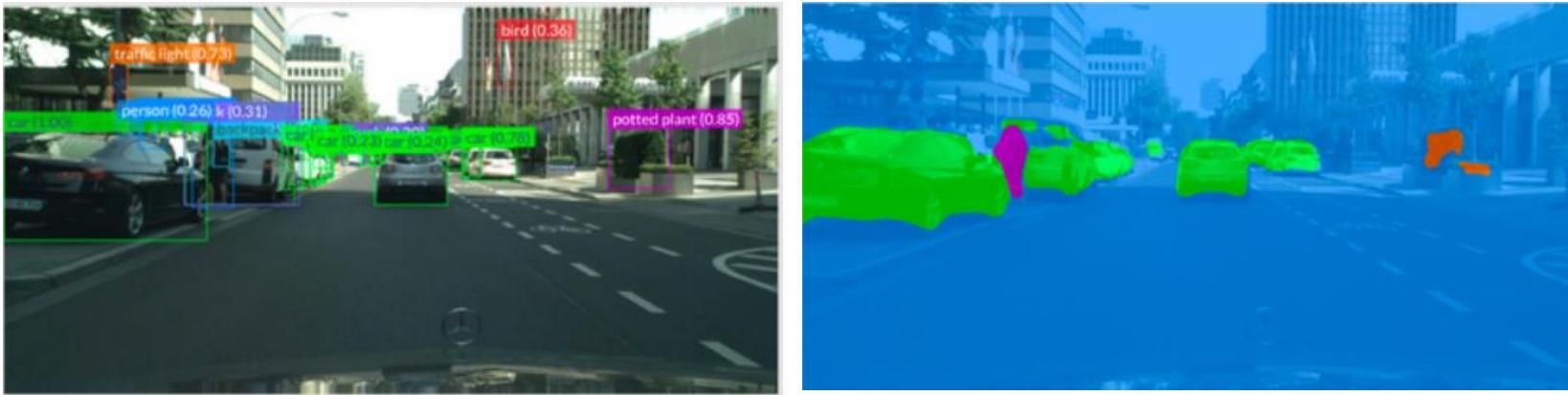
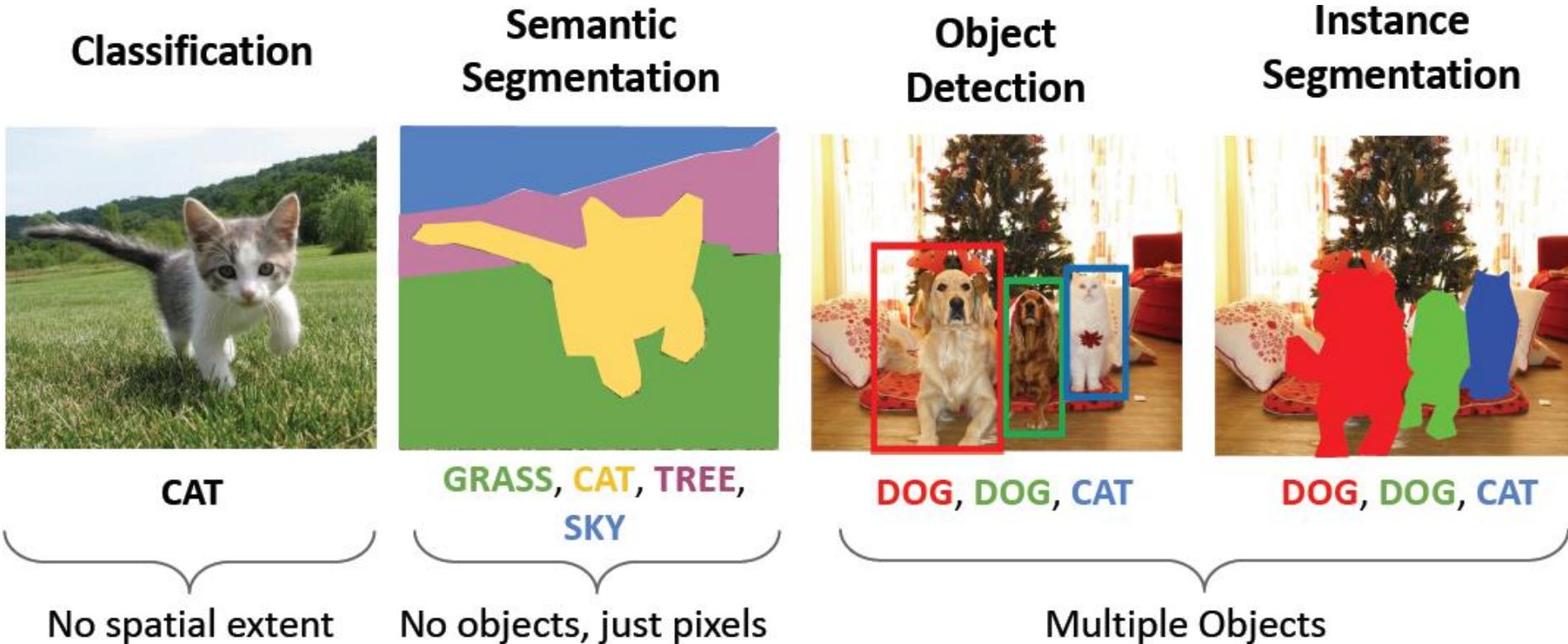


L6 Control



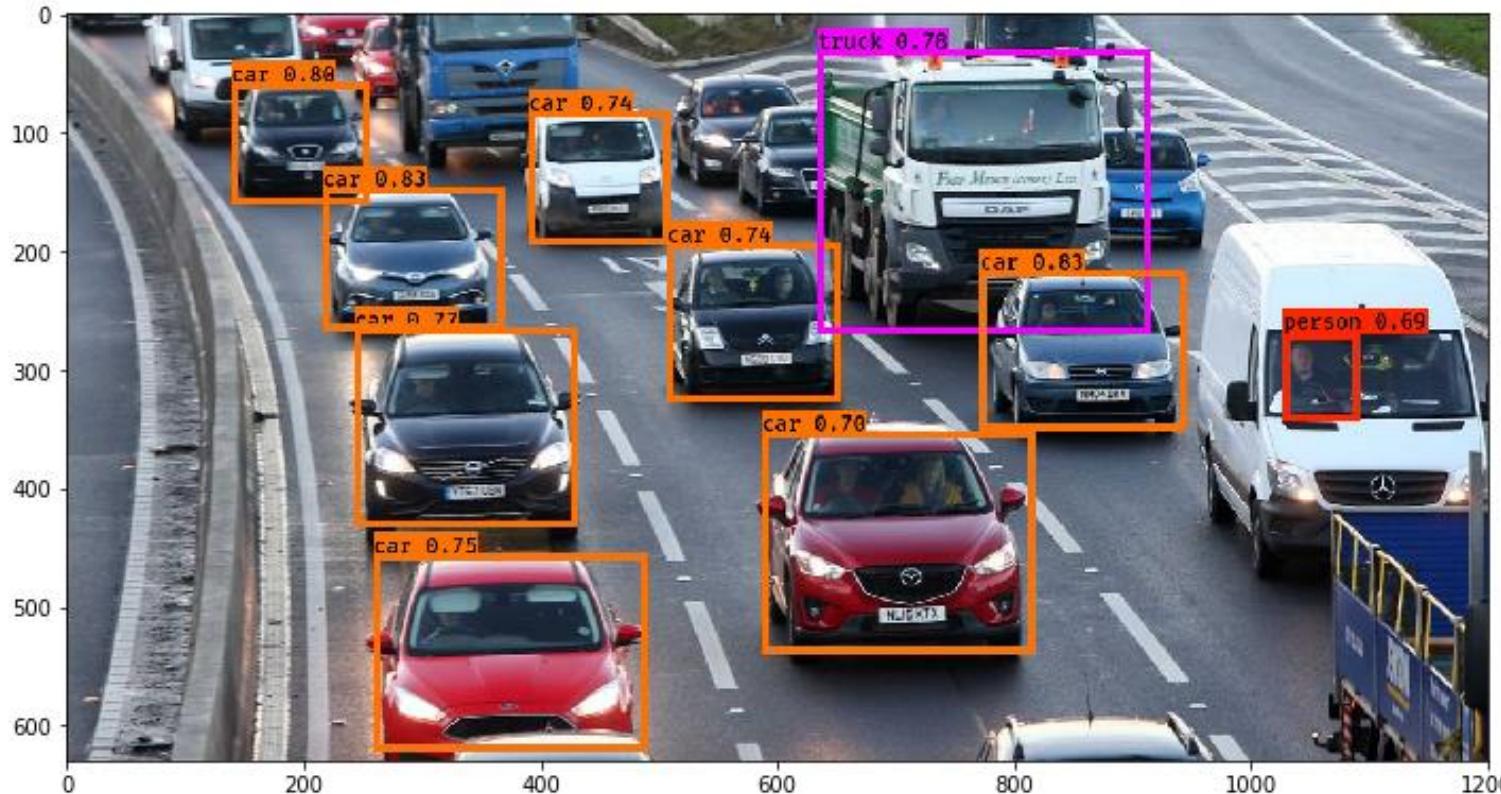
Zonghua Gu, Umeå University
Nov. 2023

Computer Vision Tasks



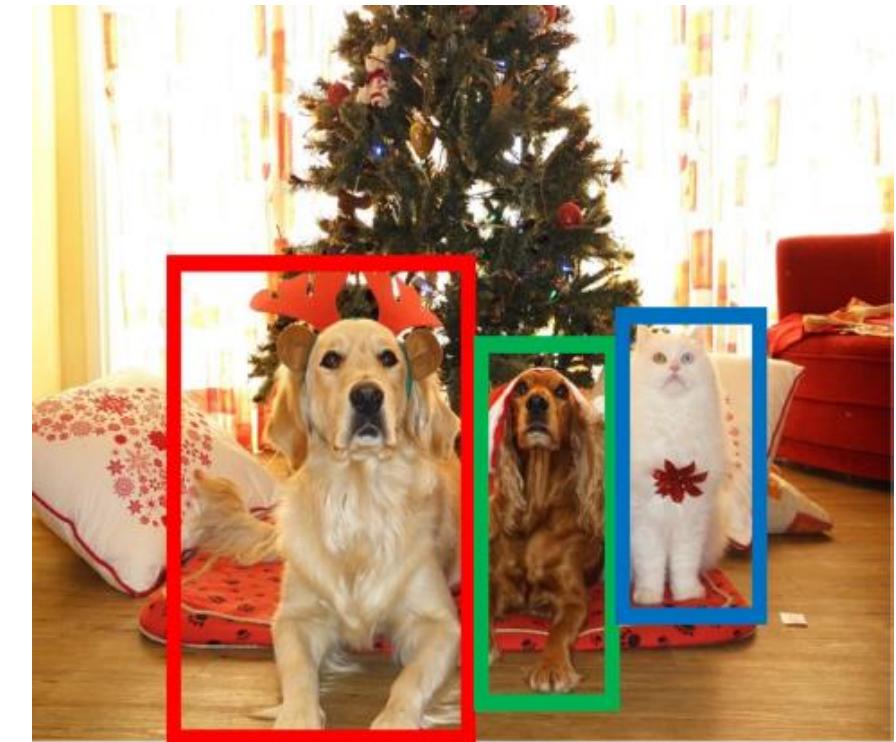
Outline

- Object detection
- Segmentation



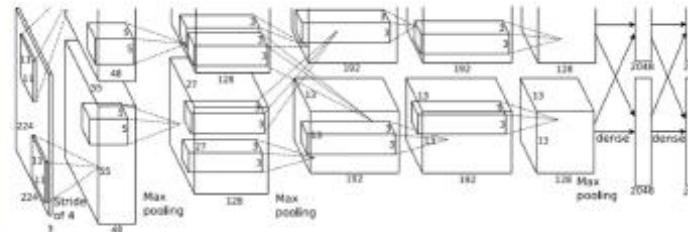
Object Detection: Task Definition

- 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

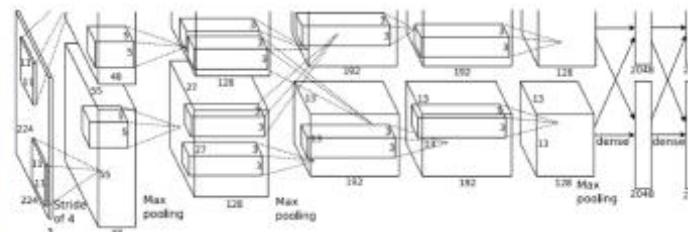


Multi-Object Detection

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



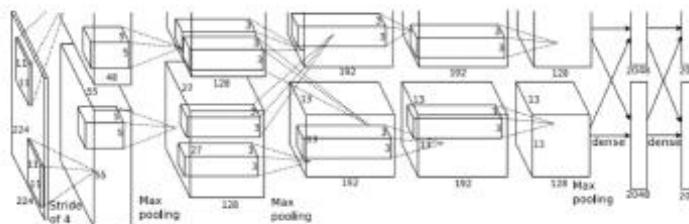
CAT: (x, y, w, h)



DOG: (x, y, w, h)

DOG: (x, y, w, h)

CAT: (x, y, w, h)



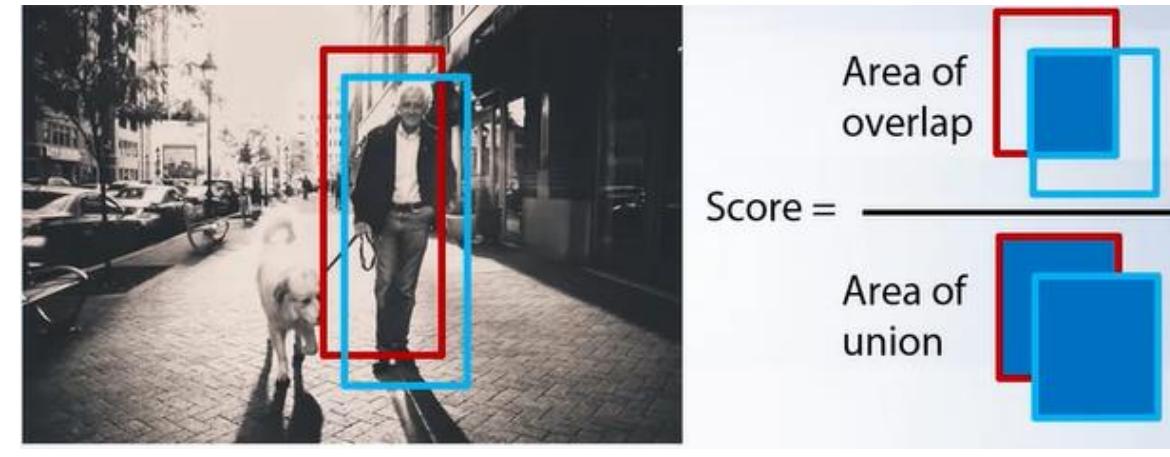
DUCK: (x, y, w, h)

DUCK: (x, y, w, h)

....

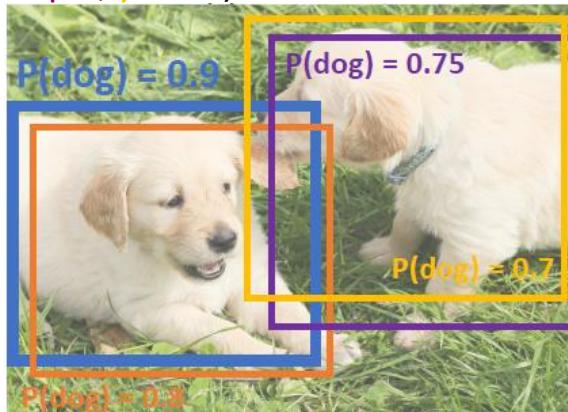
Detection Criteria (Intersection Over Union, IOU)

- Blue box: Ground Truth; Red box: model output
- Set a threshold for detection (positive result)
 $\text{IOU}(B_{\text{GT}}, B_{\text{Pred}}) \geq \theta_{IoU}$
 - Common threshold $\theta_{IoU} = 0.5$



Non-Max Suppression (NMS)

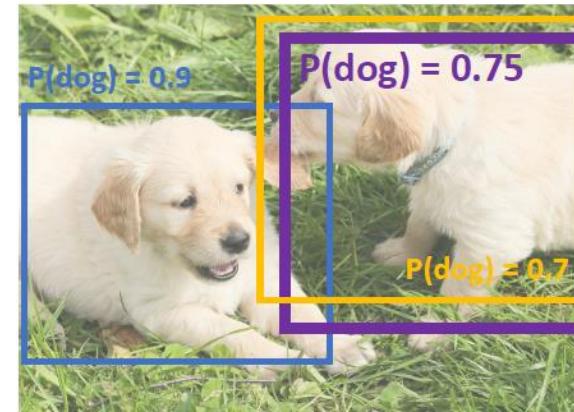
- Problem: Object detectors often output many overlapping detections
- NMS: Discard (suppresses) overlapping object boxes except the one with the maximum classification score
- For each output class
 - 1 Select next highest-scoring box b and output it as a prediction
 - 1 Discard any remaining boxes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



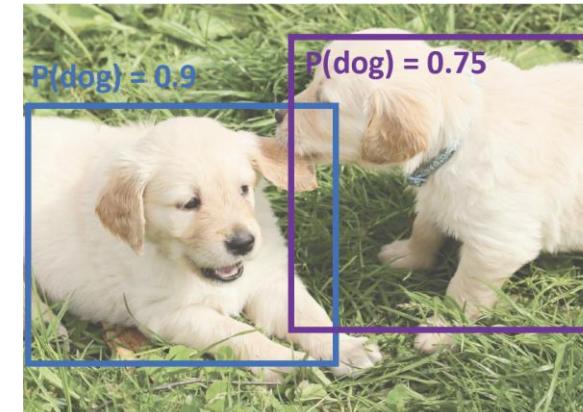
$$\text{IoU}(\blacksquare, \blacksquare) = 0.78$$

$$\text{IoU}(\blacksquare, \blacksquare) = 0.05$$

$$\text{IoU}(\blacksquare, \blacksquare) = 0.07$$

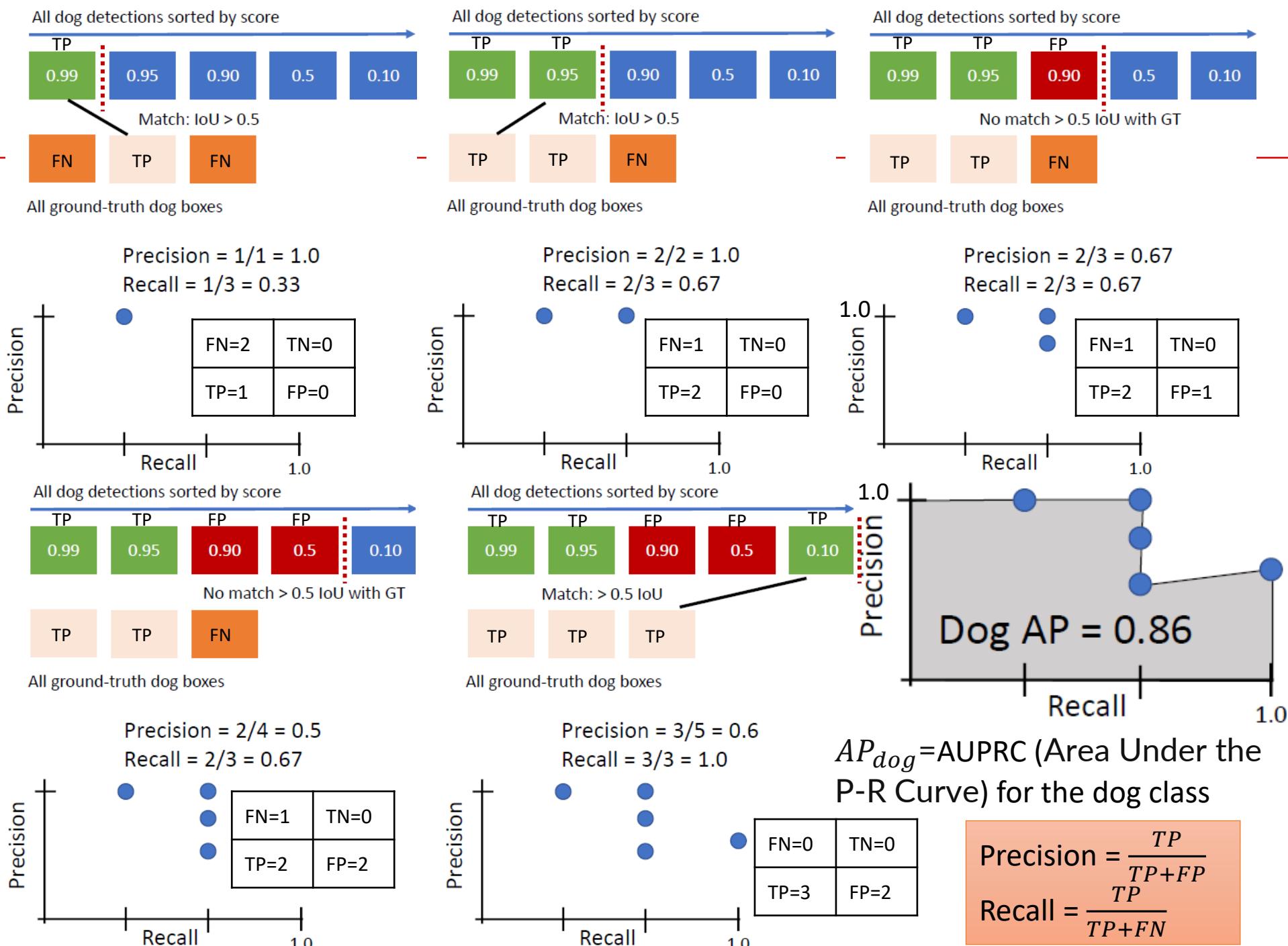


$$\text{IoU}(\blacksquare, \blacksquare) = 0.74$$



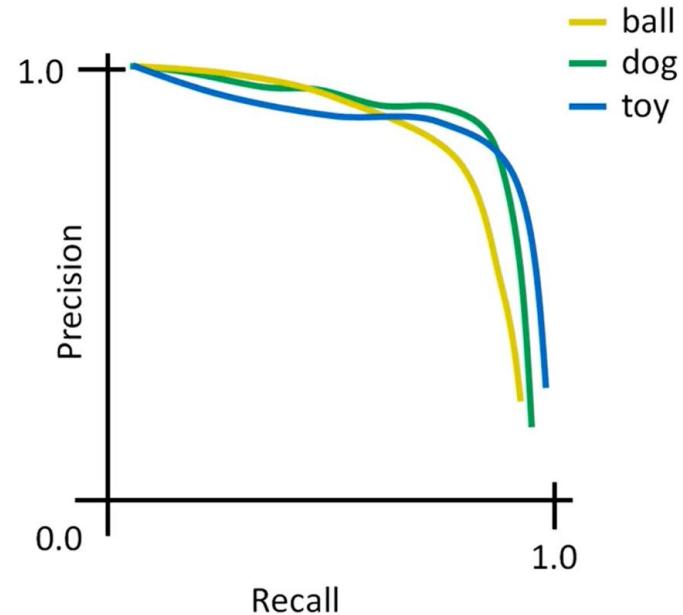
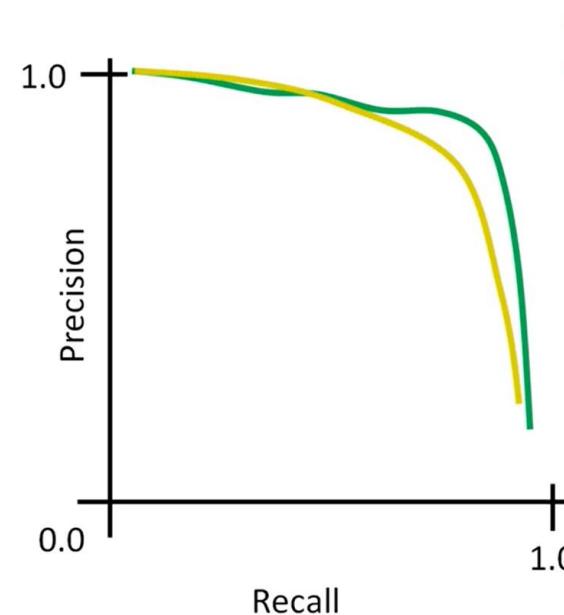
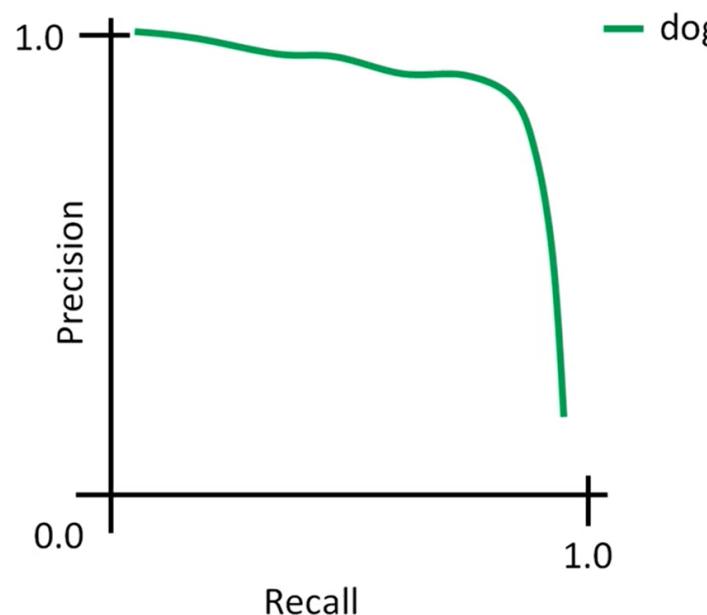
Evaluating Object Detectors: Mean Average Precision (mAP)

- 1. Run object detector on all test images (with NMS)
- 2. For each class, compute Average Precision (AP)
 - 1. For each detection with score > Conf. Score threshold, ranked from high to low score
 - 1. If it matches some GT box with IoU > thresh, mark it as TP (True Positive) and eliminate the GT
 - 2. Otherwise mark it as FP (False Positive)
 - 3. Plot a point on Precision-Recall (PR) Curve
 - 2. Average Precision (AP) for each class, e.g., $AP_{dog} = \text{AUPRC}$ (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
- Ref: Object Detection Performance Metrics
 - <https://www.coursera.org/learn/computer-vision-with-embedded-machine-learning/lecture/zDlgp/object-detection-performance-metrics>



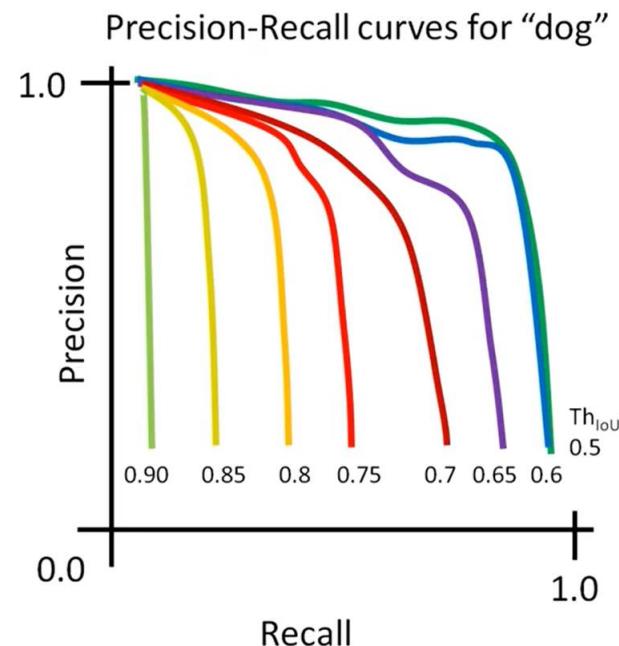
mean Average Precision (mAP)

- 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)
 - $mAP_{0.5} = \frac{1}{3}(AP_{\text{dog}} + AP_{\text{ball}} + AP_{\text{toy}})$



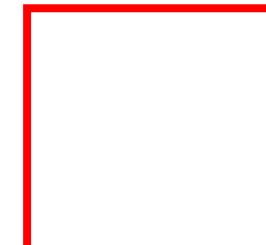
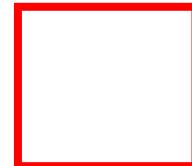
mAP

- 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 ($\text{IoU} \geq \text{threshold}$), hence precision starts at 1. If 0 match (all boxes have $\text{IoU} < \text{threshold}$), 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



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

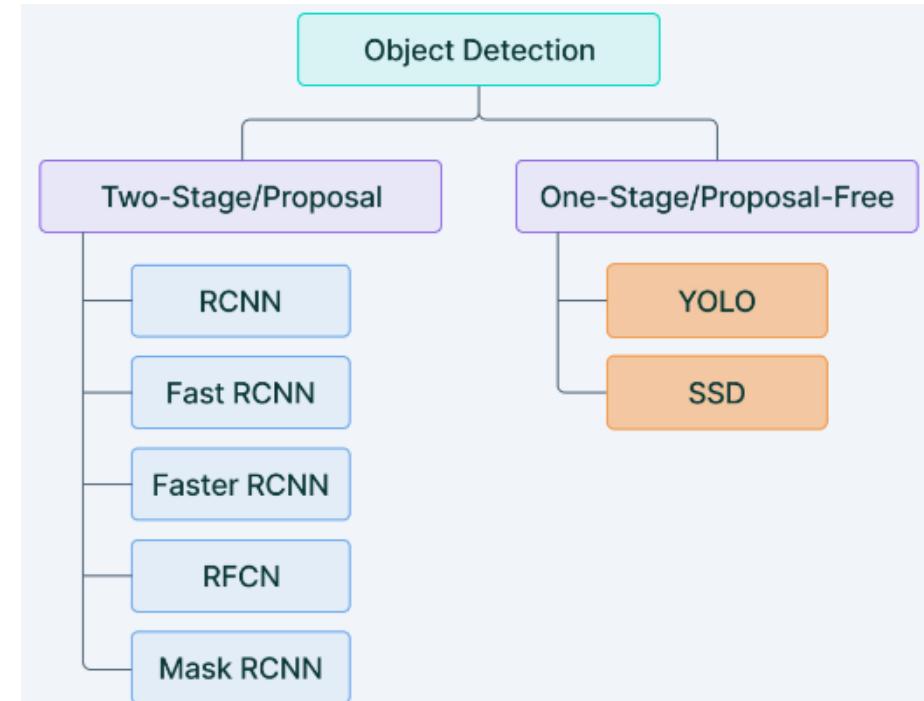
Two-stage vs. One-Stage Detector

- **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

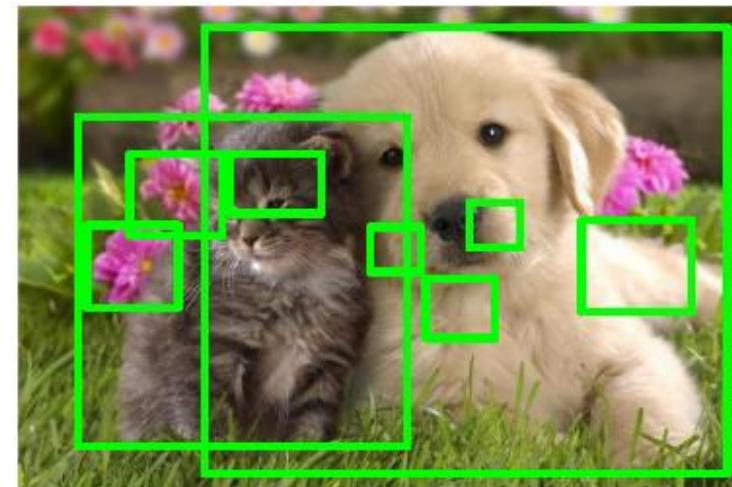
- **One-stage detector**

- Directly perform object detection, incl. classification and regression of Bboxes of the objects



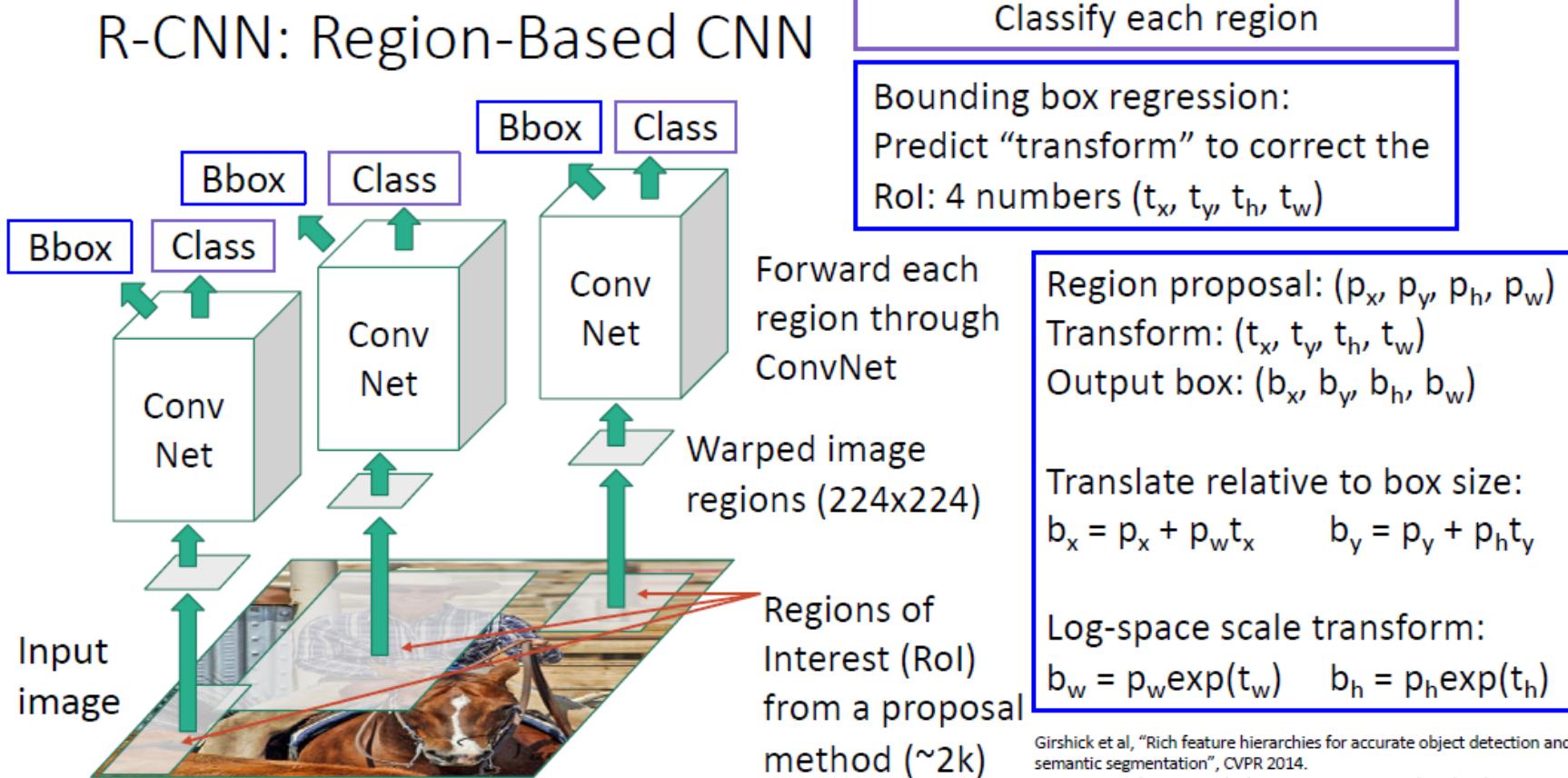
Region Proposals

- 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



R-CNN: Training Time

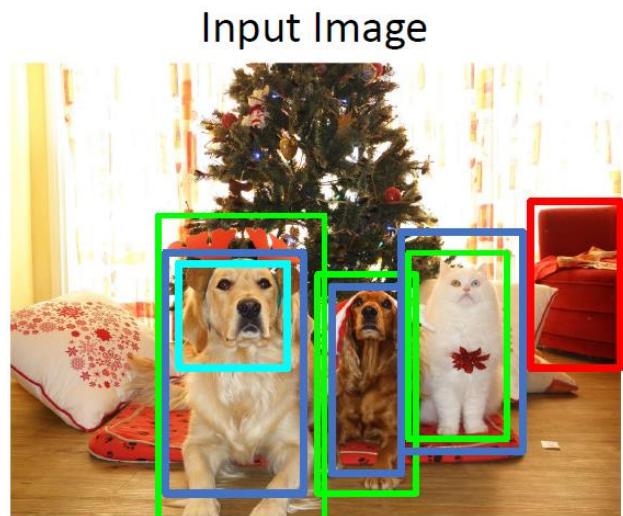
- Crop/warp each region proposal into same-size (e.g., 224×224) image regions, and run each through a CNN to get Bbox and class label for each region
- Bbox regression: transform each region proposal with learnable parameters (t_x, t_y, t_h, t_w) into a better Bbox



R-CNN Training Example

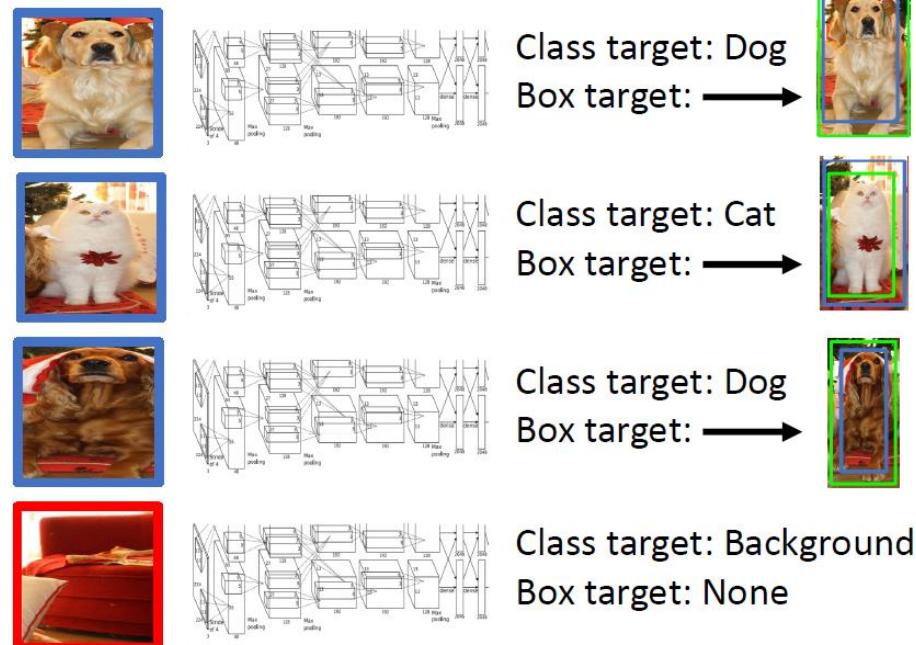
- Categorize each region proposal as positive, negative, or neutral based on overlap with ground-truth boxes
- Crop pixels from each positive and negative proposal, resize to 224 x 224
- Use the CNN for Bbox regression and classification for positive boxes; only 1-class prediction for negative boxes

"Slow" R-CNN Training



GT Boxes	Positive
Neutral	Negative

Run each region through CNN. For positive boxes predict class and box offset; for negative boxes just predict background class



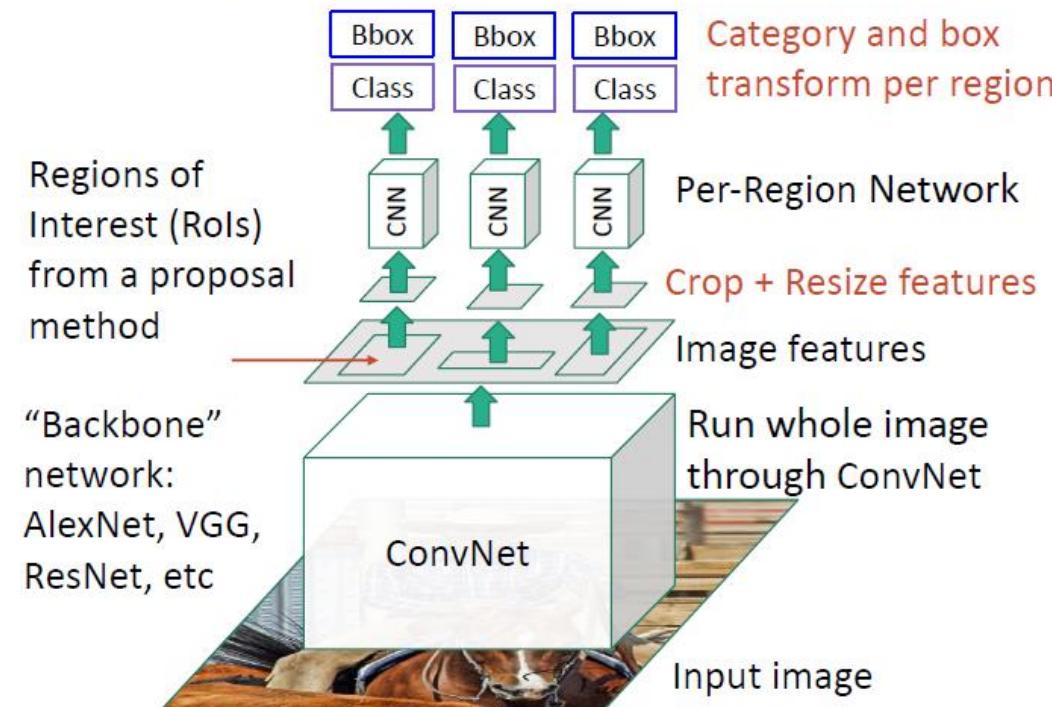
R-CNN: Test Time

- 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
- 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

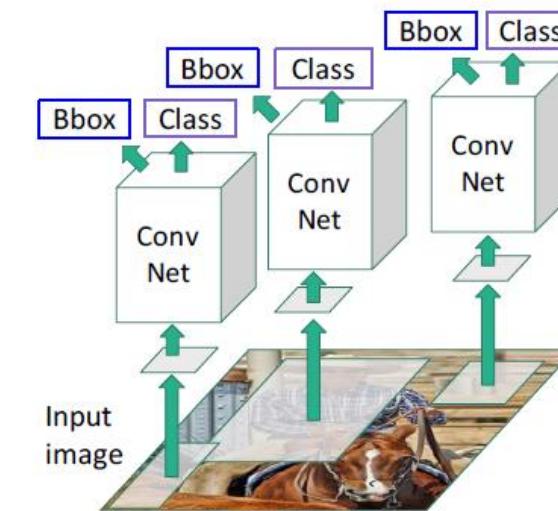
- 1. Use a backbone network to extract feature maps from the whole image
- 2. Apply Selective Search on these feature maps and get object proposals
- 3. Use a lightweight Per-Region network to perform Bbox regression and classification
- Most of the computation happens in backbone network; this saves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

Fast R-CNN

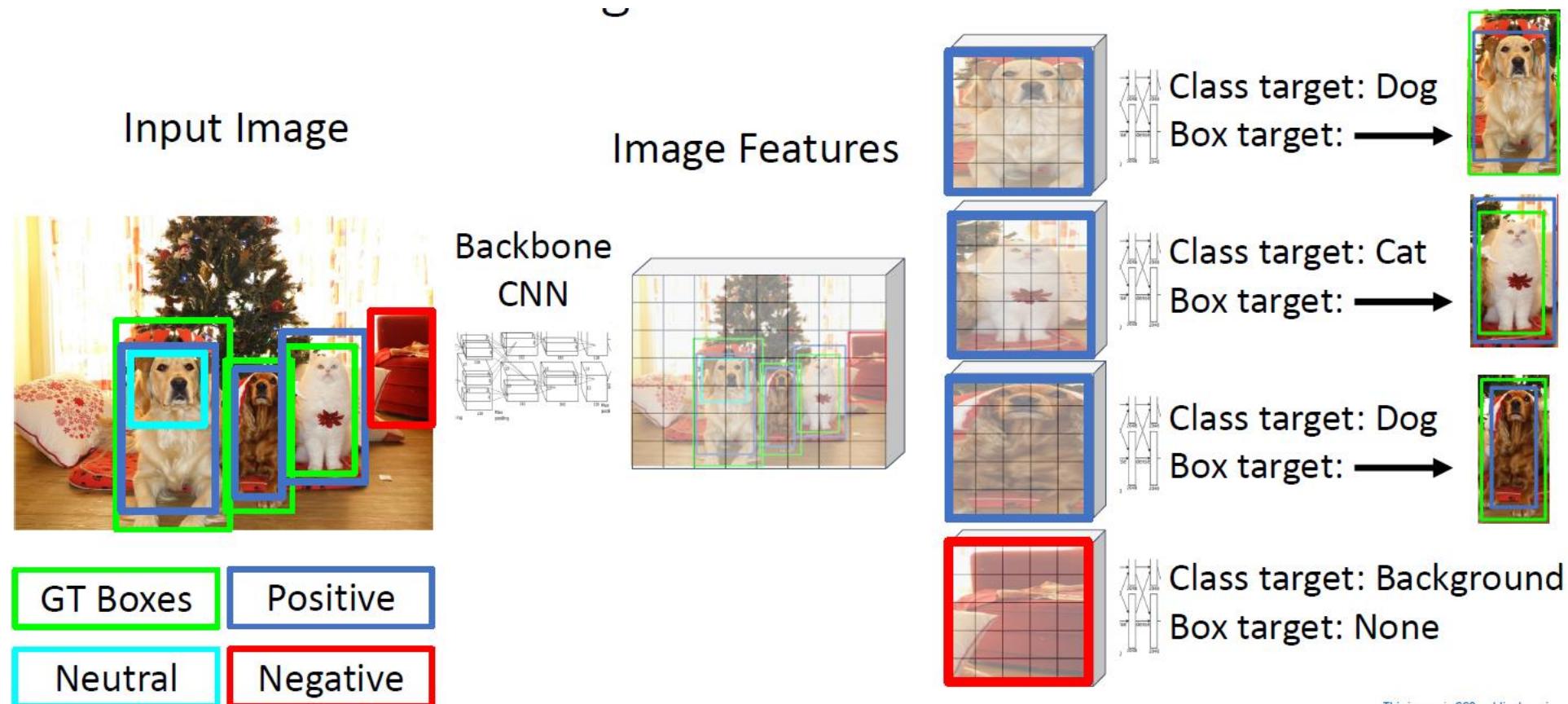


"Slow" R-CNN

Process each region independently

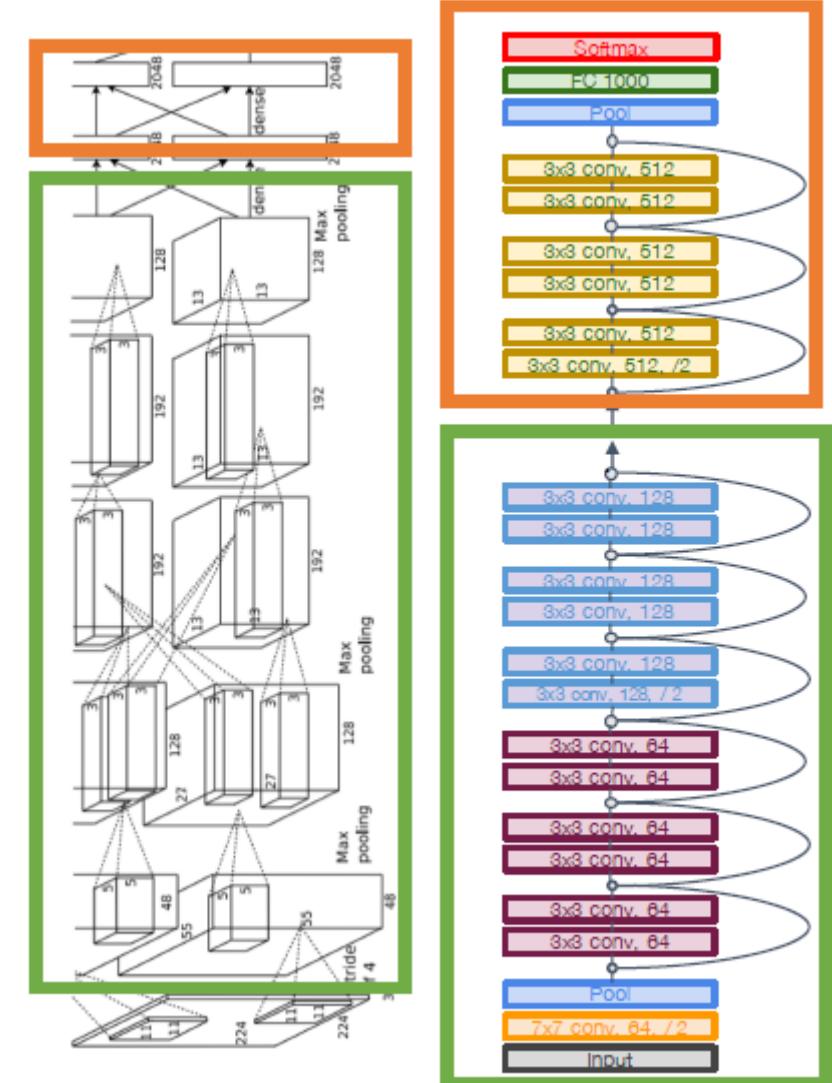


Fast R-CNN Training Example



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

Faster R-CNN

- Problem: Test time of Fast R-CNN is dominated by region proposals
- 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 a learnable CNN; Selective Search is a fixed heuristic algorithm

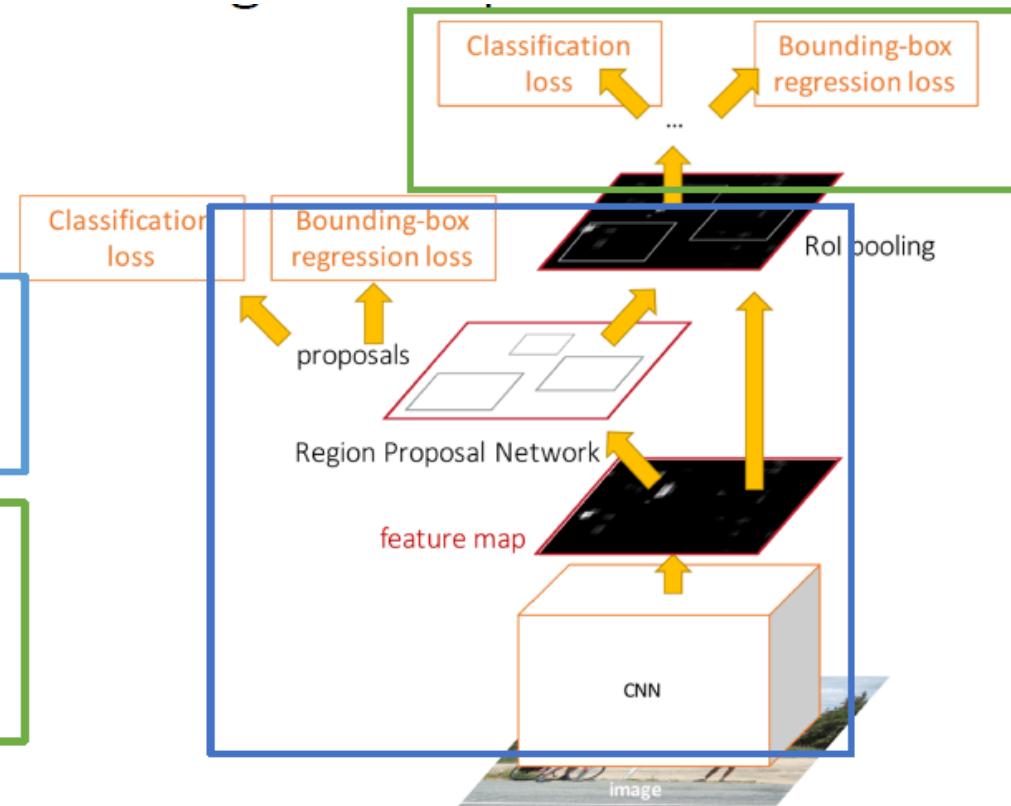
Faster R-CNN is a
Two-stage object detector

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



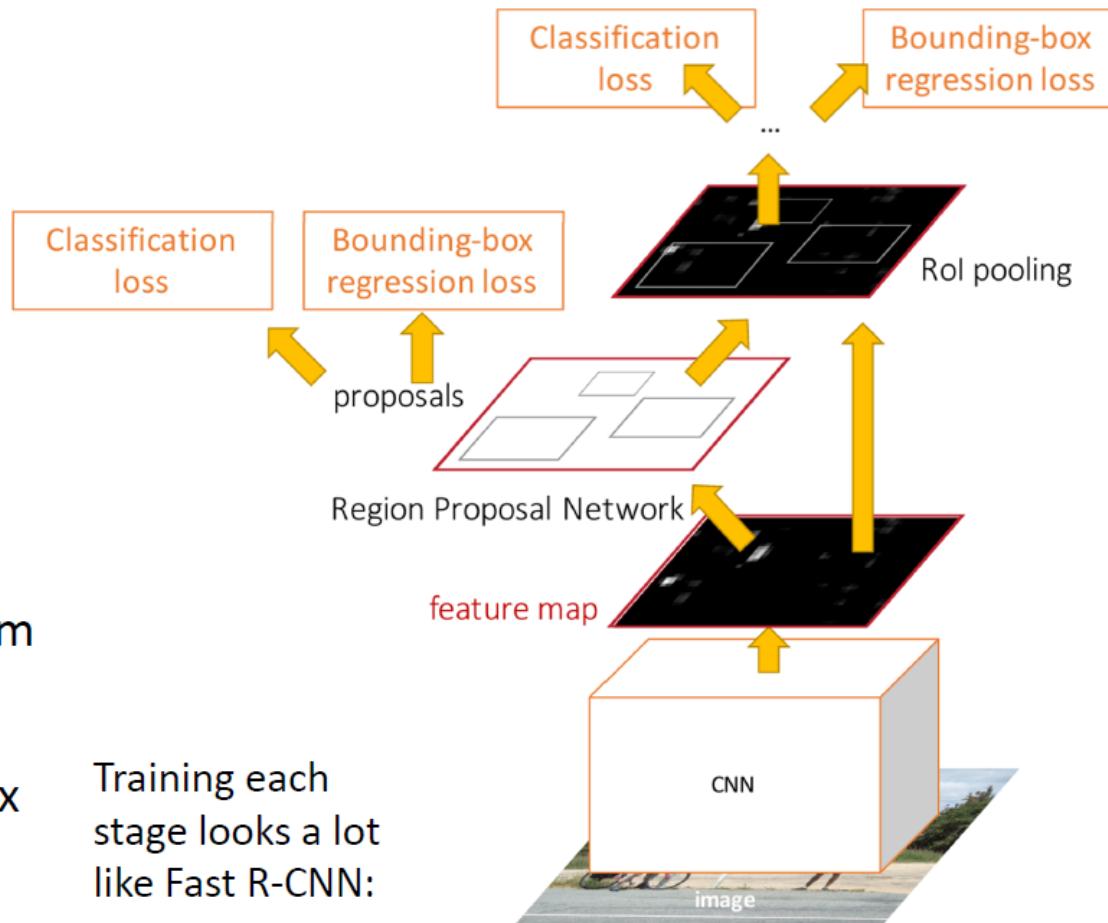
Faster R-CNN: Loss Function

Jointly train with 4 losses:

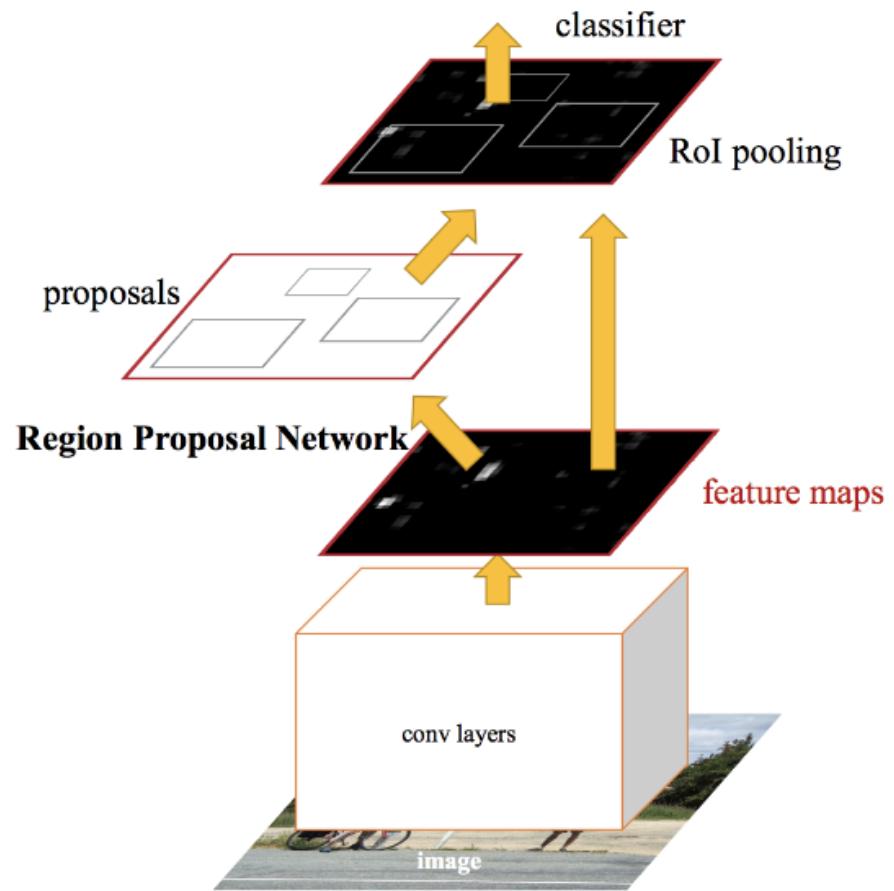
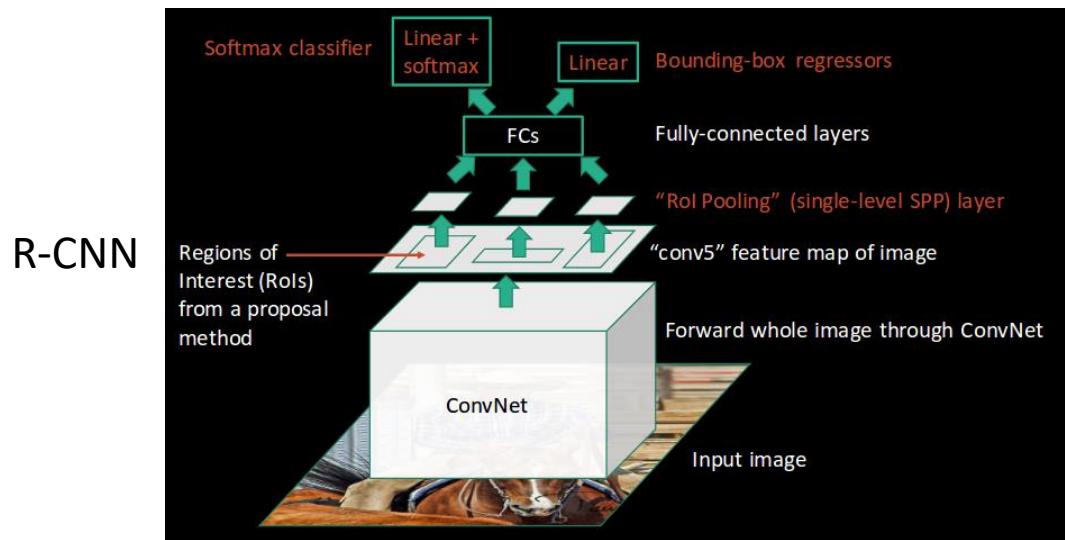
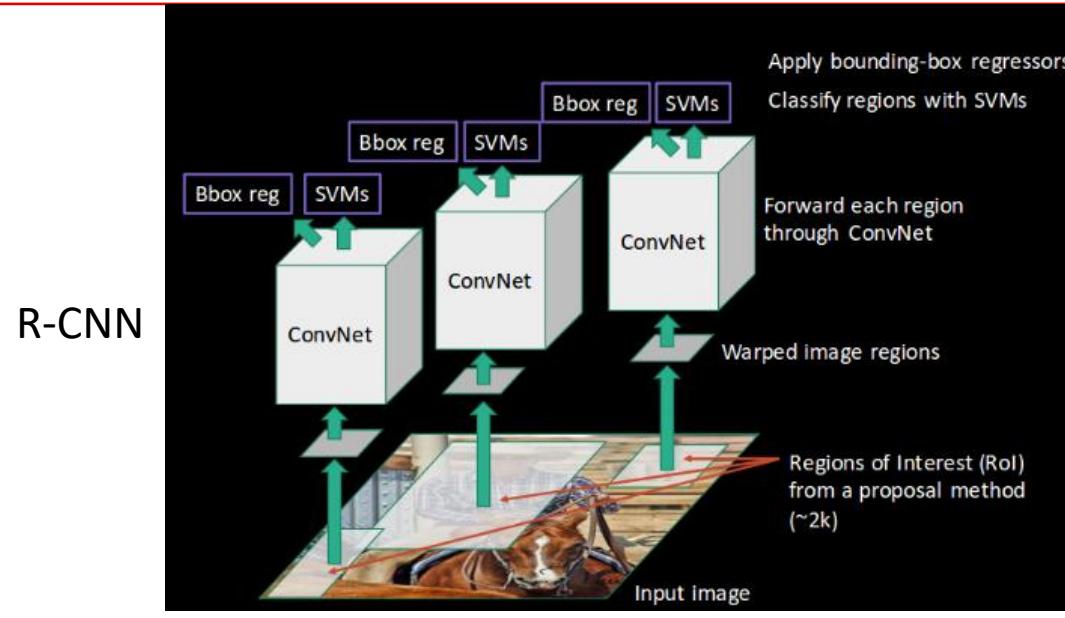
1. **RPN classification**: anchor box is object / not an object
 2. **RPN regression**: predict transform from anchor box to proposal box
 3. **Object classification**: classify proposals as background / object class
 4. **Object regression**: predict transform from proposal box to object box

Anchor -> Region Proposal -> Object Box
(Stage 1) (Stage 2)

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



3 Variants of R-CNN



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	

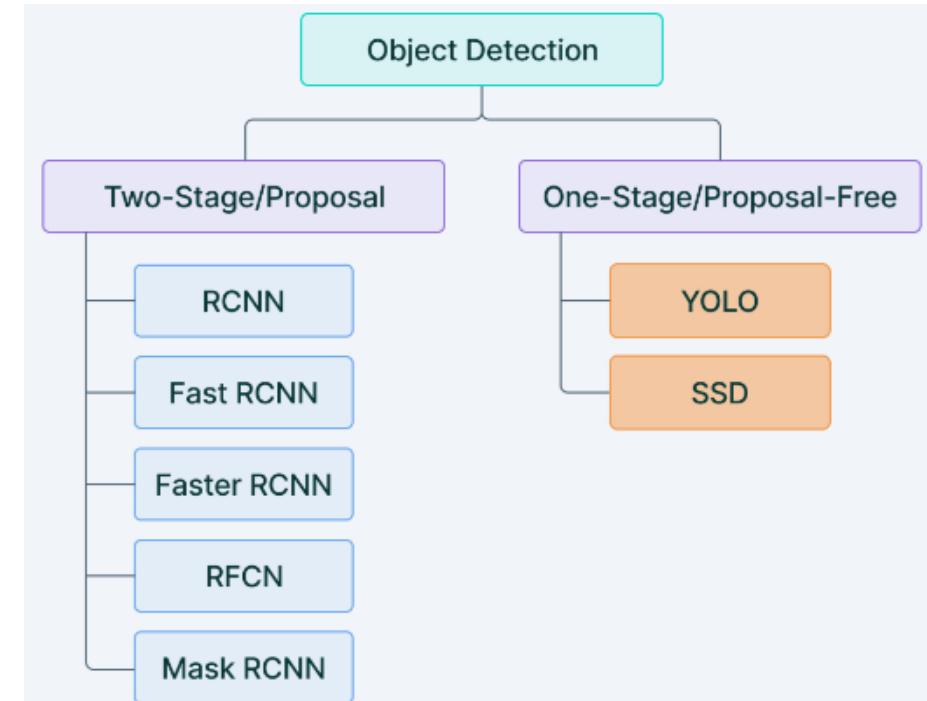
Two-stage vs. One-Stage Detector

- Two-stage detector

- 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

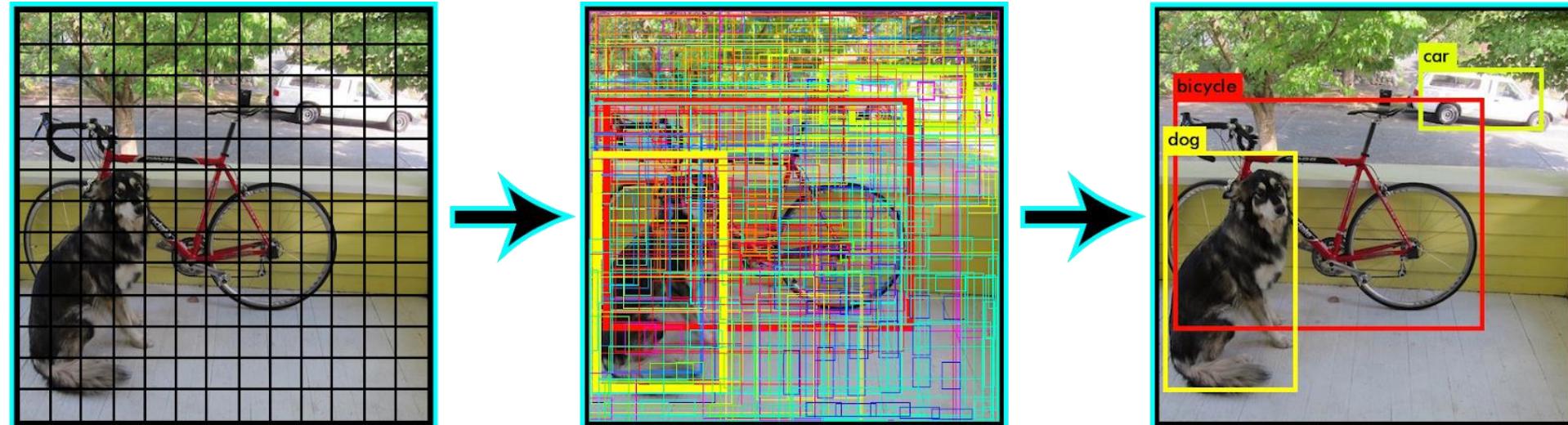
- One-stage detector

- Directly perform object detection, incl. classification and regression of Bboxes of the objects



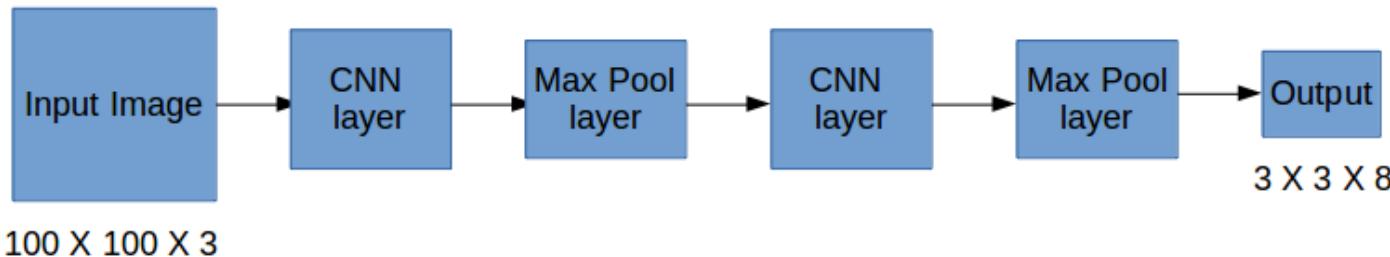
You Only Look Once (YOLO)

- 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 3×3 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 3×3 grid cells, we have an 8-D output vector. So the output dimension is $3 \times 3 \times 8$
- 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



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

Two Grid Cells

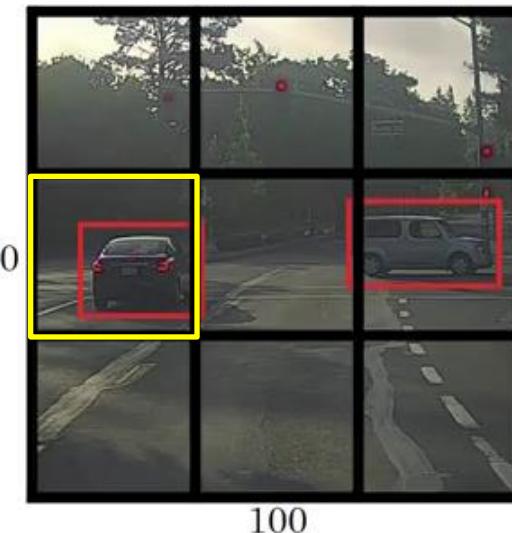
- Since there is no object in this grid, pc=0, and all the other entries are ?

$y =$	0
	?
	?
	?
	?
	?
	?
	?



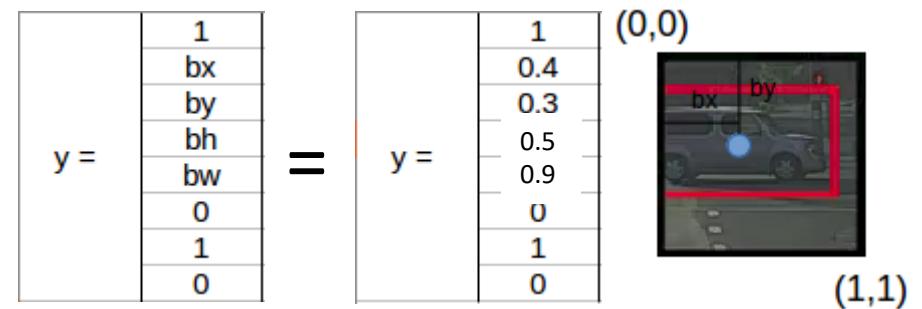
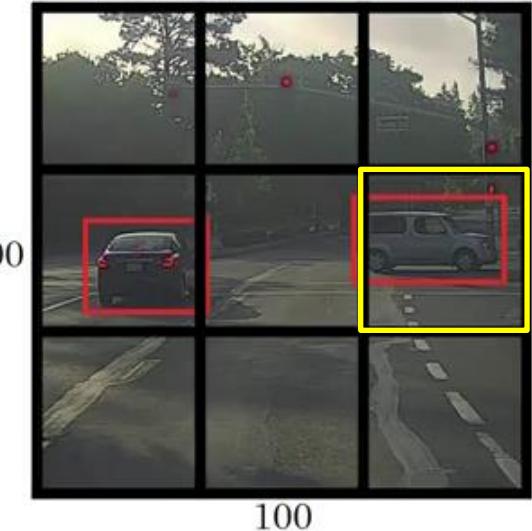
- Since there is an object (Car) in this grid, pc=1. (bx , by , bh , bw) are calculated relative to the particular grid cell
- Class label is (0,1,0) since Car is the 2nd class

$y =$	1
	bx
	by
	bh
	bw
	0
	1
	0



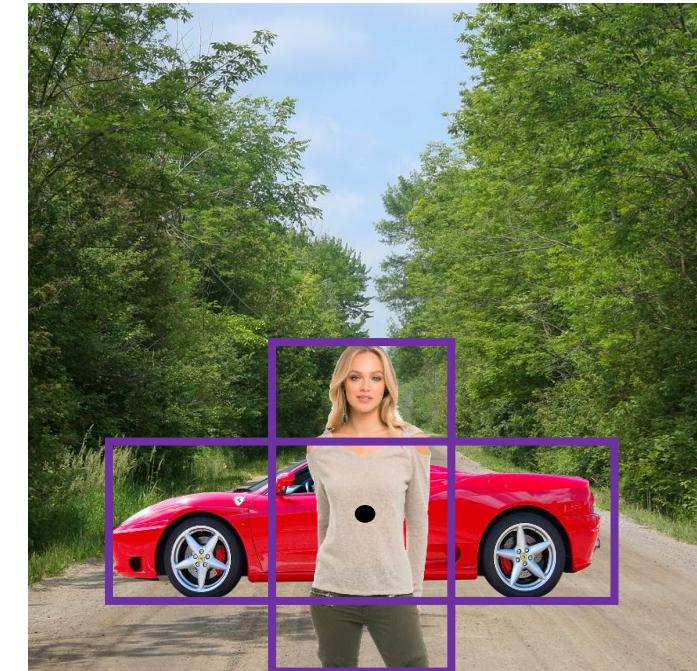
An Example Grid Cell

- $(b_x, b_y) = (0.4, 0.3)$: 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



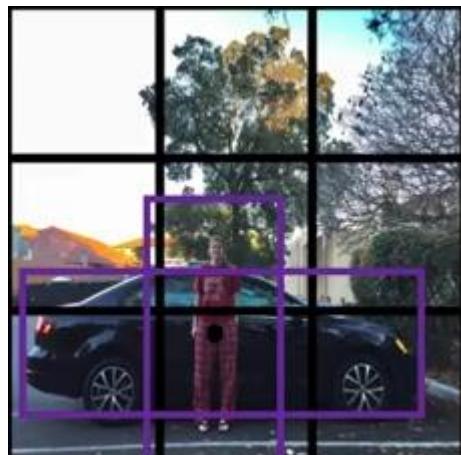
Anchor Boxes

- 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 $3 \times 3 \times 2 \times 8 = 3 \times 3 \times 16$
- 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 $3 \times 3 \times 5 \times 10 = 3 \times 3 \times 50$



Anchor box 1:



Anchor box 2:



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

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.

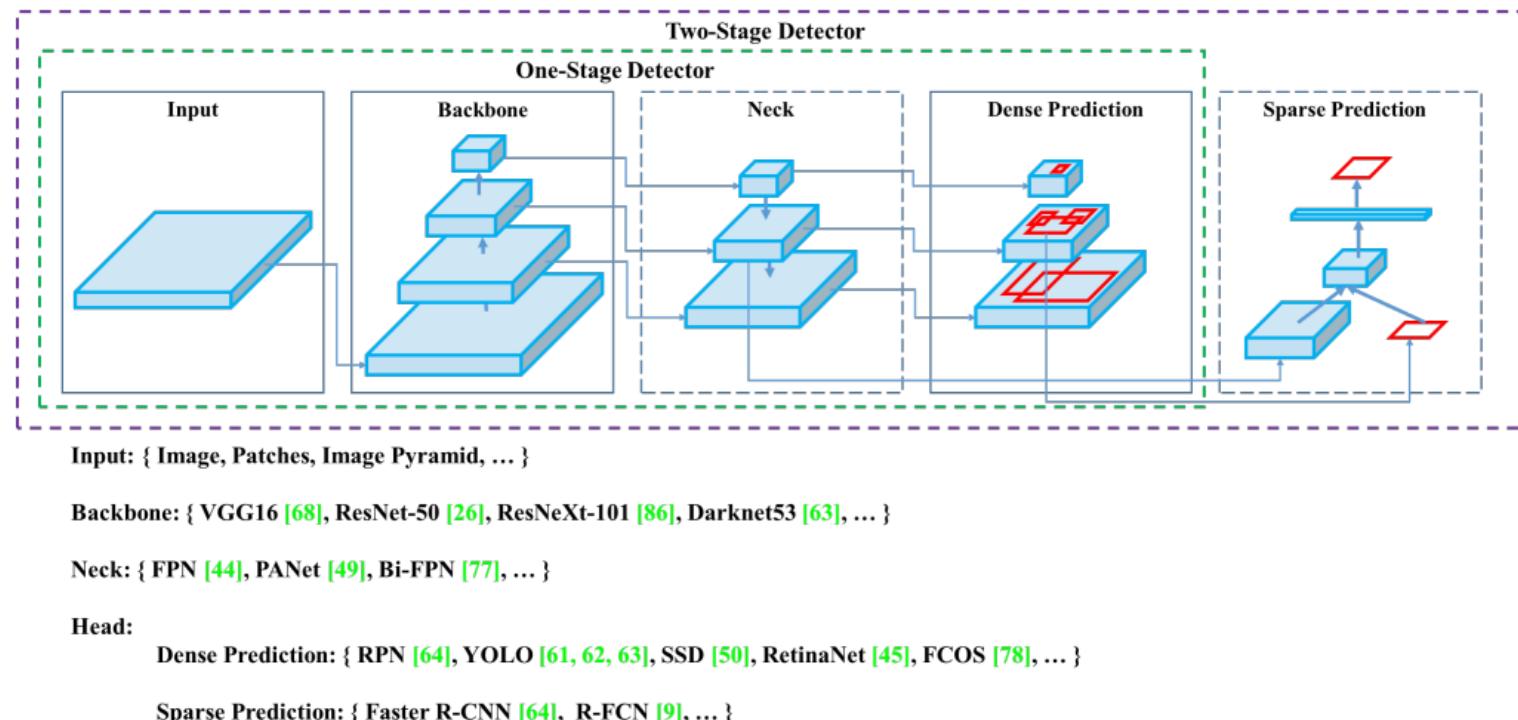
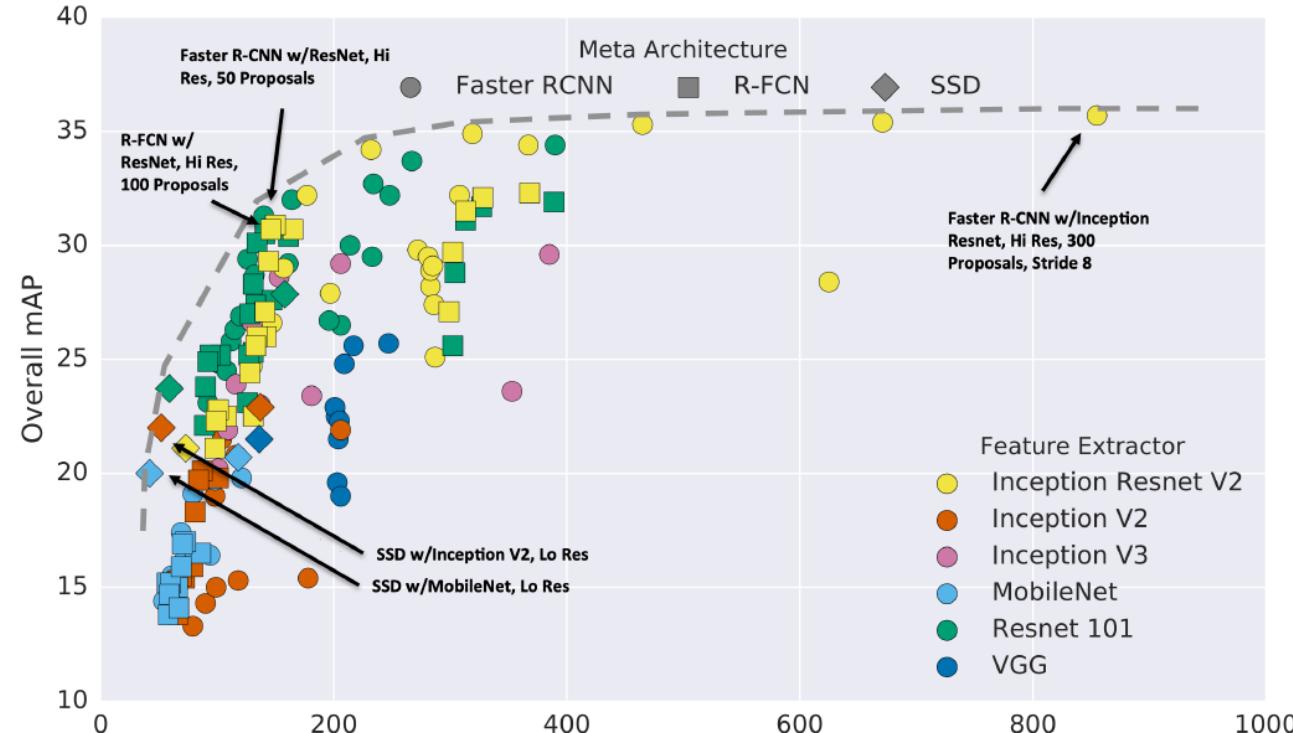


Figure 2: Object detector.

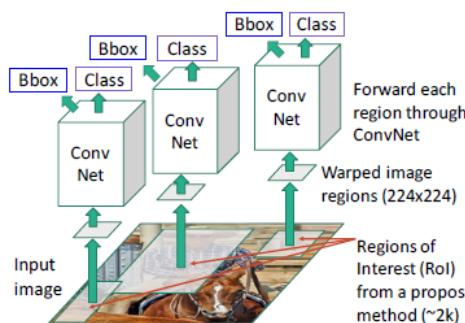
Performance Comparisons

- 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

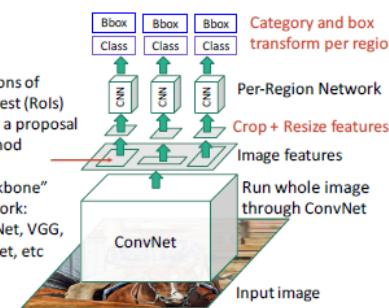


Summary of Object Detectors

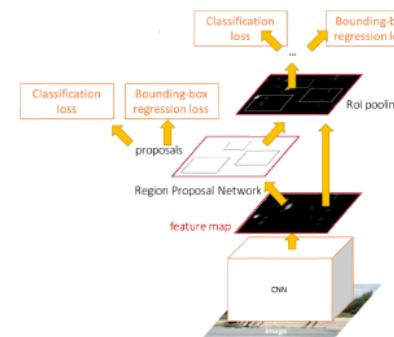
“Slow” R-CNN: Run CNN independently for each region



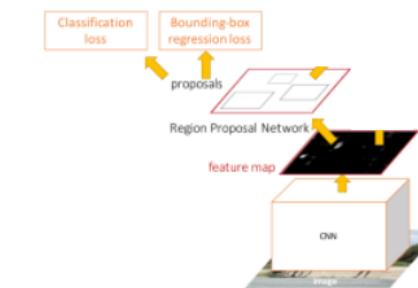
Fast R-CNN: Apply backbone network to generate feature maps once



Faster R-CNN: Compute proposals with RPN

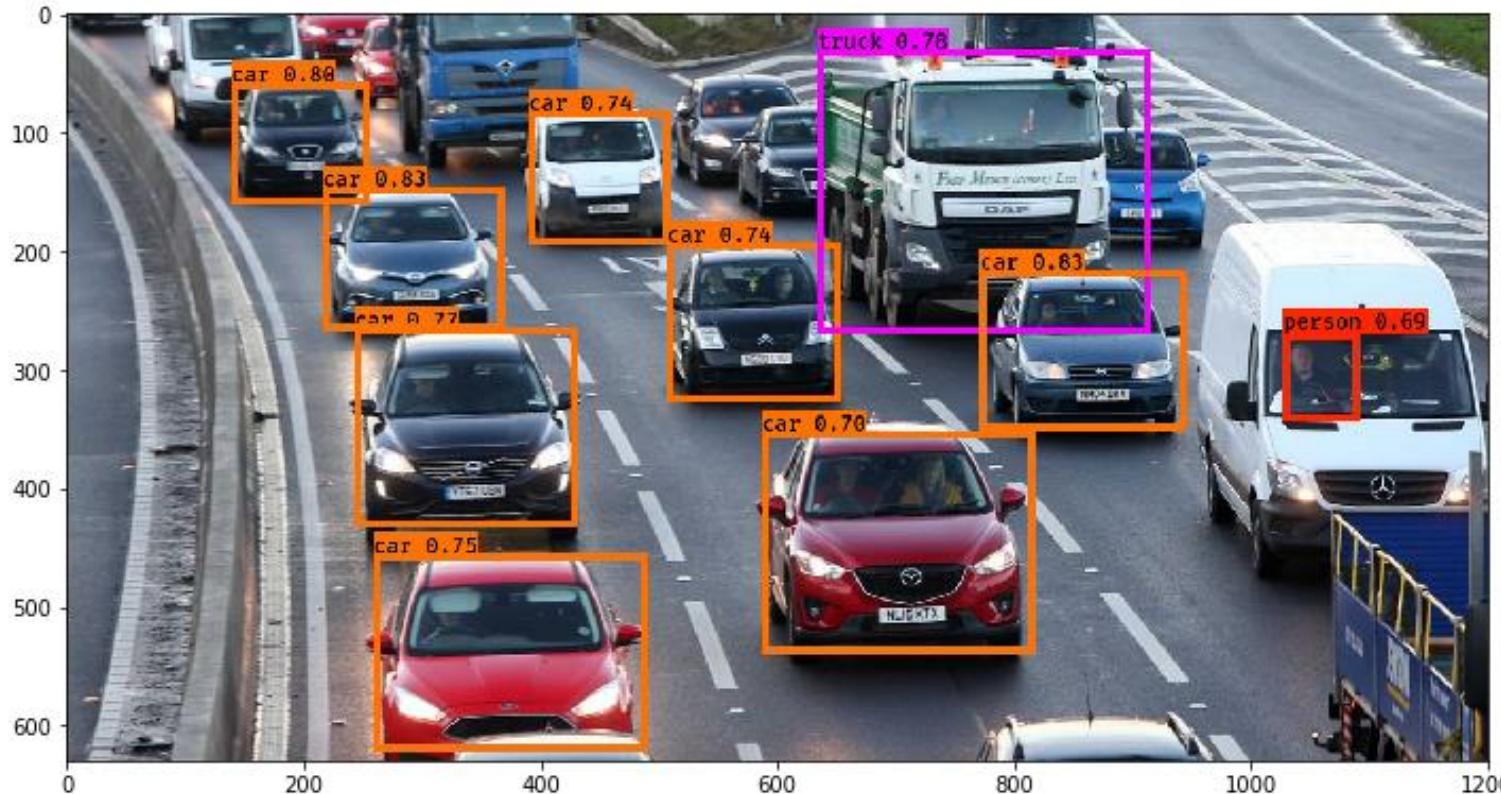


Single-Stage: Fully convolutional detector



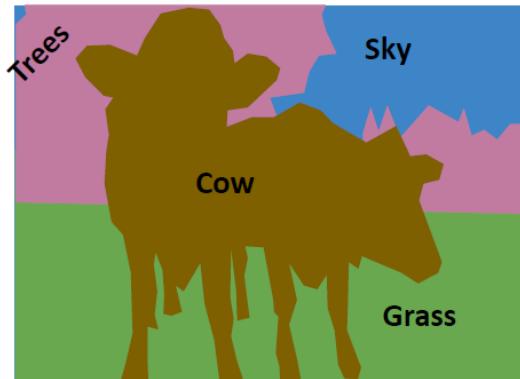
Outline

- Object detection
- Segmentation

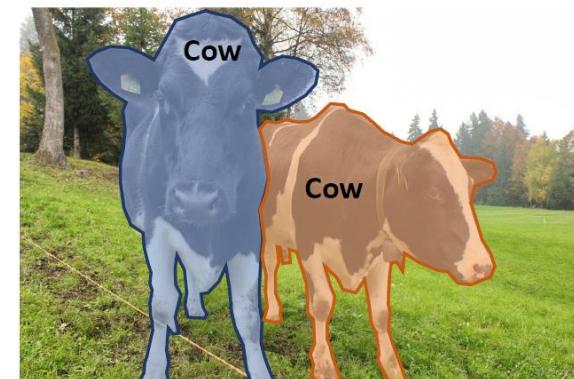


Types of Segmentation Tasks

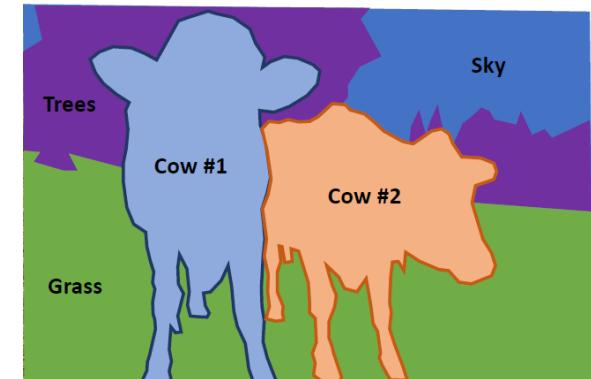
- Things vs. stuff
 - Things: Object categories that can be separated into object instances (e.g. cats, cars, person)
 - Stuff: Object categories that cannot be separated into instances (e.g. sky, grass, water, trees)
- Object Detection vs. Semantic Segmentation vs. Instance Segmentation
 - Object Detection: 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



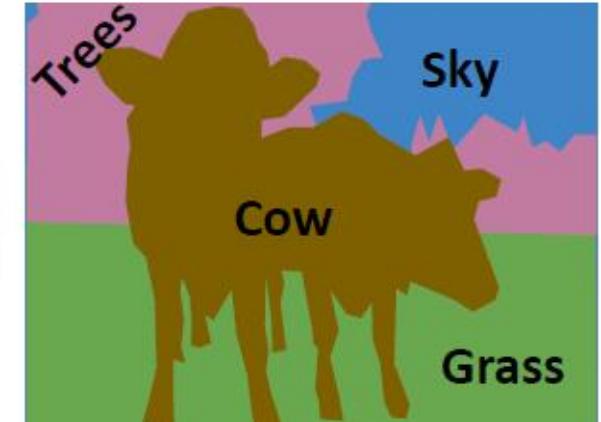
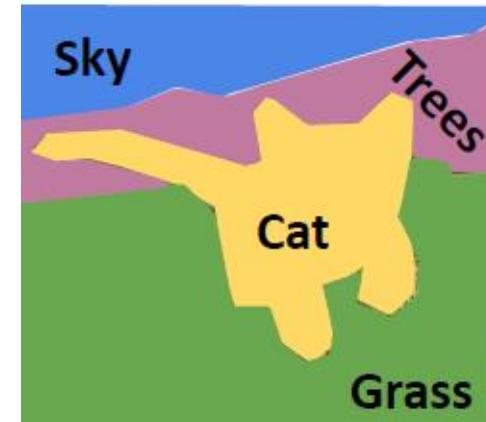
Instance Segmentation



Panoptic Segmentation

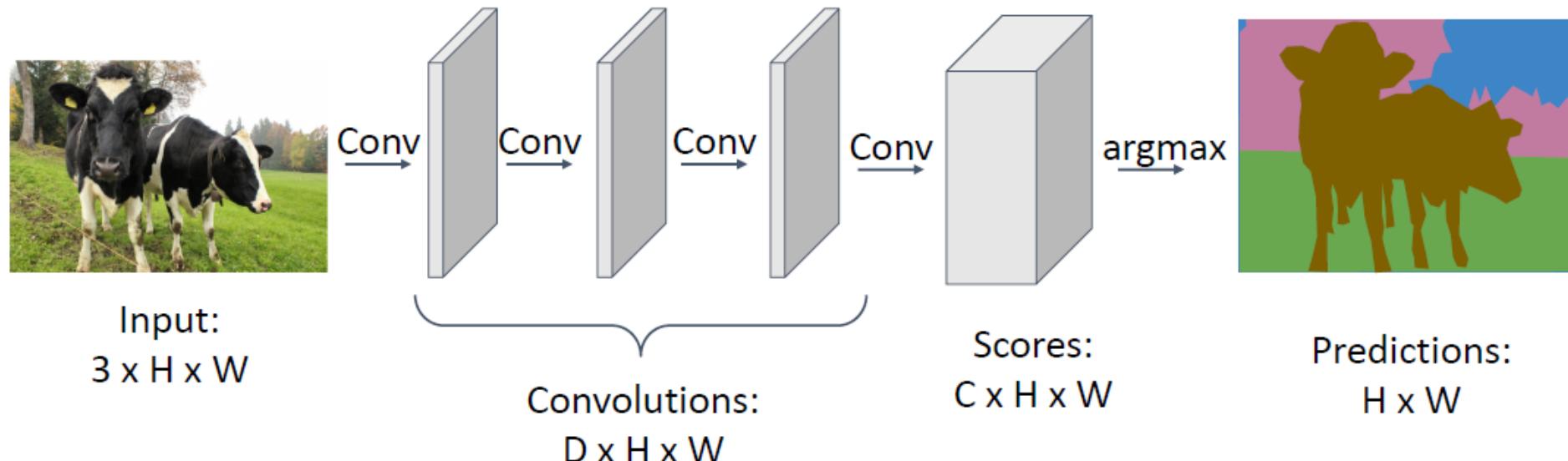
Semantic Segmentation: Task Definition

- 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)



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



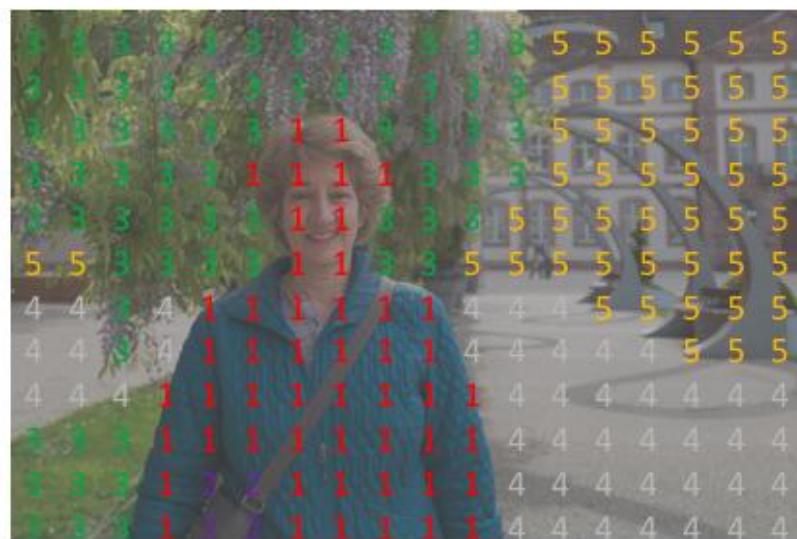
Ground Truth Per-Pixel Labels



→
segmented

- 1: Person
- 2: Purse
- 3: Plants/Grass
- 4: Sidewalk
- 5: Building/Structures

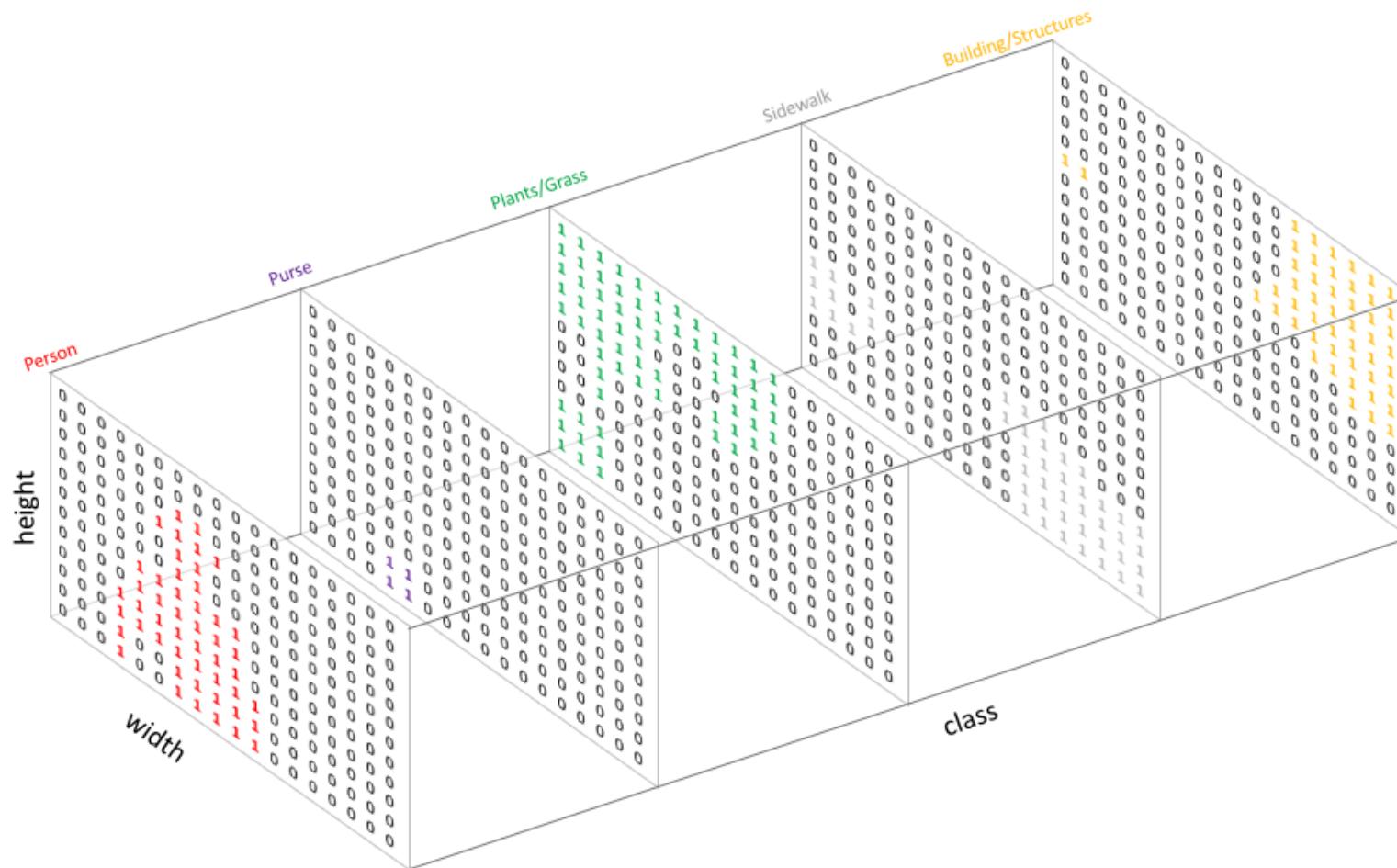
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5		
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5	
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5	5	
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5	5	
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5	5	
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5	5	
5	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5	5	
4	4	3	4	1	1	1	1	1	1	1	1	1	1	1	1	4	4	4	4	5	5	5	5
4	4	3	4	1	1	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	5	5	5
4	4	4	1	1	1	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4	4
3	3	3	1	1	1	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4	4



- 0: Background/Unknown
- 1: Person
- 2: Purse
- 3: Plants/Grass
- 4: Sidewalk
- 5: Building/Structures

Ground Truth Per-Pixel Labels

- One output channel for each of the 5 possible classes.
- Pixel-wise Cross-Entropy Loss

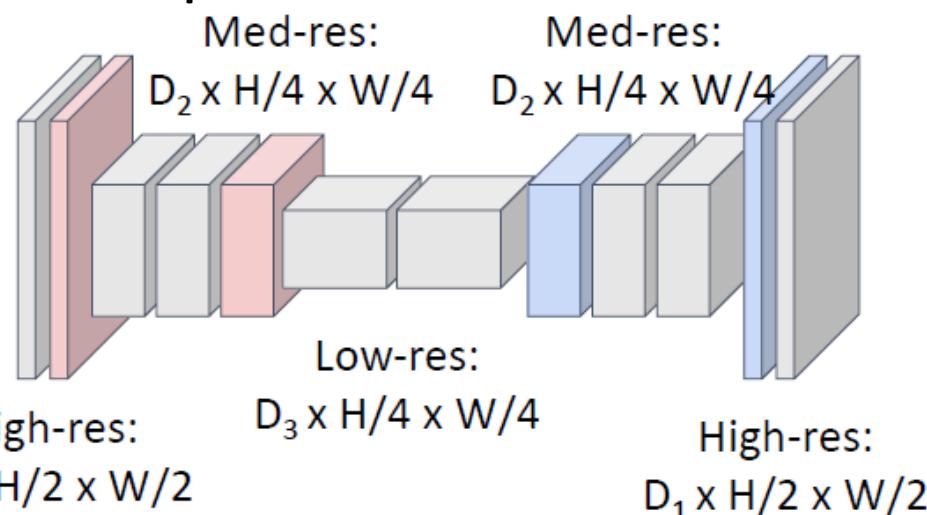


More Efficient FCN

- 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



Input:
 $3 \times H \times W$



Predictions:
 $H \times W$

Unpooling for Upsampling

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

Bed of Nails

1	2
3	4



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

Input
 $C \times 2 \times 2$

Output
 $C \times 4 \times 4$

Nearest Neighbor

1	2
3	4



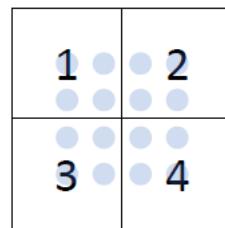
1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Input
 $C \times 2 \times 2$

Output
 $C \times 4 \times 4$

Bilinear/Bicubic Interpolation for Upsampling

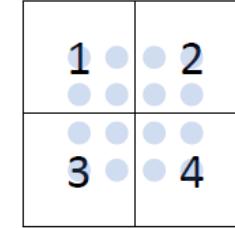
- Upsampling from a 2x2 image to a 4x4 image with bilinear (left) and bicubic (right) interpolation, to generate smoother outputs
- Each output element is computed as a linear or cubic combination of its closest neighbors; closer neighbors are given higher weights
 - Bilinear: use 4 closest neighbors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

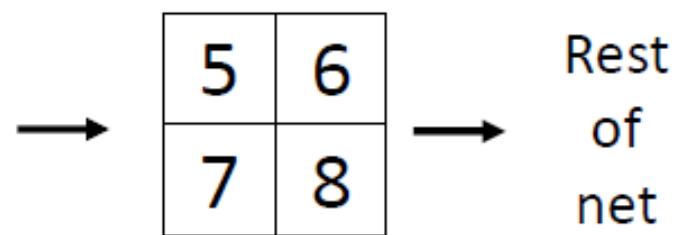
0.68	1.02	1.56	1.89
1.35	1.68	2.23	2.56
2.44	2.77	3.32	3.65
3.11	3.44	3.98	4.32

Output: C x 4 x 4

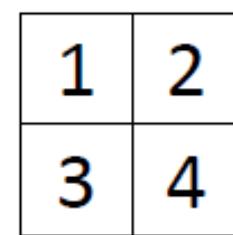
Max Unpooling

Max Pooling: Remember which position had the max

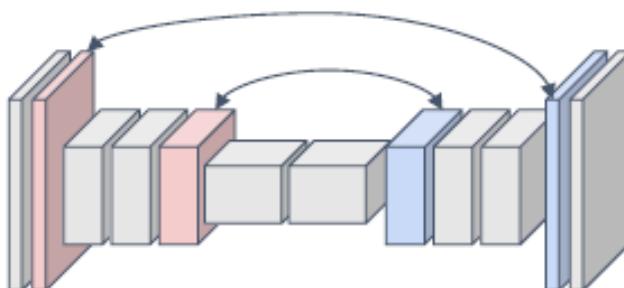
1	2	6	3
3	5	2	1
1	2	2	1
7	3	4	8



Rest
of
net



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

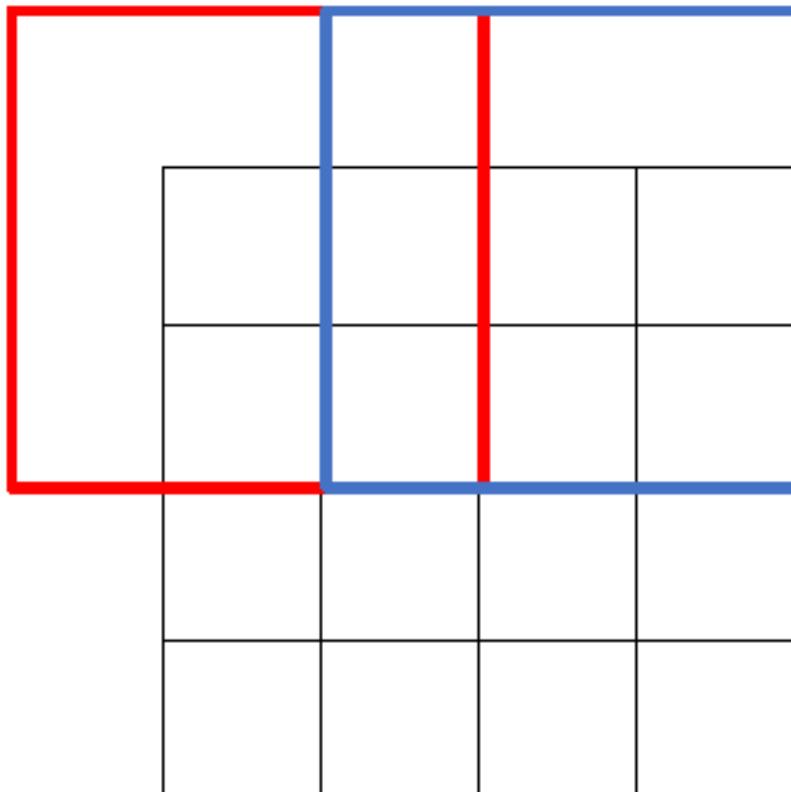


Pair each downsampling layer
with an upsampling layer

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

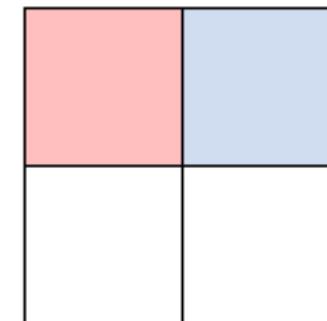
Recall: Regular Convolution

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



Input: 4×4

Dot product
between input
and filter

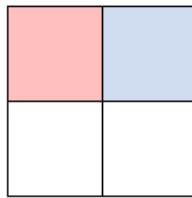


Output: 2×2

Learnable Upsampling: Transposed Convolution

3 x 3 convolution transpose, stride 2

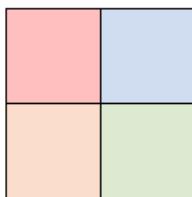
Filter moves 2 pixels in output
for every 1 pixel in input



Input: 2 x 2

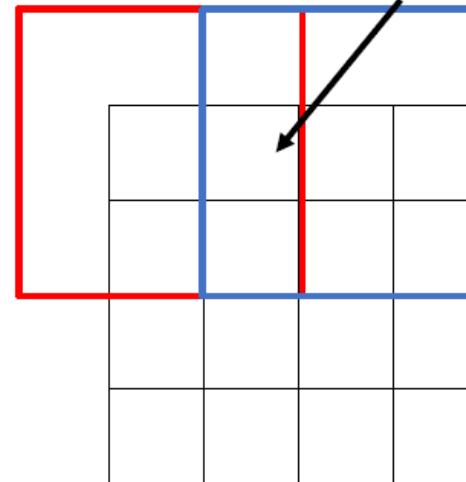
3 x 3 convolution transpose, stride 2

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...)



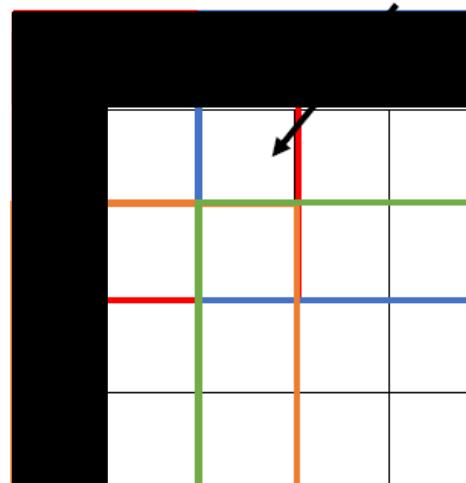
Input: 2 x 2

Sum where
output overlaps



Output: 4 x 4

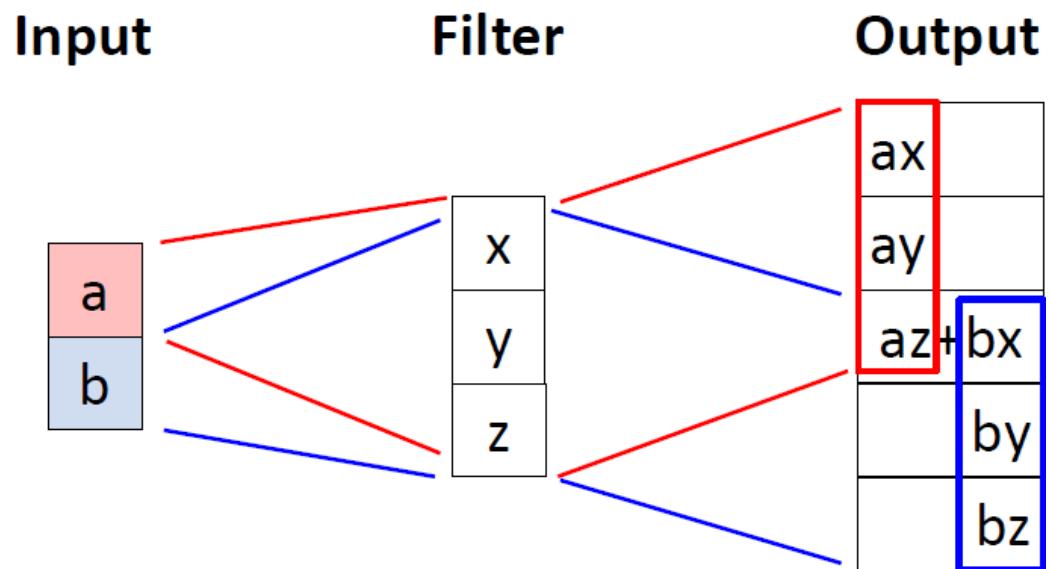
Sum where
output overlaps



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



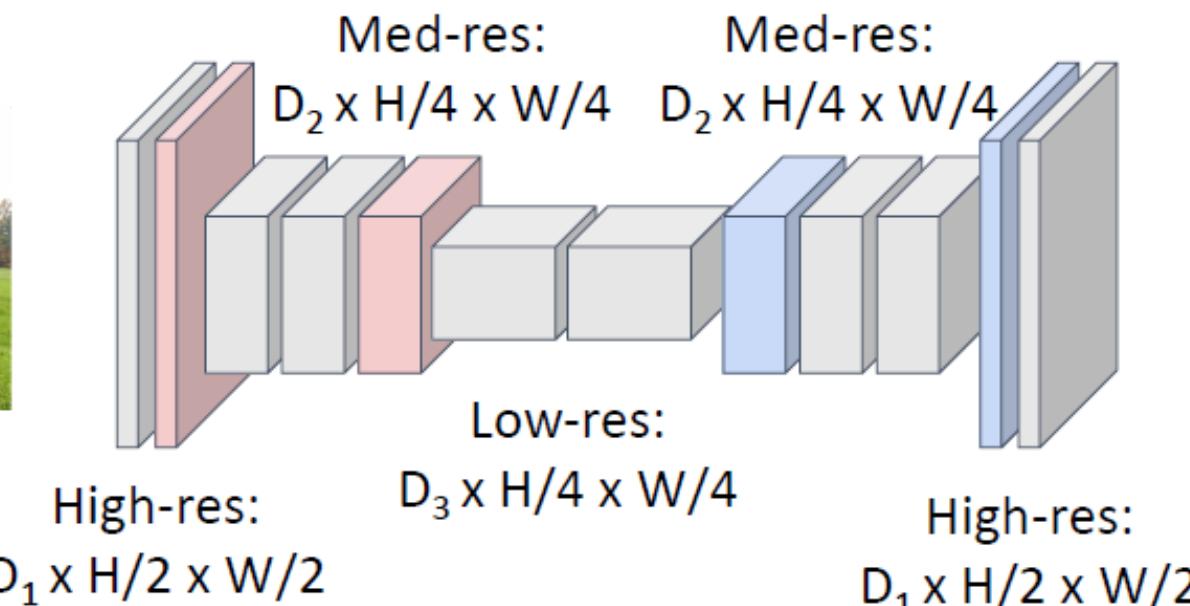
Semantic Segmentation: Fully Convolutional Network

Downsampling:
Pooling, strided
convolution



Input:
 $3 \times H \times W$

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



Upsampling:
interpolation,
transposed conv



Predictions:
 $H \times W$

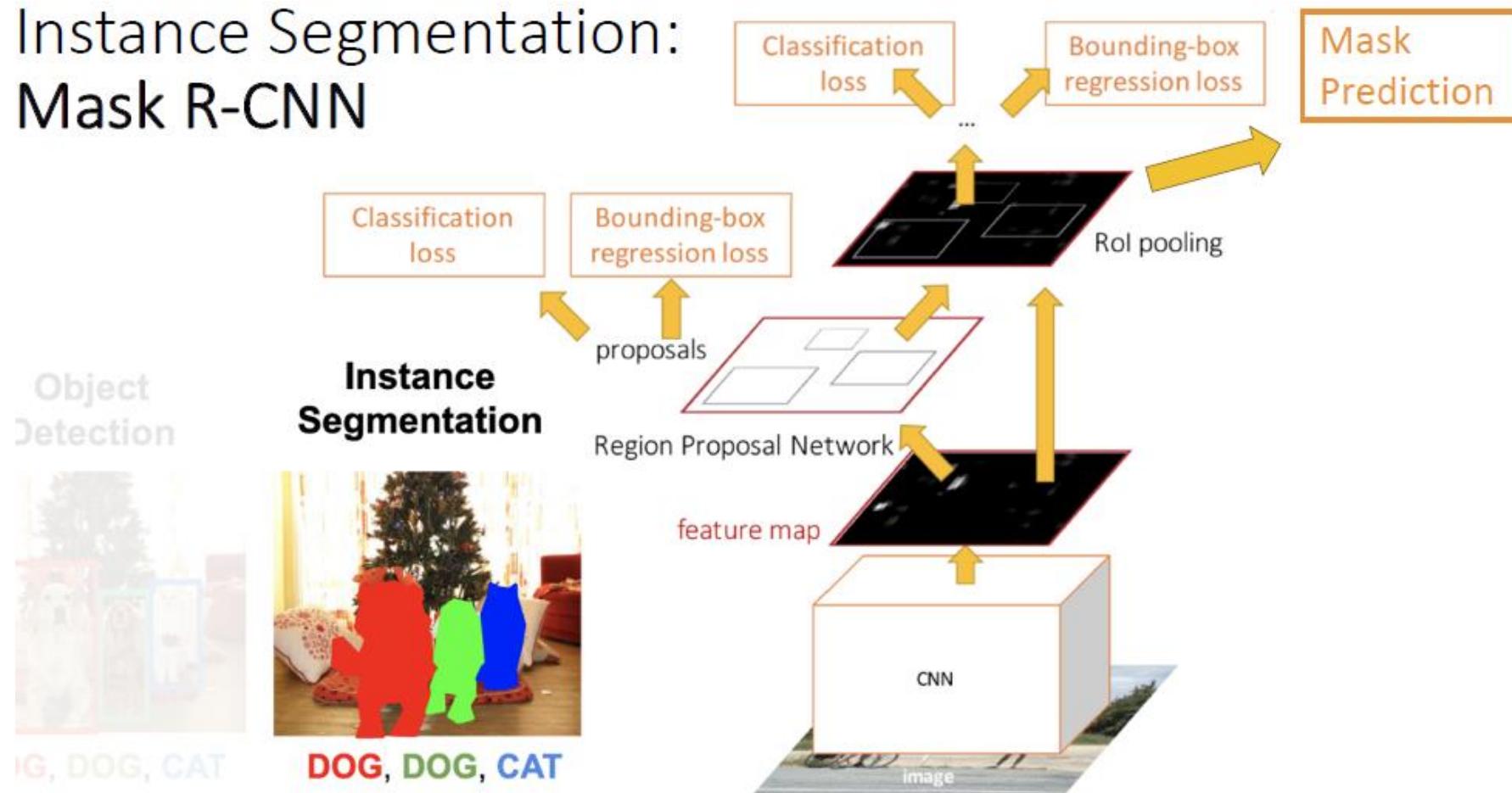
Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Loss function: Per-Pixel cross-entropy

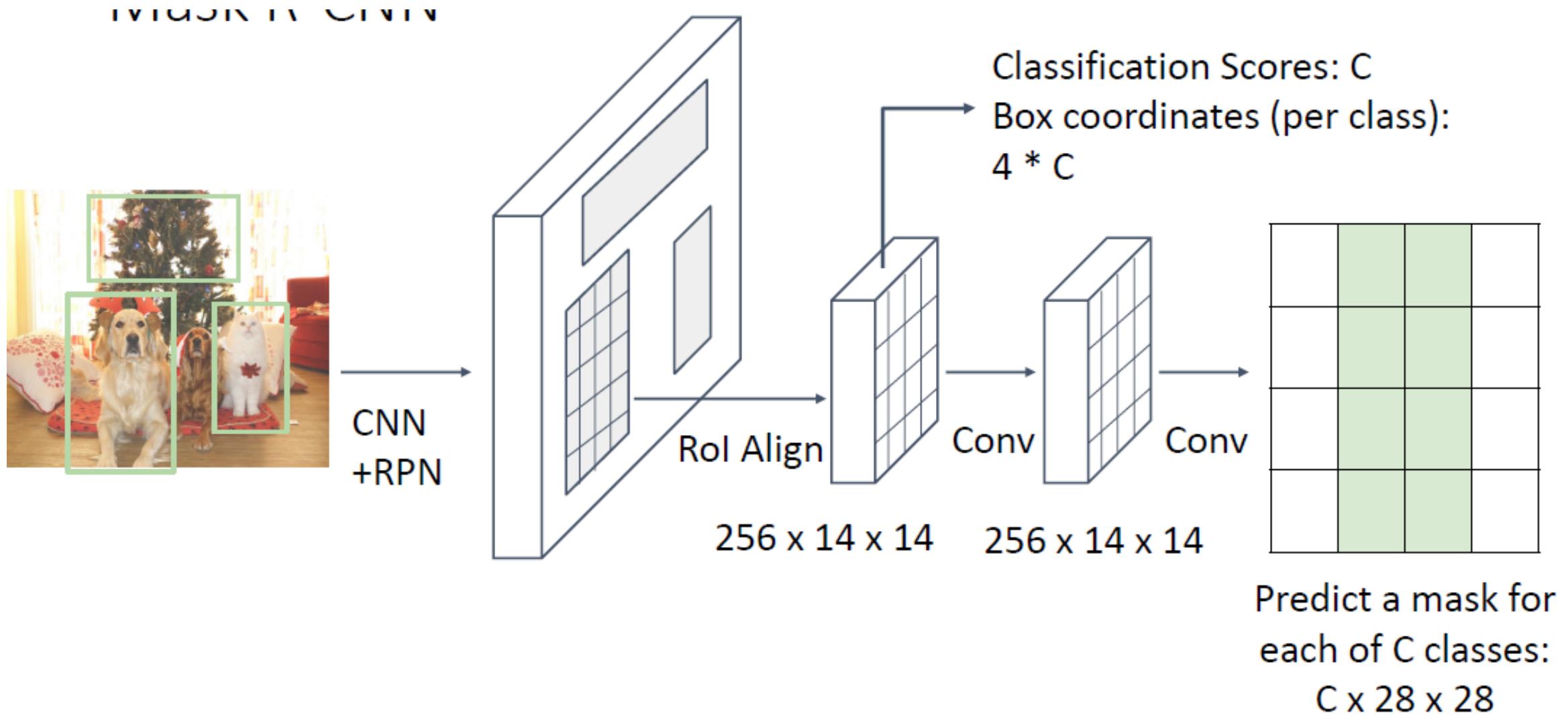
Mask R-CNN for Instance Segmentation

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

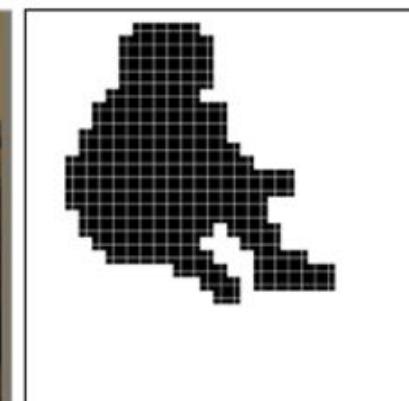
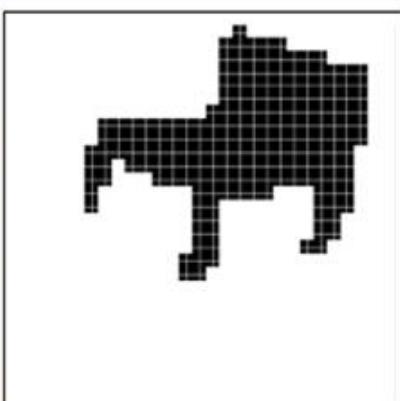
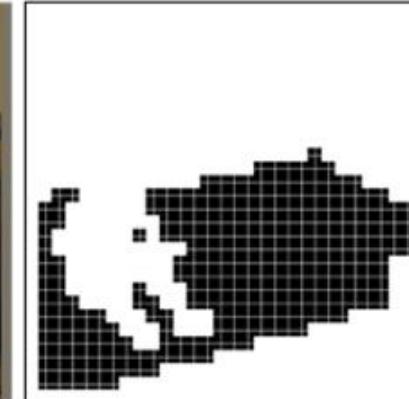
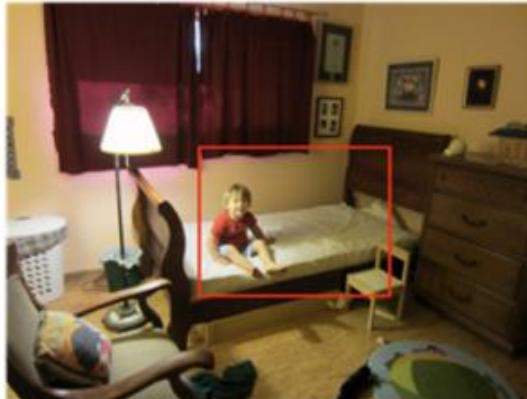
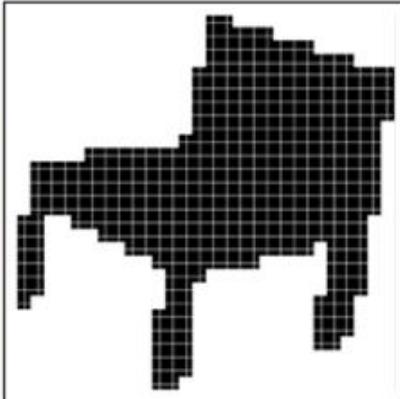
Instance Segmentation:
Mask R-CNN



Mask R-CNN for Instance Segmentation



Example Target Segmentation Masks



Target segmentation mask
for class “chair” in the Bbox

Target segmentation mask
for class “person” in the Bbox

Single-Object Detection

Detecting a single object

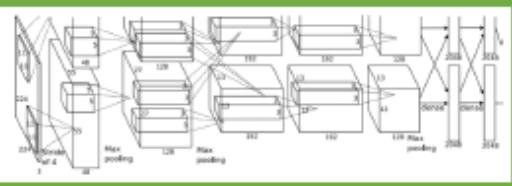


[This image is CC0 public domain](#)

Treat localization as a
regression problem!

Problem: Images can have
more than one object!

Often pretrained
on ImageNet
(Transfer learning)



Fully
Connected:
4096 to 1000

Vector:
4096

“What”

Class Scores

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

Fully
Connected:
4096 to 4

Box
Coordinates
(x, y, w, h)

“Where”

Correct label:
Cat

Softmax

Loss

Multitask
Loss

Weighted
Sum

L2 Loss

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

