NYU TANDON SCHOOL OF ENGINEERING

AI4CE

Robot Perception

Object Detection

Dr. Felix Juefei Xu and Dr. Chen Feng

cfeng@nyu.edu

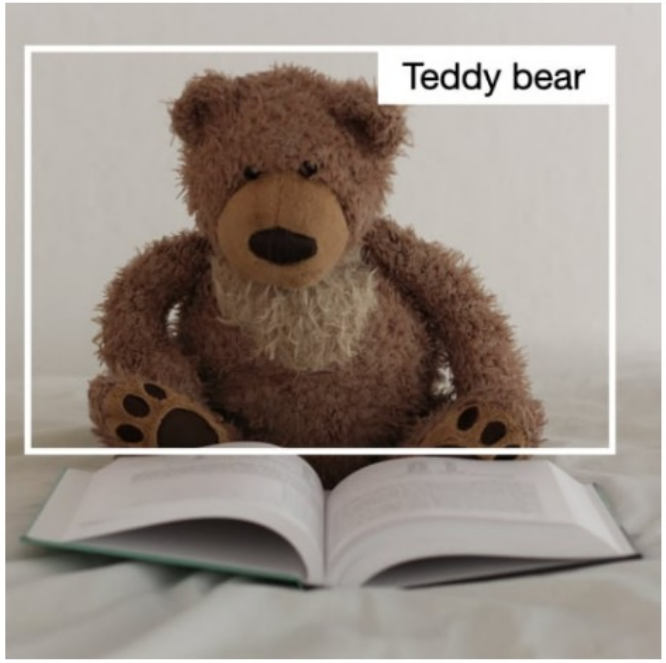
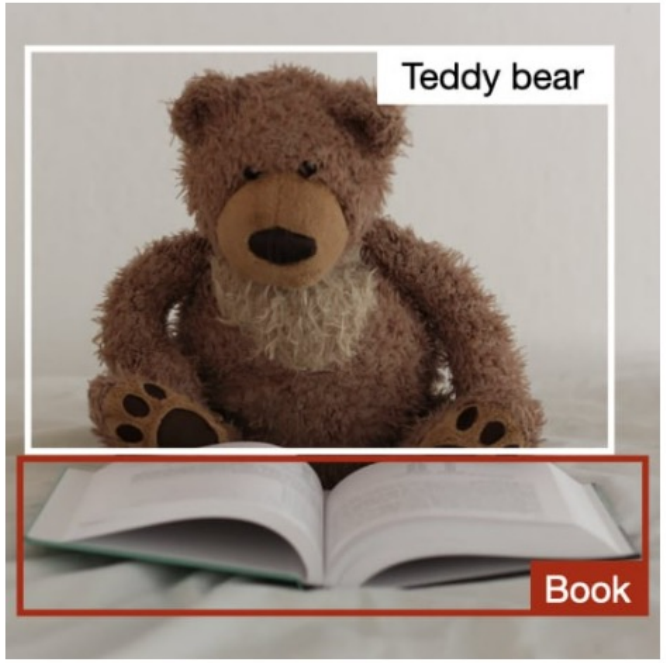
ROB-GY 6203, Fall 2022

Overview

- Before Deep Learning
  - + AdaBoost
  - + HOG
- Deep Learning for Object Detection
  - ++ R-CNN/Fast R-CNN/Faster R-CNN
  - + Anchor-based Region Proposal
  - + Single-stage Detector
  - + Anchor-free Detector

- *: know how to code
- ++: know how to derive
- +: know the concept

Different Types of Object Recognition

Image classification	Classification w. localization	Detection
		
• Classifies a picture • Predicts probability of object	• Detects an object in a picture • Predicts probability of object and where it is located	• Detects up to several objects in a picture • Predicts probabilities of objects and where they are located

Classification

- Classification algorithms are supervised algorithms to predict categorical labels

- Differs from regression which is a supervised technique to predict real-valued labels

Formal problem statement:

- **Produce a function that maps**

  $C : \mathcal{X} \rightarrow \mathcal{Y}$

- **Given a training set**

  $\{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n)\}$

  $y \in \mathcal{Y}$ label

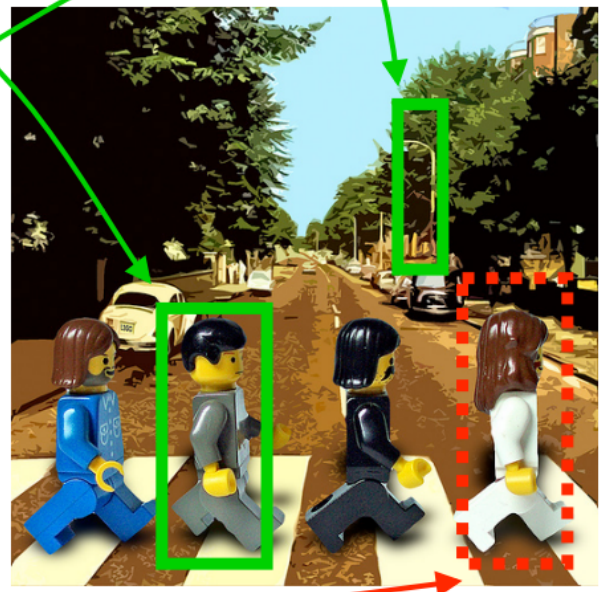
  $\mathbf{x} \in \mathcal{X}$ training sample

Slides from Arras et. al.

Classification

Error types

Predicted value of the classifier	True value: T	True value: N
T'	True Positive	False Positive
N'	False Negative	True Negative



- **Precision** = TP / (TP + FP)
  i.e., among all the detected boxes, how many of them are correct?
- **Recall** = TP / (TP + FN)
  i.e., among all the GT boxes, how many of them are correctly detected/recalled?

Many more measures...

Slides from Arras et. al.

Boosting

- An **ensemble** technique (a.k.a. committee method)
- Supervised learning: given <samples x, labels y>

- Learns an accurate **strong classifier** by combining an ensemble of inaccurate "rules of thumb"
- **Inaccurate** rule $h_i(x)$: "weak" classifier, weak learner, basis classifier, feature
- **Accurate** rule H(x): "strong" classifier, final classifier

- Other ensemble techniques exist: Bootstrap Aggregation (Bagging), Voting, Mixture of Experts, etc.

Slides from Arras et. al.

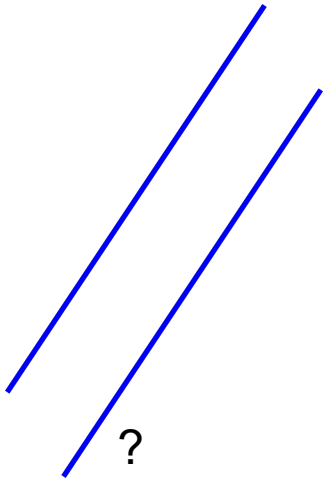
AdaBoost (Adaptive Boosting)

- Most popular algorithm: **AdaBoost**

  [Freund et al. 95], [Schapire et al. 99]

- Given an ensemble of weak classifiers $h(x_i)$, the combined strong classifier $H(x_i)$ is obtained by a **weighted majority voting scheme**

$$f(x_i) = \sum_{t=1}^{T} \alpha_t h_t(x_i) \qquad H(x_i) = \operatorname{sgn}\big(f(x_i) \big)$$



- AdaBoost in Robotics:

  [Viola et al. 01], [Treptow et al. 04], [Martínez-Mozos et al. 05], [Rottmann et al. 05] , [Monteiro et al. 06] , [Arras et al. 07]

Slides from Arras et. al.

Why is AdaBoost interesting?

- 1. It tells you what the **best "features"** are
- 2. What the **best thresholds** are, and
- 3. How to **combine them to a classifier**

- AdaBoost can be seen as a **principled feature selection strategy**
- AdaBoost is a **non-linear** classifier
- Has good generalization properties: can be proven to maximize the margin
- Quite **robust** to overfitting
- Very **simple** to implement

Slides from Arras et. al.

Weak Classifiers

- Prerequisite: weak classifier must be better than chance
  - error < 0.5 in a binary classification problem

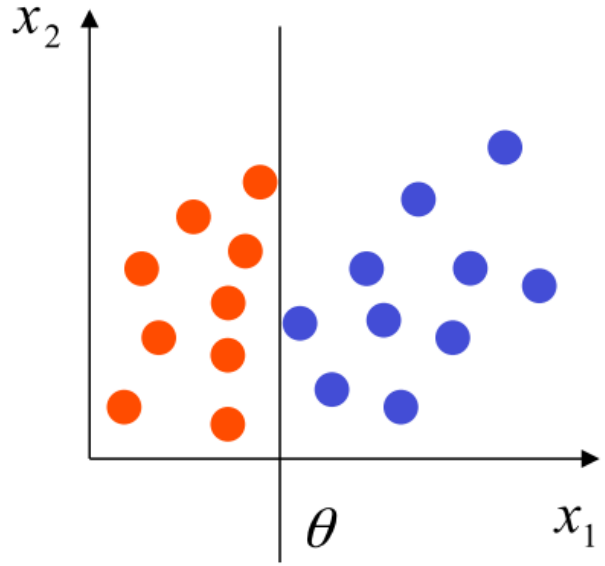
- Possible Weak Classifiers:
  - Decision stump: Single axis-parallel partition of space
  - Decision tree: Hierarchical partition of space
  - Multi-layer perceptron: General non-linear function approximators
  - Support Vector Machines (SVM): Linear classifier with RBF Kernel

- Trade-off between diversity among weak learners versus their accuracy.
- **Decision stumps are a popular choice**

Decision stump

- Simple-most type of **decision tree**
- Equivalent to linear classifier defined by affine hyperplane
- Hyperplane is orthogonal to axis with which it intersects in threshold θ
- Commonly not used on its own
- Formally,

$$h(x; j, \theta) = \begin{cases} +1 & x_j > \theta \\ -1 & \text{else} \end{cases}$$



where x is (d-dim.) training sample, j is dimension

Slides from Arras et. al.

AdaBoost

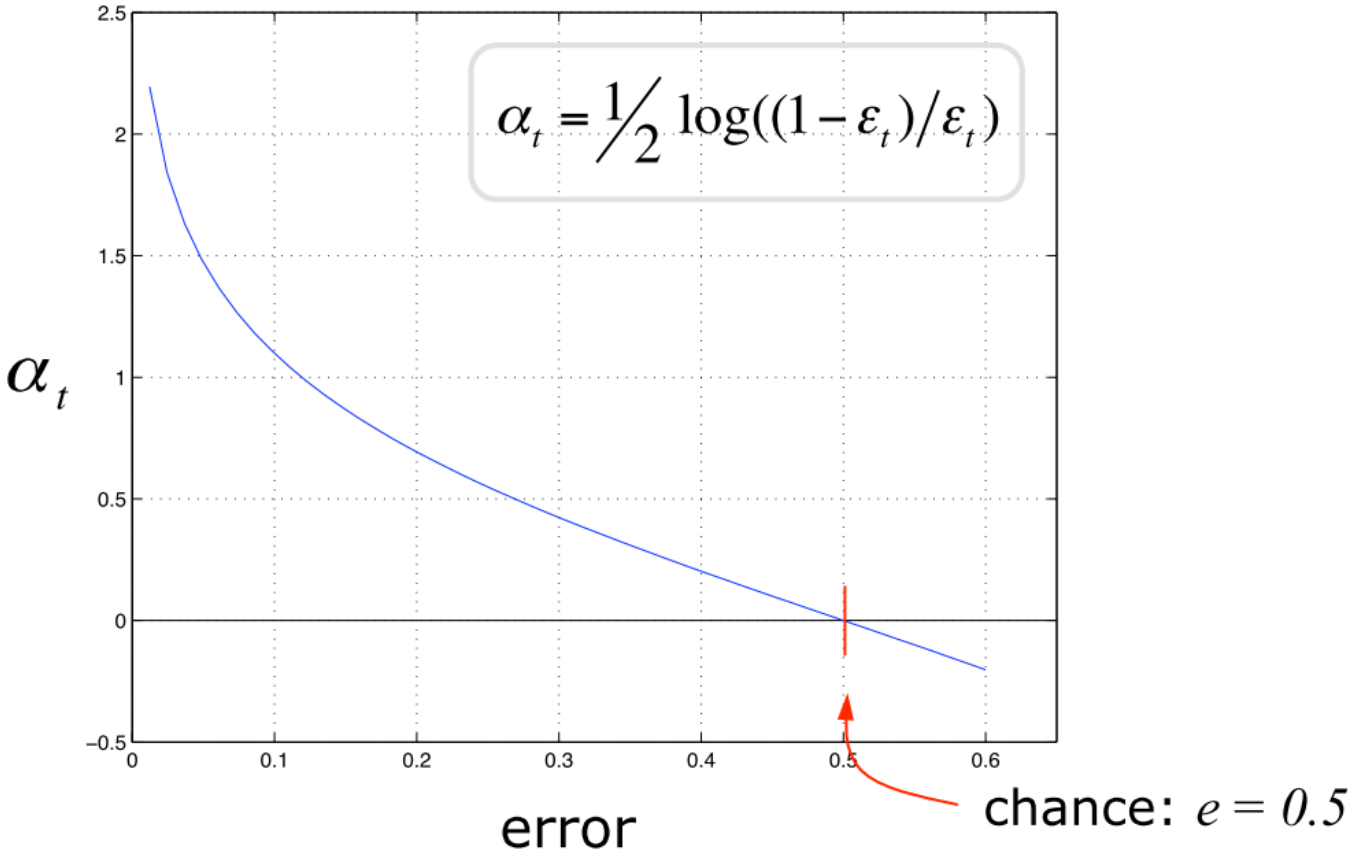
Given the **training data** $\{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n)\}$ $\mathbf{x} \in \mathcal{X}$ $y \in \mathcal{Y}$

1. Initialize weights $w_t(i) = 1/n$ → These are the data points weights, different from classifier voting weights α's
2. For $t = 1,\ldots,T$
   - Train a **weak classifier** $h_t(x)$ on weighted training data minimizing the error
     $$\varepsilon_t = \sum_{i=1}^{n} w_t(i)\, \mathrm{I}(y_i \neq h_t(x_i))$$
   - Compute voting weight of $h_t(x)$:
     $$\alpha_t = \tfrac{1}{2} \log((1-\varepsilon_t)/\varepsilon_t)$$
   - Recompute weights:
     $$w_{t+1}(i) = w_t(i) \exp\{-\alpha_t\, y_i\, h_t(x_i)\} / Z_t$$
3. Make predictions using the final **strong classifier**

Slides from Arras et. al.

Voting Weight

- Computing the **voting weight** α_t of a weak classifier
- α_t measures the **importance** assigned to $h_t(x_i)$



Slides from Arras et. al.

Weight Update

- Looking at the weight update step:

$$w_{t+1}(i) = w_t(i)\exp\{-\alpha_t y_i h_t(x_i)\} / Z_t$$

Z_t : **Normalizer** such that w_{t+1} is a prob. distribution

$$\exp\{-\alpha_t y_i h_t(x_i)\} = \begin{cases} <1, & y_i = h_t(x_i) \\ >1, & y_i \neq h_t(x_i) \end{cases}$$

➔ Weights of misclassified training samples are **increased**

➔ Weights of correctly classified samples are **decreased**

- Algorithm generates weak classifier by training the next learner on the **mistakes of the previous one**
- Now we understand the name: **AdaBoost** comes from **adaptive Boosting**

Slides from Arras et. al.

Strong Classifier

- **Training is completed…**

  The weak classifiers $h_{1\ldots T}(x)$ and their voting weight $\alpha_{1\ldots T}$ are now fix

- **The resulting strong classifier is**

$$H(x_i) = \operatorname{sgn}\left(\sum_{t=1}^{T} \alpha_t h_t(x_i)\right) \longrightarrow \text{Class Result } \{+1, -1\}$$

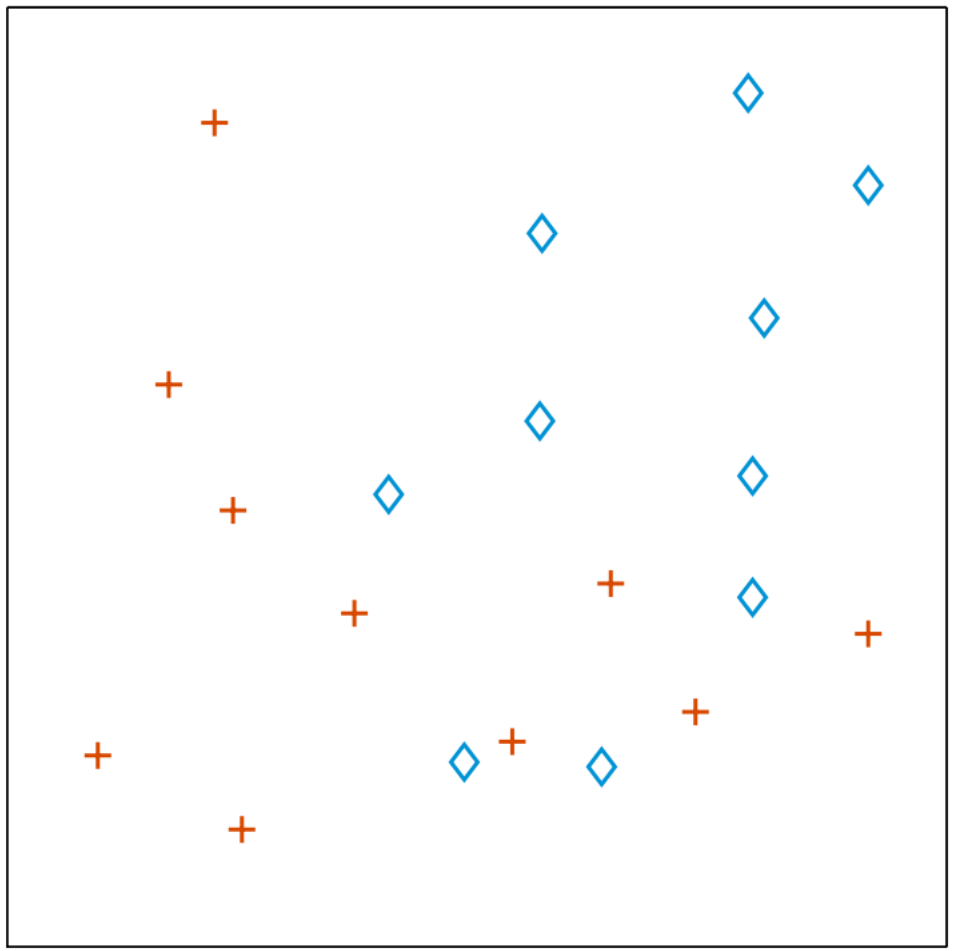
Put your data here

Weighted majority voting scheme

Slides from Arras et. al.

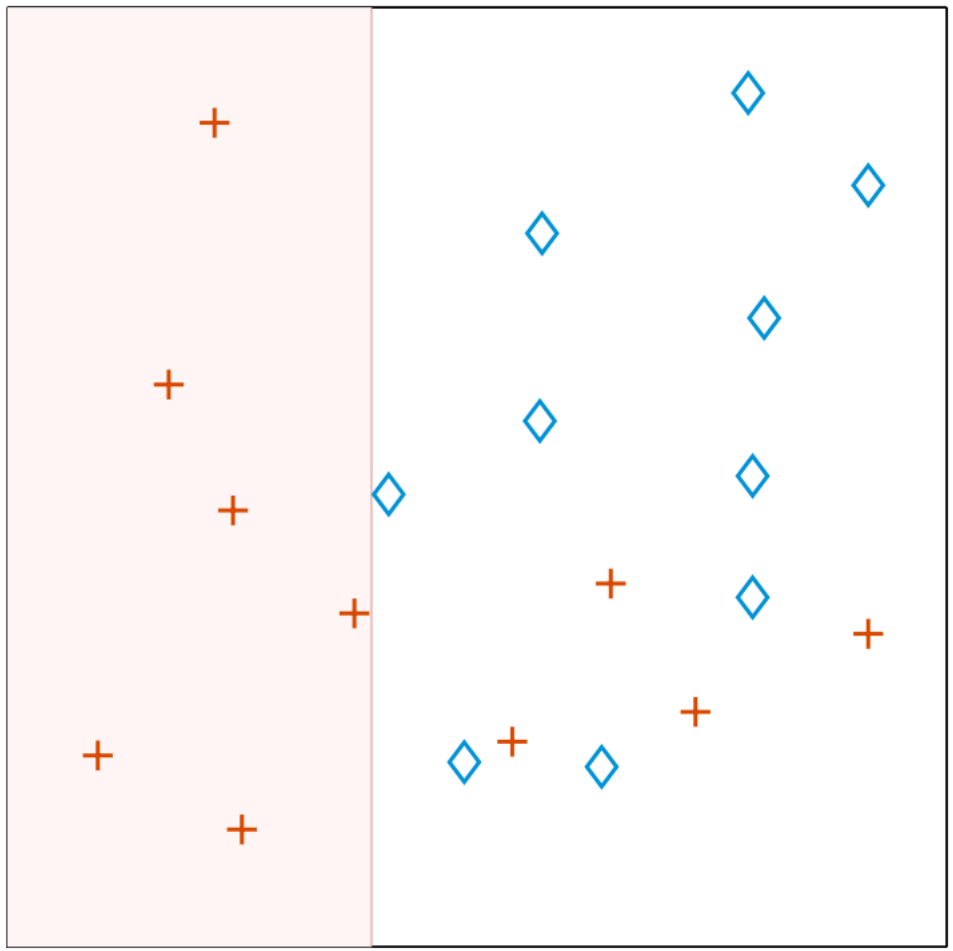
AdaBoost: Step-By-Step

- **Training data**



AdaBoost: Step-By-Step

- **Iteration 1, train weak classifier 1**



Threshold
$\theta^* = 0.37$

Dimension
$j^* = 1$

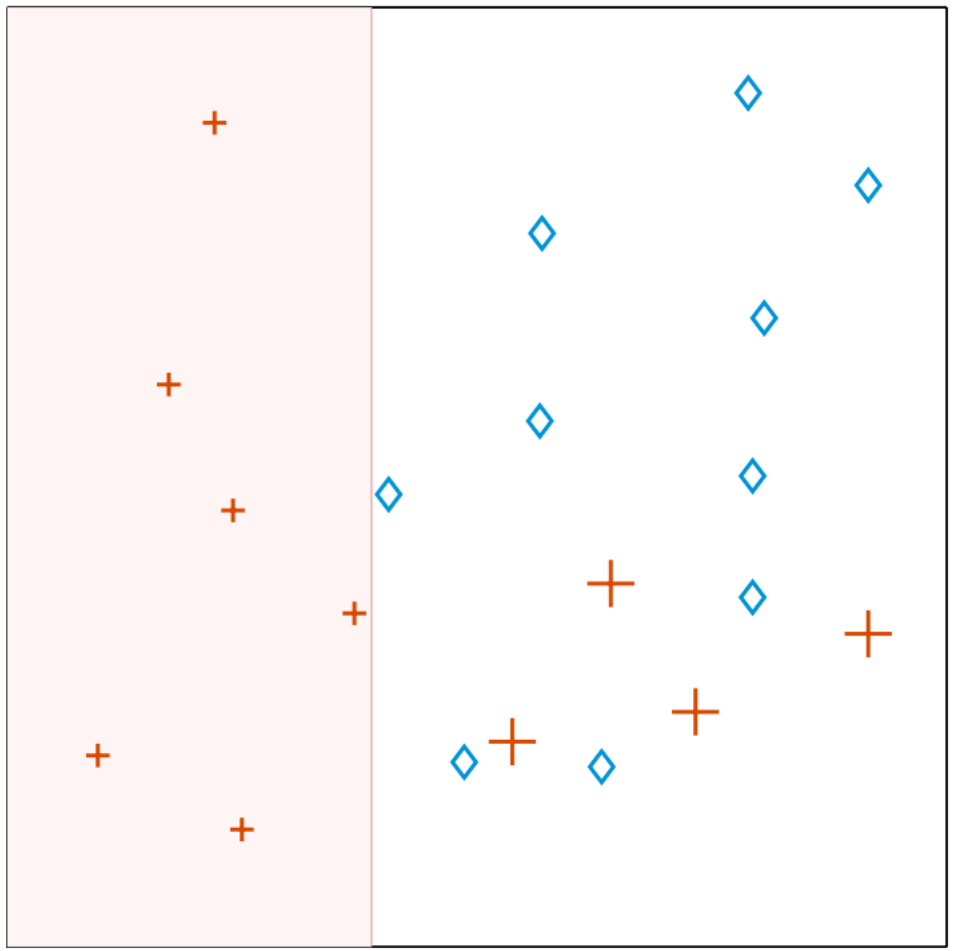
Weighted error
$e_t = 0.2$

Voting weight
$\alpha_t = 1.39$

Total error = 4

AdaBoost: Step-By-Step

- **Iteration 1, recompute weights**



Threshold
$\theta^* = 0.37$

Dimension
$j^* = 1$

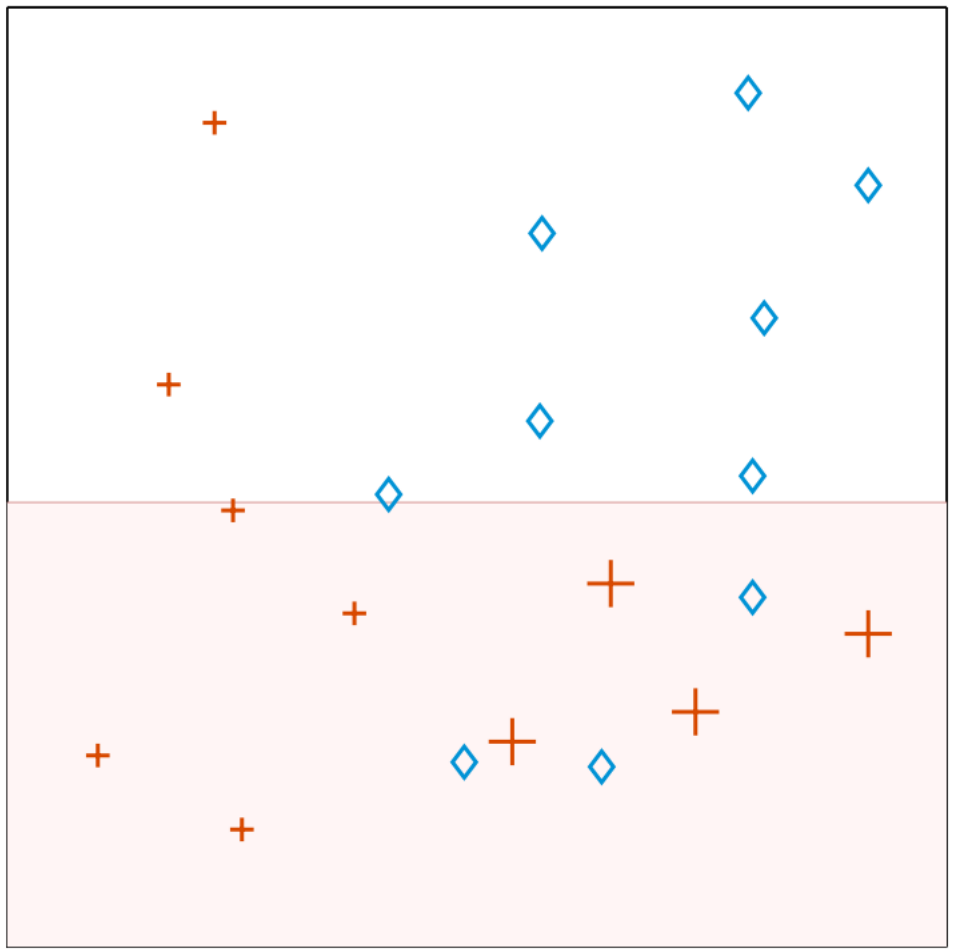
Weighted error
$e_t = 0.2$

Voting weight
$\alpha_t = 1.39$

Total error = 4

AdaBoost: Step-By-Step

- **Iteration 2, train weak classifier 2**



Threshold
$\theta^* = 0.47$

Dimension
$j^* = 2$

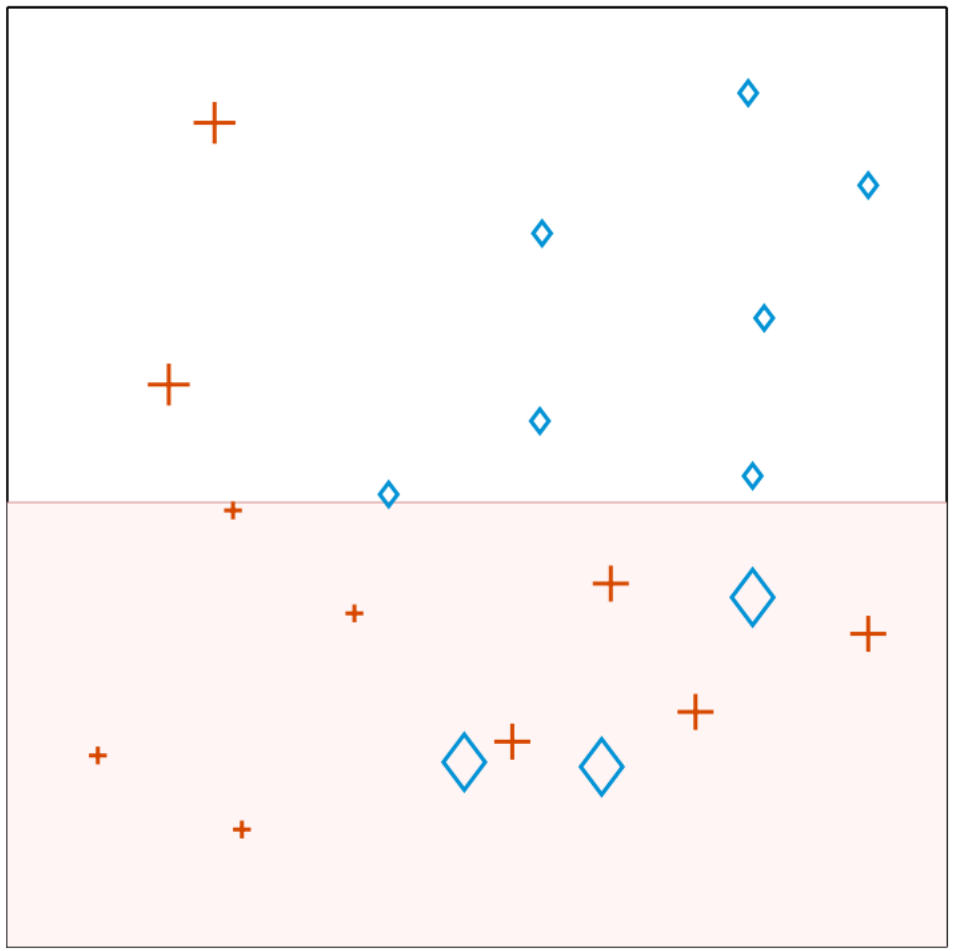
Weighted error
$e_t = 0.16$

Voting weight
$\alpha_t = 1.69$

Total error = 5

AdaBoost: Step-By-Step

- **Iteration 2, recompute weights**



Threshold
$\theta^* = 0.47$

Dimension
$j^* = 2$

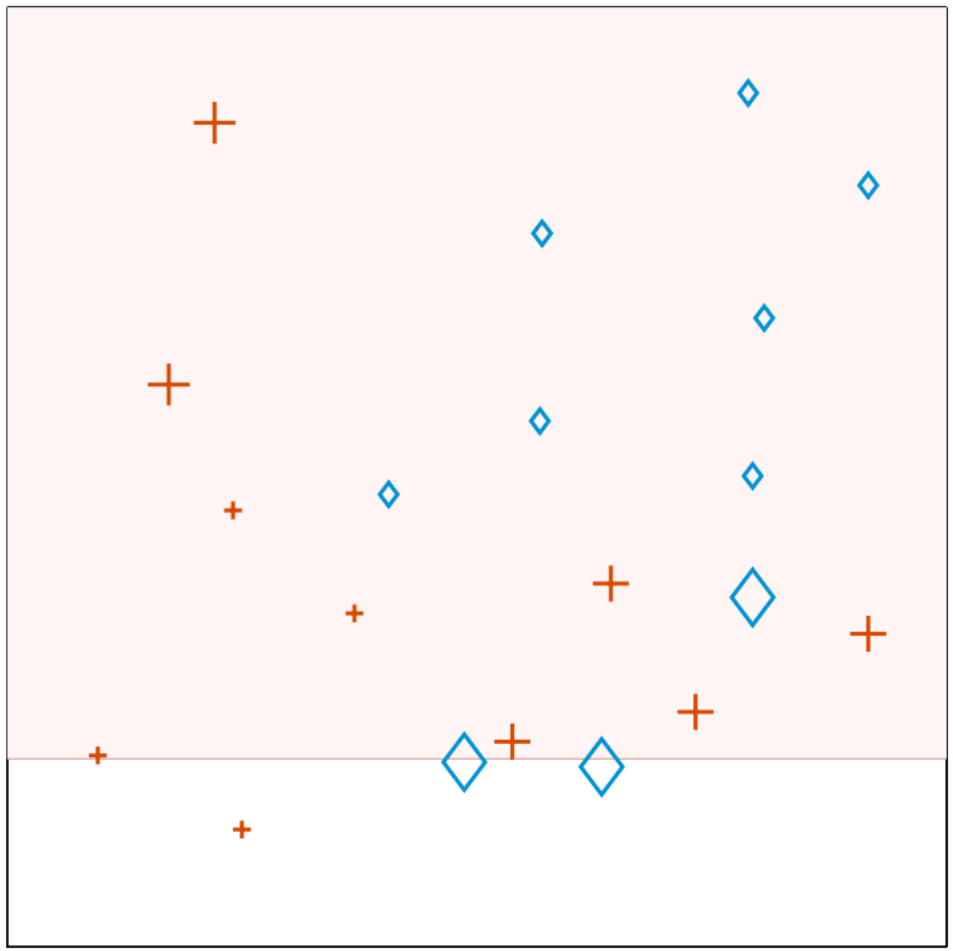
Weighted error
$e_t = 0.16$

Voting weight
$\alpha_t = 1.69$

Total error = 5

AdaBoost: Step-By-Step

- **Iteration 3, train weak classifier 3**



Threshold
$\theta^* = 0.14$

Dimension, sign
$j^* = 2$, neg

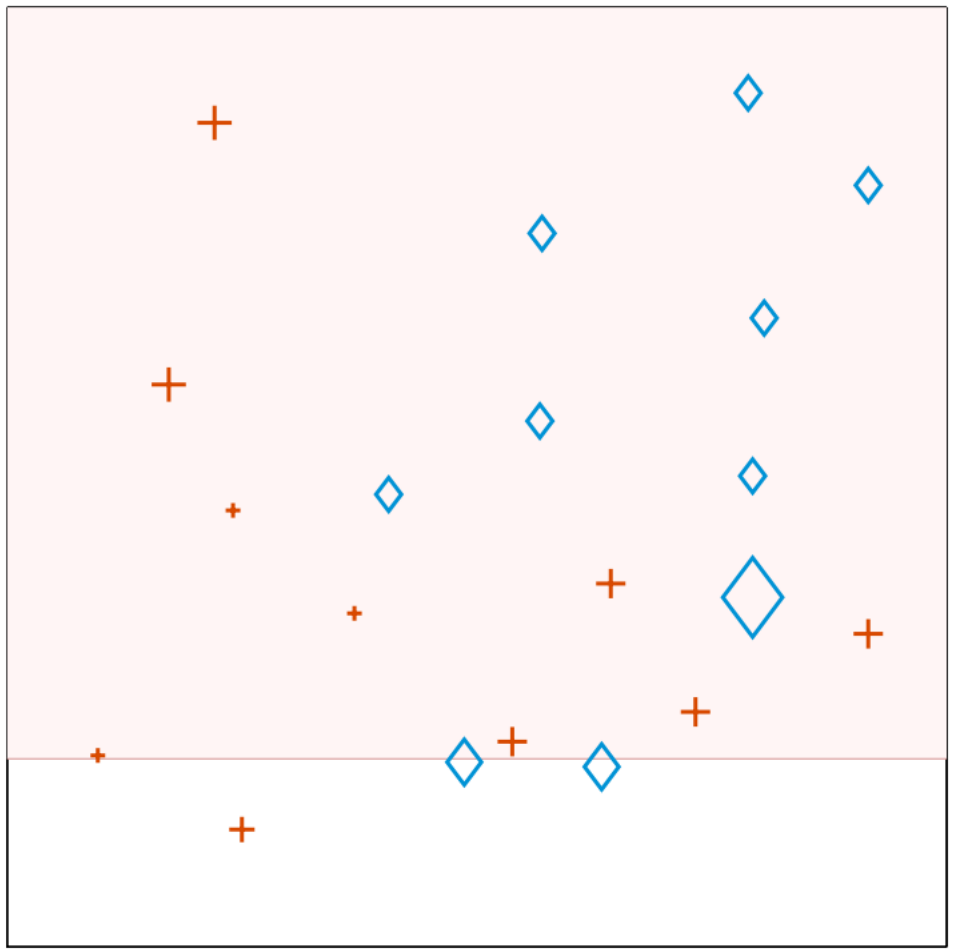
Weighted error
$e_t = 0.25$

Voting weight
$\alpha_t = 1.11$

Total error = 1

AdaBoost: Step-By-Step

- **Iteration 3, recompute weights**



Threshold
$\theta^* = 0.14$

Dimension, sign
$j^* = 2$, neg

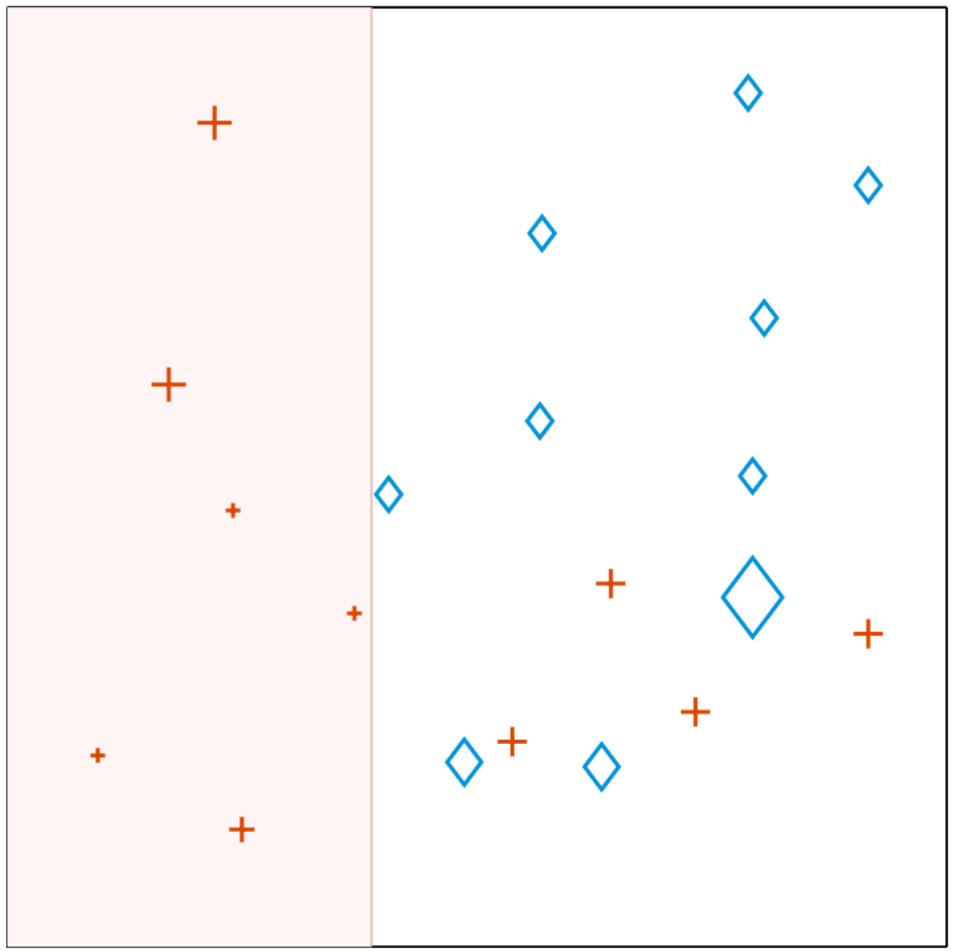
Weighted error
$e_t = 0.25$

Voting weight
$\alpha_t = 1.11$

Total error = 1

AdaBoost: Step-By-Step

- **Iteration 4, train weak classifier 4**



Threshold
$\theta^* = 0.37$

Dimension
$j^* = 1$

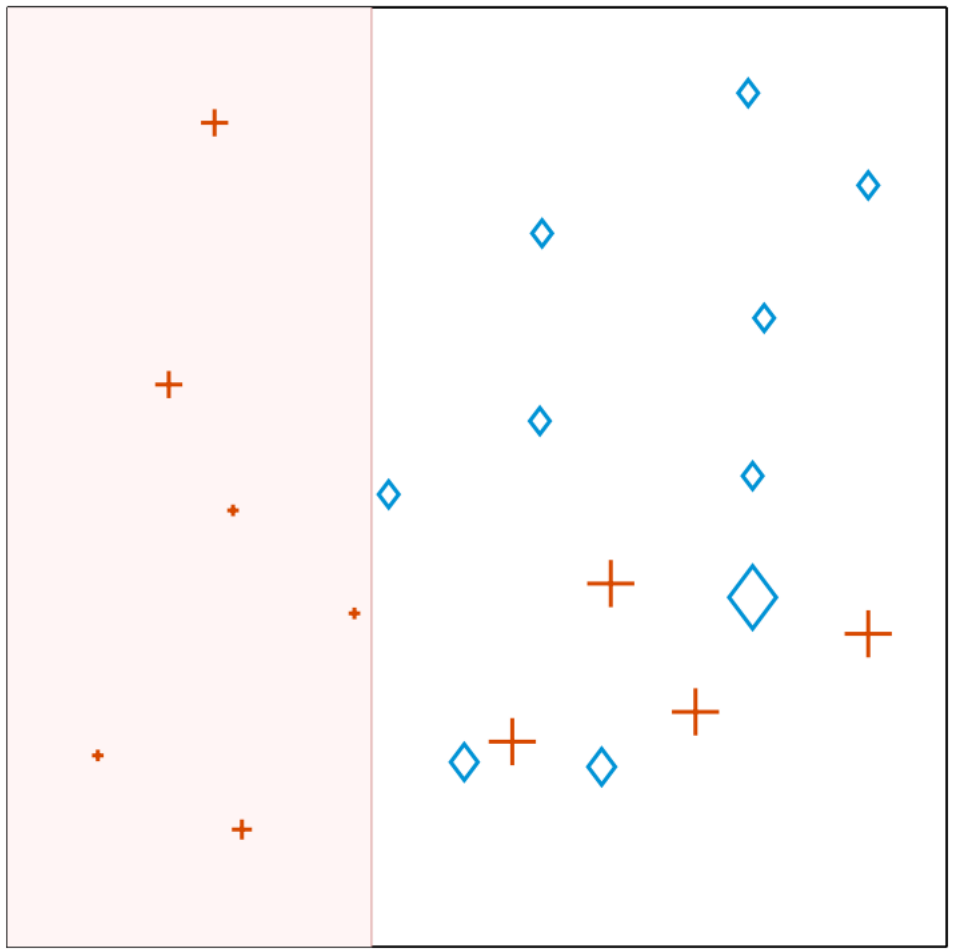
Weighted error
$e_t = 0.20$

Voting weight
$\alpha_t = 1.40$

Total error = 1

AdaBoost: Step-By-Step

- **Iteration 4, recompute weights**



Threshold
$\theta^* = 0.37$

Dimension
$j^* = 1$

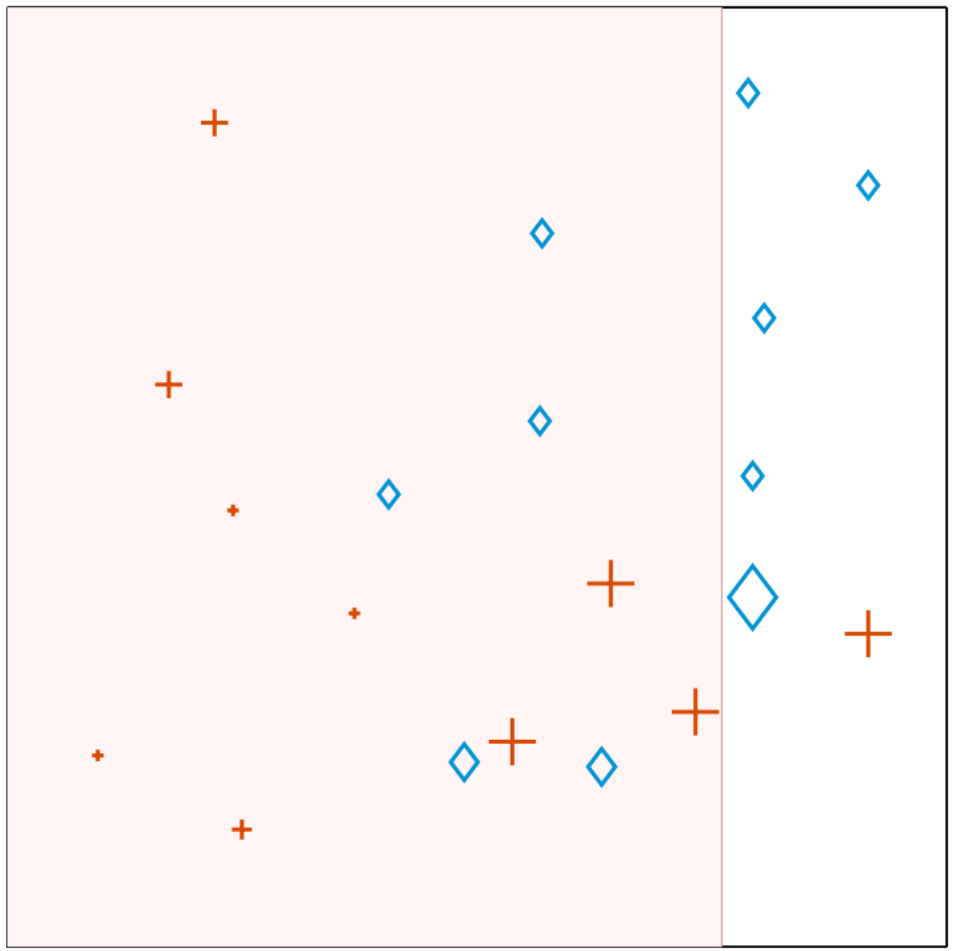
Weighted error
$e_t = 0.20$

Voting weight
$\alpha_t = 1.40$

Total error = 1

AdaBoost: Step-By-Step

- **Iteration 5, train weak classifier 5**



Threshold
$\theta^* = 0.81$

Dimension
$j^* = 1$

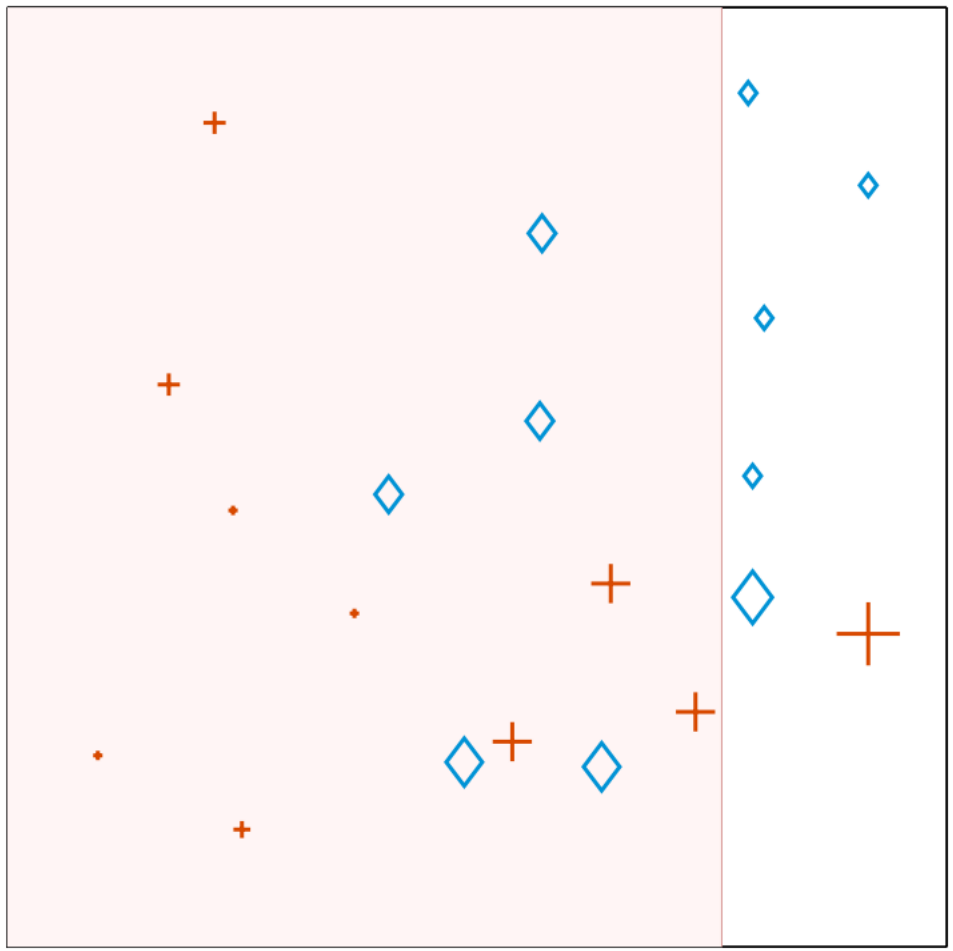
Weighted error
$e_t = 0.28$

Voting weight
$\alpha_t = 0.96$

Total error = 1

AdaBoost: Step-By-Step

- **Iteration 5, recompute weights**



Threshold
$\theta^* = 0.81$

Dimension
$j^* = 1$

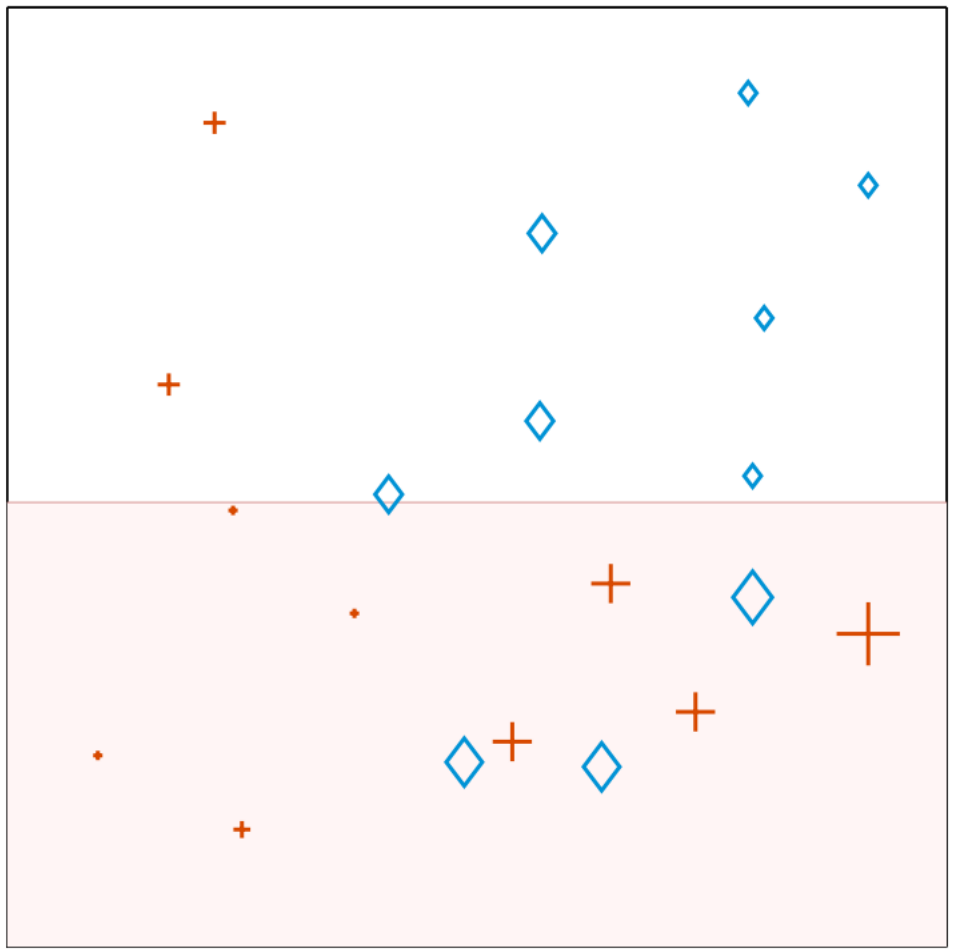
Weighted error
$e_t = 0.28$

Voting weight
$\alpha_t = 0.96$

Total error = 1

AdaBoost: Step-By-Step

- **Iteration 6, train weak classifier 6**



Threshold
$\theta^* = 0.47$

Dimension
$j^* = 2$

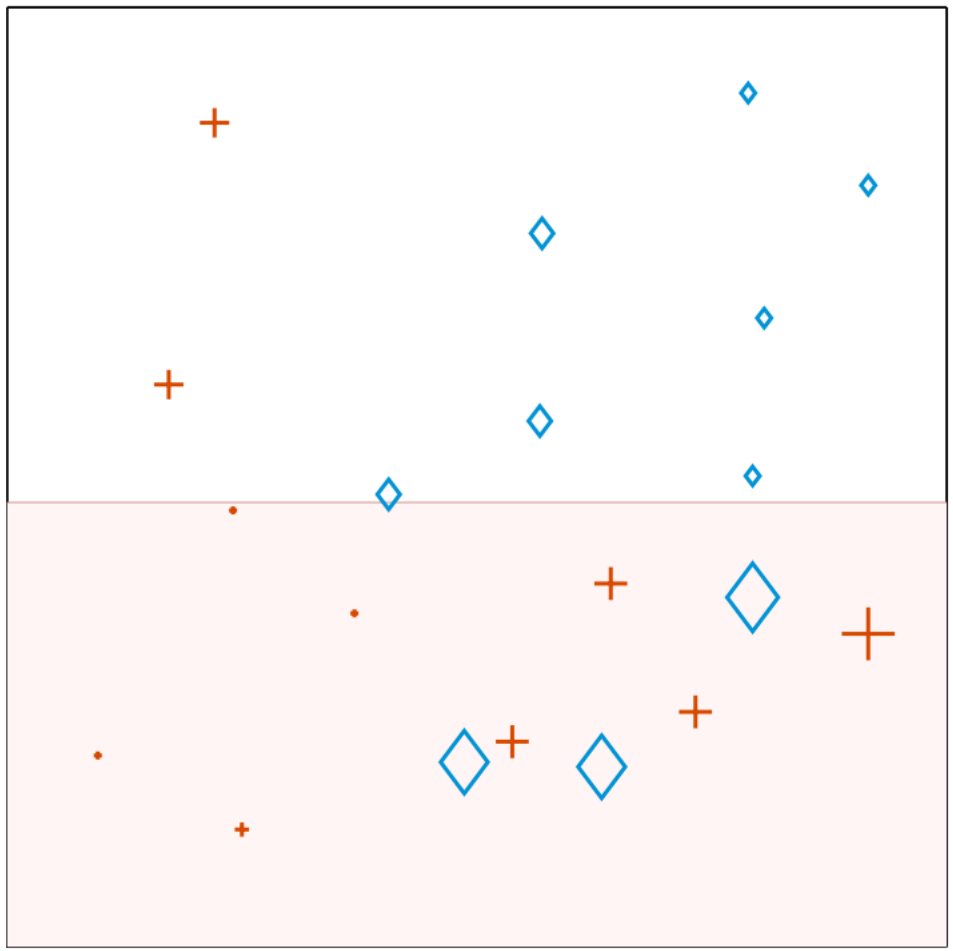
Weighted error
$e_t = 0.29$

Voting weight
$\alpha_t = 0.88$

Total error = 1

AdaBoost: Step-By-Step

- **Iteration 6, recompute weights**



Threshold
$\theta^* = 0.47$

Dimension
$j^* = 2$

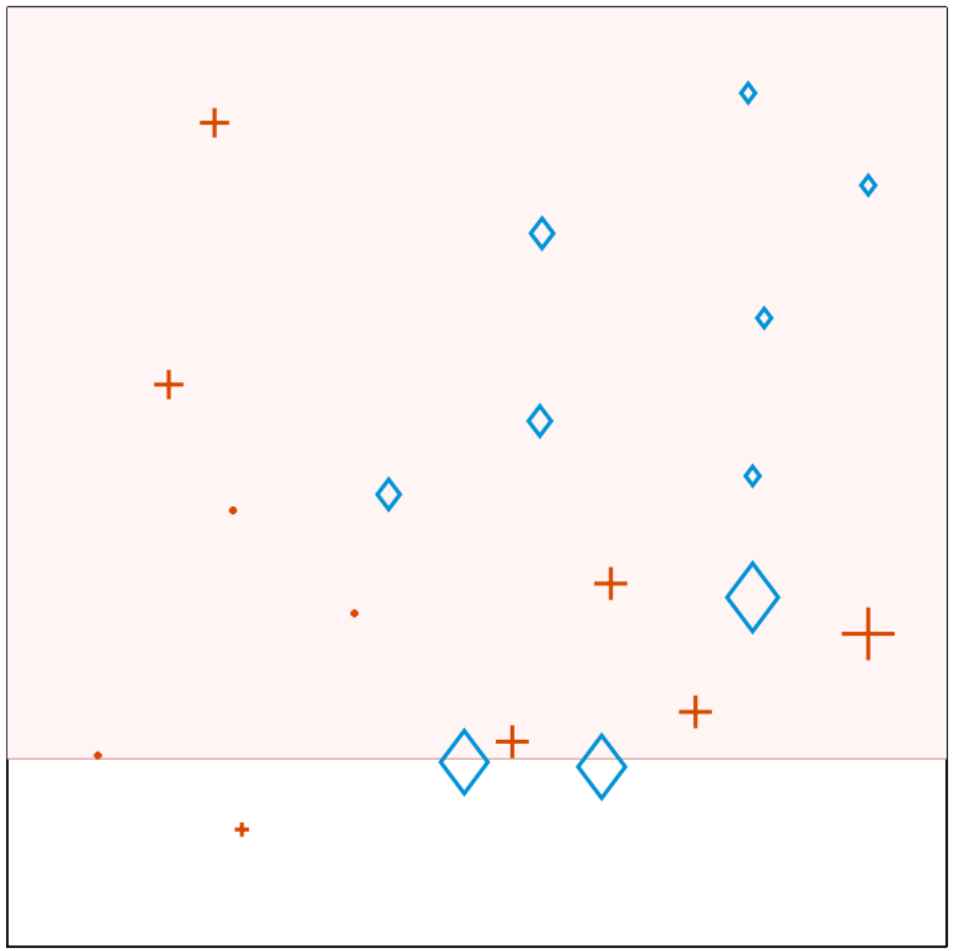
Weighted error
$e_t = 0.29$

Voting weight
$\alpha_t = 0.88$

Total error = 1

AdaBoost: Step-By-Step

- **Iteration 7, train weak classifier 7**



Threshold
$\theta^* = 0.14$

Dimension, sign
$j^* = 2$, neg

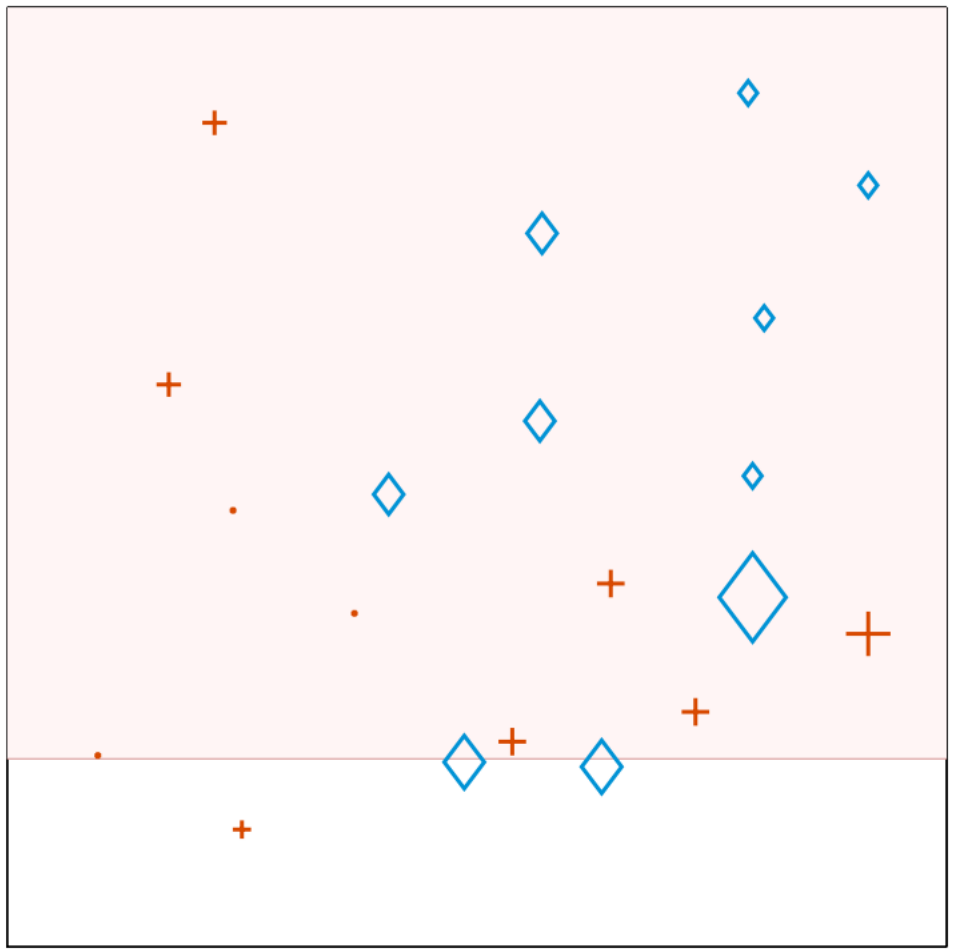
Weighted error
$e_t = 0.29$

Voting weight
$\alpha_t = 0.88$

Total error = 1

AdaBoost: Step-By-Step

- **Iteration 7, recompute weights**



Threshold
$\theta^* = 0.14$

Dimension, sign
$j^* = 2$, neg

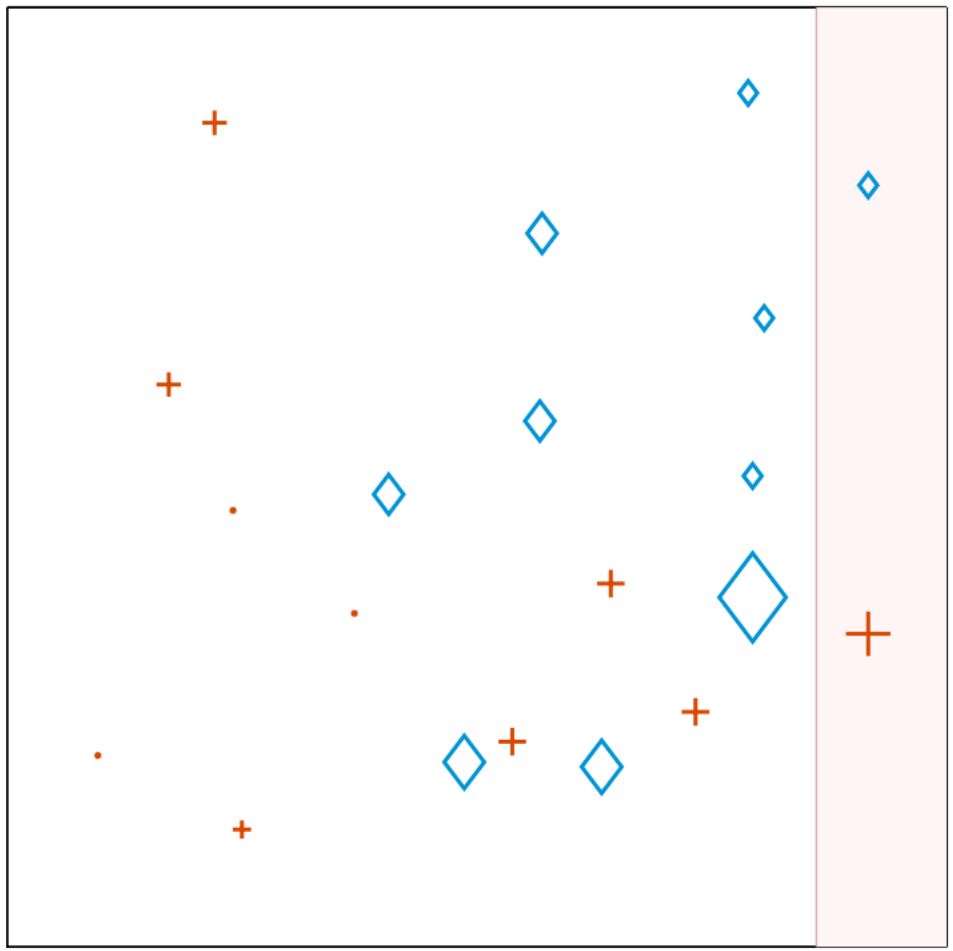
Weighted error
$e_t = 0.29$

Voting weight
$\alpha_t = 0.88$

Total error = 1

AdaBoost: Step-By-Step

- **Iteration 8, train weak classifier 8**



Threshold
$\theta^* = 0.93$

Dimension, sign
$j^* = 1$, neg

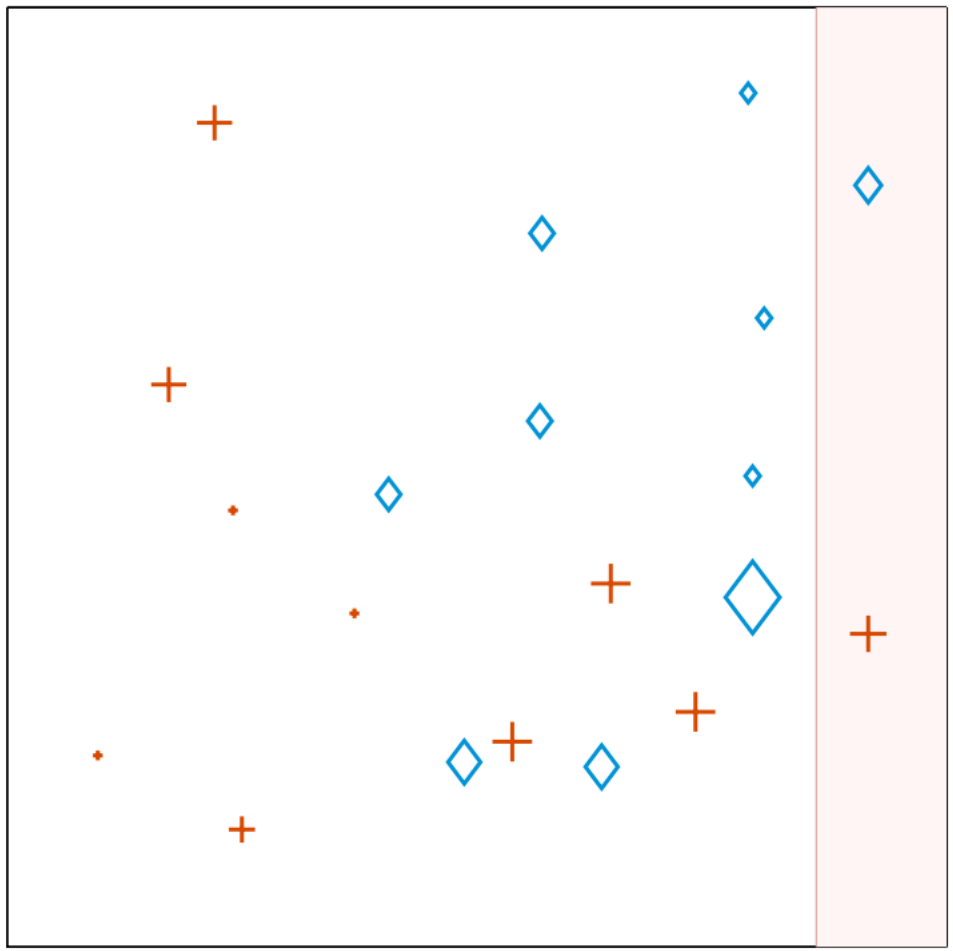
Weighted error
$e_t = 0.25$

Voting weight
$\alpha_t = 1.12$

Total error = 0

AdaBoost: Step-By-Step

- **Iteration 8, recompute weights**



Threshold
$\theta^* = 0.93$

Dimension, sign
$j^* = 1$, neg

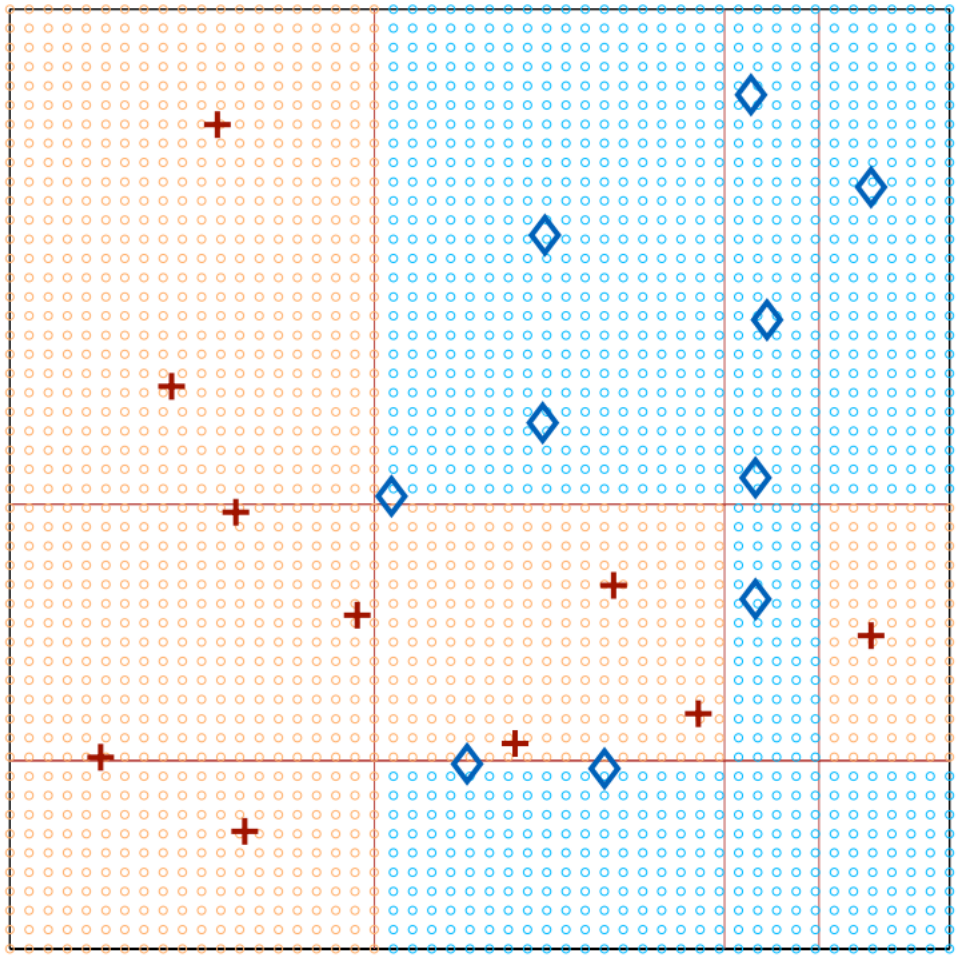
Weighted error
$e_t = 0.25$

Voting weight
$\alpha_t = 1.12$

Total error = 0

AdaBoost: Step-By-Step

- **Final Strong Classifier**



Total training error = 0

(Rare in practice)

AdaBoost Summary

- **Misclassified samples receive higher weight**. The higher the weight the "more attention" a training sample receives
- Algorithm generates weak classifier by training the next learner **on the mistakes** of the previous one
- AdaBoost minimizes **the upper bound of the training error** by properly choosing the optimal weak classifier and voting weight. AdaBoost can further be shown to maximize the margin
- **Large impact** on ML community and beyond

Slides from Arras et. al.

AdaBoost Example in Robotics

- People detection and tracking is a key component for many vision systems and for all robots in human environments

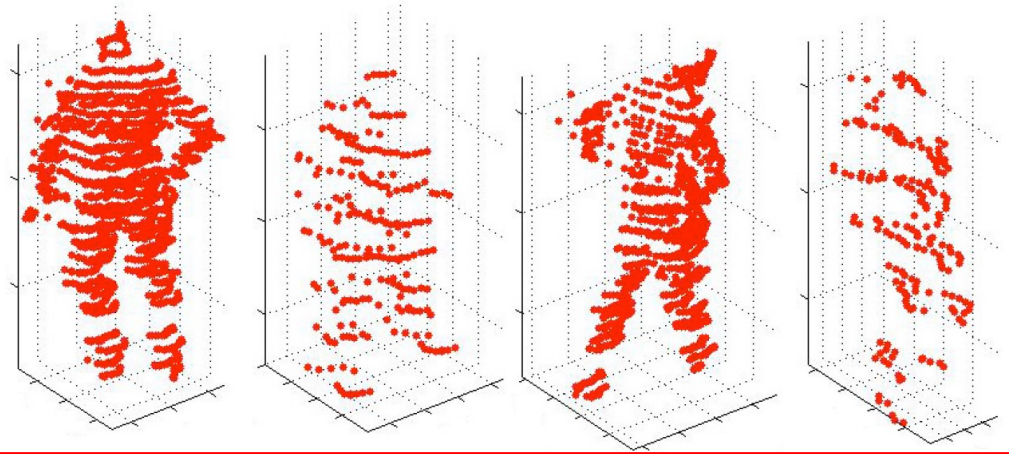
- **Where are the people?**
- **Why is it hard?**

  Range data contain little information on people

  Hard in cluttered environments

- **2D range data** from a SICK laser scanner

- Appearance of humans in **3D range data** (Velodyne scanner)



Slides from Arras et. al.

AdaBoost-based People Detection in 2D Range Scans

- Can we find **robust features** for people, legs and groups of people in 2D range data?
- What are the **best features** for people detection?
- Can we find people that **do not move**?

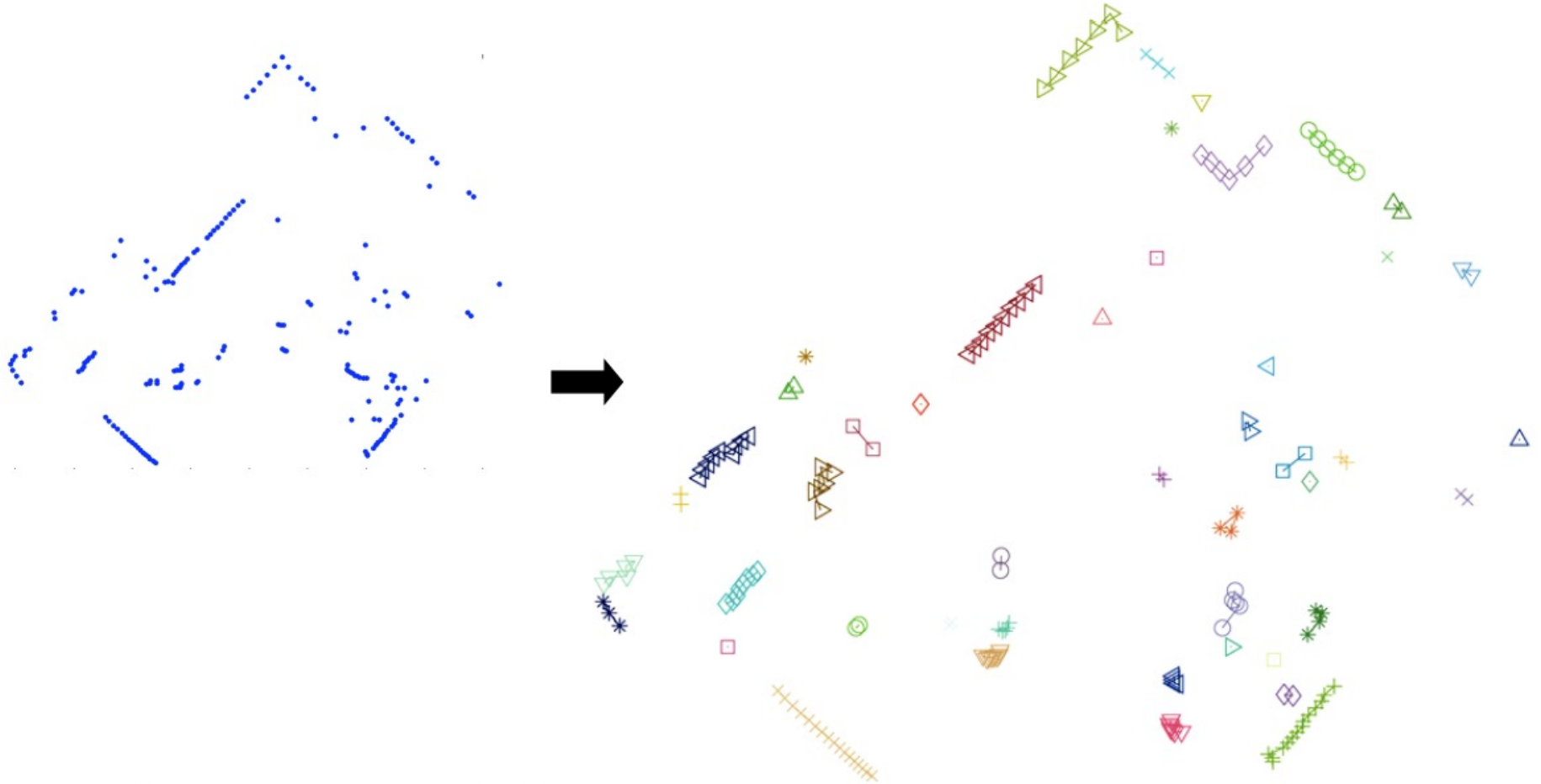
Approach:

- Classifying **groups of adjacent** beams (segments)
- Computing a set of scalar features on these groups
- **Boosting the features**

Slides from Arras et. al.

Segmentation

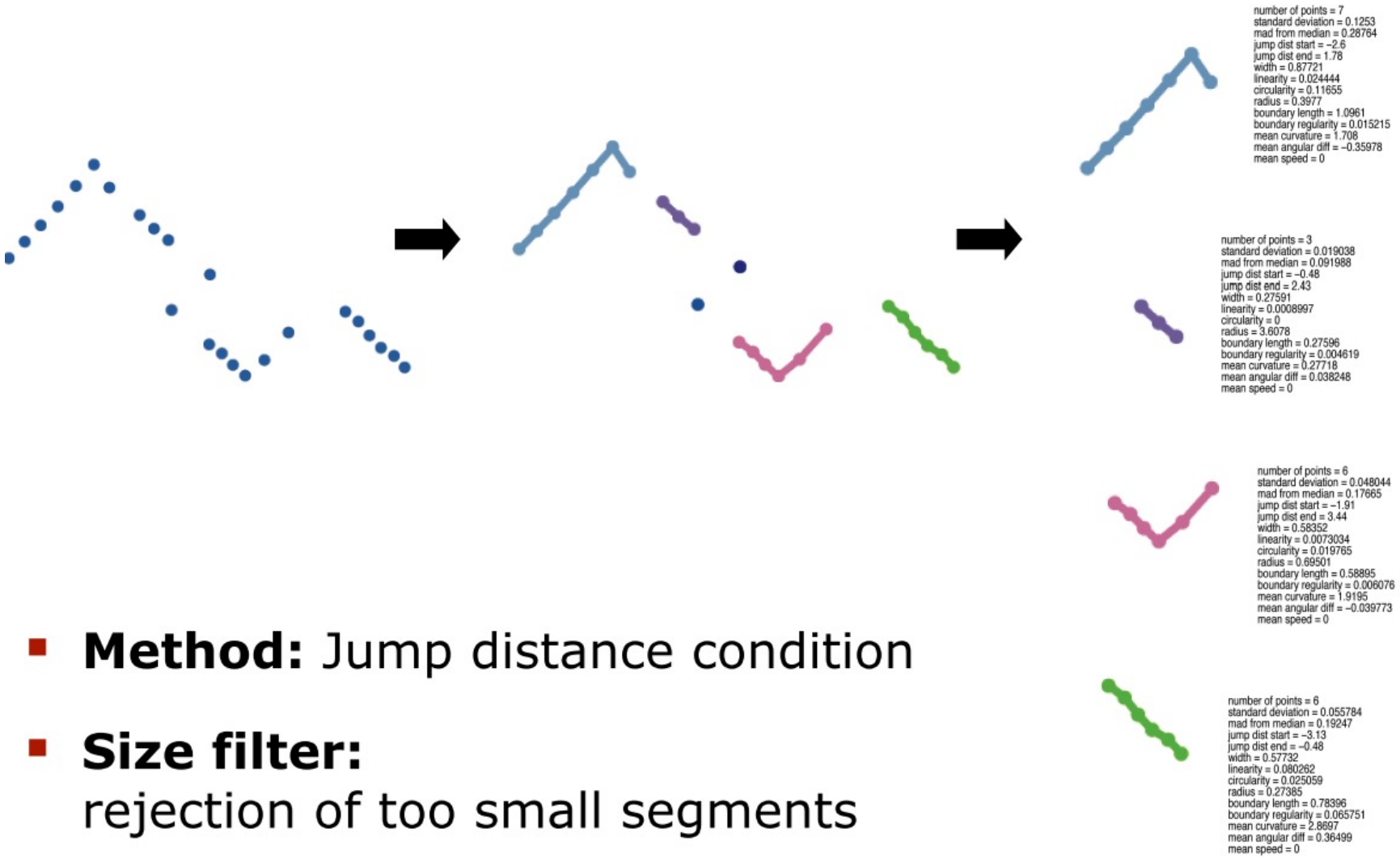
- Divide the scan into segments



Range image segmentation

Slides from Arras et. al.

Segmentation

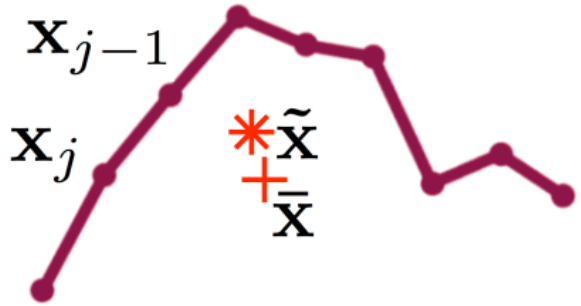


- **Method:** Jump distance condition
- **Size filter:**
  rejection of too small segments

Slides from Arras et. al.

Features per Segment

Segment S_i



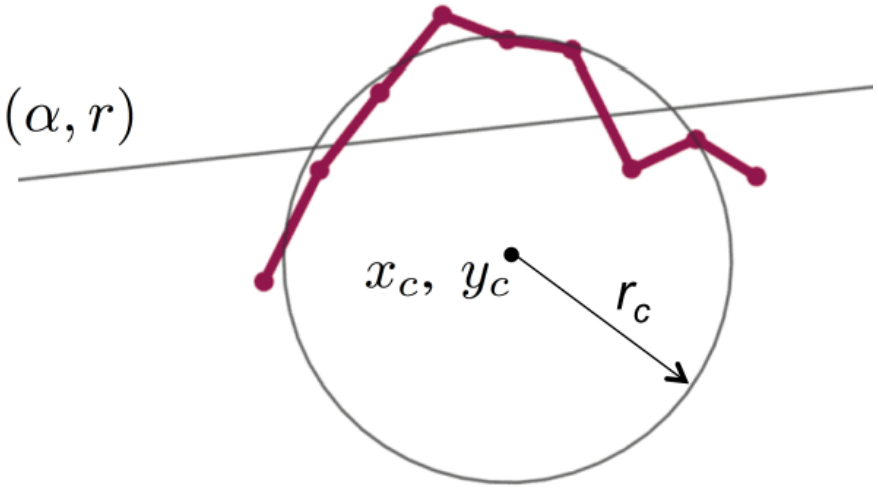
1. Number of points $n = |S_i|$
2. Standard Deviation $\sigma = \sqrt{\frac{1}{n-1}\sum ||\mathbf{x}_j - \bar{\mathbf{x}}||^2}$
3. Mean avg. deviation from median $\varsigma = \frac{1}{n}\sum ||\mathbf{x}_j - \tilde{\mathbf{x}}||$
4. Jump dist. to preceding segment $\delta_{j-1,j}$
5. Jump dist. to succeeding segment $\delta_{j,j+1}$
6. Width $w_i = ||\mathbf{x}_1 - \mathbf{x}_n||$

Feature engineering was once popular !

Slides from Arras et. al.

Features per Segment

Segment S_i



7. Linearity $s_l = \sum (x_j cos(\alpha) + y_j sin(\alpha) - r)^2$

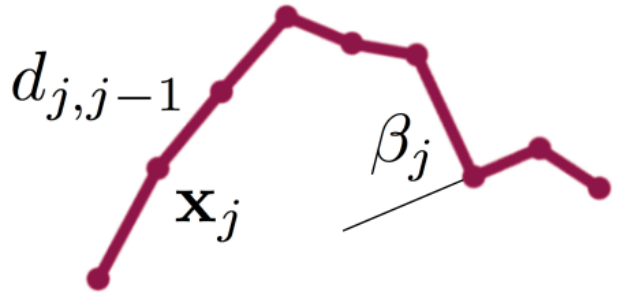
8. Circularity $s_c = \sum (r_c - \sqrt{(x_c - x_i)^2 + (y_c - y_i)^2})^2$

9. Radius r_c

Slides from Arras et. al.

Features per Segment

Segment S_i



10. Boundary Length $l = \sum_j d_{j,j-1}$

11. Boundary Regularity $\sigma_d = \sqrt{\frac{1}{n-1}\sum (d_{j,j-1} - \bar{d})^2}$

12. Mean curvature $\bar{k} = \frac{1}{n}\sum \hat{k}_j$

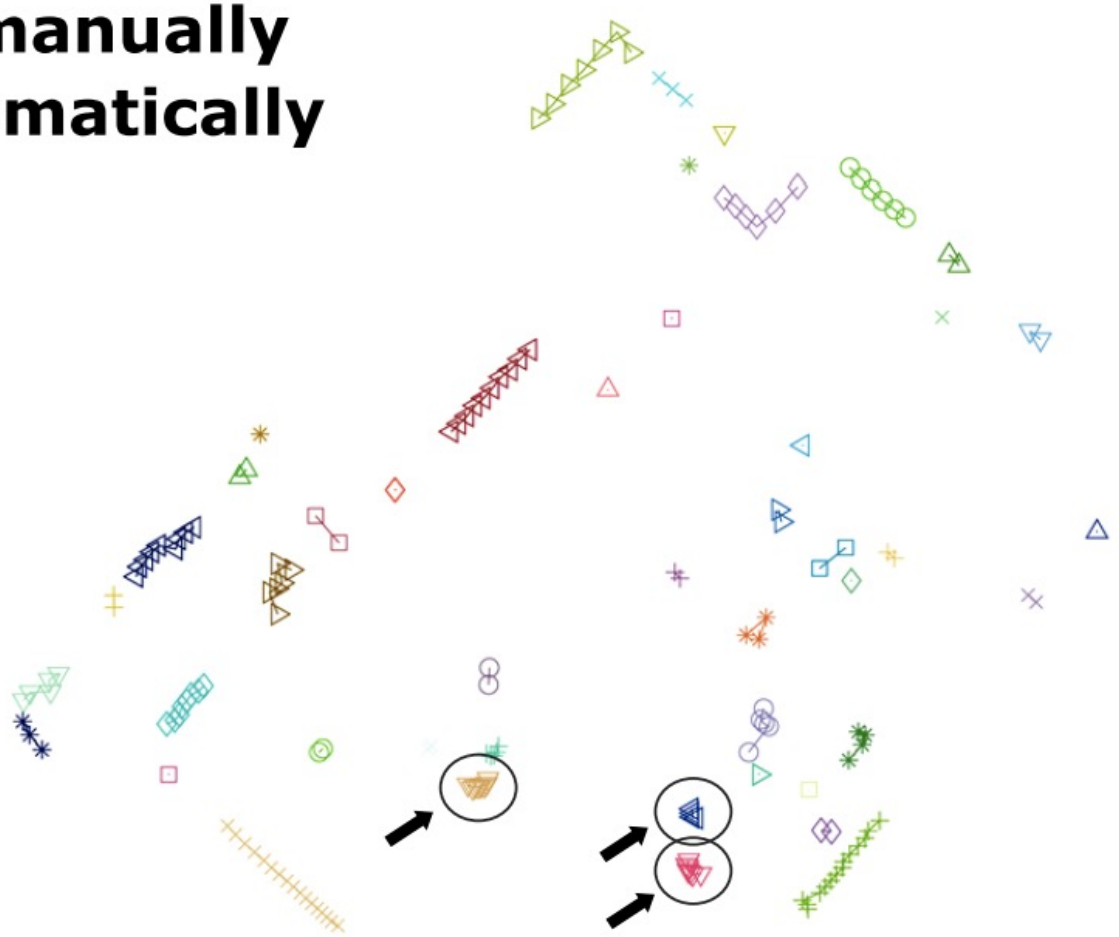
13. Mean angular difference $\bar{\beta} = \frac{1}{n}\sum \beta_j$

14. Mean speed $\bar{v} = \frac{1}{n}\sum \frac{\rho_j^{k+1} - \rho_j^k}{\Delta T}$

Slides from Arras et. al.

Data Labeling

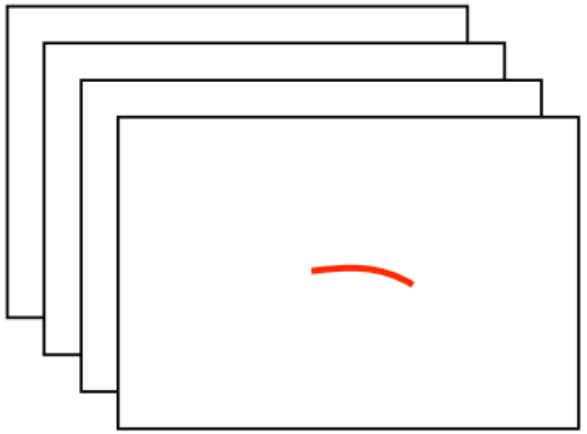
- **Mark segments** that correspond to people
- Either **manually** or **automatically**



Slides from Arras et. al.

Training

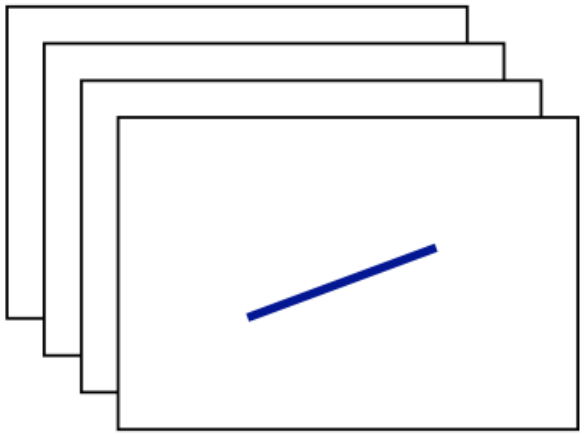
- Resulting **Training Set**



+1

Segments corresponding to people

(foreground segments)



-1

Segments corresponding to other objects

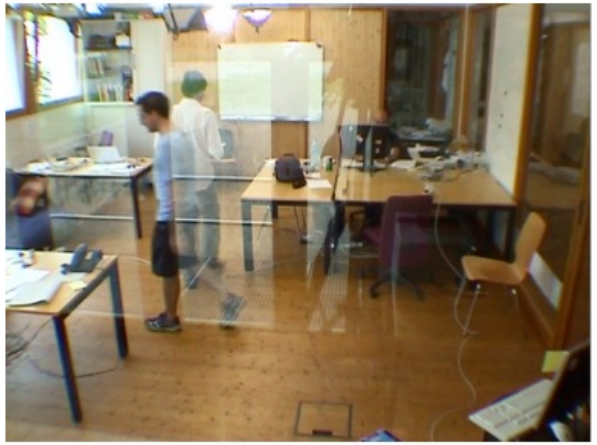
(background segments)

Slides from Arras et. al.

Evaluation

Env. 1: Corridor, no clutter

	Detected Label		
True Label	Person	No Person	Total
Person	**239** (99.58%)	**1** (0.42%)	**240**
No Person	**27** (1.03%)	**2589** (98.97%)	**2616**



Env. 2: Office, very cluttered

	Detected Label		
True Label	Person	No Person	Total
Person	**497** (97.45%)	**13** (2.55%)	**510**
No Person	**171** (2.73%)	**6073** (96.26%)	**6244**

Slides from Arras et. al.

Evaluation

Env. 1 & 2: Corridor and Office

	Detected Label		
True Label	Person	No Person	Total
Person	**722** (96.27%)	**28** (3.73%)	**750**
No Person	**225** (2.54%)	**8649** (99.88%)	**8860**

Env. 1→2: Cross-evaluation
Trained in corridor, applied in office

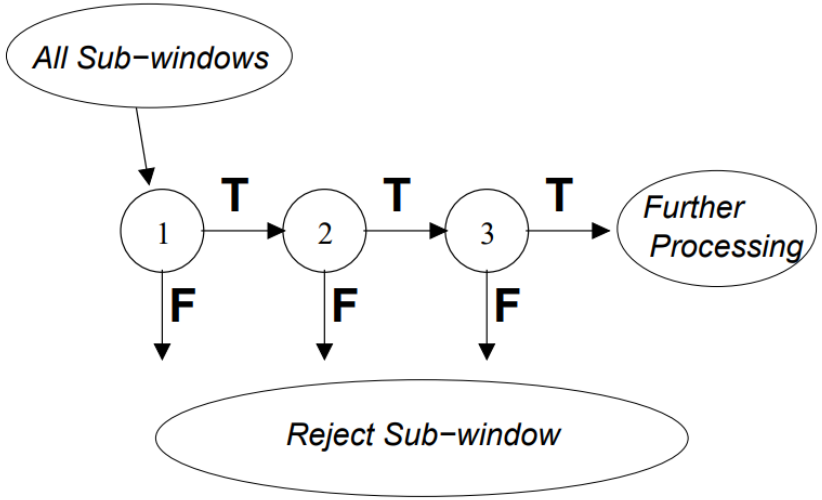
	Detected Label		
True Label	Person	No Person	Total
Person	**217** (90.42%)	**23** (9.58%)	**240**
No Person	**112** (4.28%)	**2504** (95.72%)	**2616**

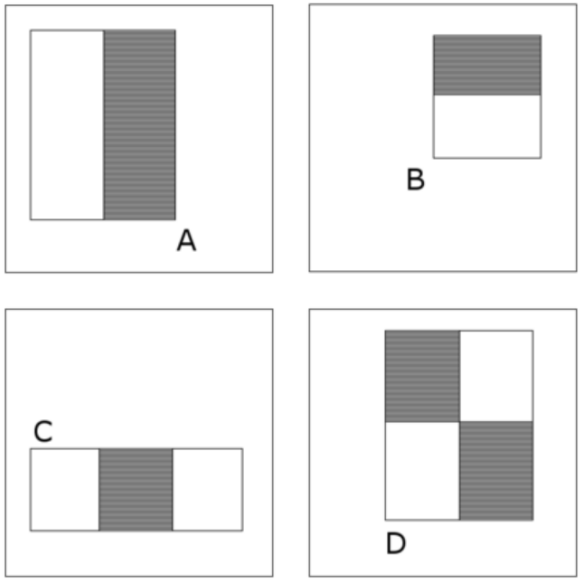
Slides from Arras et. al.

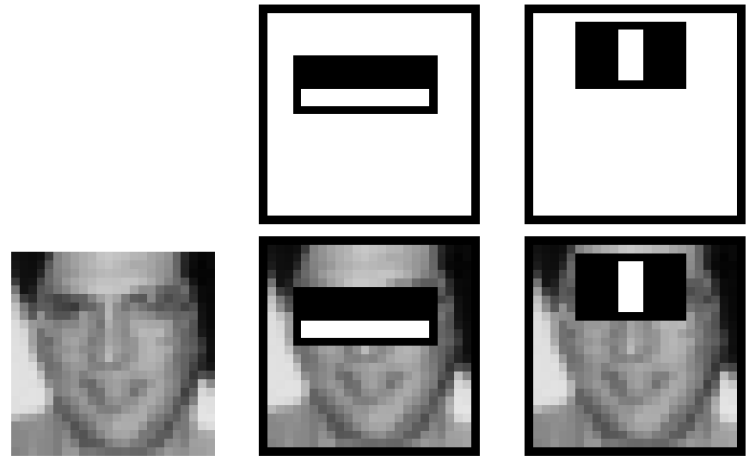
AdaBoost Example: Viola-Jones Object Detector

- **Haar Feature** Selection
- Creating an Integral Image
- AdaBoost Training

- Cascading Classifiers



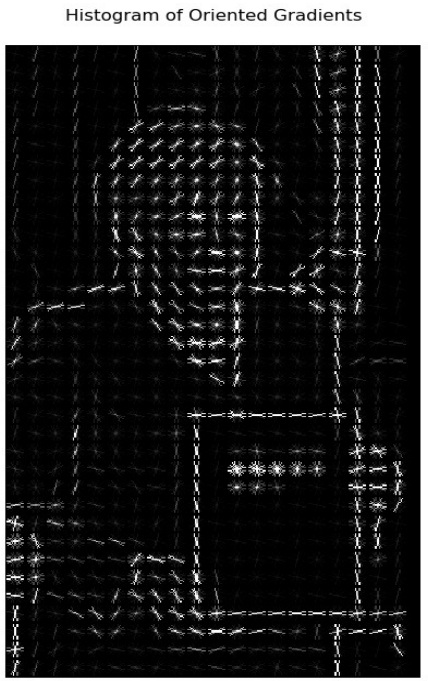




Viola, Paul, and Michael Jones. "Rapid object detection using a boosted cascade of simple features." *CVPR (1)* 1, no. 511-518 (2001): 3.

Histogram of Oriented Gradients (HOG)

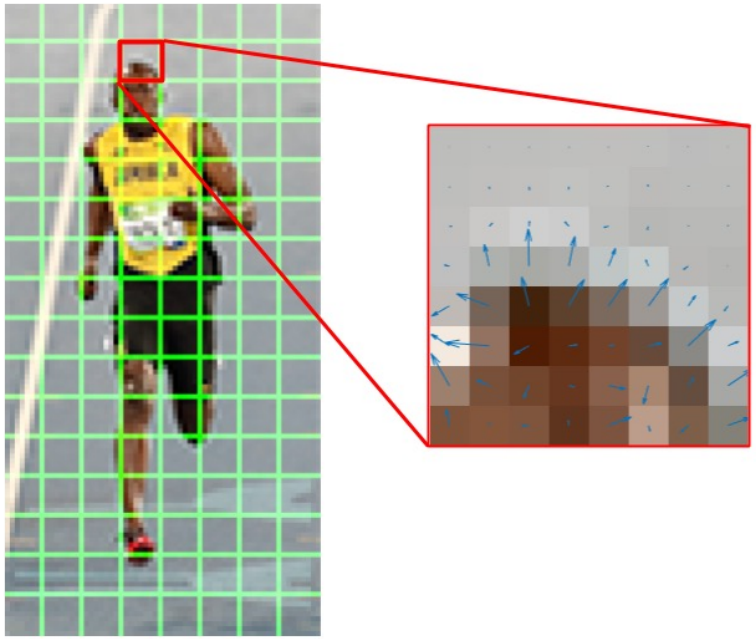
- Proposed by Dalal & Triggs in CVPR 2005 for detecting human in 2D images
- One of the most widely used feature descriptor for detecting human in 2D images



https://de.wikipedia.org/wiki/Histogram_of_oriented_gradients

Histogram of Oriented Gradients (HOG) - Algorithm

- Slide a detection window across different locations of an image.
- Divide detection window into small non-overlapping regions called cells
- Compute intensity gradient orientation histograms in each cell
- Human shapes can be described by local gradient orientations; local histograms are insensitive to small variations in translations and pose.



2	3	4	4	3	4	2	2
5	11	17	13	7	9	3	4
11	21	23	27	22	17	4	6
23	99	165	135	85	32	26	2
91	155	133	136	144	152	57	28
98	196	76	38	26	60	170	51
165	60	60	27	77	85	43	136
71	13	34	23	108	27	48	110

Gradient Magnitude

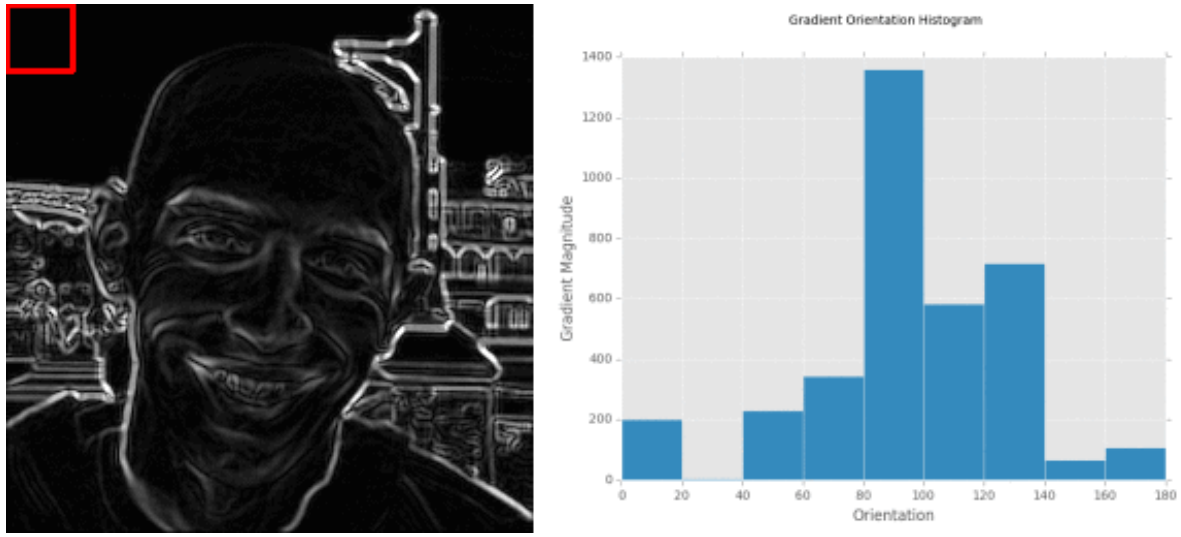
80	36	5	10	0	64	90	73
37	9	9	179	78	27	169	166
87	136	173	39	102	163	152	176
76	13	1	168	159	22	125	143
120	70	14	150	145	144	145	143
58	86	119	98	100	101	133	113
30	65	157	75	78	165	145	124
11	170	91	4	110	17	133	110

Gradient Direction

https://www.learnopencv.com/histogram-of-oriented-gradients/

Histogram of Oriented Gradients (HOG) - Algorithm

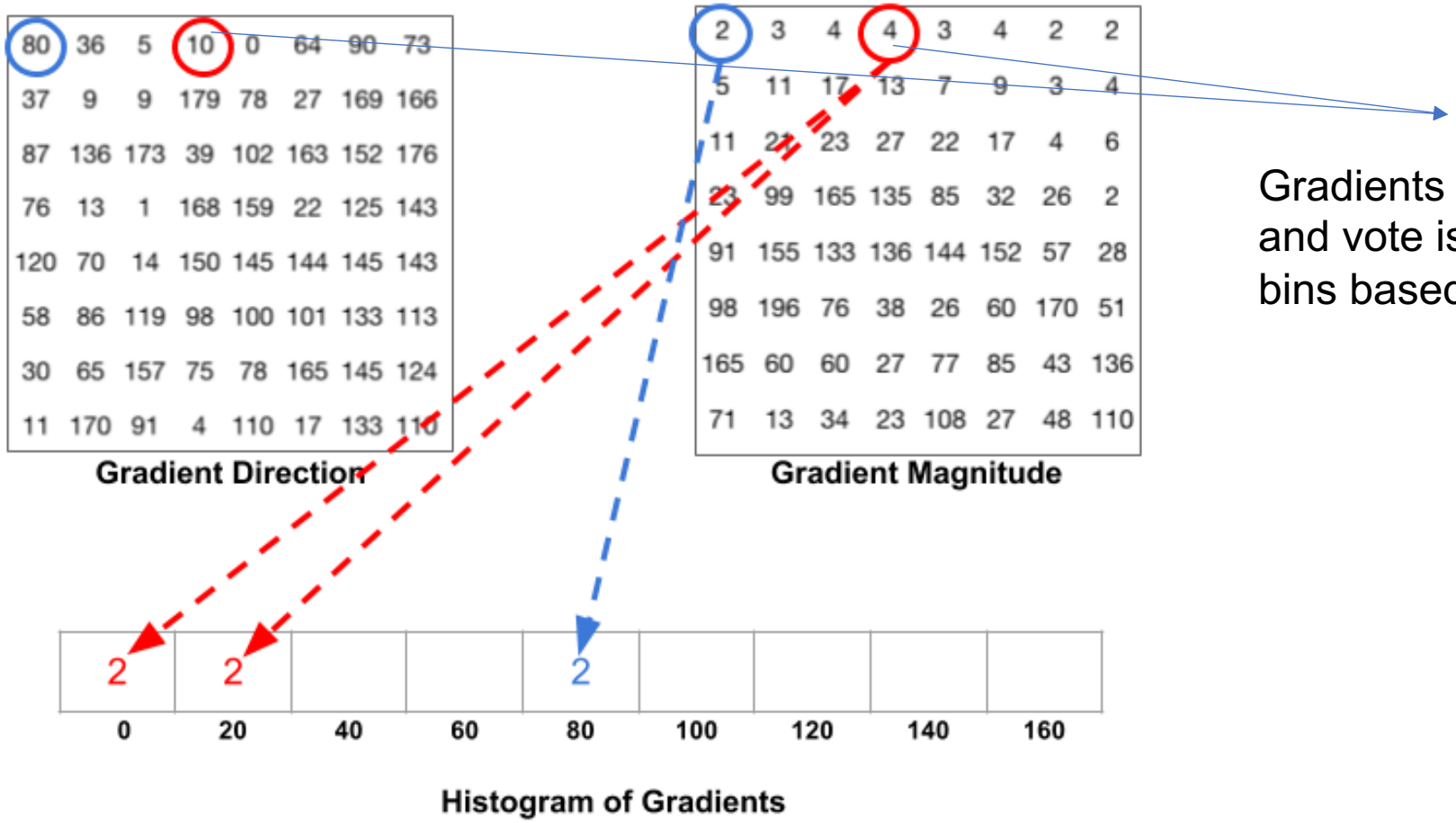
- Normalize local gradient orientation histograms over larger local regions called blocks
  - For handling variations in illuminations and image contrast
  - Blocks are made up of cells and overlapping blocks are typically used in HOG



- The normalized histograms from the individual blocks are concatenated together to form the final HoG descriptor (feature vector)

https://gurus.pyimagesearch.com/lesson-sample-histogram-of-oriented-gradients-and-car-logo-recognition/

Histogram of Oriented Gradients – Local gradient histogram

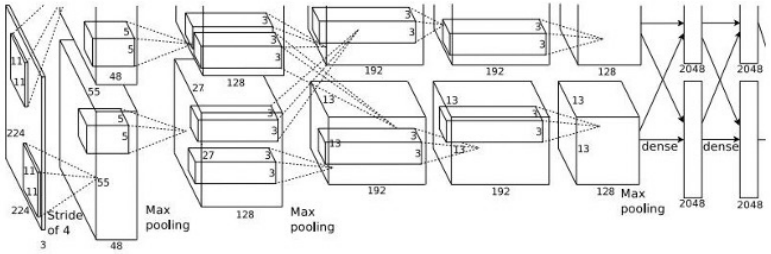


Gradients magnitude used as weight and vote is split between two closest bins based on distance to bin center

Object Detection using Deep Learning

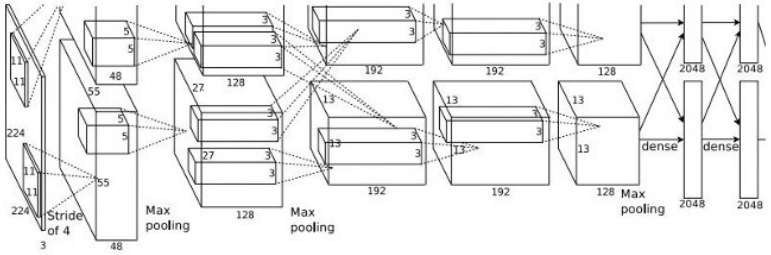
Object Detection by CNN – Challenges

Each image needs a different number of outputs



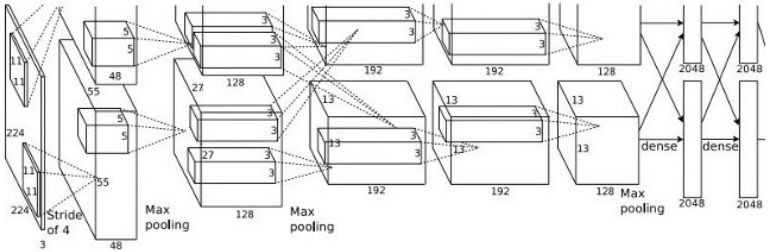
CAT: (x, y, w, h)

4 numbers



DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)

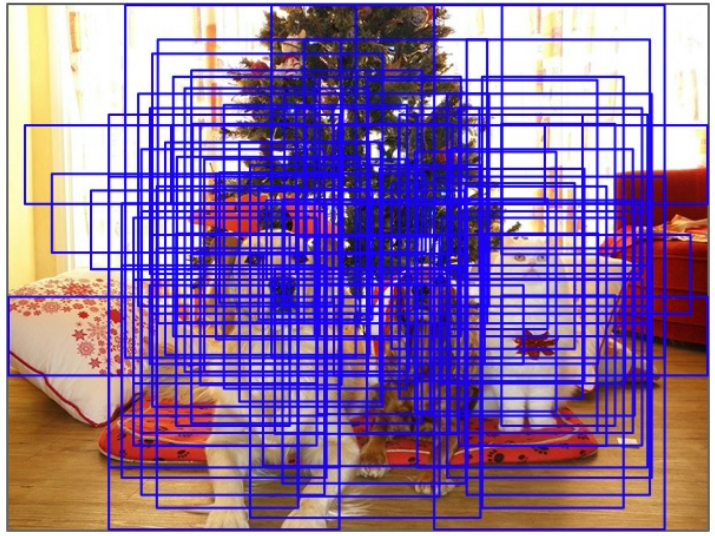
12 numbers



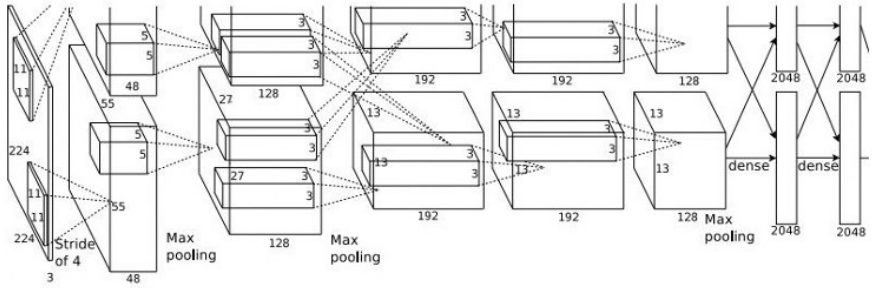
DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
....

Many numbers!

Object Detection by CNN – Challenges



Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

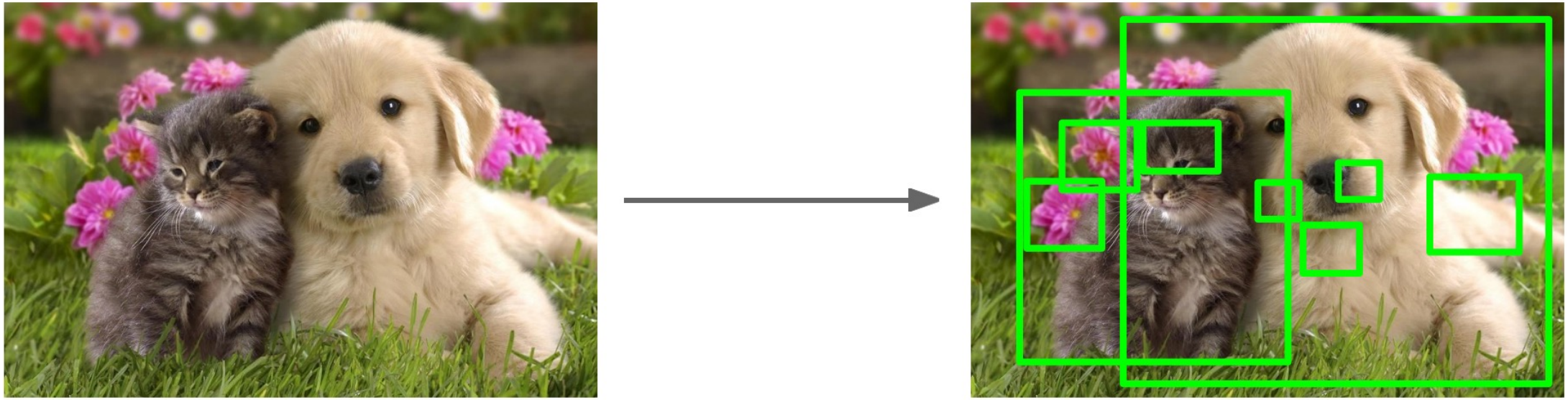


Dog? NO
Cat? YES
Background? NO

Problem: Need to apply CNN to huge number of locations, scales, and aspect ratios, very computationally expensive!

Region Proposals Can Help: Selective Search

- Find “blobby” image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 2000 region proposals in a few seconds on CPU



Alexe et al, “Measuring the objectness of image windows”, TPAMI 2012
Uijlings et al, “Selective Search for Object Recognition”, IJCV 2013
Cheng et al, “BING: Binarized normed gradients for objectness estimation at 300fps”, CVPR 2014
Zitnick and Dollar, “Edge boxes: Locating object proposals from edges”, ECCV 2014

Selective Search

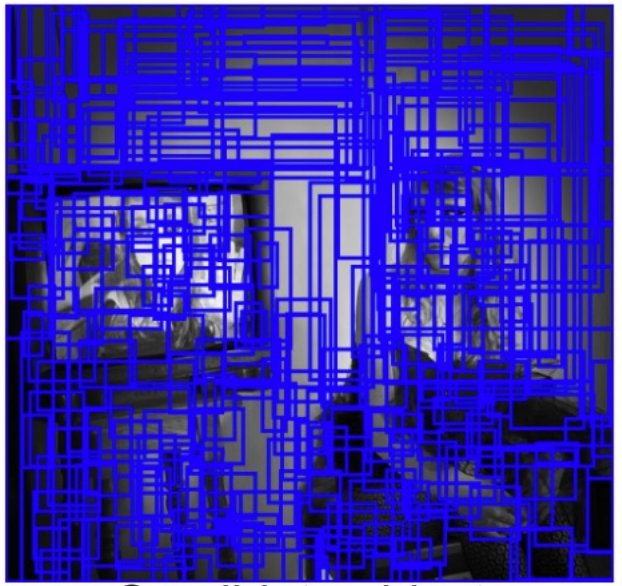
Step 1: Generate initial sub-segmentation

Goal: Generate many regions, each of which belongs to at most one object.

Using the method described by Felzenszwalb et al.

Input Image

Segmentation

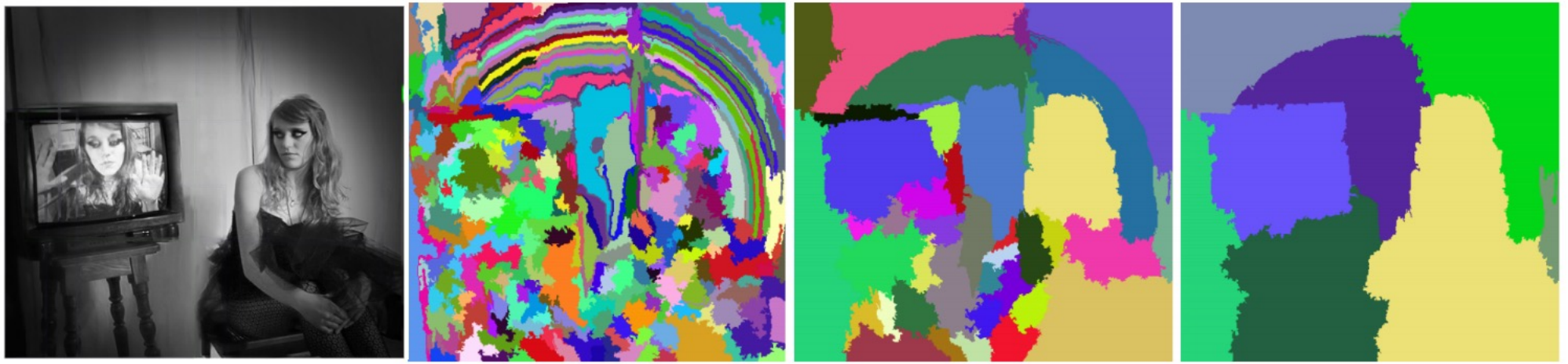


Candidate objects

[P. F. Felzenszwalb and D. P. Huttenlocher. “Efficient Graph-Based Image Segmentation.” IJCV, 59:167–181, 2004.]

Selective Search

Step 2: Recursively combine similar regions into larger ones.



Input Image | Initial Segmentation | After some iterations | After more iterations

Selective Search

Step 3: Use the generated regions to produce candidate object locations.

Input Image

R-CNN (Region-based Convolutional Neural Networks)

Input image

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

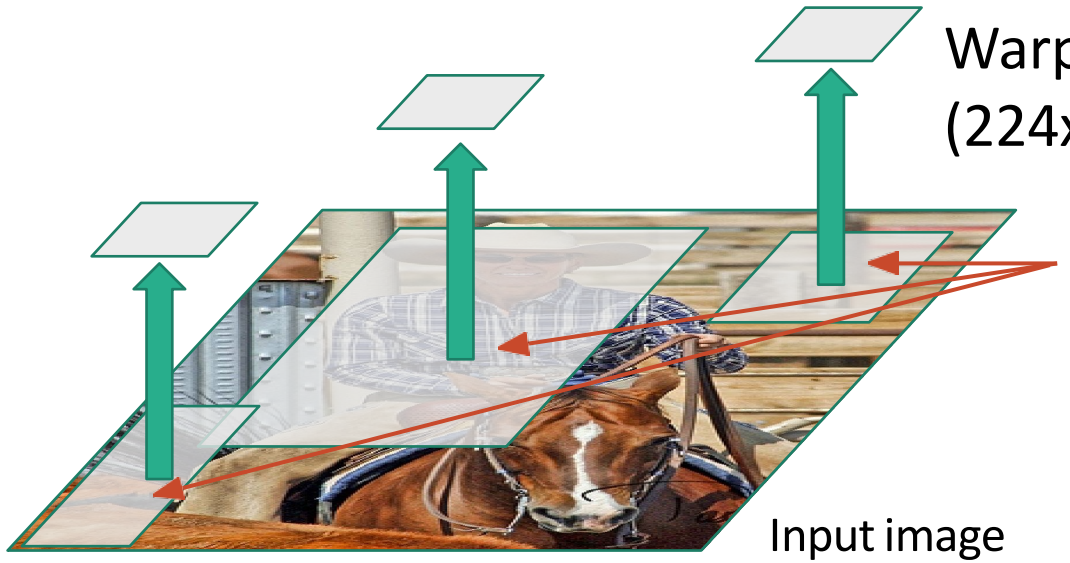
R-CNN

Regions of Interest (RoI) from a proposal method (~2k)

Input image

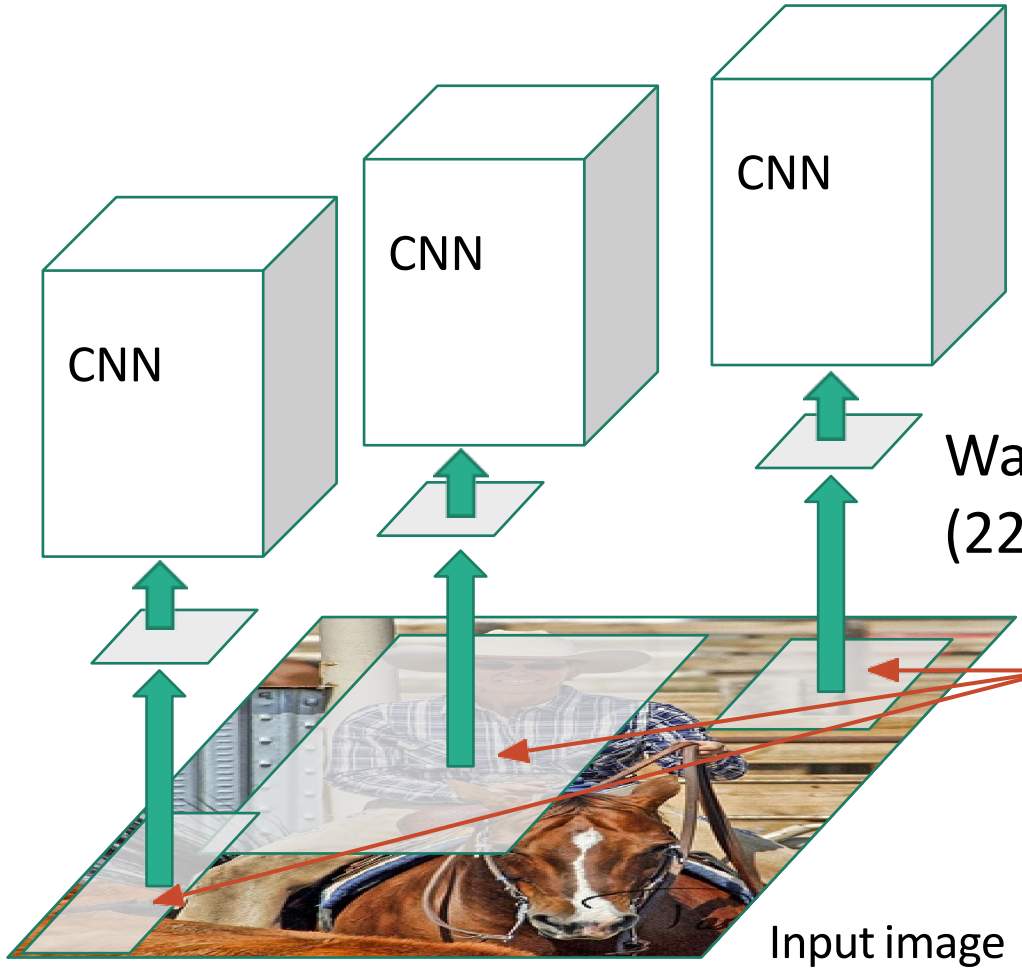
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

R-CNN



Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

R-CNN



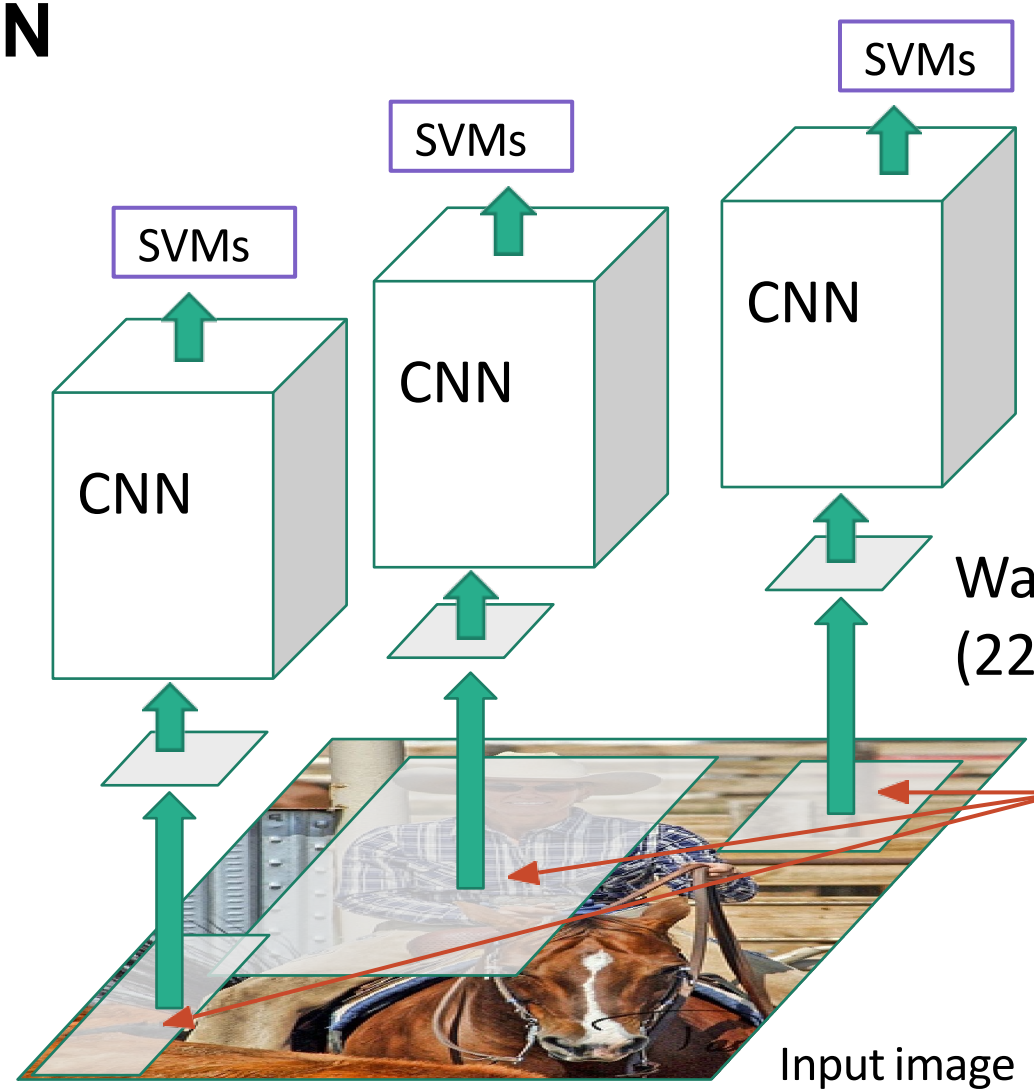
Forward each region through CNN/ConvNet (ImageNet-pretranied)

Warped image regions (224x224 pixels)

Regions of Interest (RoI) from a proposal method (~2k)

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

R-CNN



Classify regions with SVMs

Forward each region through CNN (ImageNet-pretranied)

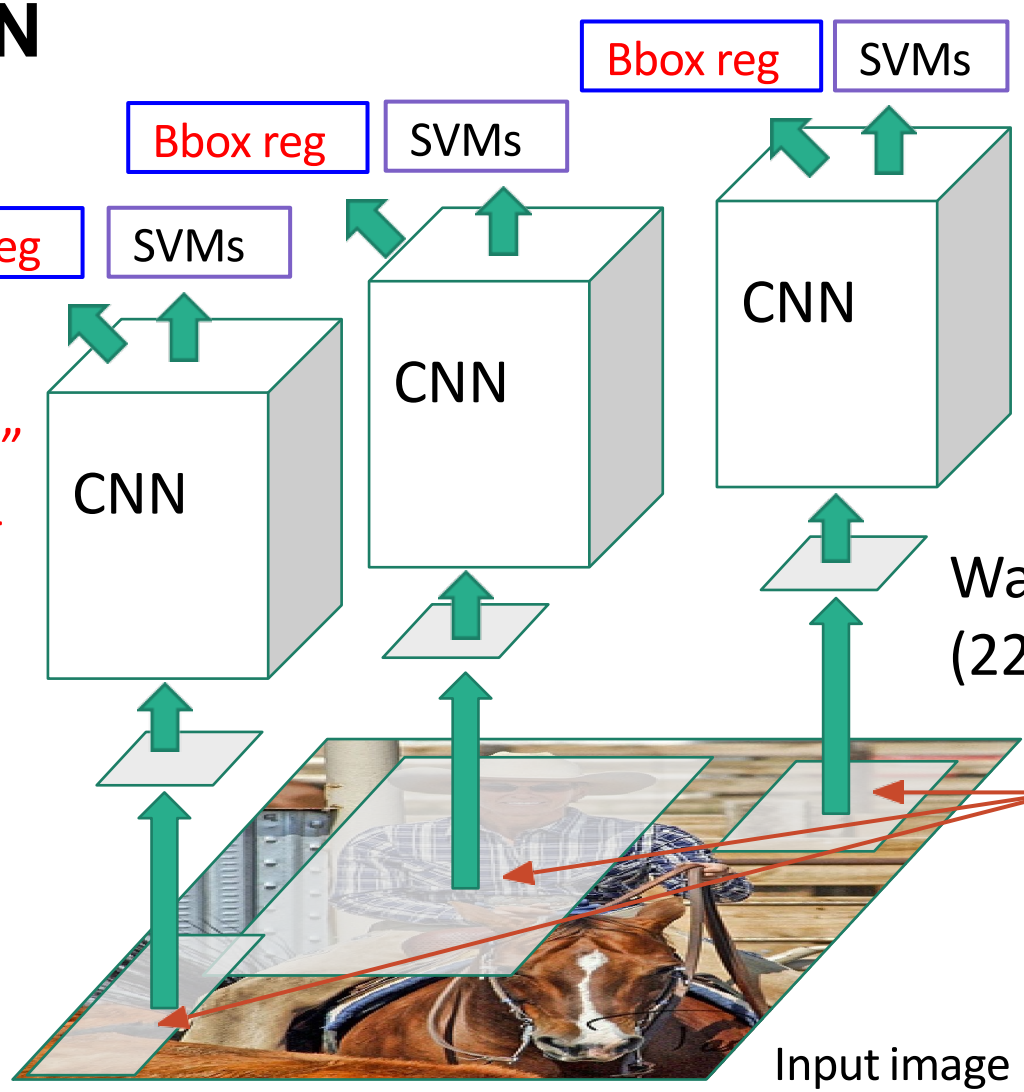
Warped image regions (224x224 pixels)

Regions of Interest (RoI) from a proposal method (~2k)

Input image

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

R-CNN



Predict "corrections" to the RoI: 4 numbers: (dx, dy, dw, dh)

Classify regions with SVMs

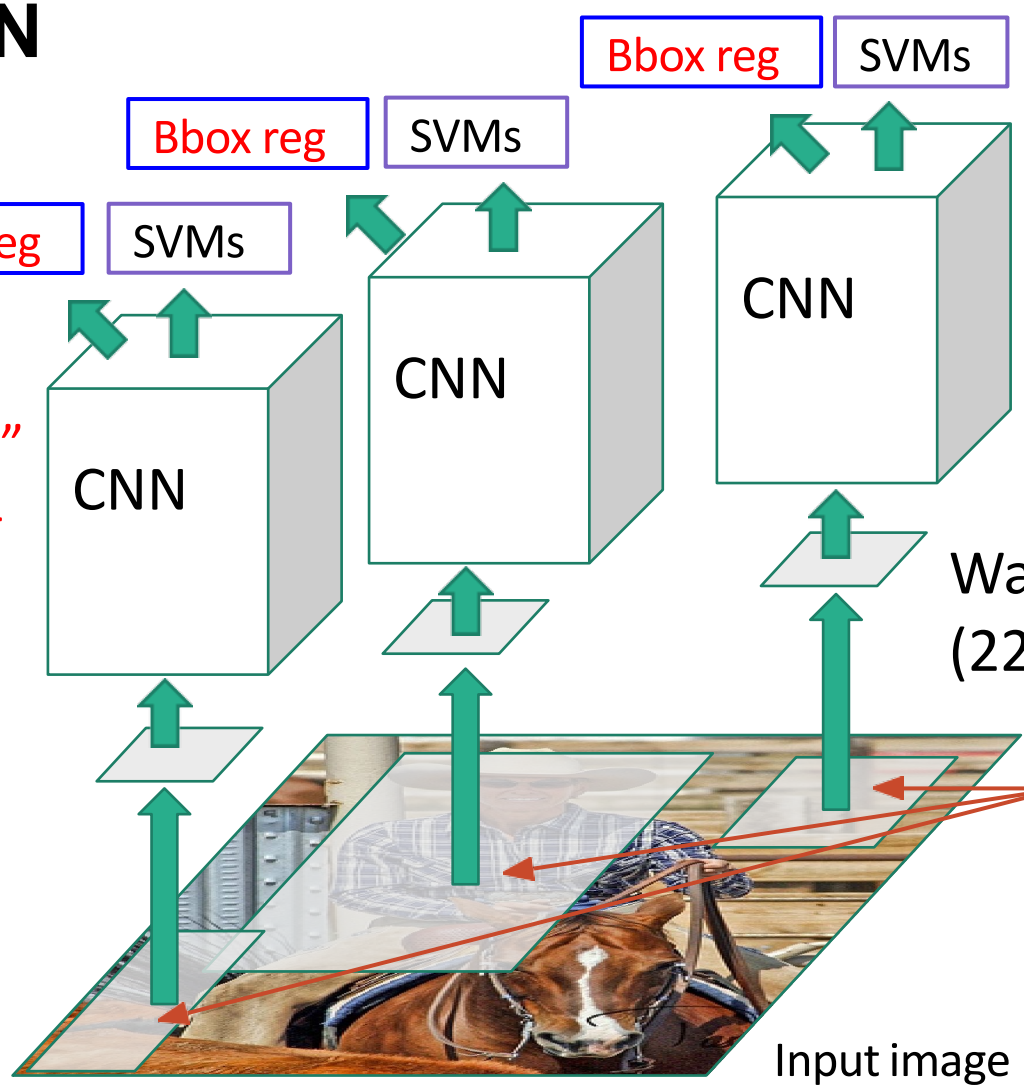
Forward each region through CNN (ImageNet-pretranied)

Warped image regions (224x224 pixels)

Regions of Interest (RoI) from a proposal method (~2k)

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

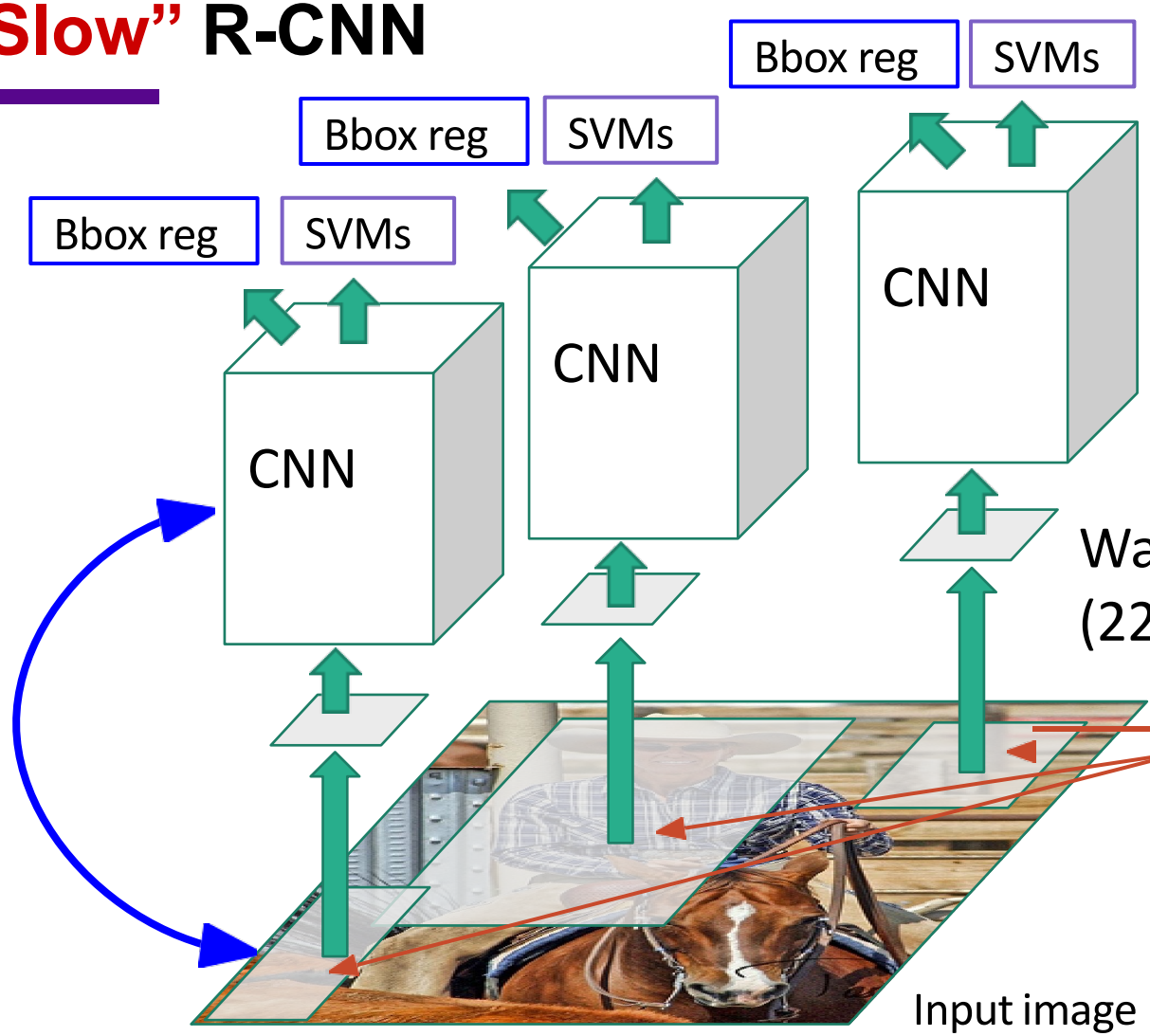
R-CNN



Problem: Very slow! Need to do ~2k independent forward passes for each image!

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

"Slow" R-CNN



Classify regions with SVMs

Forward each region through CNN

Problem: Very slow! Need to do ~2k independent forward passes for each image!

Idea: Pass the image through CNN before cropping! Crop the conv feature instead!

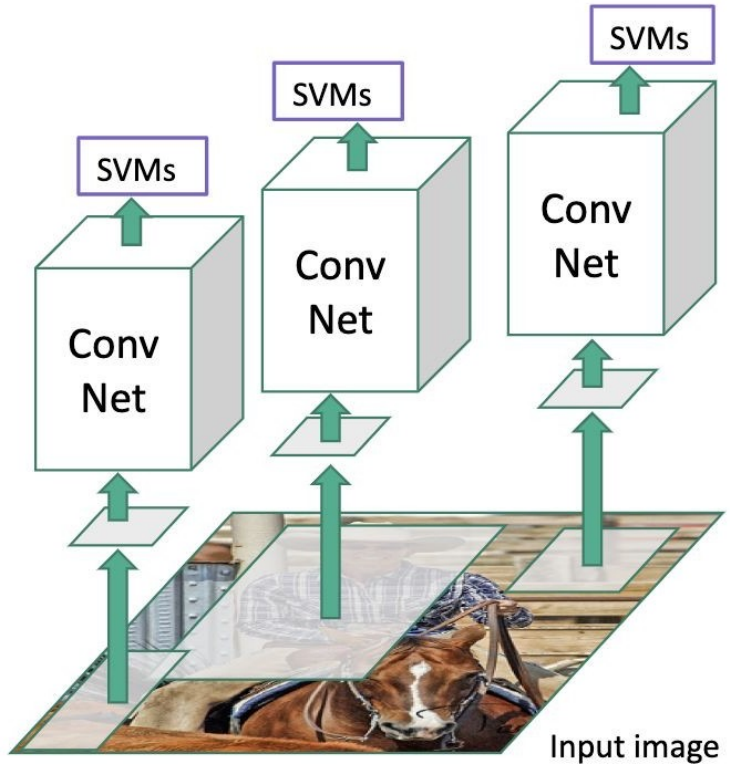
Can we remove duplicated computation?

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

Fast R-CNN

Input image

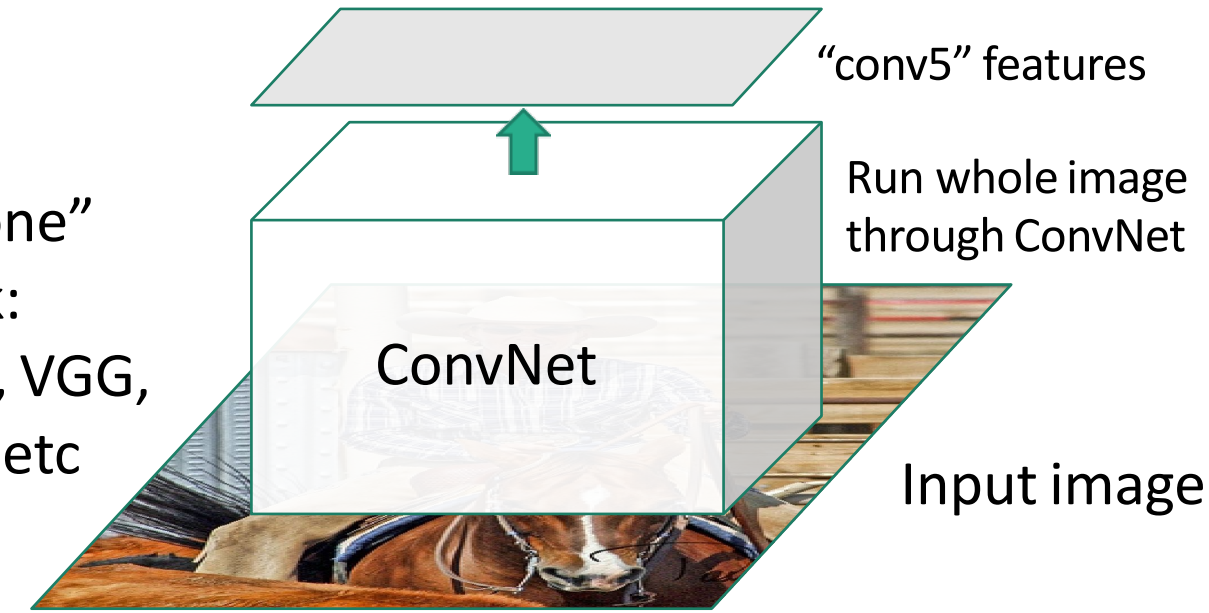
"Slow" R-CNN



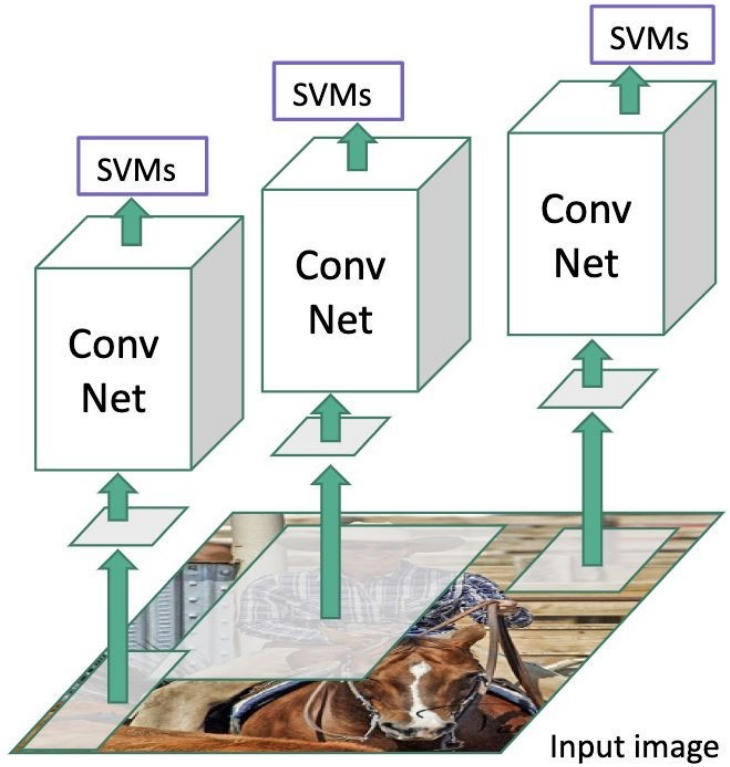
Girshick, "Fast R-CNN", ICCV 2015.

Fast R-CNN

"Backbone" network: AlexNet, VGG, ResNet, etc



"Slow" R-CNN



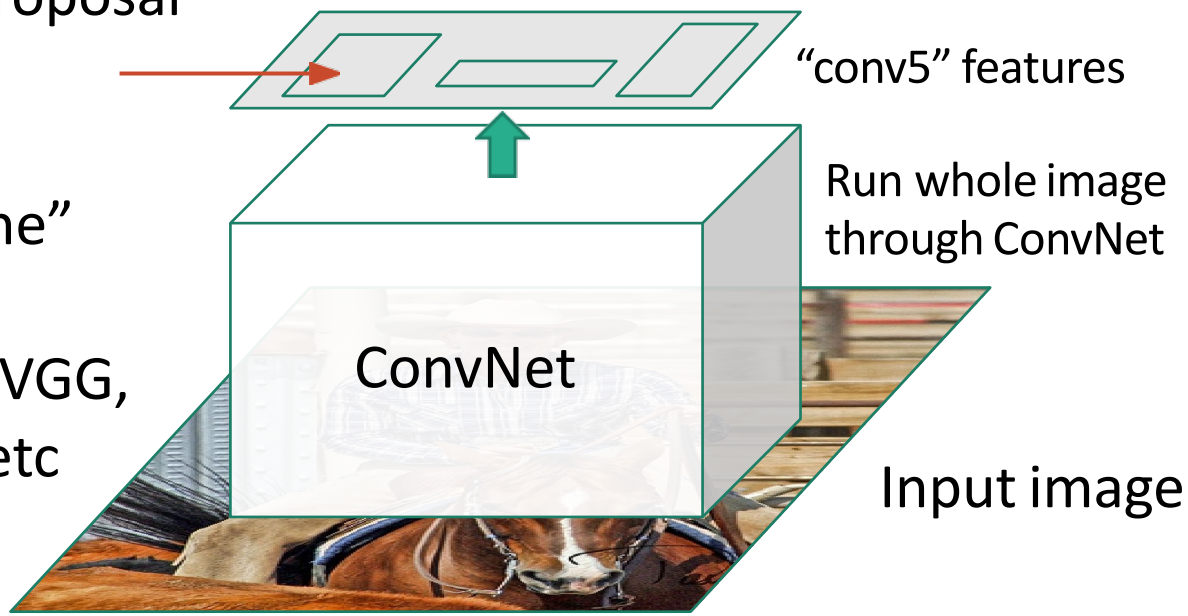
Girshick, "Fast R-CNN", ICCV 2015.

Fast R-CNN

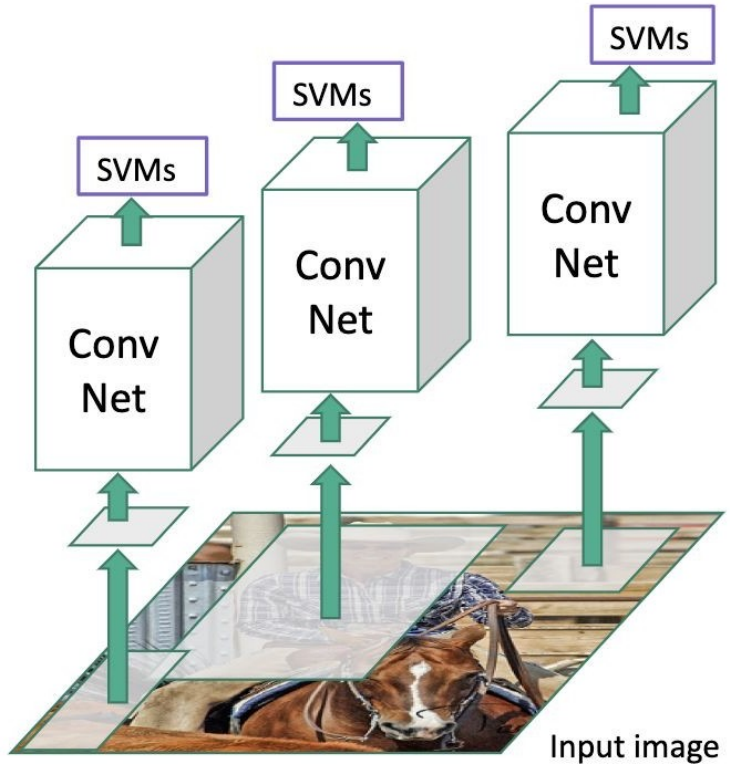
The region of interest in the image is simply projected on the feature map for each proposal

Regions of Interest (RoIs) from a proposal method

"Backbone" network: AlexNet, VGG, ResNet, etc



"Slow" R-CNN

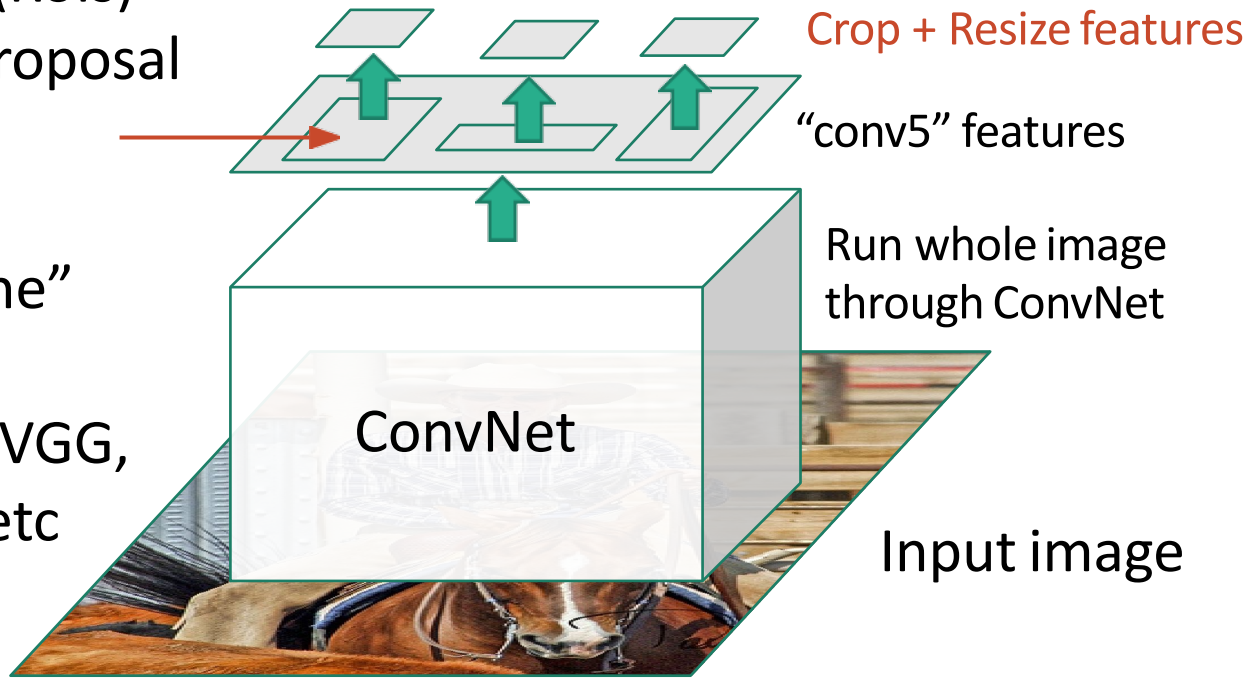


Girshick, "Fast R-CNN", ICCV 2015.

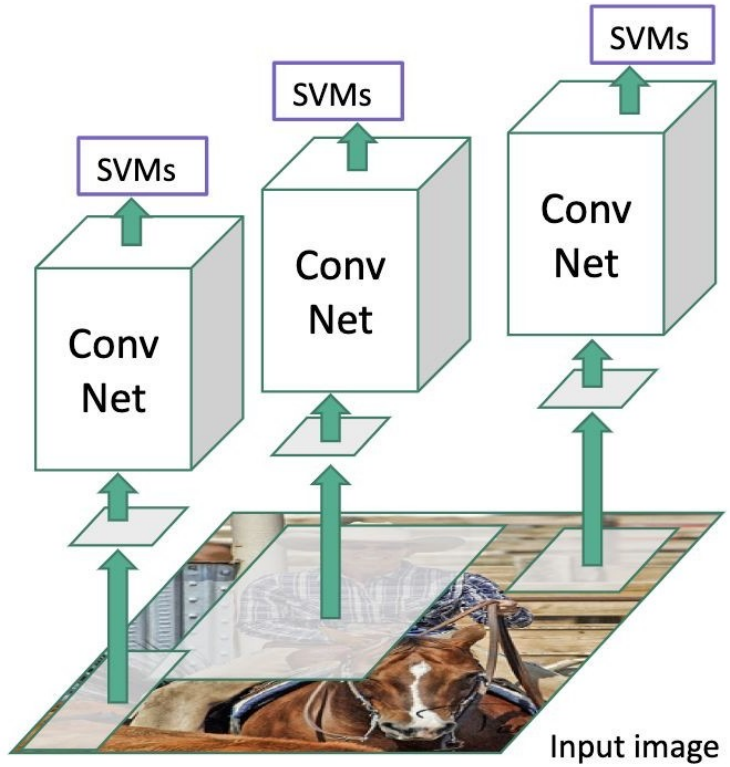
Fast R-CNN

Regions of Interest (RoIs) from a proposal method

"Backbone" network: AlexNet, VGG, ResNet, etc

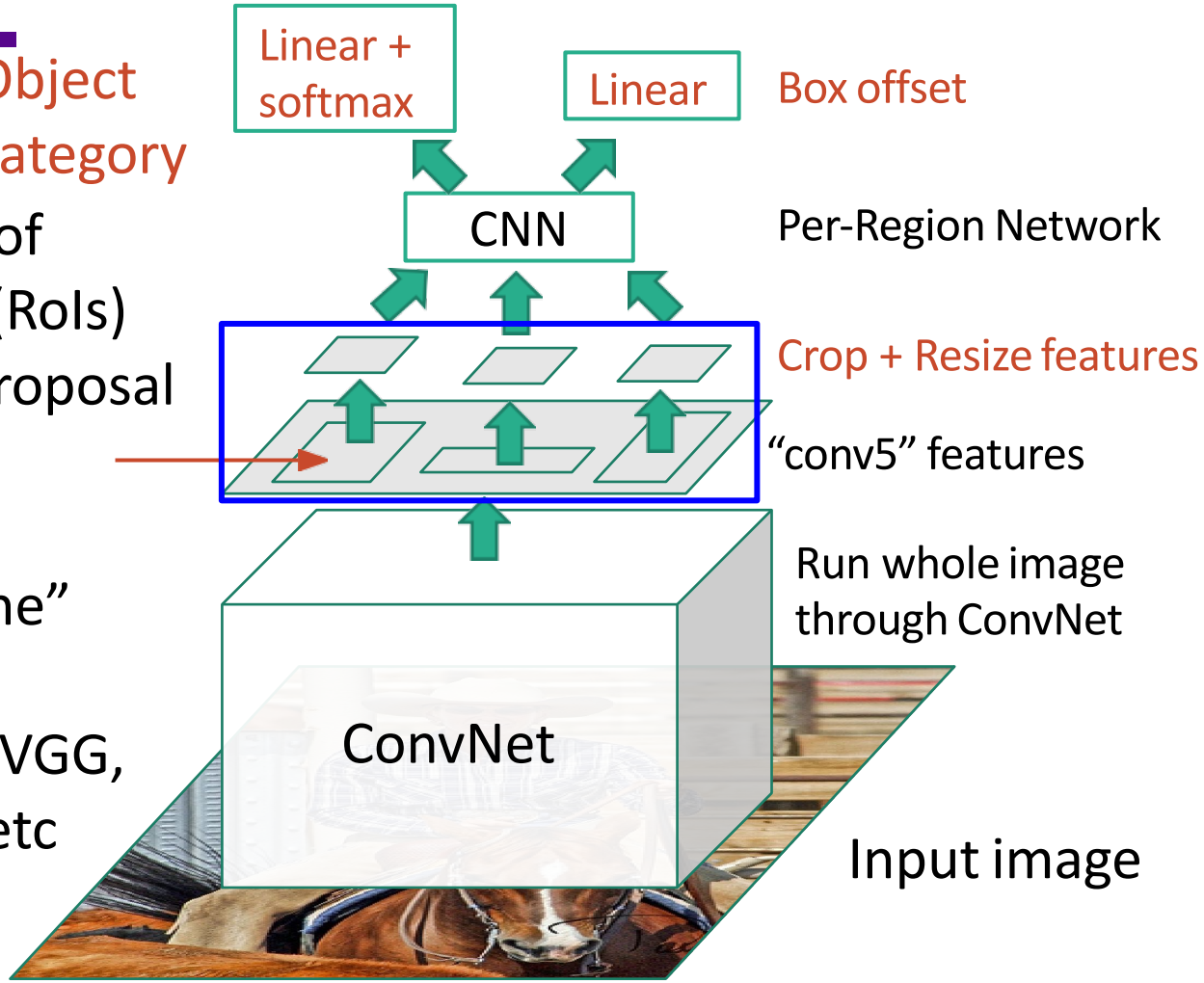


"Slow" R-CNN

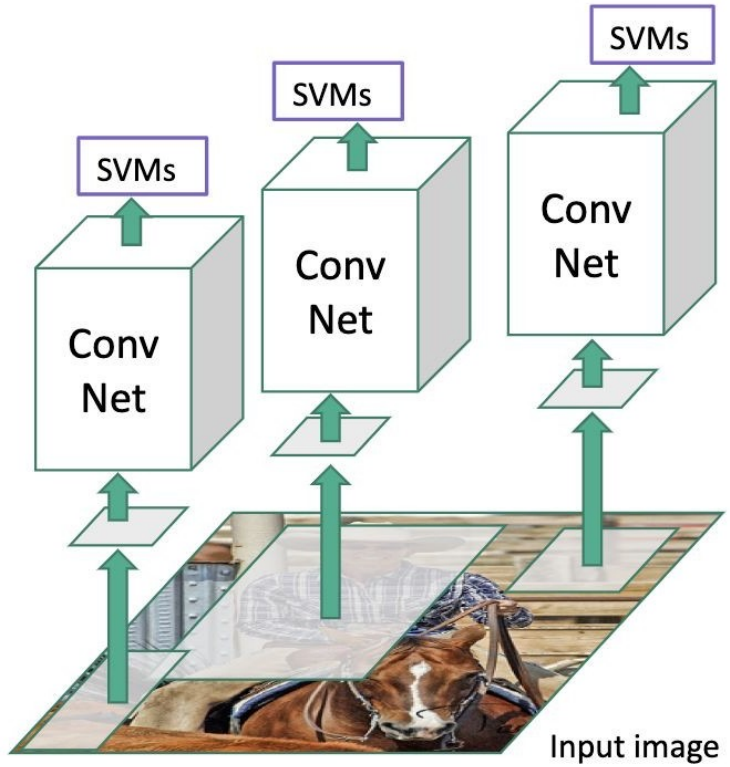


Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN

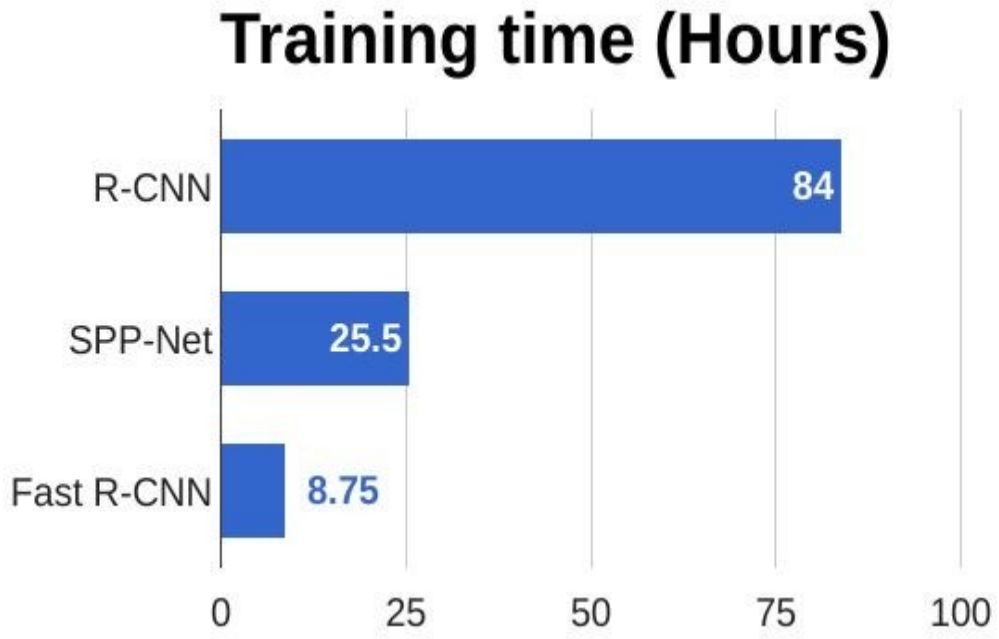


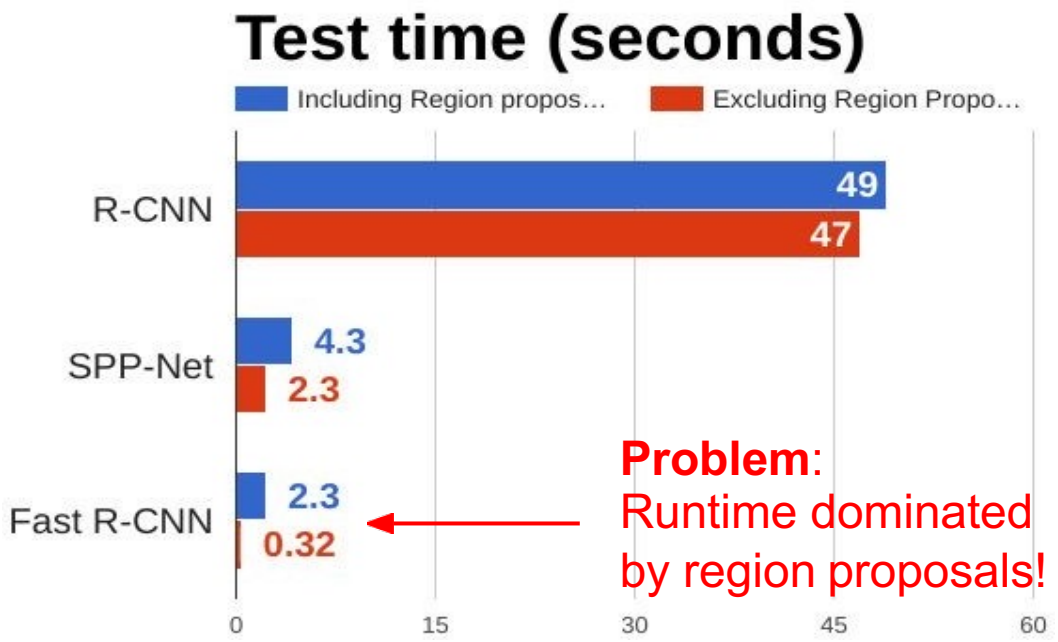
"Slow" R-CNN



Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN vs Fast R-CNN





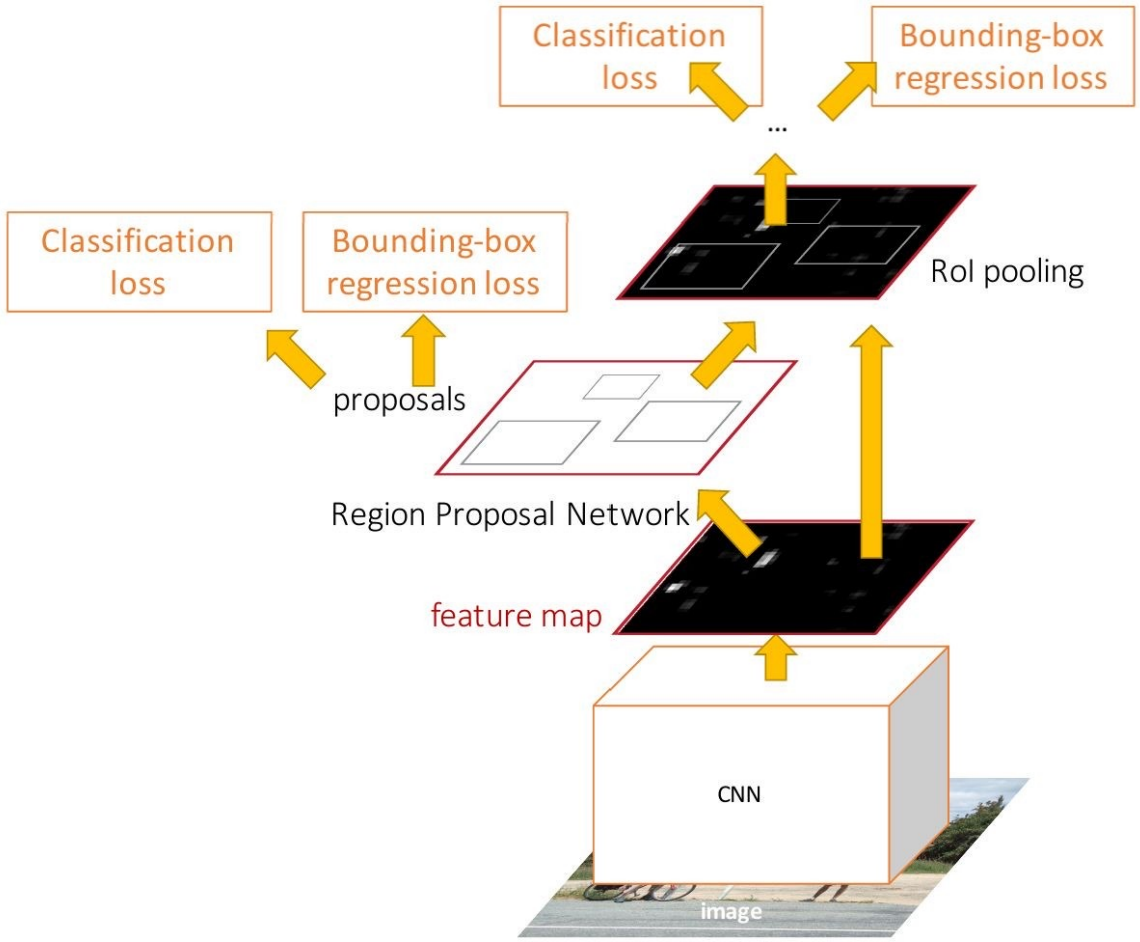
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
He et al, “Spatial pyramid pooling in deep convolutional networks for visual recognition”, ECCV 2014
Girshick, “Fast R-CNN”, ICCV 2015

Faster R-CNN

- Make CNN do region proposal.

Insert **Region Proposal Network (RPN)** to predict proposals from features

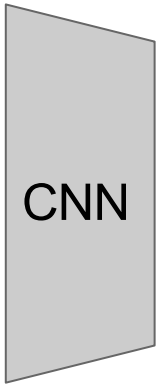
Otherwise same as Fast R-CNN: Crop features for each proposal, classify each one



Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015

Region Proposal Network (RPN)

Input Image
(e.g. 3 x 640 x 480)



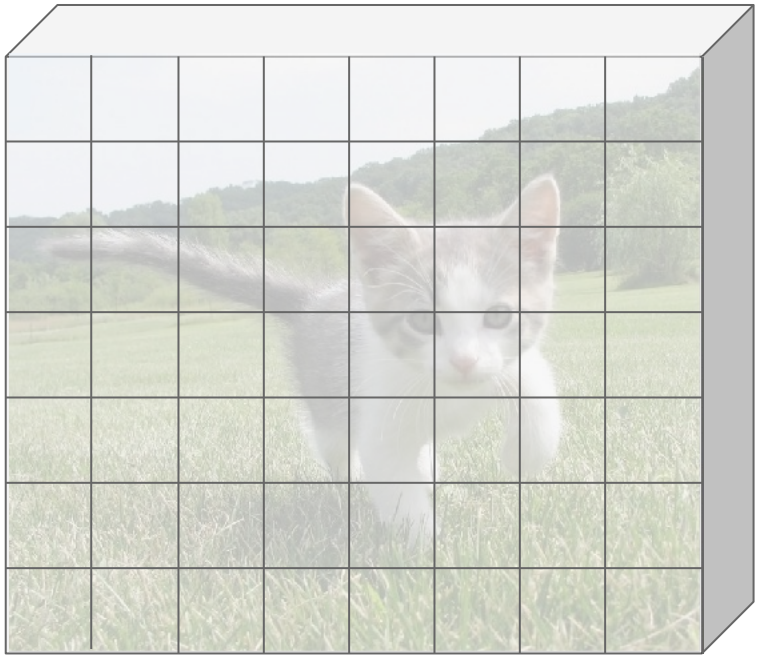


Image features
(e.g. 512 x 20 x 15)

Region Proposal Network: Anchor

Imagine an **anchor box** of fixed size at each point in the feature map

Input Image
(e.g. 3 x 640 x 480)

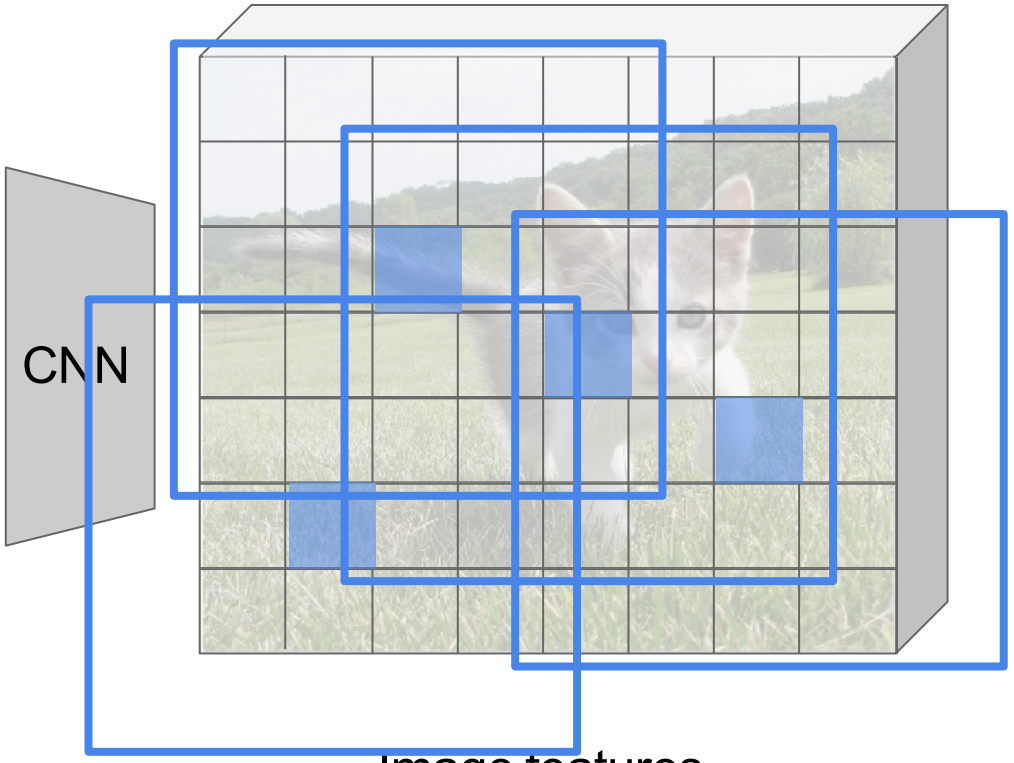


Image features
(e.g. 512 x 20 x 15)

Region Proposal Network: Anchor

Imagine an **anchor box** of fixed size at each point in the feature map

Input Image
(e.g. 3 x 640 x 480)

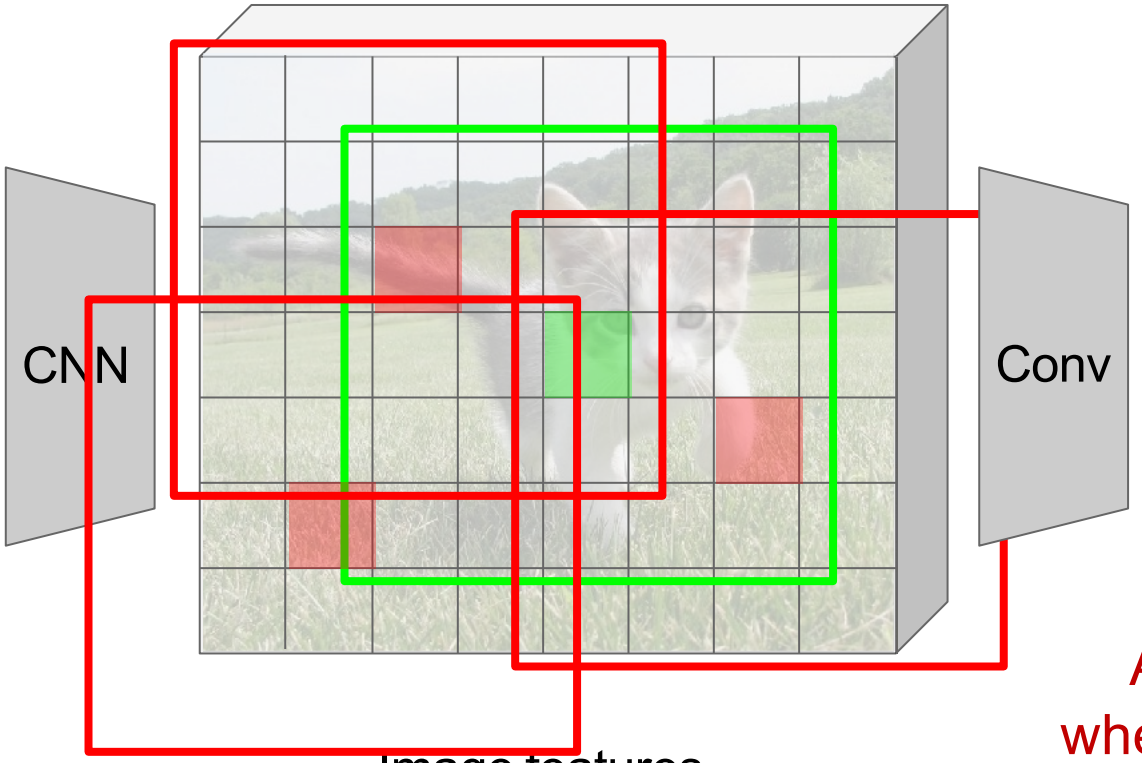


Image features
(e.g. 512 x 20 x 15)

Is this anchor an object?
1 x 20 x 15

At each point, predict whether the corresponding anchor contains an object (binary classification)

Region Proposal Network: Anchor

Imagine an **anchor box** of fixed size at each point in the feature map

Input Image
(e.g. 3 x 640 x 480)

CNN

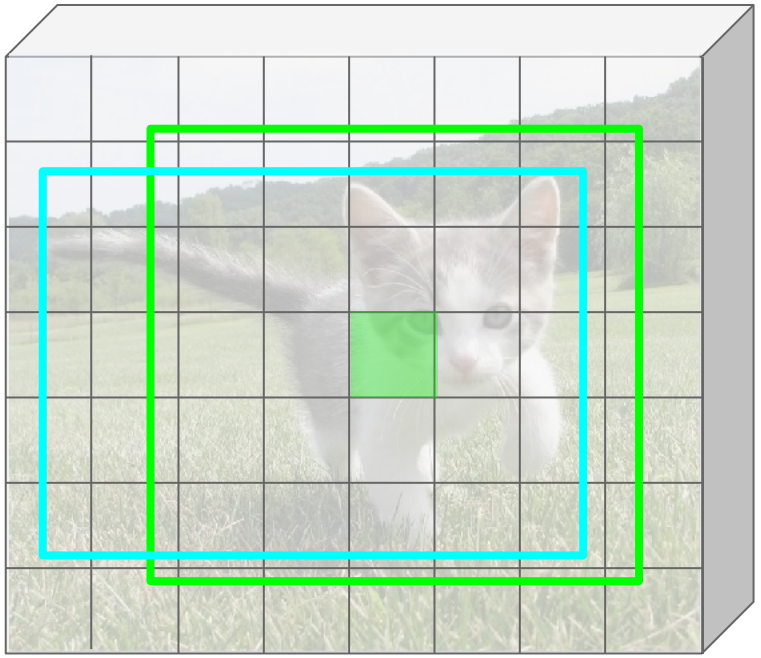


Image features
(e.g. 512 x 20 x 15)

Conv

Is this anchor an object?
1 x 20 x 15

Box corrections
4 x 20 x 15

For positive boxes, also predict a corrections from the anchor to the ground-truth box (regress 4 numbers per pixel)

Region Proposal Network: Anchor

In practice use K different anchor boxes of different size / scale at each point

Input Image
(e.g. 3 x 640 x 480)

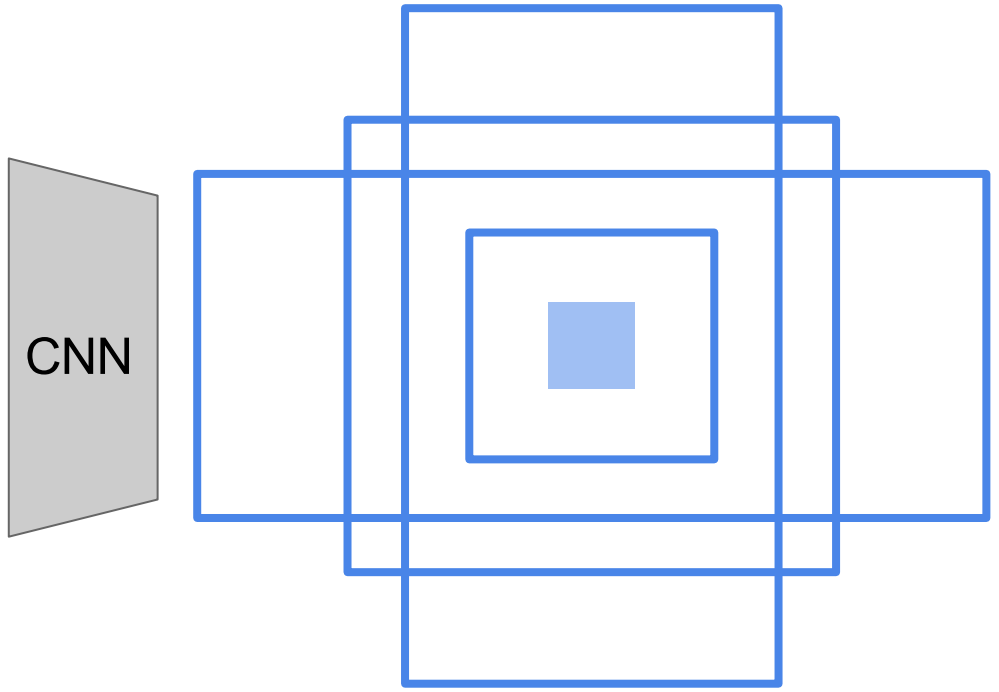


Image features
(e.g. 512 x 20 x 15)

Conv

Is this anchor an object?
K x 20 x 15

Box transforms
4K x 20 x 15

Region Proposal Network: Anchor

In practice use K different anchor boxes of different size / scale at each point

Input Image
(e.g. 3 x 640 x 480)

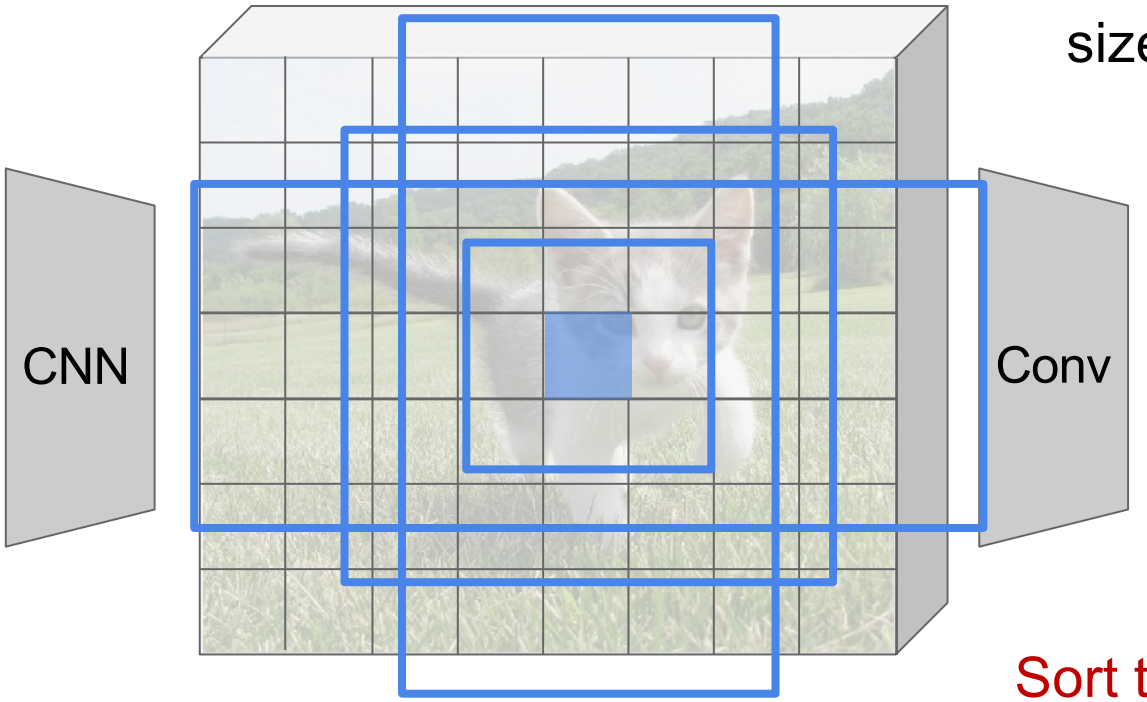


Image features
(e.g. 512 x 20 x 15)

Is this anchor an object?
K x 20 x 15

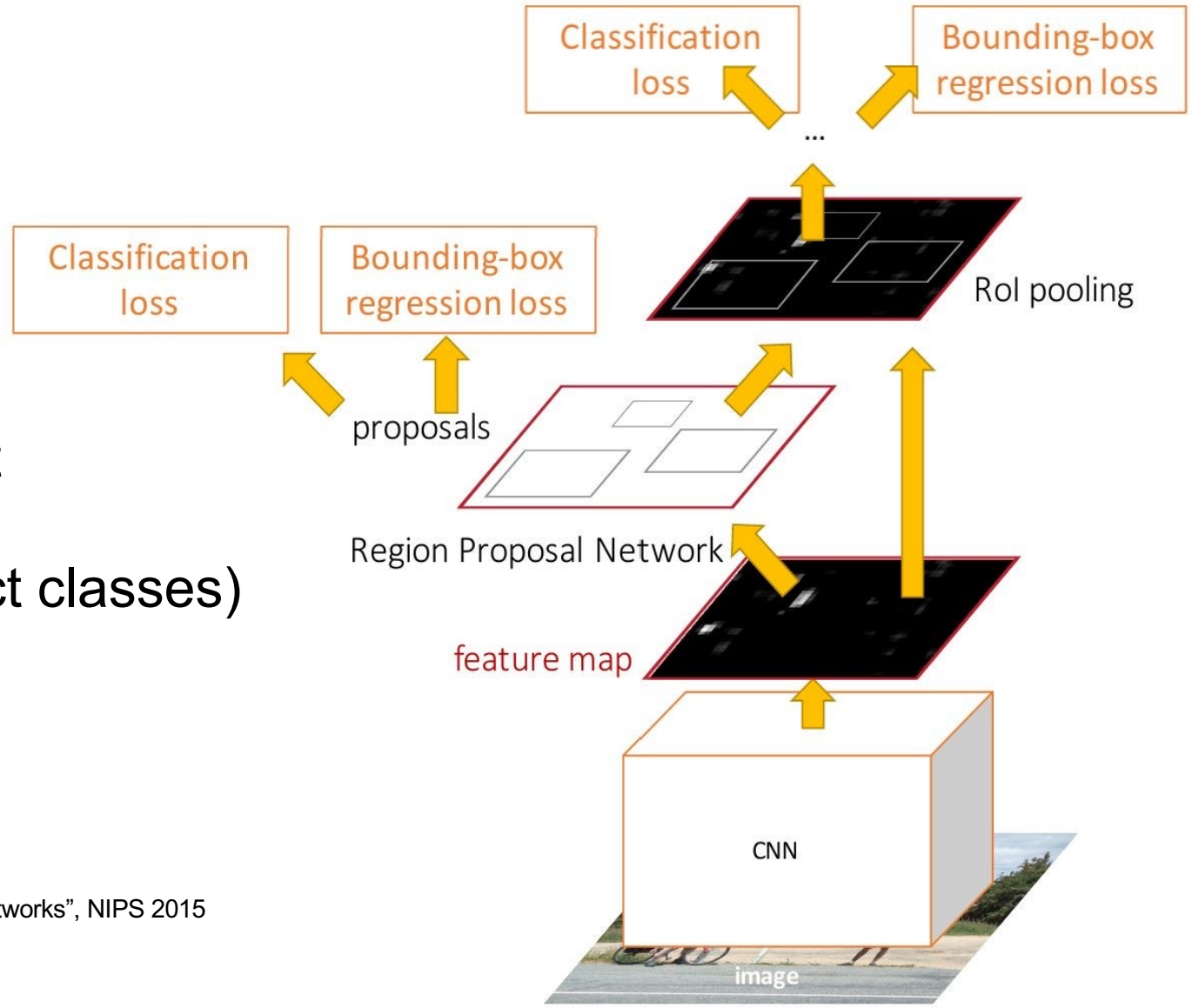
Box transforms
4K x 20 x 15

Sort the K*20*15 boxes by their “objectness” score, take top ~300 as our proposals

Faster R-CNN

Jointly train with 4 losses:

1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates



Ren et al, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, NIPS 2015

Faster R-CNN - Loss

Symbol	Explanation
p_i	Predicted probability of anchor i being an object.
p_i^*	Ground truth label (binary) of whether anchor i is an object.
t_i	Predicted four parameterized coordinates.
t_i^*	Ground truth coordinates.
N_{cls}	Normalization term, set to be mini-batch size (~256) in the paper.
N_{box}	Normalization term, set to the number of anchor locations (~2400) in the paper.
λ	A balancing parameter, set to be ~10 in the paper (so that both $\mathcal{L}_{\text{cls}}$ and $\mathcal{L}_{\text{box}}$ terms are roughly equally weighted).

The multi-task loss function combines the losses of classification and bounding box regression:

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{box}}$$

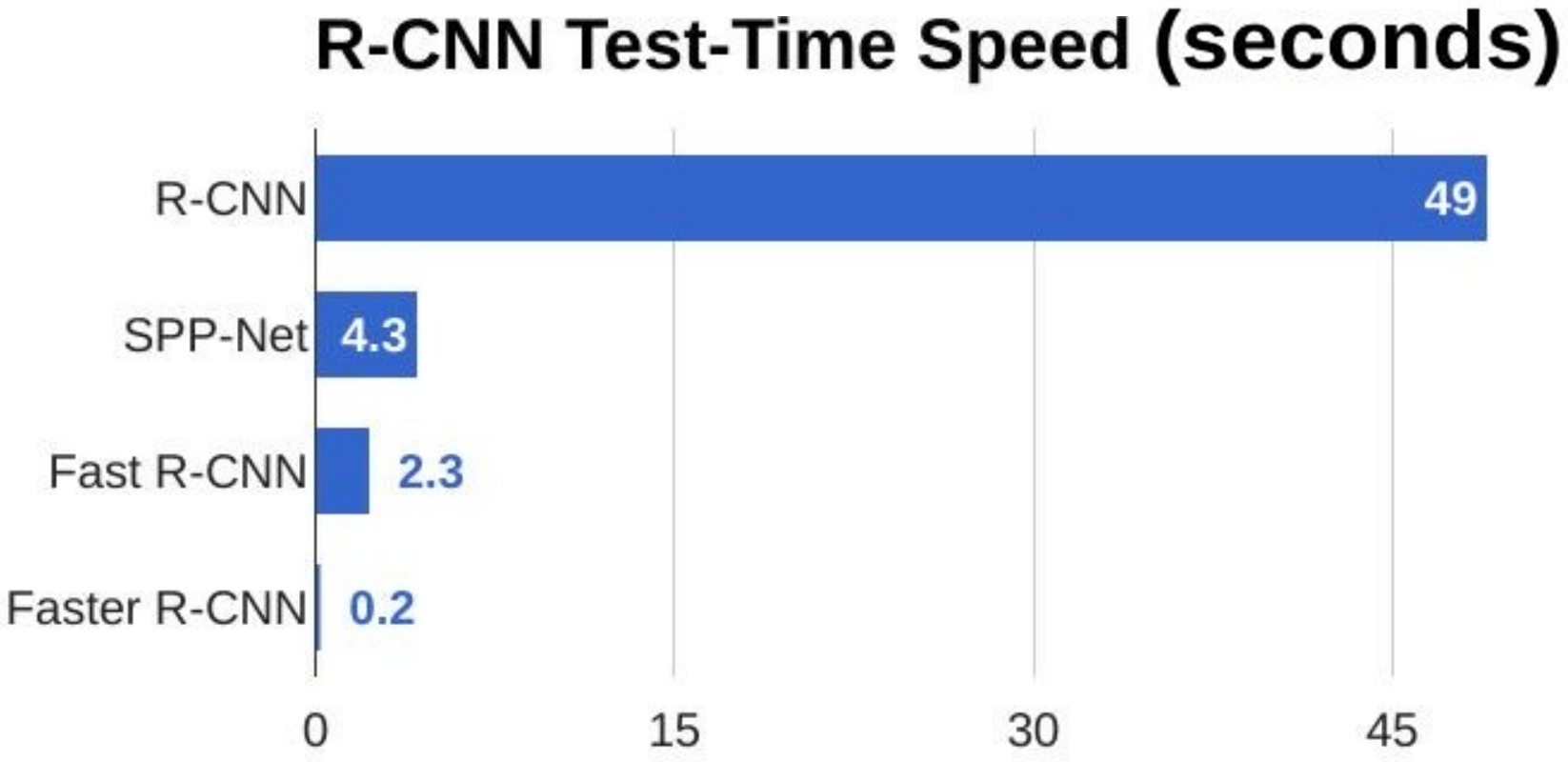
$$\mathcal{L}(\{p_i\}, \{t_i\}) = \frac{1}{N_{\text{cls}}} \sum_i \mathcal{L}_{\text{cls}}(p_i, p_i^*) + \frac{\lambda}{N_{\text{box}}} \sum_i p_i^* \cdot L_1^{\text{smooth}}(t_i - t_i^*)$$

where $\mathcal{L}_{\text{cls}}$ is the log loss function over two classes, as we can easily translate a multi-class classification into a binary classification by predicting a sample being a target object versus not. L_1^{smooth} is the smooth L1 loss.

$$\mathcal{L}_{\text{cls}}(p_i, p_i^*) = -p_i^* \log p_i - (1 - p_i^*) \log(1 - p_i)$$

https://lilianweng.github.io/lil-log/2017/12/31/object-recognition-for-dummies-part-3.html#loss-function-1

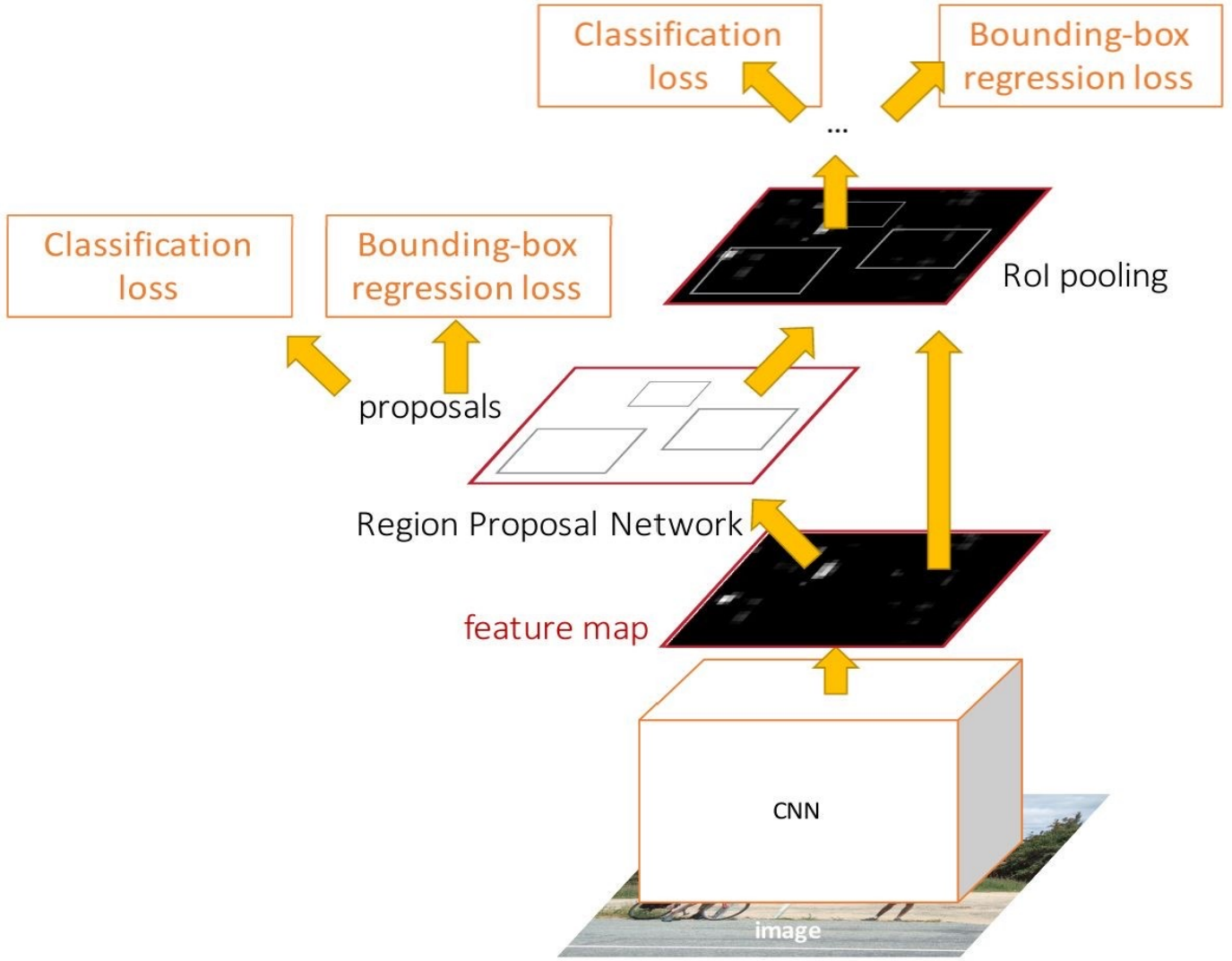
Faster R-CNN


R-CNN Test-Time Speed (seconds)
R-CNN
49
SPP-Net
4.3
Fast R-CNN
2.3
Faster R-CNN
0.2
0
15
30
45

Faster R-CNN

Glossing over many details:

- Ignore overlapping proposals with **non-max suppression**
- How are anchors determined?
- How do we sample positive / negative samples for training the RPN?
- How to parameterize bounding box regression?



Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015

Non-Max Suppression (NMS)

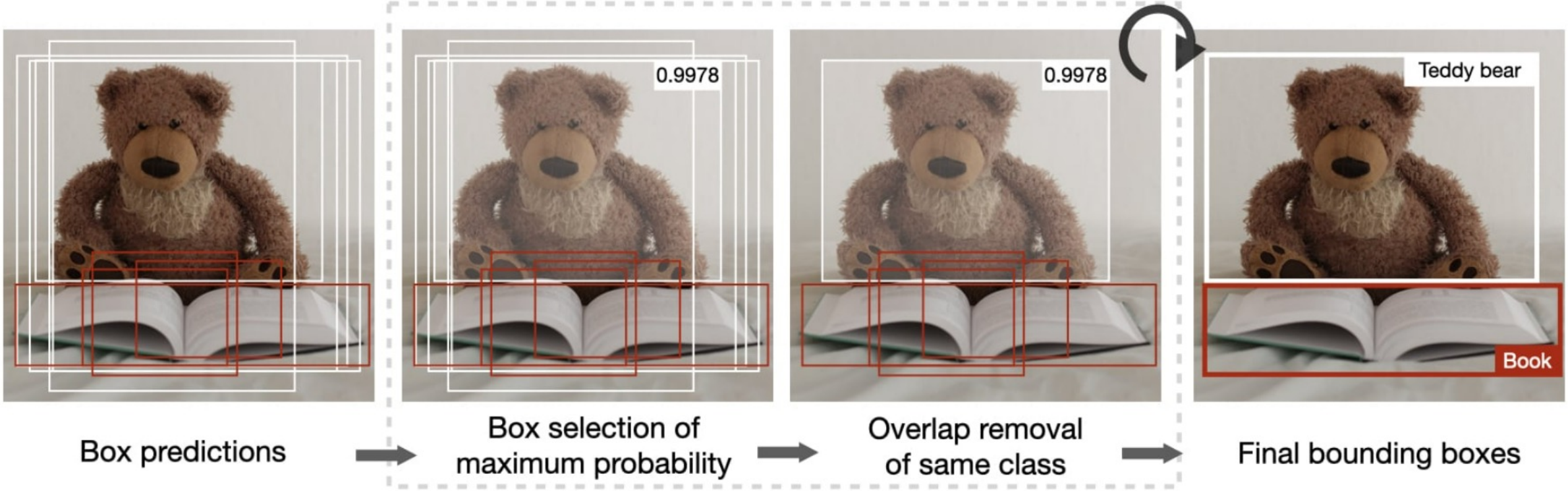


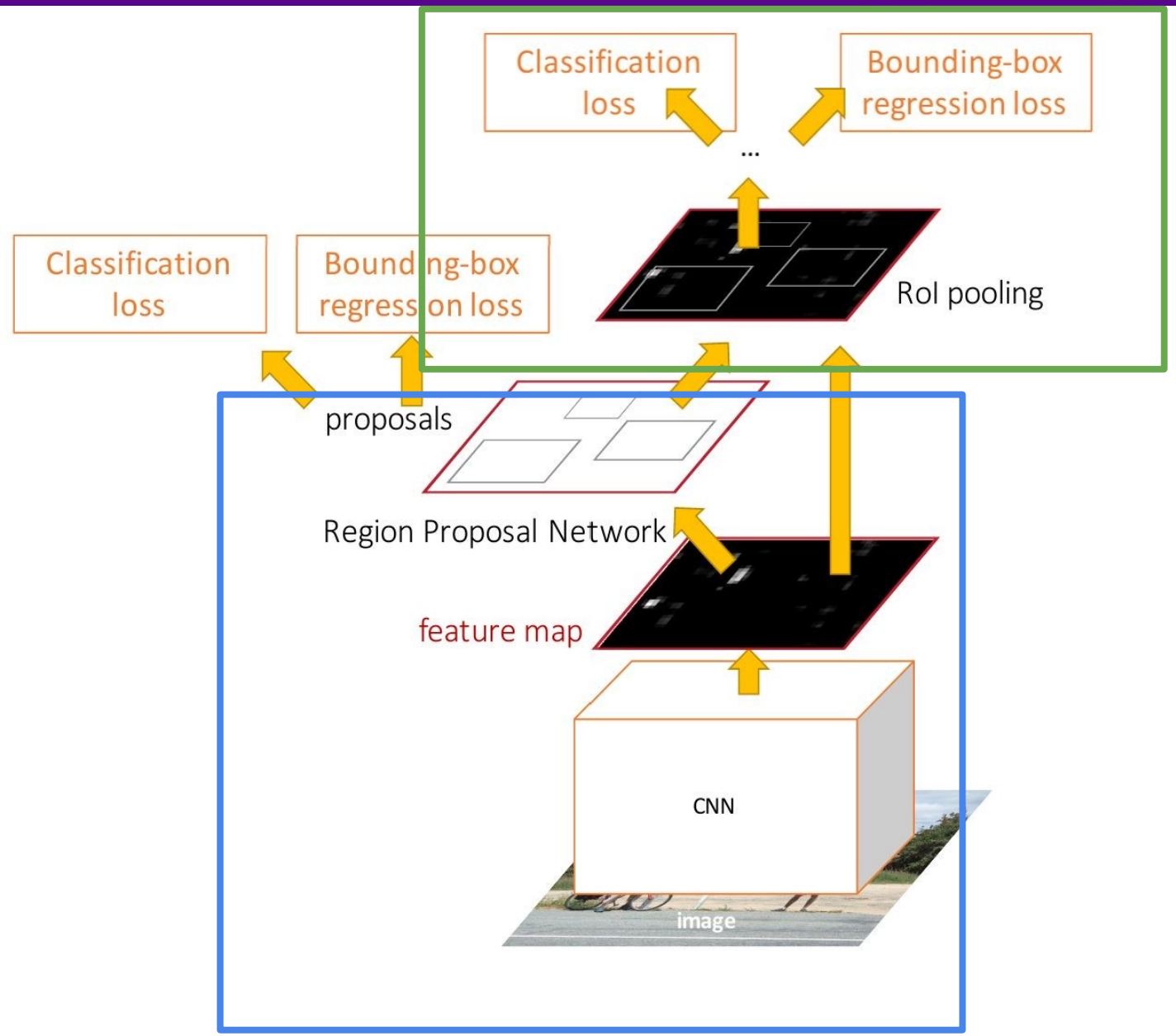
Image from: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks#object-detection

Faster R-CNN

Faster R-CNN is a
Two-stage object detector

First stage: Run once per image
- Backbone network
- Region proposal network

Second stage: Run once per region
- Crop features: RoI pool / align
- Predict object class
- Prediction bbox offset

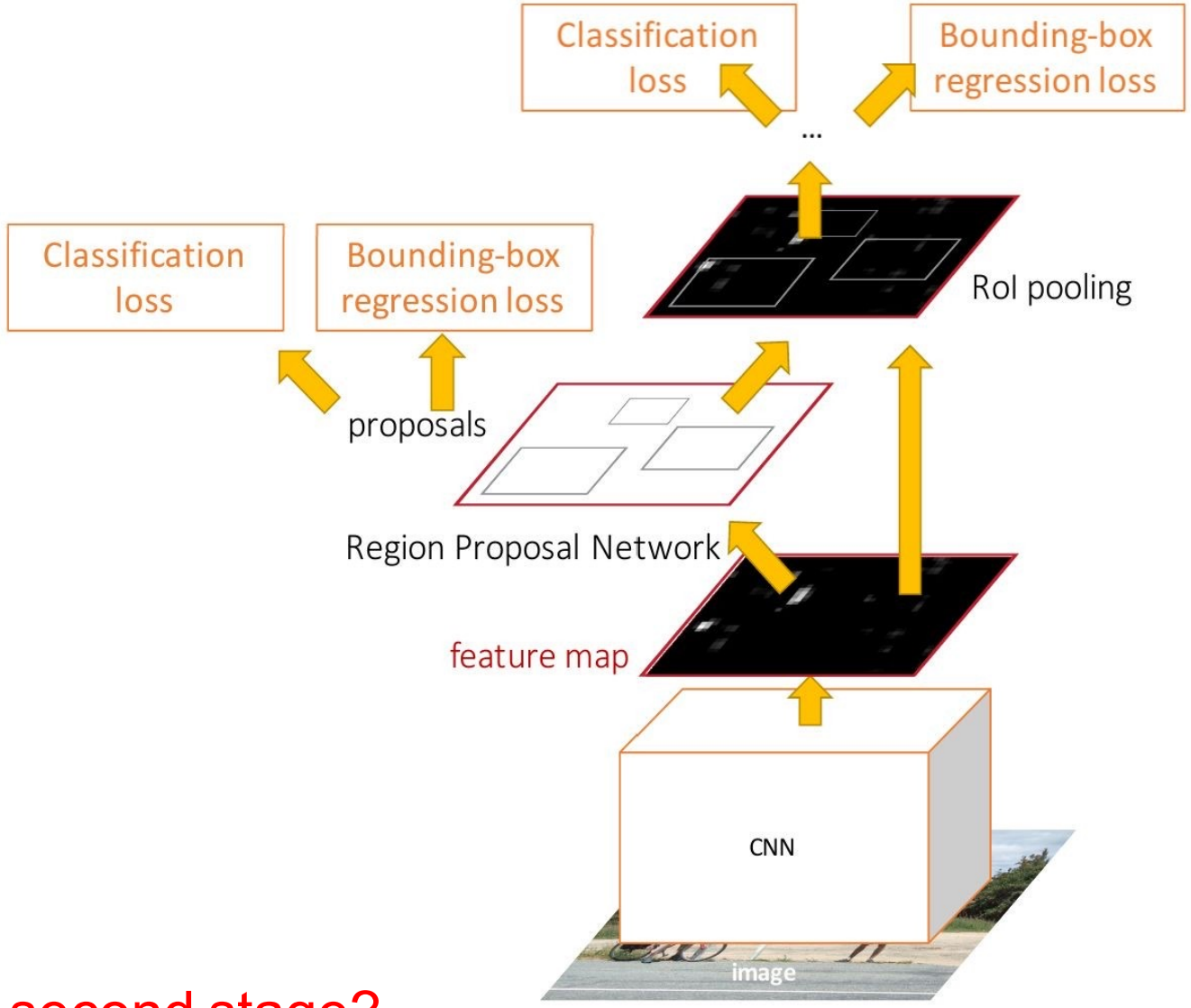


Faster R-CNN

Faster R-CNN is a
Two-stage object detector

First stage: Run once per image
- Backbone network
- Region proposal network

Second stage: Run once per region
- Crop features: RoI pool / align
- Predict object class
- Prediction bbox offset

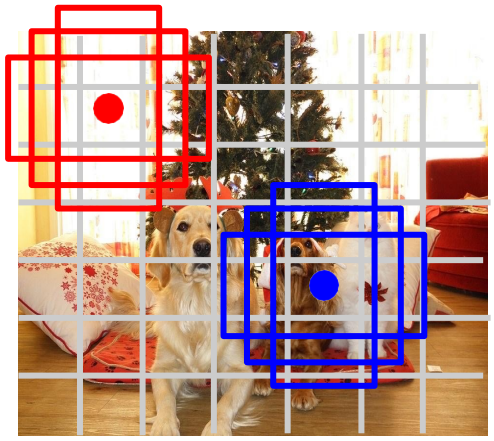


Do we really need the second stage?

Single Stage Object Detection

- A one-stage detector requires only a single pass through the neural network and predicts all the bounding boxes in one go.
- Unlike two-stage detectors, these methods are faster and don't have to deal with a large number of design choices.

Input image
3 x H x W



Divide image into grid
7 x 7

Image a set of **base boxes** centered at each grid cell
Here B = 3

Within each grid cell:

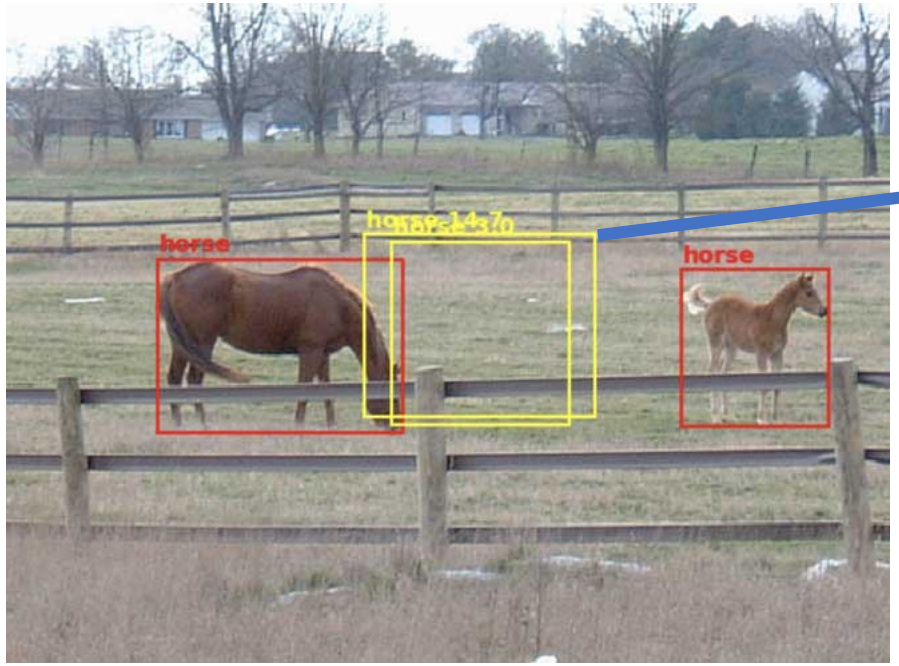
- Regress from each of the B base boxes to a final box with 5 numbers: (dx, dy, dh, dw, confidence)
- Predict scores for each of C classes (including background as a class)
- Looks a lot like RPN, but category-specific!

Output:
7 x 7 x B x (5 + C)

Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016
Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

Why Grids?

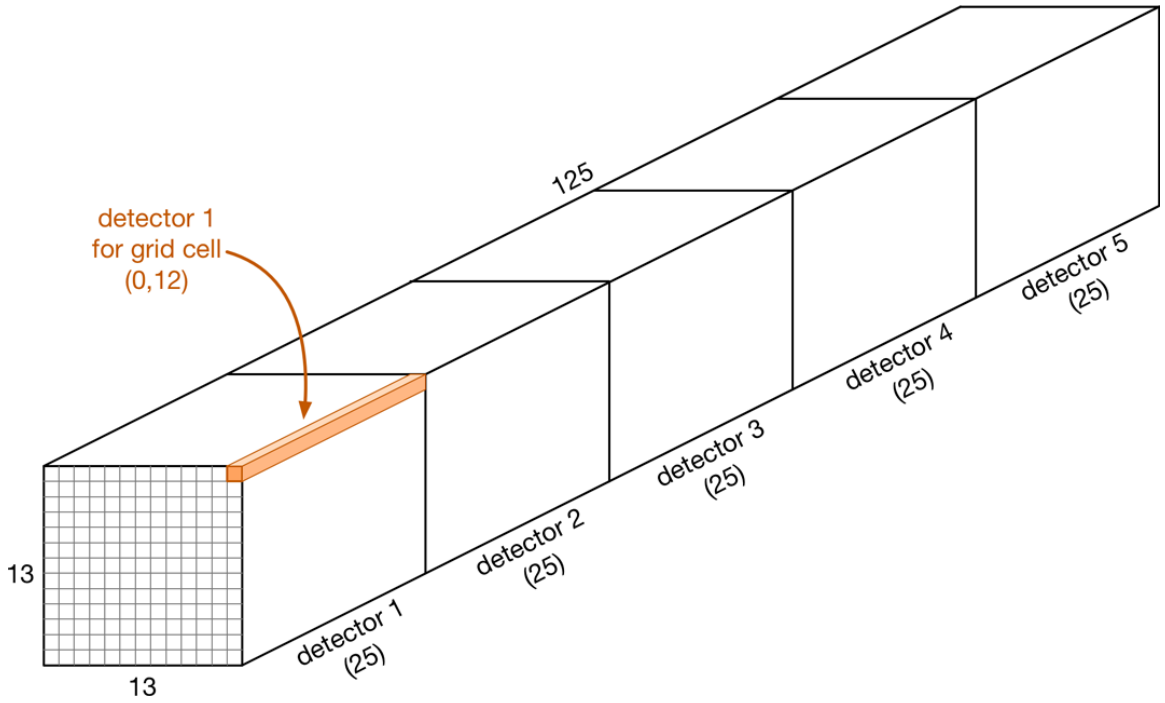
- When the network is expected to directly regress N bounding boxes, the network does not know to automatically assign its bounding boxes to the objects present.
- As a result, it is possible that all the regression outputs target the same object and therefore, becomes ineffective as shown below



The model doesn't know which bounding box should be assigned to which object, and to be safe, it puts them both somewhere in the middle

Why Grids?

- Using a **fixed grid of detectors** is the main idea that powers one-stage detectors, and what sets them apart from region proposal-based detectors such as R-CNN

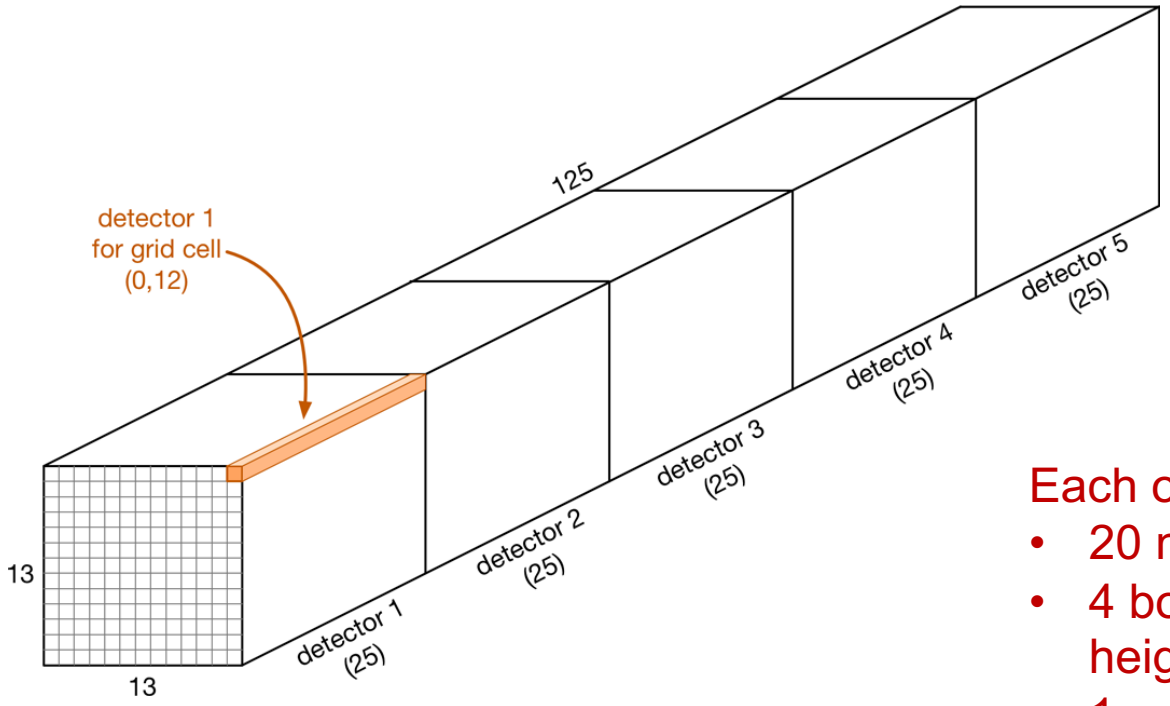


We interpret this feature map as being a grid of 13 by 13 cells. Each cell in the grid has B=5 independent object detectors, and each of these detectors predicts a single bounding box.

The key thing here is that the position of a detector is fixed: it can only detect objects located near that cell.

Why Grids?

- Using a **fixed grid of detectors** is the main idea that powers one-stage detectors, and what sets them apart from region proposal-based detectors such as R-CNN



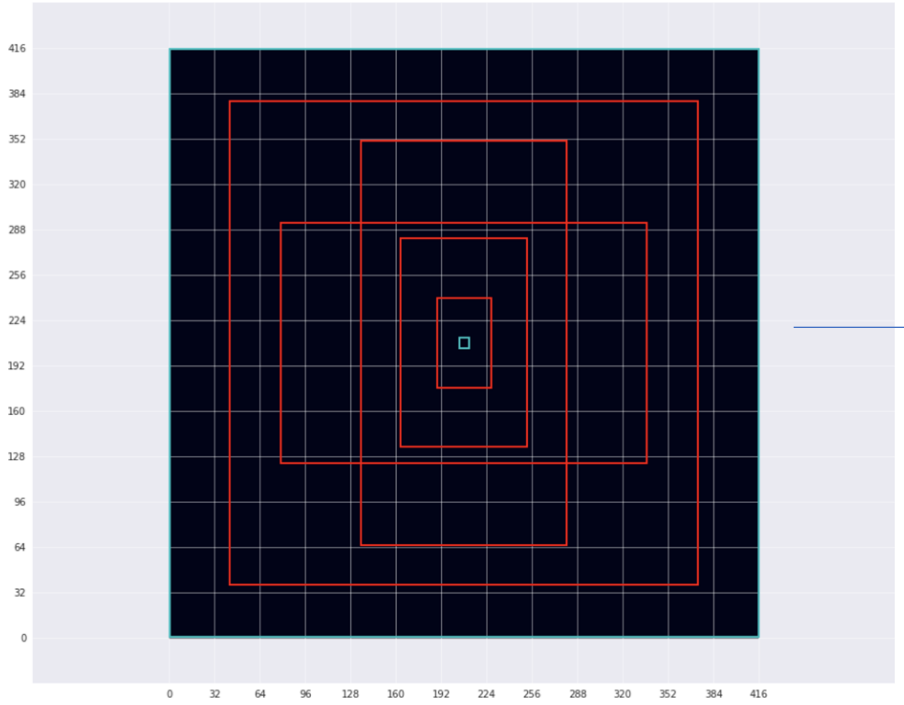
We interpret this feature map as being a grid of 13 by 13 cells. Each cell in the grid has B=5 independent object detectors, and each of these detectors predicts a single bounding box.

Each object detector produces 25 numbers:

- 20 numbers containing the class probabilities
- 4 bounding box coordinates (center x, center y, width, height)
- 1 confidence score

Anchors: What Are They?

- The grid is a useful constraint that limits *where* in the image a detector can find objects. We can also add another constraint that helps the model make better predictions, and that is a constraint on the *shape of the object*.
- Just like it's hard for a detector to learn how to predict objects that can be located anywhere, it's also hard for a detector to learn to predict objects that can be any shape or size.



These five shapes are called the **anchors** or anchor boxes

Anchors: How to Generate Them?

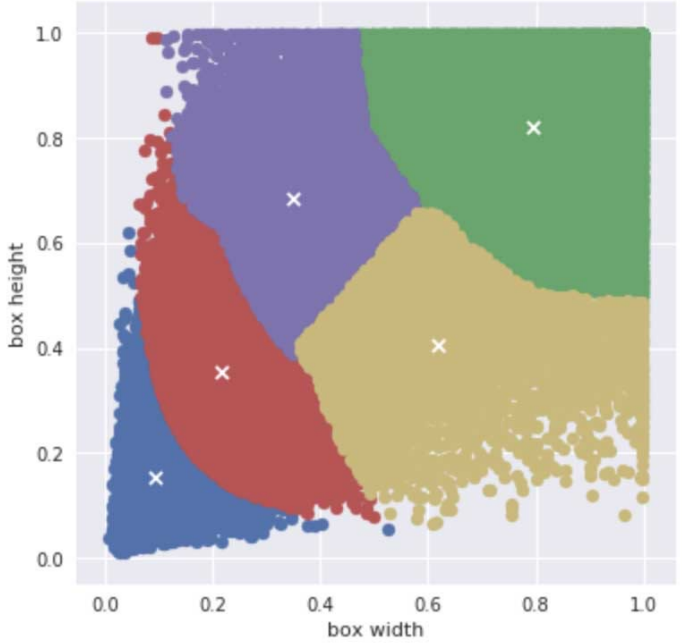
- There is one anchor for each detector in the grid cells. Just like the grid puts a location constraint on the detectors, anchors force the detectors inside the cells to each specialize in a particular object shape

- It's important to understand that these anchors are chosen beforehand. They're constants and they *won't change* during training

- YOLO chooses the anchors by running **k-means clustering** on all the bounding boxes from all the training images (with $k = 5$ so it finds the five most common object shapes). Therefore, YOLO's anchors are *specific to the dataset* that you're training (and testing) on.

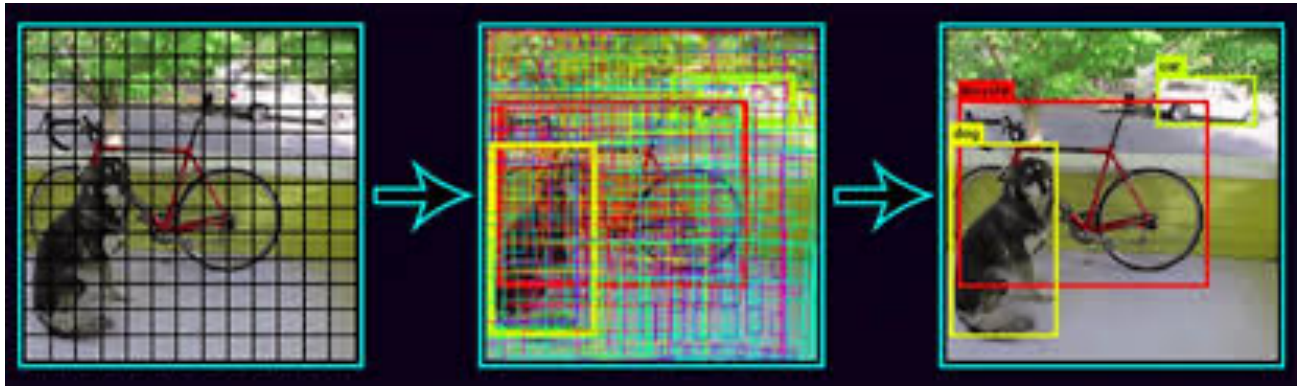
Anchors: How to Generate Them?

- These clusters represent five “averages” of the different object shapes that are present in this dataset. You can see that k-means found it necessary to group very small objects together in the blue cluster, slightly larger objects in the red cluster, and very large objects in green.
- It decided to split medium objects into two groups: one where the bounding boxes are wider than tall (yellow), and one that’s taller than wide (purple).



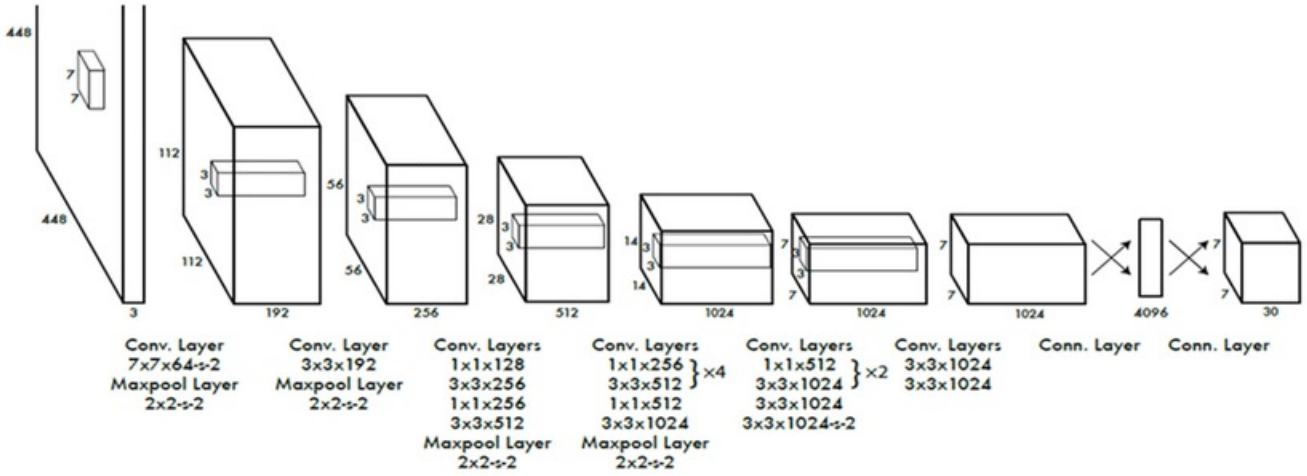
Anchors: Why They Are Effective?

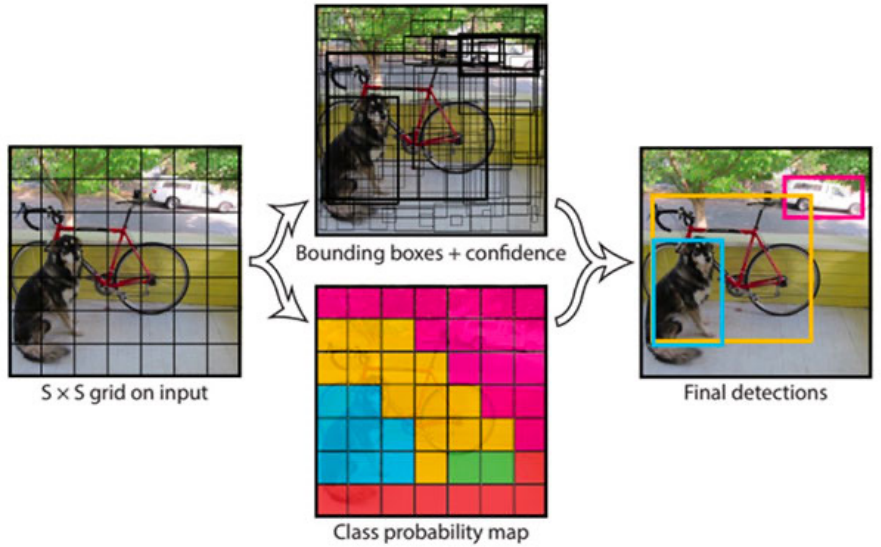
- With the presence of anchor boxes, the network only need to predict offsets or corrections to the anchors.
- The detectors don't have to work very hard to make pretty good predictions already, because predicting all zeros simply outputs the anchor box, which will be reasonably close to the true object (on average).
- Without the anchors, each detector would have to learn from scratch what the different bounding box shapes look like.



Example : YOLO (You Only Look Once)

- YOLO divides up the image into a grid of **13 by 13** cells: Each of these cells is responsible for predicting **5 bounding boxes**. A bounding box describes the rectangle that encloses an object.
- YOLO also outputs a confidence score that tells us how certain it is that the predicted bounding box actually encloses an object.





Example : YOLO (You Only Look Once)

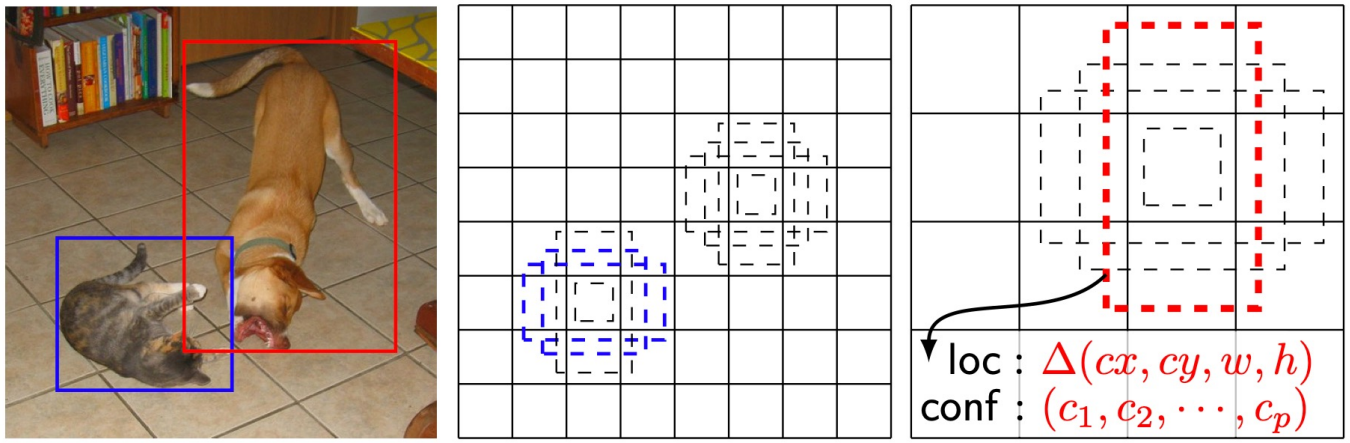
- It looks at the whole image at test time, so its predictions are informed by global context in the image.

- Since it makes predictions with a single network evaluation, unlike systems such as R-CNN which require thousands for a single image. This makes it extremely fast, more than 1000x faster than R-CNN and 100x faster than Fast R-CNN

- However, one limitation for YOLO is that it only predicts 1 type of class in one grid hence, it struggles with very small objects.

Example : SSD (Single shot Multi box detector)

- The basic essence in both YOLO of SSD is the same. However, there are few differences in details.
- The coordinate predictions in SSD *can* go outside their grid cells and therefore, in theory, a box in the bottom-right corner of the model could predict a bounding box with its center all the way over in the top-left corner of the image (but this probably won't happen in practice).



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

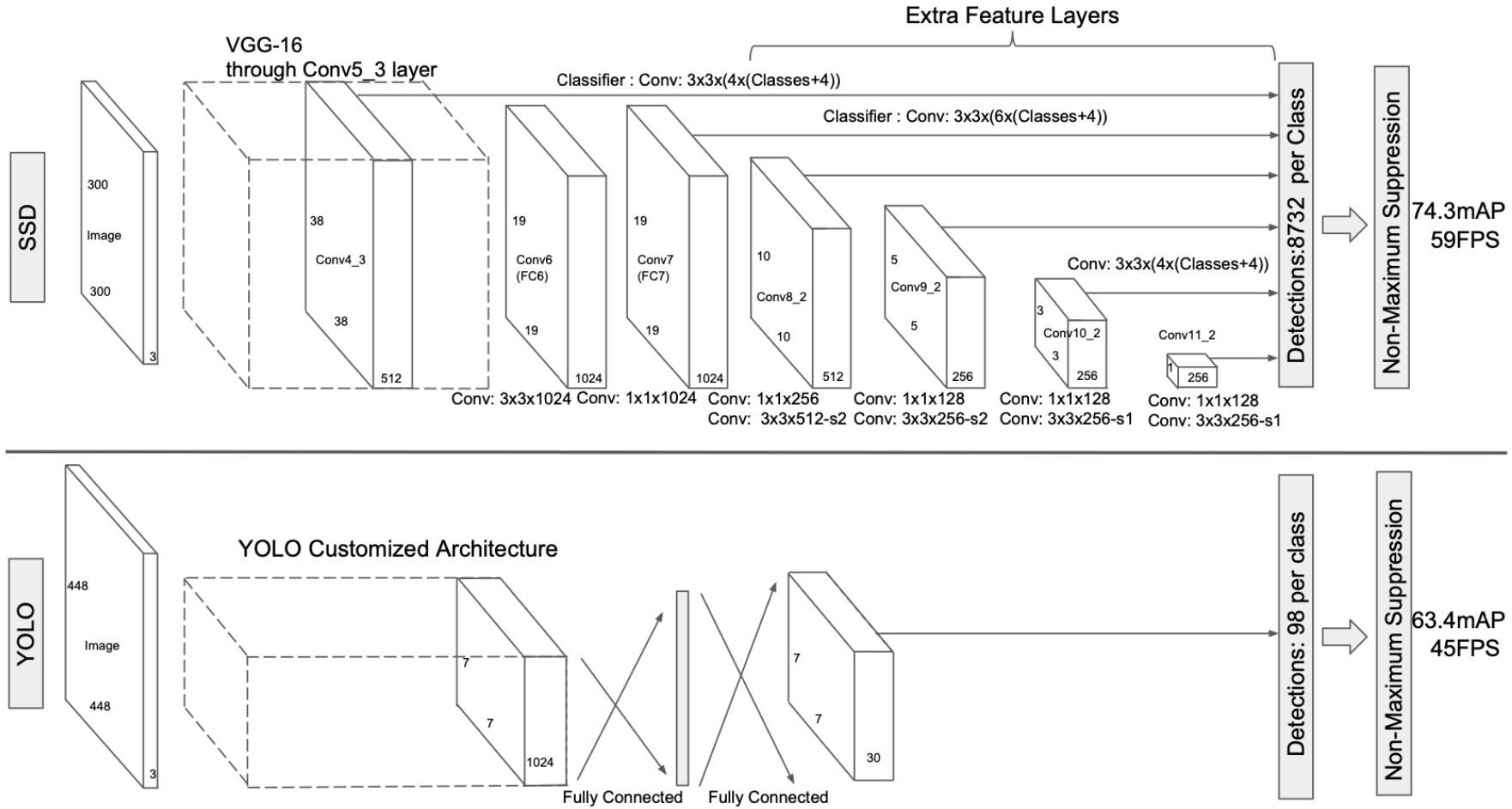
Example : SSD (Single shot Multi box detector)

- Unlike YOLO, the selection of default priors or anchor boxes is independent of the dataset being used.

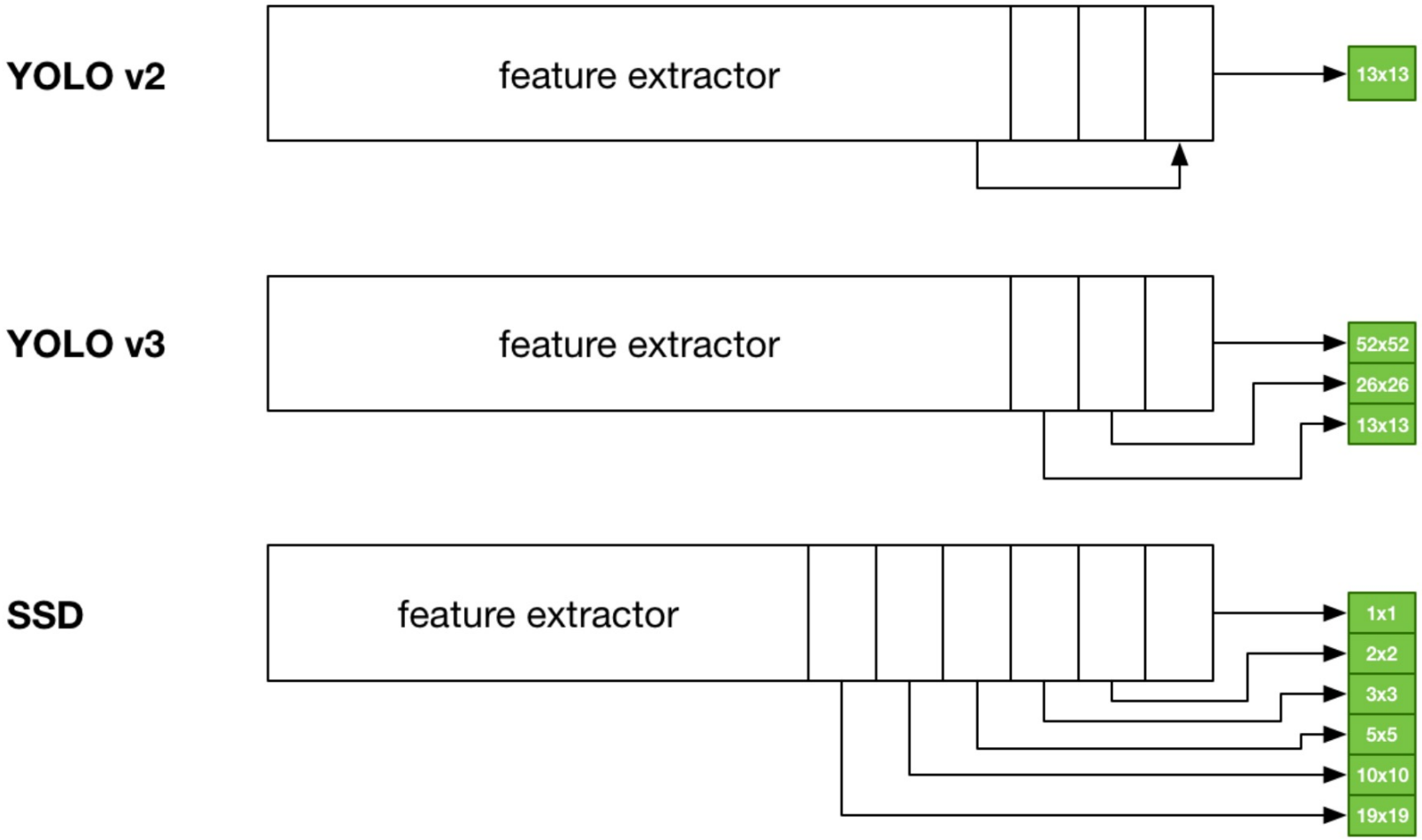
- And instead of having a confidence score indicating whether or not there is an object inside the box, SSD adds one more class label named “background” for denying the presence of object.

- SSD also uses different grid sizes at the output and unlike YOLO, it does not use the anchors to make the detectors specialize on object size, it uses the different grids for that.

YOLO vs SSD (Architecture)



YOLO vs SSD (Architecture)



SSD using different grid sizes for making the detectors specialize on object size

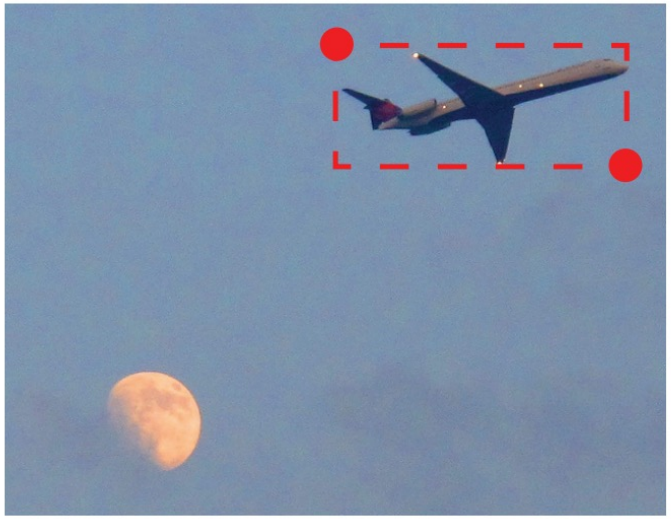
Anchor-Free Methods

- With anchors, you have large number of **hyperparameters** determining their **shape and sizes** and a lot of **manual design choices.** In practice, when building a detector on a novel dataset, you need to carefully tune those many hyperparameters because they influence the final performance significantly

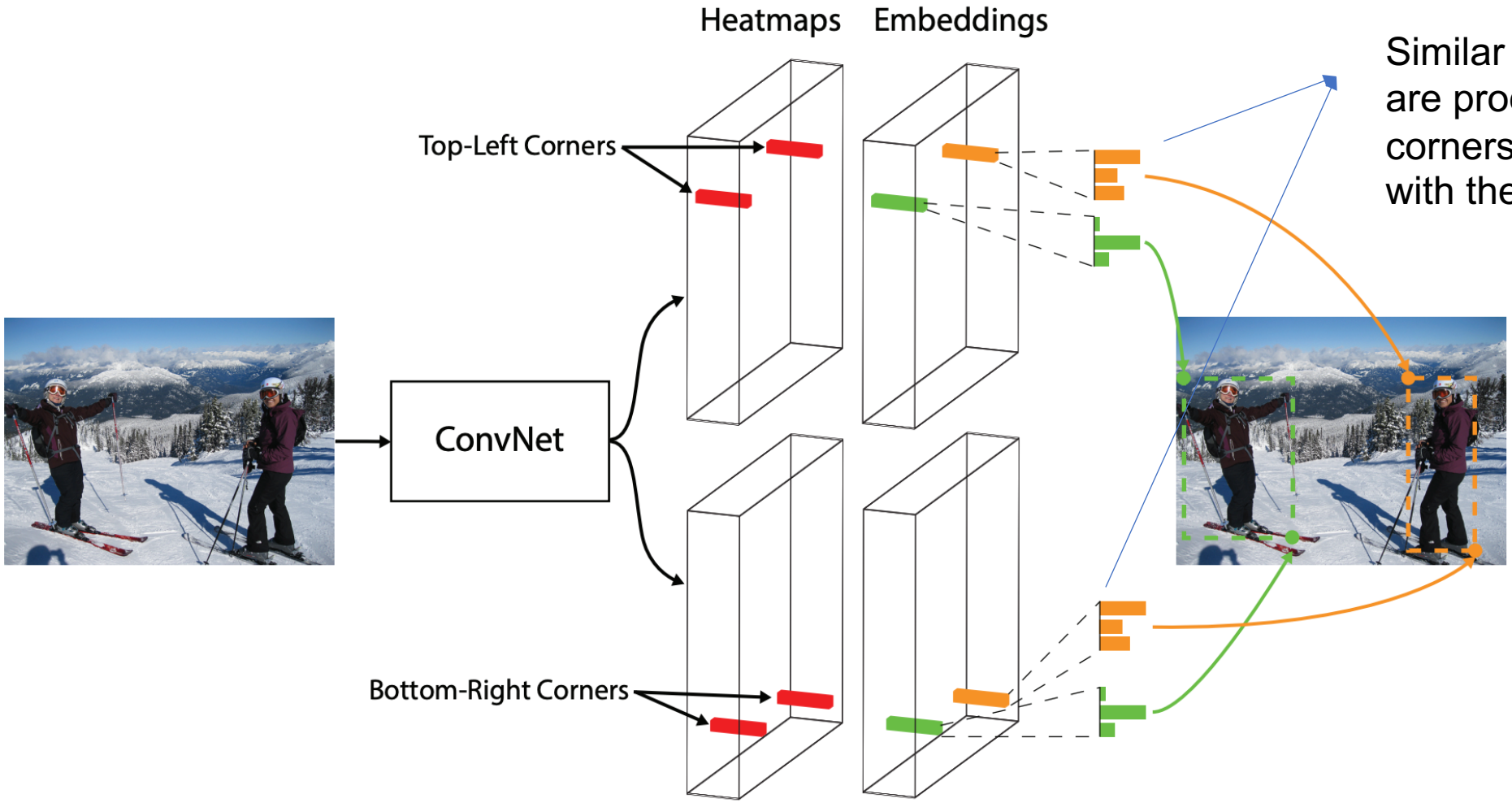
- Furthermore during training, the anchor boxes need to be compared with the ground truth which is incredibly complicated to implement. So without them you make the implementation much smaller, simpler to implement and faster.

CornerNet : Detecting Objects as Paired Keypoints

- A new one-stage approach to object detection that does away with anchor boxes.
- Detect an object as a pair of keypoints— the top-left corner and bottom-right corner of the bounding box.
- The network produces embeddings for top-left corners and bottom-right corners for each class category. These embeddings are used to match the two corners.



CornerNet : Detecting Objects as Paired Keypoints



CenterNet: Keypoint Triplets for Object Detection

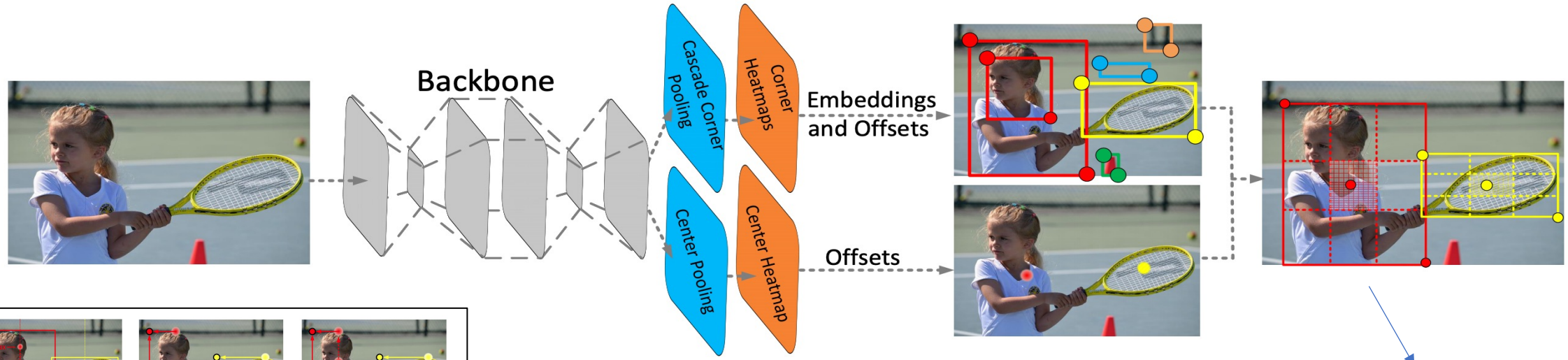
- The performance of CornerNet is restricted by its relatively **weak ability of referring to the global information** of an object since it only learns to detect corners.

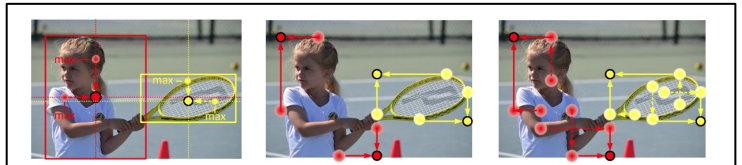
- To address this issue, CornerNet is equipped with an ability of perceiving the visual patterns within each proposed region, so that it can identify the correctness of each bounding box by itself.

- CenterNet **explores the central part of a proposal**, i.e., the region that is close to the geometric center, with one extra keypoint.

CenterNet: Keypoint Triplets for Object Detection

Corner Pooling of CornerNet produces 2 heatmaps and embeddings for the corners





(a) (b) (c)

Figure 4: (a) Center pooling takes the maximum values in both horizontal and vertical directions. (b) Corner pooling only takes the maximum values in boundary directions. (c) Cascade corner pooling takes the maximum values in both boundary directions and internal directions of objects.

Center pooling produces heatmaps for detected center keypoints

The detected center keypoints are used to determine final bounding boxes

Summary – Deep Learning based Object Detection

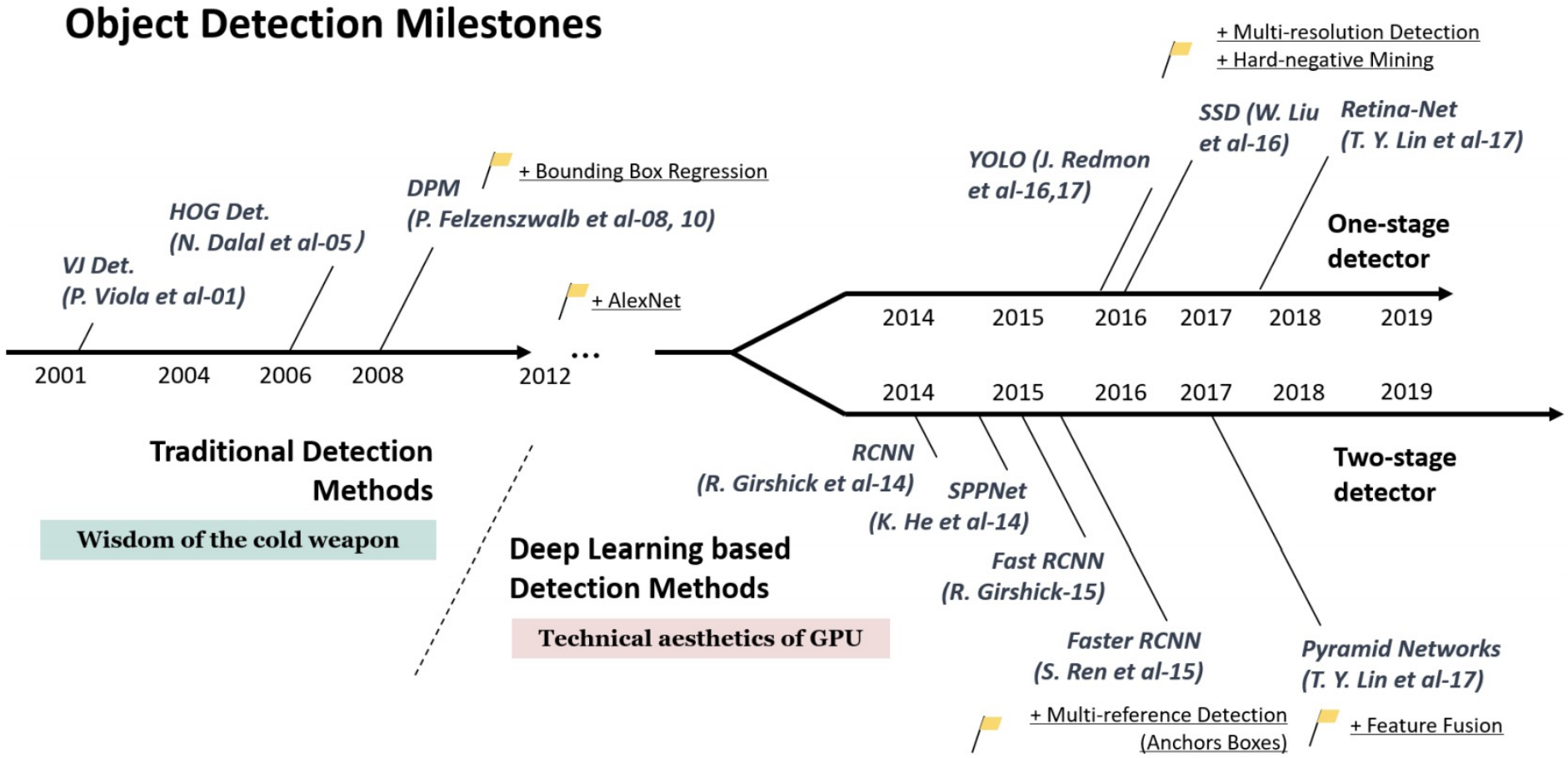


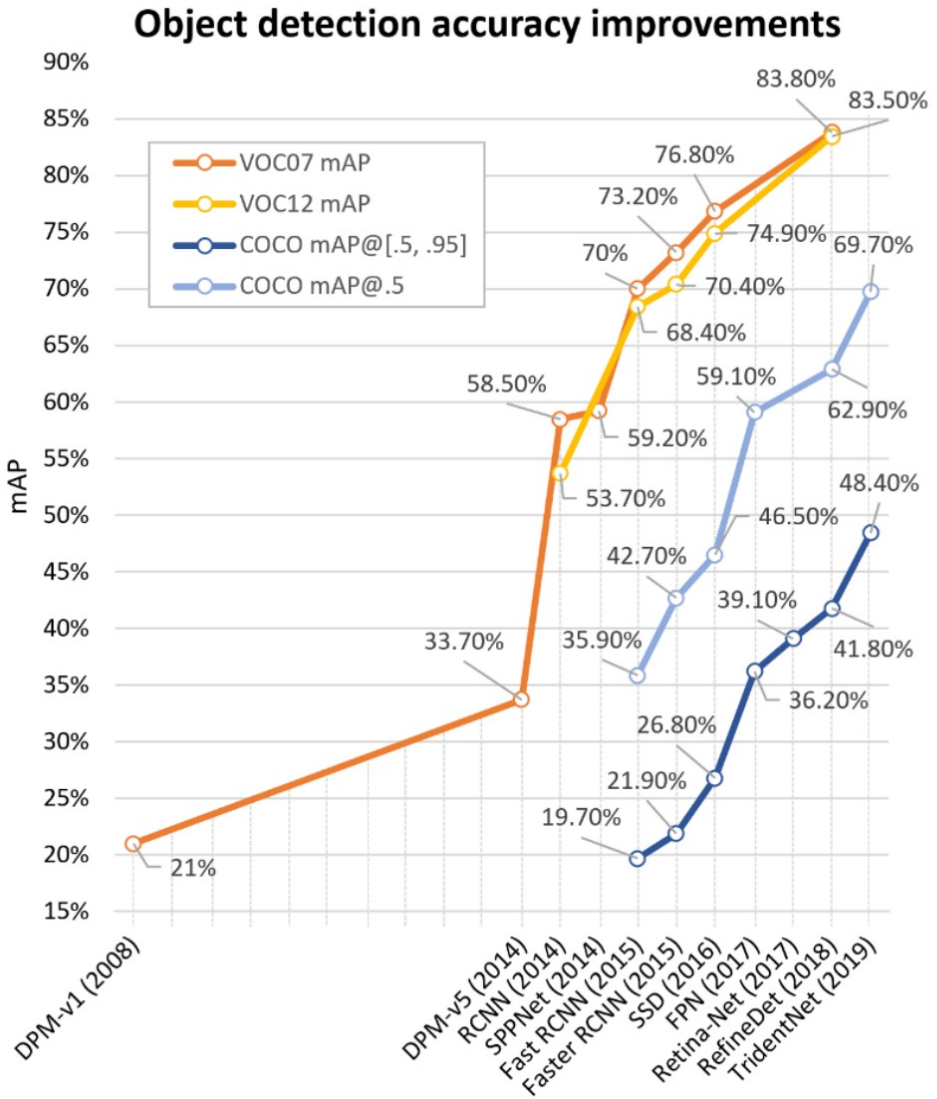
Fig. 2. A road map of object detection. Milestone detectors in this figure: VJ Det. [10, 11], HOG Det. [12], DPM [13–15], RCNN [16], SPPNet [17], Fast RCNN [18], Faster RCNN [19], YOLO [20], SSD [21], Pyramid Networks [22], Retina-Net [23].

https://arxiv.org/pdf/1905.05055.pdf

Summary – Deep Learning based Object Detection

- RetinaNet

In despite of its high speed and simplicity, the one-stage detectors have trailed the accuracy of two-stage detectors for years. T.-Y. Lin *et al.* have discovered the reasons behind and proposed RetinaNet in 2017 [23]. They claimed that the extreme foreground-background class imbalance encountered during training of dense detectors is the central cause. To this end, a new loss function named "focal loss" has been introduced in RetinaNet by reshaping the standard cross entropy loss so that detector will put more focus on hard, misclassified examples during training. Focal Loss enables the one-stage detectors to achieve comparable accuracy of two-stage detectors while maintaining very high detection speed. (COCO mAP@.5=59.1%, mAP@[.5, .95]=39.1%).

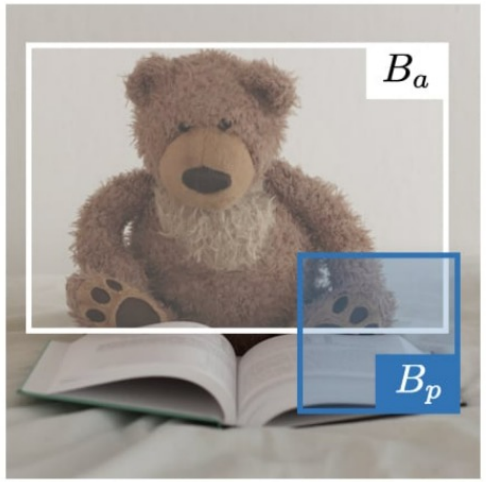


https://arxiv.org/pdf/1905.05055.pdf

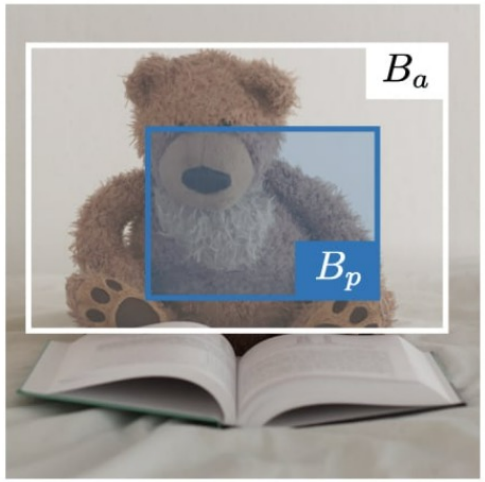
Evaluation Metrics for Object Detection

- Average Precision (AP)
  - Area Under the PR Curve
- mean Average Precision (mAP)

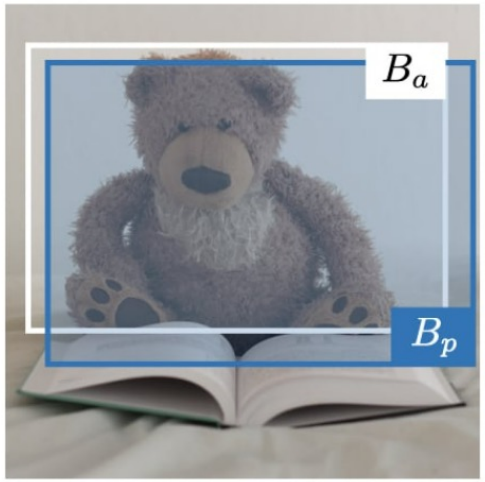
- Depends on Intersection-over-Union (IoU)



$\text{IoU}(B_p, B_a) = 0.1$



$\text{IoU}(B_p, B_a) = 0.5$



$\text{IoU}(B_p, B_a) = 0.9$

https://towardsdatascience.com/on-object-detection-metrics-with-worked-example-216f173ed31e

Next week

- Optical flow & Tracking
  - + Basics definitions
  - ++ KLT
  - ++ Mean-shift
  - + Correlation filter
- CNNs for Tracking
  - + Supervised
  - + Self-supervised

References for next week

- Sz: ch 7.1.5, 9.4

- Baker, Simon, and Iain Matthews. "Lucas-kanade 20 years on: A unifying framework." *International journal of computer vision*56.3 (2004): 221-255.
- Comaniciu, Dorin, and Peter Meer. "Mean shift: A robust approach toward feature space analysis." *IEEE Transactions on Pattern Analysis & Machine Intelligence* 5 (2002): 603-619.
- Feichtenhofer, Christoph, Axel Pinz, and Andrew Zisserman. "Detect to track and track to detect." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3038-3046. 2017.
- Wang, Xiaolong, Allan Jabri, and Alexei A. Efros. "Learning correspondence from the cycle-consistency of time." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2566-2576. 2019.