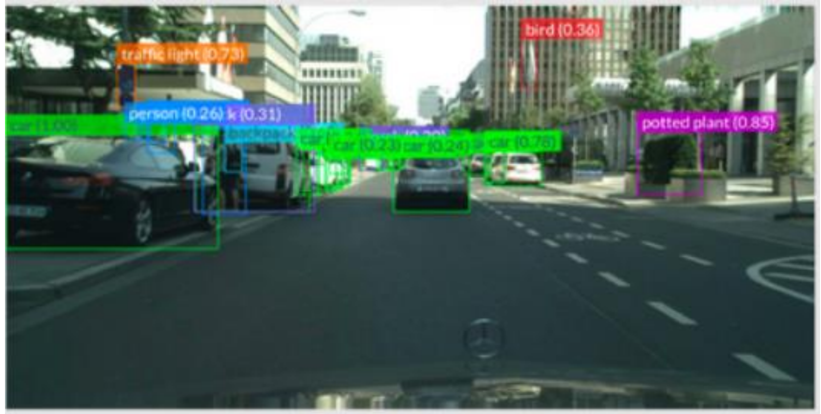
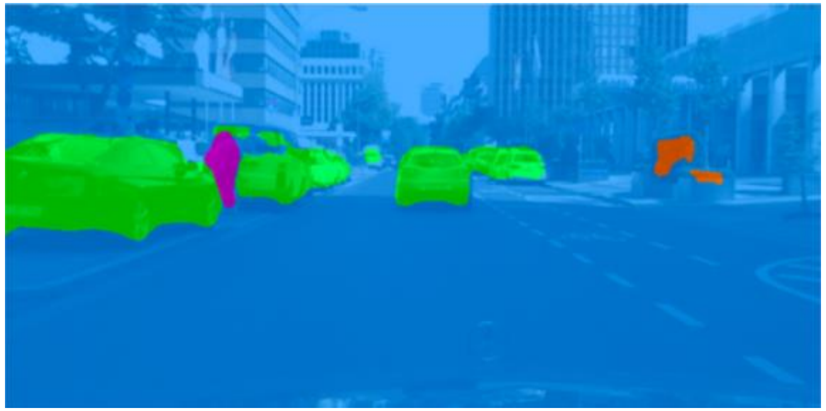
L4 Object Detection and Segmentation

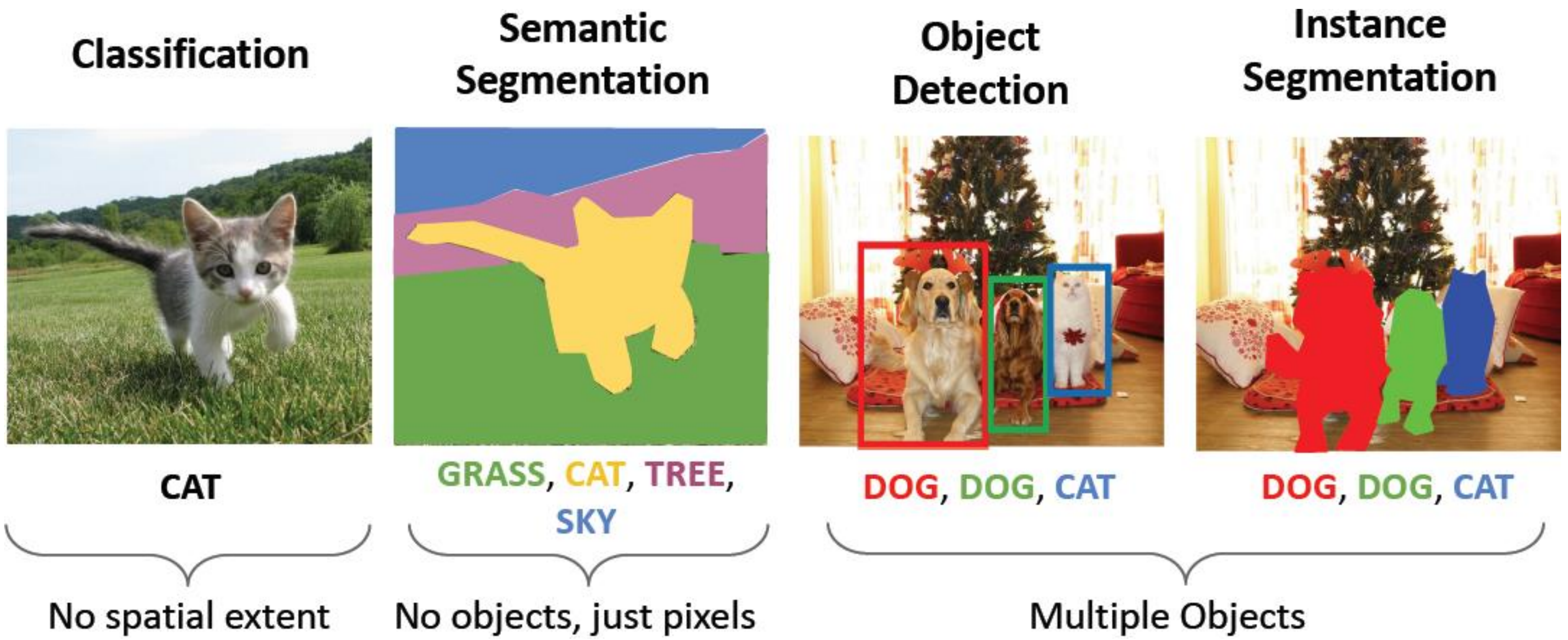




Zonghua Gu, Umeå University

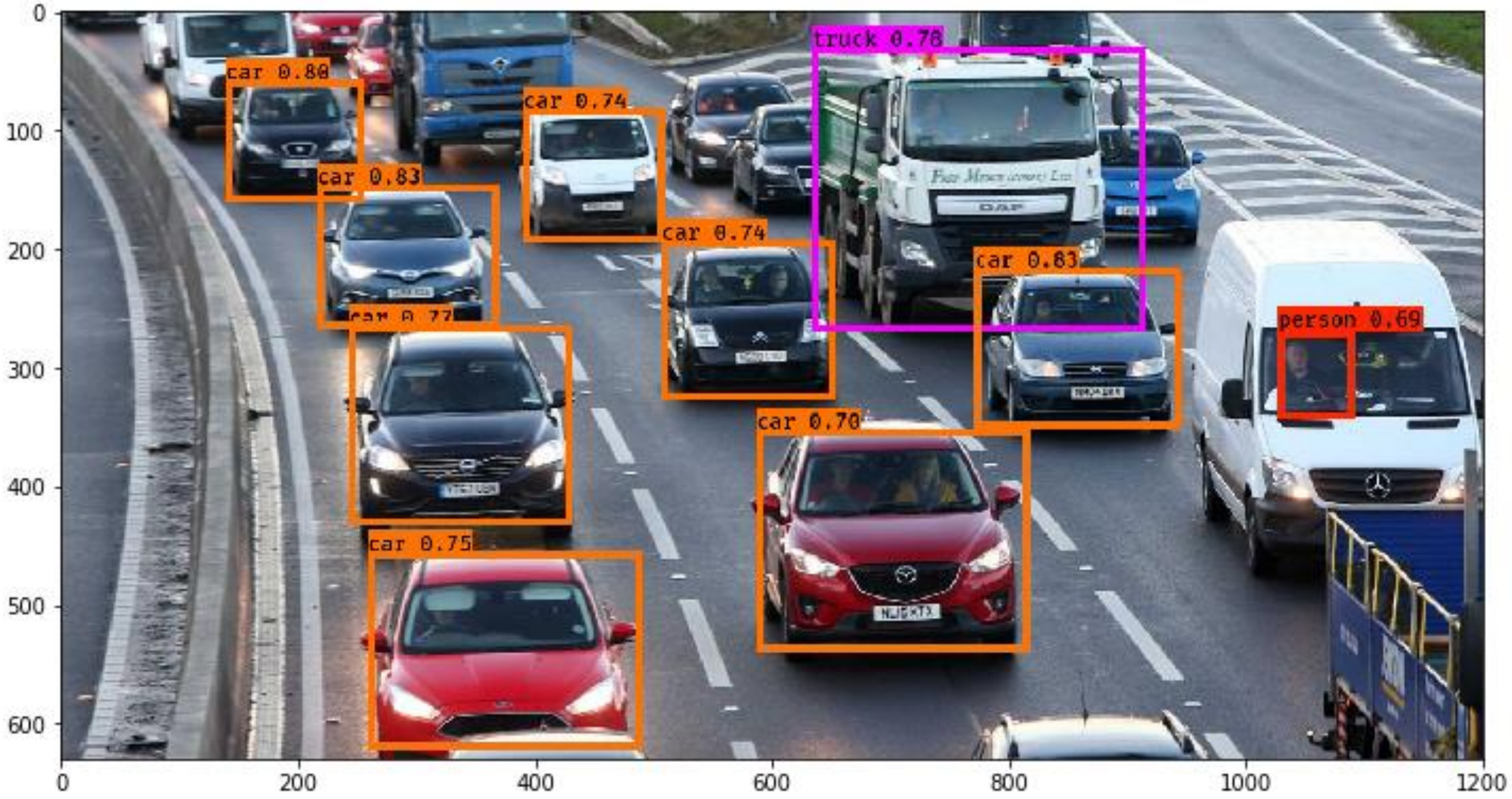
Nov. 2023

Computer Vision Tasks



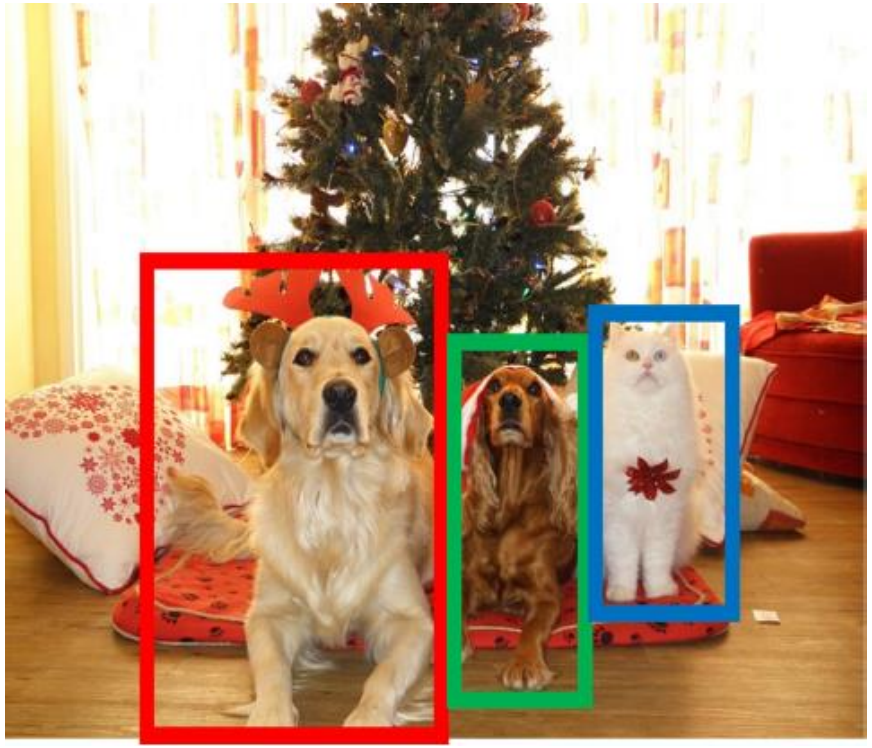
Outline

- Object detection
- Segmentation



Object Detection: Task Definition

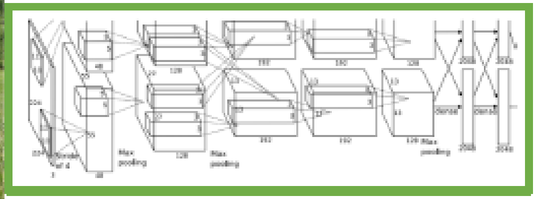
- Input: Single Image
- Output: a set of detected objects
- For each object predict:
  - WHAT: Class label (e.g., cat vs. dog)
  - WHERE: Bbox (4 numbers: x, y, width, height)
- Challenges:
  - Multiple outputs: variable numbers of objects per image
  - Multiple types of output: predict ”what” (class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



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')

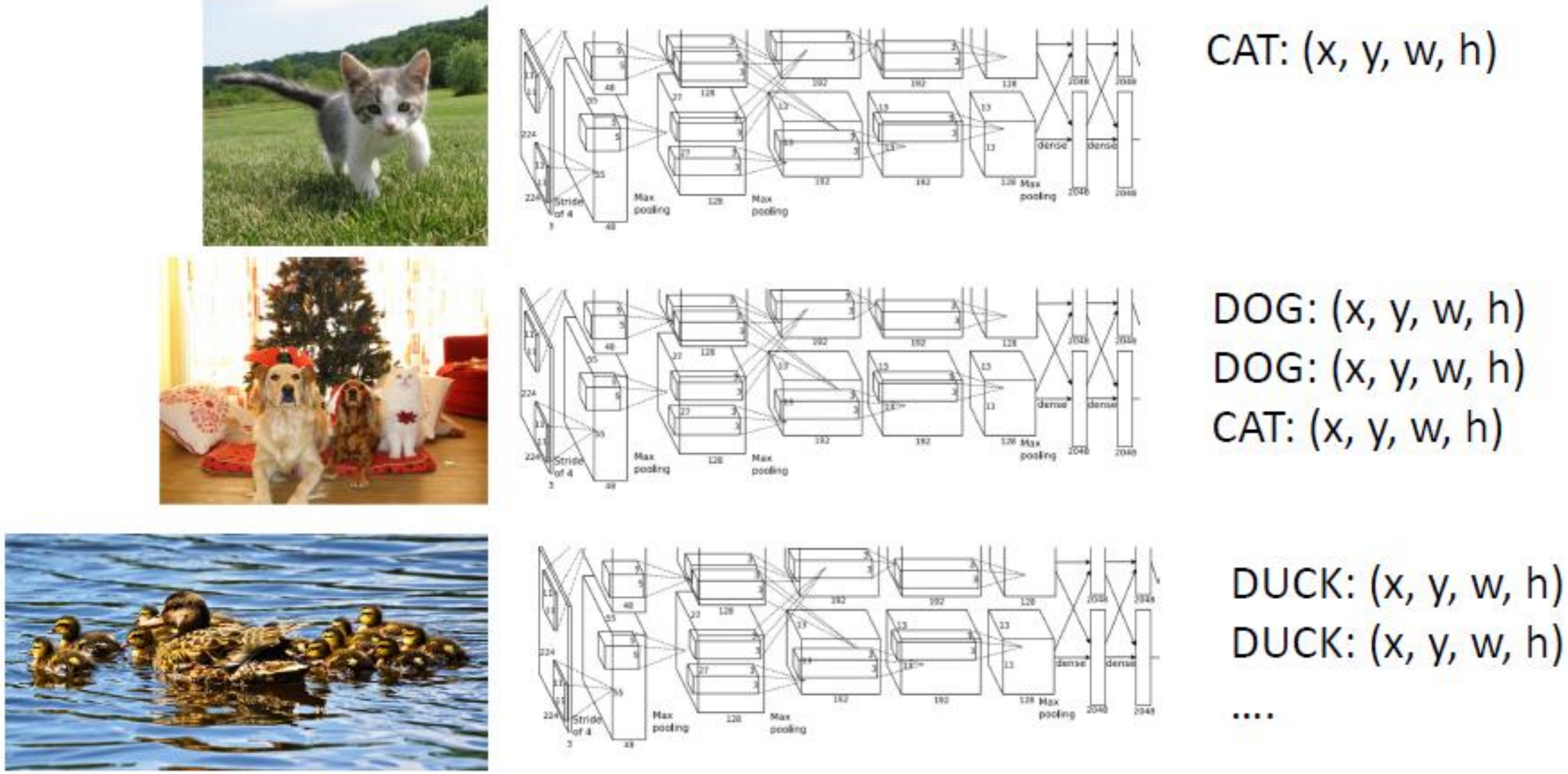
"Where"

Treat localization as a regression problem!

Problem: Images can have more than one object!

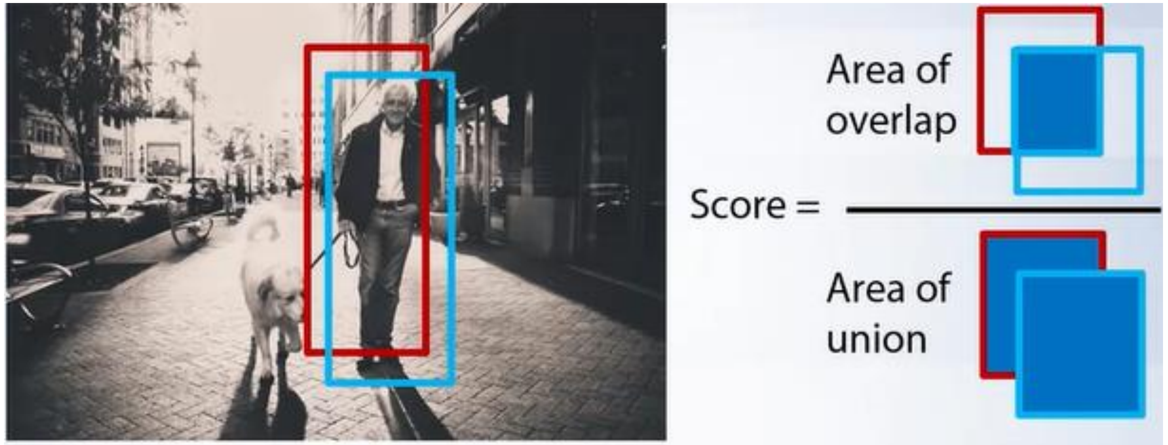
Multi-Object Detection

- Needs to predict 4 numbers for each object 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



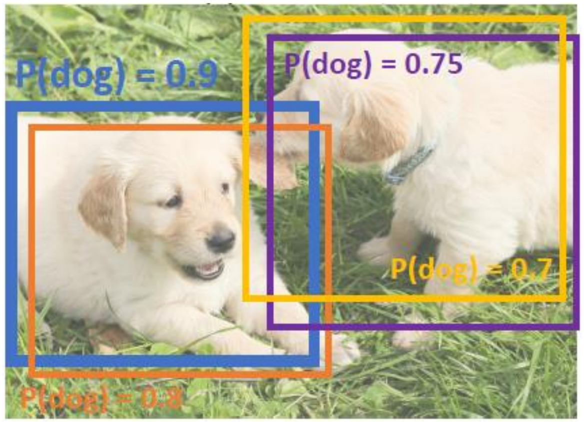
Detection Criteria (Intersection Over Union, IOU)

- 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 $\theta_{IoU} = 0.5$

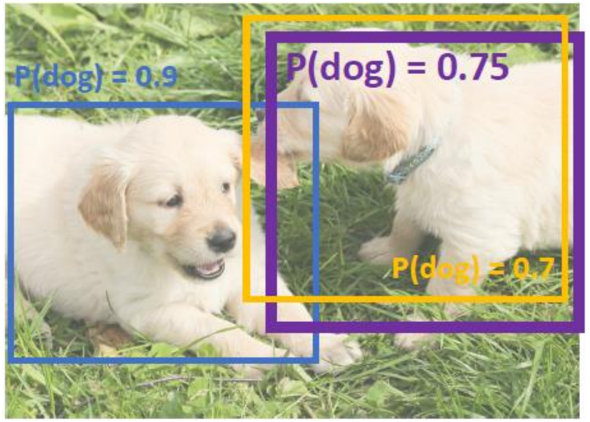


Non-Max Suppression (NMS)

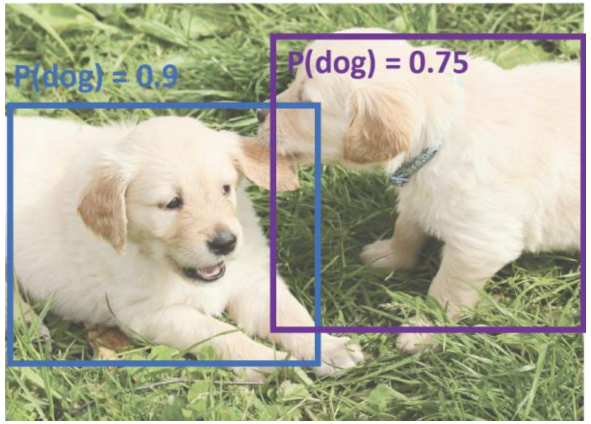
- Problem: Object detectors often output many overlapping detections
- NMS: Discard (suppresses) overlapping object boxes except the one with the maximum classification score
- For each output class
  - 1. Select next highest-scoring box b and output it as a prediction
  - 2. Discard any remaining boxes b' with IoU(b, b') > threshold
  - 3. 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 with $P(dog) = .8$, 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 with $P(dog) = .7$, since IoU(purple, yellow)=.74>.7



IoU(■, ■) = **0.78**
IoU(■, ■) = 0.05
IoU(■, ■) = 0.07



IoU(■, ■) = **0.74**

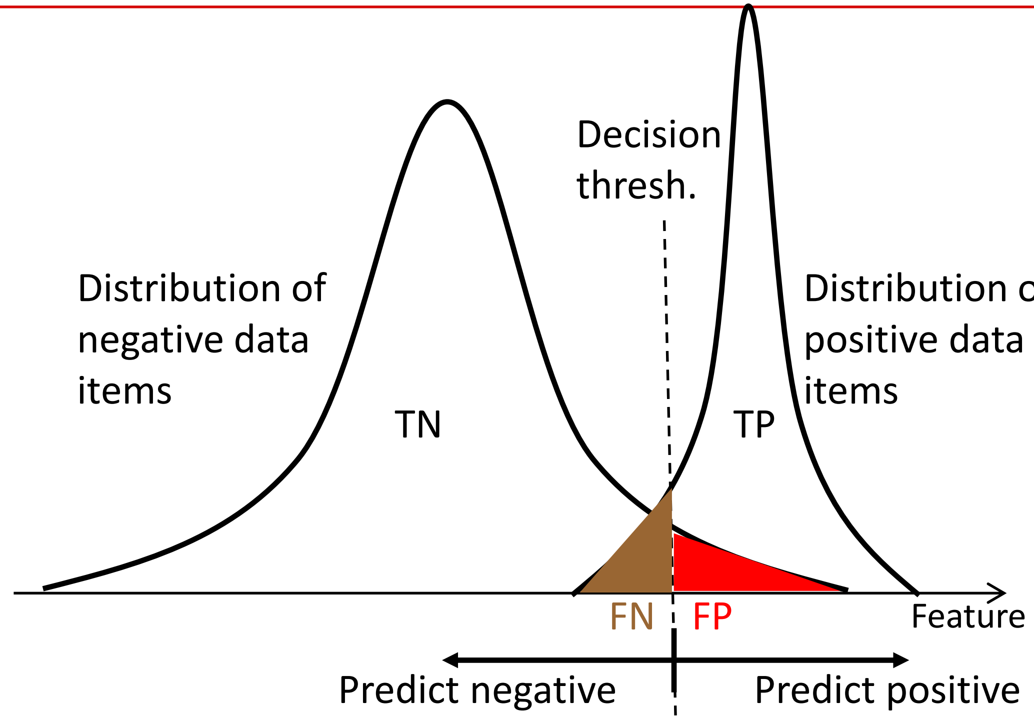


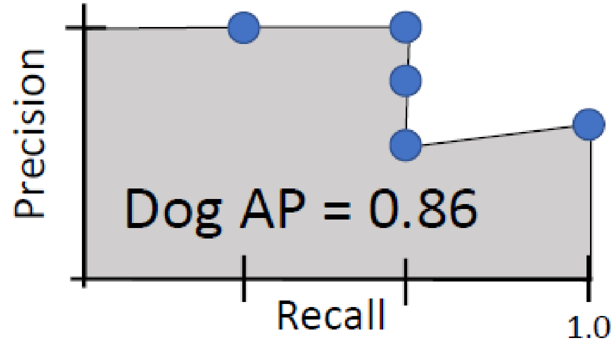
Evaluating Object Detectors: Mean Average Precision (mAP)

- 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}=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

Precision, Recall, and Average Precision

- Precision = $\frac{TP}{TP+FP}$
  - When the classifier predicts positive, it is correct ??% of the time
- Recall (True Positive Rate, TPR) = $\frac{TP}{TP+FN}$
  - Among all the positive cases, the classier correctly classifies ??% of them as positive
- In general, negative correlation between precision and recall (but not strictly monotonic)
  - Decision threshold ↓ ➔ precision ↓, recall ↑
  - Decision threshold ↑ ➔ precision ↑, recall ↓
- Average Precision (AP) for a given class is defined as the AUPRC (Area Under the P-R Curve) for the class

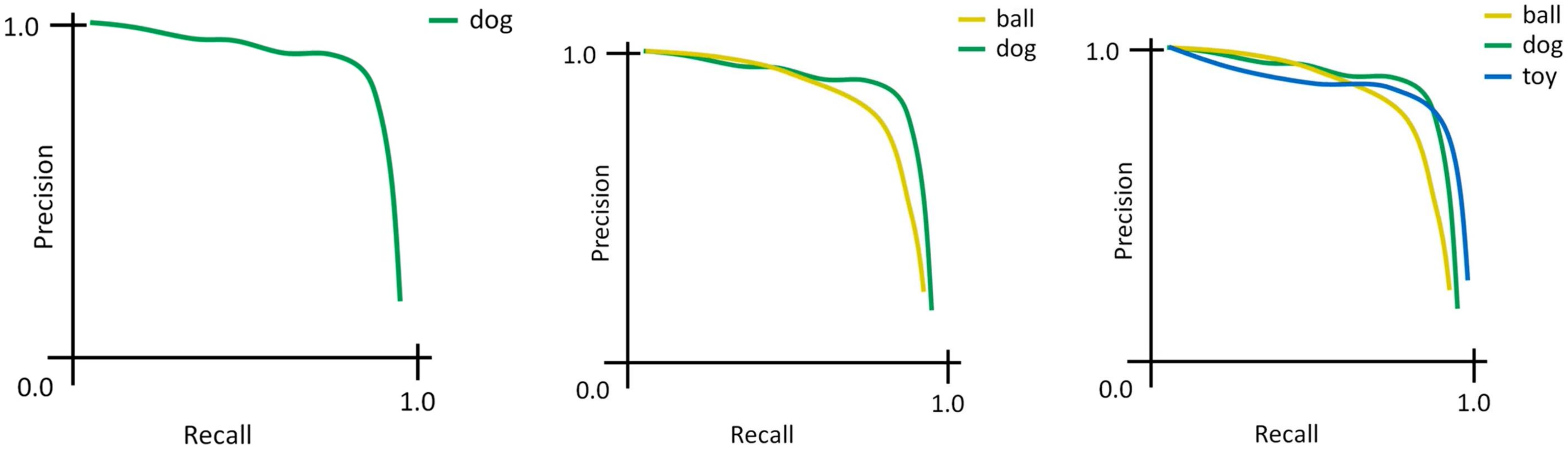




AP_{dog}=AUPRC (Area Under the P-R Curve) for the dog class

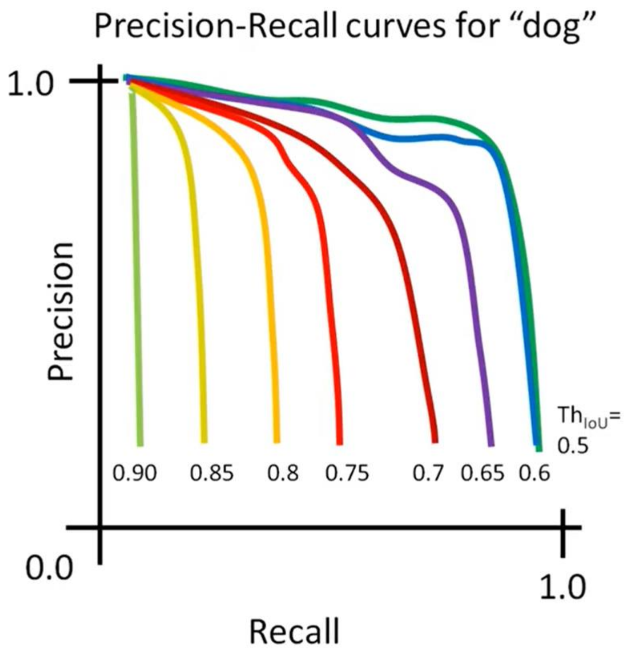
mean Average Precision (mAP)

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



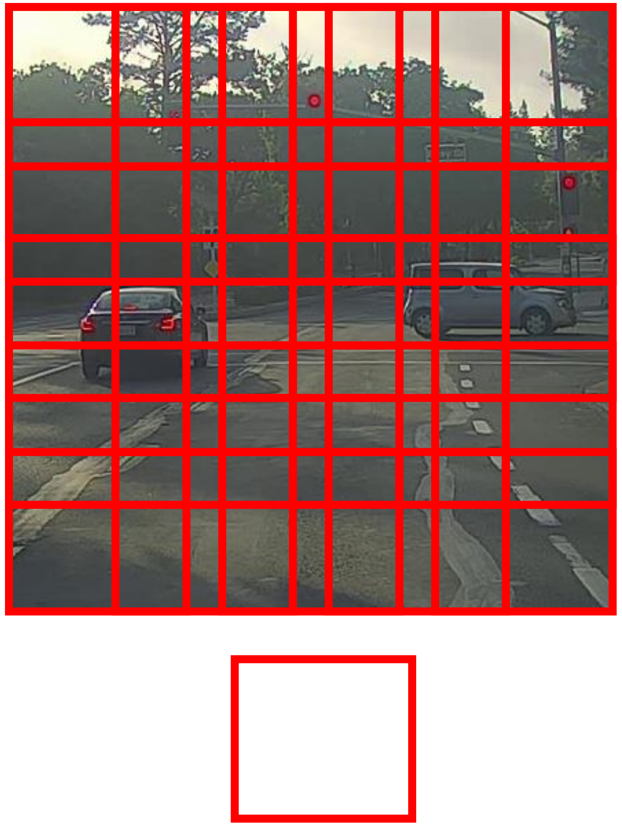
mAP

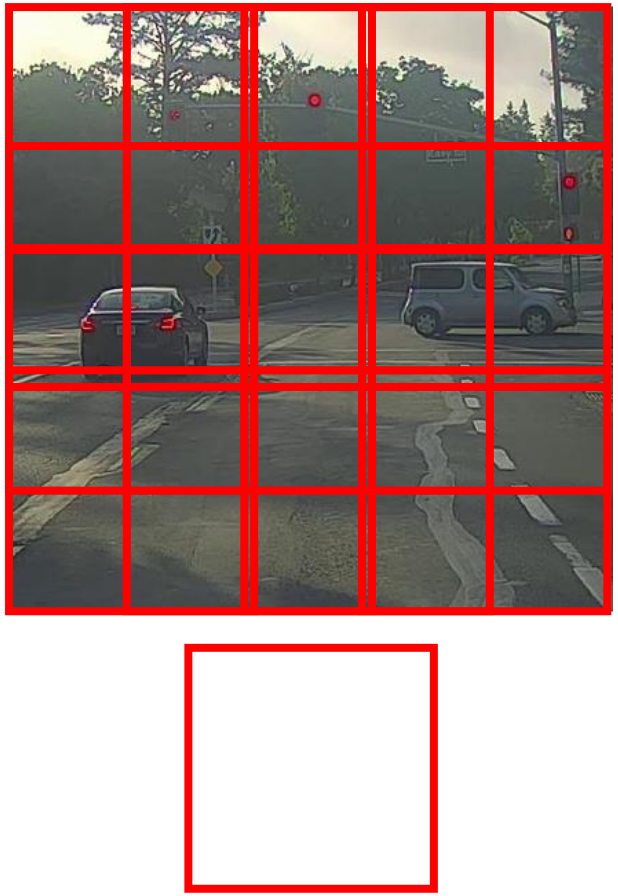
- For COCO 2017 challenge: Compute mAP@threshold for each IoU threshold and take average
  - $mAP = \frac{1}{10} \sum_i mAP_i$
  - where $i = [.5, .55, .6, .65, .7, .75, .8, .85, .9, .95]$
- Figure assumes at least one match (IoU $\geq$ threshold), hence precision starts at 1. If 0 match (all boxes have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





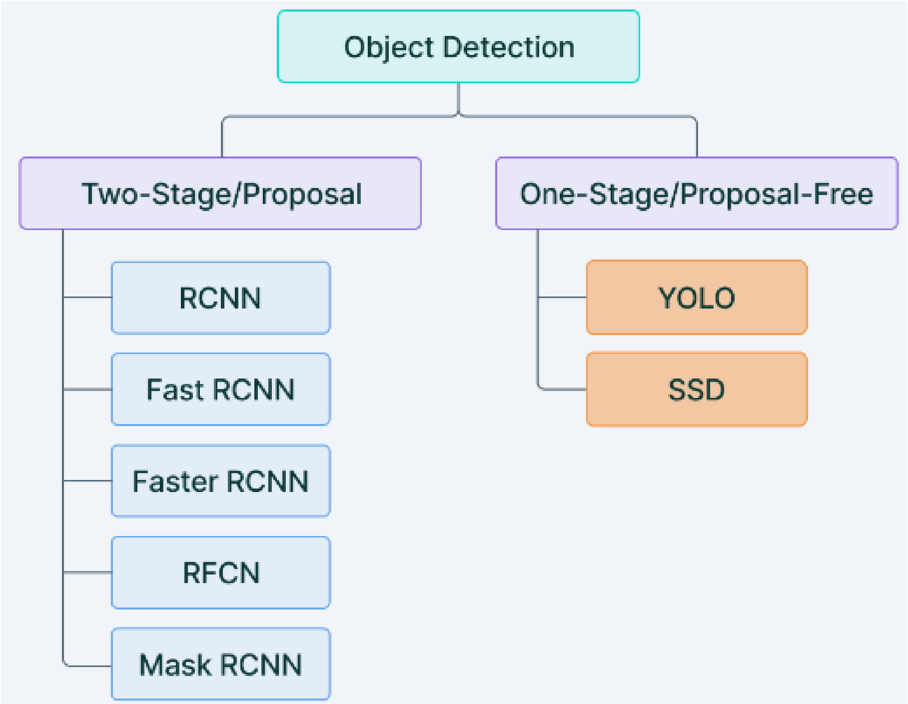
https://www.coursera.org/learn/convolutional-neural-networks/lecture/VgyWR/object-detection

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: $1 \le h \le H, 1 \le w \le W$
  - 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

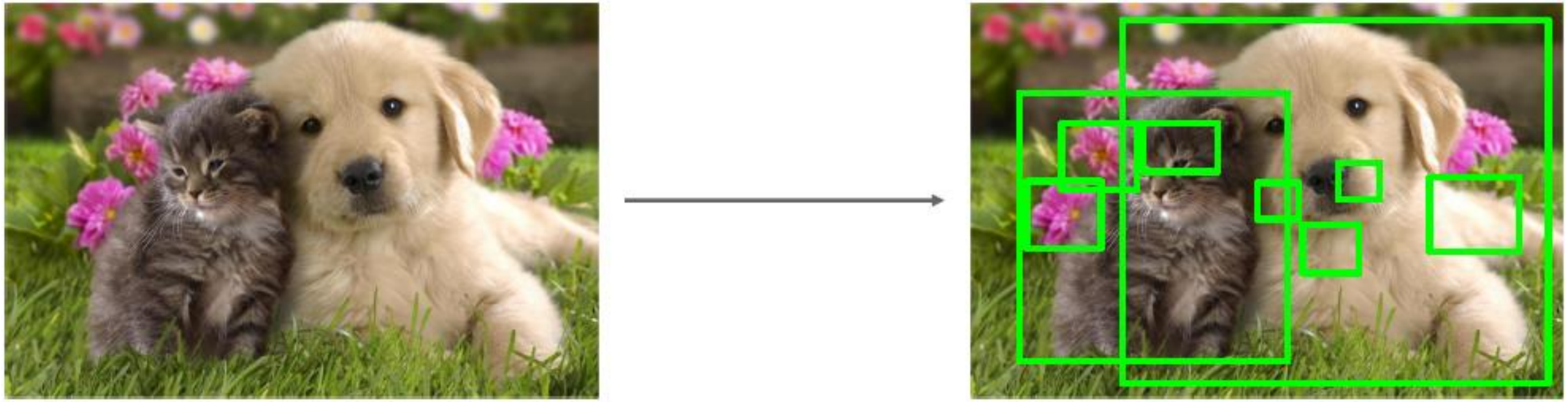
Two-stage vs. One-Stage Detector

- 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



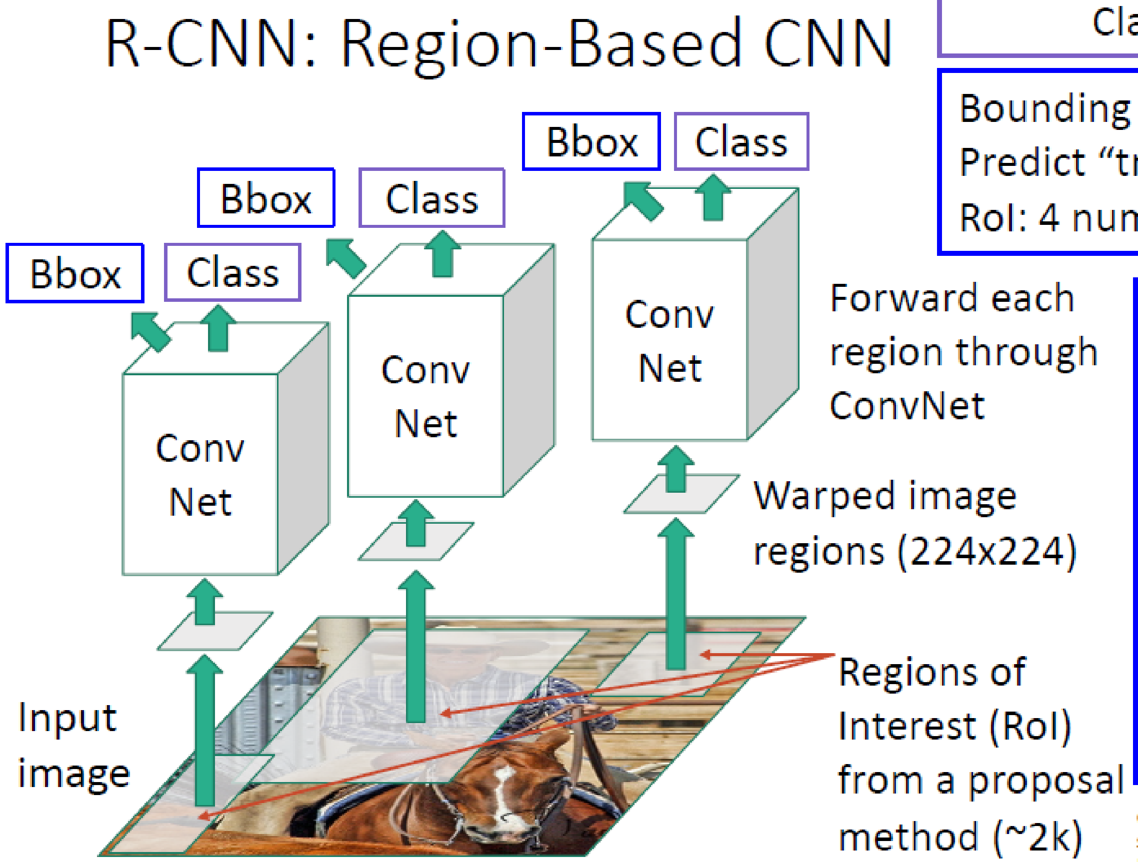
Region Proposals

- 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



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)

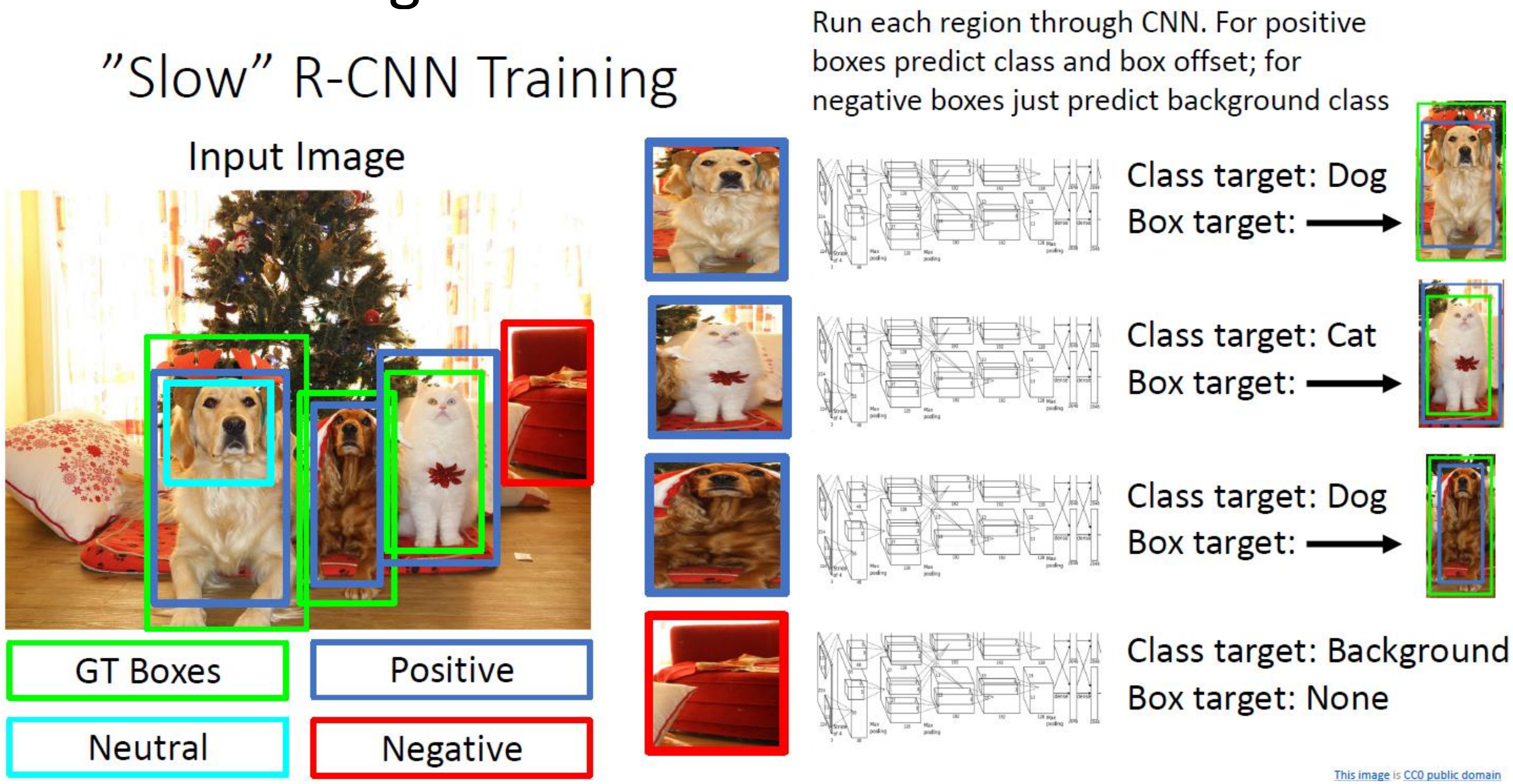
Translate relative to box size:
$b_x = p_x + p_w t_x$ $\quad$ $b_y = p_y + p_h t_y$

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

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

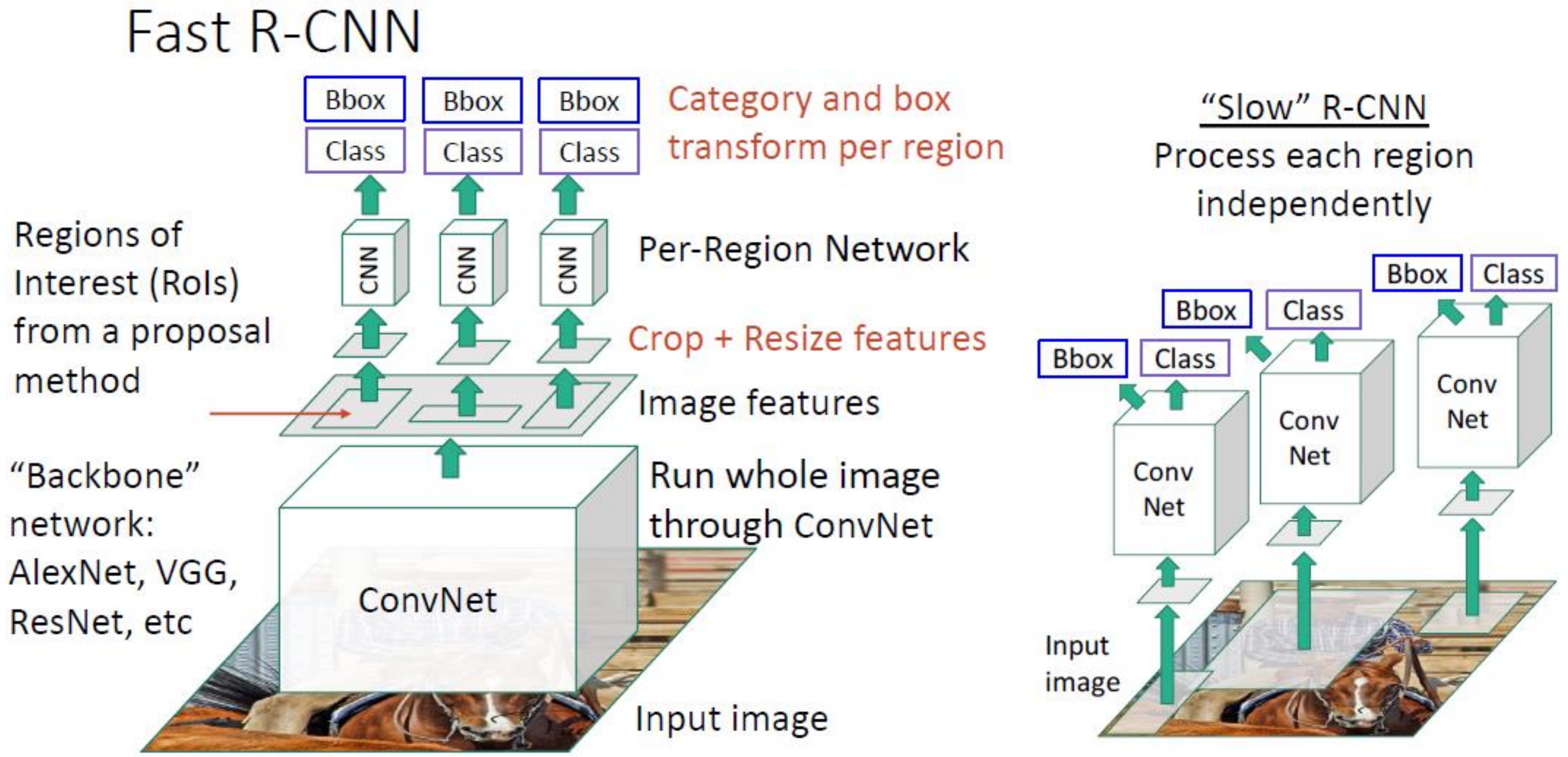


R-CNN: Test Time

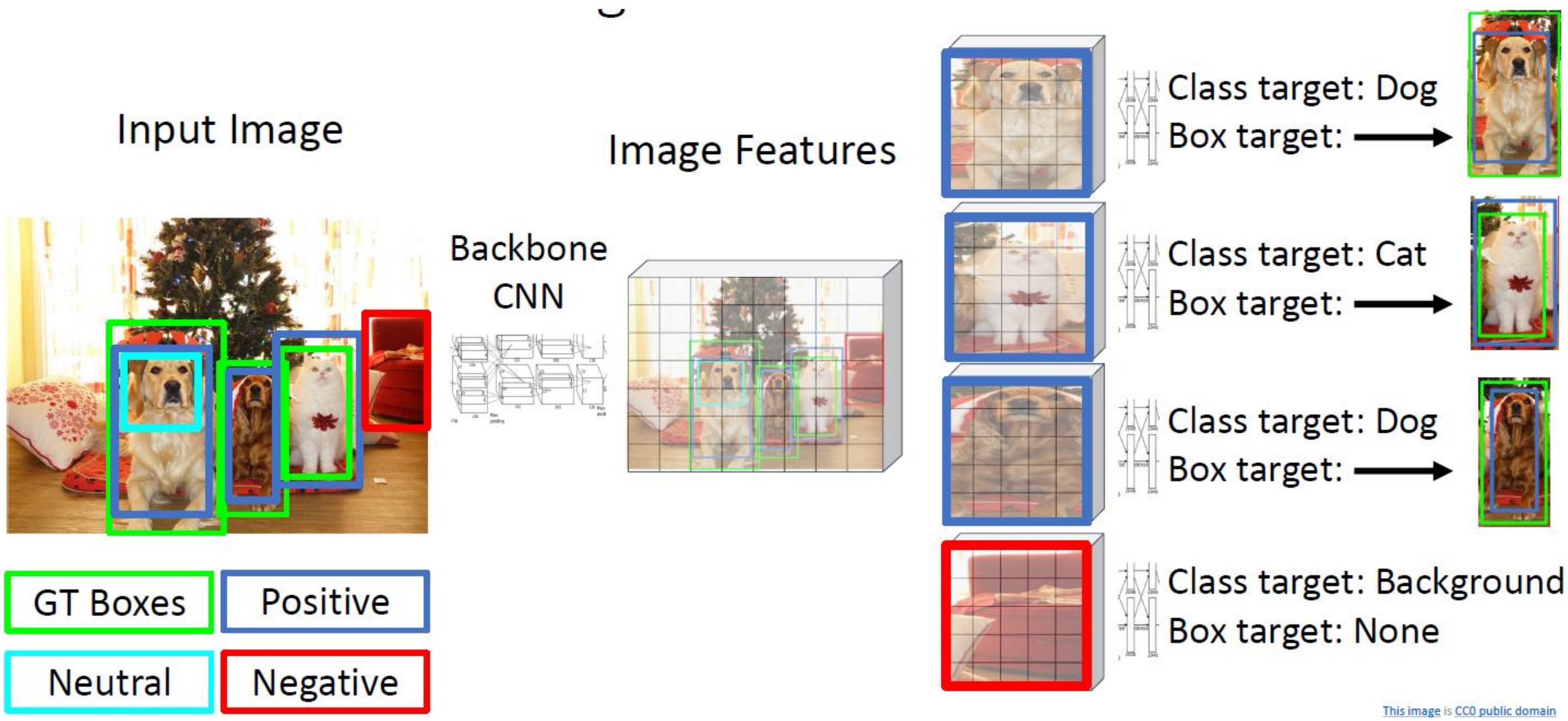
- 1. Run region proposal method to compute ~2000 region proposals
- 2. Resize each region to 224x224 (tunable hyperparam) and run independently through the CNN to get feature vectors, then use Linear SVM to predict class scores, and Linear Regression to predict Bboxes
- 3. Use scores to select a subset 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
- 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

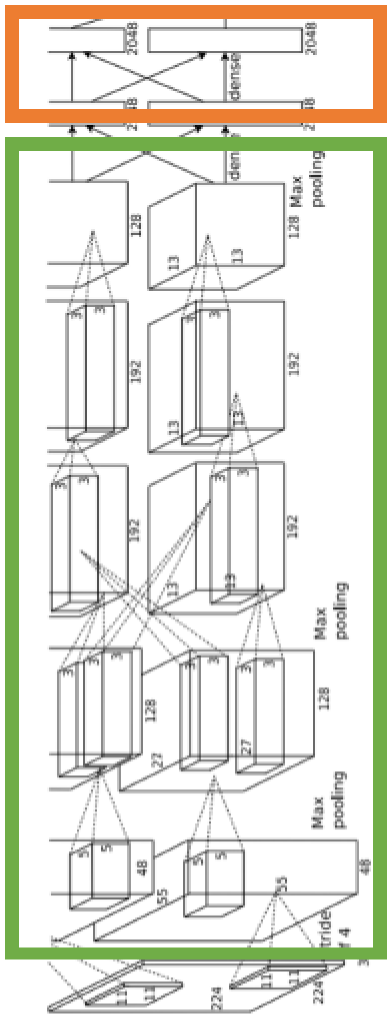


Fast R-CNN Training Example

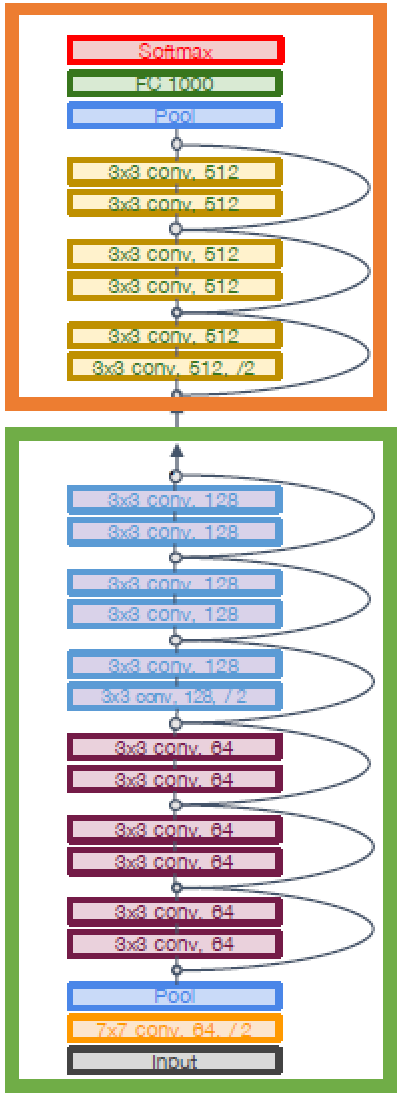


Example Backbone and Per-Region Networks

- 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



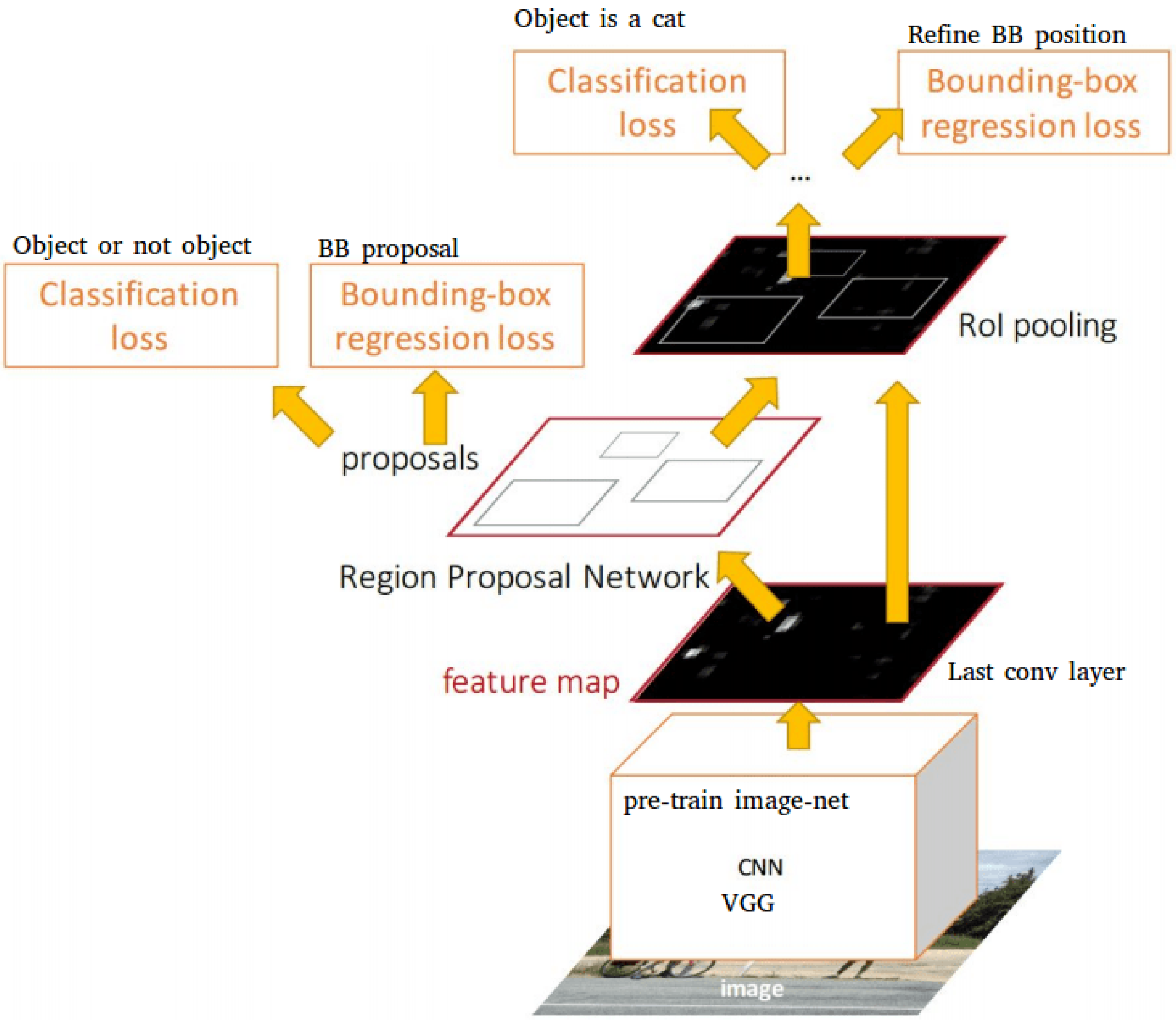
AlexNet



ResNet

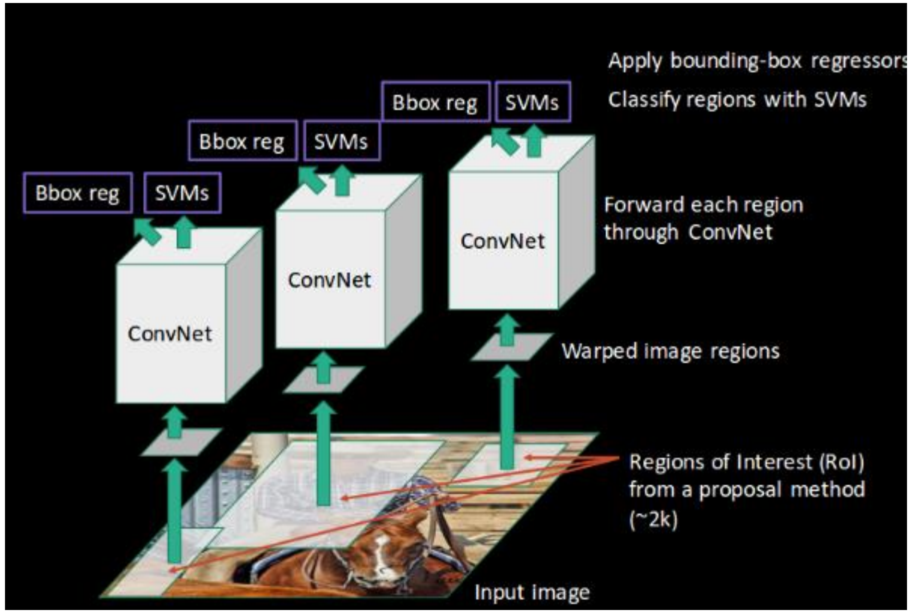
Faster R-CNN

- 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
  - RPN is a learnable CNN that replaces Selective Search used by Fast R-CNN
- Jointly train with 4 losses:
  - 1. RPN classification: anchor box is object / not an object
  - 2. RPN regression: predict transform from anchor box to Bbox proposal
  - 3. Object classification: classify proposals as background / object class
  - 4. Object regression: predict transform from proposal box to object Bbox

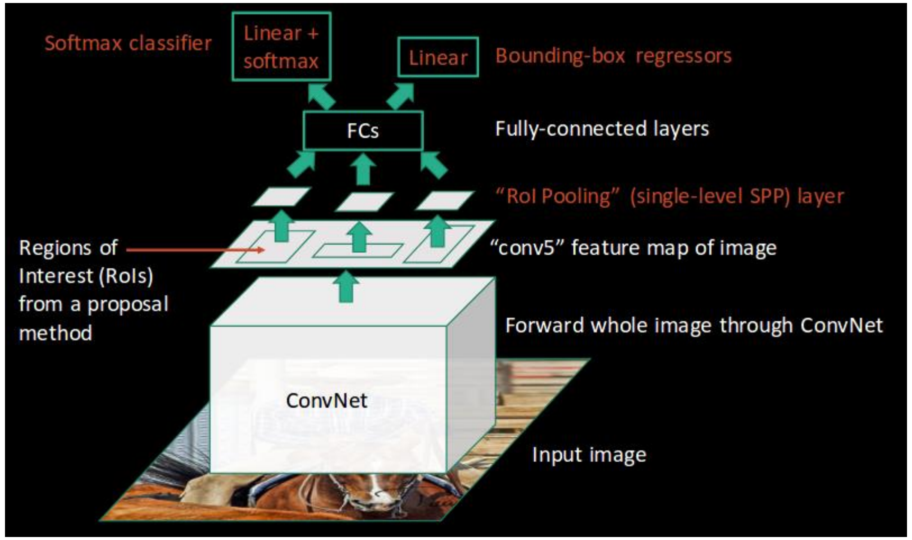


The R-CNN Family

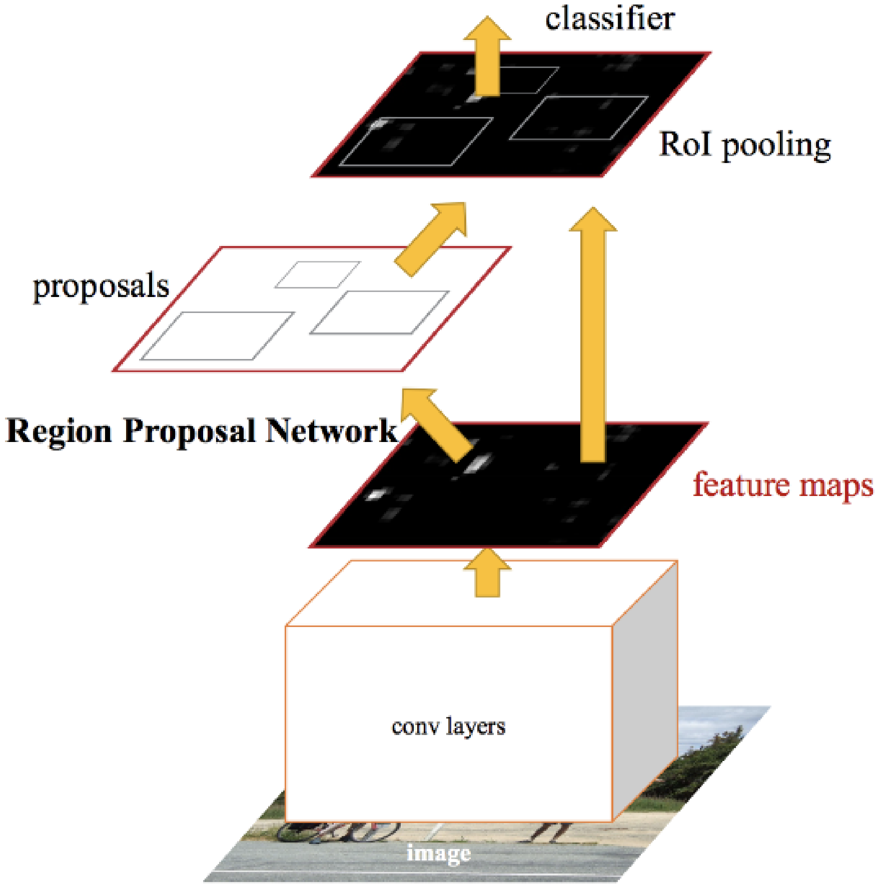
R-CNN



Fast R-CNN



Gerschick, "Fast R-CNN", ICCV 2015



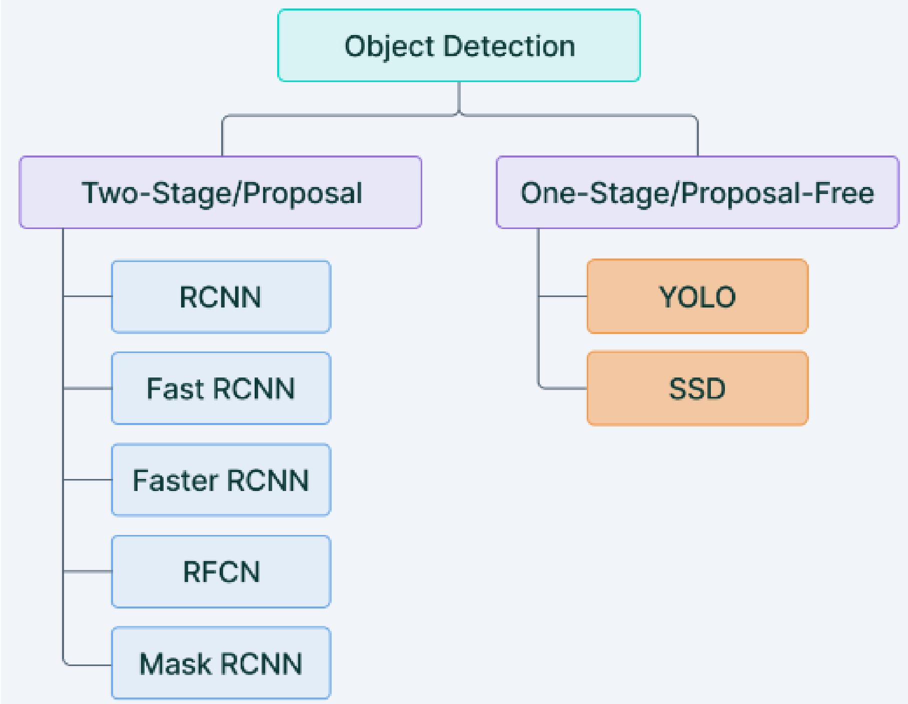
Faster R-CNN

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	

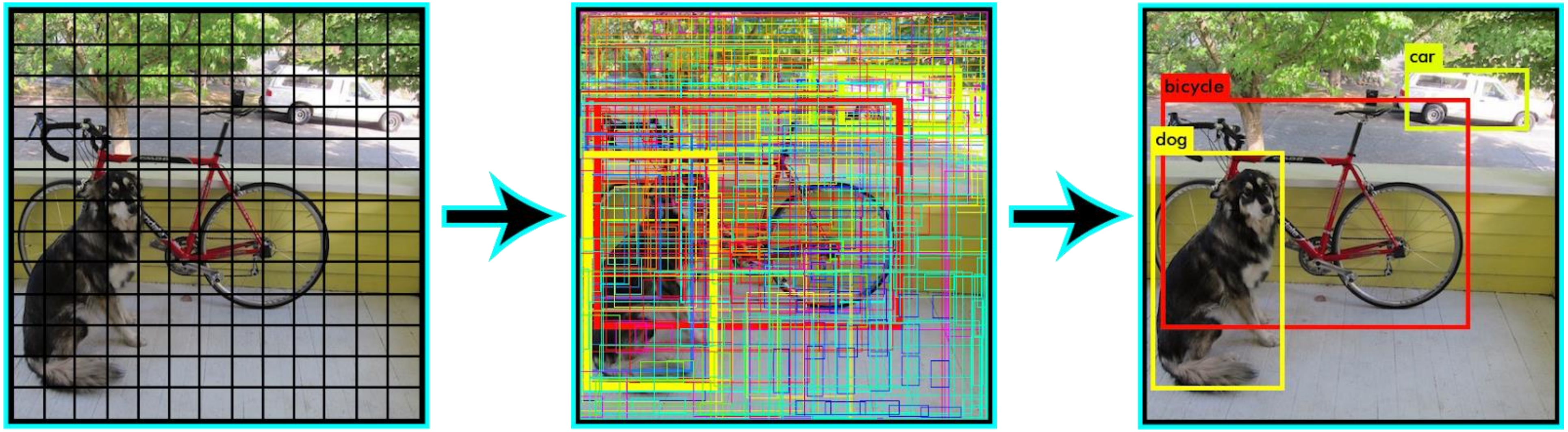
Two-stage vs. One-Stage Detector

- 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



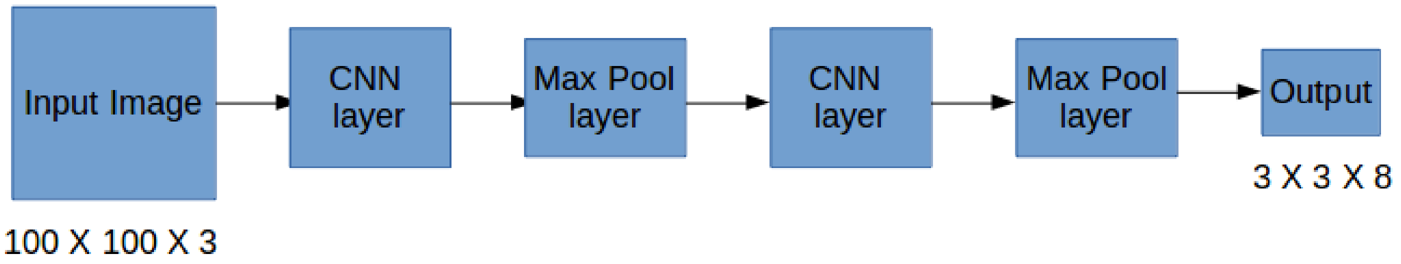
You Only Look Once (YOLO)

- 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 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 computed 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



y =	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3

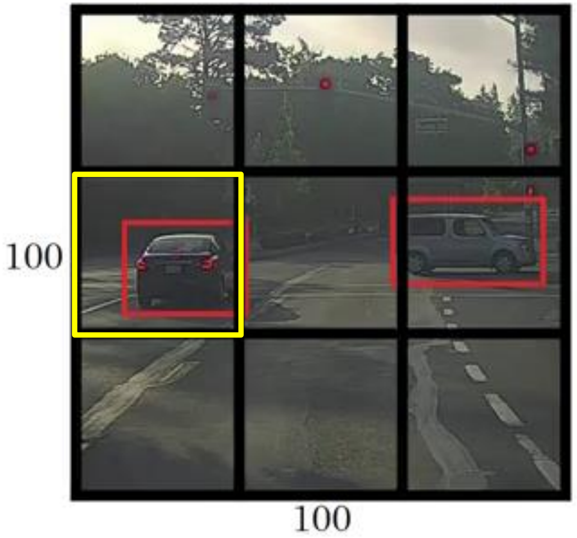
Two Grid Cells

- Since there is no object in this grid, pc=0, and all the other entries are “don’t care”, denoted as “?”

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

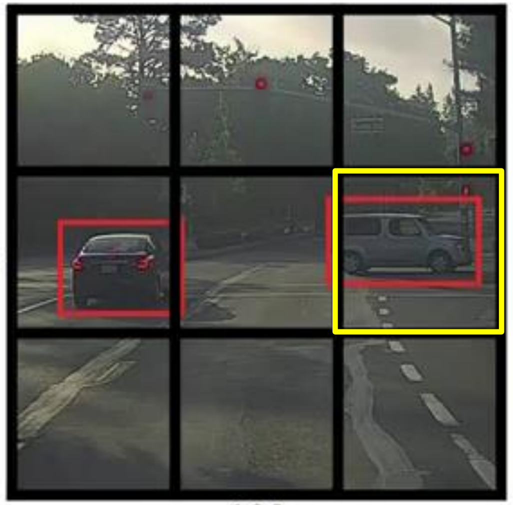
y =	1
	bx
	by
	bh
	bw
	0
	1
	0



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

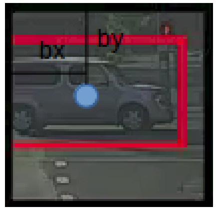
100



100

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

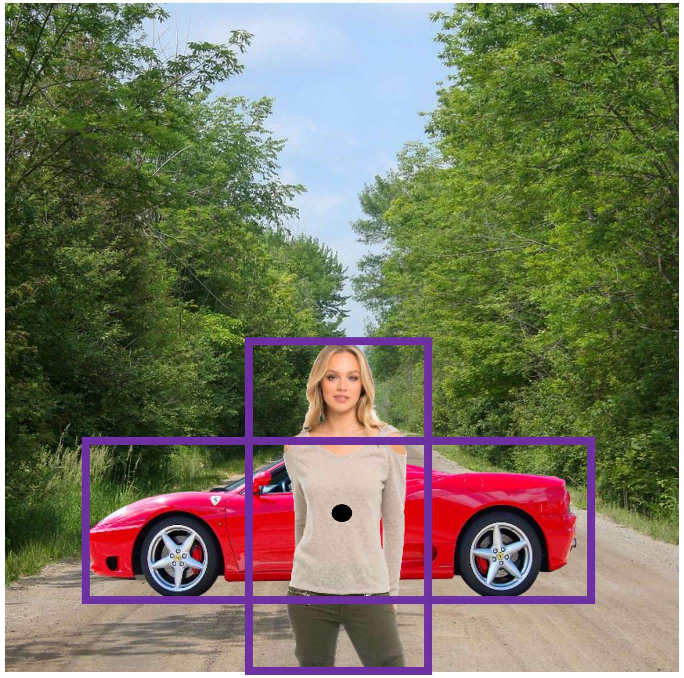
(0,0)



(1,1)

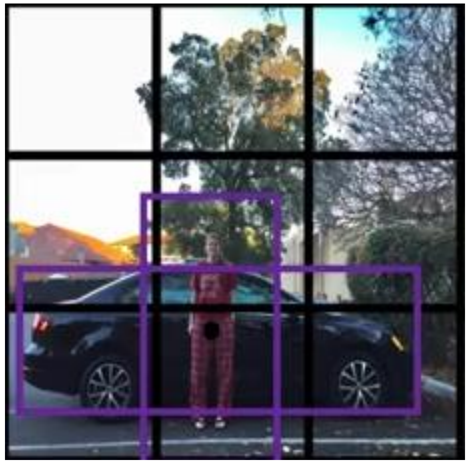
Anchor Boxes

- 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, with better-fitting anchor boxes, which helps ease the downstream Bbox regression task
- Ref: "Finally understand Anchor Boxes in Object Detection (2D and 3D)"
  - https://www.thinkautonomous.ai/blog/anchor-boxes

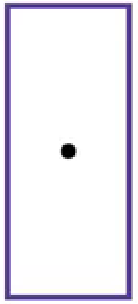


Anchor Boxes

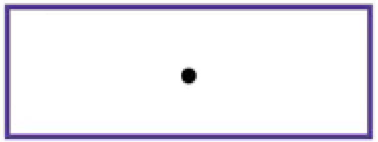
- Suppose we have 2 anchor boxes for each grid cell. Then we can detect at most two objects for each grid cell
- First 8 rows of the y label belong to anchor box 1 and the remaining 8 belong to anchor box 2. The y vector has 16 entries, and the output has dimension 3x3x2x(5+3) =3x3x16 (for each of the 3x3 grid cells, there is a y vector with size 2x8=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 3x3 grid, 2 anchor boxes per grid and the number of classes is 4, then the output has dimension 3x3x2x(5+4)= 3x3x18 (for each of the 3x3 grid cells, there is a y vector with size 2x9=18)
- Suppose we use 4x4 grid, 5 anchor boxes per grid and the number of classes is 5, then the output has dimension 4x4x5x(5+5)=4x4x50 (for each of the 4x4 grid cells, there is a y vector with size 5x10=50)



Anchor box 1:



Anchor box 2:



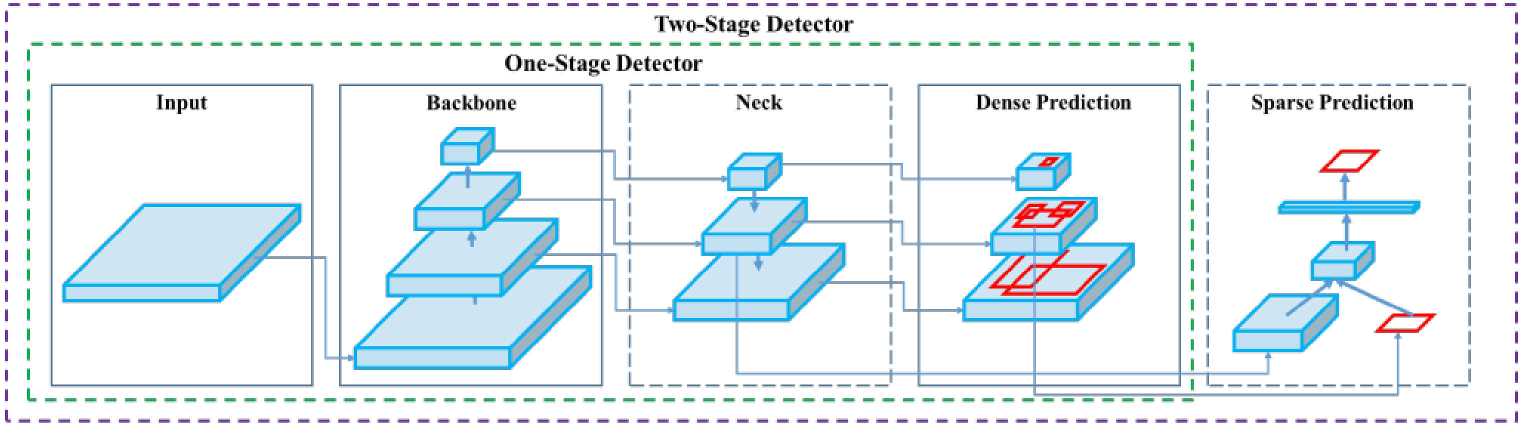
y =	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3
	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3

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
  - Dense prediction (one-stage), sparse prediction (two-stage)



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:

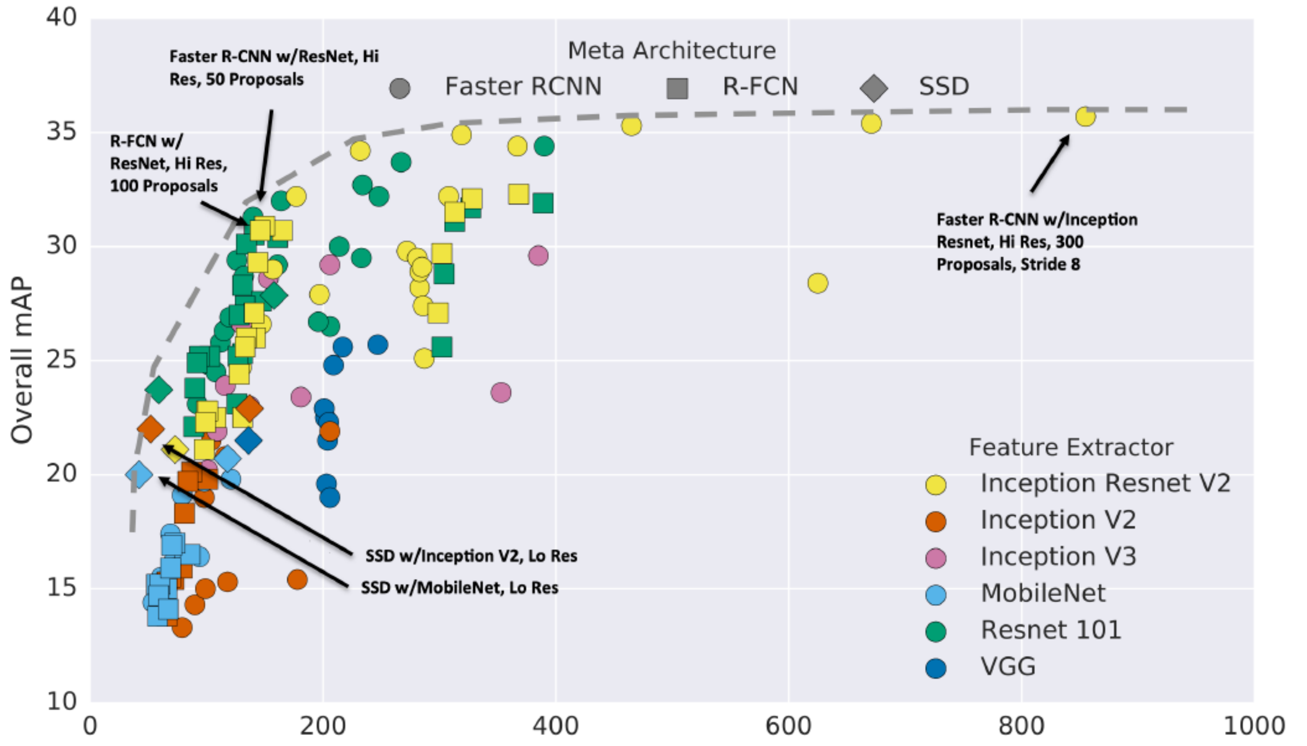
Dense Prediction: { RPN [64], YOLO [61, 62, 63], SSD [50], RetinaNet [45], FCOS [78], … }

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

Figure 2: Object detector.

Performance Comparisons

- 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

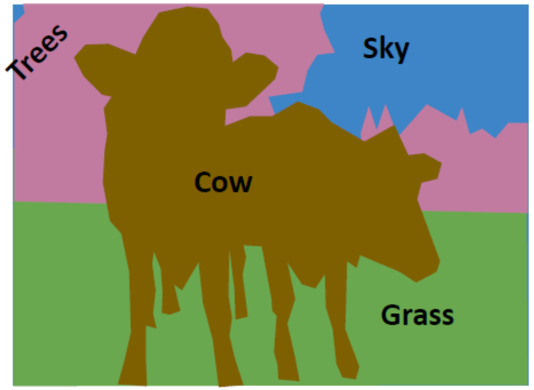


Outline

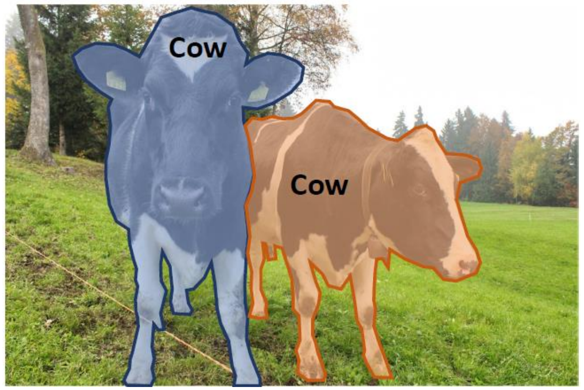
- Object detection
- Segmentation

Types of Segmentation Tasks

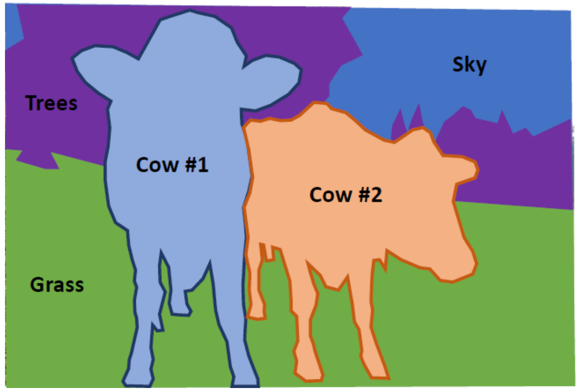
- Things vs. stuff
  - Things: Object categories that can be separated into object instances (e.g. cats, cars, person)
  - Stuff: Object categories that cannot be separated into instances (e.g. sky, grass, water, trees)
- Object Detection vs. Semantic Segmentation vs. Instance Segmentation
  - Object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



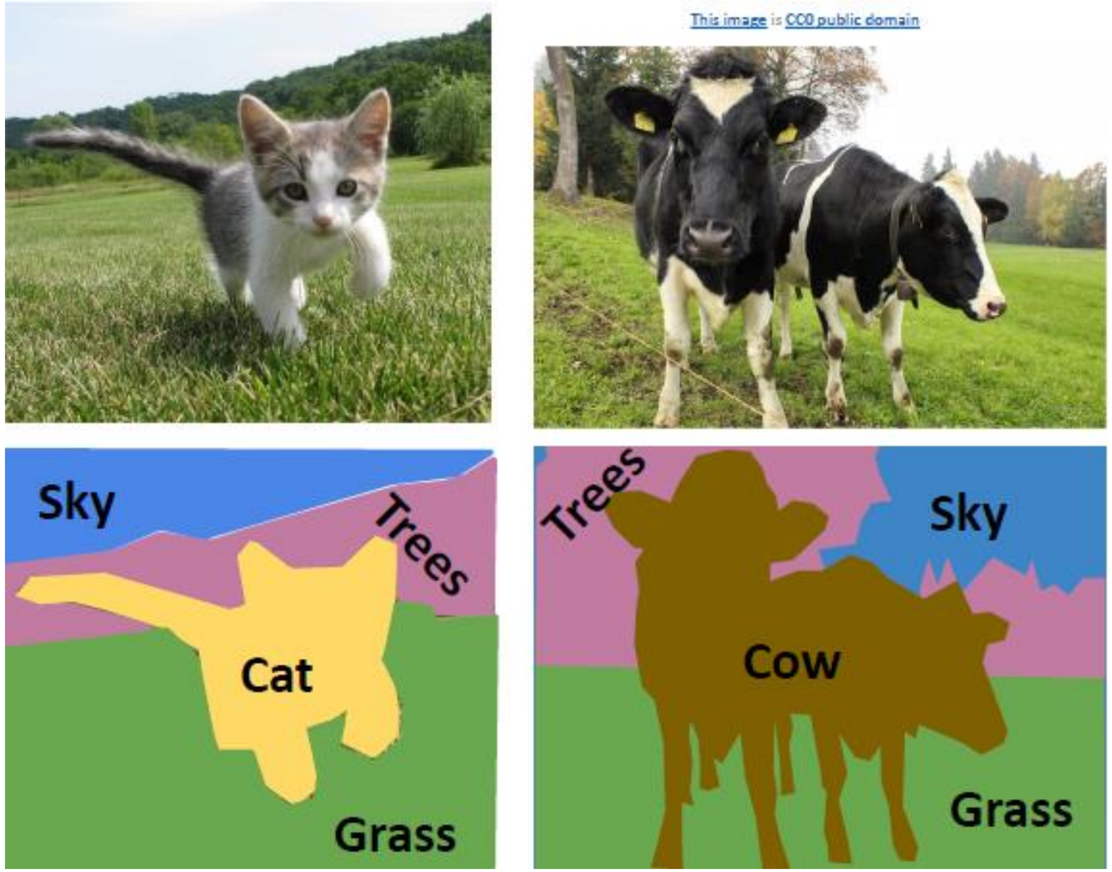
Instance Segmentation



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)



Ground Truth Per-Pixel Labels

segmented

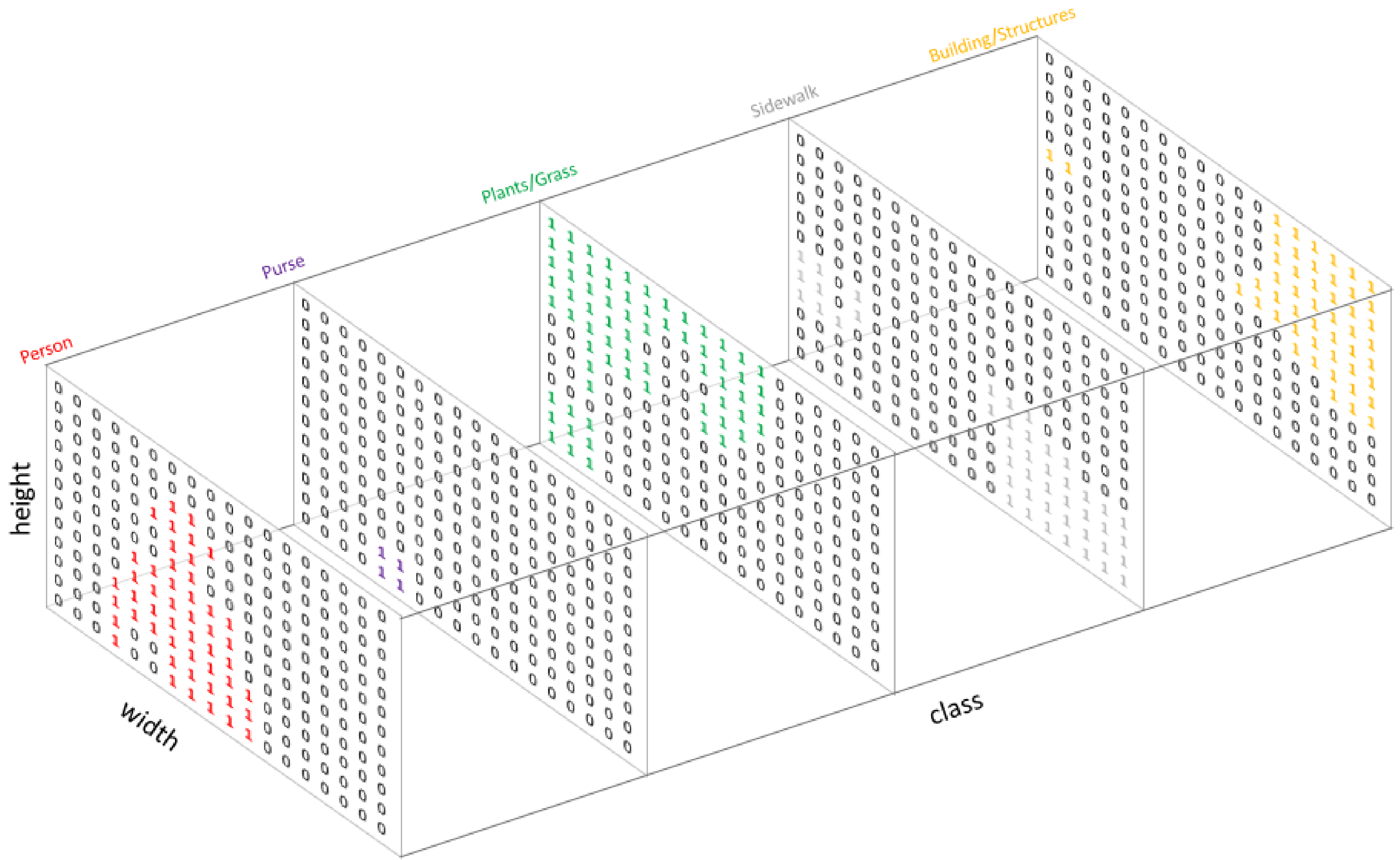
1: Person
2: Purse
3: Plants/Grass
4: Sidewalk
5: Building/Structures

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

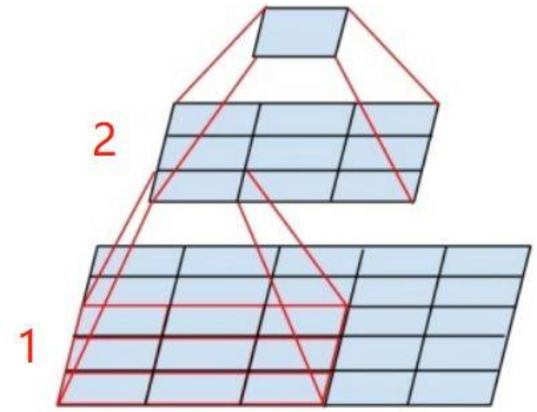
0: Background/Unknown
1: Person
2: Purse
3: Plants/Grass
4: Sidewalk
5: Building/Structures

Ground Truth Per-Pixel Labels

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

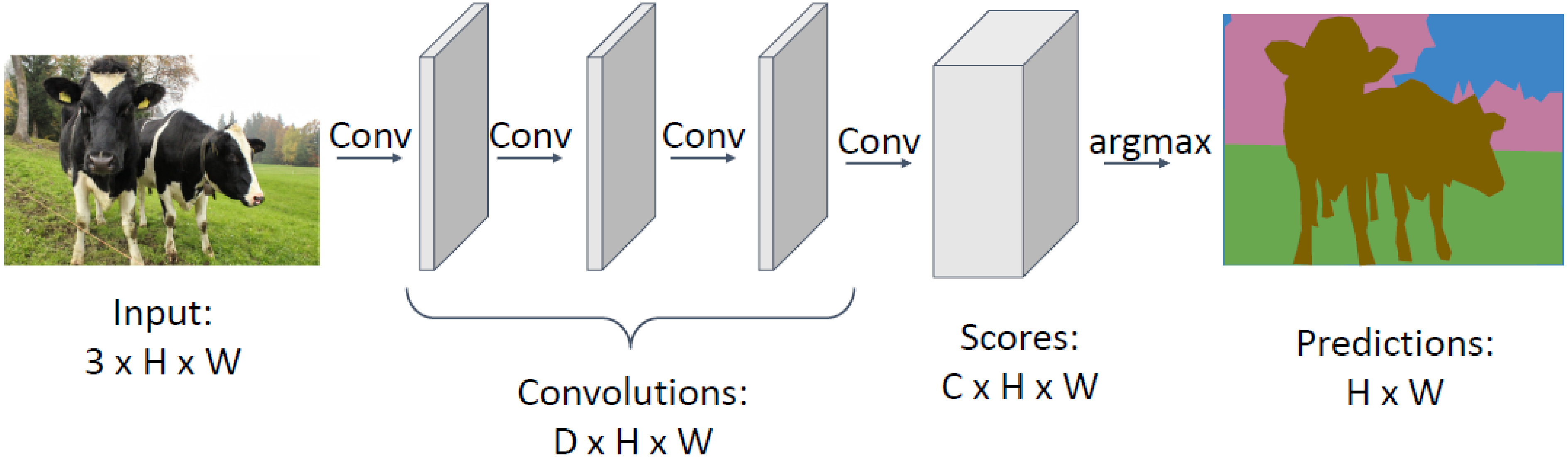


Fully Convolutional Network (FCN)



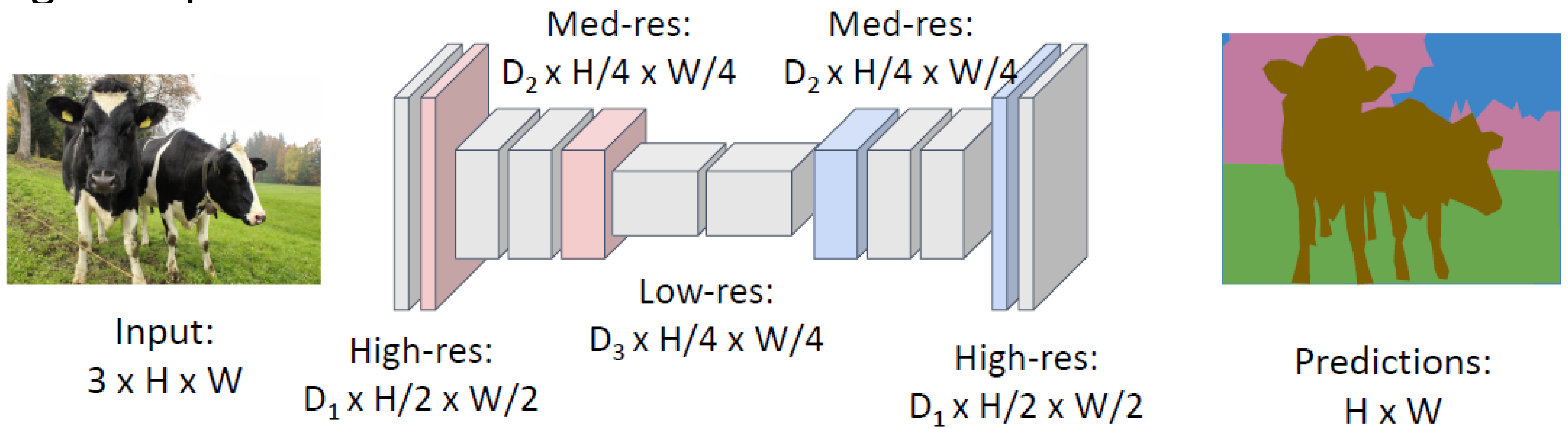
2 stacked 3×3 CONV layers w. padding $P = 1$ have the same effective receptive field as a 5×5 CONV layer

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



More Efficient FCN

- A CNN with CONV layers that perform downsampling followed by upsampling
  - Downsampling with pooling or strided convolution to reduce feature map size. It allows effective receptive field size to grow quickly in the feedforward direction, and also leads to more efficient computation
  - Upsampling with interpolation or transposed convolution to increase feature map size.
    - Methods: Bed Of Nails, Nearest Neighbor, Max-Unpooling, Bilinear/Bicubic Interpolation, Transposed Convolution
  - Downsampling followed by upsampling produces the same-size output as the original input



Bed Of Nails, Nearest Neighbor

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

Bed of Nails

Input
C x 2 x 2

1	2
3	4

→

Output
C x 4 x 4

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

Nearest Neighbor

Input
C x 2 x 2

1	2
3	4

→

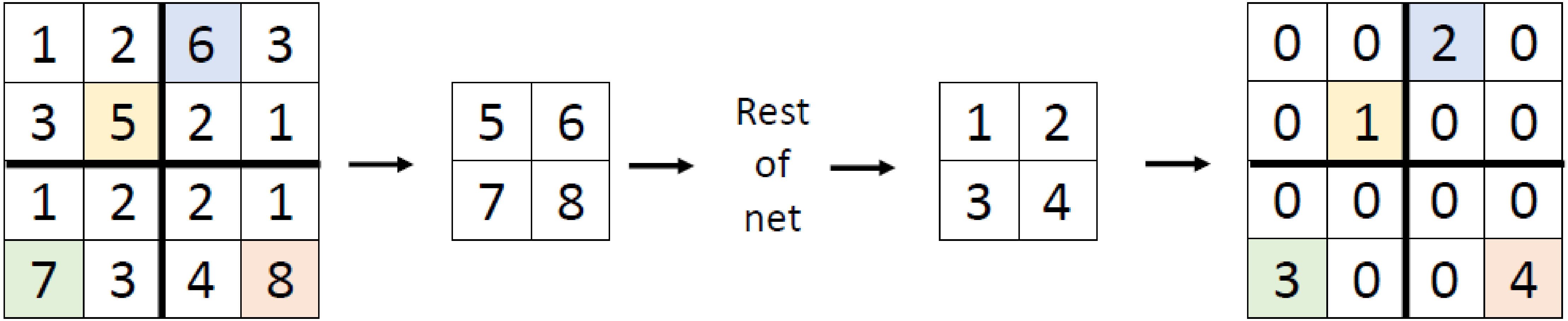
Output
C x 4 x 4

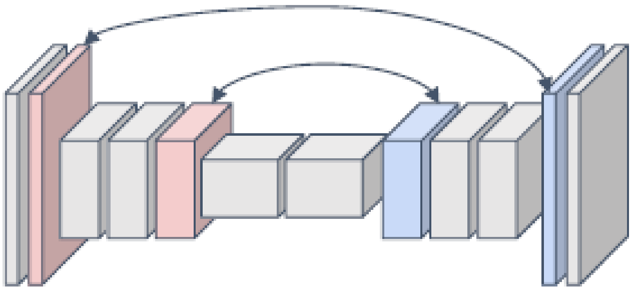
1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Max Unpooling

Max Pooling: Remember which position had the max

Max Unpooling: Place into remembered positions



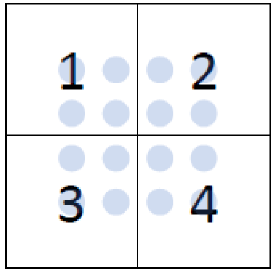


Pair each downsampling layer with an upsampling layer

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

Bilinear/Bicubic Interpolation

- Upsampling from a 2x2 image to a 4x4 image with bilinear or bicubic interpolation
- Each output element is computed as a linear or cubic combination of its closest neighbors to generate smooth outputs

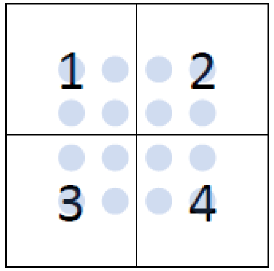


1.00	1.25	1.75	2.00
1.50	1.75	2.25	2.50
2.50	2.75	3.25	3.50
3.00	3.25	3.75	4.00

Input: C x 2 x 2

Output: C x 4 x 4

Bilinear interpolation



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

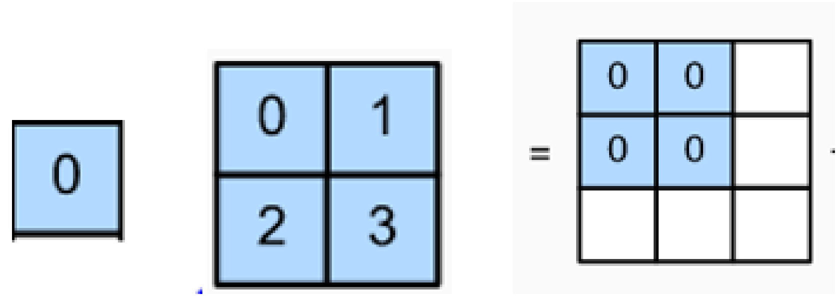
Input: C x 2 x 2

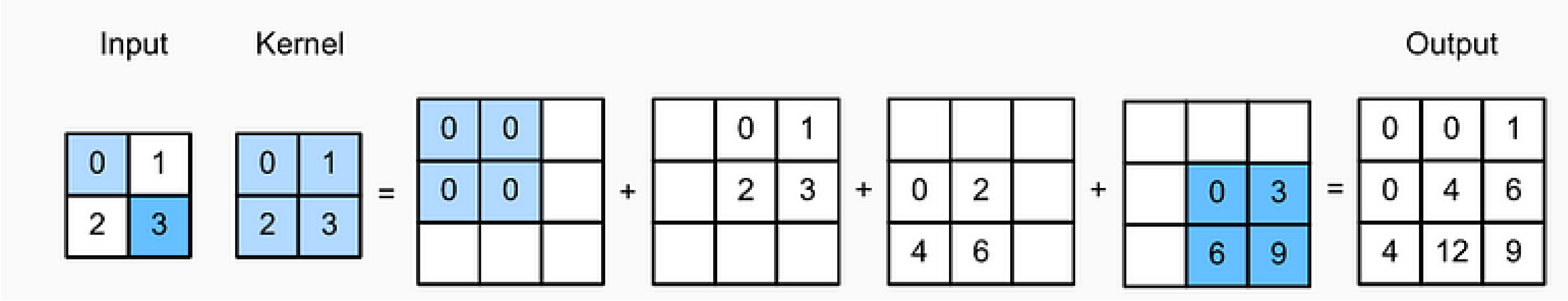
Output: C x 4 x 4

Bicubic interpolation

Transposed Convolution

- To upsample a 2x2 input feature map to a 3x3 output feature map by transposed convolution, we apply a kernel of size 2x2 with unit stride and zero padding
- Right figure: take the upper left element of the input feature map and multiply it with every element of the kernel
- Bottom figure: do it for all the remaining elements of the input feature map, and add the output elements of the over-lapping positions to get output feature map
  - May need to trim top row and left column to get desired output feature map size
- It has many names: Transposed Convolution, Deconvolution, Upconvolution, Fractionally-strided convolution

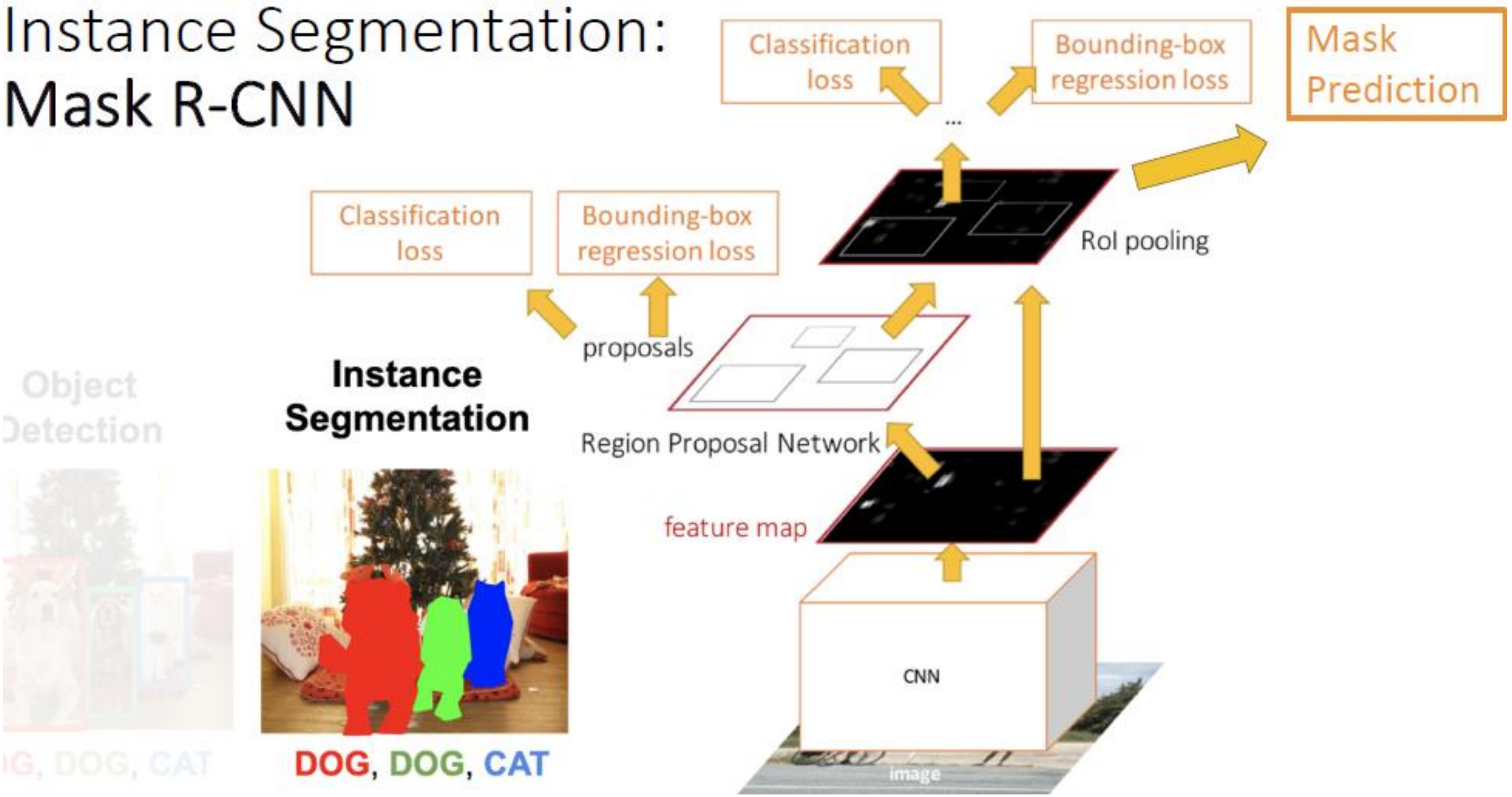




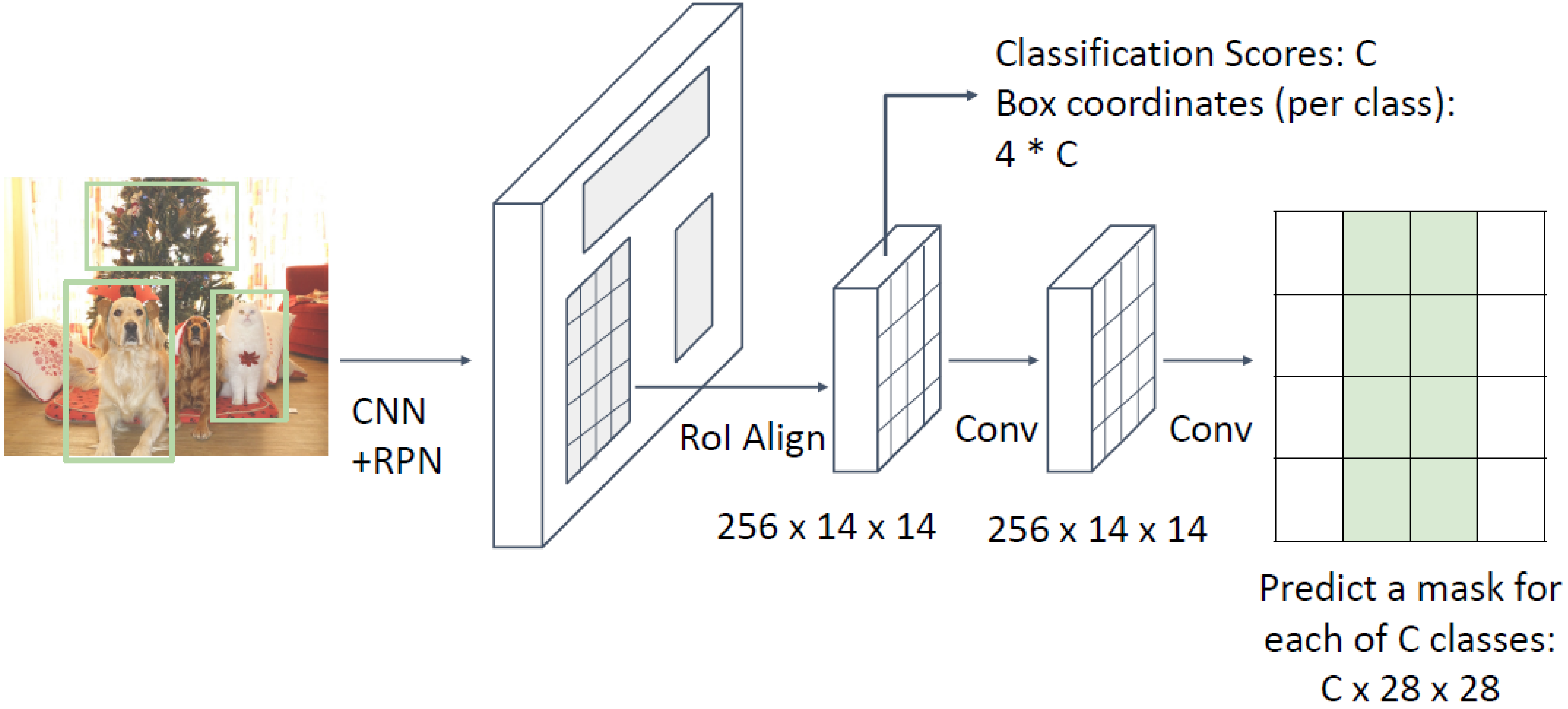
https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba

Mask R-CNN for Instance Segmentation

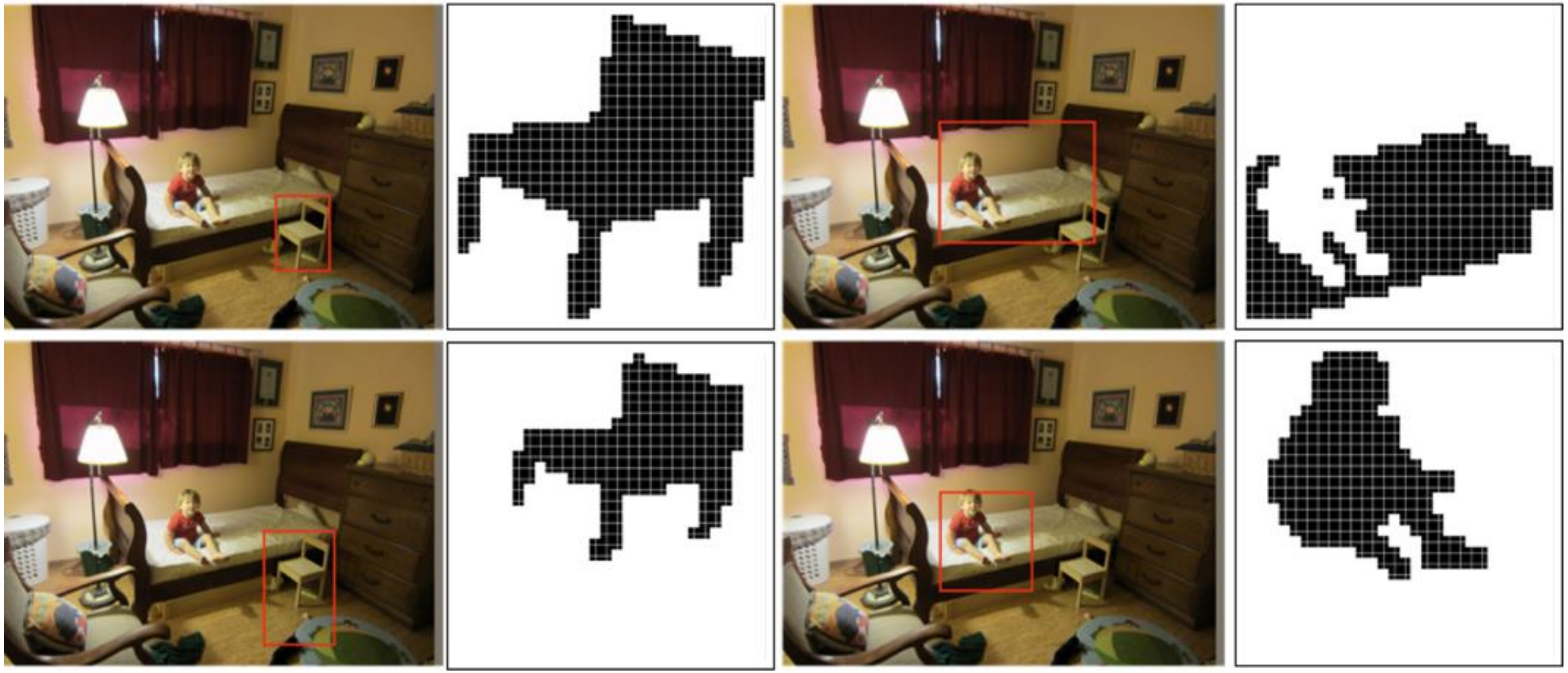
- Add an extra “Mask Prediction” head on top of Faster R-CNN for Object Detection



Mask R-CNN for Instance Segmentation



Example Target Segmentation Masks



Target segmentation mask
for class “chair” in the Bbox

Target segmentation mask
for class “person” in the Bbox