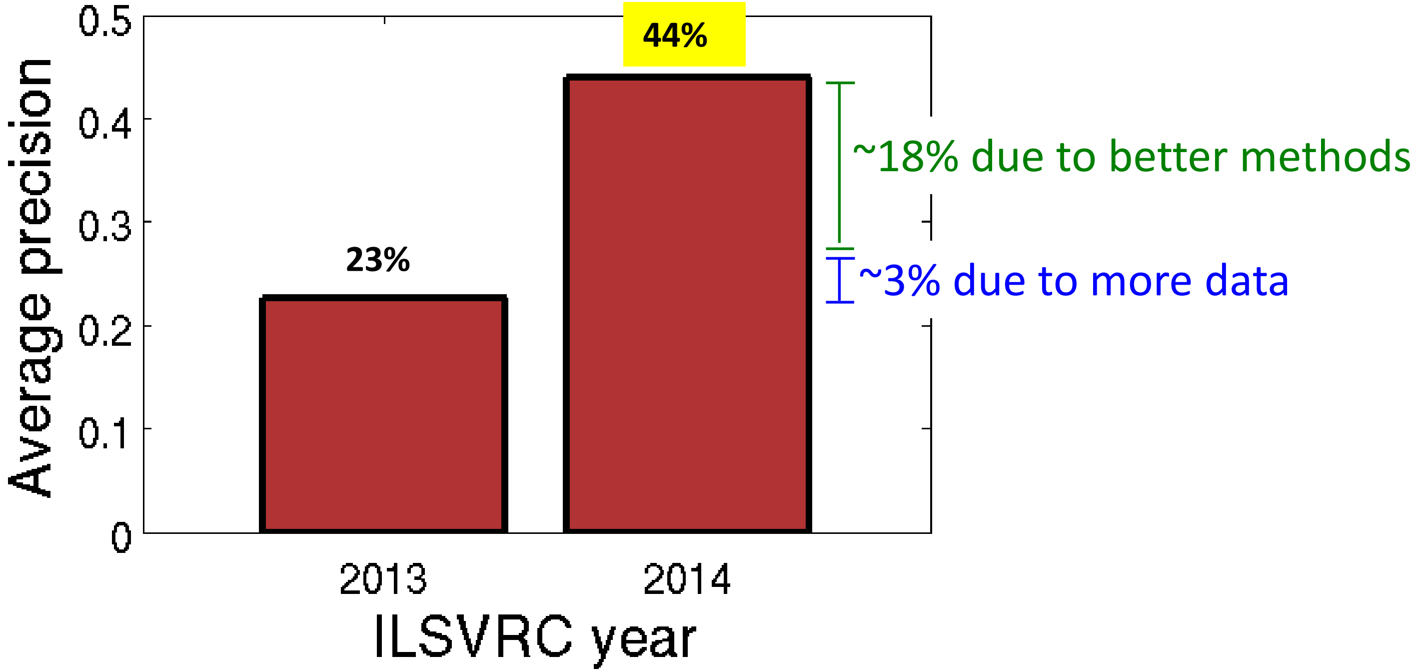
~~20 object classes 22,591 images~~

200 object classes	**517,840 images**	**DET**
1000 object classes	**1,431,167 images**	**CLS-LOC**



http://image-net.org/challenges/LSVRC/

ILSVRC detection in 2014 (Deep learning)



1.9x increase in object detection average precision in one year

Russakovsky* and Deng* et al., ImageNet Large Scale Visual Recognition Challenge, http://arxiv.org/abs/1409.0575

Microsoft COCO: Common Objects in Context

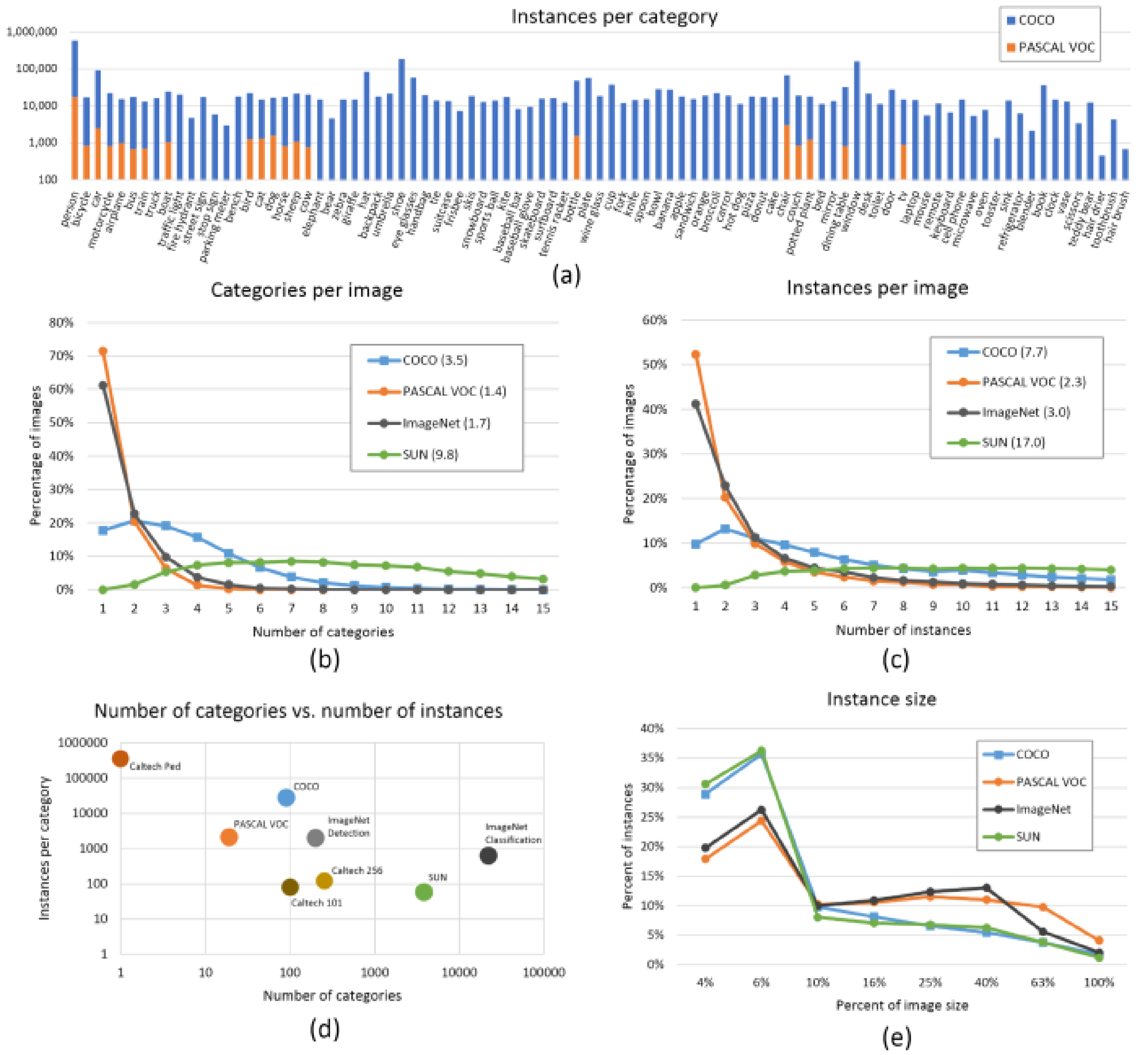
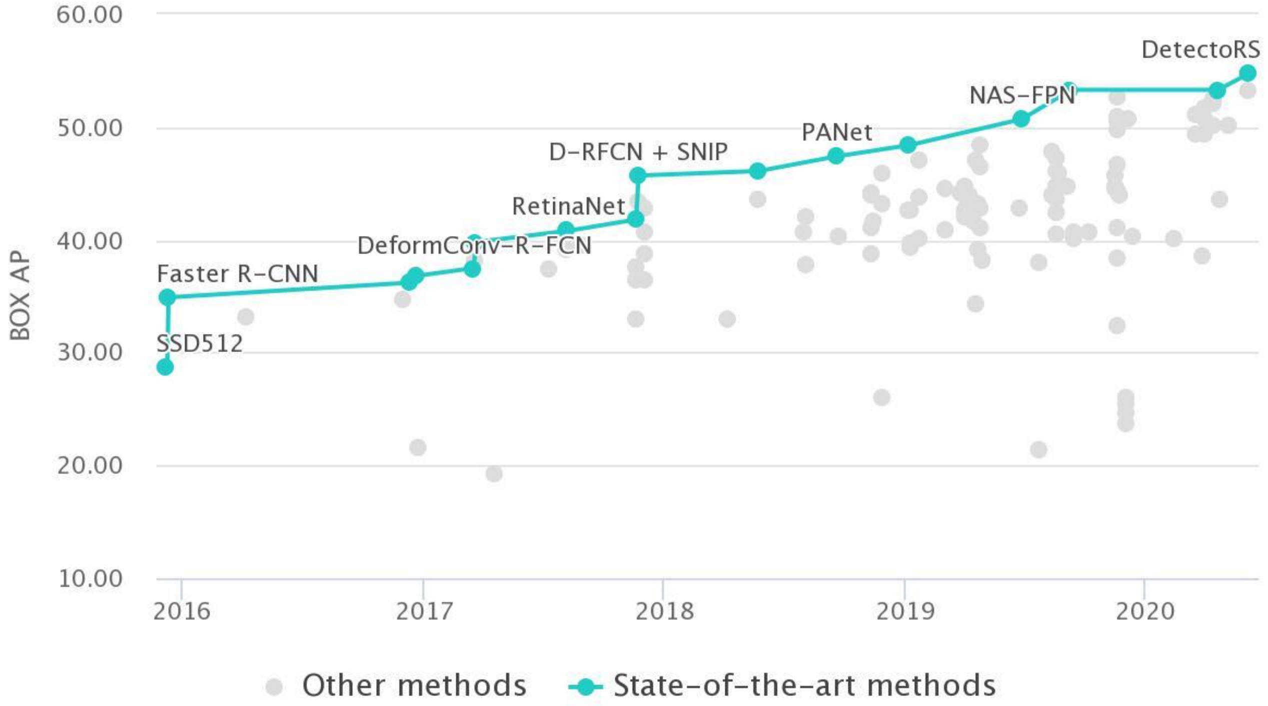


Fig. 5: (a) Number of annotated instances per category for MS COCO and PASCAL VOC. (b,c) Number of annotated categories and annotated instances, respectively, per image for MS COCO, ImageNet Detection, PASCAL VOC and SUN (average number of categories and instances are shown in parentheses). (d) Number of categories vs. the number of instances per category for a number of popular object recognition datasets. (e) The distribution of instance sizes for the MS COCO, ImageNet Detection, PASCAL VOC and SUN datasets.

[COCO - Common Objects in Context (cocodataset.org)](https://cocodataset.org)

State of the art methods

Network models evaluated on COCOtest-dev object detection database (2013-)		
Network model name	box AP	AP75
SSD512 [33]	28.8%	30.3%
RefineDet512(VGG-16) [62]	33.0%	35.5%
YOLO-v4-608 [63]	43.5%	47.0%
Faster R-CNN(LIP-ResNet-101-MD w FPN) [64]	43.9%	48.1%
PP-YOLO [65]	45.2%	49.9%
Cascade Mask R-CNN(ResNeXt152, multi-scale) [66]	53.3%	58.5%
SpineNet-190 [67]	54.3	
DetectoRS(ResNeXt-101-32x4d, multi-scale) [68]	54.7%	60.1%
EfficientDet-D7x(multi-scale) [69]	55.1%	59.9%
CSP-p6 + Mish(multi-scale) [70]	55.2%	60.7%
DetectoRS(ResNeXt-101-64x4d, multi-scale) [68]	55.7%	61.1%



State of the art methods

Do you still need the old methods?

Network models evaluated on COCOtest-dev object detection database (2013-)		
Network model name	box AP	AP75
SSD512 [33]	28.8%	30.3%
RefineDet512(VGG-16) [62]	33.0%	35.5%
YOLO-v4-608 [63]	43.5%	47.0%
Faster R-CNN(LIP-ResNet-101-MD w FPN) [64]	43.9%	48.1%
PP-YOLO [65]	45.2%	49.9%
Cascade Mask R-CNN(ResNeXt152, multi-scale) [66]	53.3%	58.5%
SpineNet-190 [67]	54.3	
DetectoRS(ResNeXt-101-32x4d, multi-scale) [68]	54.7%	60.1%
EfficientDet-D7x(multi-scale) [69]	55.1%	59.9%
CSP-p6 + Mish(multi-scale) [70]	55.2%	60.7%
DetectoRS(ResNeXt-101-64x4d, multi-scale) [68]	55.7%	61.1%

Network models evaluated on COCO real time object detection database (2017-)		
Network model name	mAP	FPS
NAS-FPNLite MobileNetV2 [71]	25.7%	3
YOLOv3-608 [31]	33.0%	20
SSD512-HarDNet85 [72]	35.1%	39.0
Mask R-CNN X-152-32x8d [73]	40.3%	3
YOLOv4-608 [63]	43.5%	62.0
CenterNet HarDNet-85 [72]	43.6%	45.0
SpineNet-49 [74]	45.3%	29.1
NAS-FPN AmoebaNet [71]	48.3%	3.6
EfficientDet-D7x(single-scale) [69]	55.1%	6.5

Outline

- Overview: What is Object detection?
- Top methods for object detection
- **Object detection with Sliding Window and Feature Extraction(HoG)**
  - Sliding Window technique
  - HOG: Gradient based Features
  - Machine Learning
    - Support Vector Machine (SVM)
  - Non-Maximum Suppression (NMS)
- Implementation examples
- Deformable Part-based Model (DPM)

Histograms of Oriented Gradients for Human Detection

Navneet Dalal and Bill Triggs
INRIA Rhône-Alps, 655 avenue de l'Europe, Montbonnot 38334, France
{Navneet.Dalal,Bill.Triggs}@inrialpes.fr, http://lear.inrialpes.fr

Abstract

We study the question of feature sets for robust visual object recognition, adopting linear SVM based human detection as a test case. After reviewing existing edge and gradient based descriptors, we show experimentally that grids of Histograms of Oriented Gradient (HOG) descriptors significantly outperform existing feature sets for human detection. We study the influence of each stage of the computation on performance, concluding that fine-scale gradients, fine orientation binning, relatively coarse spatial binning, and high-quality local contrast normalization in overlapping descriptor blocks are all important for good results. The new approach gives near-perfect separation on the original MIT pedestrian database, so we introduce a more challenging dataset containing over 1800 annotated human images with a large range of pose variations and backgrounds.

1 Introduction

We briefly discuss previous work on human detection in §2, give an overview of our method §3, describe our data sets in §4 and give a detailed description and experimental evaluation of each stage of the process in §5–6. The main conclusions are summarized in §7.

2 Previous Work

There is an extensive literature on object detection, but here we mention just a few relevant papers on human detection [18, 17, 22, 16, 20]. See [6] for a survey. Papageorgiou *et al* [18] describe a pedestrian detector based on a polynomial SVM using rectified Haar wavelets as input descriptors, with a parts (subwindow) based variant in [17]. Depoortere *et al* give an optimized version of this [2]. Gavrila & Philomen [8] take a more direct approach, extracting edge images and matching them to a set of learned exemplars using chamfer distance. This has been used in a practical real-time pedestrian detection system [7]. Viola *et al* [22] build an efficient

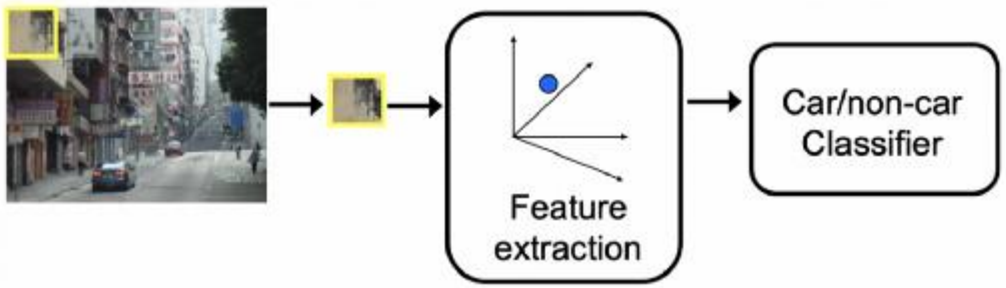
- CVPR 2005

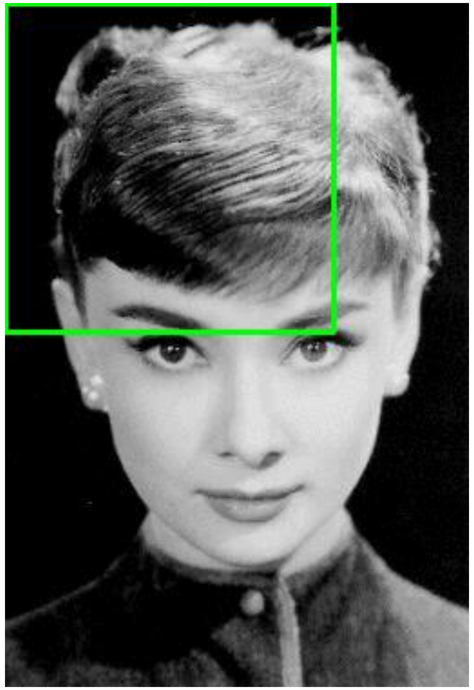
Sliding Window Technique

- Classification problem:
  - Score for a category

Sliding Window Technique

- Score every subwindow
  - extract features from the image window
  - classifier decides based on the given features.
- It is a brute-force approach





Window-based detection: strengths

Pros

- Sliding window detection and global appearance descriptors:
  - Simple detection protocol to implement
  - Good feature choices critical
  - Past successes for certain classes

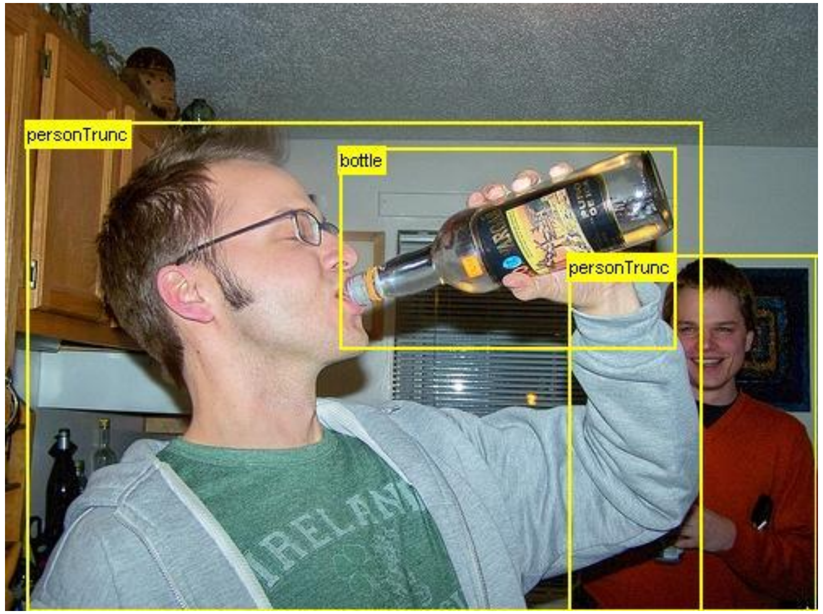
Cons

- High computational complexity
  - For example: 250,000 locations x 30 orientations x 4 scales = 30,000,000 evaluations!
  - If training binary detectors independently, means cost increases linearly with number of classes
- With so many windows, false positive rate better be low

Slide: Kristen Grauman

Cons (continued)

- Not all objects are “box” shaped

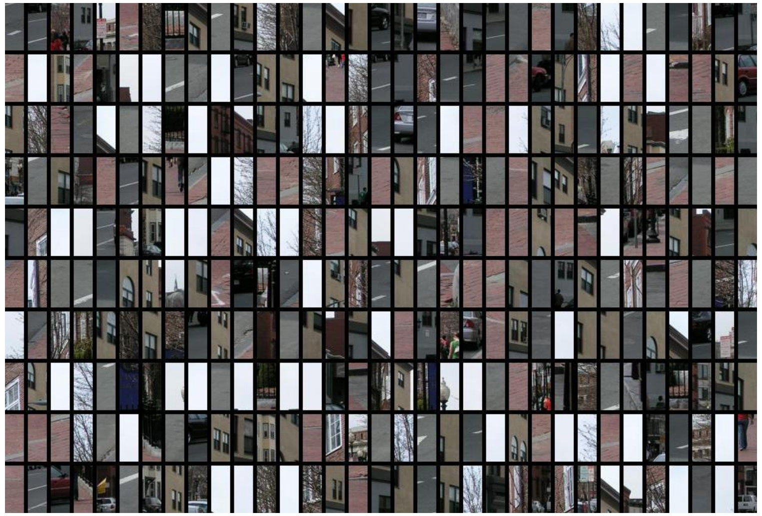


Slide: Kristen Grauman

Limitations (continued)

- If considering windows in isolation, context is lost

Sliding window



Detector's view

Figure credit: Derek Hoiem

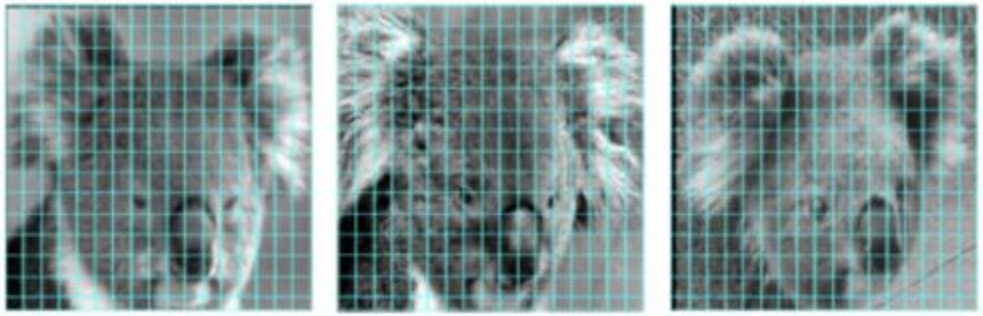
Slide: Kristen Grauman

Outline

- Overview: What is Object detection?
- Top methods for object detection
- **Object detection with Sliding Window and Feature Extraction(HoG)**
  - Sliding Window technique
  - **HOG: Gradient based Features**
  - Machine Learning
    - Support Vector Machine (SVM)
  - Non-Maximum Suppression (NMS)
- Implementation examples
- Deformable Part-based Model (DPM)

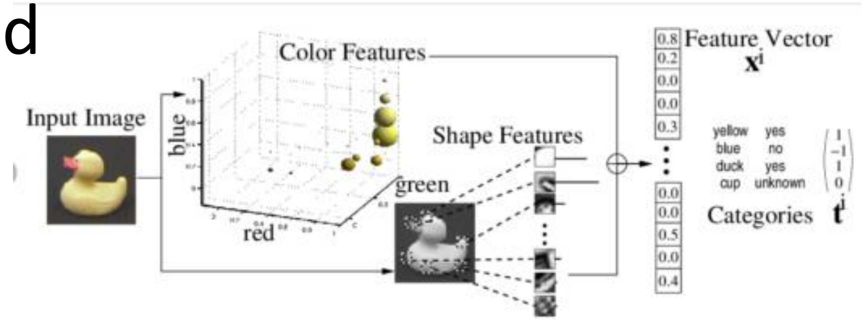
Let’s examine possible feature vectors

- Pixel based (as a vector)
  - Sensitive to small shifts



(1) (2) (3)

- Color based



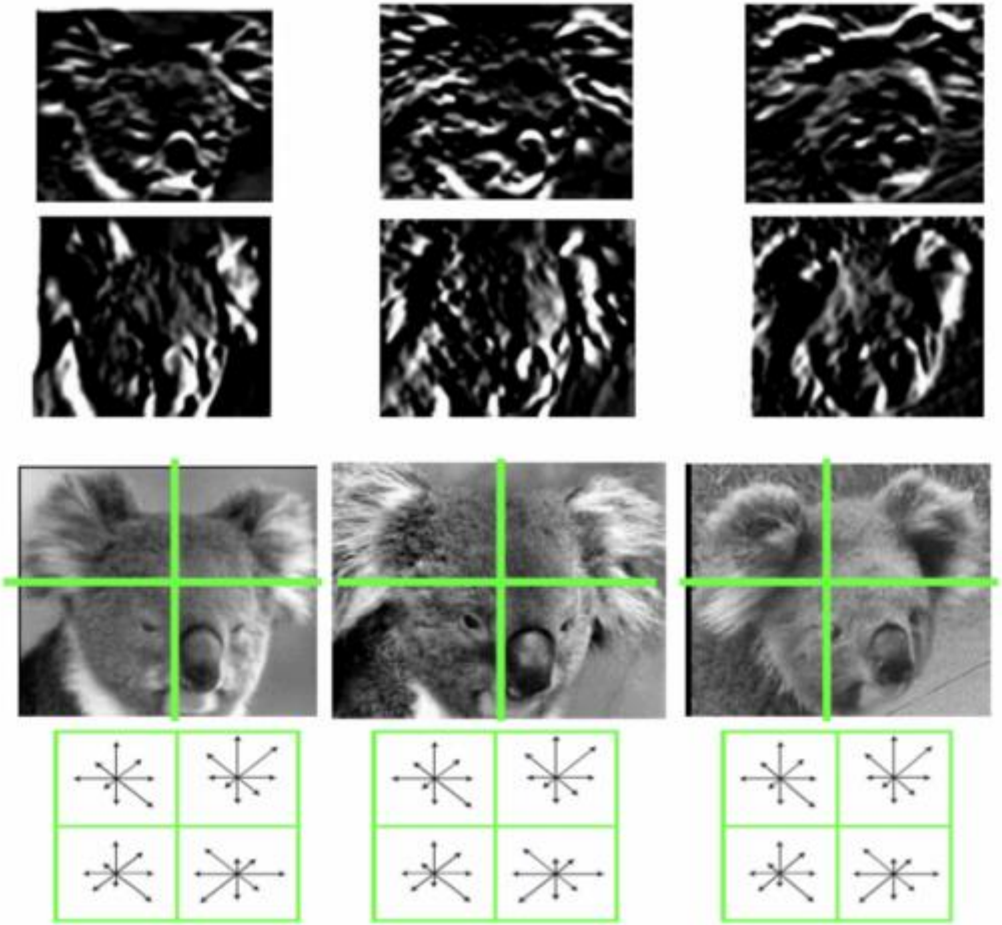
- color-based representations are sensitive to color (illumination)

(1)

(2)

Gradient-based representations

- summarize the local distribution of gradients with histograms
- invariance to small shifts and rotations
- offers more spatial information compared to a single global histogram
- Includes contrast normalization
  - reduce the impact of variable illumination (color)



Histograms of Oriented Gradients (HOG)

- Step 1: Extract a square window (called "block") of some size around the pixel location of interest.
- Step 2: Divide block into a square grid of sub-blocks (called "cells") (2x2 grid in our example, resulting in four cells).
- Step 3: Compute orientation histogram of each cell.
- Step 4: Concatenate the four histograms.
- Step 5: normalize v using one of the three options:
  - Option 1 (L2): Divide v by its Euclidean norm.
  - Option 2 (L1): Divide v by its L1 norm (the L1 norm is the sum of all absolute values of v).
  - Option 3 (L2-Hys):
    - Divide v by its Euclidean norm.
    - In the resulting vector, clip any value over 0.2
    - Then, renormalize the resulting vector by dividing again by its Euclidean norm

Histogram of Oriented Gradients (HOG)

- Angles range from 0 to 180 or from 0 to 360 degrees?
  - In the Dalal & Triggs paper, a range of 0 to 180 degrees is used
- Number of orientation bins.
  - Usually 9 bins, each bin covering 20 degrees.
- Cell size.
  - Cells of size 8x8 pixels are often used. (64 → 9)
- Block size.
  - Blocks of size 2x2 cells (16x16 pixels) are often used.
- HOG feature has 36 dimensions.
  - 4 cells * 9 orientation bins.

Calculate HOG Descriptor vector

- The 16×16 window then moves by 8 pixels and a normalized 36×1 vector is calculated over this window and the process is repeated for the image
- To calculate the final feature vector for the entire image patch, the 36×1 vectors are concatenated into one giant vector.
- Example: an input picture of size 64×64
  - The 16×16 block has 7 positions horizontally and 7 position vertically.
  - In one 16×16 block we have 4 histograms which after normalization concatenate to form a 36×1 vector.
  - This block moves 7 positions horizontally and vertically totalling it to 7×7 = 49 positions.
  - we concatenate them all into one gaint vector we obtain a 36×49 = 1764 dimensional vector.

Outline

- Overview: What is Object detection?
- Top methods for object detection
- **Object detection with Sliding Window and Feature Extraction(HoG)**
  - Sliding Window technique
  - HOG: Gradient based Features
  - **Machine Learning**
    - **Support Vector Machine (SVM)**
  - Non-Maximum Suppression (NMS)
- Implementation examples
- Deformable Part-based Model (DPM)

Support vector machines

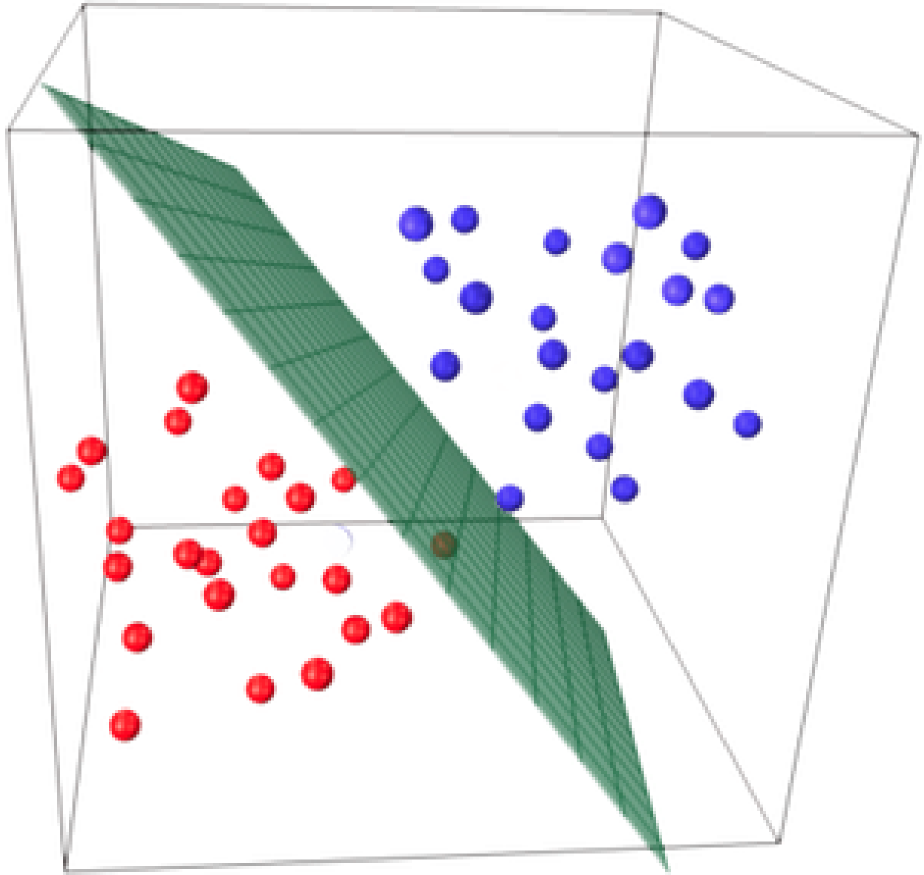
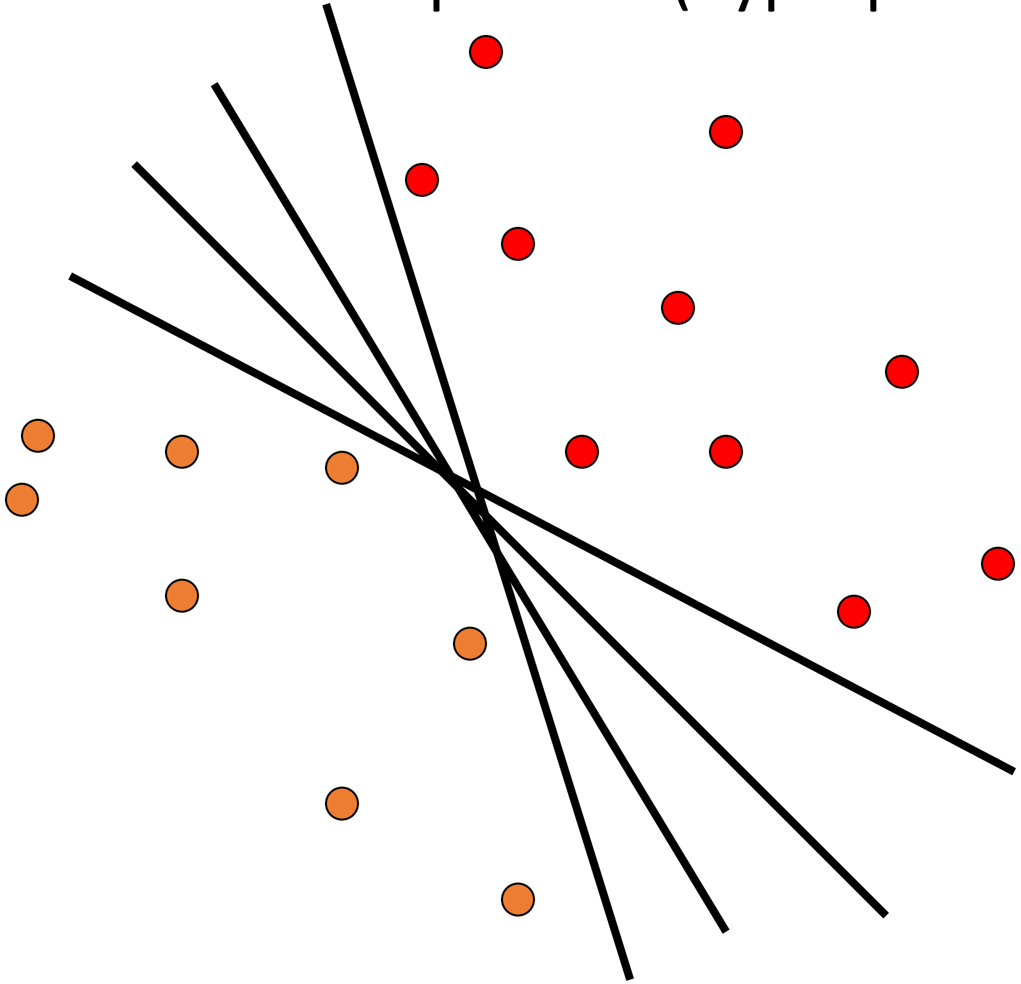


Image source

Support vector machines

- When the data is linearly separable, there may be more than one separator (hyperplane)

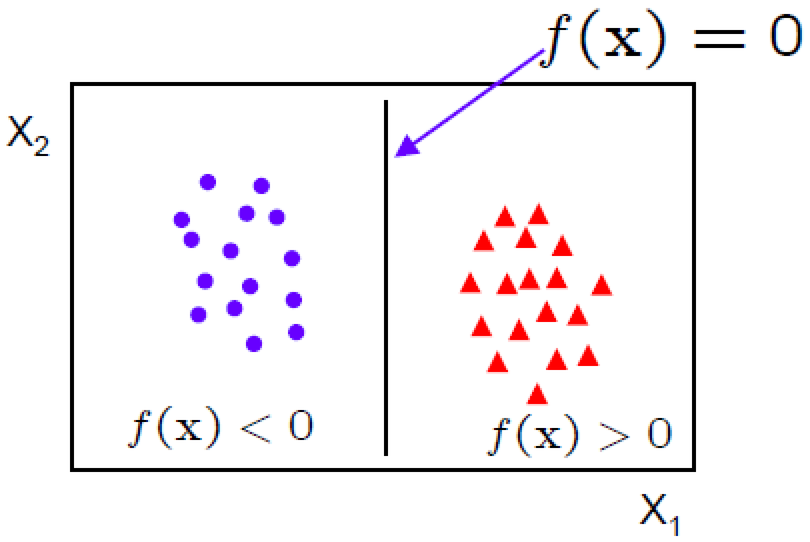


Which separator is best?

Linear classifiers

A linear classifier has the form

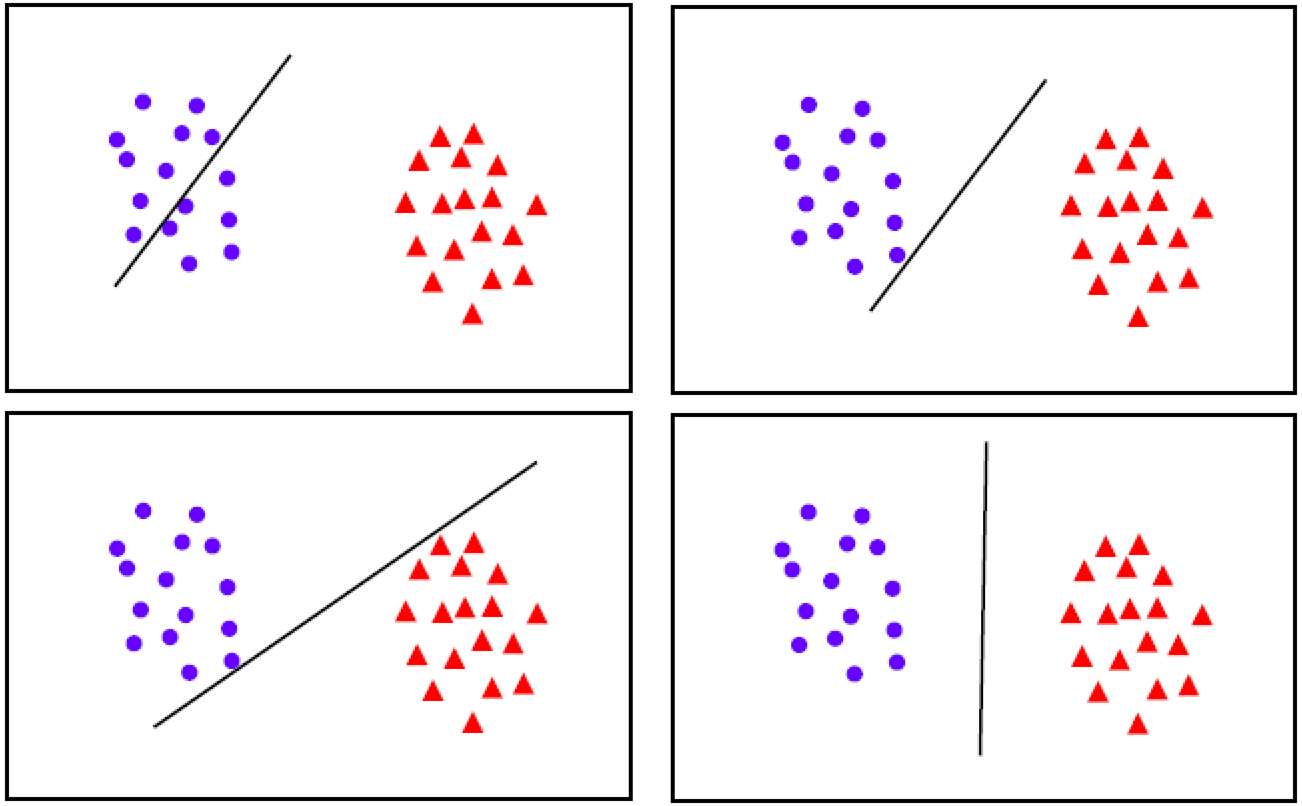
$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$



- in 2D the discriminant is a line
- $\mathbf{w}$ is the normal to the line, and b the bias
- $\mathbf{w}$ is known as the weight vector

Slide from: Lecture 2: The SVM classifier
C19 Machine Learning Hilary 2013 A. Zisserman

What is the best w?

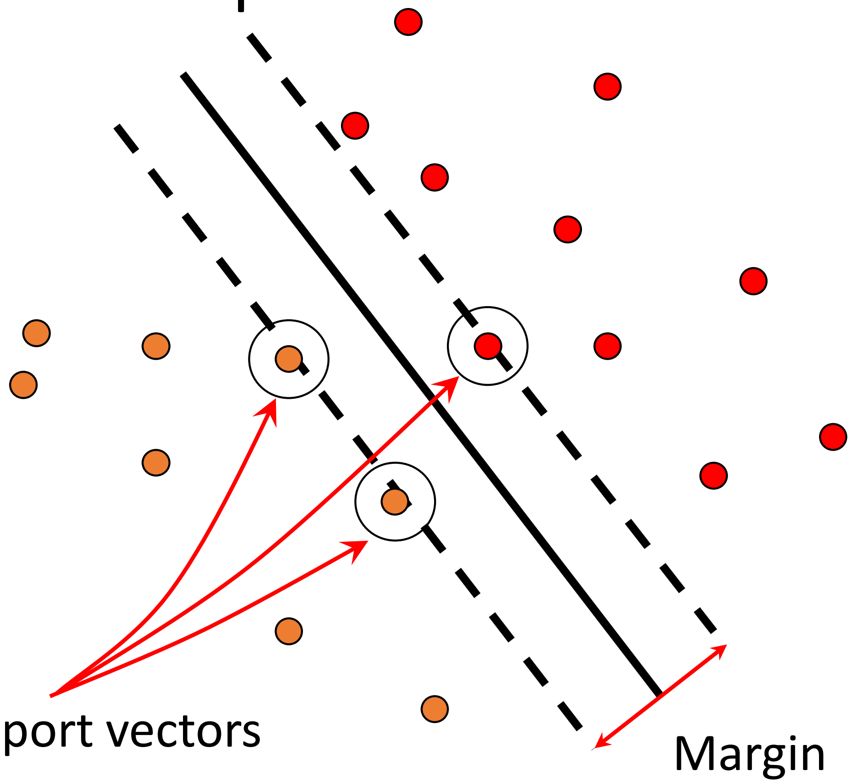


- maximum margin solution: most stable under perturbations of the inputs

Slide from: Lecture 2: The SVM classifier
C19 Machine Learning Hilary 2013 A. Zisserman

Support vector machines

- Find hyperplane that maximizes the *margin* between the positive and negative examples



$$\mathbf{x} \text{ positive } (y = 1): \quad \mathbf{x} \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x} \text{ negative } (y = -1): \quad \mathbf{x} \cdot \mathbf{w} + b \leq -1$$

For support vectors, $\mathbf{x} \cdot \mathbf{w} + b = \pm 1$

Distance between point and hyperplane: $\dfrac{|\mathbf{x} \cdot \mathbf{w} + b|}{\|\mathbf{w}\|}$

Therefore, the margin is $2 / \|\mathbf{w}\|$

C. Burges, [A Tutorial on Support Vector Machines for Pattern Recognition](), Data Mining and Knowledge Discovery, 1998

Finding the maximum margin hyperplane

1. Maximize margin $2 / \|\mathbf{w}\|$
2. Correctly classify all training data:

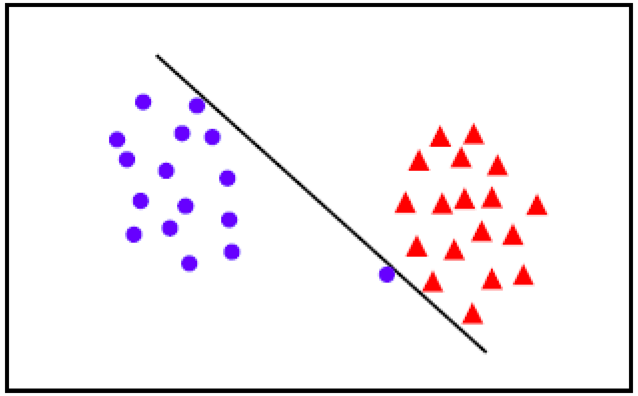
$$\mathbf{x}_i \text{ positive } (y_i = 1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative } (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

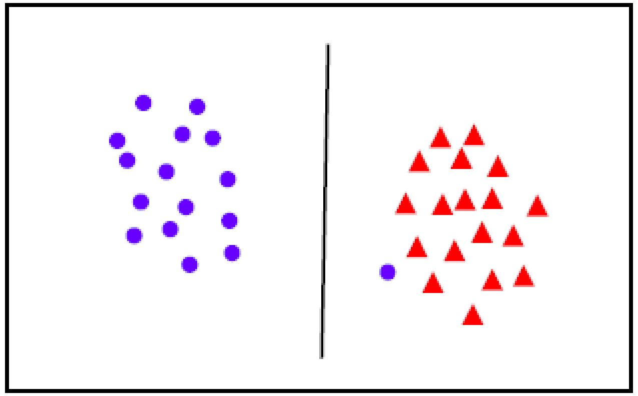
- *Quadratic optimization problem*:
- $$\boxed{\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1}$$

C. Burges, [A Tutorial on Support Vector Machines for Pattern Recognition](), Data Mining and Knowledge Discovery, 1998

Linear separability again: What is the best w?



• the points can be linearly separated but there is a very narrow margin



• but possibly the large margin solution is better, even though one constraint is violated

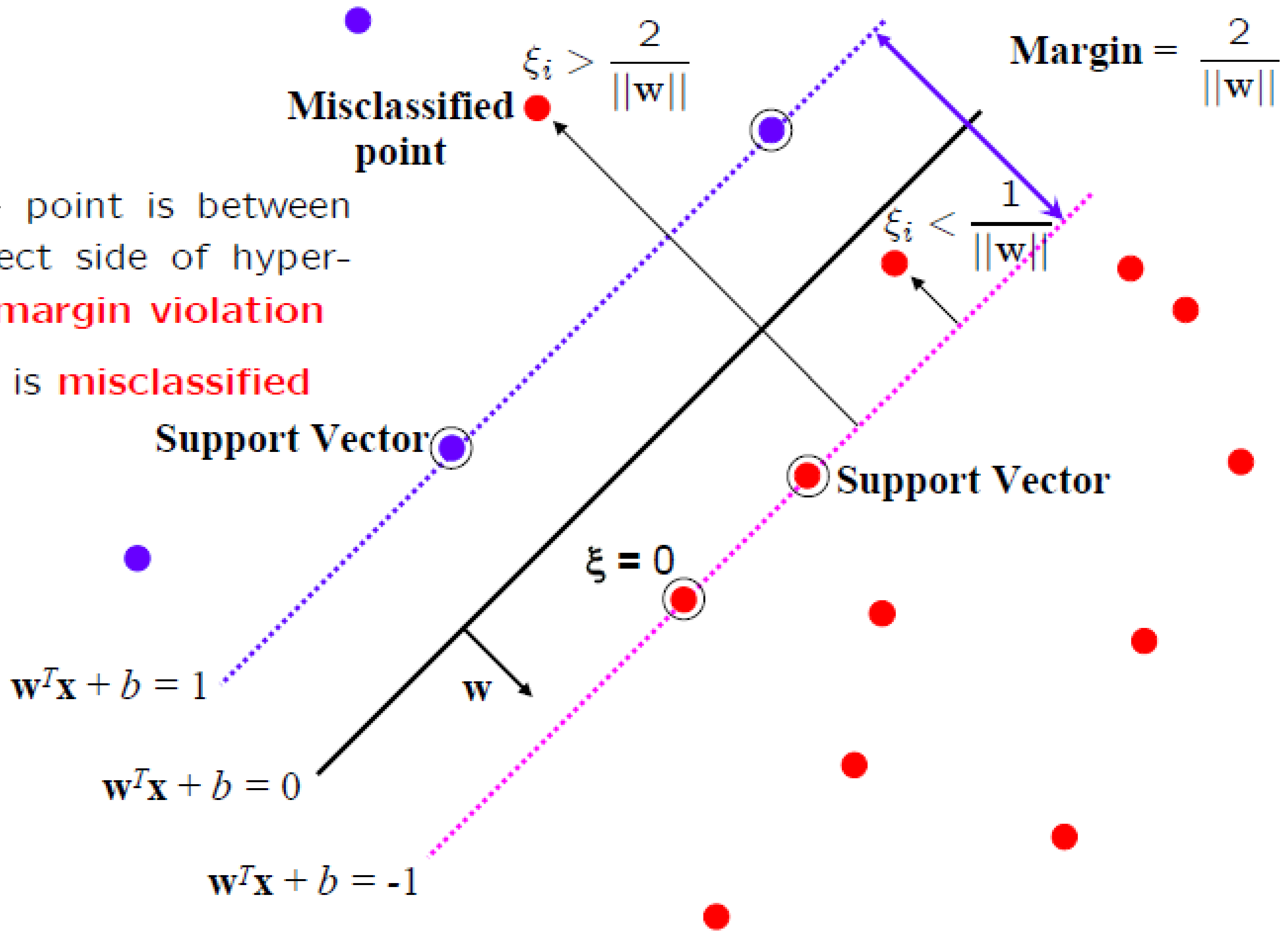
In general there is a trade off between the margin and the number of mistakes on the training data

Slide from: Lecture 2: The SVM classifier
C19 Machine Learning Hilary 2013 A. Zisserman

Introduce "slack" variables

$$\xi_i \geq 0$$

- for $0 < \xi \leq \frac{1}{||\mathbf{w}||}$ point is between margin and correct side of hyper-plane. This is a margin violation
- for $\xi > \frac{1}{||\mathbf{w}||}$ point is misclassified



Slide from: Lecture 2: The SVM classifier
C19 Machine Learning Hilary 2013 A. Zisserman

SVM training in general

- Separable data:

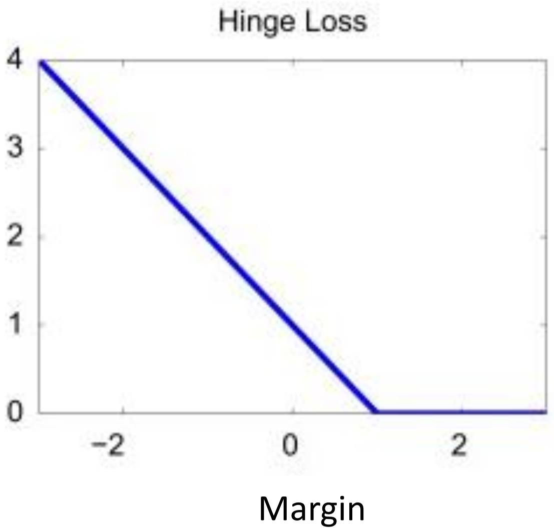
$$\underbrace{\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2}_{\text{Maximize margin}} \quad \underbrace{\text{subject to} \quad y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1}_{\text{Classify training data correctly}}$$

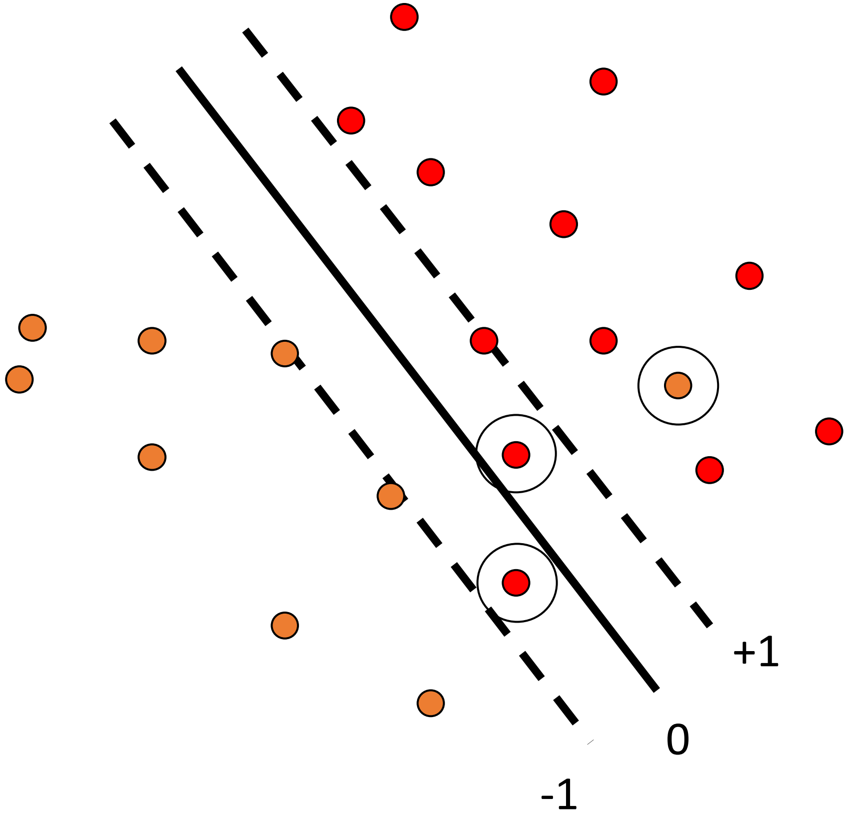
- Non-separable data:

$$\min_{\mathbf{w},b} \quad \underbrace{\frac{1}{2}\|\mathbf{w}\|^2}_{\text{Maximize margin}} + C \underbrace{\sum_{i=1}^{n} \max\left(0, 1 - y_i(\mathbf{w}\cdot\mathbf{x}_i + b)\right)}_{\text{Minimize classification mistakes}}$$

SVM training in general

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\max\left(0, 1 - y_i(\mathbf{w}\cdot\mathbf{x}_i + b)\right)$$

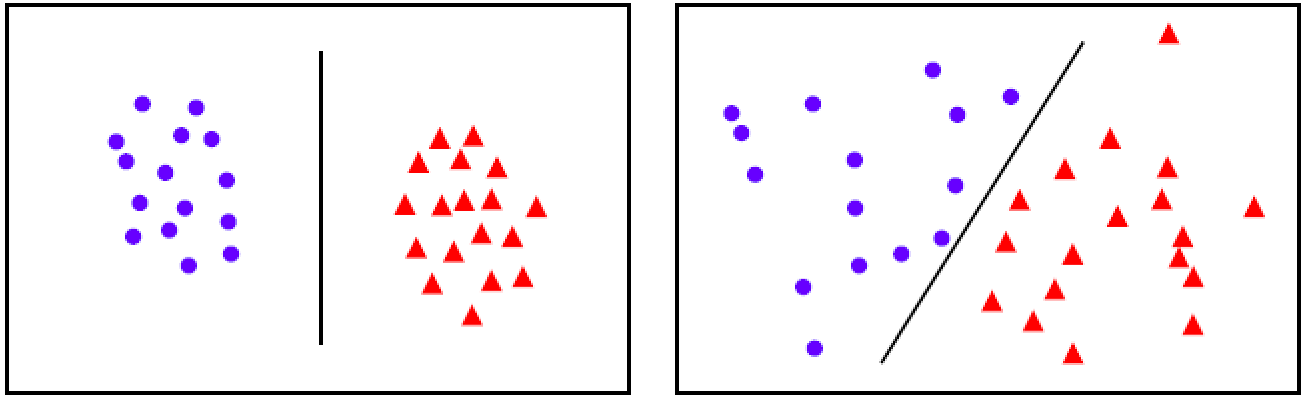




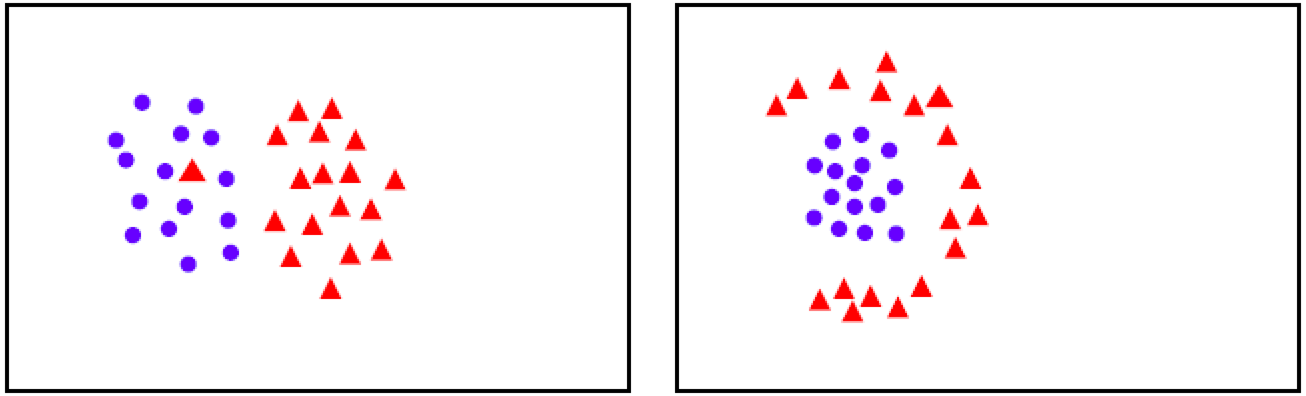
- Demo: http://cs.stanford.edu/people/karpathy/svmjs/demo

Linear separability

linearly
separable



not
linearly
separable



Slide from: Lecture 2: The SVM classifier
C19 Machine Learning Hilary 2013 A. Zisserman

Nonlinear SVMs

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

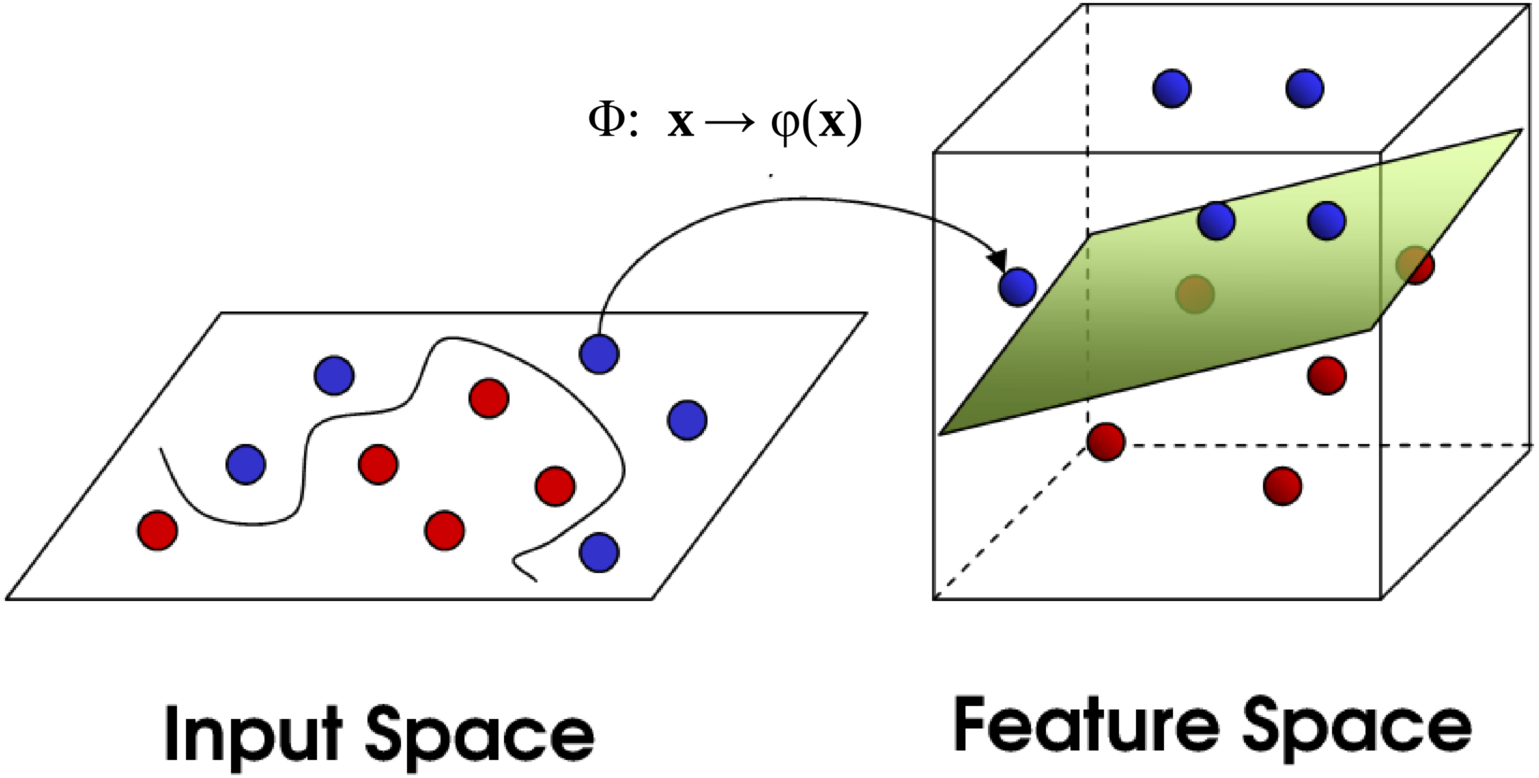
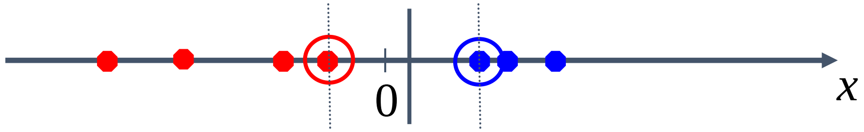


Image source

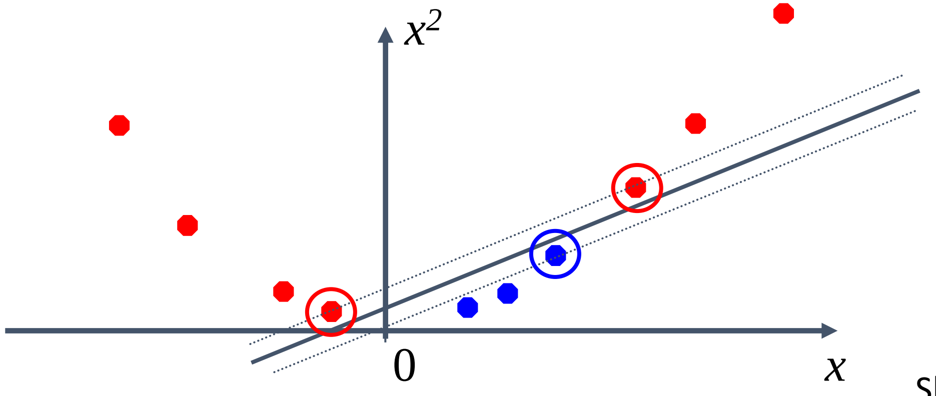
Nonlinear SVMs

- Linearly separable dataset in 1D:



- Non-separable dataset in 1D:

- We can map the data to a *higher-dimensional space*:



Slide credit: Andrew Moore

The kernel trick

- Linear SVM decision function:

$$\mathbf{w}\cdot\mathbf{x}+b=\sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} \;+b$$

learned weight

Support vector

C. Burges, [A Tutorial on Support Vector Machines for Pattern Recognition](), Data Mining and Knowledge Discovery, 1998

The kernel trick

- Linear SVM decision function:

$$\mathbf{w}\cdot\mathbf{x}+b=\sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} \ +b$$

- Kernel SVM decision function:

$$\sum_i \alpha_i y_i \varphi(\mathbf{x}_i)\cdot\varphi(\mathbf{x}) \ +b=\sum_i \alpha_i y_i K(\mathbf{x}_i,\mathbf{x}) \ +b$$

- This gives a nonlinear decision boundary in the original feature space

C. Burges, A Tutorial on Support Vector Machines for Pattern Recognition, Data Mining and Knowledge Discovery, 1998

The kernel trick

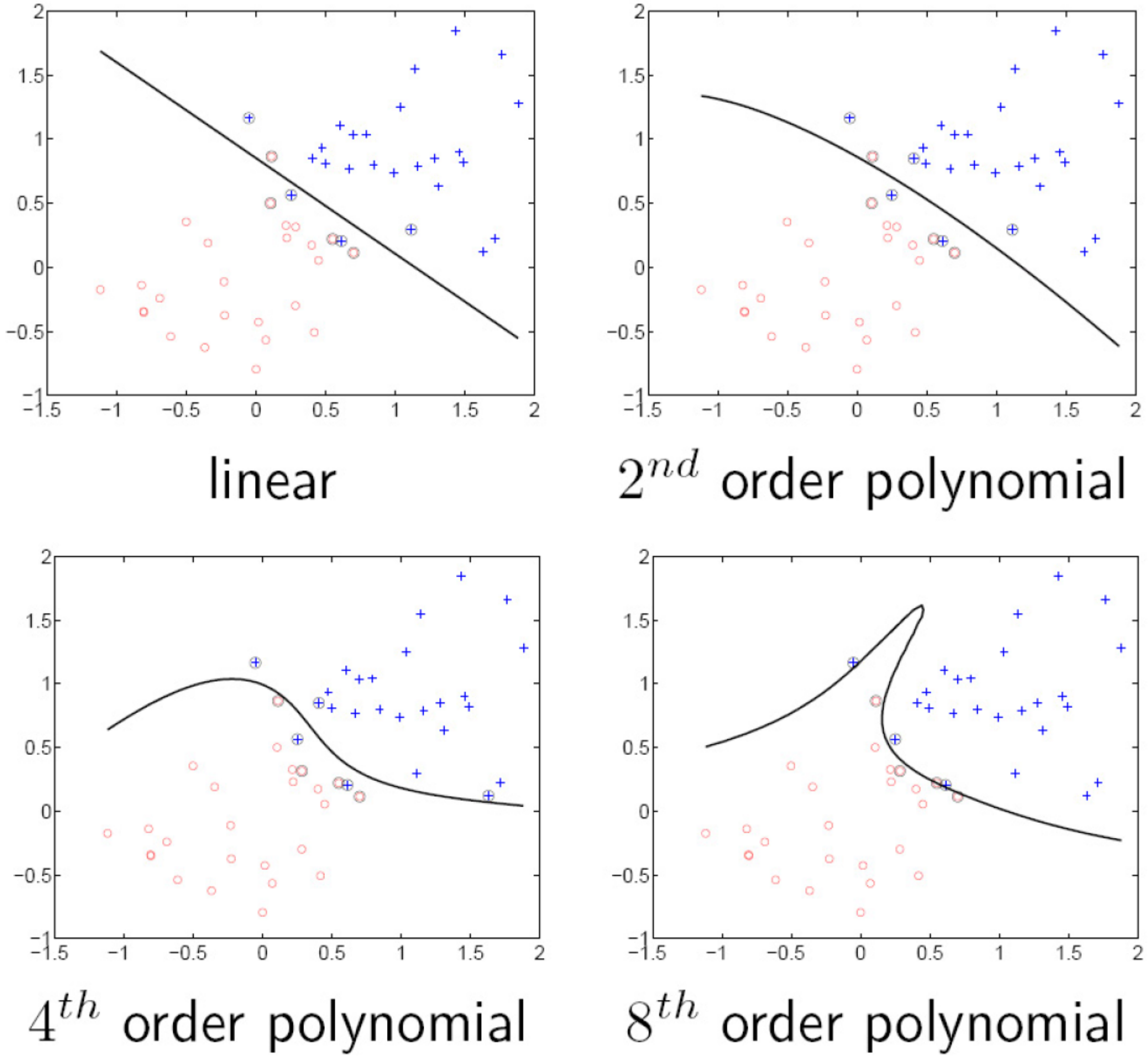
- Instead of explicitly computing the lifting transformation $\boldsymbol{\varphi}(\mathbf{x})$, define a kernel function K such that

$$K(\mathbf{x},\mathbf{y}) = \boldsymbol{\varphi}(\mathbf{x}) \cdot \boldsymbol{\varphi}(\mathbf{y})$$

- (to be valid, the kernel function must satisfy *Mercer's condition*)

Polynomial kernel:

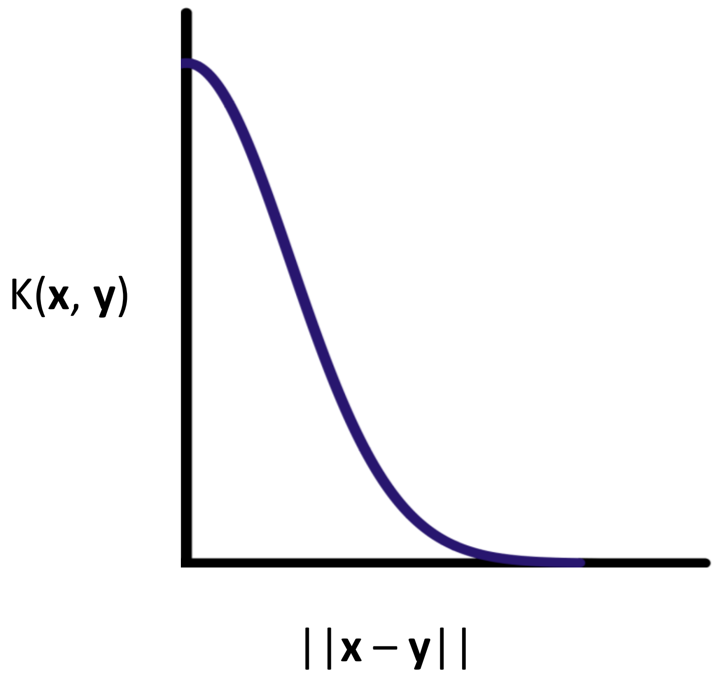
$$K(\mathbf{x},\mathbf{y}) = (c + \mathbf{x}\cdot\mathbf{y})^d$$



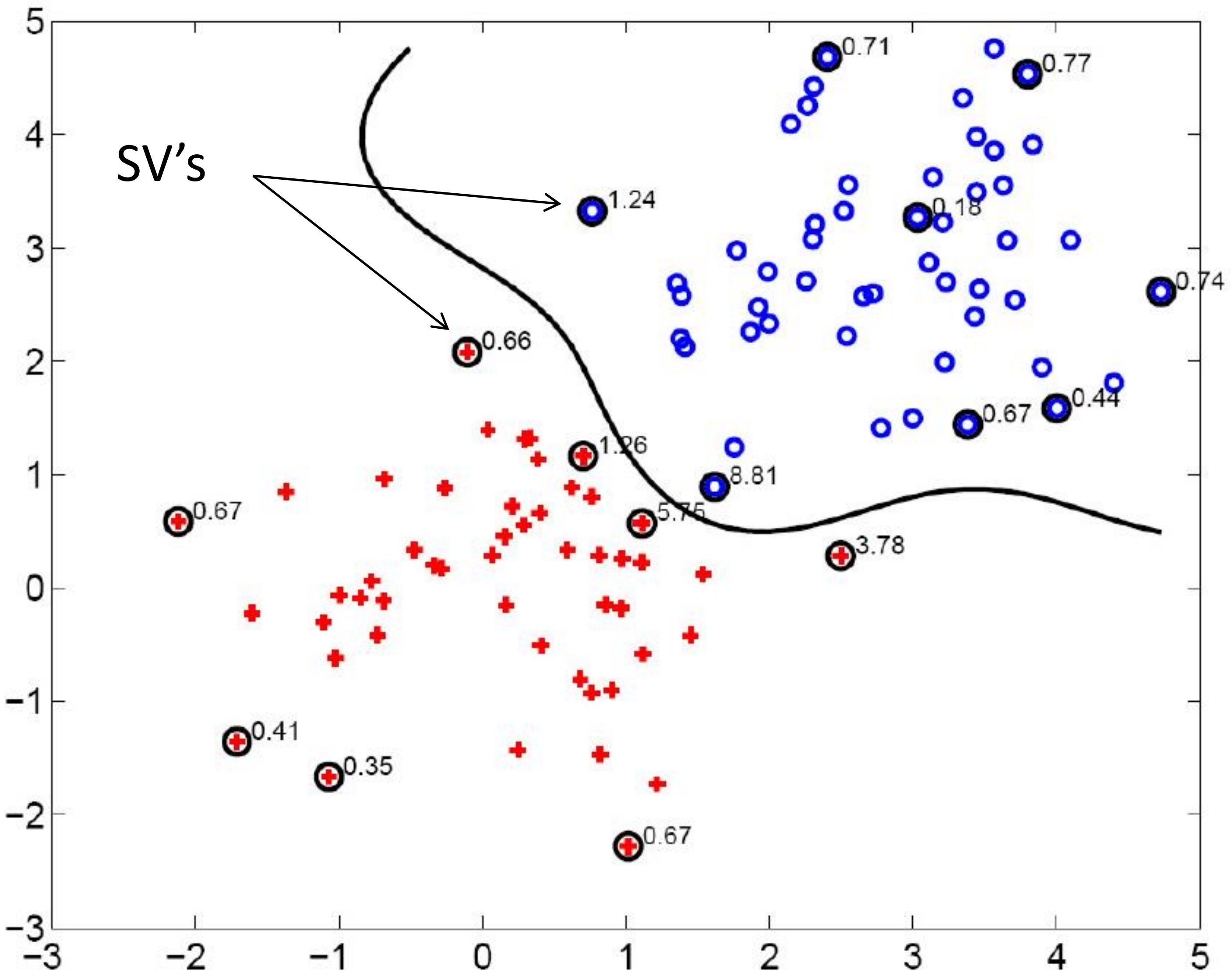
Gaussian kernel

- Also known as the radial basis function (RBF) kernel:

$$K(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{1}{\sigma^2}\|\mathbf{x}-\mathbf{y}\|^2\right)$$



Gaussian kernel



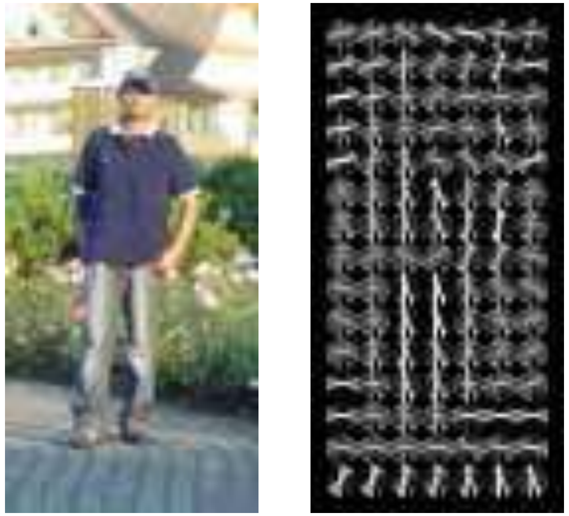
- Demo: http://cs.stanford.edu/people/karpathy/svmjs/demo

SVMs: Pros and cons

- Pros
  - Kernel-based framework is very powerful, flexible
  - Training is convex optimization, globally optimal solution can be found
  - Amenable to theoretical analysis
  - SVMs work very well in practice, even with very small training sample sizes

- Cons
  - No “direct” multi-class SVM, must combine two-class SVMs (e.g., with one-vs-others)
  - Computation, memory (esp. for nonlinear SVMs)

Person detection with HoG's & linear SVM's (so far)



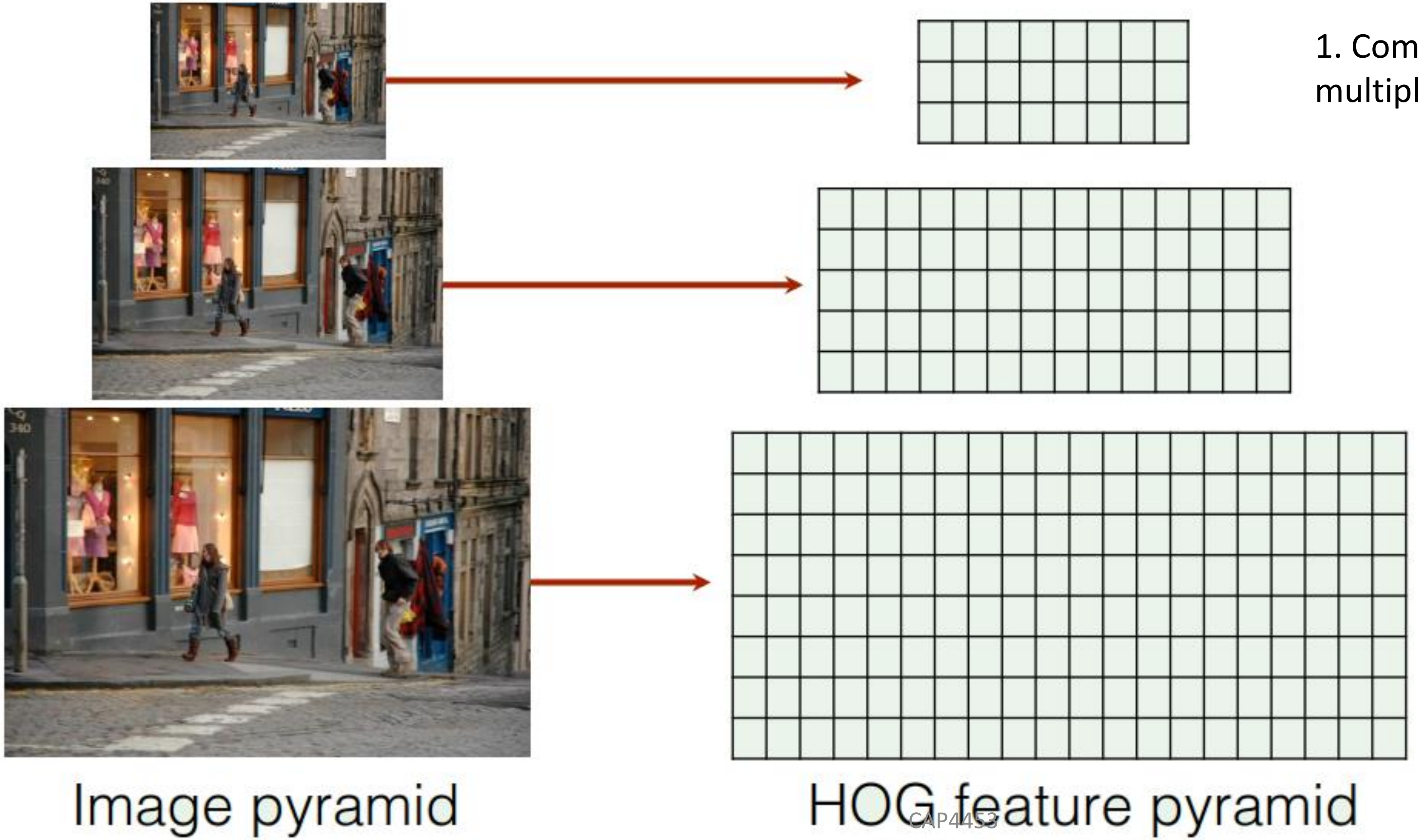
- Histogram of oriented gradients (HoG): Map each grid cell in the input window to a histogram counting the gradients per orientation.
- Train a linear SVM using training set of pedestrian vs. non-pedestrian windows.

Dalal & Triggs, CVPR 2005

The Dalal & Triggs detector

Image pyramid

The Dalal & Triggs detector



1. Compute HOG of the whole image at multiple resolutions!

The Dalal & Triggs detector

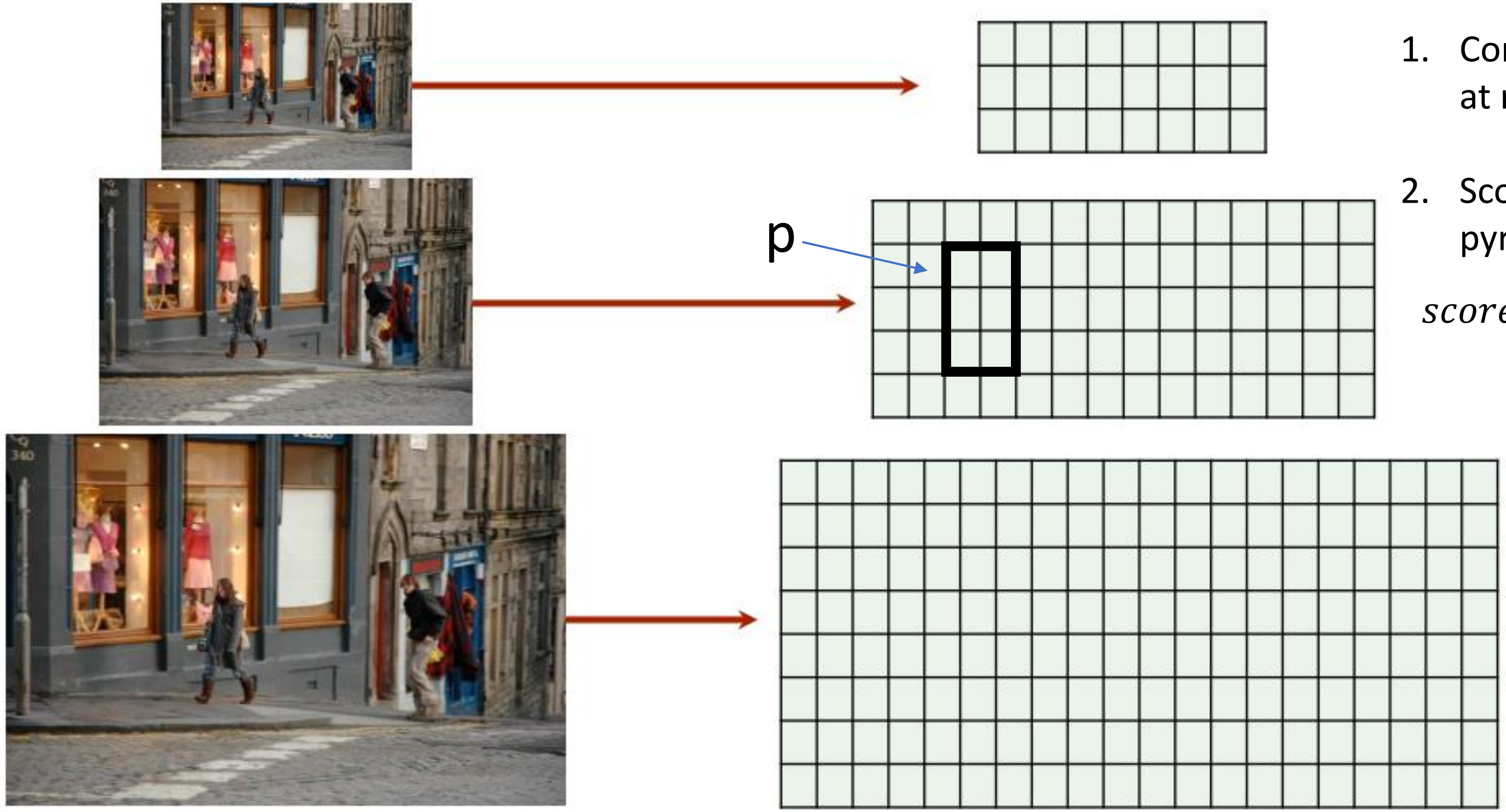


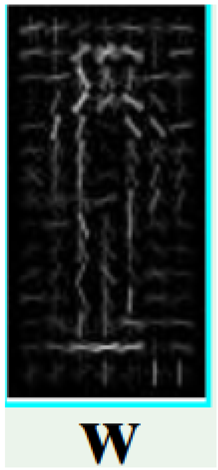
Image pyramid

HOG feature pyramid

1. Compute HOG of the whole image at multiple resolutions!
2. Score every window of the feature pyramid

$$score(I,p) = w \cdot \phi(I,p)$$

FROM TRAINING



The Dalal & Triggs detector

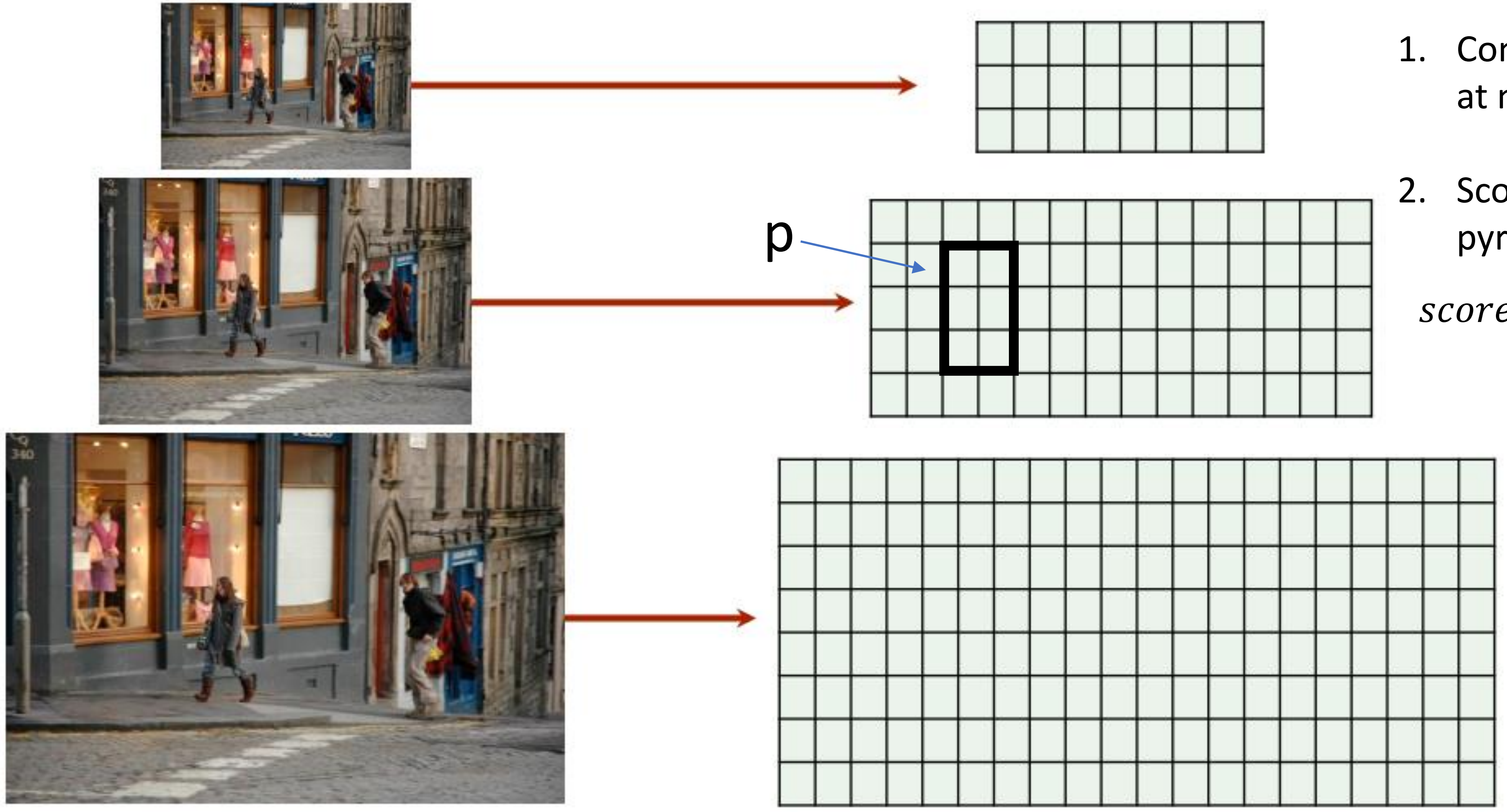


Image pyramid HOG feature pyramid

1. Compute HOG of the whole image at multiple resolutions!
2. Score every window of the feature pyramid

$$score(I, p) = w \cdot \phi(I, p)$$

FROM TRAINING

W

3. Apply non-maximal suppression (NMS)

Outline

- Overview: What is Object detection?
- Top methods for object detection
- **Object detection with Sliding Window and Feature Extraction(HoG)**
  - Sliding Window technique
  - HOG: Gradient based Features
  - Machine Learning
    - Support Vector Machine (SVM)
  - **Non-Maximum Suppression (NMS)**
- Implementation examples
- Deformable Part-based Model (DPM)

Non-Maximum Suppression

Non-Maximum Suppression

Algorithm 1 Non-Max Suppression

procedure NMS(B,c)
    $B_{nms} \leftarrow \emptyset$
    Initialize empty set
    for $b_i \in B$ **do**
    => Iterate over all the boxes
        $discard \leftarrow \text{False}$
        Take boolean variable and set it as false. This variable indicates whether b(i) should be kept or discarded
        for $b_j \in B$ **do**
        Start another loop to compare with b(i)
            if $\text{same}(b_i, b_j) > \boldsymbol{\lambda}_{\mathbf{nms}}$ **then**
            If both boxes having same IOU
                if $\text{score}(c, b_j) > \text{score}(c, b_i)$ **then**
                    $discard \leftarrow \text{True}$
                    Compare the scores. If score of b(i) is less than that of b(j), b(i) should be discarded, so set the flag to True.
        if not $discard$ **then**
            $B_{nms} \leftarrow B_{nms} \cup b_i$
            Once b(i) is compared with all other boxes and still the discarded flag is False, then b(i) should be considered. So add it to the final list.
    return B_{nms}
    Do the same procedure for remaining boxes and return the final list

Outline

- Overview: What is Object detection?
- Top methods for object detection
- **Object detection with Sliding Window and Feature Extraction(HoG)**
  - Sliding Window technique
  - HOG: Gradient based Features
  - Machine Learning
    - Support Vector Machine (SVM)
  - Non-Maximum Suppression (NMS)
- **Implementation examples**
- Deformable Part-based Model (DPM)

Implementation example (car detector)
Get the data. UIUC Car Database

- 550 positive images

- 500 negatives

Implementation example (car detector)

- Extract features

```python
print("Calculating the descriptors for the positive samples and saving them")
for im_path in glob.glob(os.path.join(pos_im_path, "*")):
    im = imread(im_path)
    #print(im.shape)
    if des_type == "HOG":
        fd = hog(im, orientations, pixels_per_cell, cells_per_block, visualize=visualize, block_norm='L2-Hys')
    fd_name = os.path.split(im_path)[1].split(".")[0] + ".feat"
    fd_path = os.path.join(pos_feat_ph, fd_name)
    joblib.dump(fd, fd_path)
print("Positive features saved in {}".format(pos_feat_ph))

print("Calculating the descriptors for the negative samples and saving them")
for im_path in glob.glob(os.path.join(neg_im_path, "*")):
    im = imread(im_path)
    if des_type == "HOG":
        fd = hog(im,  orientations, pixels_per_cell, cells_per_block, visualize=visualize, block_norm='L2-Hys')
    fd_name = os.path.split(im_path)[1].split(".")[0] + ".feat"
    fd_path = os.path.join(neg_feat_ph, fd_name)
    joblib.dump(fd, fd_path)
print("Negative features saved in {}".format(neg_feat_ph))
```

```
[hog]
min_wdw_sz: [100, 40]
step_size: [10, 10]
orientations: 9
pixels_per_cell: [8, 8]
cells_per_block: [3, 3]
visualize: True
normalize: True
```

```
[paths]
pos_feat_ph: ../data/features/pos
neg_feat_ph: ../data/features/neg
model_path: ../data/models/svm.model
```

[VladKha/object_detector: Object detector from HOG + Linear SVM framework (github.com)]

Implementation example (car detector)

- Train SVM with HOG features

```python
# Import the required modules
from skimage.feature import local_binary_pattern
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
import joblib
import argparse as ap
import glob
import os
from config import *

if __name__ == "__main__":
    # Parse the command line arguments
    parser = ap.ArgumentParser()
    parser.add_argument('-p', "--posfeat", help="Path to the positive features directory", required=True)
    parser.add_argument('-n', "--negfeat", help="Path to the negative features directory", required=True)
    parser.add_argument('-c', "--classifier", help="Classifier to be used", default="LIN_SVM")
    args = vars(parser.parse_args())
    #print(str(args))

    pos_feat_path =  args["posfeat"]
    neg_feat_path = args["negfeat"]

    #print(pos_feat_path)

    # Classifiers supported
    clf_type = args['classifier']

    fds = []
    labels = []
    # Load the positive features
    for feat_path in glob.glob(os.path.join(pos_feat_path,"*.feat")):
        print(feat_path)
        fd = joblib.load(feat_path)
        fds.append(fd)
        labels.append(1)

    # Load the negative features
    for feat_path in glob.glob(os.path.join(neg_feat_path,"*.feat")):
        fd = joblib.load(feat_path)
        fds.append(fd)
        labels.append(0)

    if clf_type is "LIN_SVM":
        clf = LinearSVC()
        print("Training a Linear SVM Classifier")
        print(fds)
        print(labels)

        clf.fit(fds, labels)
        # If feature directories don't exist, create them
        if not os.path.isdir(os.path.split(model_path)[0]):
            os.makedirs(os.path.split(model_path)[0])
        joblib.dump(clf, model_path)
        print("Classifier saved to {}".format(model_path))
```

Implementation example (car detector)

```python
from skimage.transform import pyramid_gaussian
from skimage.io import imread
from skimage.feature import hog
import joblib
import cv2
import argparse as ap
from nms import nms
from config import *
import numpy as np
```

Test

- Load image
- Loop over different pyramid images
  - loop the window position
    - Compute HOG for each window
    - Compute score

```python
# Downscale the image and iterate
for im_scaled in pyramid_gaussian(im, downscale=downscale):
    print(im_scaled.shape)
    # This list contains detections at the current scale
    cd = []
    # If the width or height of the scaled image is less than
    # the width or height of the window, then end the iterations.
    if im_scaled.shape[0] < min_wdw_sz[1] or im_scaled.shape[1] < min_wdw_sz[0]:
        break
    for (x, y, im_window) in sliding_window(im_scaled, min_wdw_sz, step_size):
        print('x,y: ' + str(x) + ' ' +str(y))
        if im_window.shape[0] != min_wdw_sz[1] or im_window.shape[1] != min_wdw_sz[0]:
            continue
        # Calculate the HOG features
        if visualize:
            (fd,imgVis)= hog(im_window, orientations, pixels_per_cell, cells_per_block, visualize=True, block_norm='L2-Hy
            cv2.imshow('HOGinput' , imgVis )
            cv2.waitKey(30)
        else:
            fd= hog(im_window, orientations, pixels_per_cell, cells_per_block, visualize=False, block_norm='L2-Hys')

        fd = fd[np.newaxis,:]
        #print(fd.shape)
        pred = clf.predict(fd)
        if pred == 1:
            print("Detection:: Location -> (" + str(x)+ "," + str(y) +")")
            #print("Scale -> "+ str(scale) + "| Confidence Score " + clf.decision_function(fd) +"\n")
            print("Scale ->  {} | Confidence Score {} \n".format(scale,clf.decision_function(fd)))
            detections.append((x, y, clf.decision_function(fd),
                int(min_wdw_sz[0]*(downscale**scale)),
                int(min_wdw_sz[1]*(downscale**scale))))
            cd.append(detections[-1])
```

Implementation example (car detector)

```python
from skimage.transform import pyramid_gaussian
from skimage.io import imread
from skimage.feature import hog
import joblib
import cv2
import argparse as ap
from nms import nms
from config import *
import numpy as np
```

Test

- Load image
- Loop over different pyramid images
  - loop the window position
    - Compute HOG for each window
    - Compute score
- Perform NMS

```python
# Downscale the image and iterate
for im_scaled in pyramid_gaussian(im, downscale=downscale):
    print(im_scaled.shape)
    # This list contains detections at the current scale
    cd = []
    # If the width or height of the scaled image is less than
    # the width or height of the window, then end the iterations.
    if im_scaled.shape[0] < min_wdw_sz[1] or im_scaled.shape[1] < min_wdw_sz[0]:
        break
    for (x, y, im_window) in sliding_window(im_scaled, min_wdw_sz, step_size):
        print('x,y: ' + str(x) + ' ' +str(y))
        if im_window.shape[0] != min_wdw_sz[1] or im_window.shape[1] != min_wdw_sz[0]:
            continue
        # Calculate the HOG features
        if visualize:
            (fd,imgVis)= hog(im_window, orientations, pixels_per_cell, cells_per_block, visualize=True, block_norm='L2-Hys
            cv2.imshow('HOGinput' , imgVis )
            cv2.waitKey(30)
        else:
            fd= hog(im_window, orientations, pixels_per_cell, cells_per_block, visualize=False, block_norm='L2-Hys')

        fd = fd[np.newaxis,:]
        #print(fd.shape)
        pred = clf.predict(fd)
        if pred == 1:
            print("Detection:: Location -> (" + str(x)+ "," + str(y) +")")
            #print("Scale -> "+ str(scale) + "| Confidence Score " + clf.decision_function(fd) +"\n")
            print("Scale ->  {} | Confidence Score {} \n".format(scale,clf.decision_function(fd)))
            detections.append((x, y, clf.decision_function(fd),
                int(min_wdw_sz[0]*(downscale**scale)),
                int(min_wdw_sz[1]*(downscale**scale))))
            cd.append(detections[-1])
```

```python
# Perform Non Maxima Suppression
detections = nms(detections, threshold)
```

Testing (different pyramid levels)

NMS

Before NMS

After NMS

Questions?