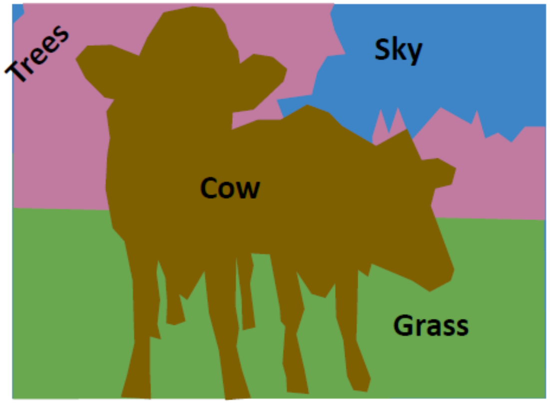
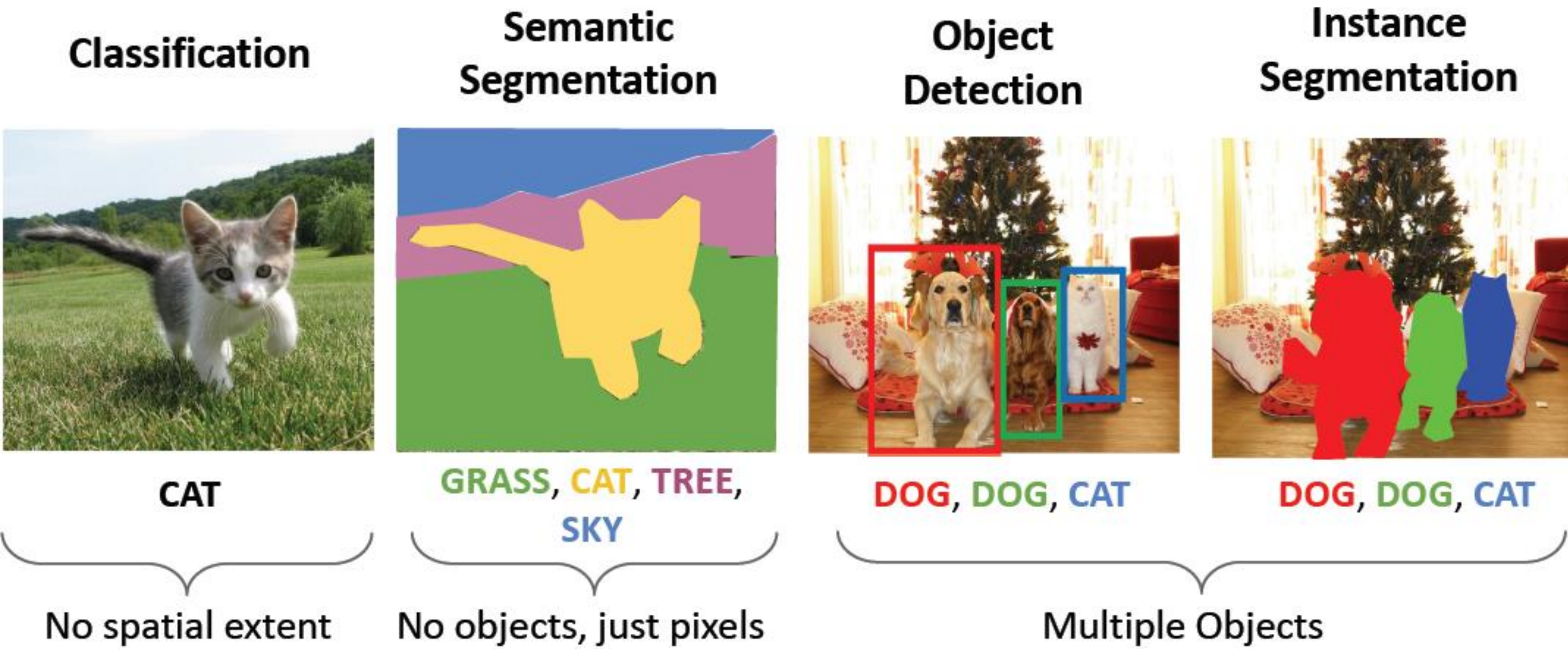
L4.2 Object Detection and Segmentation

Zonghua Gu 2021



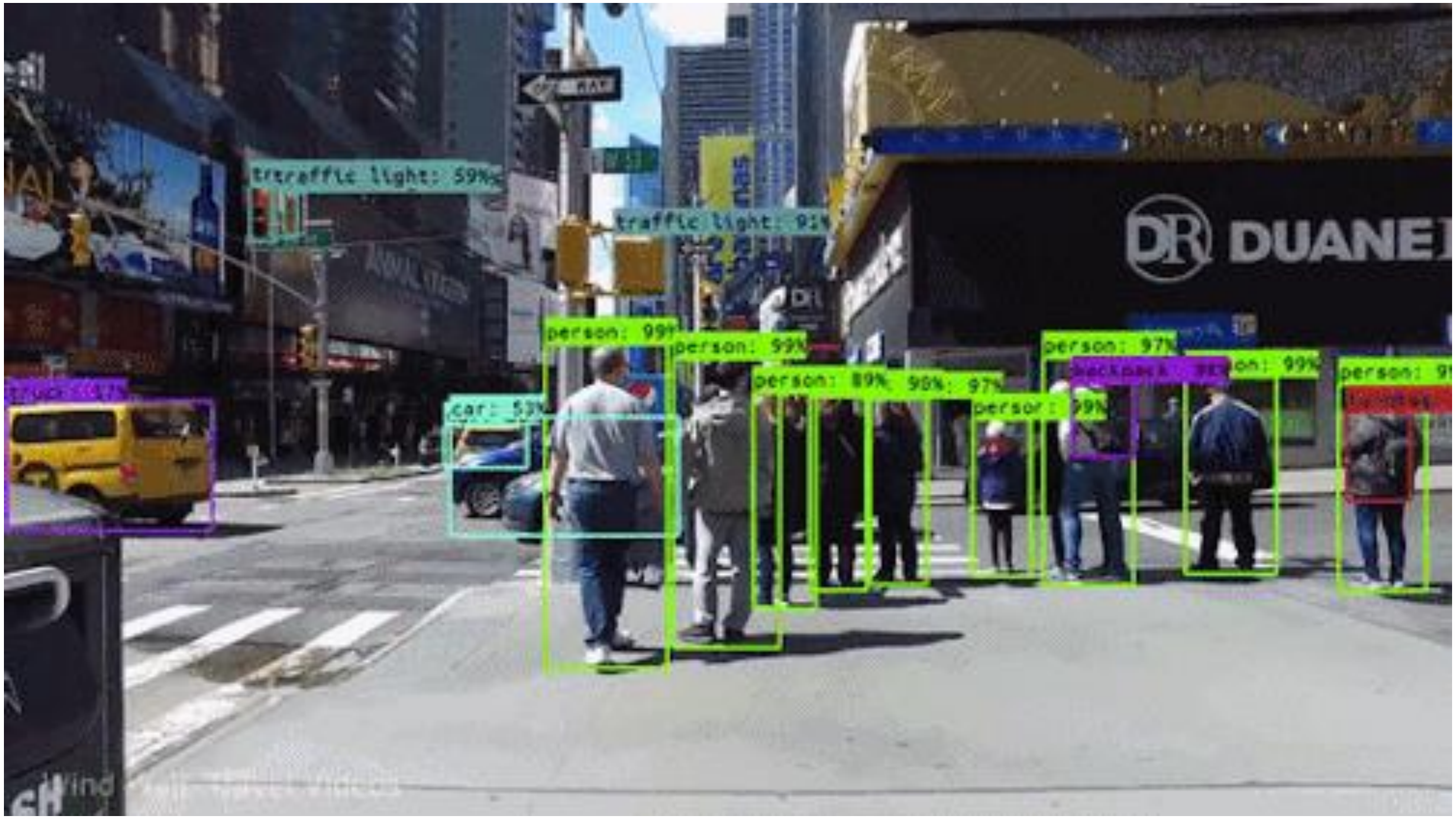
Acknowledgement: Based EECS 498 Deep Learning for Computer Vision, U Michigan

Computer Vision Tasks



Outline

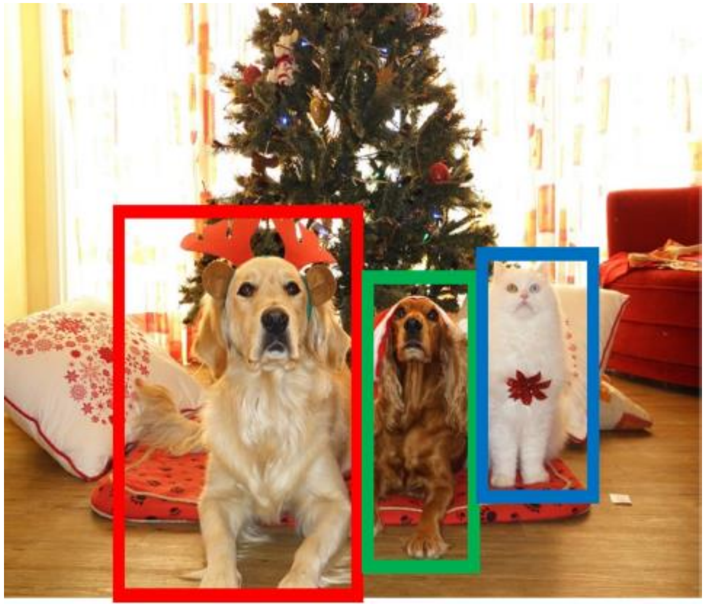
- Object detection
- Segmentation



https://towardsdatascience.com/object-detection-using-deep-learning-approaches-an-end-to-end-theoretical-perspective-4ca27eee8a9a

Object Detection: Task Definition

- Input: Single Image
- Output: a set of detected objects
- For each object predict:
  - 1. Class label (e.g., cat vs. dog)
  - 2. Bounding box (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” (bounding box)
  - Large images: Classification works at 224x224 or lower; need higher resolution for detection, often ~800x600



Single-Object Detection

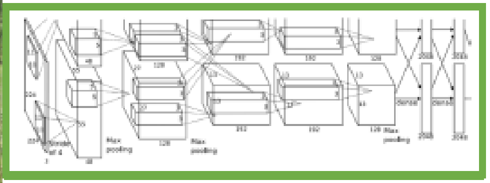
Detecting a single object

"What"

Correct label:
Cat

Often pretrained
on ImageNet
(Transfer learning)

This image is CC0 public domain



Fully
Connected:
4096 to 1000

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

Softmax
Loss

Multitask
Loss

Weighted
Sum

Loss

Vector:
4096

Fully
Connected:
4096 to 4

Box
Coordinates
(x, y, w, h)

L2 Loss

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

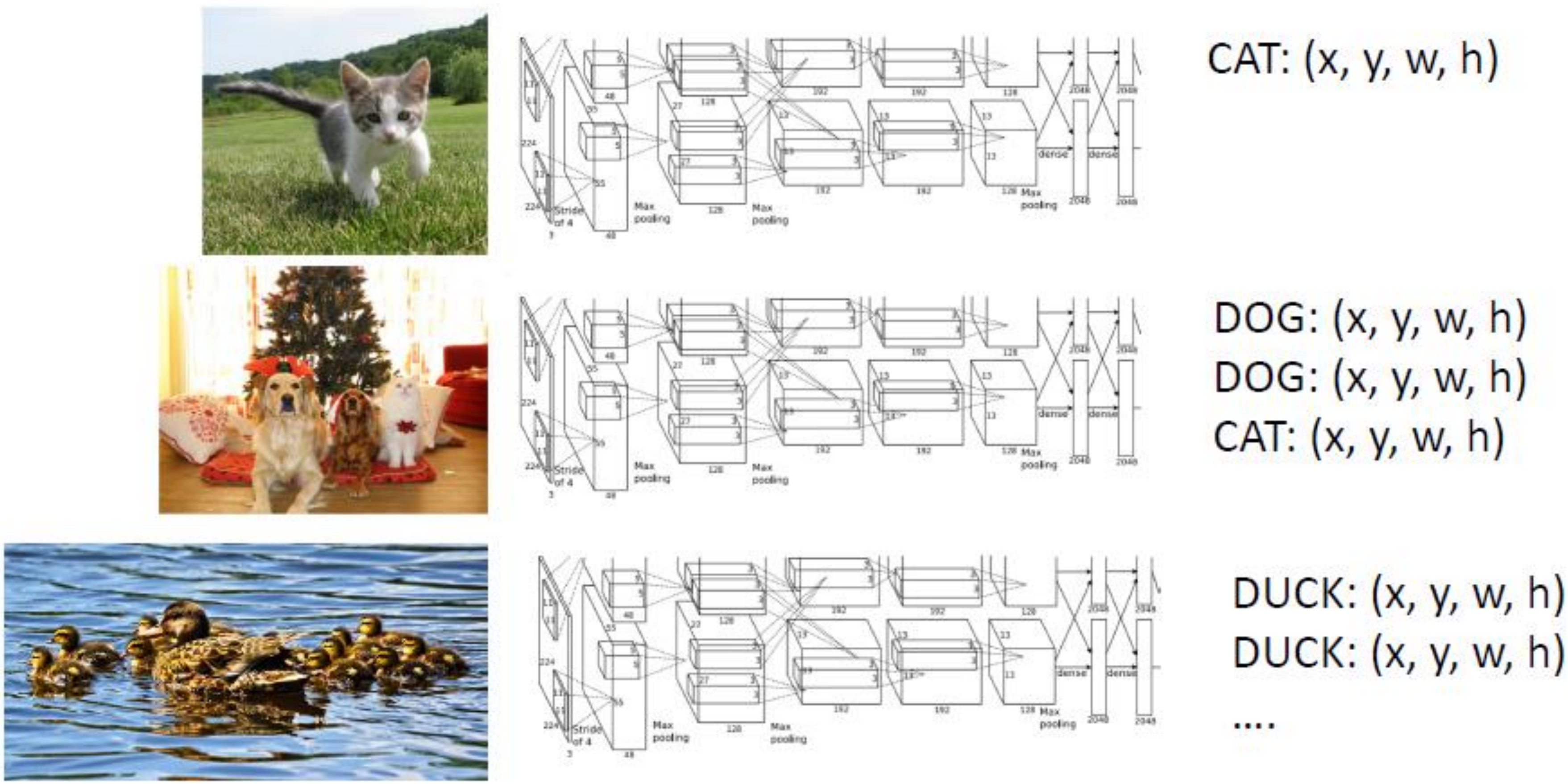
Treat localization as a regression problem!

Problem: Images can have more than one object!

"Where"

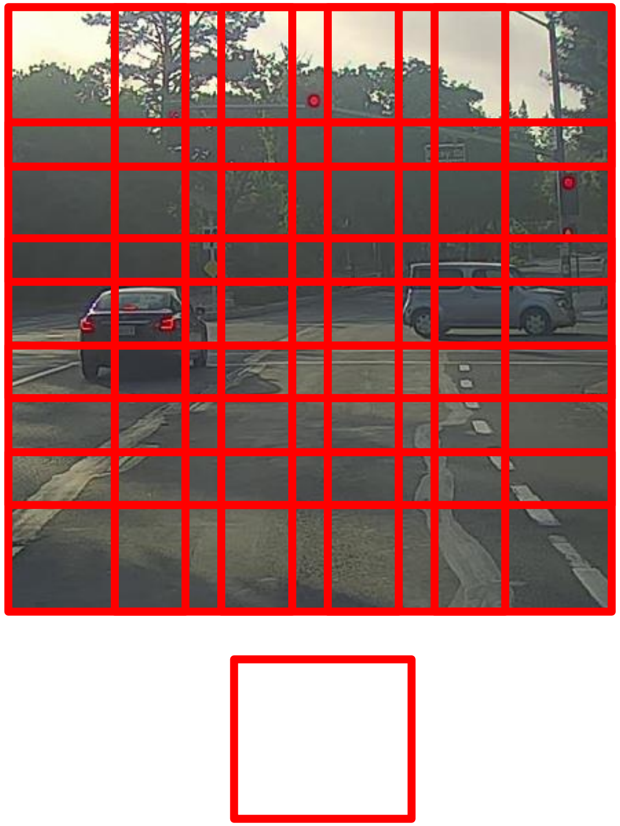
Multi-Object Detection

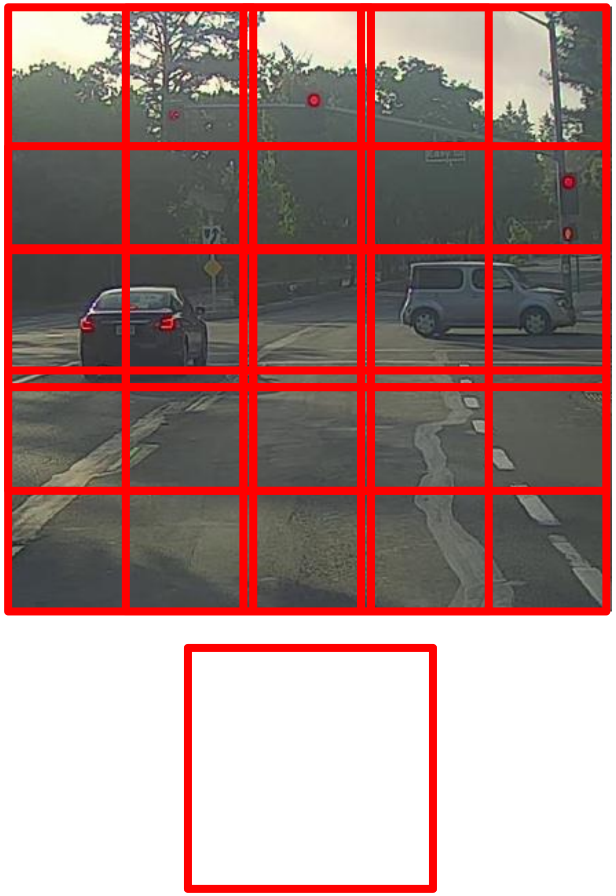
- Needs to predict 4 numbers for each object bounding box (x, y, w, h)
  - (x, y) are coordinates of the box center; (w, h) are its width and height
- 4N numbers for N objects



Detecting Multiple Objects: Sliding Window

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





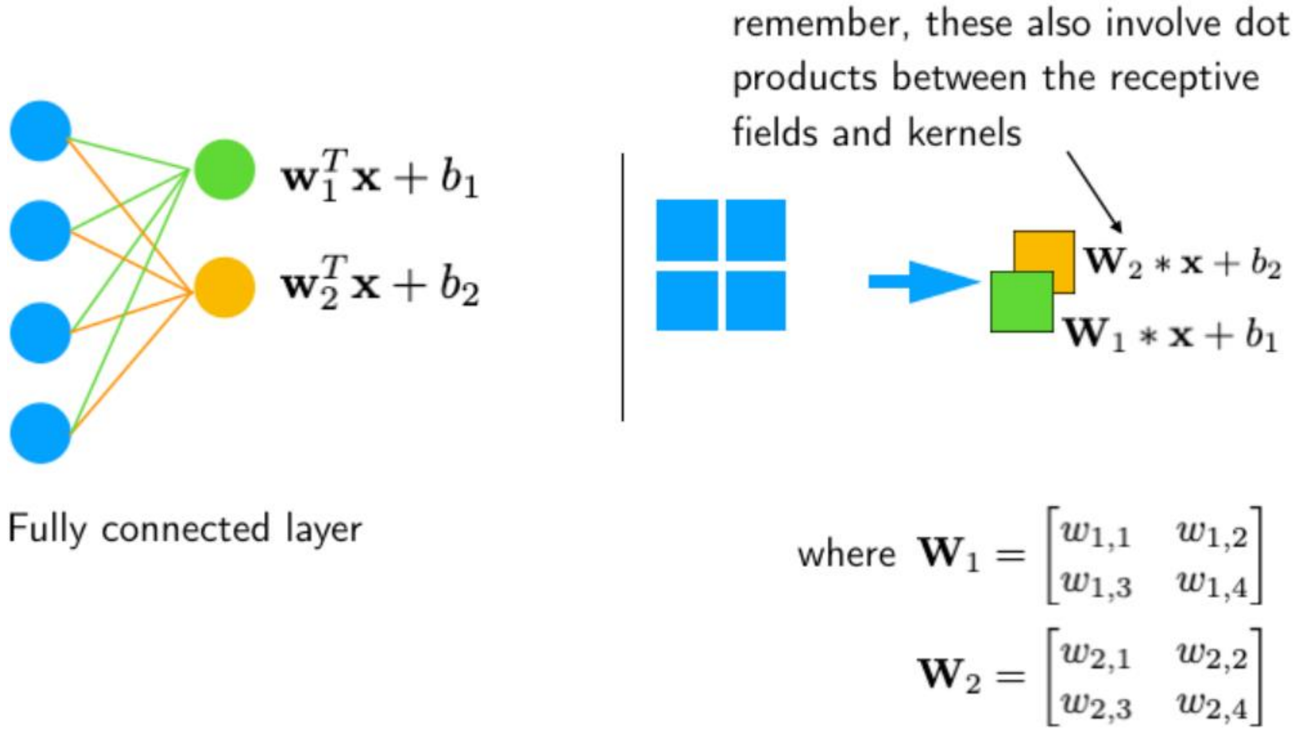
https://www.coursera.org/learn/convolutional-neural-networks/lecture/VgyWR/object-detection

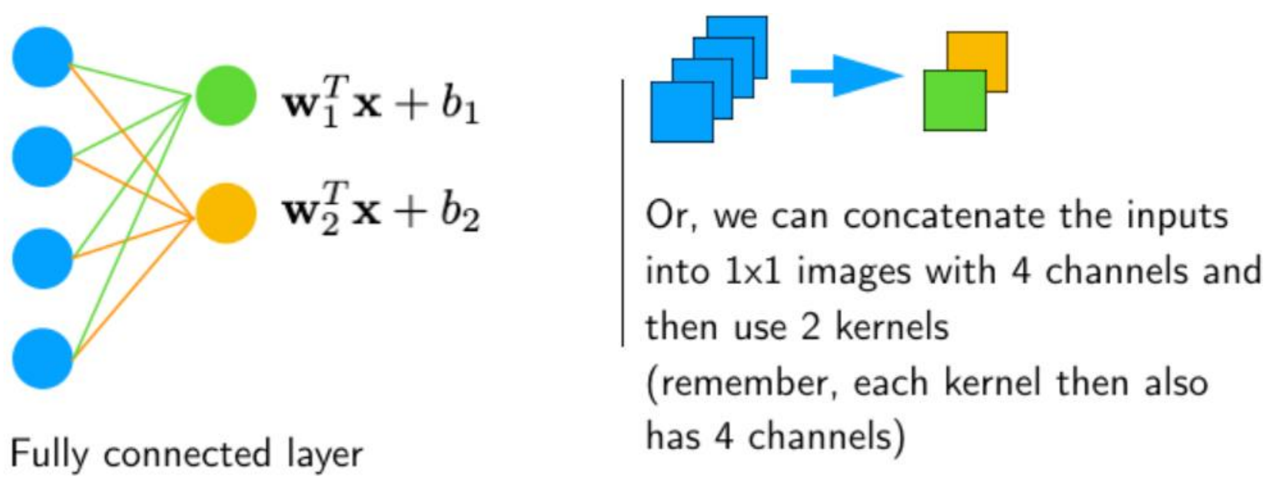
Sliding Window Computational Complexity

- 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!

Tuning a FC layer into an equivalent CONV layer

- Two methods:
- 1) Upper left: With input volume $N_1 \times N_1 \times D_1$, set the filter volume $N_1 \times N_1 \times D_1$, i.e., $F = N_1$, stride $S = 1$, no pad. Then each filter generates a single output in the next layer (since $N_2 = \frac{1}{S}(N_1 - F) + 1 = 1$). Set the number of filters to be the number of neurons in the next layer
- 2) Lower left: Convert the input volume $N_1 \times N_1 \times D_1$ into $1 \times 1 \times (N_1 * N_1 * D_1)$. set the filter volume$1 \times 1 \times (N_1 * N_1 * D_1)$, i.e., $F = 1$, stride $S = 1$, no pad. Then each filter generates a single output in the next layer (since $N_2 = \frac{1}{S}(1 - 1) + 1 = 1$). Set the number of filters to be the number of neurons in the next layer

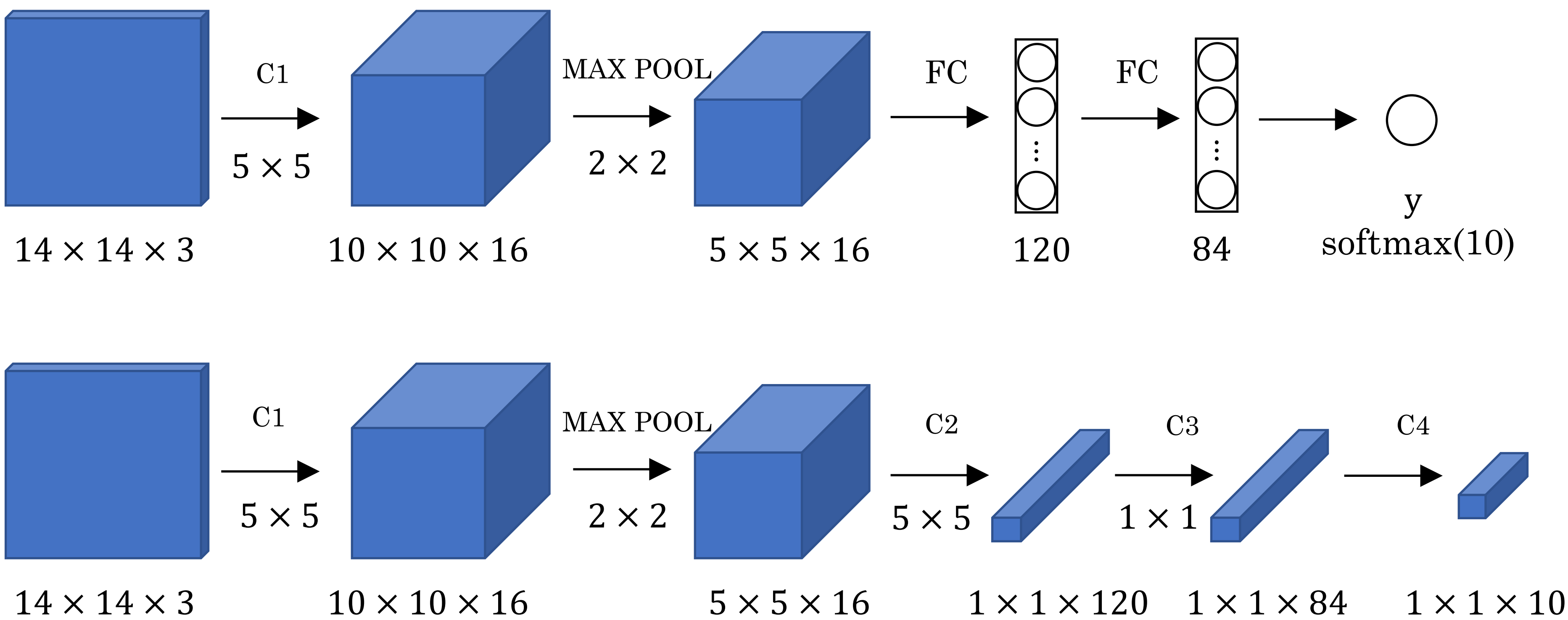




https://sebastianraschka.com/faq/docs/fc-to-conv.html

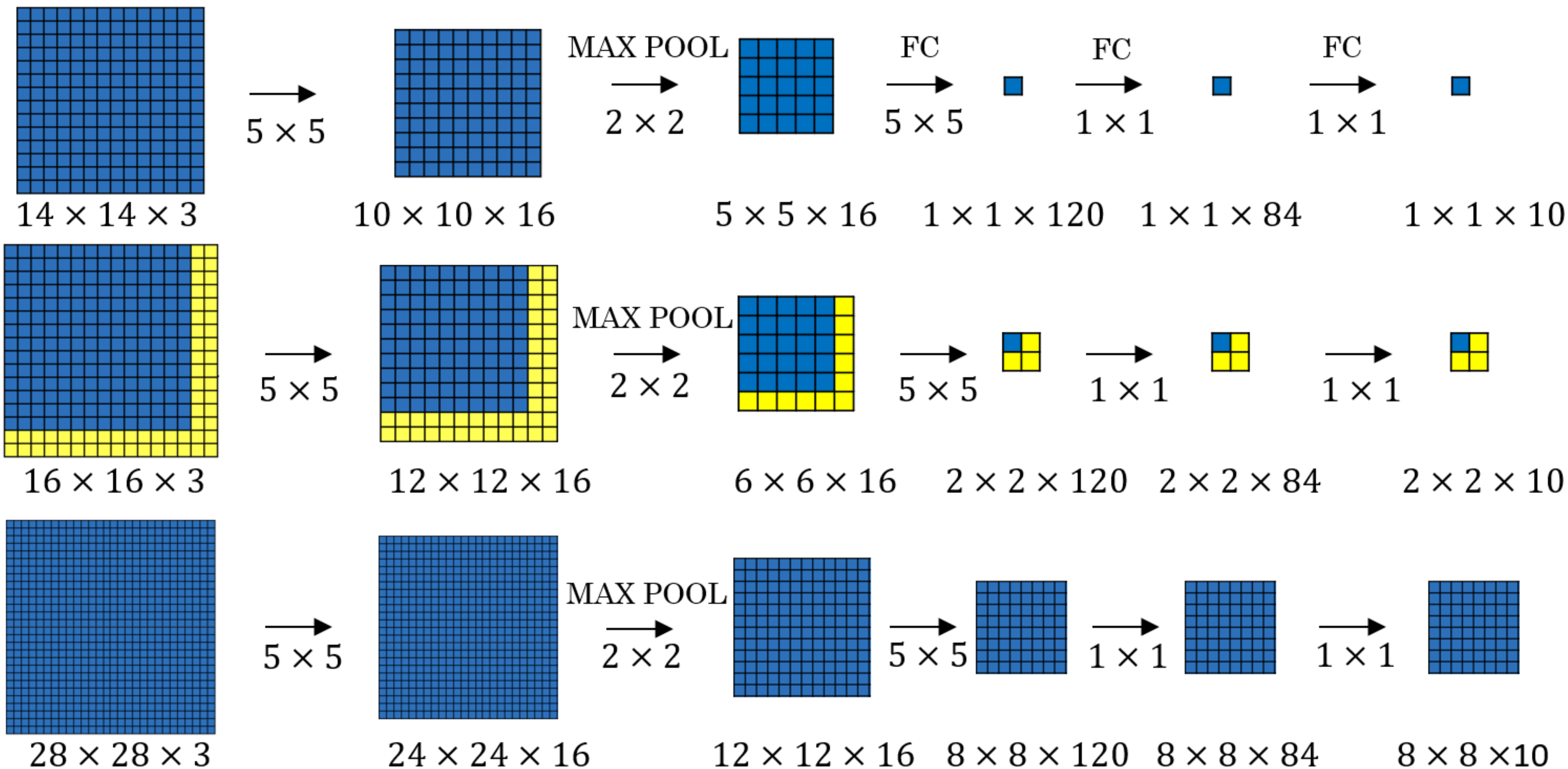
FC-to-CONV Conversion Applied

- Recall: at each CONV layer, each filter implicitly has the same depth as its input volume, and the number of filters implicitly equals the depth of its output volume
  - e.g., layer C2 has 120 $5 \times 5 \times 16$ filters; layer C3 has 84 $1 \times 1 \times 120$ filters; layer C4 has 10 $1 \times 1 \times 84$ filters



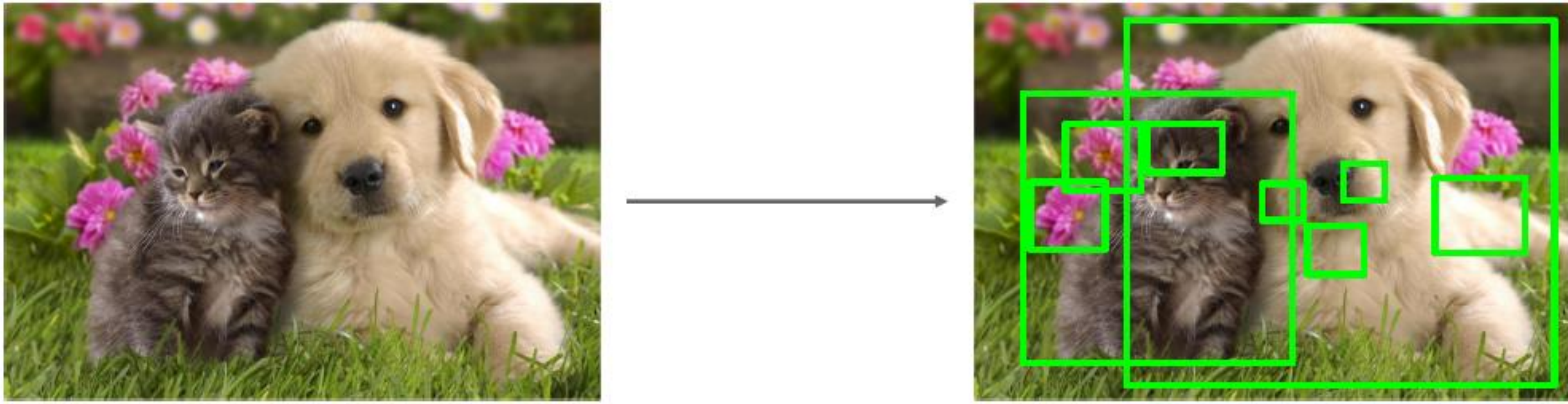
Convolution Implementation of Sliding Windows

- If the input image is larger than the CNN's input size, then this transformation allows us to "slide" the entire CNN across many spatial positions in the larger input image, and generate multiple SoftMax output vectors in one forward pass, whereas the original architecture with FC layers only computes one SoftMax output vector in one forward pass. This improves computation efficiency by reducing redundant computations between overlapping sliding windows
- Middle row: outputs $2 * 2 = 4$ probability vectors of size 4, corresponding to each of the $2 * 2 = 4$ possible positions of sliding a 14×14 box across the 16×16 input image with stride 2
- Bottom row: outputs $8 * 8 = 64$ probability vectors of size 4, corresponding to each of the $8 * 8 = 64$ possible positions of sliding a 14×14 box across the 28×28 input image with stride 2



Region Proposals

- Generating region proposals: find a small set of boxes that are likely to cover all objects, based on heuristics (no learning): e.g., look for “blob-like” image regions
  - Relatively fast to run: e.g. Selective Search gives 2000 region proposals in a few seconds on CPU

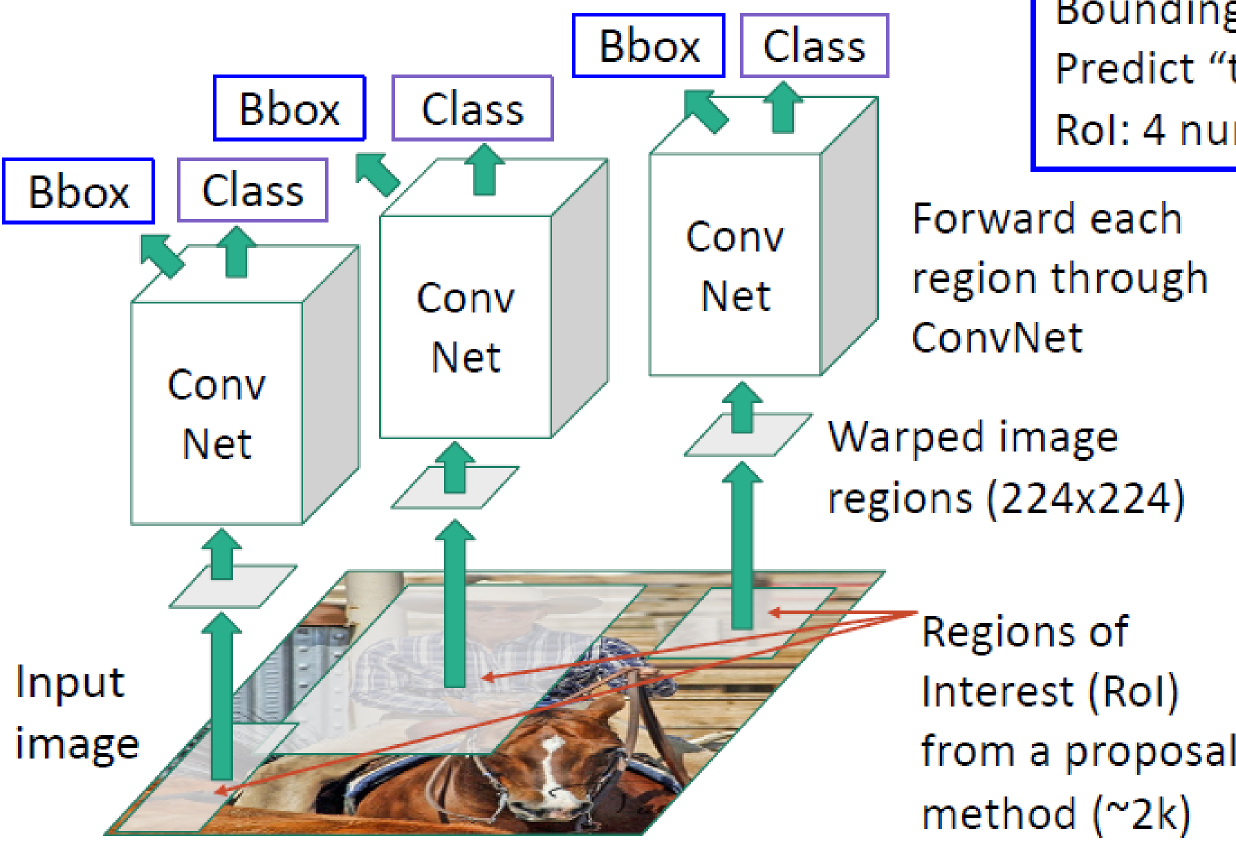


Important

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 bounding box and class label for each region
- Bounding box regression: transform each region proposal with learnable parameters (t_x, t_y, t_h, t_w) into a better bounding box

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)

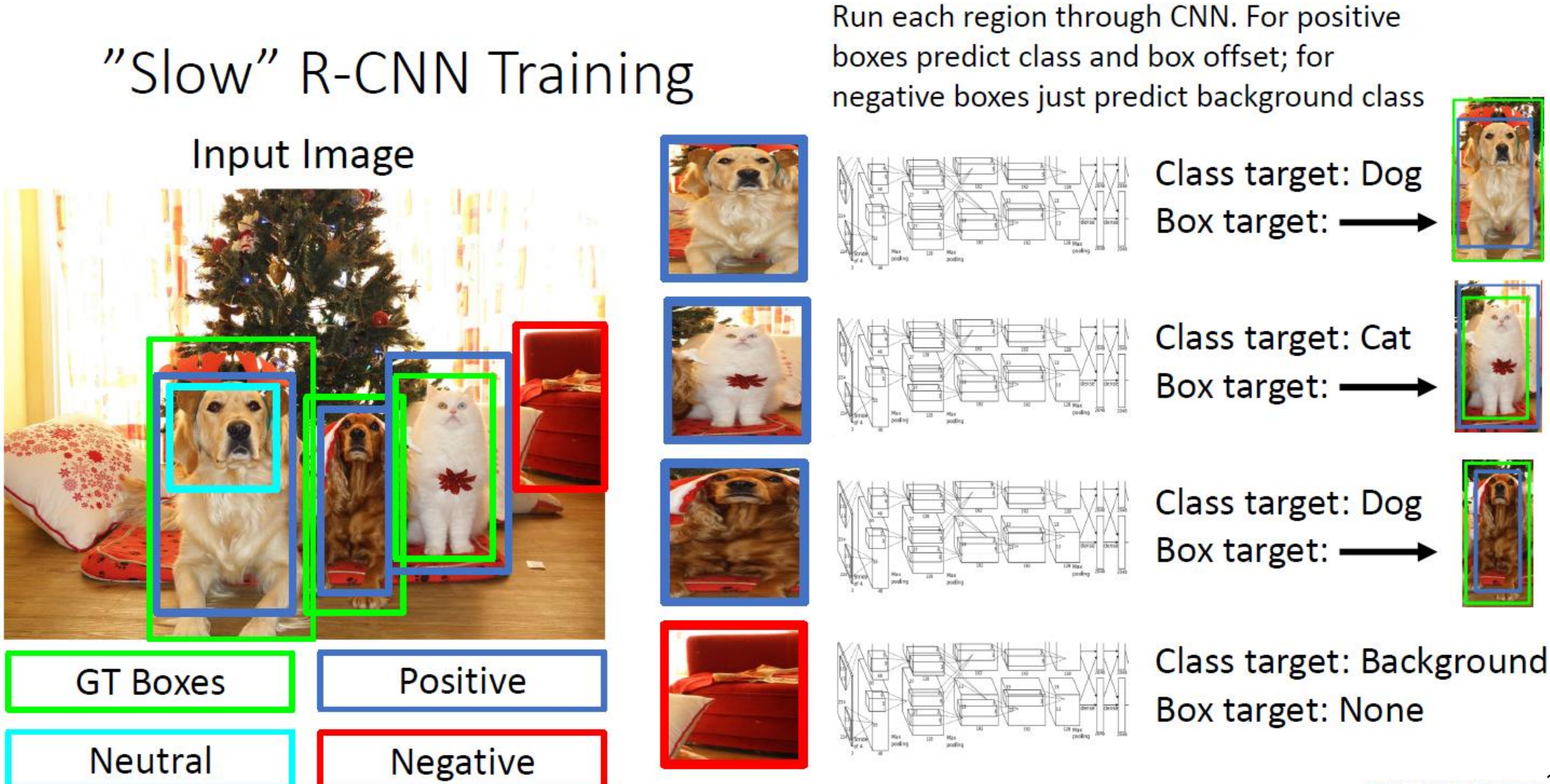
Translate relative to box size:
$b_x = p_x + p_w t_x \qquad b_y = p_y + p_h t_y$

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

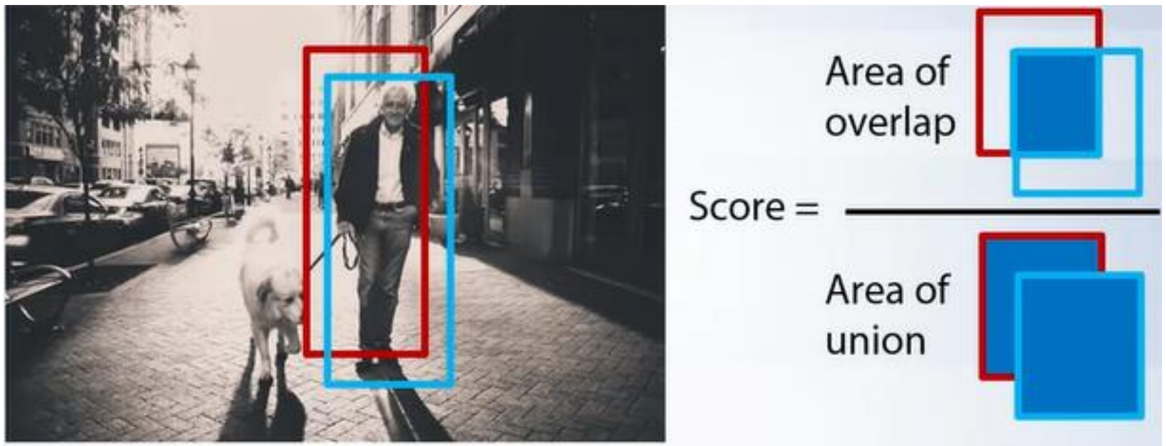
- Categorize each region proposal as positive, negative, or neutral based on overlap with ground-truth boxes
- Crop pixels from each positive and negative proposal, resize to 224 x 224
- Use the CNN for Bbox regression and classification for positive boxes; only 1-class prediction for negative boxes



R-CNN: Test Time

- 1. Run region proposal method to compute ~2000 region proposals
- 2. Resize each region to 224×224 (tunable hyperparams) and run independently through the CNN to predict class scores and Bbox transform
- 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

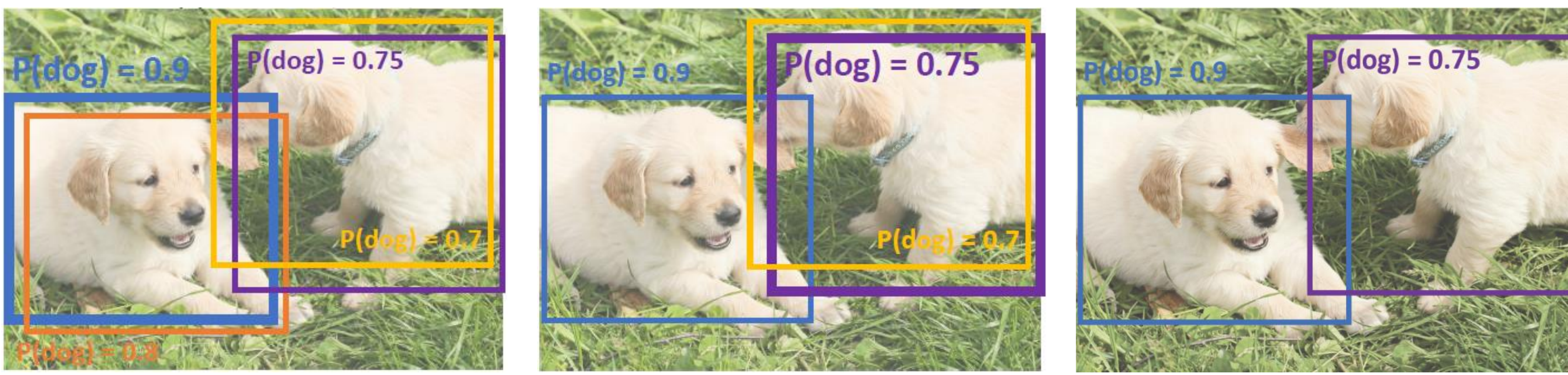
Detection Criteria (Intersection Over Union, IOU)



- 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)

- NMS discards (suppresses) overlapping object boxes except the one with the maximum classification score
- 1. For each output class
  - 1.1 Select next highest-scoring box b and output it as a prediction
  - 1.2 Discard any remaining boxes b' with IoU(b, b') > threshold
  - 1.3. If any boxes remain, GOTO 1.1
- 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

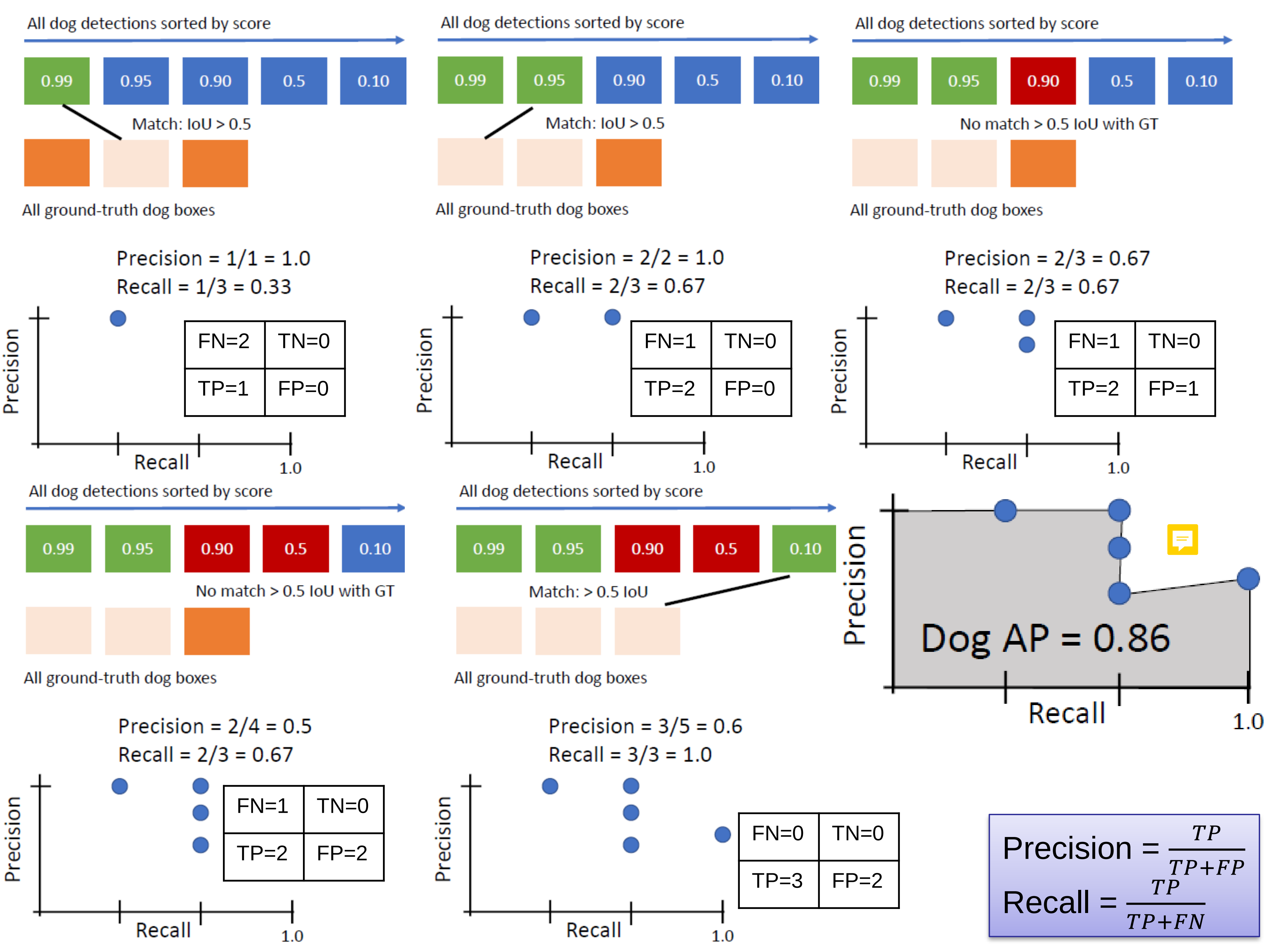


IoU(■, ■) = **0.78**
IoU(■, ■) = 0.05
IoU(■, ■) = 0.07

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) = area under Precision vs Recall Curve
  - 1. For each detection (highest score to lowest score)
    - 1. If it matches some GT box with IoU > 0.5, 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) = area under PR curve
- 3. Mean Average Precision (mAP) = average of AP for each class
- 4. For “COCO mAP”: Compute mAP@thresh for each IoU threshold (0.5, 0.55, 0.6, …, 0.95) and take average

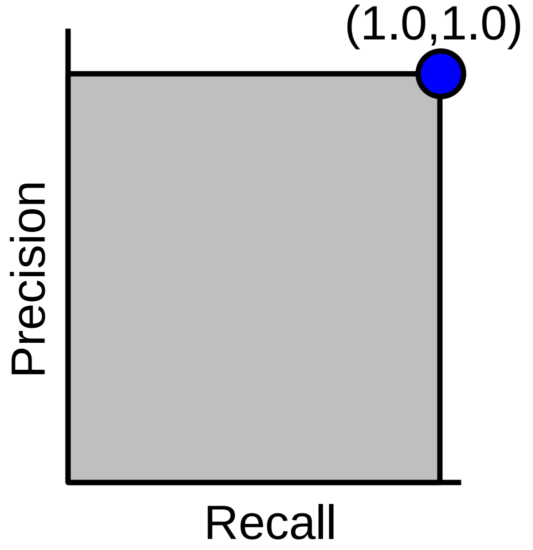


All dog detections sorted by score
0.99
0.95
0.90
0.5
0.10
Match: IoU > 0.5
All ground-truth dog boxes
Precision = 1/1 = 1.0
Recall = 1/3 = 0.33
FN=2
TN=0
TP=1
FP=0
Precision
Recall
1.0
All dog detections sorted by score
Match: IoU > 0.5
All ground-truth dog boxes
Precision = 2/2 = 1.0
Recall = 2/3 = 0.67
FN=1
TN=0
TP=2
FP=0
All dog detections sorted by score
No match > 0.5 IoU with GT
All ground-truth dog boxes
Precision = 2/3 = 0.67
Recall = 2/3 = 0.67
FN=1
TN=0
TP=2
FP=1
All dog detections sorted by score
No match > 0.5 IoU with GT
All ground-truth dog boxes
Precision = 2/4 = 0.5
Recall = 2/3 = 0.67
FN=1
TN=0
TP=2
FP=2
All dog detections sorted by score
Match: > 0.5 IoU
All ground-truth dog boxes
Precision = 3/5 = 0.6
Recall = 3/3 = 1.0
FN=0
TN=0
TP=3
FP=2
Dog AP = 0.86
Precision = $\frac{TP}{TP+FP}$
Recall = $\frac{TP}{TP+FN}$

Perfect Detection

- To get the perfect AP = 1.0, we need Precision = $\frac{TP}{TP+FP} = 1.0$, Recall = $\frac{TP}{TP+FN} = 1.0$, i.e., Hit all N GT boxes with IoU > 0.5, and have no FP detections ranked above any TP

FN=0	TN=0
TP=N	FP=0



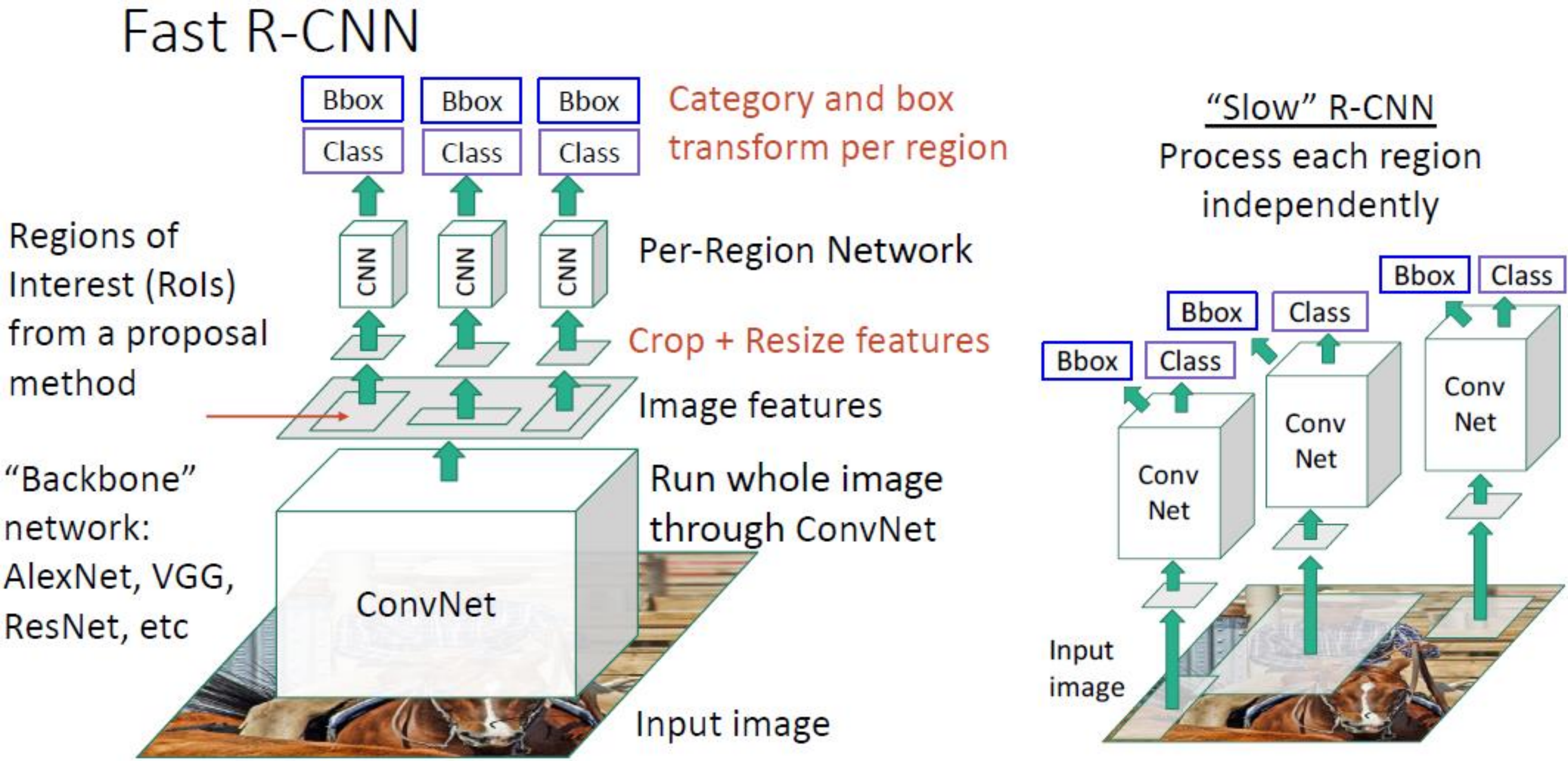
mAP

- Suppose we have 3 classes, and have computed AP values for each classwith IoU threshold .5 as:
  - Car AP = .65, Cat AP = .80, Dog AP = .86, then mAP@.5=.77
- For COCO mAP: Compute mAP@threshold for each IoU threshold (0.5, 0.55, 0.6, …, 0.95) and take average
  - COCO mAP = .4

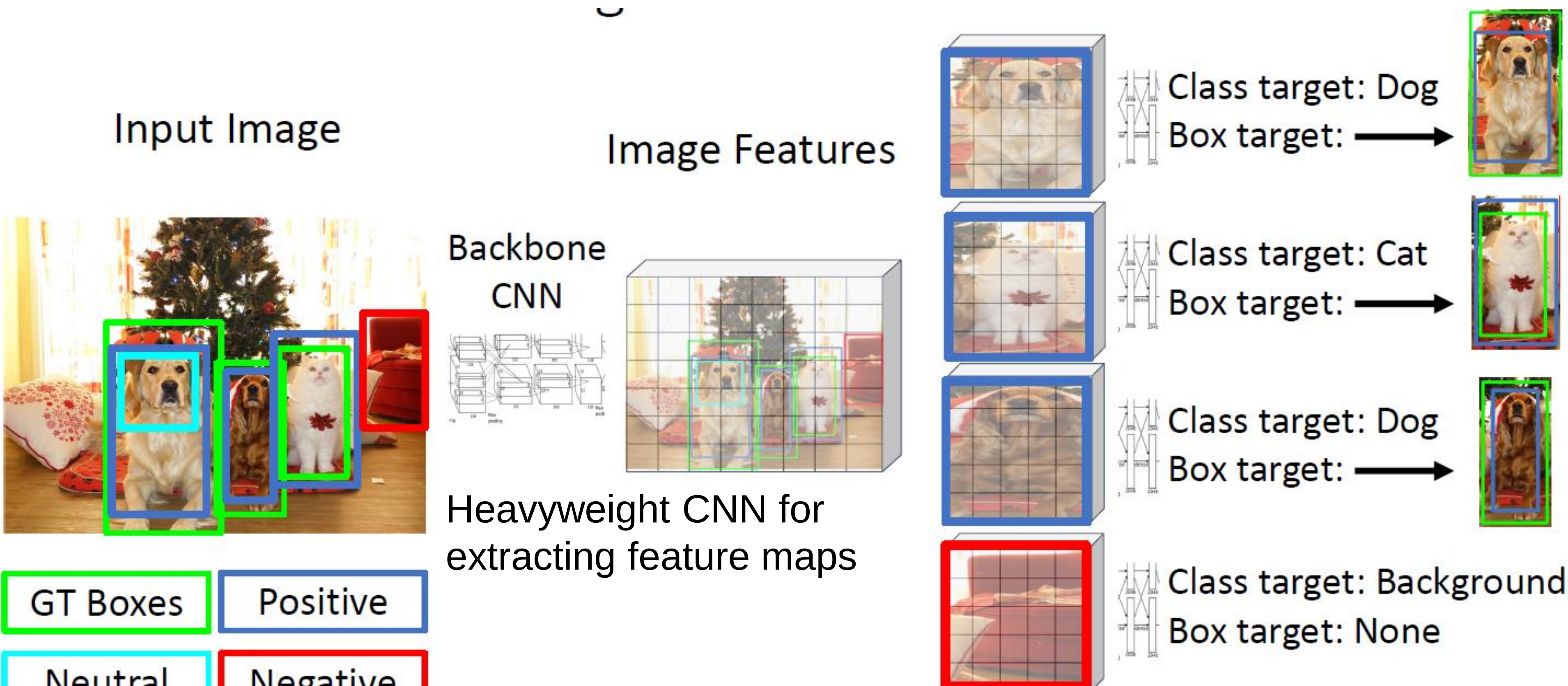
Important

Fast R-CNN

- 1. Use a backbone network to extract feature maps from the whole image; 2. Generate region proposals based on the feature maps; 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



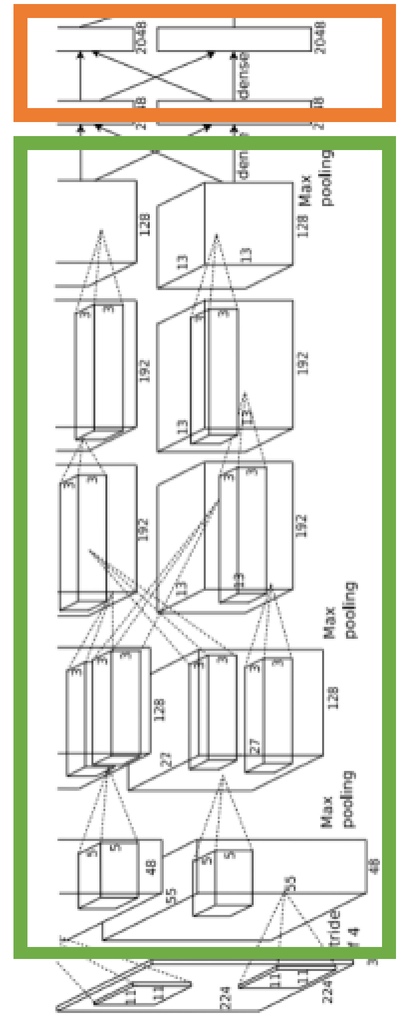
Fast R-CNN Training Example



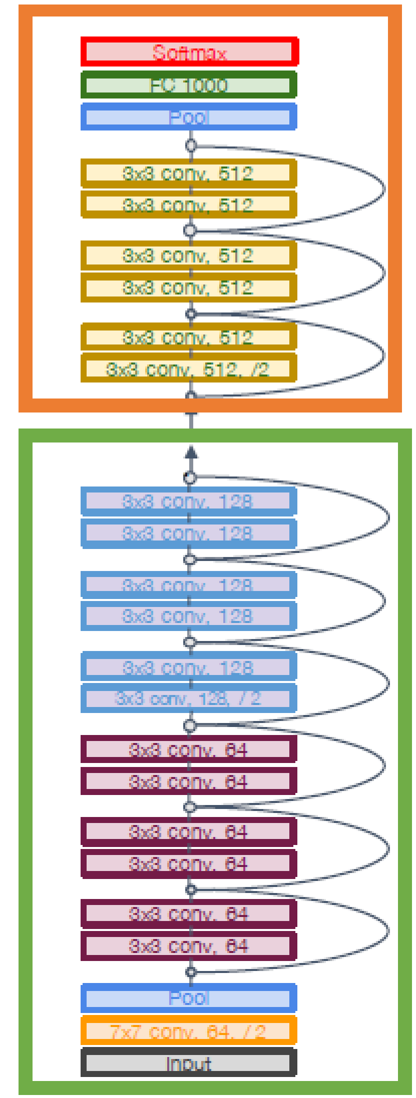
Lightweight NN for Bbox
regression & classification

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



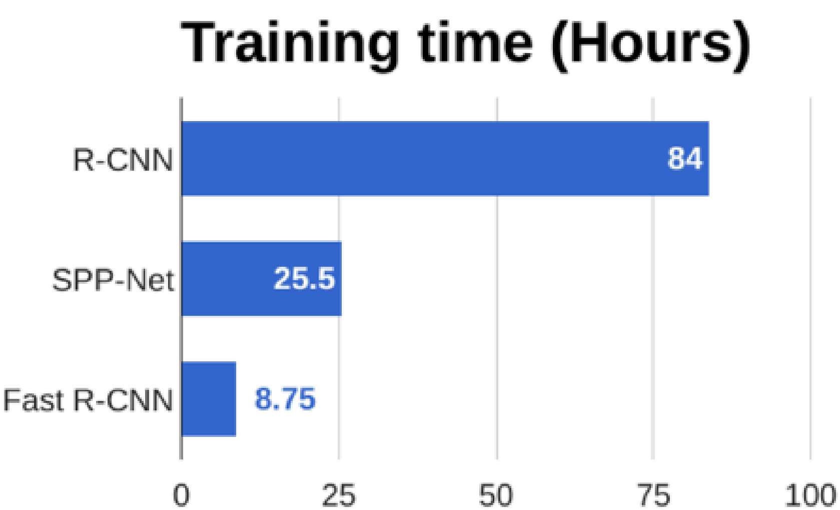
AlexNet

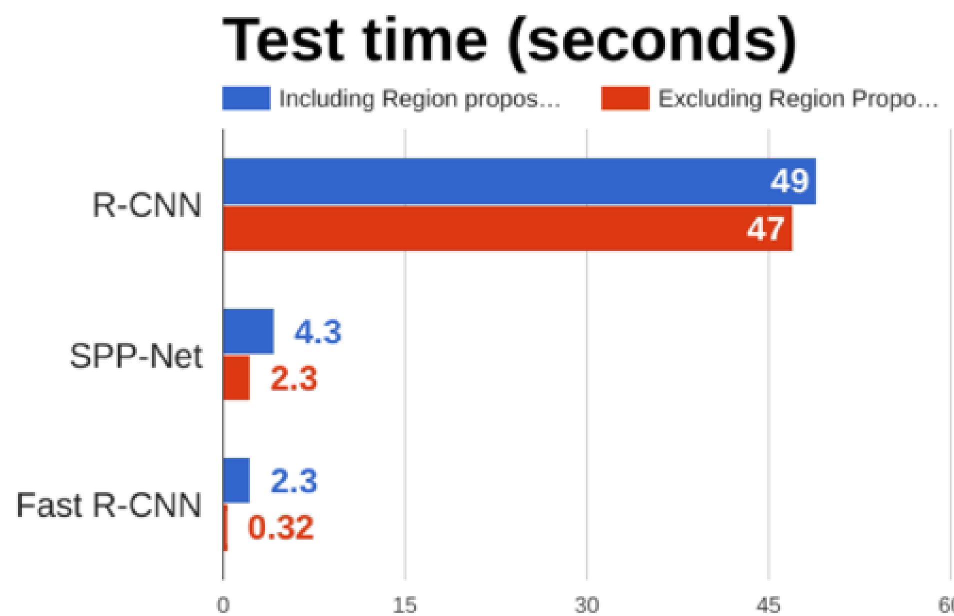


ResNet

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





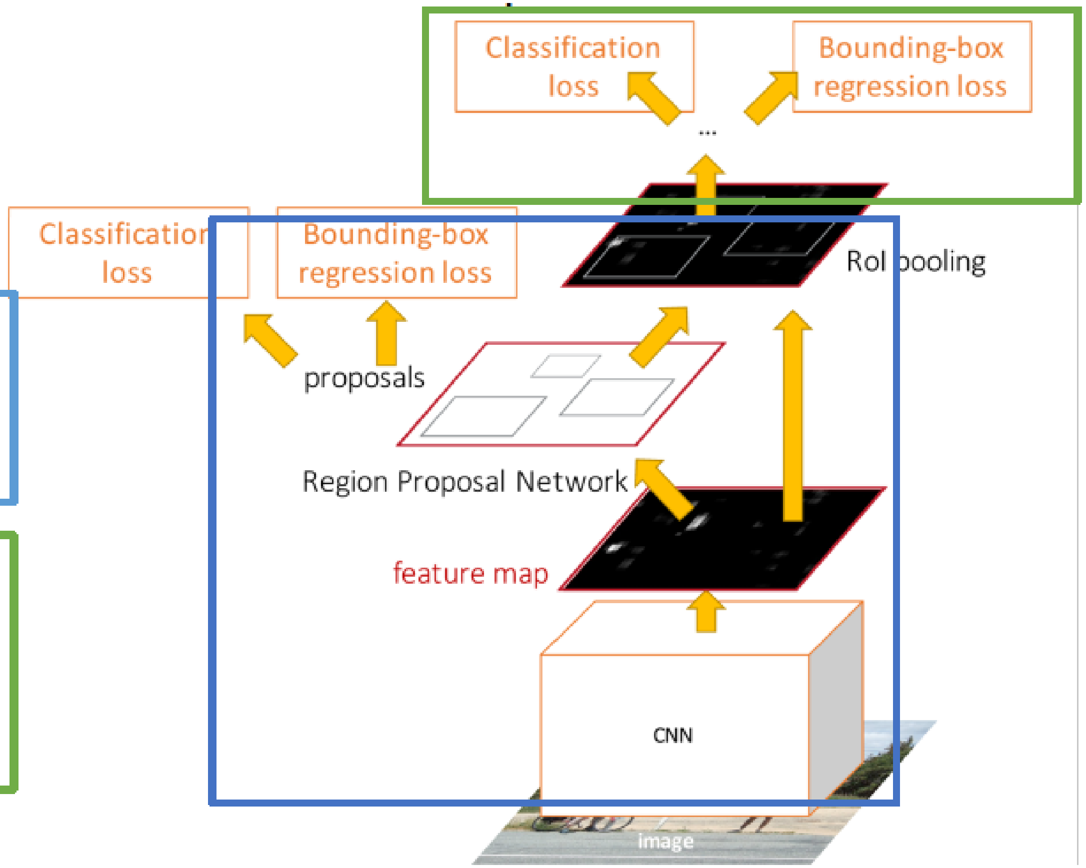
Faster R-CNN

- Use Region Proposal Network (RPN) to predict proposals from feature maps generated by the backbone network
- The rest the same as Fast R-CNN

Faster R-CNN is a
Two-stage object detector

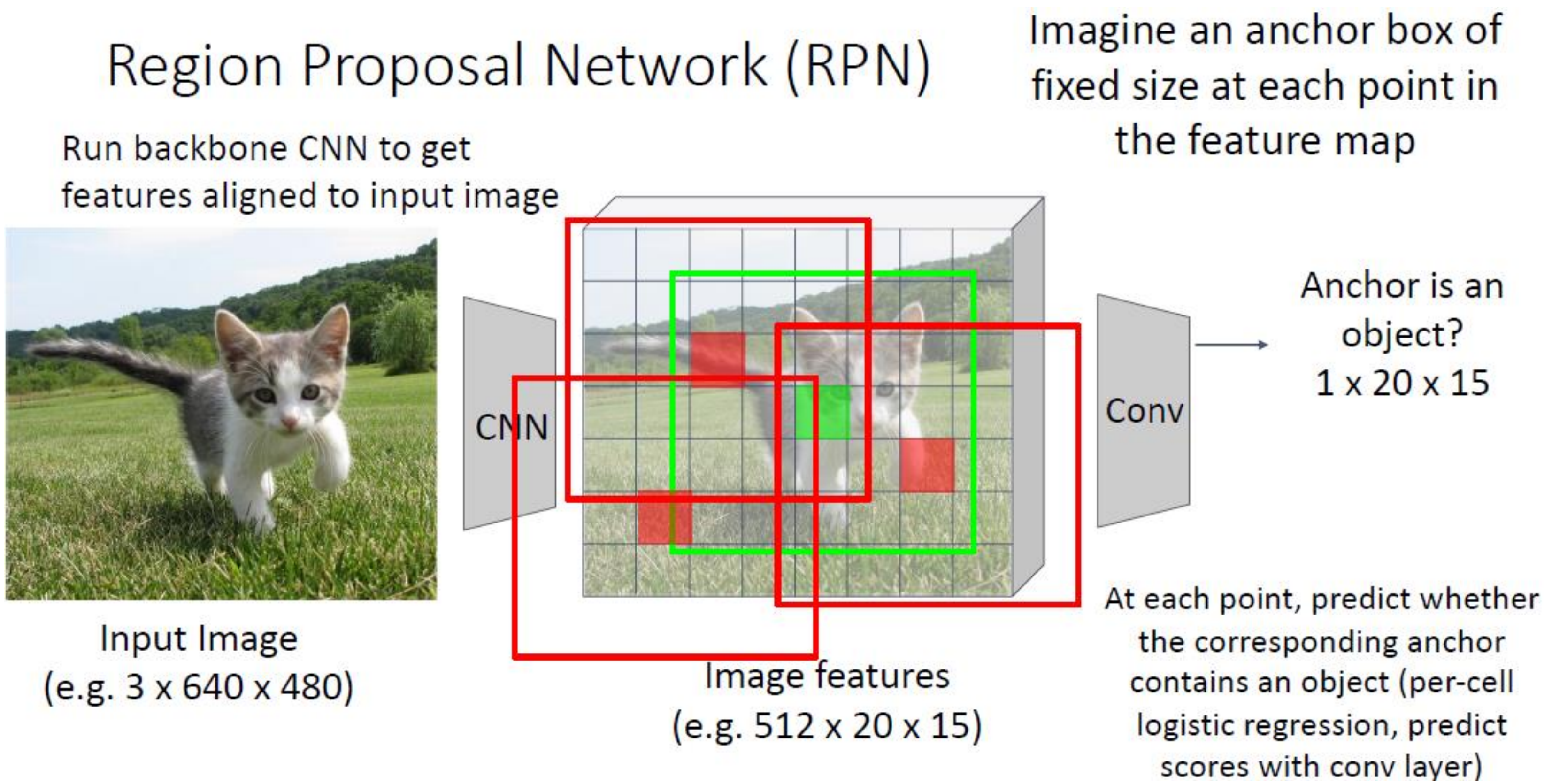
First stage: Run once per image
- Backbone network
- Region proposal network

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



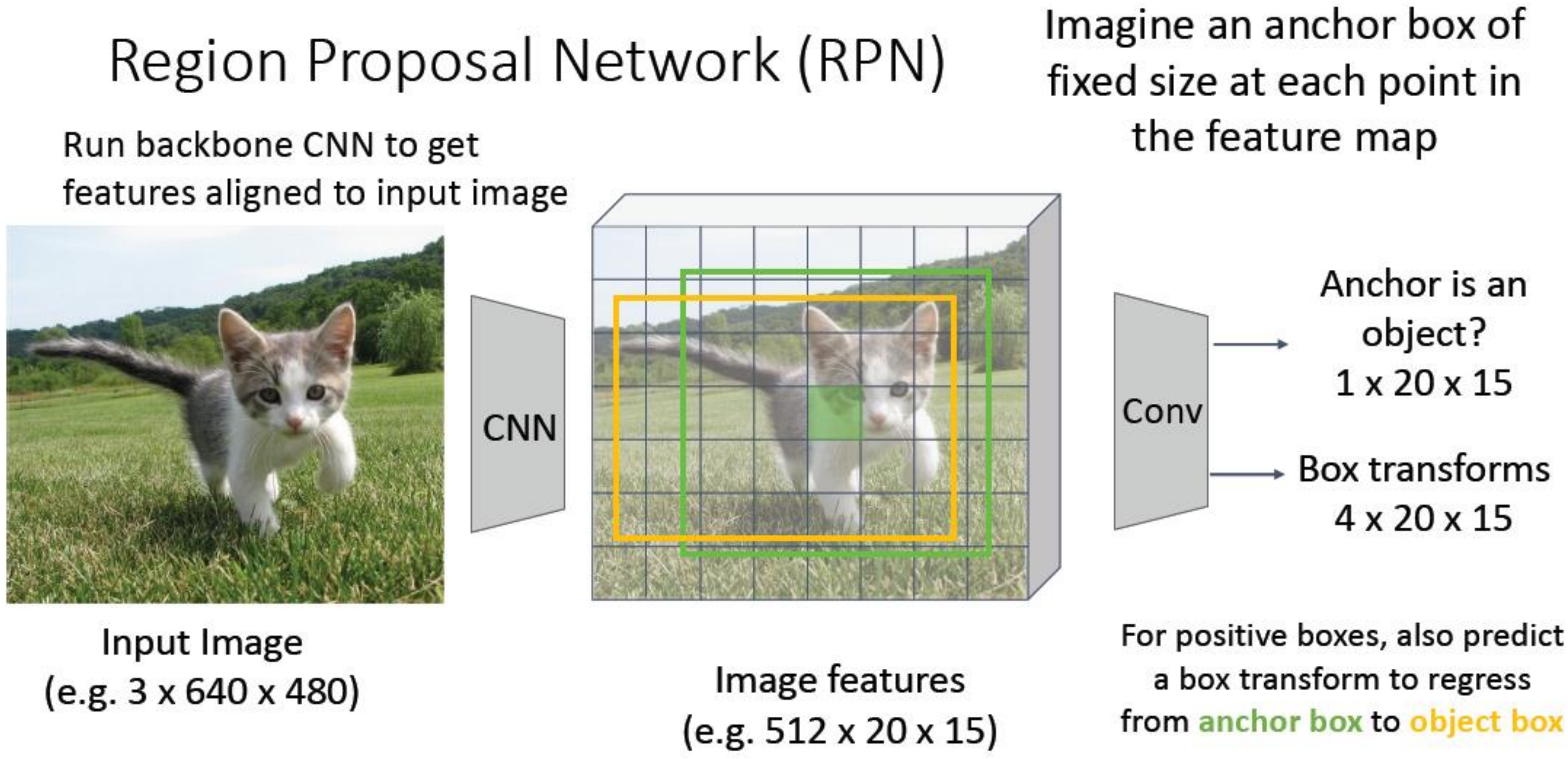
Region Proposal Network

- Perform binary prediction for each anchor box
- The red anchor boxes are predicted false (contains no object) and discarded; the green anchor box is predicted true (contains an object) and survives



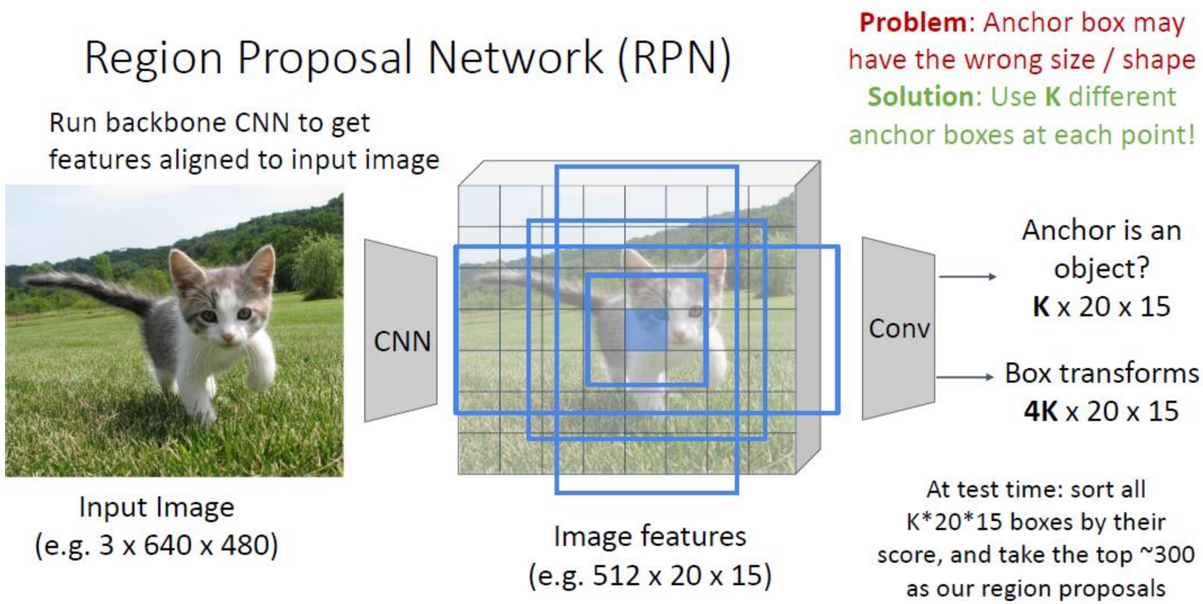
Region Proposal Network

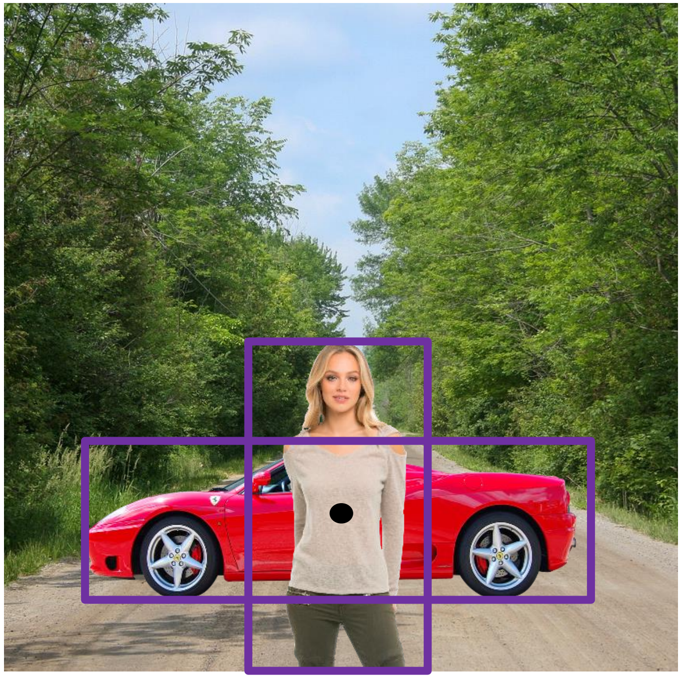
- Bbox regression: transform each positive anchor box (green), into a better-fitting (higher IoU with the Ground-Truth Bbox) object box (yellow)



Region Proposal Network

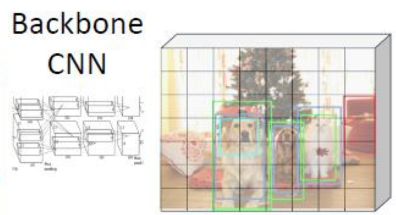
- Place K anchor boxes centered at each position in the feature map, each with different sizes and aspect ratios ($K = 4$ in left fig)
  - This allows better-fitting anchor boxes (left fig), which helps ease the downstream Bbox regression task, and detection of multiple objects centered at the same position (right fig)



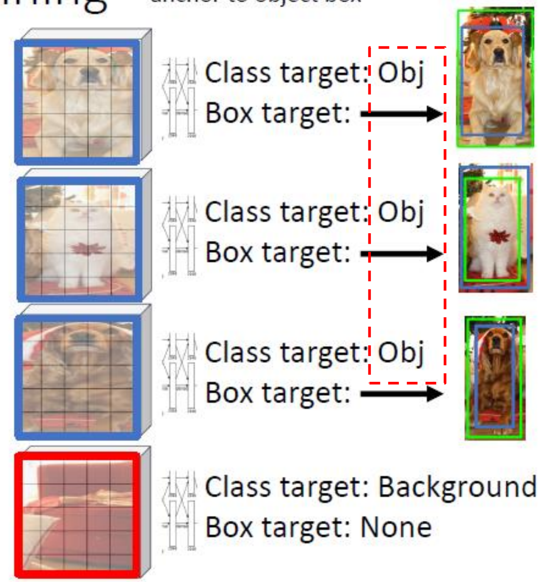


Faster R-CNN Training: RPN Training

RPN predicts Object / Background for each anchor, as well as regresses from anchor to object box



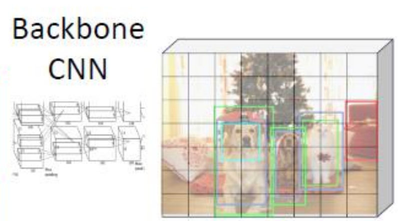
RPN gives lots of **anchors** which we classify as pos / neg / neutral by matching with ground-truth



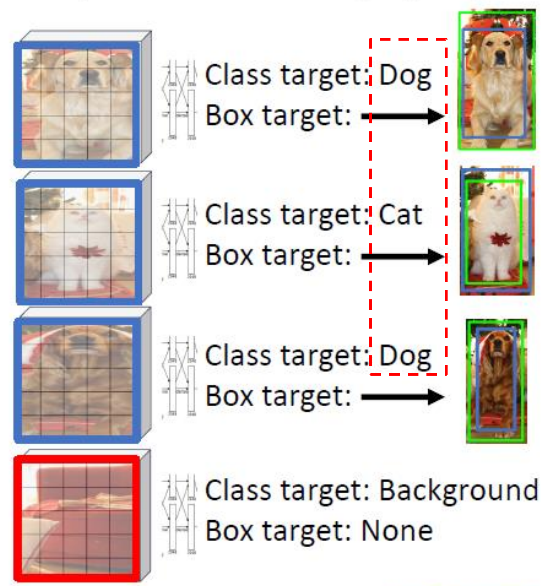
This image is CC0 public domain

Faster R-CNN Training: Stage 2

Crop features for each proposal, use them to predict class and box targets per region



Now proposals come from RPN rather than selective search, but otherwise this works the same as Fast R-CNN training



This image is CC0 public domain

Important

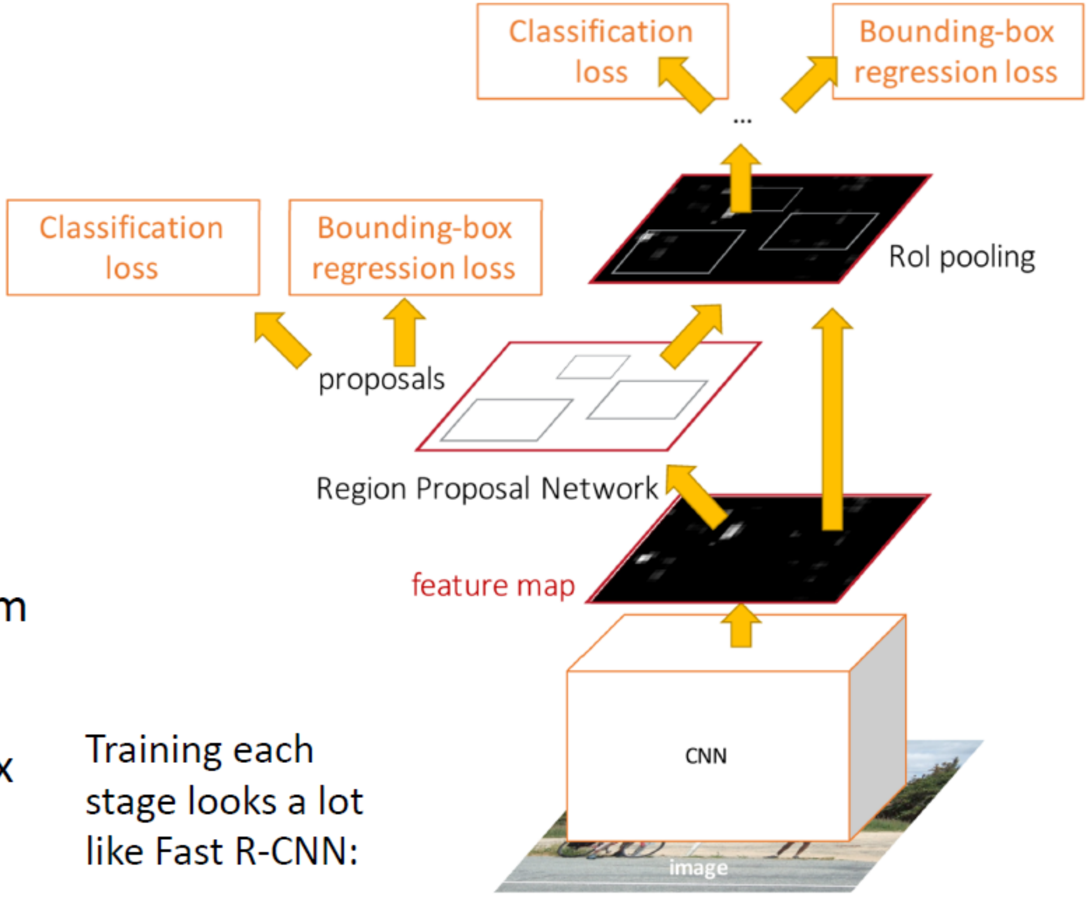
Faster R-CNN: Loss Function

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

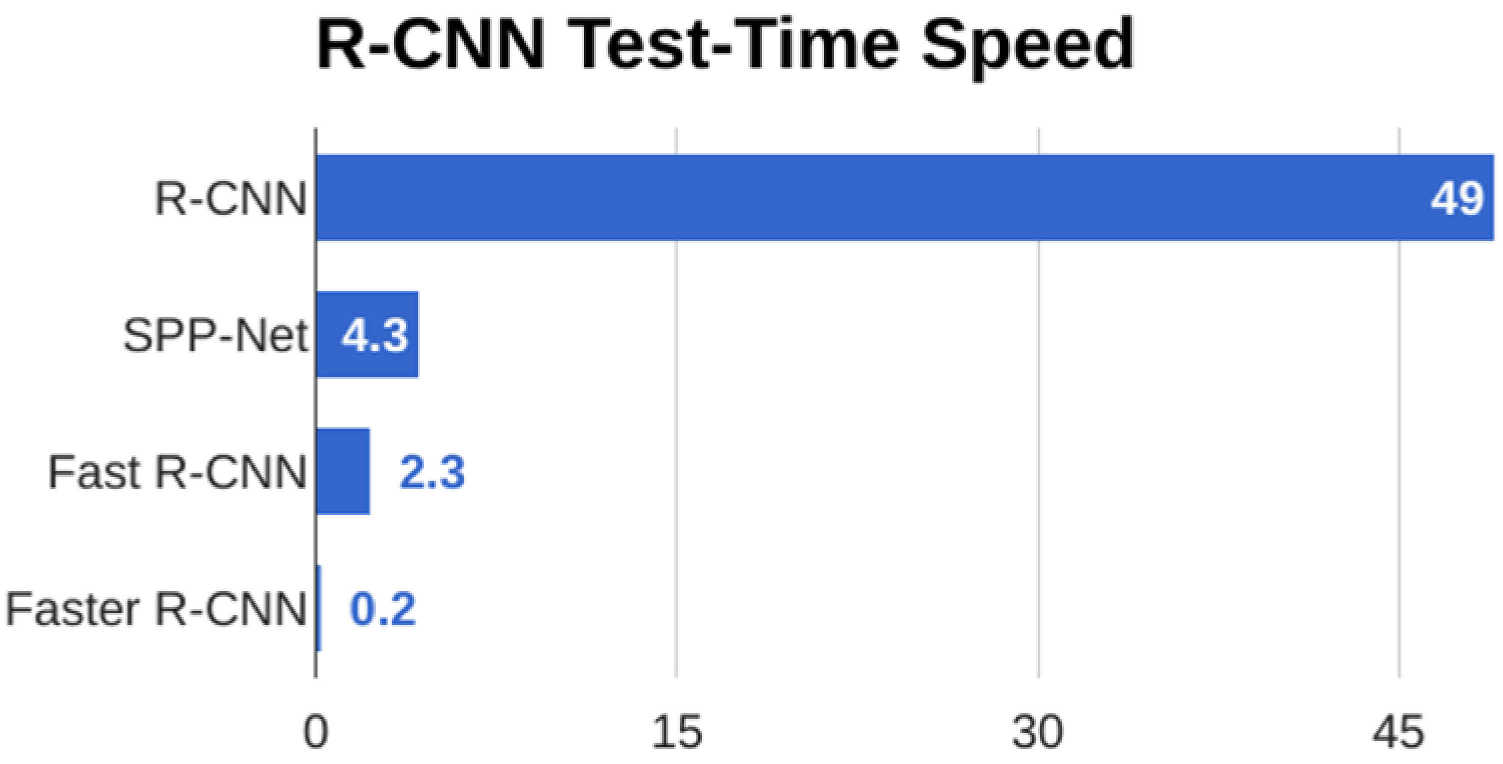
Anchor -> Region Proposal -> Object Box

(Stage 1) (Stage 2)



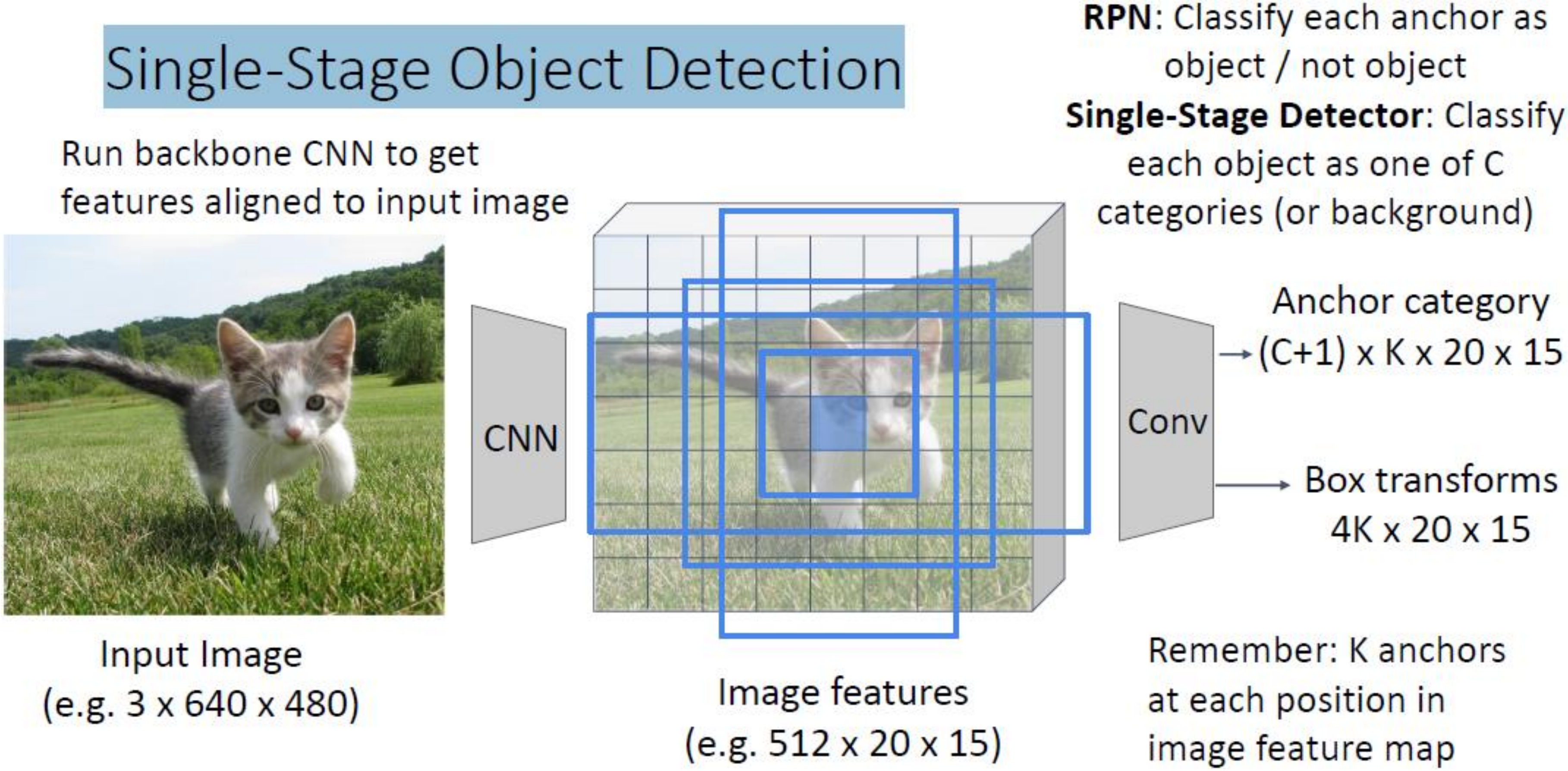
Training each stage looks a lot like Fast R-CNN:

Faster R-CNN: Performance



Single-Stage Object Detection

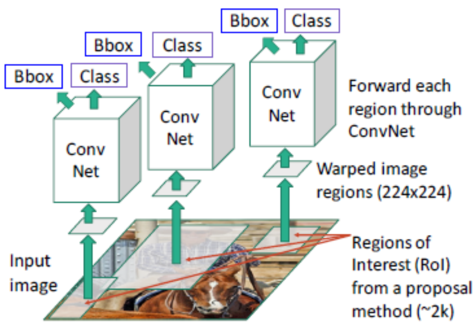
- Instead of the binary (object/not object) classifier for each of the Kx20x15 anchors in Faster-RCNN, we classify each anchor into C+1 classes (including the background), in addition to regression of 4Kx20x15 box transforms from anchor box to object box
- Sometimes use class-specific regression: Predict different box transforms for each class, with Cx4Kx20x15 box transforms



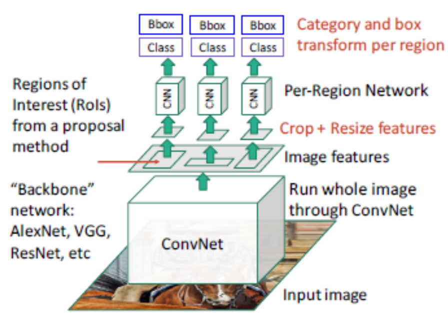
Important

Summary of Object Detectors

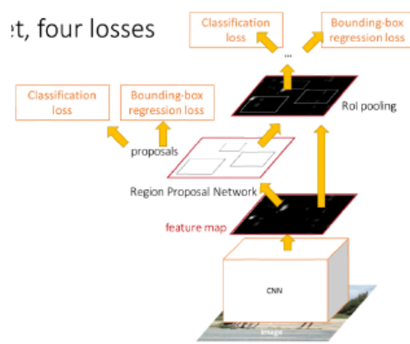
“Slow” R-CNN: Run CNN independently for each region



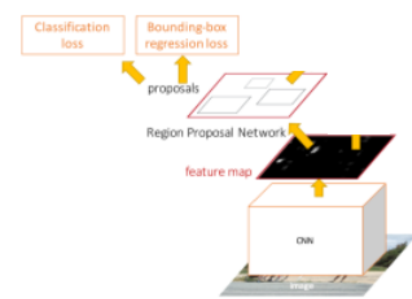
Fast R-CNN: Apply differentiable cropping to shared image features



Faster R-CNN: Compute proposals with CNN

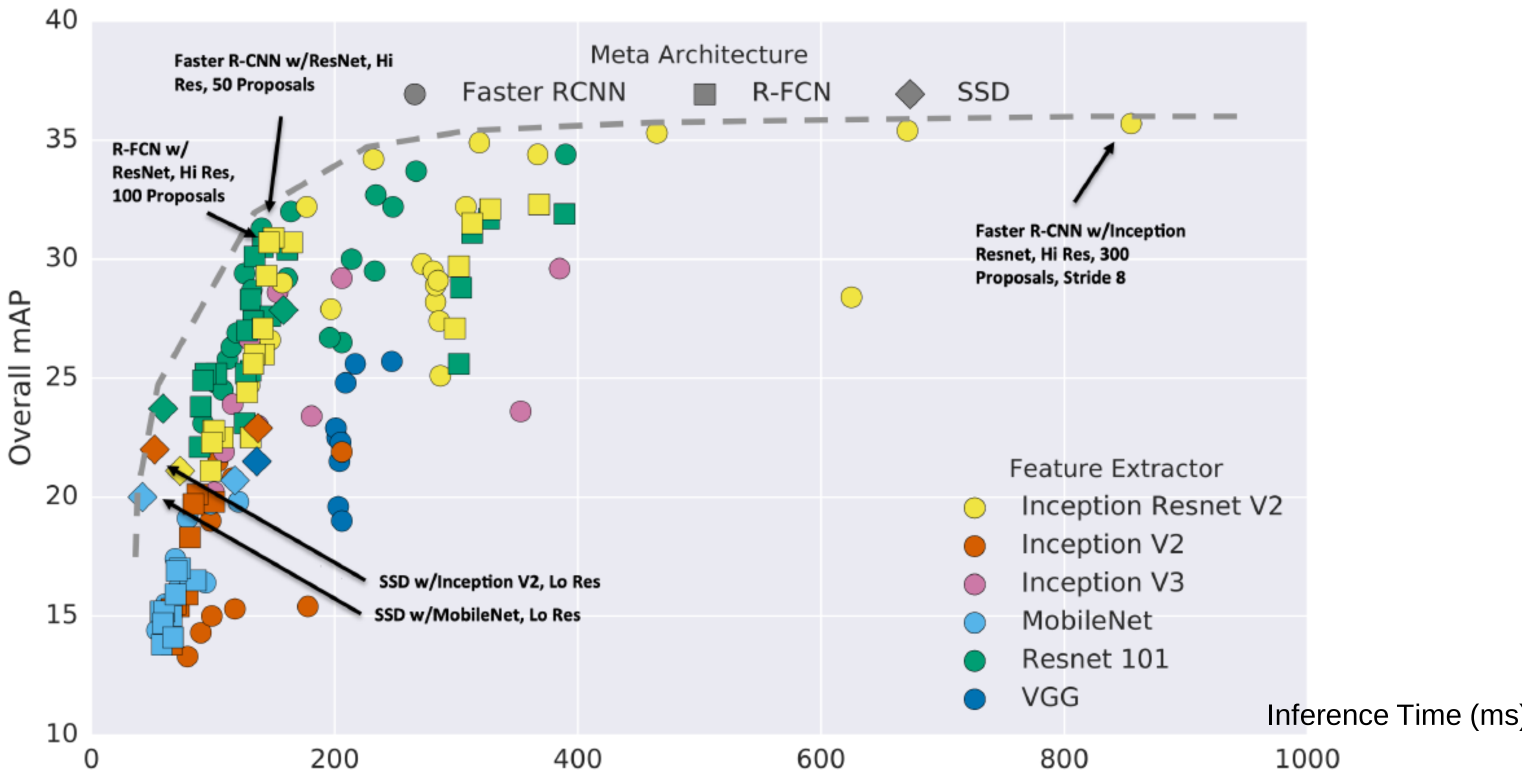


Single-Stage: Fully convolutional detector



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
- Diminishing returns for slower methods



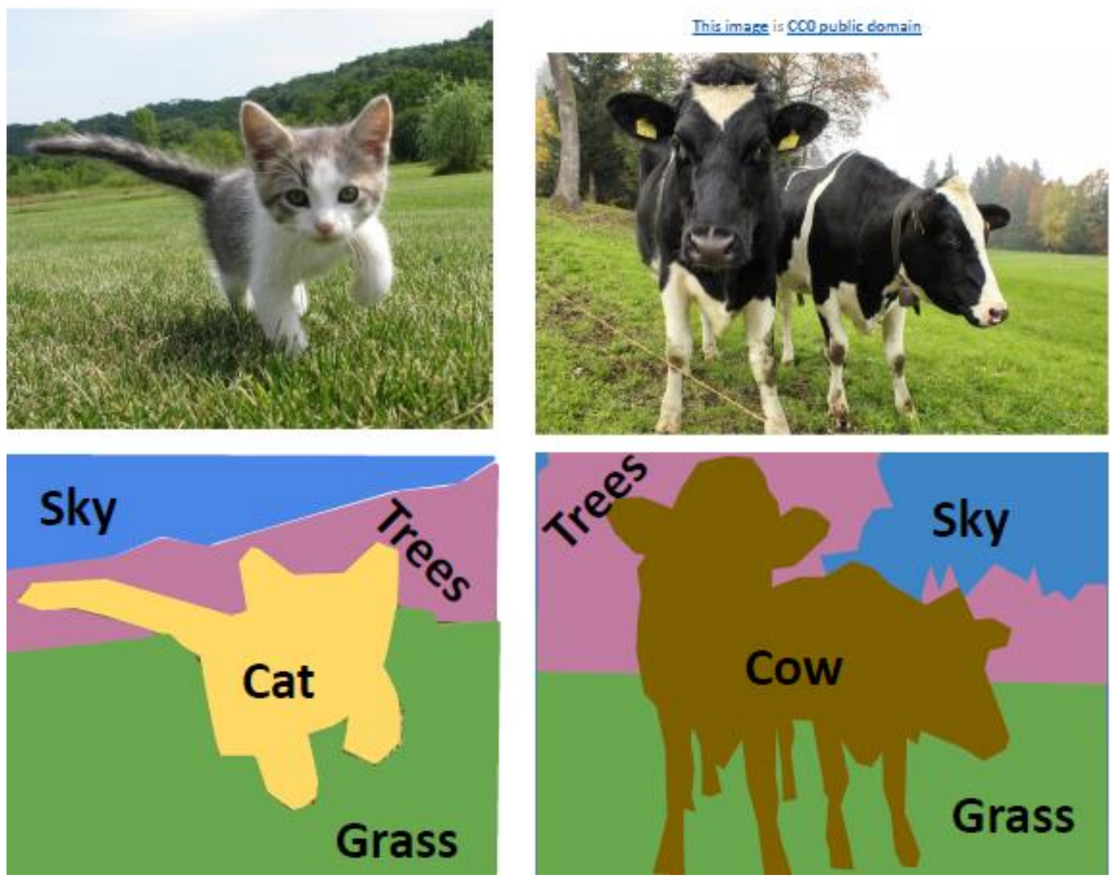
Outline

- Object detection
- Segmentation

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

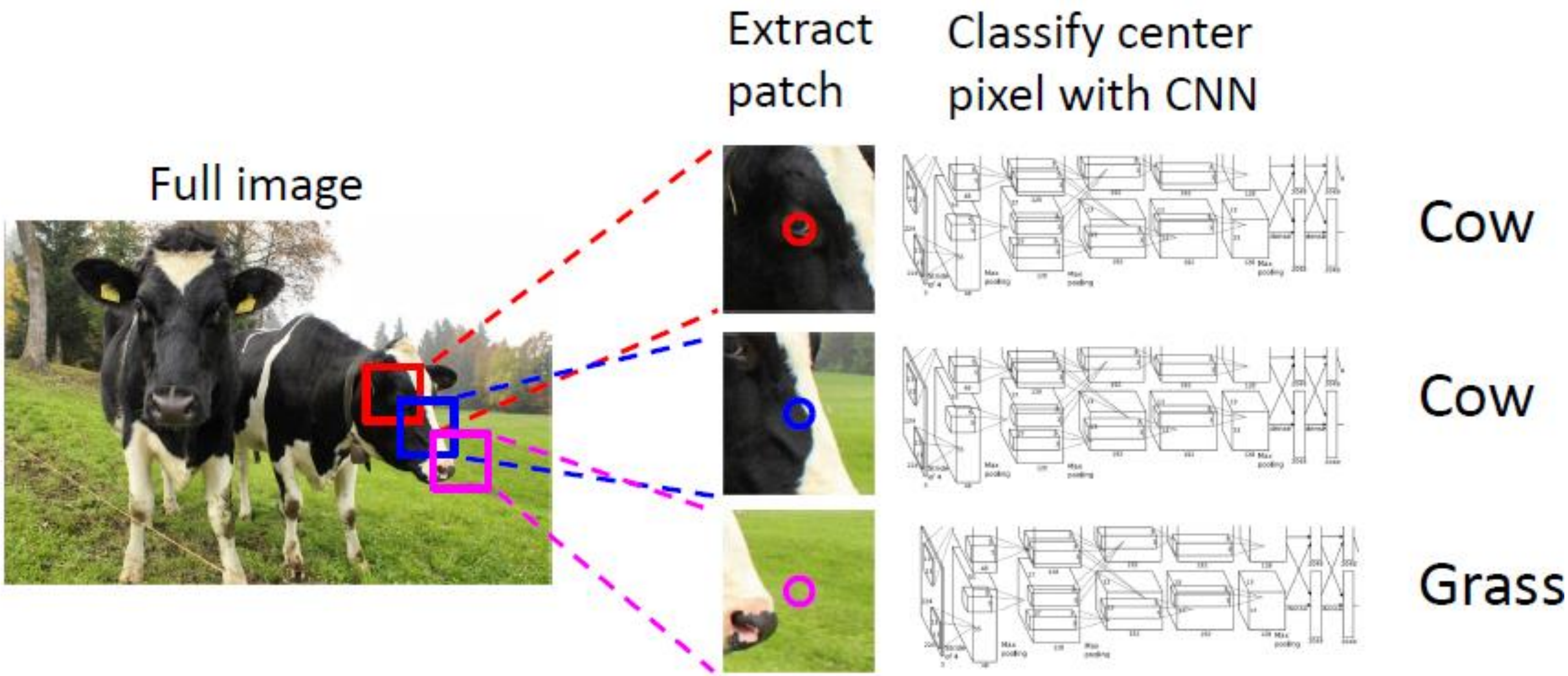
Semantic Segmentation: Task Definition

- 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)



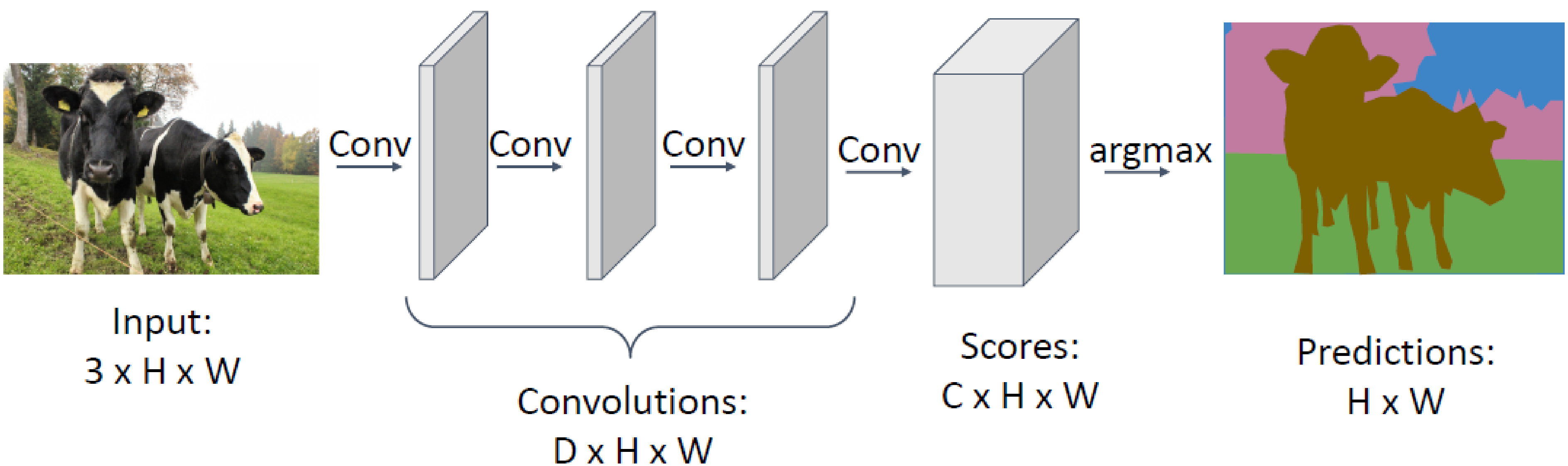
An Early Approach: Sliding Windows

- Slide a box across the image, and apply a CNN to classify each crop's center pixel
- Computationally inefficient



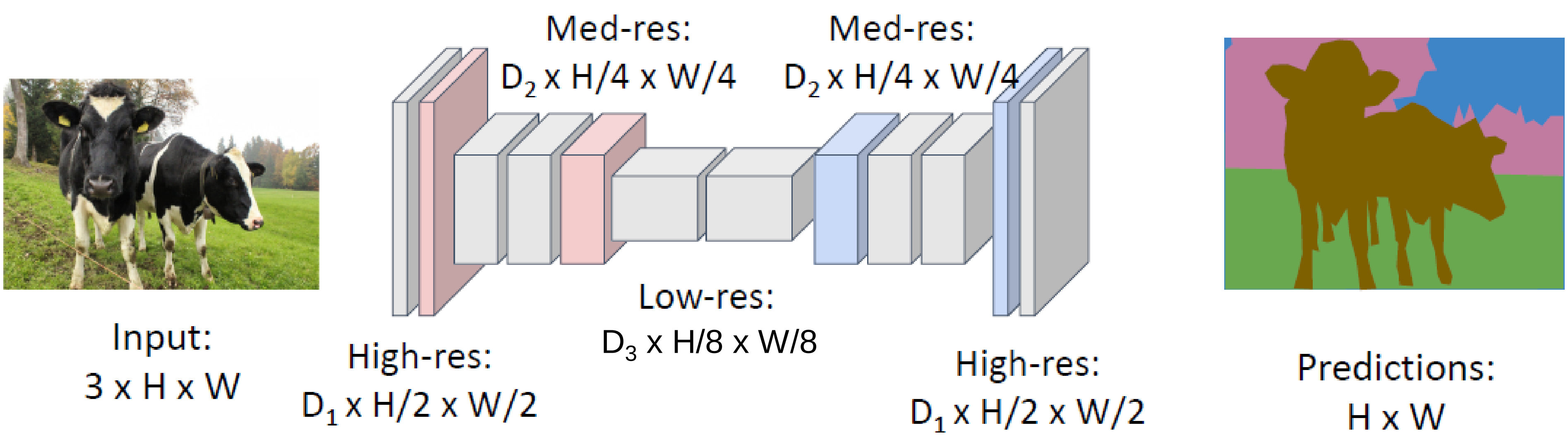
Fully Convolutional Network

- A CNN with only CONV layers, no Fully-Connected layer(s), 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 is 1+2L
  - Problem #2: Convolution on high-res images without downsampling is expensive



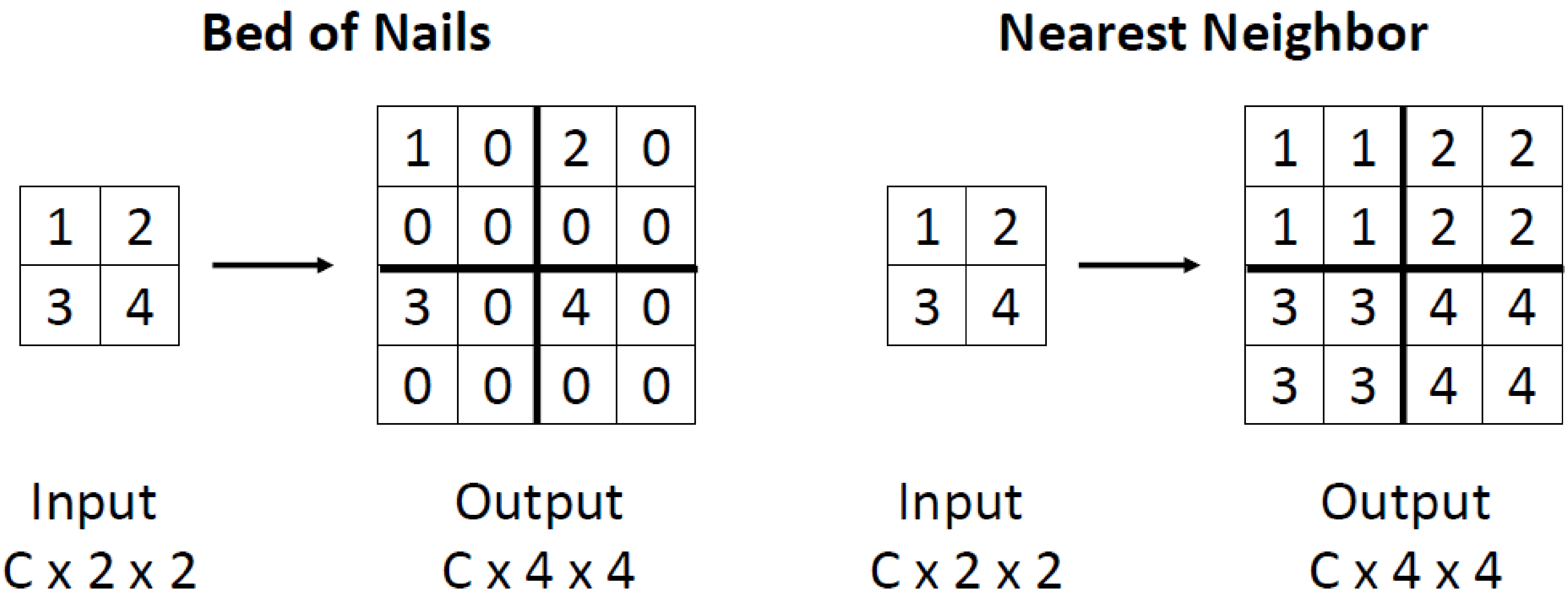
Fully Convolutional Network

- 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 output with the same size as input



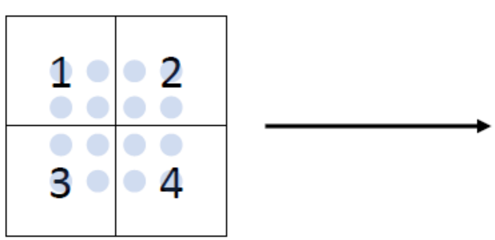
Unpooling for Upsampling

- Fig shows 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

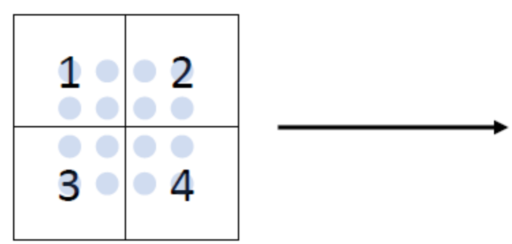
- Fig shows upsampling from a 2x2 image to a 4x4 image with bilinear (left) and bicubic (right) interpolation, to generate smoother outputs
- Each output element is computed as a linear or cubic combination of its closest neighbors; closer neighbors are given higher weights



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

Max Unpooling: Place into remembered positions

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

→

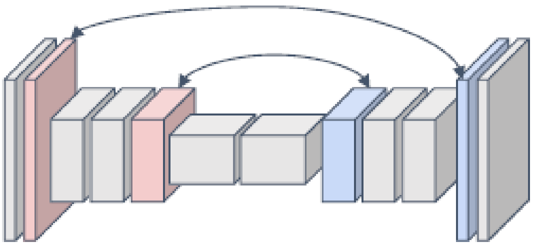
5	6
7	8

→ Rest of net →

1	2
3	4

→

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

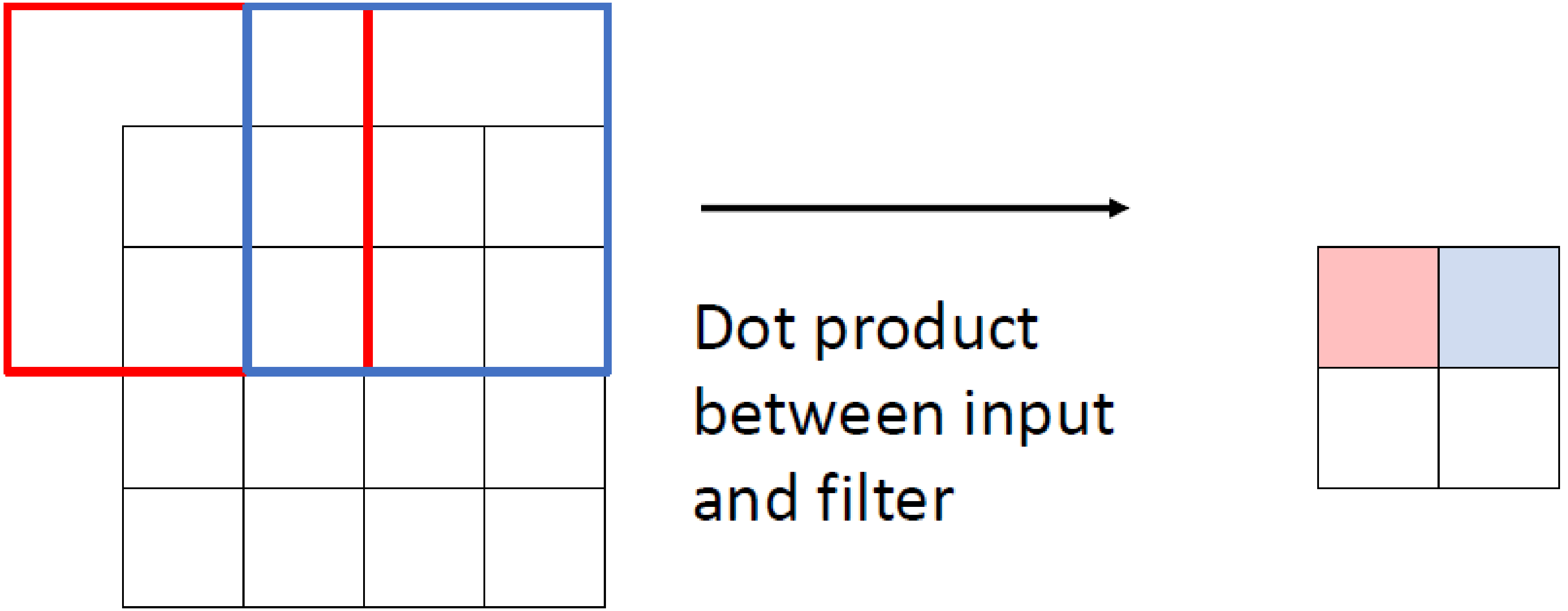


Pair each downsampling layer with an upsampling layer

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

Recall: Regular Convolution

Recall: Normal 3 x 3 convolution, <u>stride 2</u>, pad 1



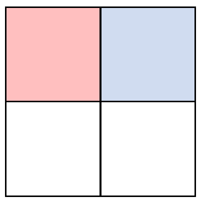
Input: 4 x 4

Output: 2 x 2

Learnable Upsampling: Transposed Convolution

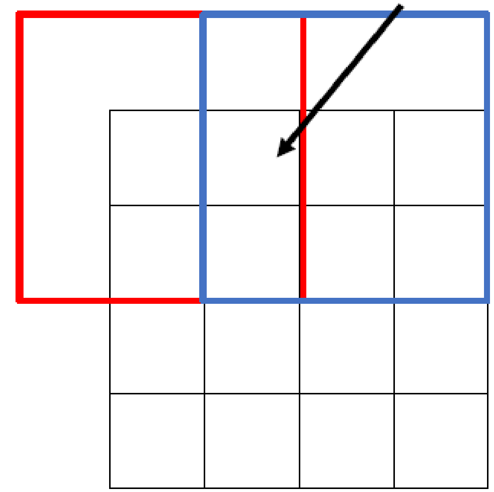
3 x 3 **convolution transpose**, stride 2

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

Sum where output overlaps

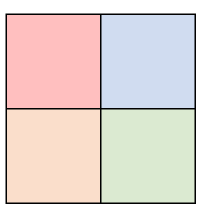


Input: 2 x 2

Output: 4 x 4

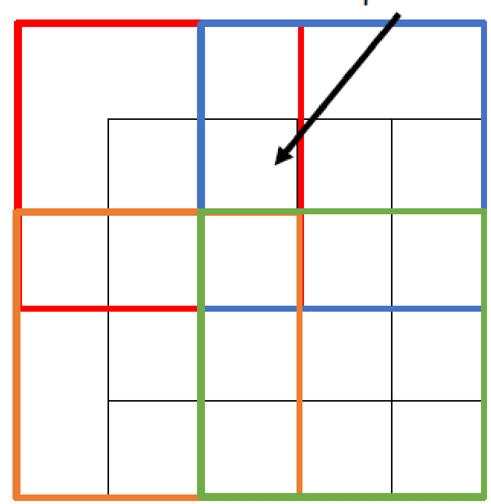
3 x 3 **convolution transpose**, stride 2

This gives 5x5 output – need to trim one pixel from top and left to give 4x4 output



Weight filter by input value and copy to output

Sum where output overlaps

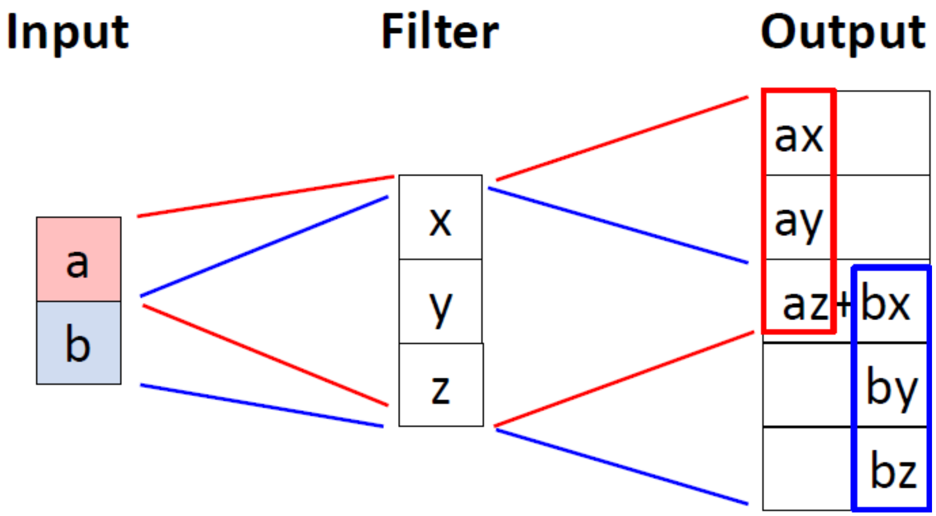


Input: 2 x 2

Output: 4 x 4

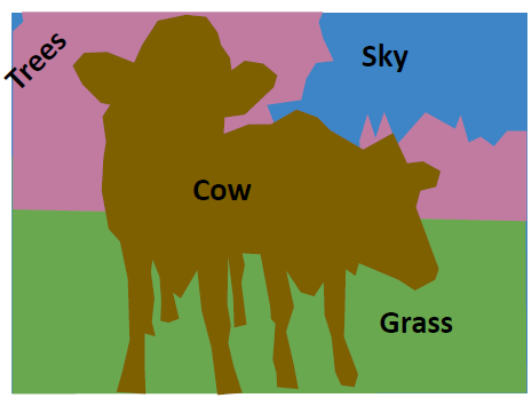
Transposed Convolution Example

- Fig shows a 1D toy example:
  - Output has copies of filter weighted by input
  - Stride 2: Move 2 pixels in output for each pixel in input
  - Sum at overlaps
- The filter moves at a slower pace than with unit stride
- It has many names: Transposed Convolution, Deconvolution, Upconvolution, Fractionally-strided convolution

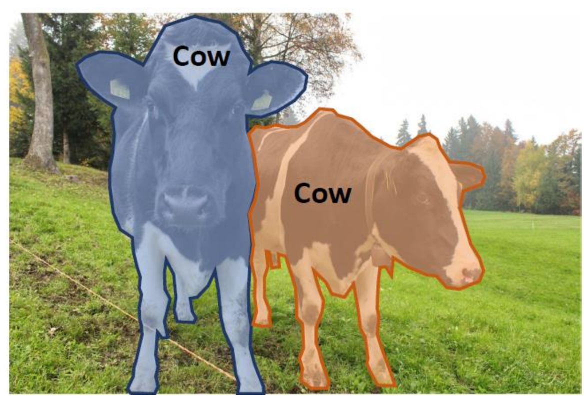


Types of Segmentation Tasks

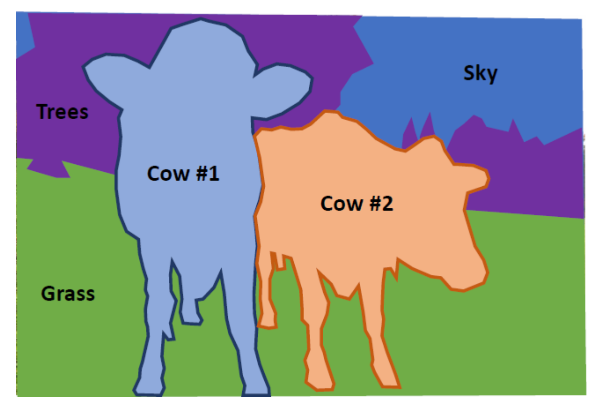
- Things: Object categories that can be separated into object instances (e.g. cats, cars, person)
- Stuff: Object categories that cannot be separated into instances (e.g. sky, grass, water, trees)
- Object Detection: Detects individual object instances, but only gives bounding box (things only)
- Semantic 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

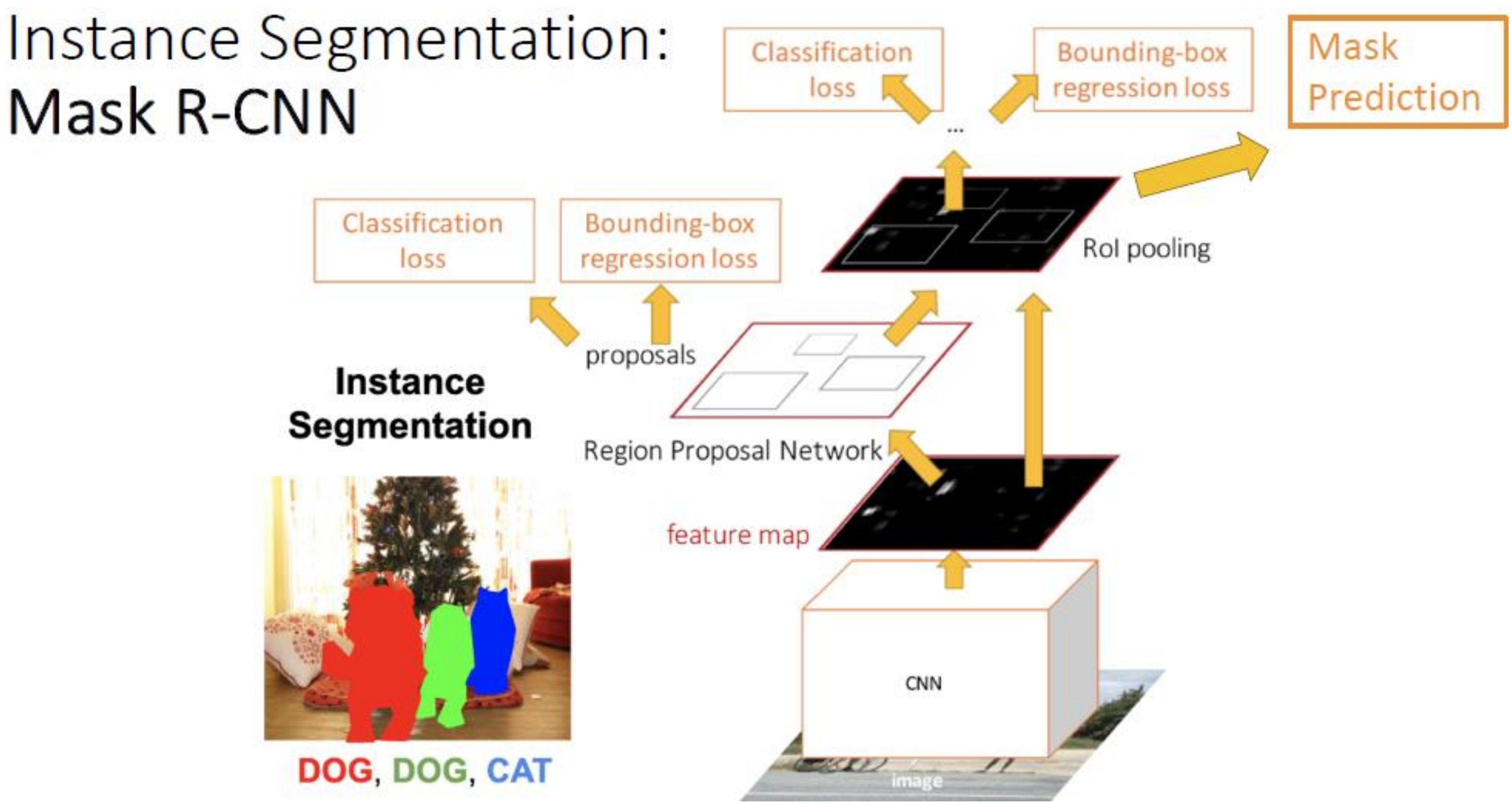


Panoptic Segmentation

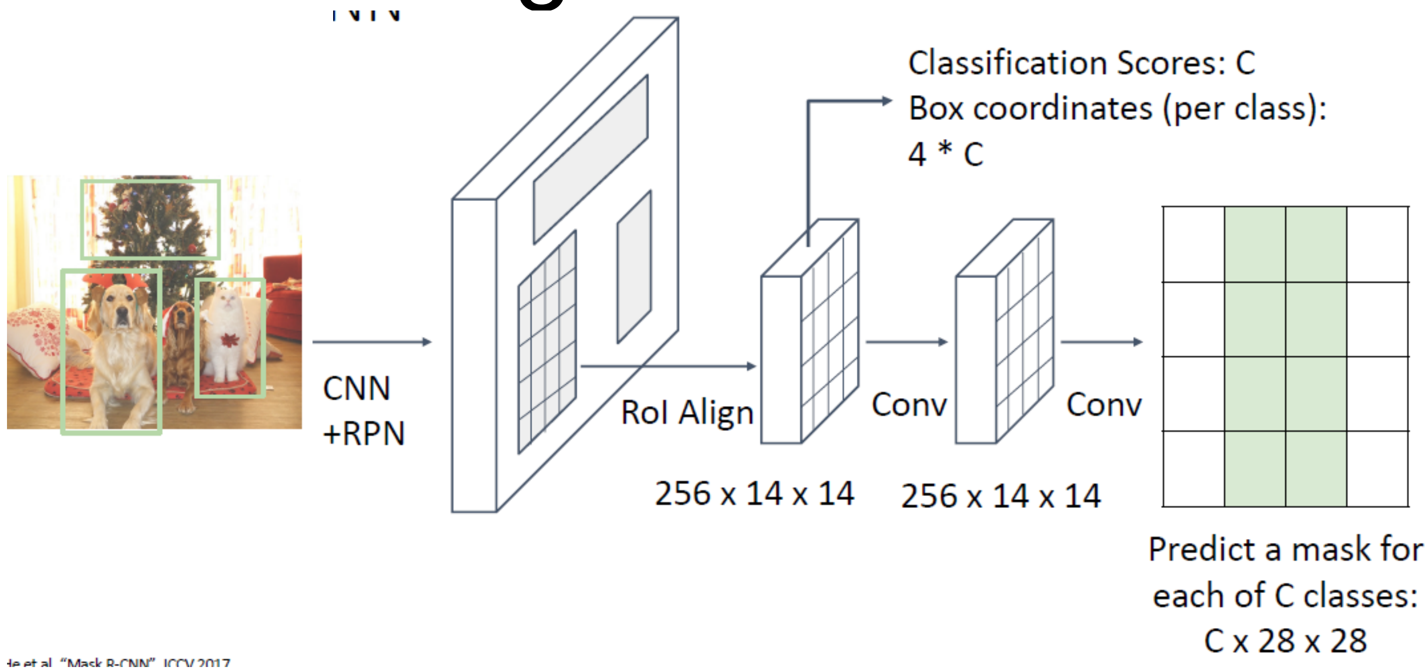
Important

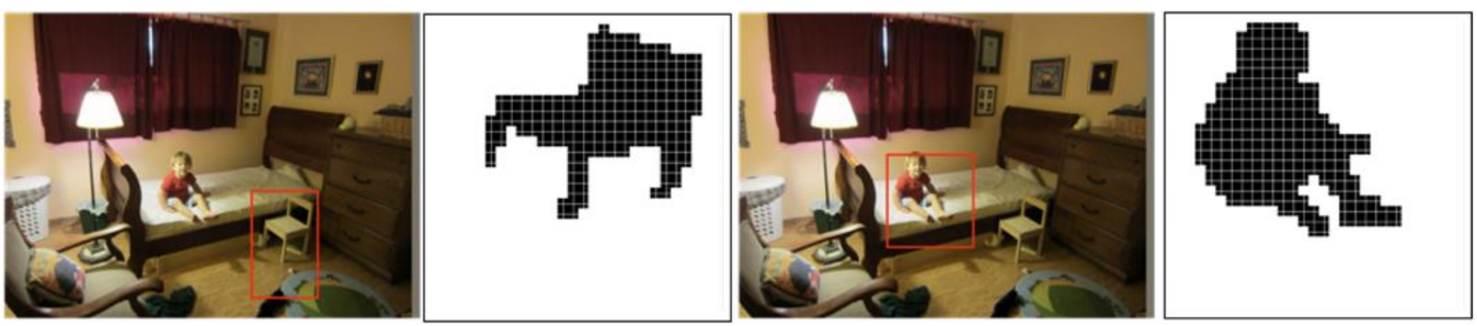
Mask R-CNN for Instance Segmentation

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



Mask R-CNN for Instance Segmentation



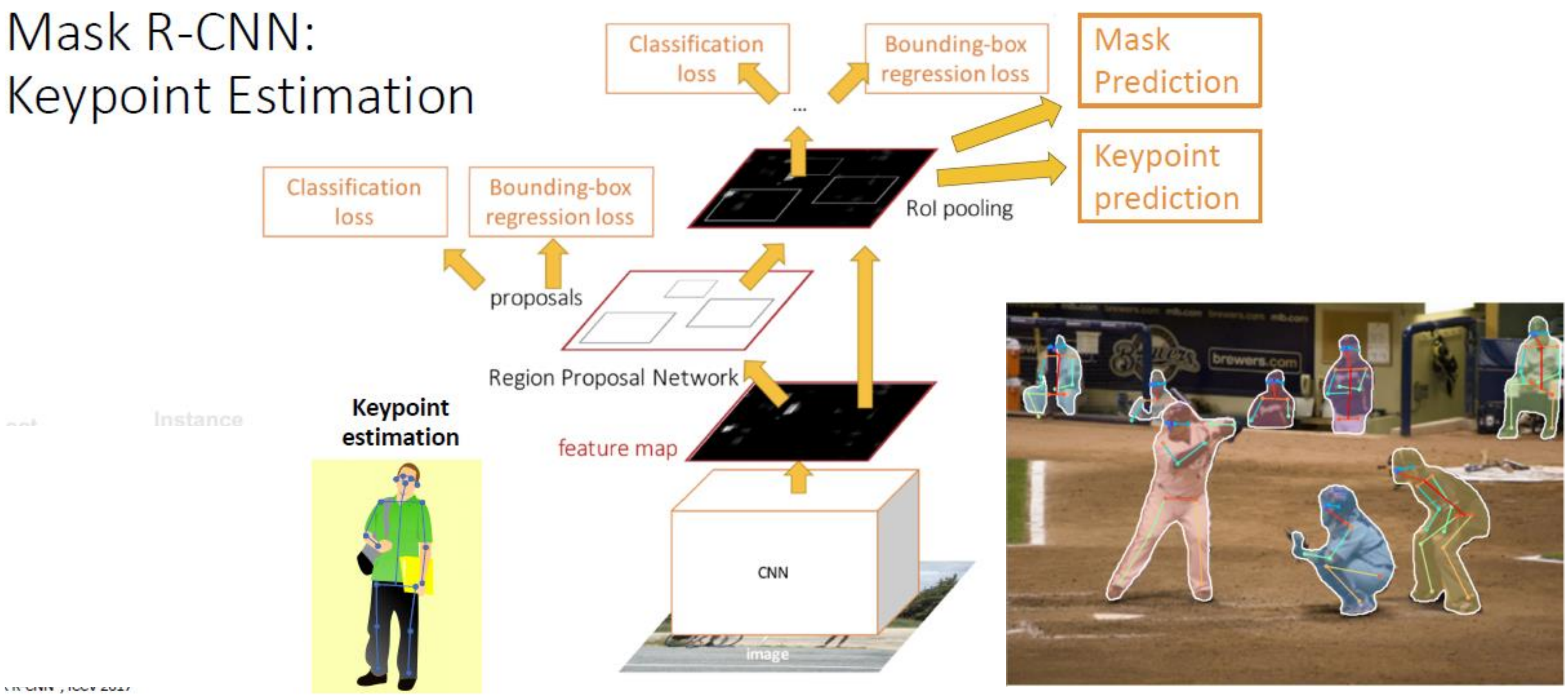


Target segmentation mask for class “chair” in the Bbox

Target segmentation mask for class “person” in the Bbox

Mask R-CNN for Keypoint Estimation

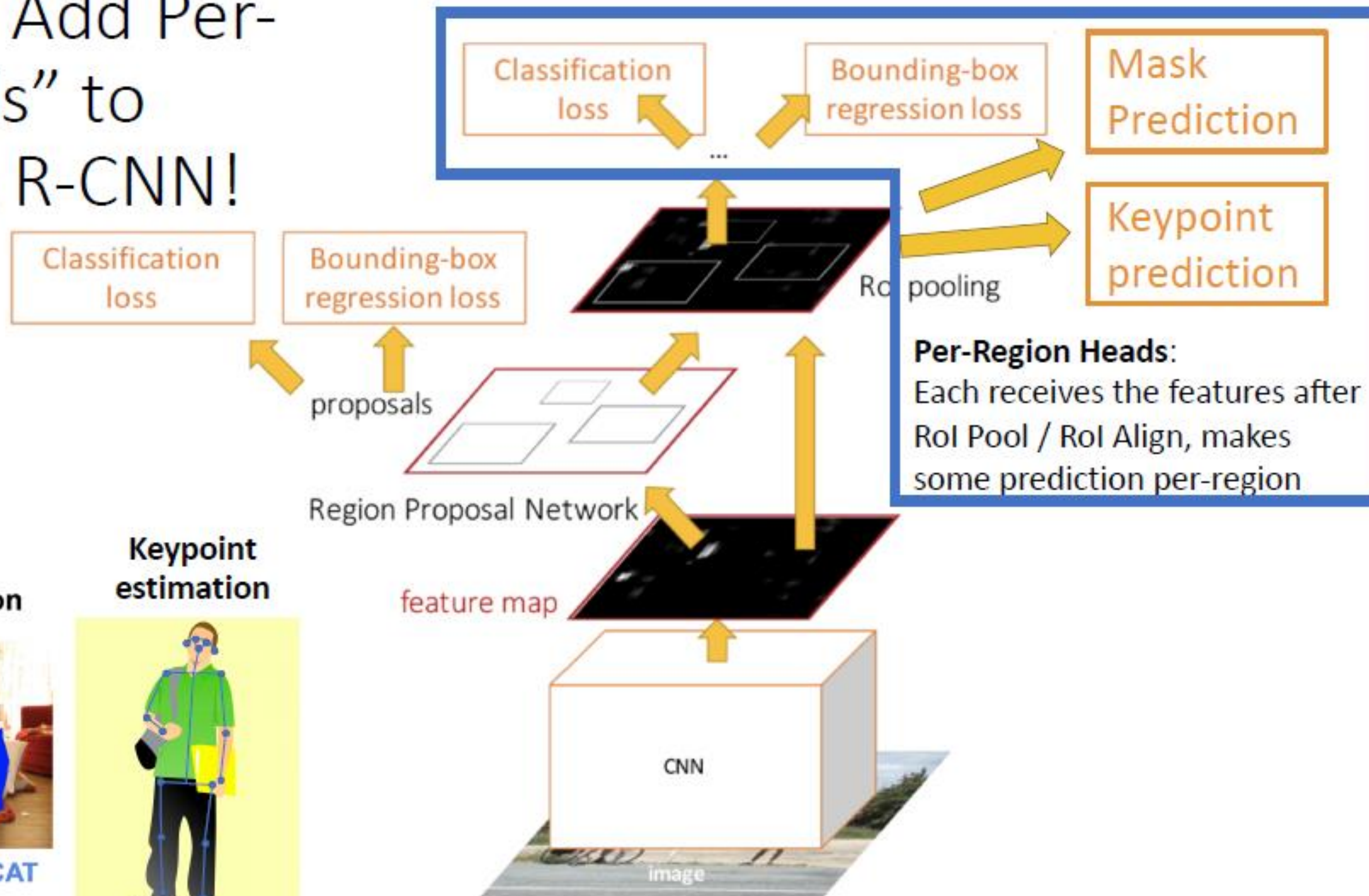
- Add an extra "Keypoint Prediction" head to perform joint Instance Segmentation and Pose Estimation
  - Example keypoints: Left / Right shoulder, elbow, wrist, hip, knee, ankle…

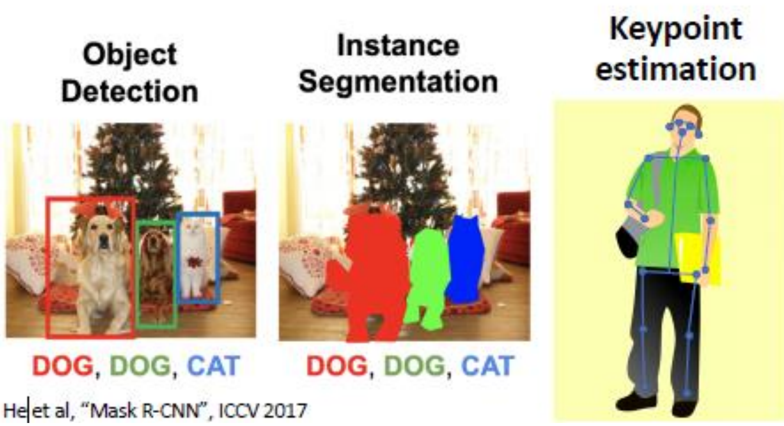


Important

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