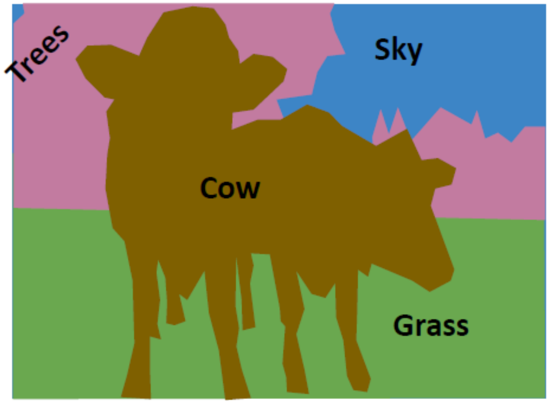
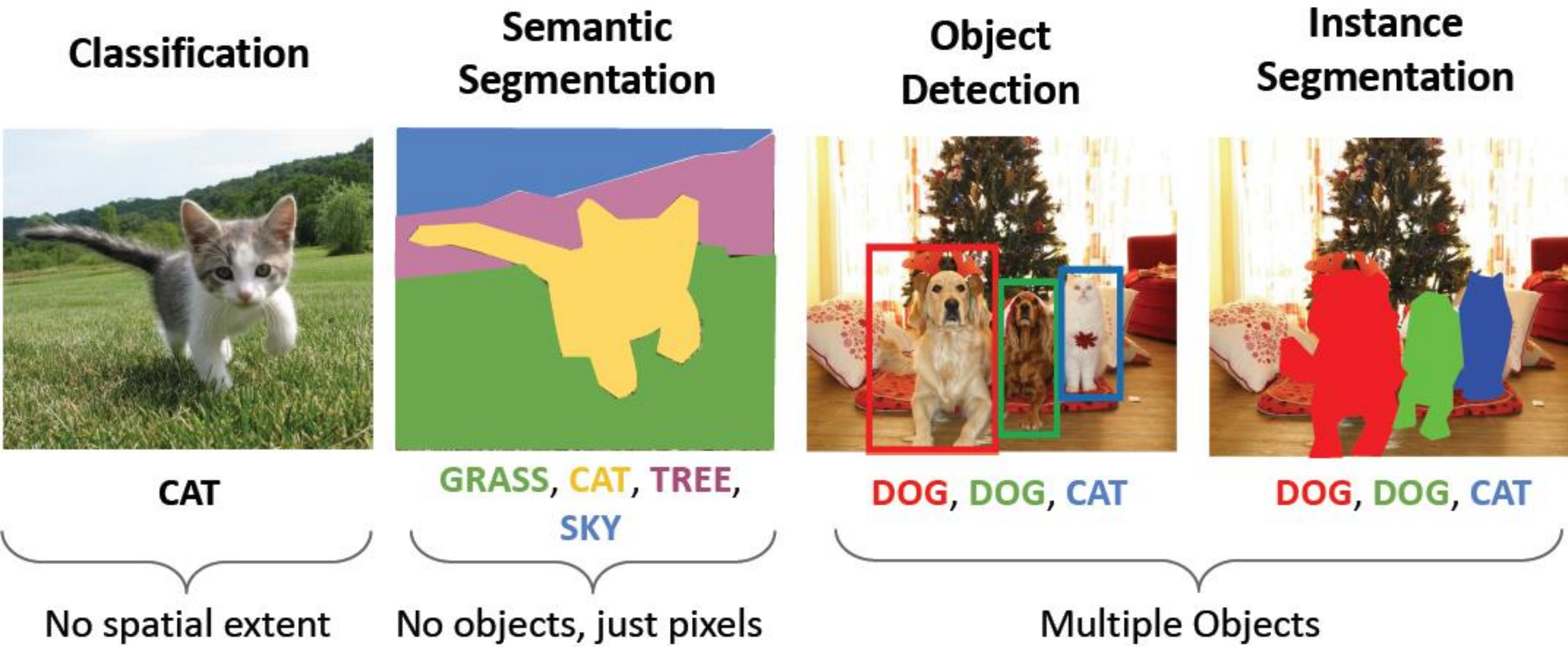
L4 Object Detection and Segmentation

Zonghua Gu 2022



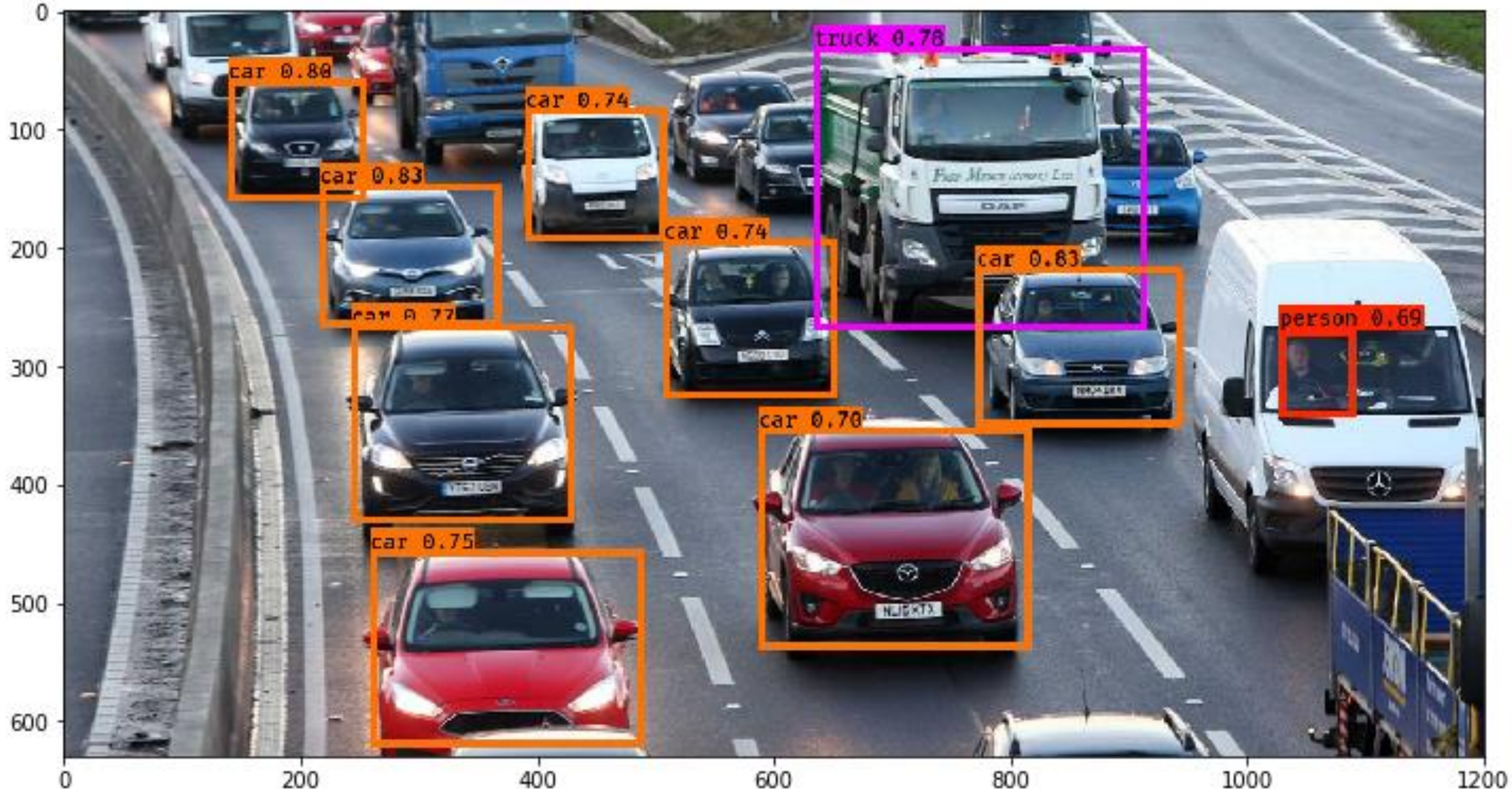
Acknowledgement: Based EECS 498 Deep Learning for Computer Vision, U Michigan

Computer Vision Tasks



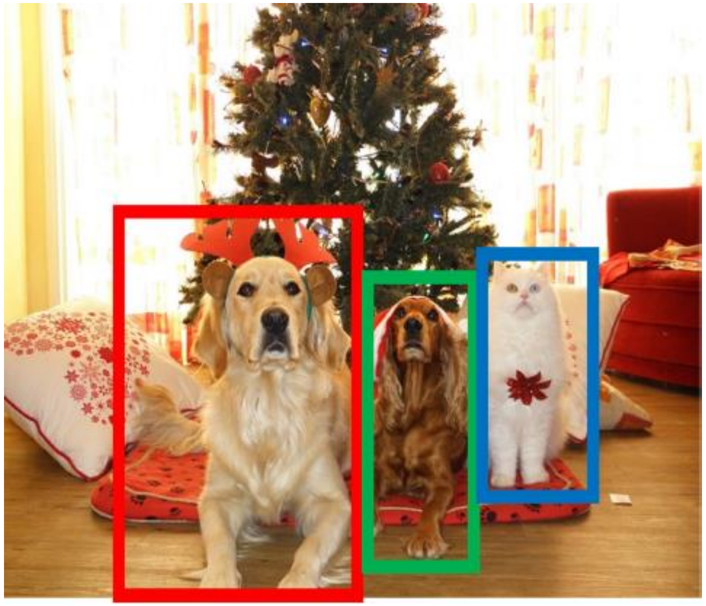
Outline

- Object detection
- Segmentation



Object Detection: Task Definition

- Input: Single Image
- Output: a set of detected objects
- For each object predict:
  - WHAT: Class label (e.g., cat vs. dog)
  - WHERE: Bbox (4 numbers: x, y, width, height)
- Challenges:
  - Multiple outputs: variable numbers of objects per image
  - Multiple types of output: predict ”what” (class label) as well as “where” (Bbox)
  - Large images: Classification works at 224x224 or lower; need higher resolution for detection, often ~800x600

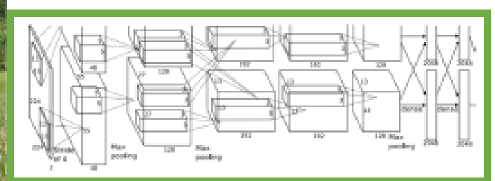


Single-Object Detection

Detecting a single object

This image is CC0 public domain

Often pretrained on ImageNet (Transfer learning)



Vector: 4096

Fully Connected: 4096 to 1000

"What"

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Correct label: Cat

Softmax Loss

Multitask Loss

Weighted Sum

Loss

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

L2 Loss

Correct box: (x', y', w', h')

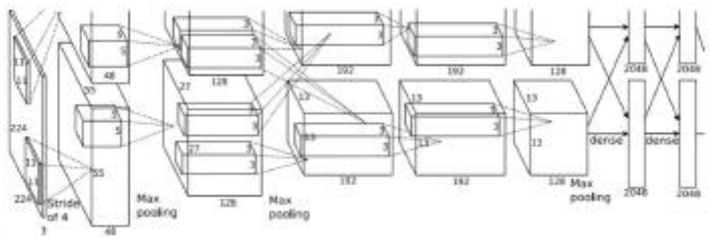
"Where"

Treat localization as a regression problem!

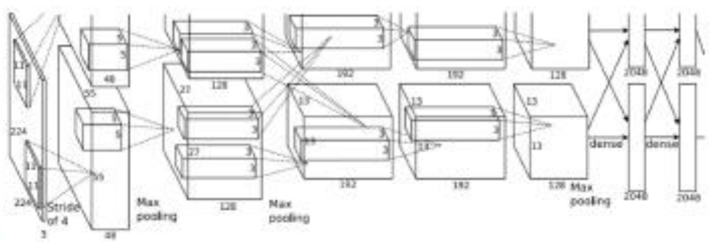
Problem: Images can have more than one object!

Multi-Object Detection

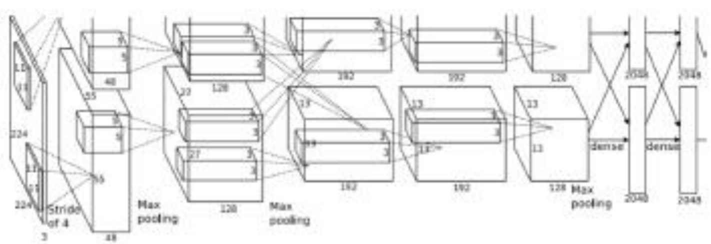
- Needs to predict 4 numbers for each object Bbox (x, y, w, h)
  - (x, y) are coordinates of the box center; (w, h) are its width and height
- 4N numbers for N objects

CAT: (x, y, w, h)

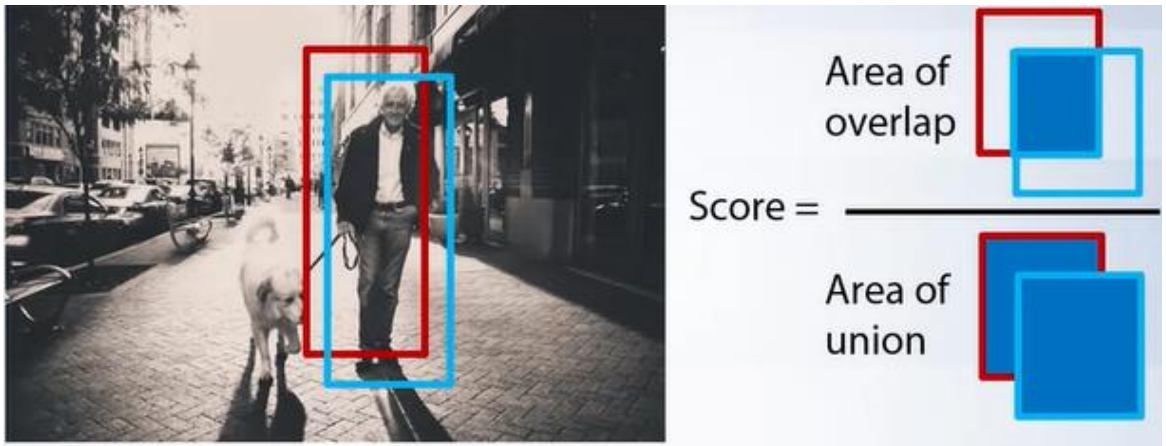



DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)

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

Detection Criteria (Intersection Over Union, IOU)

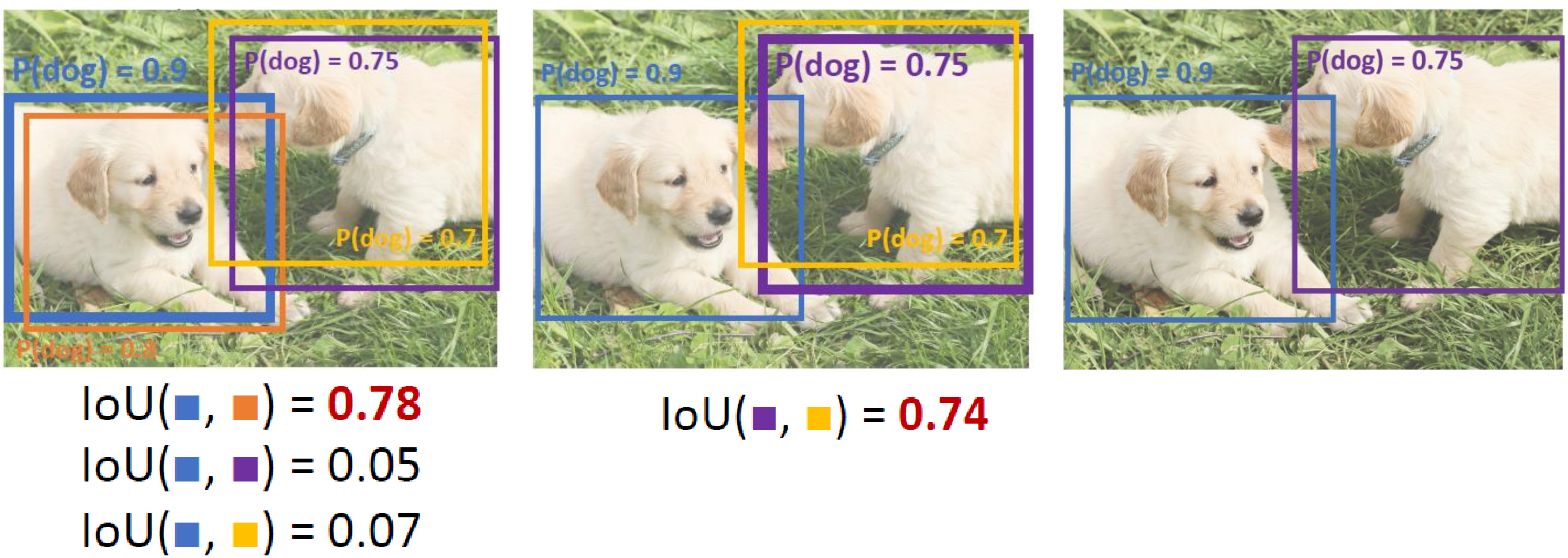


- Blue box: Ground Truth; Red box: model output
- Set a threshold for detection (positive result)

  $\mathrm{IOU}(\mathrm{B}_{\mathrm{GT}}, \mathrm{B}_{\mathrm{Pred}}) \geq \theta_{IoU}$
  - Common threshold $\theta_{IoU} = 0.5$

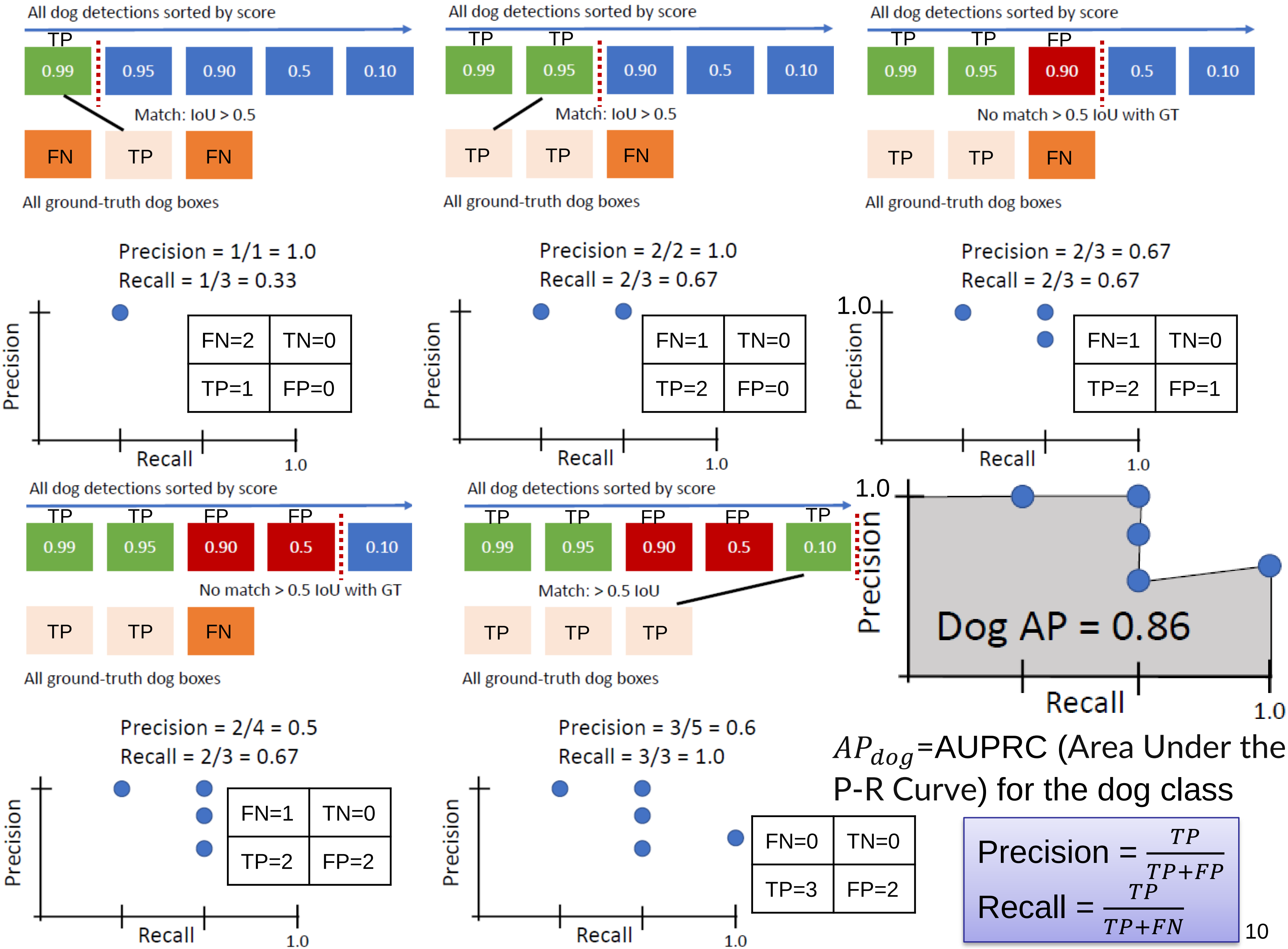
Non-Max Suppression (NMS)

- Problem: Object detectors often output many overlapping detections
- NMS: Discard (suppresses) overlapping object boxes except the one with the maximum classification score
- For each output class
  - 1 Select next highest-scoring box b and output it as a prediction
  - 1 Discard any remaining boxes b' with IoU(b, b') > threshold
  - 1. If any boxes remain, GOTO 1
- Example:
  - Assume threshold=.7
  - Blue box has the highest classification score $P(dog) = .9$. Output the blue box, and discard the orange box since IoU(blue, orange)=.78>.7.
  - The next highest-scoring box is the purple box with $P(dog) = .75$. Output the purple box, and discard the yellow box since IoU(purple, yellow)=.74>.7



Evaluating Object Detectors: Mean Average Precision (mAP)

- 1. Run object detector on all test images (with NMS)
- 2. For each class, compute Average Precision (AP)
  - 1. For each detection (highest score to lowest score)
    - 1. If it matches some GT box with IoU > thresh, mark it as positive and eliminate the GT
    - 2. Otherwise mark it as negative
    - 3. Plot a point on PR Curve
  - 2. Average Precision (AP) for each class, e.g., AP_{dog}=AUPRC (Area Under PR Curve) for the dog class
- 3. mean Average Precision (mAP) = average of APs for each class
- 4. For “COCO mAP”: compute mAP@thresh for each IoU threshold and take average
- FYI: Object Detection Performance Metrics
  - https://www.coursera.org/learn/computer-vision-with-embedded-machine-learning/lecture/zDIgp/object-detection-performance-metrics



AP_{dog}=AUPRC (Area Under the P-R Curve) for the dog class

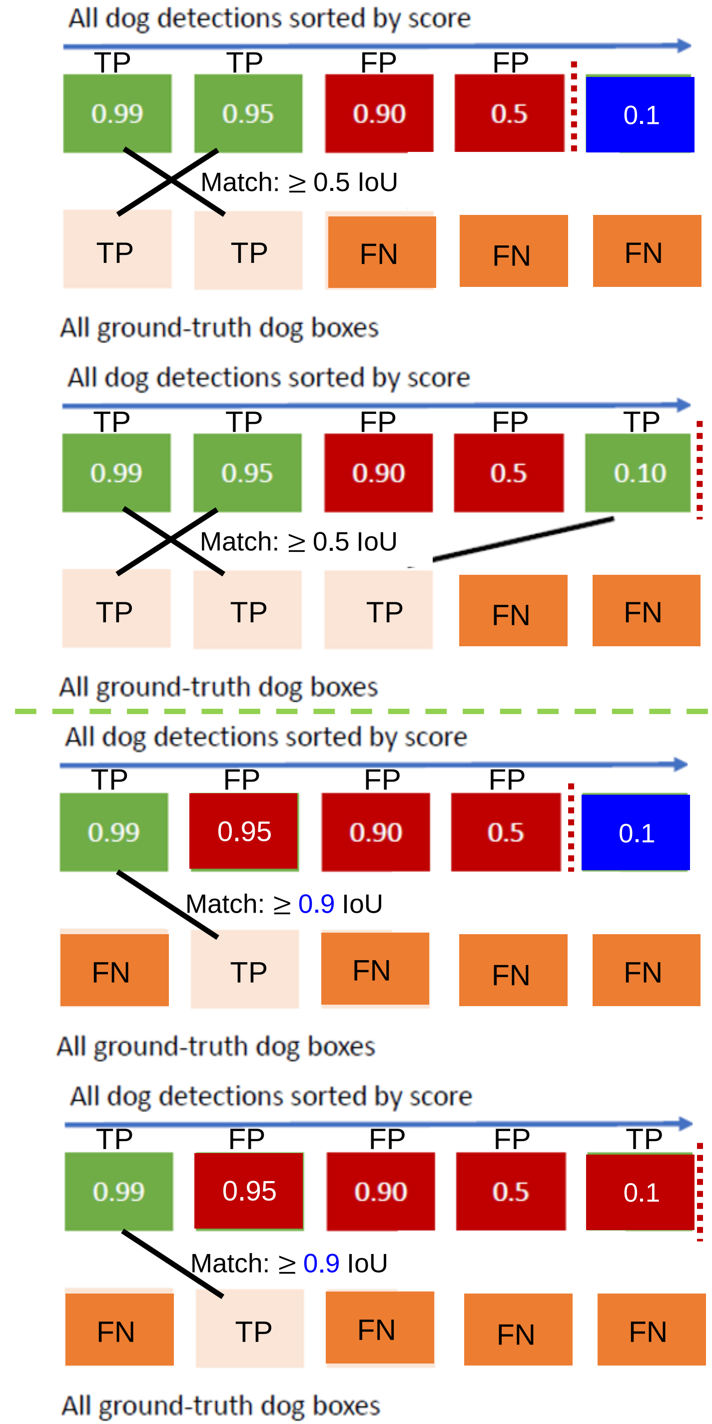
$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{Recall} = \frac{TP}{TP+FN}$$

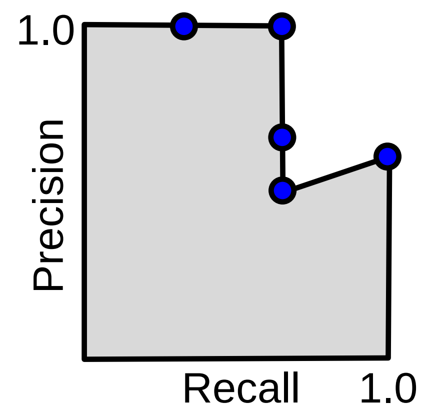
IoU Threshold and Conf. Score Threshold

- Suppose an image contains 5 dogs in it, i.e., 5 GT boxes with label Dog (instead of 3 in previous slide).
- With IoU threshold .5, 3 GT boxes are detected
  - With conf. score threshold of .5, 2 out of 5 GT boxes are detected with Precision= $\frac{TP}{TP+FP} = \frac{2}{2+2} = .5$; Recall= $\frac{TP}{TP+FN} = \frac{2}{2+3} = .4$
  - With score threshold of .1, 3 out of 5 GT boxes are detected with Precision= $\frac{3}{3+2} = .6$; Recall= $\frac{3}{3+2} = .6$ (reduced score threshold ⇒ increased recall)
- With IoU threshold .9, only 1 GT box is detected
  - With conf. score threshold of either .5 or .1, only 1 out of 5 GT boxes are detected with Precision= $\frac{1}{1+4} = .2$; Recall= $\frac{1}{1+4} = .2$
- Conf. score threshold determines decision boundary between positive (Dog) and negative (non-Dog) items among all predicted boxes (green and red)
  - Conf. score threshold ↓ ⇒ TP ↑, FN ↓, FP ↑ (same as classification)
  - Minimum number of FNs is equal to number of unmatched GT boxes, i.e., some FNs may remain even with lowest conf. score threshold (unlike classification, where FNs can always be reduced to 0 by reducing conf. score threshold)
- IoU threshold determines number of matched predicted boxes
  - IoU threshold ↓ ⇒ matched predicted boxes ↑ (more green, fewer red)

FN	TN
TP	FP



- Suppose an image contains 3 dogs in it, i.e., 3 GT boxes with label Dog ($TP + FN = 3$). Prediction confidence scores of Dog class for 5 predicted boxes: (.99, .95, .9, .5, .1)
- Green indicates positive items (matching a GT box w. IoU$\geq .5$)
- Red indicates negative items (no matching a GT box w. IoU$\geq .5$)
- With score threshold of .1, 3 out of 3 GT boxes are detected with Recall= 1.0



FN=2	TN=0
TP=1	FP=0

$P = 1.0, R = .33$

FN=1	TN=0
TP=2	FP=0

$P = 1.0, R = .67$

FN=1	TN=0
TP=2	FP=1

$P = .67, R = .67$

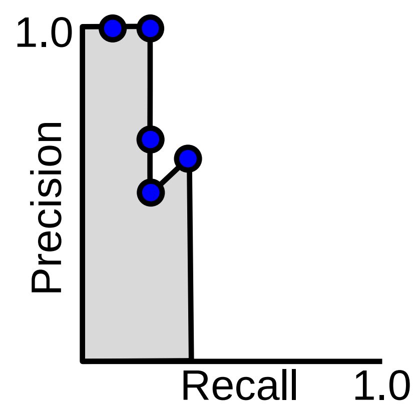
FN=1	TN=0
TP=2	FP=2

$P = .5, R = .67$

FN=0	TN=0
TP=3	FP=2

$P = .6, R = 1.0$

- **Less accurate box regression**: Suppose an image contains 10 dogs in it, i.e., 10 GT boxes with label Dog ($TP + FN = 10$). Prediction confidence scores of Dog class for 5 predicted boxes: (.99, .95, .9, .5, .1)
- Green indicates positive items (matching a GT box w. IoU$\geq .5$)
- Red indicates negative items (no matching a GT box w. IoU$\geq .5$)
- Precision = $\frac{TP}{TP+FP}$ is not affected; Recall = $\frac{TP}{TP+FN}$ is proportionally reduced due to larger FN (undetected GT boxes)
- With score threshold of .1, 3 out of 10 GT boxes are detected with Recall= $.3$



FN=9	TN=0
TP=1	FP=0

$P = 1.0, R = .1$

FN=8	TN=0
TP=2	FP=0

$P = 1.0, R = .2$

FN=8	TN=0
TP=2	FP=1

$P = .67, R = .2$

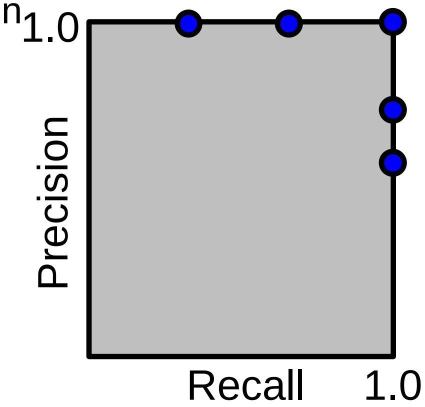
FN=8	TN=0
TP=2	FP=2

$P = .5, R = .2$

FN=7	TN=0
TP=3	FP=2

$P = .6, R = .3$

- **More accurate conf. score prediction**: Suppose an image contains 3 dogs in it, i.e., 3 GT boxes with label Dog. Prediction confidence scores of Dog class for 5 predicted boxes: (.99, .95, .93, .9, .5)
- Green indicates positive items (matching a GT box w. IoU$\geq .5$)
- Red indicates negative items (no matching a GT box w. IoU$\geq .5$)
- Precision = $\frac{TP}{TP+FP}$ stays at 1.0 (no FP) until $P = 1.0, R = 1.0$, with perfect AUPRC=1.0; Precision drops after reaching Recall=1.0, but this does not affect AUPRC



- With score threshold of .5, 3 out of 3 GT boxes are detected with Recall= 1.0

FN=2	TN=0
TP=1	FP=0

$P = 1.0, R = .33$

FN=1	TN=0
TP=2	FP=0

$P = 1.0, R = .67$

FN=0	TN=0
TP=3	FP=0

$P = 1.0, R = 1.0$

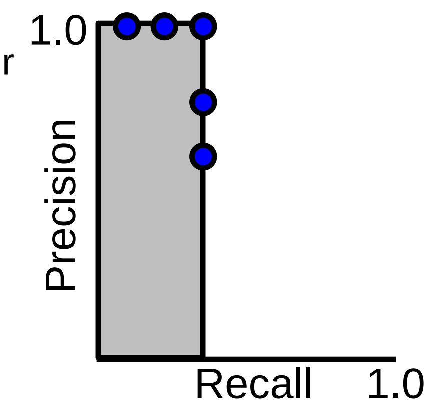
FN=0	TN=0
TP=3	FP=1

$P = .75, R = 1.0$

FN=0	TN=0
TP=3	FP=2

$P = .6, R = 1.0$

- **Less accurate box regression:** Suppose an image contains 10 dogs in it, i.e., 10 GT boxes with label Dog. Prediction confidence scores of Dog class for 5 predicted boxes: (.99, .95, .93, .9, .5)
- Green indicates positive items (matching a GT box w. IoU$\geq .5$)
- Red indicates negative items (no matching a GT box w. IoU$\geq .5$)
- Precision = $\frac{TP}{TP+FP}$ is not affected; Recall = $\frac{TP}{TP+FN}$ is proportionally reduced due to larger FN (undetected GT boxes)
- With score threshold of .5, 3 out of 10 GT boxes are detected with Recall= $.3$



FN=9	TN=0
TP=1	FP=0

$P = 1.0, R = .1$

FN=8	TN=0
TP=2	FP=0

$P = 1.0, R = .2$

FN=7	TN=0
TP=3	FP=0

$P = 1.0, R = .3$

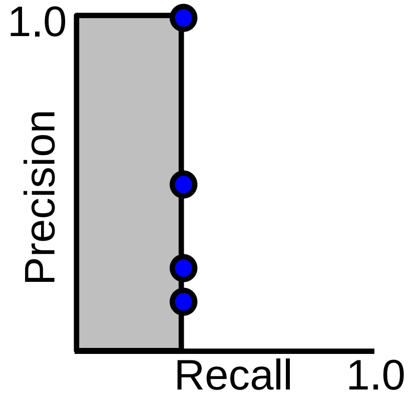
FN=7	TN=0
TP=3	FP=1

$P = .75, R = .3$

FN=7	TN=0
TP=3	FP=2

$P = .6, R = .3$

- **Higher IoU threshold for matching with GT box:** Suppose an image contains 3 dogs in it, i.e., 3 GT boxes with label Dog. Prediction confidence scores of Dog class for 5 predicted boxes (.99, .95, .9, .5, .1)
- Green indicates positive items (matching a GT box w. IoU≥ .9)
- Red indicates negative items (no matching a GT box w. IoU≥ .9)
- Precision = $\frac{TP}{TP+FP}$ stays at 1.0 (no FP) until $P = 1.0, R = 1.0$, with perfect AUPRC=1.0; Precision drops after reaching Recall=1.0, but this does not affect AUPRC
- With score threshold of .99, 1 out of 3 GT boxes are detected with Recall= .33



FN=2	TN=0
TP=1	FP=0

$P = 1.0, R = .33$

FN=2	TN=0
TP=1	FP=1

$P = .5, R = .33$

FN=2	TN=0
TP=1	FP=2

$P = .33, R = .33$

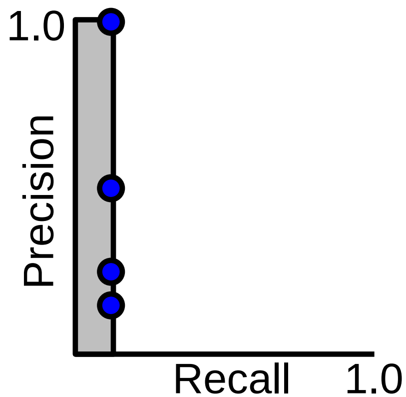
FN=2	TN=0
TP=1	FP=3

$P = .25, R = .33$

FN=2	TN=0
TP=1	FP=4

$P = .2, R = .33$

- **Less accurate box regression:** Suppose an image contains 10 dogs in it, i.e., 10 GT boxes with label Dog. Prediction confidence scores of Dog class for 5 predicted boxes: (.99, .95, .9, .5, .1)
- Green indicates positive items (matching a GT box w. IoU≥ .9)
- Red indicates negative items (no matching a GT box w. IoU≥ .9)
- Precision = $\frac{TP}{TP+FP}$ is not affected; Recall = $\frac{TP}{TP+FN}$ is proportionally reduced due to larger FN (undetected GT boxes)
- With score threshold of .99, 1 out of 10 GT boxes are detected with Recall= .1



FN=9	TN=0
TP=1	FP=0

$P = 1.0, R = .1$

FN=9	TN=0
TP=1	FP=1

$P = .5, R = .1$

FN=9	TN=0
TP=1	FP=2

$P = .33, R = .1$

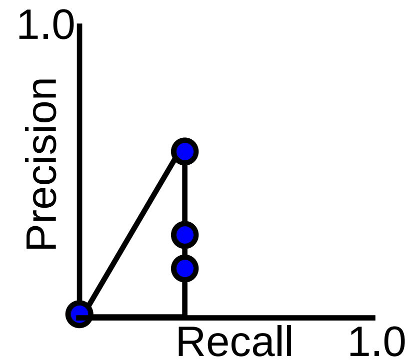
FN=9	TN=0
TP=1	FP=3

$P = .25, R = .1$

FN=9	TN=0
TP=1	FP=4

$P = .2, R = .1$

- **Less accurate conf. score prediction:** Suppose an image contains 3 dogs in it, i.e., 3 GT boxes with label Dog. Prediction confidence scores of Dog class for 5 predicted boxes (.99, .95, .9, .5, .1)
- Green indicates positive items (matching a GT box w. IoU≥ .9)
- Red indicates negative items (no matching a GT box w. IoU≥ .9)
- Precision = $\frac{TP}{TP+FP}$ stays at 1.0 (no FP) until $P = 1.0, R = 1.0$, with perfect AUPRC=1.0; Precision drops after reaching Recall=1.0, but this does not affect AUPRC
- With score threshold of .99, 1 out of 3 GT boxes are detected with Recall= .33



FN=3	TN=0
TP=0	FP=1

$P = 0, R = 0$

→

FN=2	TN=0
TP=1	FP=1

$P = .5, R = .33$

→

FN=2	TN=0
TP=1	FP=2

$P = .33, R = .33$

→

FN=2	TN=0
TP=1	FP=3

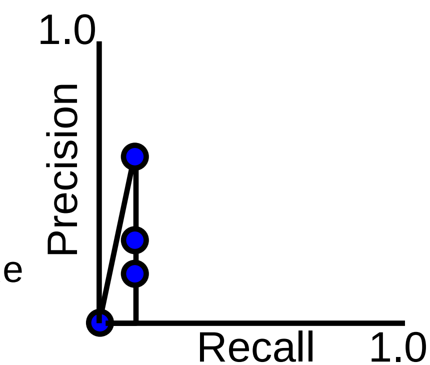
$P = .25, R = .33$

→

FN=2	TN=0
TP=1	FP=4

$P = .2, R = .33$

- **Less accurate box regression:** Suppose an image contains 10 dogs in it, i.e., 10 GT boxes with label Dog. Prediction confidence scores of Dog class for 5 predicted boxes: (.99, .95, .9, .5, .1)
- Green indicates positive items (matching a GT box w. IoU≥ .9)
- Red indicates negative items (no matching a GT box w. IoU≥ .9)
- Precision = $\frac{TP}{TP+FP}$ is not affected; Recall = $\frac{TP}{TP+FN}$ is proportionally reduced due to larger FN (undetected GT boxes)
- With score threshold of .99, 1 out of 10 GT boxes are detected with Recall= .1



FN=10	TN=0
TP=0	FP=0

$P = 0, R = 0$

→

FN=9	TN=0
TP=1	FP=1

$P = .5, R = .1$

→

FN=9	TN=0
TP=1	FP=2

$P = .33, R = .1$

→

FN=9	TN=0
TP=1	FP=3

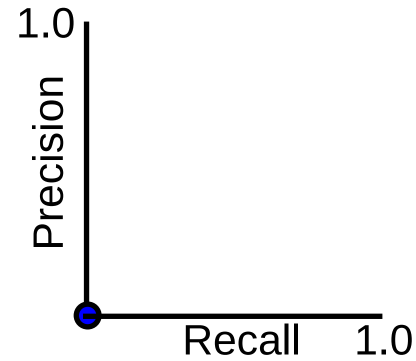
$P = .25, R = .1$

→

FN=9	TN=0
TP=1	FP=4

$P = .2, R = .1$

- **Very high IoU threshold for matching with GT box:** Suppose an image contains 3 dogs in it, i.e., 3 GT boxes with label Dog. Prediction confidence scores of Dog class for 5 predicted boxes (.99, .95, .9, .5, .1)
- Green indicates positive items (matching a GT box w. IoU≥ .99)
- Red indicates negative items (no matching a GT box w. IoU≥ .99)
- Precision = $\frac{TP}{TP+FP}$ and Recall = $\frac{TP}{TP+FN}$ both stay at 0, since TP= 0



FN=3	TN=0
TP=0	FP=1

$P = 0, R = 0$

FN=3	TN=0
TP=0	FP=2

$P = 0, R = 0$

FN=3	TN=0
TP=0	FP=3

$P = 0, R = 0$

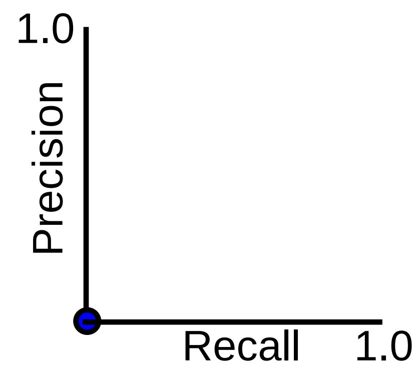
FN=3	TN=0
TP=0	FP=4

$P = 0, R = 0$

FN=3	TN=0
TP=0	FP=5

$P = 0, R = 0$

- **Very high standard for matching with GT box:** Suppose an image contains 10 dogs in it, i.e., 10 GT boxes with label Dog. Prediction confidence scores of Dog class for 5 predicted boxes (.99, .95, .9, .5, .1)
- Green indicates positive items (matching a GT box w. IoU≥ .99)
- Red indicates negative items (no matching a GT box w. IoU≥ .99)
- Precision = $\frac{TP}{TP+FP}$ and Recall = $\frac{TP}{TP+FN}$ both stay at 0, since TP= 0



FN=10	TN=0
TP=0	FP=0

$P = 0, R = 0$

FN=10	TN=0
TP=0	FP=2

$P = 0, R = 0$

FN=10	TN=0
TP=0	FP=2

$P = 0, R = 0$

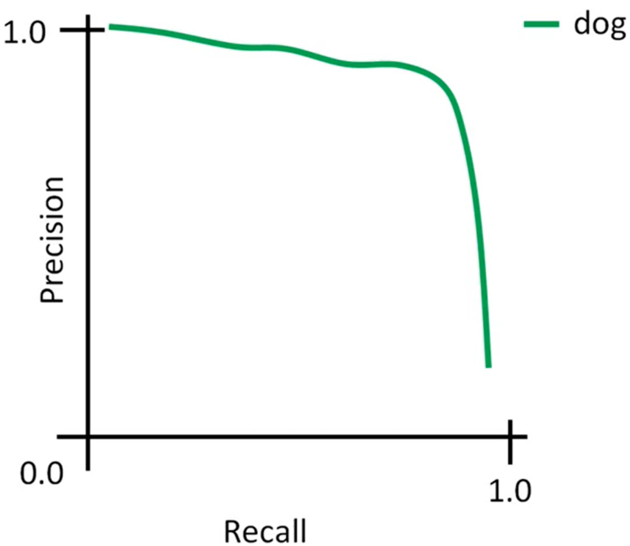
FN=10	TN=0
TP=0	FP=2

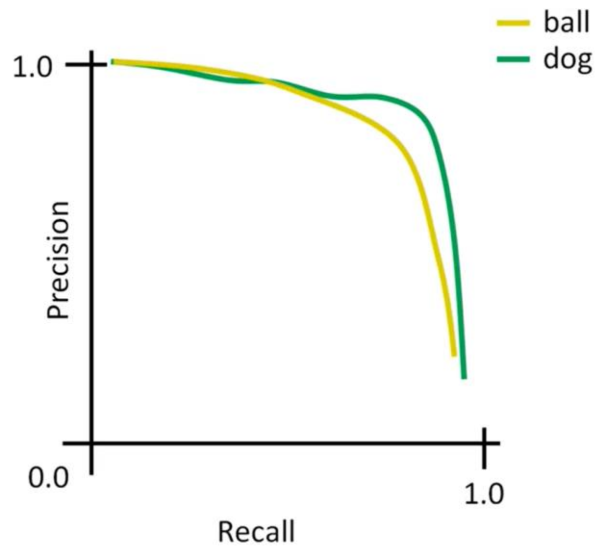
$P = 0, R = 0$

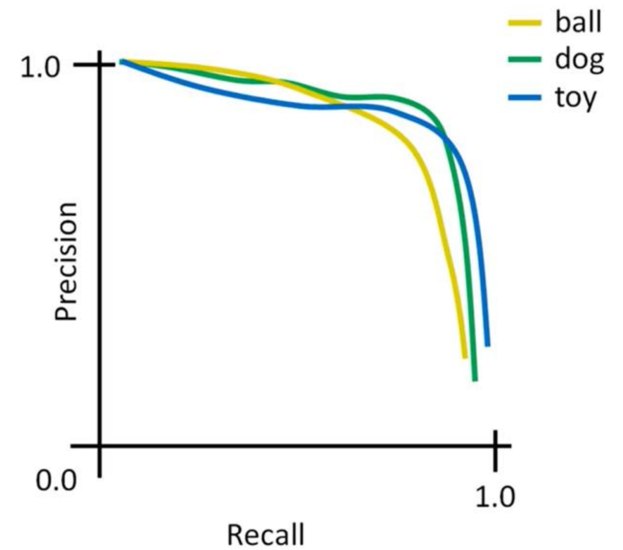
FN=10	TN=0
TP=0	FP=2

$P = 0, R = 0$

mean Average Precision (mAP)

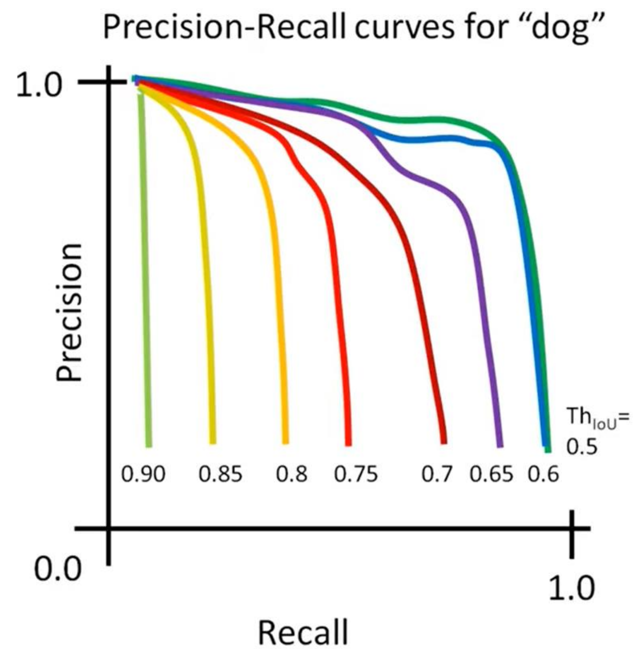






- AP for each class (either dog, or ball, or toy) w. IoU threshold 0.5 is AUPRC under each curve (either dog, or ball, or toy)
- mAP for all 3 classes w. IoU threshold 0.5 is the average AP among 3 classes (dog, ball, toy)
  - $mAP_{0.5} = \frac{1}{3}(AP_{\text{dog}} + AP_{\text{ball}} + AP_{\text{toy}})$

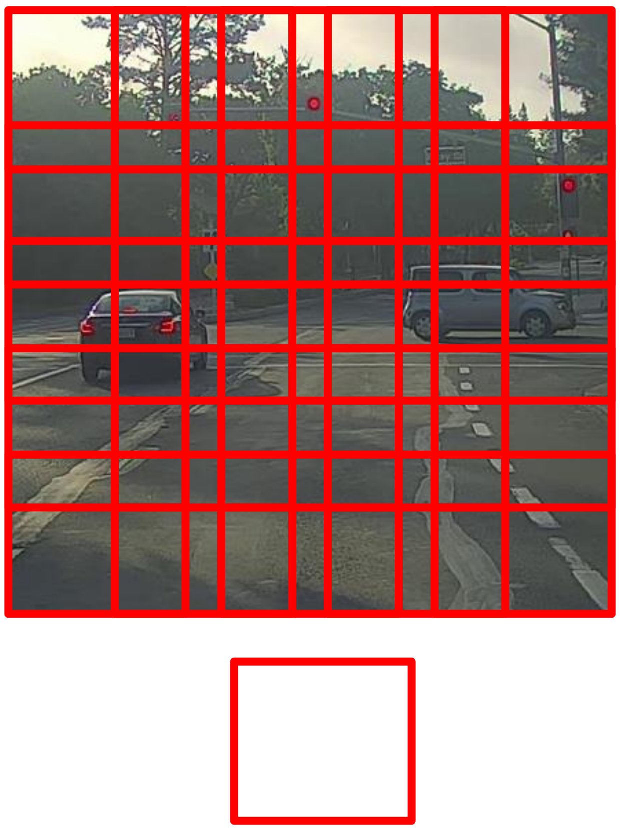
mAP

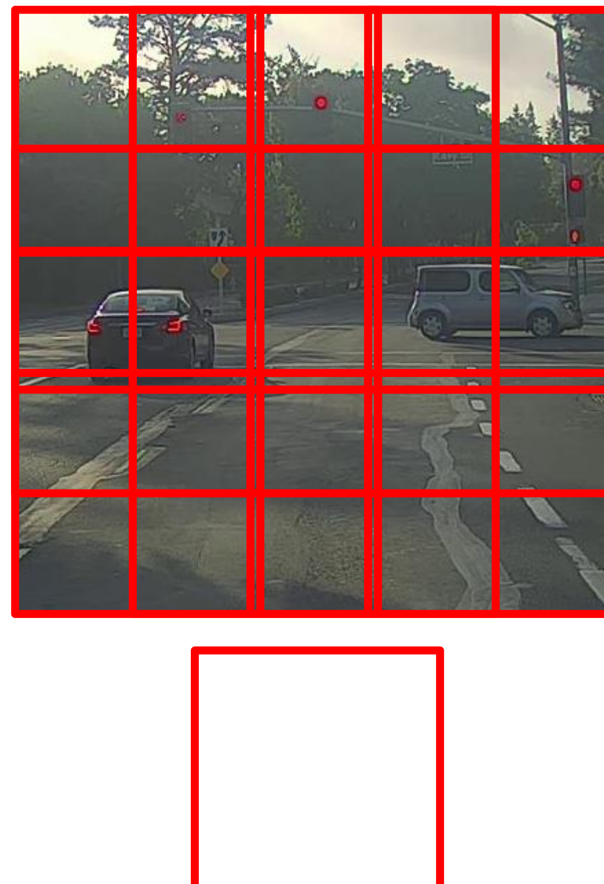


- For COCO 2017 challenge: Compute mAP@threshold for each IoU threshold and take average
  - $mAP = \frac{1}{10} \sum_i mAP_i$
  - where $i = [.5, .55, .6, .65, .7, .75, .8, .85, .9, .95]$
- Figure assumes at least one match (IoU $\geq$ threshold), hence precision starts at 1. If 0 match (all boxes have (IoU < threshold), then $mAP = 0$, since $TP = 0$, and P-R curve has only one point (0,0)

Detecting Multiple Objects: Sliding Window

- Slide a box across the image, and apply a CNN to classify each image patch as object or background





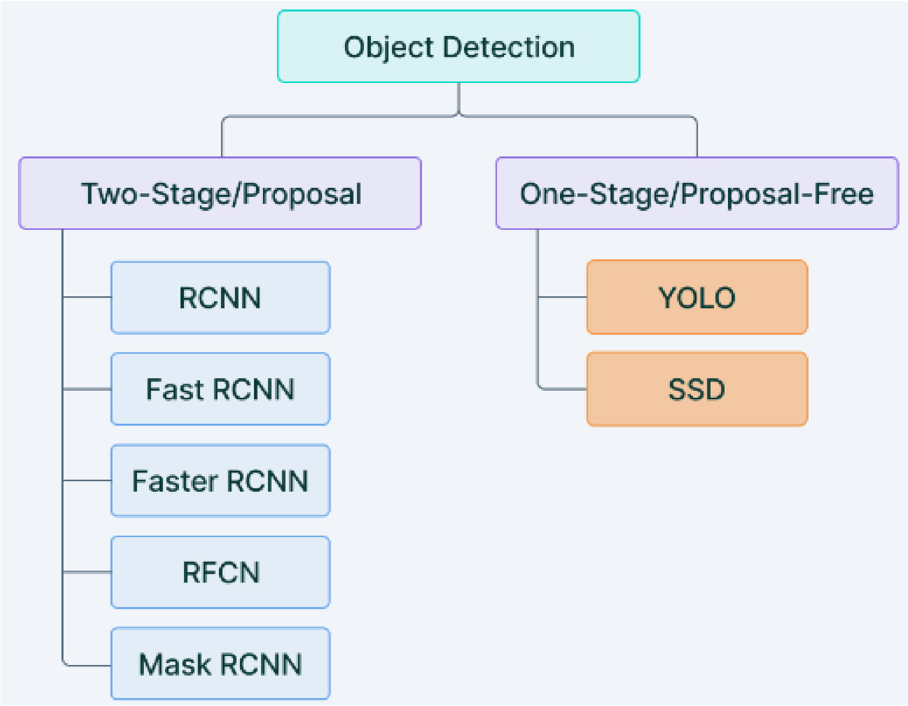
https://www.coursera.org/learn/convolutional-neural-networks/lecture/VgyWR/object-detection

Sliding Window Computational Complexity

- Total number of possible box positions in an image of size $H \times W$:
  - Consider a box of size $h \times w$:
  - Possible x positions: $W - w + 1$; Possible y positions: $H - h + 1$ (assuming stride of 1)
  - Total # possible positions: $(W - w + 1)(H - h + 1)$
  - Consider all possible box sizes: $1 \leq h \leq H, 1 \leq w \leq W$
  - Total # possible boxes: $\sum_{w=1}^{W} \sum_{h=1}^{H} (W - w + 1)(H - h + 1) = \frac{H(H+1)}{2} \frac{W(W+1)}{2}$
  - For an 800x600 image, that is 57 million!
- Can be more efficient with convolution implementation of sliding windows, but still too slow to be practical

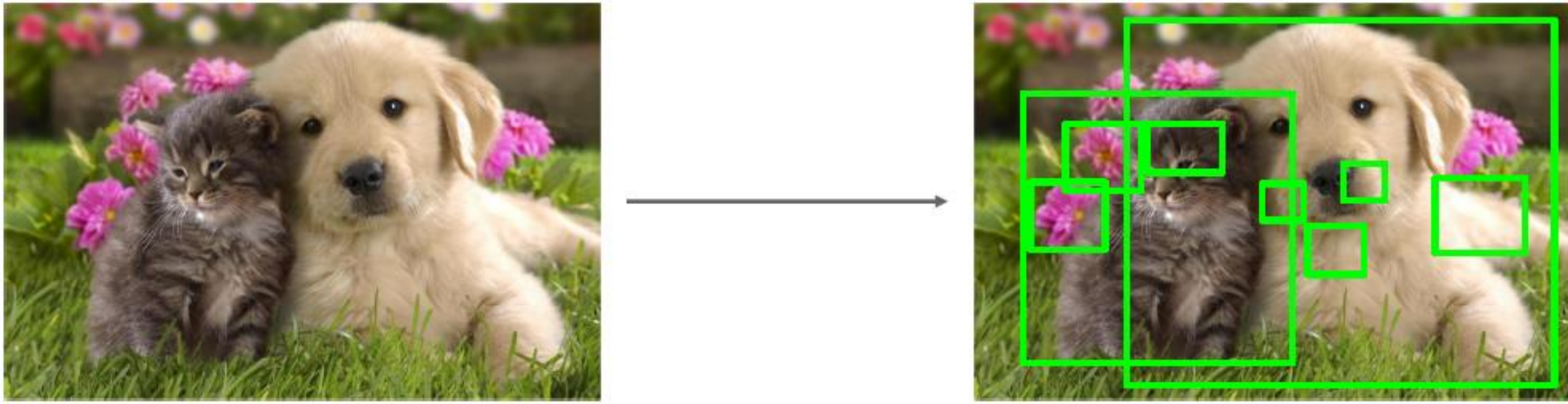
Two-stage vs. One-Stage Detector

- Two-stage detector
  - 1st step: generate Regions of Interests (Region Proposals) that are likely to contain objects
  - 2nd step: perform object detection, incl. classification and regression of Bboxes of the objects
- One-stage detector
  - Directly perform object detection, incl. classification and regression of Bboxes of the objects



Region Proposals

- Generating region proposals: find a small set of boxes that are likely to cover all objects, based on Selective Search, e.g., look for "blob-like" image regions
  - Relatively fast to run: e.g. can generate ~2000 region proposals in a few seconds on CPU

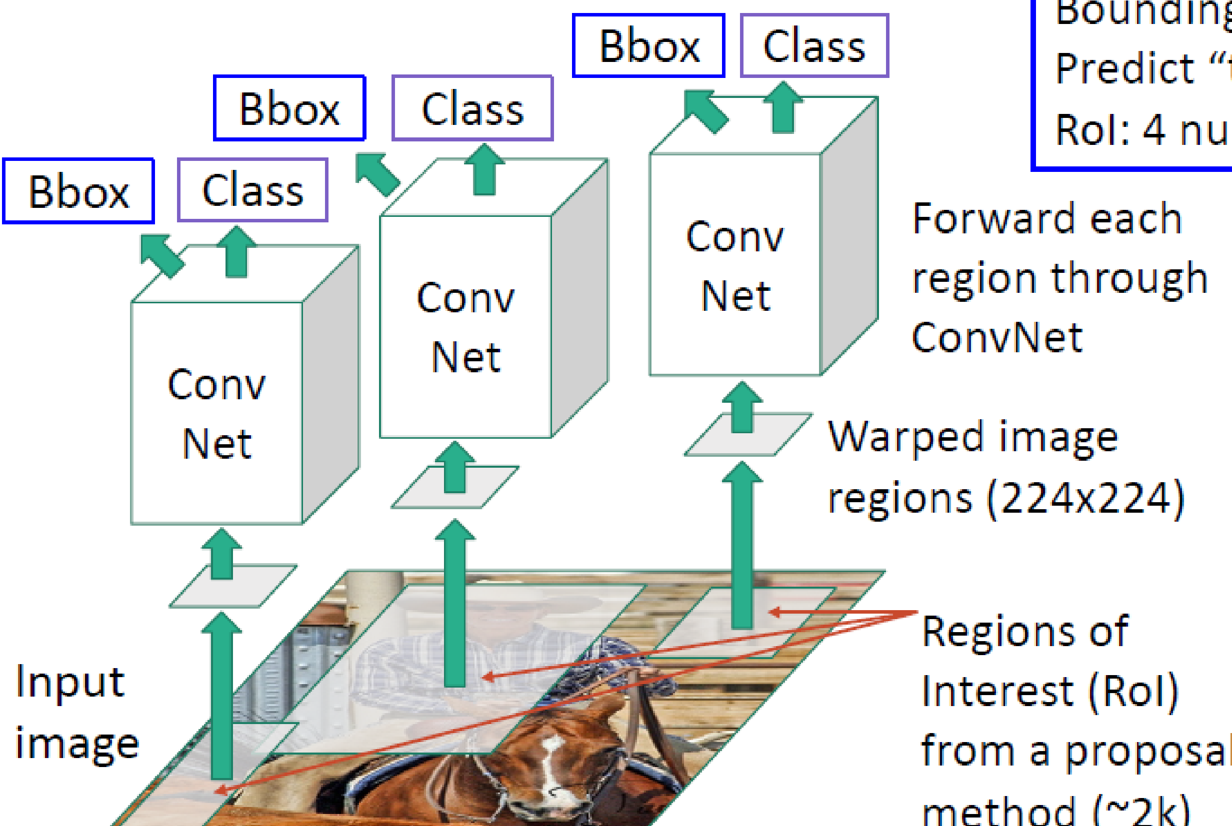


Important

R-CNN: Training Time

- Crop/warp each region proposal into same-size (e.g., 224×224) image regions, and run each through a CNN to get Bbox and class label for each region
- Bbox regression: transform each region proposal with learnable parameters (t_x, t_y, t_h, t_w) into a better Bbox

R-CNN: Region-Based CNN



Classify each region

Bounding box regression:
Predict "transform" to correct the RoI: 4 numbers (t_x, t_y, t_h, t_w)

Region proposal: (p_x, p_y, p_h, p_w)
Transform: (t_x, t_y, t_h, t_w)
Output box: (b_x, b_y, b_h, b_w)

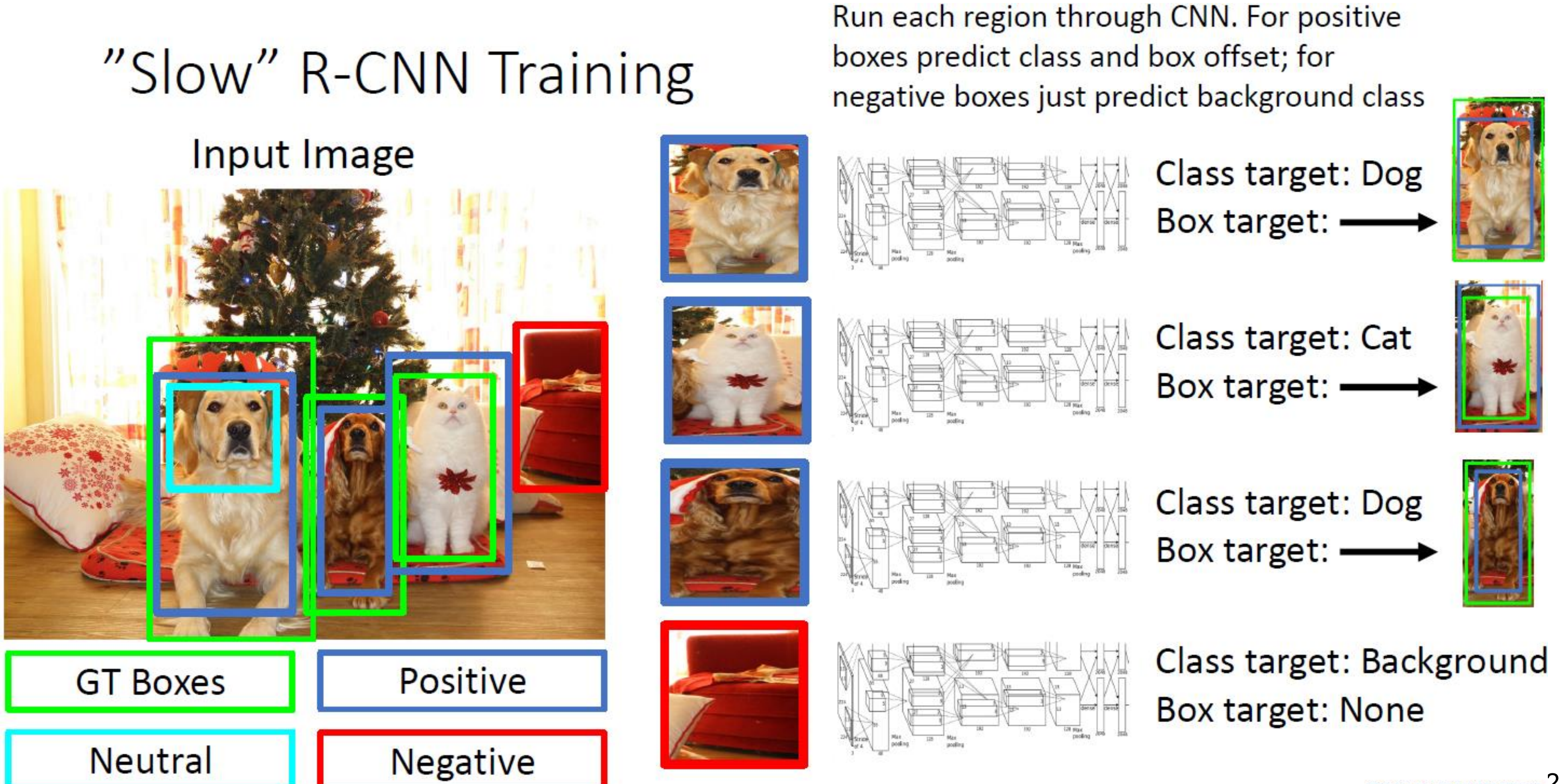
Translate relative to box size:
$b_x = p_x + p_w t_x \qquad b_y = p_y + p_h t_y$

Log-space scale transform:
$b_w = p_w \exp(t_w) \qquad b_h = p_h \exp(t_h)$

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

R-CNN Training Example

- Categorize each region proposal as positive, negative, or neutral based on overlap with ground-truth boxes
- Crop pixels from each positive and negative proposal, resize to 224 x 224
- Use the CNN for Bbox regression and classification for positive boxes; only 1-class prediction for negative boxes



R-CNN: Test Time

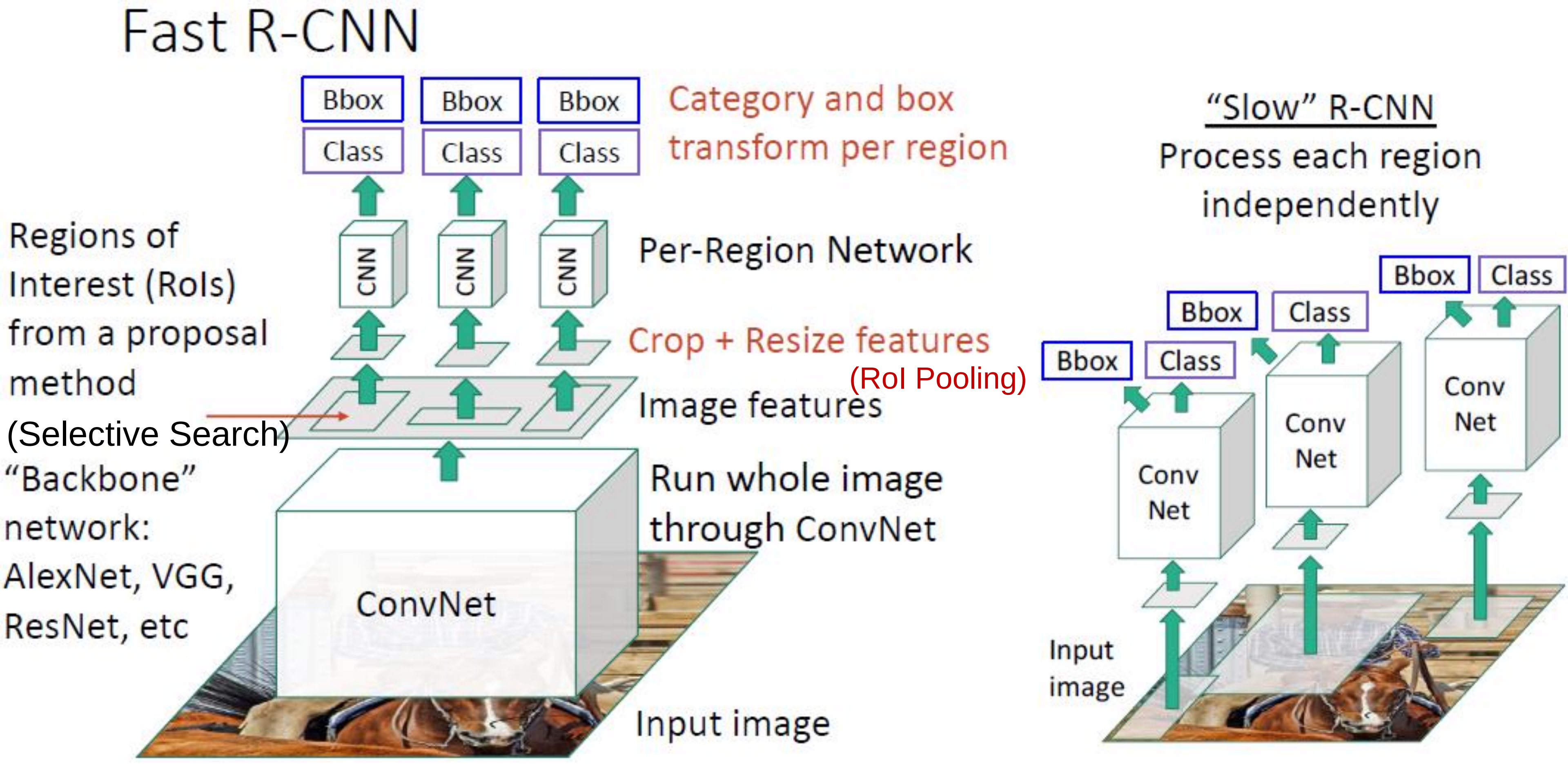
- 1. Run region proposal method to compute ~2000 region proposals
- 2. Resize each region to 224x224 (tunable hyperparam) and run independently through the CNN to get feature vectors, then use Linear SVM to predict class scores, and Linear Regression to predict Bboxes
- 3. Use scores to select a subset of region proposals to output
  - Many choices here: threshold on background score (e.g., output bottom K proposals with lowest background scores), or per-class (e.g., output top K proposals with highest classification scores for the given class)…
- 4. Compare with ground-truth Bboxes

R-CNN is Slow

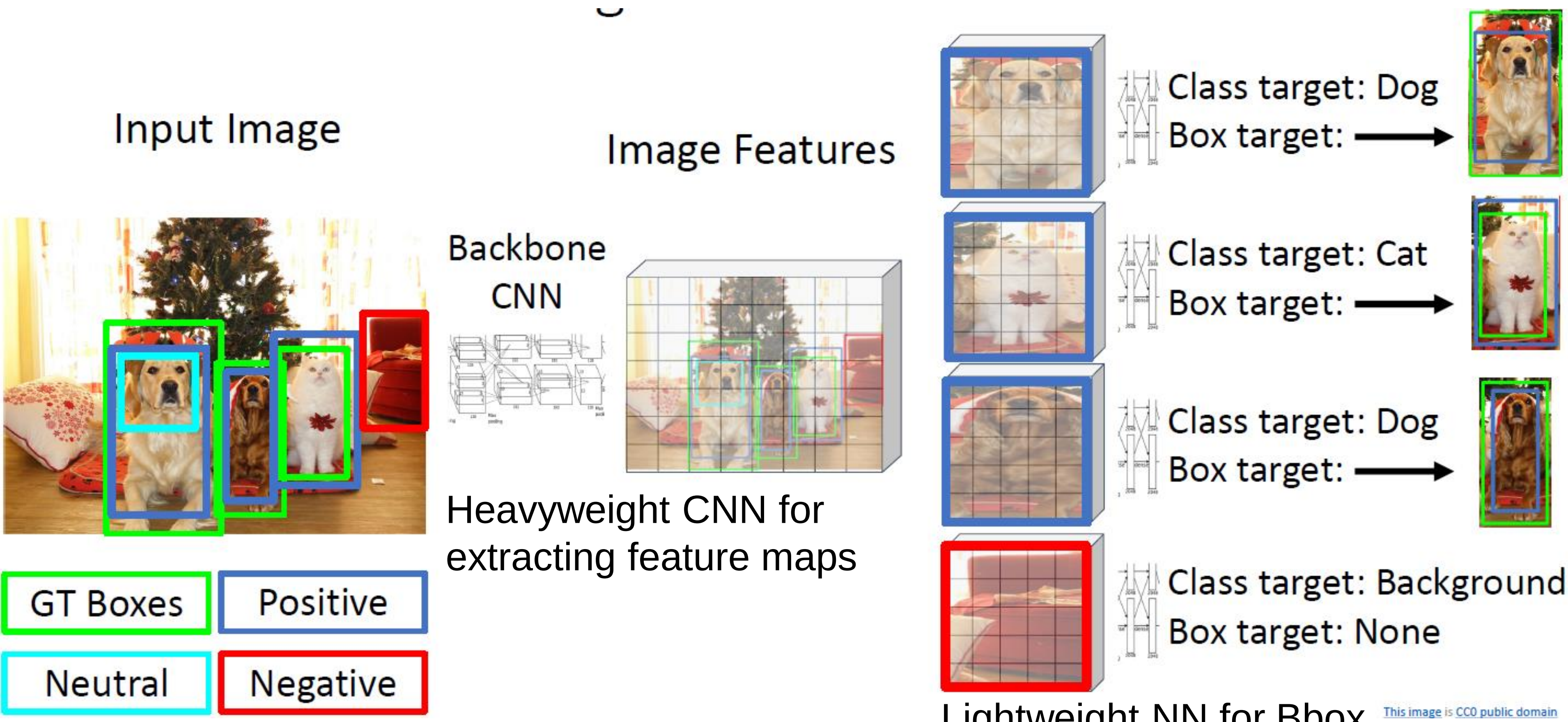
- Inference time: ~40-50s per image
  - Extracting ~2000 regions for each image based on selective search
  - Extracting feature vectors using CNN for every image region.
  - Suppose we have N images, then the number of CNN features will be N*2000

Fast R-CNN

- 1. Use a backbone network to extract feature maps from the whole image
- 2. Apply Selective Search on these feature maps and get object proposals
- 3. Use a lightweight Per-Region network to perform Bbox regression and classification
- Most of the computation happens in backbone network; this saves work for overlapping region proposals compared to R-CNN



Fast R-CNN Training Example

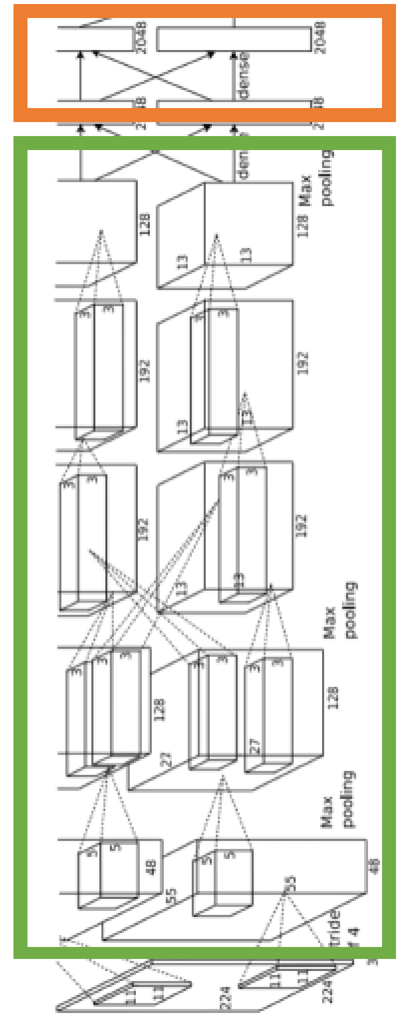


Heavyweight CNN for extracting feature maps

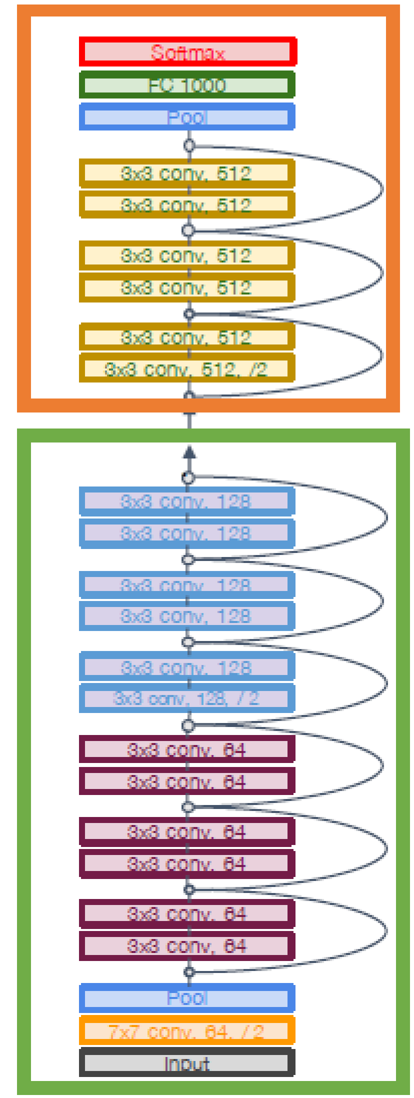
Lightweight NN for Bbox regression & classification

Example Backbone and Per-Region Networks

- When using AlexNet for detection, 5 CONV layers are used for backbone and 2 FC layers are used for per-region network
- For ResNet, the last stage (CONV+FC) is used as per-region network; the rest of the network is used as backbone



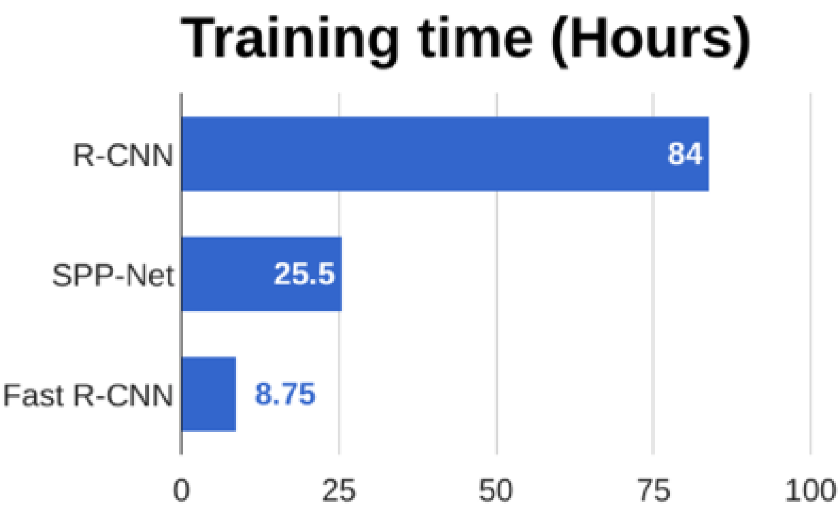
AlexNet

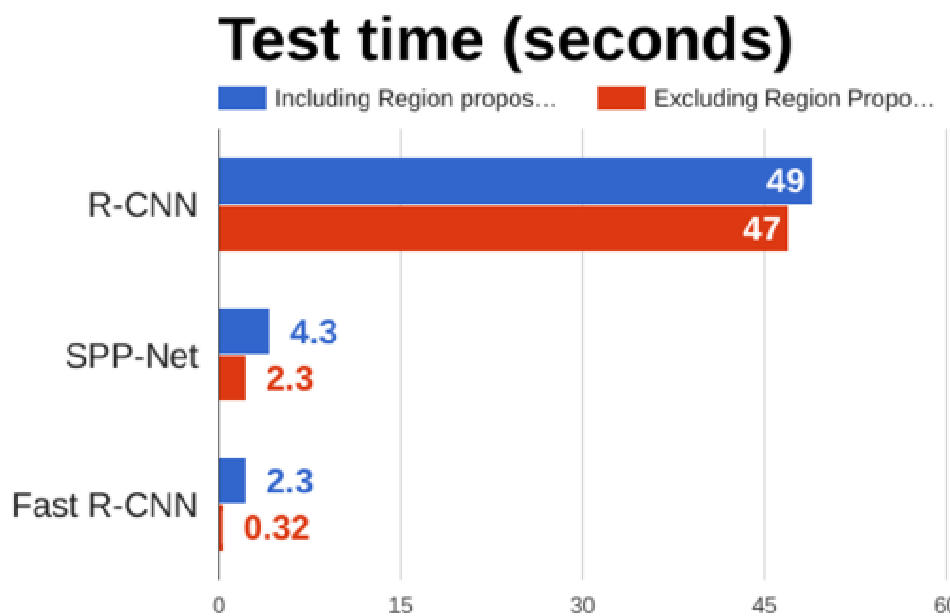


ResNet

Fast R-CNN Performance

- Problem: Test time of Fast R-CNN is dominated by region proposals
- Solution: instead of using the heuristic ”Selective Search” algorithm on CPU, let’s learn them with a CNN instead





Faster R-CNN

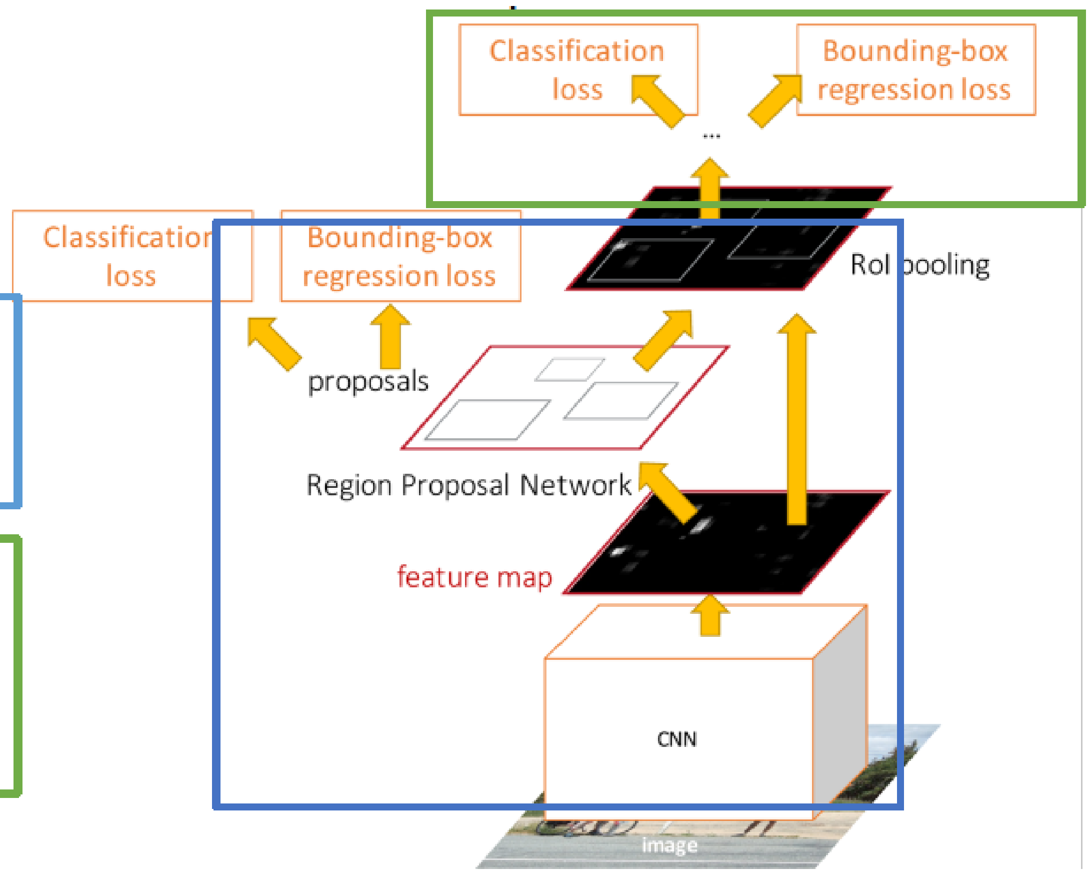
- Use Region Proposal Network (RPN) to generate region proposals from feature maps output by the backbone network
  - vs. Selective Search used by Fast R-CNN
  - RPN is learnable/trainable; Selective Search is a fixed heuristic algorithm
- The rest the same as Fast R-CNN

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: Loss Function

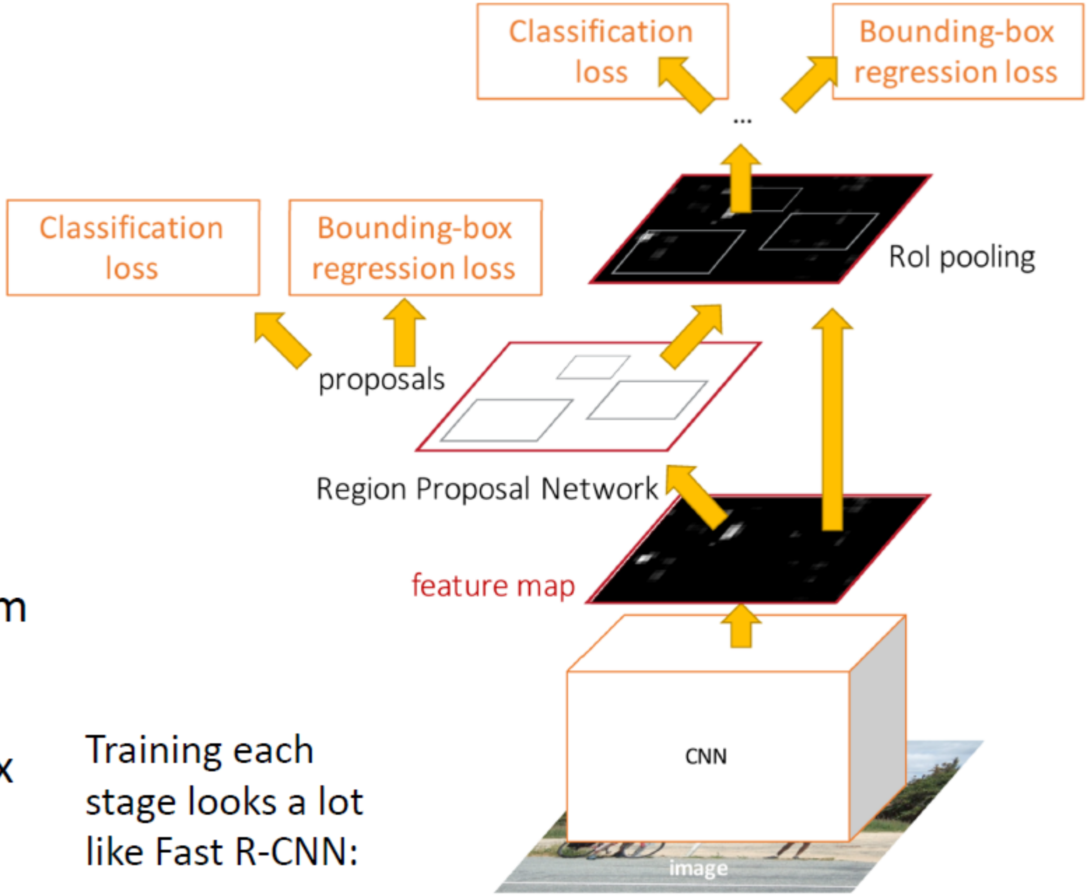
Jointly train with 4 losses:

1. **RPN classification**: anchor box is object / not an object
2. **RPN regression**: predict transform from anchor box to proposal box
3. **Object classification**: classify proposals as background / object class
4. **Object regression**: predict transform from proposal box to object box

Anchor -> Region Proposal -> Object Box

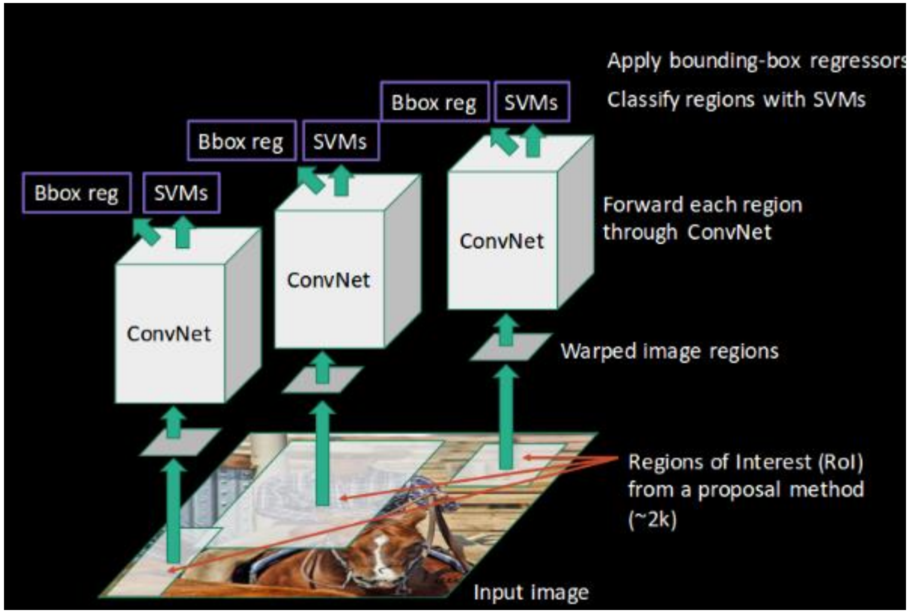
(Stage 1) (Stage 2)

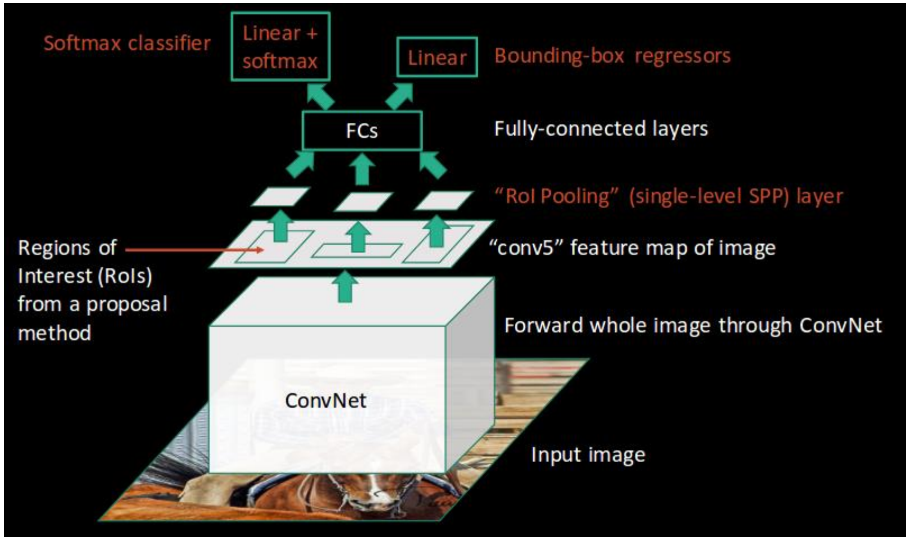
Training each stage looks a lot like Fast R-CNN:



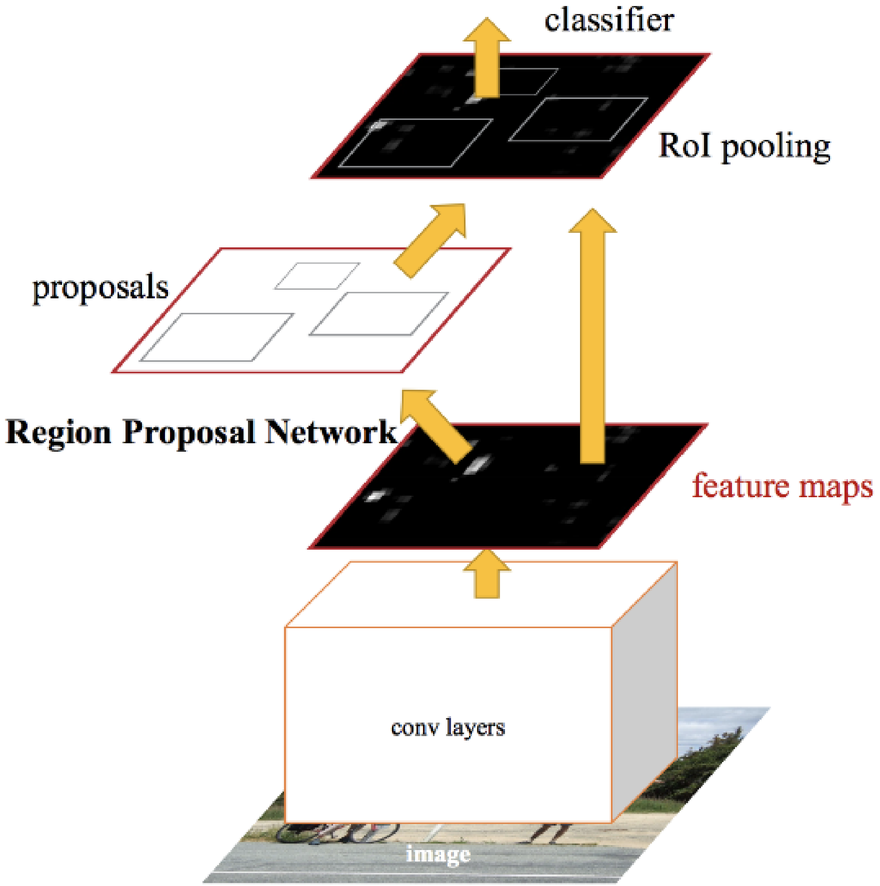
3 Variants of R-CNN

R-CNN





Fast R-CNN



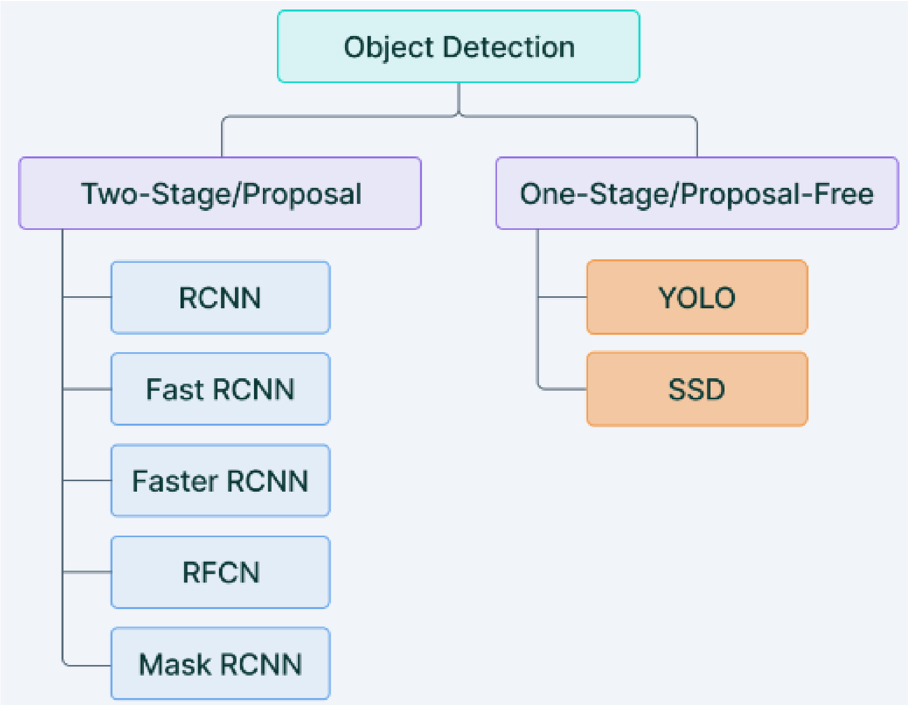
Faster R-CNN

Summary of 3 Variants of R-CNN

Algorithm	Characteristics	Pred. Time/Img (s)	Limitations
R-CNN	Use Selective Search on the input image to generate regions. Extracts ~2000 regions from each image.	40-50	High computation time as each region is passed to the CNN separately
Fast R-CNN	Each image is passed only once to the CNN and feature maps are extracted. Use Selective Search on feature maps to generate predictions. Combines all the three models used in RCNN together.	2	Relatively high computation time using selective search
Faster R-CNN	Replaces Selective Search with Region Proposal Network to make the algorithm much faster.	0.2	

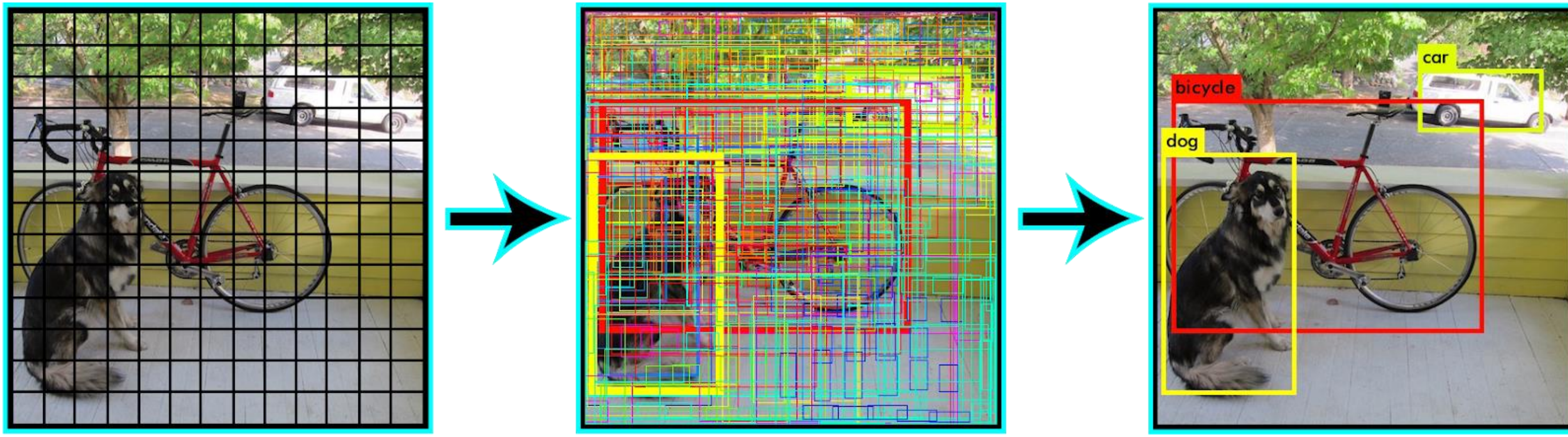
Two-stage vs. One-Stage Detector

- Two-stage detector
  - 1^{st} step: generate Regions of Interests (Region Proposals) that are likely to contain objects
  - 2^{nd} step: perform object detection, incl. classification and regression of Bboxes of the objects
- One-stage detector
  - Directly perform object detection, incl. classification and regression of Bboxes of the objects



YOLO

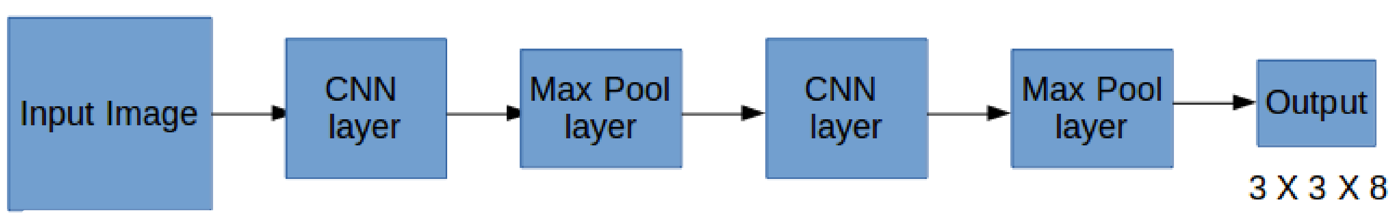
- Divide the input image into n-by-n grids
- For each grid, predict Bboxes and their class labels for objects (if any are found)
  - The Bboxes are highlighted by yellow color in the second step.
- Apply Non-Maximal Suppression based on IoU, we suppress Bboxes with lower probability scores to achieve final Bboxes



Label for a Grid cell

- Suppose we divide the image into a 3x3 grid. We define 3 classes (Pedestrian, Car, Motorcycle)
- For each grid cell, the label y is an 8-D vector
  - p_c defines whether an object is present in the grid or not (it is the probability)
  - (b_x, b_y, b_h, b_w) specify the Bbox if there is an object
  - (c_1, c_2, c_3) represent one-hot vector as the target label (or the class confidence scores computer by SoftMax during inference)
- For each of the 3x3 grid cells, we have an 8-D output vector. So the output dimension is 3x3x8
- Even if an object may span more than one grid, it will only be assigned to a single grid cell in which its mid-point is located

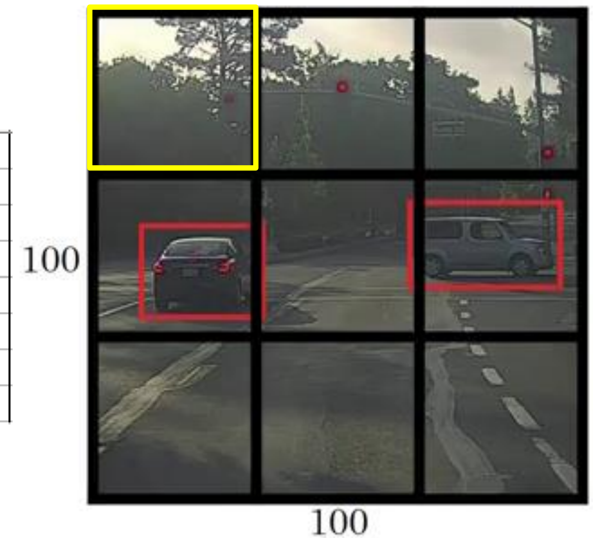
	pc
	bx
	by
y =	bh
	bw
	c1
	c2
	c3



Two Grid Cells

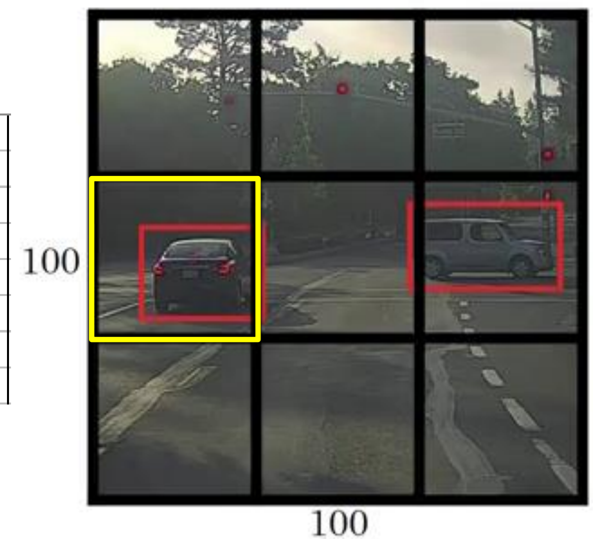
- Since there is no object in this grid, pc=0, and all the other entries are ?

y =	0
	?
	?
	?
	?
	?
	?
	?



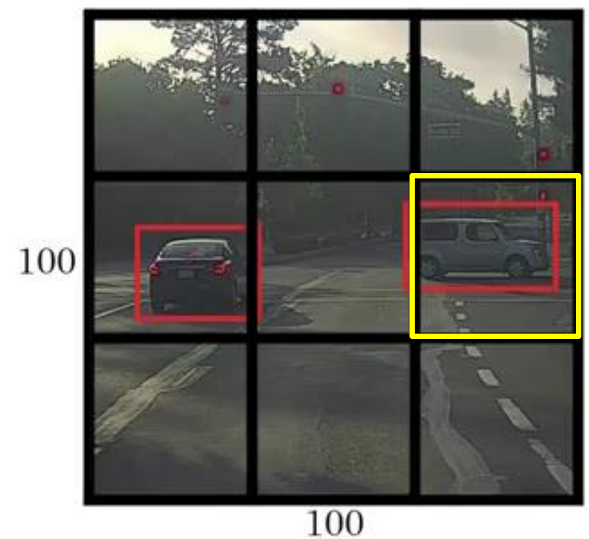
- Since there is an object (Car) in this grid, pc=1. (bx, by, bh, bw) are calculated relative to the particular grid cell
- Class label is (0,1,0) since Car is the 2nd class

y =	1
	bx
	by
	bh
	bw
	0
	1
	0

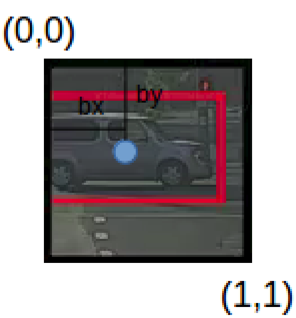


An Example Grid Cell

- (b_x, b_y)=(0.4, 0.3): coordinates of the midpoint of the object with respect to this grid
- (b_h, b_w)=(0.5, 0.9):
  - b_h = 0.5 is ratio of the height of the Bbox (red box) to the height of the grid cell
  - b_w = 0.9 is ratio of the width of the Bbox to the width of the grid cell
- b_x and b_y will never exceed 1, as the midpoint will always lie within the grid. Whereas b_h and b_w can exceed 1 if the dimensions of the Bbox are more than the dimension of the grid
- Class label is (0,1,0) for the Car class

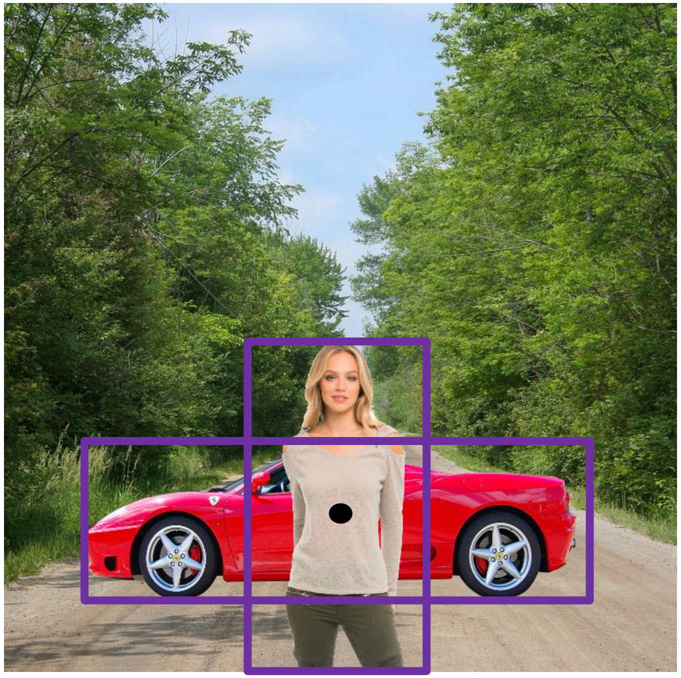


$$y = \begin{bmatrix} 1 \\ bx \\ by \\ bh \\ bw \\ 0 \\ 1 \\ 0 \end{bmatrix} = y = \begin{bmatrix} 1 \\ 0.4 \\ 0.3 \\ 0.5 \\ 0.9 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$



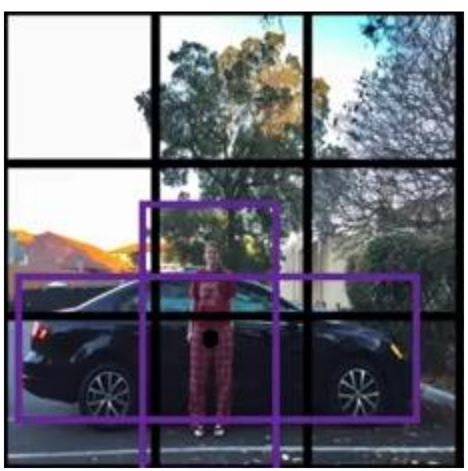
Anchor Boxes

- Place K anchor boxes centered at each position in the feature map, each with different sizes and aspect ratios ($K = 2$ in the fig)
  - This allows detection of multiple objects centered at the same position, and better-fitting anchor boxes, which helps ease the downstream Bbox regression task

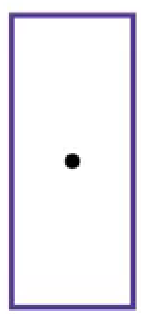


Anchor Boxes

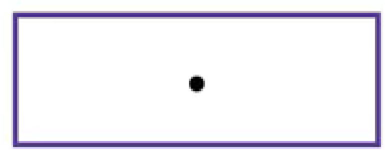
- Suppose we have 2 anchor boxes for each grid cell. Then we can detect at most two objects for each grid cell
- First 8 rows of the y label belong to anchor box 1 and the remaining 8 belongs to anchor box 2. The y vector has 16 entries, and the output has dimension 3x3x2x8=3x3x16
- The objects are assigned to the anchor boxes based on the similarity of the Bboxes and the anchor box shape, e.g., the person is assigned to anchor box 1 and the car is assigned to anchor box 2
- Suppose we use 5 anchor boxes per grid and the number of classes is 5, then the output has dimension 3x3x5x10= 3x3x50



Anchor box 1:



Anchor box 2:



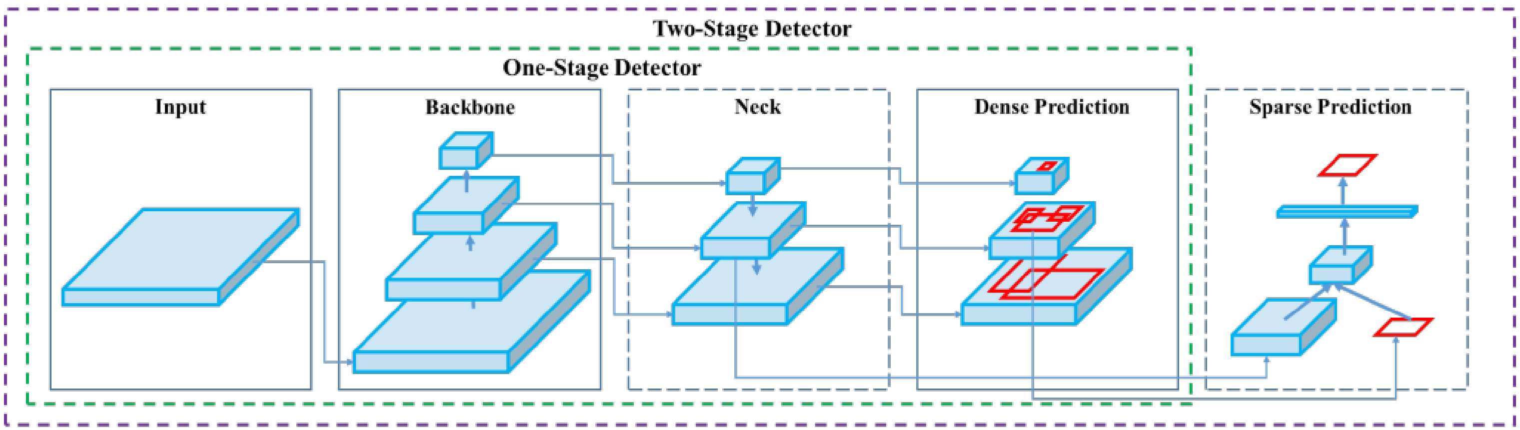
	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3
y =	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3

Realistic YOLO Dimensions

- Input image has shape (608, 608, 3)
- The CNN output has dimension (19, 19, 5, 85), where each grid cell returns 5*85 numbers, with total dimension of 19*19*5*85
  - 5 is the number of anchor boxes per grid
  - 85 = 5+80, where 5 refers to (pc, bx, by, bh, bw), and 80 is the number of classes
- Finally, compute IoU and perform Non-Max Suppression

YOLO with FPN

- Backbone extracts essential features of an image and feeds them to the Head through Neck
- Neck is a Feature Pyramid Network (FPN) that collects feature maps extracted by the Backbone and creates feature pyramids
  - An FPN is a feature extractor that takes a single-scale image of an arbitrary size as input, and outputs proportionally sized feature maps at multiple levels
- Head consists of output layers that make final detections.



Input: { Image, Patches, Image Pyramid, … }

Backbone: { VGG16 [68], ResNet-50 [26], ResNeXt-101 [86], Darknet53 [63], … }

Neck: { FPN [44], PANet [49], Bi-FPN [77], … }

Head:

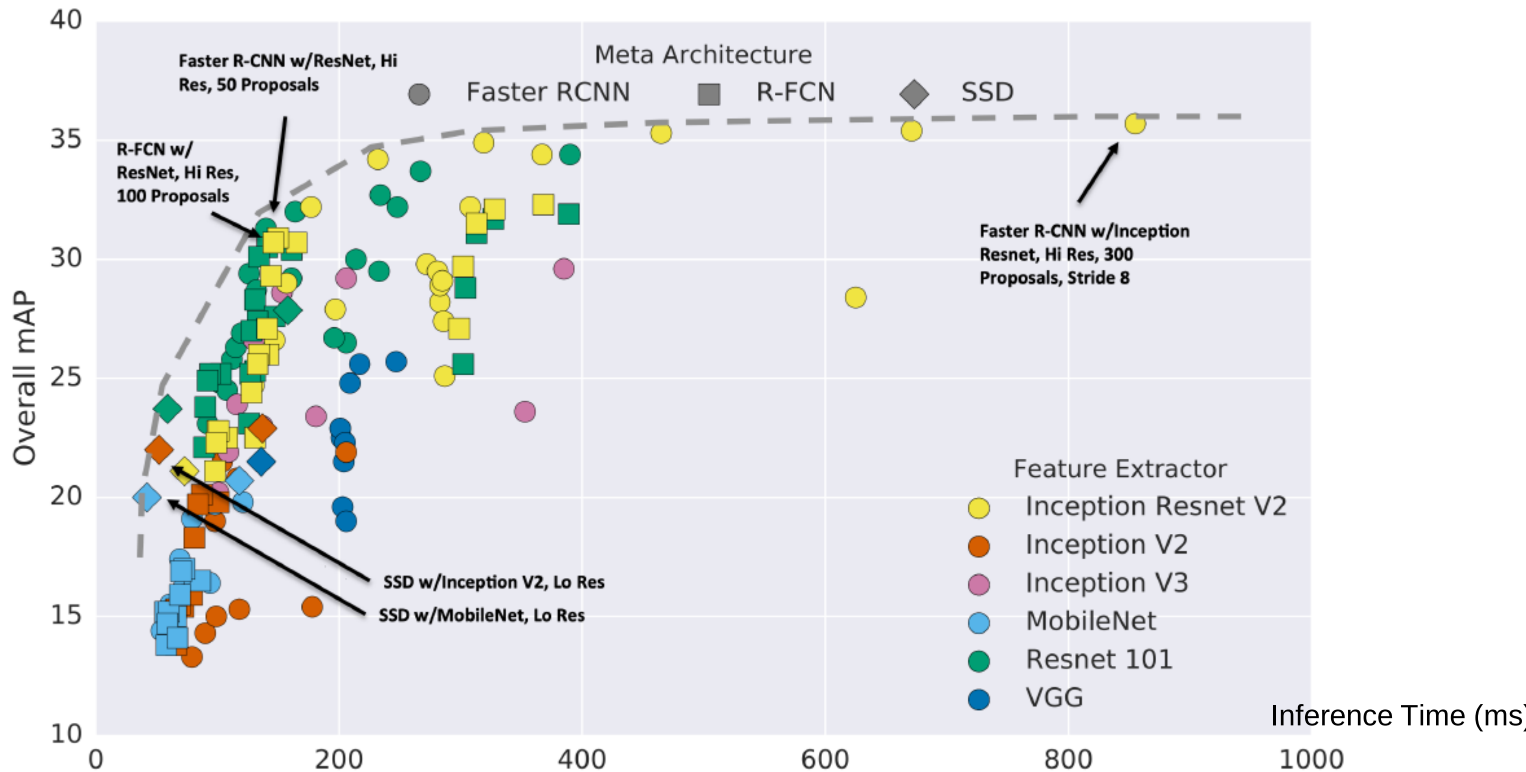
Dense Prediction: { RPN [64], YOLO [61, 62, 63], SSD [50], RetinaNet [45], FCOS [78], … }

Sparse Prediction: { Faster R-CNN [64], R-FCN [9], … }

Figure 2: Object detector.

Performance Comparisons (2017)

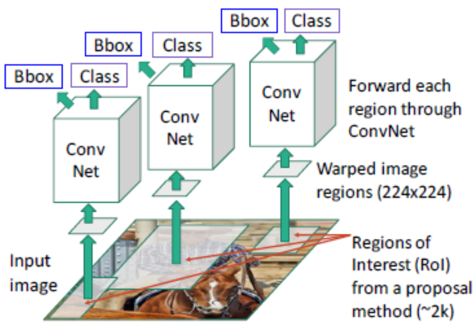
- Two stage method (Faster R-CNN) get the best accuracy, but are slower
- Single-stage methods (SSD) are much faster, but don't perform as well
- Bigger backbones improve performance, but are slower



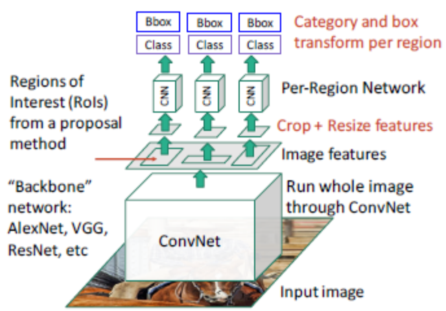
Important

Summary of Object Detectors

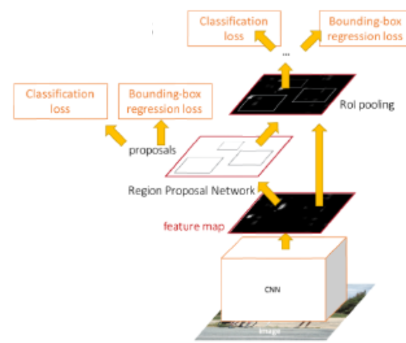
“Slow” R-CNN: Run CNN independently for each region



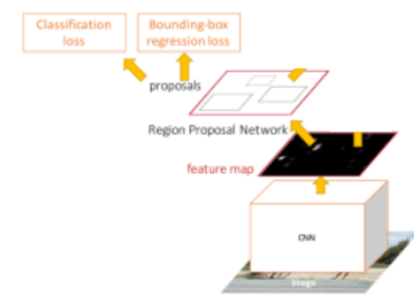
Fast R-CNN: Apply backbone network to generate feature maps once



Faster R-CNN: Compute proposals with RPN



Single-Stage: Fully convolutional detector



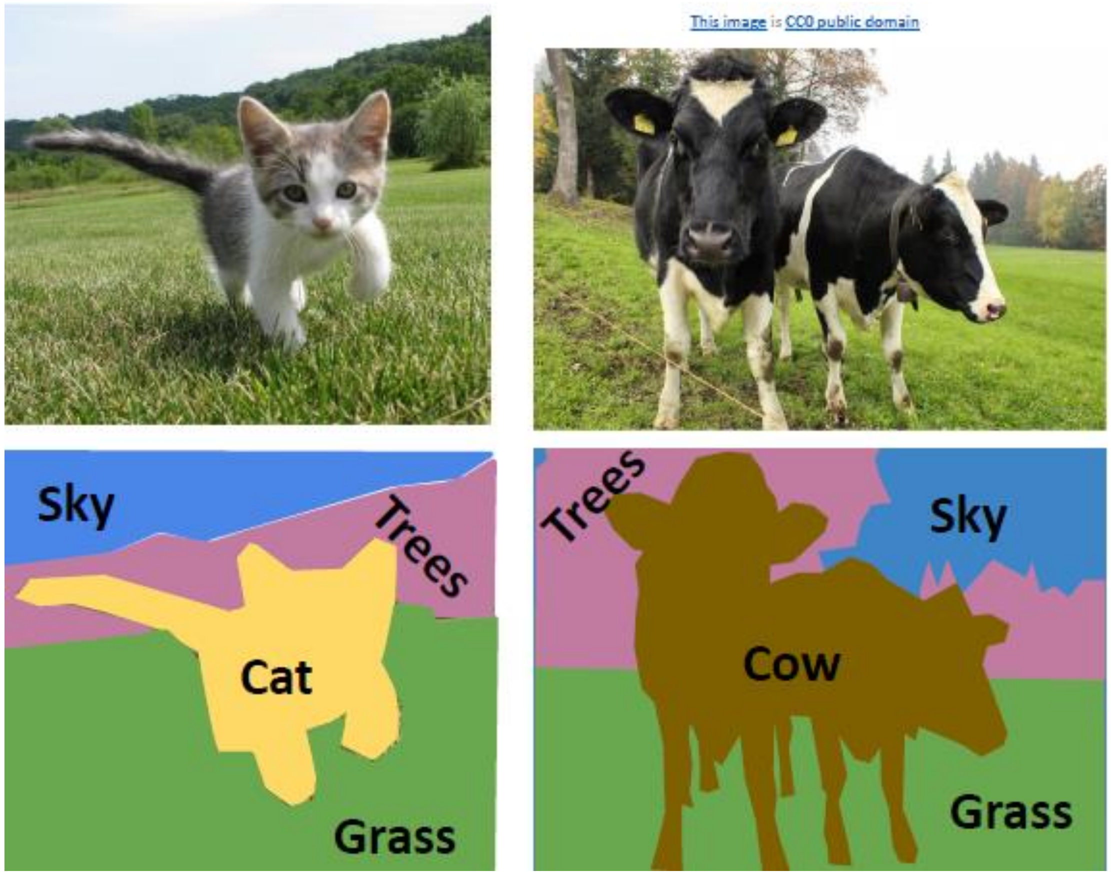
Outline

- Object detection
- Segmentation

https://medium.com/hackernoon/semantic-segmentation-datasets-for-autonomous-driving-1182ebd2aff0

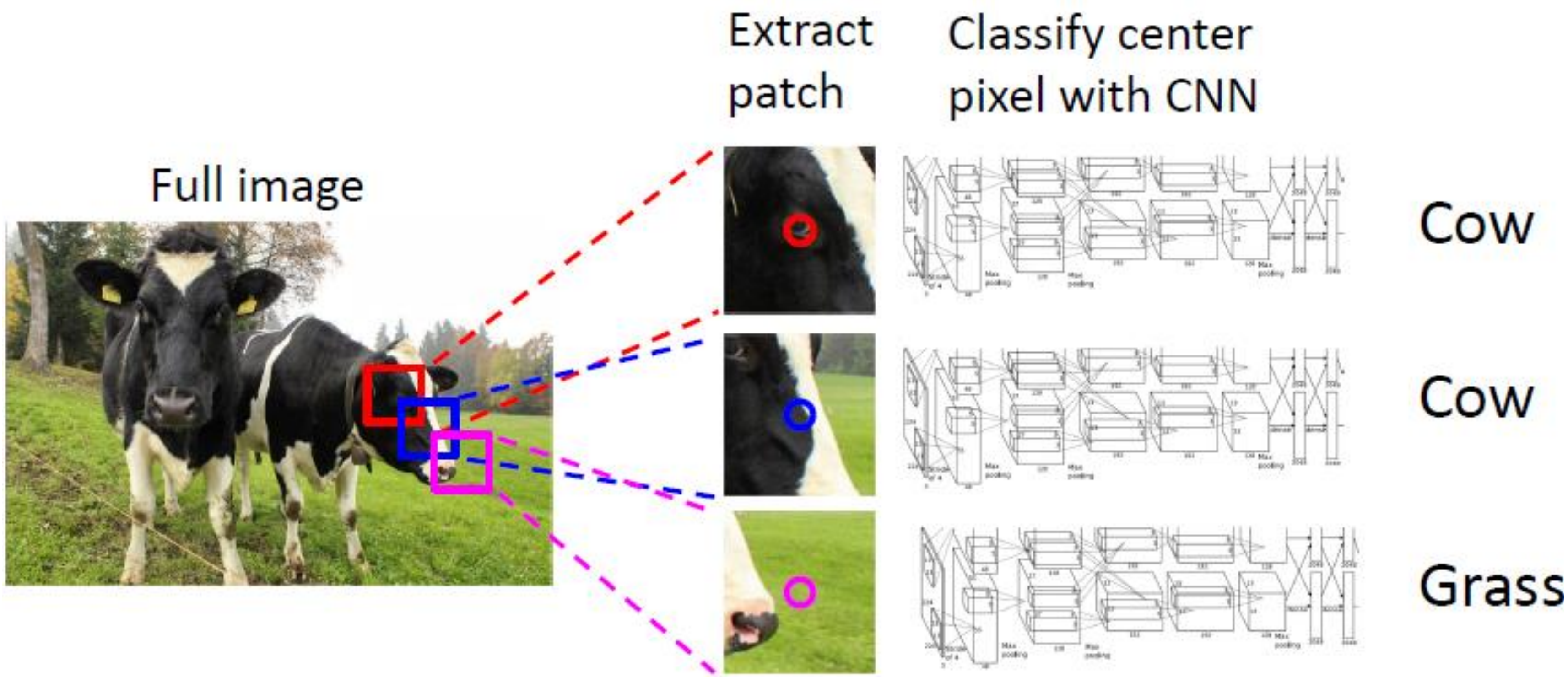
Semantic Segmentation: Task Definition

- Label each pixel in the image with a class label
- Don’t differentiate among multiple instances (e.g., pixels of the 2 cows are given the same label)



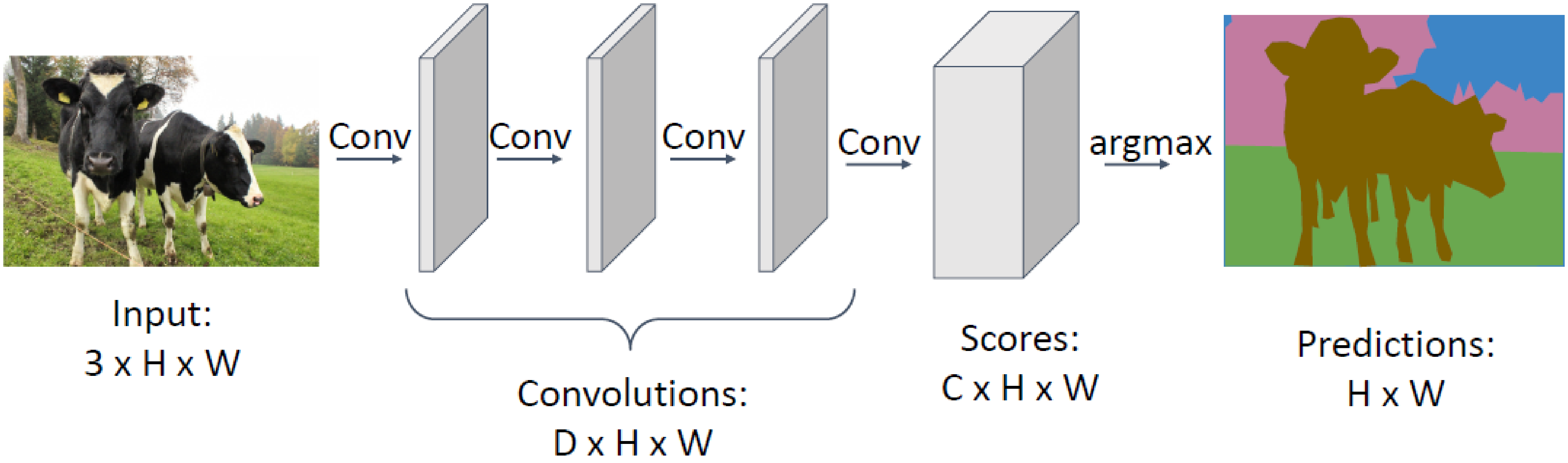
An Early Approach: Sliding Windows

- Slide a box across the image, and apply a CNN to classify each crop’s center pixel
- Inefficient, not reusing shared features between overlapping patches

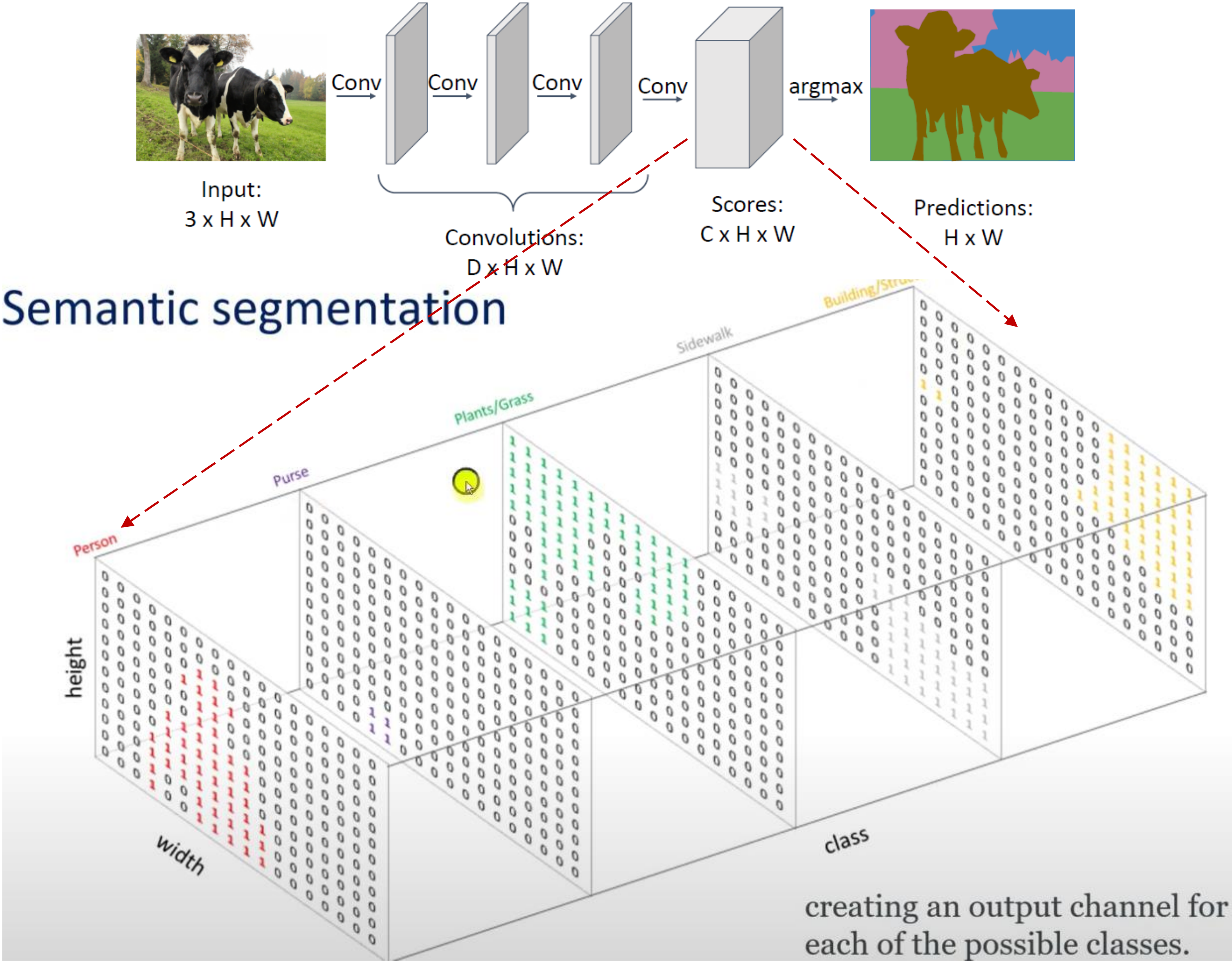


Fully Convolutional Network (FCN)

- A CNN with only CONV layers, no FC layers, for making predictions for all pixels all at once. Loss function is per-pixel Cross-Entropy loss
  - Problem #1: Effective receptive field size grows linearly in the feedforward direction with number of CONV layers: with L 3x3 CONV layers, receptive field grows slowly as 2L+1 (3x3, 5x5, 7x7…)
  - Problem #2: Convolution on high-res images without downsampling is expensive



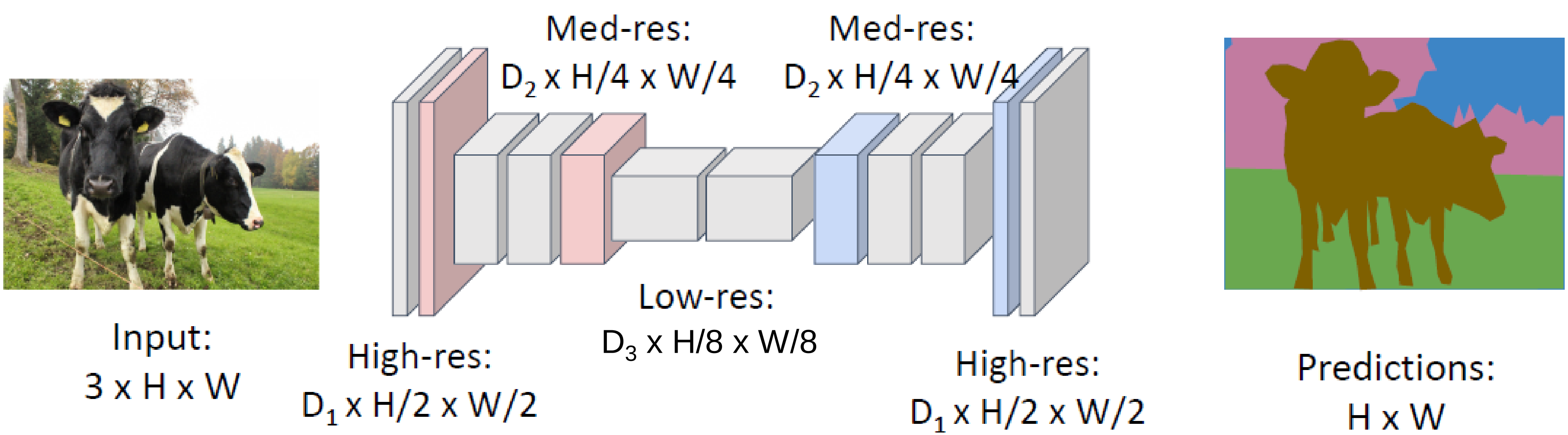
FCN Example



Conv
Conv
Conv
Conv
argmax
Input:
3 x H x W
Convolutions:
D x H x W
Scores:
C x H x W
Predictions:
H x W
Semantic segmentation
Person
Purse
Plants/Grass
Sidewalk
height
width
class
creating an output channel for each of the possible classes.

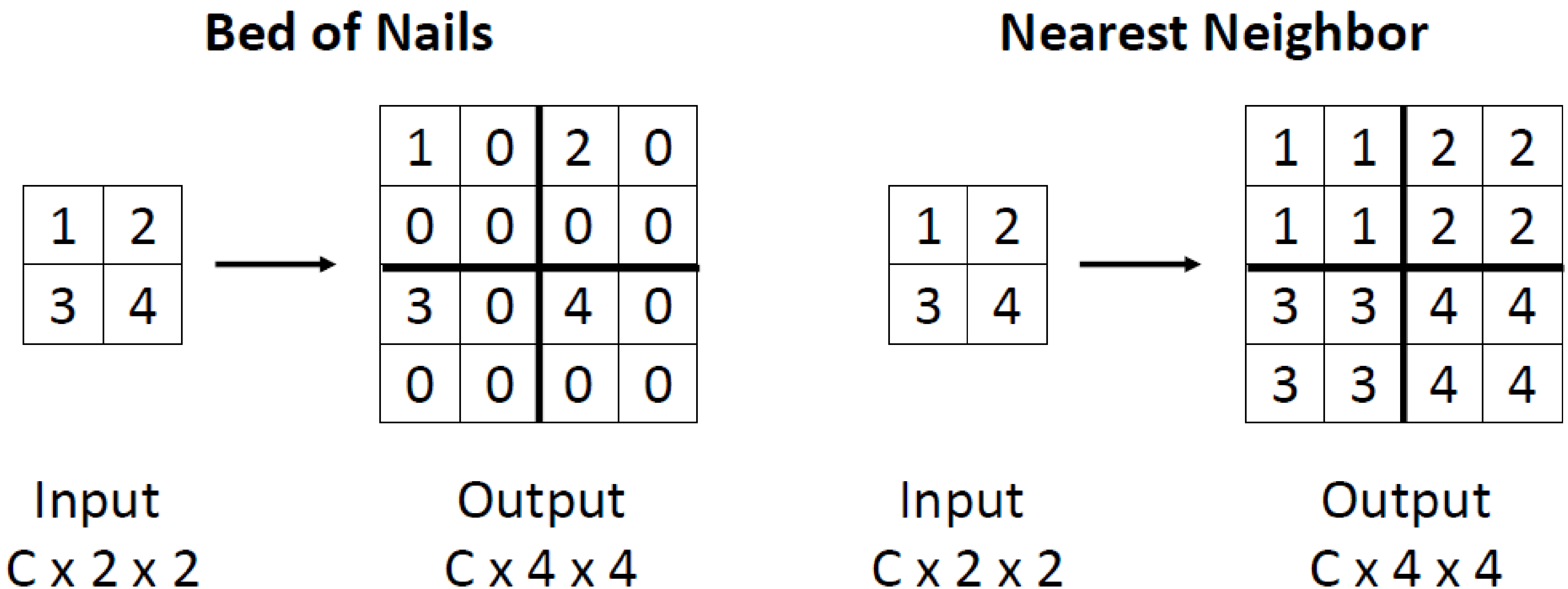
More Efficient FCN

- A CNN with CONV layers that perform downsampling followed by upsampling
  - Downsampling (with pooling or strided convolution) allows effective receptive field size to grow more quickly in the feedforward direction. It also leads to more efficient computation
  - Upsampling with interpolation or transposed convolution to get same-size output as input



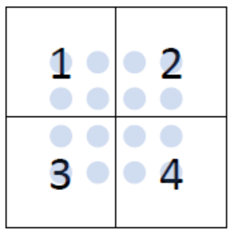
Unpooling for Upsampling

- Upsampling from a 2x2 image to a 4x4 image, by either inserting 0s (Bed of Nails), or duplicating elements (Nearest Neighbor)



Bilinear/Bicubic Interpolation for Upsampling

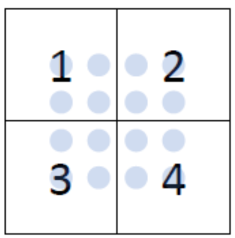
- Upsampling from a 2x2 image to a 4x4 image with bilinear (left) and bicubic (right) interpolation, to generate smoother outputs
- Each output element is computed as a linear or cubic combination of its closest neighbors; closer neighbors are given higher weights
  - Bilinear: use 4 closest neighbors in x and y to construct linear approximations
  - Bicubic: use 3 closest neighbors in x and y to construct cubic approximations



Input: C x 2 x 2

1.00	1.25	1.75	2.00
1.50	1.75	2.25	2.50
2.50	2.75	3.25	3.50
3.00	3.25	3.75	4.00

Output: C x 4 x 4



Input: C x 2 x 2

0.68	1.02	1.56	1.89
1.35	1.68	2.23	2.56
2.44	2.77	3.32	3.65
3.11	3.44	3.98	4.32

Output: C x 4 x 4

Max Unpooling

Max Pooling: Remember which position had the max

1	2	6	3
3	5	2	1
1	2	2	1
7	3	4	8

→

5	6
7	8

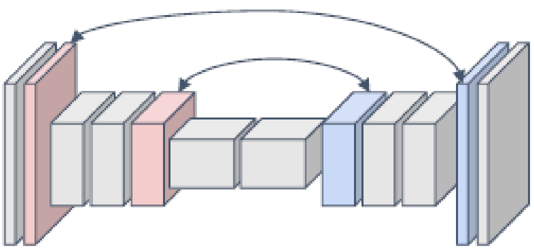
→ Rest of net →

1	2
3	4

→

Max Unpooling: Place into remembered positions

0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

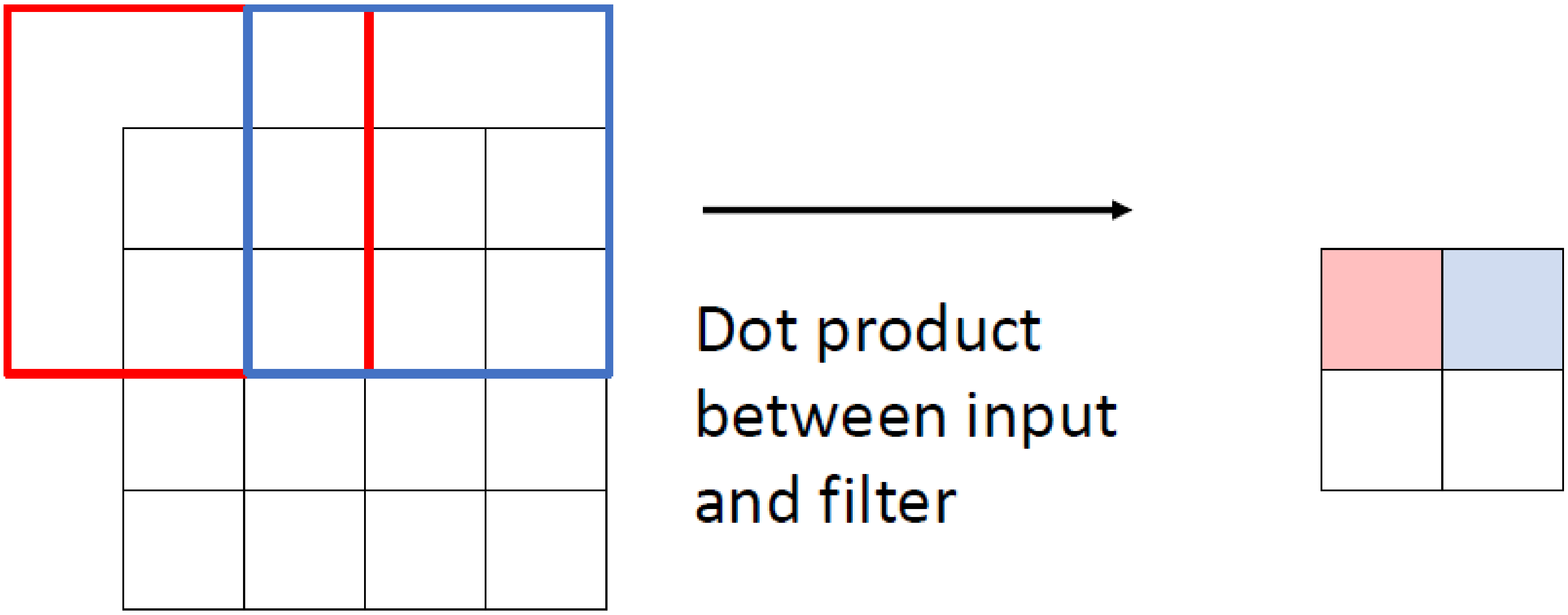


Pair each downsampling layer with an upsampling layer

Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Recall: Regular Convolution

Recall: Normal 3 x 3 convolution, stride 2, pad 1



Dot product between input and filter

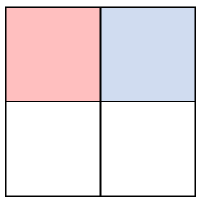
Input: 4 x 4

Output: 2 x 2

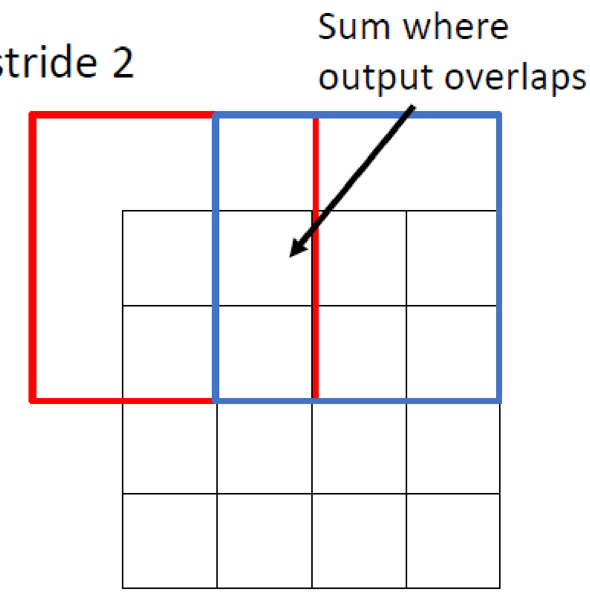
Learnable Upsampling: Transposed Convolution

3 x 3 **convolution transpose**, stride 2

Filter moves 2 pixels in <u>output</u>
for every 1 pixel in <u>input</u>



Weight filter by input value and copy to output

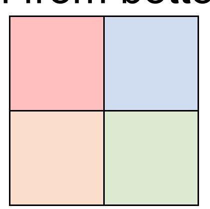


Input: 2 x 2

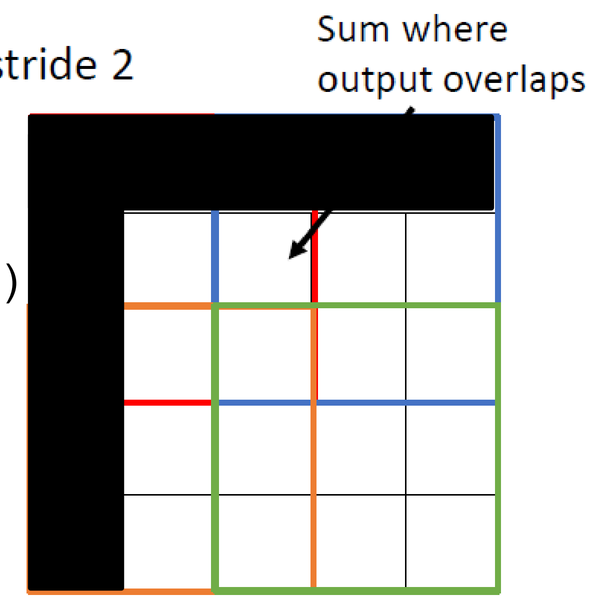
Output: 4 x 4

3 x 3 **convolution transpose**, stride 2

This gives 5x5 output – need to trim one pixel from top and left to give 4x4 output
(can also trim from bottom and right, or bottom and left…)



Weight filter by input value and copy to output

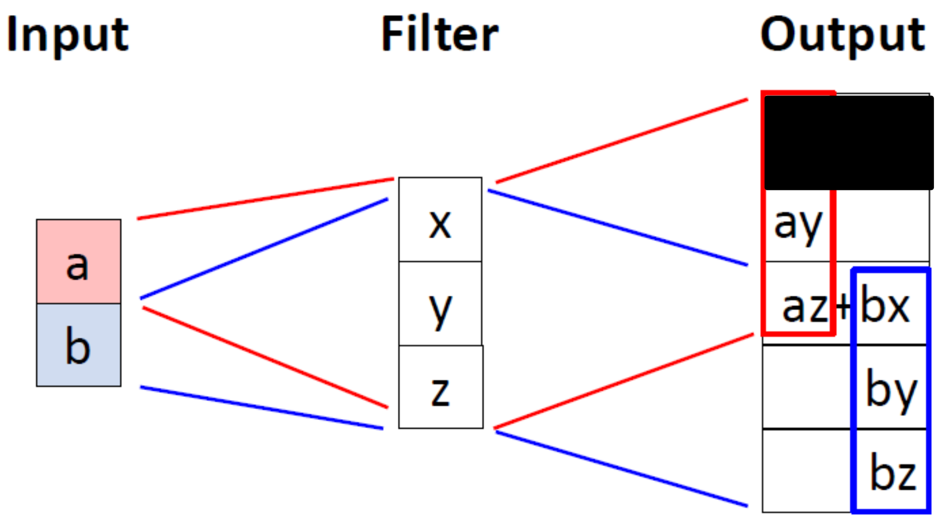


Input: 2 x 2

Output: 4 x 4

Transposed Convolution 1D Example

- Fig shows a 1D example of size 3 filter, stride 2:
  - Output has copies of filter weighted by input
  - Move 2 pixels in output for each pixel in input
  - Sum at overlaps (az+bx)
  - Crop one pixel (either top or bottom) to make output 2x input
- The filter moves at a slower pace than with unit stride
- It has many names: Transposed Convolution, Deconvolution, Upconvolution, Fractionally-strided convolution



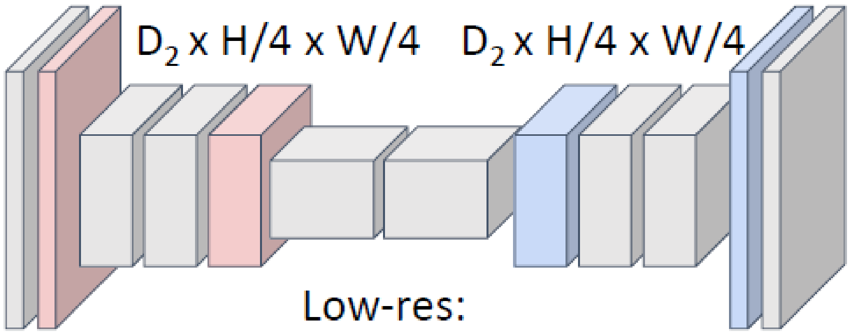
Semantic Segmentation: Fully Convolutional Network

Downsampling: Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling: linterpolation, transposed conv

Input: 3 x H x W



High-res: D_1 x H/2 x W/2

Med-res: D_2 x H/4 x W/4

Low-res: D_3 x H/4 x W/4

Med-res: D_2 x H/4 x W/4

High-res: D_1 x H/2 x W/2

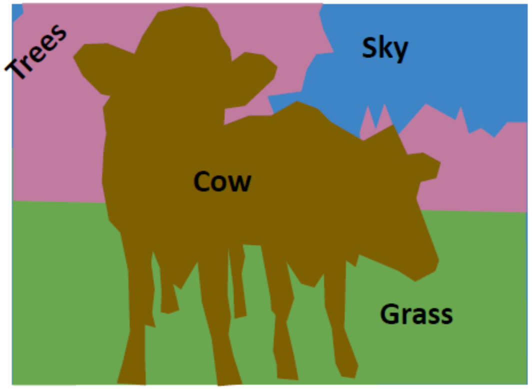
Predictions: H x W

Loss function: Per-Pixel cross-entropy

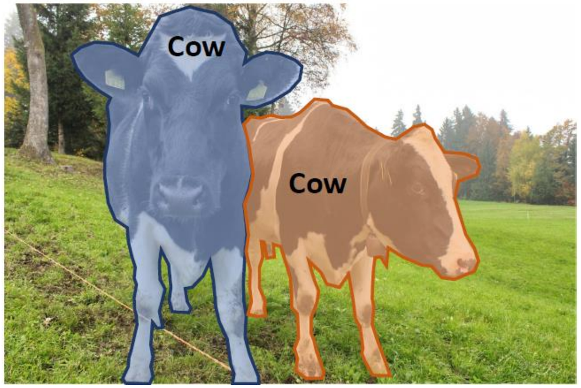
Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Types of Segmentation Tasks

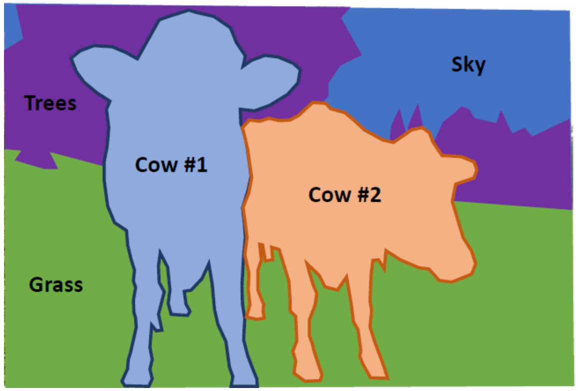
- Things vs. stuff
  - Things: Object categories that can be separated into object instances (e.g. cats, cars, person)
  - Stuff: Object categories that cannot be separated into instances (e.g. sky, grass, water, trees)
- Object Detection vs. Semantic Segmentation vs. Instance Segmentation
  - Object Detection: Detects object instances, but only gives Bbox (things only)
  - Semantic Segmentation: Label all pixels, but merges instances (both things and stuff)
  - Instance Segmentation: Detect all object instances and label the pixels that belong to each object (things only)
    - Approach: Perform object detection, then predict a segmentation mask for each object
  - Panoptic Segmentation: In addition to Instance Segmentation, also label the pixels that belong to each thing



Semantic Segmentation



Instance Segmentation

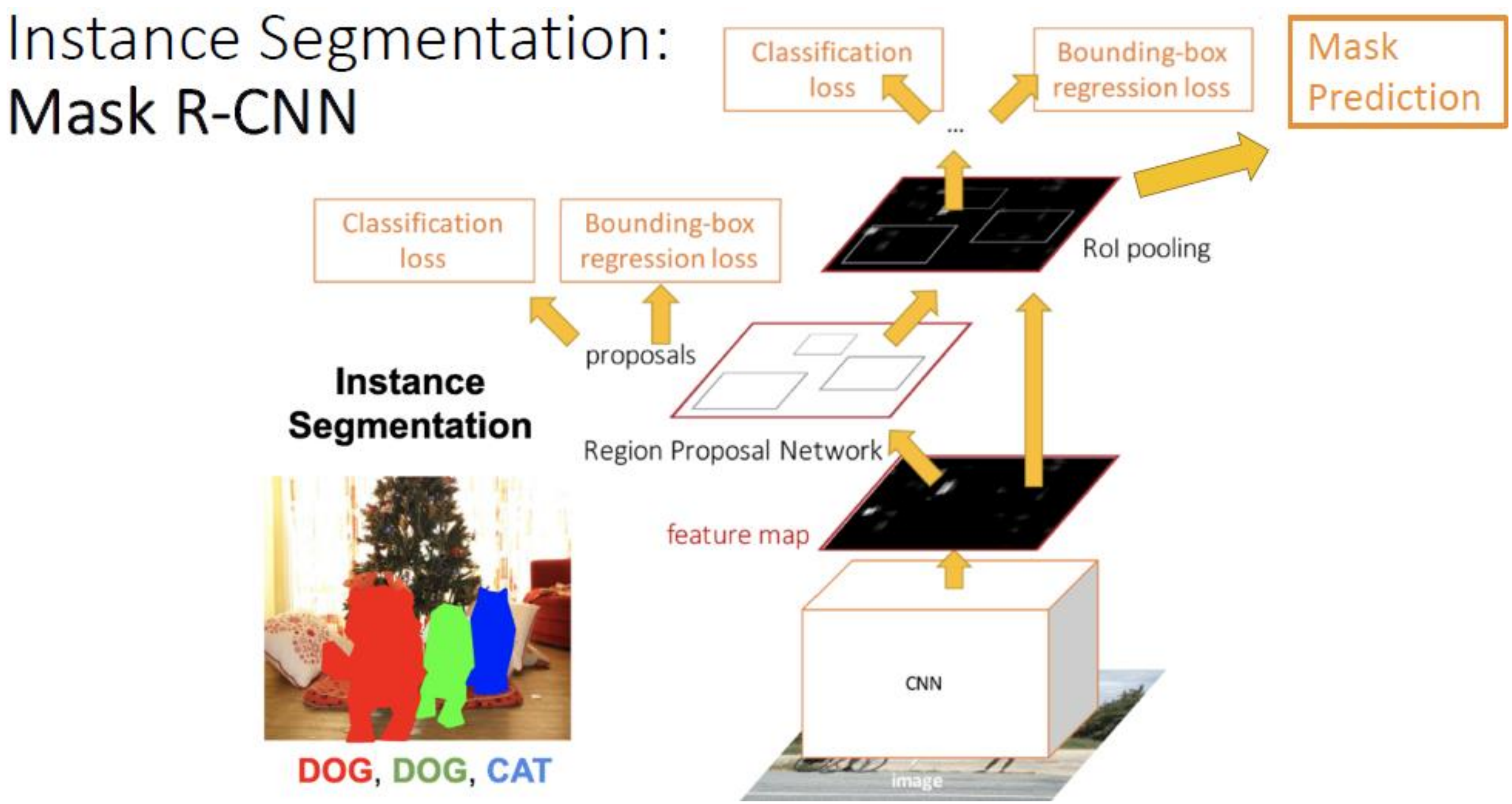


Panoptic Segmentation

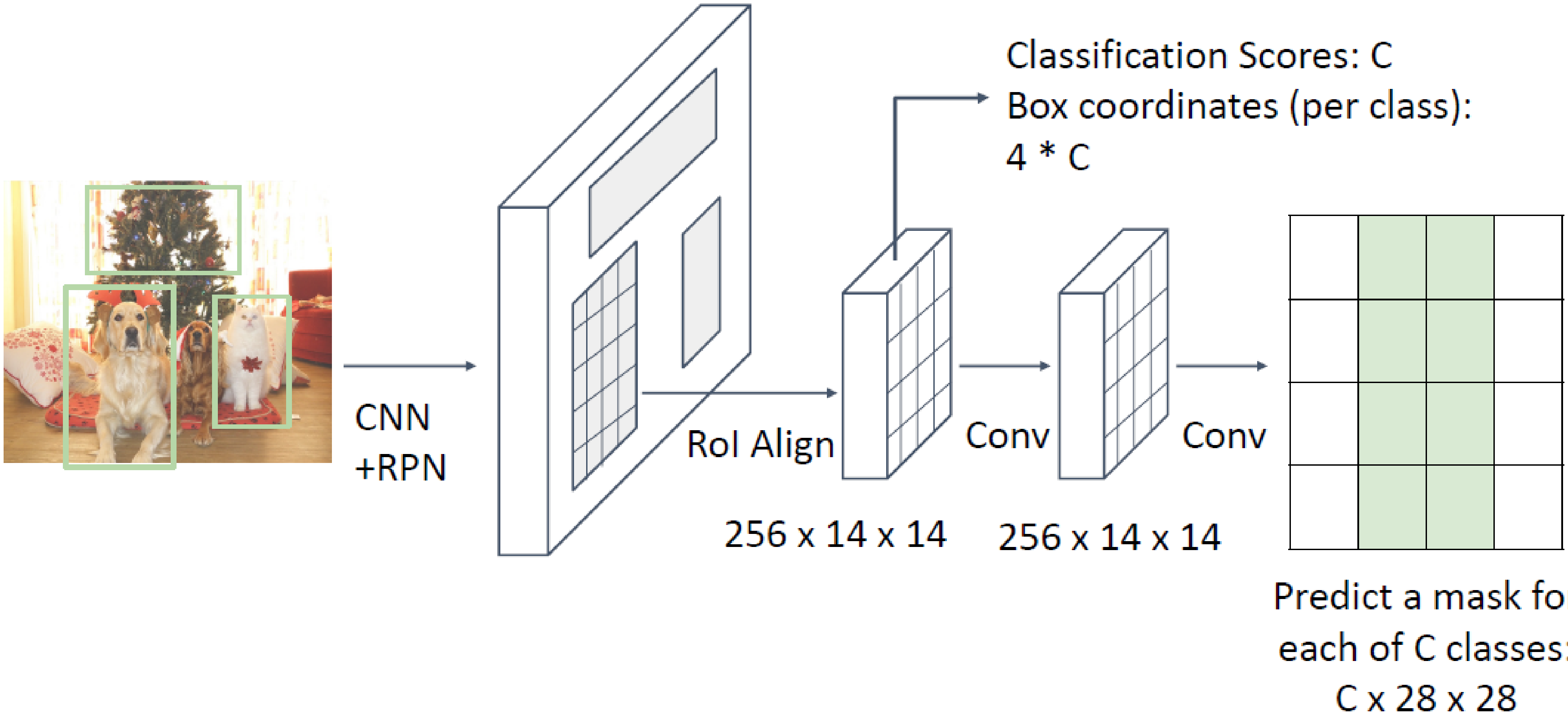
Important

Mask R-CNN for Instance Segmentation

- Add an extra "Mask Prediction" head on top of Faster R-CNN for Object Detection

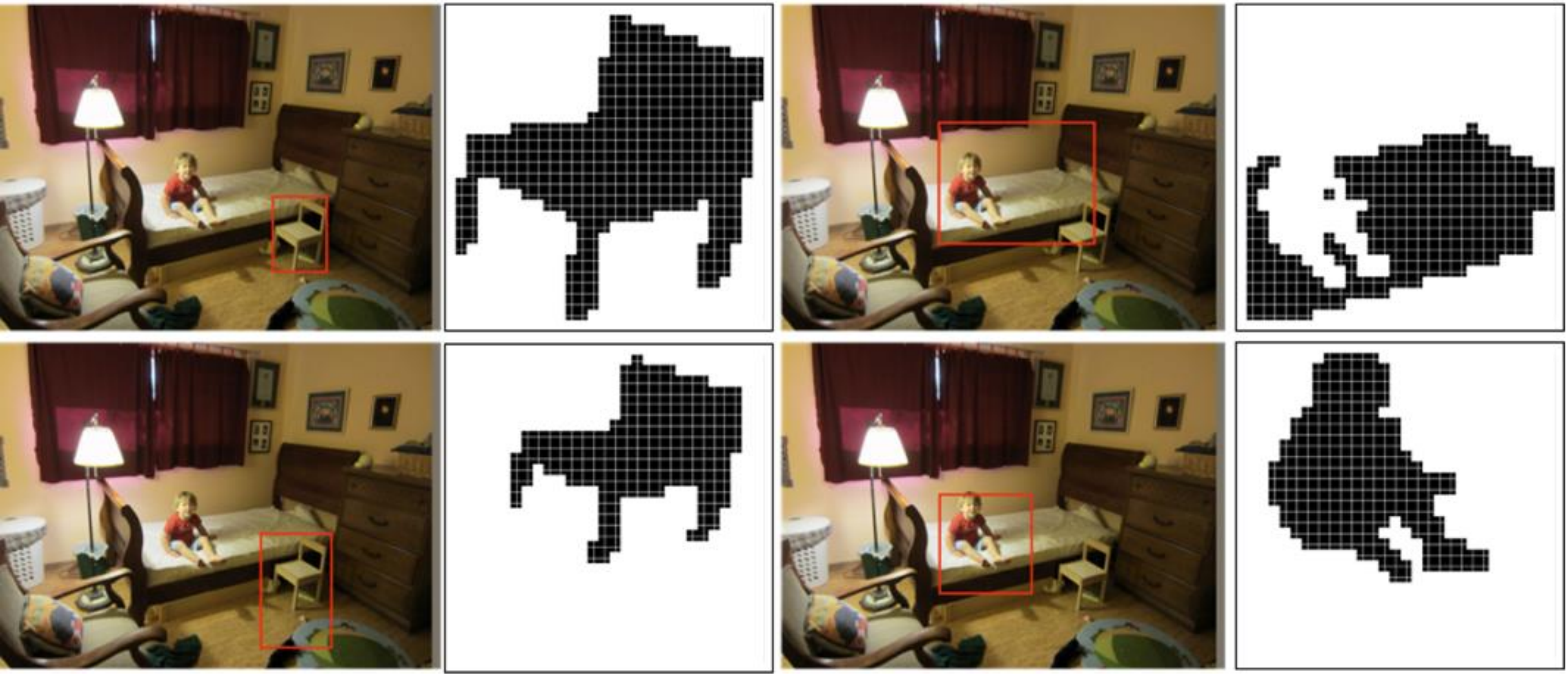


Mask R-CNN for Instance Segmentation



He et al, "Mask R-CNN", ICCV 2017

Example Target Segmentation Masks

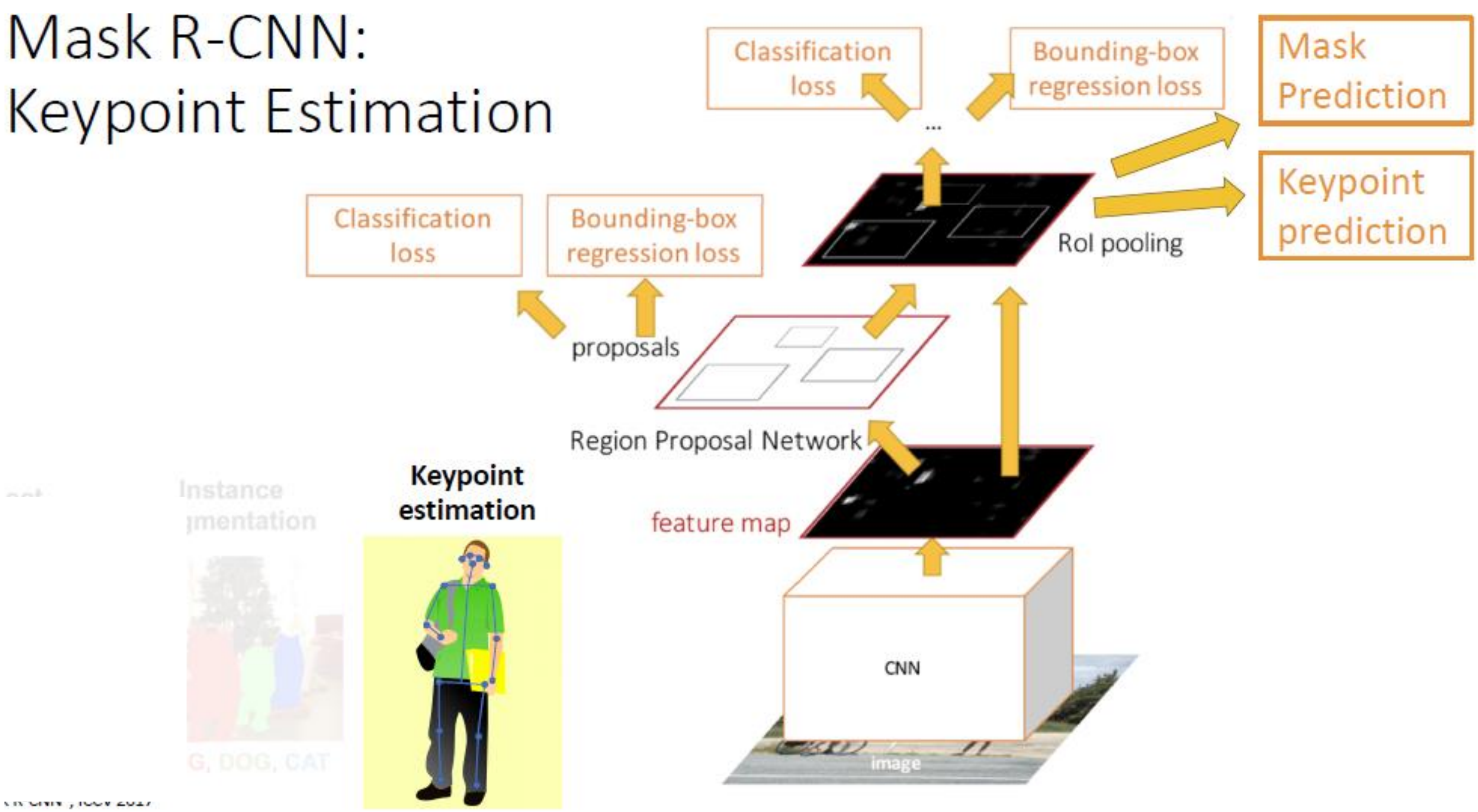


Target segmentation mask for class “chair” in the Bbox

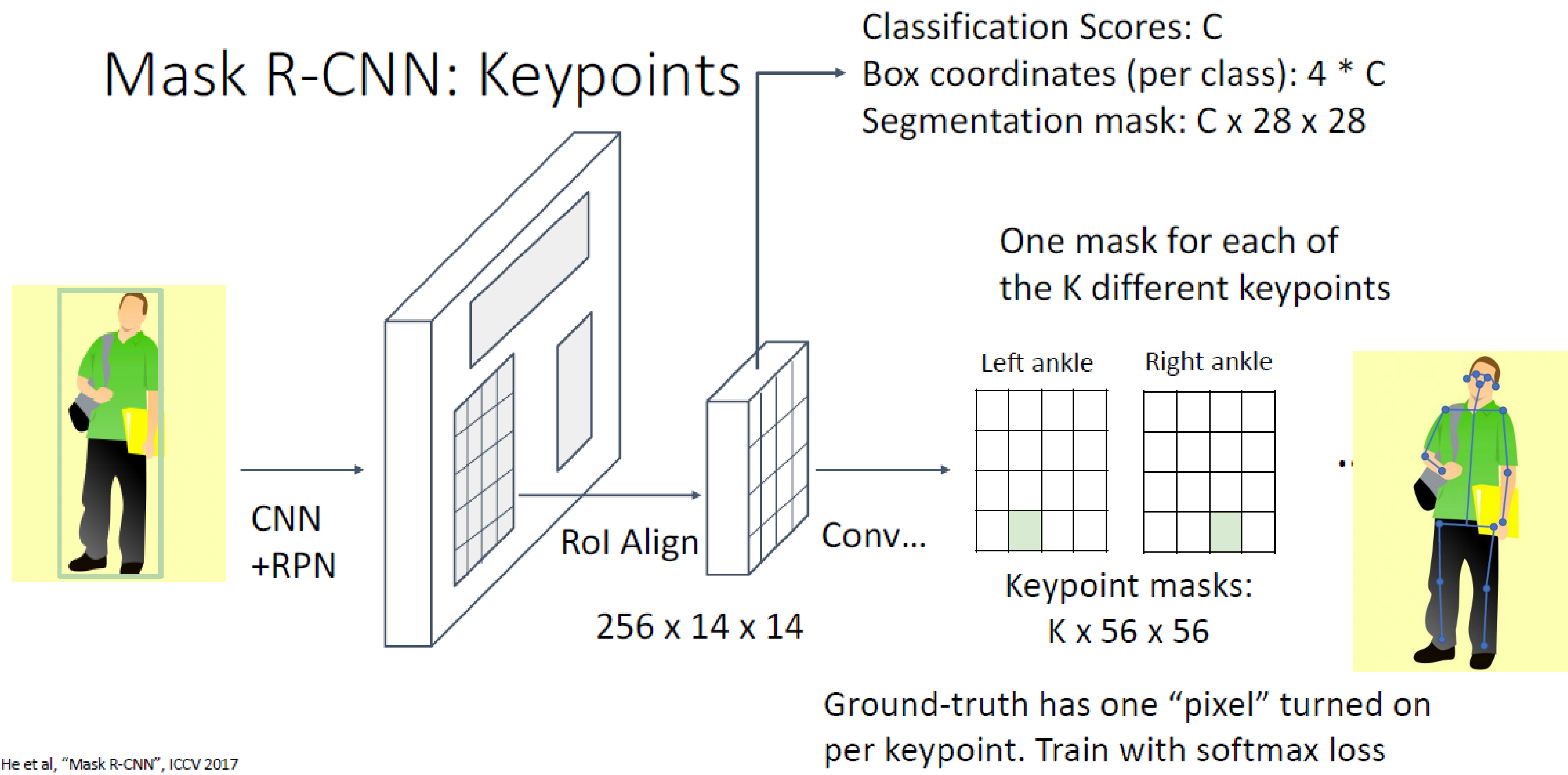
Target segmentation mask for class “person” in the Bbox

Mask R-CNN for Keypoint Estimation

- Add an extra “Keypoint Prediction” head to perform joint Instance Segmentation and Pose Estimation
  - Example keypoints: left/right shoulder, elbow, wrist, hip, knee, ankle…



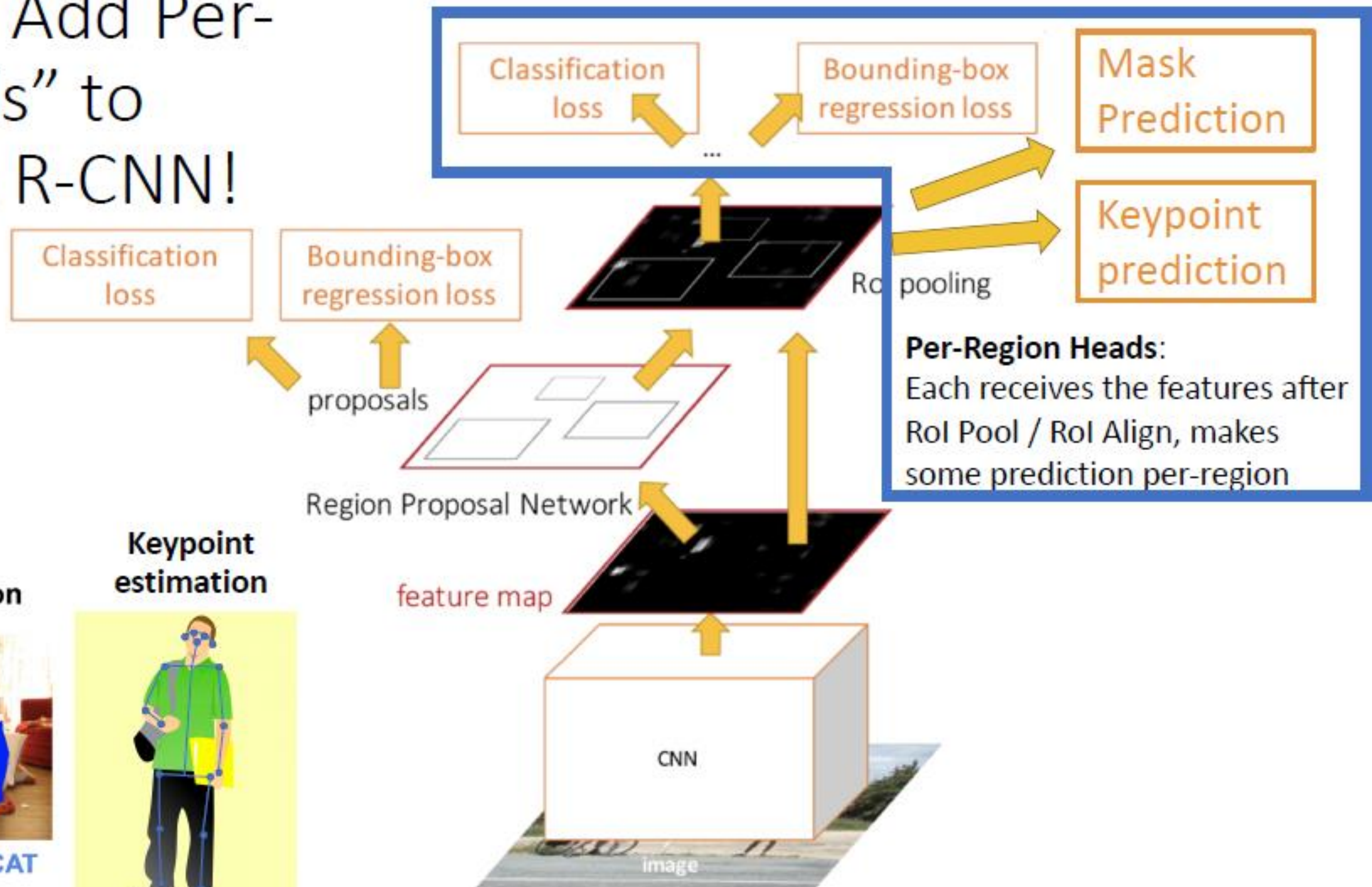
Mask R-CNN: Keypoints

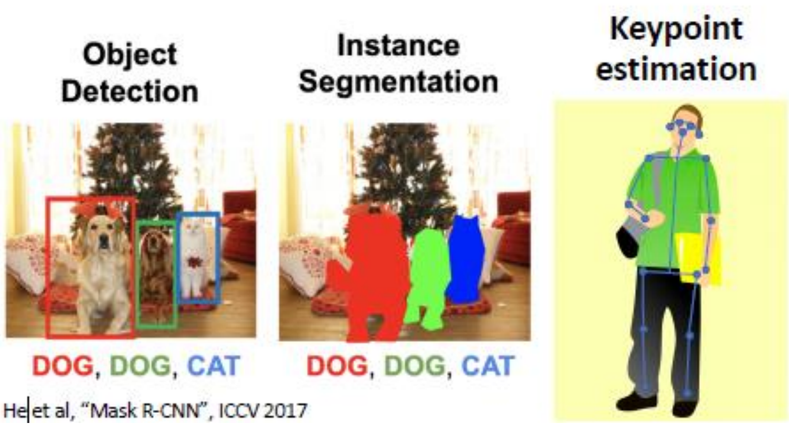


Important

Summary of Per-Region Heads for Different Tasks

General Idea: Add Per-Region "Heads" to Faster / Mask R-CNN!





He et al, "Mask R-CNN", ICCV 2017