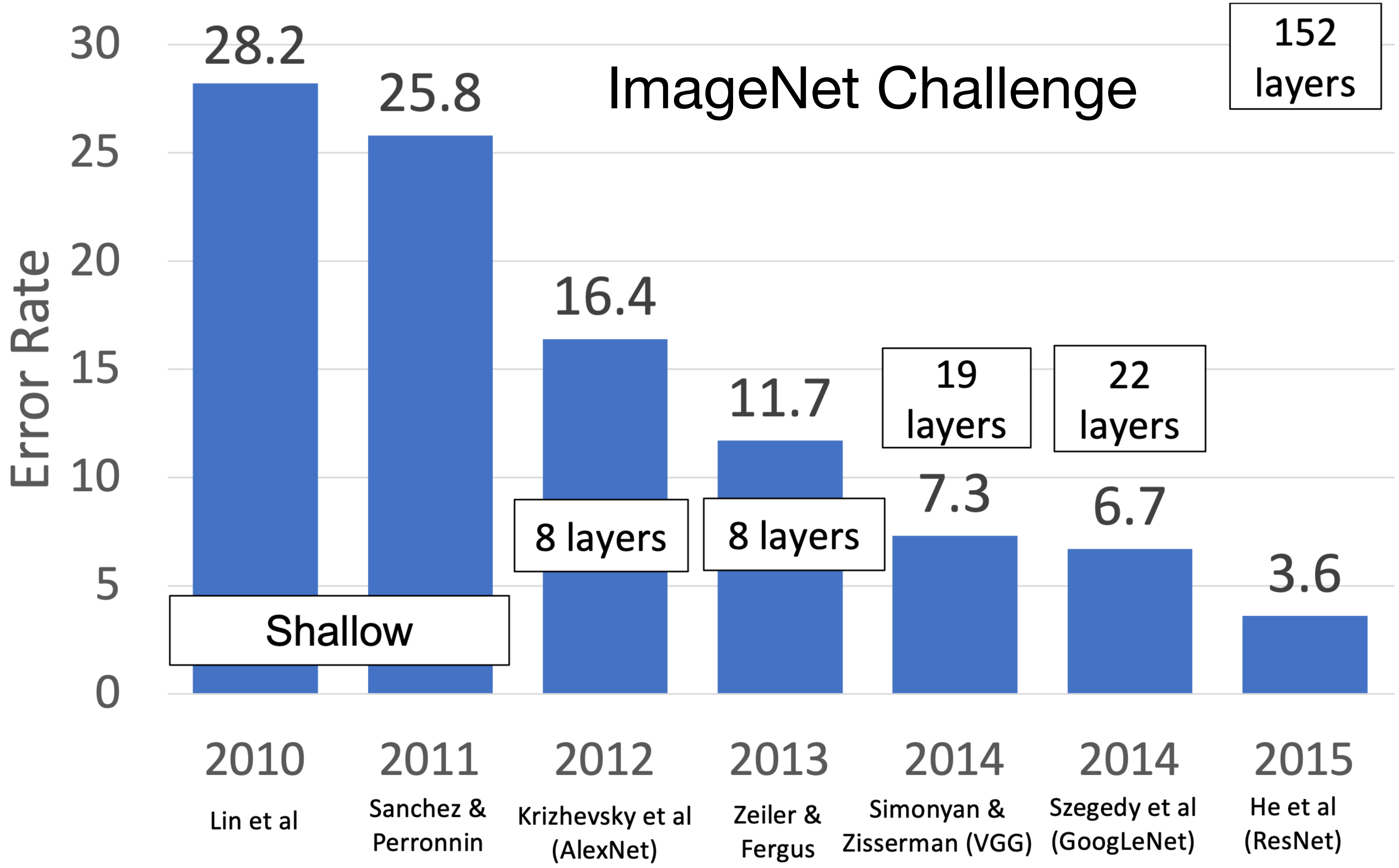
# DeepRob

Lecture 9
Object Detection
University of Michigan | Department of Robotics

DeepRob

# Recap: CNN architectures for Classification




Some key concepts:

VGG: stacked Conv+ReLU, pool

Stem network (GoogLeNet)

Residual Network (ResNet)

# Object Detection: Task definition

**Input:** Single RGB image

**Output:** A set of detected objects;

For each object predict:

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

# 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"):

$$\text{IoU} = \frac{\textit{Area of Intersection}}{\textit{Area of Union}}$$

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

# Detecting a single object

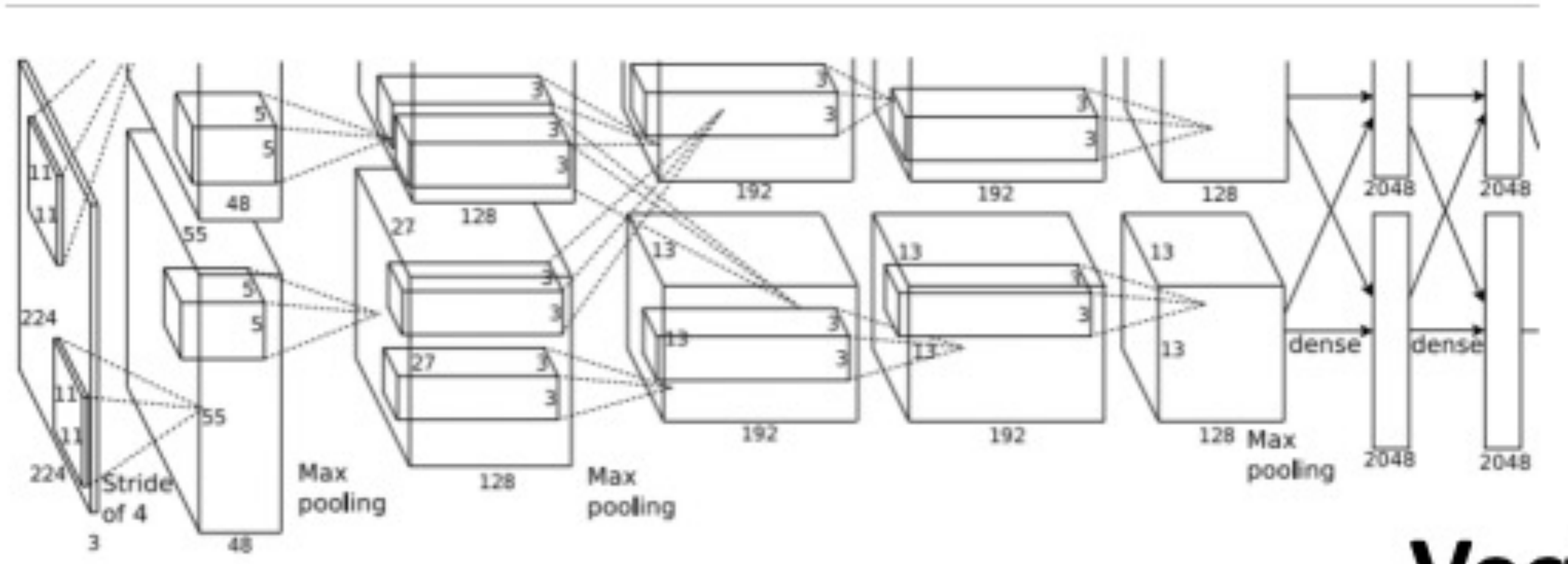




Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

**Vector:**
4096

**Fully connected:**
4096 to 10

What??

**Class scores:**
Chocolate Pretzels: 0.9
Granola Bar: 0.02
Potato Chips: 0.02
Water Bottle: 0.02
Popcorn: 0.01
....

**Correct Label:**
Chocolate Pretzels

Softmax Loss

**Treat localization as a regression problem!**

# Detecting a single object

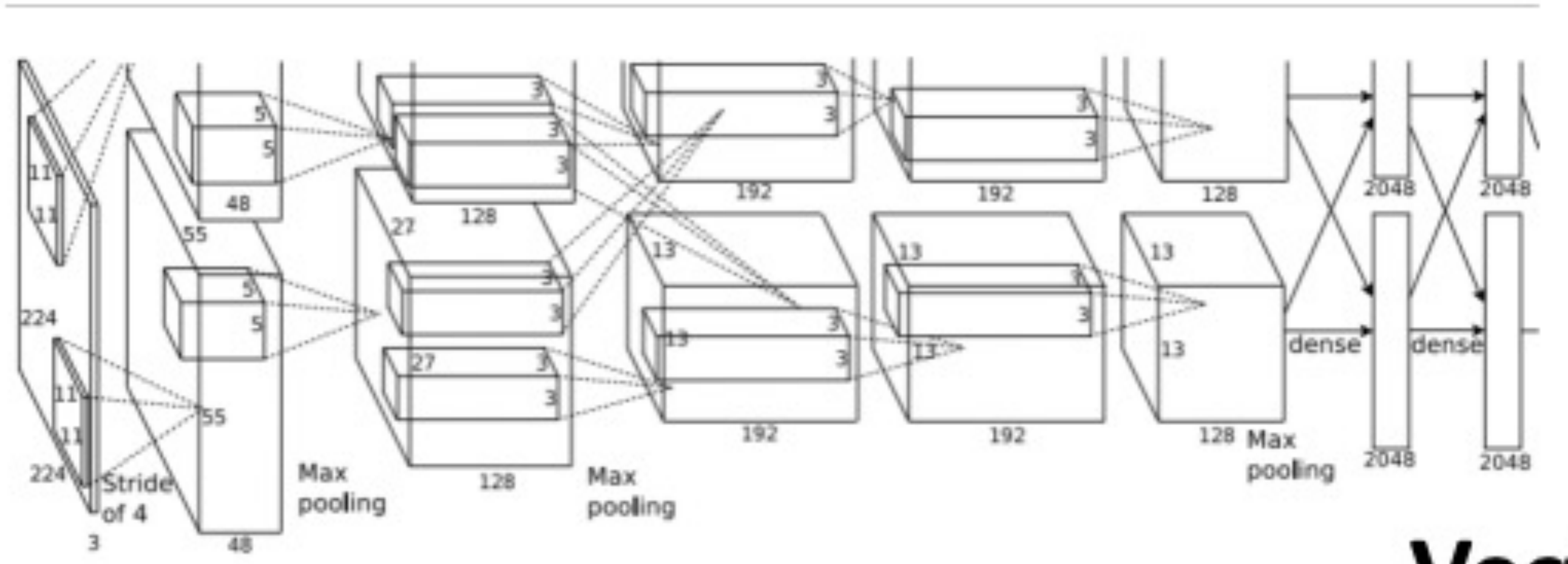




Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

**Vector:** 4096

What??

**Fully connected:** 4096 to 10

**Class scores:**
Chocolate Pretzels: 0.9
Granola Bar: 0.02
Potato Chips: 0.02
Water Bottle: 0.02
Popcorn: 0.01
....

→ Softmax Loss

**Correct Label:**
Chocolate Pretzels

**Fully connected:** 4096 to 10

**Box coordinates:**
(x, y, w, h)

→ L2 Loss

**Correct coordinates:**
(x’, y’, w’, h’)

Where??

**Treat localization as a regression problem!**

# Detecting a single object

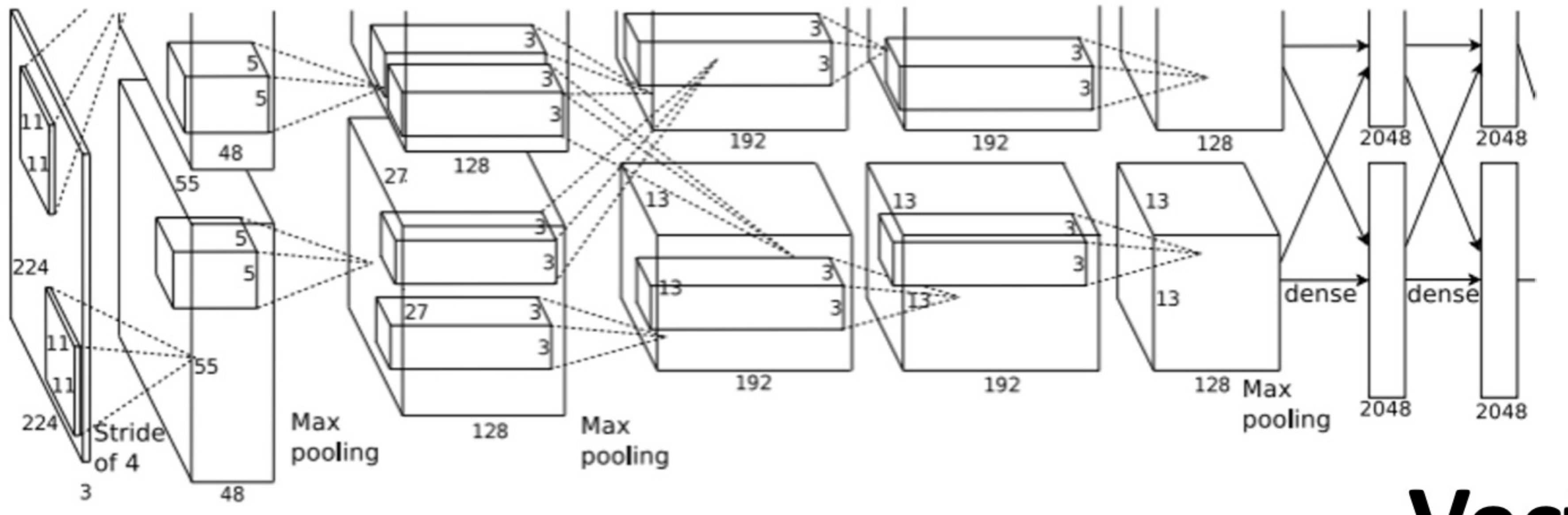




Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

**Vector:**
4096

**Fully connected:**
4096 to 10

**Class scores:**

**Chocolate Pretzels: 0.9**
What??
**Granola Bar: 0.02**
**Potato Chips: 0.02**
**Water Bottle: 0.02**
**Popcorn: 0.01**
**....**

**Correct Label:**

**Chocolate Pretzels**

**Softmax Loss**

**Fully connected:**
4096 to 10

**Box coordinates:**

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

Where??

**L2 Loss**

**Correct coordinates:**

**(x’, y', w’, h’)**

Multitask Loss

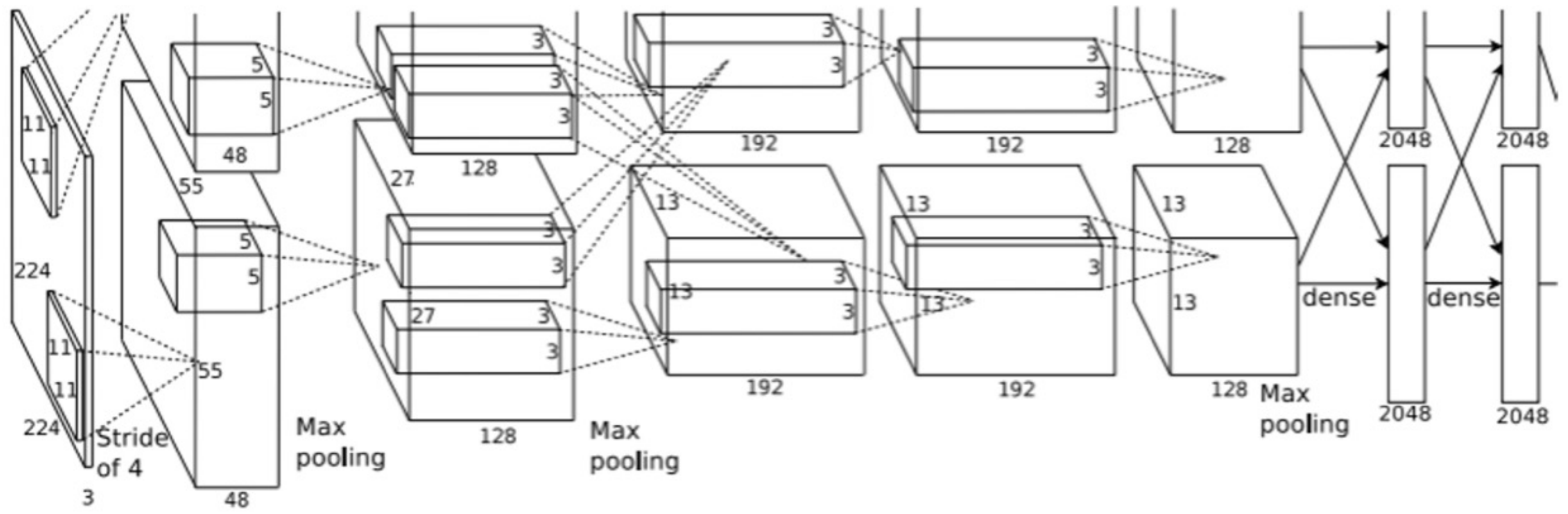
**Weighted Sum**

**Loss**

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

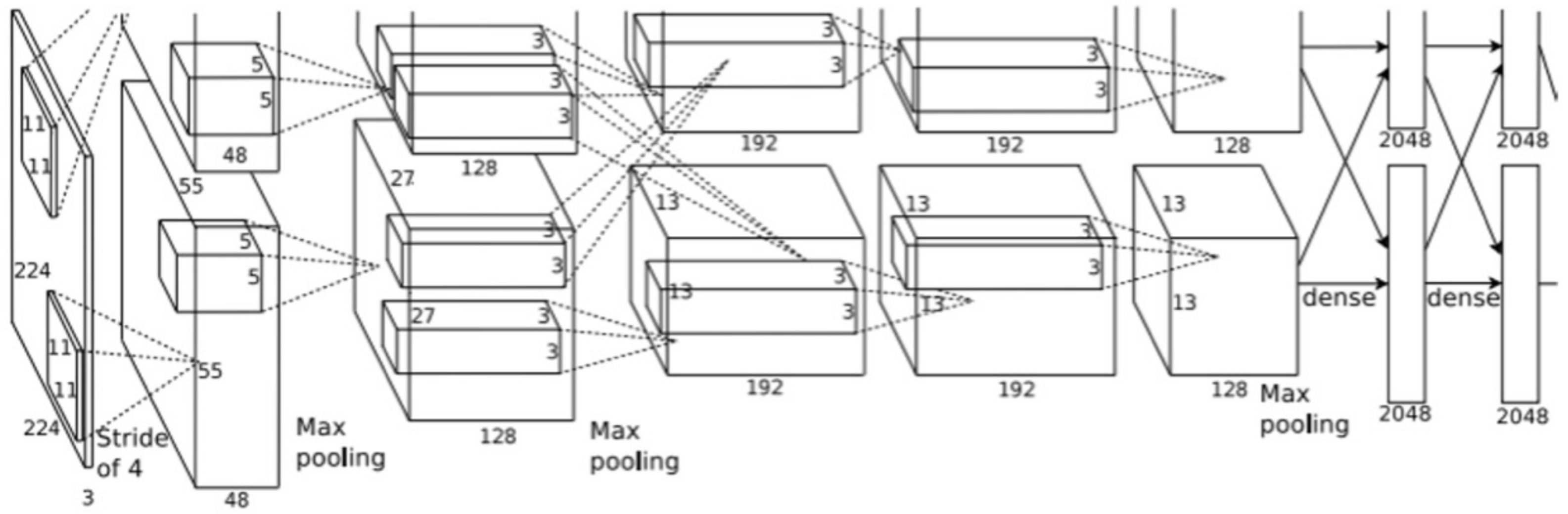
**Treat localization as a regression problem!**

# Detecting Multiple Objects



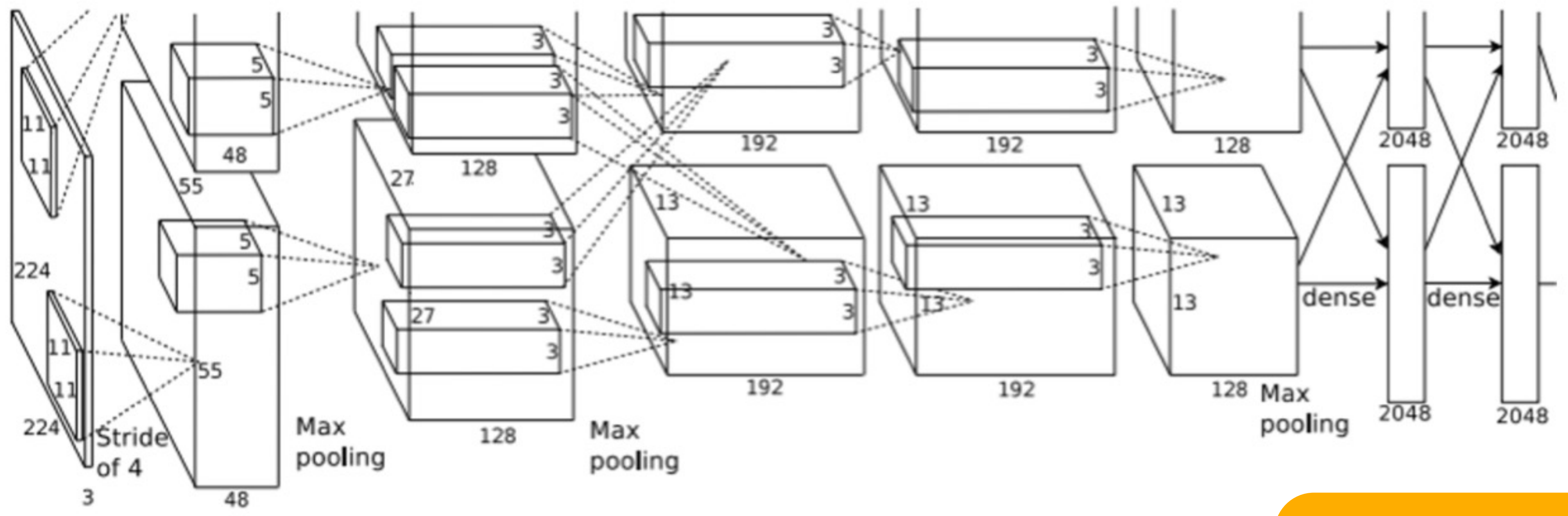
Hershey’s: (x, y, w, h)

4 numbers



Hershey’s: (x, y, w, h)
Flipz: (x, y, w, h)
Reese’s (x, y, w, h)

12 numbers



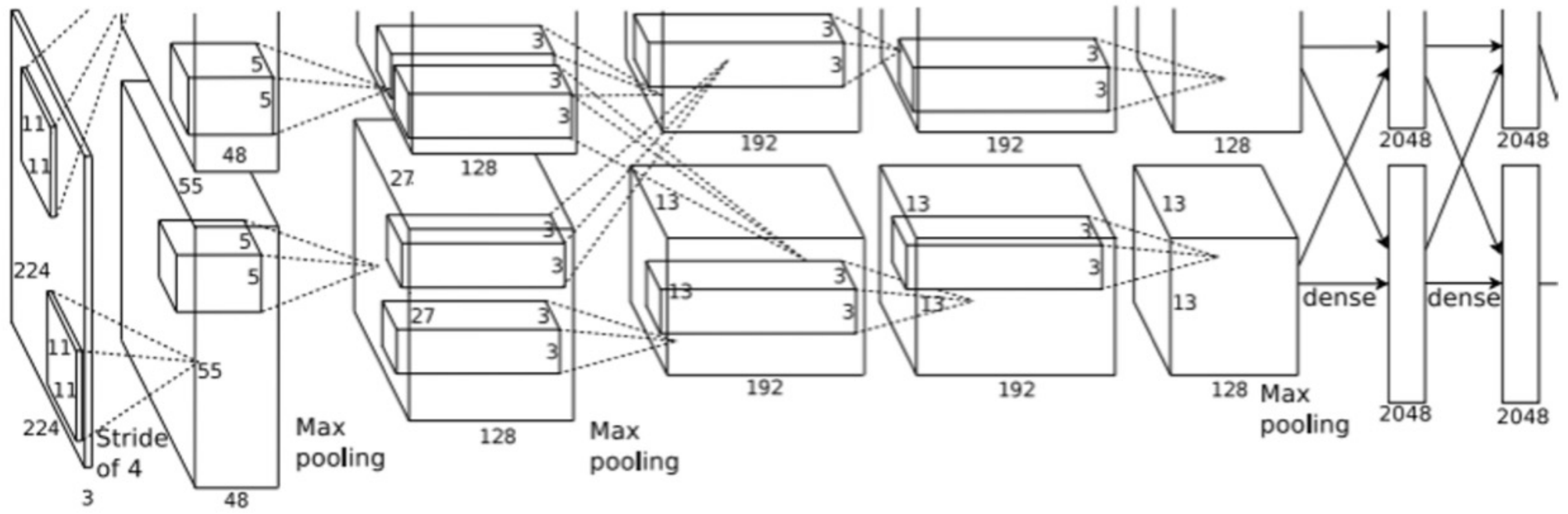
Chips: (x, y, w, h)
Chips: (x, y, w, h)
…..

Many numbers!

**Need different numbers of output per image**

# Detecting Multiple Objects: Sliding Window


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




**Hershey's: No**

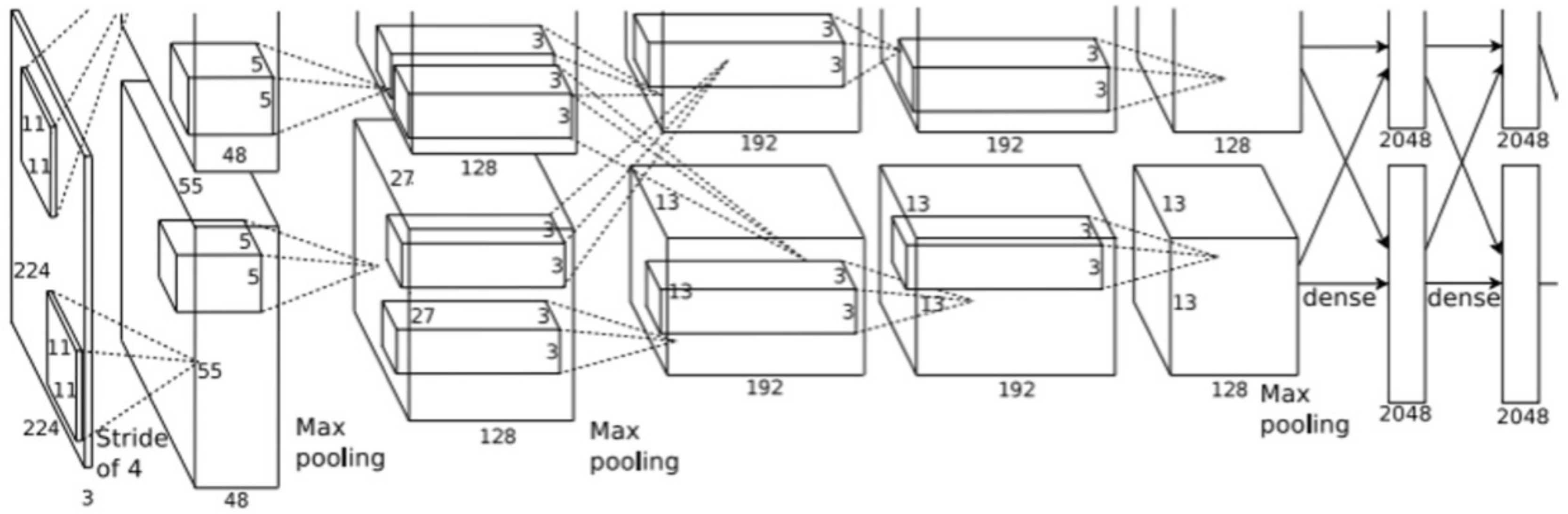
**Flipz: No**

**Reese's: 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**


**Hershey's: No**

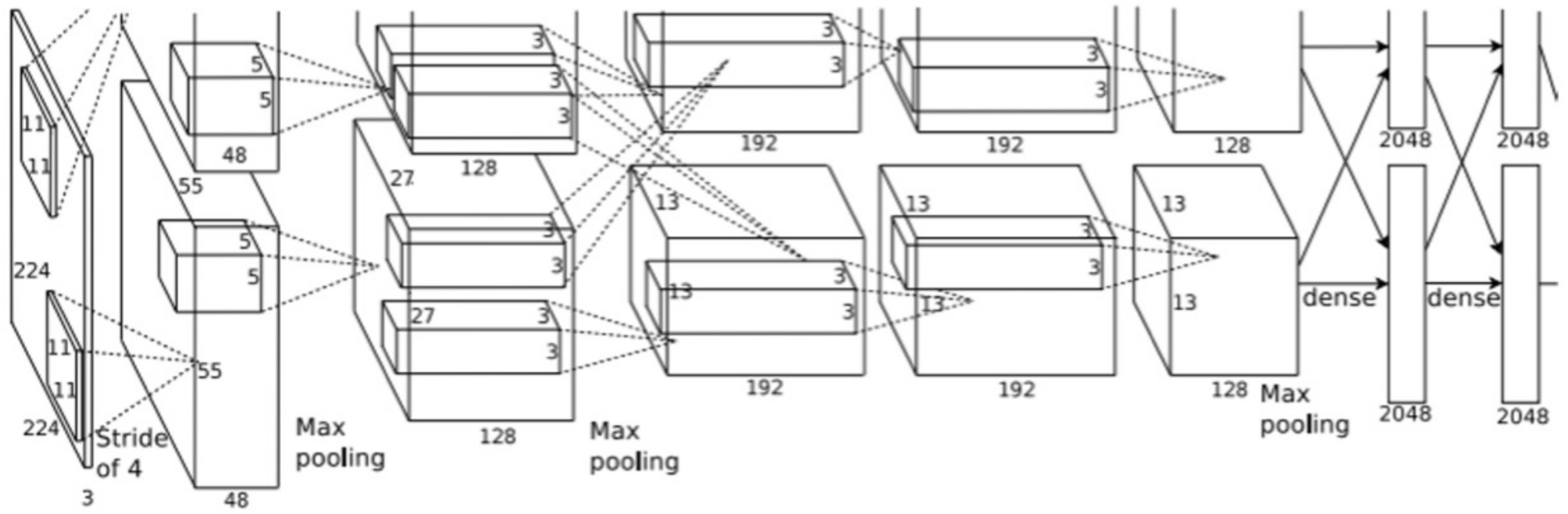
**Flipz: Yes**

**Reese's: 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**




**Hershey's: No**

**Flipz: No**

**Reese's: 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?**

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

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

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

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

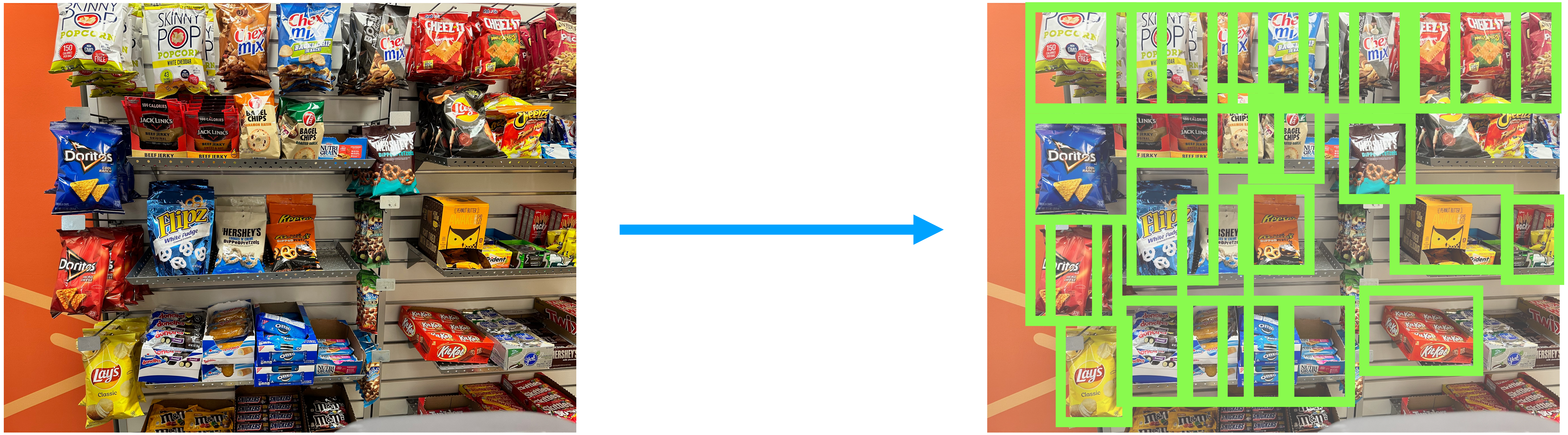
**Total possible boxes:**

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

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

# 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

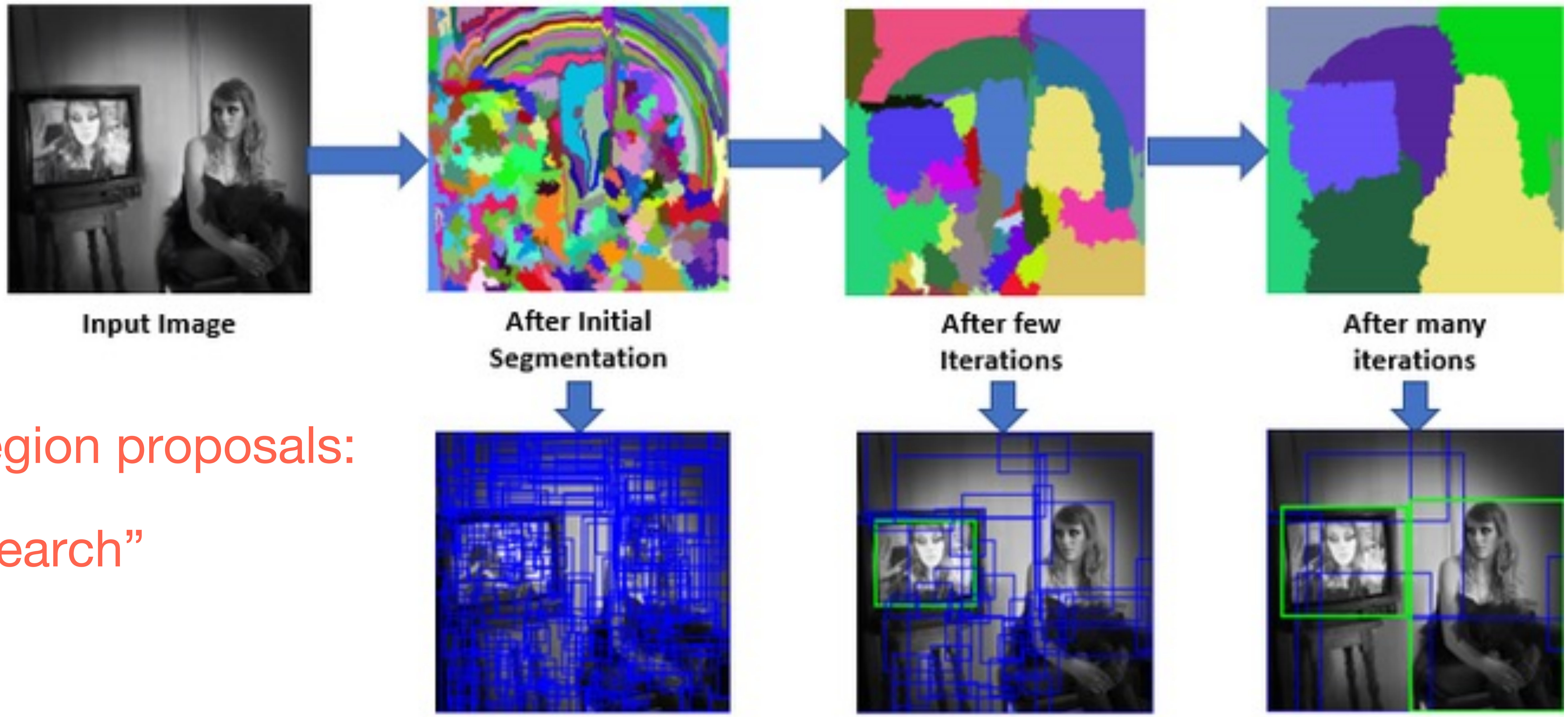
# R-CNN: Region-Based CNN

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




Generate region proposals:

“selective search”

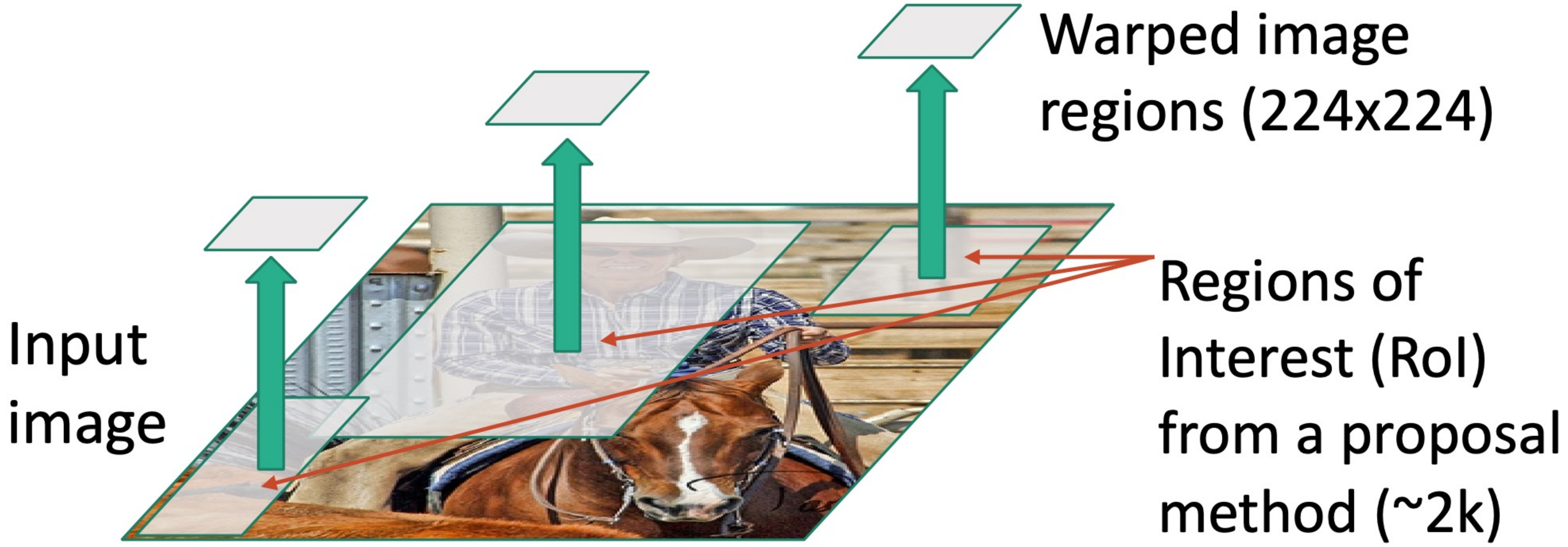
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

# R-CNN: Region-Based CNN

R-CNN: Region-Based CNN

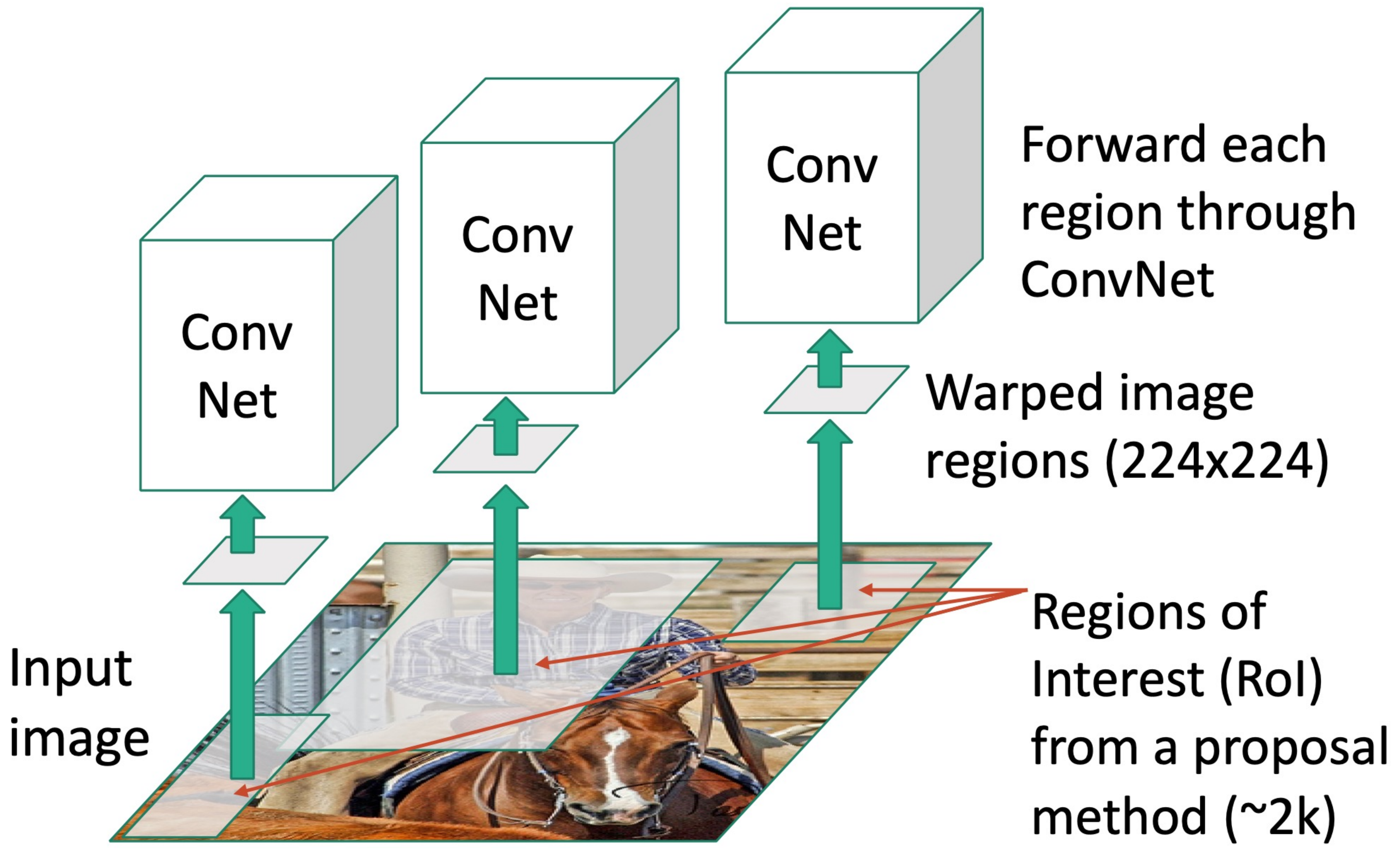
# R-CNN: Region-Based CNN

## R-CNN: Region-Based CNN



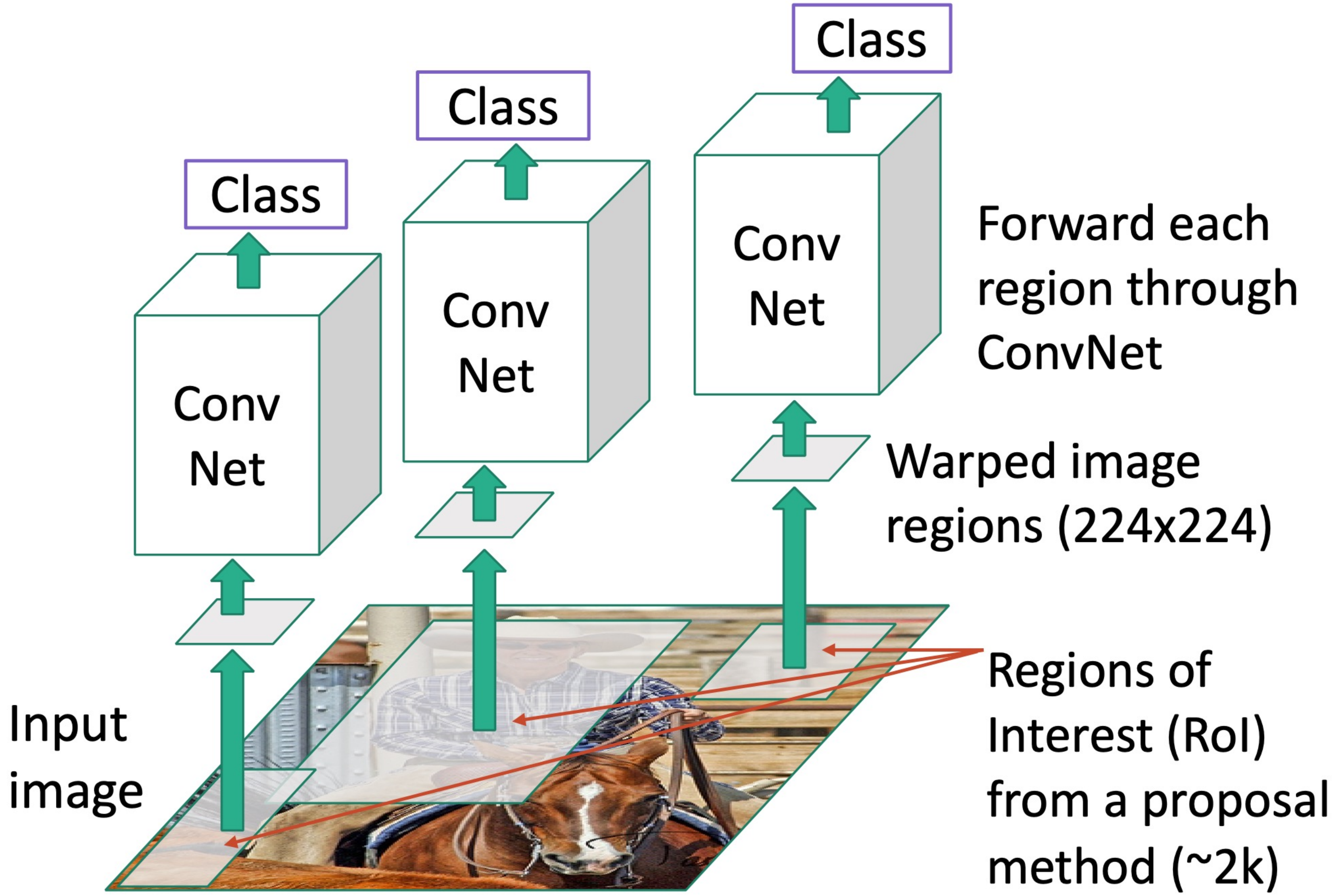
# R-CNN: Region-Based CNN

R-CNN: Region-Based CNN



# R-CNN: Region-Based CNN

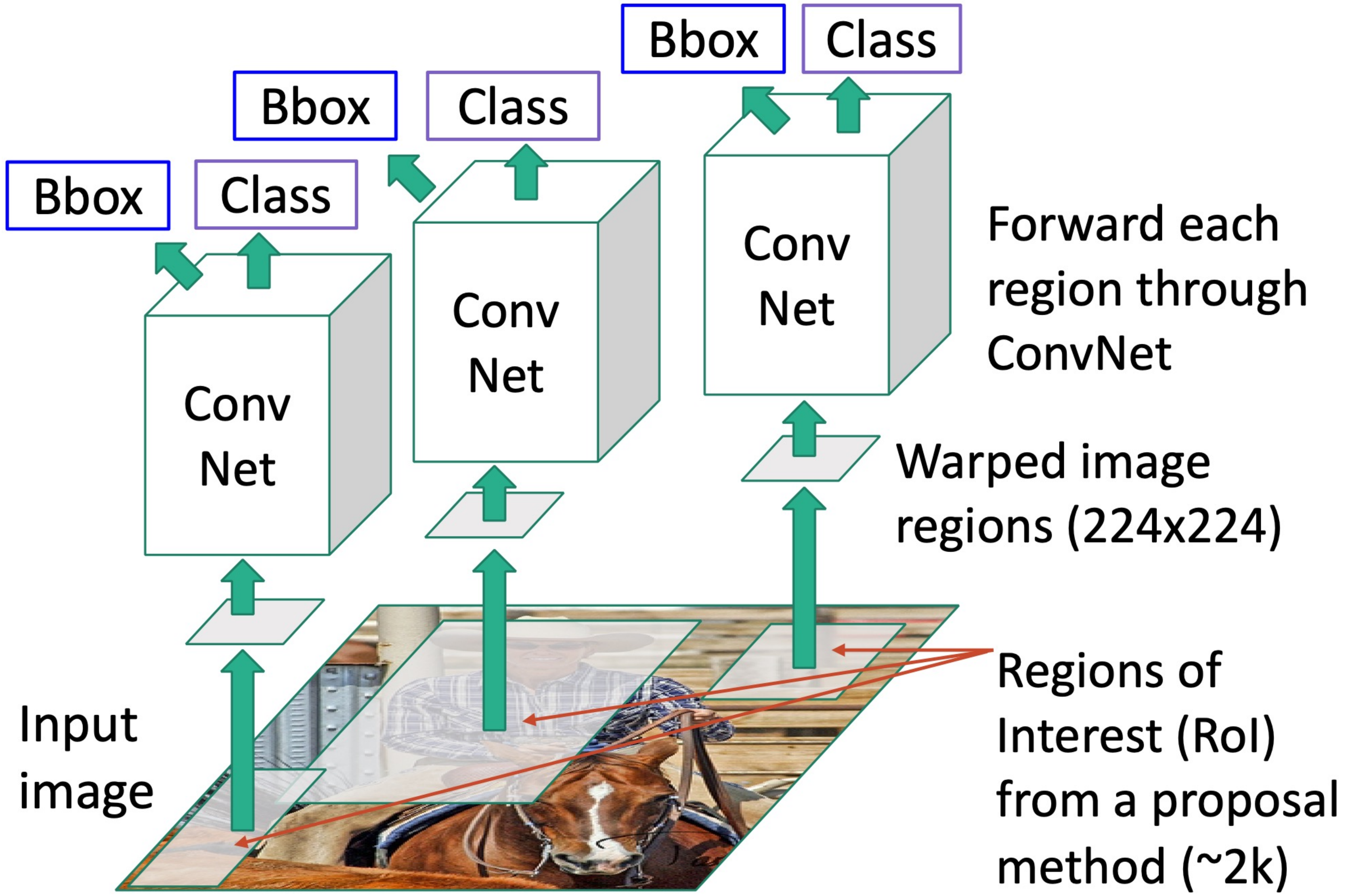
R-CNN: Region-Based CNN




Classify each region

# R-CNN: Region-Based CNN

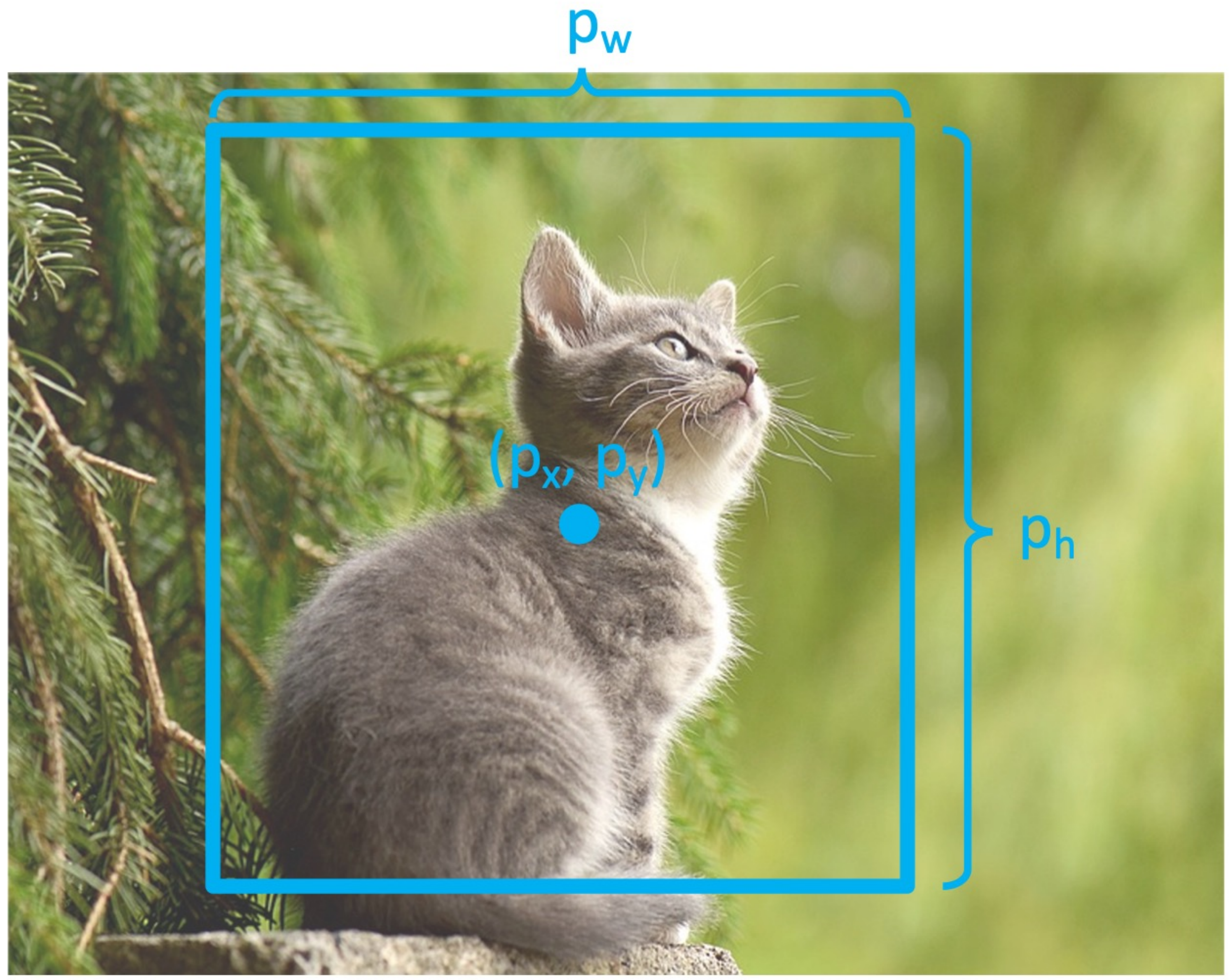
R-CNN: Region-Based CNN




Classify each region

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

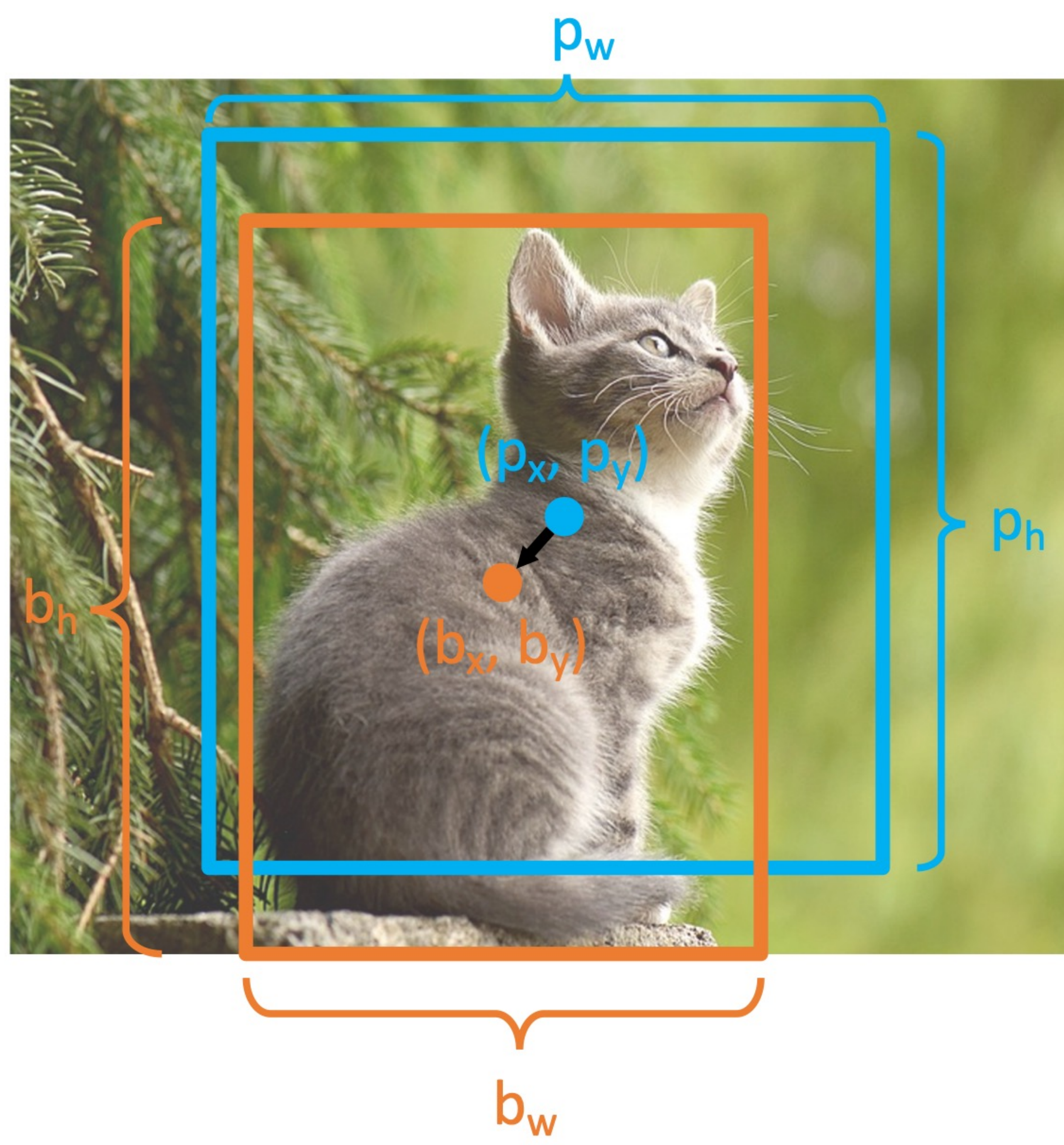
# R-CNN: Box Regression




Consider a region proposal with center $(p_x, p_y)$, width $p_w$, height $p_h$

Model predicts a transform $(t_x, t_y, t_w, t_h)$ to correct the region proposal

# R-CNN: Box Regression




Consider a region proposal with center $(p_x, p_y)$, width $p_w$, height $p_h$

Model predicts a transform $(t_x, t_y, t_w, t_h)$ to correct the region proposal

The output box is defined by:

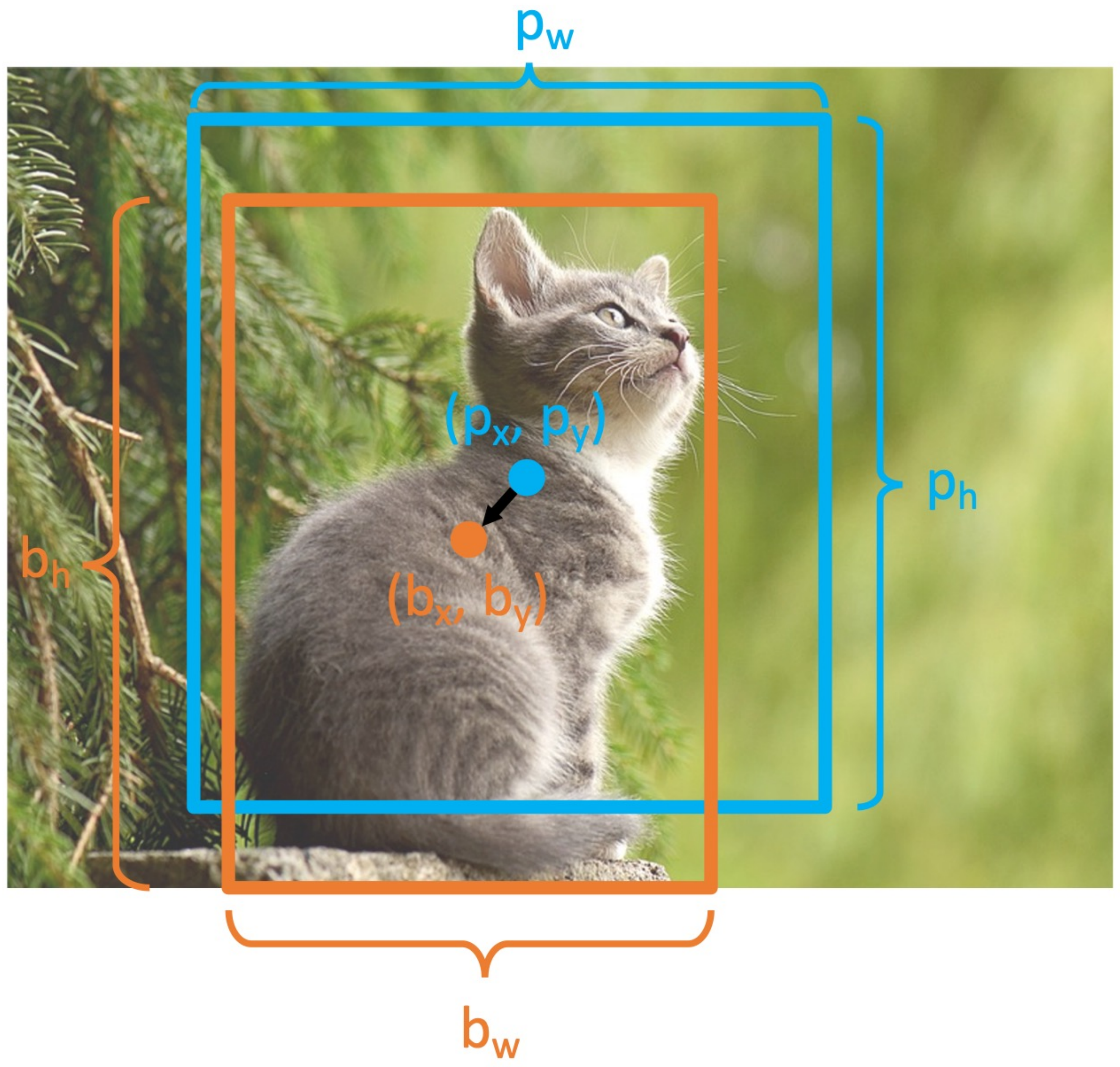
$$b_x = p_x + p_w t_x$$
$$b_y = p_y + p_h t_y$$

Shift center by amount relative to proposal size

$$b_w = p_w \exp(t_w)$$
$$b_h = p_h \exp(t_h)$$

Scale proposal; exp ensures that scaling factor is > 0

# R-CNN: Box Regression




Consider a region proposal with center $(p_x, p_y)$, width $p_w$, height $p_h$

Model predicts a transform $(t_x, t_y, t_w, t_h)$ to correct the region proposal

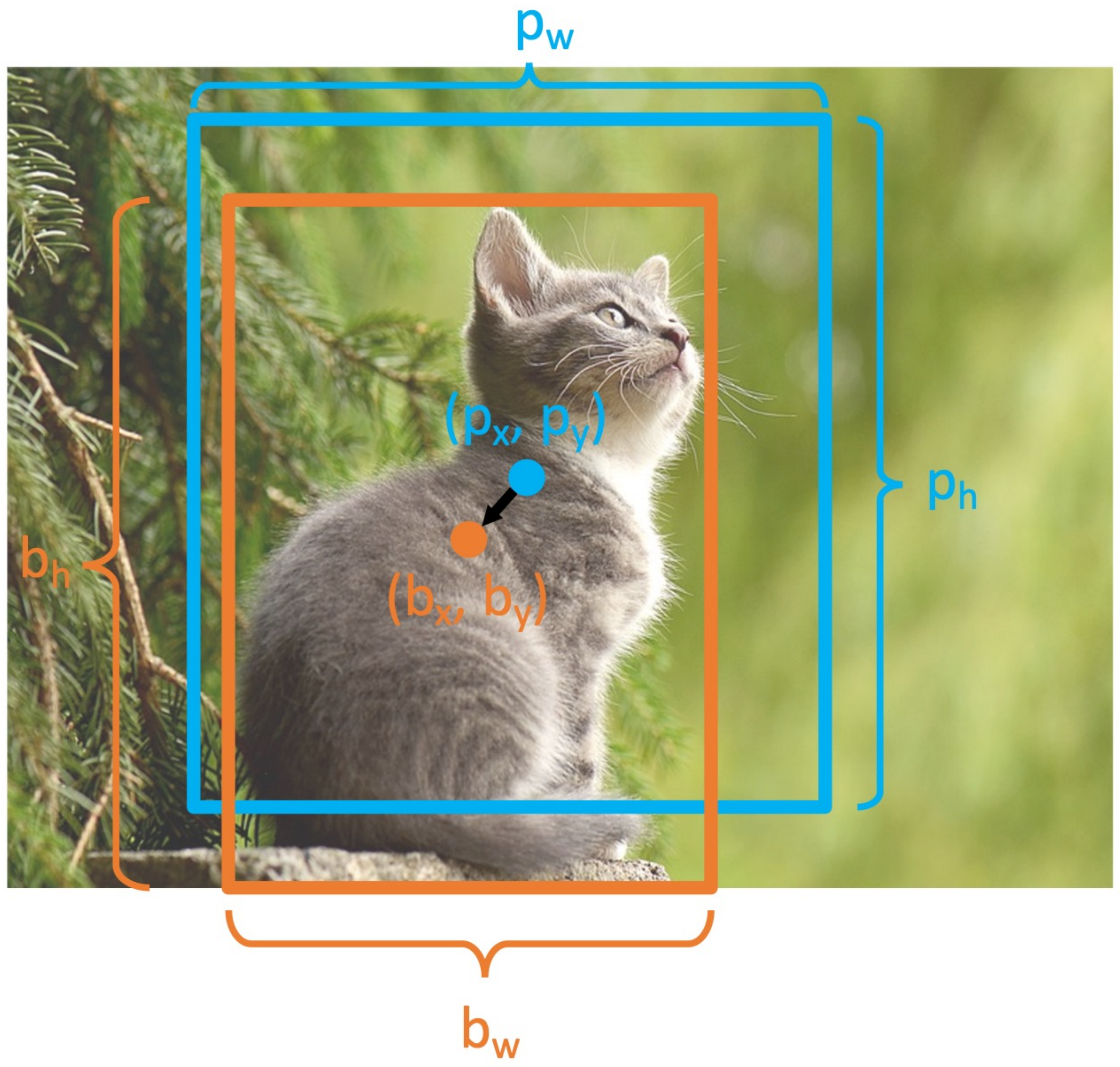
The output box is defined by:

$$b_x = p_x + p_w t_x$$
$$b_y = p_y + p_h t_y$$
$$b_w = p_w \exp(t_w)$$
$$b_h = p_h \exp(t_h)$$

When transform is 0, output = proposal

L2 regularization encourages leaving proposal unchanged

# R-CNN: Box Regression




Consider a region proposal with center $(p_x, p_y)$, width $p_w$, height $p_h$

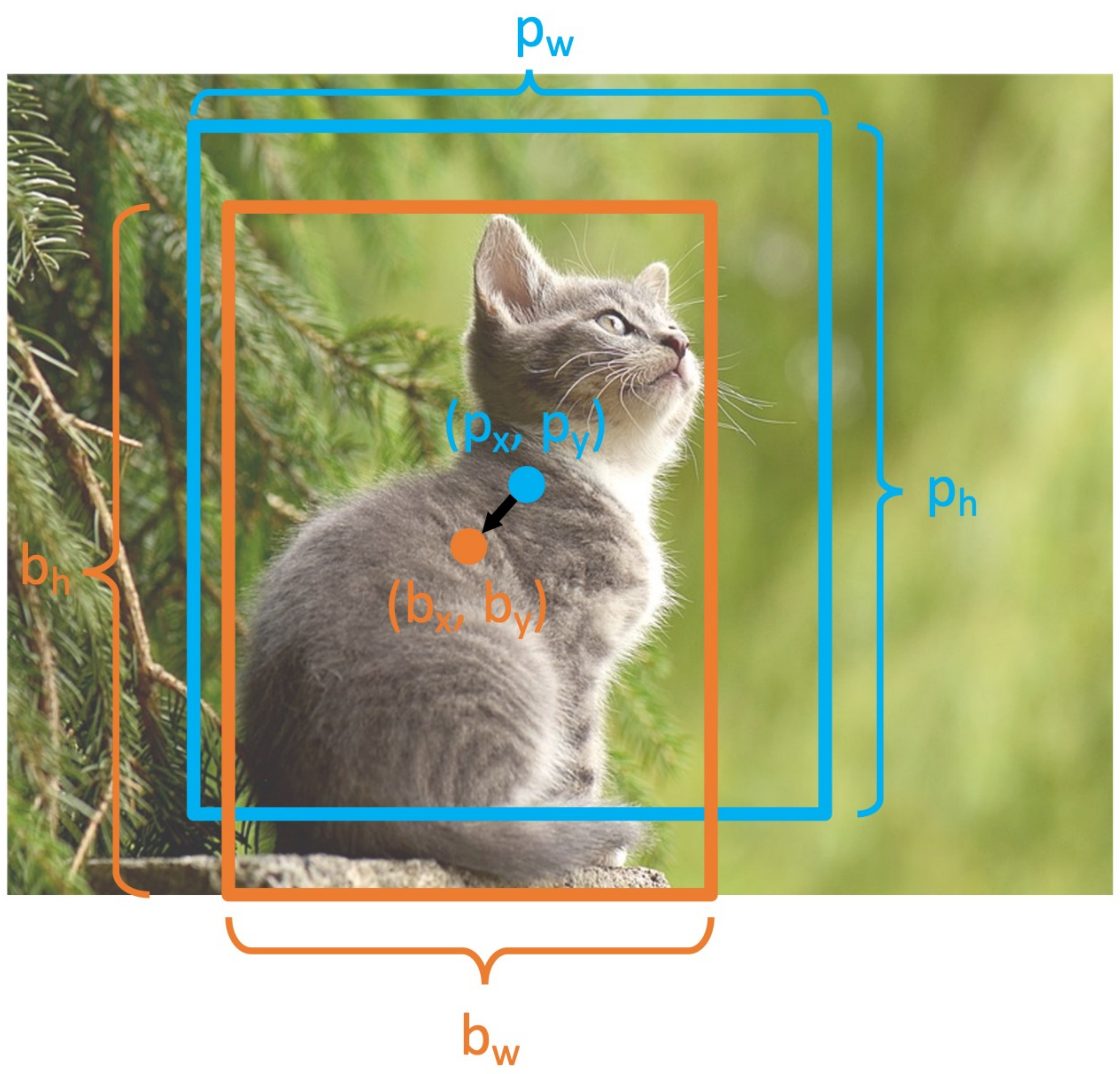
Model predicts a transform $(t_x, t_y, t_w, t_h)$ to correct the region proposal

The output box is defined by:

$$b_x = p_x + p_w t_x$$
$$b_y = p_y + p_h t_y$$
$$b_w = p_w \exp(t_w)$$
$$b_h = p_h \exp(t_h)$$

Scale / Translation invariance: Transform encodes *relative* difference between proposal and output; important since CNN doesn't see absolute size or position after cropping

# R-CNN: Box Regression




Consider a region proposal with center $(p_x, p_y)$, width $p_w$, height $p_h$

Model predicts a transform $(t_x, t_y, t_w, t_h)$ to correct the region proposal

The output box is defined by:

$$b_x = p_x + p_w t_x$$
$$b_y = p_y + p_h t_y$$
$$b_w = p_w \exp(t_w)$$
$$b_h = p_h \exp(t_h)$$

Given proposal and target output, we can solve for the transform the network should output:

$$t_x = (b_x - p_x)/p_w$$
$$t_y = (b_y - p_y)/p_h$$
$$t_w = \log(b_w/p_w)$$
$$t_h = \log(b_h/p_h)$$

# R-CNN: Training

**Input Image**


Ground Truth

# R-CNN: Training

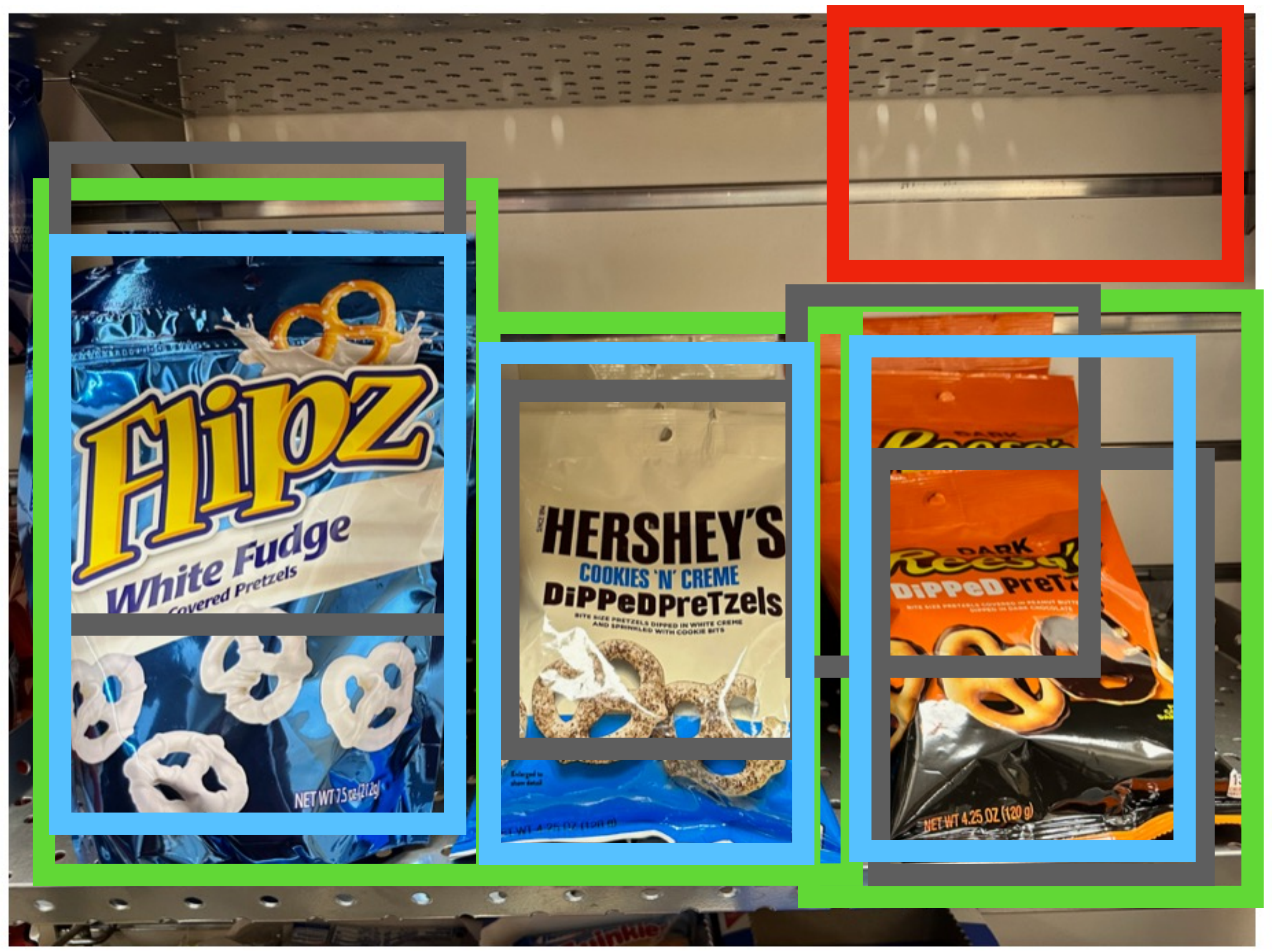
**Input Image**


Ground Truth

Region Proposals

# R-CNN: Training

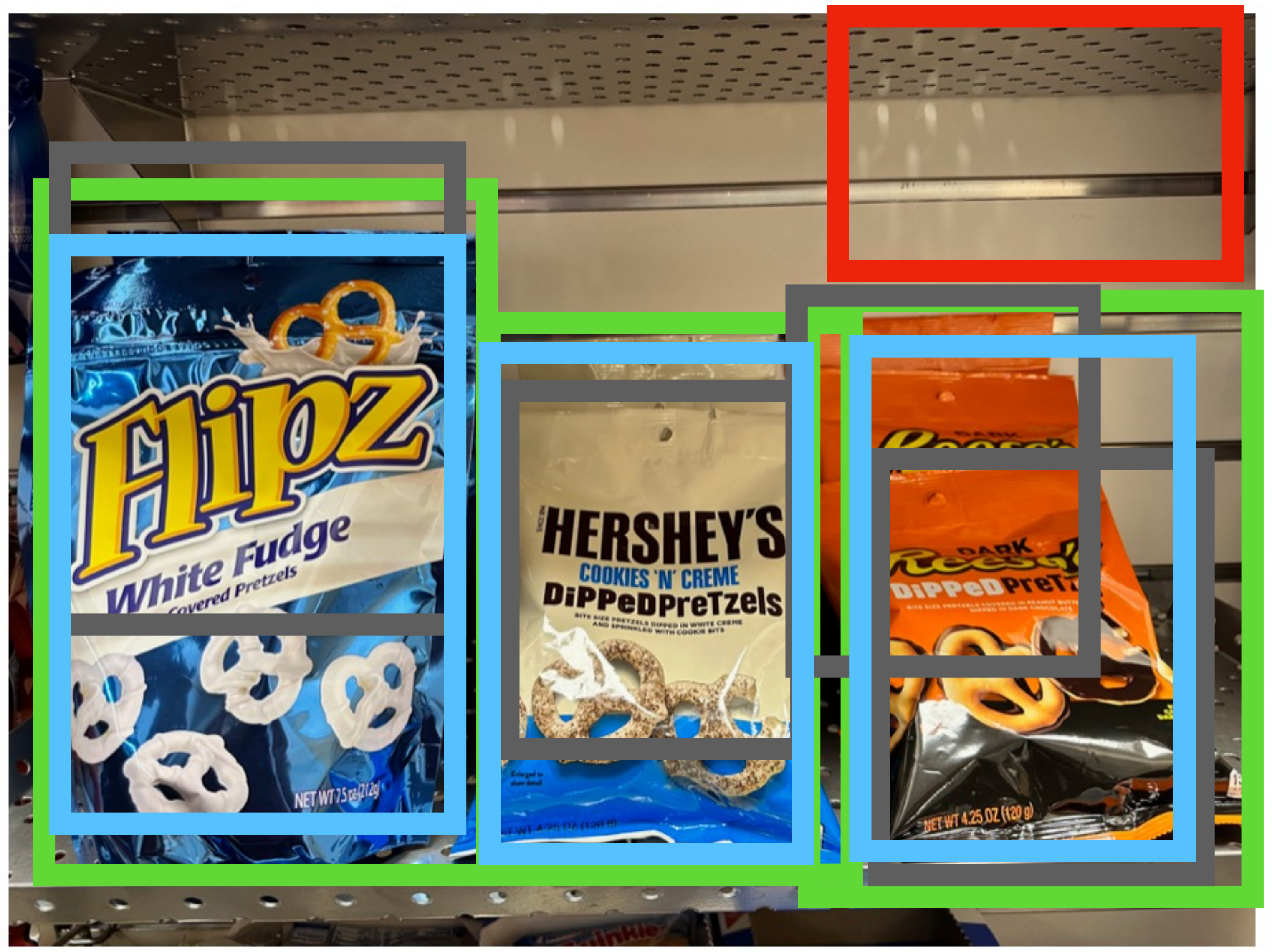
**Input Image**




| | |
|---|---|
| Ground Truth | Positive |
| Neutral | Negative |

# R-CNN: Training

**Input Image**




| | |
|---|---|
| Ground Truth | Positive |
| Neutral | Negative |

Categorize each region proposal as positive, negative or neutral based on overlap with the Ground truth boxes:

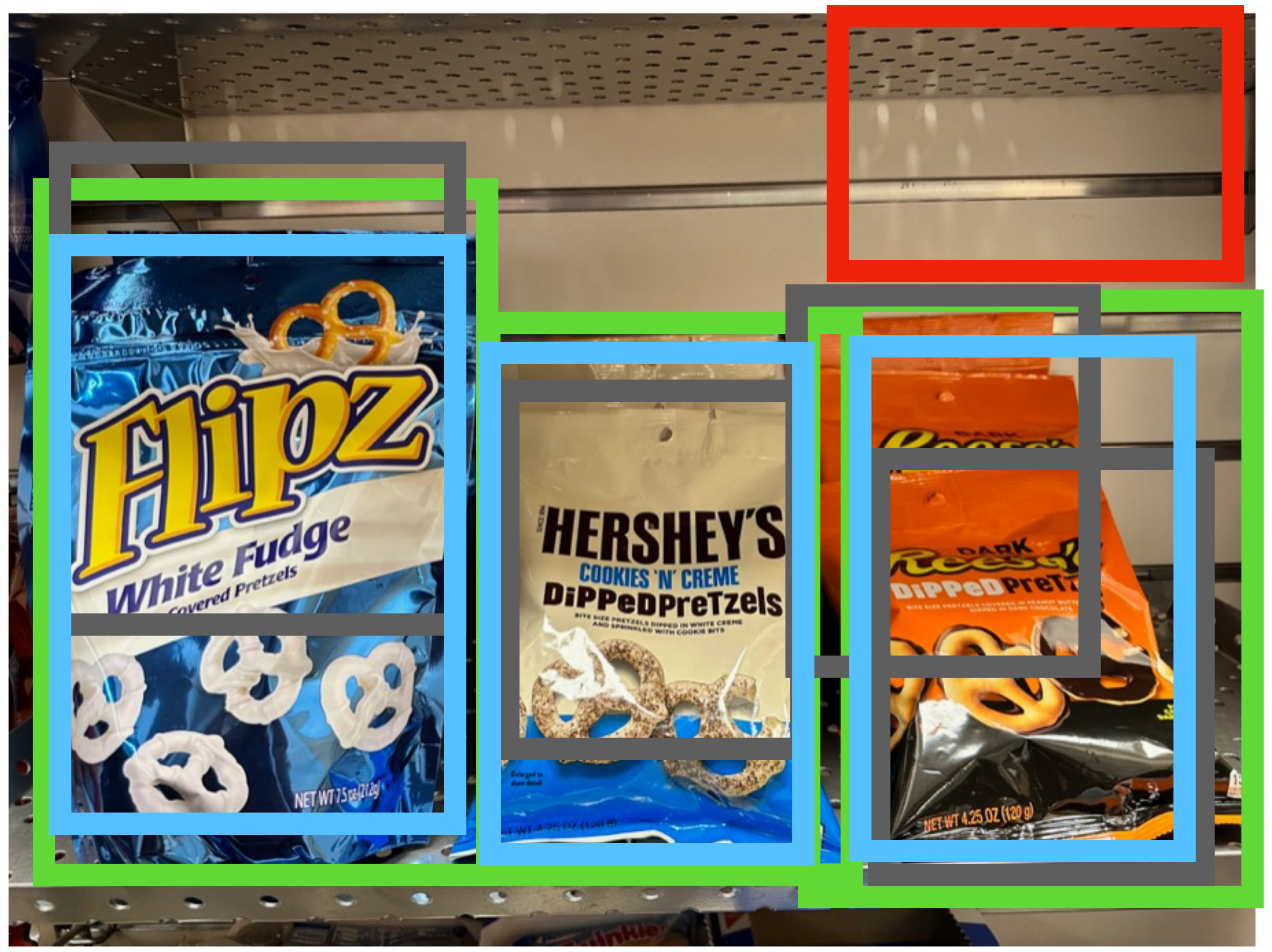
Positive: > 0.5 IoU with a GT box

Negative: < 0.3 IoU with all GT boxes

Neutral: between 0.3 and 0.5 IoU with GT boxes

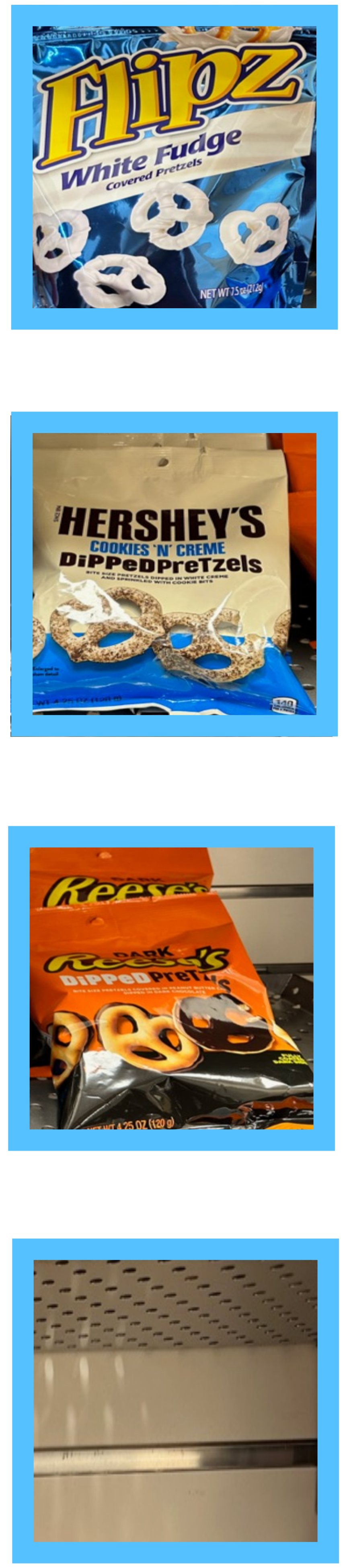
# R-CNN: Training

**Input Image**




Ground Truth | Positive

Neutral | Negative

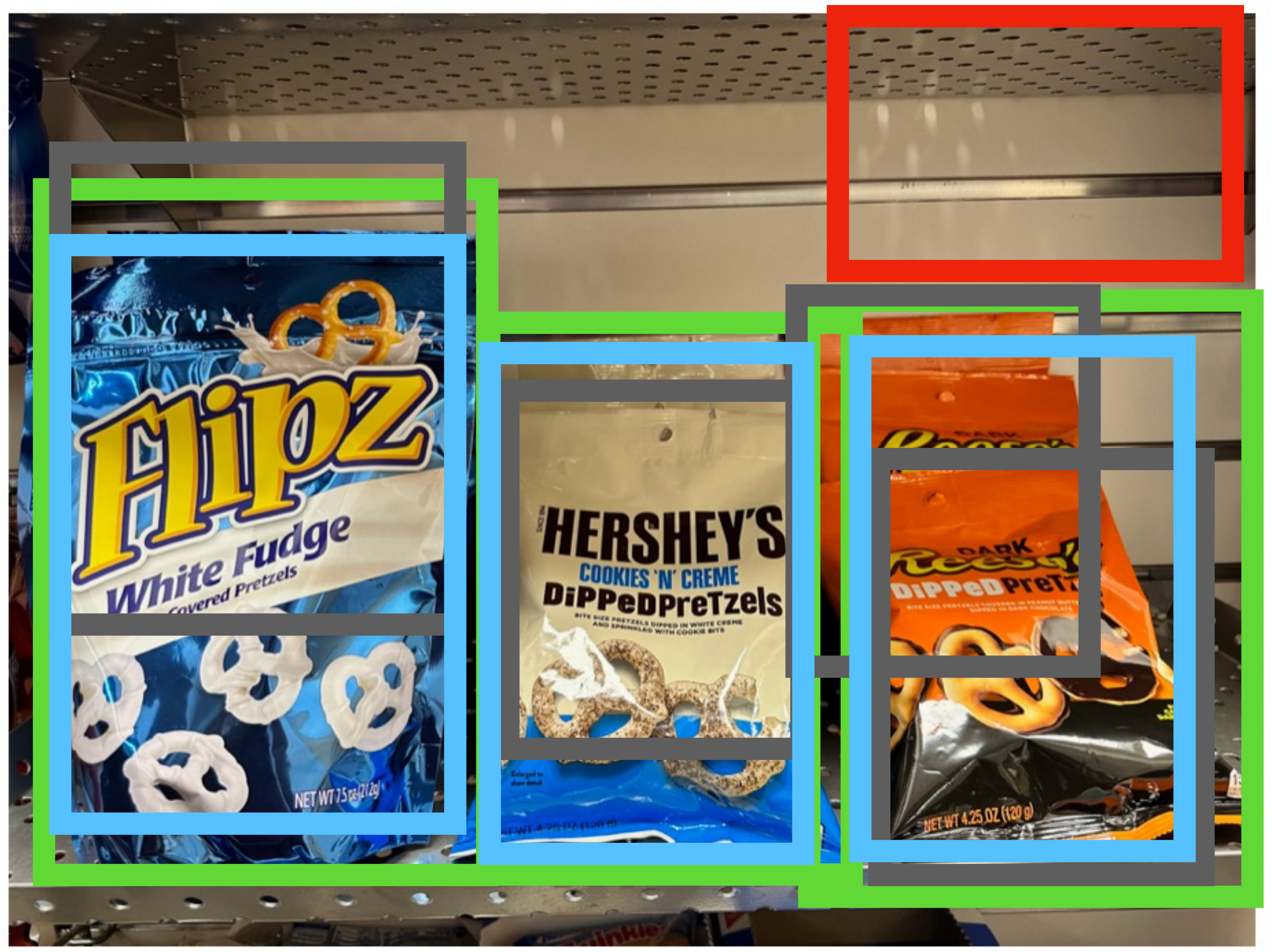



Crop pixels from each positive and negative proposal, resize to 224 x 224

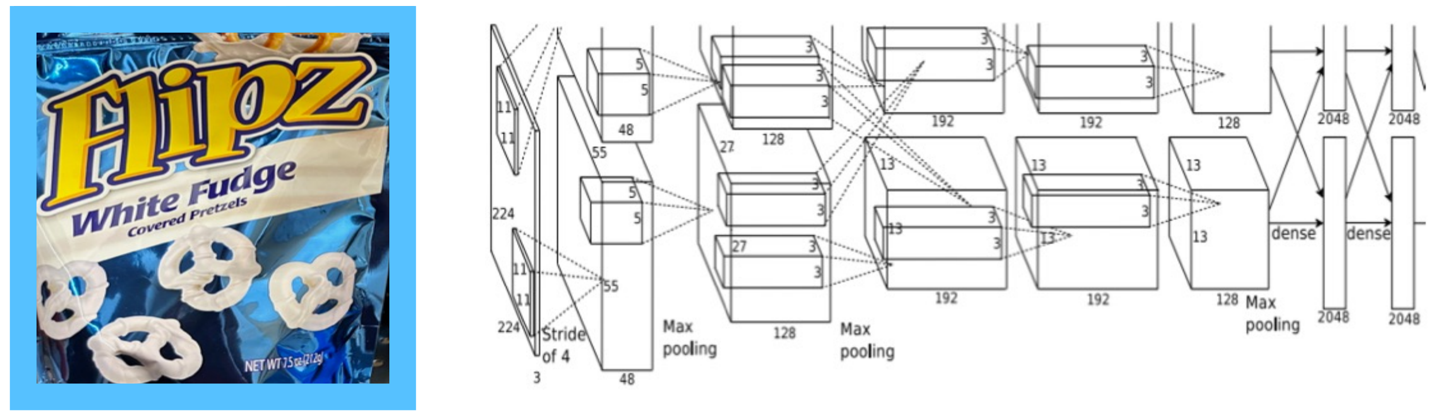
Run each region through CNN
Positive regions: predict class and transform
Negative regions: just predict class

# R-CNN: Training

**Input Image**

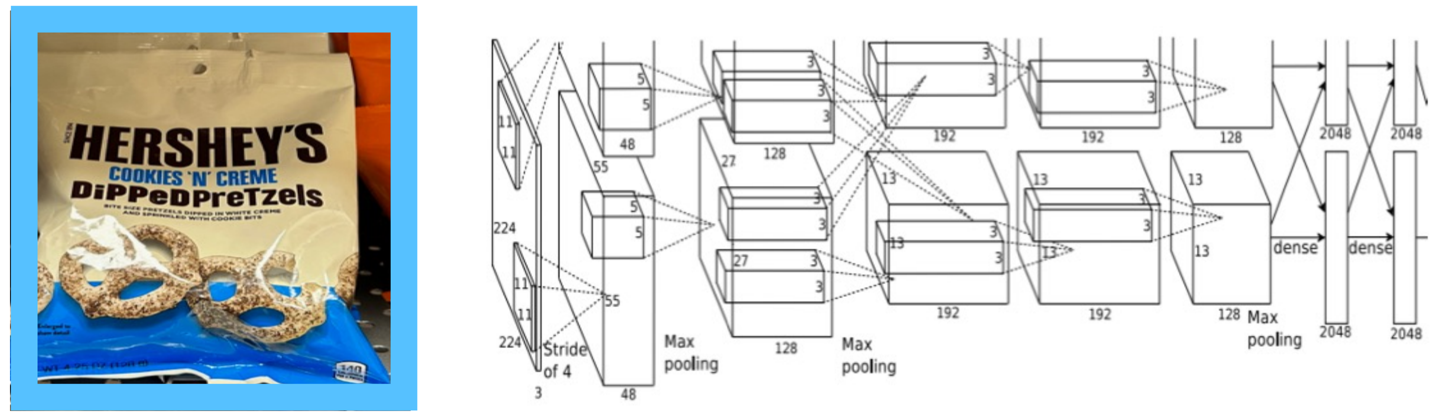



| | |
|---|---|
| Ground Truth | Positive |
| Neutral | Negative |



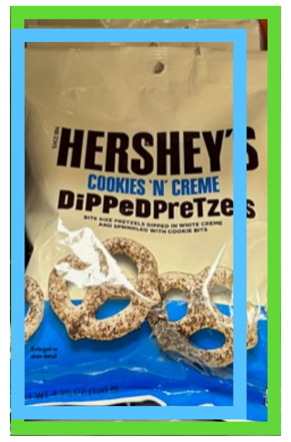
**Class target:** Flipz

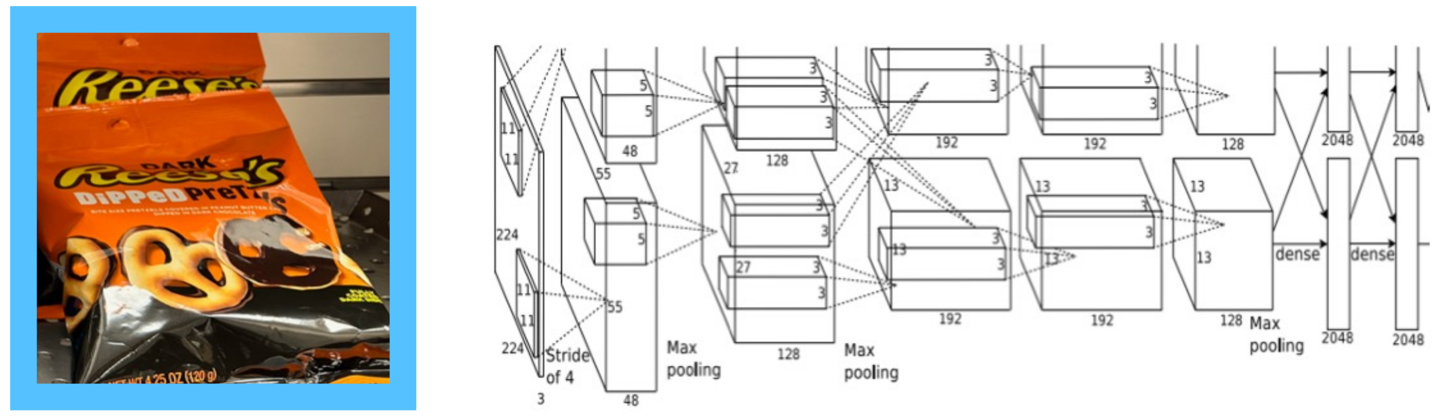
**Box target:** →



**Class target:** Hershey's

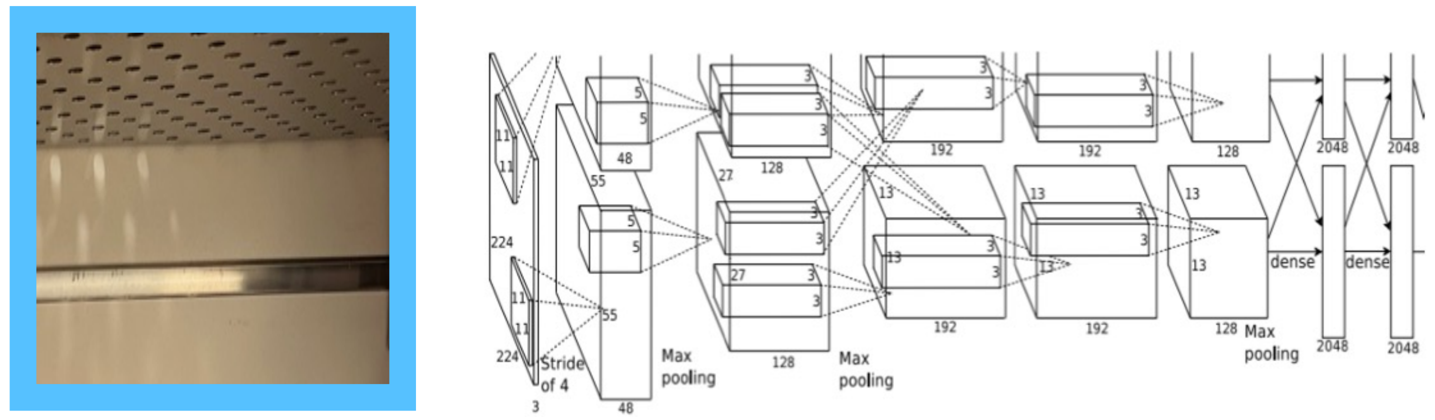
**Box target:** →





**Class target:** Reese's

**Box target:** →



**Class target:** Background

**Box target:** None

Run each region through CNN
Positive regions: predict class and transform
Negative regions: just predict class

# R-CNN: Test time

**Input Image**


Region Proposals

**<u>Run proposal method:</u>**

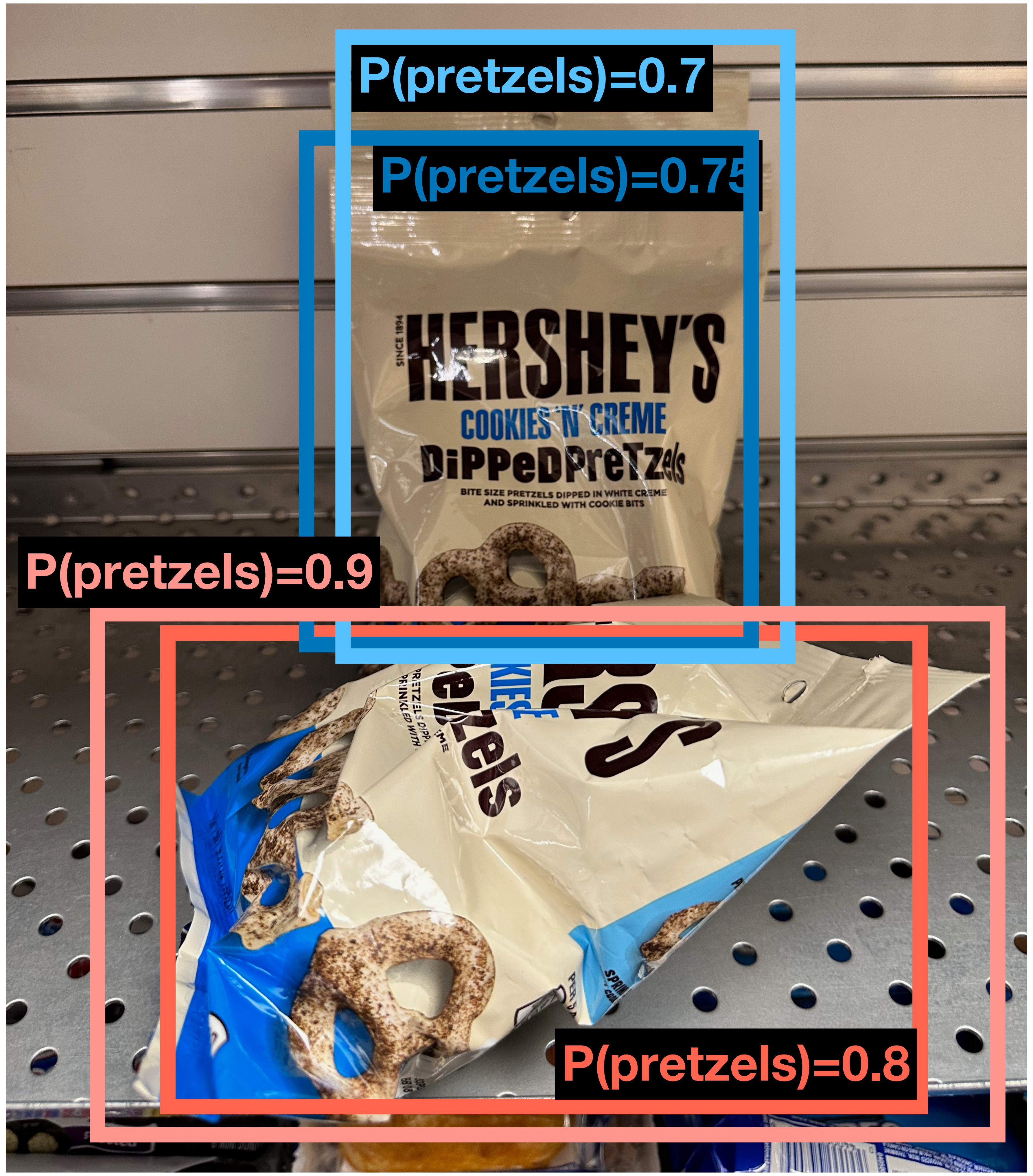
1. Run CNN on each proposal to get class scores, transforms
2. Threshold class scores to get a set of detections

**2 Problems:**

1. CNN often outputs overlapping boxes
2. How to set thresholds?

# Overlapping Boxes

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



# Overlapping Boxes: Non-Max Suppression (NMS)

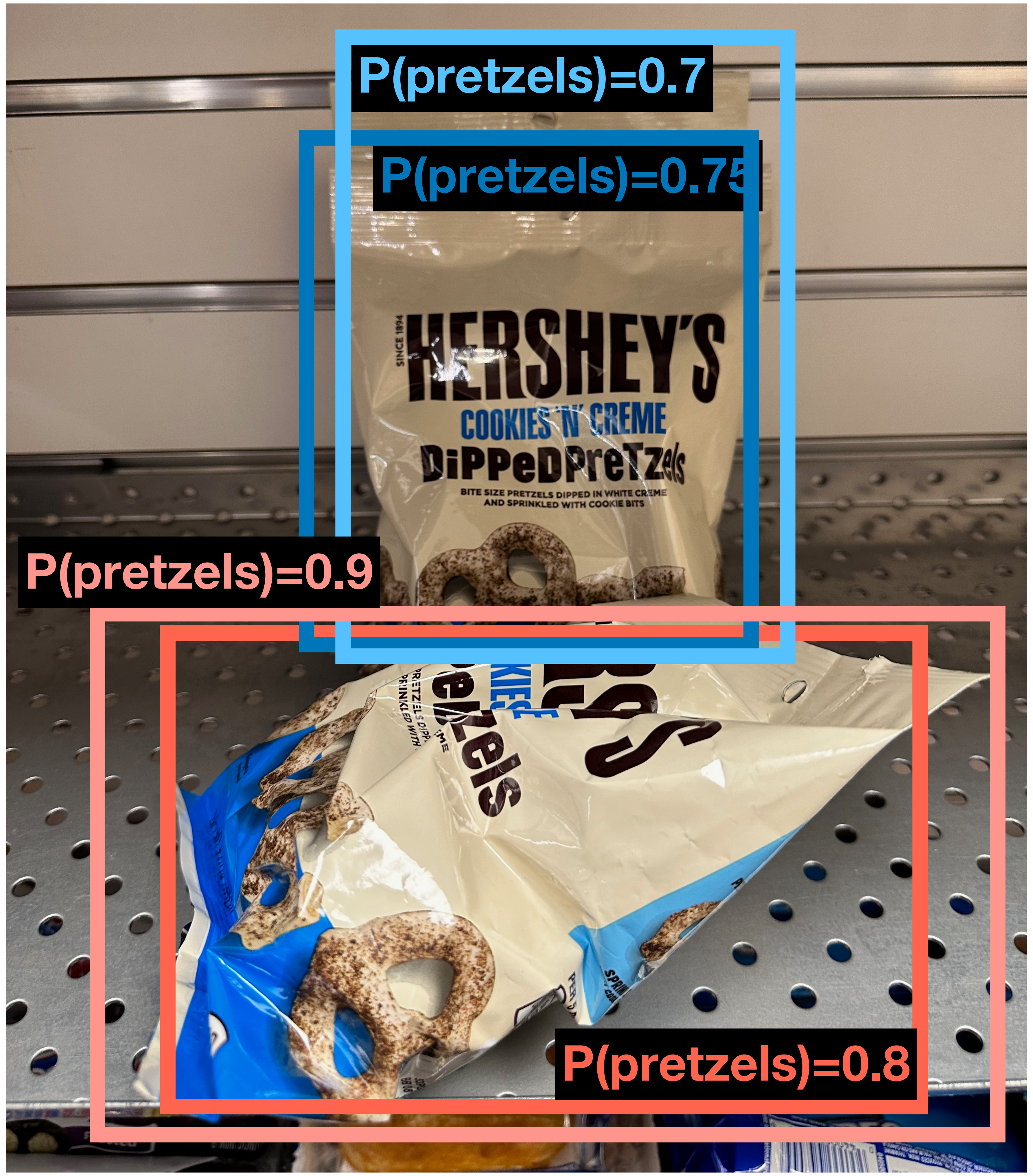
**Problem:** Object detectors often output many overlapping detections

**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



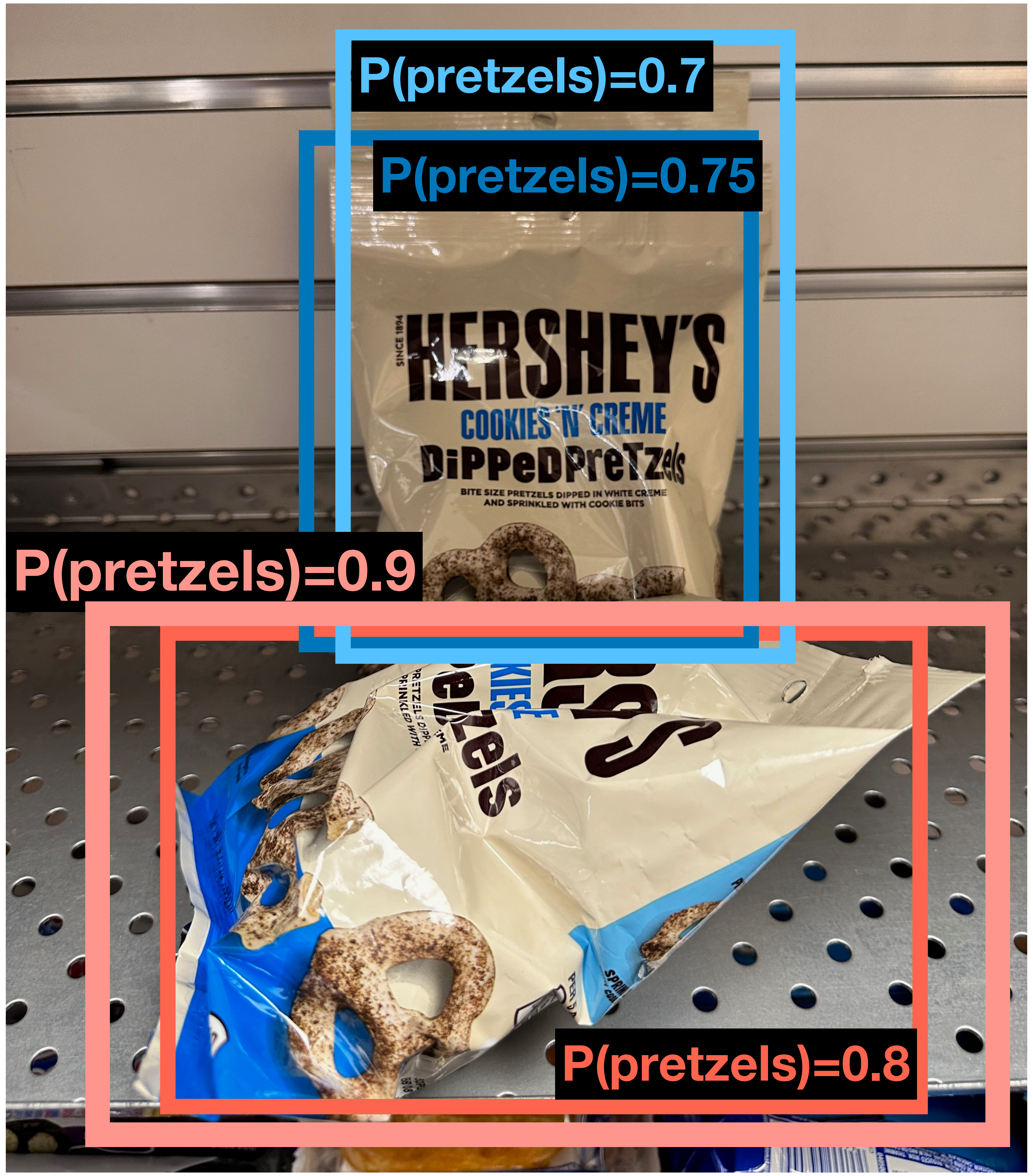
# Overlapping Boxes: Non-Max Suppression (NMS)

**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( , ) = 0.8
IoU( , ) = 0.03
IoU( , ) = 0.05



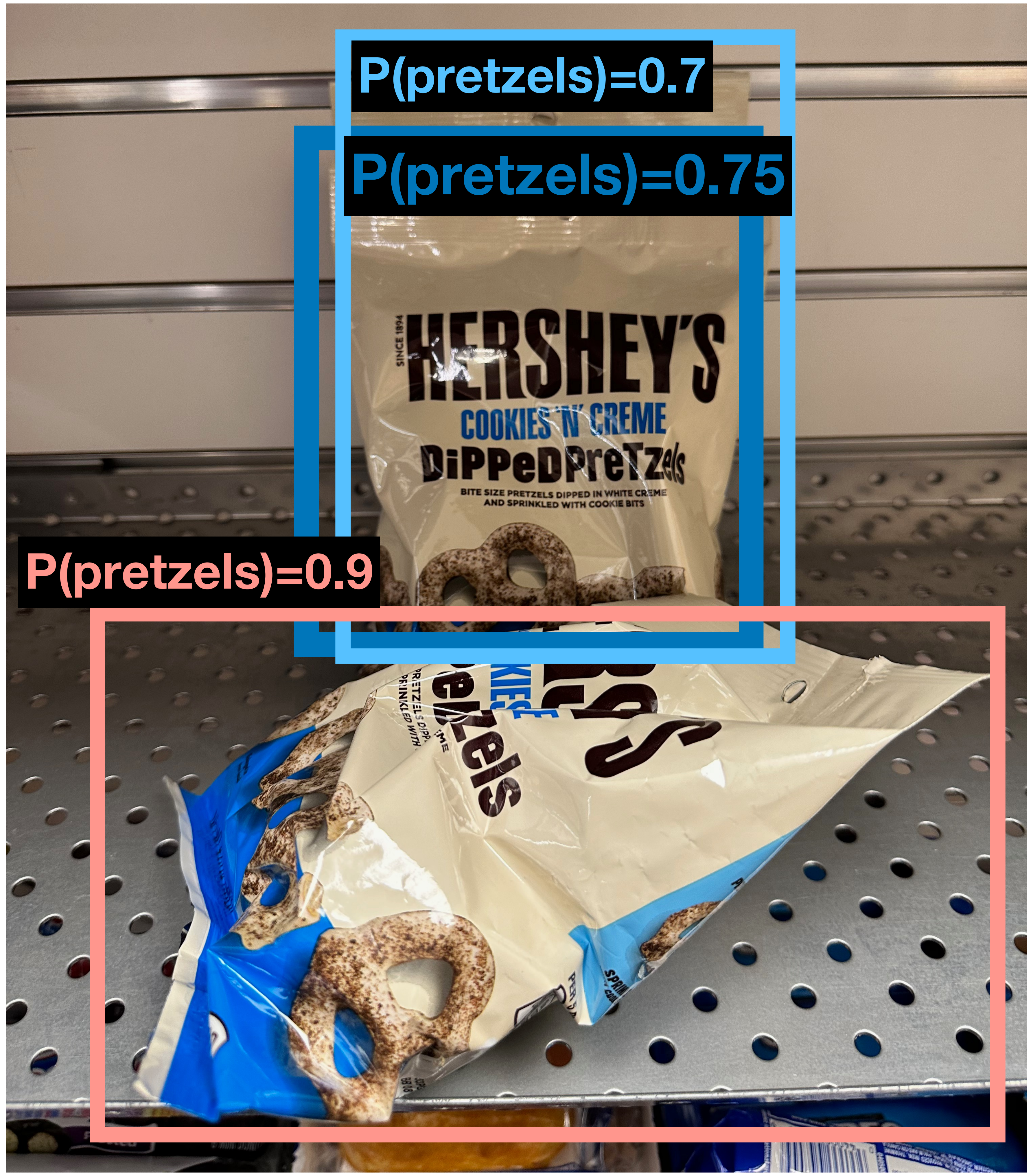
# Overlapping Boxes: Non-Max Suppression (NMS)

**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( , ) = 0.85



# Overlapping Boxes: Non-Max Suppression (NMS)

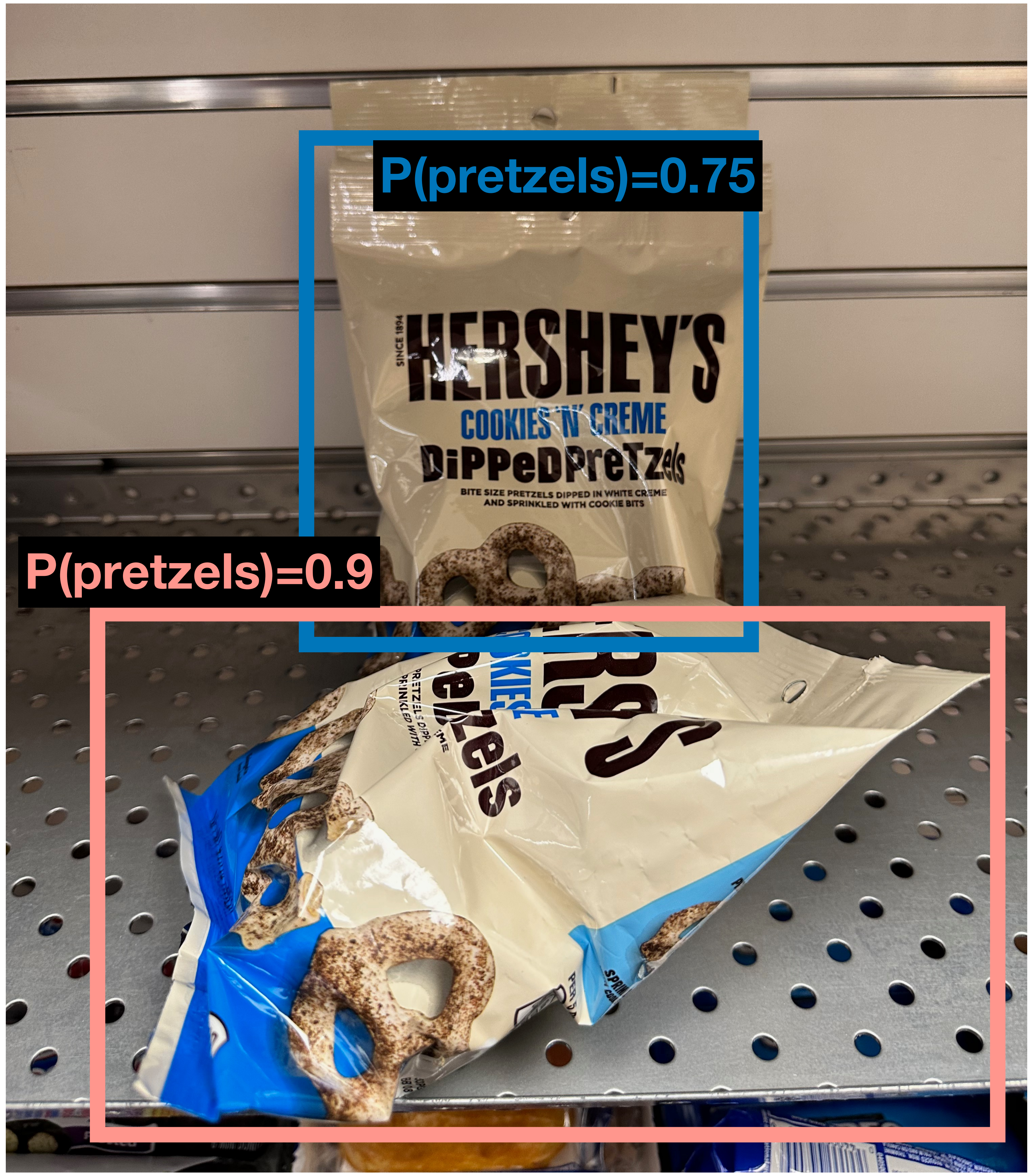
**Problem:** Object detectors often output many overlapping detections

**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

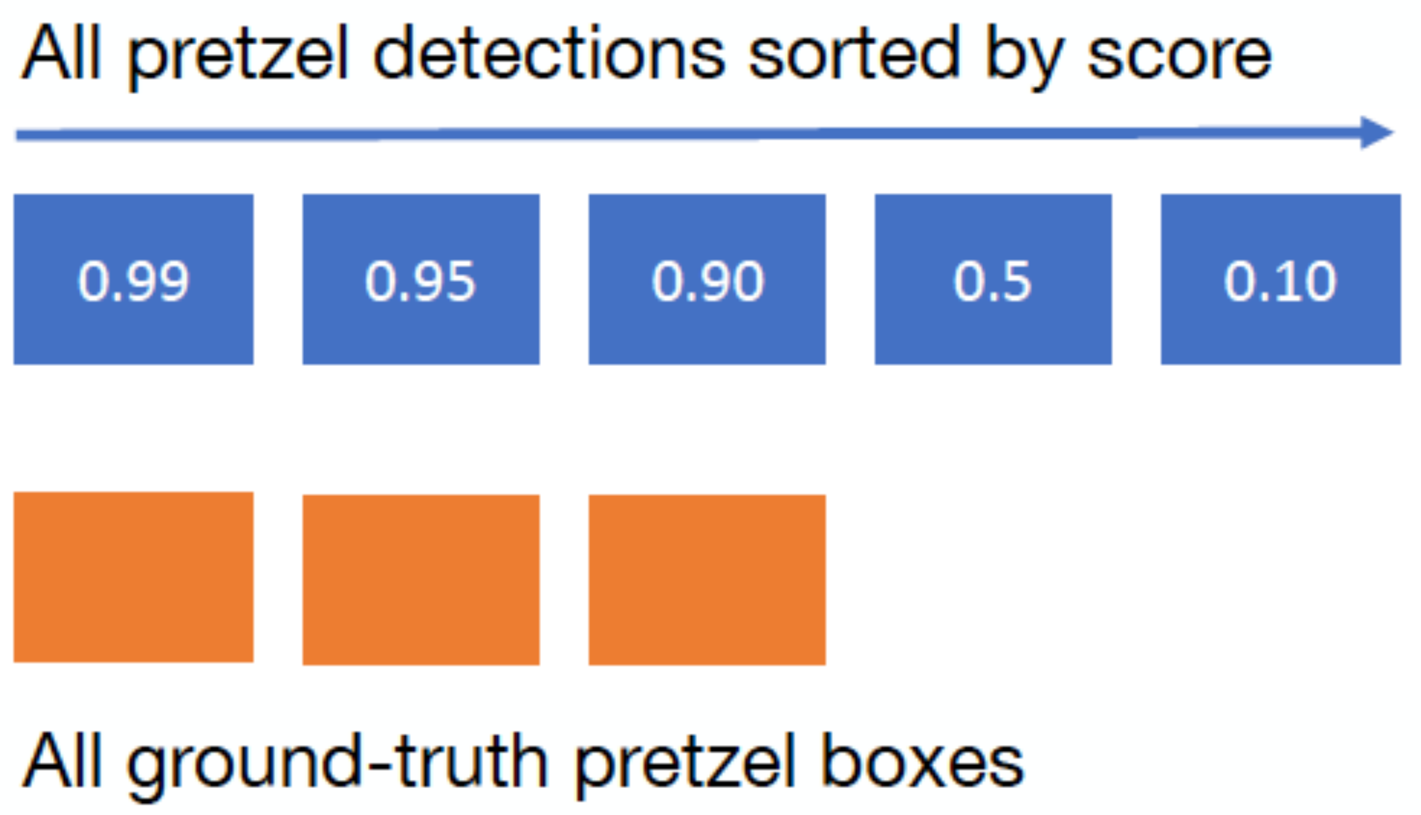
**Problem:** NMS may eliminate "good" boxes when objects are highly overlapping… no good solution

# Evaluating Object Detectors: Mean Average Precision (mAP) and P-R Curve

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

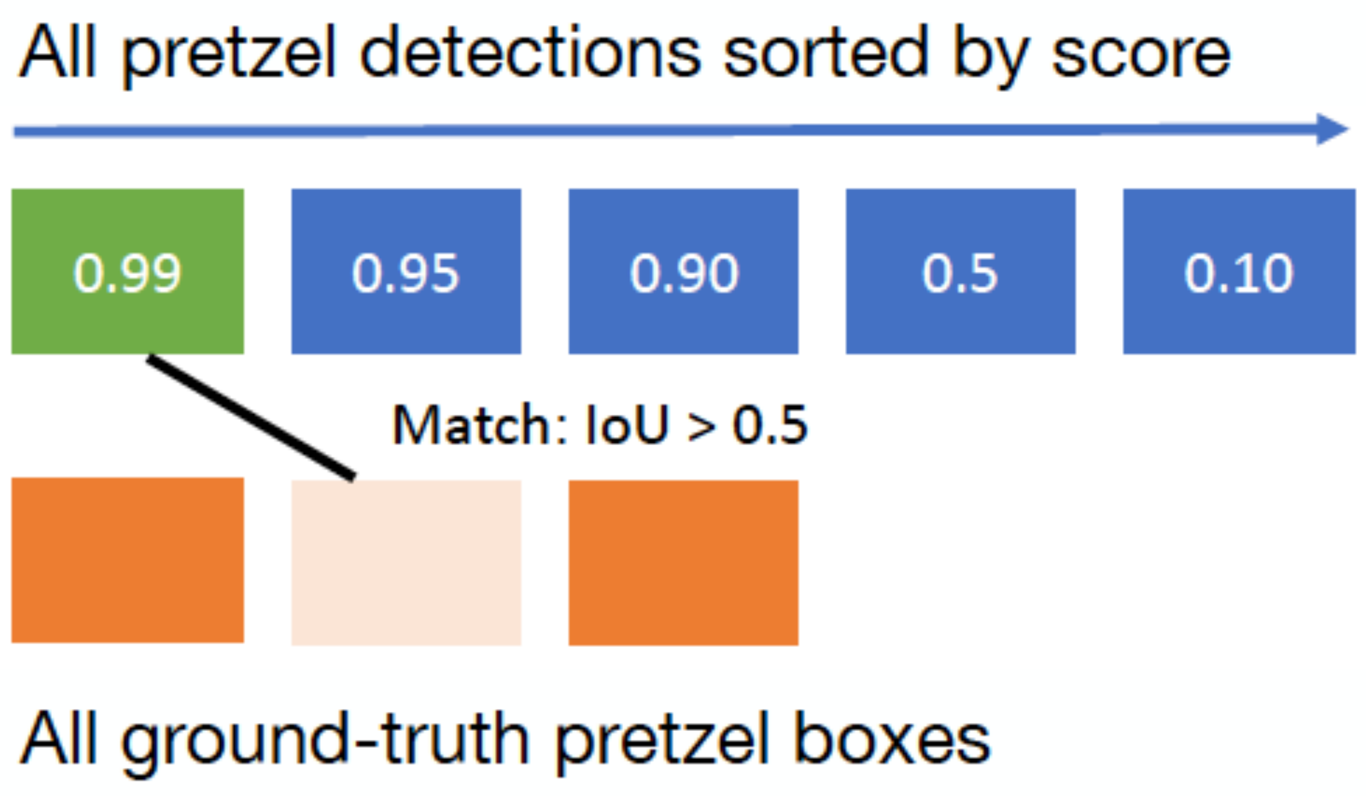
# Evaluating Object Detectors: Mean Average Precision (mAP) and P-R Curve

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)



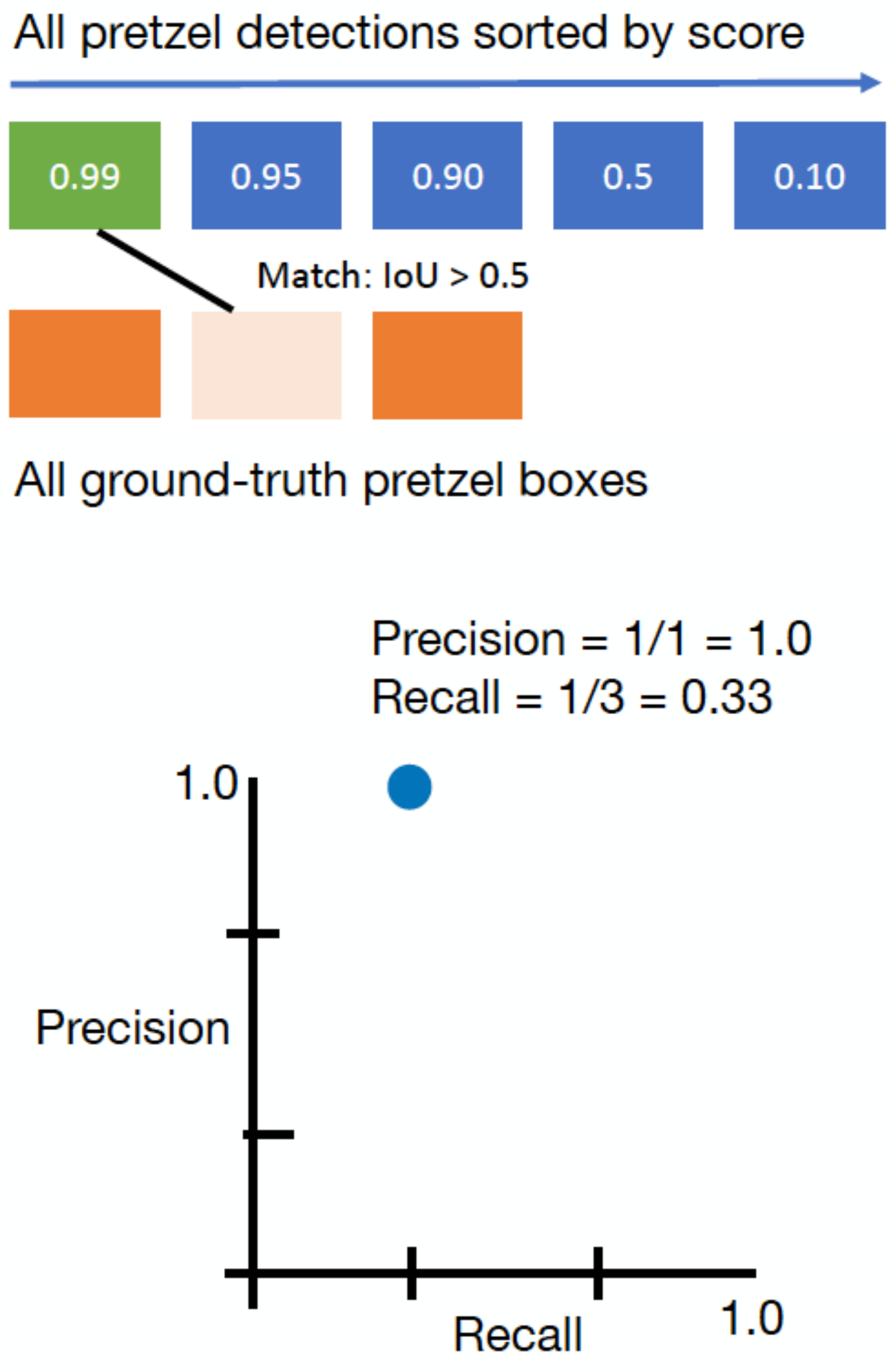
# Evaluating Object Detectors: Mean Average Precision (mAP) and P-R Curve

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) and P-R Curve

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

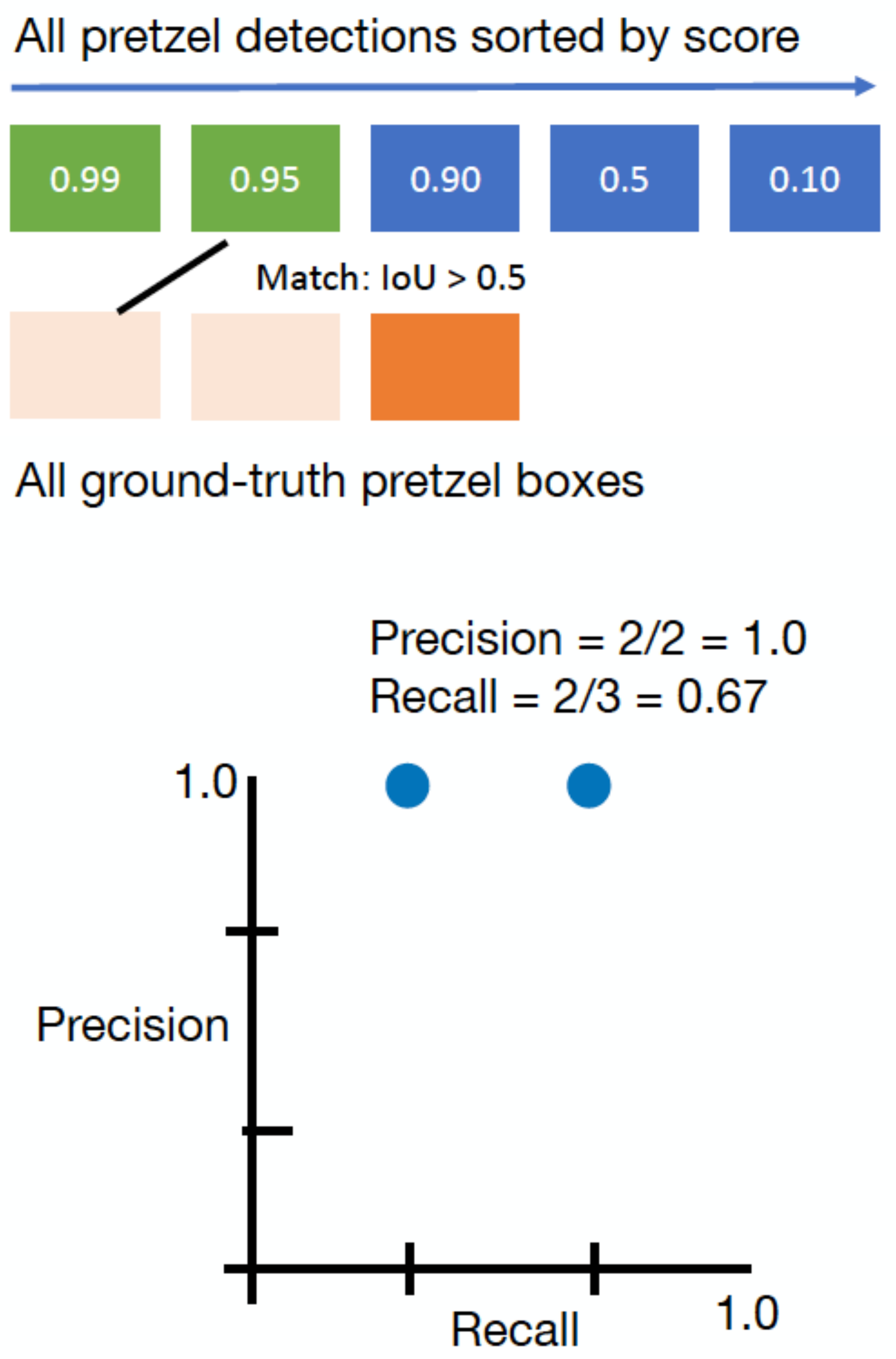



$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

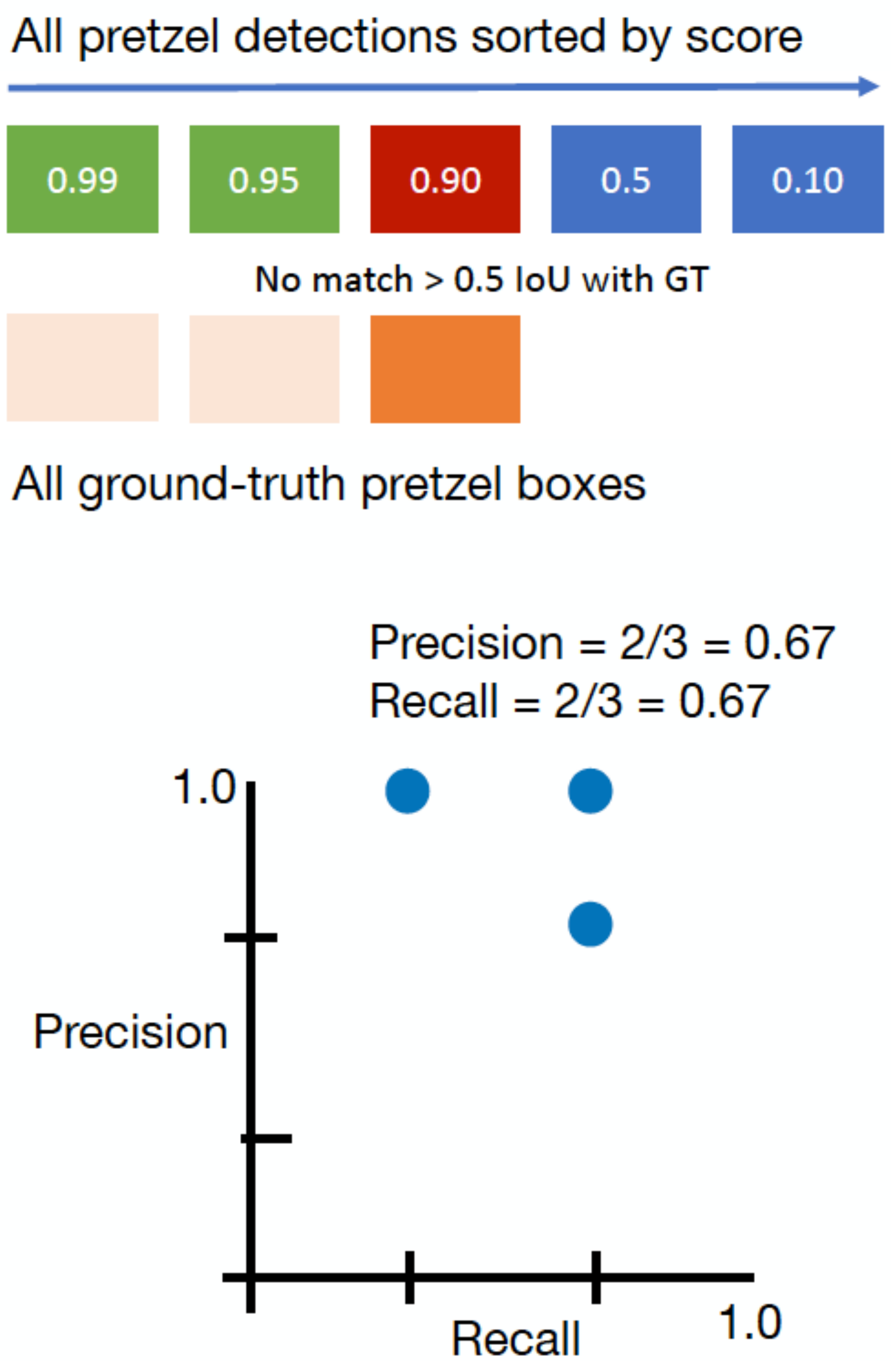
# Evaluating Object Detectors: Mean Average Precision (mAP) and P-R Curve

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



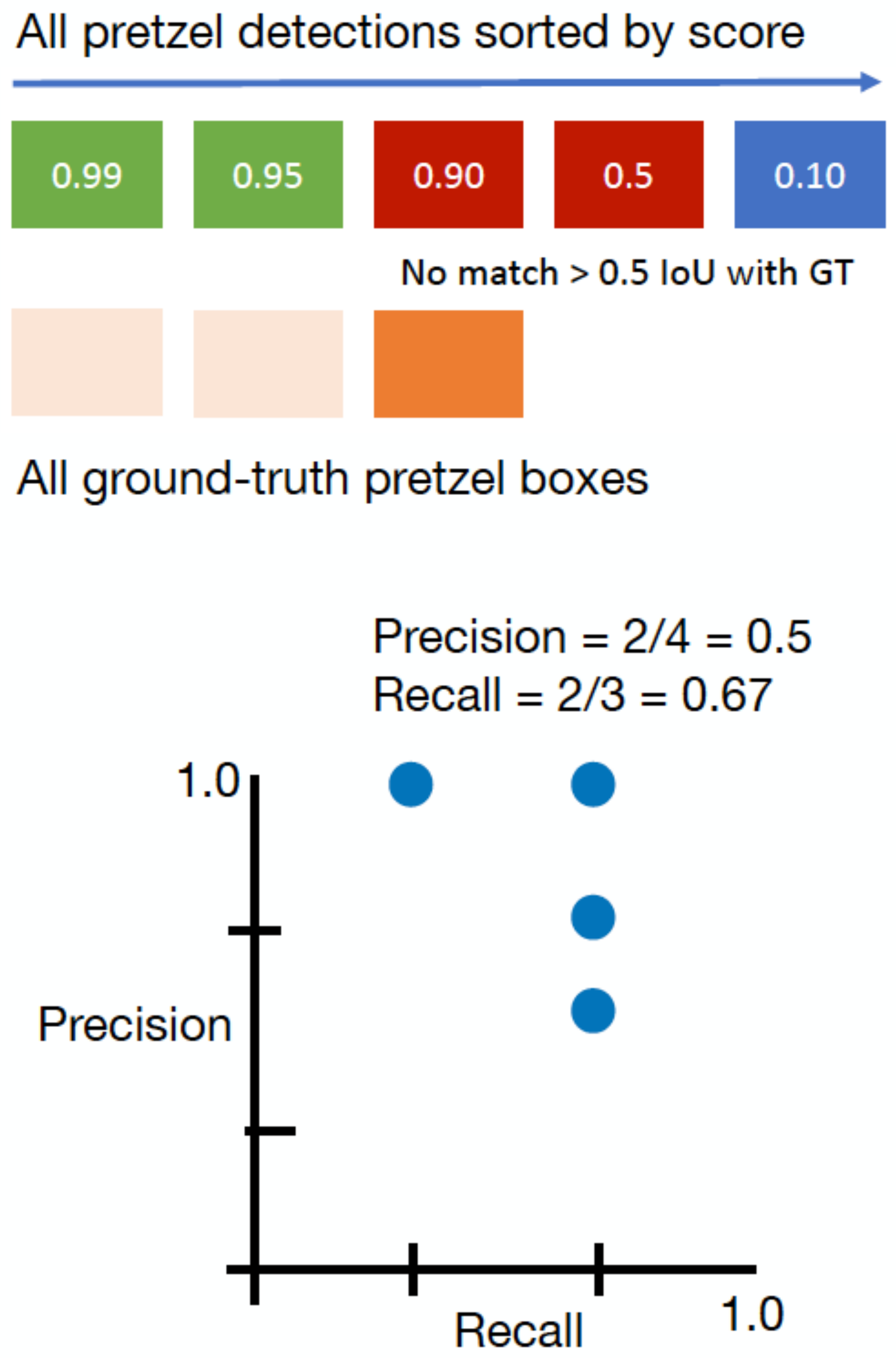
# Evaluating Object Detectors: Mean Average Precision (mAP) and P-R Curve

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



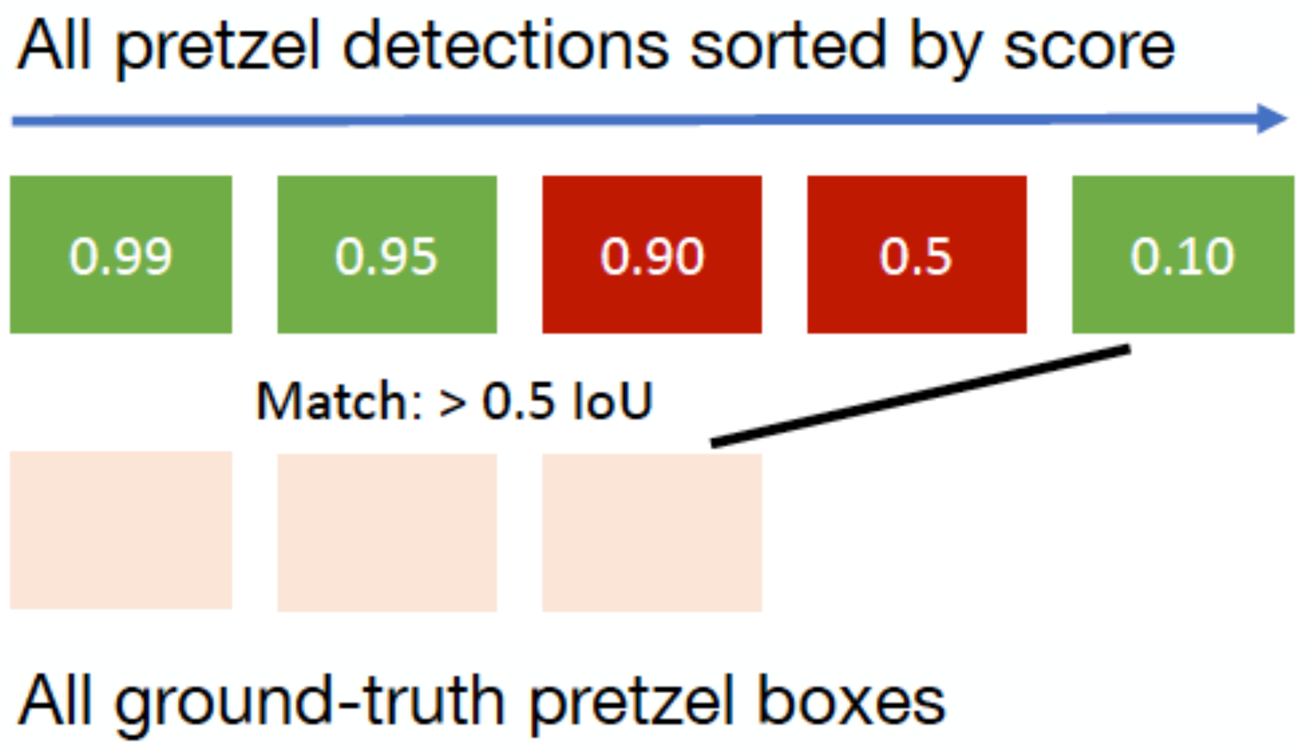
# Evaluating Object Detectors: Mean Average Precision (mAP) and P-R Curve

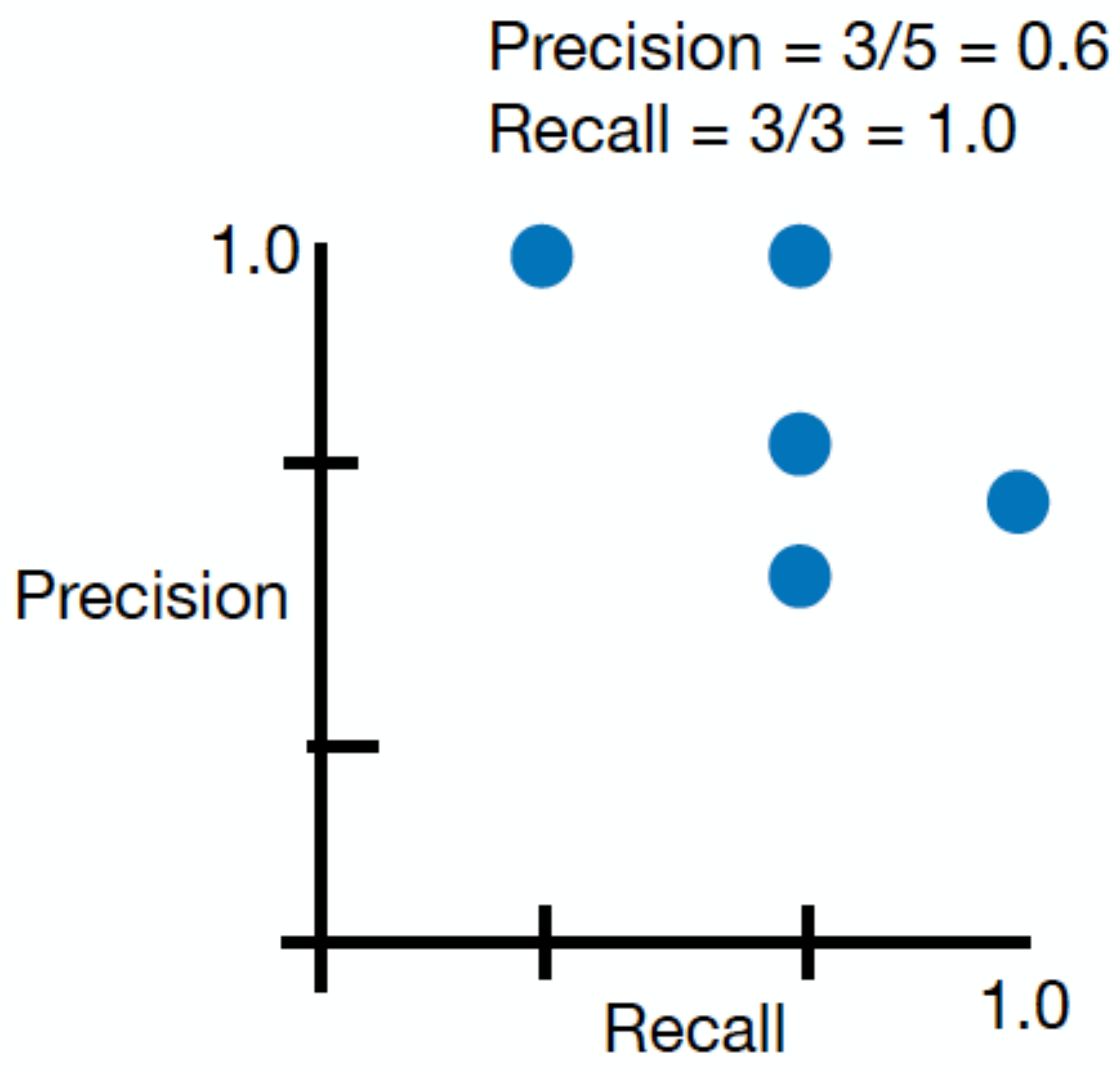
1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
    1. For each detection (highest score to lowest score)
        1. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
   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



# Evaluating Object Detectors: Mean Average Precision (mAP) and P-R Curve

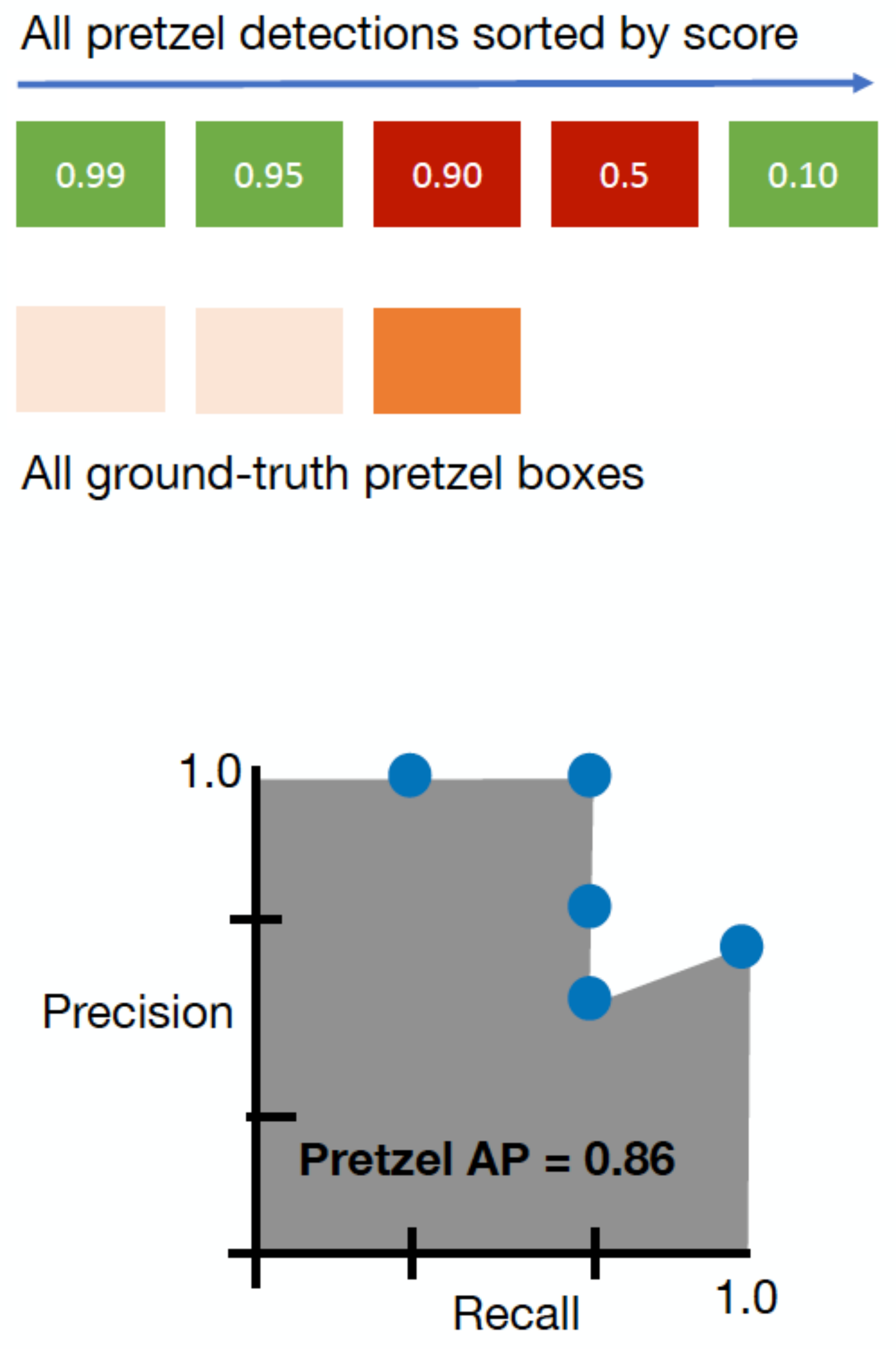
1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
   1. For each detection (highest score to lowest score)
      1. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
      2.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



# Evaluating Object Detectors: Mean Average Precision (mAP) and P-R Curve

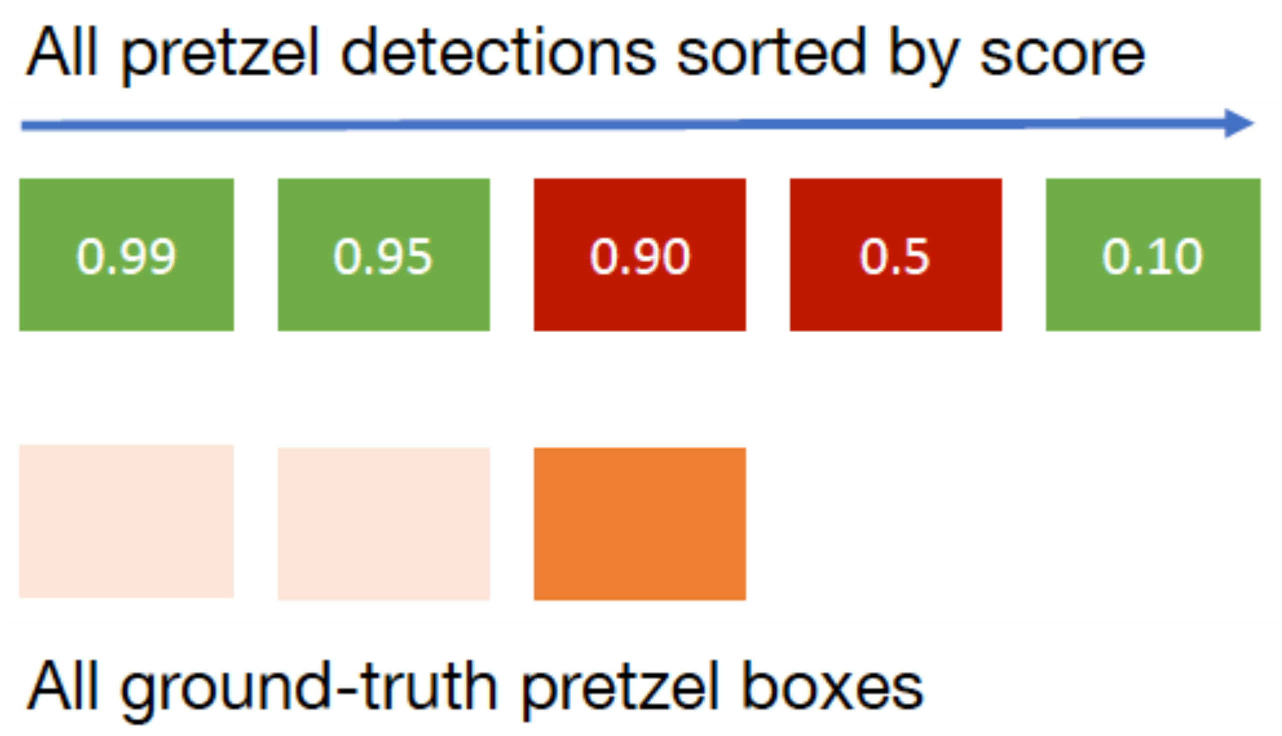
1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
    1. For each detection (highest score to lowest score)
        1. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
   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

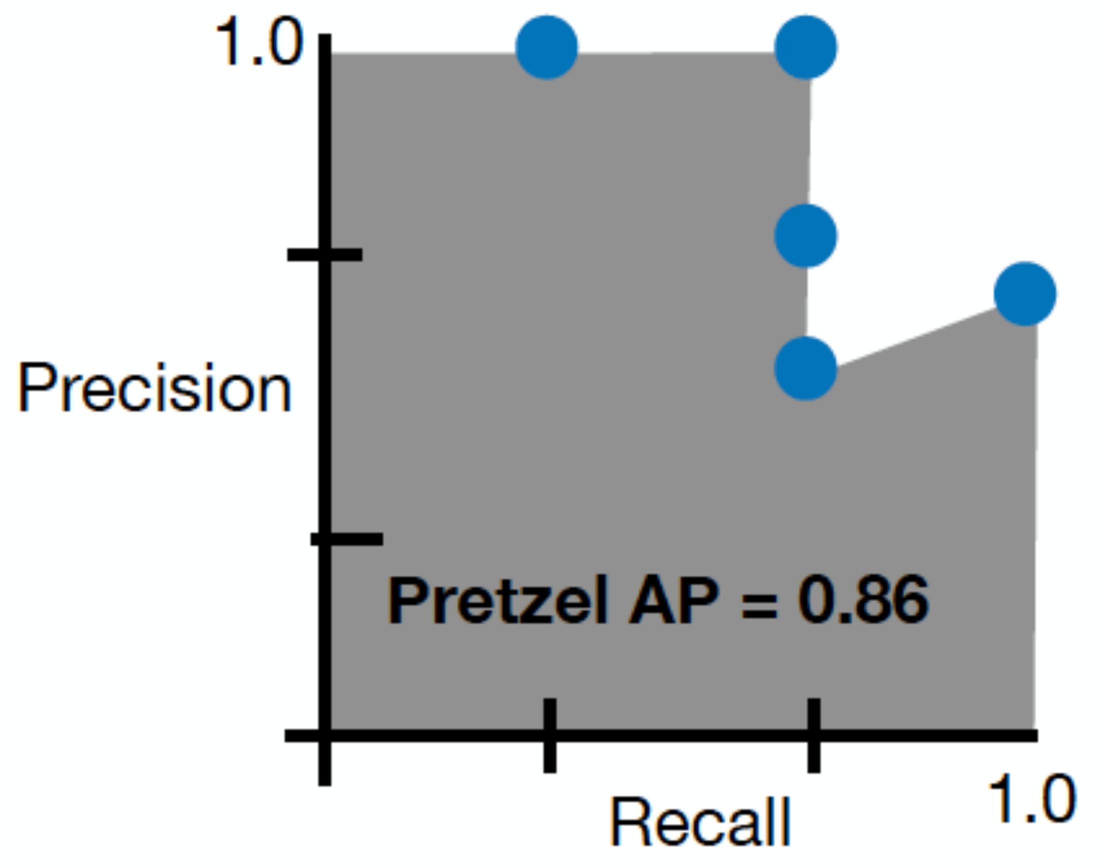


# Evaluating Object Detectors: Mean Average Precision (mAP) and P-R Curve

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) and P-R Curve

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

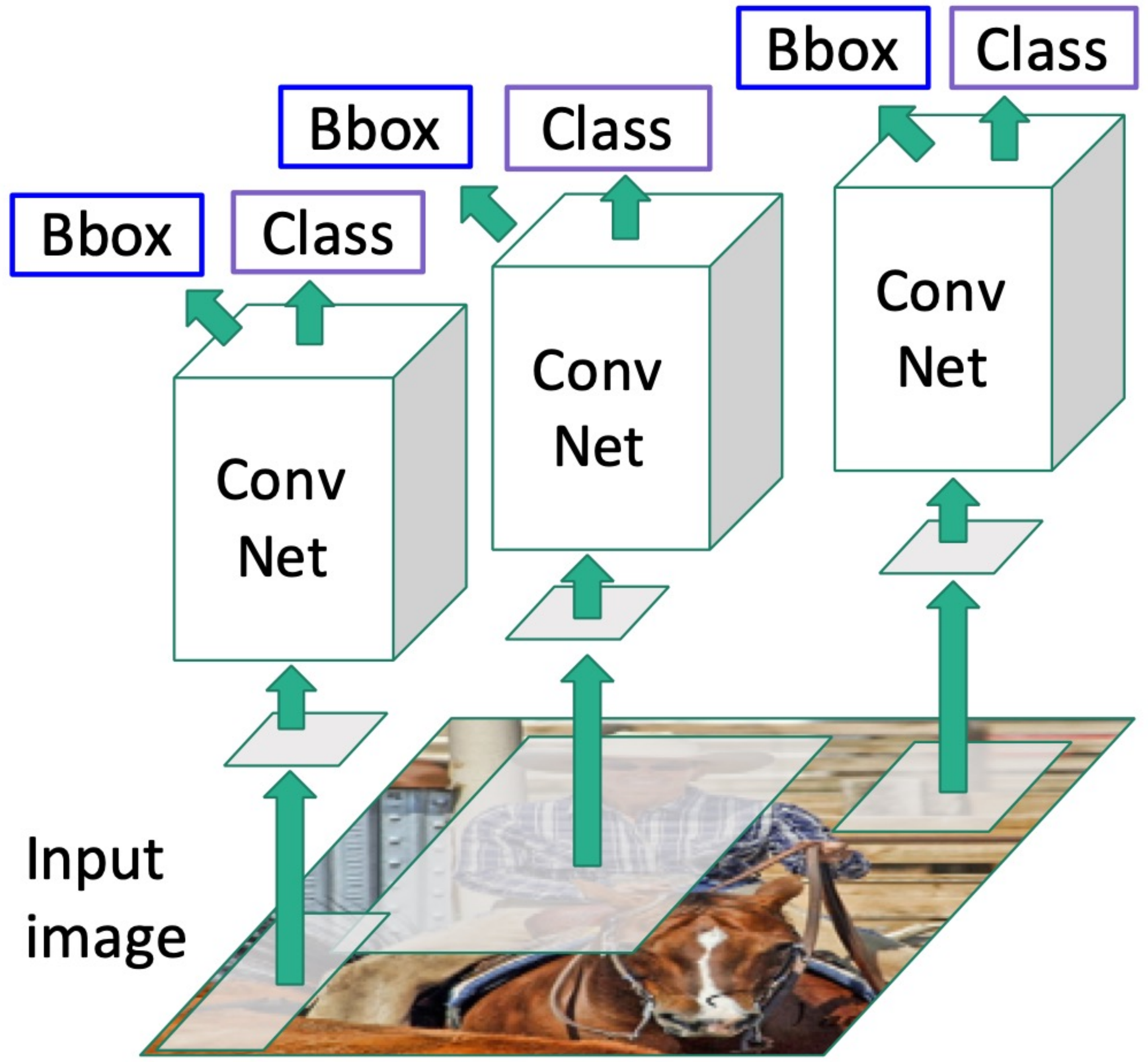
Flipz AP = 0.60
Hershey's AP = 0.85
Reese's AP = 0.81
mAP@0.5 = 0.75

"COCO Evaluator"

# Fast R-CNN

Input image

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



# Fast R-CNN

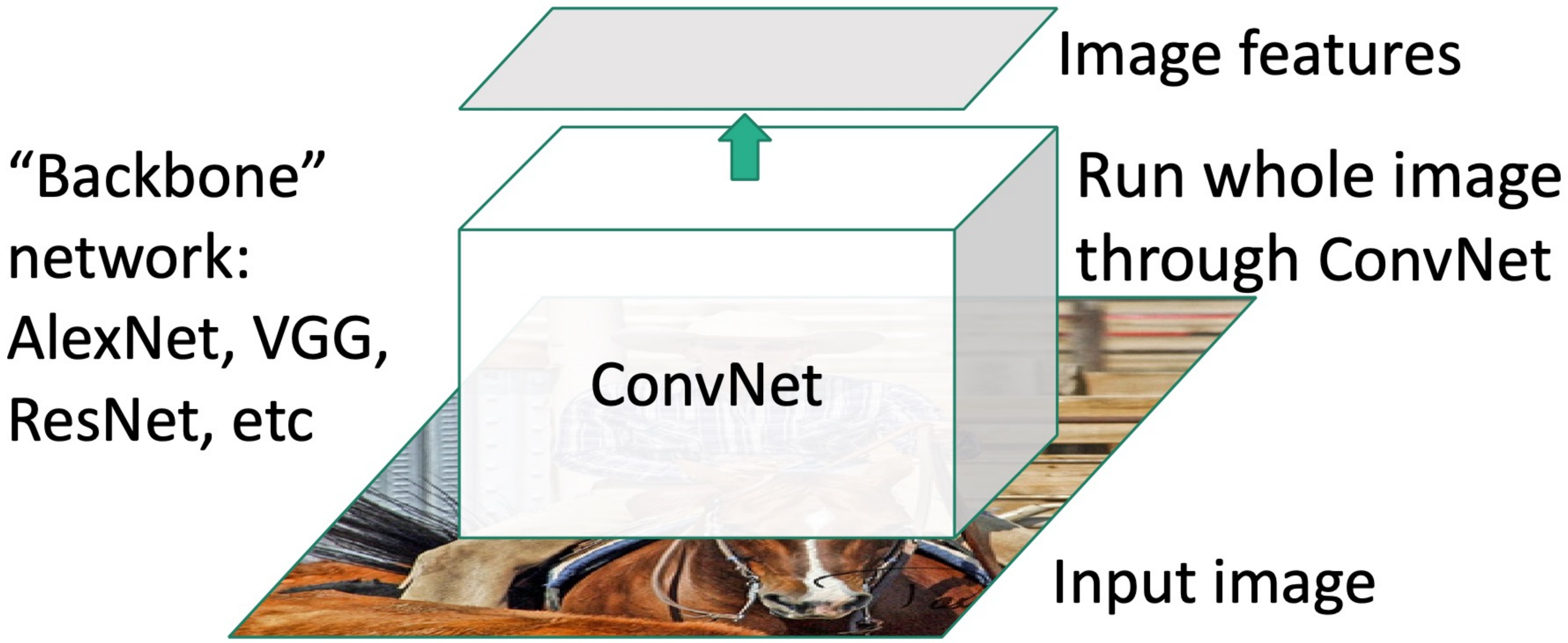
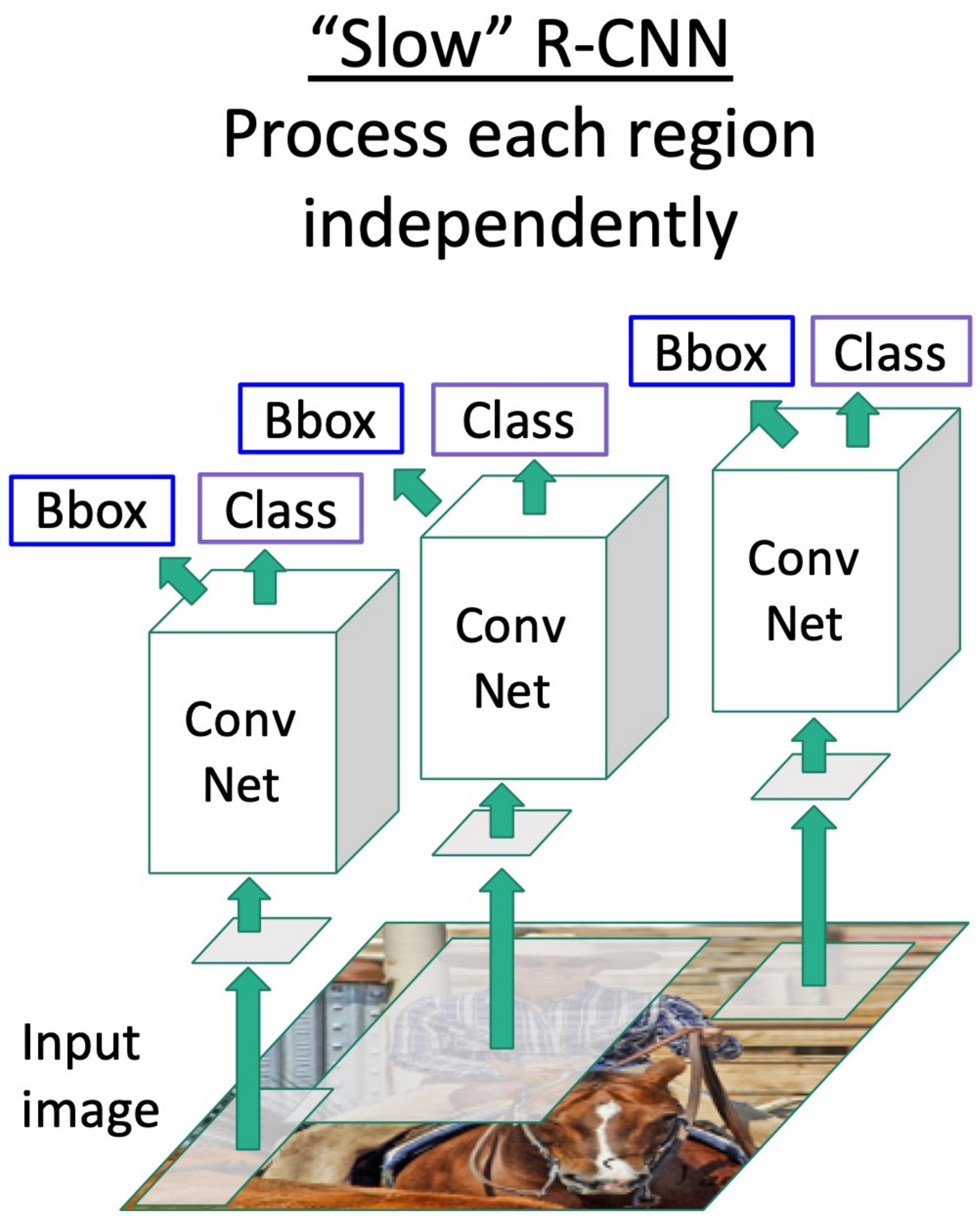


Image features
"Backbone" network: AlexNet, VGG, ResNet, etc
ConvNet
Run whole image through ConvNet
Input image

"Slow" R-CNN
Process each region independently
Bbox
Class
Bbox
Class
Bbox
Class
Conv Net
Conv Net
Conv Net
Input image

# Fast R-CNN

Regions of Interest (RoIs) from a proposal method

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

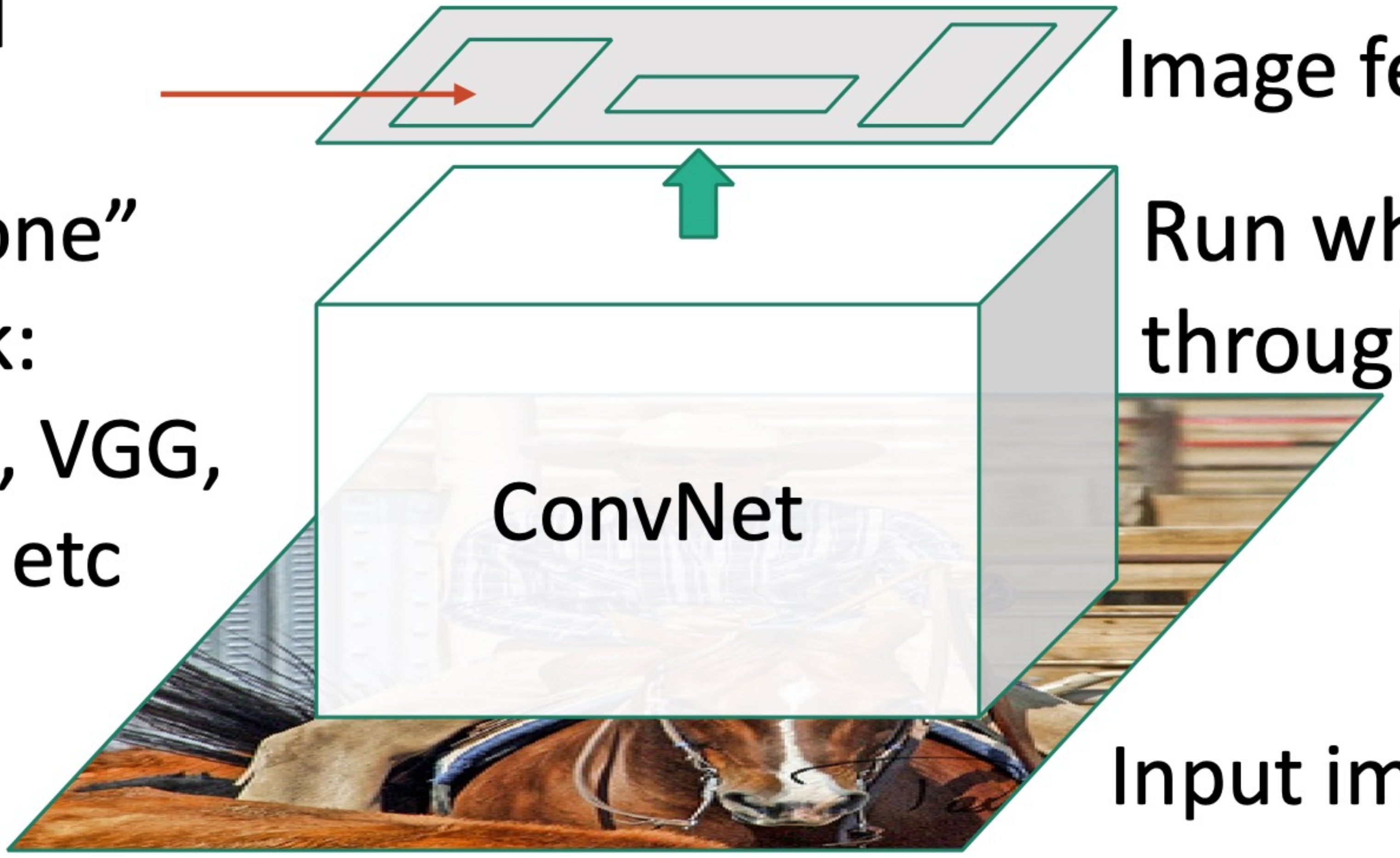


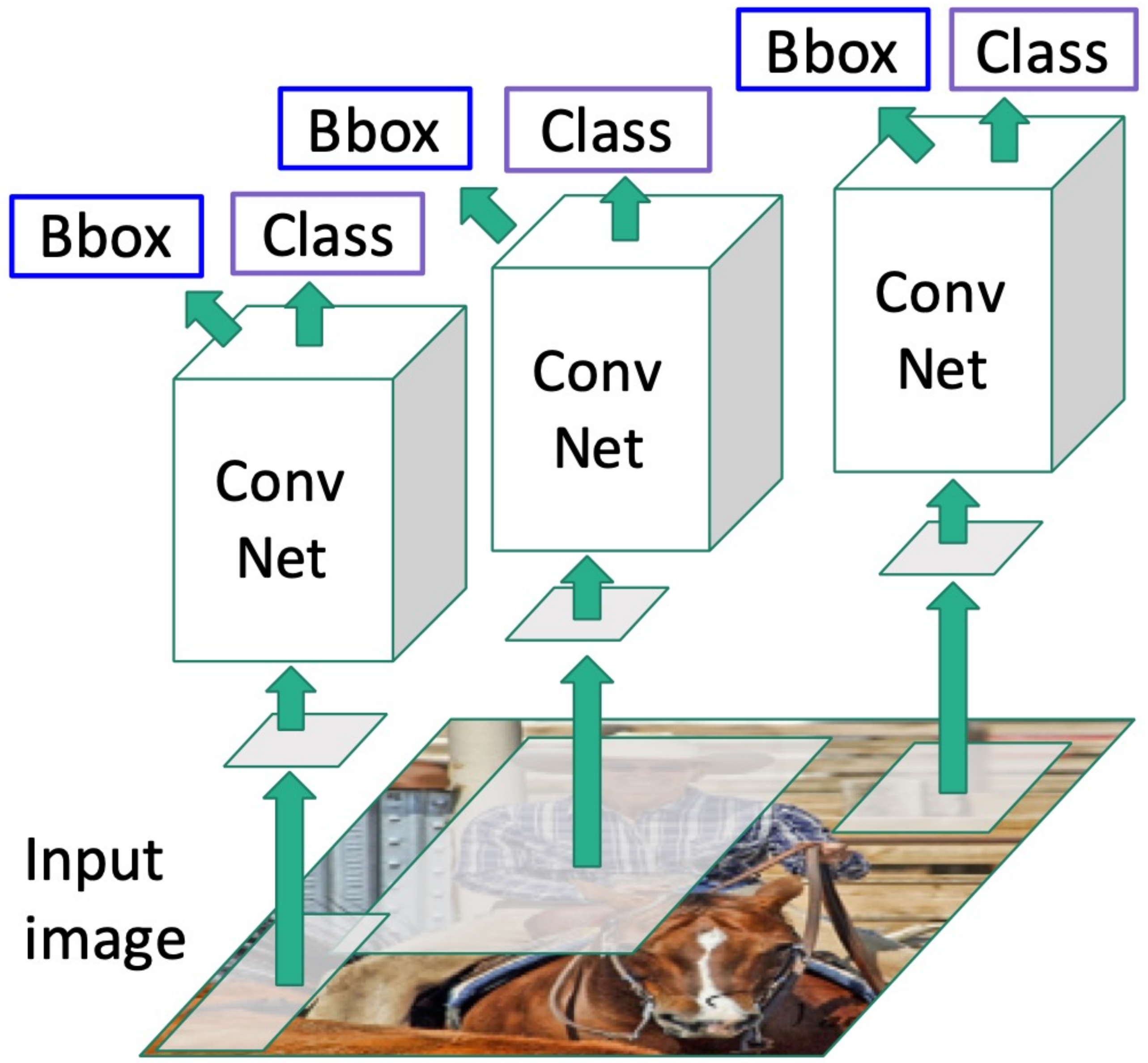

Image features

Run whole image through ConvNet

Input image

"Slow" R-CNN

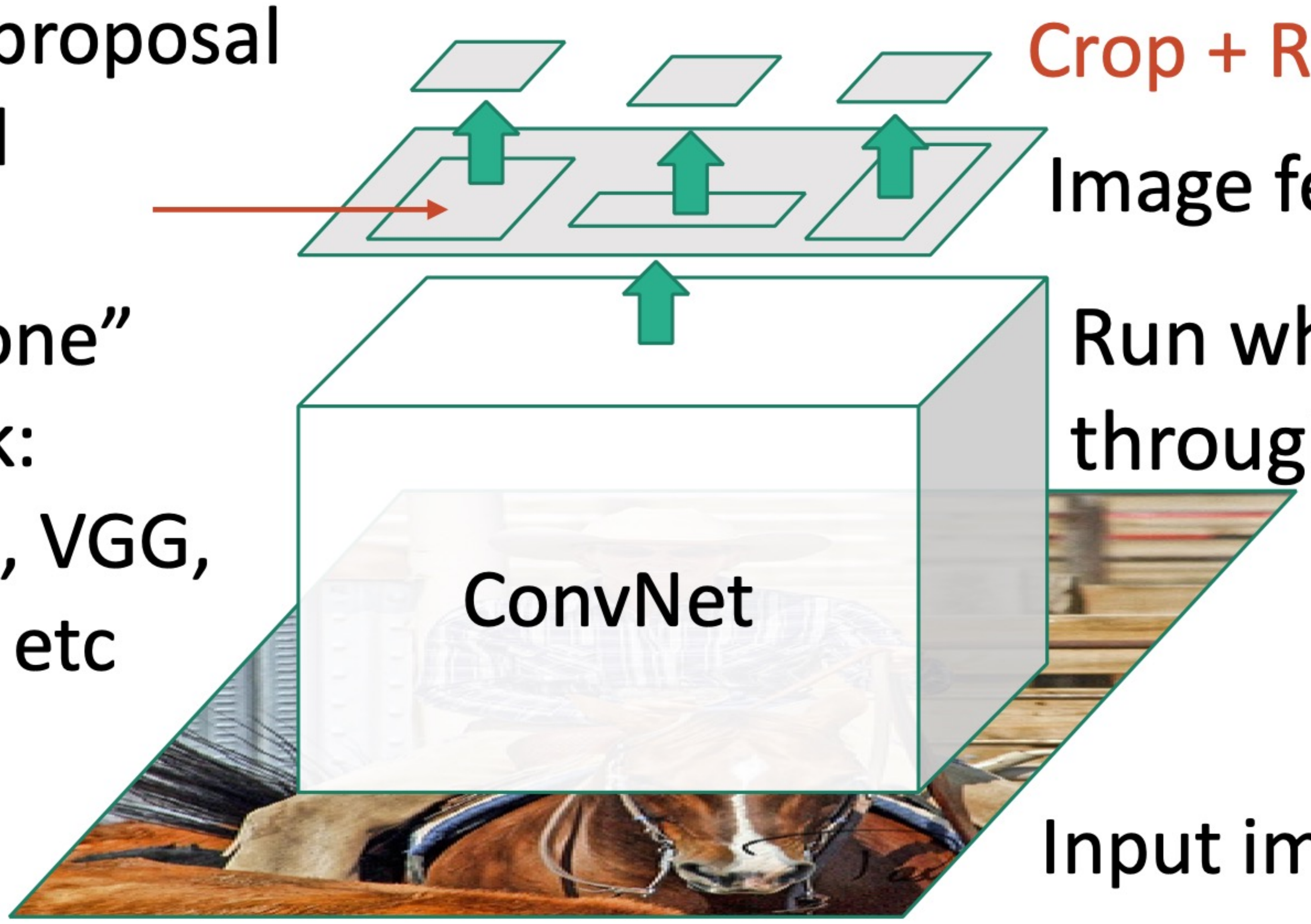
Process each region independently



# Fast R-CNN

Regions of Interest (RoIs) from a proposal method

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



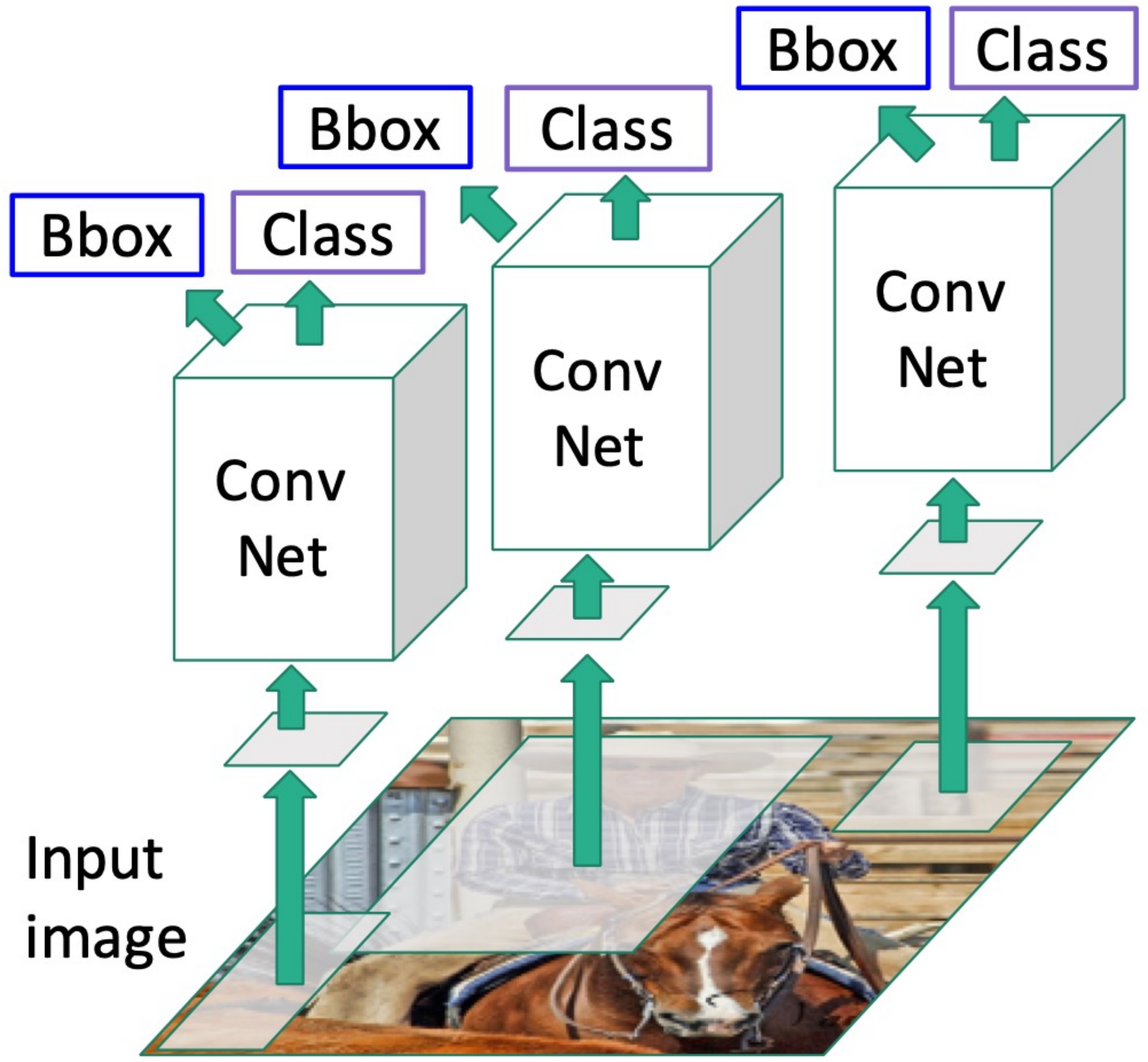

Crop + Resize features

Image features

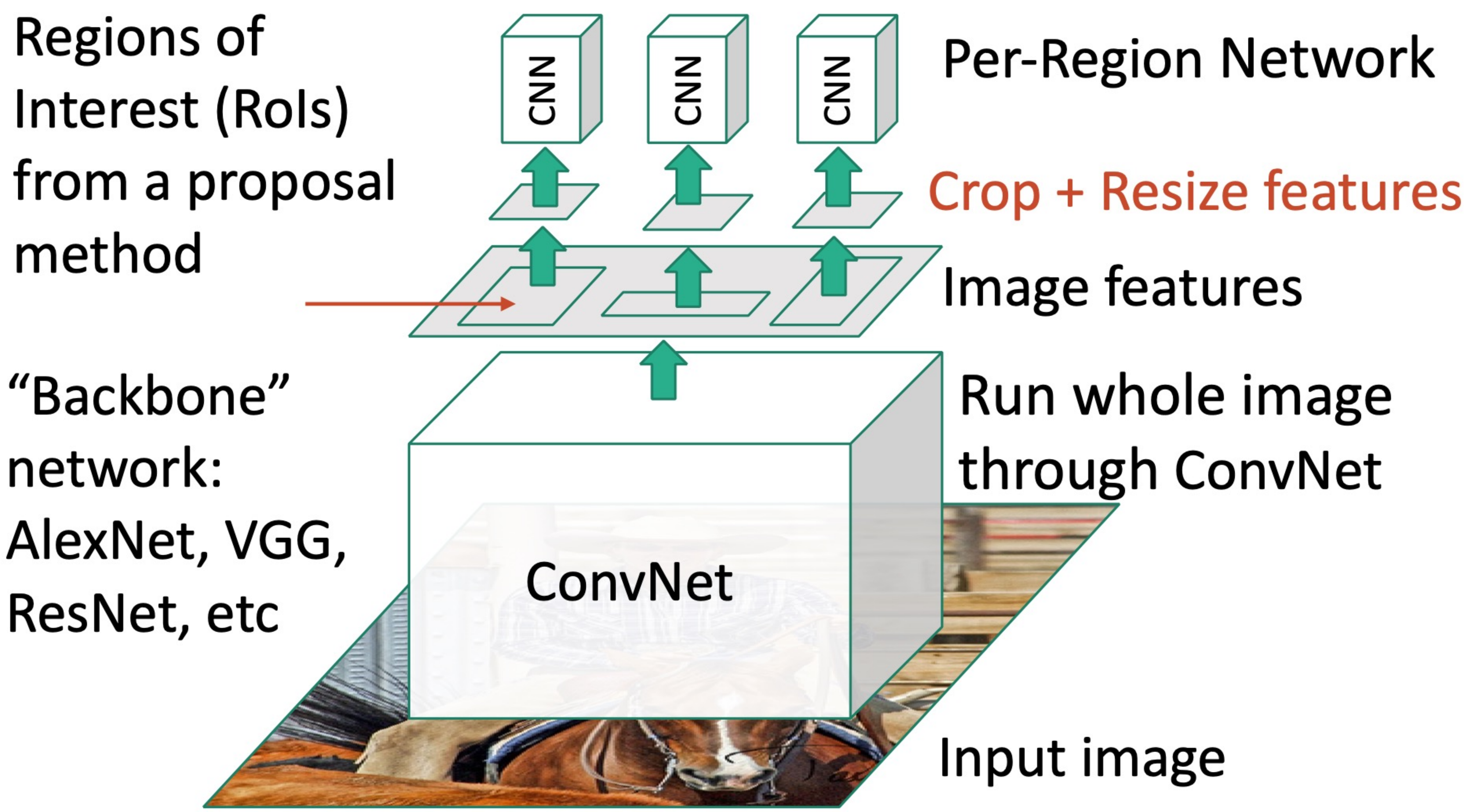
Run whole image through ConvNet

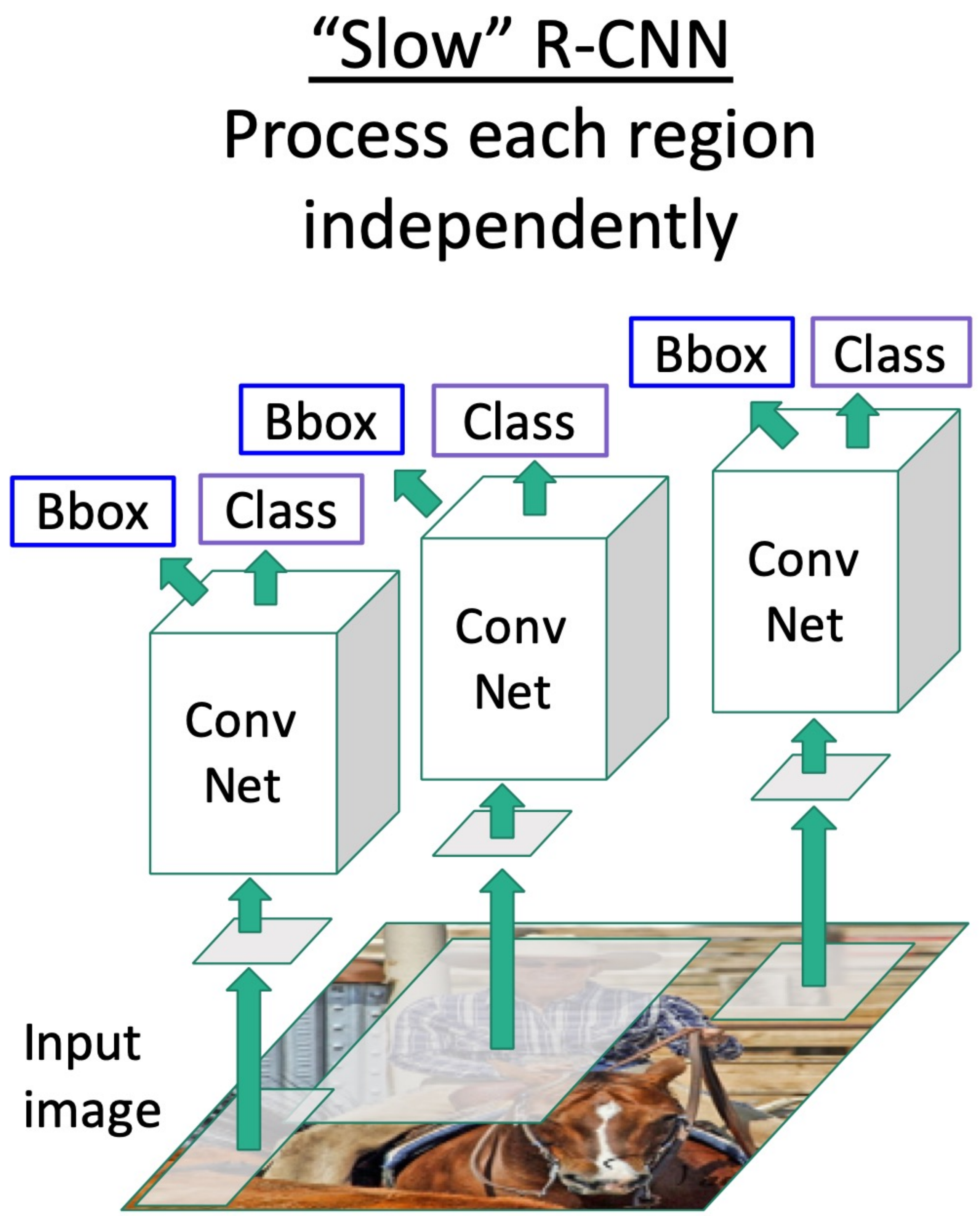
Input image

"Slow" R-CNN
Process each region independently

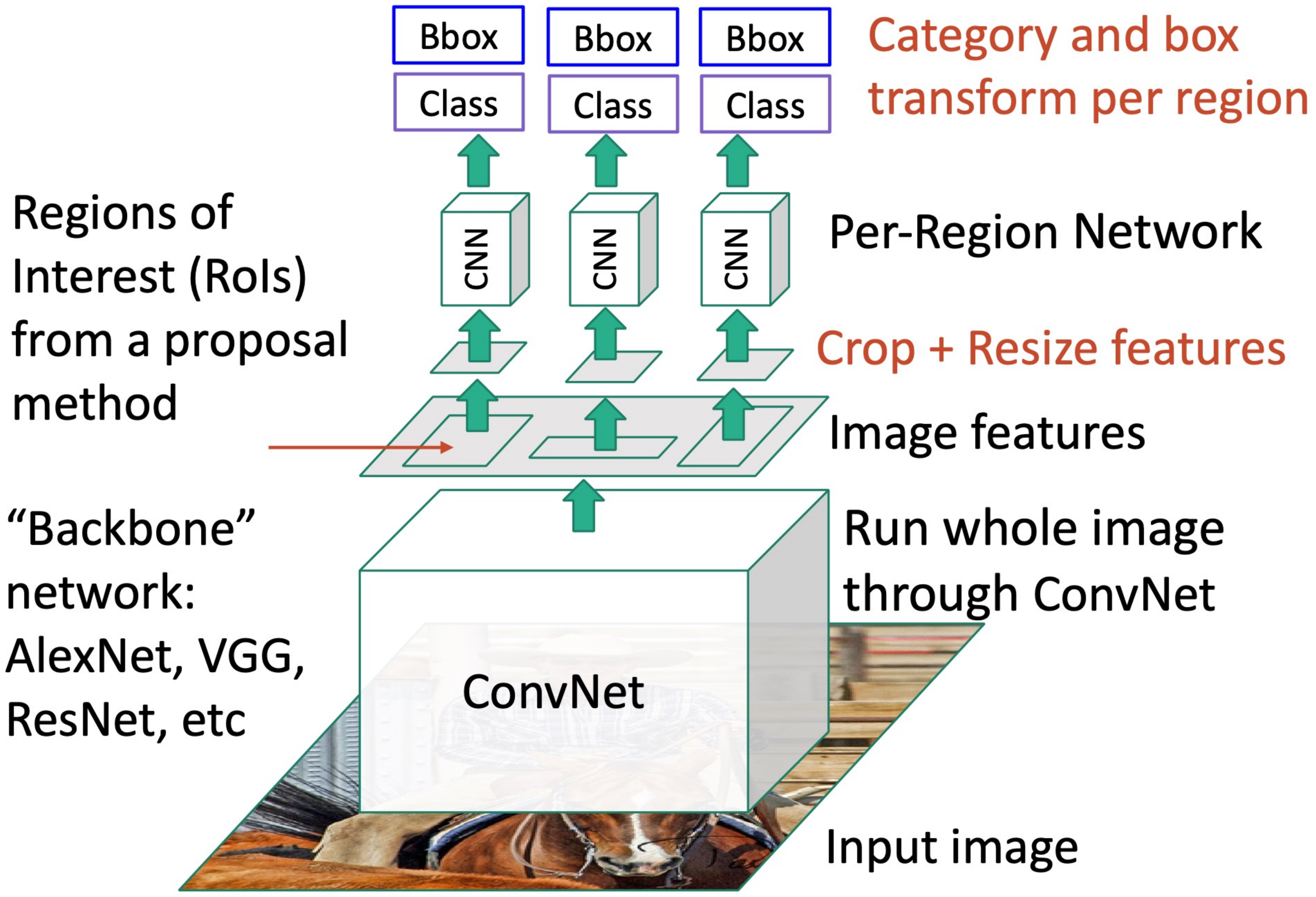


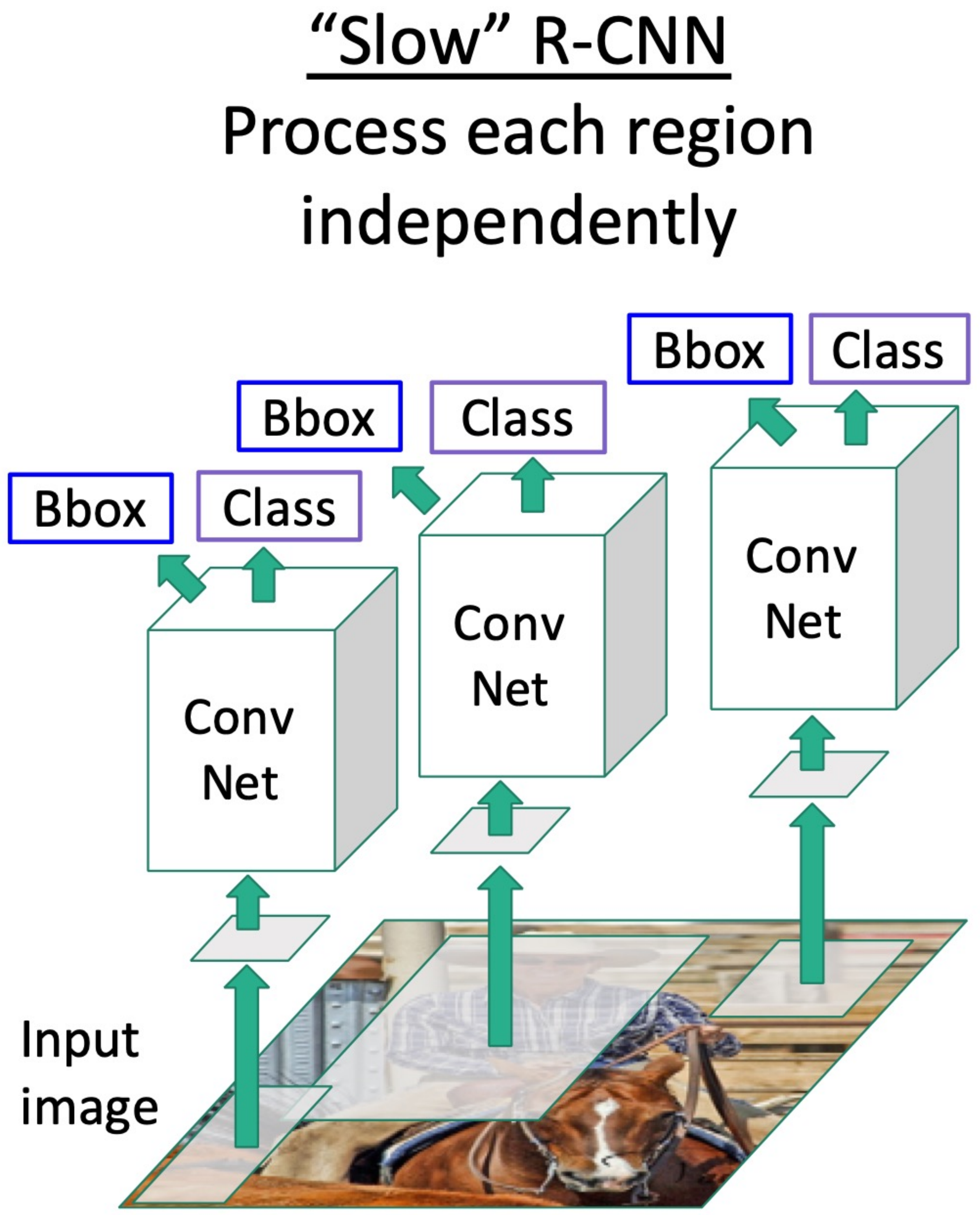
# Fast R-CNN


Regions of Interest (RoIs) from a proposal method
"Backbone" network: AlexNet, VGG, ResNet, etc
CNN
CNN
CNN
ConvNet
Per-Region Network
Crop + Resize features
Image features
Run whole image through ConvNet
Input image

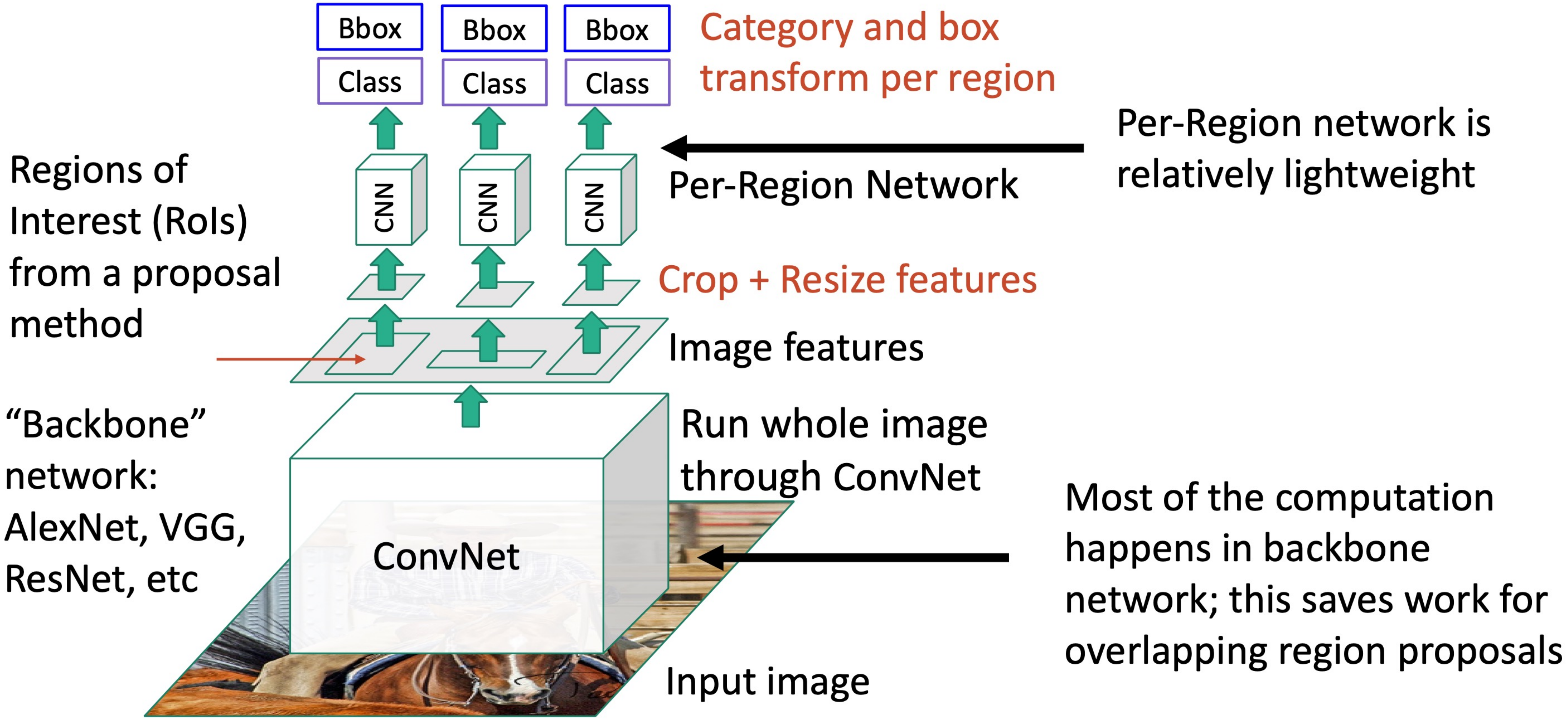

"Slow" R-CNN
Process each region independently
Bbox
Class
Bbox
Class
Bbox
Class
Conv Net
Conv Net
Conv Net
Input image

# Fast R-CNN


Bbox
Class
Bbox
Class
Bbox
Class
Category and box transform per region
CNN
CNN
CNN
Per-Region Network
Crop + Resize features
Regions of Interest (RoIs) from a proposal method
Image features
"Backbone" network: AlexNet, VGG, ResNet, etc
ConvNet
Run whole image through ConvNet
Input image


"Slow" R-CNN
Process each region independently
Bbox
Class
Bbox
Class
Bbox
Class
Conv Net
Conv Net
Conv Net
Input image

# Fast R-CNN

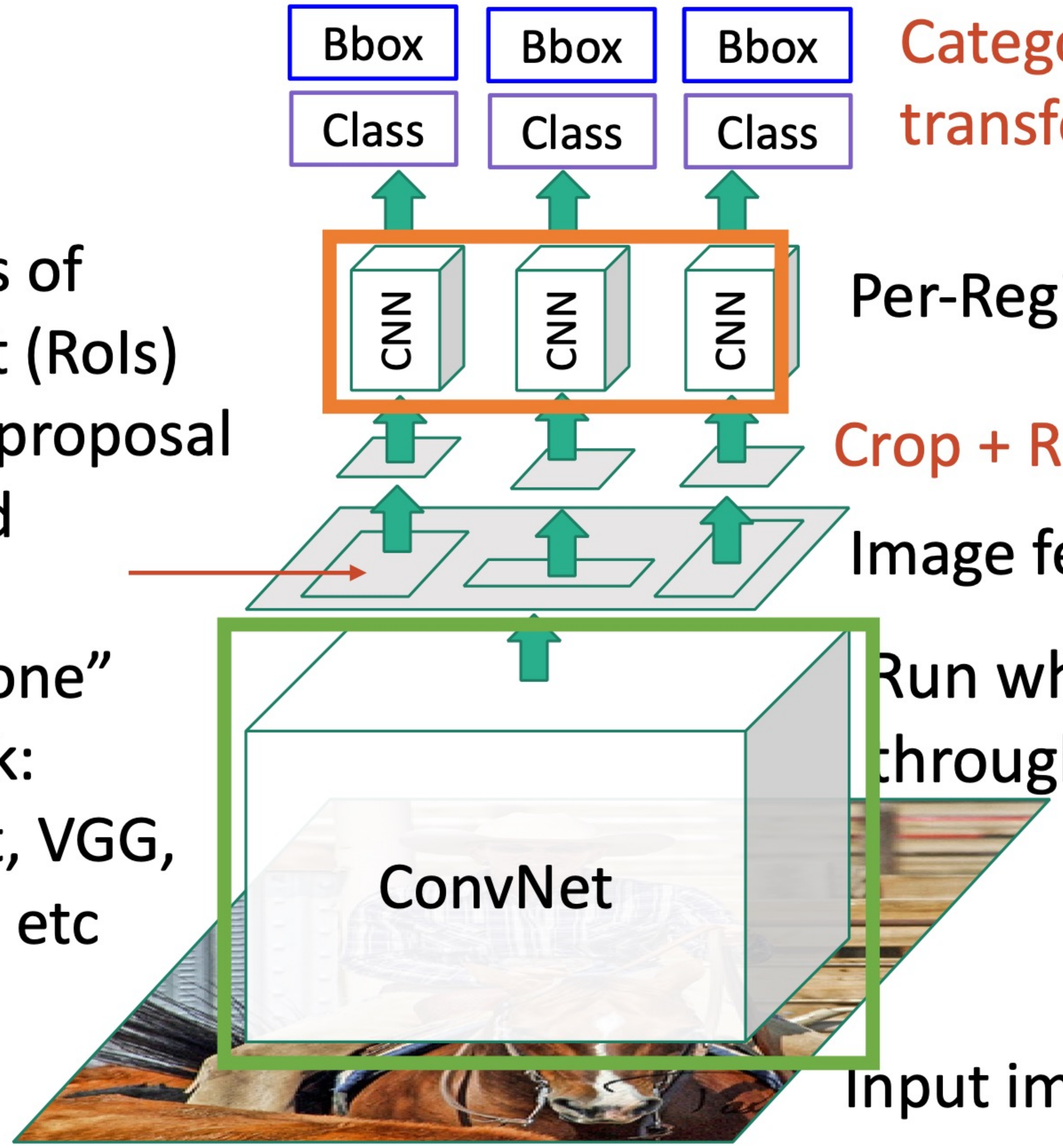


Bbox
Bbox
Bbox
Class
Class
Class
Category and box transform per region
Per-Region network is relatively lightweight
Regions of Interest (RoIs) from a proposal method
CNN
CNN
CNN
Per-Region Network
Crop + Resize features
Image features
"Backbone" network: AlexNet, VGG, ResNet, etc
Run whole image through ConvNet
ConvNet
Most of the computation happens in backbone network; this saves work for overlapping region proposals
Input image

# Fast R-CNN

Regions of Interest (RoIs) from a proposal method

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




Category and box transform per region

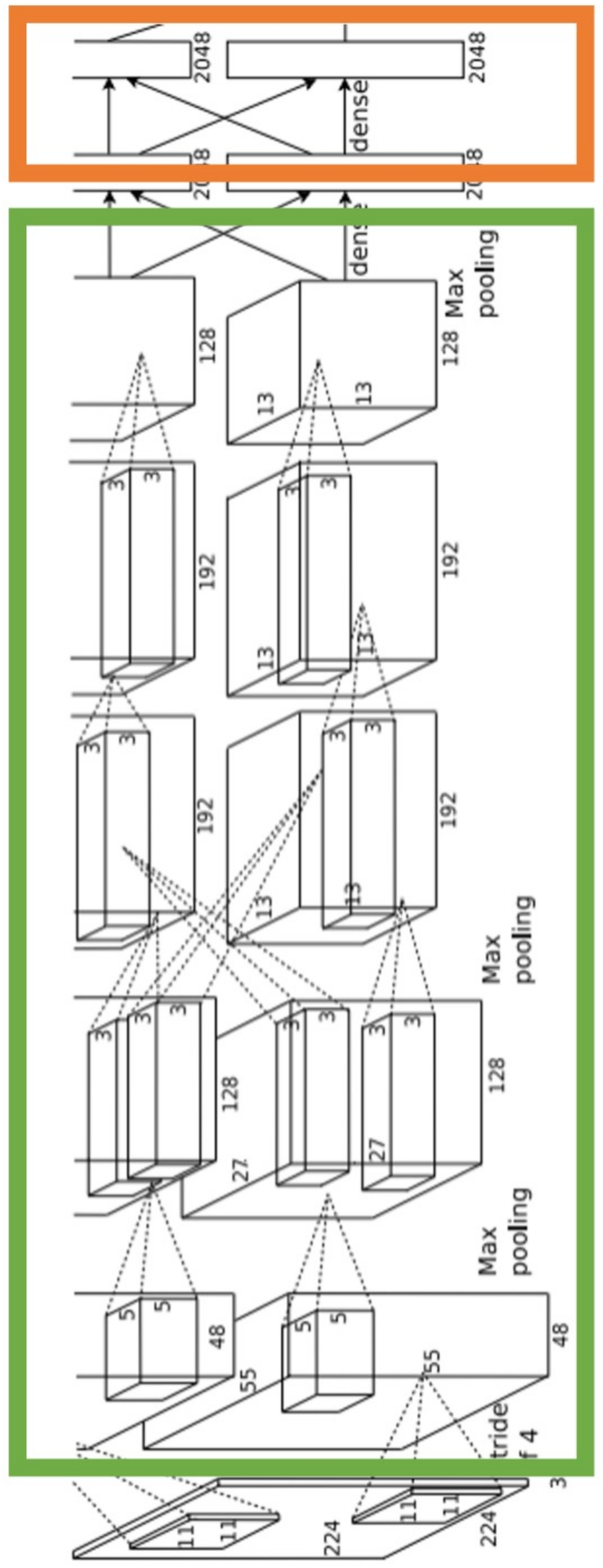
Per-Region Network

Crop + Resize features

Image features

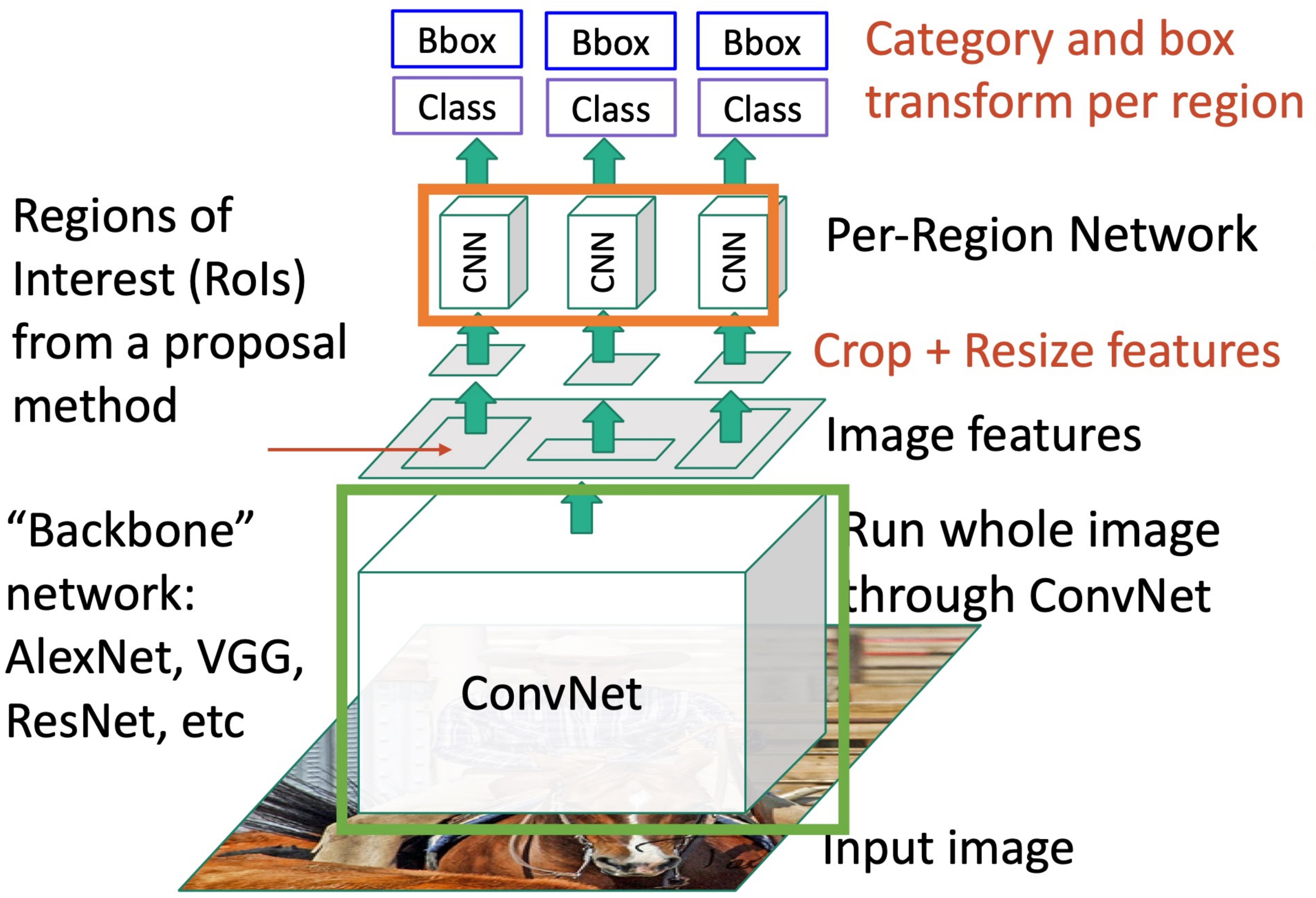
Run whole image through ConvNet

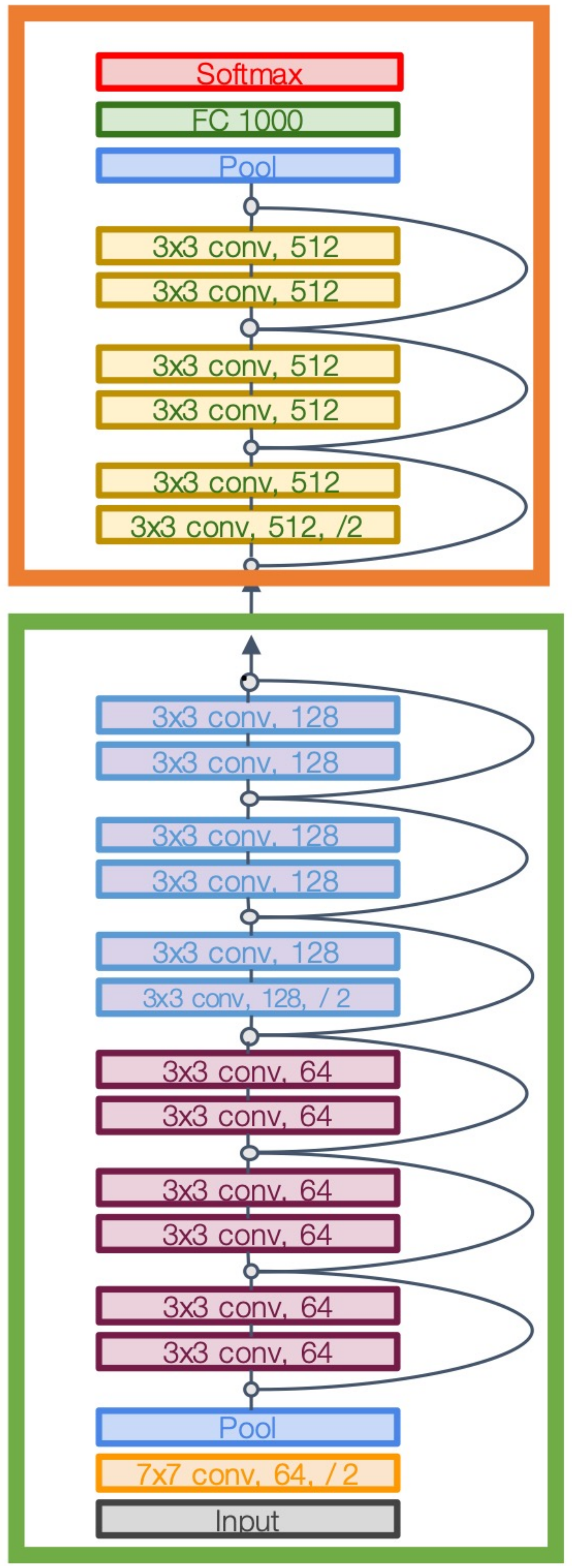
Input image



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

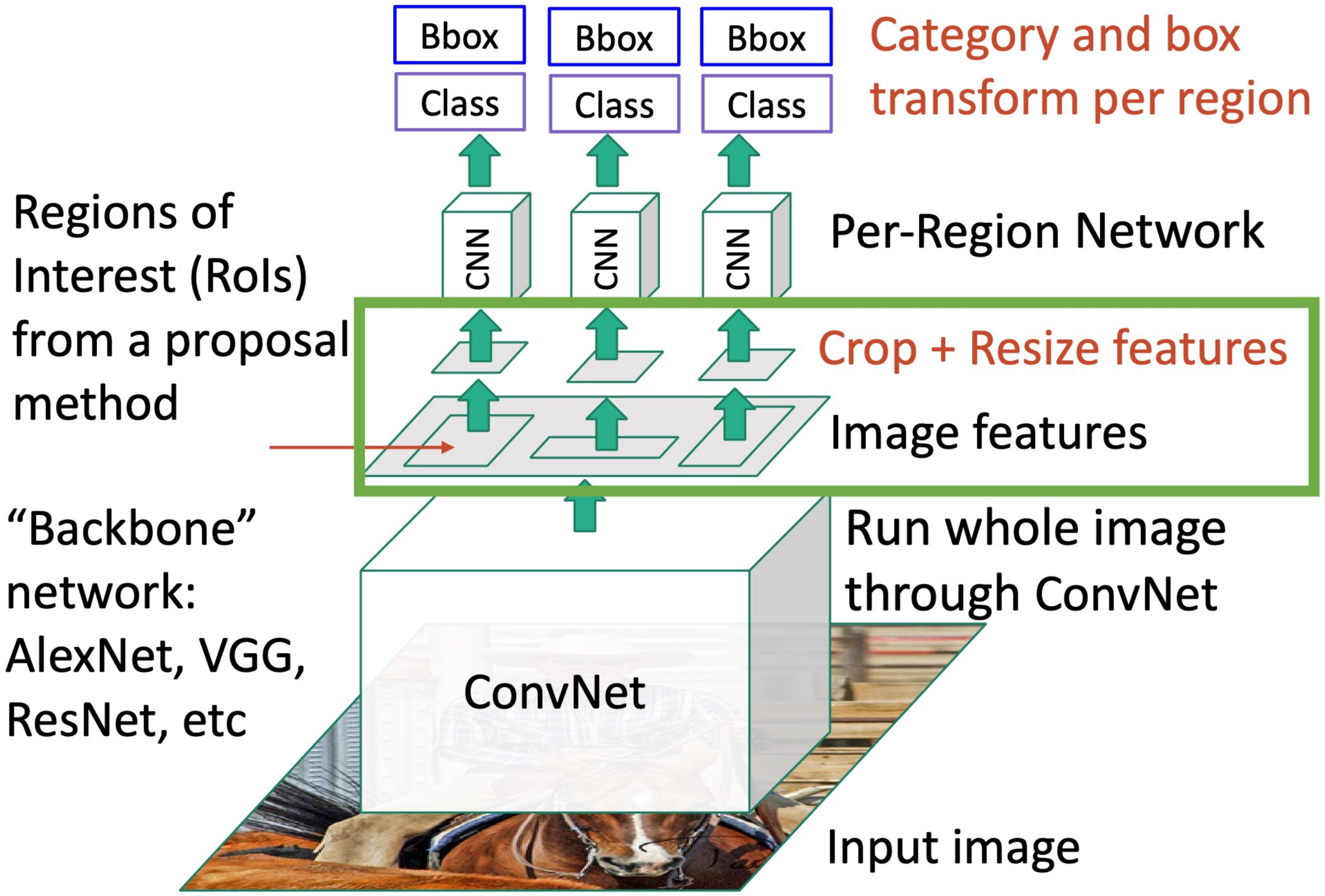
# Fast R-CNN




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

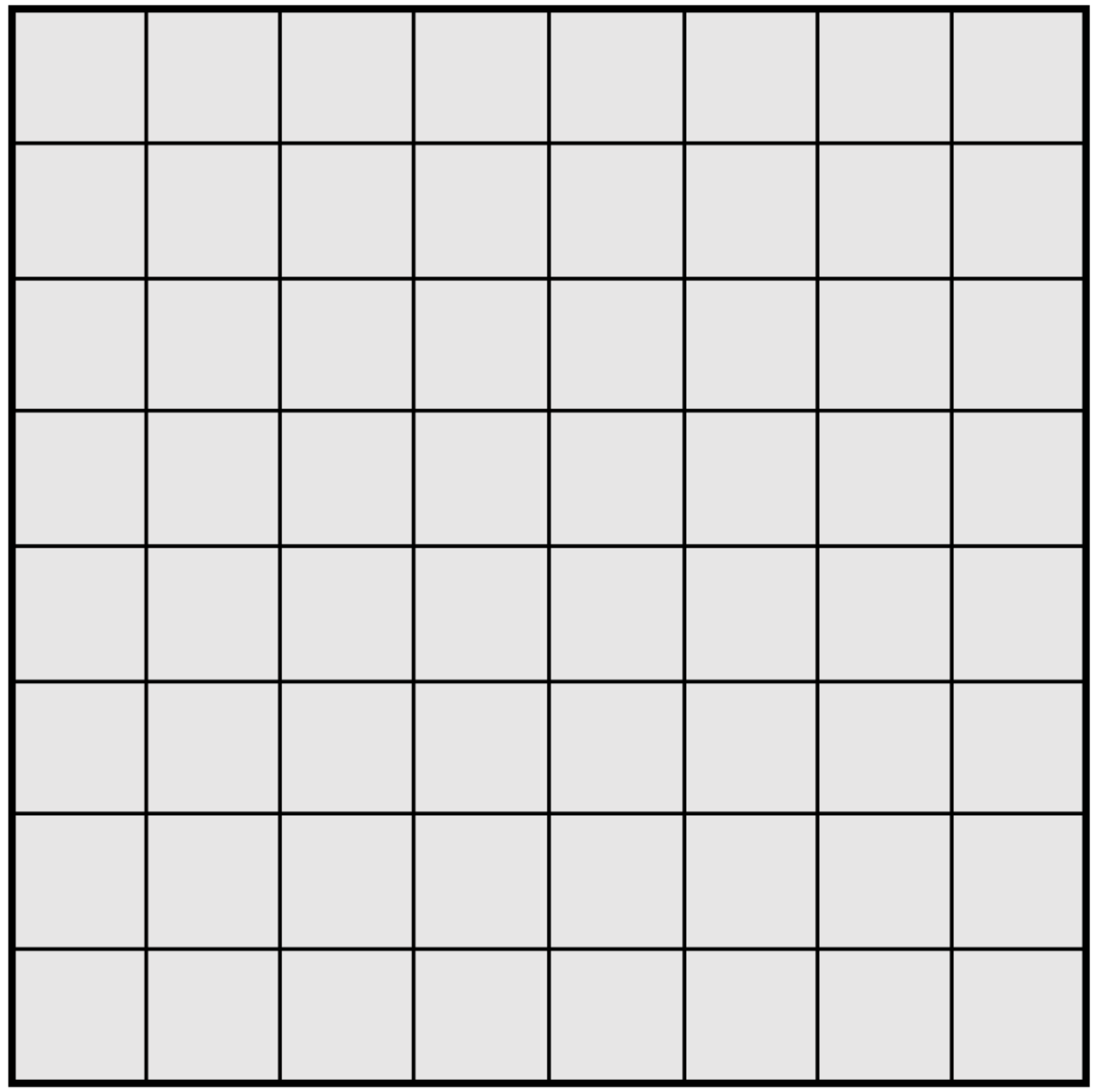
# Fast R-CNN



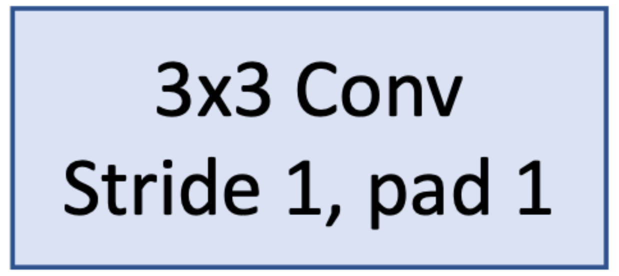

How to crop features?

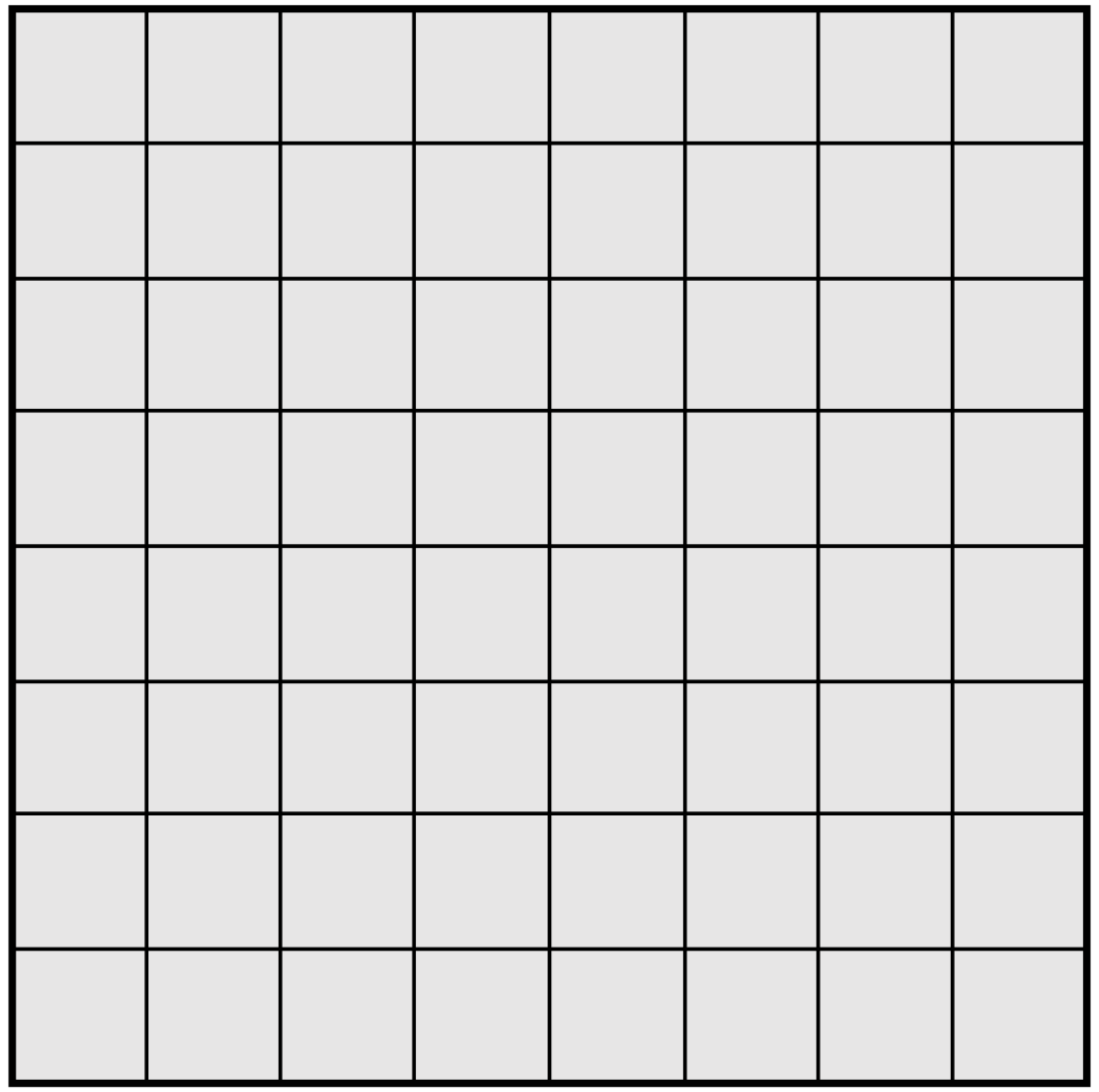
# Recall: Receptive Fields

Every position in the output feature map depends on a 3x3 receptive field in the input



Input Image: 8 x 8

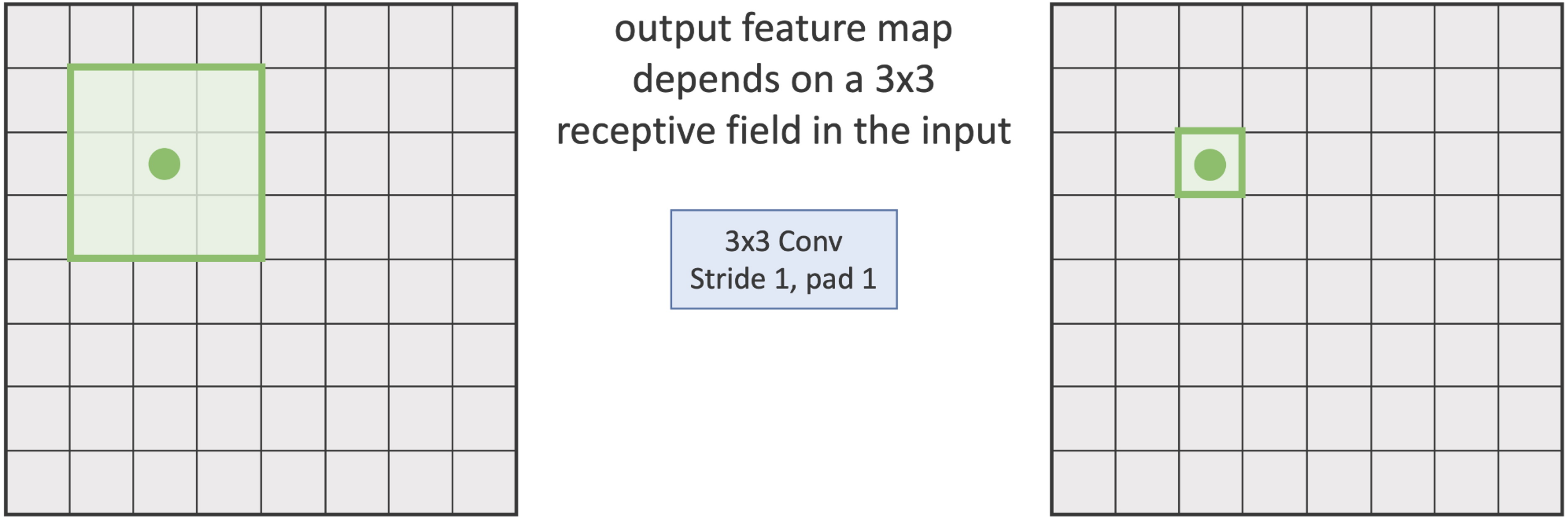


Output Image: 8 x 8

# Recall: Receptive Fields

Every position in the output feature map depends on a 3x3 receptive field in the input

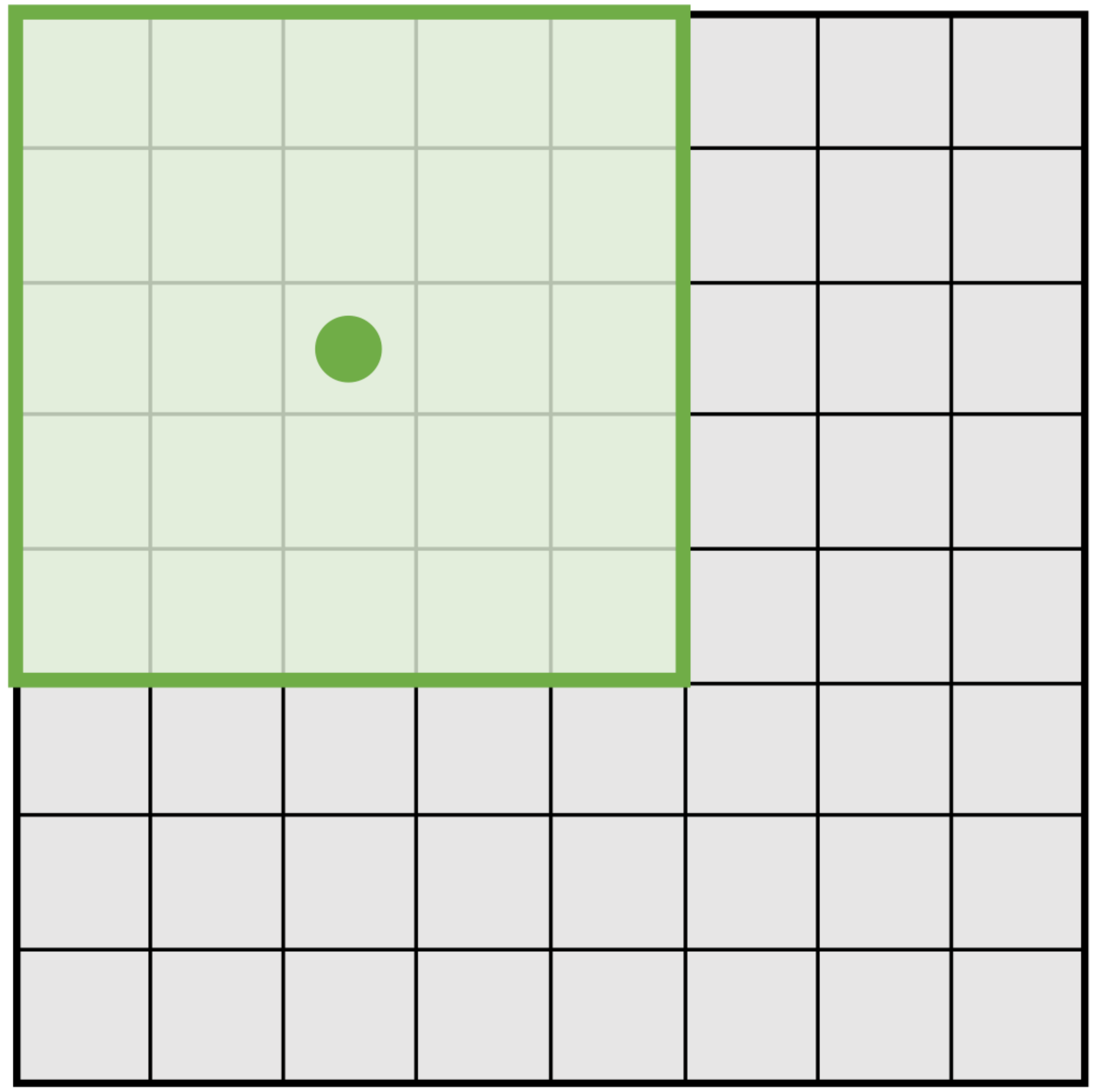



Input Image: 8 x 8

Output Image: 8 x 8

# Recall: Receptive Fields

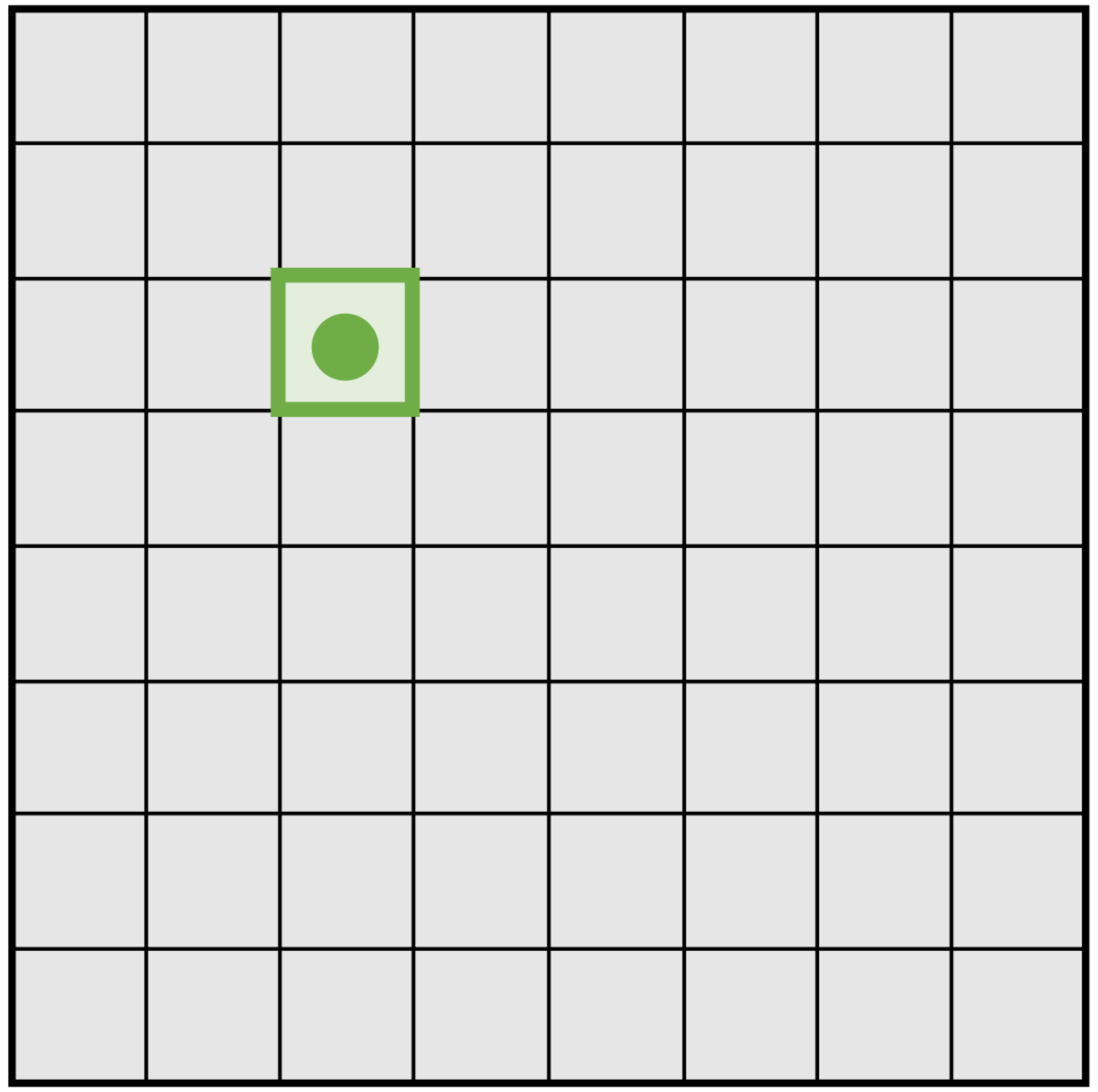
Every position in the output feature map depends on a <u>5x5</u> receptive field in the input



Input Image: 8 x 8

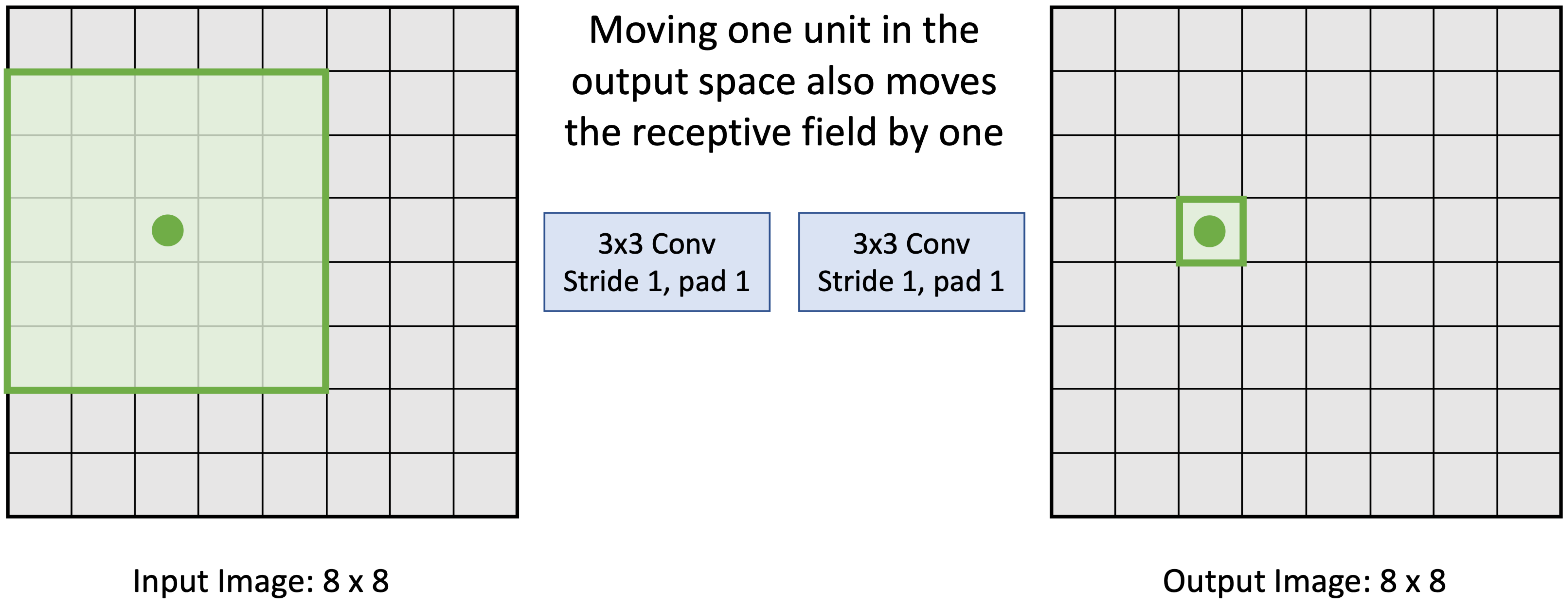
3x3 Conv
Stride 1, pad 1

3x3 Conv
Stride 1, pad 1

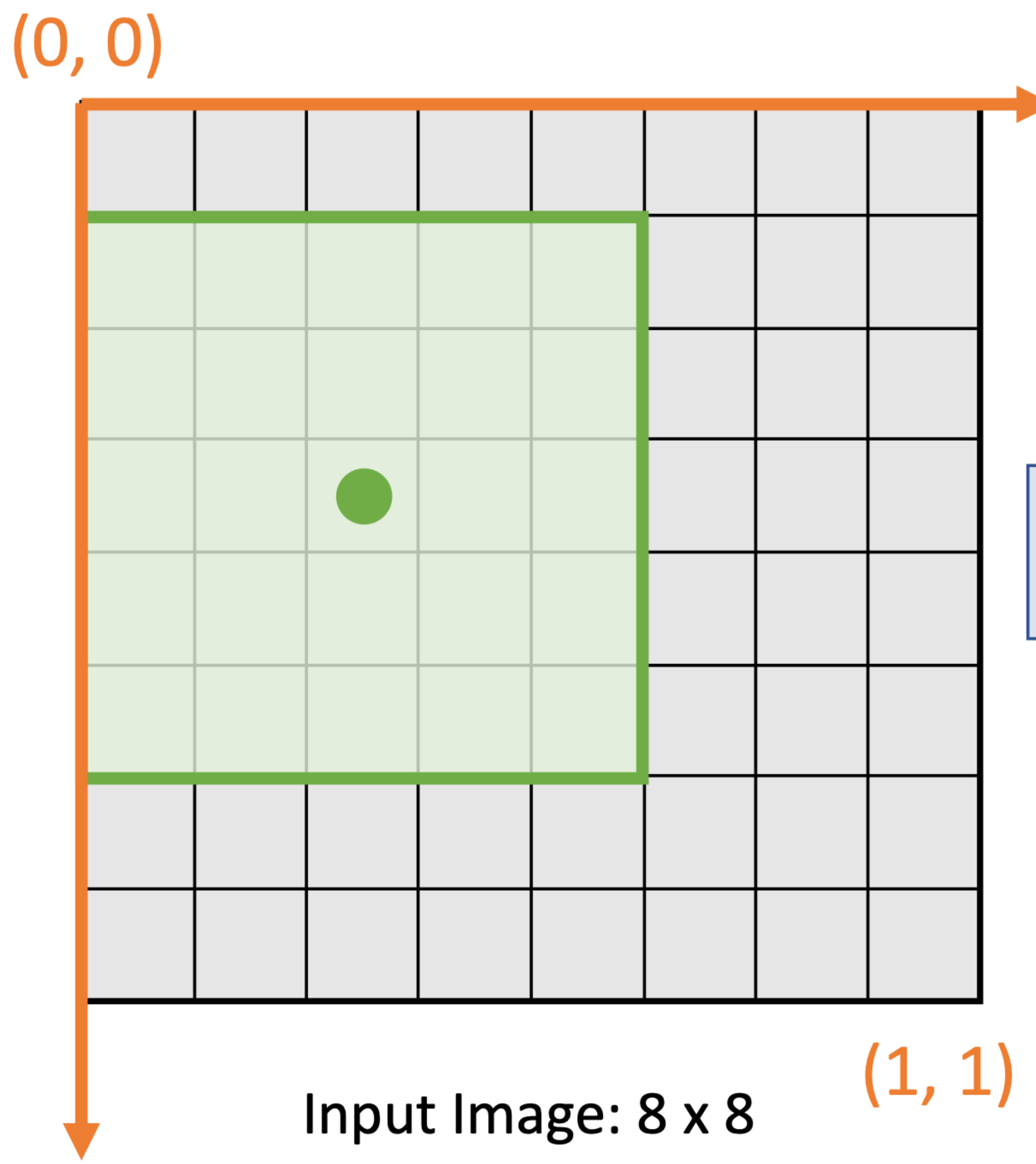


Output Image: 8 x 8

# Recall: Receptive Fields



# Recall: Receptive Fields

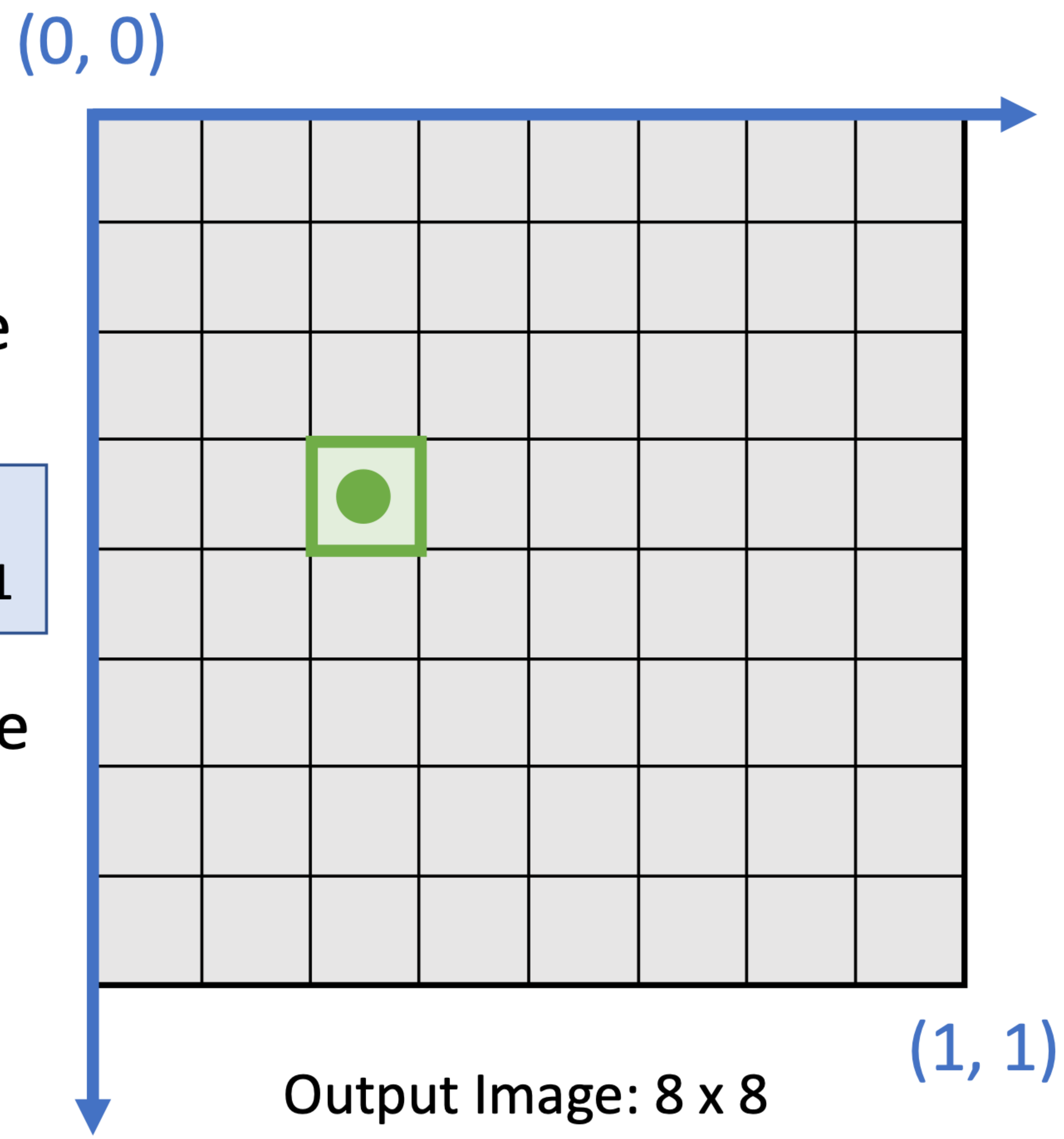



Moving one unit in the output space also moves the receptive field by one

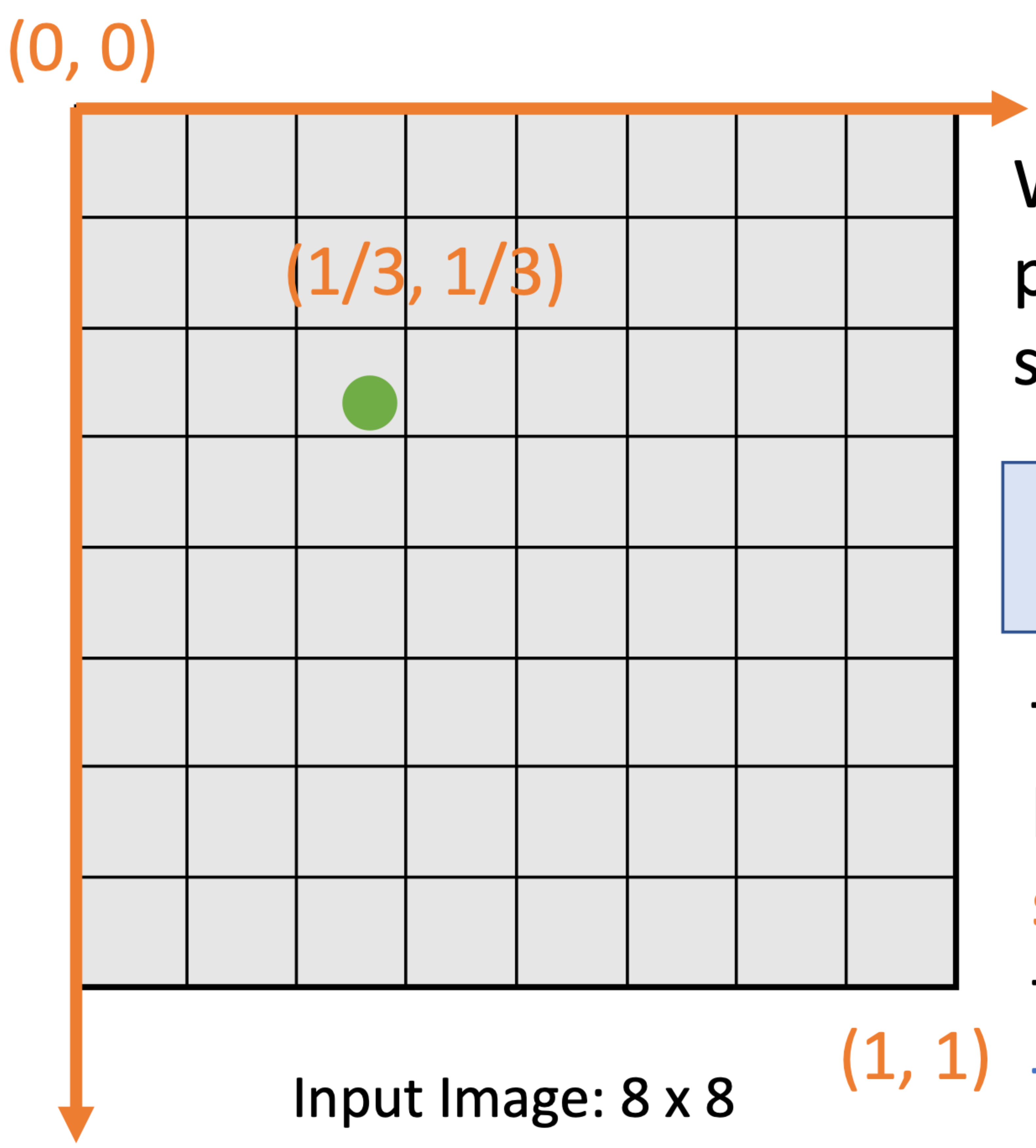
3x3 Conv
Stride 1, pad 1

3x3 Conv
Stride 1, pad 1

There is a correspondence between the coordinate system of the input and the coordinate system of the output



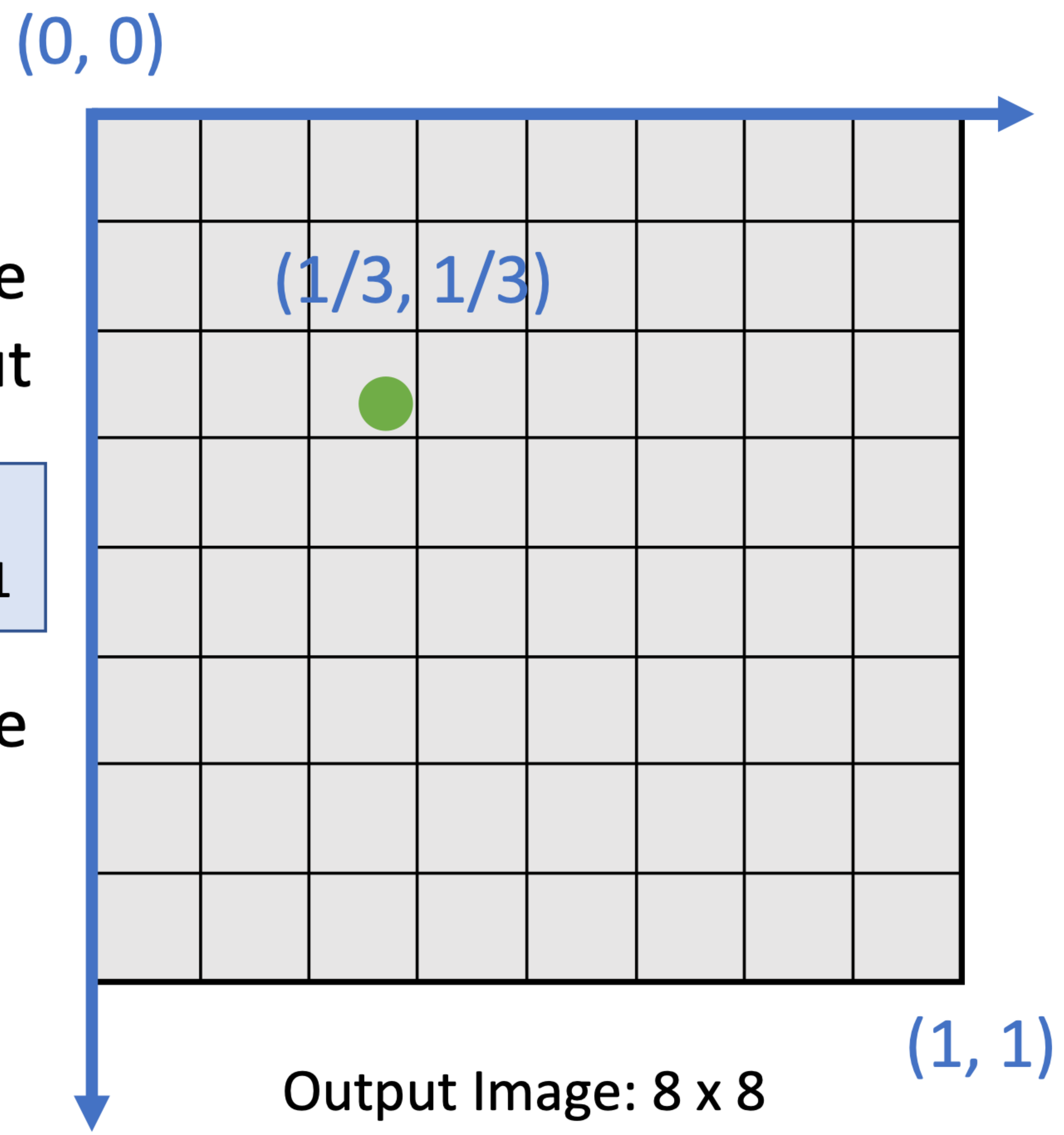
# Projecting Points




We can align arbitrary points between coordinate system of input and output

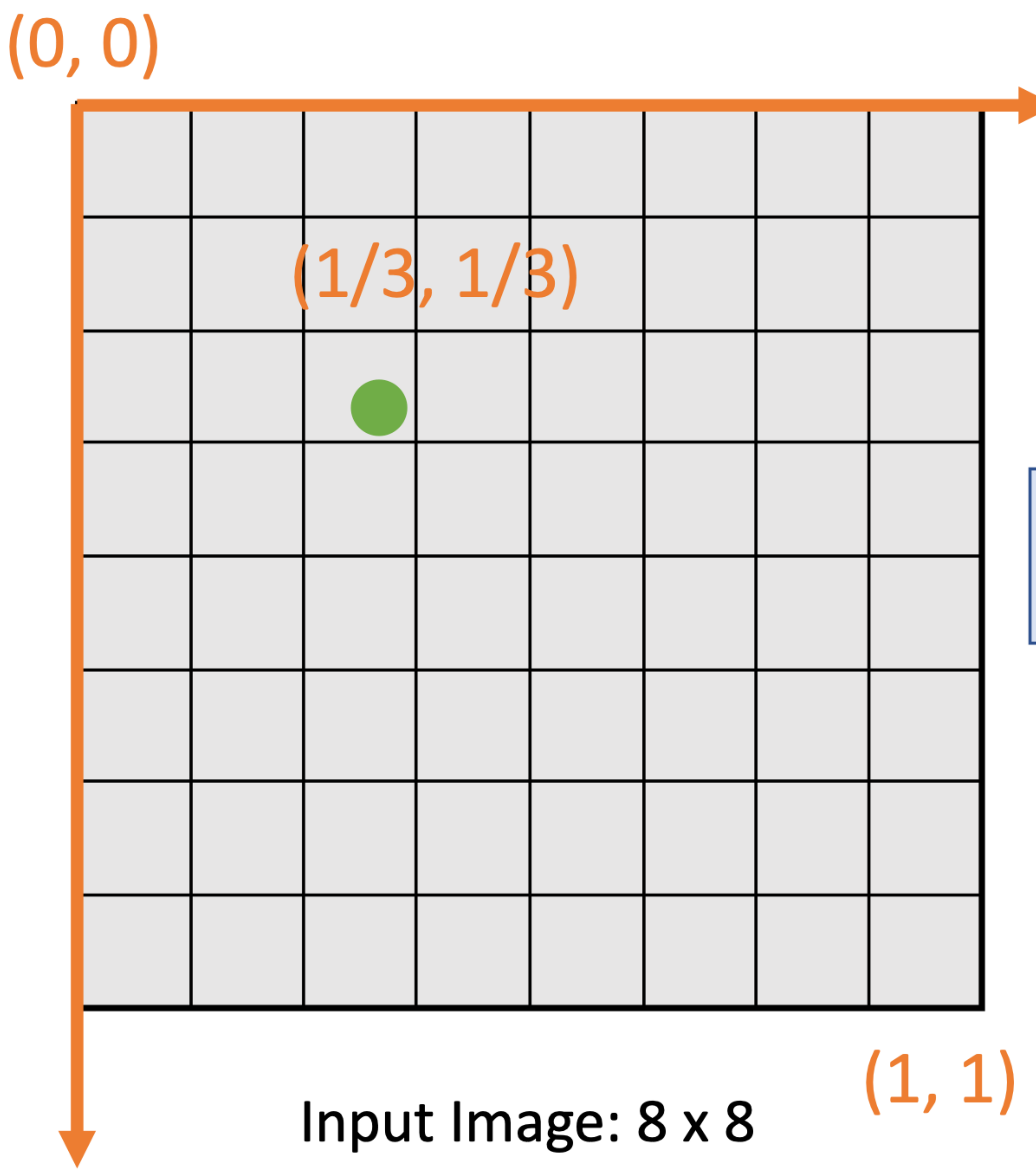
| 3x3 Conv Stride 1, pad 1 | 3x3 Conv Stride 1, pad 1 |
|---|---|

There is a correspondence between the coordinate system of the input and the coordinate system of the output



# Projecting Points

Same logic holds for more complicated CNNs, even if spatial resolution of input and output are different

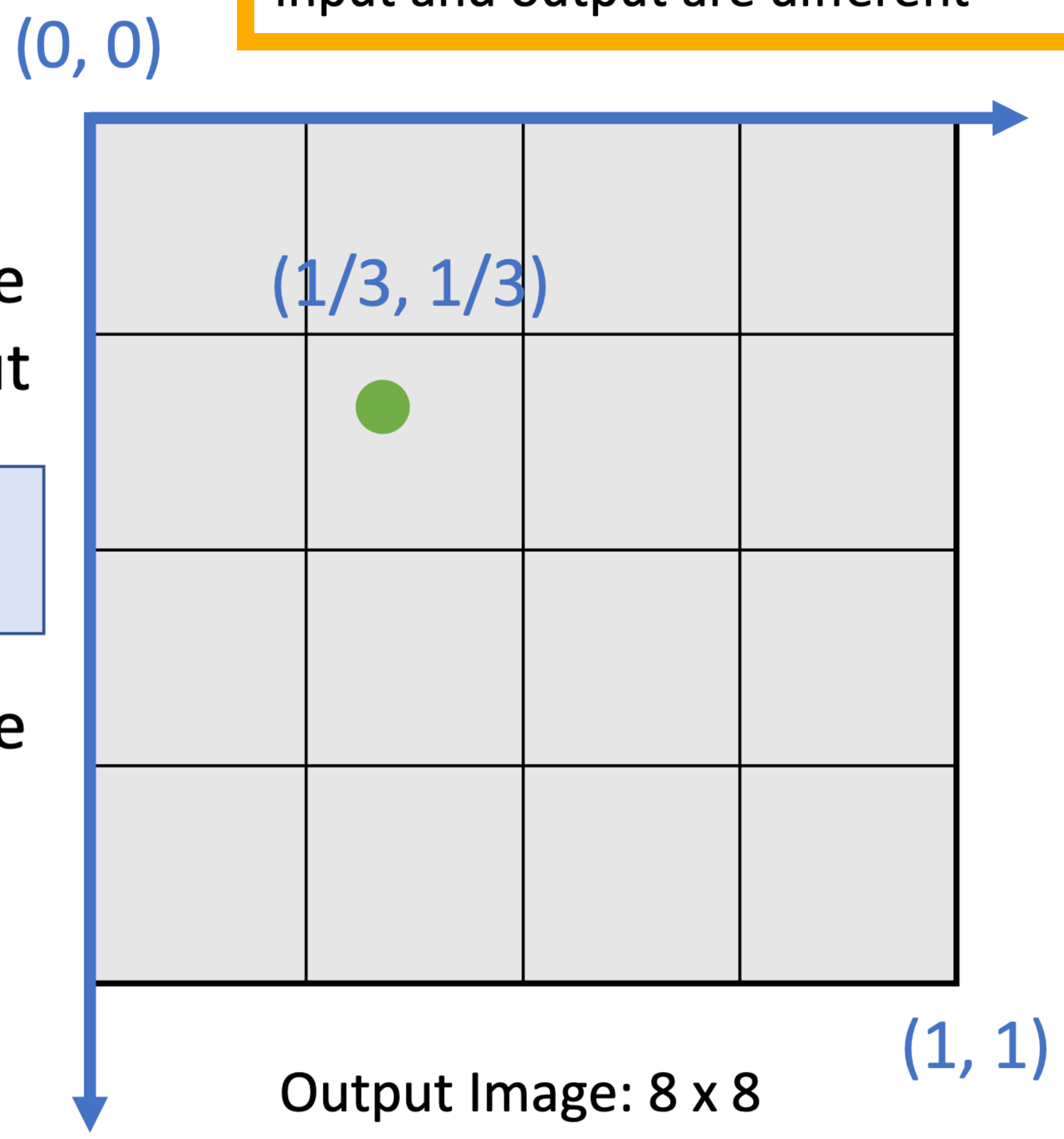



We can align arbitrary points between coordinate system of input and output

3x3 Conv
Stride 1, pad 1

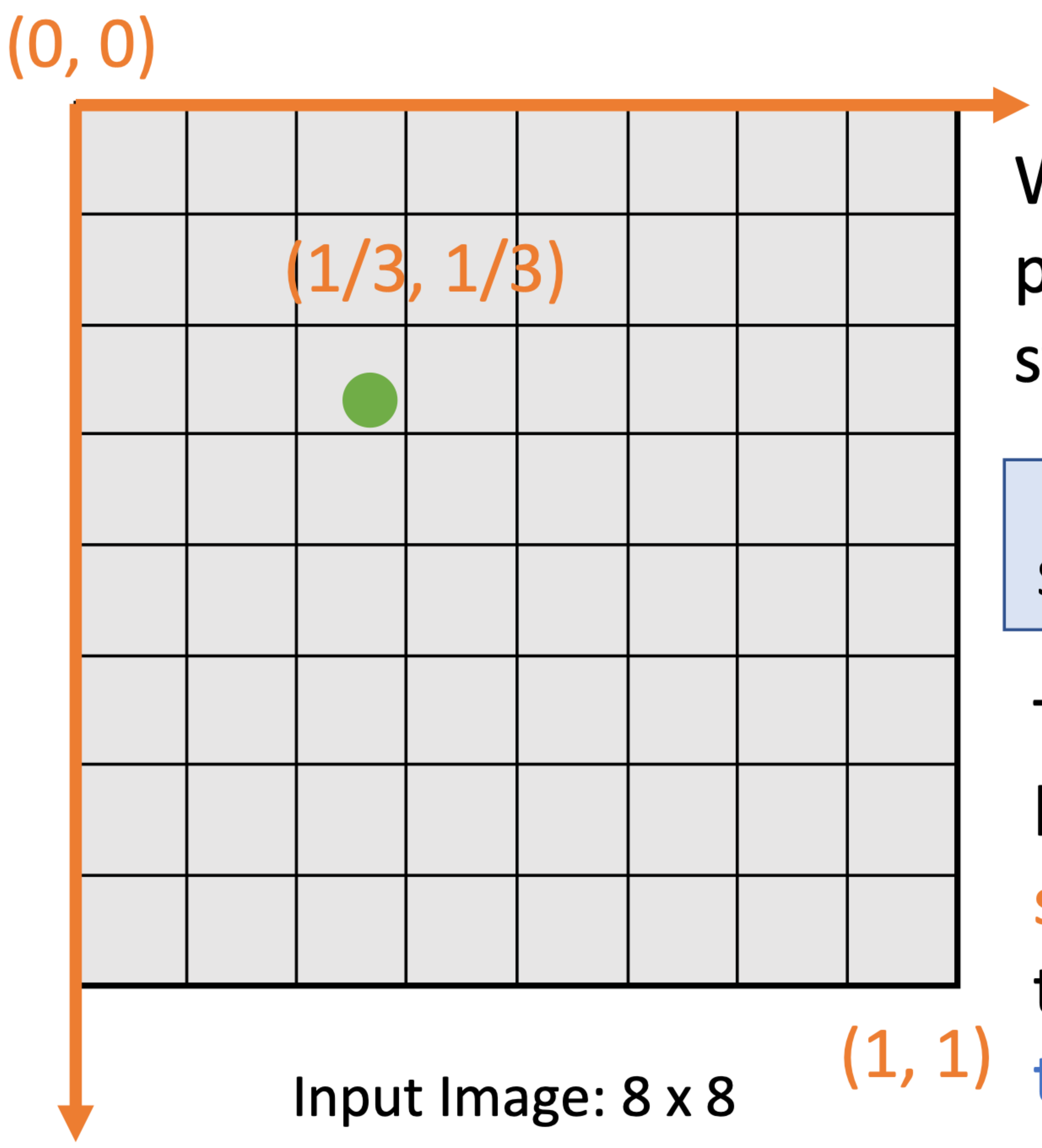
2x2 MaxPool
Stride 2

There is a correspondence between the coordinate system of the input and the coordinate system of the output



# Projecting Points

Same logic holds for more complicated CNNs, even if spatial resolution of input and output are different



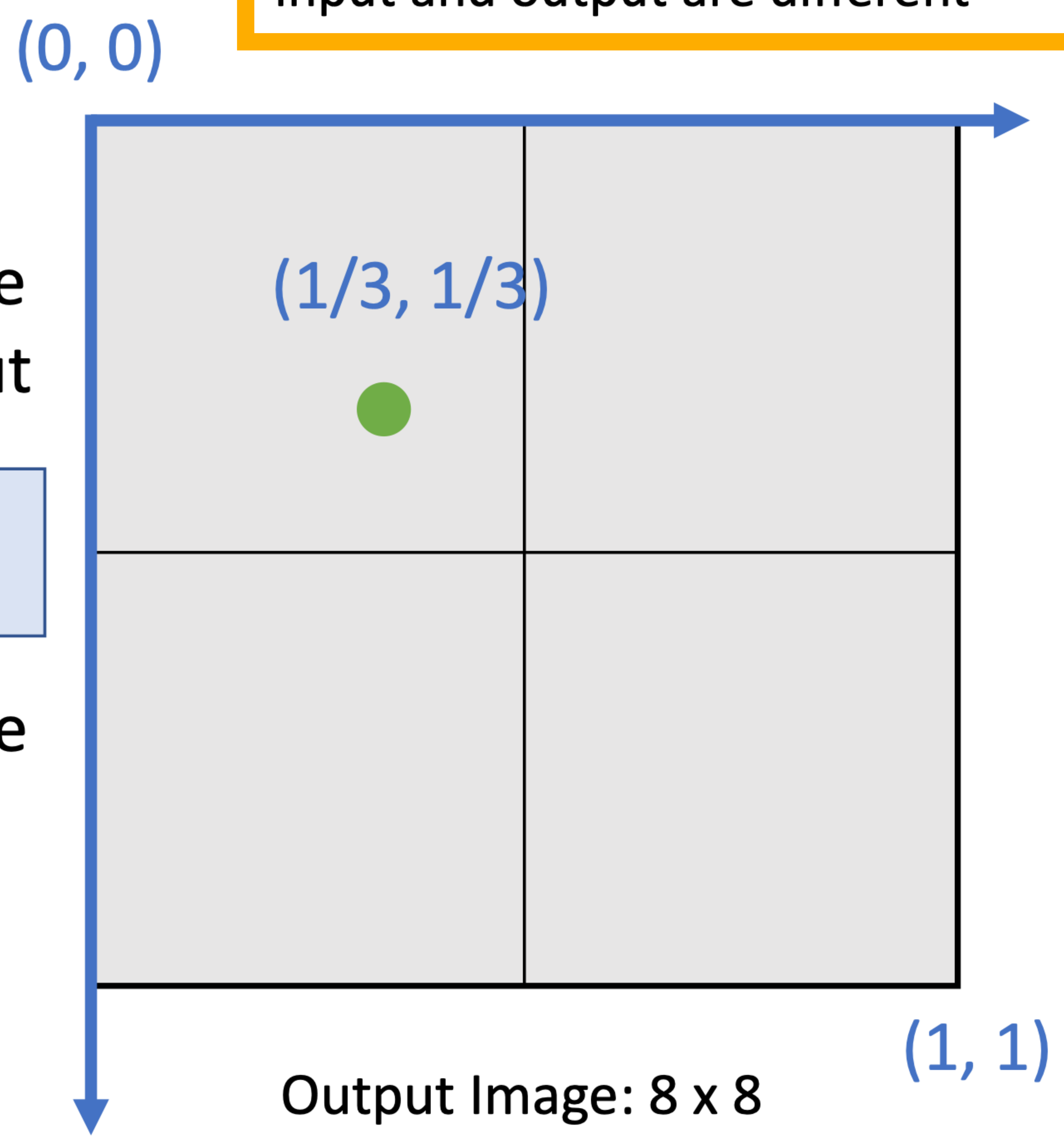

Input Image: 8 x 8

We can align arbitrary points between coordinate system of input and output

3x3 Conv
Stride 1, pad 1

4x4 MaxPool
Stride 4

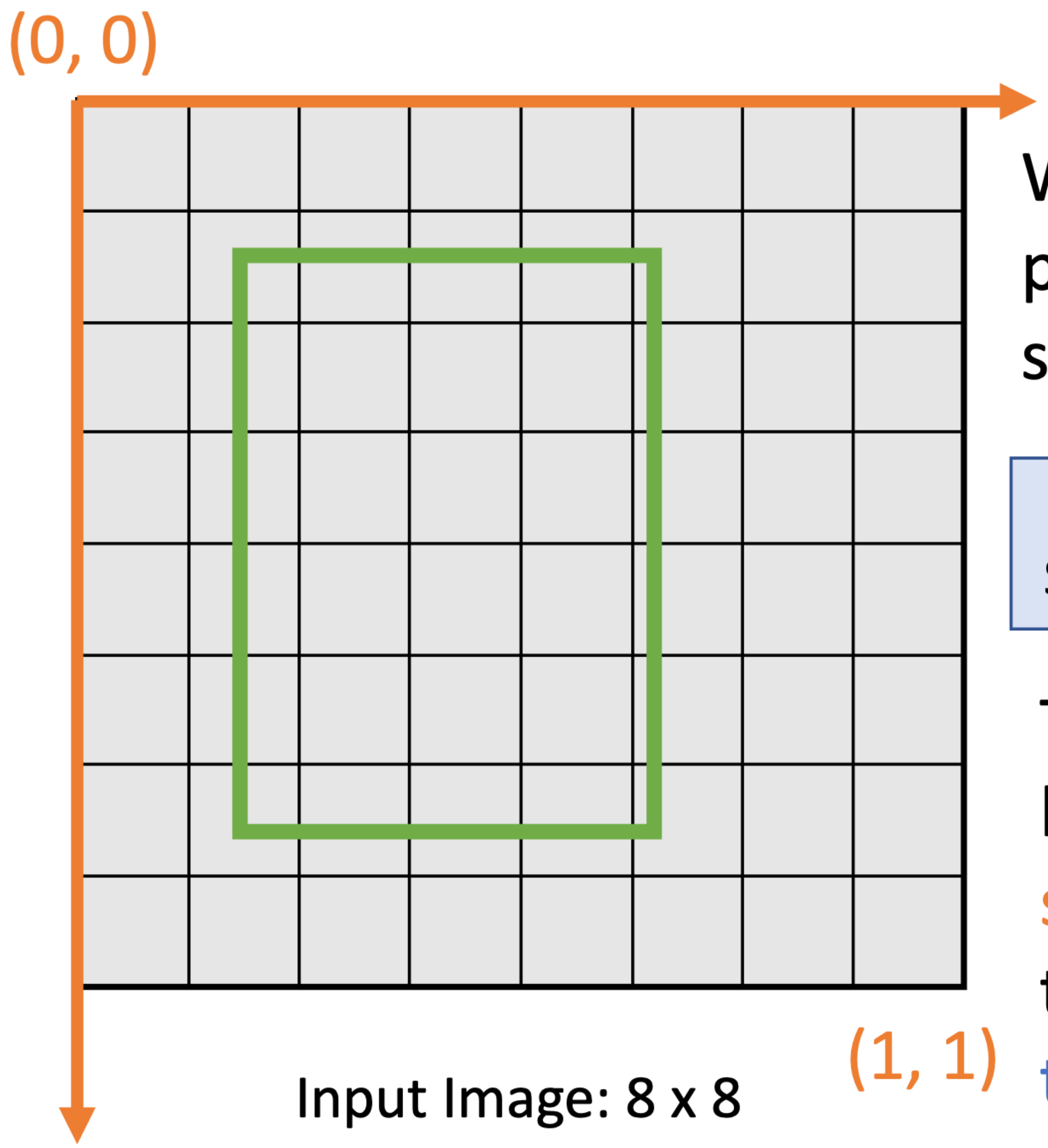
There is a correspondence between the coordinate system of the input and the coordinate system of the output




Output Image: 8 x 8

# Projecting Points

We can use this idea to project **bounding boxes** between an input image and a feature map



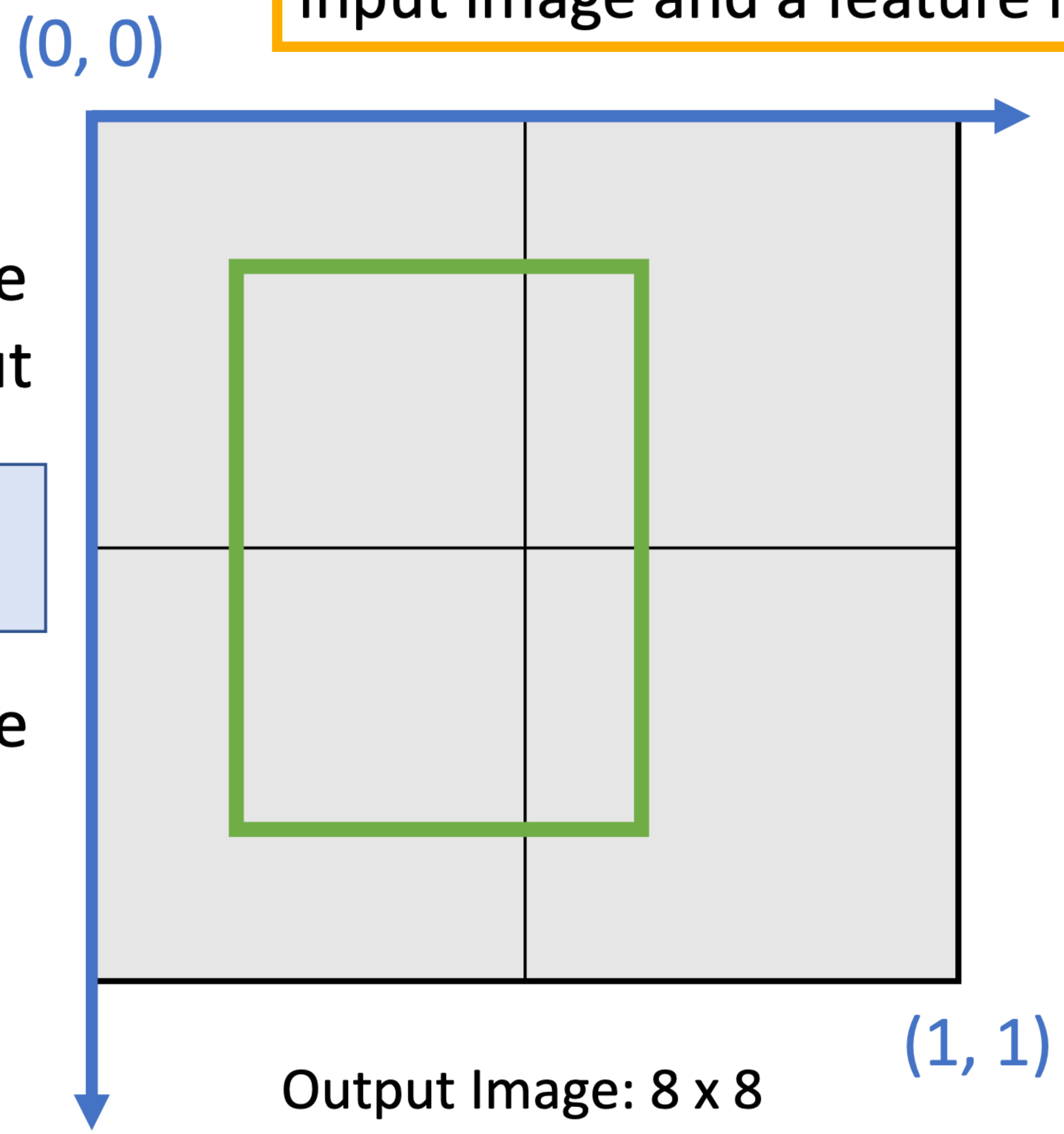

Input Image: 8 x 8

We can align arbitrary points between coordinate system of input and output

3x3 Conv
Stride 1, pad 1

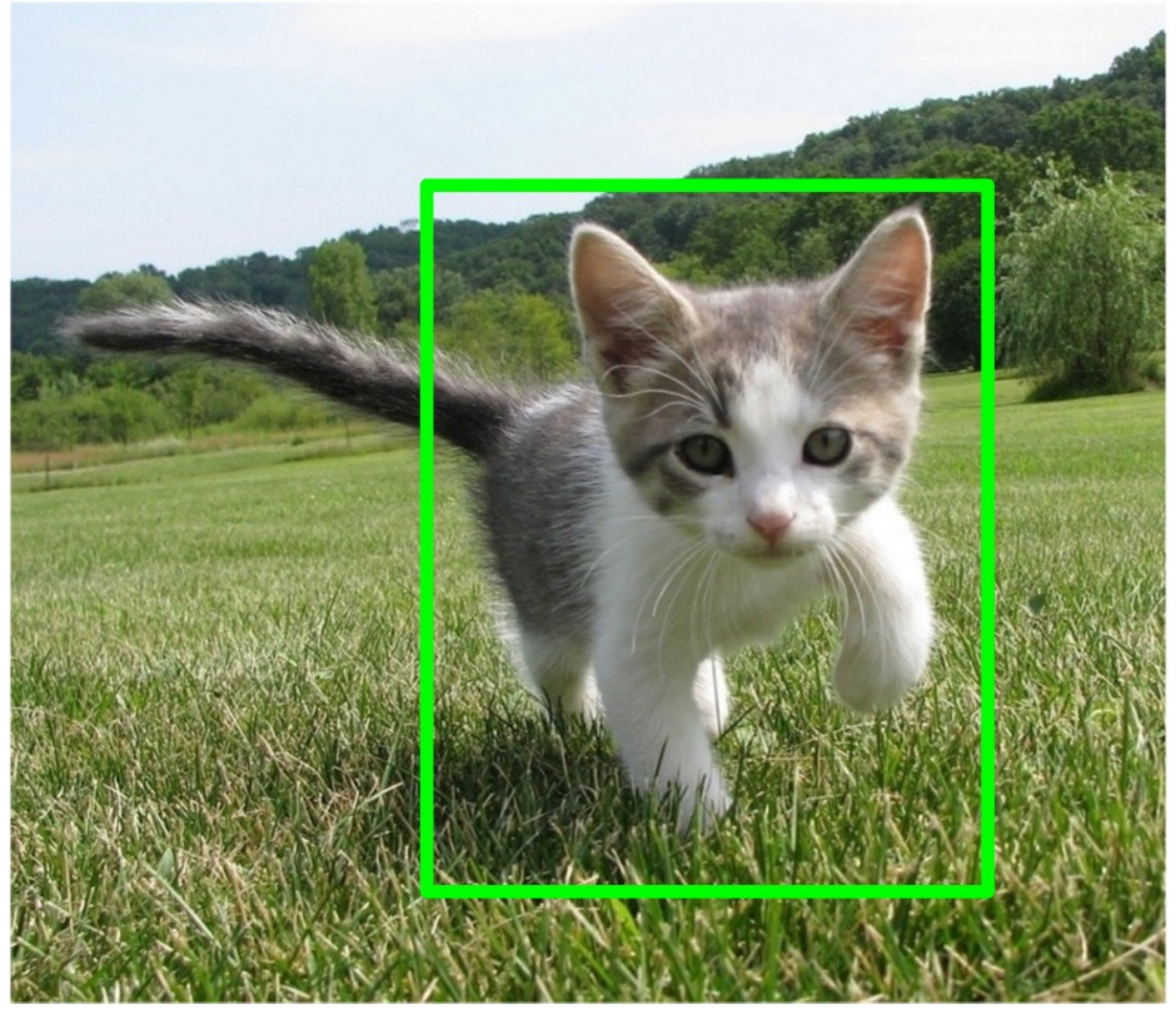
4x4 MaxPool
Stride 4

There is a correspondence between the coordinate system of the input and the coordinate system of the output




Output Image: 8 x 8

# Cropping Features: RoI Pool



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

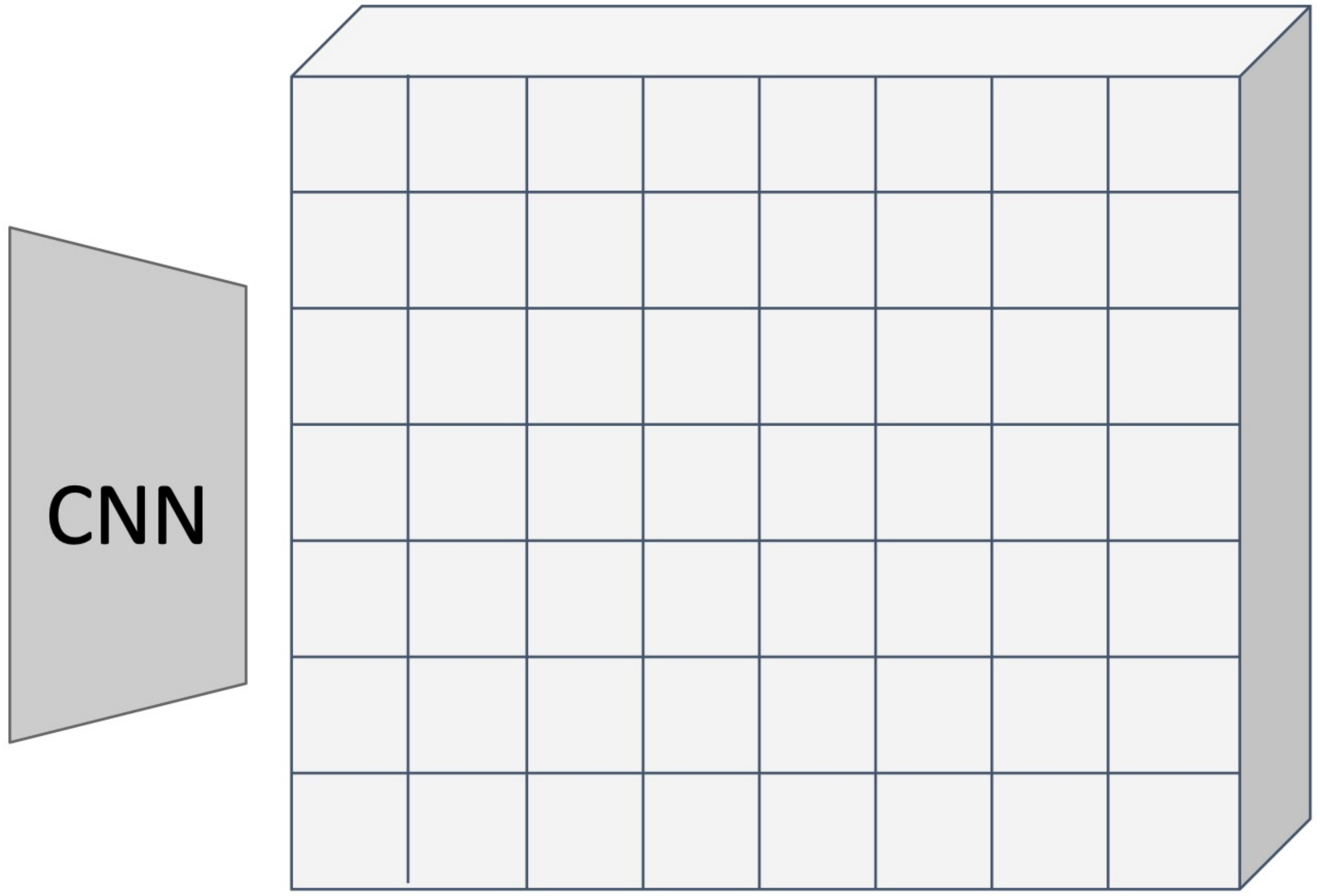
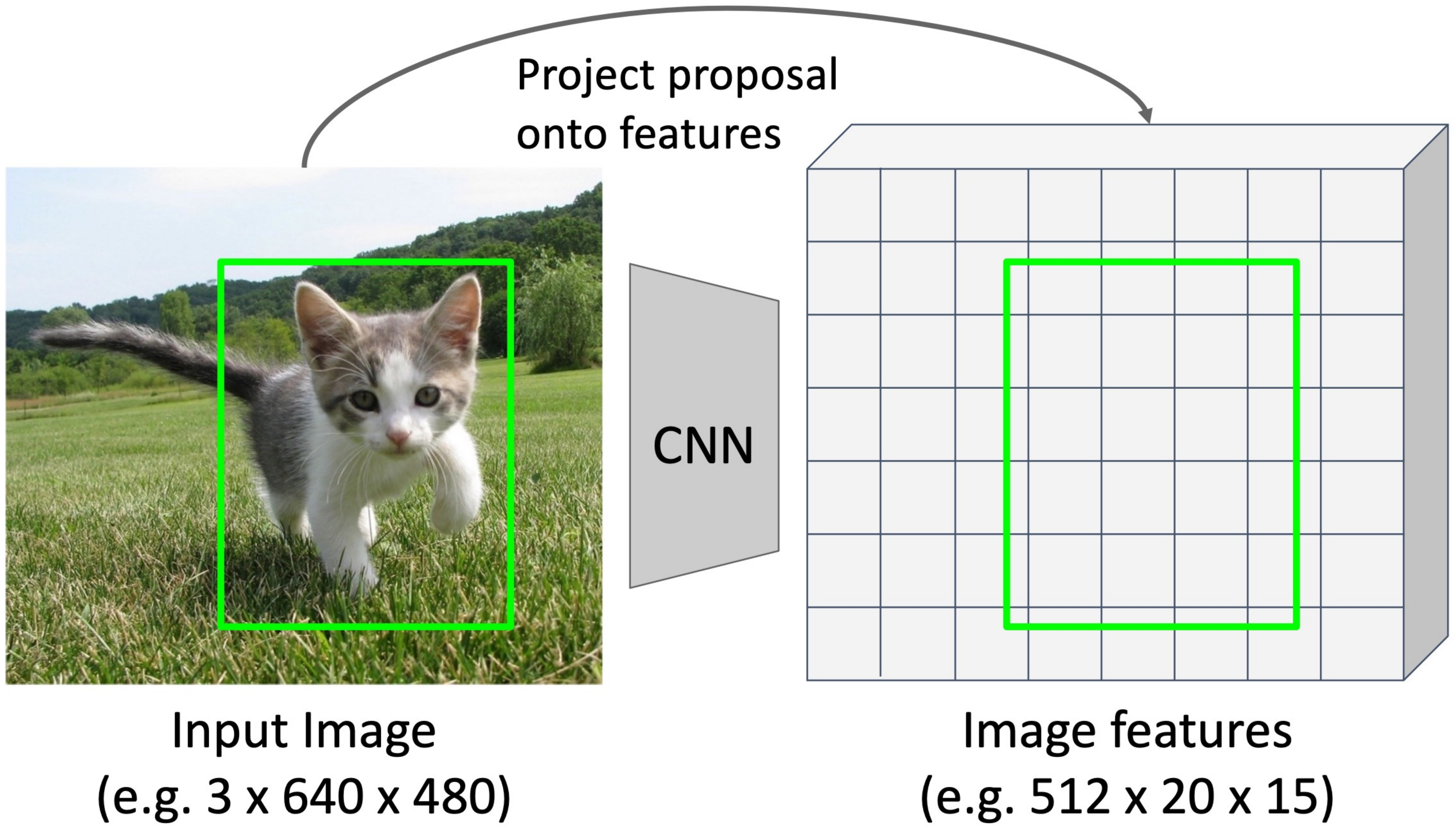



Image features
(e.g. 512 x 20 x 15)

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

Girshick, “Fast R-CNN”, ICCV 2015.

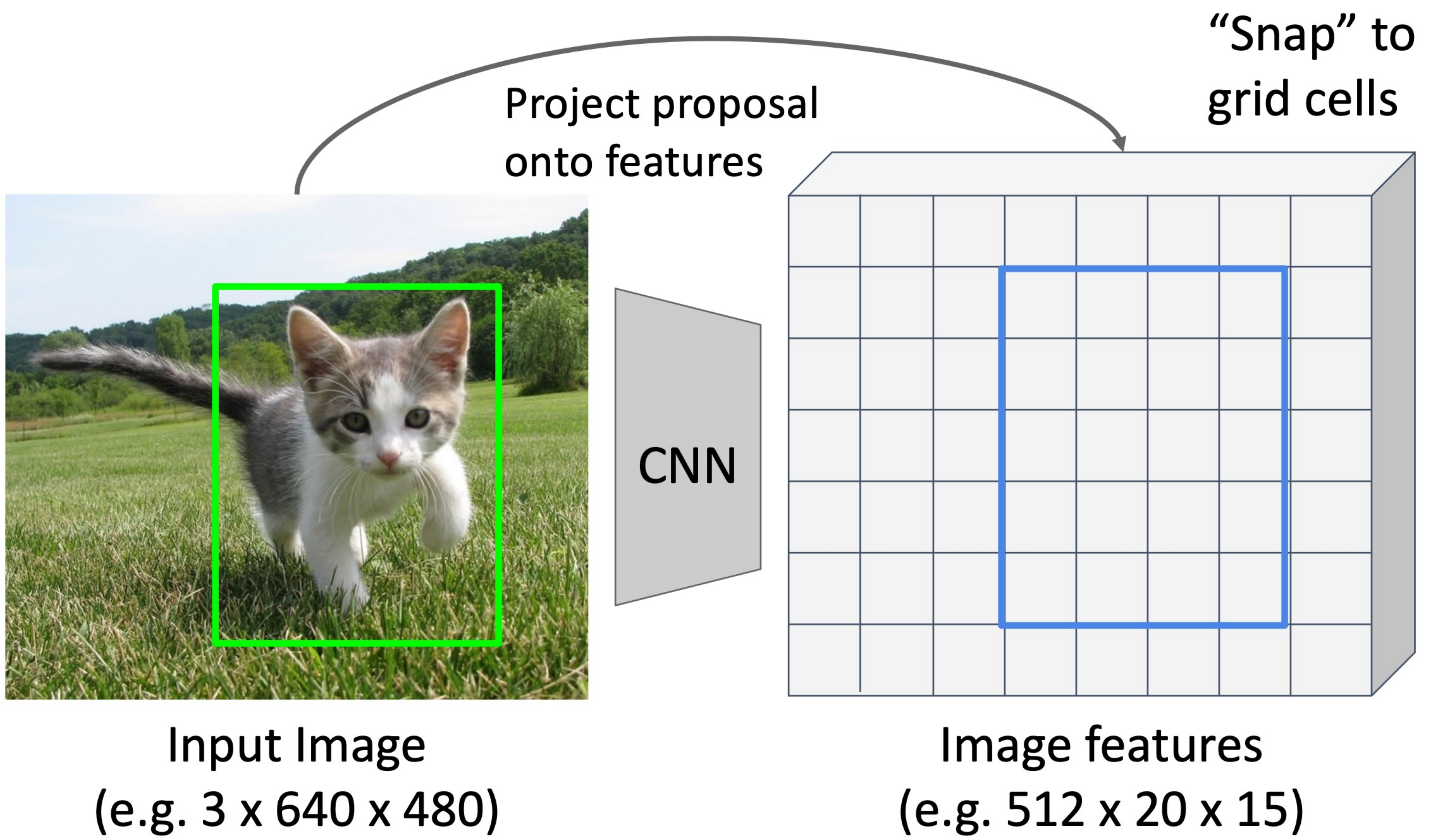
# Cropping Features: RoI Pool




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

Girshick, “Fast R-CNN”, ICCV 2015.

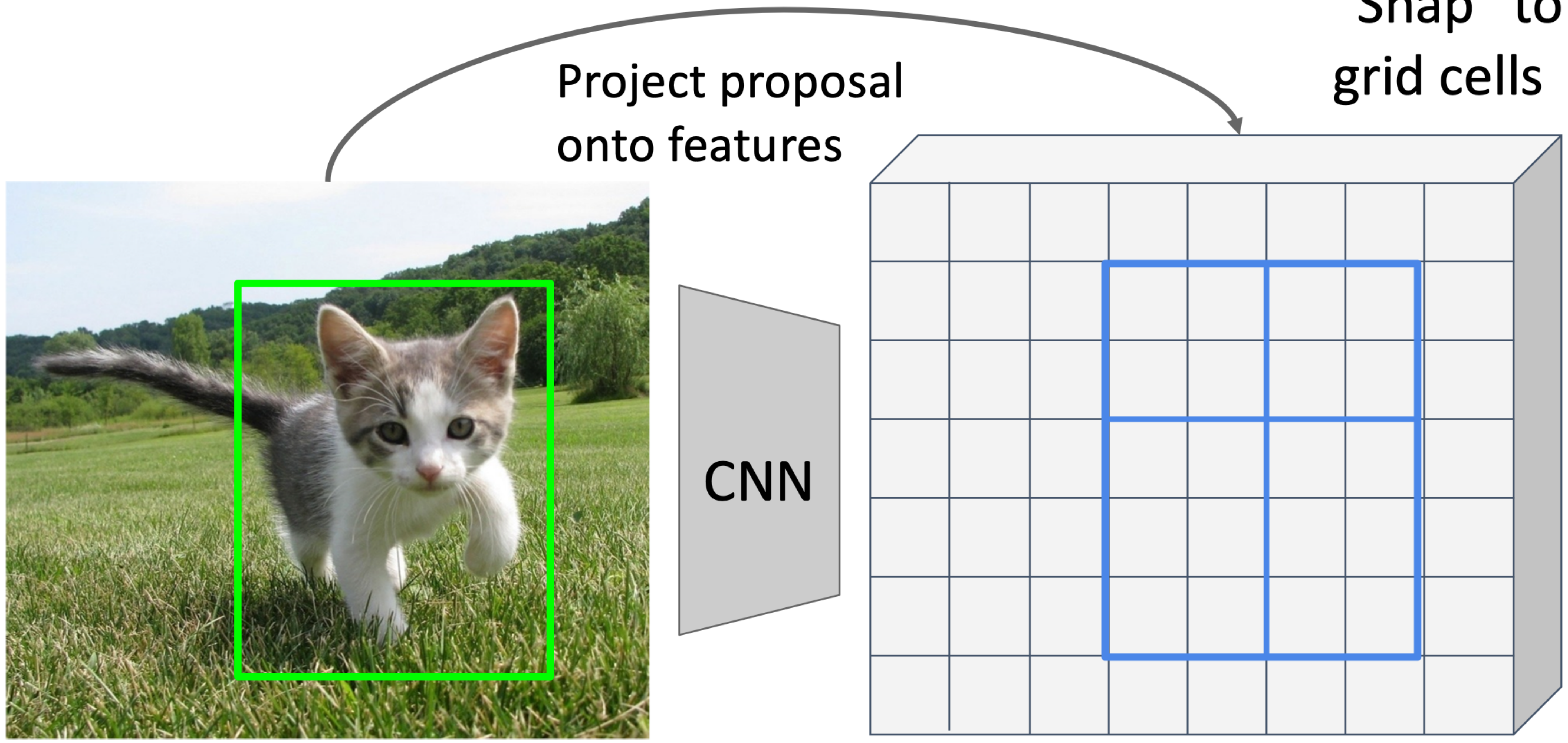
# Cropping Features: RoI Pool




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

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

# Cropping Features: RoI Pool




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

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

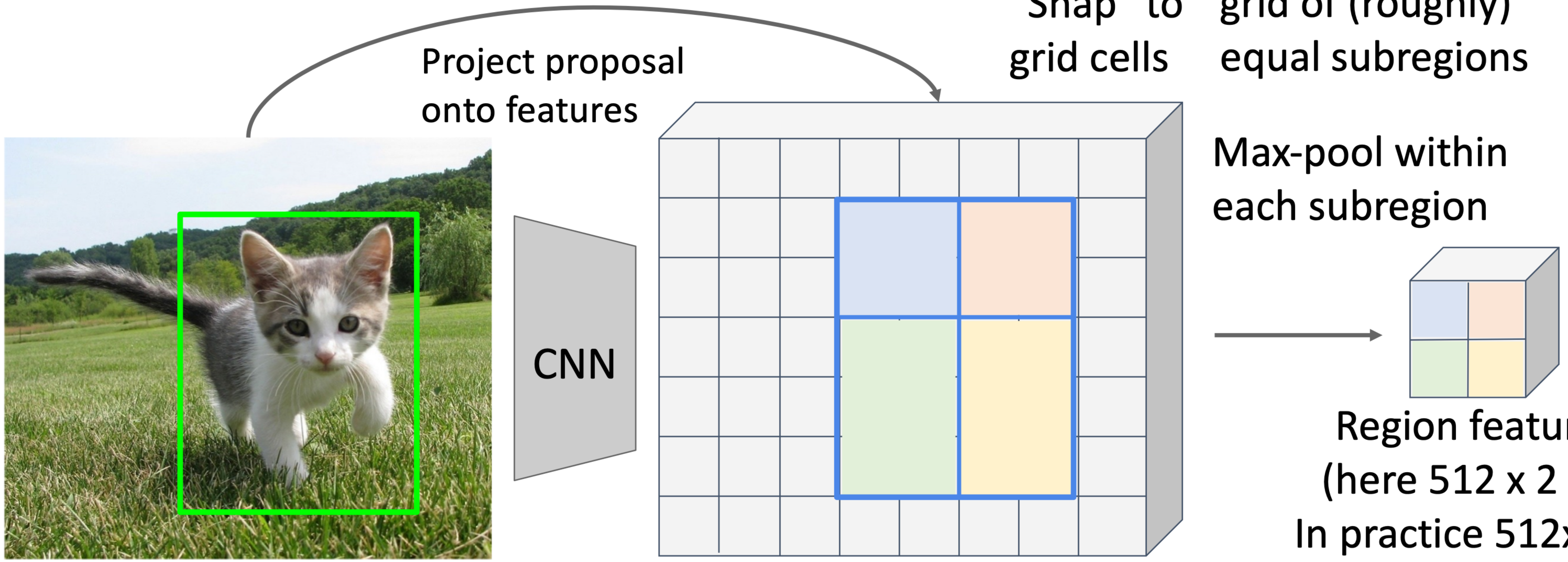
"Snap" to grid cells

Divide into 2x2 grid of (roughly) equal subregions

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

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

# Cropping Features: RoI Pool




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

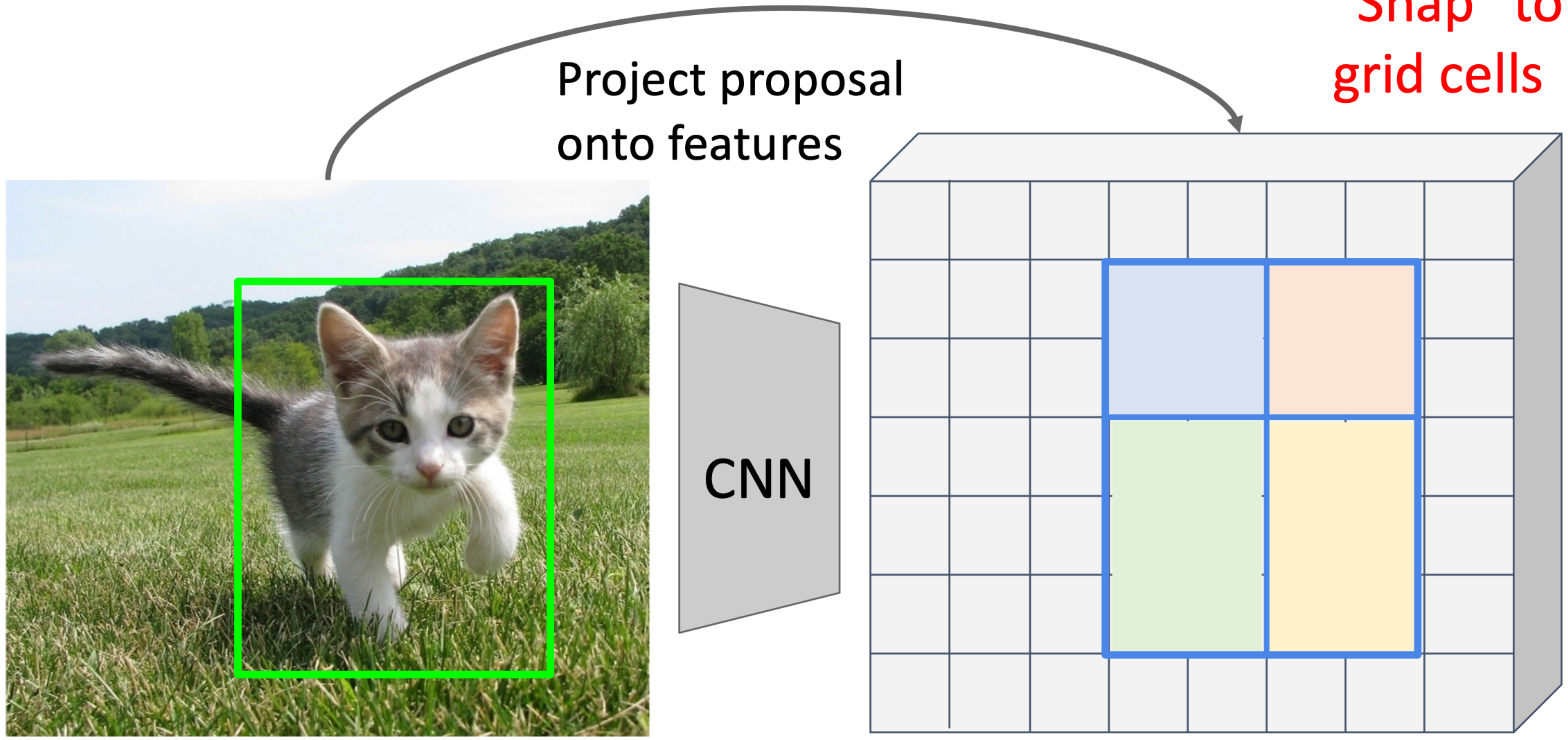
Image features
(e.g. 512 x 20 x 15)

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

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

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

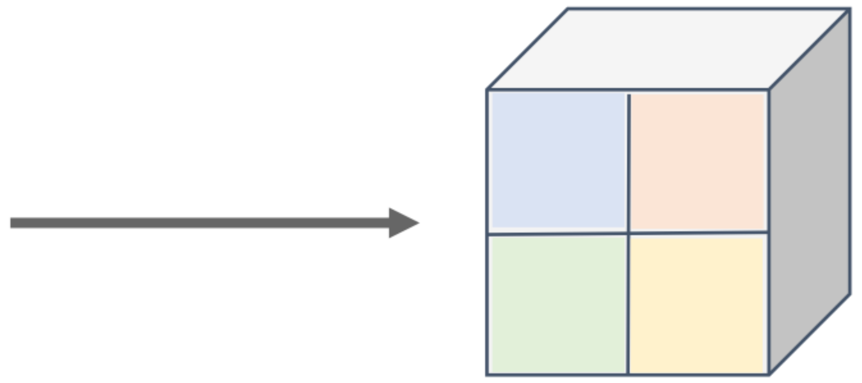
# Cropping Features: RoI Pool




"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 512x7x7)

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

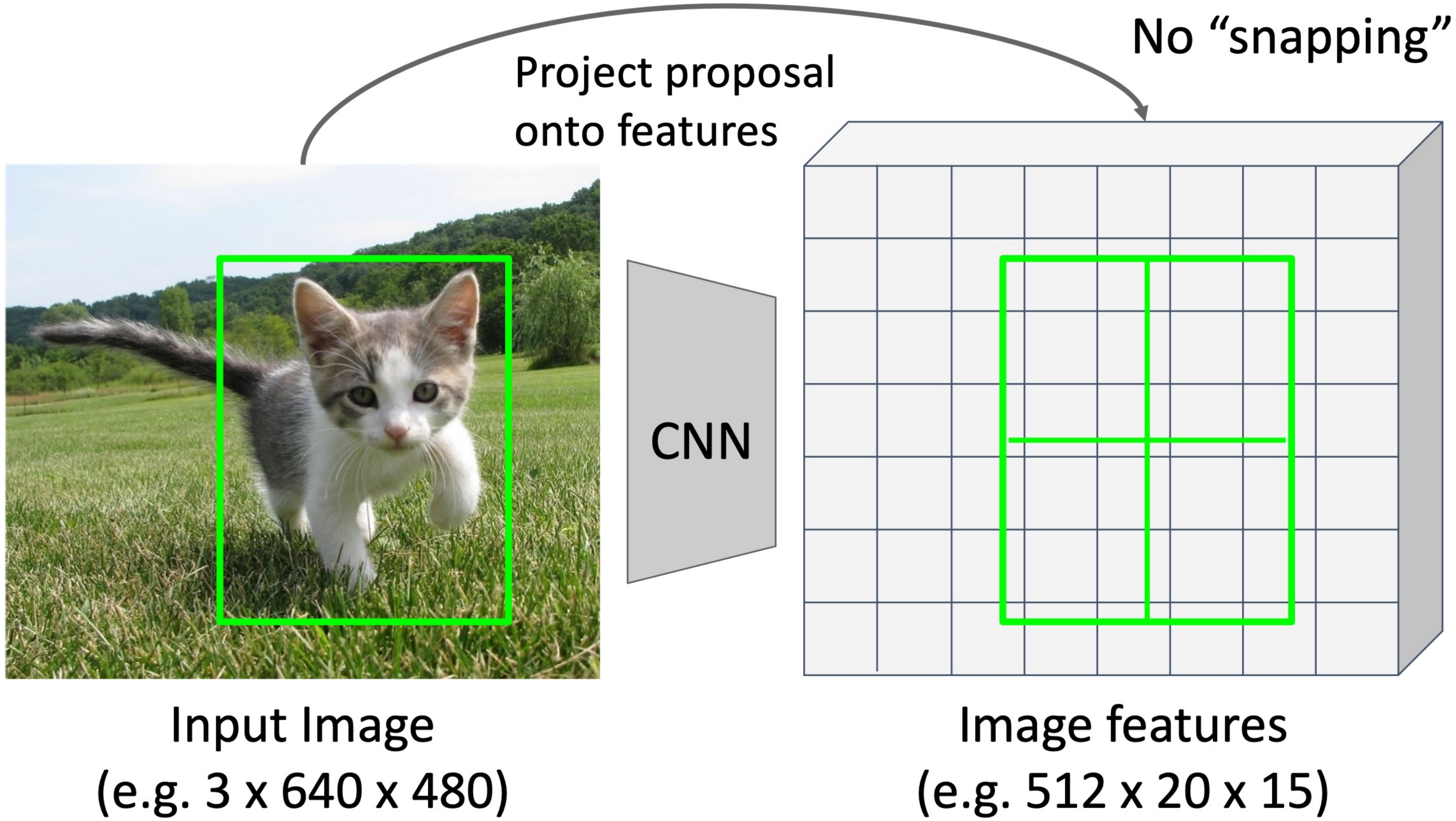
Problem: Slight misalignment due to snapping; different-sized subregions is weird

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

# Cropping Features: RoI Align

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

No "snapping"!




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

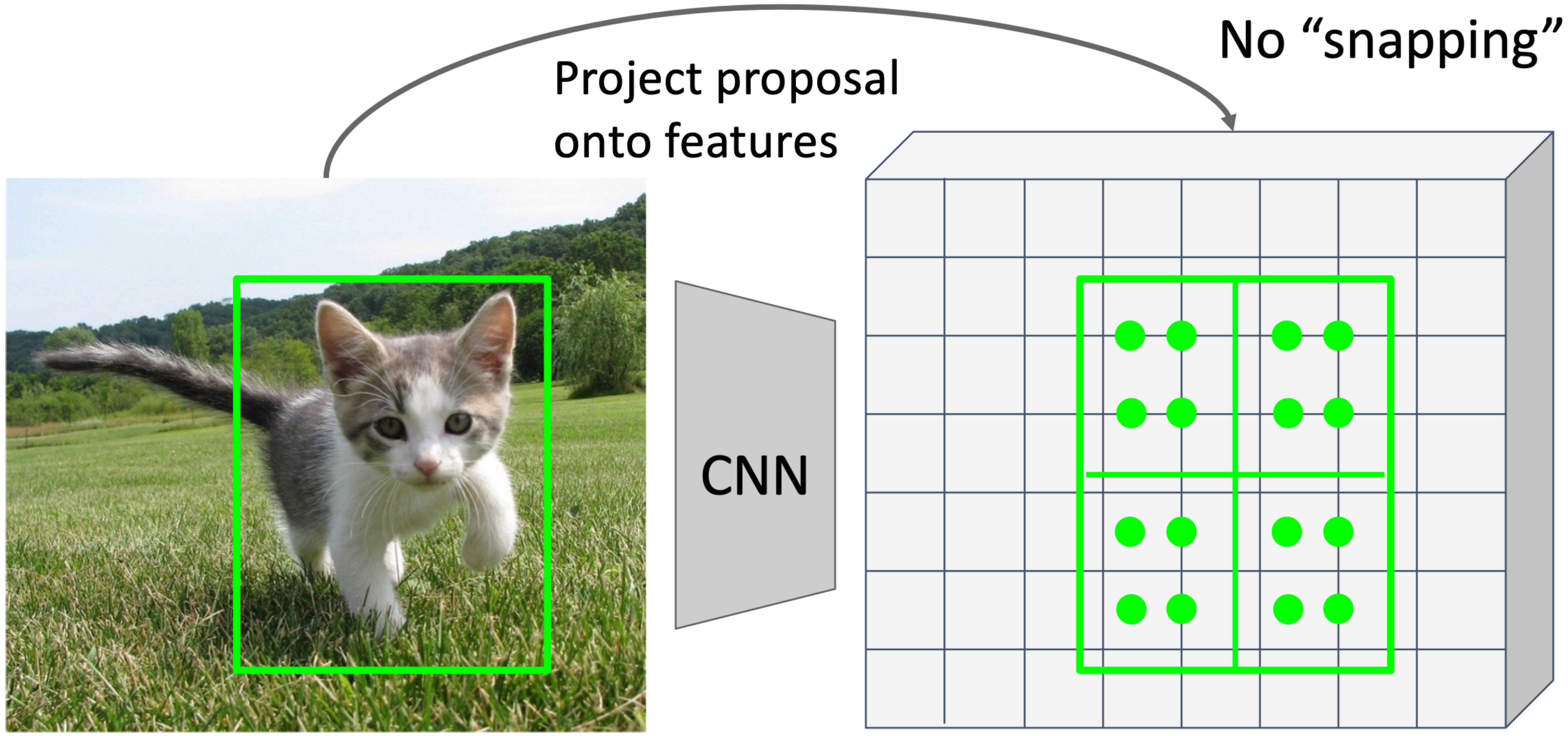
Girshick, "Fast R-CNN", ICCV 2015.

# Cropping Features: RoI 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**



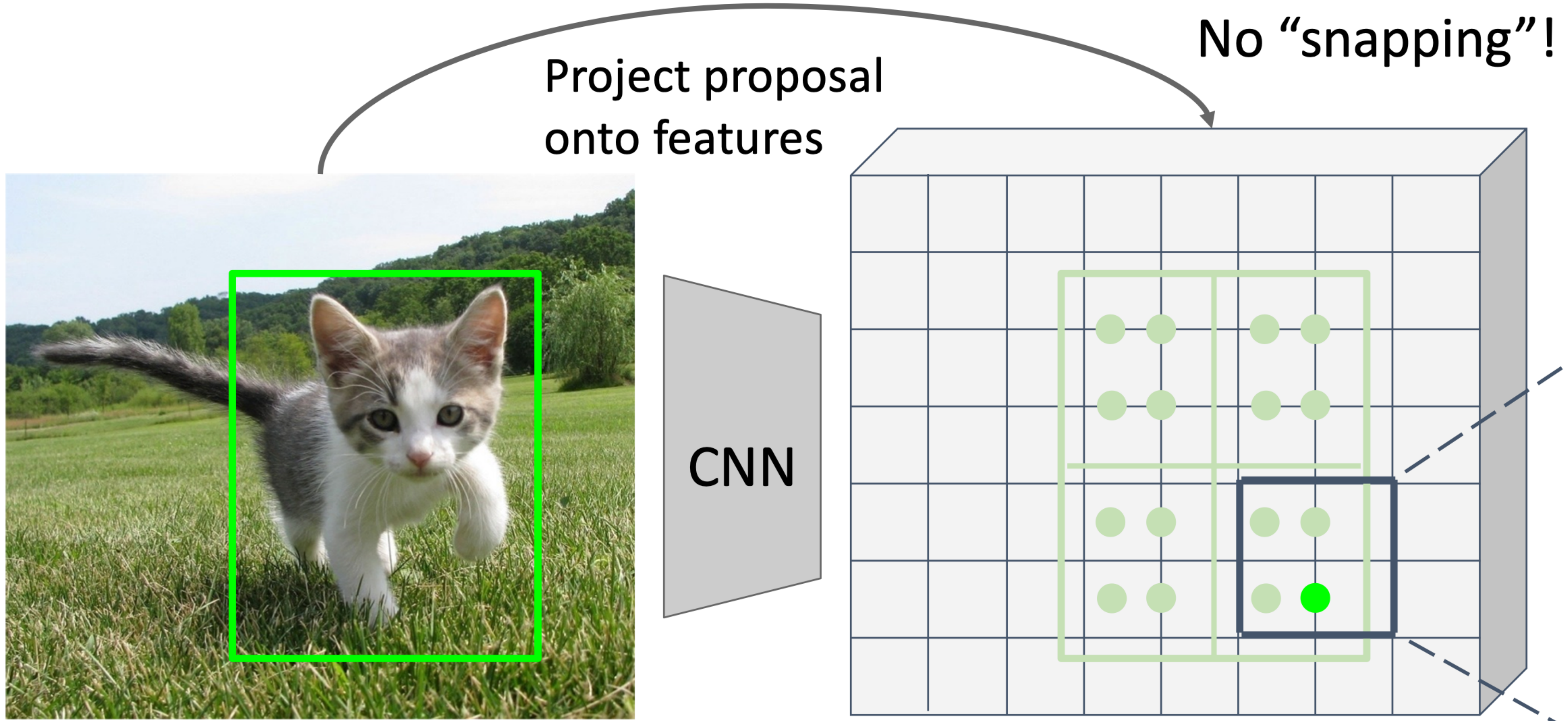

Input Image (e.g. 3 x 640 x 480)

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

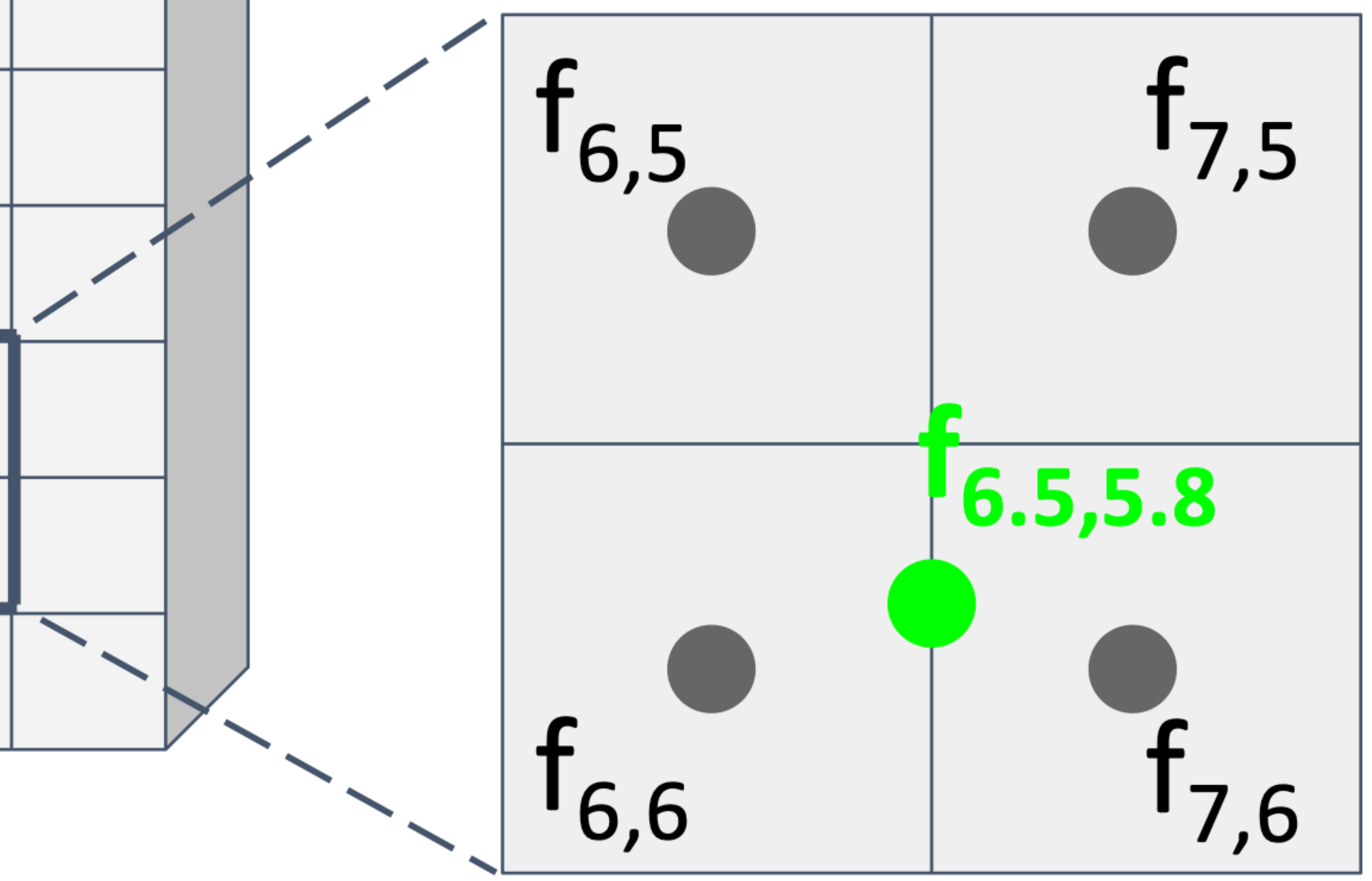
He et al, “Mask R-CNN”, ICCV 2017

# 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**




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

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

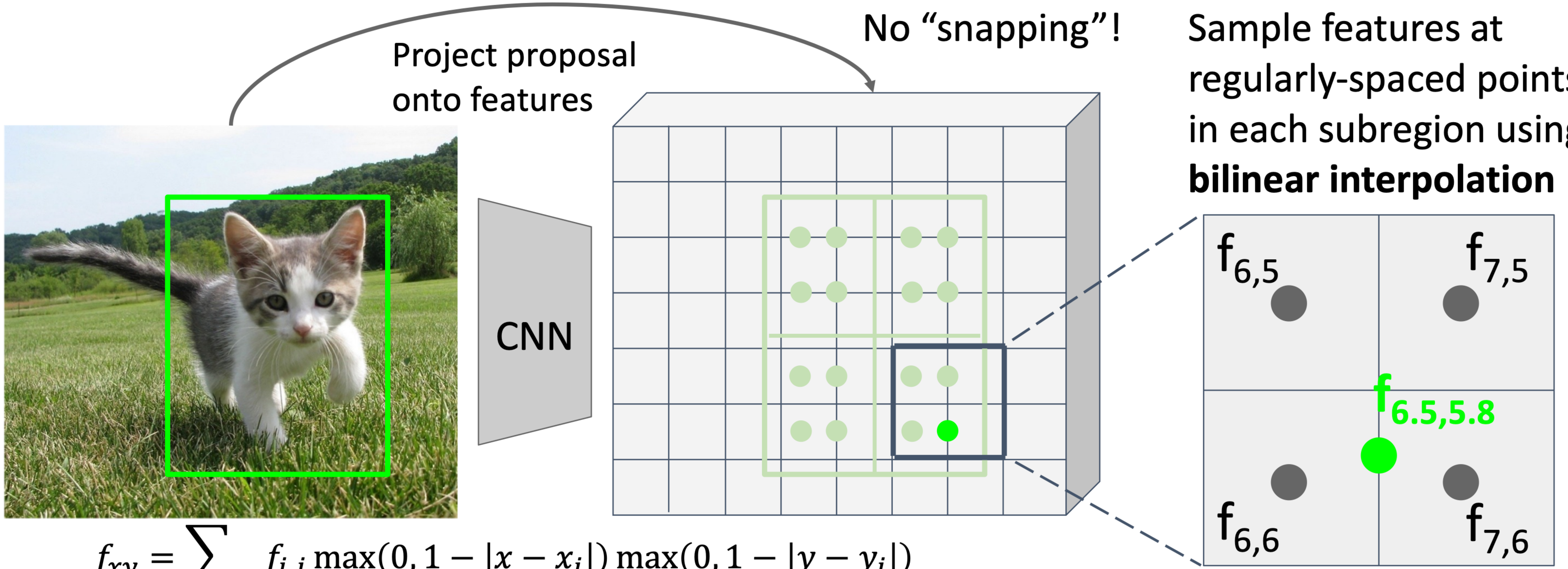
He et al, "Mask R-CNN", ICCV 2017

# 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**



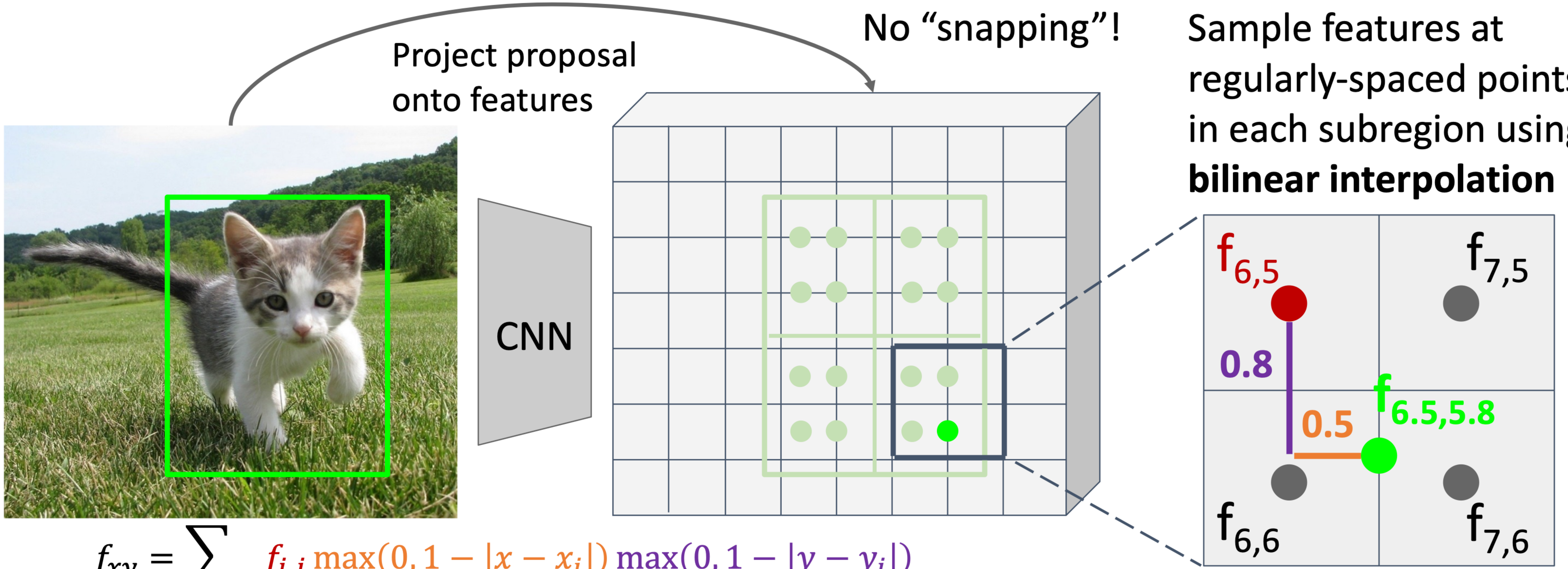

$$f_{xy} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - x_i|) \max(0, 1 - |y - y_i|)$$

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

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

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

# Cropping Features: RoI Align




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

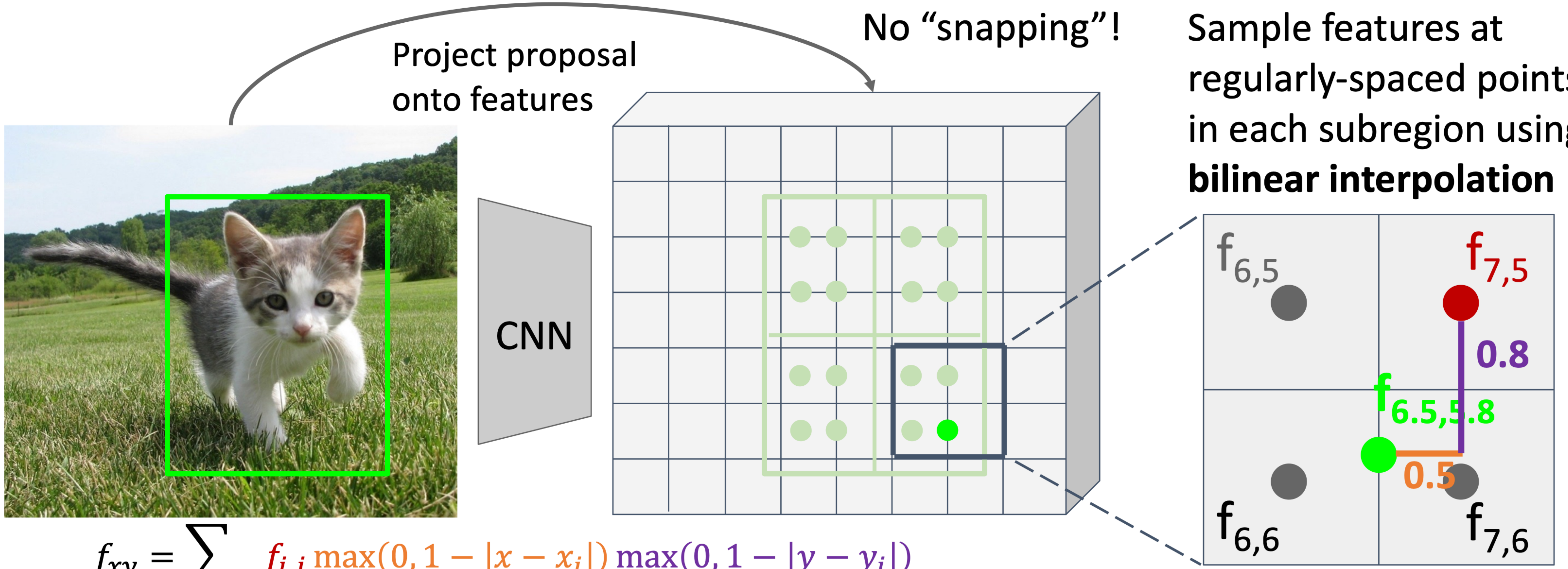
$$f_{xy} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - x_i|) \max(0, 1 - |y - y_i|)$$

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

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

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

# Cropping Features: RoI Align




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

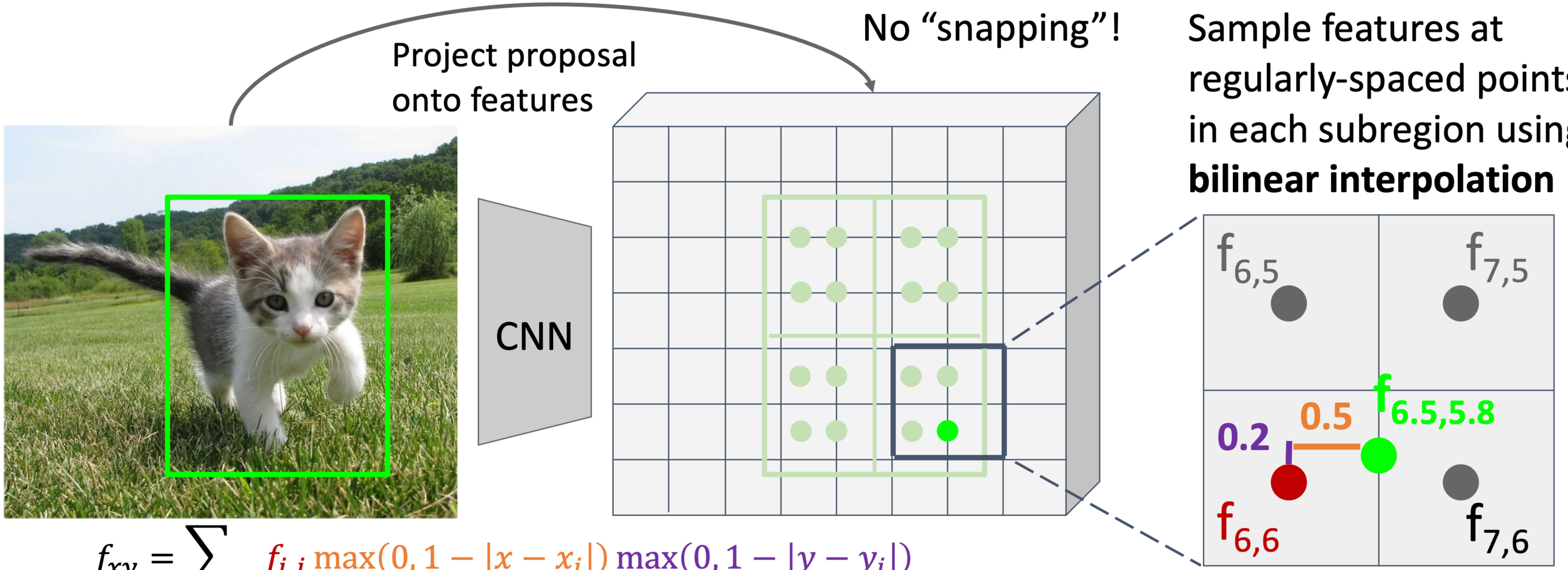
Feature $f_{xy}$ for point (x, y) is a linear combination of features at its four neighboring grid cells:

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

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

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

# Cropping Features: RoI Align




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

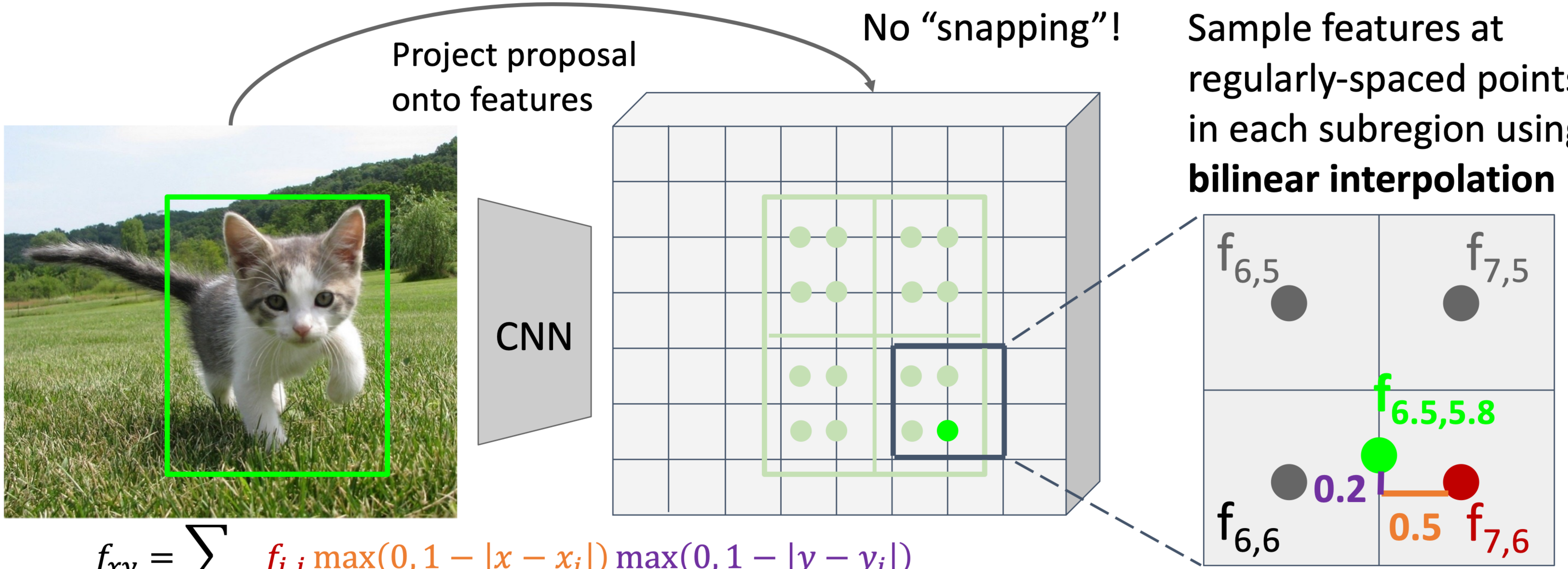
$$f_{xy} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - x_i|) \max(0, 1 - |y - y_i|)$$

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

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

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

# Cropping Features: RoI Align




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

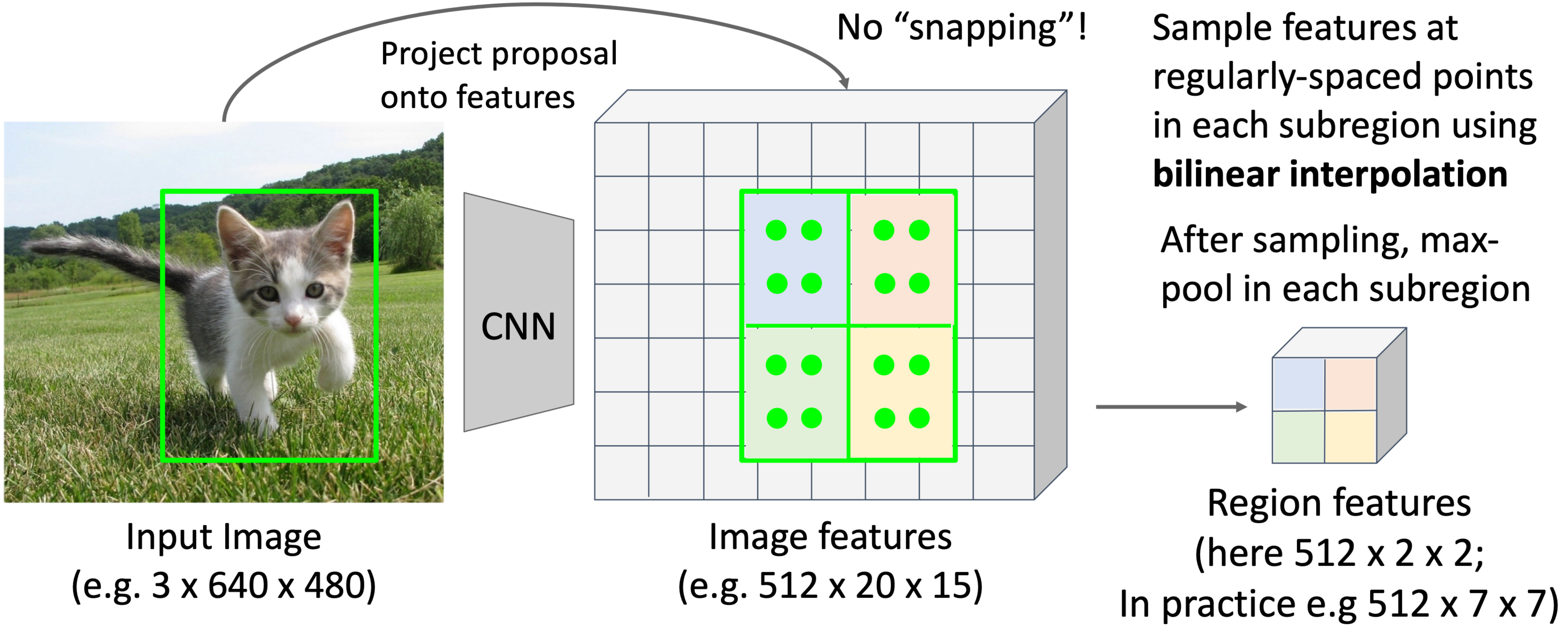
$$f_{xy} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - x_i|) \max(0, 1 - |y - y_i|)$$

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

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

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

# Cropping Features: RoI Align



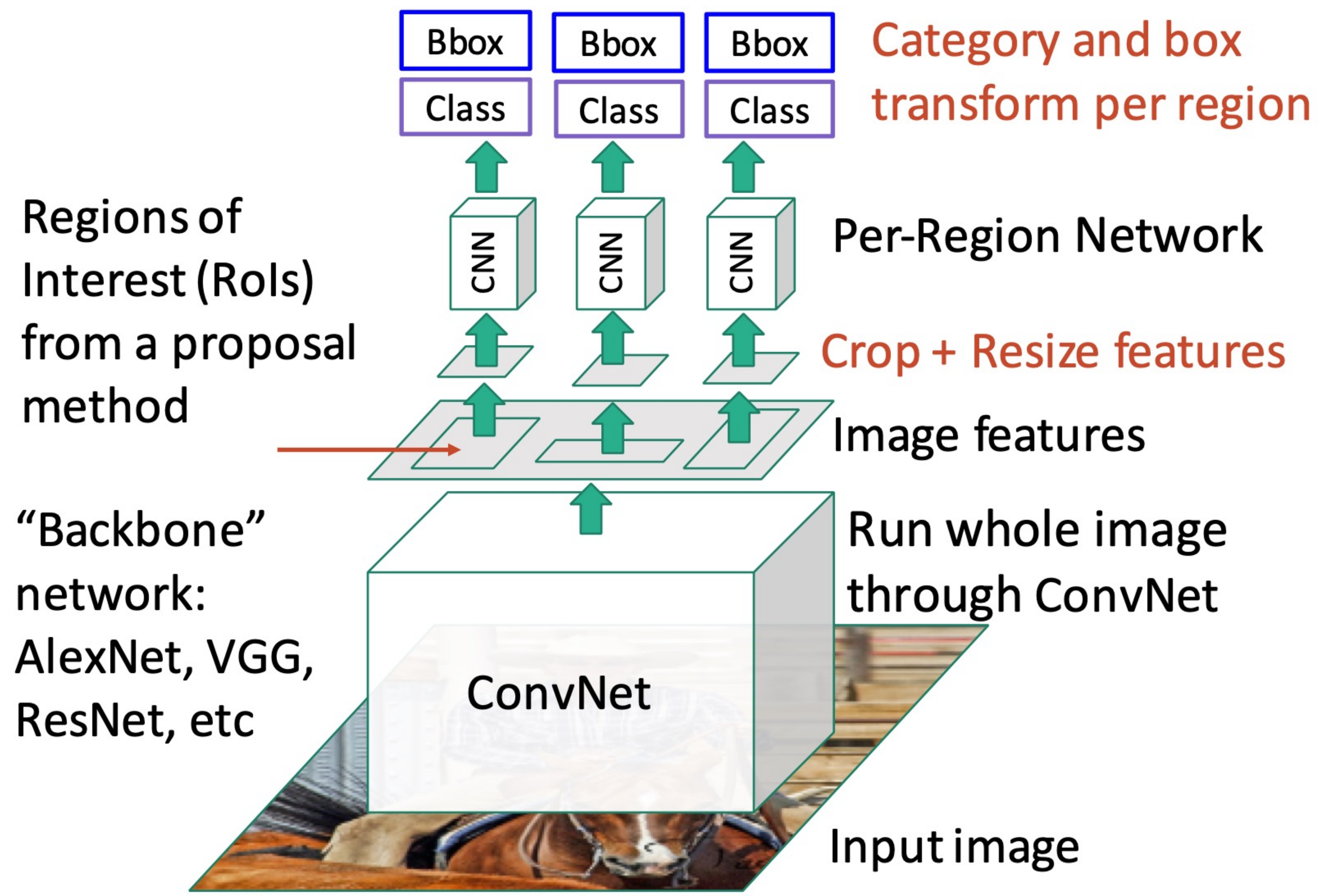

Sample features at regularly-spaced points in each subregion using **bilinear interpolation**

After sampling, max-pool in each subregion

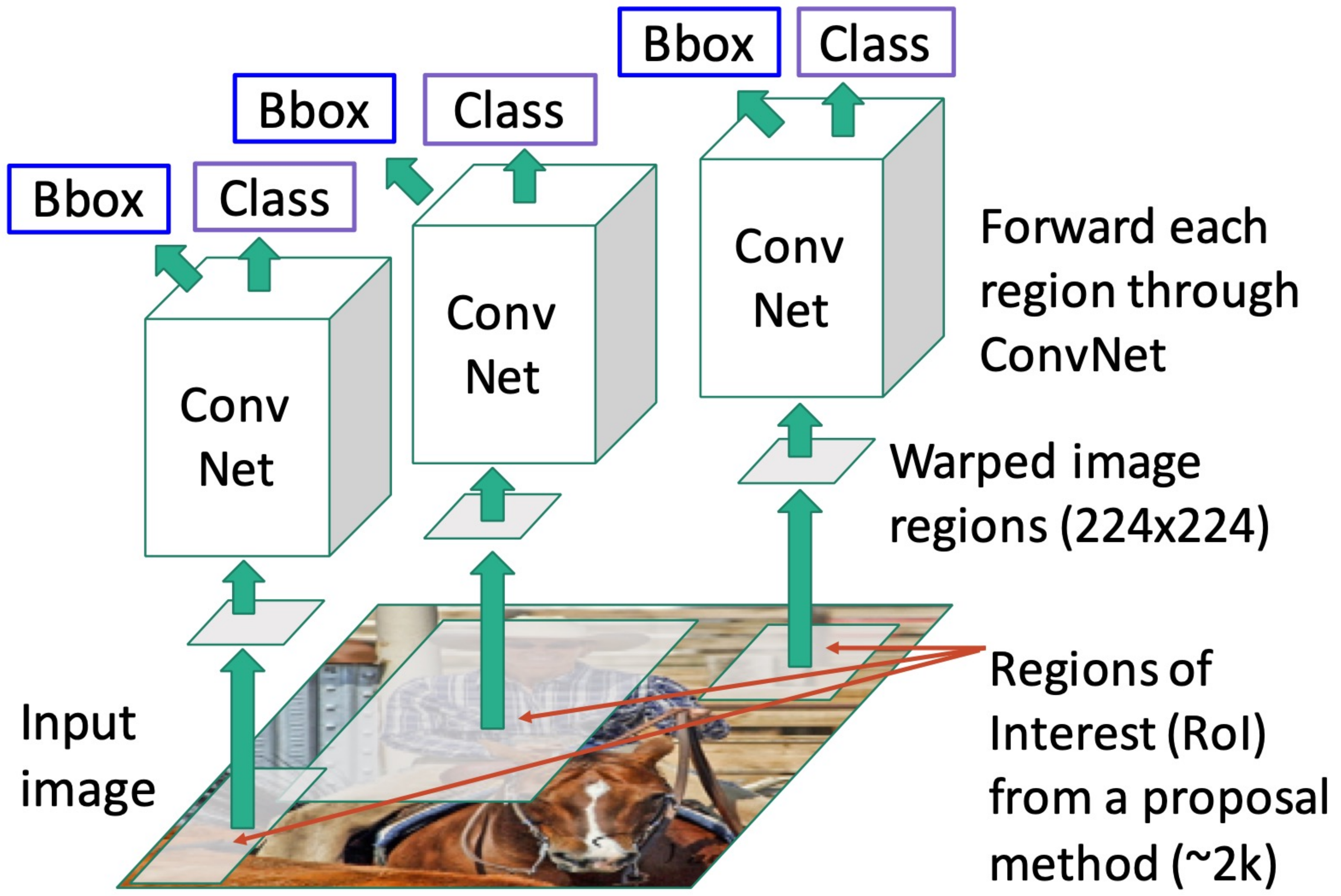
He et al, "Mask R-CNN", ICCV 2017

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

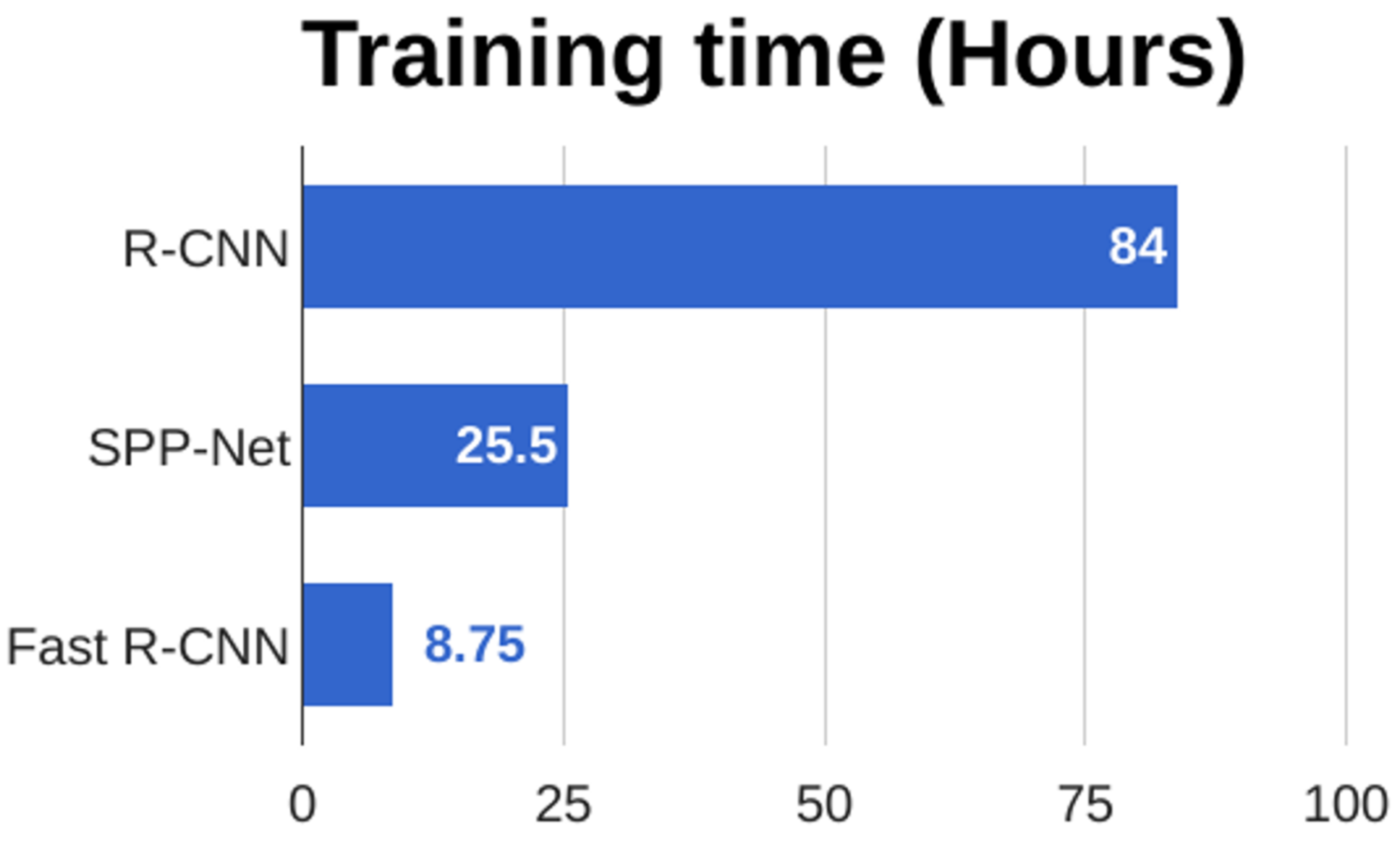
**Fast R-CNN:** Apply differentiable cropping to shared image features

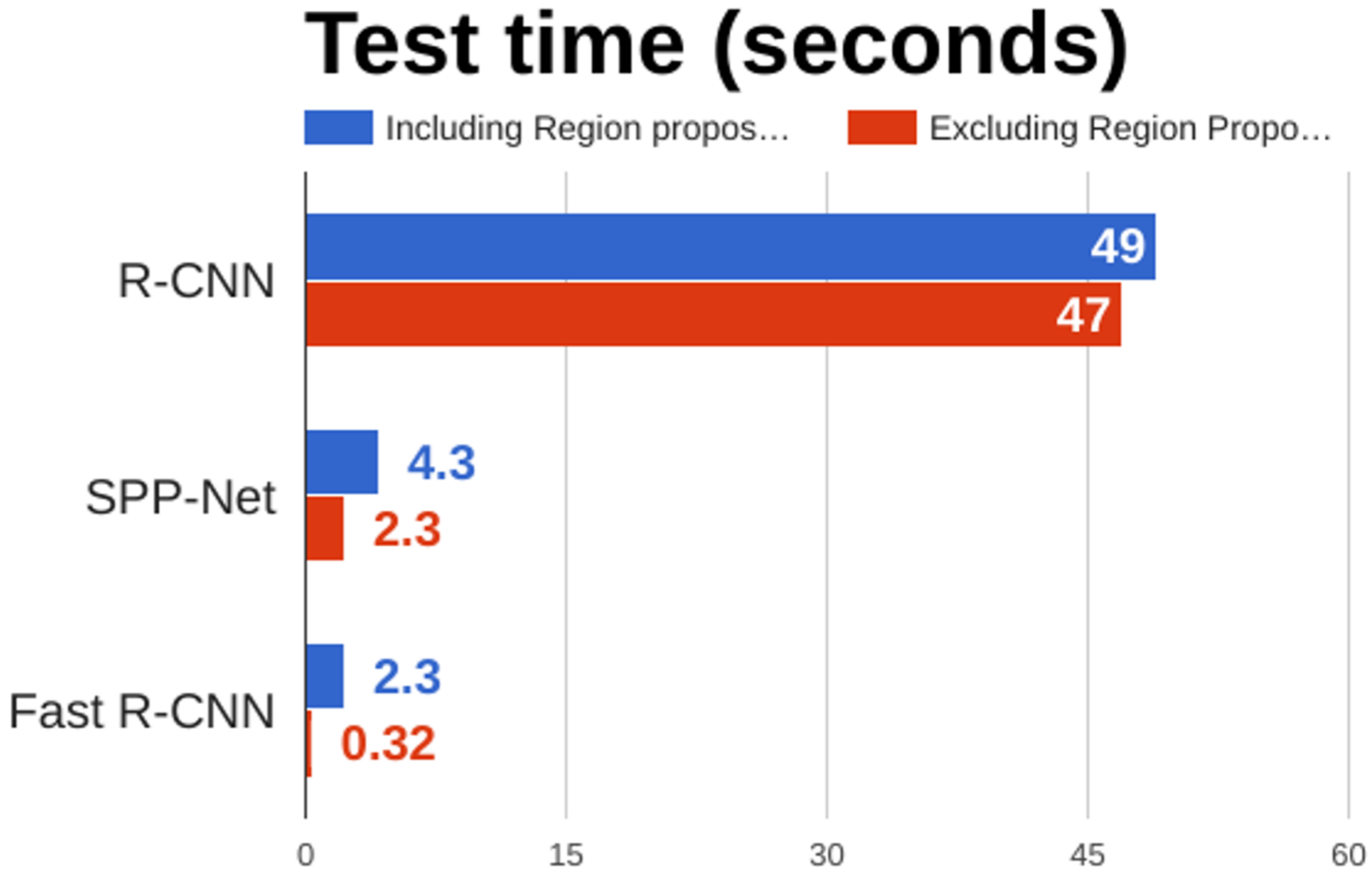



**"Slow" R-CNN:** Apply differentiable cropping to shared image features



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

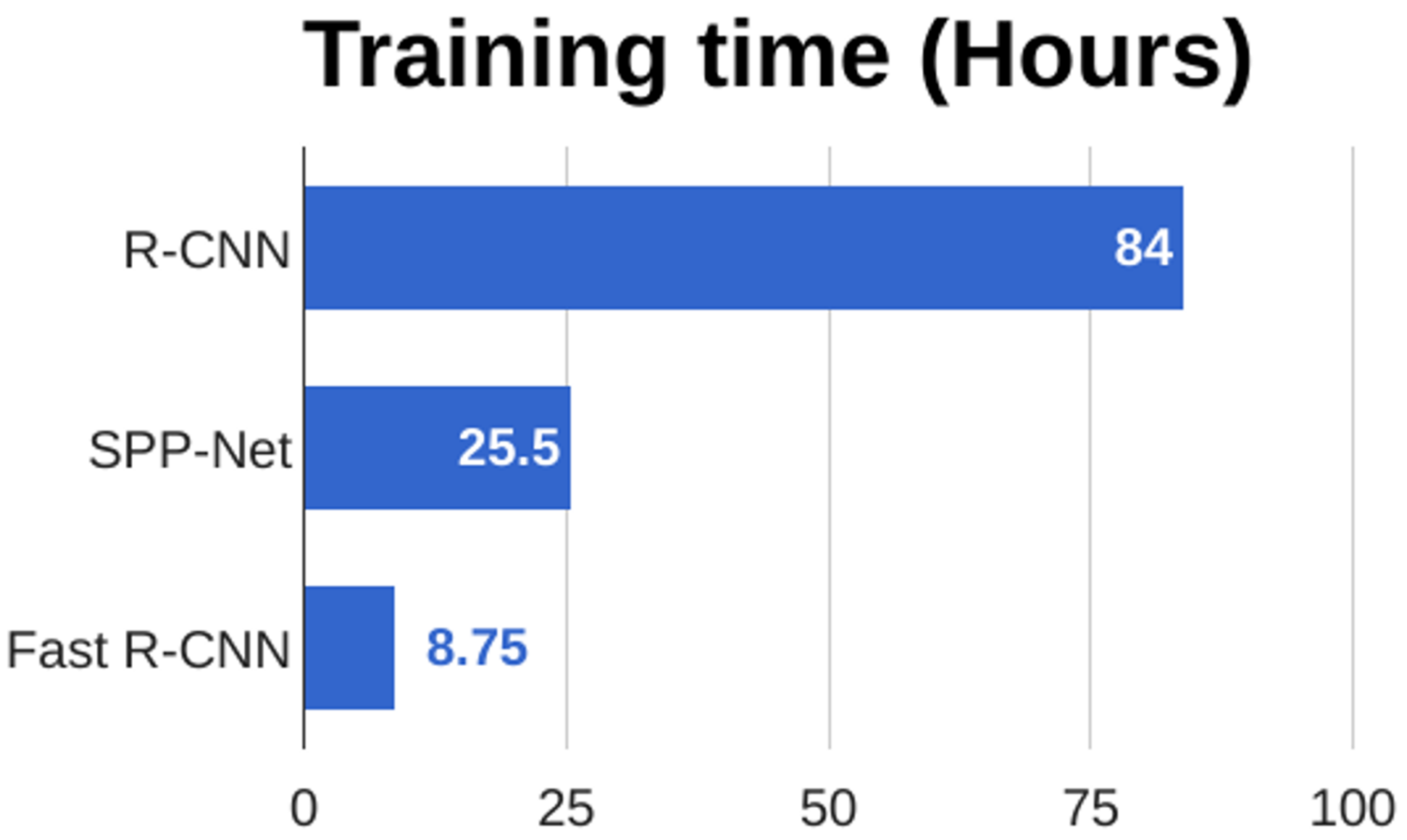


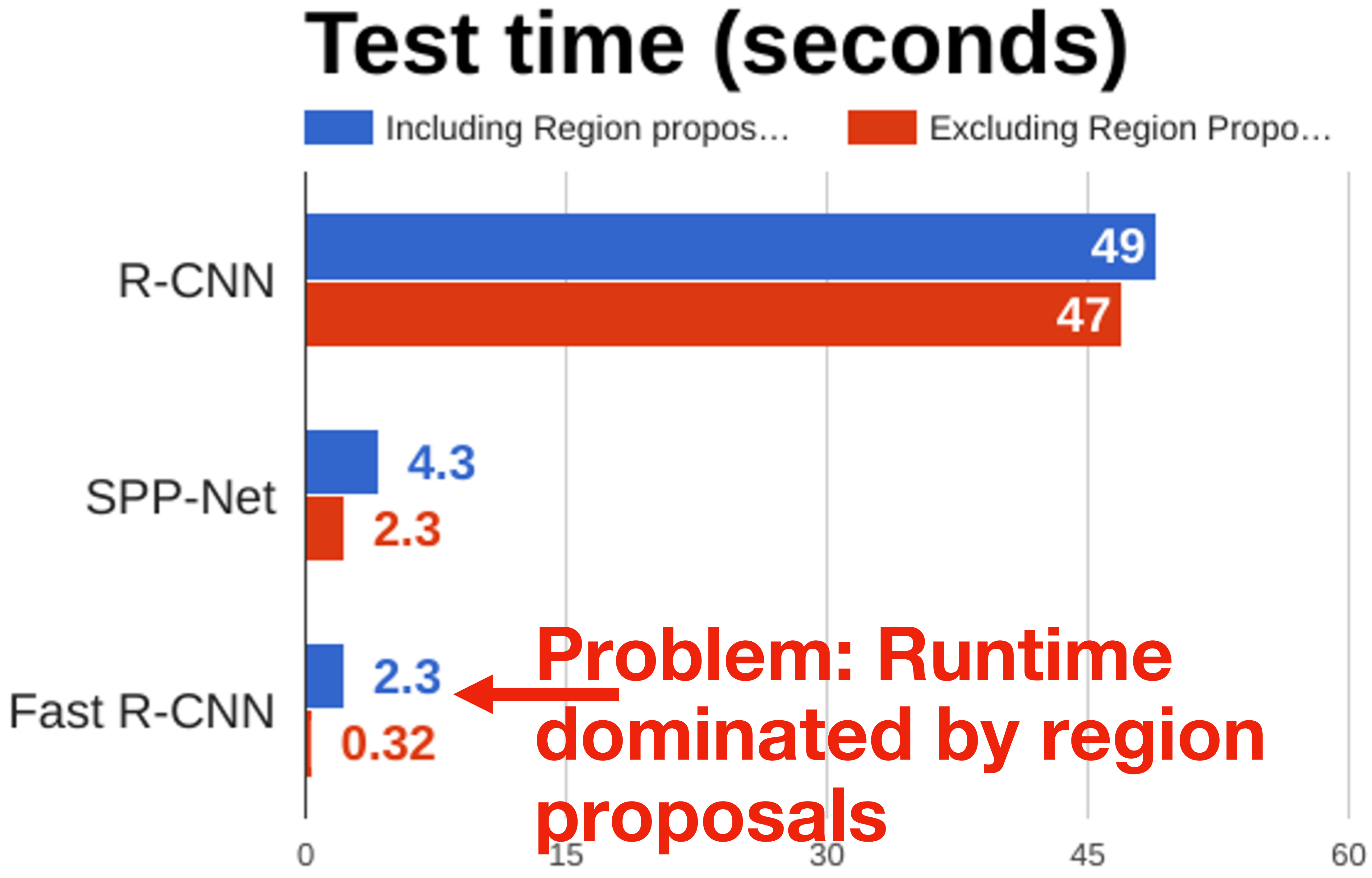




Girshick et al, "Rich feature hierarchies for accurate object detection ar

He et al, "Spatial pyramid pooling in deep convolutional networks for vi

# Fast R-CNN vs “Slow” R-CNN



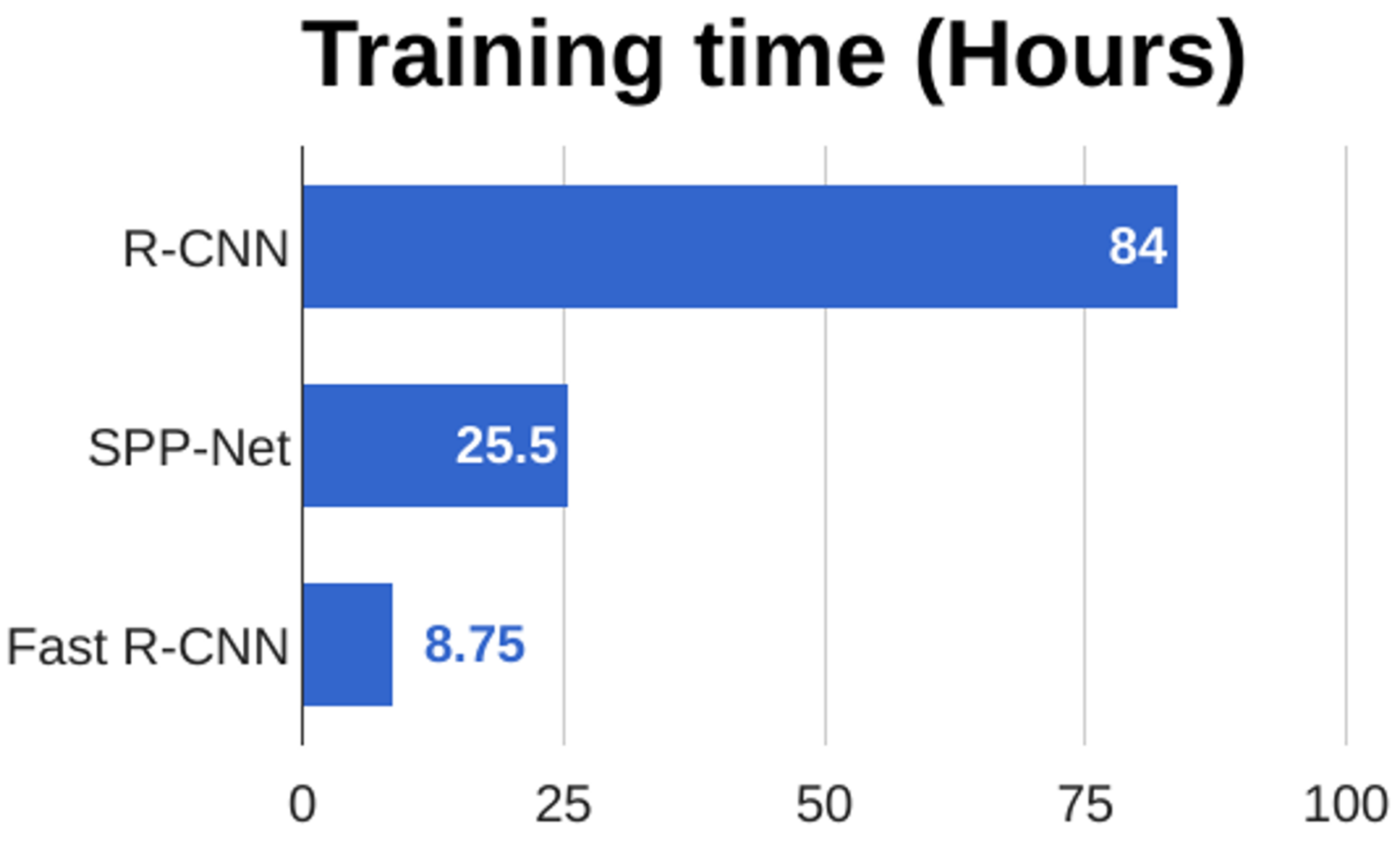



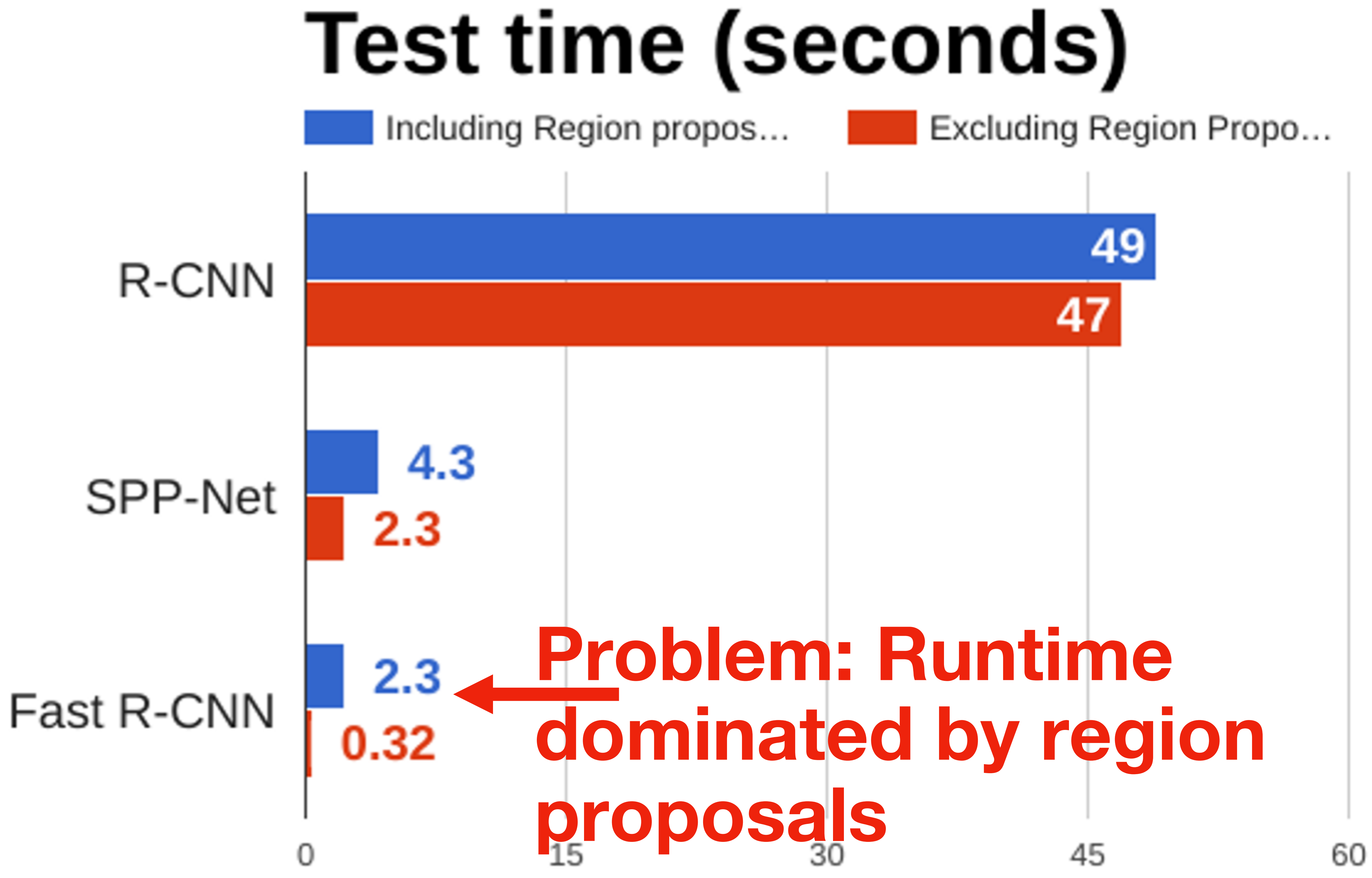

**Problem: Runtime dominated by region proposals**

Girshick et al, “Rich feature hierarchies for accurate object detection ar

He et al, “Spatial pyramid pooling in deep convolutional networks for vi

# 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**

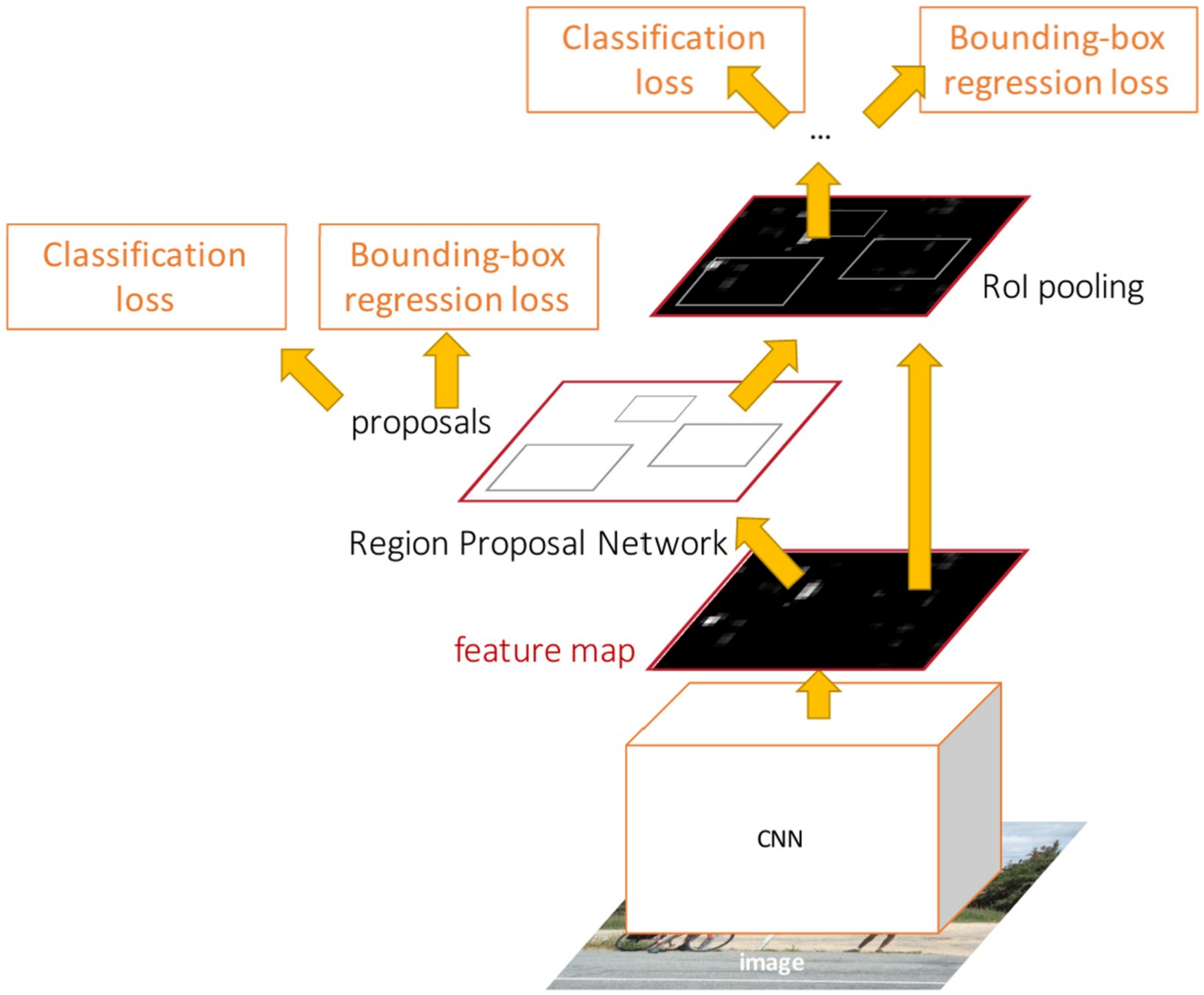
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

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

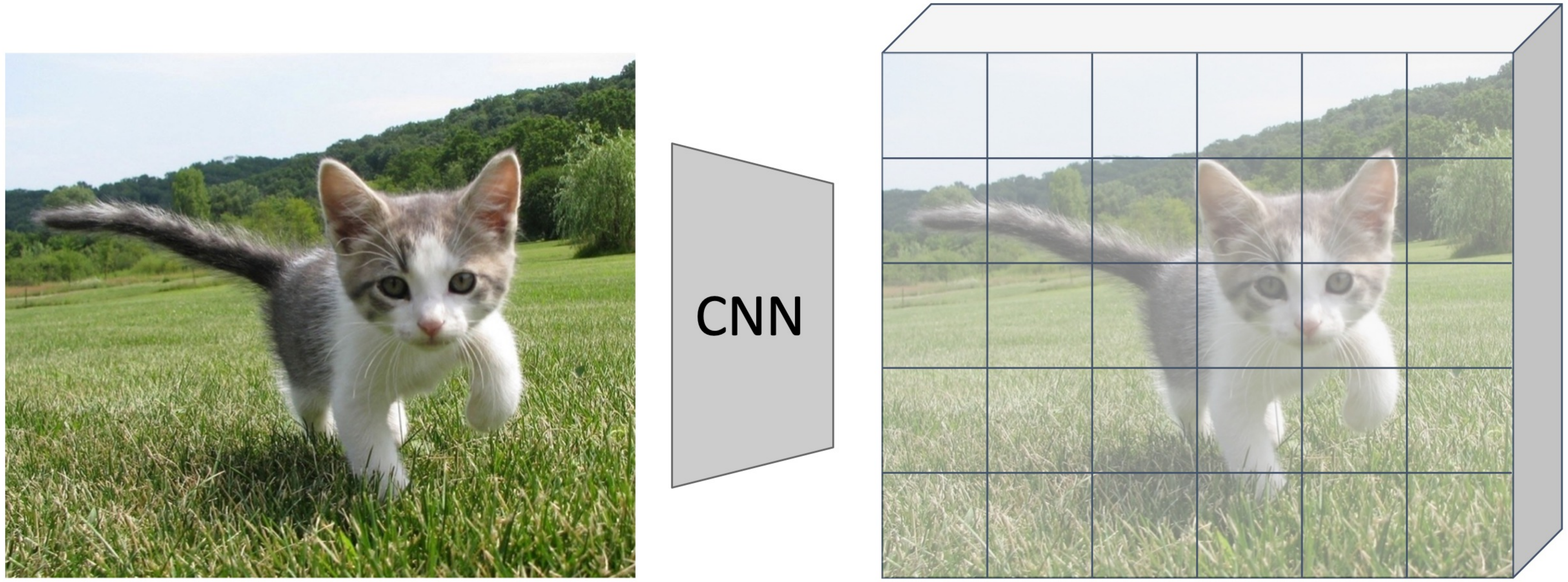
Insert **Region Proposal Network (RPN)** to predict proposals from features

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



# Region Proposal Network (RPN)

Run backbone CNN to get features aligned to input image



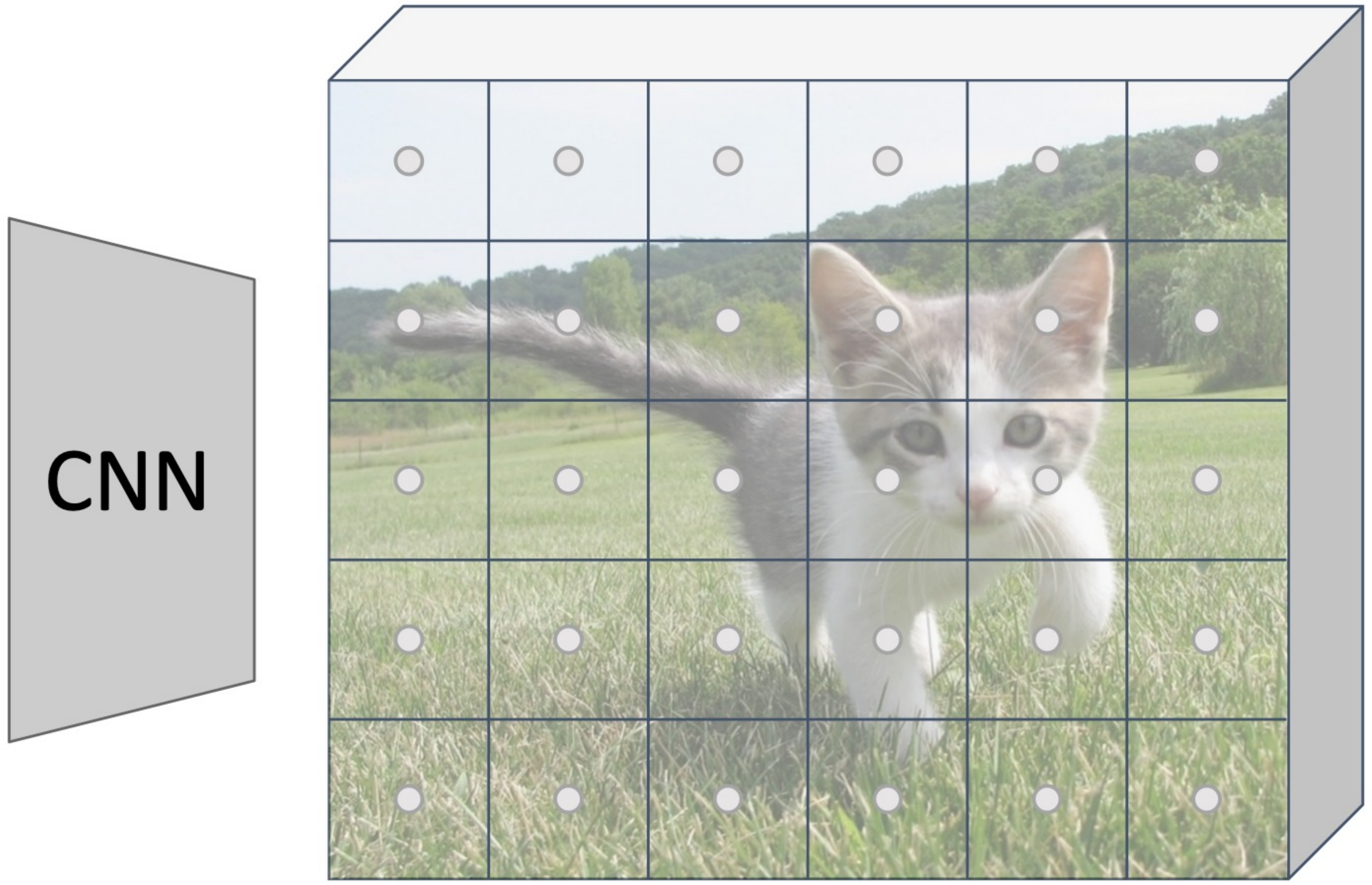

Input Image
(e.g. 3 x 640 x 480)

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

# Region Proposal Network (RPN)

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input




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

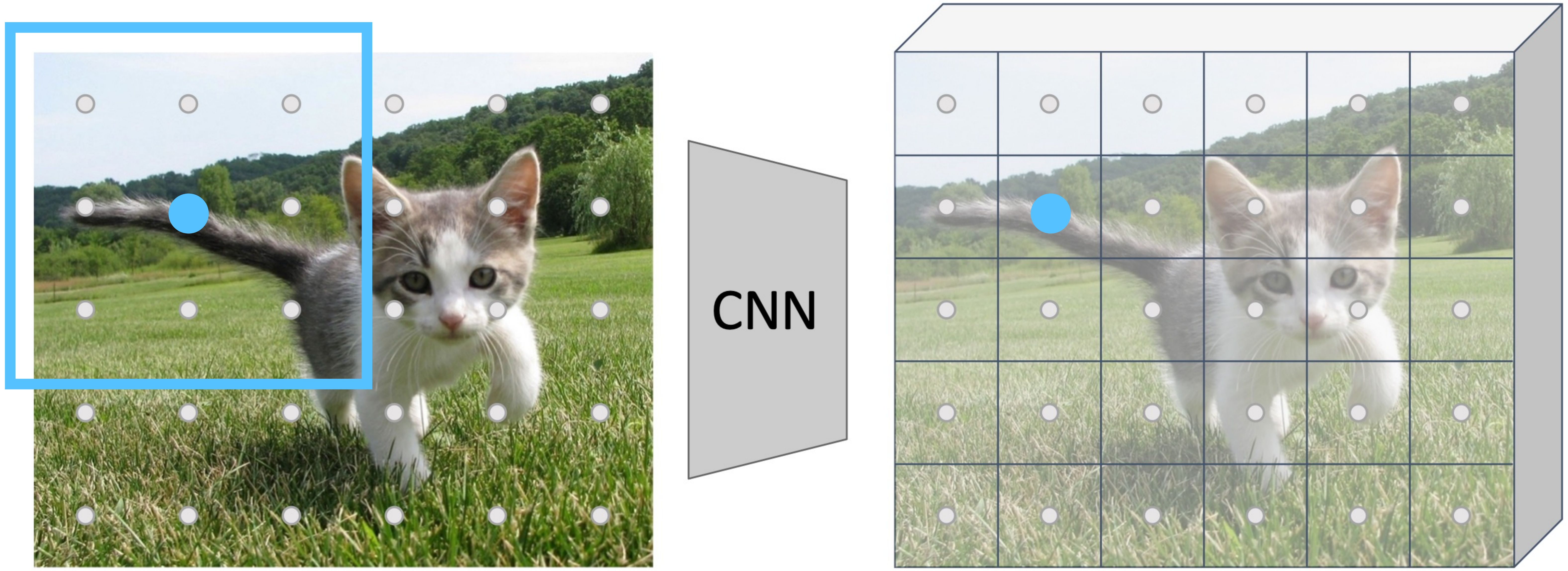
Image features
(e.g. 512 x 5 x 6)

# Region Proposal Network (RPN)

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

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input




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

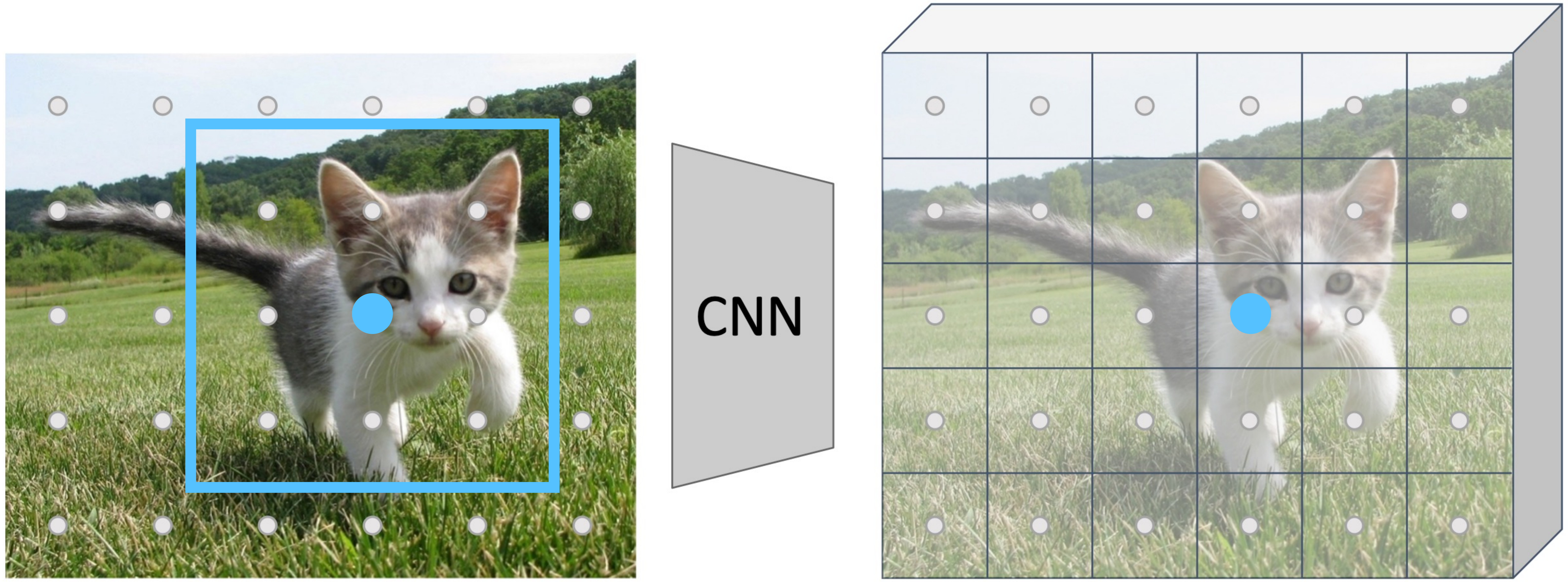
Image features
(e.g. 512 x 5 x 6)

# Region Proposal Network (RPN)

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

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input




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

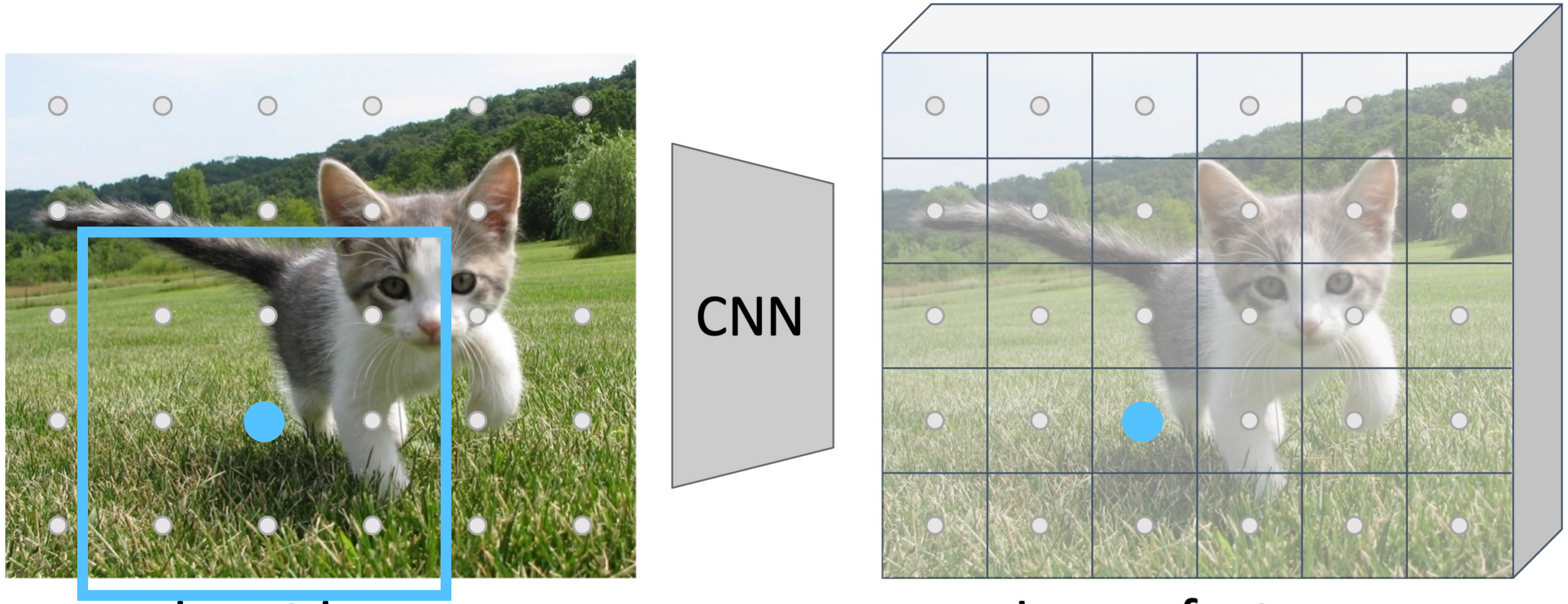
Image features
(e.g. 512 x 5 x 6)

# Region Proposal Network (RPN)

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

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input




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

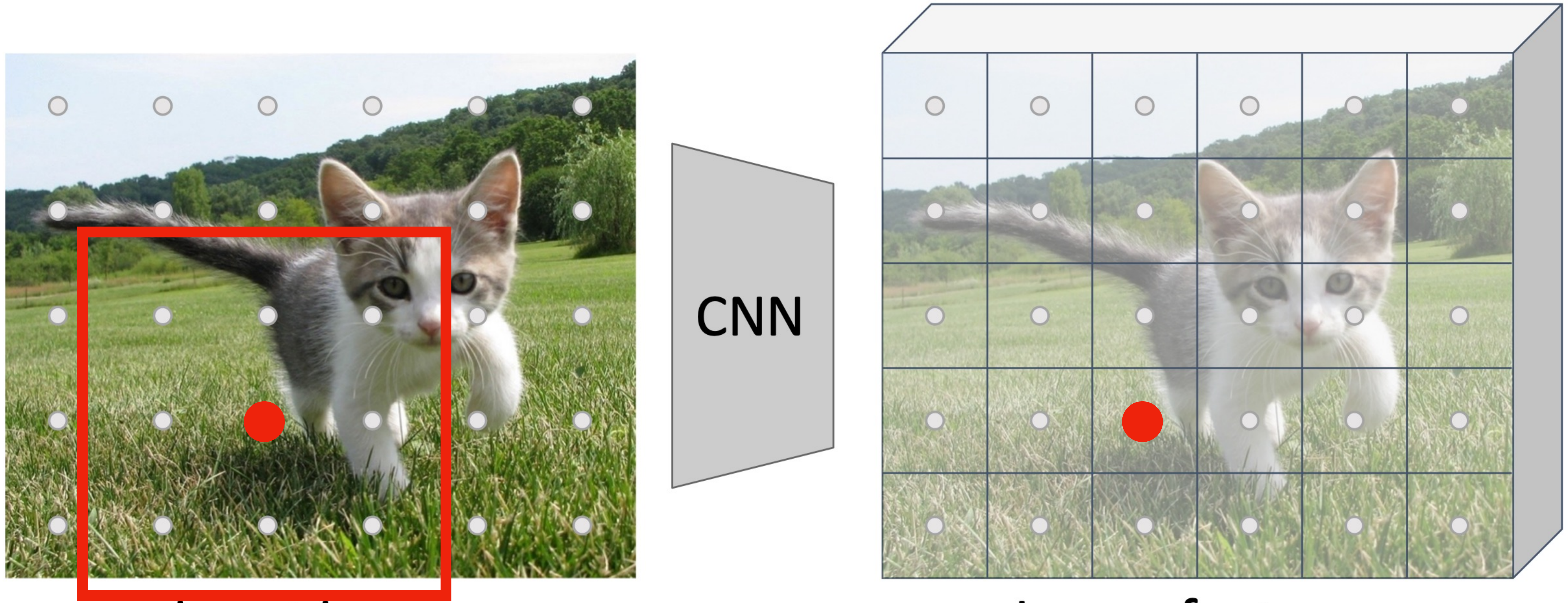
Image features
(e.g. 512 x 5 x 6)

# Region Proposal Network (RPN)

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

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input




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

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

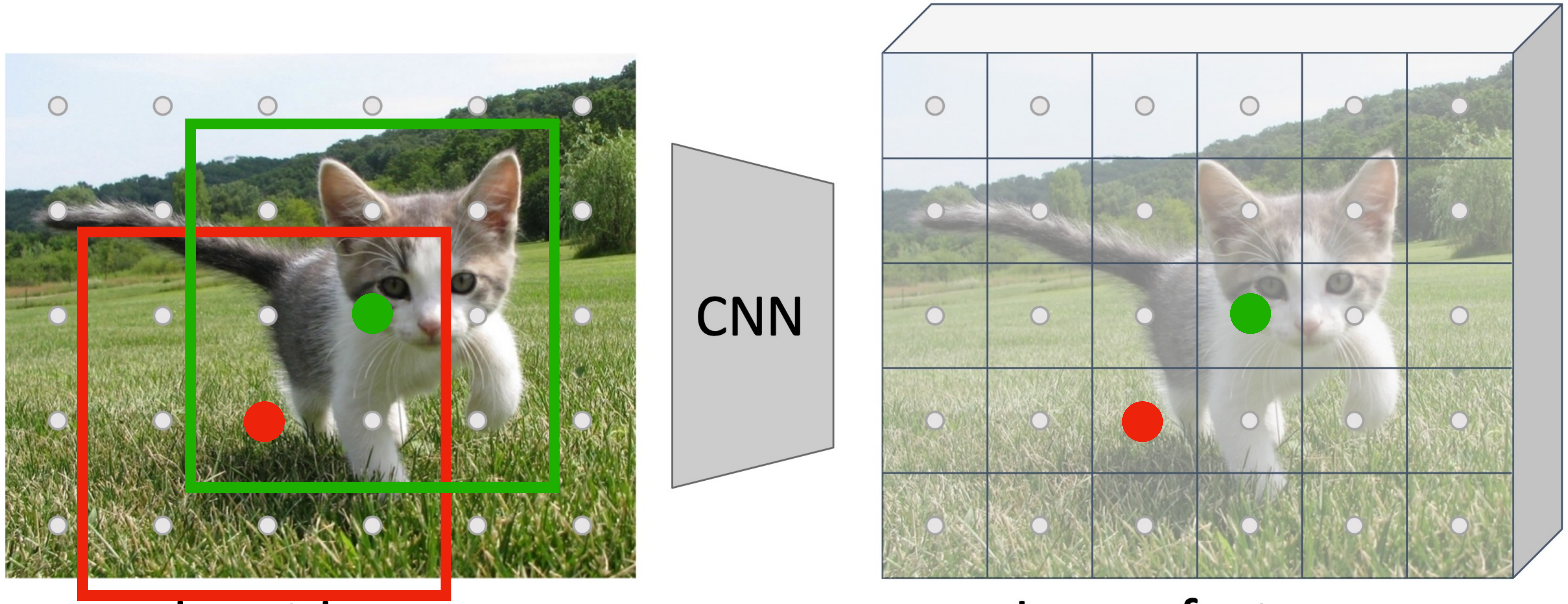
Classify each anchor as positive (object) or negative (no object)

# Region Proposal Network (RPN)

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

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input




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

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

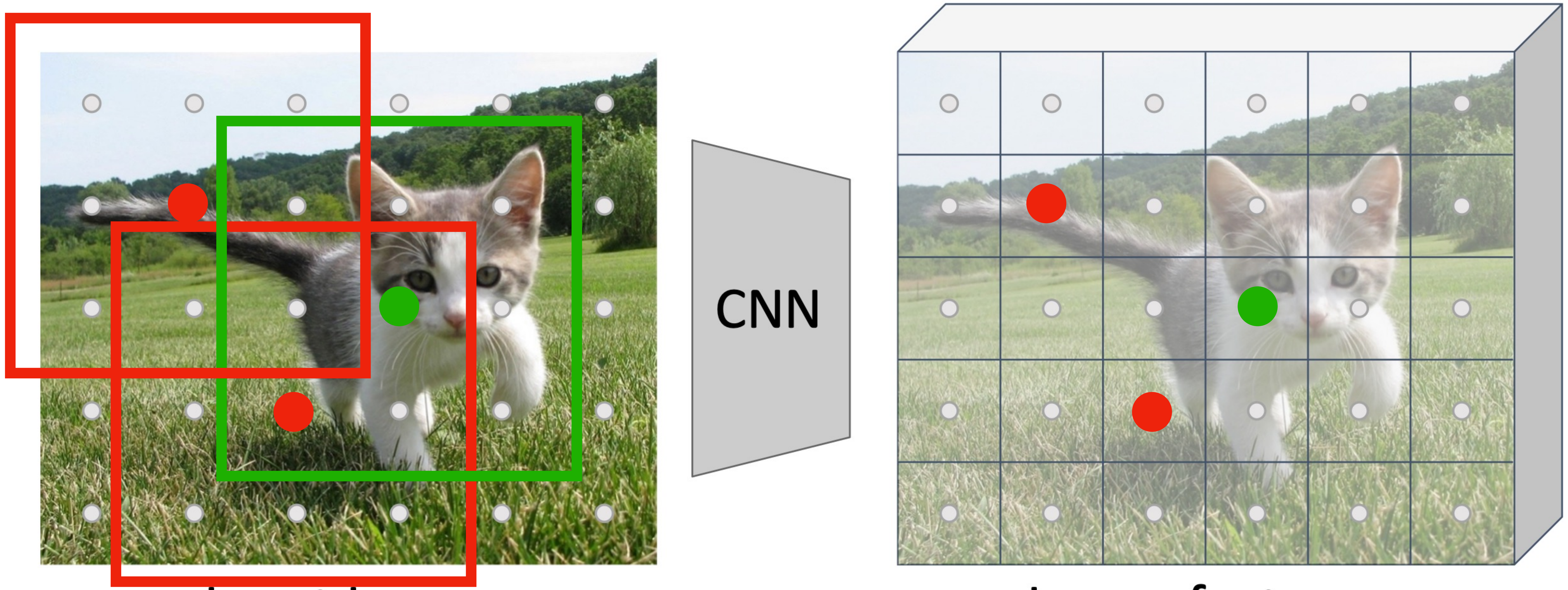
Classify each anchor as positive (object) or negative (no object)

# Region Proposal Network (RPN)

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

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input




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

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

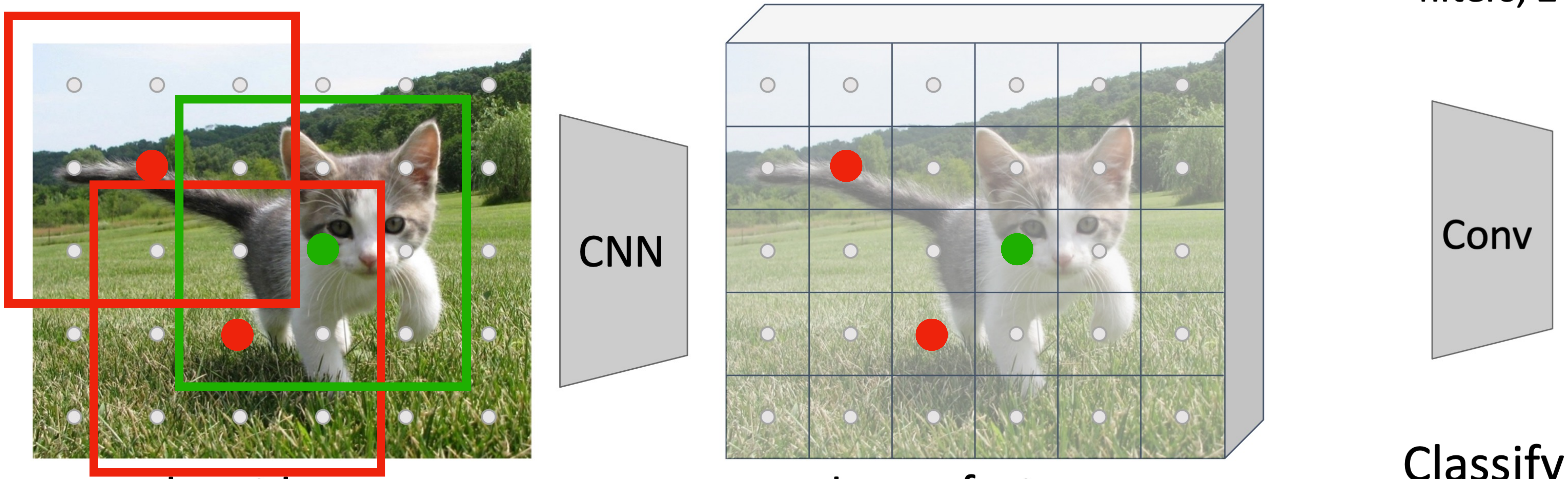
Classify each anchor as positive (object) or negative (no object)

# Region Proposal Network (RPN)

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input

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




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

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

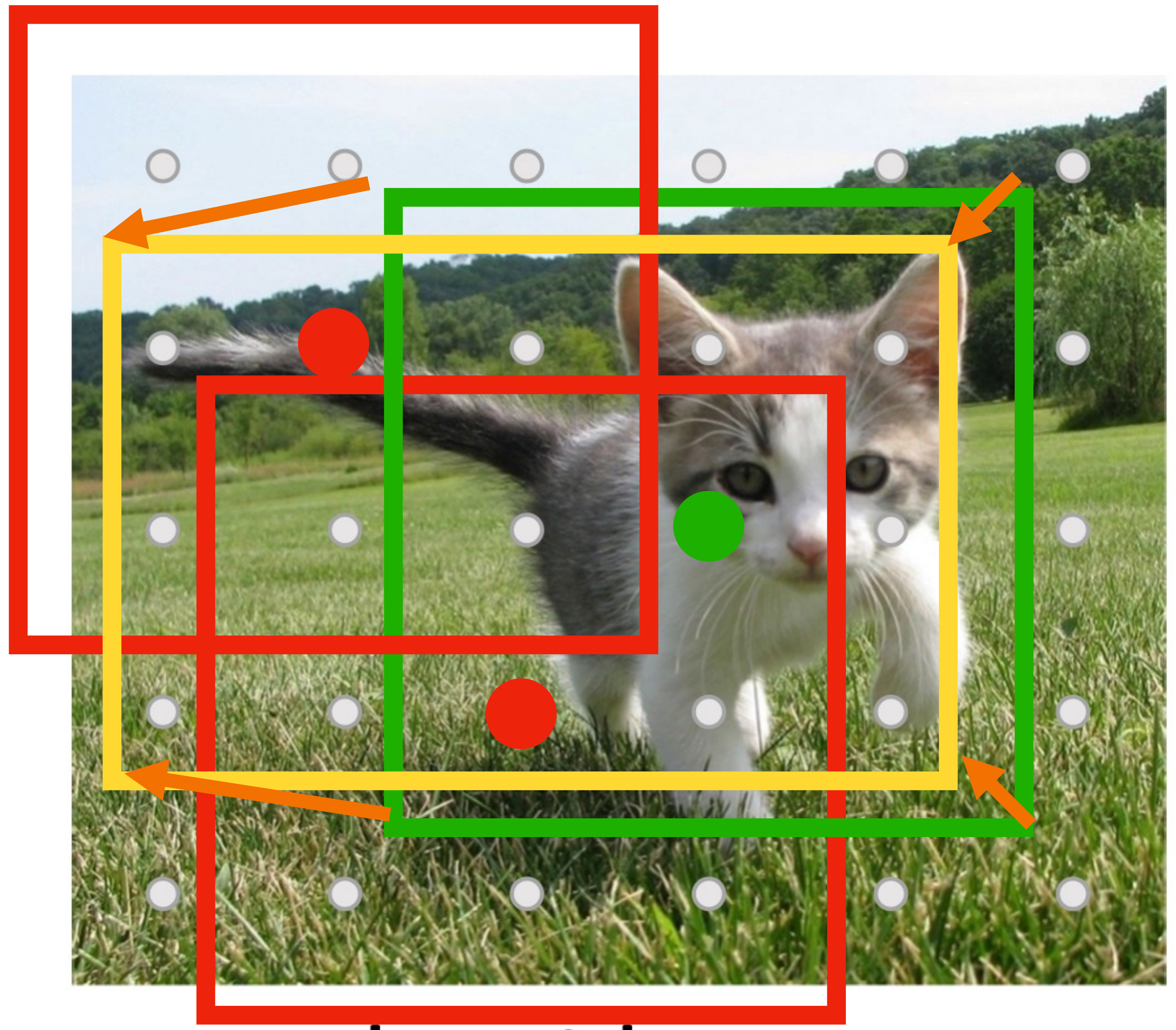
Anchor is object?
2 x 5 x 6

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

Ren et al, “Faster R-NN: Towards Real-Time Object Detection with Reg

# Region Proposal Network (RPN)

Run backbone CNN to get features aligned to input image



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


Each feature corresponds to a point in the input

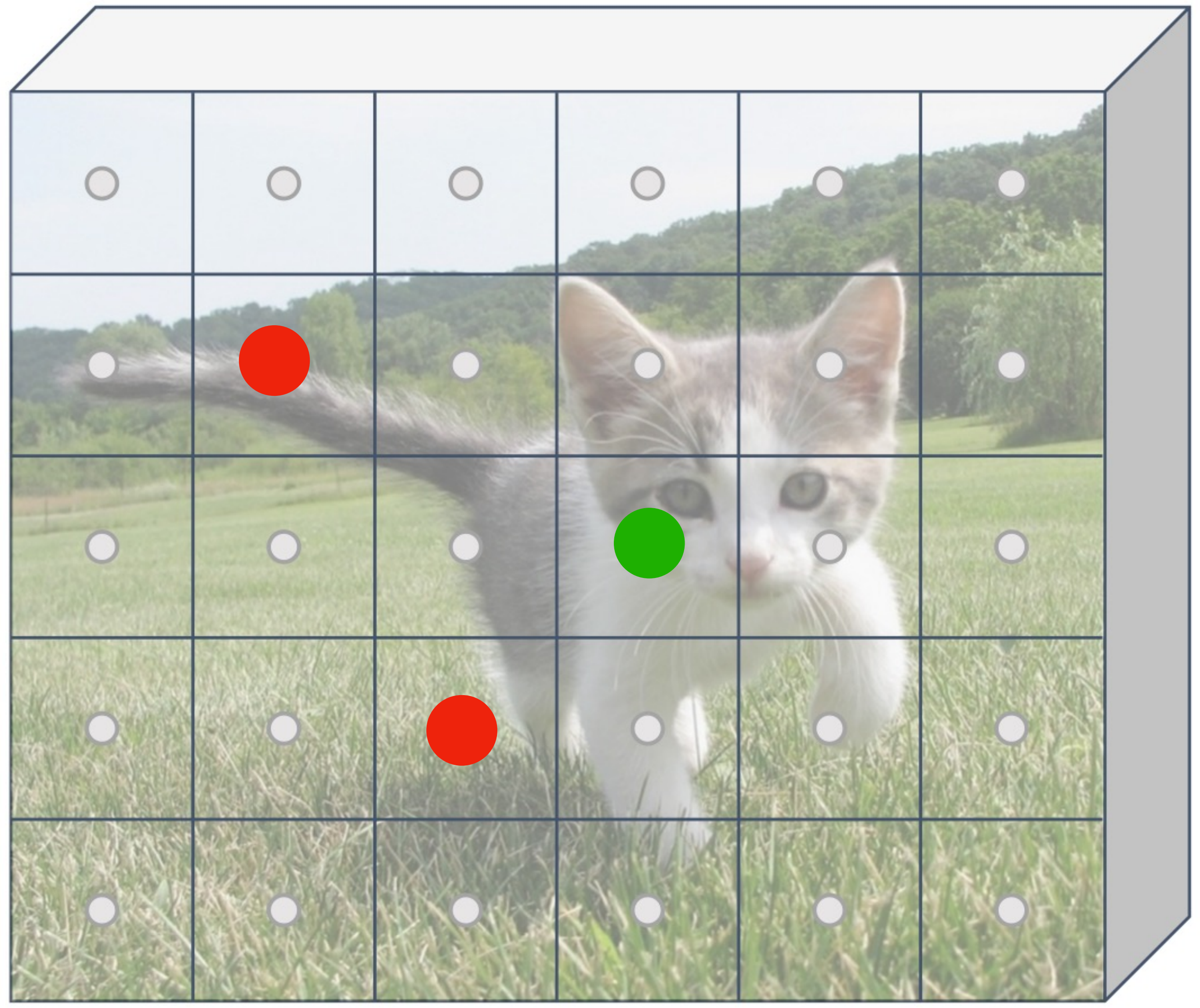
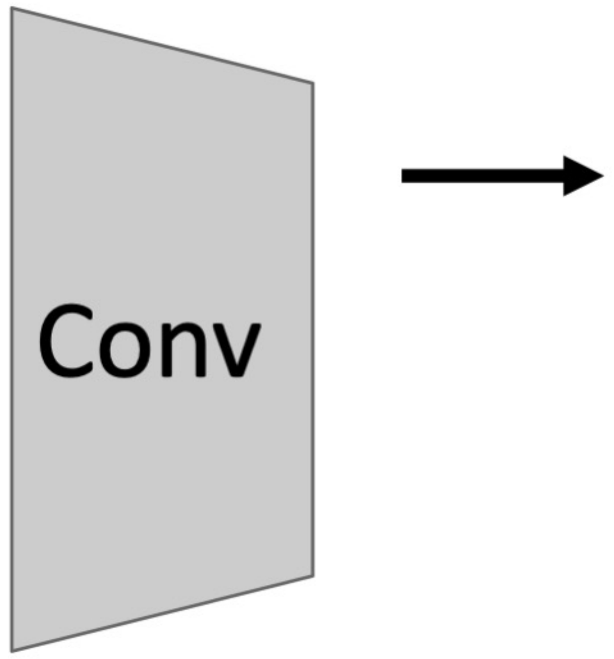


Image features
(e.g. 512 x 5 x 6)

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




Anchor is object?
2 x 5 x 6

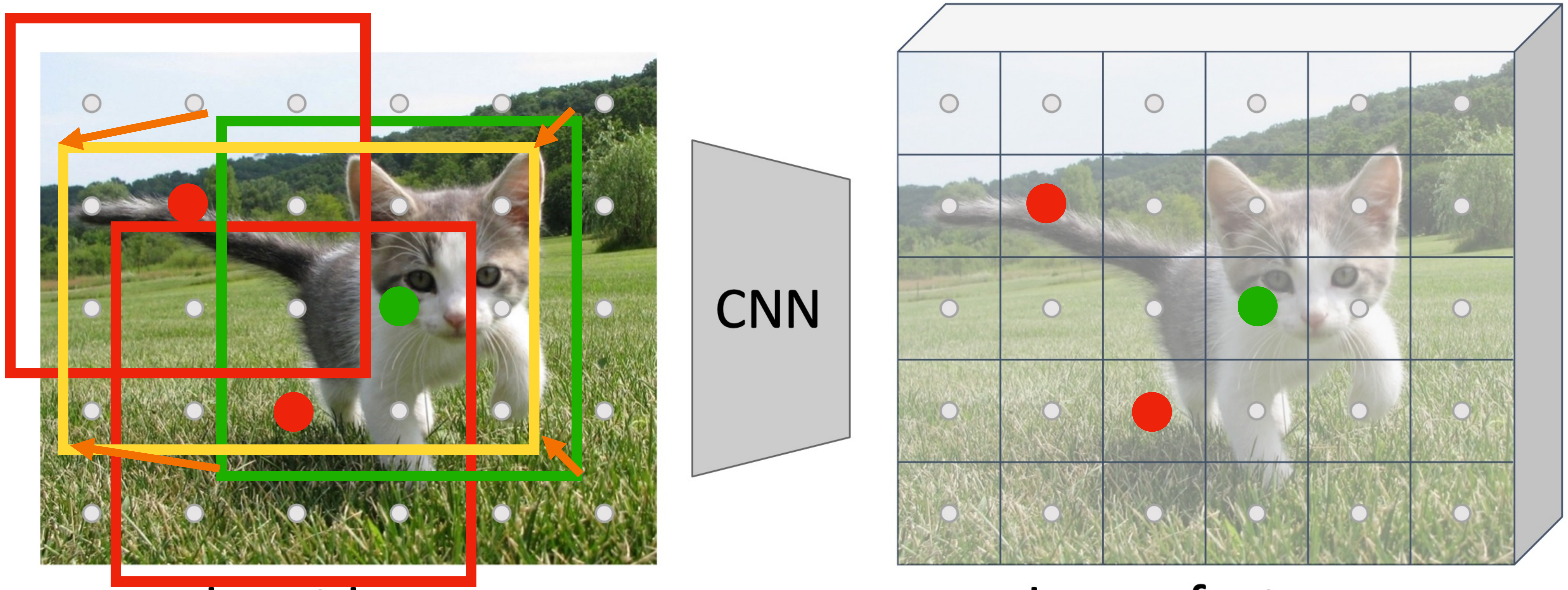
Classify each anchor as positive (object) or negative (no object)

# Region Proposal Network (RPN)

Run backbone CNN to get features aligned to input image

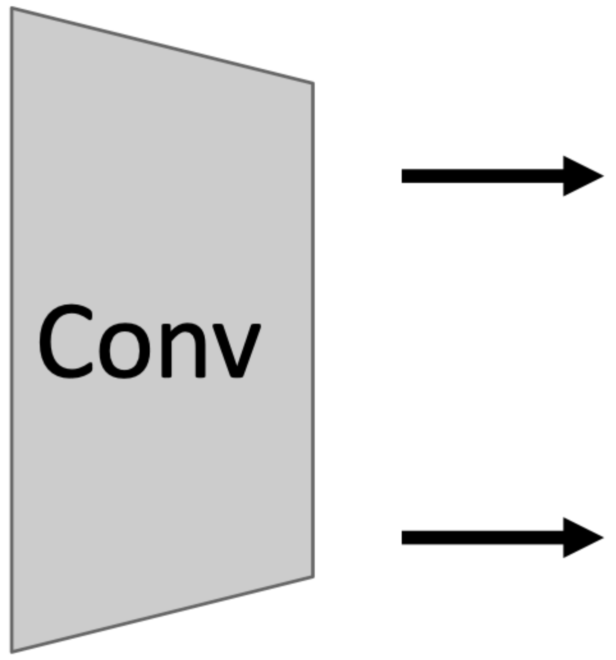
Each feature corresponds to a point in the input

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




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

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




Anchor is object?
2 x 5 x 6

Anchor transforms
4 x 5 x 6

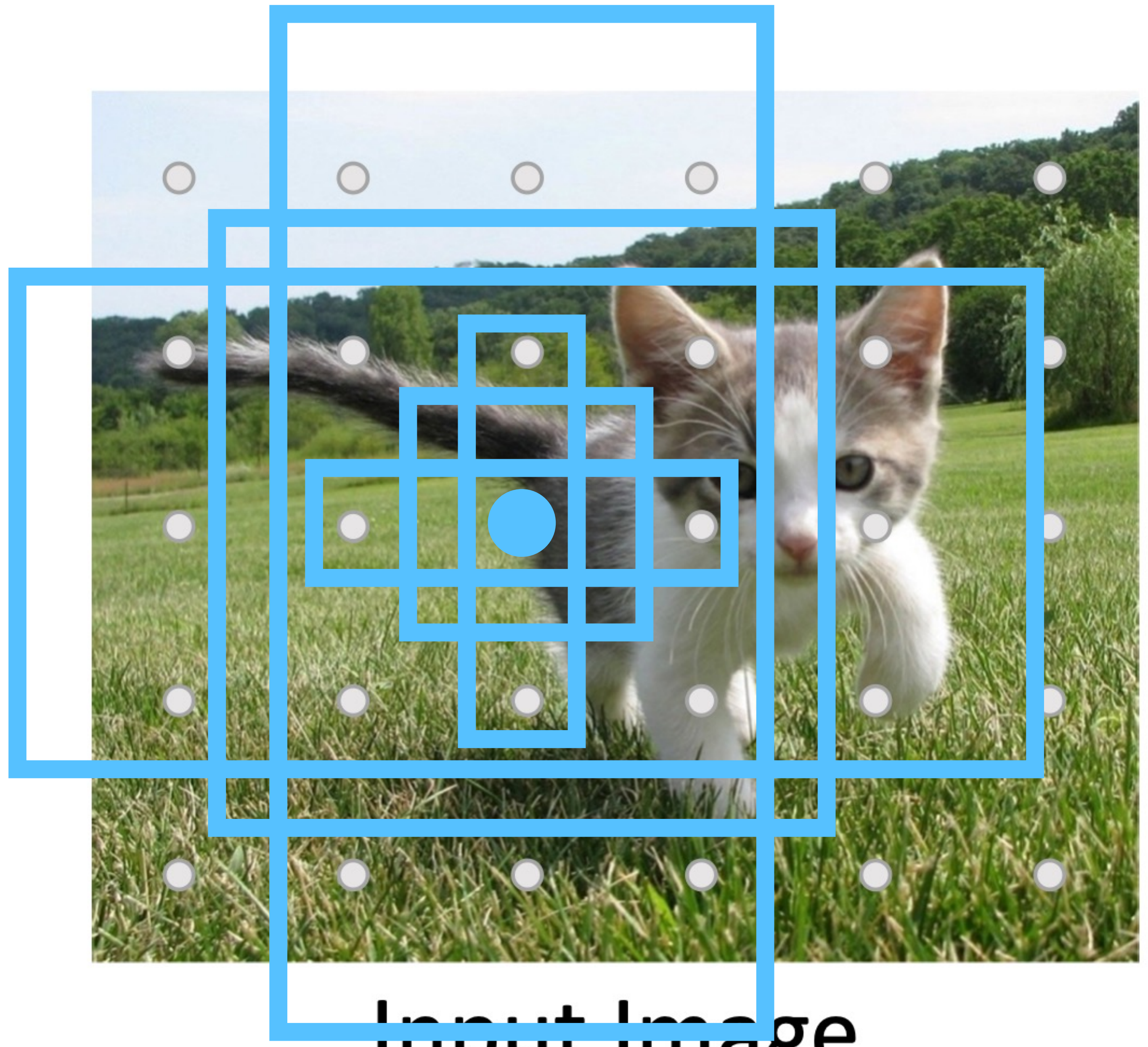
Classify each anchor as positive (object) or negative (no object)

# Region Proposal Network (RPN)

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input

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



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

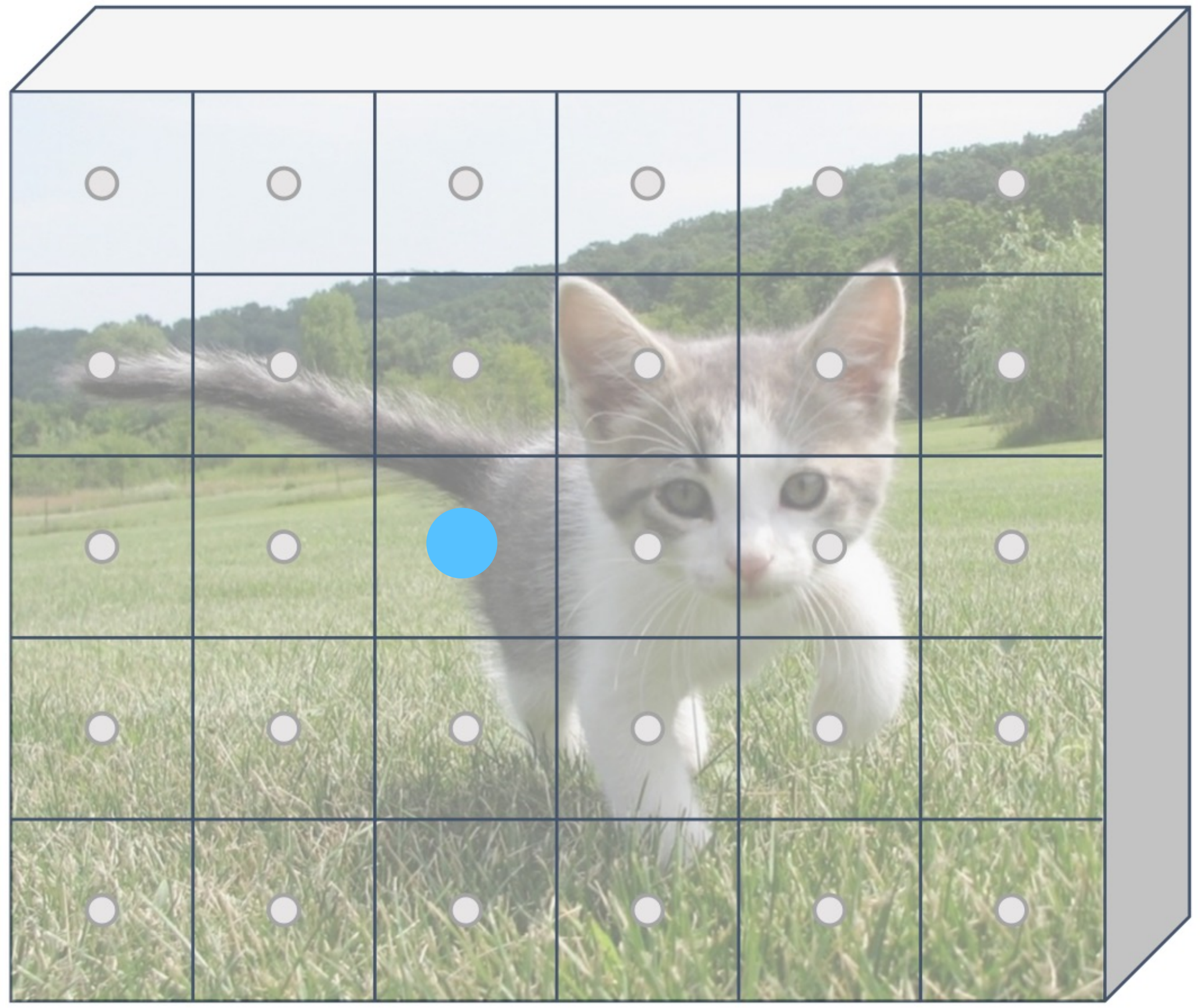
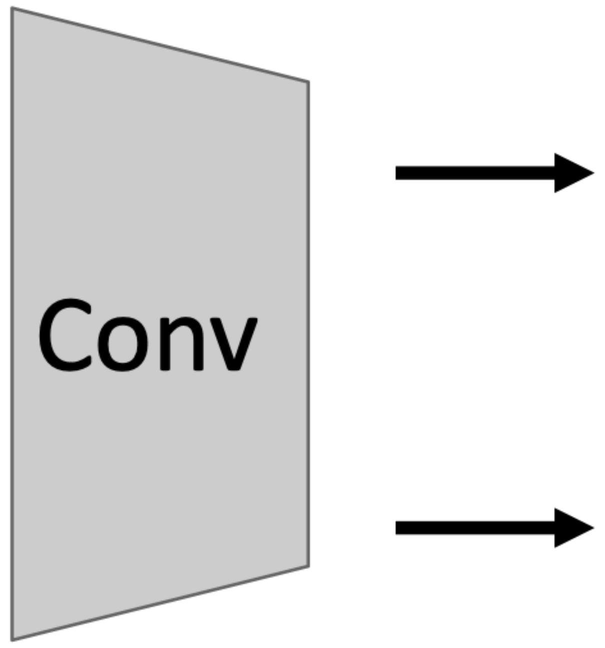



Image features
(e.g. 512 x 5 x 6)




Anchor is object?
2K x 5 x 6

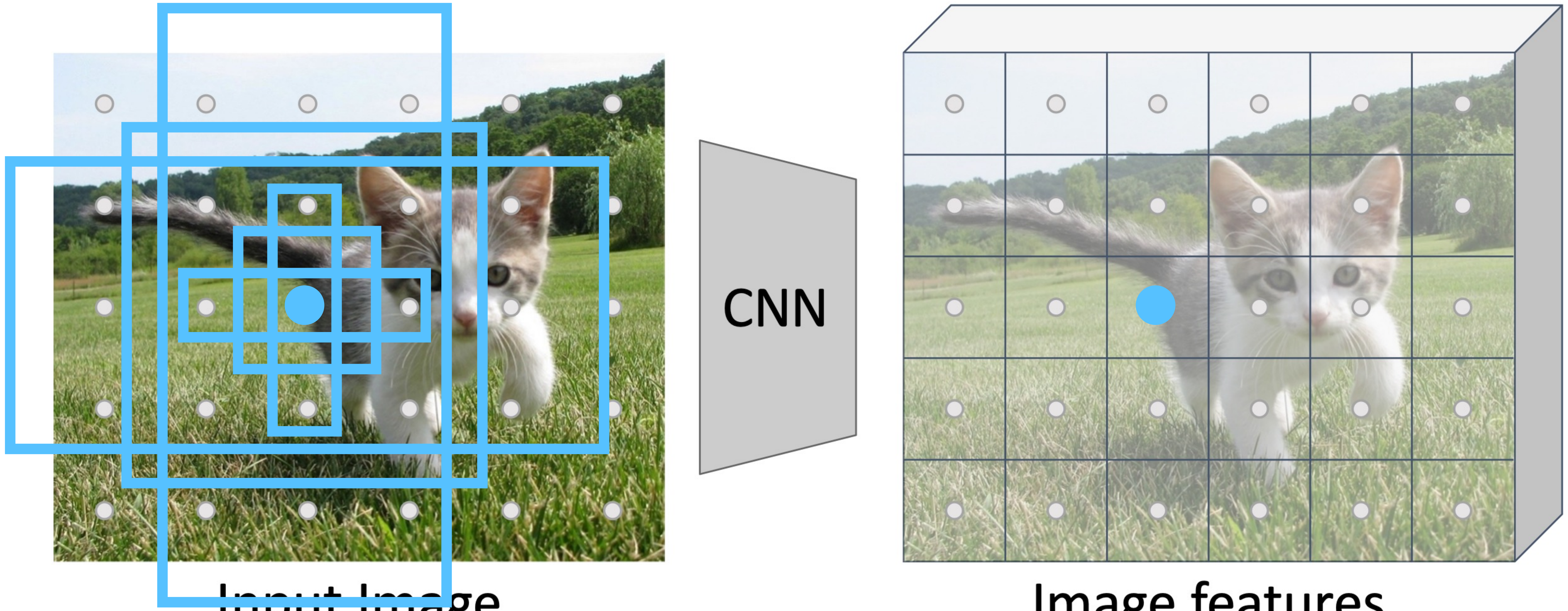
Anchor transforms
4K x 5 x 6

# Region Proposal Network (RPN)

Run backbone CNN to get features aligned to input image

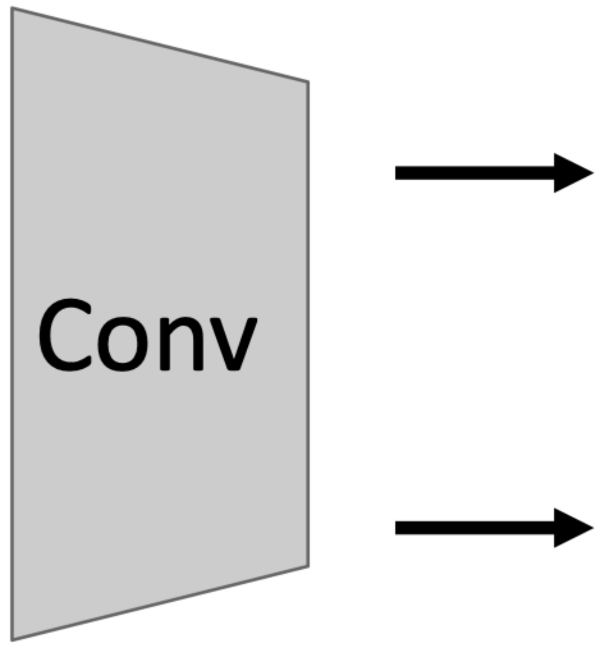
Each feature corresponds to a point in the input

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




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

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




Anchor is object?
2K x 5 x 6

Anchor transforms
4K x 5 x 6

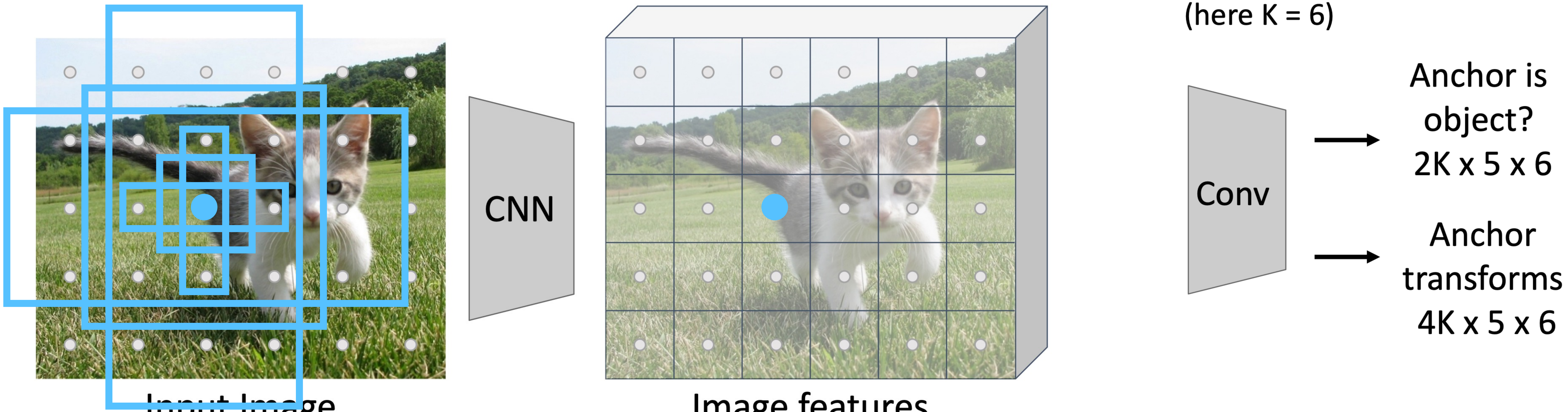
During training, supervised positive / negative anchors and box transforms like R-CNN

# Region Proposal Network (RPN)

Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input

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




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

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

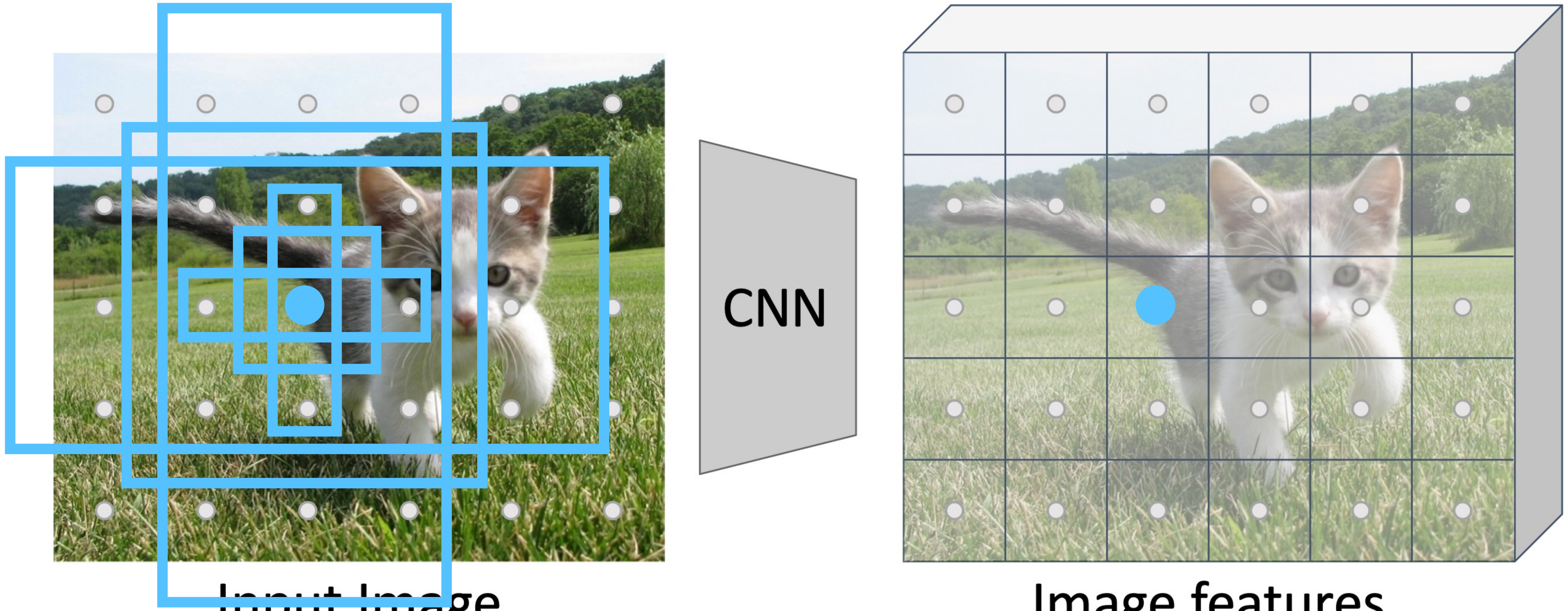
Positive anchors: >= 0.7 IoU with some GT box (plus highest IoU to each GT)

# Region Proposal Network (RPN)

Run backbone CNN to get features aligned to input image

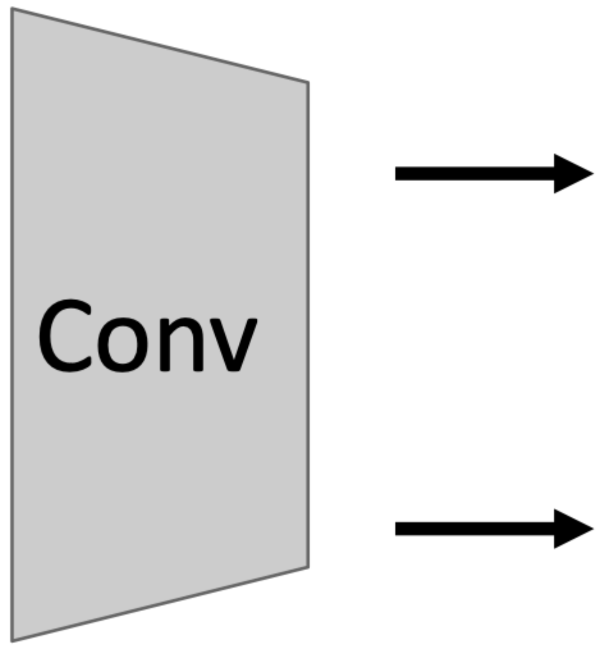
Each feature corresponds to a point in the input

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




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

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



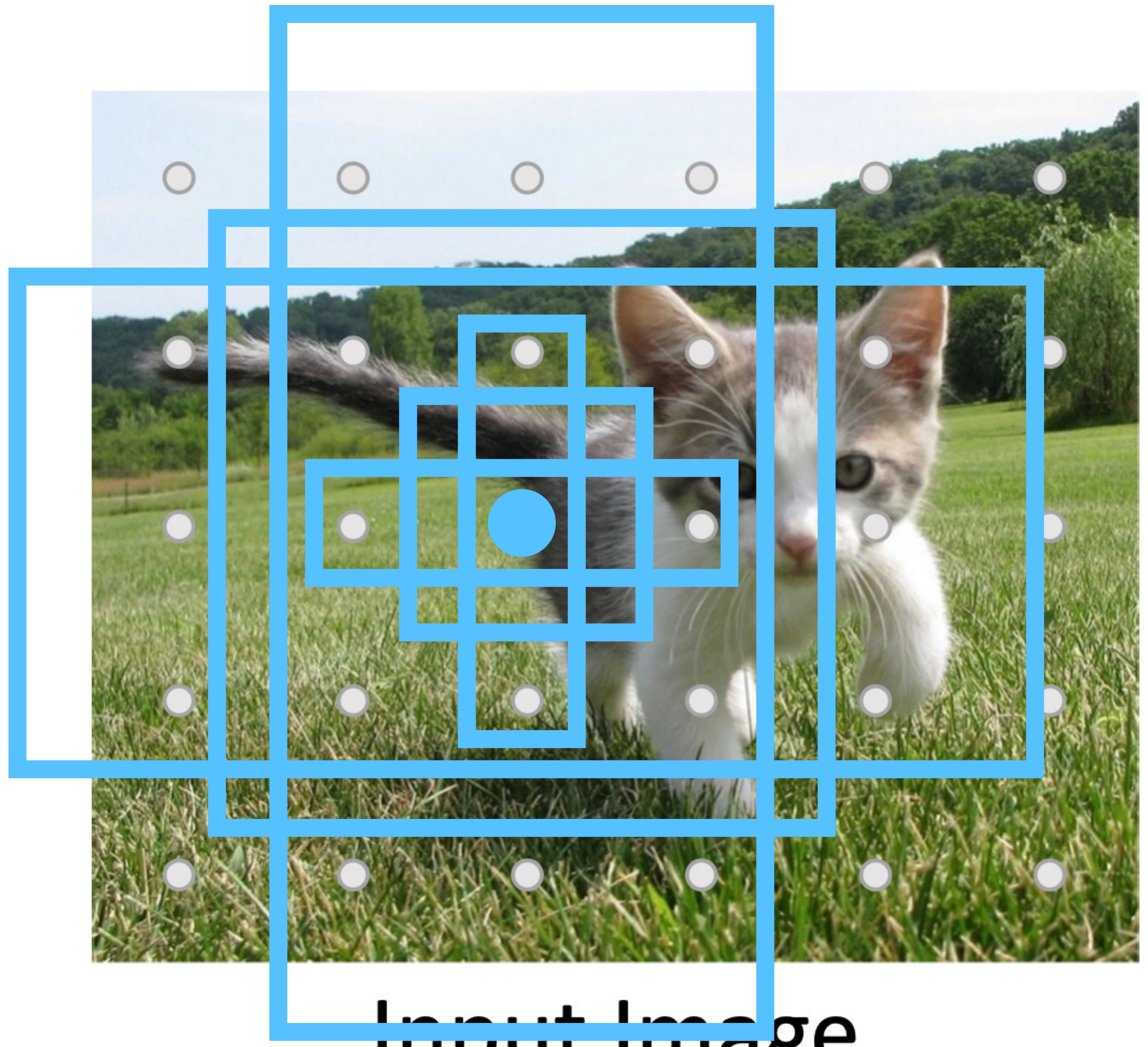

Anchor is object?
2K x 5 x 6

Anchor transforms
4K x 5 x 6

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

# Region Proposal Network (RPN)

Run backbone CNN to get features aligned to input image



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


Each feature corresponds to a point in the input

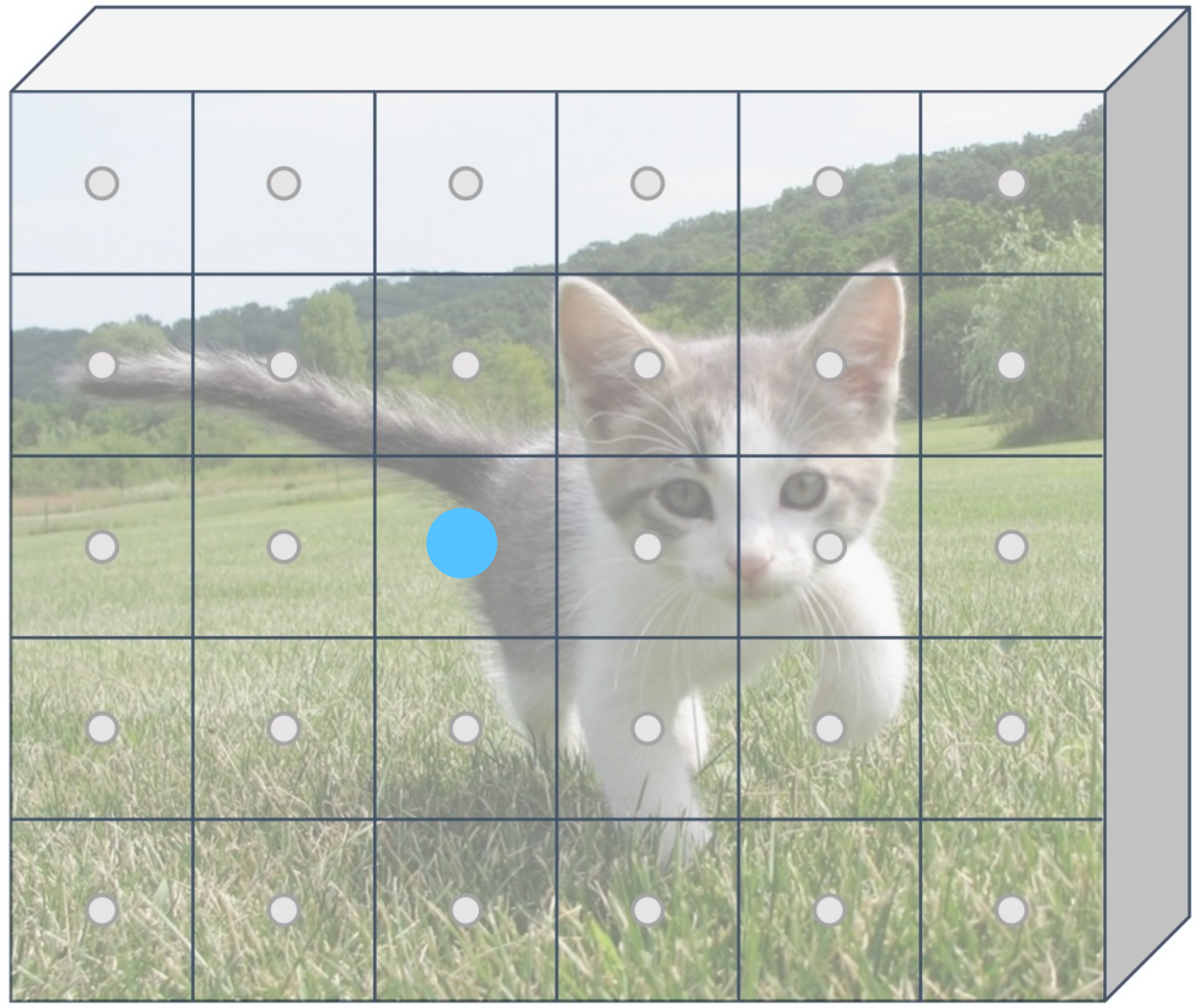
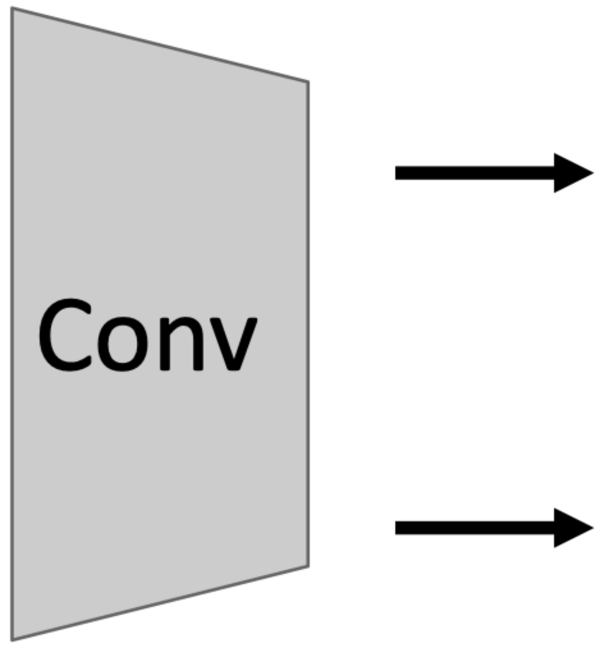


Image features
(e.g. 512 x 5 x 6)

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




Anchor is object?
2K x 5 x 6

Anchor transforms
4K x 5 x 6

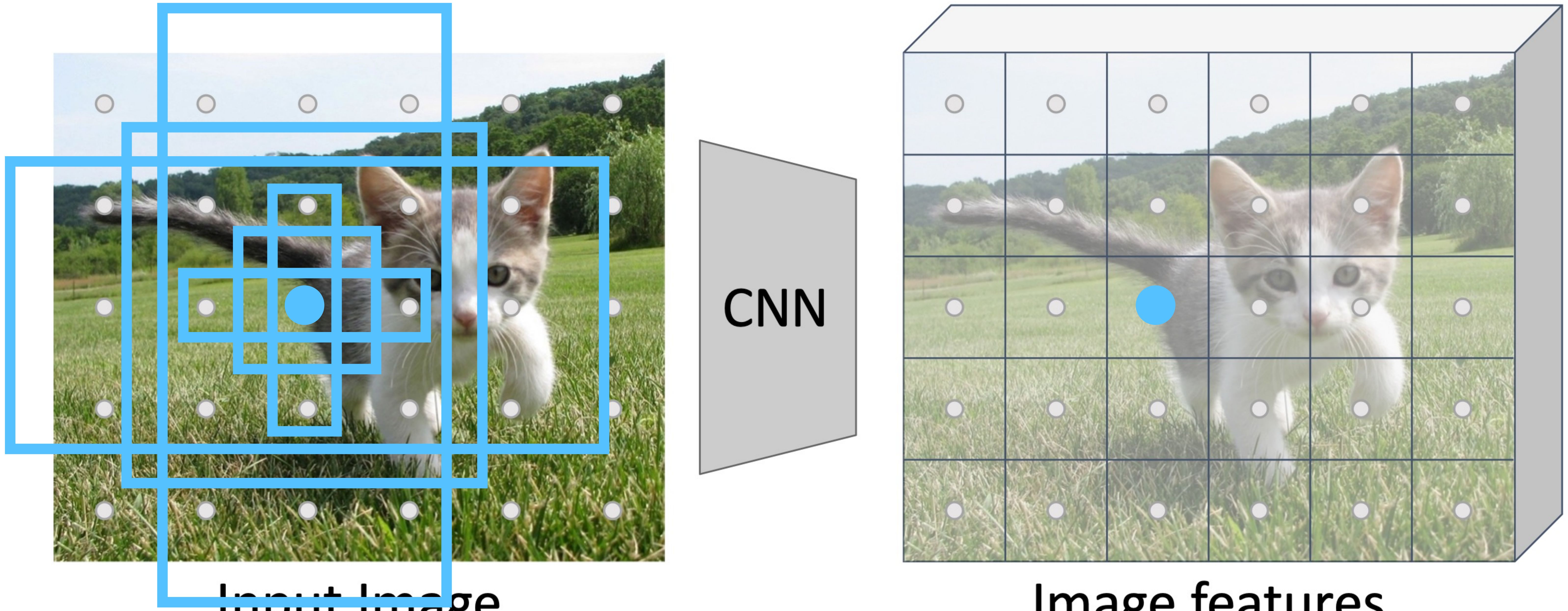
Neutral anchors: between 0.3 and 0.7 IoU with all GT boxes; ignored during training

# Region Proposal Network (RPN)

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

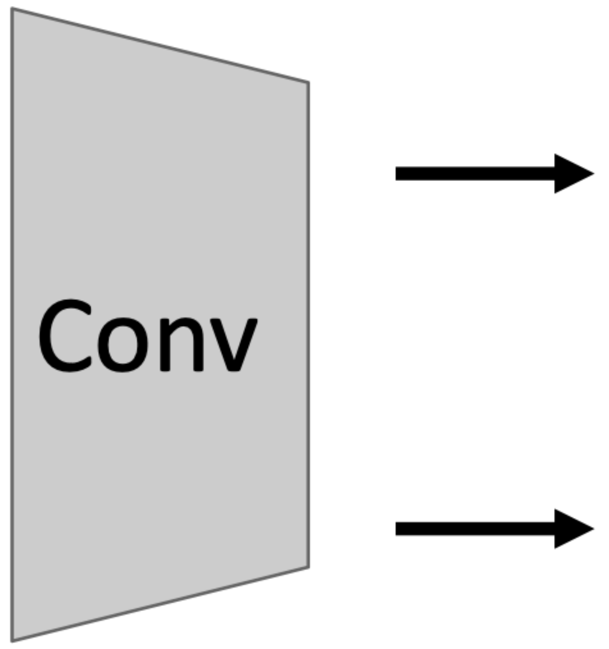
Run backbone CNN to get features aligned to input image

Each feature corresponds to a point in the input




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

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




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

# Faster R-CNN: Learnable Region Proposals

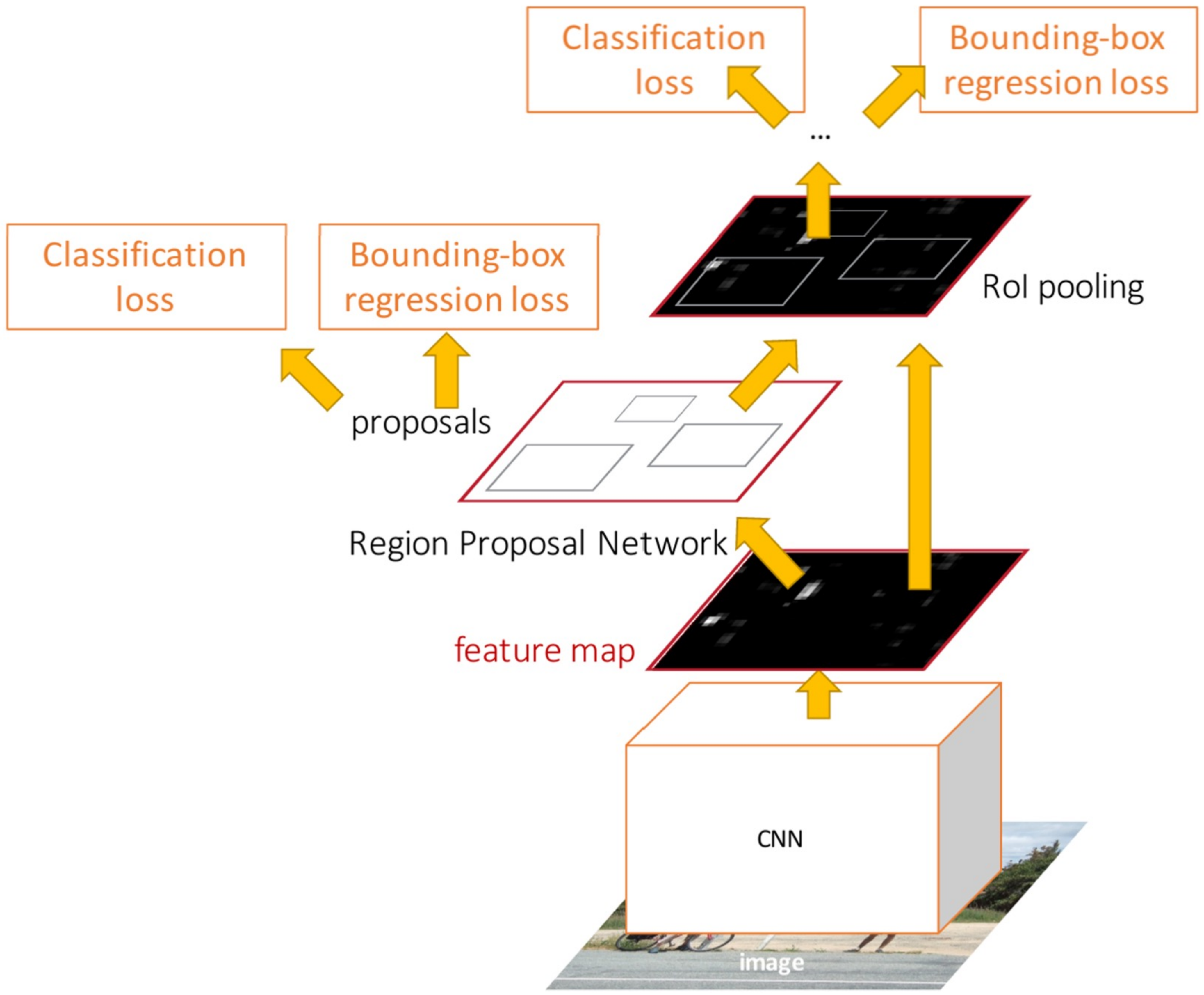
**Jointly train four losses:**

1. **RPN classification:** anchor box is object / not an object

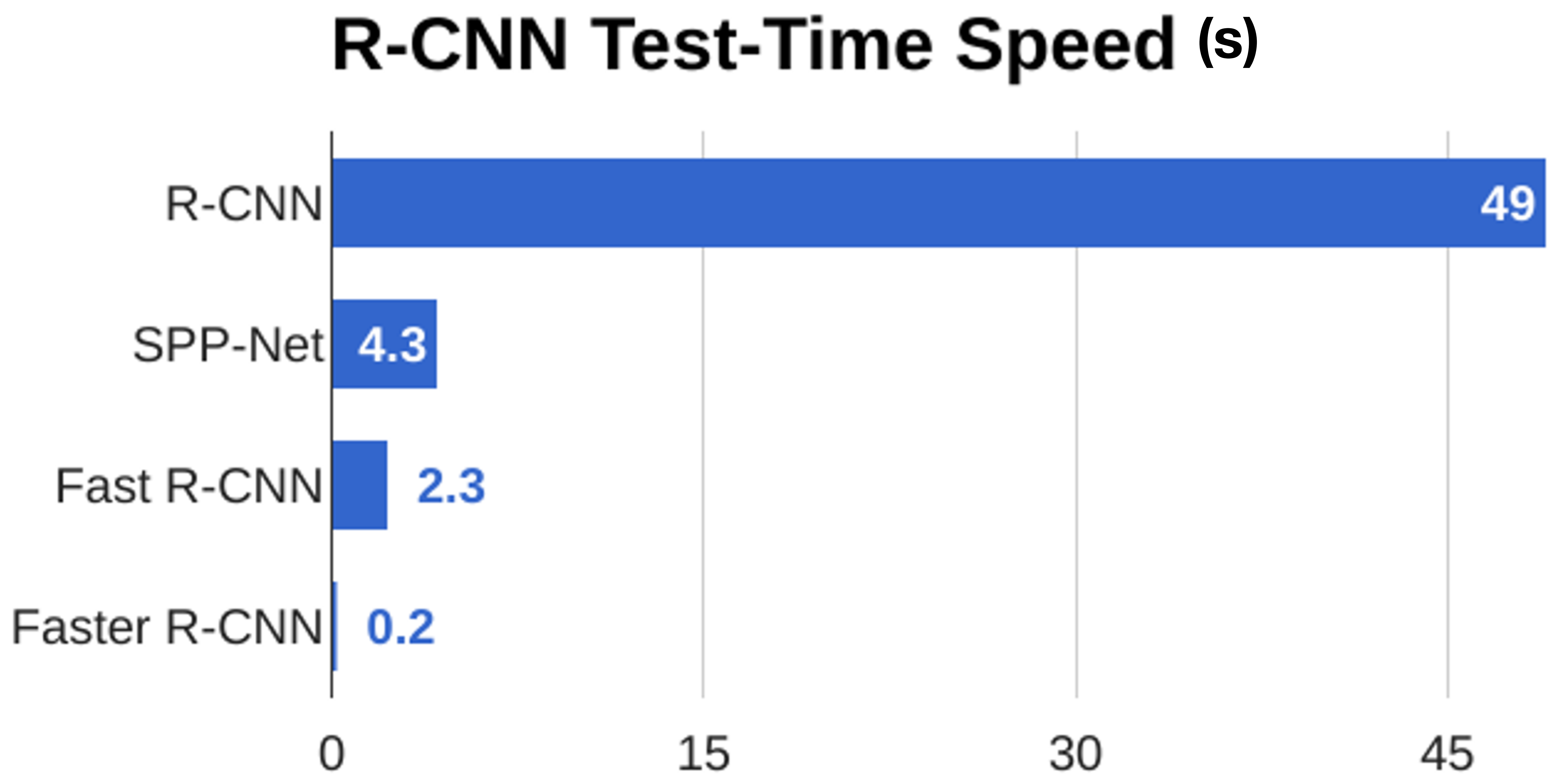
2. **RPN regression:** predict transform from anchor box to proposal box

3. **Object classification:** classify proposals as background / object class

4. **Object regression:** predict transform from proposal box to object box



# Faster R-CNN: Learnable Region Proposals



# Extend Faster R-CNN to Image Segmentation: Mask R-CNN

**Classification**




"Chocolate Pretzels"

No spatial extent

**Semantic Segmentation**



Chocolate Pretzels, Shelf

No objects, just pixels

**Object Detection**




**Instance Segmentation**


Flipz, Hershey's, Keese's

Multiple objects

# Extend Faster R-CNN to Instance Segmentation: Mask R-CNN

**Instance Segmentation**

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

**Approach**

Perform object detection then predict a segmentation mask for each object detected!

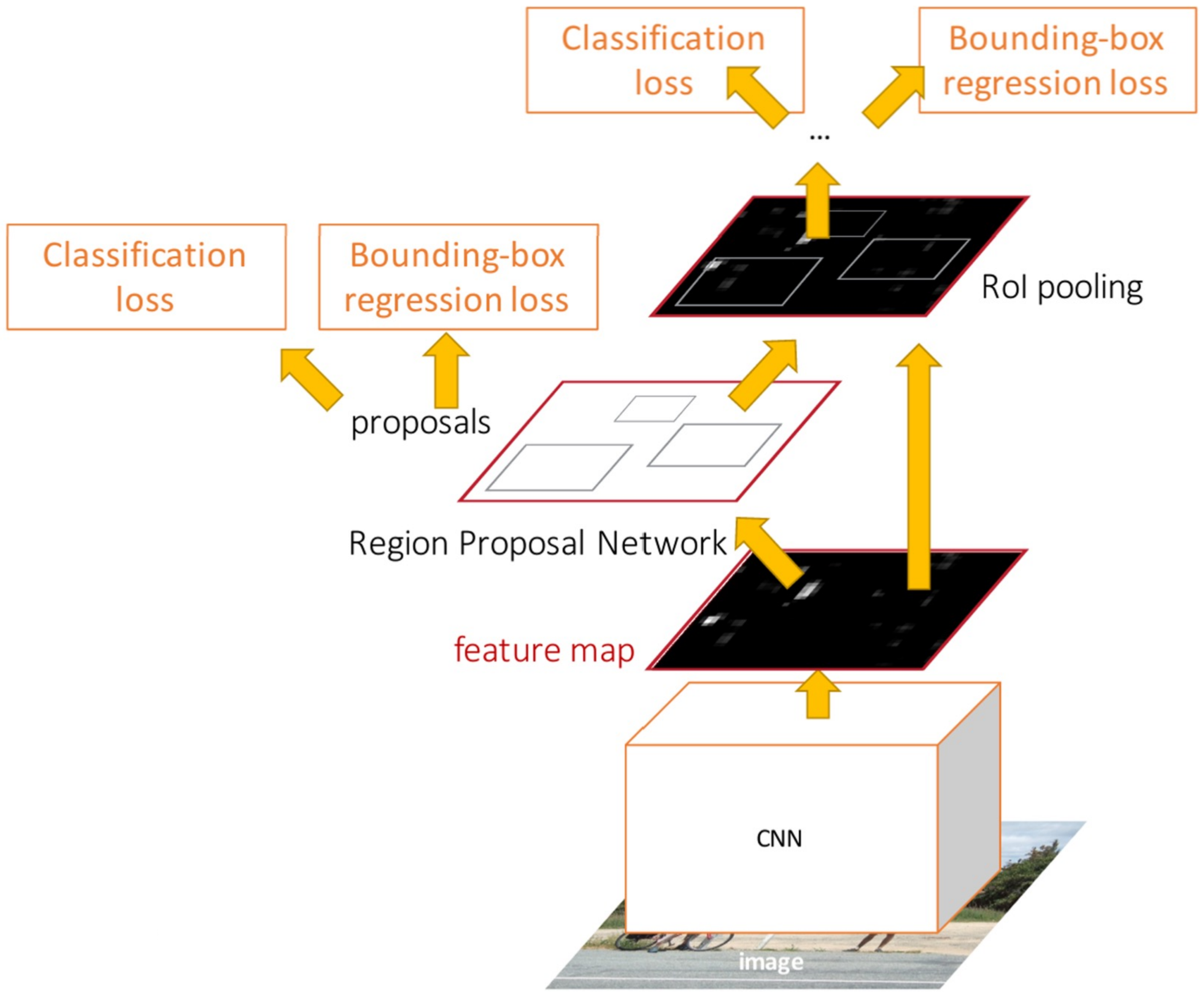
# Extend Faster R-CNN into Mask R-CNN

**Faster R-CNN**

1. **Feature Extraction** at the image-level
2. **Regions of Interest** proposal from feature map
3. **In Parallel**

    1. **Object classification:** classify proposals

    2. **Object regression:** predict transforr from proposal box to object box




Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Re

Figure copyright 2015, Ross Girshick

# Extend Faster R-CNN into Mask R-CNN

**Faster R-CNN**

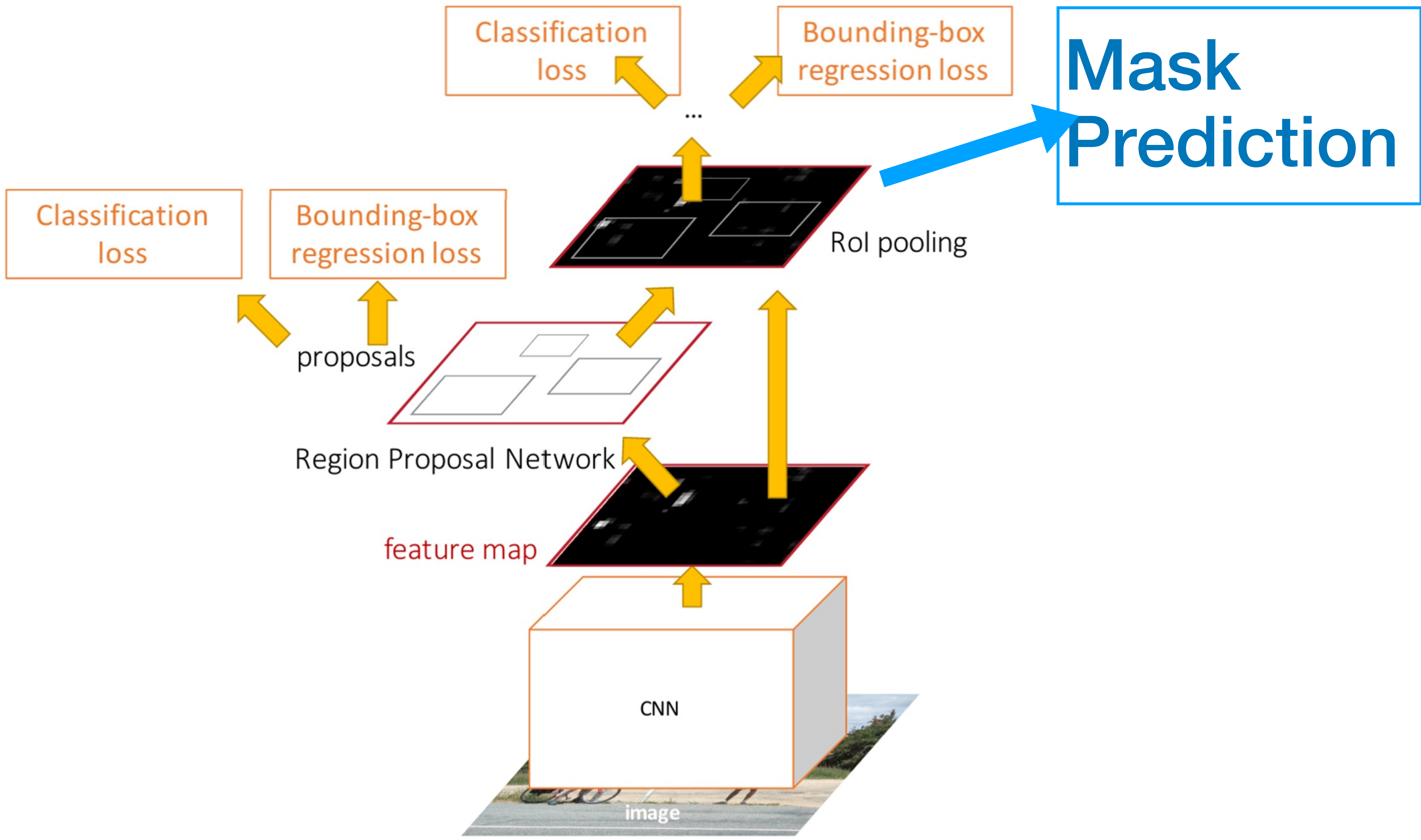
**Mask R-CNN**

1. **Feature Extraction** at the image-level
2. **Regions of Interest** proposal from feature map
3. **In Parallel**

   a. **Object Classification:** classify proposals

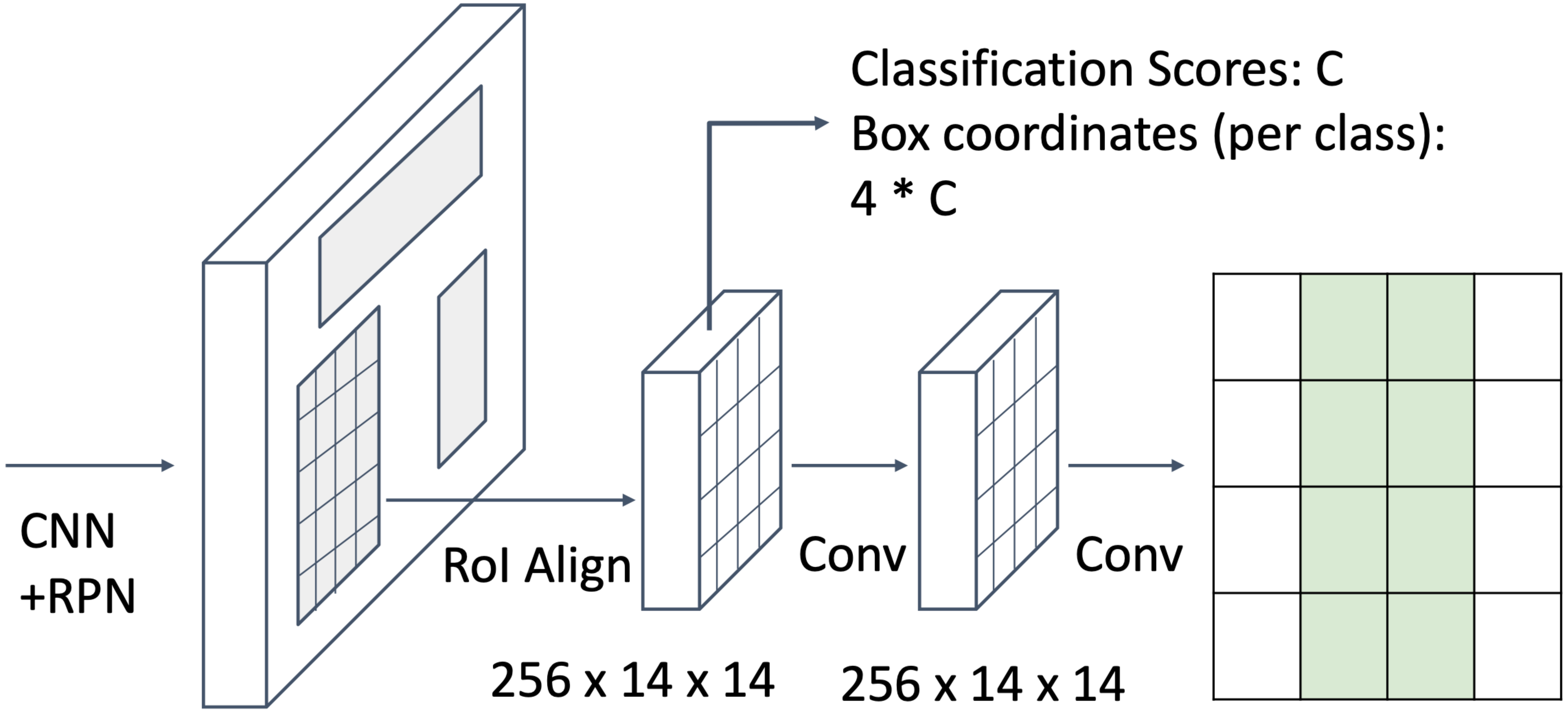
   b. **Object Regression:** predict transform from proposal box to object box

   c. **Mask Prediction:** predict a binary mask for every region




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

# Mask R-CNN


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

He et al., “Mask R-CNN”, ICCV 2017

# Mask R-CNN: Very Good Results!

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

# Mask R-CNN for Human Pose Estimation

Mask R-CNN

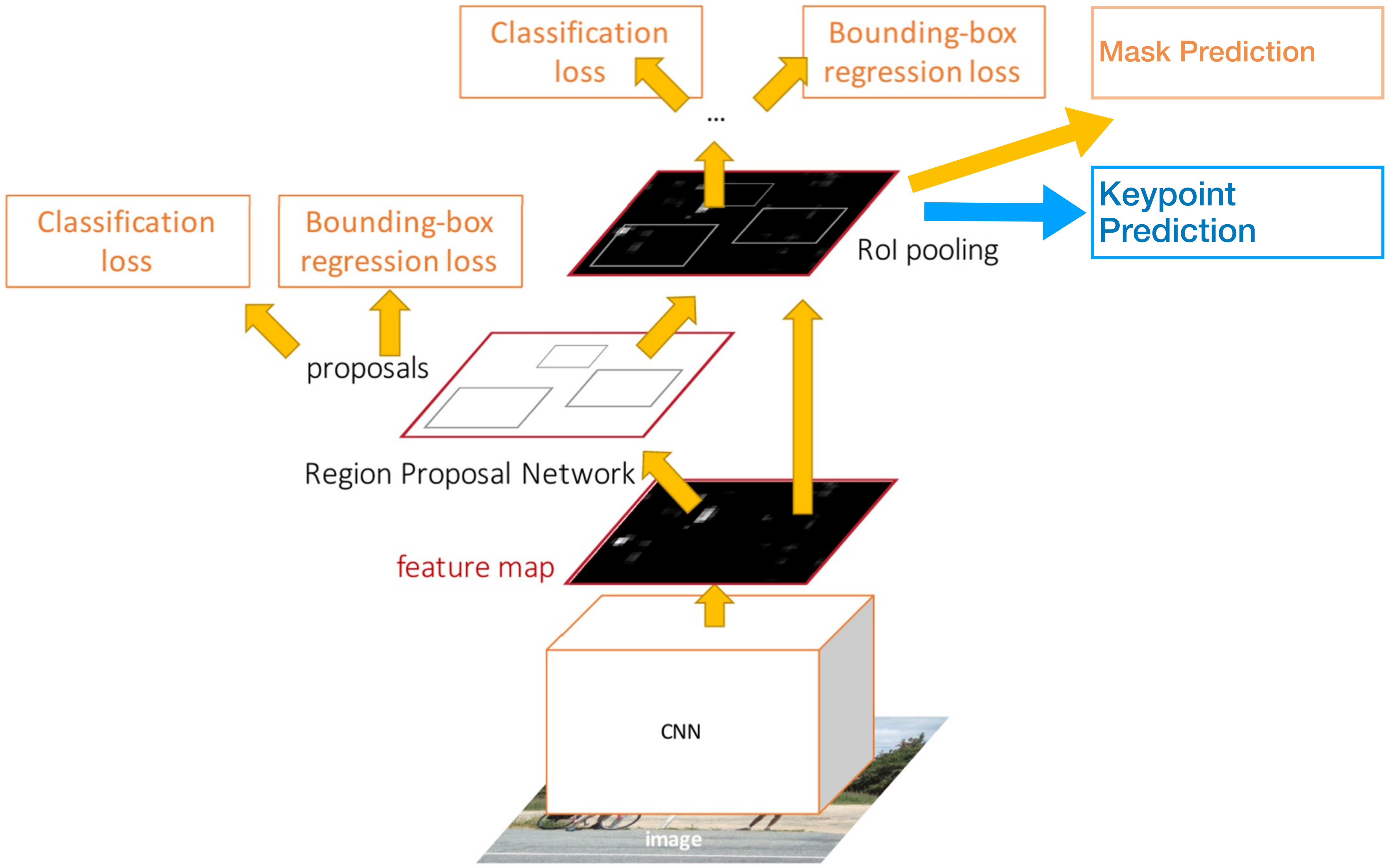
1. **Feature Extraction** at the image-level
2. **Regions of Interest** proposal from feature map
3. **In Parallel**

   a. **Object Classification:** classify proposals

   b. **Object Regression:** predict transform from proposal box to object box

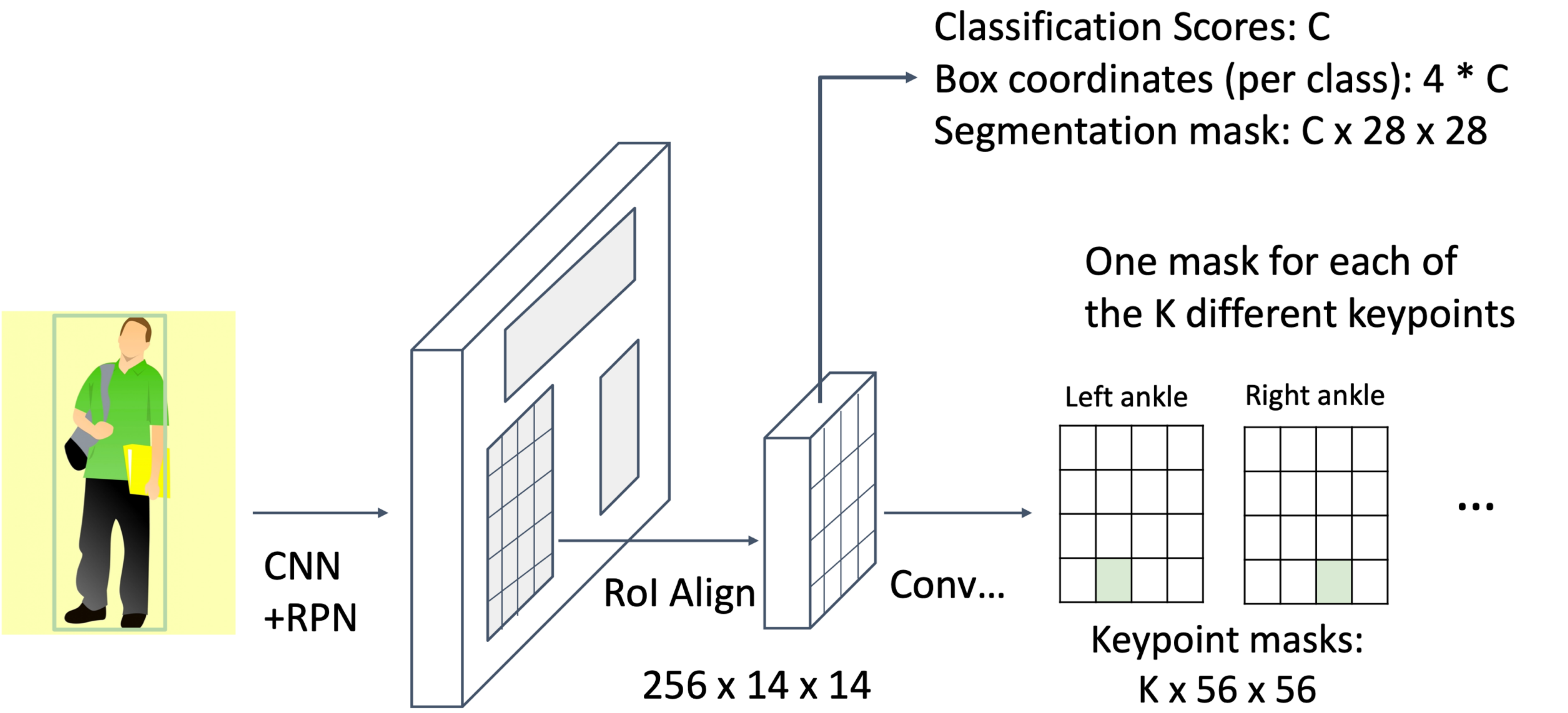
   c. **Mask Prediction:** predict a binary mask for every region

   d. **Keypoint Prediction:** predict binary mask for human key points




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

# Mask R-CNN for Human Pose Estimation




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

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

# Mask R-CNN for Human Pose Estimation

He et al., “Mask R-CNN”, ICCV 2017

# Two Stage vs One Stage Detectors

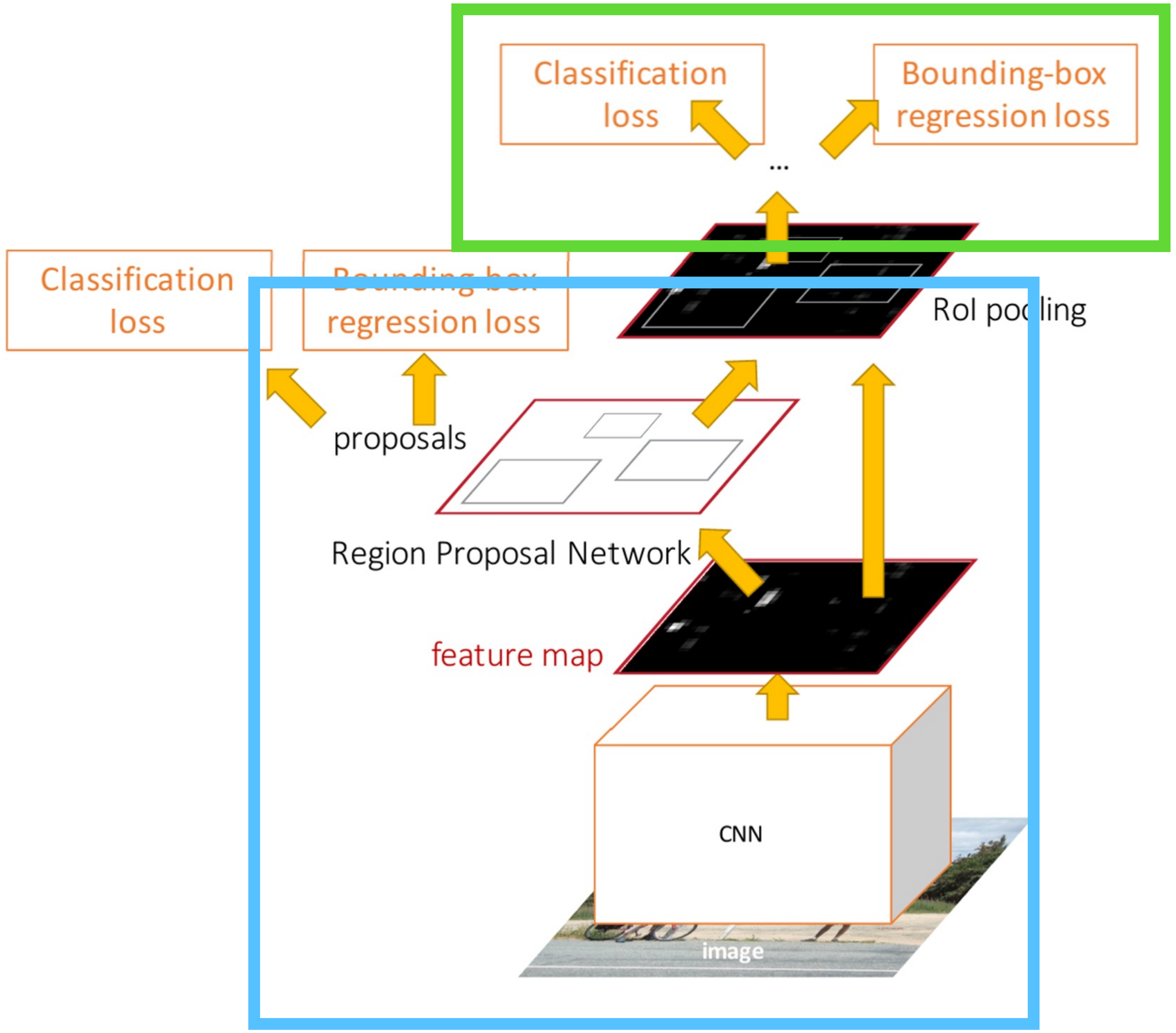
**Faster R-CNN is a two-stage object detector**

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




Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Re

Figure copyright 2015, Ross Girshick

DeepRob

Lecture 9
Object Detection
University of Michigan | Department of Robotics