12. Object Detection and Image Segmentation

GEV6135 Deep Learning for Visual Recognition and Applications

Kibok Lee

Assistant Professor of

Applied Statistics / Statistics and Data Science

Dec 1, 2022

연세대학교
YONSEI UNIVERSITY

Assignment 6

- Due **Friday 12/2, 11:59pm KST**
- Convolutional networks
  - Modularized implementation (loss is given!)
  - BatchNorm is given, but should be plugged into DeepConvNet

- Before submitting your work, we recommend you
  - Re-download clean files
  - Copy-paste your solution to clean py
  - Re-run clean ipynb only once
- If you feel difficult, consider to take **option 2**.

Assignment 7

- Due **Wednesday 12/14, 11:59pm KST**
- PyTorch autograd and nn
  - 3 different abstraction levels
  - Residual networks

- Before submitting your work, we recommend you
  - Re-download clean files
  - Copy-paste your solution to clean py
  - Re-run clean ipynb only once
- If you feel difficult, consider to take **option 2**.

So Far: Image Classification

This image is CC0 public domain

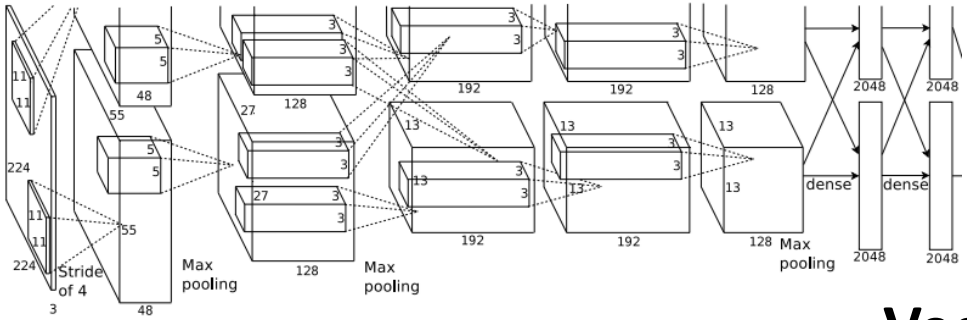


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Vector:
4096

Fully-Connected:
4096 to 1000

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

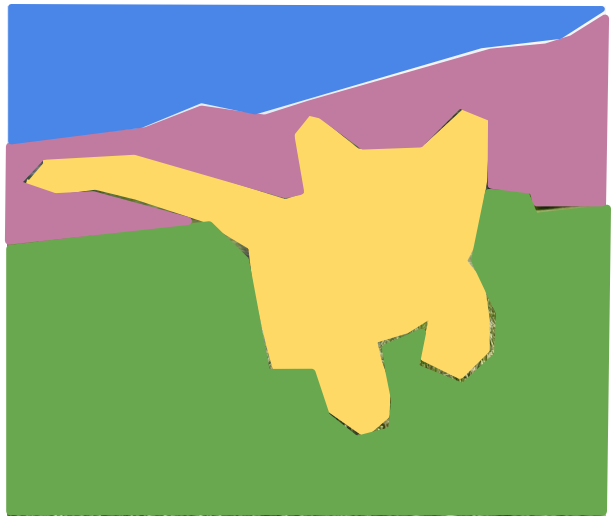
Computer Vision Tasks

Classification

CAT

No spatial extent

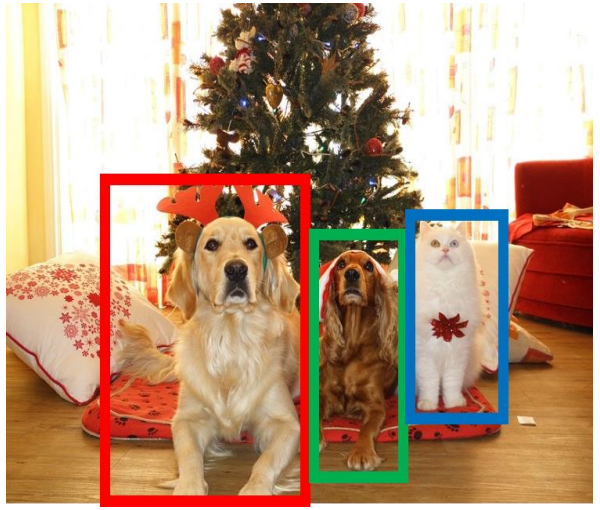
Semantic Segmentation



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

No objects, just pixels

Object Detection



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

Instance Segmentation

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

Multiple Objects

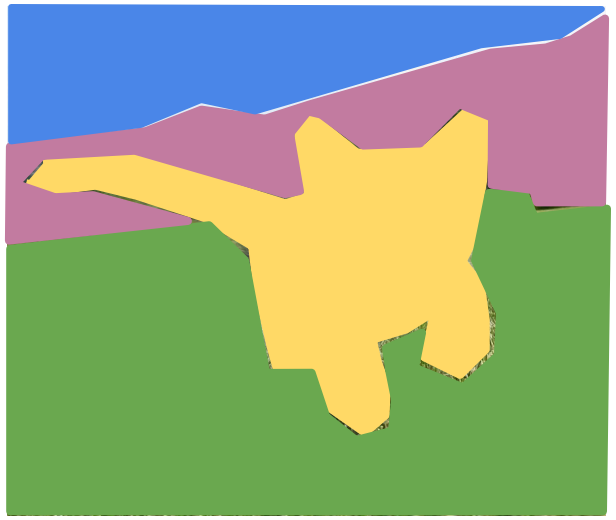
Computer Vision Task: Semantic Segmentation

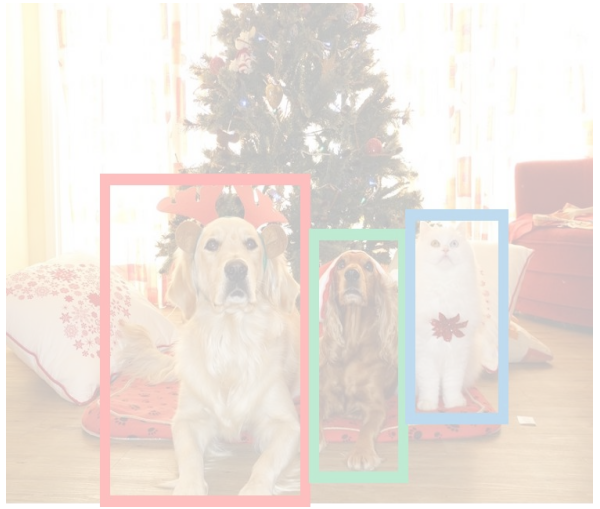
Classification

Semantic Segmentation

Object Detection

Instance Segmentation





CAT

GRASS, CAT, TREE, SKY

DOG, DOG, CAT

DOG, DOG, CAT

No spatial extent

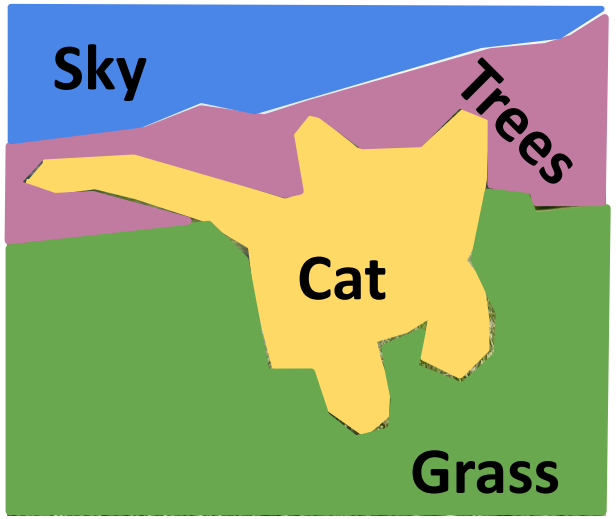
No objects, just pixels

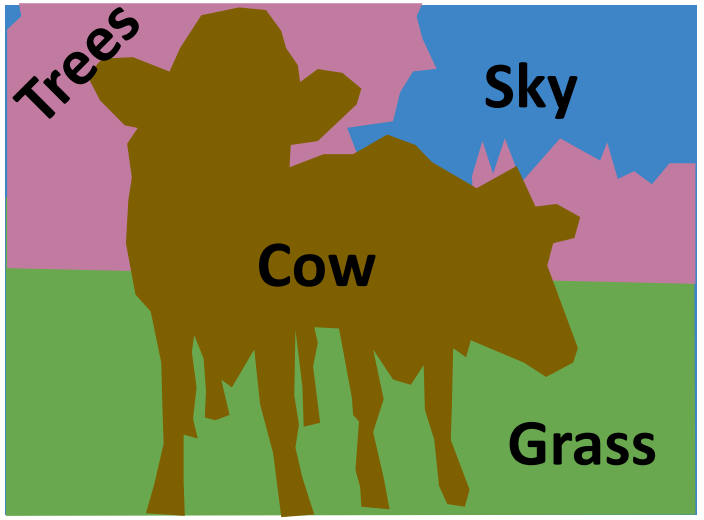
Multiple Objects

Semantic Segmentation

Label each pixel in the image with a category label

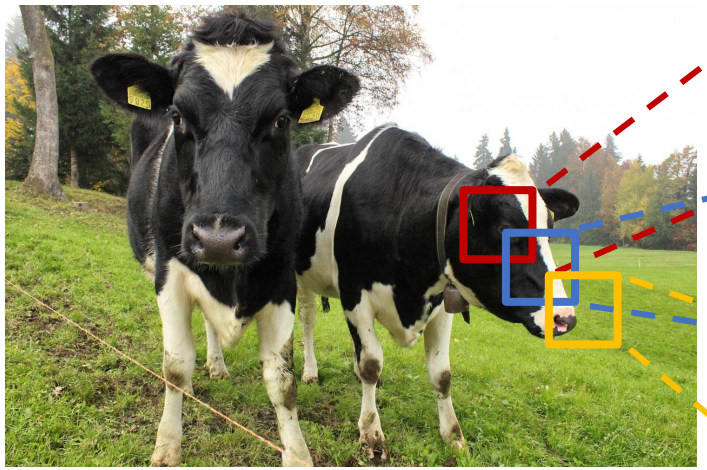
Don’t differentiate instances, only care about pixels



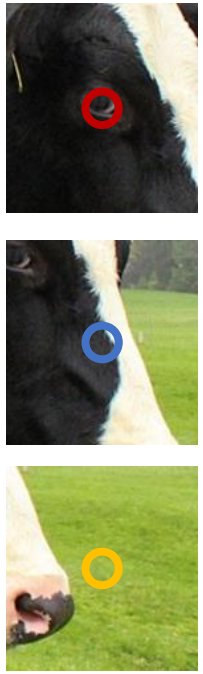


Semantic Segmentation Idea: Sliding Window

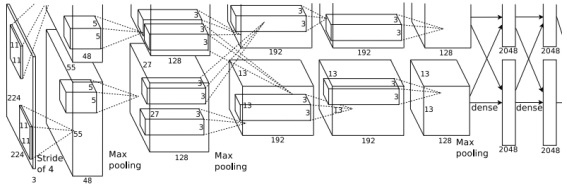
Full image



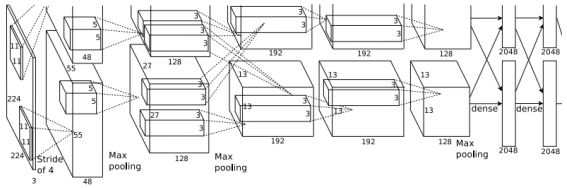
Extract patch



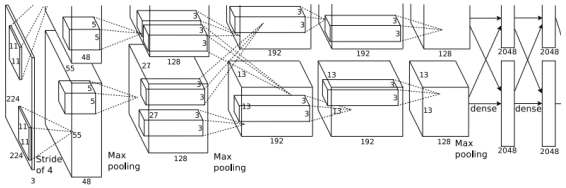
Classify center pixel with CNN



Cow



Cow



Grass

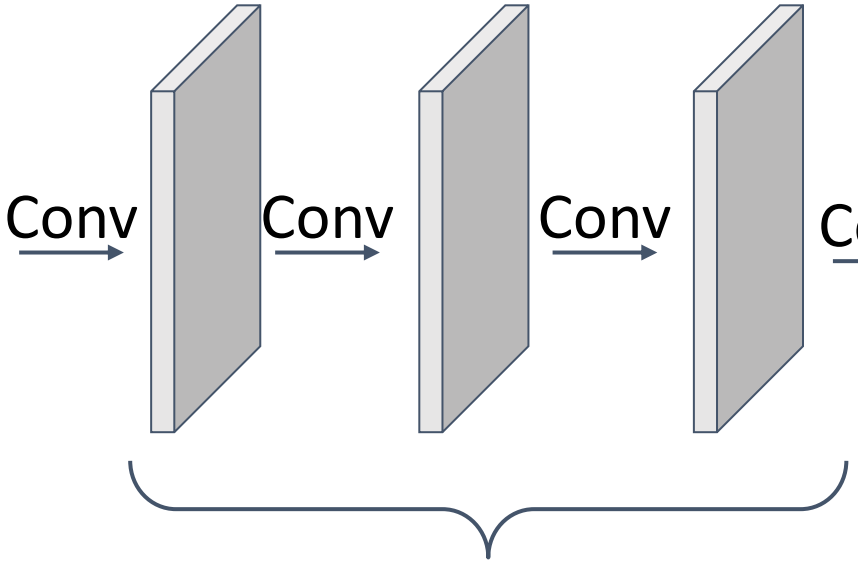
Problem: Very inefficient! Not reusing shared features between overlapping patches

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

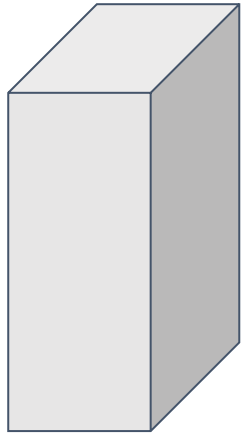
Semantic Segmentation: Fully Convolutional Network

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

argmax

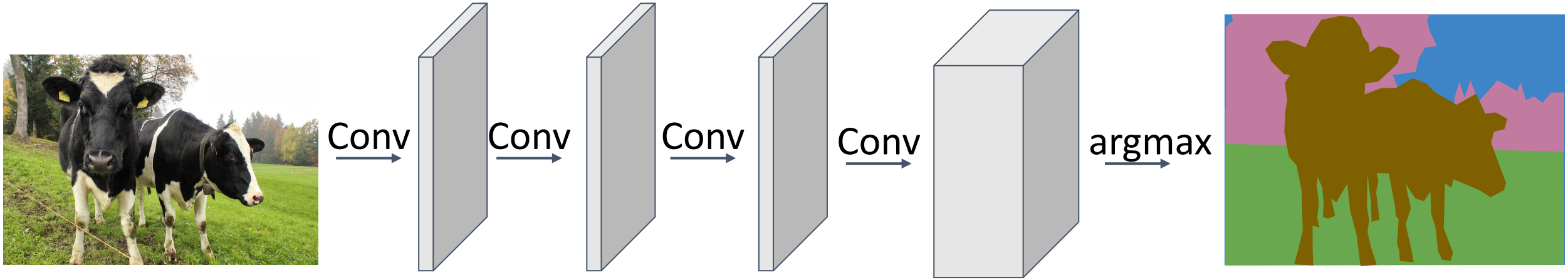
Predictions:
H x W

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

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

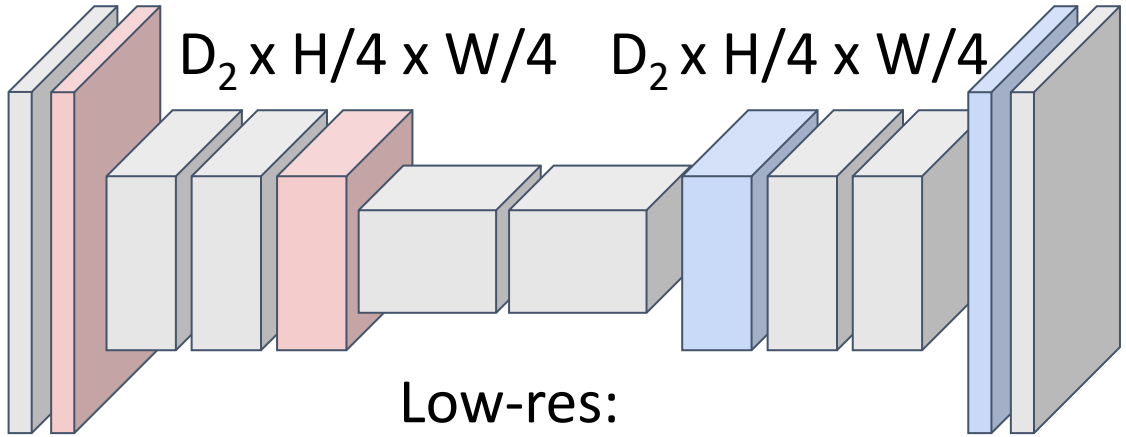
Semantic Segmentation: Fully Convolutional Network

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

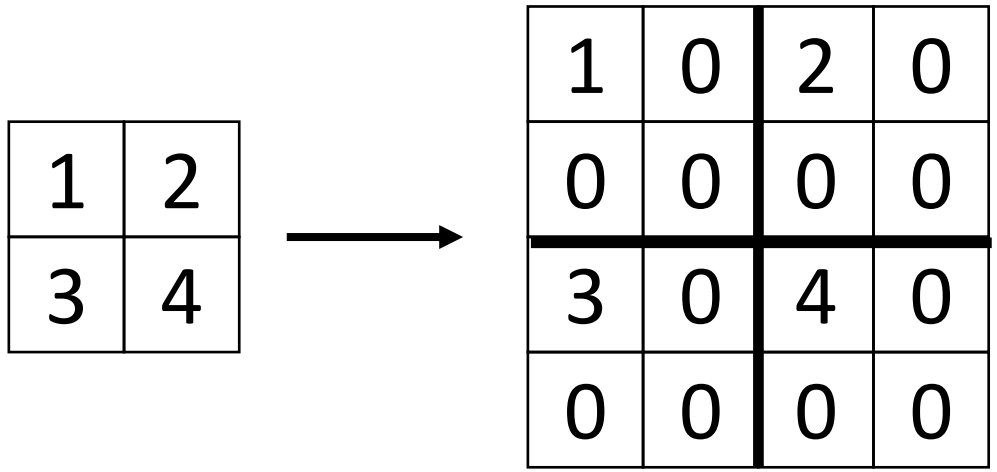


Predictions:
H x W

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”

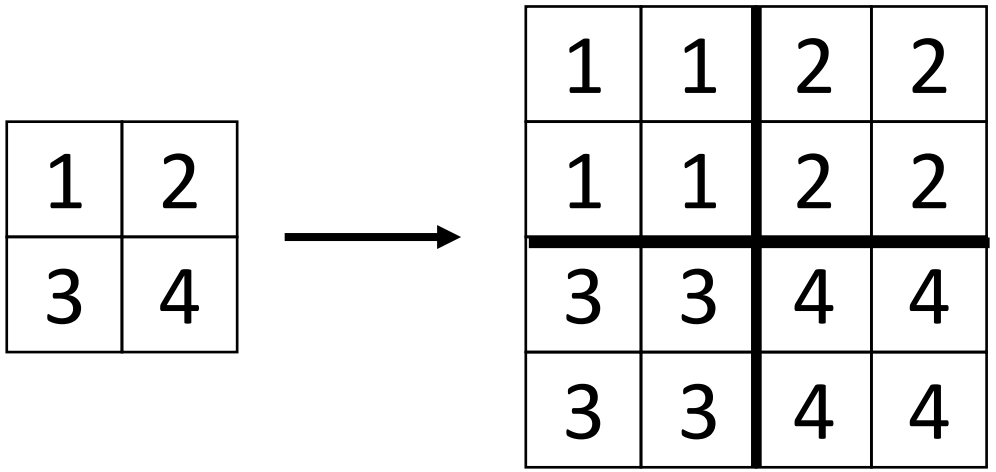
Bed of Nails



Input
C x 2 x 2

Output
C x 4 x 4

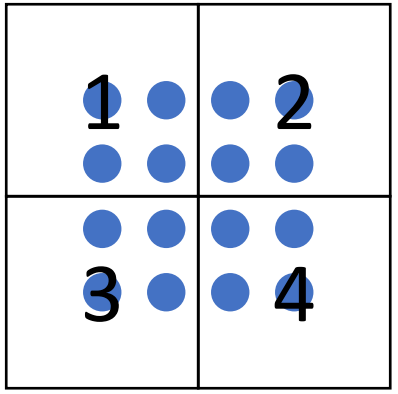
Nearest Neighbor



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

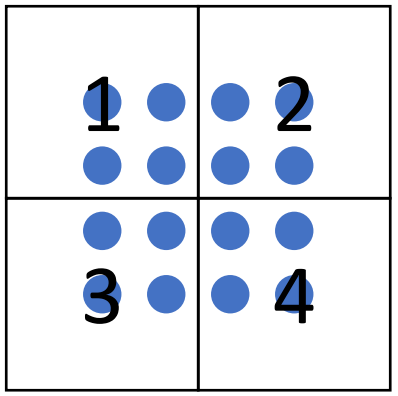
Output: C x 4 x 4

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

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

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

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

→

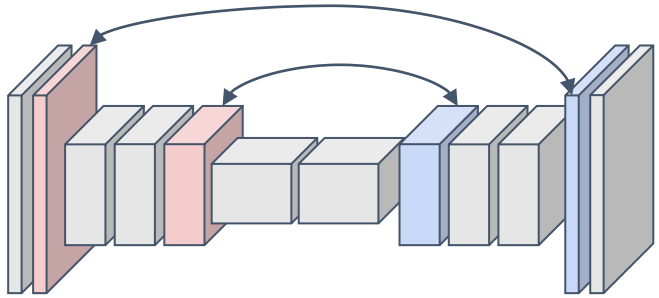
5	6
7	8

→ Rest of net →

1	2
3	4

→

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

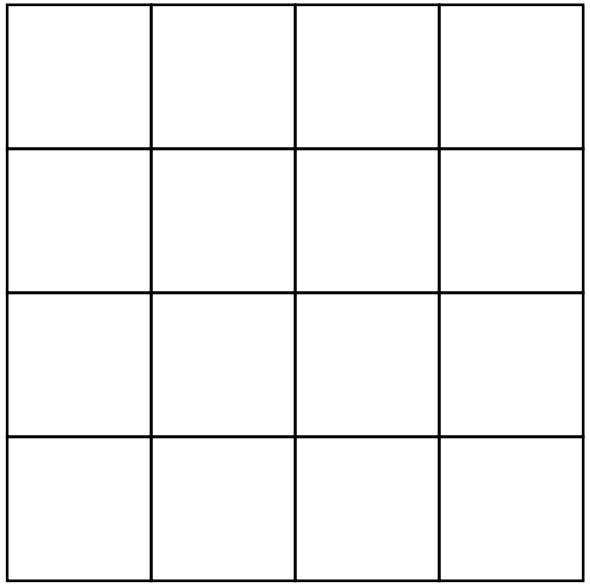


Pair each downsampling layer with an upsampling layer

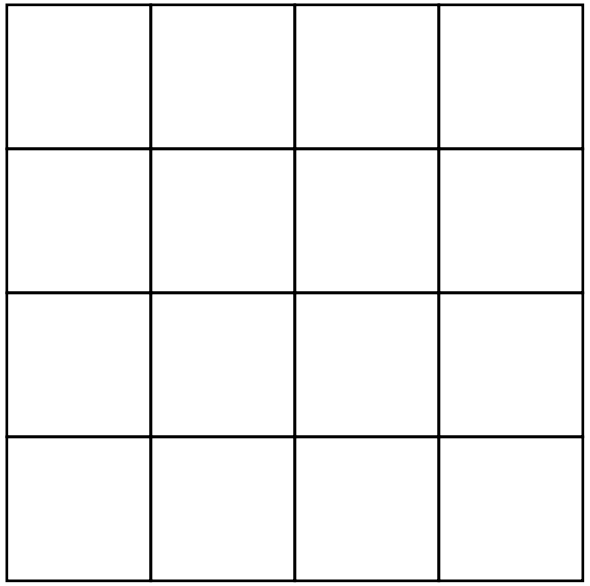
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Learnable Upsampling: Transposed Convolution

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



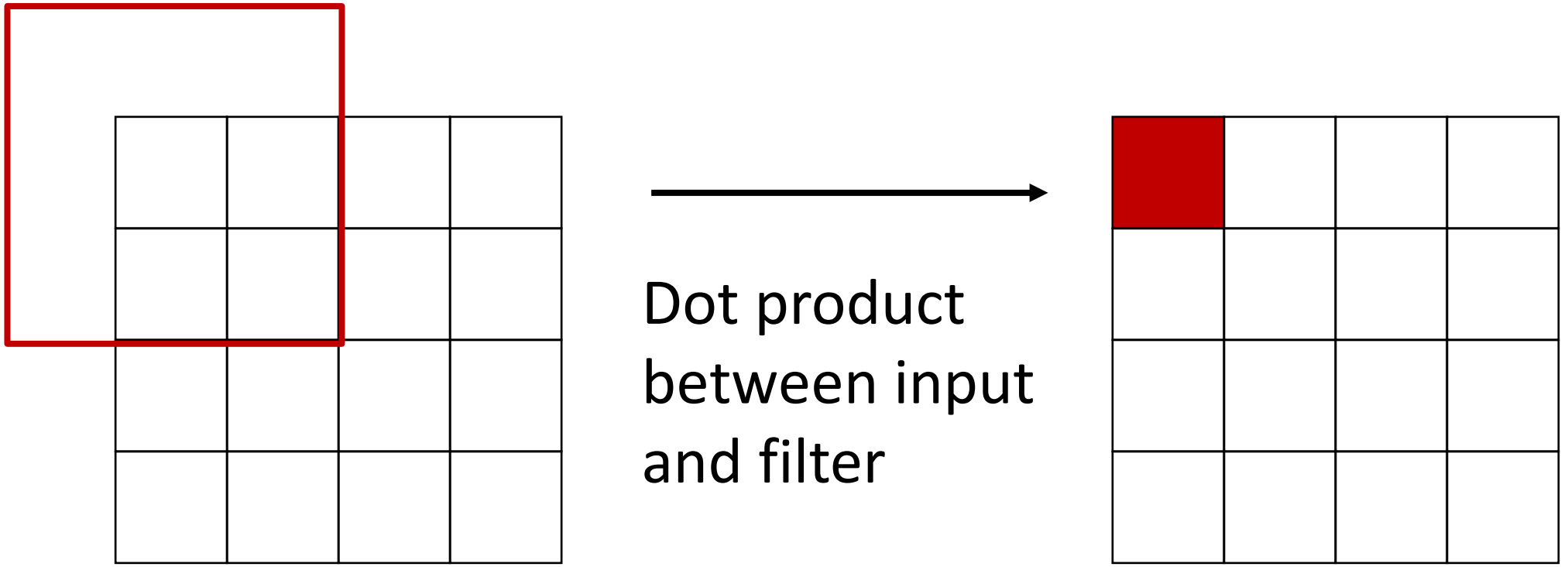
Input: 4 x 4



Output: 4 x 4

Learnable Upsampling: Transposed Convolution

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

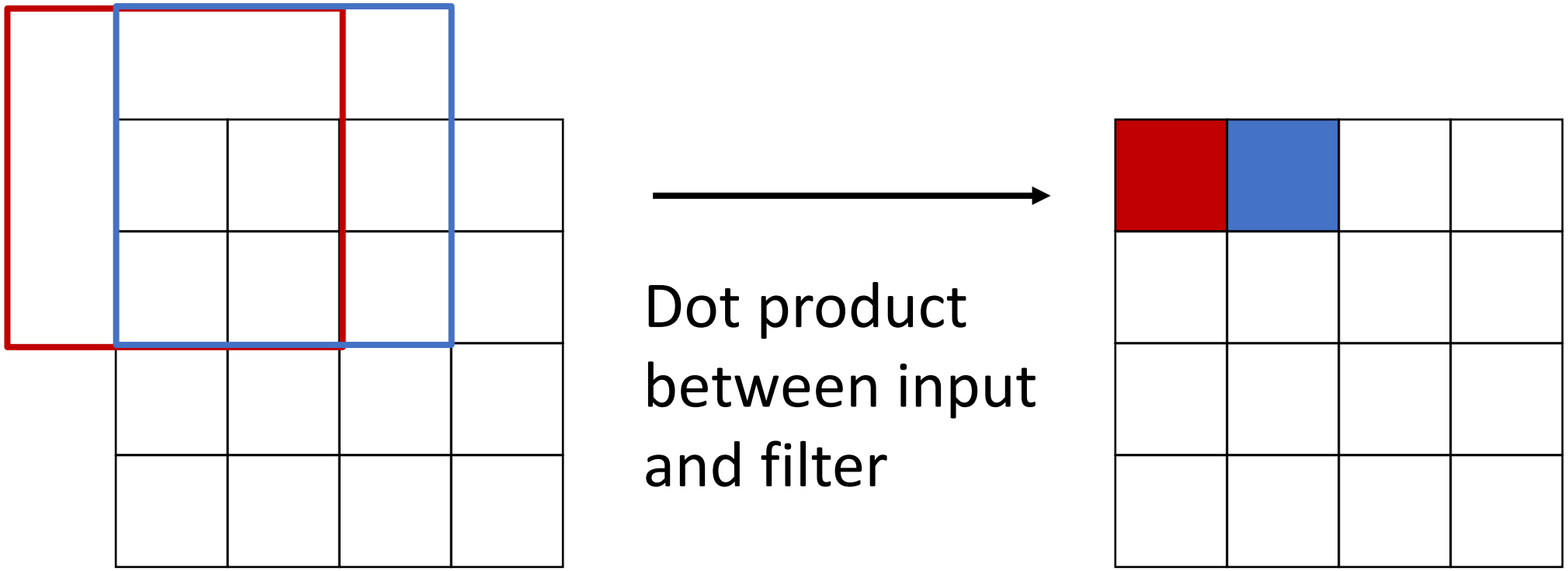


Input: 4 x 4

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

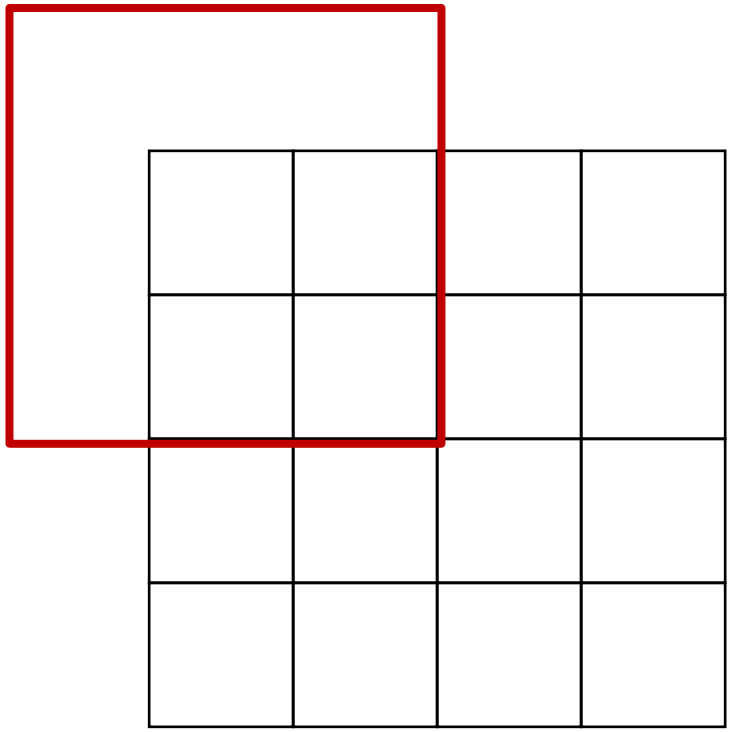
Recall: Normal 3 x 3 convolution, stride 2, pad 1

Input: 4 x 4

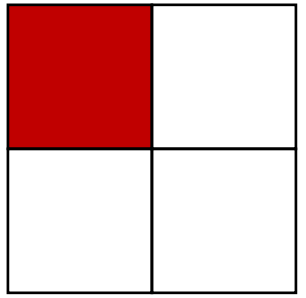
Output: 2 x 2

Learnable Upsampling: Transposed Convolution

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



Dot product between input and filter

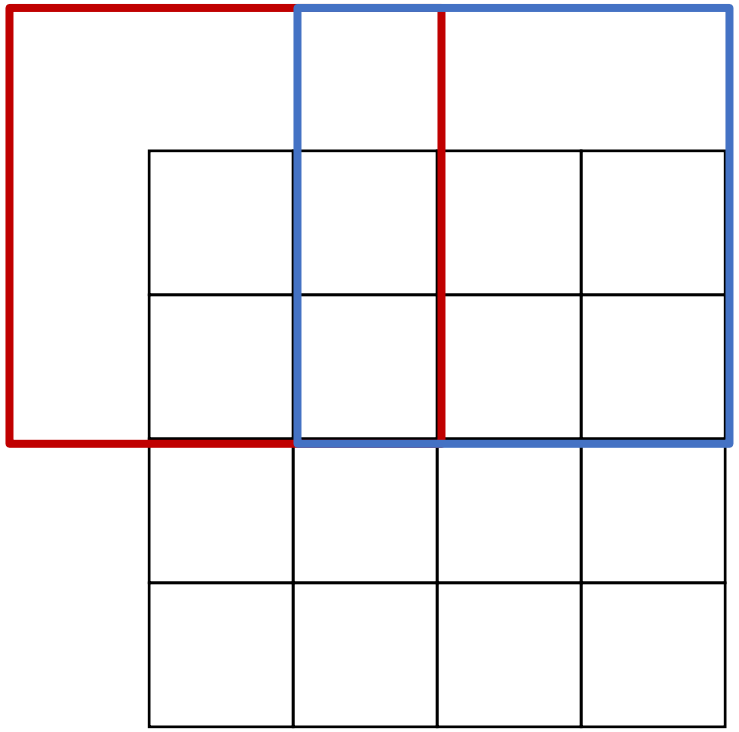


Input: 4 x 4

Output: 2 x 2

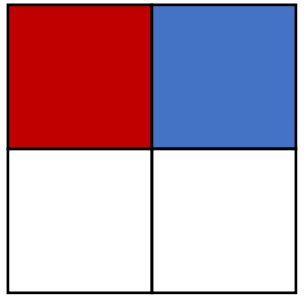
Learnable Upsampling: Transposed Convolution

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



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

Dot product between input and filter

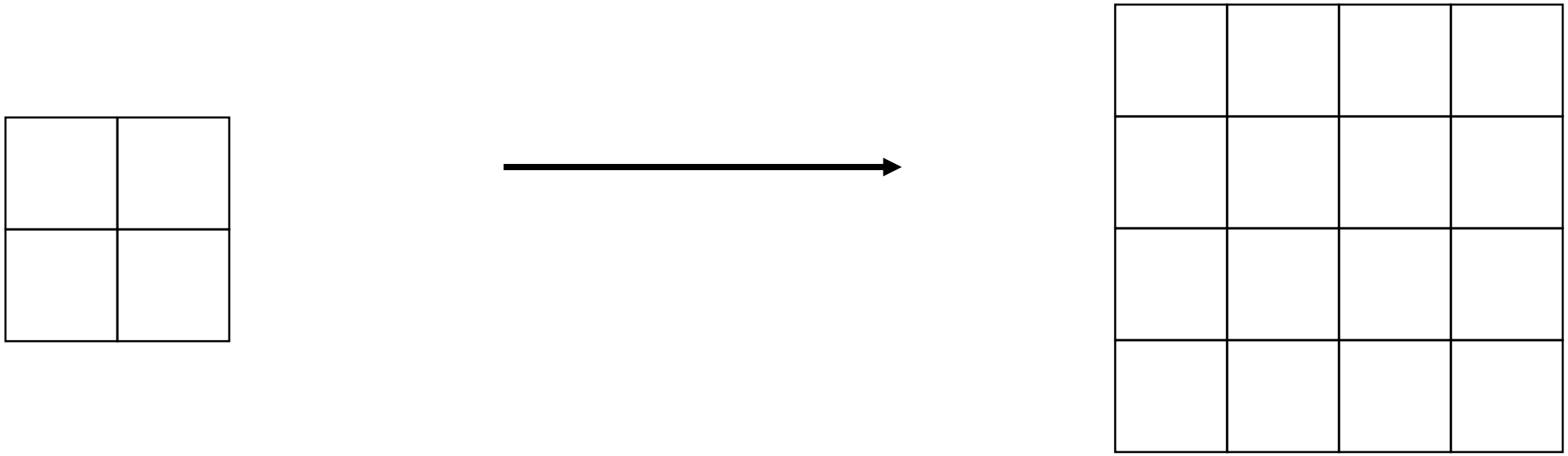


Input: 4 x 4

Output: 2 x 2

Learnable Upsampling: Transposed Convolution

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

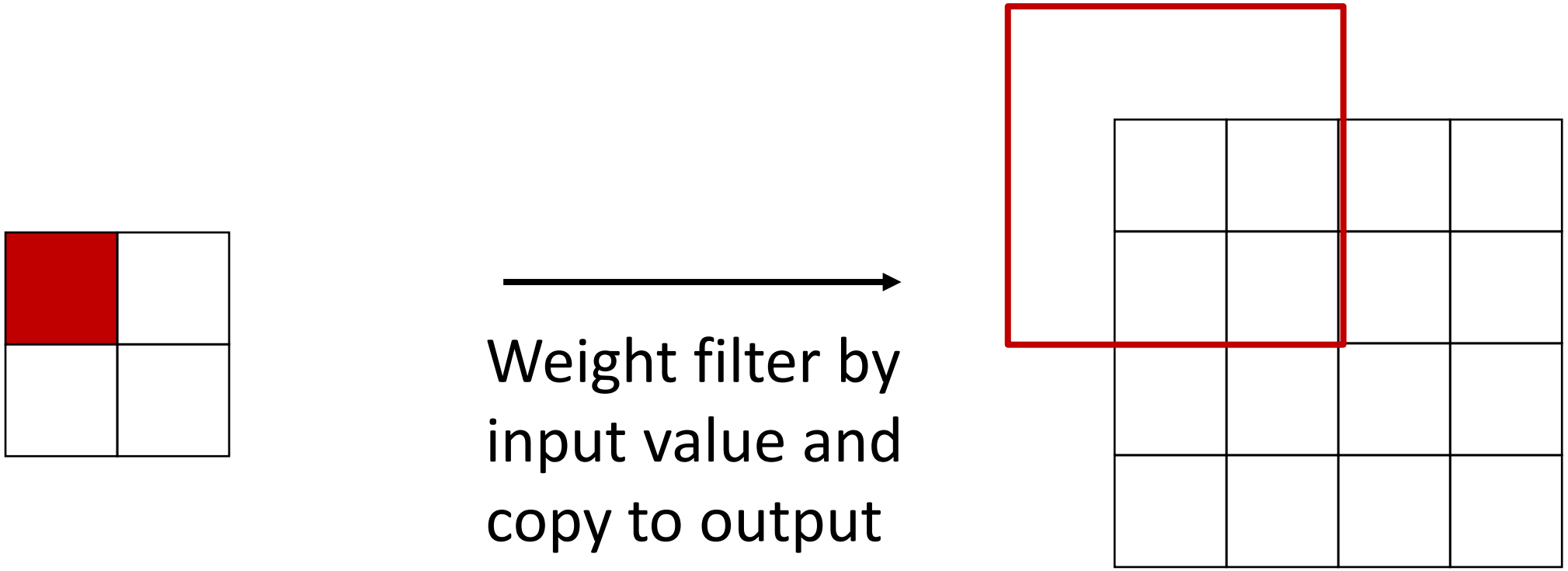


Input: 2 x 2

Output: 4 x 4

Learnable Upsampling: Transposed Convolution

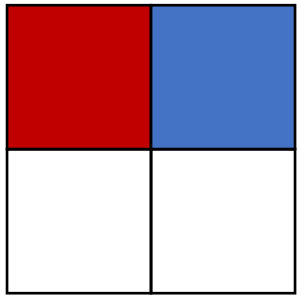
3 x 3 **convolution transpose**, stride 2



Learnable Upsampling: Transposed Convolution

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

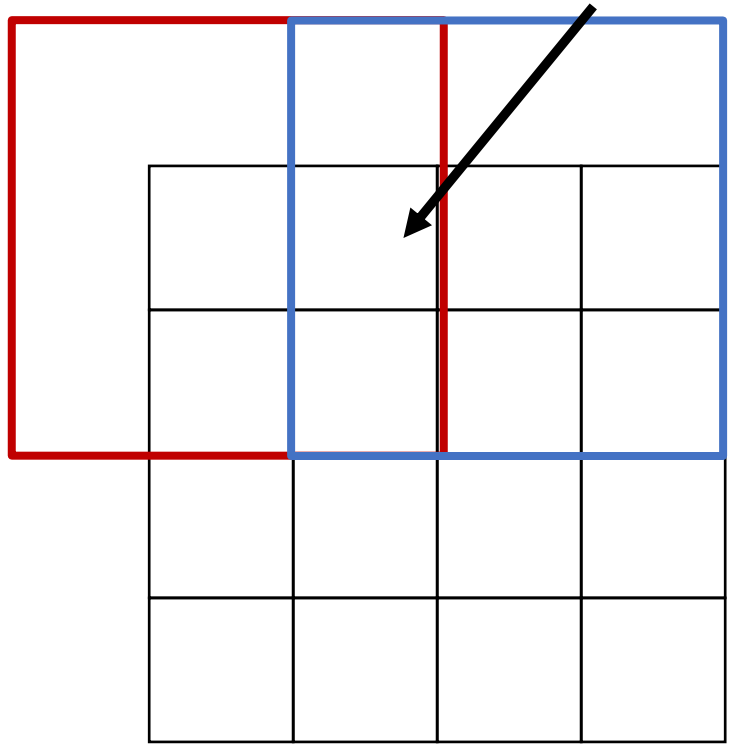
Filter moves 2 pixels in output
for every 1 pixel in input



Input: 2 x 2

Weight filter by
input value and
copy to output

Sum where
output overlaps



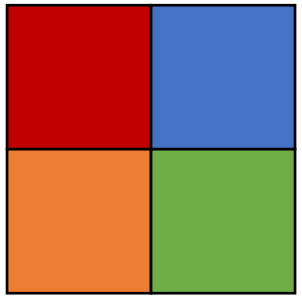
Output: 4 x 4

Learnable Upsampling: Transposed Convolution

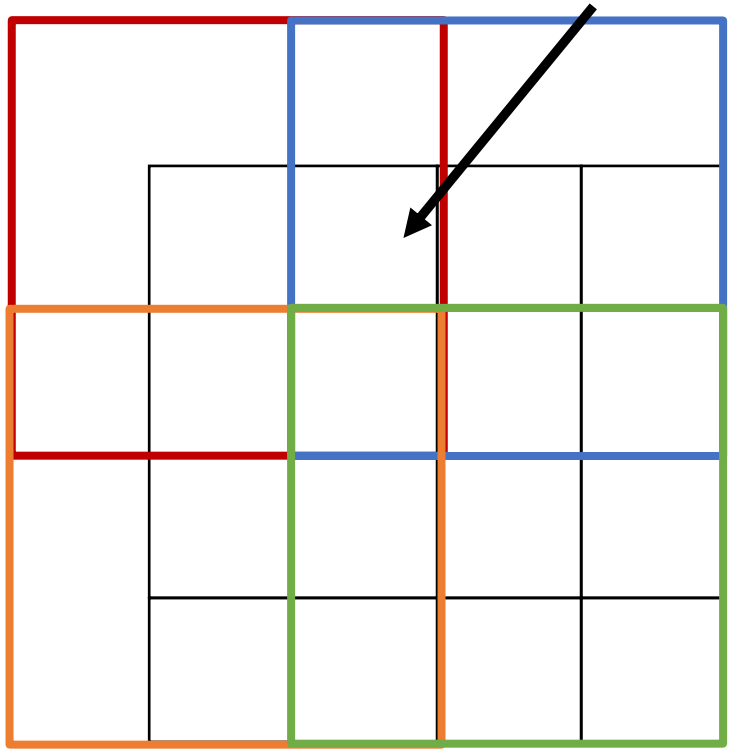
3 x 3 **convolution transpose**, stride 2

Sum where output overlaps

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



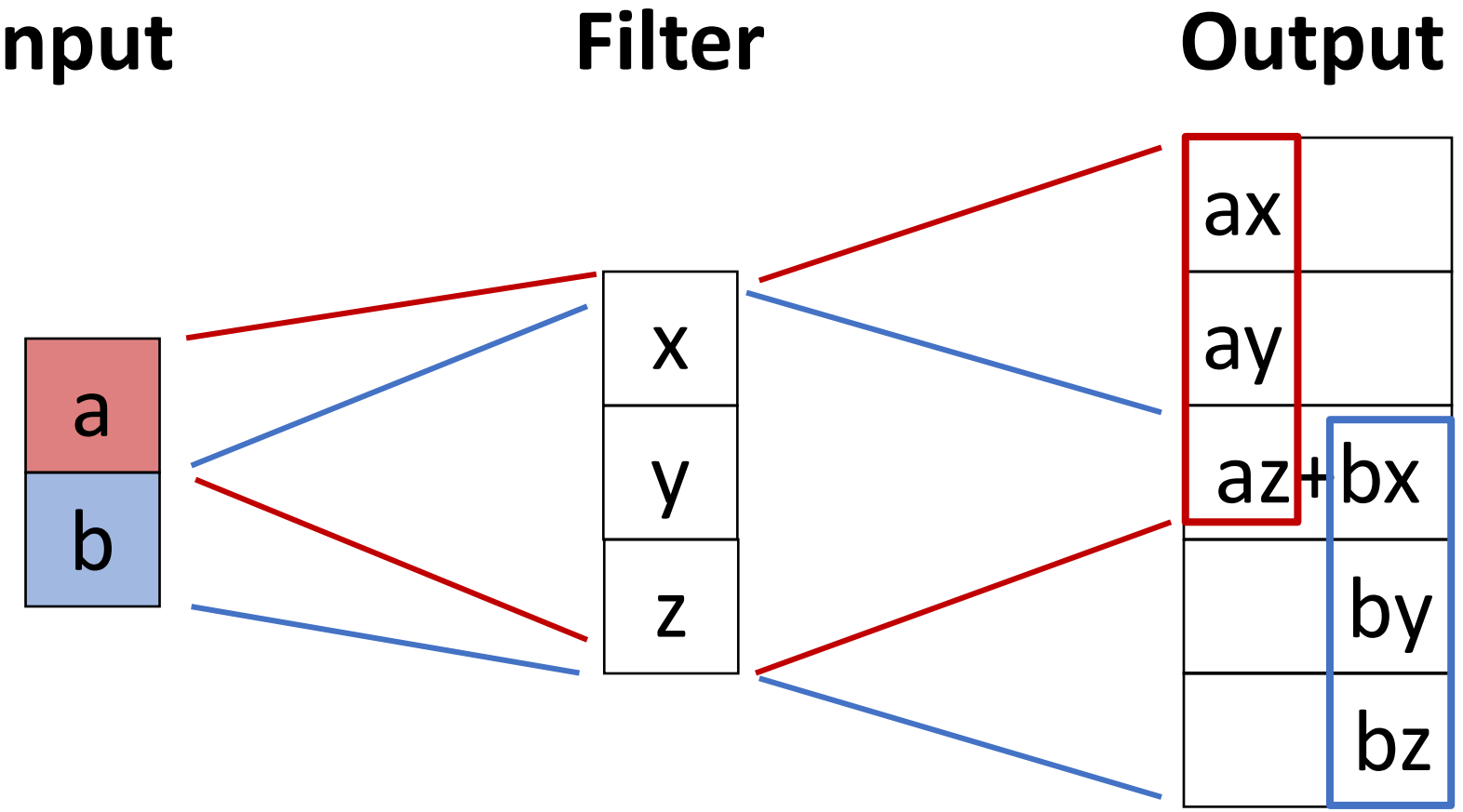
Weight filter by input value and copy to output



Input: 2 x 2

Output: 4 x 4

Transposed Convolution: 1D example

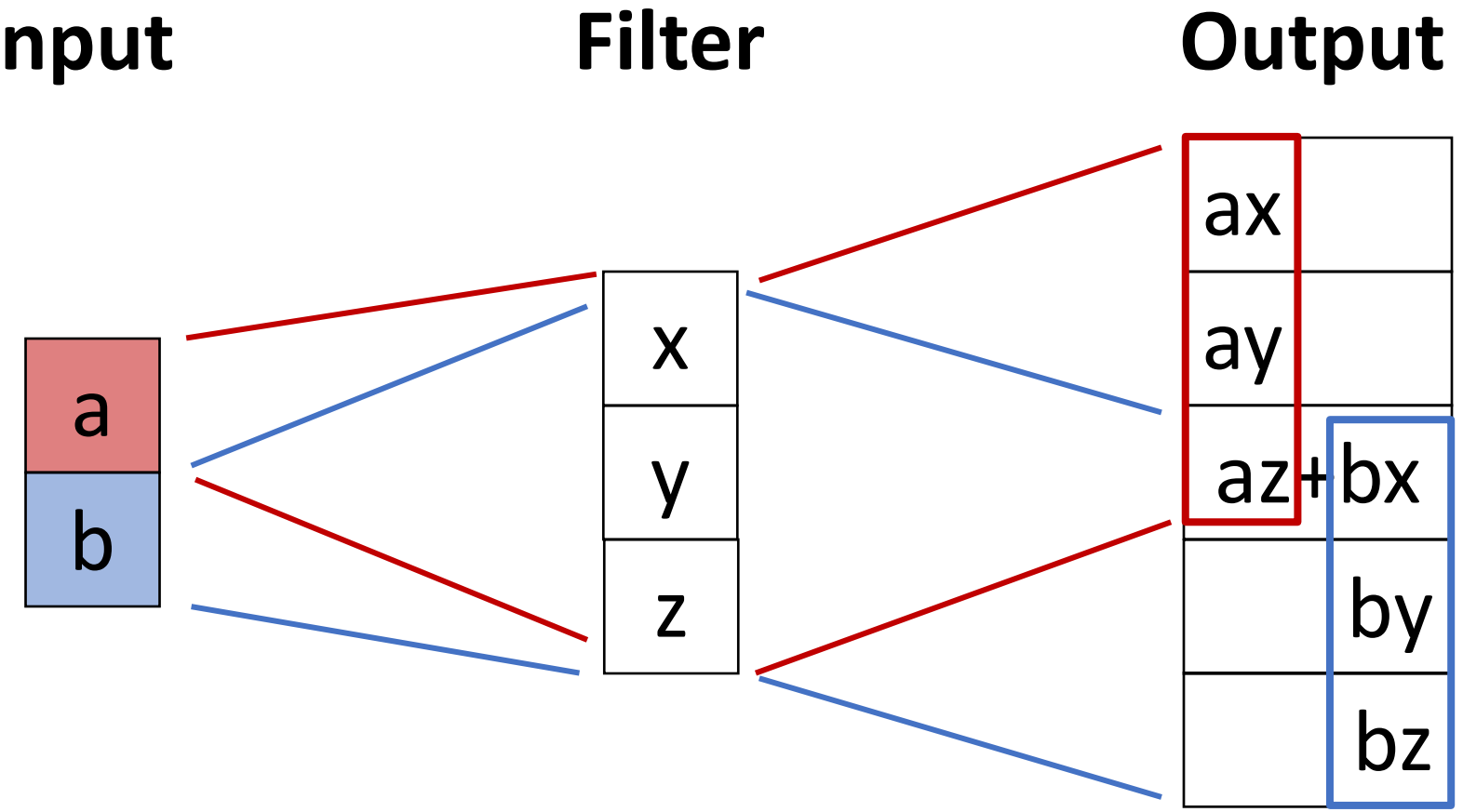


Output has copies of filter weighted by input

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

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

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

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

Transposed convolution multiplies by the transpose of the same matrix:

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

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

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

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

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

Example: 1D conv, kernel size=3, stride=2, padding=1

Transposed convolution multiplies by the transpose of the same matrix:

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

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

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

Recall: PyTorch Convolution Layer

CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

Recall: PyTorch Convolution Layers

CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

CLASS torch.nn.Conv1d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

CLASS torch.nn.Conv3d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

PyTorch Transposed Convolution Layers

CLASS torch.nn.ConvTranspose2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *output_padding=0*, *groups=1*, *bias=True*, *dilation=1*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

CLASS torch.nn.ConvTranspose1d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *output_padding=0*, *groups=1*, *bias=True*, *dilation=1*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

CLASS torch.nn.ConvTranspose3d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *output_padding=0*, *groups=1*, *bias=True*, *dilation=1*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

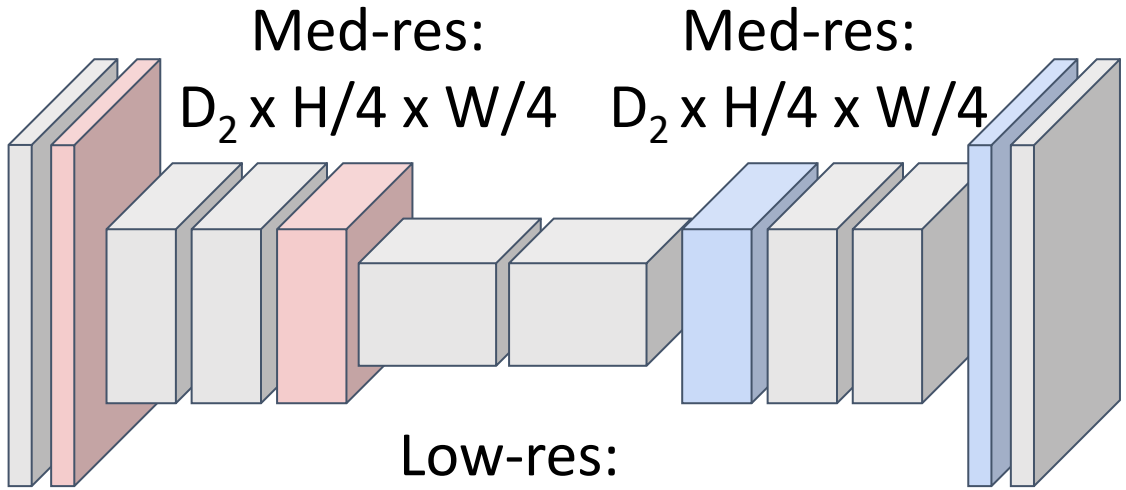
Semantic Segmentation: Fully Convolutional Network

Downsampling: Pooling, strided convolution

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

Upsampling: interpolation, 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

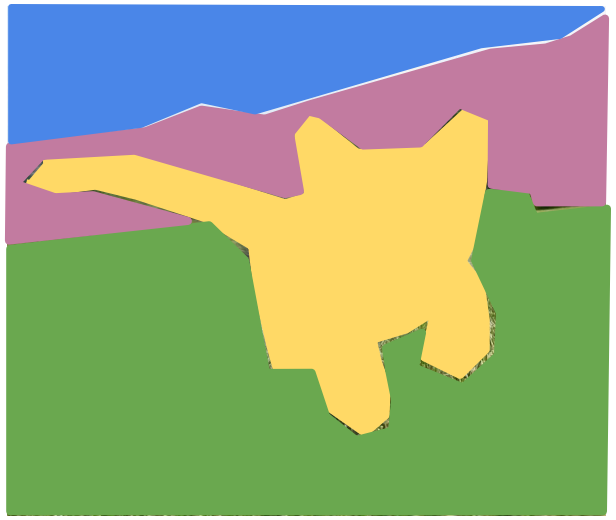
Computer Vision Task: Semantic Segmentation

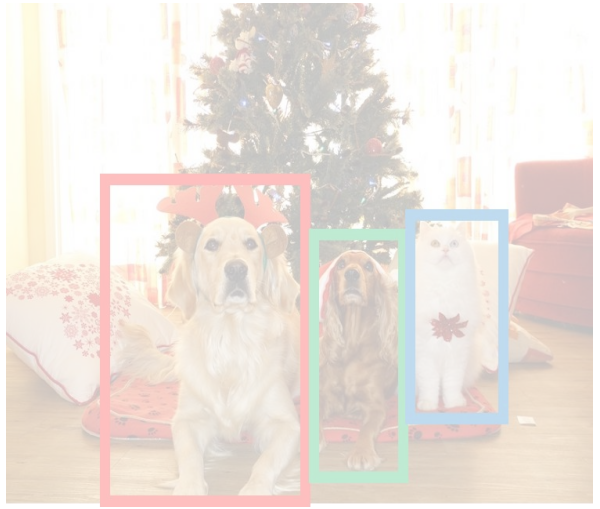
Classification

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

Computer Vision Task: Object Detection

Classification

CAT

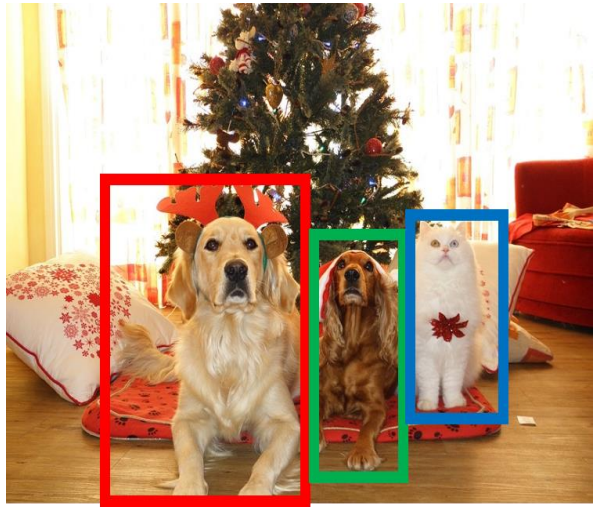
No spatial extent

Semantic Segmentation

GRASS, CAT, TREE, SKY

No objects, just pixels

Object Detection



DOG, DOG, CAT

Instance Segmentation

DOG, DOG, CAT

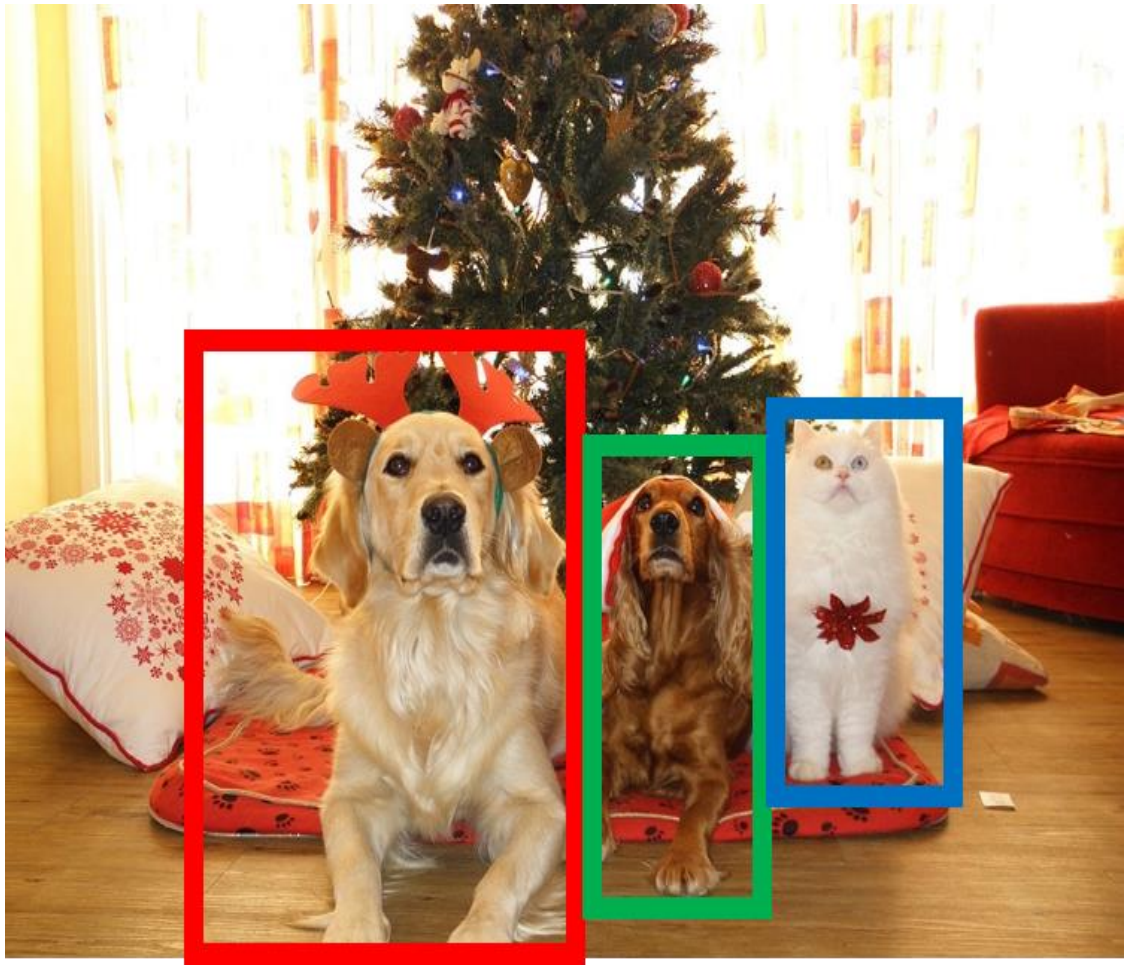
Multiple Objects

Object Detection: Task Definition

Input: Single RGB Image

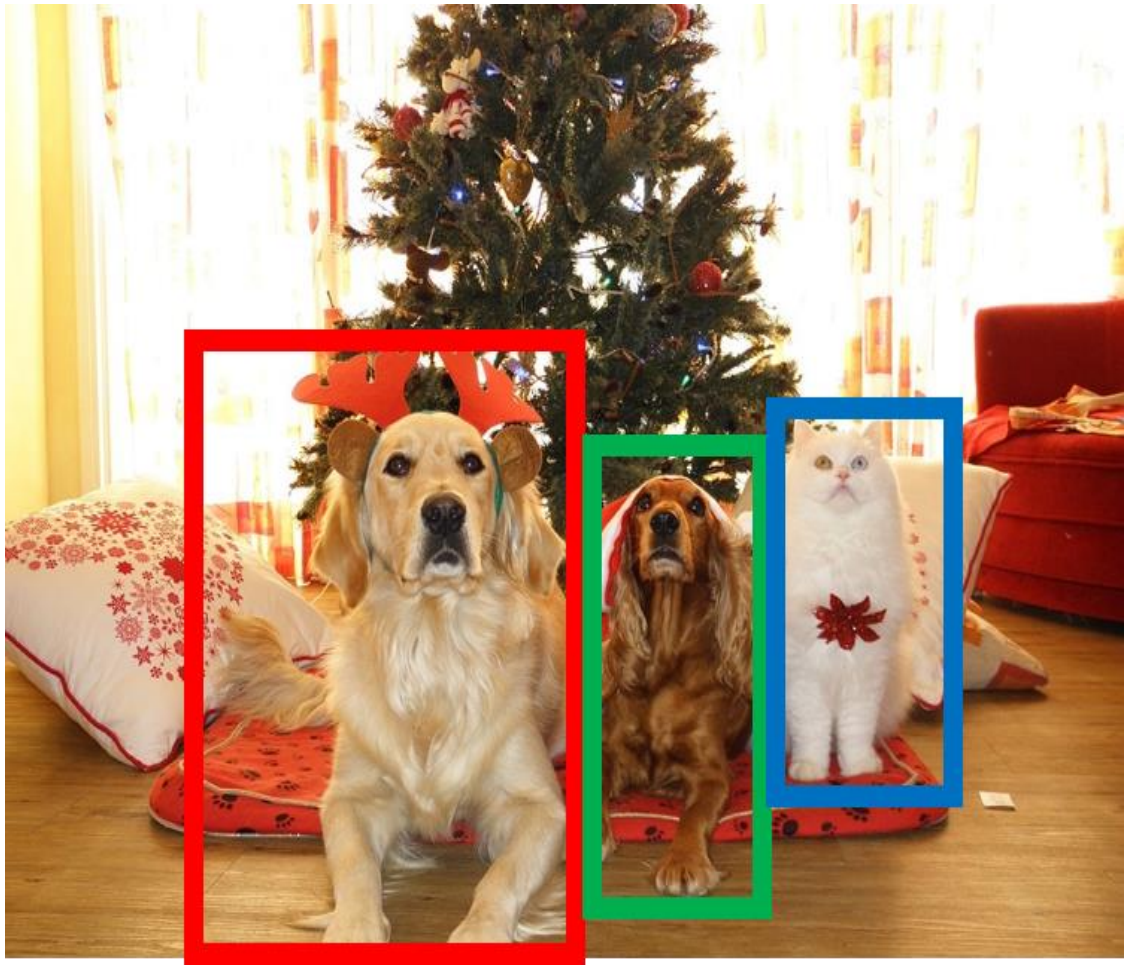
Output: A set of detected objects;
For each object, predict:

1. Category label (from fixed, known set of categories)
2. Bounding box (four numbers: x, y, width, height)



Object Detection: Challenges

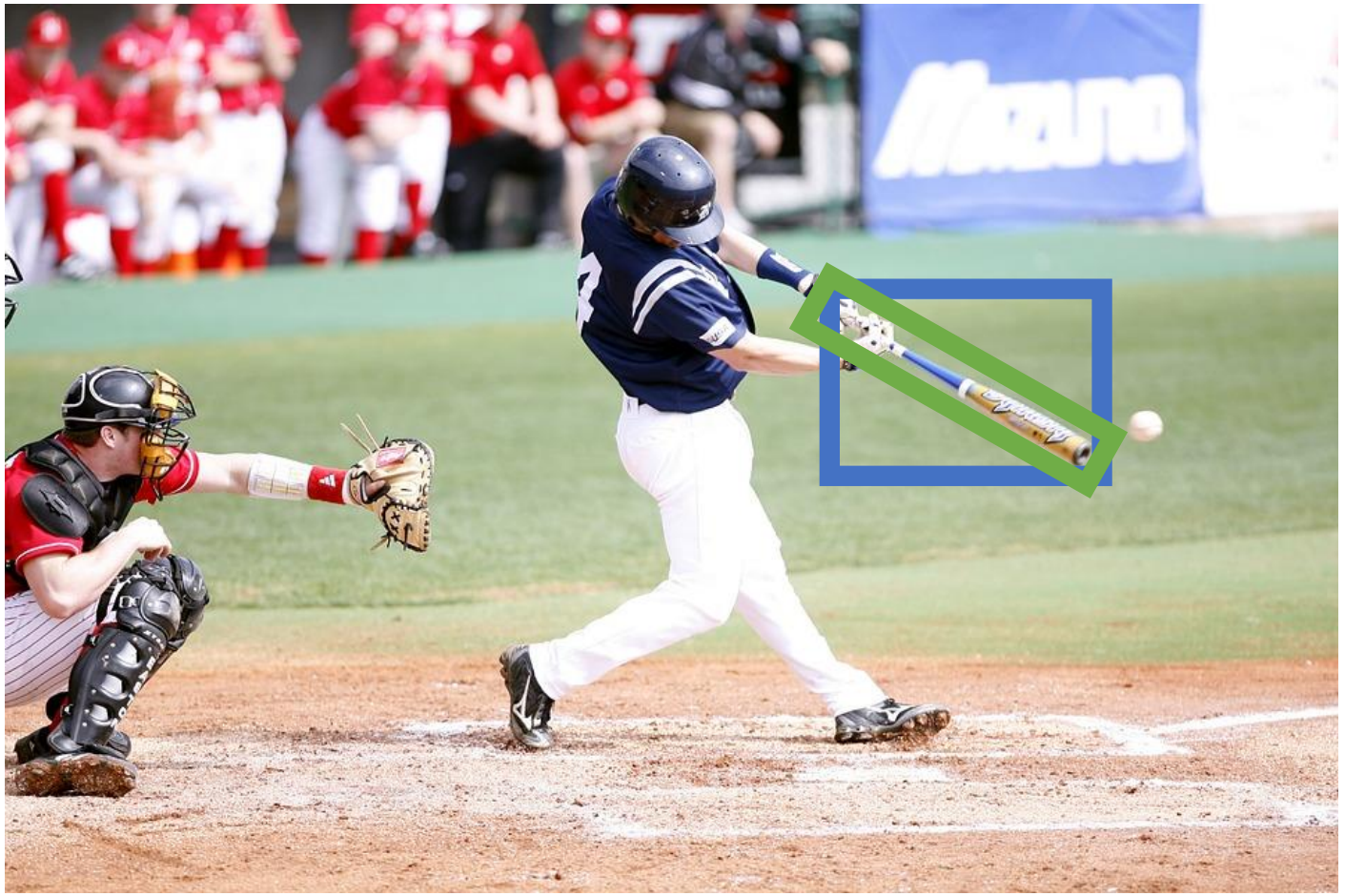
- **Multiple outputs**: Need to output variable numbers of objects per image
- **Multiple types of output**: Need to predict “what” (category label) as well as “where” (bounding box)
- **Large images**: Classification works at 224x224; need higher resolution for detection, often ~800x600



Bounding Boxes

Bounding boxes are typically *axis-aligned*

Oriented boxes are much less common

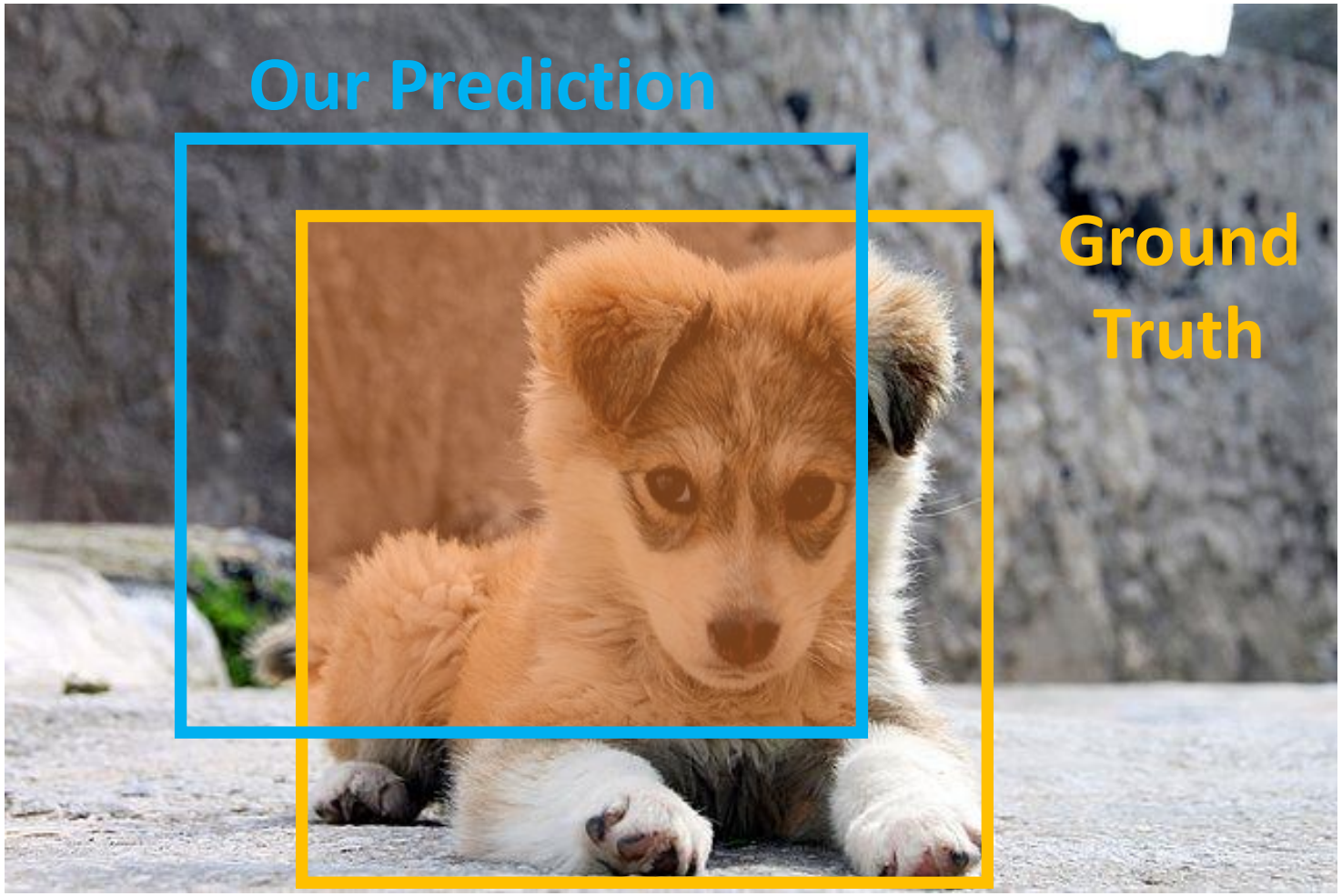


Comparing Boxes: Intersection over Union (IoU)

How can we compare our prediction to the ground-truth box?

Intersection over Union (IoU)
(Also called “Jaccard similarity” or “Jaccard index”):

$$\frac{Area\ of\ Intersection}{Area\ of\ Union}$$

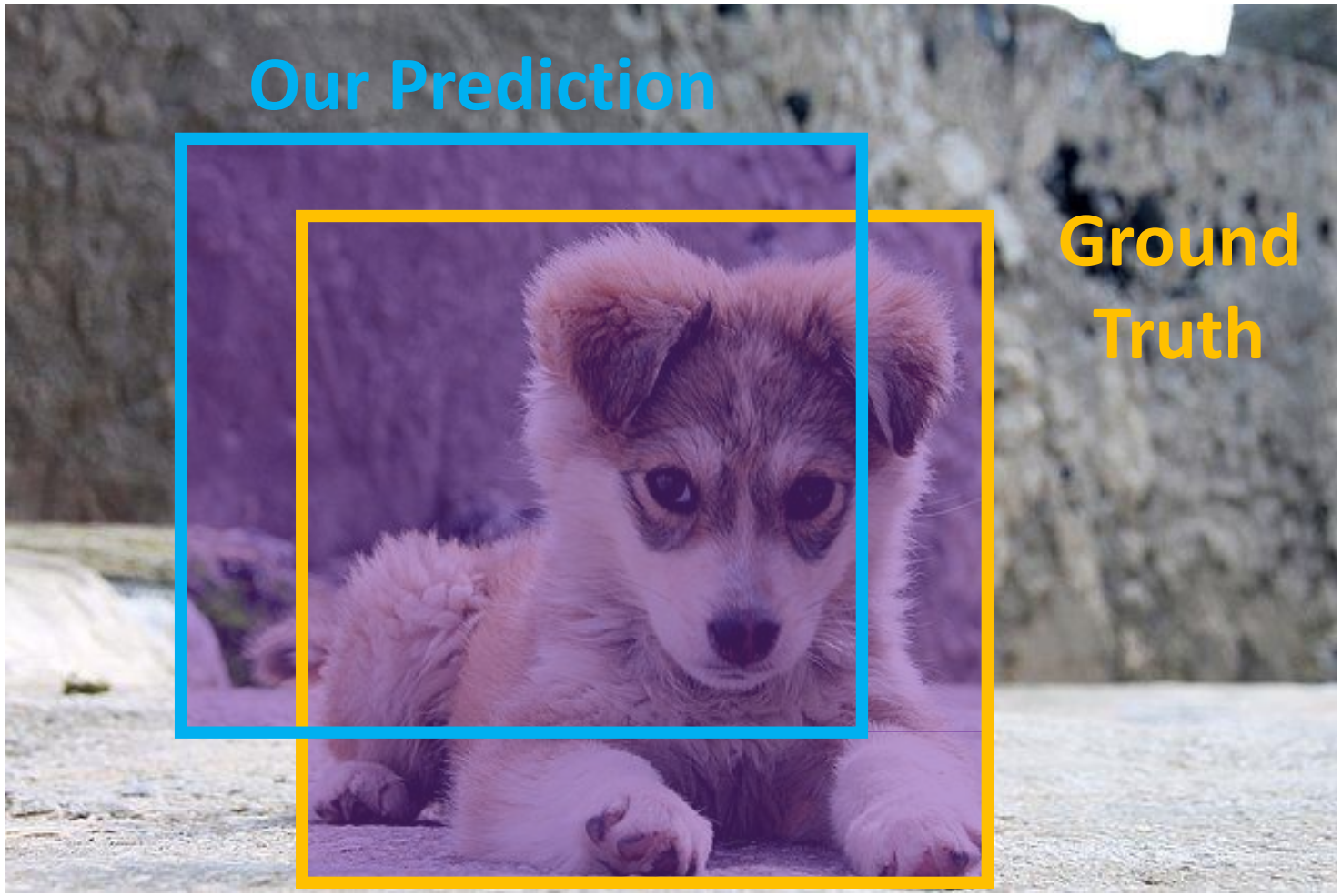


Comparing Boxes: Intersection over Union (IoU)

How can we compare our prediction to the ground-truth box?

Intersection over Union (IoU)
(Also called “Jaccard similarity” or “Jaccard index”):

$$\frac{\textit{Area of Intersection}}{\textit{Area of Union}}$$



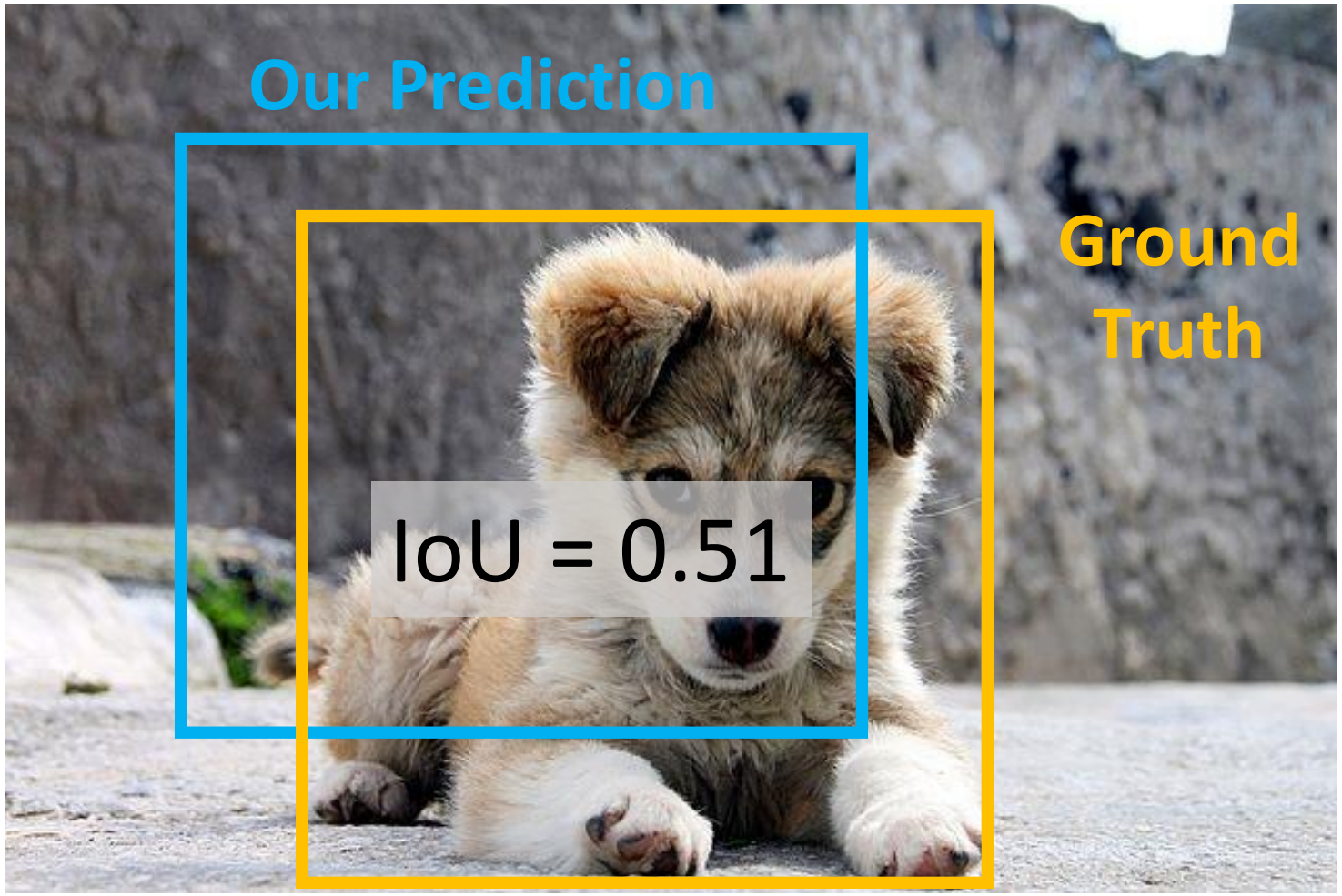
Comparing Boxes: Intersection over Union (IoU)

How can we compare our prediction to the ground-truth box?

Intersection over Union (IoU)
(Also called “Jaccard similarity” or “Jaccard index”):

$$\frac{Area\ of\ Intersection}{Area\ of\ Union}$$

IoU > 0.5 is “decent”



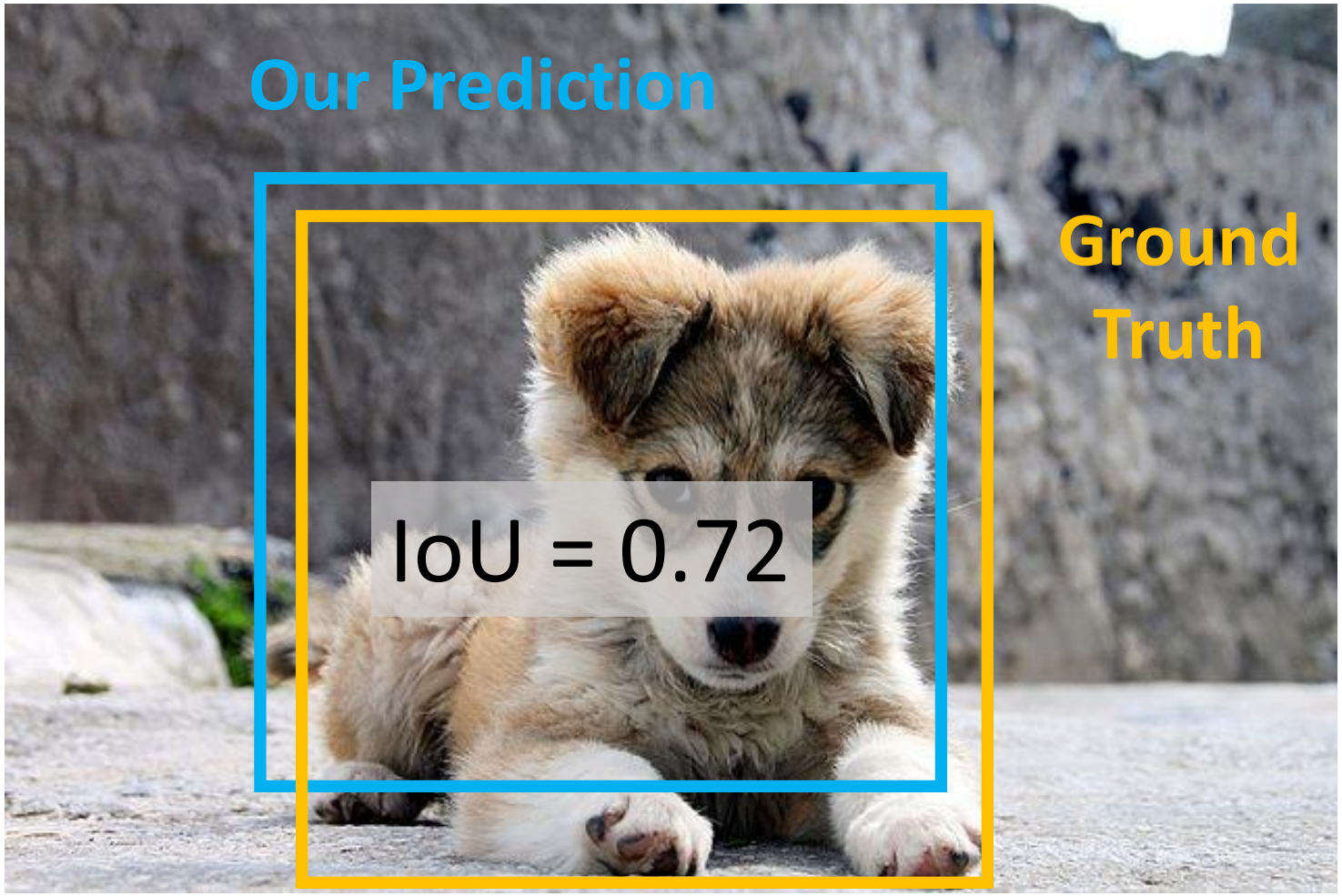
Comparing Boxes: Intersection over Union (IoU)

How can we compare our prediction to the ground-truth box?

Intersection over Union (IoU)
(Also called “Jaccard similarity” or “Jaccard index”):

$$\frac{\textit{Area of Intersection}}{\textit{Area of Union}}$$

IoU > 0.5 is “decent”,
IoU > 0.7 is “pretty good”,



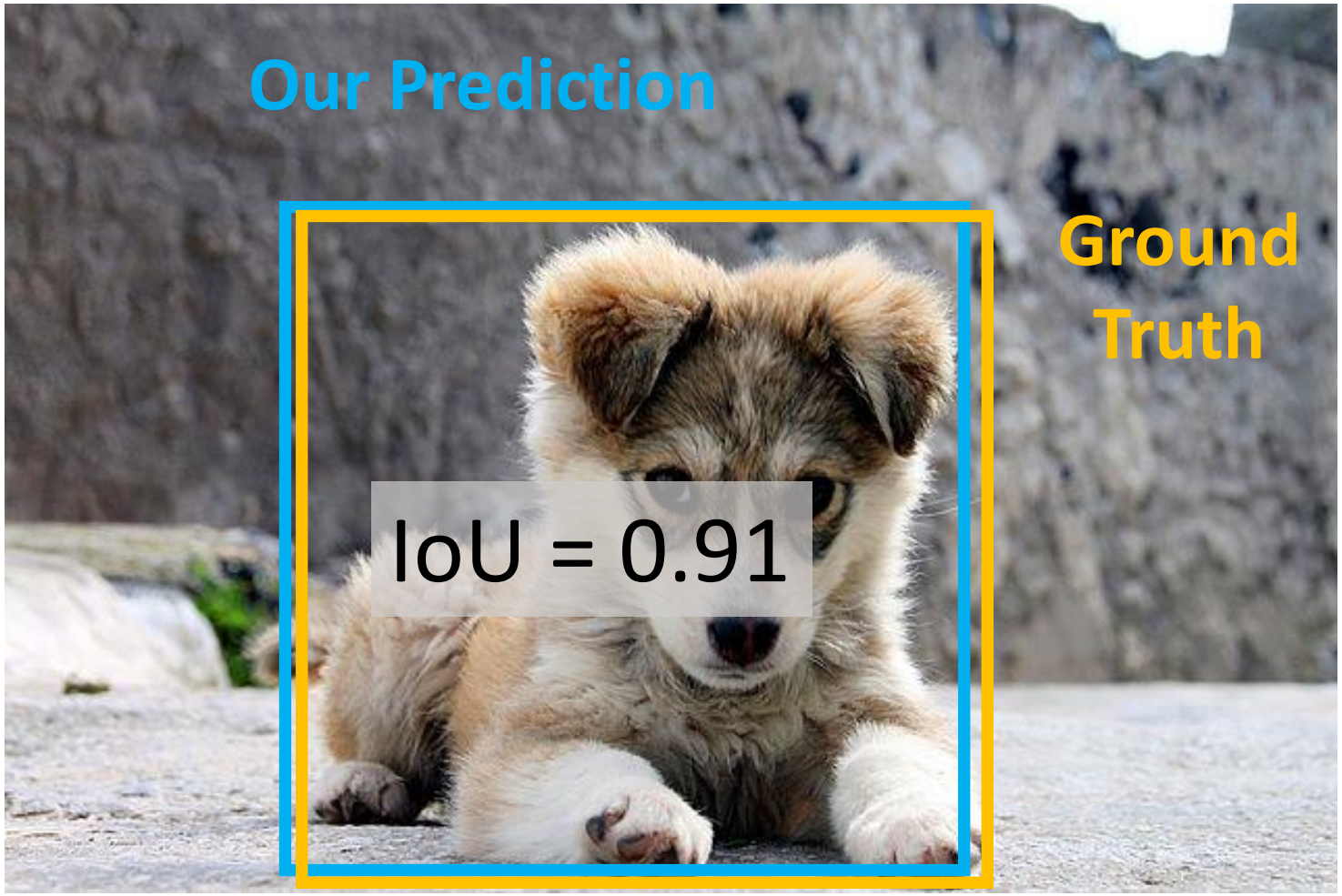
Comparing Boxes: Intersection over Union (IoU)

How can we compare our prediction to the ground-truth box?

Intersection over Union (IoU)
(Also called “Jaccard similarity” or “Jaccard index”):

$$\frac{\textit{Area of Intersection}}{\textit{Area of Union}}$$

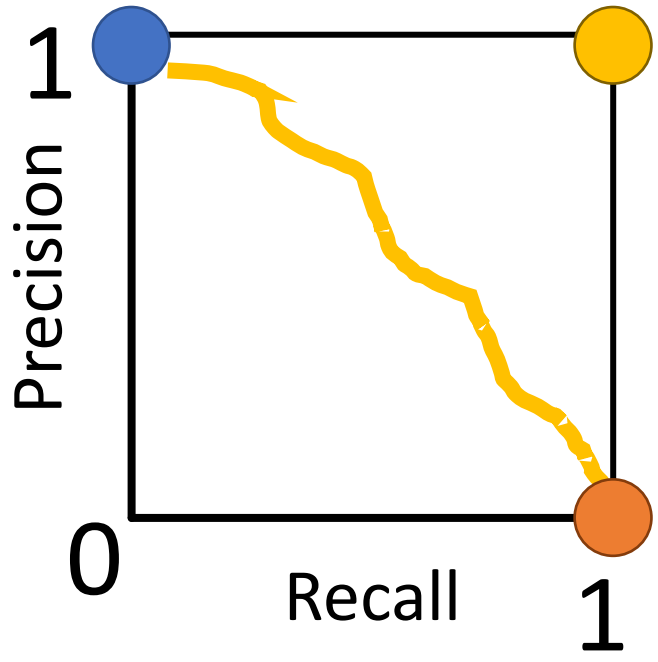
IoU > 0.5 is “decent”,
IoU > 0.7 is “pretty good”,
IoU > 0.9 is “almost perfect”



Evaluating Detectors

- True detection: high IoU
- Precision: # true detections / # all detections
- Recall: # true detections / # ground truths

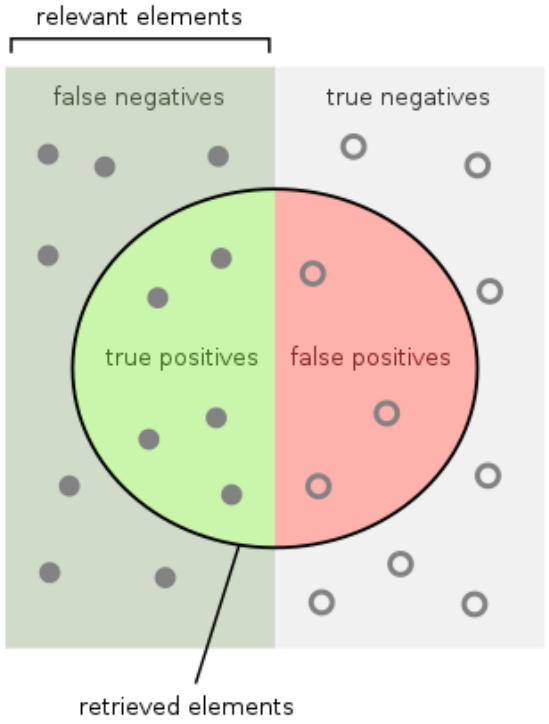
Reject everything: no mistakes



Ideal!

Summarize by area under curve (avg. precision)

Accept everything: Miss nothing



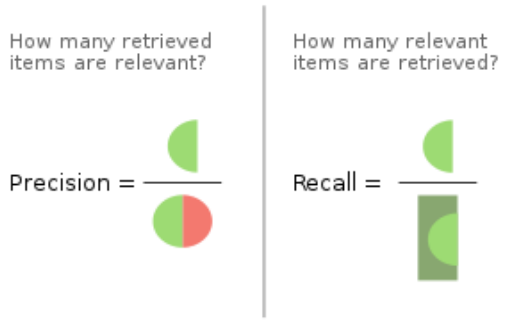
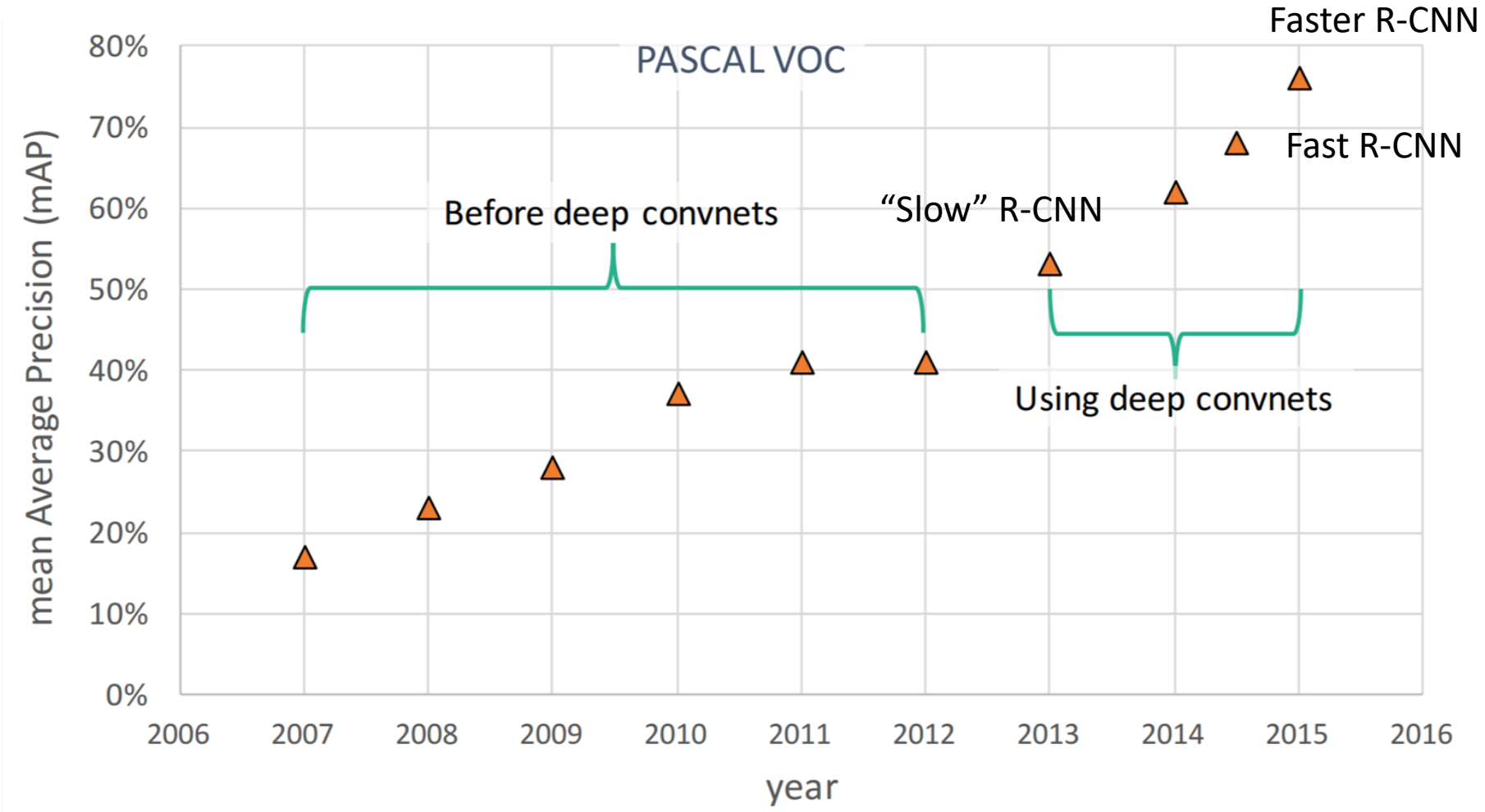


Image source: wikipedia

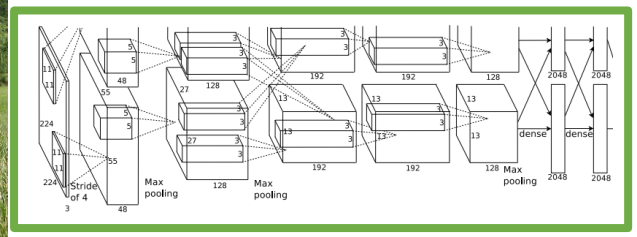
Object Detection Progress



Detecting a single object

Often pretrained
on ImageNet
(Transfer learning)

This image is CC0 public domain



Vector:
4096

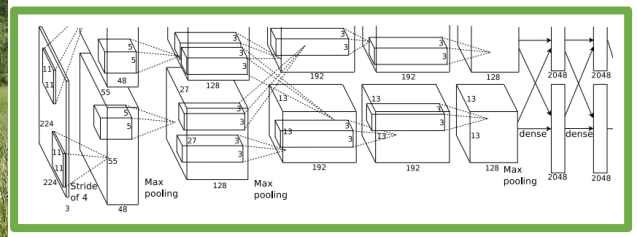
Treat localization as a regression problem!

Detecting a single object "What"

Correct label:
Cat

Often pretrained on ImageNet (Transfer learning)

This image is CC0 public domain



Fully Connected: 4096 to 1000

Vector: 4096

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

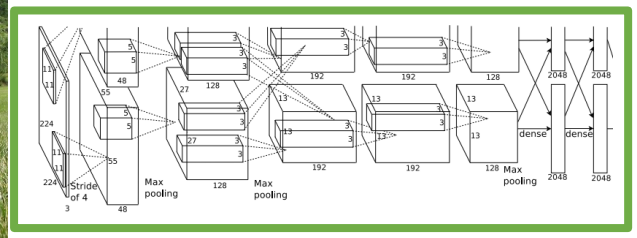
Softmax Loss

Treat localization as a regression problem!

Detecting a single object "What"

This image is CC0 public domain

Often pretrained on ImageNet (Transfer learning)



Vector: 4096

Fully Connected: 4096 to 1000

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

Softmax Loss

Correct label:
Cat

Treat localization as a regression problem!

Fully Connected: 4096 to 4

Box Coordinates
(x, y, w, h)

L2 Loss

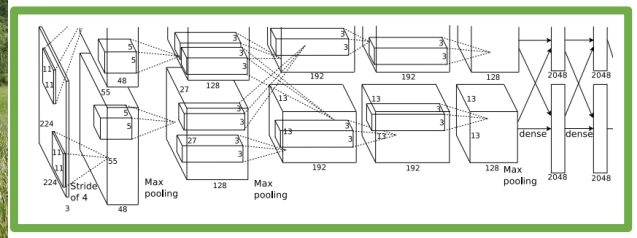
Correct box:
(x', y', w', h')

"Where"

Detecting a single object "What"

This image is CC0 public domain

Often pretrained on ImageNet (Transfer learning)



Vector: 4096

Fully Connected: 4096 to 1000

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

Correct label:
Cat

Softmax Loss

Multitask Loss

Weighted Sum → **Loss**

$$L = L_{cls} + \lambda L_{reg}$$

Fully Connected: 4096 to 4

Box Coordinates
(x, y, w, h)

L2 Loss

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

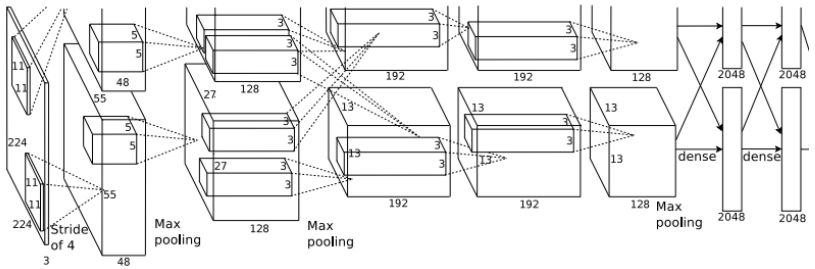
Treat localization as a regression problem!

"Where"

Problem: Images can have more than one object!

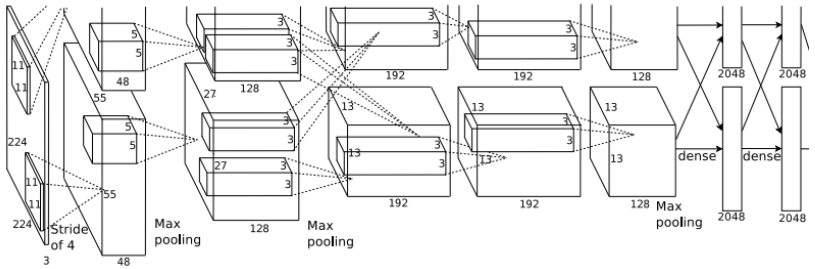
Detecting Multiple Objects

Need different numbers of outputs per image



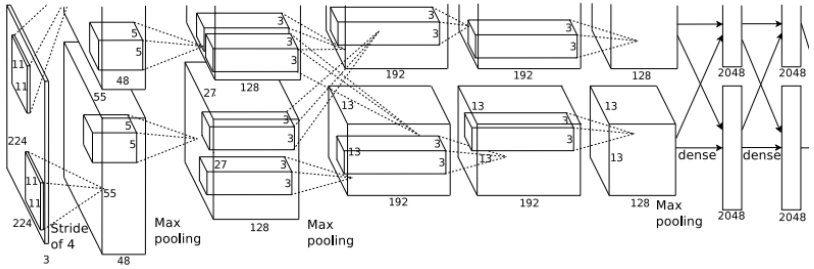
CAT: (x, y, w, h)

4 numbers



DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)

12 numbers

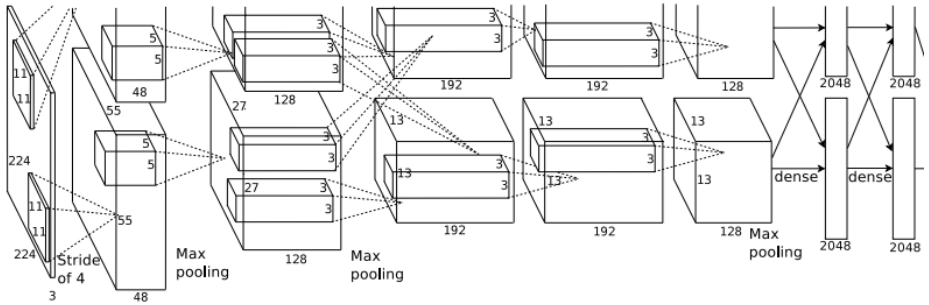


DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
....

Many numbers!

Detecting Multiple Objects: **Sliding Window**

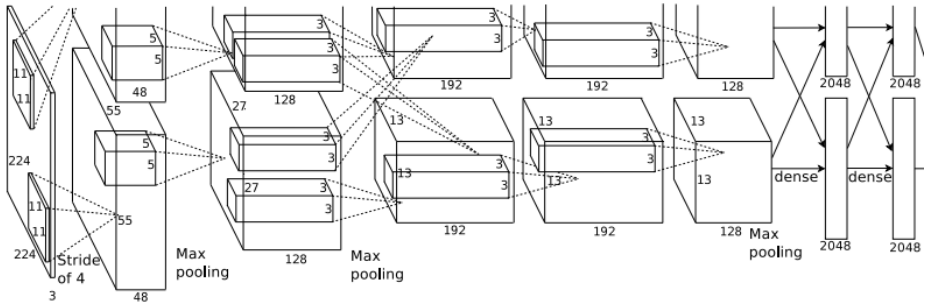
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? NO
Cat? NO
Background? YES

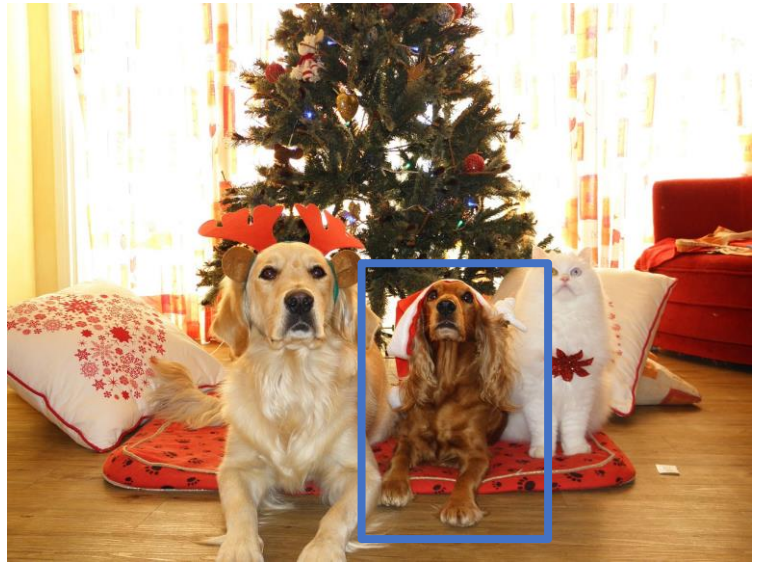
Detecting Multiple Objects: **Sliding Window**

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

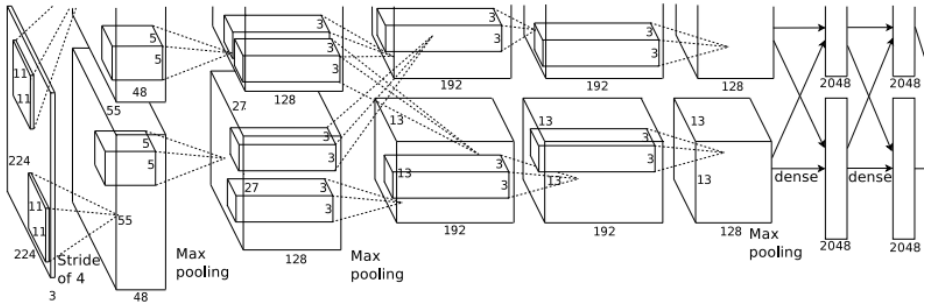


Dog? YES
Cat? NO
Background? NO

Detecting Multiple Objects: **Sliding Window**



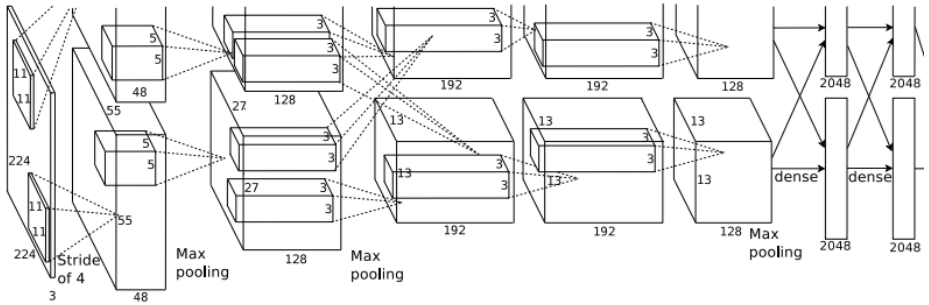
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? YES
Cat? NO
Background? NO

Detecting Multiple Objects: **Sliding Window**

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? NO
Cat? YES
Background? NO

Detecting Multiple Objects: **Sliding Window**

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

800 x 600 image has ~58M boxes! No way we can evaluate them all

Question: How many possible boxes are there in an image of size H x W?

Consider a box of size h x w:
Possible x positions: W – w + 1
Possible y positions: H – h + 1
Possible positions:
(W – w + 1) * (H – h + 1)

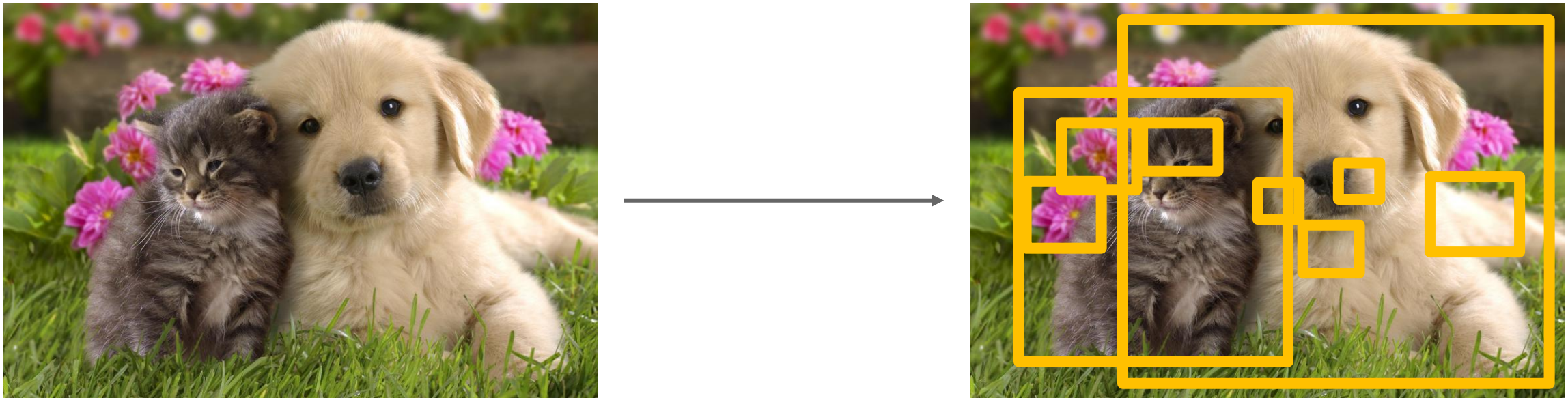
Total possible boxes:

$$\sum_{h=1}^{H}\sum_{w=1}^{W}(W-w+1)(H-h+1)$$

$$=\frac{H(H+1)}{2}\frac{W(W+1)}{2}$$

Region Proposals

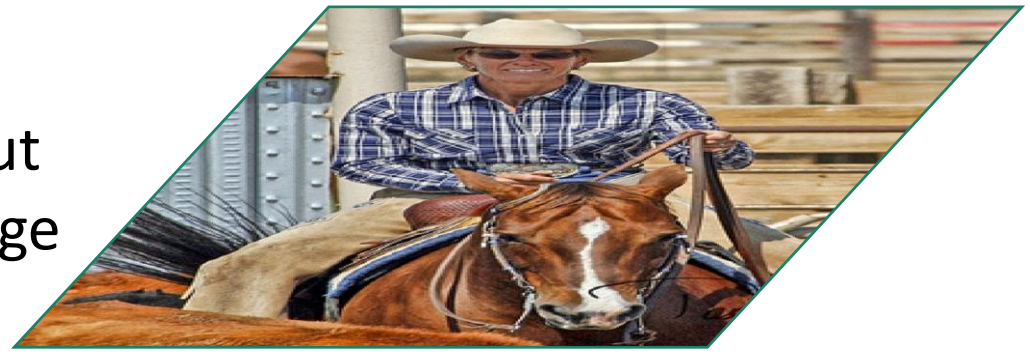
- Find a small set of boxes that are likely to cover all objects
- Often based on heuristics: e.g., look for “blob-like” image regions
- Relatively fast to run; e.g., Selective Search gives 2000 region proposals in a few seconds on CPU



Alexe et al, “Measuring the objectness of image windows”, TPAMI 2012
Uijlings et al, “Selective Search for Object Recognition”, IJCV 2013
Cheng et al, “BING: Binarized normed gradients for objectness estimation at 300fps”, CVPR 2014
Zitnick and Dollar, “Edge boxes: Locating object proposals from edges”, ECCV 2014

R-CNN: Region-Based CNN

Input image



Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

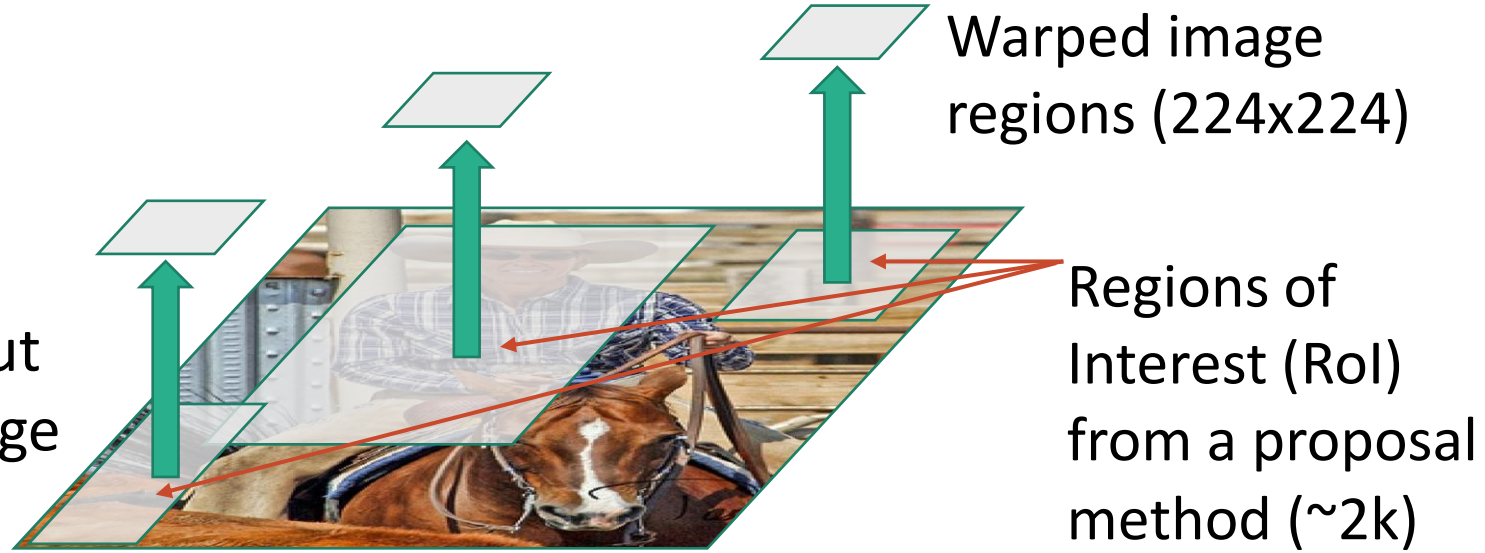
R-CNN: Region-Based CNN

Input image

Regions of Interest (RoI) from a proposal method (~2k)

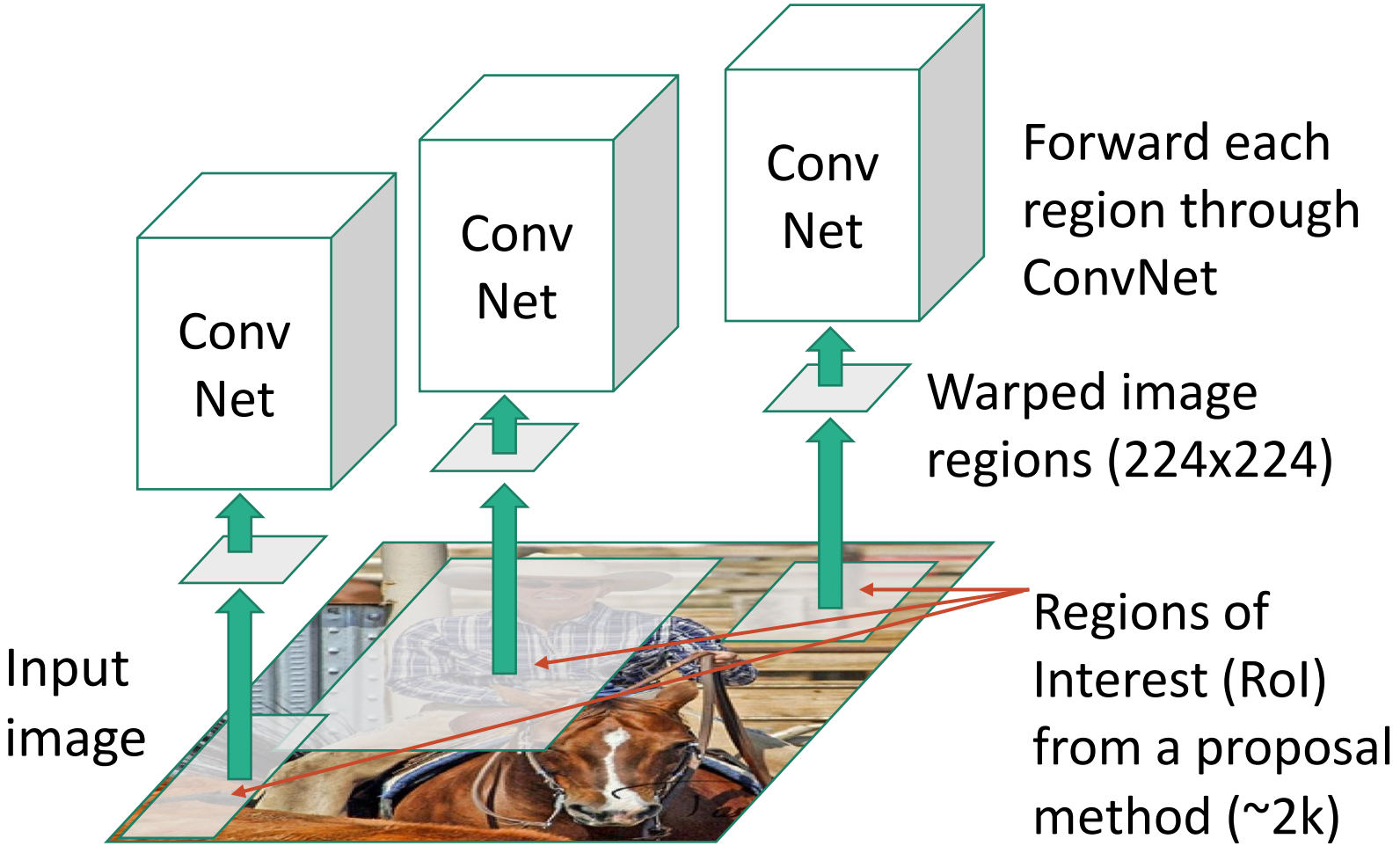
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

R-CNN: Region-Based CNN



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

R-CNN: Region-Based CNN

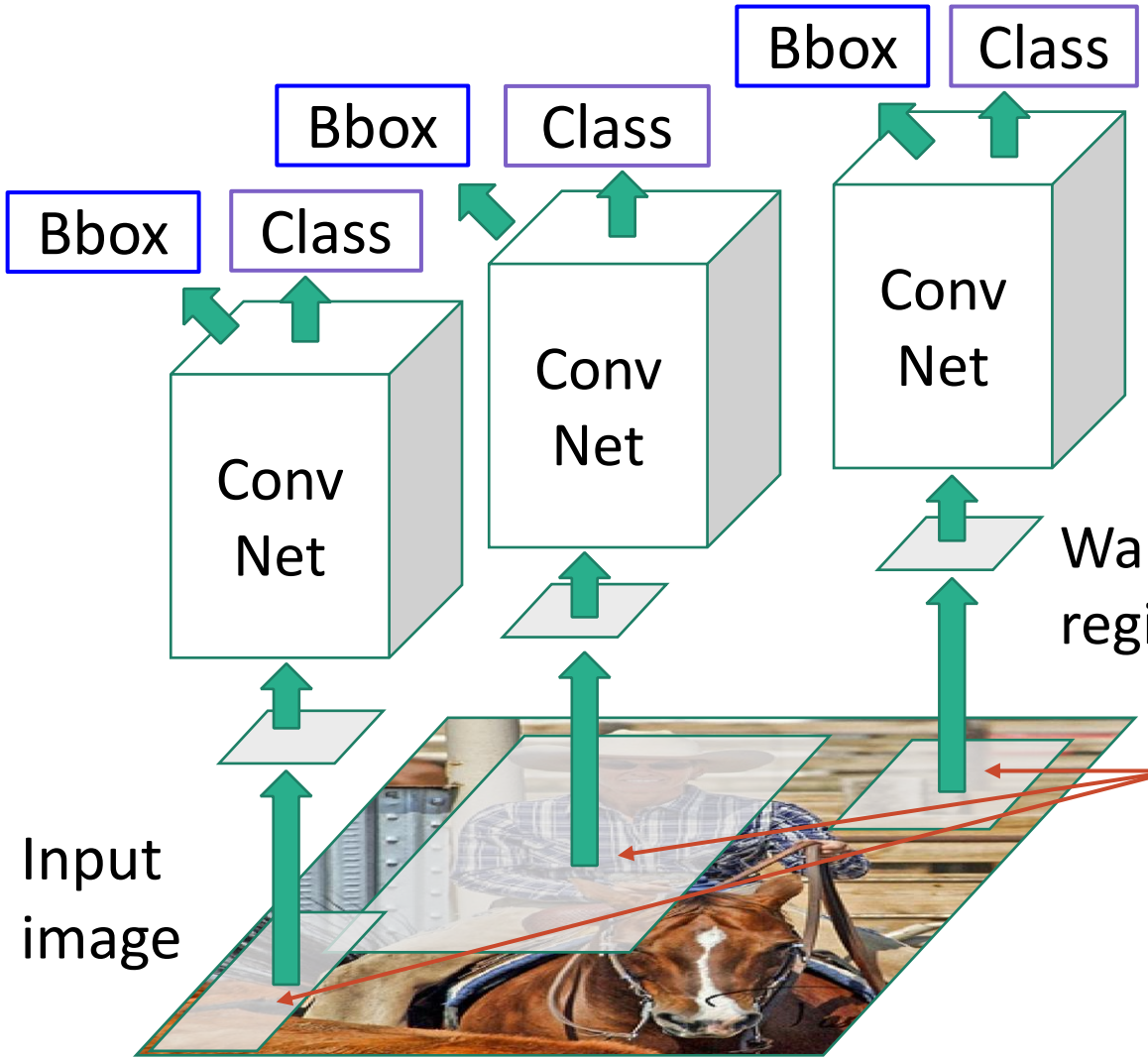


Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN: Region-Based CNN

Classify each region

Bounding box regression:
Predict "transform" to correct the RoI: 4 numbers (t_x, t_y, t_h, t_w)



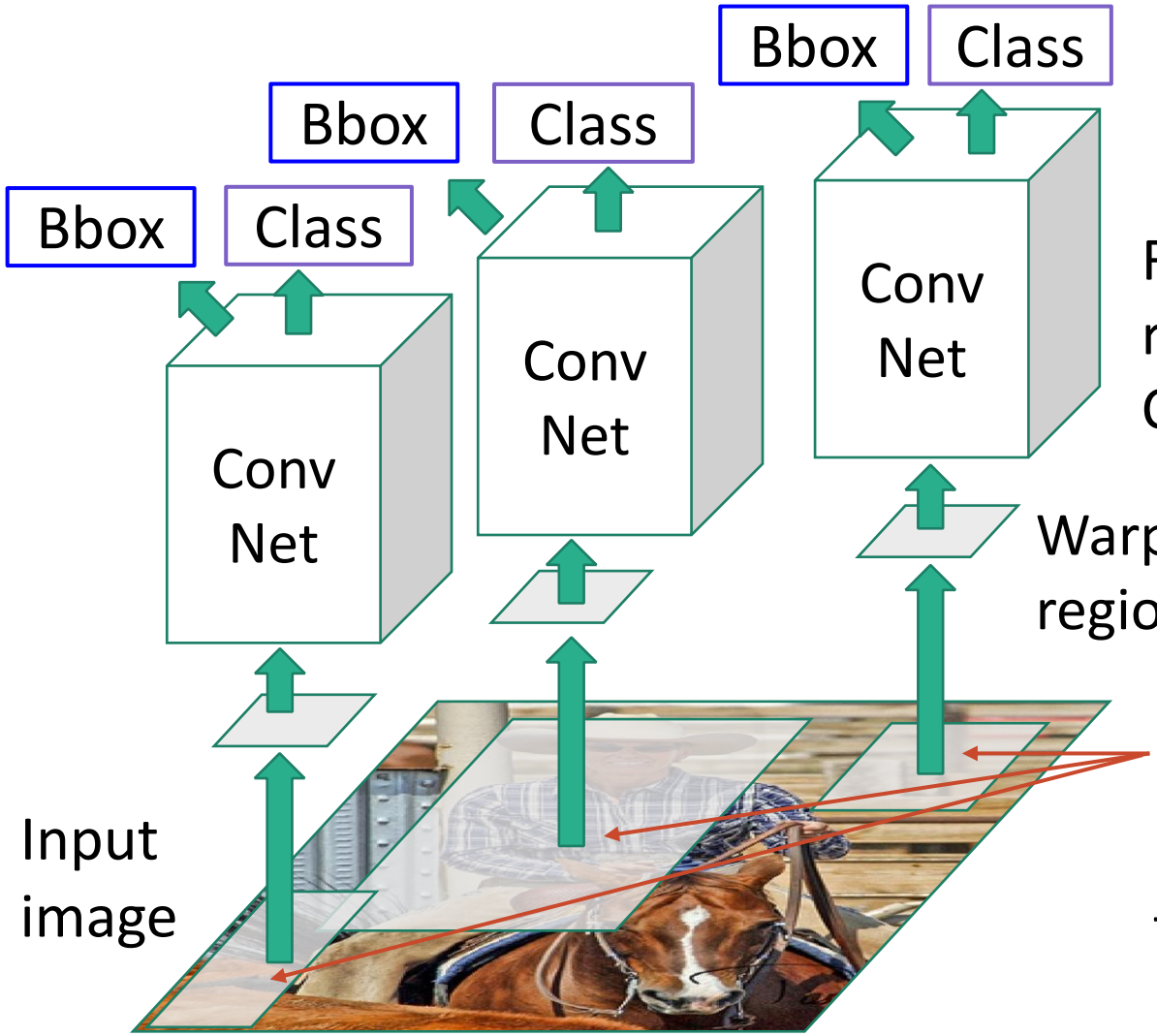
Forward each region through ConvNet

Warped image regions (224x224)

Regions of Interest (RoI) from a proposal method (~2k)

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN: Region-Based CNN



Forward each region through ConvNet

Warped image regions (224x224)

Regions of Interest (RoI) from a proposal method (~2k)

Classify each region

Bounding box regression:
Predict "transform" to correct the RoI: 4 numbers (t_x, t_y, t_h, t_w)

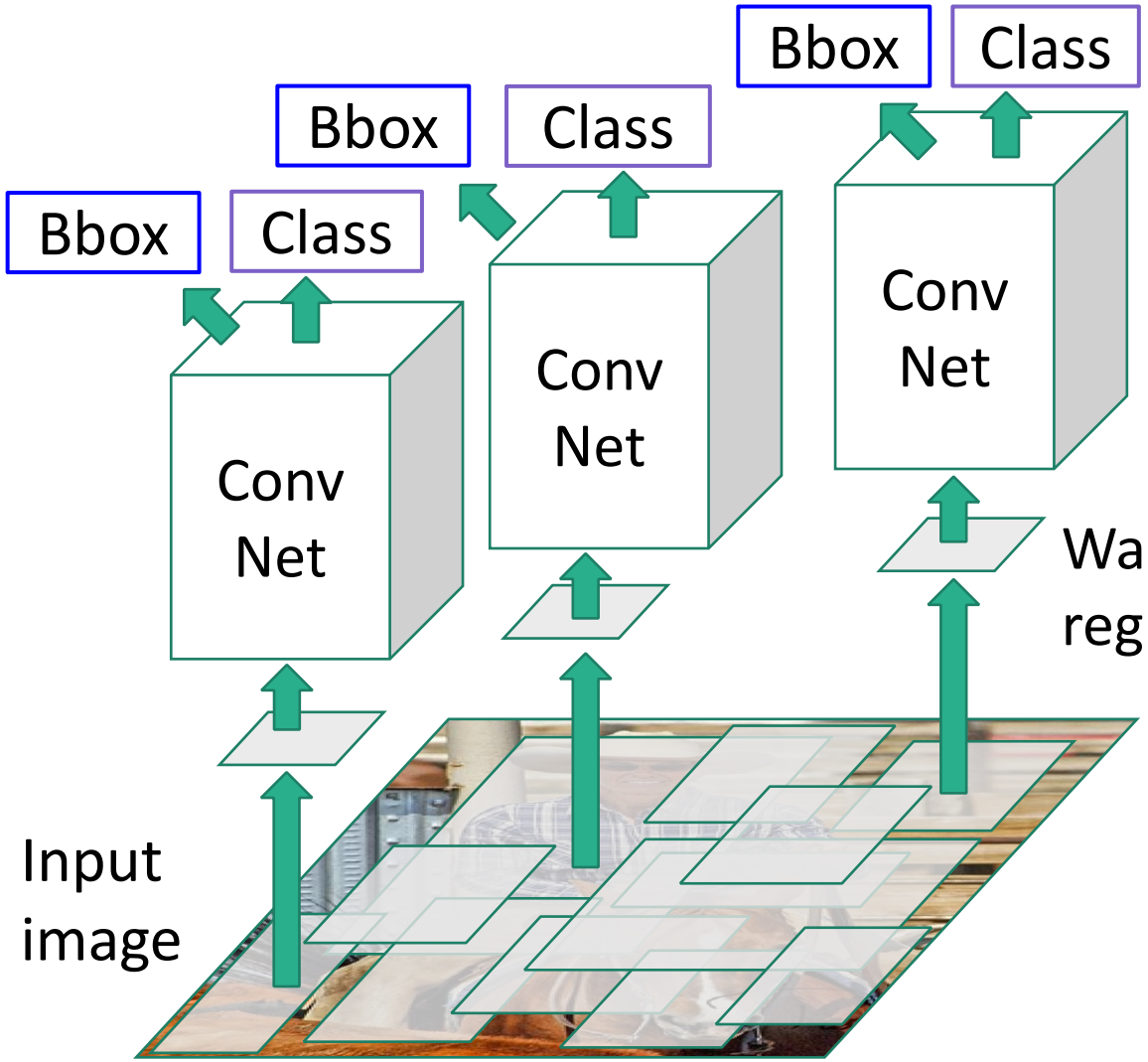
Region proposal: (p_x, p_y, p_h, p_w)
Transform: (t_x, t_y, t_h, t_w)
Output box: (b_x, b_y, b_h, b_w)

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

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

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

R-CNN: Region-Based CNN



Forward each region through ConvNet

Warped image regions (224x224)

Regions of Interest (RoI) from a proposal method (~2k)

Classify each region

Bounding box regression:
Predict "transform" to correct the RoI: 4 numbers (t_x, t_y, t_h, t_w)

Problem: Very slow! Need to do ~ 2k forward passes through CNN per image.

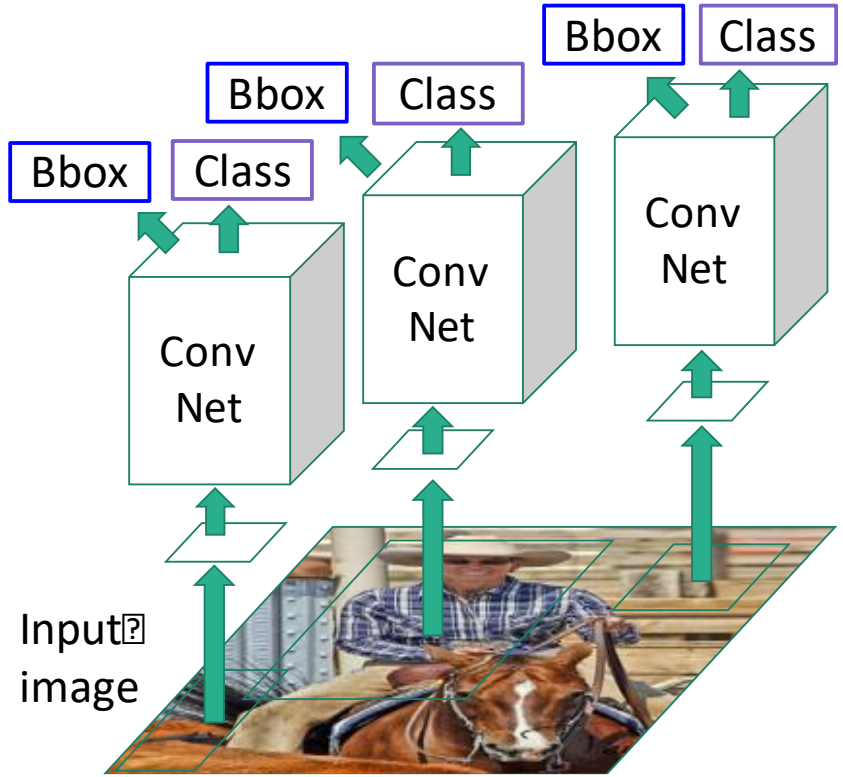
Idea: Pass the image through ConvNet before cropping!

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN

Input image

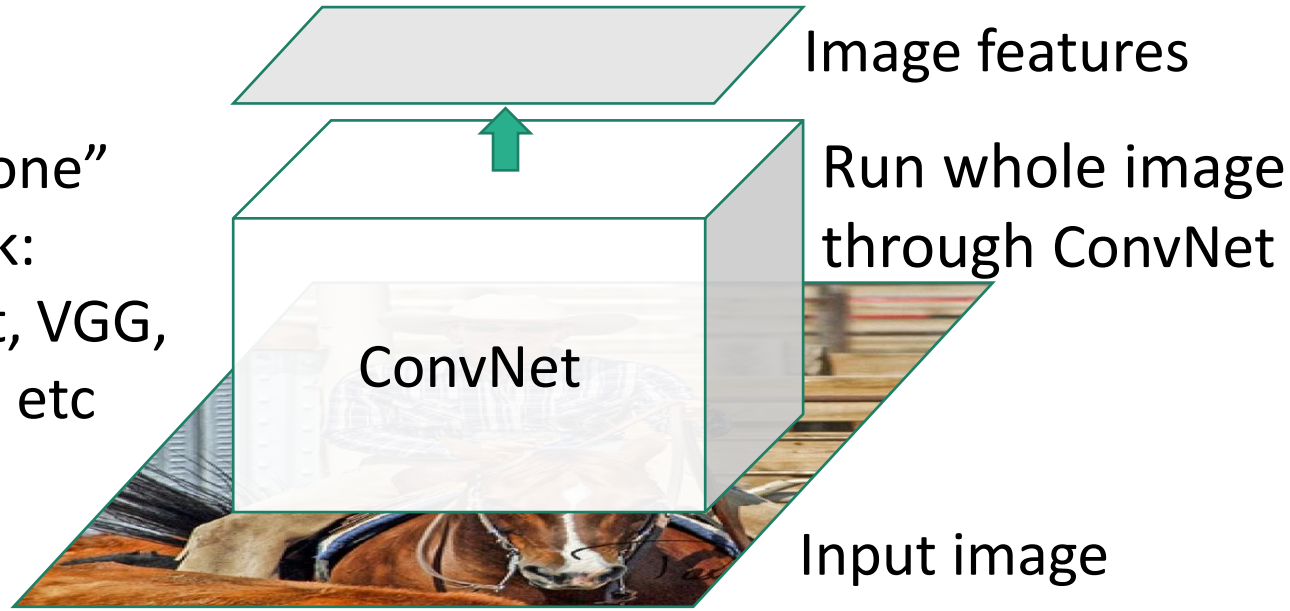
"Slow" R-CNN
Process each region independently



Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

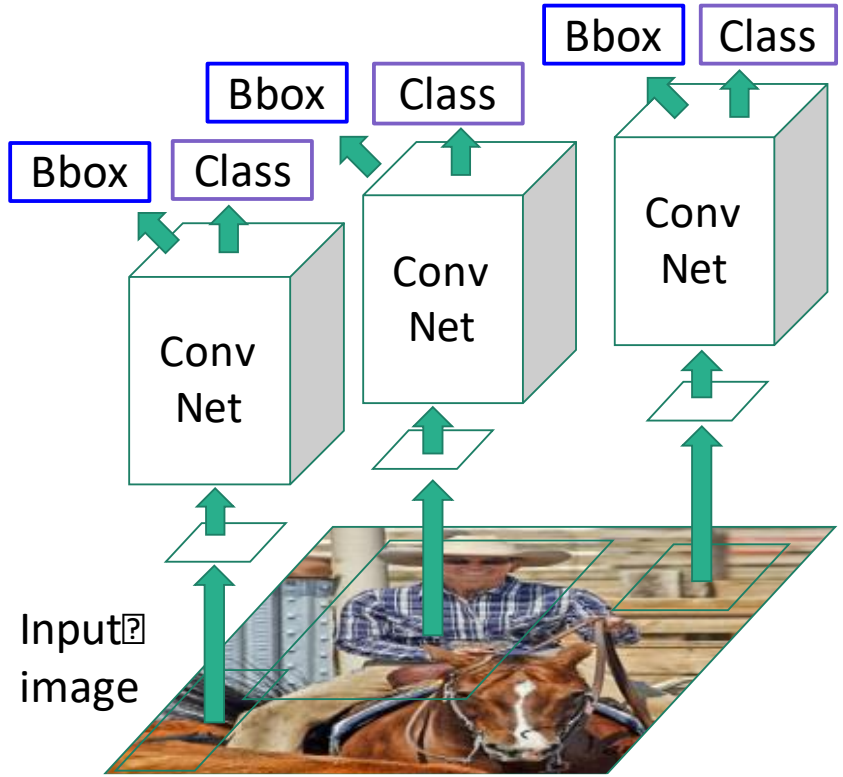
Fast R-CNN

"Backbone" network: AlexNet, VGG, ResNet, etc



"Slow" R-CNN

Process each region independently

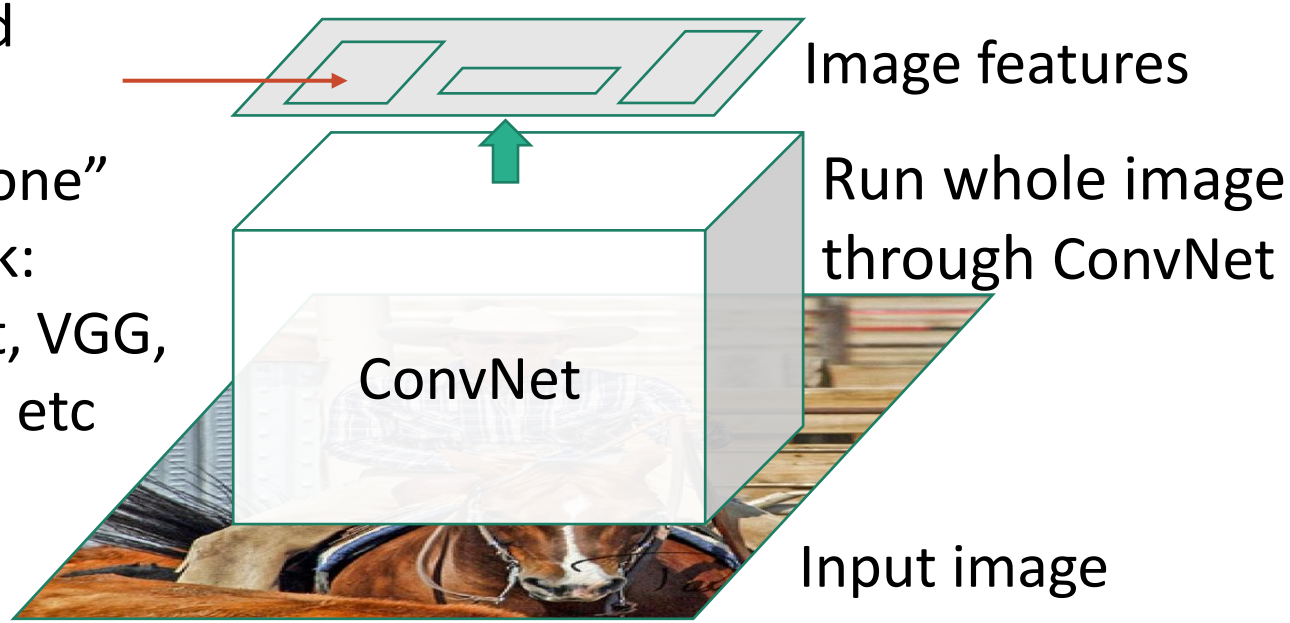


Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN

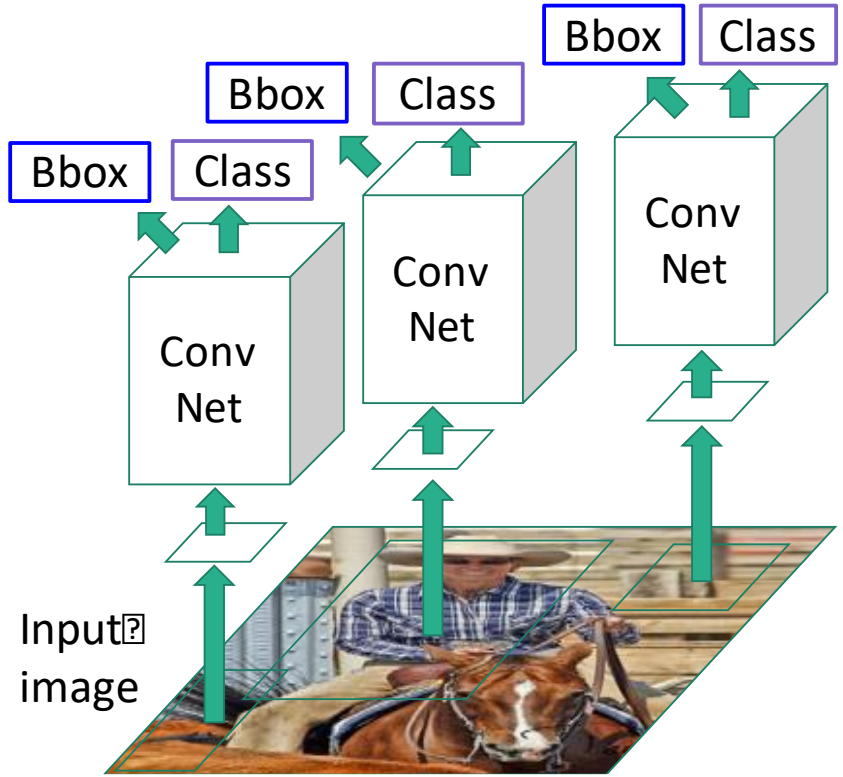
Regions of Interest (RoIs) from a proposal method

"Backbone" network: AlexNet, VGG, ResNet, etc



"Slow" R-CNN

Process each region independently

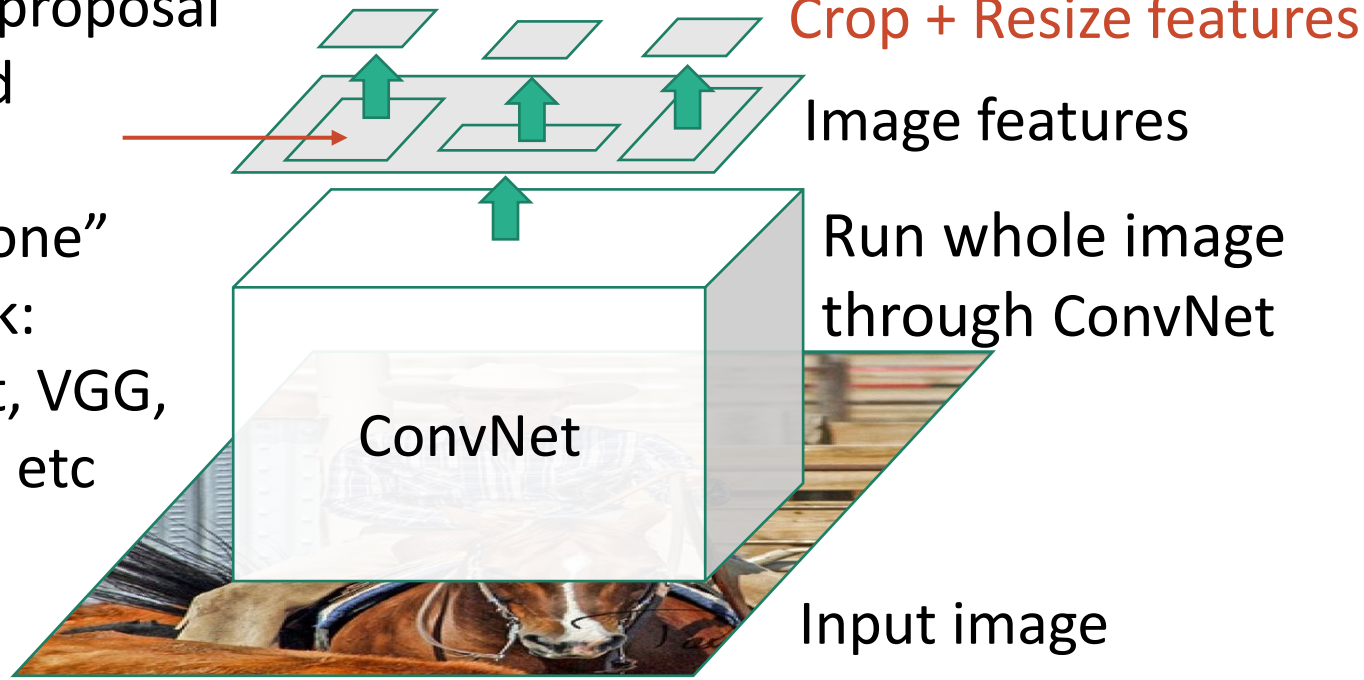


Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN

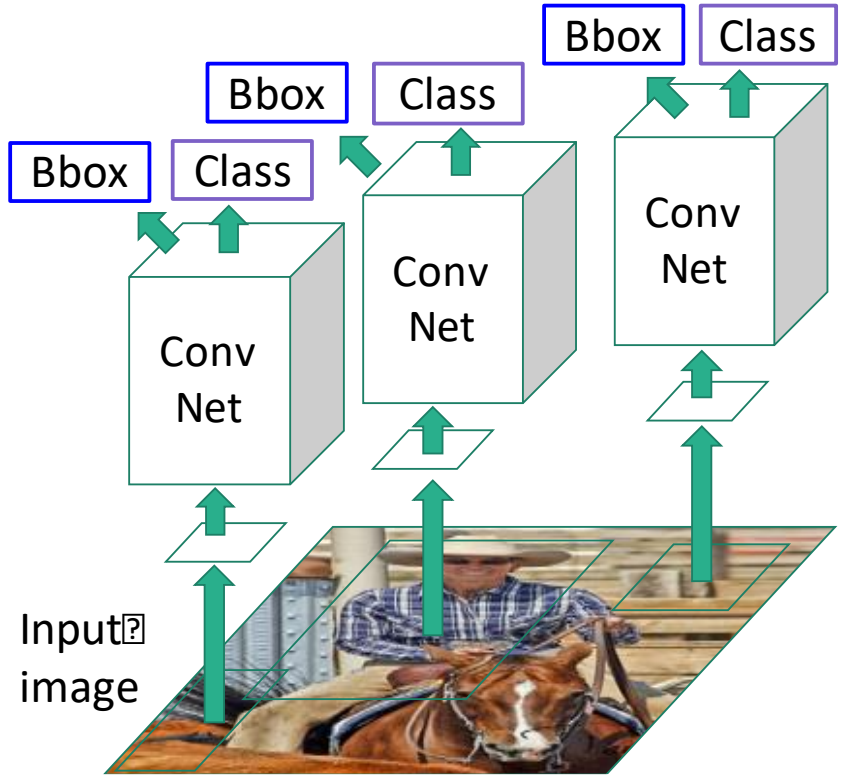
Regions of Interest (RoIs) from a proposal method

"Backbone" network: AlexNet, VGG, ResNet, etc



"Slow" R-CNN

Process each region independently

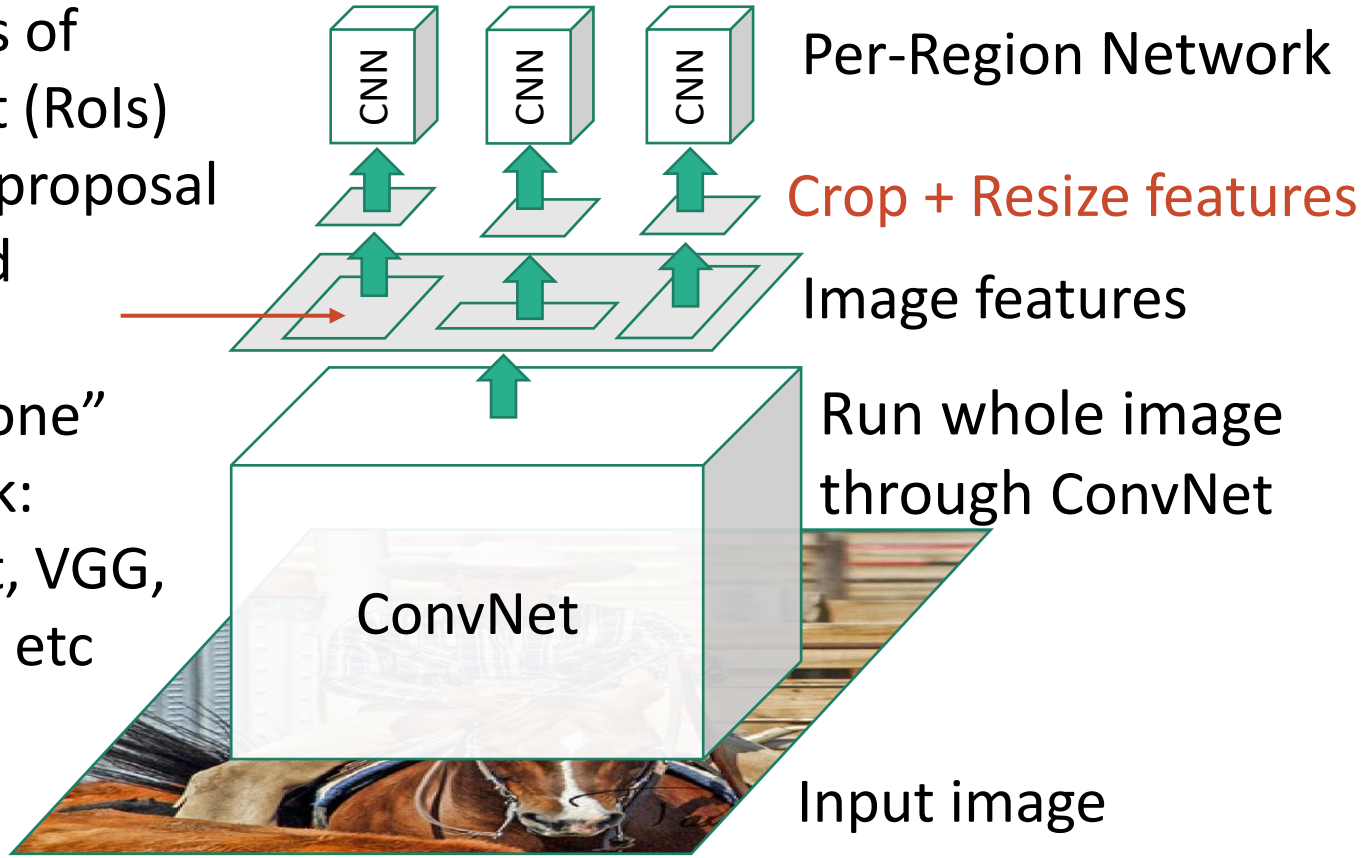


Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN

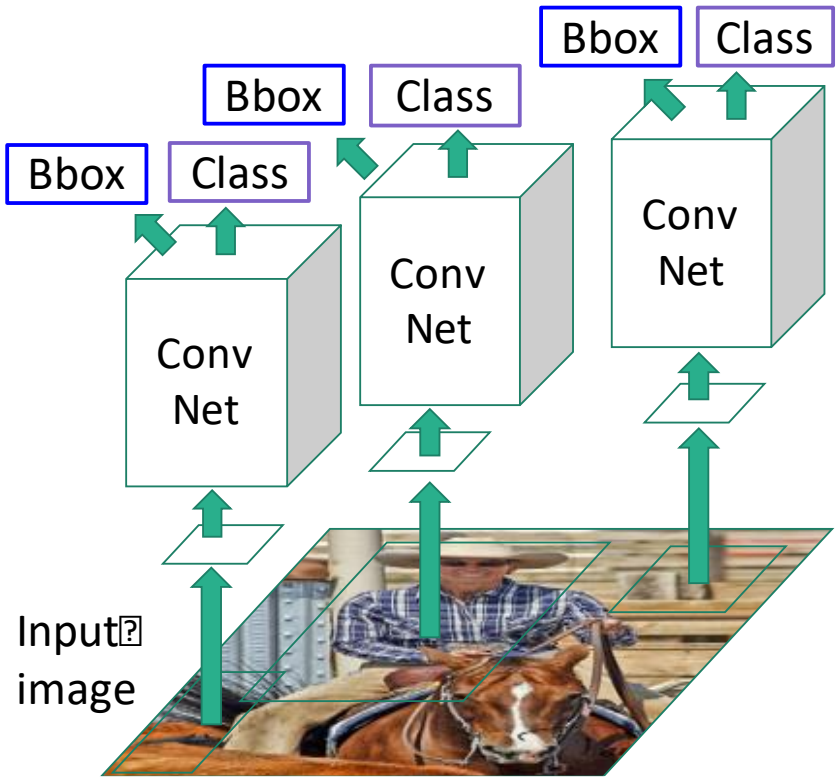
Regions of Interest (RoIs) from a proposal method

"Backbone" network: AlexNet, VGG, ResNet, etc



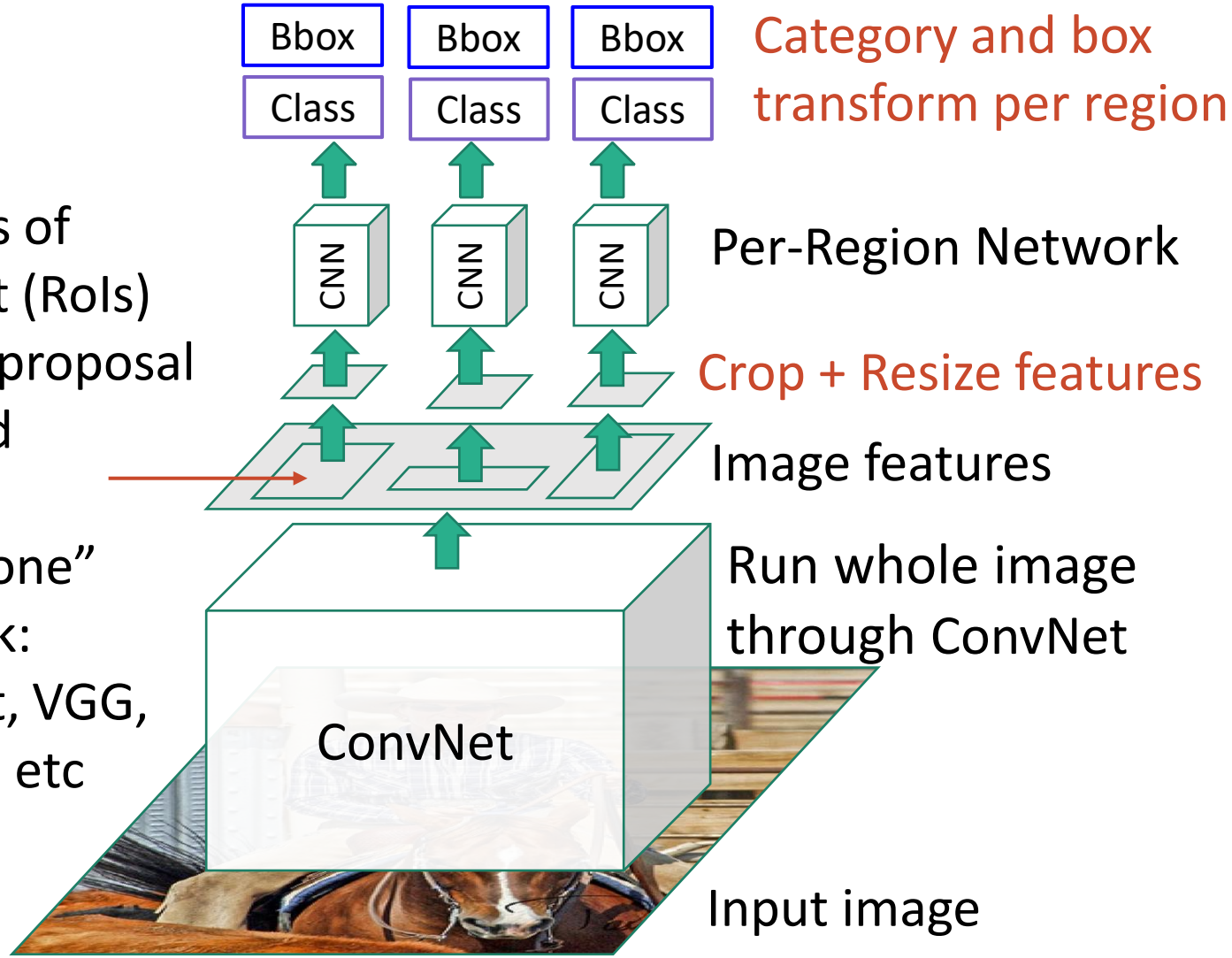
"Slow" R-CNN

Process each region independently



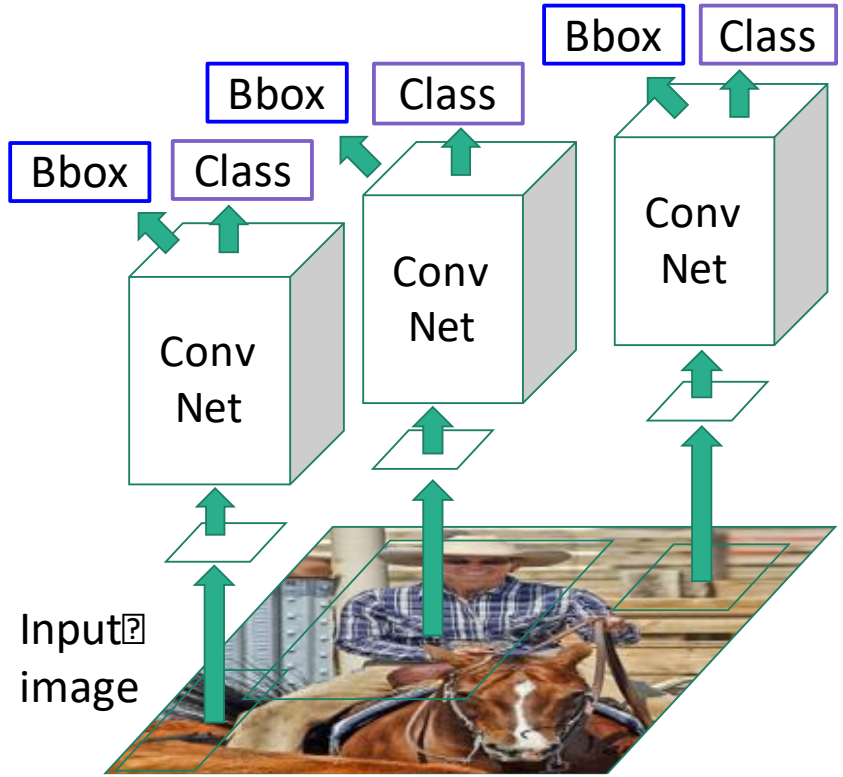
Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN



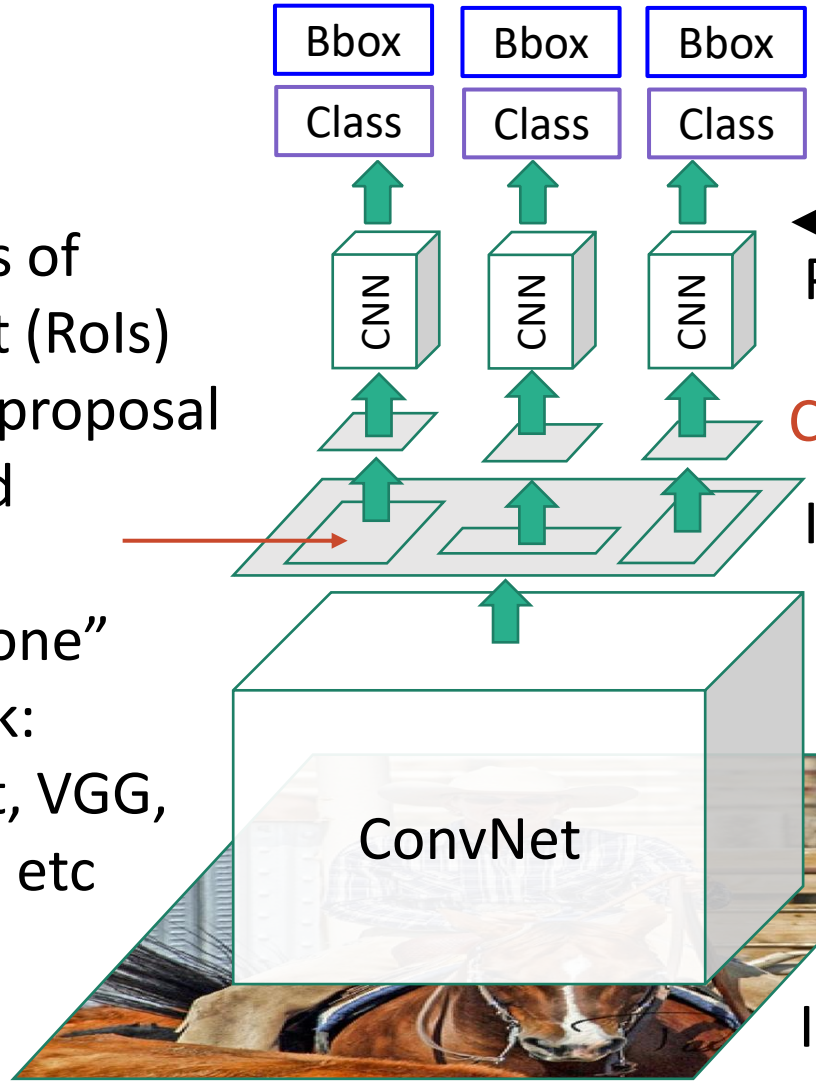
"Slow" R-CNN

Process each region independently



Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN



Category and box transform per region

Per-Region network is relatively lightweight

Per-Region Network

Regions of Interest (RoIs) from a proposal method

Crop + Resize features

Image features

"Backbone" network: AlexNet, VGG, ResNet, etc

Run whole image through ConvNet

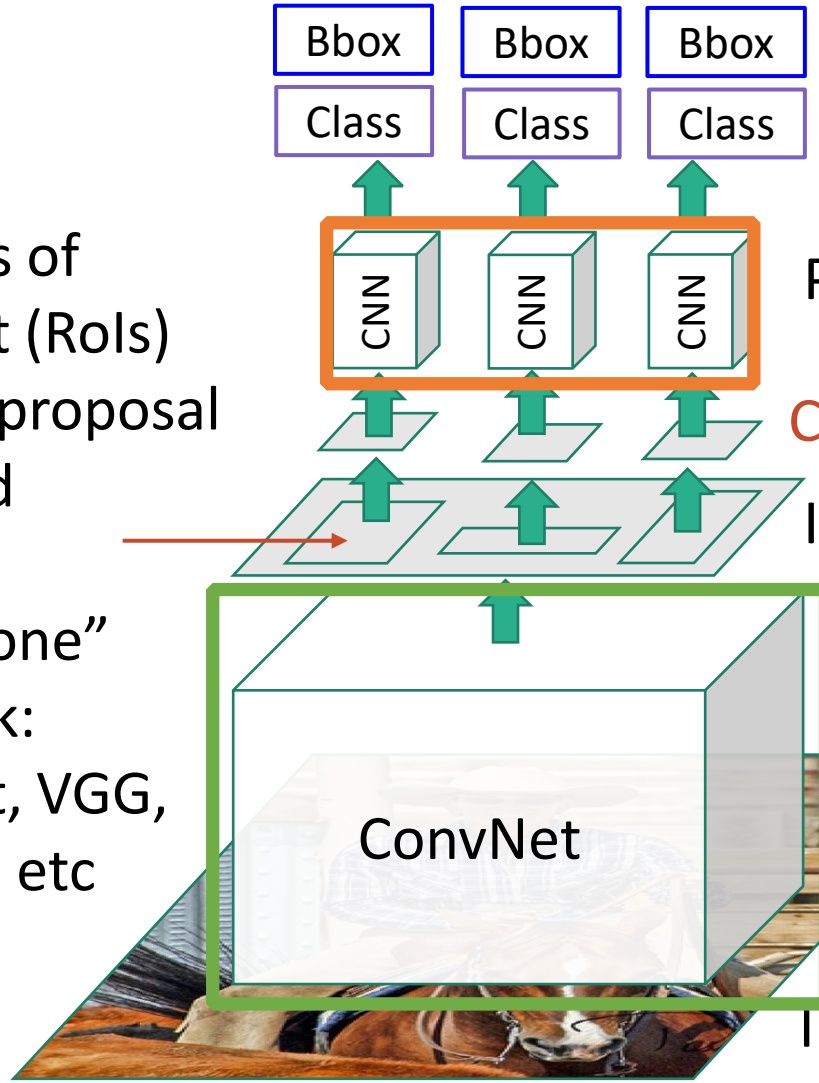
ConvNet

Most of the computation happens in backbone network; this saves work for overlapping region proposals

Input image

Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN



Category and box transform per region

Per-Region Network

Crop + Resize features

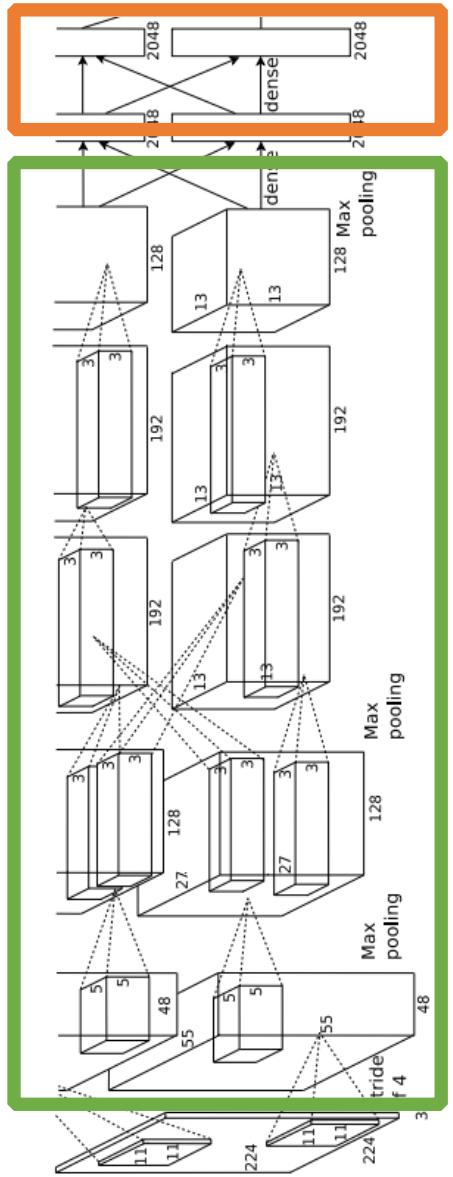
Image features

Run whole image through ConvNet

Input image

Regions of Interest (RoIs) from a proposal method

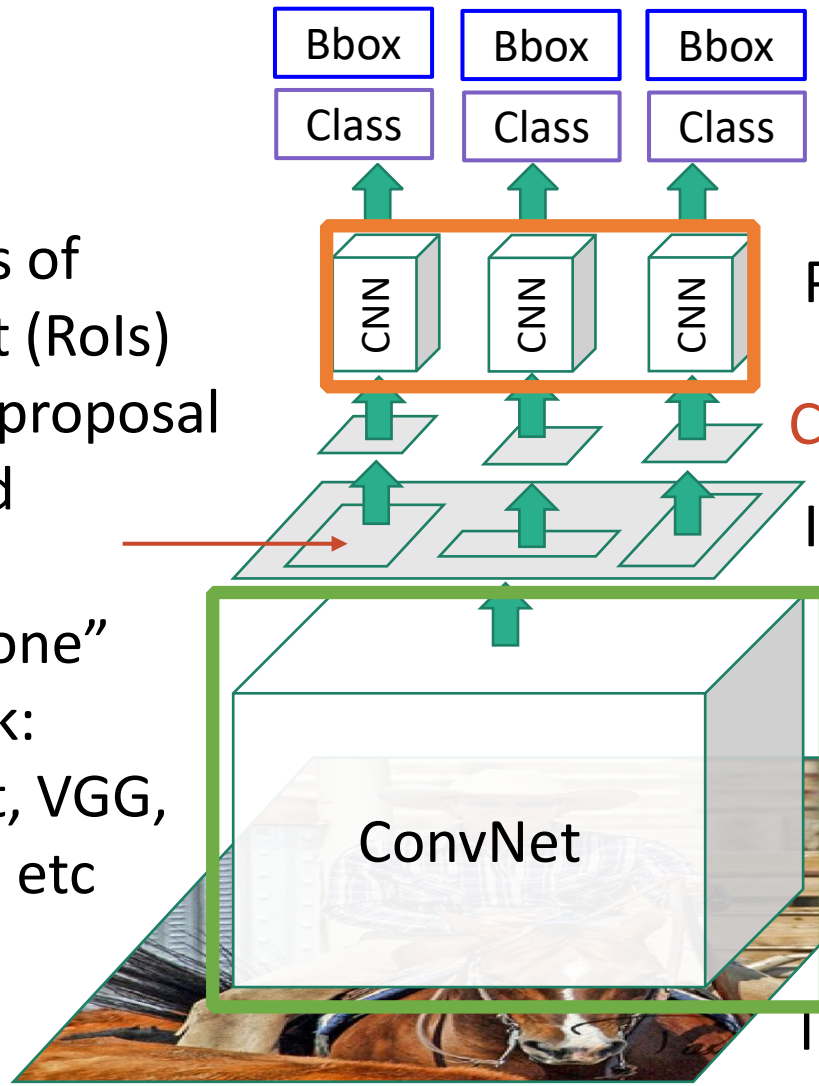
"Backbone" network: AlexNet, VGG, ResNet, etc



Example: When using AlexNet for detection, five conv layers are used for backbone and two FC layers are used for per-region network

Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN



Category and box transform per region

Per-Region Network

Crop + Resize features

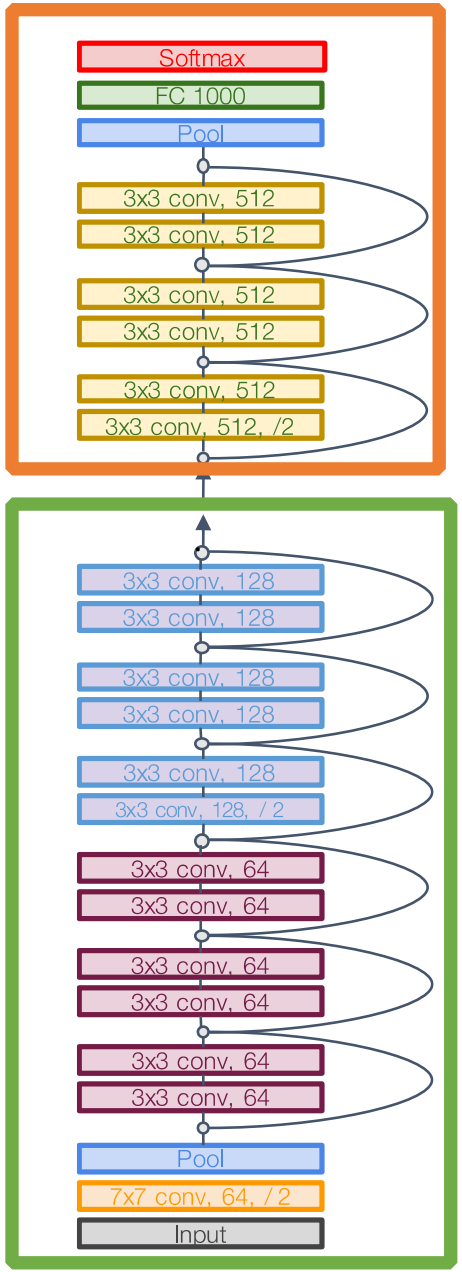
Image features

Run whole image through ConvNet

Input image

Regions of Interest (RoIs) from a proposal method

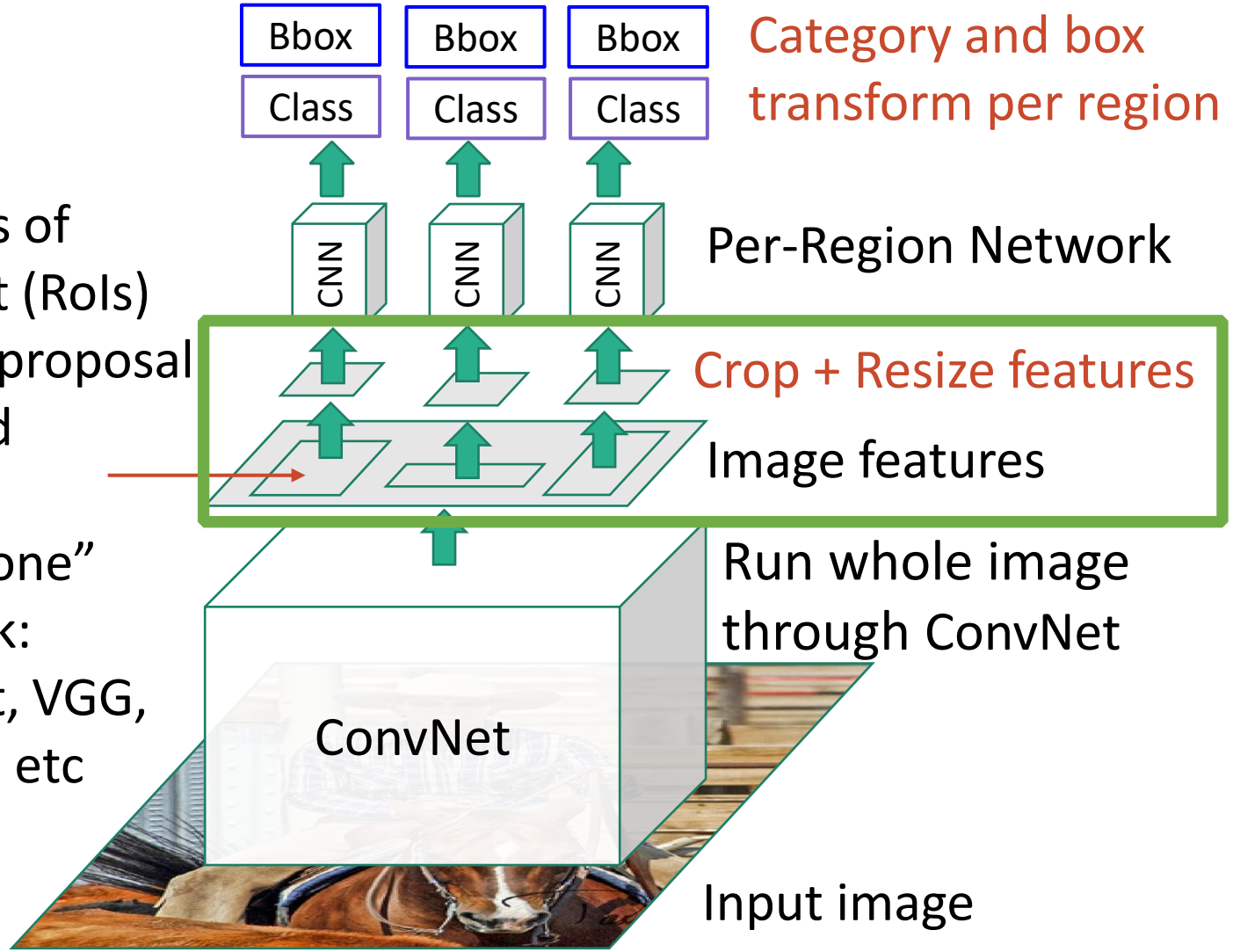
"Backbone" network: AlexNet, VGG, ResNet, etc



Example:
For ResNet, last stage is used as per-region network; the rest of the network is used as backbone

Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN



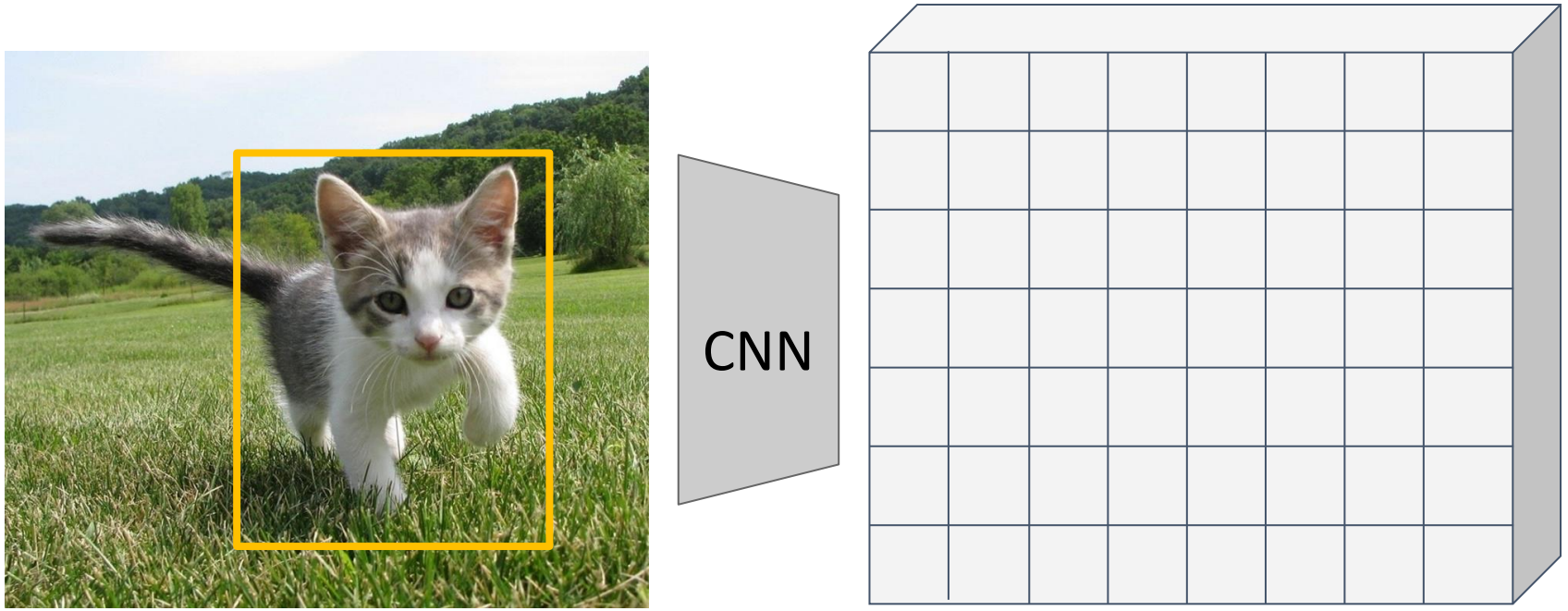
Regions of Interest (RoIs) from a proposal method

"Backbone" network: AlexNet, VGG, ResNet, etc

How to crop features?

Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Cropping Features: RoI Pool



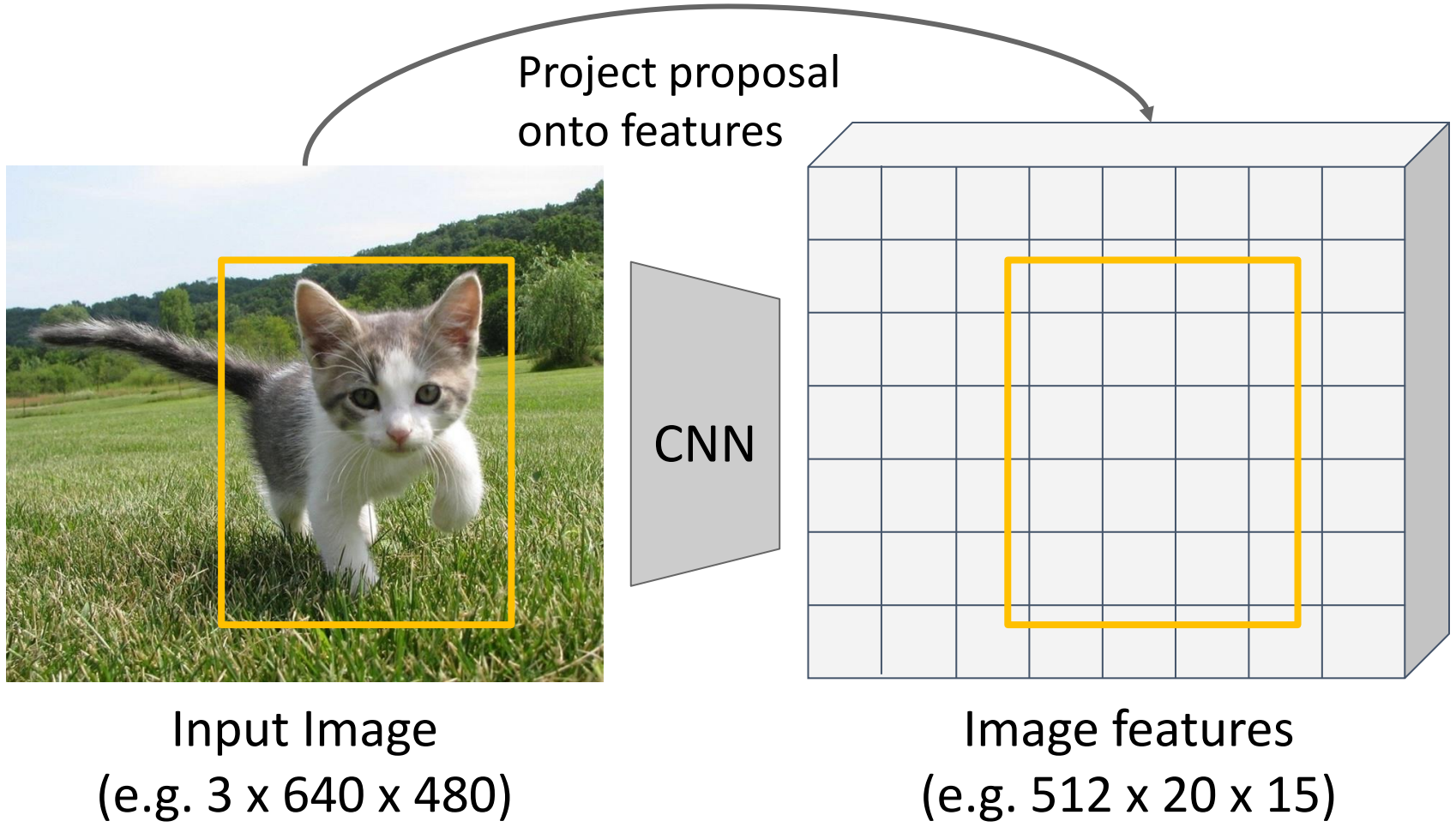
Input Image
(e.g. 3 x 640 x 480)

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

Want features for the box of a fixed size (2x2 in this example, 7x7 or 14x14 in practice)

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

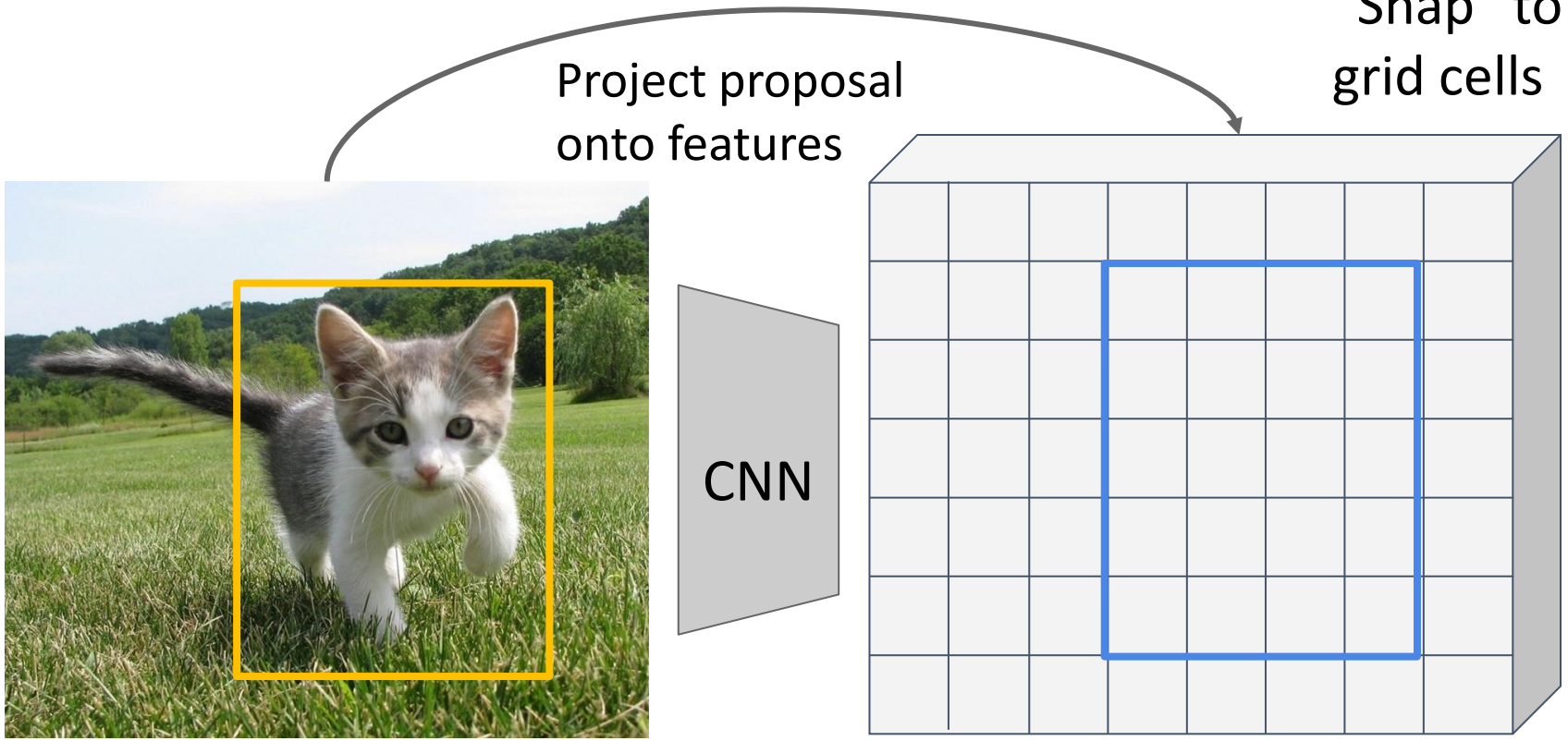
Cropping Features: RoI Pool



Want features for the box of a fixed size (2x2 in this example, 7x7 or 14x14 in practice)

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

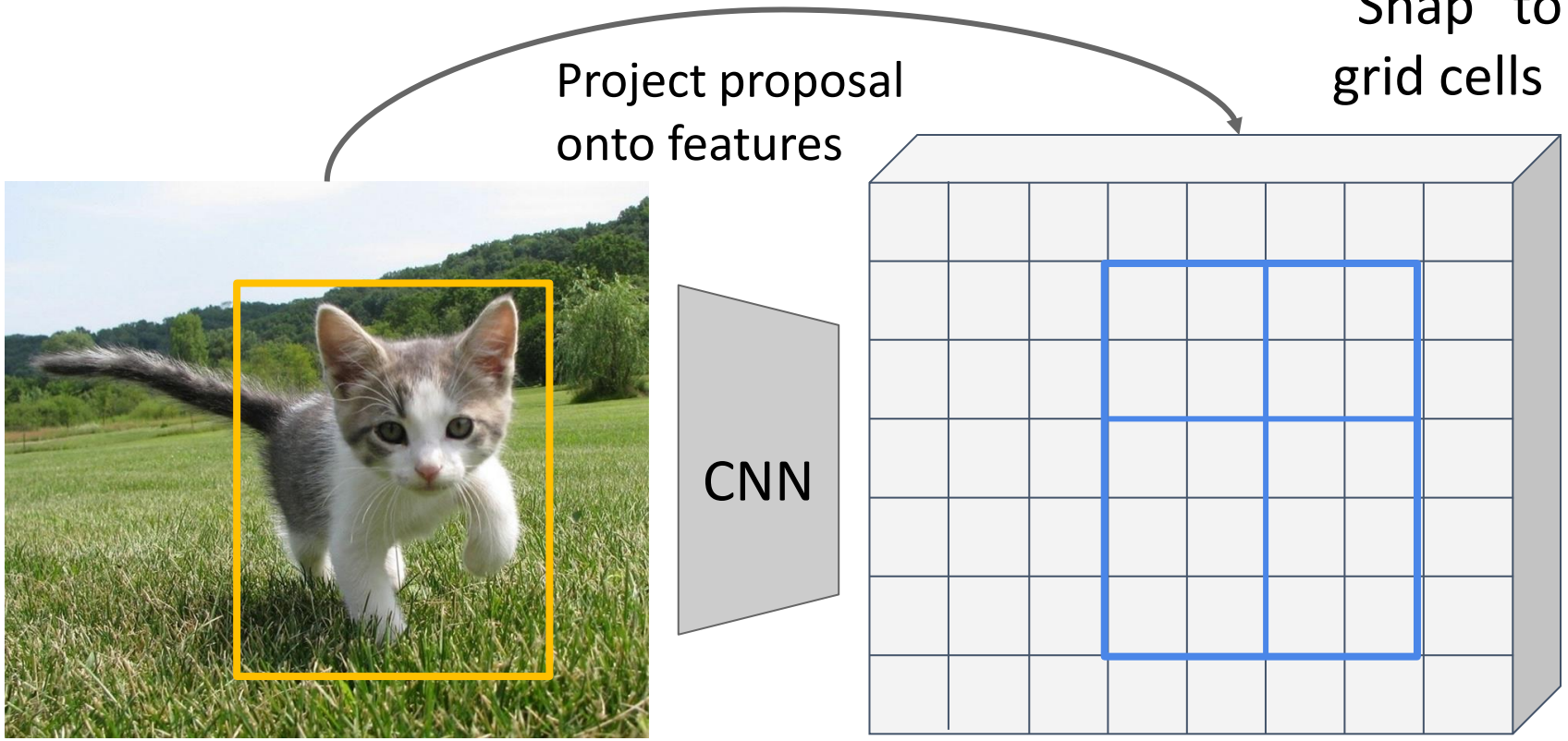
Cropping Features: RoI Pool



Want features for the box of a fixed size (2x2 in this example, 7x7 or 14x14 in practice)

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

Cropping Features: RoI Pool



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

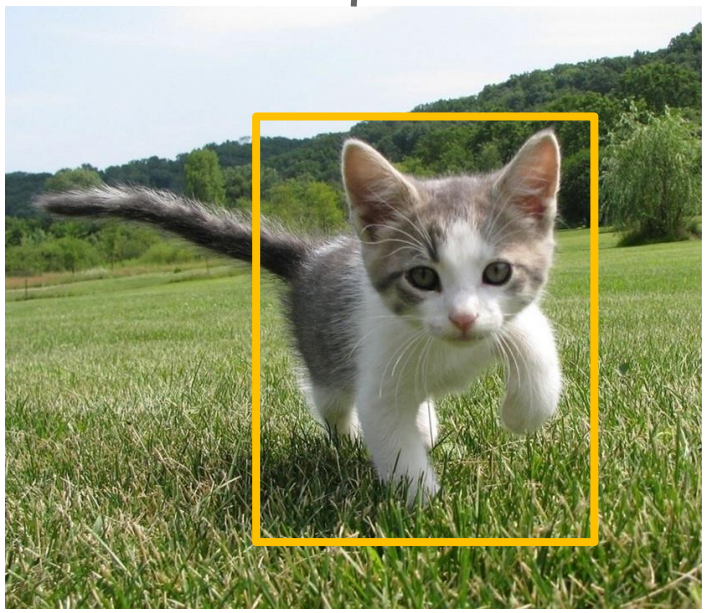
Image features
(e.g. 512 x 20 x 15)

Divide into 2x2
grid of (roughly)
equal subregions

Want features for the
box of a fixed size
(2x2 in this example,
7x7 or 14x14 in practice)

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

Cropping Features: RoI Pool



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

Project proposal onto features

CNN

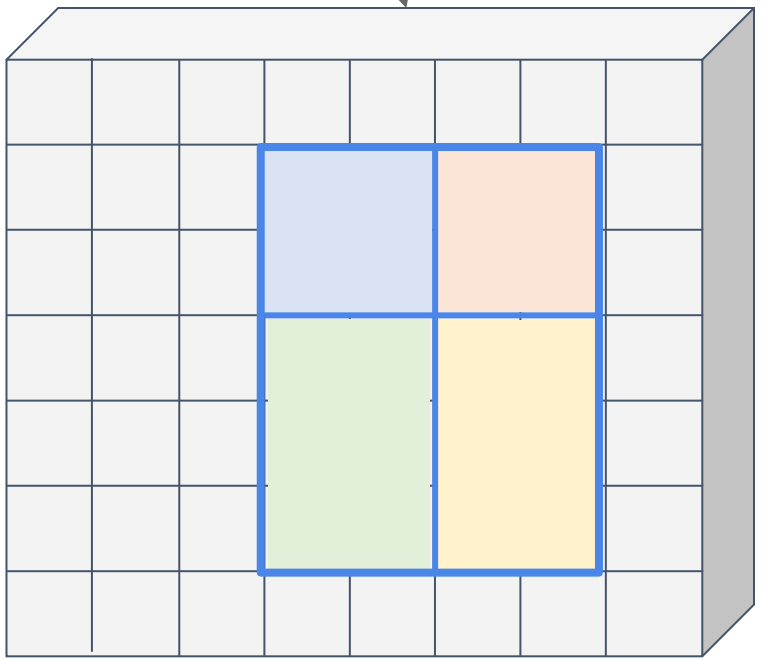
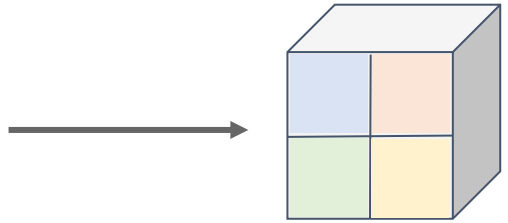


Image features
(e.g. 512 x 20 x 15)

"Snap" to grid cells

Divide into 2x2 grid of (roughly) equal subregions

Max-pool within each subregion

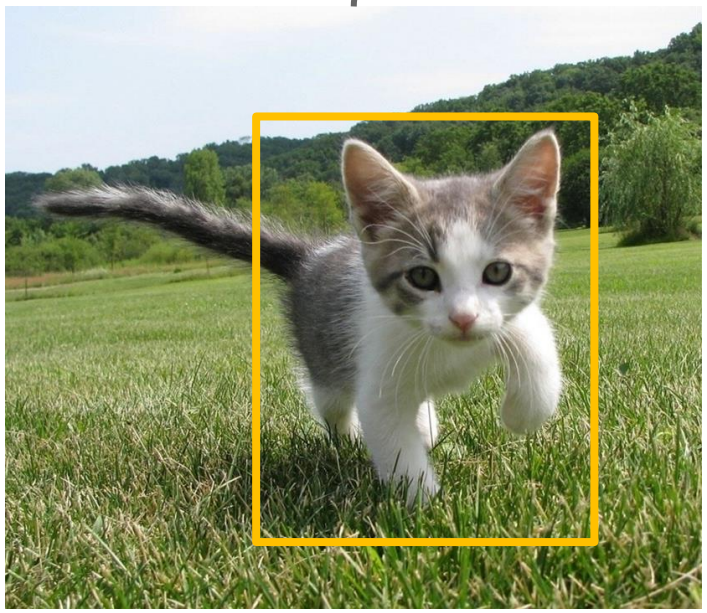


Region features
(here 512 x 2 x 2;
In practice 512x7x7)

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

CNN

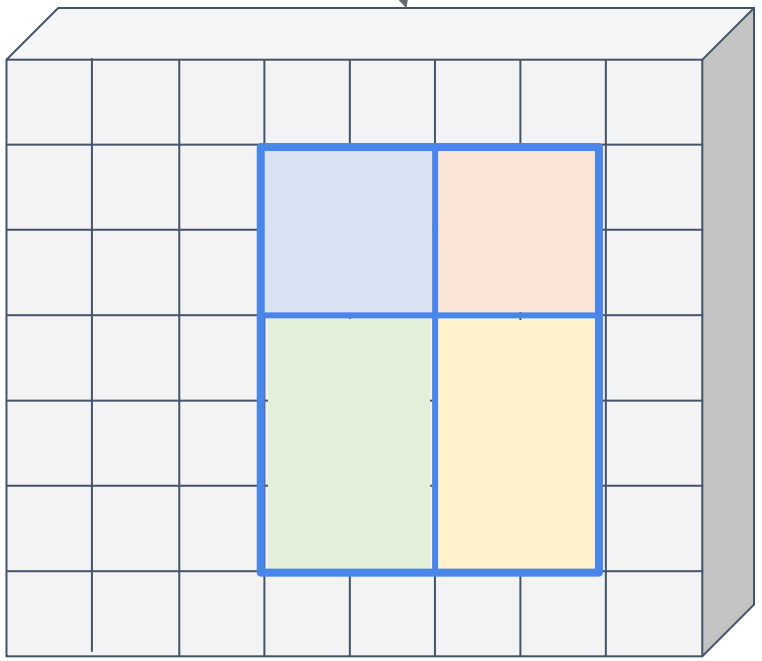
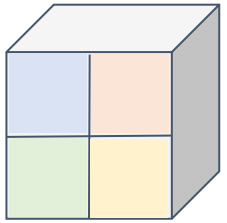


Image features
(e.g. 512 x 20 x 15)

"Snap" to grid cells

Divide into 2x2 grid of (roughly) equal subregions

Max-pool within each subregion



Region features
(here 512 x 2 x 2;
In practice 512x7x7)

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

Problem: Slight misalignment due to snapping; different-sized subregions is weird

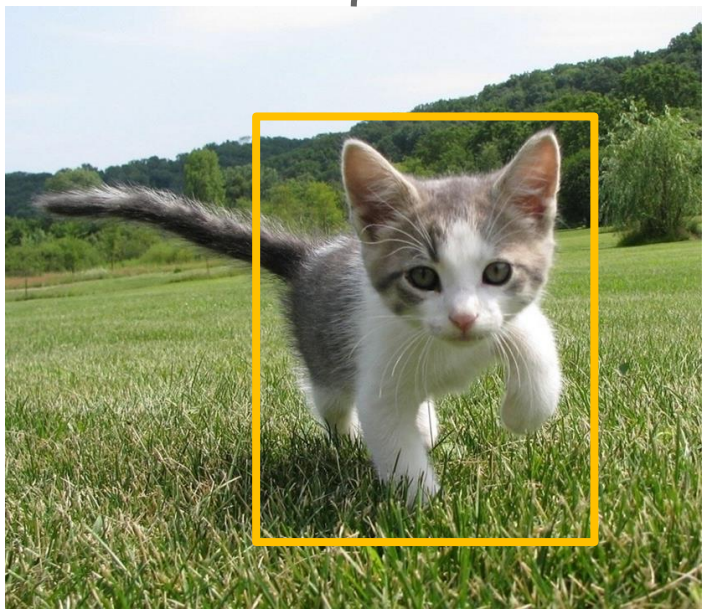
Girshick, "Fast R-CNN", ICCV 2015.

Cropping Features: RoI Align

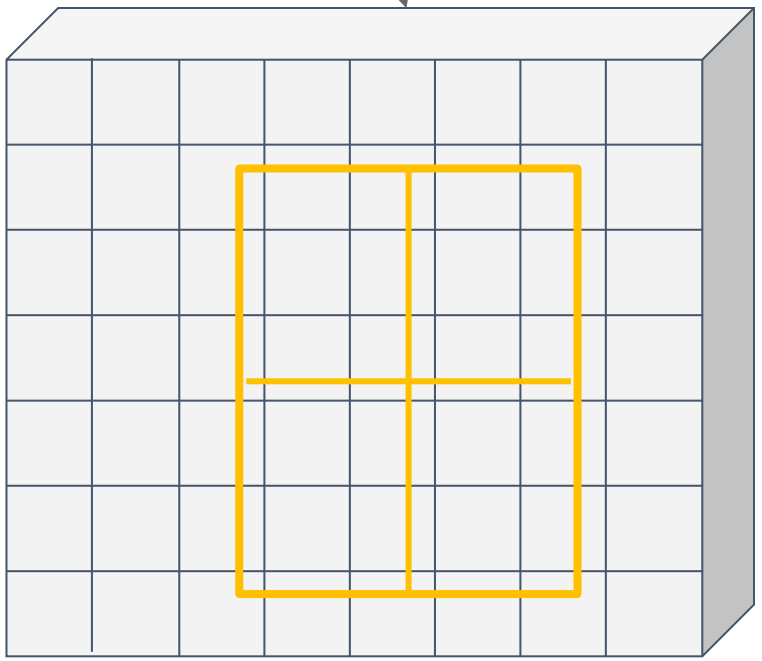
Divide into equal-sized subregions
(may not be aligned to grid!)

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)

Want features for the
box of a fixed size
(2x2 in this example,
7x7 or 14x14 in practice)

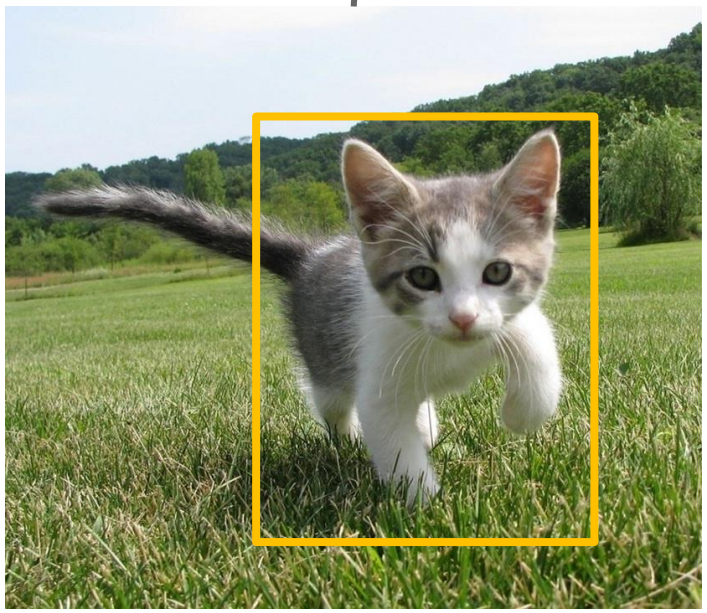
He et al, “Mask R-CNN”, ICCV 2017.

Cropping Features: RoI Align

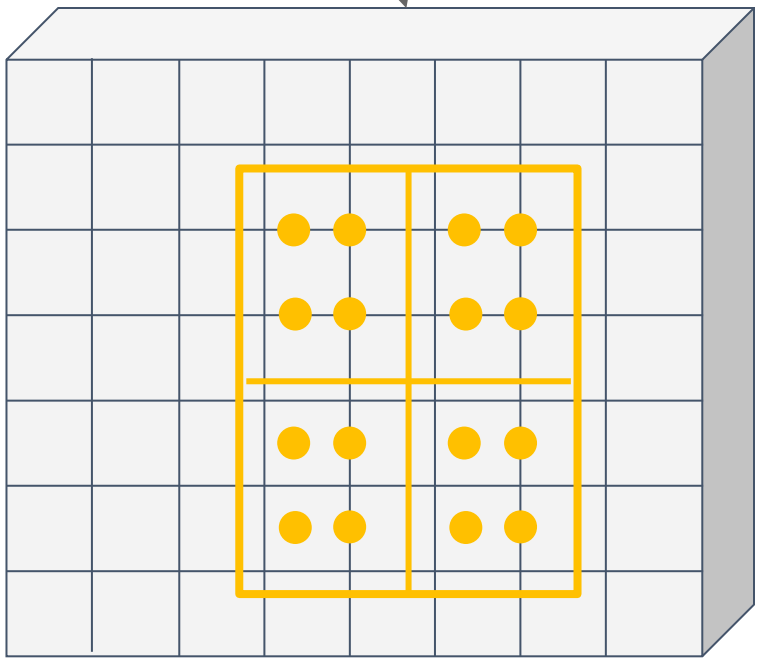
Divide into equal-sized subregions (may not be aligned to grid!)

No “snapping”!

Project proposal onto features



CNN



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**

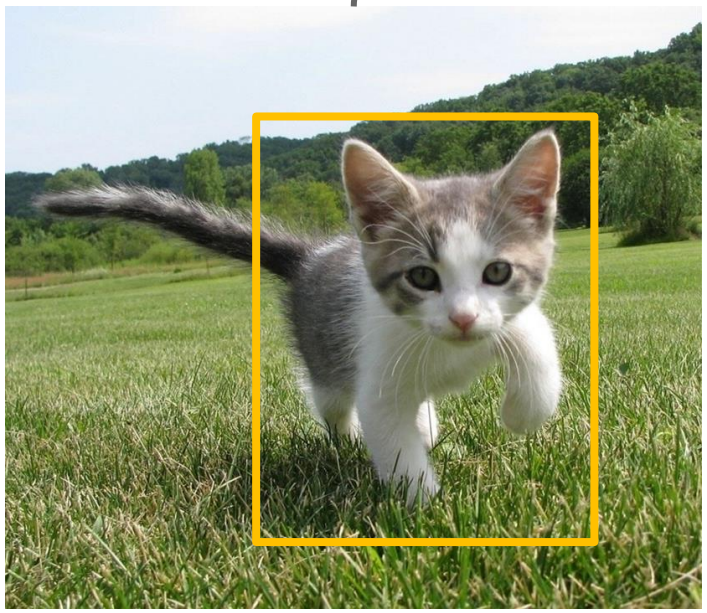
He et al, “Mask R-CNN”, ICCV 2017

Cropping Features: RoI Align

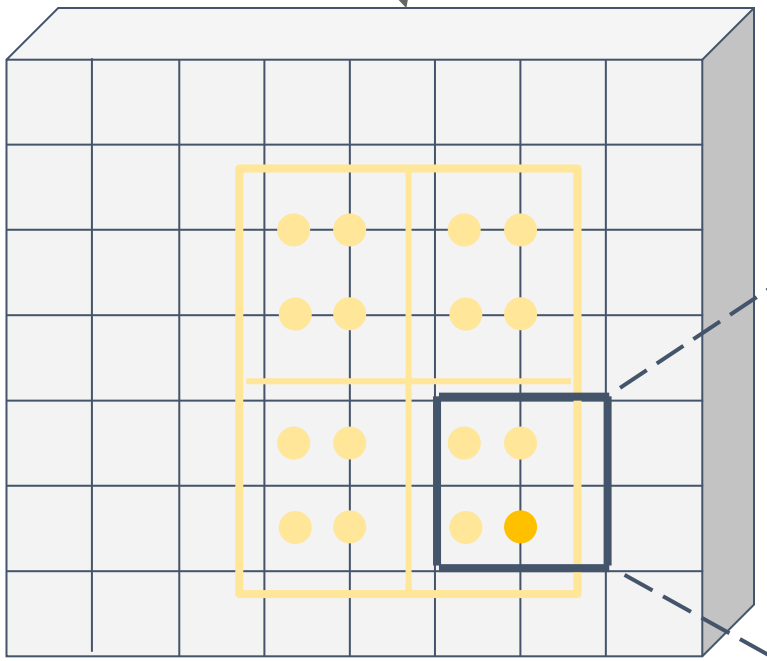
Divide into equal-sized subregions (may not be aligned to grid!)

No "snapping"!

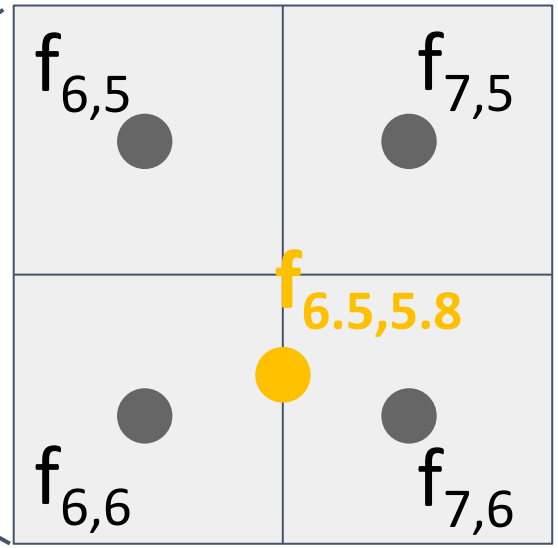
Project proposal onto features



CNN



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



$$f_{xy} = \sum_{i,j=1}^{2} f_{i,j} \max(0, 1 - |x - x_i|) \max(0, 1 - |y - y_j|)$$

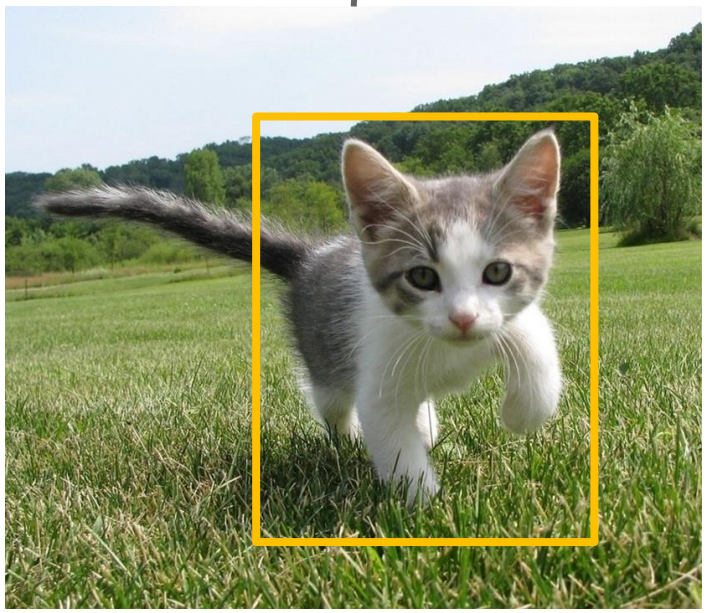
Feature f_{xy} for point (x, y) is a linear combination of features at its four neighboring grid cells:

Cropping Features: RoI Align

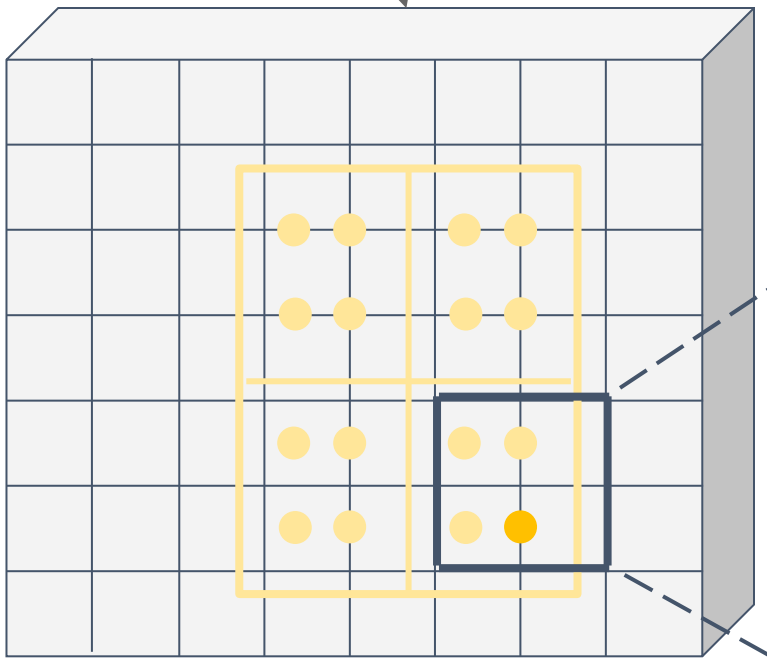
Divide into equal-sized subregions (may not be aligned to grid!)

No "snapping"!

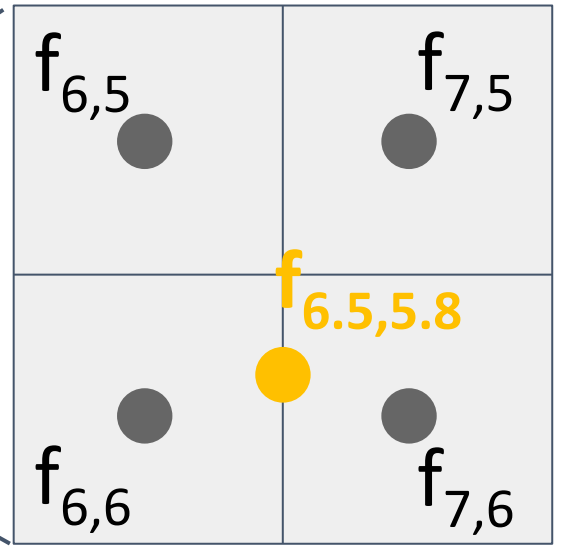
Project proposal onto features



CNN



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

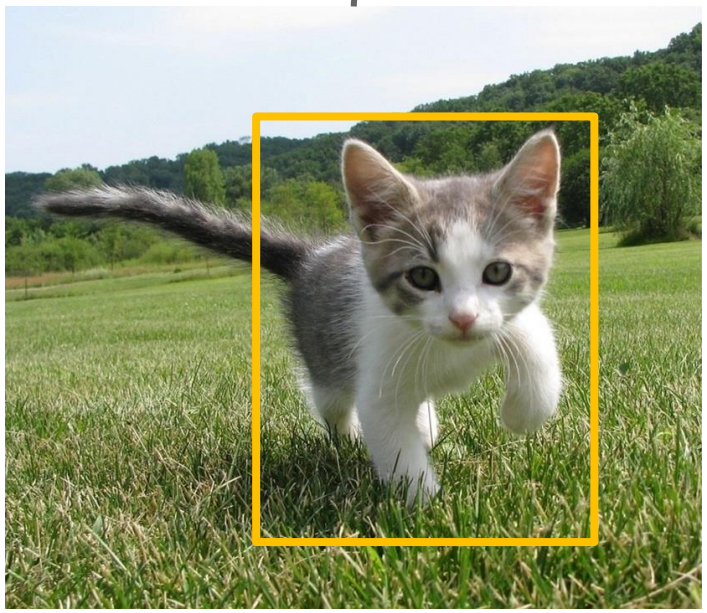


$$f_{xy} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - x_i|) \max(0, 1 - |y - y_i|)$$

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

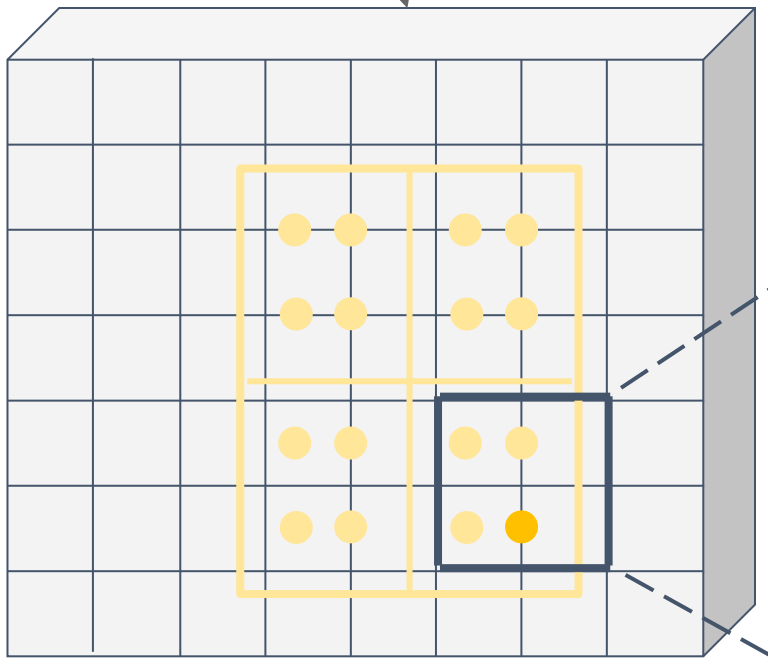
Feature f_{xy} for point (x, y) is a linear combination of features at its four neighboring grid cells:

Cropping Features: RoI Align



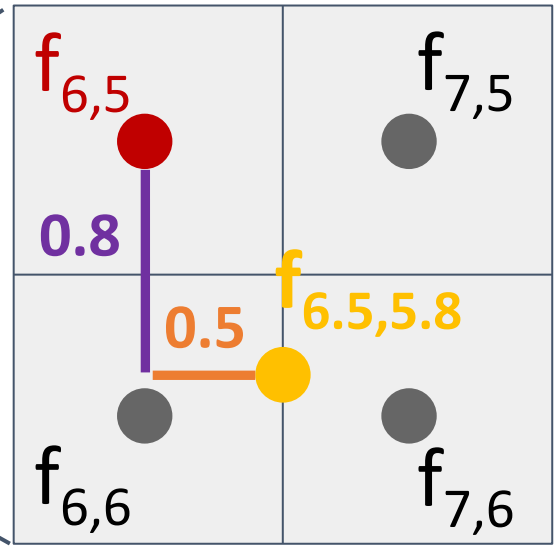
Project proposal onto features

CNN



No “snapping”!

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

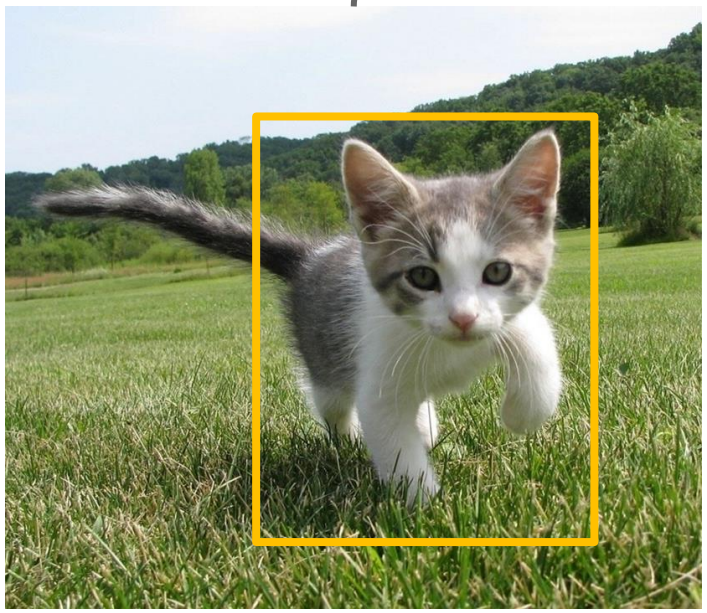


$$f_{xy} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - x_i|) \max(0, 1 - |y - y_i|)$$

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

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

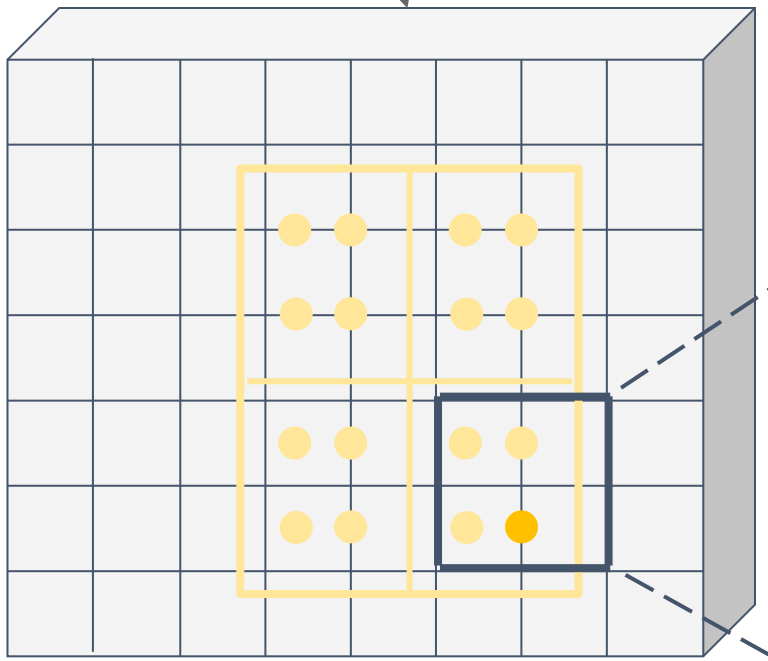
Cropping Features: RoI Align



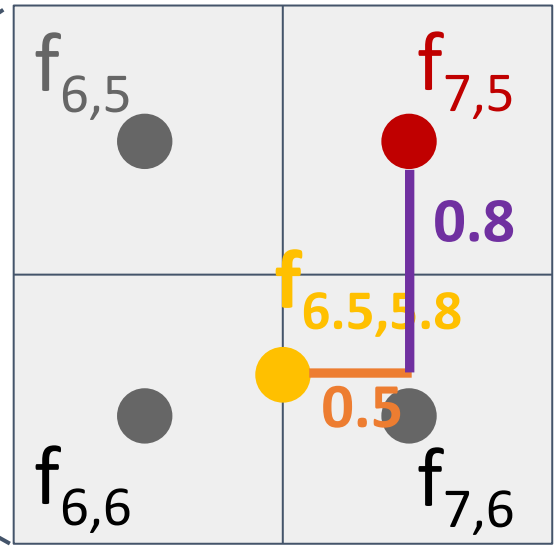
Project proposal onto features

CNN

No “snapping”!



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

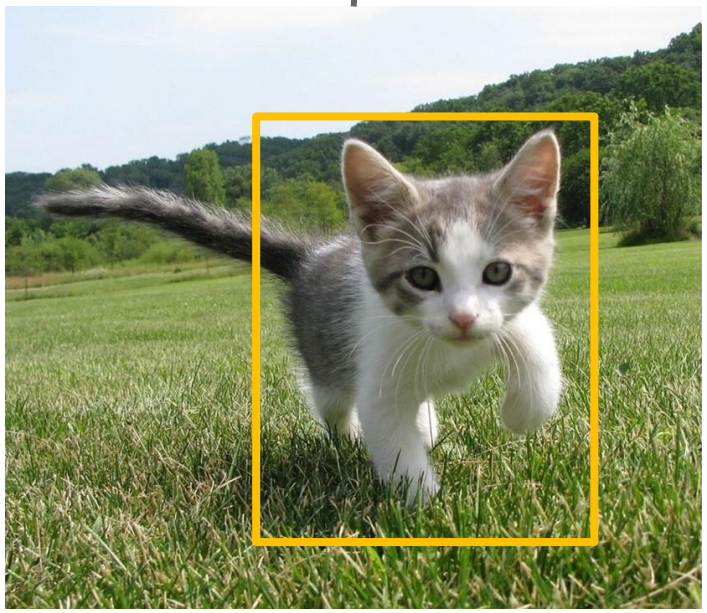


$$f_{xy} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - x_i|) \max(0, 1 - |y - y_i|)$$

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

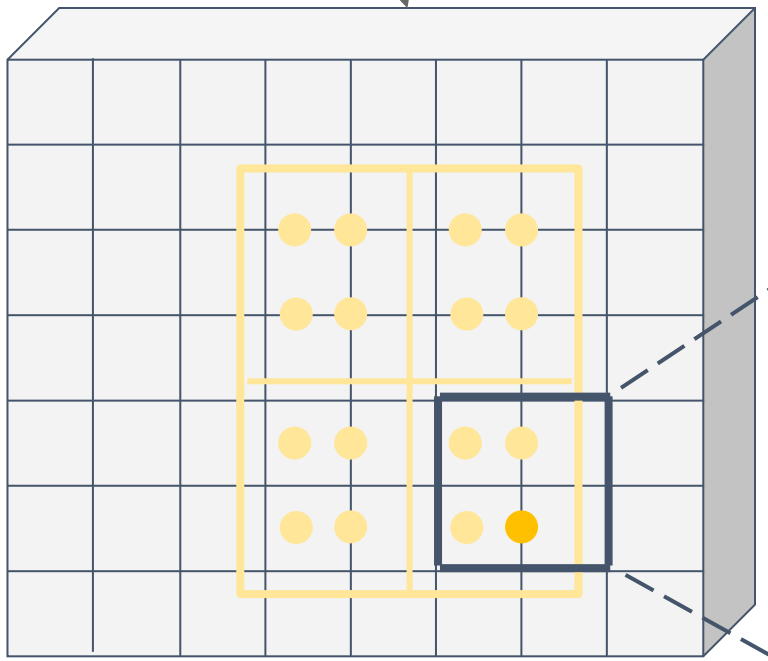
Feature f_{xy} for point (x, y) is a linear combination of features at its four neighboring grid cells:

Cropping Features: RoI Align



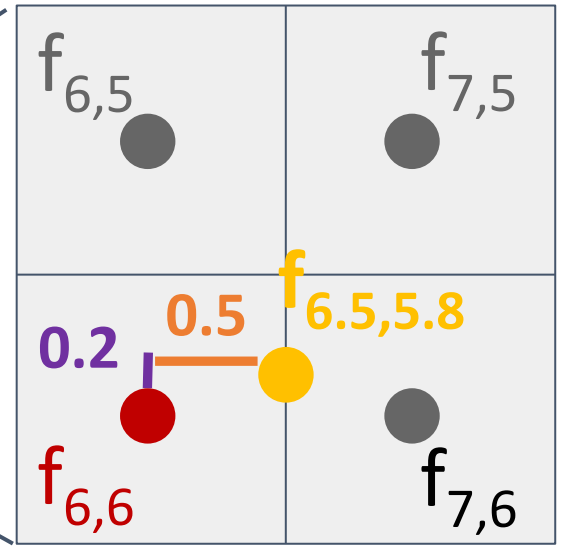
Project proposal onto features

CNN



No “snapping”!

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

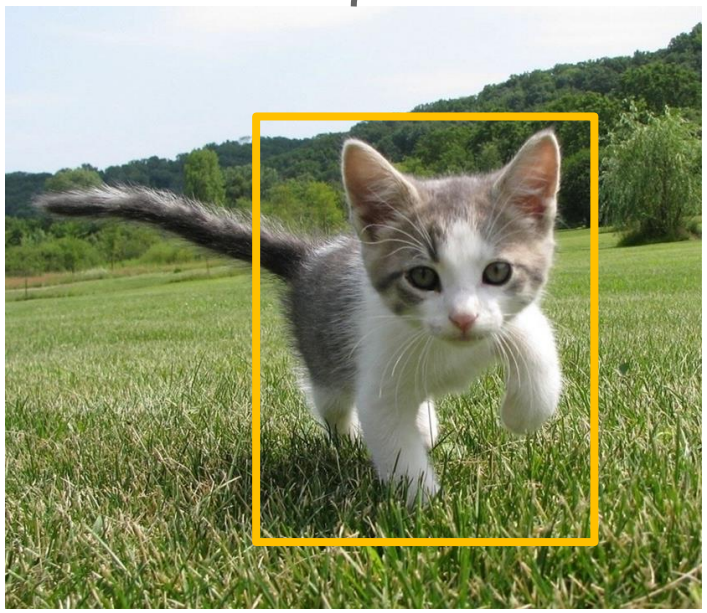


$$f_{xy} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - x_i|) \max(0, 1 - |y - y_i|)$$

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

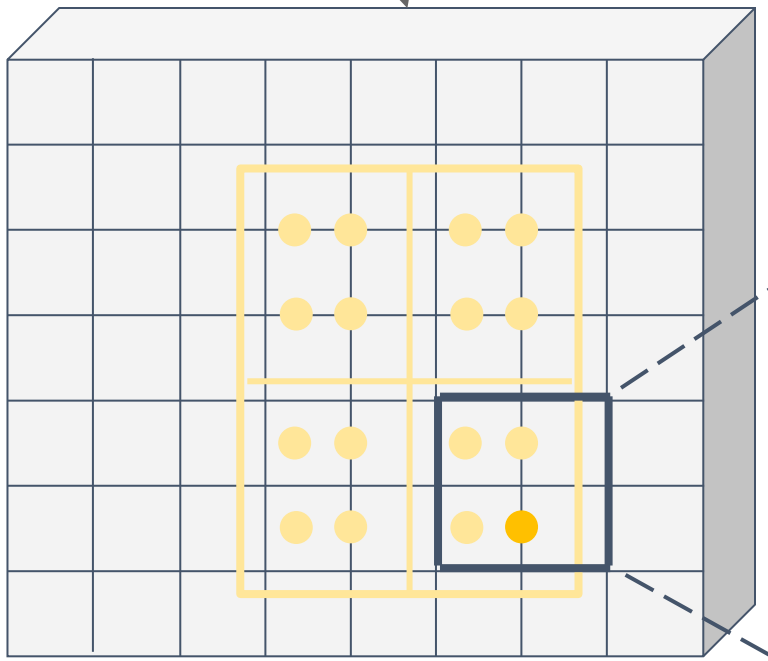
Feature f_{xy} for point (x, y) is a linear combination of features at its four neighboring grid cells:

Cropping Features: RoI Align



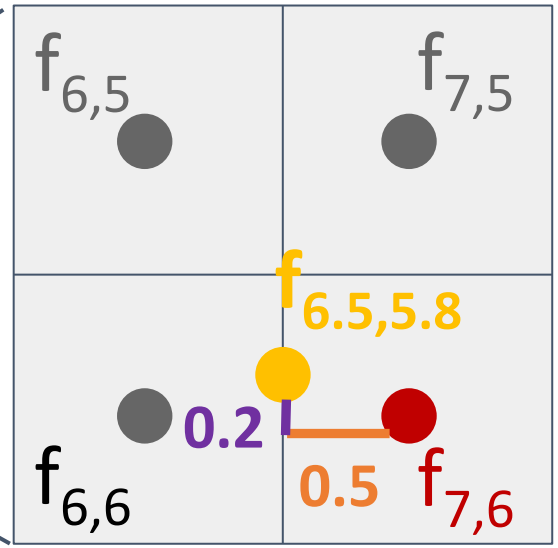
Project proposal onto features

CNN



No “snapping”!

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

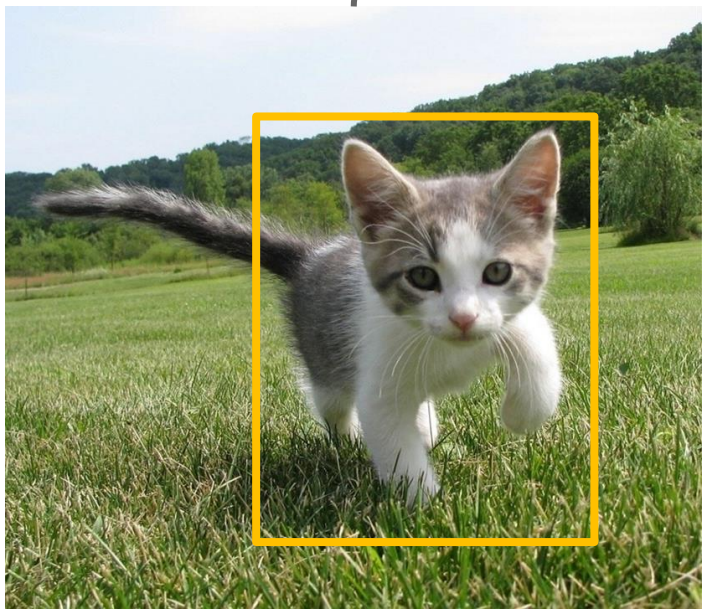


$$f_{xy} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - x_i|) \max(0, 1 - |y - y_i|)$$

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

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

Cropping Features: RoI Align



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

Project proposal onto features

No “snapping”!

CNN

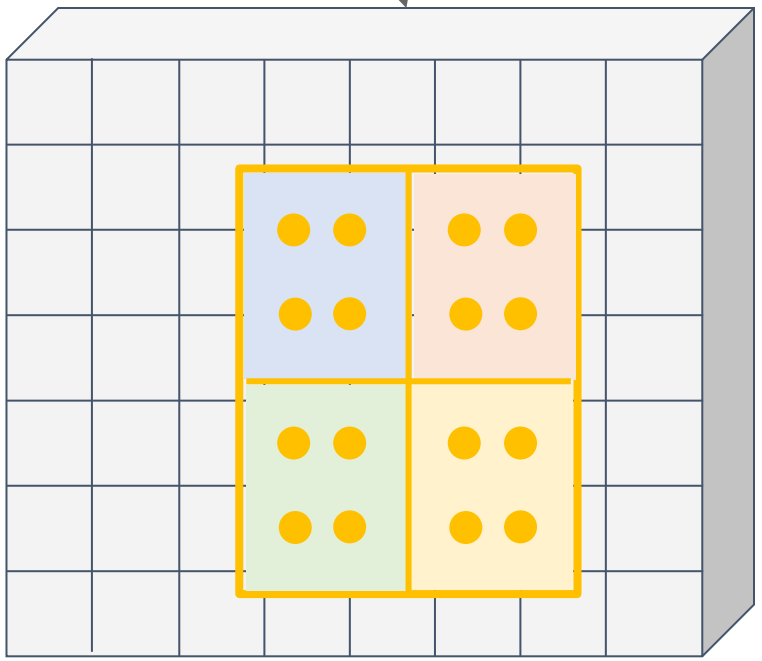
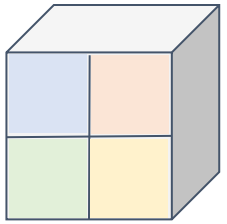


Image features
(e.g. 512 x 20 x 15)

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

After sampling, max-pool in each subregion

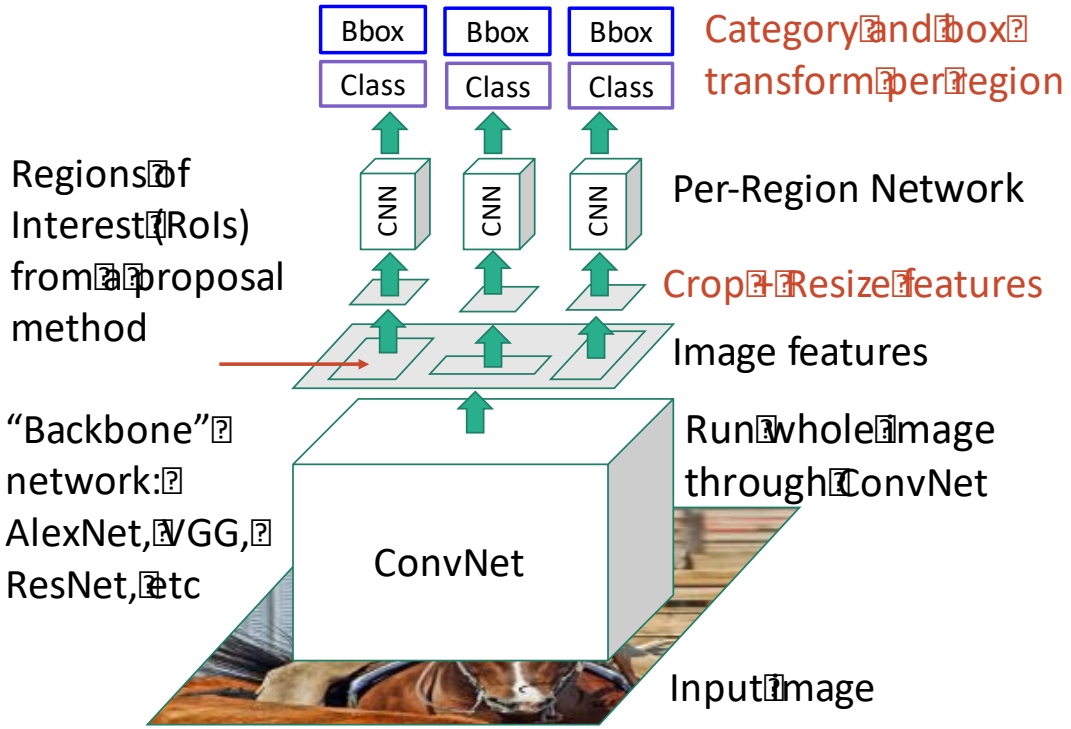


Region features
(here 512 x 2 x 2;
In practice e.g 512 x 7 x 7)

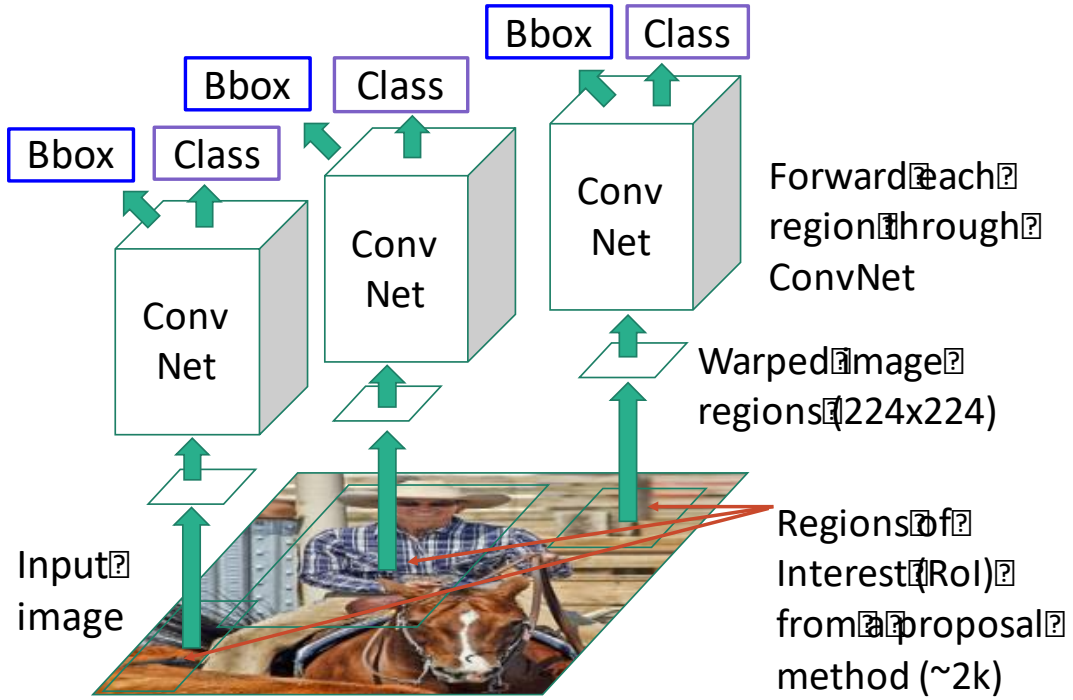
He et al, “Mask R-CNN”, ICCV 2017

Fast R-CNN vs. “Slow” R-CNN

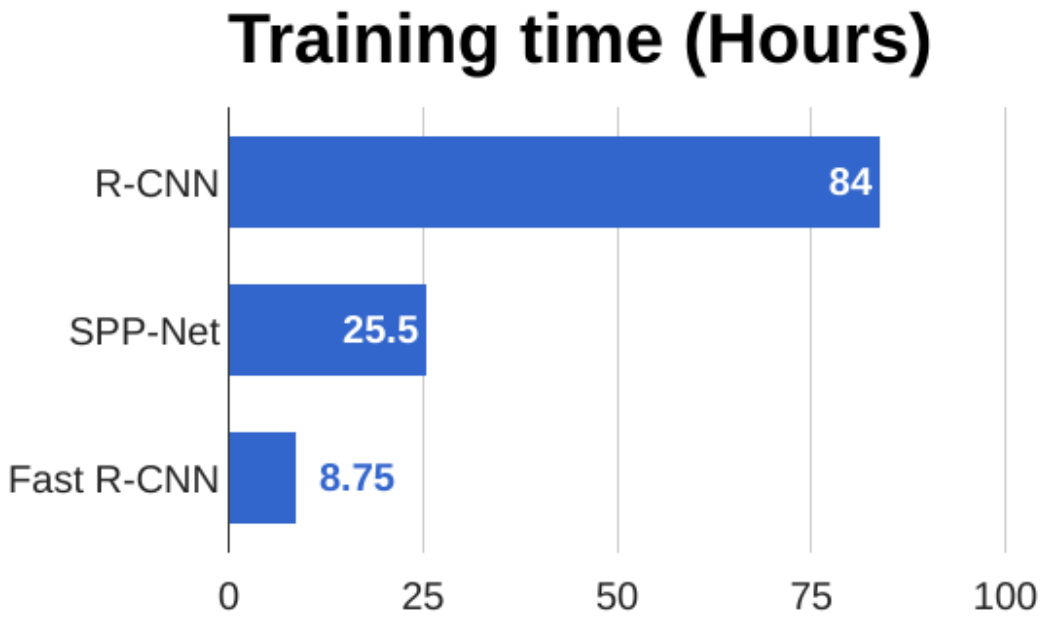
Fast R-CNN: Apply differentiable cropping to shared image features

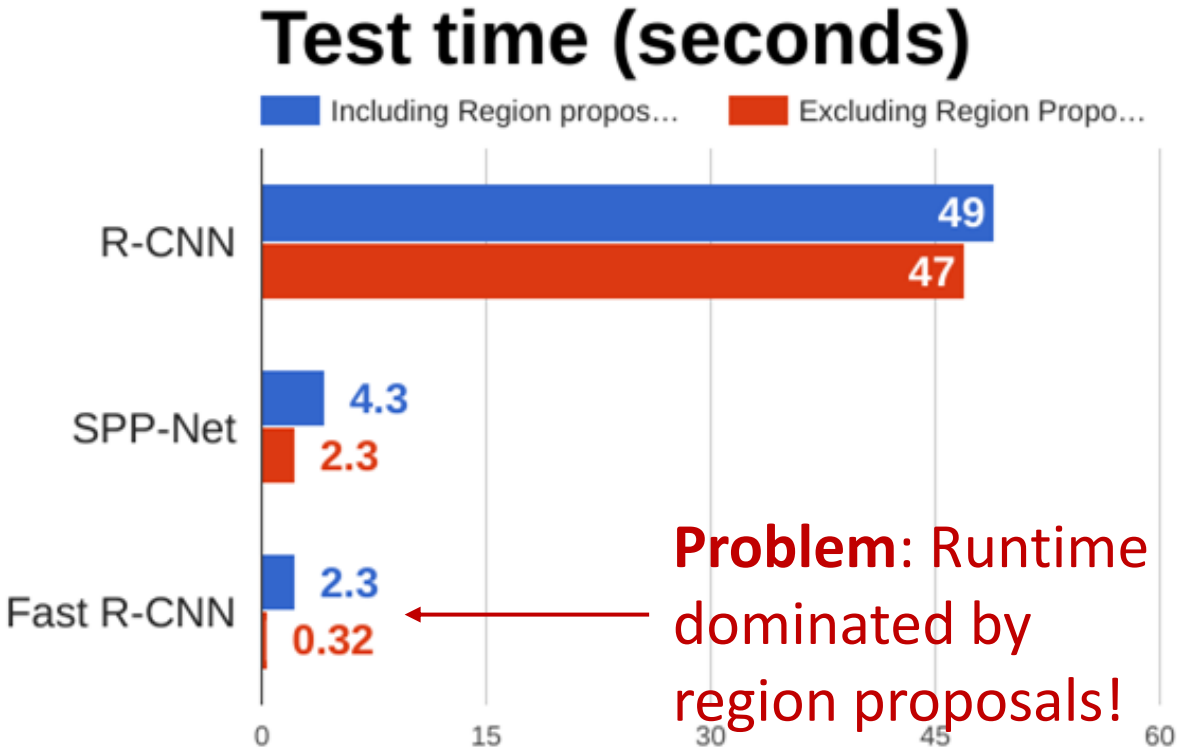


“Slow” R-CNN: Apply differentiable cropping to shared image features



Fast R-CNN vs. “Slow” R-CNN





Problem: Runtime dominated by region proposals!

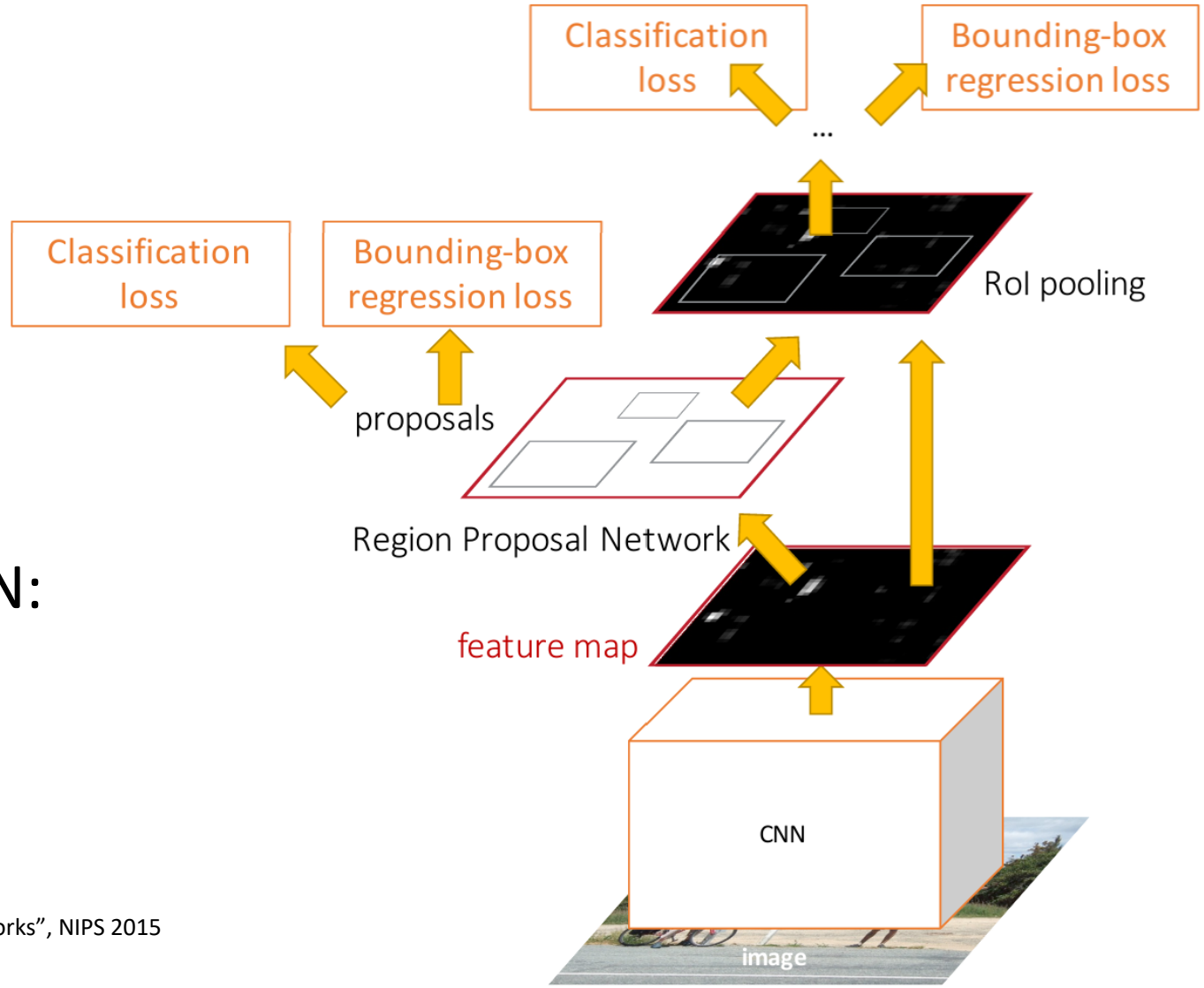
Recall: Region proposals computed by heuristic “Selective Search” algorithm on CPU -- let’s learn them with a CNN instead!

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
He et al, “Spatial pyramid pooling in deep convolutional networks for visual recognition”, ECCV 2014
Girshick, “Fast R-CNN”, ICCV 2015

Faster R-CNN: Learnable Region Proposals

Insert **Region Proposal Network (RPN)** to predict proposals from features

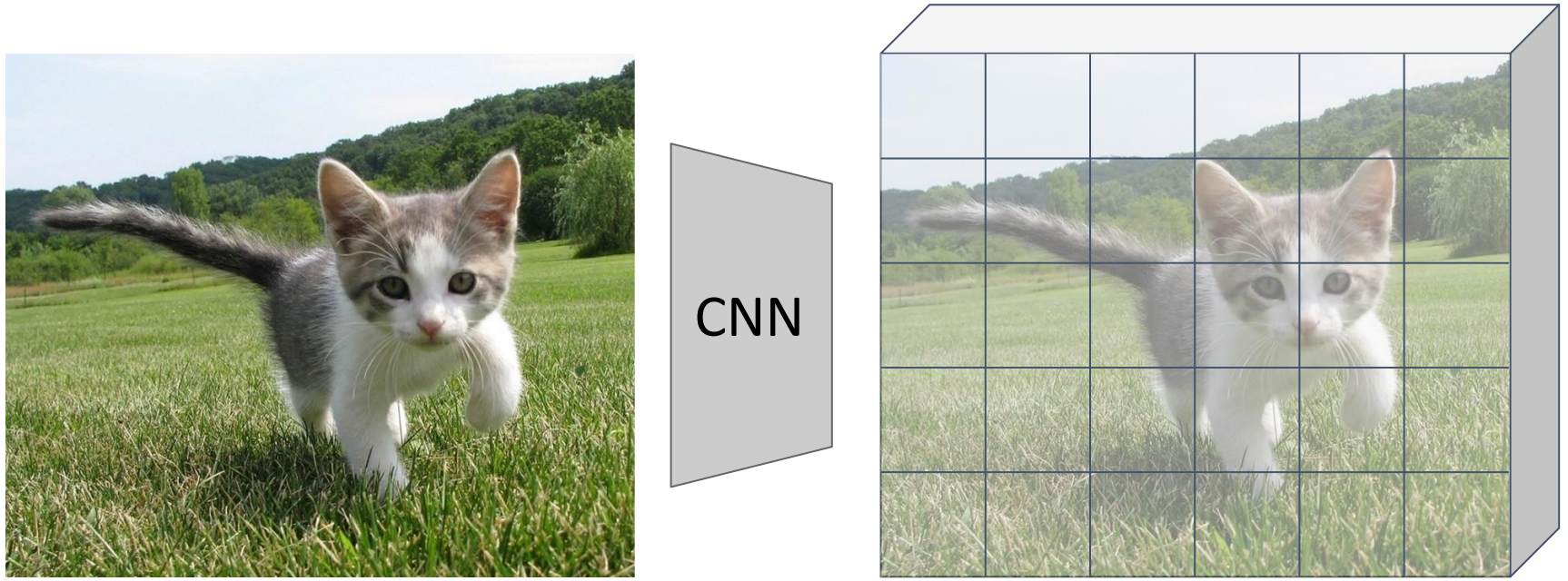
Otherwise same as Fast R-CNN: Crop features for each proposal, classify each one



Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015
Figure copyright 2015, Ross Girshick; reproduced with permission

Region Proposal Network (RPN)

Run backbone CNN to get
features aligned to input image



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

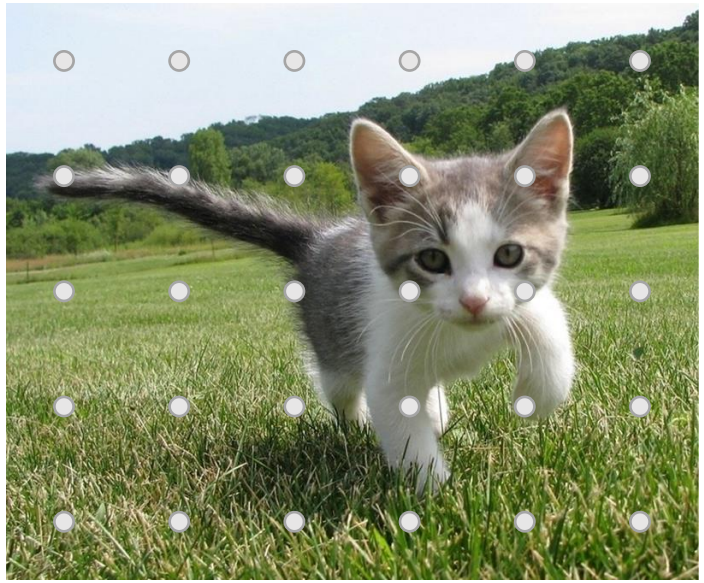
Image features
(e.g. 512 x 20 x 15)

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

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)

CNN

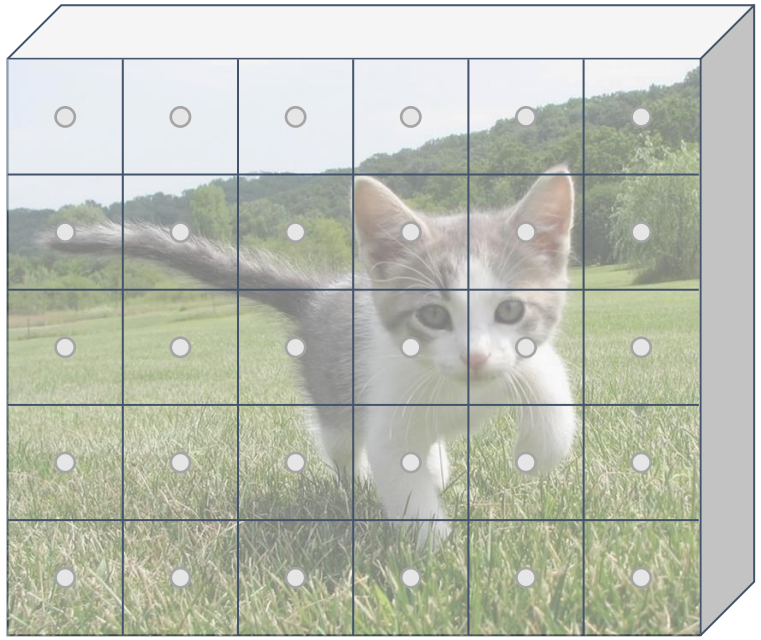


Image features
(e.g. 512 x 20 x 15)

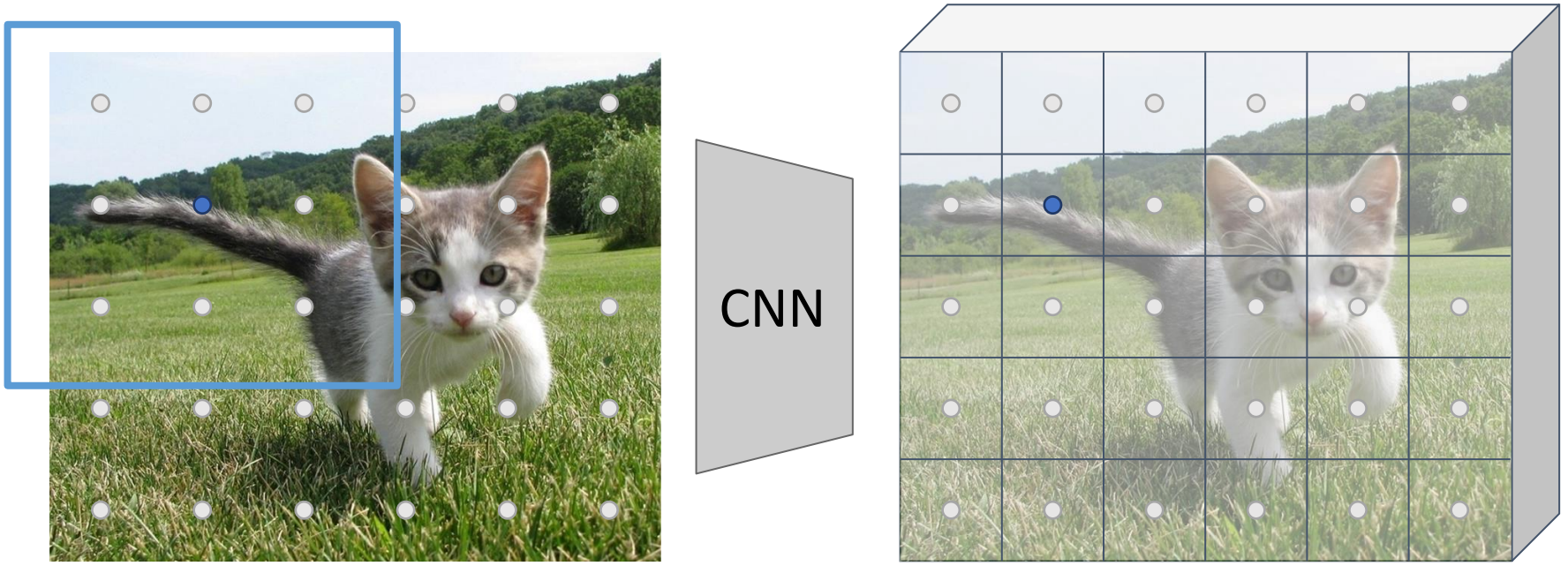
Ren et al, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, NIPS 2015

Region Proposal Network (RPN)

Imagine an **anchor box** of fixed size at each point in the feature map

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 20 x 15)

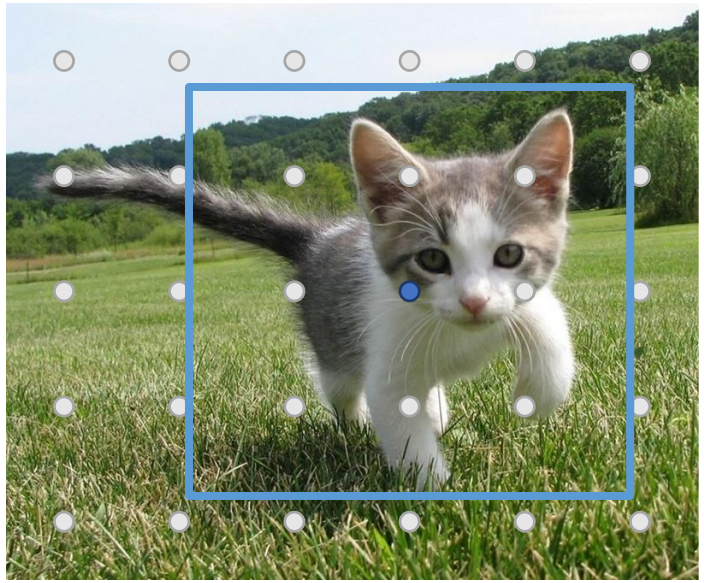
Ren et al, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, NIPS 2015

Region Proposal Network (RPN)

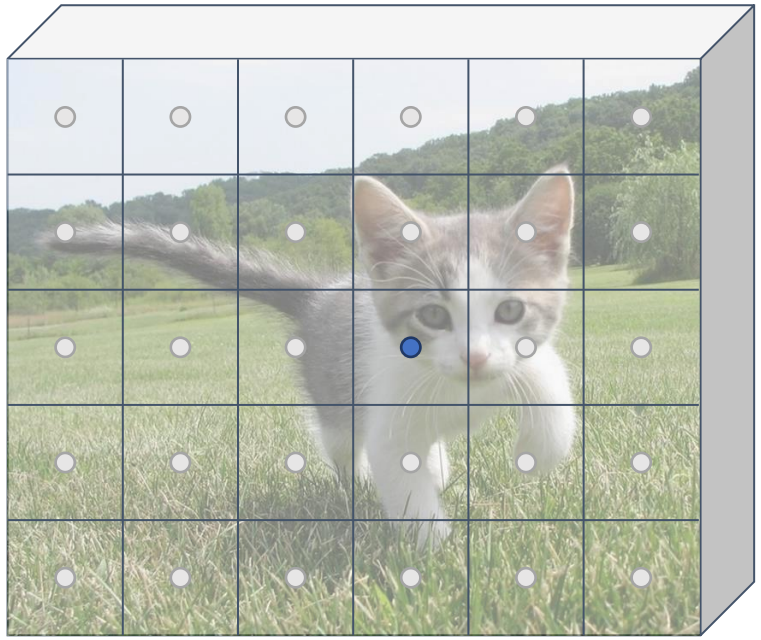
Imagine an **anchor box** of fixed size at each point in the feature map

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input



CNN



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

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

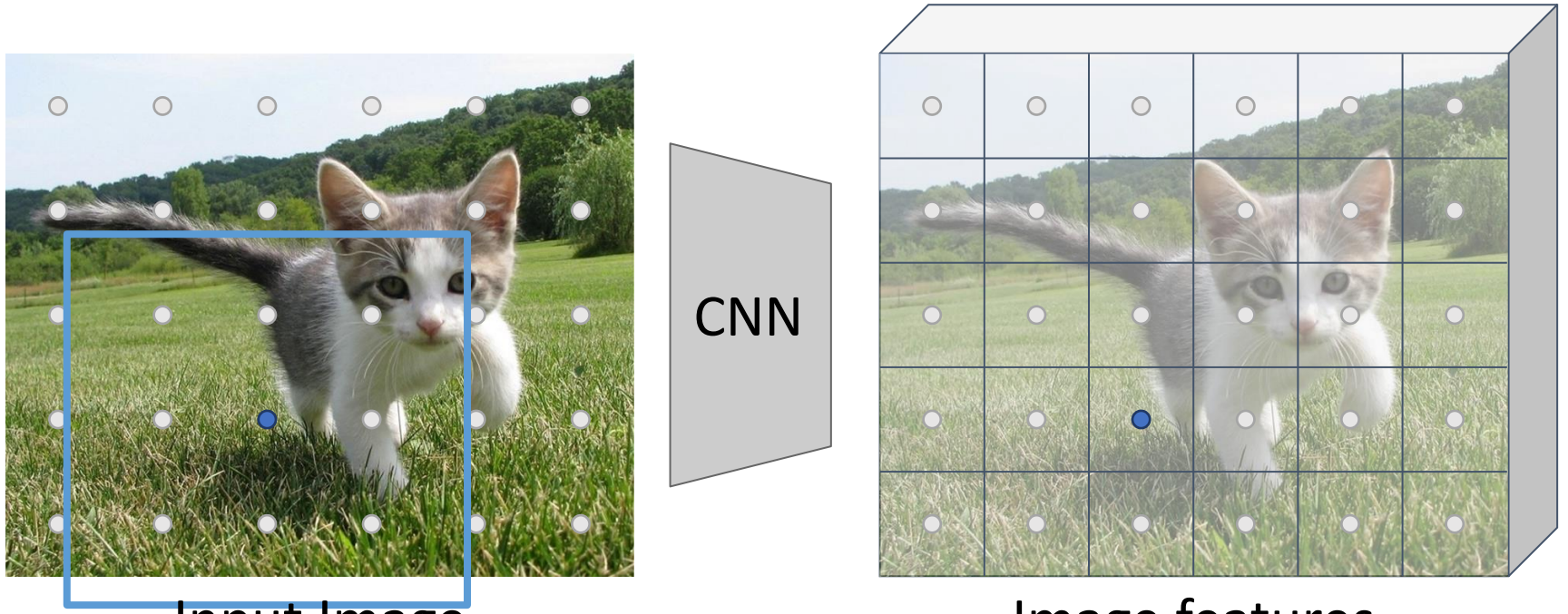
Ren et al, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, NIPS 2015

Region Proposal Network (RPN)

Imagine an **anchor box** of fixed size at each point in the feature map

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 20 x 15)

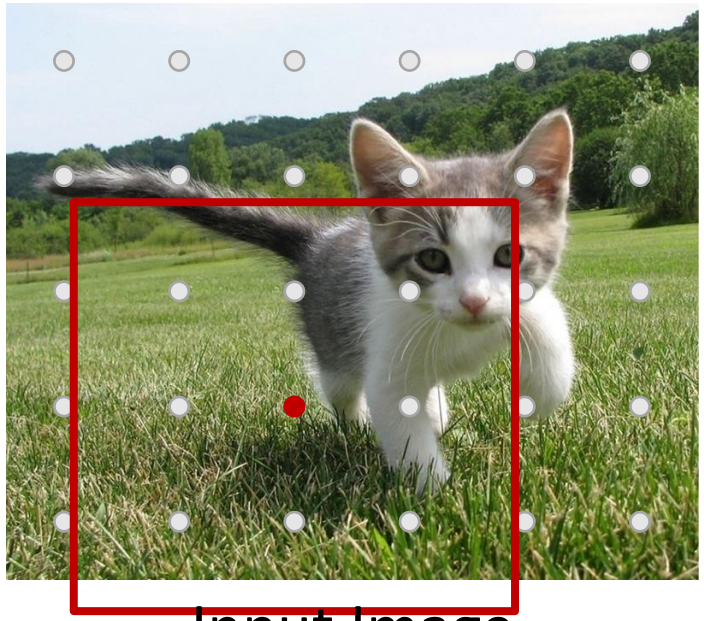
Ren et al, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, NIPS 2015

Region Proposal Network (RPN)

Imagine an **anchor box** of fixed size at each point in the feature map

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)

CNN

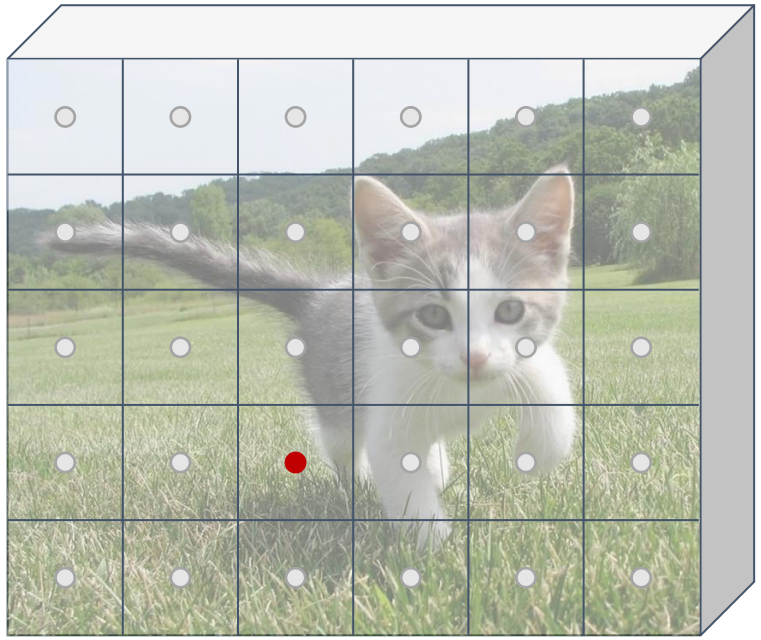


Image features
(e.g. 512 x 20 x 15)

Classify each anchor as positive (object) or negative (no object)

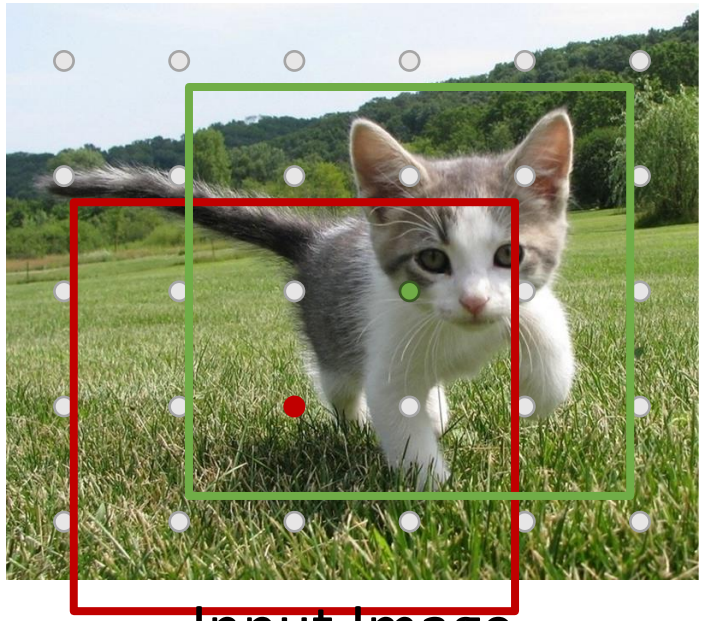
Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015

Region Proposal Network (RPN)

Imagine an **anchor box** of fixed size at each point in the feature map

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)

CNN

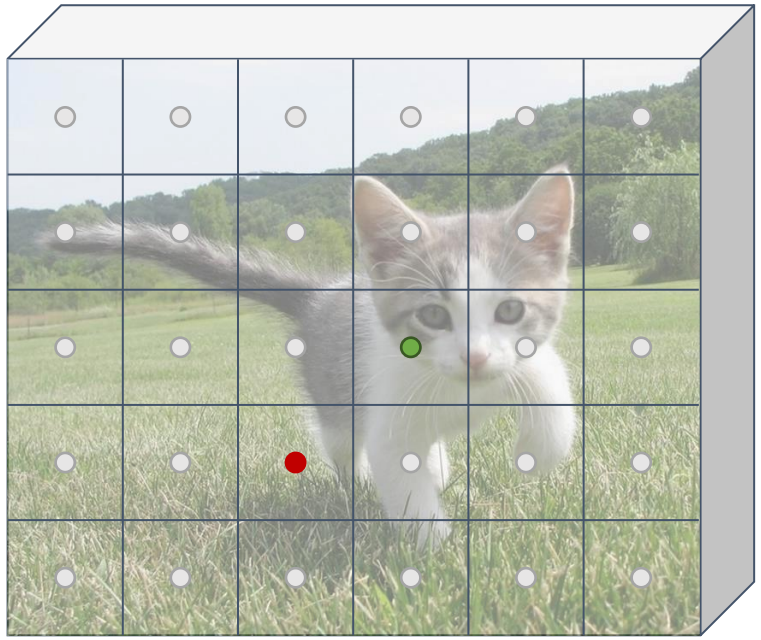


Image features
(e.g. 512 x 20 x 15)

Classify each anchor as positive (object) or negative (no object)

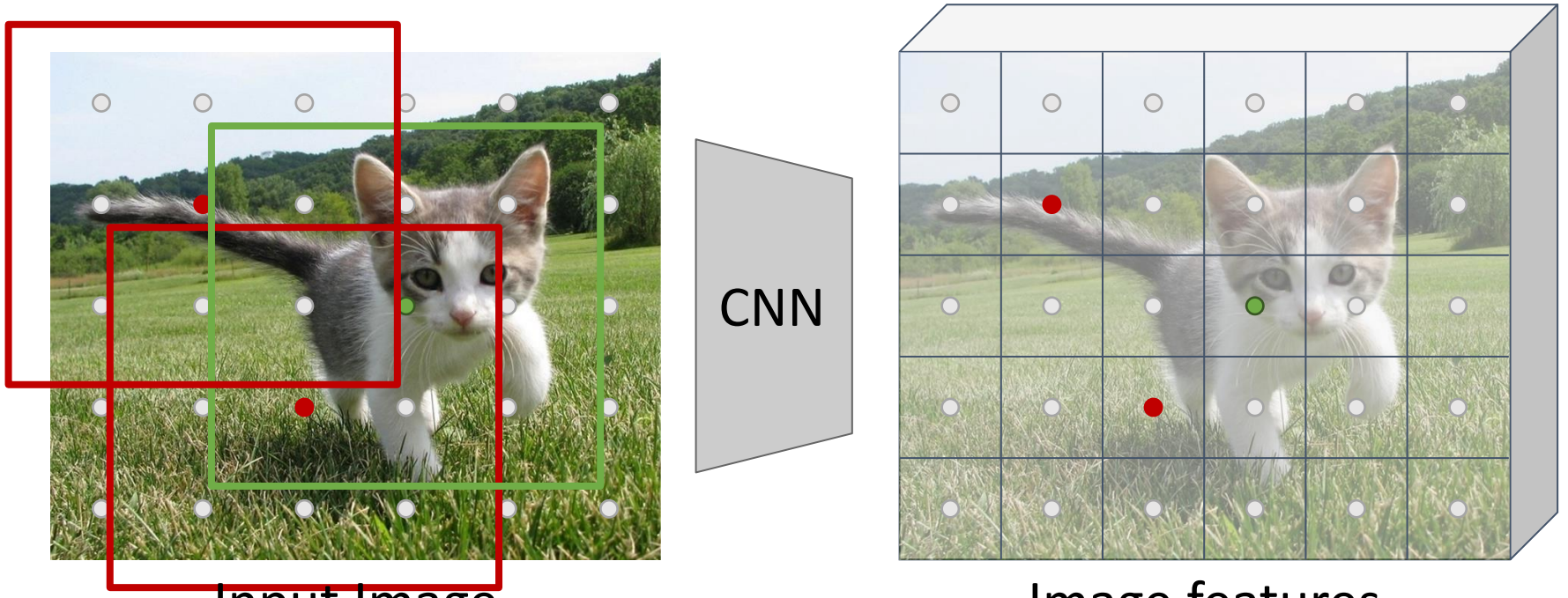
Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015

Region Proposal Network (RPN)

Imagine an **anchor box** of fixed size at each point in the feature map

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)

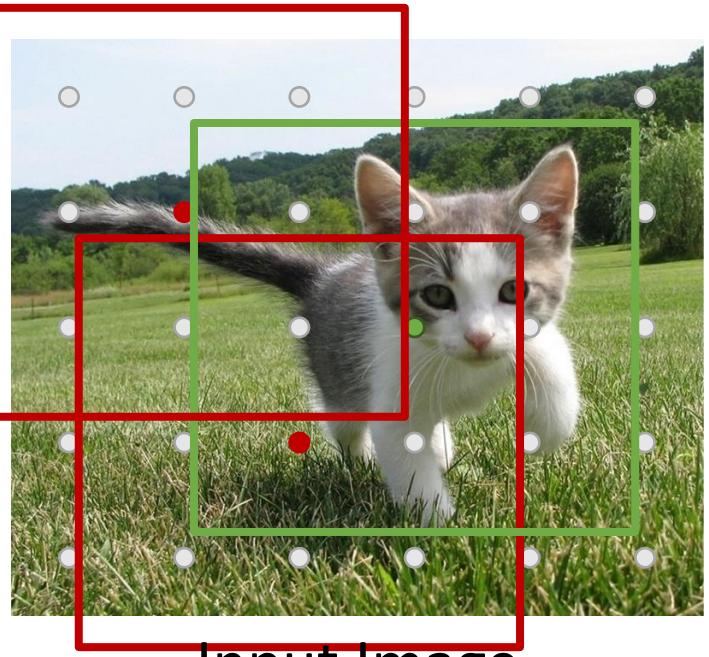
Image features
(e.g. 512 x 20 x 15)

Classify each anchor as positive (object) or negative (no object)

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

Region Proposal Network (RPN)

Run backbone CNN to get features aligned to input image



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

CNN

Each feature corresponds to a point in the input

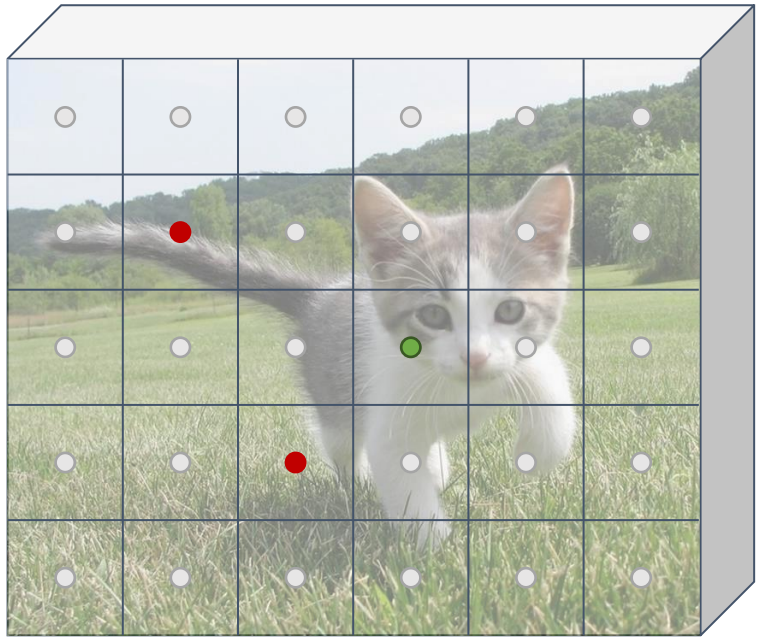


Image features
(e.g. 512 x 20 x 15)

Conv

Predict object vs. not object scores for all anchors with a conv layer (512 input filters, 2 output filters)

Anchor is object?
2 x 20 x 15

Classify each anchor as positive (object) or negative (no object)

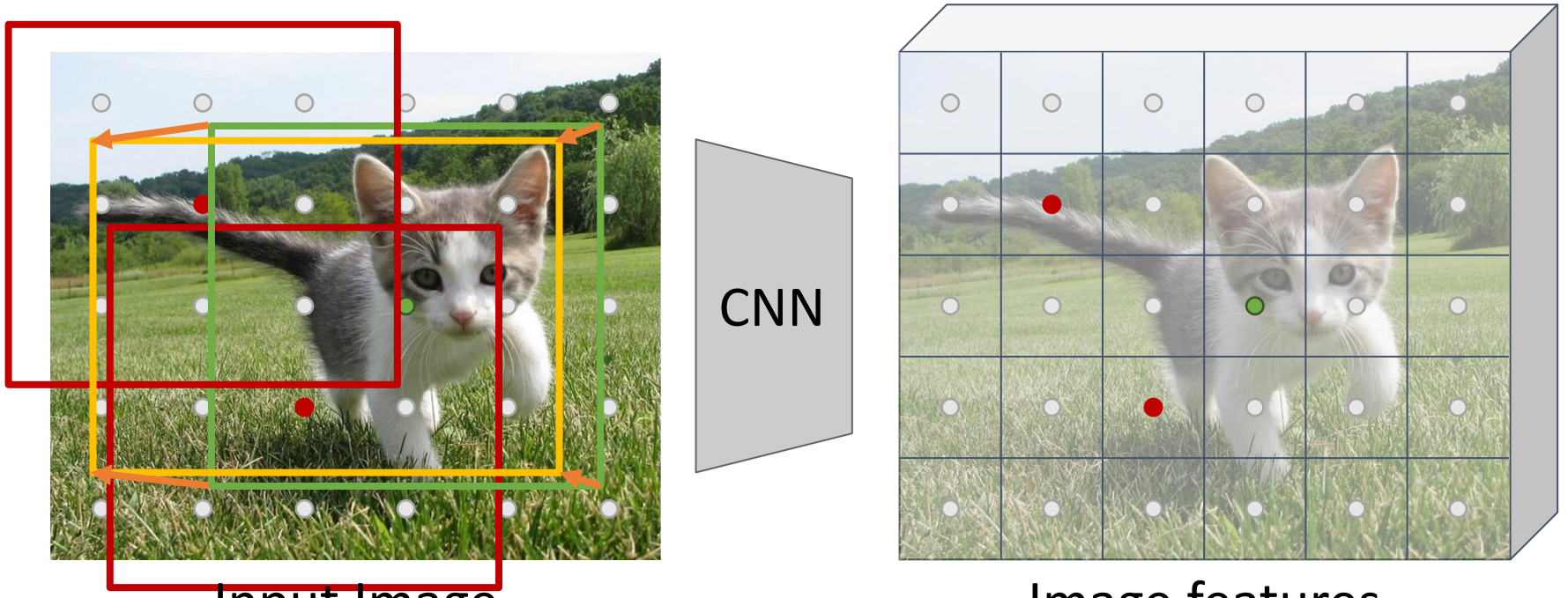
Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015

Region Proposal Network (RPN)

For positive anchors, also predict a transform that converting the anchor to the GT box (like R-CNN)

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)

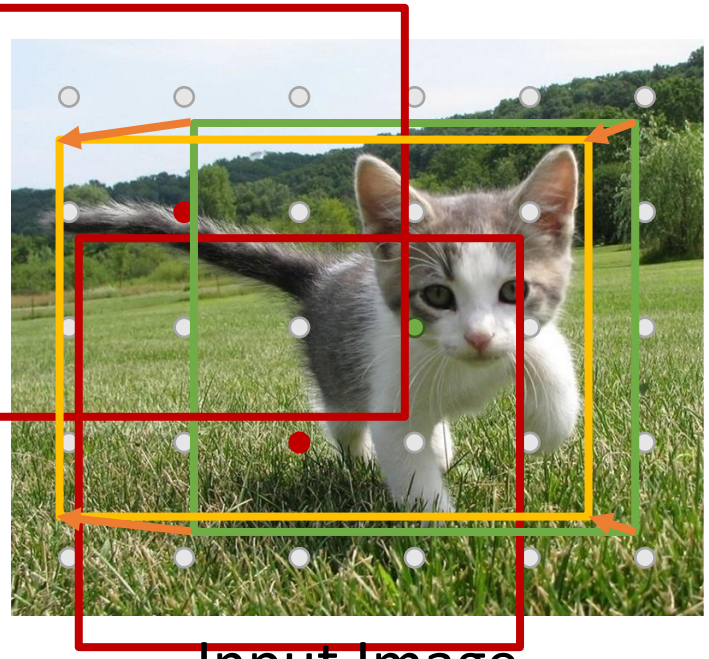
Image features
(e.g. 512 x 20 x 15)

Classify each anchor as positive (object) or negative (no object)

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

Region Proposal Network (RPN)

Run backbone CNN to get features aligned to input image



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

CNN

Each feature corresponds to a point in the input

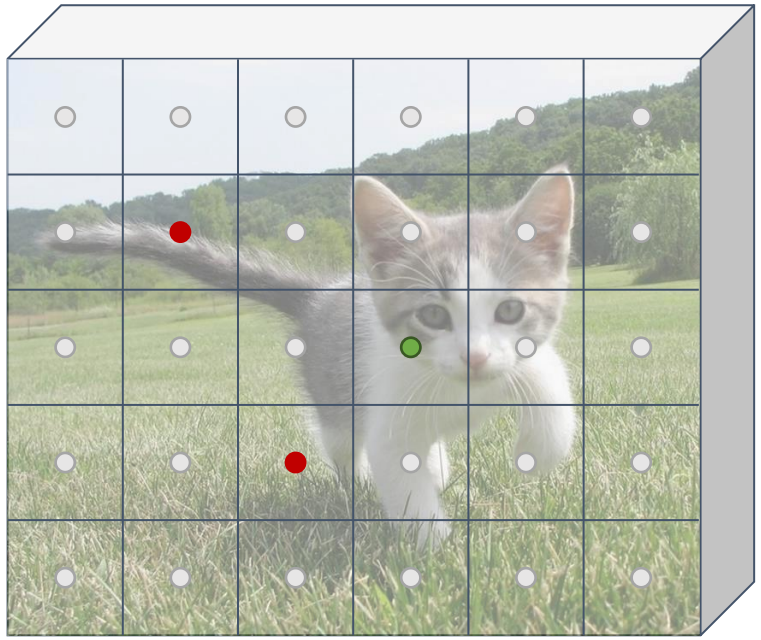


Image features
(e.g. 512 x 20 x 15)

Conv

Anchor is object?
2 x 20 x 15

Anchor transforms
4 x 20 x 15

For positive anchors, also predict a transform that converting the anchor to the GT box (like R-CNN)
Predict transforms with conv

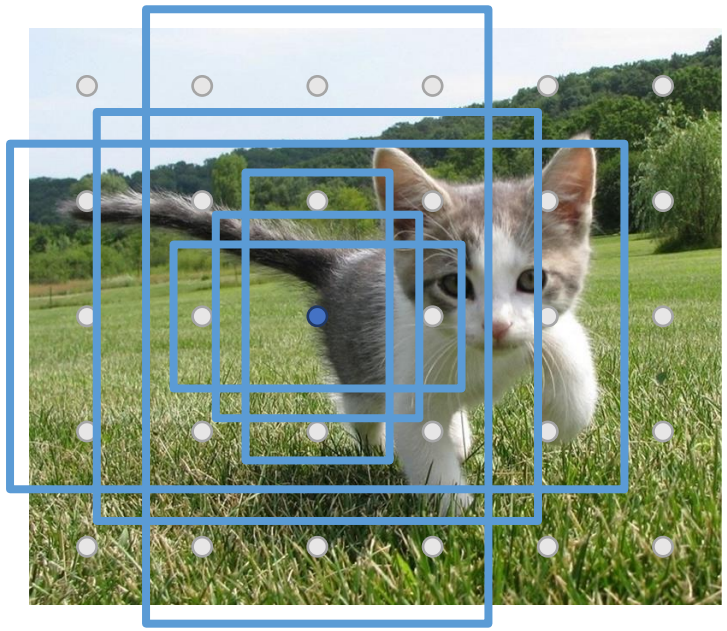
Classify each anchor as positive (object) or negative (no object)

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

Region Proposal Network (RPN)

In practice: Rather than using one anchor per point, instead consider K different anchors with different size and scale (here K = 6)

Run backbone CNN to get features aligned to input image



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

CNN

Each feature corresponds to a point in the input

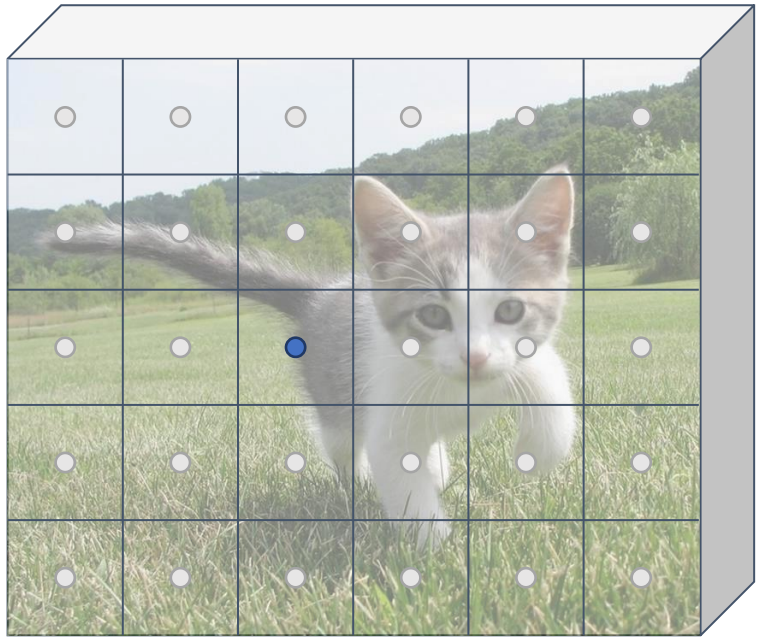


Image features
(e.g. 512 x 20 x 15)

Conv

Anchor is object?
2K x 20 x 15

Anchor transforms
4K x 20 x 15

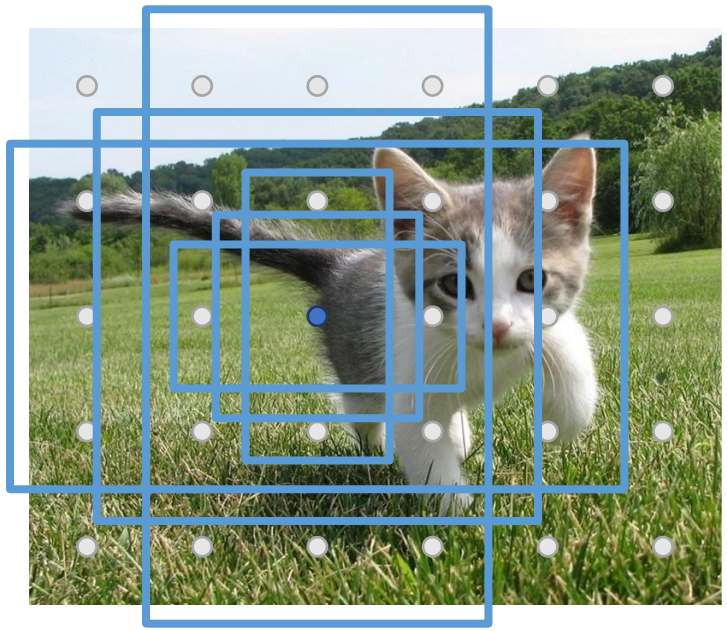
During training, supervised positive / negative anchors and box transforms like R-CNN

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

Region Proposal Network (RPN)

In practice: Rather than using one anchor per point, instead consider K different anchors with different size and scale (here K = 6)

Run backbone CNN to get features aligned to input image



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

CNN

Each feature corresponds to a point in the input

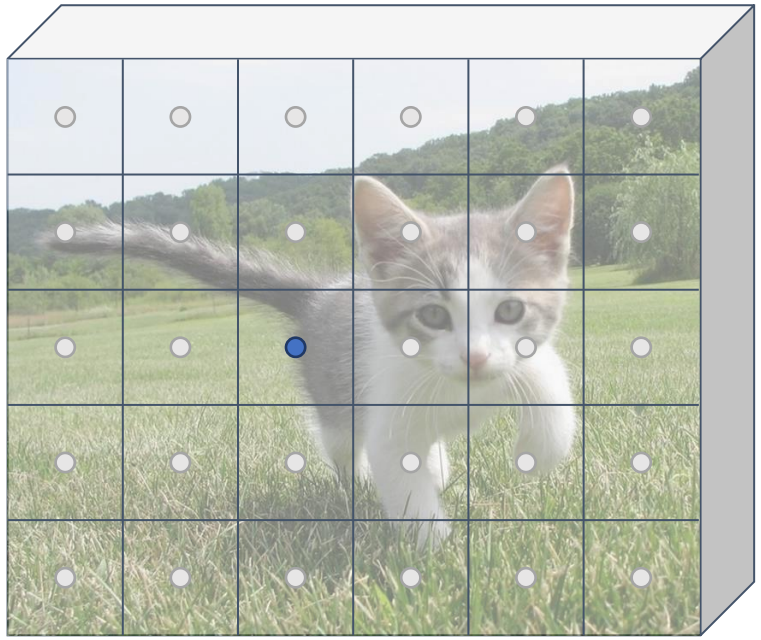


Image features
(e.g. 512 x 20 x 15)

Conv

Anchor is object?
2K x 20 x 15

Anchor transforms
4K x 20 x 15

Positive anchors: >= 0.7 IoU with some GT box (plus highest IoU to each GT)

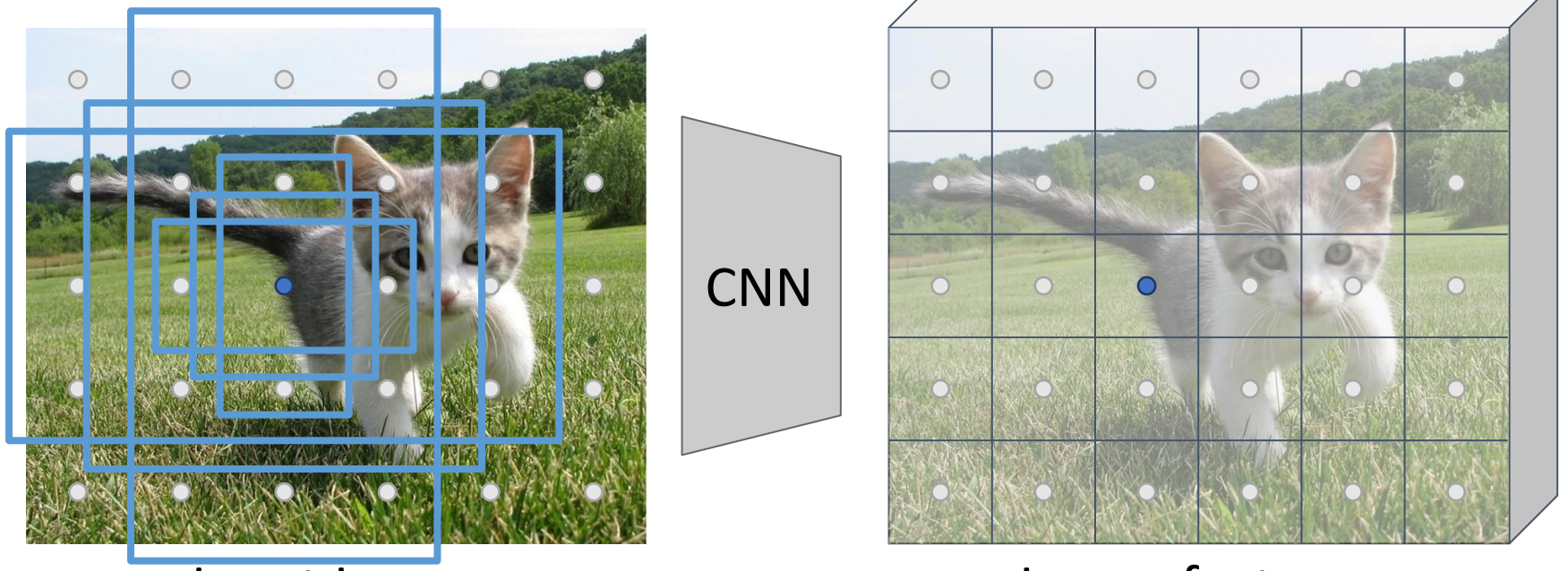
Ren et al, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, NIPS 2015

Region Proposal Network (RPN)

In practice: Rather than using one anchor per point, instead consider K different anchors with different size and scale (here K = 6)

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 20 x 15)

Negative anchors: < 0.3 IoU with all GT boxes. Don't supervised transforms for negative boxes.

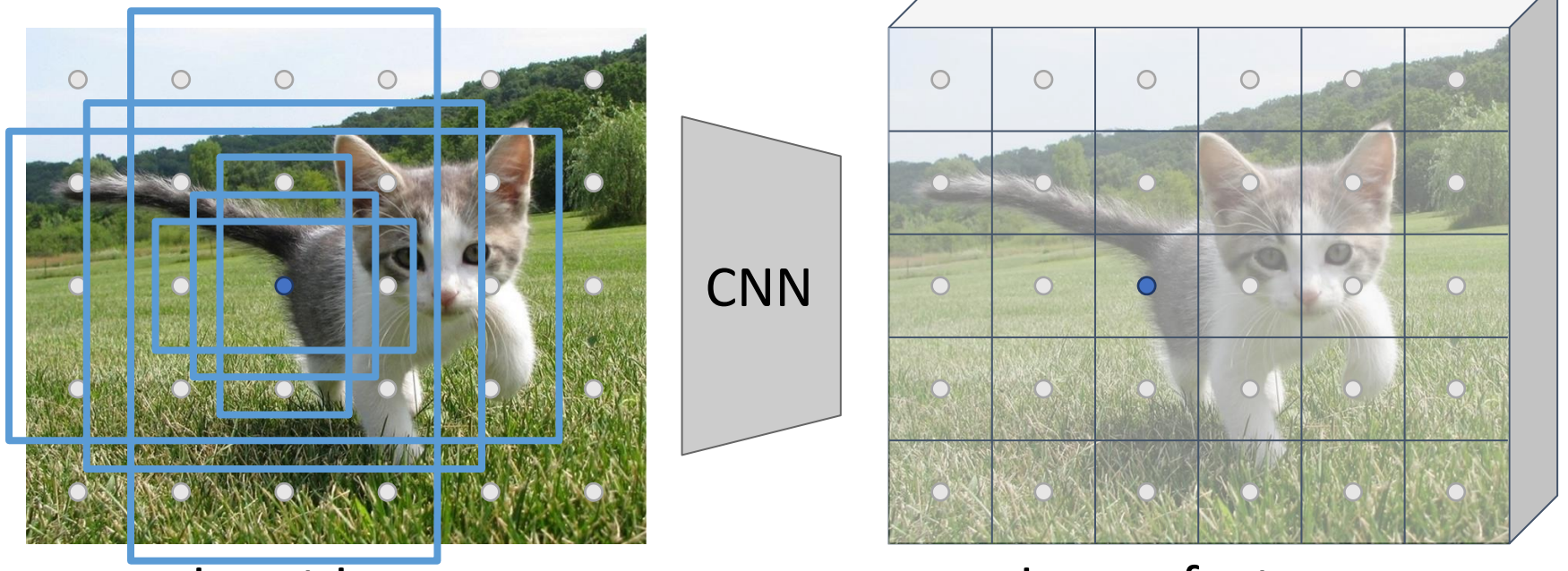
Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015

Region Proposal Network (RPN)

In practice: Rather than using one anchor per point, instead consider K different anchors with different size and scale (here K = 6)

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 20 x 15)

Neutral anchors: between 0.3 and 0.7 IoU with all GT boxes; ignored during training

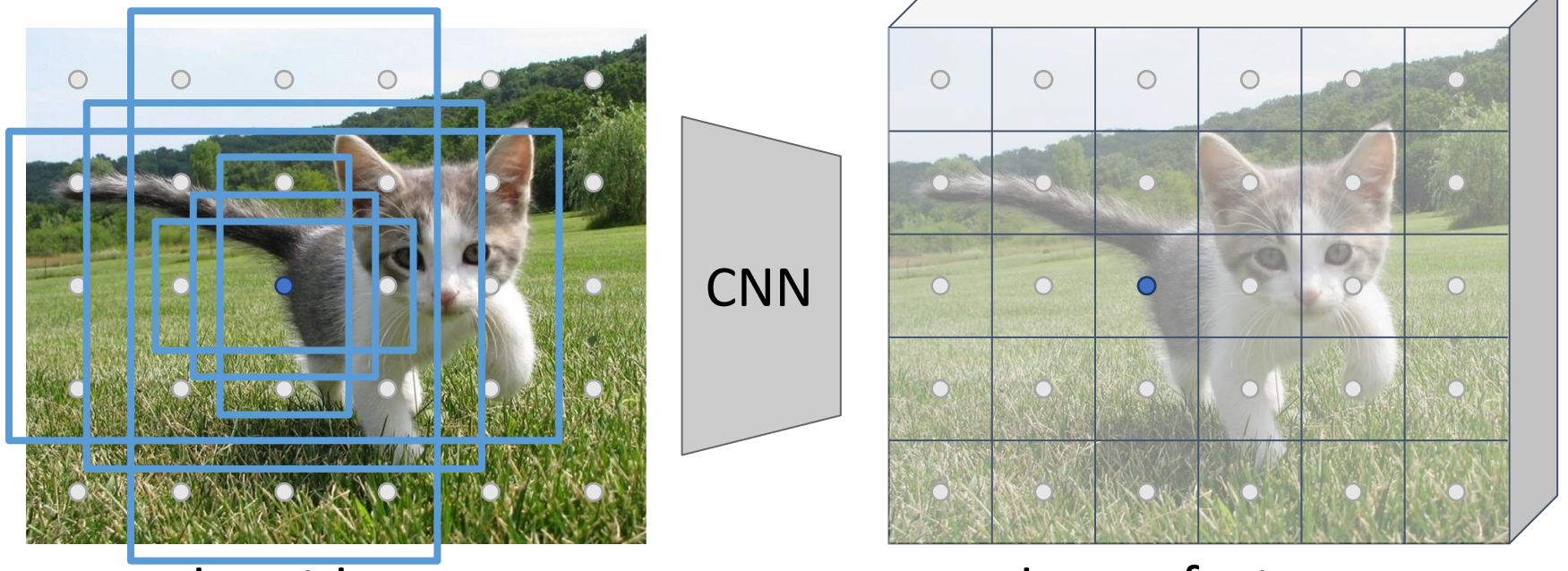
Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015

Region Proposal Network (RPN)

In practice: Rather than using one anchor per point, instead consider K different anchors with different size and scale (here K = 6)

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 20 x 15)

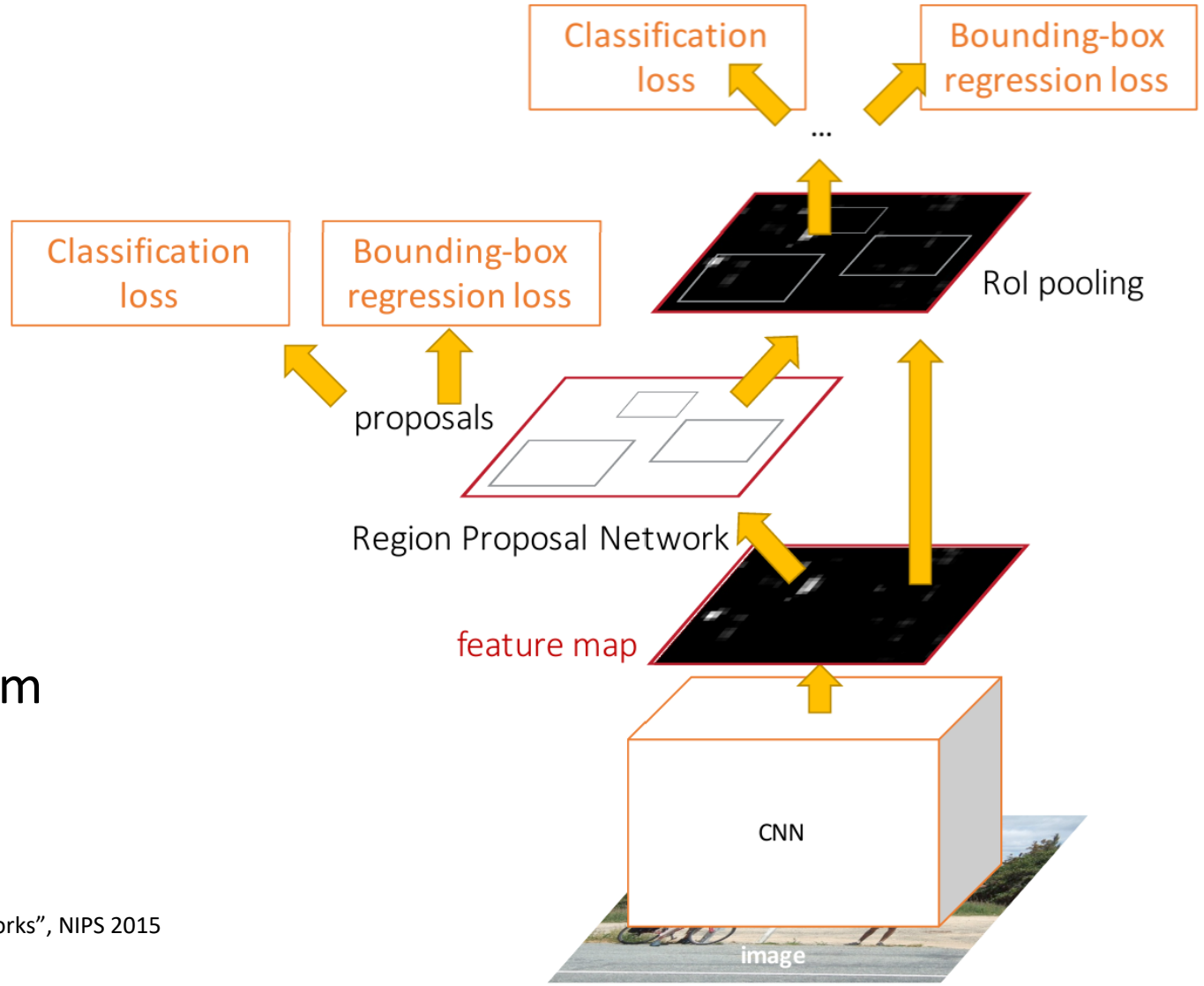
At test-time, sort all K*20*15 boxes by their positive score, take top 300 as our region proposals

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

Faster R-CNN: Learnable Region Proposals

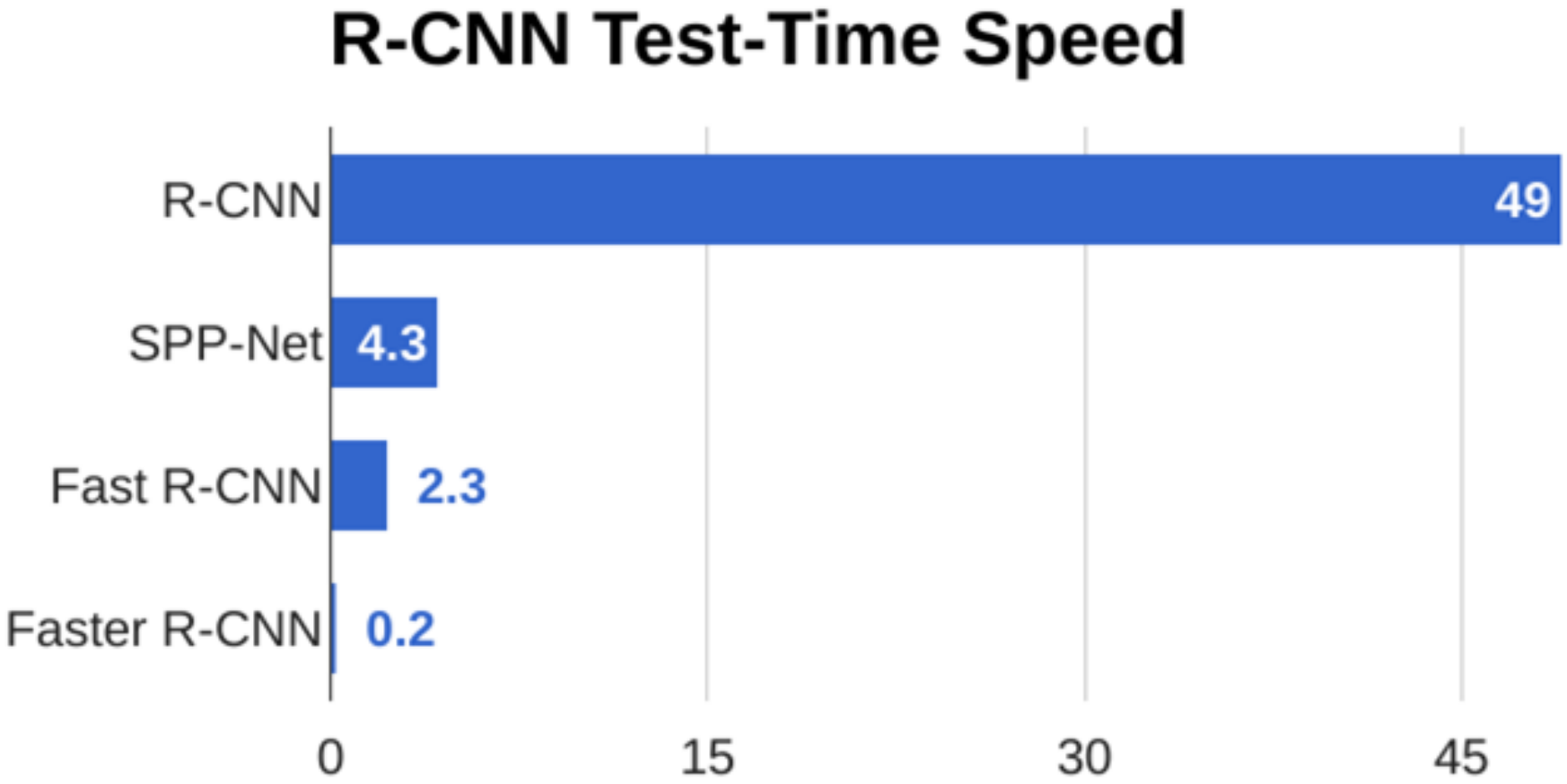
Jointly train with 4 losses:

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



Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015
Figure copyright 2015, Ross Girshick; reproduced with permission

Faster R-CNN: Learnable Region Proposals



Object Detection Progress

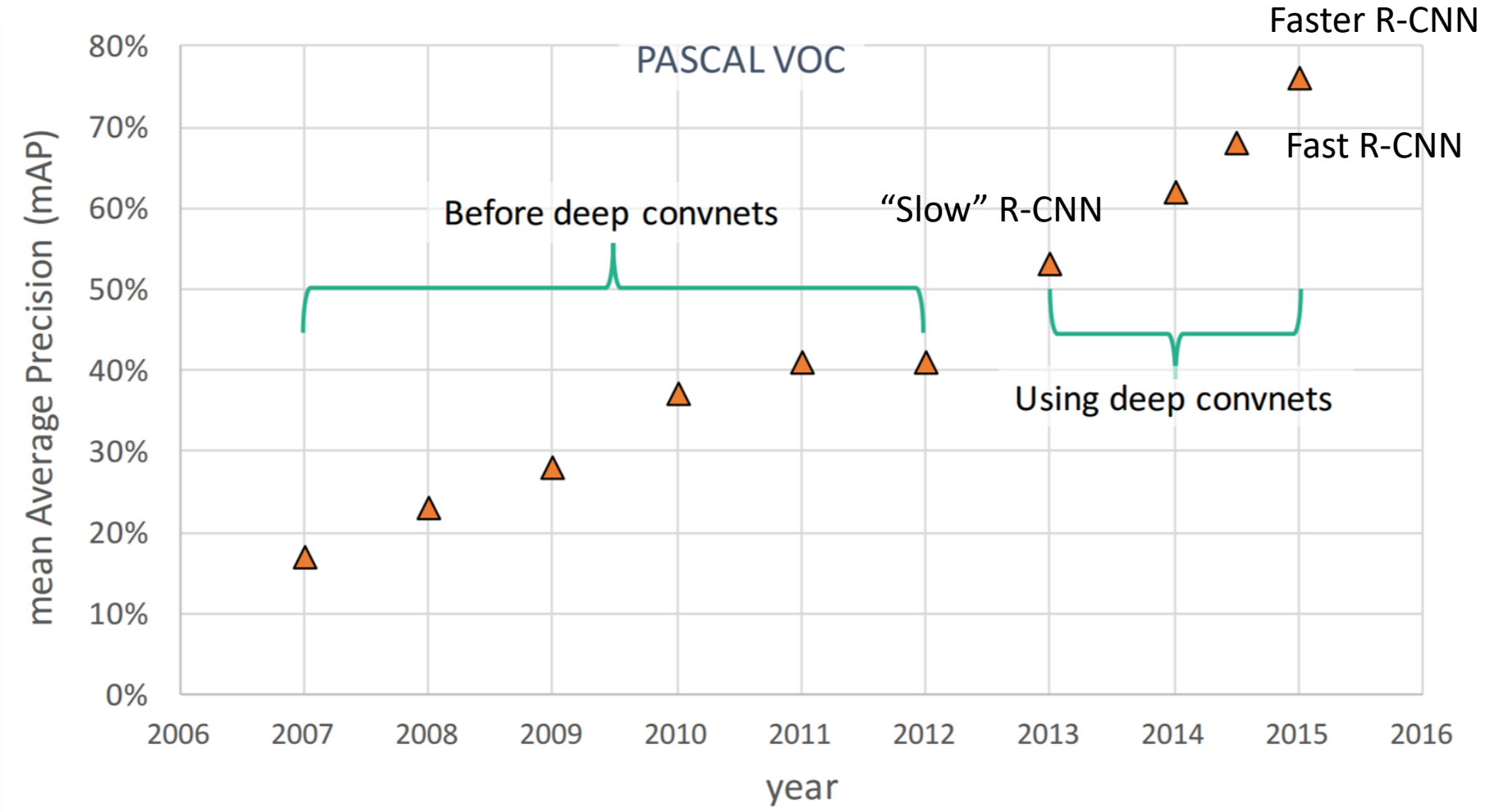


Figure copyright Ross Girshick, 2015.
Reproduced with permission.

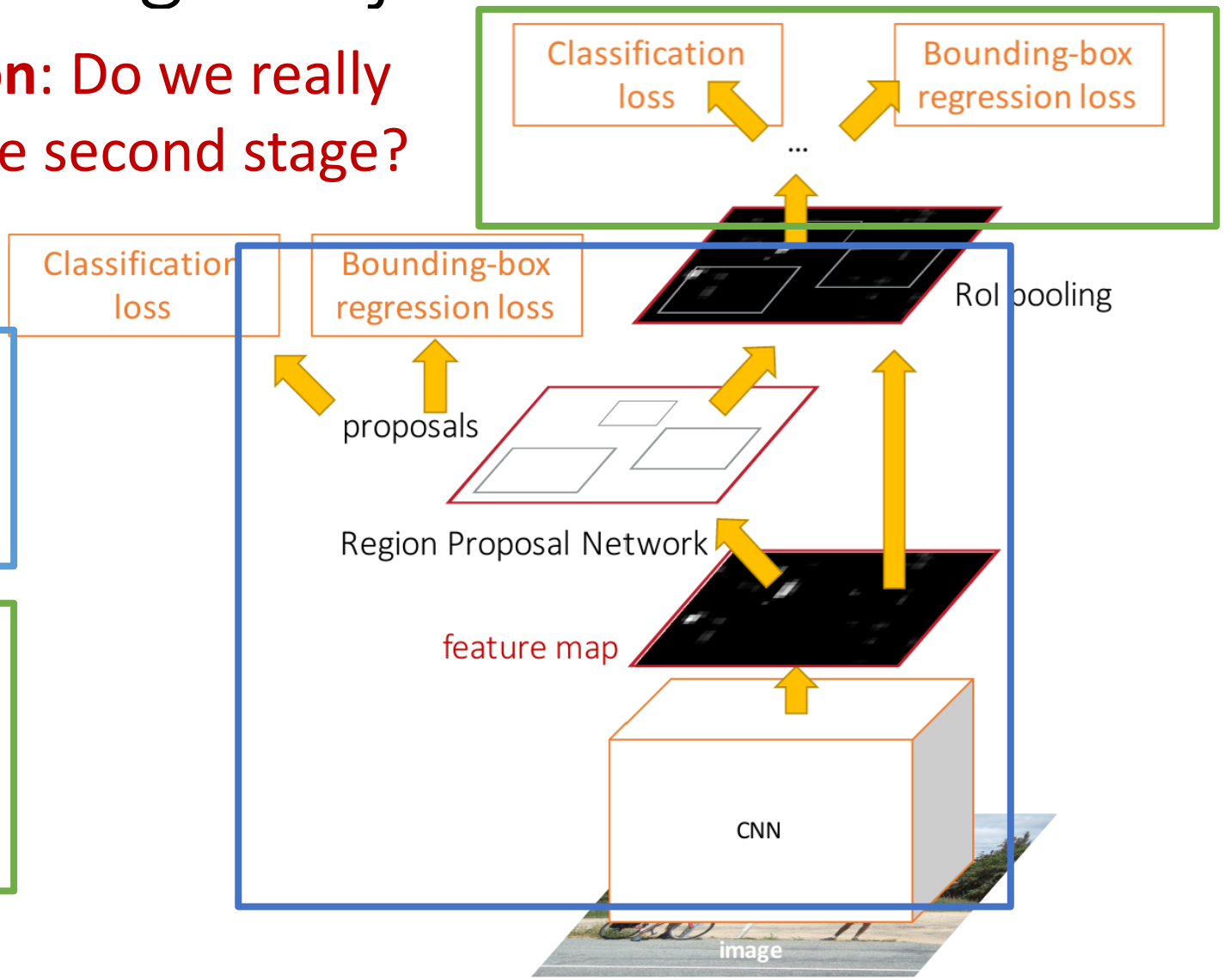
Faster R-CNN: Two-Stage Object Detector

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

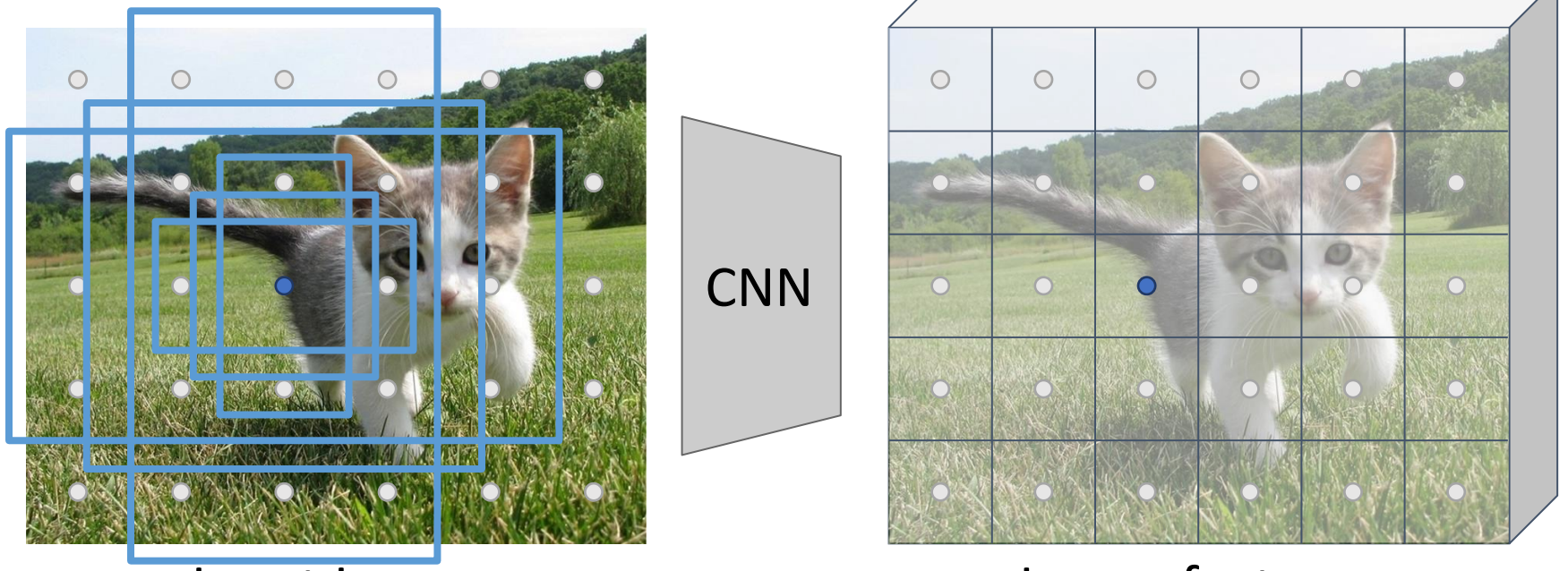


Single-Stage Detectors: RetinaNet

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 20 x 15)

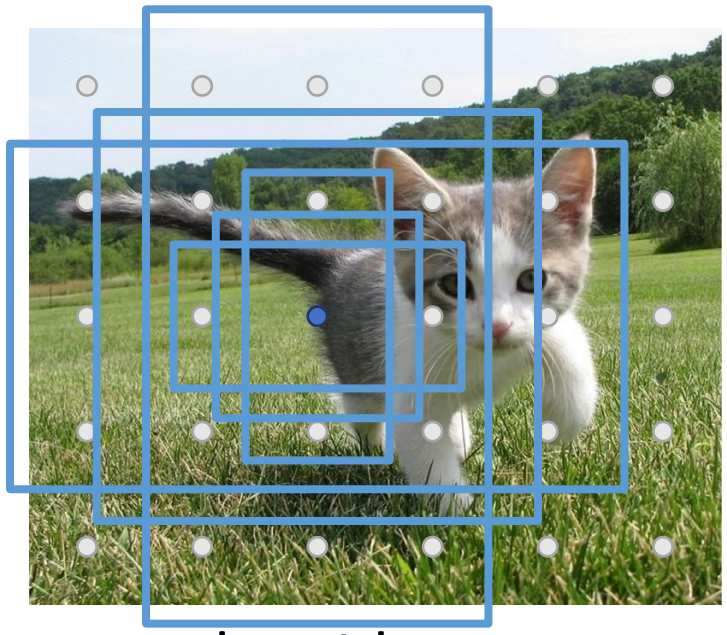
Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

Single-Stage Detectors: RetinaNet

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

Solution: "Focal Loss"; see the paper for details

Run backbone CNN to get features aligned to input image



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

CNN

Each feature corresponds to a point in the input

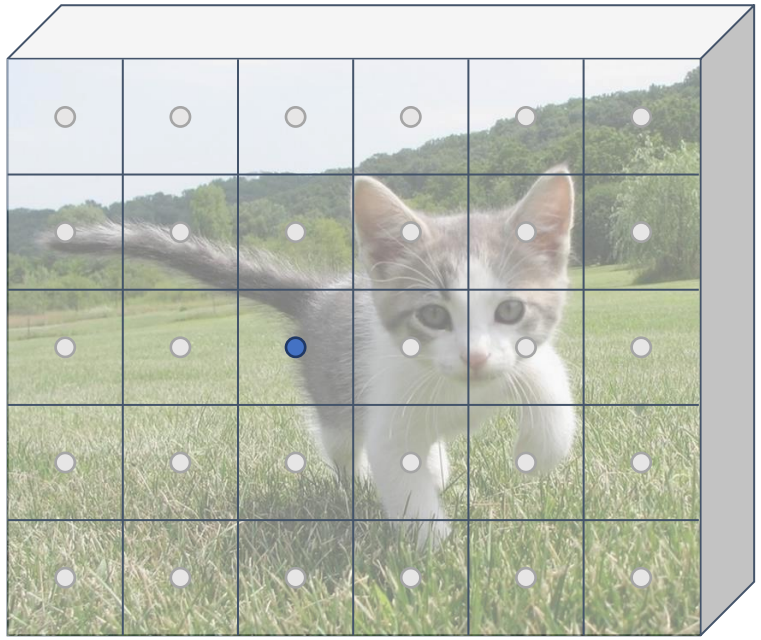


Image features
(e.g. 512 x 20 x 15)

Conv

→ Anchor classification
(C+1)K x 20 x 15

→ Anchor transforms
4K x 20 x 15

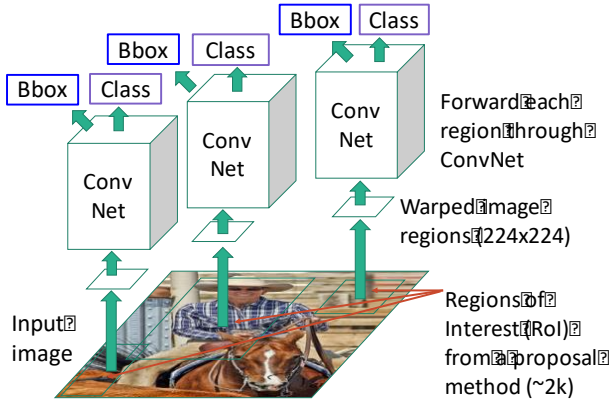
$$\mathrm{CE}(p_t) = -\log(p_t)$$

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

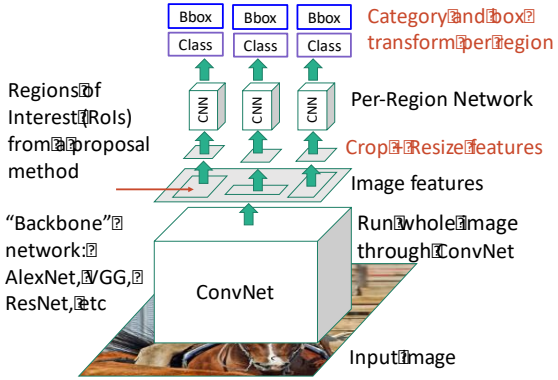
Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

Summary: Object Detection

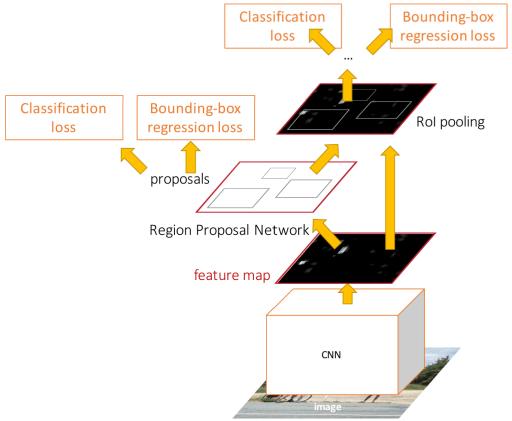
“Slow” R-CNN: Run CNN independently for each region



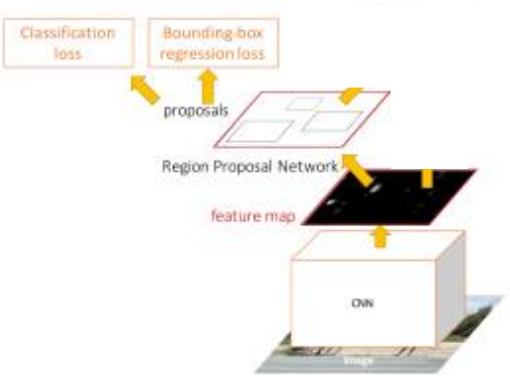
Fast R-CNN: Apply differentiable cropping to shared image features



Faster R-CNN: Compute proposals with CNN



Single-Stage: Fully convolutional detector (e.g., RetinaNet)



Computer Vision Task: Object Detection

Classification

CAT

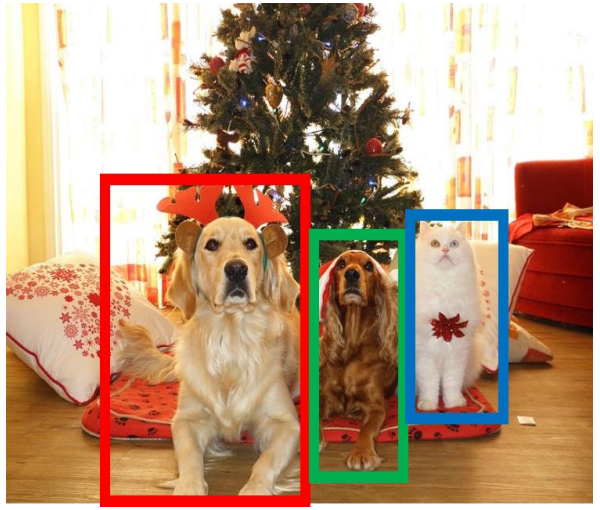
No spatial extent

Semantic Segmentation

GRASS, CAT, TREE, SKY

No objects, just pixels

Object Detection



DOG, DOG, CAT

Instance Segmentation

DOG, DOG, CAT

Multiple Objects

Computer Vision Task: Instance Segmentation

Classification

CAT

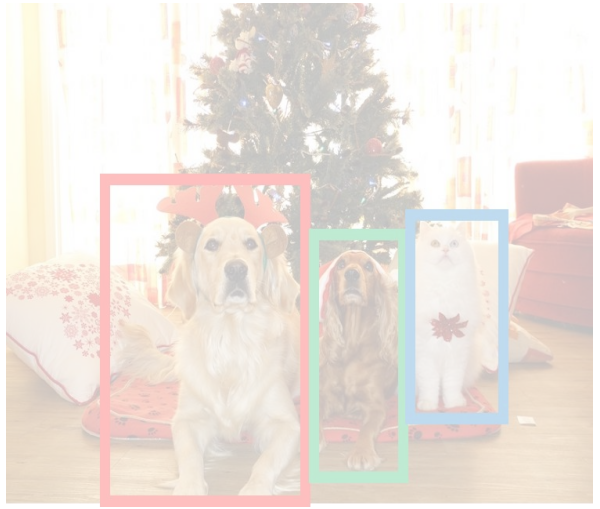
No spatial extent

Semantic Segmentation

GRASS, CAT, TREE, SKY

No objects, just pixels

Object Detection



DOG, DOG, CAT

Instance Segmentation

DOG, DOG, CAT

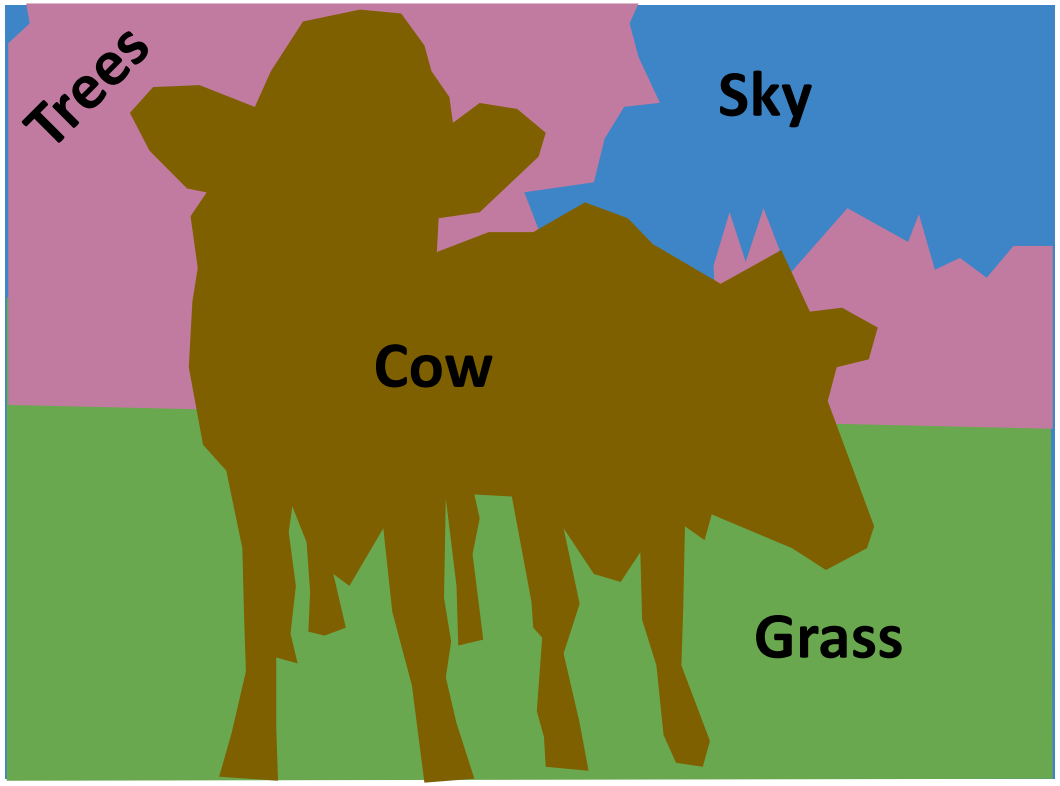
Multiple Objects

Computer Vision Tasks

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

This image is CC0 public domain

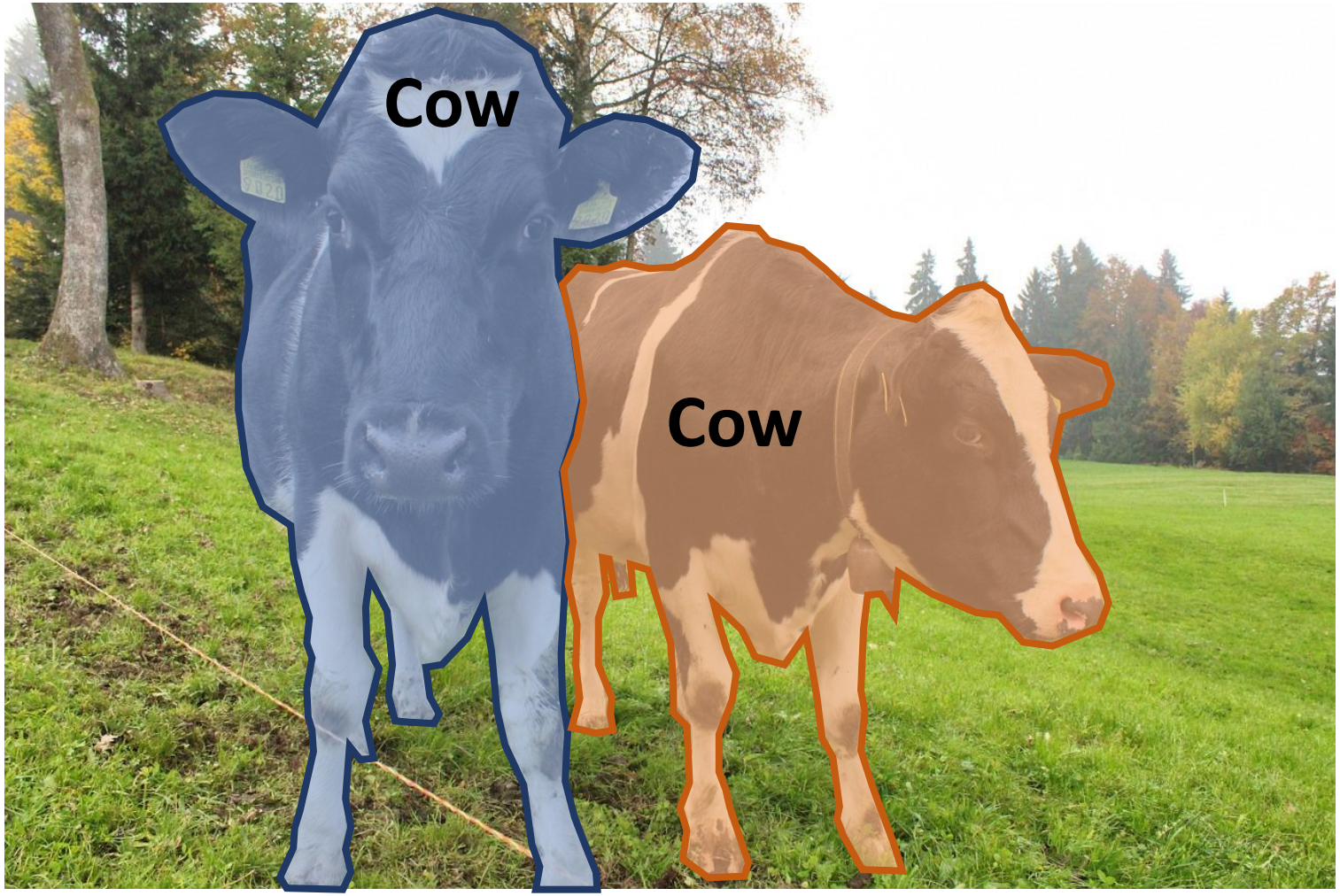
Semantic Segmentation: Gives per-pixel labels, but merges instances



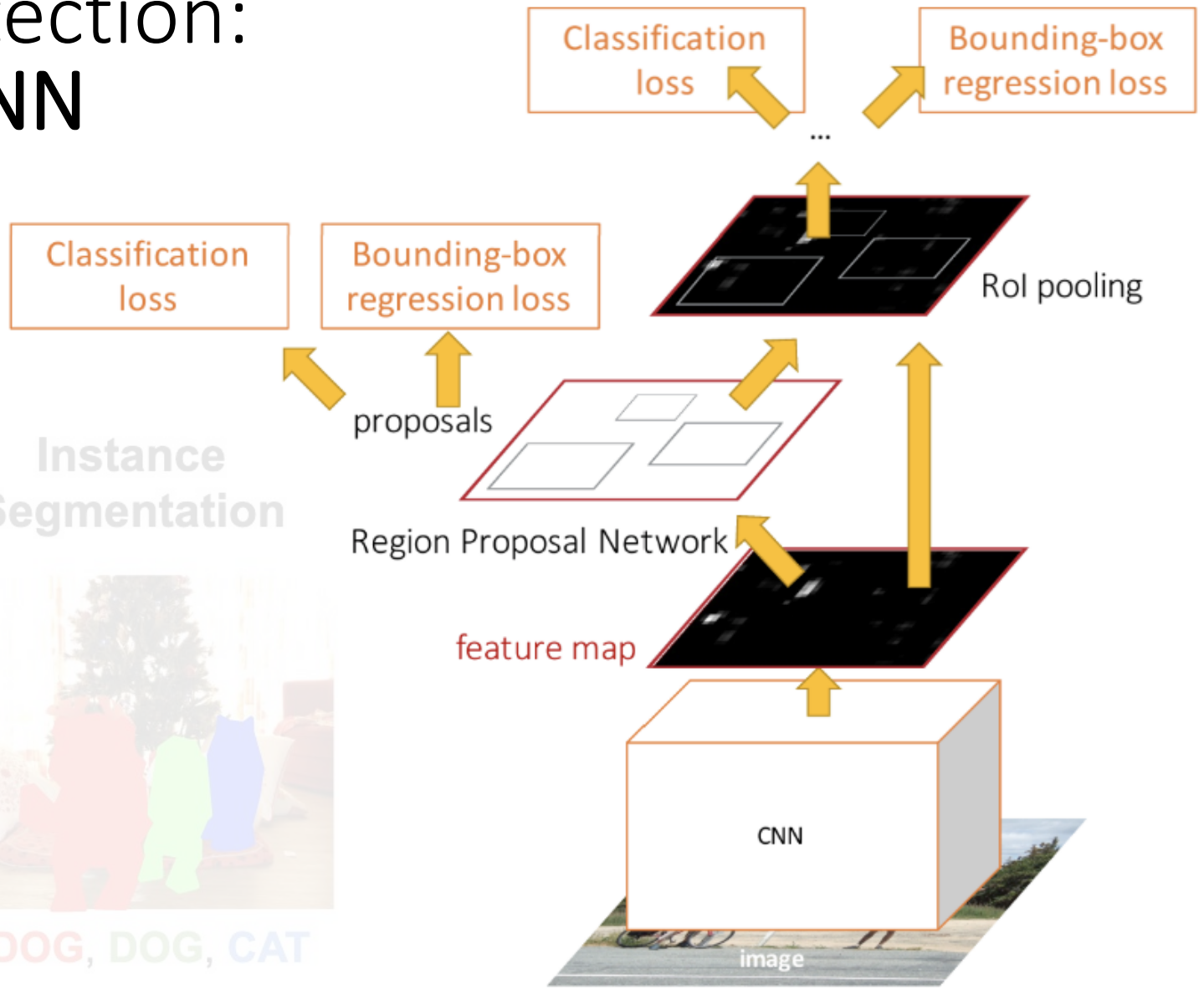
Computer Vision Task: Instance Segmentation

Instance Segmentation: Detect all objects in the image, and identify the pixels that belong to each object.

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



Object Detection: **Faster R-CNN**



Object Detection

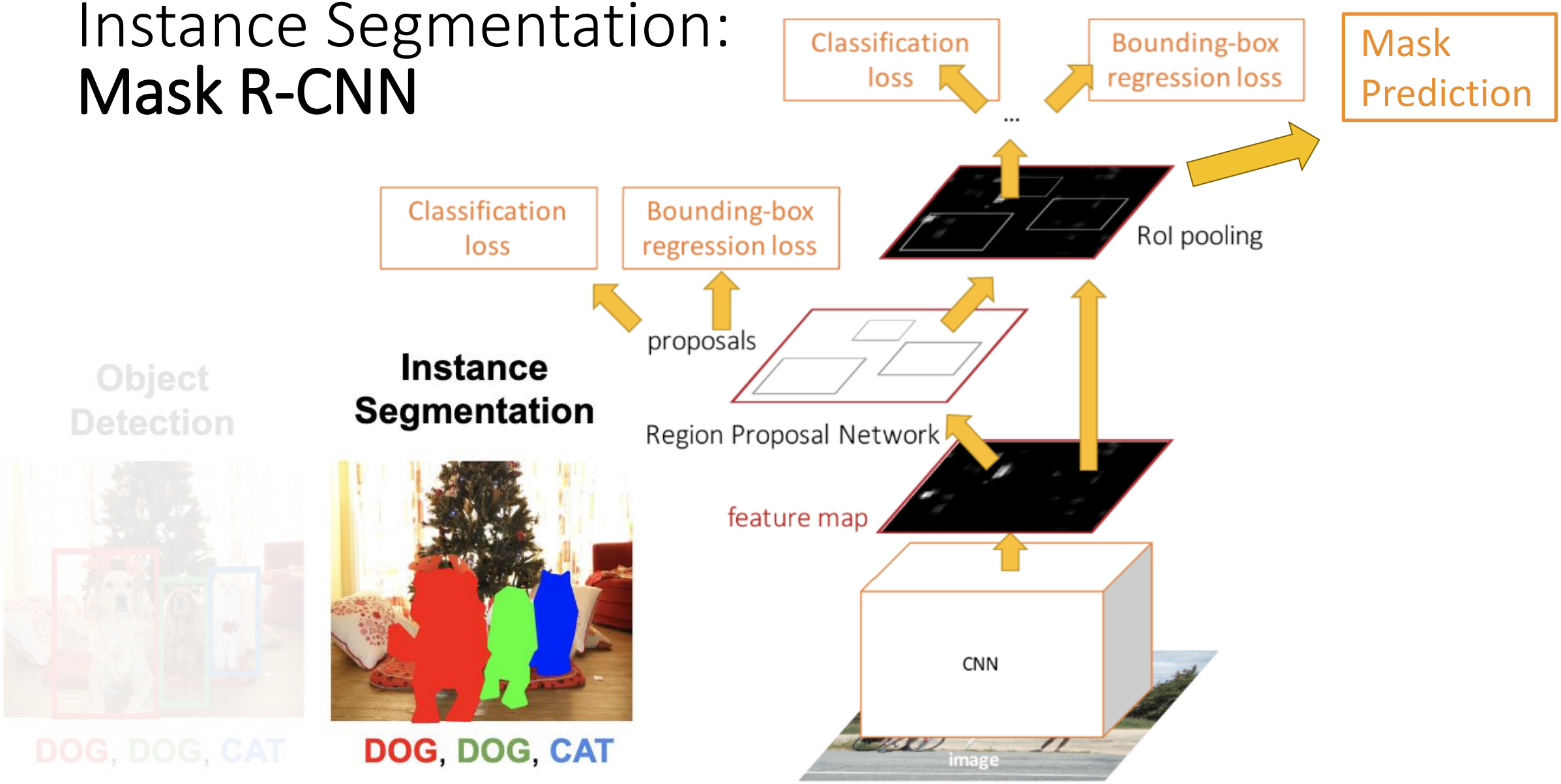
DOG, DOG, CAT

Instance Segmentation

DOG, DOG, CAT

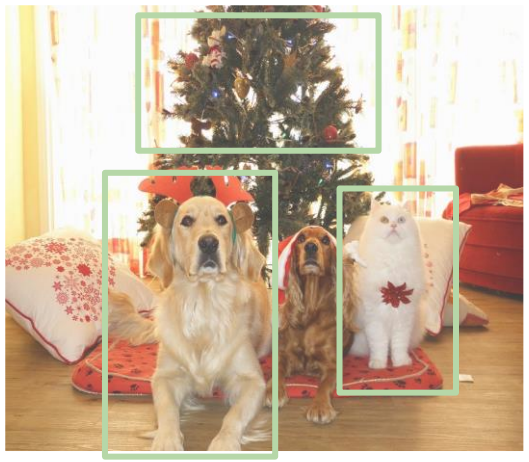
Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NeurIPS 2015

Instance Segmentation: **Mask R-CNN**

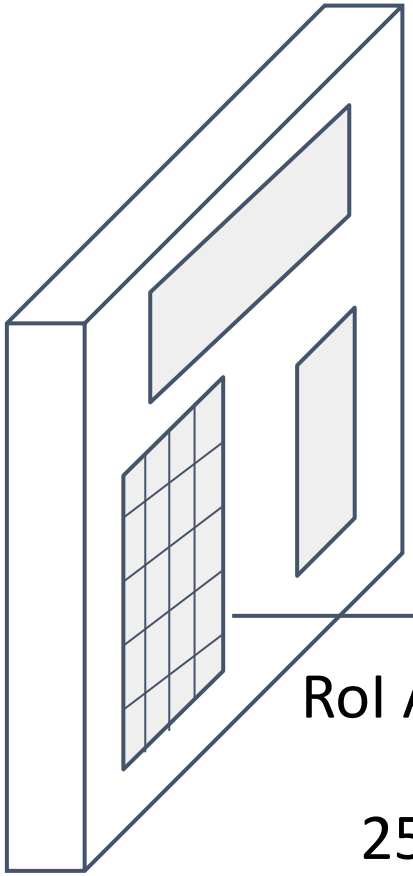


He et al, "Mask R-CNN", ICCV 2017

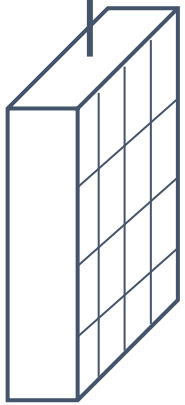
Mask R-CNN



CNN
+RPN

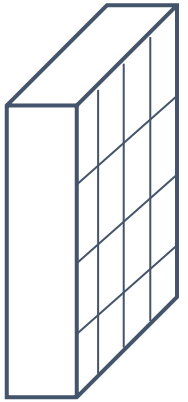


RoI Align



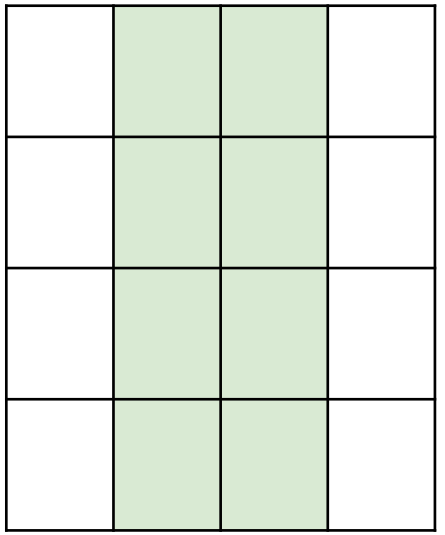
256 x 14 x 14

Conv



256 x 14 x 14

Conv

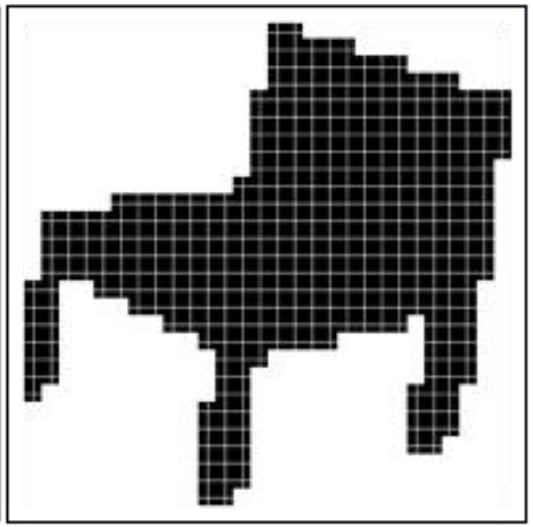


Classification Scores: C
Box coordinates (per class):
4 * C

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

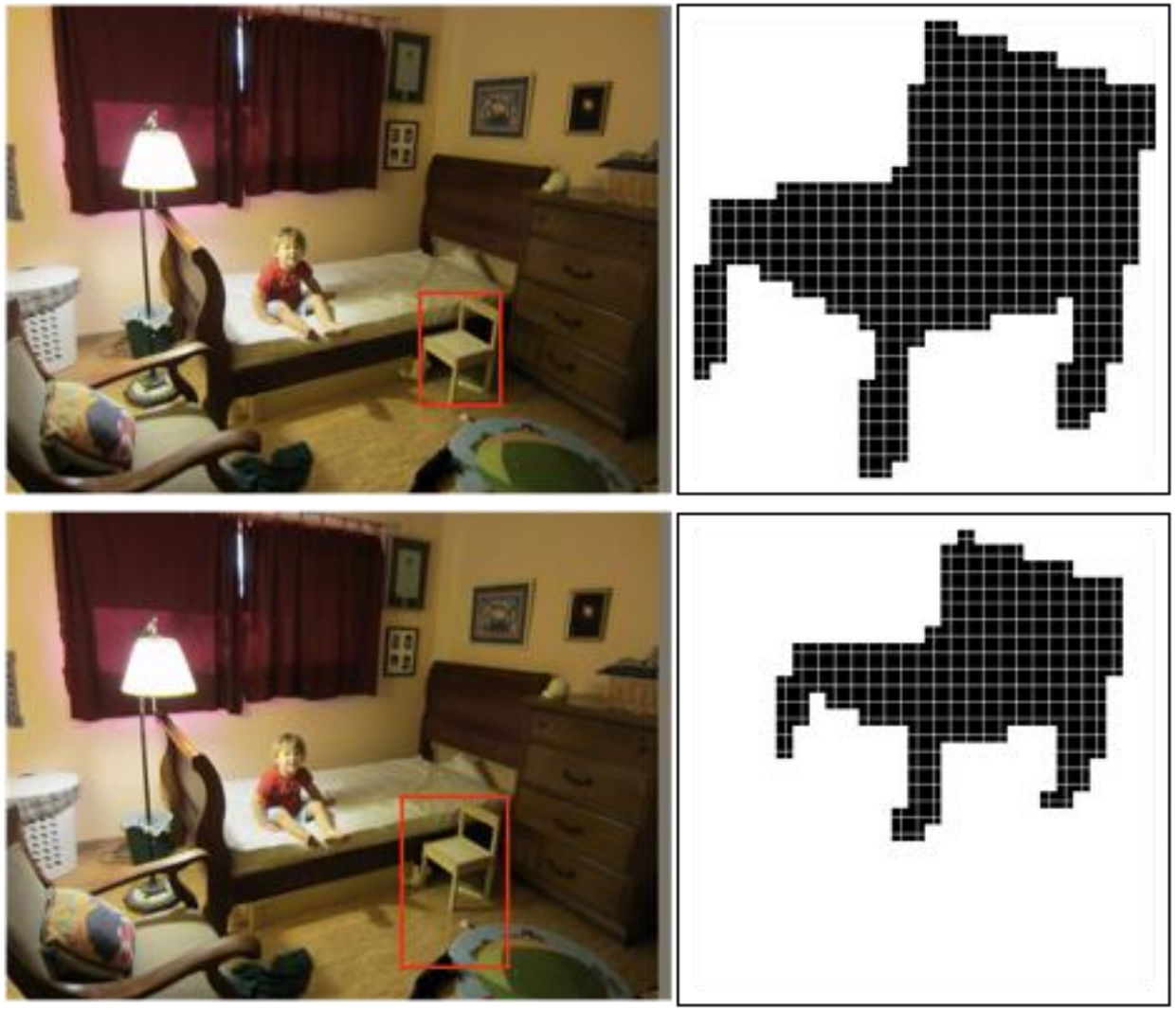
He et al, “Mask R-CNN”, ICCV 2017

Mask R-CNN: Example Training Targets



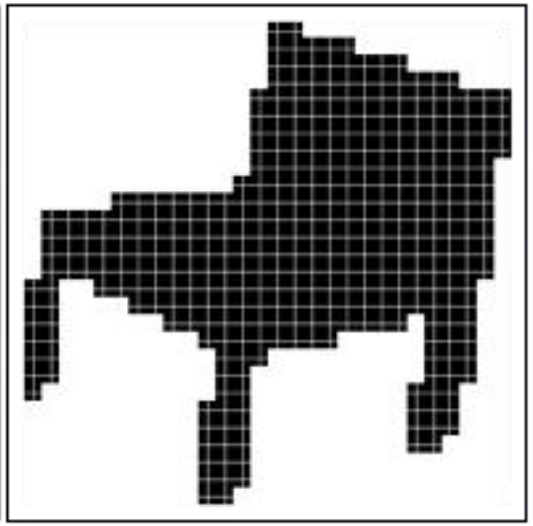
He et al, “Mask R-CNN”, ICCV 2017

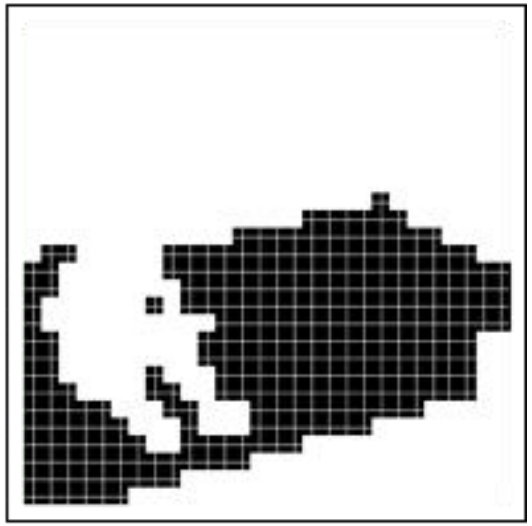
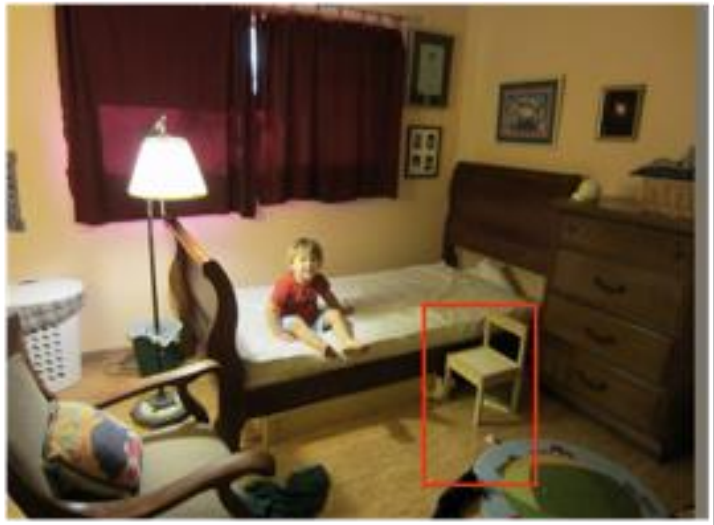
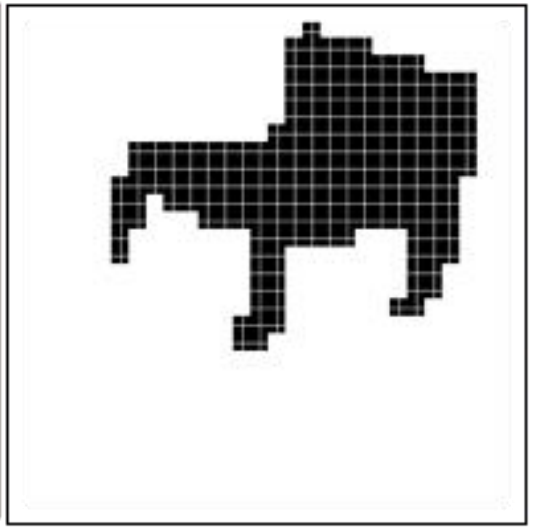
Mask R-CNN: Example Training Targets



He et al, “Mask R-CNN”, ICCV 2017

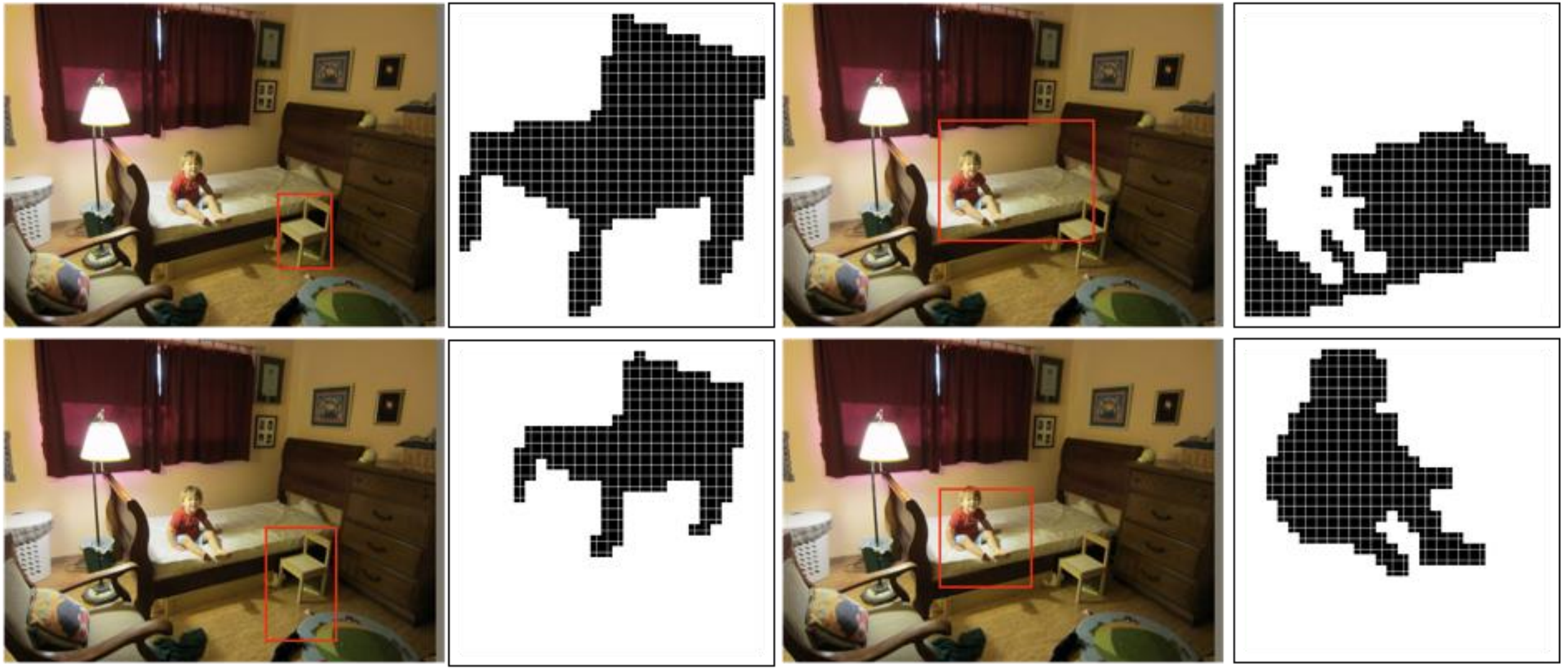
Mask R-CNN: Example Training Targets

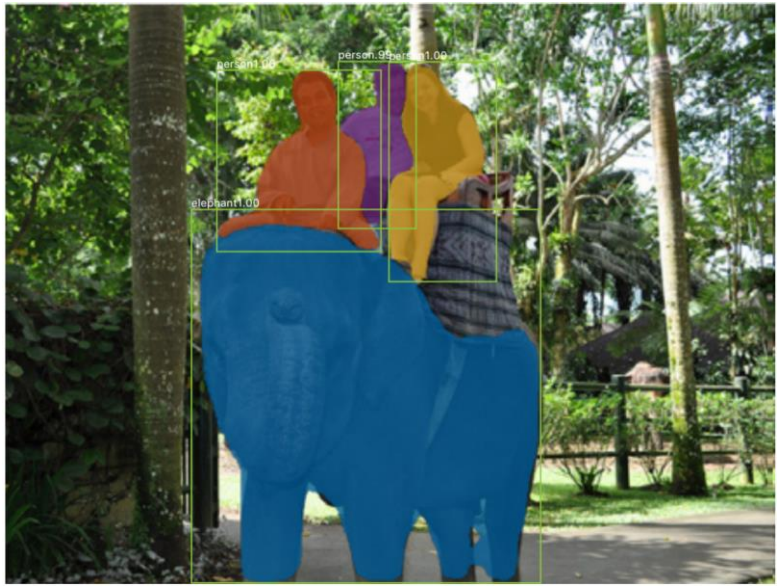
He et al, "Mask R-CNN", ICCV 2017

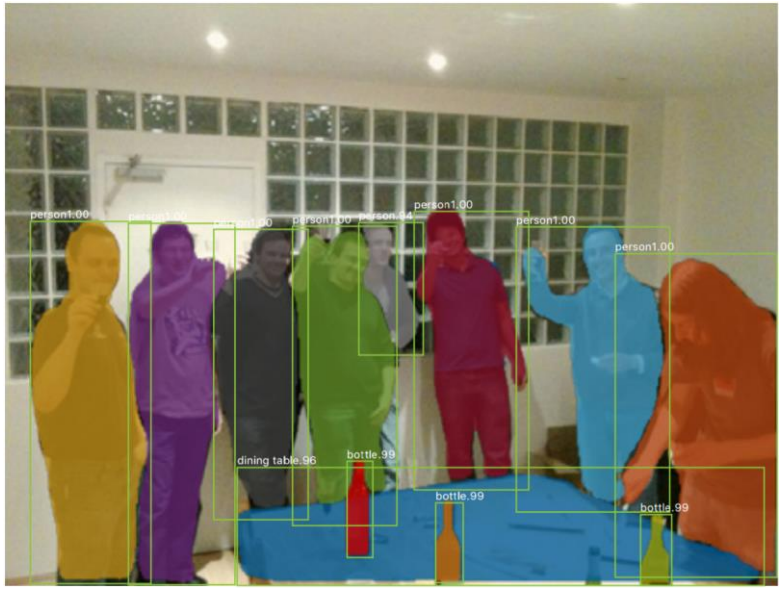
Mask R-CNN: Example Training Targets



He et al, “Mask R-CNN”, ICCV 2017

Mask R-CNN: Very Good Results!

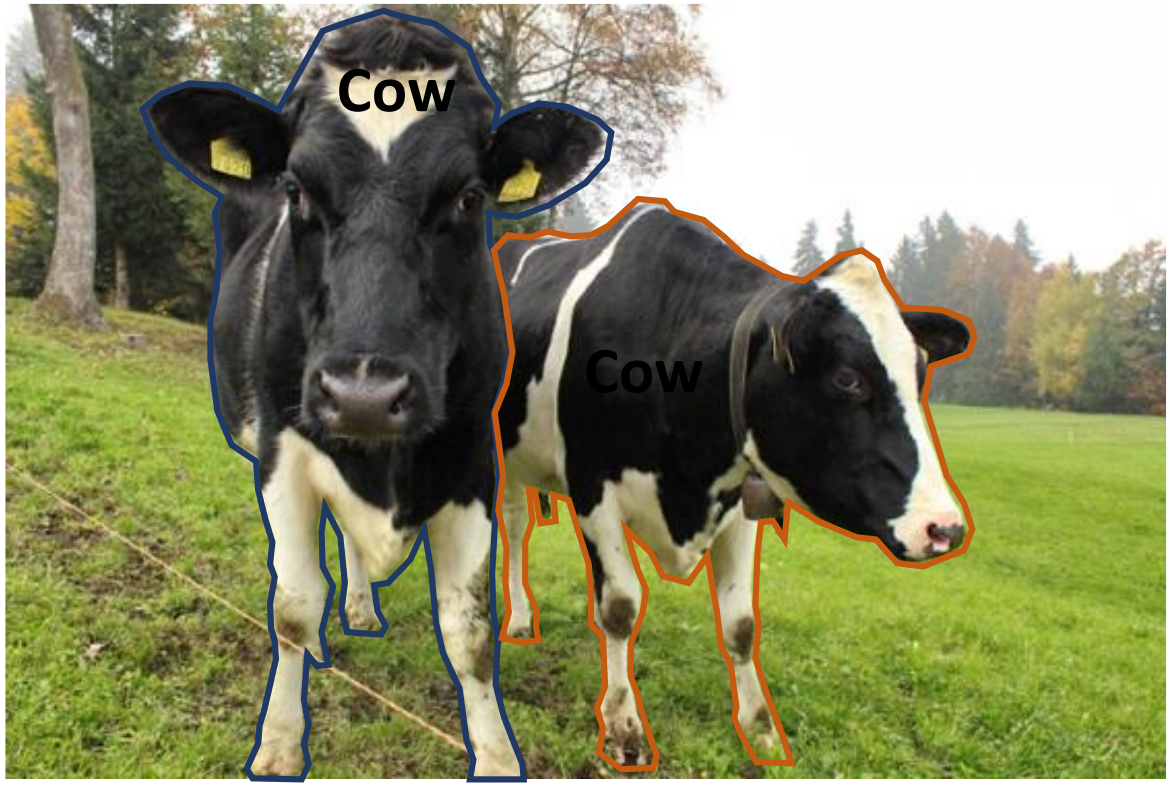




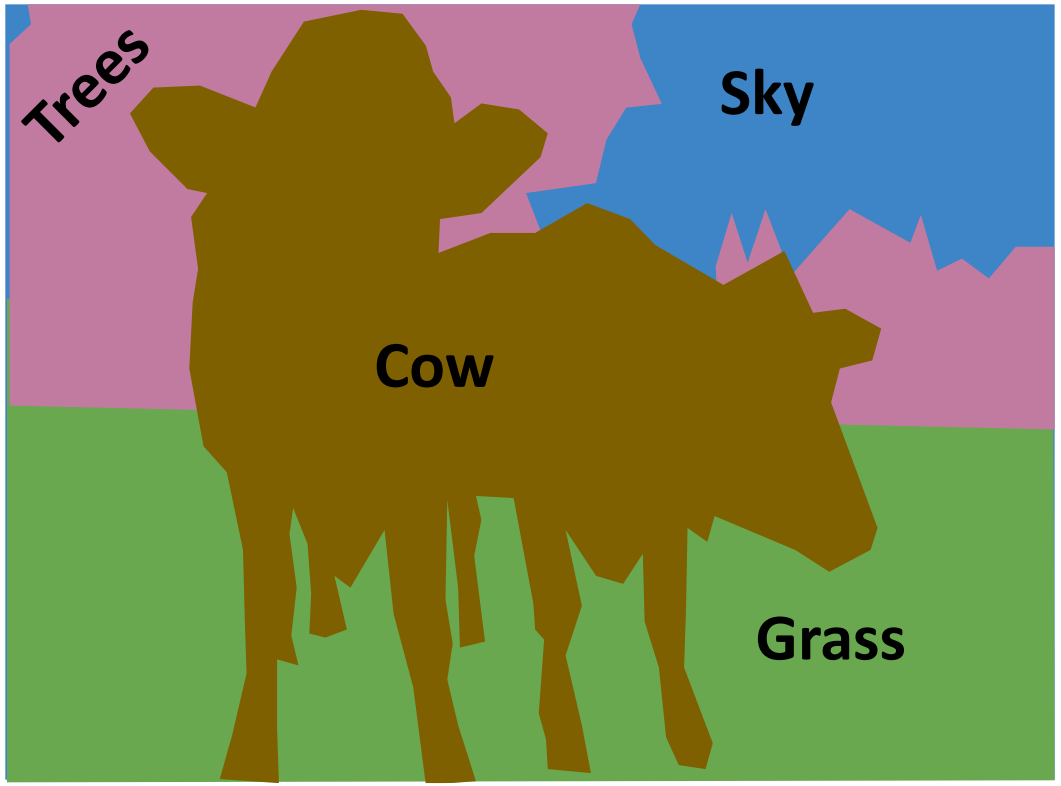
He et al, “Mask R-CNN”, ICCV 2017

Beyond Instance Segmentation

Instance Segmentation: Separate object instances, but only things



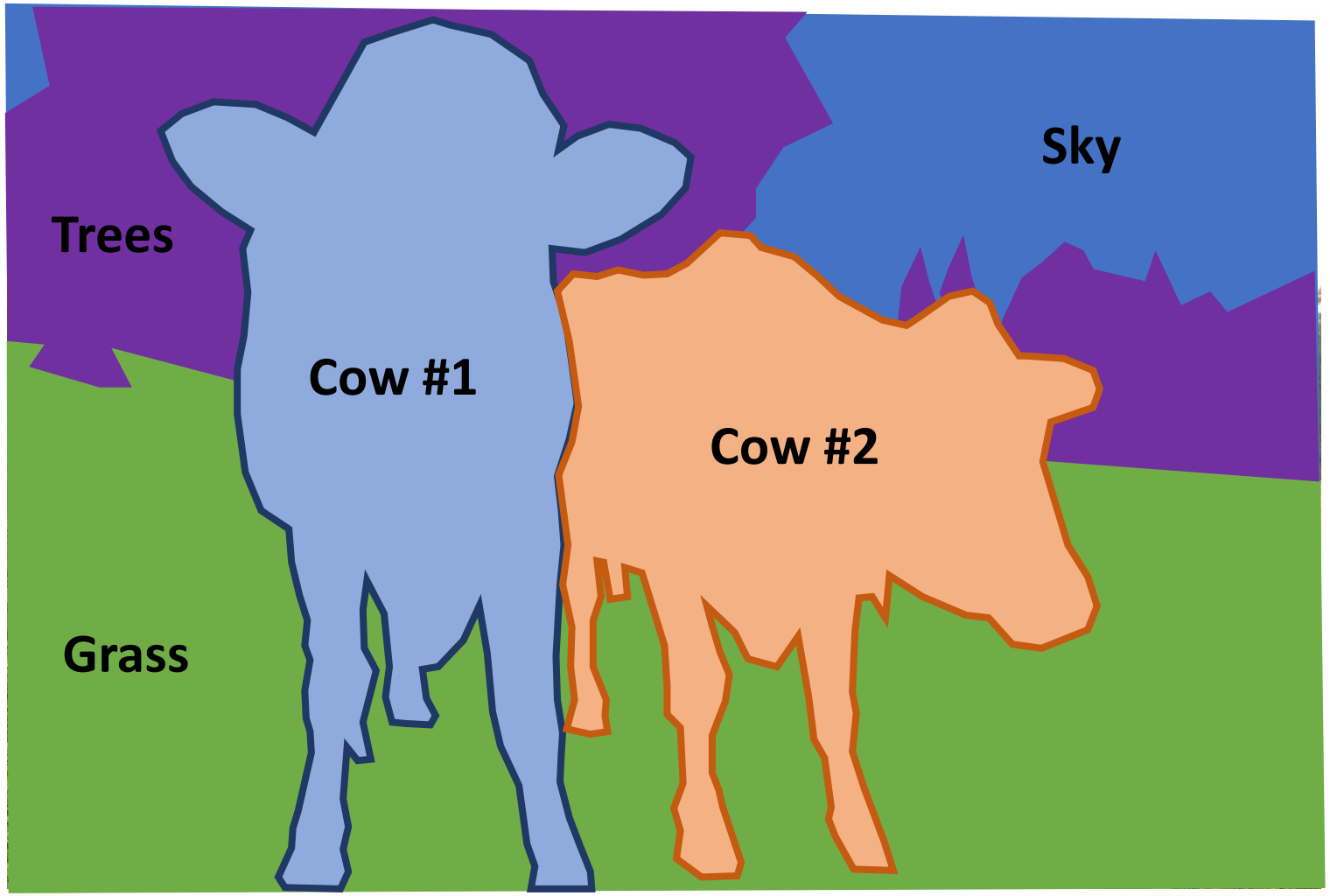
Semantic Segmentation: Identify both things and stuff, but doesn't separate instances



Beyond Instance Segmentation: **Panoptic Segmentation**

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

For “thing” categories also separate into instances

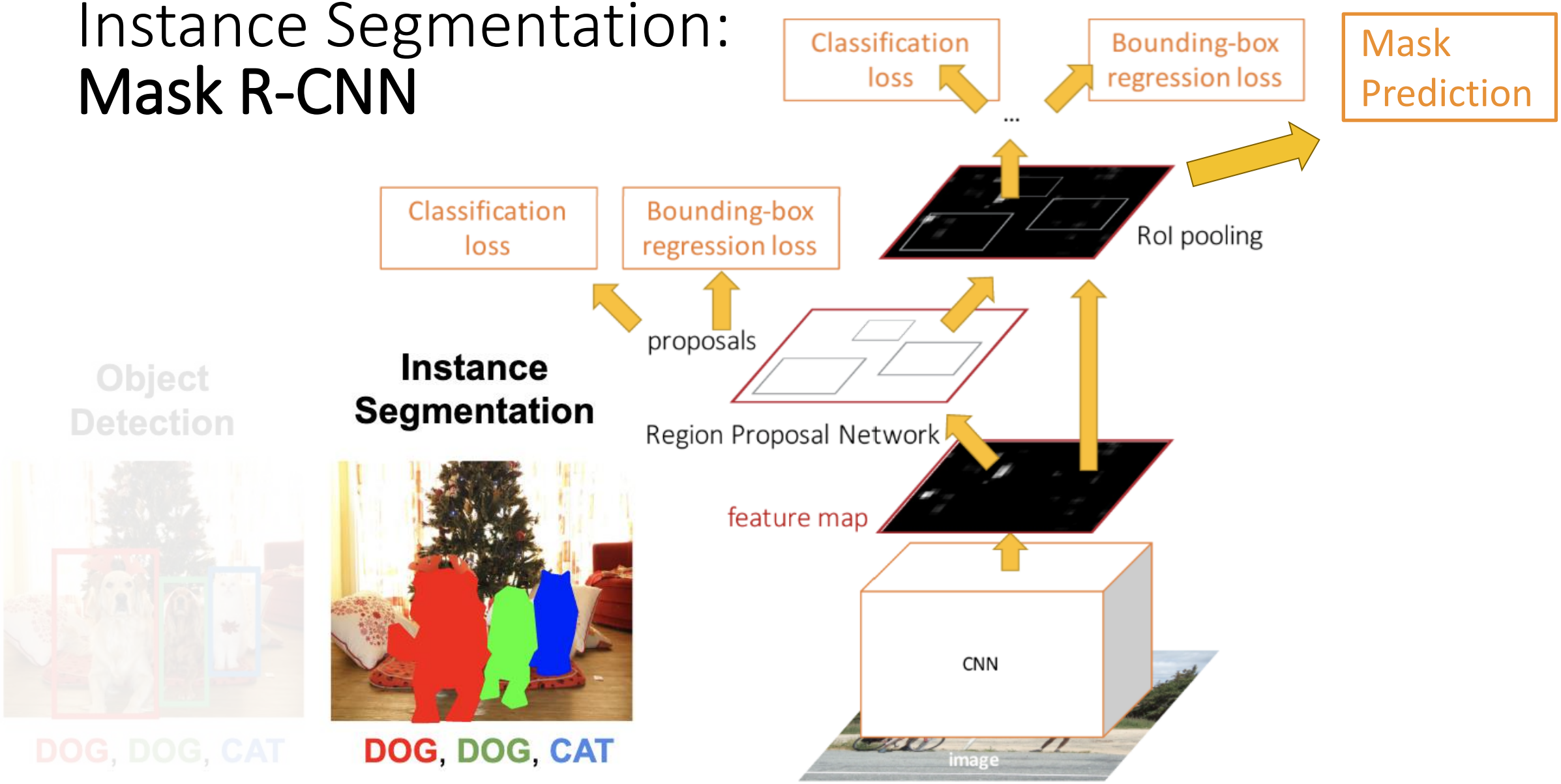


Kirillov et al, “Panoptic Segmentation”, CVPR 2019
Kirillov et al, “Panoptic Feature Pyramid Networks”, CVPR 2019

Beyond Instance Segmentation: **Panoptic Segmentation**

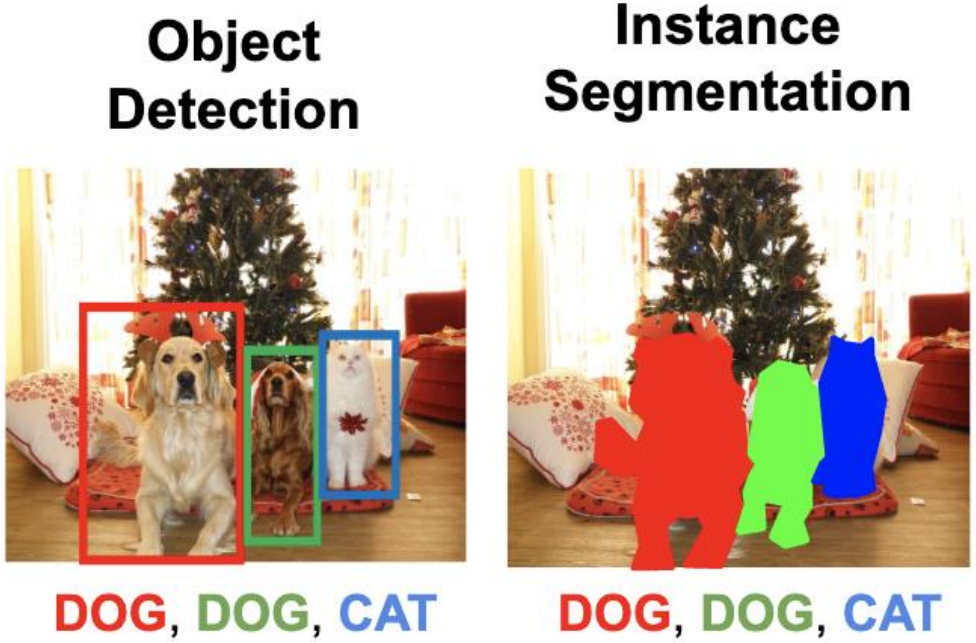
Kirillov et al, "Panoptic Feature Pyramid Networks", CVPR 2019

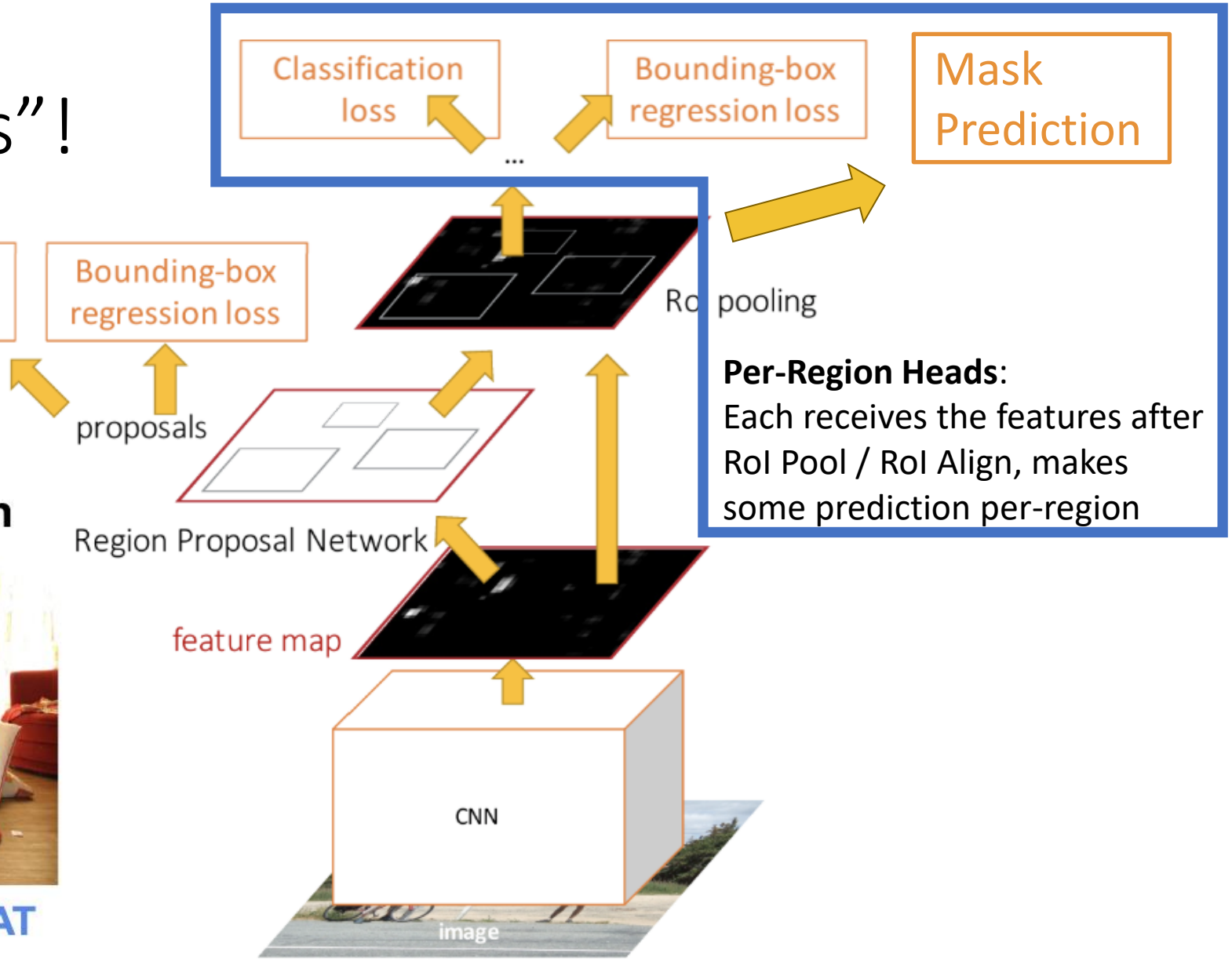
Instance Segmentation: **Mask R-CNN**



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

General Idea: Add Per-Region "Heads"!

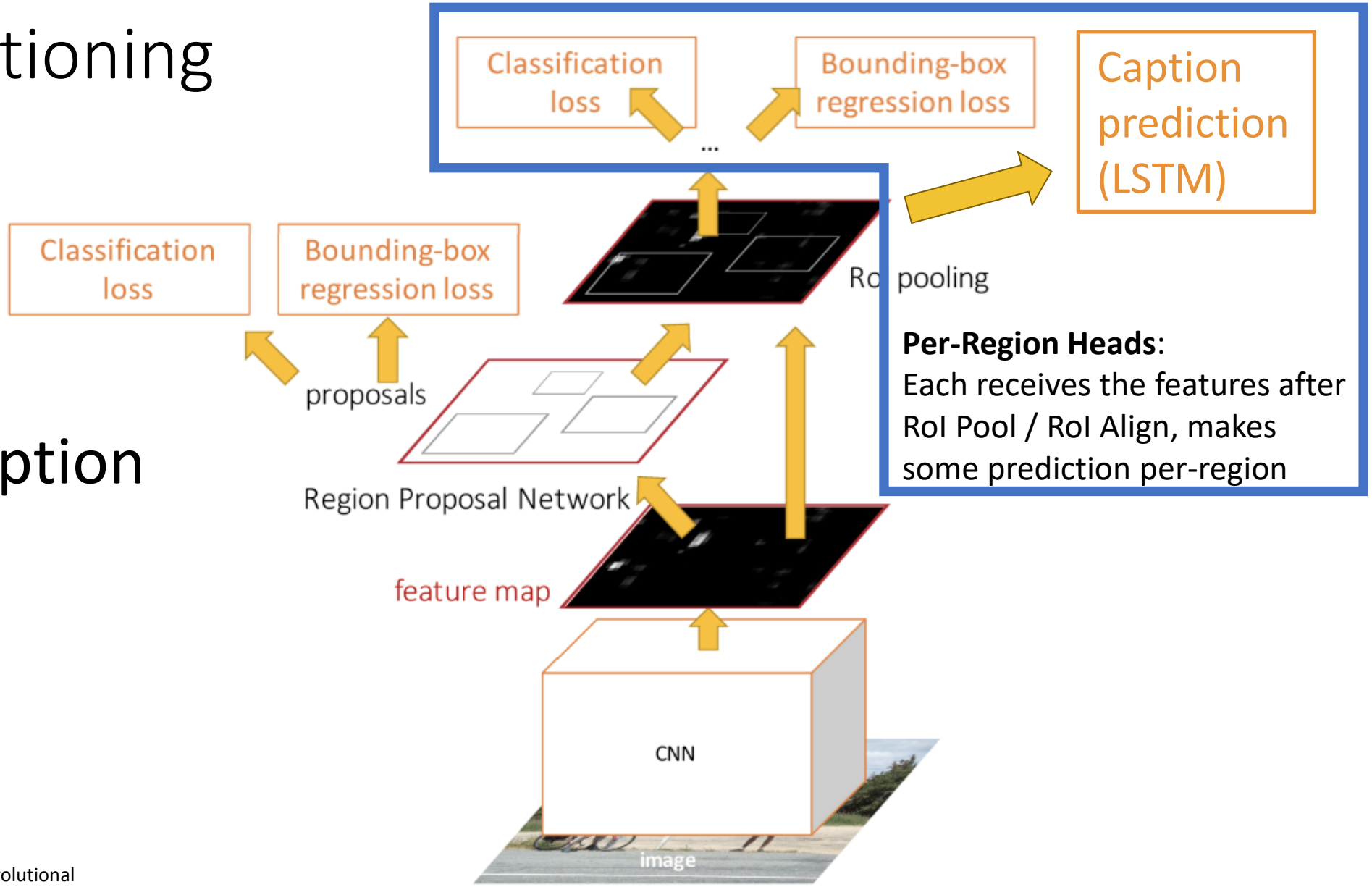




He et al, "Mask R-CNN", ICCV 2017

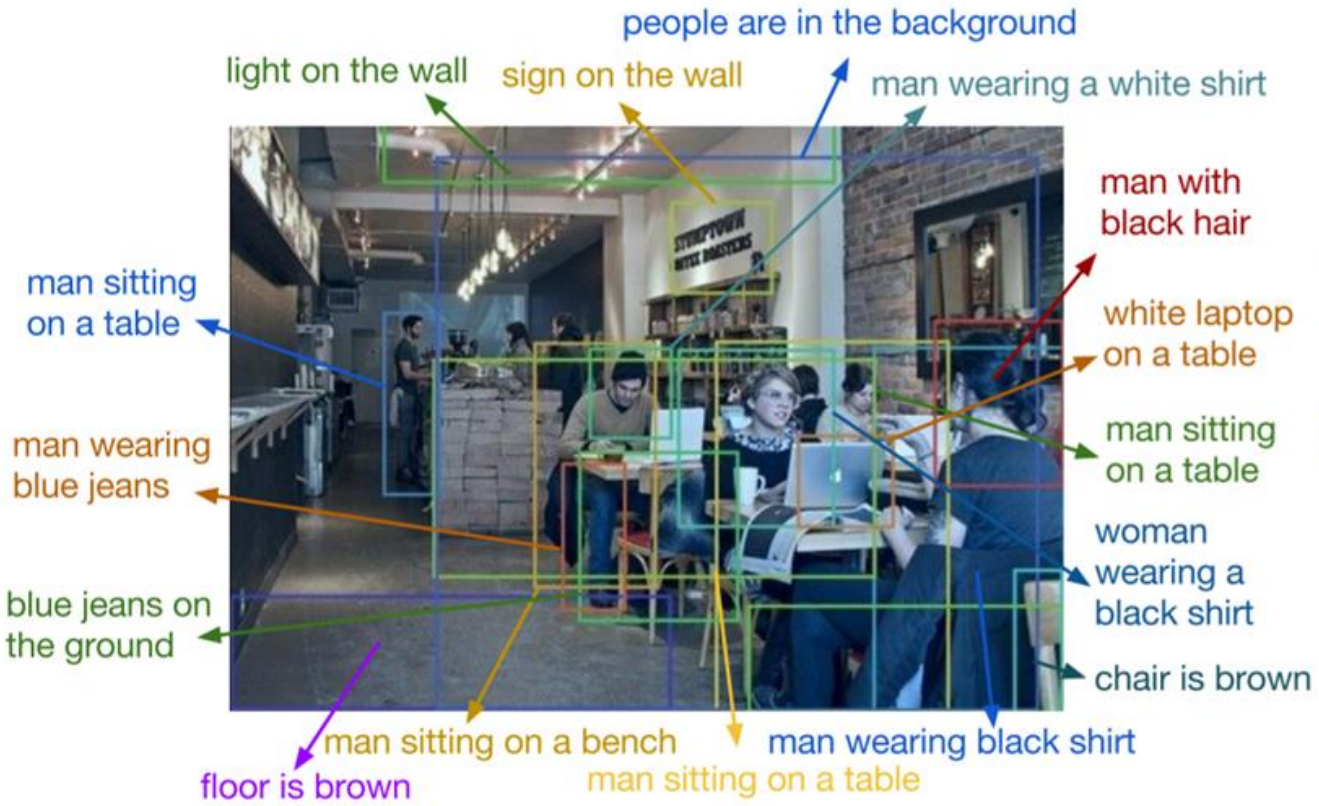
Dense Captioning

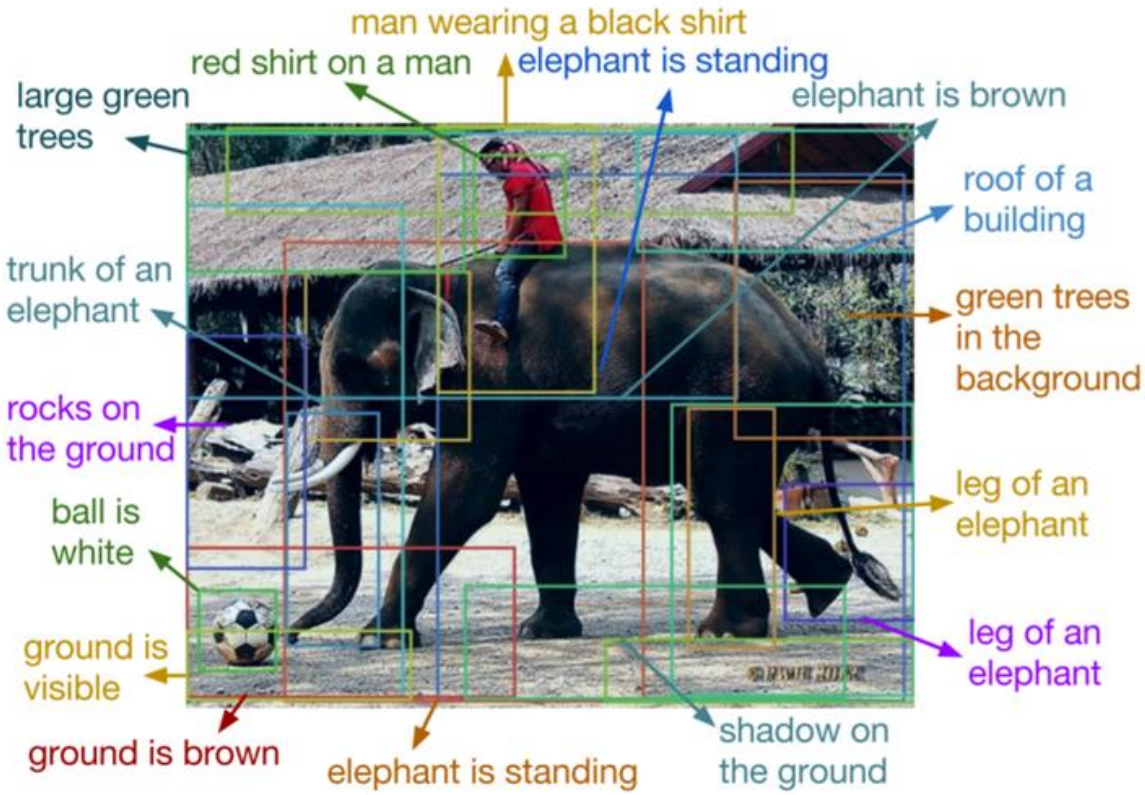
Predict a caption per region!



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

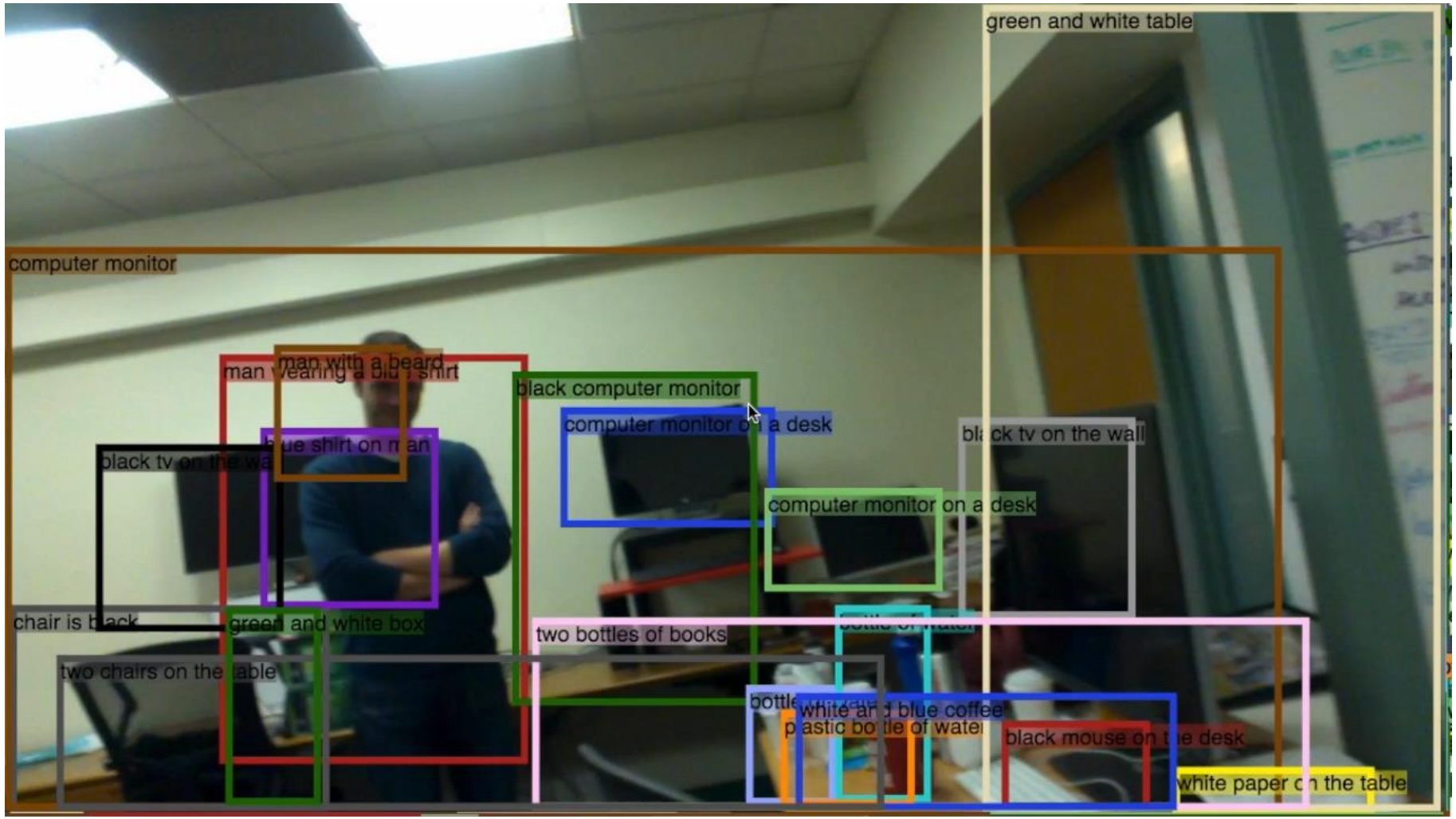
Dense Captioning





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

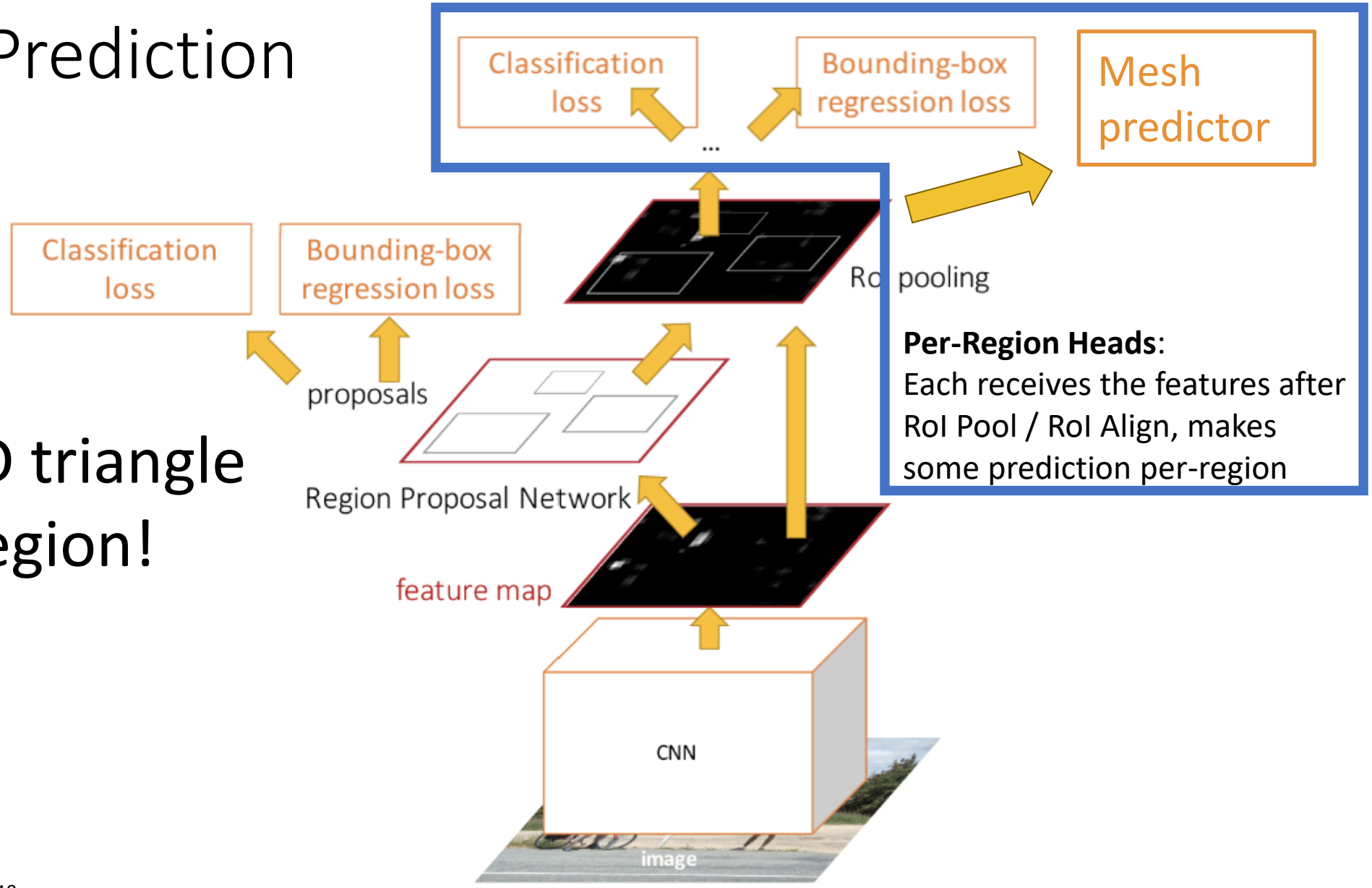
Dense Captioning



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!

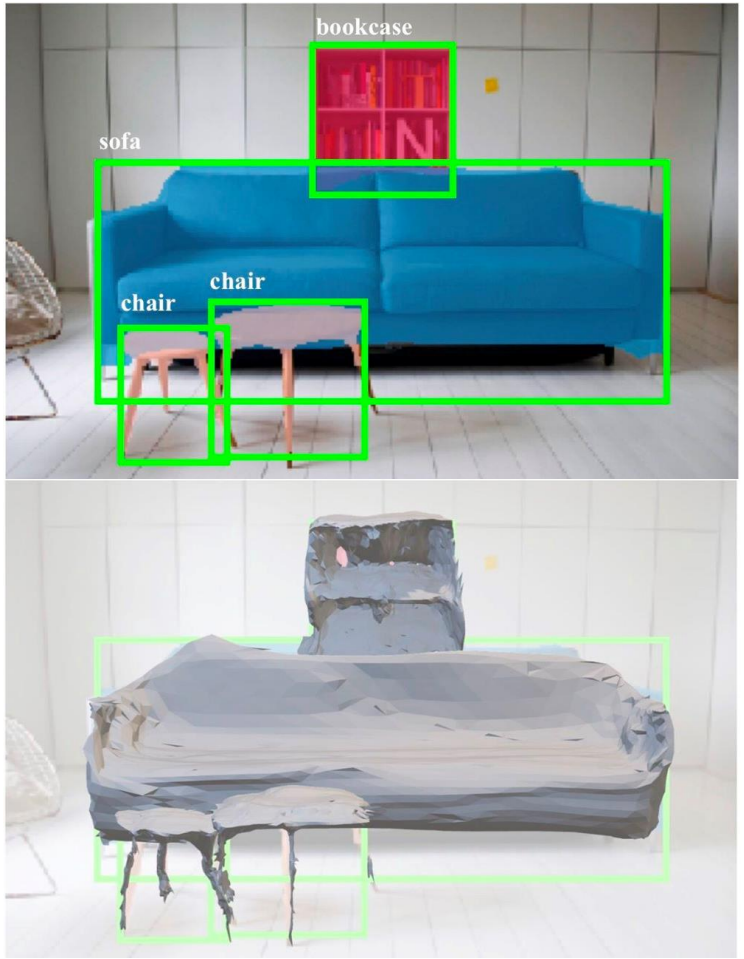


Gkioxari, Malik, and Johnson, "Mesh R-CNN", ICCV 2019

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

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

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



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

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

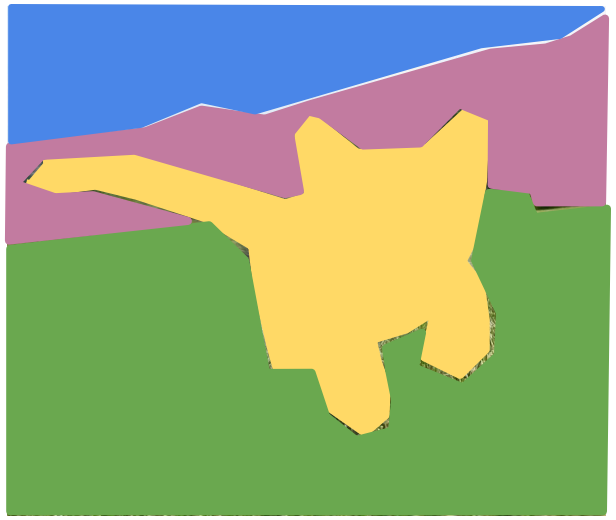
Summary: Many Computer Vision Tasks!

Classification

CAT

No spatial extent

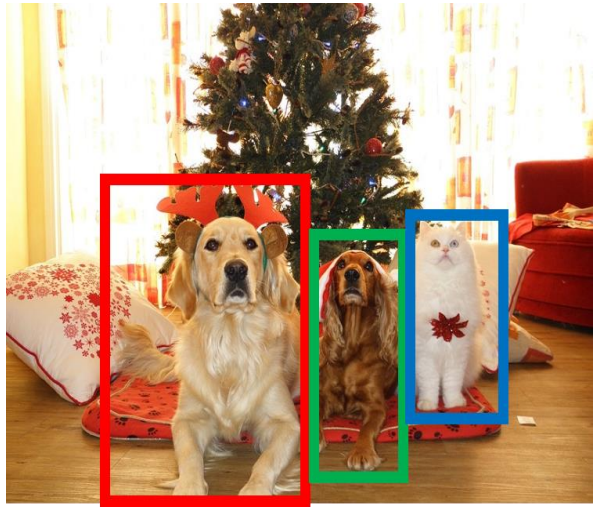
Semantic Segmentation



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

No objects, just pixels

Object Detection



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

Instance Segmentation

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

Multiple Objects