# Lecture 15: Object Detection

# Reminder: A4

A4 due **Wednesday, November 13, 11:59pm**

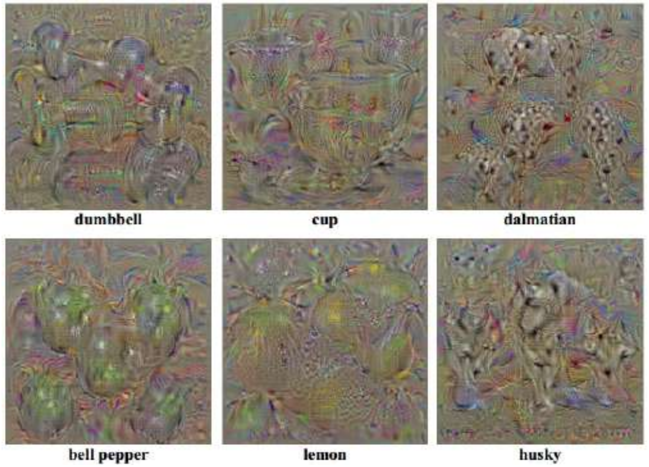
A4 covers:
- PyTorch autograd
- Residual networks
- Recurrent neural networks
- Attention
- Feature visualization
- Style transfer
- Adversarial examples

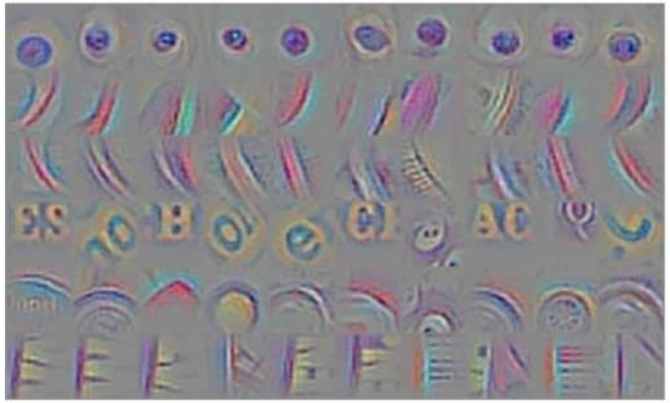
# Last Time: Visualizing and Understanding CNNs

Nearest Neighbor

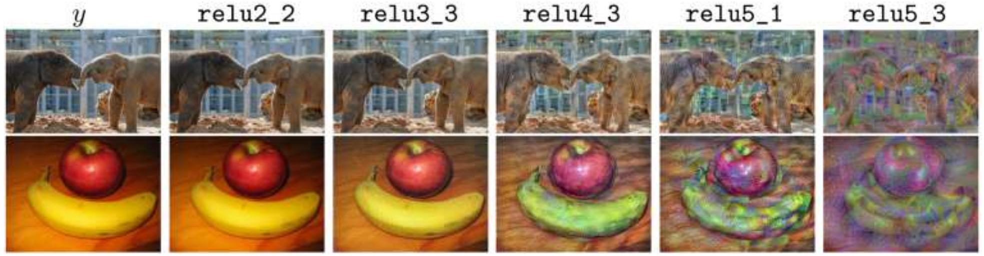
Maximally Activating Patches

Synthetic Images via Gradient Ascent



(Guided) Backprop




Feature Inversion

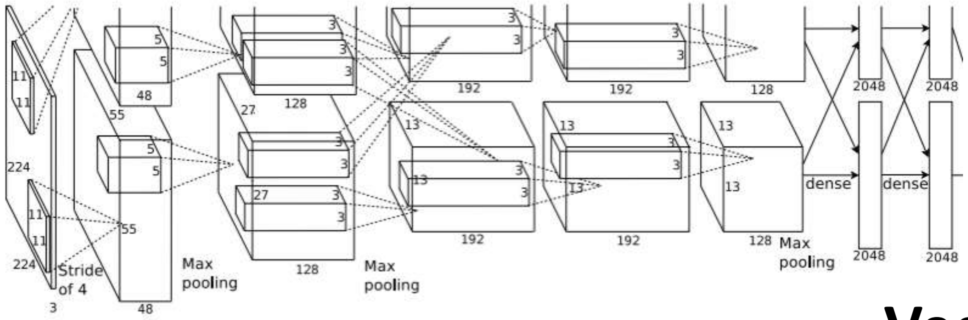
# Last Time: Making art with CNNs

DeepDream

Style Transfer

# So far: Image Classification

This image is CC0 public domain



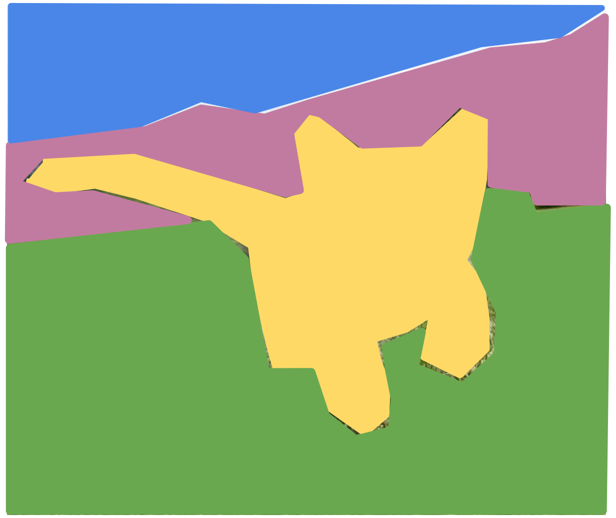
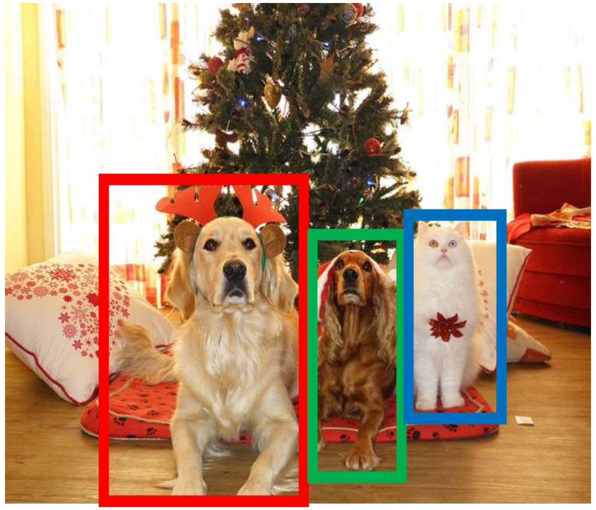



**Vector:** 4096

**Fully-Connected**: 4096 to 1000

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

# Computer Vision Tasks

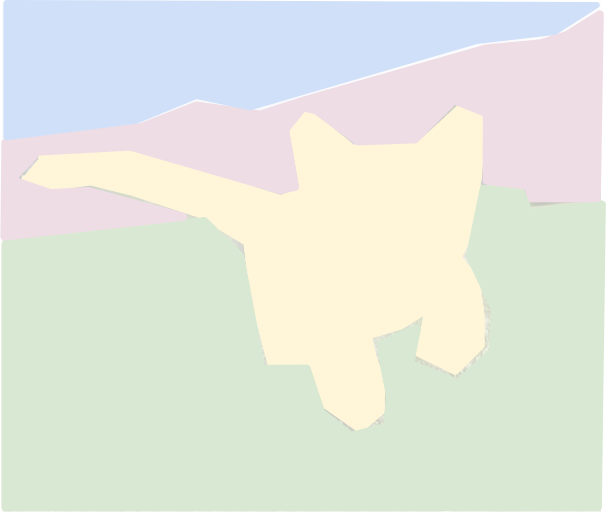
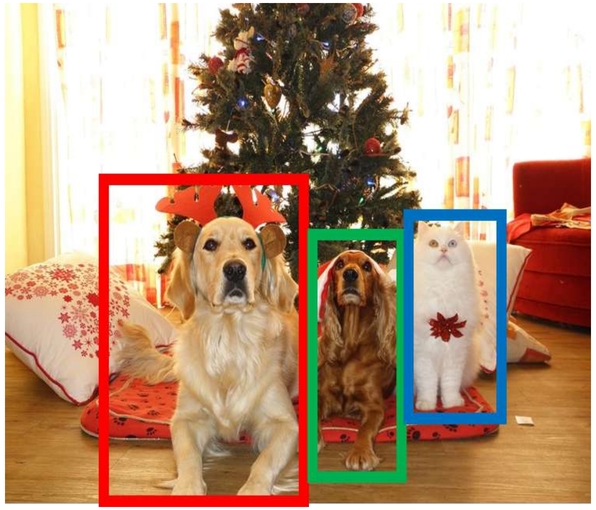
| Classification | Semantic Segmentation | Object Detection | Instance Segmentation |
| --- | --- | --- | --- |
|  |  |  |  |
| CAT | GRASS, CAT, TREE, SKY | DOG, DOG, CAT | DOG, DOG, CAT |

No spatial extent

No objects, just pixels

Multiple Objects

# Today: Object Detection

| 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

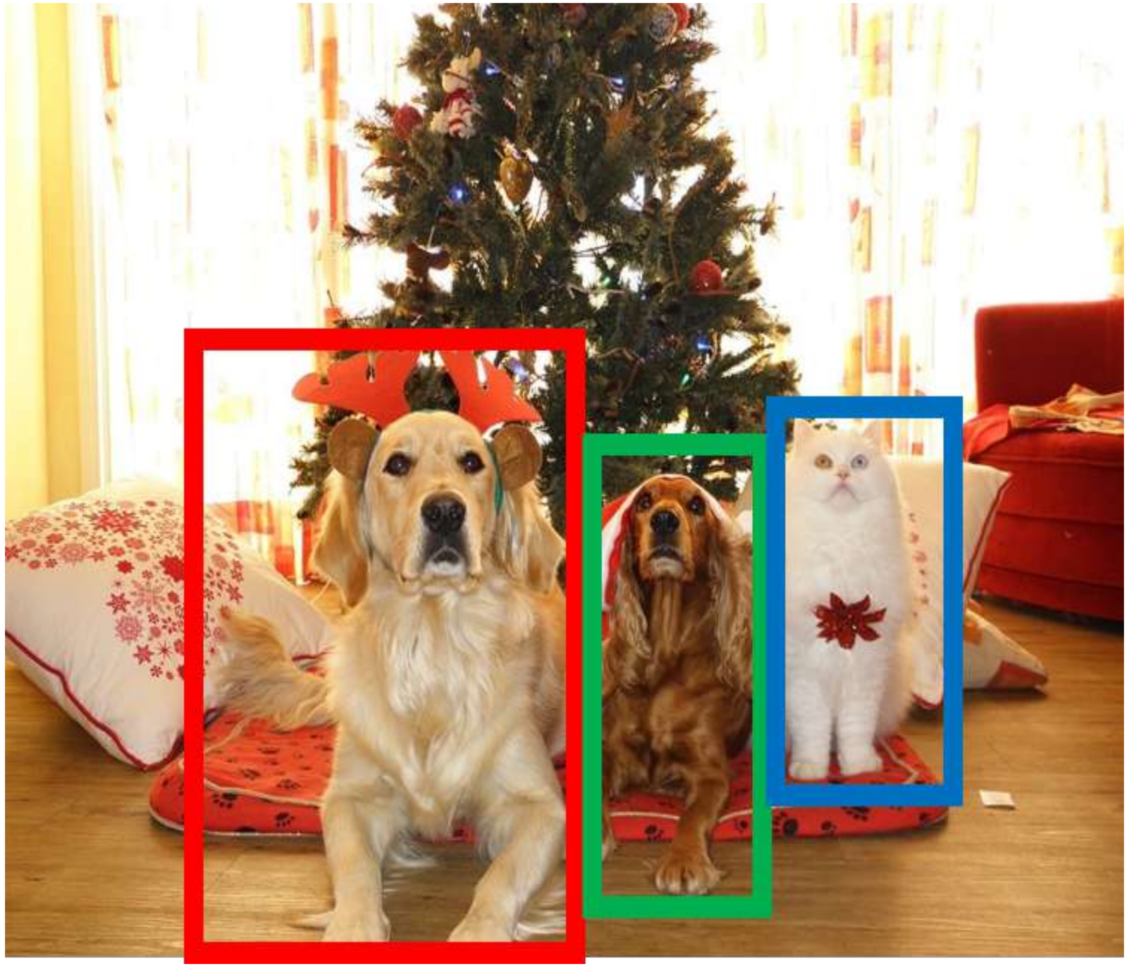
Multiple Objects

# Object Detection: Task Definition

**Input**: Single RGB Image

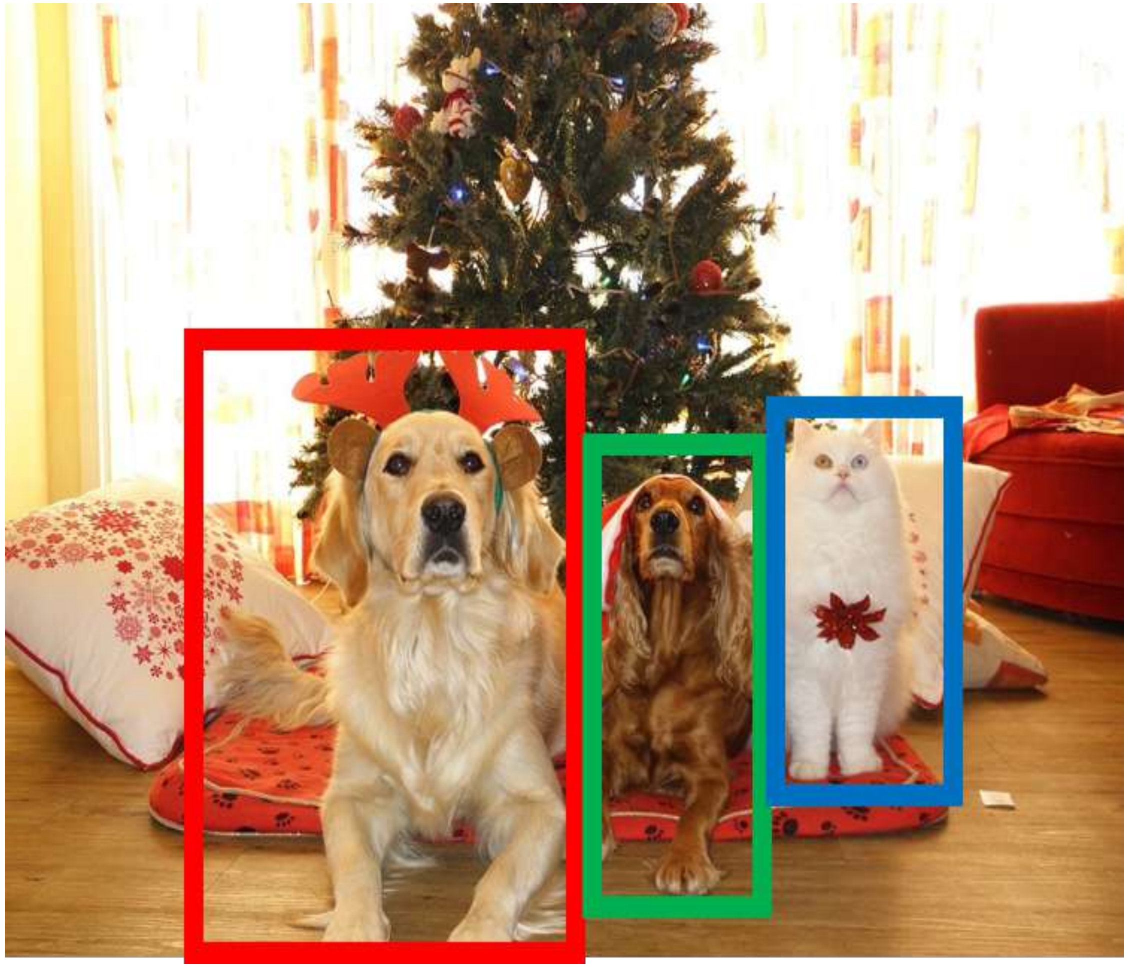
**Output**: A <u>set</u> of detected objects;
For each object predict:

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



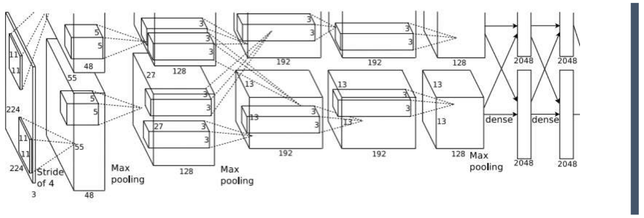
# Object Detection: Challenges

- **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"

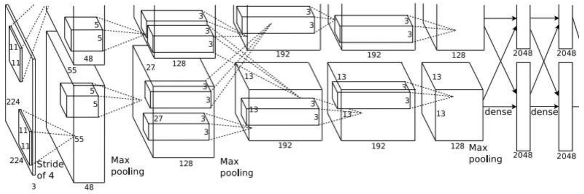
**Correct label:**
Cat

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

**Softmax Loss**

**Fully Connected**: 4096 to 1000

This image is CC0 public domain




**Vector:** 4096

# Detecting a single object

“What”

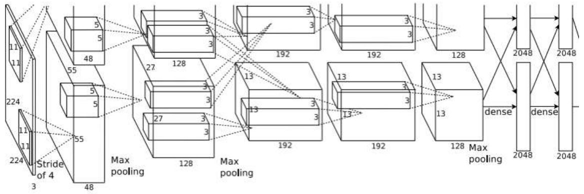
**Correct label:**
Cat

**Fully Connected:**
4096 to 1000

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

**Softmax Loss**

This image is CC0 public domain



**Vector:**
4096

**Fully Connected:**
4096 to 4

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

**L2 Loss**

**Correct box:**
(x’, y’, w’, h’)

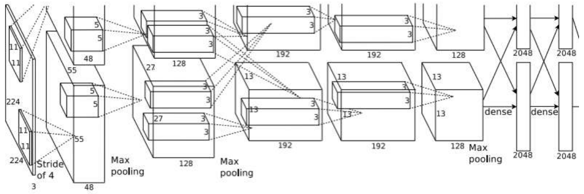
Treat localization as a regression problem!

“Where”

# Detecting a single object

"What"

This image is CC0 public domain



**Vector:**
4096

**Fully Connected**:
4096 to 1000

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

**Correct label:**
Cat

**Softmax Loss**

**Weighted Sum**

**Loss**

**Fully Connected**:
4096 to 4

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

**L2 Loss**

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

"Where"

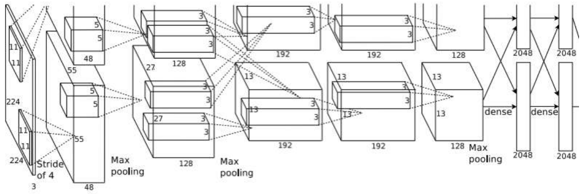
Treat localization as a regression problem!

# 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**

Multitask Loss

**Weighted Sum**

**Loss**

**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"

# Detecting a single object

"What"

**Correct label:**
Cat

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

**Softmax Loss**

Multitask Loss

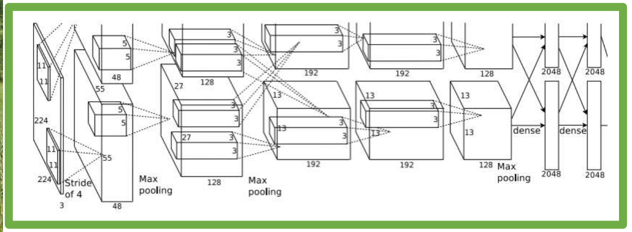
**Weighted Sum**

**Loss**

Often pretrained on ImageNet (Transfer learning)

**Fully Connected:** 4096 to 1000

This image is CC0 public domain



**Vector:** 4096

**Fully Connected:** 4096 to 4

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

**L2 Loss**

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

"Where"

Treat localization as a regression problem!

# Detecting a single object

"What"

**Correct label:**
Cat

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

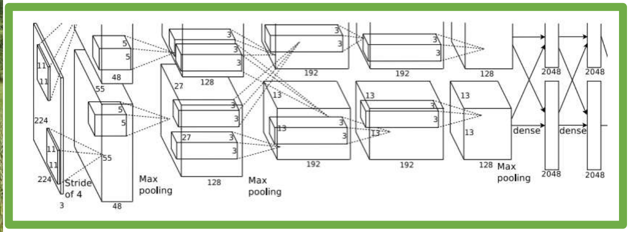
**Softmax Loss**

Multitask Loss

Often pretrained on ImageNet (Transfer learning)

**Fully Connected**: 4096 to 1000

This image is CC0 public domain



**Vector:** 4096

**Weighted Sum**

**Loss**

**Fully Connected**: 4096 to 4

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

**L2 Loss**

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

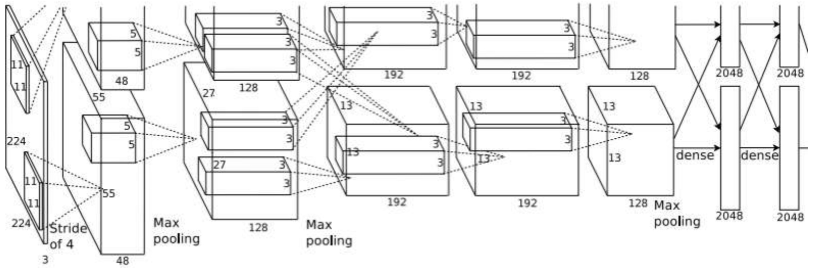
Treat localization as a regression problem!

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

"Where"

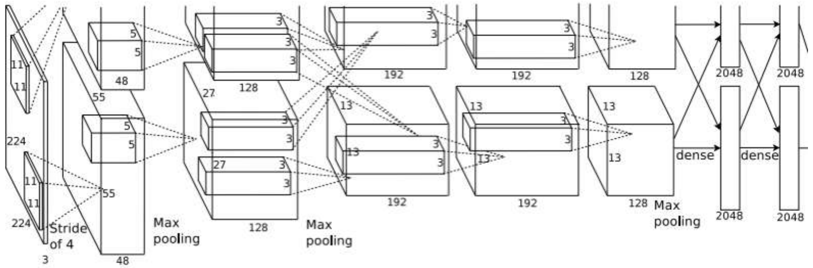
# Detecting Multiple Objects

Need different numbers of outputs per image



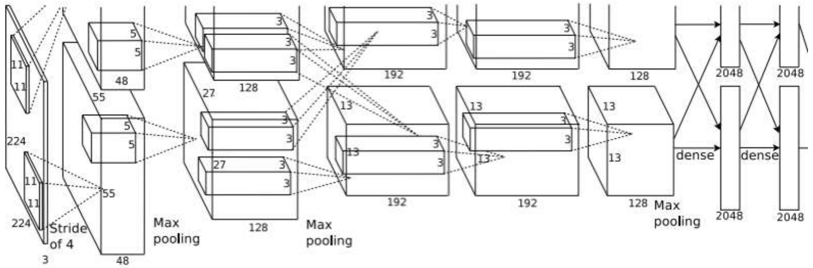

CAT: (x, y, w, h)

4 numbers




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

16 numbers

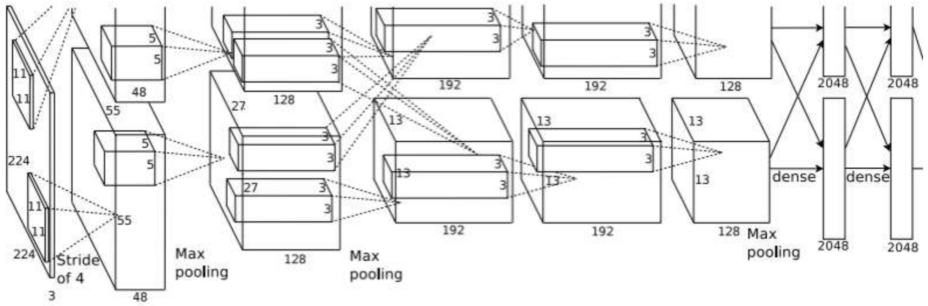



DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
....

Many numbers!

# Detecting Multiple Objects: **Sliding Window**

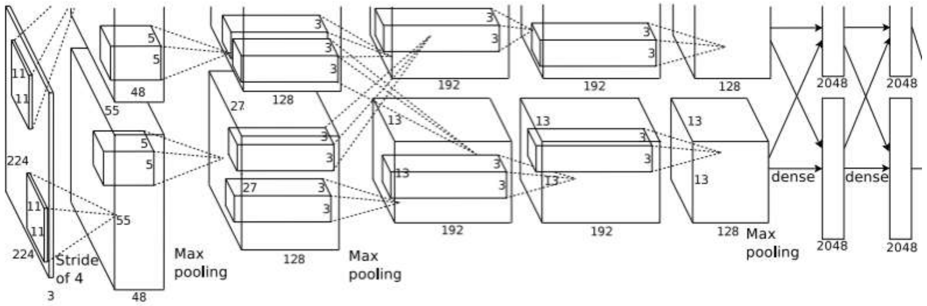
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background




Dog? NO
Cat? NO
Background? YES

# Detecting Multiple Objects: **Sliding Window**

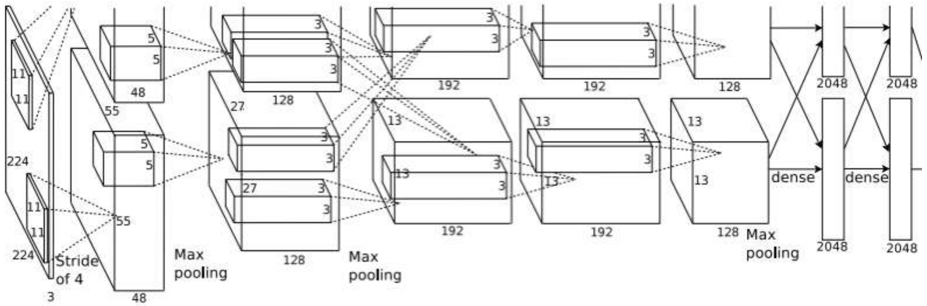
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background




Dog? YES
Cat? NO
Background? NO

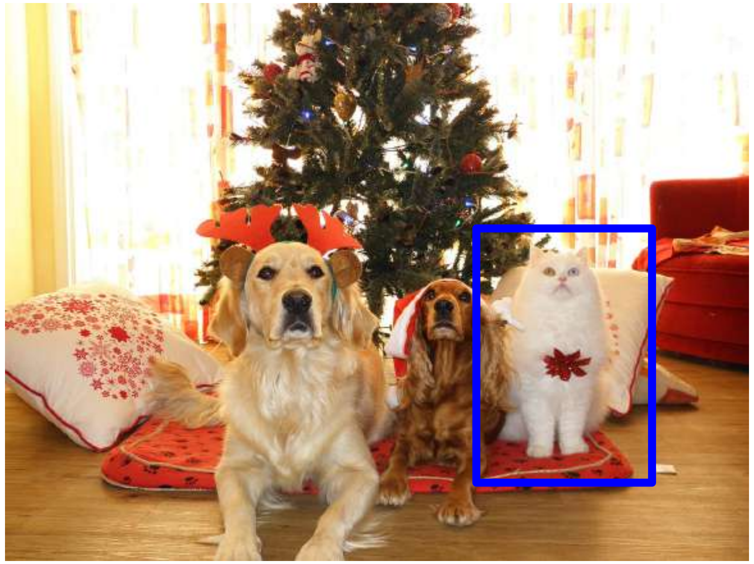
# Detecting Multiple Objects: **Sliding Window**

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

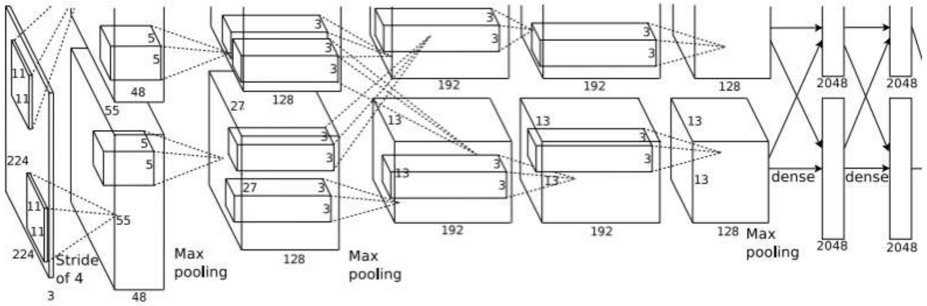


Dog? YES
Cat? NO
Background? NO

# Detecting Multiple Objects: **Sliding Window**

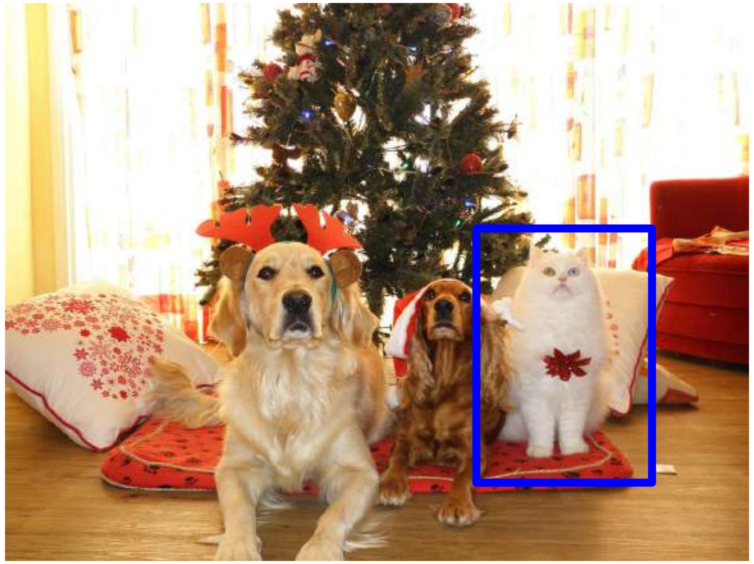


Apply a CNN to many different crops of the image, CNN classifies each crop as object or background




Dog? NO
Cat? YES
Background? NO

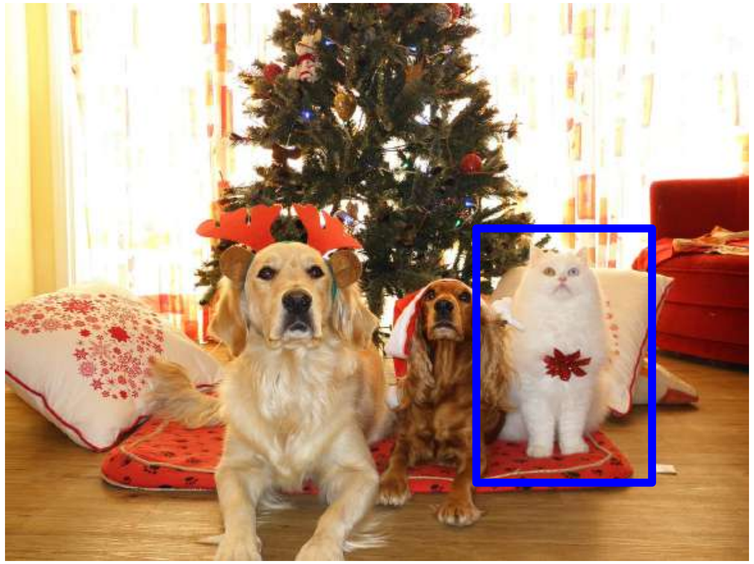
# Detecting Multiple Objects: **Sliding Window**



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 x W?

# Detecting Multiple Objects: **Sliding Window**

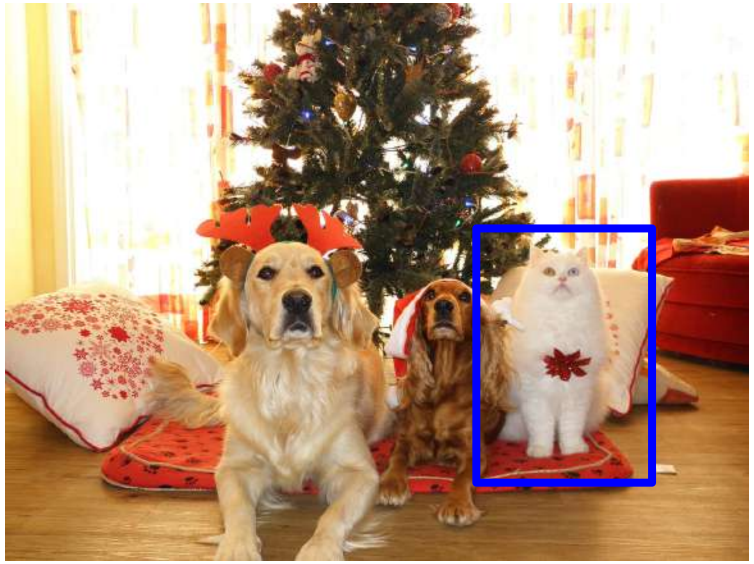


Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

**Question**: How many possible boxes are there in an image of size H 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)

# 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 x W?

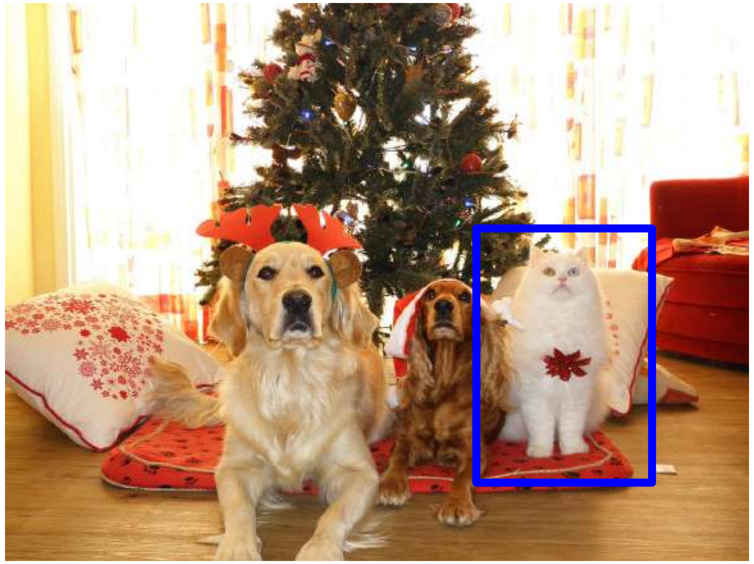
Consider a box of size h x w:
Possible x positions: W – w + 1
Possible y positions: H – h + 1
Possible positions:
(W – w + 1) * (H – h + 1)

Total possible boxes:

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

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

# 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)

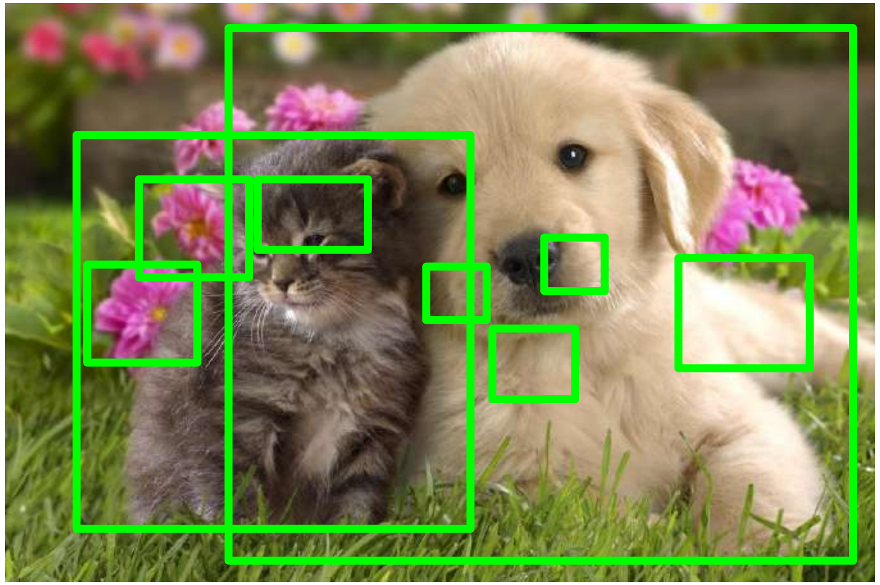
Total possible boxes:

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

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

# 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.

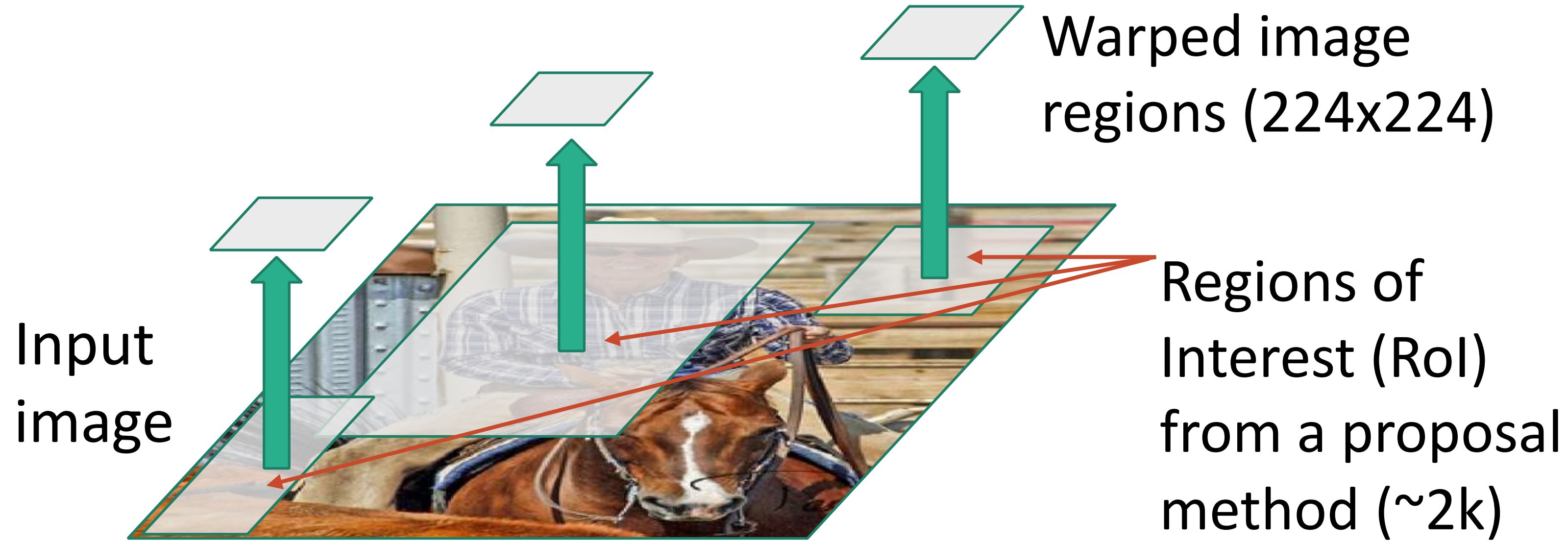
# R-CNN: Region-Based CNN

Input image

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

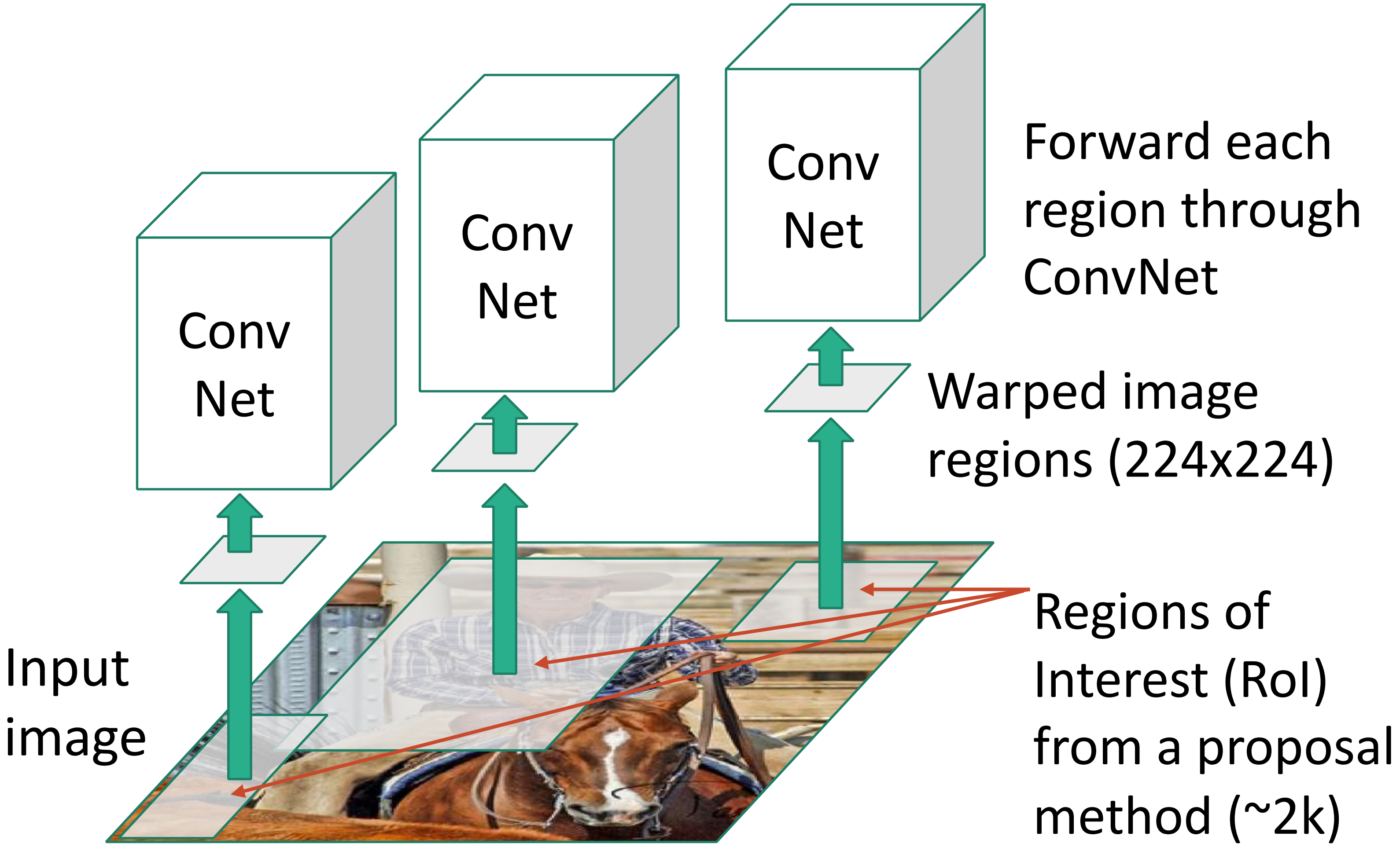
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

# R-CNN: Region-Based CNN




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

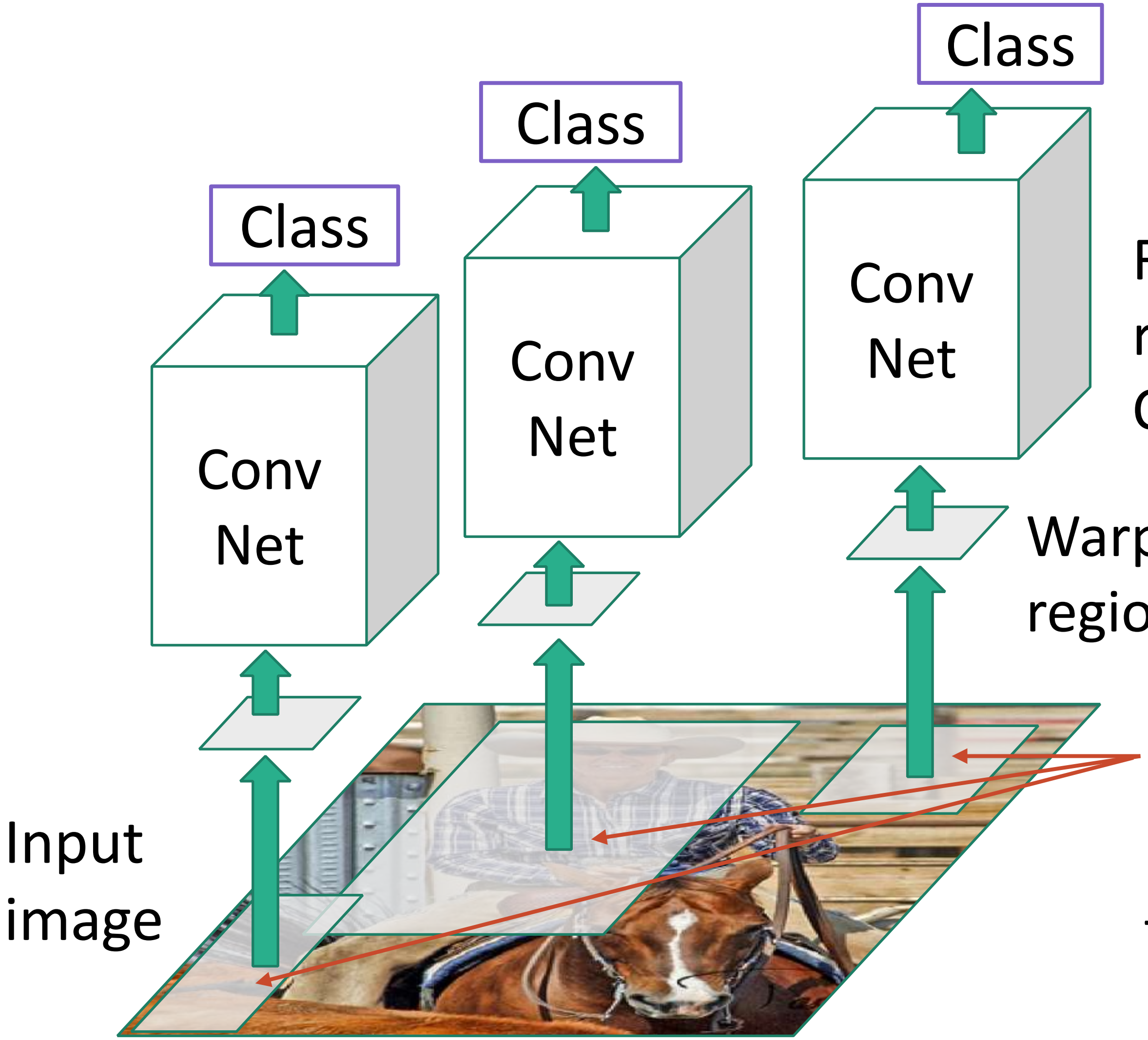
# R-CNN: Region-Based CNN




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




Forward each region through ConvNet

Warped image regions (224x224)

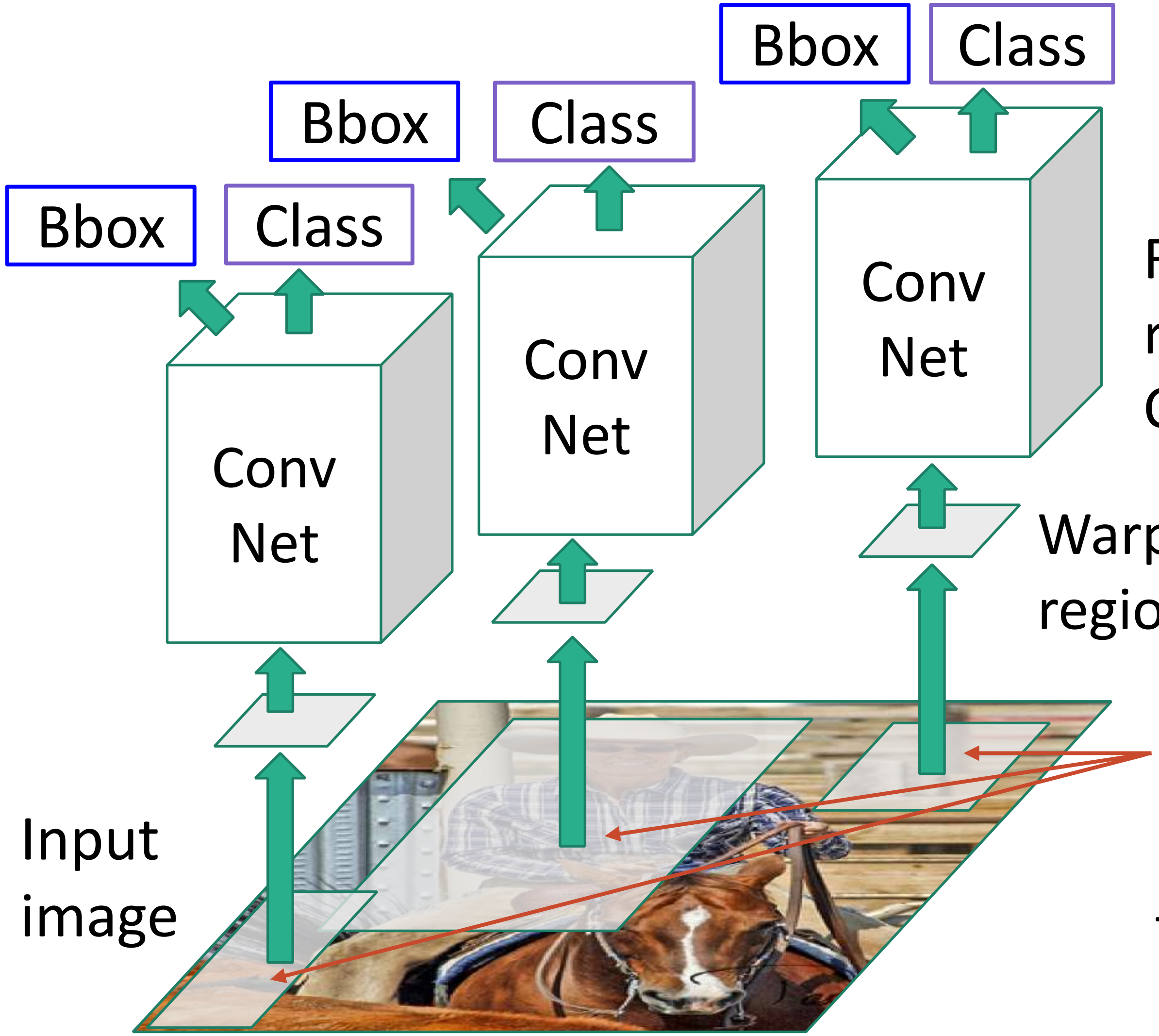
Regions of Interest (RoI) from a proposal method (~2k)

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

# R-CNN: Region-Based CNN

Classify each region

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



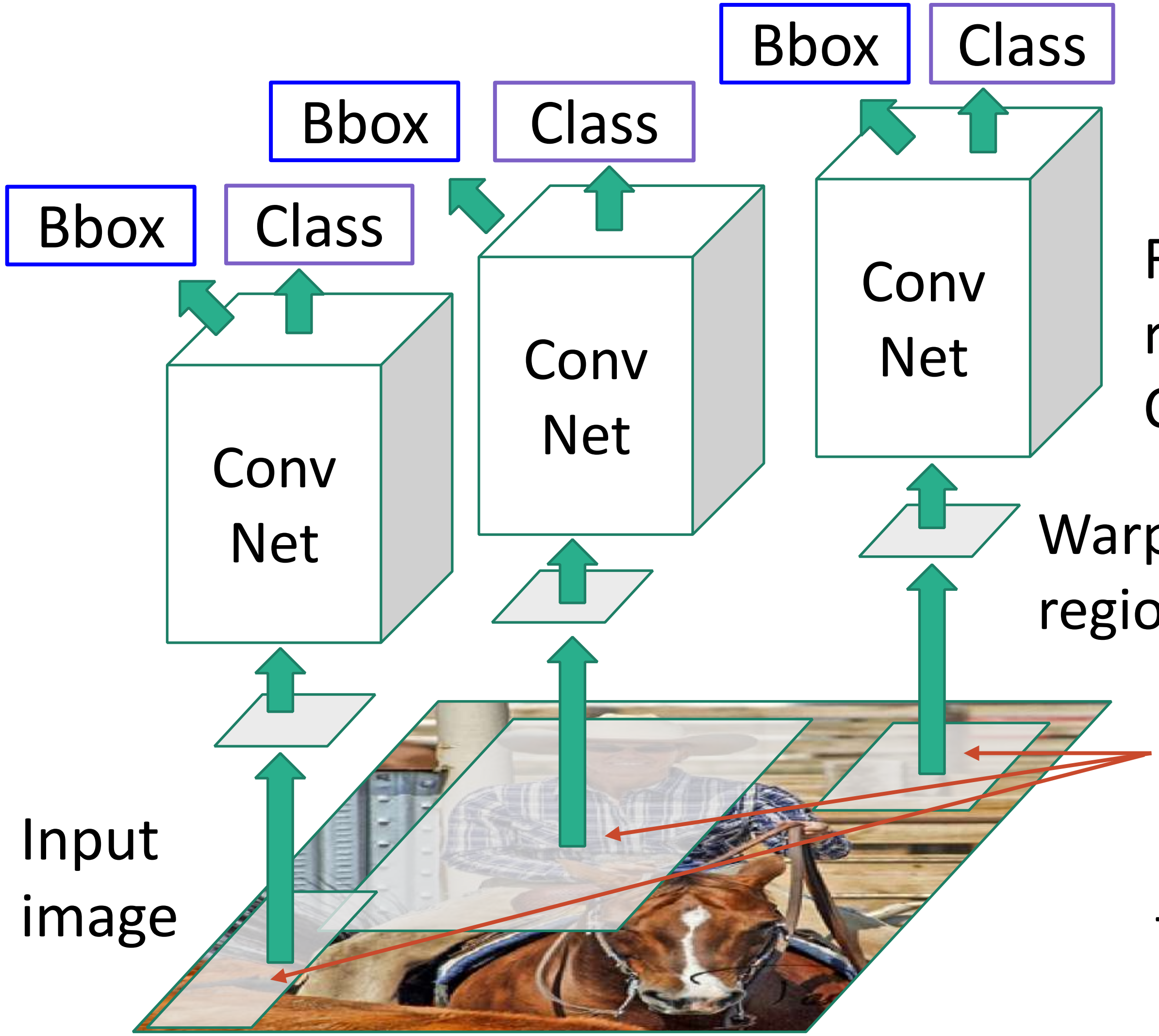

Forward each region through ConvNet

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.

# 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)

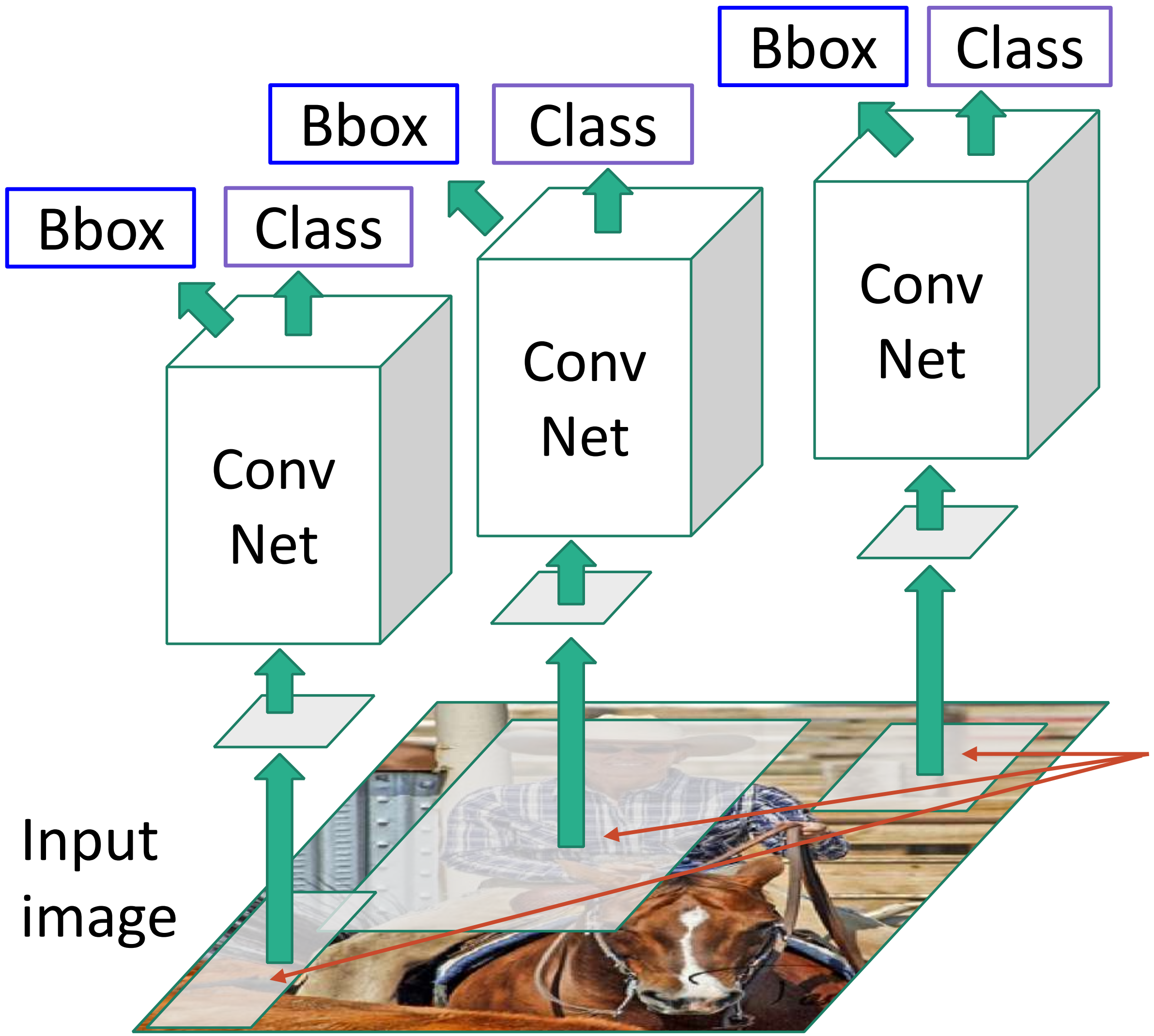
Region proposal: $(p_x, p_y, p_h, p_w)$
Transform: $(t_x, t_y, t_h, t_w)$
Output box: $(b_x, b_y, b_h, b_w)$

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

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

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

# R-CNN: Test-time



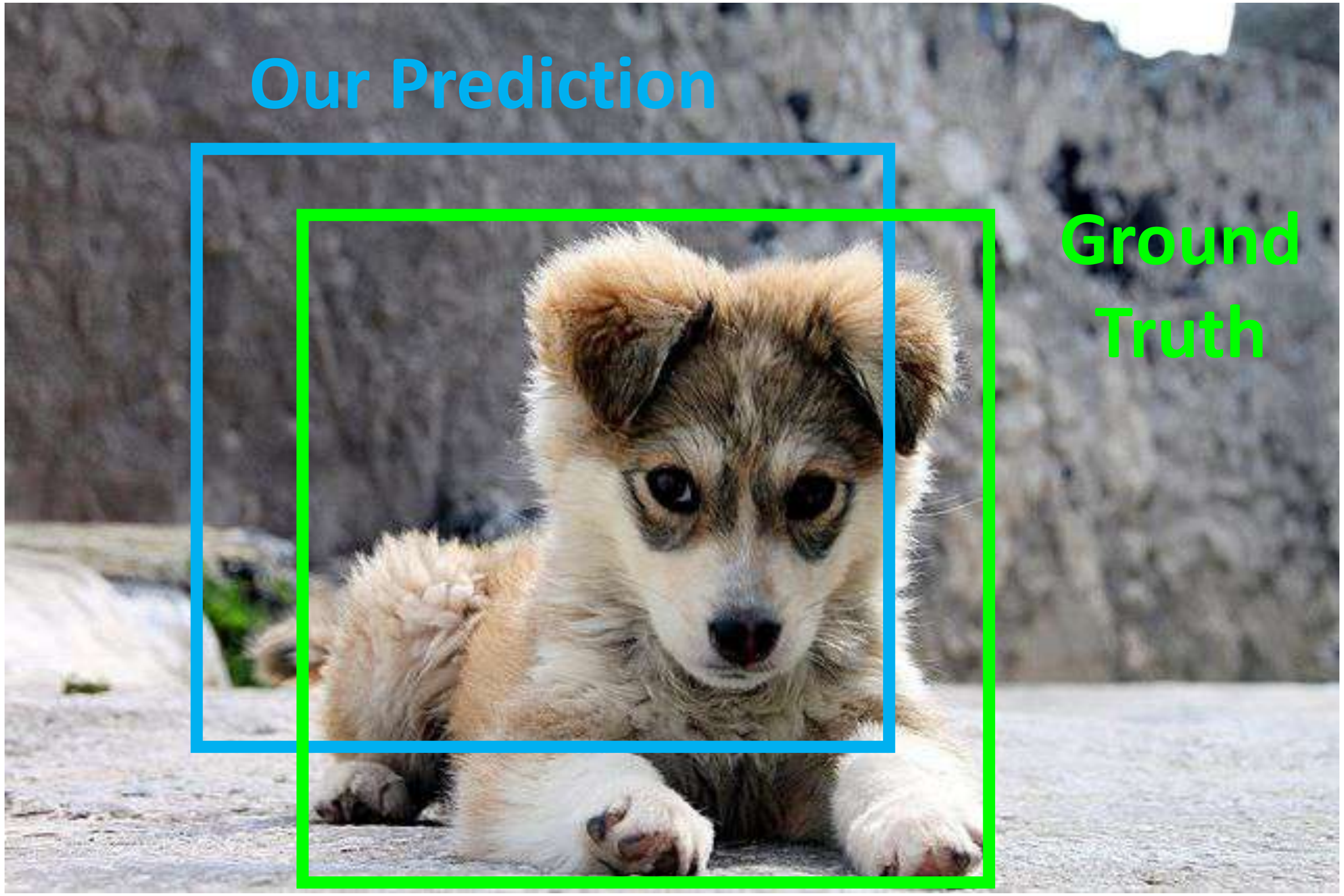

Input: Single RGB Image

1. Run region proposal method to compute ~2000 region proposals
2. Resize each region to 224x224 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?

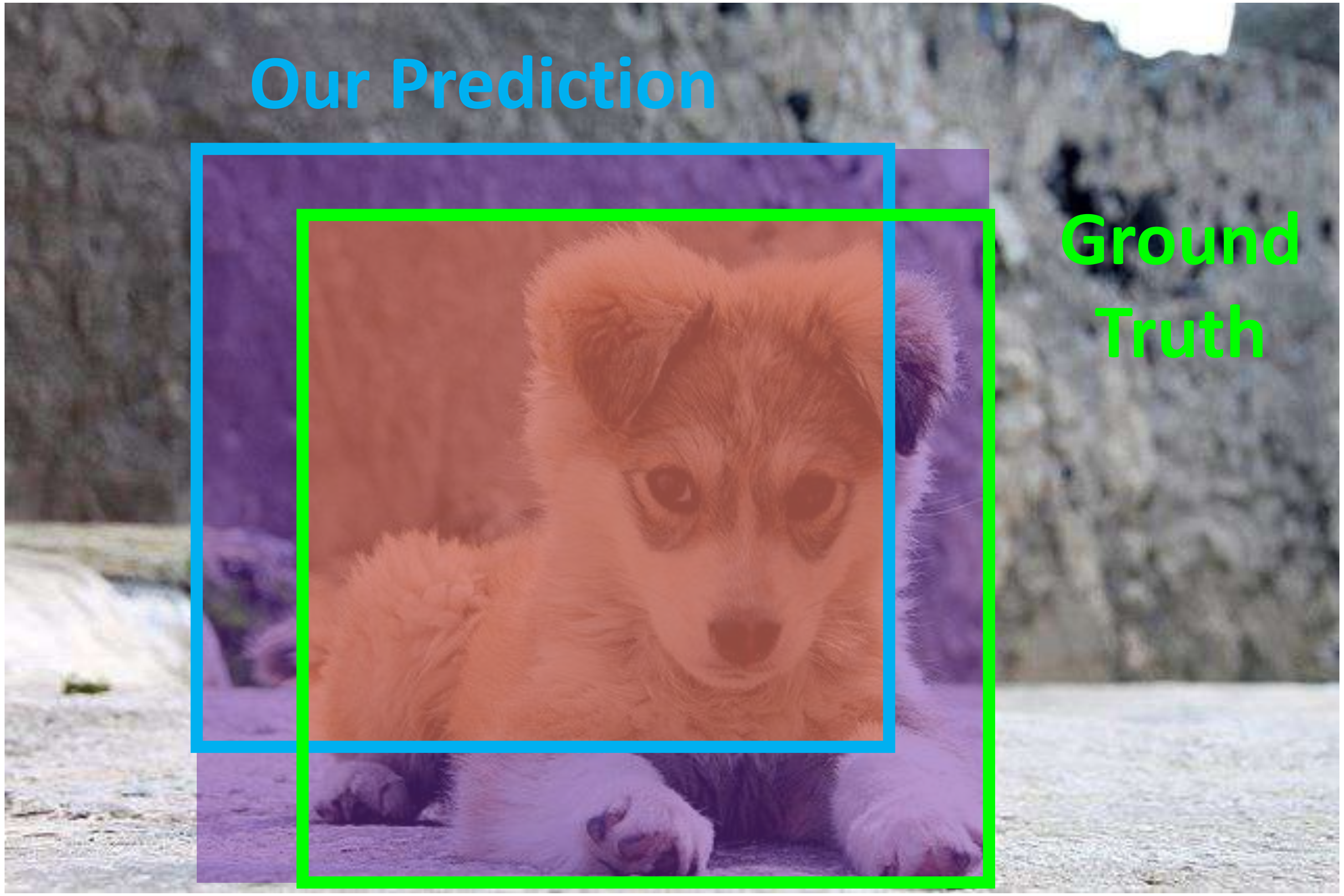


# Comparing Boxes: Intersection over Union (IoU)

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

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

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



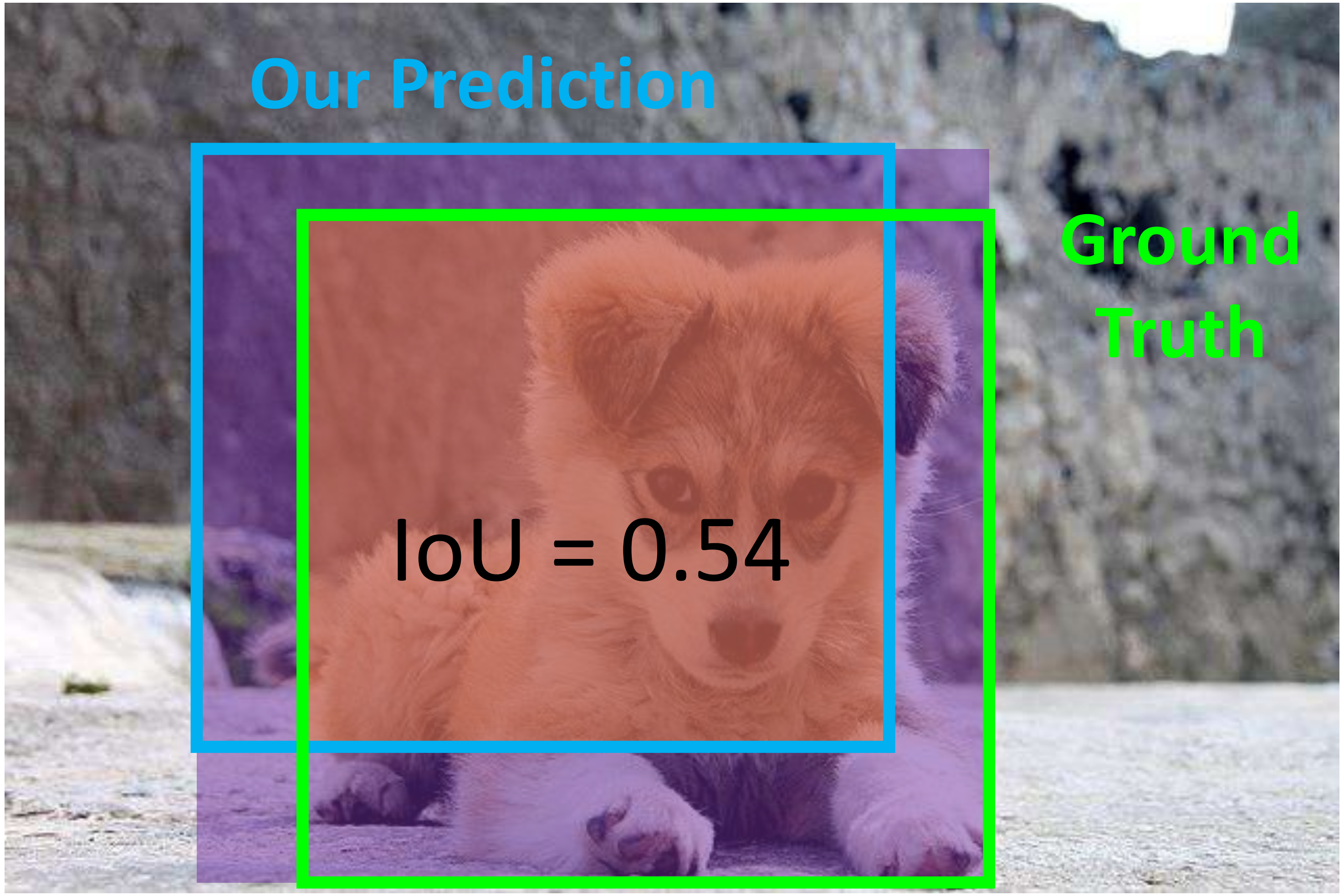
# Comparing Boxes: Intersection over Union (IoU)

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

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

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

IoU > 0.5 is “decent”



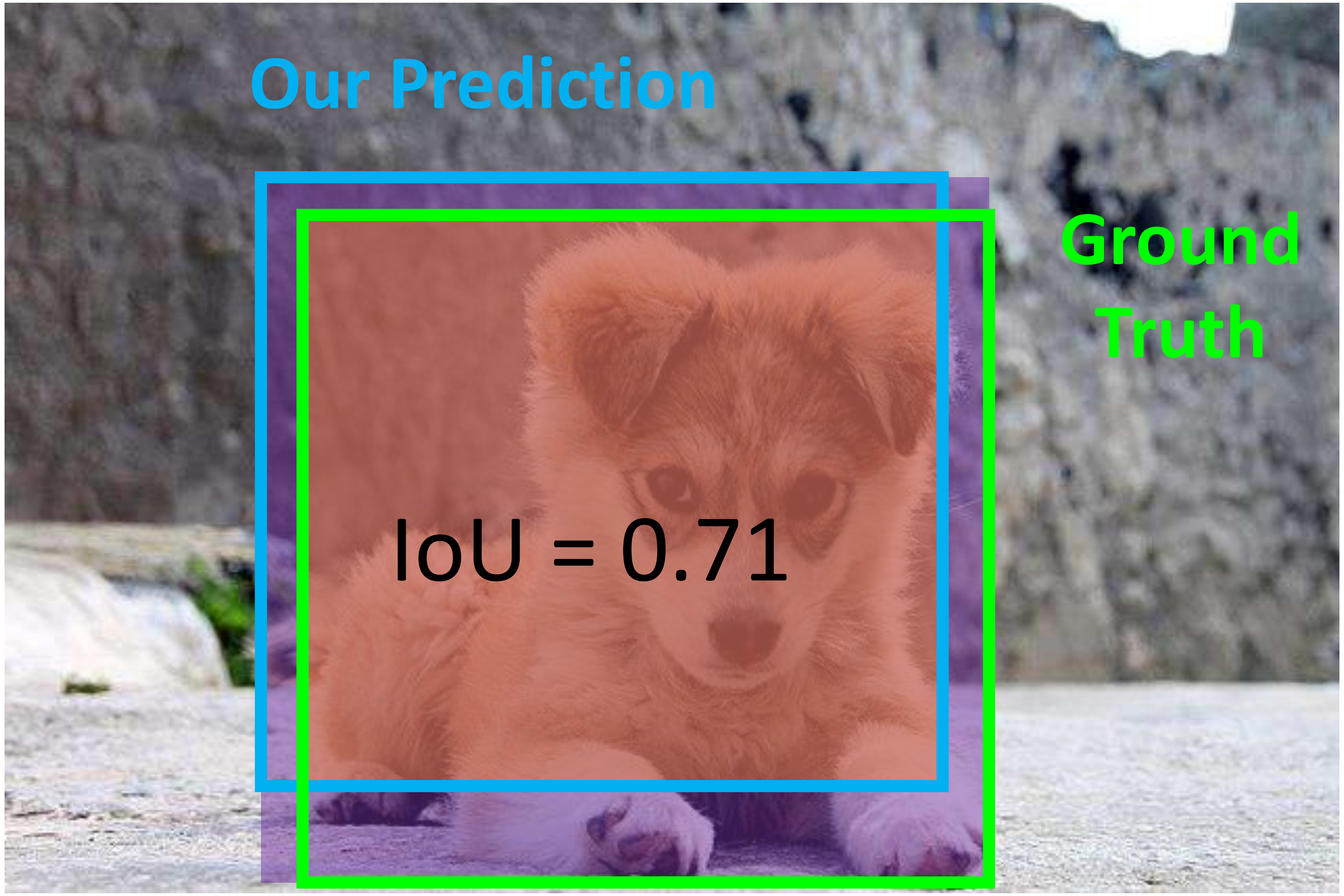
# Comparing Boxes: Intersection over Union (IoU)

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?

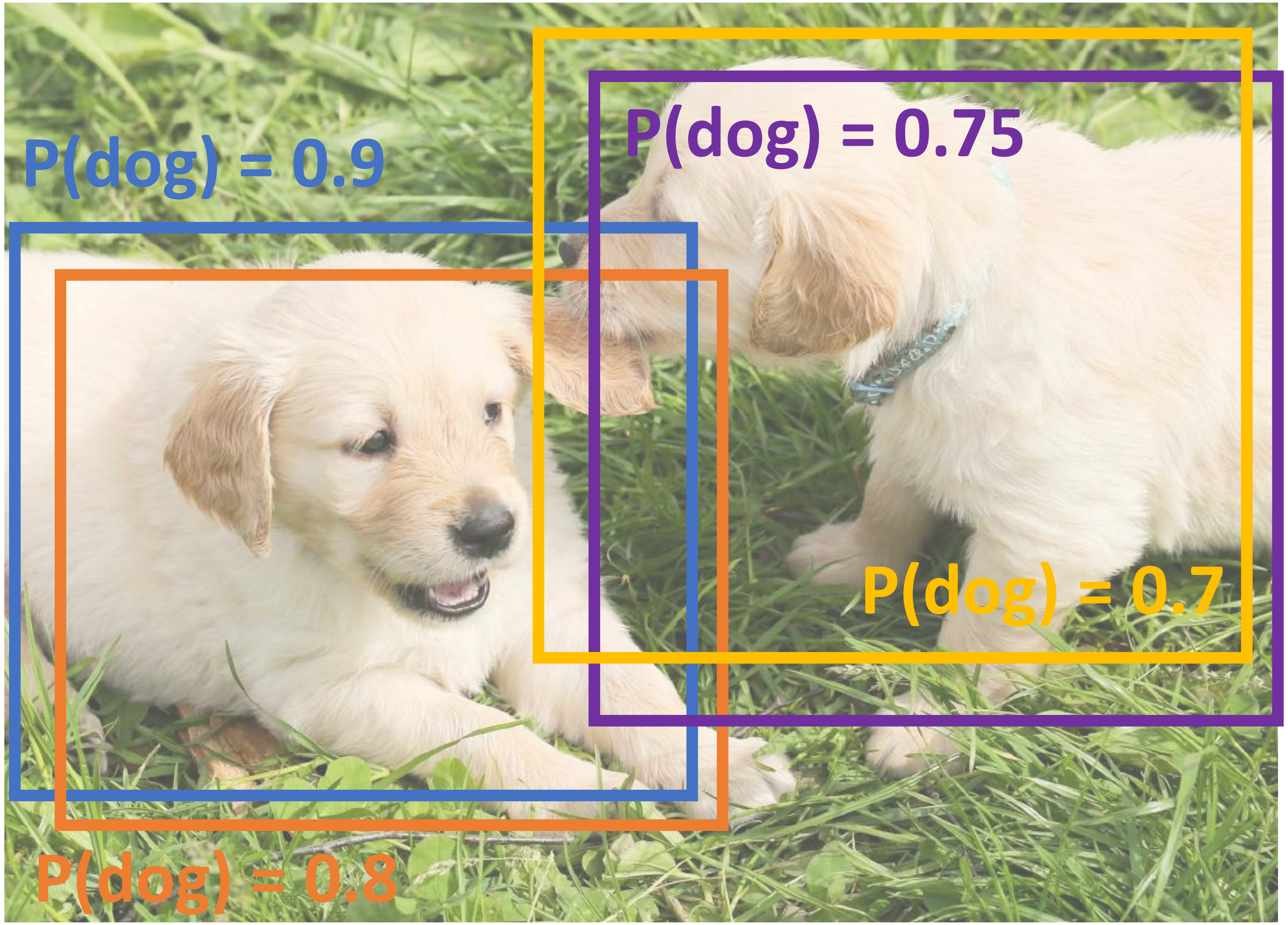
**Intersection over Union** (IoU)
(Also called "Jaccard similarity" or "Jaccard index"):

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

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

# Overlapping Boxes

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



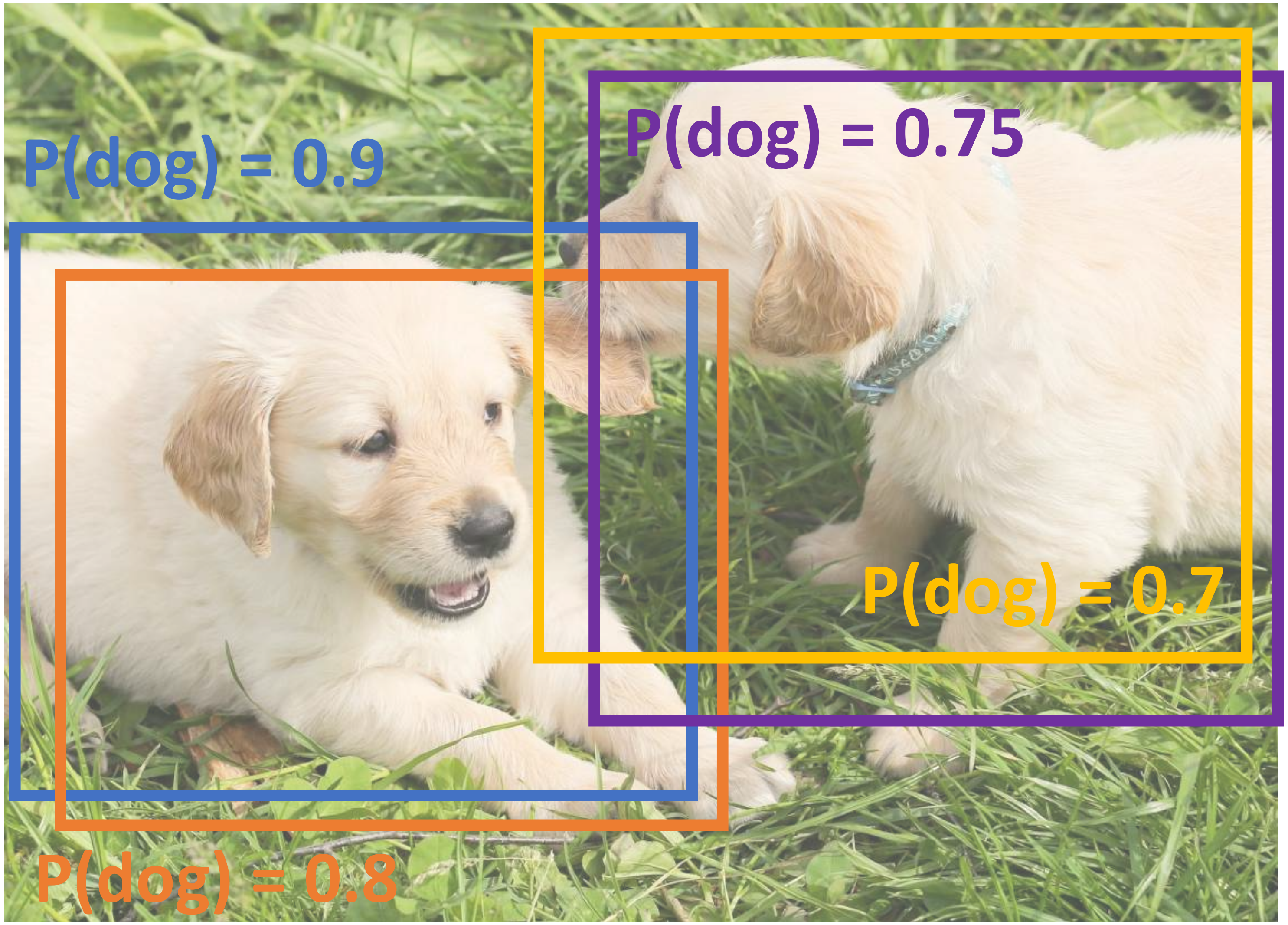

Puppy image is CC0 Public Domain

# 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 IoU > threshold (e.g. 0.7)
3. If any boxes remain, GOTO 1



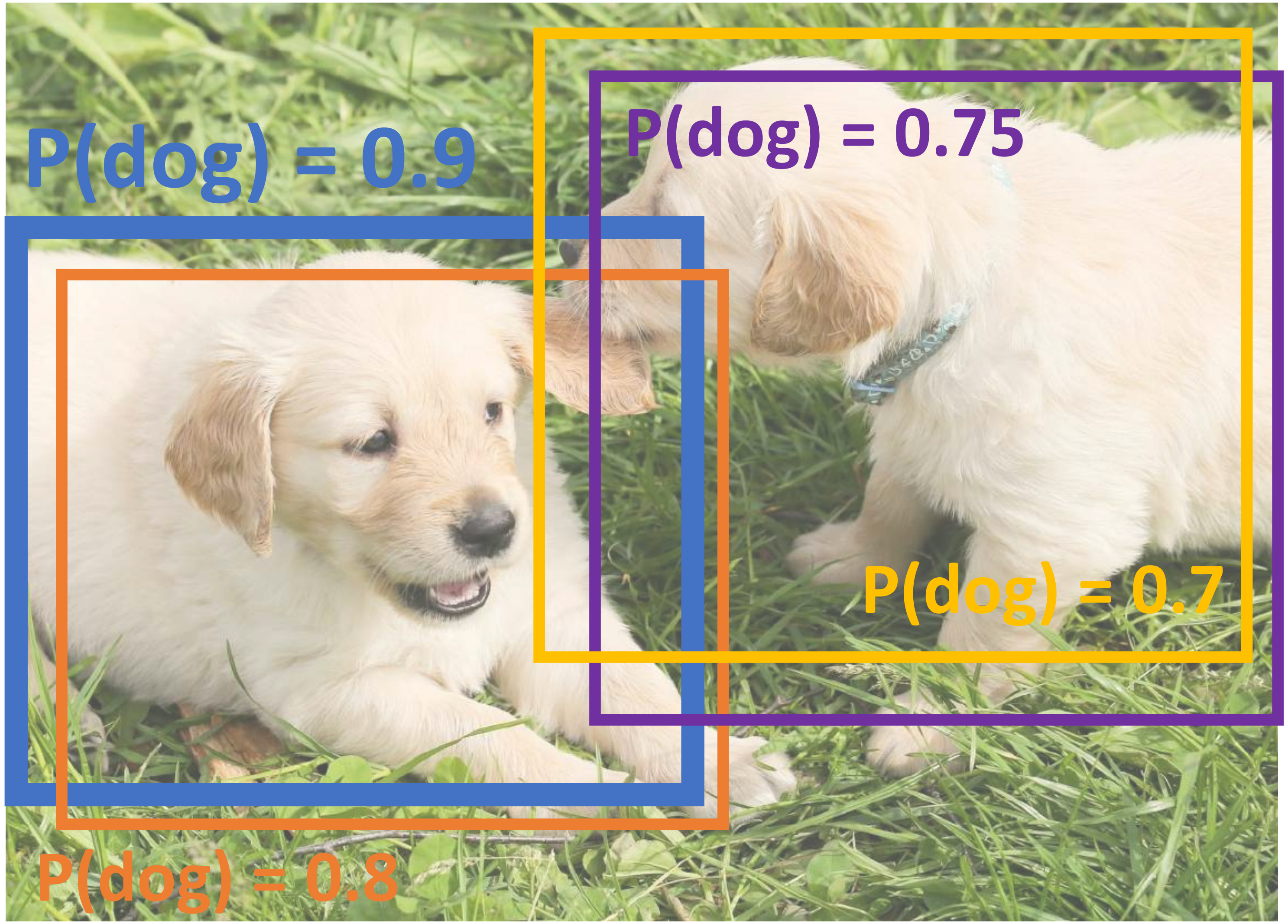
# Overlapping Boxes: **Non-Max Suppression (NMS)**

**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 IoU > threshold (e.g. 0.7)
3. If any boxes remain, GOTO 1

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



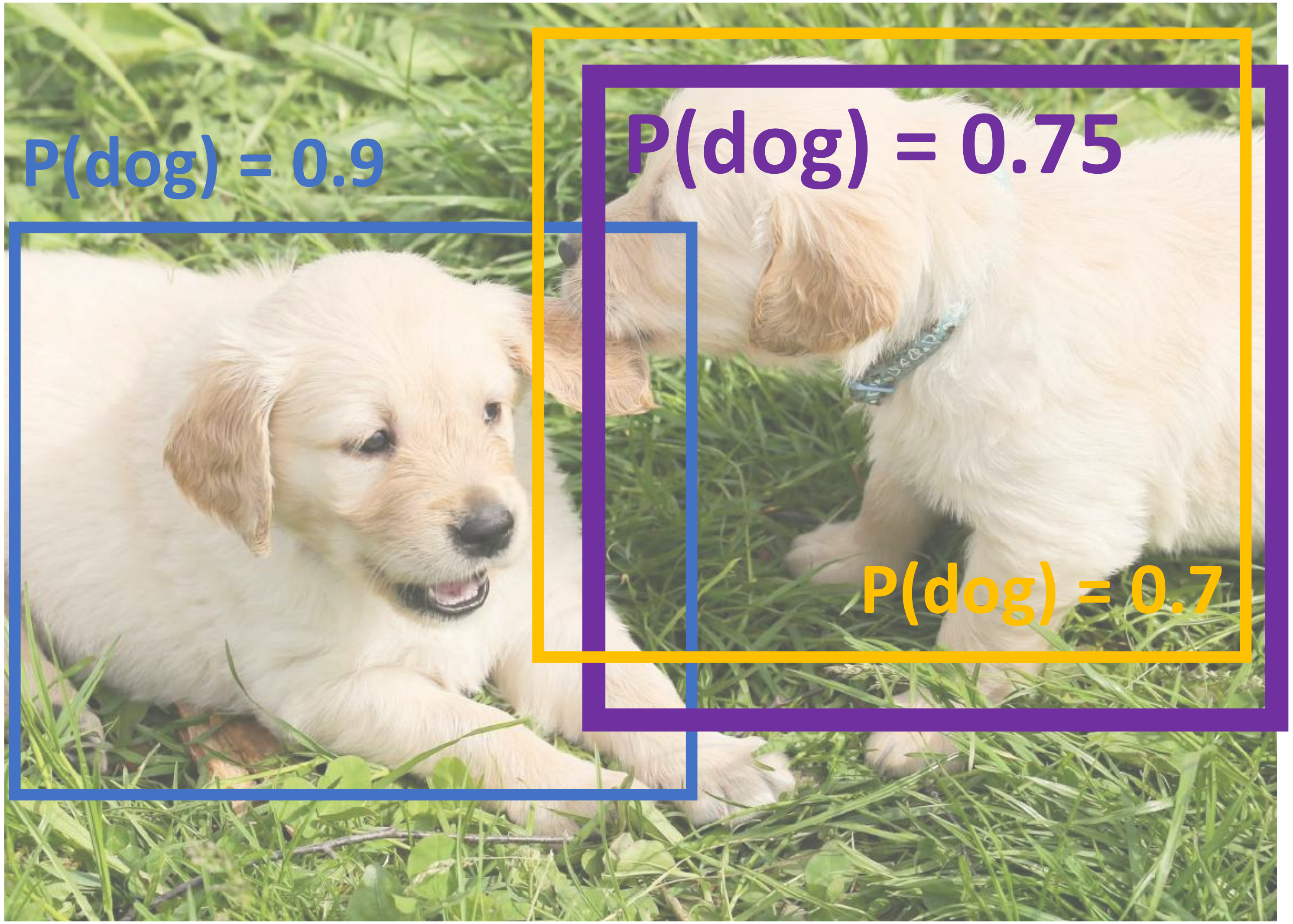
# Overlapping Boxes: **Non-Max Suppression (NMS)**

**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 IoU > threshold (e.g. 0.7)
3. If any boxes remain, GOTO 1

IoU(■, ■) = **0.74**

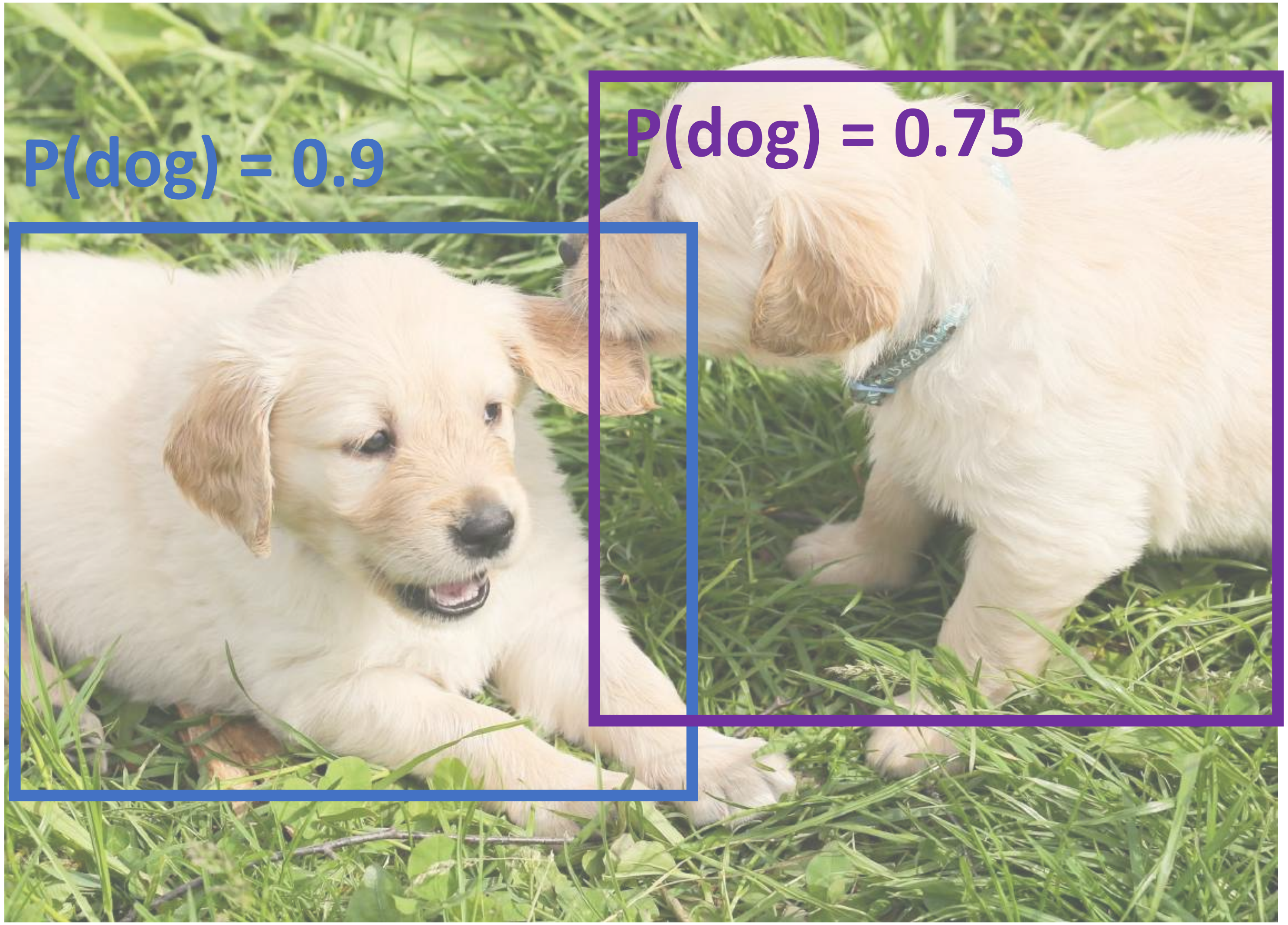


# Overlapping Boxes: **Non-Max Suppression (NMS)**

**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 IoU > threshold (e.g. 0.7)
3. If any boxes remain, GOTO 1



# 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 IoU > threshold (e.g. 0.7)
3. If any boxes remain, GOTO 1

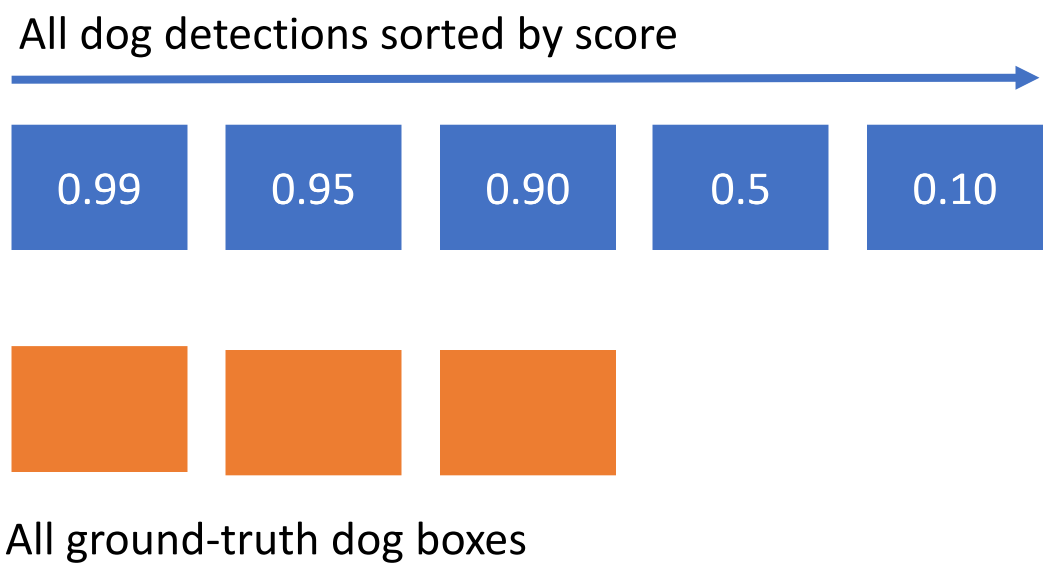
**Problem**: NMS may eliminate "good" boxes when objects are highly overlapping... no good solution =(

# Evaluating Object Detectors: Mean Average Precision (mAP)

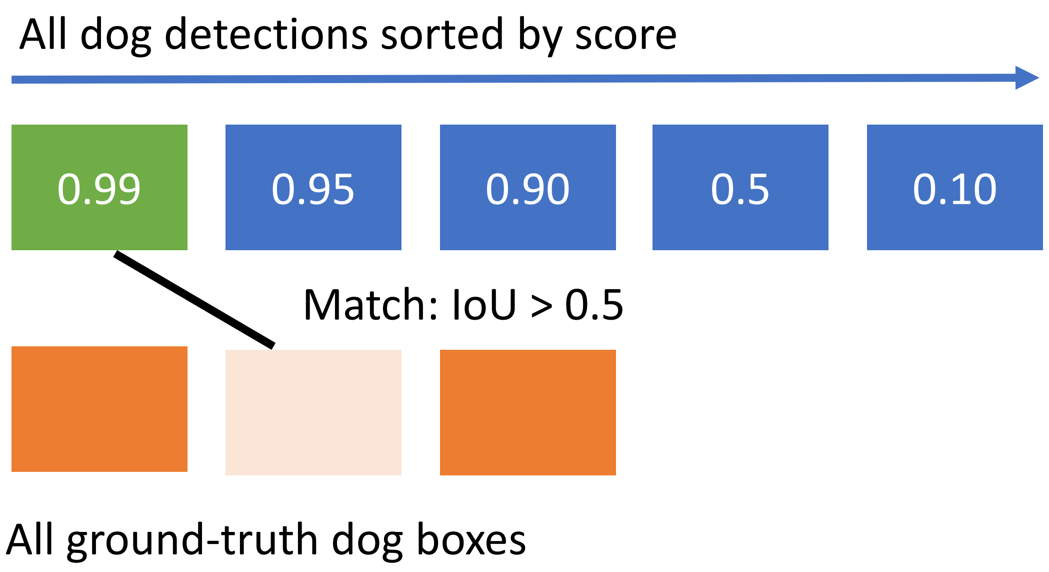
1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve

# Evaluating Object Detectors: Mean Average Precision (mAP)

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



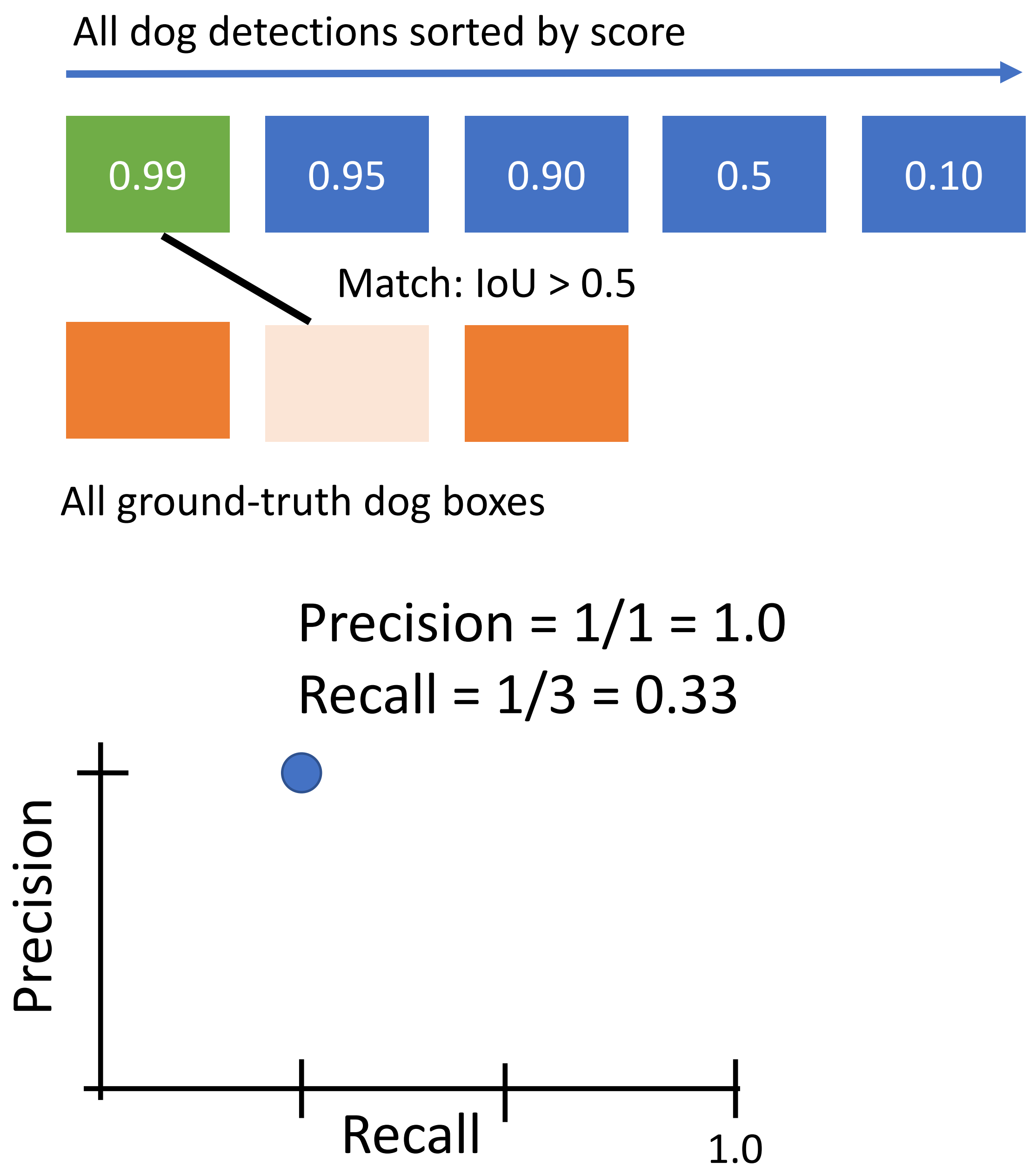
# Evaluating Object Detectors: Mean Average Precision (mAP)




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

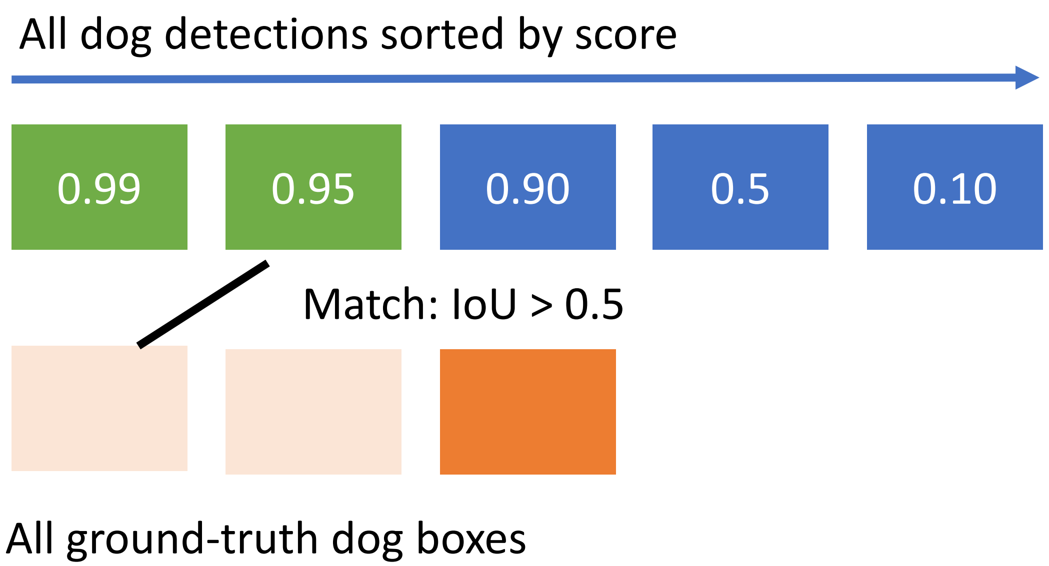
# Evaluating Object Detectors: Mean Average Precision (mAP)

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

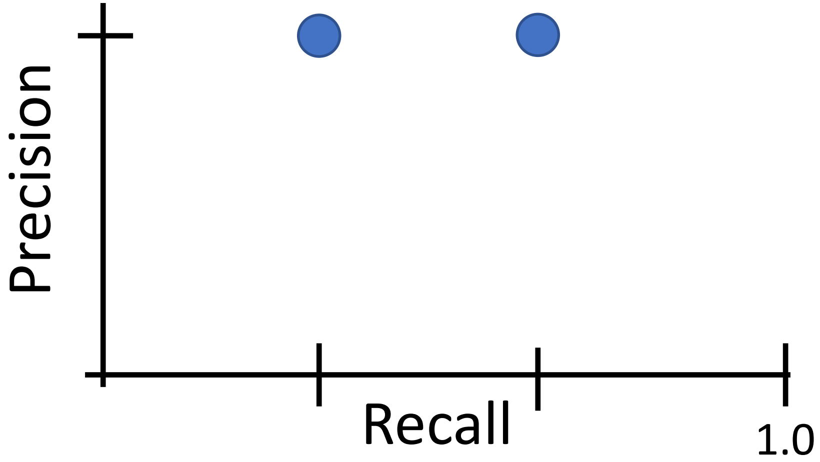


# Evaluating Object Detectors: Mean Average Precision (mAP)

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

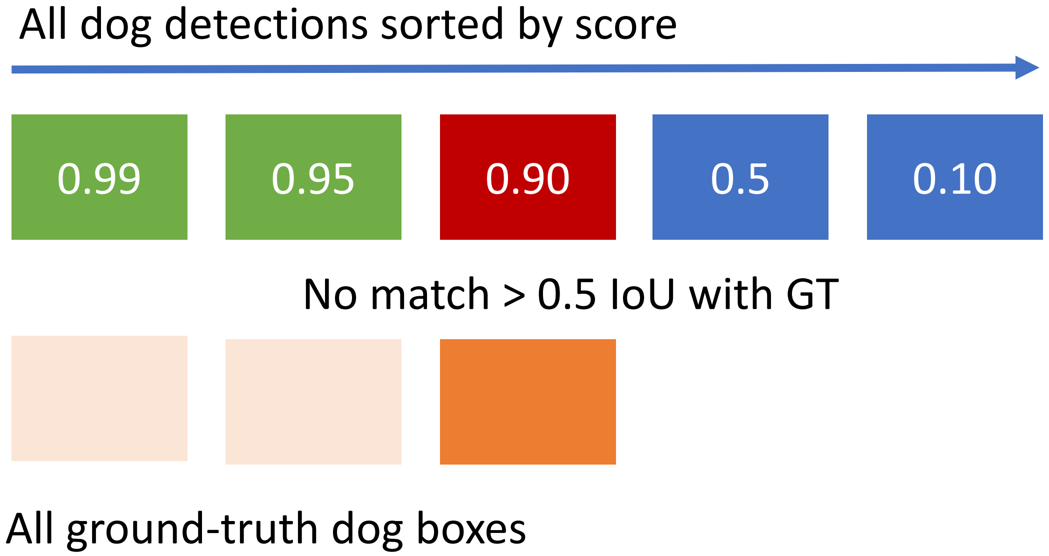



Precision = 2/2 = 1.0
Recall = 2/3 = 0.67

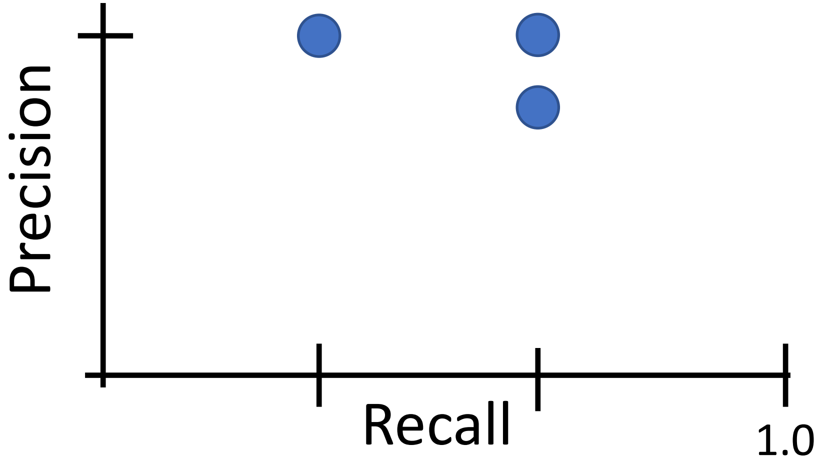


# Evaluating Object Detectors: Mean Average Precision (mAP)

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

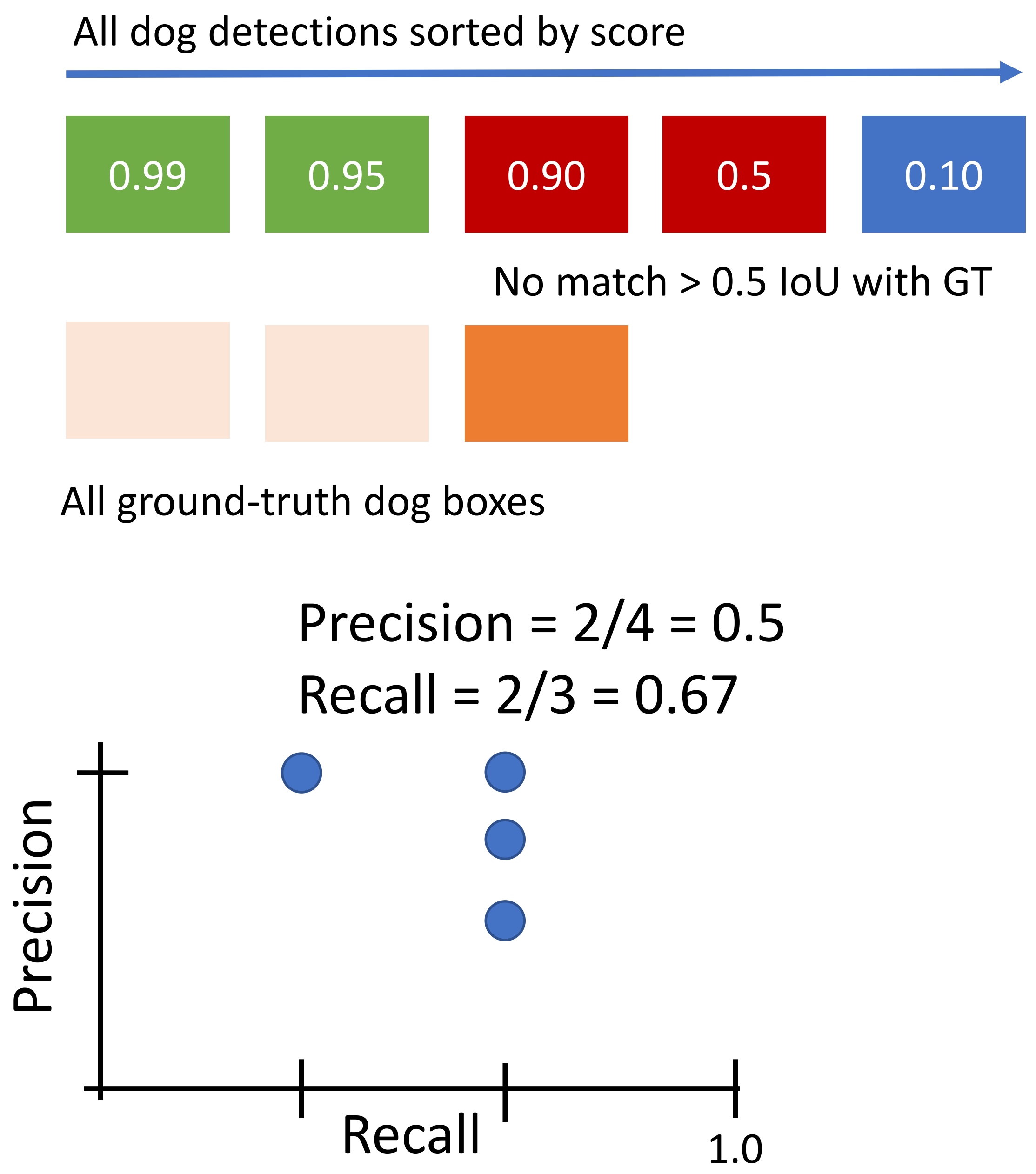



Precision = 2/3 = 0.67
Recall = 2/3 = 0.67



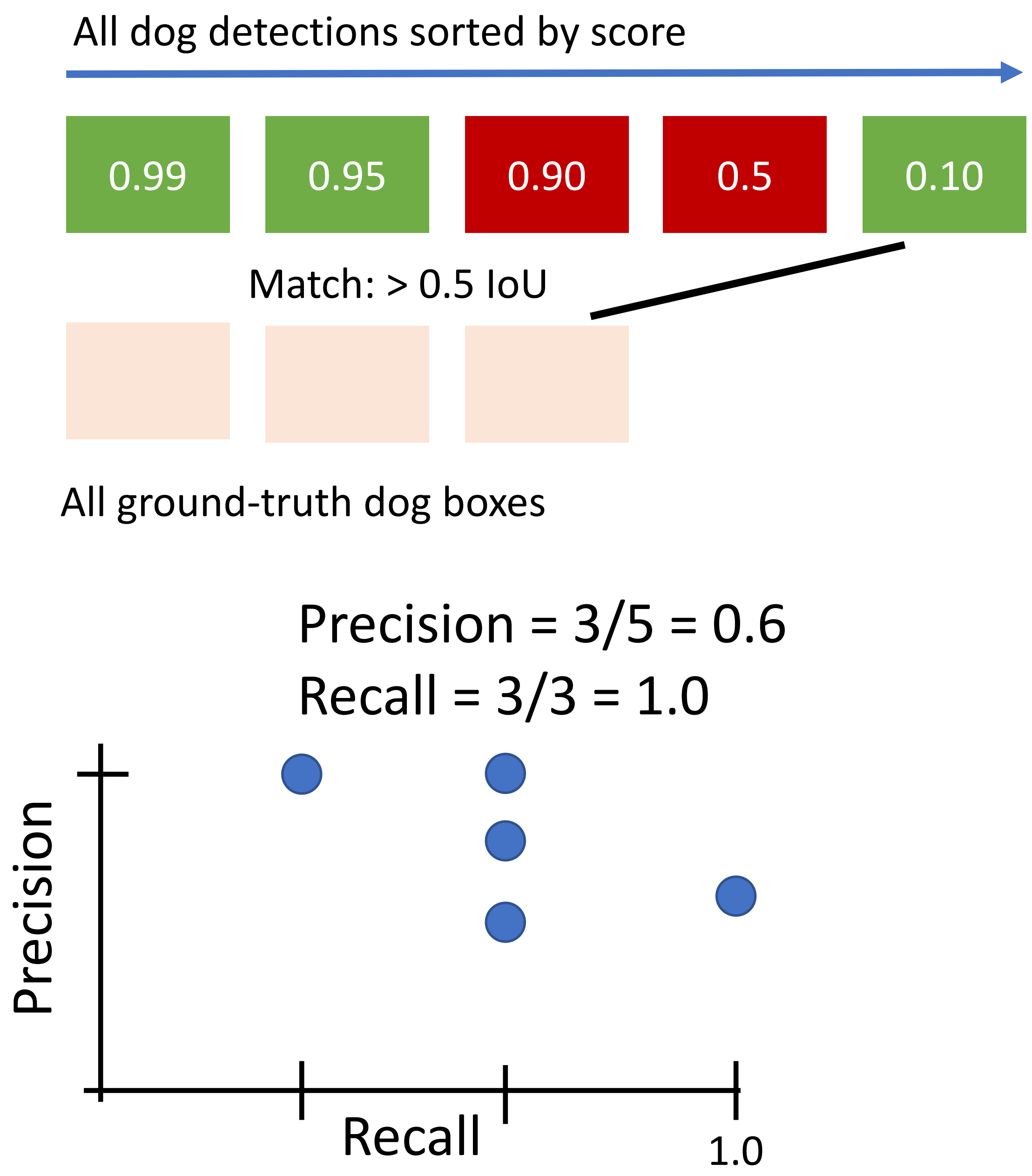
# Evaluating Object Detectors: Mean Average Precision (mAP)

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



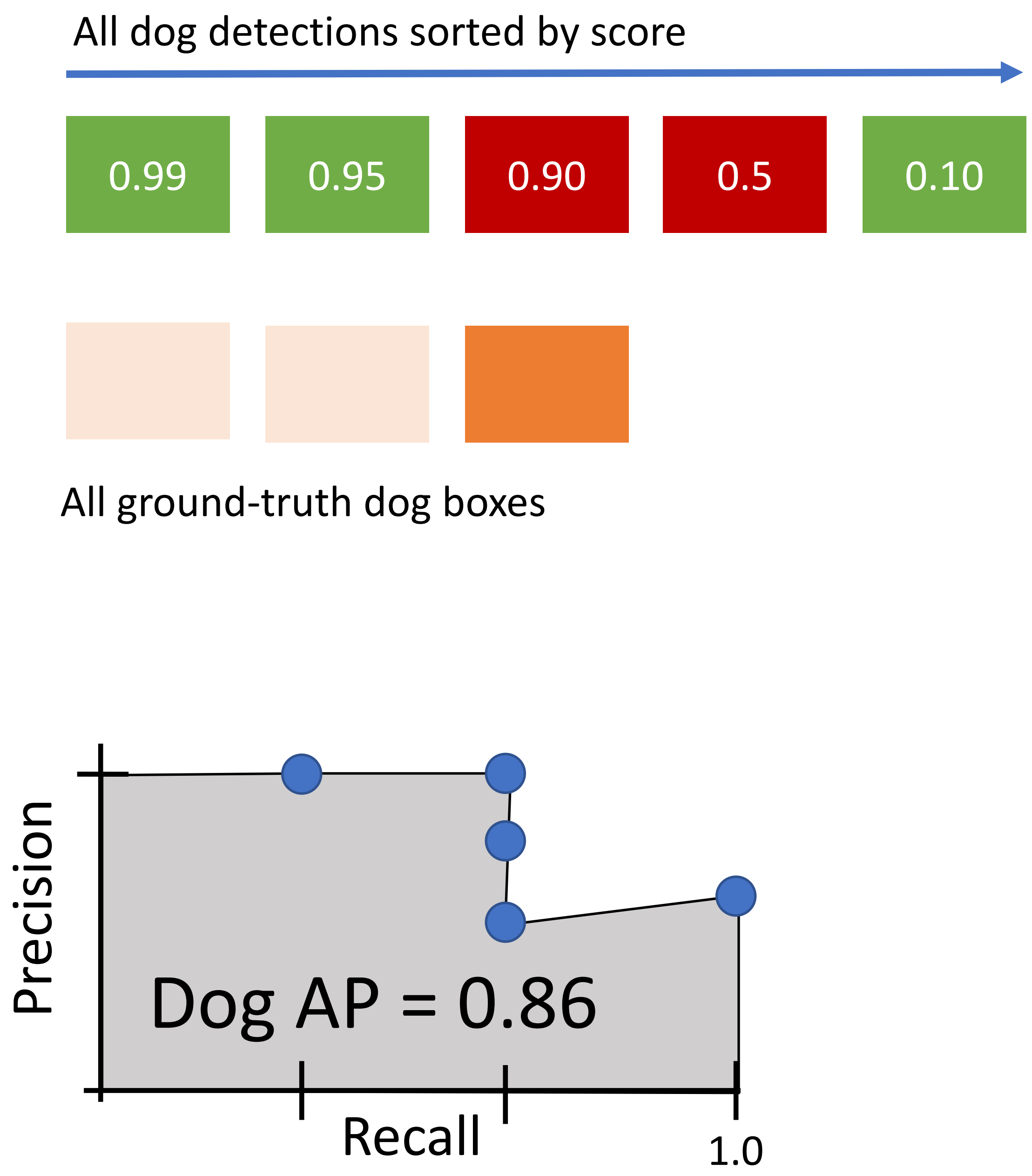
# Evaluating Object Detectors: Mean Average Precision (mAP)

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



# Evaluating Object Detectors: Mean Average Precision (mAP)

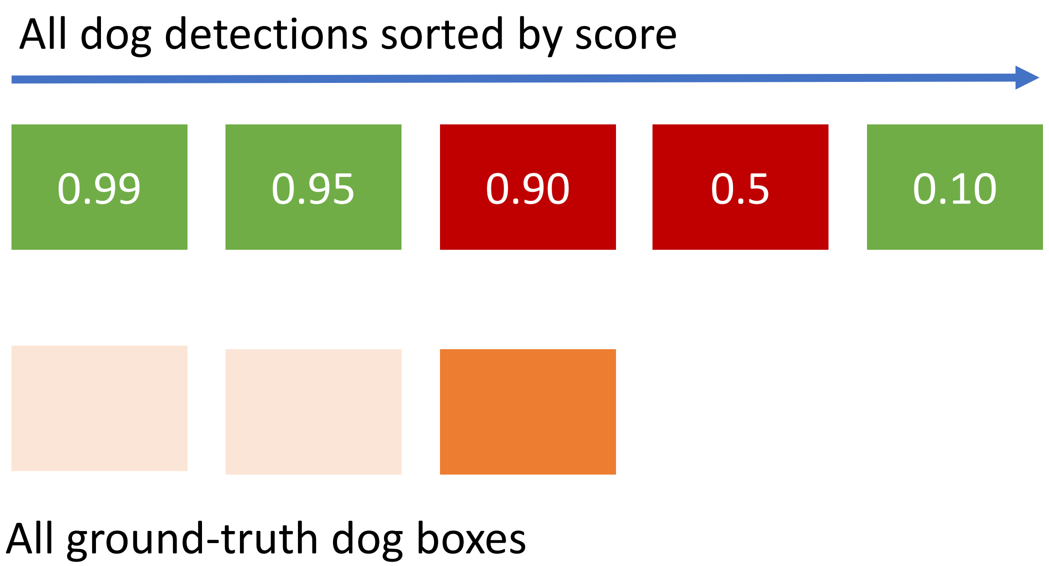
1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
   1. For each detection (highest score to lowest score)
      1. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
      2.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

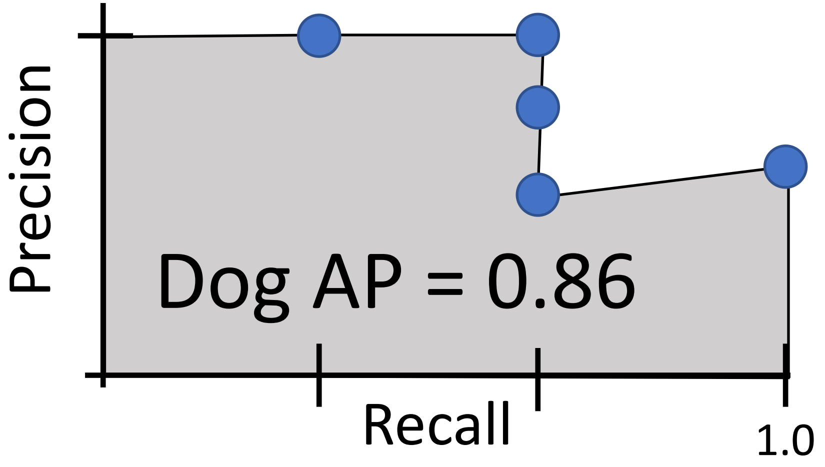


# Evaluating Object Detectors: Mean Average Precision (mAP)

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

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



# Evaluating Object Detectors: Mean Average Precision (mAP)

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

Car AP = 0.65

Cat AP = 0.80

Dog AP = 0.86

mAP@0.5 = 0.77

# Evaluating Object Detectors: Mean Average Precision (mAP)

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

mAP@0.5 = 0.77

mAP@0.55 = 0.71

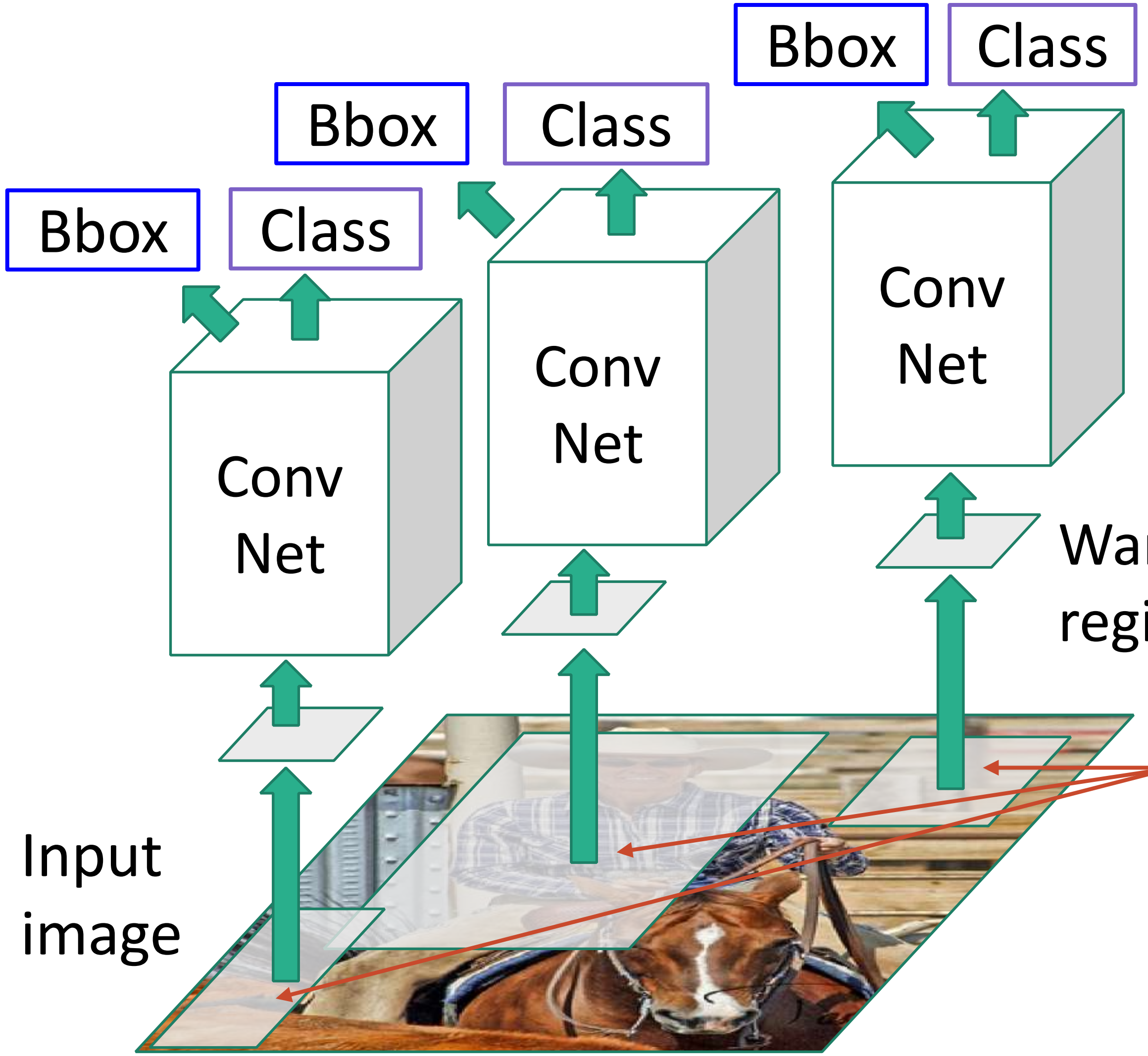
mAP@0.60 = 0.65

…

mAP@0.95 = 0.2

COCO mAP = 0.4

# R-CNN: Region-Based CNN




Classify each region

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

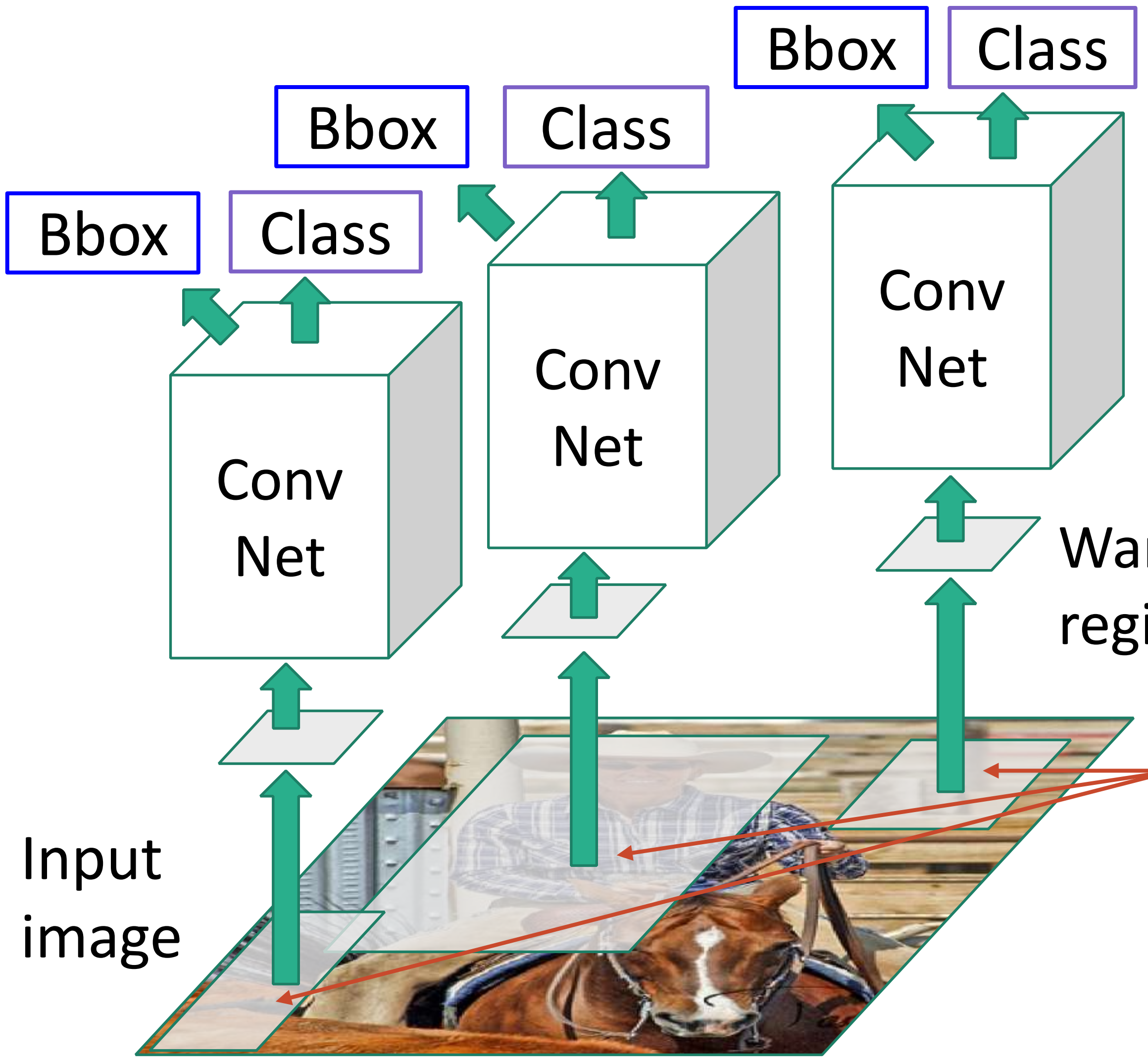
Forward each region through ConvNet

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.

# 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

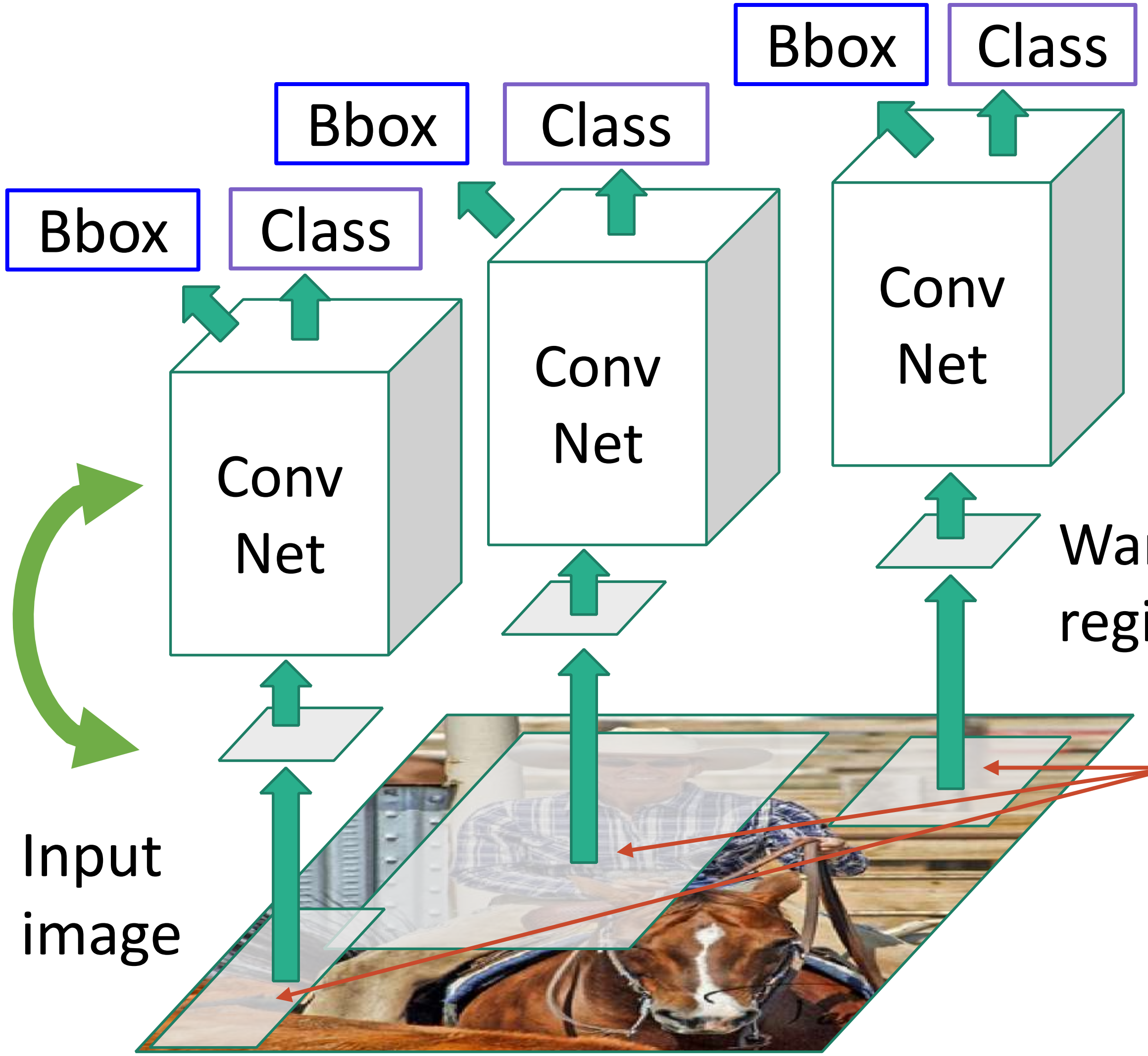
Warped image regions (224x224)

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.

# 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)

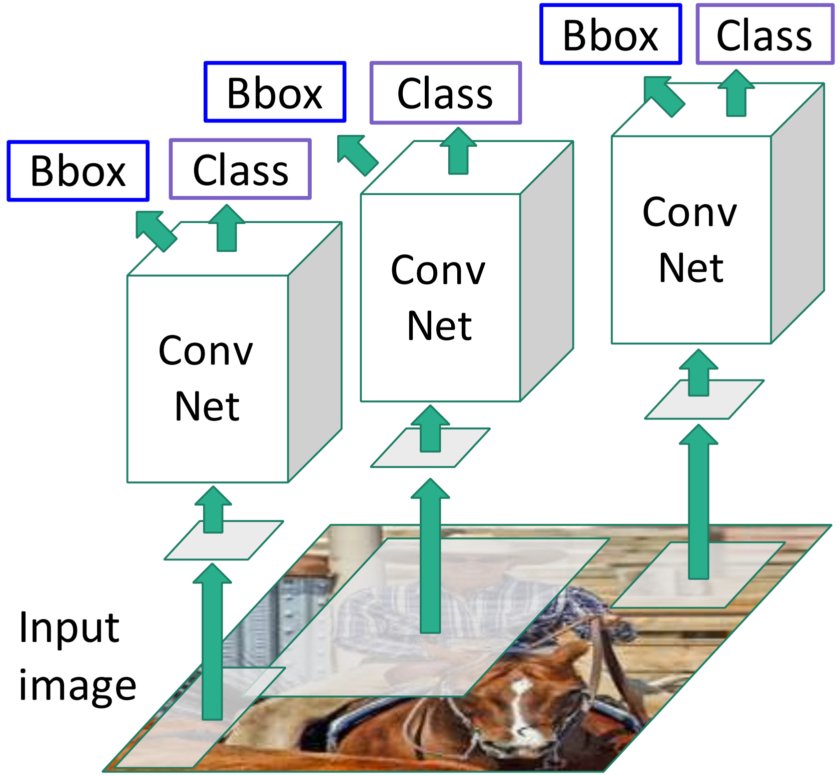
Regions of Interest (RoI) from a proposal method (~2k)

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

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

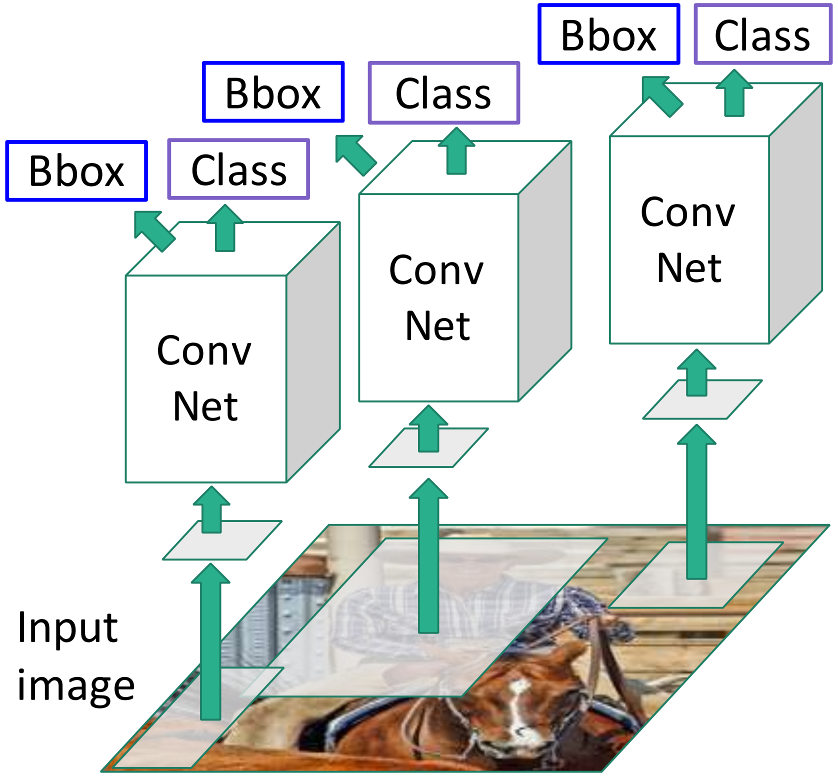
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

<u>"Slow" R-CNN</u>
Process each region independently



# Fast R-CNN

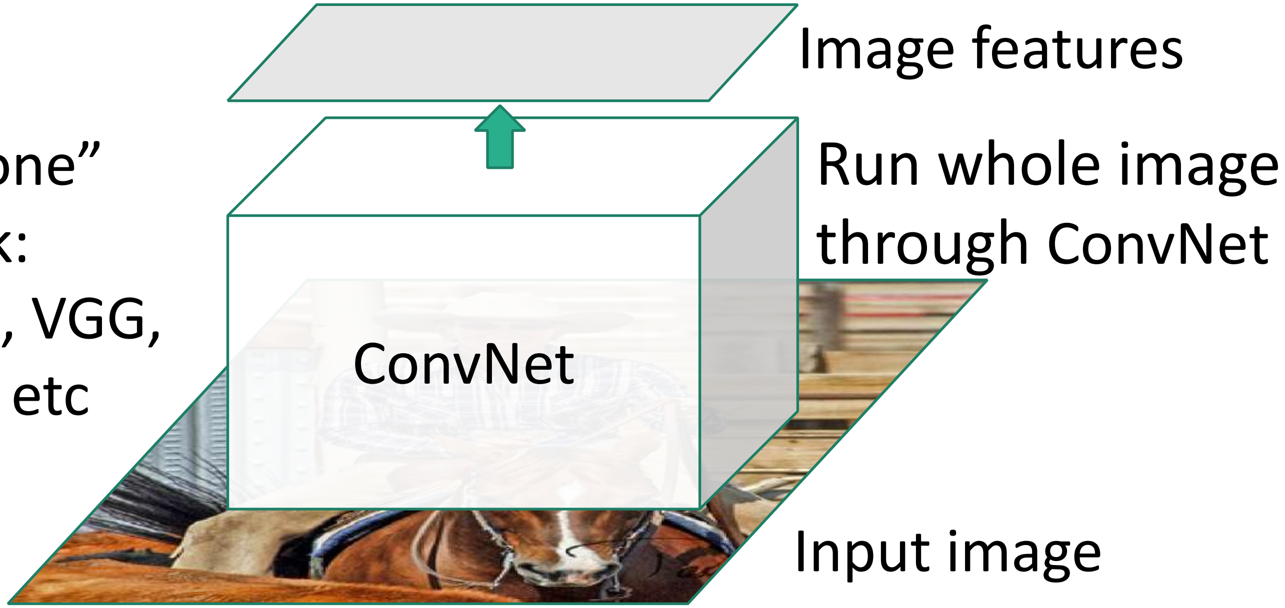

"Slow" R-CNN
Process each region independently

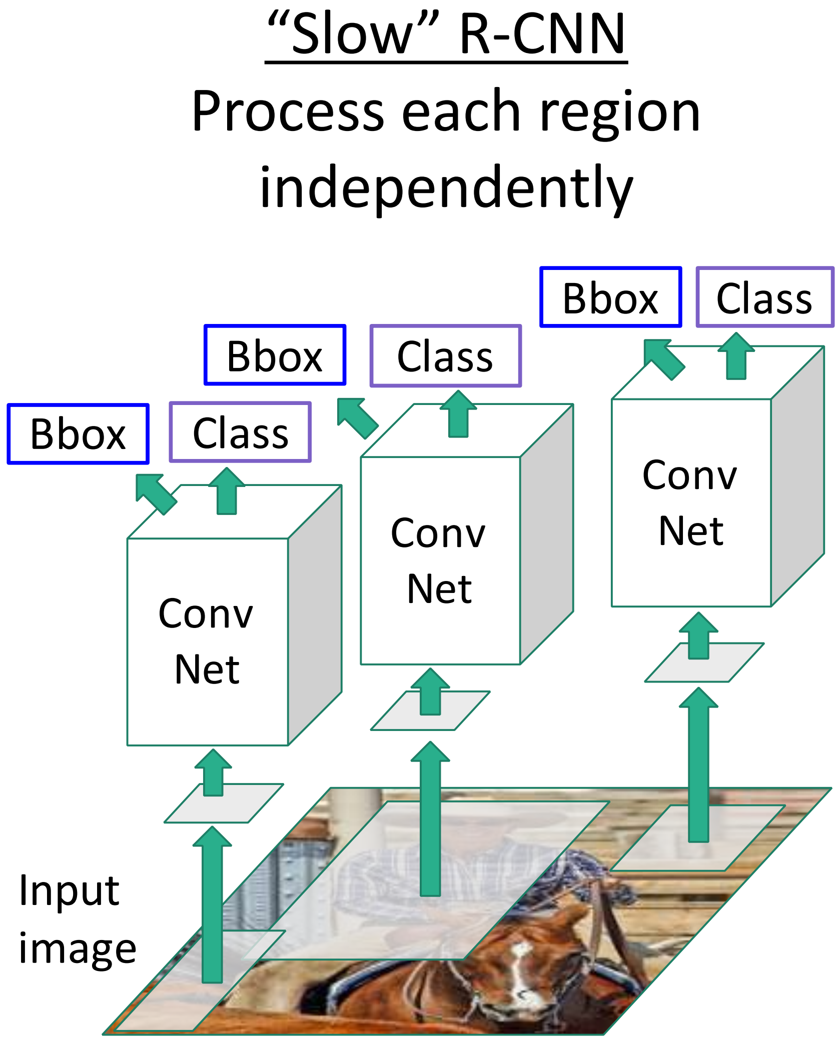



Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

# Fast R-CNN

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




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

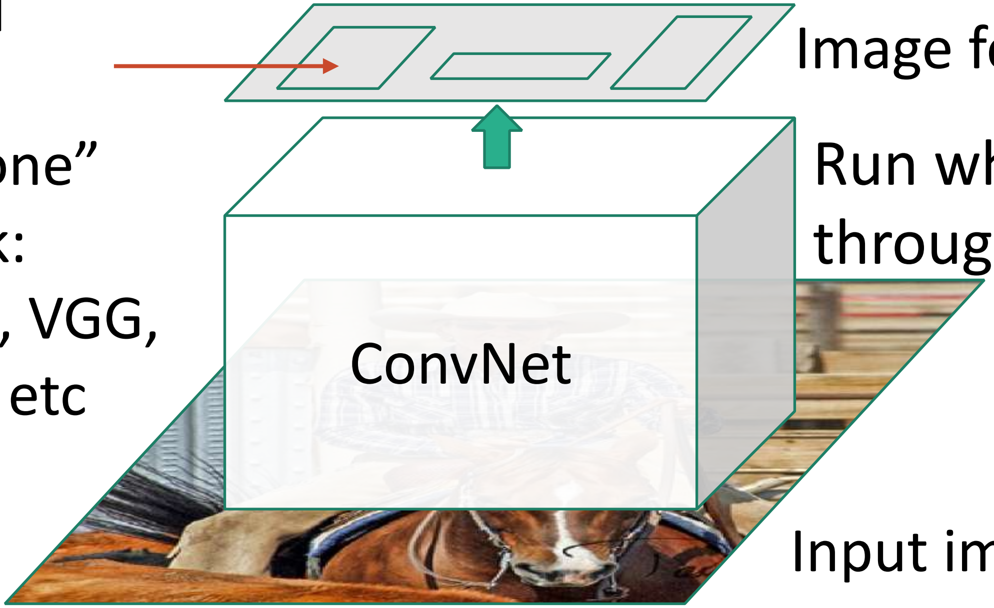
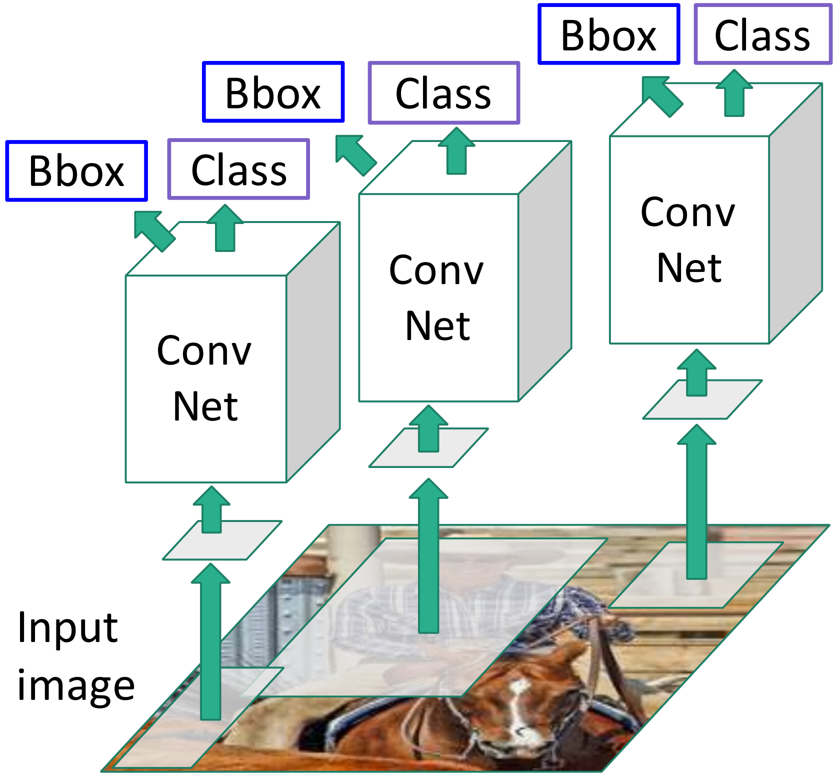



Image features

Run whole image through ConvNet

Input image

"Slow" R-CNN
Process each region independently

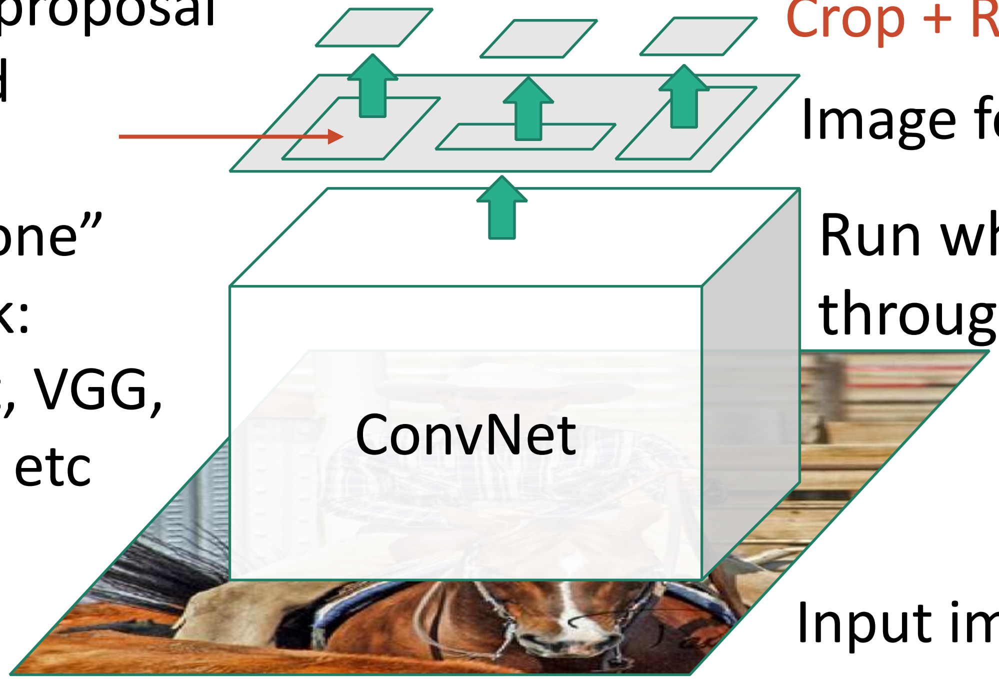



Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

# Fast R-CNN

Regions of Interest (RoIs) from a proposal method

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




Crop + Resize features

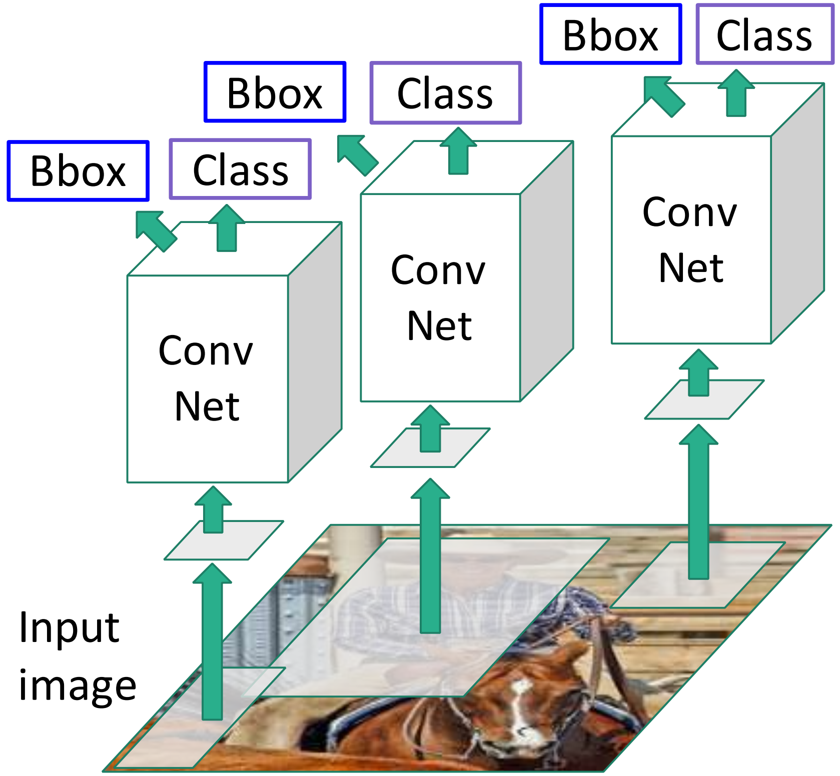
Image features

Run whole image through ConvNet

Input image

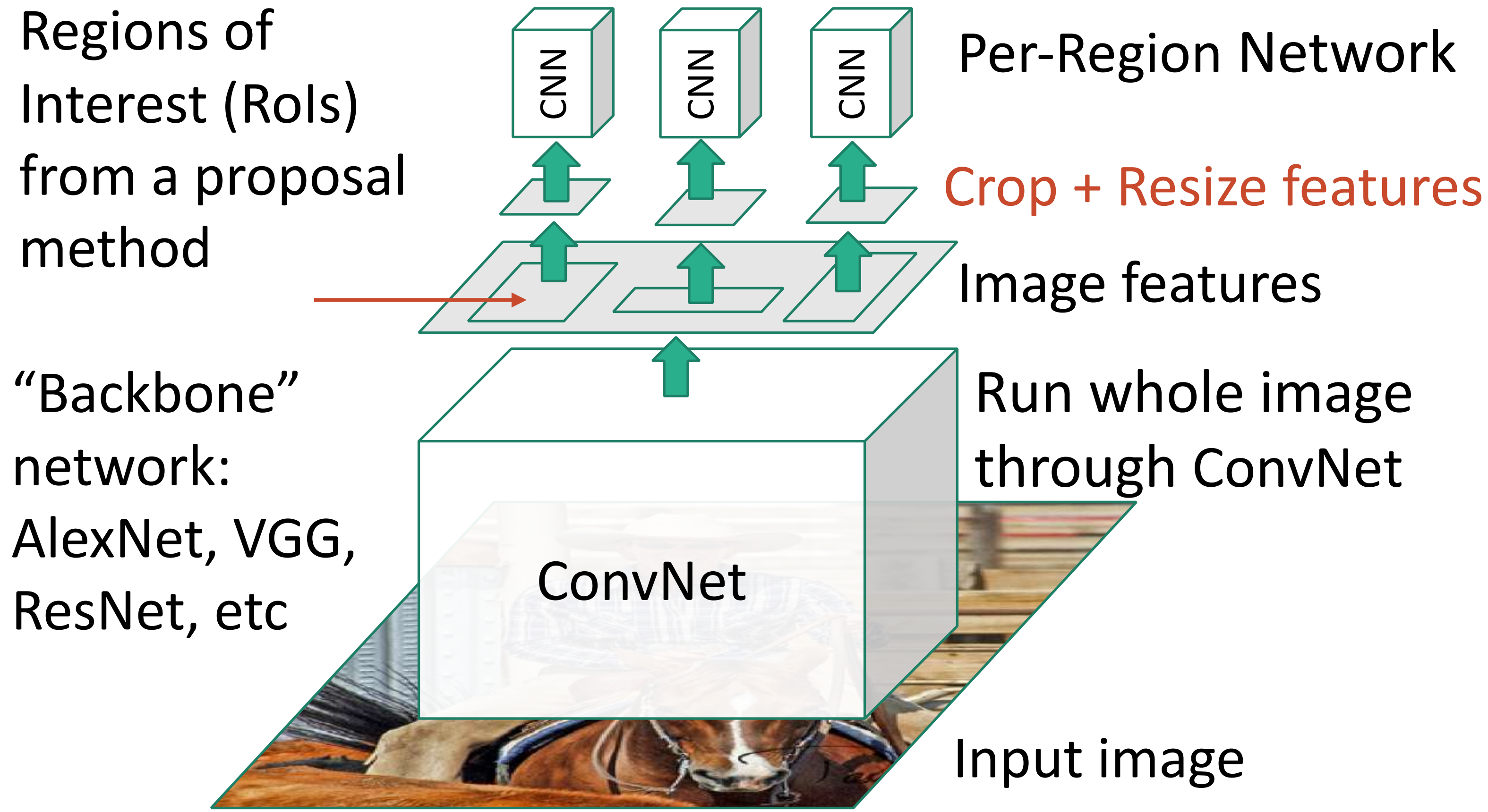
<u>"Slow" R-CNN</u>

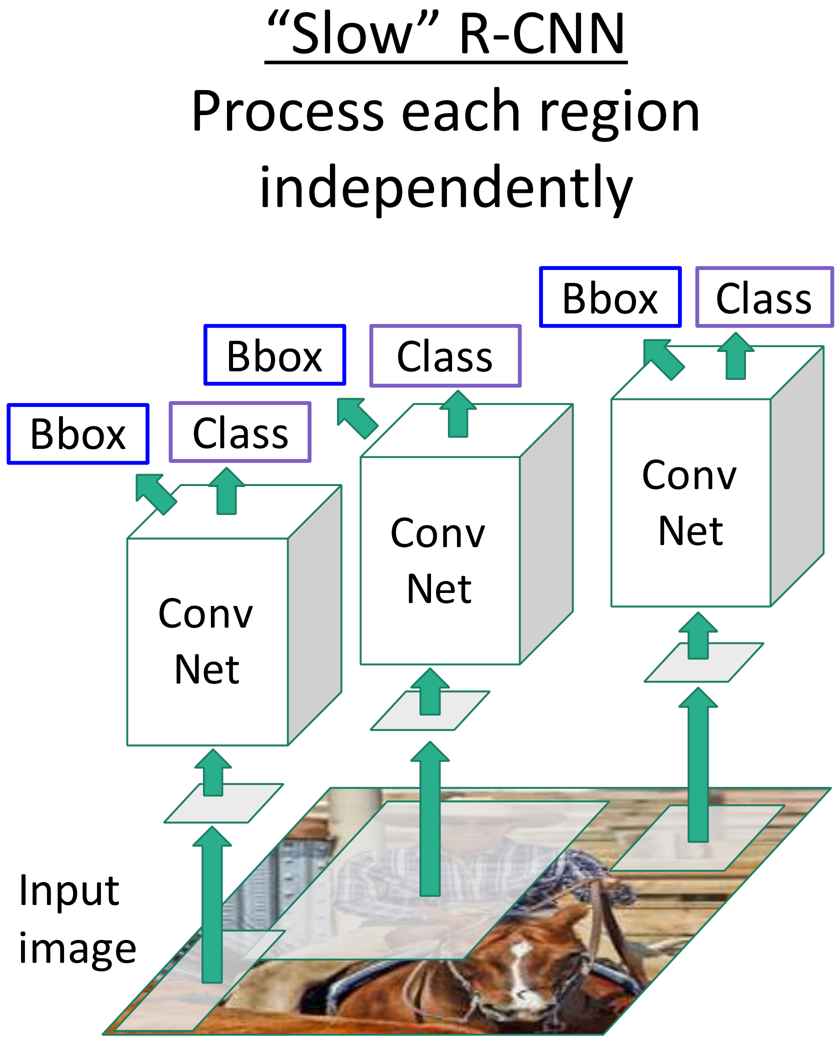
Process each region independently




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

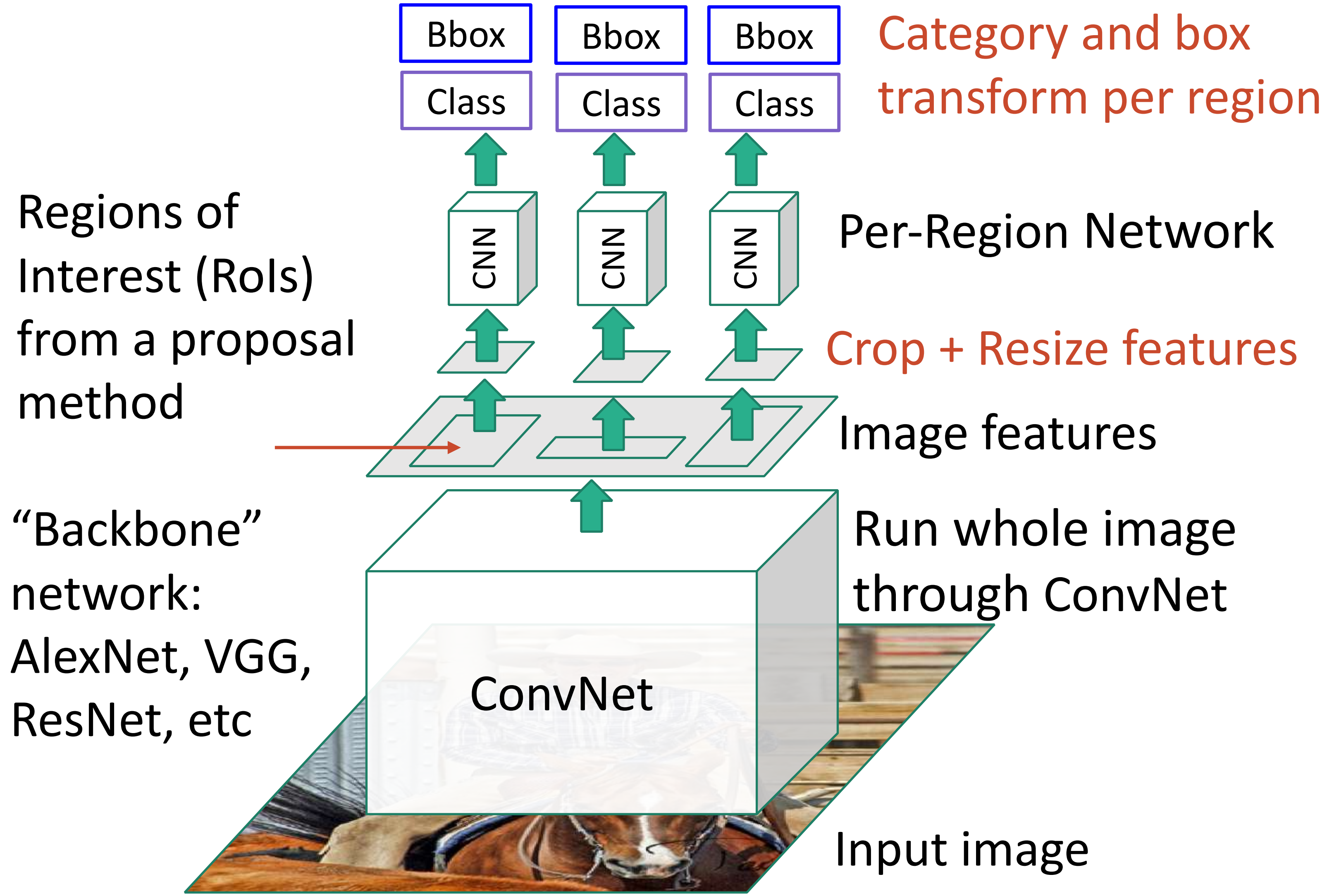
# Fast R-CNN

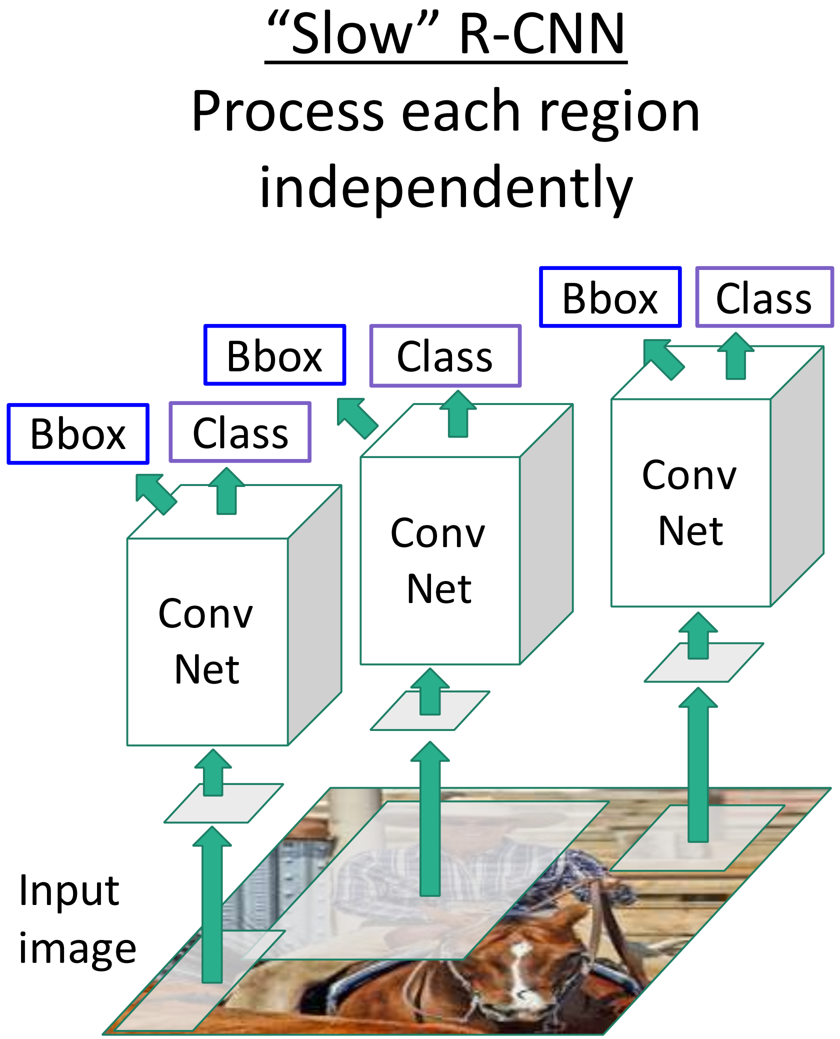



Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

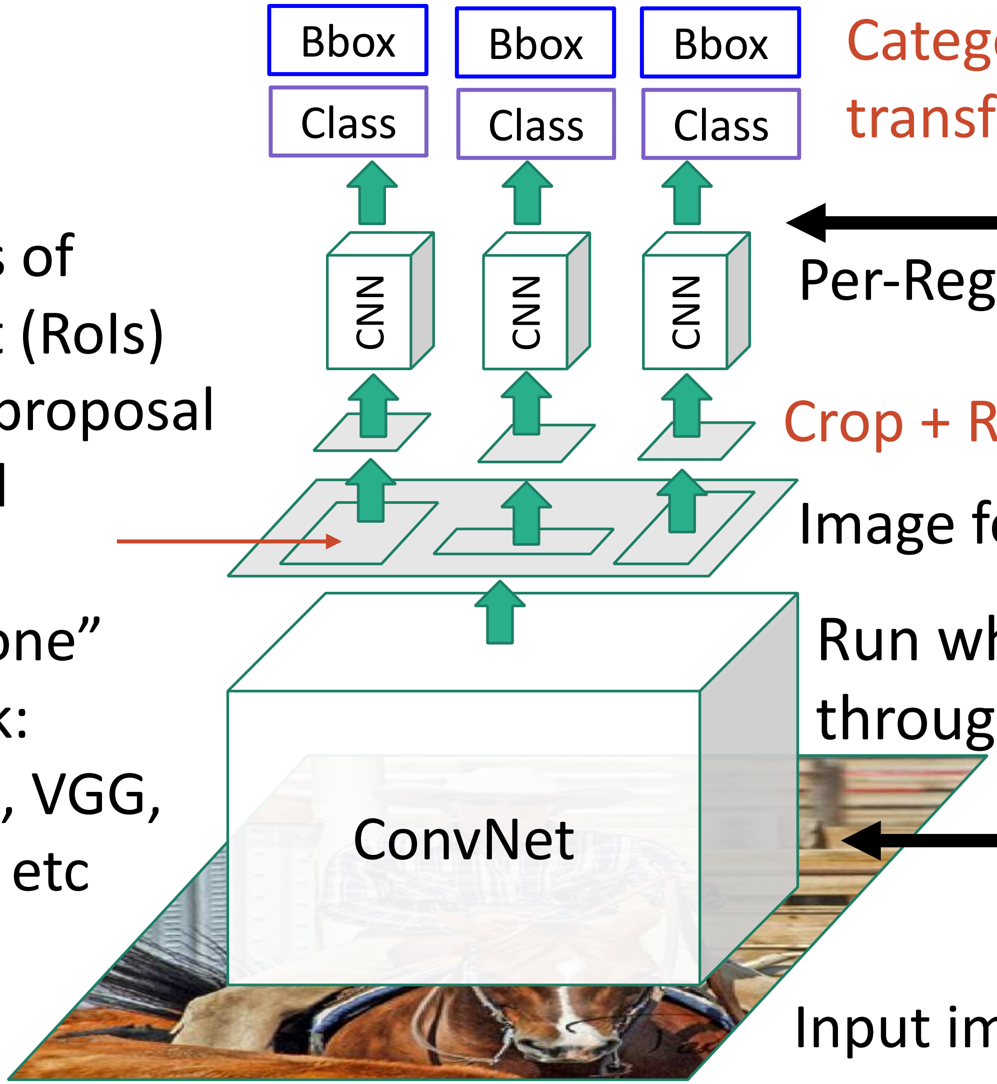
# Fast R-CNN




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

Regions of Interest (RoIs) from a proposal method

Per-Region Network

Crop + Resize features

Image features

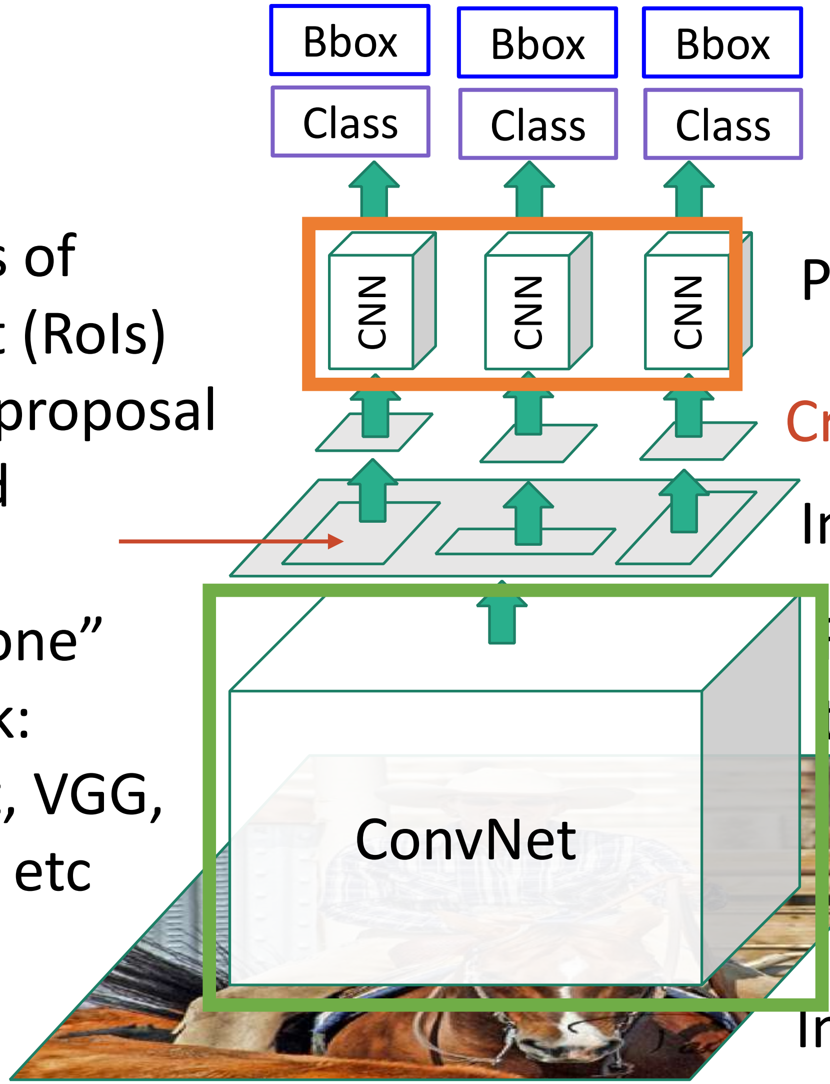
"Backbone" network: AlexNet, VGG, ResNet, etc

Run whole image through ConvNet

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

Input image

# Fast R-CNN




Category and box transform per region

Per-Region Network

Crop + Resize features

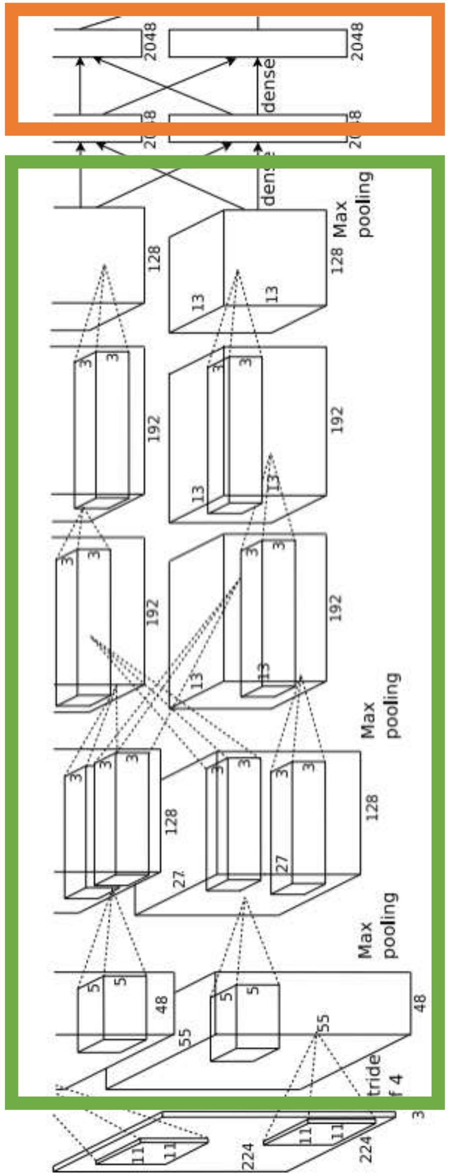
Image features

Run whole image through ConvNet

Input image

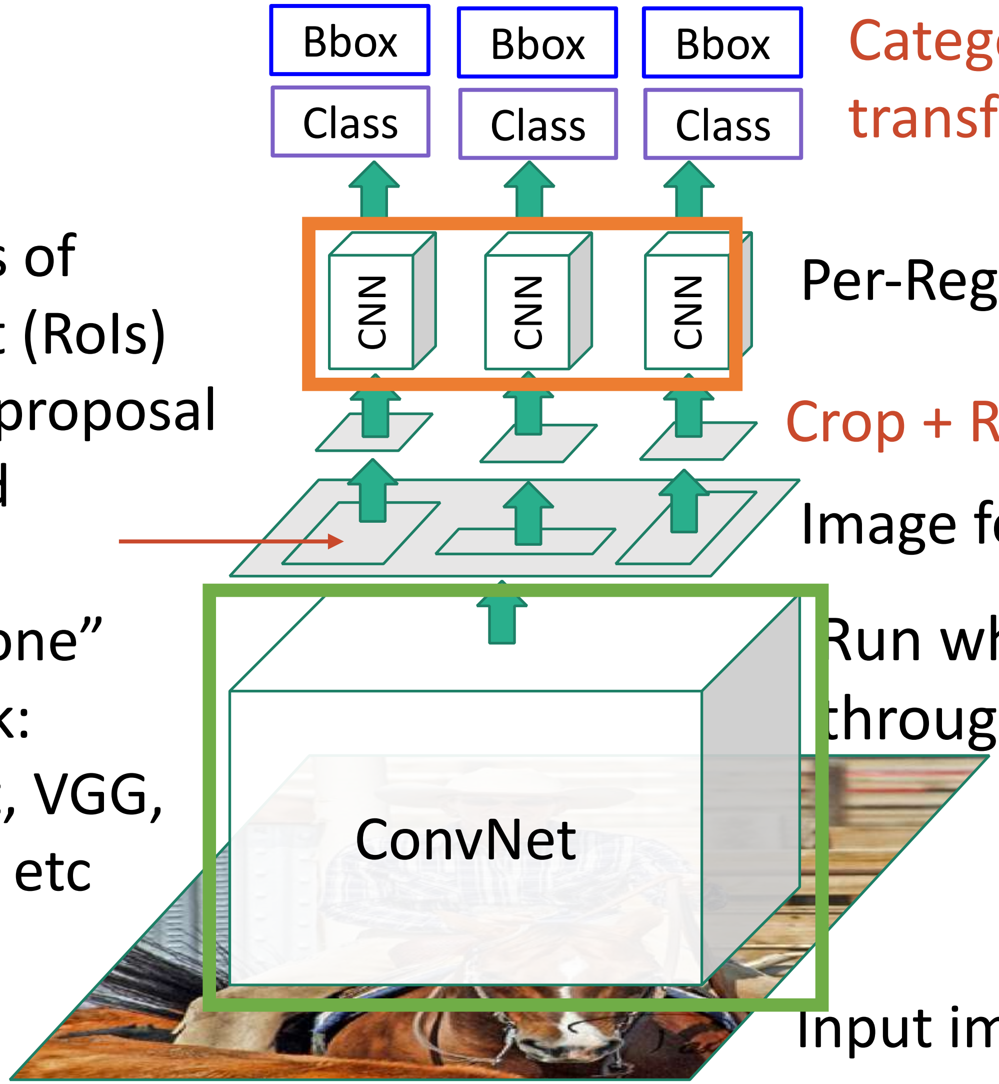
Regions of Interest (RoIs) from a proposal method

"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

# Fast R-CNN




Category and box transform per region

Regions of Interest (RoIs) from a proposal method

Per-Region Network

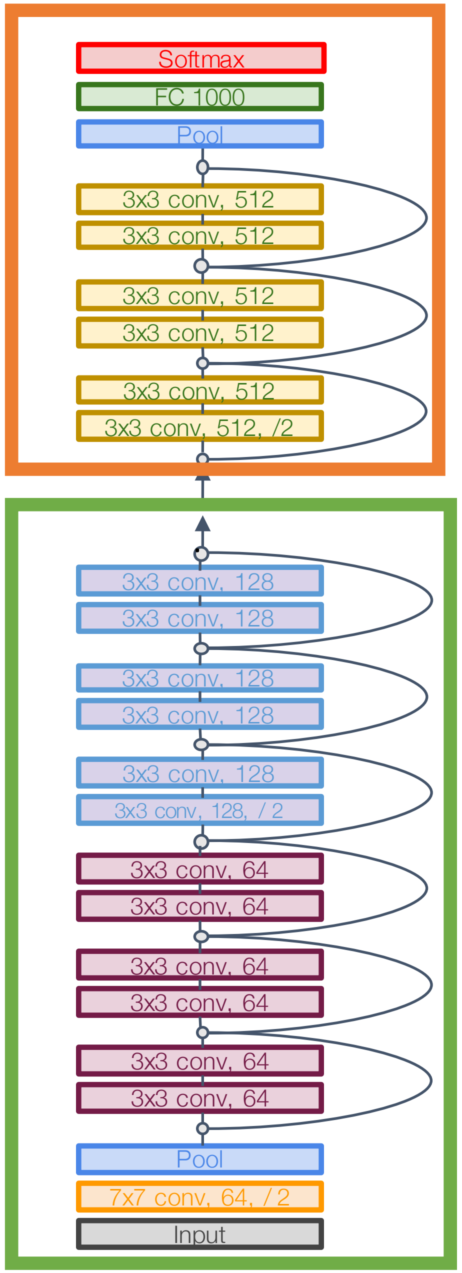
Crop + Resize features

Image features

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

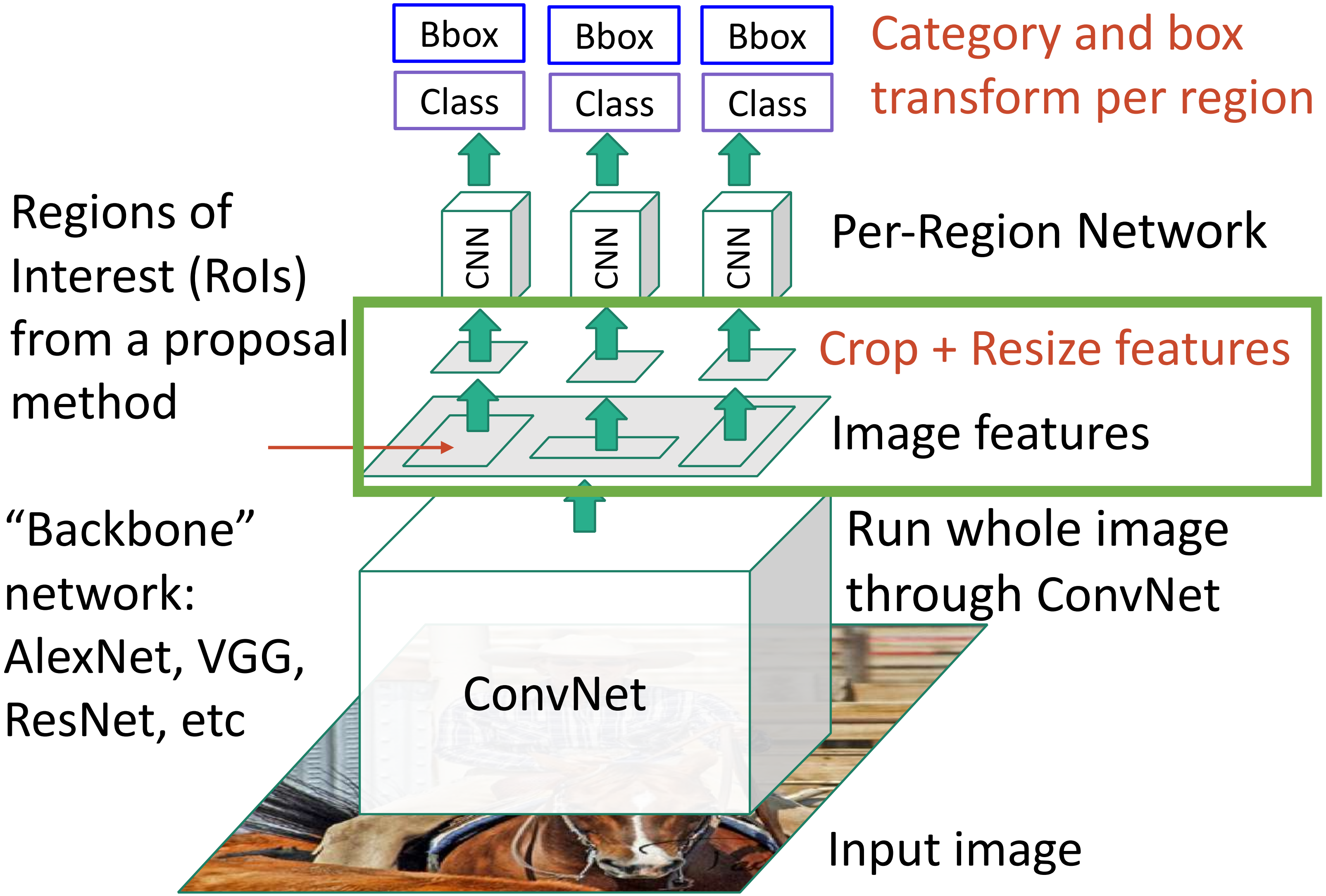
Run whole image through ConvNet

Input image




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

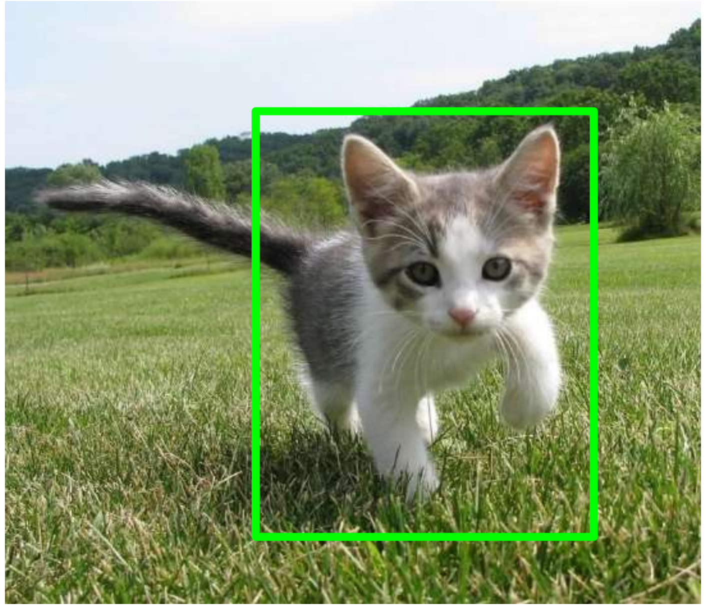
# Fast R-CNN




How to crop features?

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

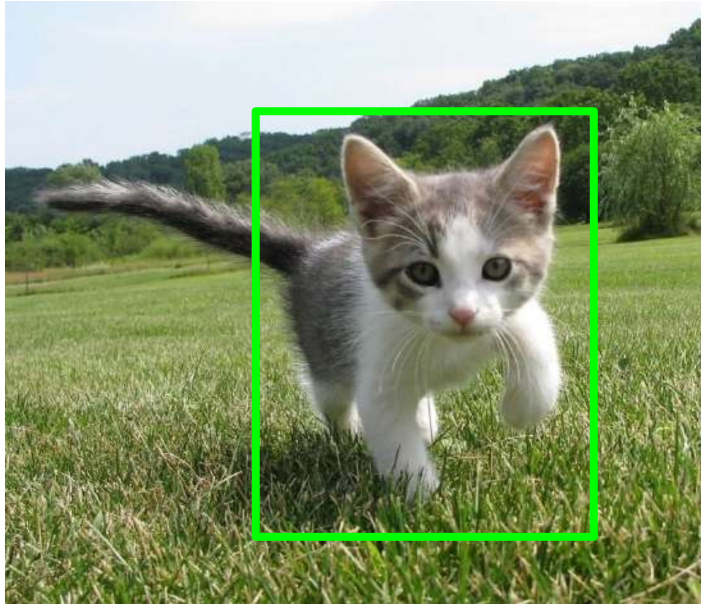
# Cropping Features: RoI Pool



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

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

# Cropping Features: RoI Pool



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

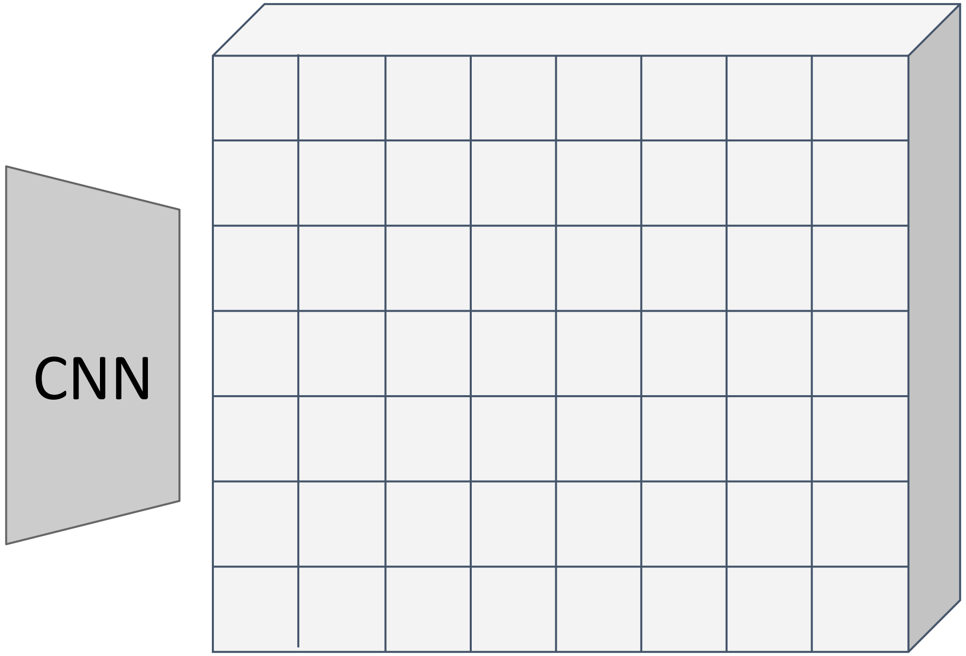
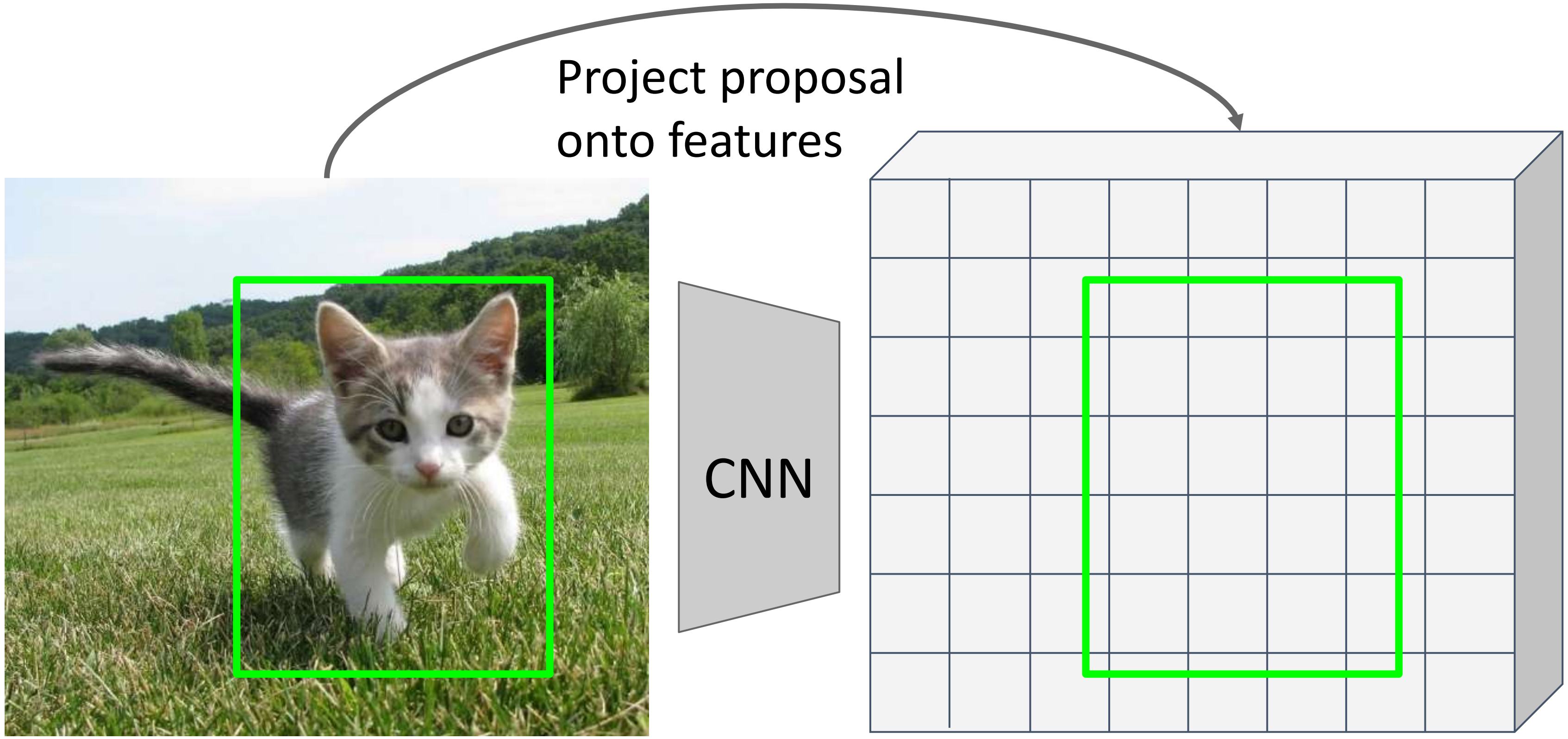



Image features
(e.g. 512 x 20 x 15)

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

# Cropping Features: RoI Pool

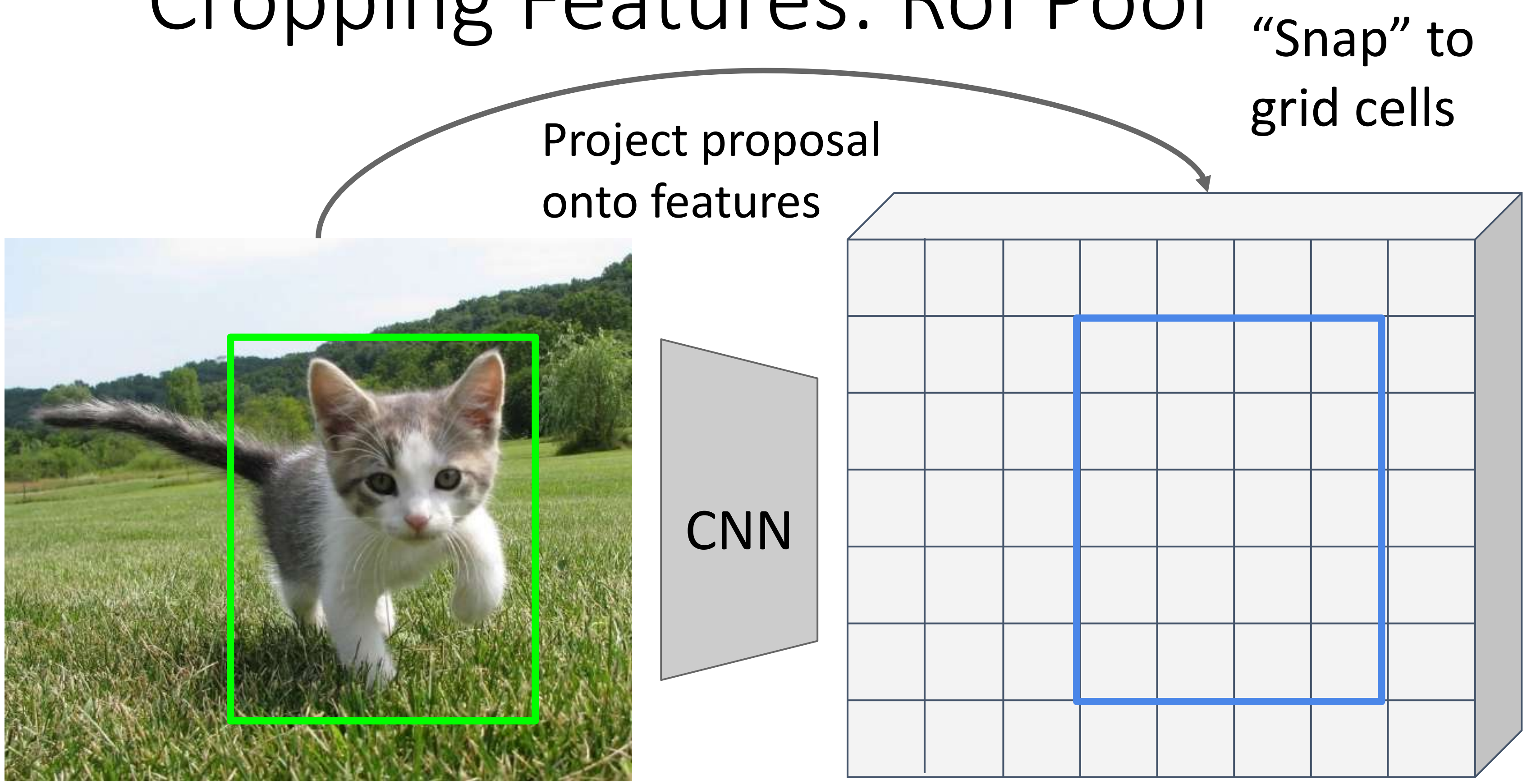



Input Image
(e.g. 3 x 640 x 480)

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

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

# Cropping Features: RoI Pool




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

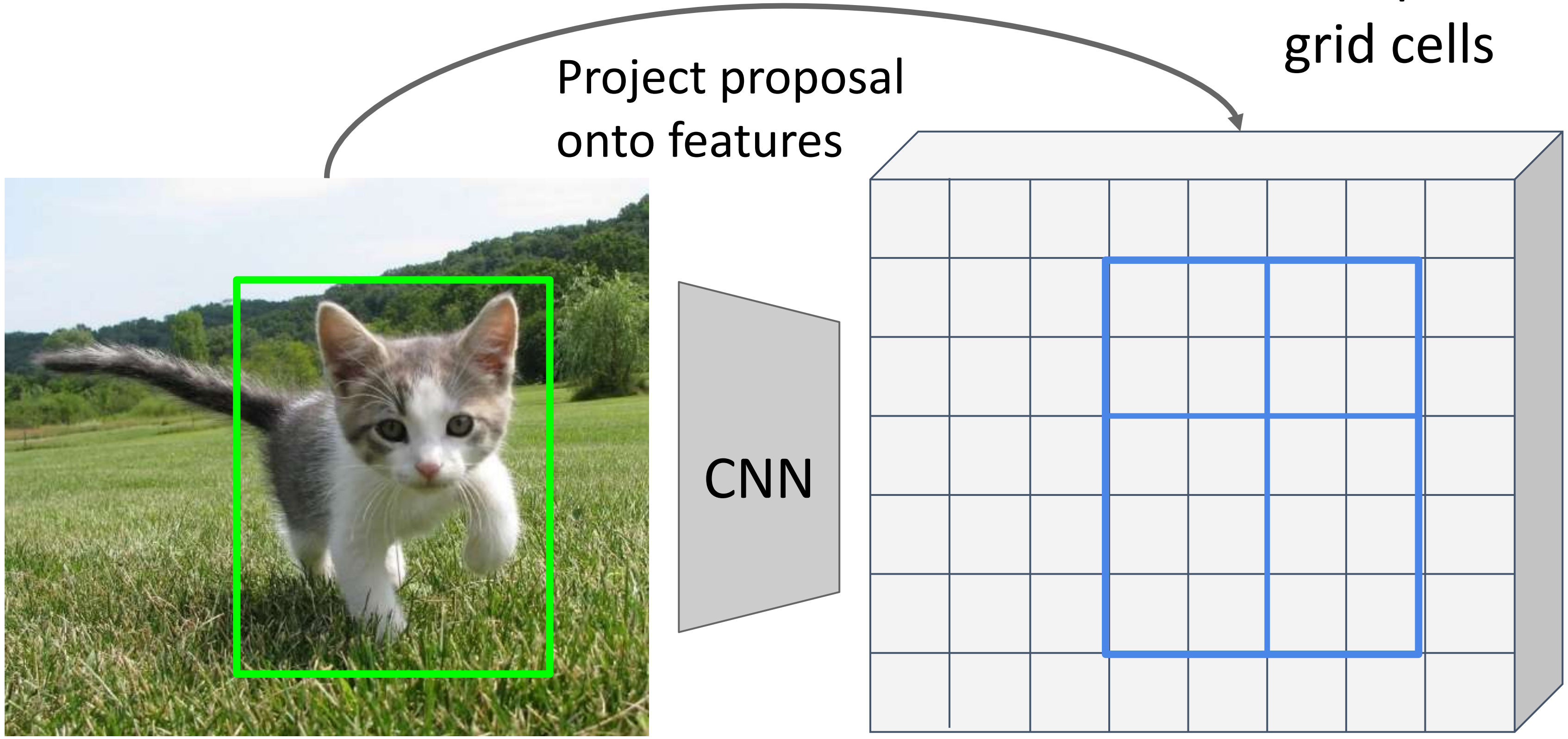
Image features
(e.g. 512 x 20 x 15)

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

# Cropping Features: RoI Pool

"Snap" to grid cells

Divide into 2x2 grid of (roughly) equal subregions




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

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

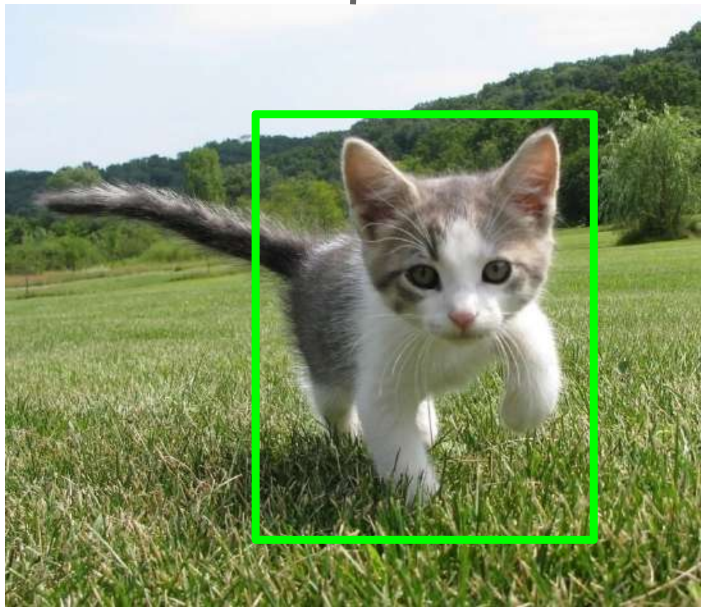
Girshick, "Fast R-CNN", ICCV 2015.

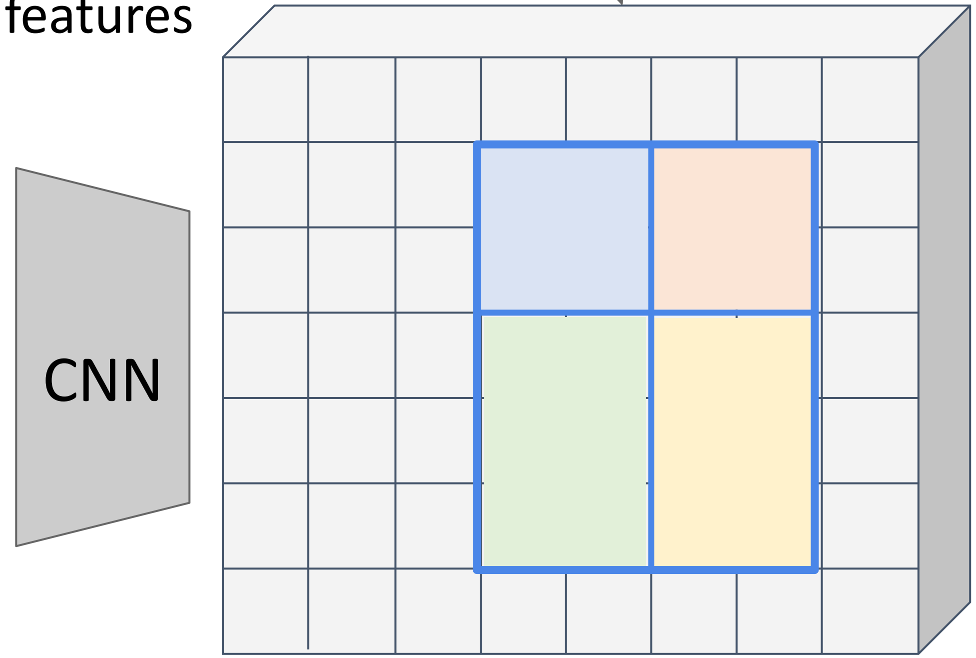
# Cropping Features: RoI Pool

Project proposal
onto features

"Snap" to
grid cells

Divide into 2x2
grid of (roughly)
equal subregions

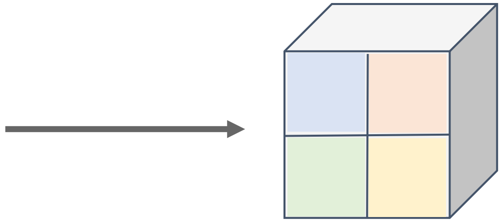





Input Image
(e.g. 3 x 640 x 480)

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

Max-pool within
each subregion



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

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

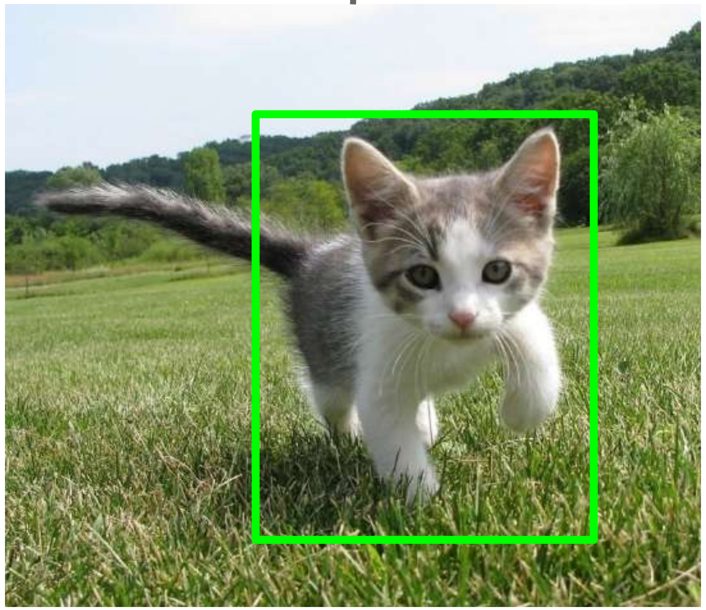
Girshick, "Fast R-CNN", ICCV 2015.

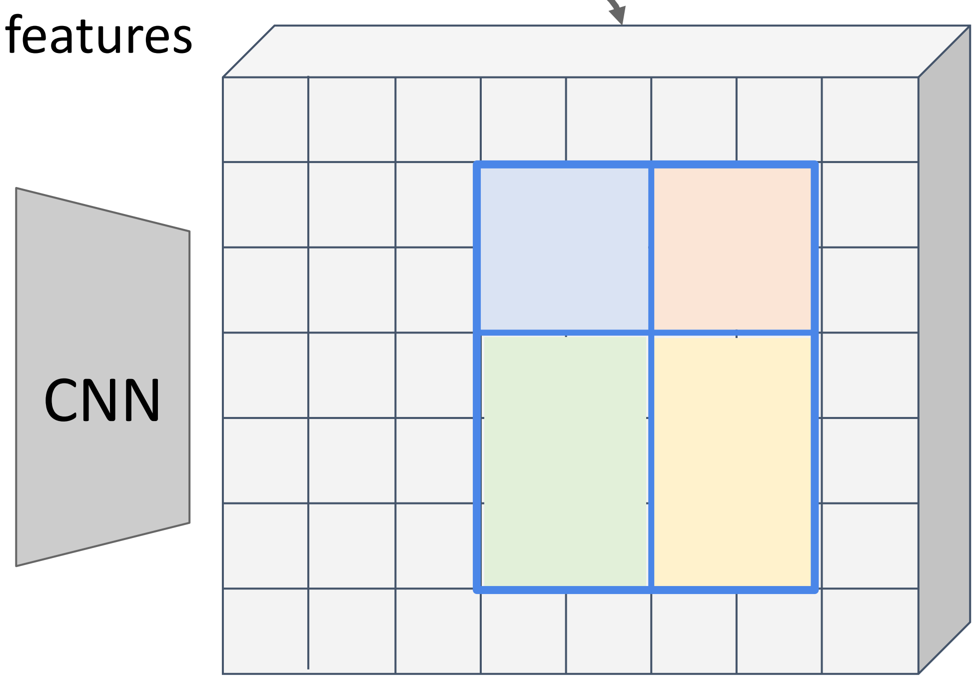
# Cropping Features: RoI Pool

"Snap" to grid cells

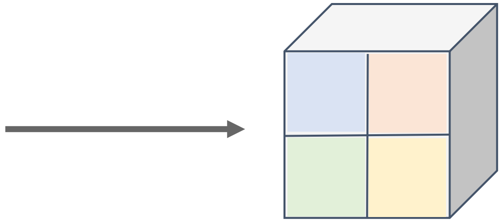
Divide into 2x2 grid of (roughly) equal subregions

Project proposal onto features






Max-pool within each subregion



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

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

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

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

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

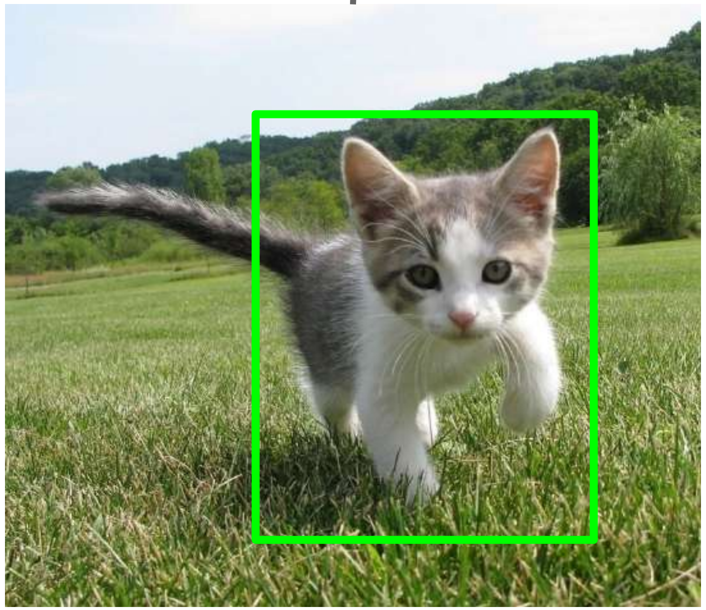
# Cropping Features: RoI Pool

Project proposal onto features

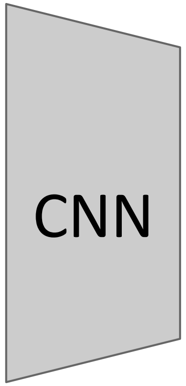
"Snap" to grid cells

Divide into 2x2 grid of (roughly) equal subregions

Max-pool within each subregion



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



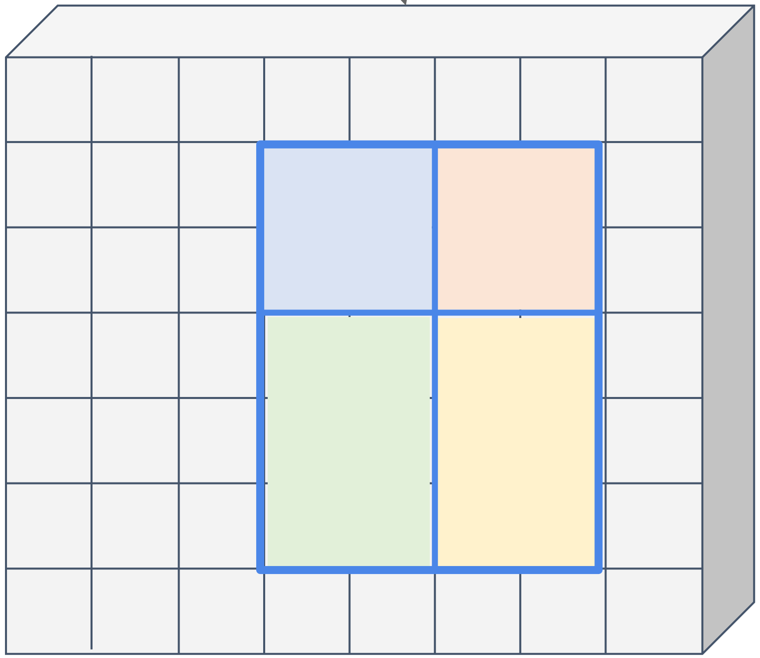
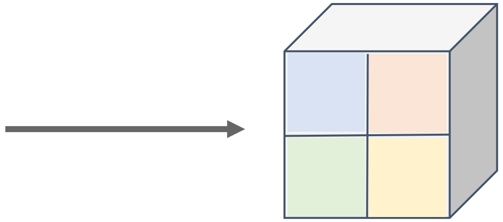



Image features
(e.g. 512 x 20 x 15)



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

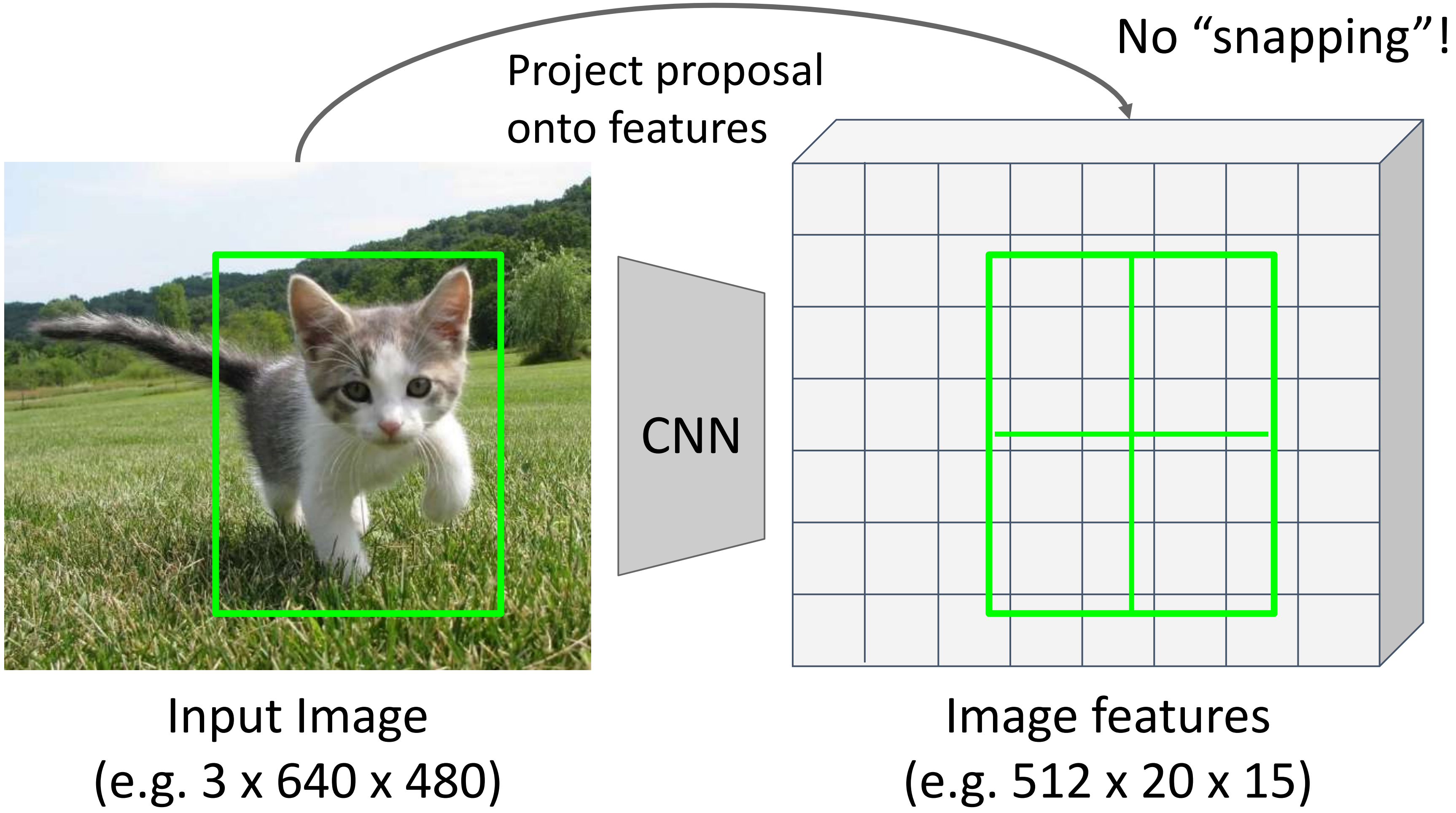
Region features always the same size even if input regions have different sizes!

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!)



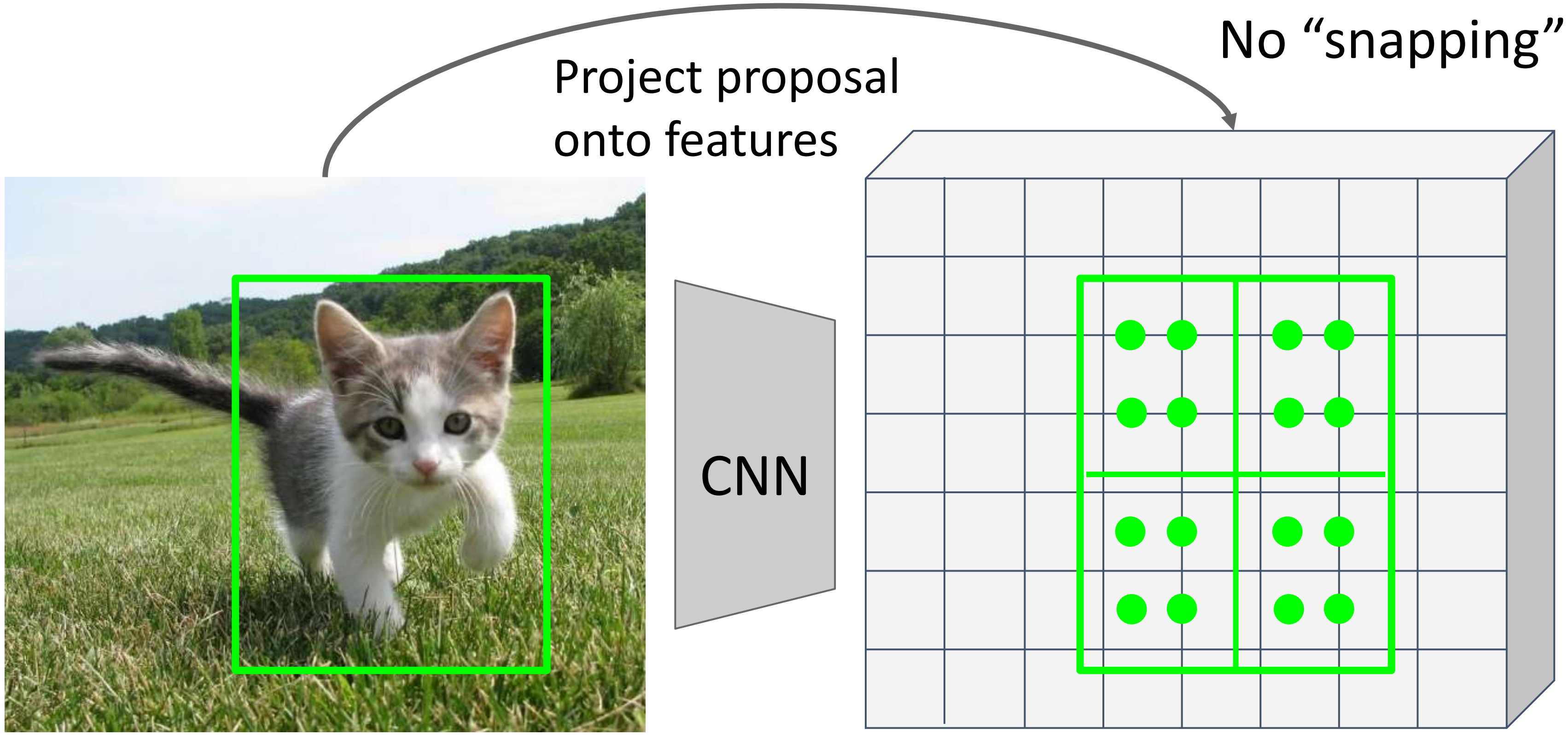

He et al, “Mask R-CNN”, ICCV 2017

# Cropping Features: RoI Align

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

No “snapping”!

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



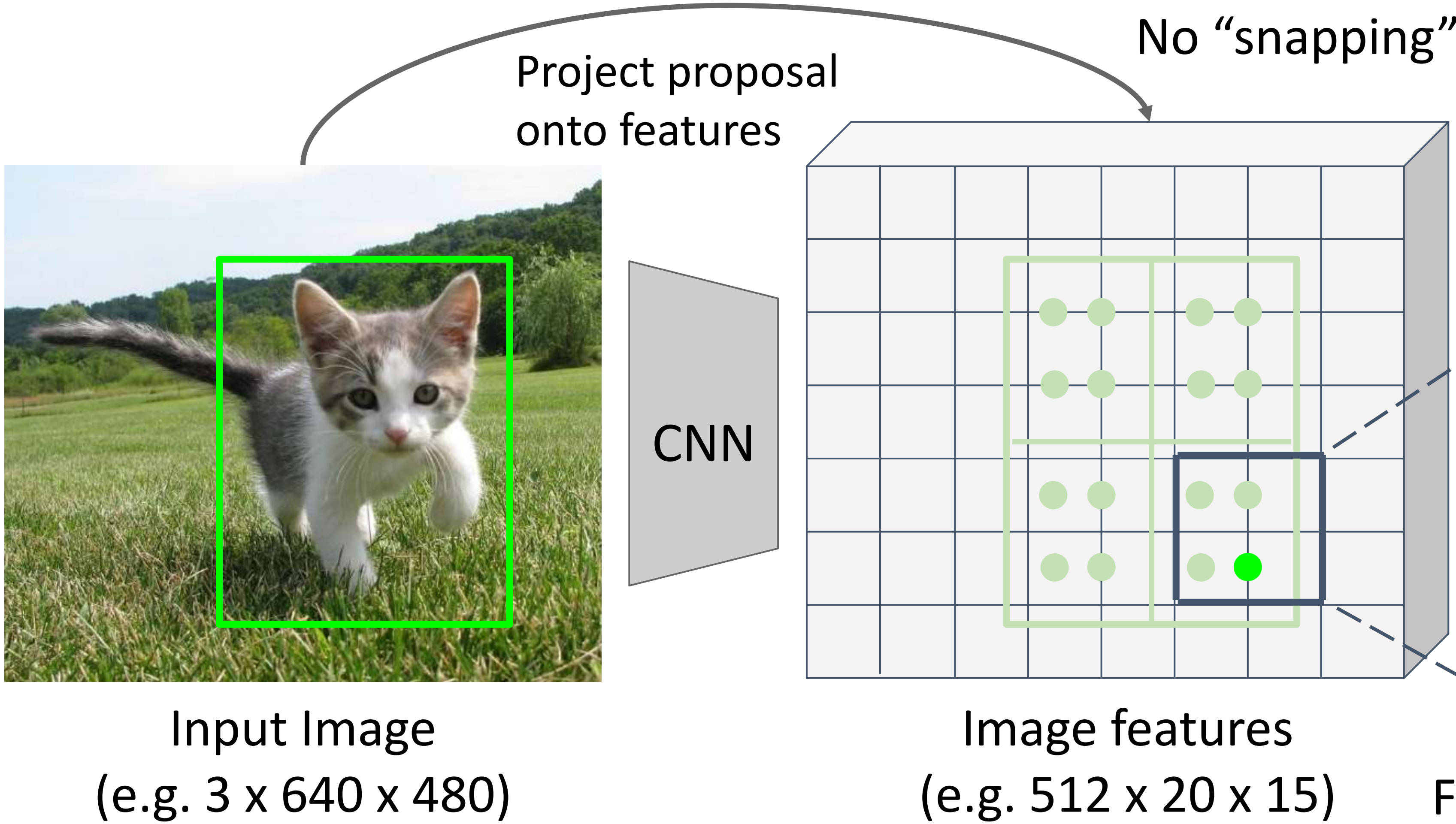

Input Image
(e.g. 3 x 640 x 480)

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

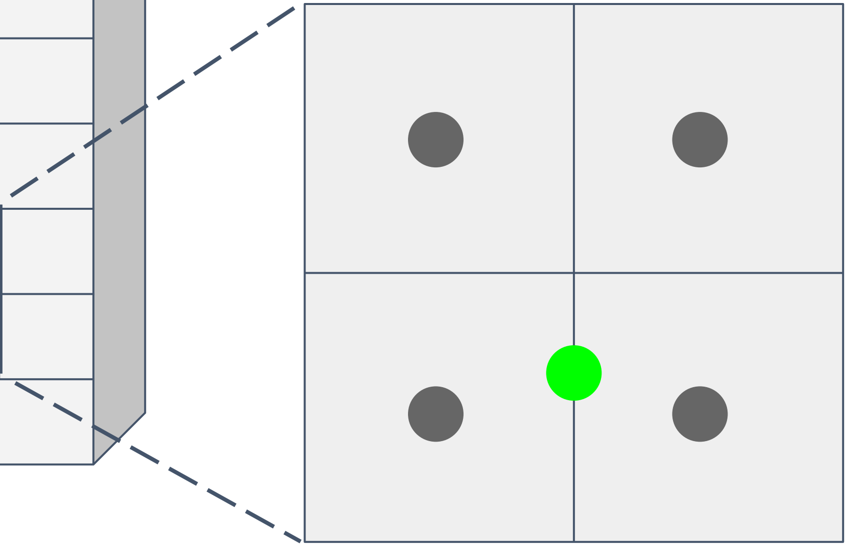
He et al, “Mask R-CNN”, ICCV 2017

# Cropping Features: RoI Align

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



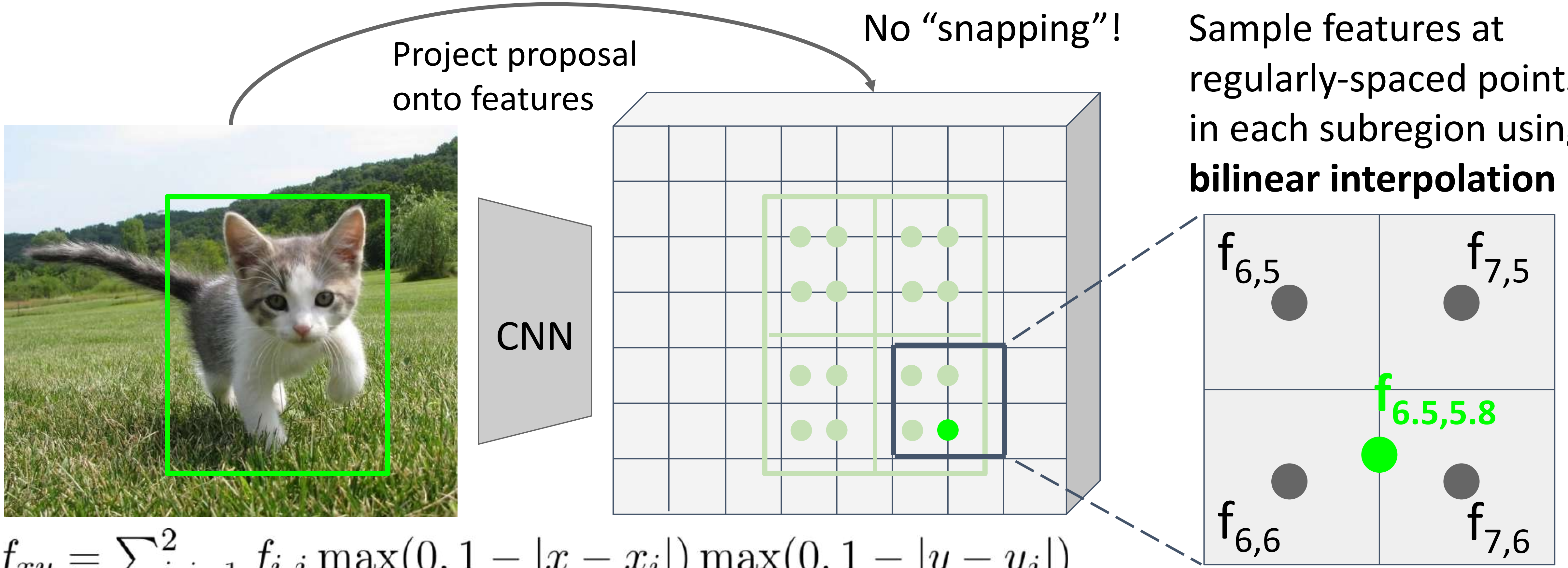

Sample features at regularly-spaced points in each subregion using **bilinear interpolation**



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!)



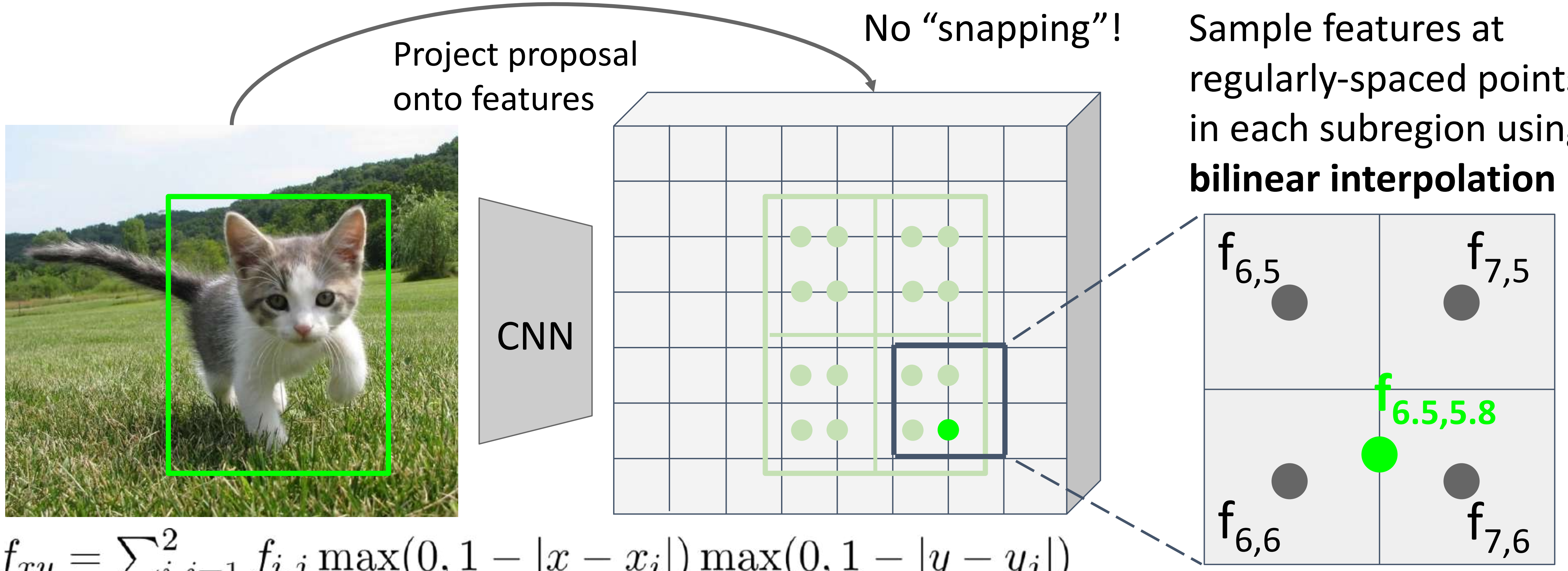

Sample features at regularly-spaced points in each subregion using **bilinear interpolation**

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

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!)




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

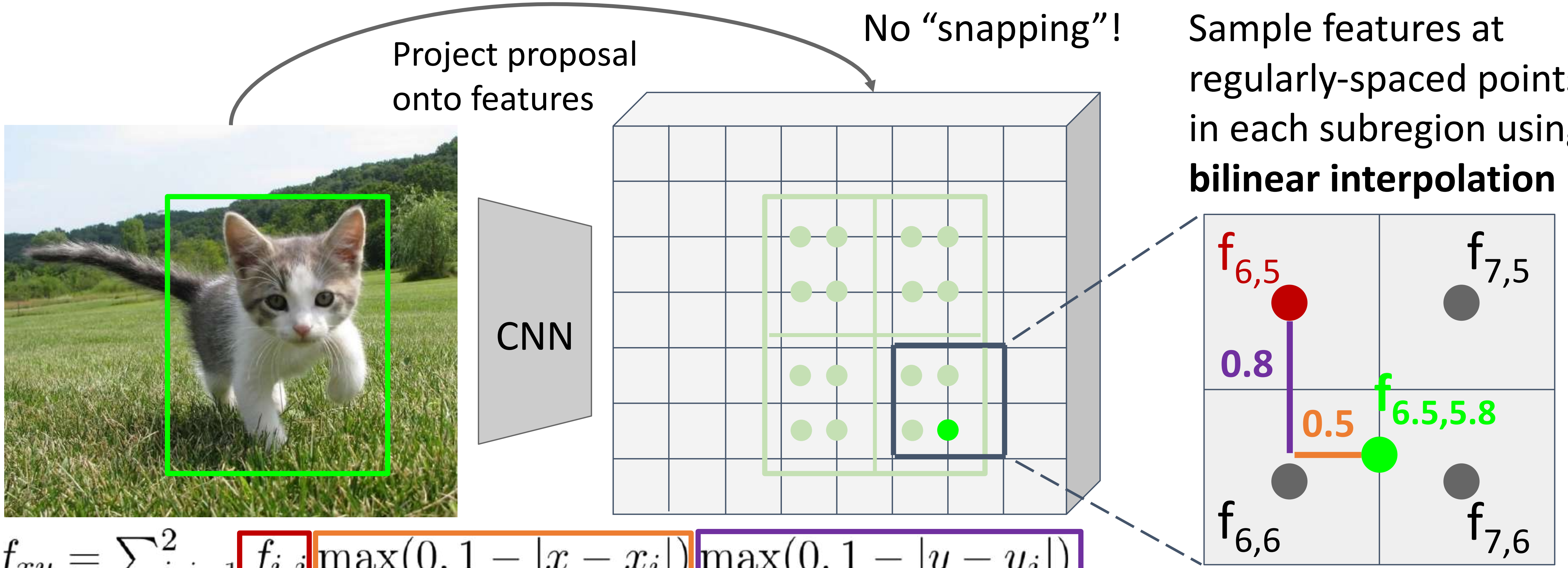
$$f_{xy} = \sum_{i,j=1}^{2} f_{i,j} \max(0, 1 - |x - x_i|) \max(0, 1 - |y - y_j|)$$

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

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

# Cropping Features: RoI Align

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




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

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

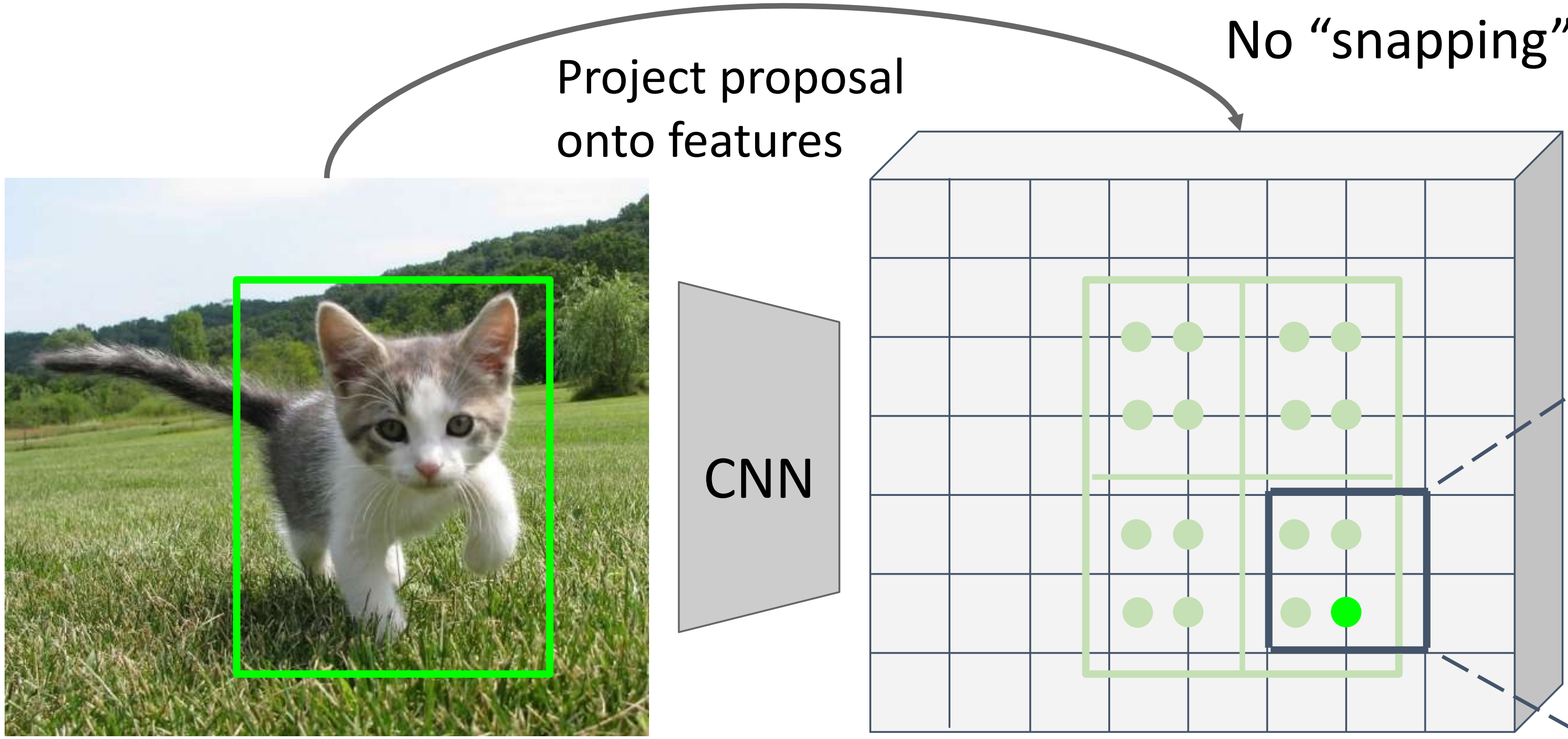
$f_{6.5,5.8} = (f_{6,5} * 0.5 * 0.2) + (f_{7,5} * 0.5 * 0.2)$
$+ (f_{6,6} * 0.5 * 0.8) + (f_{7,6} * 0.5 * 0.8)$

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

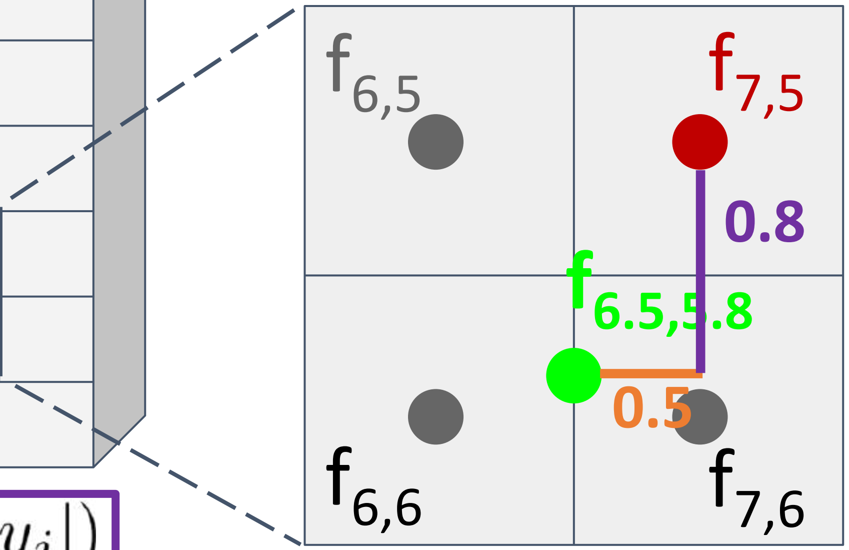
# Cropping Features: RoI Align

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

No “snapping”!




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




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

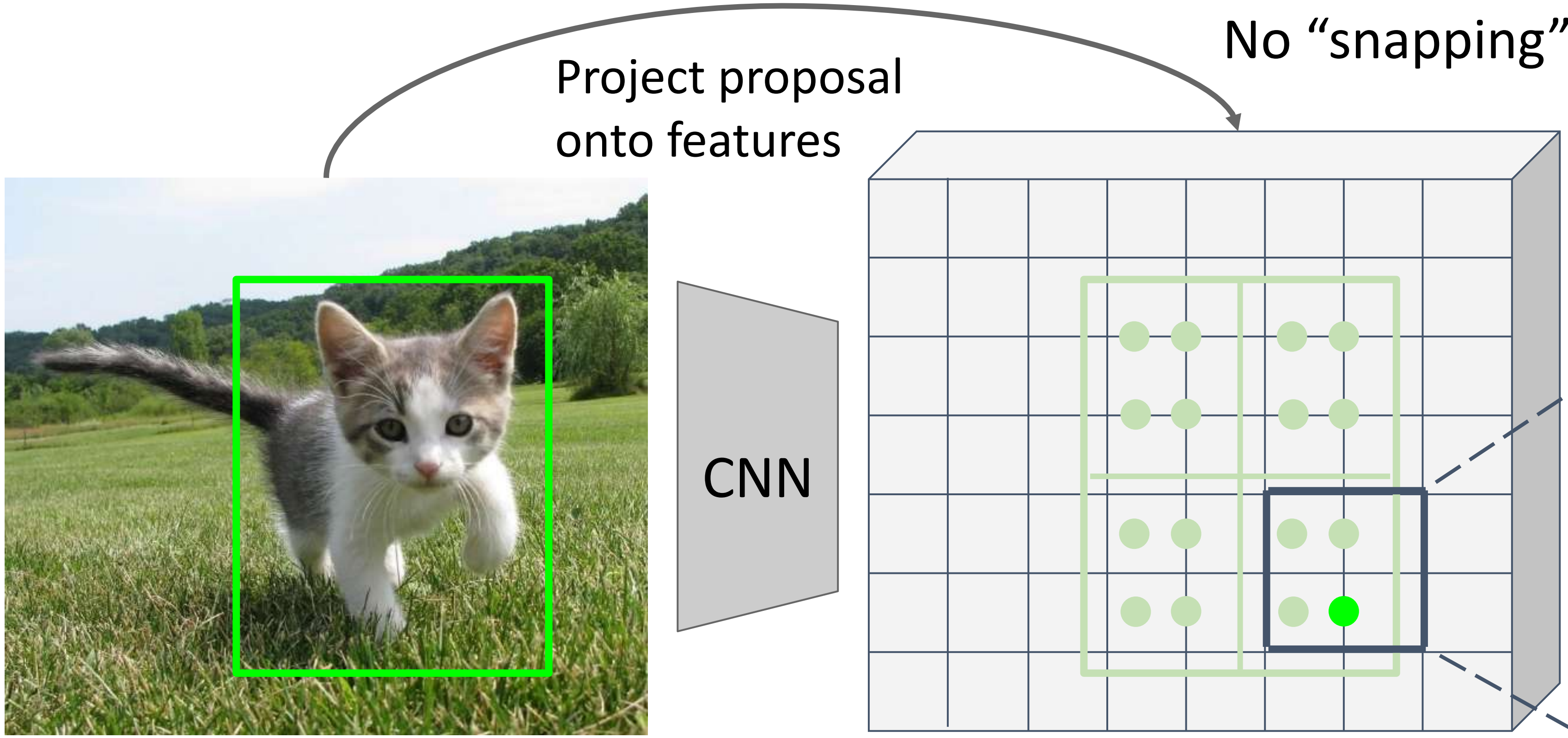
$$f_{6.5,5.8} = (f_{6,5} * 0.5 * 0.2) + (f_{7,5} * 0.5 * 0.2) + (f_{6,6} * 0.5 * 0.8) + (f_{7,6} * 0.5 * 0.8)$$

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

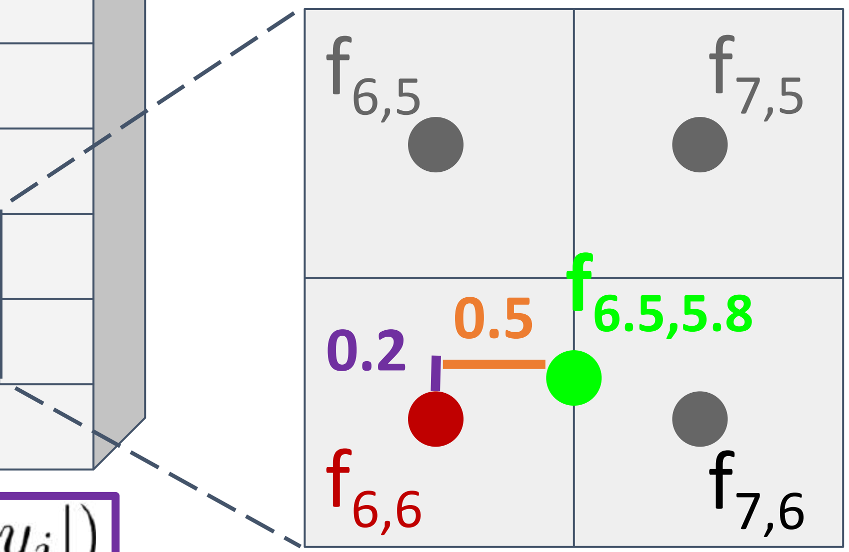
# Cropping Features: RoI Align

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

No "snapping"!




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



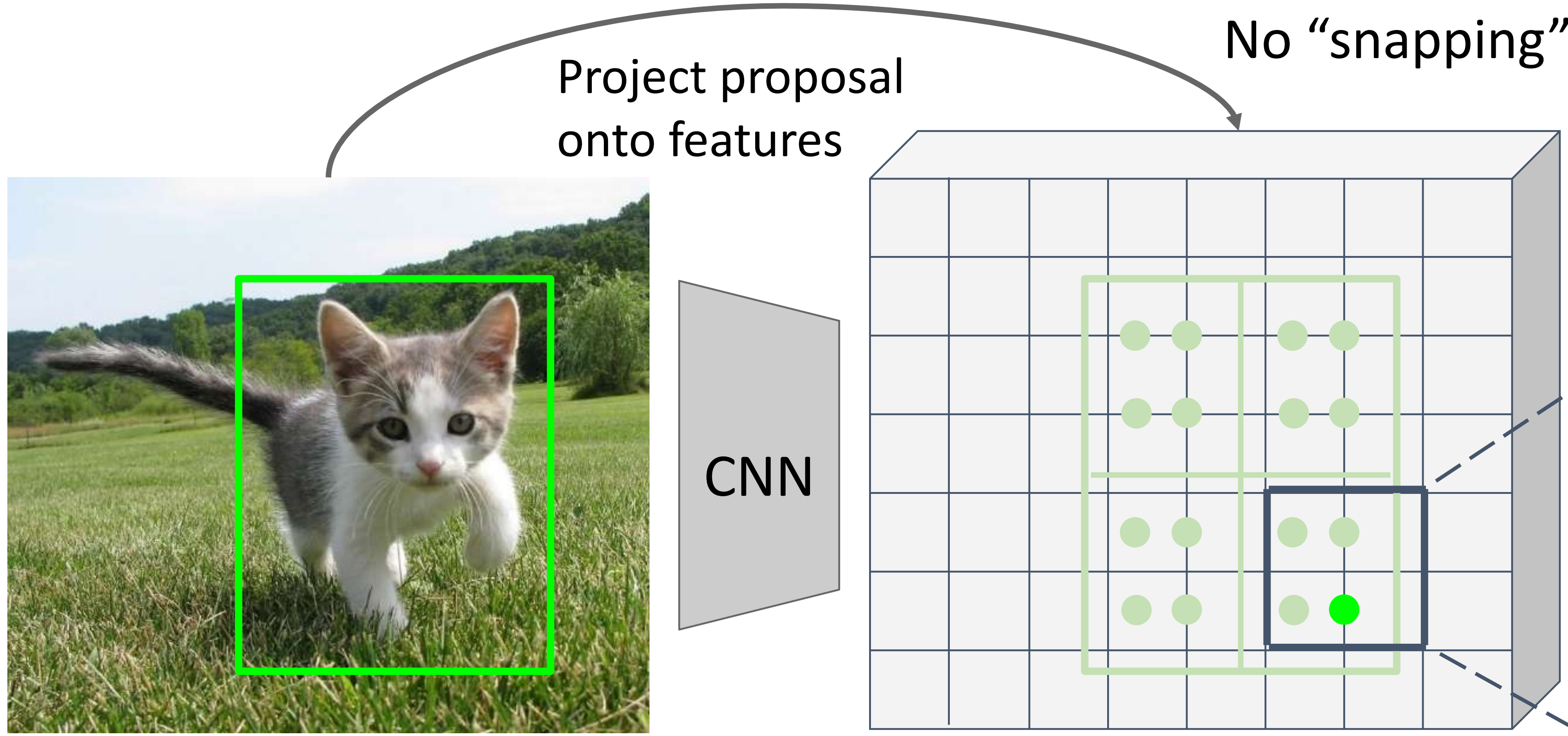

$$f_{xy} = \sum_{i,j=1}^{2} \boxed{f_{i,j}} \boxed{\max(0, 1 - |x - x_i|)} \boxed{\max(0, 1 - |y - y_j|)}$$

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

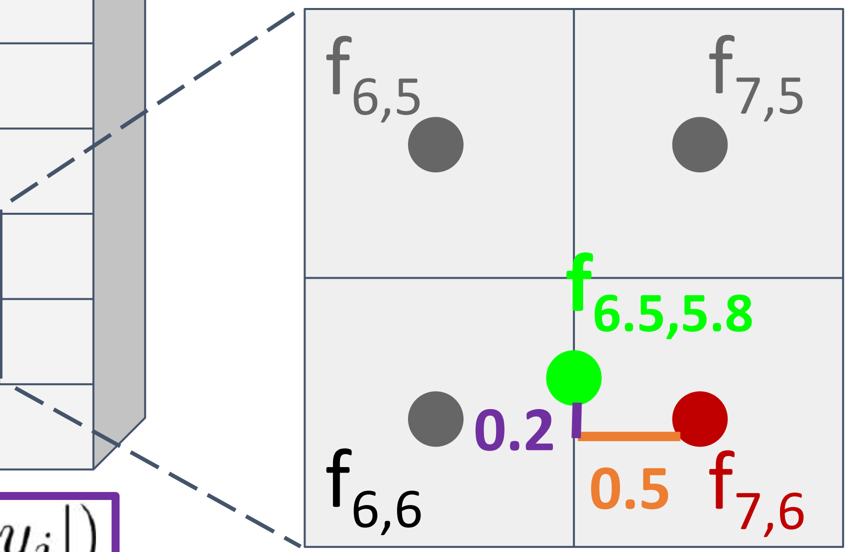
Feature $f_{xy}$ for point (x, y) is a linear combination of features at its four neighboring grid cells:

# Cropping Features: RoI Align

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




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



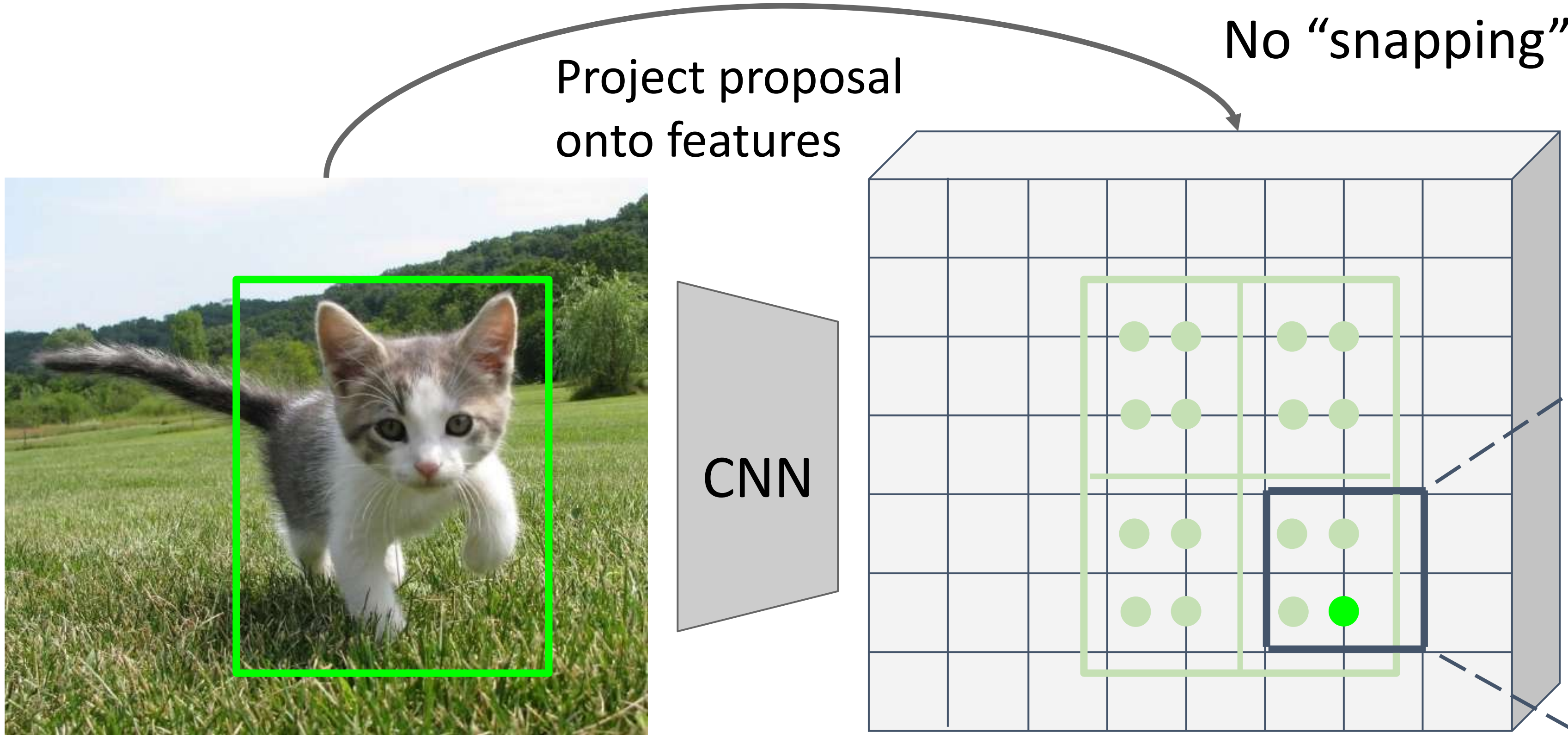

$$f_{xy} = \sum_{i,j=1}^{2} \boxed{f_{i,j}} \boxed{\max(0, 1-|x-x_i|)} \boxed{\max(0, 1-|y-y_j|)}$$

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

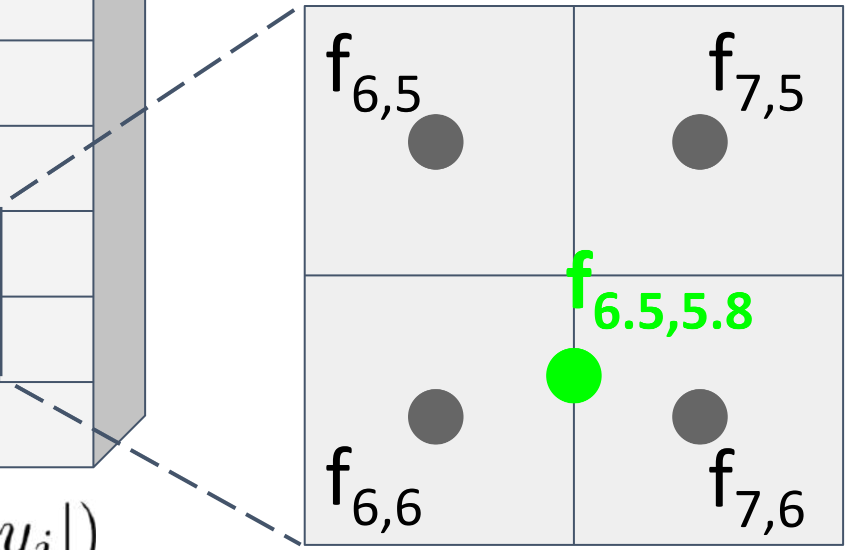
Feature $f_{xy}$ for point $(x, y)$ is a linear combination of features at its four neighboring grid cells:

# Cropping Features: RoI Align

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




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



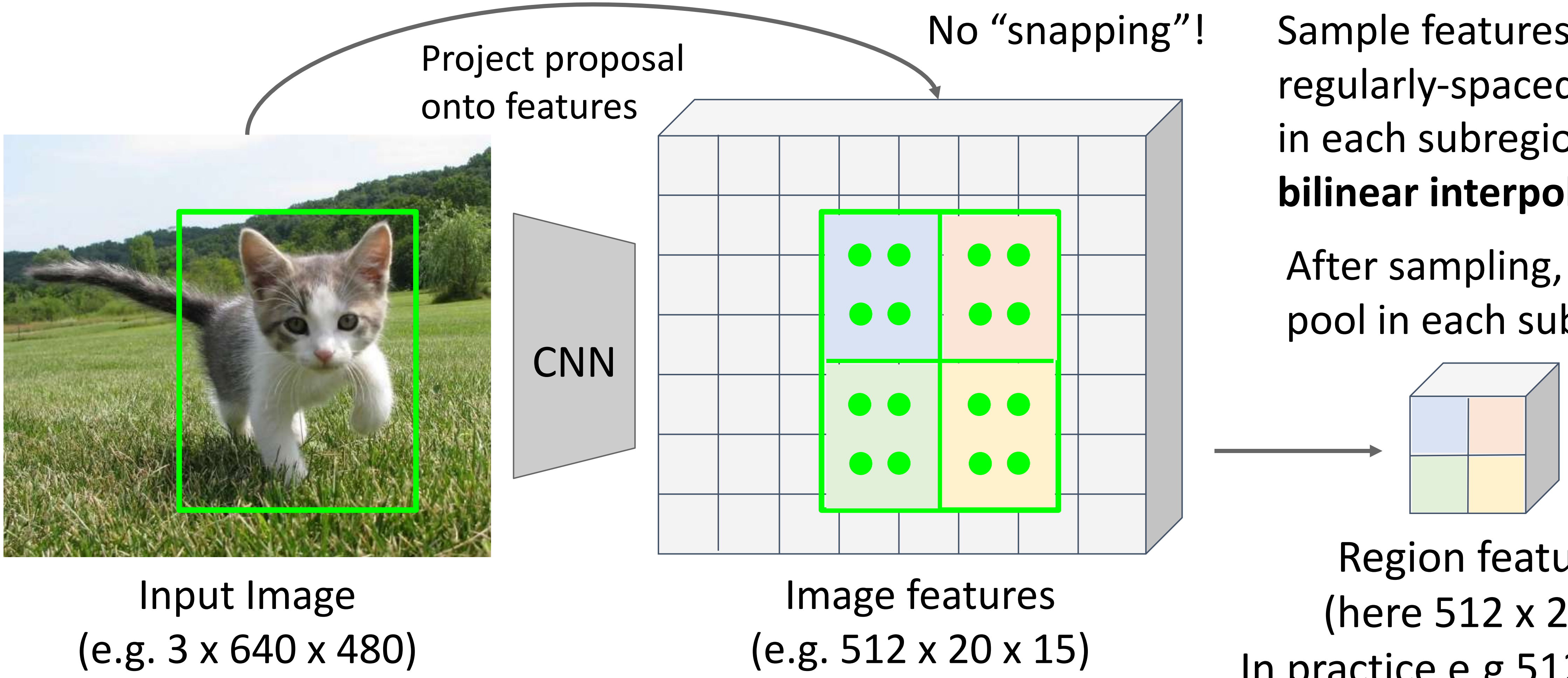

$$f_{xy} = \sum_{i,j=1}^{2} f_{i,j} \max(0, 1 - |x - x_i|) \max(0, 1 - |y - y_j|)$$

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

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

# Cropping Features: RoI Align

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



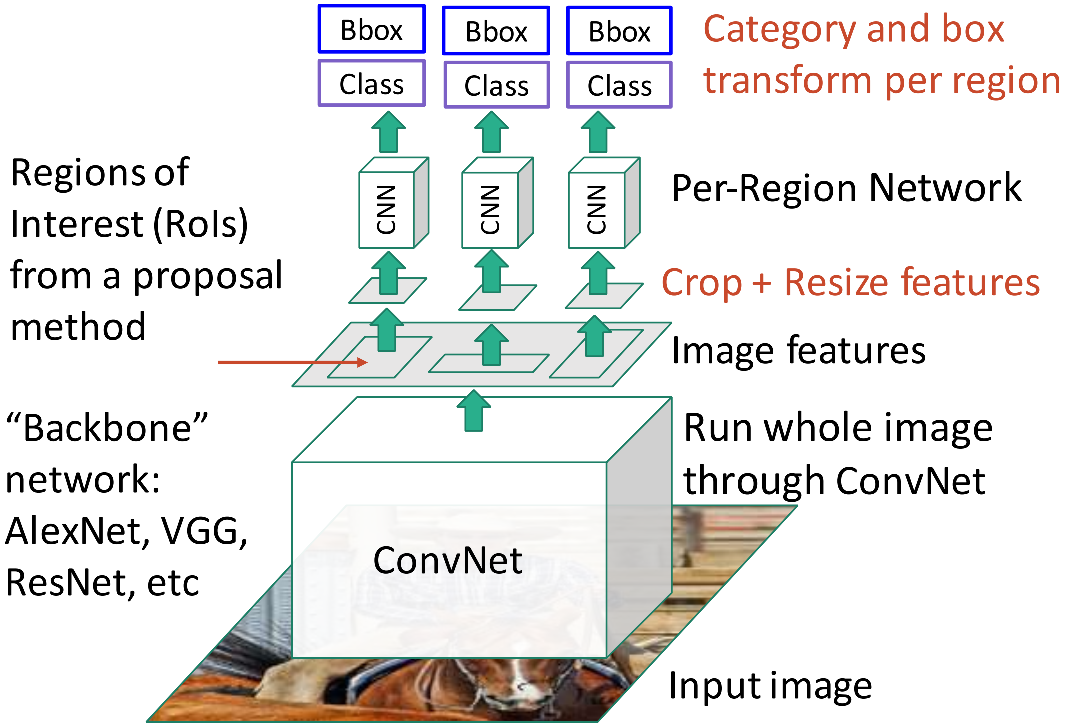

Sample features at regularly-spaced points in each subregion using **bilinear interpolation**

After sampling, max-pool in each subregion

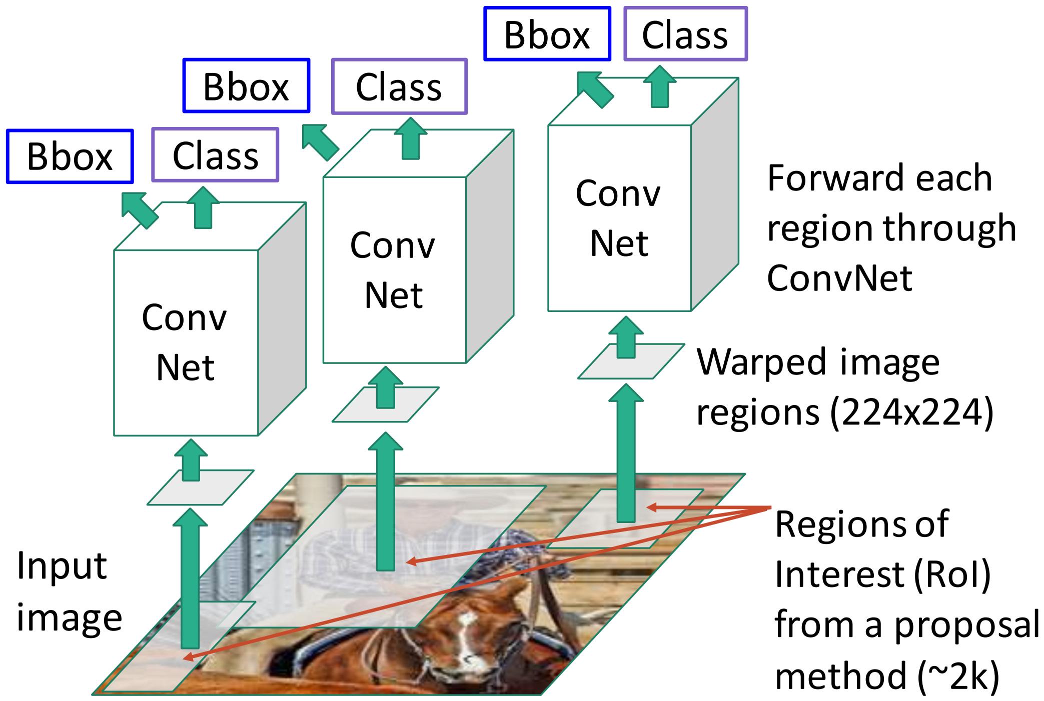
He et al, "Mask R-CNN", ICCV 2017

# Fast R-CNN vs “Slow” R-CNN

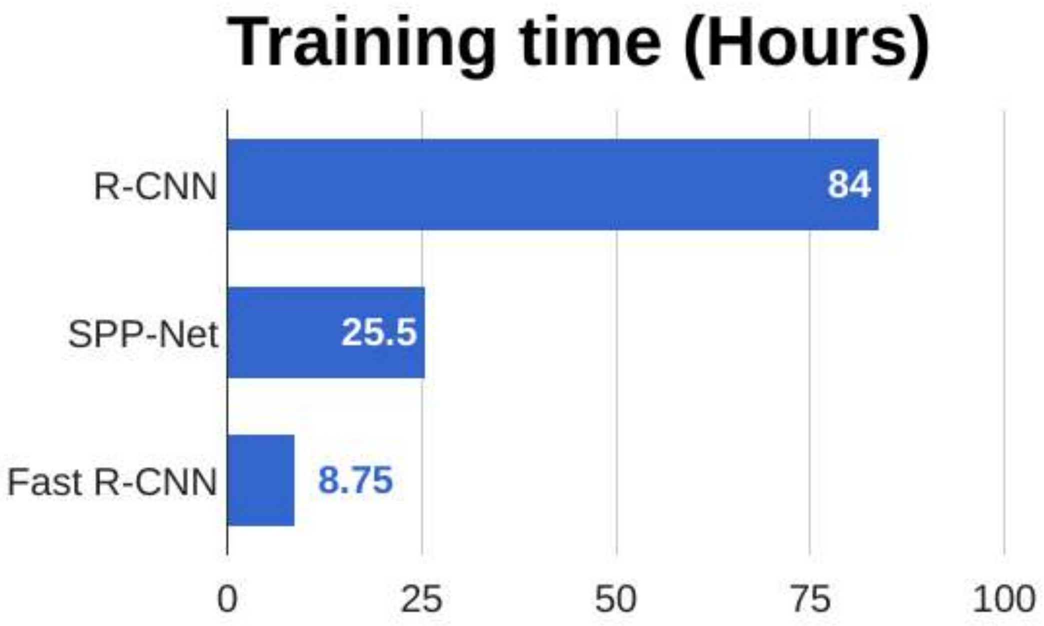
**Fast R-CNN**: Apply differentiable cropping to shared image features

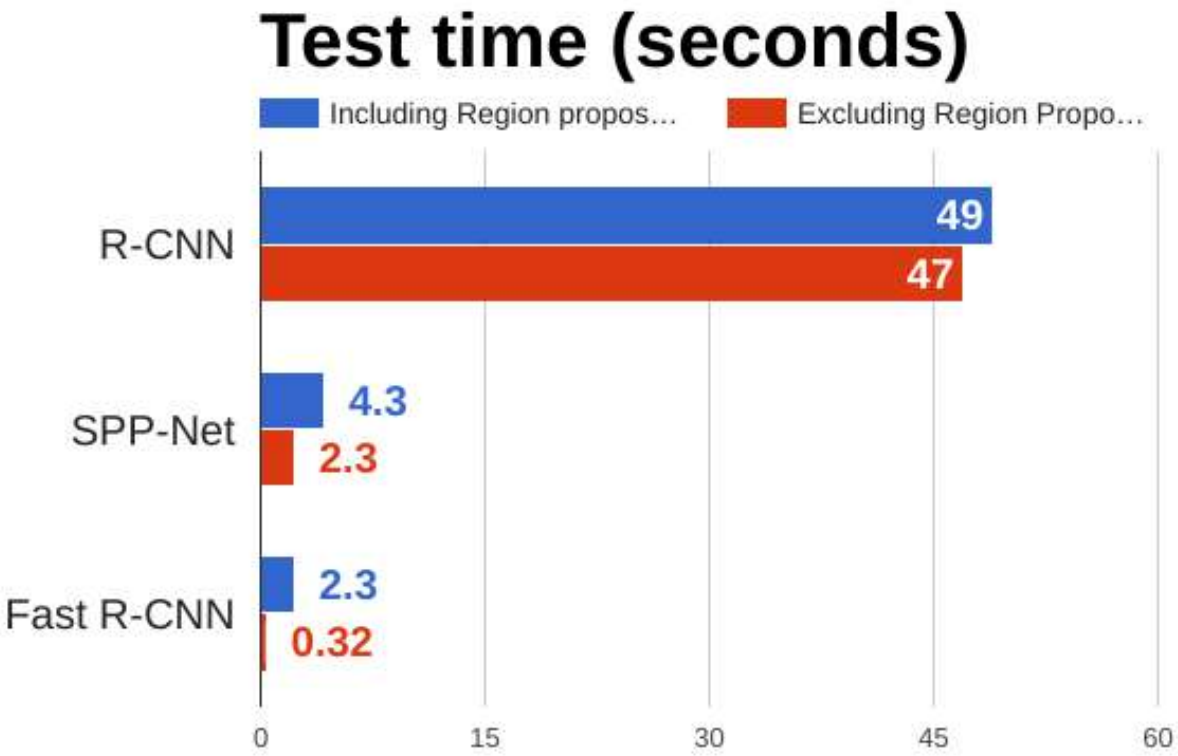



**“Slow” R-CNN**: Apply differentiable cropping to shared image features



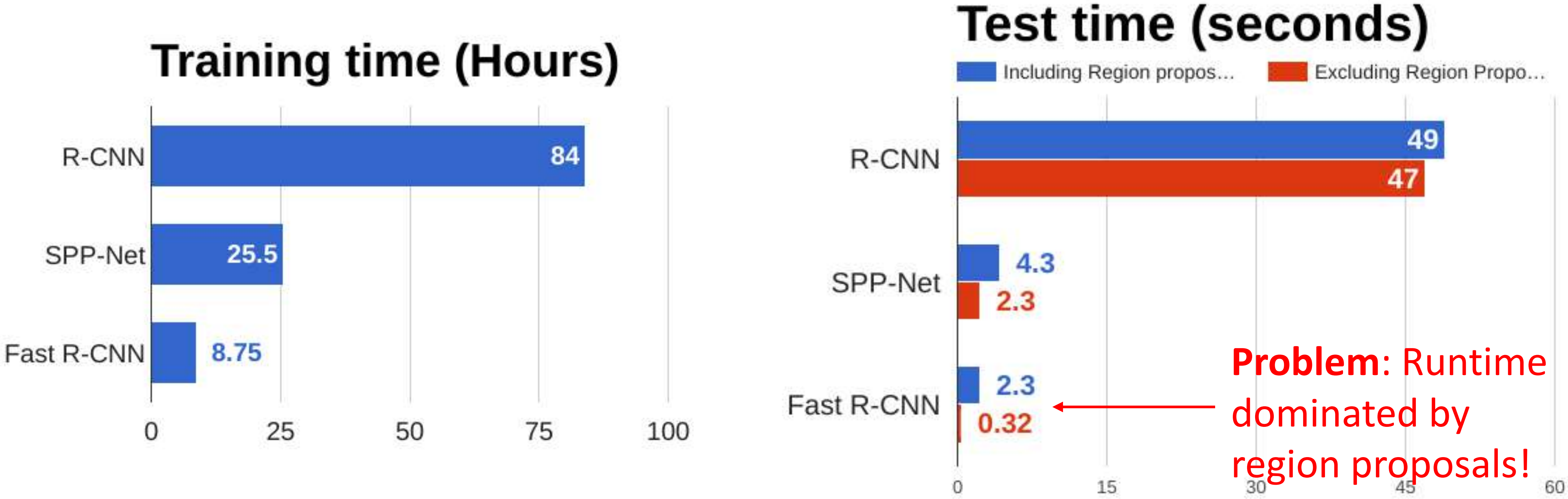
# Fast R-CNN vs "Slow" R-CNN




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

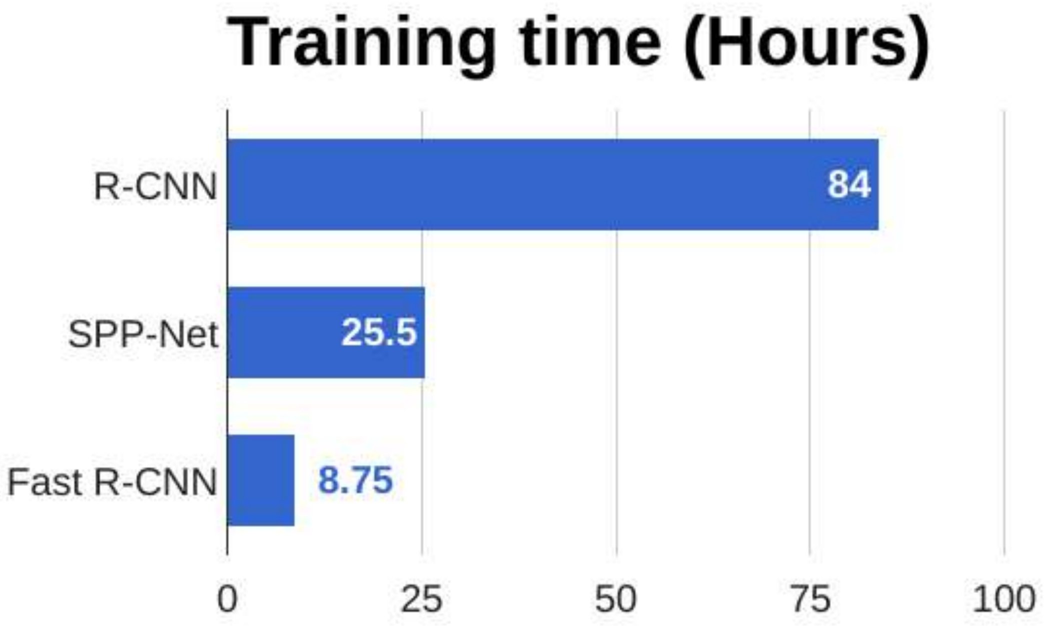
# Fast R-CNN vs “Slow” R-CNN

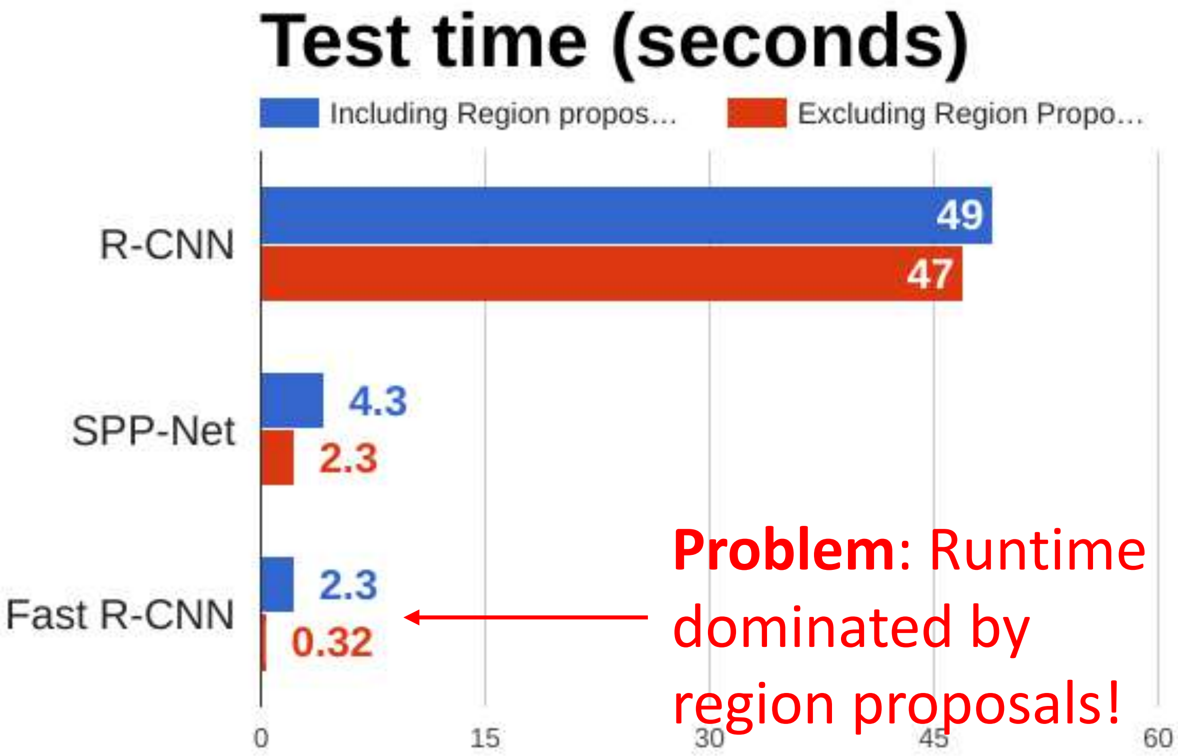



**Problem**: Runtime dominated by region proposals!

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




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

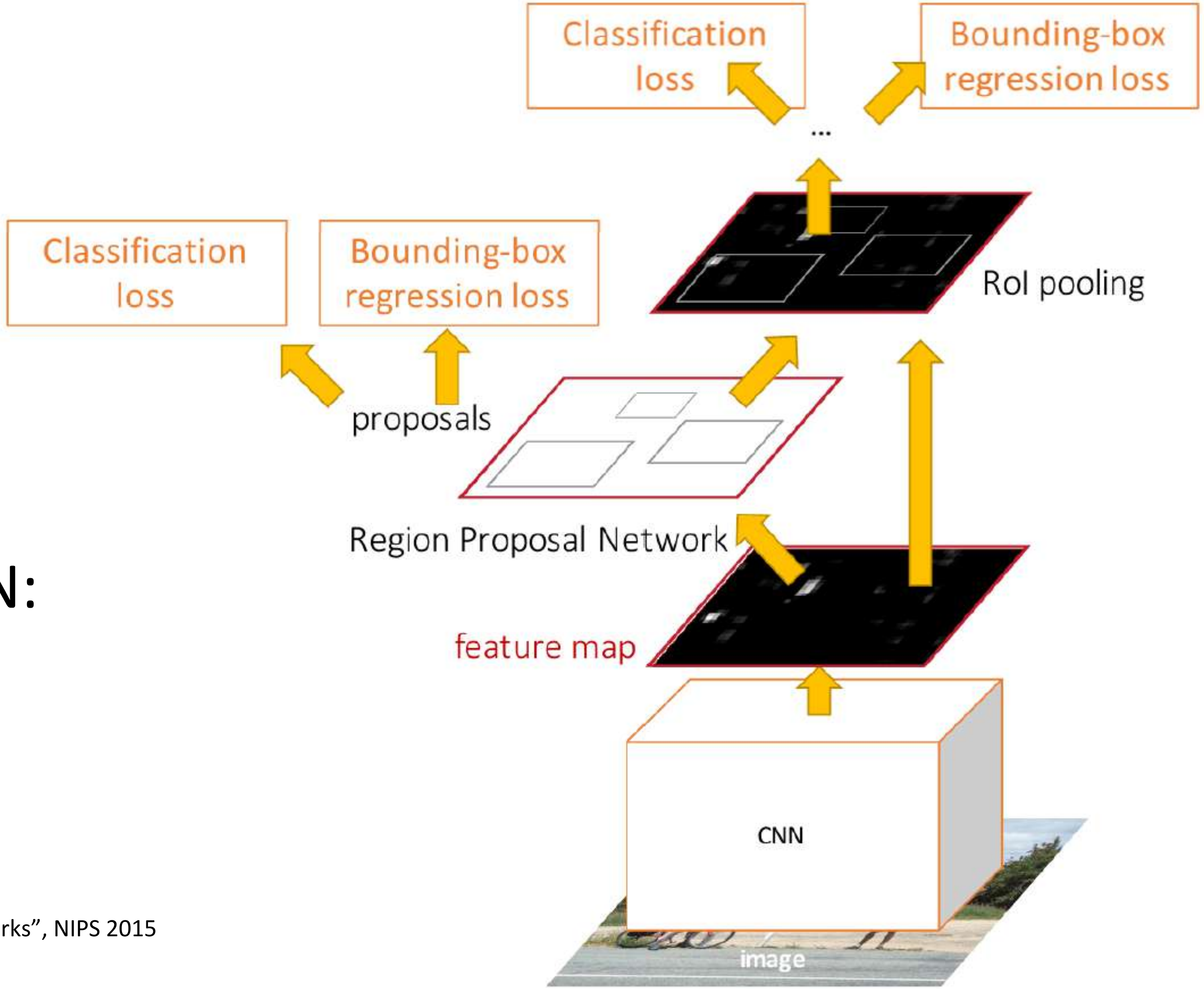
**Recall**: Region proposals computed by heuristic ”Selective Search” algorithm on CPU -- let’s learn them with a CNN instead!


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

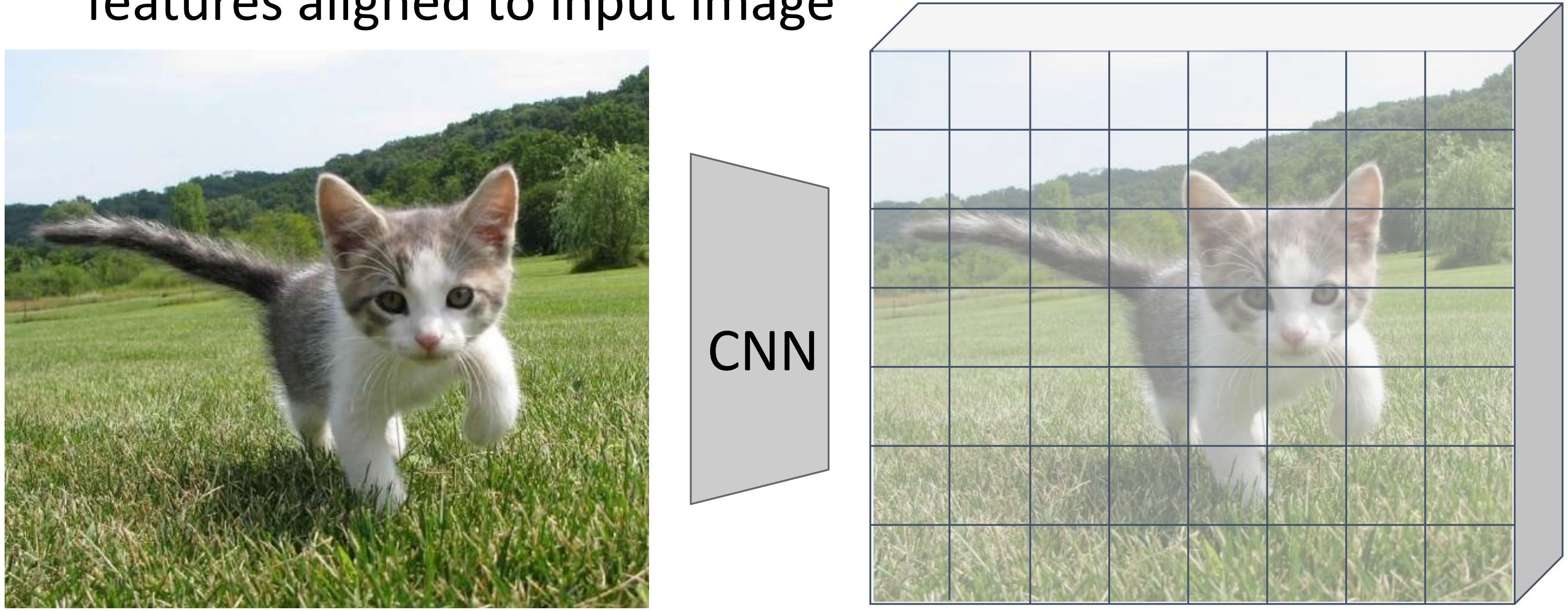
Otherwise same as Fast R-CNN: Crop features for each proposal, classify each one




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

# 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)

# 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

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

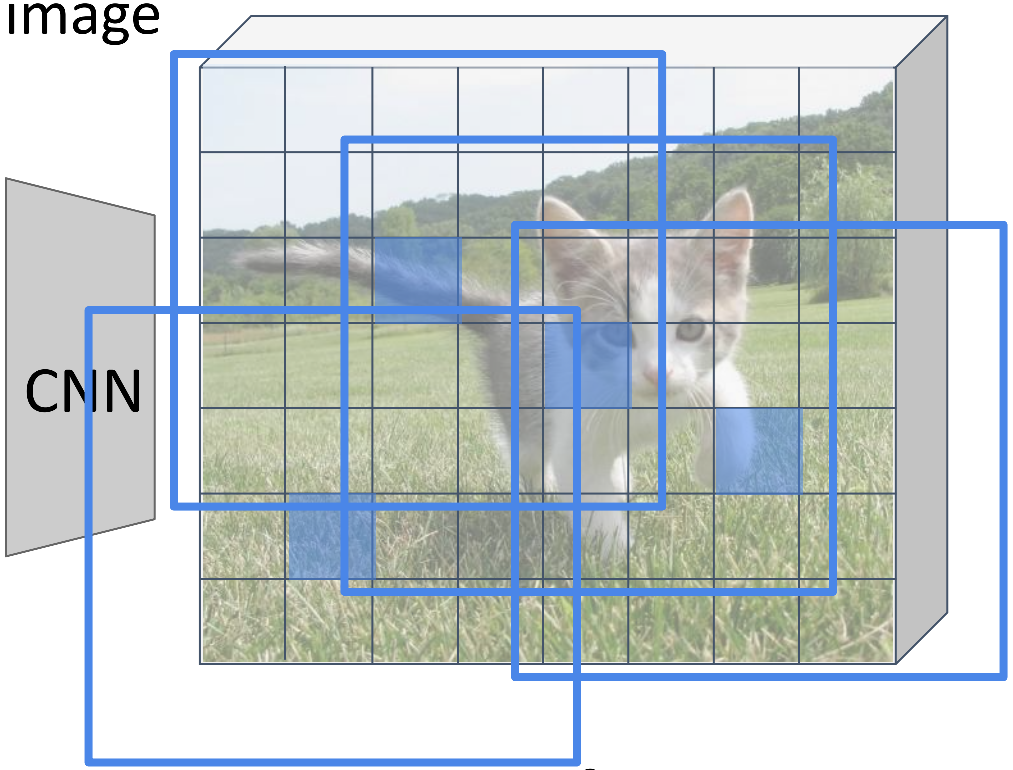
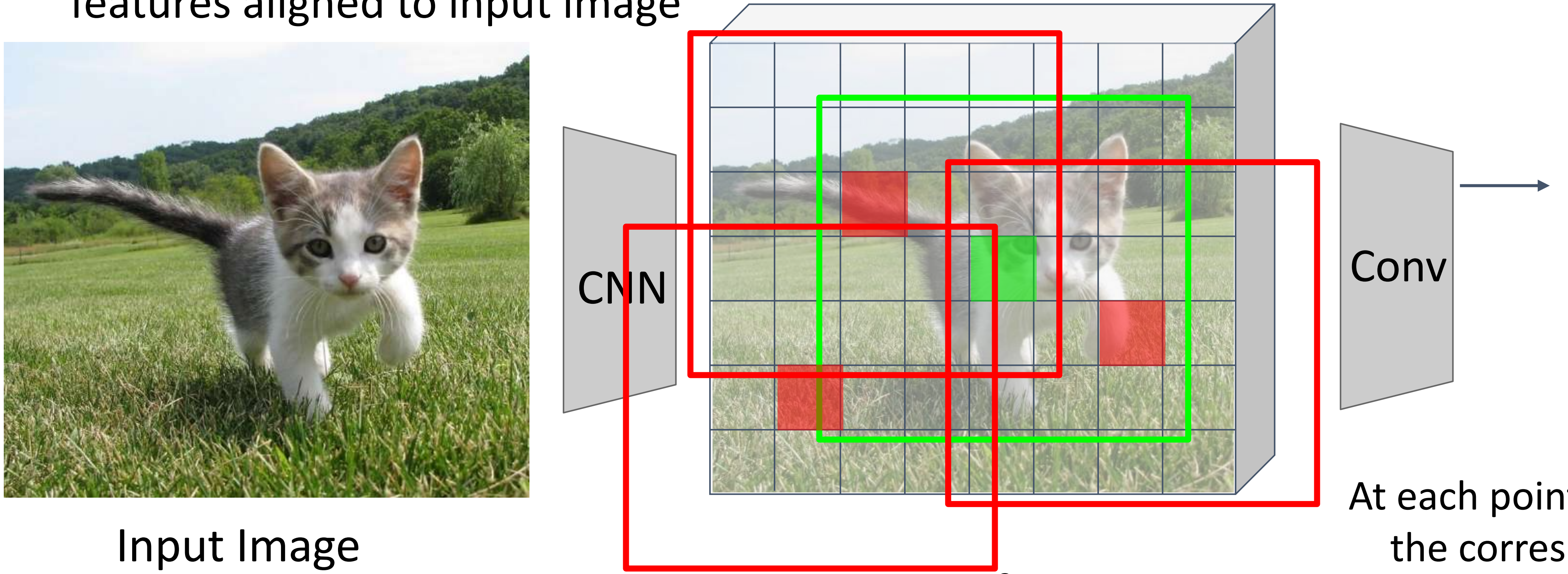



Image features
(e.g. 512 x 20 x 15)

# 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




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

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

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)

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

Run backbone CNN to get features aligned to input image

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

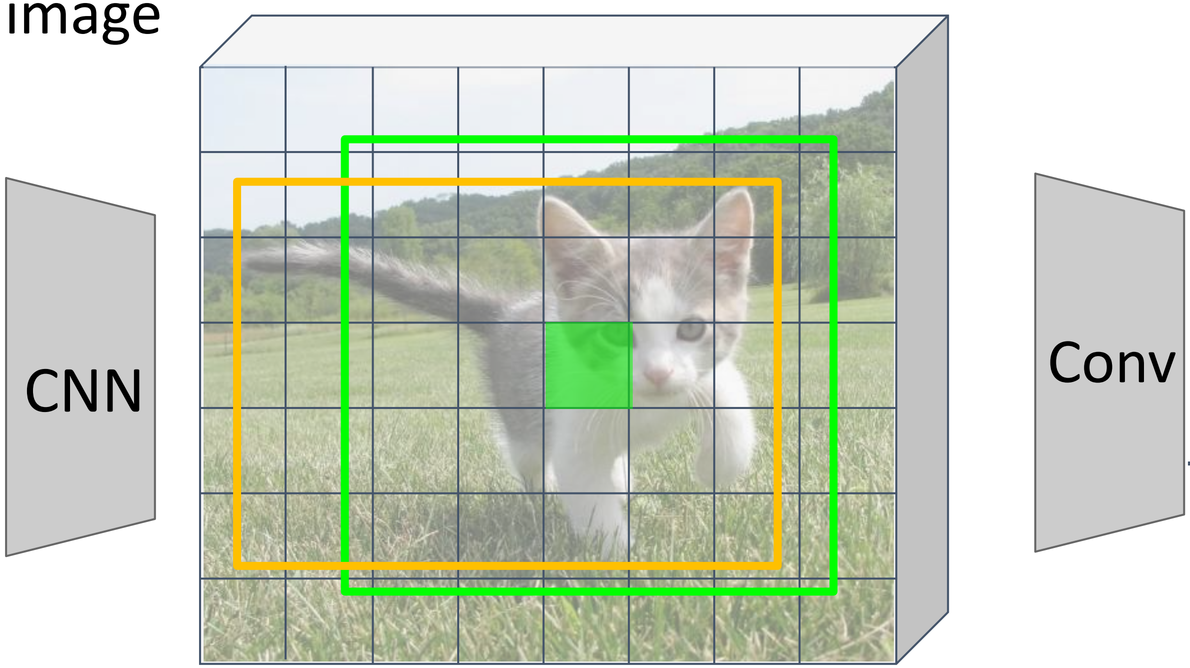



Image features
(e.g. 512 x 20 x 15)

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)

**Problem**: Anchor box may have the wrong size / shape
**Solution**: Use **K** different anchor boxes at each point!

Run backbone CNN to get features aligned to input image

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

CNN

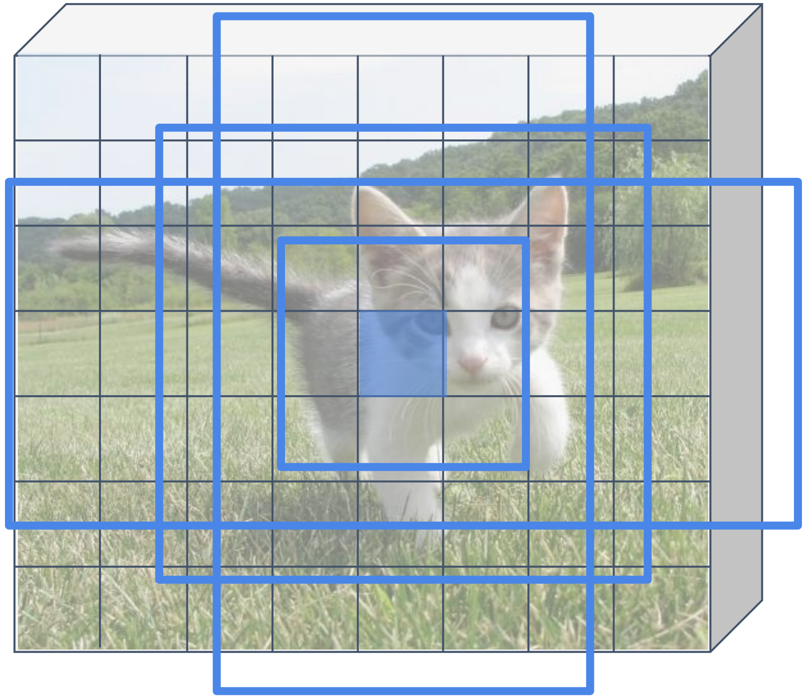


Image features
(e.g. 512 x 20 x 15)

Conv

Anchor is an object?
**K** x 20 x 15

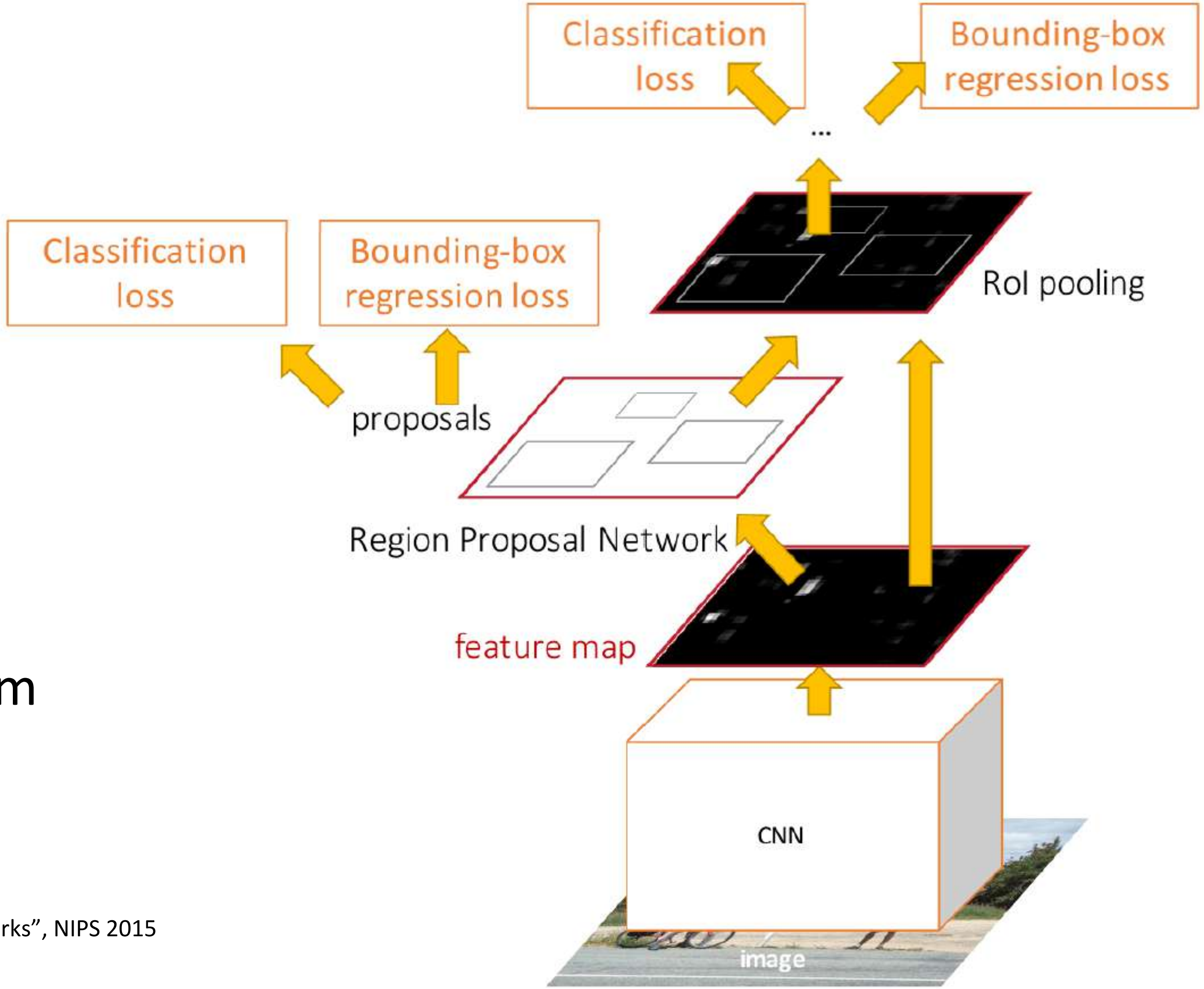
Box transforms
**4K** x 20 x 15

At test time: sort all K*20*15 boxes by their score, and take the top ~300 as our region proposals

# Faster R-CNN: Learnable Region Proposals

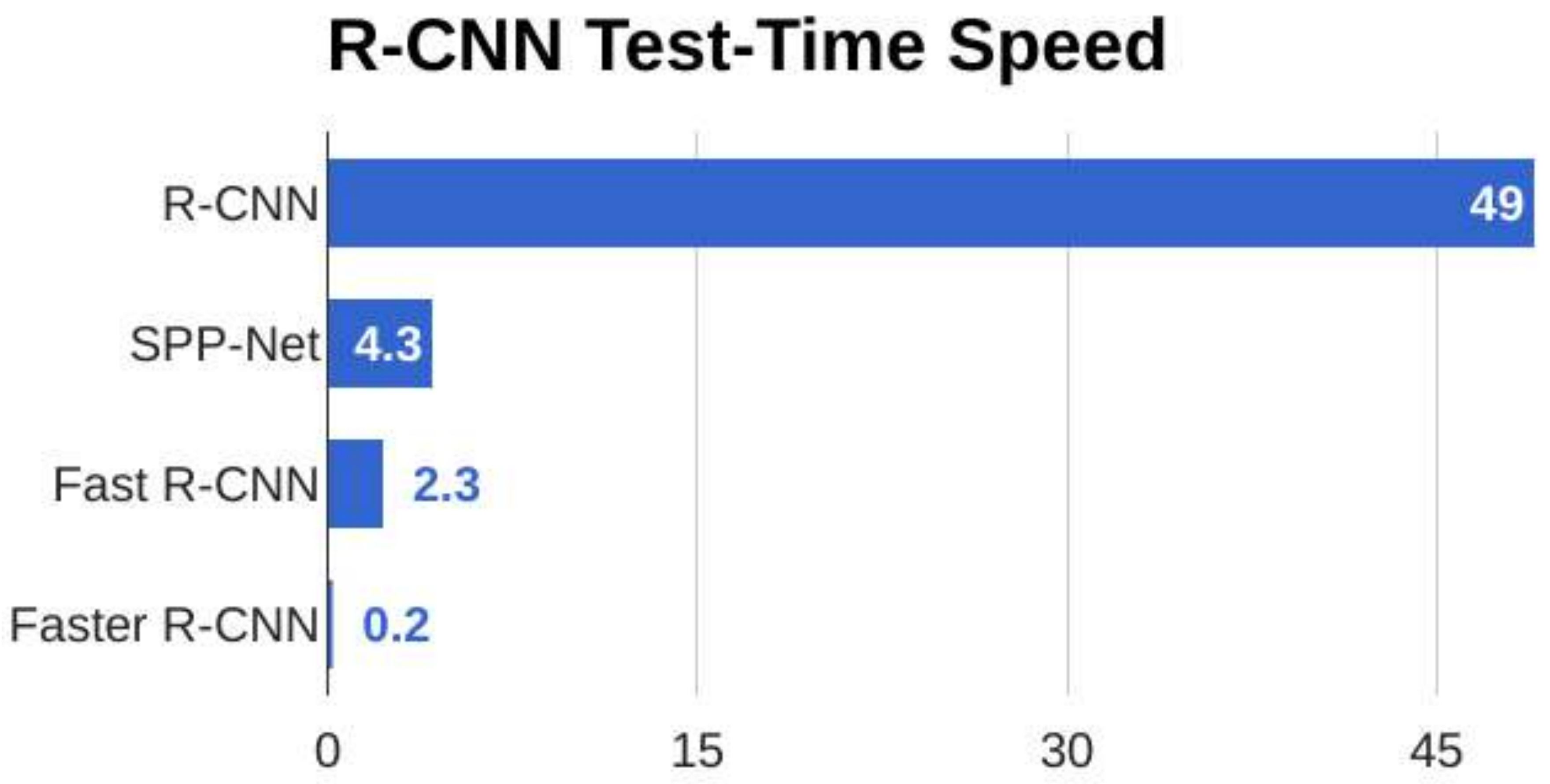
Jointly train with 4 losses:

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

# Faster R-CNN: Learnable Region Proposals

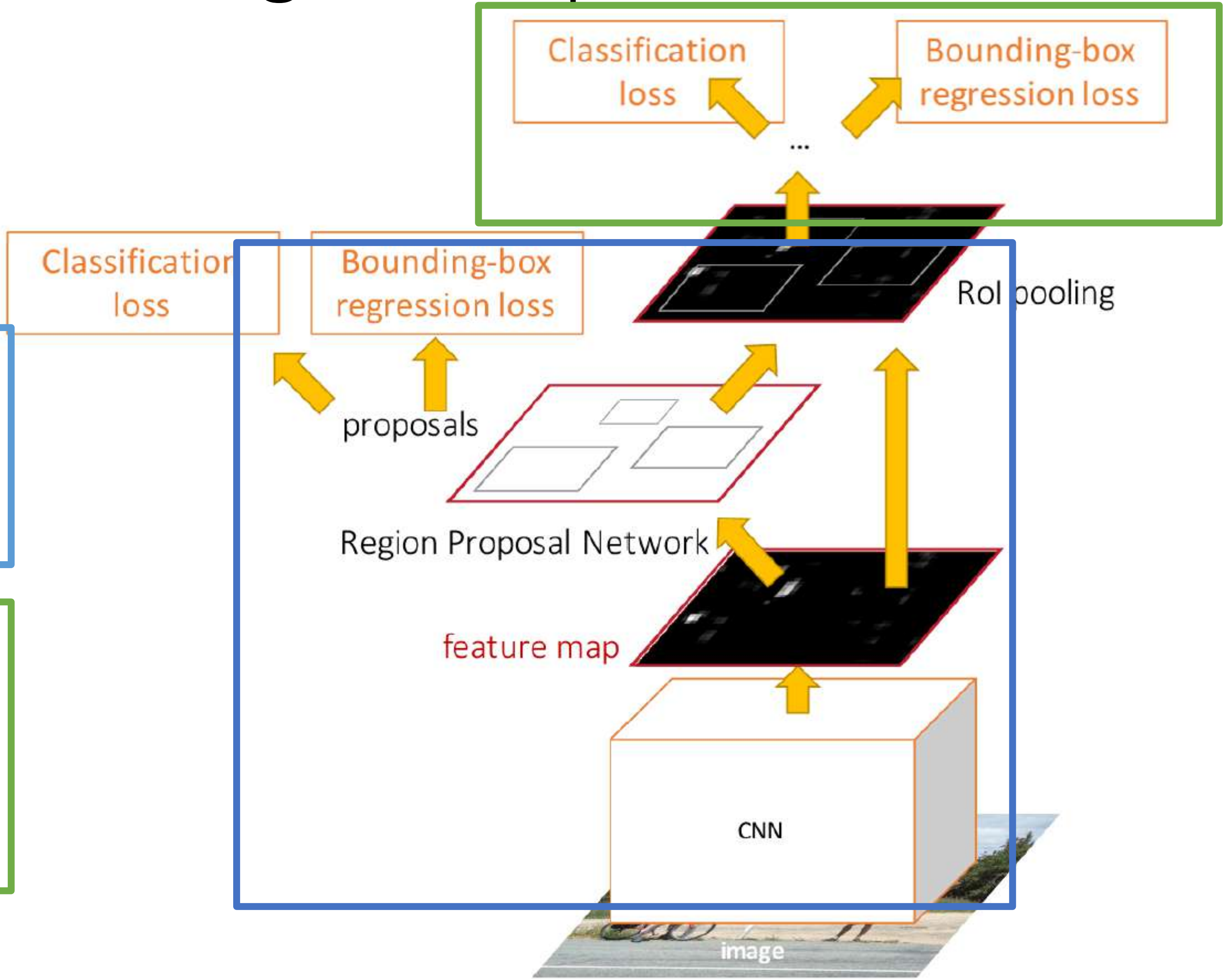


# Faster R-CNN: Learnable Region Proposals

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



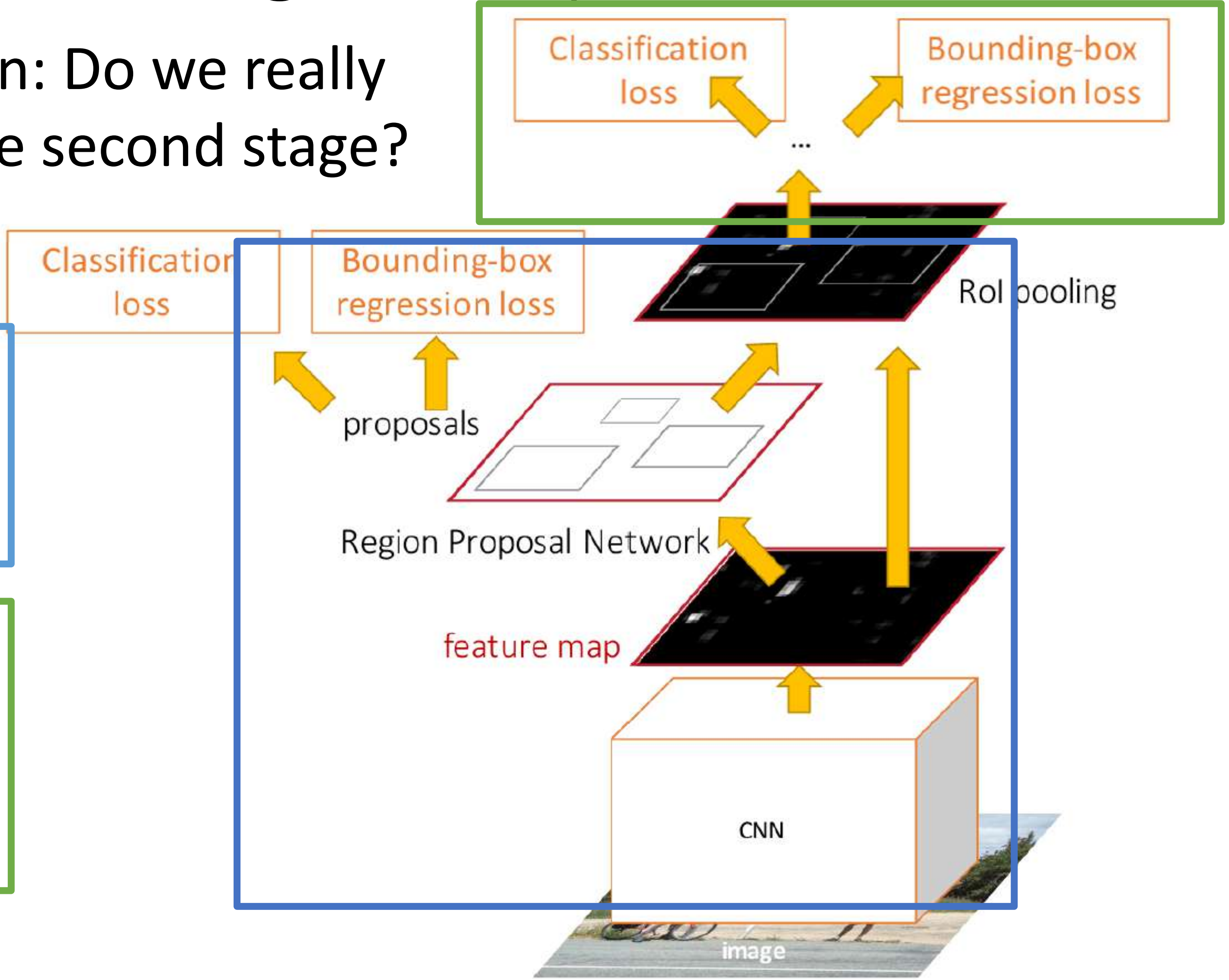
# Faster R-CNN: Learnable Region Proposals

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

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

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

Run backbone CNN to get features aligned to input image

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

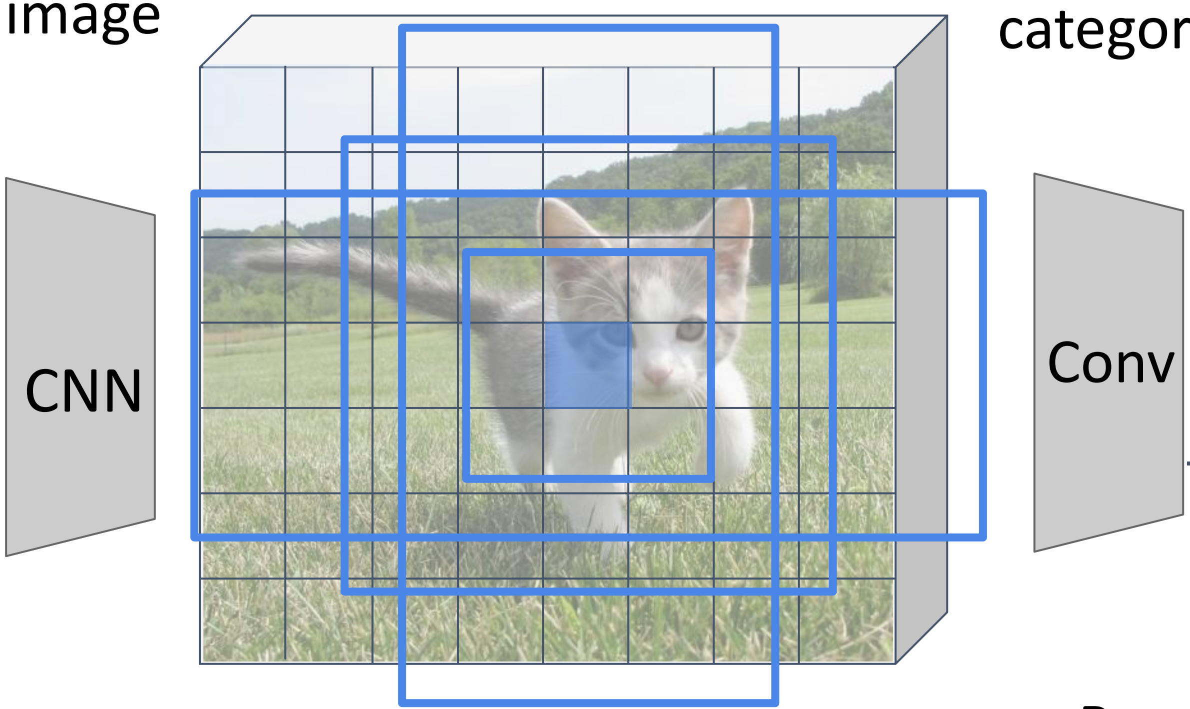



Image features
(e.g. 512 x 20 x 15)

Anchor category
(C+1) x K x 20 x 15

Box transforms
4K x 20 x 15

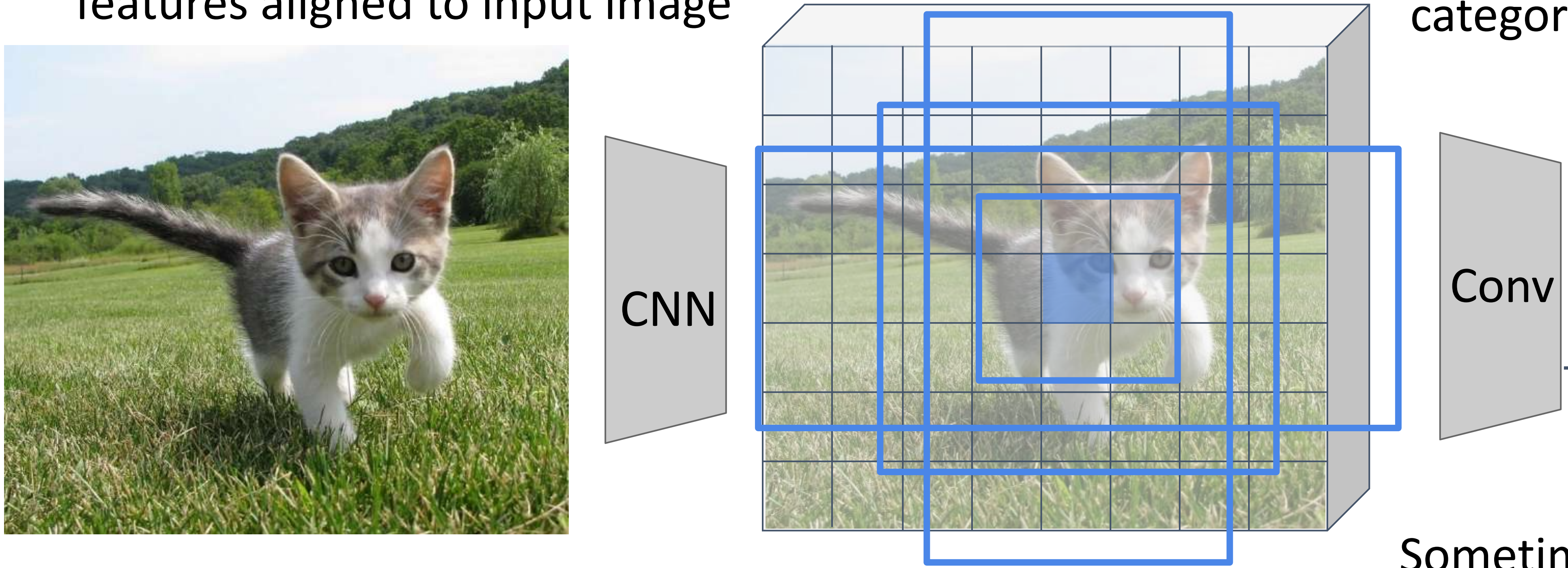
Remember: K anchors at each position in image feature map

# Single-Stage Object Detection

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

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

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)

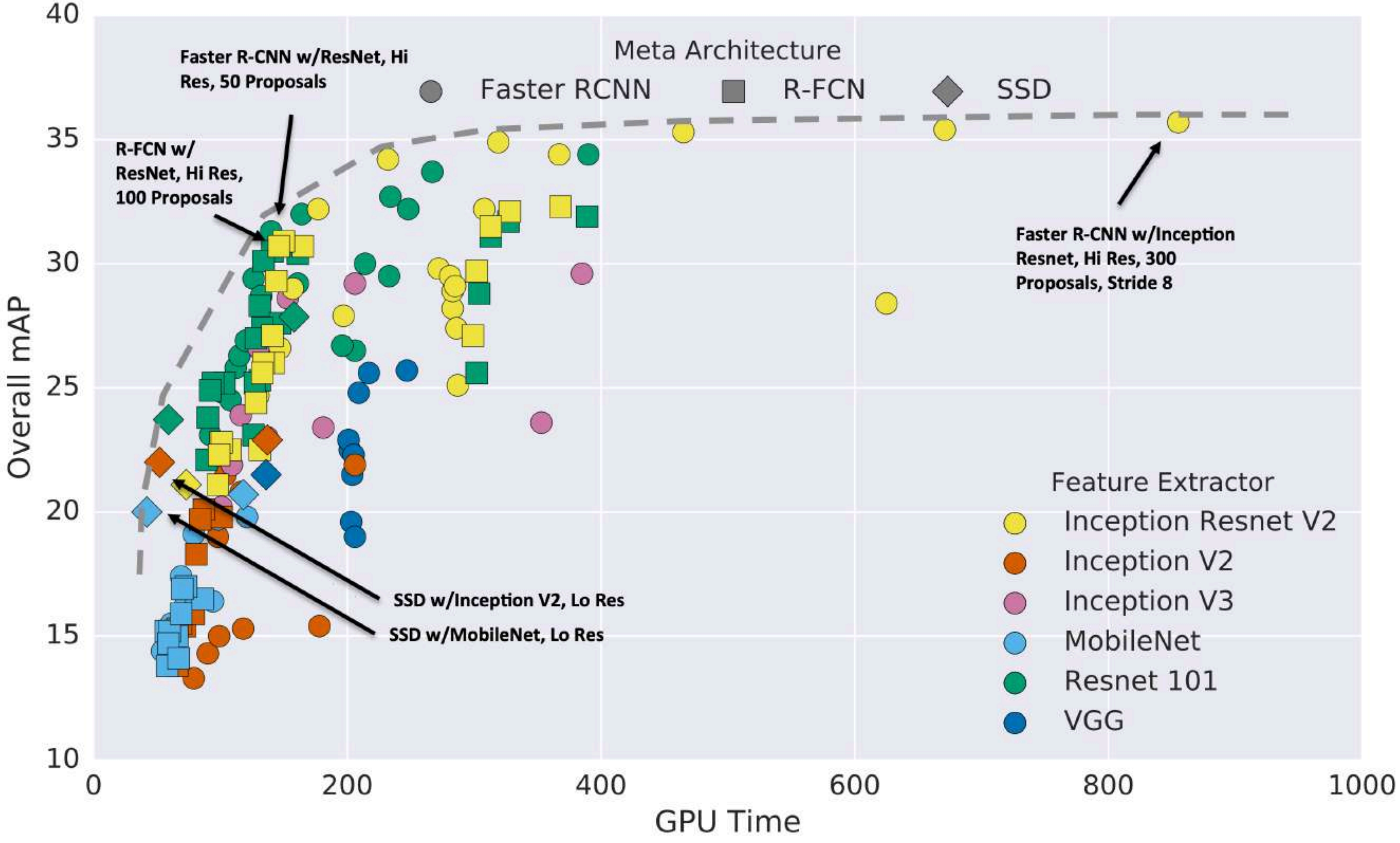
Anchor category
→ (C+1) x K x 20 x 15

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

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

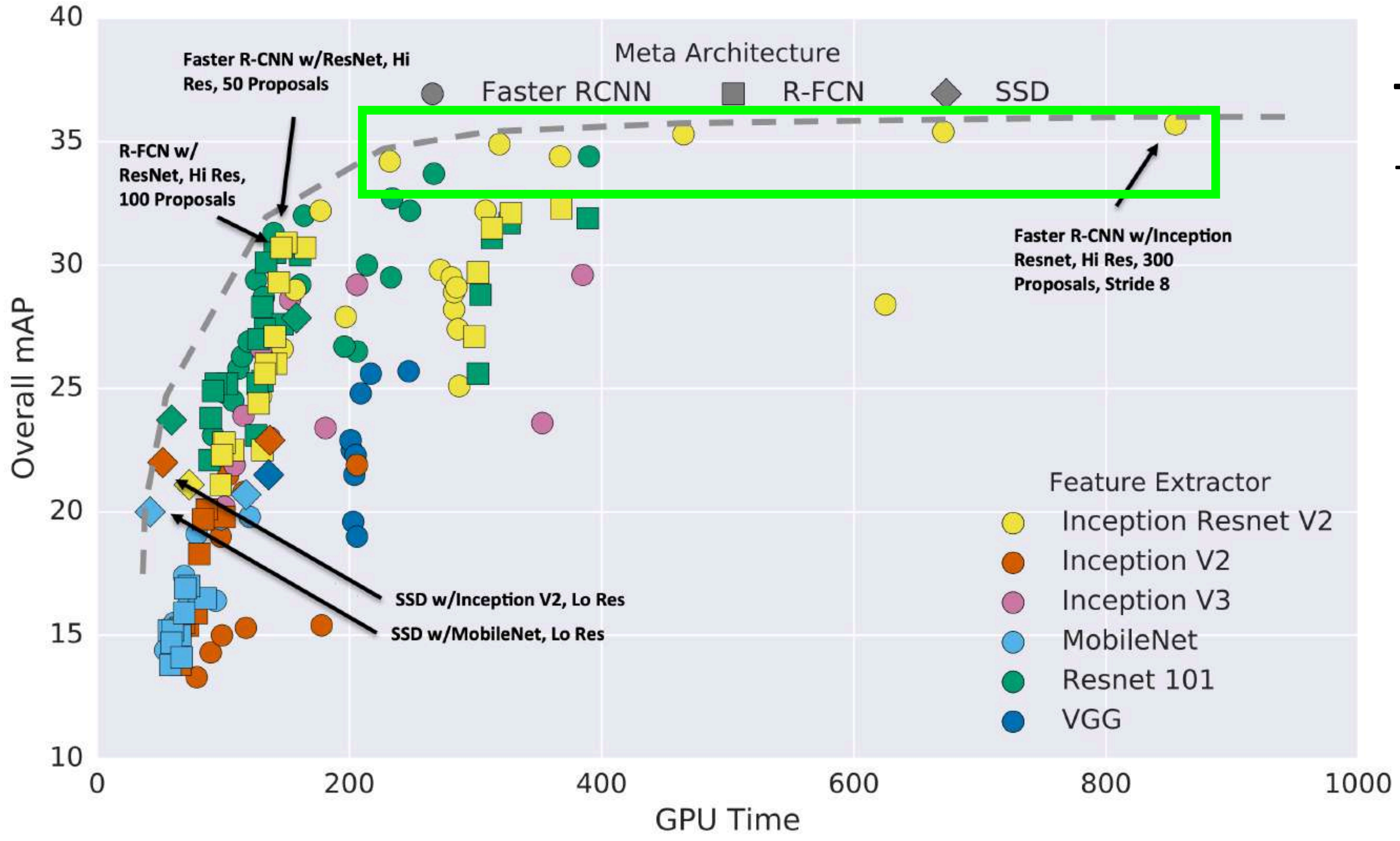
Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016
Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

# Object Detection: Lots of variables!




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

# Object Detection: Lots of variables!

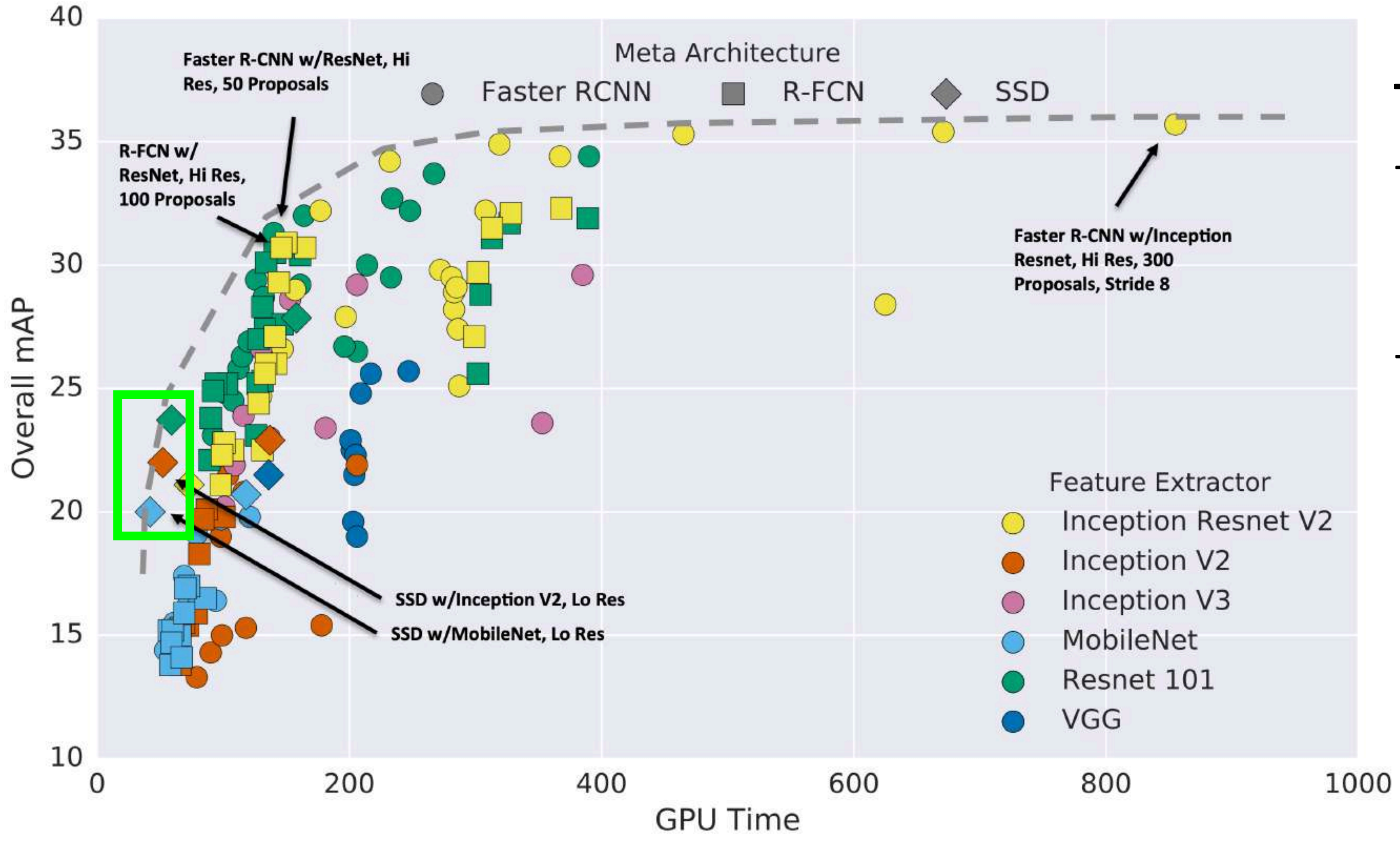



**Takeaways:**

- 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!

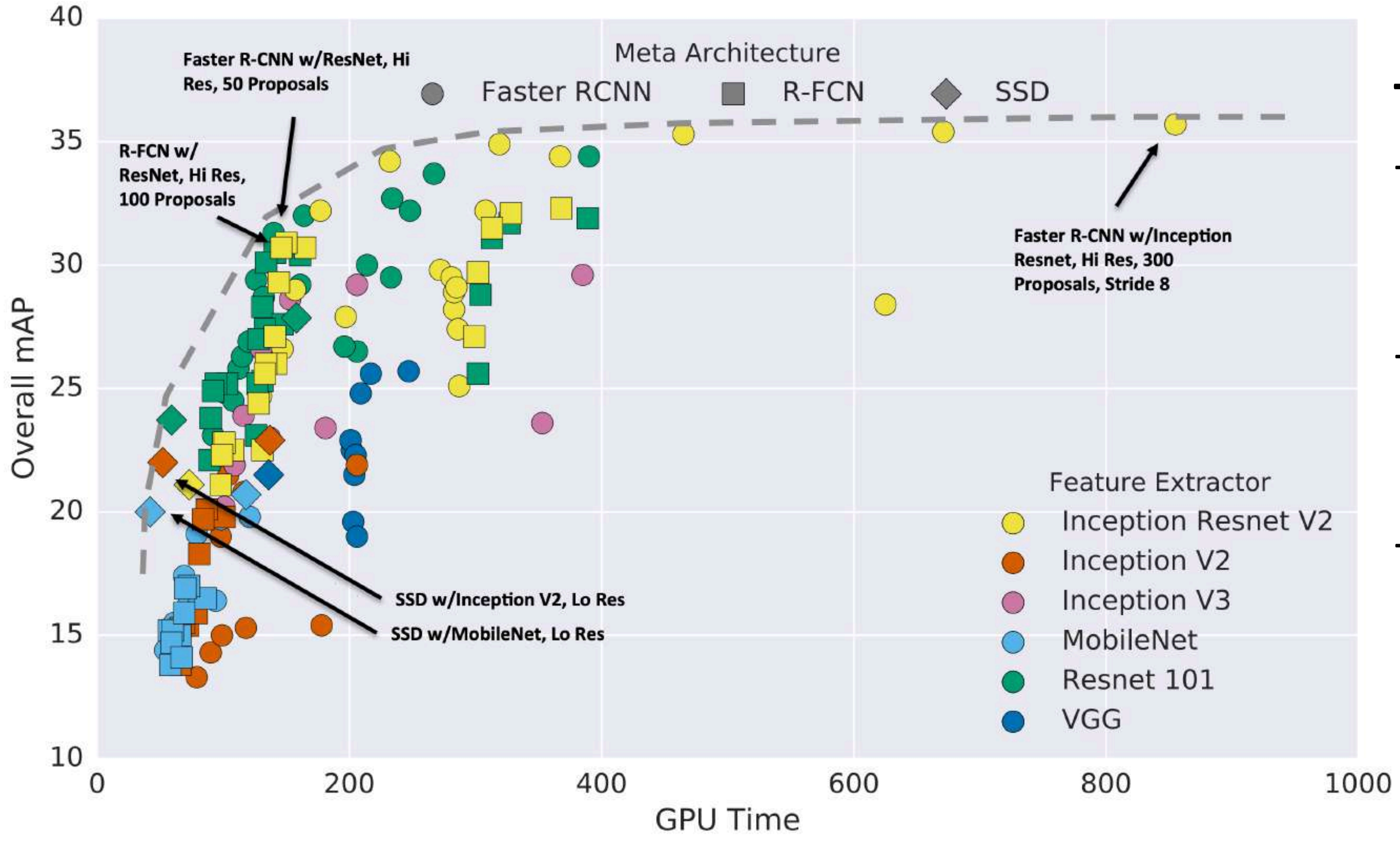



**Takeaways:**

- 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!

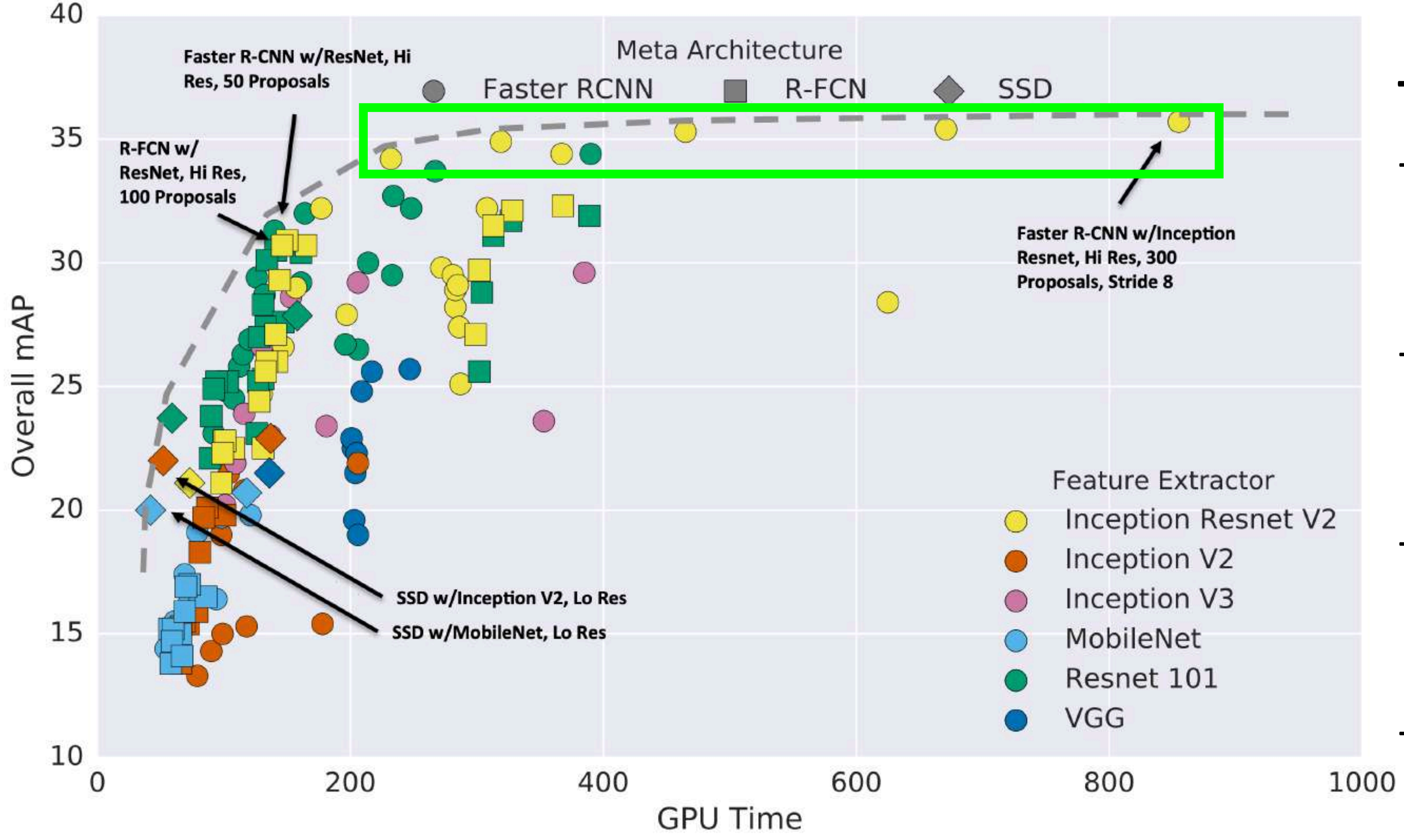



**Takeaways:**

- 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!



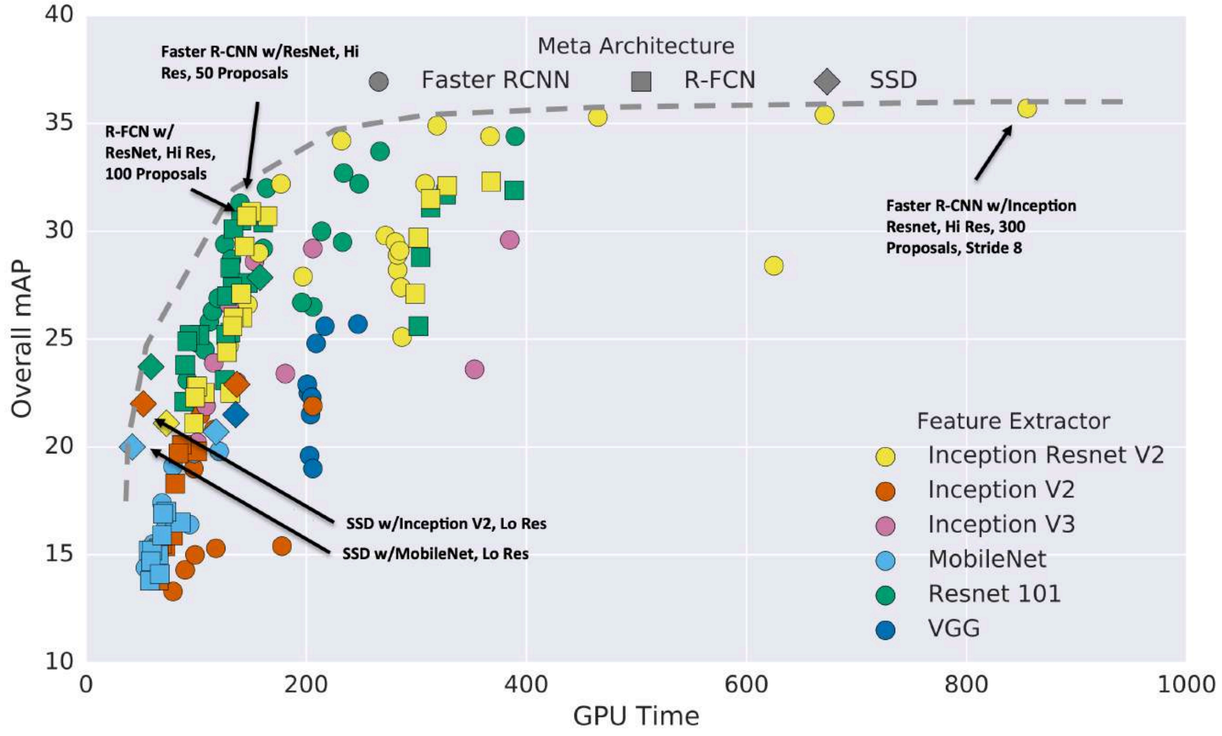

**Takeaways:**

- 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:

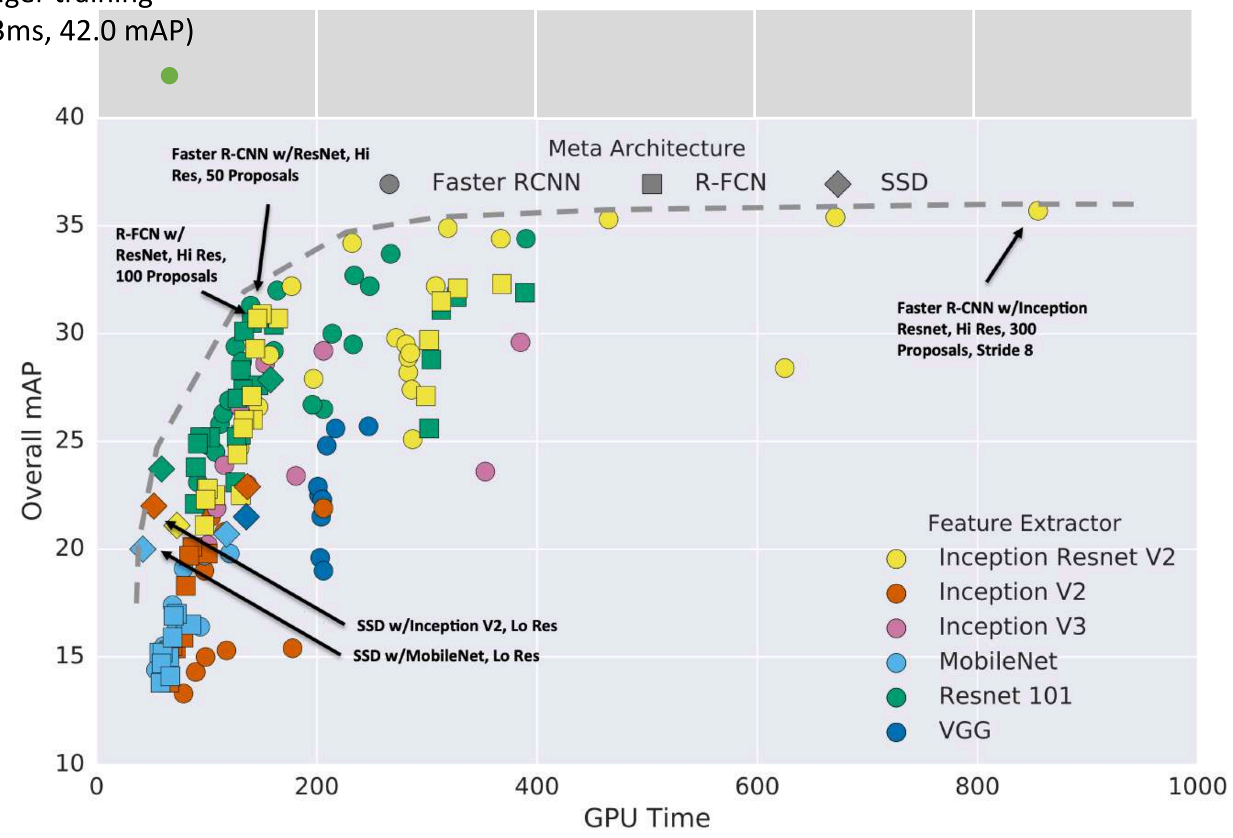



Huang et al, "Speed/accuracy trade-offs for modern convolutional object detectors", CVPR 2017

Wu et al, Detectron2, GitHub 2019

# Object Detection: Lots of variables!

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



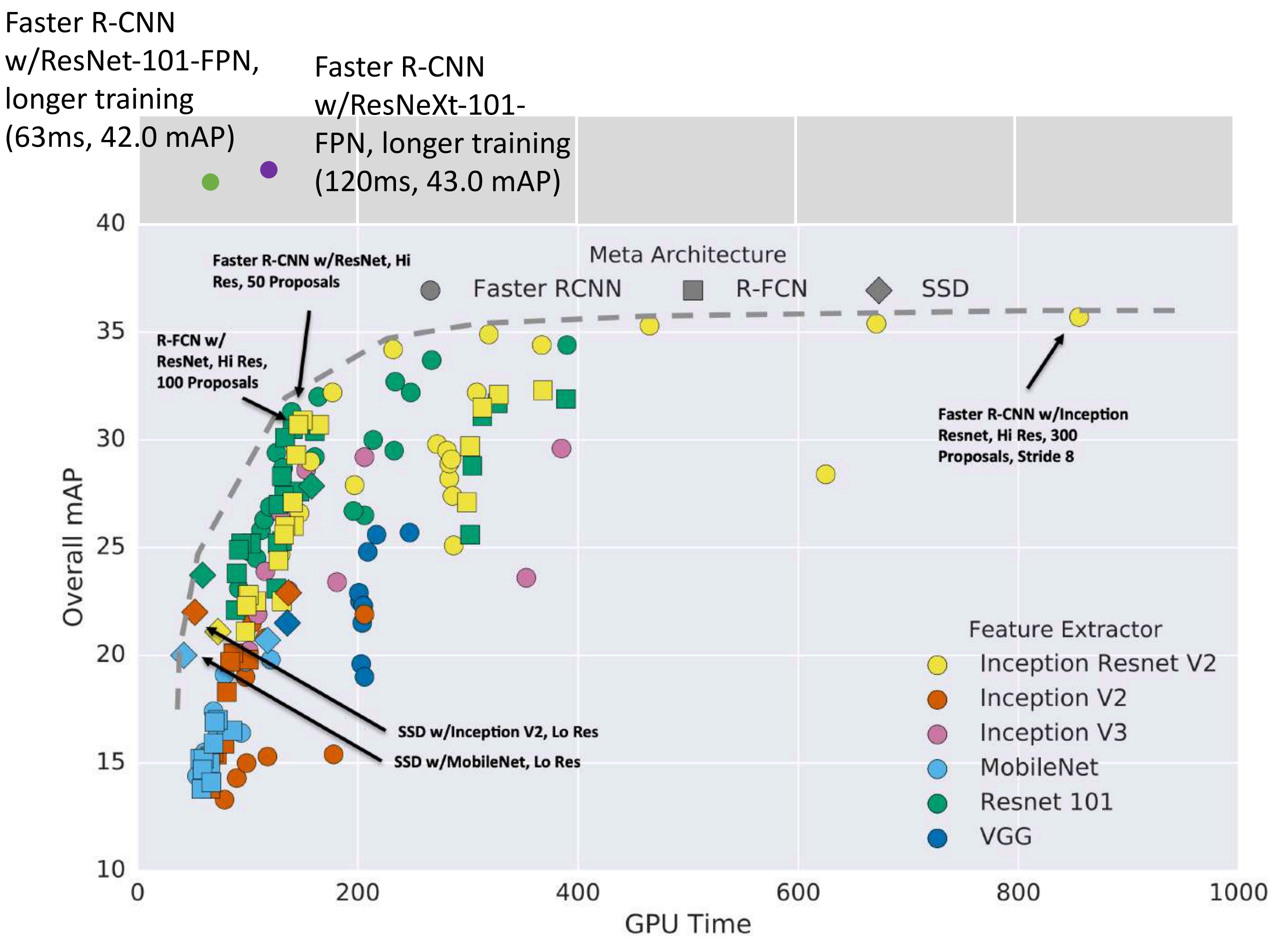

These results are a few years old … since then GPUs have gotten faster, and we've improved performance with many tricks:

- 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!



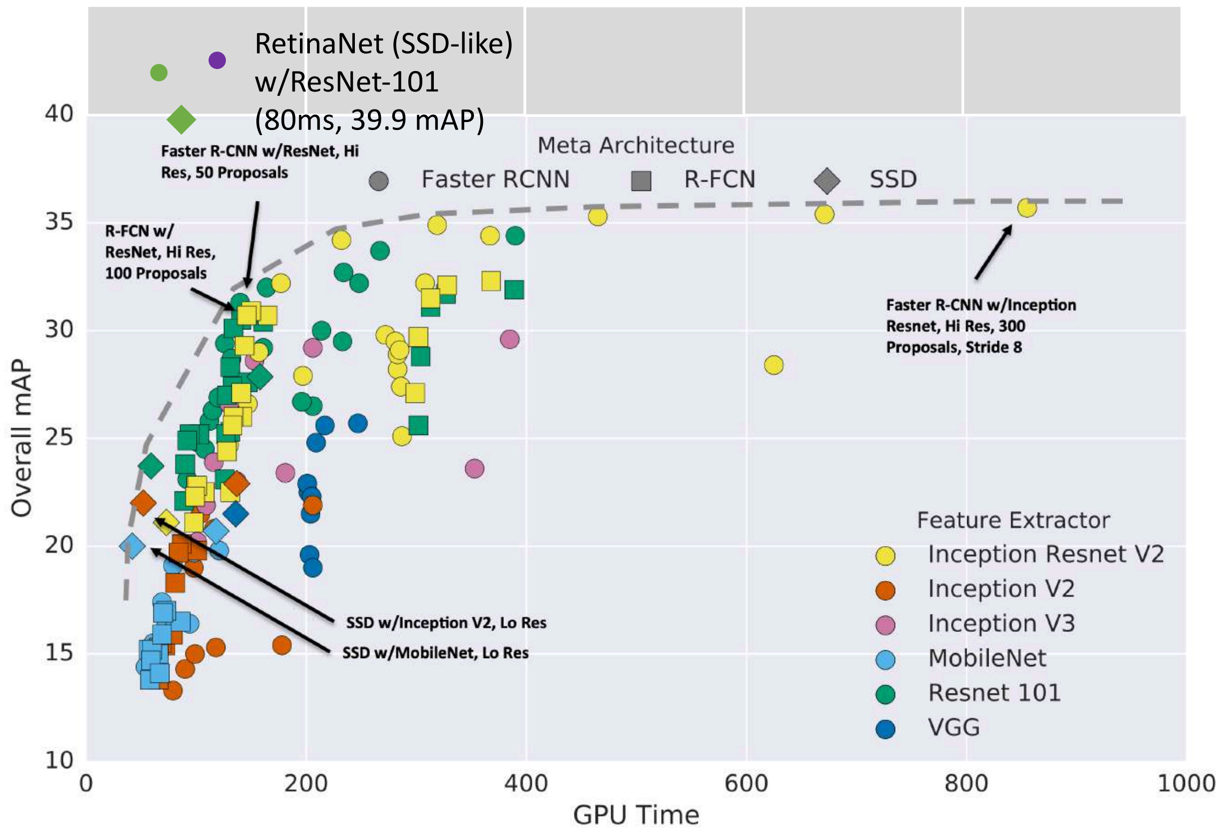

These results are a few years old … since then GPUs have gotten faster, and we've improved performance with many tricks:

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

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!



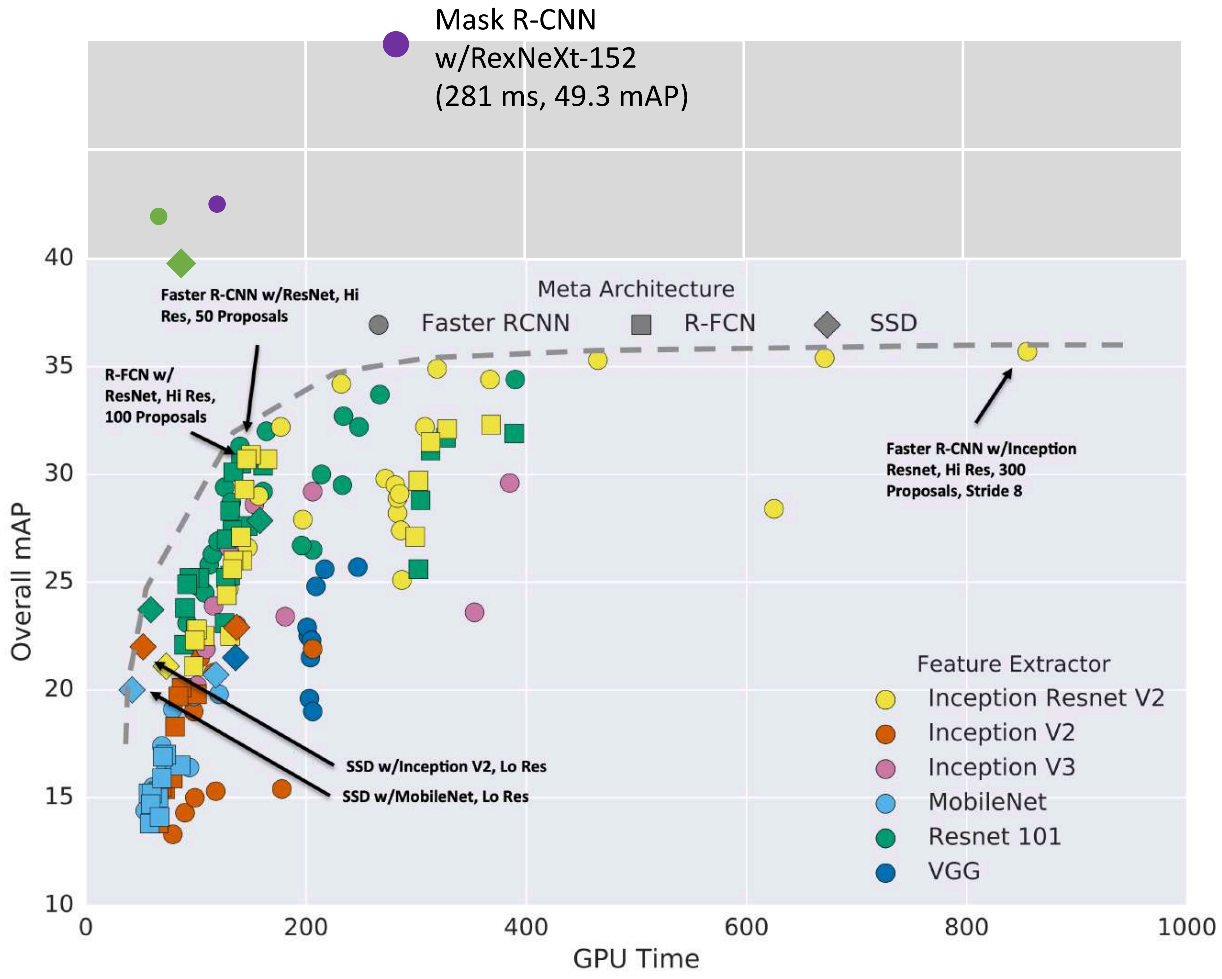

These results are a few years old … since then GPUs have gotten faster, and we've improved performance with many tricks:

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

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!



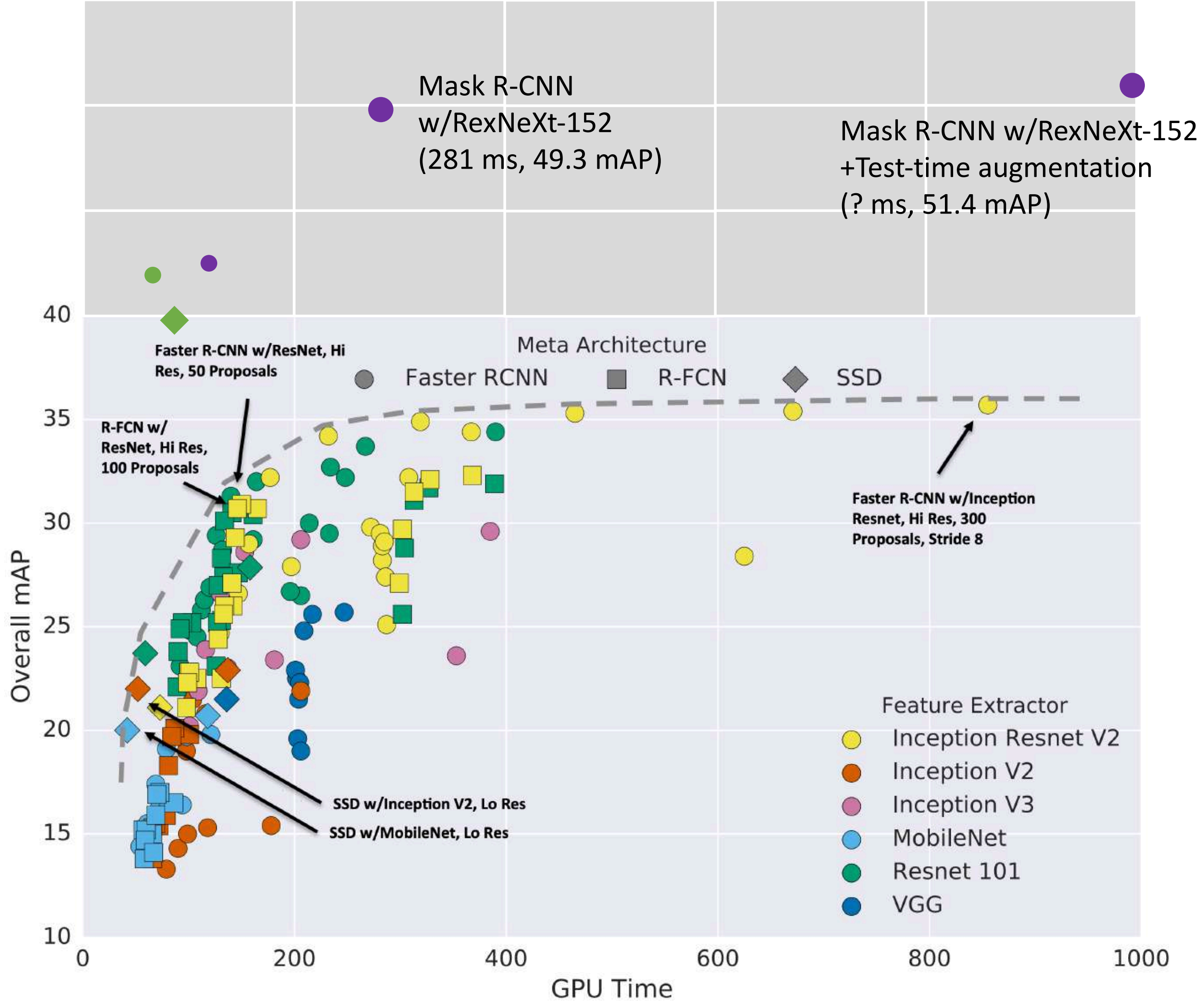

These results are a few years old … since then GPUs have gotten faster, and we've improved performance with many tricks:

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

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!



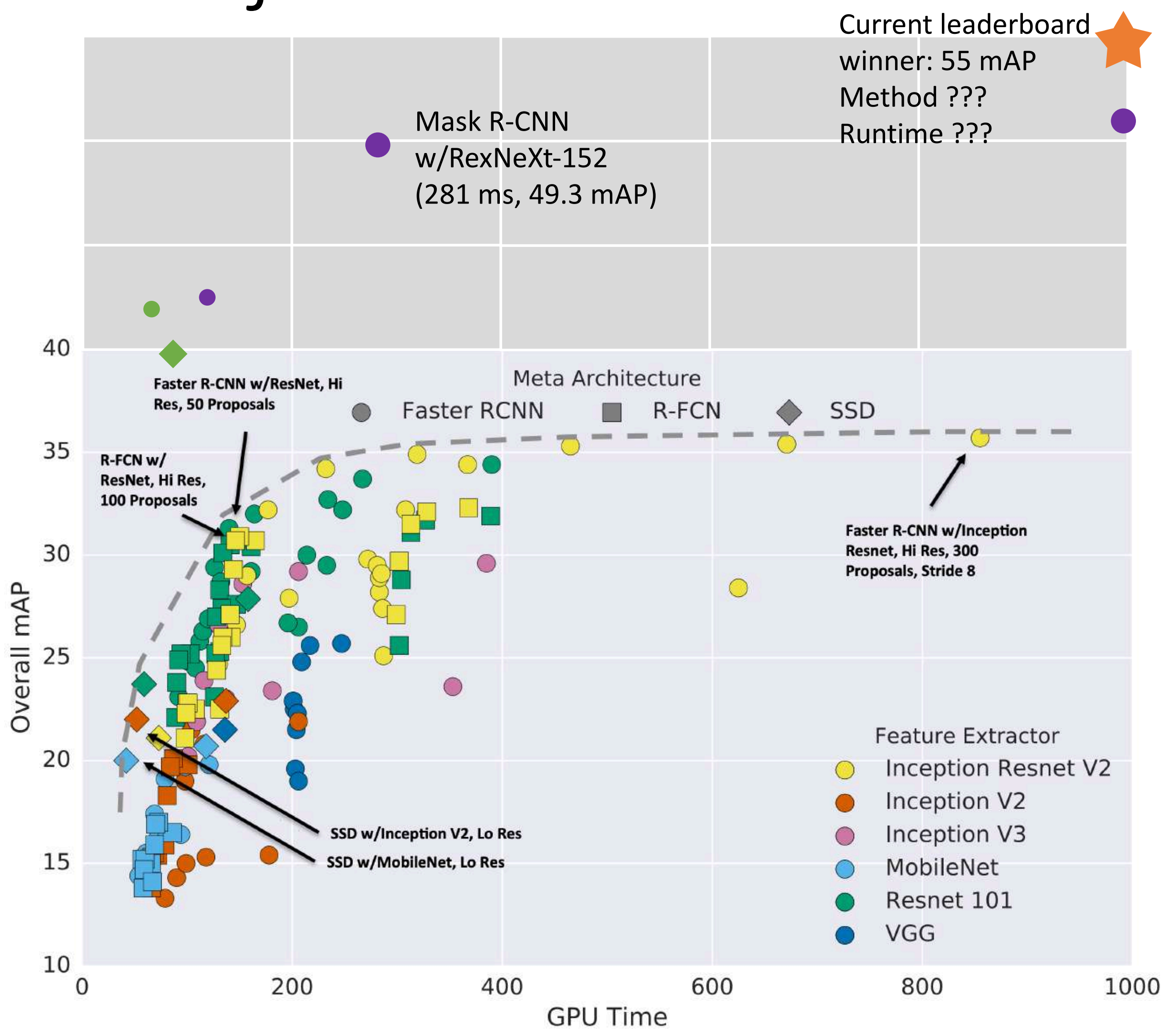

These results are a few years old … since then GPUs have gotten faster, and we've improved performance with many tricks:

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

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
- Single-Stage methods have improved
- Very big models work better
- Test-time augmentation pushes numbers up
- Big ensembles, more data, etc

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

https://competitions.codalab.org/competitions/20794#results

# Object Detection: Open-Source Code

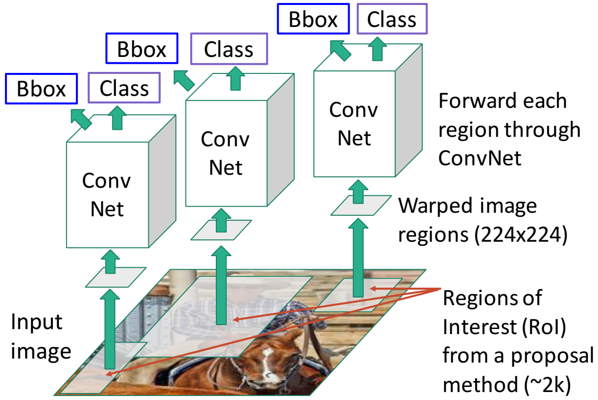
Object detection is hard! Don't implement it yourself
(Unless you are working on A5…)

**TensorFlow Detection API:**
https://github.com/tensorflow/models/tree/master/research/object_detection
Faster R-CNN, SSD, RFCN, Mask R-CNN

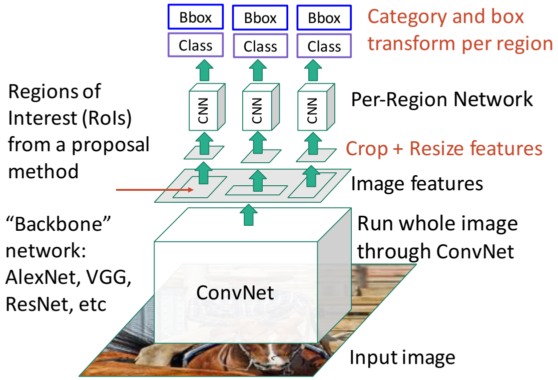
**Detectron2 (PyTorch):**
https://github.com/facebookresearch/detectron2
Fast / Faster / Mask R-CNN, RetinaNet

# Summary

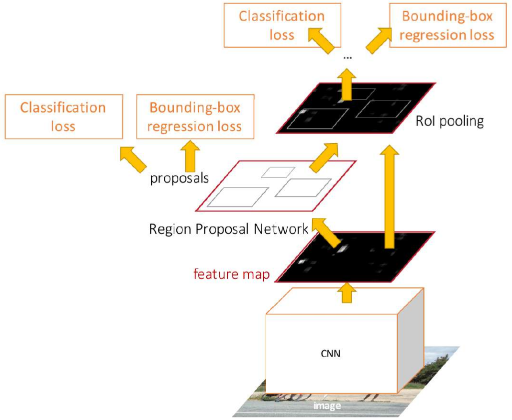
**"Slow" R-CNN**: Run CNN independently for each region



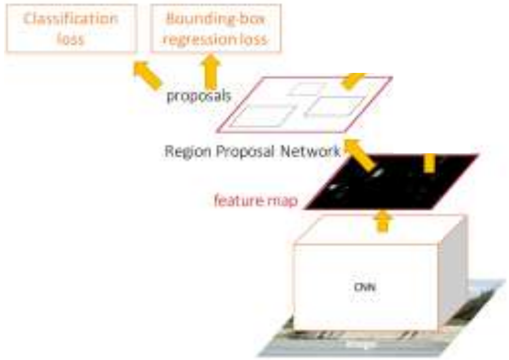

**Fast R-CNN**: Apply differentiable cropping to shared image features




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




**Single-Stage**: Fully convolutional detector



# Next Time:
# More localization methods:
# Segmentation, Keypoint Estimation