Lecture 16:
Detection + Segmentation

Reminder: A4

A4 due **Friday October 30, 11:59pm**

A4 covers:

- PyTorch autograd
- Residual networks
- Recurrent neural networks
- Attention
- Feature visualization
- Style transfer
- Adversarial examples

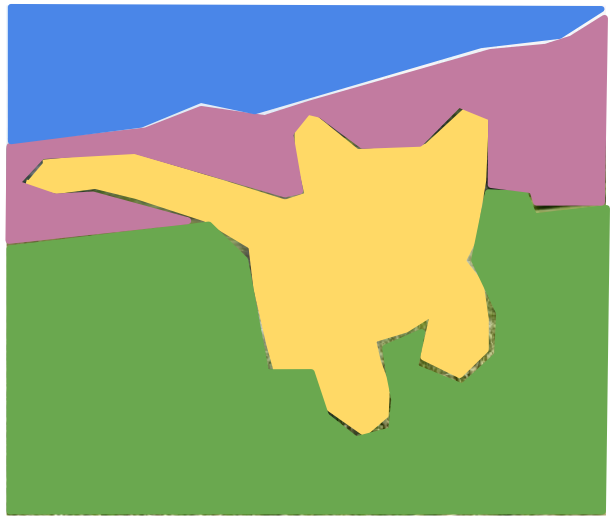
Last Time: Computer Vision Tasks

Classification

CAT

No spatial extent

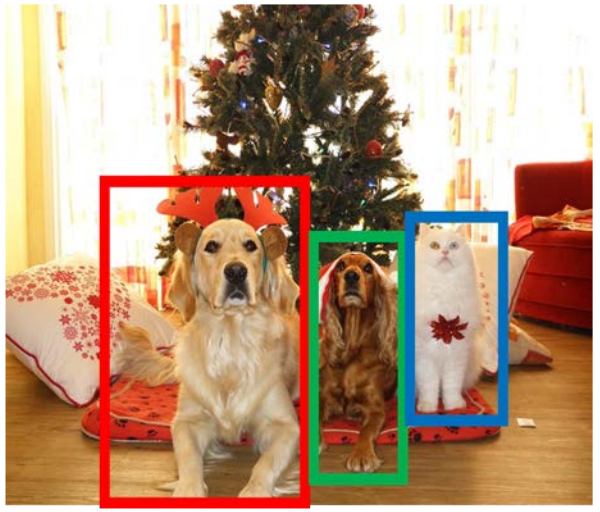
Semantic Segmentation



GRASS, **CAT**, **TREE**, **SKY**

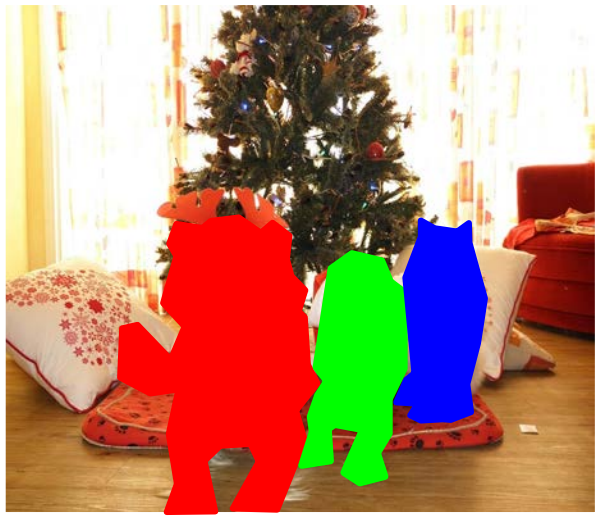
No objects, just pixels

Object Detection



DOG, **DOG**, **CAT**

Instance Segmentation



DOG, **DOG**, **CAT**

Multiple Objects

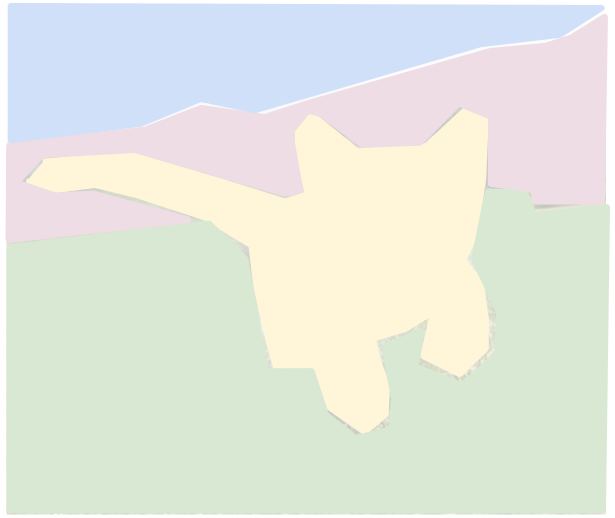
Last Time: Object Detection

Classification

CAT

No spatial extent

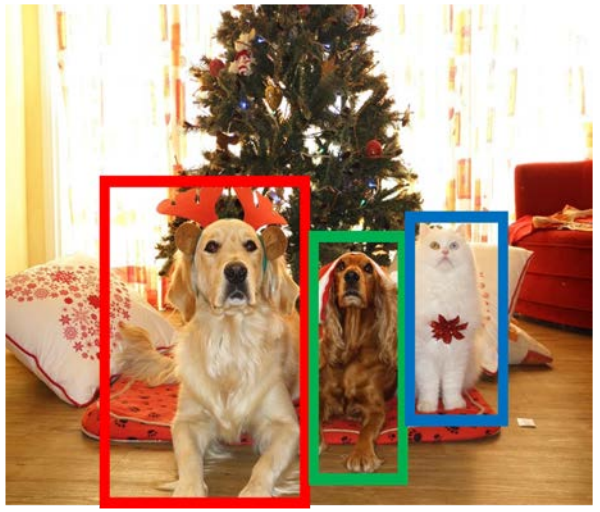
Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

Object Detection



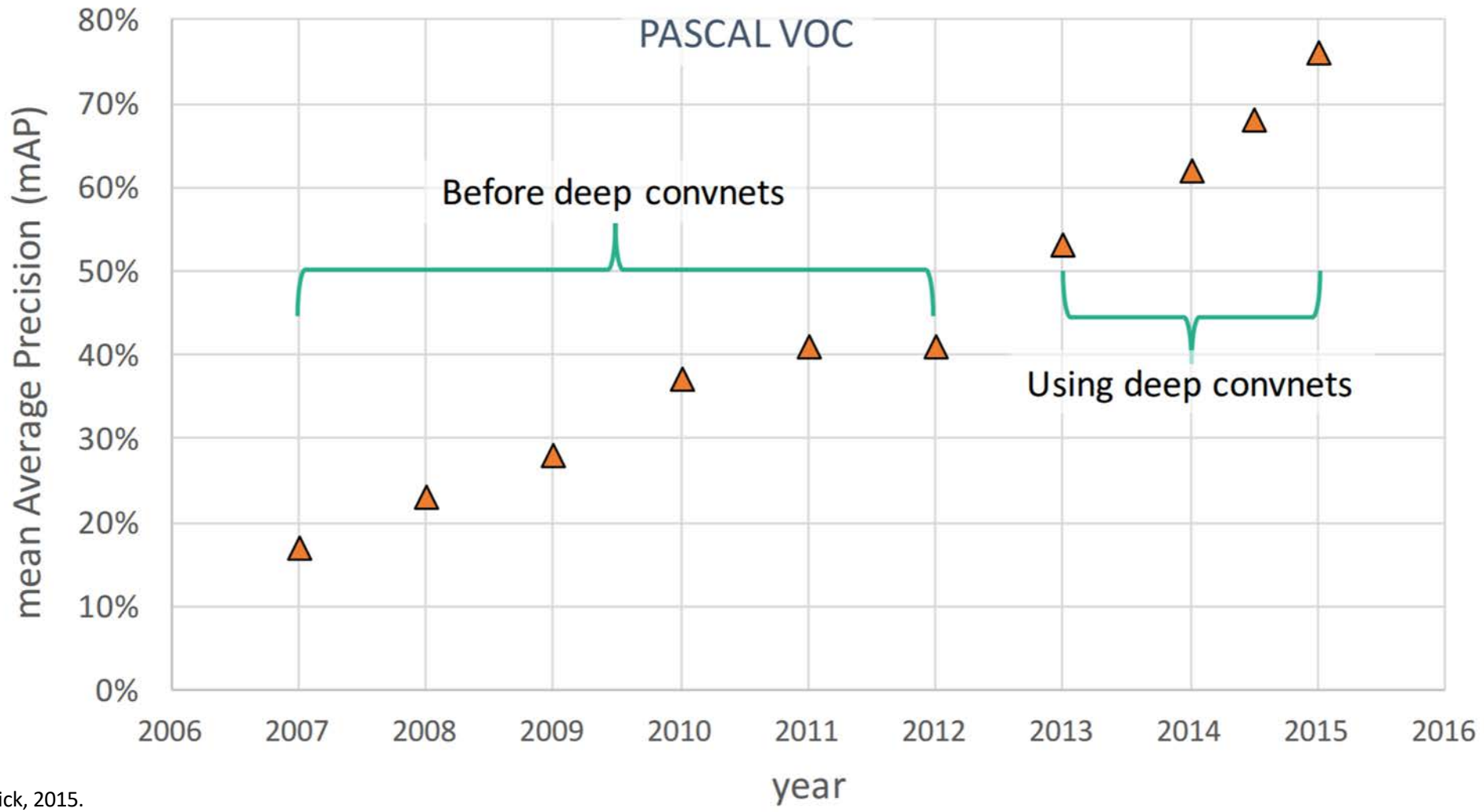
DOG, DOG, CAT

Instance Segmentation

DOG, DOG, CAT

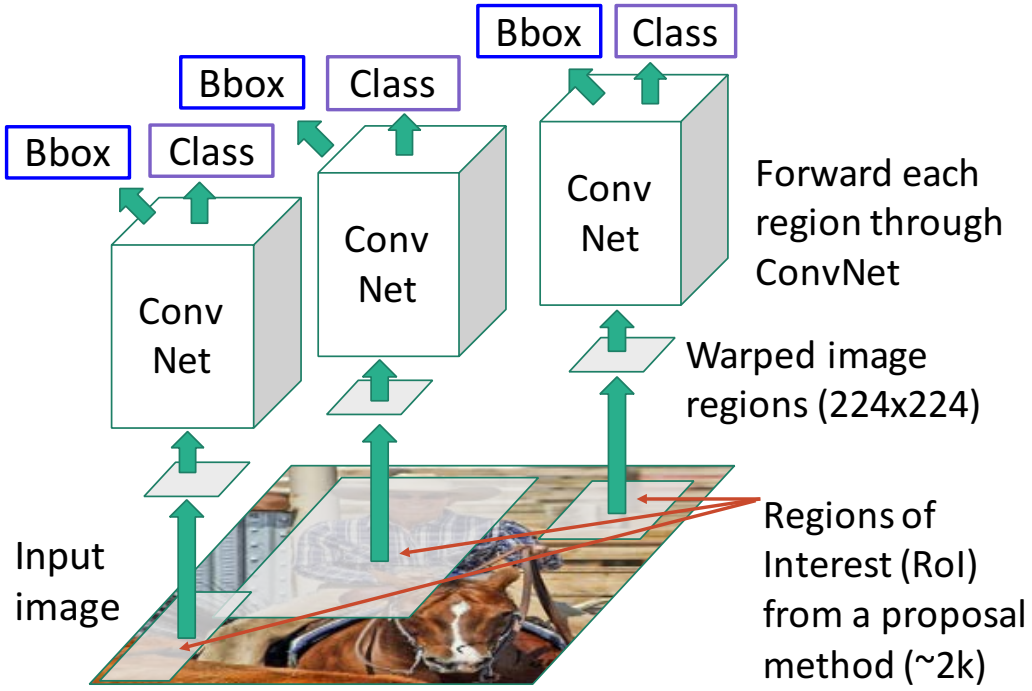
Multiple Objects

Object Detection: Impact of Deep Learning

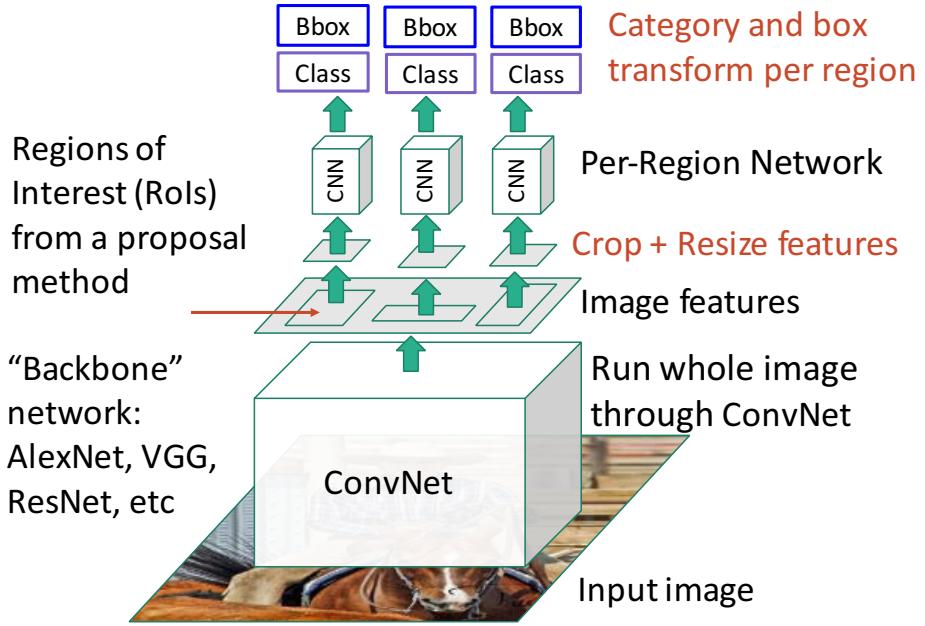


Last Time: Object Detection Methods

"Slow" R-CNN: Run CNN independently for each region

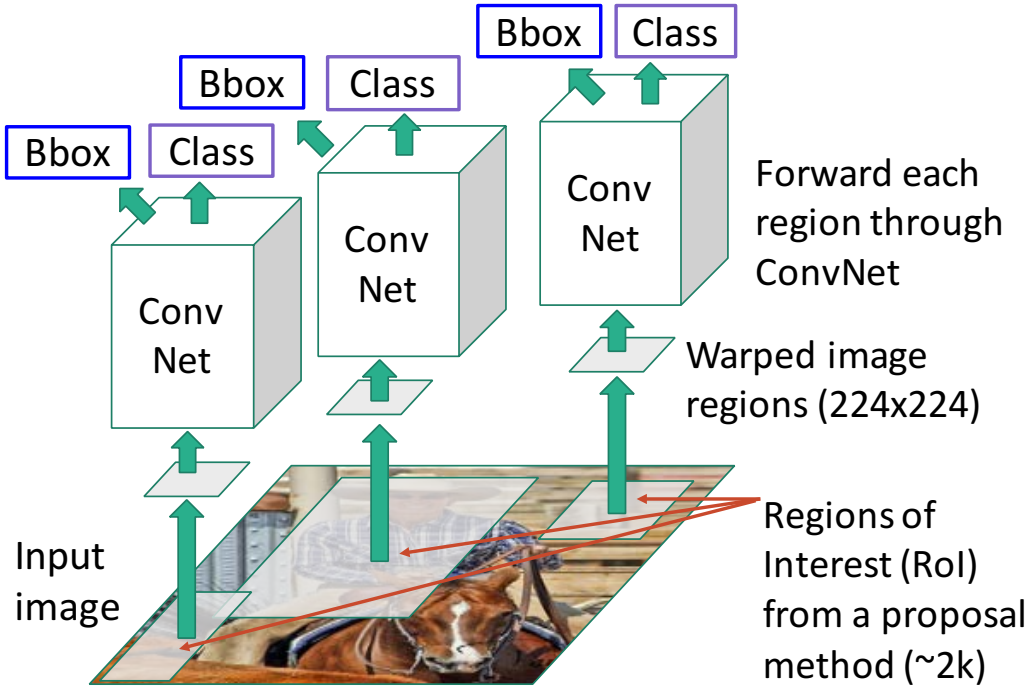


Fast R-CNN: Apply differentiable cropping to shared image features

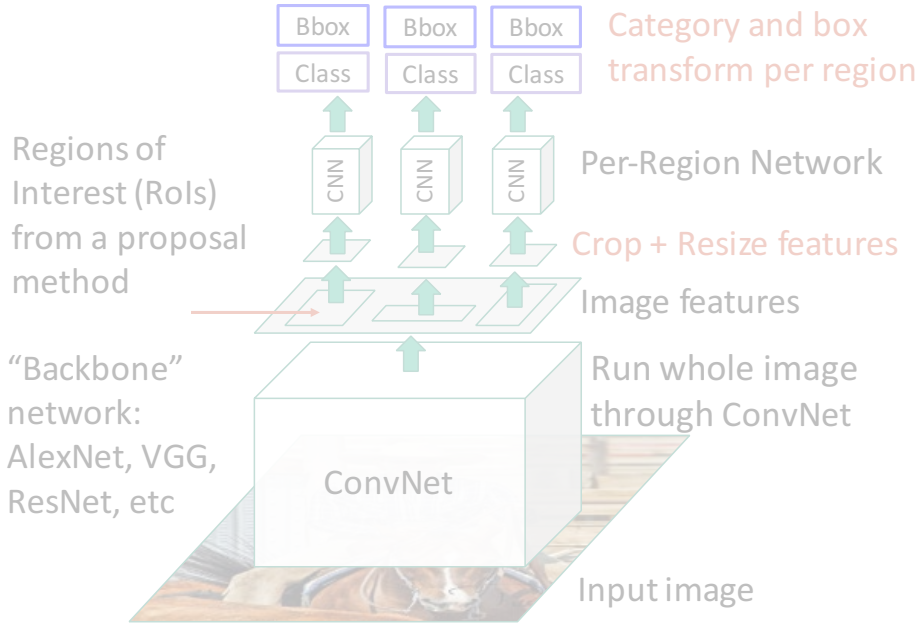


Recap: Slow R-CNN Training

"Slow" R-CNN: Run CNN independently for each region

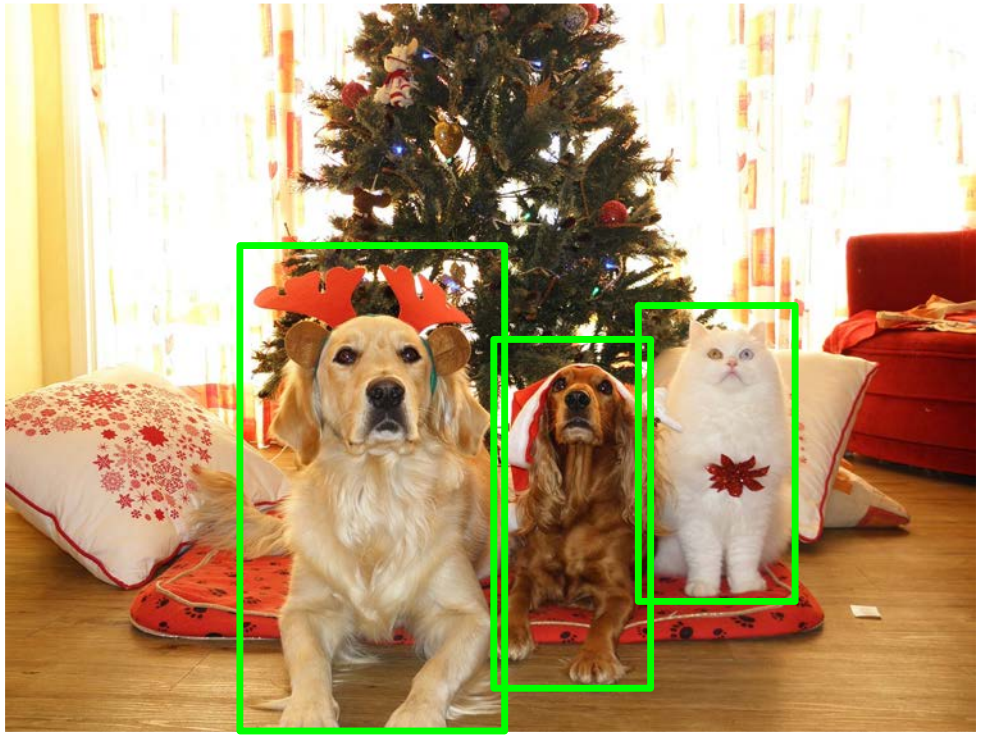


Fast R-CNN: Apply differentiable cropping to shared image features



”Slow” R-CNN Training

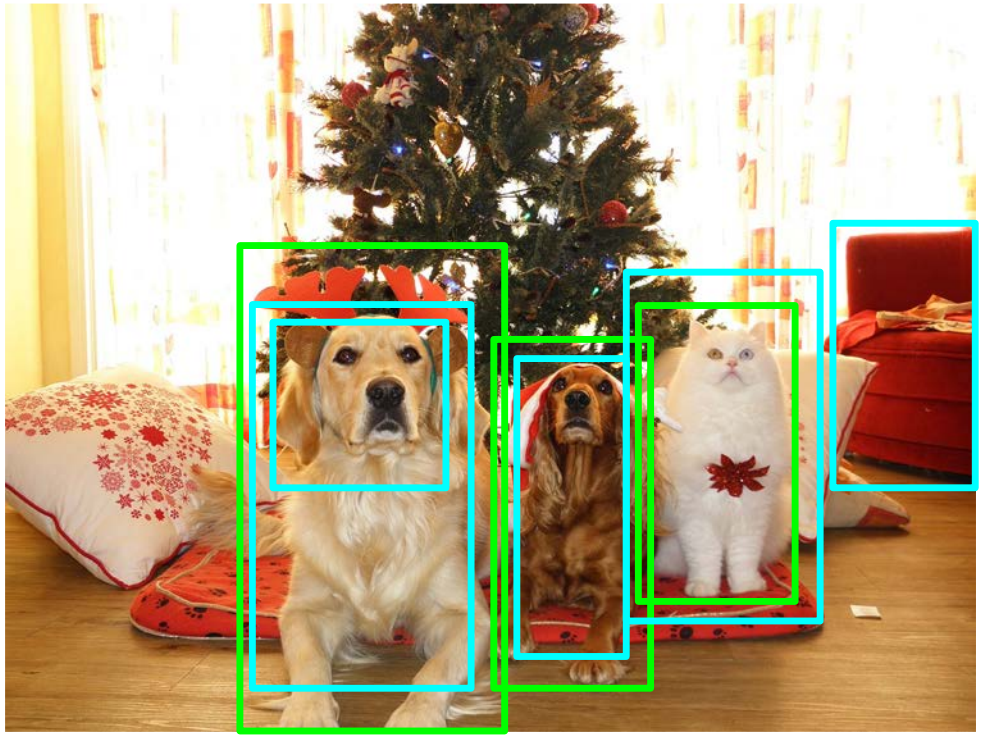
Input Image



Ground-Truth boxes

”Slow” R-CNN Training

Input Image

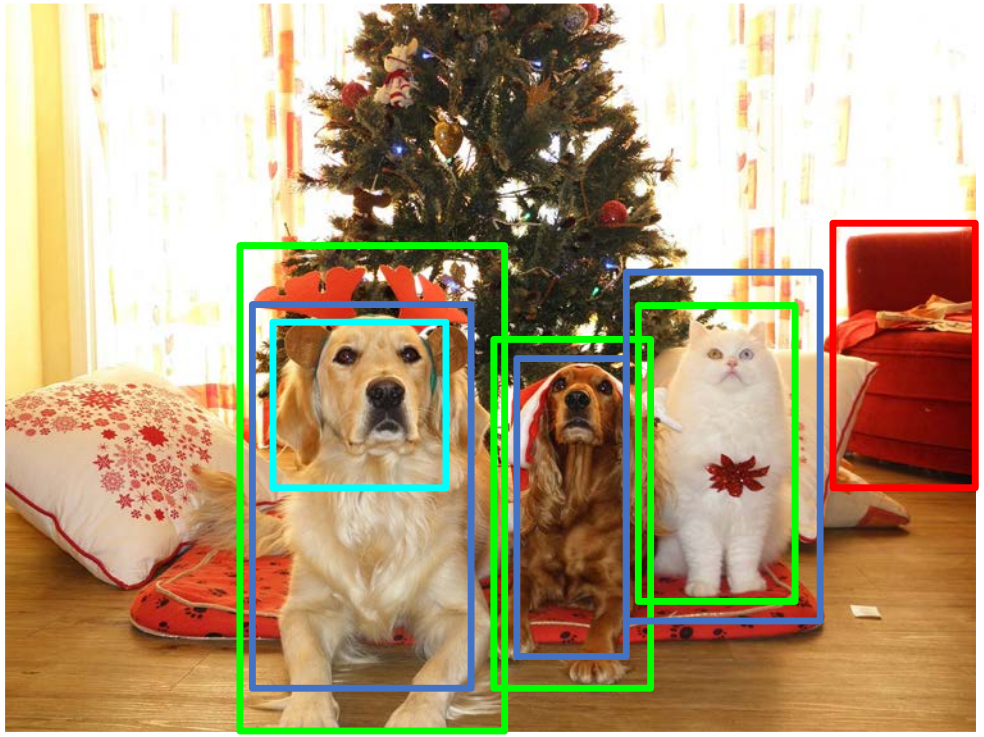


Ground-Truth boxes

Region Proposals

”Slow” R-CNN Training

Input Image



GT Boxes

Positive

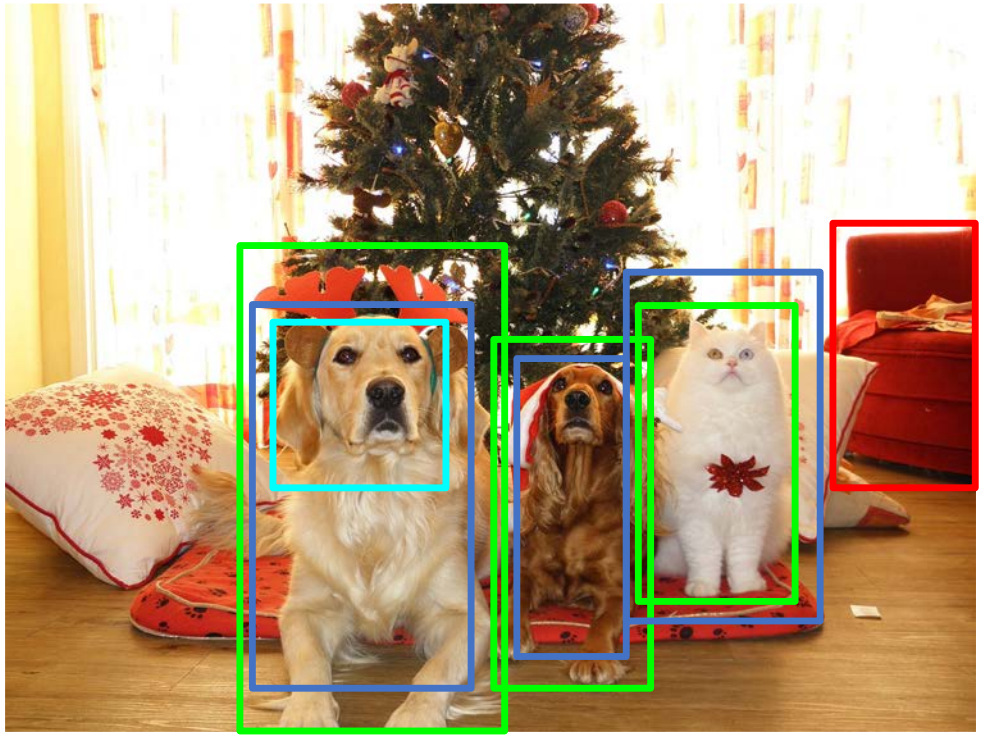
Neutral

Negative

Categorize each region proposal as positive, negative, or neutral based on overlap with ground-truth boxes

”Slow” R-CNN Training

Input Image



GT Boxes

Positive

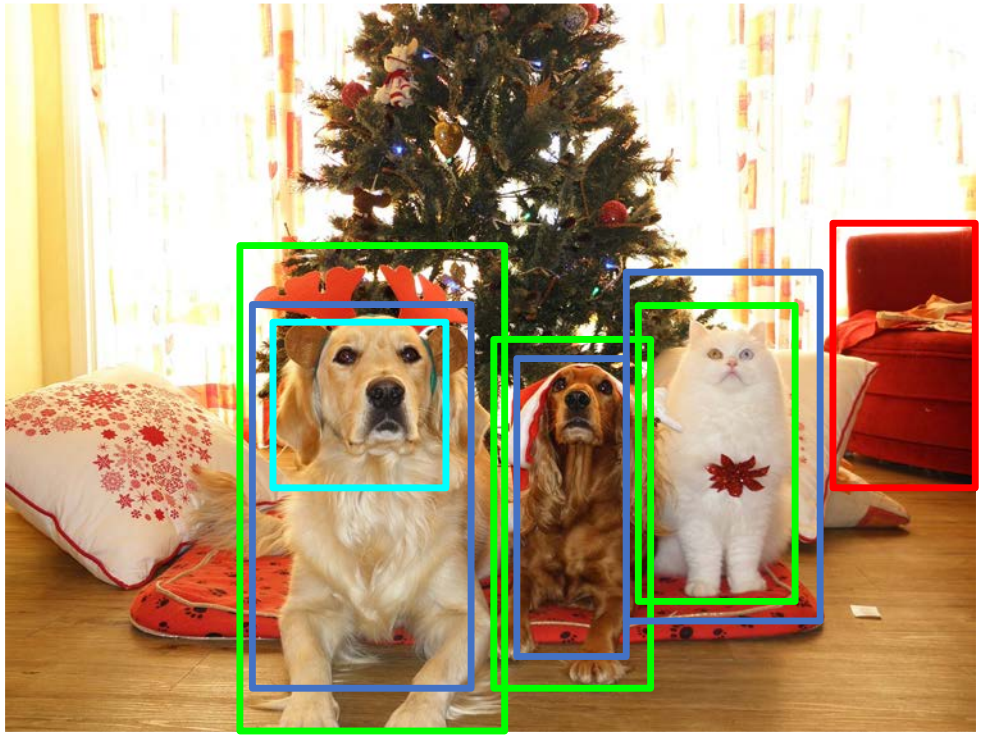
Neutral

Negative

Crop pixels from each positive and negative proposal, resize to 224 x 224

”Slow” R-CNN Training

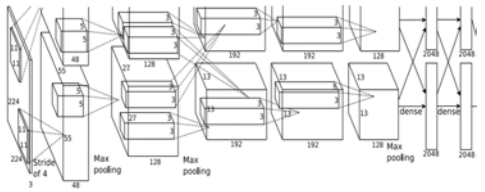
Input Image



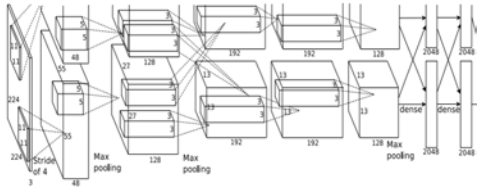
GT Boxes | Positive

Neutral | Negative

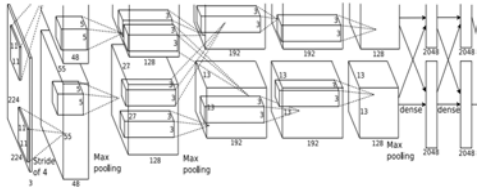
Run each region through CNN. For positive boxes predict class and box offset; for negative boxes just predict background class



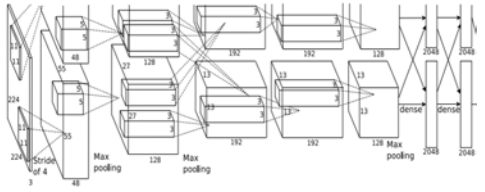
Class target: Dog
Box target: →



Class target: Cat
Box target: →



Class target: Dog
Box target: →

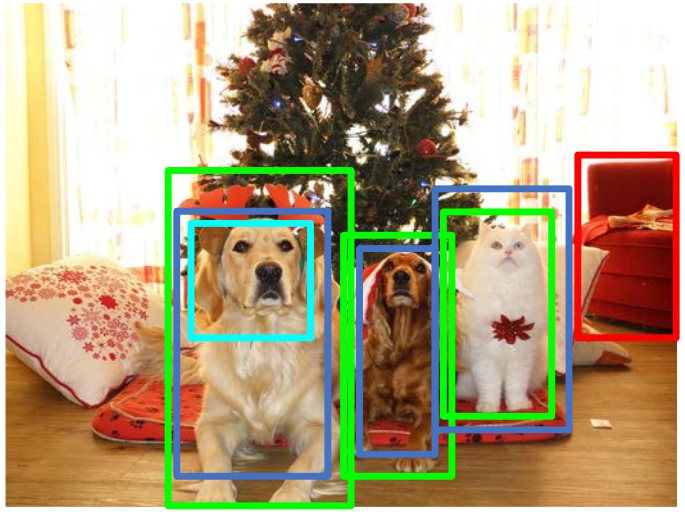


Class target: Background
Box target: None

Fast R-CNN Training

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

Input Image



GT Boxes | Positive

Neutral | Negative

Backbone CNN

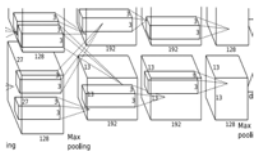
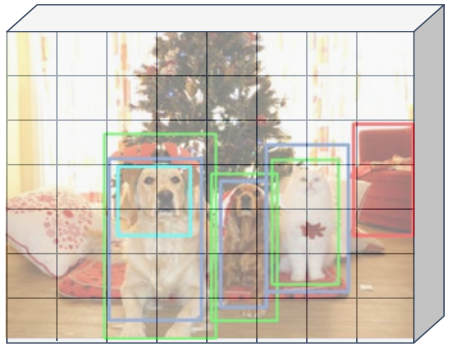
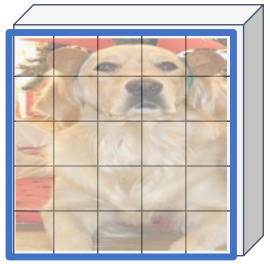


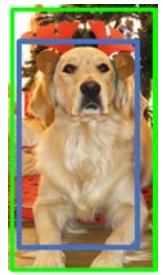
Image Features

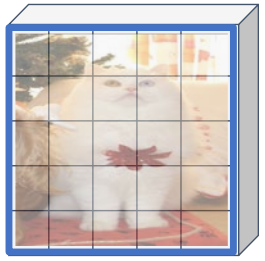




Class target: Dog

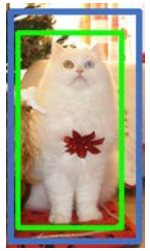
Box target:

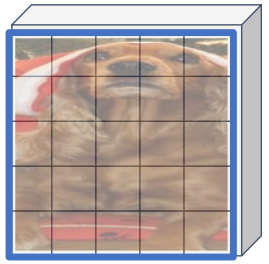




Class target: Cat

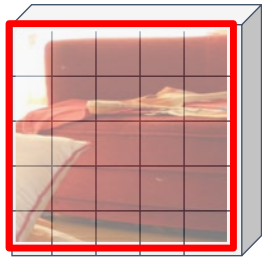
Box target:





Class target: Dog

Box target:



Class target: Background

Box target: None

Faster R-CNN: Learnable Region Proposals

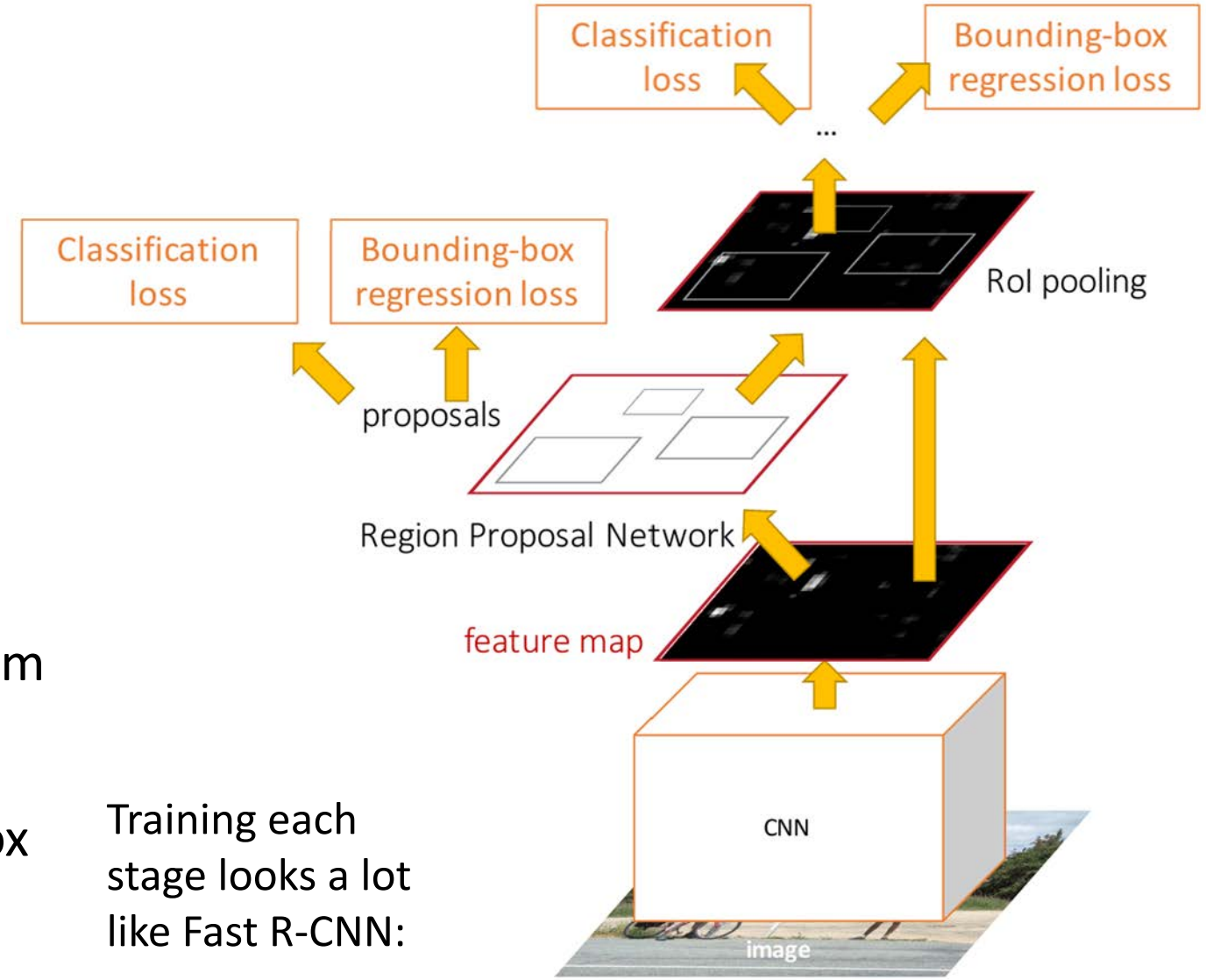
Jointly train with 4 losses:

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)

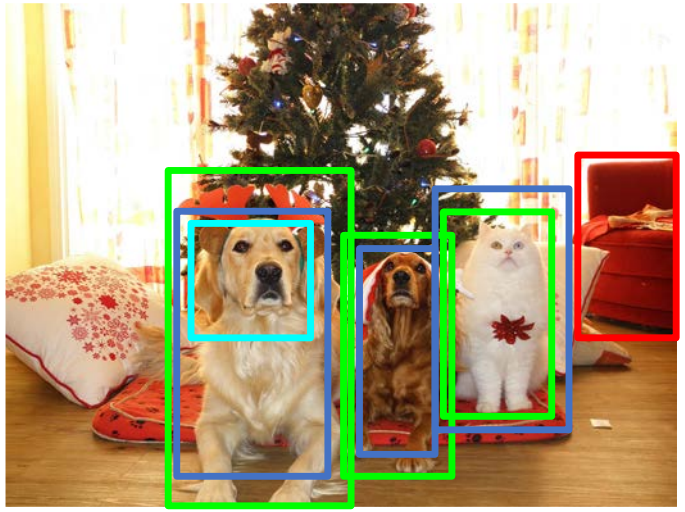
Training each stage looks a lot like Fast R-CNN:



Faster R-CNN Training: RPN Training

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

Input Image



Backbone CNN

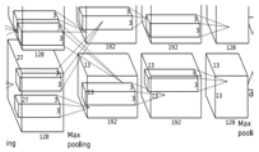
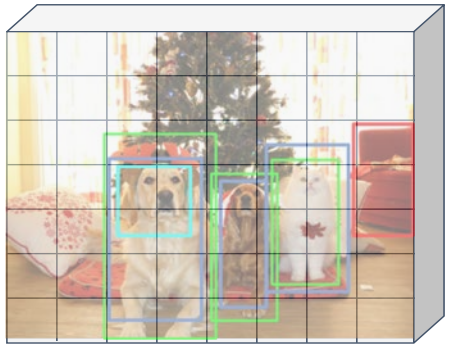
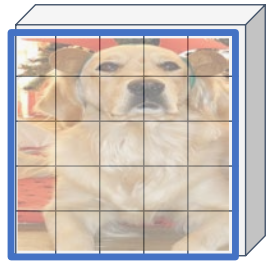


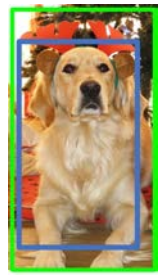
Image Features

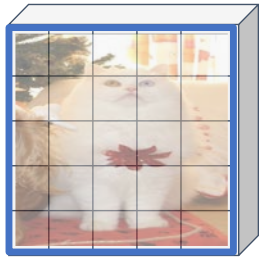


RPN gives lots of **anchors** which we classify as pos / neg / neutral by matching with ground-truth

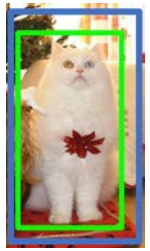


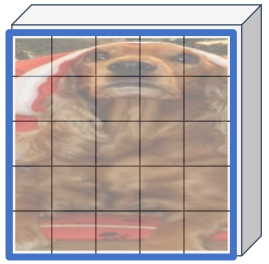
Class target: Obj
Box target: →



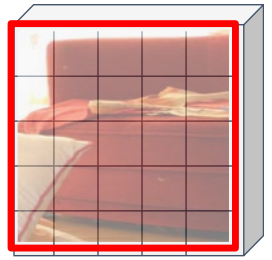


Class target: Obj
Box target: →





Class target: Obj
Box target: →



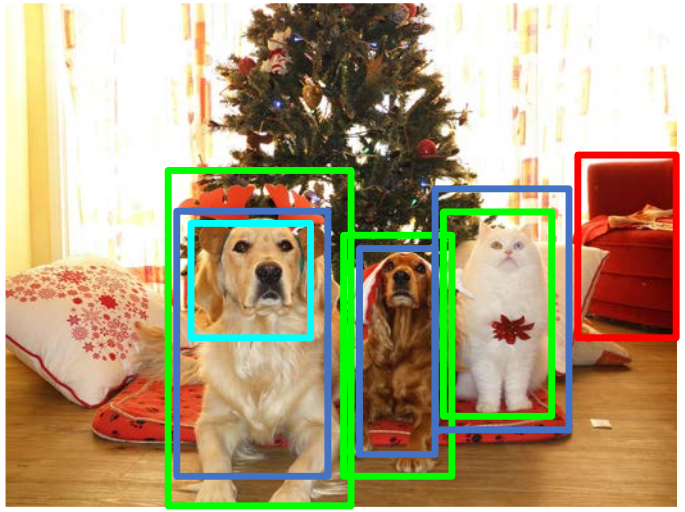
Class target: Background
Box target: None

GT Boxes | Positive | Neutral | Negative

Faster R-CNN Training: Stage 2

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

Input Image



GT Boxes | Positive

Neutral | Negative

Backbone CNN

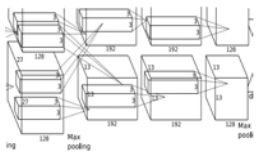
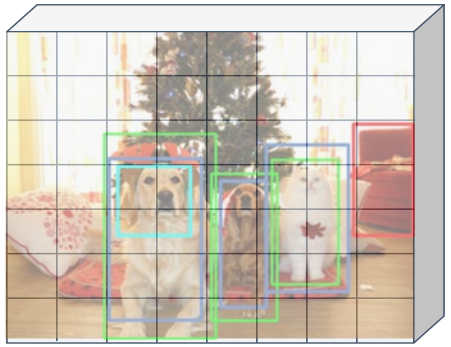
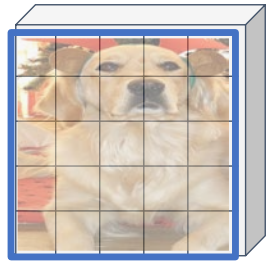


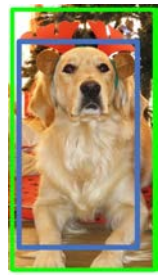
Image Features

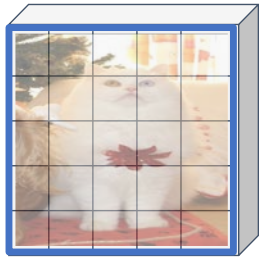


Now proposals come from RPN rather than selective search, but otherwise this works the same as Fast R-CNN training

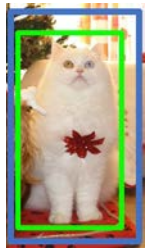


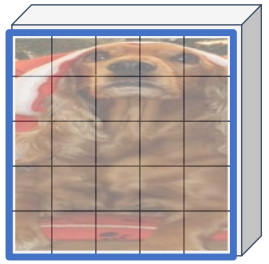
Class target: Dog
Box target:



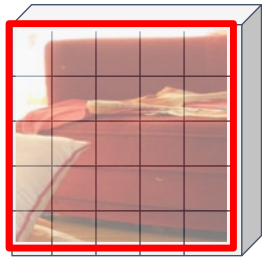


Class target: Cat
Box target:





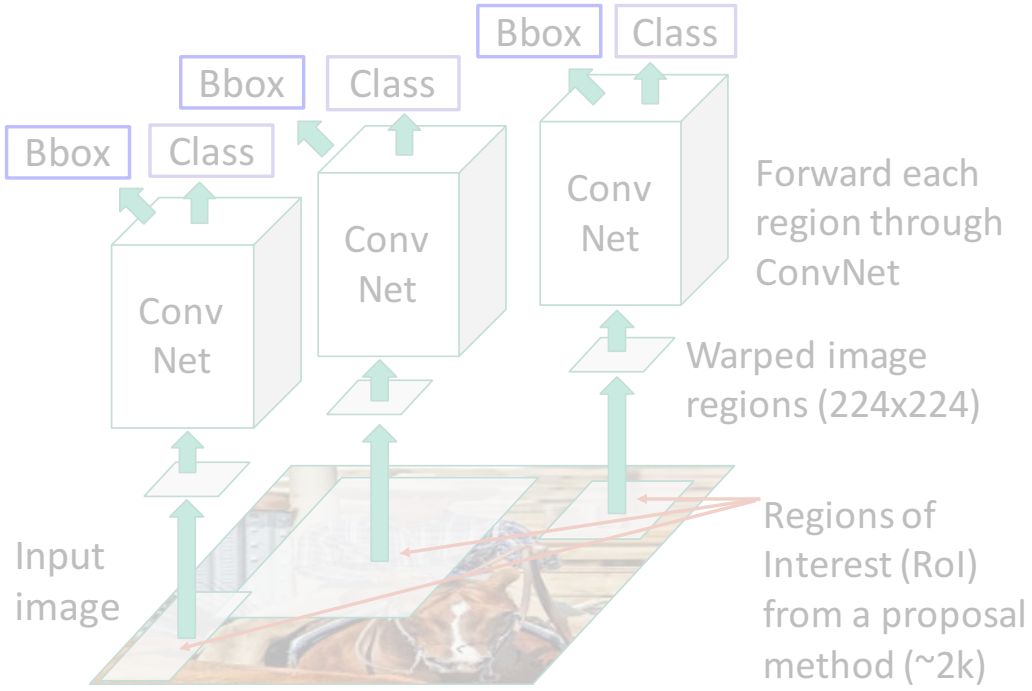
Class target: Dog
Box target:



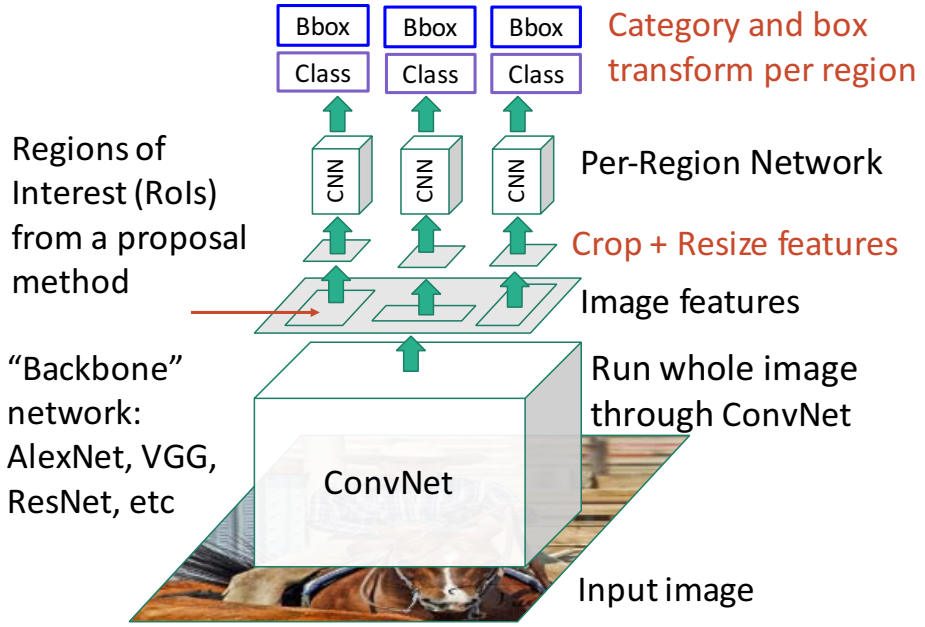
Class target: Background
Box target: None

Recap: Fast R-CNN Feature Cropping

"Slow" R-CNN: Run CNN independently for each region

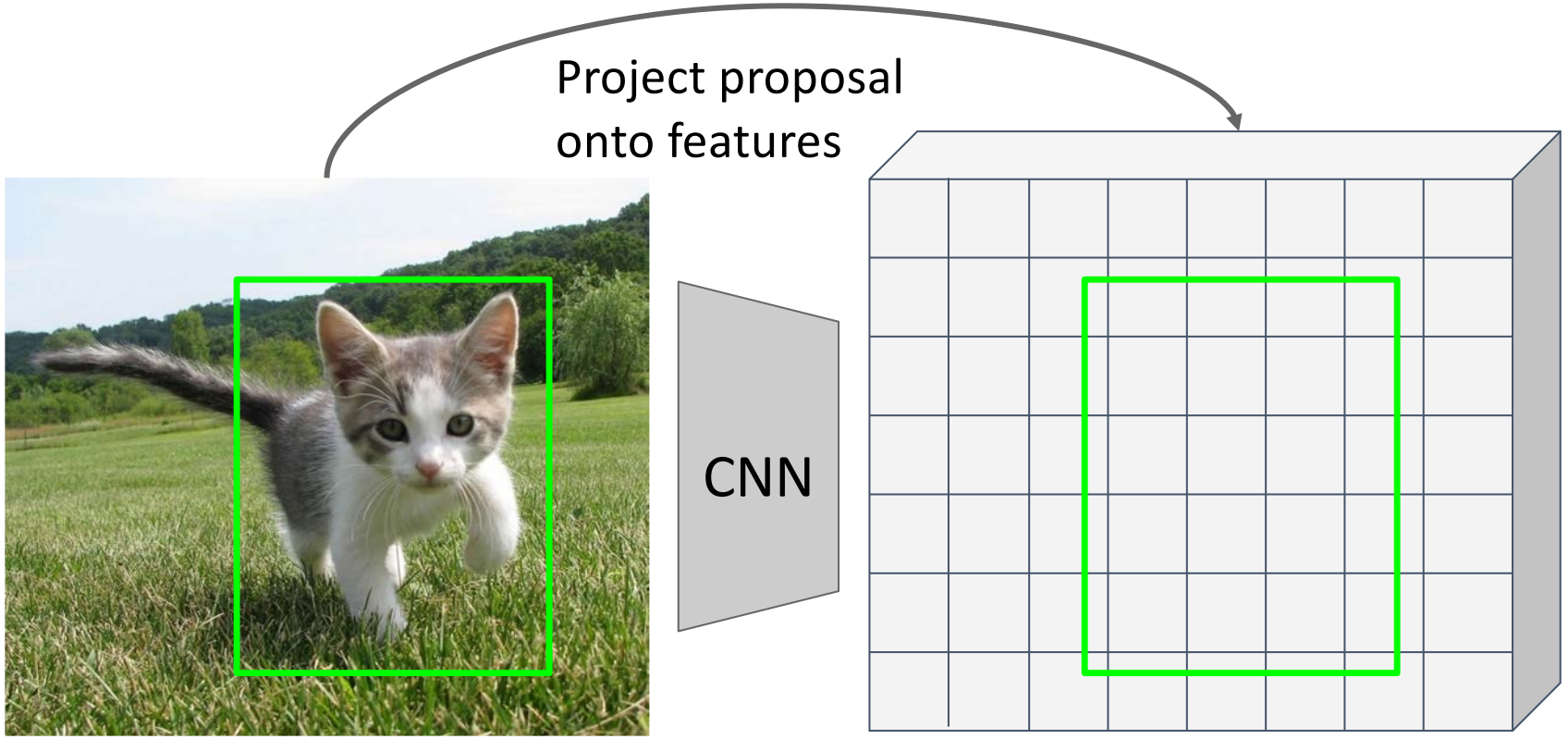


Fast R-CNN: Apply differentiable cropping to shared image features



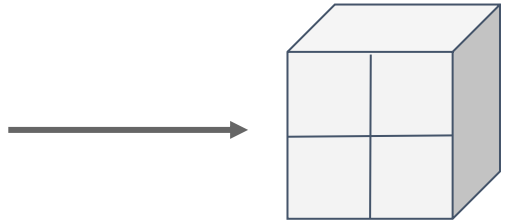
Cropping Features

Goal: Crop features for region proposal, and resize to a fixed size for downstream processing, in a differentiable way



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

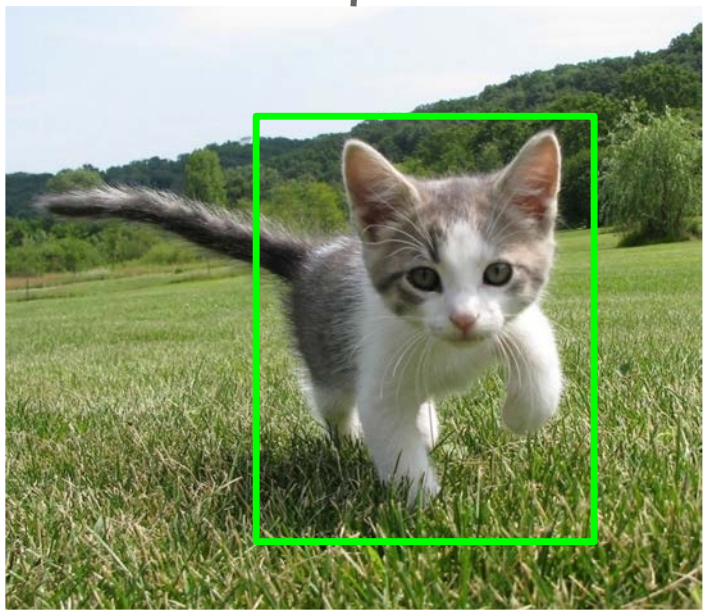
Image features
(e.g. 512 x 20 x 15)



Region features
(here 512 x 2 x 2;
In practice e.g 512 x 7 x 7)

Girshick, "Fast R-CNN", ICCV 2015.

Cropping Features: RoI Pool



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

Project proposal onto features

"Snap" to grid cells

CNN

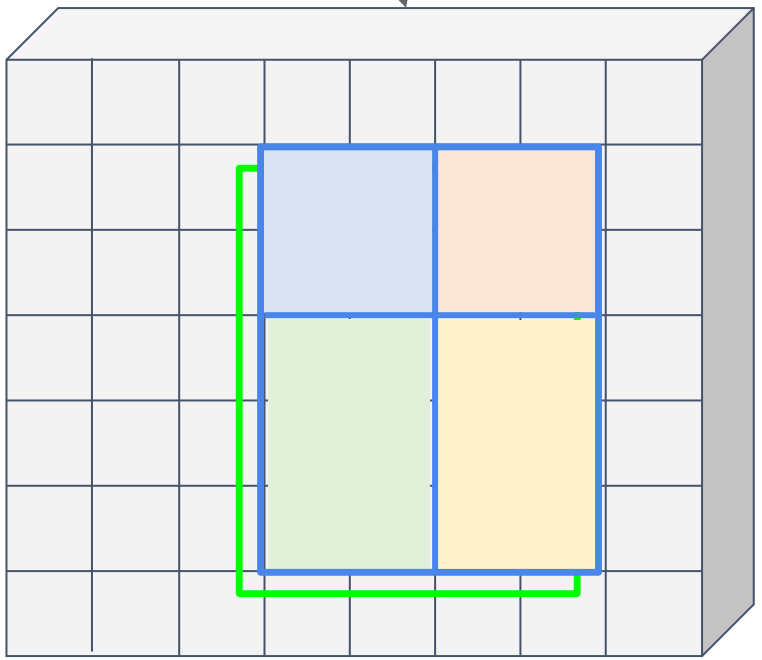
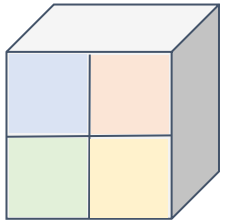


Image features
(e.g. 512 x 20 x 15)

Divide into 2x2 grid of (roughly) equal subregions

Max-pool within each subregion

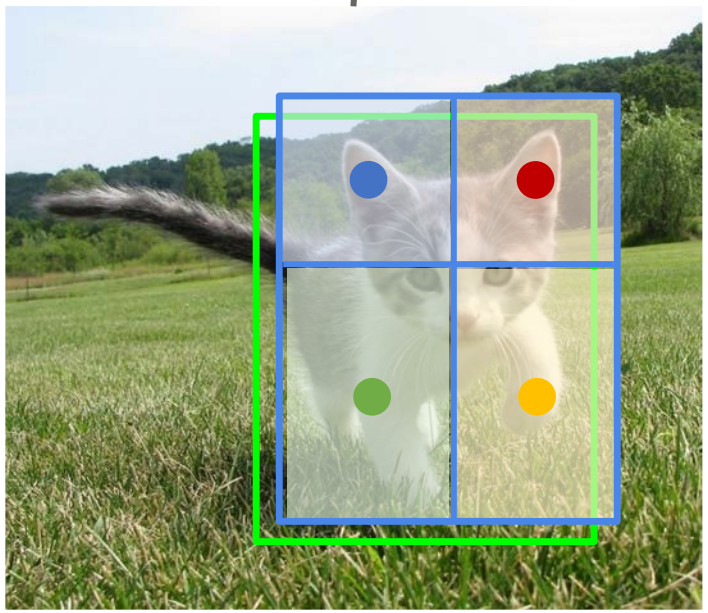


Region features
(here 512 x 2 x 2;
In practice e.g 512 x 7 x 7)

Region features always the same size even if input regions have different sizes!

Girshick, "Fast R-CNN", ICCV 2015.

Cropping Features: RoI Pool



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

Project proposal onto features

"Snap" to grid cells

CNN

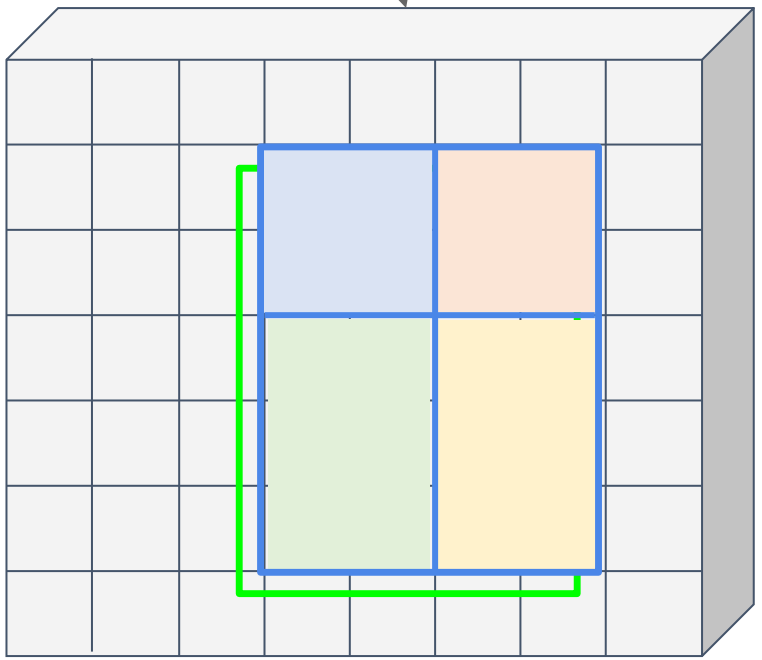
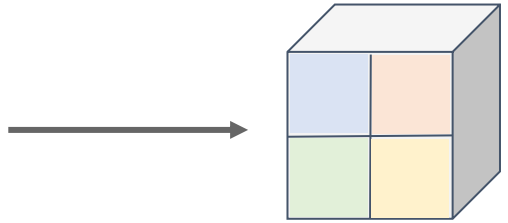


Image features
(e.g. 512 x 20 x 15)

Divide into 2x2 grid of (roughly) equal subregions

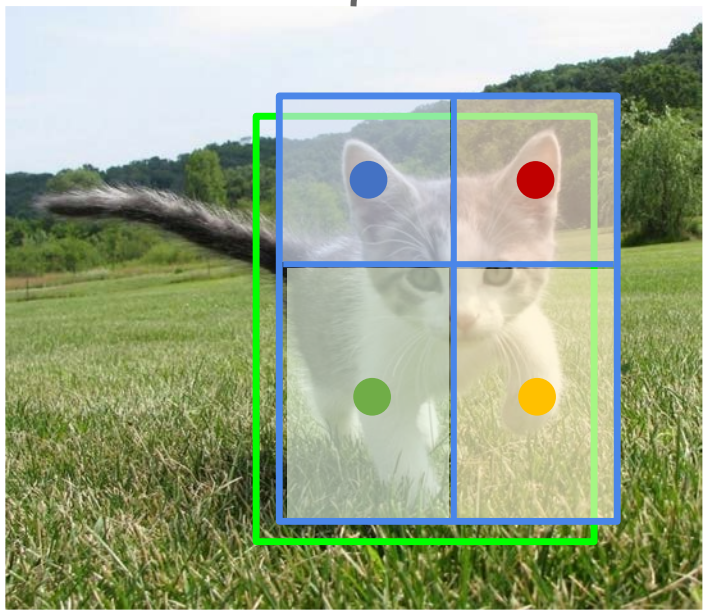
Max-pool within each subregion



Region features
(here 512 x 2 x 2;
In practice e.g 512 x 7 x 7)

Problem #1: Misaligned features due to snapping

Cropping Features: RoI Pool



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

Project proposal onto features

"Snap" to grid cells

CNN

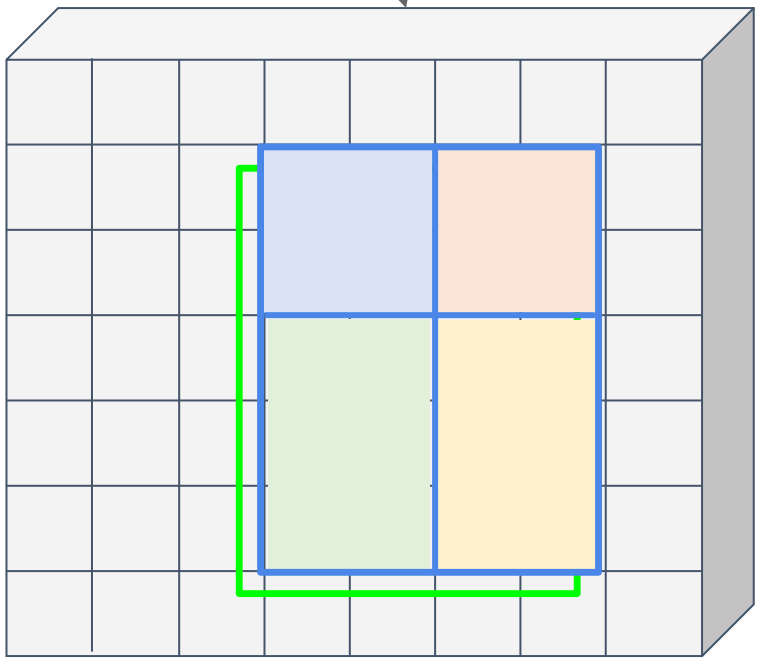
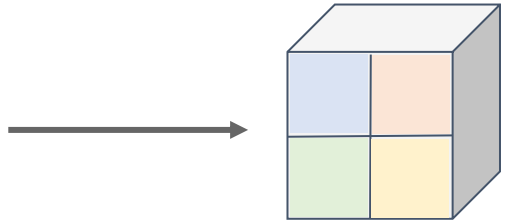


Image features
(e.g. 512 x 20 x 15)

Divide into 2x2 grid of (roughly) equal subregions

Max-pool within each subregion



Region features
(here 512 x 2 x 2;
In practice e.g 512 x 7 x 7)

Problem #1: Misaligned features due to snapping

Problem #2: Can't backprop to box coordinates

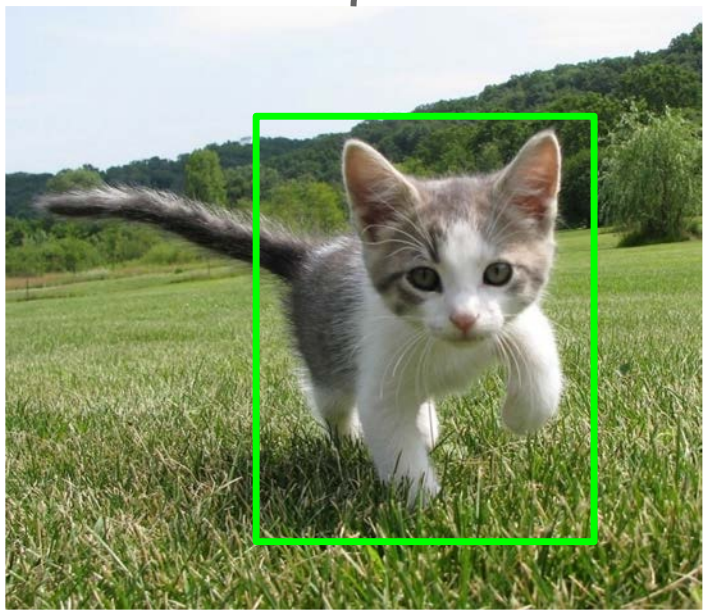
Region Features = f(image features, box coordinates)

Cropping Features: RoI Align

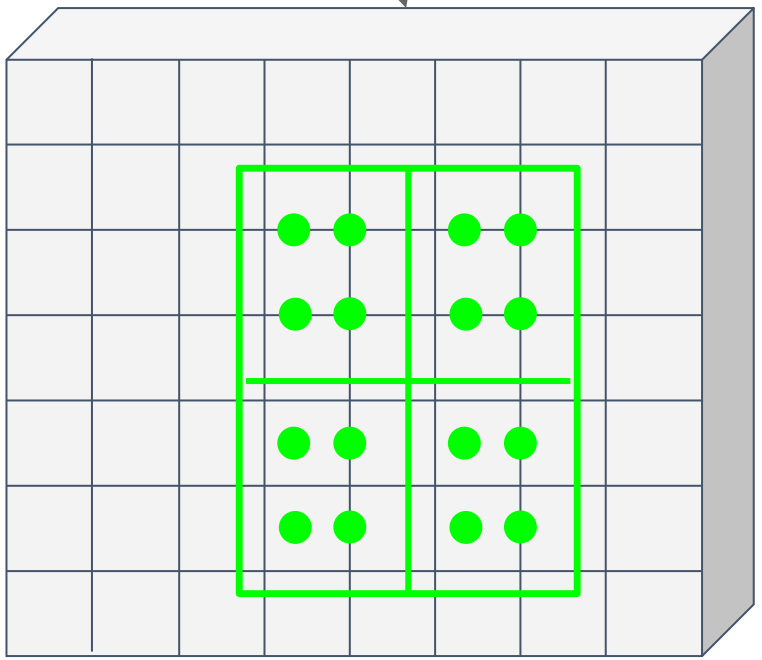
Divide into equal-sized subregions (may not be aligned to grid!)

No "snapping"!

Project proposal onto features



CNN



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

Image features
(e.g. 512 x 20 x 15)

Sample features at regularly-spaced points in each subregion using **bilinear interpolation**

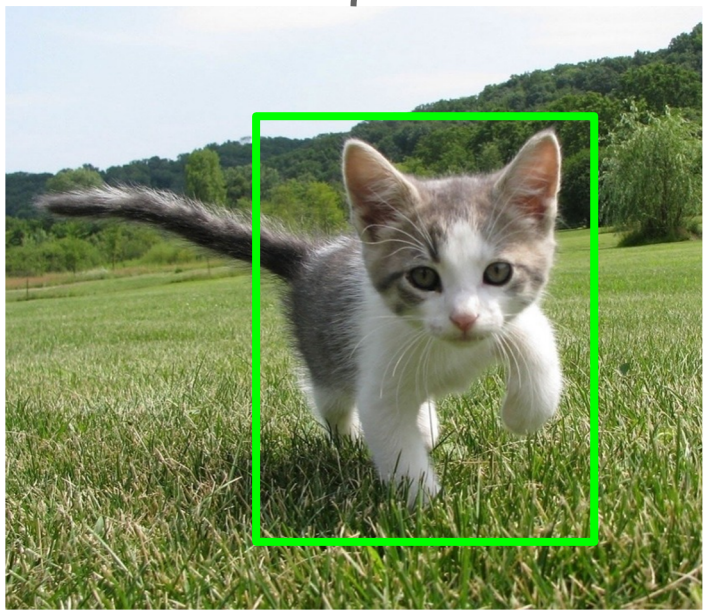
He et al, "Mask R-CNN", ICCV 2017

Cropping Features: RoI Align

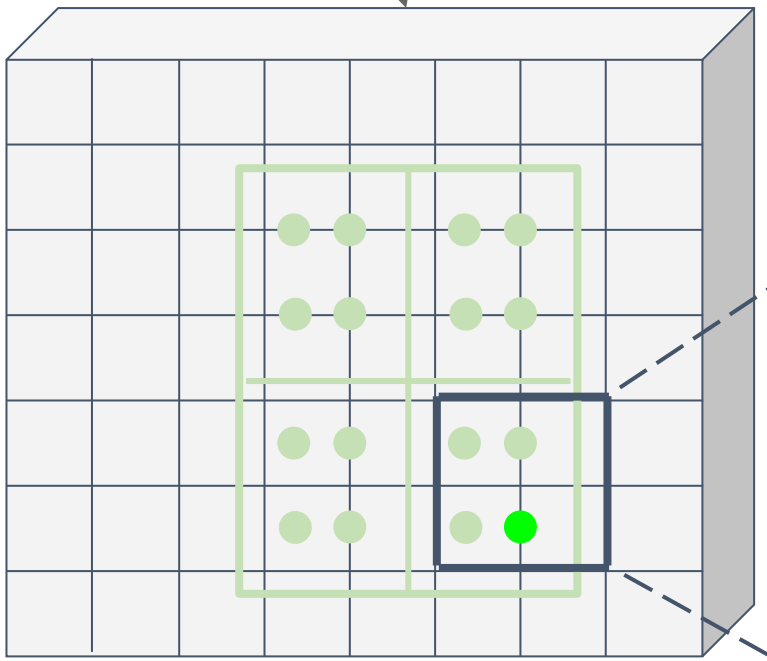
Divide into equal-sized subregions (may not be aligned to grid!)

No "snapping"!

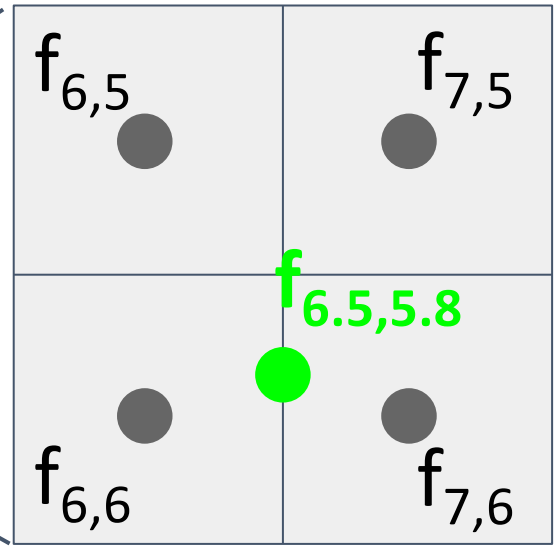
Project proposal onto features



CNN



Sample features at regularly-spaced points in each subregion using **bilinear interpolation**



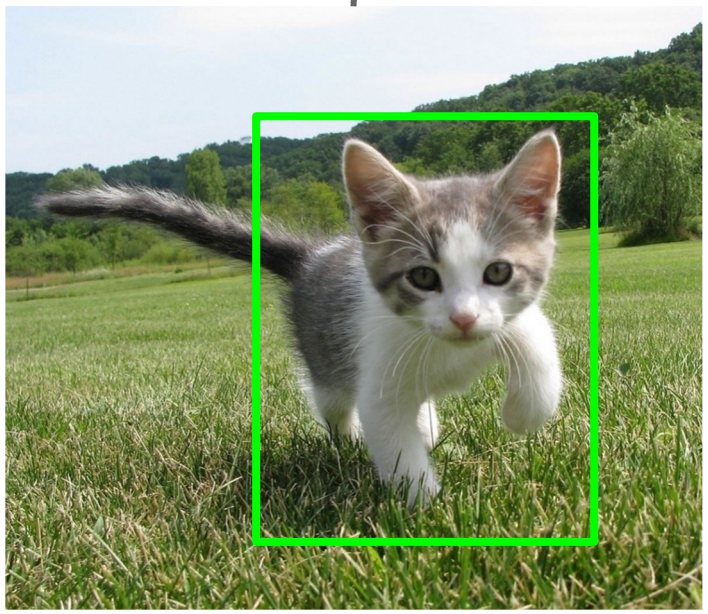
$$f_{x,y} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - i|) \max(0, 1 - |y - j|)$$

$$i \in \{\lfloor x \rfloor - 1, \ldots, \lceil x \rceil + 1\}$$

$$j \in \{\lfloor y \rfloor - 1, \ldots, \lceil y \rceil + 1\}$$

Feature f_{xy} for point (x, y) is a linear combination of features at its four neighboring grid cells:

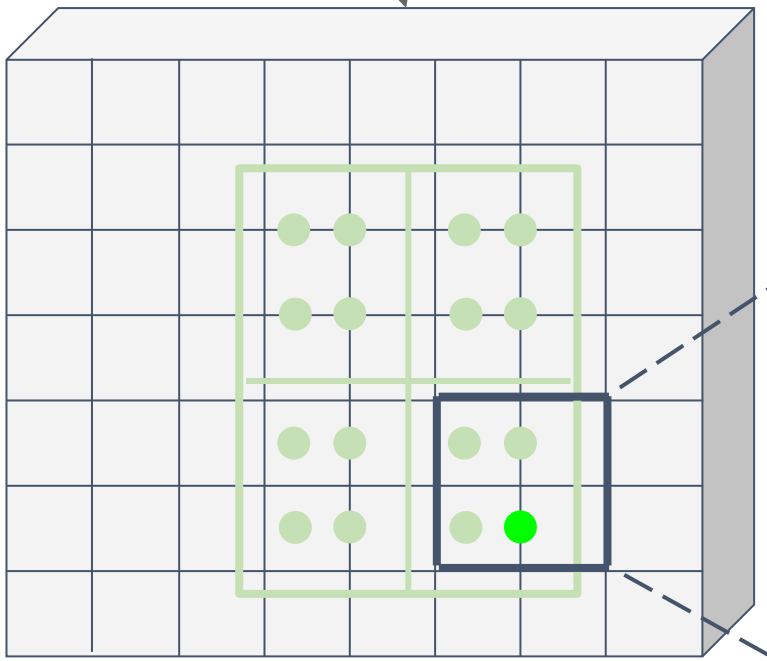
Cropping Features: RoI Align



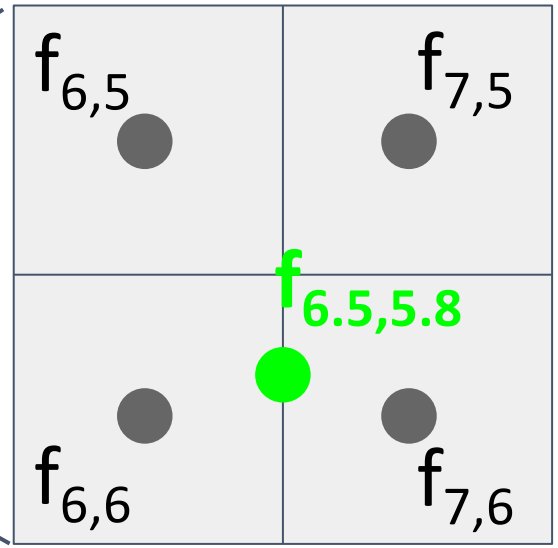
Project proposal onto features

No “snapping”!

CNN



Sample features at regularly-spaced points in each subregion using **bilinear interpolation**



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

$f_{6.5,5.8}$ = ($f_{6,5}$ * 0.5 * 0.2) + ($f_{7,5}$ * 0.5 * 0.2)
+ ($f_{6,6}$ * 0.5 * 0.8) + ($f_{7,6}$ * 0.5 * 0.8)

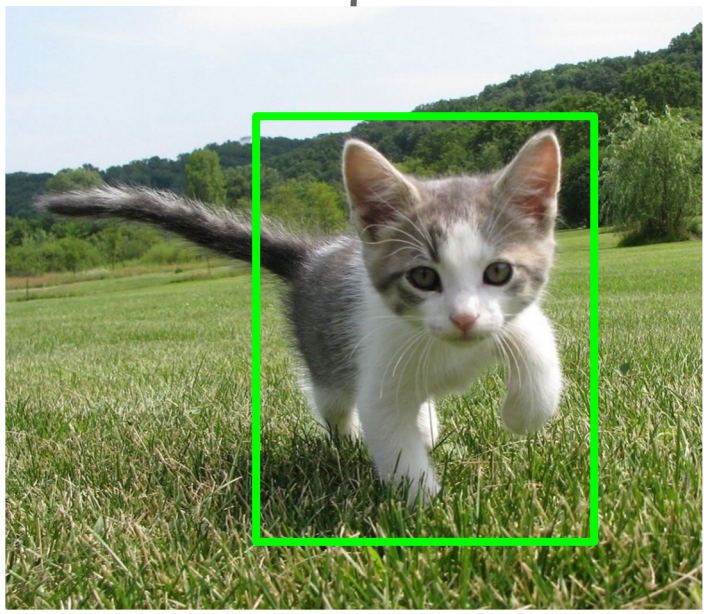
Feature f_{xy} for point (x, y) is a linear combination of features at its four neighboring grid cells:

Cropping Features: RoI Align

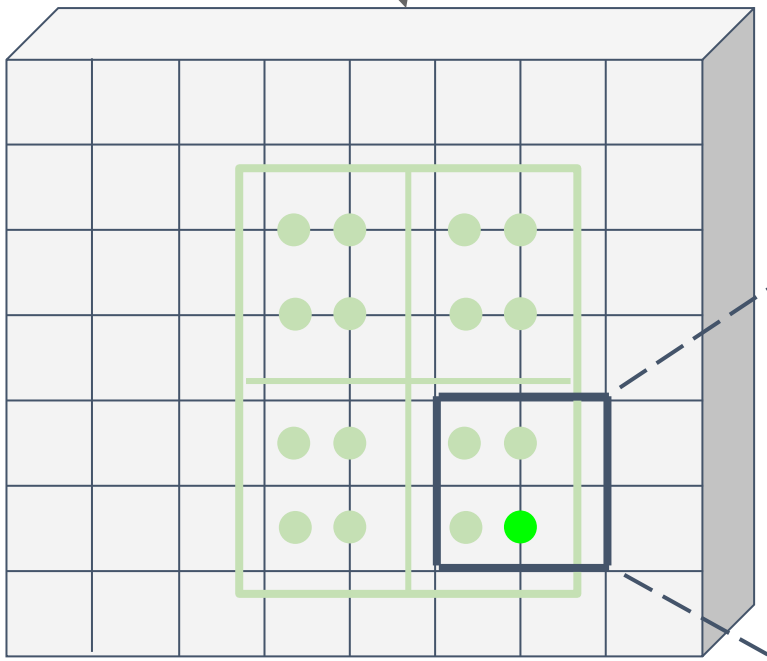
Divide into equal-sized subregions (may not be aligned to grid!)

No "snapping"!

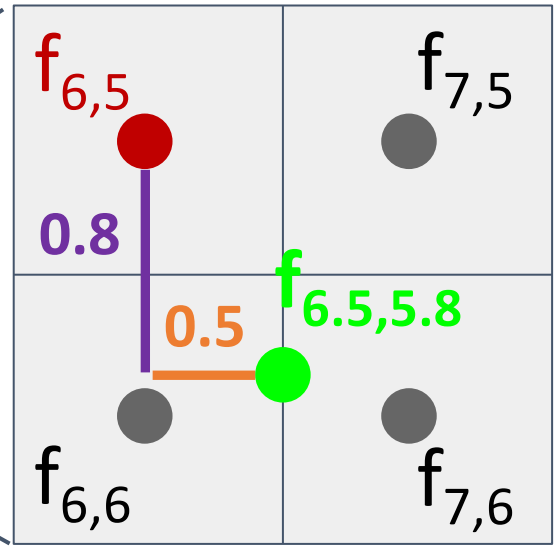
Project proposal onto features



CNN



Sample features at regularly-spaced points in each subregion using **bilinear interpolation**

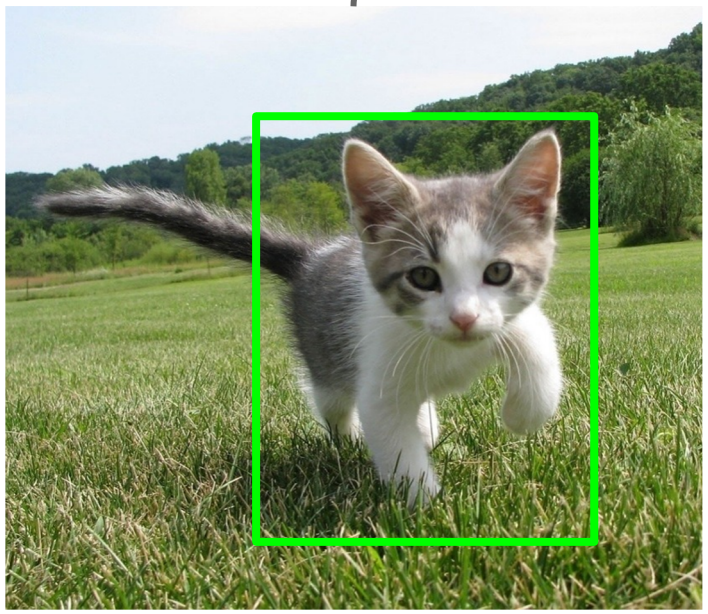


$$f_{x,y} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - i|) \max(0, 1 - |y - j|)$$

$f_{6.5,5.8}$ = ($f_{6,5}$ * 0.5 * 0.2) + ($f_{7,5}$ * 0.5 * 0.2)
+ ($f_{6,6}$ * 0.5 * 0.8) + ($f_{7,6}$ * 0.5 * 0.8)

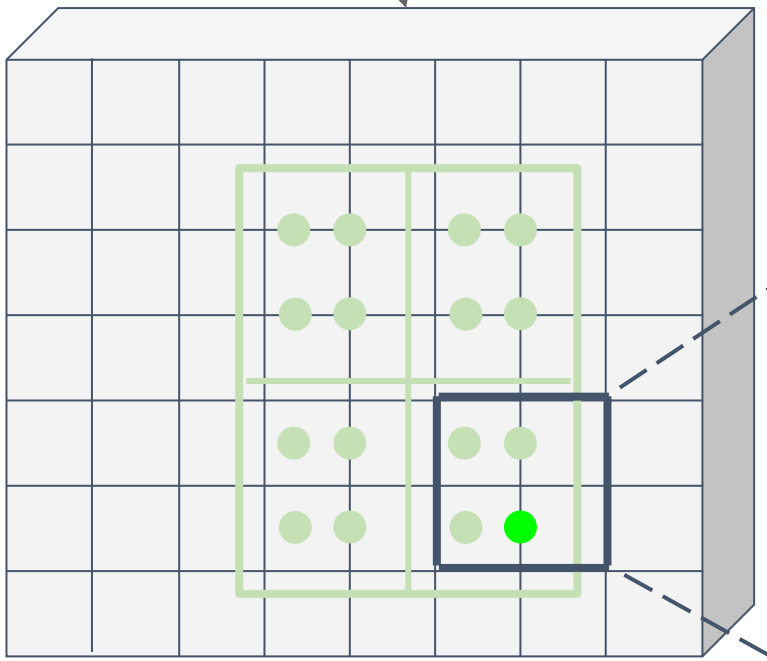
Feature f_{xy} for point (x, y) is a linear combination of features at its four neighboring grid cells:

Cropping Features: RoI Align



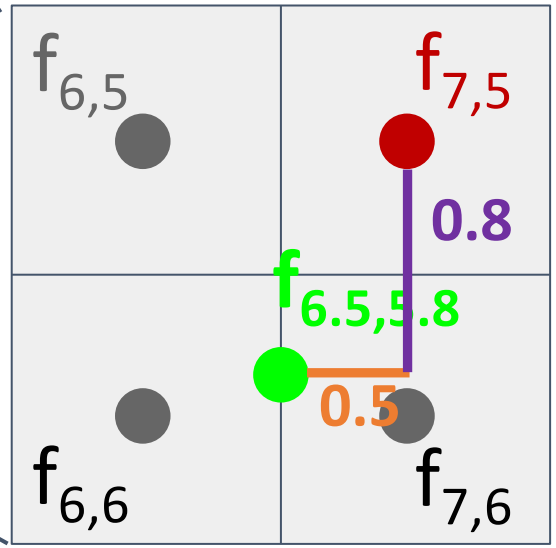
Project proposal onto features

CNN



No “snapping”!

Sample features at regularly-spaced points in each subregion using **bilinear interpolation**



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

$f_{6.5,5.8} = (f_{6,5} * 0.5 * 0.2) + (f_{7,5} * 0.5 * 0.2)$
$+ (f_{6,6} * 0.5 * 0.8) + (f_{7,6} * 0.5 * 0.8)$

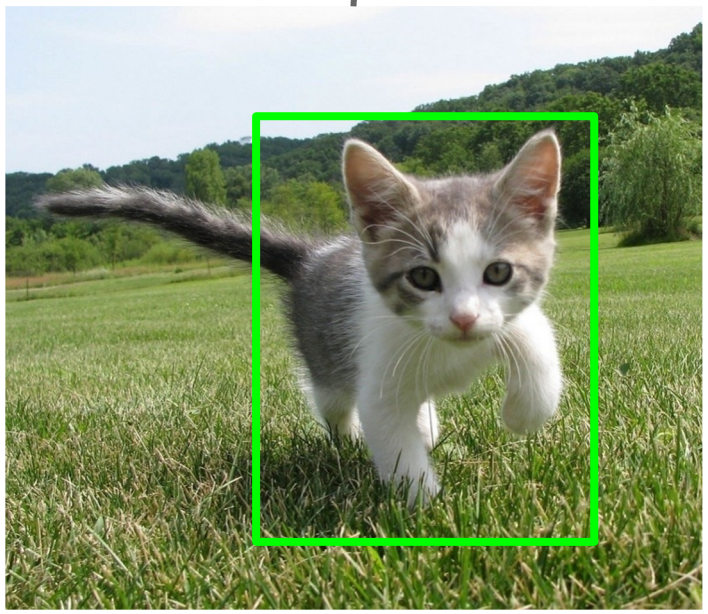
Feature f_{xy} for point (x, y) is a linear combination of features at its four neighboring grid cells:

Cropping Features: RoI Align

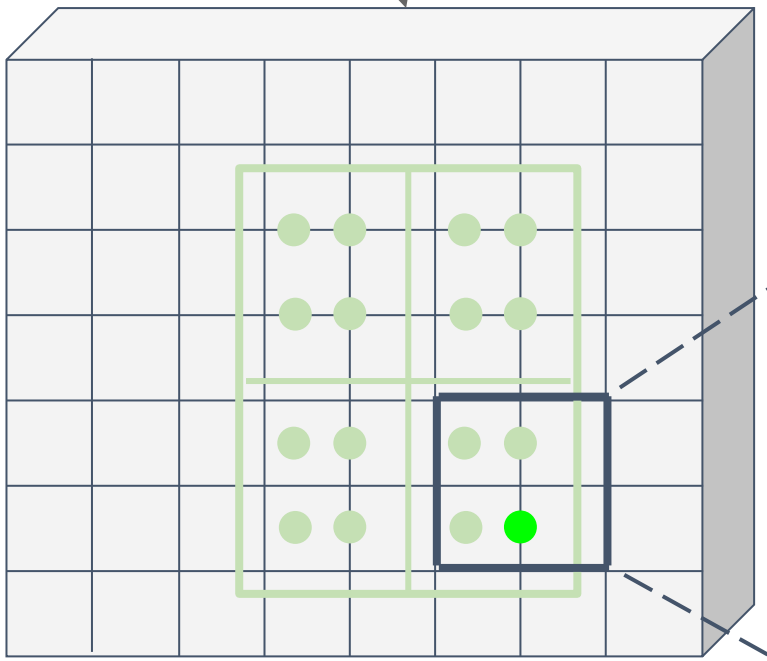
Divide into equal-sized subregions (may not be aligned to grid!)

No "snapping"!

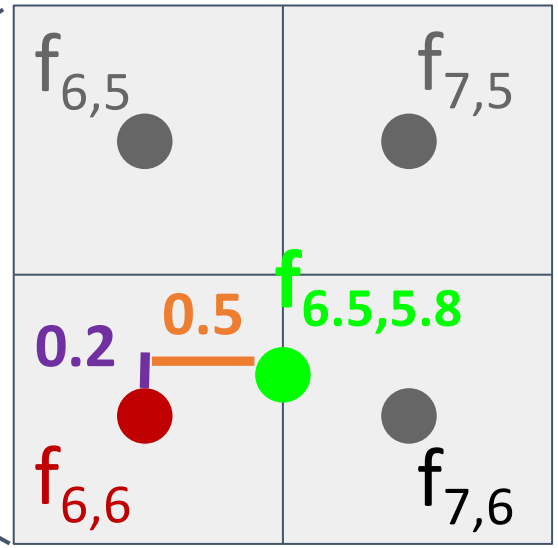
Project proposal onto features



CNN



Sample features at regularly-spaced points in each subregion using **bilinear interpolation**

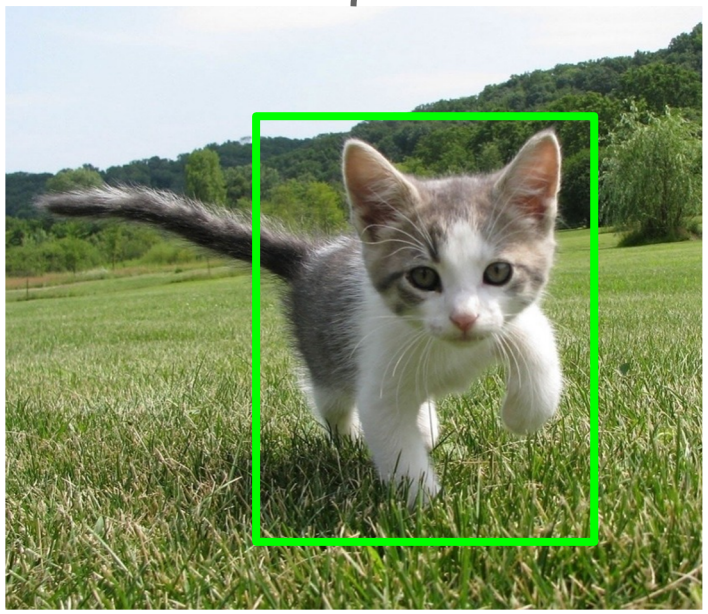


$$f_{x,y} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - i|) \max(0, 1 - |y - j|)$$

$f_{6.5,5.8} = (f_{6,5} * 0.5 * 0.2) + (f_{7,5} * 0.5 * 0.2) + (f_{6,6} * 0.5 * 0.8) + (f_{7,6} * 0.5 * 0.8)$

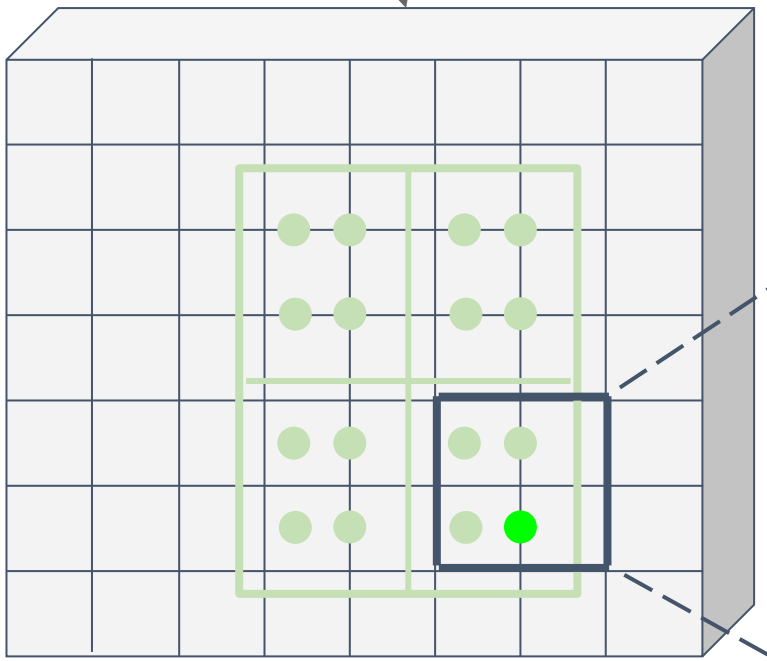
Feature f_{xy} for point (x, y) is a linear combination of features at its four neighboring grid cells:

Cropping Features: RoI Align



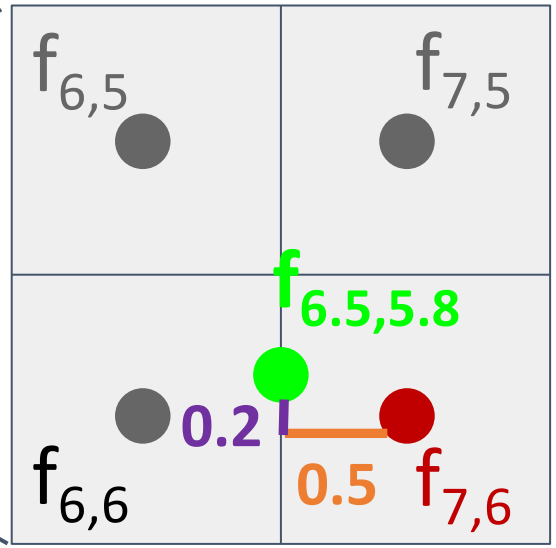
Project proposal onto features

CNN



No “snapping”!

Sample features at regularly-spaced points in each subregion using **bilinear interpolation**



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

$f_{6.5,5.8}$ = ($f_{6,5}$ * 0.5 * 0.2) + ($f_{7,5}$ * 0.5 * 0.2)
+ ($f_{6,6}$ * 0.5 * 0.8) + ($f_{7,6}$ * 0.5 * 0.8)

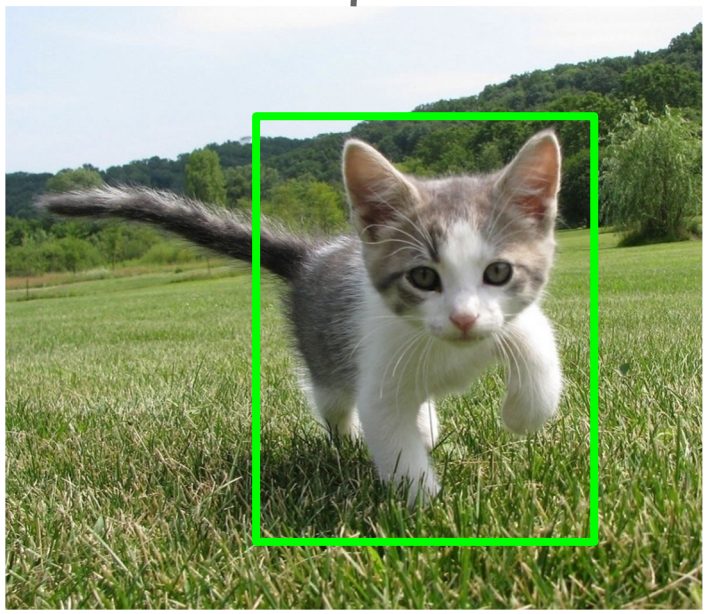
Feature f_{xy} for point (x, y) is a linear combination of features at its four neighboring grid cells:

Cropping Features: RoI Align

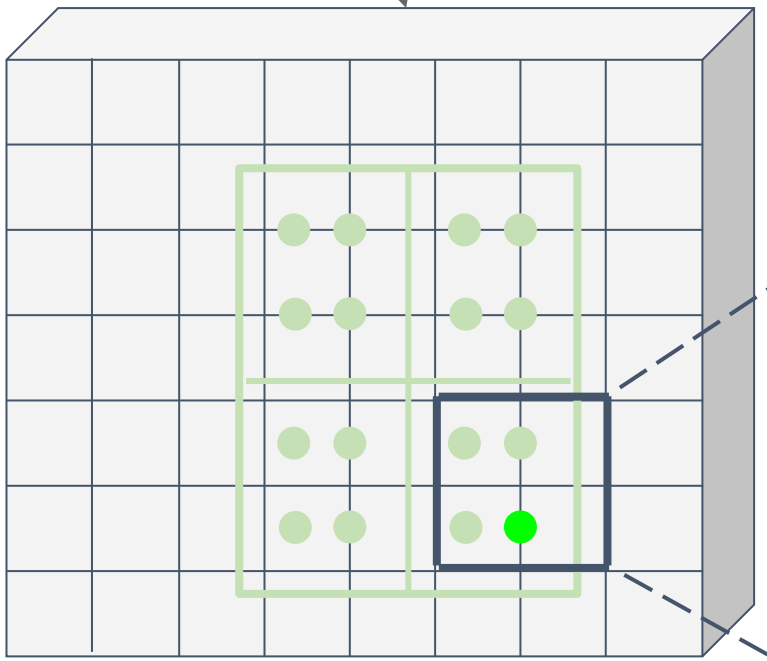
Divide into equal-sized subregions (may not be aligned to grid!)

No "snapping"!

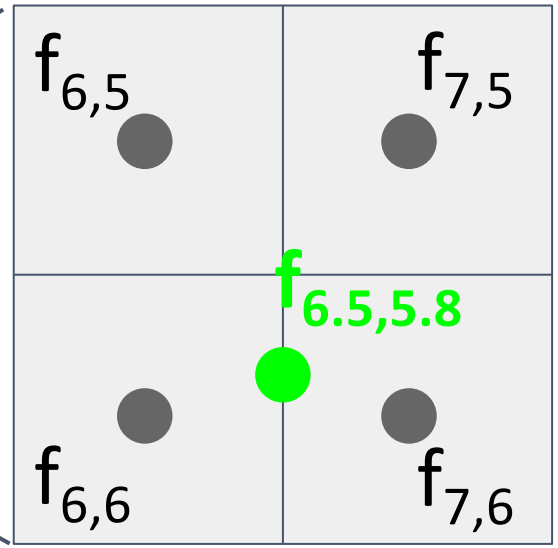
Project proposal onto features



CNN



Sample features at regularly-spaced points in each subregion using **bilinear interpolation**



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

This is differentiable! Upstream gradient for sampled feature will flow backward into each of the four nearest-neighbor gridpoints

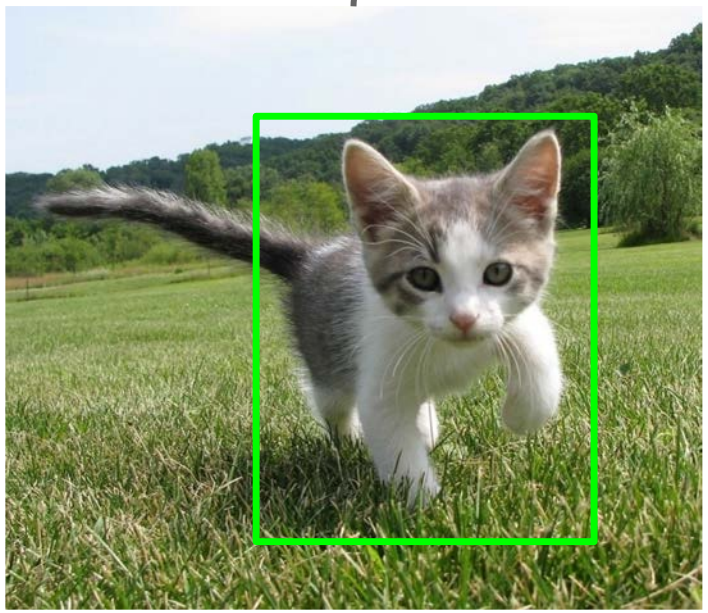
Feature f_{xy} for point (x, y) is a linear combination of features at its four neighboring grid cells:

Cropping Features: RoI Align

Divide into equal-sized subregions (may not be aligned to grid!)

No "snapping"!

Project proposal onto features



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

CNN

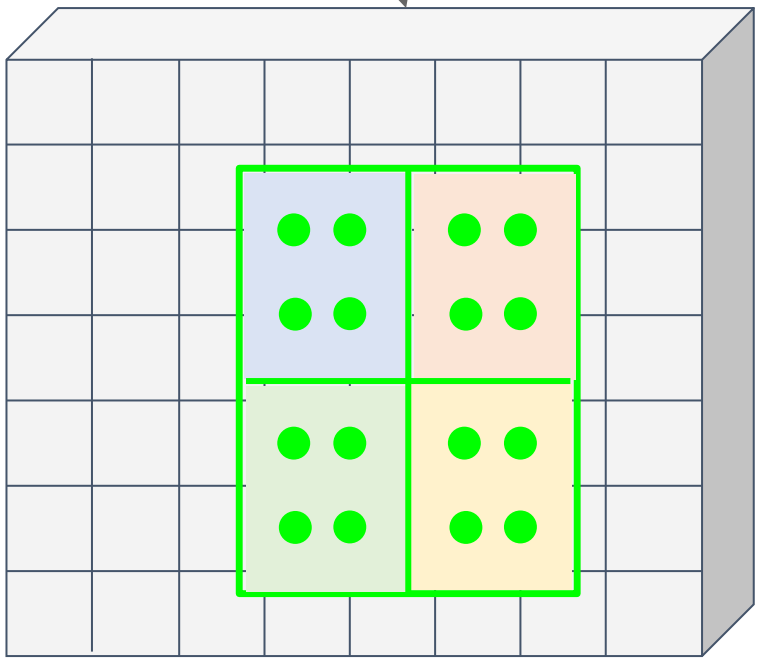
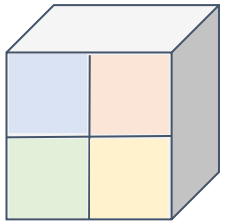


Image features
(e.g. 512 x 20 x 15)

Sample features at regularly-spaced points in each subregion using **bilinear interpolation**

After sampling, max-pool in each subregion



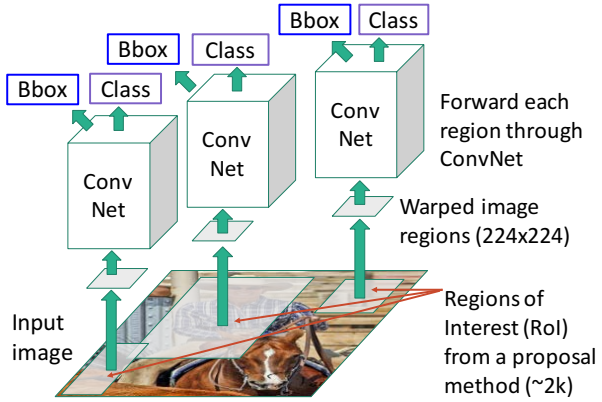
Region features
(here 512 x 2 x 2;
In practice e.g 512 x 7 x 7)

Output features now aligned to input box! And we can backprop to box coordinates!

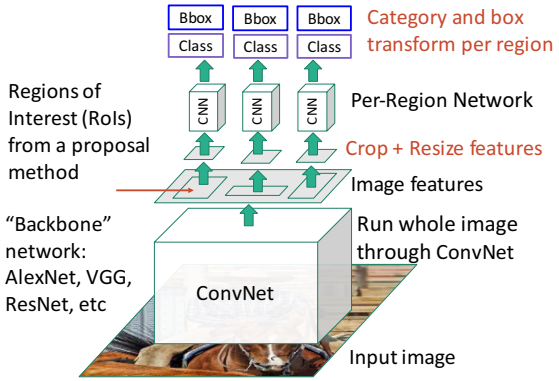
Object Detection Methods

Both of these rely on anchor boxes.
Can we do detection without anchors?

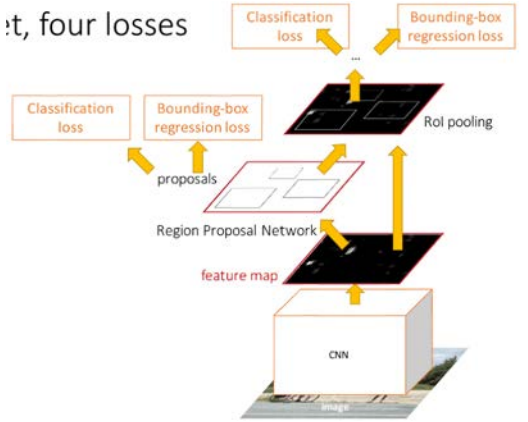
"Slow" R-CNN: Run CNN independently for each region



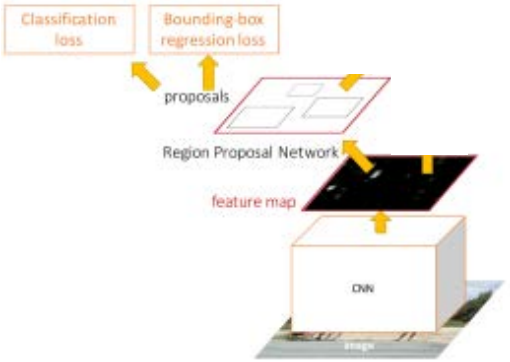
Fast R-CNN: Apply differentiable cropping to shared image features



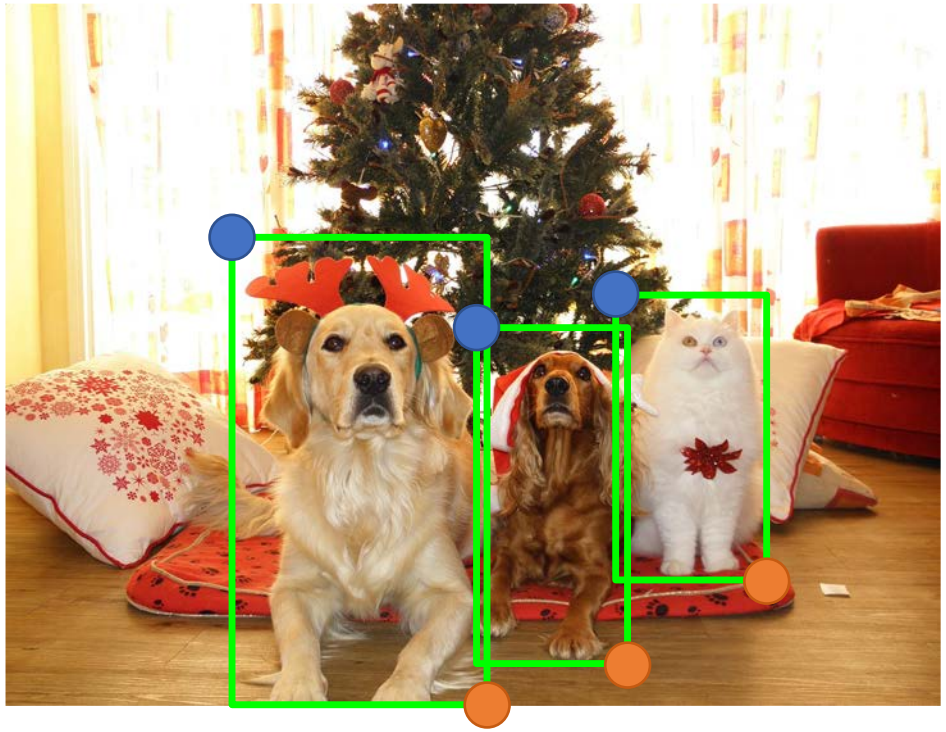
Faster R-CNN: Compute proposals with CNN



Single-Stage: Fully convolutional detector

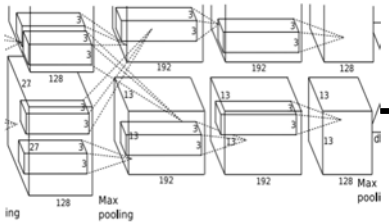


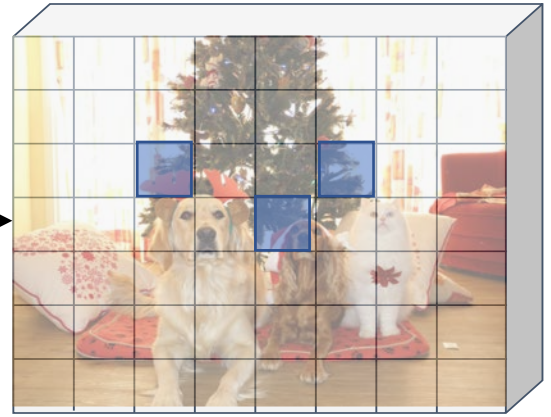
Detection without Anchors: CornerNet



Represent bounding boxes by pairs of corners

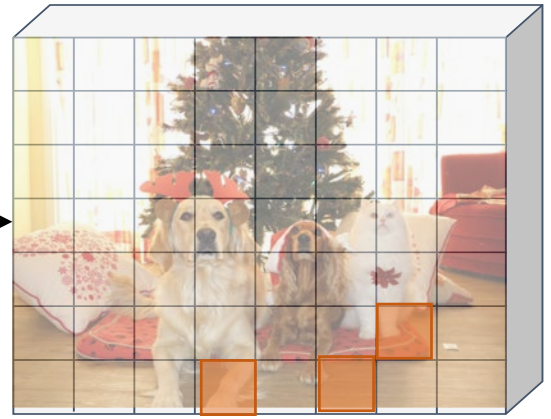
Backbone CNN





Upper left corners
Heatmap: C x H x W
Embeddings: D x H x W

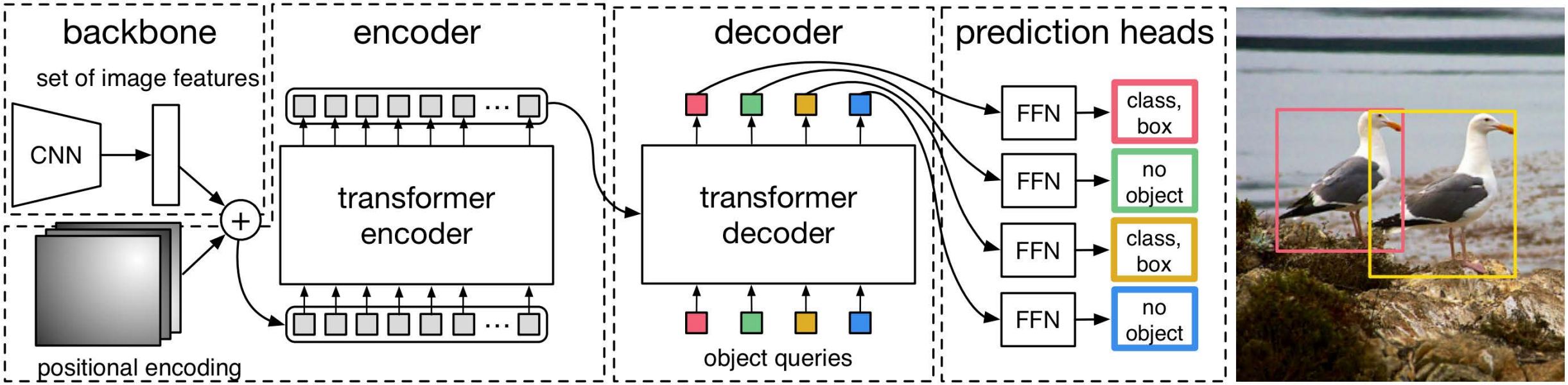
Matching corners are trained to have similar embeddings



Lower right corners
Heatmap: C x H x W
Embeddings: D x H x W

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

Detection without Anchors: Transformers



(+) No RPN!
(+) No anchors!
(+) No NMS!

(-) Slow and brittle to train
(-) Worse than Faster R-CNN for small objects

Carion et al, "End-to-End Object Detection with Transformers", ECCV 2020

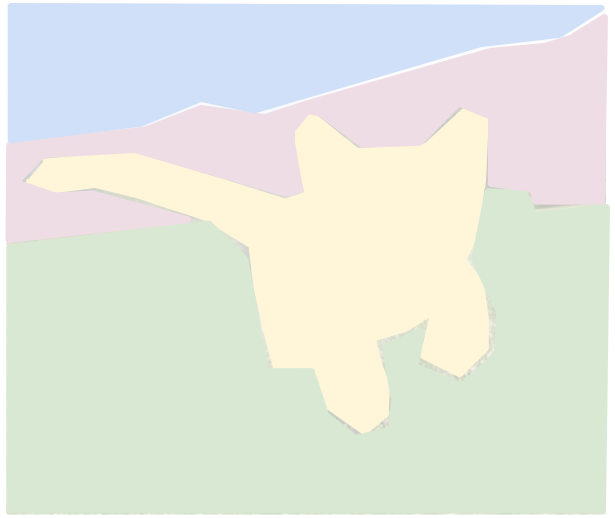
Computer Vision Tasks: Object Detection

Classification

CAT

No spatial extent

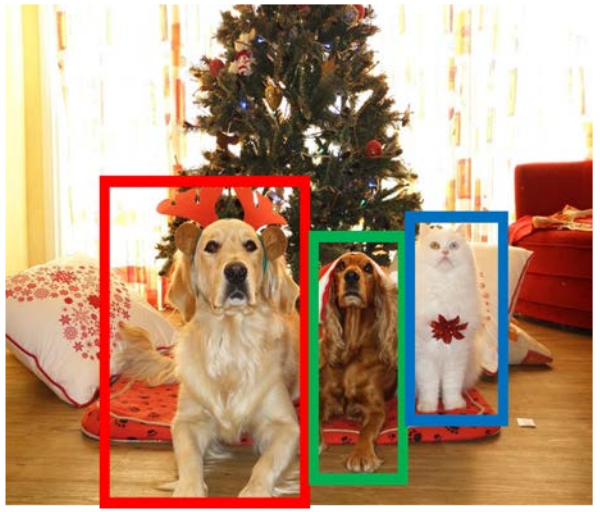
Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

Object Detection



DOG, DOG, CAT

Instance Segmentation

DOG, DOG, CAT

Multiple Objects

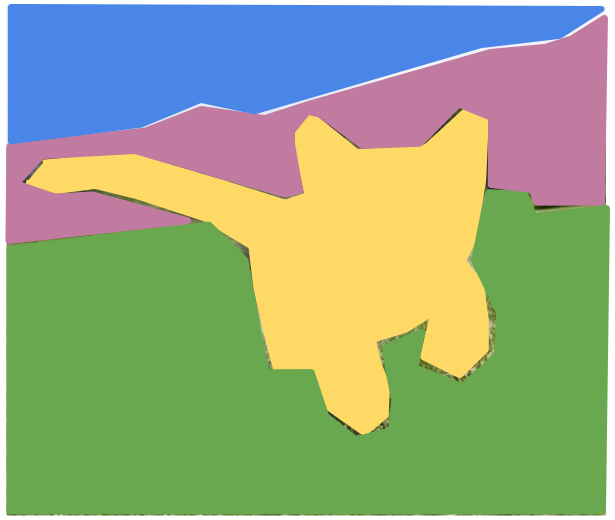
Computer Vision Tasks: Semantic Segmentation

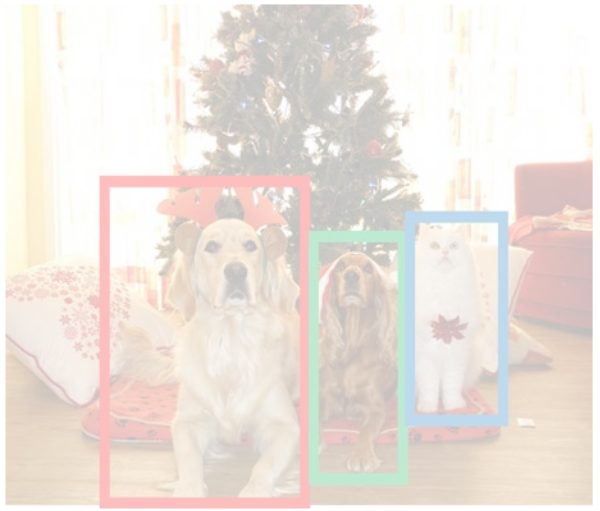
Classification

Semantic Segmentation

Object Detection

Instance Segmentation





CAT

GRASS, CAT, TREE, SKY

DOG, DOG, CAT

DOG, DOG, CAT

No spatial extent

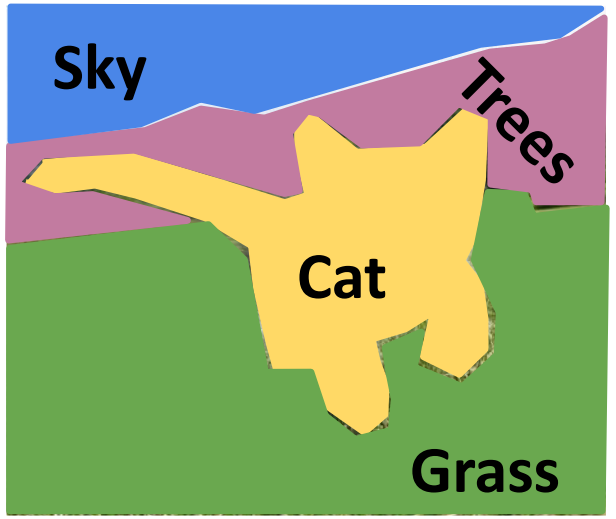
No objects, just pixels

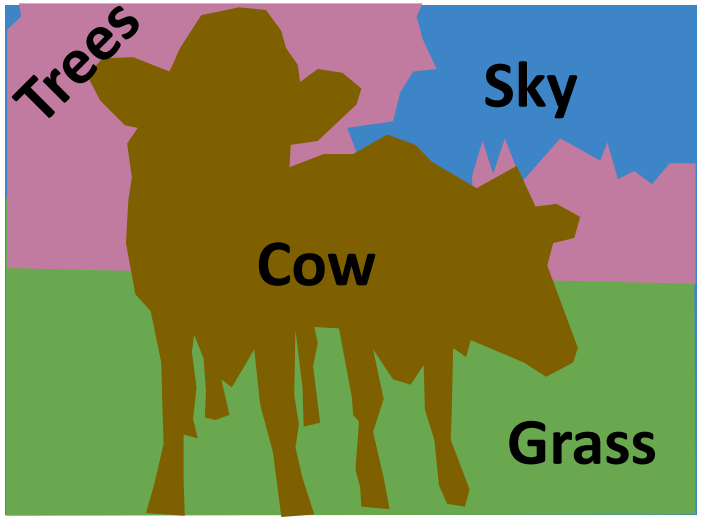
Multiple Objects

Semantic Segmentation

Label each pixel in the image with a category label

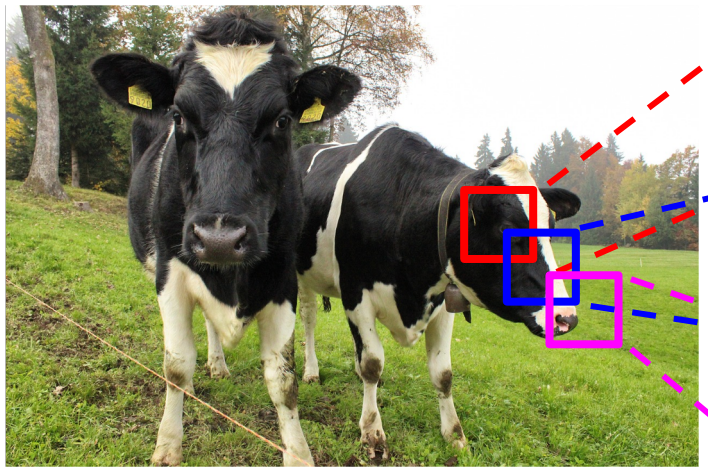
Don’t differentiate instances, only care about pixels



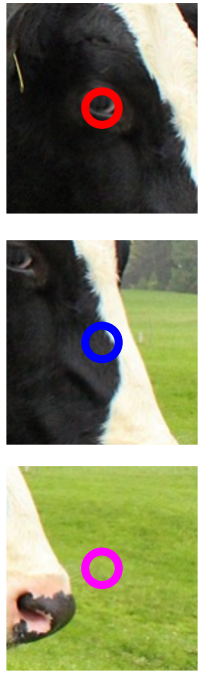


Semantic Segmentation Idea: Sliding Window

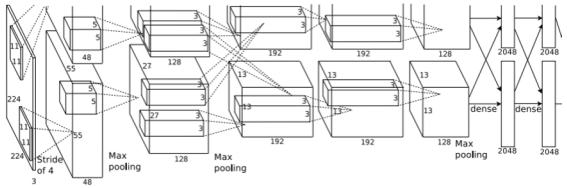
Full image



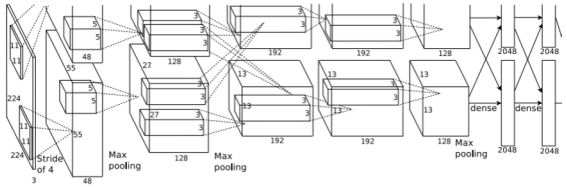
Extract patch



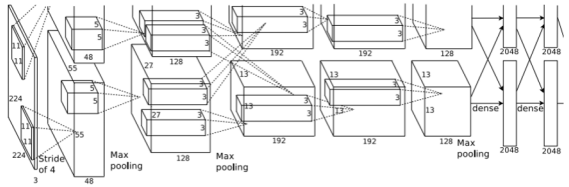
Classify center pixel with CNN



Cow



Cow



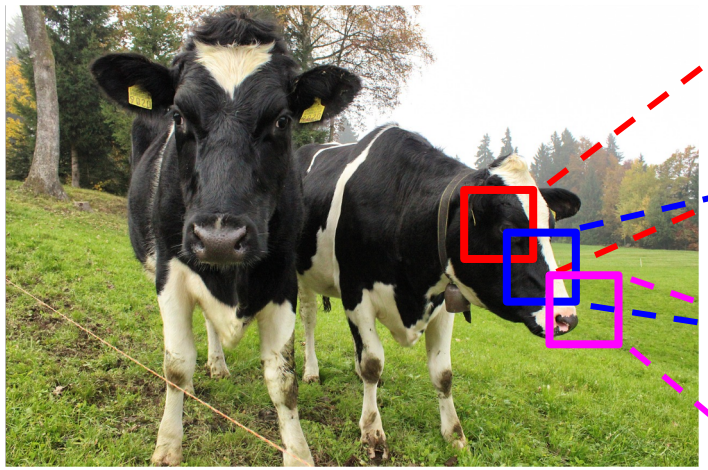
Grass

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013

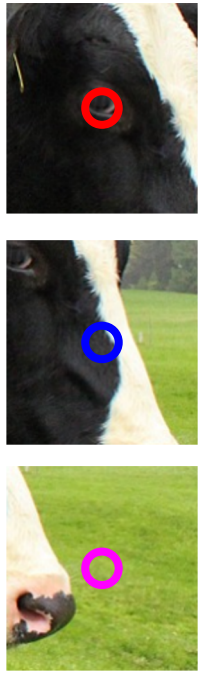
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

Semantic Segmentation Idea: Sliding Window

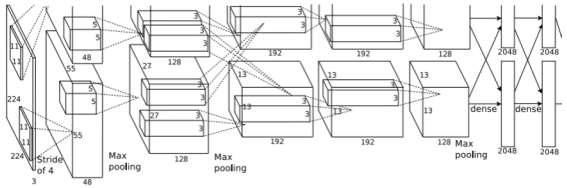
Full image



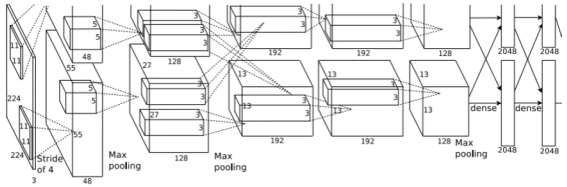
Extract patch



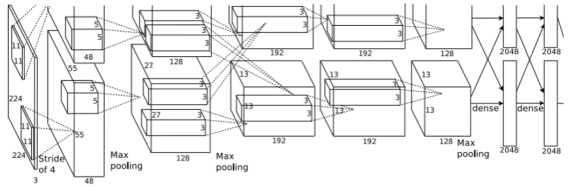
Classify center pixel with CNN



Cow



Cow



Grass

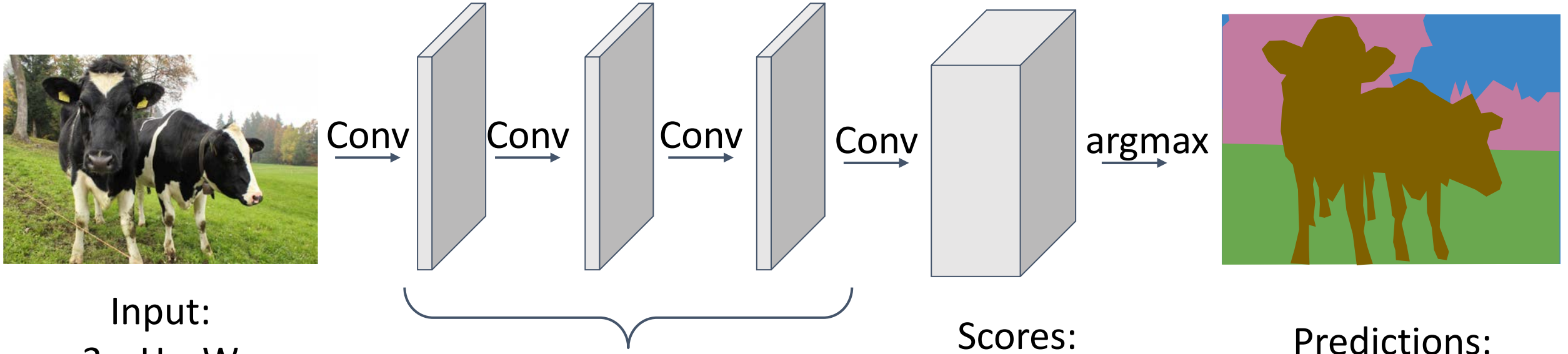
Problem: Very inefficient! Not reusing shared features between overlapping patches

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!



Input:
3 x H x W

Convolutions:
D x H x W

Scores:
C x H x W

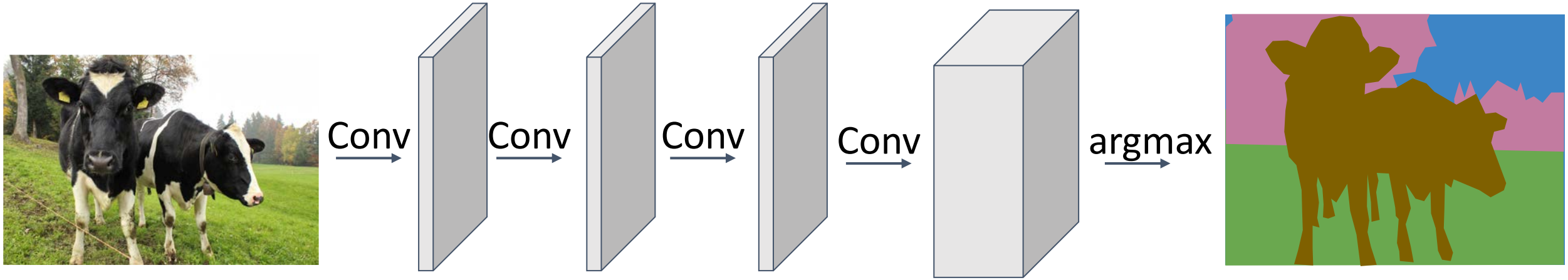
Predictions:
H x W

Loss function: Per-Pixel cross-entropy

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!



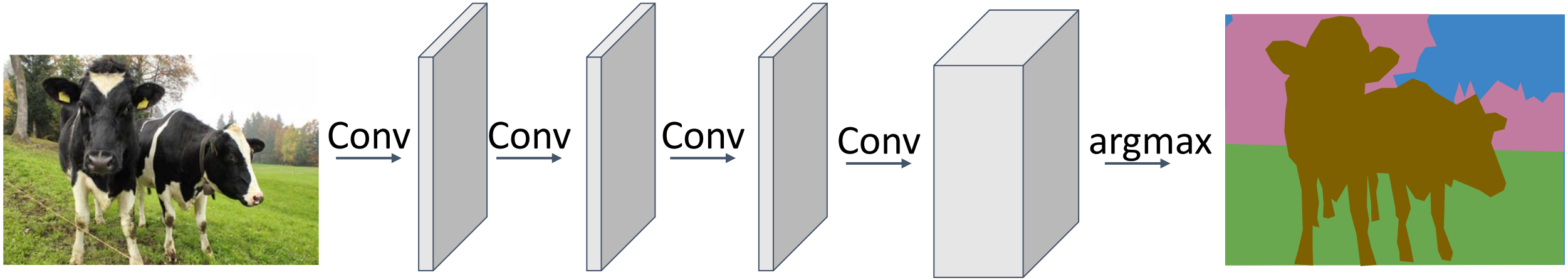
Input:
3 x H x W

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

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

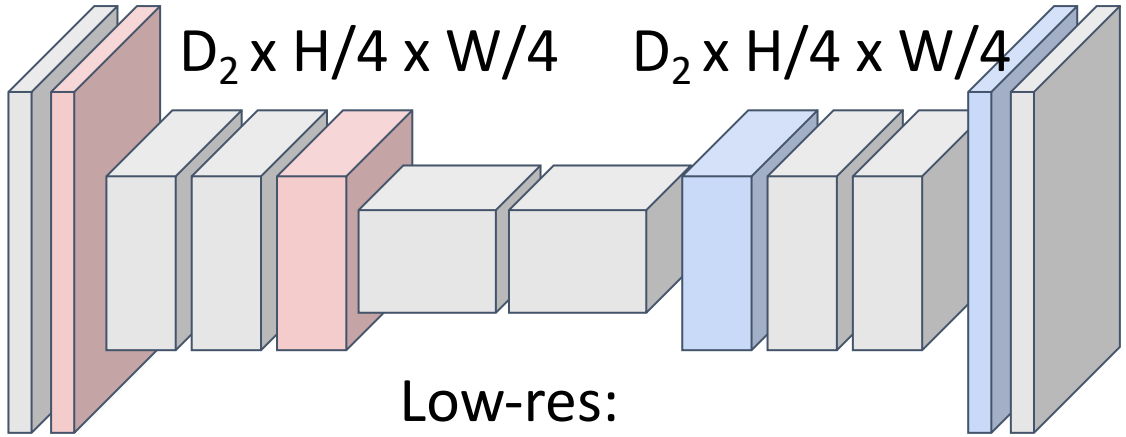
Problem #2: Convolution on high res images is expensive! Recall ResNet stem aggressively downsamples

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



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

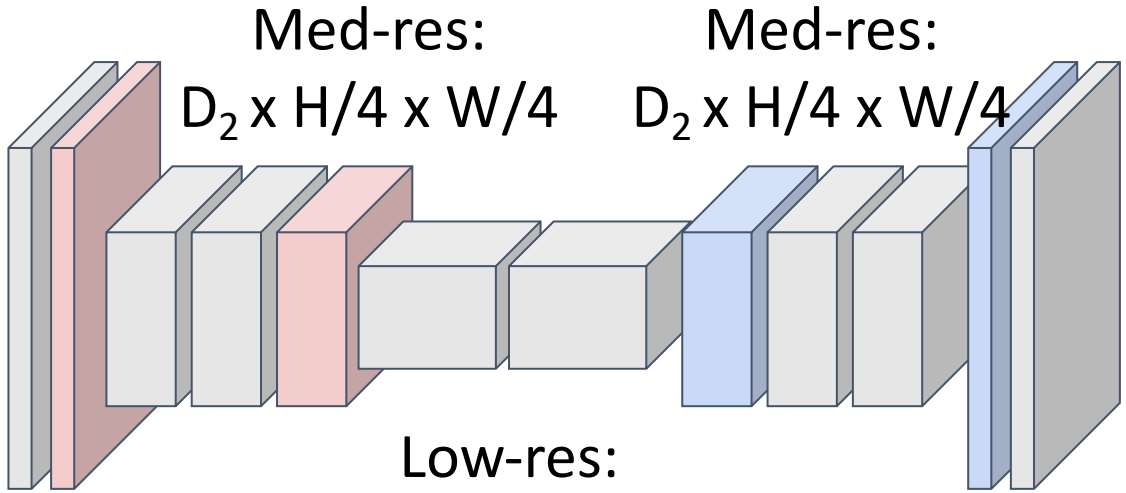
Semantic Segmentation: Fully Convolutional Network

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

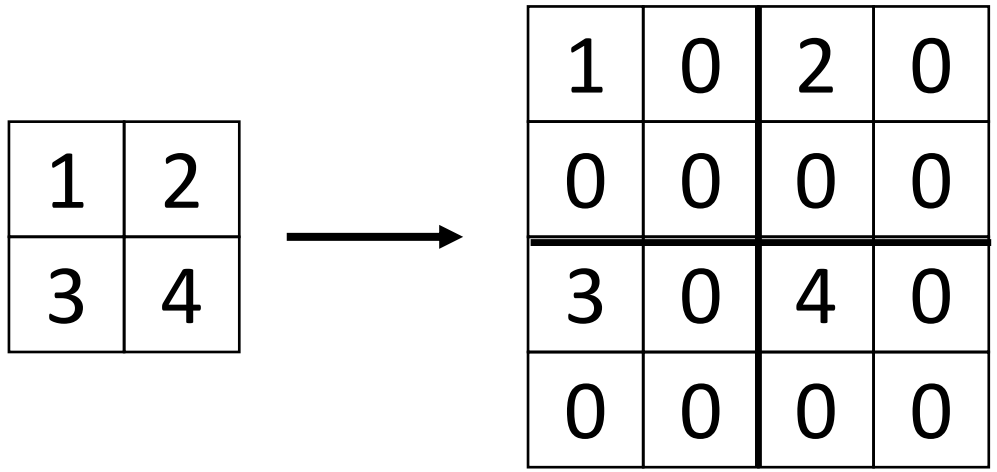


Predictions:
H x W

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

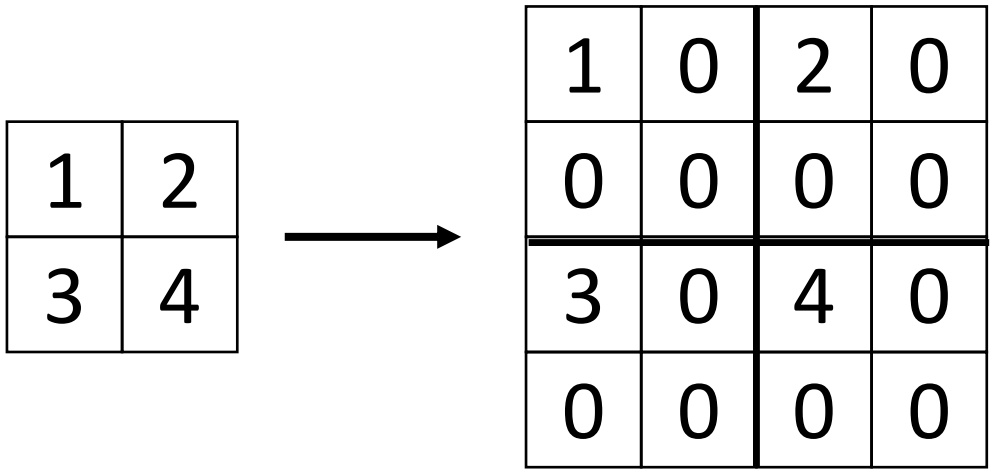


Input
C x 2 x 2

Output
C x 4 x 4

In-Network Upsampling: “Unpooling”

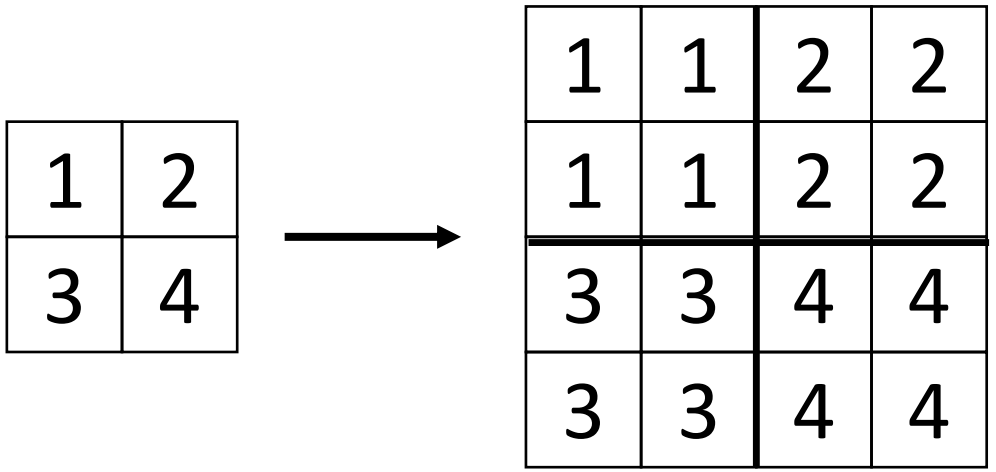
Bed of Nails



Input
C x 2 x 2

Output
C x 4 x 4

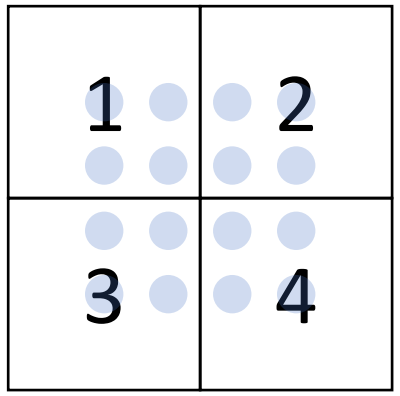
Nearest Neighbor



Input
C x 2 x 2

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

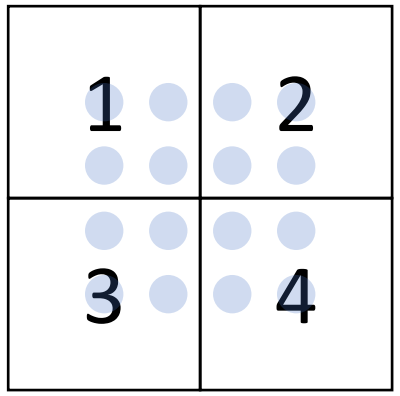
Output: C x 4 x 4

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

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

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

In-Network Upsampling: Bicubic 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

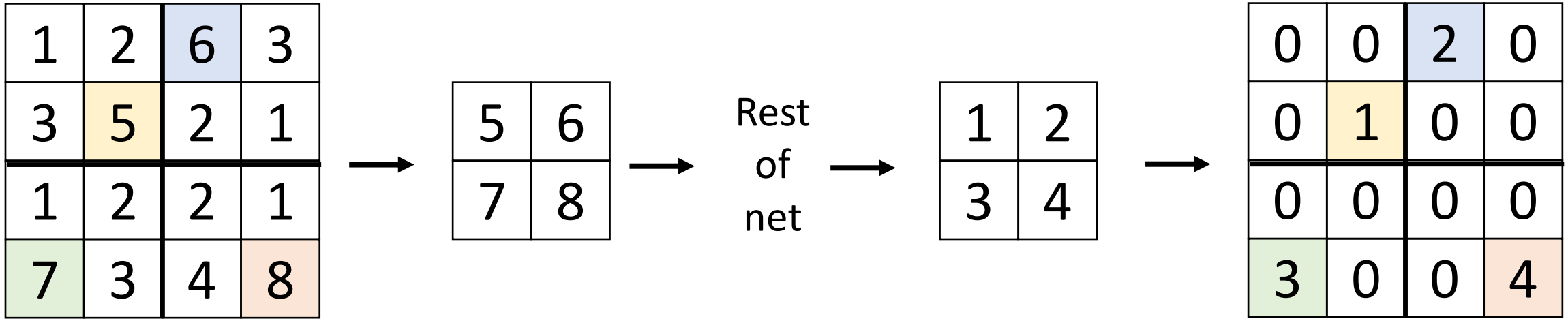
Output: C x 4 x 4

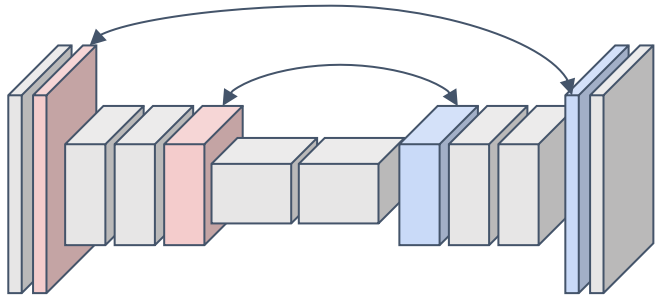
Use **three** closest neighbors in x and y to construct **cubic** approximations
(This is how we normally resize images!)

In-Network Upsampling: “Max Unpooling”

Max Pooling: Remember which position had the max

Max Unpooling: Place into remembered positions



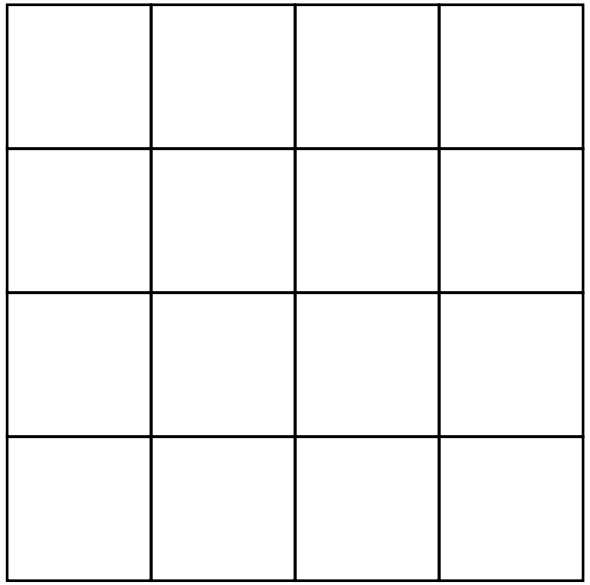


Pair each downsampling layer with an upsampling layer

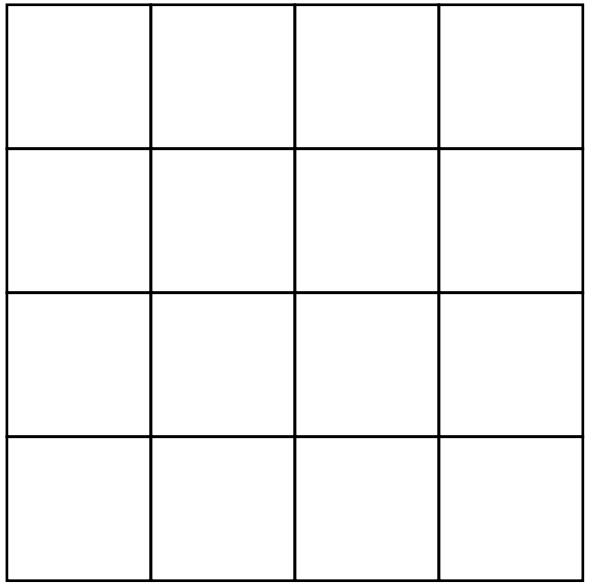
Noh et al, “Learning Deconvolution Network for Semantic Segmentation”, ICCV 2015

Learnable Upsampling: Transposed Convolution

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



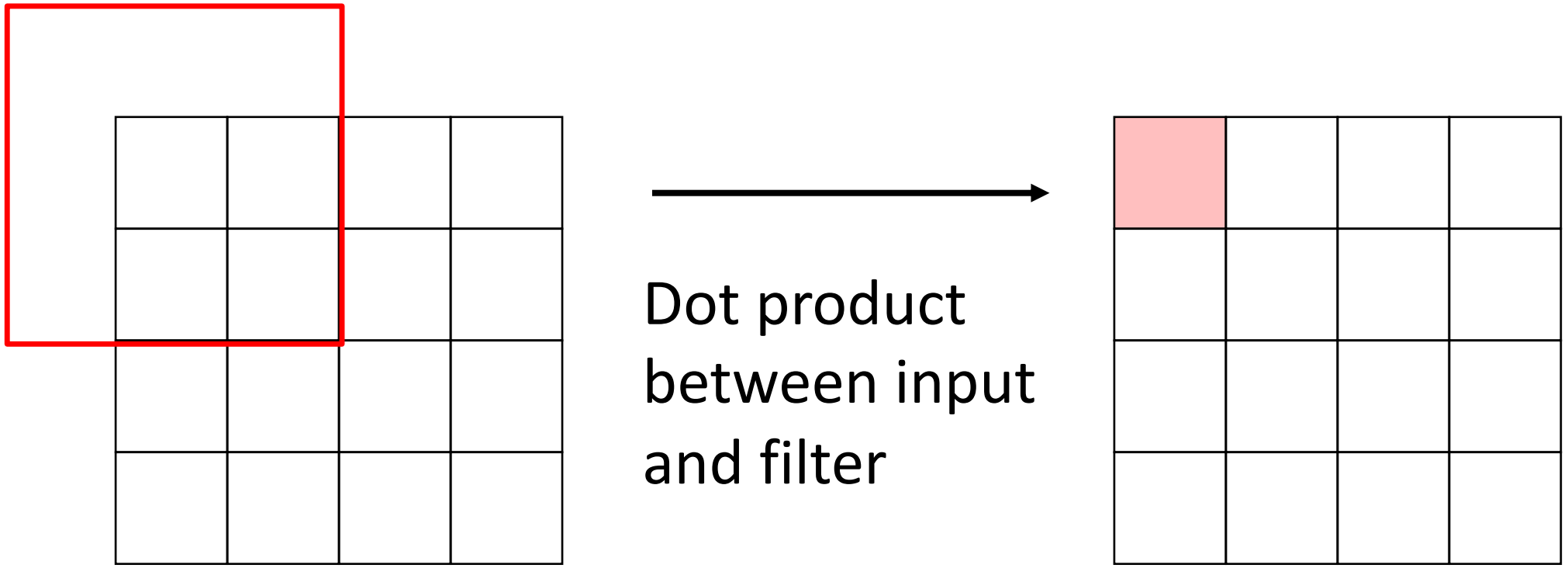
Input: 4 x 4



Output: 4 x 4

Learnable Upsampling: Transposed Convolution

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

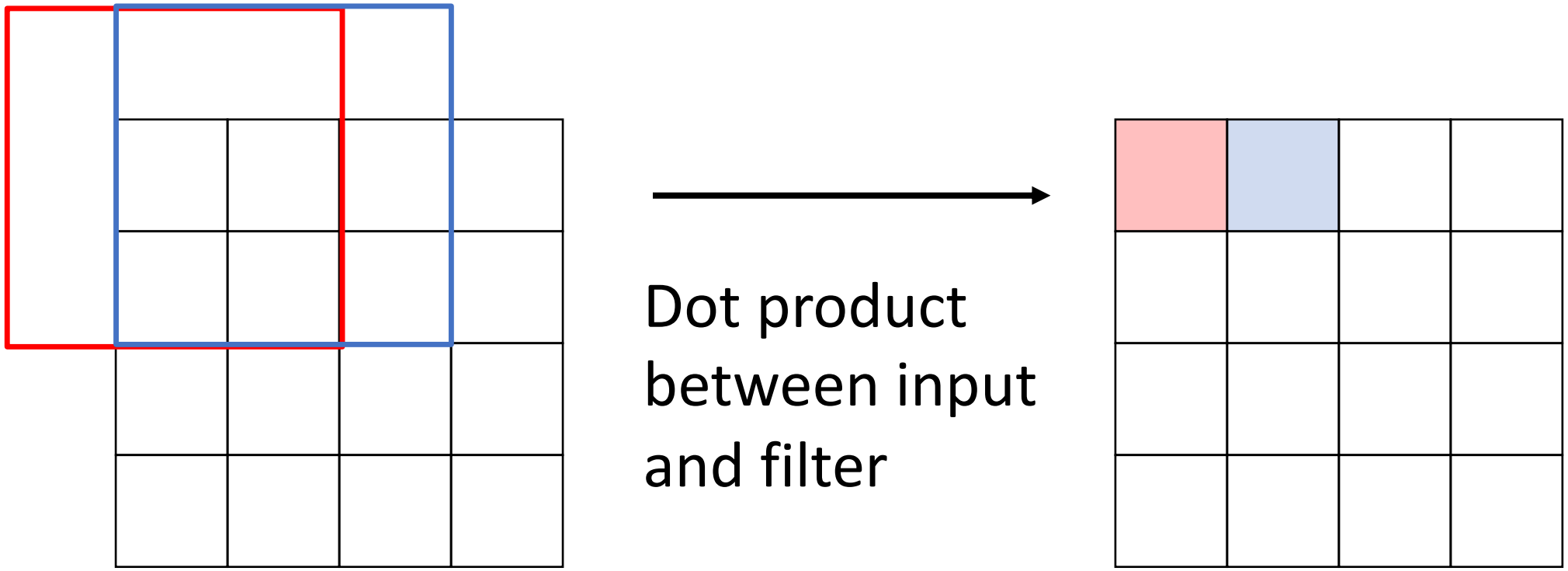


Input: 4 x 4

Output: 4 x 4

Learnable Upsampling: Transposed Convolution

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

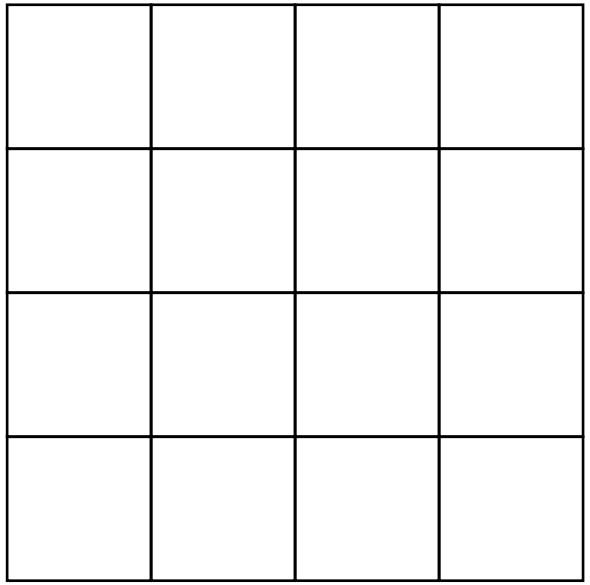


Input: 4 x 4

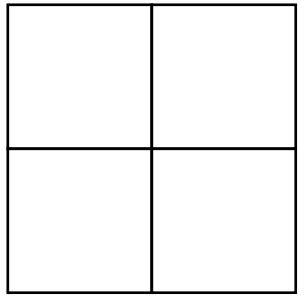
Output: 4 x 4

Learnable Upsampling: Transposed Convolution

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



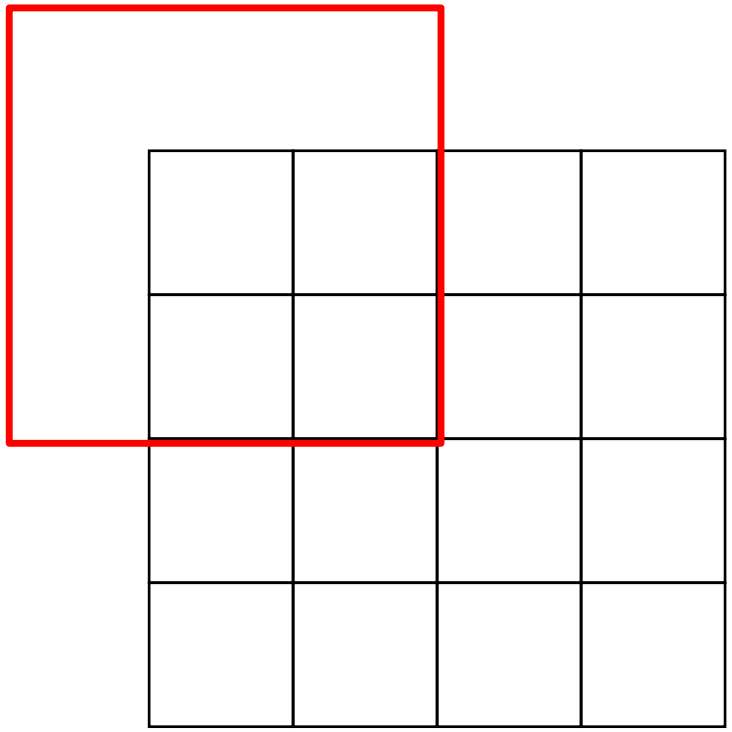
Input: 4 x 4



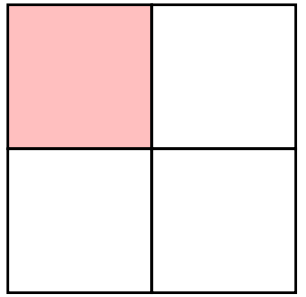
Output: 2 x 2

Learnable Upsampling: Transposed Convolution

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



Dot product between input and filter

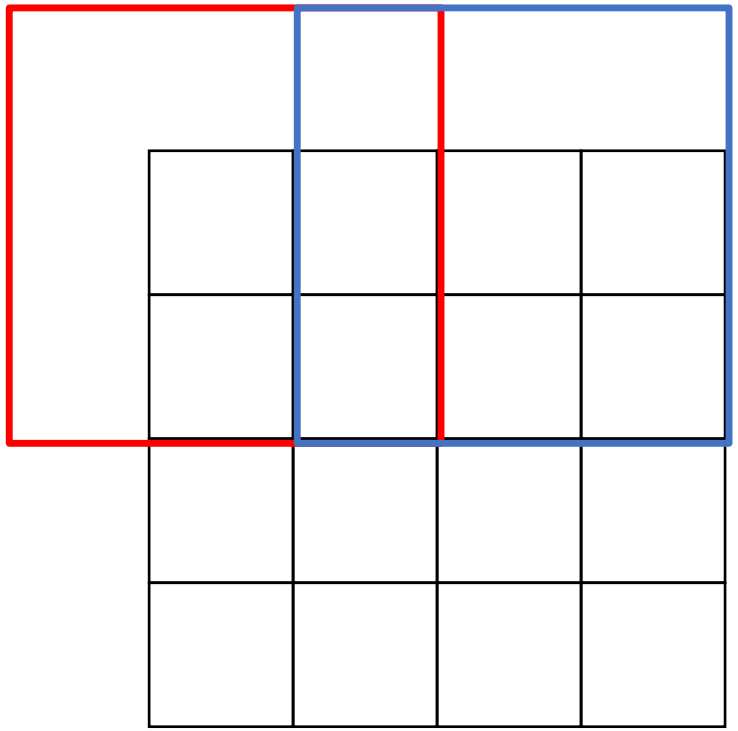


Input: 4 x 4

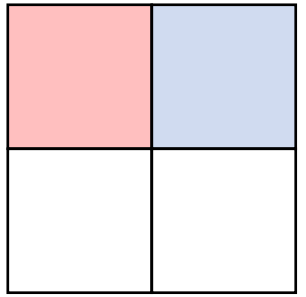
Output: 2 x 2

Learnable Upsampling: Transposed Convolution

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



Dot product between input and filter

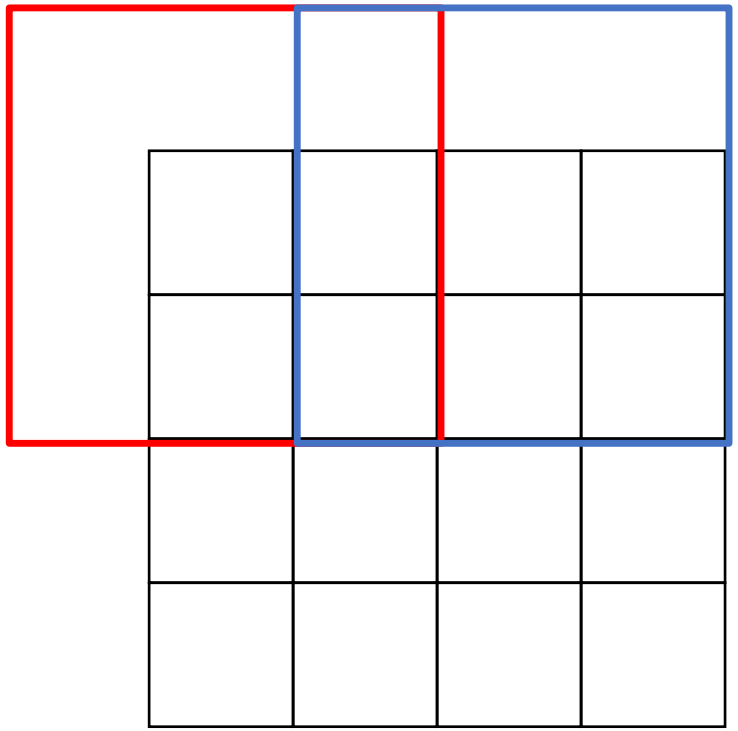


Input: 4 x 4

Output: 2 x 2

Learnable Upsampling: Transposed Convolution

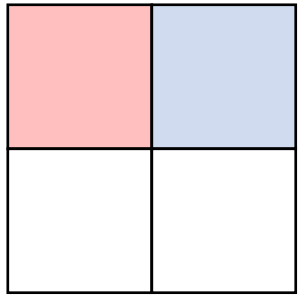
Recall: Normal 3 x 3 convolution, stride 2, pad 1



Convolution with stride > 1 is “Learnable Downsampling”

Can we use stride < 1 for “Learnable Upsampling”?

Dot product between input and filter

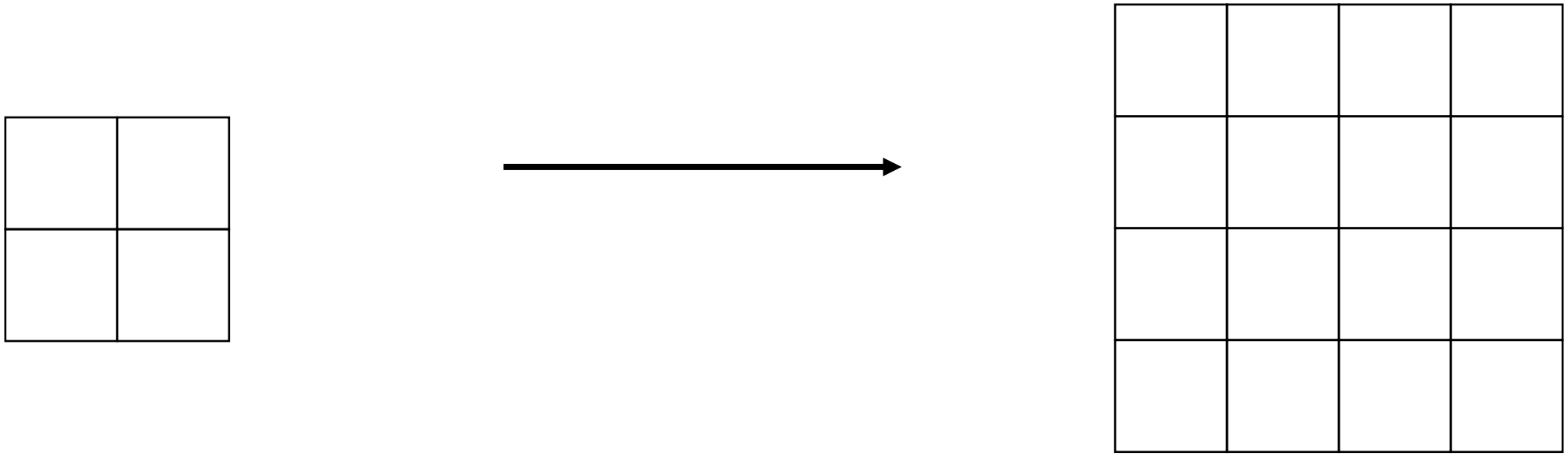


Input: 4 x 4

Output: 2 x 2

Learnable Upsampling: Transposed Convolution

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

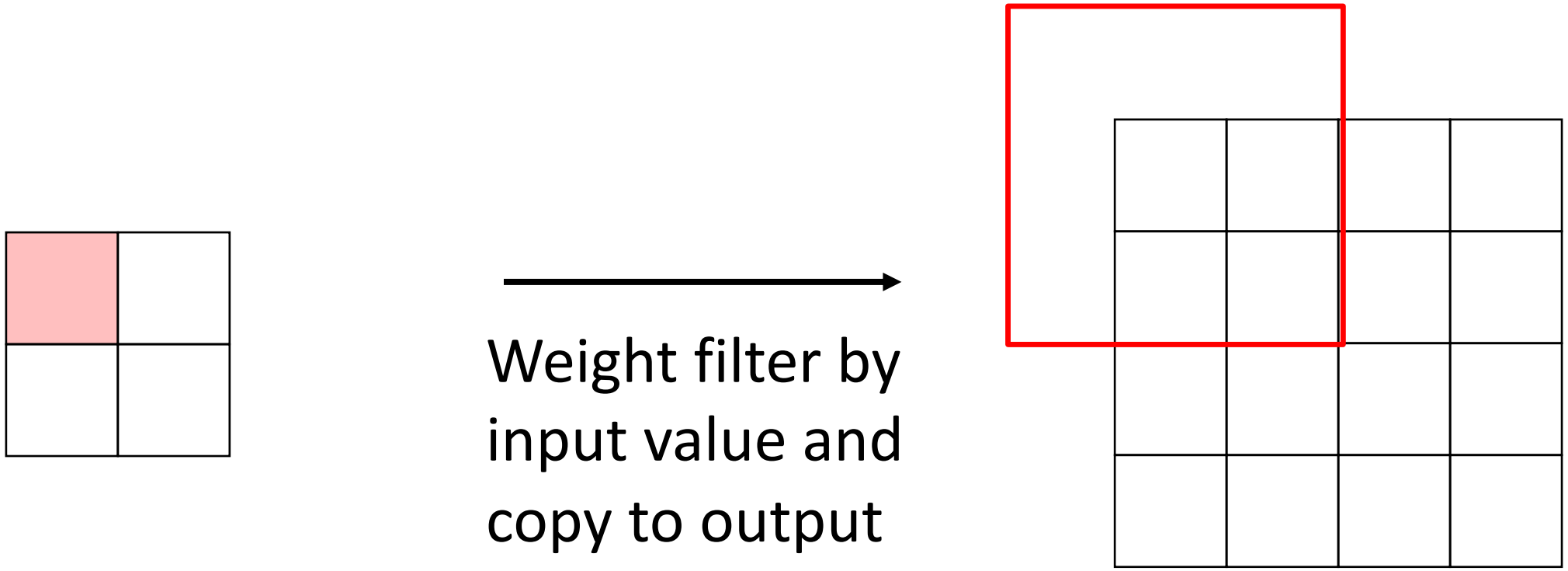


Input: 2 x 2

Output: 4 x 4

Learnable Upsampling: Transposed Convolution

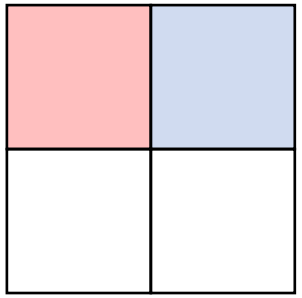
3 x 3 **convolution transpose**, stride 2



Learnable Upsampling: Transposed Convolution

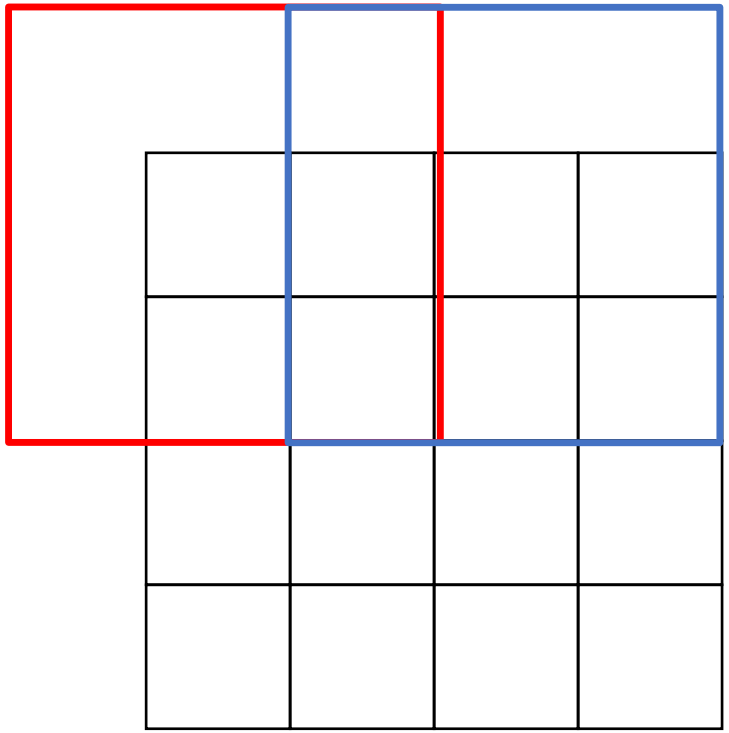
3 x 3 **convolution transpose**, stride 2

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

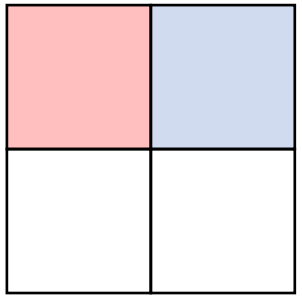


Output: 4 x 4

Learnable Upsampling: Transposed Convolution

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

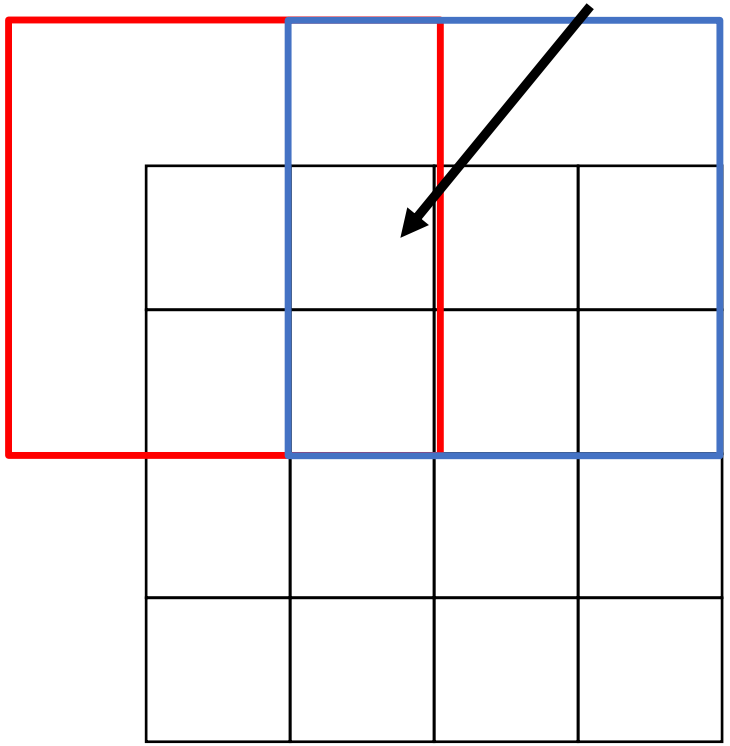
Filter moves 2 pixels in output
for every 1 pixel in input



Input: 2 x 2

Weight filter by input value and copy to output

Sum where output overlaps



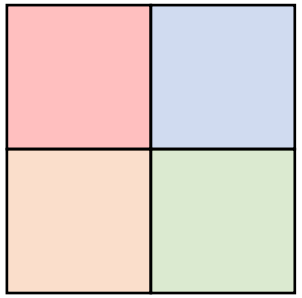
Output: 4 x 4

Learnable Upsampling: Transposed Convolution

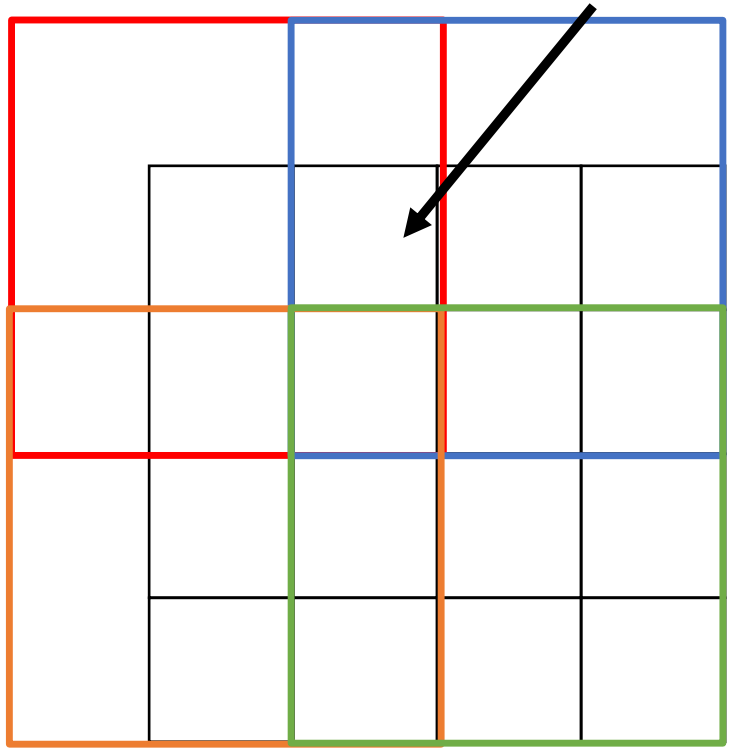
3 x 3 **convolution transpose**, stride 2

Sum where output overlaps

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



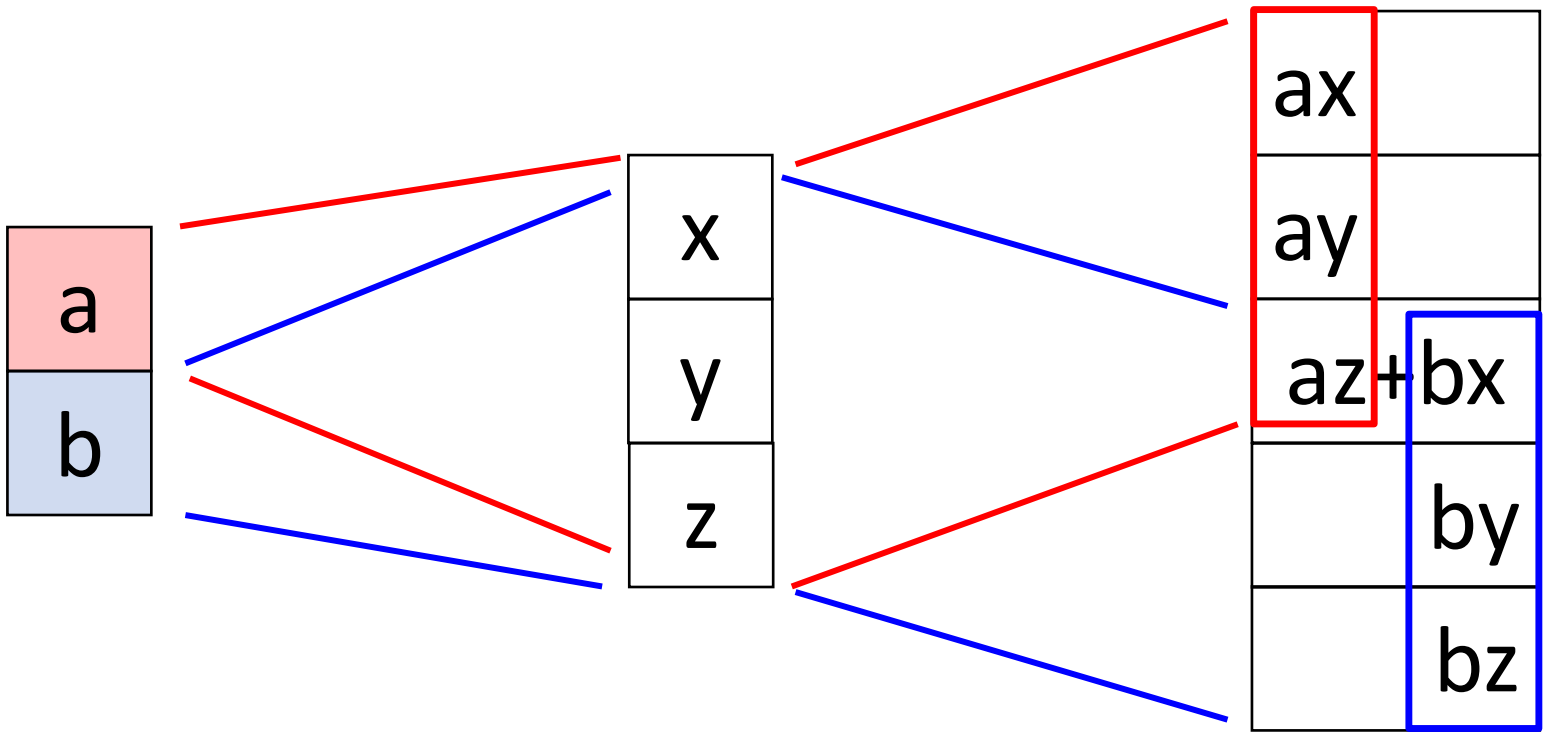
Weight filter by input value and copy to output



Input: 2 x 2

Output: 4 x 4

Transposed Convolution: 1D example

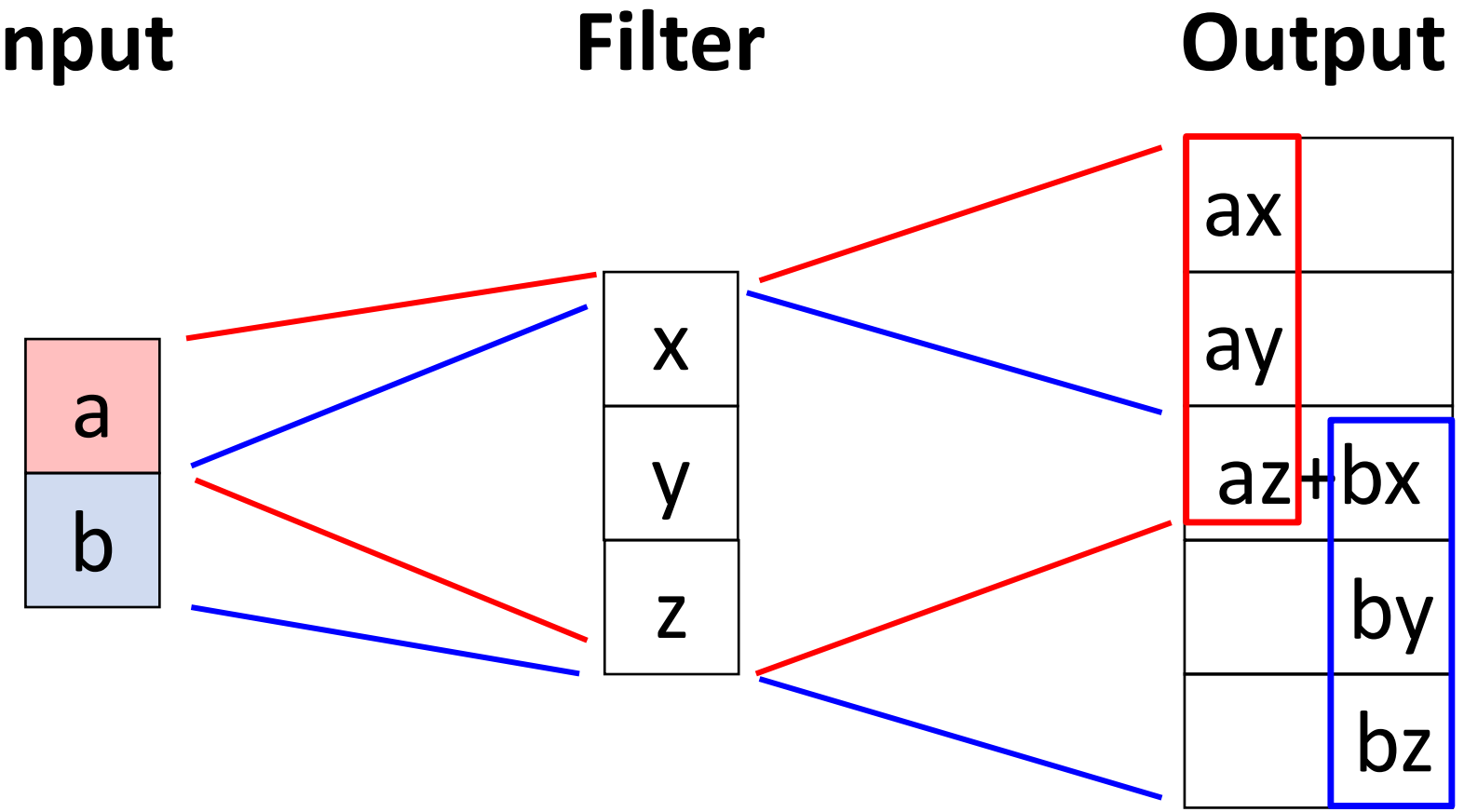


Output has copies of filter weighted by input

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
- Transposed Convolution (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, stride=2, 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

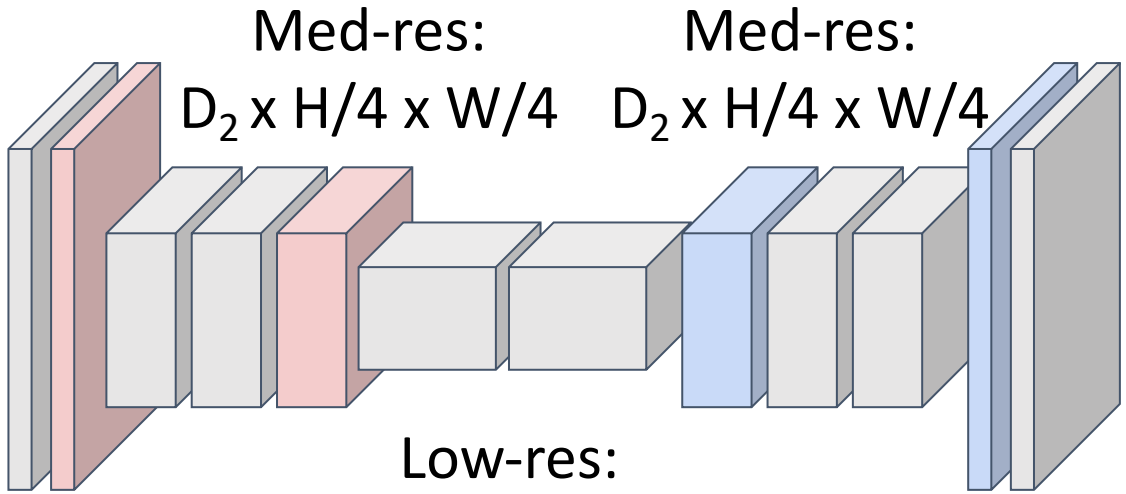
Semantic Segmentation: Fully Convolutional Network

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 x H x W



Predictions:
H x W

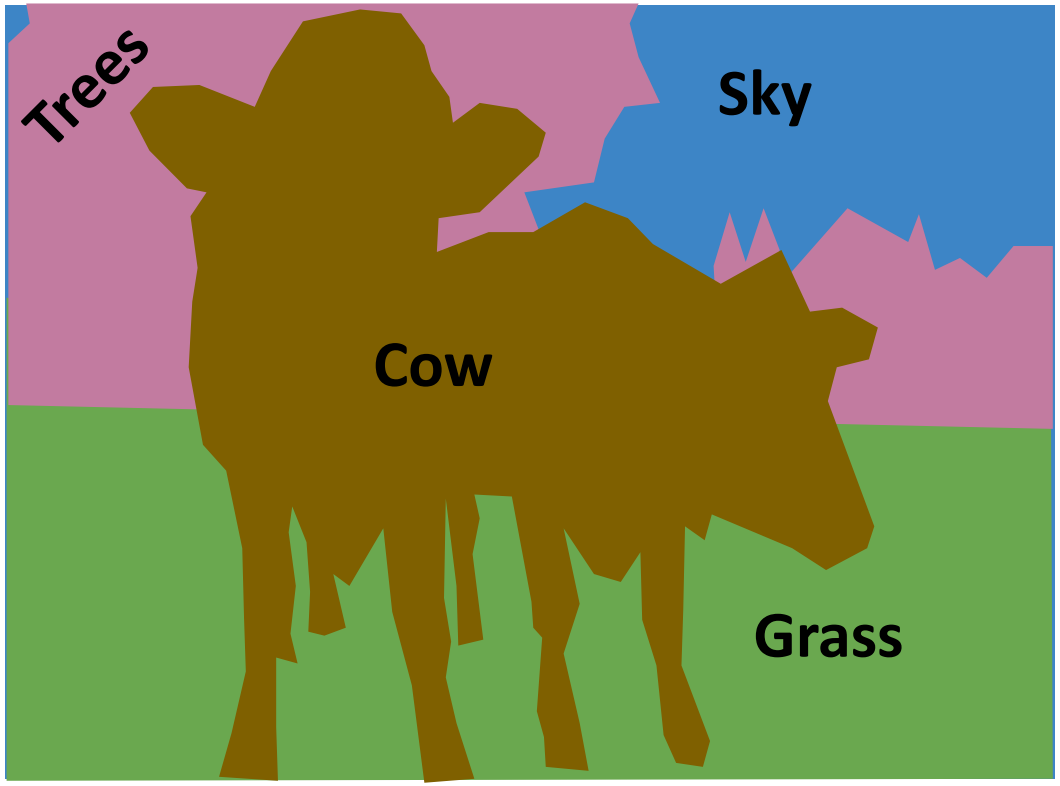
Loss function: Per-Pixel cross-entropy

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

Computer Vision Tasks

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

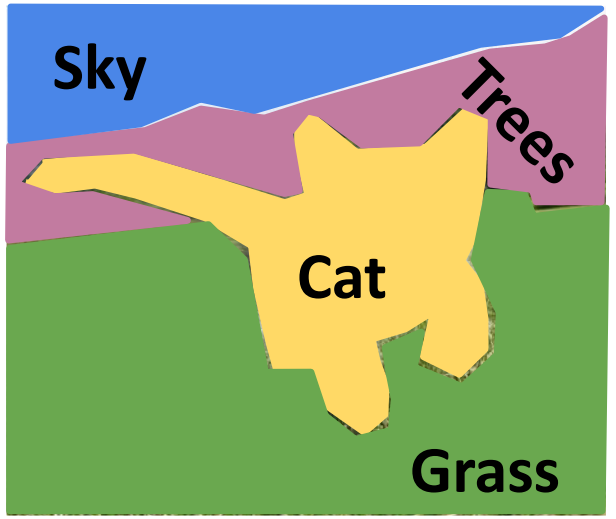
Semantic Segmentation: Gives per-pixel labels, but merges instances

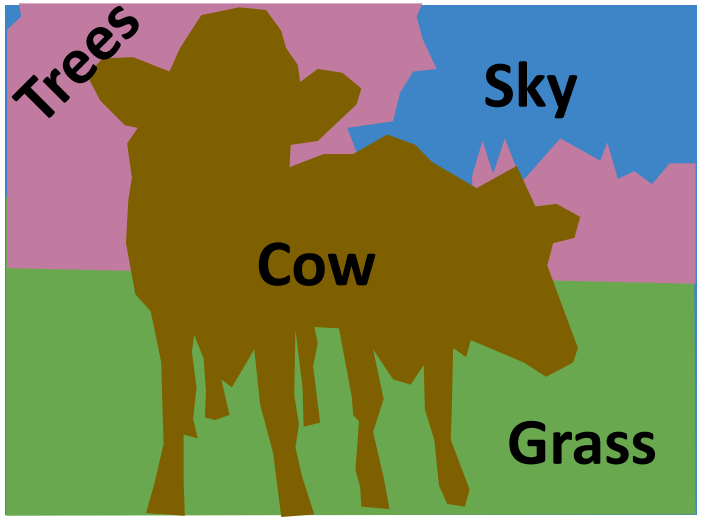


Things and Stuff

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)

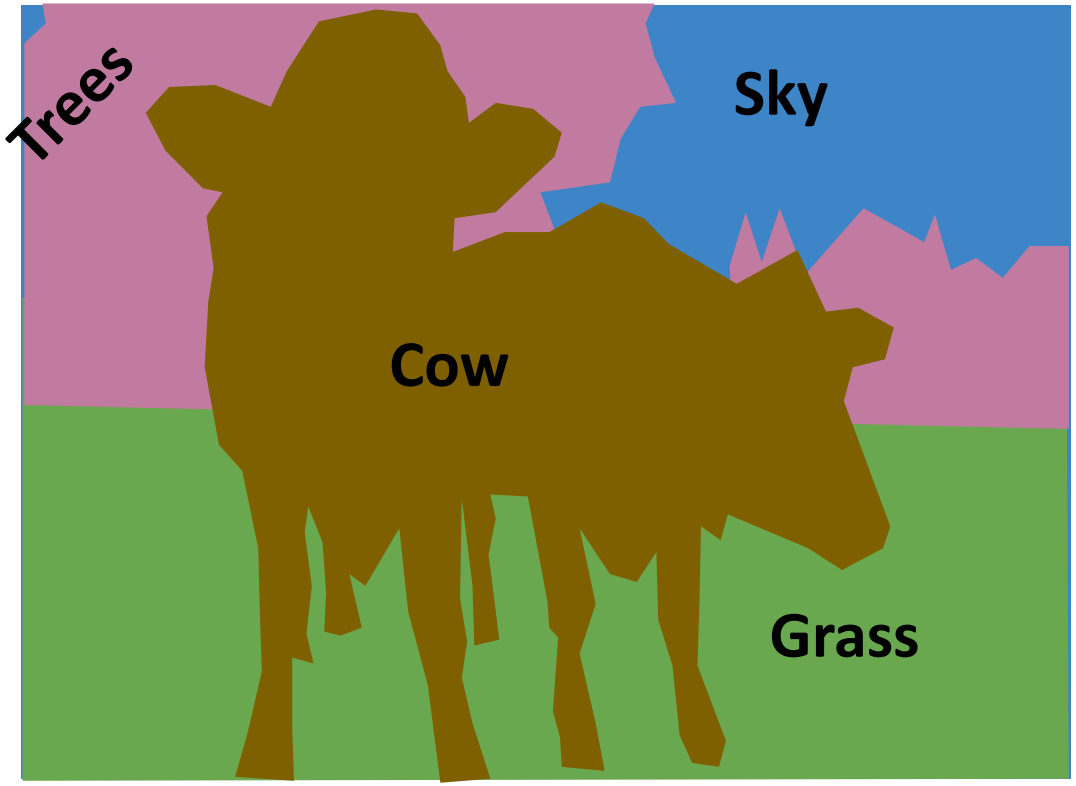




Computer Vision Tasks

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)



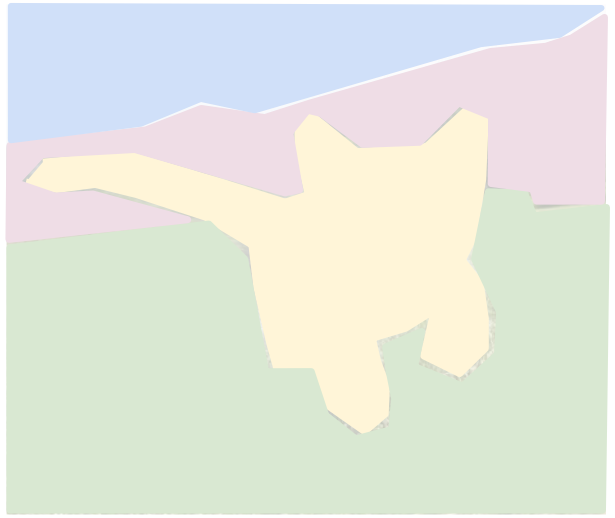
Computer Vision Tasks: Instance Segmentation

Classification

CAT

No spatial extent

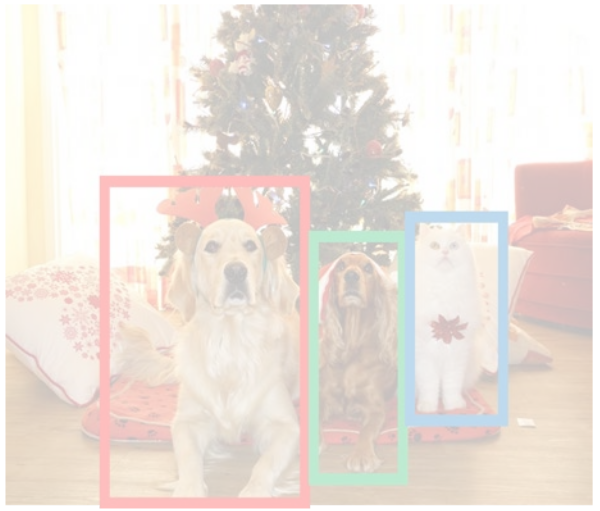
Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

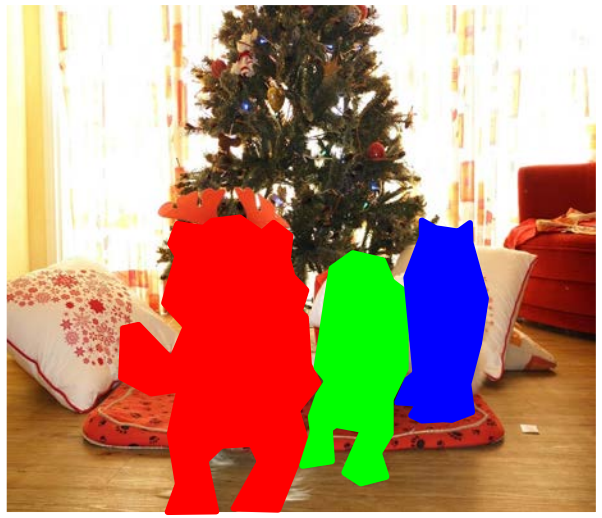
Object Detection



DOG, DOG, CAT

Multiple Objects

Instance Segmentation

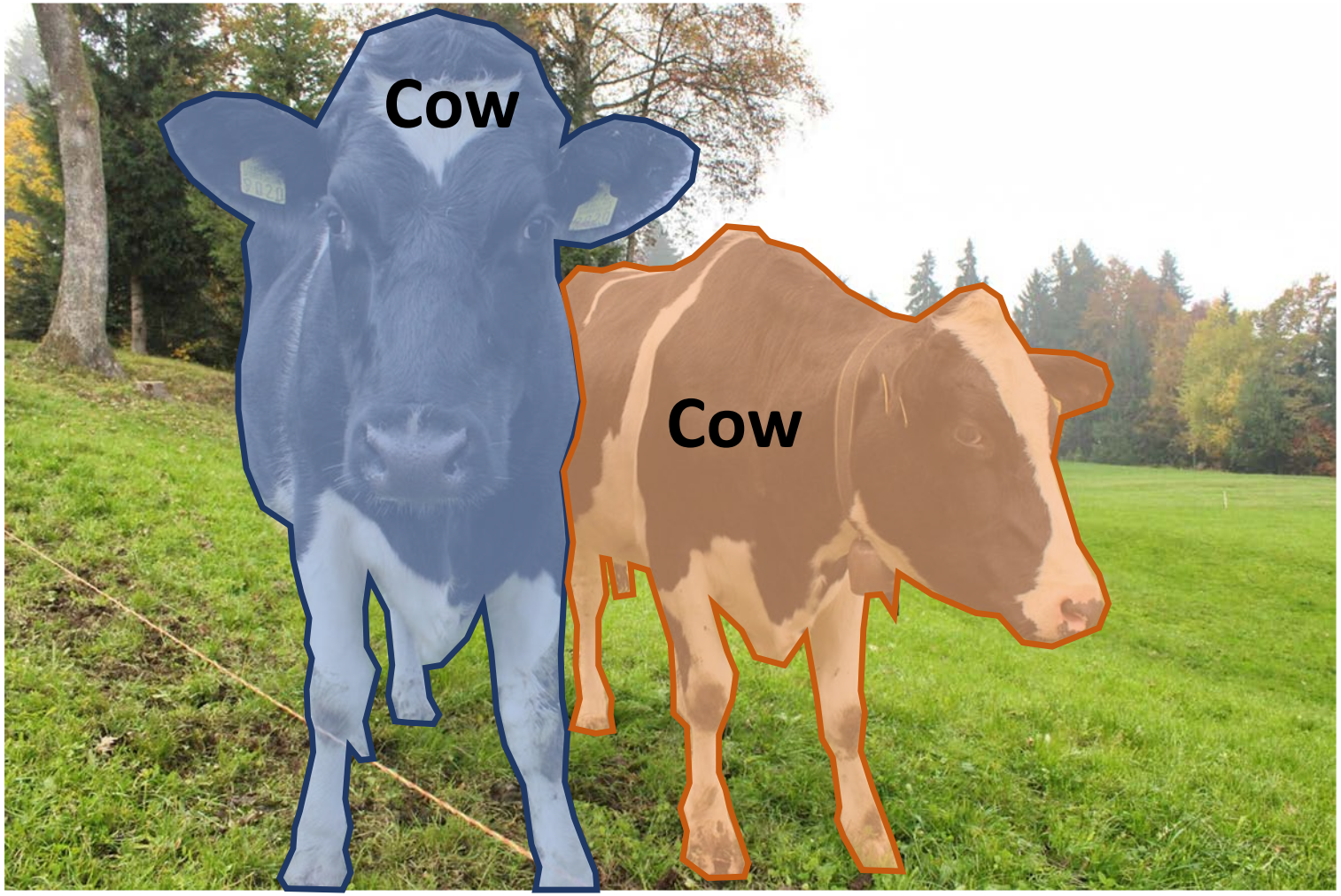


DOG, DOG, CAT

Multiple Objects

Computer Vision Tasks: Instance Segmentation

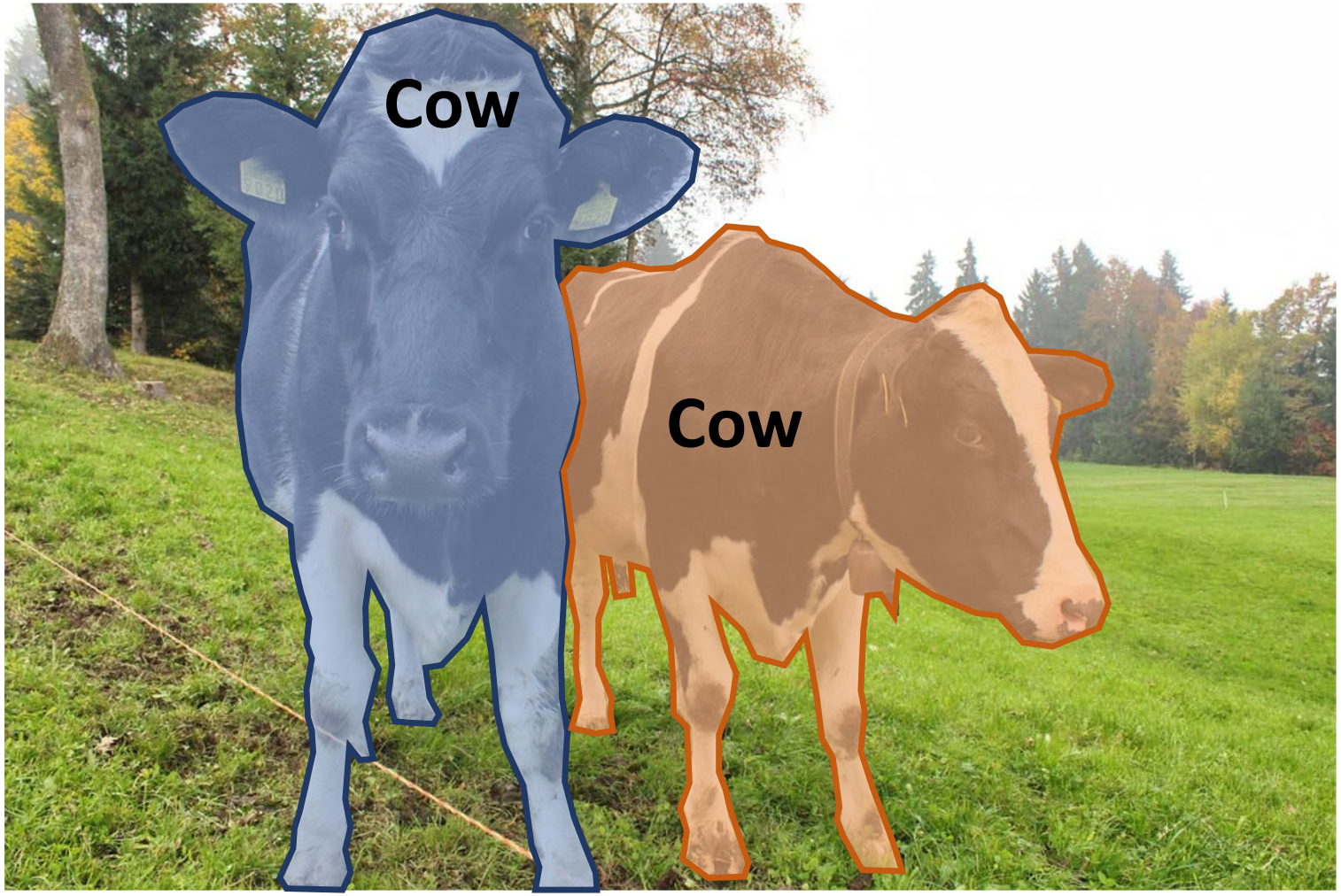
Instance Segmentation: Detect all objects in the image, and identify the pixels that belong to each object (Only things!)



Computer Vision Tasks: Instance Segmentation

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

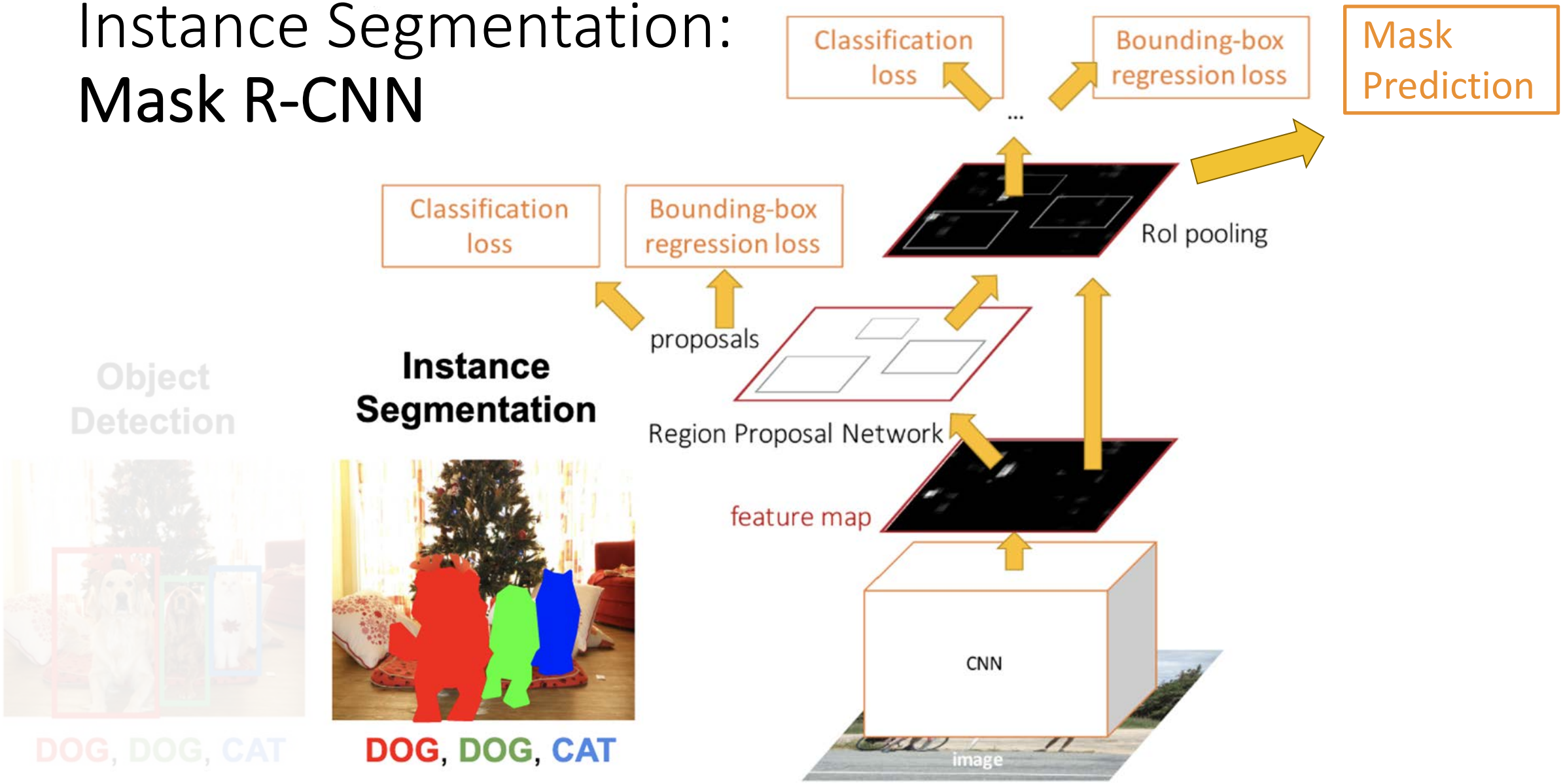
Approach: Perform object detection, then predict a segmentation mask for each object!



Object Detection: Faster R-CNN

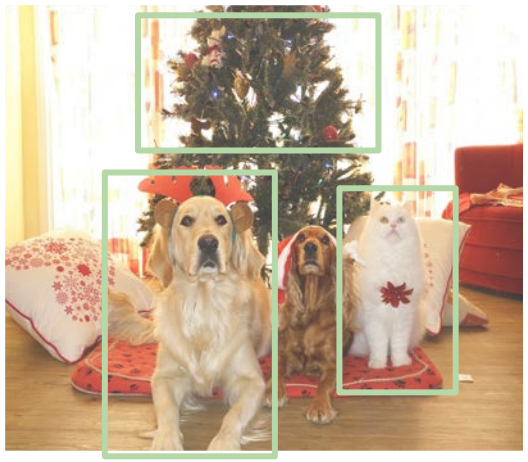
Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NeurIPS 2015

Instance Segmentation: Mask R-CNN

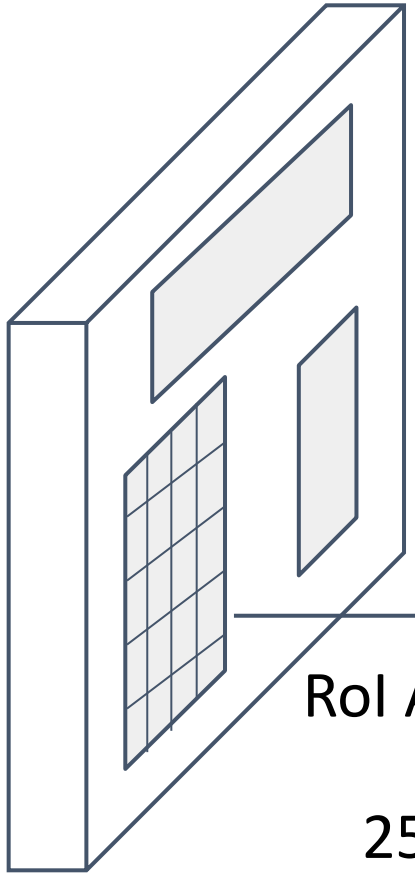


He et al, "Mask R-CNN", ICCV 2017

Mask R-CNN

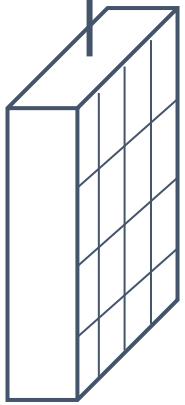


CNN
+RPN



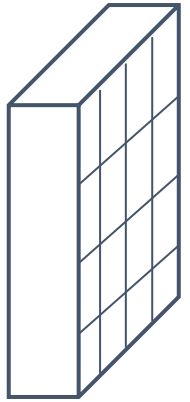
Classification Scores: C
Box coordinates (per class):
4 * C

RoI Align



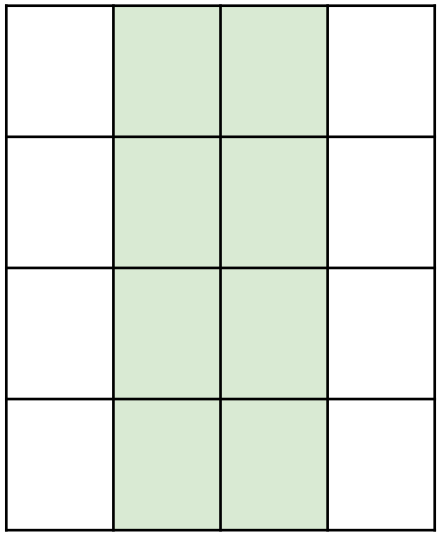
256 x 14 x 14

Conv



256 x 14 x 14

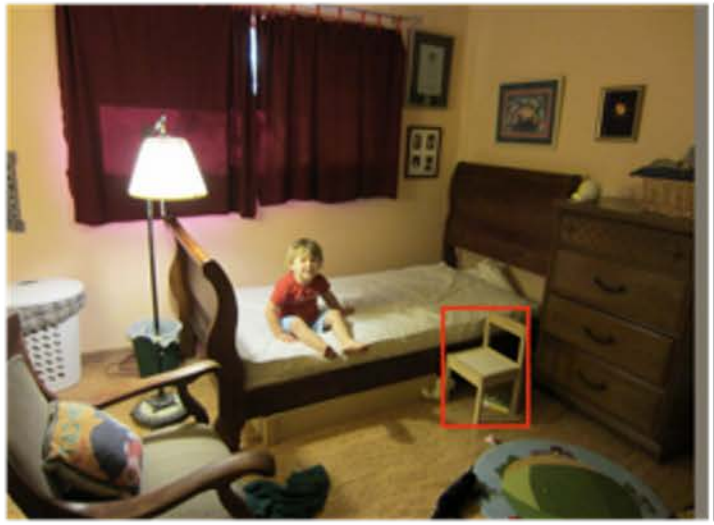
Conv

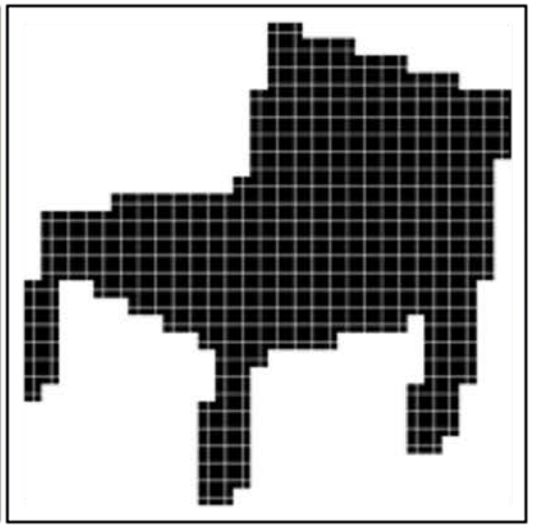


Predict a mask for each of C classes:
C x 28 x 28

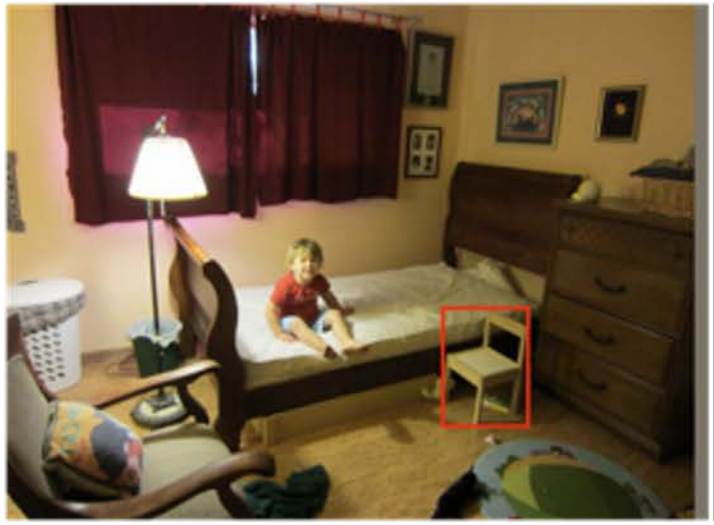
He et al, "Mask R-CNN", ICCV 2017

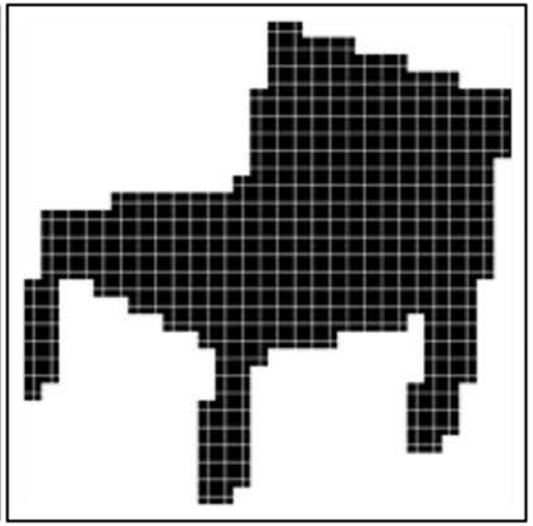
Mask R-CNN: Example Training Targets

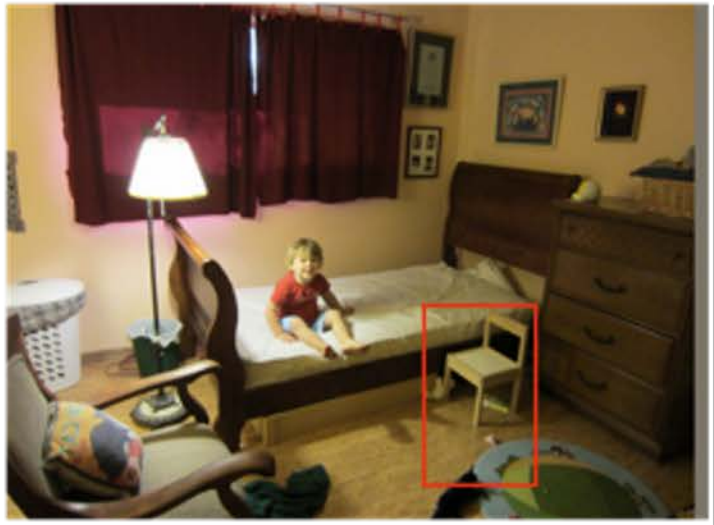


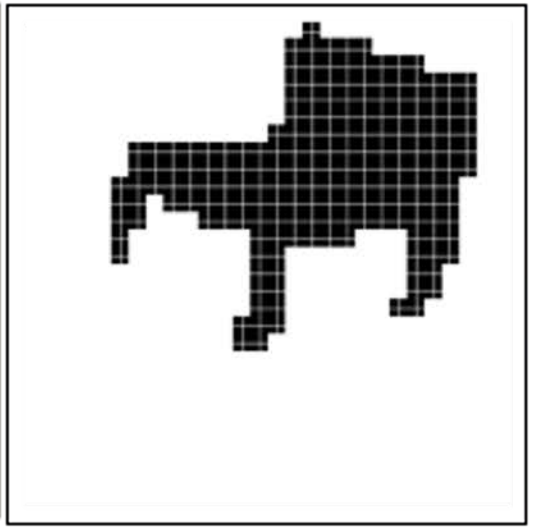


Mask R-CNN: Example Training Targets

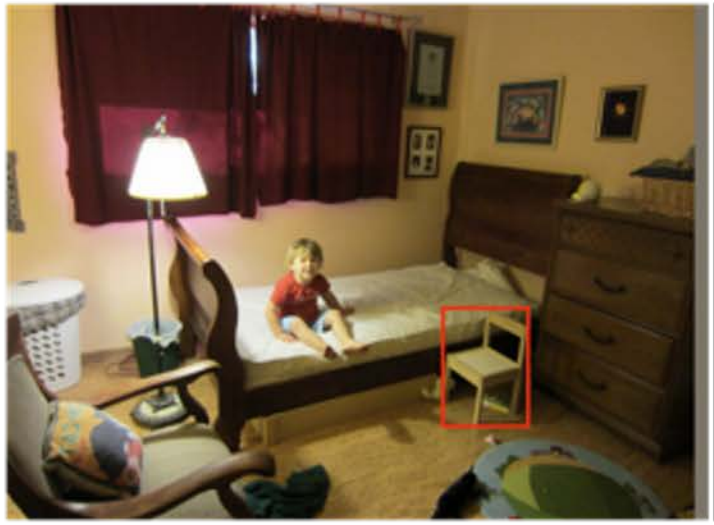
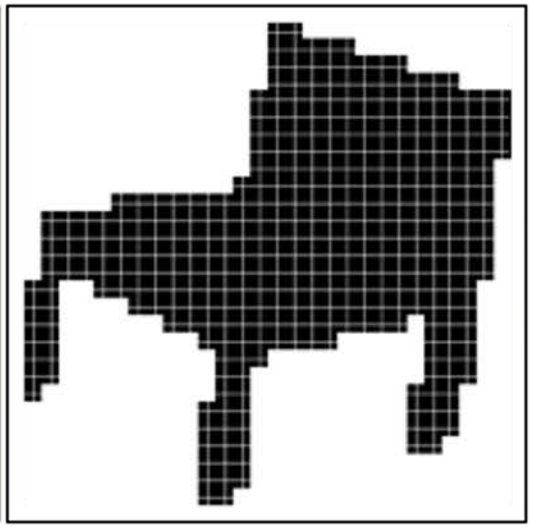
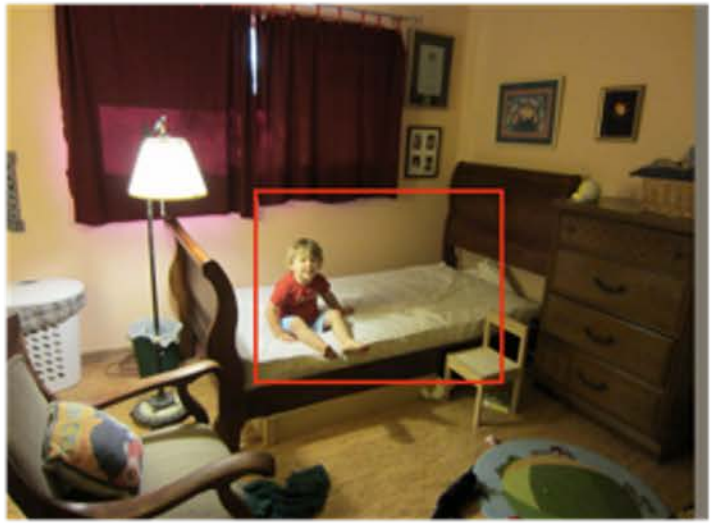
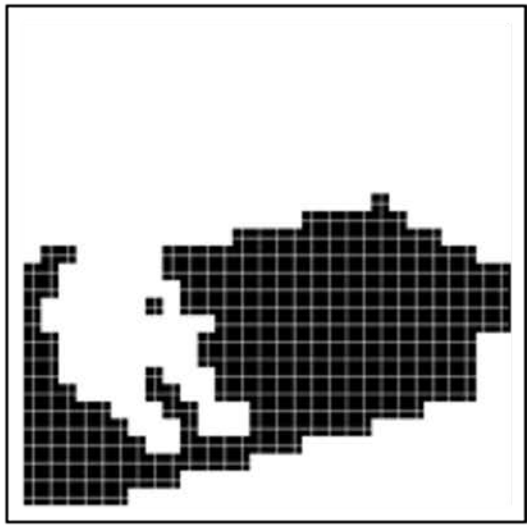


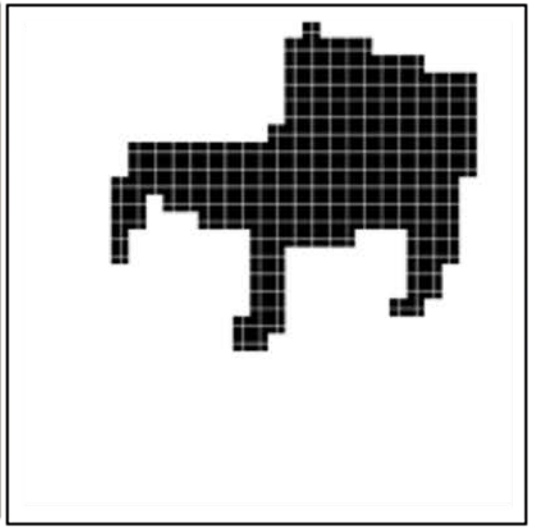




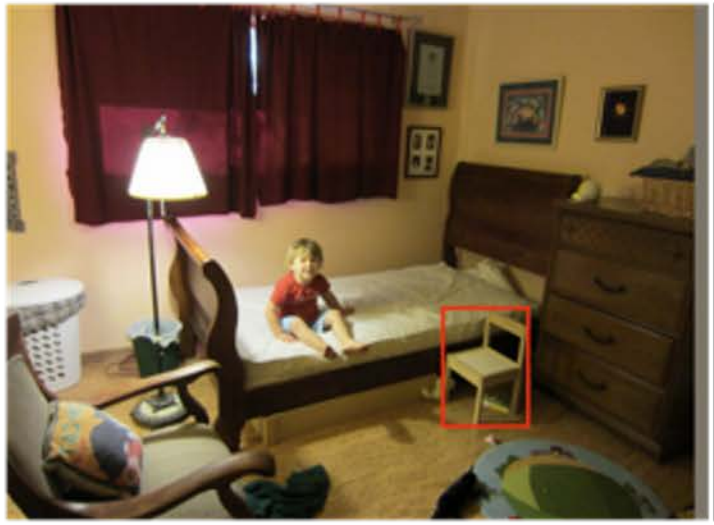
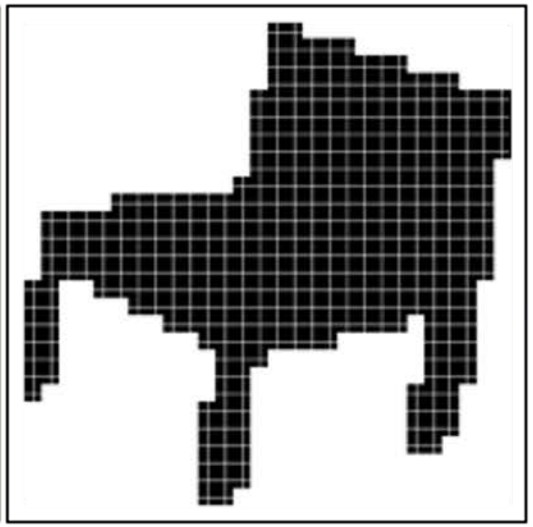
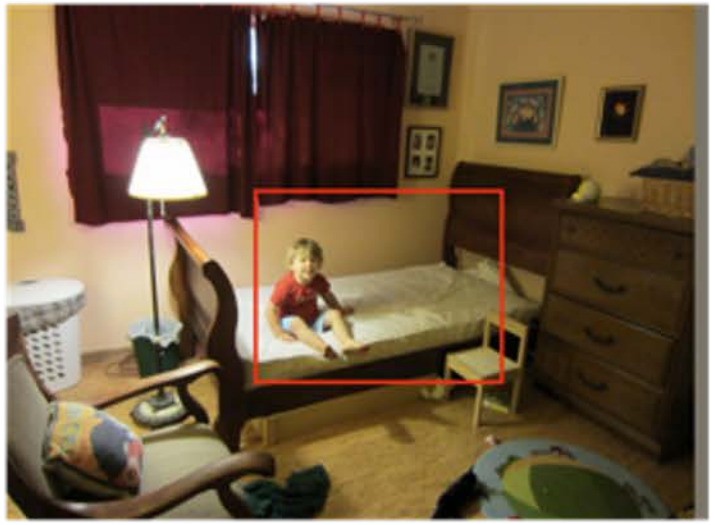
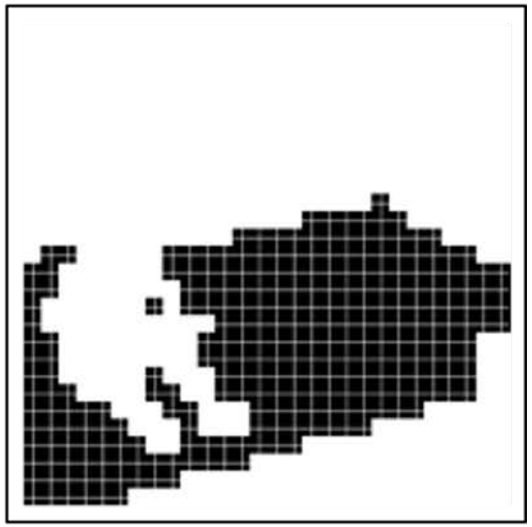


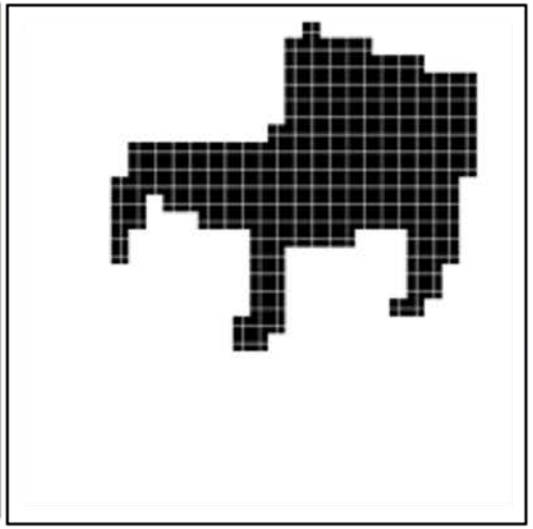
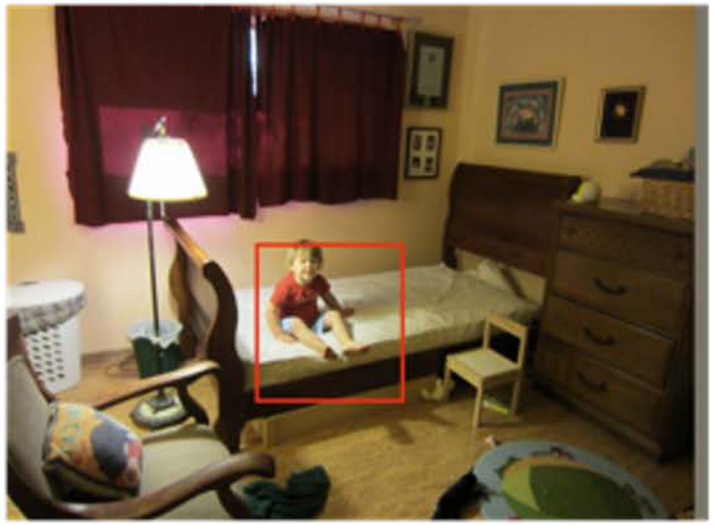
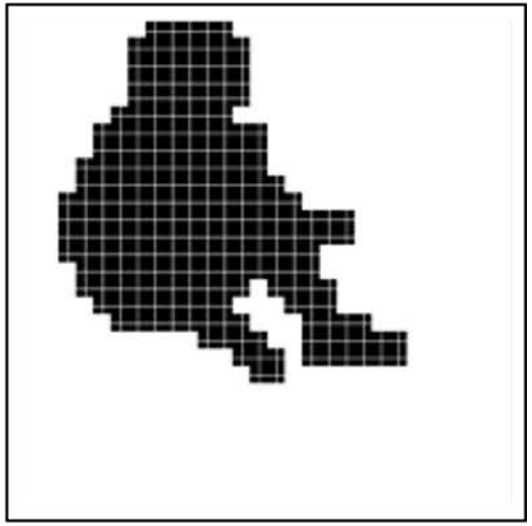
Mask R-CNN: Example Training Targets

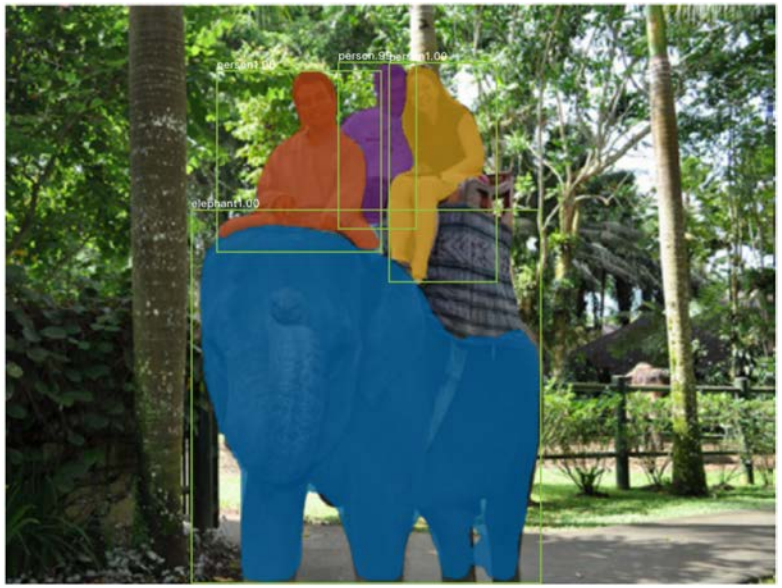


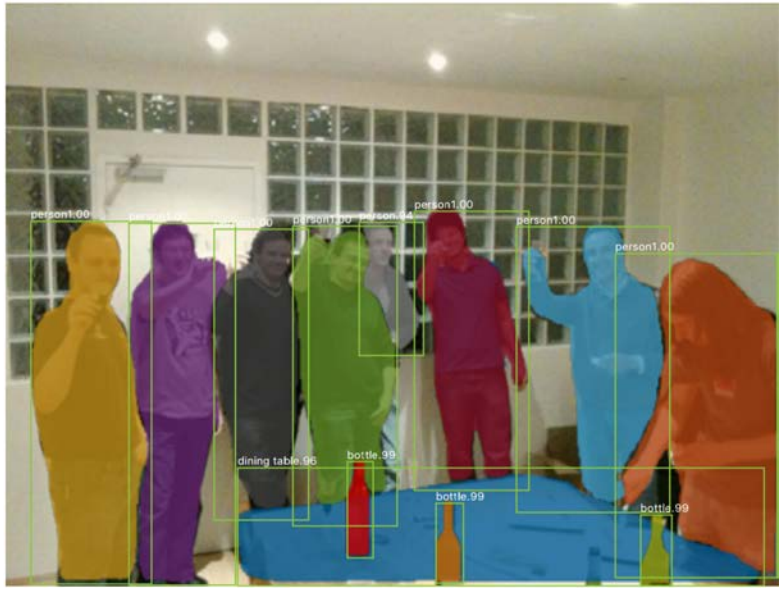
Mask R-CNN: Example Training Targets

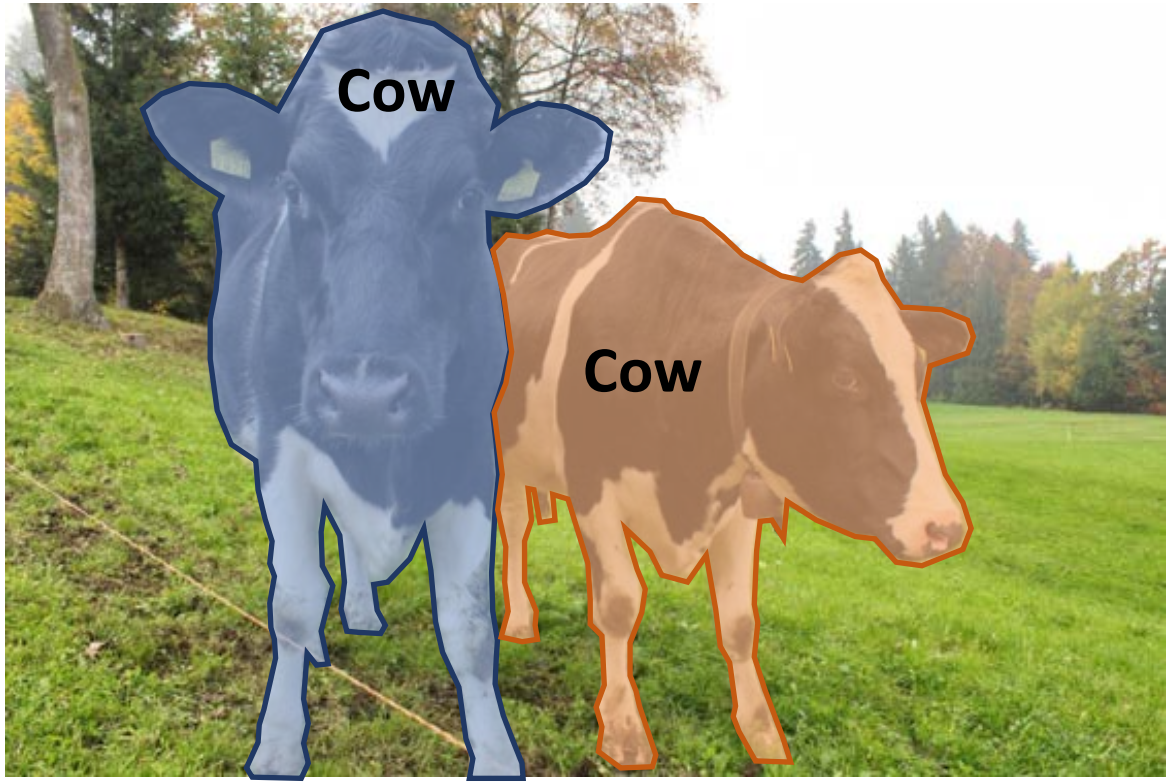
Mask R-CNN: Very Good Results!



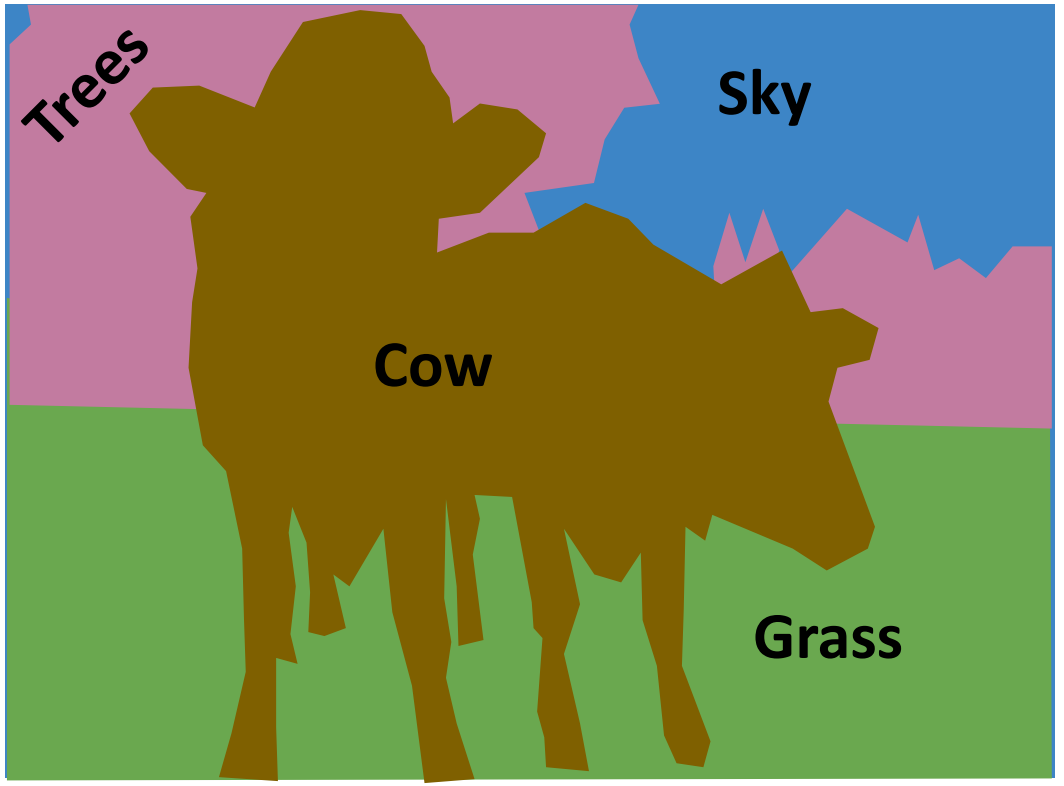


Beyond Instance Segmentation

Instance Segmentation: Separate object instances, but only things



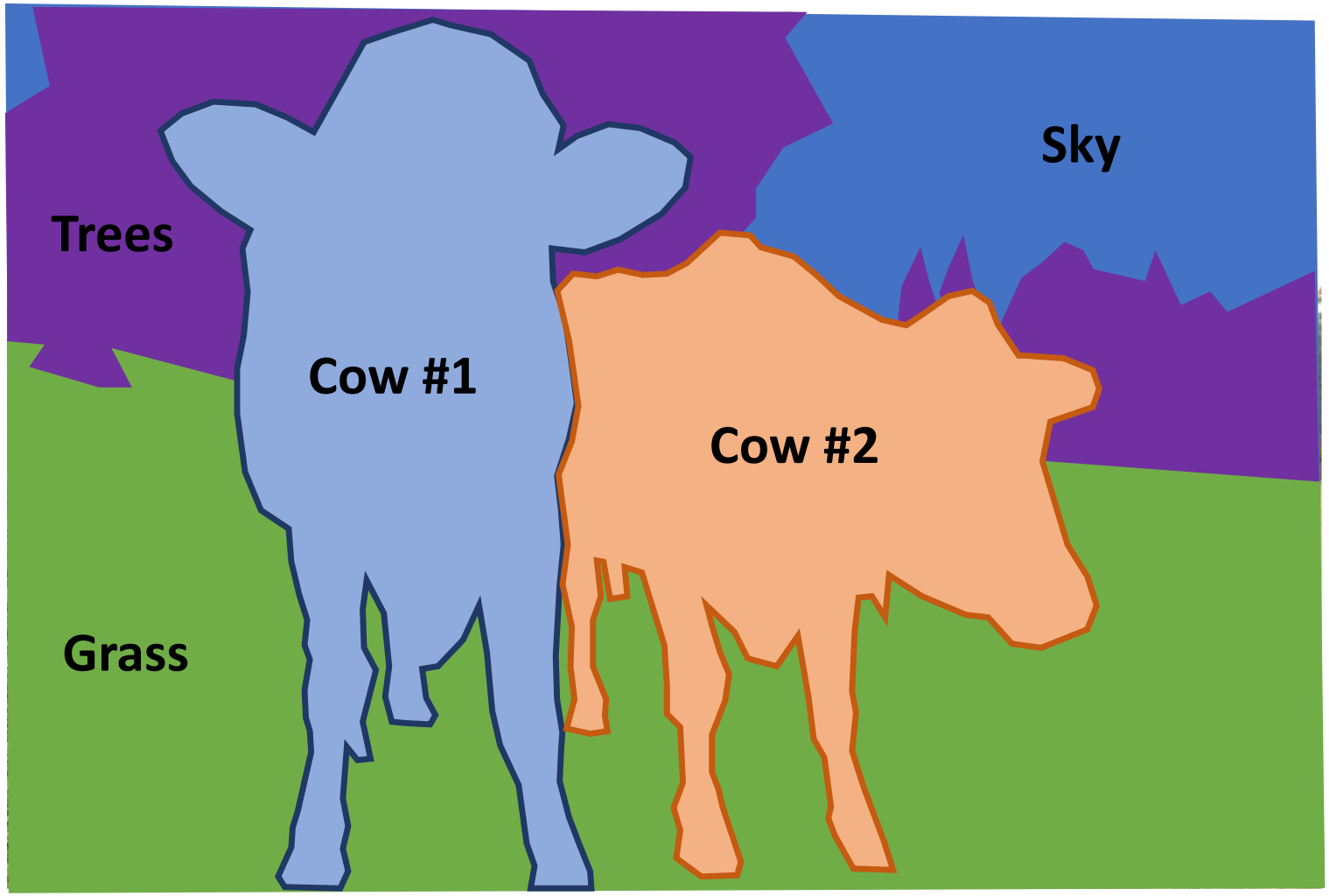
Semantic Segmentation: Identify both things and stuff, but doesn't separate instances



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**

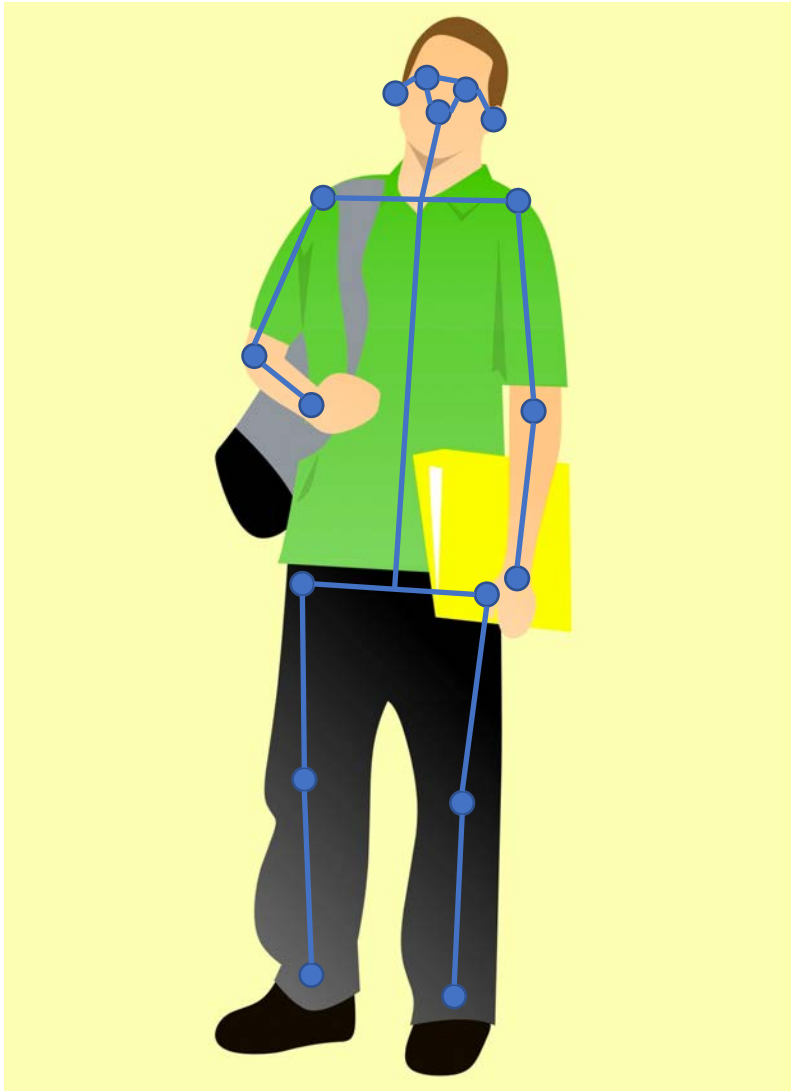
Kirillov et al, "Panoptic Feature Pyramid Networks", CVPR 2019

Beyond Instance Segmentation: Human Keypoints

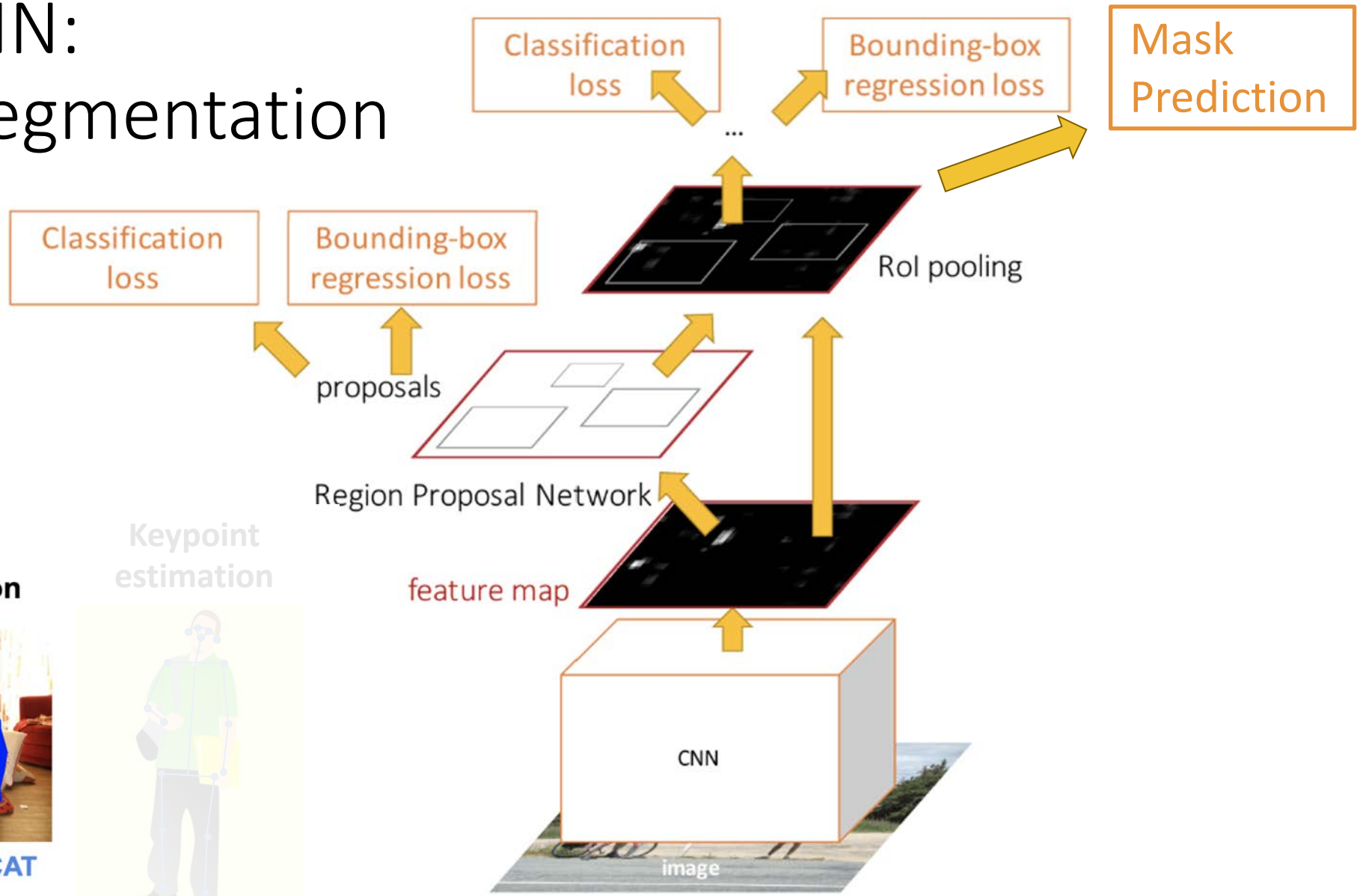
Represent the pose of a human
by locating a set of **keypoints**

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



Object Detection

DOG, DOG, CAT

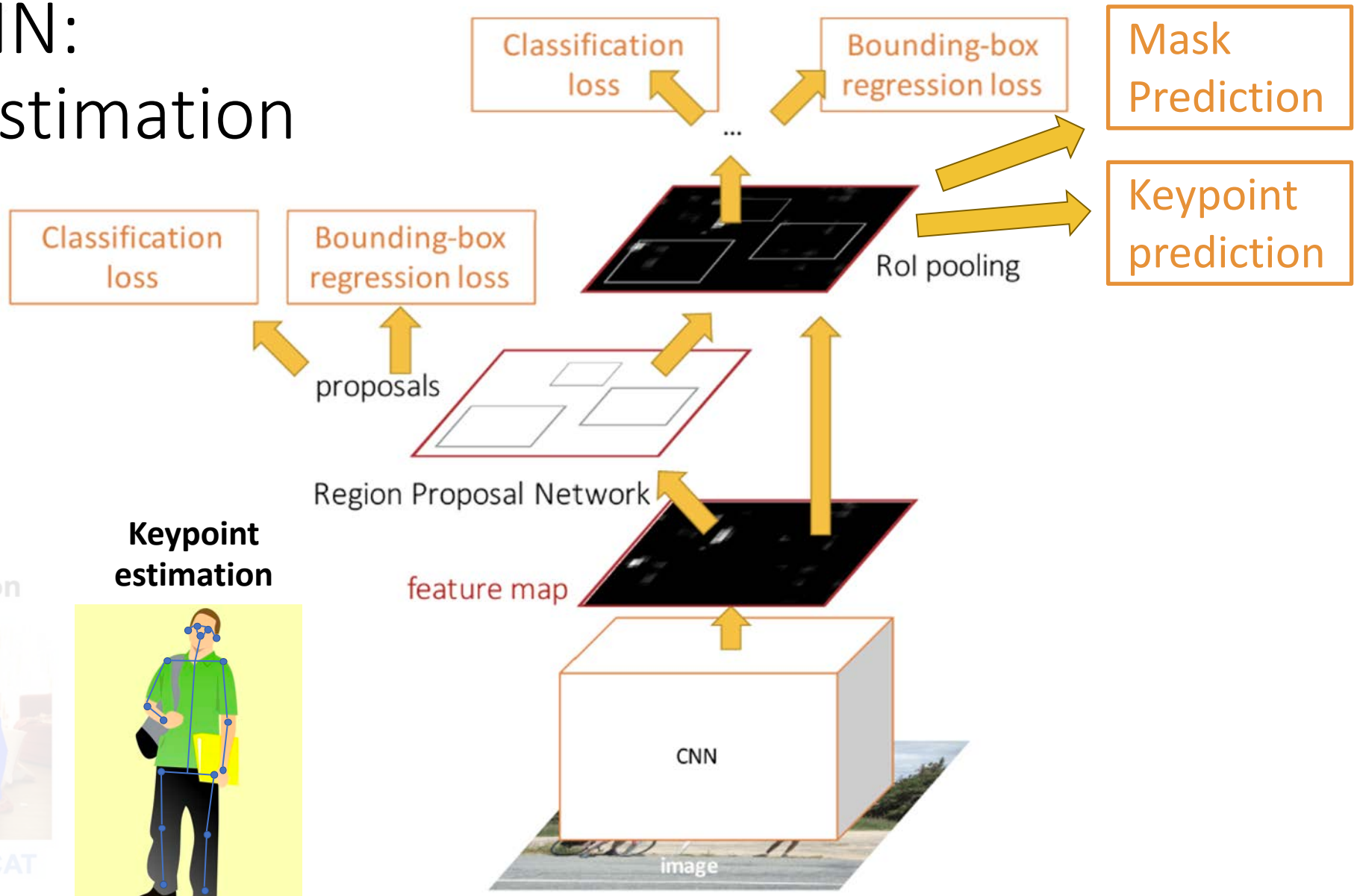
Instance Segmentation

DOG, DOG, CAT

Keypoint estimation

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

Mask R-CNN: Keypoint Estimation



Object Detection

DOG, DOG, CAT

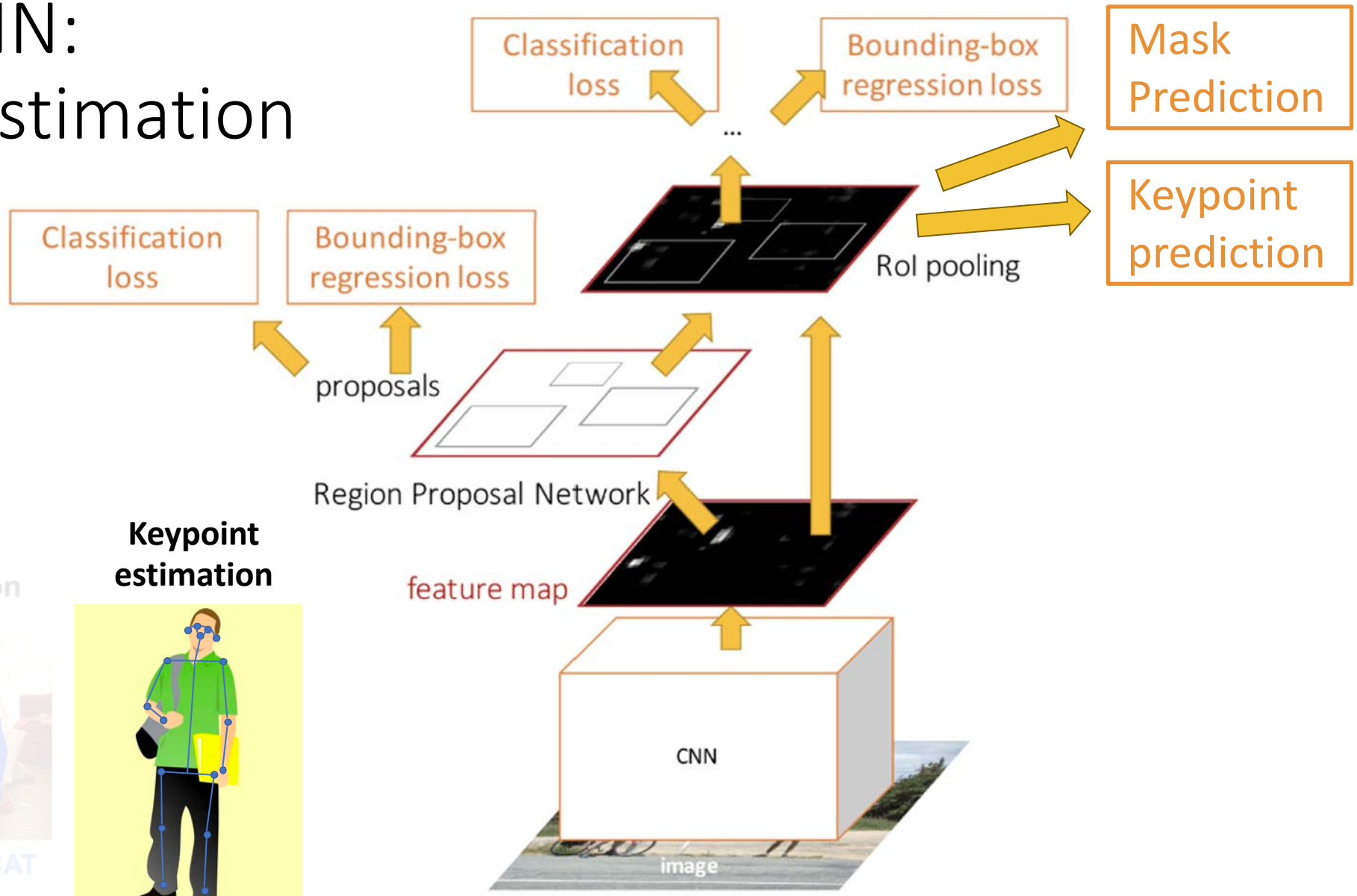
Instance Segmentation

DOG, DOG, CAT

Keypoint estimation

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

Mask R-CNN: Keypoint Estimation



Object Detection

DOG, DOG, CAT

Instance Segmentation

DOG, DOG, CAT

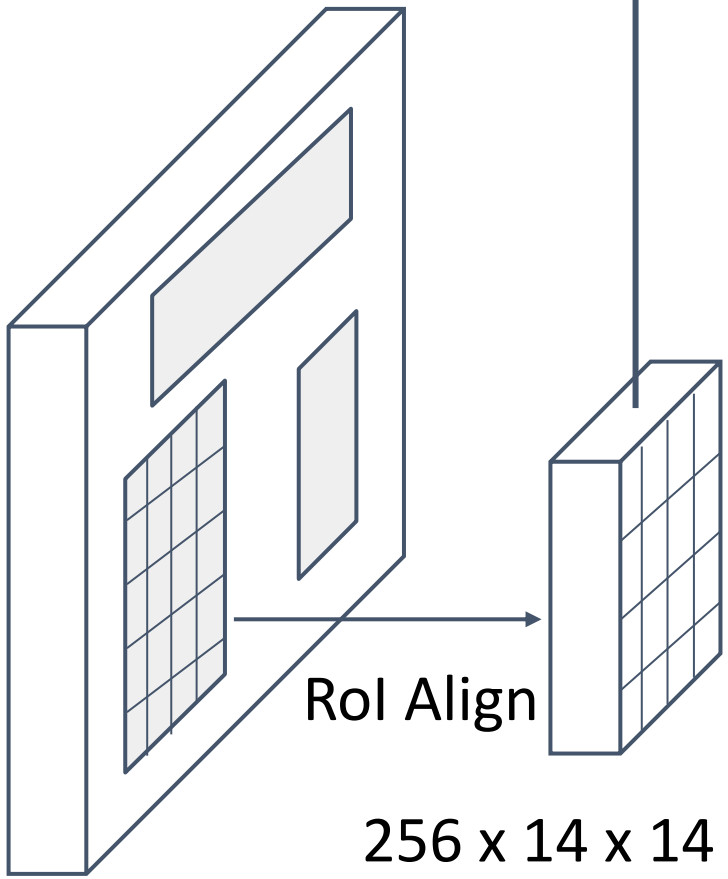
Keypoint estimation

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

CNN
+RPN

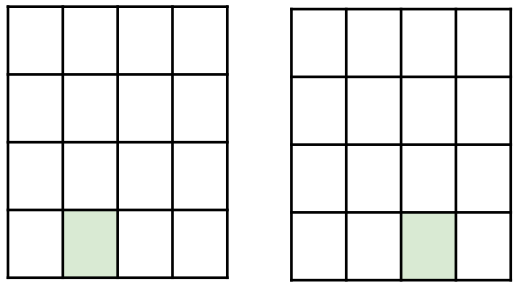


RoI Align

256 x 14 x 14

Conv…

One mask for each of
the K different keypoints



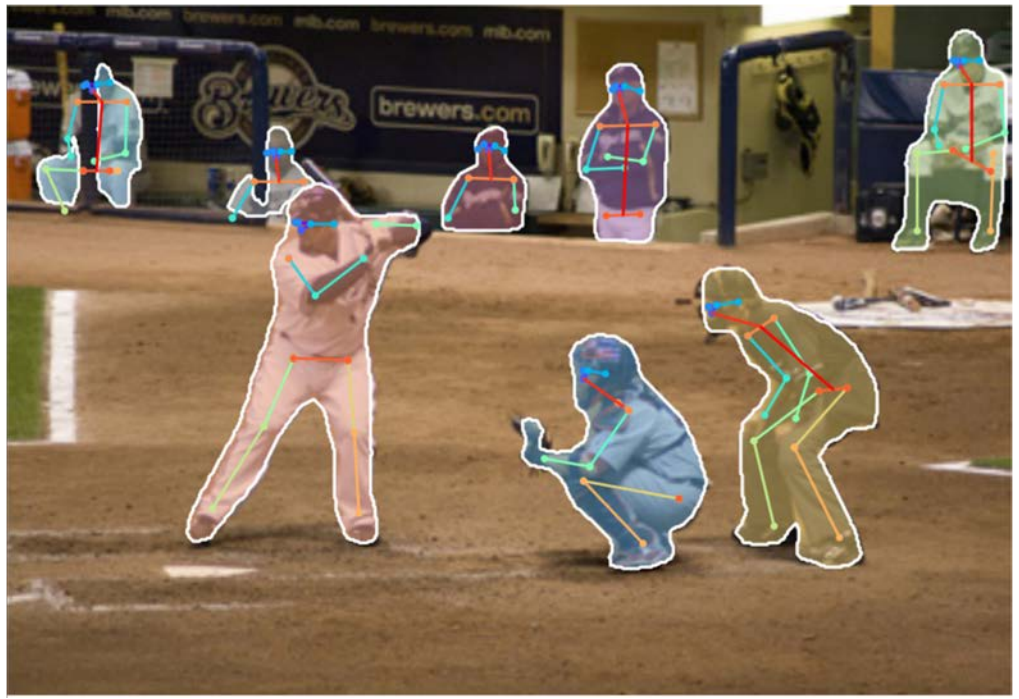
…

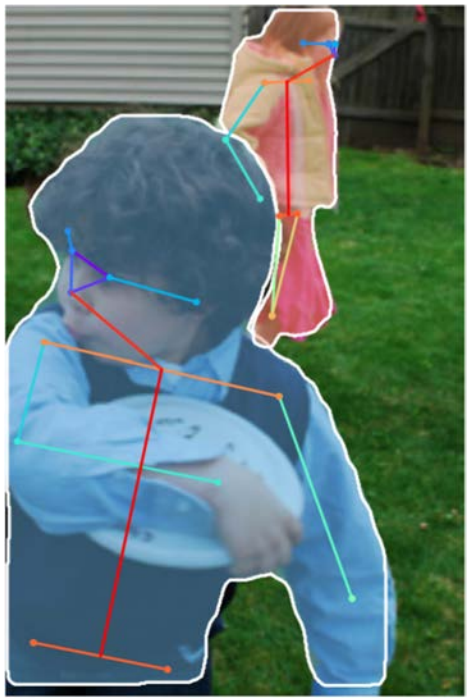
Keypoint masks:
K x 56 x 56

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

He et al, “Mask R-CNN”, ICCV 2017

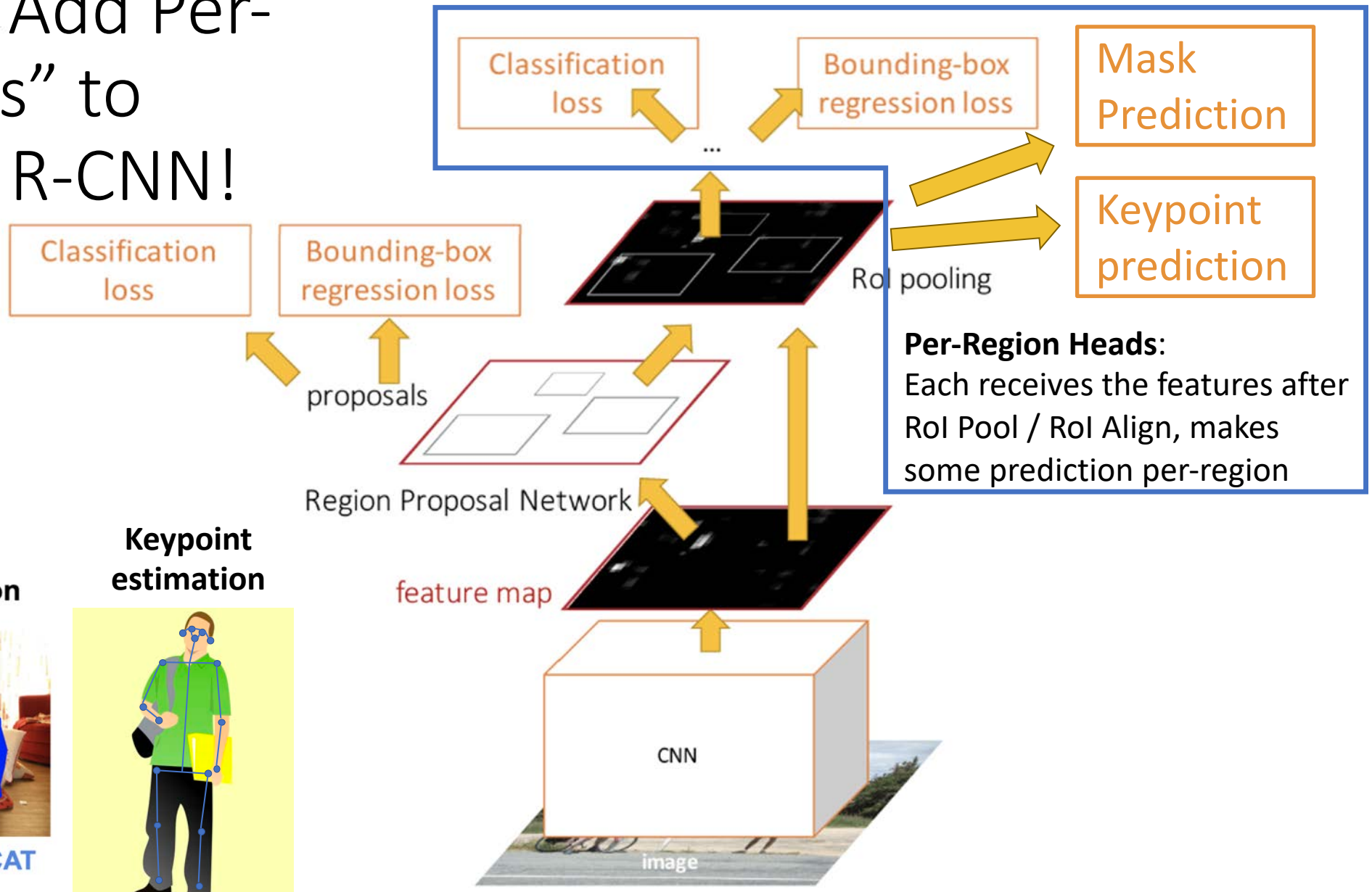
Joint Instance Segmentation and Pose Estimation





He et al, “Mask R-CNN”, ICCV 2017

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



Object Detection

DOG, DOG, CAT

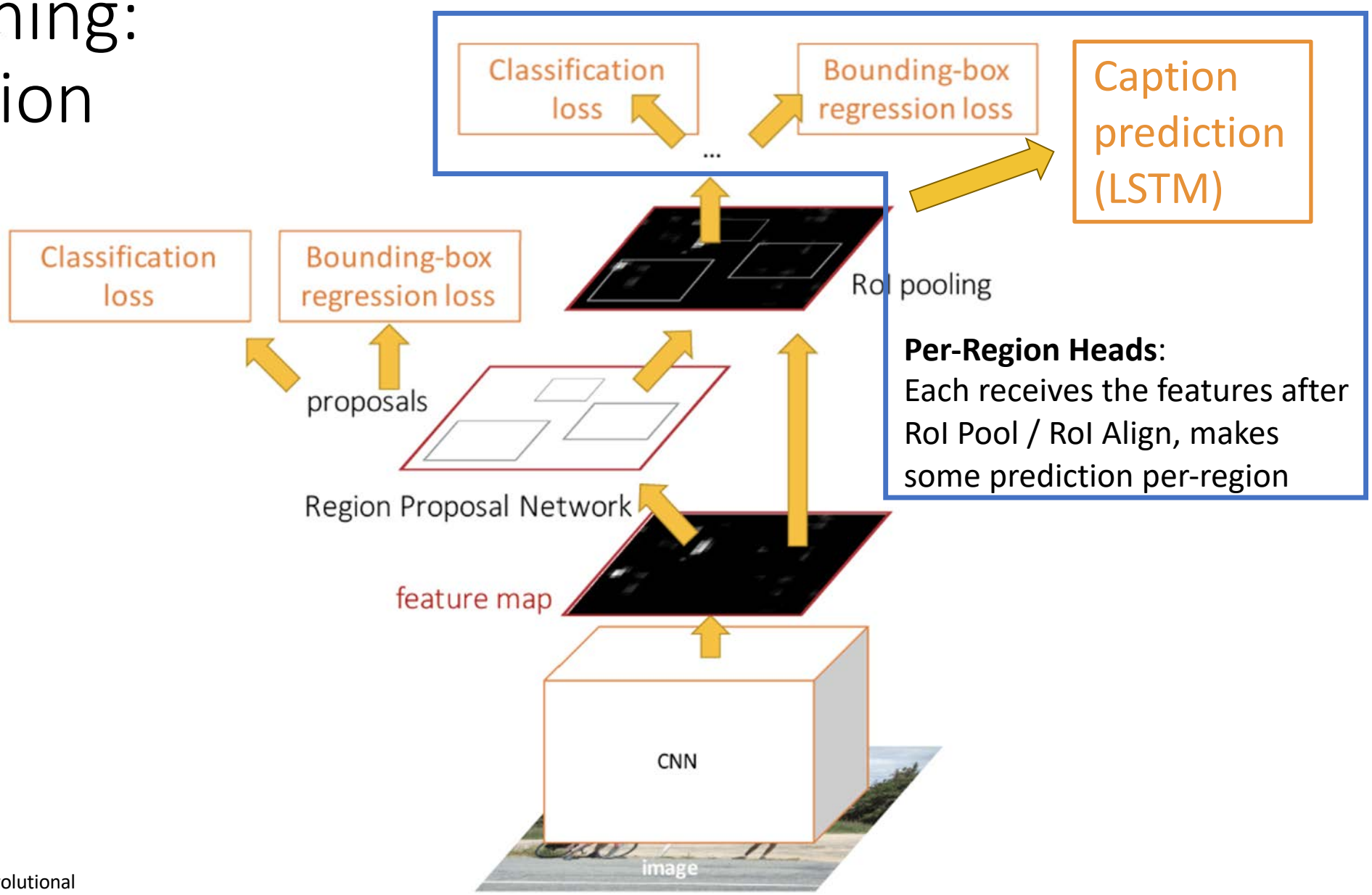
Instance Segmentation

DOG, DOG, CAT

Keypoint estimation

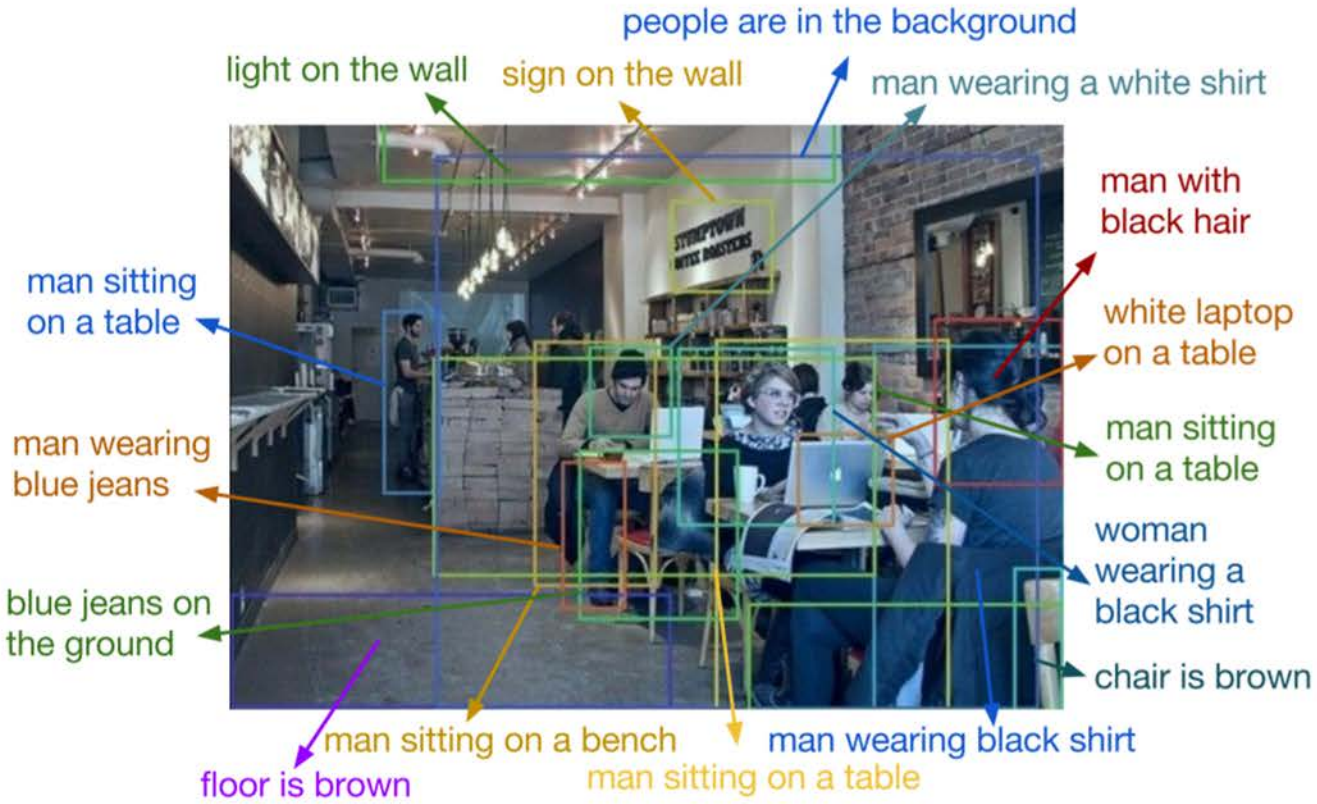
He et al, "Mask R-CNN", ICCV 2017

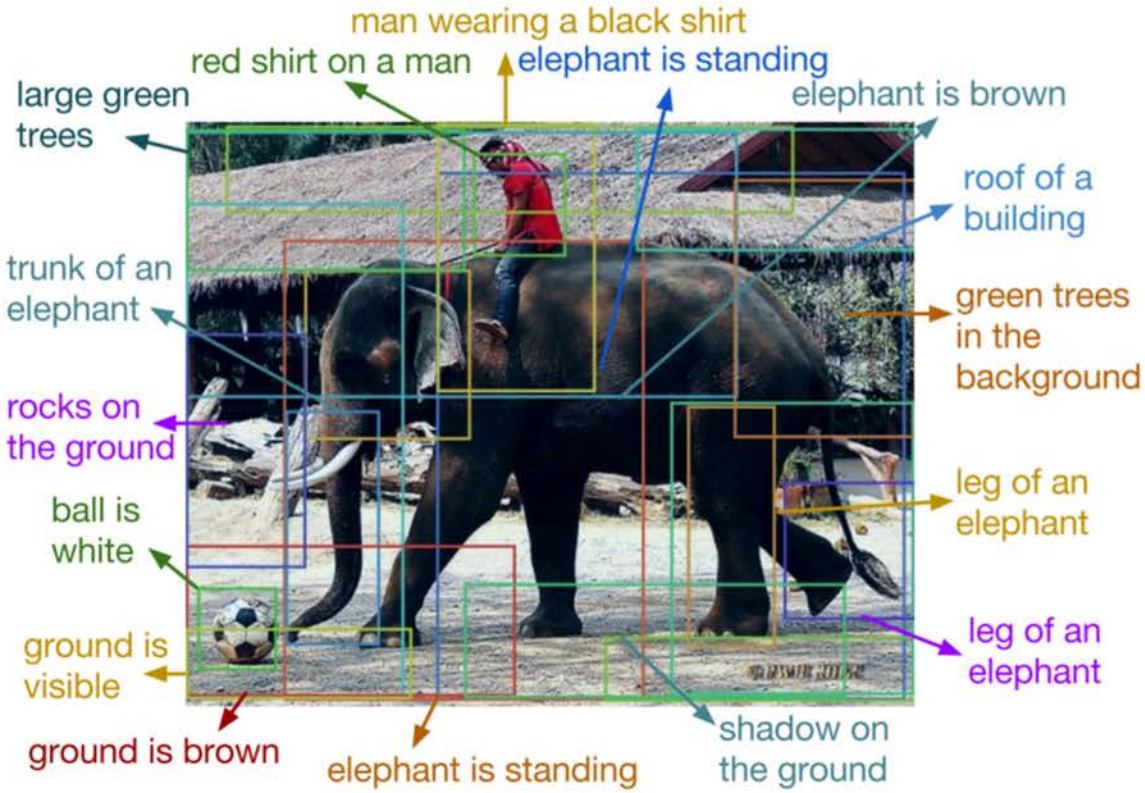
Dense Captioning: Predict a caption per region!



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

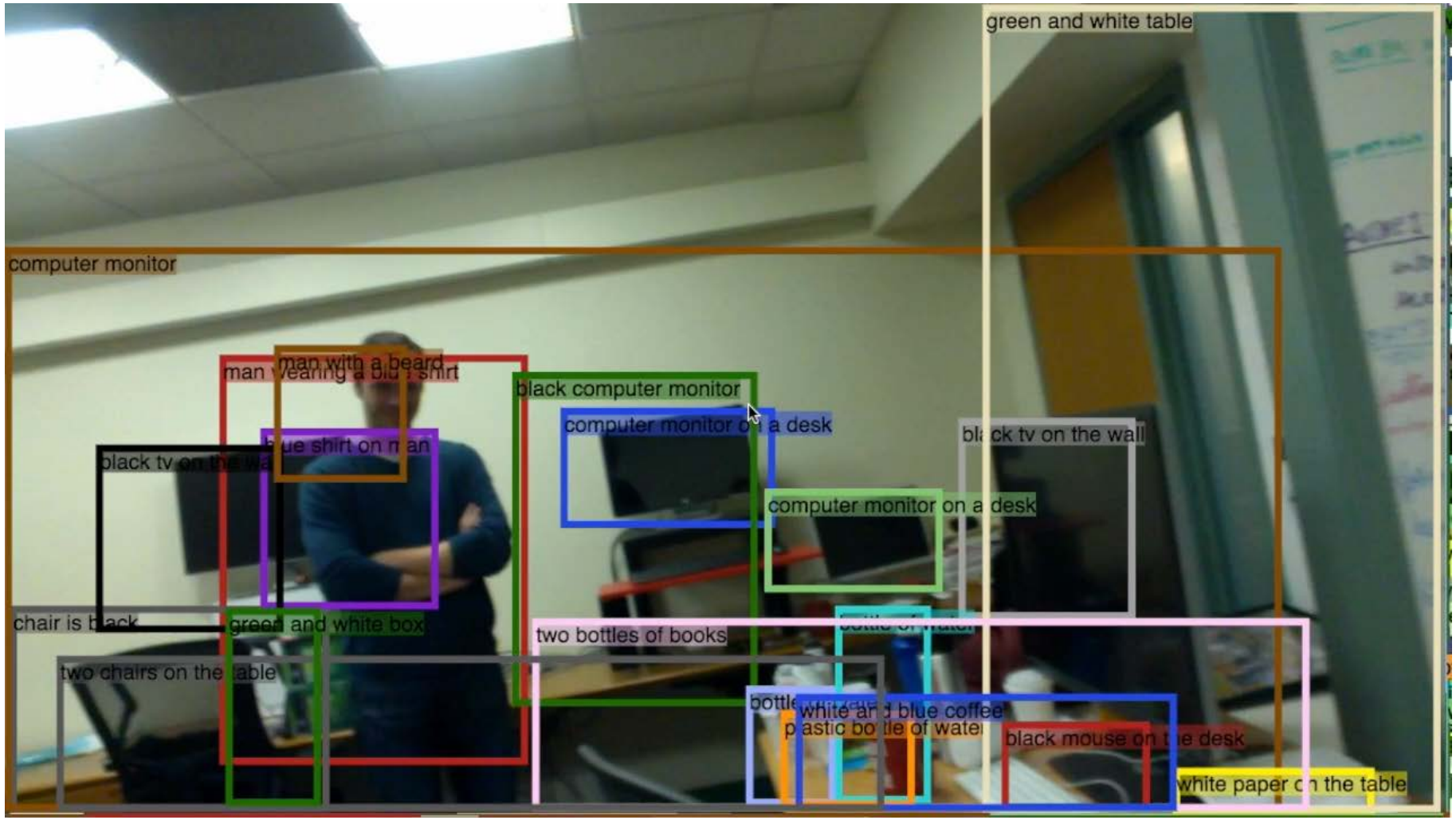
Dense Captioning





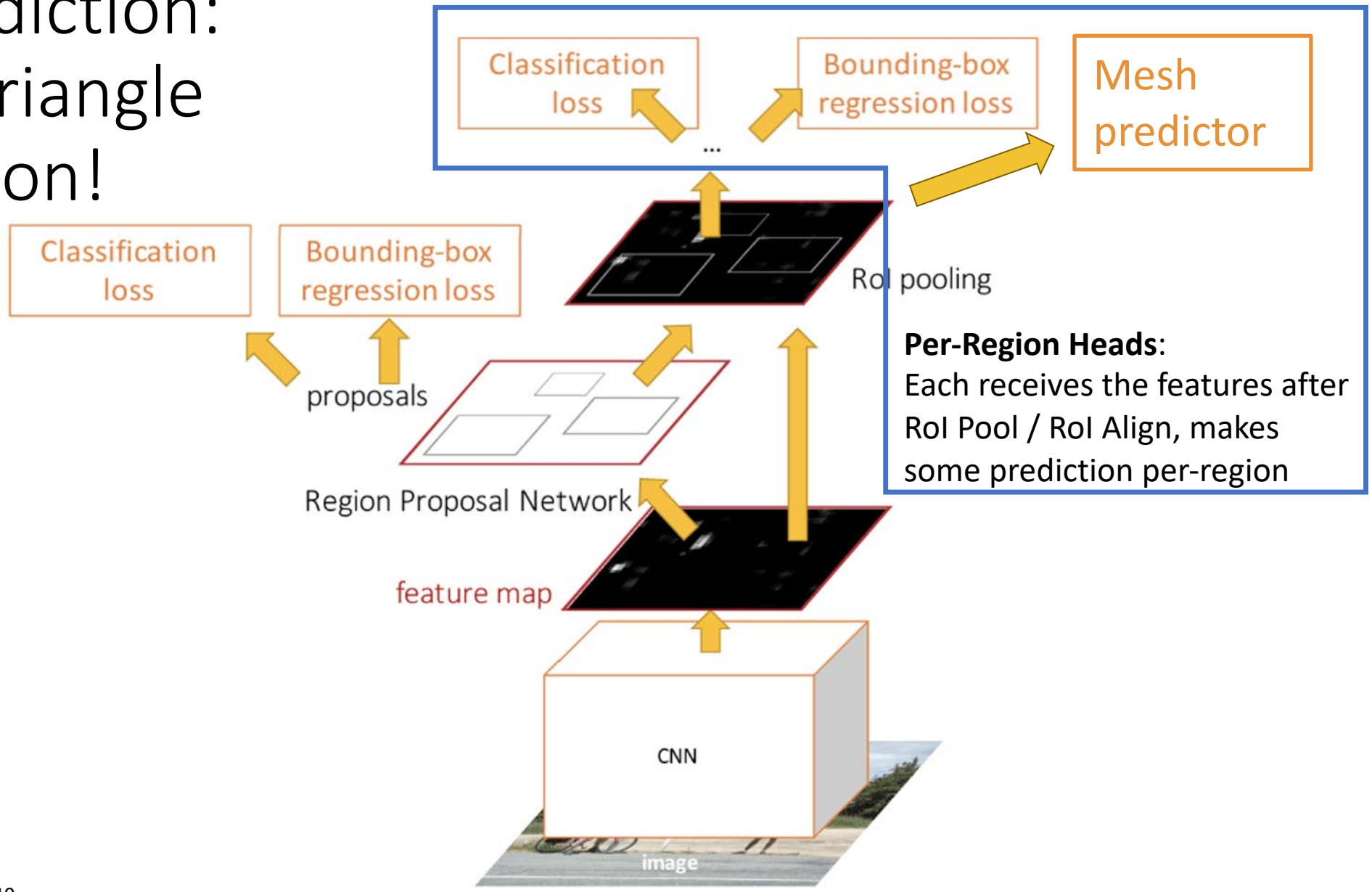
Johnson, Karpathy, and Fei-Fei, "DenseCap: Fully Convolutional Localization Networks for Dense Captioning", CVPR 2016

Dense Captioning



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

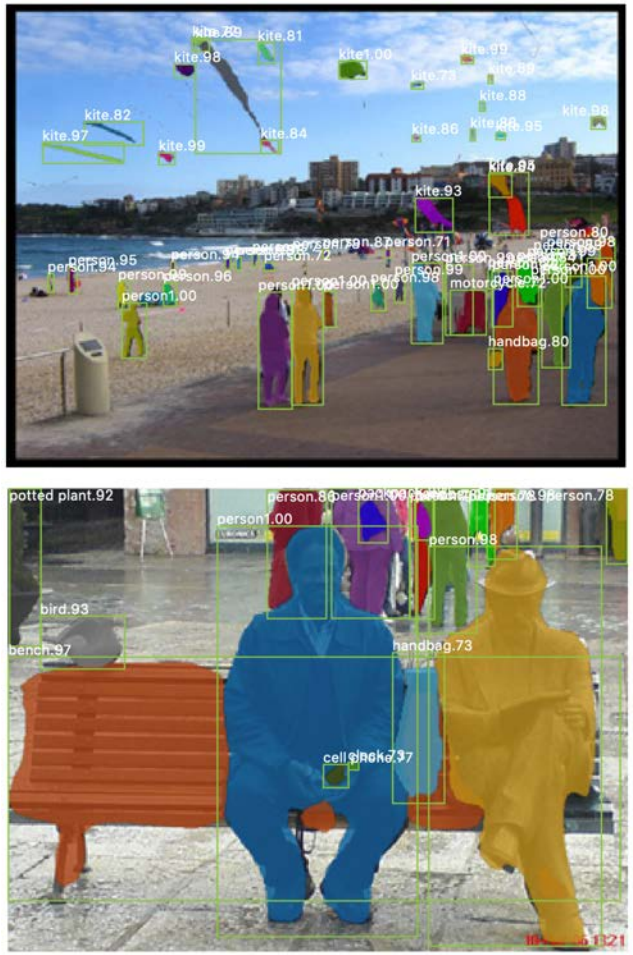
3D Shape Prediction: Predict a 3D triangle mesh per region!



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

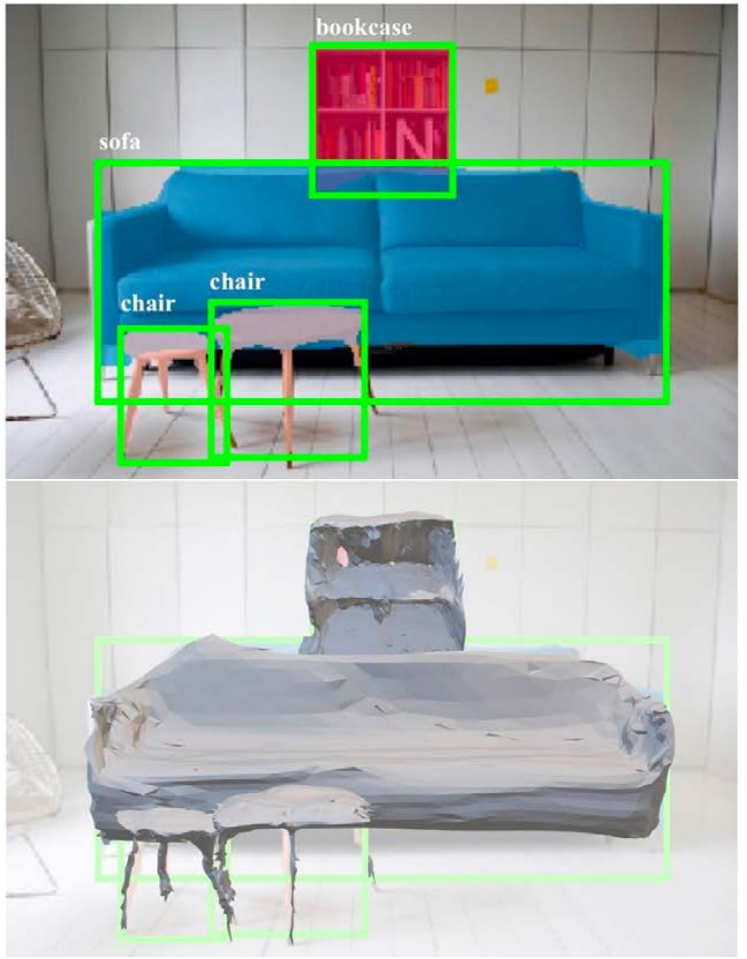
3D Shape Prediction: Mask R-CNN + Mesh Head

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



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

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



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

More details next time!

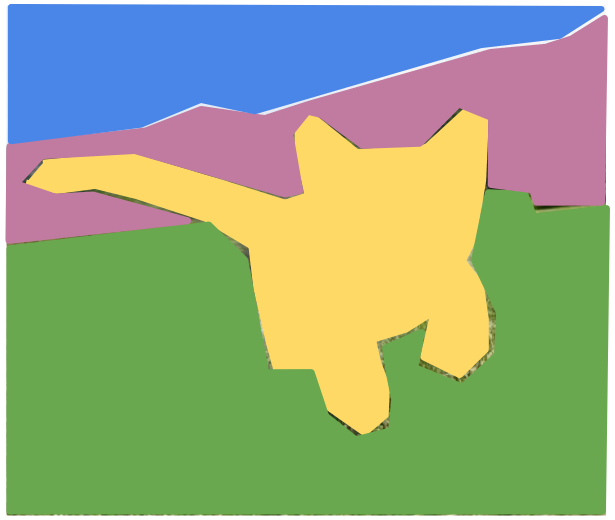
Summary: Many Computer Vision Tasks!

Classification

CAT

No spatial extent

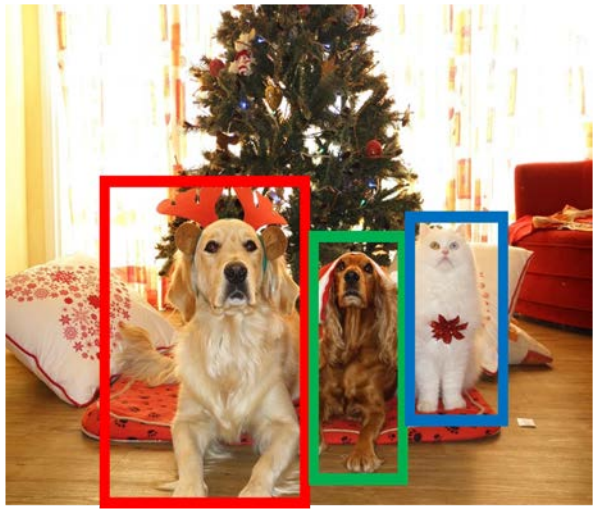
Semantic Segmentation



GRASS, **CAT**, **TREE**, **SKY**

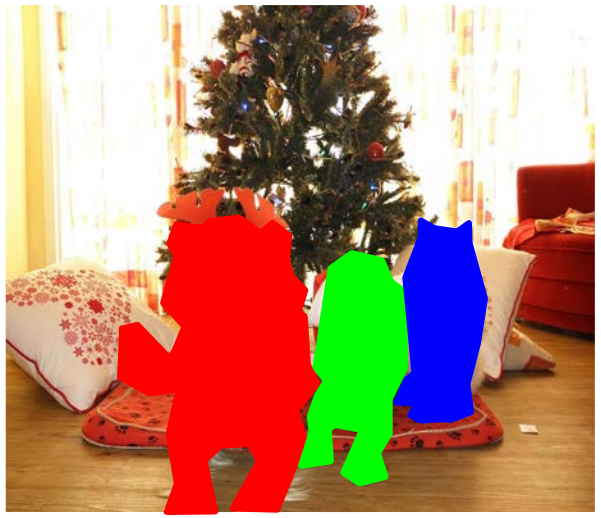
No objects, just pixels

Object Detection



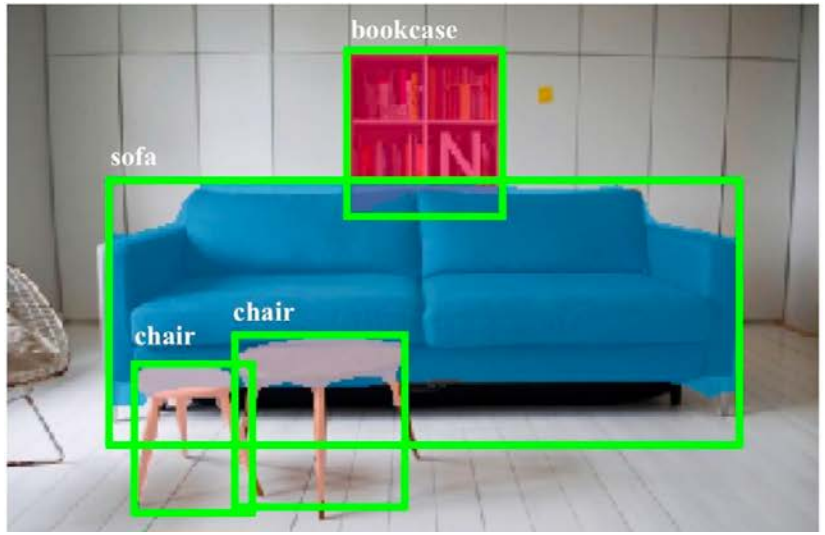
DOG, **DOG**, **CAT**

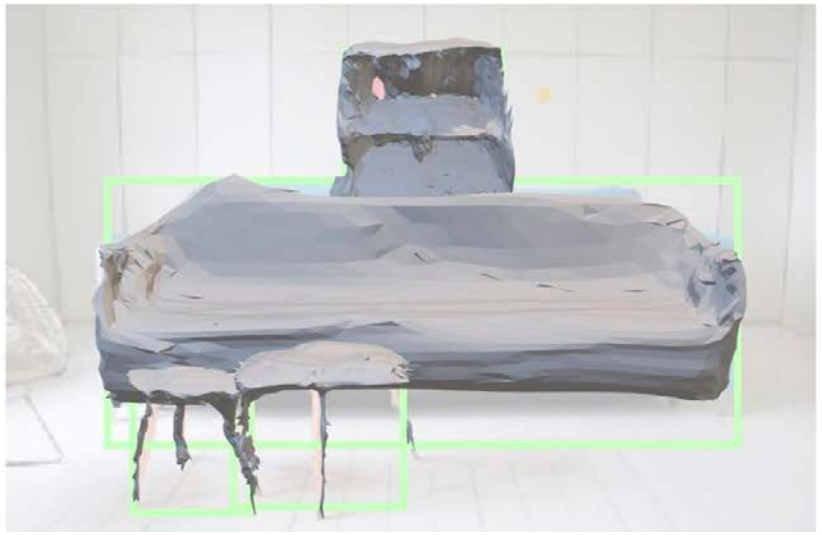
Instance Segmentation



DOG, **DOG**, **CAT**

Multiple Objects





Next Time: 3D Vision