Lecture 14:
Image Segmentation

Admin: Midterm + A3 Grades

Midterm grades: Should be out tomorrow

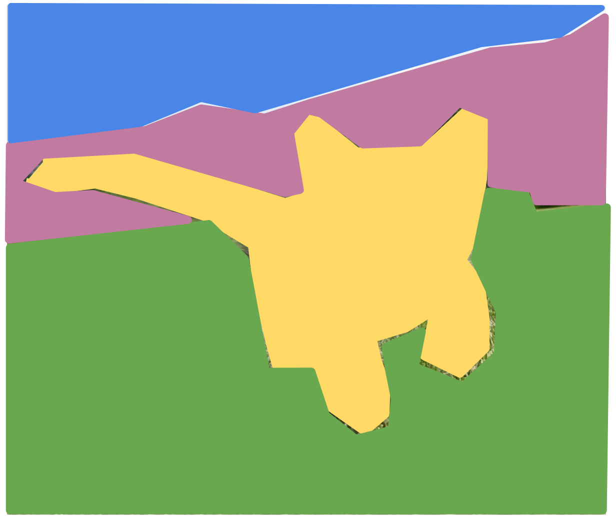
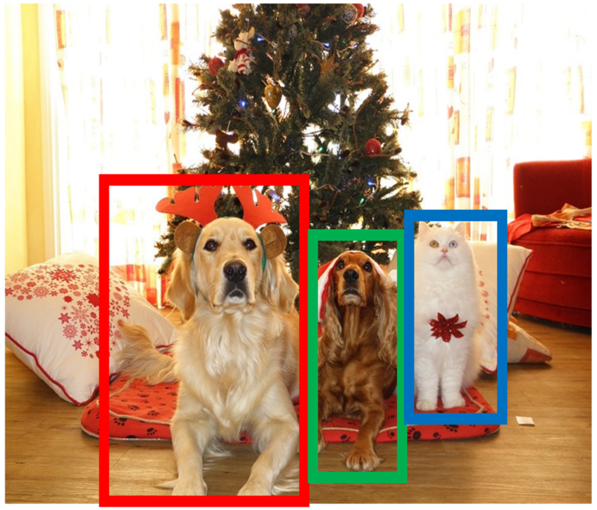
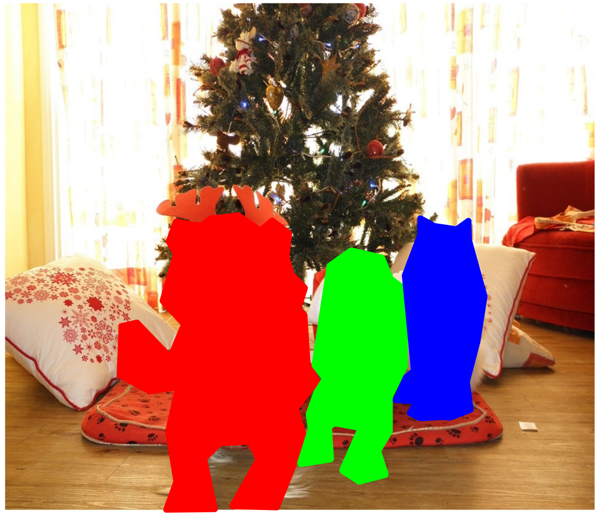
A3 grades: Later this week or early next week

A4 Update

Will be out tomorrow (?!?)

Due 2 weeks after release – will update calendar

Last Time: Localization Tasks

Classification	Semantic Segmentation	Object Detection	Instance Segmentation
			
CAT	GRASS, CAT, TREE, SKY	DOG, DOG, CAT	DOG, DOG, CAT

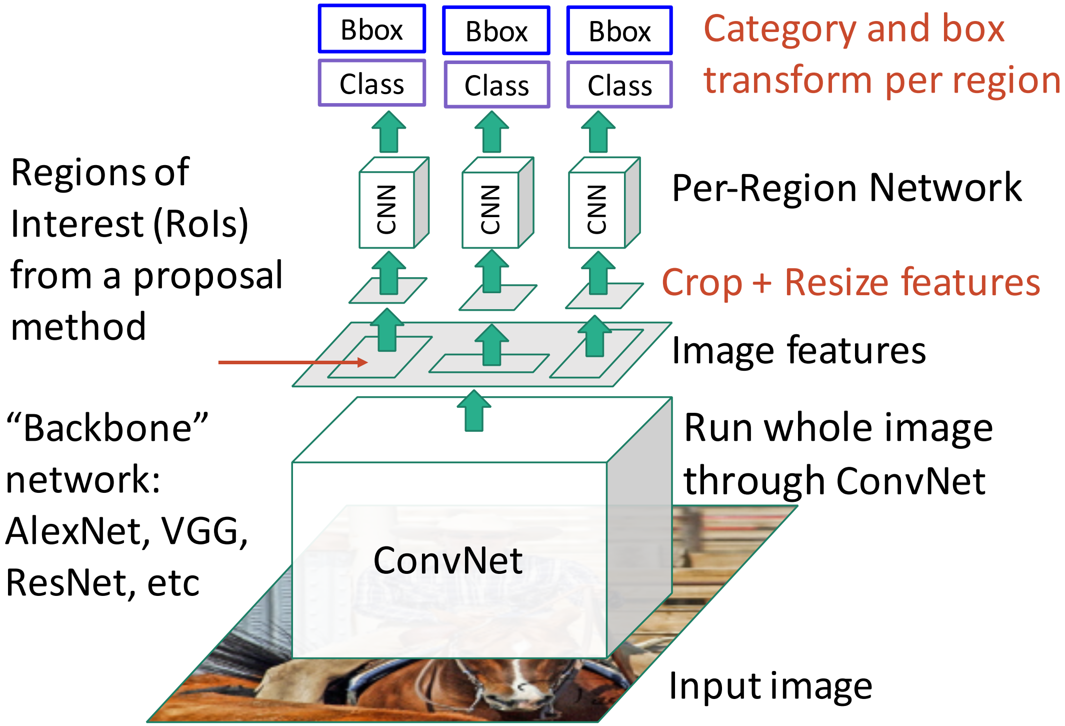
No spatial extent

No objects, just pixels

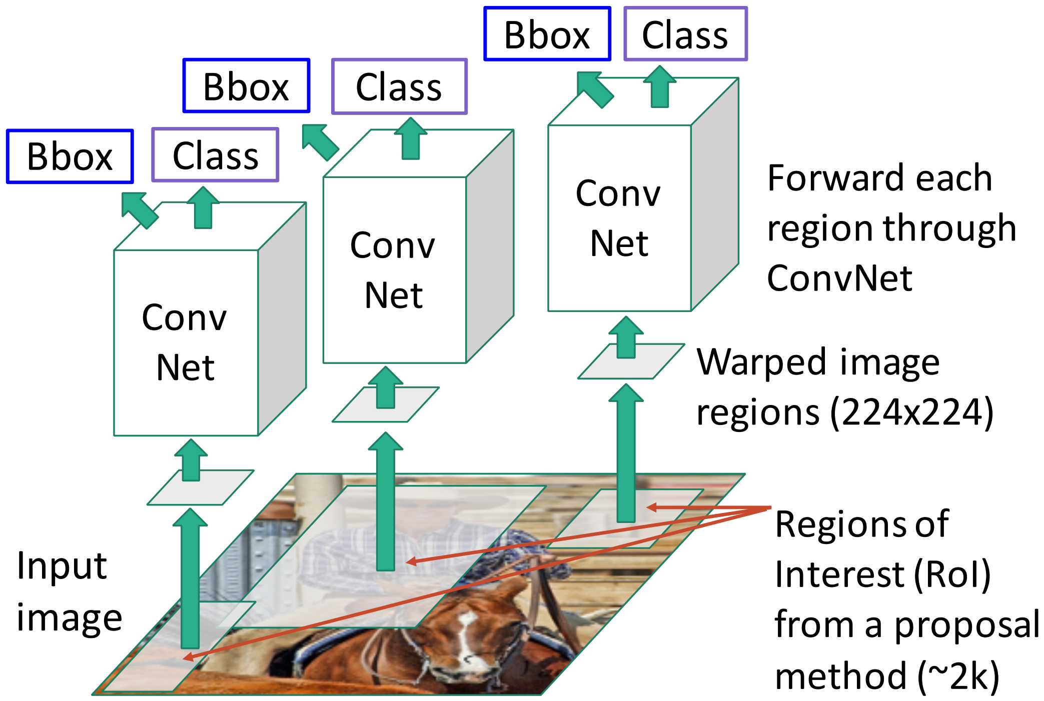
Multiple Objects

Last Time: Fast R-CNN

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



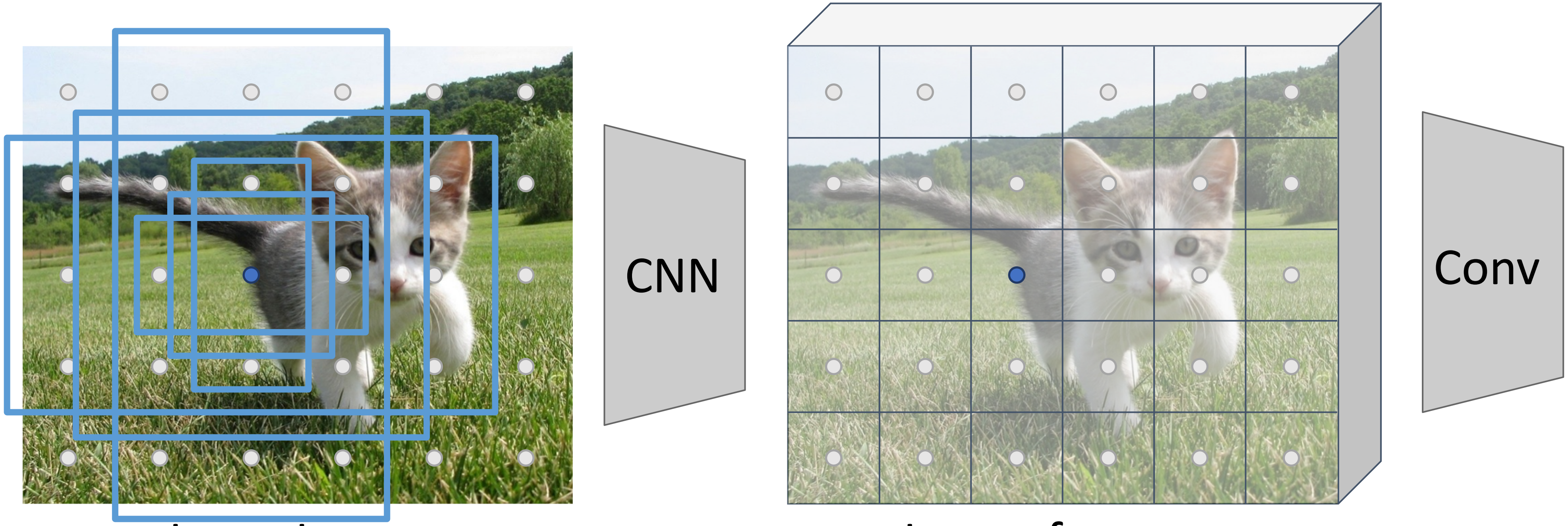
"Slow" R-CNN: Apply differentiable cropping to shared image features



Last Time: Region Proposal Network (RPN)

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input



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

Image features
(e.g. 512 x 5 x 6)

→ Anchor is object?
2K x 5 x 6

→ Anchor transforms
4K x 5 x 6

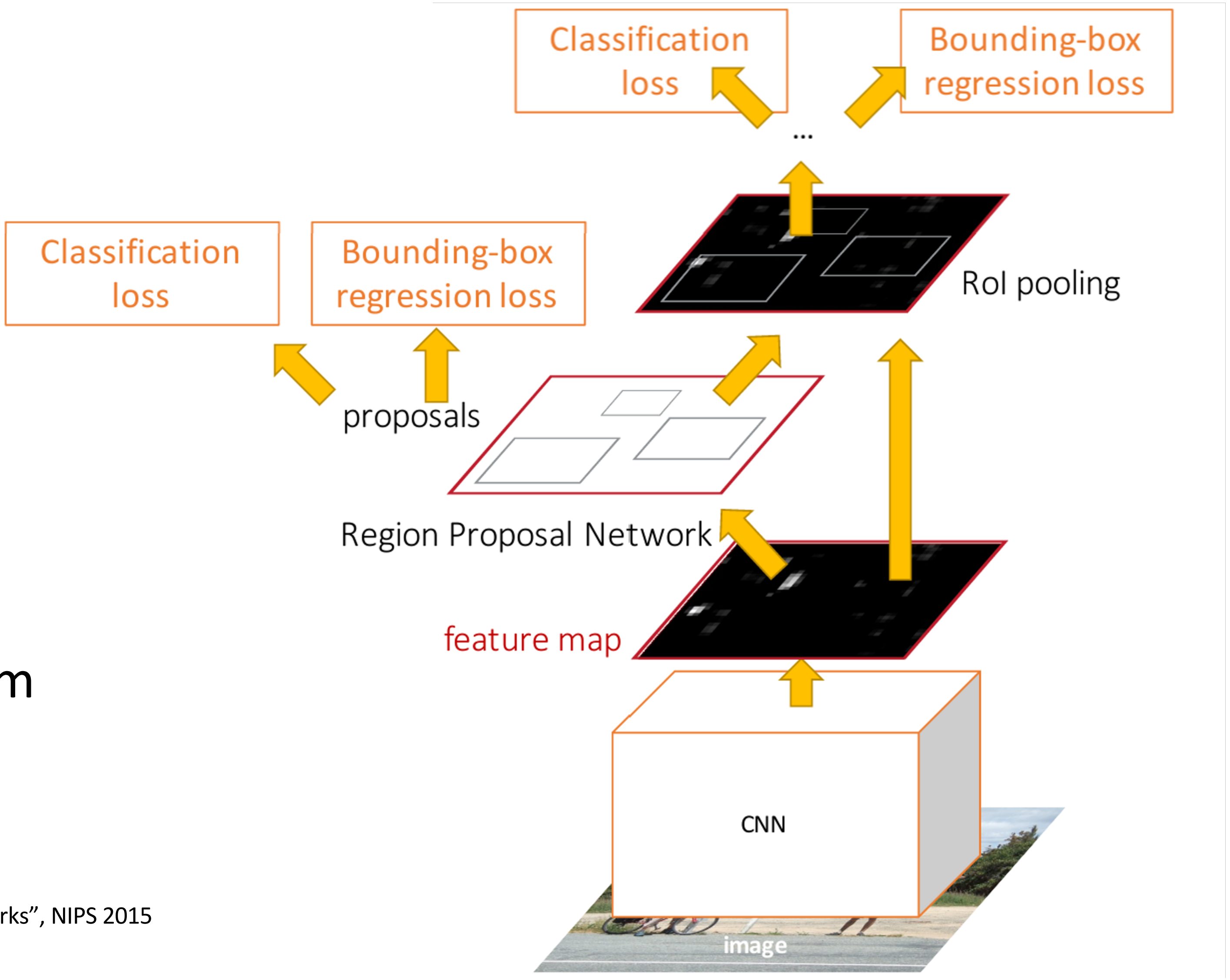
At test-time, sort all K*5*6 boxes by their positive score, take top 300 as our region proposals

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

Last Time: Faster 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 proposal box
3. **Object classification**: classify proposals as background / object class
4. **Object regression**: predict transform from proposal box to object box

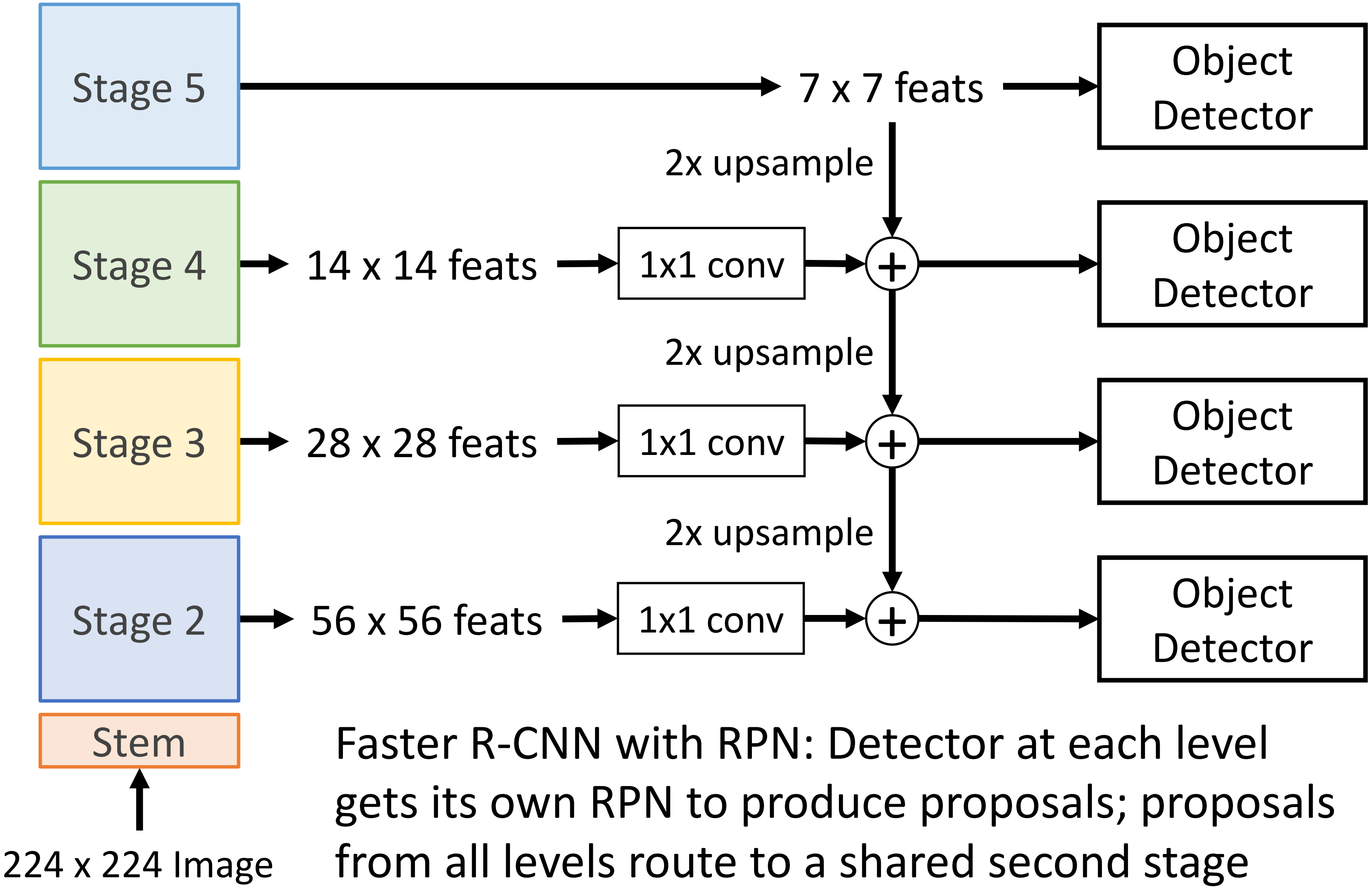


Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015

Last Time: Feature Pyramid Network (FPN)

Add *top down connections* that feed information from high level features back down to lower level features

Efficient multiscale features where all levels benefit from the whole backbone! Widely used in practice



Faster R-CNN with RPN: Detector at each level gets its own RPN to produce proposals; proposals from all levels route to a shared second stage

Lin et al, "Feature Pyramid Networks for Object Detection", ICCV 2017

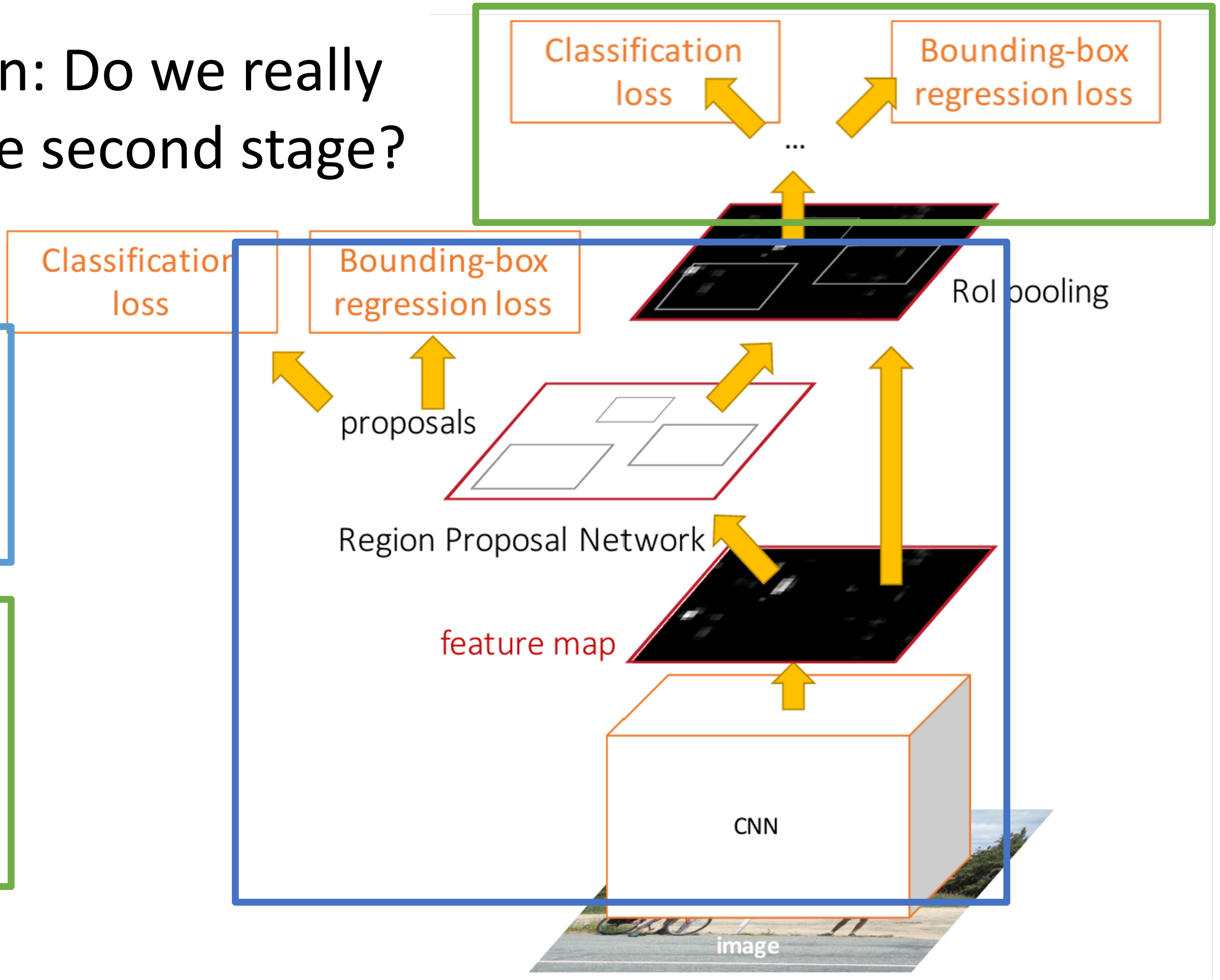
Two Stage Object Detectors

Question: Do we really need the second stage?

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

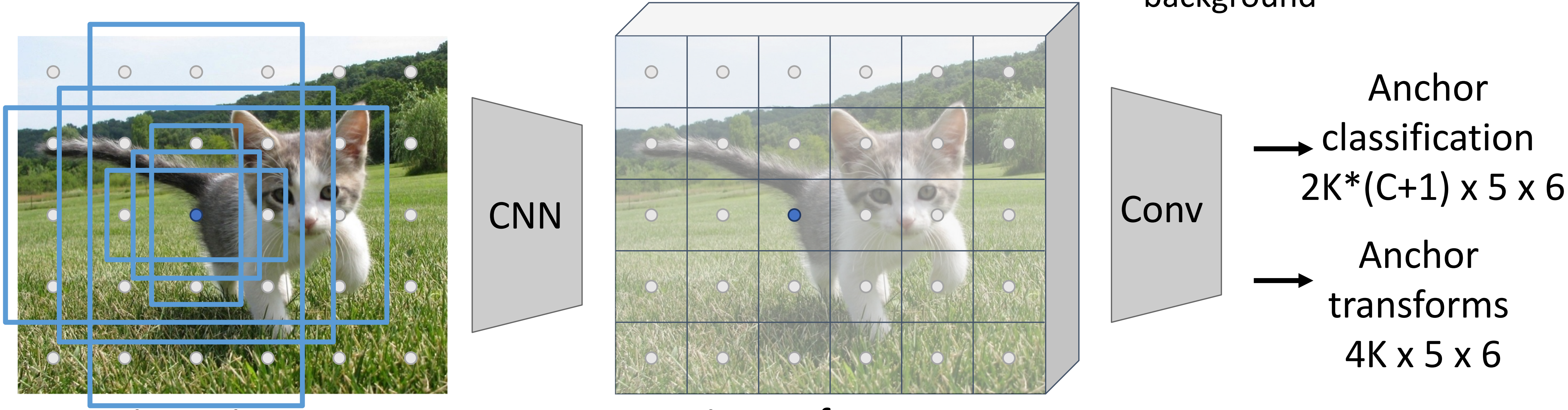


Single-Stage Detectors: RetinaNet

Similar to RPN – but rather than classify anchors as object/no object, directly predict object category (among C categories) or background

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input



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

Image features
(e.g. 512 x 5 x 6)

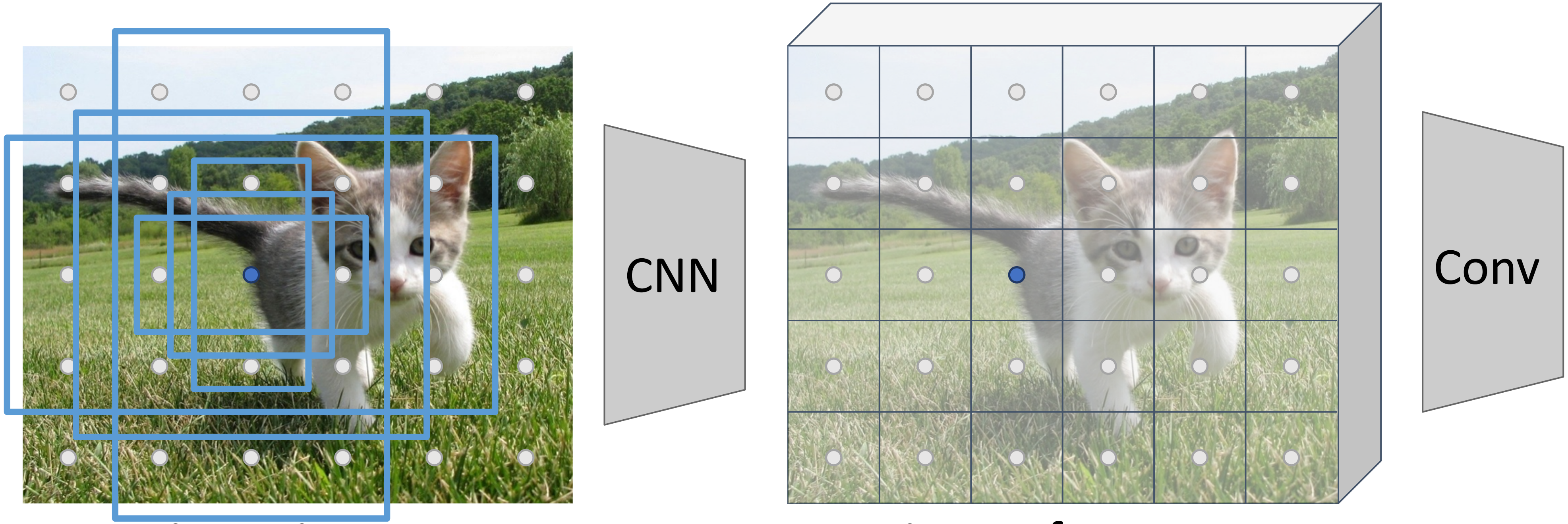
Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

Single-Stage Detectors: RetinaNet

Problem: class imbalance – many more background anchors vs non-background

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input



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

Image features
(e.g. 512 x 5 x 6)

Anchor classification
2K*(C+1) x 5 x 6

Anchor transforms
4K x 5 x 6

Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

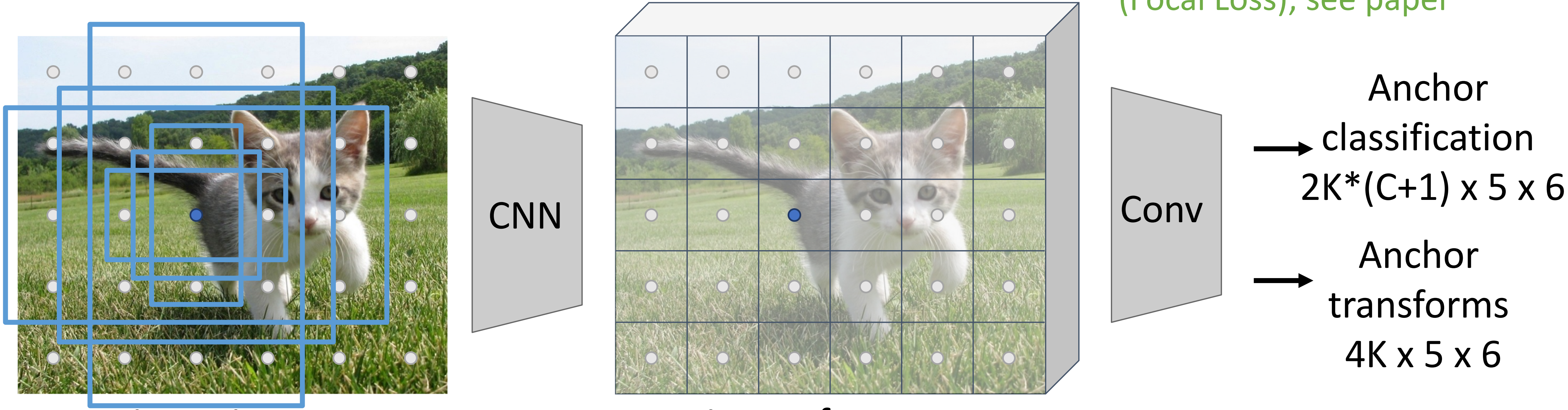
Single-Stage Detectors: RetinaNet

Problem: class imbalance – many more background anchors vs non-background

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input

Solution: new loss function (Focal Loss); see paper



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

Image features (e.g. 512 x 5 x 6)

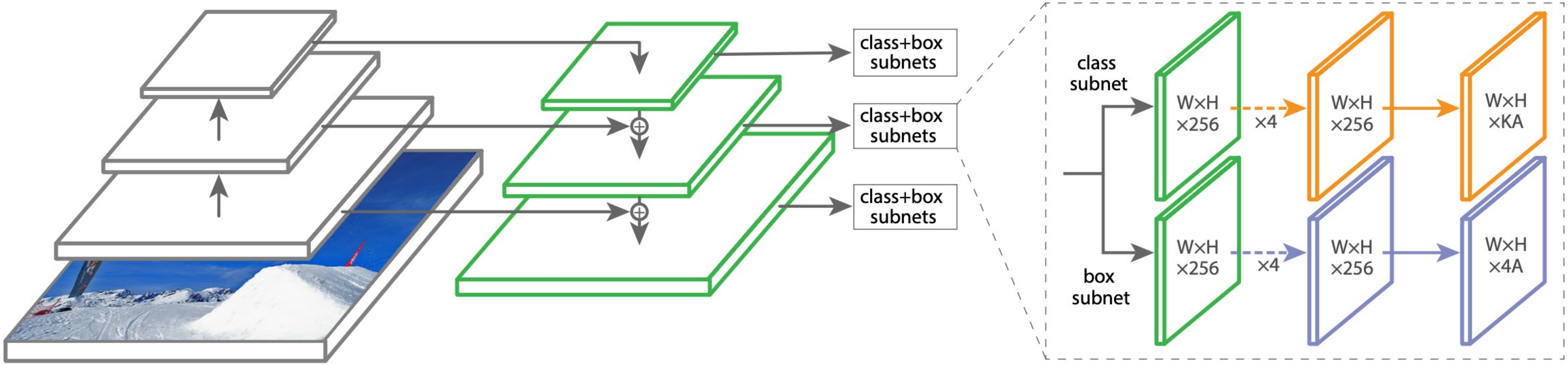
$$\mathrm{CE}(p_t) = -\log(p_t)$$

$$\mathrm{FL}(p_t) = -(1-p_t)^\gamma \log(p_t)$$

Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

Single-Stage Detectors: RetinaNet

In practice, RetinaNet also uses Feature Pyramid Network to handle multiscale



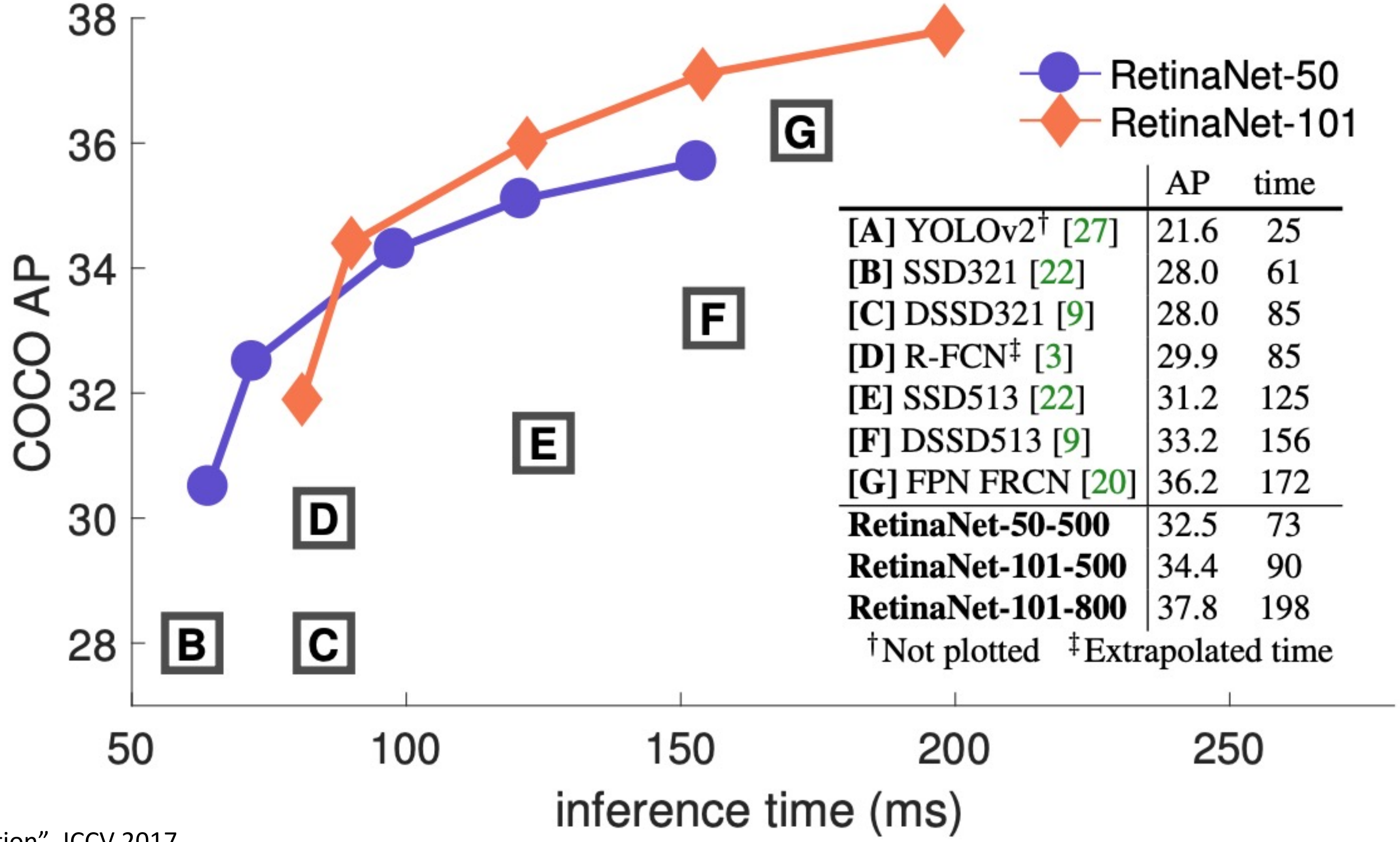
(a) ResNet (b) feature pyramid net (c) class subnet (top) (d) box subnet (bottom)

Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

Figure credit: Lin et al, ICCV 2017

Single-Stage Detectors: RetinaNet

Single-Stage detectors can be much faster than two-stage detectors



	AP	time
[A] YOLOv2† [27]	21.6	25
[B] SSD321 [22]	28.0	61
[C] DSSD321 [9]	28.0	85
[D] R-FCN‡ [3]	29.9	85
[E] SSD513 [22]	31.2	125
[F] DSSD513 [9]	33.2	156
[G] FPN FRCN [20]	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	37.8	198

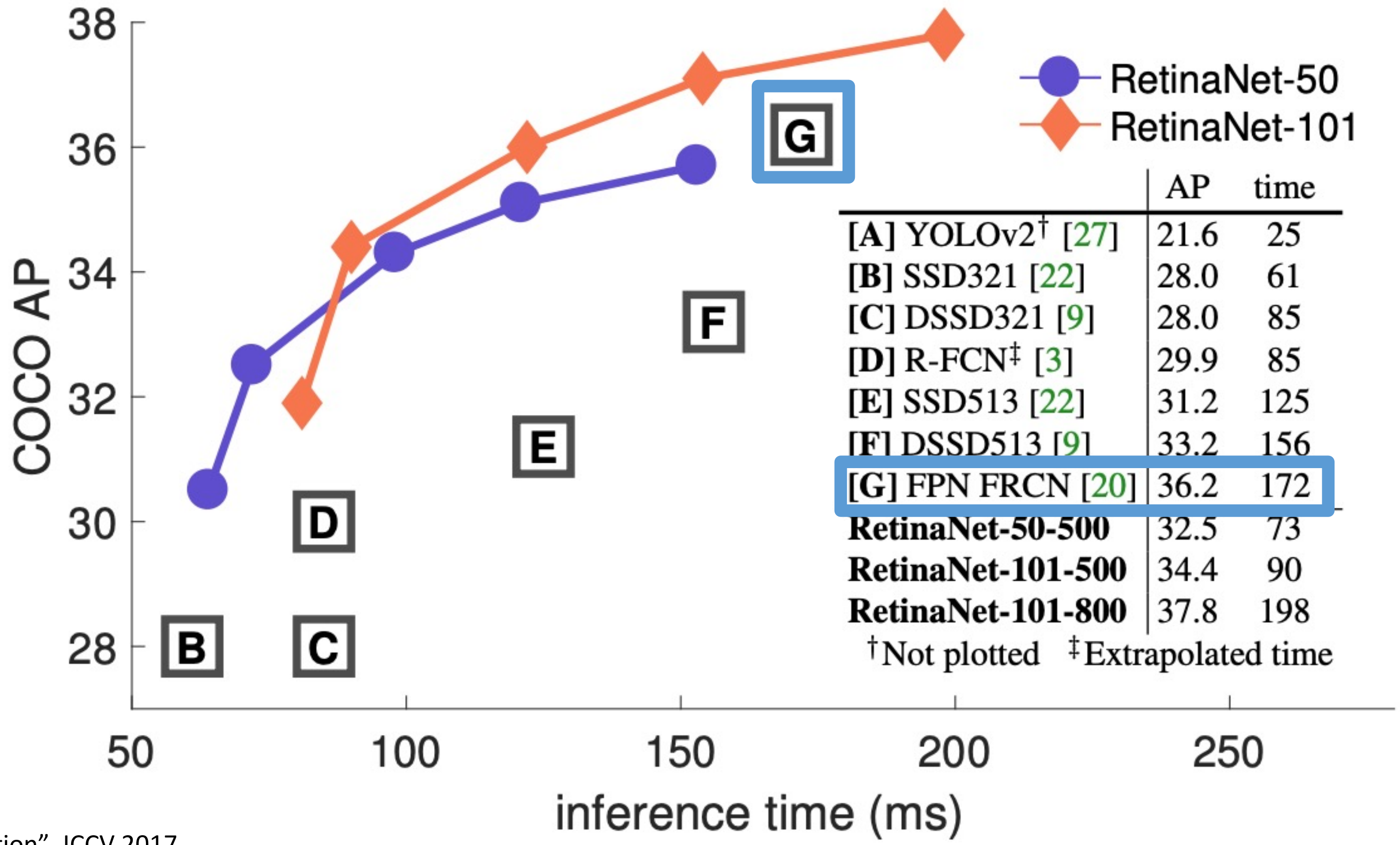
†Not plotted ‡Extrapolated time

Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

Figure credit: Lin et al, ICCV 2017

Single-Stage Detectors: RetinaNet

Single-Stage detectors can be much faster than two-stage detectors



	AP	time
[A] YOLOv2† [27]	21.6	25
[B] SSD321 [22]	28.0	61
[C] DSSD321 [9]	28.0	85
[D] R-FCN‡ [3]	29.9	85
[E] SSD513 [22]	31.2	125
[F] DSSD513 [9]	33.2	156
[G] FPN FRCN [20]	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	37.8	198

†Not plotted ‡Extrapolated time

Faster R-CNN with Feature Pyramid Network

Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

Figure credit: Lin et al, ICCV 2017

Anchor-Free Detectors

Can we do object detection without anchors?

CornerNet: Law and Deng, "CornerNet: Detecting Objects as Paired Keypoints", ECCV 2018

CenterNet: Zhou et al, "Objects as Points", arXiv 2019

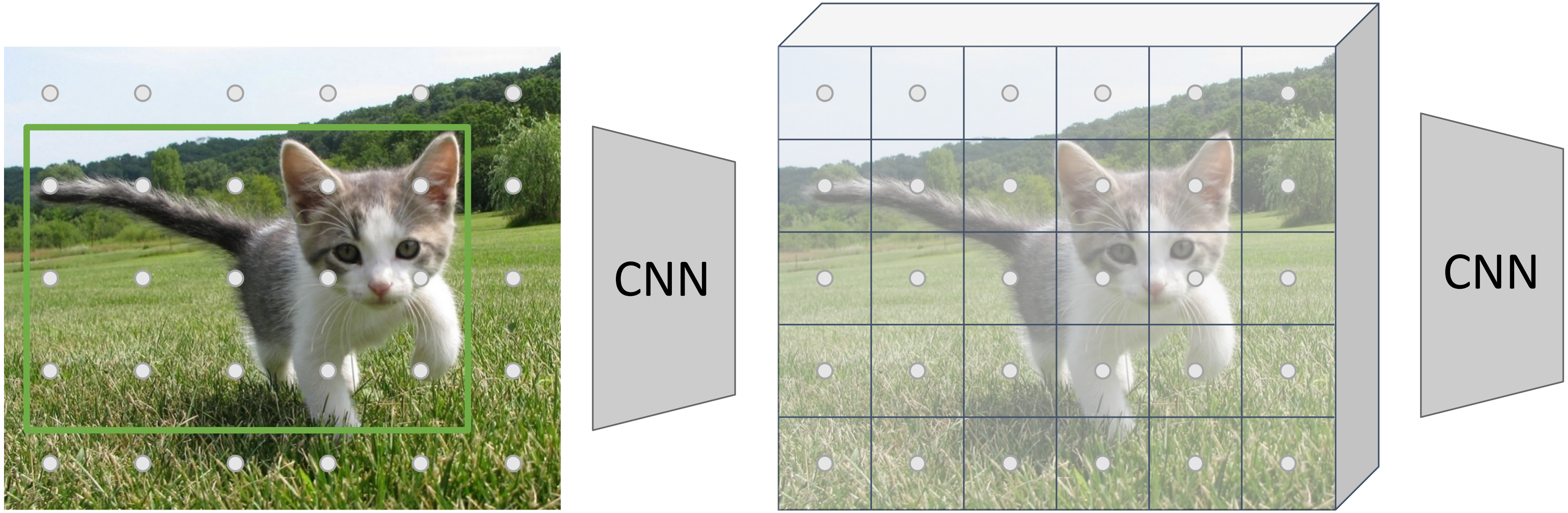
FCOS: Tian et al, "FCOS: Fully Convolutional One-Stage Object Detection", ICCV 2019

Single-Stage Detectors: FCOS

"Anchor-free" detector

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input



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

Image features
(e.g. 512 x 5 x 6)

Tian et al, "FCOS: Fully Convolutional One-Stage Object Detection", ICCV 2019

Single-Stage Detectors: FCOS

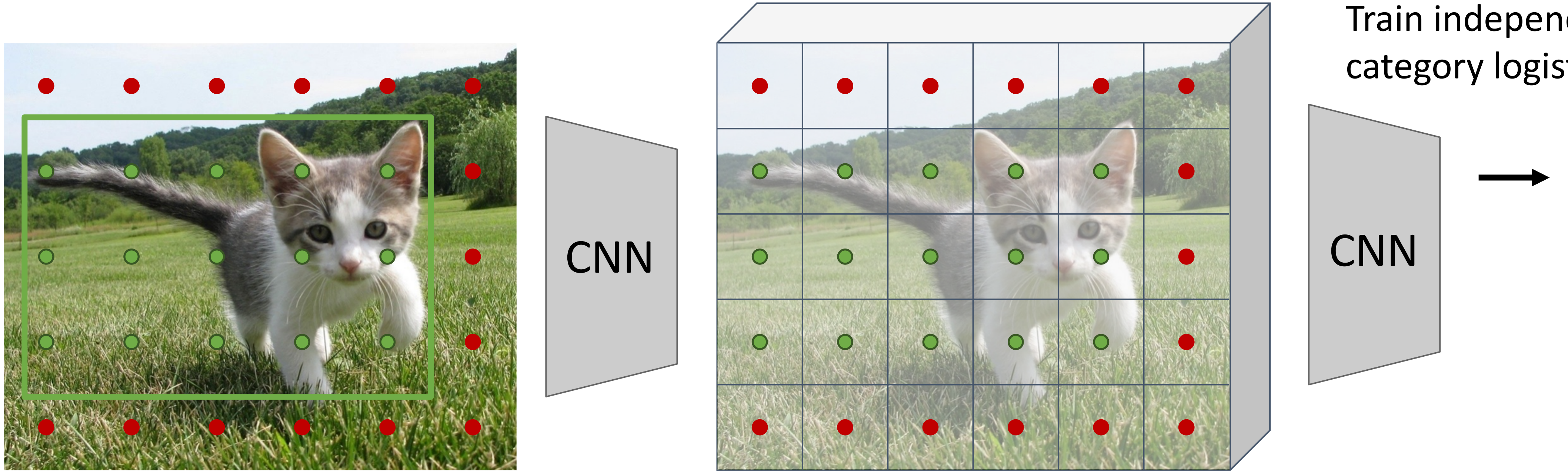
"Anchor-free" detector

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input

Classify points as positive if they fall into a GT box, or negative if they don't

Train independent per-category logistic regressors



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

Image features
(e.g. 512 x 5 x 6)

Class scores
C x 5 x 6

Tian et al, "FCOS: Fully Convolutional One-Stage Object Detection", ICCV 2019

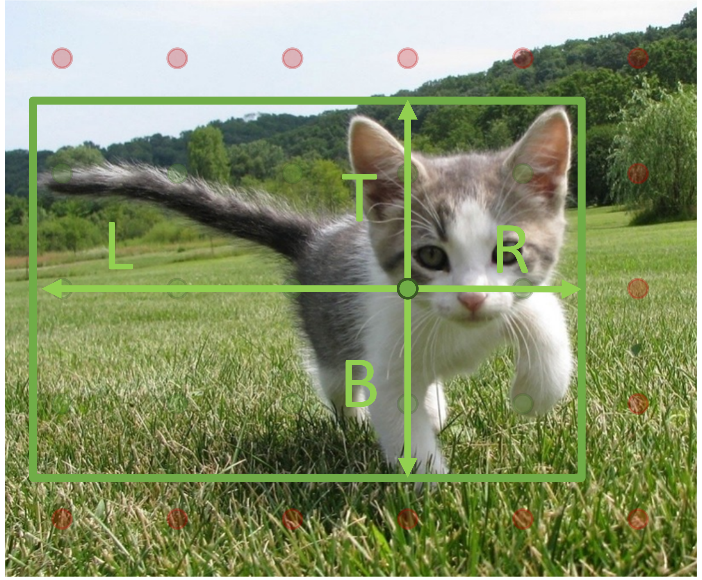
Single-Stage Detectors: FCOS

"Anchor-free" detector

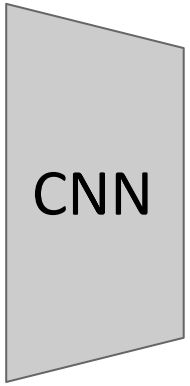
Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input

For positive points, also regress distance to left, right, top, and bottom of ground-truth box (with L2 loss)



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



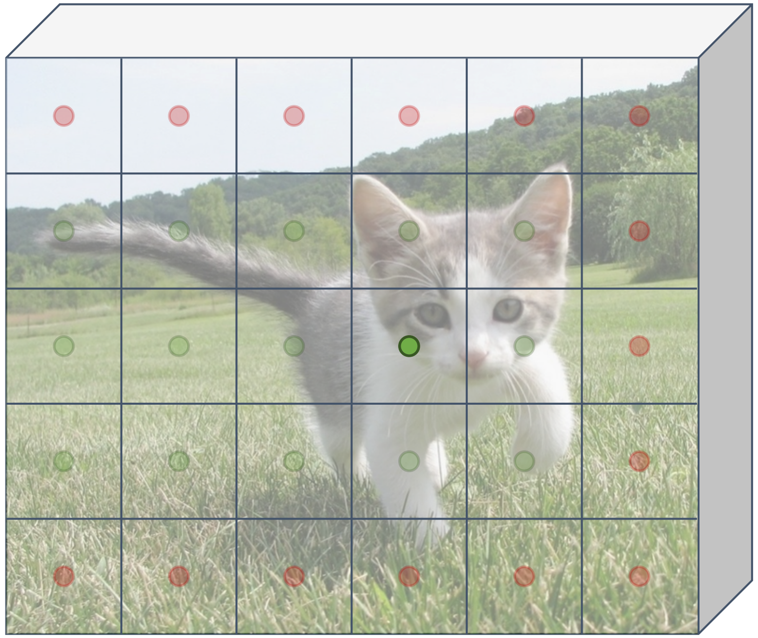
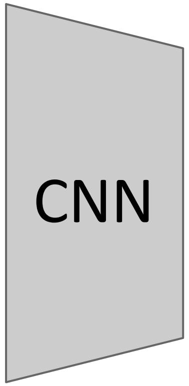


Image features
(e.g. 512 x 5 x 6)



→ Class scores
C x 5 x 6

→ Box edges
4 x 5 x 6

Tian et al, "FCOS: Fully Convolutional One-Stage Object Detection", ICCV 2019

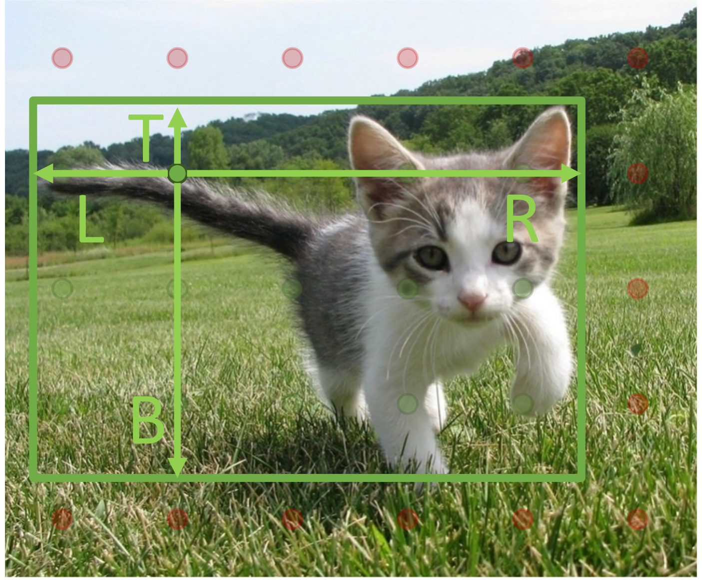
Single-Stage Detectors: FCOS

"Anchor-free" detector

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input

For positive points, also regress distance to left, right, top, and bottom of ground-truth box (with L2 loss)



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

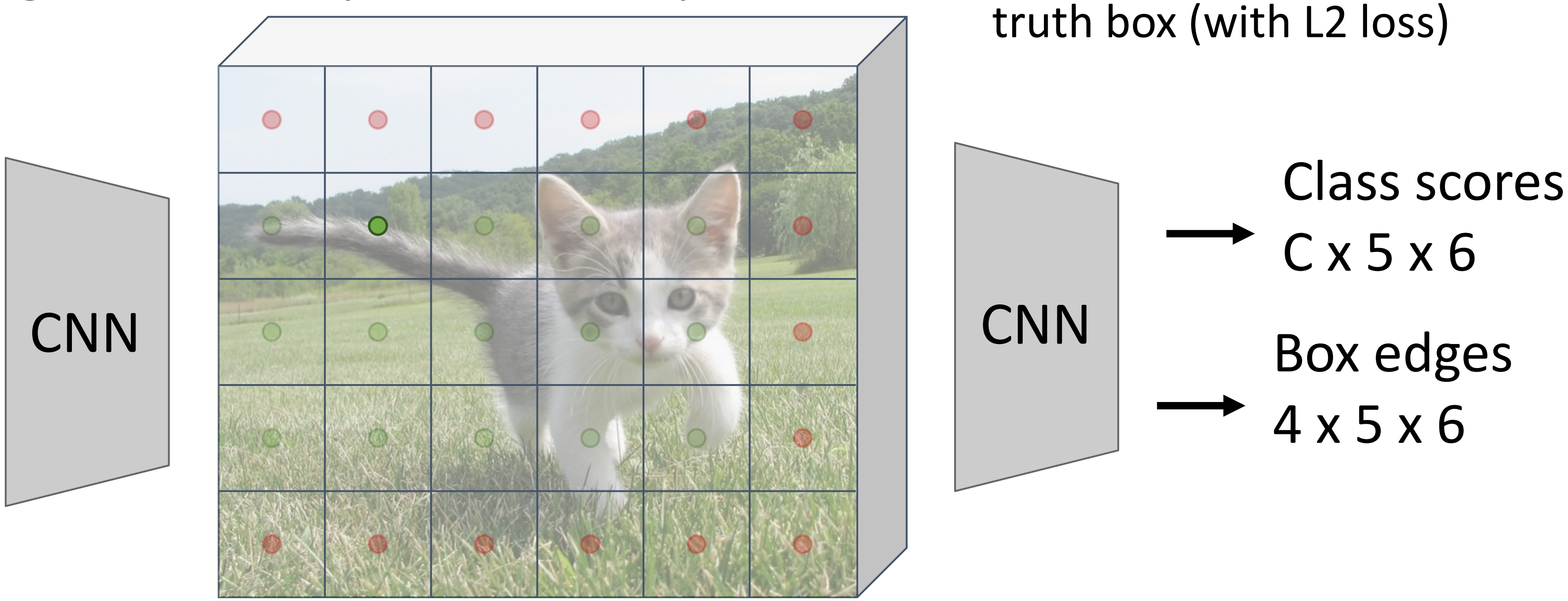


Image features
(e.g. 512 x 5 x 6)

Tian et al, "FCOS: Fully Convolutional One-Stage Object Detection", ICCV 2019

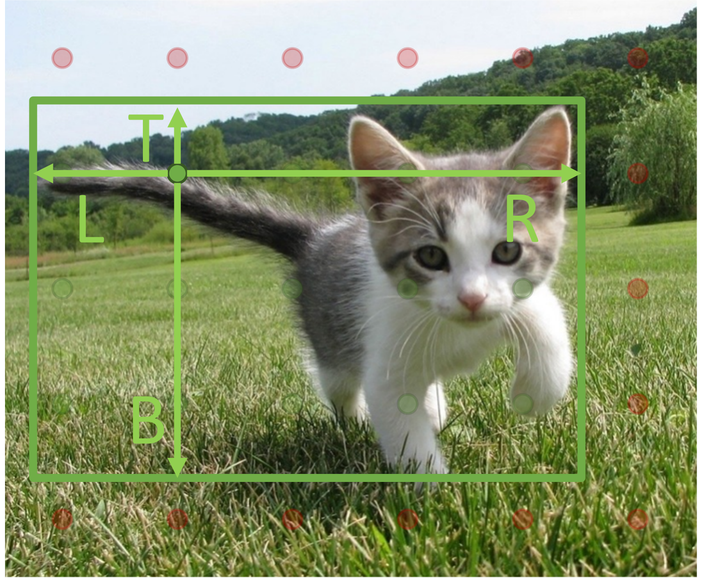
Single-Stage Detectors: FCOS

"Anchor-free" detector

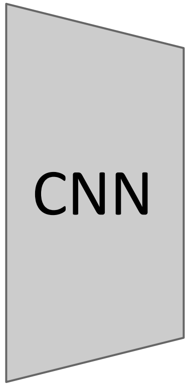
Finally, predict "centerness" for all positive points (using logistic regression loss)

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input



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



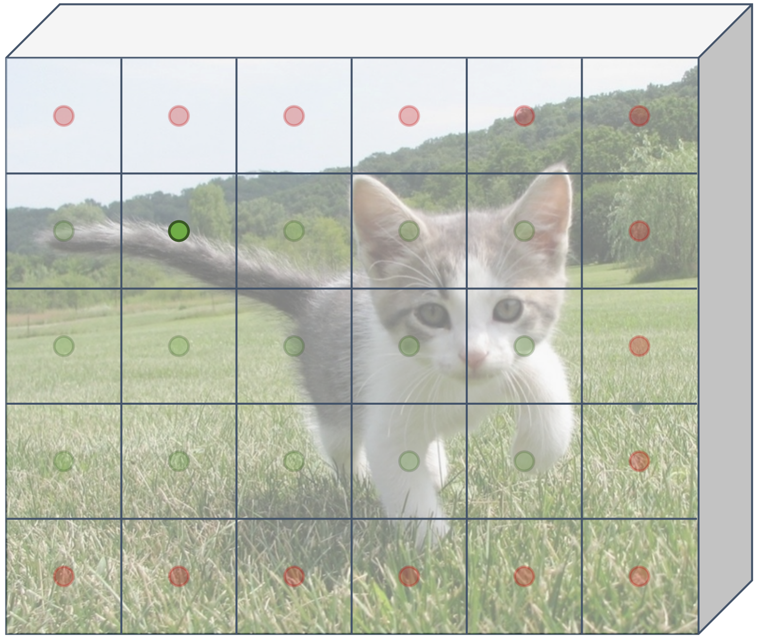
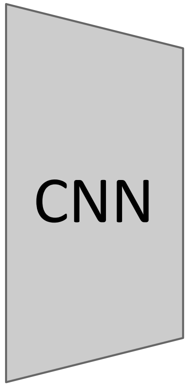


Image features
(e.g. 512 x 5 x 6)



→ Class scores
C x 5 x 6

→ Box edges
4 x 5 x 6

→ Centerness
1 x 5 x 6

$$centerness = \sqrt{\frac{\min(L,R)}{\max(L,R)} \cdot \frac{\min(T,B)}{\max(T,B)}}$$

Ranges from 1 at box center to 0 at box edge

Tian et al, "FCOS: Fully Convolutional One-Stage Object Detection", ICCV 2019

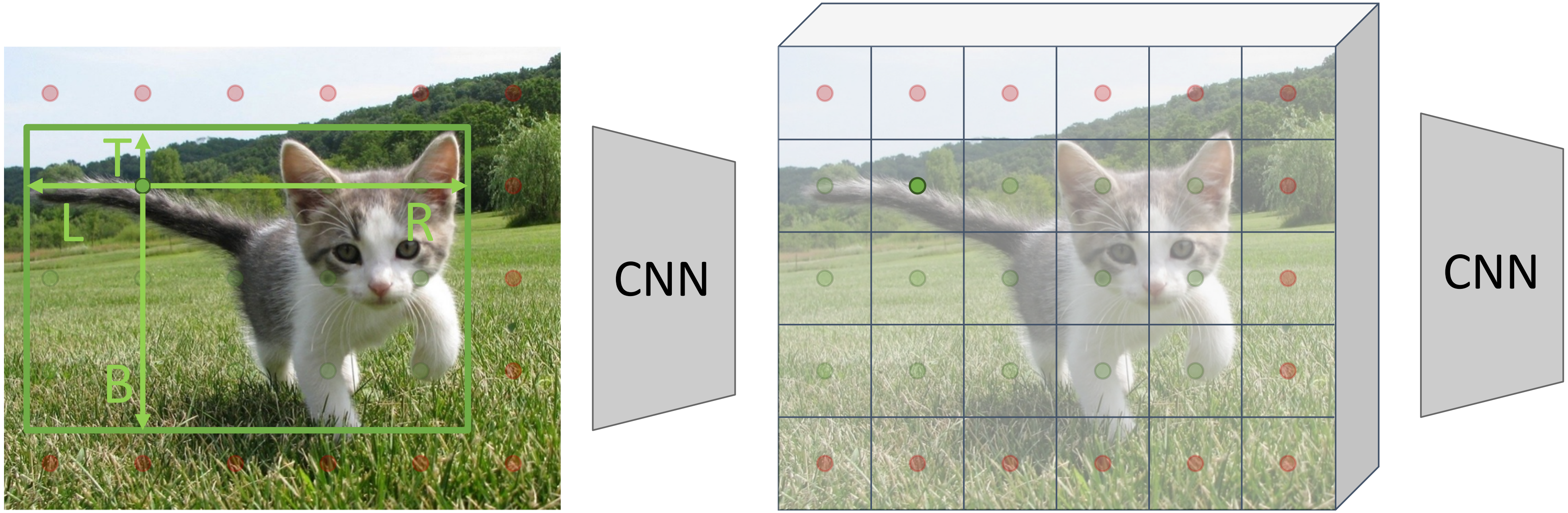
Single-Stage Detectors: FCOS

"Anchor-free" detector

Test-time: predicted "confidence" for the box from each point is product of its class score and centerness

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input



Class scores
C x 5 x 6

Box edges
4 x 5 x 6

Centerness
1 x 5 x 6

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

Image features
(e.g. 512 x 5 x 6)

$$centerness = \sqrt{\frac{\min(L,R)}{\max(L,R)} \cdot \frac{\min(T,B)}{\max(T,B)}}$$

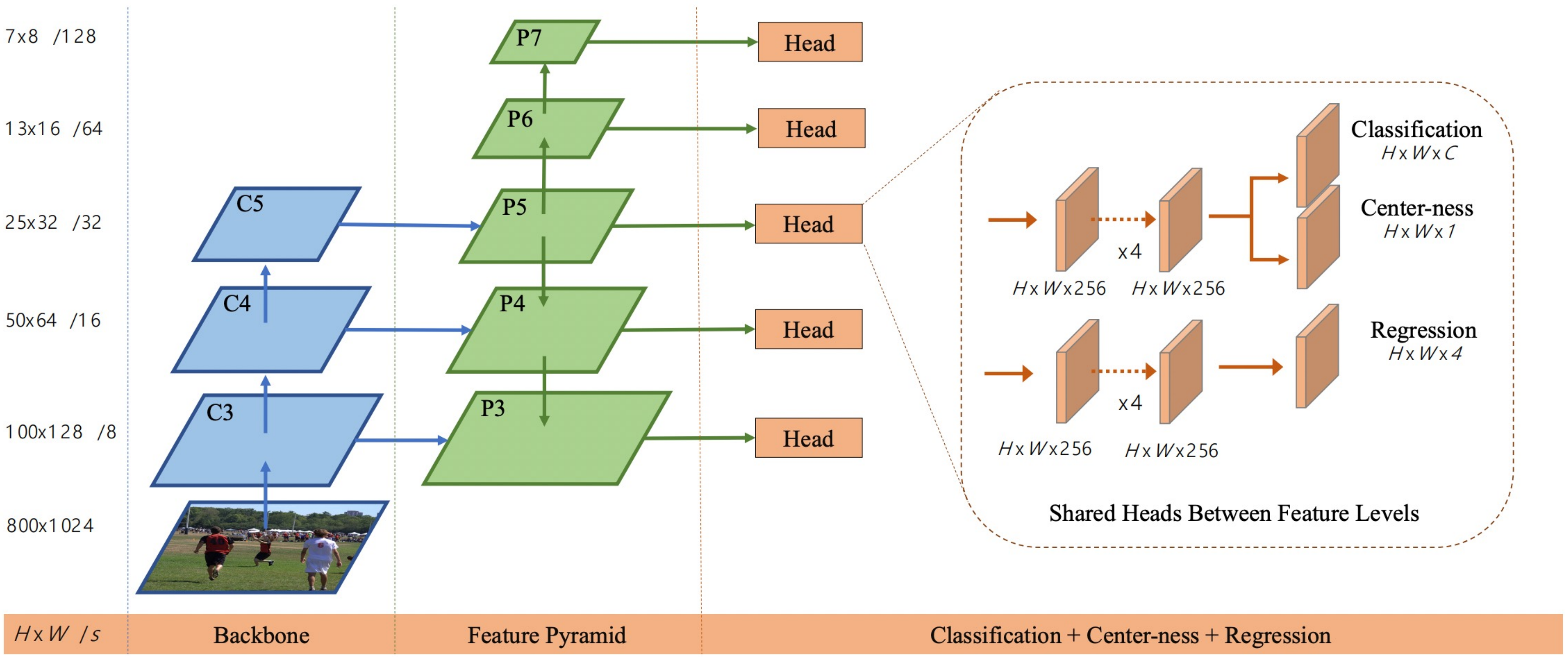
Ranges from 1 at box center to 0 at box edge

Tian et al, "FCOS: Fully Convolutional One-Stage Object Detection", ICCV 2019

Single-Stage Detectors: FCOS

"Anchor-free" detector

FCOS also uses a Feature Pyramid Network with heads shared across stages

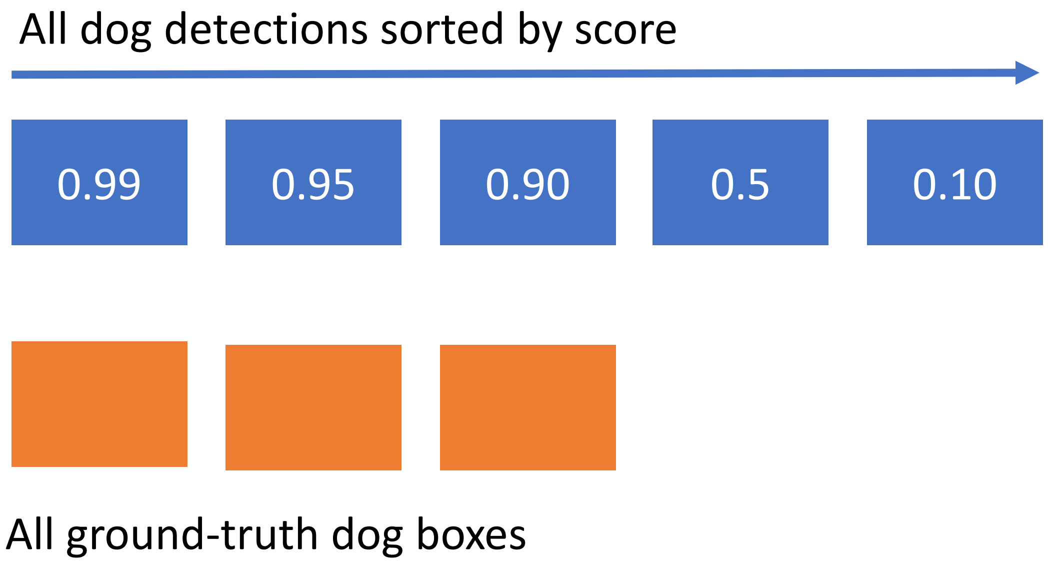


Tian et al, "FCOS: Fully Convolutional One-Stage Object Detection", ICCV 2019

Evaluating Object Detectors: Mean Average Precision (mAP)

1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve

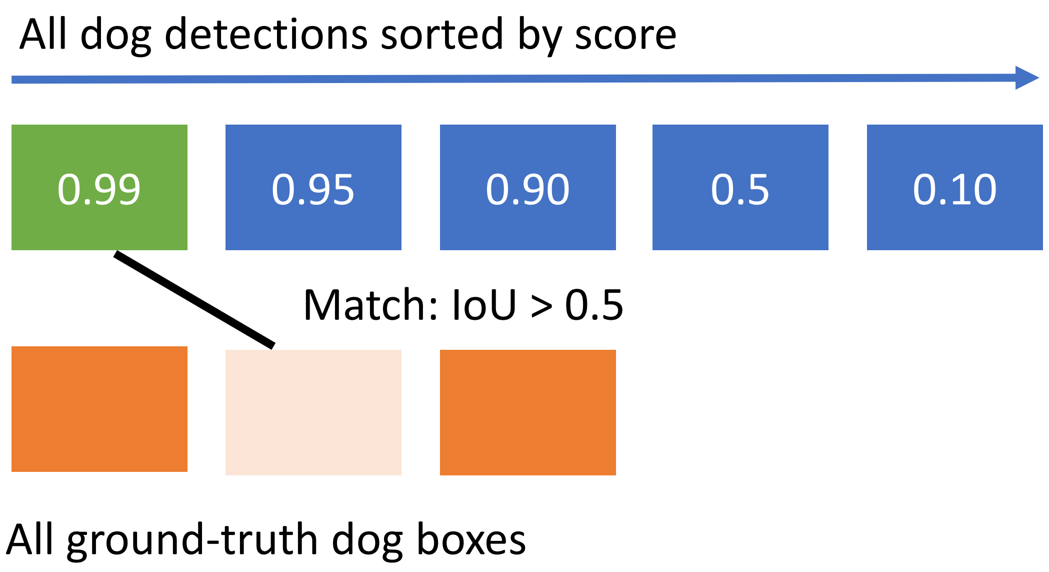
Evaluating Object Detectors: Mean Average Precision (mAP)



1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
   1. For each detection (highest score to lowest score)

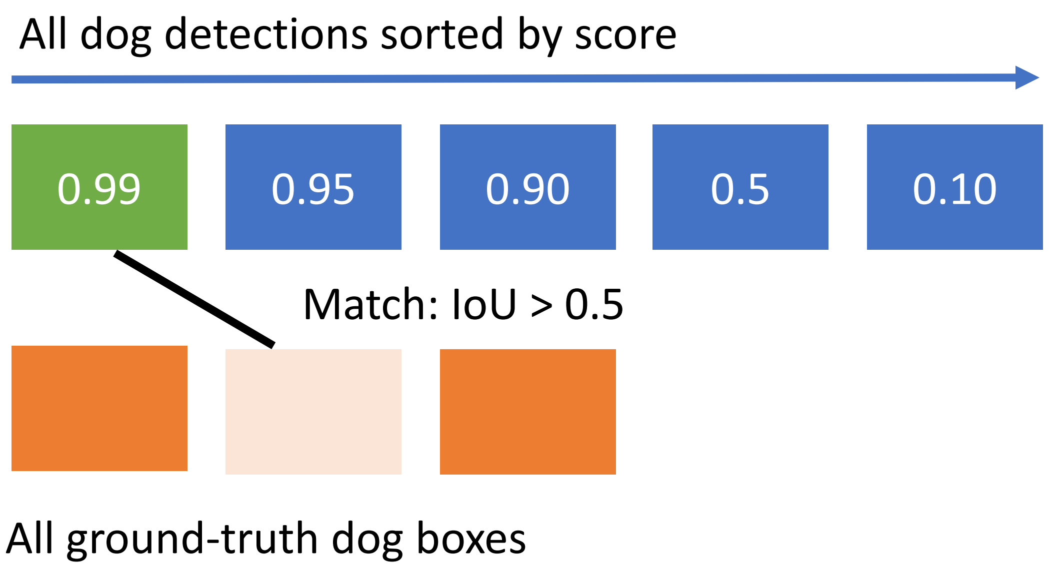
Evaluating Object Detectors: Mean Average Precision (mAP)

1. Run object detector on all test images (with NMS)
2. For each category, 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

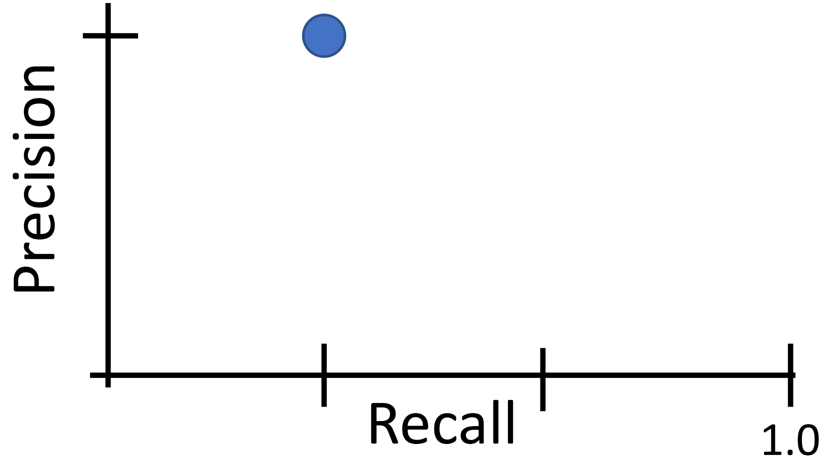


Evaluating Object Detectors: Mean Average Precision (mAP)

1. Run object detector on all test images (with NMS)
2. For each category, 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

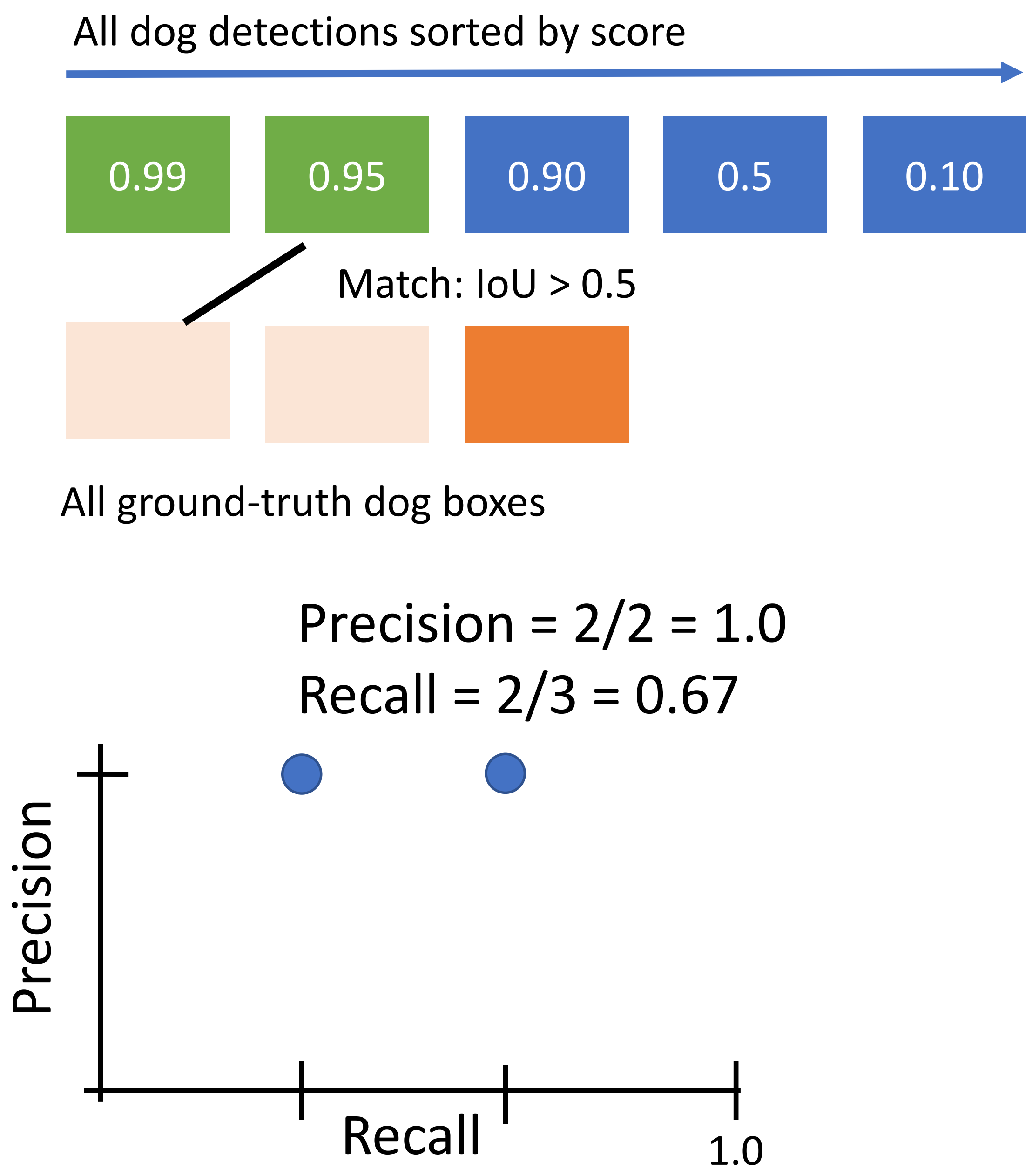


Precision = 1/1 = 1.0
Recall = 1/3 = 0.33



Evaluating Object Detectors: Mean Average Precision (mAP)

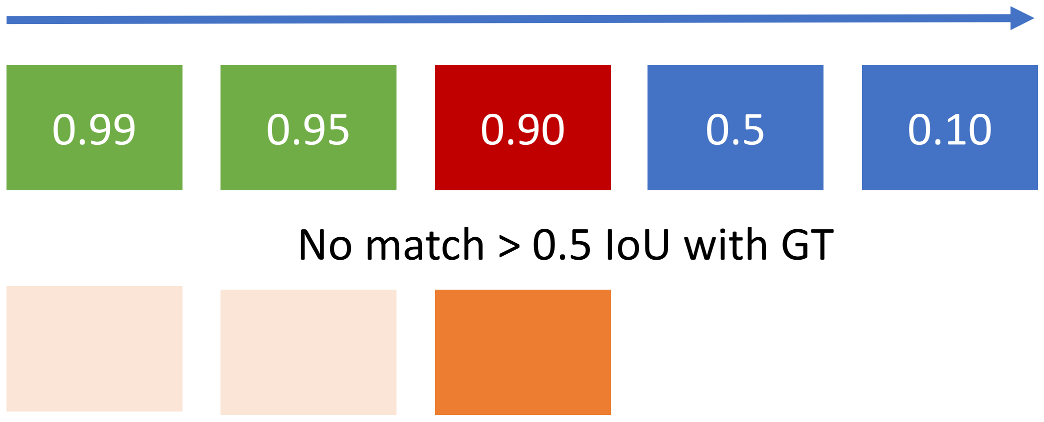
1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
    1. For each detection (highest score to lowest score)
        1. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
        2. Otherwise mark it as negative
        3. Plot a point on PR Curve



Evaluating Object Detectors: Mean Average Precision (mAP)

1. Run object detector on all test images (with NMS)
2. For each category, 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

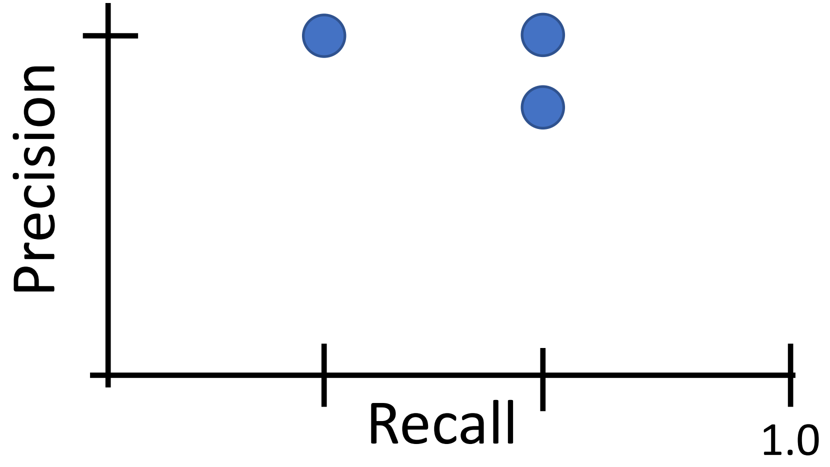
All dog detections sorted by score



All ground-truth dog boxes

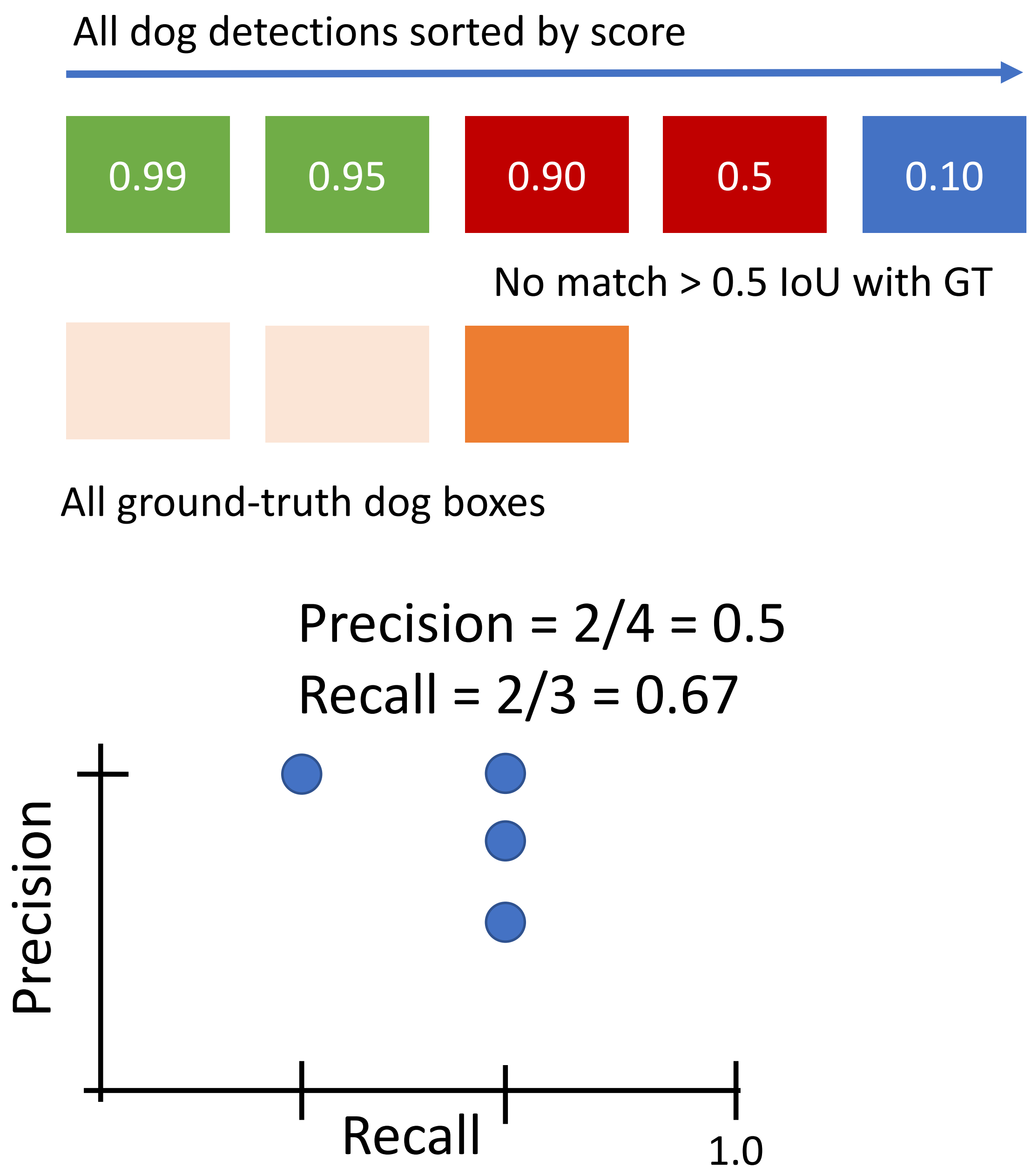
Precision = 2/3 = 0.67

Recall = 2/3 = 0.67



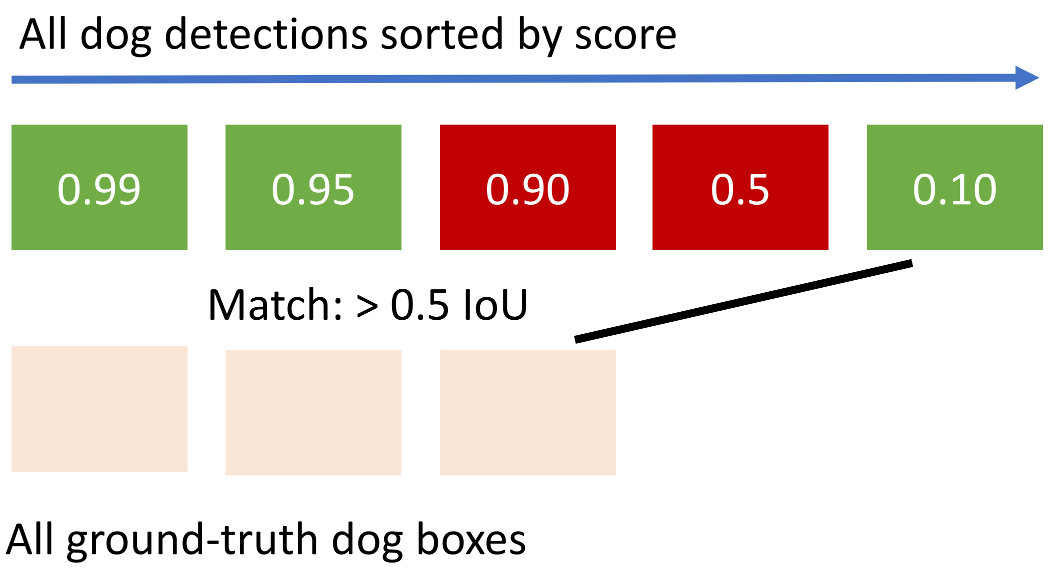
Evaluating Object Detectors: Mean Average Precision (mAP)

1. Run object detector on all test images (with NMS)
2. For each category, 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



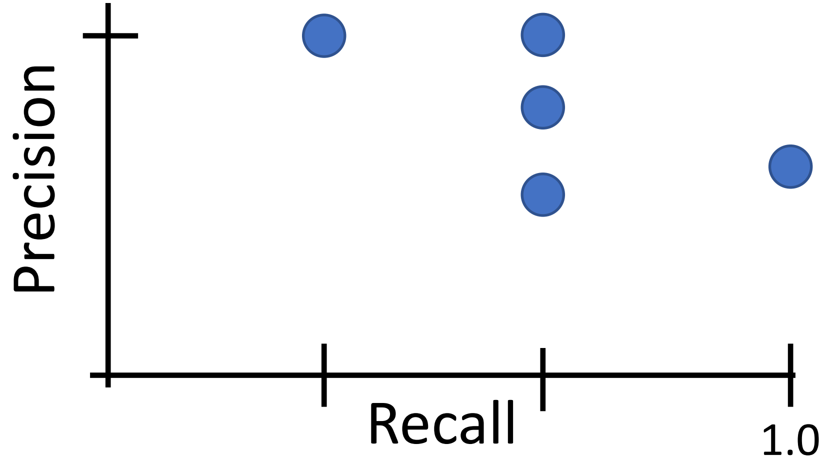
Evaluating Object Detectors: Mean Average Precision (mAP)

1. Run object detector on all test images (with NMS)
2. For each category, 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

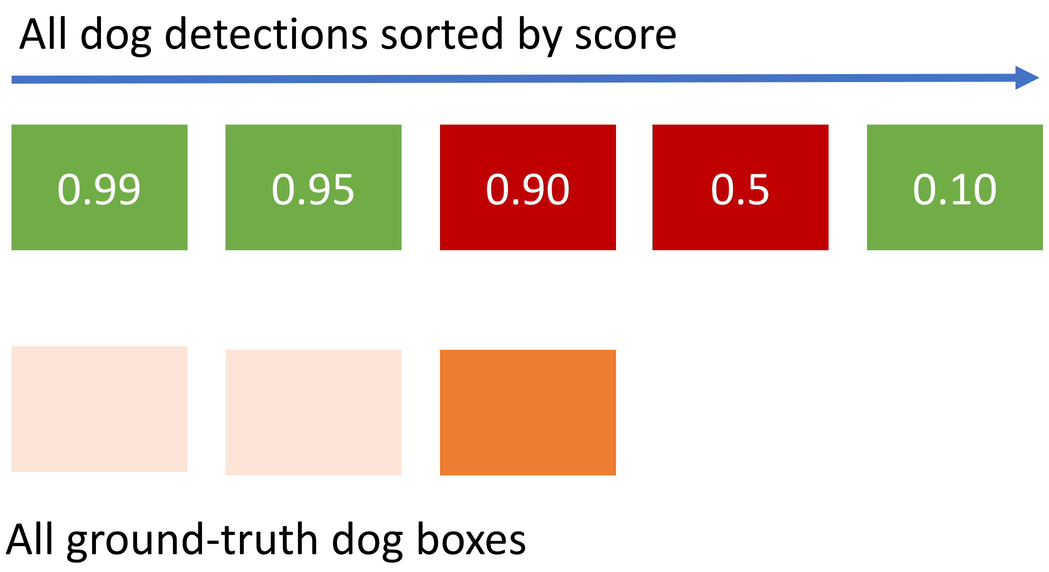


Precision = 3/5 = 0.6

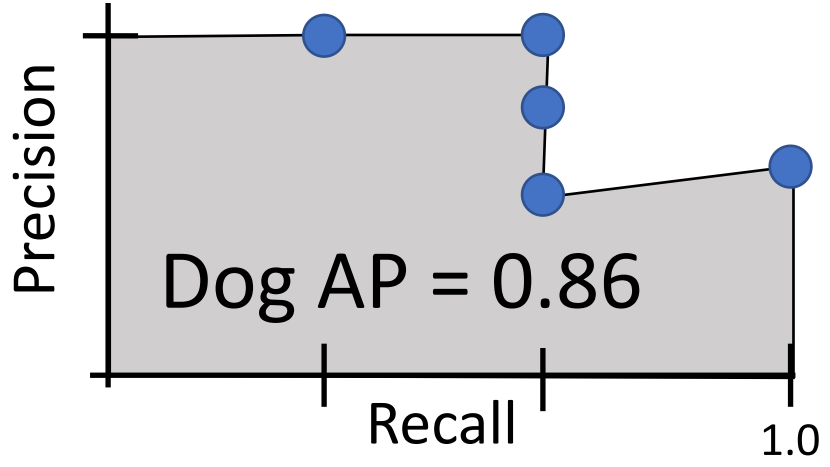
Recall = 3/3 = 1.0



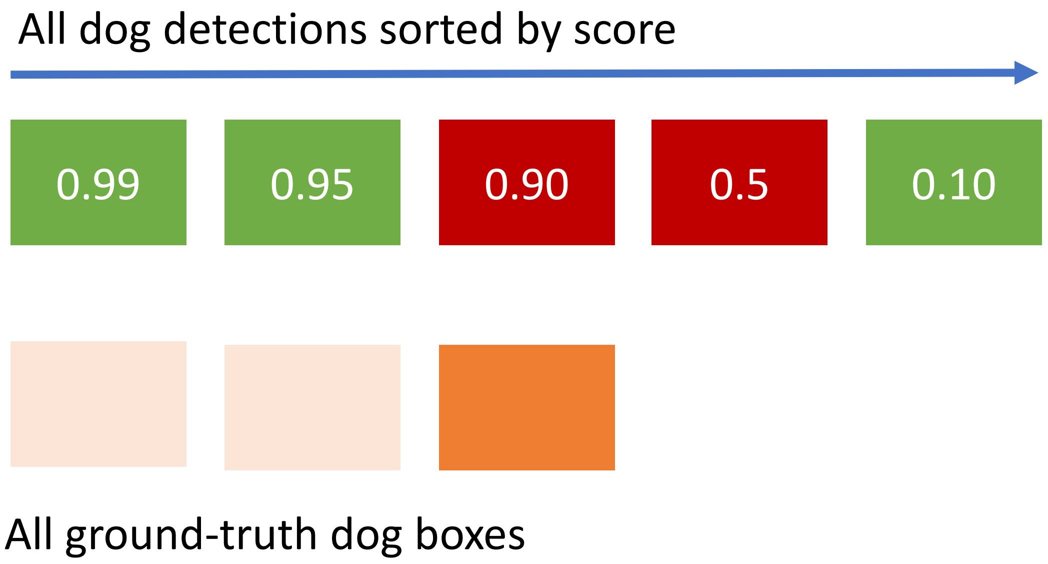
Evaluating Object Detectors: Mean Average Precision (mAP)



1. Run object detector on all test images (with NMS)
2. For each category, 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

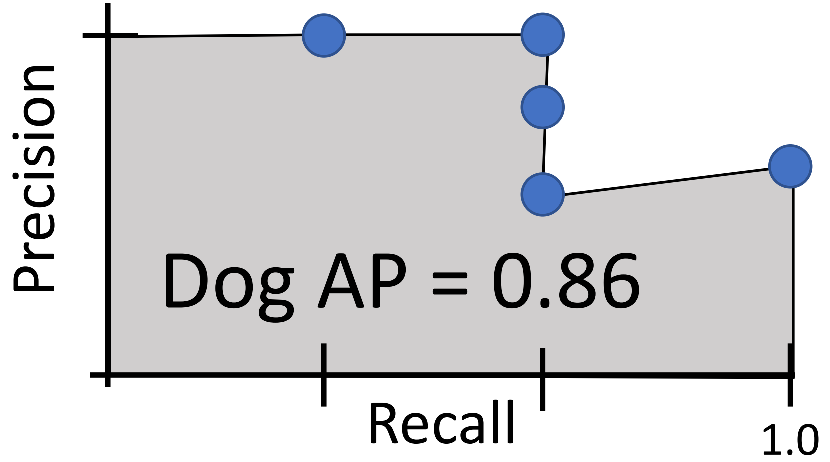


Evaluating Object Detectors: Mean Average Precision (mAP)



1. Run object detector on all test images (with NMS)
2. For each category, 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

How to get AP = 1.0: Hit all GT boxes with IoU > 0.5, and have no "false positive" detections ranked above any "true positives"



Evaluating Object Detectors: Mean Average Precision (mAP)

1. Run object detector on all test images (with NMS)
2. For each category, 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 category

Car AP = 0.65

Cat AP = 0.80

Dog AP = 0.86

mAP@0.5 = 0.77

Evaluating Object Detectors: Mean Average Precision (mAP)

1. Run object detector on all test images (with NMS)
2. For each category, 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 category
4. For "COCO mAP": Compute mAP@thresh for each IoU threshold (0.5, 0.55, 0.6, …, 0.95) and take average

mAP@0.5 = 0.77
mAP@0.55 = 0.71
mAP@0.60 = 0.65
…
mAP@0.95 = 0.2

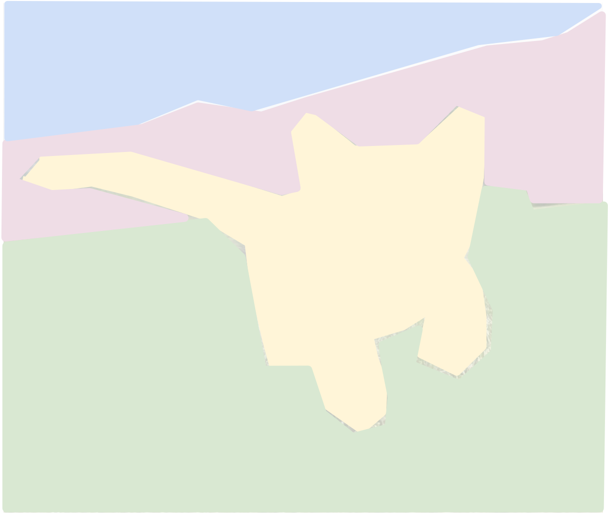
COCO mAP = 0.4

Computer Vision Tasks: Object Detection

Classification

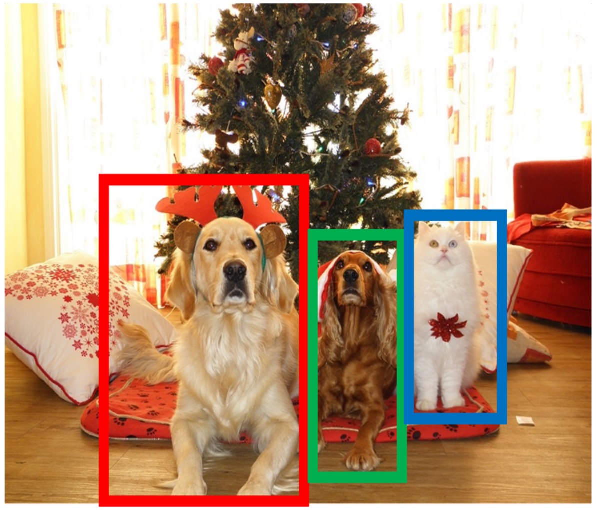
CAT

Semantic Segmentation



GRASS, CAT, TREE, SKY

Object Detection



DOG, DOG, CAT

Instance Segmentation

DOG, DOG, CAT

No spatial extent

No objects, just pixels

Multiple Objects

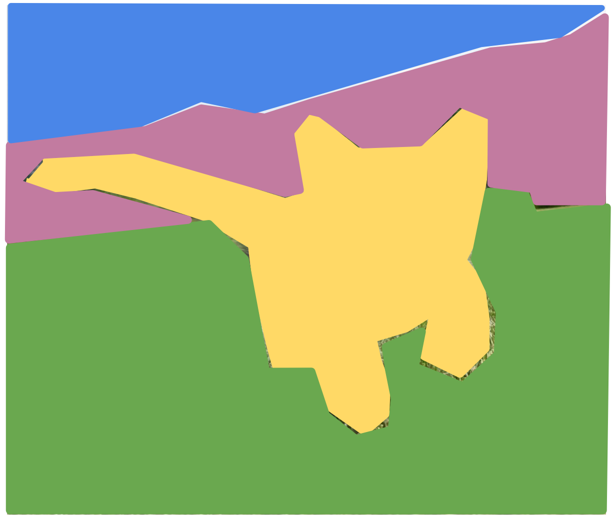
Computer Vision Tasks: Semantic Segmentation

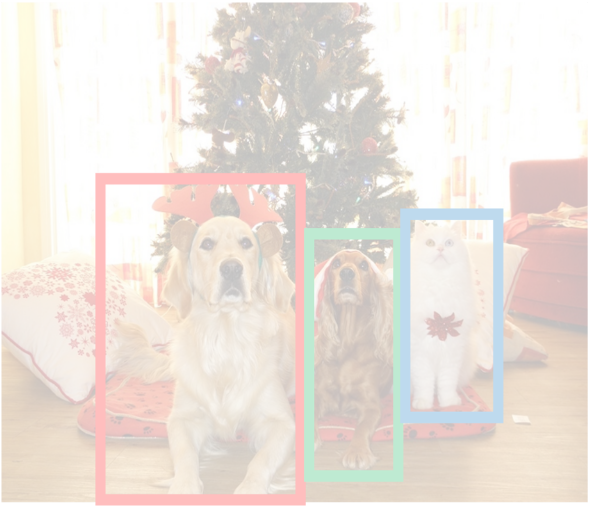
Classification

Semantic Segmentation

Object Detection

Instance Segmentation





CAT

GRASS, CAT, TREE, SKY

DOG, DOG, CAT

DOG, DOG, CAT

No spatial extent

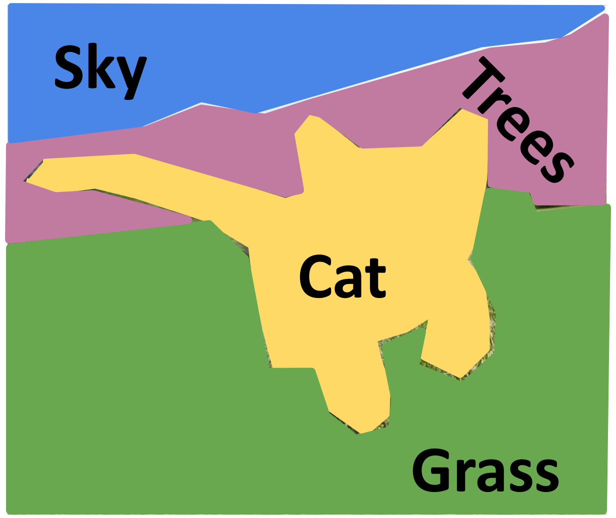
No objects, just pixels

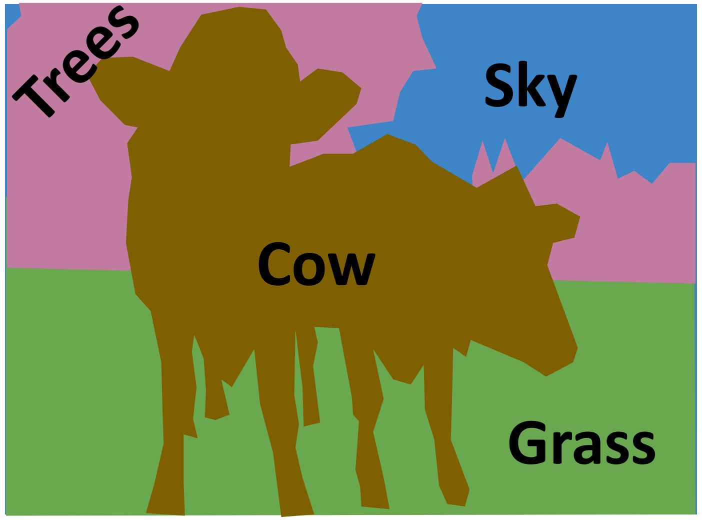
Multiple Objects

Semantic Segmentation

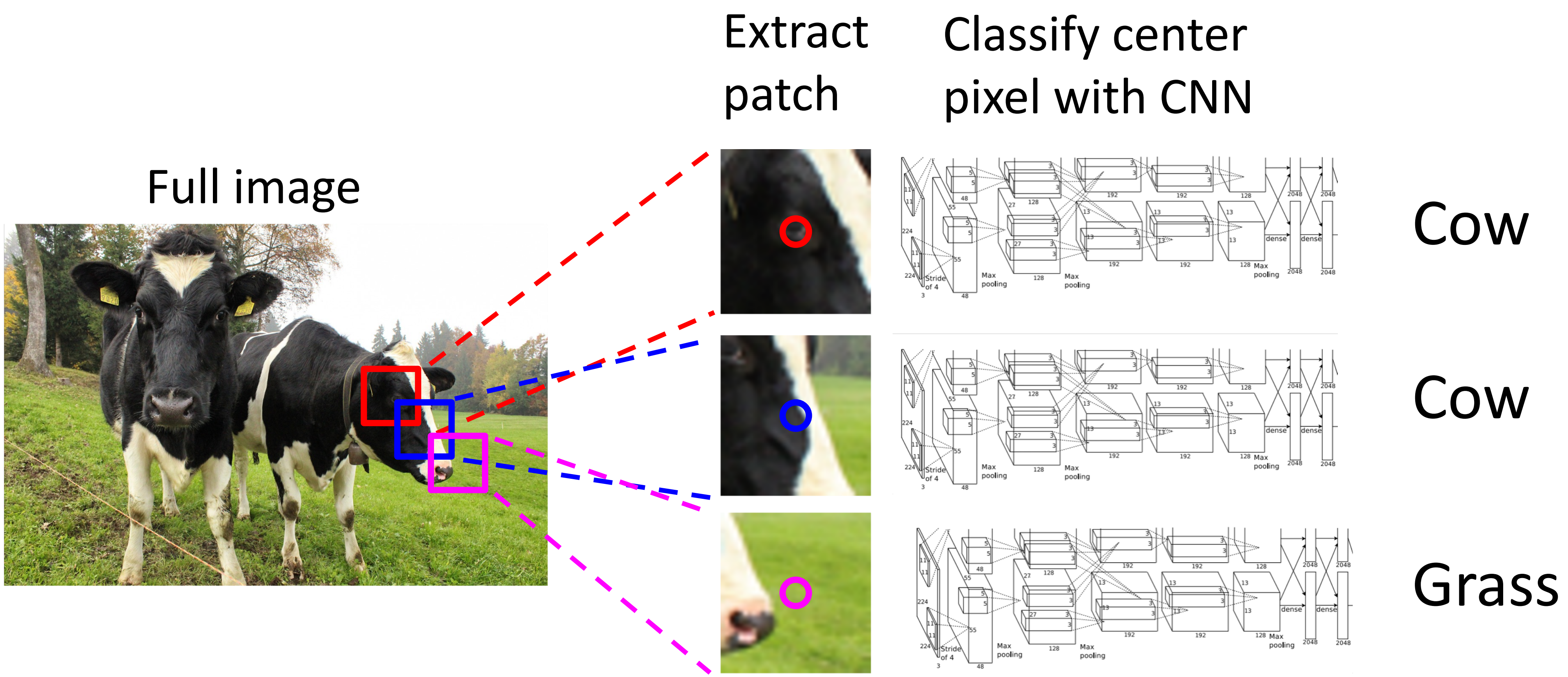
Label each pixel in the image with a category label

Don't differentiate instances, only care about pixels





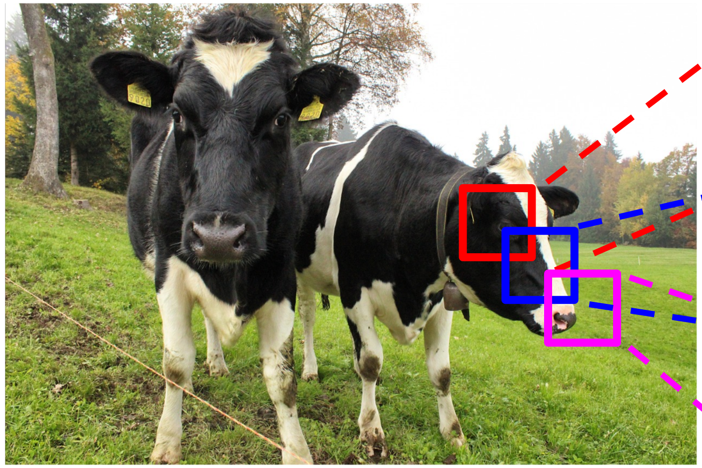
Semantic Segmentation Idea: Sliding Window



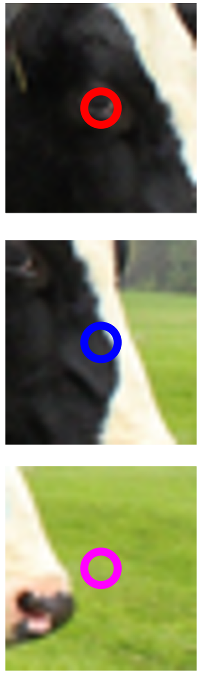
Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

Semantic Segmentation Idea: Sliding Window

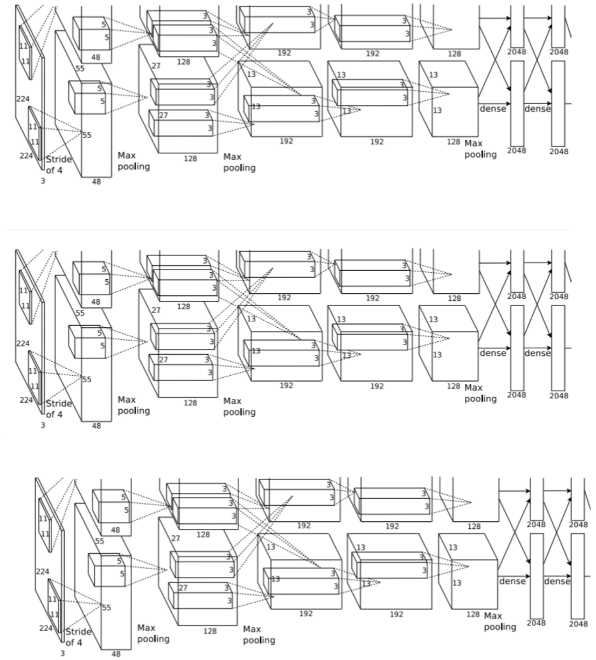
Full image



Extract patch



Classify center pixel with CNN



Cow

Cow

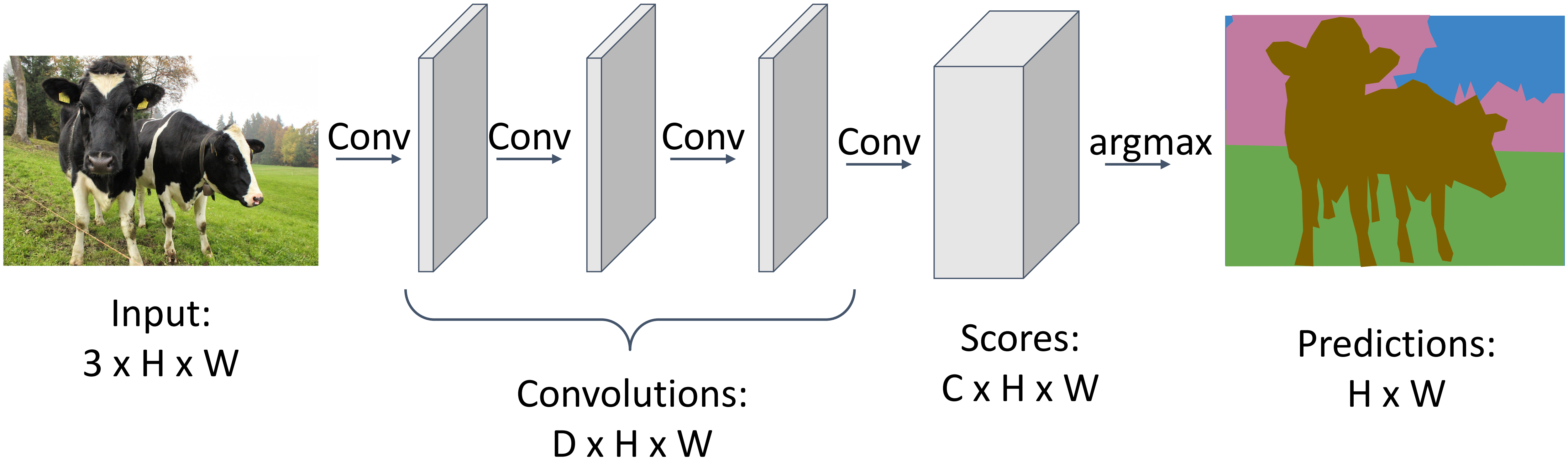
Grass

Problem: Very inefficient! Not reusing shared features between overlapping patches

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

Semantic Segmentation: Fully Convolutional Network

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!



Loss function: Per-Pixel cross-entropy

Long et al, “Fully convolutional networks for semantic segmentation”, CVPR 2015

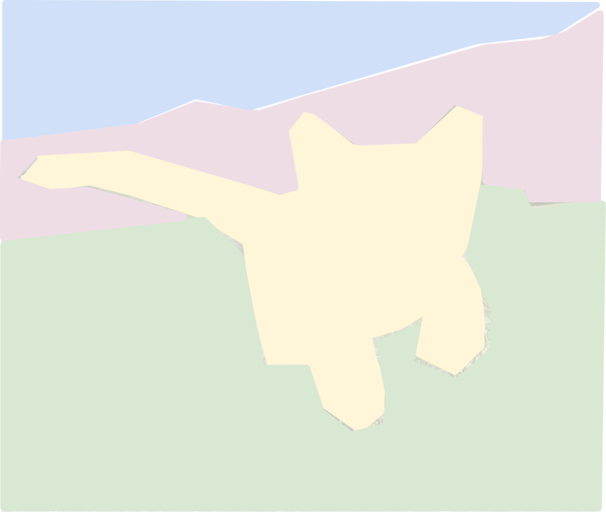
Summary: Beyond Image Classification

Classification

CAT

No spatial extent

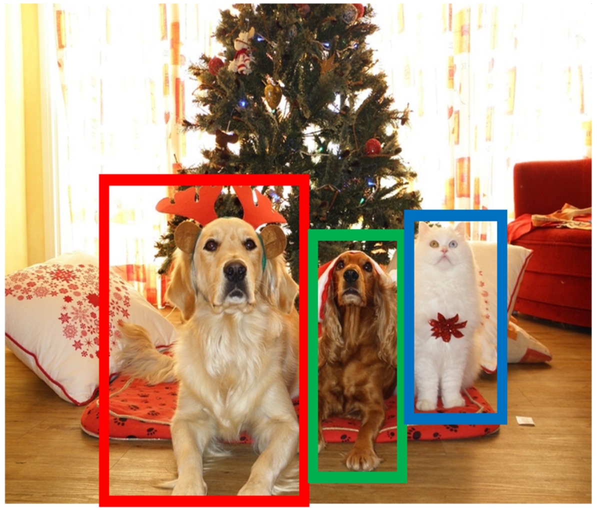
Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

Object Detection



DOG, DOG, CAT

Instance Segmentation

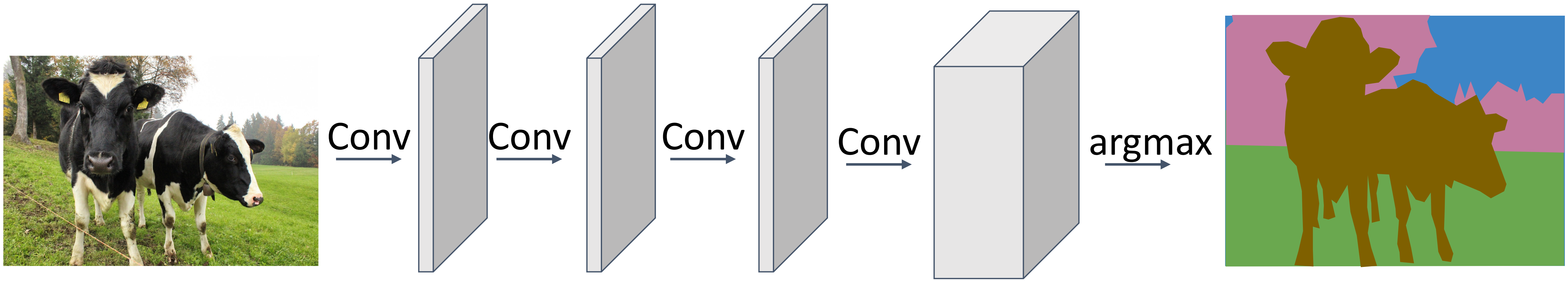
DOG, DOG, CAT

Multiple Objects

This image is CC0 public domain

Semantic Segmentation: Fully Convolutional Network

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!



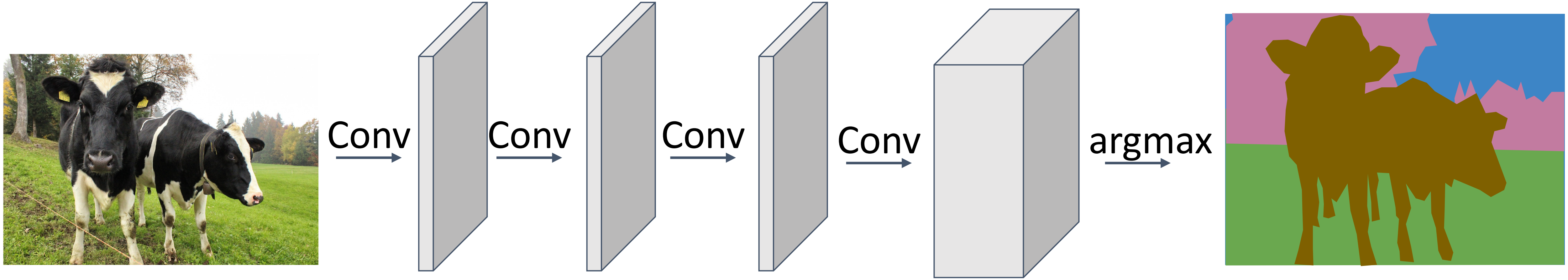
Input:
3 x H x W

Problem #1: Effective receptive field size is linear in number of conv layers: With L 3x3 conv layers, receptive field is 1+2L

Long et al, "Fully convolutional networks for semantic segmentation", CVPR 2015

Semantic Segmentation: Fully Convolutional Network

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!



Input:
3 x H x W

Problem #1: Effective receptive field size is linear in number of conv layers: With L 3x3 conv layers, receptive field is 1+2L

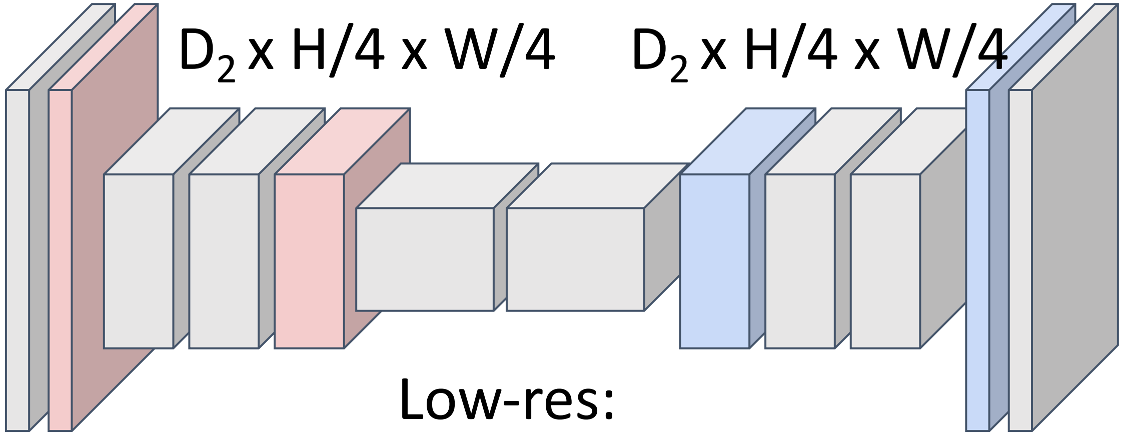
Problem #2: Convolution on high res images is expensive! Recall ResNet stem aggressively downsamples

Long et al, “Fully convolutional networks for semantic segmentation”, CVPR 2015

Semantic Segmentation: Fully Convolutional Network

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

Input:
3 x H x W



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

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

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

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

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

Predictions:
H x W

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

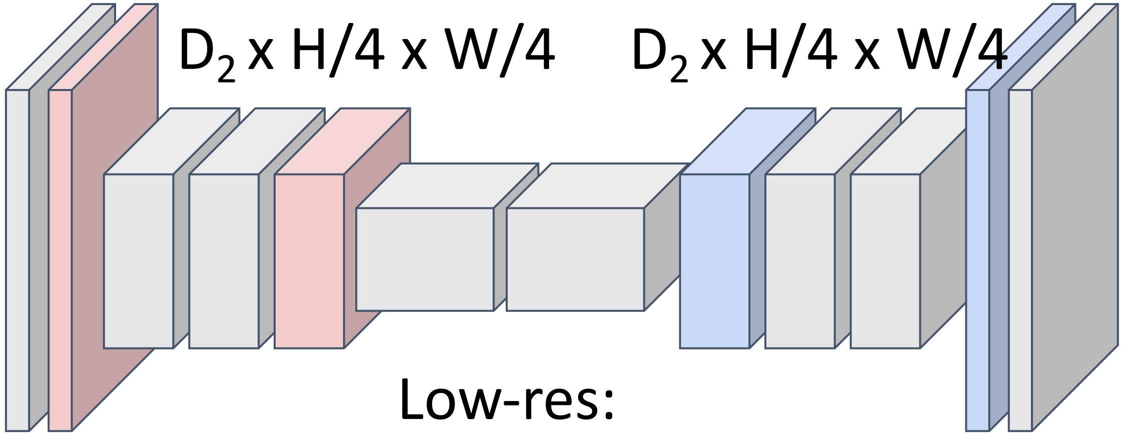
Semantic Segmentation: Fully Convolutional Network

Downsampling: Pooling, strided convolution

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

Upsampling: ???

Input: 3 x H x W



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

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

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

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

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

Predictions: H x W

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

In-Network Upsampling: “Unpooling”

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 x 2 x 2

Output
C x 4 x 4

In-Network Upsampling: “Unpooling”

Bed of Nails

1	2
3	4

Input
C x 2 x 2

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

Output
C x 4 x 4

Nearest Neighbor

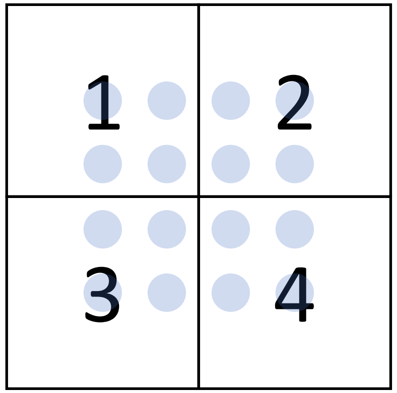
1	2
3	4

Input
C x 2 x 2

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

Output
C x 4 x 4

In-Network Upsampling: Bilinear Interpolation



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

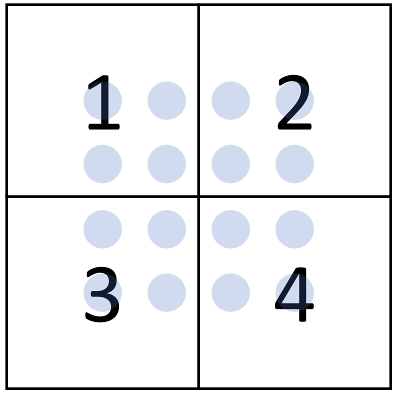
Output: C x 4 x 4

$$f_{x,y} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - i|) \max(0, 1 - |y - j|)$$

$$i \in \{\lfloor x \rfloor - 1, \dots, \lceil x \rceil + 1\}$$
$$j \in \{\lfloor y \rfloor - 1, \dots, \lceil y \rceil + 1\}$$

Use two closest neighbors in x and y
to construct linear approximations

In-Network Upsampling: Bicubic Interpolation



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

Use **three** closest neighbors in x and y to construct **cubic** approximations
(This is how we normally resize images!)

In-Network Upsampling: “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

→

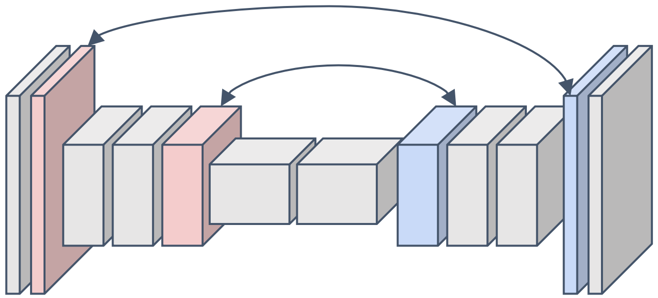
5	6
7	8

→ Rest of net →

1	2
3	4

→

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

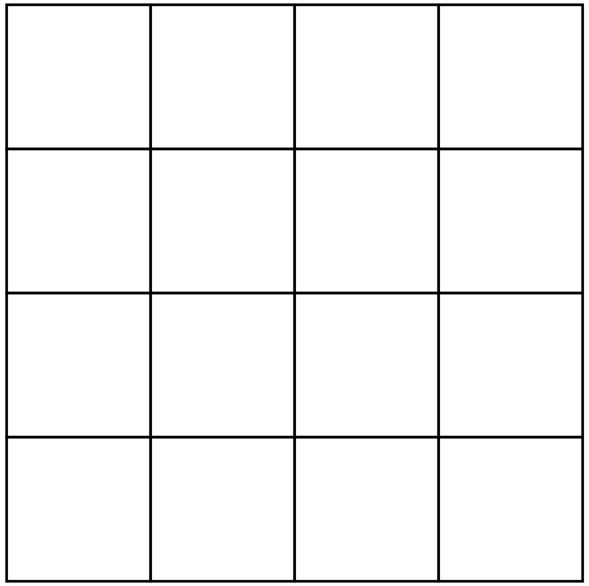


Pair each downsampling layer with an upsampling layer

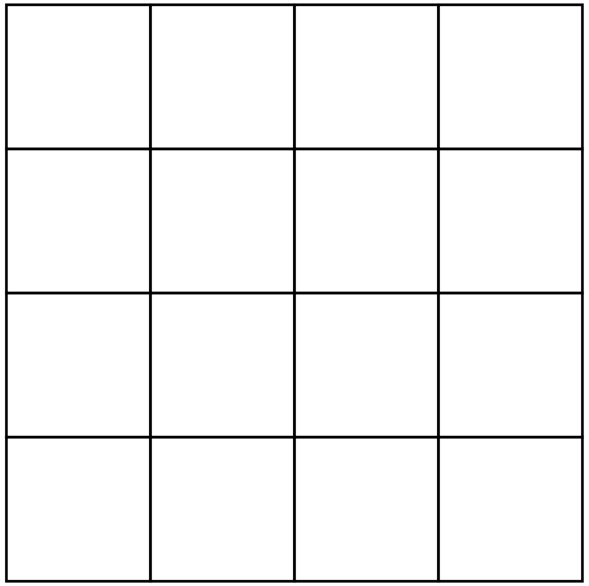
Noh et al, “Learning Deconvolution Network for Semantic Segmentation”, ICCV 2015

Learnable Upsampling: Transposed Convolution

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



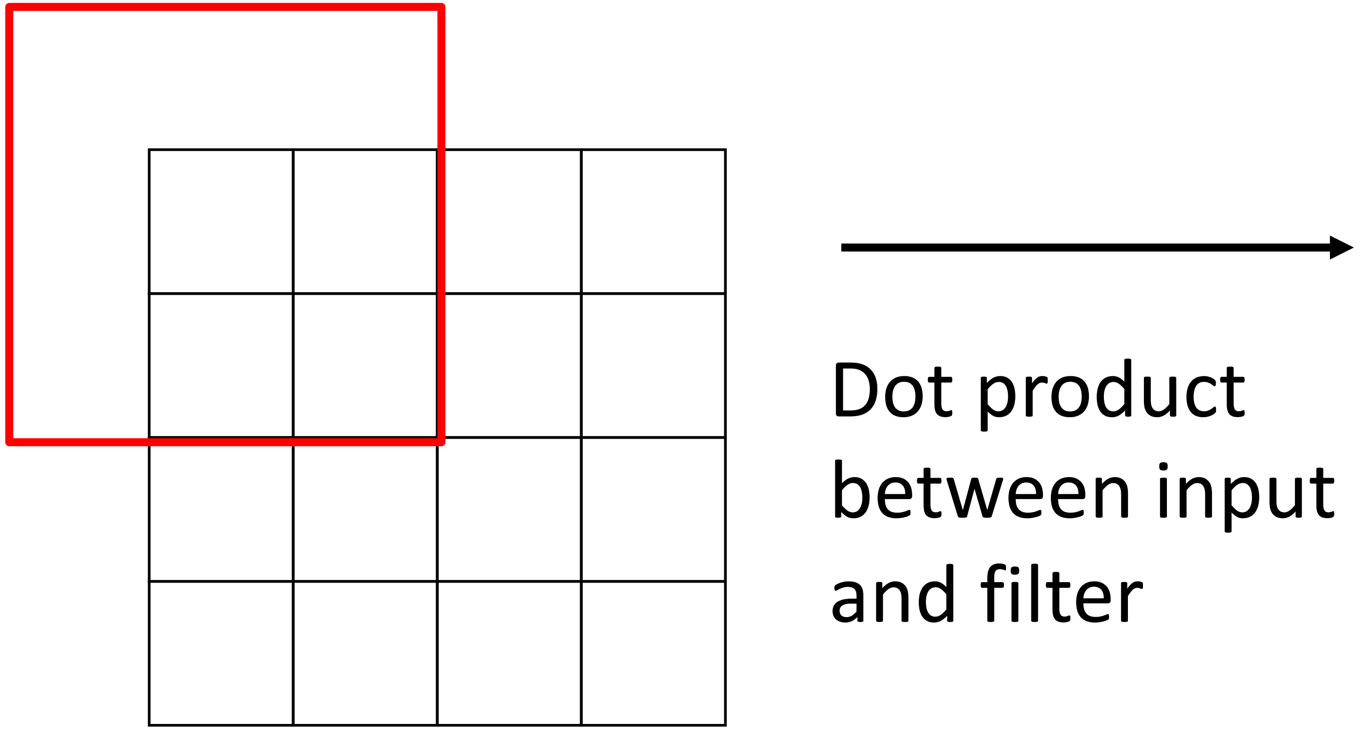
Input: 4 x 4



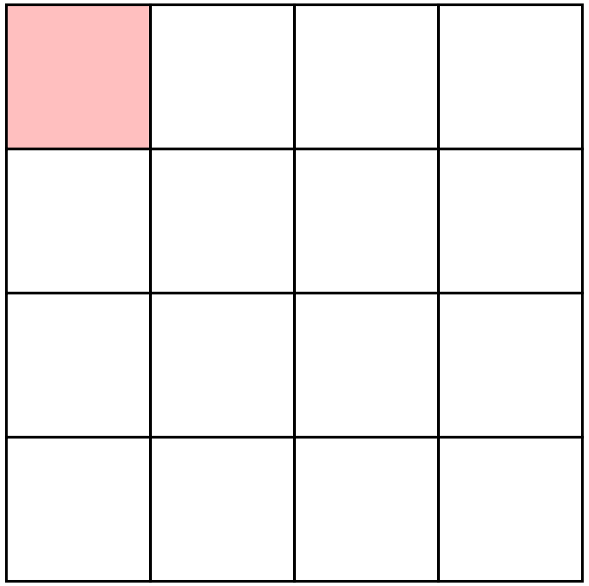
Output: 4 x 4

Learnable Upsampling: Transposed Convolution

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



Dot product between input and filter

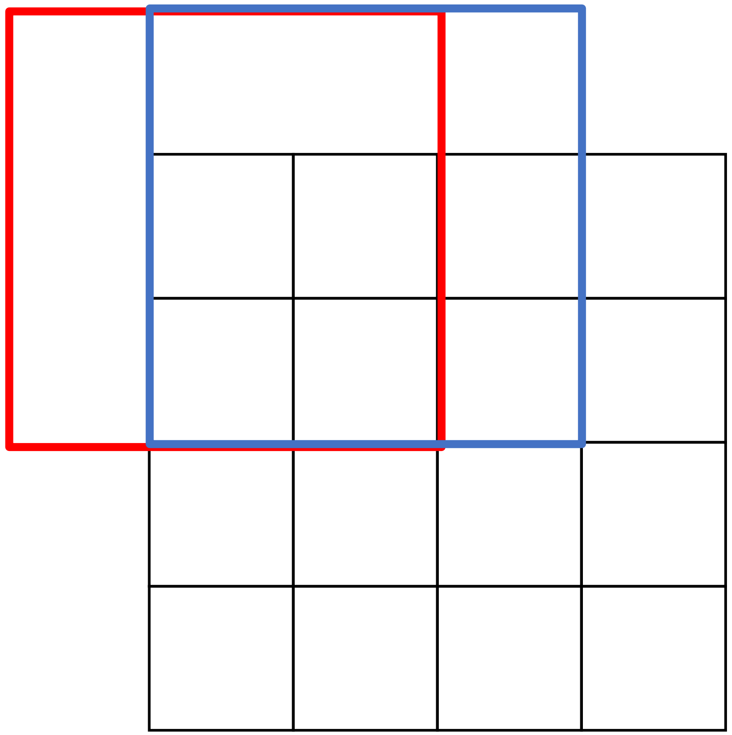


Input: 4 x 4

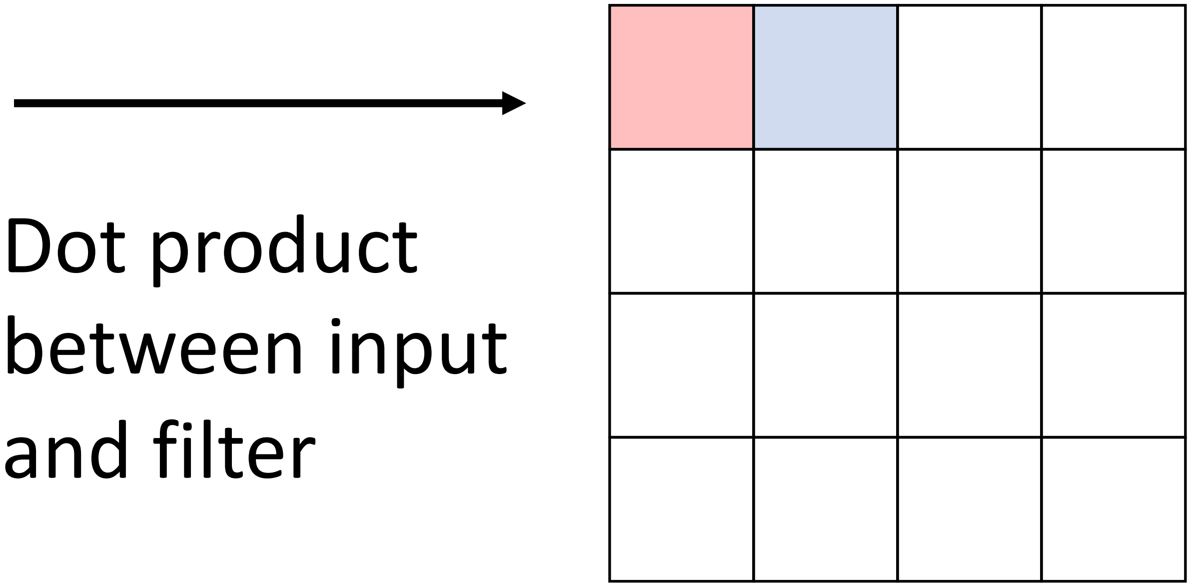
Output: 4 x 4

Learnable Upsampling: Transposed Convolution

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



Dot product between input and filter



Input: 4 x 4

Output: 4 x 4

Learnable Upsampling: Transposed Convolution

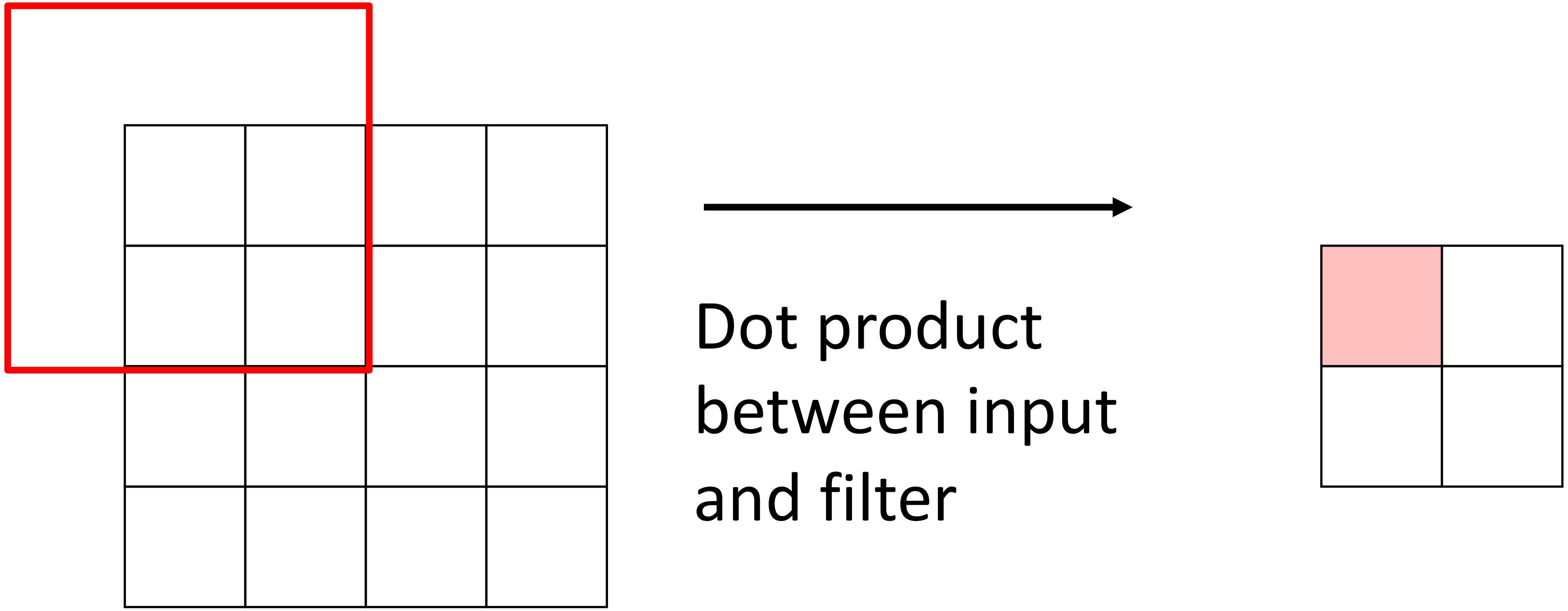
Recall: Normal 3 x 3 convolution, stride 2, pad 1

Input: 4 x 4

Output: 2 x 2

Learnable Upsampling: Transposed Convolution

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

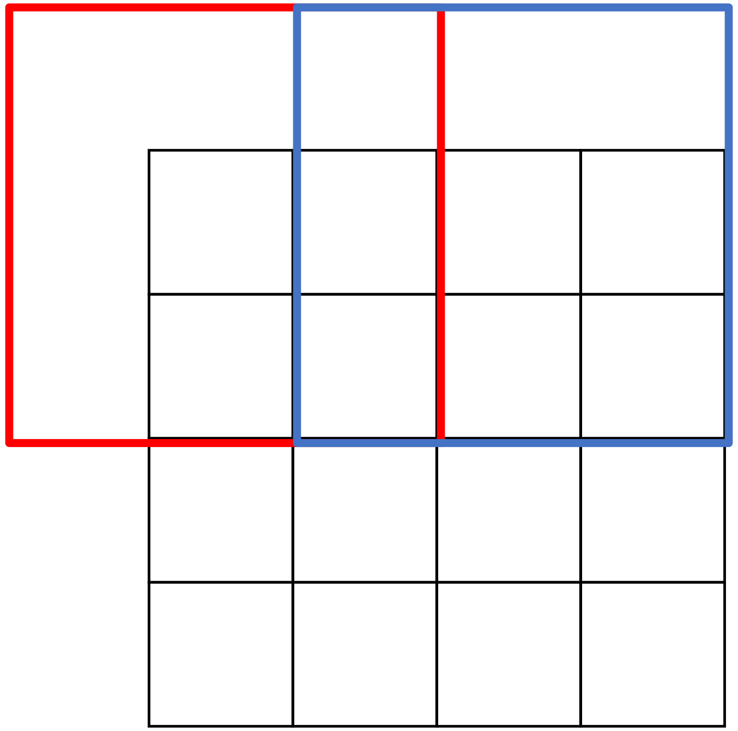


Input: 4 x 4

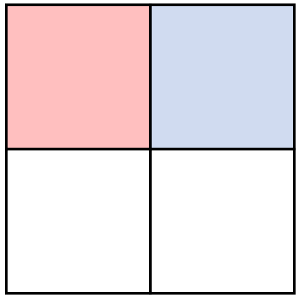
Output: 2 x 2

Learnable Upsampling: Transposed Convolution

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



Dot product between input and filter

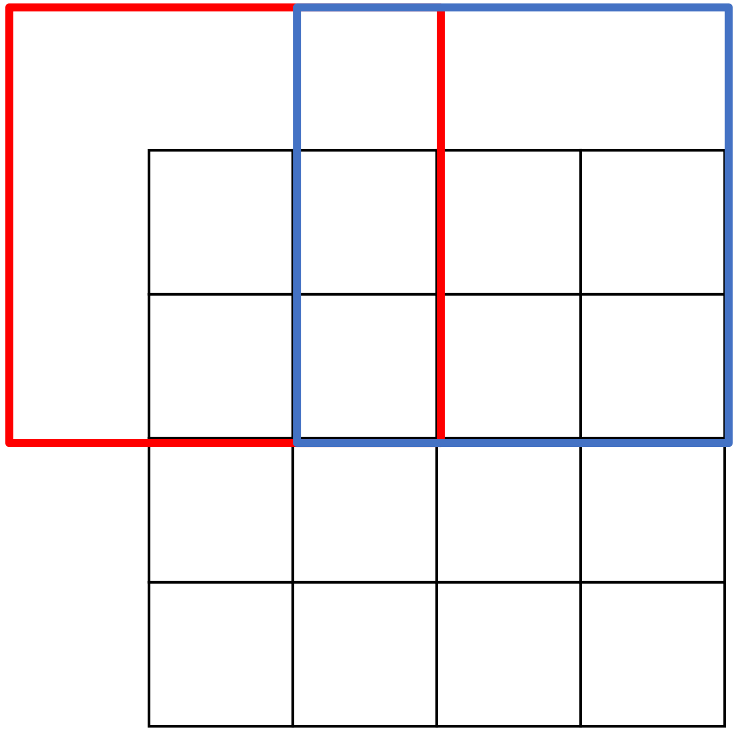


Input: 4 x 4

Output: 2 x 2

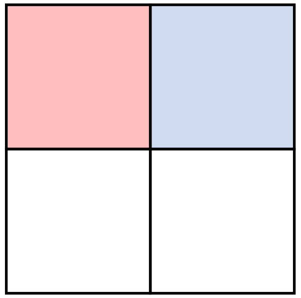
Learnable Upsampling: Transposed Convolution

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



Convolution with stride > 1 is "Learnable Downsampling"
Can we use stride < 1 for "Learnable Upsampling"?

Dot product between input and filter

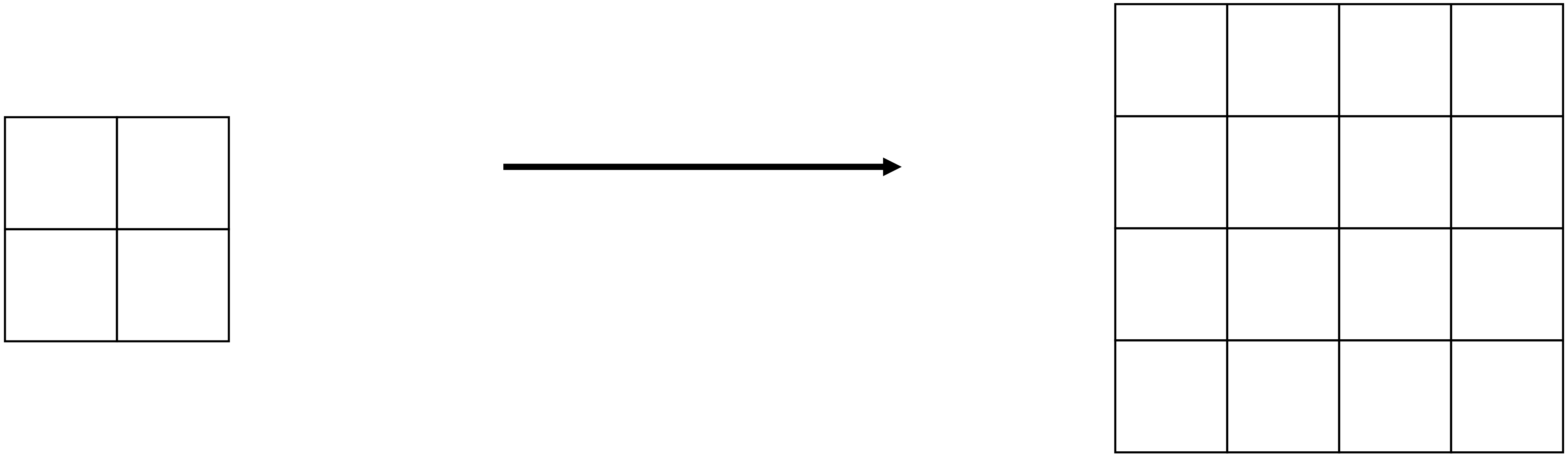


Input: 4 x 4

Output: 2 x 2

Learnable Upsampling: Transposed Convolution

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

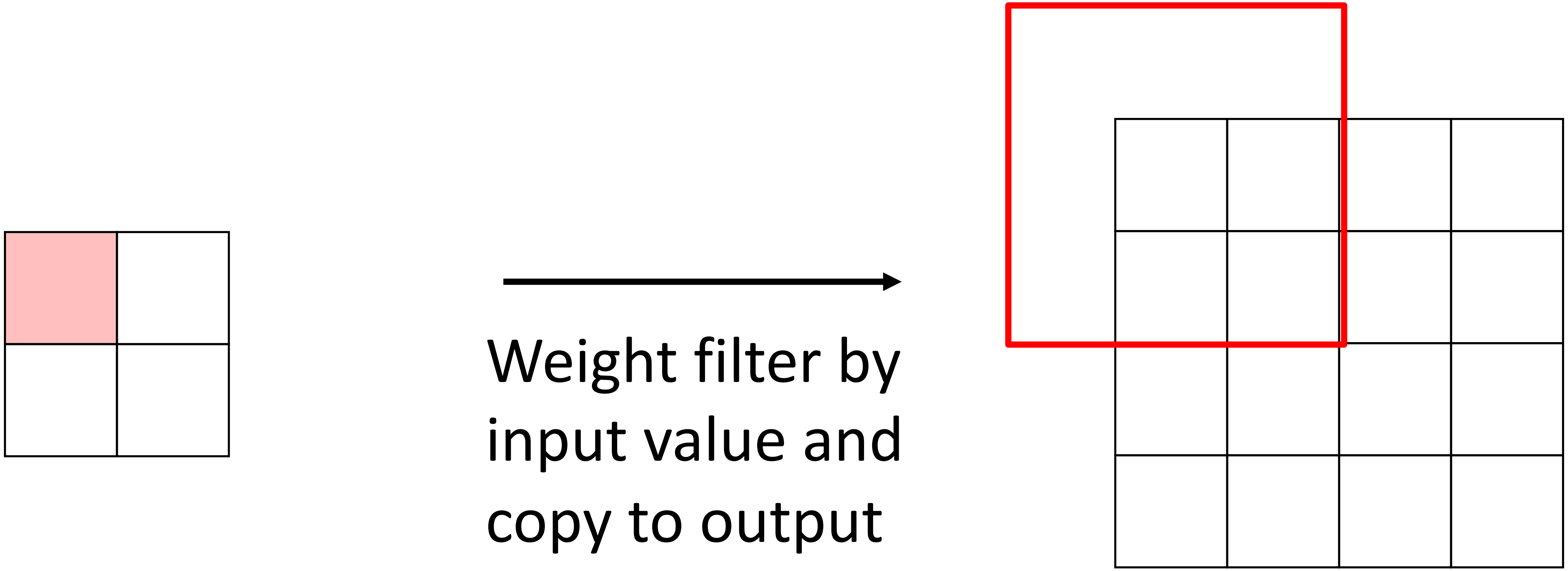


Input: 2 x 2

Output: 4 x 4

Learnable Upsampling: Transposed Convolution

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



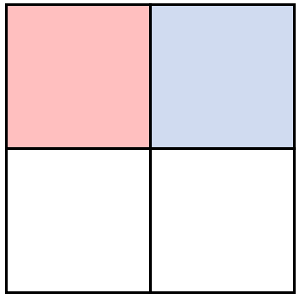
Input: 2 x 2

Output: 4 x 4

Learnable Upsampling: Transposed Convolution

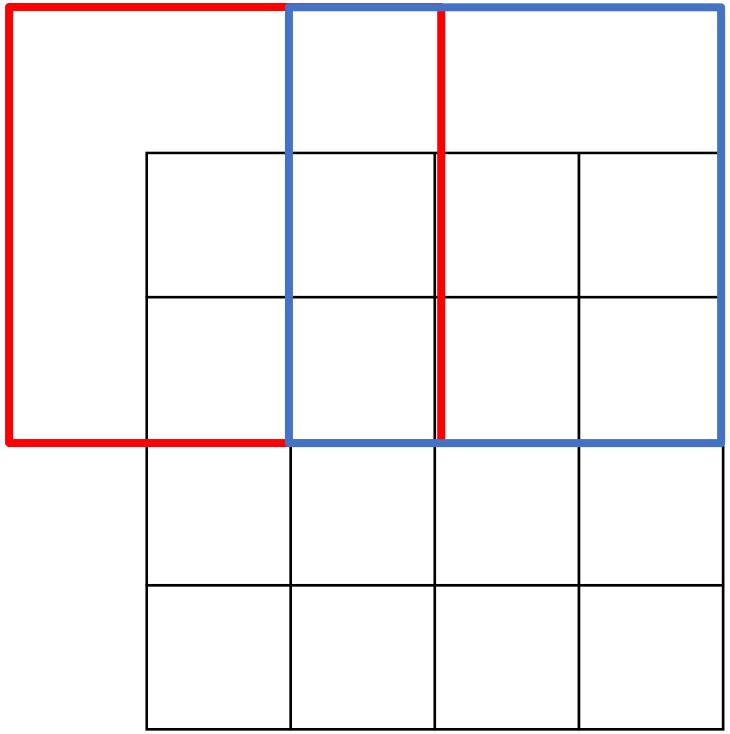
3 x 3 **convolution transpose**, stride 2

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



Input: 2 x 2

Weight filter by
input value and
copy to output

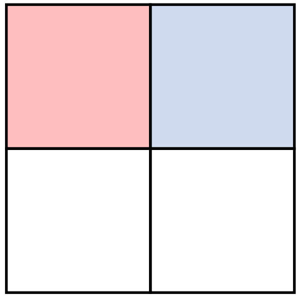


Output: 4 x 4

Learnable Upsampling: Transposed Convolution

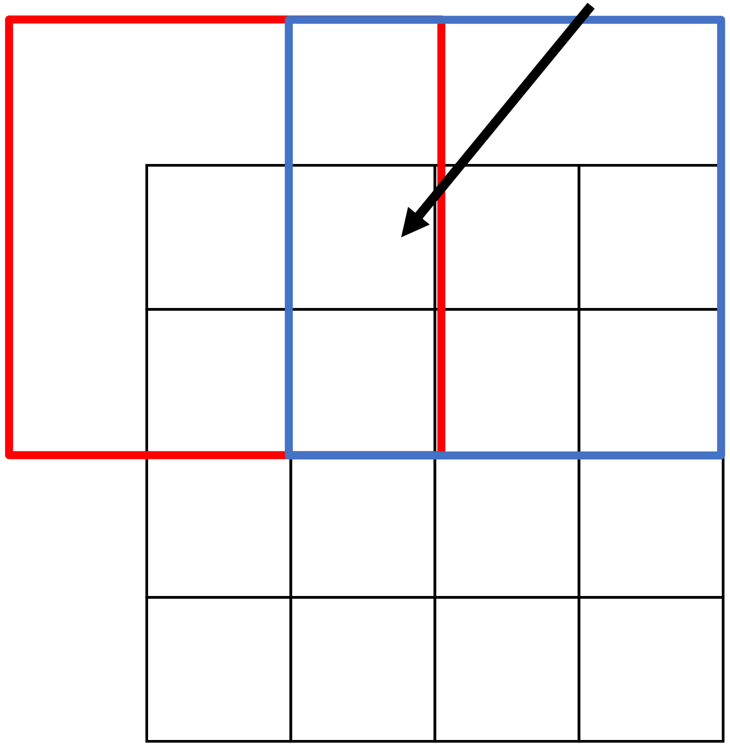
3 x 3 **convolution transpose**, stride 2

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



Input: 2 x 2

Weight filter by input value and copy to output



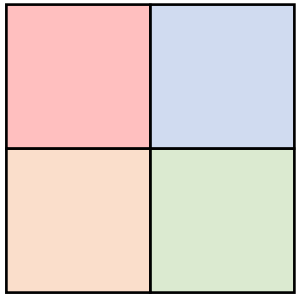
Output: 4 x 4

Learnable Upsampling: Transposed Convolution

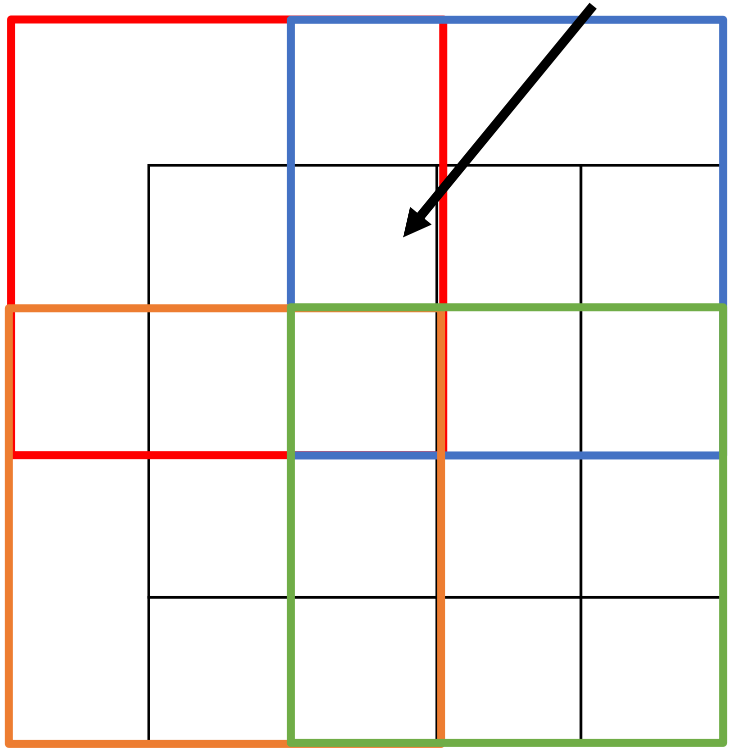
3 x 3 **convolution transpose**, stride 2

Sum where output overlaps

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



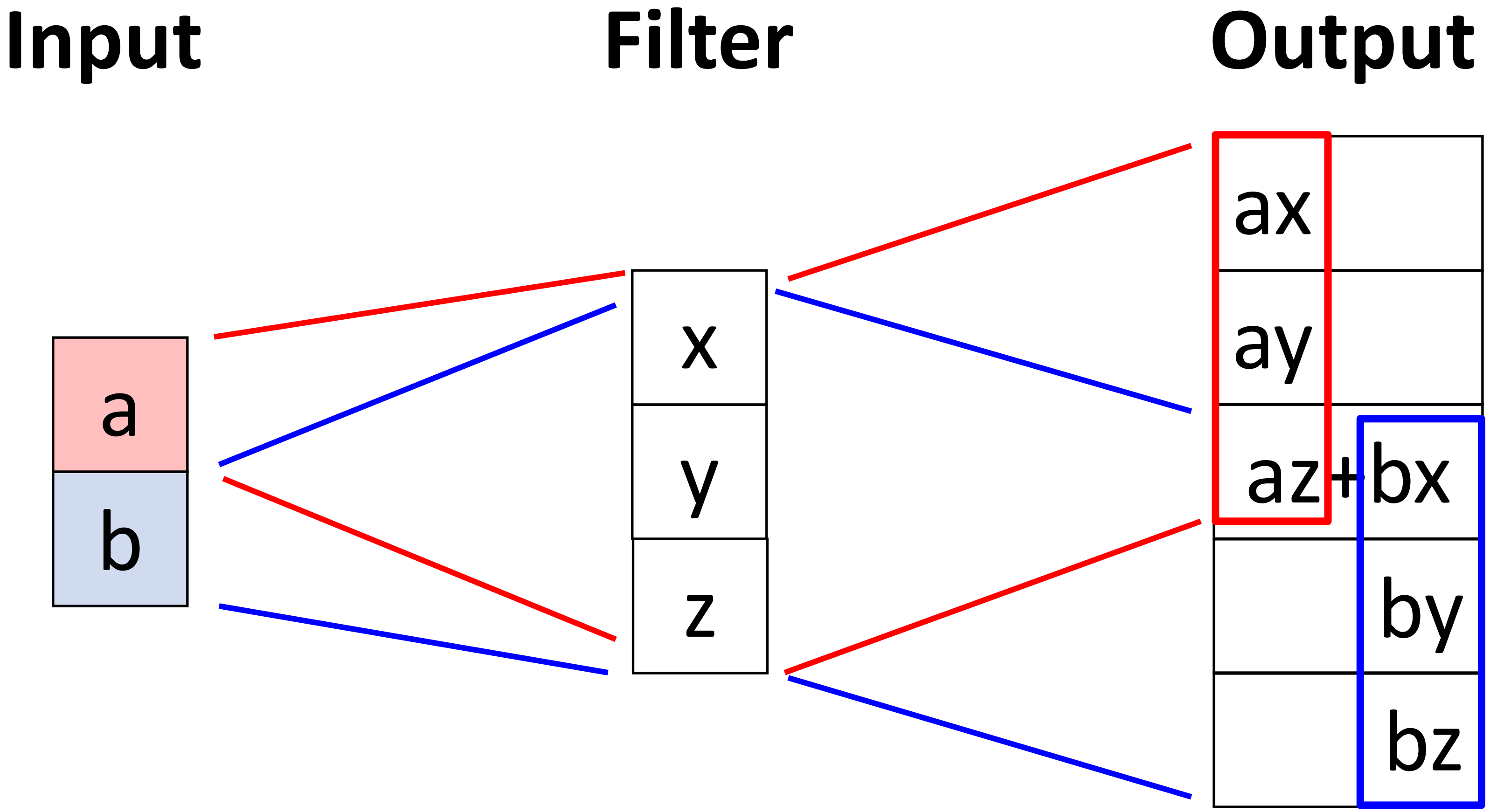
Weight filter by input value and copy to output



Input: 2 x 2

Output: 4 x 4

Transposed Convolution: 1D example

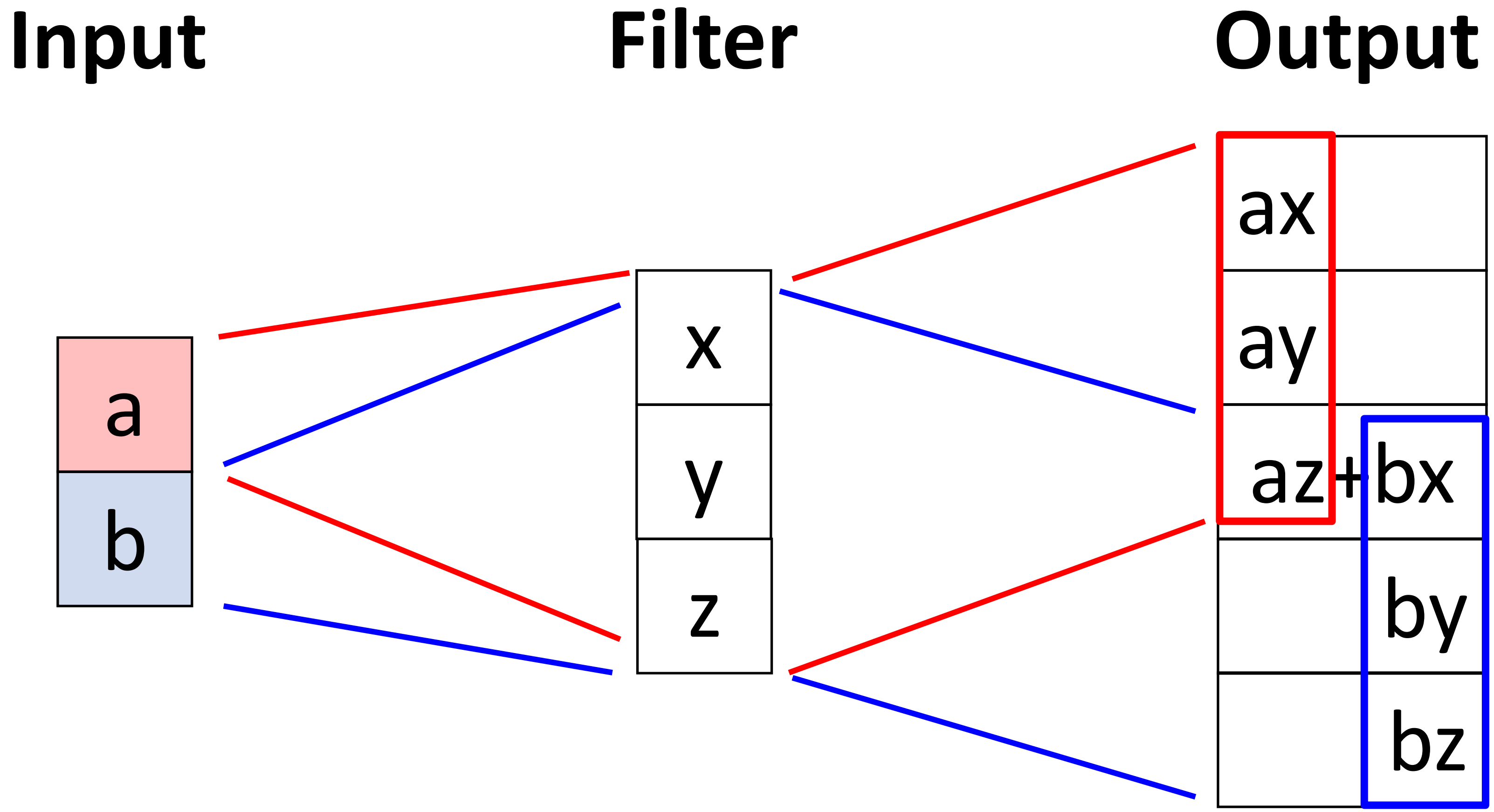


Output has copies of filter weighted by input

Stride 2: Move 2 pixels output for each pixel in input

Sum at overlaps

Transposed Convolution: 1D example



This has many names:

- Deconvolution (bad)!
- Upconvolution
- Fractionally strided convolution
- Backward strided convolution
- <u>Transposed Convolution</u> (best name)

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

Transposed convolution multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix}$$

When stride=1, transposed conv is just a regular conv (with different padding rules)

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, <u>stride=2</u>, padding=1

Transposed convolution multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, <u>stride=2</u>, padding=1

Transposed convolution multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

When stride>1, transposed convolution cannot be expressed as normal conv

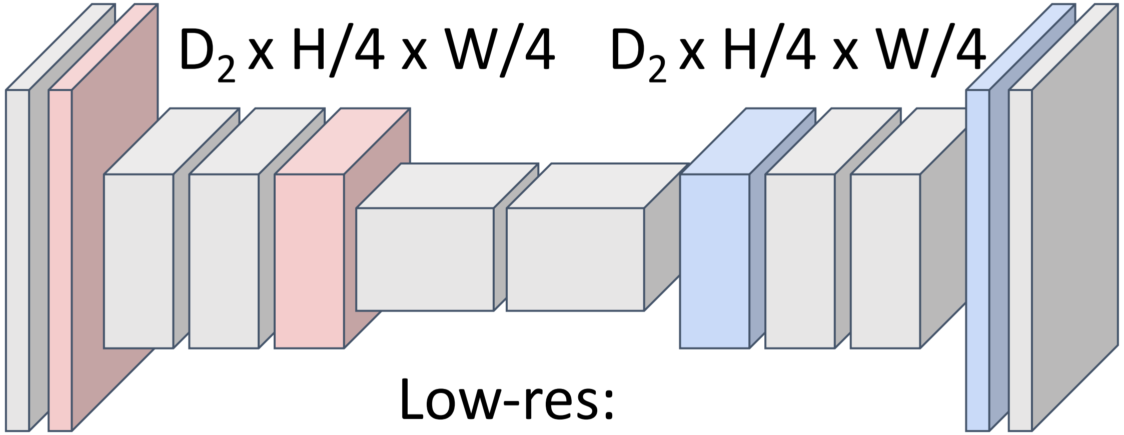
Semantic Segmentation: Fully Convolutional Network

Downsampling: Pooling, strided convolution

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

Upsampling: linterpolation, transposed conv

Input: $3 \times H \times W$



Med-res: $D_2 \times H/4 \times W/4$

Med-res: $D_2 \times H/4 \times W/4$

Low-res: $D_3 \times H/4 \times W/4$

High-res: $D_1 \times H/2 \times W/2$

High-res: $D_1 \times H/2 \times W/2$

Predictions: $H \times W$

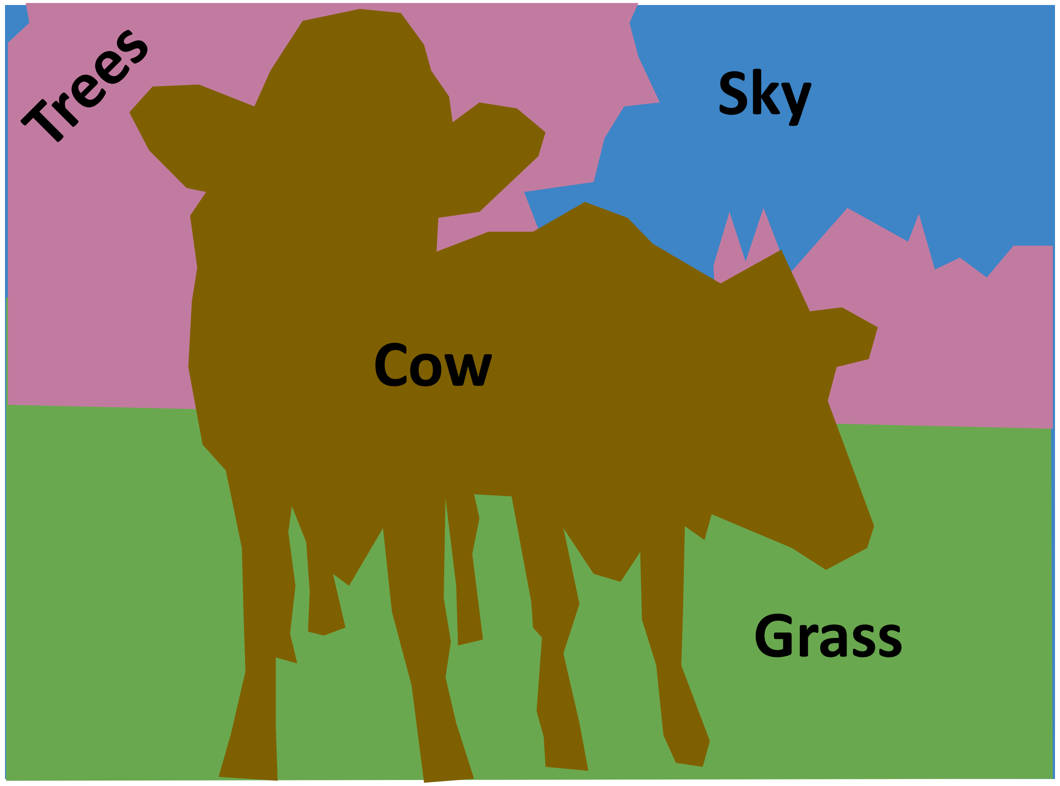
Loss function: Per-Pixel cross-entropy

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

Computer Vision Tasks

Object Detection: Detects individual object instances, but only gives box

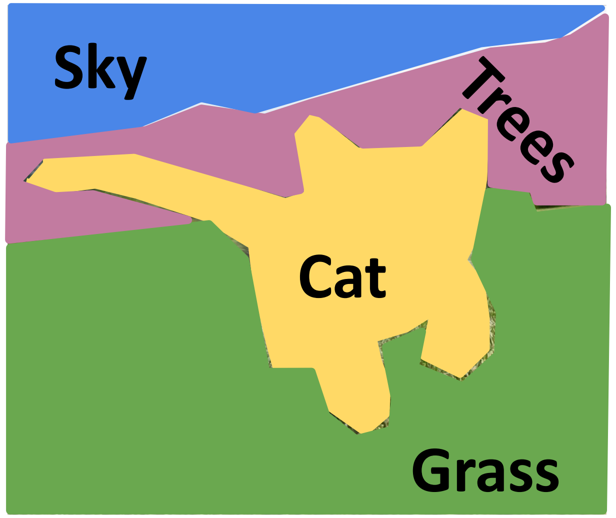
Semantic Segmentation: Gives per-pixel labels, but merges instances

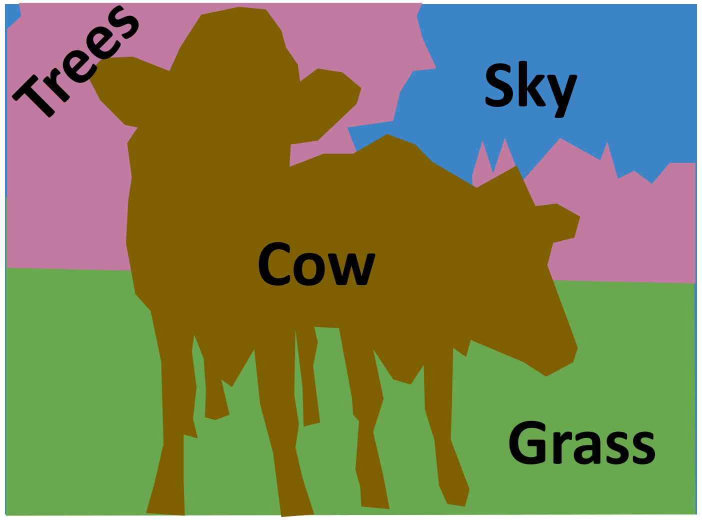


Things and Stuff

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)

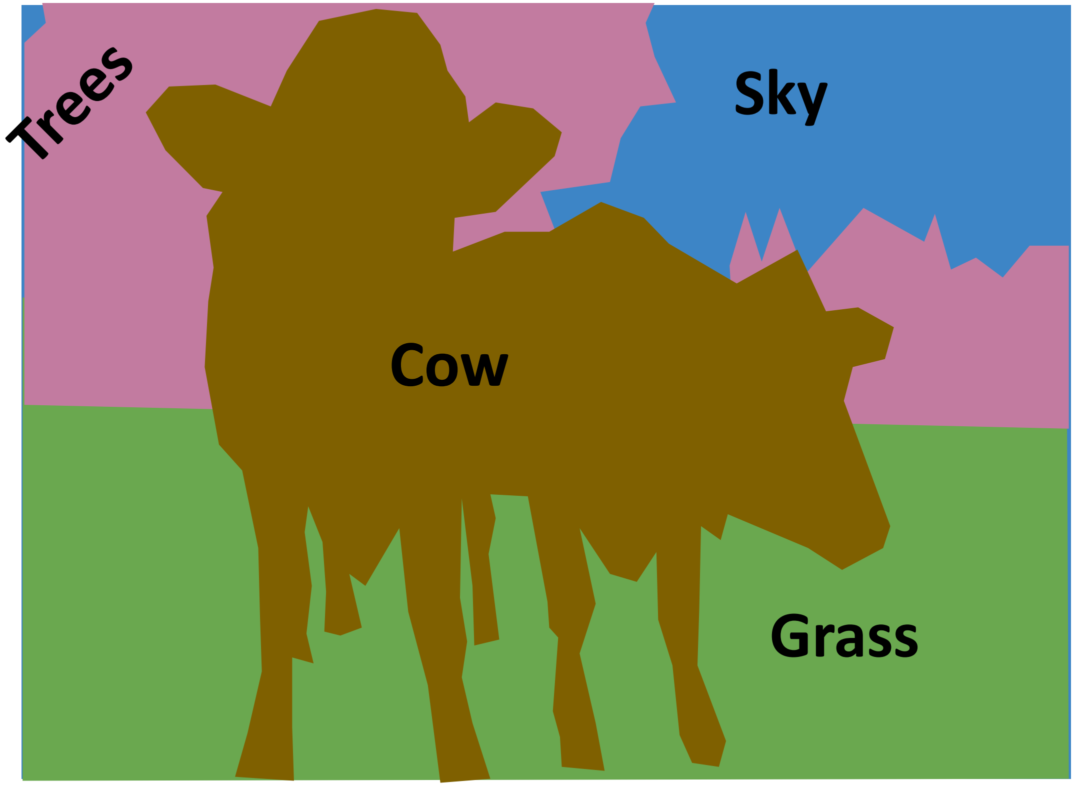




Computer Vision Tasks

Object Detection: Detects individual object instances, but only gives box (Only things!)

Semantic Segmentation: Gives per-pixel labels, but merges instances (Both things and stuff)



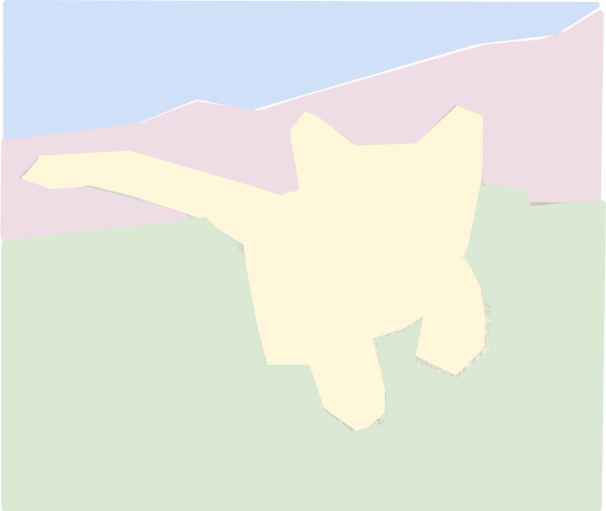
Computer Vision Tasks: Instance Segmentation

Classification

CAT

No spatial extent

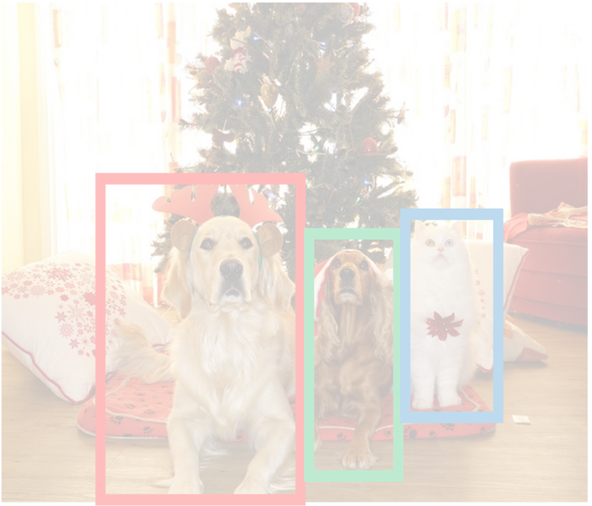
Semantic Segmentation



GRASS, CAT, TREE, SKY

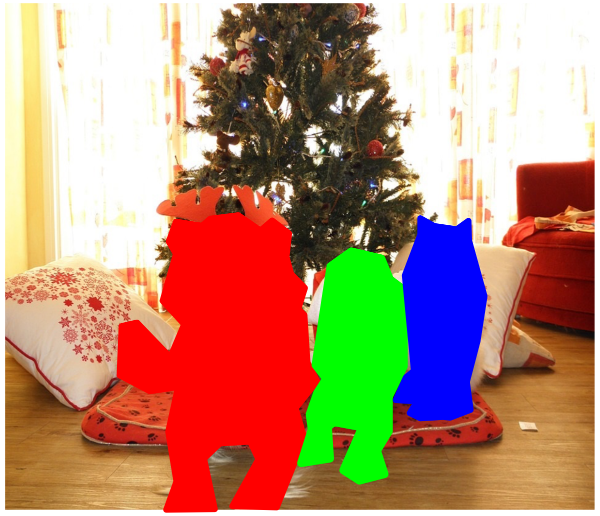
No objects, just pixels

Object Detection



DOG, DOG, CAT

Instance Segmentation

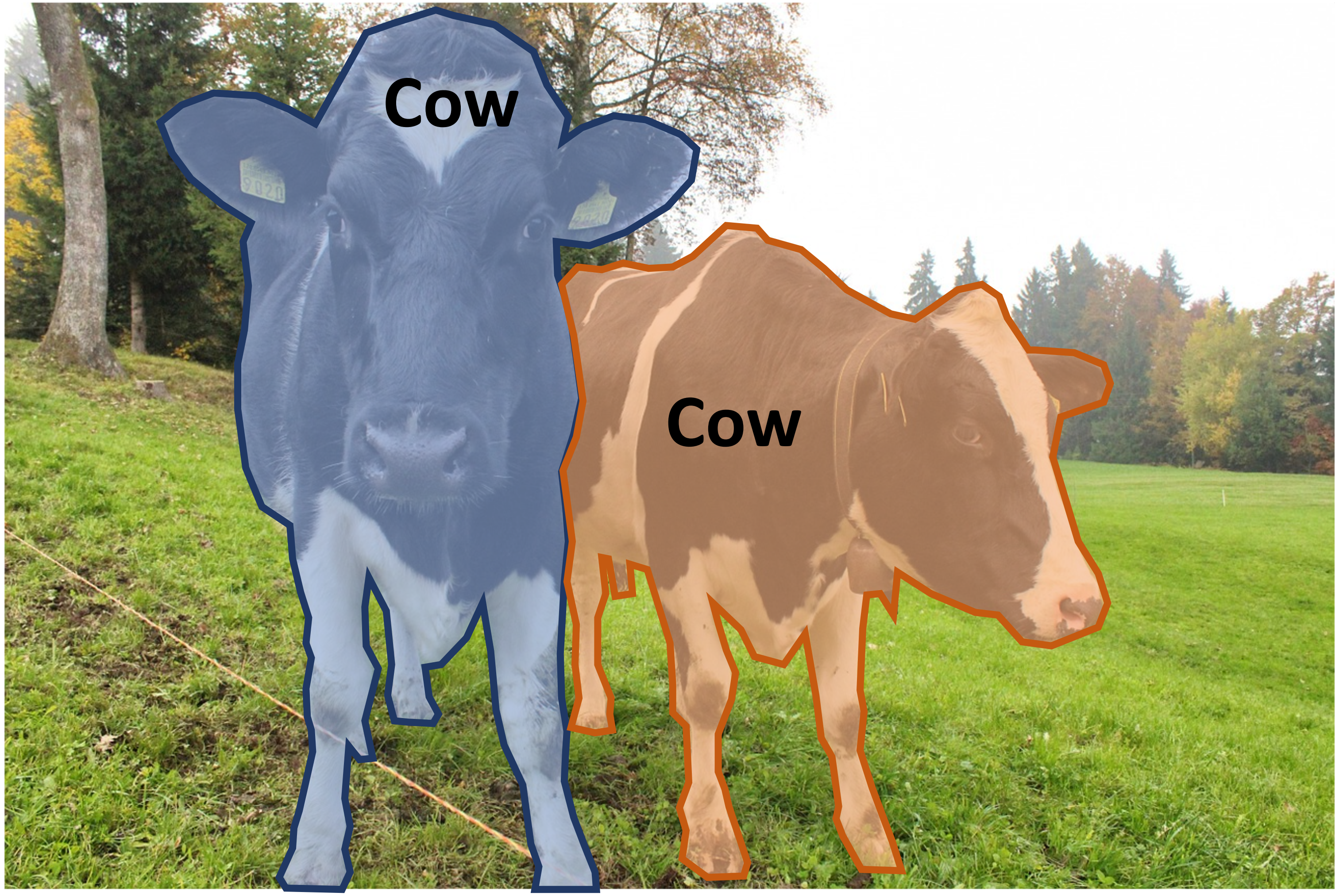


DOG, DOG, CAT

Multiple Objects

Computer Vision Tasks: Instance Segmentation

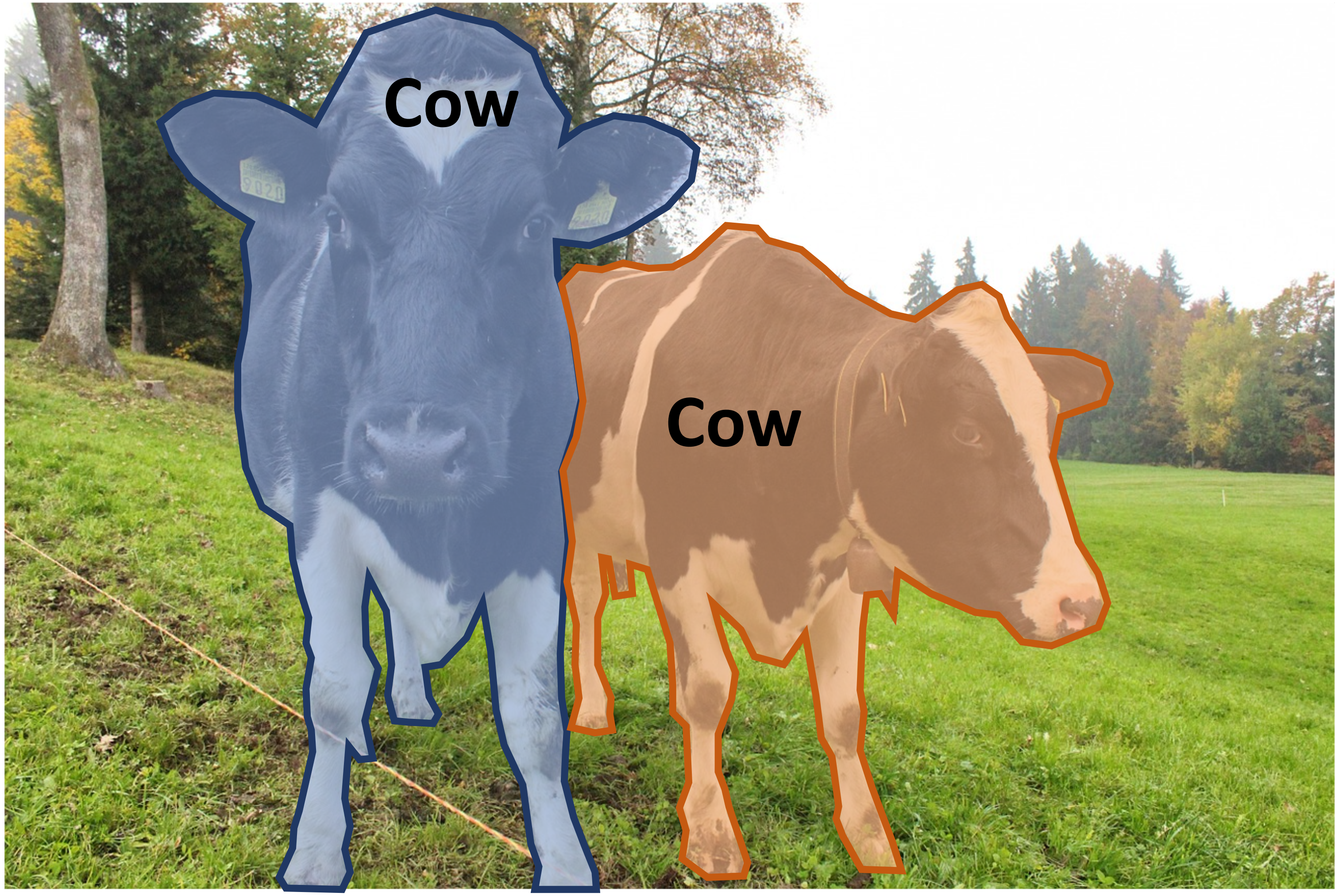
Instance Segmentation: Detect all objects in the image, and identify the pixels that belong to each object (Only things!)



Computer Vision Tasks: Instance Segmentation

Instance Segmentation: Detect all objects in the image, and identify the pixels that belong to each object (Only things!)

Approach: Perform object detection, then predict a segmentation mask for each object!



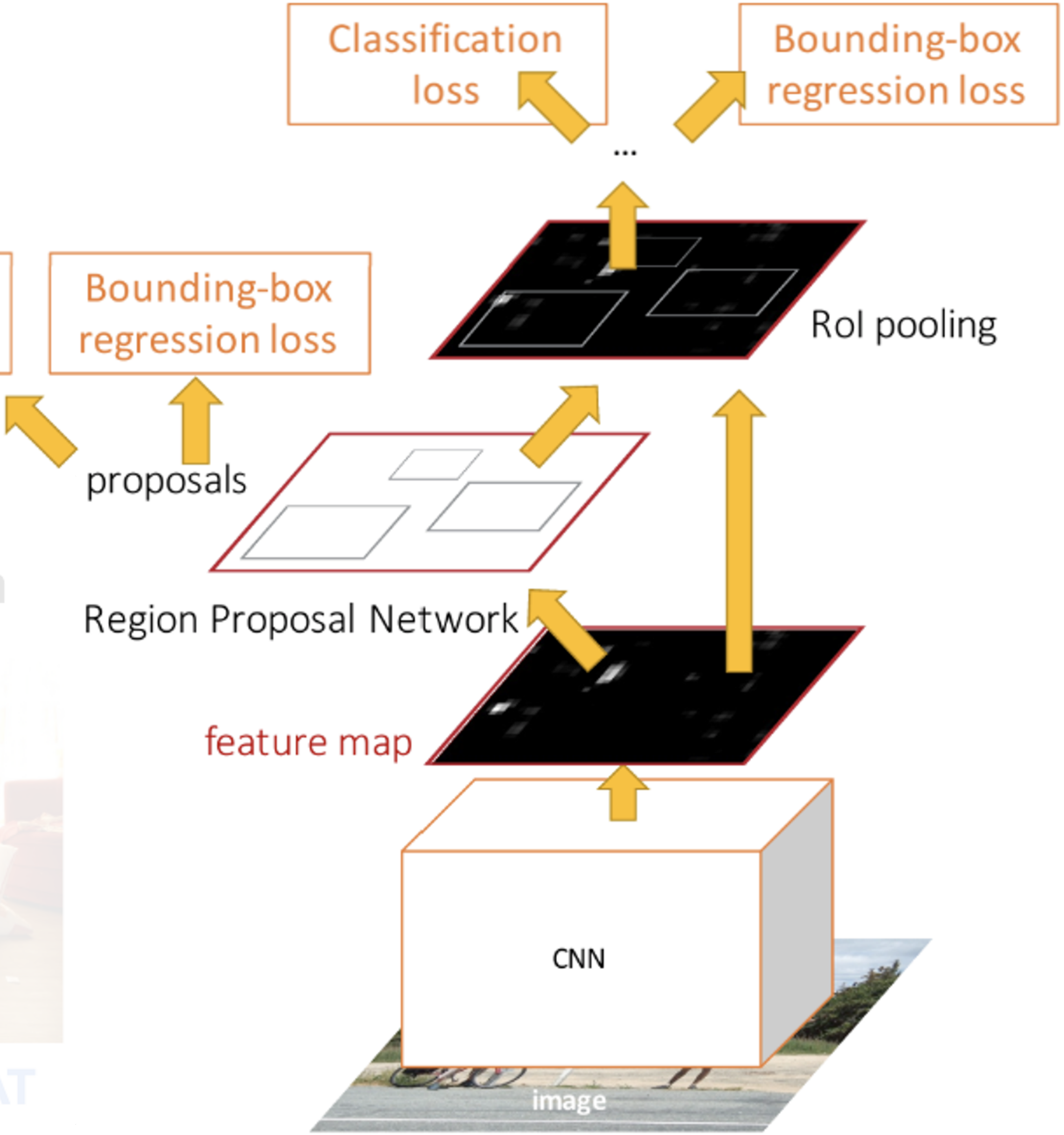
Object Detection: Faster R-CNN

Object Detection

DOG, DOG, CAT

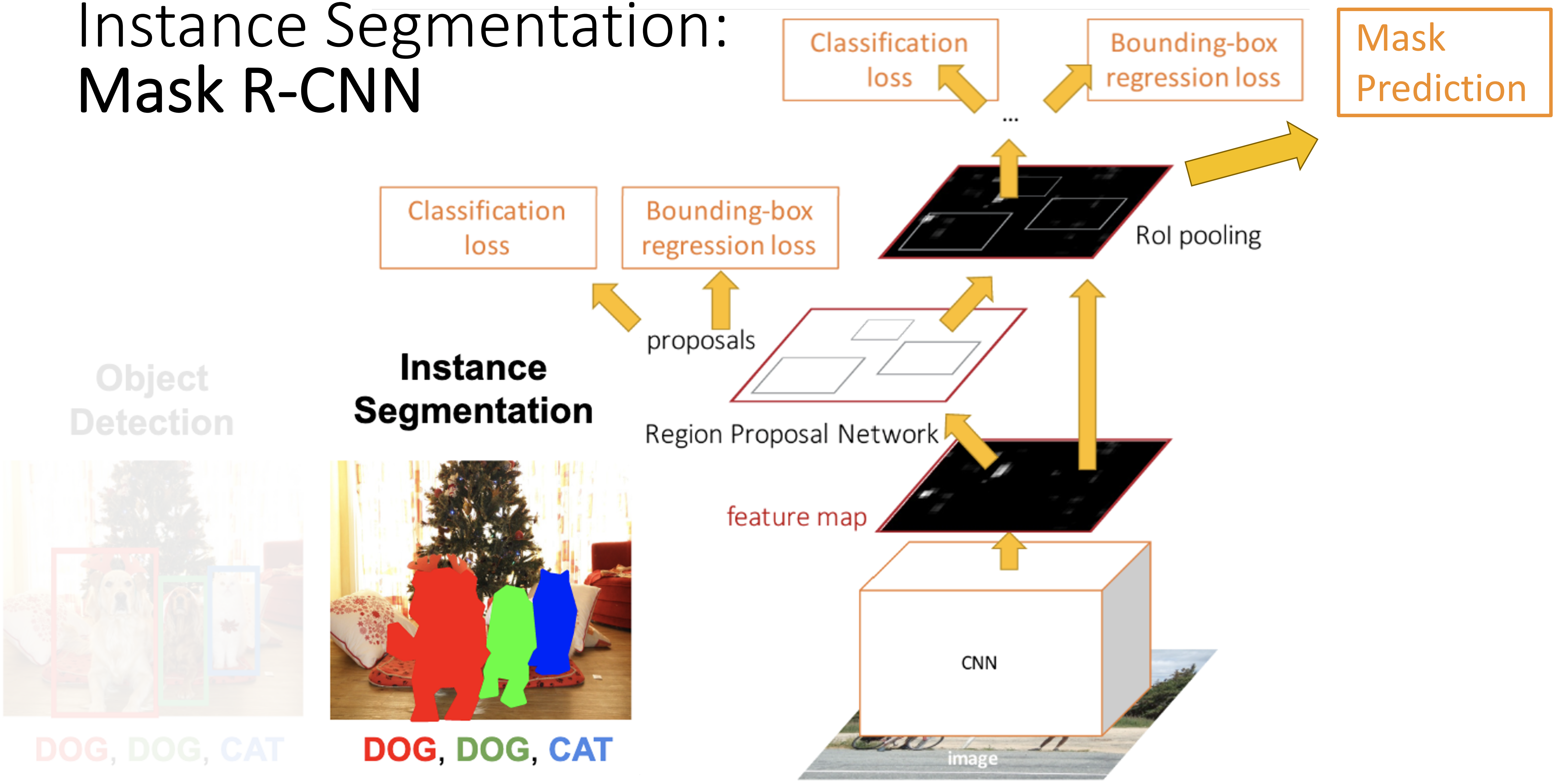
Instance Segmentation

DOG, DOG, CAT



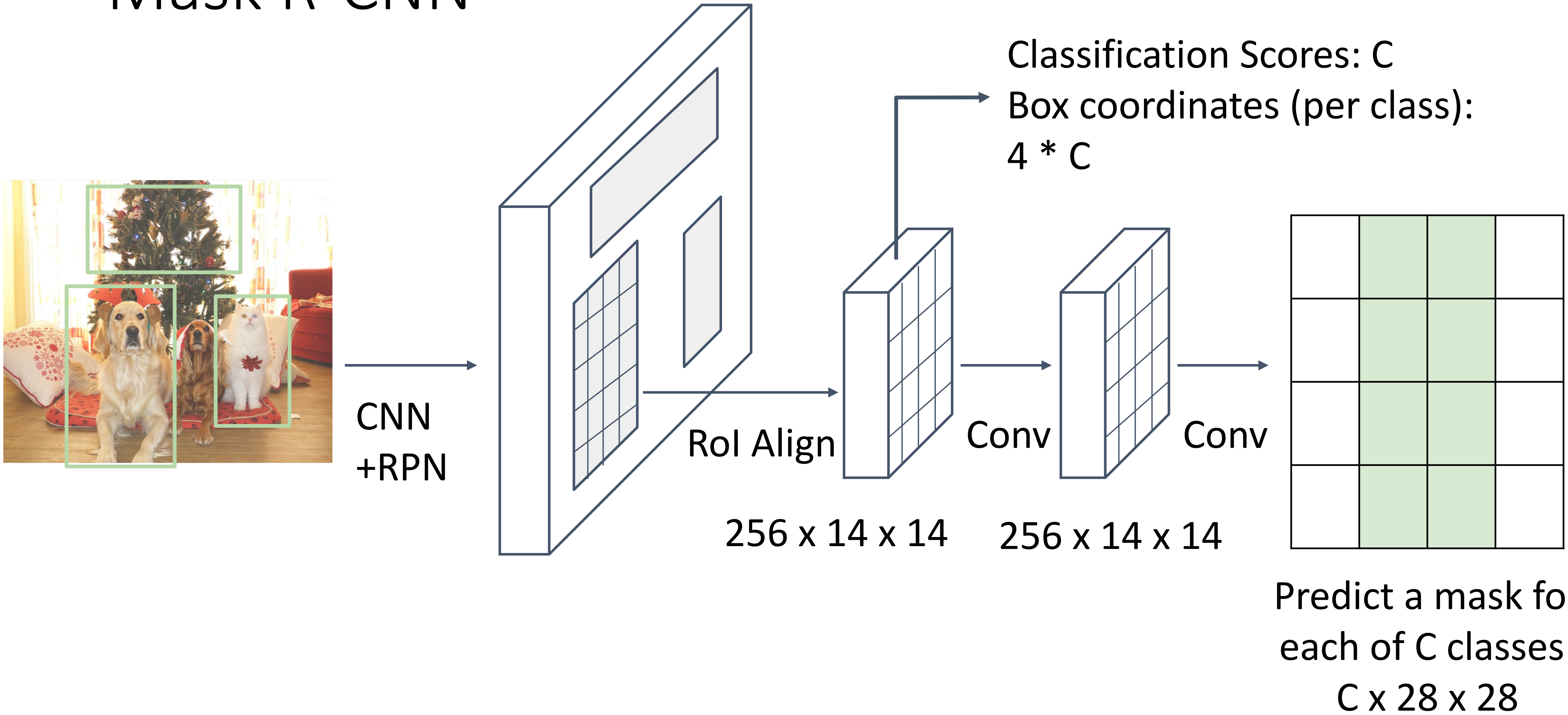
Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NeurIPS 2015

Instance Segmentation:
Mask R-CNN



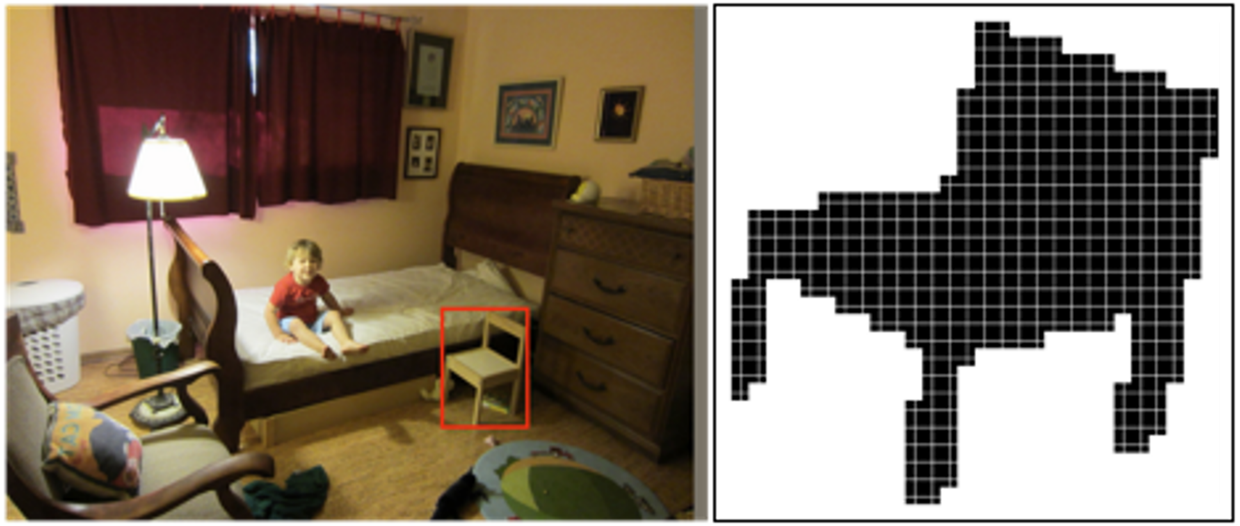
He et al, "Mask R-CNN", ICCV 2017

Mask R-CNN

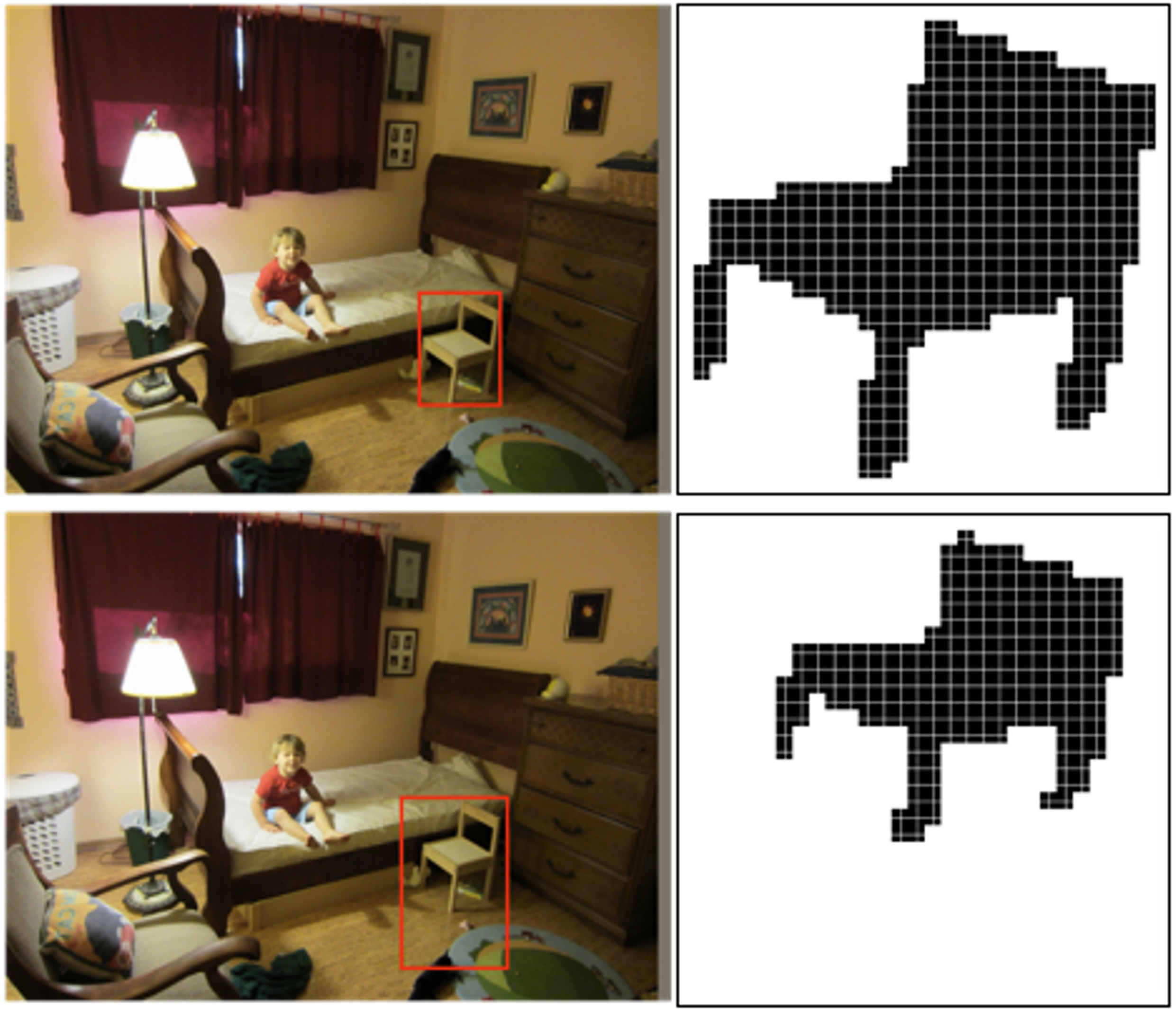


He et al, “Mask R-CNN”, ICCV 2017

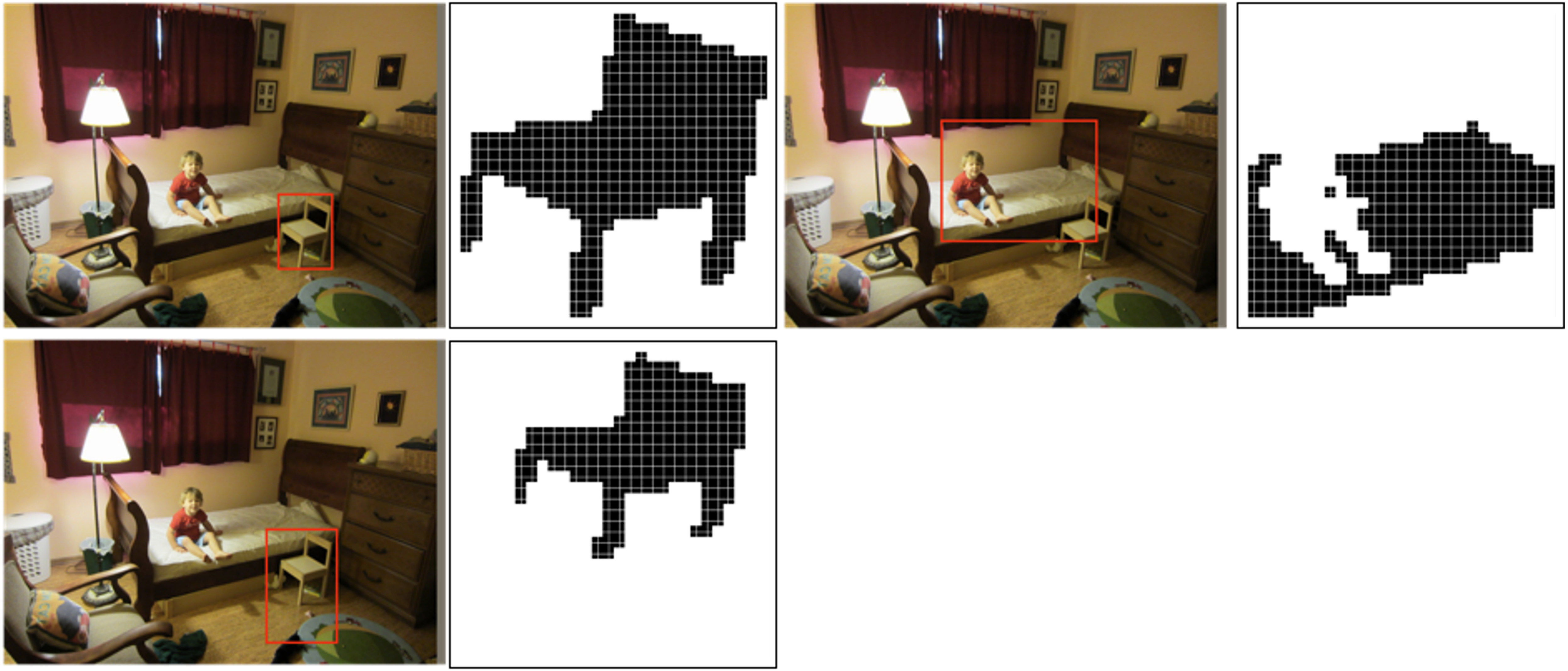
Mask R-CNN: Example Training Targets



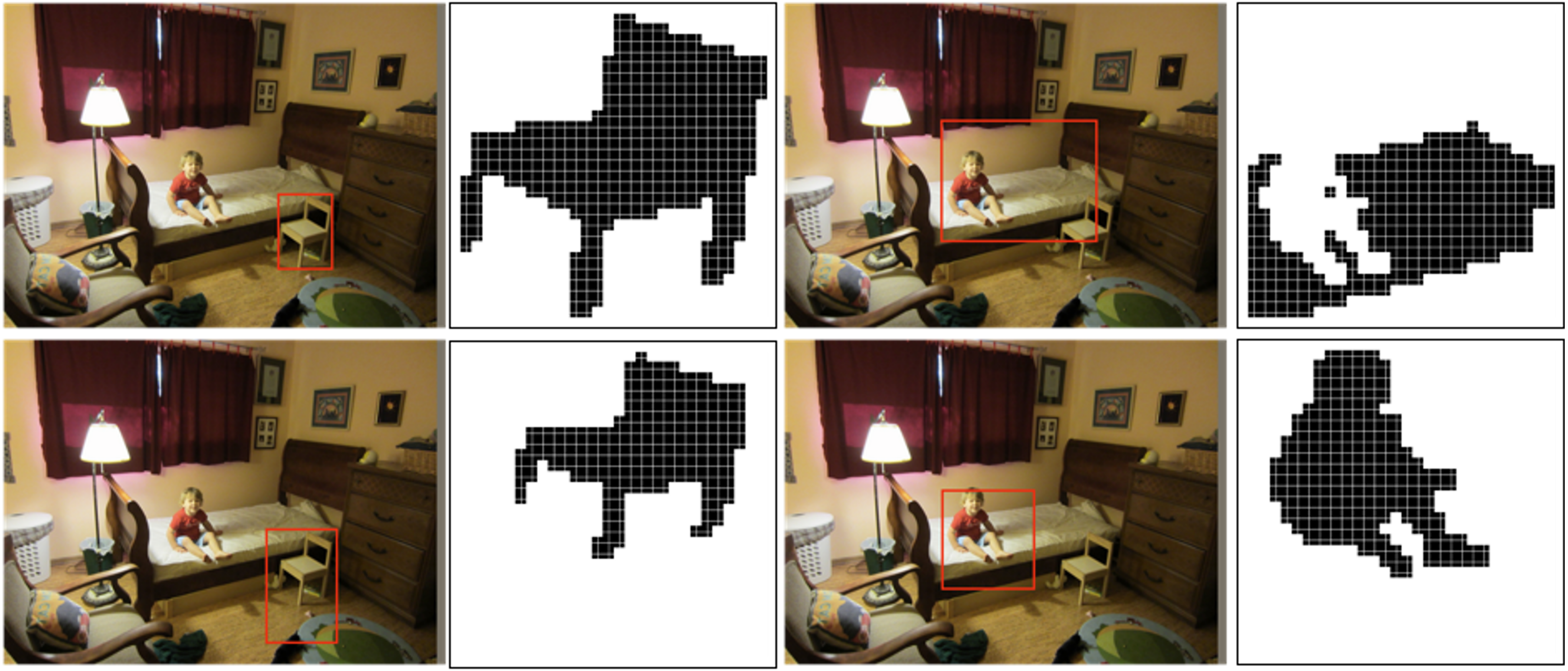
Mask R-CNN: Example Training Targets



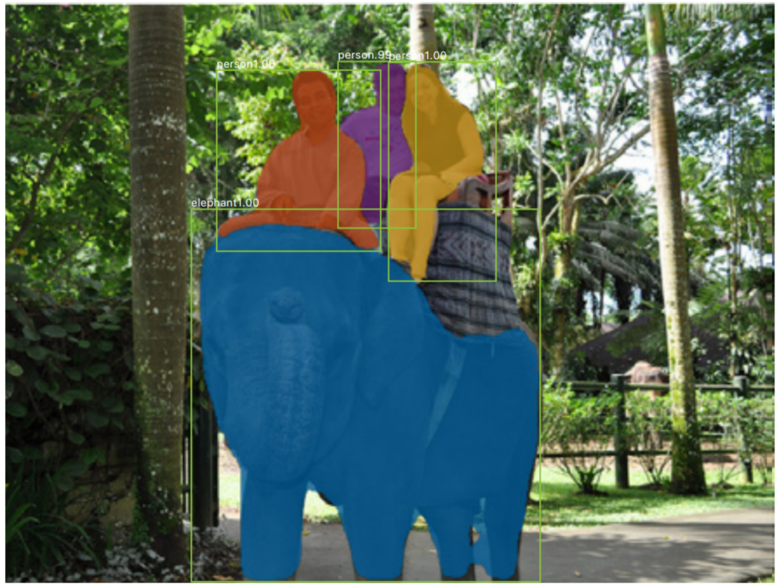
Mask R-CNN: Example Training Targets

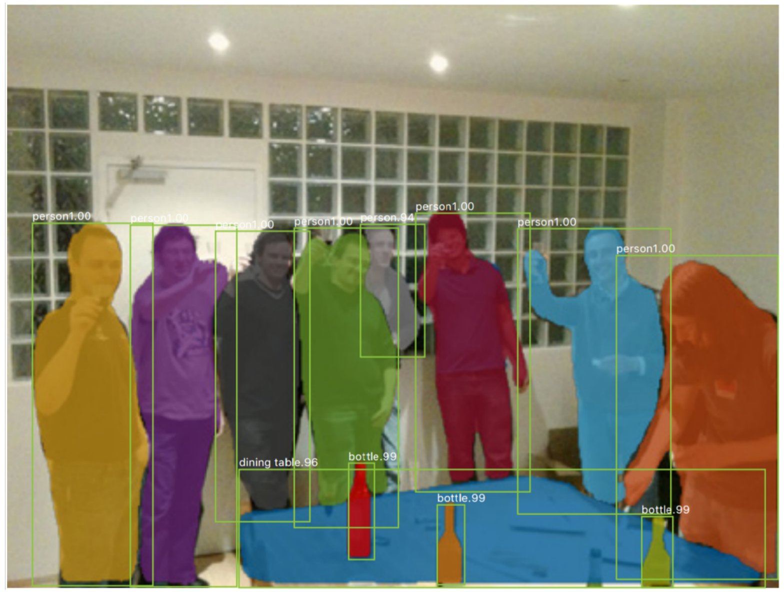


Mask R-CNN: Example Training Targets



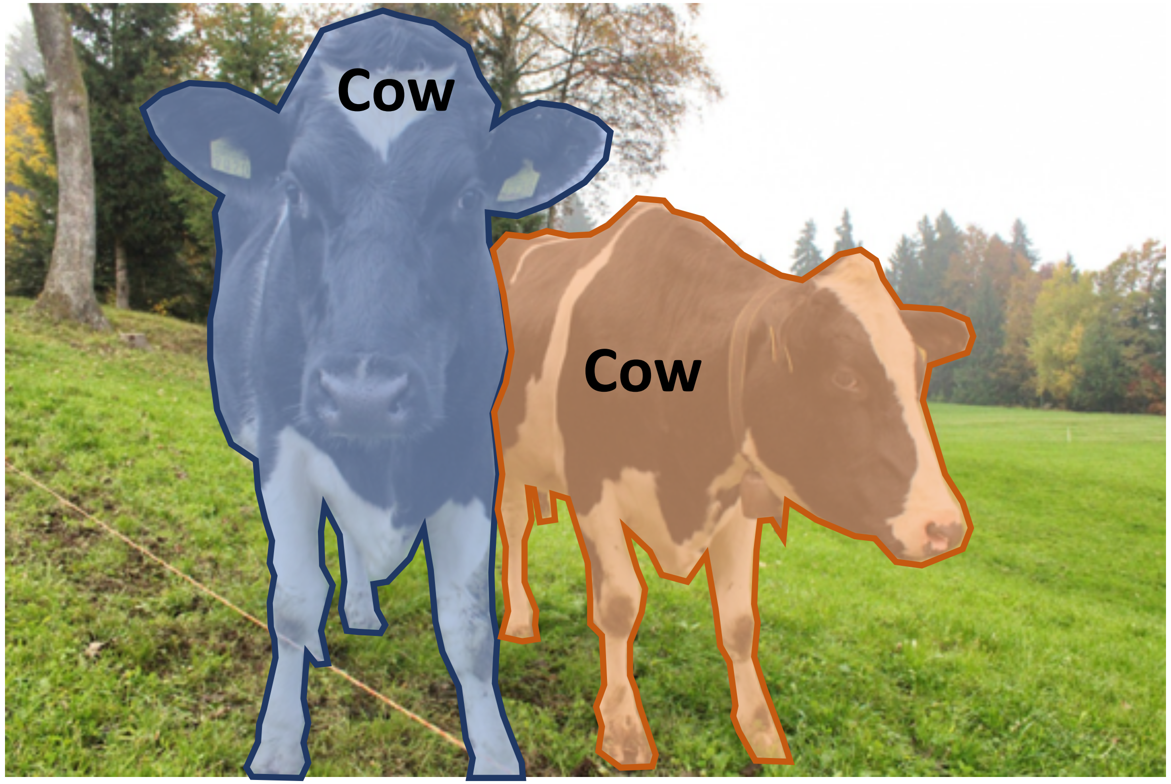
Mask R-CNN: Very Good Results!



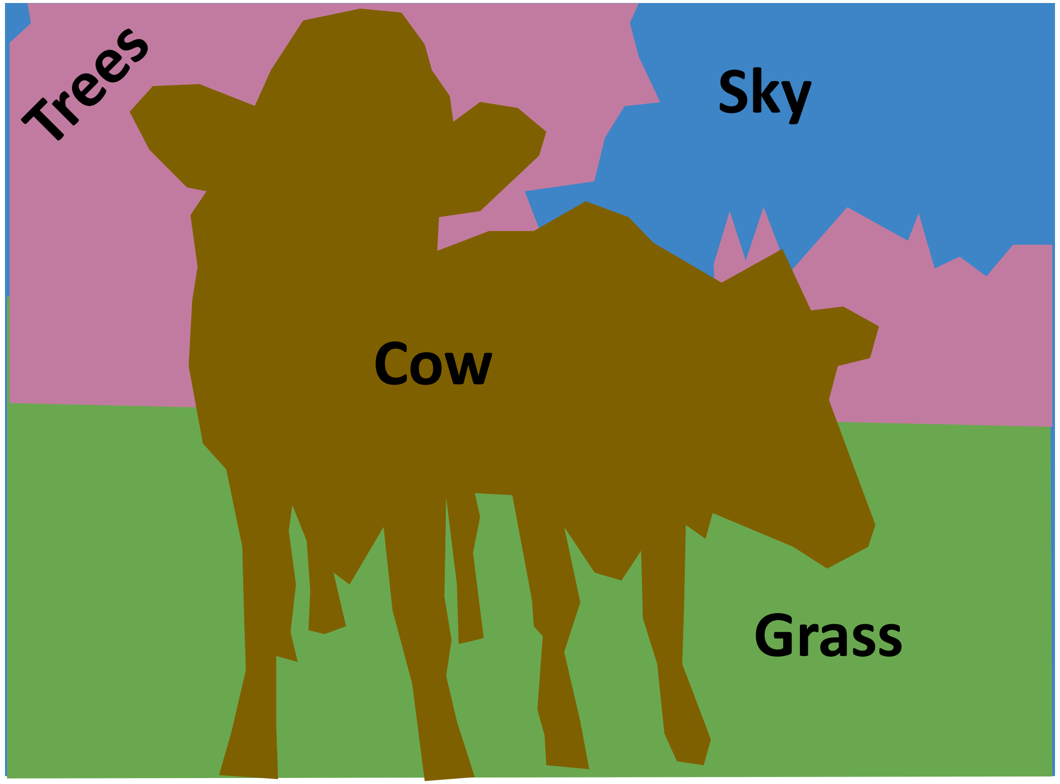


Beyond Instance Segmentation

Instance Segmentation: Separate object instances, but only things



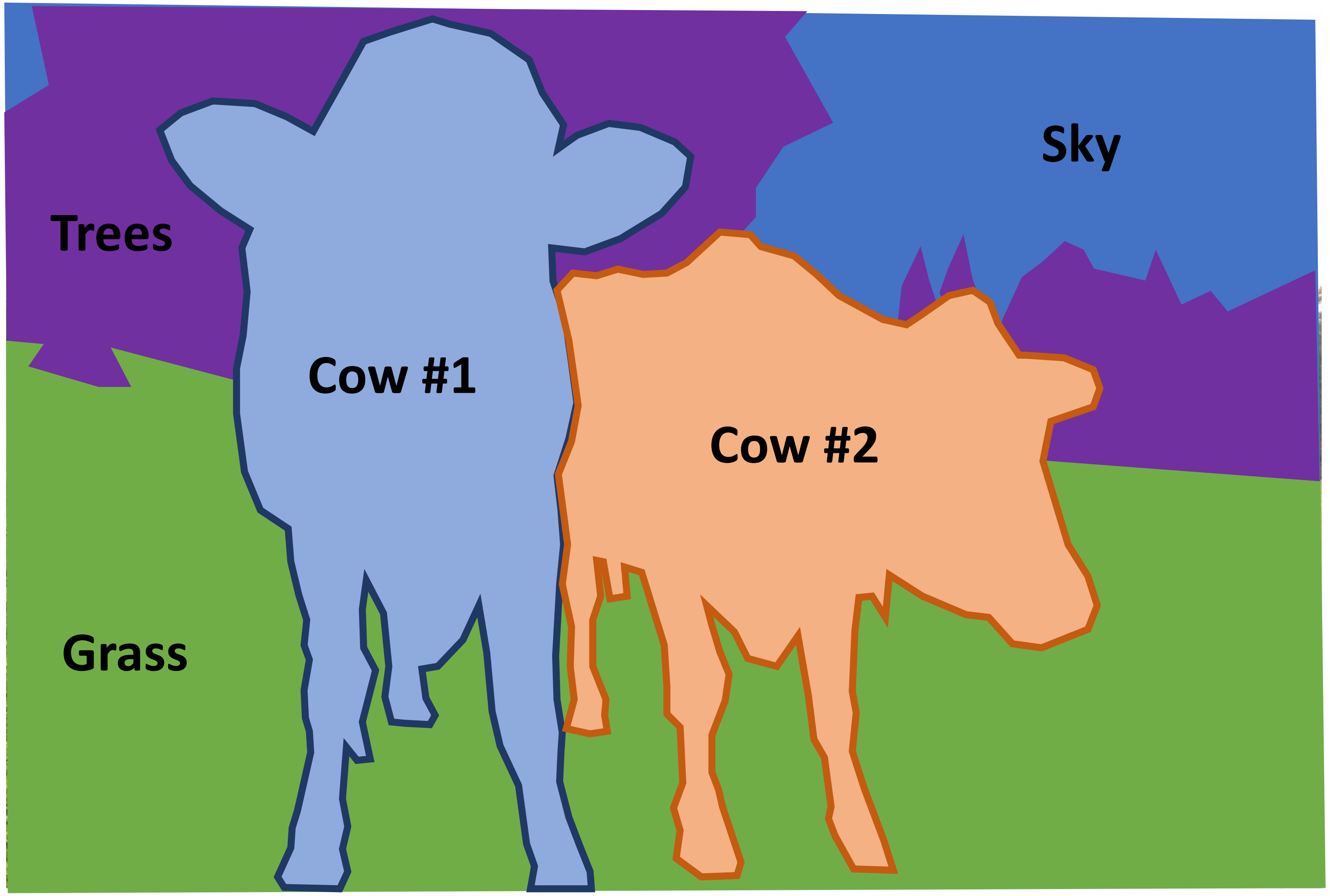
Semantic Segmentation: Identify both things and stuff, but doesn't separate instances



Beyond Instance Segmentation: **Panoptic Segmentation**

Label all pixels in the image (both things and stuff)

For "thing" categories also separate into instances



Kirillov et al, "Panoptic Segmentation", CVPR 2019
Kirillov et al, "Panoptic Feature Pyramid Networks", CVPR 2019

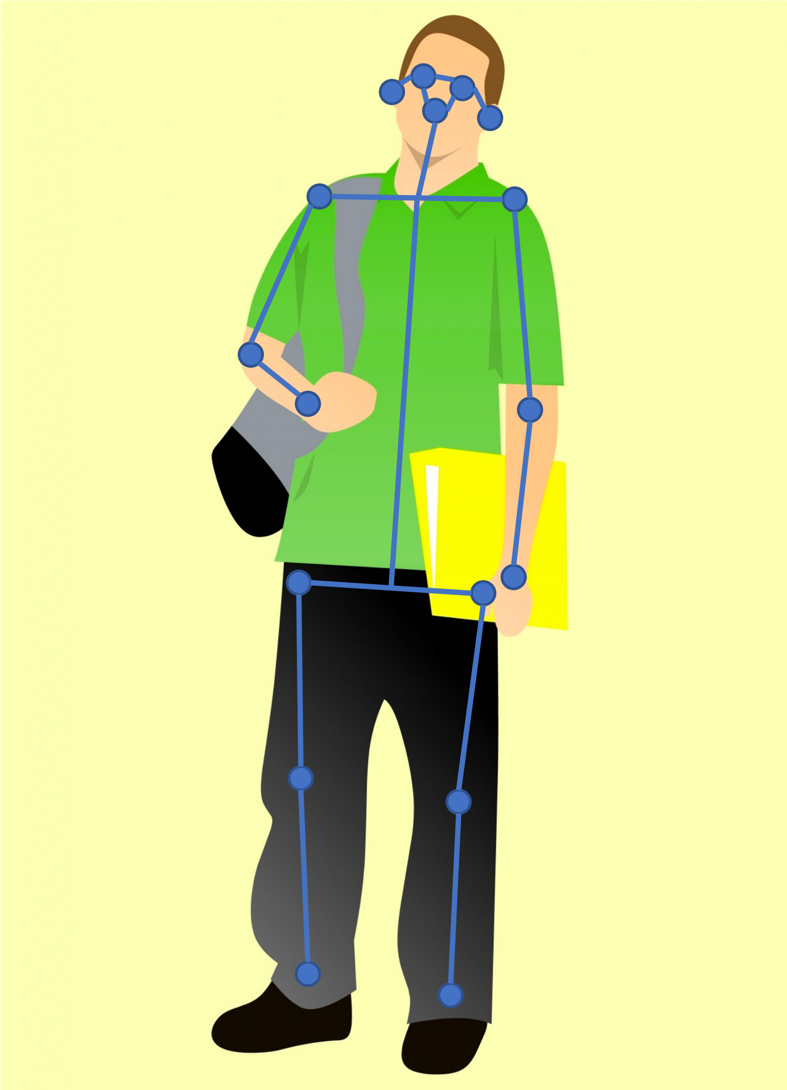
Beyond Instance Segmentation: **Panoptic Segmentation**

Kirillov et al, “Panoptic Feature Pyramid Networks”, CVPR 2019

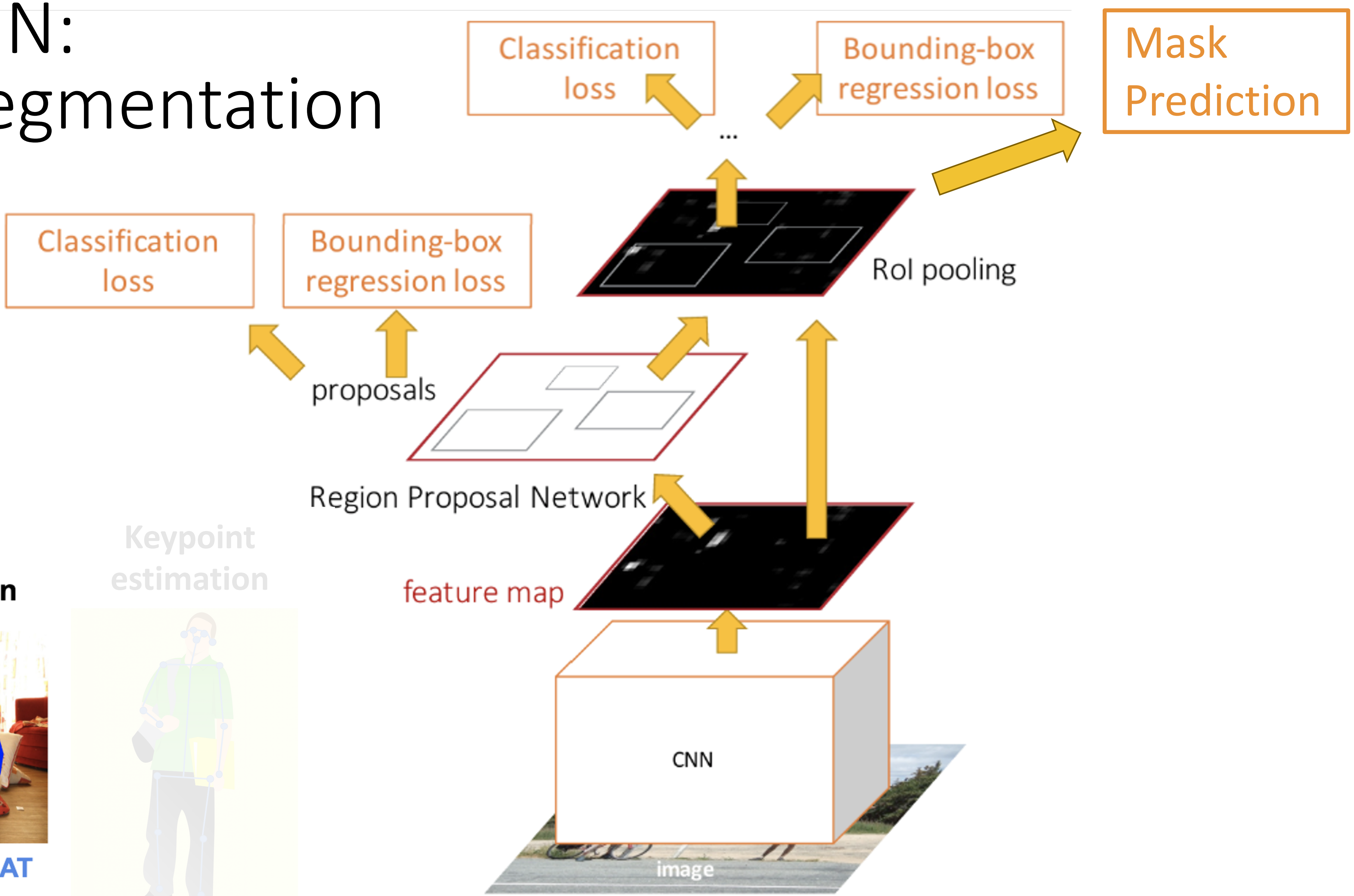
Beyond Instance Segmentation: Human Keypoints

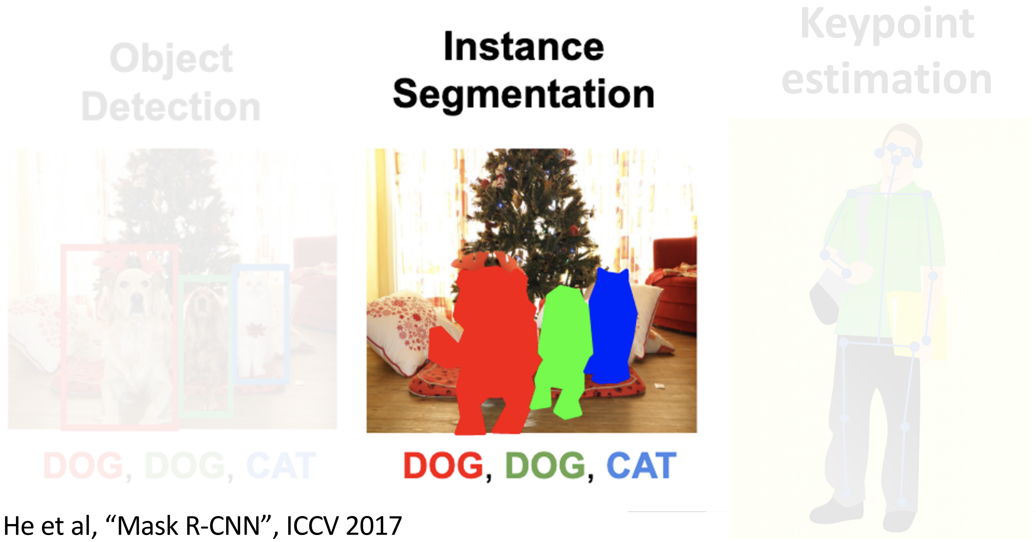
Represent the pose of a human by locating a set of **keypoints**

e.g. 17 keypoints:
- Nose
- Left / Right eye
- Left / Right ear
- Left / Right shoulder
- Left / Right elbow
- Left / Right wrist
- Left / Right hip
- Left / Right knee
- Left / Right ankle



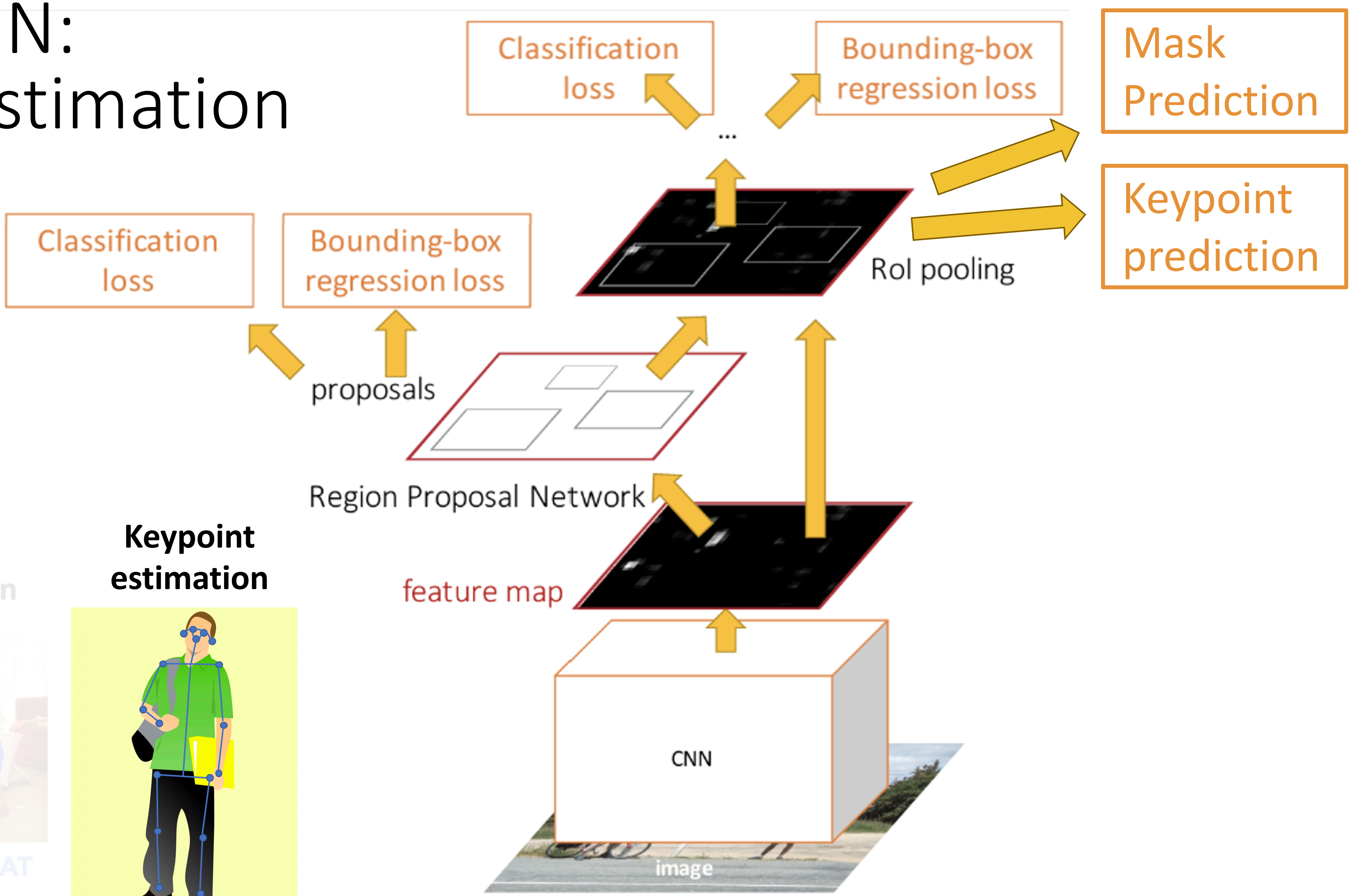
Mask R-CNN: Instance Segmentation

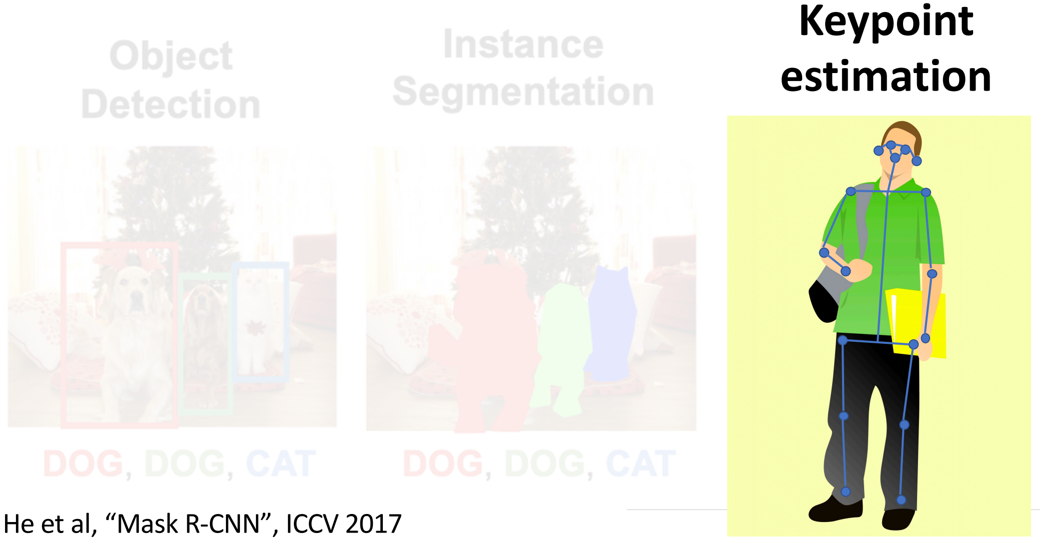




He et al, "Mask R-CNN", ICCV 2017

Mask R-CNN: Keypoint Estimation



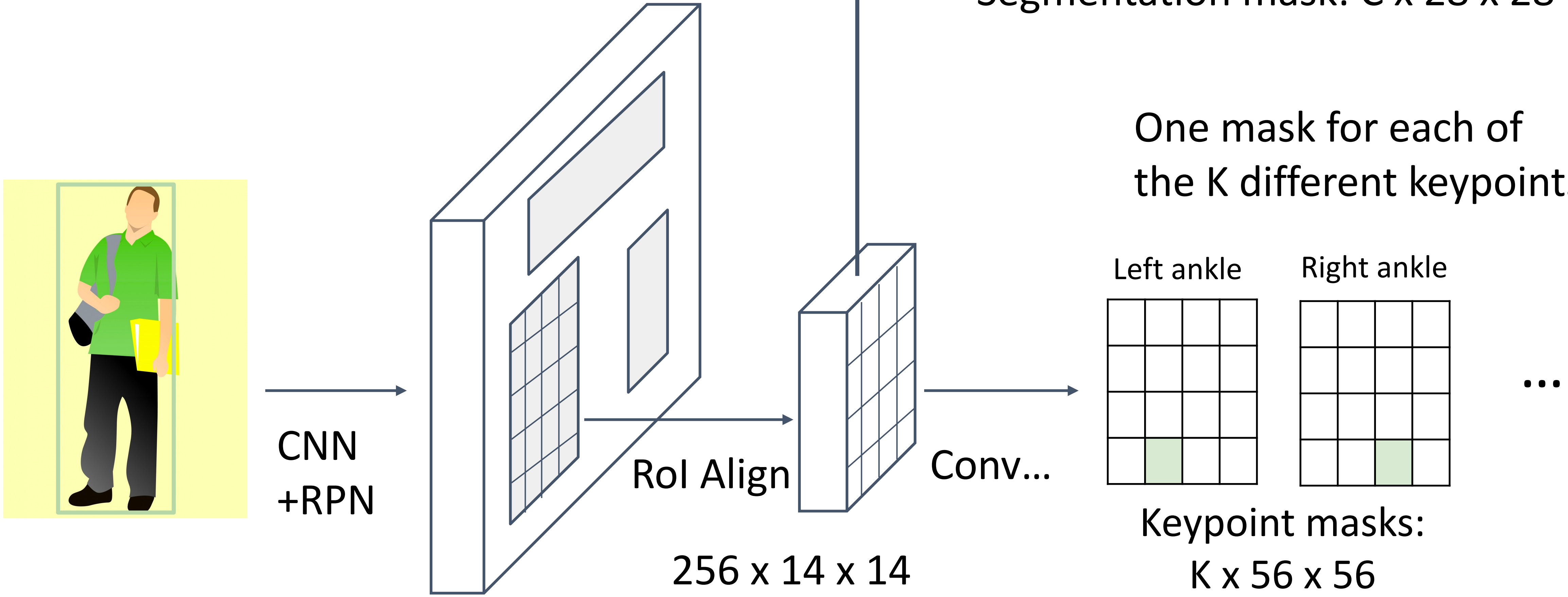


He et al, "Mask R-CNN", ICCV 2017

Mask R-CNN: Keypoints

Classification Scores: C
Box coordinates (per class): 4 * C
Segmentation mask: C x 28 x 28

One mask for each of
the K different keypoints



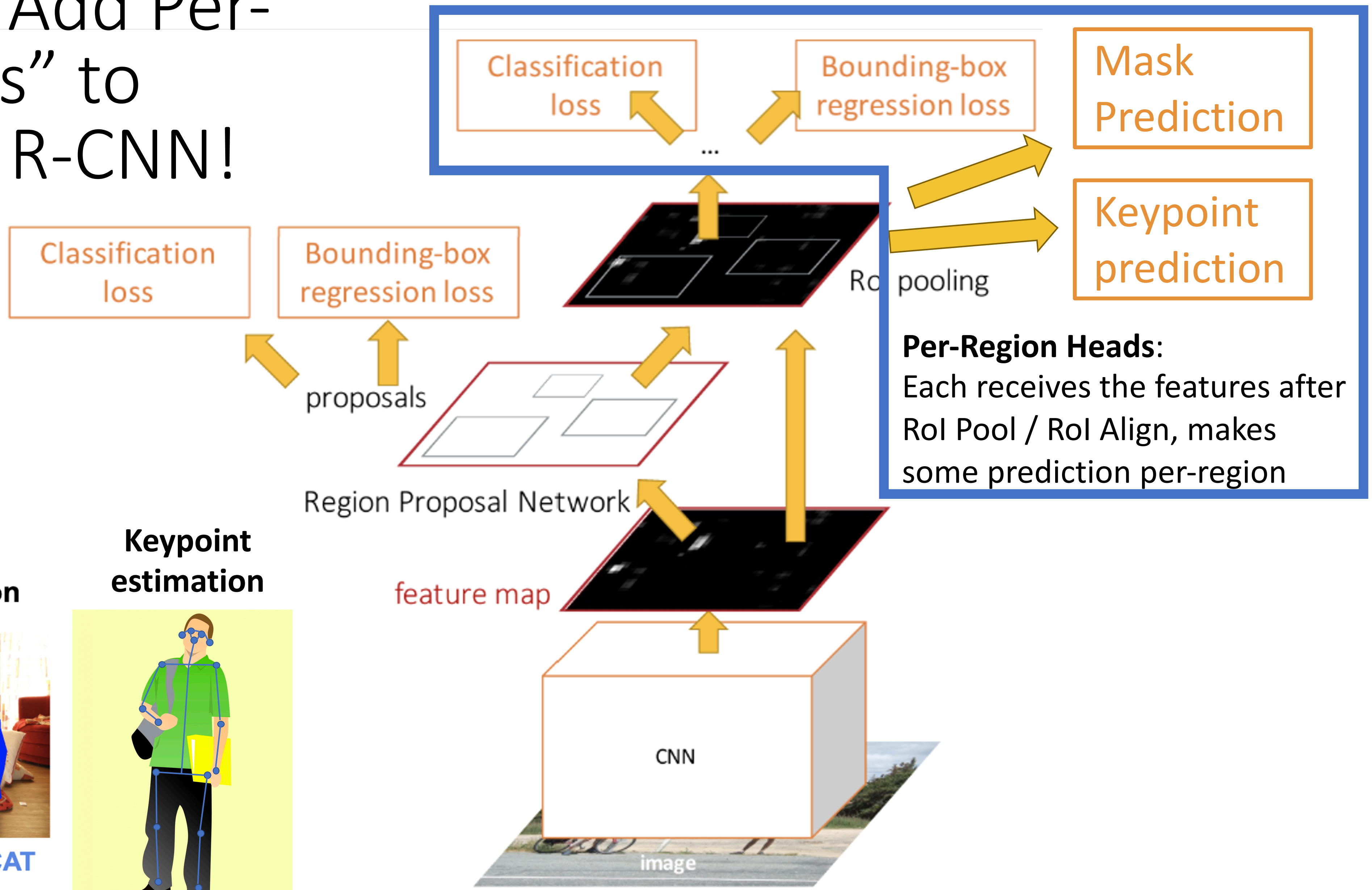
Ground-truth has one "pixel" turned on
per keypoint. Train with softmax loss

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

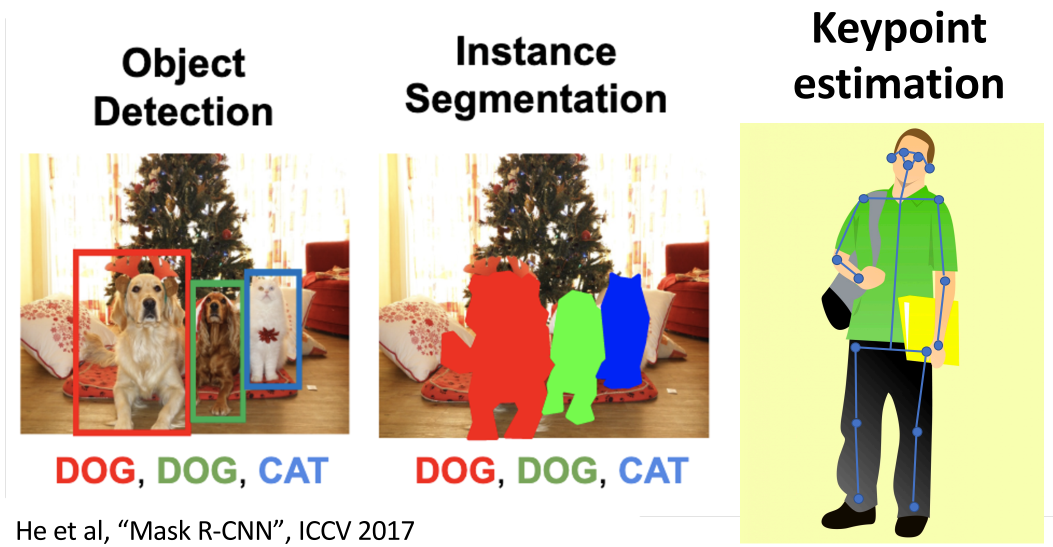
Joint Instance Segmentation and Pose Estimation

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

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

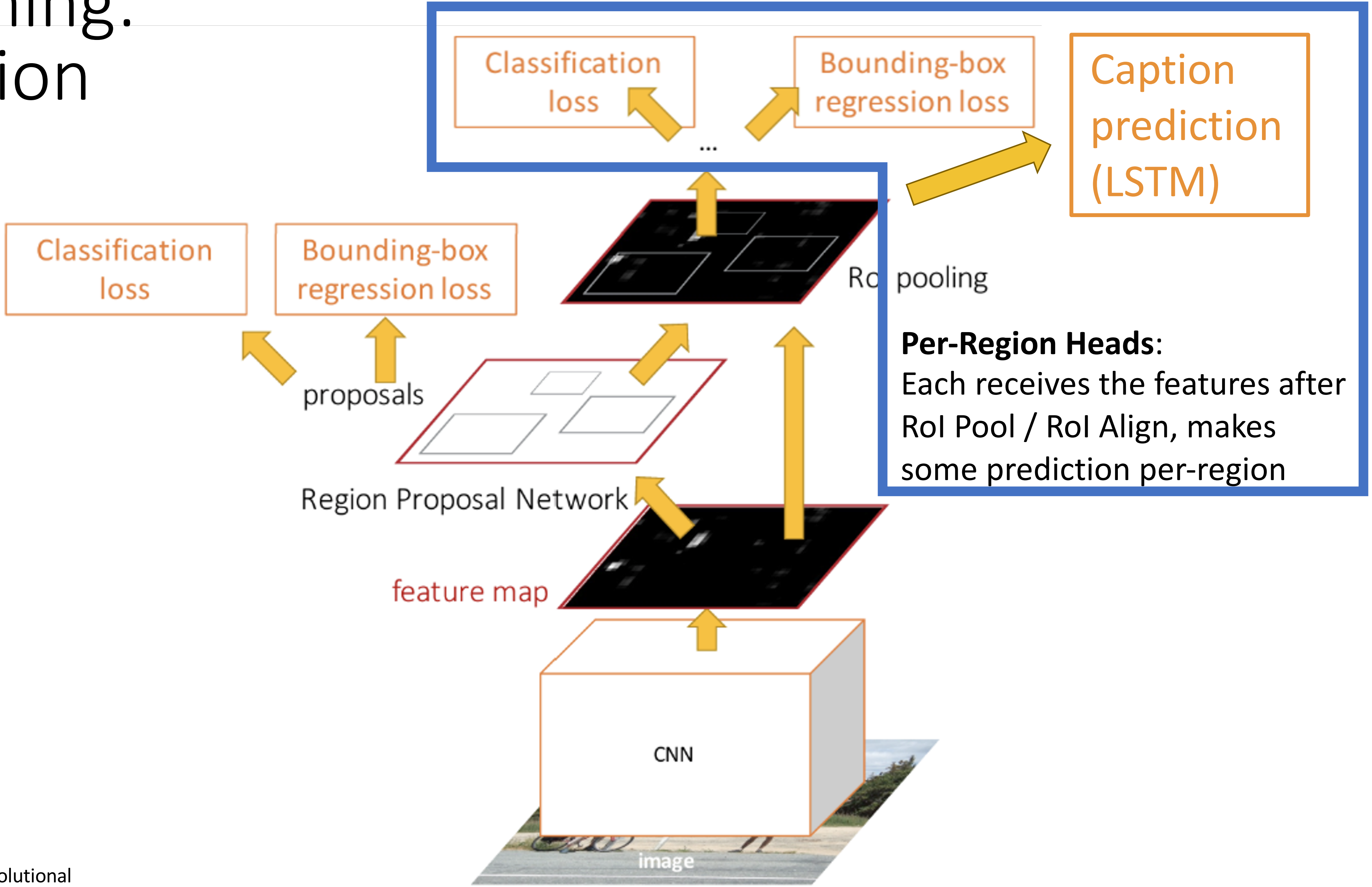


Per-Region Heads: Each receives the features after RoI Pool / RoI Align, makes some prediction per-region



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

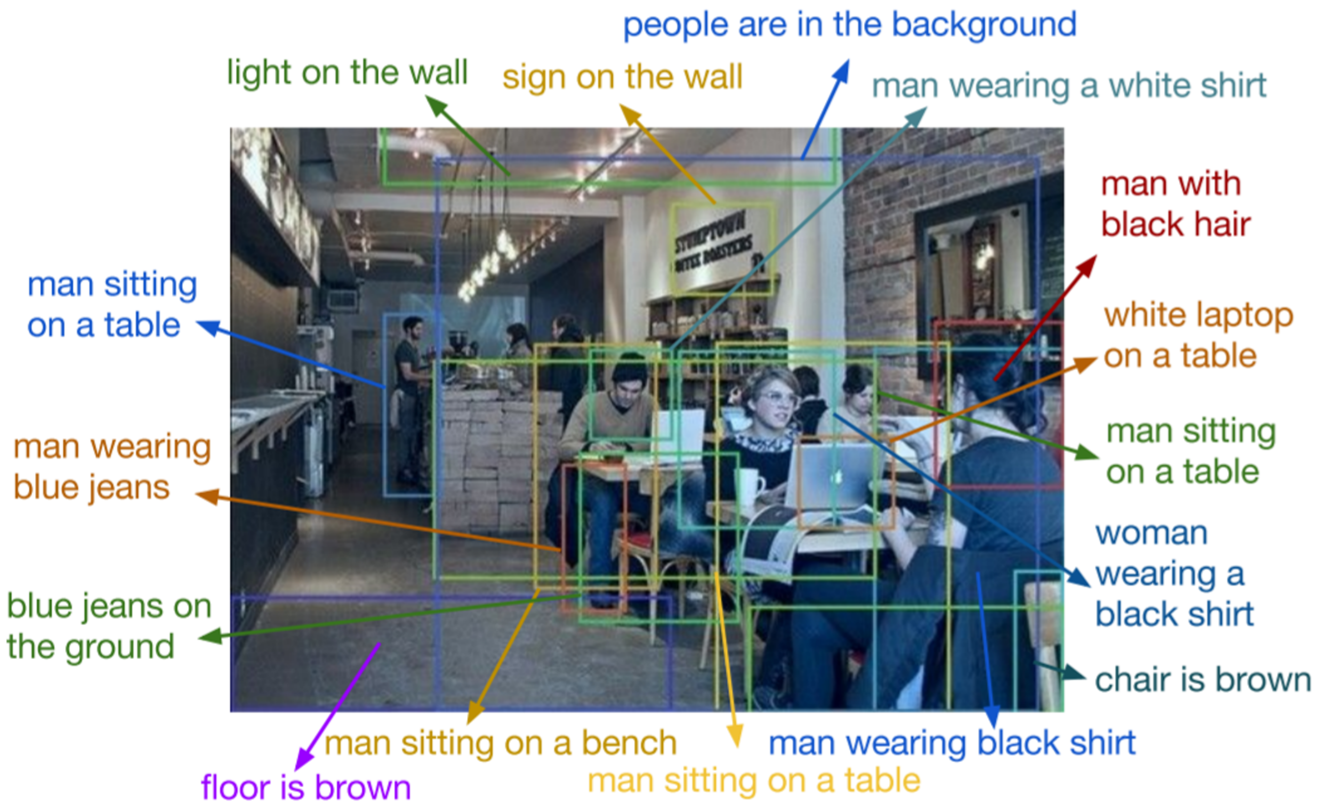
Dense Captioning: Predict a caption per region!

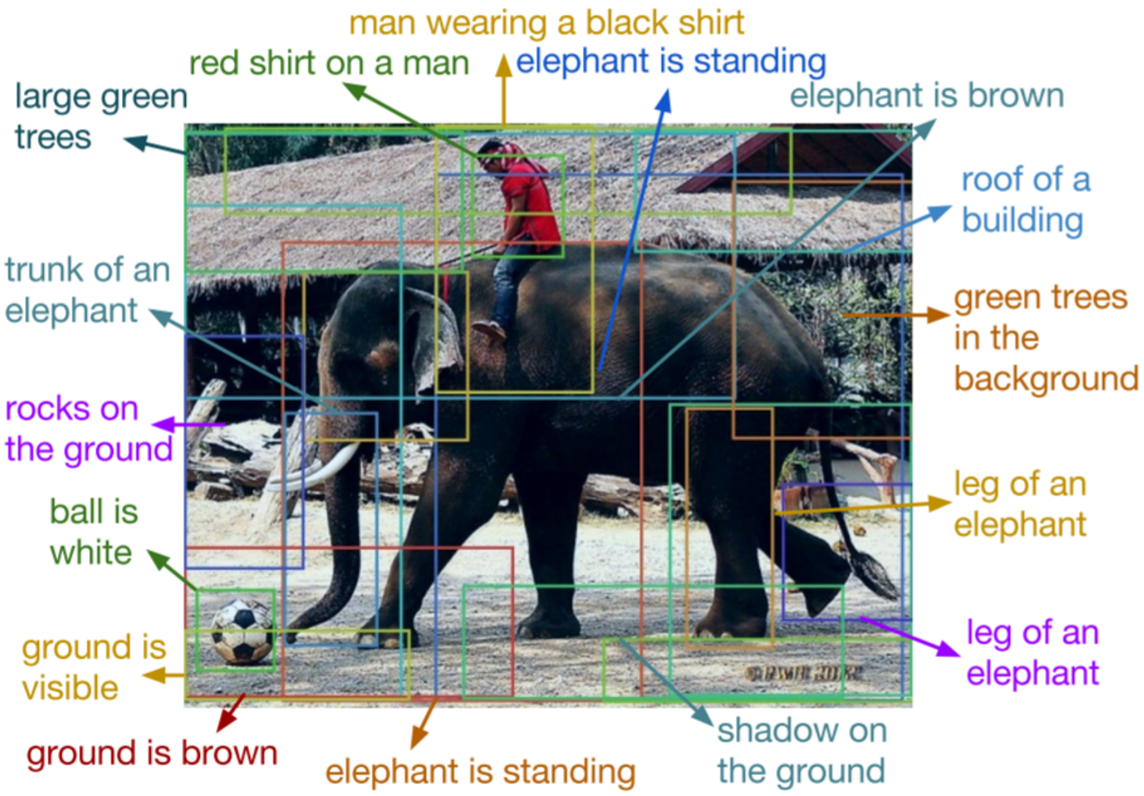


Per-Region Heads: Each receives the features after RoI Pool / RoI Align, makes some prediction per-region

Johnson, Karpathy, and Fei-Fei, "DenseCap: Fully Convolutional Localization Networks for Dense Captioning", CVPR 2016

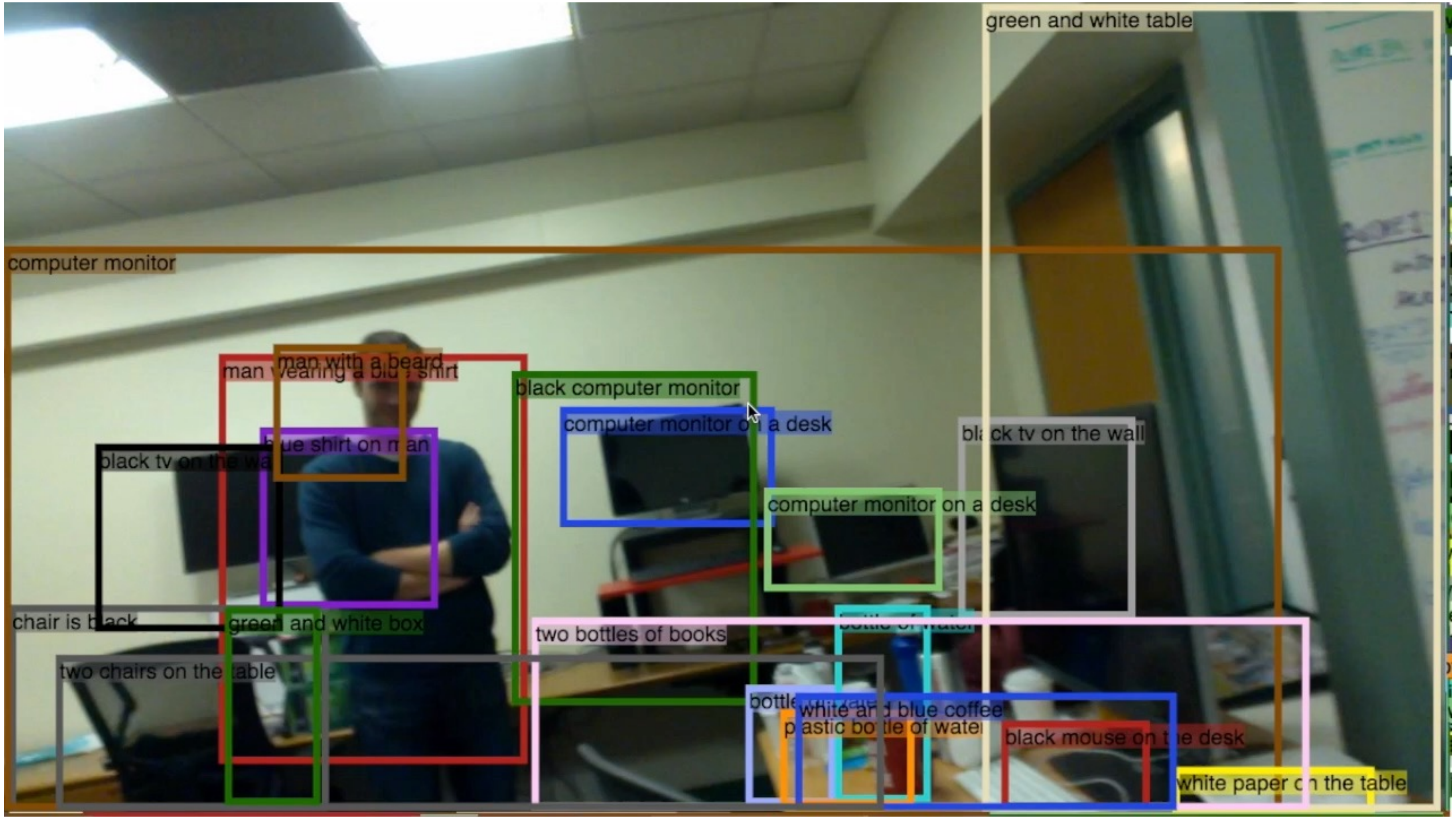
Dense Captioning





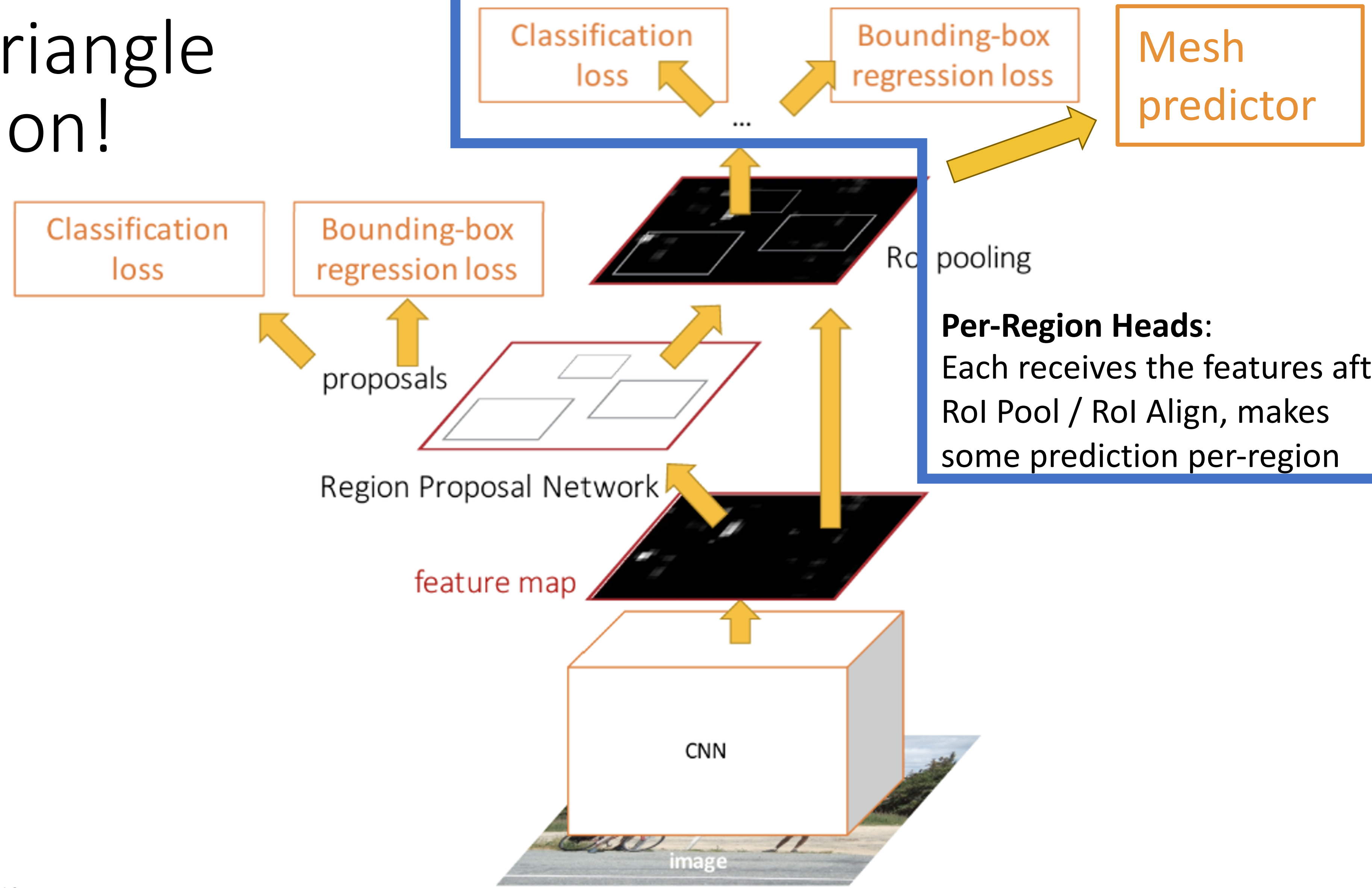
Johnson, Karpathy, and Fei-Fei, "DenseCap: Fully Convolutional Localization Networks for Dense Captioning", CVPR 2016

Dense Captioning



Johnson, Karpathy, and Fei-Fei, "DenseCap: Fully Convolutional Localization Networks for Dense Captioning", CVPR 2016

3D Shape Prediction: Predict a 3D triangle mesh per region!

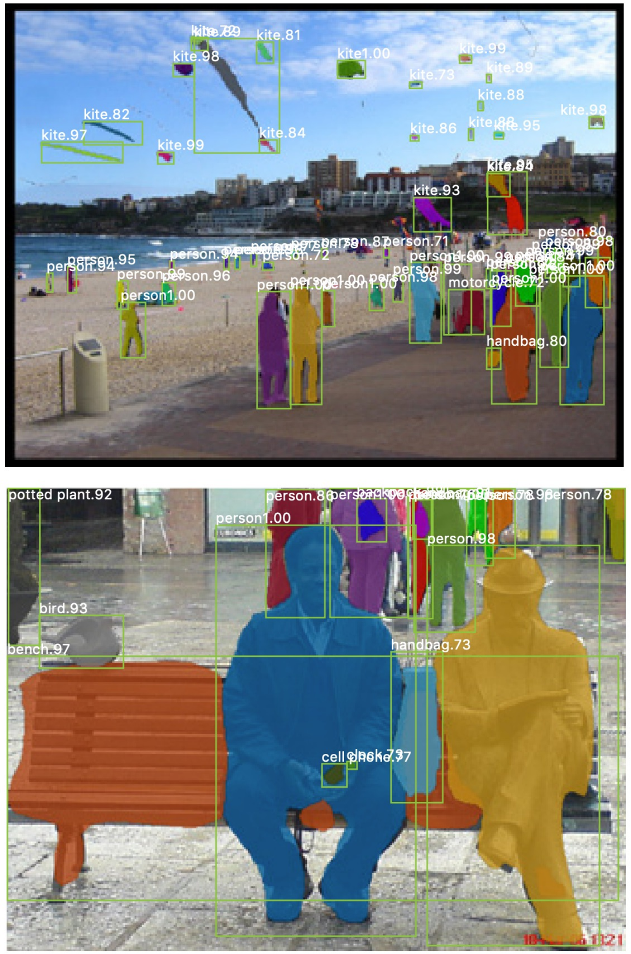


Per-Region Heads:
Each receives the features after RoI Pool / RoI Align, makes some prediction per-region

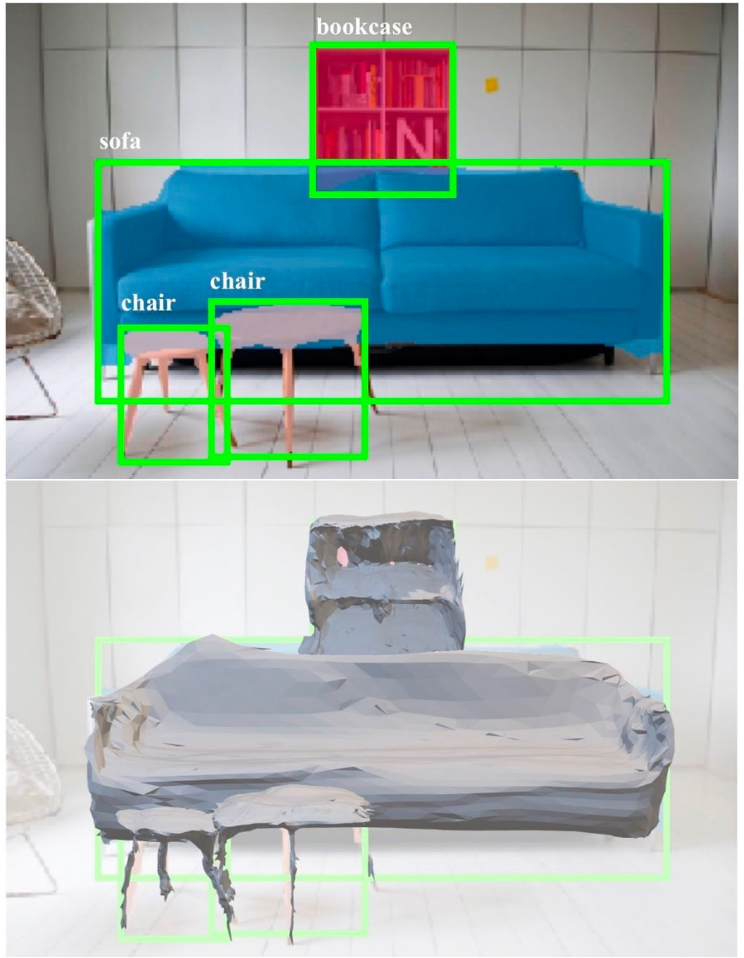
Gkioxari, Malik, and Johnson, "Mesh R-CNN", ICCV 2019

3D Shape Prediction: Mask R-CNN + Mesh Head

Mask R-CNN:
2D Image -> 2D shapes



Mesh R-CNN:
2D Image -> **3D** shapes

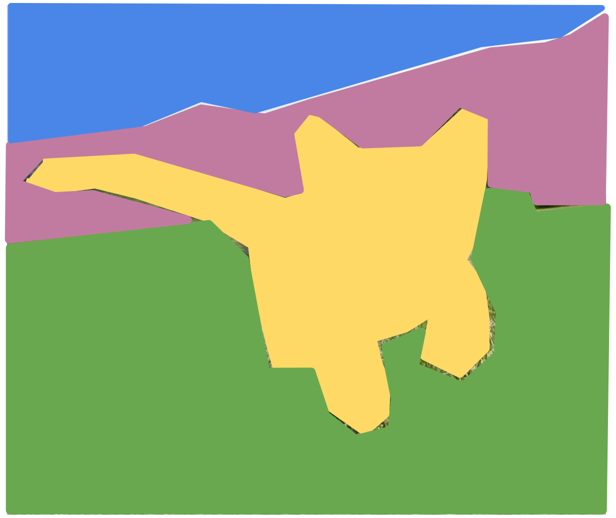
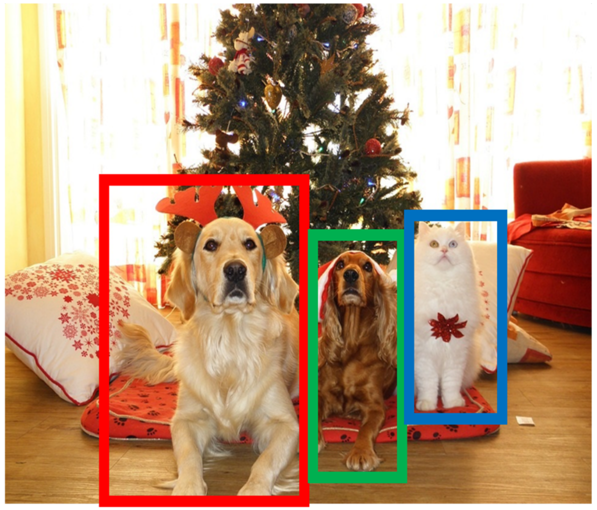
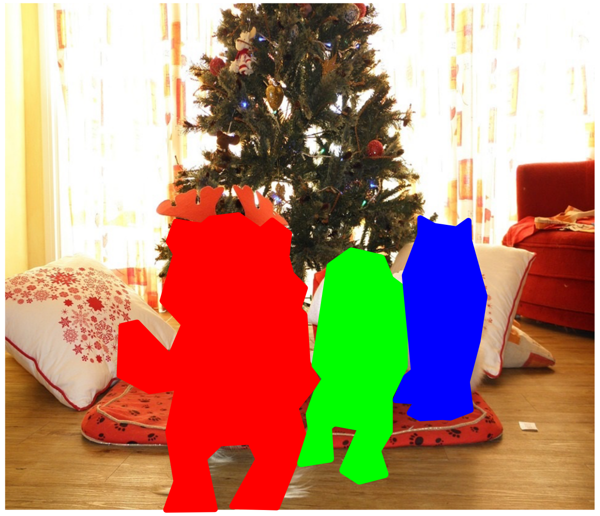


More details next time!

He, Gkioxari, Dollár, and Girshick, “Mask R-CNN”, ICCV 2017

Gkioxari, Malik, and Johnson, “Mesh R-CNN”, ICCV 2019

Summary: Many Computer Vision Tasks!

Classification	Semantic Segmentation	Object Detection	Instance Segmentation
			
CAT	**GRASS, CAT, TREE, SKY**	**DOG, DOG, CAT**	**DOG, DOG, CAT**
No spatial extent	No objects, just pixels	Multiple Objects	

This image is CC0 public domain

Next Time:
Recurrent Neural Networks