CAP 4453 Robot Vision

Dr. Gonzalo Vaca-Castaño

gonzalo.vacacastano@ucf.edu

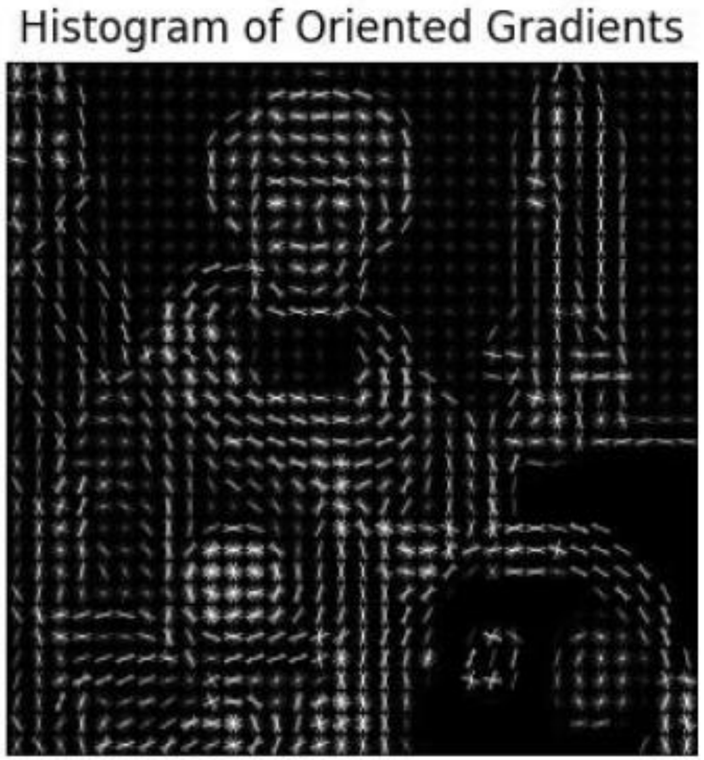
Administrative details

- Issues submitting homework

Credits

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

Histogram of Oriented Gradients (HOG)



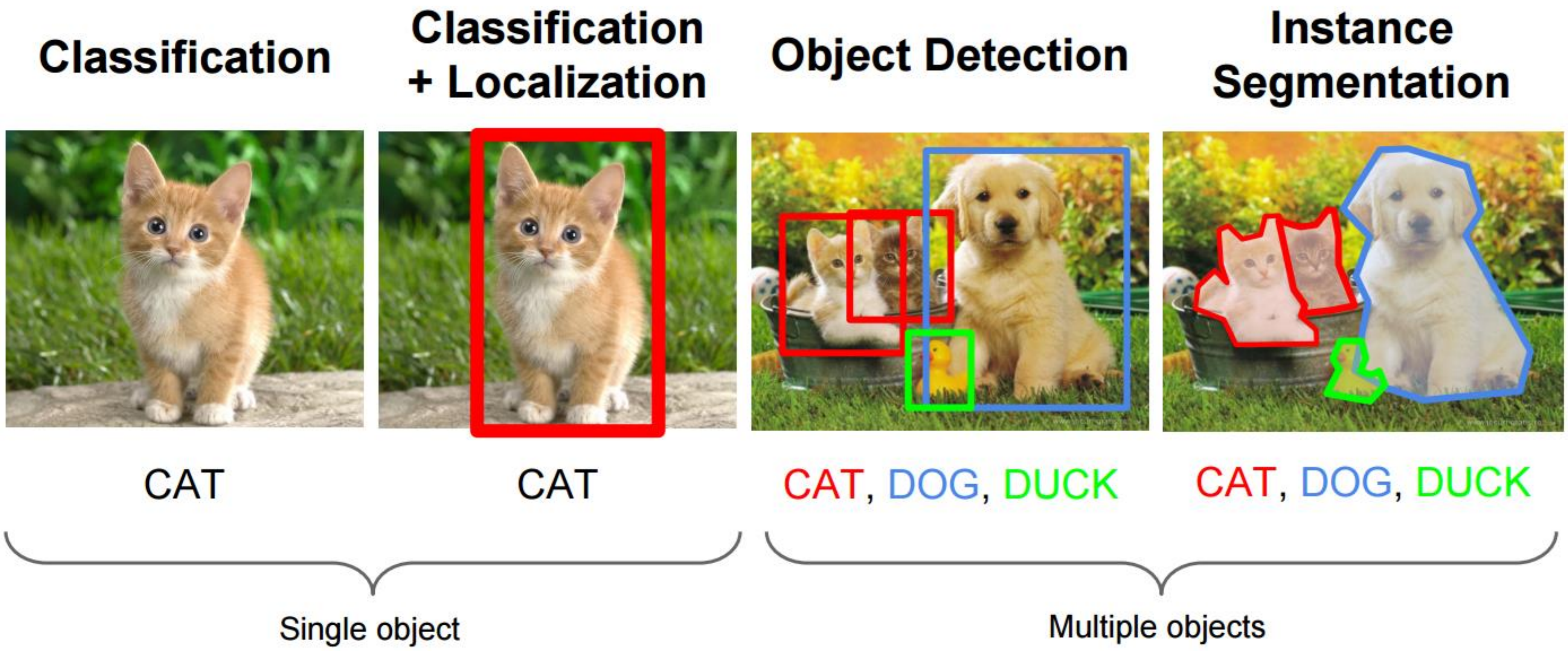
Robot Vision

13. Object detection I

Outline

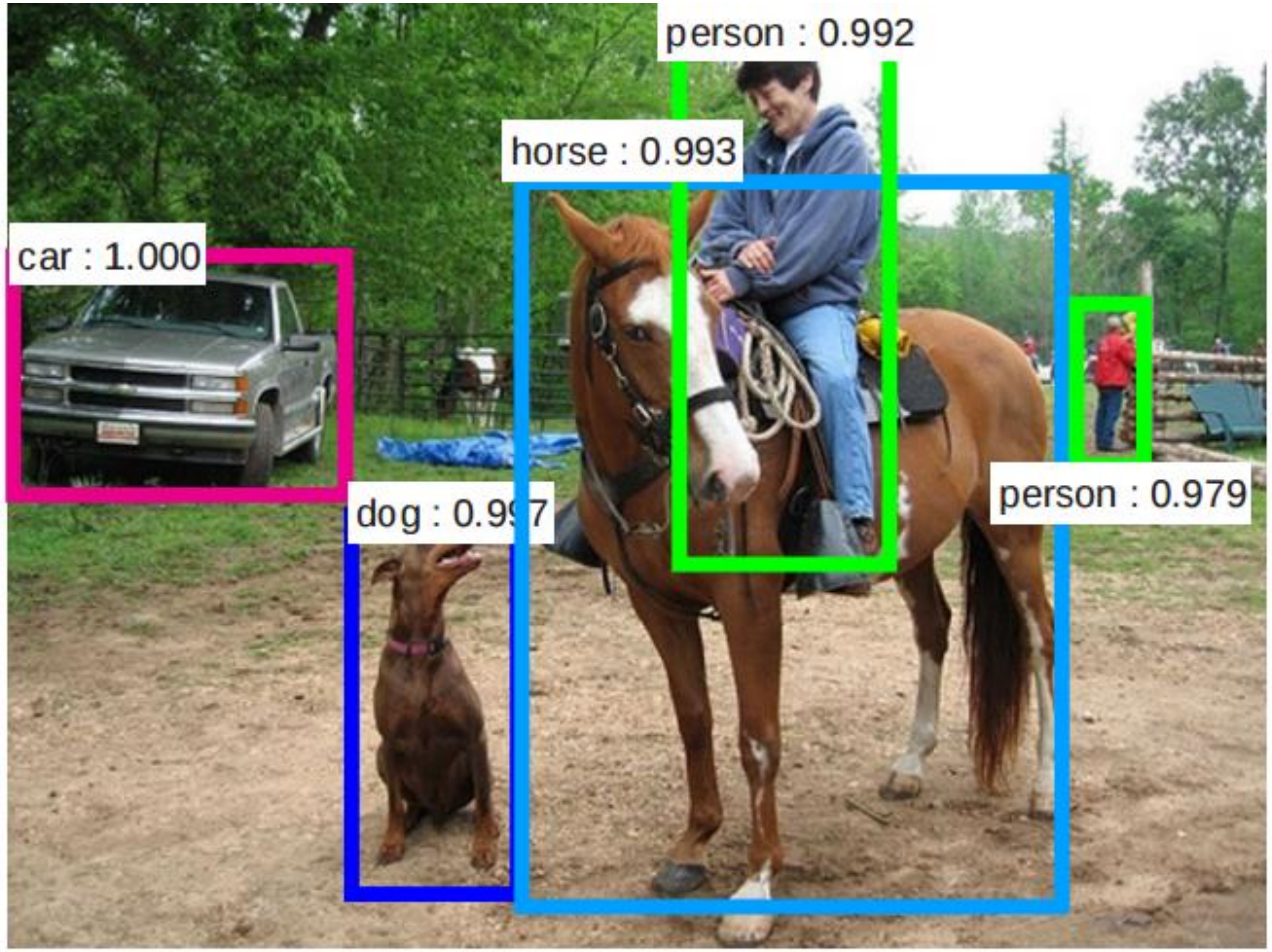
- **Overview: What is Object detection?**
- Top methods for object detection
- Object detection with Sliding Window and Feature Extraction(HoG)
  - Sliding Window technique
  - HOG: Gradient based Features
  - Machine Learning
    - Support Vector Machine (SVM)
  - Non-Maxima Suppression (NMS)
- Implementation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

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

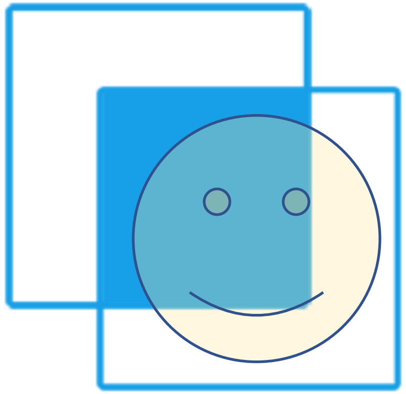
VOC: 20 classes

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

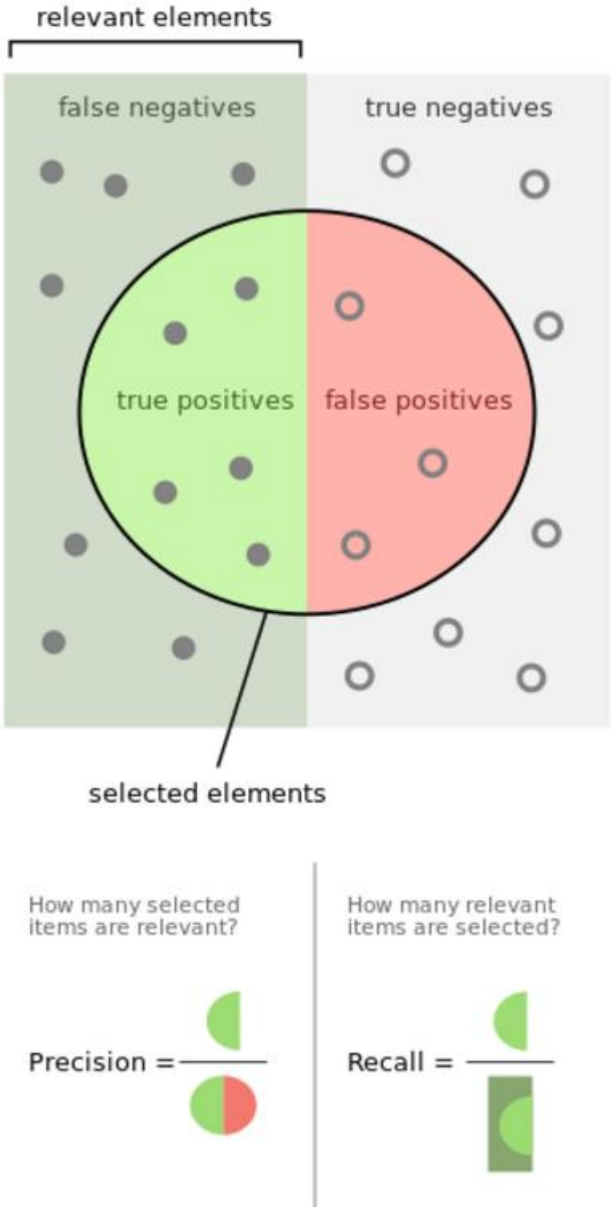
$$score_{iou} > 0.5$$

All of these:

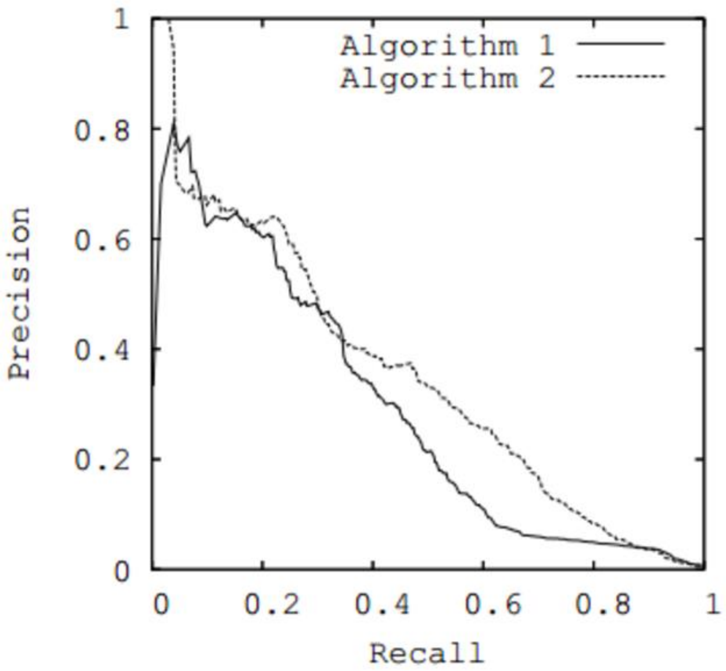
$score_{iou} > 0.5$
$score_{iou} > 0.55$
$score_{iou} > 0.6$
$score_{iou} > 0.65$
$score_{iou} > 0.7$
$score_{iou} > 0.75$
$score_{iou} > 0.8$
$score_{iou} > 0.9$
$score_{iou} > 0.95$

Terms

Recall
Precision
mAP
IoU

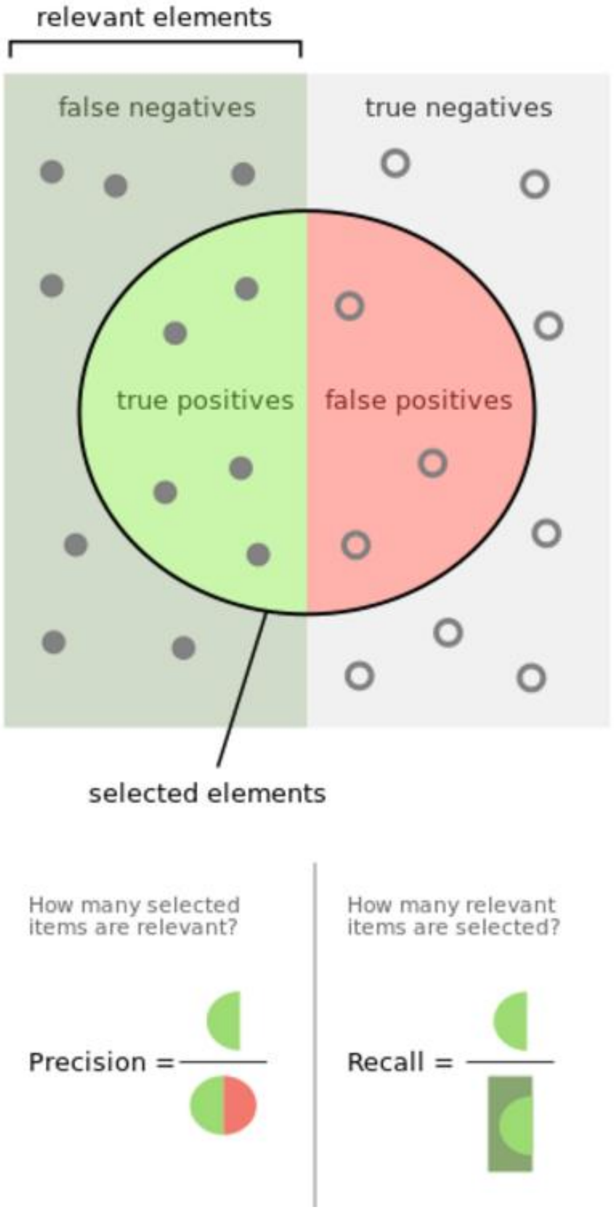


Possible detection
- Bounding box
- Label
- ***score***

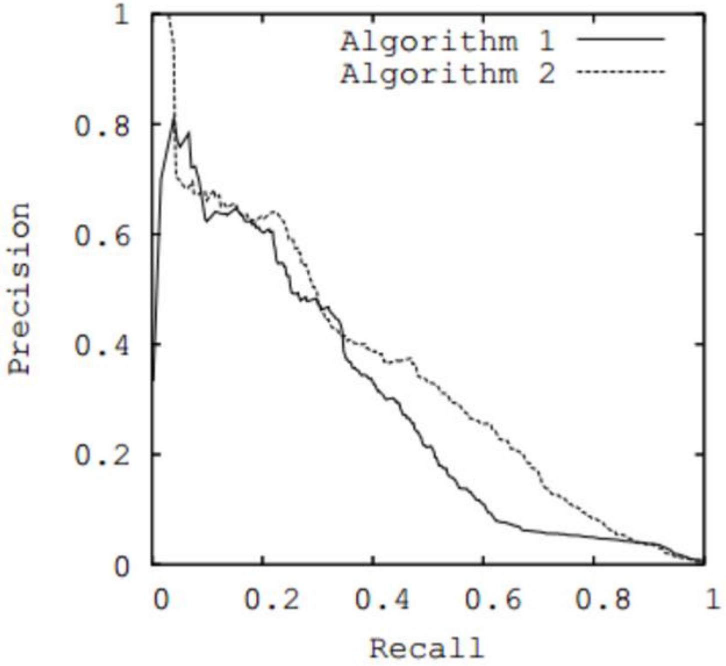


Terms

Recall
Precision
mAP
IoU



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



Average precision (AP): Area under curve

Terms

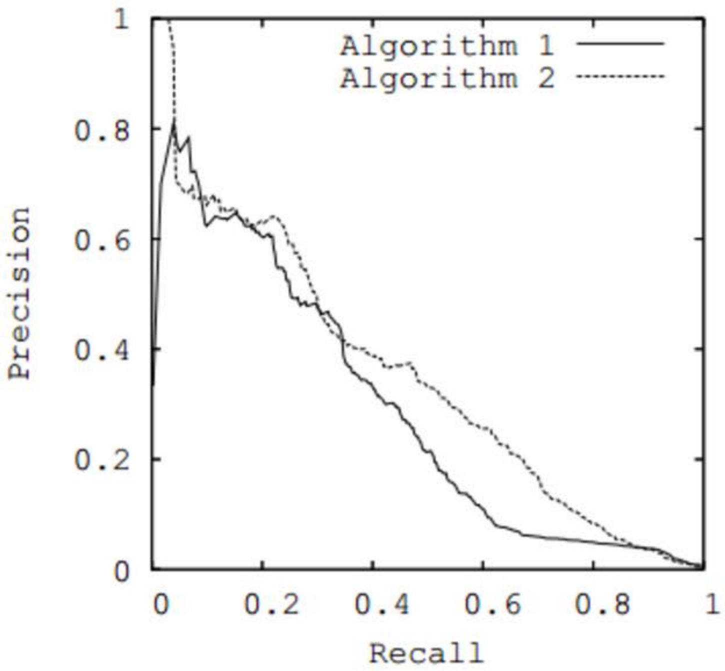
Possible detection

- 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)

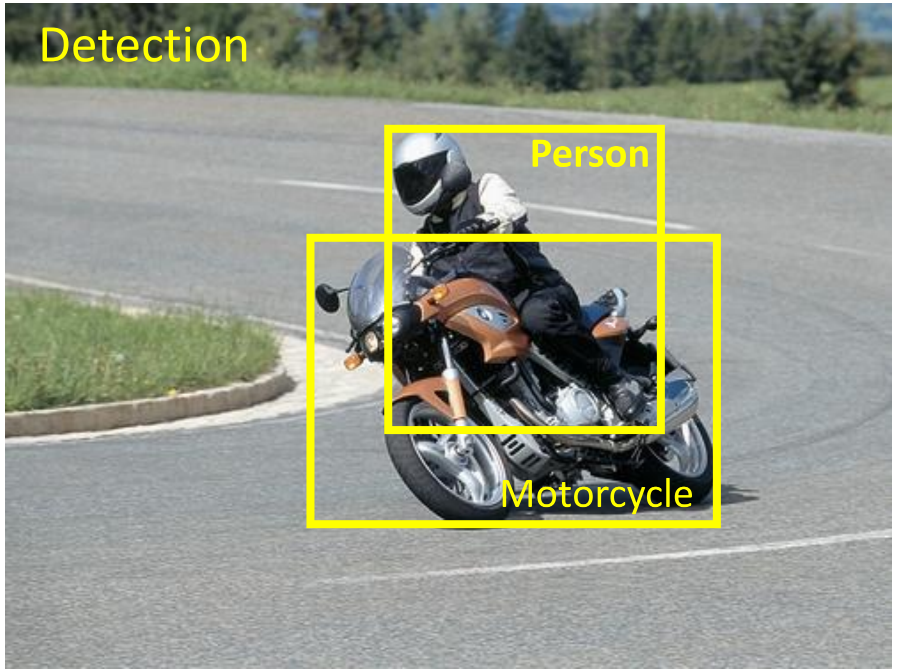
Popular algorithms for object detection

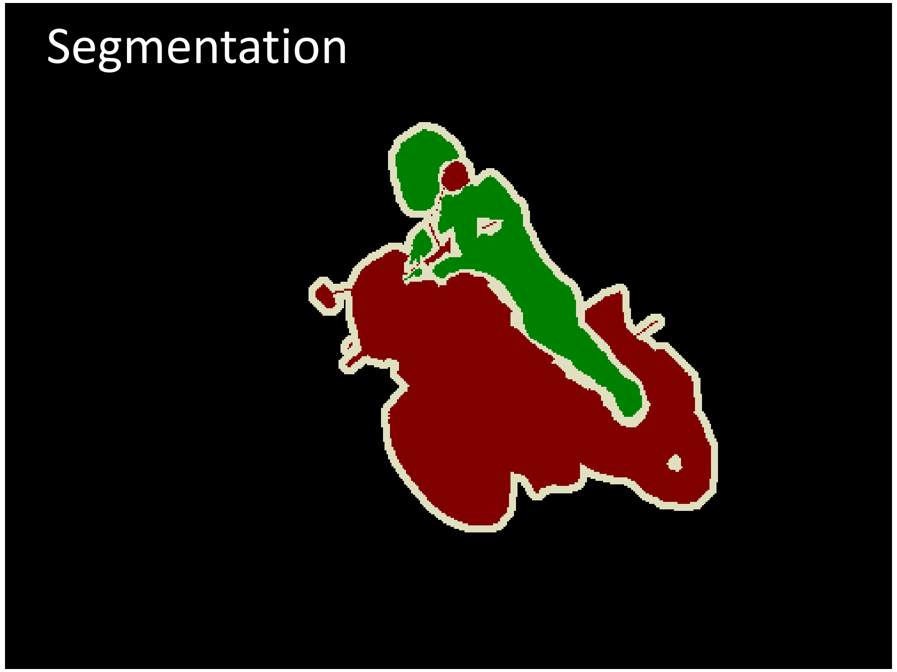
- Pre-DeepLearning
  - HOG + SVM (Dalal, Triggs)
  - Deformable Part-based Model (DPM)
- Deep learning
  - Fast R-CNN
  - Faster R-CNN
  - Region-based Convolutional Neural Networks (R-CNN)
  - Region-based Fully Convolutional Network
  - Single Shot Detector (SSD)
  - YOLO (You Only Look Once)

PASCAL VOC 2005-2012

20 object classes **22,591 images**

Classification: person, motorcycle



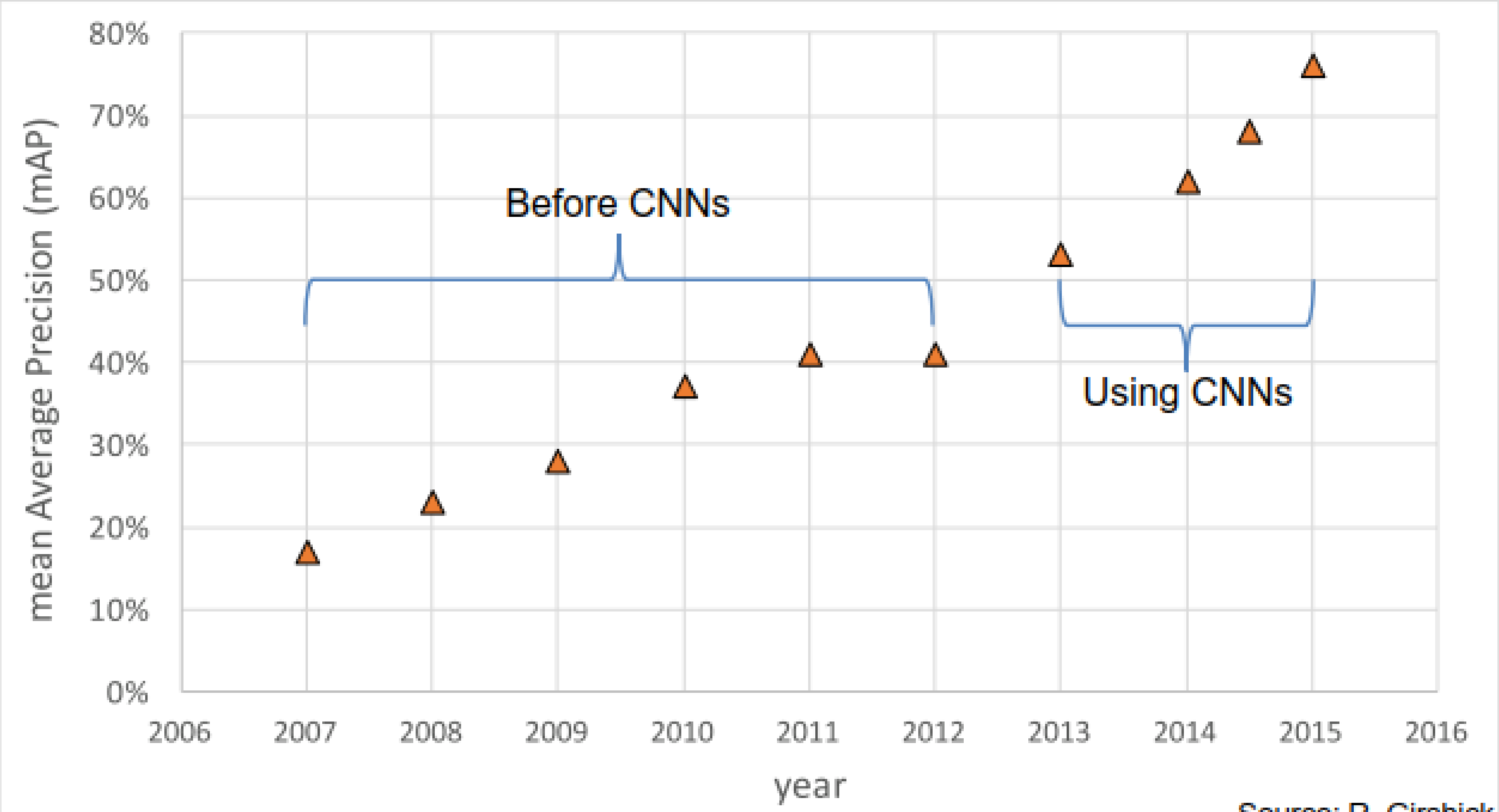


Action: riding bicycle

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

Object detection progress

PASCAL VOC

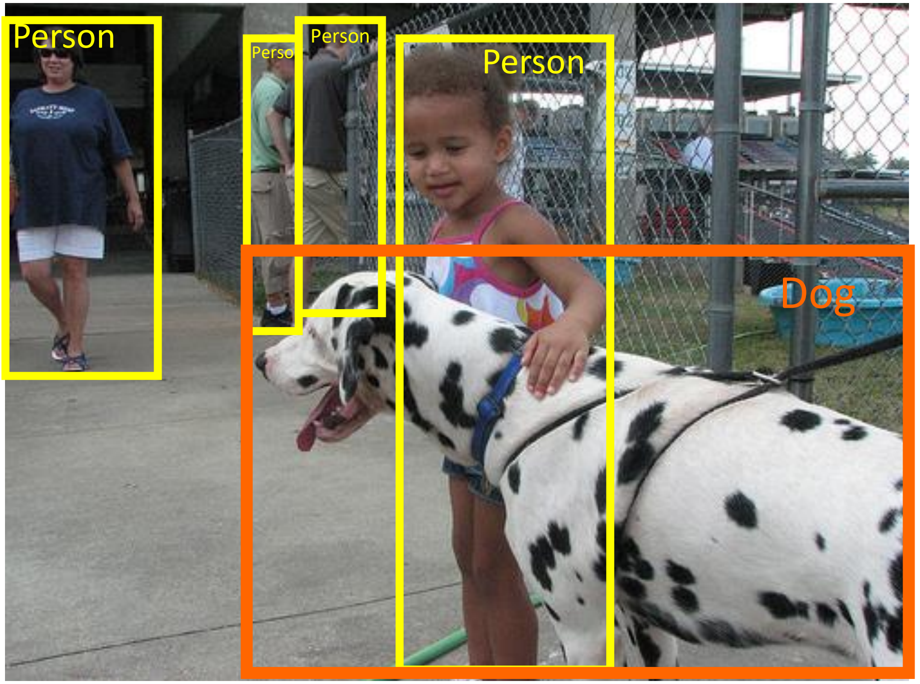


Source: R. Girshick

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

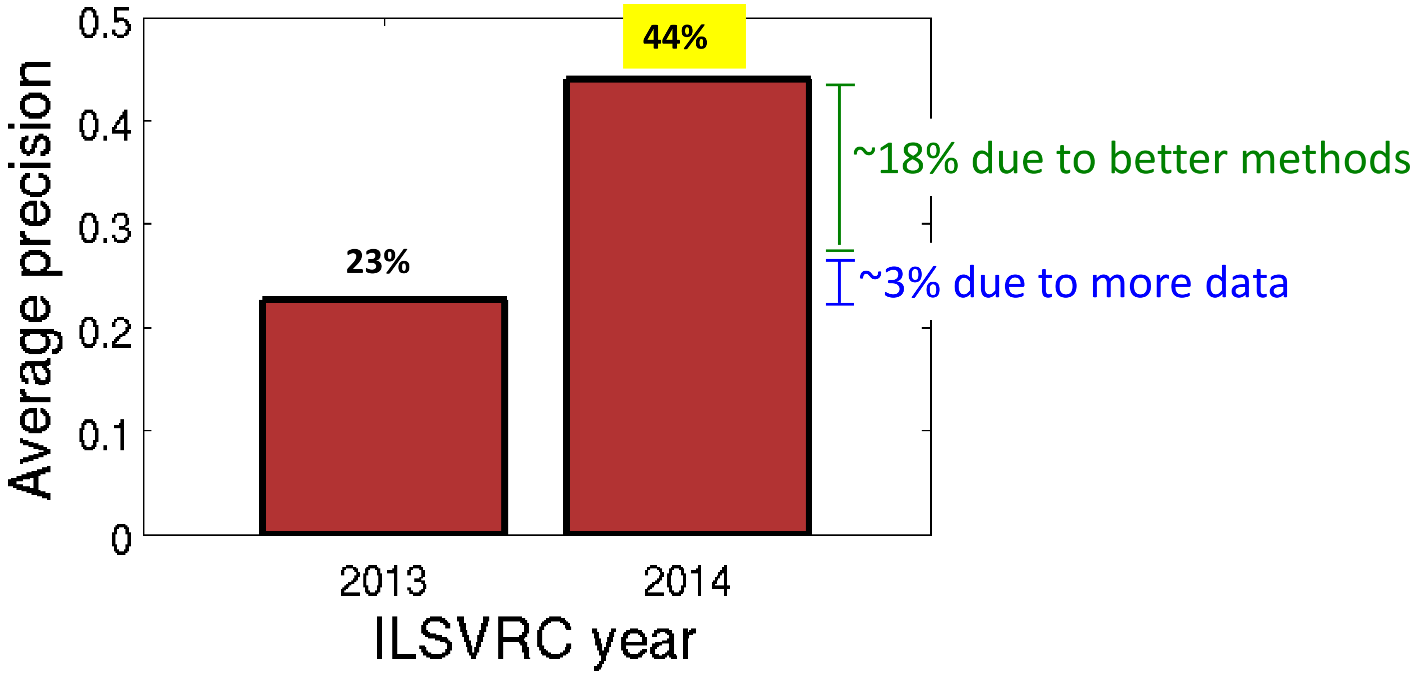
~~20 object classes 22,591 images~~

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

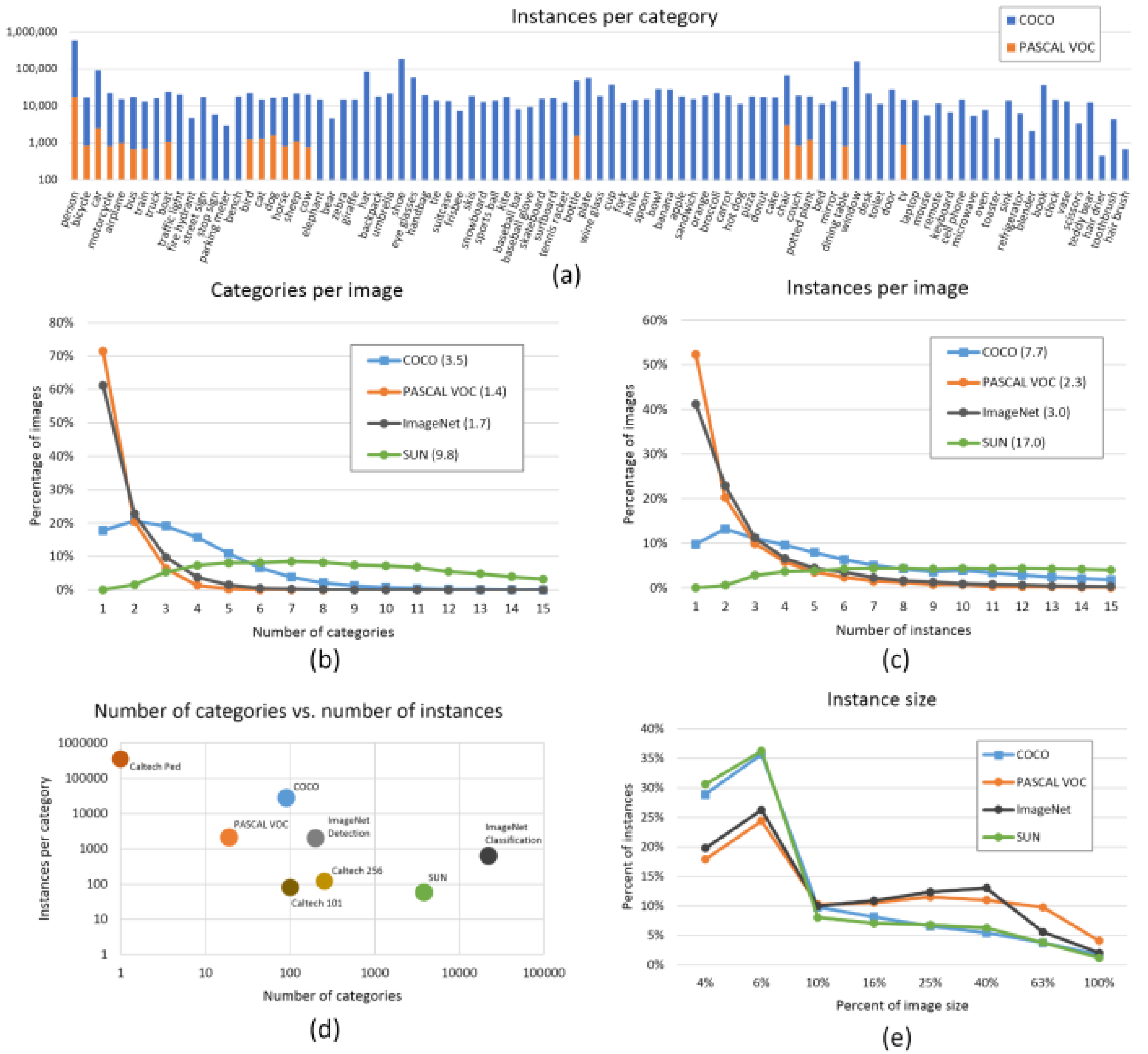
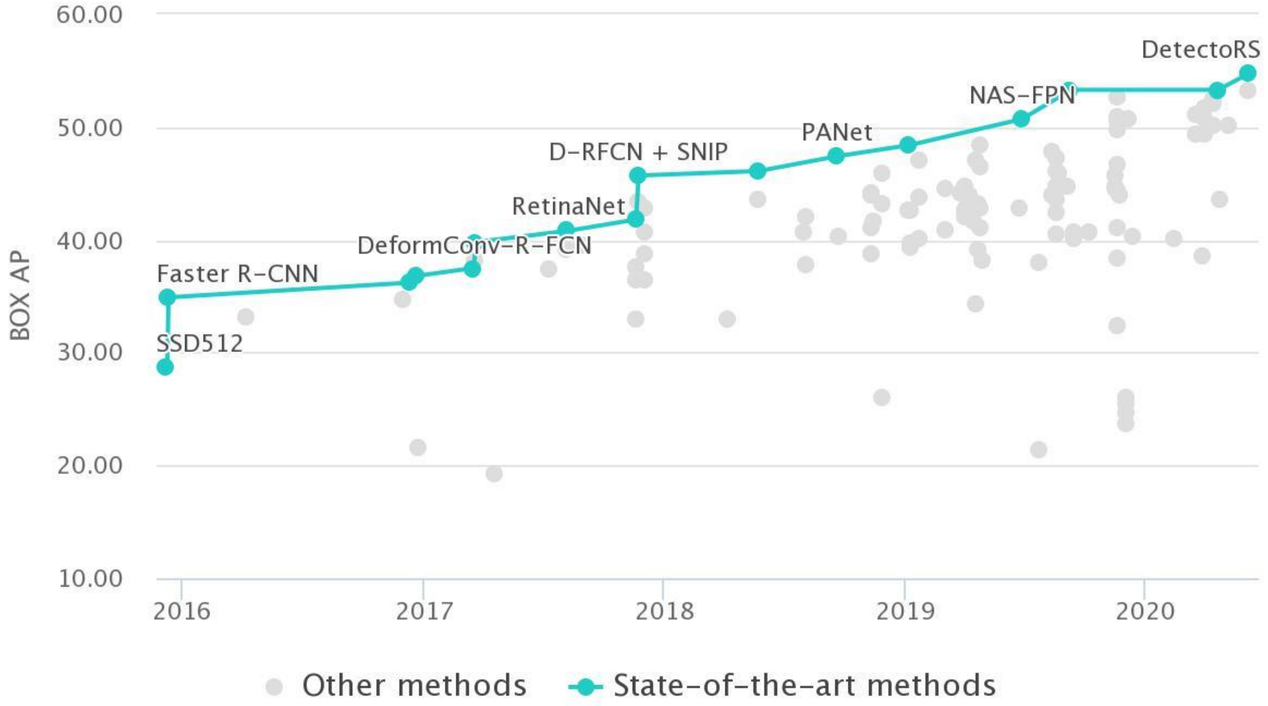


Fig. 5: (a) Number of annotated instances per category for MS COCO and PASCAL VOC. (b,c) Number of annotated categories and annotated instances, respectively, per image for MS COCO, ImageNet Detection, PASCAL VOC and SUN (average number of categories and instances are shown in parentheses). (d) Number 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)](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

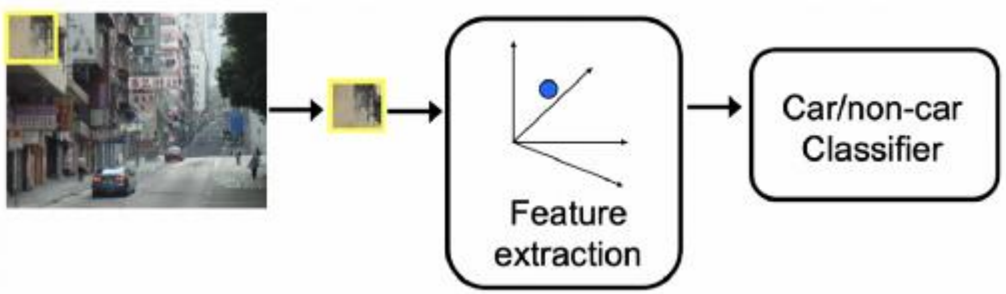
- CVPR 2005

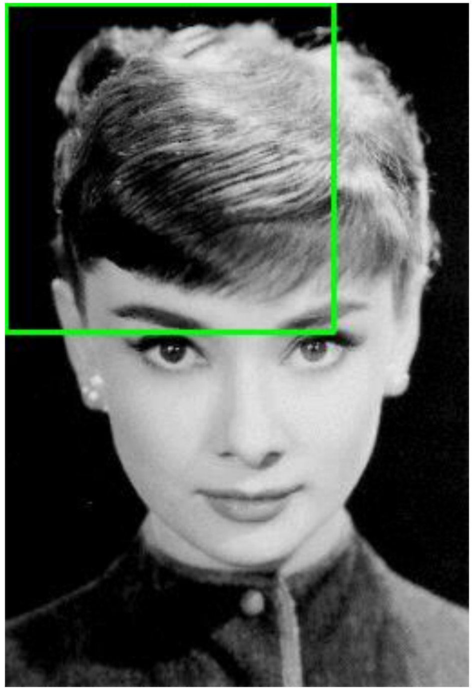
Sliding Window Technique

- 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

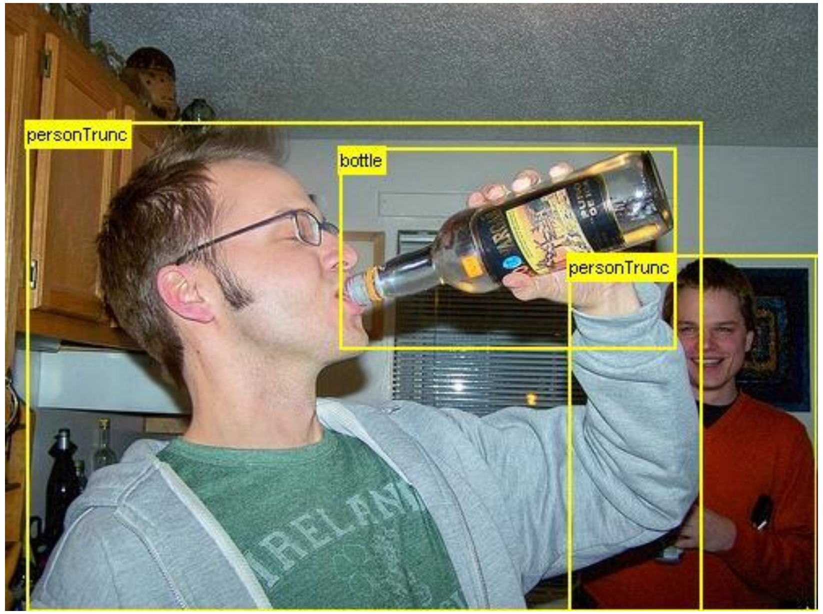
Cons

- 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

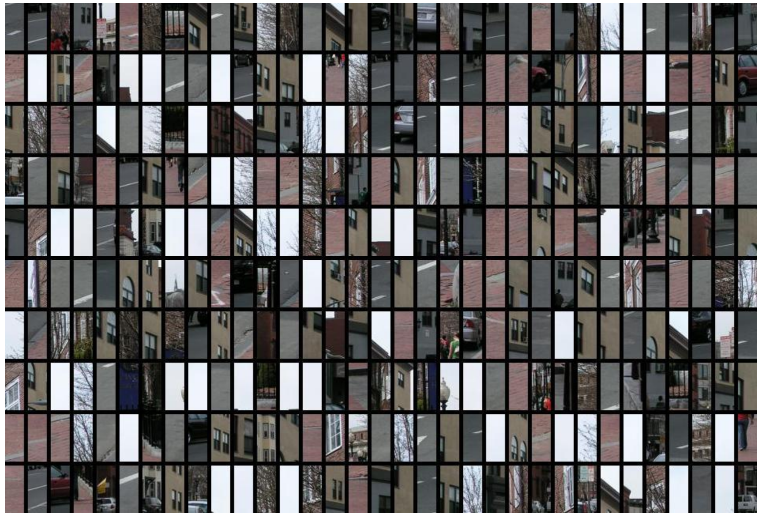


Slide: Kristen Grauman

Limitations (continued)

- If considering windows in isolation, context is lost

Sliding window



Detector's view

Figure credit: Derek Hoiem

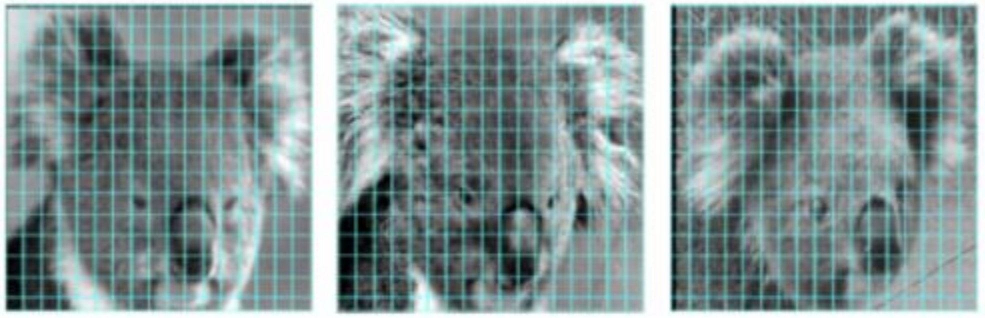
Slide: Kristen Grauman

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)

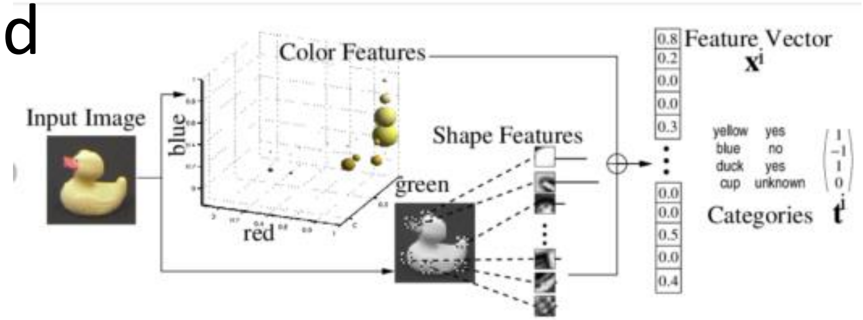
Let's examine possible feature vectors

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



(1) (2) (3)

- Color based

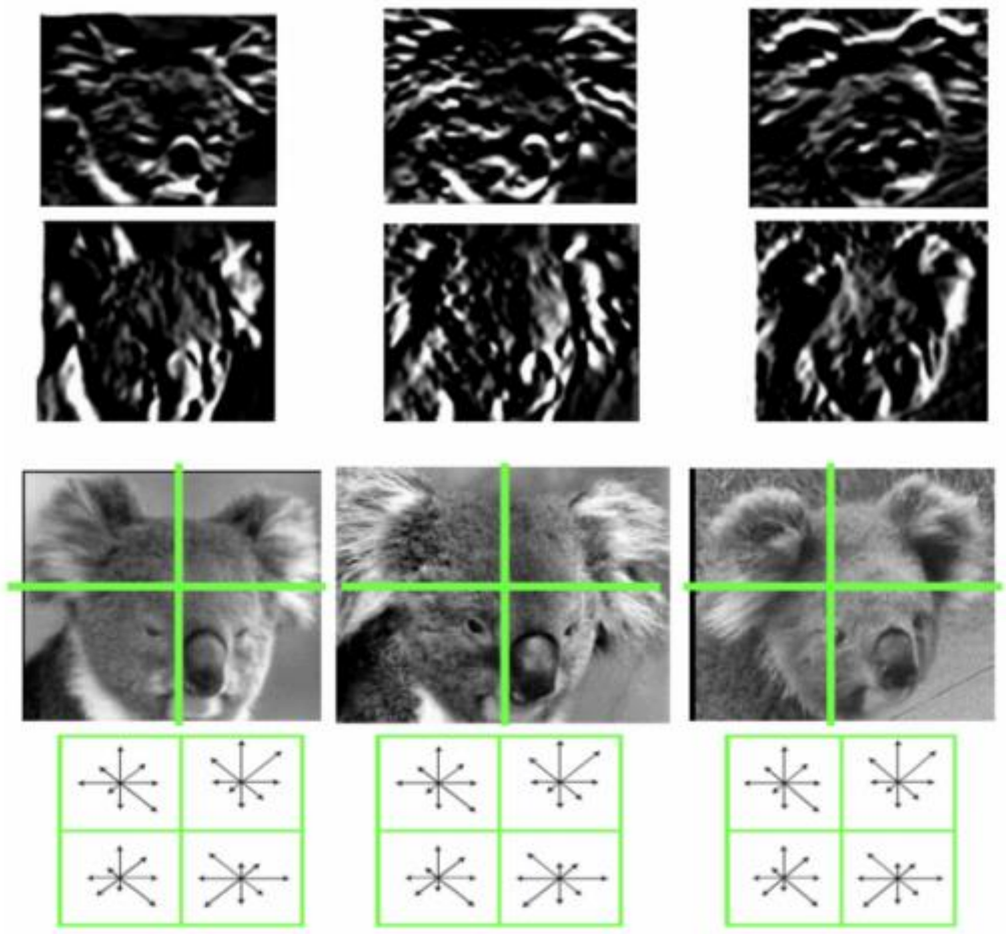


  - color-based representations are sensitive to color (illumination)

(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

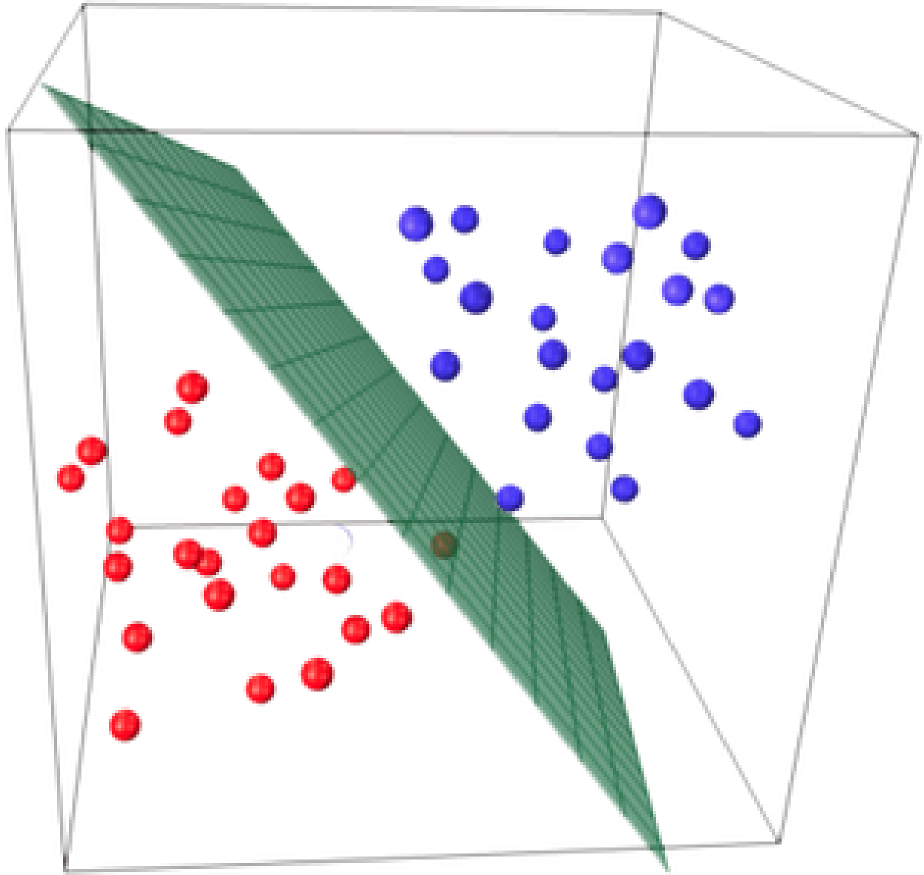
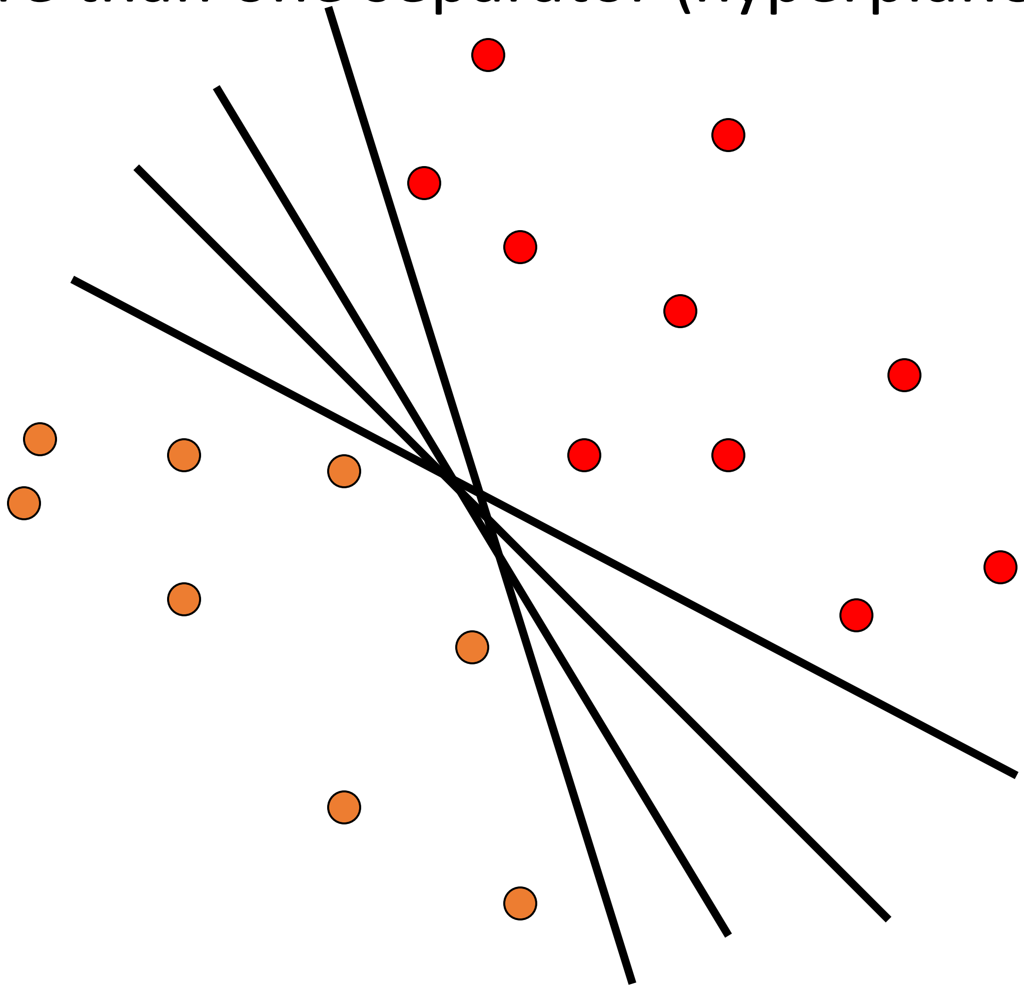


Image source

Support vector machines

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

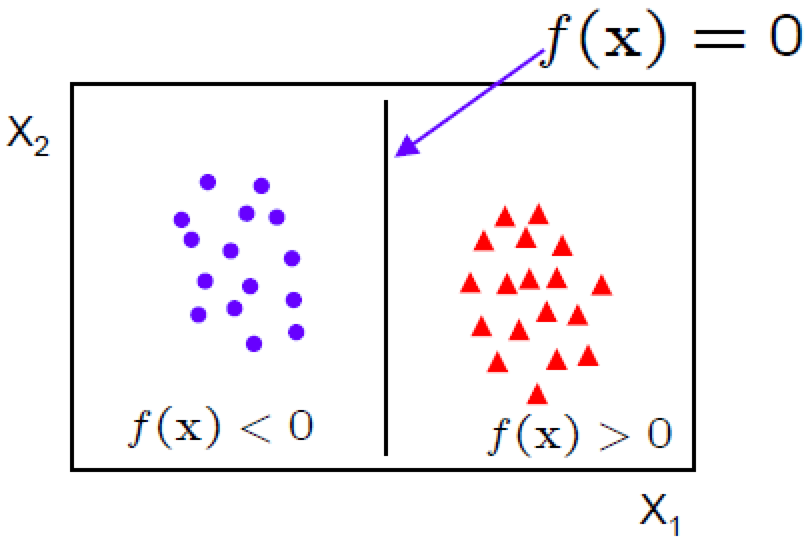


Which separator is best?

Linear classifiers

A linear classifier has the form

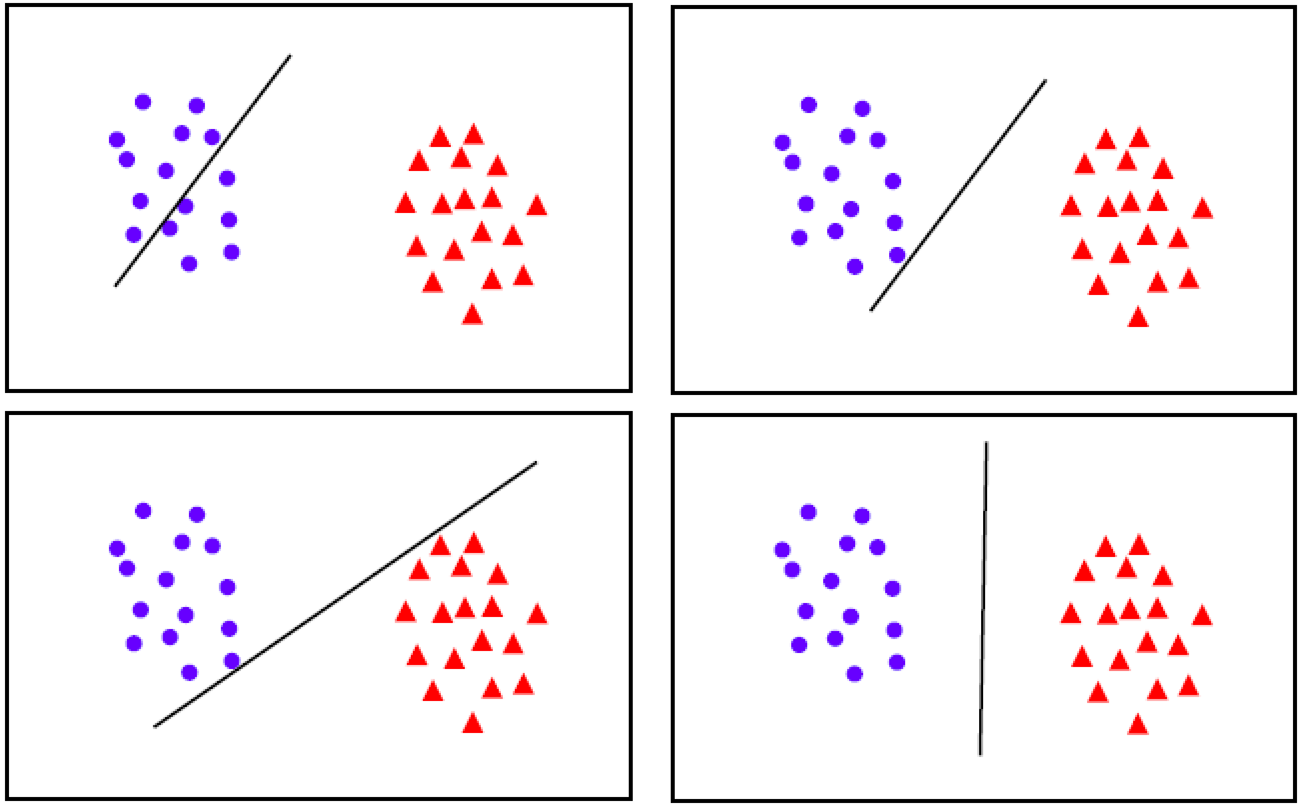
$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$



- 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?

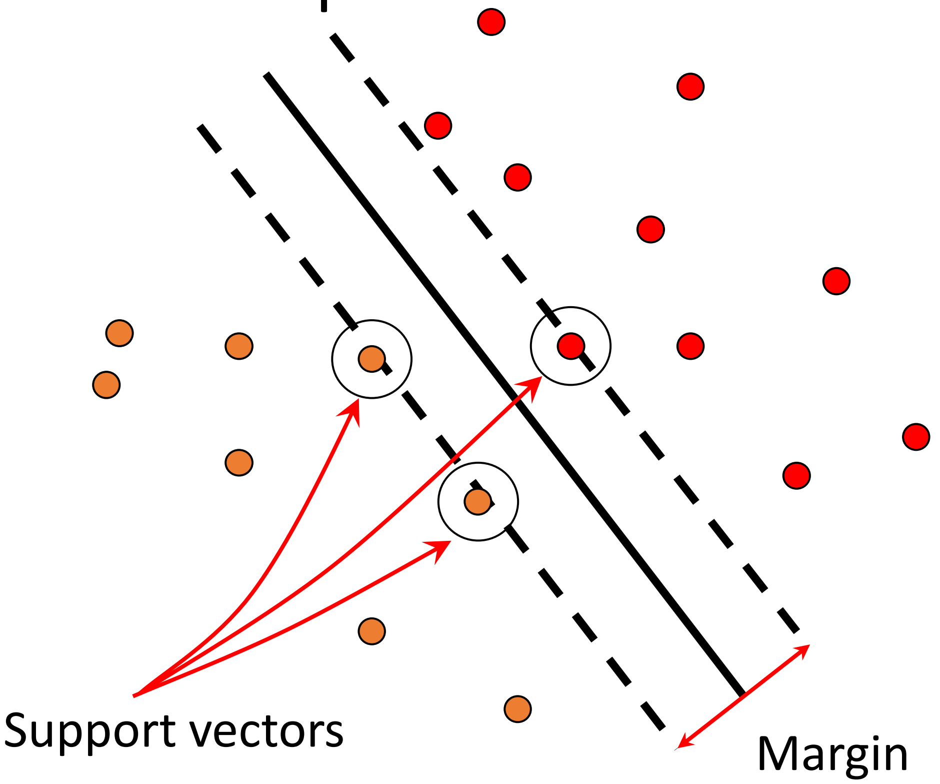


• maximum margin solution: most stable under perturbations of the inputs

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



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

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

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

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

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:

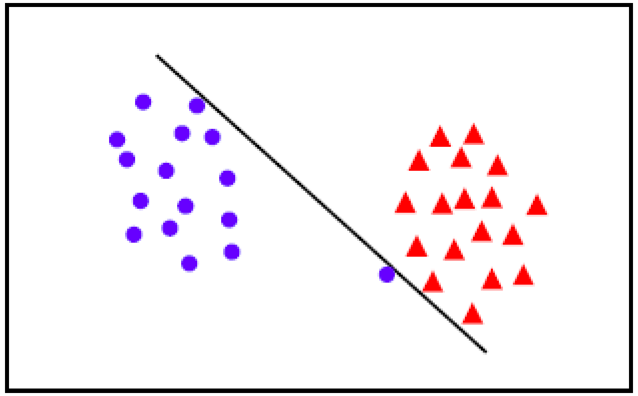
$$\mathbf{x}_i \text{ positive } (y_i = 1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

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

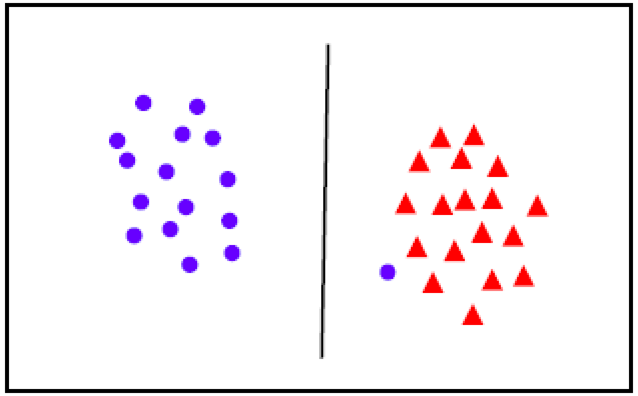
- *Quadratic optimization problem*:
- $$\boxed{\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1}$$

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

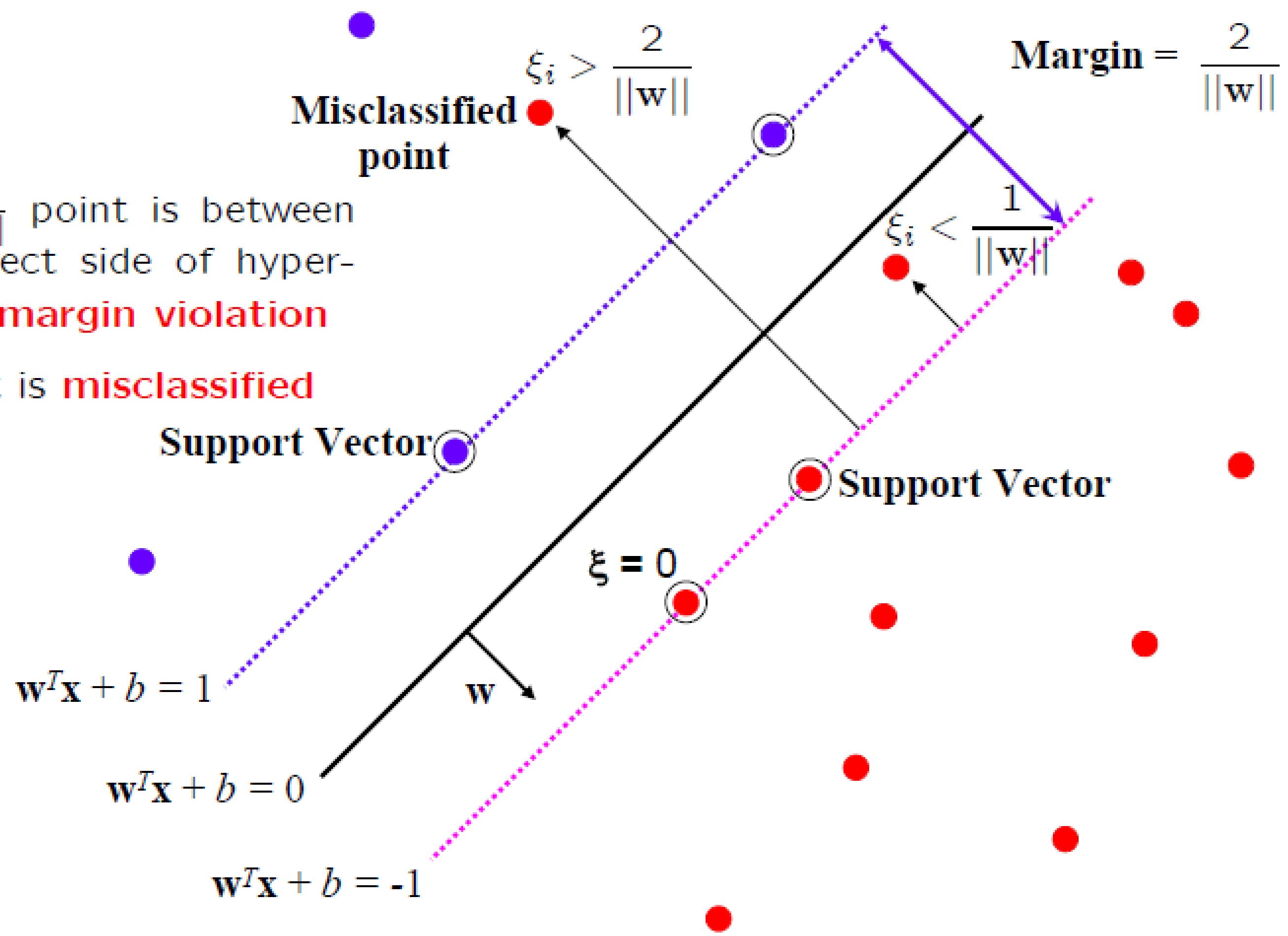
In general there is a trade off between the margin and the number of mistakes on the training data

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

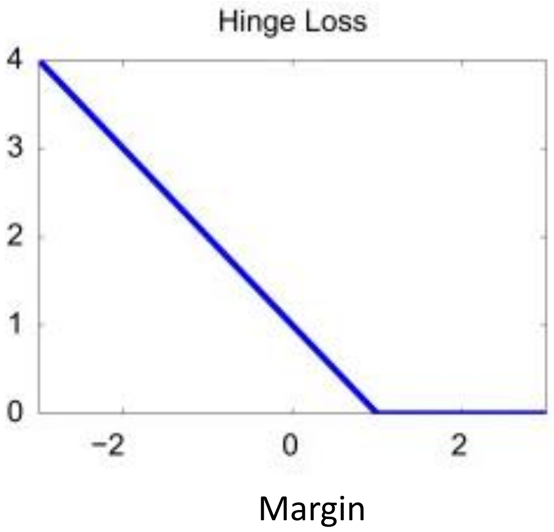
- Separable data: $$\min_{\mathbf{w},b} \underbrace{\frac{1}{2}\|\mathbf{w}\|^2}_{\text{Maximize margin}} \quad \underbrace{\text{subject to} \quad y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1}_{\text{Classify training data correctly}}$$

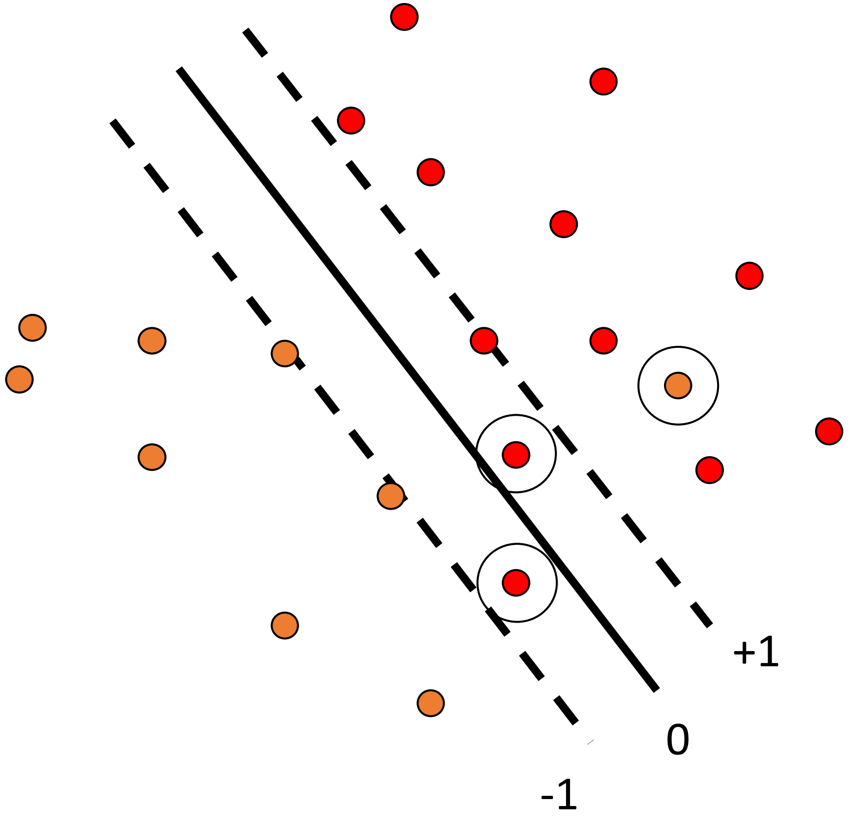
- Non-separable data:

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

SVM training in general

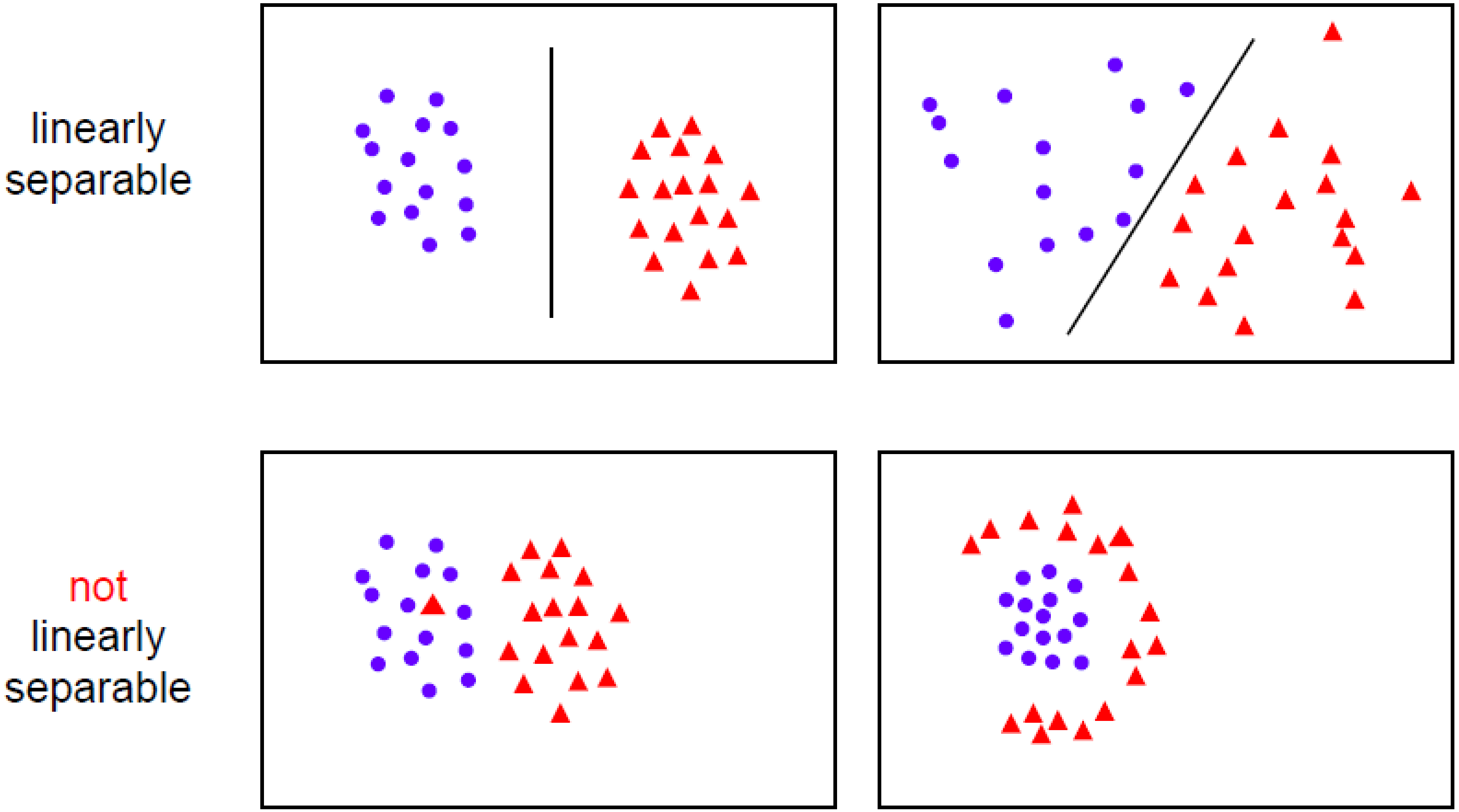
$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \max\left(0, 1 - y_i(\mathbf{w}\cdot\mathbf{x}_i + b)\right)$$





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

Linear separability



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

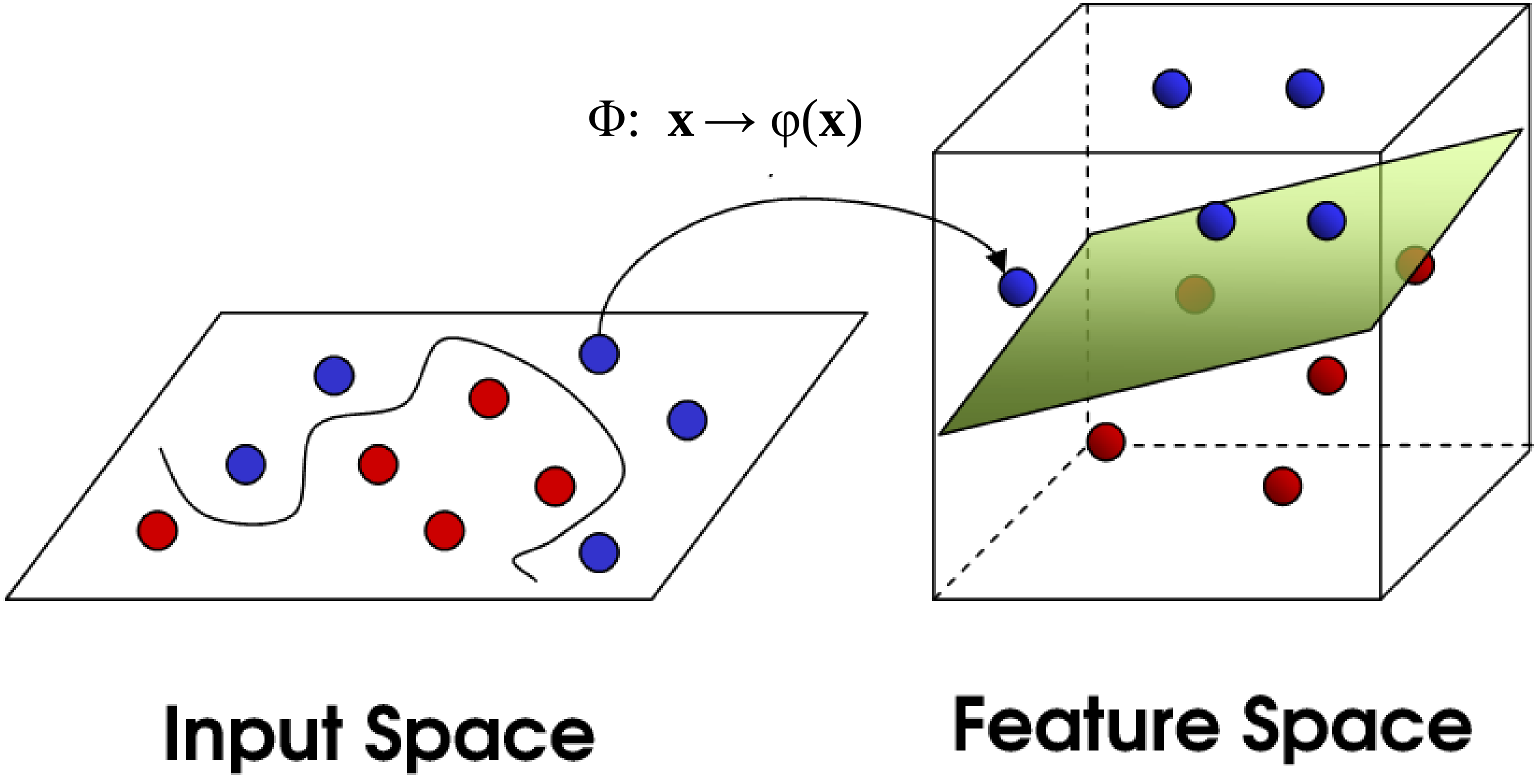
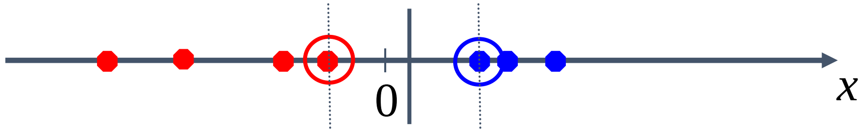


Image source

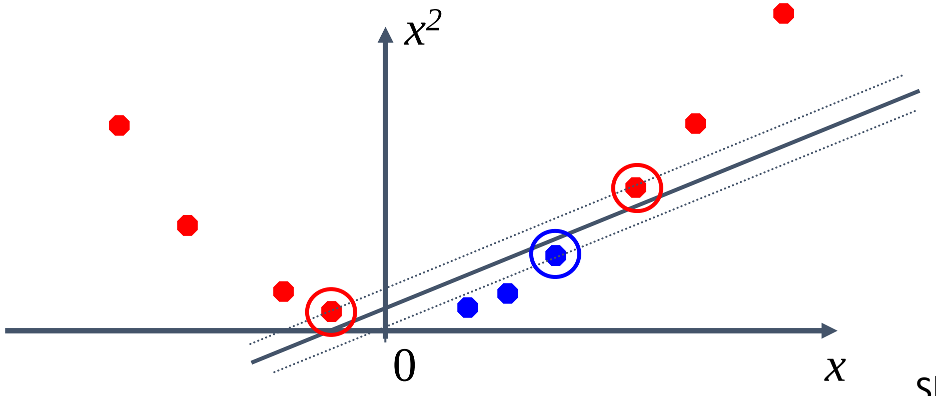
Nonlinear SVMs

- 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

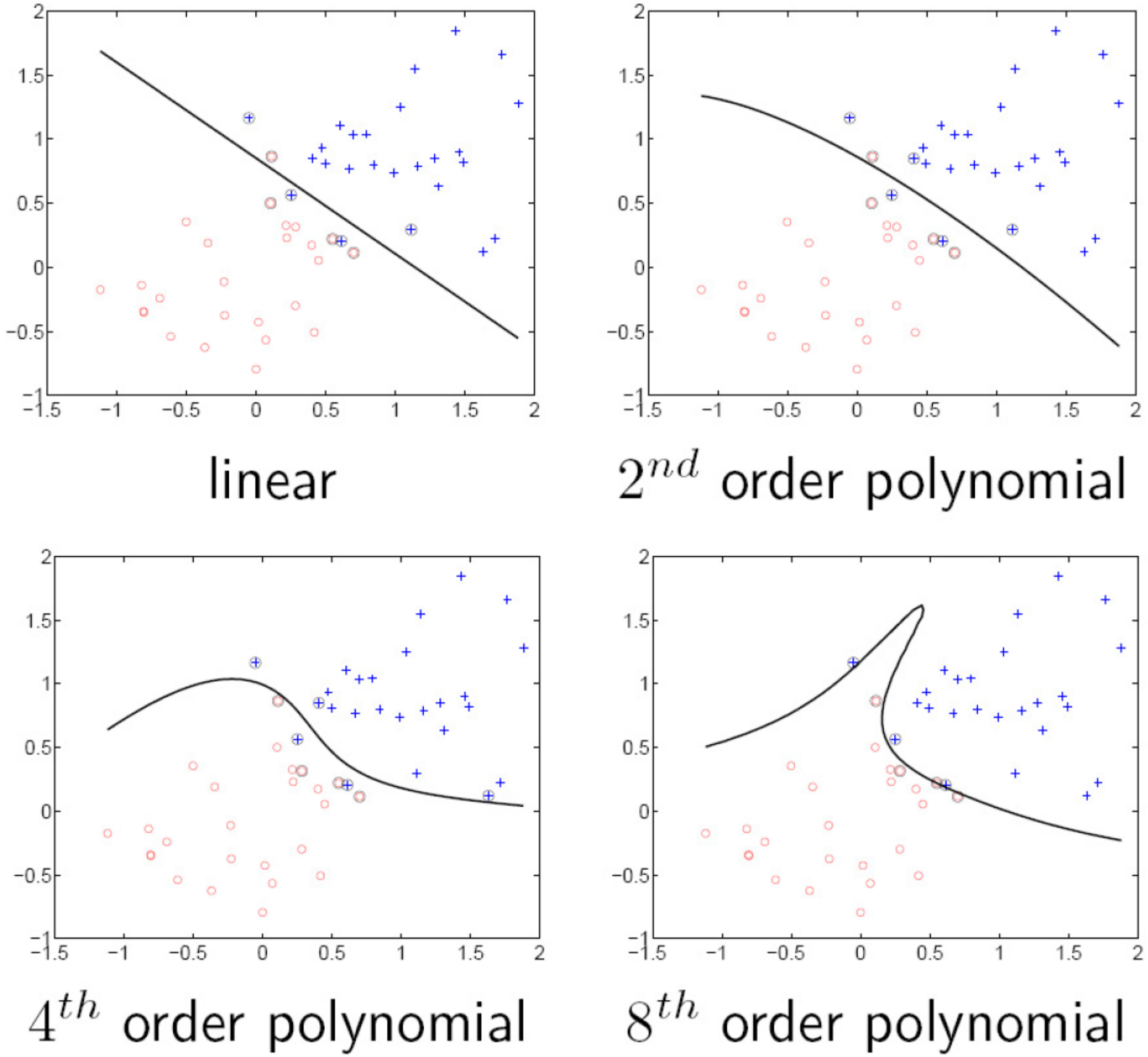
- Instead of explicitly computing the lifting transformation $\varphi(\mathbf{x})$, define a kernel function K such that

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

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

Polynomial kernel:

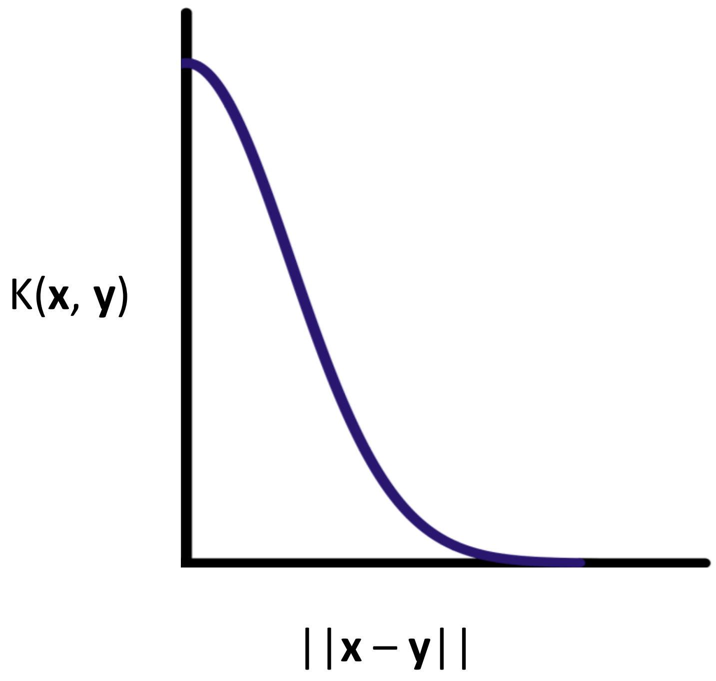
$$K(\mathbf{x},\mathbf{y}) = (c + \mathbf{x}\cdot\mathbf{y})^d$$



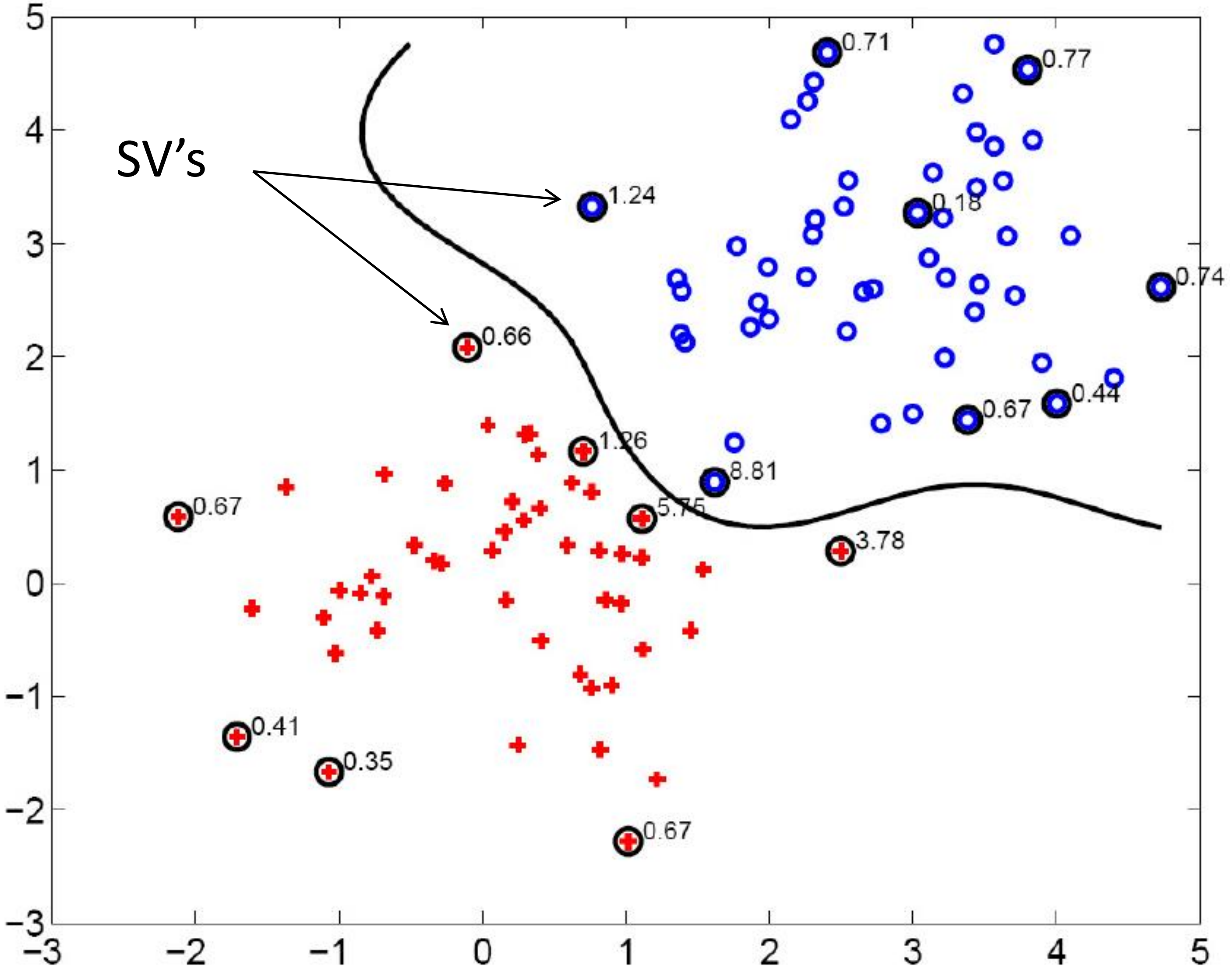
Gaussian kernel

- 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



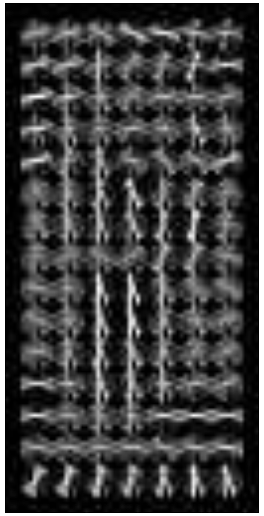
- Demo: http://cs.stanford.edu/people/karpathy/svmjs/demo

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)



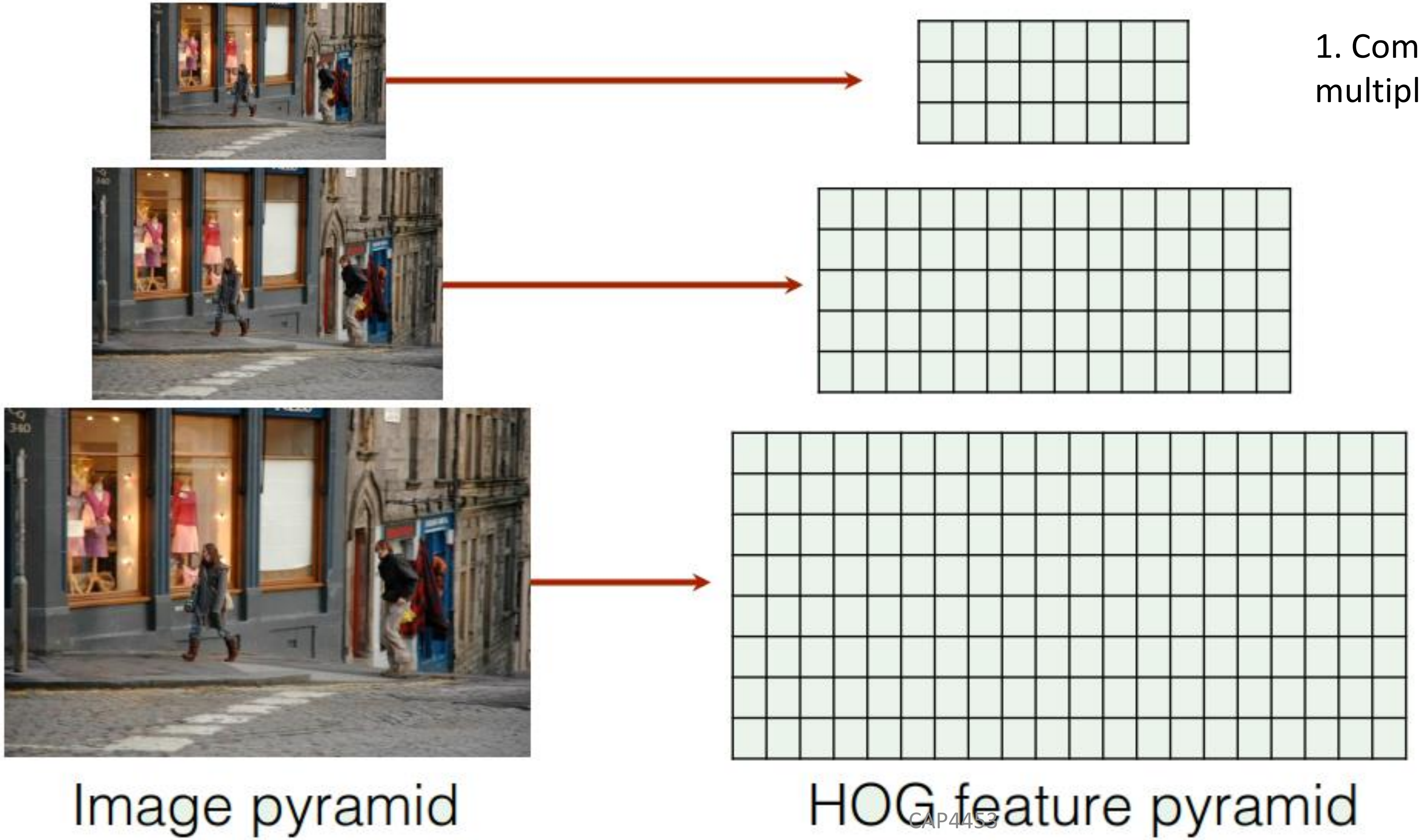
- Histogram of oriented gradients (HoG): Map each grid cell in the input window to a histogram counting the gradients per orientation.
- Train a linear SVM using training set of pedestrian vs. non-pedestrian windows.

Dalal & Triggs, CVPR 2005

The Dalal & Triggs detector

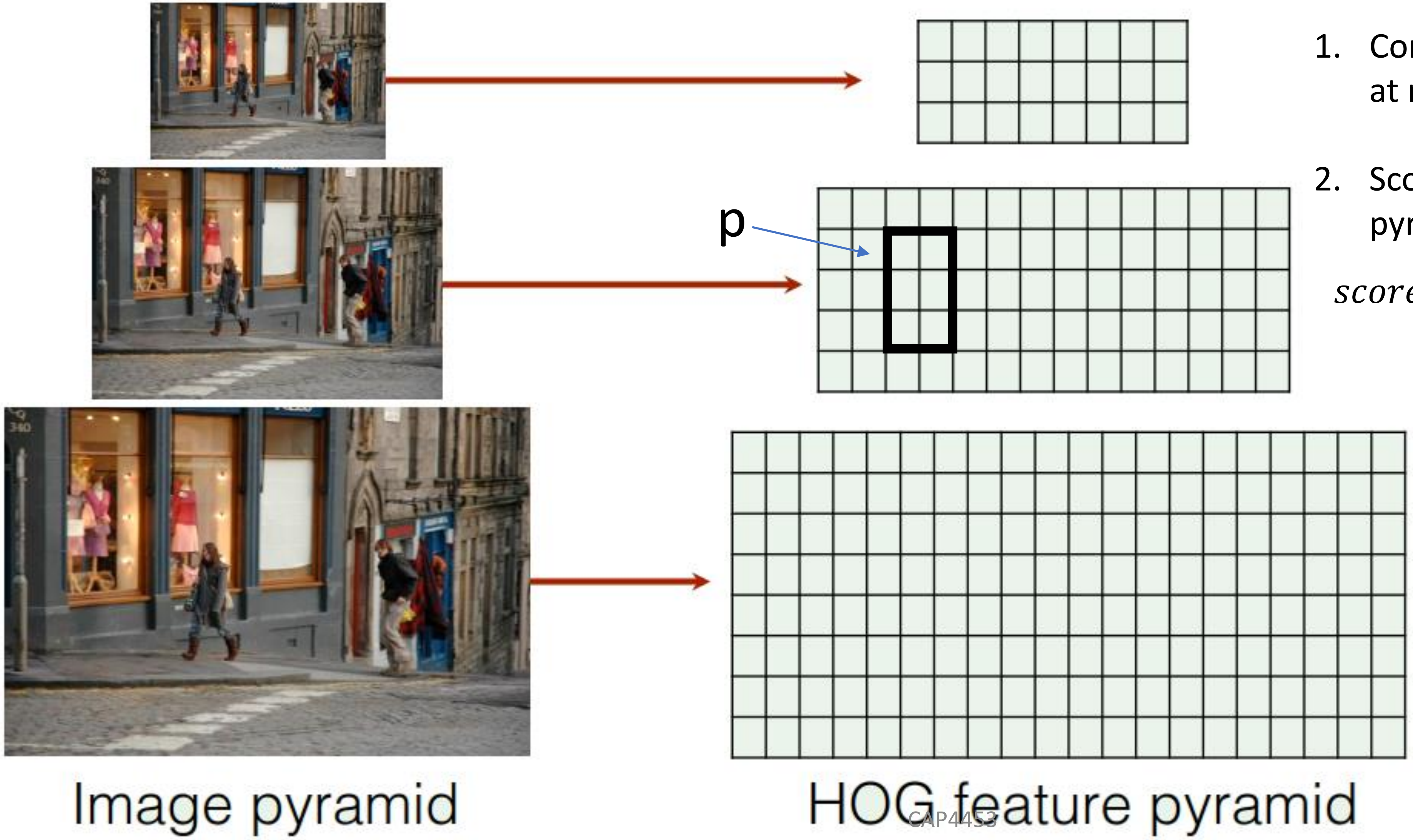
Image pyramid

The Dalal & Triggs detector



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

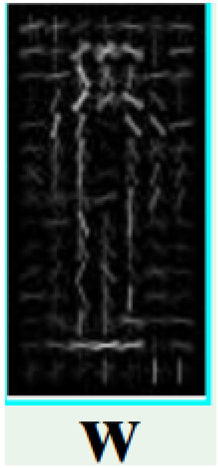
The Dalal & Triggs detector



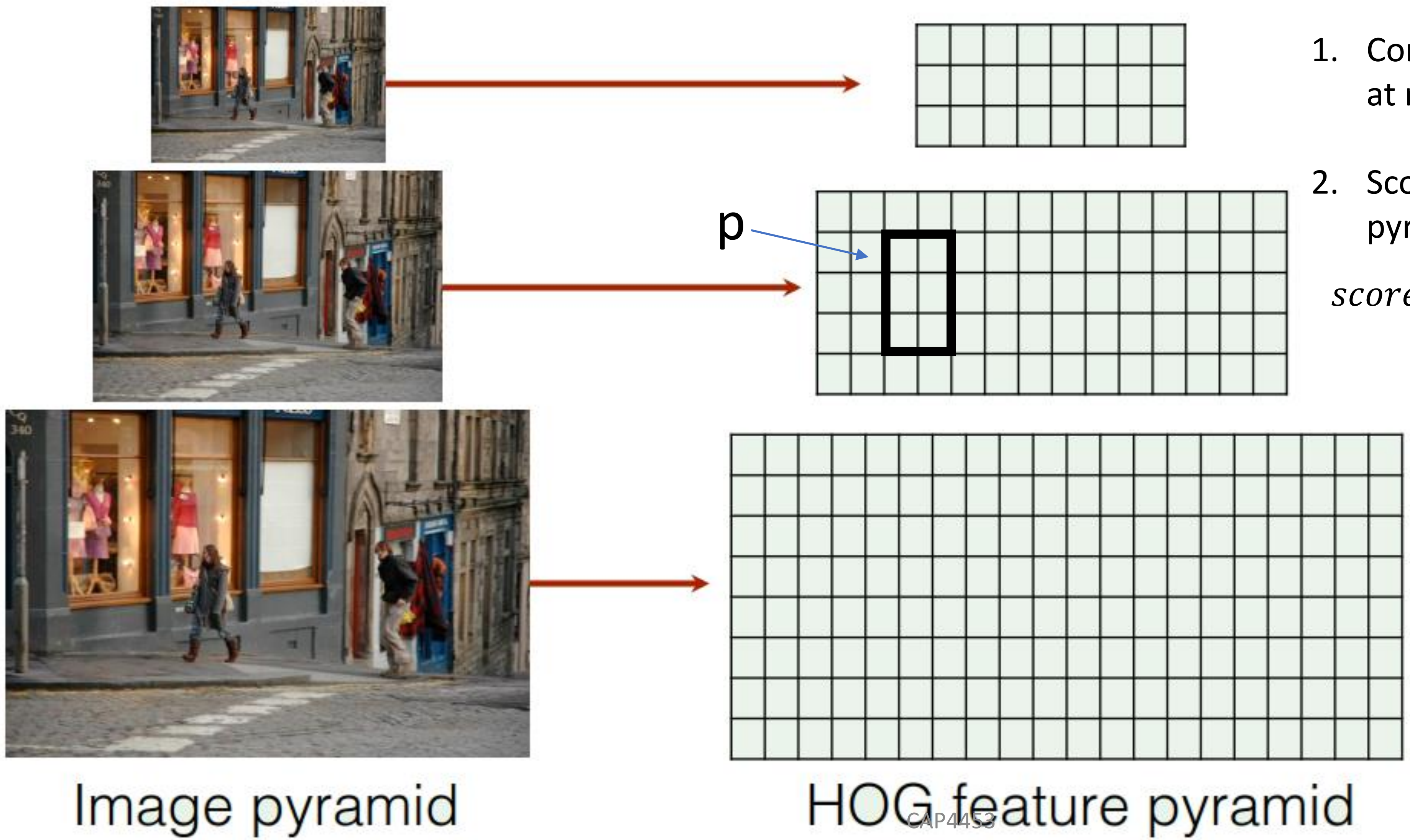
1. Compute HOG of the whole image at multiple resolutions!
2. Score every window of the feature pyramid

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

FROM TRAINING



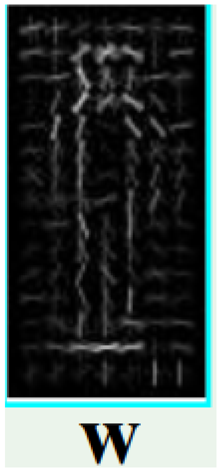
The Dalal & Triggs detector



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



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**
    $discard \leftarrow$ 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 $\text{same}(b_i, b_j) > \boldsymbol{\lambda_{nms}}$ **then** **If both boxes having same IOU**
        if $\text{score}(c, b_j) > \text{score}(c, b_i)$ **then**
          $discard \leftarrow$ 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** **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.**
      $B_{nms} \leftarrow B_{nms} \cup b_i$
  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)](https://github.com/VladKha/object_detector)

Implementation example (car detector)

- Train SVM with HOG features

```
# 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)

```
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

```
# 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])
```

Implementation example (car detector)

```
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

```
# 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])
```

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

Testing (different pyramid levels)

NMS

Before NMS

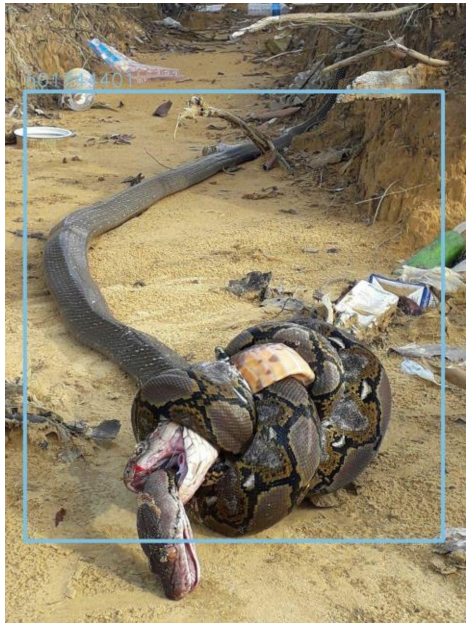
After NMS

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-Maxima Suppression (NMS)
- Implementation examples
- Deformable Part-based Model (DPM)
- **Object detection using deeplearning**

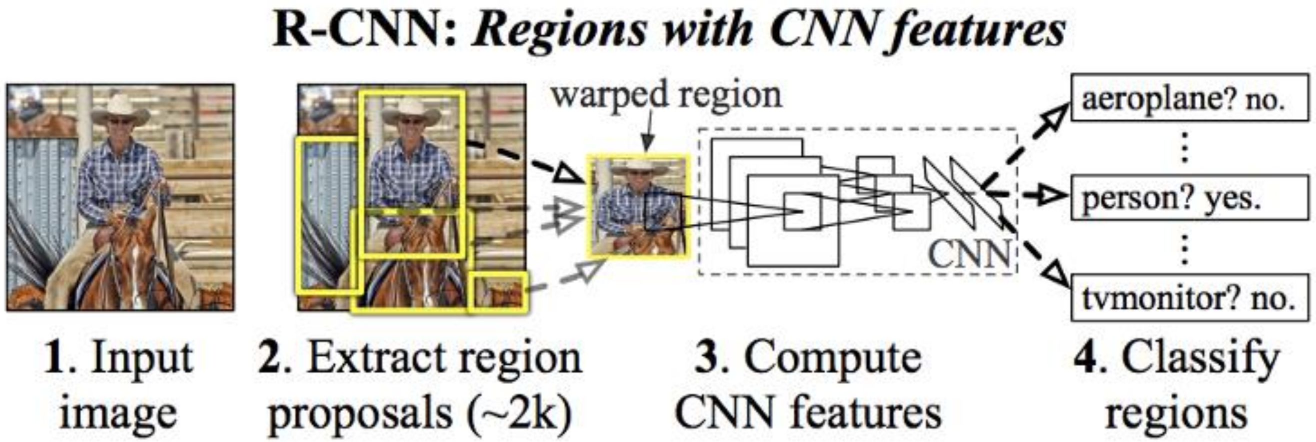
Object detection

- **R-CNN**
- **Fast R-CNN**
- **Faster R-CNN**
- **SSD**
- **YOLO — You Only Look Once**
  - **Multiple versions**



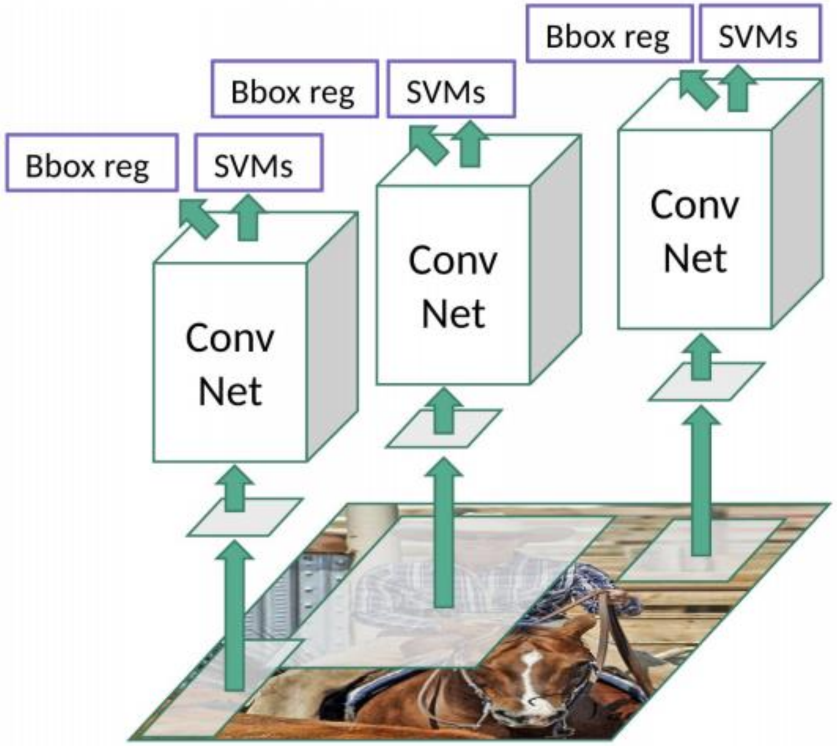
Object detection

R-CNN (2013)



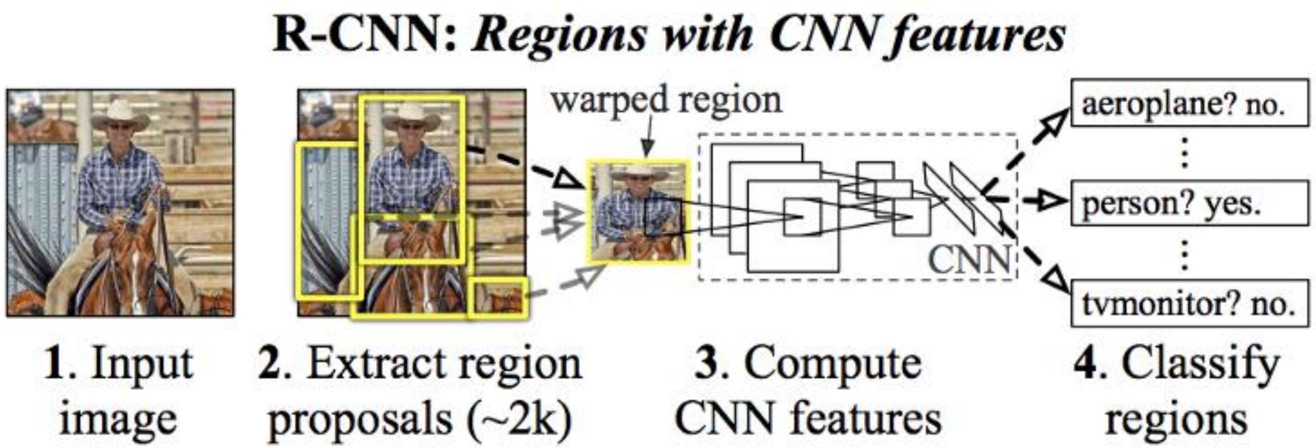
Selective Search:

1. Generate initial sub-segmentation, we generate many candidate regions
2. Use greedy algorithm to recursively combine similar regions into larger ones
3. Use the generated regions to produce the final candidate region proposals



https://arxiv.org/pdf/1311.2524.pdf

Object detection
R-CNN (2013)

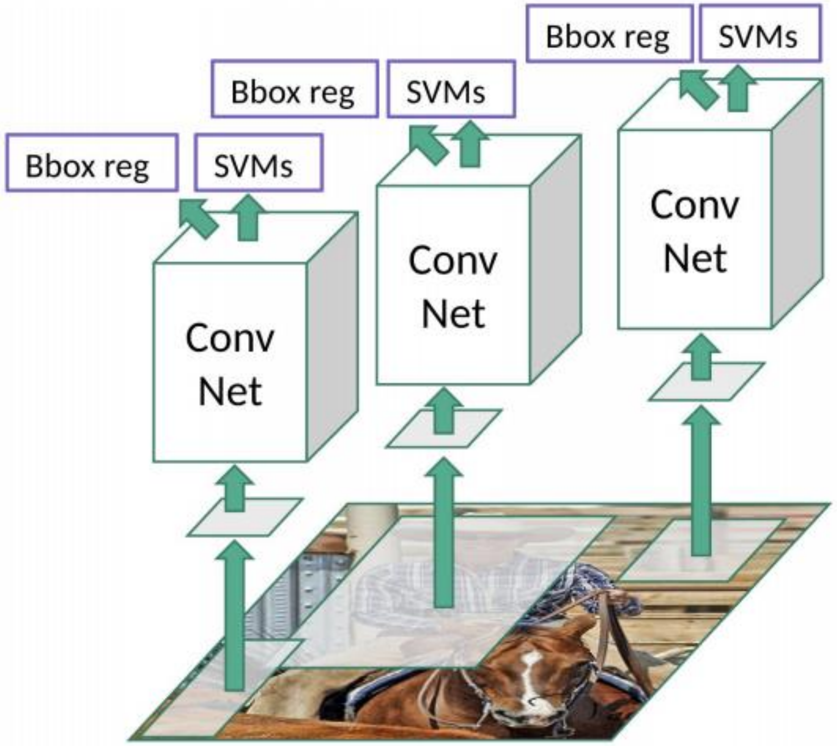


Problems with R-CNN

•It still takes a huge amount of time to train the network as you would have to classify 2000 region proposals per image.

•It cannot be implemented real time as it takes around 47 seconds for each test image.

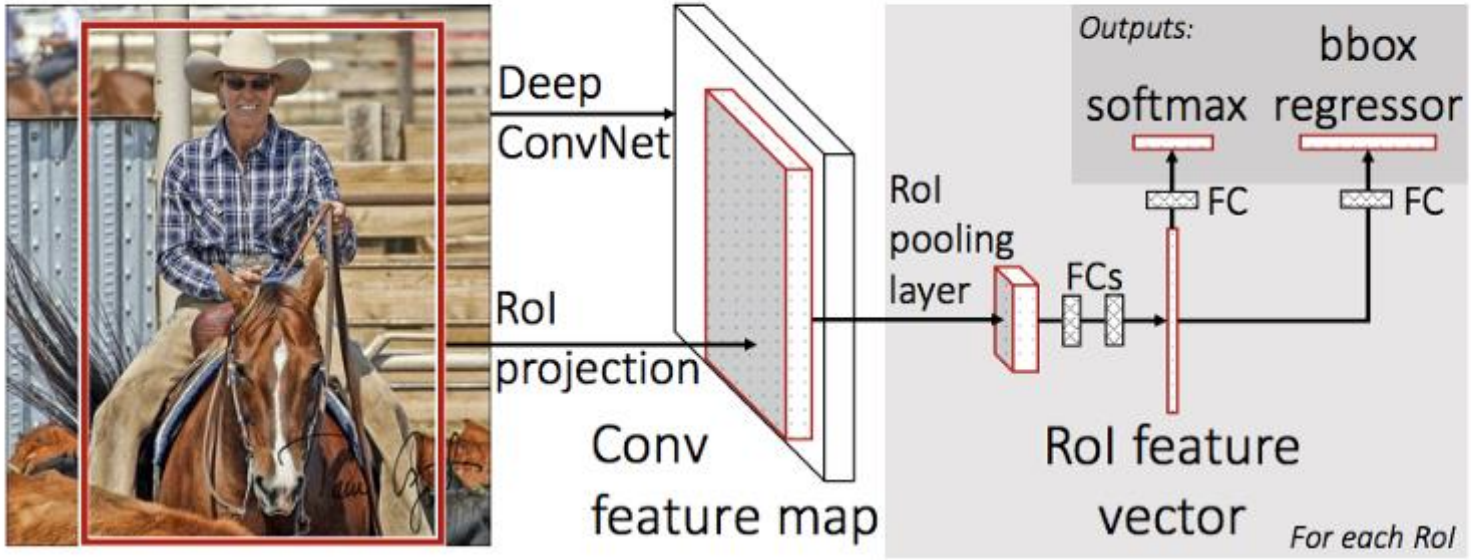
•The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.



https://arxiv.org/pdf/1311.2524.pdf

Object detection

FAST R-CNN (2014)

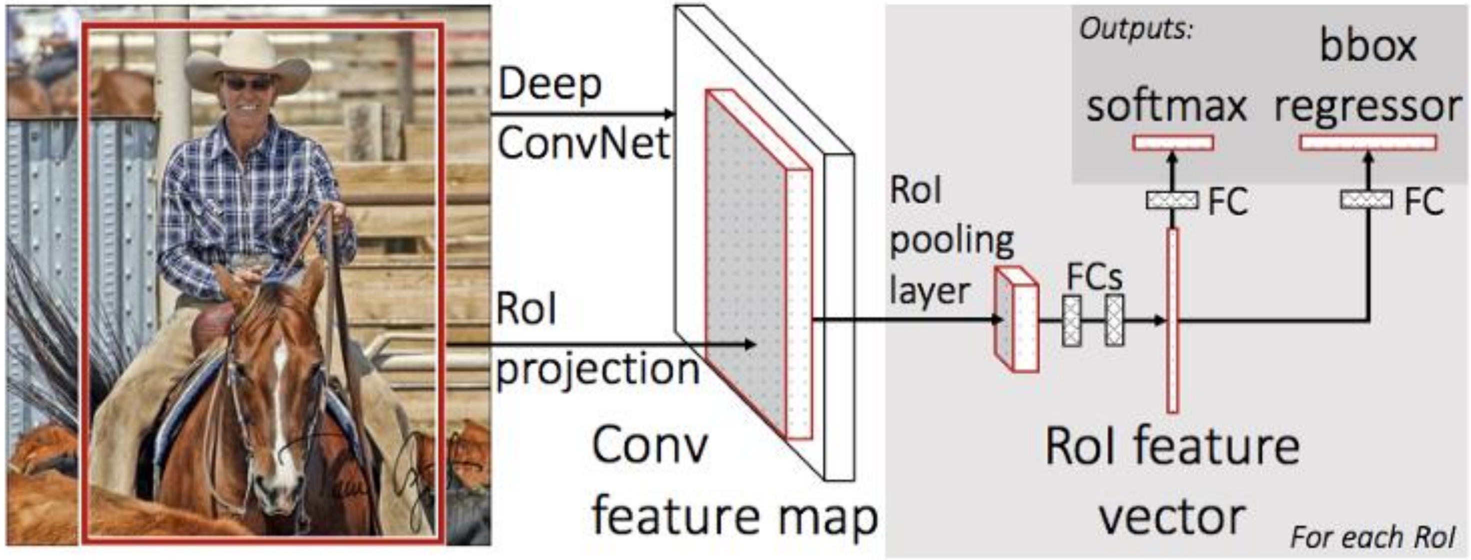


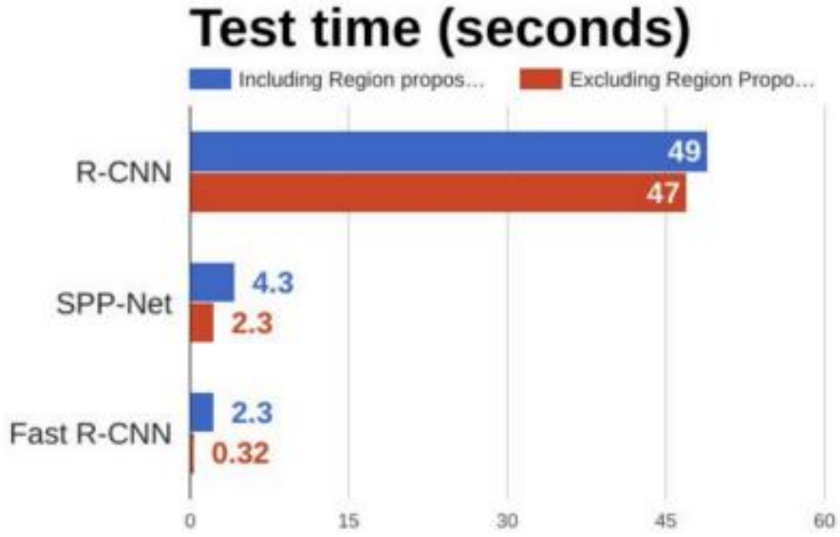
- We feed the input image to the CNN to generate a convolutional feature map. From the convolutional feature map
- We identify the region of proposals and warp them into squares
- Using a RoI pooling layer we reshape them into a fixed size
- they can be fed into a fully connected layer

https://arxiv.org/pdf/1504.08083.pdf

Object detection

FAST R-CNN (2014)



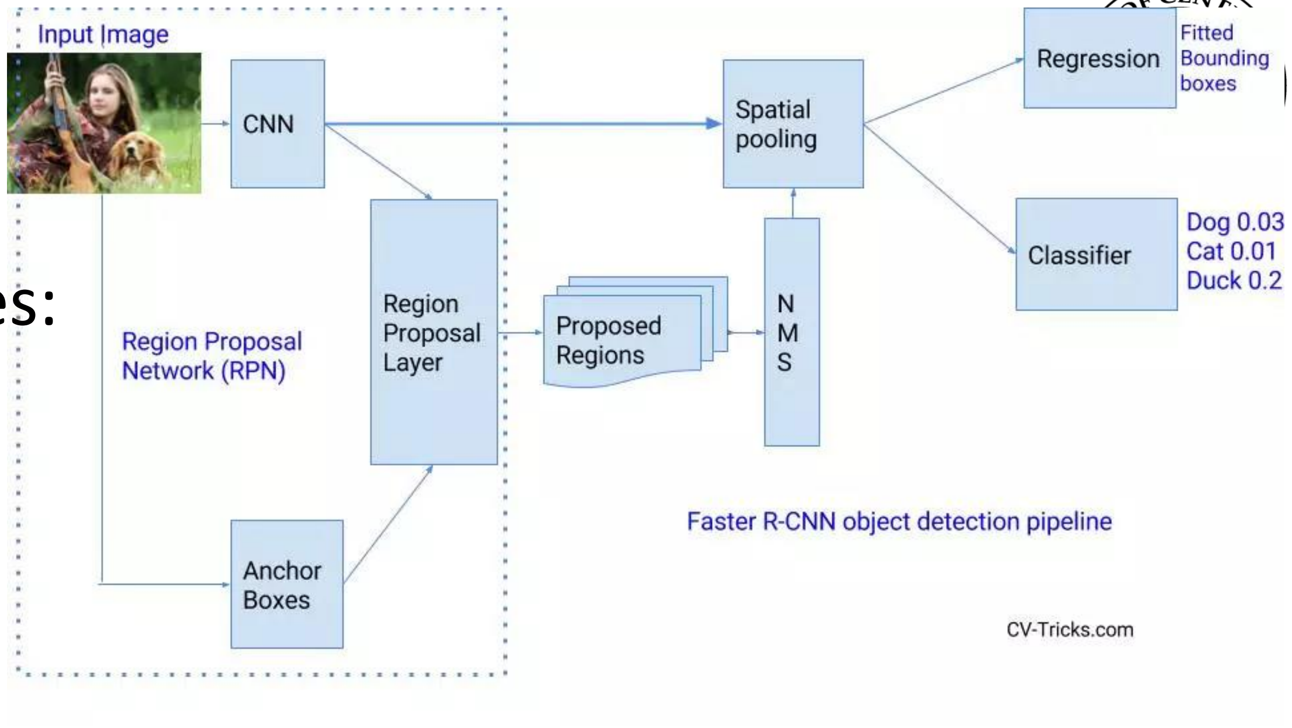


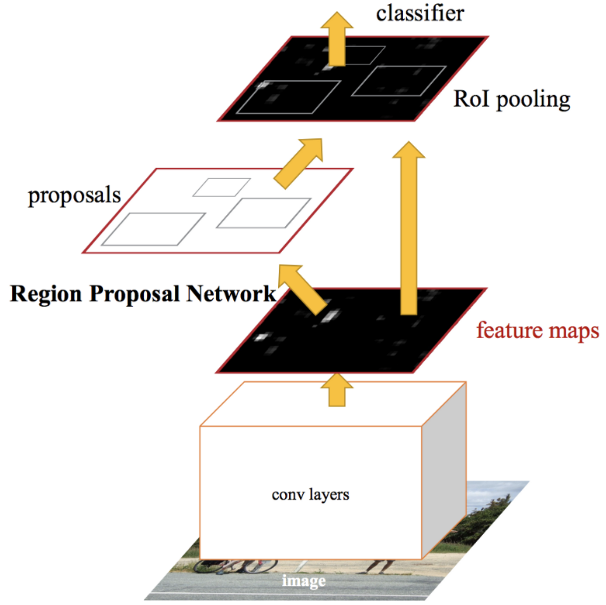
https://arxiv.org/pdf/1504.08083.pdf

Object detection

FASTER CNN (2015)

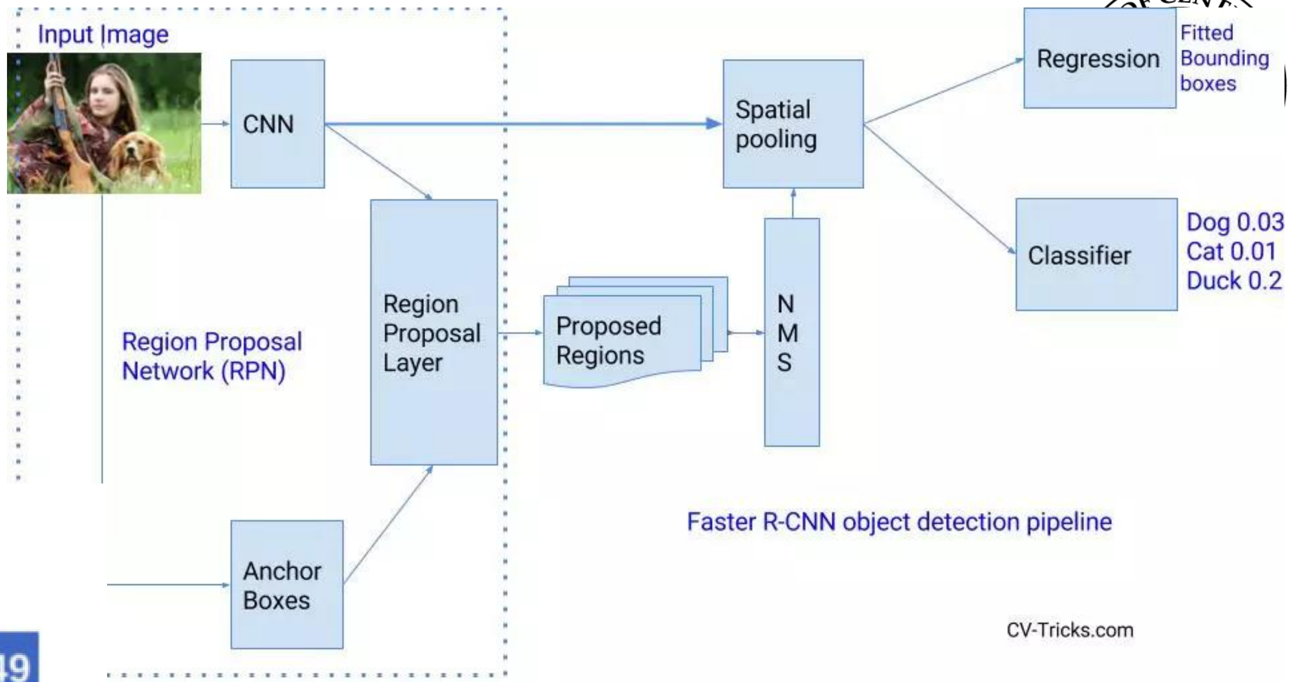
- At each location, the original paper uses:
- 3 kinds of anchor boxes for scale
  - **128x 128, 256×256 and 512×512**.
- **it uses three aspect ratios**
  - **1:1, 2:1 and 1:2**.
- So, In total at each location, we have 9 boxes on which RPN predicts the probability of it being background or foreground.

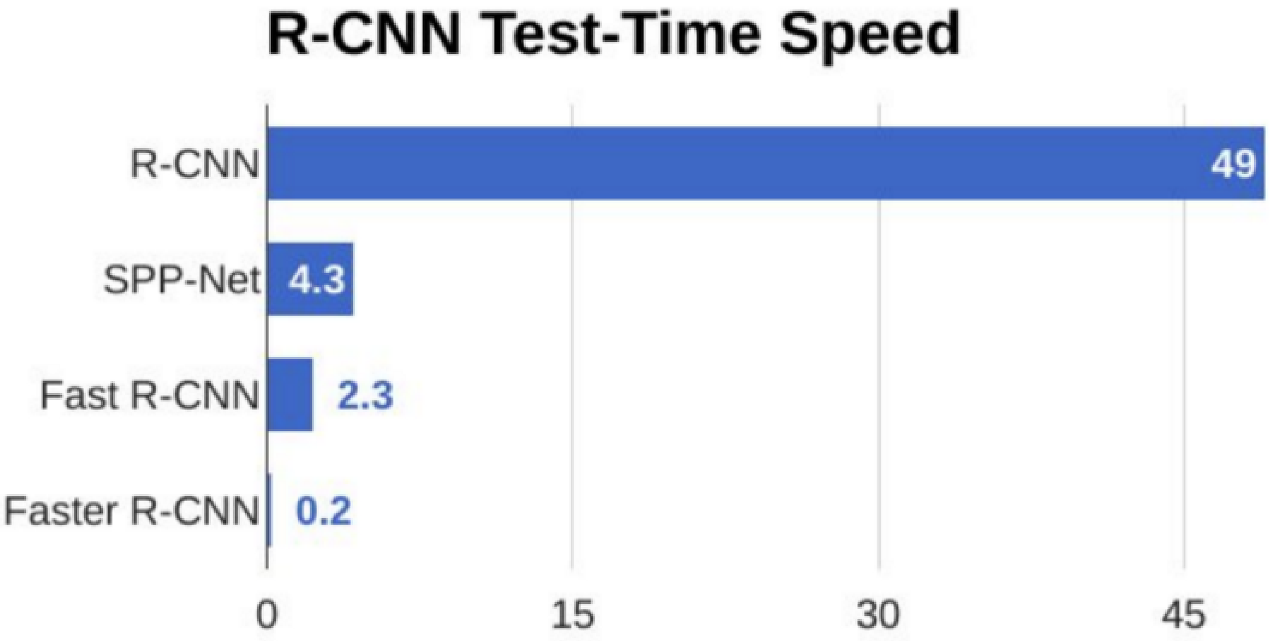




https://arxiv.org/pdf/1506.01497.pdf

Object detection
FASTER CNN (2015)





https://arxiv.org/pdf/1506.01497.pdf

Object detection
YOLO (2015)

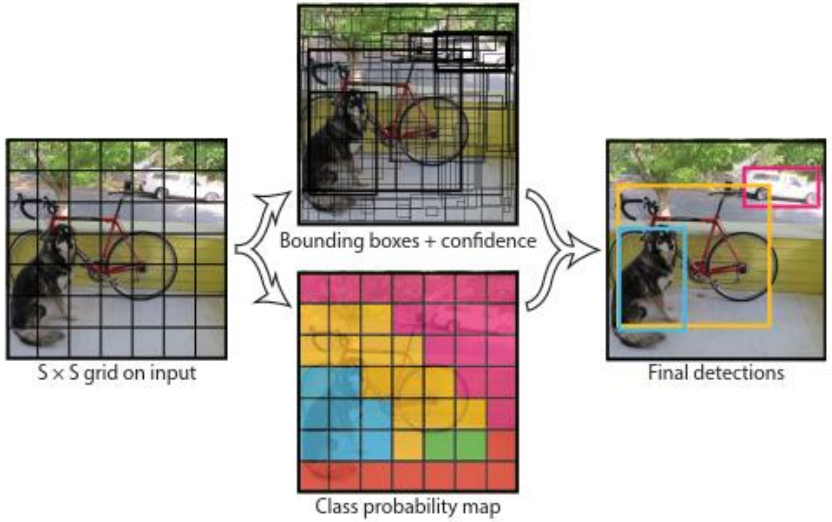


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

- an image and split it into an SxS grid,
- within each of the grid we take B bounding boxes.
- For each of the bounding box,
  - the network outputs a class probability and offset values for the bounding box.
  - The bounding boxes having the class probability above a threshold value is selected and used to locate the object within the image.

https://arxiv.org/pdf/1506.02640v5.pdf

Object detection

YOLO (2015)

For evaluating YOLO on PASCAL VOC, we use $S = 7$, $B = 2$. PASCAL VOC has 20 labelled classes so $C = 20$. Our final prediction is a $7 \times 7 \times 30$ tensor.

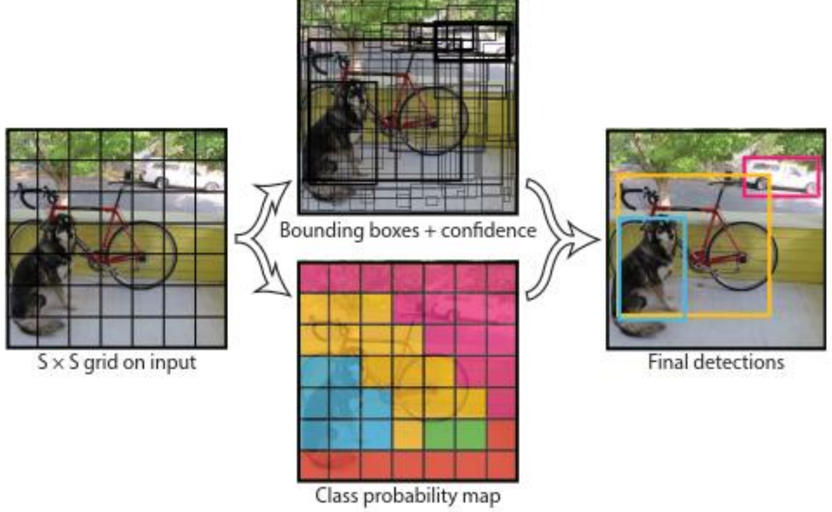


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

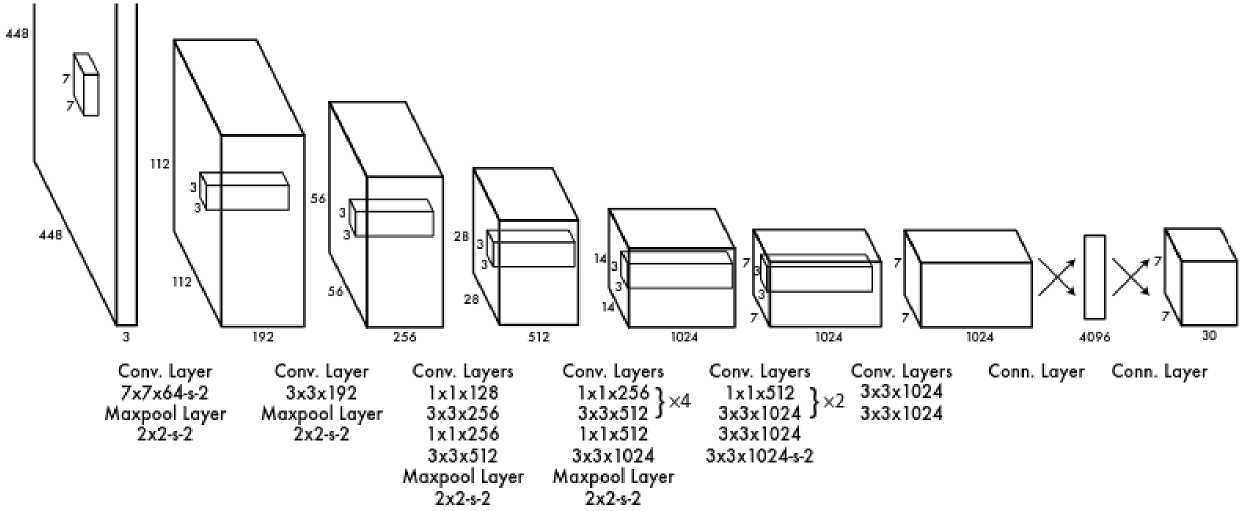
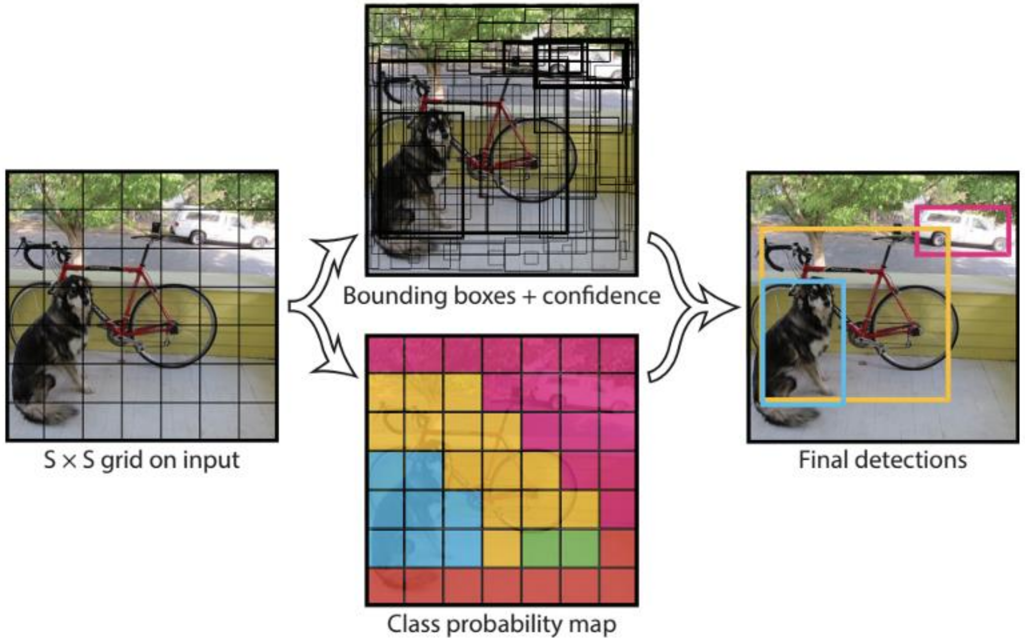


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

https://arxiv.org/pdf/1506.02640v5.pdf

Object detection

YOLO (2015)



- The limitation of YOLO algorithm is that it struggles with small objects within the image, for example it might have difficulties in detecting a flock of birds. This is due to the spatial constraints of the algorithm.

https://arxiv.org/pdf/1506.02640v5.pdf

Object detection

SSD (2016)

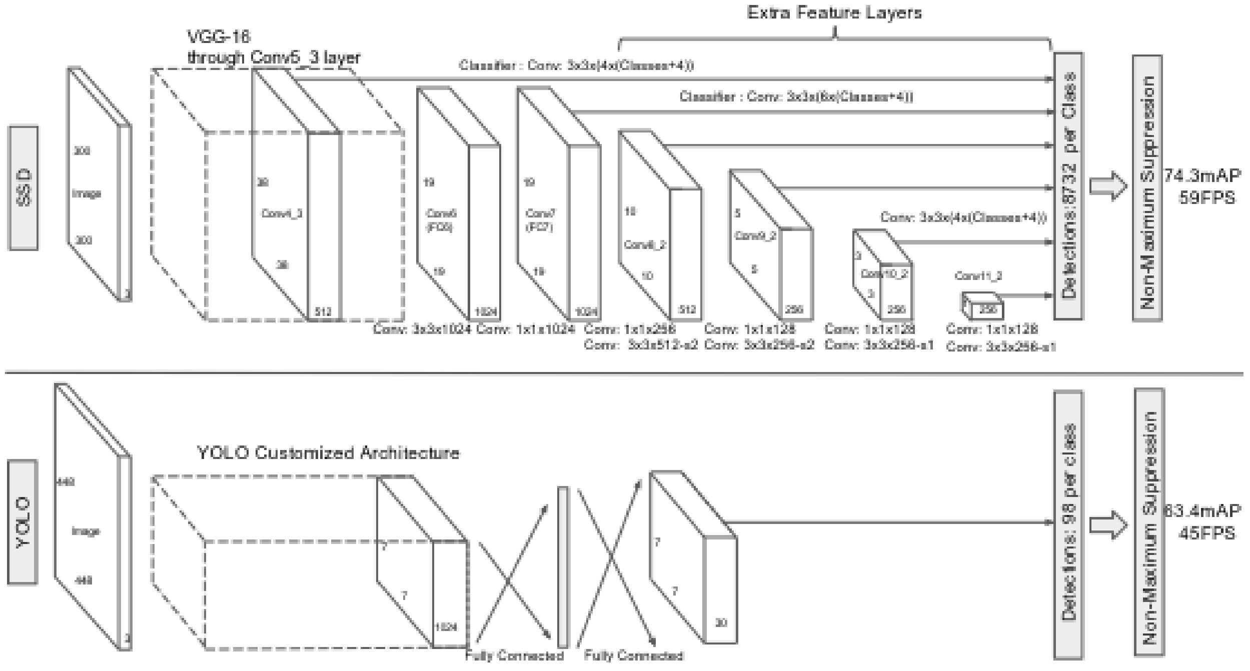


Fig. 2: A comparison between two single shot detection models: SSD and YOLO [5]. Our SSD model adds several feature layers to the end of a base network, which predict the offsets to default boxes of different scales and aspect ratios and their associated confidences. SSD with a 300×300 input size significantly outperforms its 448×448 YOLO counterpart in accuracy on VOC2007 `test` while also improving the speed.

- Multi-scale feature maps for detection (handle scale)
- uses anchor boxes at various aspect ratio similar to Faster-RCNN
- learns the off-set rather than learning the box

https://arxiv.org/pdf/1512.02325

SSD Loss

$$L(x,c,l,g) = \frac{1}{N}(L_{conf}(x,c) + \alpha L_{loc}(x,l,g))$$

$$L_{loc}(x,l,g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx,cy,w,h\}} x_{ij}^{k} \mathrm{smooth}_{\mathrm{L1}}(l_i^m - \hat{g}_j^m)$$

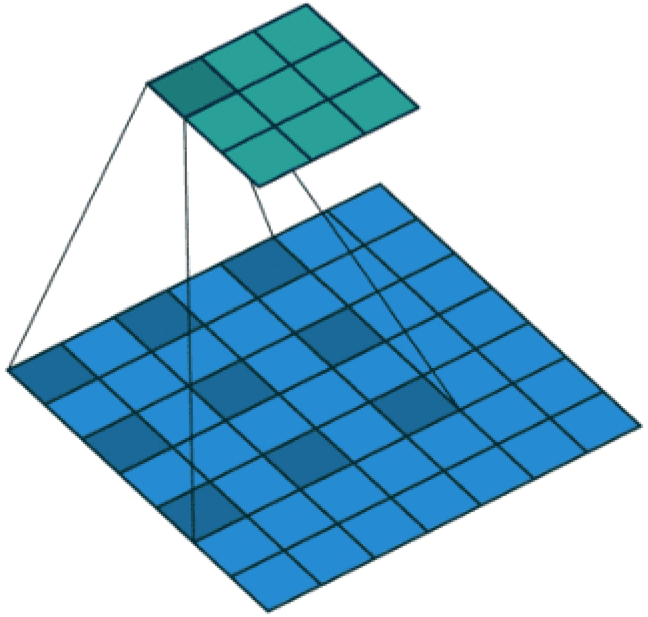
$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^{w} = \log\left(\frac{g_j^w}{d_i^w}\right) \qquad \hat{g}_j^{h} = \log\left(\frac{g_j^h}{d_i^h}\right)$$

$$L_{conf}(x,c) = -\sum_{i \in Pos}^{N} x_{ij}^{p} log(\hat{c}_i^p) - \sum_{i \in Neg} log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

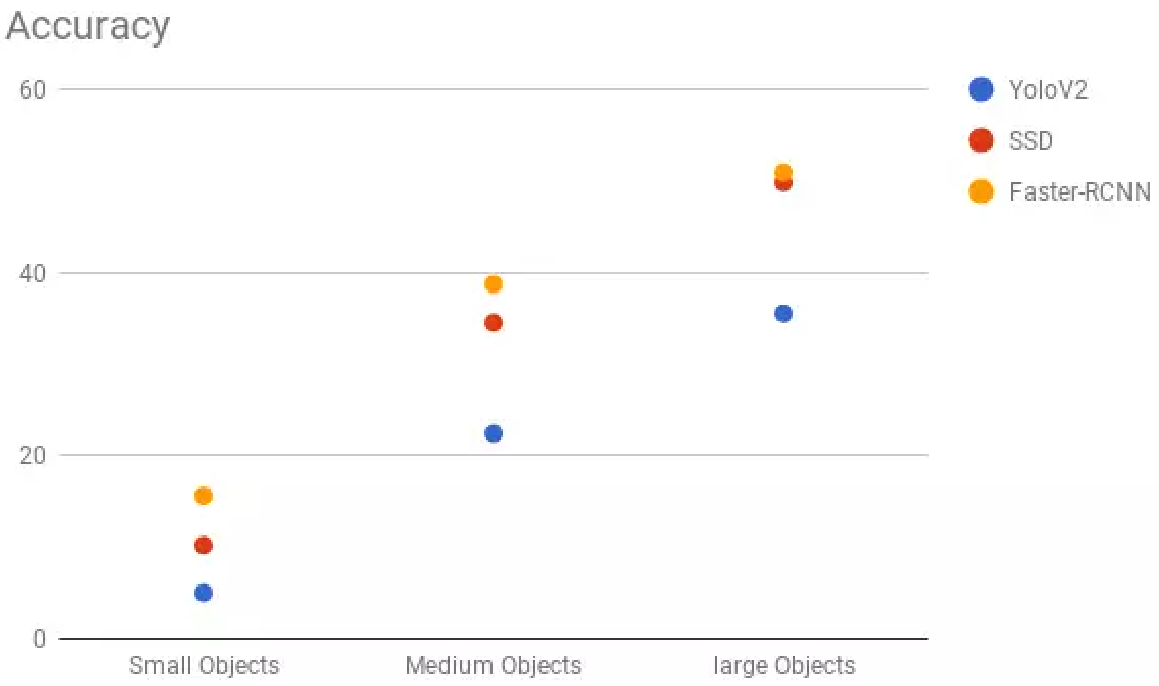
SSD training details

- **Hard Negative Mining**
- **Data Augmentation**
  - entire original input image
  - Sample a patch so that the overlap with objects is 0.1, 0.3, 0.5, 0.7 or 0.9.
  - Randomly sample a patch
- **Atrous Convolution**

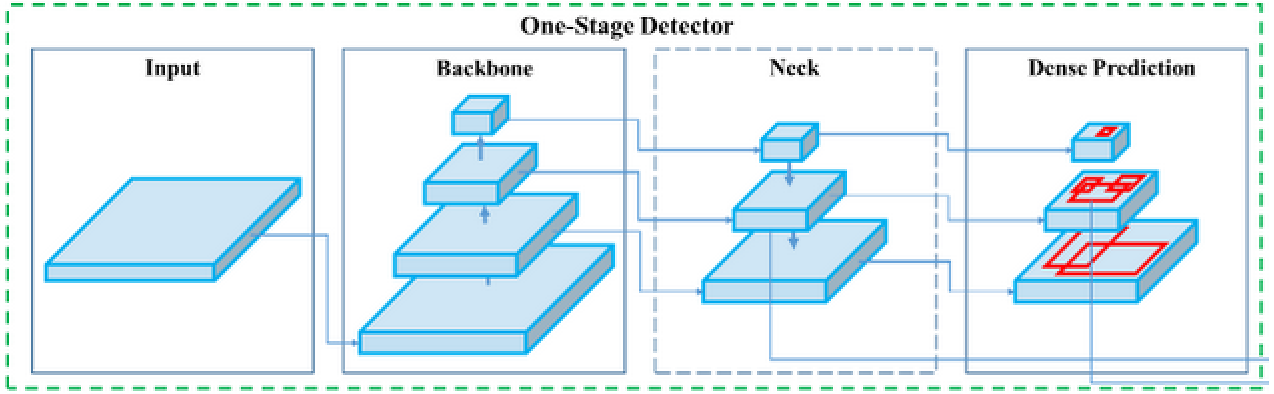


Object detection

COMPARISON OF THESE CLASSICAL METHODS



Modern object detectors (single stage)

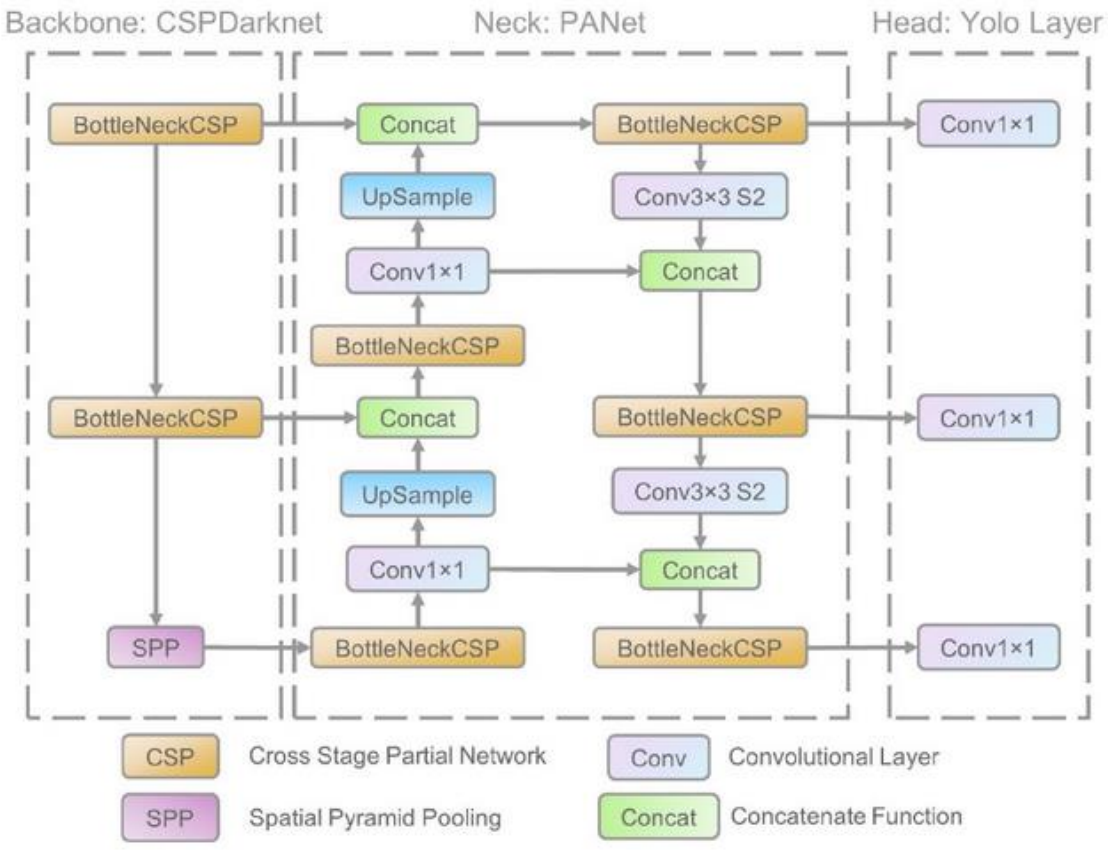


Model Backbone
The backbone is a pre-trained network used to extract rich feature representation for images. This helps **reducing the spatial resolution** of the image and **increasing its feature** (channel) **resolution**.

Model Neck
The model neck is used to extract feature pyramids. This helps the model to generalize well to objects on different sizes and scales.

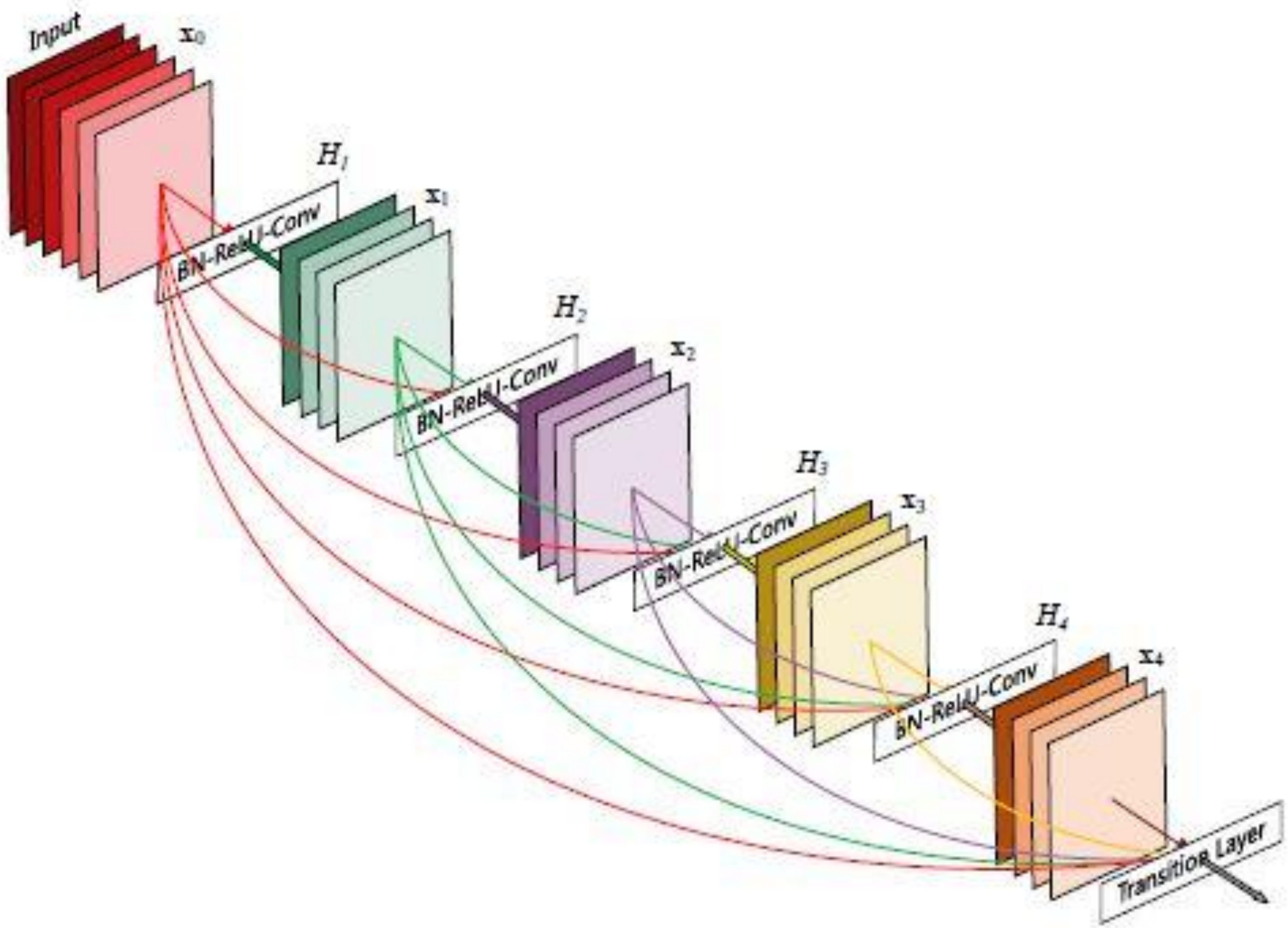
Model Head
The model head is used to perform the final stage operations. It applies anchor boxes on feature maps and render the final output: **classes** , **objectness scores** and **bounding boxes**.

Yolo V5



Backbone: CSPDarknet
Neck: PANet
Head: Yolo Layer
BottleNeckCSP
Concat
BottleNeckCSP
Conv1×1
UpSample
Conv3×3 S2
Conv1×1
Concat
BottleNeckCSP
BottleNeckCSP
Concat
BottleNeckCSP
Conv1×1
UpSample
Conv3×3 S2
Conv1×1
Concat
SPP
BottleNeckCSP
BottleNeckCSP
Conv1×1
CSP
Cross Stage Partial Network
Conv
Convolutional Layer
SPP
Spatial Pyramid Pooling
Concat
Concatenate Function

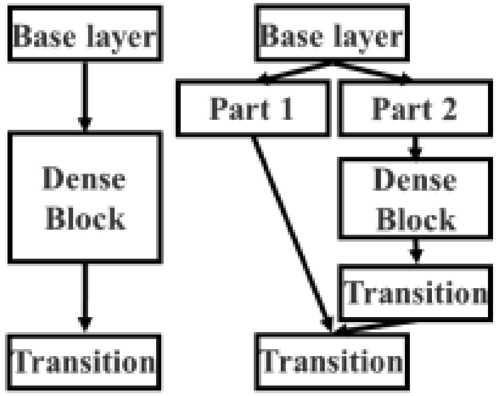
DenseNet


Input
x_0
H_1
BN-ReLU-Conv
x_1
H_2
BN-ReLU-Conv
x_2
H_3
BN-ReLU-Conv
x_3
H_4
BN-ReLU-Conv
x_4
Transition Layer

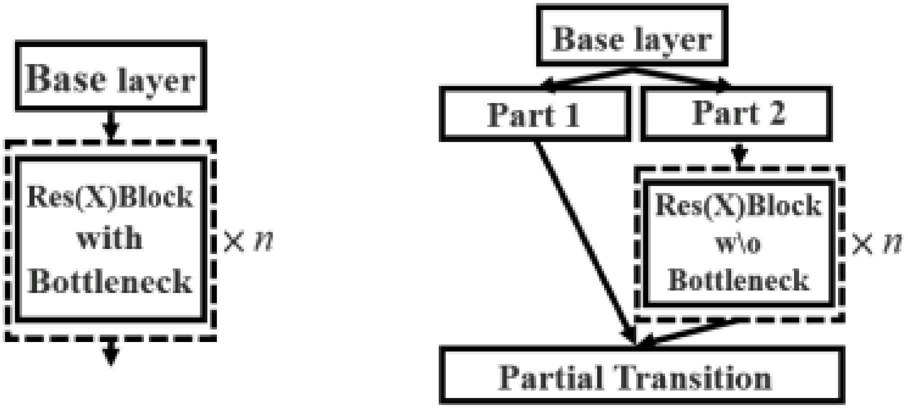
Backbone: Cross Stage Partial Network

it uses residual and dense blocks: overcome the **vanishing gradient problem**.

***CSP** network preserves the advantage of DenseNet's feature reuse characteristics and helps reducing the excessive amount of redundant gradient information by truncating the gradient flow.*



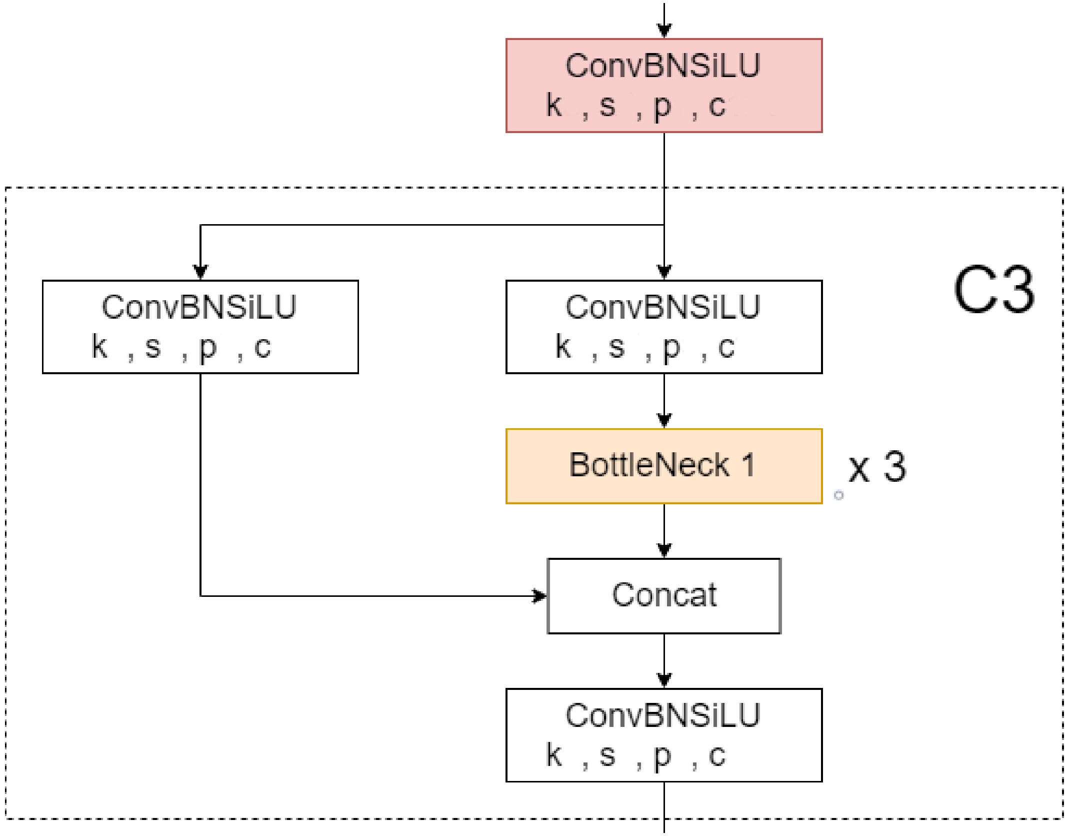
(a) DenseNet (b) CSPDenseNet

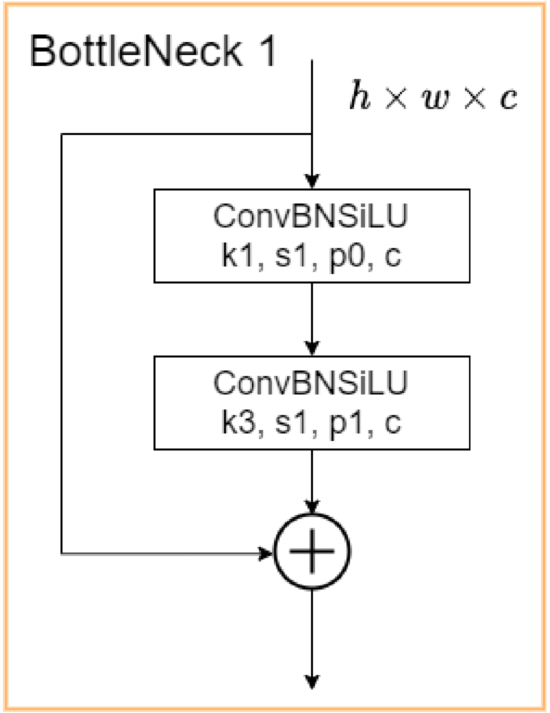


(a) ResNe(X)t (b) CSPResNe(X)t

BottleNeckCSP module architecture.

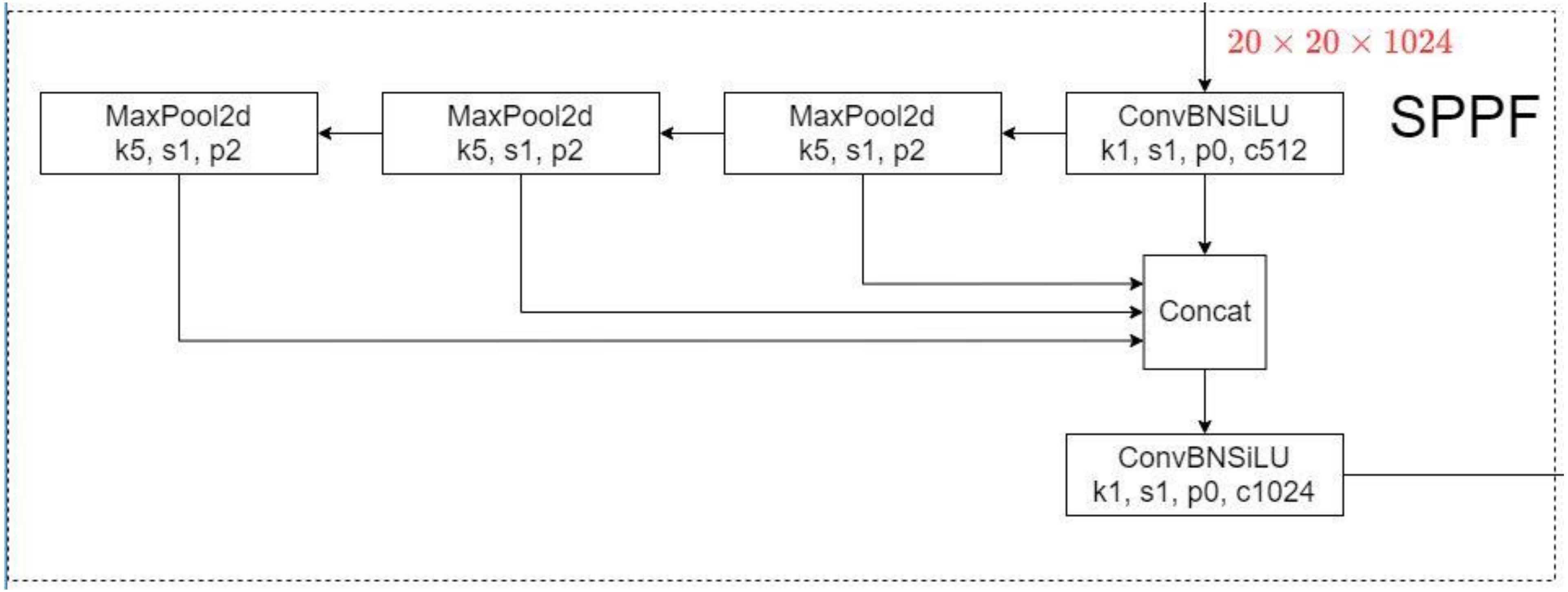
YOLOv5 employs CSPNet strategy to partition the feature map of the base layer into two parts and then merges them through a cross-stage hierarchy





Applying this strategy comes with big advantages to YOLOv5, since it helps reducing the number of parameters and helps reducing an important amount of computation (less FLOPS) which lead to **increasing the inference speed** that is crucial parameter in real-time object detection models.

Spatial Pyramid Pooling (SPP)



Head of the network

YOLOv5 uses the same head as YOLOv3 and YOLOv4.
three convolution layers that predicts the location of the bounding boxes (x,y,height,width), the scores and the objects classes.
The equation to compute the target coordinates for the bounding boxes have changed from previous versions, the difference is shown in the figure bellow.

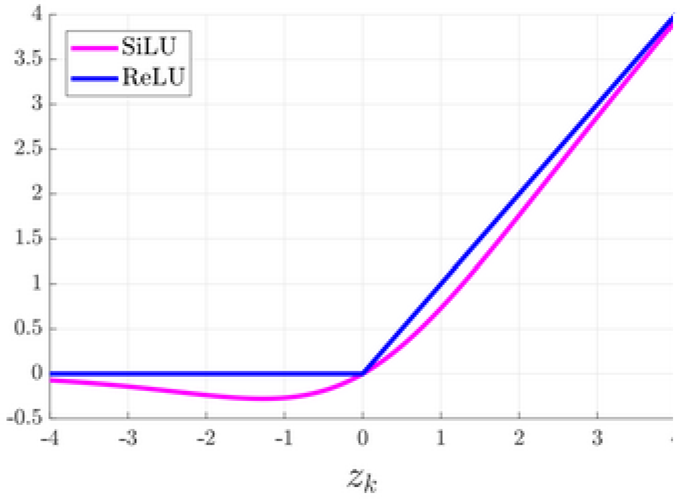
$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w \cdot e^{t_w} \\
b_h &= p_h \cdot e^{t_h}
\end{aligned}
$$

(a)

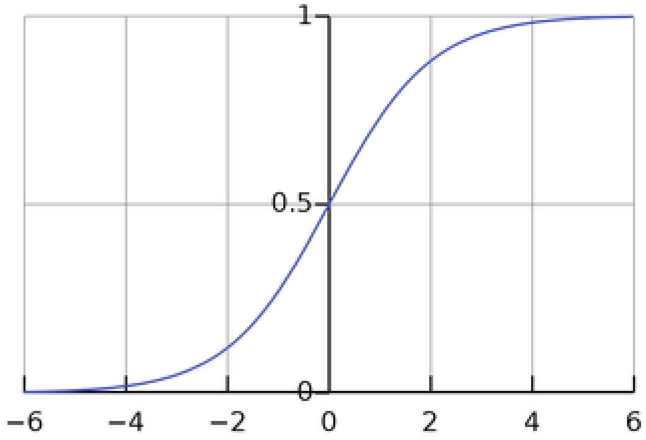
$$
\begin{aligned}
b_x &= (2 \cdot \sigma(t_x) - 0.5) + c_x \\
b_y &= (2 \cdot \sigma(t_y) - 0.5) + c_y \\
b_w &= p_w \cdot (2 \cdot \sigma(t_w))^2 \\
b_h &= p_h \cdot (2 \cdot \sigma(t_h))^2
\end{aligned}
$$

(b)

Activation Function



(a)



(b)

SiLU stands for Sigmoid Linear Unit and it is also called the **swish** activation function.
It has been used with the convolution operations in **the hidden layers**

Sigmoid activation function has been used with the convolution operations in **the output layer**.

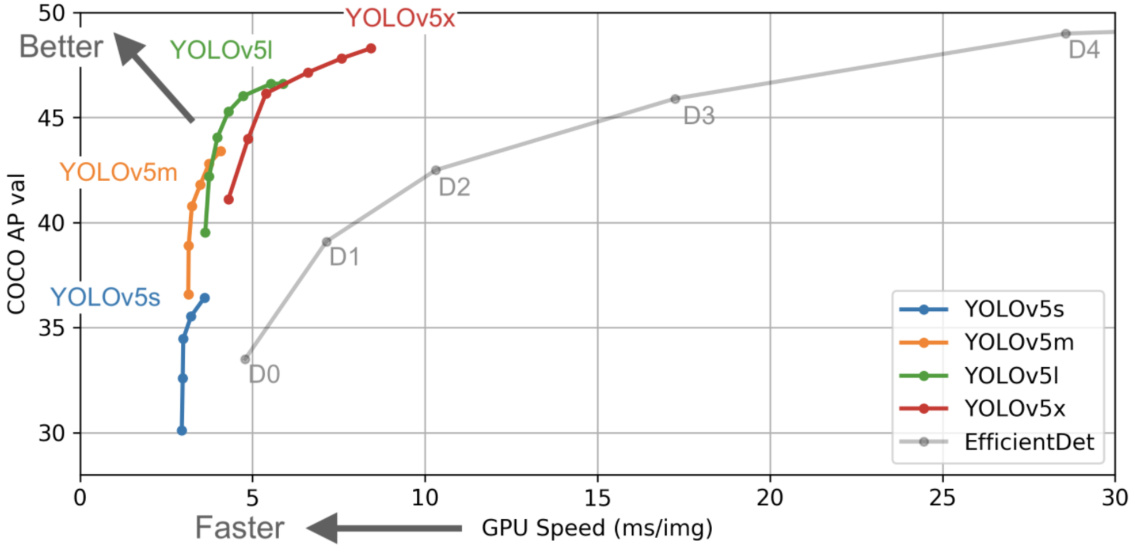
Loss function

- YOLOv5 returns three outputs:
  - the **classes** of the detected objects,
  - their **bounding boxes**
  - **objectness scores**.
- it uses **BCE** (Binary Cross Entropy) to compute the **classes loss** and the **objectness loss**.
- **CIoU** (Complete Intersection over Union) **loss** to compute the **location loss**.

$$Loss = \lambda_1 L_{cls} + \lambda_2 L_{obj} + \lambda_3 L_{loc}$$

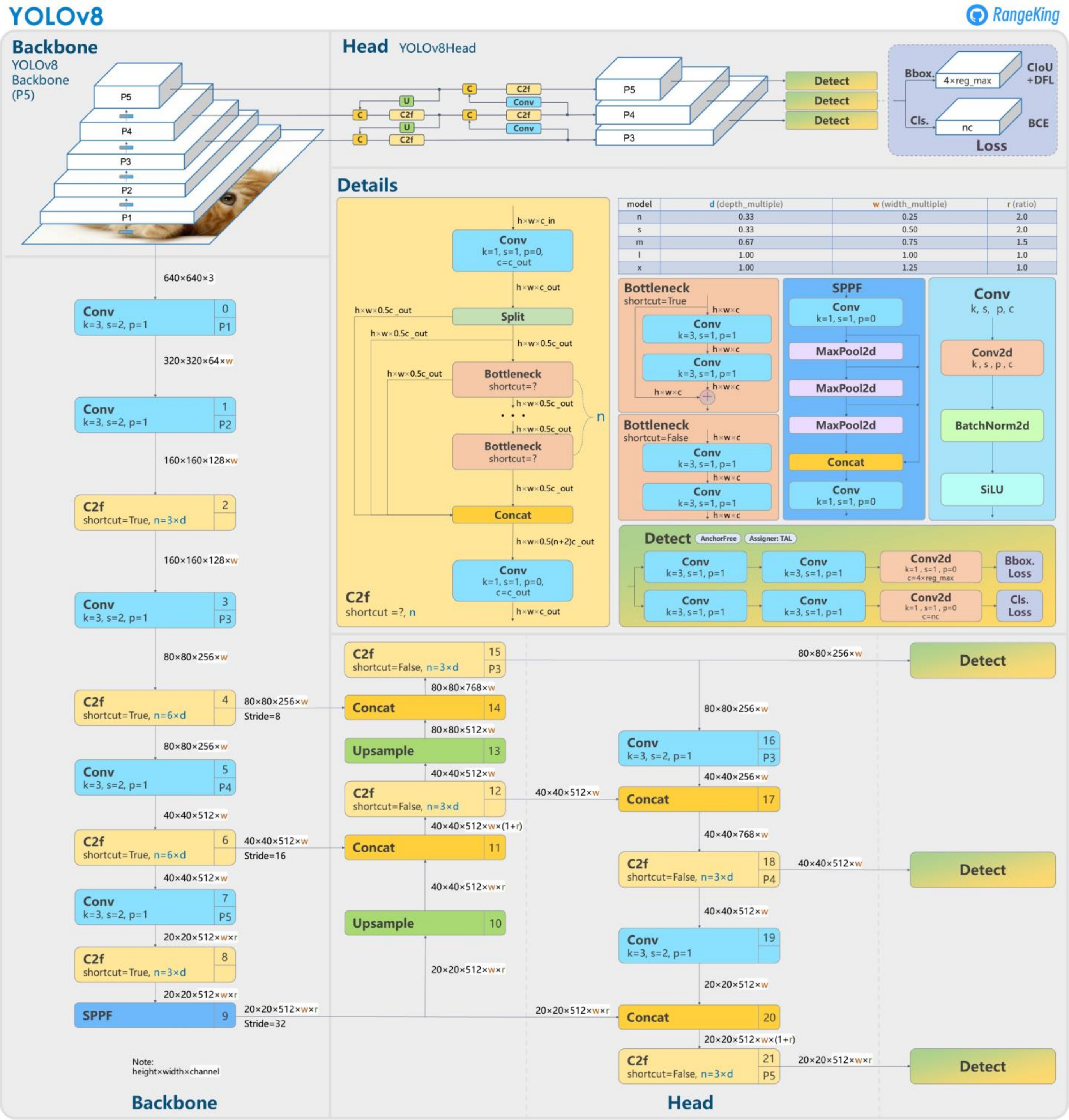
Yolo V5

- **Real-time performance:** excels at real-time object detection, achieving high frame rates on even modest hardware.
- **High accuracy:** YOLOv5 also delivers impressive accuracy. Different versions like “s,” “m,” “l,” and “x” offer a trade-off between speed and accuracy.
- **Flexible and customizable:** The model is open-source and readily customizable for specific tasks and datasets.
- **Easy to use:** YOLOv5 is built with user-friendliness in mind, featuring clear documentation and readily available pre-trained models.



Yolo v8

- **State-of-the-Art:** Delivers cutting-edge accuracy and speed, competing with other top models like Efficient and DETR.
- **Multi-Task Capable:** Handles diverse tasks like object detection, instance segmentation, and image classification within one framework.
- **Anchor-Free Detection:** Eliminates the need for pre-defined anchor boxes, simplifying the architecture and improving accuracy.
- **Streamlined Design:** Easy to use and customize, with readily available pre-trained models and a vibrant community.
- **Open-Source and Scalable:** Freely available under the GNU General Public License and adaptable to various platforms, from edge devices to cloud AI



Changes compared to YOLOv5:

- Replace the C3 module with the C2f module
- Replace the first 6x6 Conv with 3x3 Conv in the Backbone
- Delete two Convs (No.10 and No.14 in the YOLOv5 config)
- Replace the first 1x1 Conv with 3x3 Conv in the Bottleneck
- Use decoupled head and delete the objectness branch

Yolov5 vs Yolov8

Feature	YOLOv5	YOLOv8
Architecture	Anchor-based	Anchor-free
Neck Module	Convolutional connection layers present	Convolutional connection layers removed
Head Module	Single head for class and bounding box predictions	Split head for class and bounding box predictions
Objectness Prediction	Outputs abjectness score	No objectness output, directly predicts center point and size of bounding boxes
Loss Function	Focal Loss + IOU Loss	TAL (Tangent-Aided Loss) + DFL (Dynamic Focal Loss)

Feature	YOLOv5	YOLOv8
Accuracy (mAP50)	Varies depending on model size and dataset	Generally higher than YOLOv5 for similar model sizes
Speed (FPS)	Varies depending on model size and hardware	Generally faster than YOLOv5 for similar model sizes
Model Size	Generally larger than YOLOv8	More compact, requires fewer parameters
User Interface/Experience (UI/UX)	Less user-friendly	Significantly improved UI/UX, easier to use and customize
Training Ease	More complex training regime	Simpler training process, often converges faster
Community Support	Large and active community	Growing community, but not as large as YOLOv5 yet

Yolo v8?

Speed comparisons:

- **Benchmark results:** Some benchmarks show YOLOv8 as slightly faster for certain model sizes, particularly on image inference. However, for video and live camera applications, YOLOv5 often holds the edge in speed.
- **Individual results may vary:** Hardware, dataset size, and specific model configurations can significantly affect performance. What's faster for one person might not be faster for another.

Beyond speed:

- **Accuracy:** YOLOv8 generally demonstrates slightly higher accuracy on object detection tasks compared to YOLOv5.
- **Model size:** YOLOv8 models tend to be smaller and have fewer parameters, potentially leading to faster training times and lower deployment memory requirements.
- **Ease of use:** Both frameworks are user-friendly with extensive documentation and community support.

Questions?