

# Lecture 15: Object Detection



# Reminder: A4

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

A4 covers:

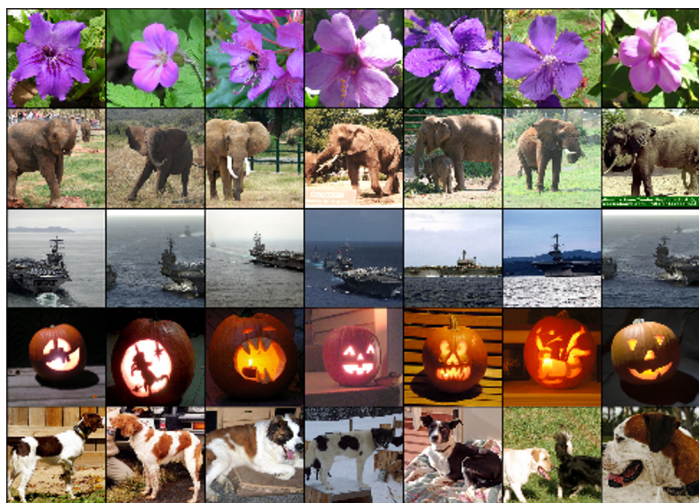
- PyTorch autograd
- Residual networks
- Recurrent neural networks
- Attention
- Feature visualization
- Style transfer
- Adversarial examples

# Last Time: Visualizing and Understanding CNNs

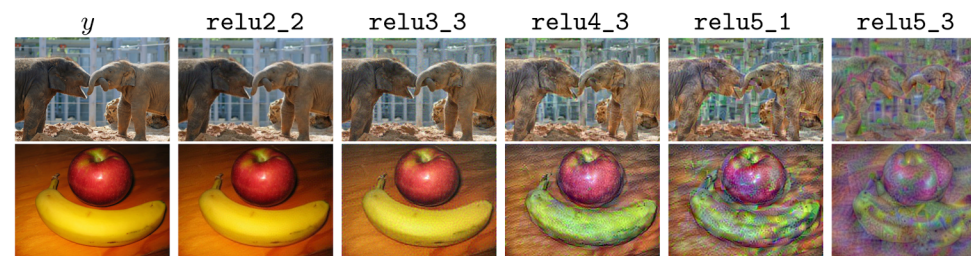
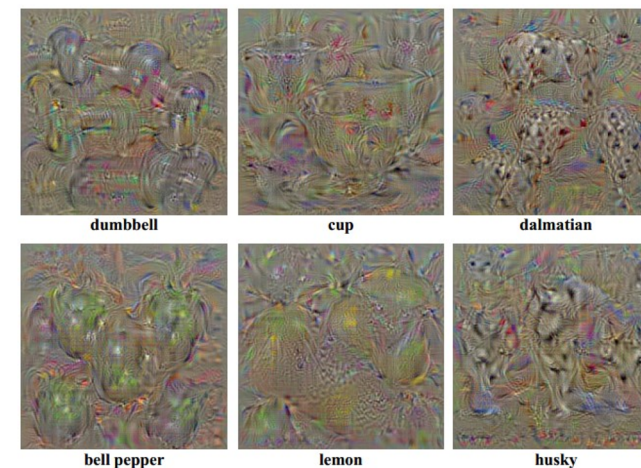
Maximally Activating Patches

Synthetic Images via  
Gradient Ascent

Nearest Neighbor



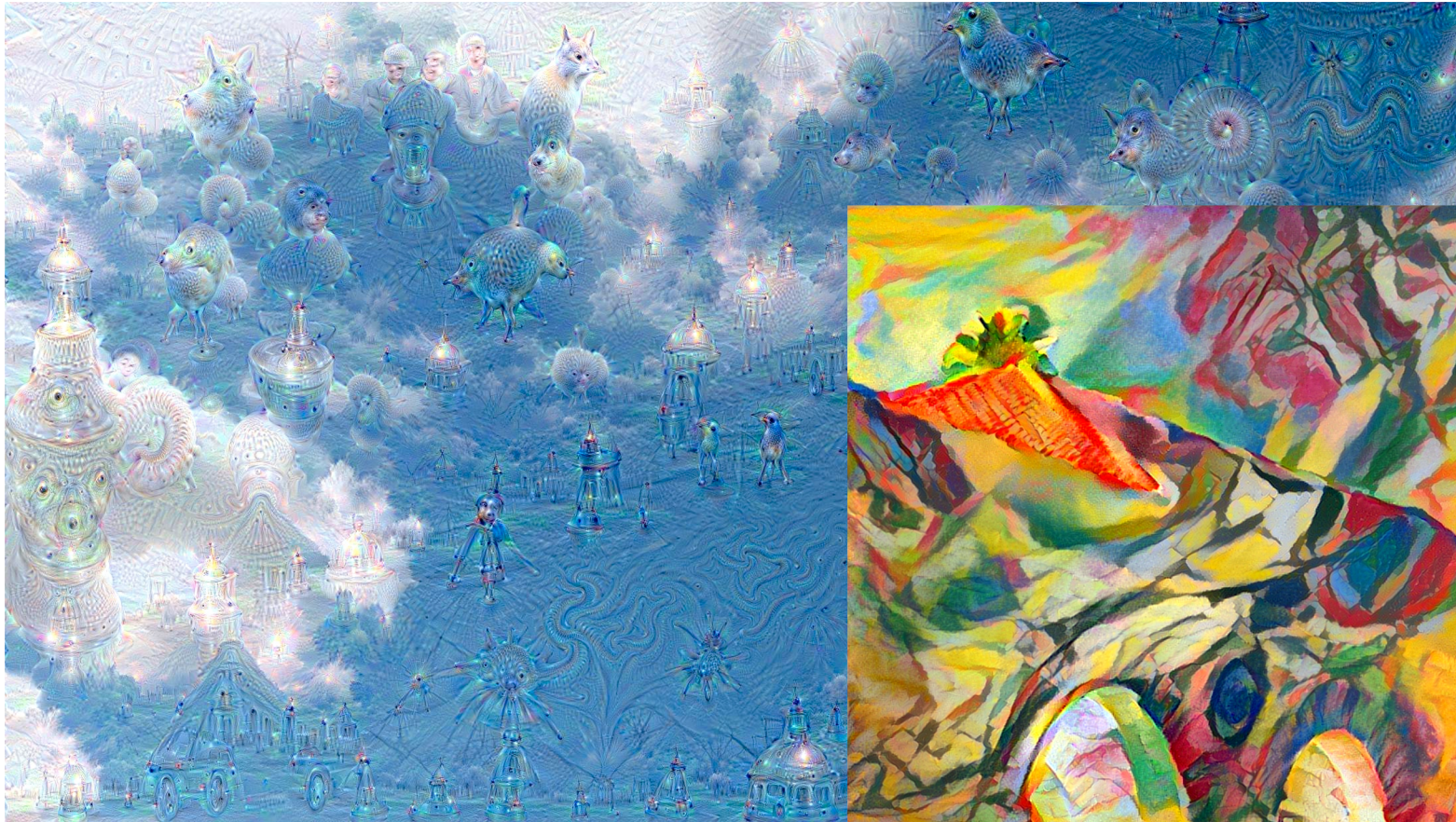
(Guided) Backprop



Feature Inversion



# Last Time: Making art with CNNs



DeepDream

Style Transfer





# So far: Image Classification



[This image](#) is [CC0 public domain](#)

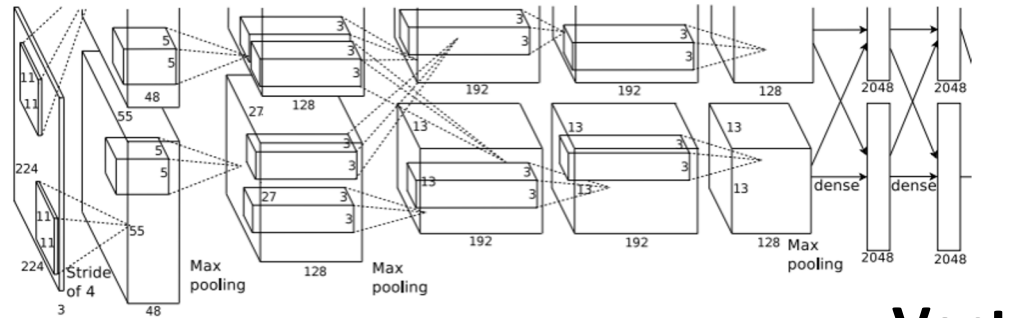


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

**Vector:**  
4096

→  
**Fully-Connected:**  
4096 to 1000

**Class Scores**

Cat: 0.9

Dog: 0.05

Car: 0.01

...

# Computer Vision Tasks

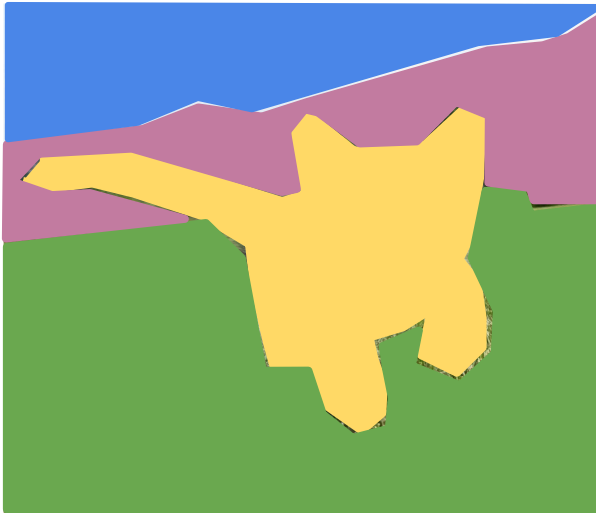
## Classification



**CAT**

No spatial extent

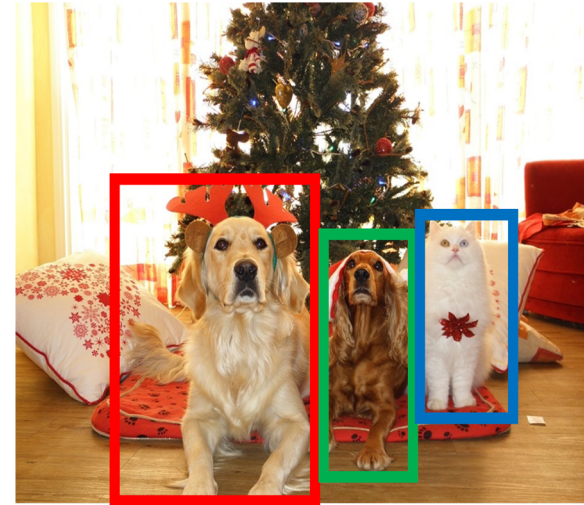
## Semantic Segmentation



**GRASS, CAT, TREE, SKY**

No objects, just pixels

## Object Detection



**DOG, DOG, CAT**

Multiple Objects

## Instance Segmentation



**DOG, DOG, CAT**

[This image is CC0 public domain](#)

# Today: Object Detection

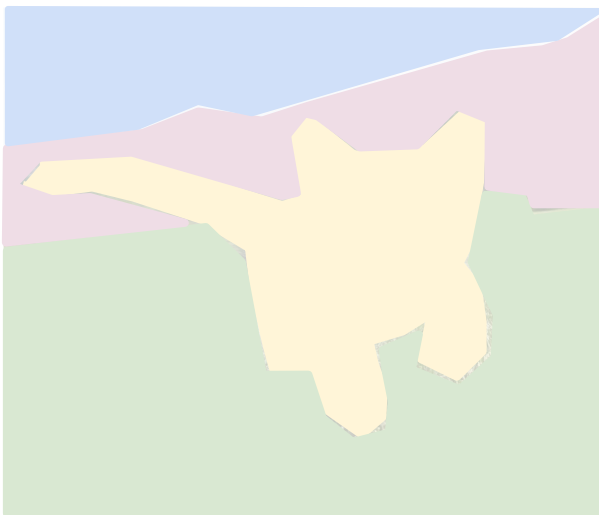
## Classification



CAT

No spatial extent

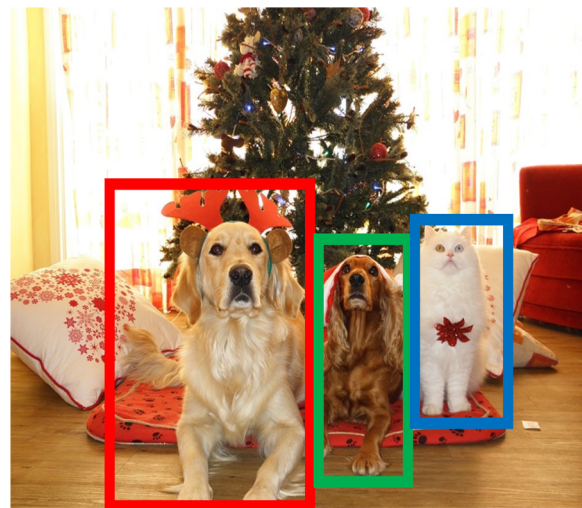
## Semantic Segmentation



GRASS, CAT, TREE,  
SKY

No objects, just pixels

## Object Detection



DOG, DOG, CAT

Multiple Objects

## Instance Segmentation



DOG, DOG, CAT

[This image is CC0 public domain](#)

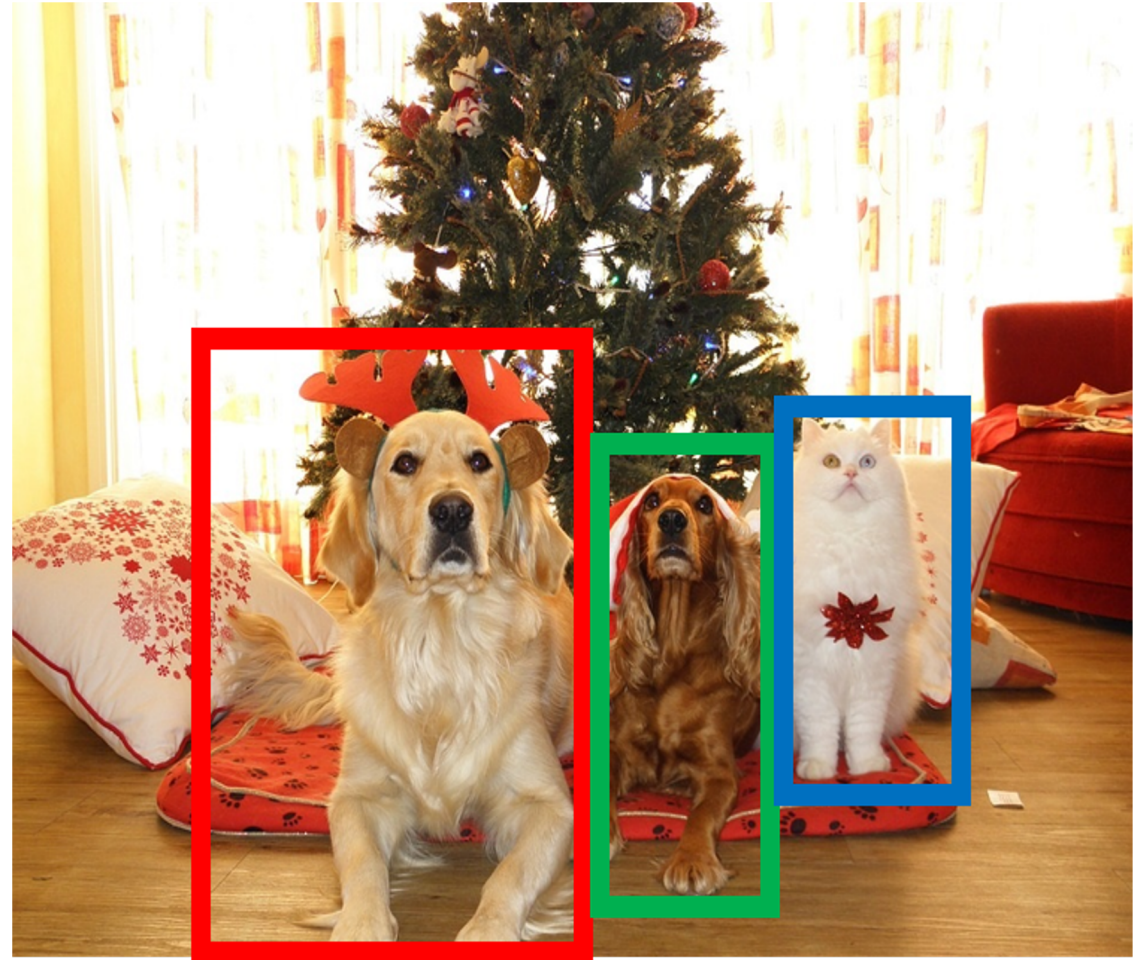


# Object Detection: Task Definition

**Input:** Single RGB Image

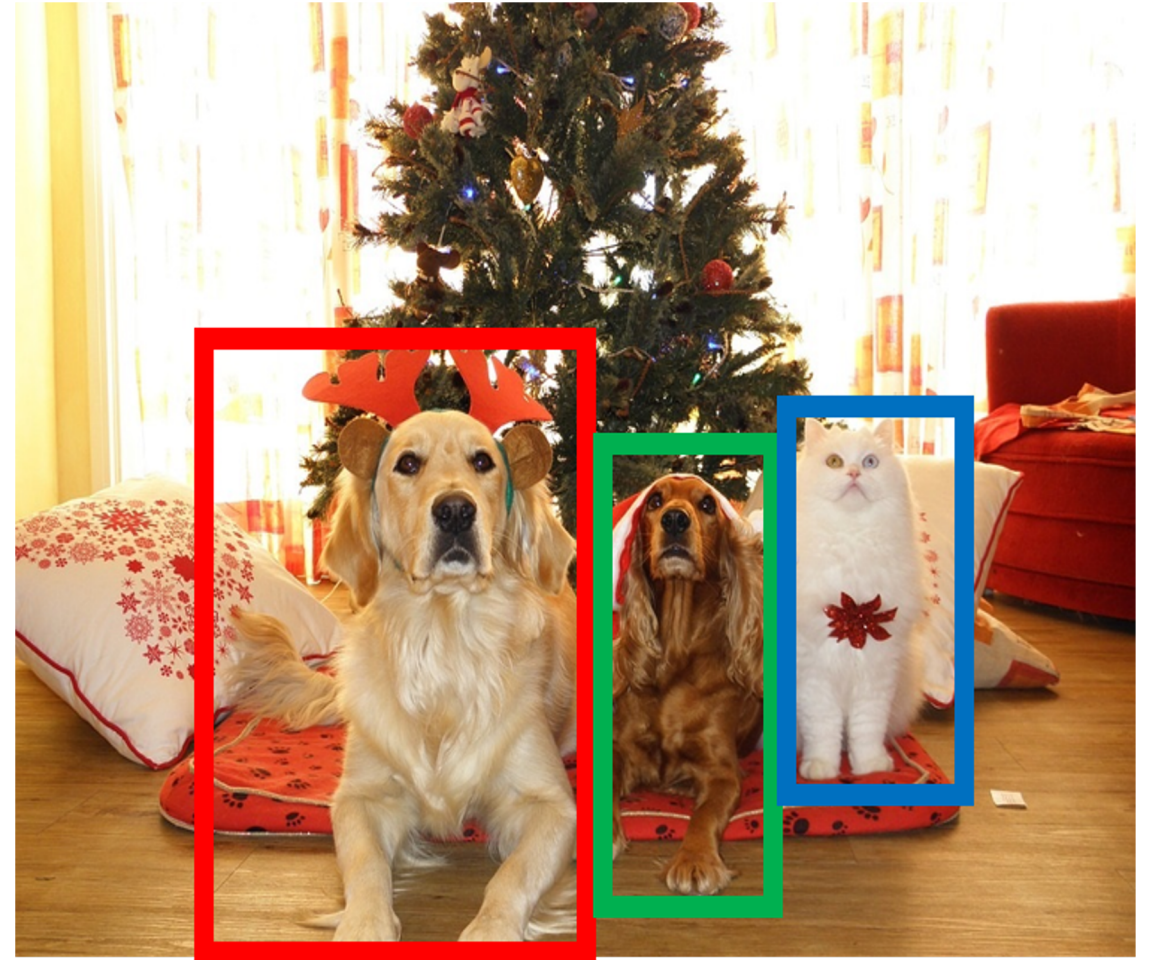
**Output:** A set of detected objects;  
For each object predict:

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

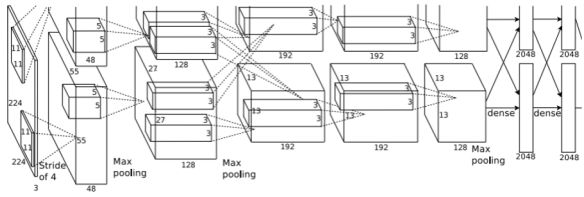




# Detecting a single object



[This image](#) is [CC0 public domain](#)



**Vector:**  
4096

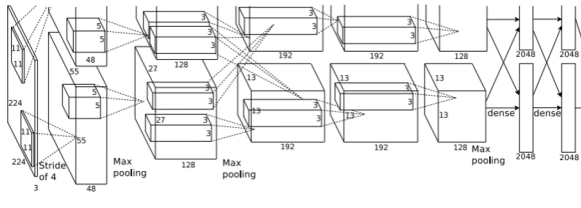
# Detecting a single object

“What”

Correct label:  
Cat



[This image](#) is [CC0 public domain](#)



**Vector:**  
4096

Fully  
Connected:  
4096 to 1000

**Class Scores**

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

**Softmax  
Loss**

# Detecting a single object

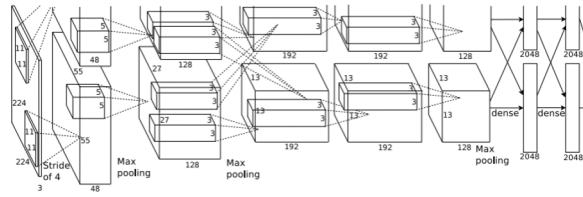
“What”

Correct label:  
Cat



[This image](#) is [CC0 public domain](#)

Treat localization as a regression problem!



Fully  
Connected:  
4096 to 1000

**Class Scores**

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

**Softmax  
Loss**

**Vector:**  
4096

Fully  
Connected:  
4096 to 4

**Box  
Coordinates**  
(x, y, w, h)

**L2 Loss**

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

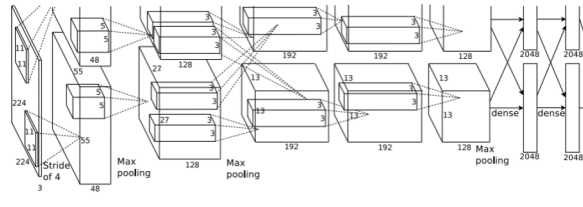
“Where”

# Detecting a single object



[This image](#) is [CC0 public domain](#)

Treat localization as a regression problem!



**Vector:**  
4096

“Where”

Fully  
Connected:  
4096 to 1000

“What”

**Class Scores**

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

Fully  
Connected:  
4096 to 4

**Box  
Coordinates**  
(x, y, w, h)

**Correct label:**  
Cat

**Softmax  
Loss**

**Weighted  
Sum**

**Loss**

**L2 Loss**

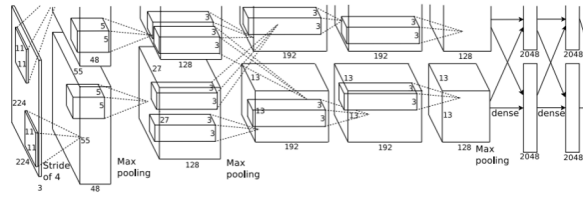
**Correct box:**  
(x', y', w', h')

# Detecting a single object



[This image](#) is [CC0 public domain](#)

Treat localization as a regression problem!



**Vector:**  
4096

“Where”

Fully  
Connected:  
4096 to 1000

“What”

**Class Scores**

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

Fully  
Connected:  
4096 to 4

**Box  
Coordinates**  
(x, y, w, h)

**Correct label:**  
Cat

**Softmax  
Loss**

Multitask  
Loss

**Weighted  
Sum**

**Loss**

**L2 Loss**

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

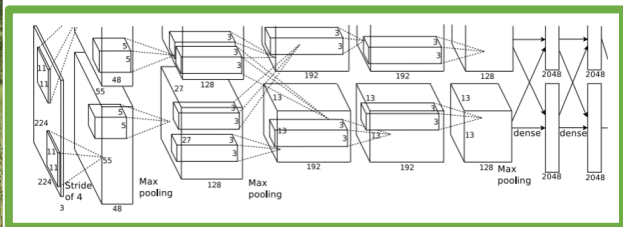
# Detecting a single object

Often pretrained  
on ImageNet  
(Transfer learning)



[This image is CC0 public domain](#)

Treat localization as a  
regression problem!



**Vector:**  
4096

“Where”

Fully  
Connected:  
4096 to 1000

“What”

**Class Scores**

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

**Correct label:**

Cat

**Softmax  
Loss**

Multitask  
Loss

**Weighted  
Sum**

**Loss**

Fully  
Connected:  
4096 to 4

**Box  
Coordinates**  
(x, y, w, h)

**L2 Loss**

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

# Detecting a single object

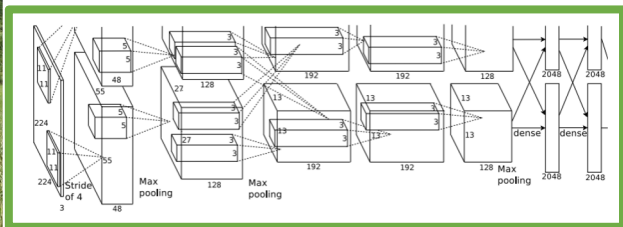
Often pretrained  
on ImageNet  
(Transfer learning)



[This image](#) is [CC0 public domain](#)

Treat localization as a  
regression problem!

**Problem:** Images can have  
more than one object!



**Vector:**  
4096

“Where”

“What”

Fully  
Connected:  
4096 to 1000

**Class Scores**

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

Fully  
Connected:  
4096 to 4

**Box  
Coordinates**  
(x, y, w, h)

**Correct label:**  
Cat

**Softmax  
Loss**

Multitask  
Loss

**Weighted  
Sum**

**Loss**

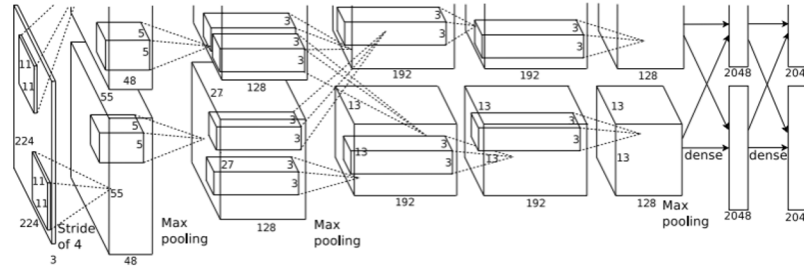
**L2 Loss**

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



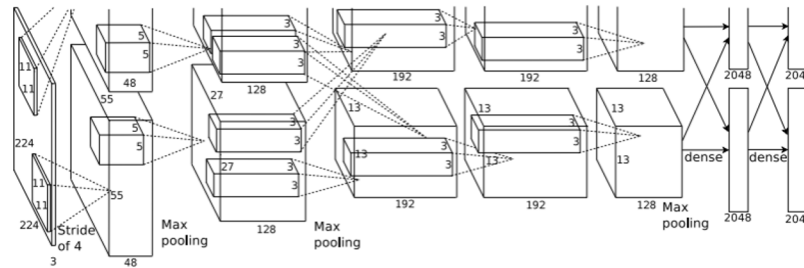
# Detecting Multiple Objects

Need different numbers  
of outputs per image



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

4 numbers

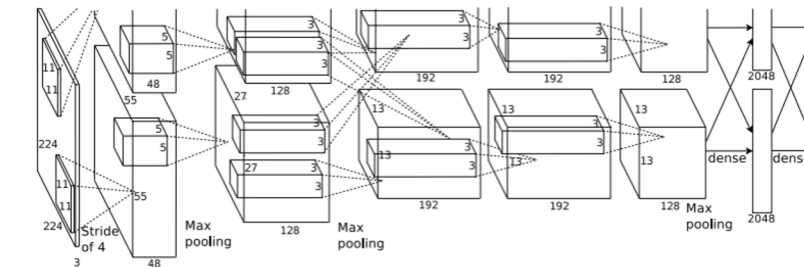


DOG:  $(x, y, w, h)$

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

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

16 numbers



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

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

....

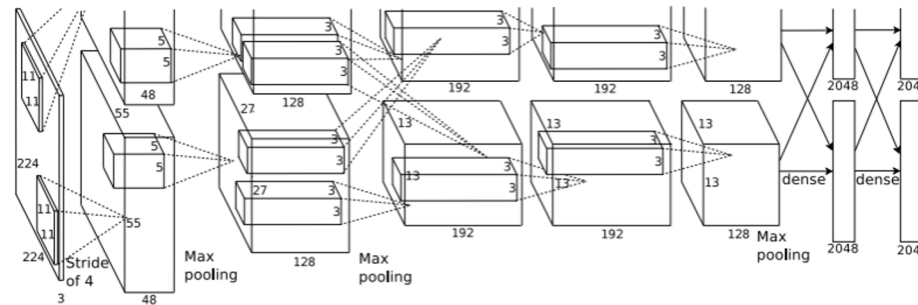
Many  
numbers!

[Duck image](#) is free to use under the [Pixabay license](#)



# Detecting Multiple Objects: Sliding Window

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



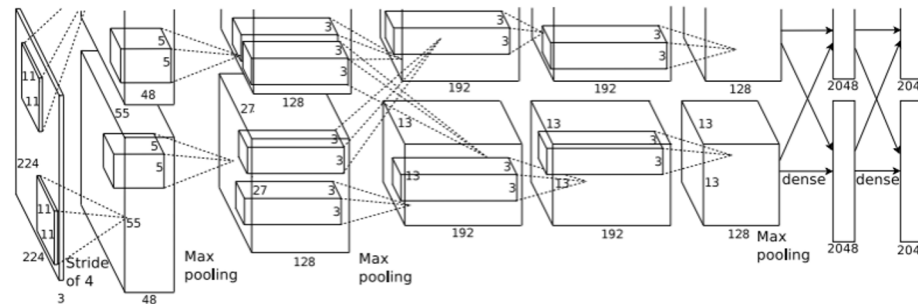
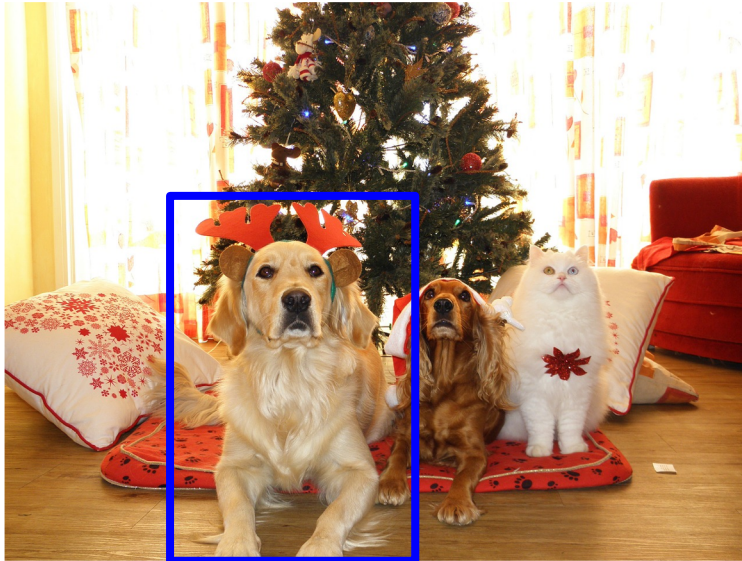
Dog? **NO**

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



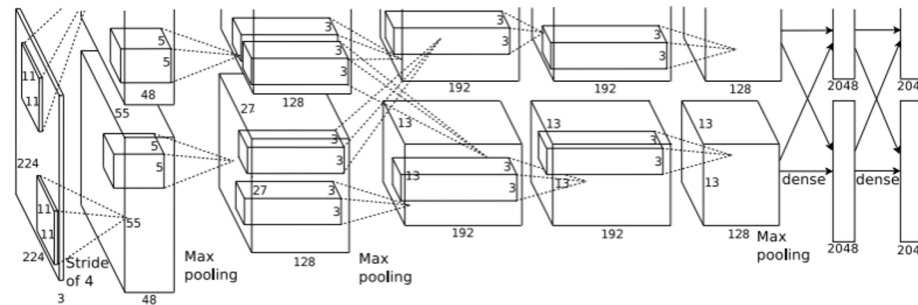
Dog? **YES**

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



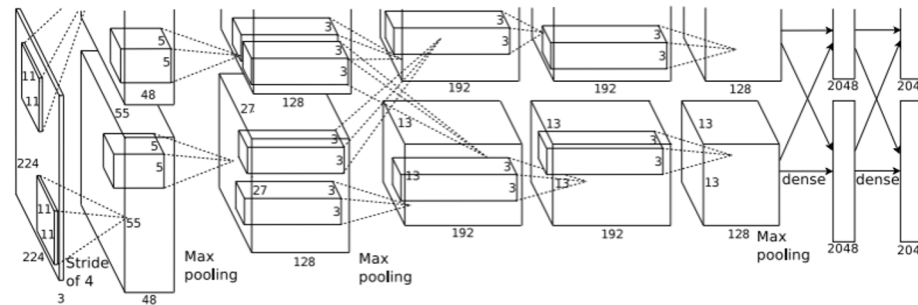
Dog? YES

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

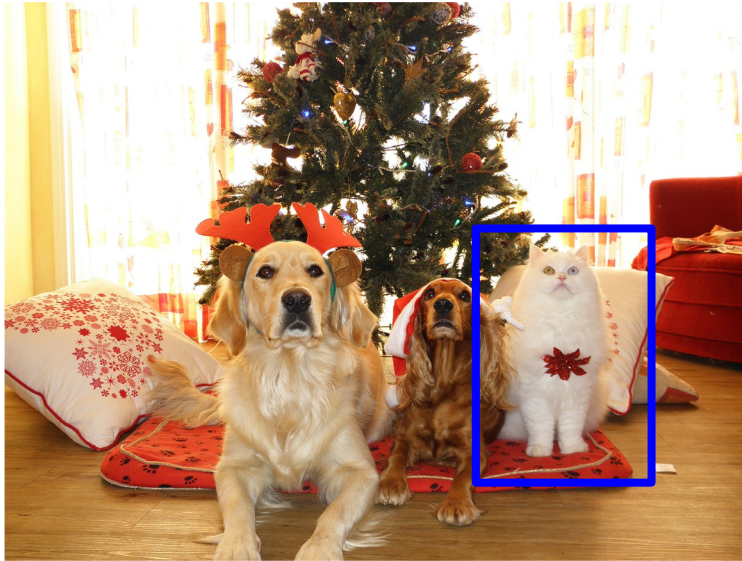
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

**Question:** How many possible boxes are there in an image of size  $H \times W$ ?



# Detecting Multiple Objects: Sliding Window

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



**Question:** How many possible boxes are there in an image of size  $H \times W$ ?

Consider a box of size  $h \times w$ :

Possible x positions:  $W - w + 1$

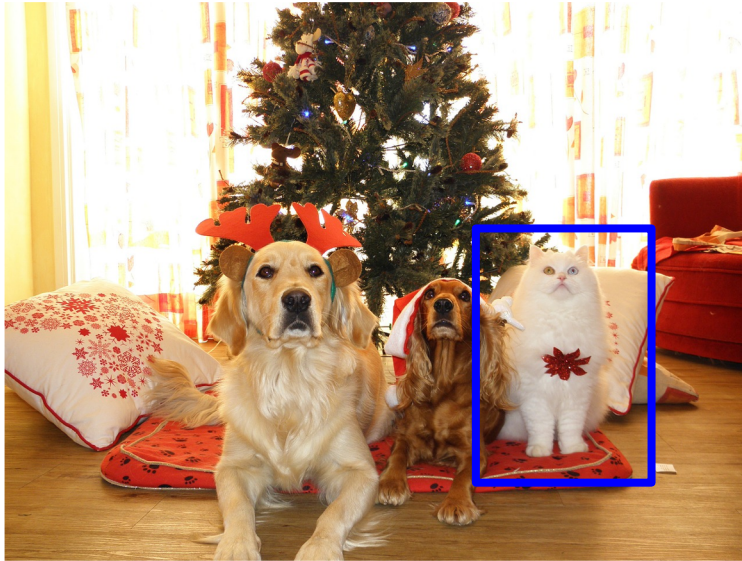
Possible y positions:  $H - h + 1$

Possible positions:

$$(W - w + 1) * (H - h + 1)$$

# Detecting Multiple Objects: Sliding Window

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



**Question:** How many possible boxes are there in an image of size  $H \times W$ ?

Consider a box of size  $h \times w$ :

Possible x positions:  $W - w + 1$

Possible y positions:  $H - h + 1$

Possible positions:

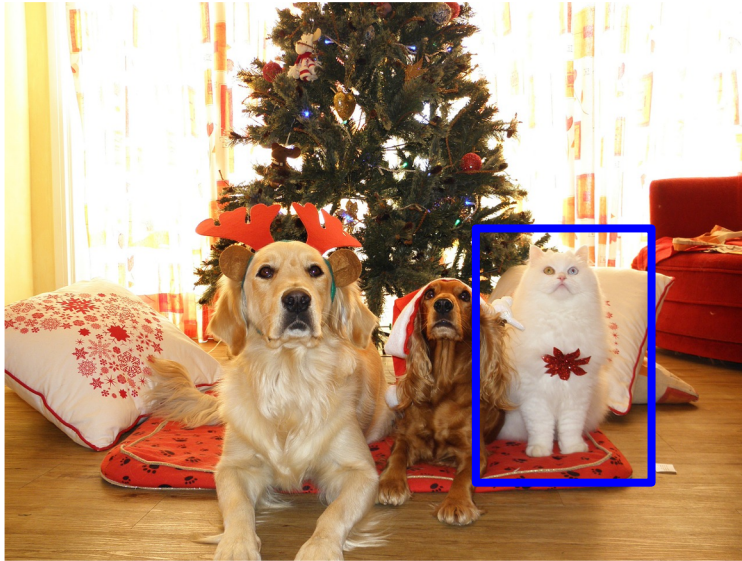
$$(W - w + 1) * (H - h + 1)$$

Total possible boxes:

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

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

# 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  $H \times W$ ?

Consider a box of size  $h \times w$ :

Possible x positions:  $W - w + 1$

Possible y positions:  $H - h + 1$

Possible positions:

$$(W - w + 1) * (H - h + 1)$$

Total possible boxes:

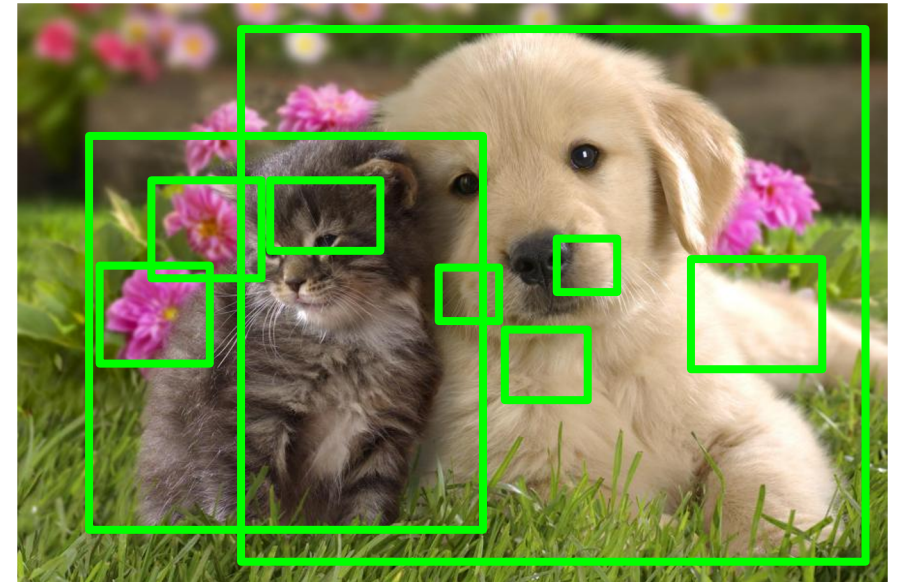
$$\sum_{h=1}^H \sum_{w=1}^W (W - w + 1)(H - h + 1)$$

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



# 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



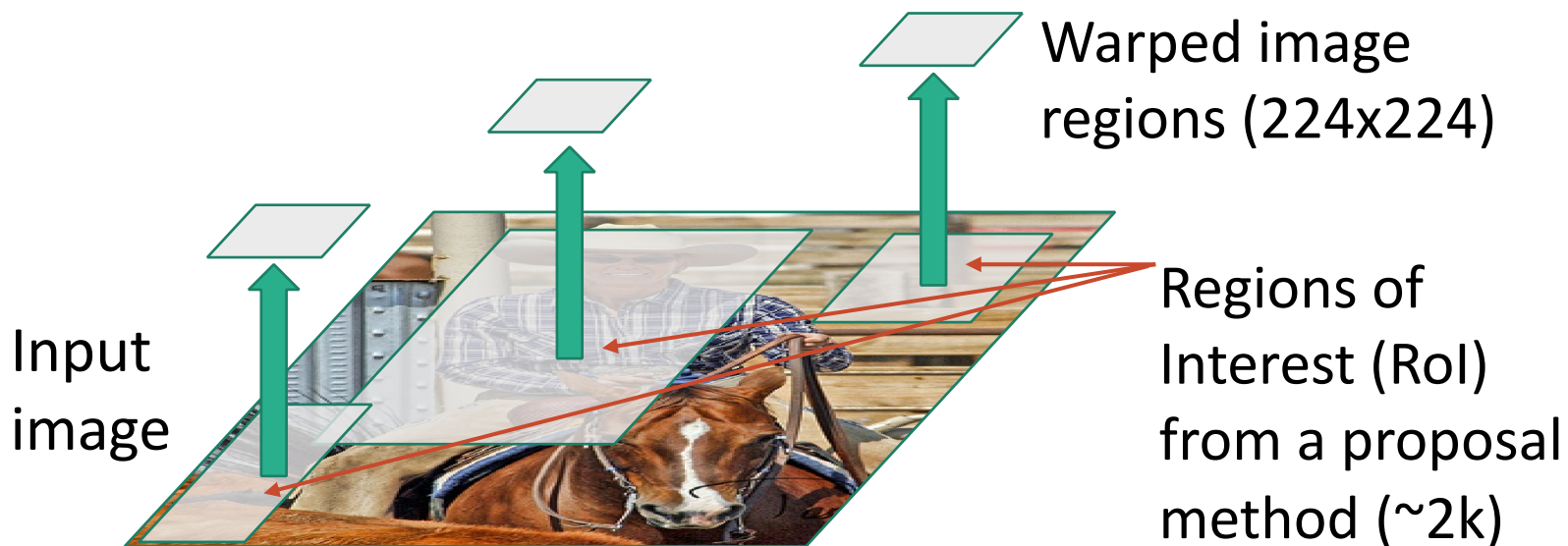
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.  
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

# R-CNN: Region-Based CNN



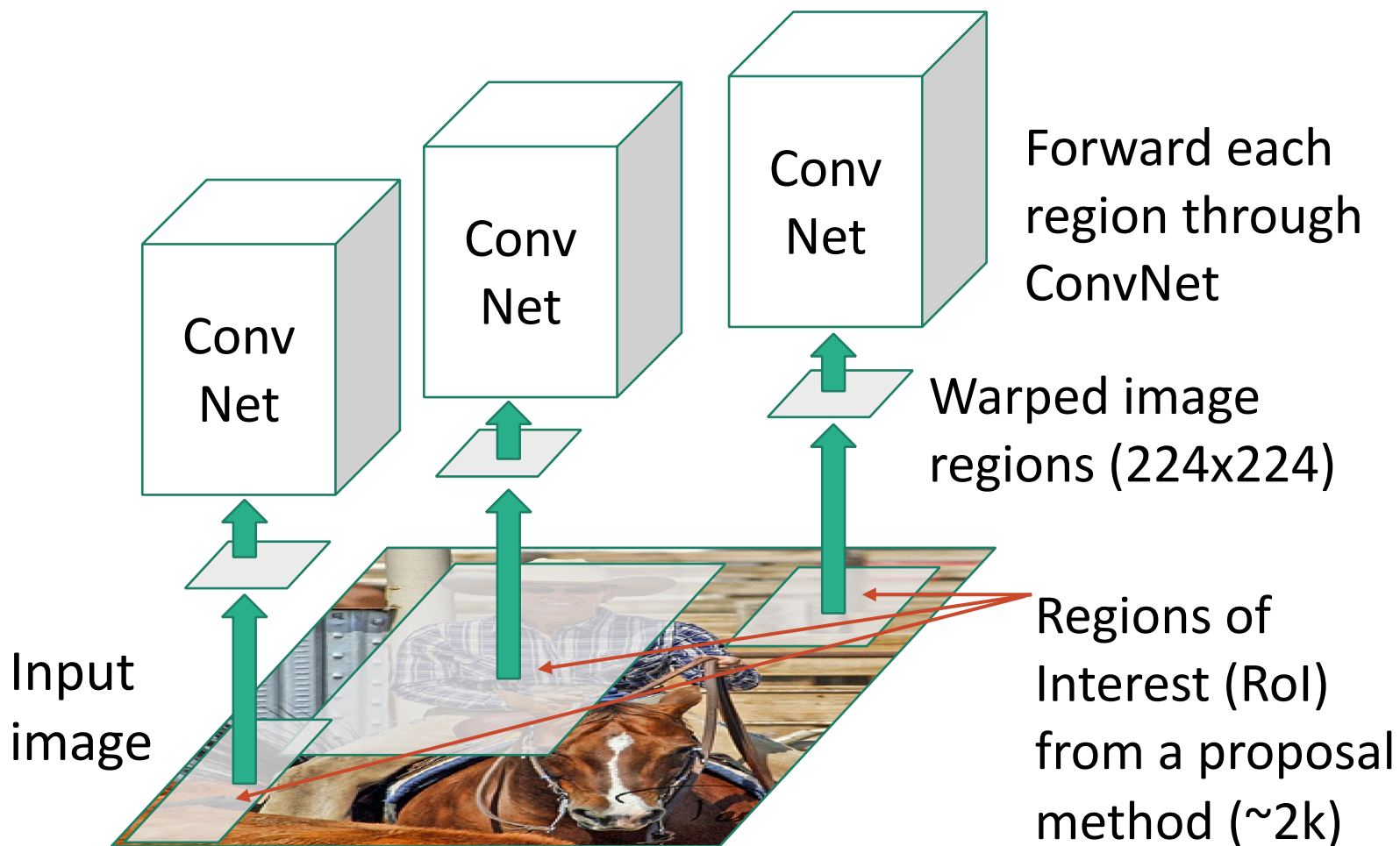
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.  
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

# R-CNN: Region-Based CNN



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

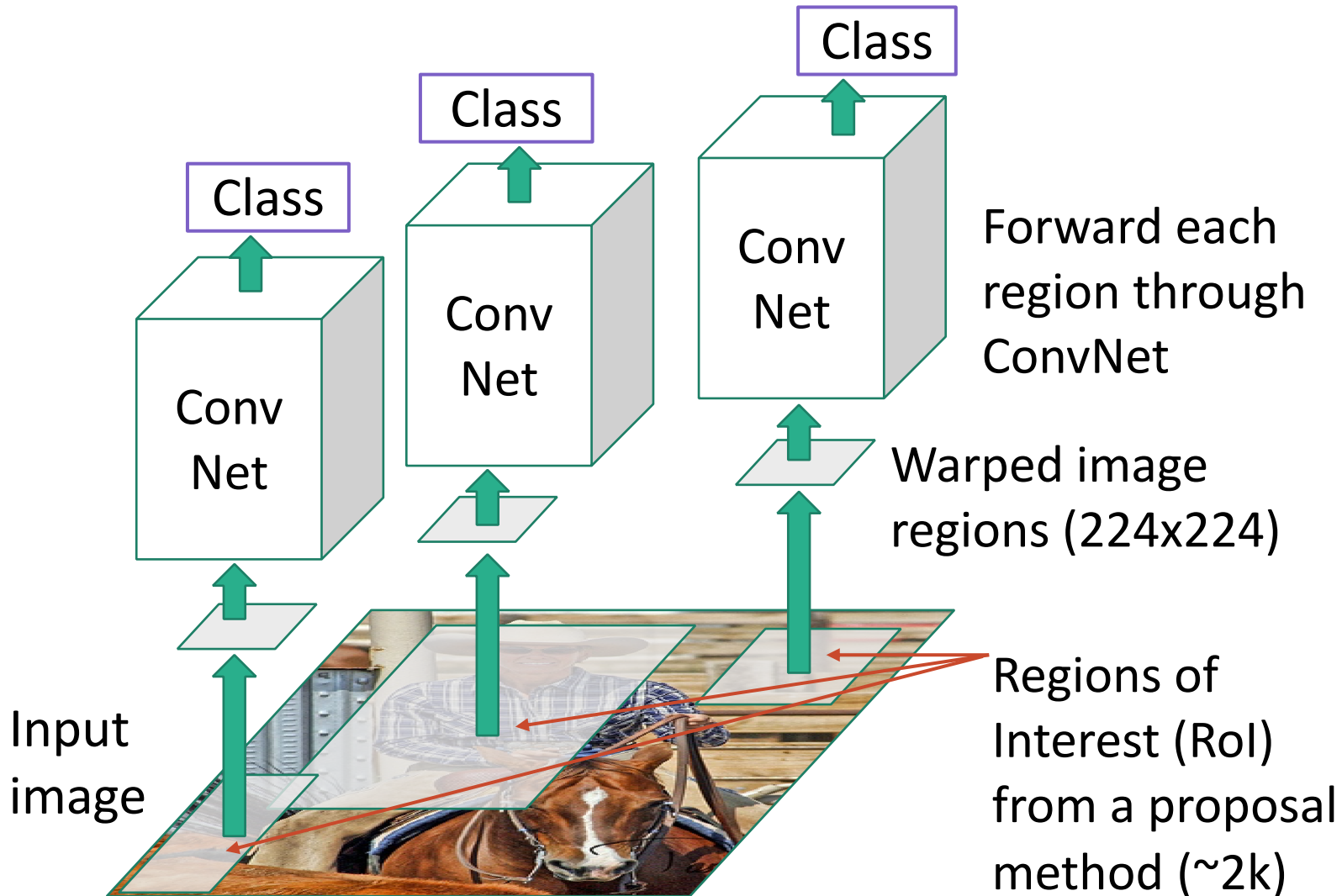
# R-CNN: Region-Based CNN



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

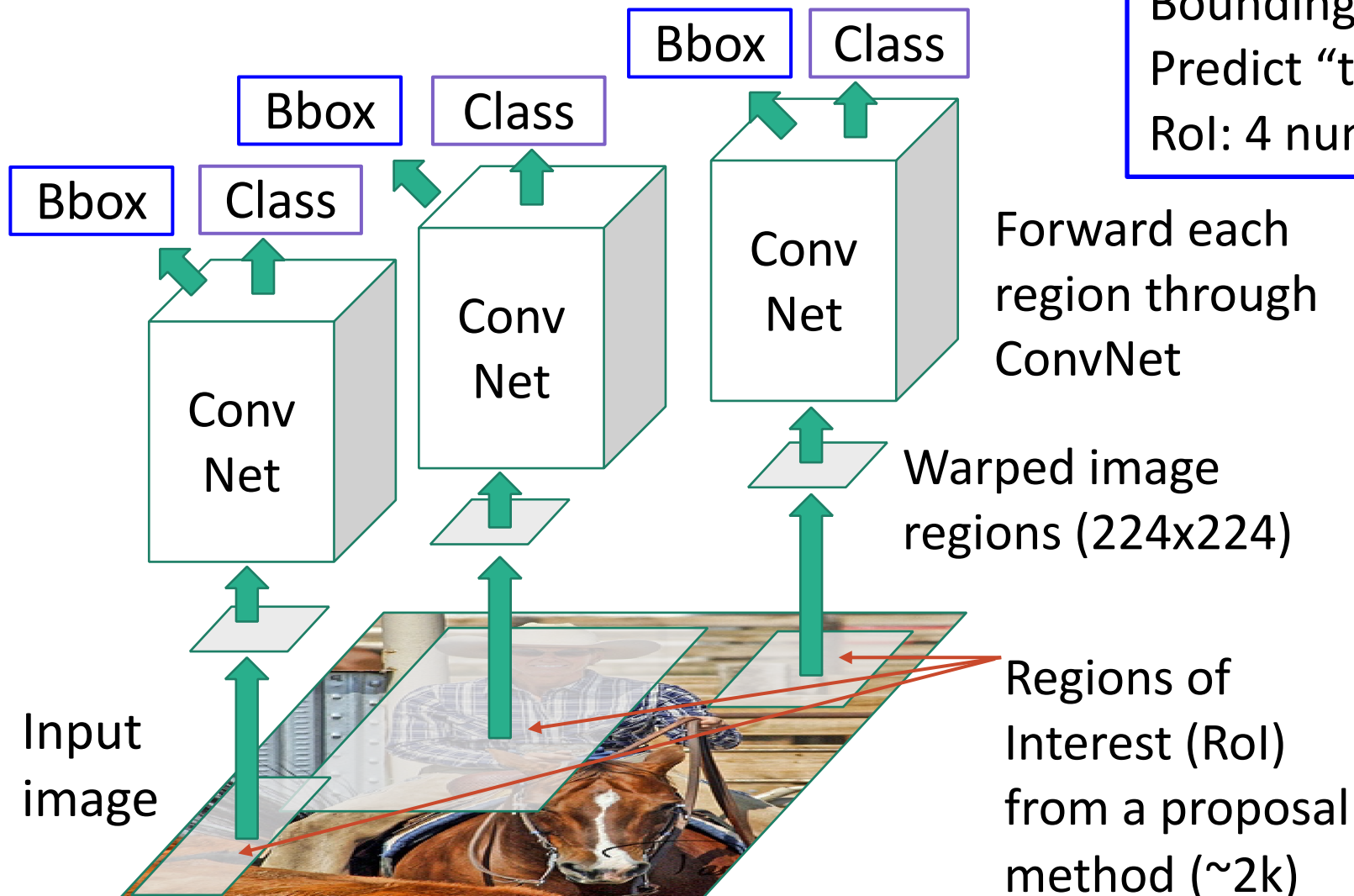
Classify each region



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 ( $t_x, t_y, t_h, t_w$ )

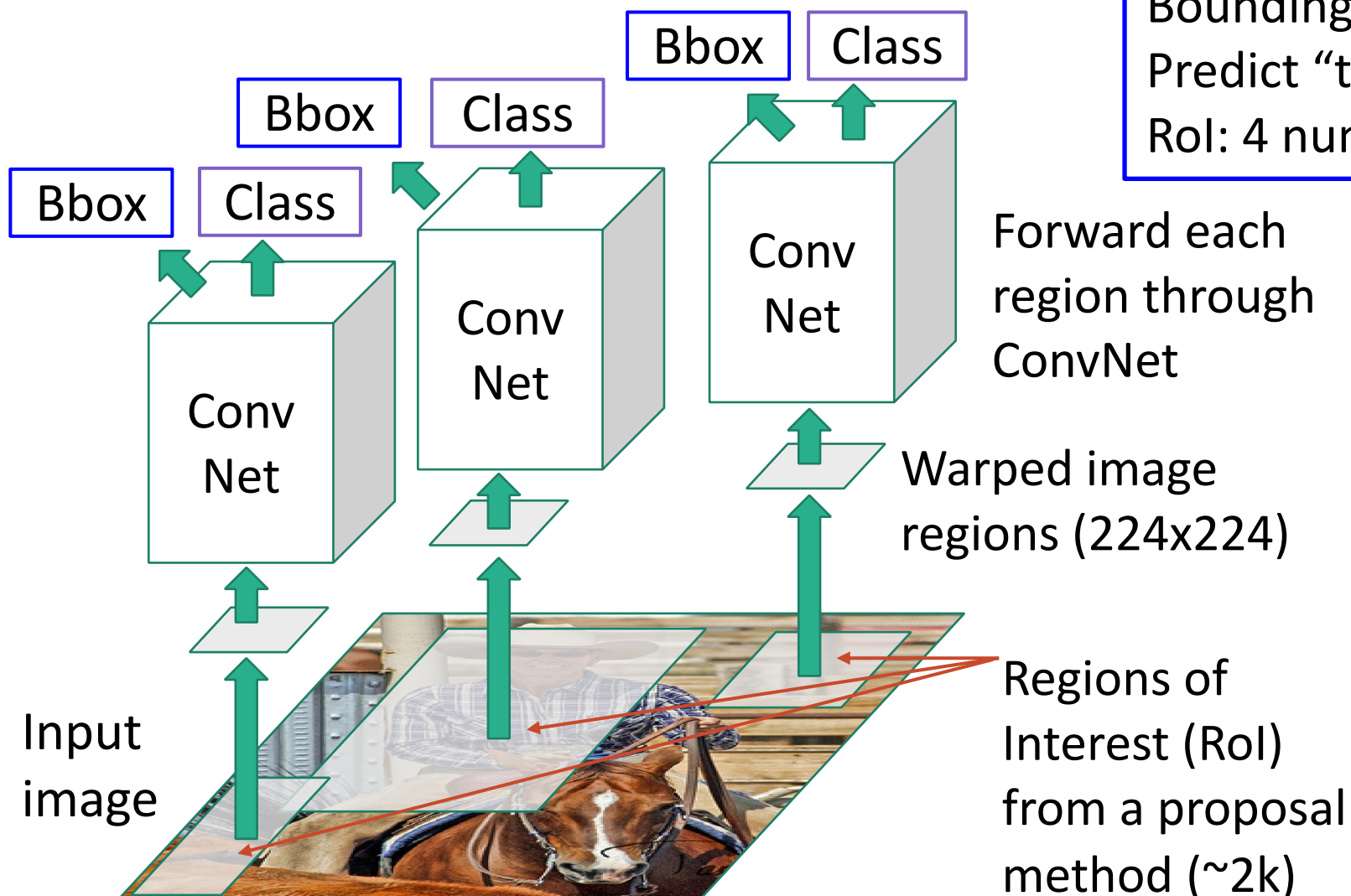
Forward each  
region through  
ConvNet

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



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: ( $b_x, b_y, b_h, b_w$ )

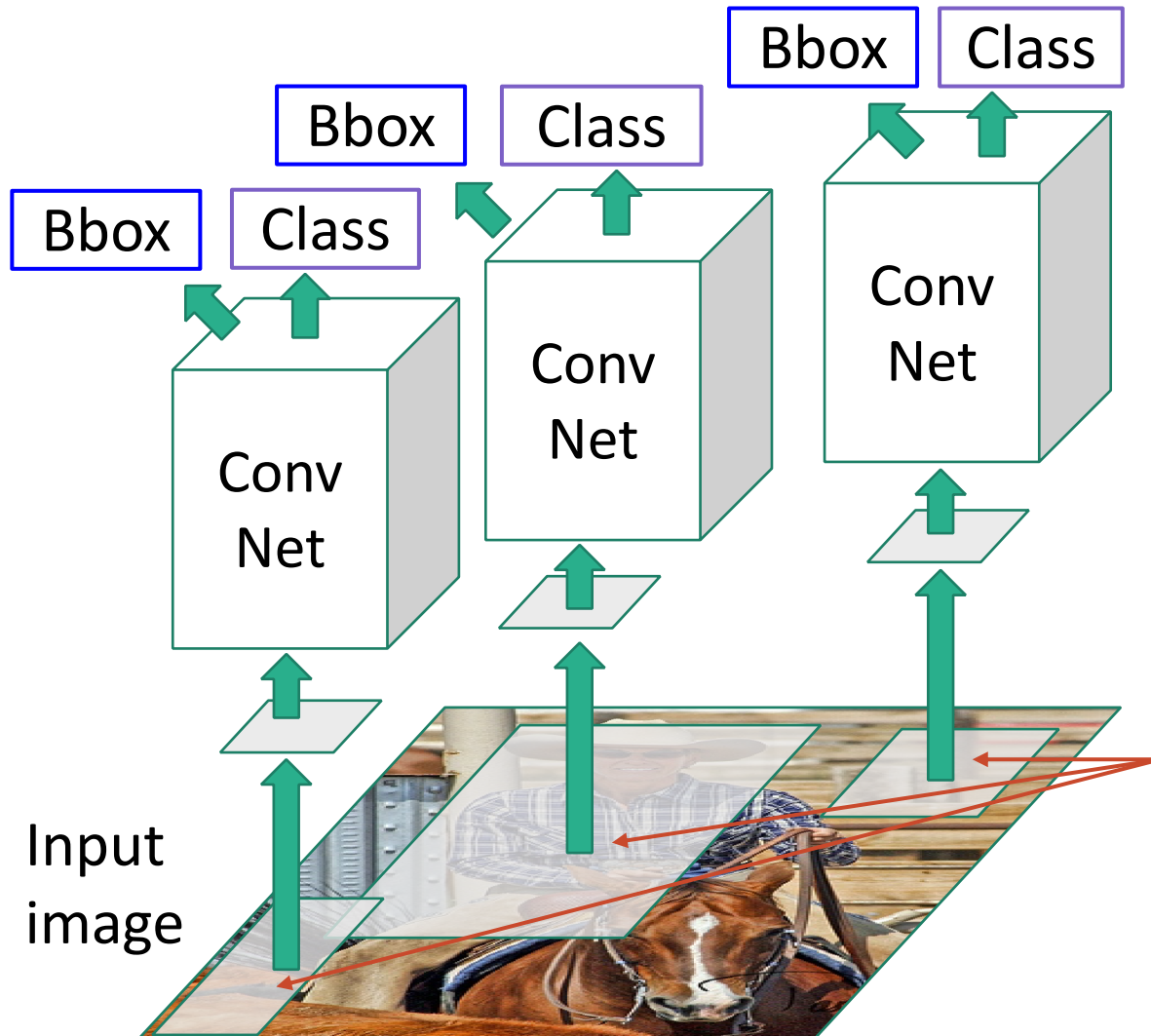
Translate relative to box size:  
 $b_x = p_x + p_w t_x$        $b_y = p_y + p_h t_y$

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

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: Test-time



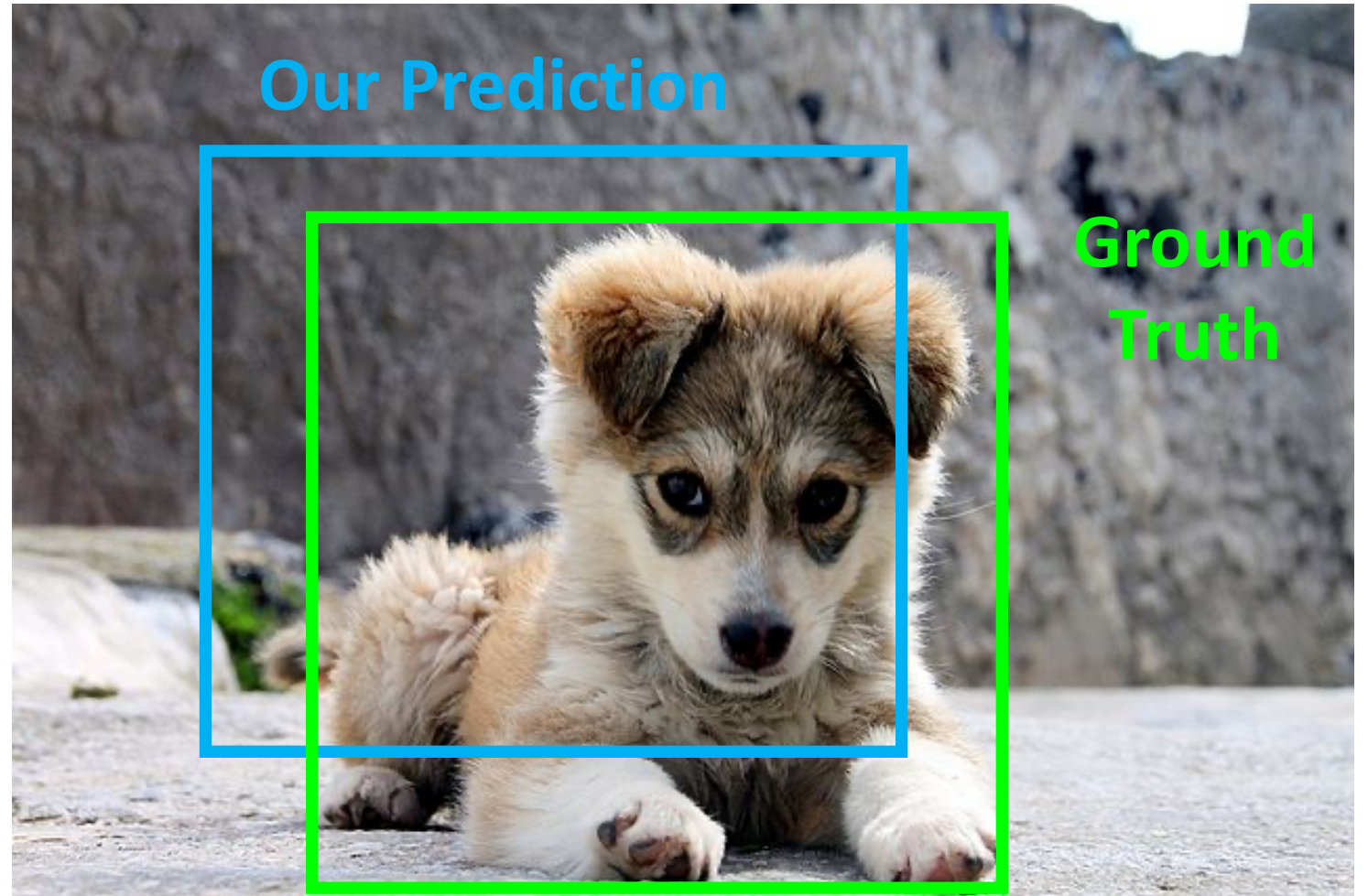
Input: Single RGB Image

1. Run region proposal method to compute  $\sim 2000$  region proposals
2. Resize each region to  $224 \times 224$  and run independently through CNN to predict class scores and bbox transform
3. Use scores to select a subset of region proposals to output  
(Many choices here: threshold on background, or per-category? Or take top K proposals per image?)
4. Compare with ground-truth boxes

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

# Comparing Boxes: Intersection over Union (IoU)

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



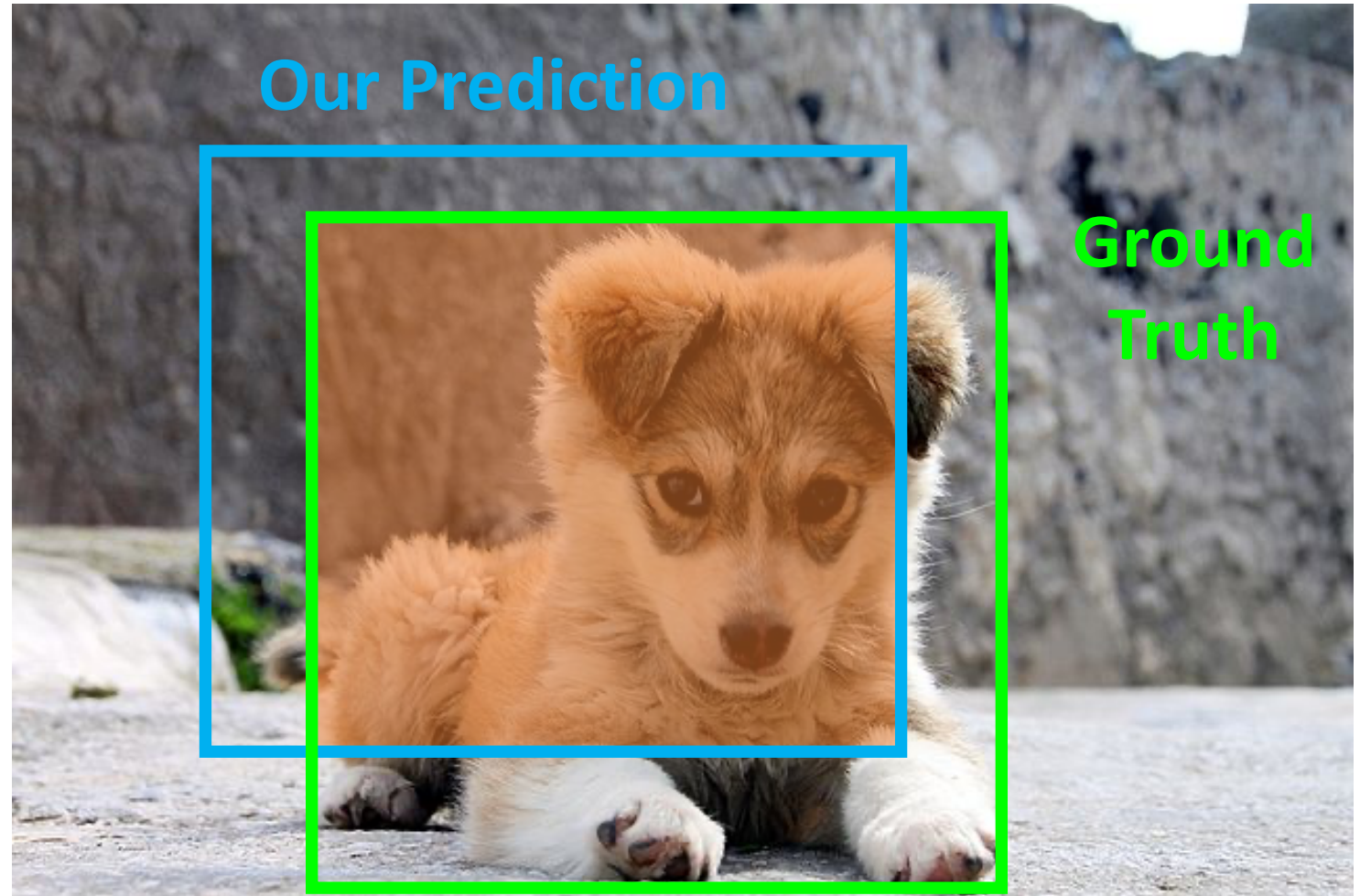
[Puppy image](#) is licensed under [CC-A 2.0 Generic license](#). Bounding boxes and text added by Justin Johnson.

# 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”):

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



[Puppy image](#) is licensed under [CC-A 2.0 Generic license](#). Bounding boxes and text added by Justin Johnson.

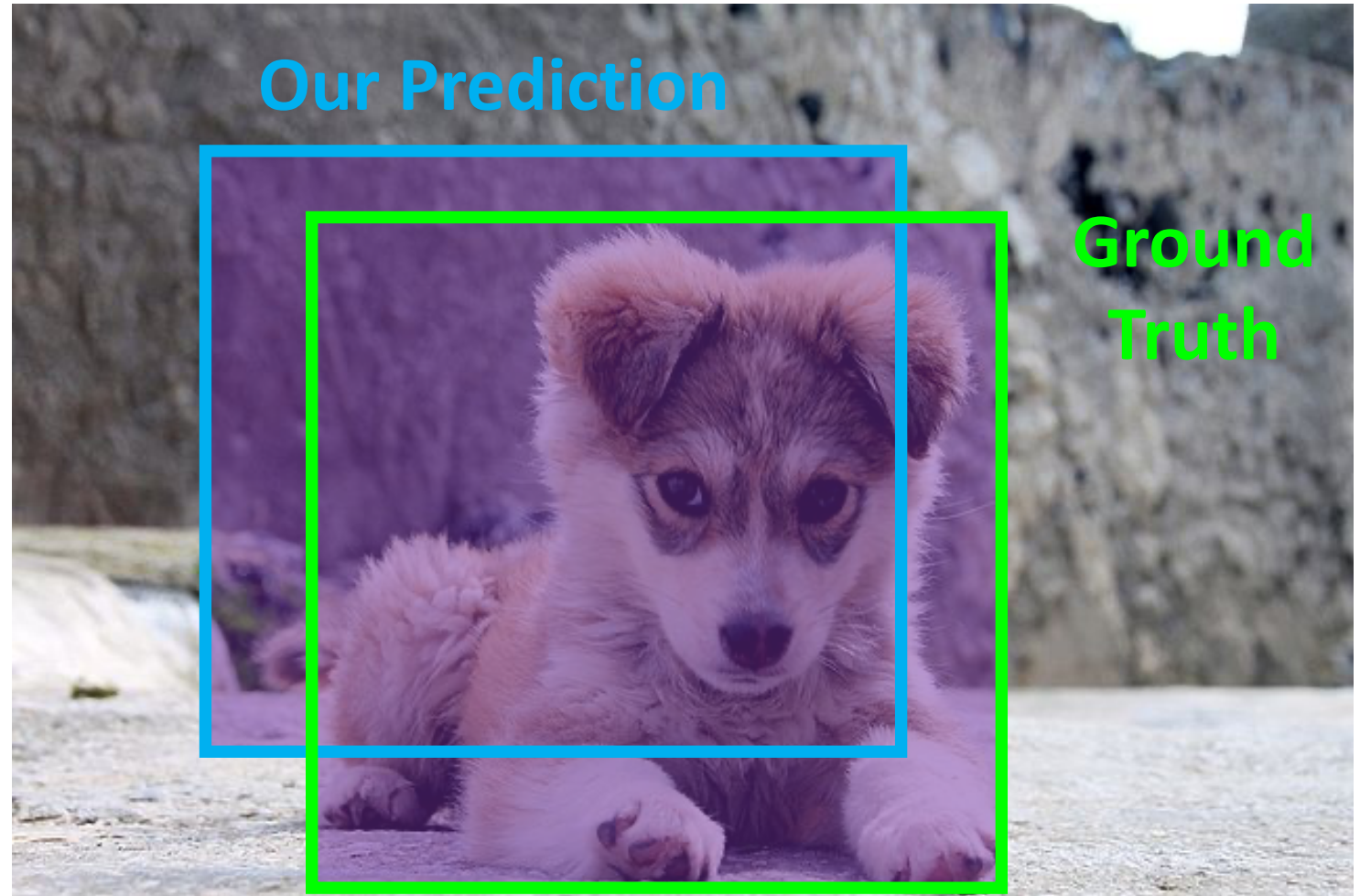


# Comparing Boxes: Intersection over Union (IoU)

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

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

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



[Puppy image](#) is licensed under [CC-A 2.0 Generic license](#). Bounding boxes and text added by Justin Johnson.

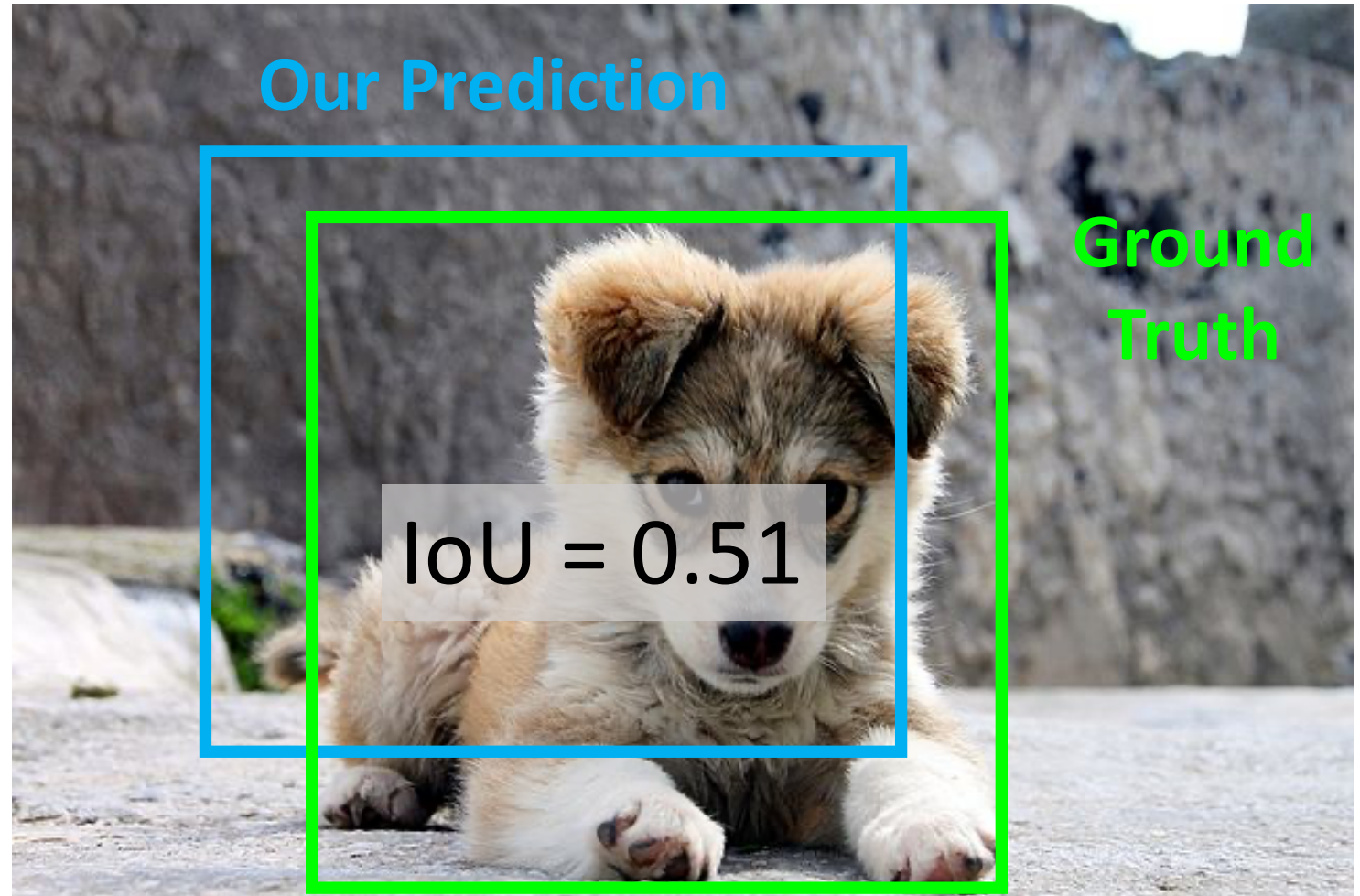
# Comparing Boxes: Intersection over Union (IoU)

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

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

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

$\text{IoU} > 0.5$  is “decent”



[Puppy image](#) is licensed under [CC-A 2.0 Generic license](#). Bounding boxes and text added by Justin Johnson.



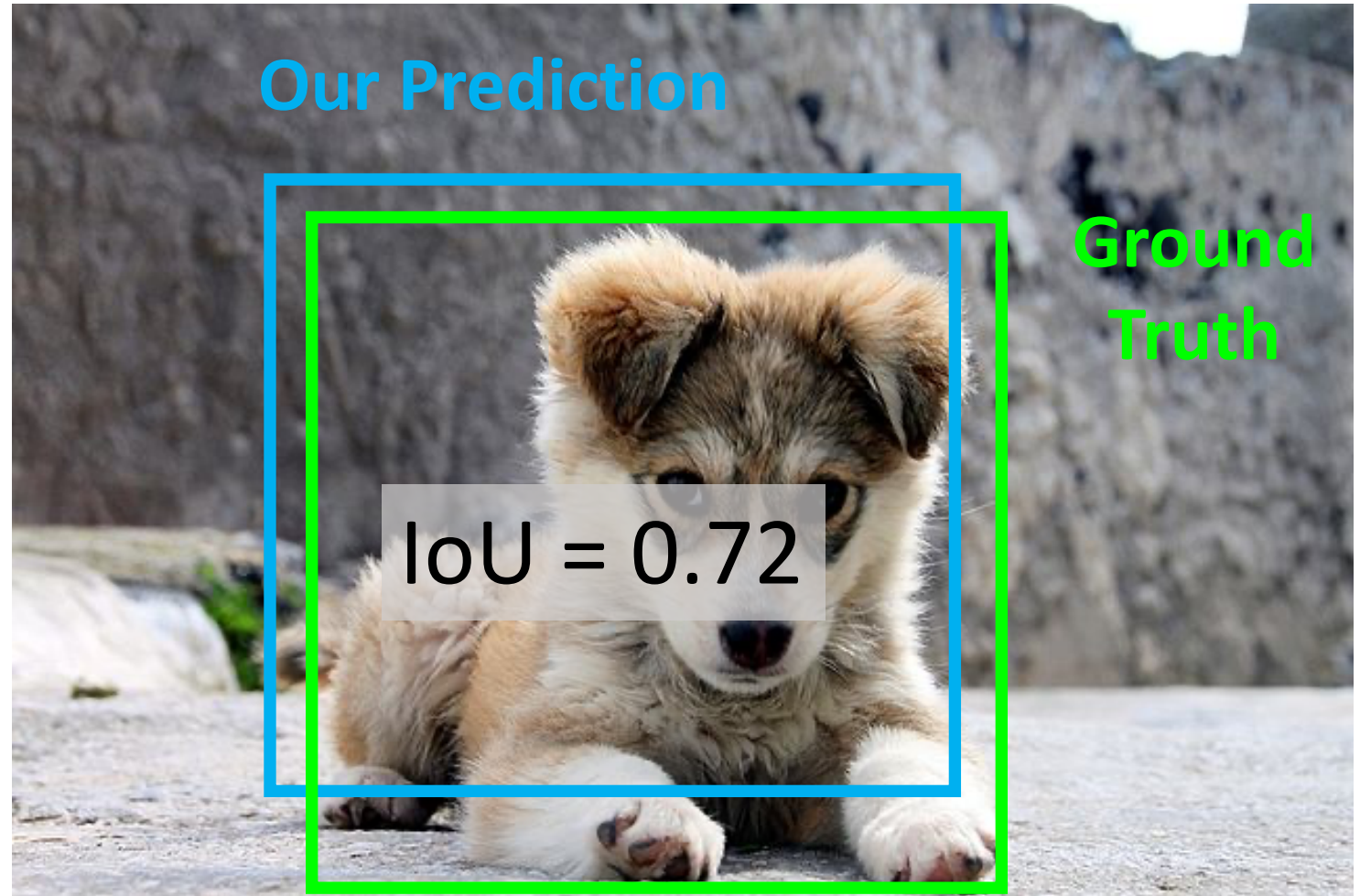
# Comparing Boxes: Intersection over Union (IoU)

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

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

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

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



[Puppy image](#) is licensed under [CC-A 2.0 Generic license](#). Bounding boxes and text added by Justin Johnson.

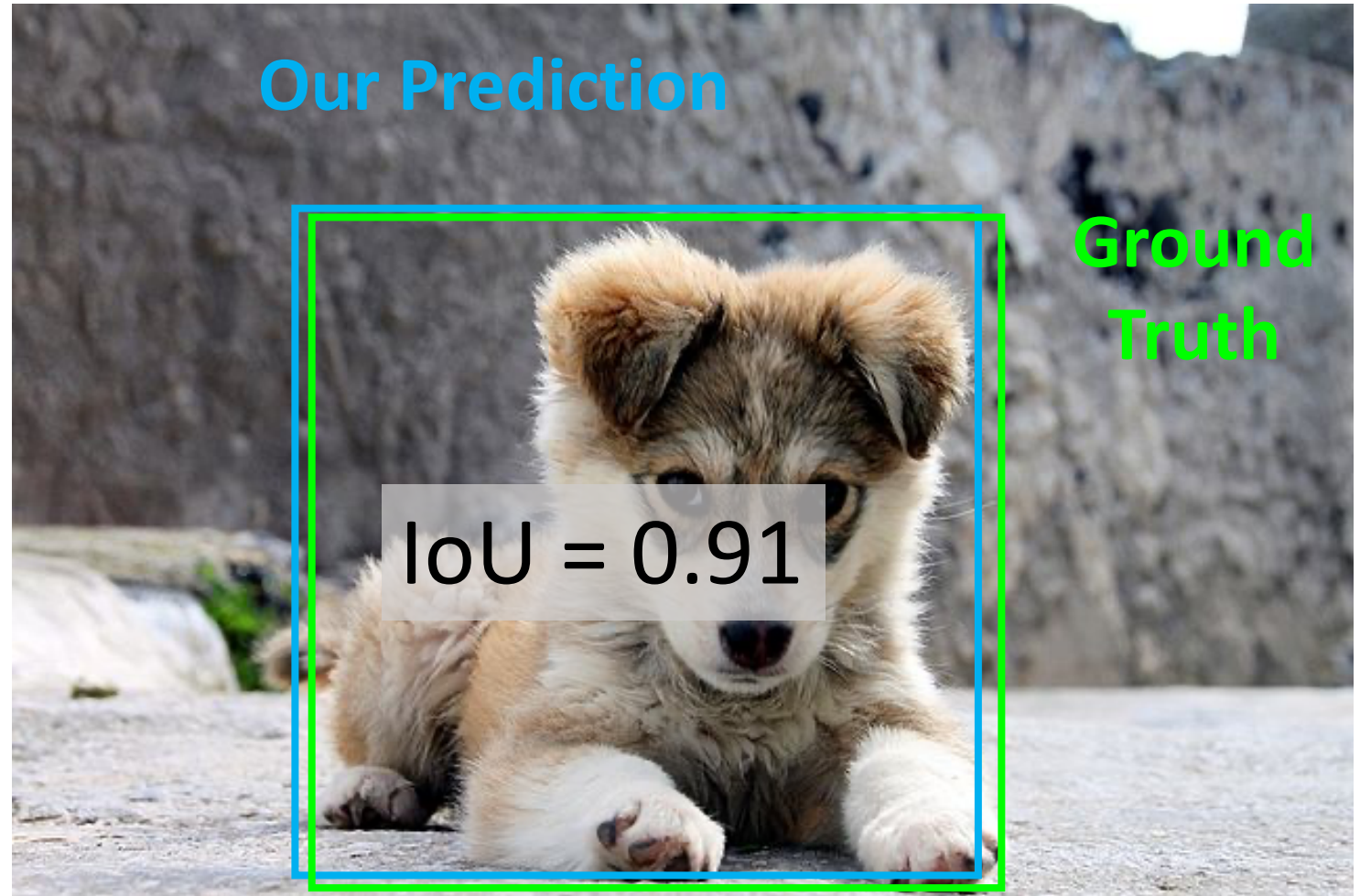
# Comparing Boxes: Intersection over Union (IoU)

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

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

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

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

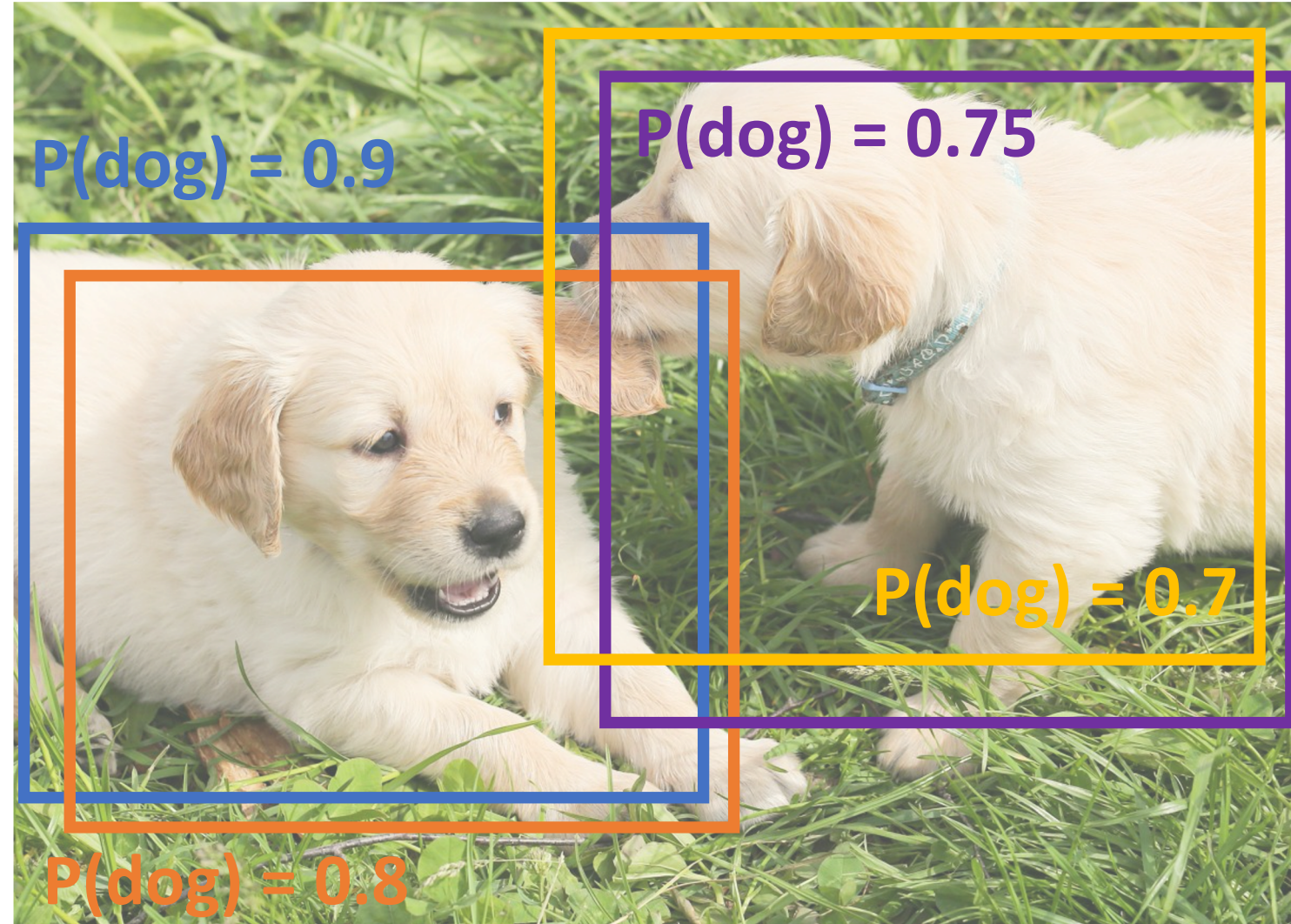


[Puppy image](#) is licensed under [CC-A 2.0 Generic license](#). Bounding boxes and text added by Justin Johnson.



# Overlapping Boxes

**Problem:** Object detectors often output many overlapping detections:



Puppy image is [CC0 Public Domain](#)

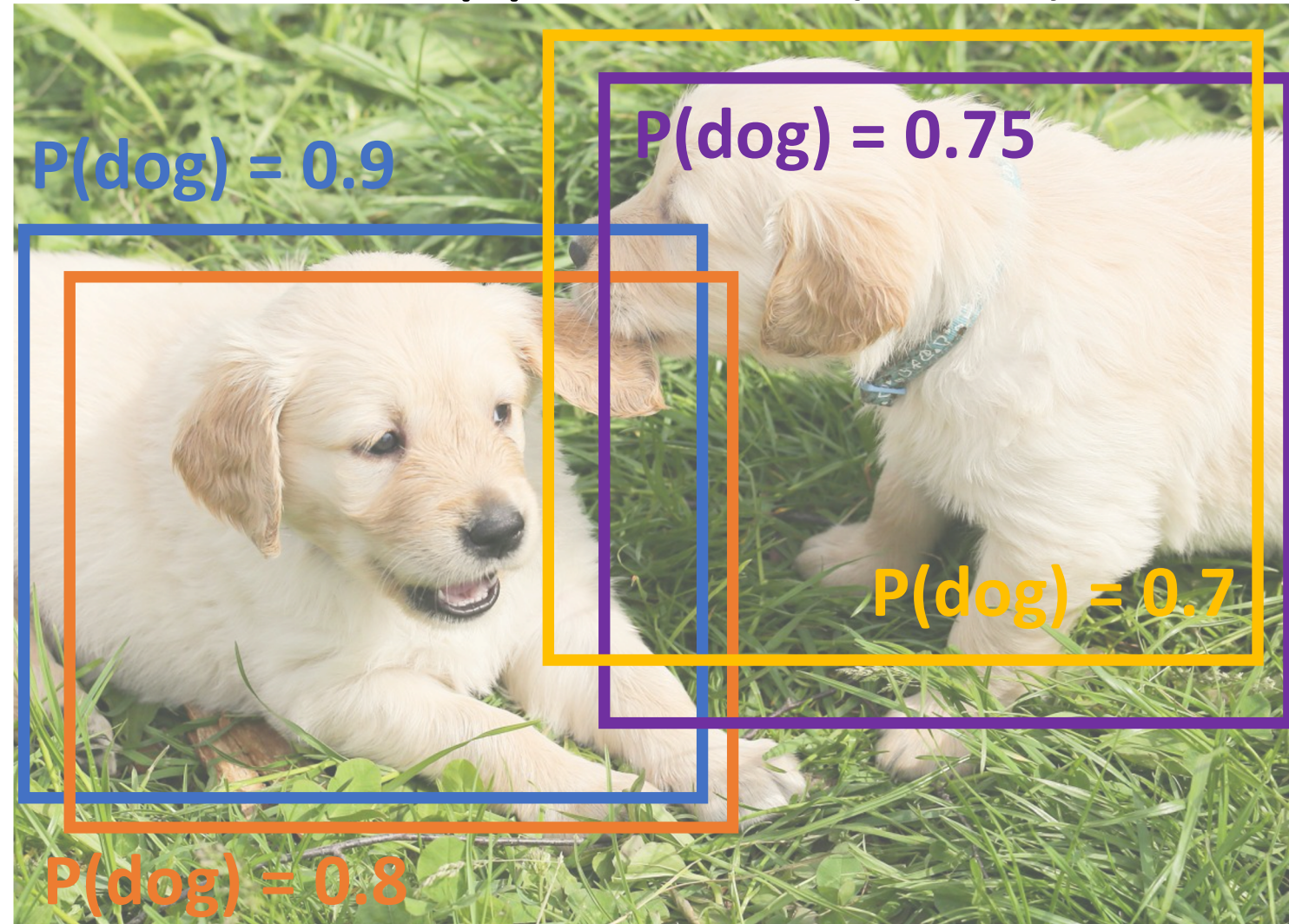


# Overlapping Boxes: Non-Max Suppression (NMS)

**Problem:** Object detectors often output many overlapping detections:

**Solution:** Post-process raw detections using **Non-Max Suppression (NMS)**

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with  $\text{IoU} > \text{threshold}$  (e.g. 0.7)
3. If any boxes remain, GOTO 1



Puppy image is [CC0 Public Domain](#)



# Overlapping Boxes: Non-Max Suppression (NMS)

**Problem:** Object detectors often output many overlapping detections:

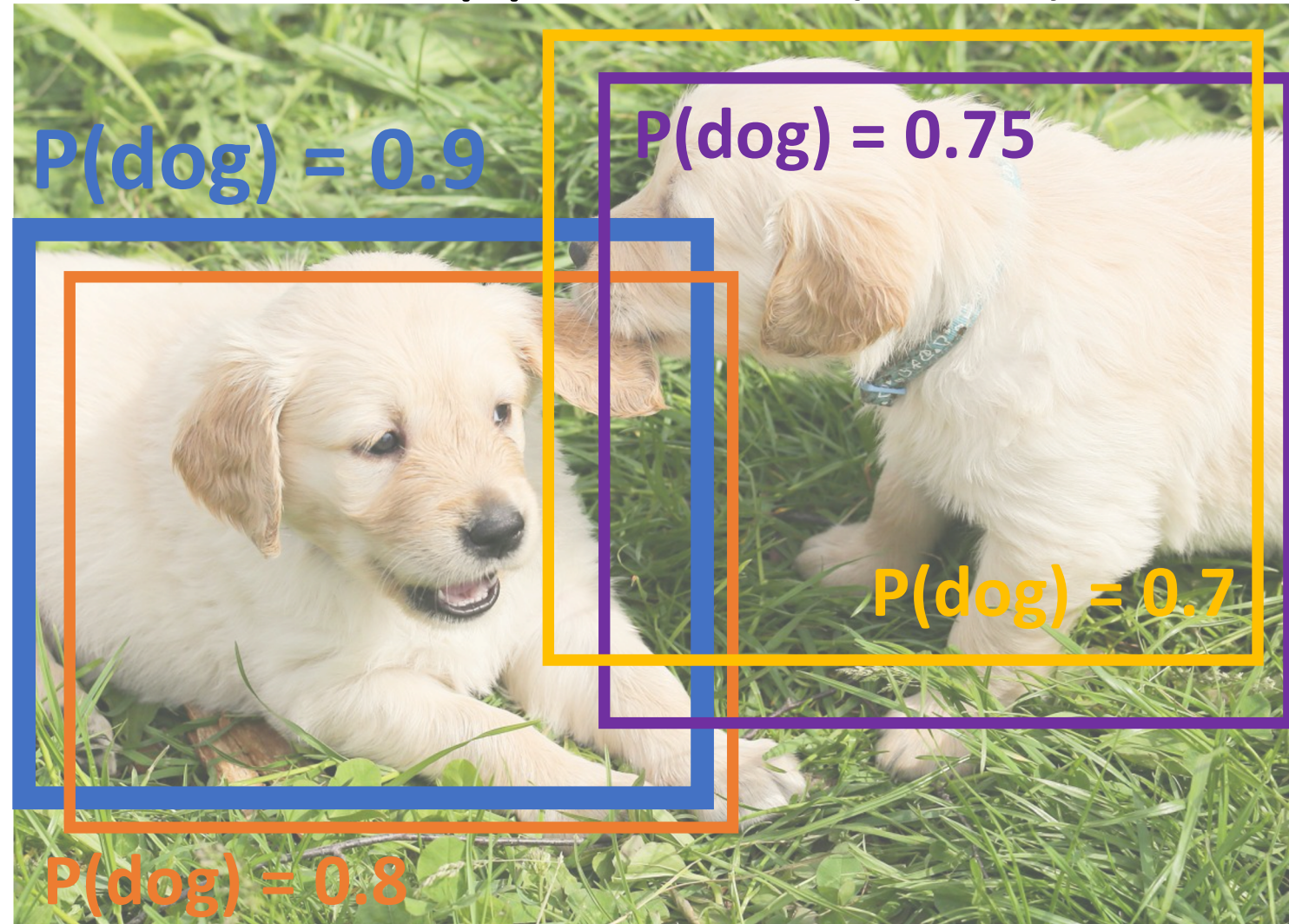
**Solution:** Post-process raw detections using **Non-Max Suppression (NMS)**

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with  $\text{IoU} > \text{threshold}$  (e.g. 0.7)
3. If any boxes remain, GOTO 1

$$\text{IoU}(\text{blue box}, \text{orange box}) = \mathbf{0.78}$$

$$\text{IoU}(\text{blue box}, \text{purple box}) = 0.05$$

$$\text{IoU}(\text{blue box}, \text{yellow box}) = 0.07$$



Puppy image is [CC0 Public Domain](#)



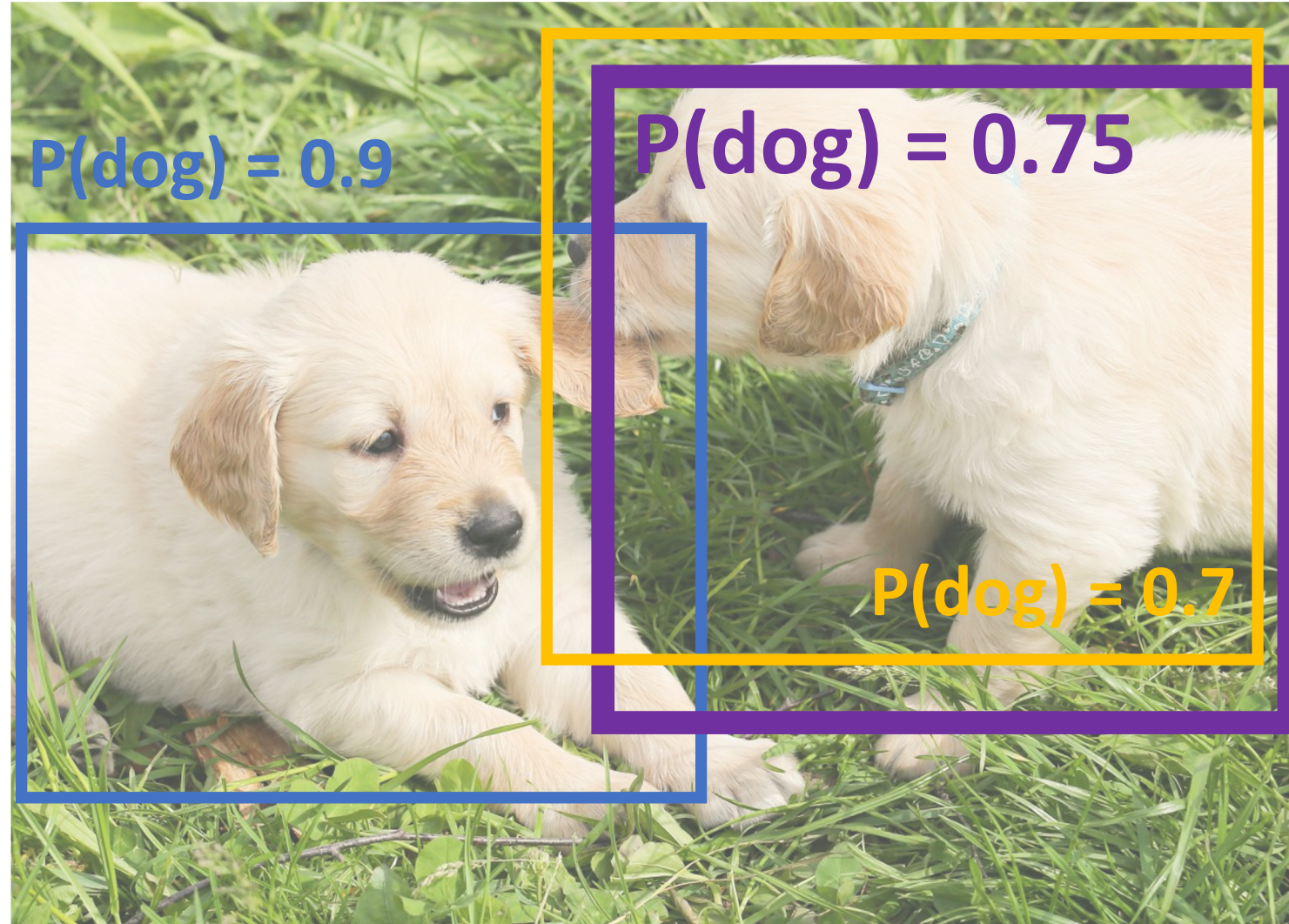
# Overlapping Boxes: Non-Max Suppression (NMS)

**Problem:** Object detectors often output many overlapping detections:

**Solution:** Post-process raw detections using **Non-Max Suppression (NMS)**

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with  $\text{IoU} > \text{threshold}$  (e.g. 0.7)
3. If any boxes remain, GOTO 1

$$\text{IoU}(\blacksquare, \blacksquare) = \mathbf{0.74}$$



Puppy image is [CC0 Public Domain](#)

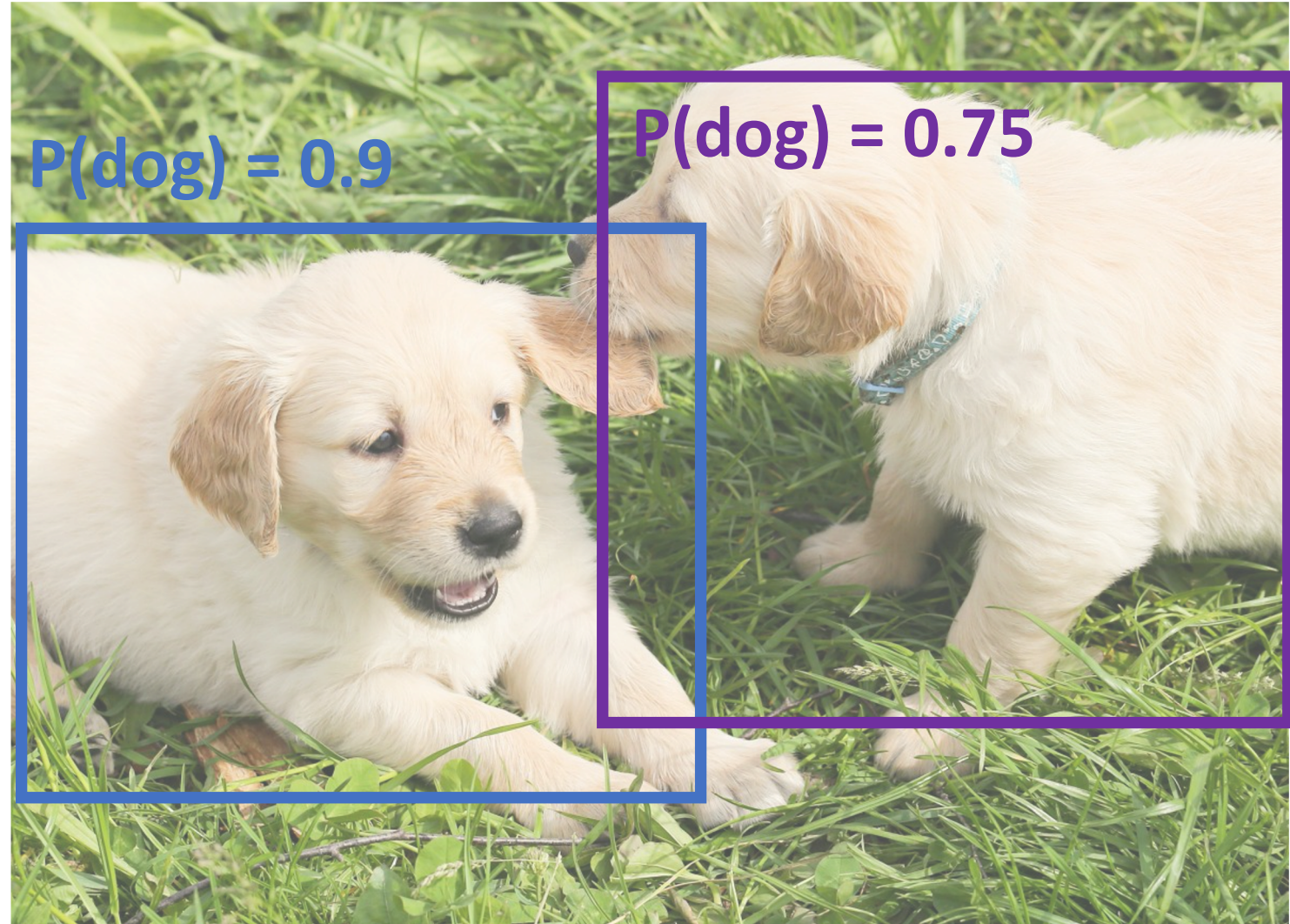


# Overlapping Boxes: Non-Max Suppression (NMS)

**Problem:** Object detectors often output many overlapping detections:

**Solution:** Post-process raw detections using **Non-Max Suppression (NMS)**

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with  $\text{IoU} > \text{threshold}$  (e.g. 0.7)
3. If any boxes remain, GOTO 1



Puppy image is [CC0 Public Domain](#)



# Overlapping Boxes: Non-Max Suppression (NMS)

**Problem:** Object detectors often output many overlapping detections:

**Solution:** Post-process raw detections using **Non-Max Suppression (NMS)**

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with  $\text{IoU} > \text{threshold}$  (e.g. 0.7)
3. If any boxes remain, GOTO 1

**Problem:** NMS may eliminate "good" boxes when objects are highly overlapping... no good solution =(



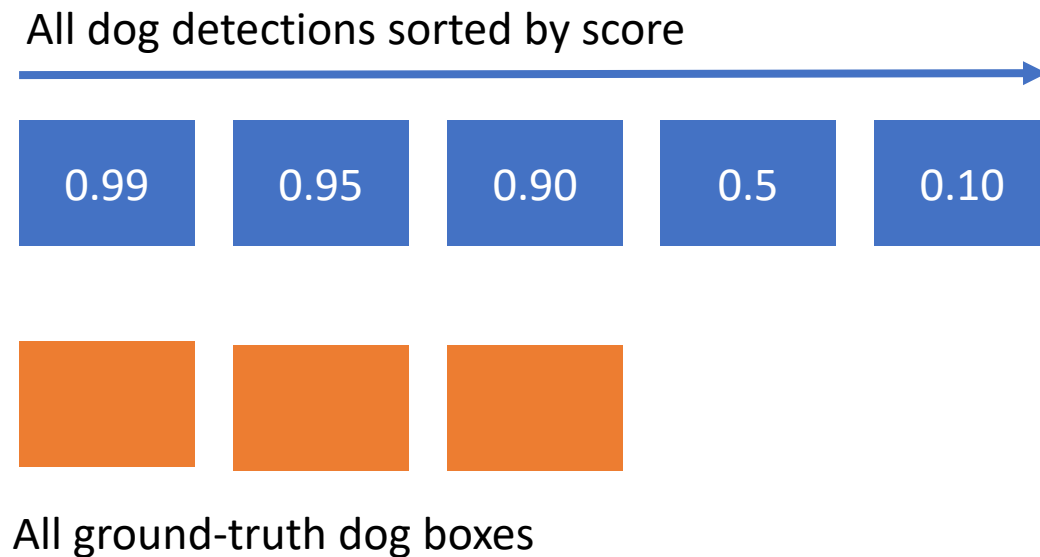
[Crowd image](#) is free for commercial use under the [Pixabay license](#)

# Evaluating Object Detectors: Mean Average Precision (mAP)

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

# Evaluating Object Detectors: Mean Average Precision (mAP)

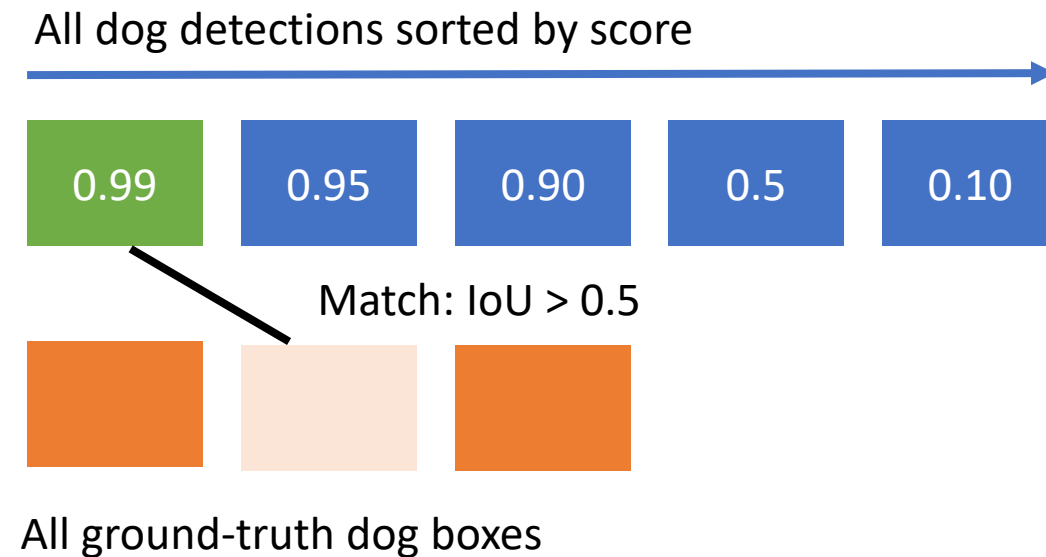
1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
  1. For each detection (highest score to lowest score)





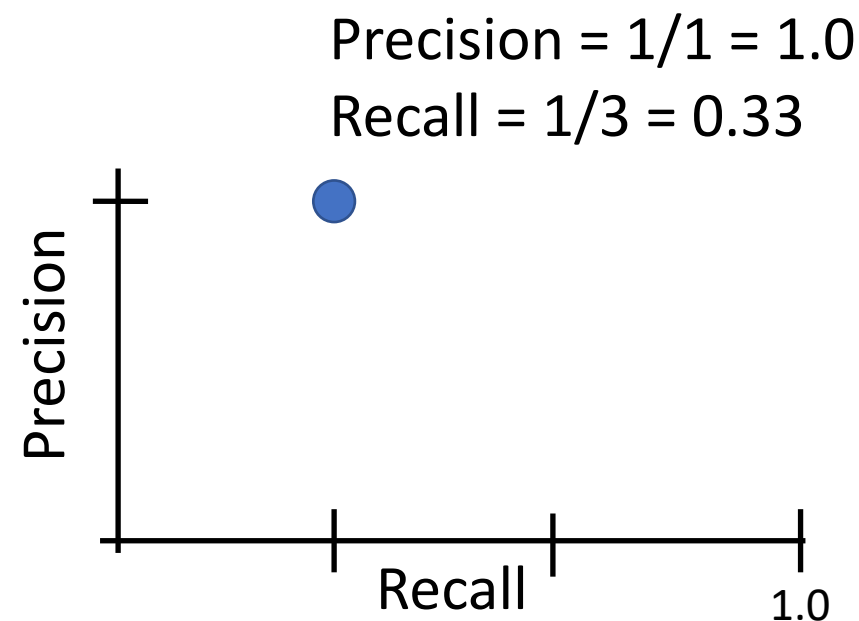
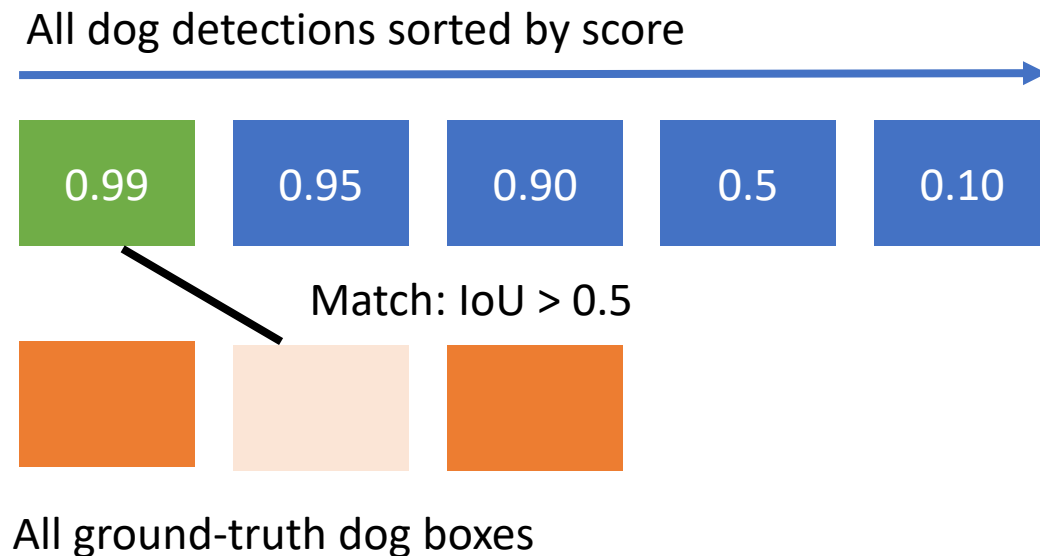
# Evaluating Object Detectors: Mean Average Precision (mAP)

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



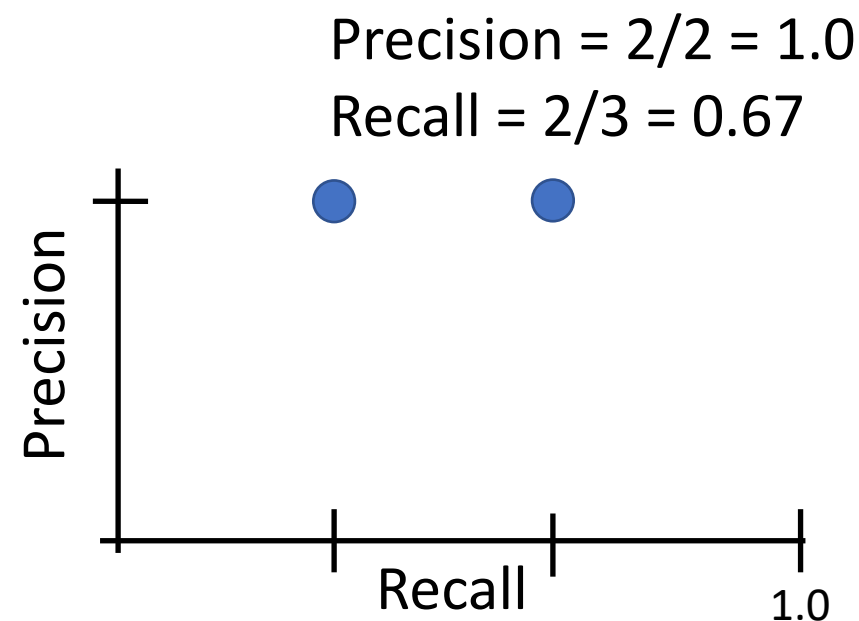
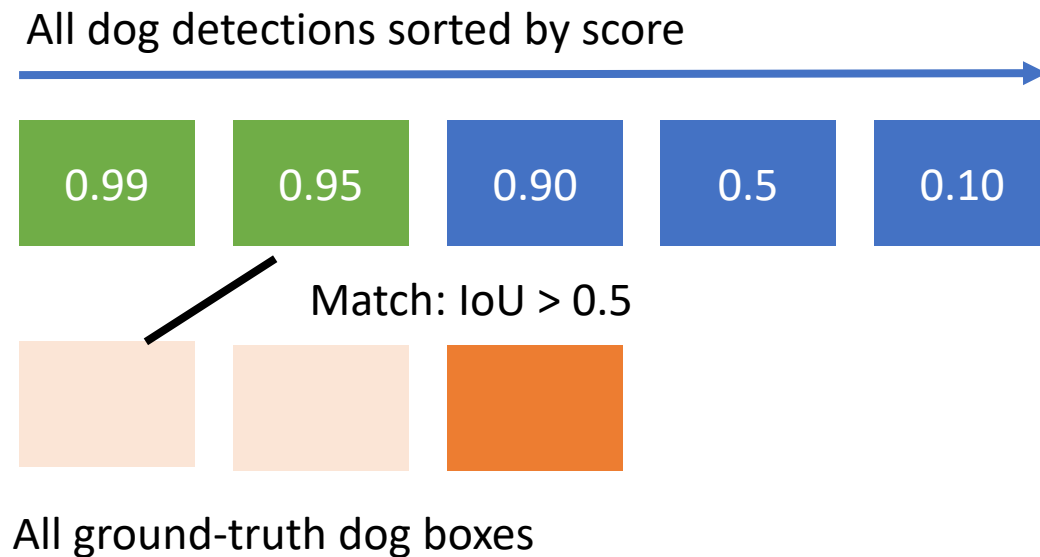
# Evaluating Object Detectors: Mean Average Precision (mAP)

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



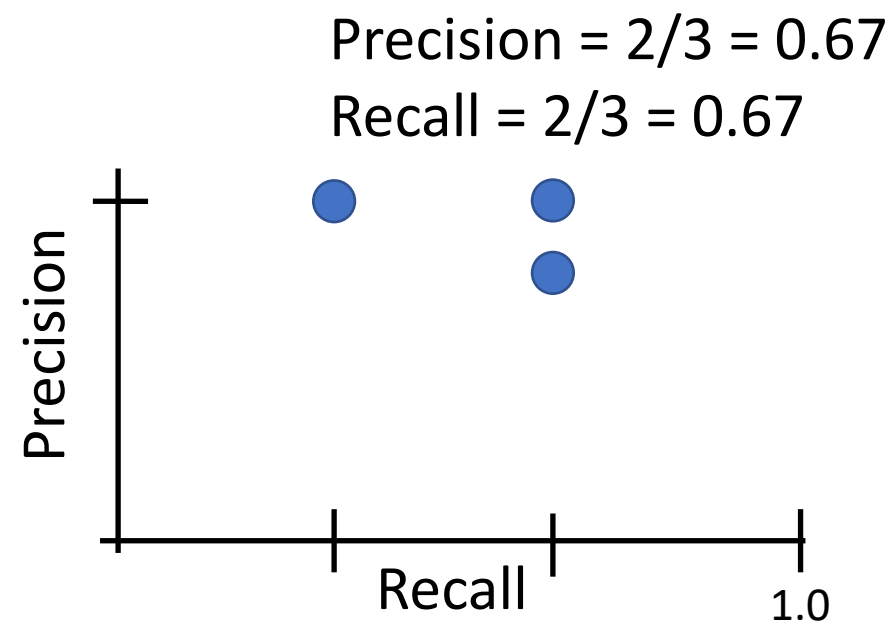
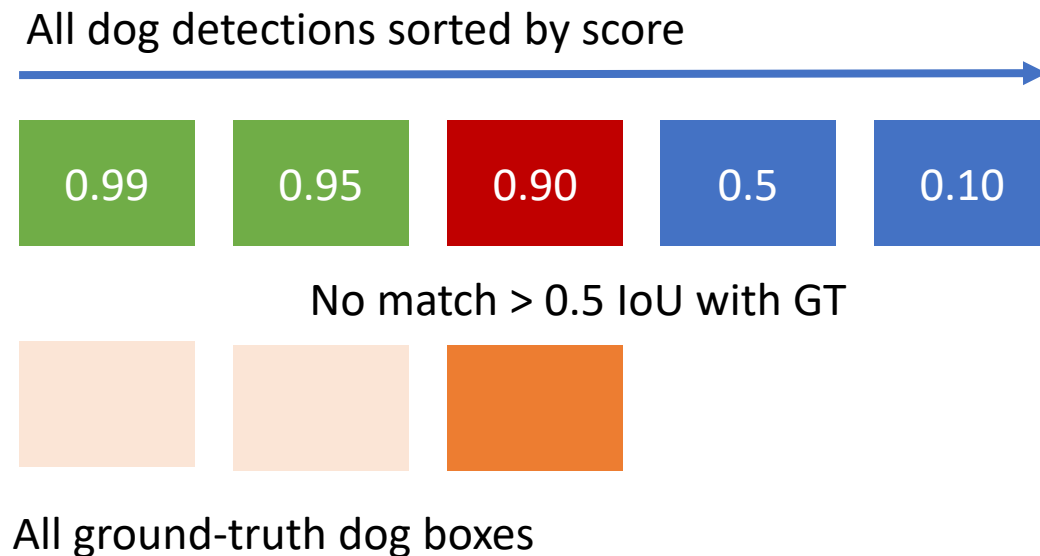
# Evaluating Object Detectors: Mean Average Precision (mAP)

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



# Evaluating Object Detectors: Mean Average Precision (mAP)

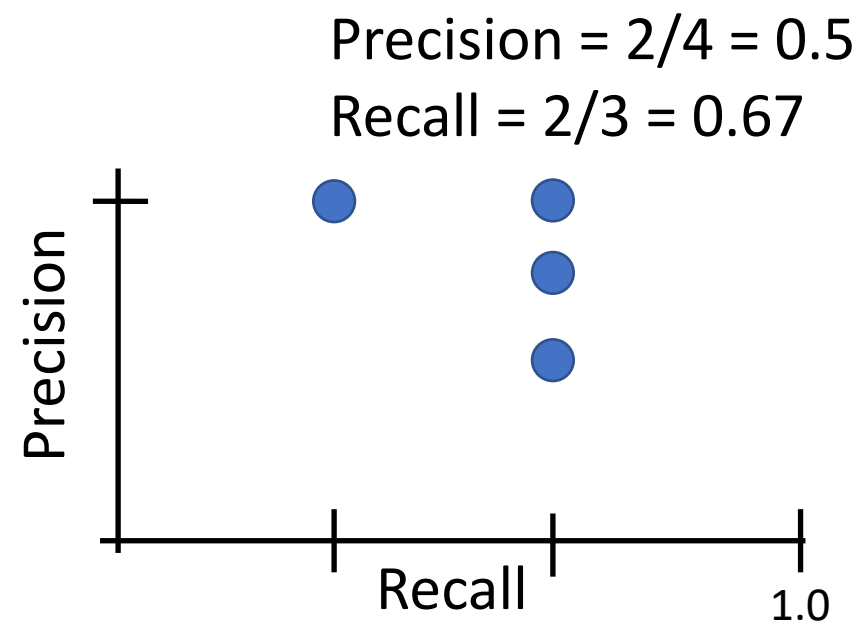
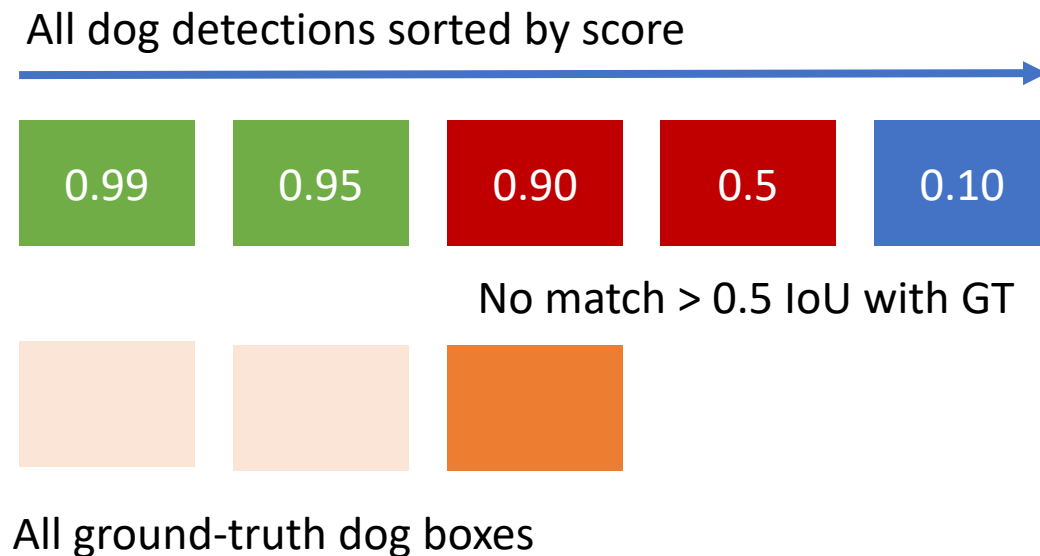
1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
  1. For each detection (highest score to lowest score)
    1. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
    2. Otherwise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





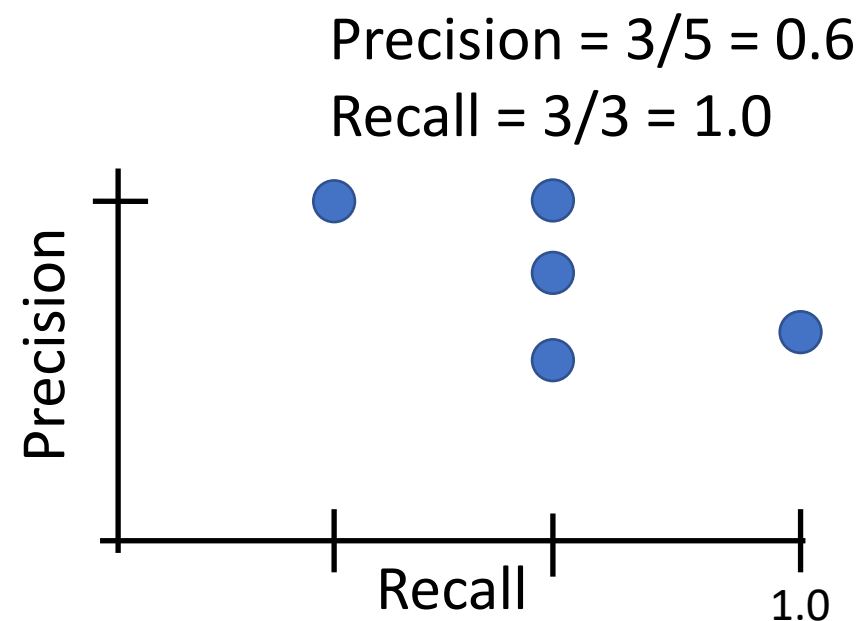
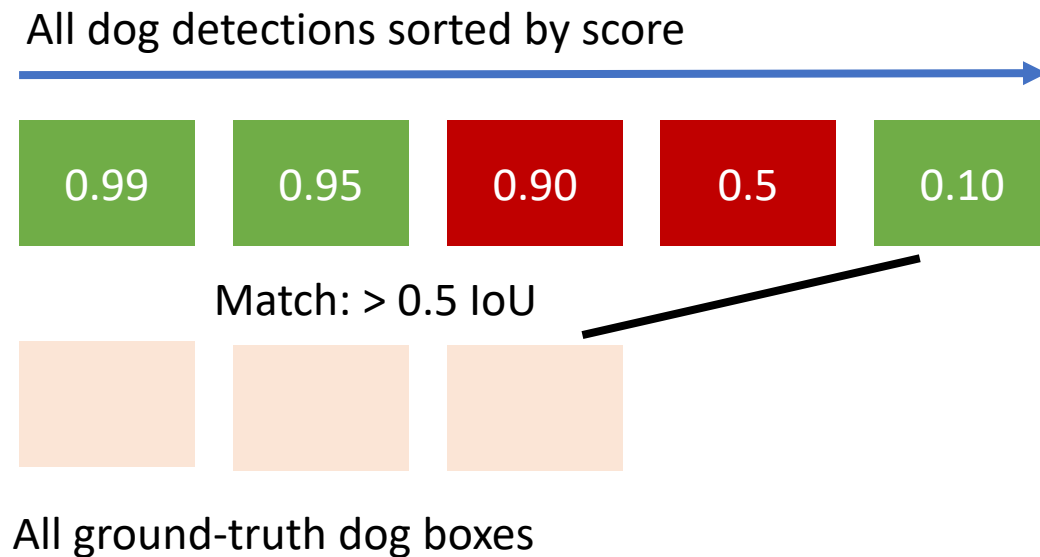
# Evaluating Object Detectors: Mean Average Precision (mAP)

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



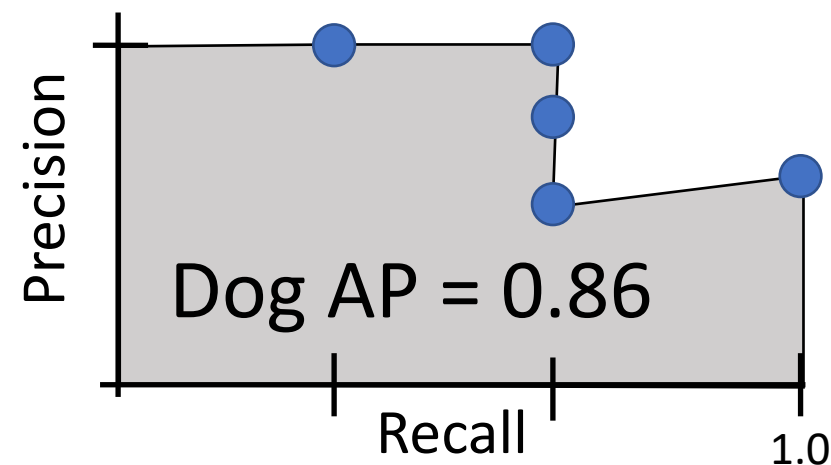
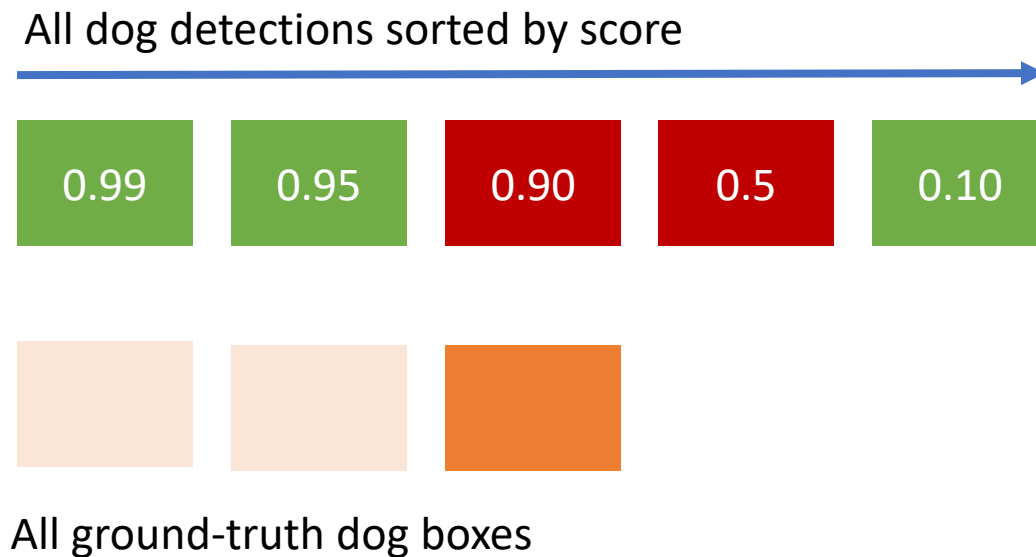
# Evaluating Object Detectors: Mean Average Precision (mAP)

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



# Evaluating Object Detectors: Mean Average Precision (mAP)

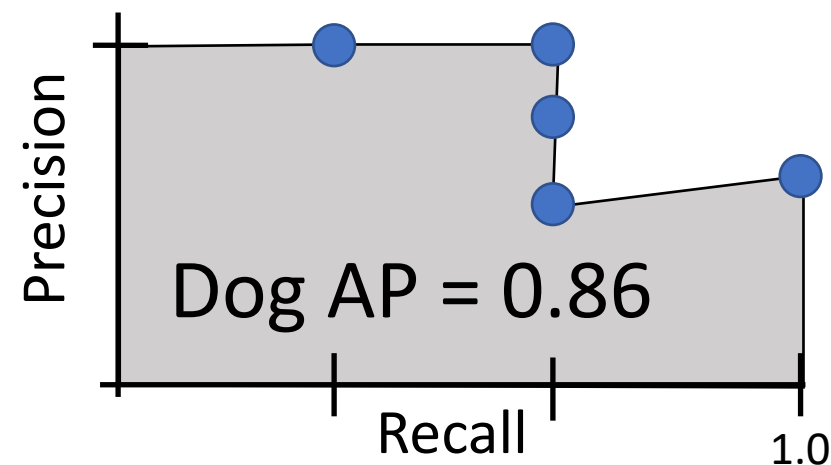
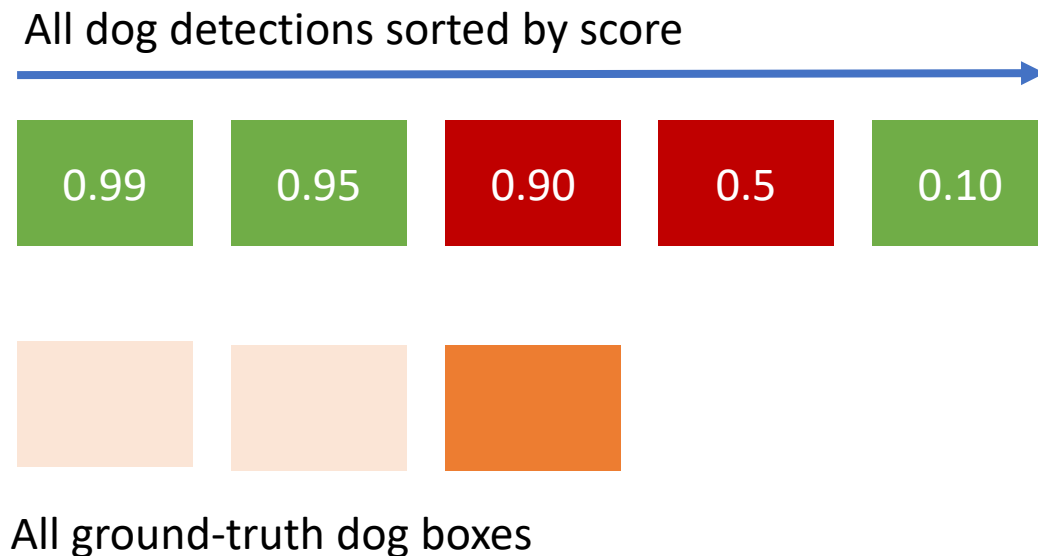
1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
  1. For each detection (highest score to lowest score)
    1. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
    2. Otherwise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



# Evaluating Object Detectors: Mean Average Precision (mAP)

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

**How to get AP = 1.0: Hit all GT boxes with  $\text{IoU} > 0.5$ , and have no “false positive” detections ranked above any “true positives”**





# Evaluating Object Detectors: Mean Average Precision (mAP)

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

Car AP = 0.65

Cat AP = 0.80

Dog AP = 0.86

mAP@0.5 = 0.77

# Evaluating Object Detectors: Mean Average Precision (mAP)

1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
  1. For each detection (highest score to lowest score)
    1. If it matches some GT box with  $\text{IoU} > 0.5$ , mark it as positive and eliminate the GT
    2. Otherwise mark it as negative
    3. Plot a point on PR Curve
  2. Average Precision (AP) = area under PR curve
3. Mean Average Precision (mAP) = average of AP for each category
4. For “COCO mAP”: Compute  $\text{mAP@thresh}$  for each IoU threshold (0.5, 0.55, 0.6, ..., 0.95) and take average

$$\text{mAP@0.5} = 0.77$$

$$\text{mAP@0.55} = 0.71$$

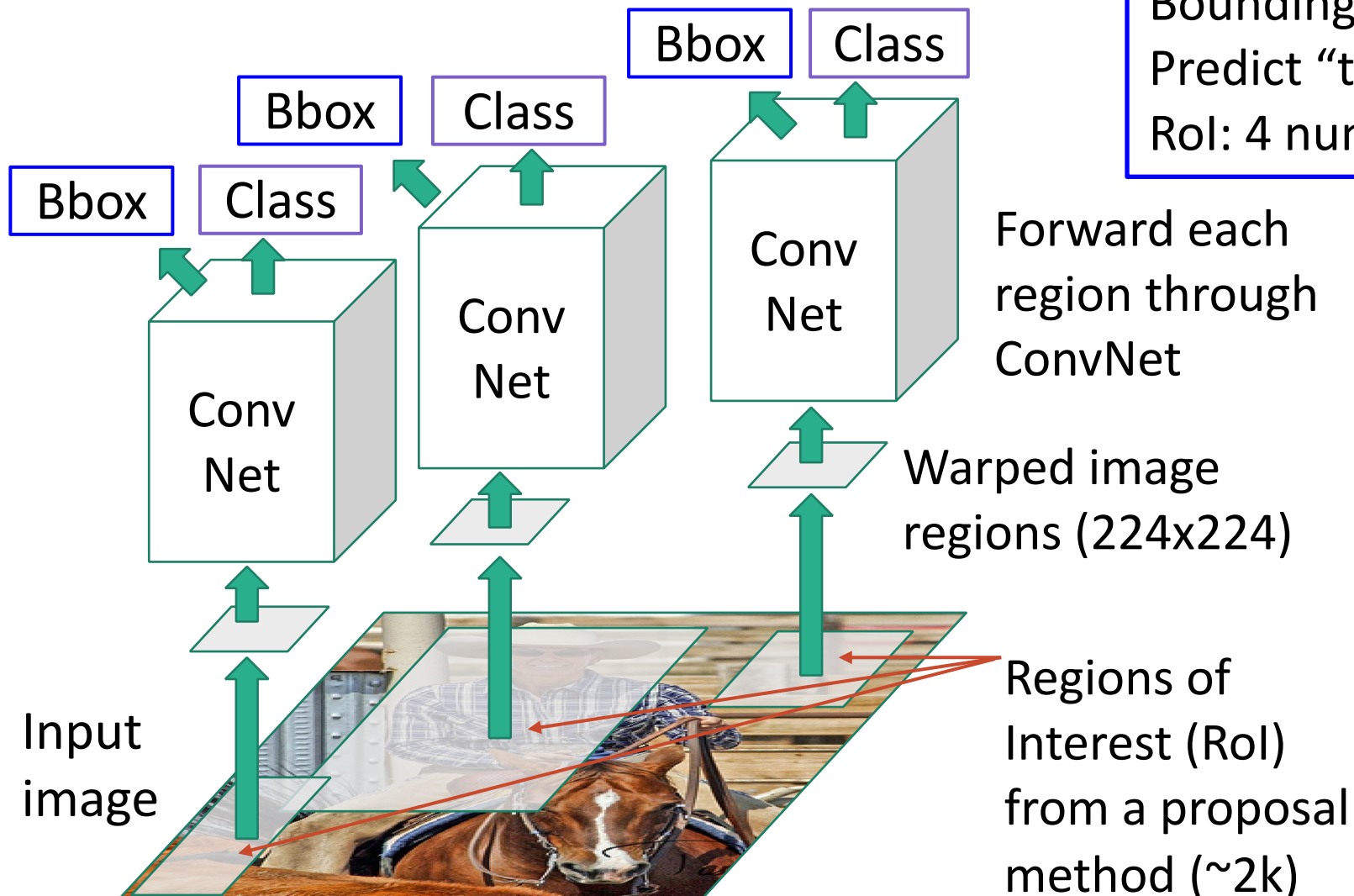
$$\text{mAP@0.60} = 0.65$$

...

$$\text{mAP@0.95} = 0.2$$

$$\text{COCO mAP} = 0.4$$

# R-CNN: Region-Based CNN

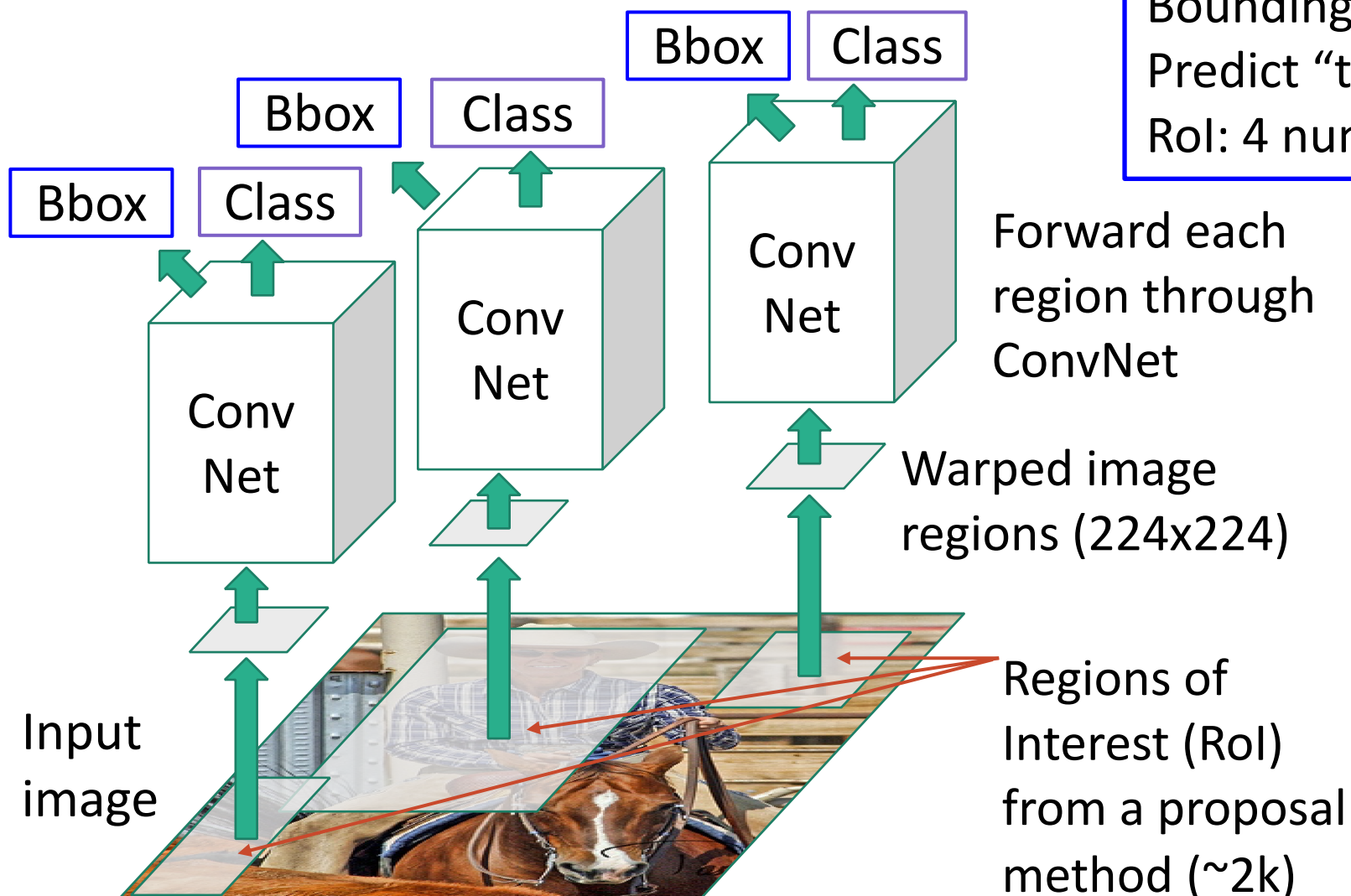


Classify each region

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

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)

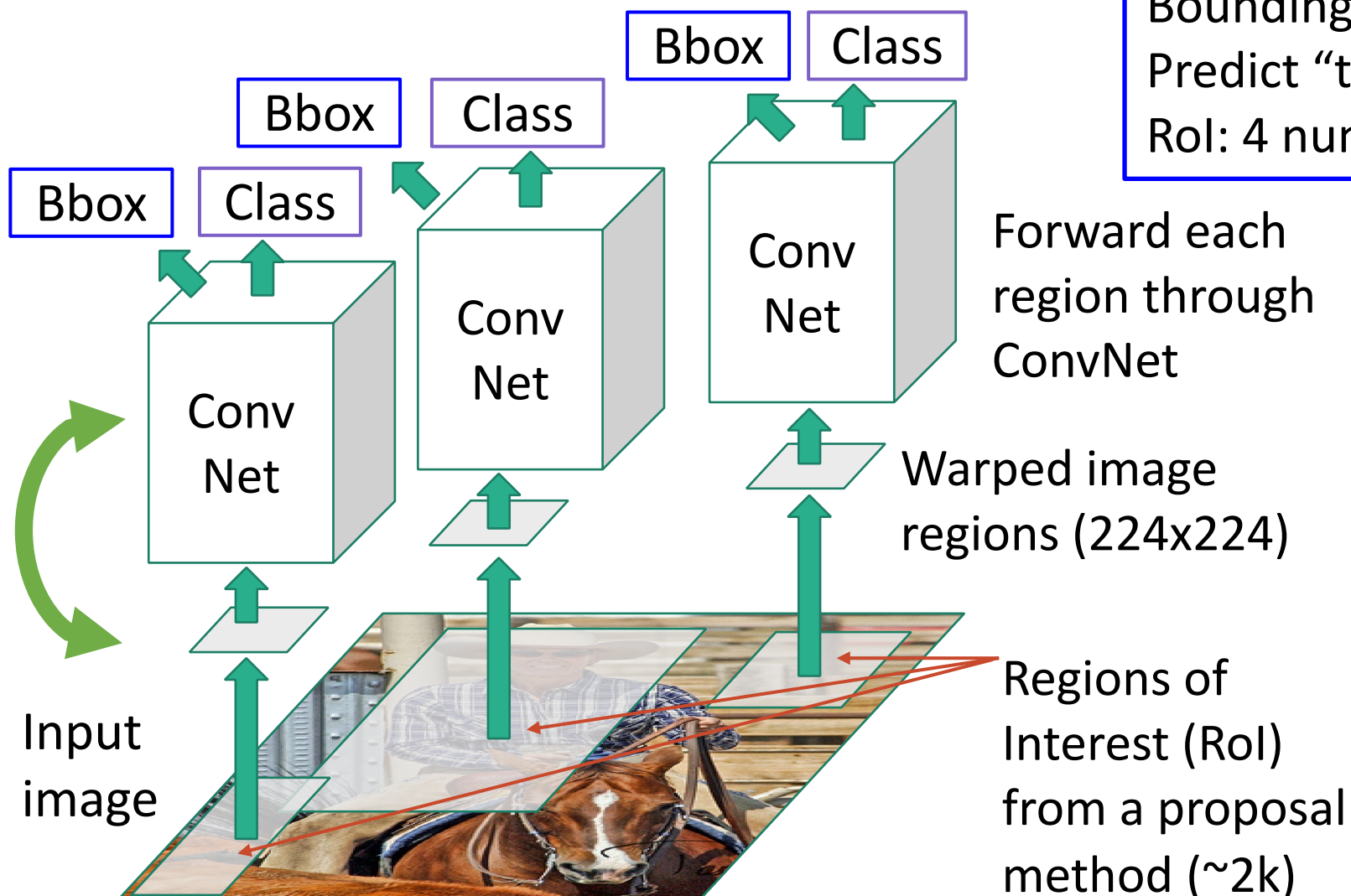
Regions of  
Interest (RoI)  
from a proposal  
method (~2k)

**Problem: Very slow!**  
Need to do ~2k forward  
passes for each image!

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)

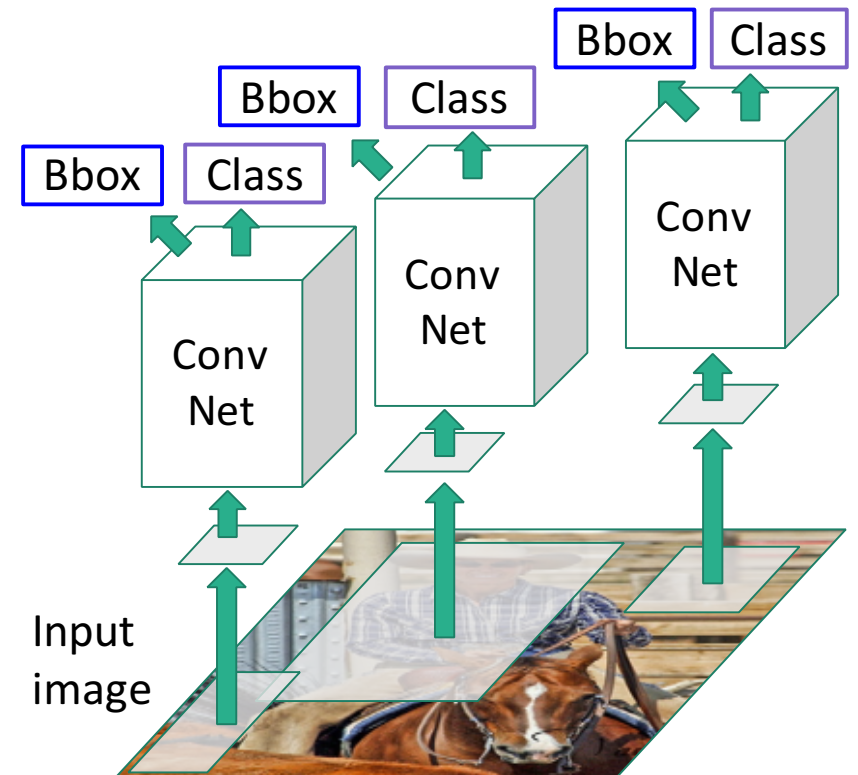
**Problem:** Very slow!  
Need to do ~2k forward  
passes for each image!

**Solution:** Run CNN  
\*before\* warping!

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.  
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

## “Slow” R-CNN

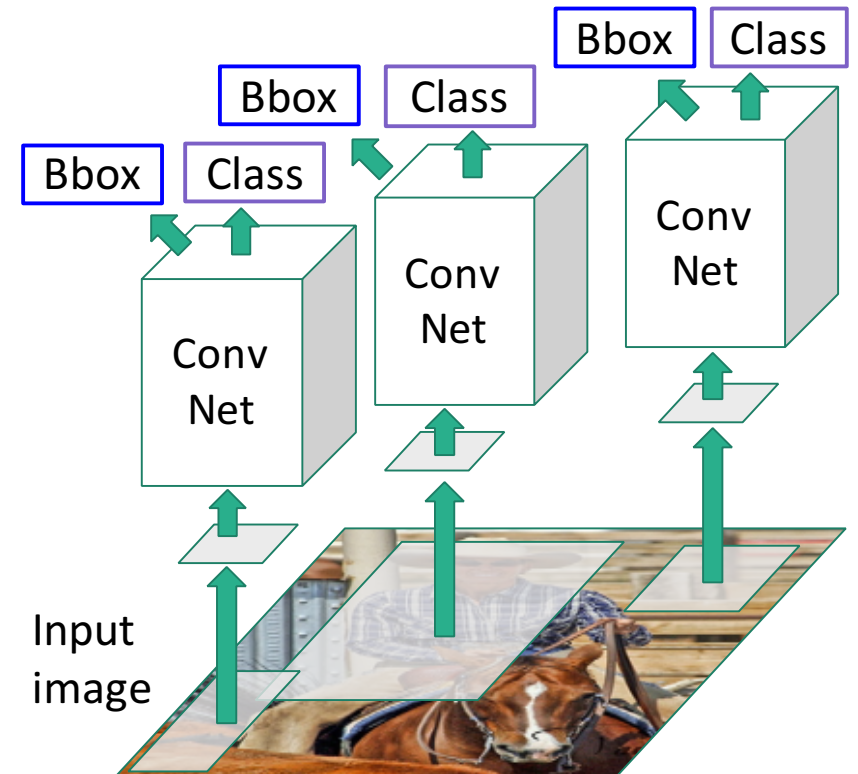
Process each region  
independently



# Fast R-CNN



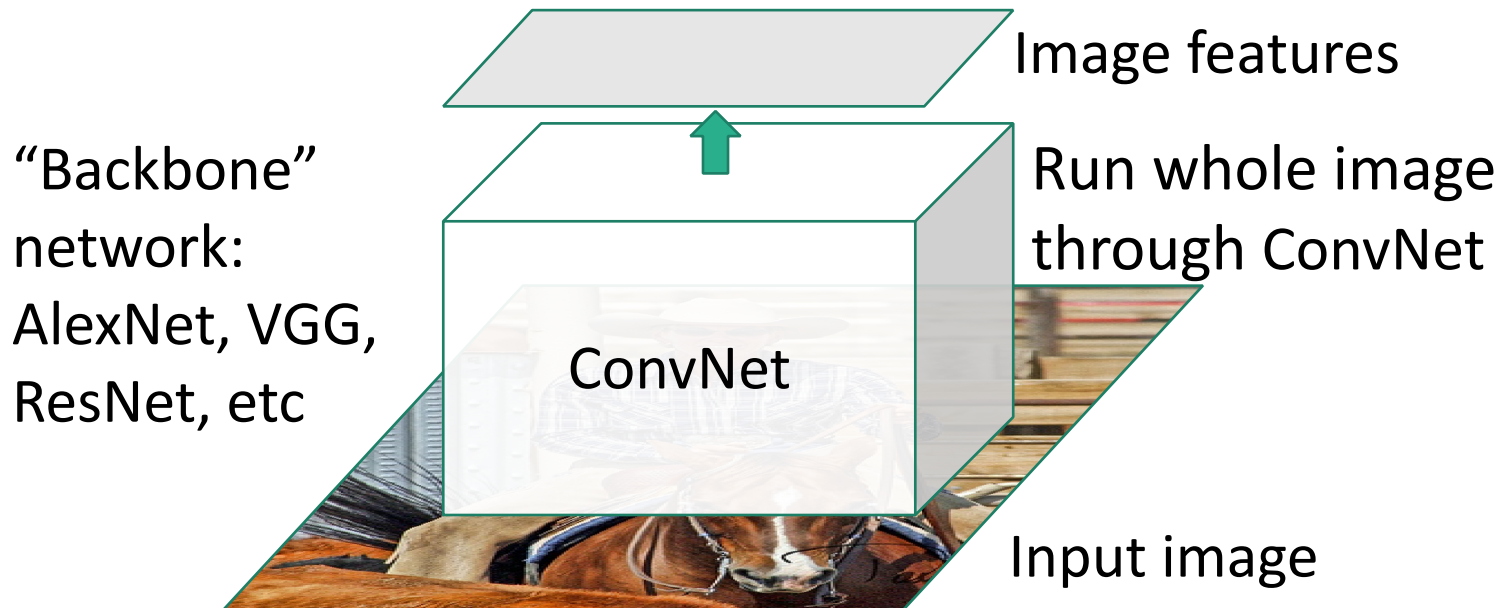
Input image



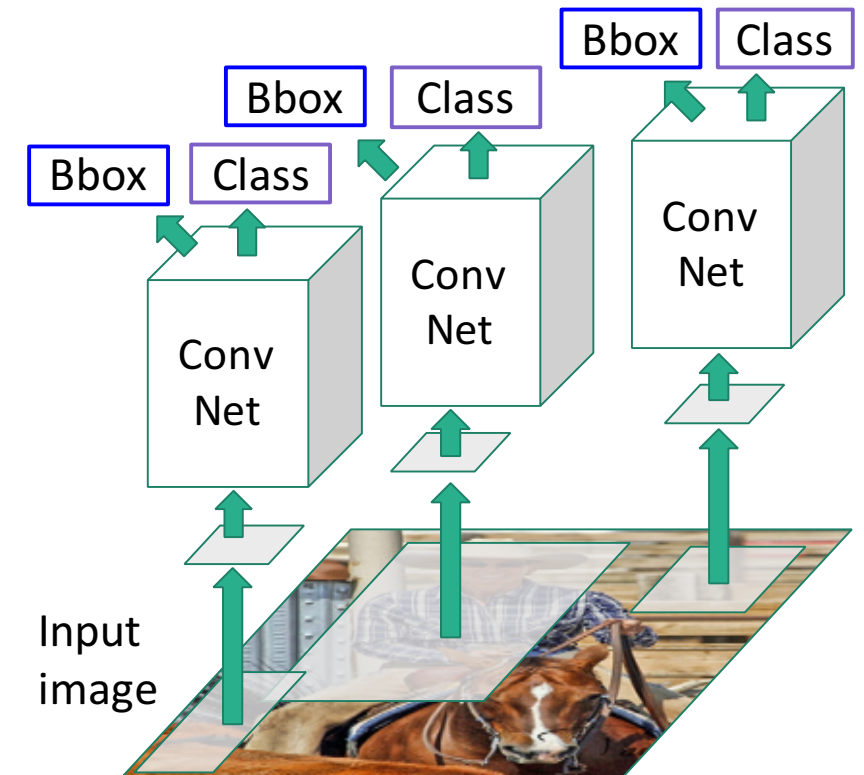
“Slow” R-CNN  
Process each region  
independently



# Fast R-CNN



“Slow” R-CNN  
Process each region  
independently

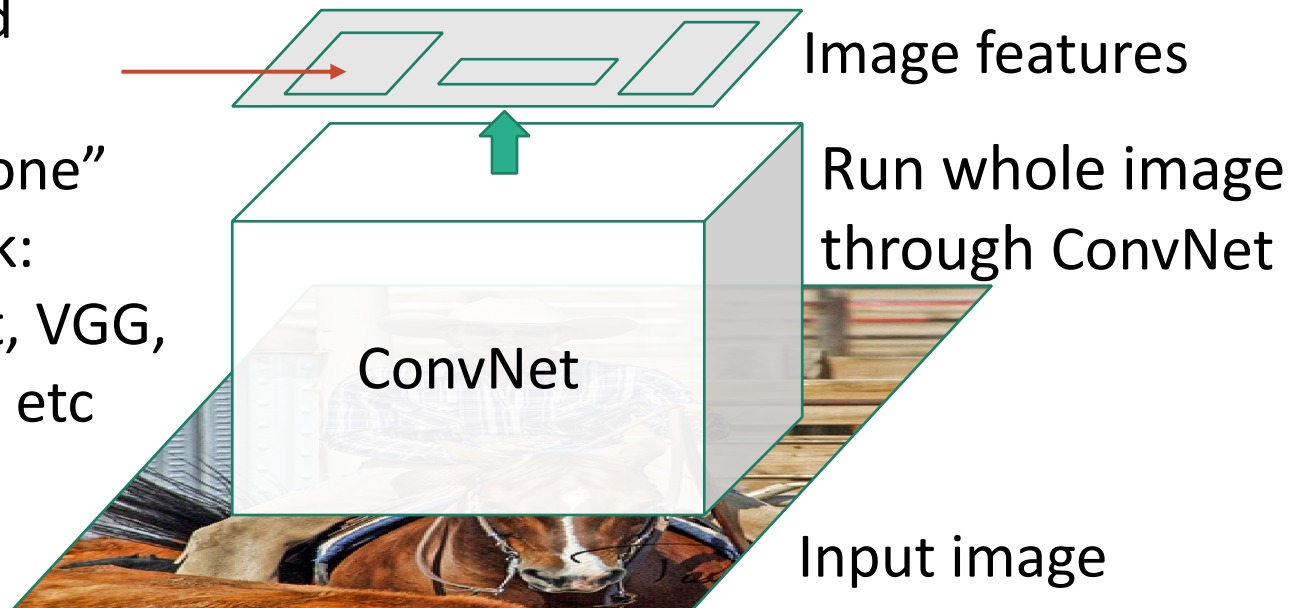


Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

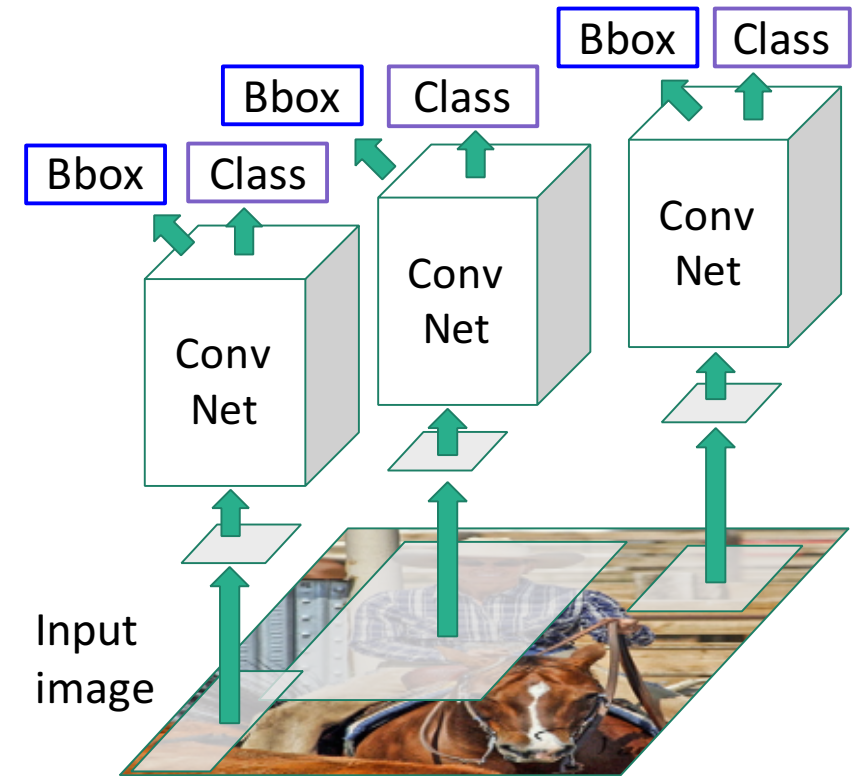
# Fast R-CNN

Regions of Interest (RoIs) from a proposal method

“Backbone” network:  
AlexNet, VGG, ResNet, etc



“Slow” R-CNN  
Process each region independently

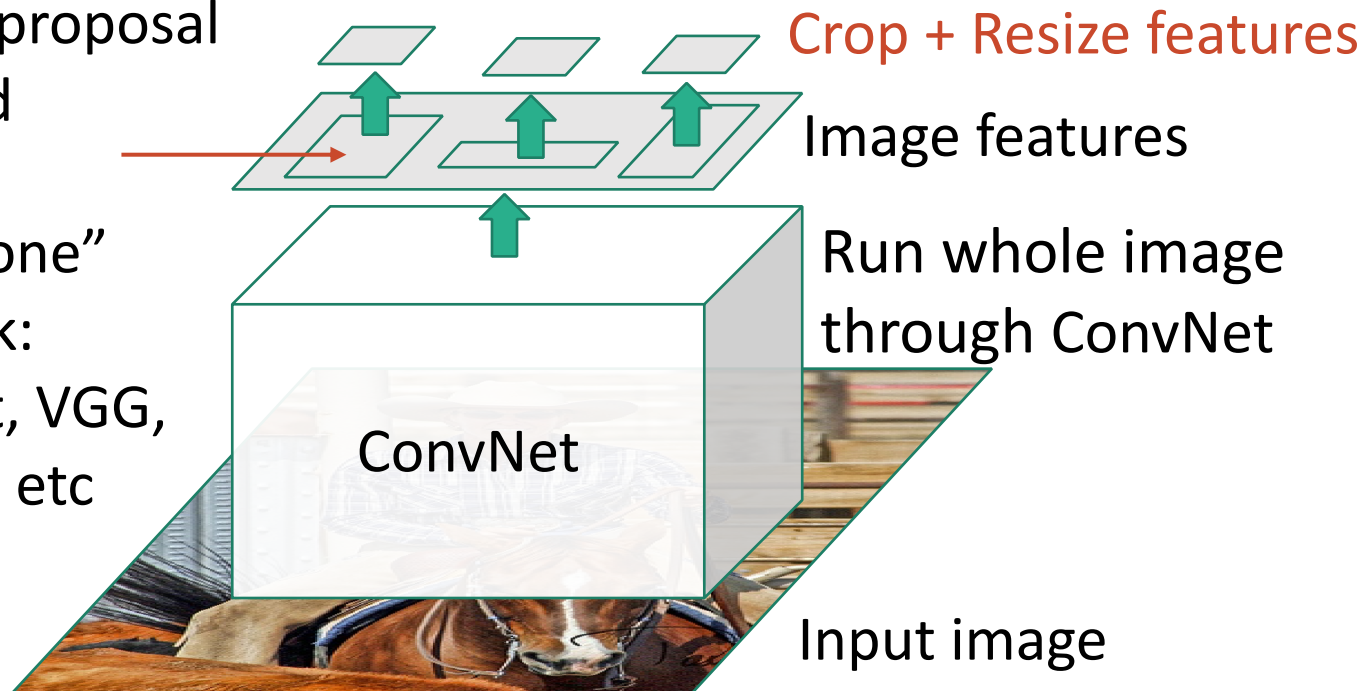


Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

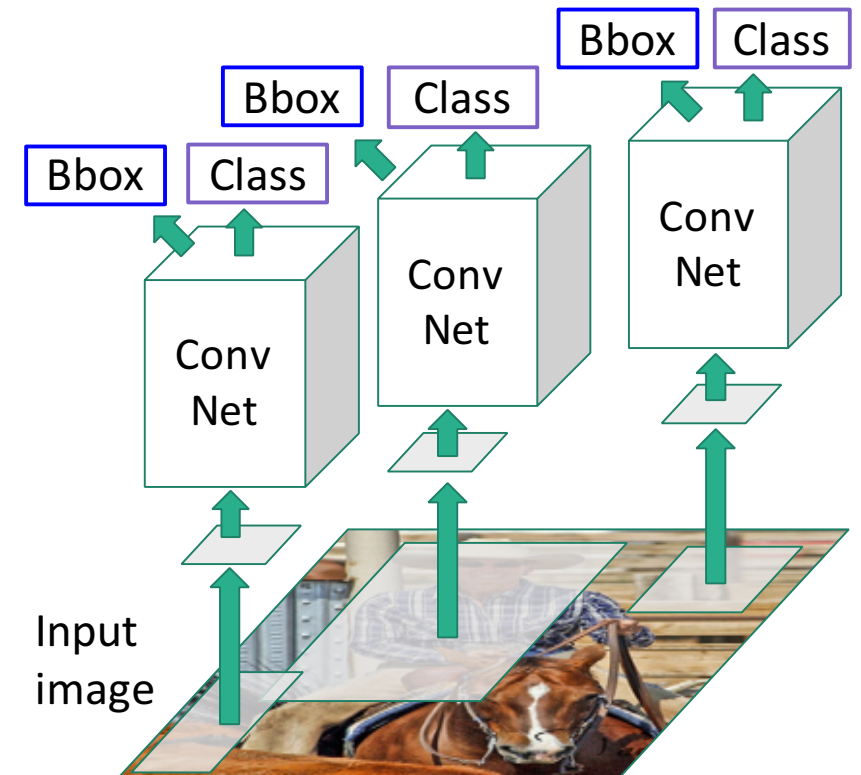
# Fast R-CNN

Regions of Interest (RoIs) from a proposal method

“Backbone” network:  
AlexNet, VGG, ResNet, etc



“Slow” R-CNN  
Process each region independently

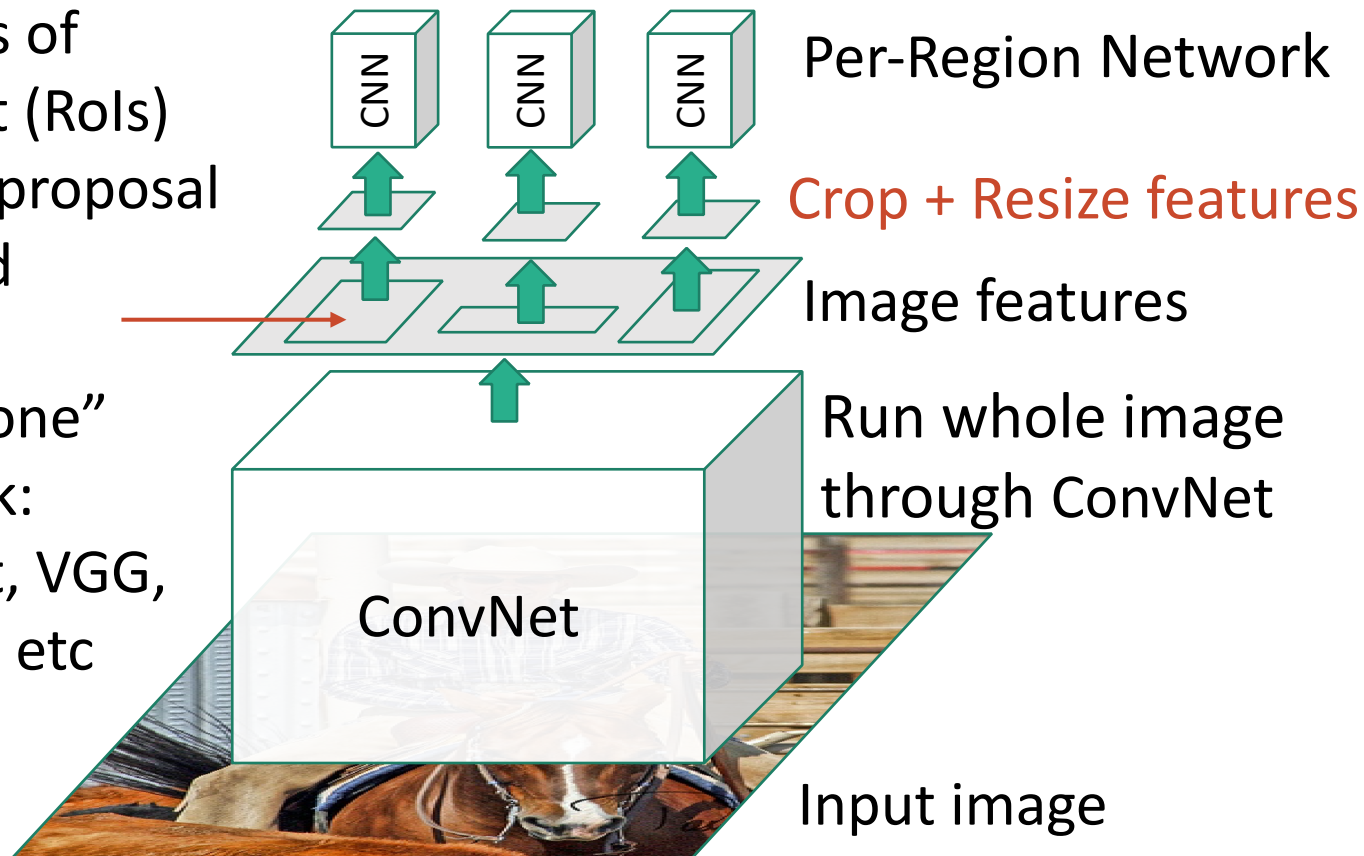


Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

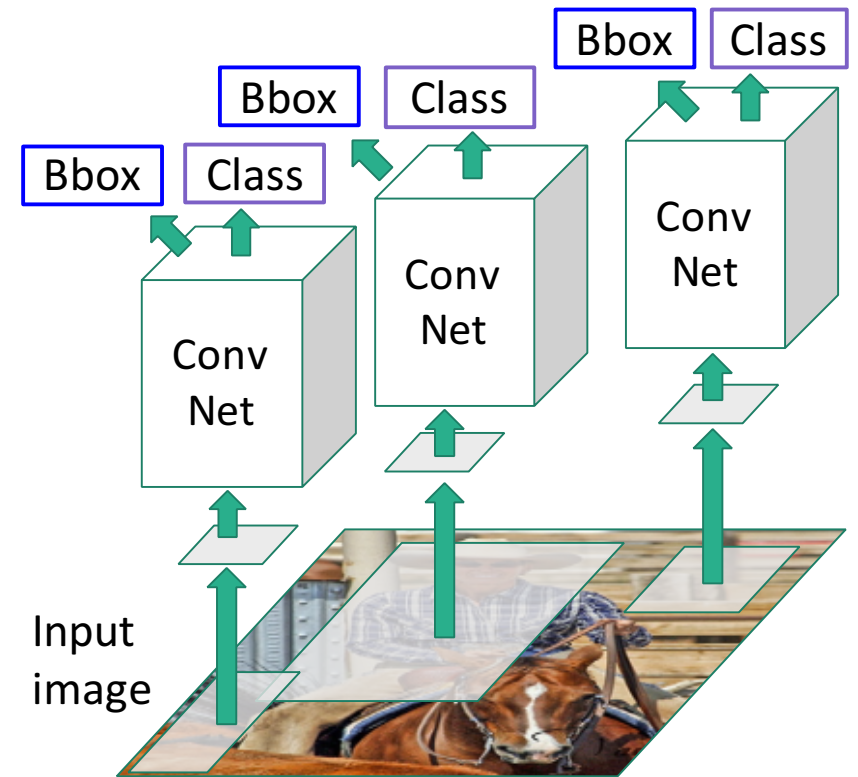
# Fast R-CNN

Regions of Interest (RoIs) from a proposal method

“Backbone” network:  
AlexNet, VGG, ResNet, etc



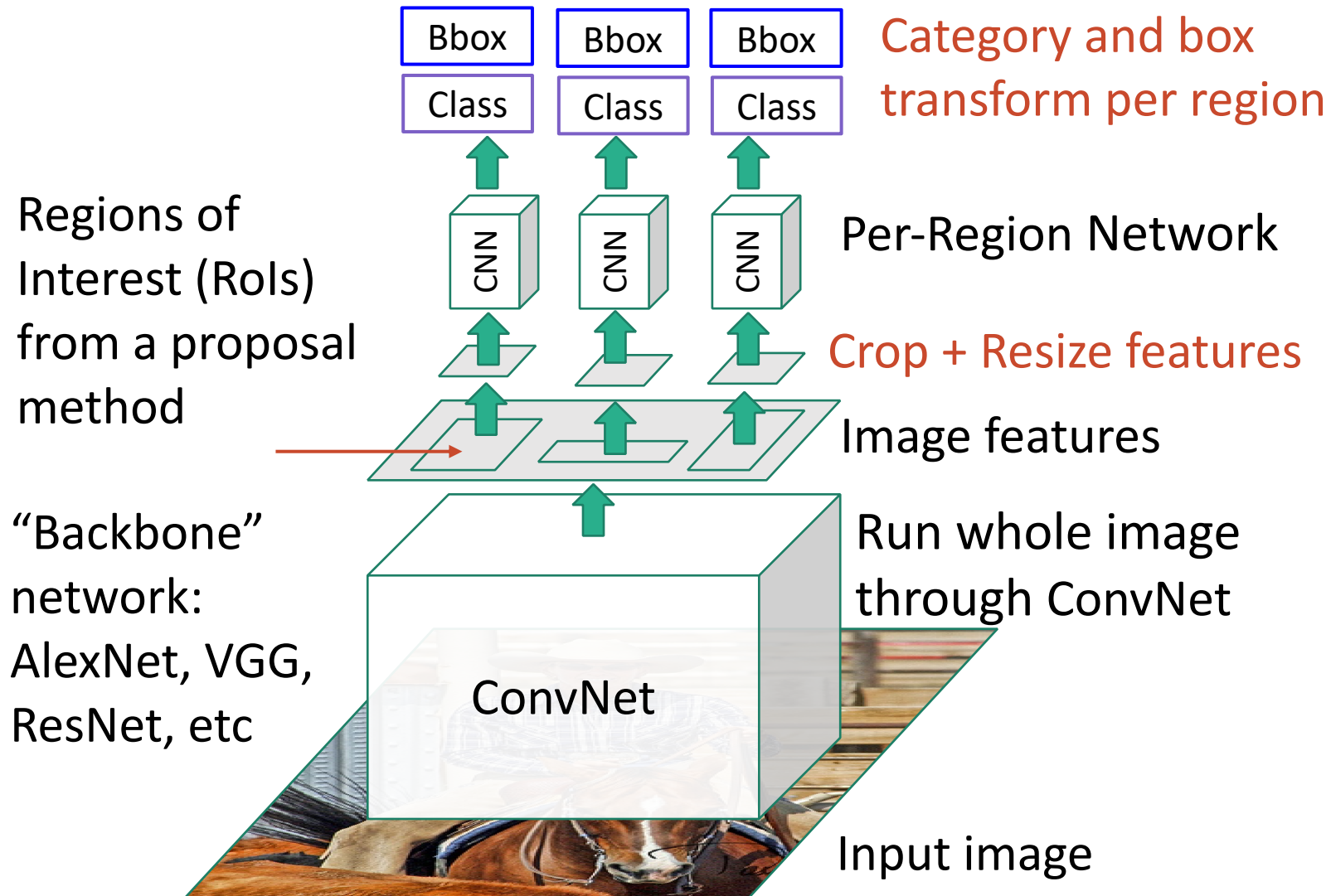
“Slow” R-CNN  
Process each region independently



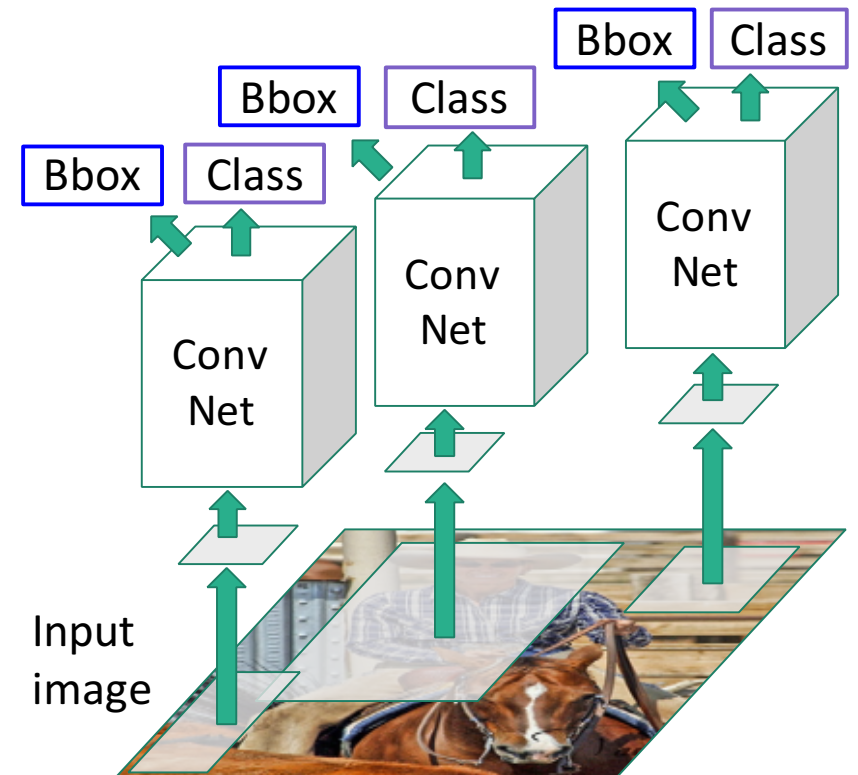
Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.



# Fast R-CNN

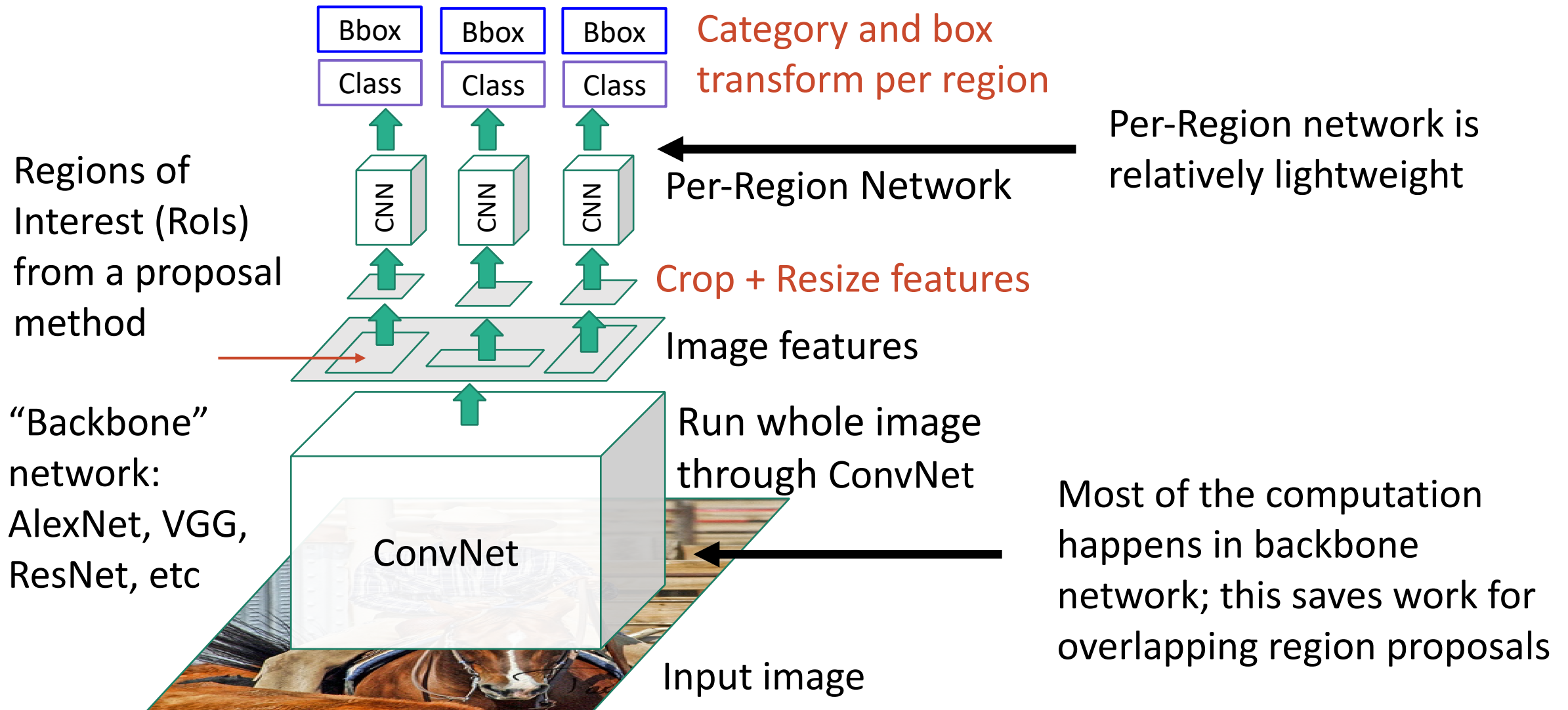


“Slow” R-CNN  
Process each region independently



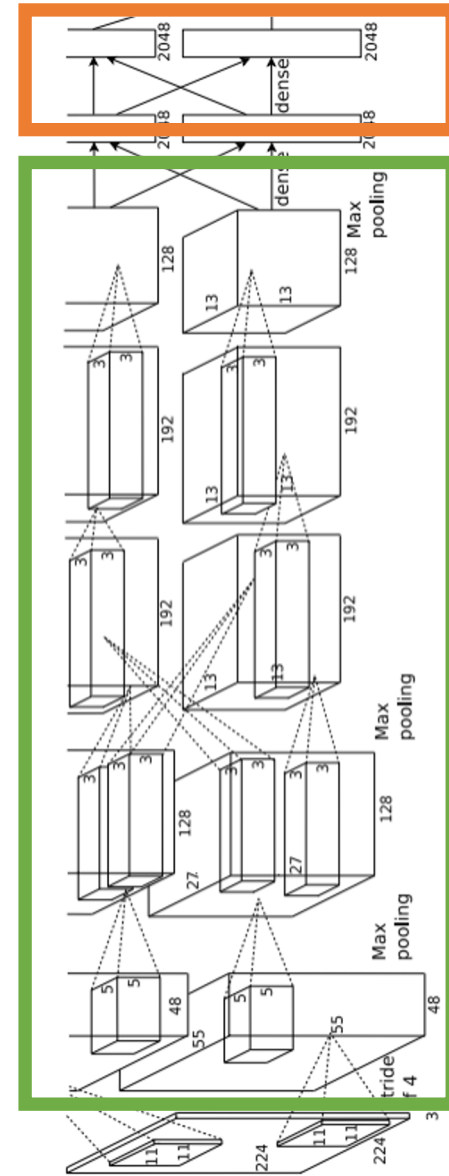
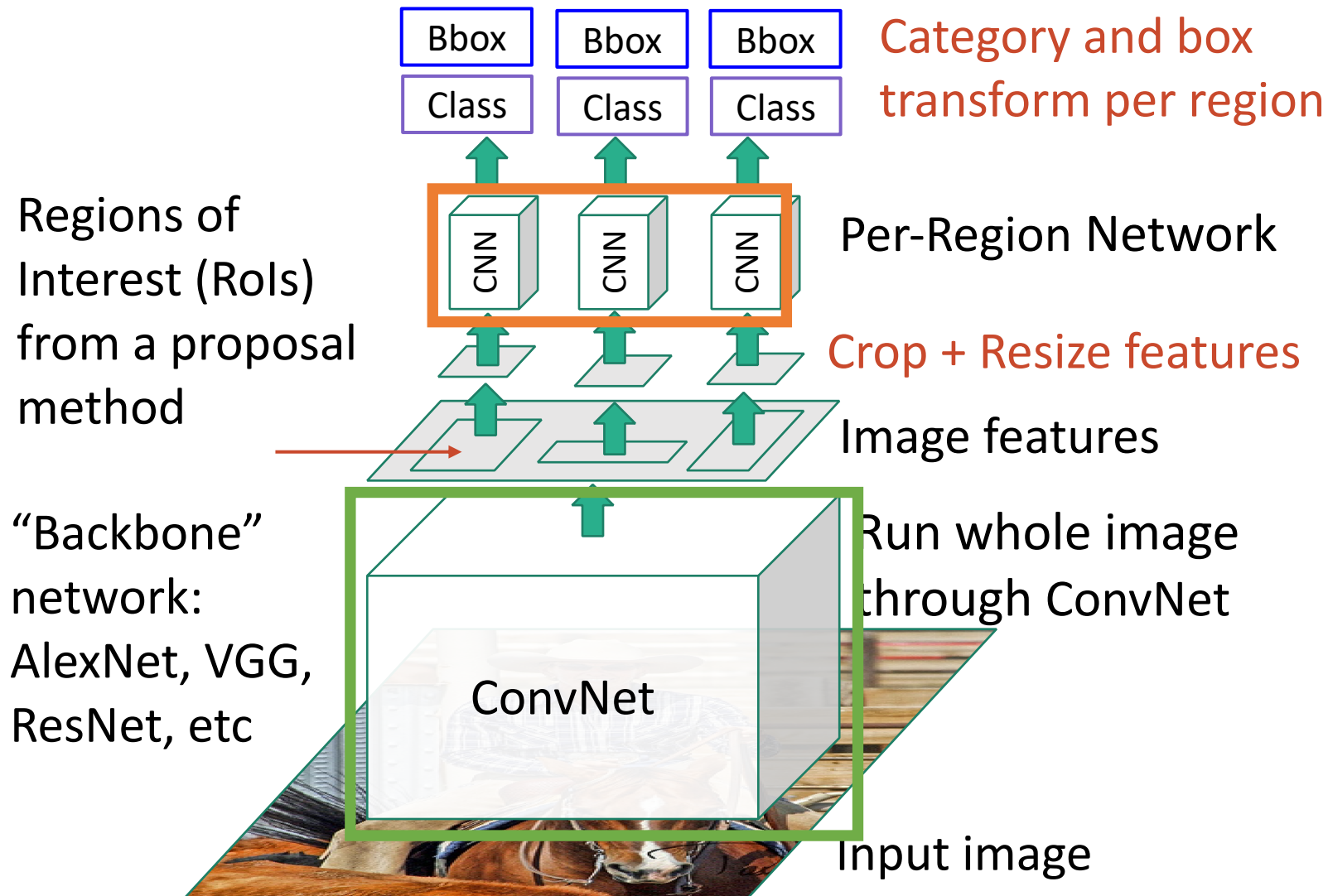
Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

# Fast R-CNN



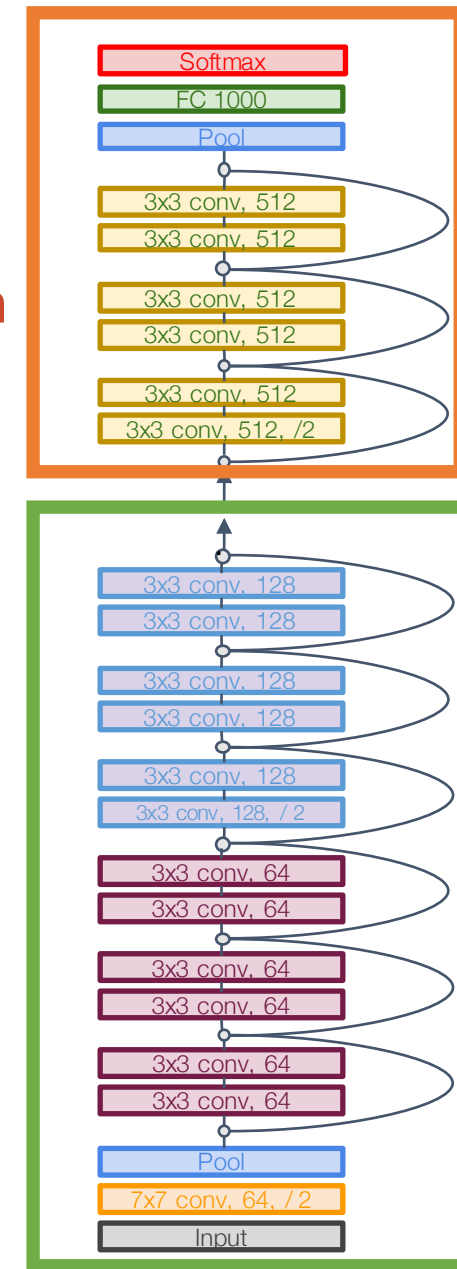
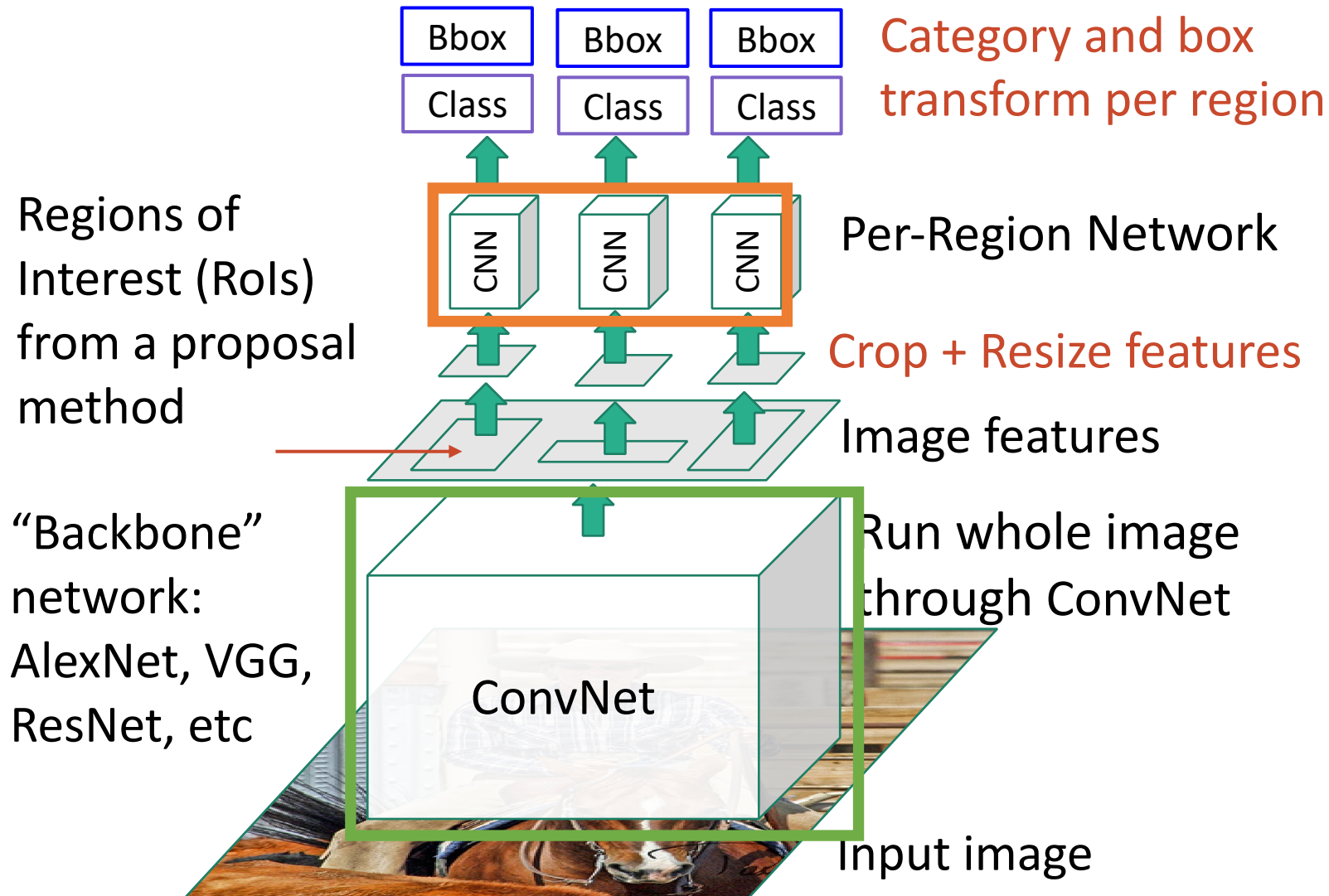
Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

# Fast R-CNN



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

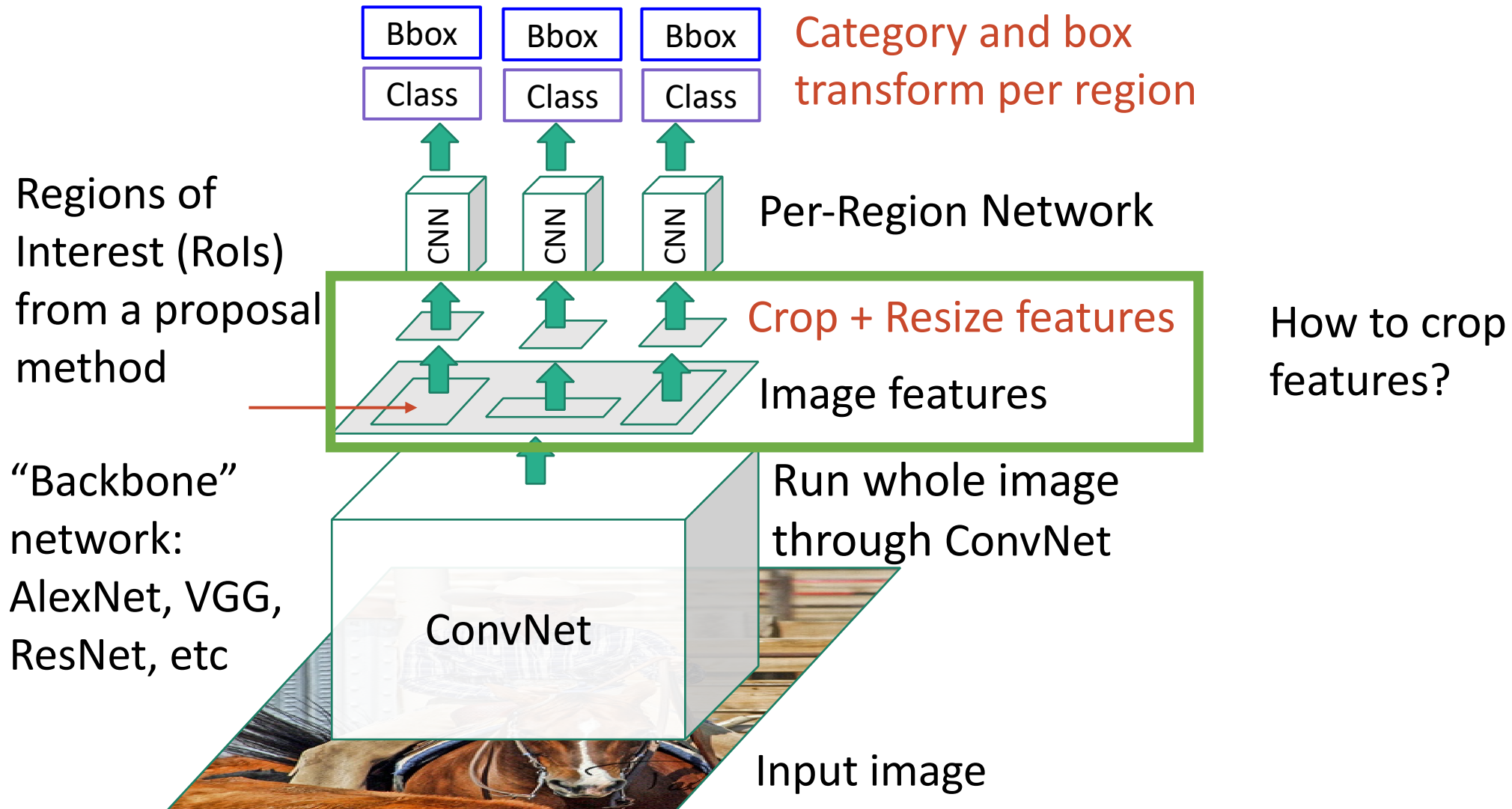
# Fast R-CNN



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

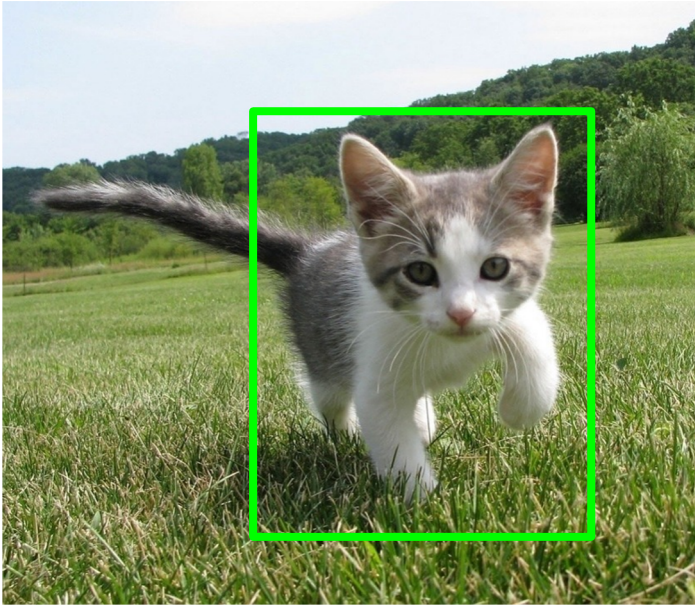


# Fast R-CNN



Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

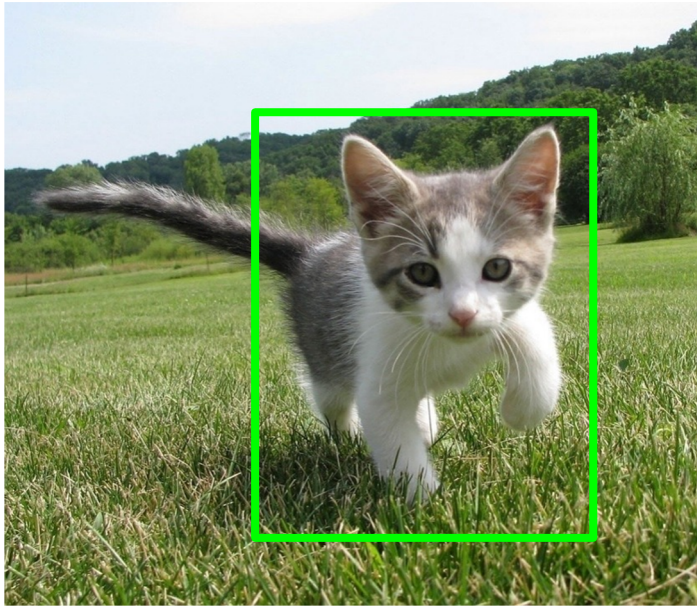
# Cropping Features: RoI Pool



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

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

# Cropping Features: RoI Pool



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

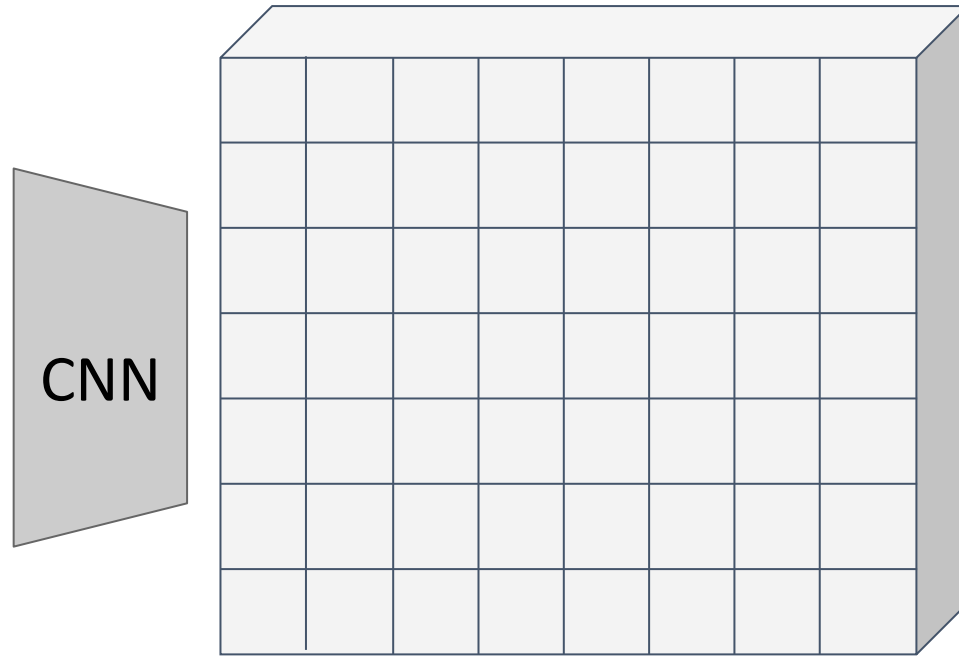
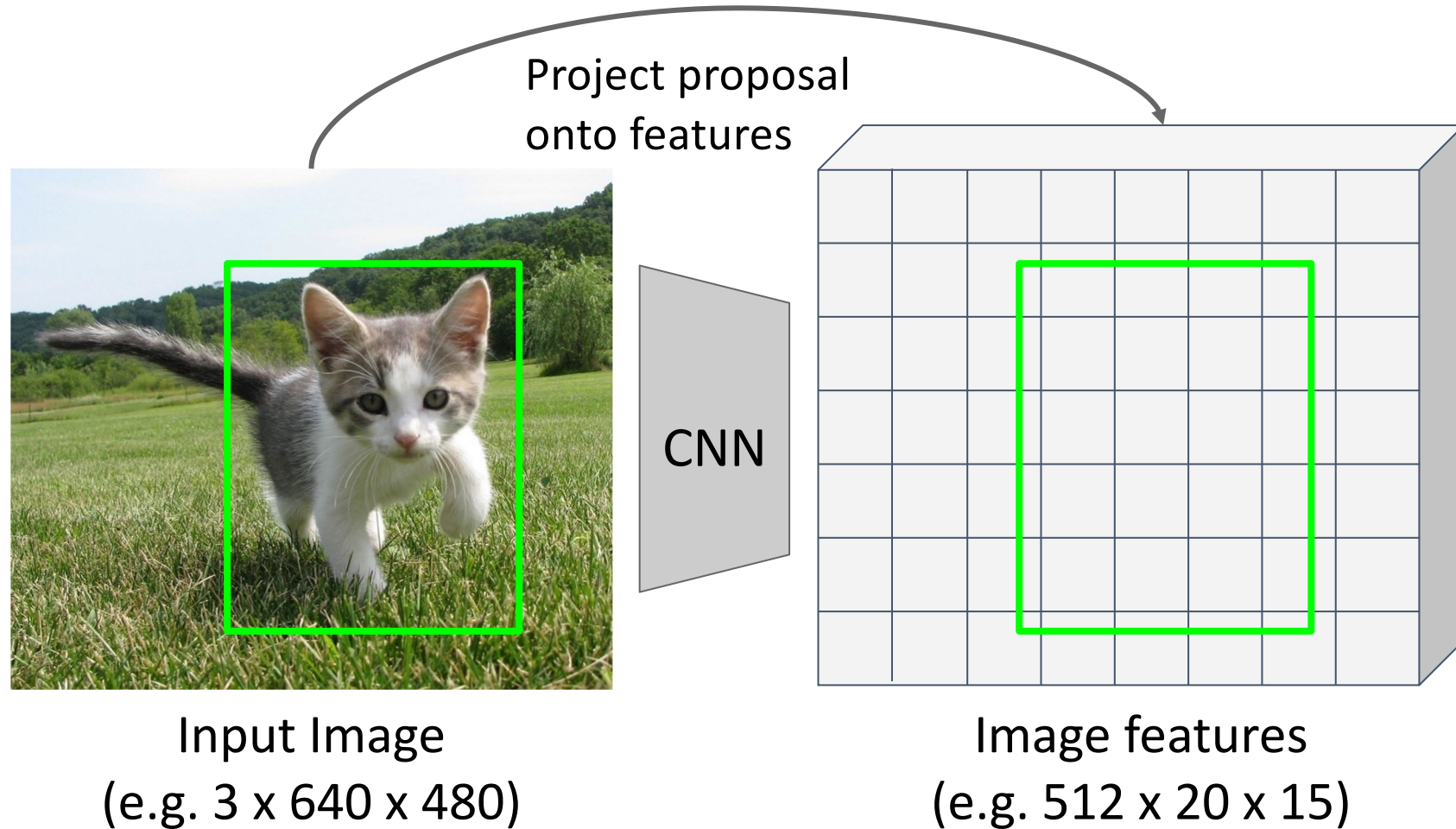


Image features  
(e.g. 512 x 20 x 15)

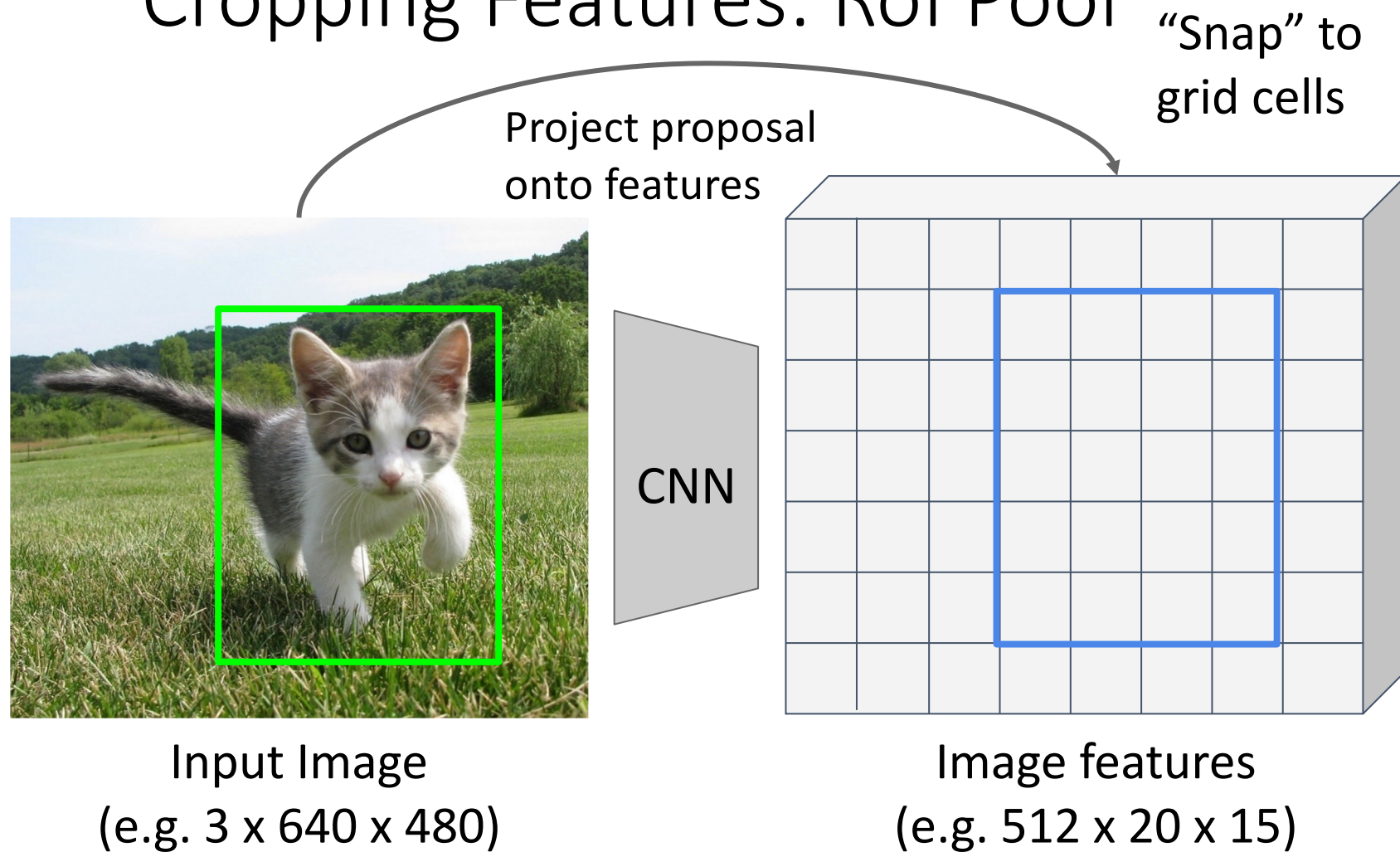
# Cropping Features: RoI Pool



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

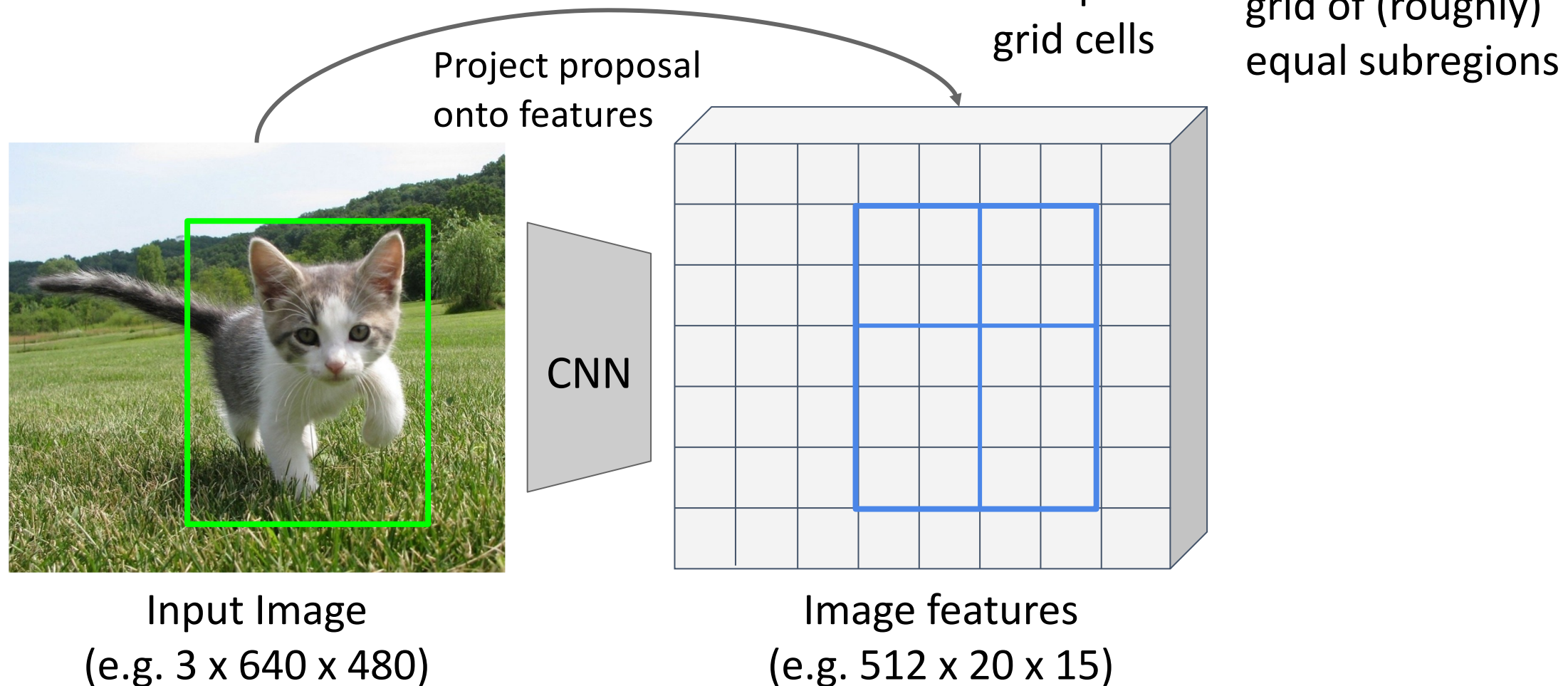


# Cropping Features: RoI Pool



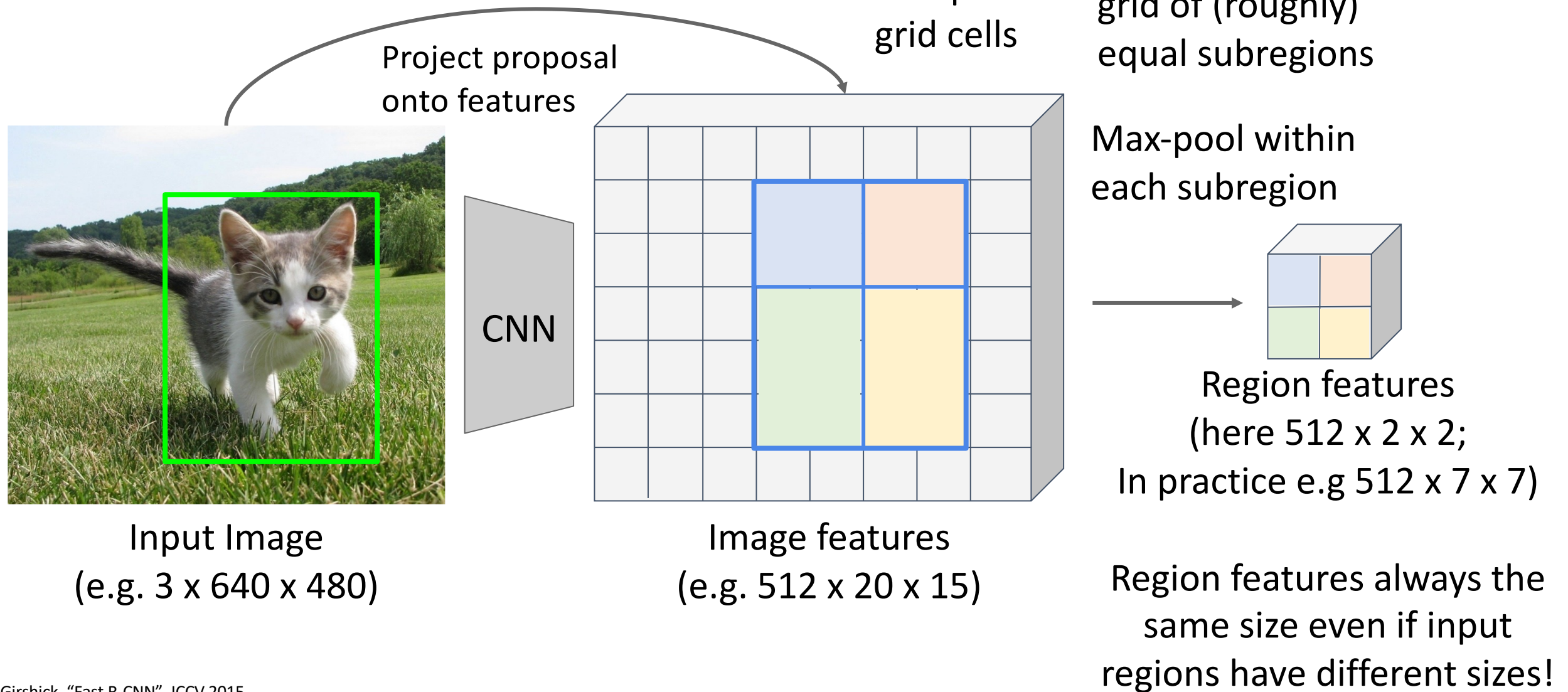
Girshick, "Fast R-CNN", ICCV 2015.

# Cropping Features: RoI Pool



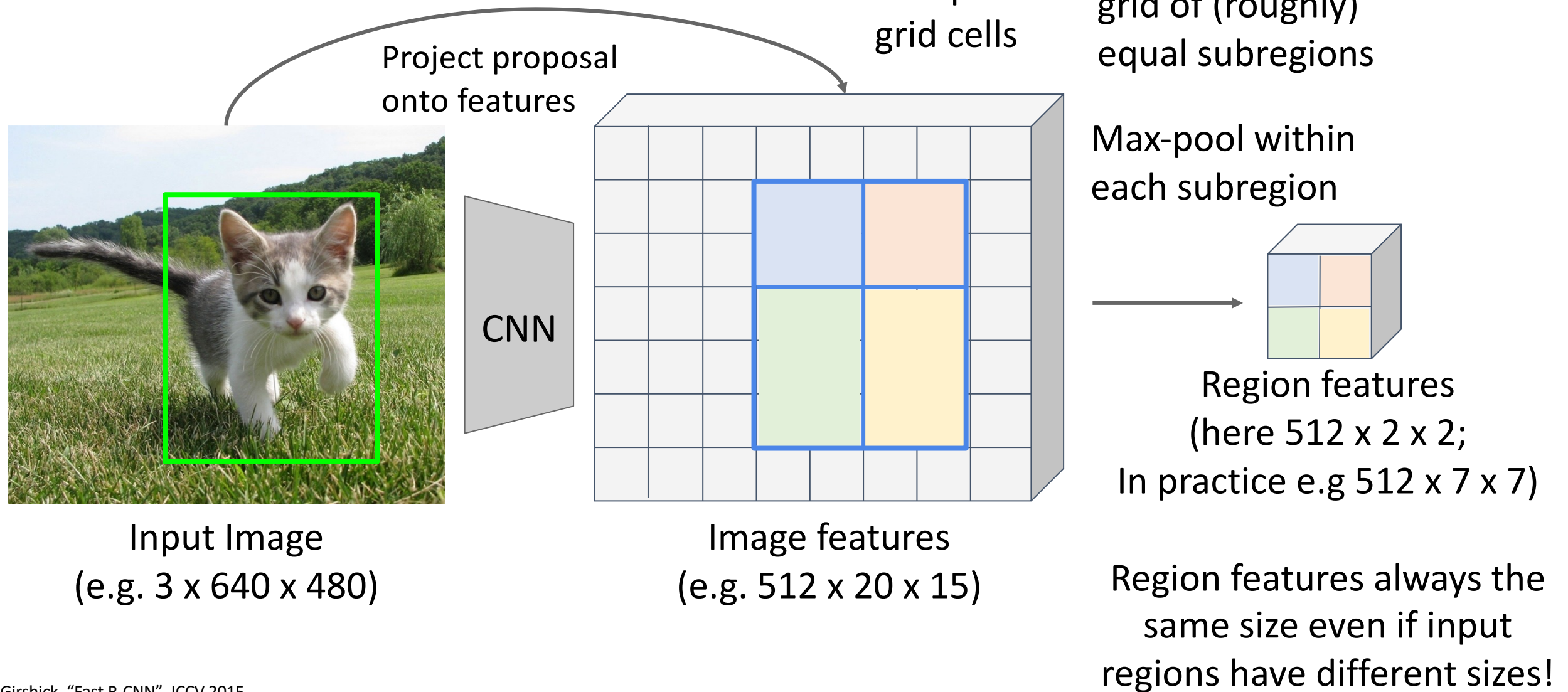
Girshick, "Fast R-CNN", ICCV 2015.

# Cropping Features: RoI Pool



Girshick, “Fast R-CNN”, ICCV 2015.

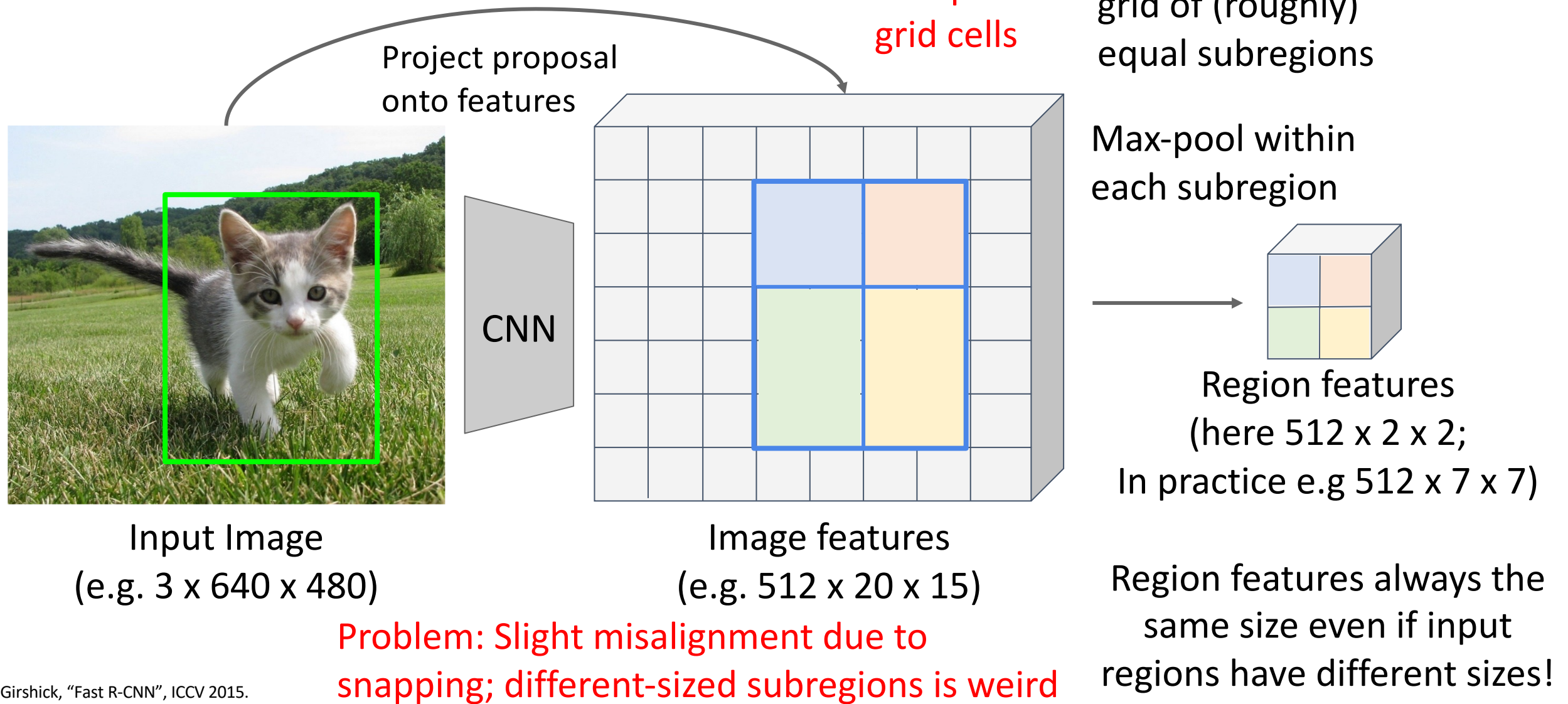
# Cropping Features: RoI Pool



Girshick, “Fast R-CNN”, ICCV 2015.

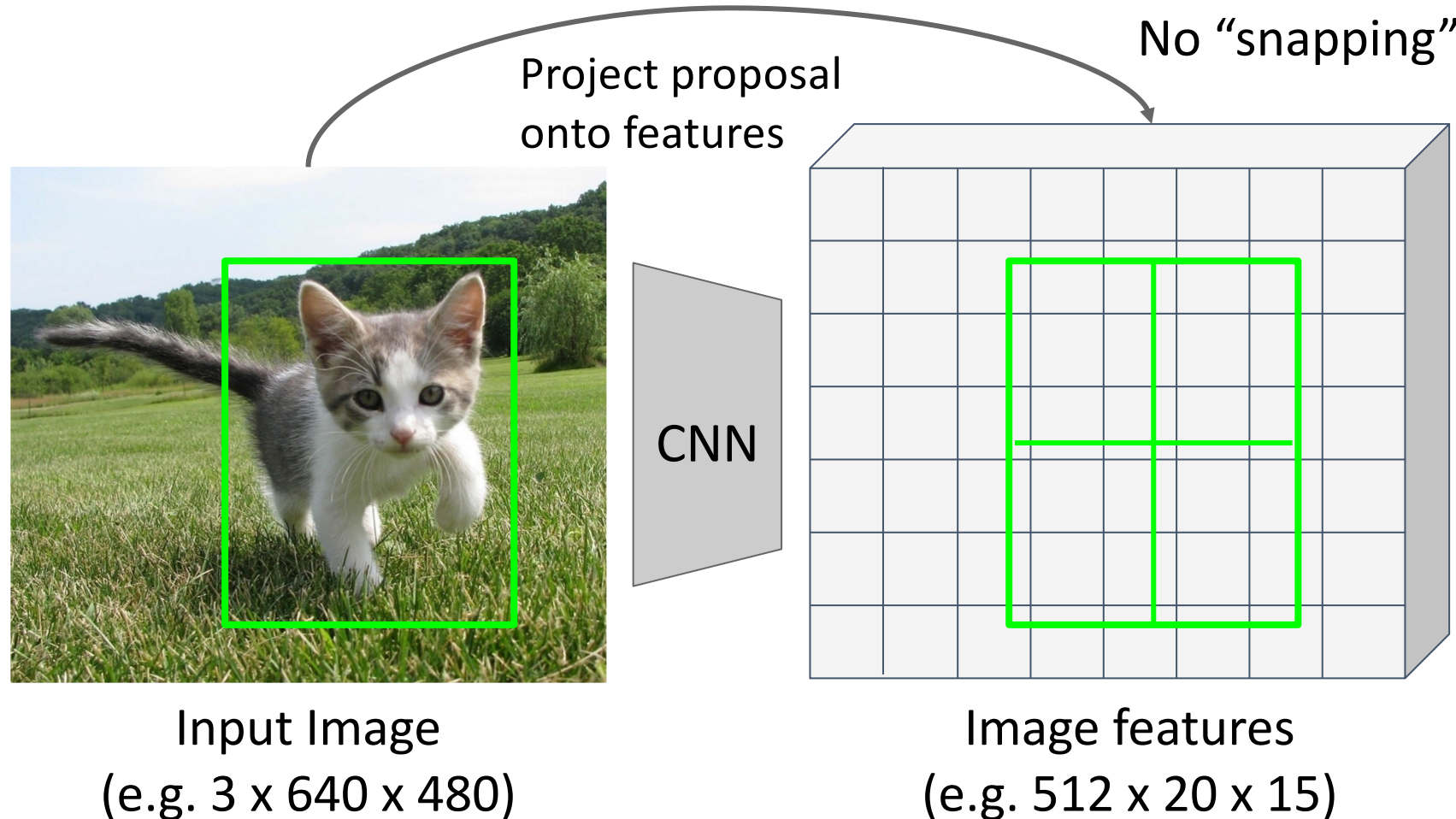


# Cropping Features: RoI Pool



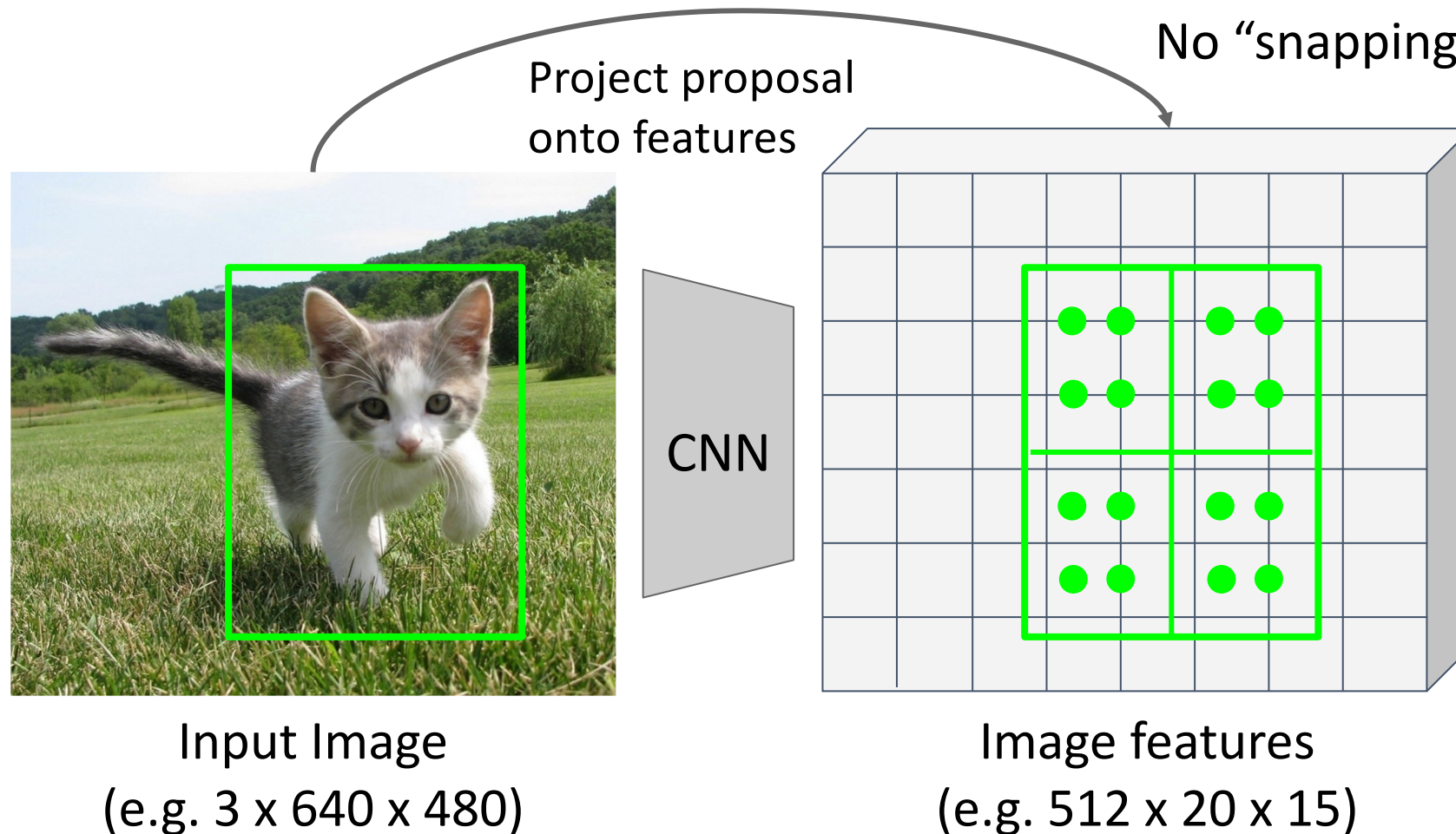
# Cropping Features: RoI Align

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



# Cropping Features: RoI Align

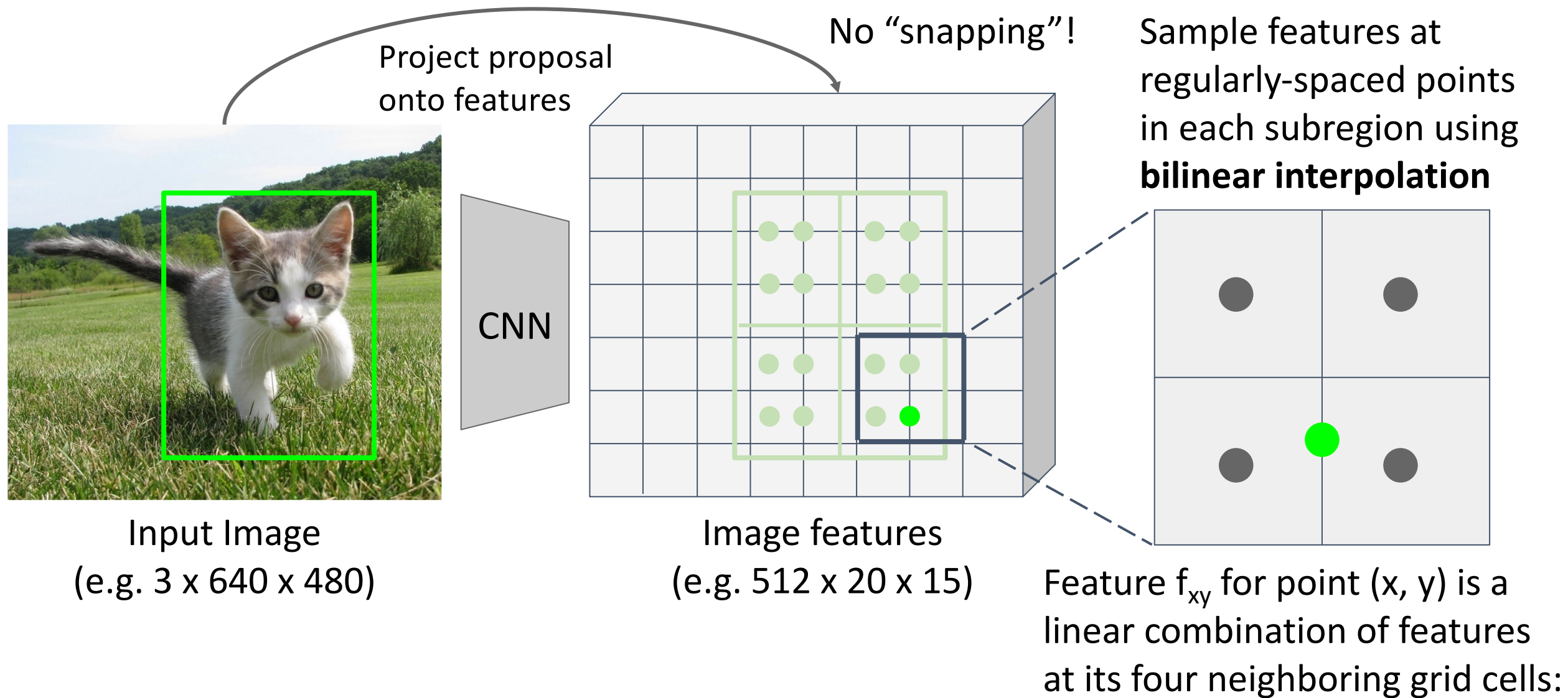
Divide into equal-sized subregions  
(may not be aligned to grid!)



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

# Cropping Features: RoI Align

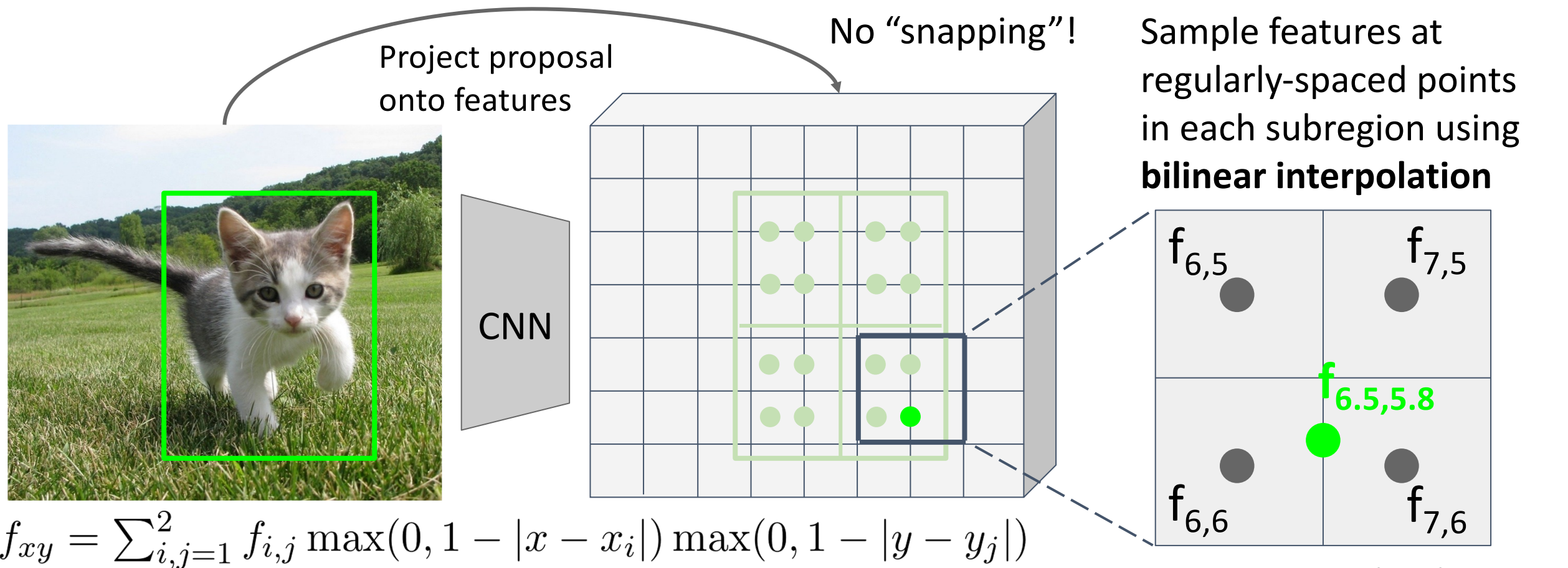
Divide into equal-sized subregions  
(may not be aligned to grid!)





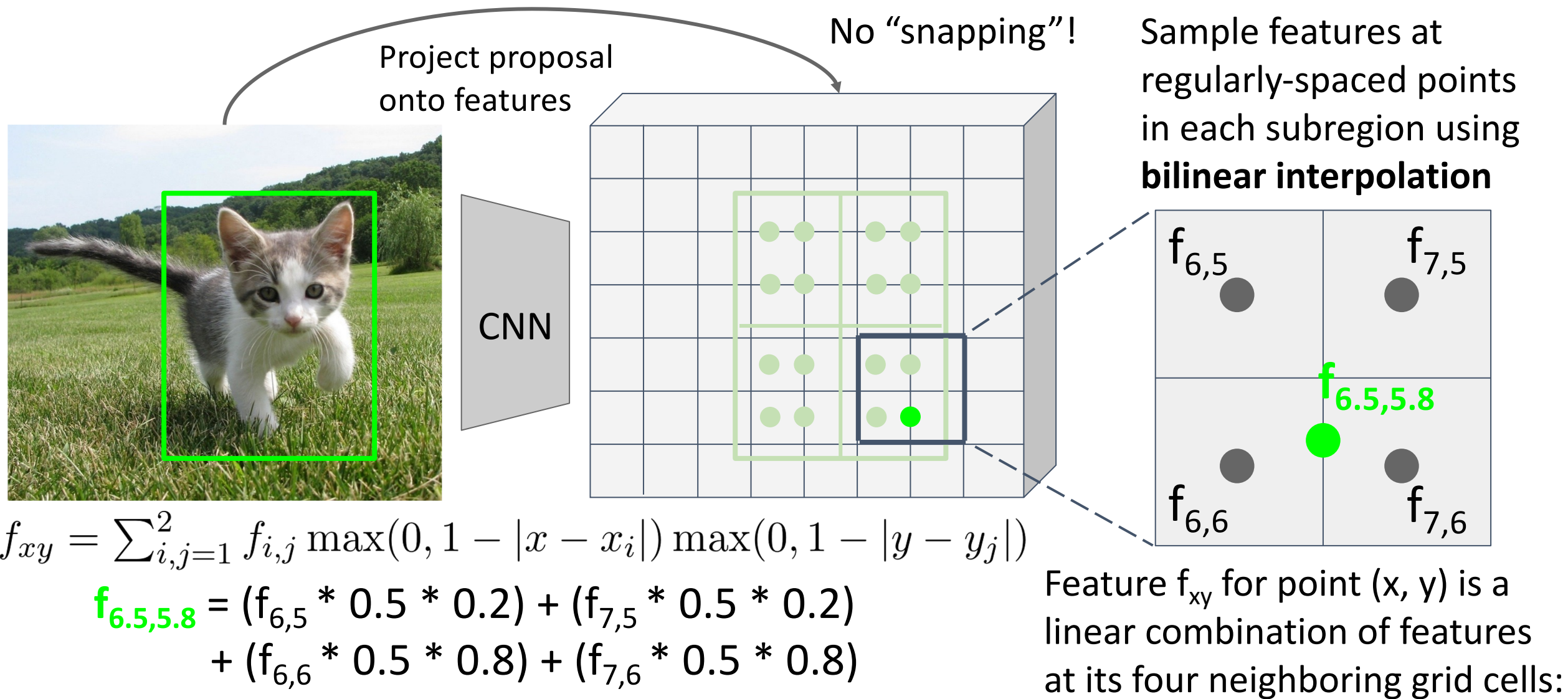
# Cropping Features: RoI Align

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



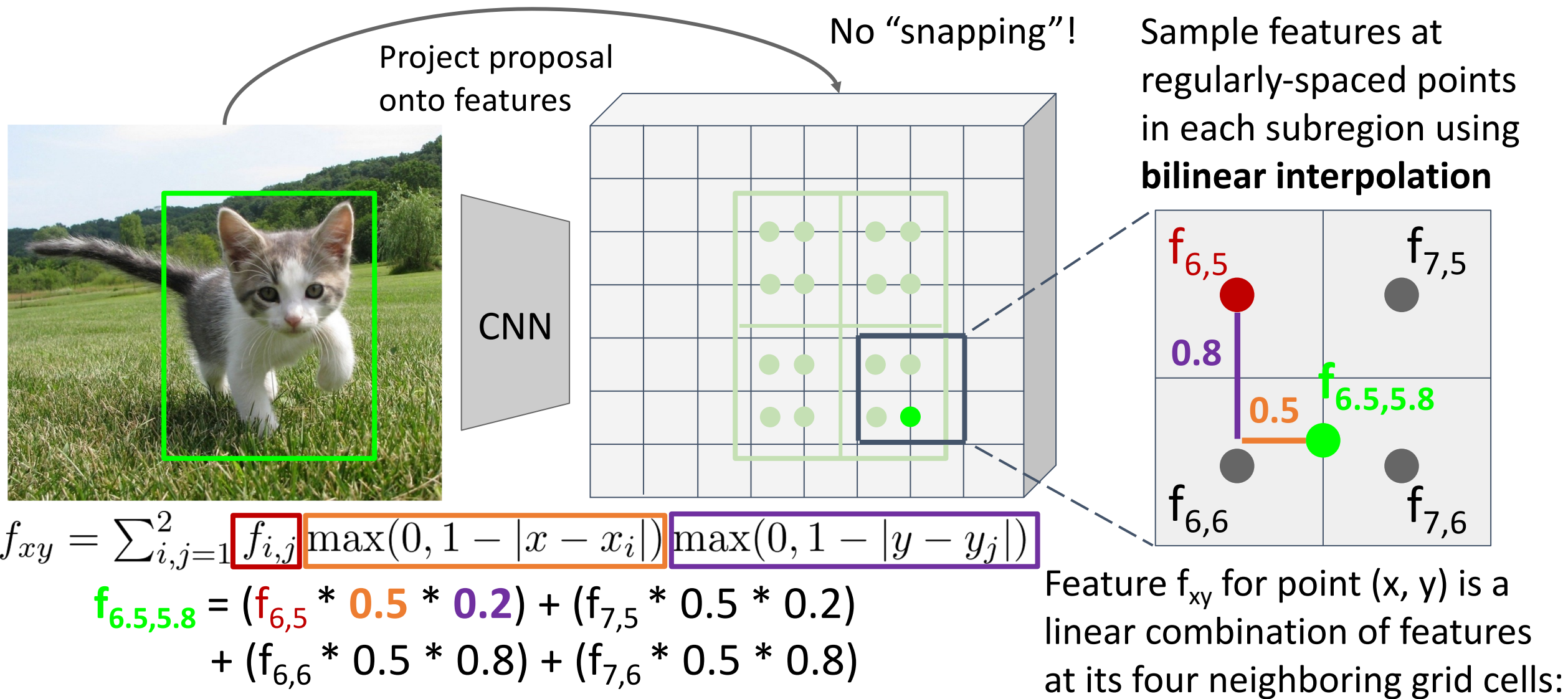
# Cropping Features: RoI Align

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



# Cropping Features: RoI Align

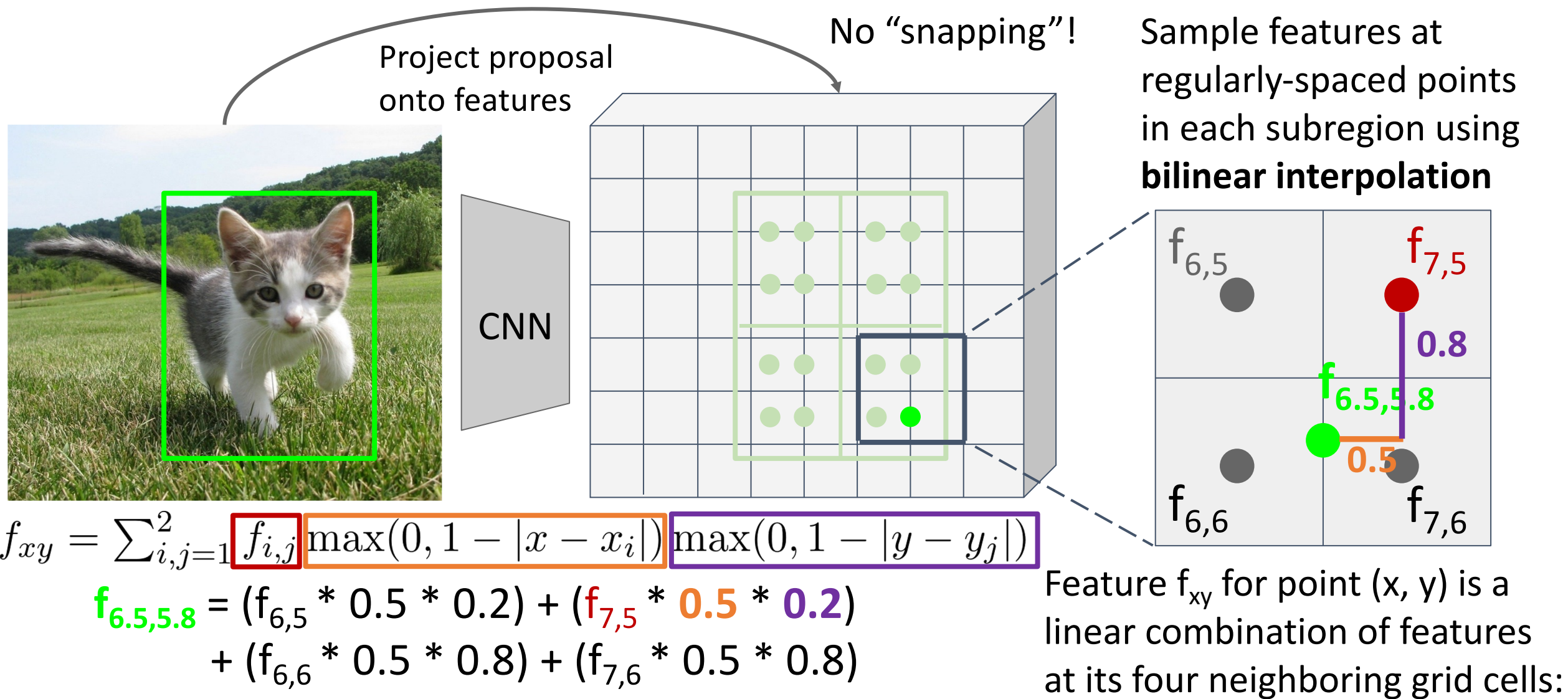
Divide into equal-sized subregions  
(may not be aligned to grid!)





# Cropping Features: RoI Align

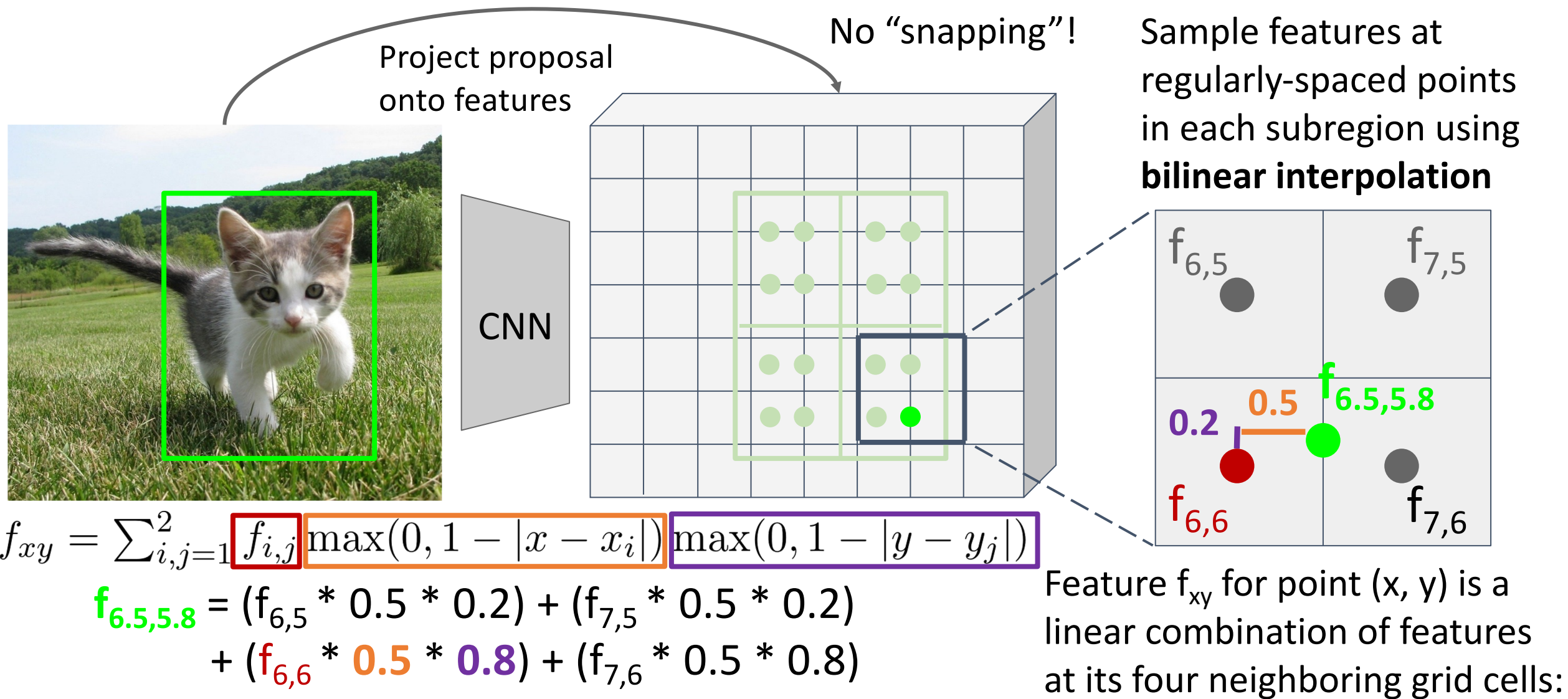
Divide into equal-sized subregions  
(may not be aligned to grid!)





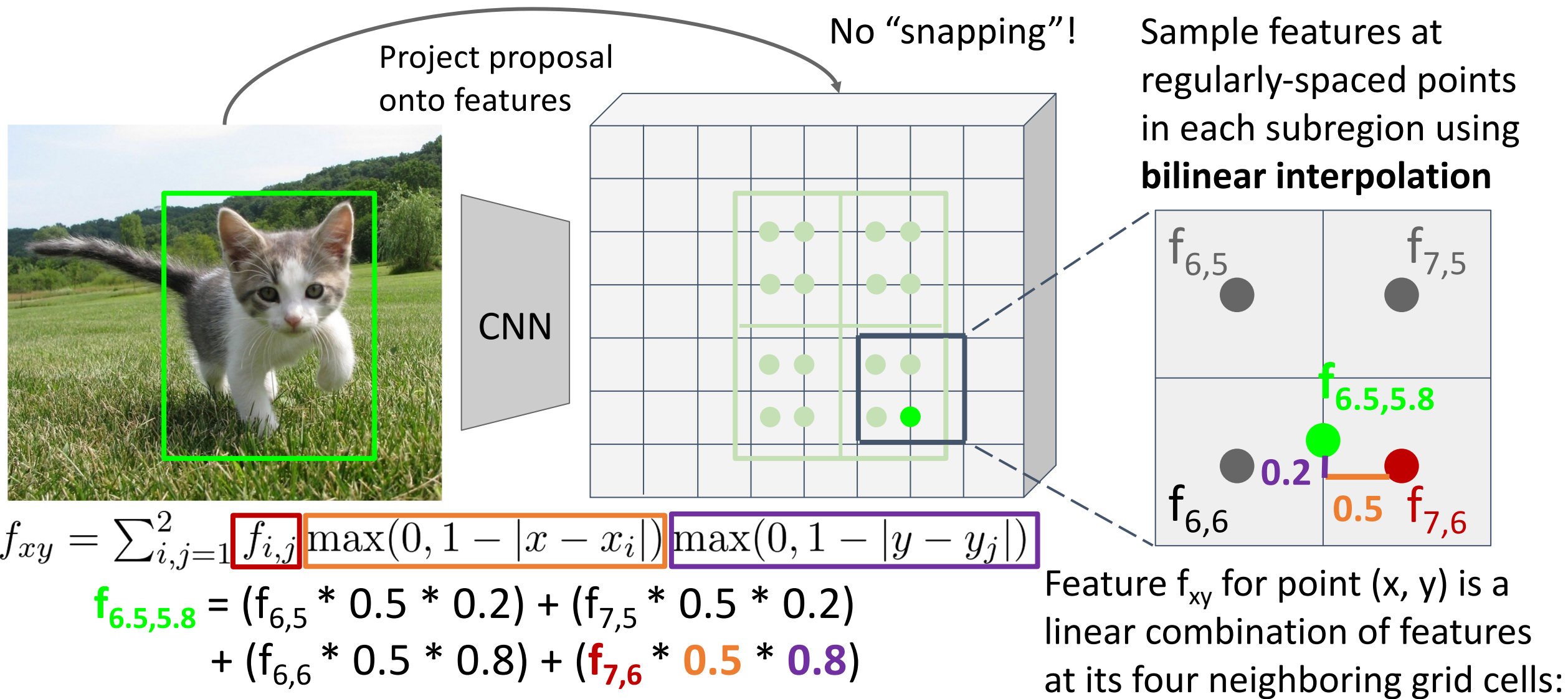
# Cropping Features: RoI Align

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



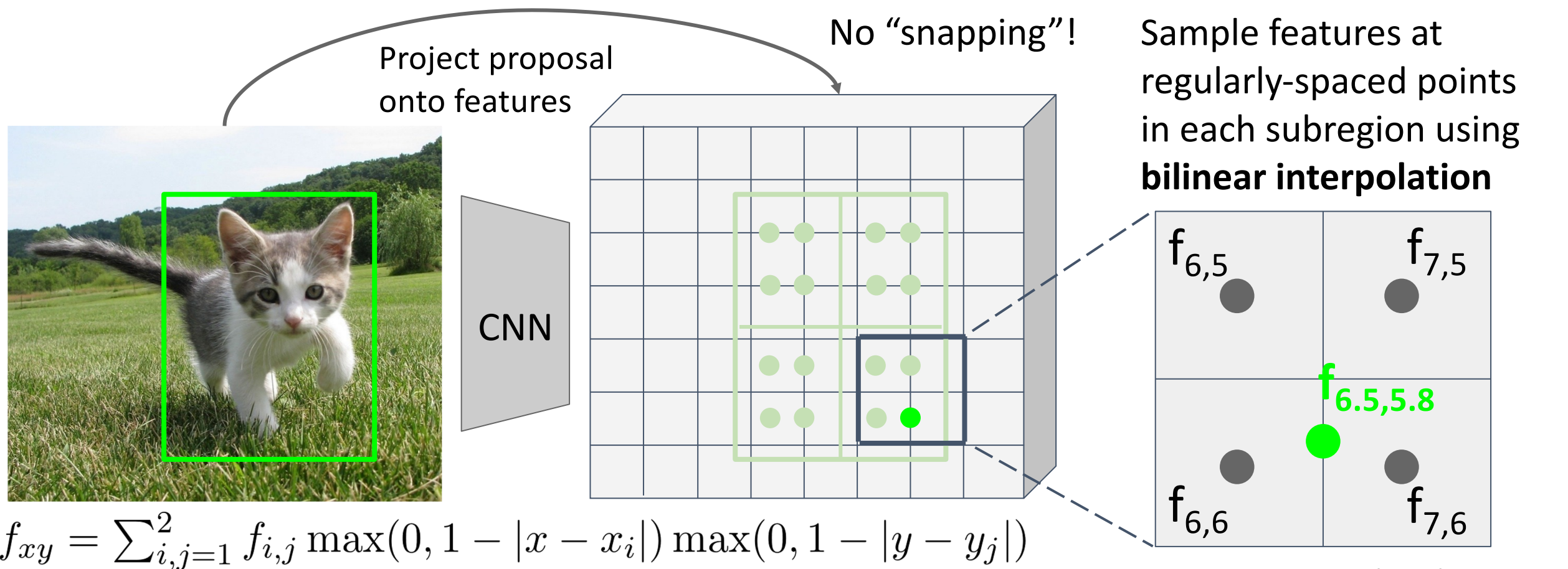
# Cropping Features: RoI Align

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



# Cropping Features: RoI Align

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



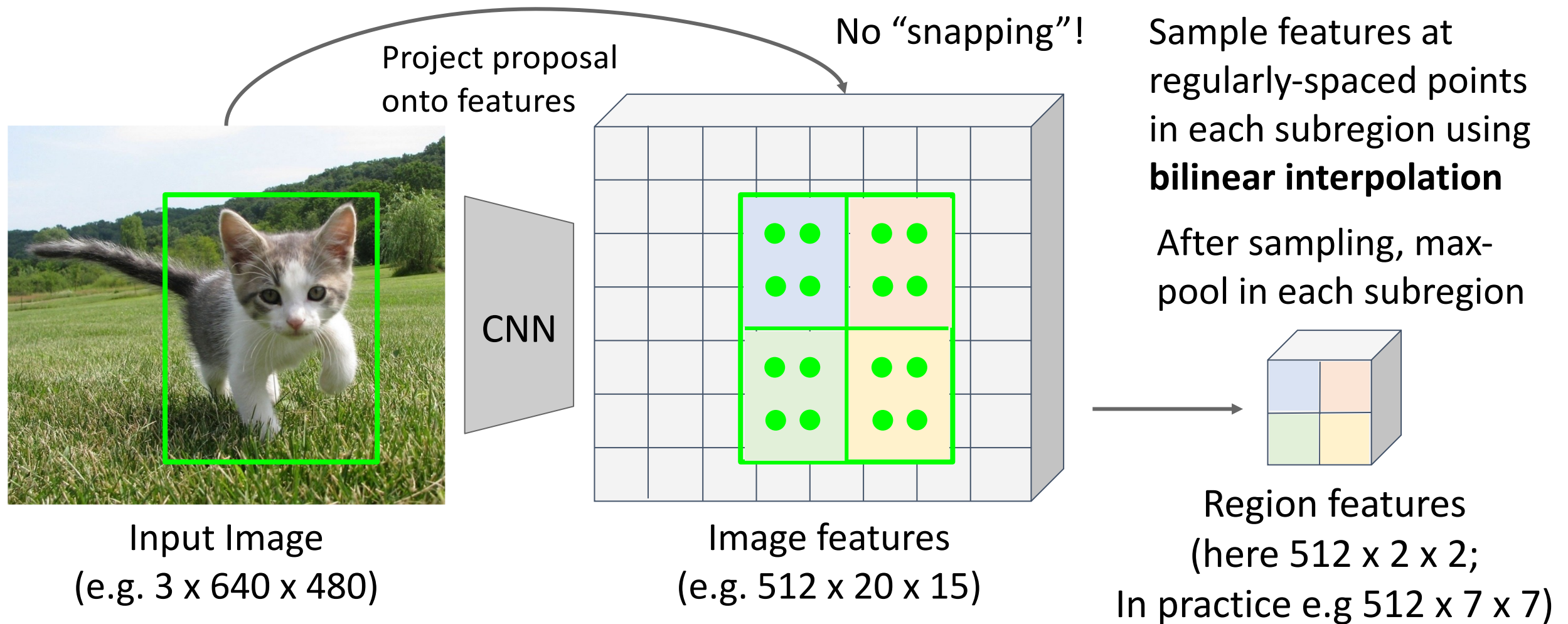
This is differentiable! Upstream gradient for sampled feature will flow backward into each of the four nearest-neighbor gridpoints

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



# Cropping Features: RoI Align

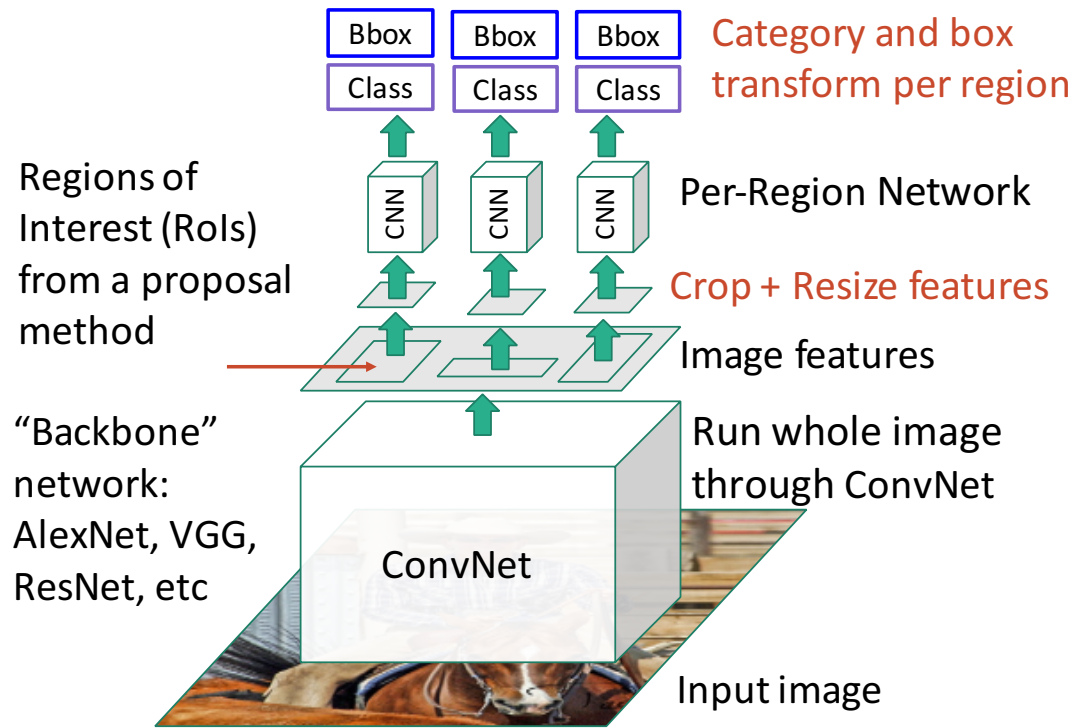
Divide into equal-sized subregions  
(may not be aligned to grid!)



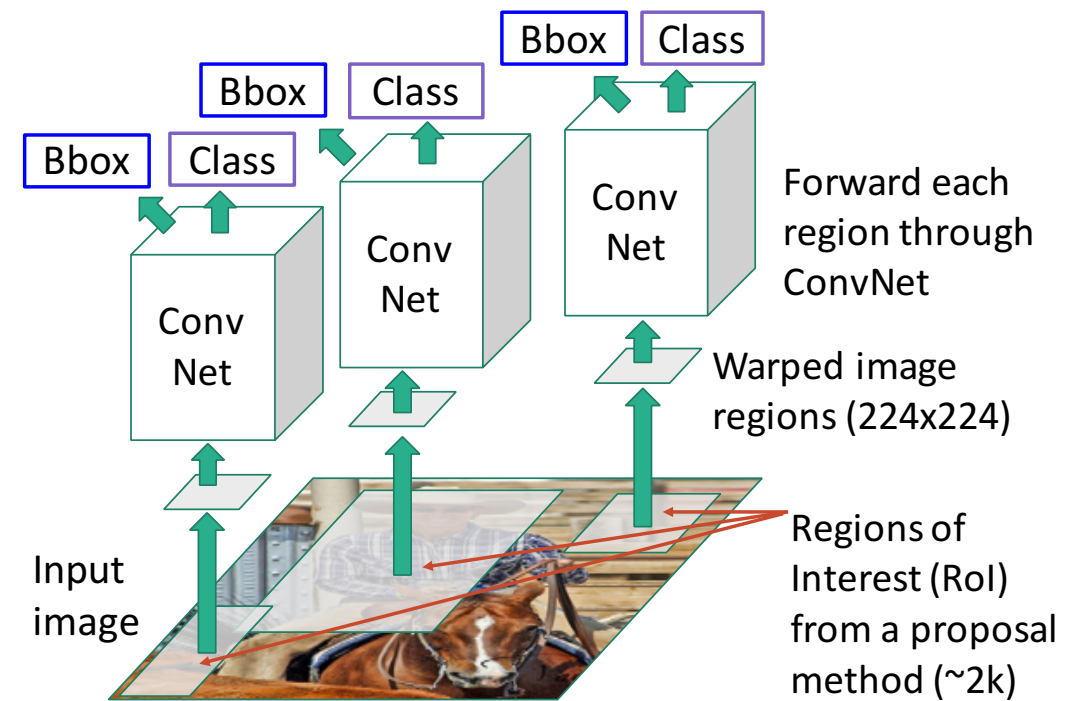


# Fast R-CNN vs “Slow” R-CNN

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

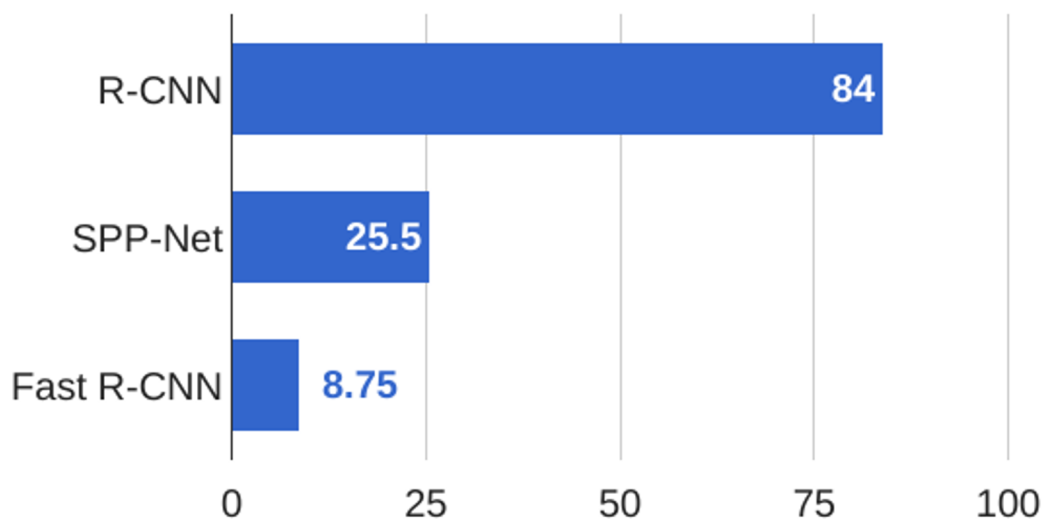


**“Slow” R-CNN:** Apply differentiable cropping to shared image features

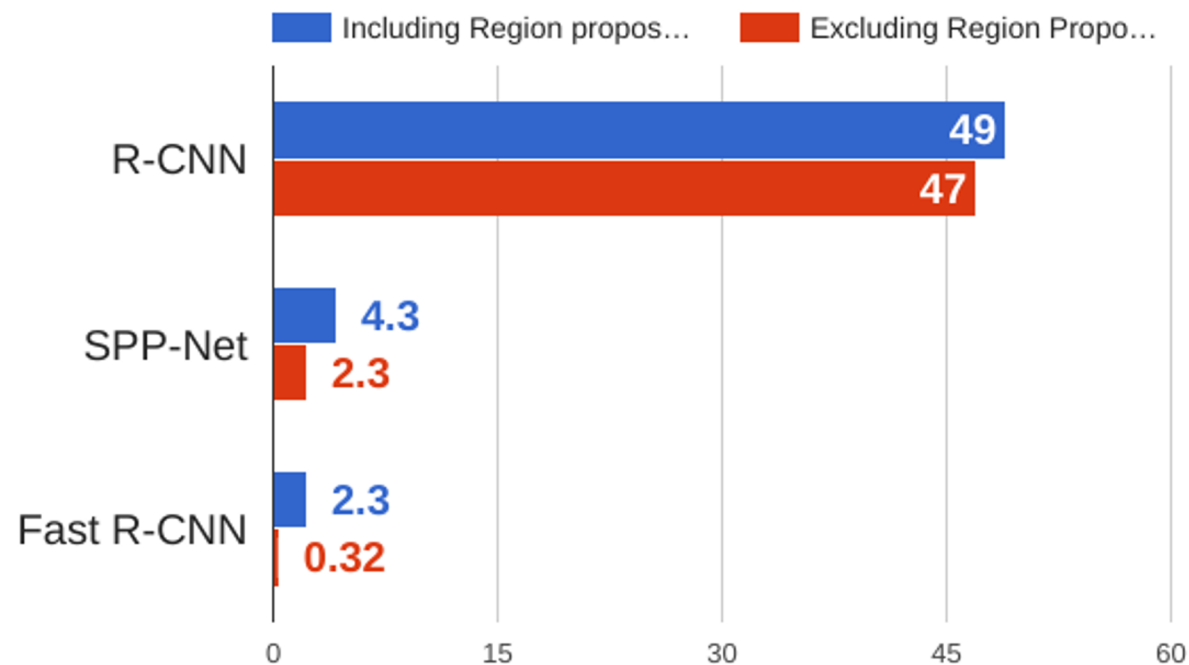


# Fast R-CNN vs “Slow” R-CNN

## Training time (Hours)



## Test time (seconds)



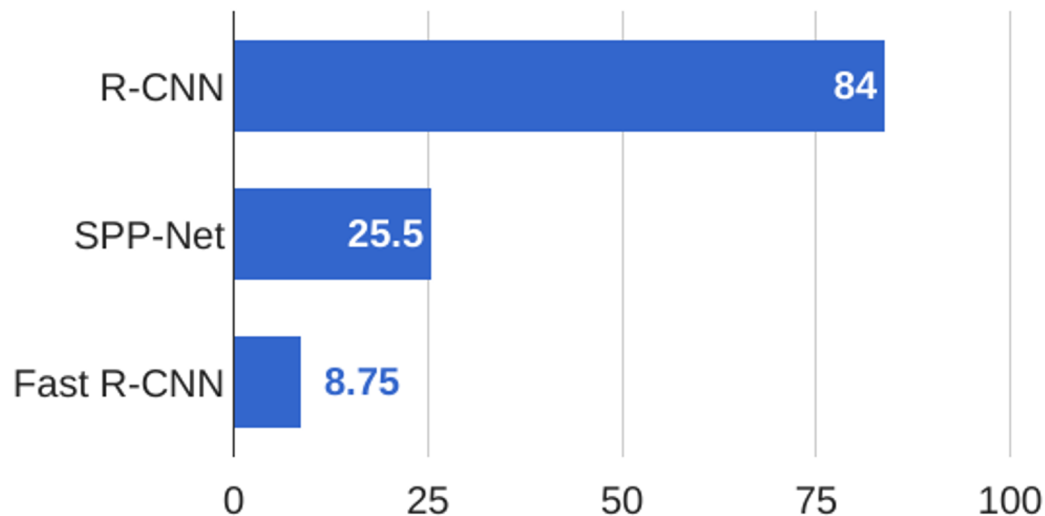
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

He et al, “Spatial pyramid pooling in deep convolutional networks for visual recognition”, ECCV 2014

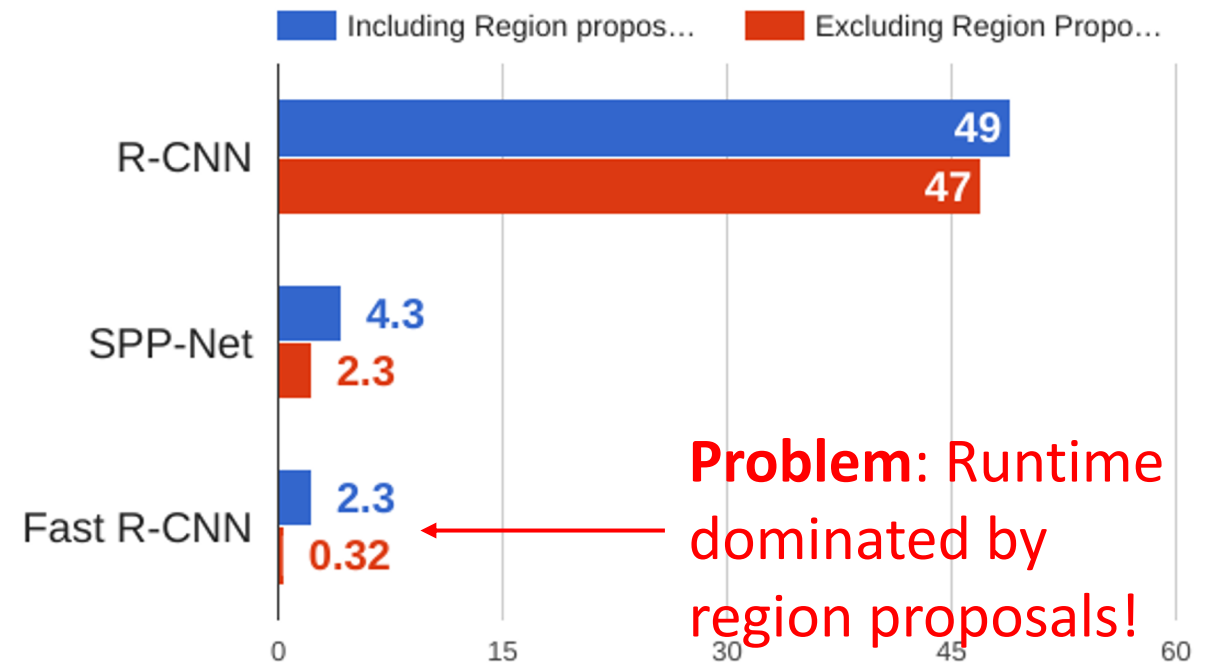
Girshick, “Fast R-CNN”, ICCV 2015

# Fast R-CNN vs “Slow” R-CNN

## Training time (Hours)



## Test time (seconds)



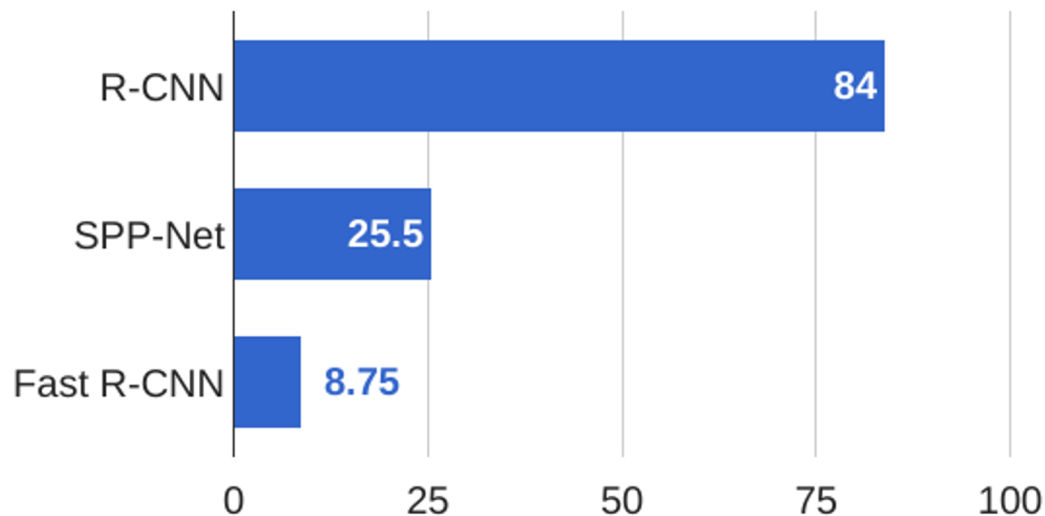
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

He et al, “Spatial pyramid pooling in deep convolutional networks for visual recognition”, ECCV 2014

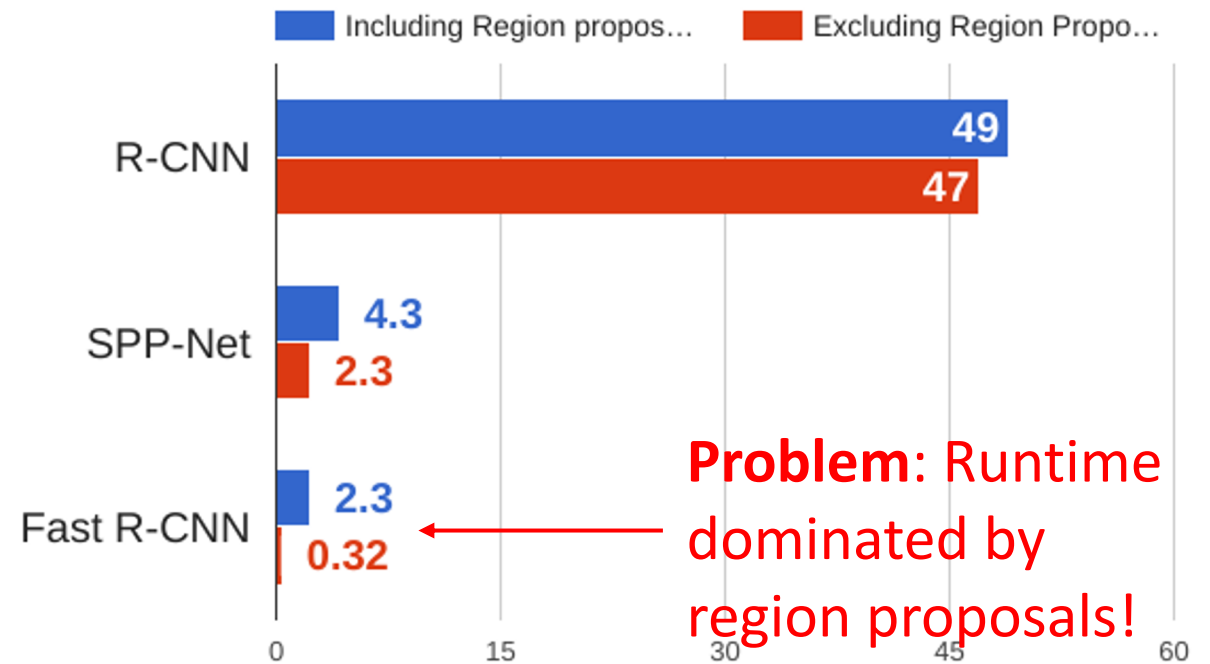
Girshick, “Fast R-CNN”, ICCV 2015

# Fast R-CNN vs “Slow” R-CNN

## Training time (Hours)



## Test time (seconds)



**Problem:** Runtime dominated by region proposals!

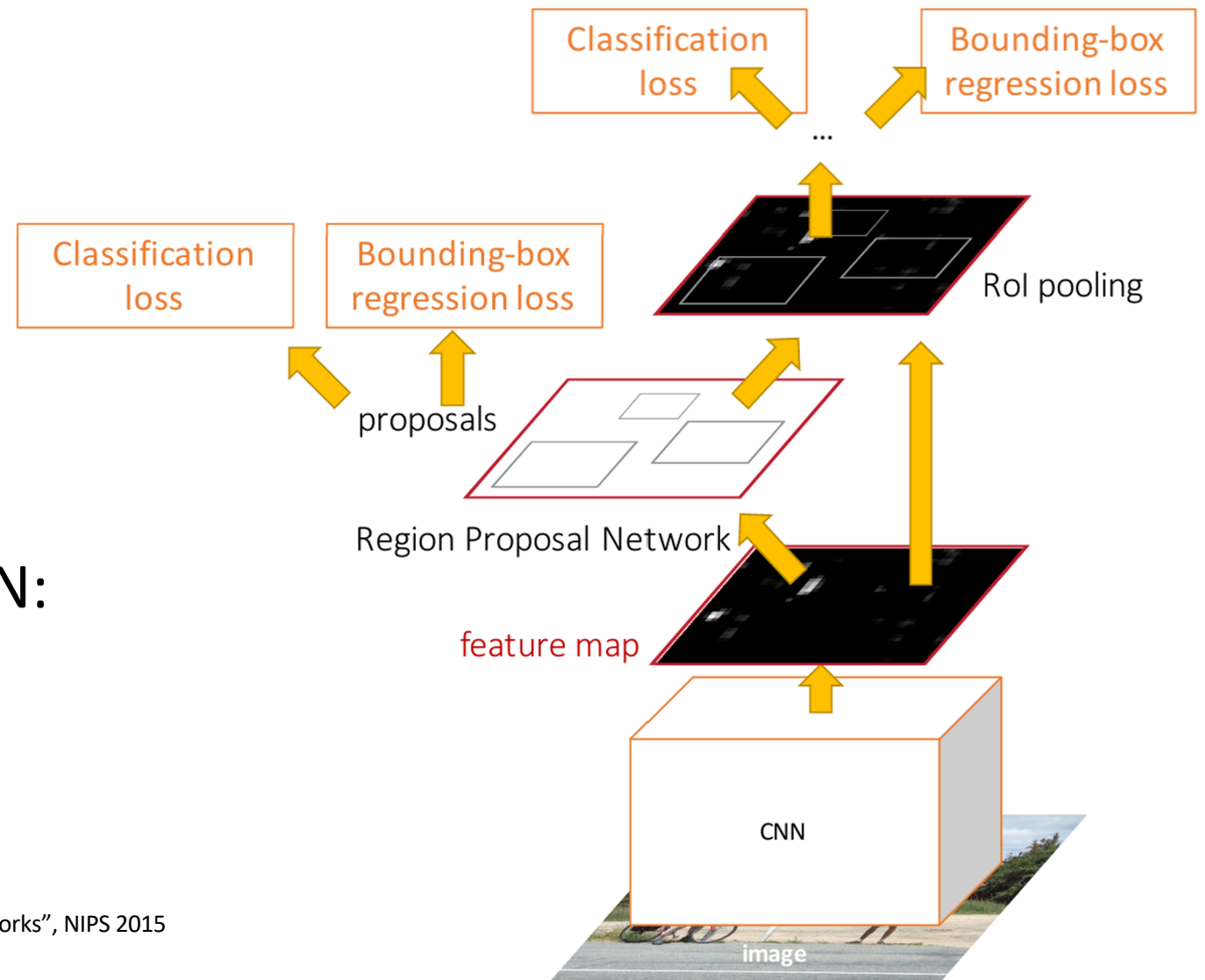
**Recall:** Region proposals computed by heuristic “Selective Search” algorithm on CPU -- let’s learn them with a CNN instead!



# 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)

CNN

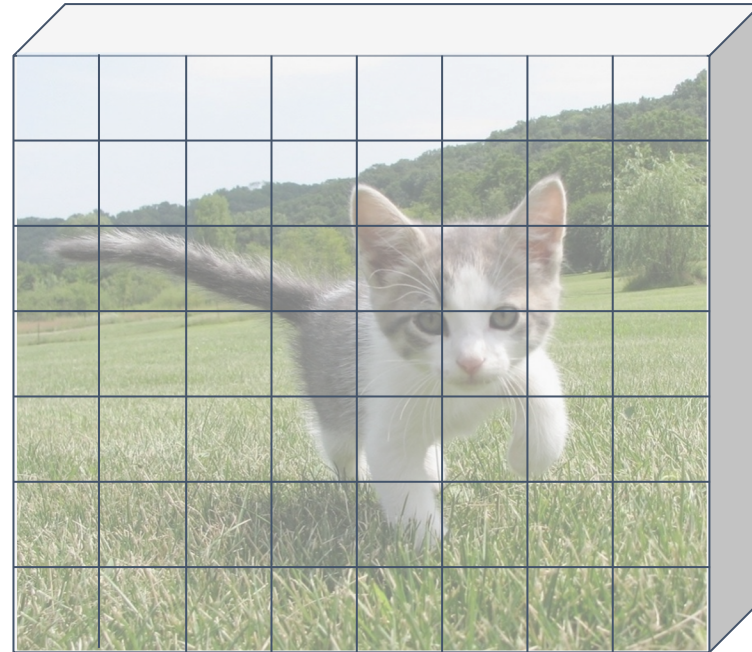


Image features  
(e.g. 512 x 20 x 15)

# Region Proposal Network (RPN)

Run backbone CNN to get  
features aligned to input image



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

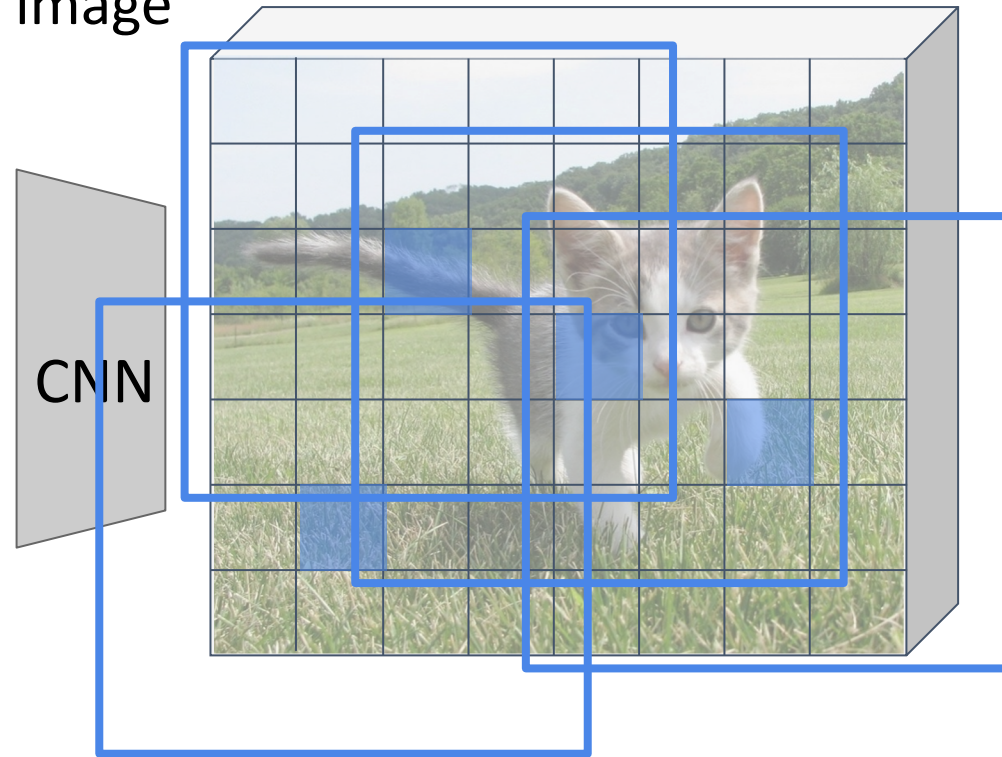


Image features  
(e.g. 512 x 20 x 15)

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



# Region Proposal Network (RPN)

Run backbone CNN to get features aligned to input image



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

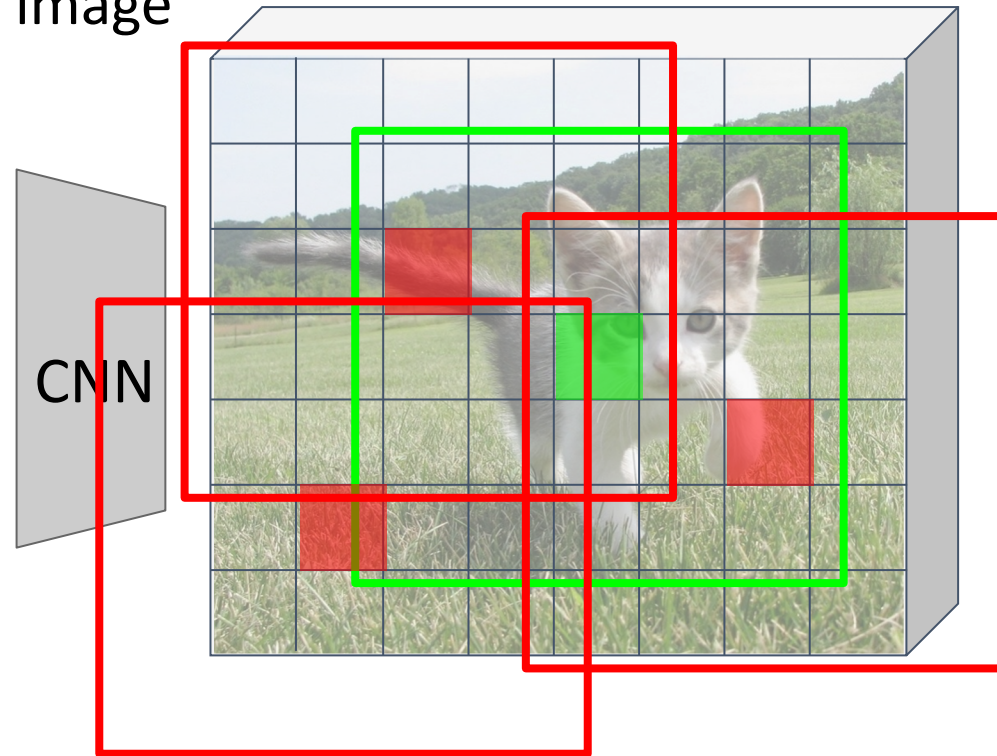


Image features  
(e.g. 512 x 20 x 15)

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

Anchor is an object?  
1 x 20 x 15

At each point, predict whether the corresponding anchor contains an object (per-cell logistic regression, predict scores with conv layer)



# Region Proposal Network (RPN)

Run backbone CNN to get features aligned to input image



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

CNN

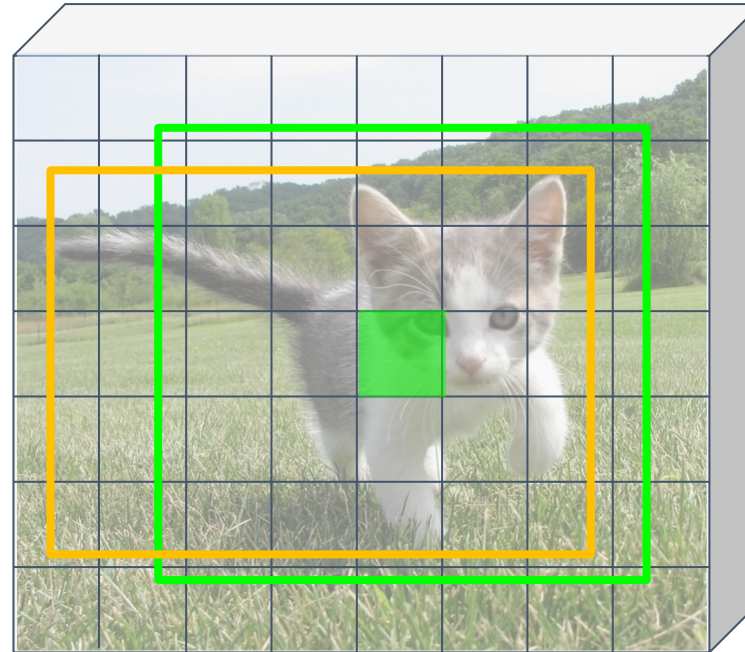


Image features  
(e.g. 512 x 20 x 15)

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

Conv

Anchor is an object?  
1 x 20 x 15

Box transforms  
4 x 20 x 15

For positive boxes, also predict a box transform to regress from **anchor box** to **object box**

# Region Proposal Network (RPN)

Run backbone CNN to get features aligned to input image



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

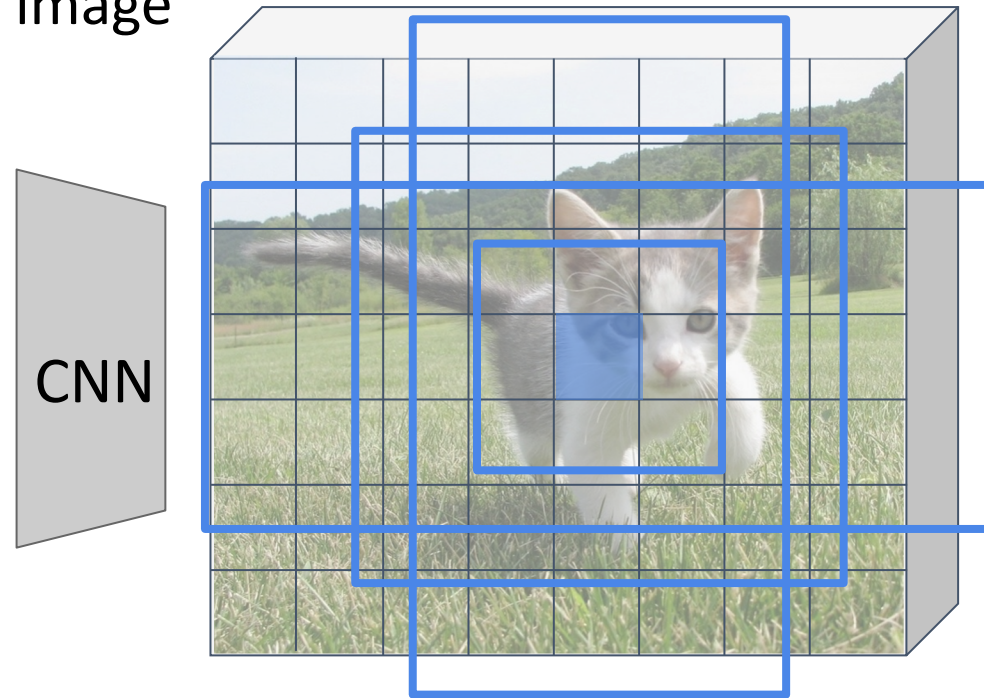


Image features  
(e.g. 512 x 20 x 15)

**Problem:** Anchor box may have the wrong size / shape

**Solution:** Use **K** different anchor boxes at each point!



Anchor is an object?

**K** x 20 x 15

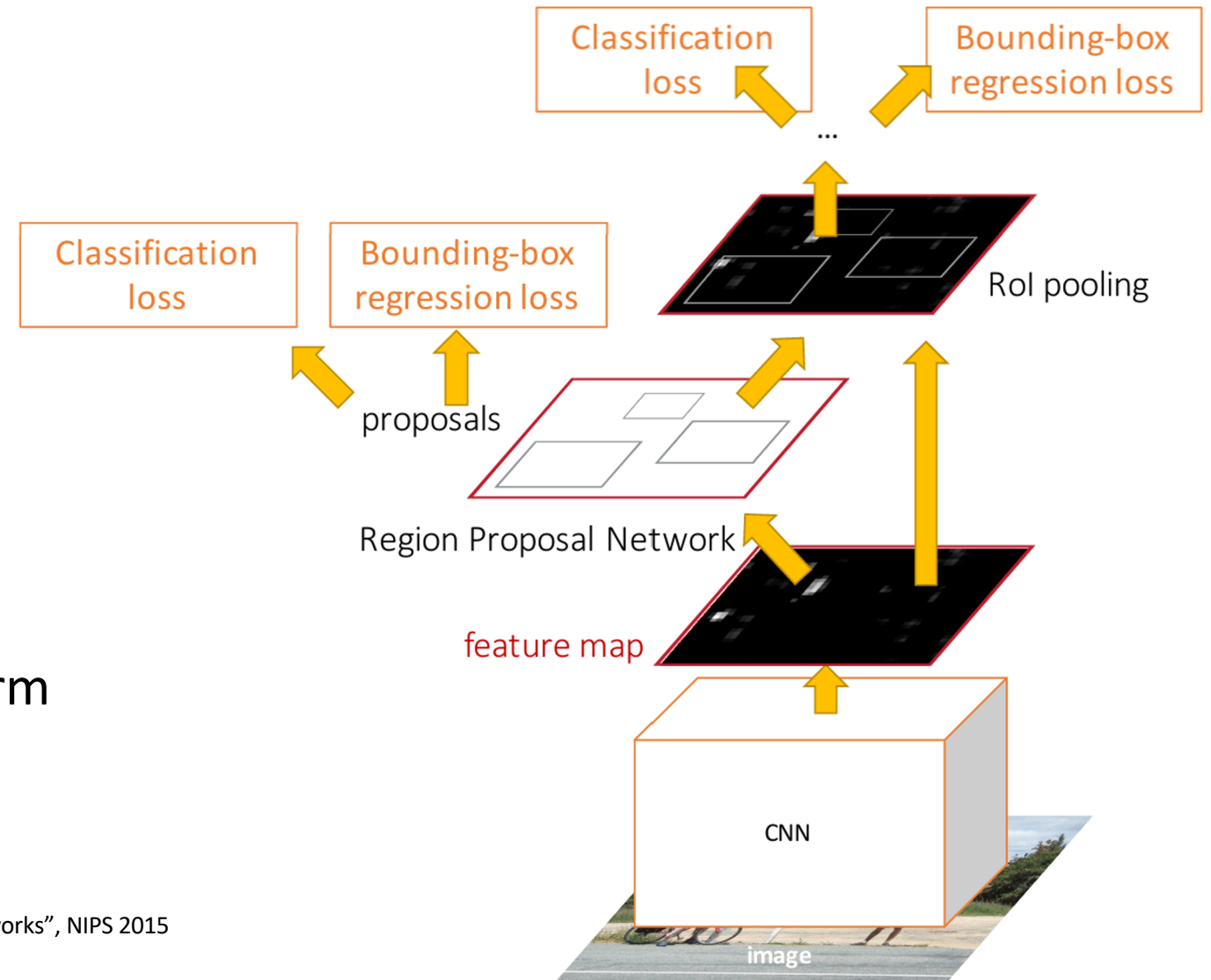
Box transforms  
**4K** x 20 x 15

At test time: sort all  $K \times 20 \times 15$  boxes by their score, and take the top  $\sim 300$  as our region proposals

# Faster R-CNN: Learnable Region Proposals

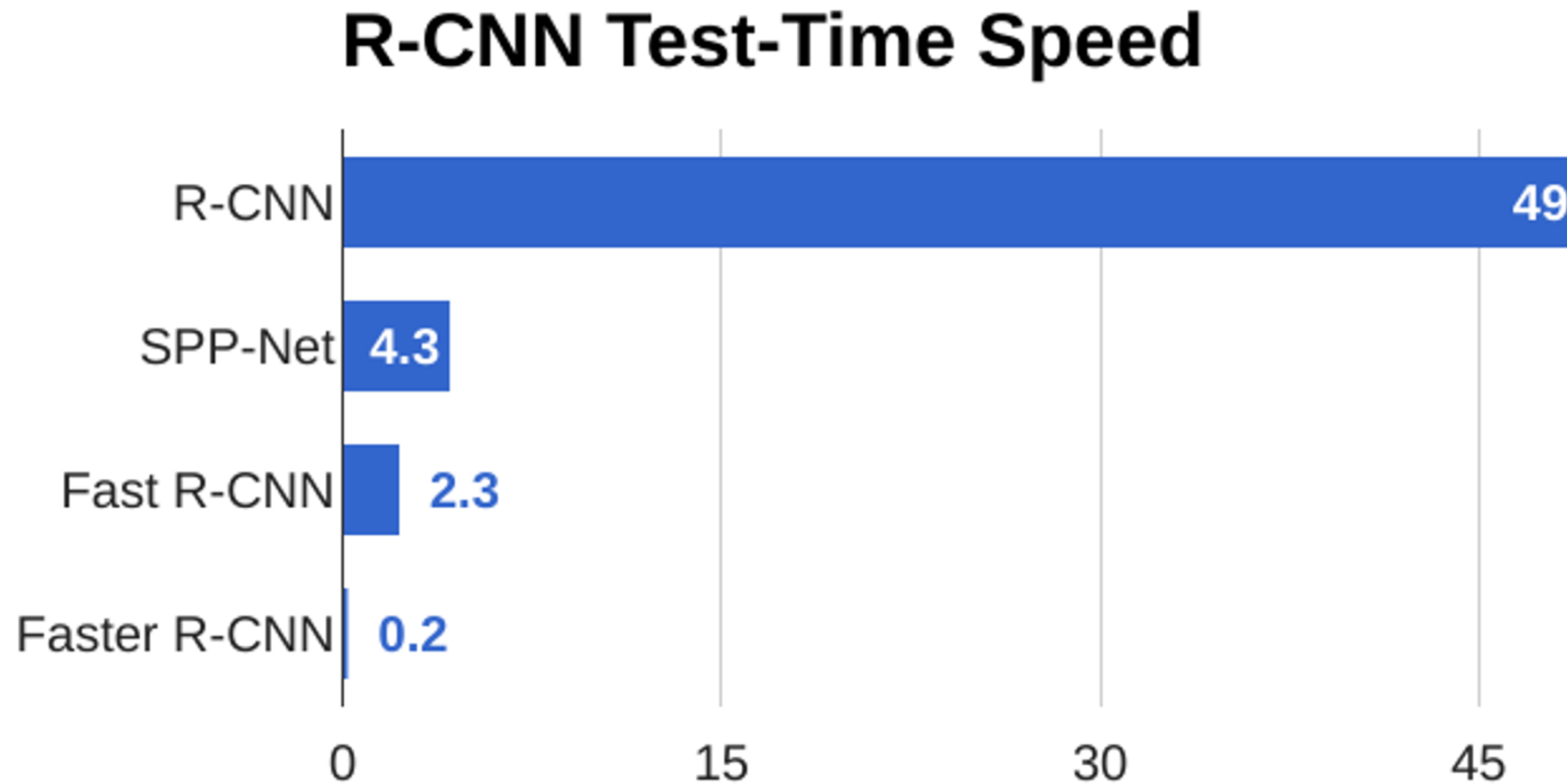
Jointly train with 4 losses:

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





# Faster R-CNN: Learnable Region Proposals

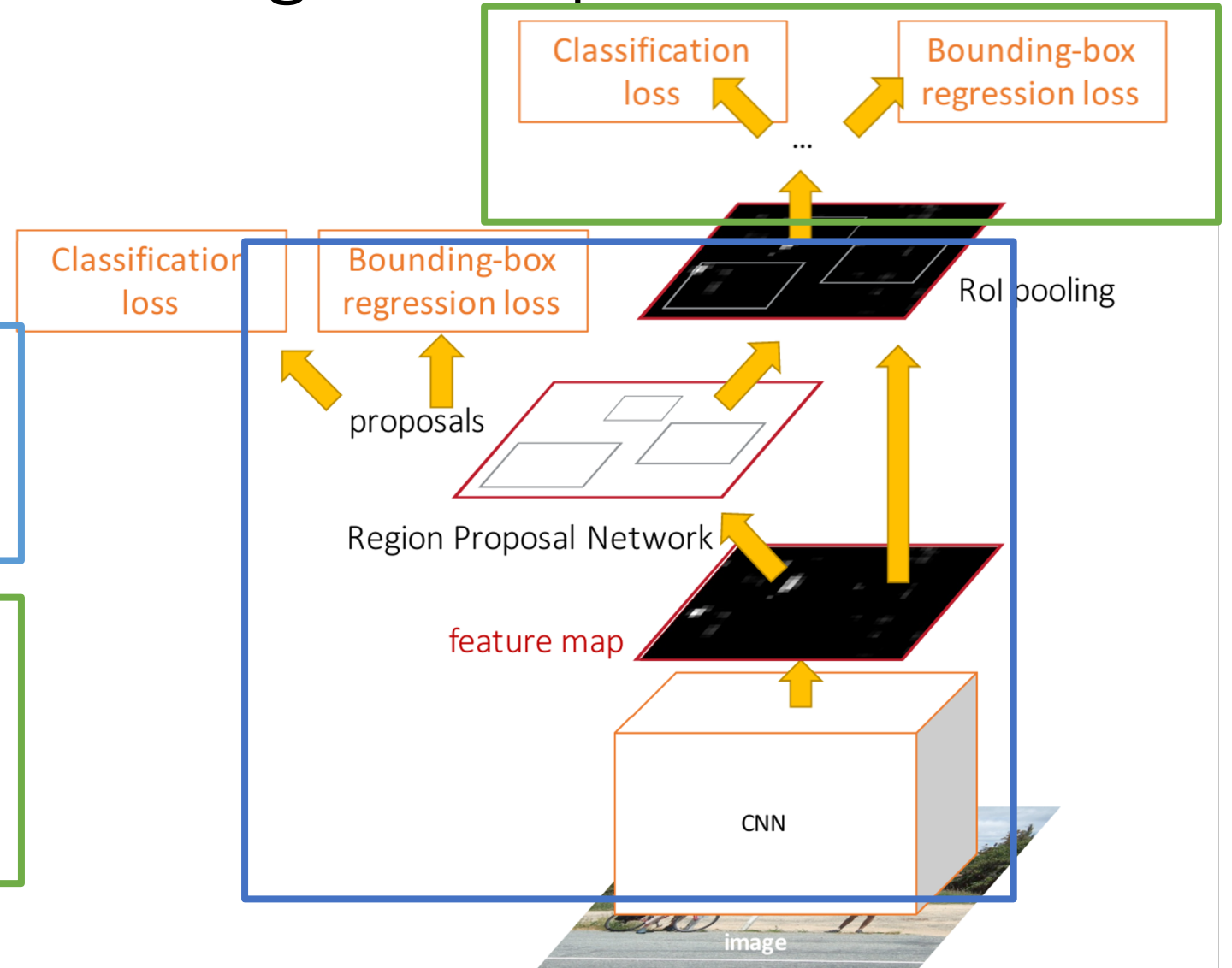
Faster R-CNN is a  
**Two-stage object detector**

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



# Faster R-CNN: Learnable Region Proposals

Question: Do we really need the second stage?

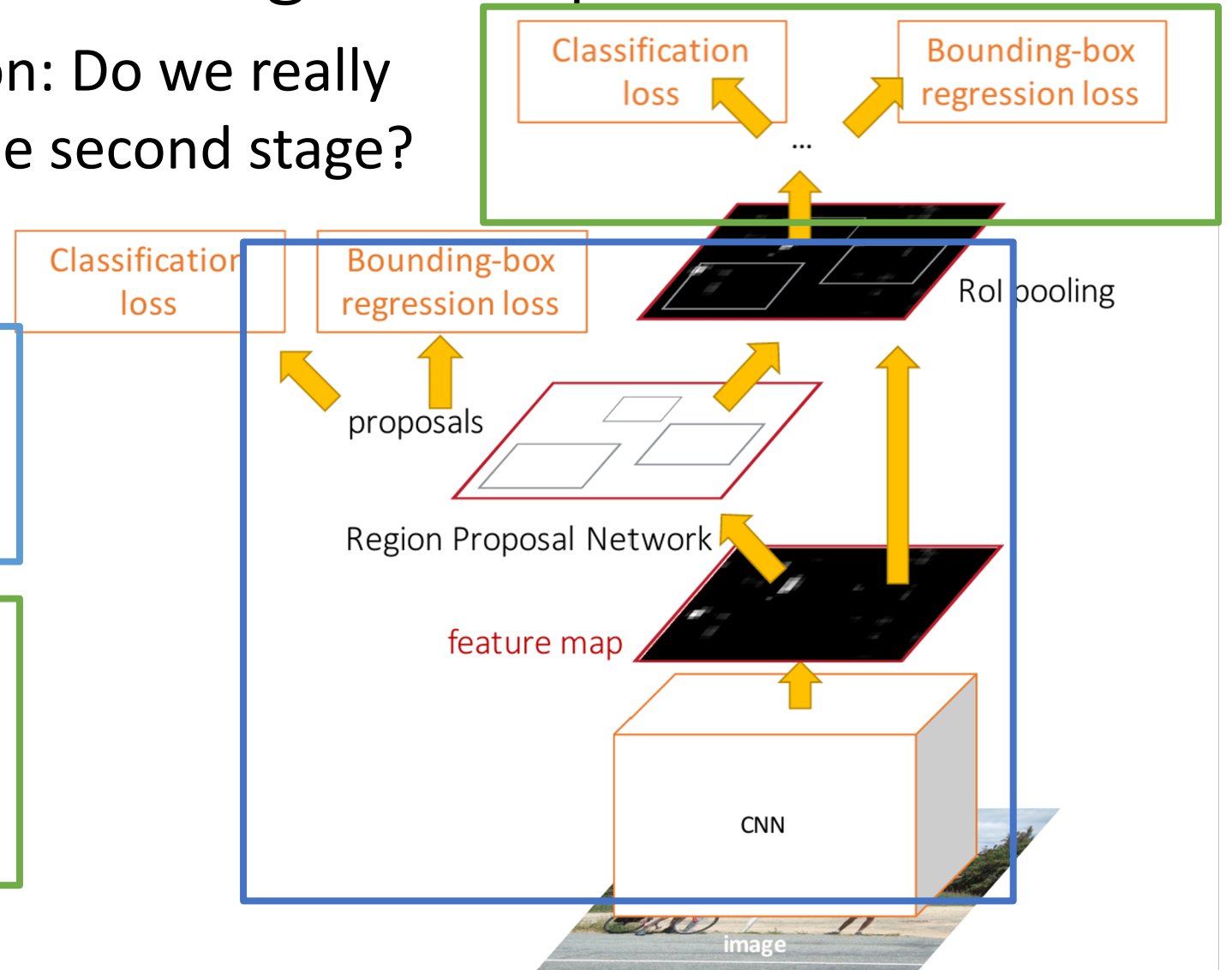
Faster R-CNN is a  
**Two-stage object detector**

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 Object Detection

Run backbone CNN to get features aligned to input image



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

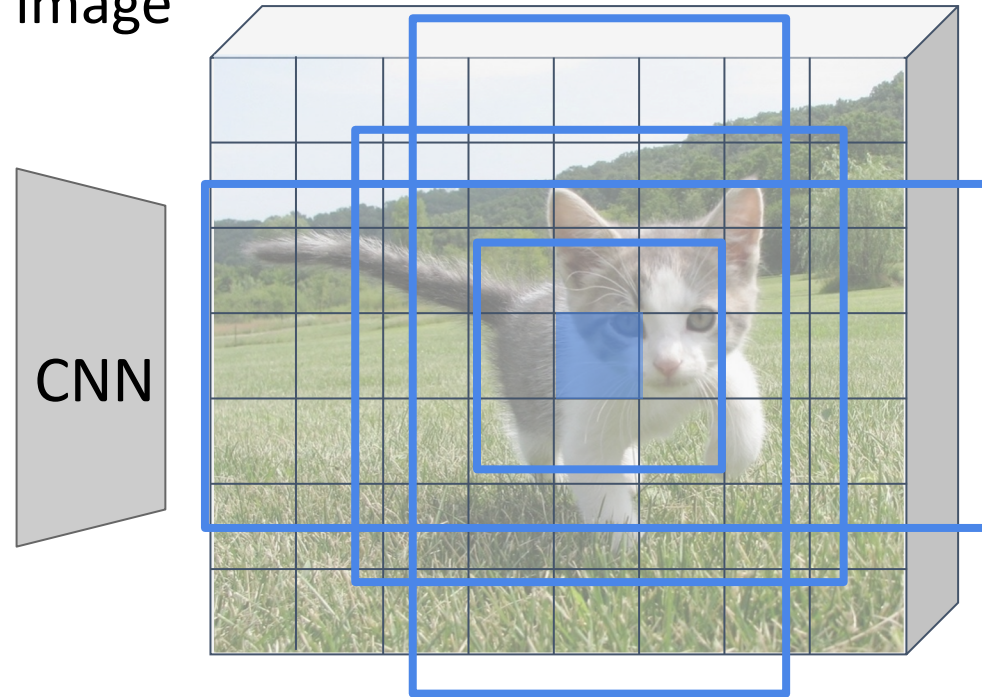
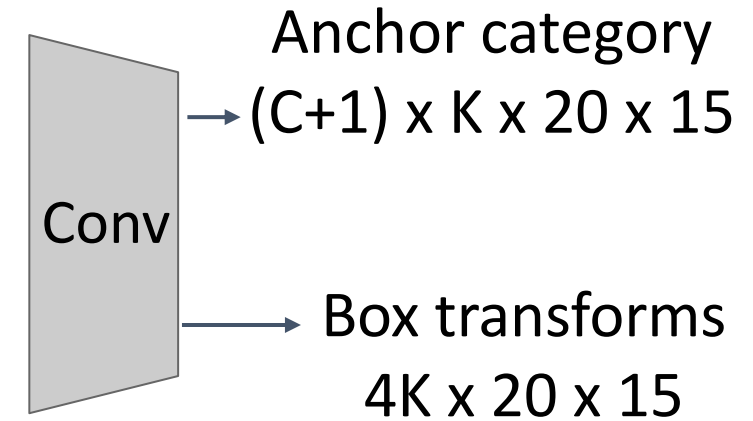


Image features  
(e.g. 512 x 20 x 15)

**RPN:** Classify each anchor as object / not object

**Single-Stage Detector:** Classify each object as one of C categories (or background)



Remember: K anchors at each position in image feature map



# Single-Stage Object Detection

Run backbone CNN to get features aligned to input image



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

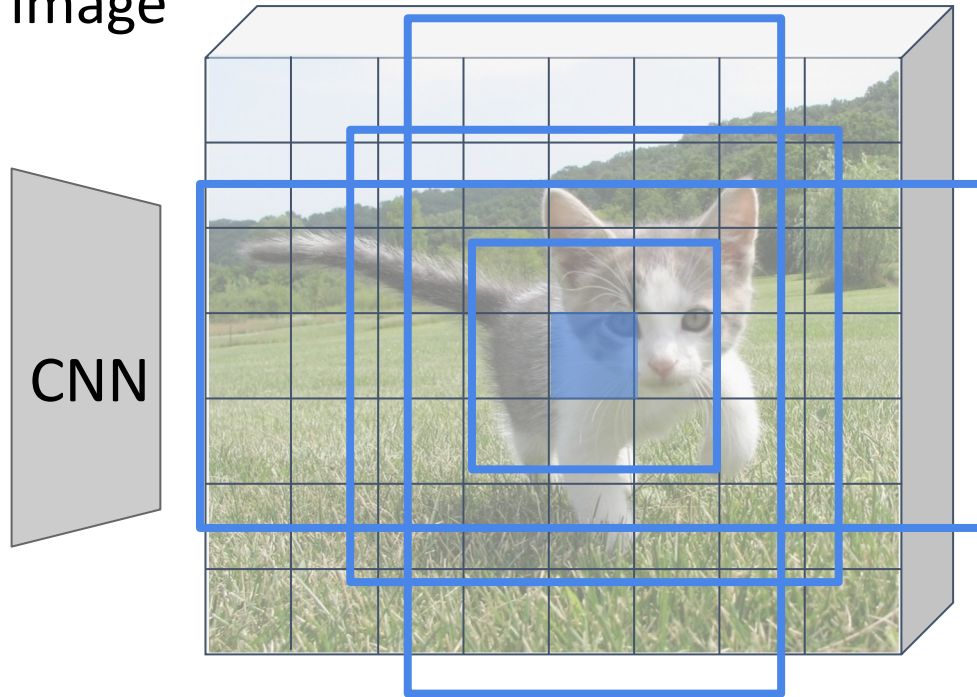


Image features  
(e.g. 512 x 20 x 15)

**RPN:** Classify each anchor as object / not object

**Single-Stage Detector:** Classify each object as one of C categories (or background)

Anchor category  
→  $(C+1) \times K \times 20 \times 15$

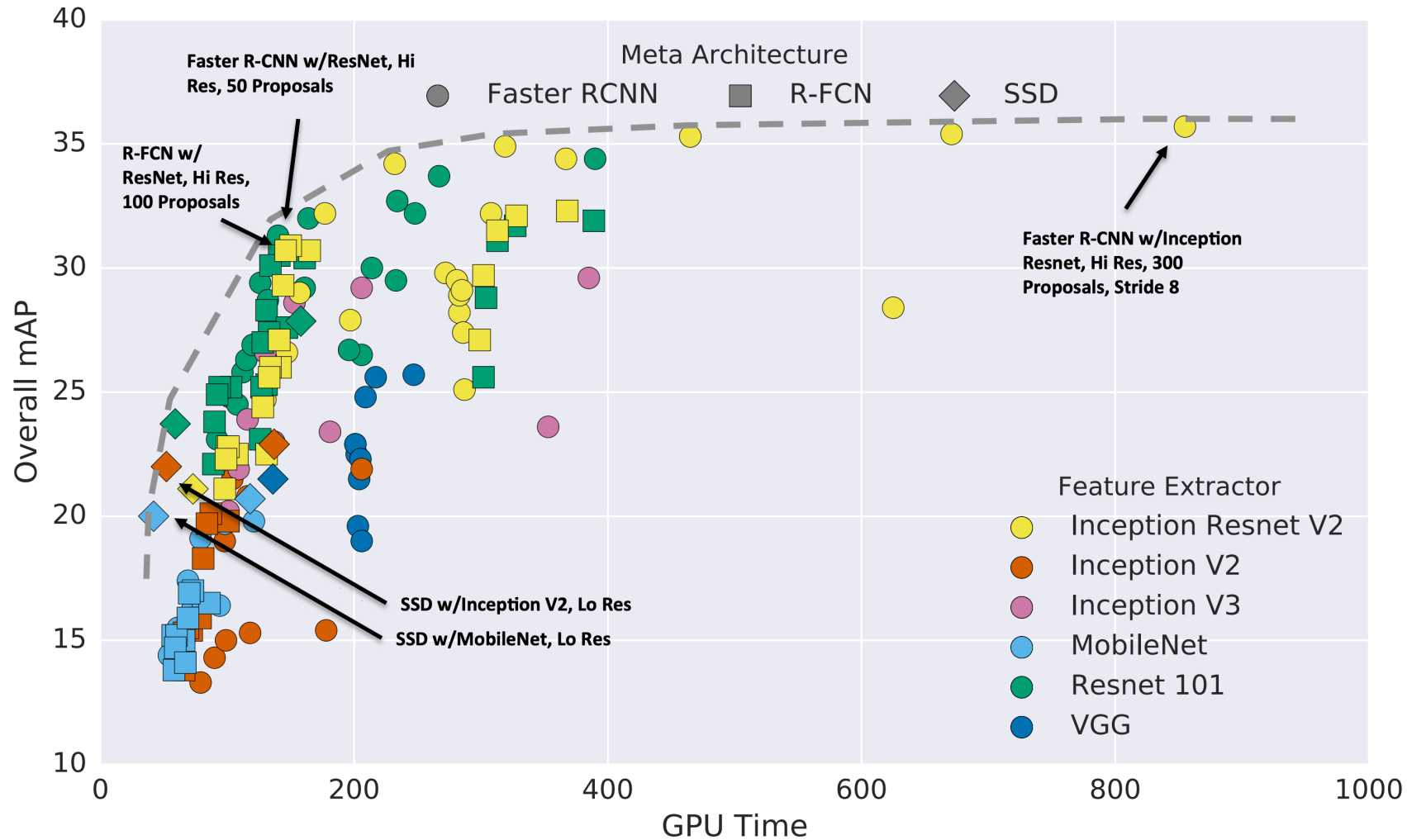
Conv  
→ Box transforms  
 $C \times 4K \times 20 \times 15$

Sometimes use **category-specific regression**: Predict different box transforms for each category

Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016  
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016  
Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

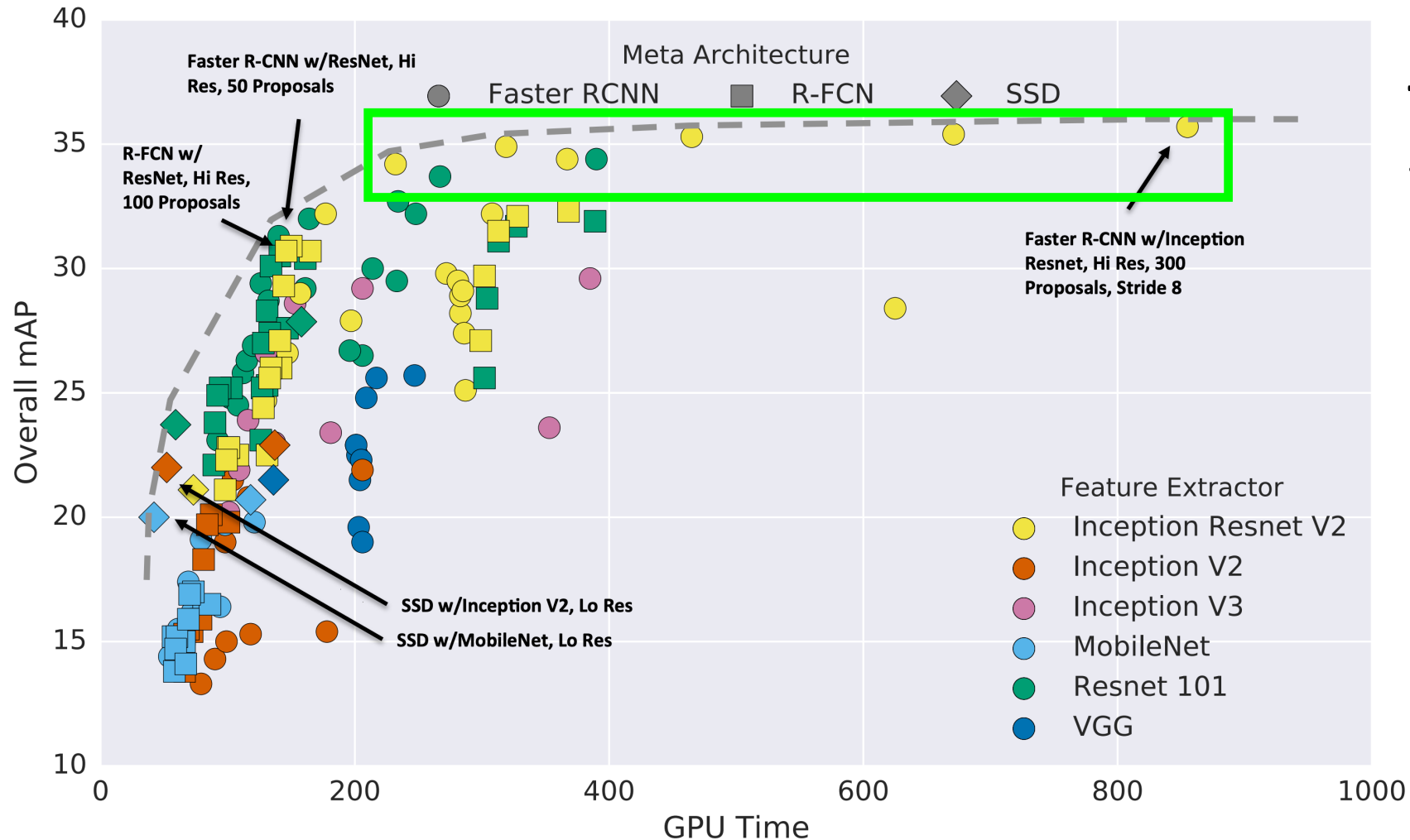


# Object Detection: Lots of variables!



Huang et al, "Speed/accuracy trade-offs for modern convolutional object detectors", CVPR 2017

# Object Detection: Lots of variables!

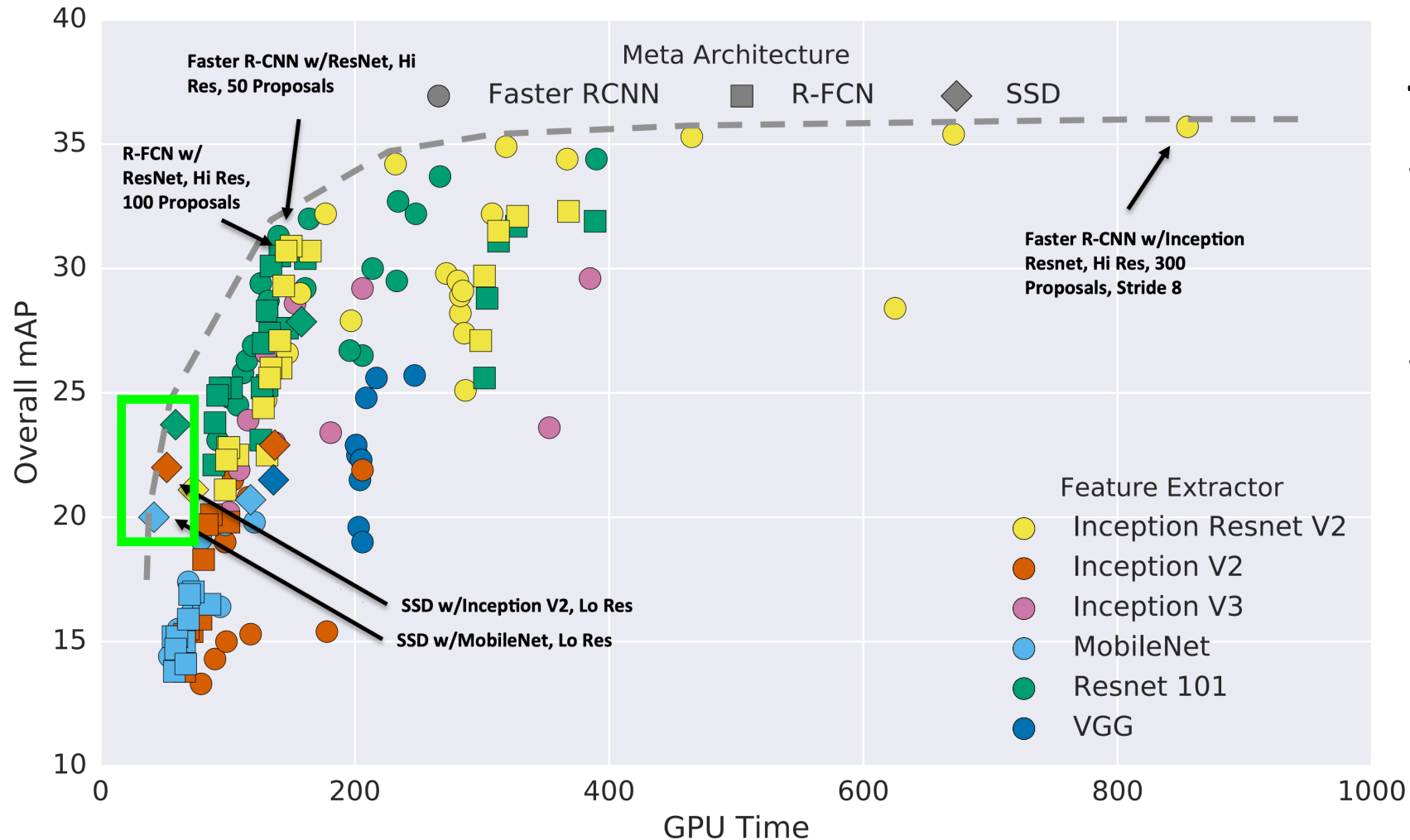


## Takeaways:

- Two stage method (Faster R-CNN) get the best accuracy, but are slower

Huang et al, "Speed/accuracy trade-offs for modern convolutional object detectors", CVPR 2017

# Object Detection: Lots of variables!

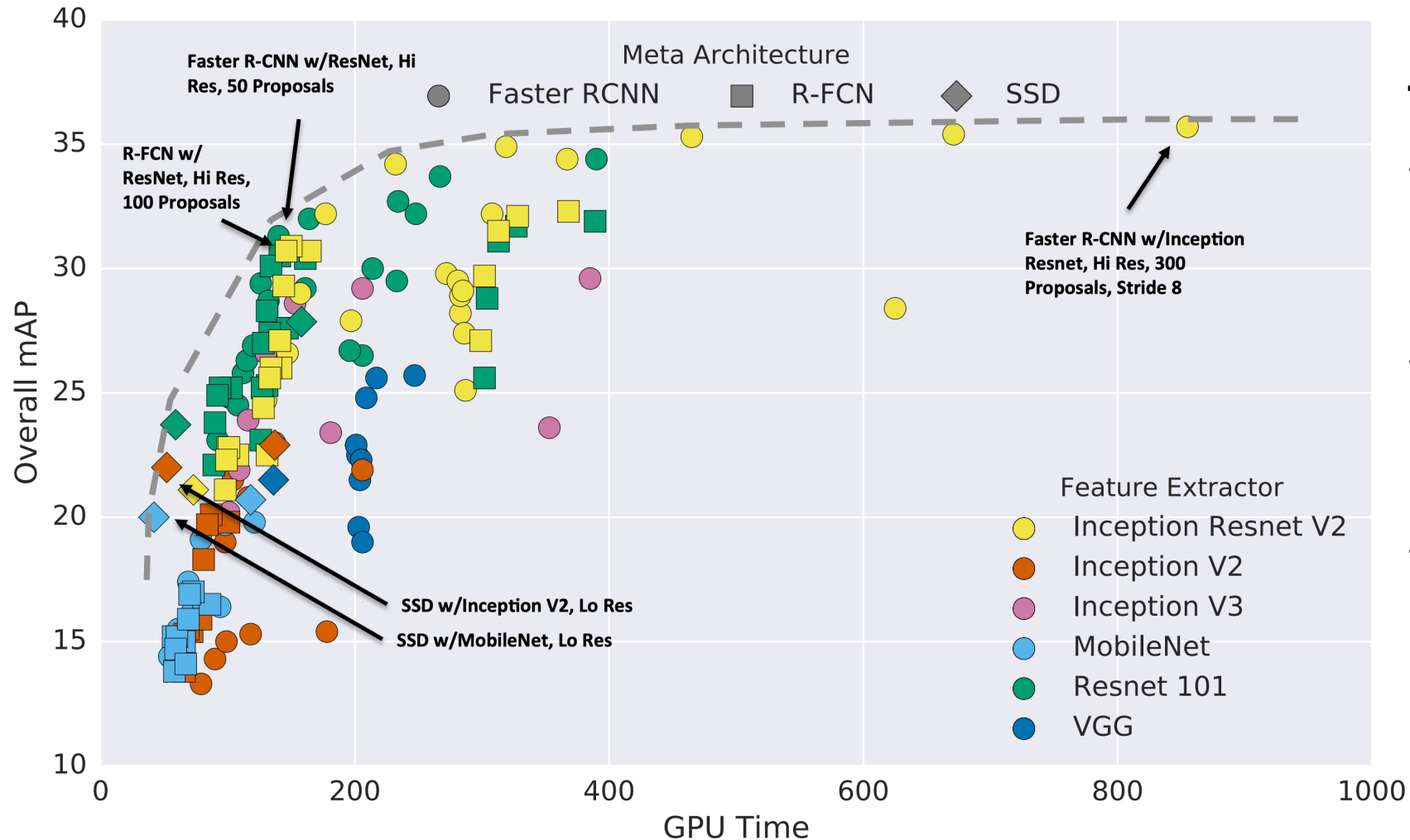


## Takeaways:

- Two stage method (Faster R-CNN) get the best accuracy, but are slower
- Single-stage methods (SSD) are much faster, but don't perform as well

Huang et al, "Speed/accuracy trade-offs for modern convolutional object detectors", CVPR 2017

# Object Detection: Lots of variables!



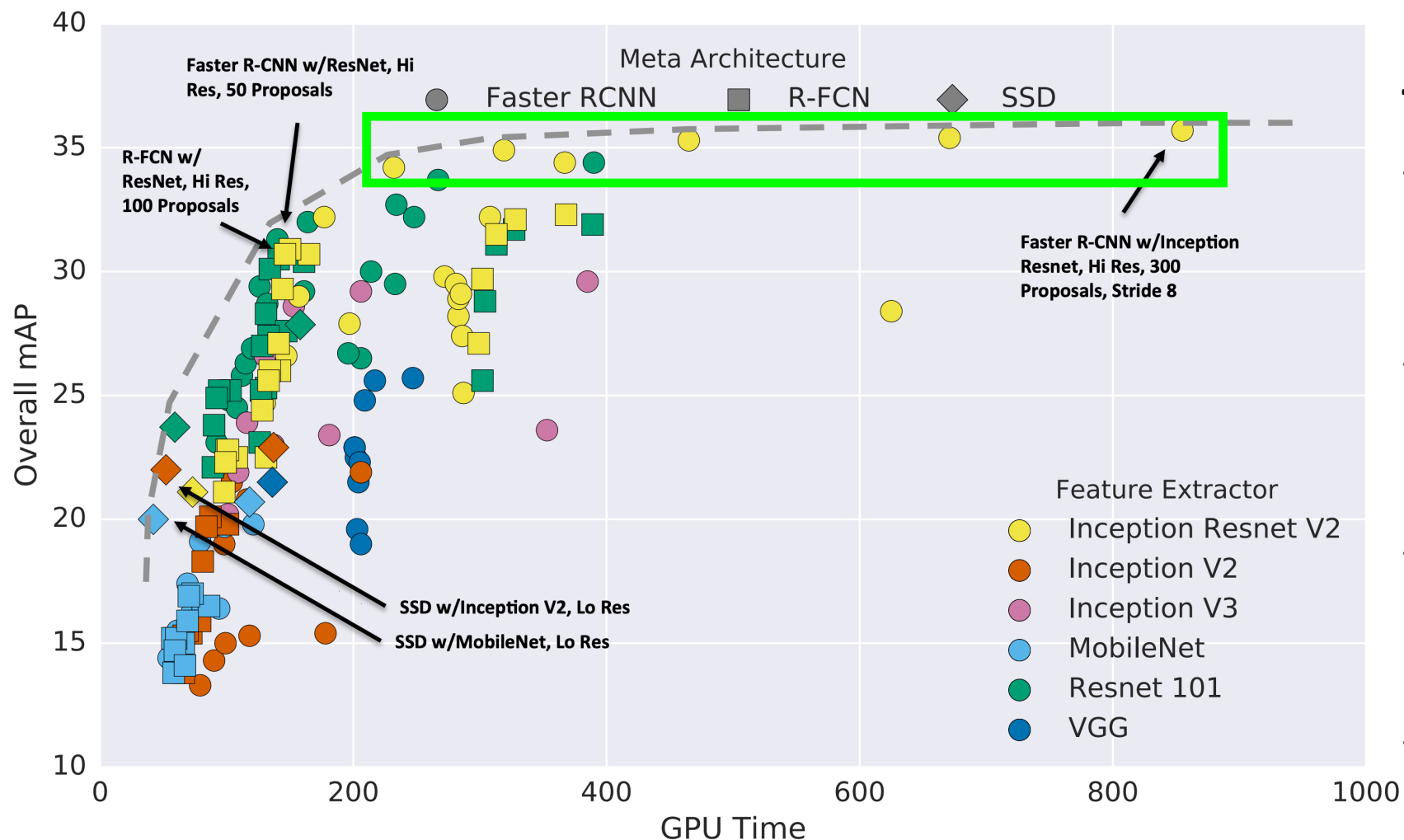
## Takeaways:

- Two stage method (Faster R-CNN) get the best accuracy, but are slower
- Single-stage methods (SSD) are much faster, but don't perform as well
- Bigger backbones improve performance, but are slower

Huang et al, "Speed/accuracy trade-offs for modern convolutional object detectors", CVPR 2017



# Object Detection: Lots of variables!



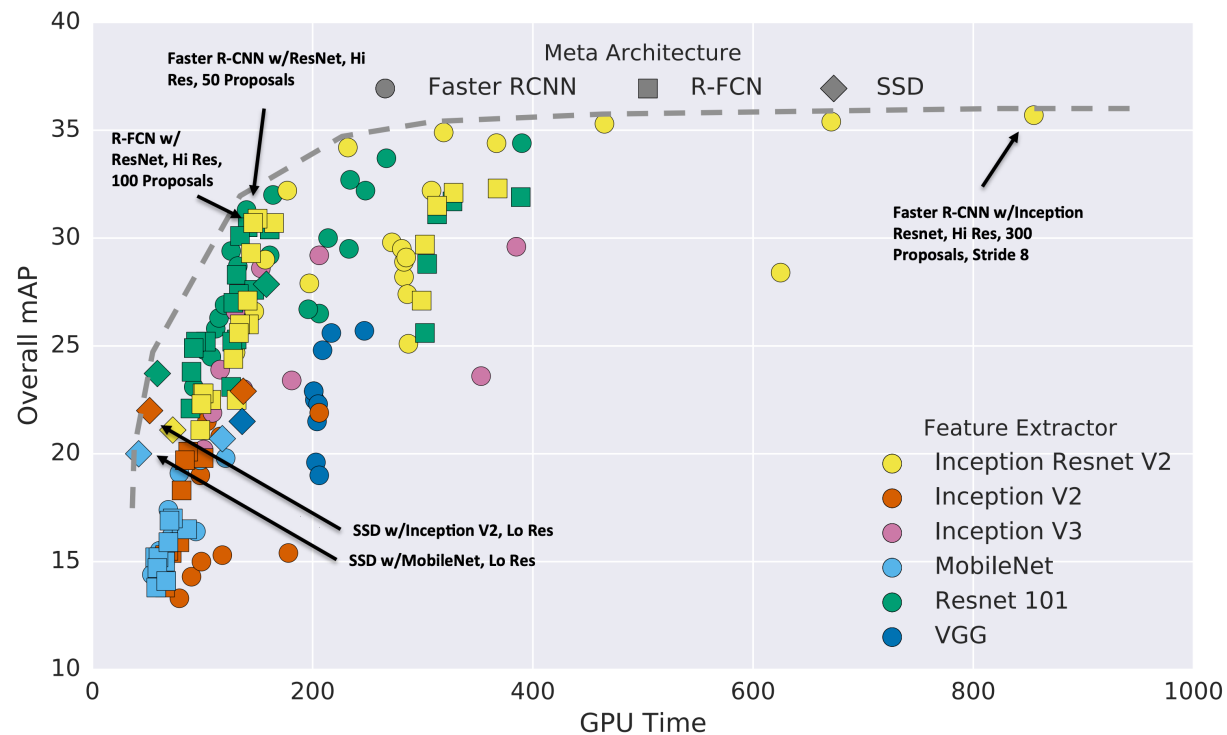
## Takeaways:

- Two stage method (Faster R-CNN) get the best accuracy, but are slower
- Single-stage methods (SSD) are much faster, but don't perform as well
- Bigger backbones improve performance, but are slower
- Diminishing returns for slower methods

Huang et al, "Speed/accuracy trade-offs for modern convolutional object detectors", CVPR 2017

# Object Detection: Lots of variables!

These results are a few years old ... since then GPUs have gotten faster, and we've improved performance with many tricks:

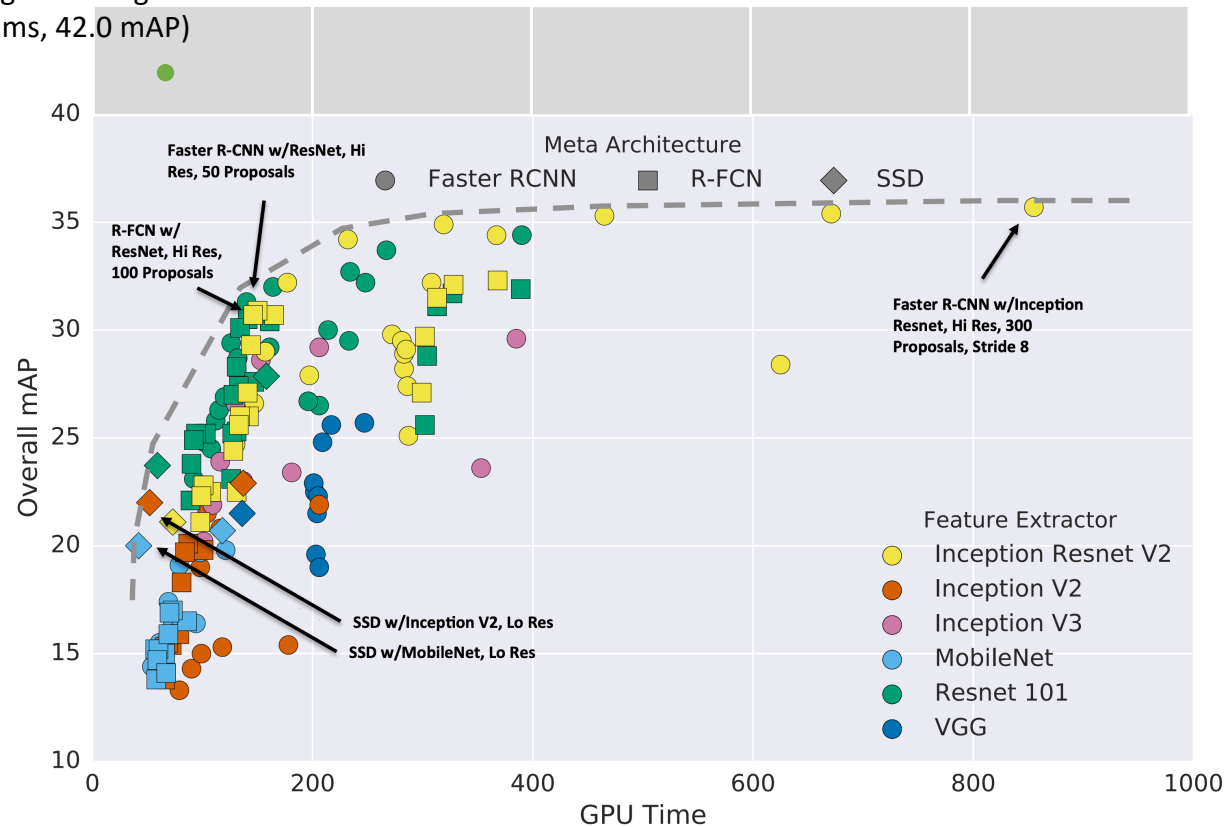


Huang et al, "Speed/accuracy trade-offs for modern convolutional object detectors", CVPR 2017

Wu et al, Detectron2, GitHub 2019

# Object Detection: Lots of variables!

Faster R-CNN  
w/ResNet-101-FPN,  
longer training  
(63ms, 42.0 mAP)



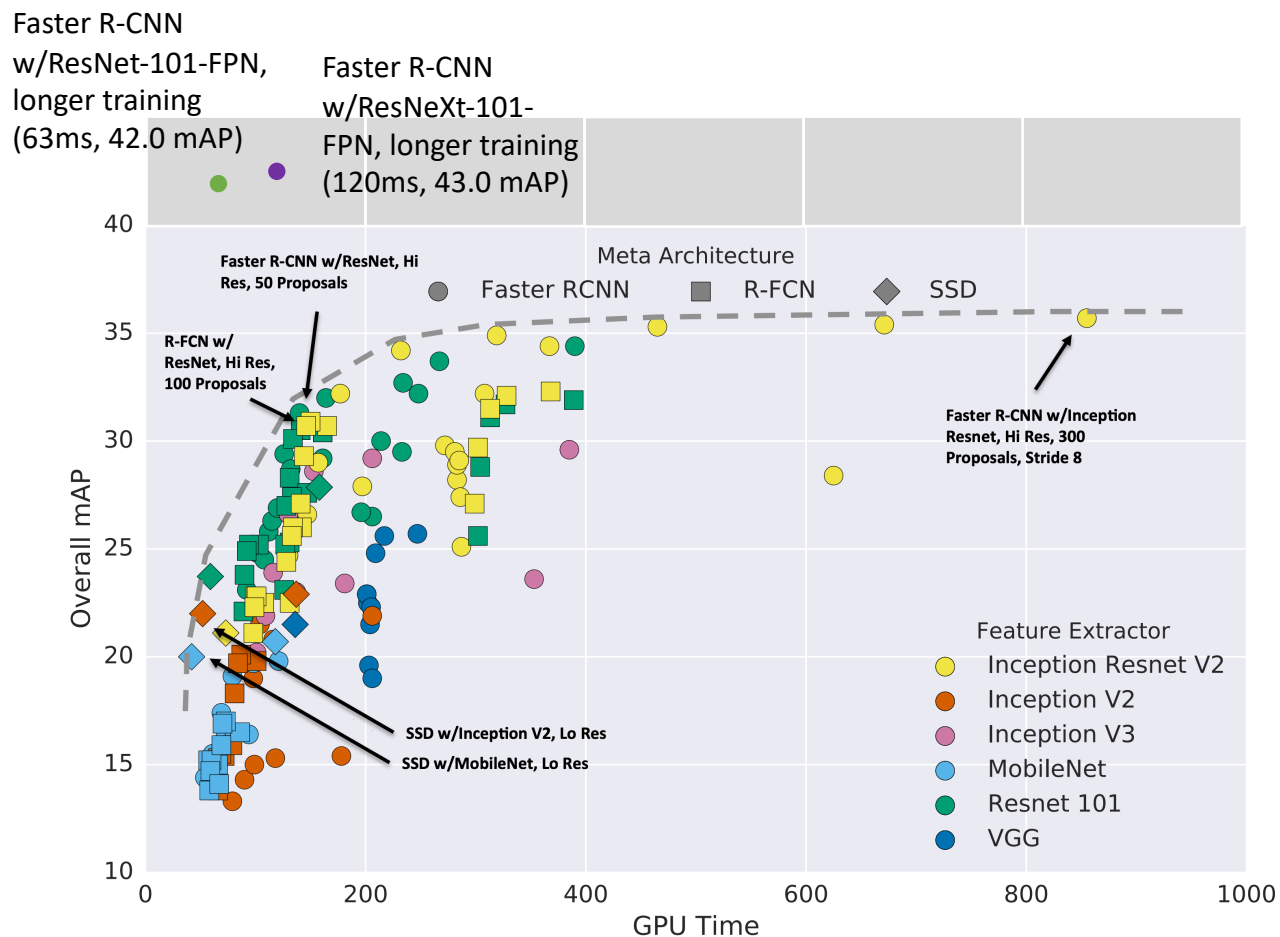
These results are a few years old ... since then GPUs have gotten faster, and we've improved performance with many tricks:

- Train longer!
- Multiscale backbone: Feature Pyramid Networks

Huang et al, "Speed/accuracy trade-offs for modern convolutional object detectors", CVPR 2017

Wu et al, Detectron2, GitHub 2019

# Object Detection: Lots of variables!

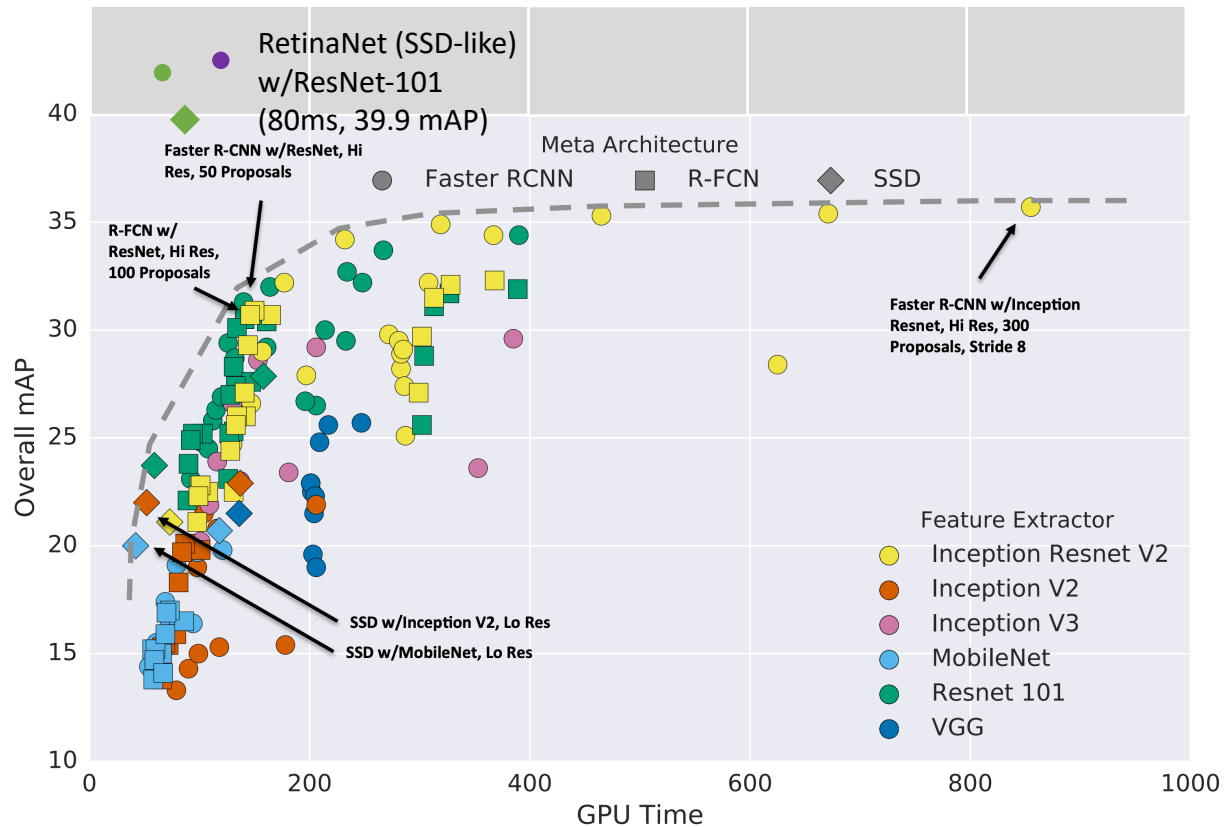


These results are a few years old ... since then GPUs have gotten faster, and we've improved performance with many tricks:

- Train longer!
- Multiscale backbone: Feature Pyramid Networks
- Better backbone: ResNeXt



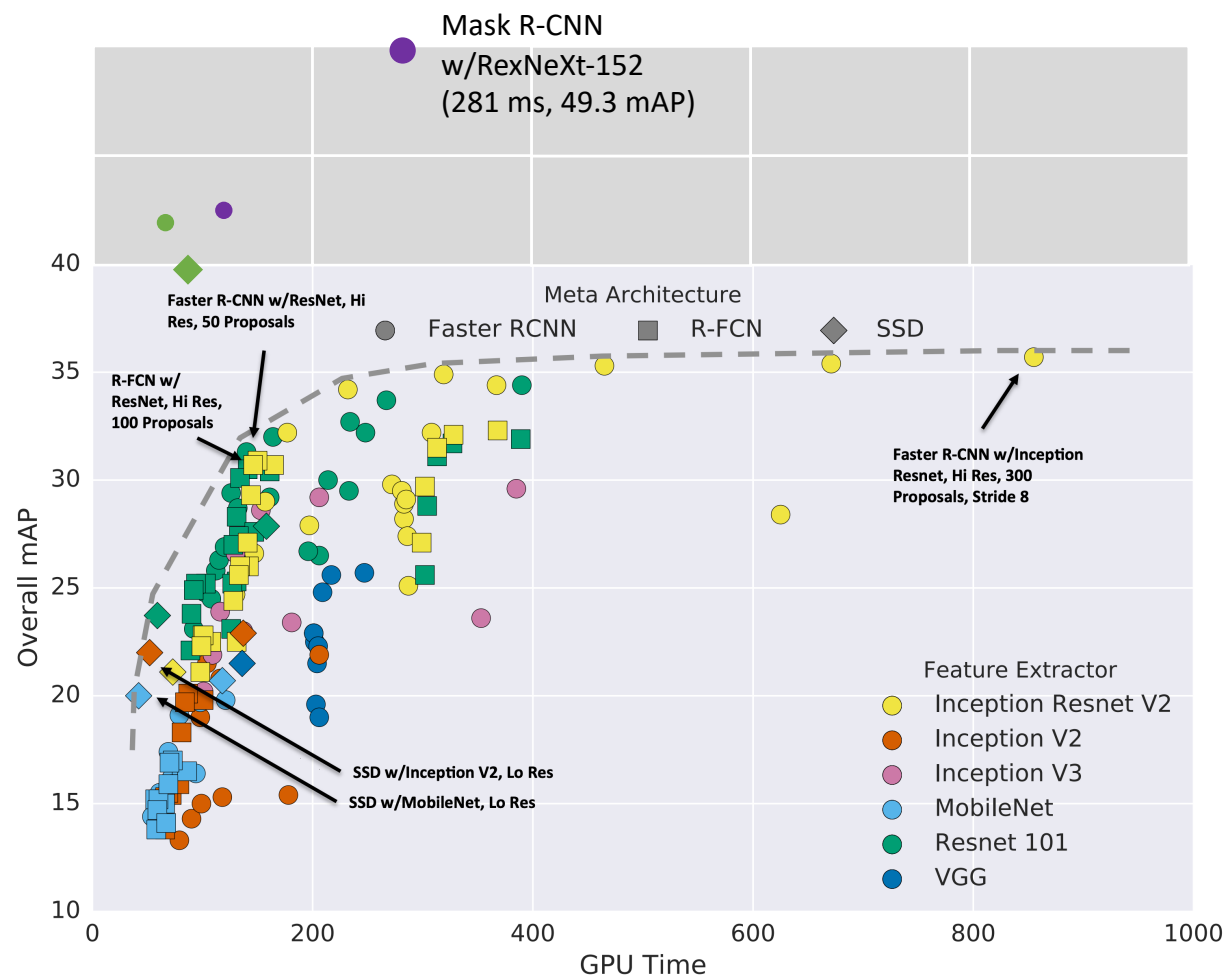
# Object Detection: Lots of variables!



These results are a few years old ... since then GPUs have gotten faster, and we've improved performance with many tricks:

- Train longer!
- Multiscale backbone: Feature Pyramid Networks
- Better backbone: ResNeXt
- Single-Stage methods have improved

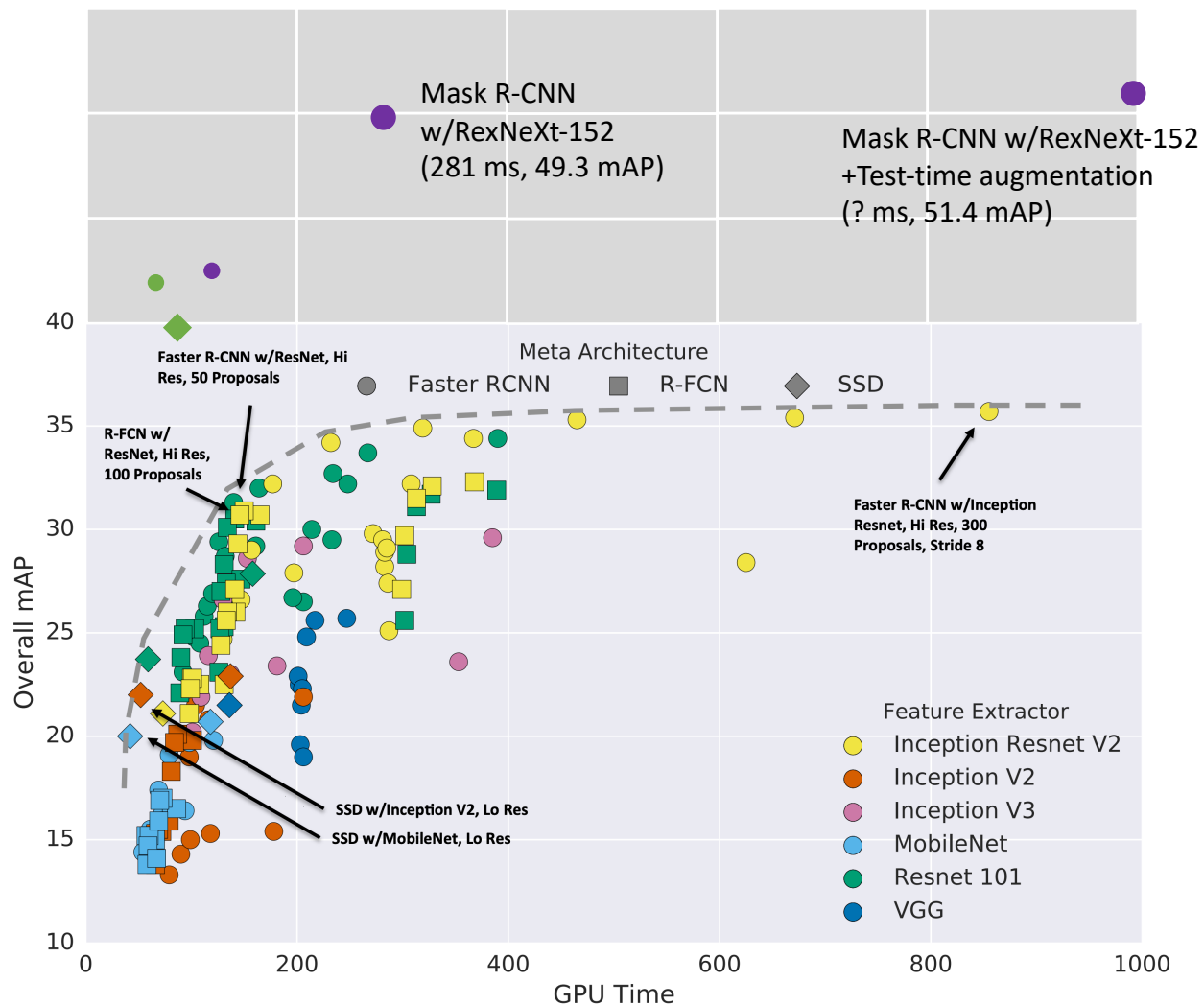
# Object Detection: Lots of variables!



These results are a few years old ... since then GPUs have gotten faster, and we've improved performance with many tricks:

- Train longer!
- Multiscale backbone: Feature Pyramid Networks
- Better backbone: ResNeXt
- Single-Stage methods have improved
- Very big models work better

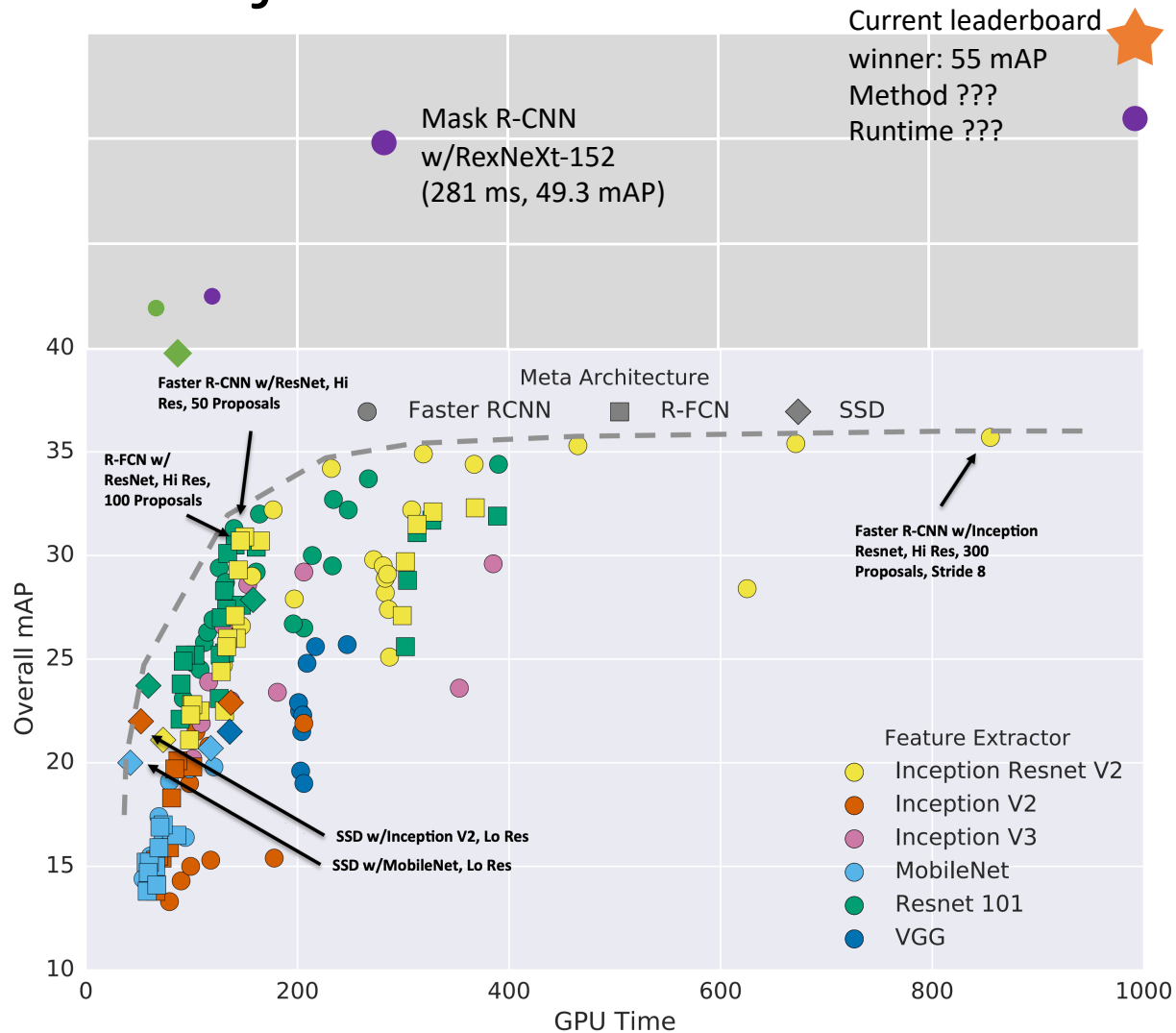
# Object Detection: Lots of variables!



These results are a few years old ... since then GPUs have gotten faster, and we've improved performance with many tricks:

- Train longer!
- Multiscale backbone: Feature Pyramid Networks
- Better backbone: ResNeXt
- Single-Stage methods have improved
- Very big models work better
- Test-time augmentation pushes numbers up

# Object Detection: Lots of variables!



These results are a few years old ... since then GPUs have gotten faster, and we've improved performance with many tricks:

- Train longer!
- Multiscale backbone: Feature Pyramid Networks
- Better backbone: ResNeXt
- Single-Stage methods have improved
- Very big models work better
- Test-time augmentation pushes numbers up
- Big ensembles, more data, etc



# Object Detection: Open-Source Code

Object detection is hard! Don't implement it yourself  
(Unless you are working on A5...)

## **TensorFlow Detection API:**

[https://github.com/tensorflow/models/tree/master/research/object\\_detection](https://github.com/tensorflow/models/tree/master/research/object_detection)

Faster R-CNN, SSD, RFCN, Mask R-CNN

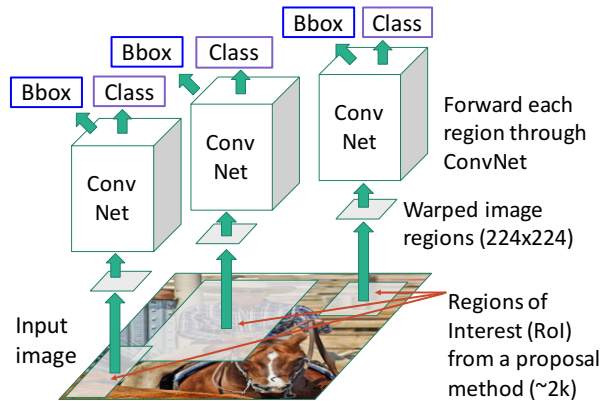
## **Detectron2 (PyTorch):**

<https://github.com/facebookresearch/detectron2>

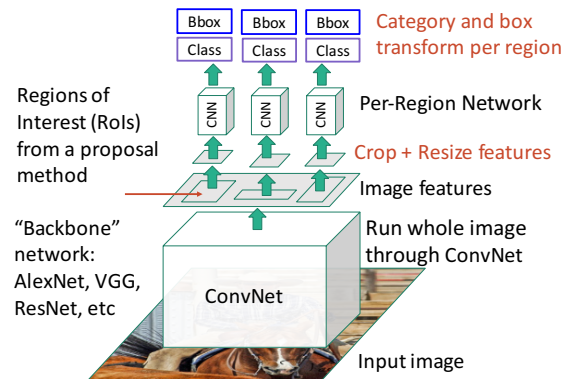
Fast / Faster / Mask R-CNN, RetinaNet

# Summary

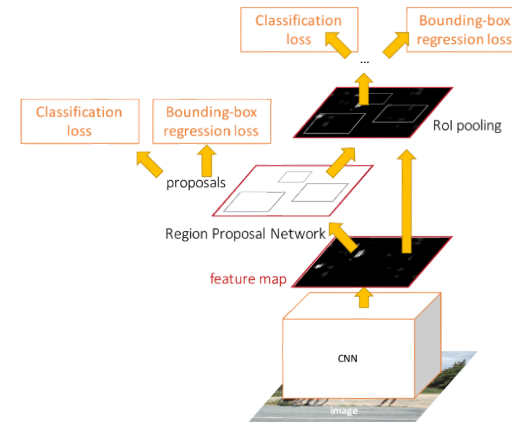
**“Slow” R-CNN:** Run CNN independently for each region



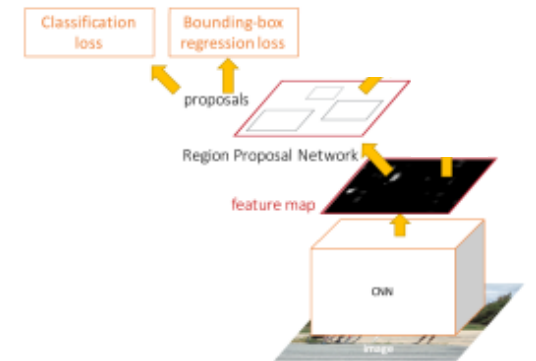
**Fast R-CNN:** Apply differentiable cropping to shared image features



**Faster R-CNN:** Compute proposals with CNN



**Single-Stage:** Fully convolutional detector



Next Time:  
More localization methods:  
Segmentation, Keypoint Estimation