AI6126 Advanced Computer Vision

Last update: 3 February 2022 10:50am

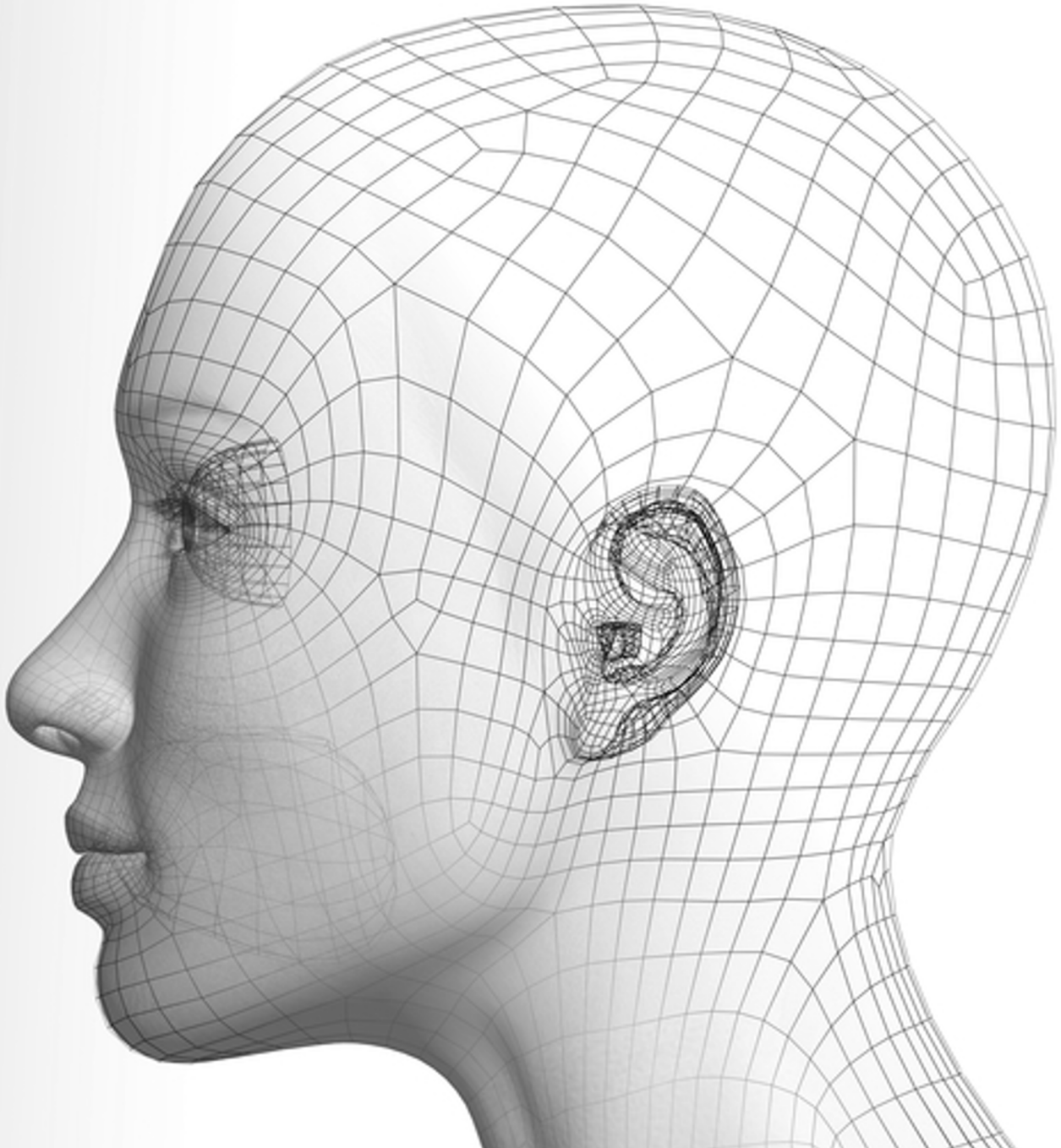
Object Detection

Chen-Change Loy

吕健勤

http://www.ntu.edu.sg/home/ccloy/

https://twitter.com/ccloy



Slide credits

- Justin Johnson EECS 498-007 / 598-005

So far …

Image classification

This image is CC0 public domain

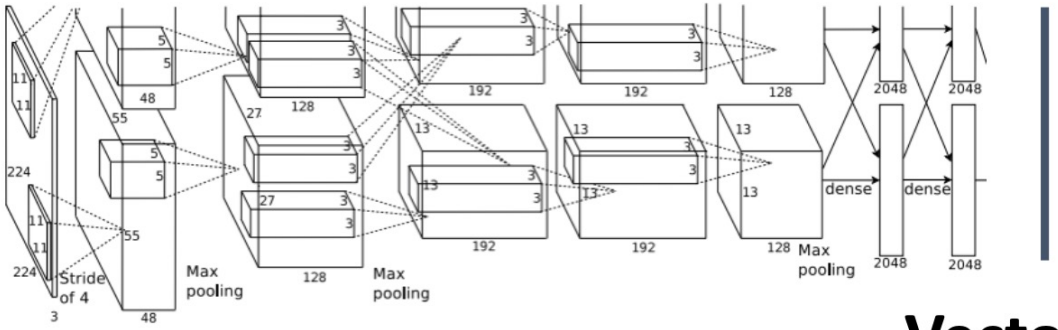


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Vector:
4096

Fully-Connected:
4096 to 1000

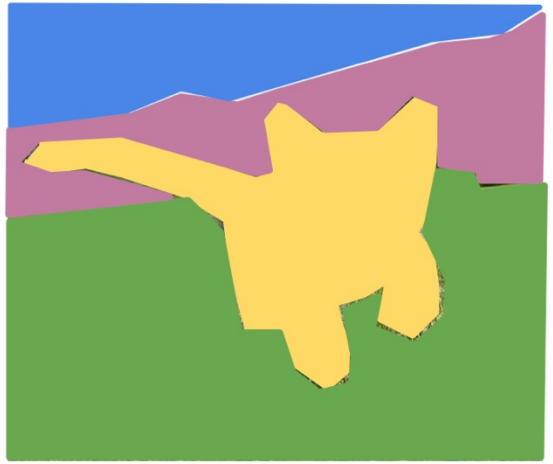
Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
…

Computer vision tasks

Classification

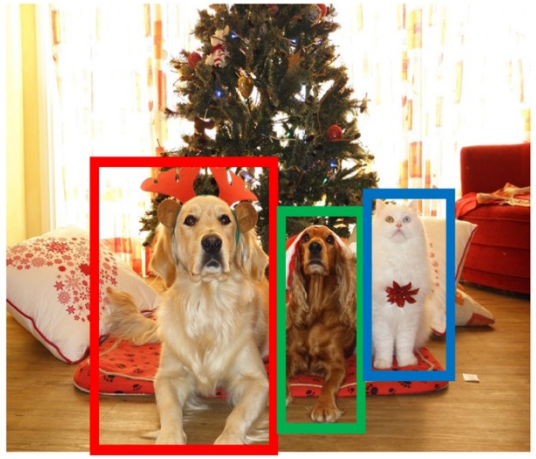
CAT

Semantic Segmentation



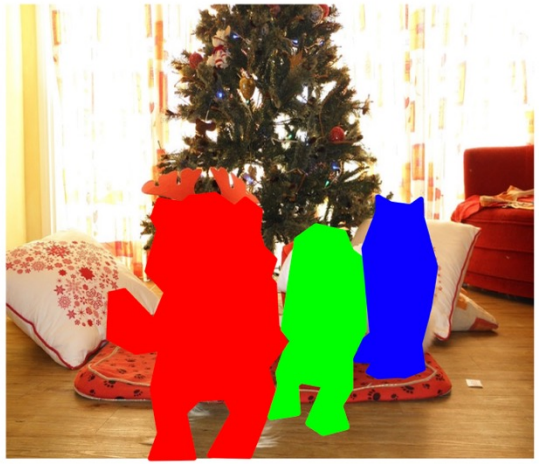
GRASS, CAT, TREE, SKY

Object Detection



DOG, DOG, CAT

Instance Segmentation



DOG, DOG, CAT

No spatial extent

No objects, just pixels

Multiple Objects

Outline

- Task definition
- Region-based CNN and basic concepts
  - Intersection over Union (IoU)
  - Non-Max Suppression (NMS)
  - Mean Average Precision (mAP)
- Fast R-CNN
- Faster R-CNN

- *Tutorial 2*: Faster-RCNN

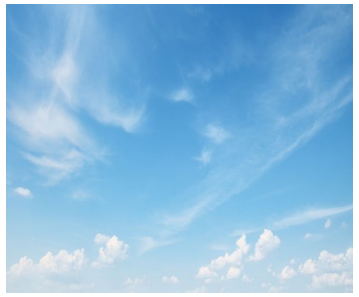
Task Definition

Task definition

- Things




- Stuff

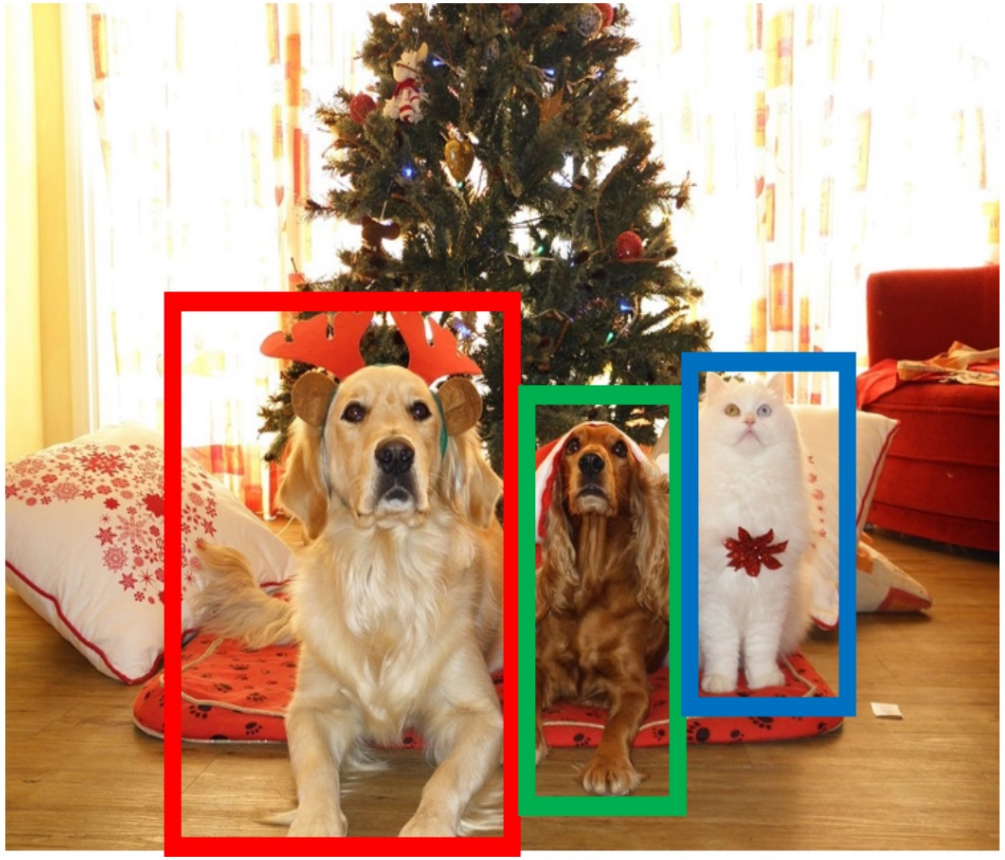



Task definition

Input: Single RGB Image

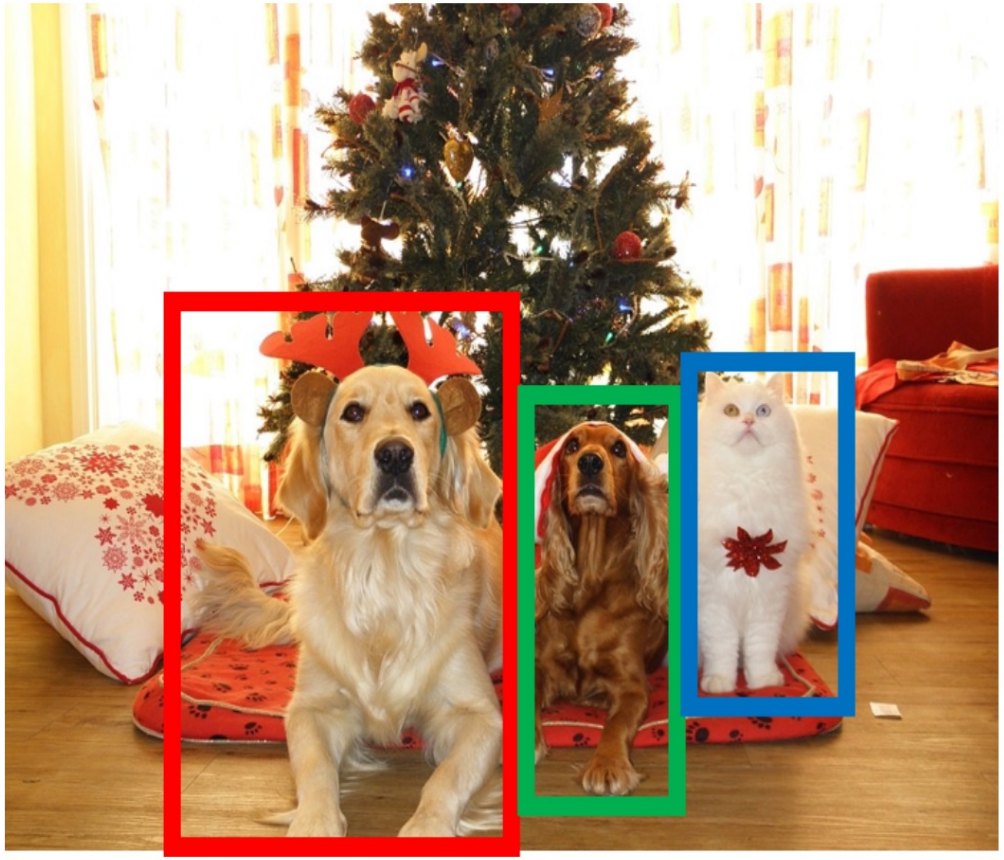
Output: A set of detected objects;
For each object predict:

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



Challenges

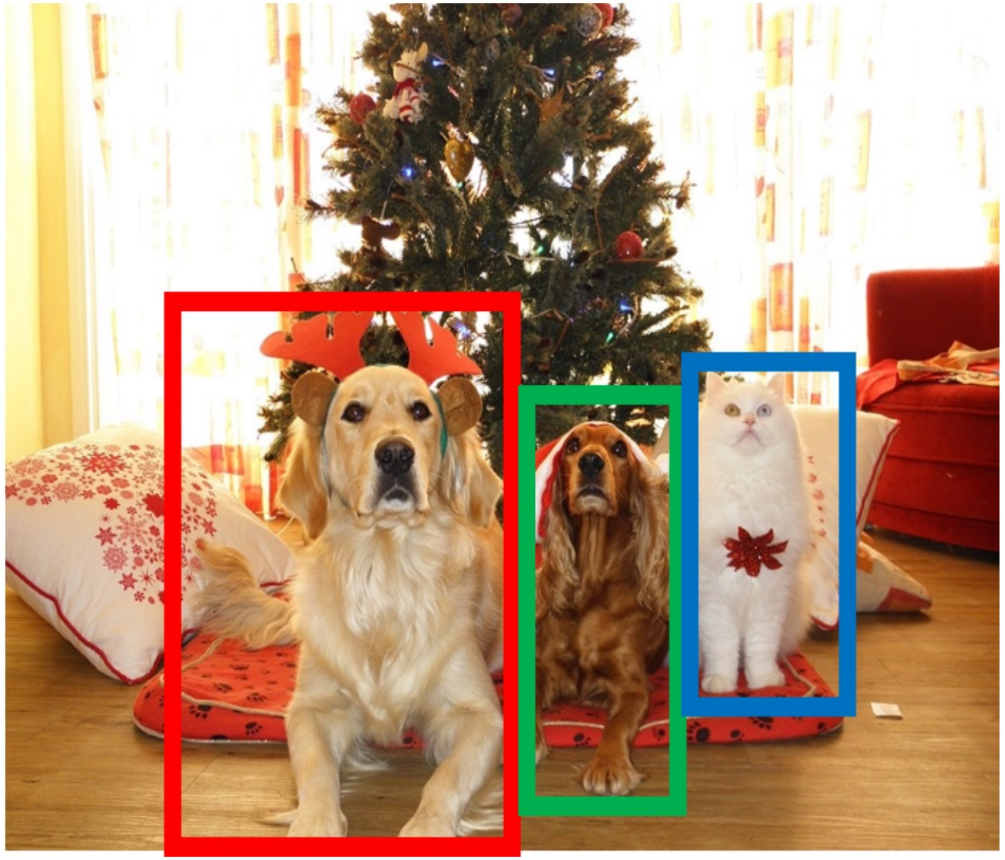
- **Multiple outputs**: Need to output variable numbers of objects per image
- **Multiple types of output**: Need to predict ”what” (category label) as well as “where” (bounding box)
- **Large images**: Classification works at 224x224; need higher resolution for detection, often ~800x600



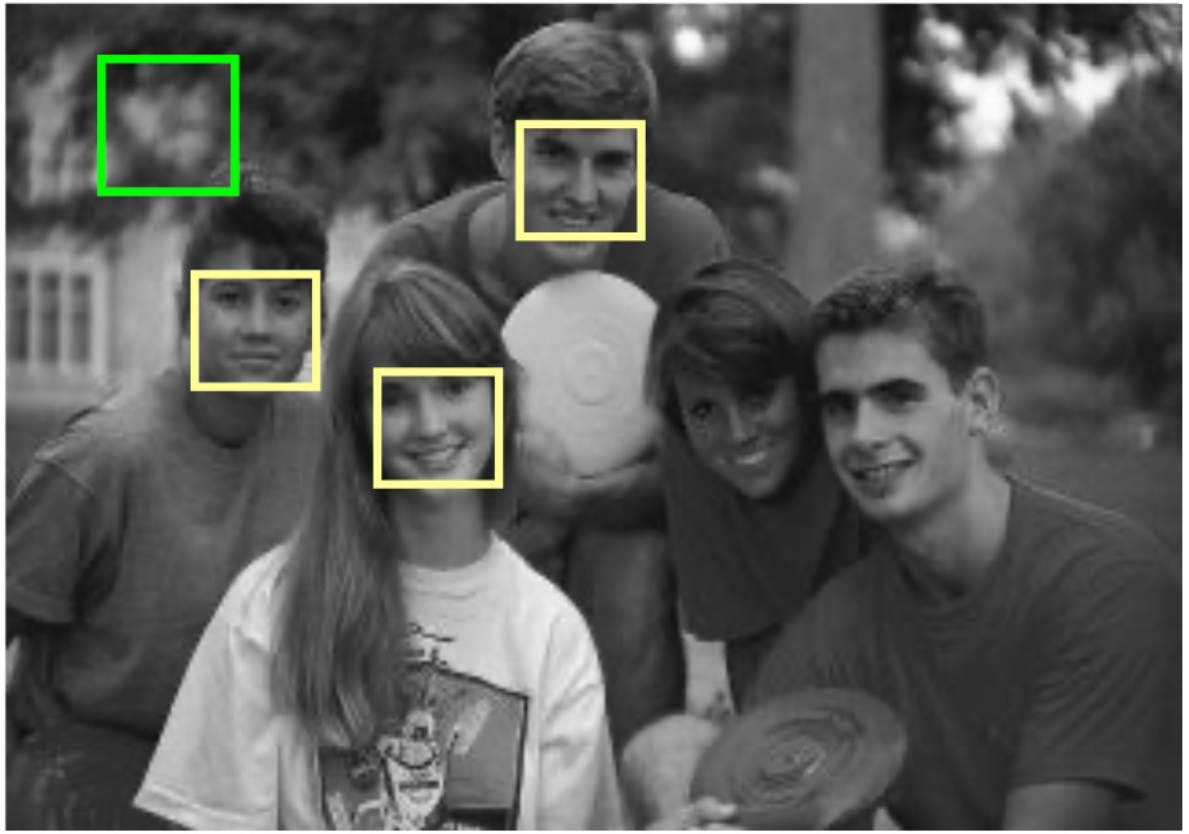
Challenges

There are many “close” **false positives**, corresponding to “close but not correct” bounding boxes.

The detector must find the true positives while **suppressing** these close false positives



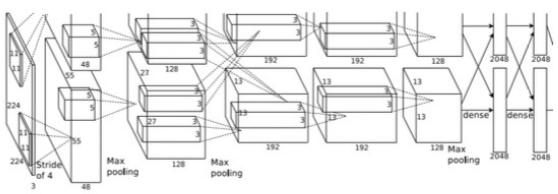
Challenges



The pipeline

Detecting a single object

This image is CC0 public domain

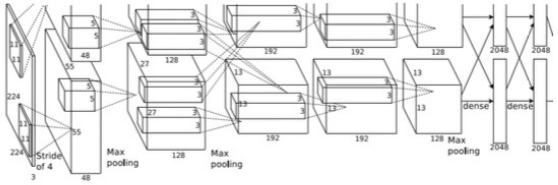


Vector:
4096

The pipeline

Detecting a single object

This image is CC0 public domain



Vector:
4096

Fully Connected:
4096 to 1000

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

"What"

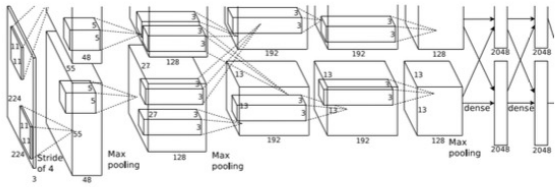
Softmax Loss

Correct label:
Cat

The pipeline

Detecting a single object

This image is CC0 public domain



Vector:
4096

Fully Connected:
4096 to 1000

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

"What"

Softmax Loss

Correct label:
Cat

Fully Connected:
4096 to 4

Box Coordinates
(x, y, w, h)

"Where"

L2 Loss

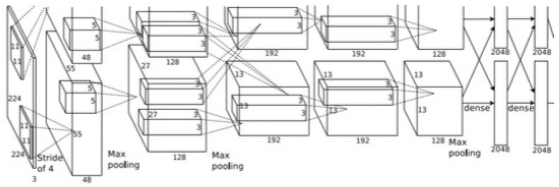
Correct box:
(x', y', w', h')

Treat localization as a regression problem!

The pipeline

Detecting a single object

This image is CC0 public domain



Vector: 4096

Fully Connected: 4096 to 1000

"What"

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

Correct label: Cat

Softmax Loss

Multitask loss

Weighted Sum

Loss

Fully Connected: 4096 to 4

"Where"

Box Coordinates (x, y, w, h)

L2 Loss

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

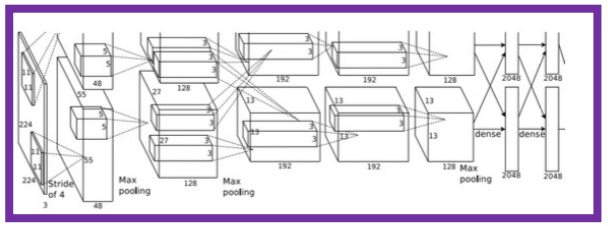
Treat localization as a regression problem!

The pipeline

Detecting a single object

This image is CC0 public domain

Often pretrained on ImageNet (Transfer learning)



Vector: 4096

Fully Connected: 4096 to 1000

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

"What"

Correct label: Cat

Softmax Loss

Multitask loss

Weighted Sum → **Loss**

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

L2 Loss

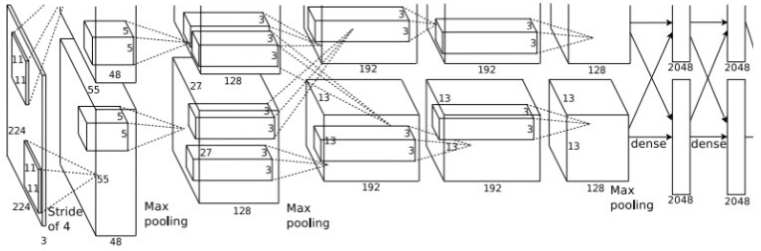
Correct box: (x', y', w', h')

"Where"

Treat localization as a regression problem!

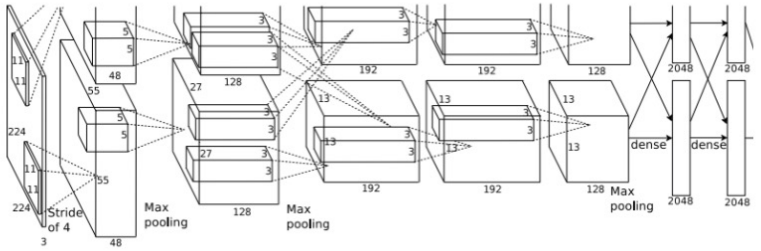
The pipeline

Detecting multiple objects – **need different numbers of outputs per image**



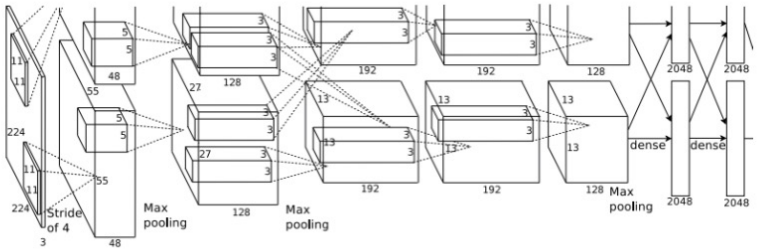
CAT: (x, y, w, h)

4 numbers



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

12 numbers



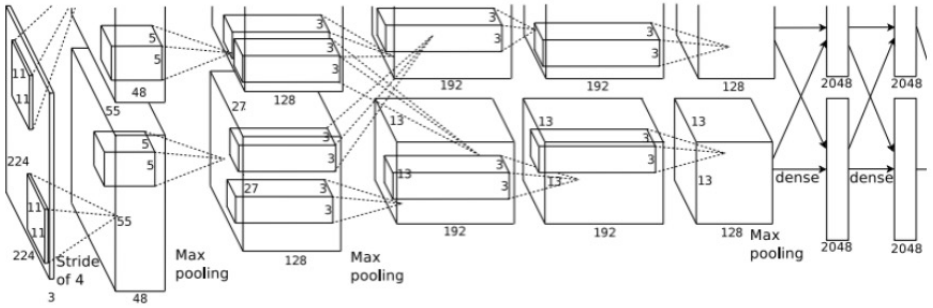
DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
....

Many numbers!

Sliding window

Detecting multiple objects

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

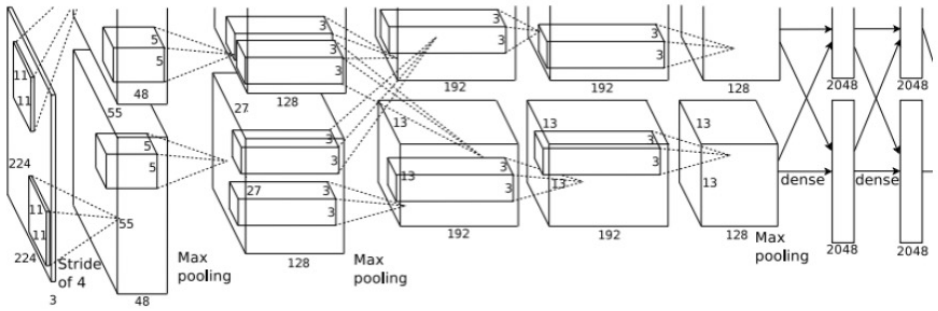


Dog? NO
Cat? NO
Background? YES

Sliding window

Detecting multiple objects

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

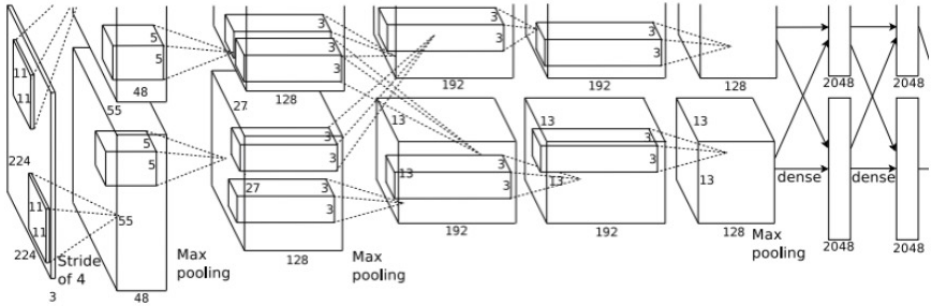


Dog? YES
Cat? NO
Background? NO

Sliding window

Detecting multiple objects

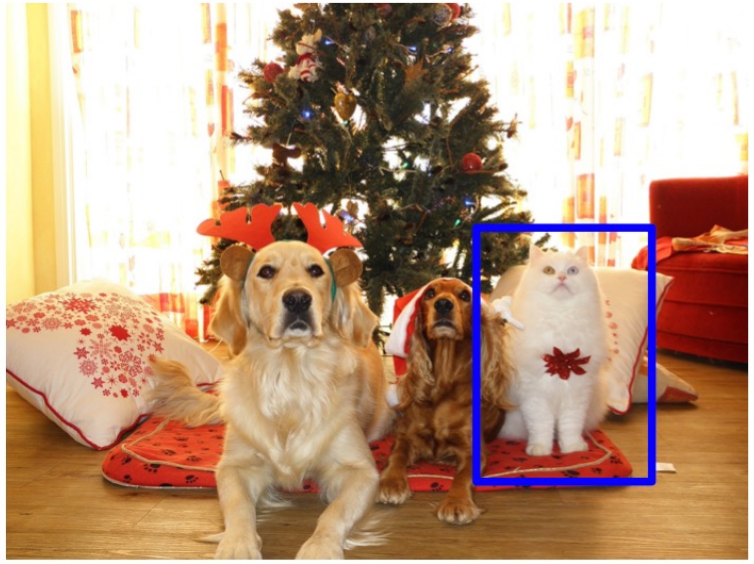
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



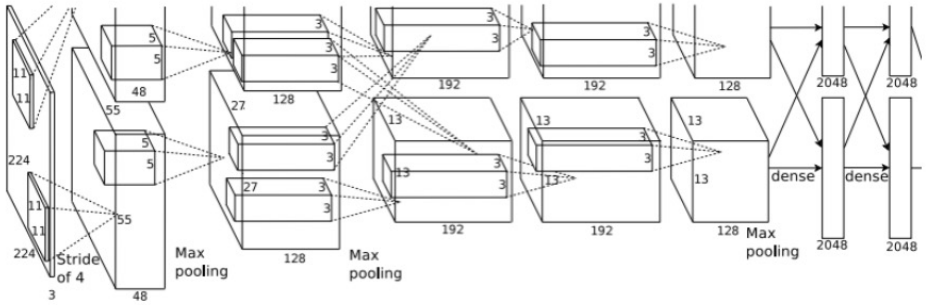
Dog? YES
Cat? NO
Background? NO

Sliding window

Detecting multiple objects



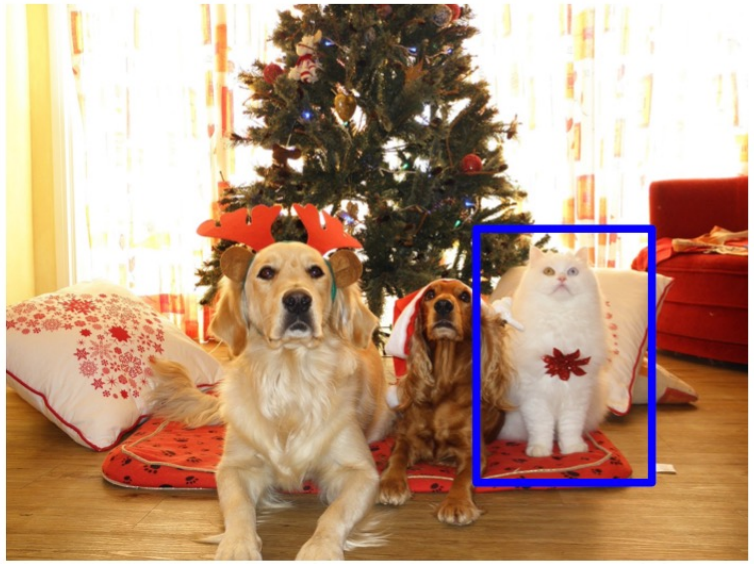
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? NO
Cat? YES
Background? NO

Sliding window

Detecting multiple objects

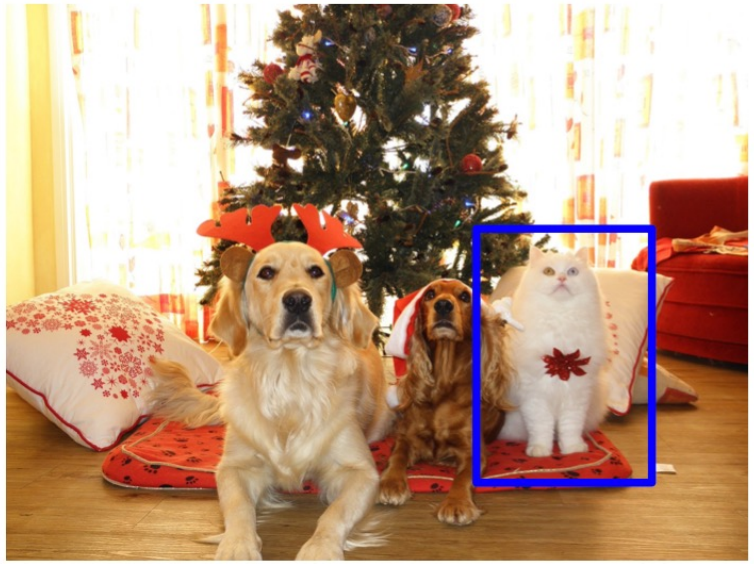


Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

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

Sliding window

Detecting multiple objects



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

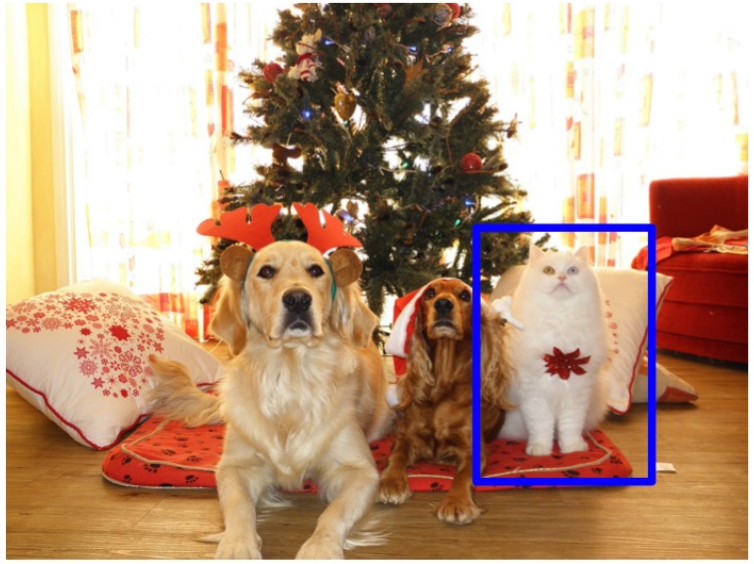
Question How many possible boxes are there in an image of size H x W?

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)

Sliding window

Detecting multiple objects



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)

Even worse if we consider boxes of different sizes and ratios

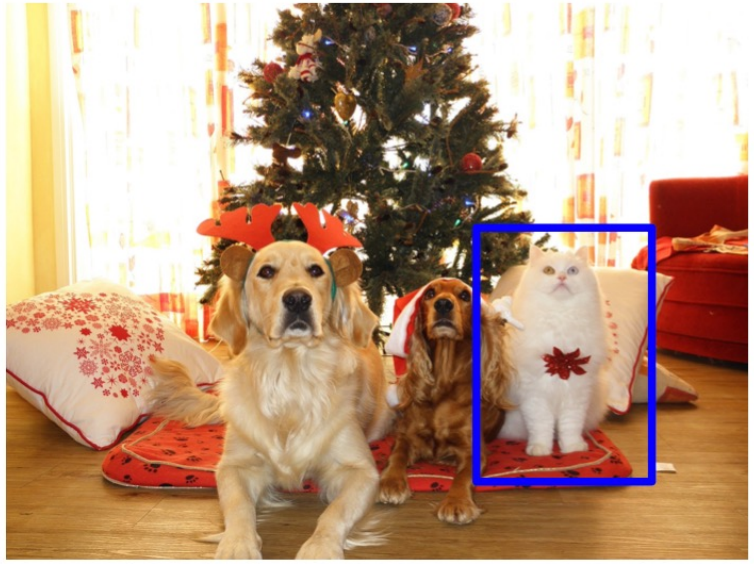
Total possible boxes:

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

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

Sliding window

Detecting multiple objects



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)

Even worse if we consider boxes of different sizes and ratios

Total possible boxes:

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

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

Outline

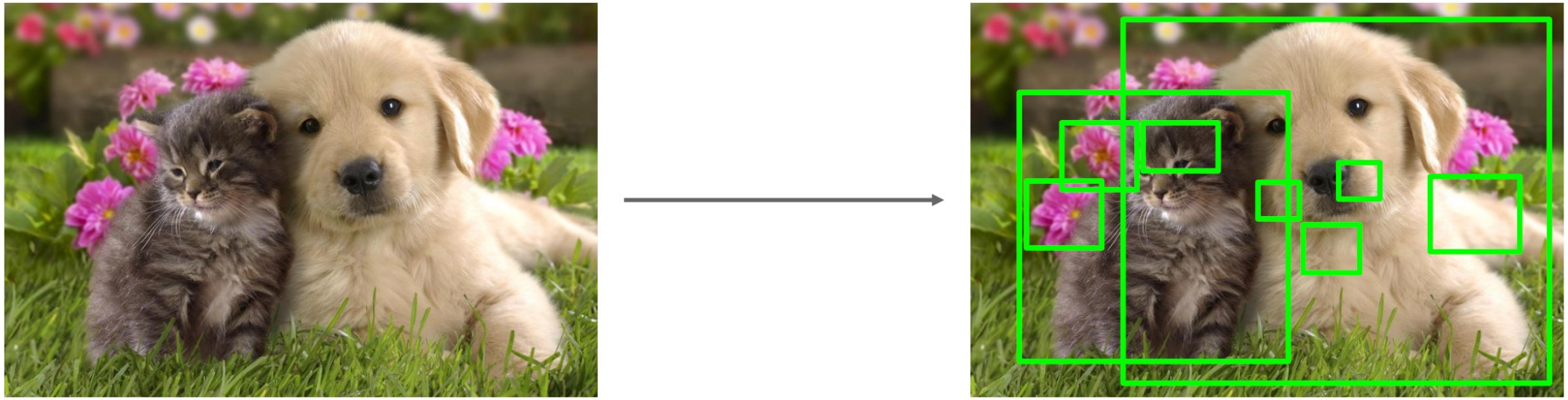
- Task definition
- Region-based CNN and basic concepts
  - Intersection over Union (IoU)
  - Non-Max Suppression (NMS)
  - Mean Average Precision (mAP)
- Fast R-CNN
- Faster R-CNN

- *Tutorial 2*: Faster-RCNN

Region-based CNN and basic concepts

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

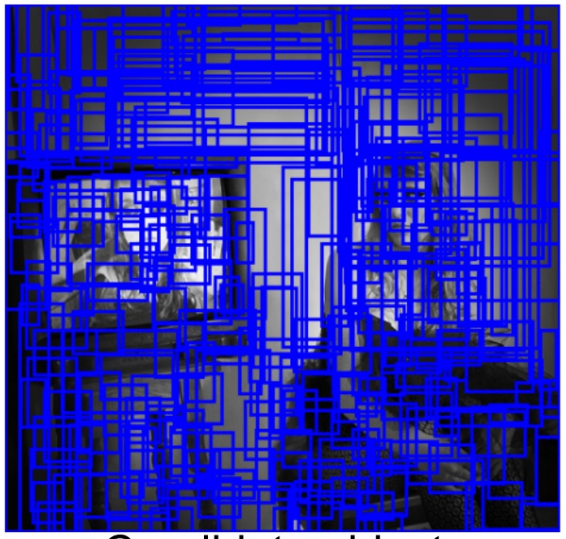
Selective Search

Step 1: Generate initial sub-segmentation

Goal: Generate many regions, each of which belongs to at most one object.

Input Image

Segmentation



Candidate objects

P. F. Felzenszwalb and D. P. Huttenlocher. “Efficient Graph-Based Image Segmentation.” IJCV, 59:167–181, 2004.

Selective Search

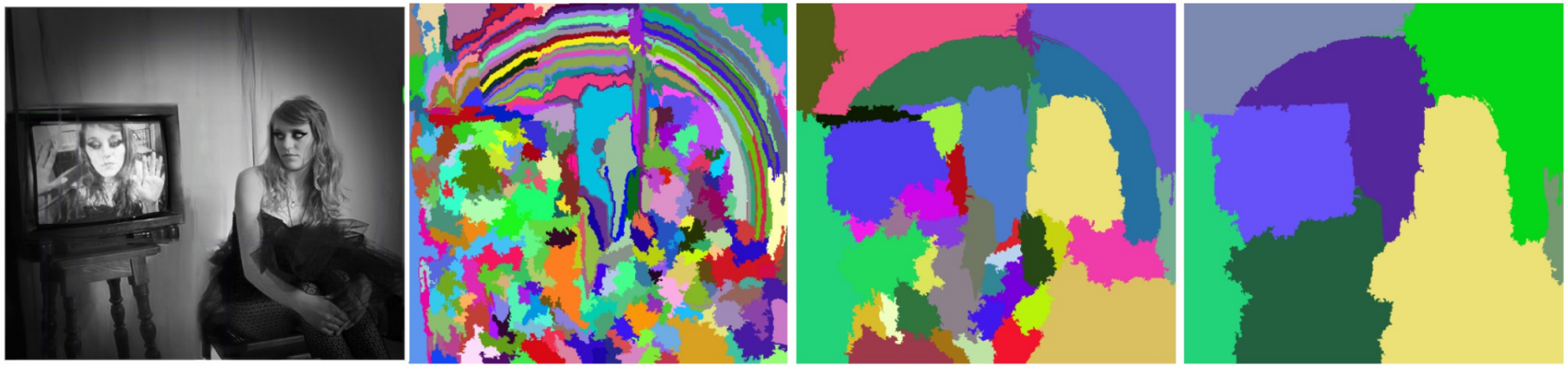
Step 2: Recursively combine similar regions into larger ones.

Greedy algorithm:

1. From set of regions, choose two that are most *similar*.
2. Combine them into a single, larger region.
3. Repeat until only one region remains.

Color Similarity – HSV (hue, saturation, value)
Texture Similarity – HOG-like features

This yields a hierarchy of successively larger regions, just like we want.



Input Image | Initial Segmentation | After some iterations | After more iterations

Selective Search

Step 2: Use the generated regions to produce candidate object locations.

Input Image

Region-based CNN

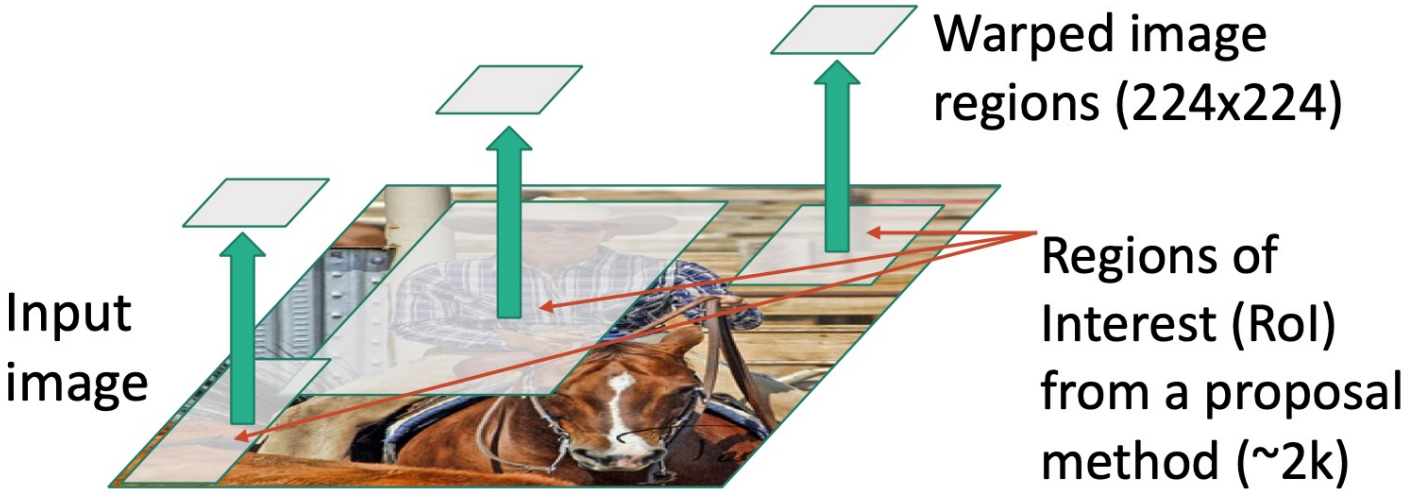
Input image

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014. Figure copyright Ross Girshick, 2015

Region-based CNN

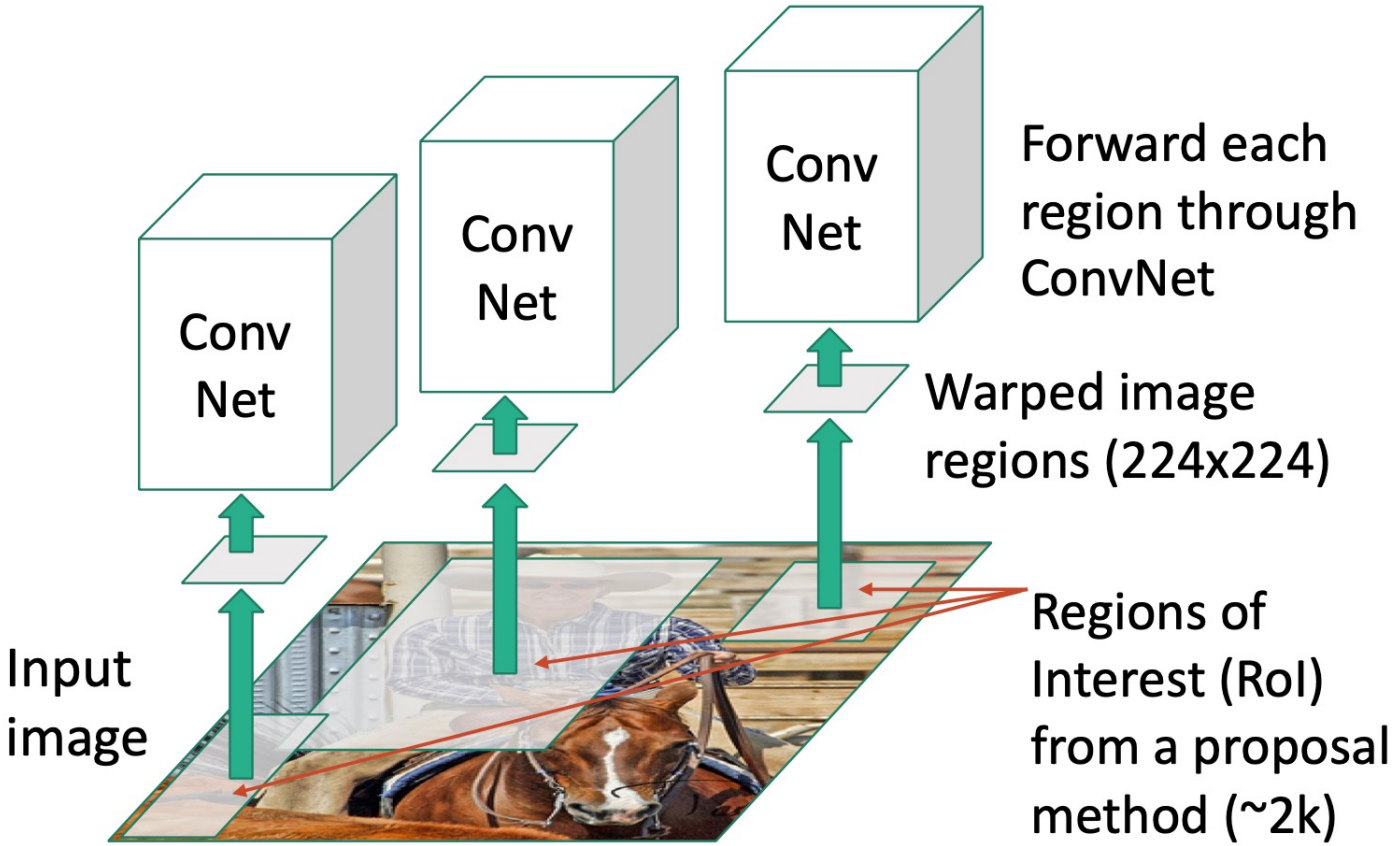
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014. Figure copyright Ross Girshick, 2015

Region-based CNN



Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014. Figure copyright Ross Girshick, 2015

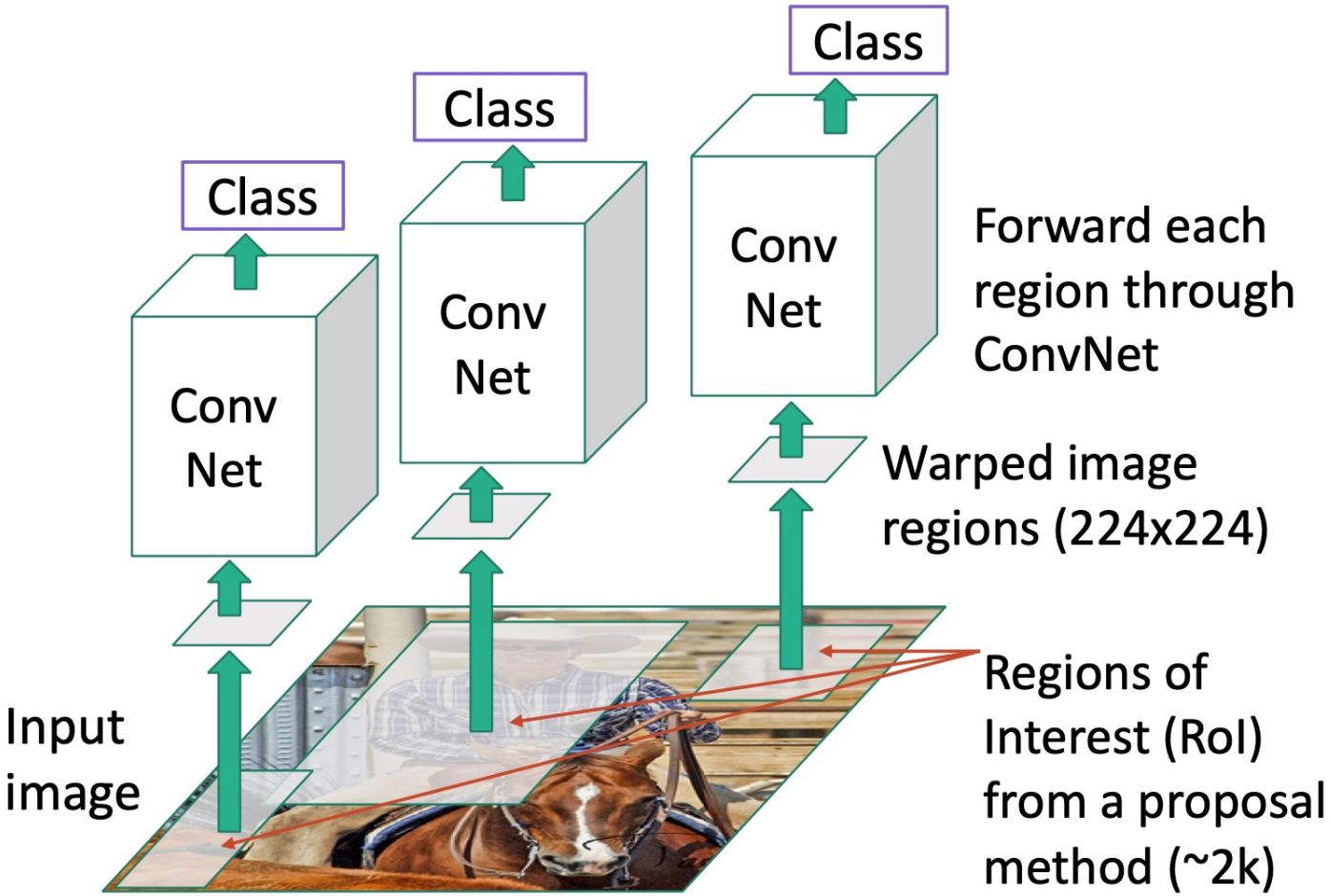
Region-based CNN



Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014. Figure copyright Ross Girshick, 2015

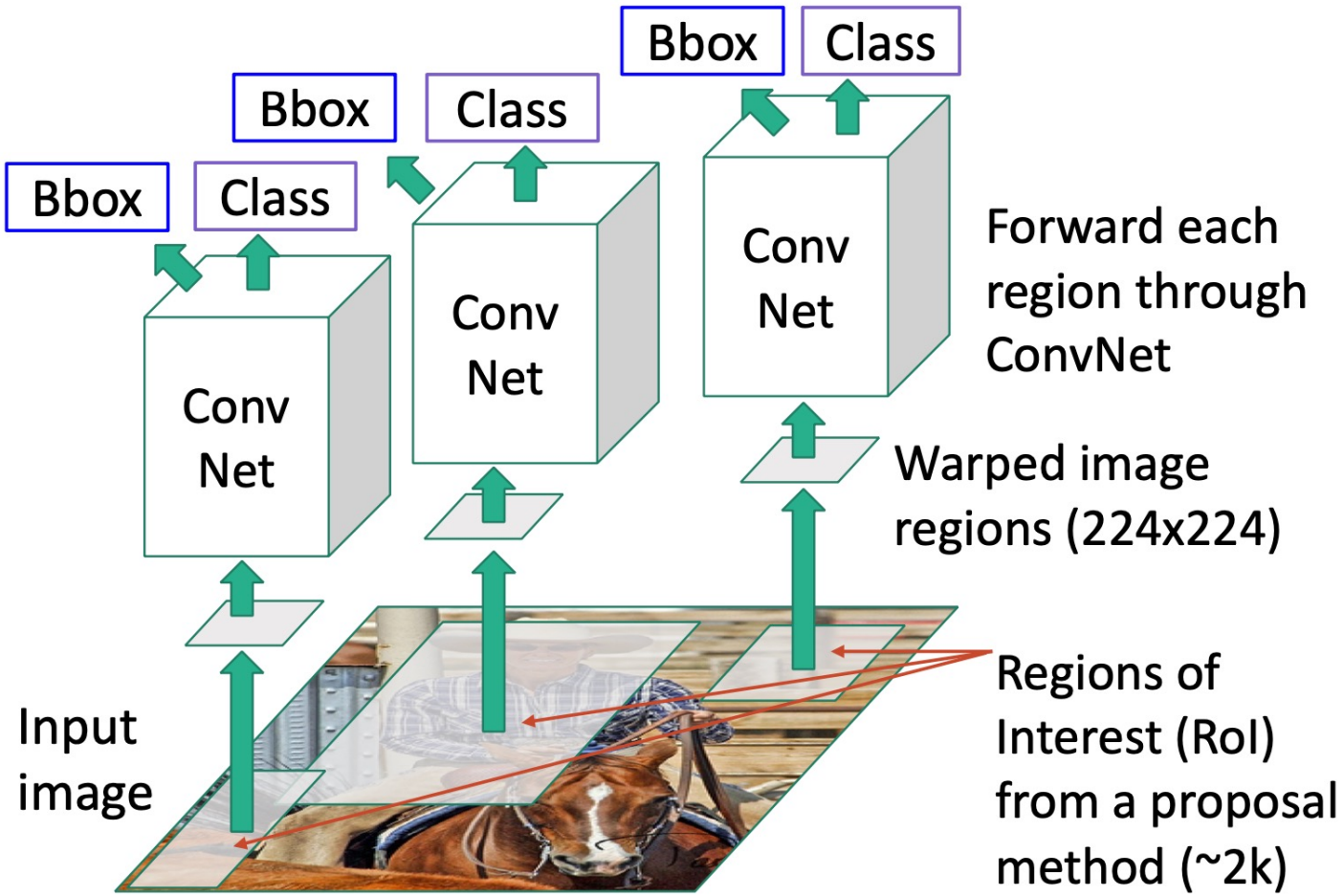
Region-based CNN

Classify each region



Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014. Figure copyright Ross Girshick, 2015

Region-based CNN

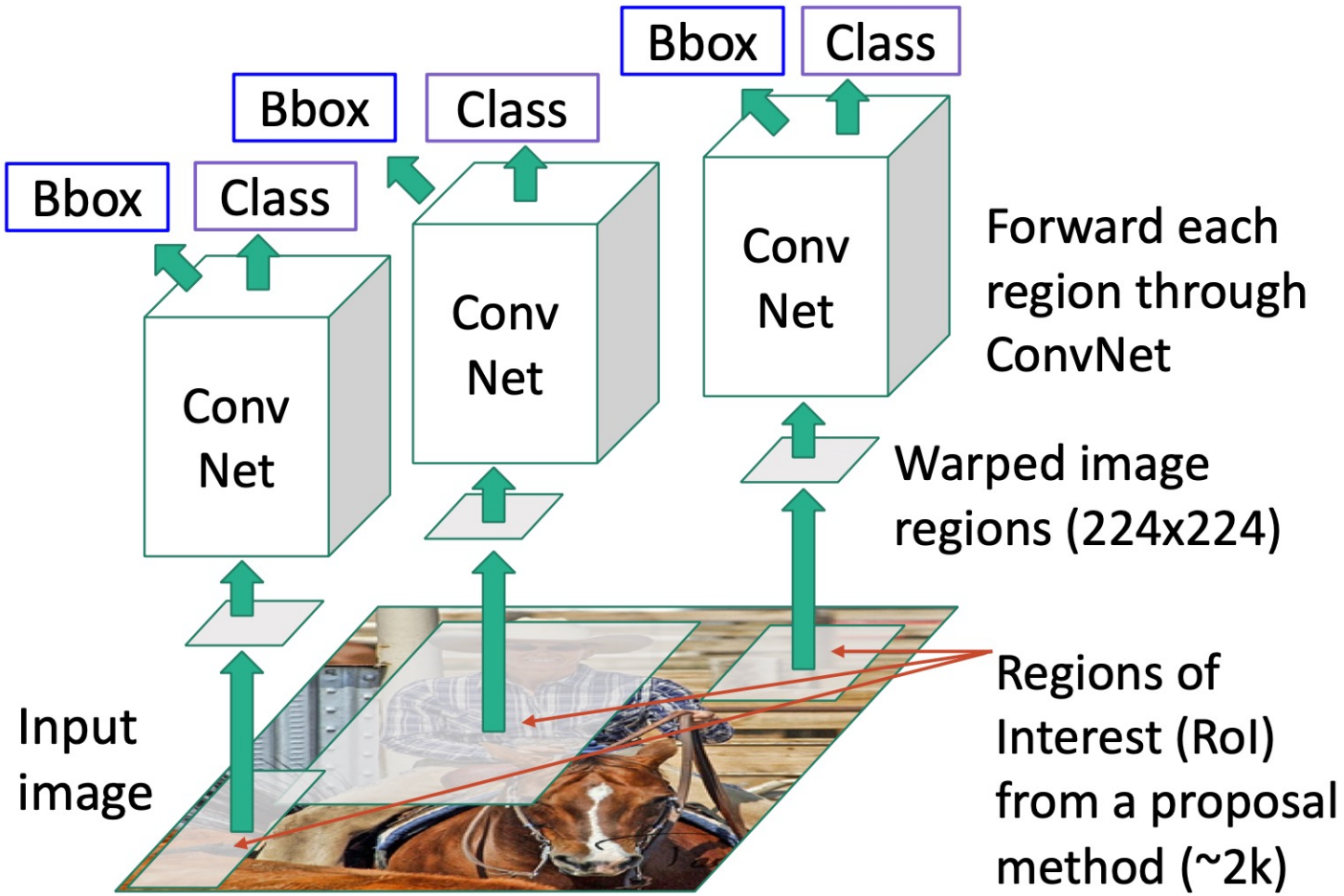


Classify each region

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

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

Region-based CNN



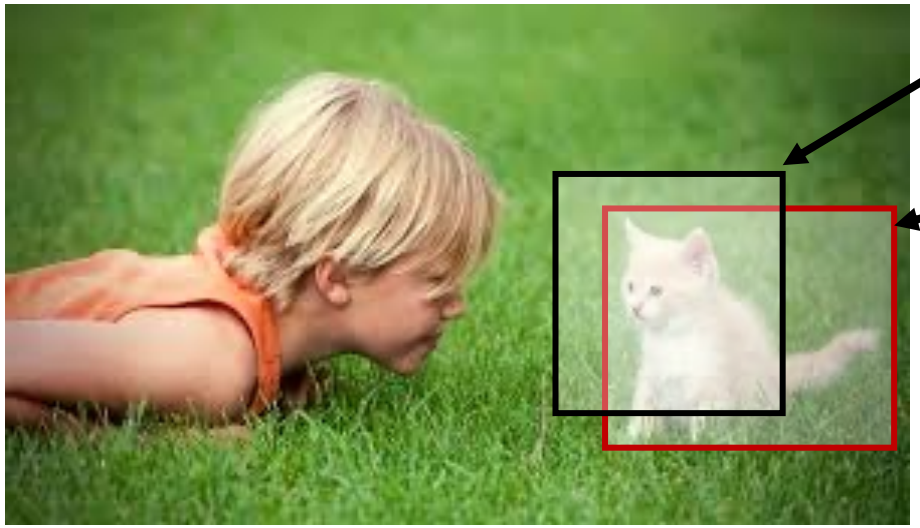
Classify each region

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

What is the target values (t_x, t_y, t_h, t_w) ?

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

Region-based CNN



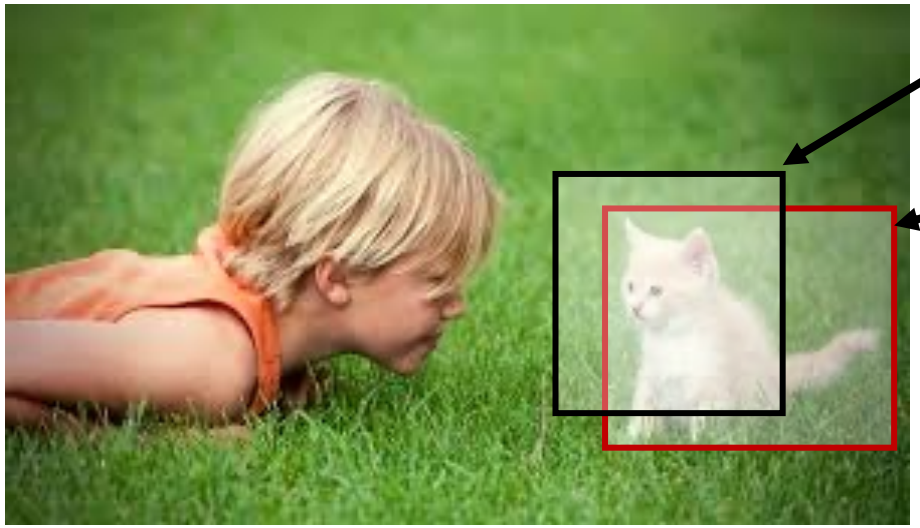
Classify each region

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

What is the target values (t_x, t_y, t_h, t_w) ?

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

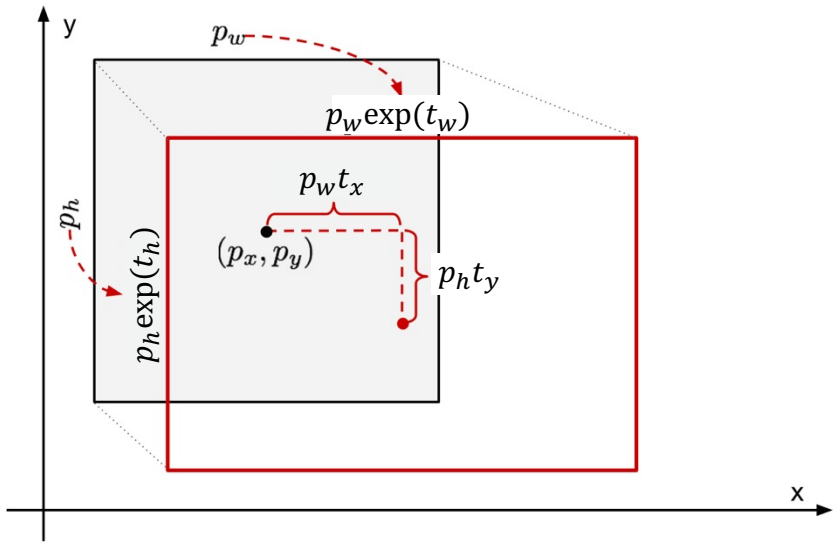
Region-based CNN



Classify each region

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

What is the target values (t_x, t_y, t_h, t_w) ?



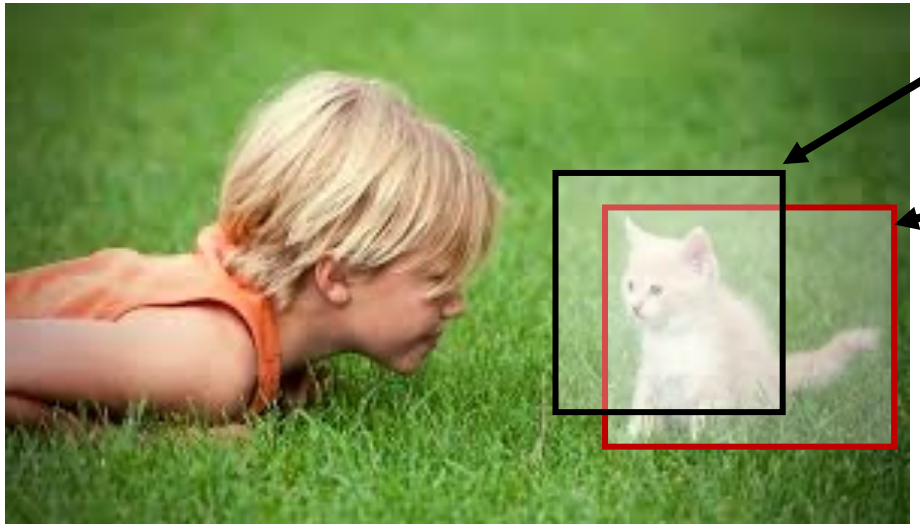
Region proposal: (p_x, p_y, p_h, p_w)
Transform: (t_x, t_y, t_h, t_w)
Output box: (b_x, b_y, b_h, b_w)

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

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

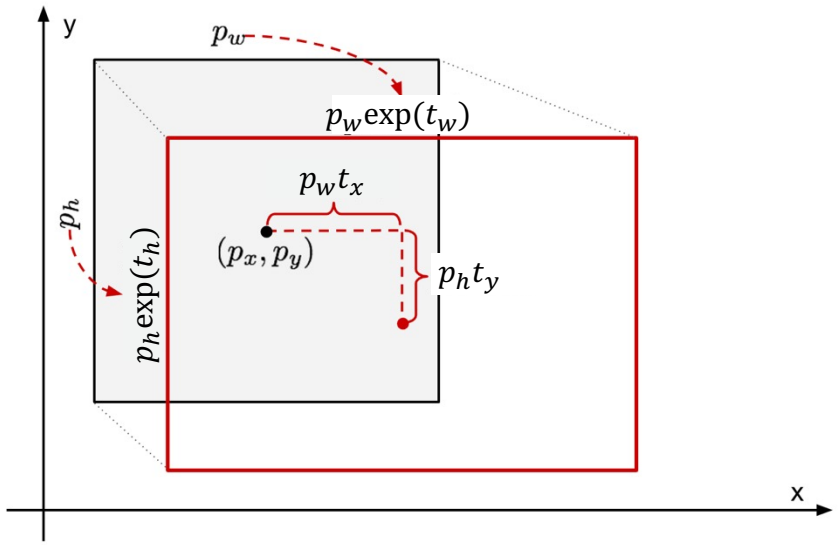
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014. Figure copyright Ross Girshick, 2015

Region-based CNN



Classify each region

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



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

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

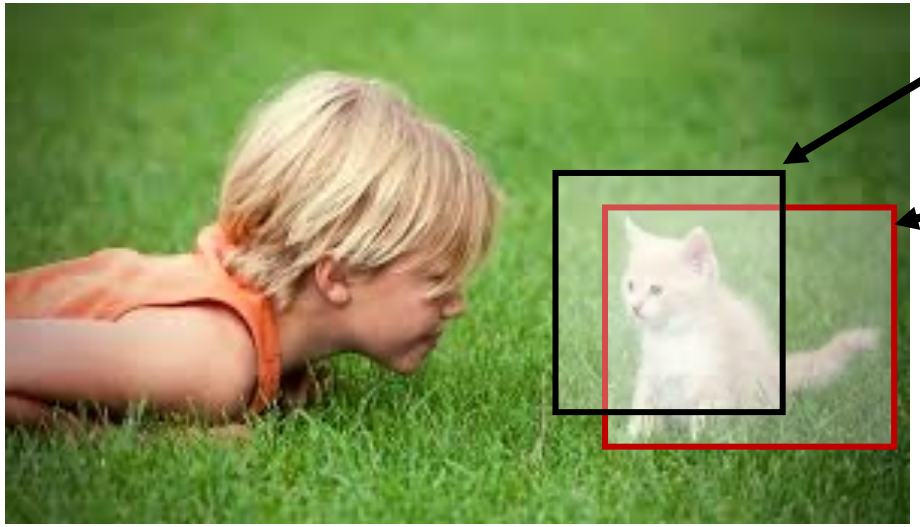
Log-space scale transform:
$b_w = p_w \exp(t_w) \quad b_h = p_h \exp(t_h)$

What is the target values (t_x, t_y, t_h, t_w) ?

$$\left(\frac{b_x - p_x}{p_w} , \frac{b_y - p_y}{p_h} , \log \frac{b_w}{p_w} , \log \frac{b_h}{p_h} \right)$$

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

Region-based CNN

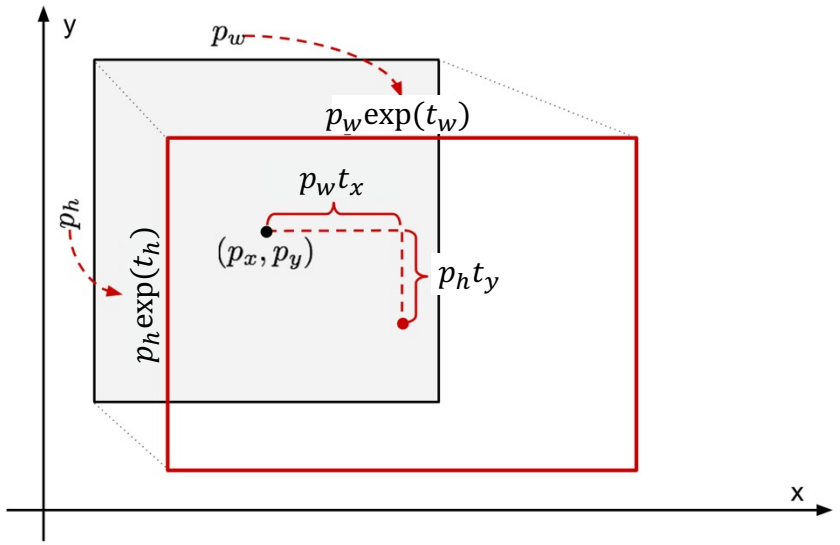


proposal

its corresponding ground truth box coordinates

Classify each region

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



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

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

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

What is the target values (t_x, t_y, t_h, t_w) ?

$$\left(\frac{b_x - p_x}{p_w} , \frac{b_y - p_y}{p_h} , \log \frac{b_w}{p_w} , \log \frac{b_h}{p_h} \right)$$

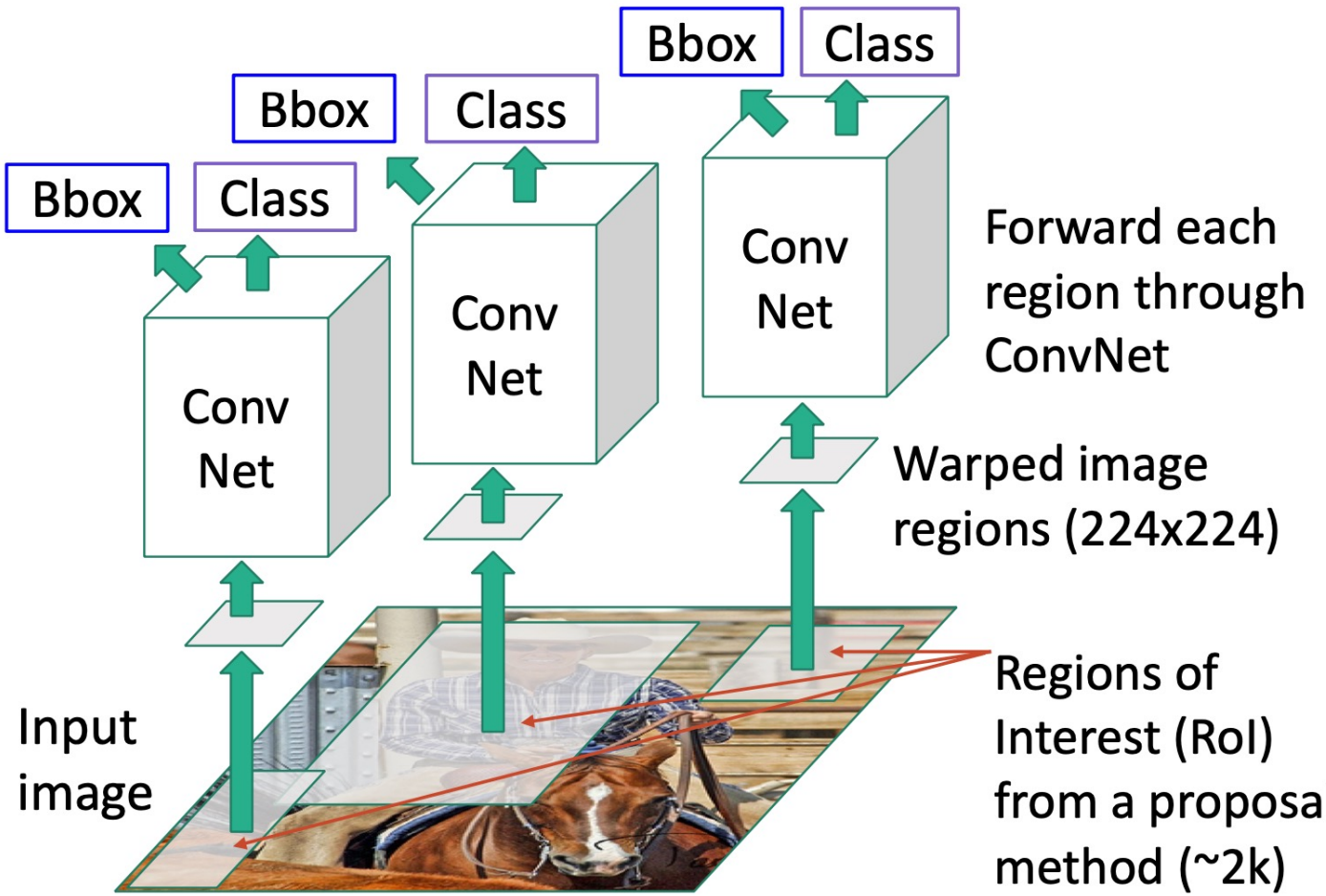
A standard regression model can solve the problem by minimizing the sum of squared loss with regularization.

$$\mathcal{L}_{\text{reg}} = \sum_{i \in \{x,y,w,h\}} (t_i - d_i(\mathbf{p}))^2 + \lambda \|\mathbf{w}\|^2$$

Output from bounding box regression function

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

Region-based CNN



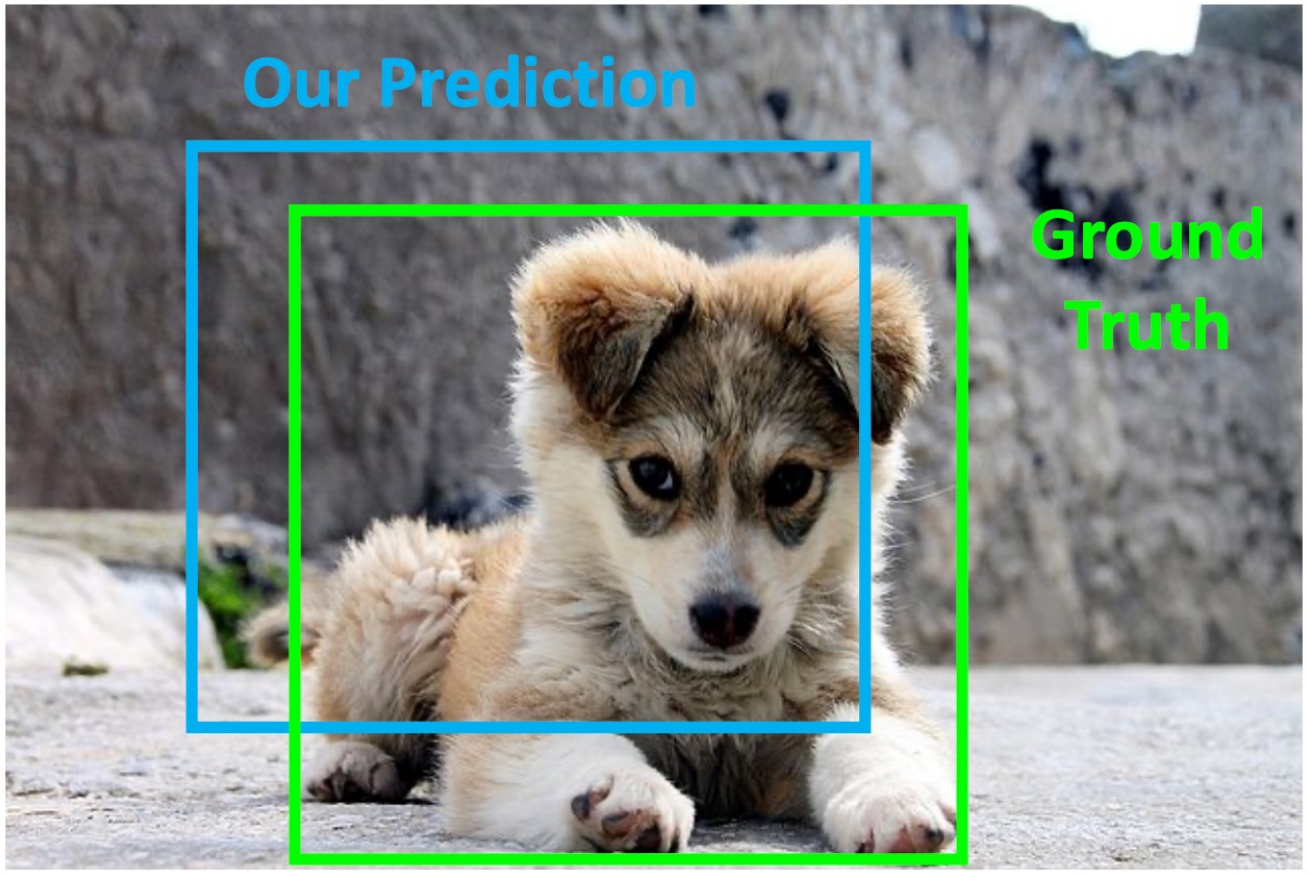
Test time
Input: Single RGB Image

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

Comparing Boxes: Intersection over Union (IoU)

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

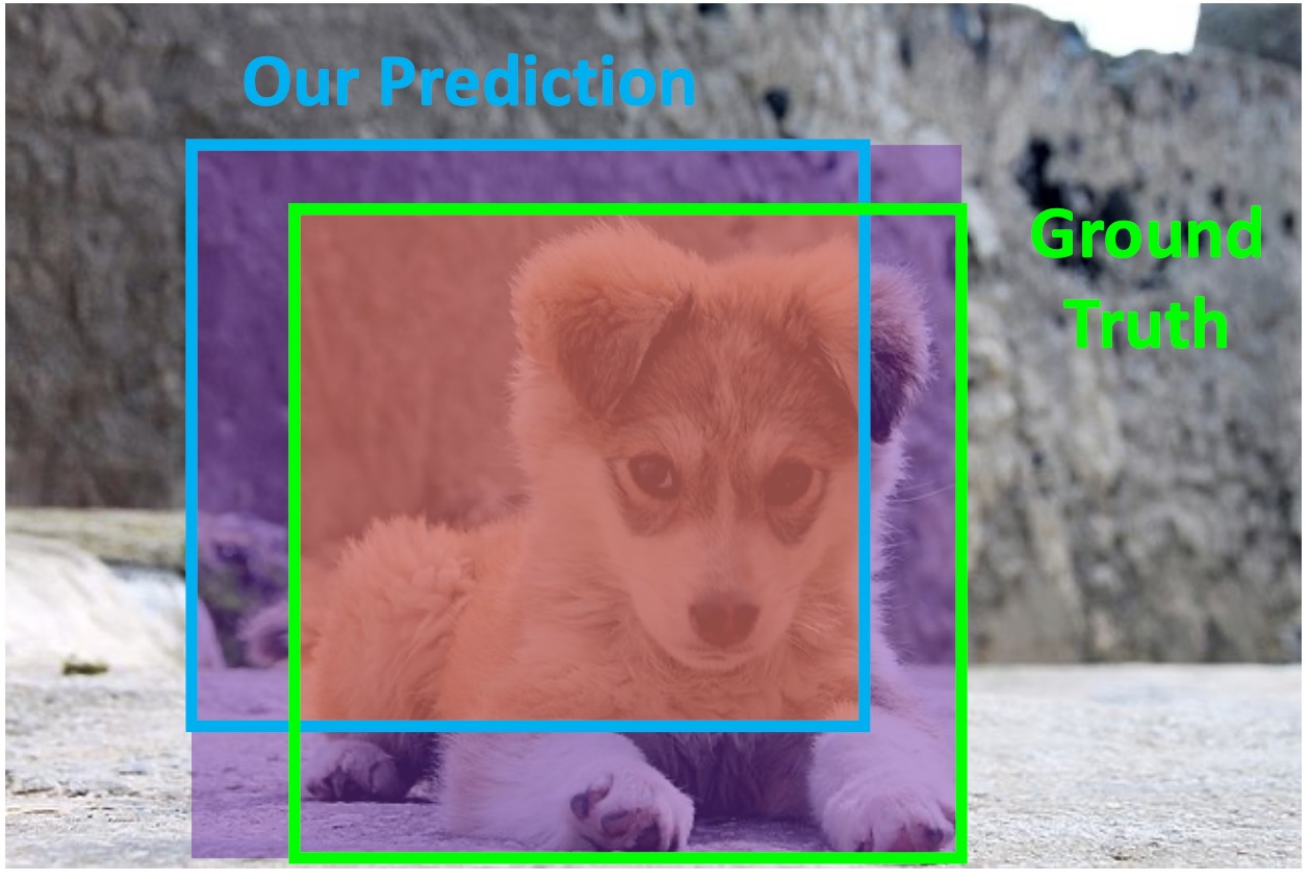


Comparing Boxes: Intersection over Union (IoU)

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

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

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



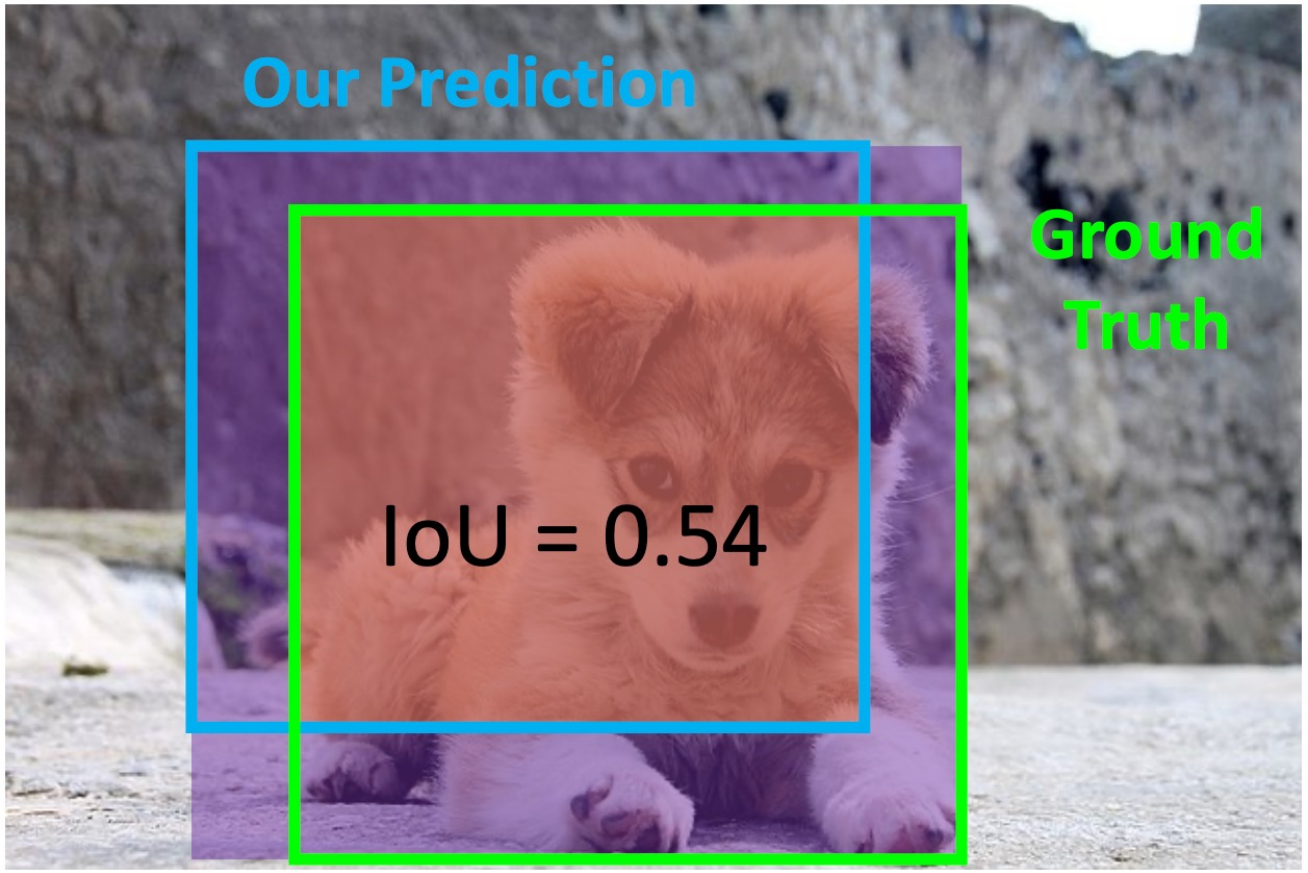
Comparing Boxes: Intersection over Union (IoU)

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

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

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

IoU > 0.5 is “decent”



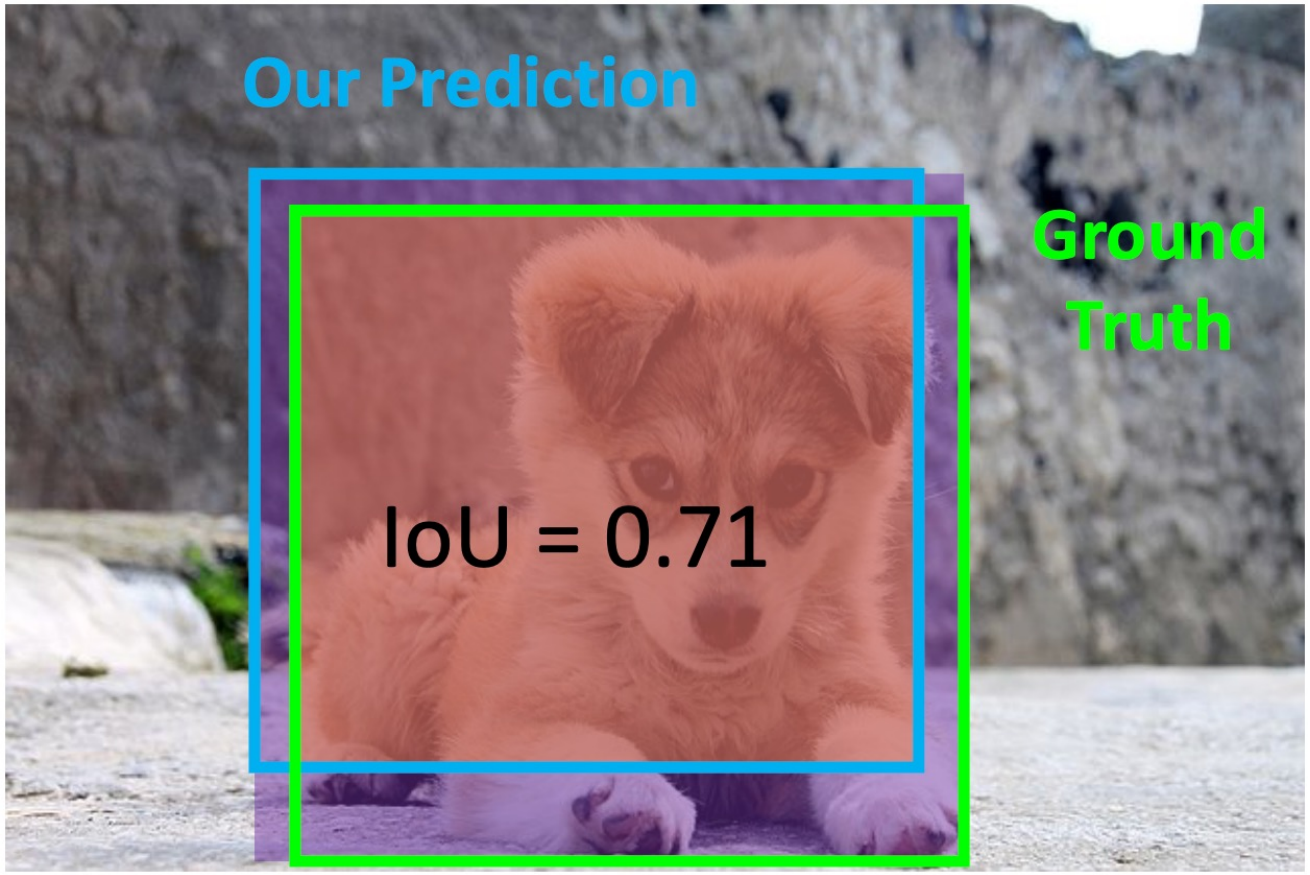
Comparing Boxes: Intersection over Union (IoU)

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

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

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

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?

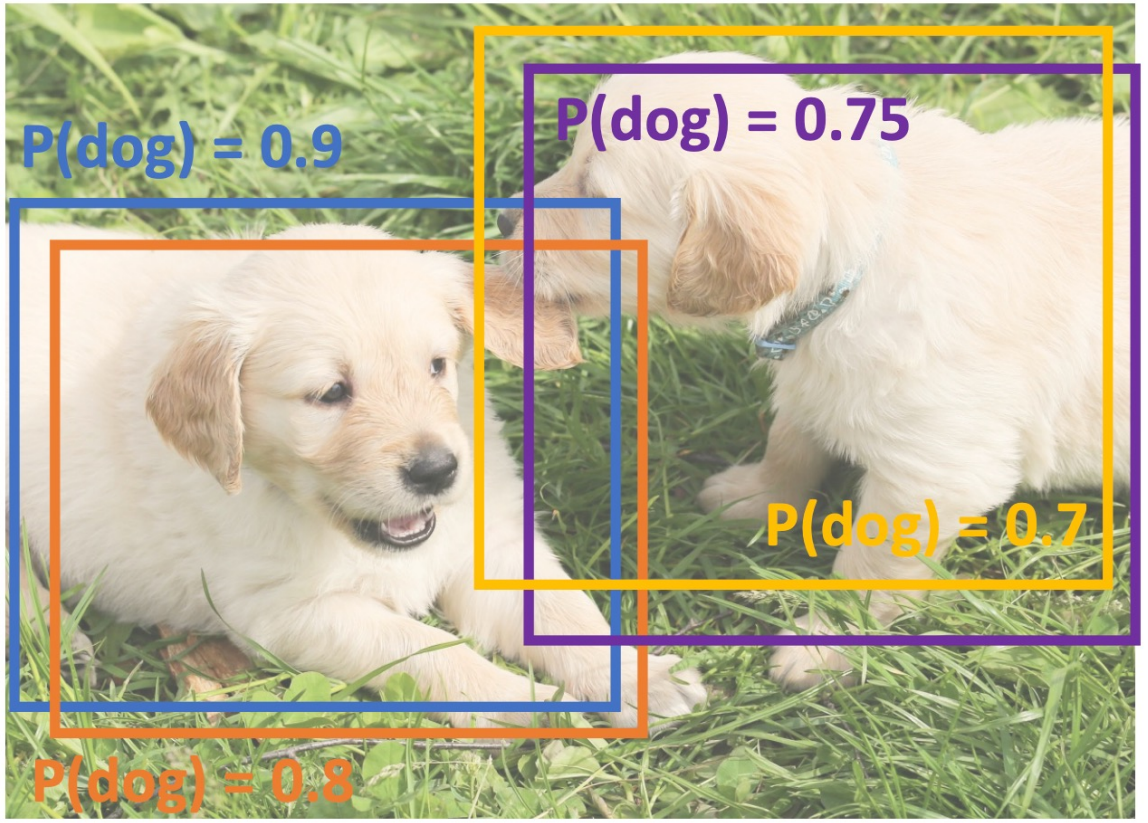
Intersection over Union (IoU)
(Also called “Jaccard similarity” or “Jaccard index”):

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

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

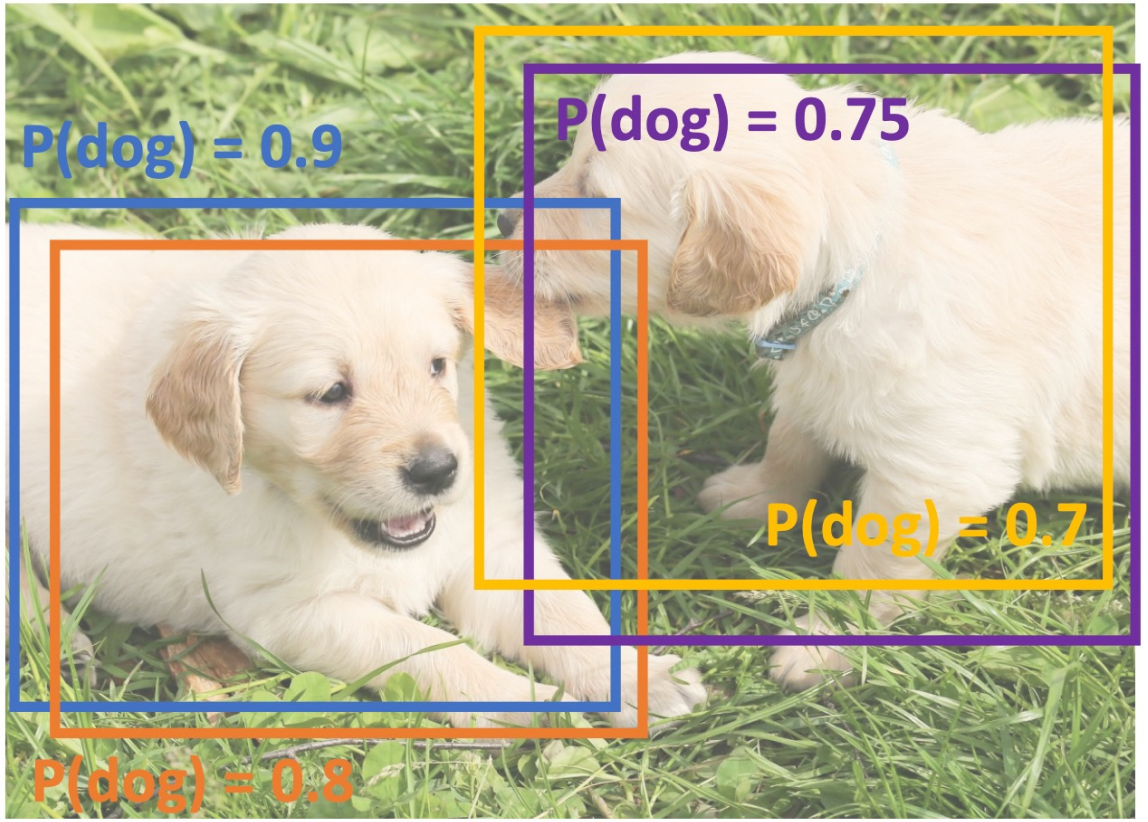


Non-Max Suppression (NMS)

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



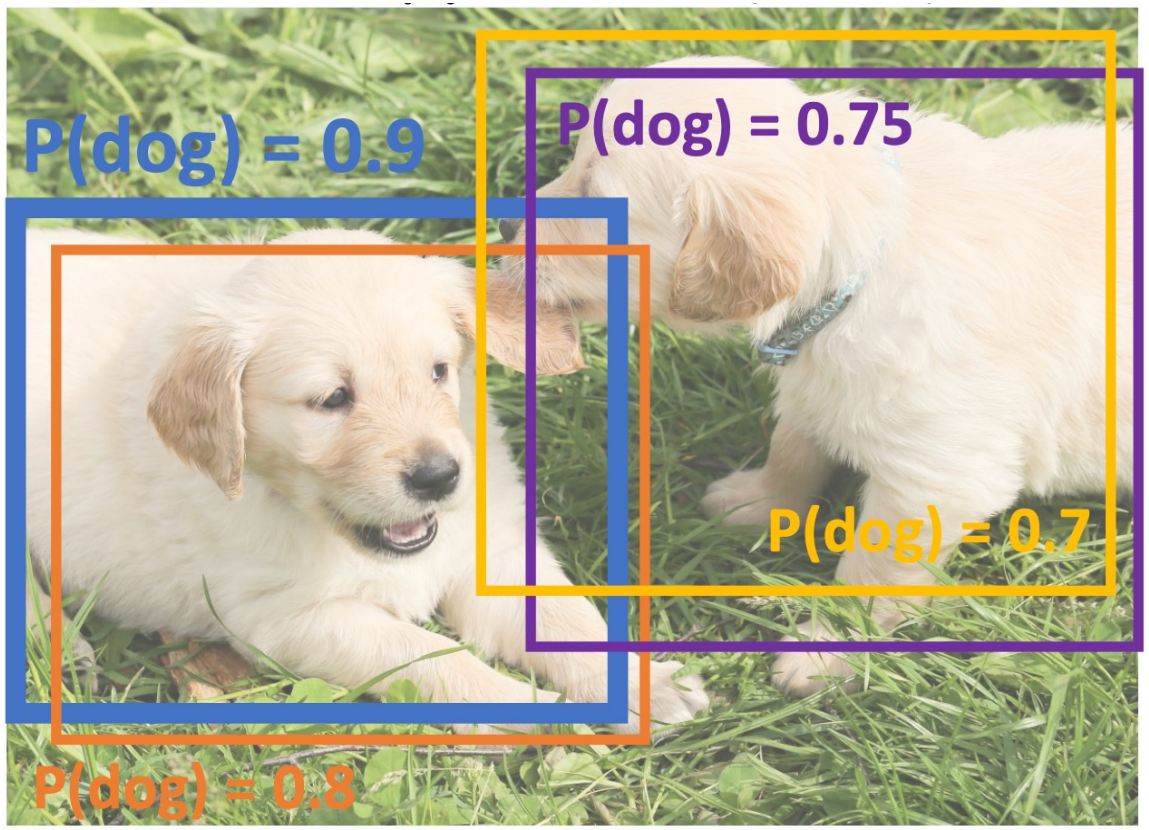
Non-Max Suppression (NMS)

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



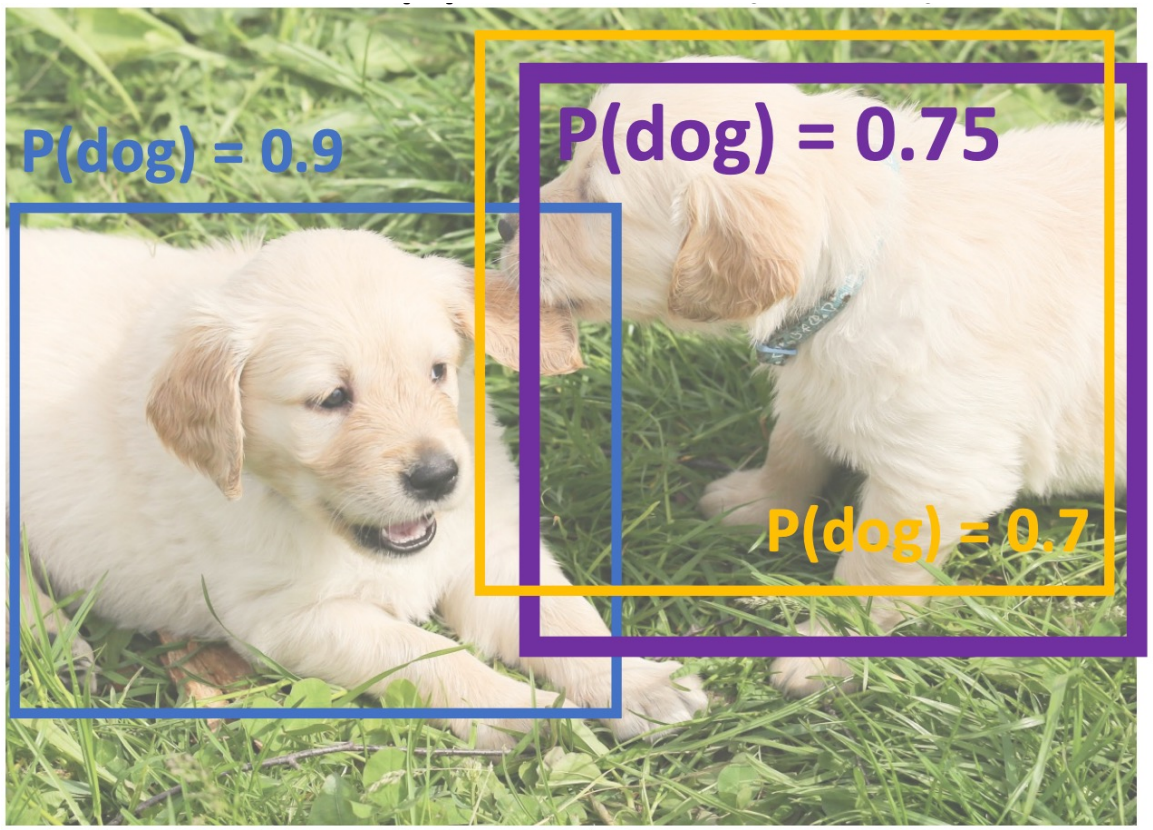
Non-Max Suppression (NMS)

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**

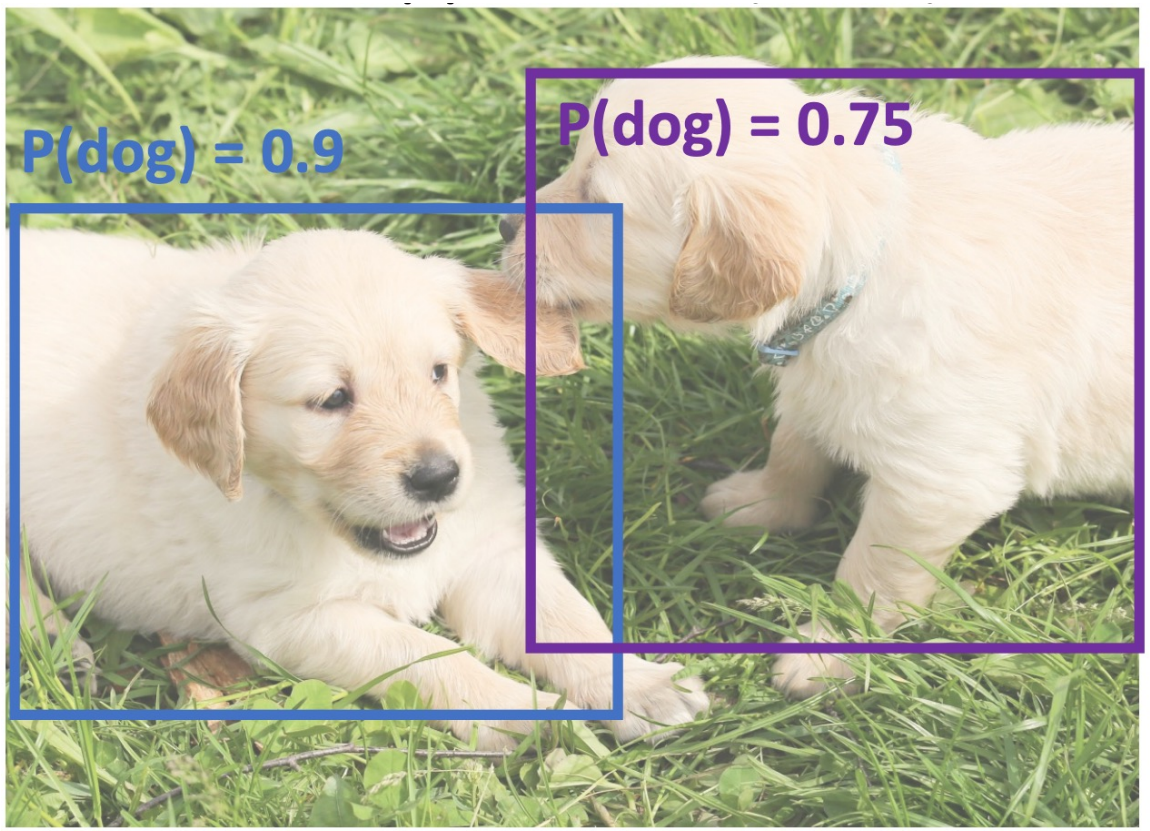


Non-Max Suppression (NMS)

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



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

Mean Average Precision (mAP)

Evaluating Object Detectors

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

Mean Average Precision (mAP)

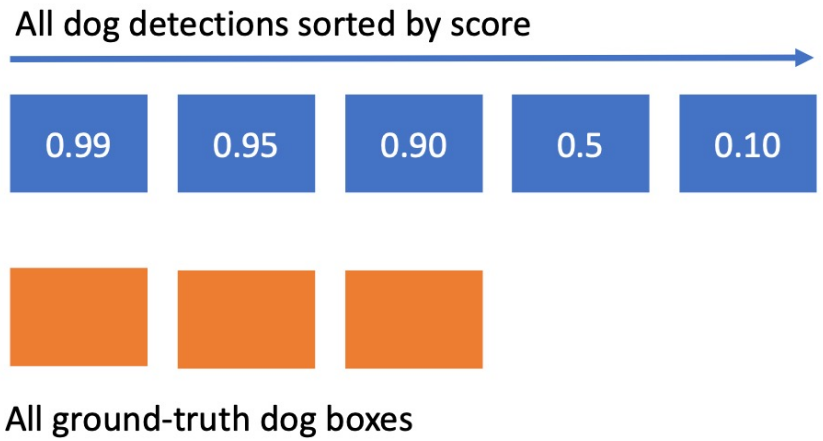
Evaluating Object Detectors

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

Mean Average Precision (mAP)

Evaluating Object Detectors

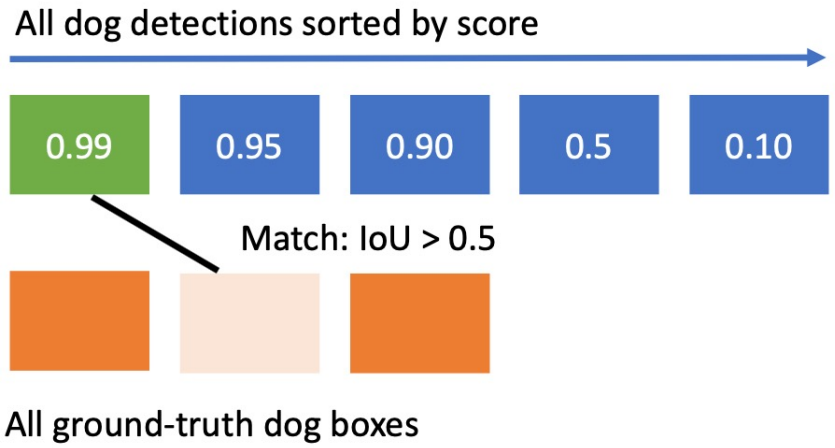
1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP)
   = area under Precision vs Recall Curve
   a. For each detection (highest score to lowest score)



Mean Average Precision (mAP)

Evaluating Object Detectors

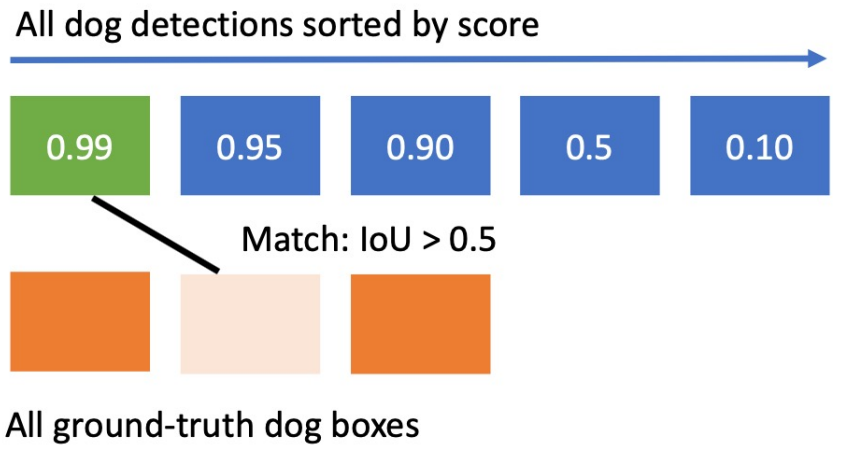
1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP)
   = area under Precision vs Recall Curve
   a. For each detection (highest score to lowest score)
      i. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
   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

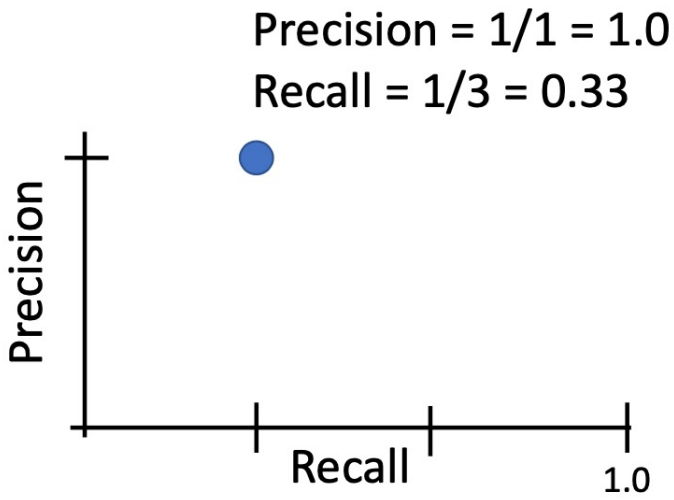


Mean Average Precision (mAP)

Evaluating Object Detectors

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

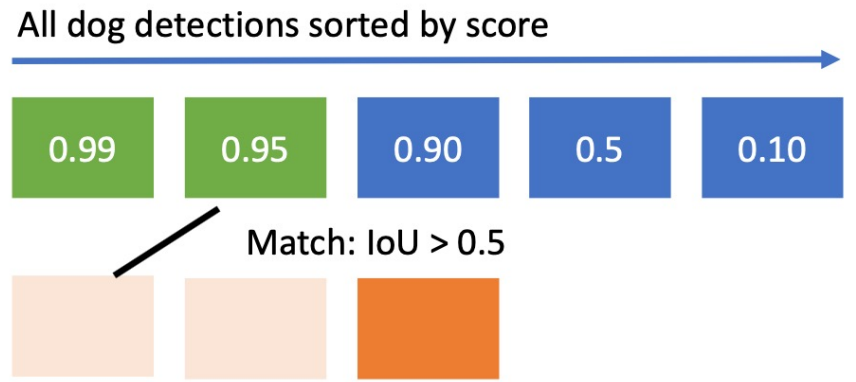


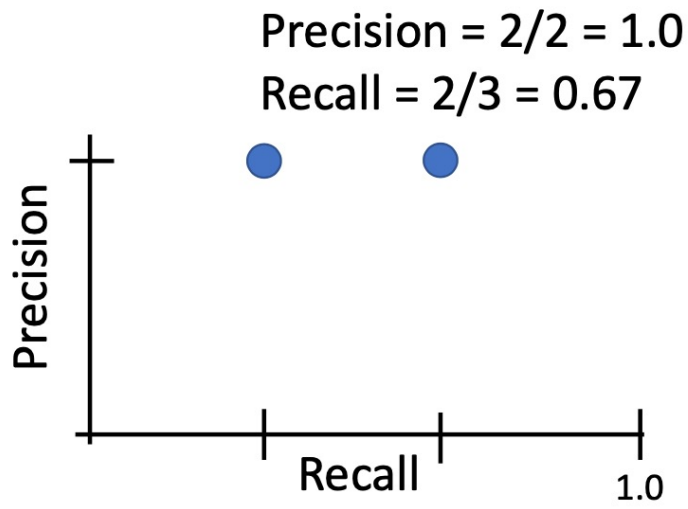


Mean Average Precision (mAP)

Evaluating Object Detectors

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

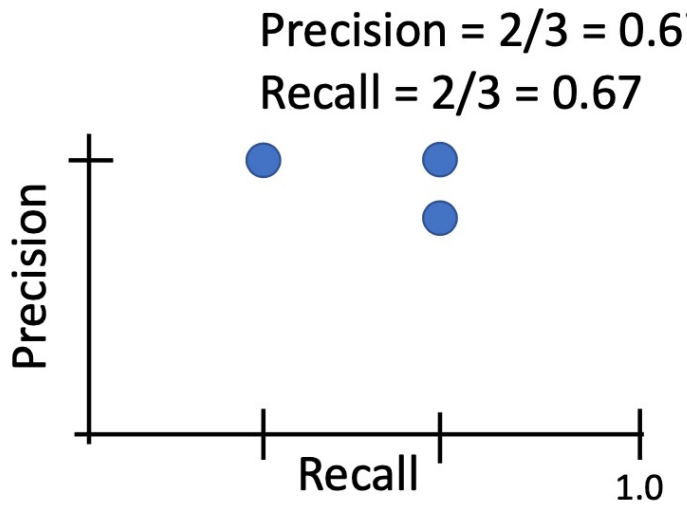




Mean Average Precision (mAP)

Evaluating Object Detectors

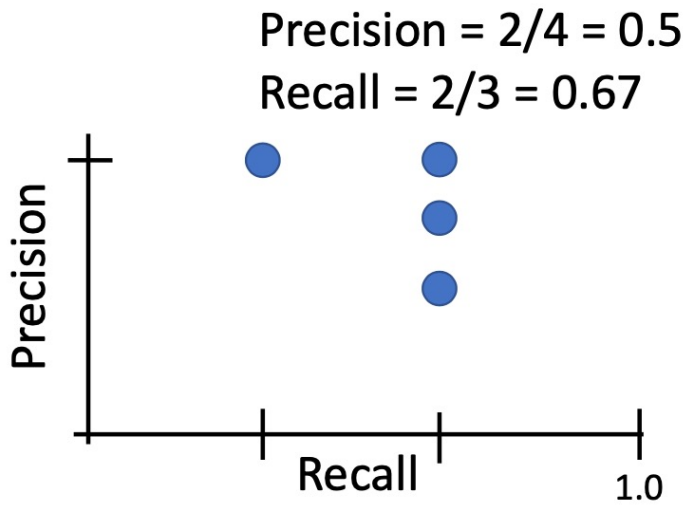
1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
   a. For each detection (highest score to lowest score)
      i. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
      ii.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



Mean Average Precision (mAP)

Evaluating Object Detectors

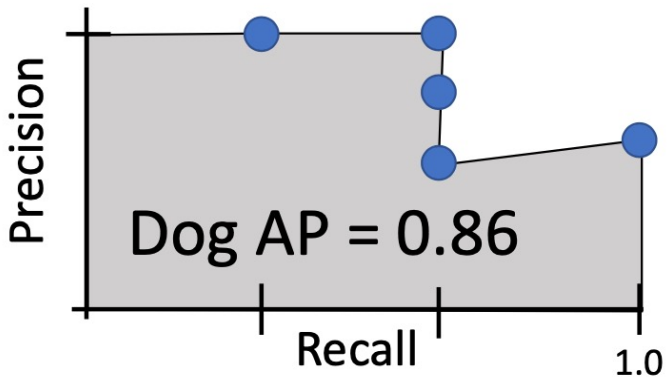
1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
   a. For each detection (highest score to lowest score)
      i. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
      ii.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



Mean Average Precision (mAP)

Evaluating Object Detectors

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

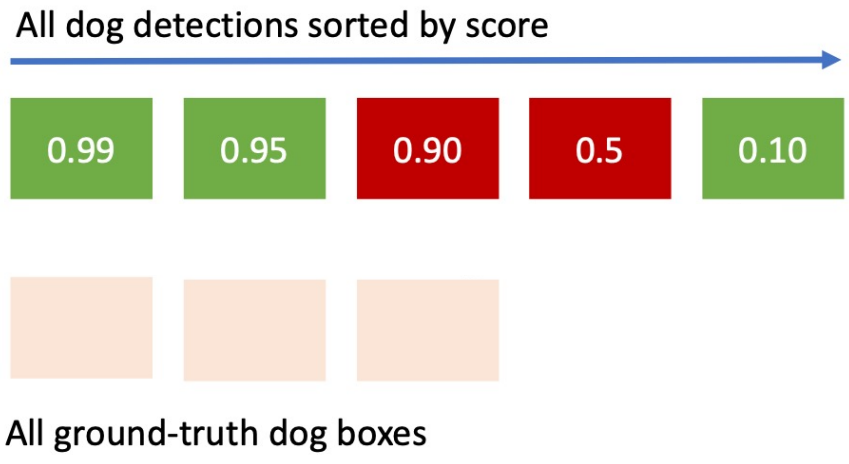


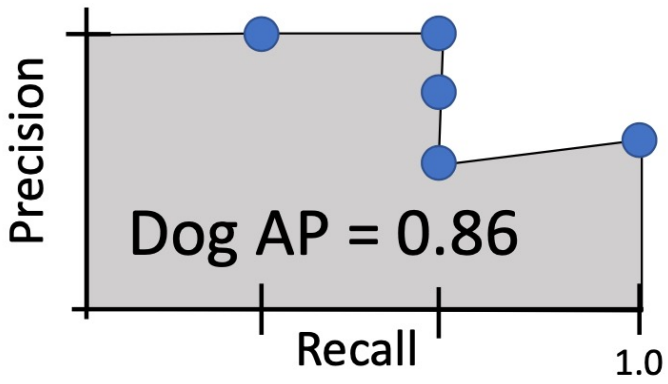
Mean Average Precision (mAP)

Evaluating Object Detectors

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





Mean Average Precision (mAP)

Evaluating Object Detectors

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

Mean Average Precision (mAP)

Evaluating Object Detectors

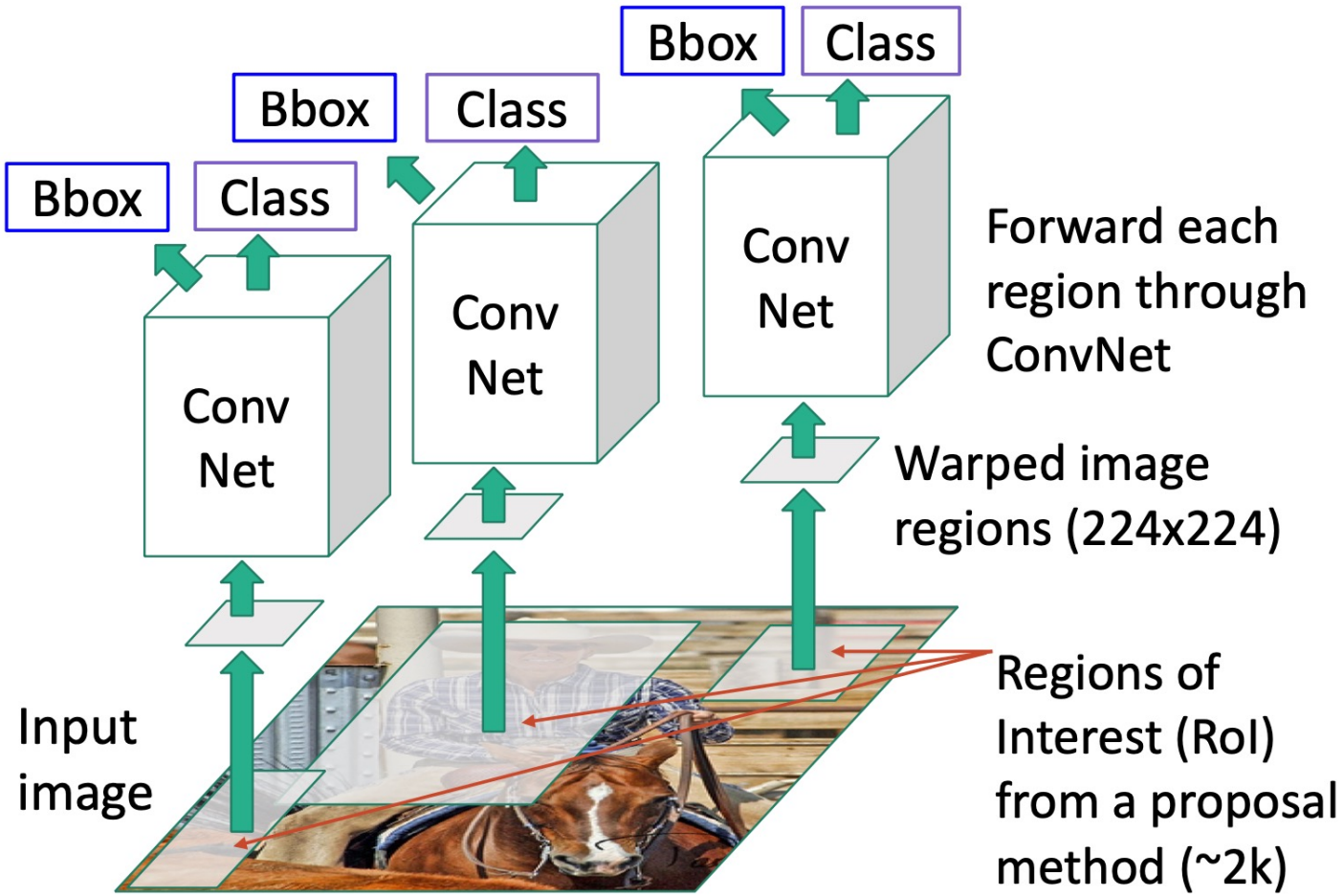
1. Run object detector on all test images (with NMS)
2. For each category, compute Average Precision (AP) = area under Precision vs Recall Curve
   a. For each detection (highest score to lowest score)
      i. If it matches some GT box with IoU > 0.5, mark it as positive and eliminate the GT
      ii.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

https://cocodataset.org/

Region-based CNN



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

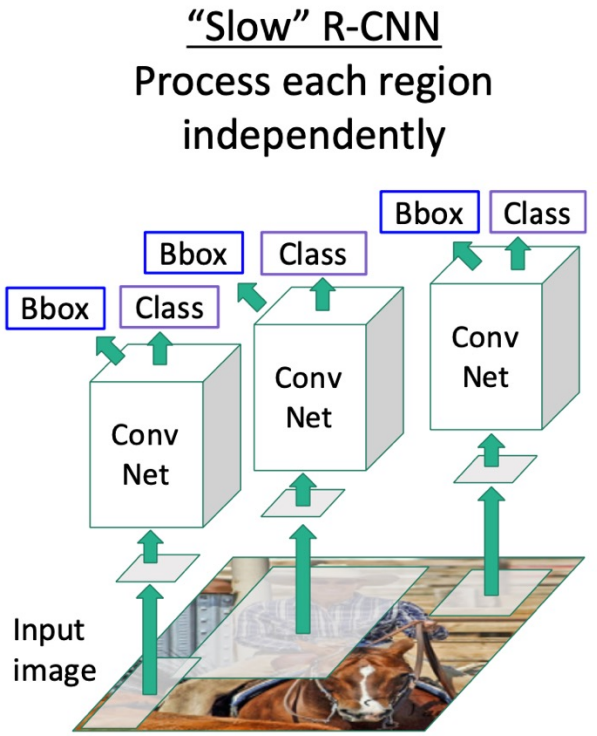
Outline

- Task definition
- Region-based CNN and basic concepts
  - Intersection over Union (IoU)
  - Non-Max Suppression (NMS)
  - Mean Average Precision (mAP)
- Fast R-CNN
- Faster R-CNN

- *Tutorial 2*: Faster-RCNN

Fast R-CNN

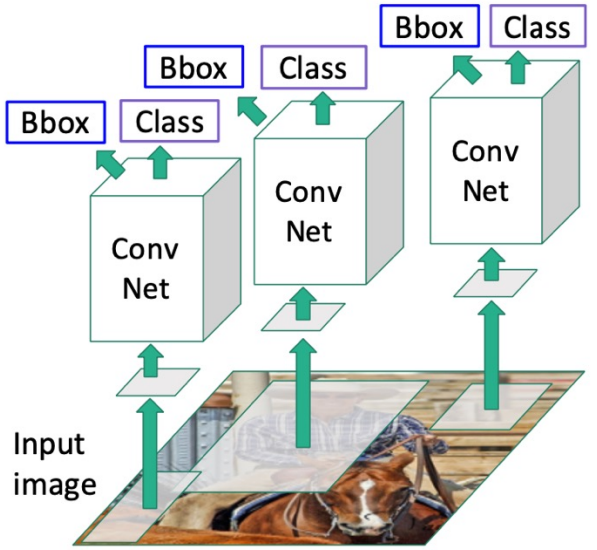
Fast R-CNN



Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015

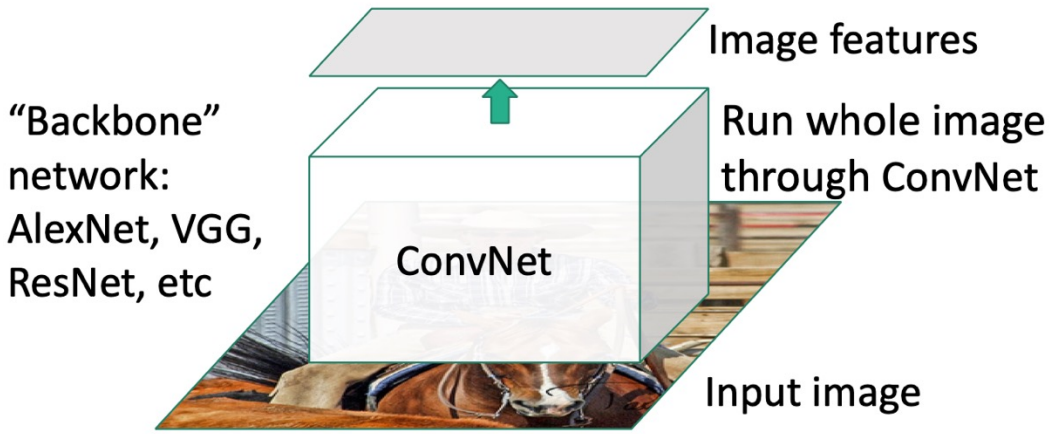
Fast R-CNN

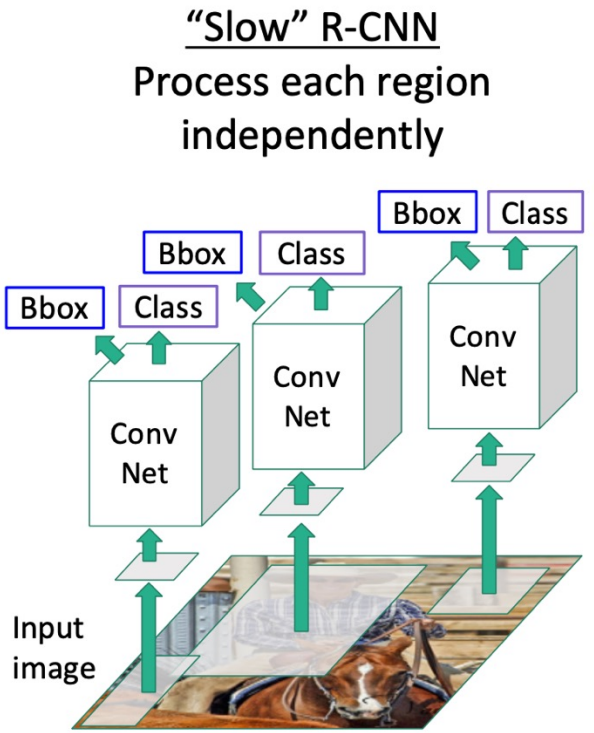
Input image



Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015

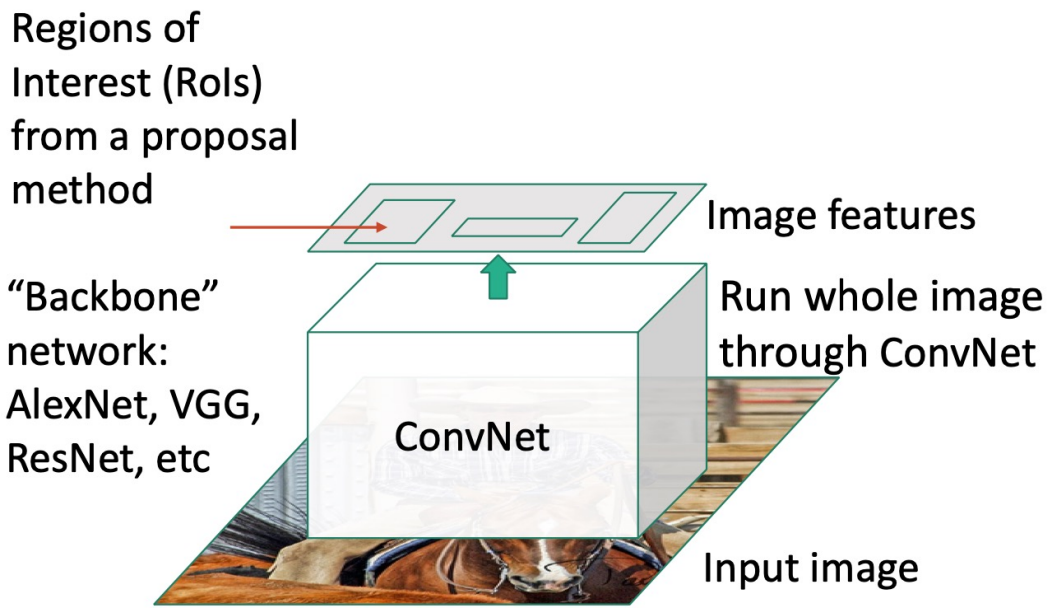
Fast R-CNN

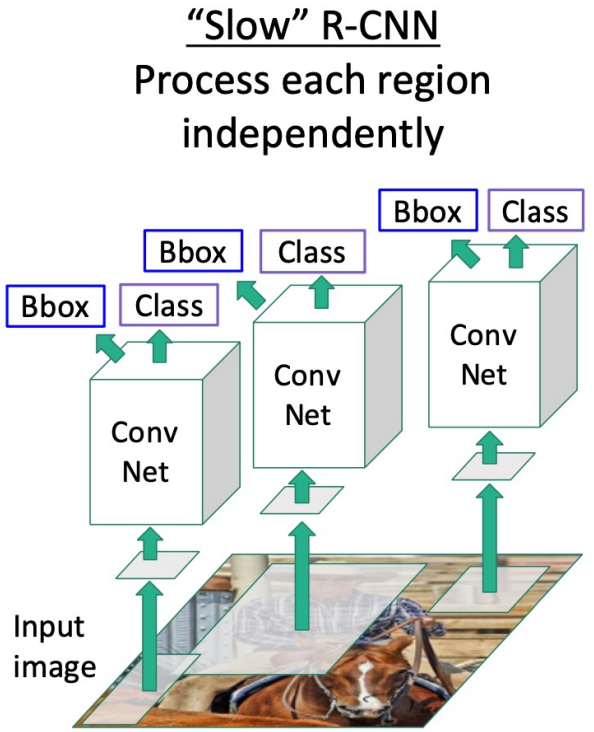




Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015

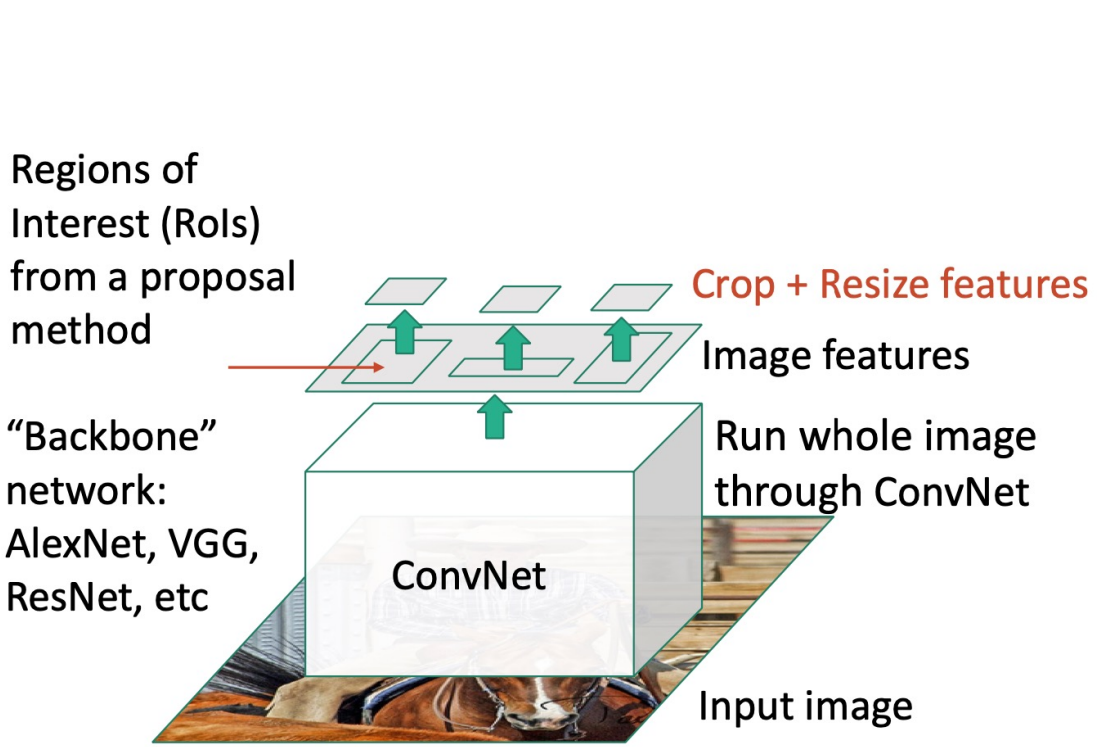
Fast R-CNN

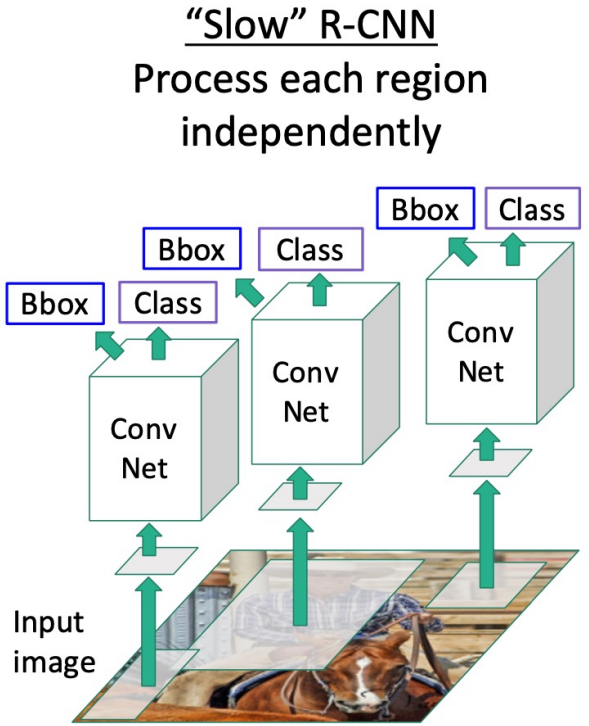




Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015

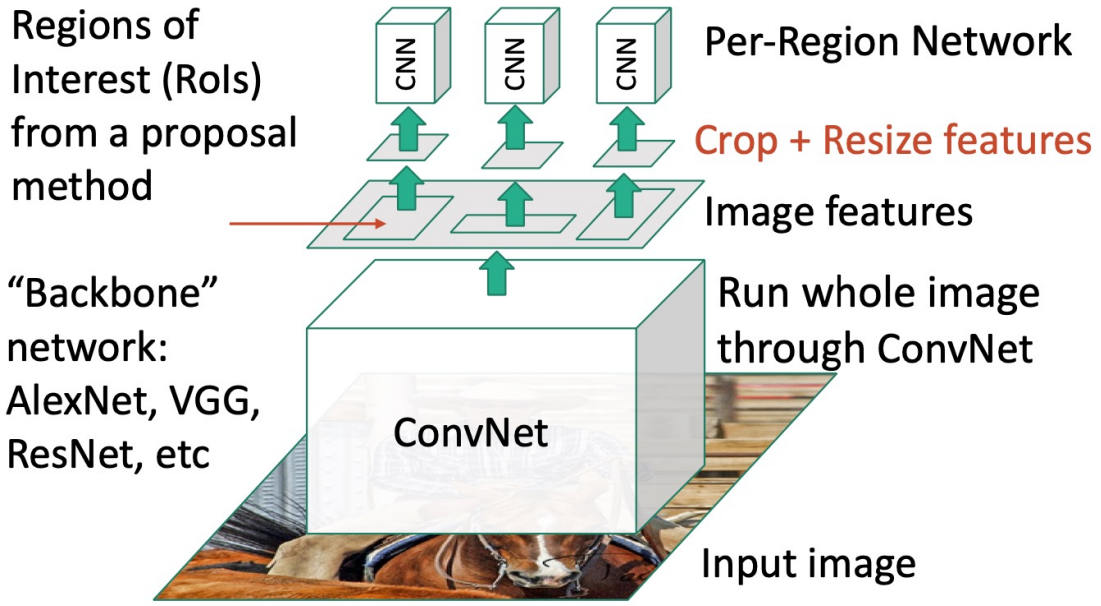
Fast R-CNN

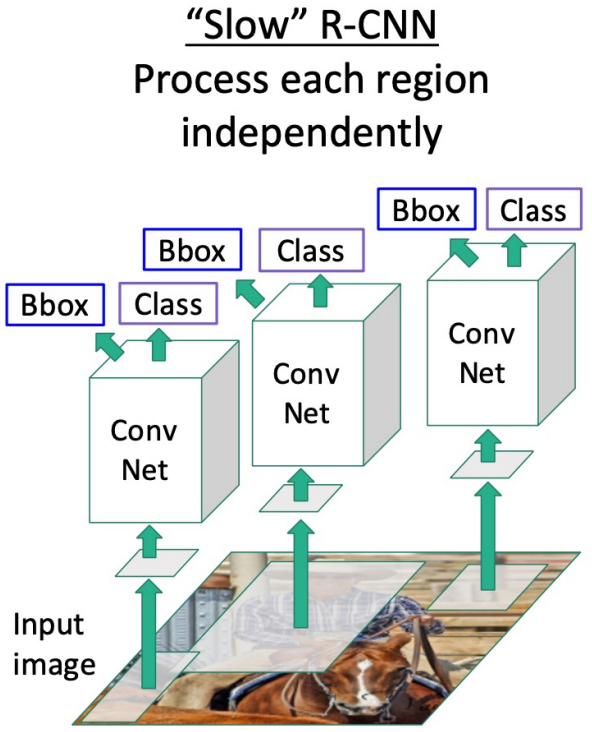




Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015

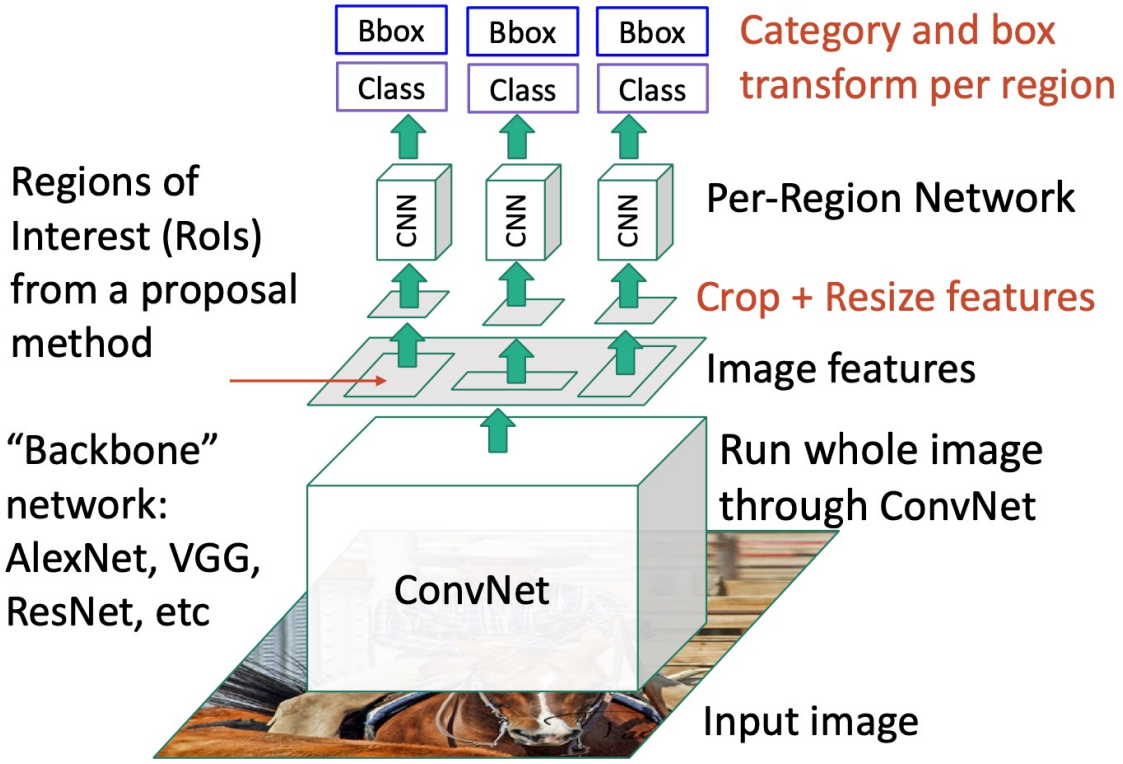
Fast R-CNN

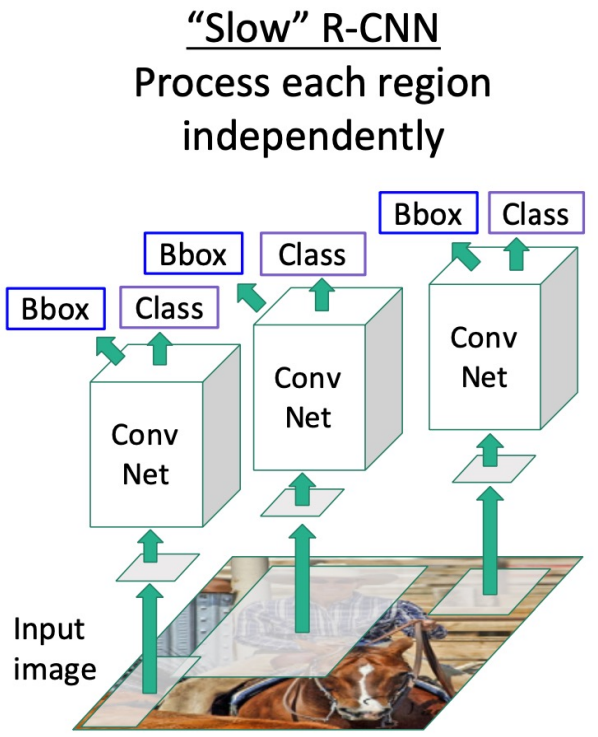




Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015

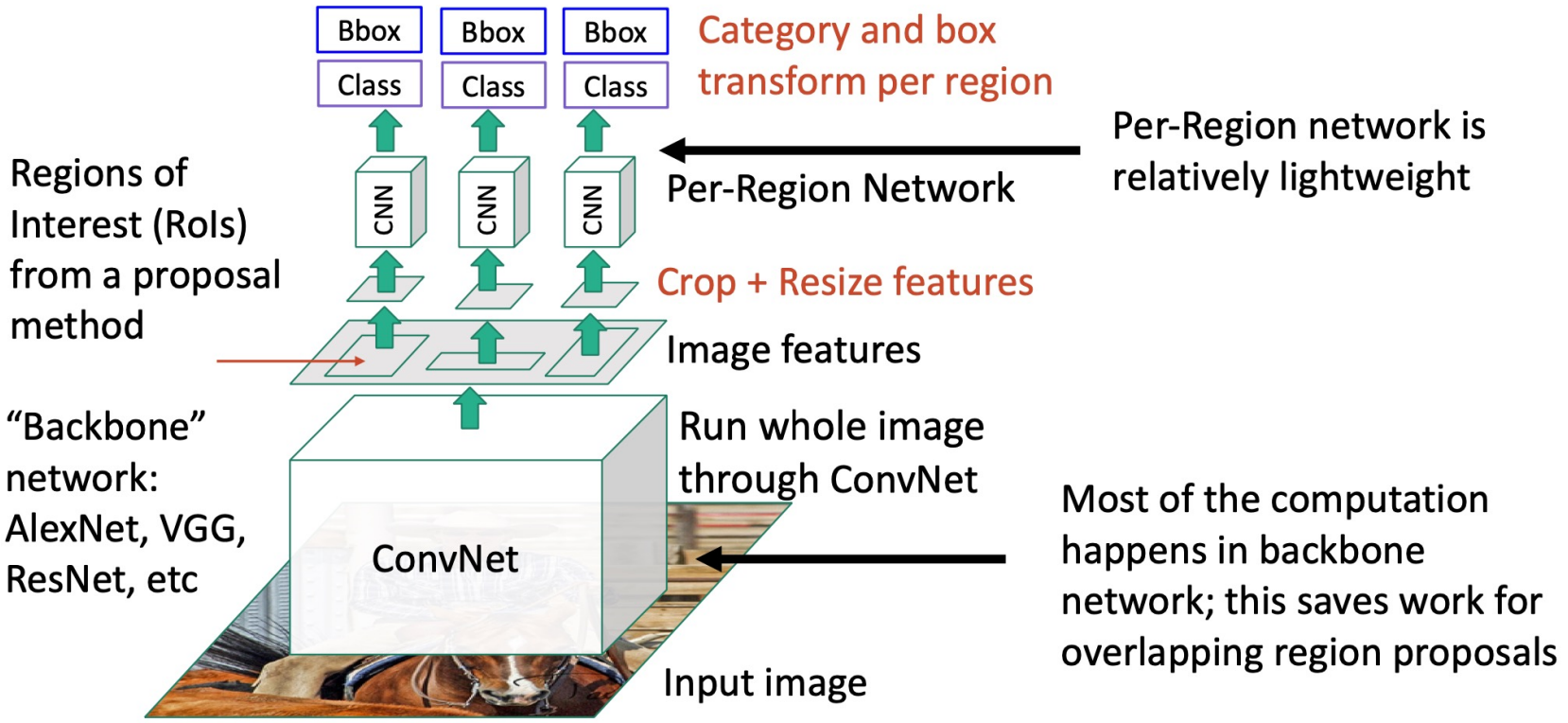
Fast R-CNN





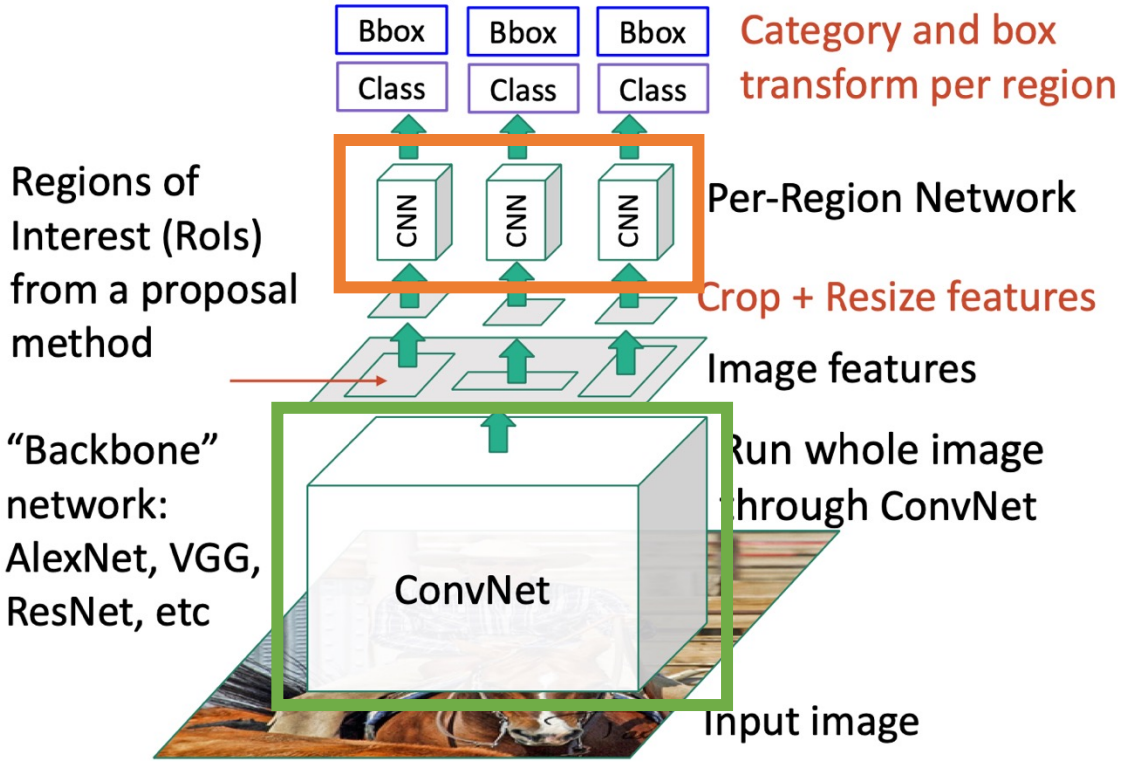
Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015

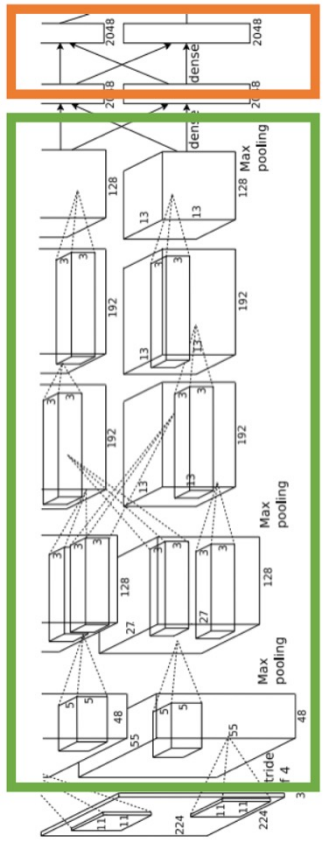
Fast R-CNN



Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015

Fast R-CNN

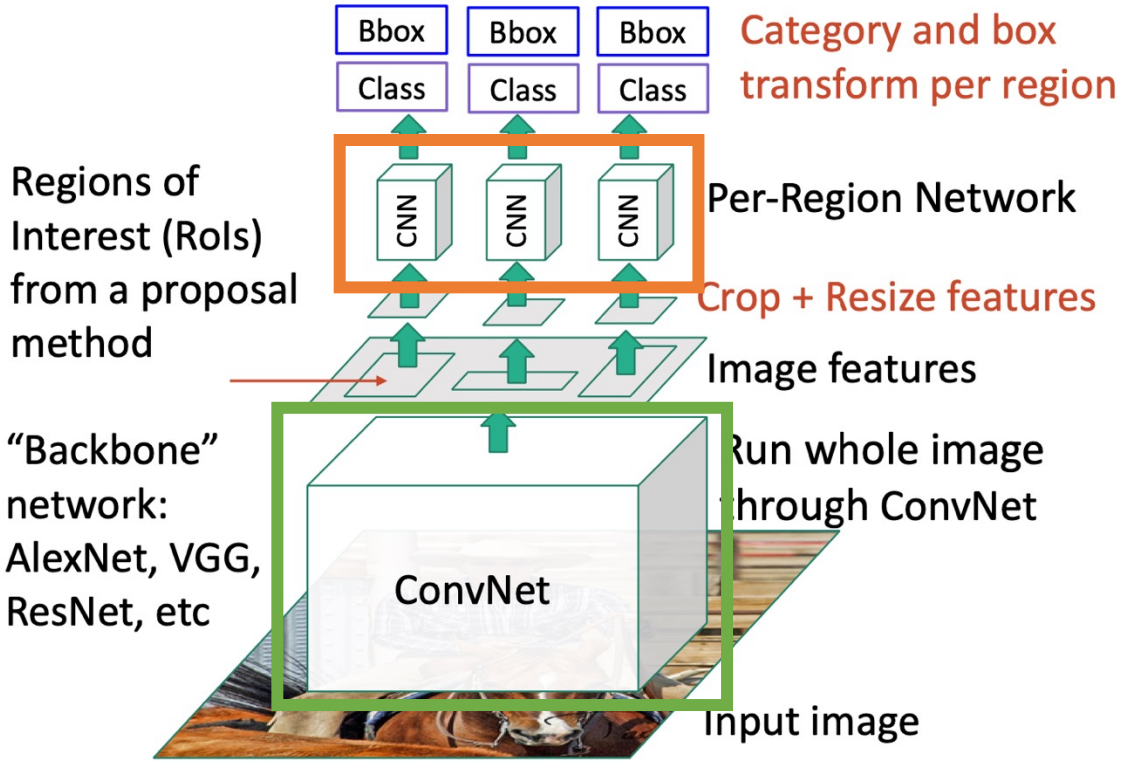


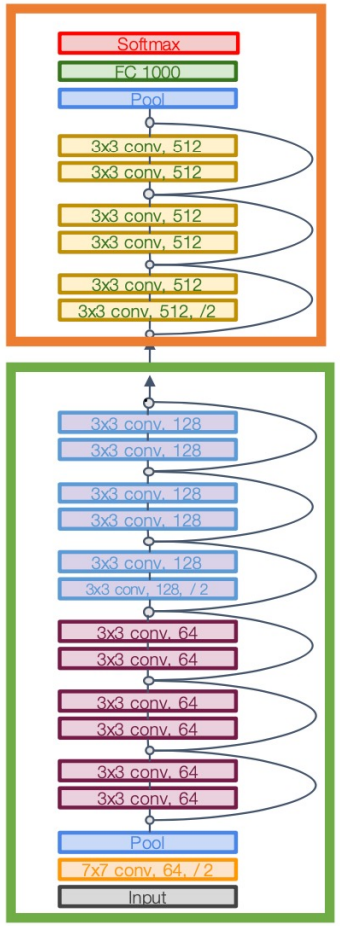


Example: When using AlexNet for detection, five conv layers are used for backbone and two FC layers are used for per-region network

Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015

Fast R-CNN

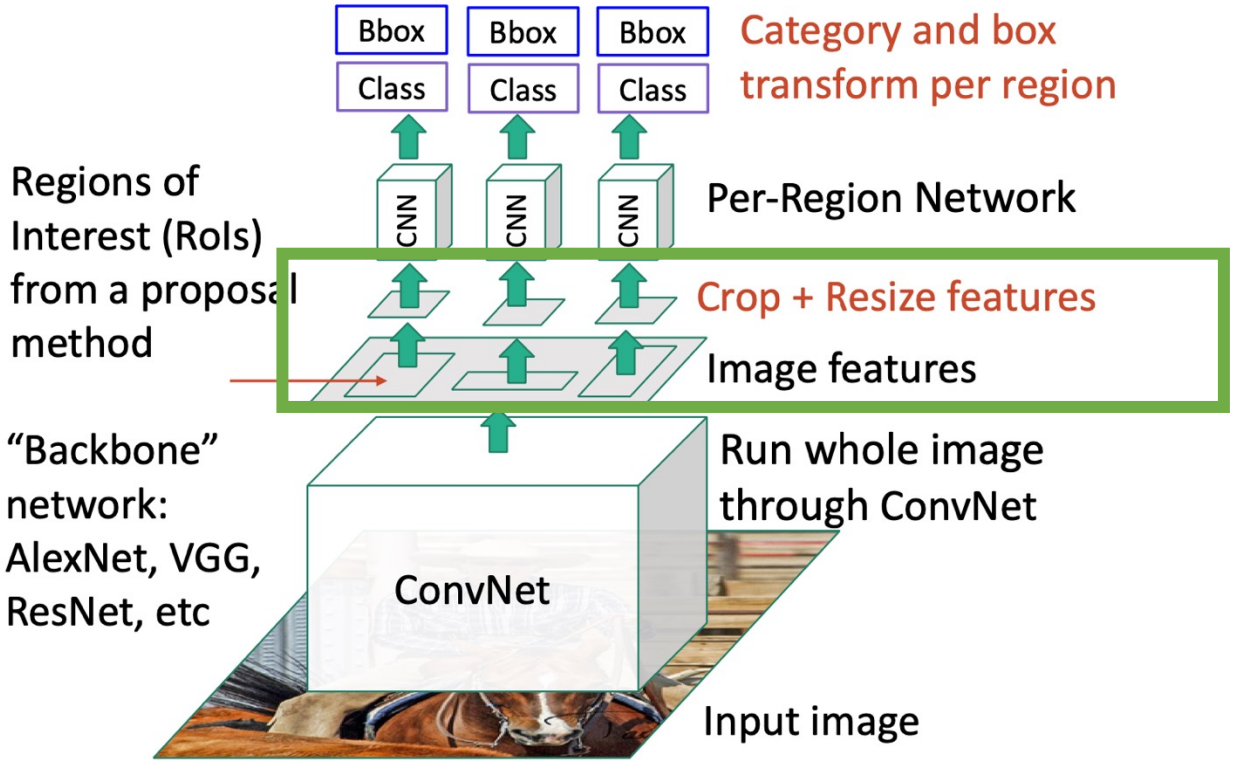




Example:
For ResNet, last stage is used as per-region network; the rest of the network is used as backbone

Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015

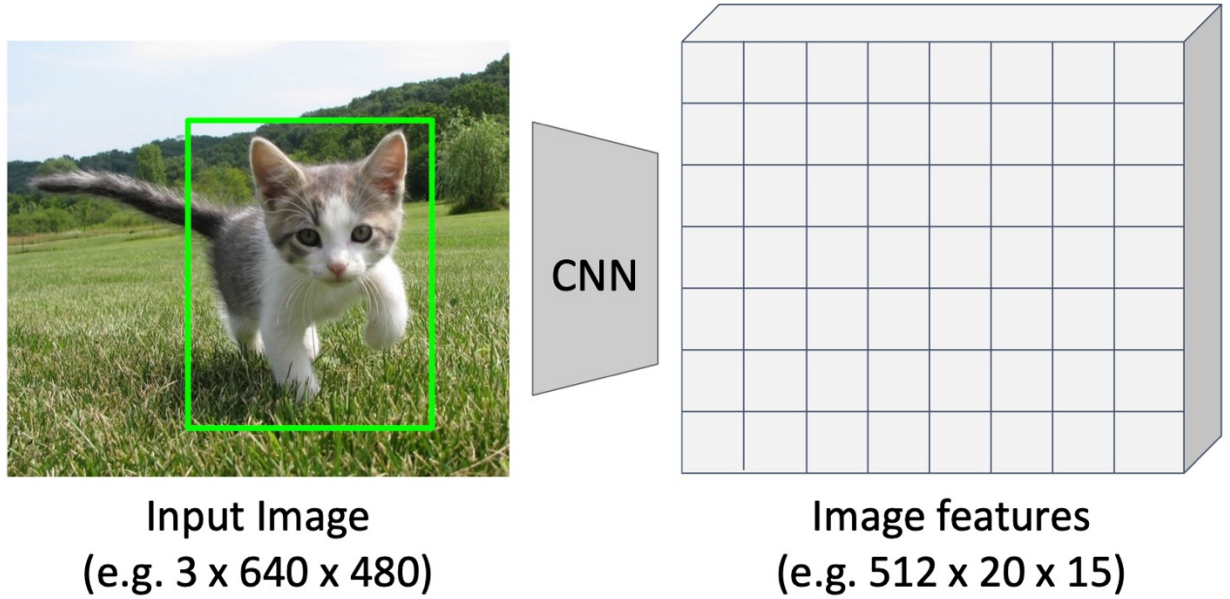
Fast R-CNN



How to crop features?

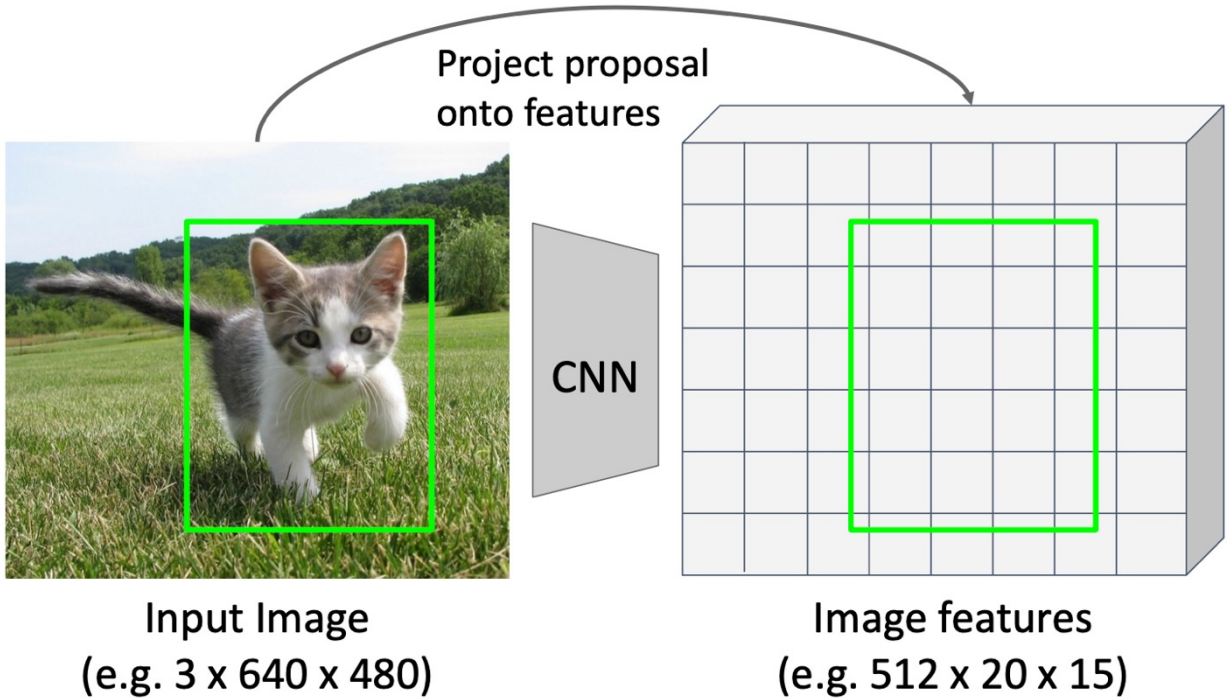
Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015

Cropping features – RoI Pool



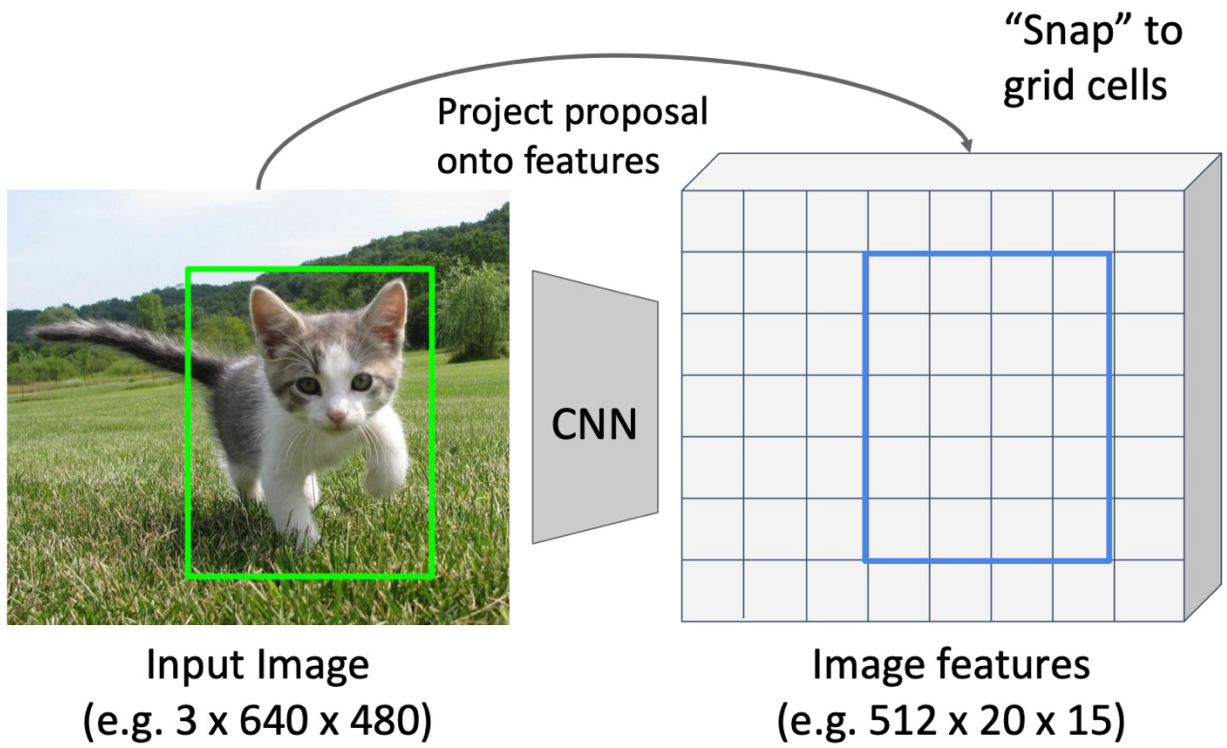
Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015

Cropping features – RoI Pool



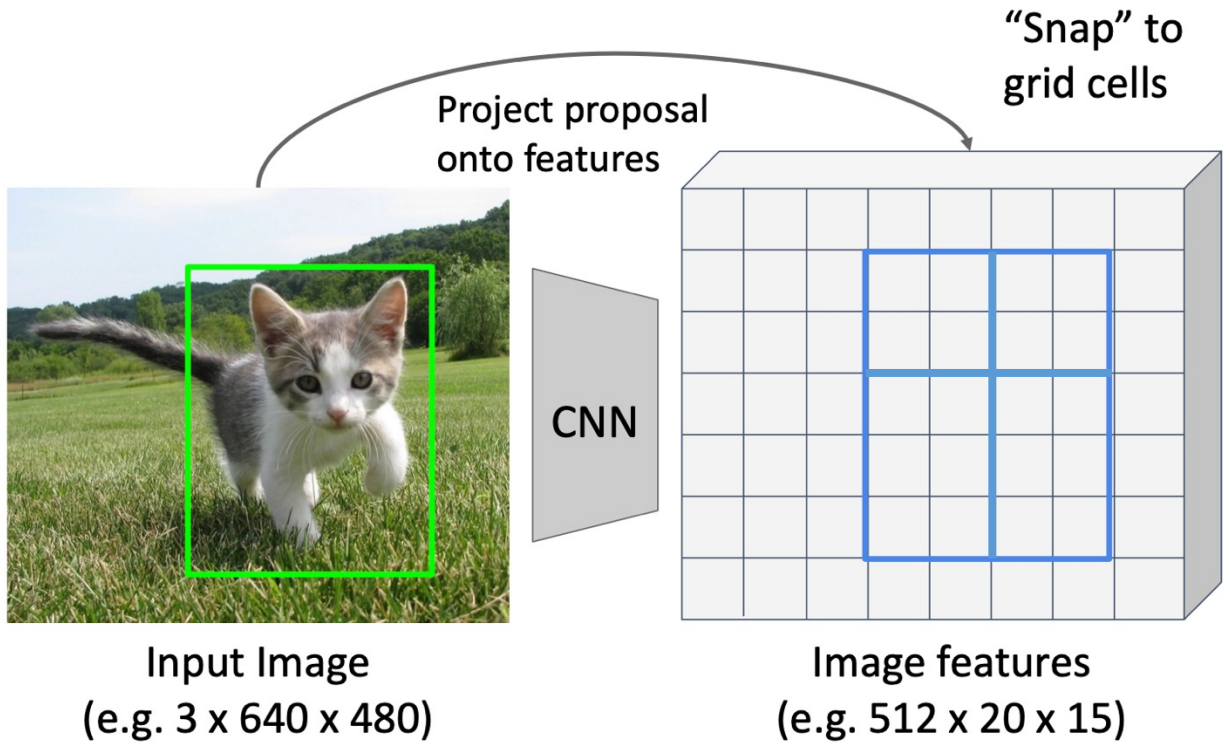
Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015

Cropping features – RoI Pool



Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015

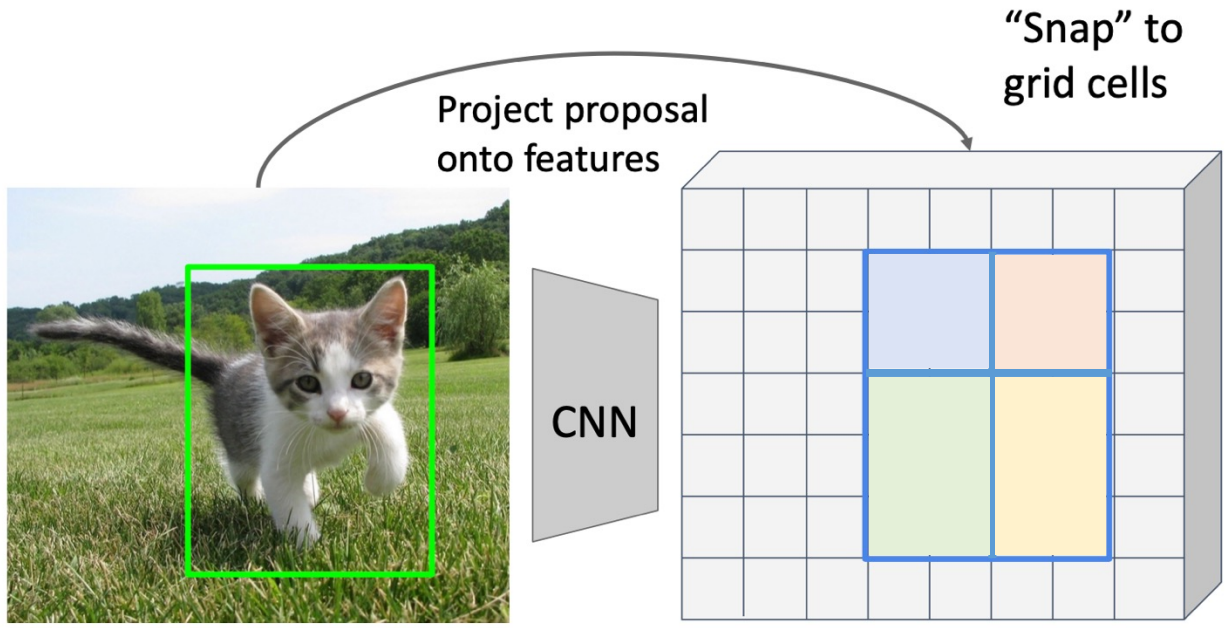
Cropping features – RoI Pool



Divide into 2x2 grid of (roughly) equal subregions

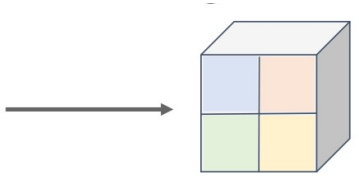
Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015

Cropping features – RoI Pool



Divide into 2x2 grid of (roughly) equal subregions

Max-pool within each subregion

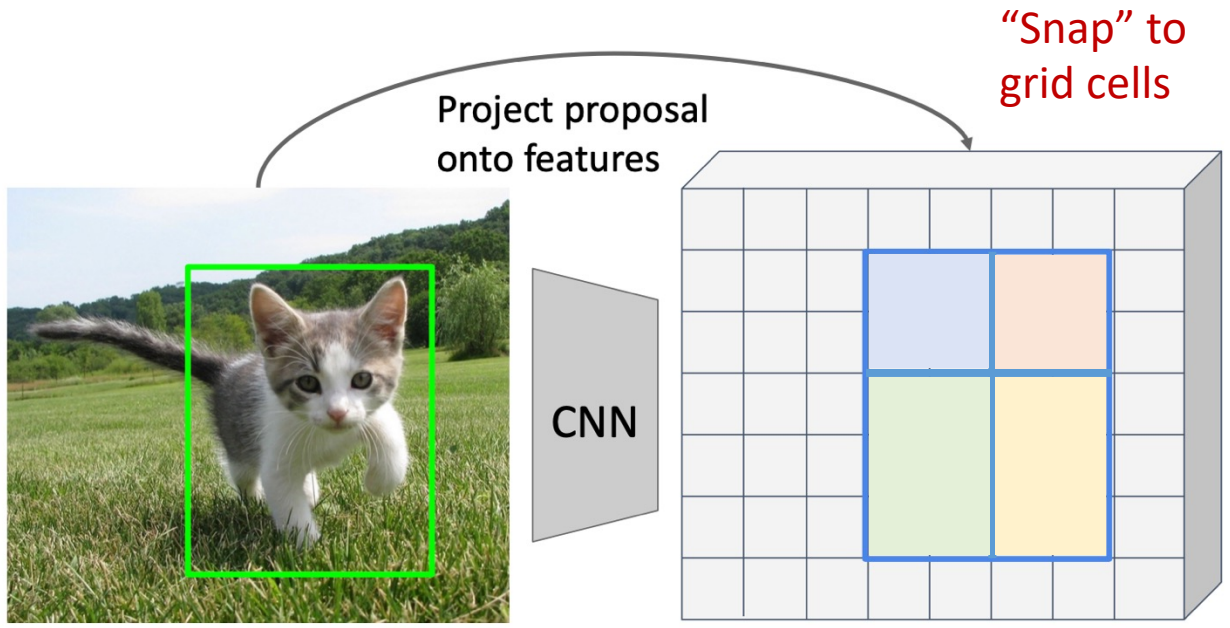


Region features
(here 512 x 2 x 2;
In practice e.g 512 x 7 x 7)

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

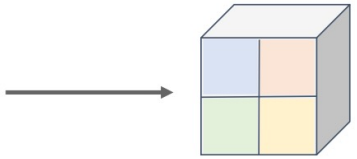
Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015

Cropping features – RoI Pool



Divide into 2x2 grid of (roughly) equal subregions

Max-pool within each subregion



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

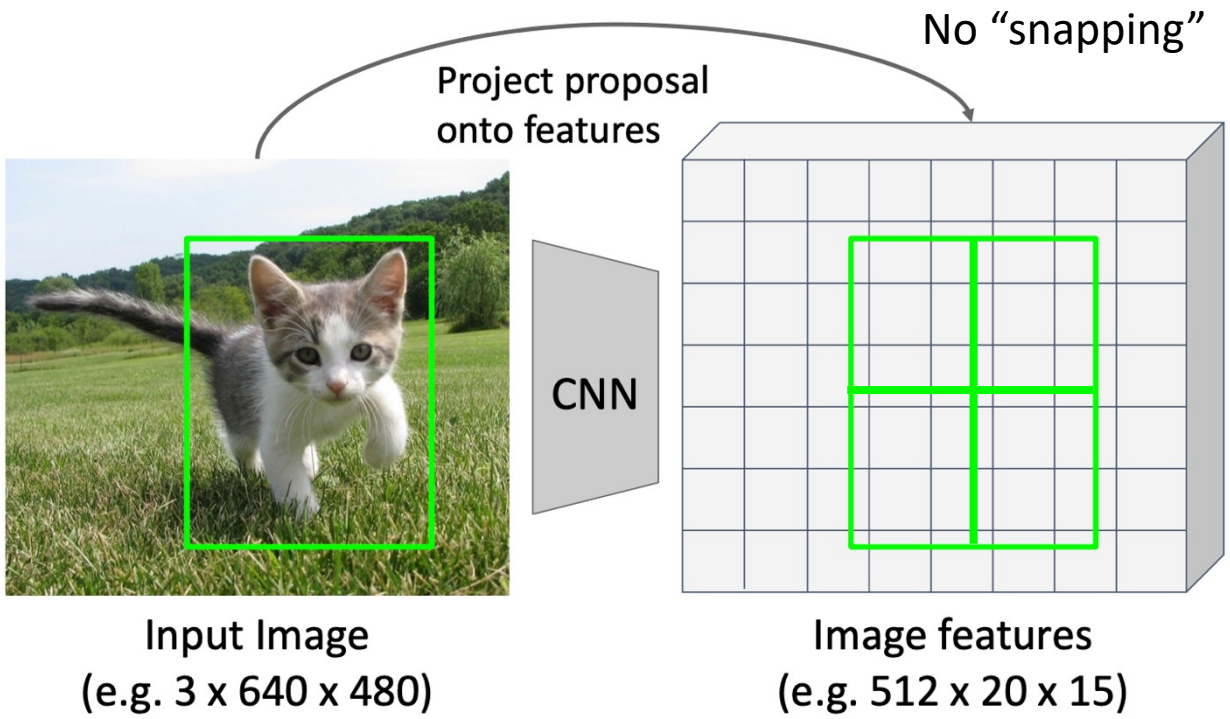
Region features always the same size even if input regions have different sizes!

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

RoI pooling first quantizes a floating-number RoI to the discrete granularity of the feature map. These quantizations introduce misalignments between the RoI and the extracted features. While this may not impact classification (which is robust to small translations), it has a large negative effect on predicting pixel-accurate masks.

Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015

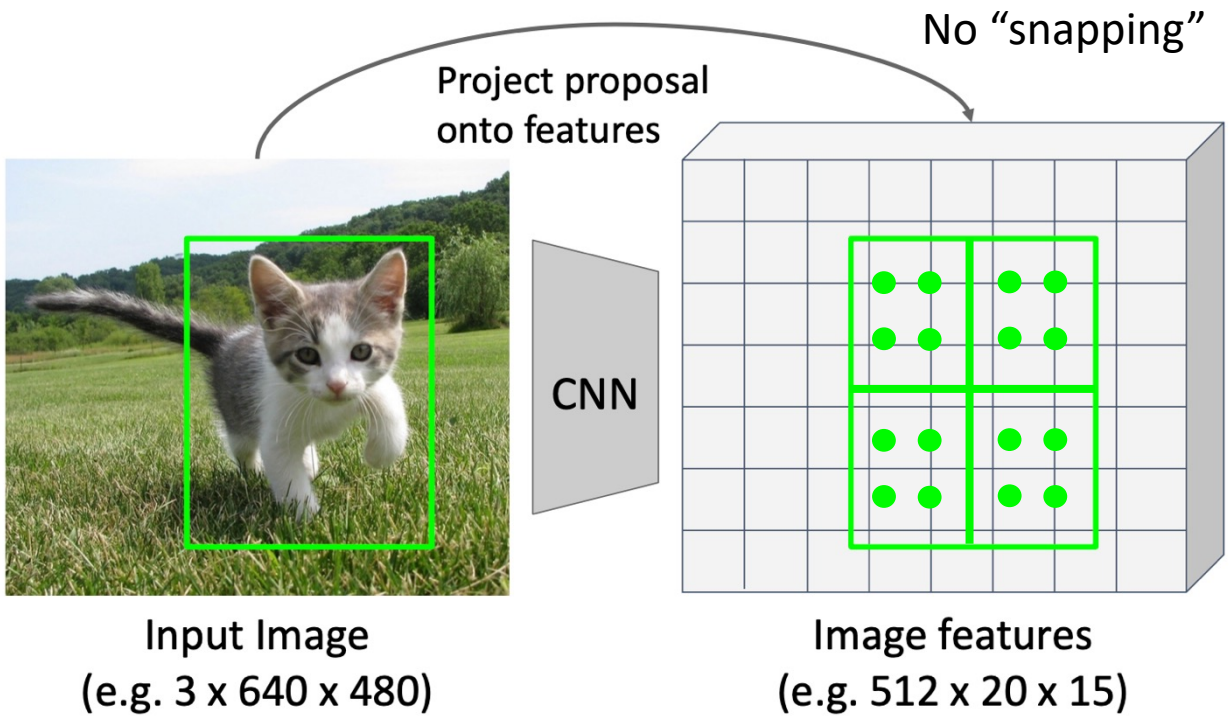
Cropping features – RoI Align



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

Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015

Cropping features – RoI Align



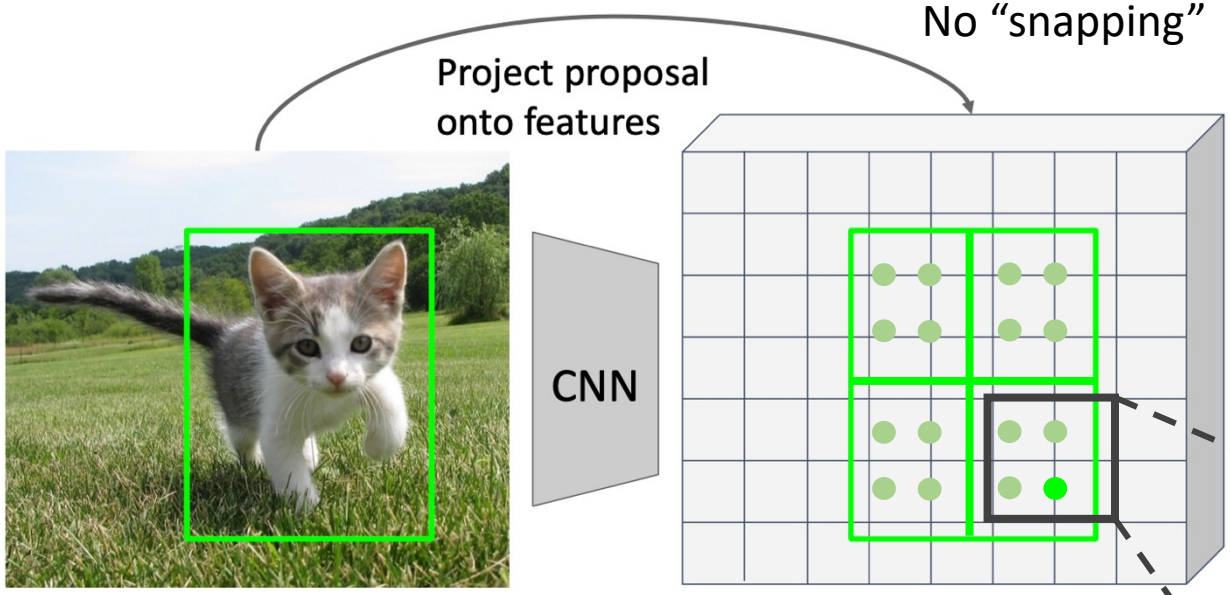
Divide into equal-sized subregions (may not be aligned to grid!)

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

We note that the results are not sensitive to the exact sampling locations, or how many points are sampled, as long as no quantization is performed.

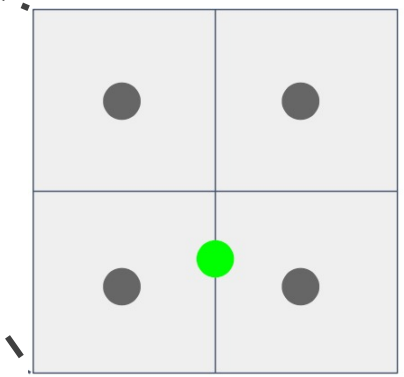
Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015

Cropping features – RoI Align



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

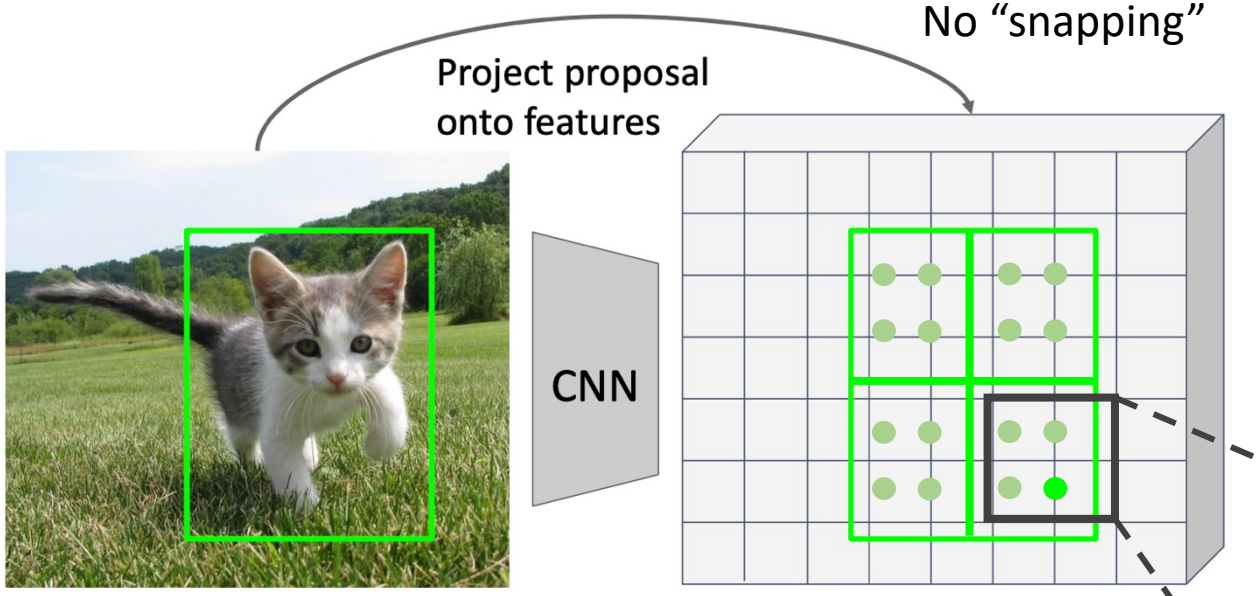
Sample features at regularly-spaced points in each subregion using **bilinear interpolation**



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

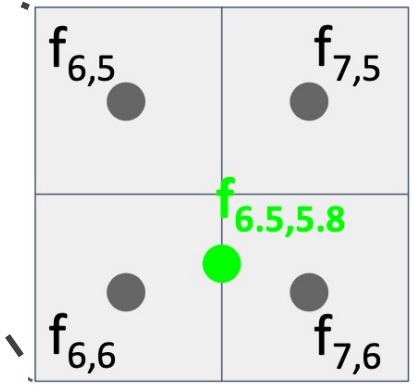
Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015

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**

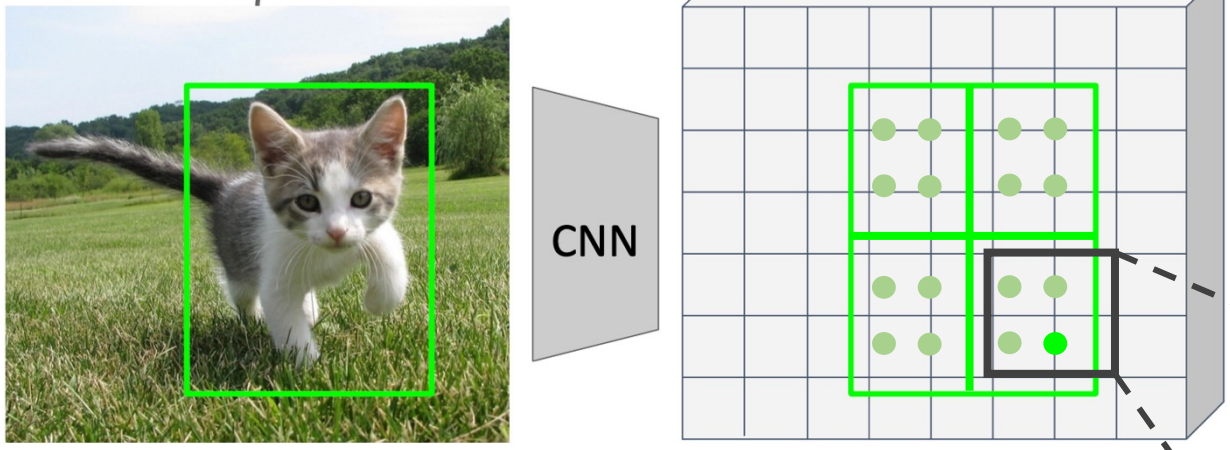


Feature f_{xy} for point (x,y) is a linear combination of features at its four neighboring grid cells

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

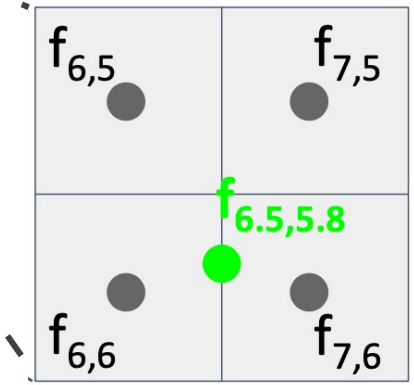
Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015

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**



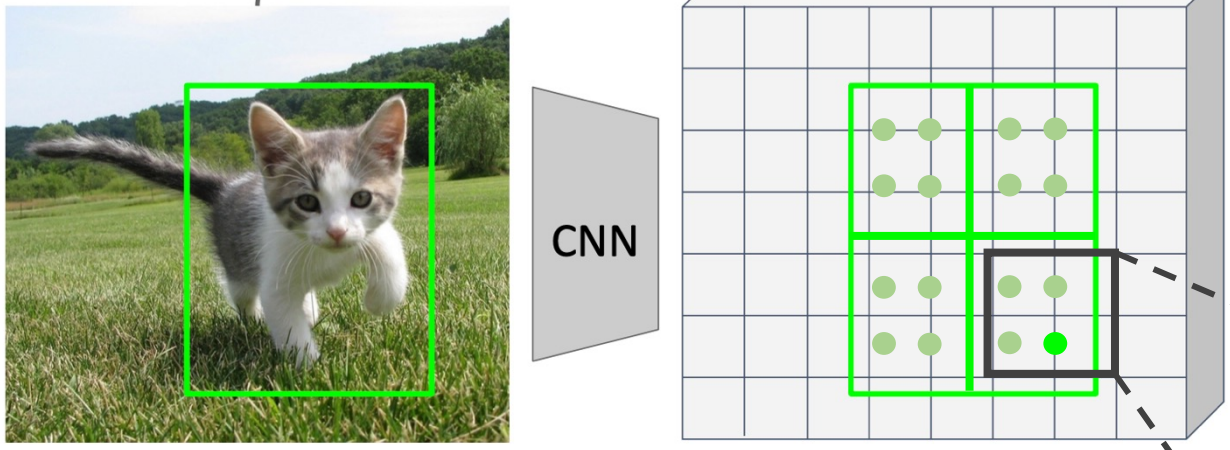
Feature f_{xy} for point (x,y) is a linear combination of features at its four neighboring grid cells

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

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

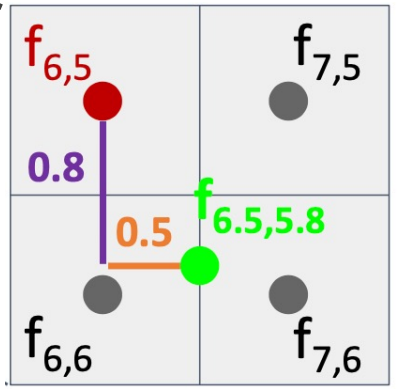
Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015

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**



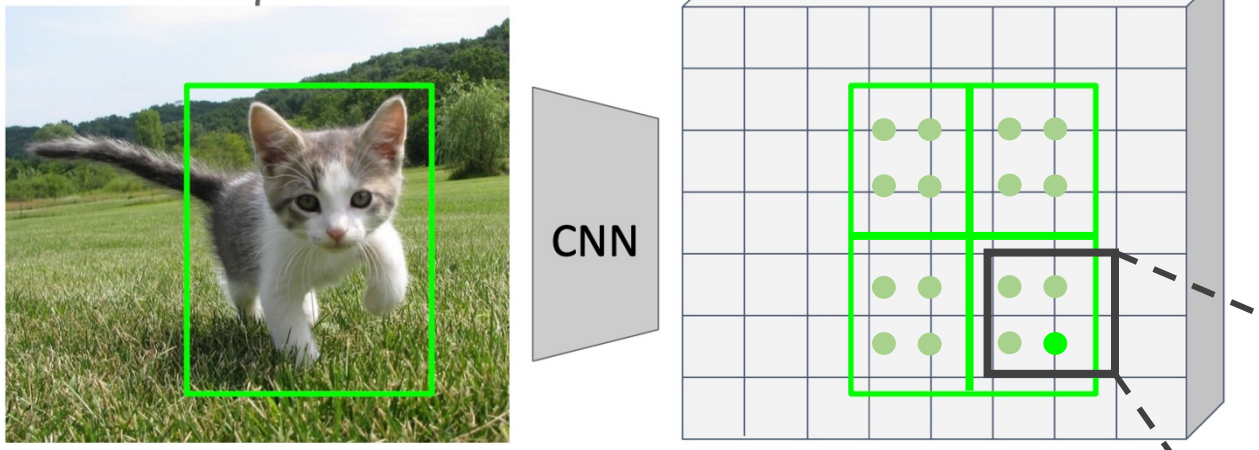
Feature f_{xy} for point (x,y) is a linear combination of features at its four neighboring grid cells

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

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

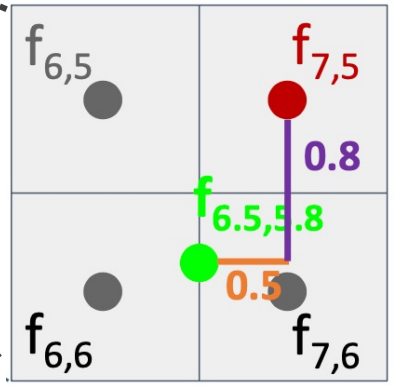
Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015

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**



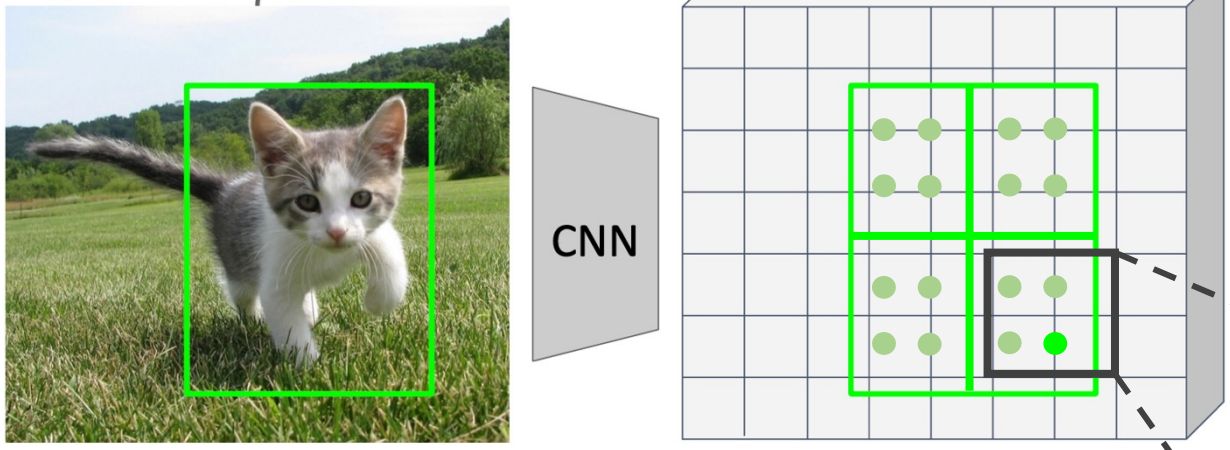
Feature f_{xy} for point (x,y) is a linear combination of features at its four neighboring grid cells

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

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

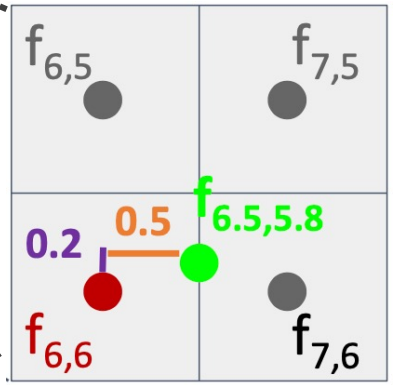
Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015

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**



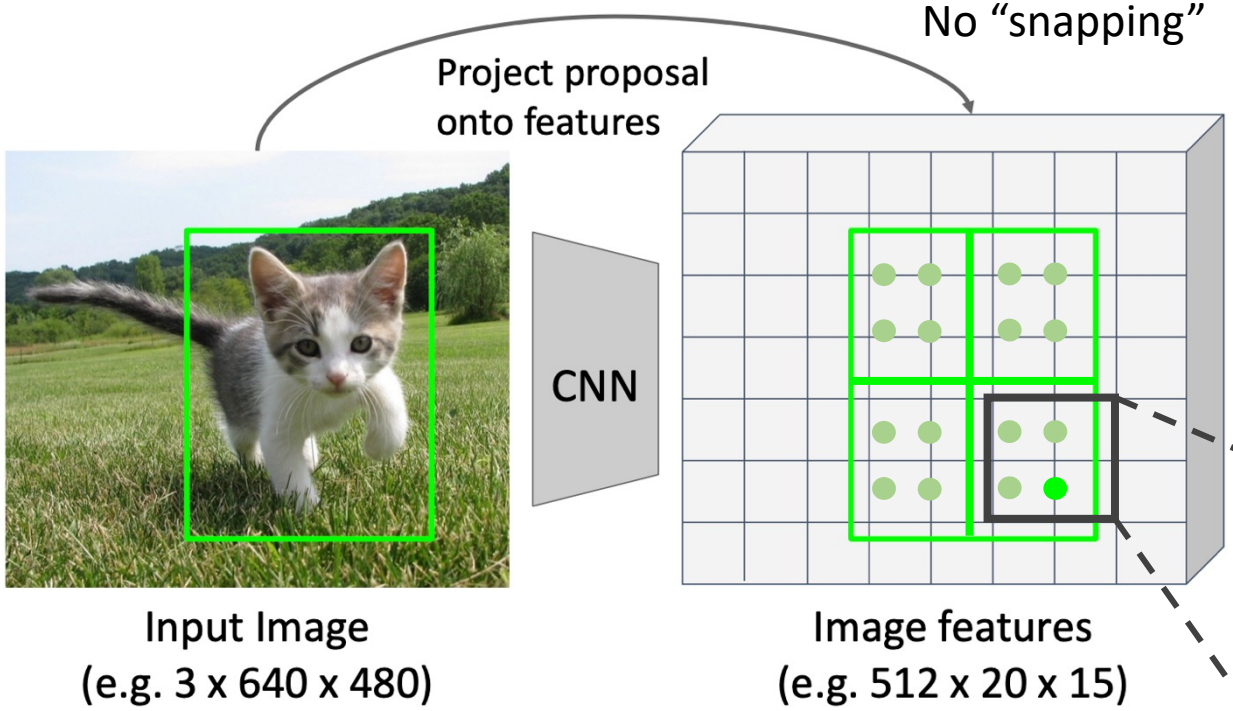
Feature f_{xy} for point (x,y) is a linear combination of features at its four neighboring grid cells

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

$f_{6.5,5.8}$ = ($f_{6,5}$ * 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)

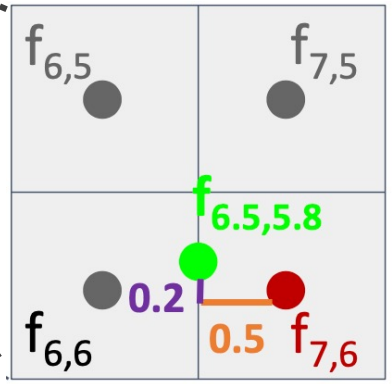
Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015

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**

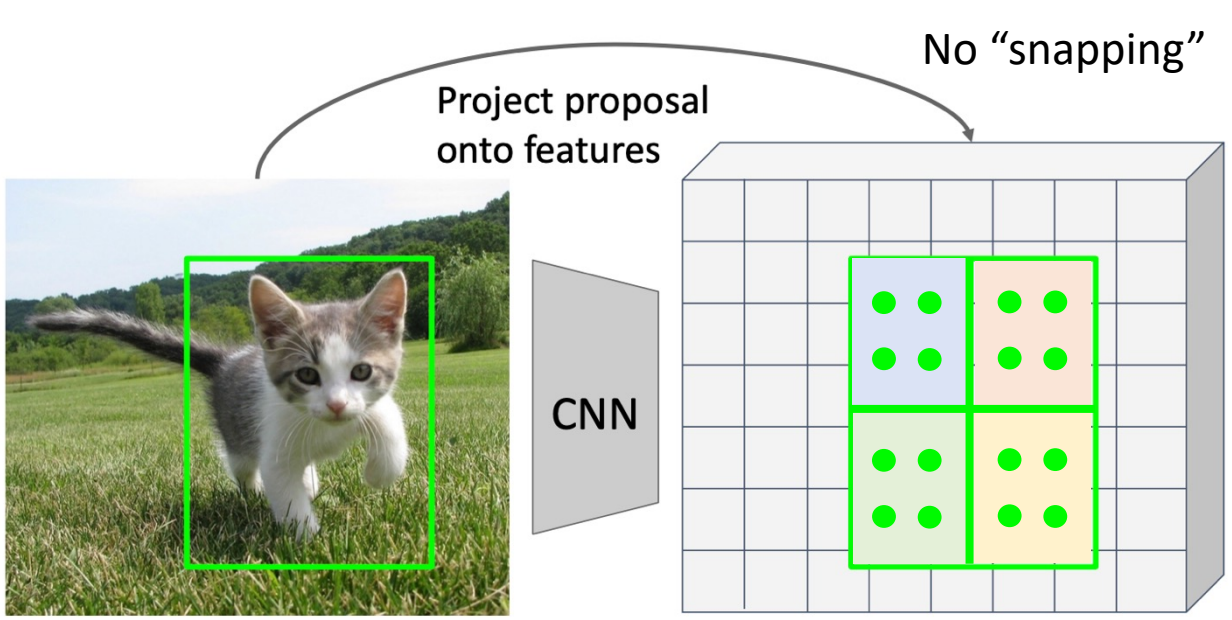


Feature f_{xy} for point (x,y) is a linear combination of features at its four neighboring grid cells

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

Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015

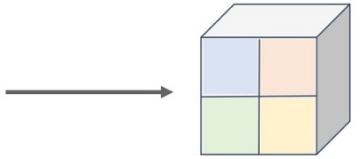
Cropping features – RoI Align



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

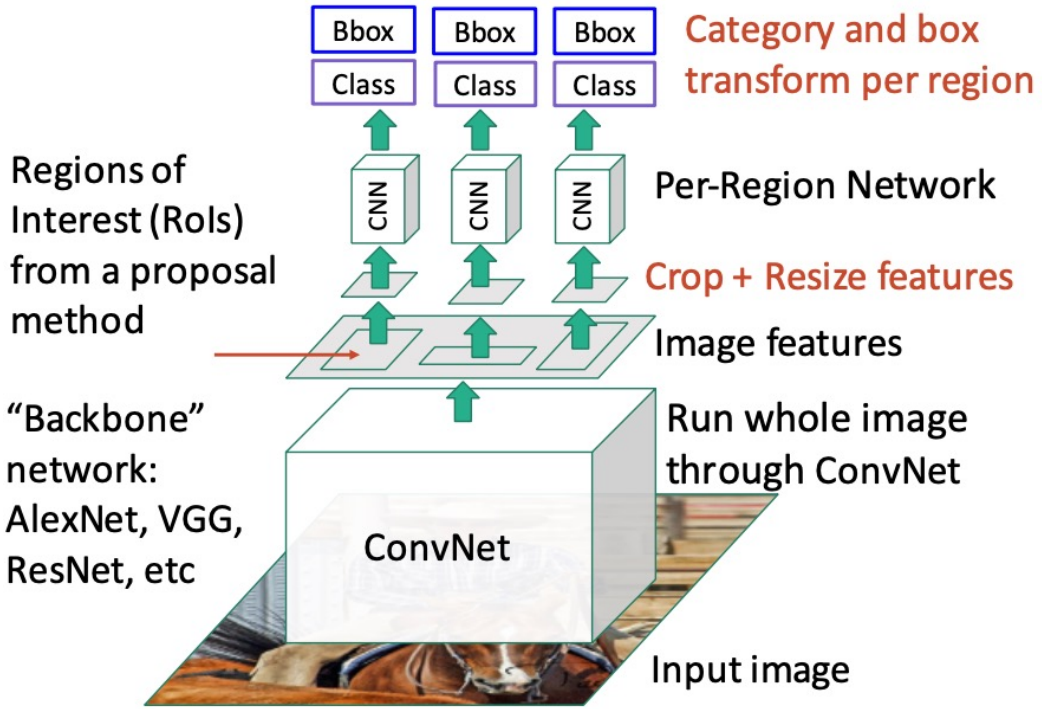


Region features
(here 512 x 2 x 2;
In practice e.g 512 x 7 x 7)

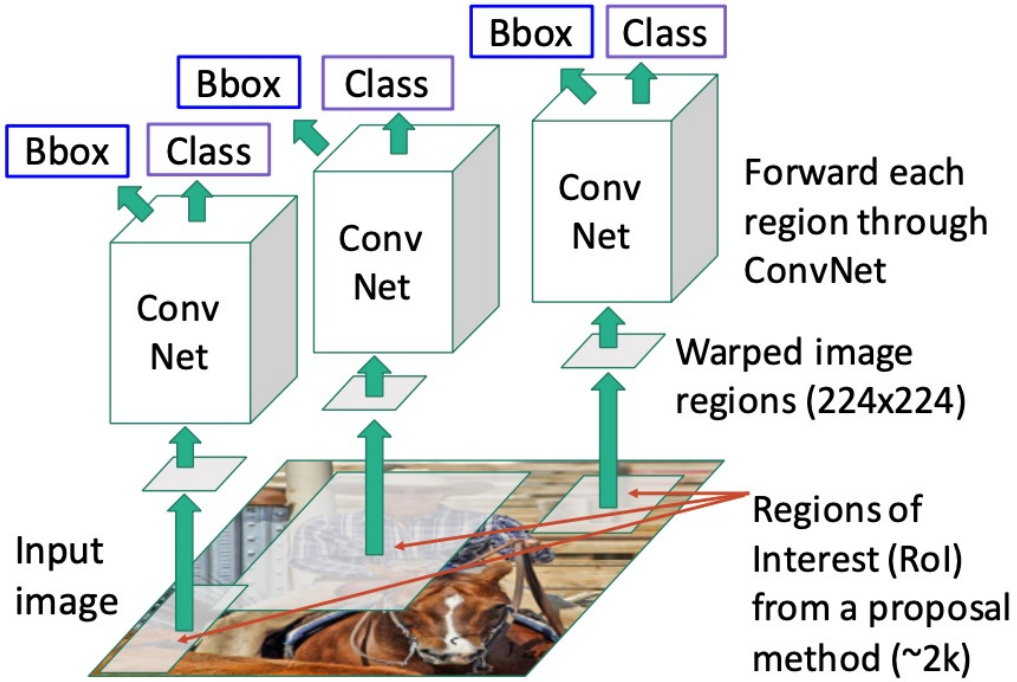
Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015

Fast R-CNN vs “Slow” R-CNN

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

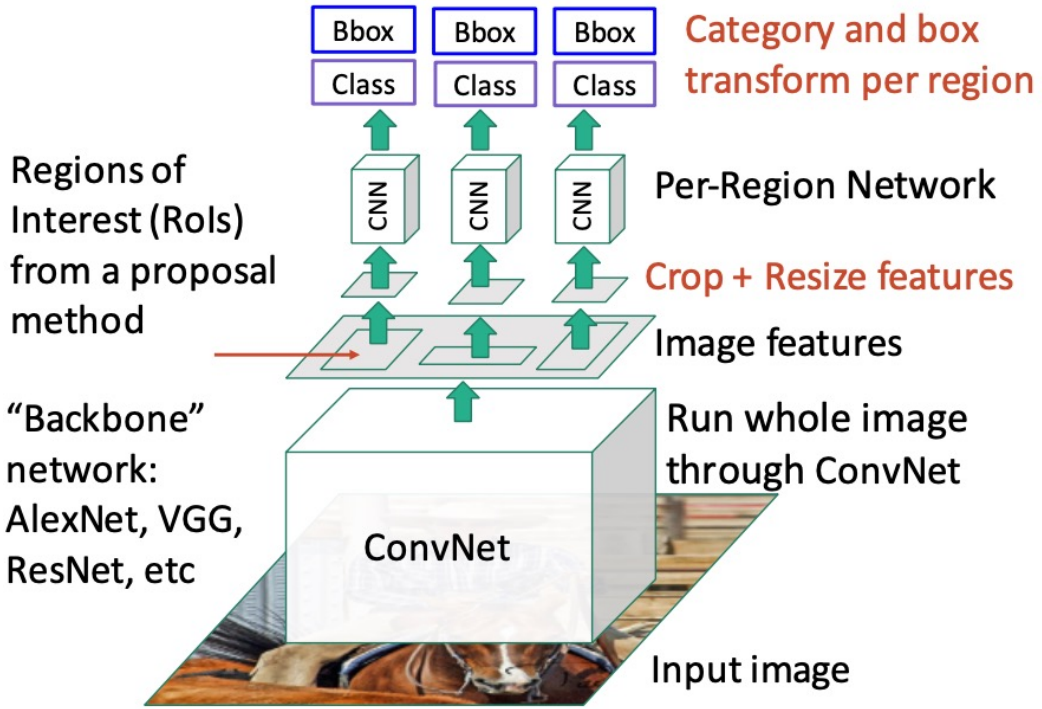


“Slow” R-CNN: Run CNN independently for each region

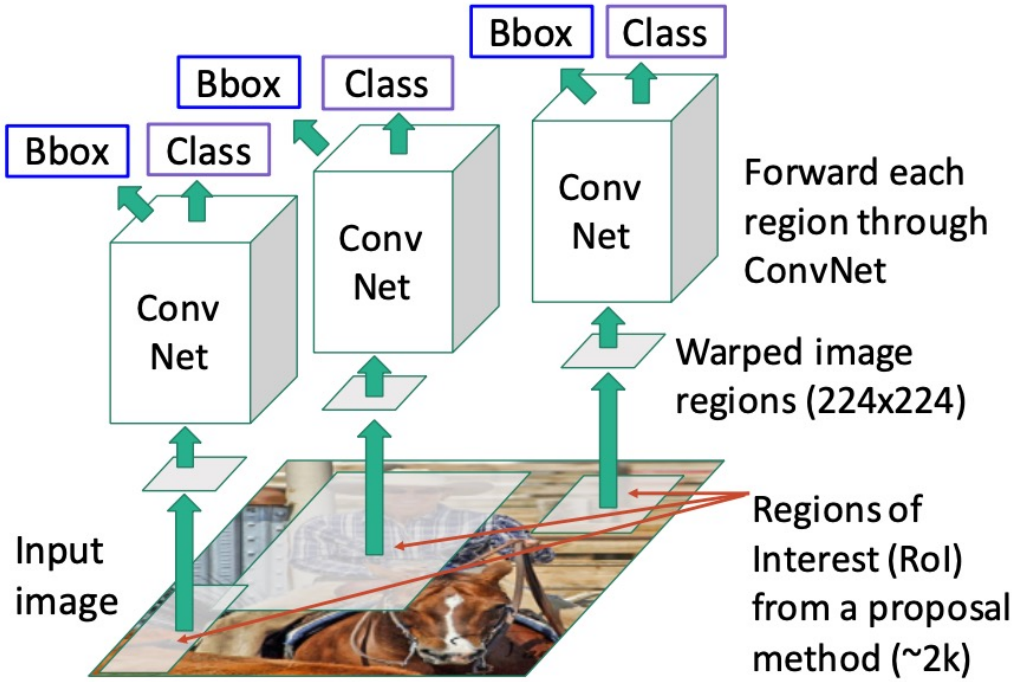


Fast R-CNN vs “Slow” R-CNN

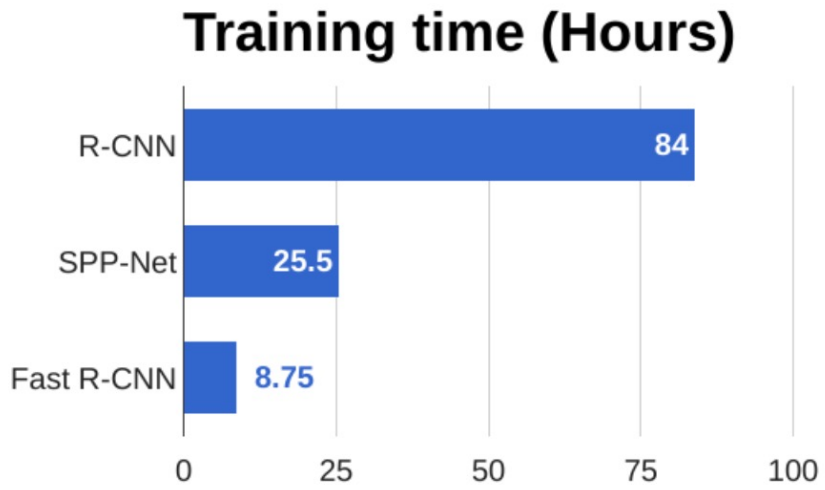
Fast R-CNN: Apply differentiable cropping to shared image features

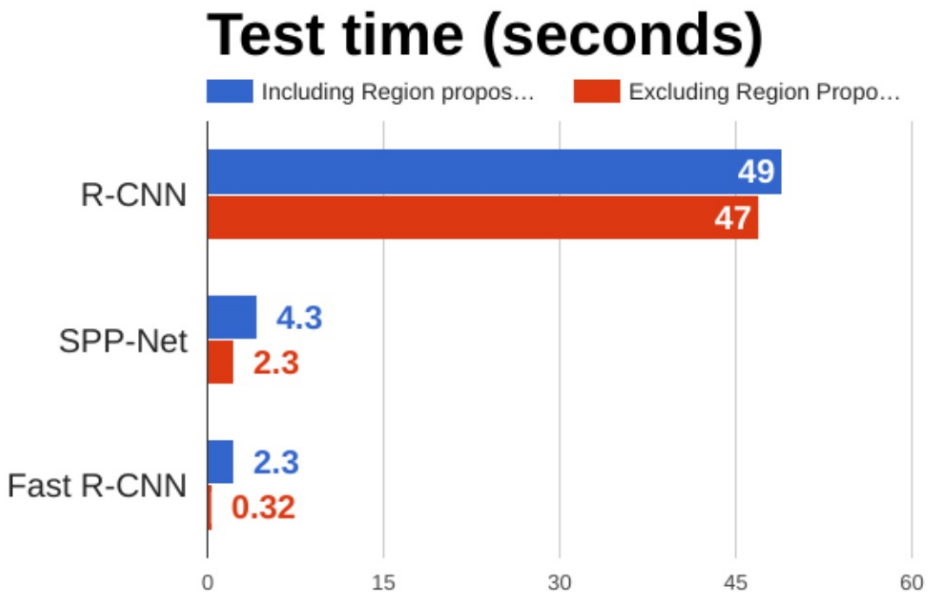


“Slow” R-CNN: Run CNN independently for each region



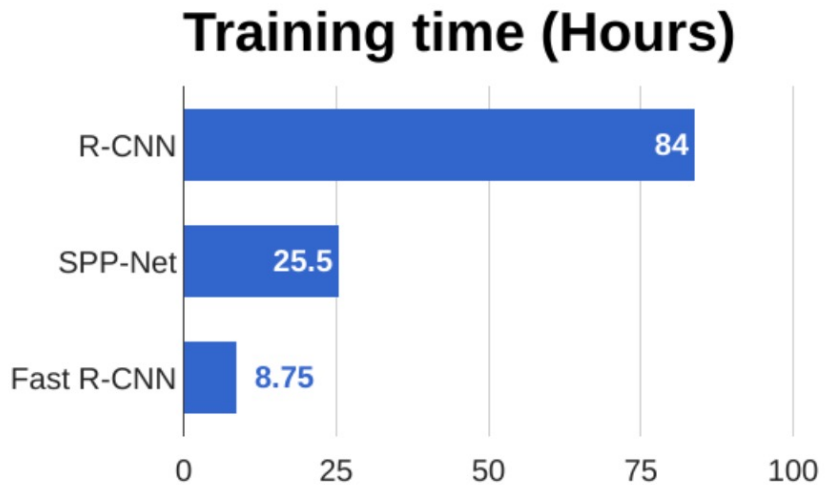
Fast R-CNN vs “Slow” R-CNN

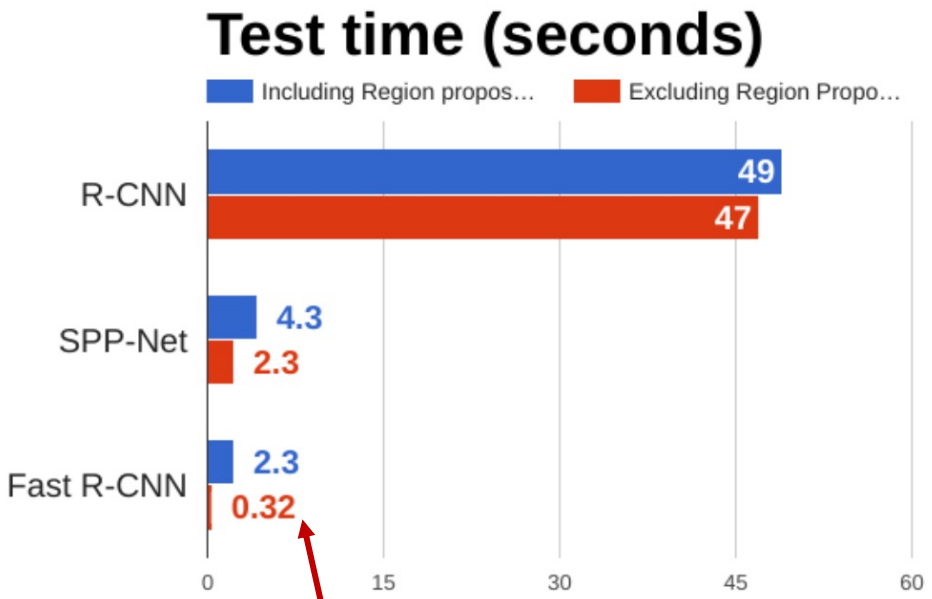




Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014. He et al, “Spatial pyramid pooling in deep convolutional networks for visual recognition”, ECCV 2014
Girshick, “Fast R-CNN”, ICCV 2015

Fast R-CNN vs “Slow” R-CNN

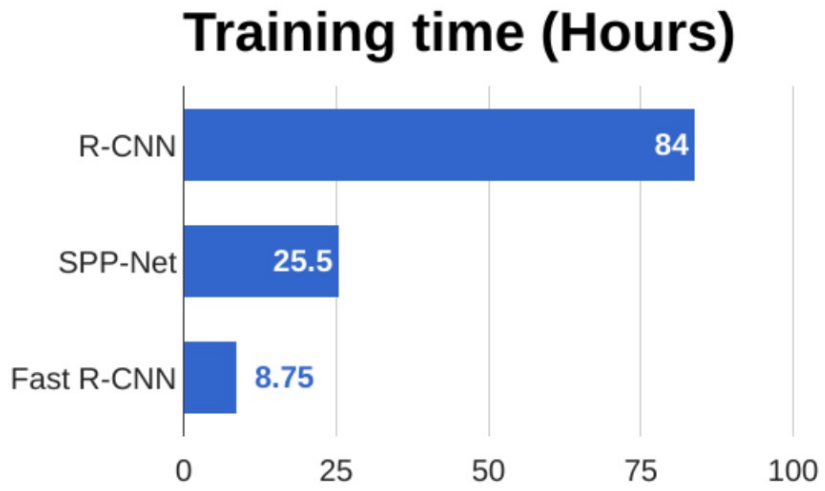


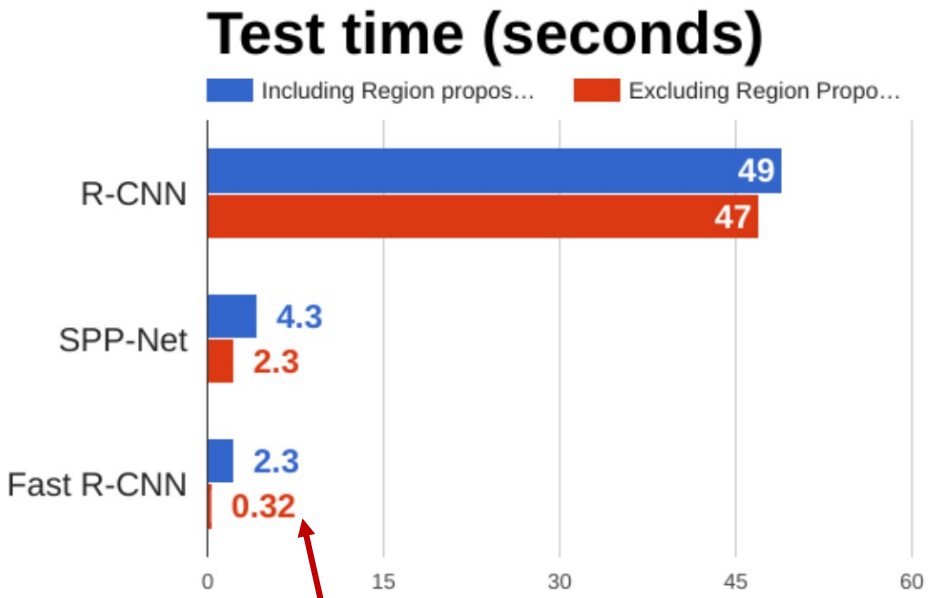


Problem: Runtime dominated by region proposals!

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





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

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

Outline

- Task definition
- Region-based CNN and basic concepts
  - Intersection over Union (IoU)
  - Non-Max Suppression (NMS)
  - Mean Average Precision (mAP)
- Fast R-CNN
- Faster R-CNN

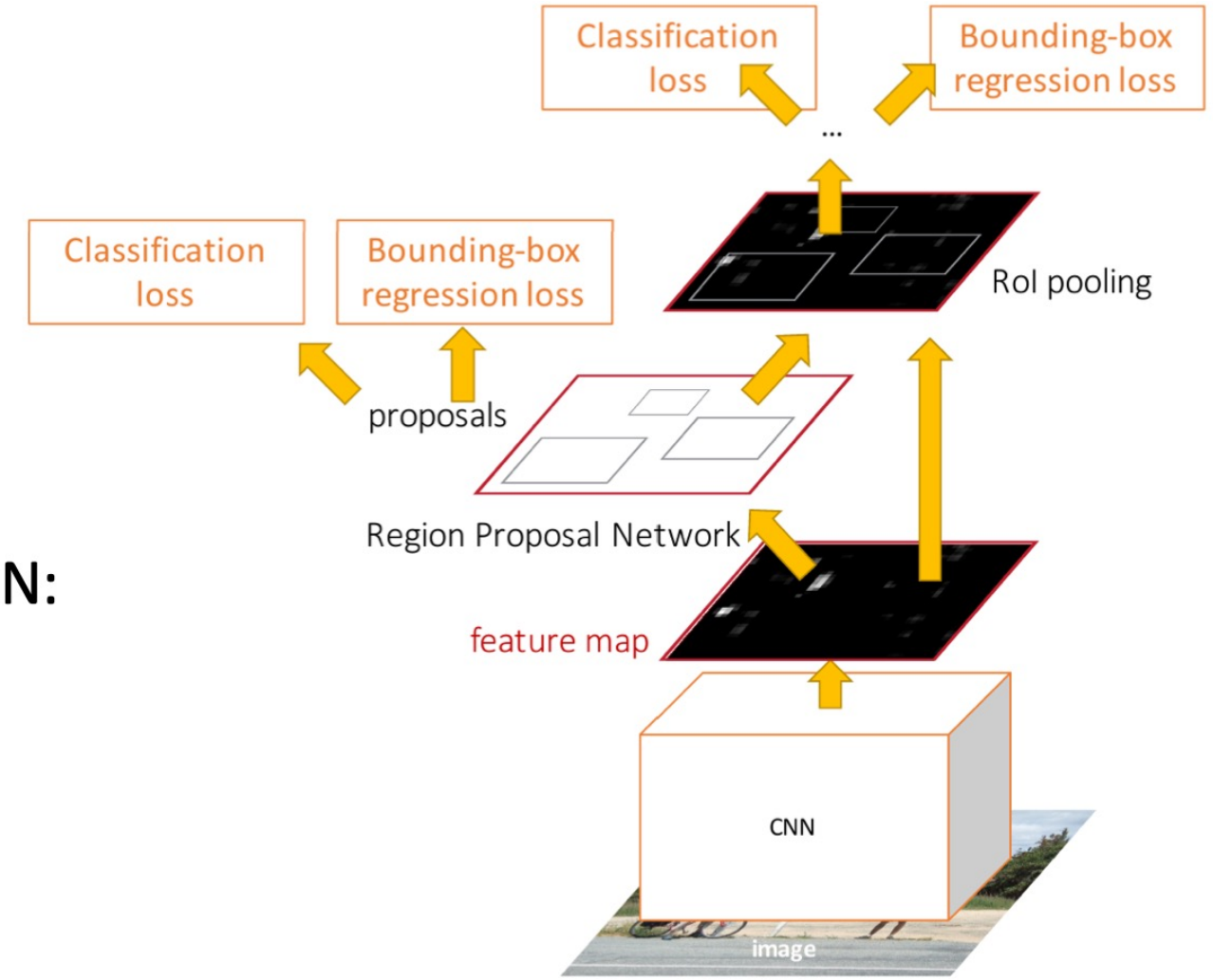
- *Tutorial 2*: Faster-RCNN

Faster R-CNN

Faster R-CNN: Learnable Region Proposals

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

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



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

Region Proposal Network (RPN)

Run backbone CNN to get
features aligned to input image

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

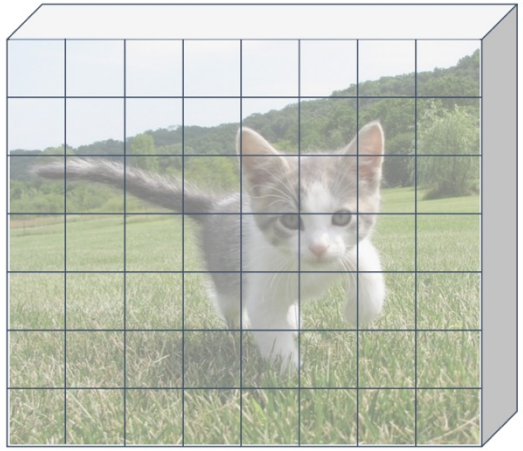
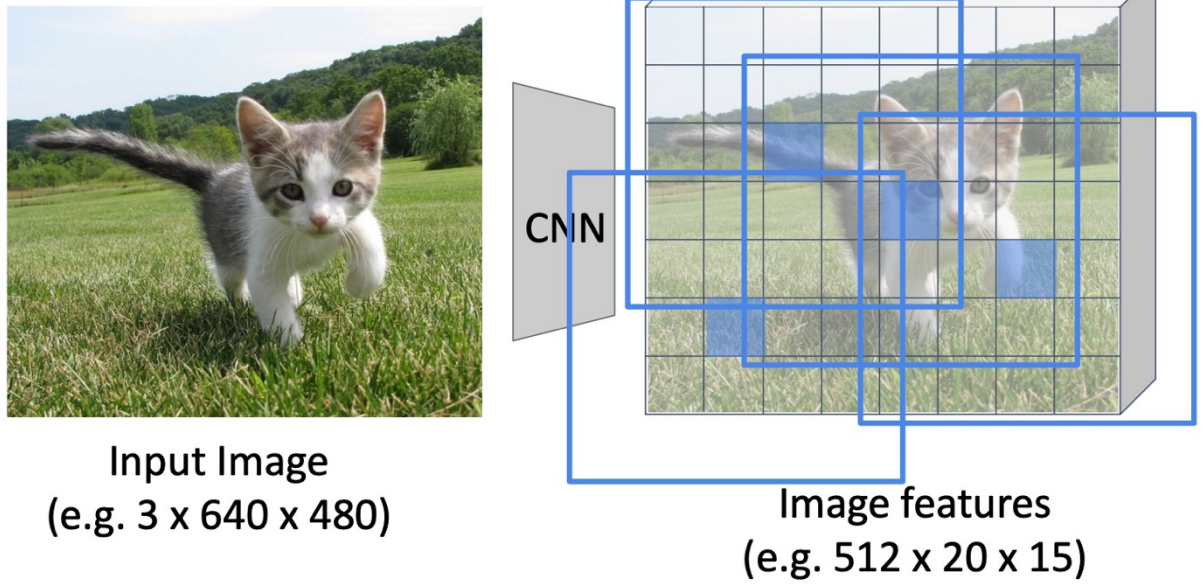


Image features
(e.g. 512 x 20 x 15)

Region Proposal Network (RPN)

Run backbone CNN to get features aligned to input image



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

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)

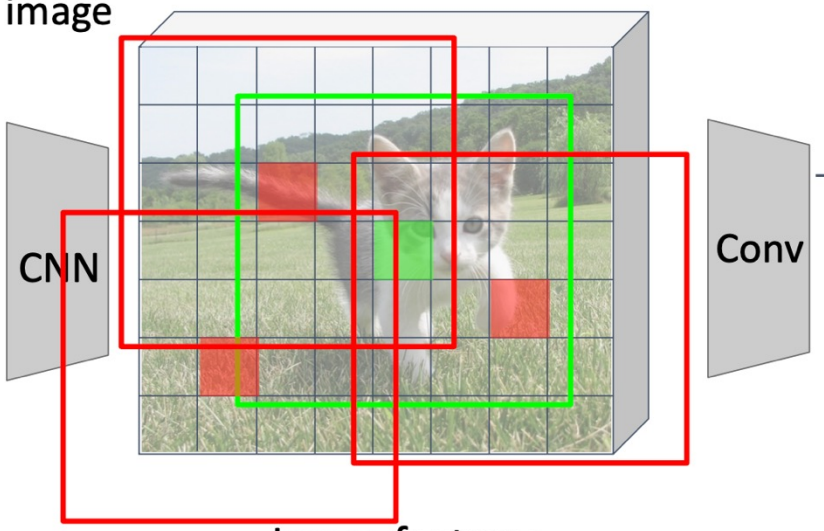


Image features
(e.g. 512 x 20 x 15)

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)

Run backbone CNN to get features aligned to input image

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

CNN

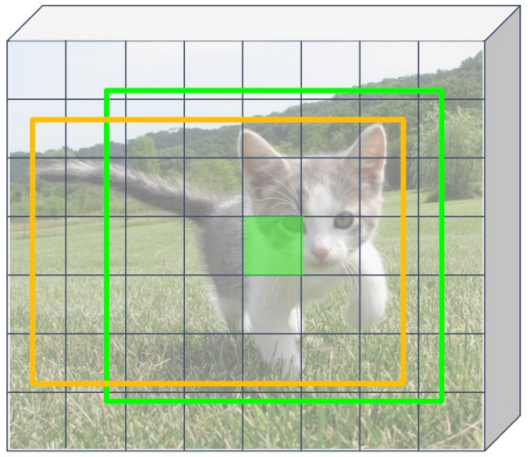


Image features
(e.g. 512 x 20 x 15)

Conv

Anchor is an object?
1 x 20 x 15

Box transforms
4 x 20 x 15

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

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

Region Proposal Network (RPN)

Run backbone CNN to get features aligned to input image

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

CNN

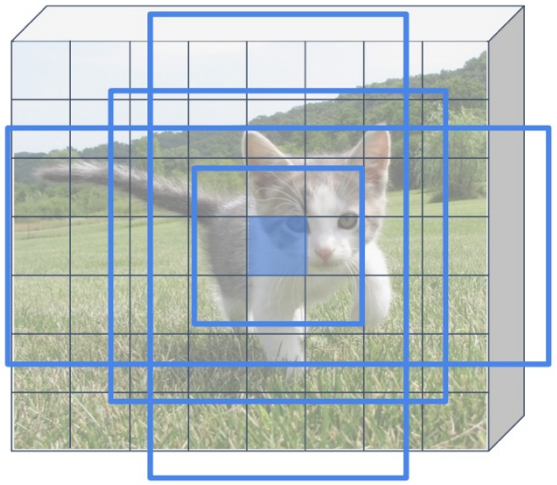


Image features
(e.g. 512 x 20 x 15)

Conv

Anchor is an object?
K x 20 x 15

Box transforms
4K x 20 x 15

Problem: Anchor box may have the wrong size / shape

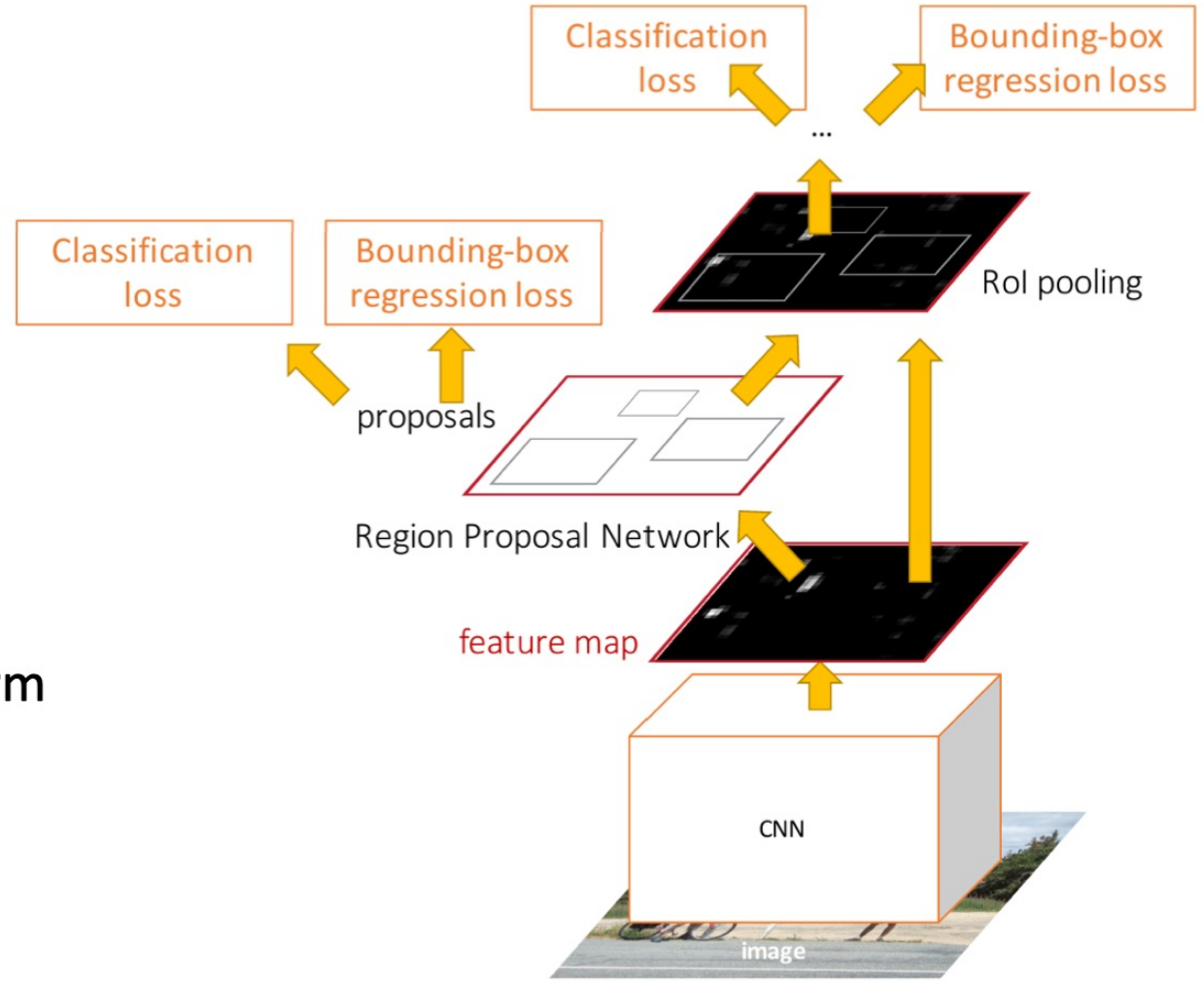
Solution: Use **K** different anchor boxes at each point!

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

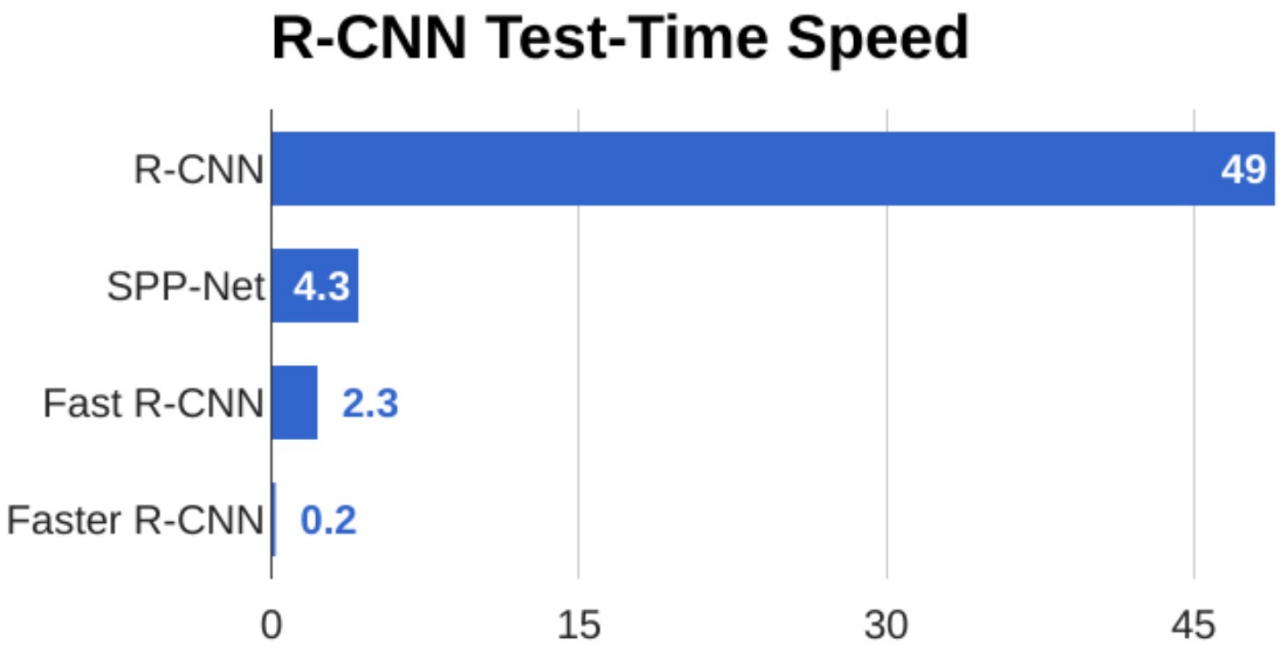
Jointly train with 4 losses:

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 Figure copyright 2015, Ross Girshick

Faster R-CNN: Learnable Region Proposals

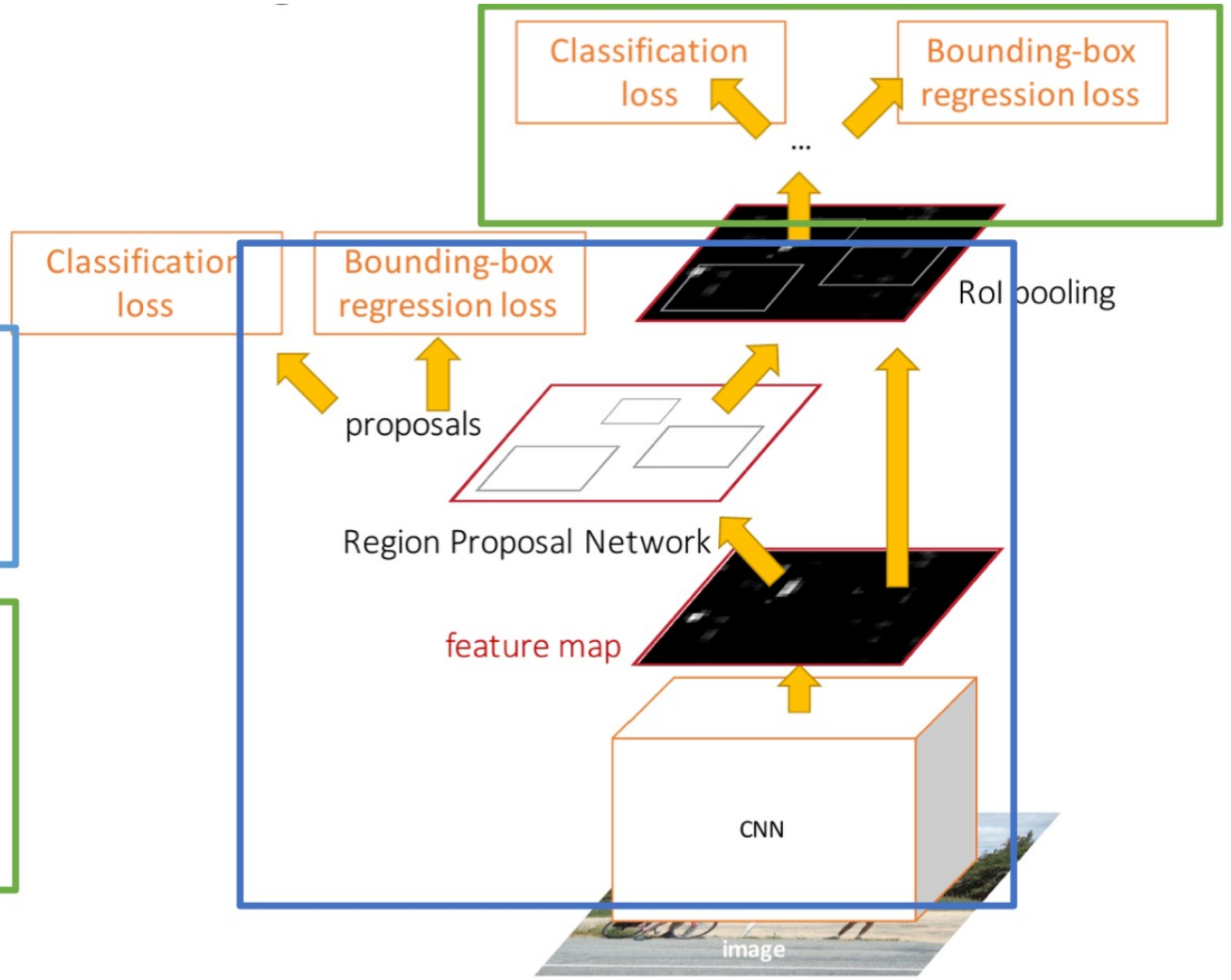


Faster R-CNN: Learnable Region Proposals

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



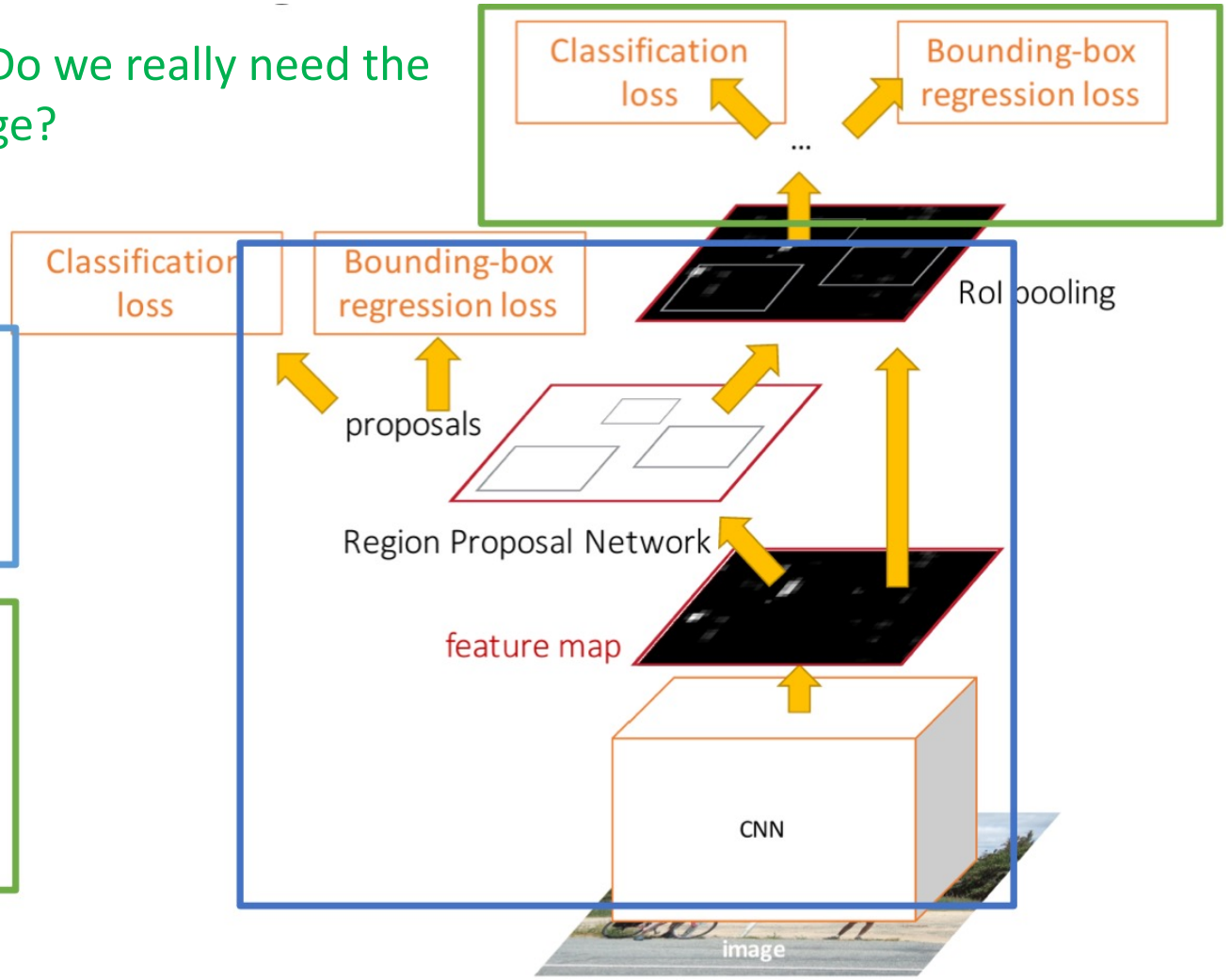
Faster R-CNN: Learnable Region Proposals

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)

CNN

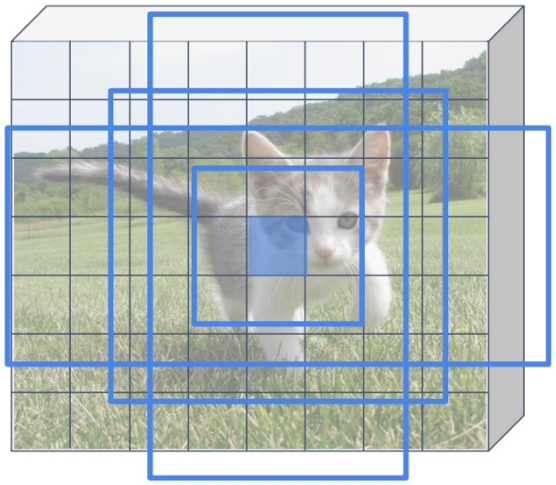


Image features
(e.g. 512 x 20 x 15)

Conv

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)

CNN

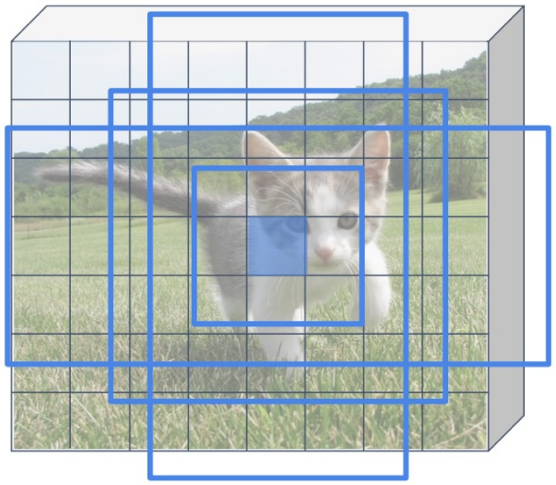


Image features
(e.g. 512 x 20 x 15)

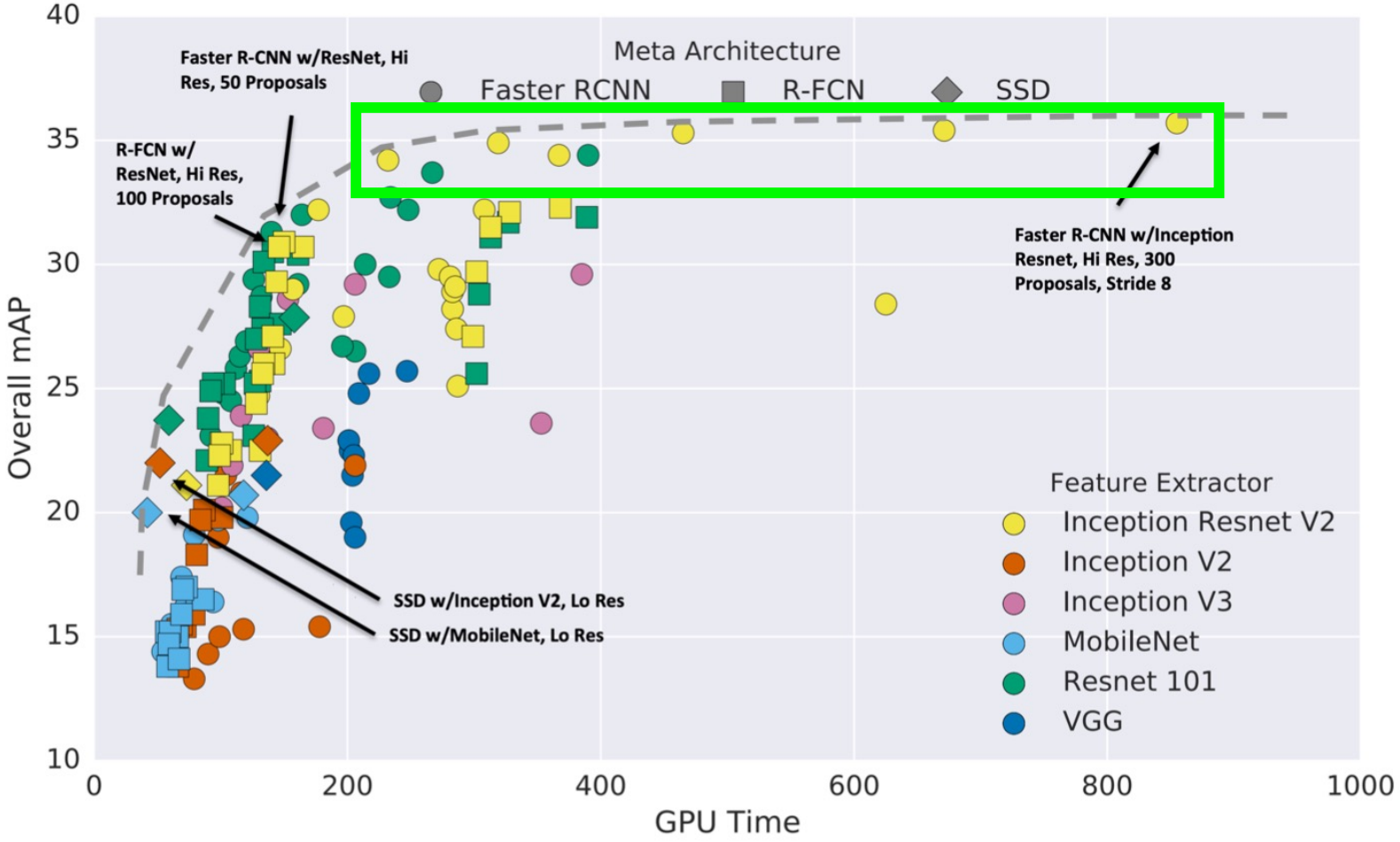
Conv

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

Object detection: lots of variables!

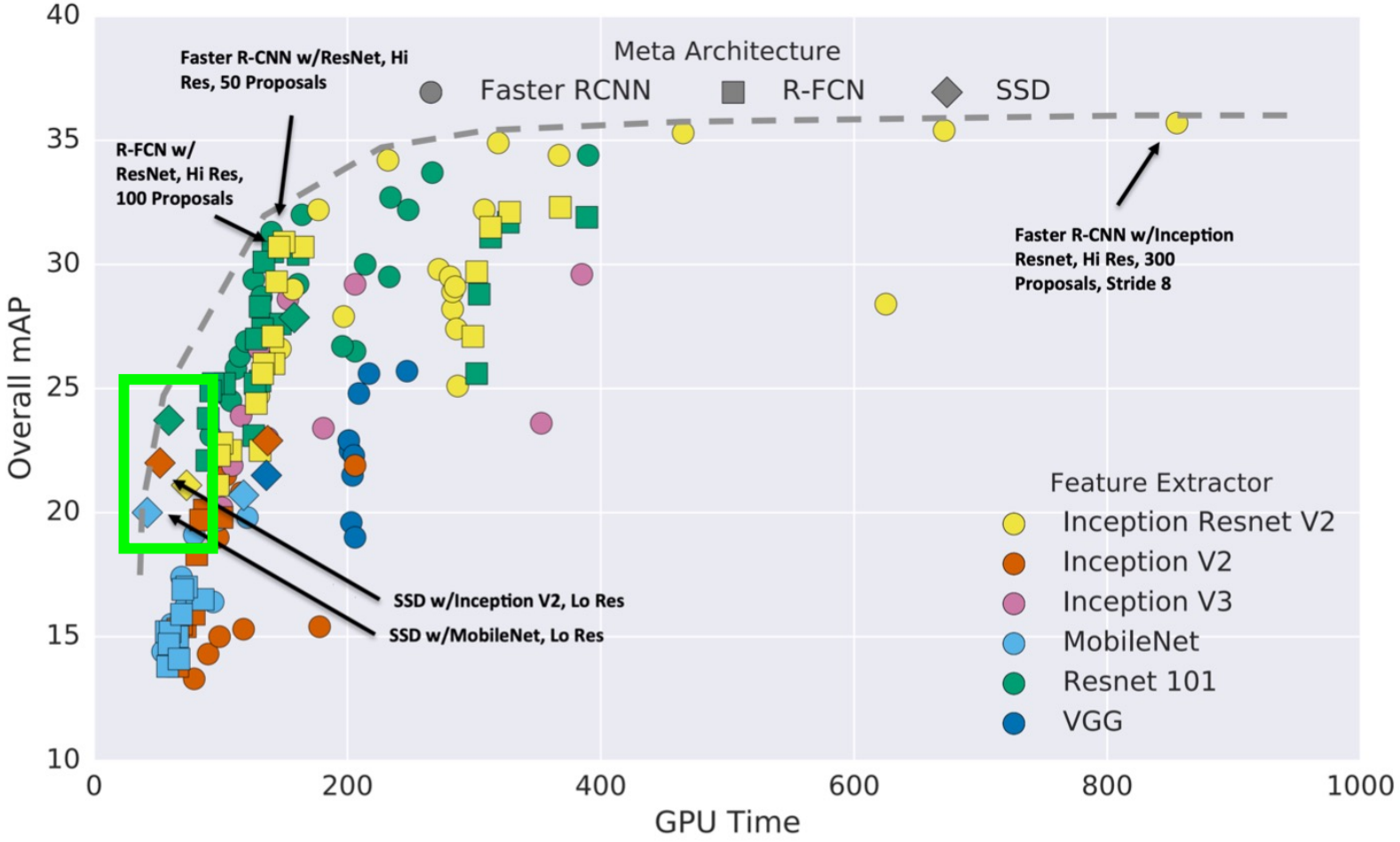


Takeaways:

- 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!

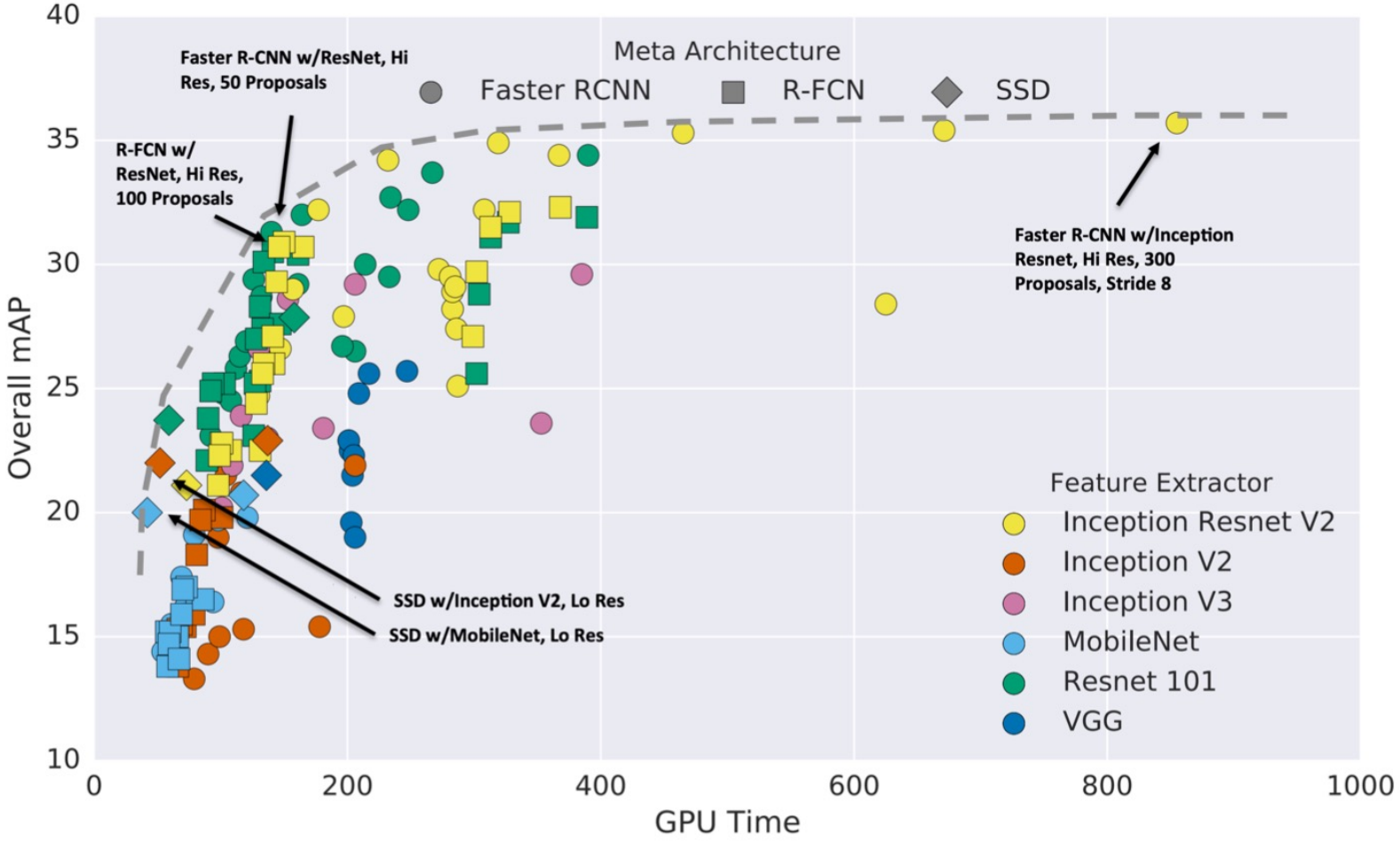


Takeaways:

- 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!

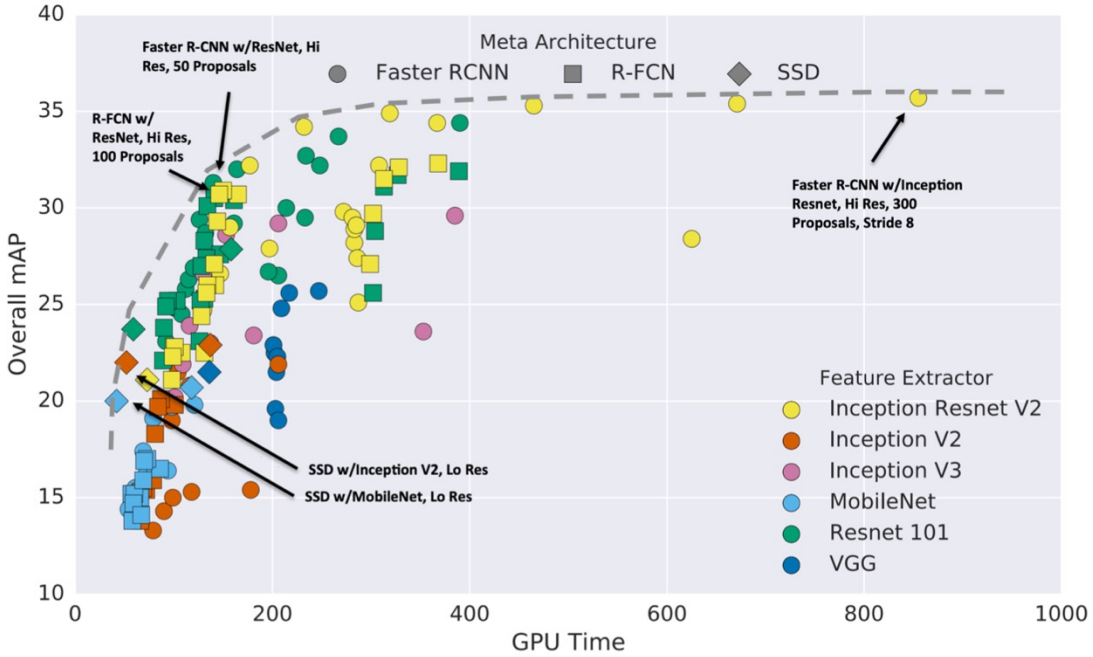


Takeaways:

- 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
- Diminishing returns for slower methods

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

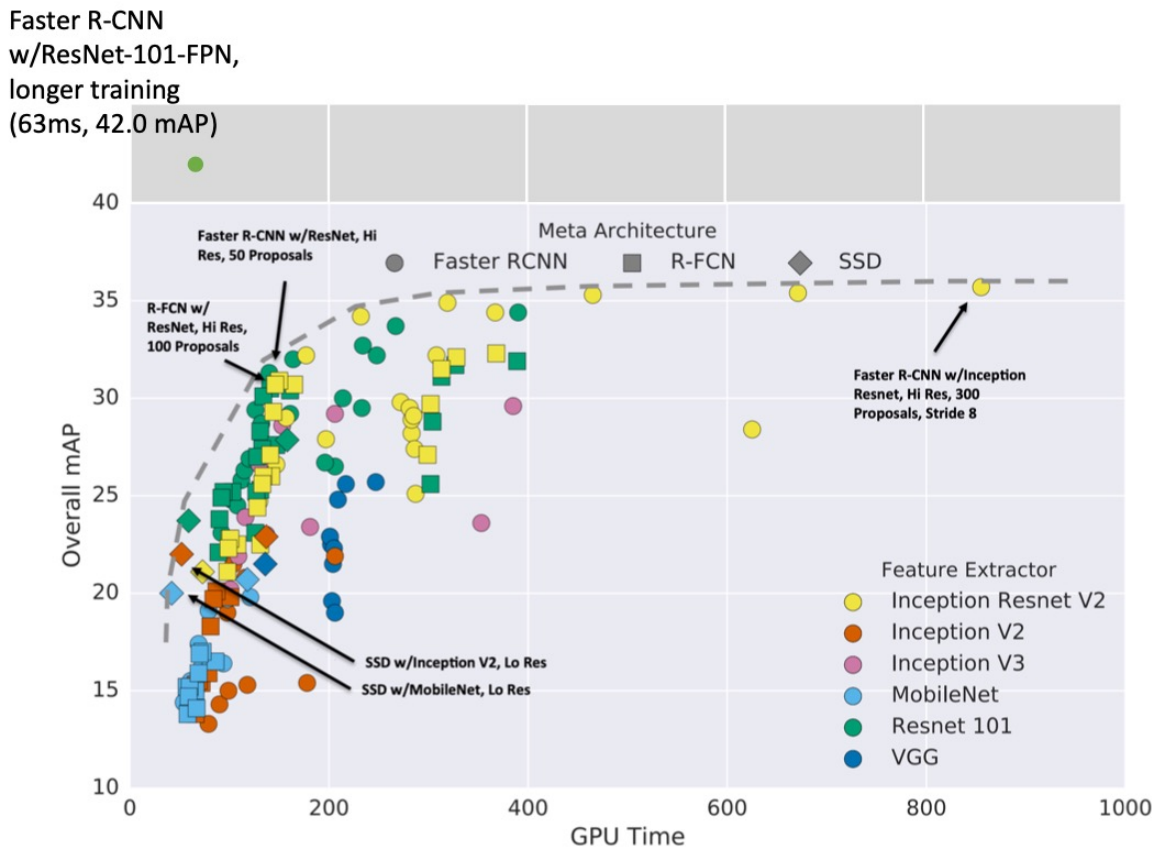
Object detection: lots of variables!



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

Object detection: lots of variables!

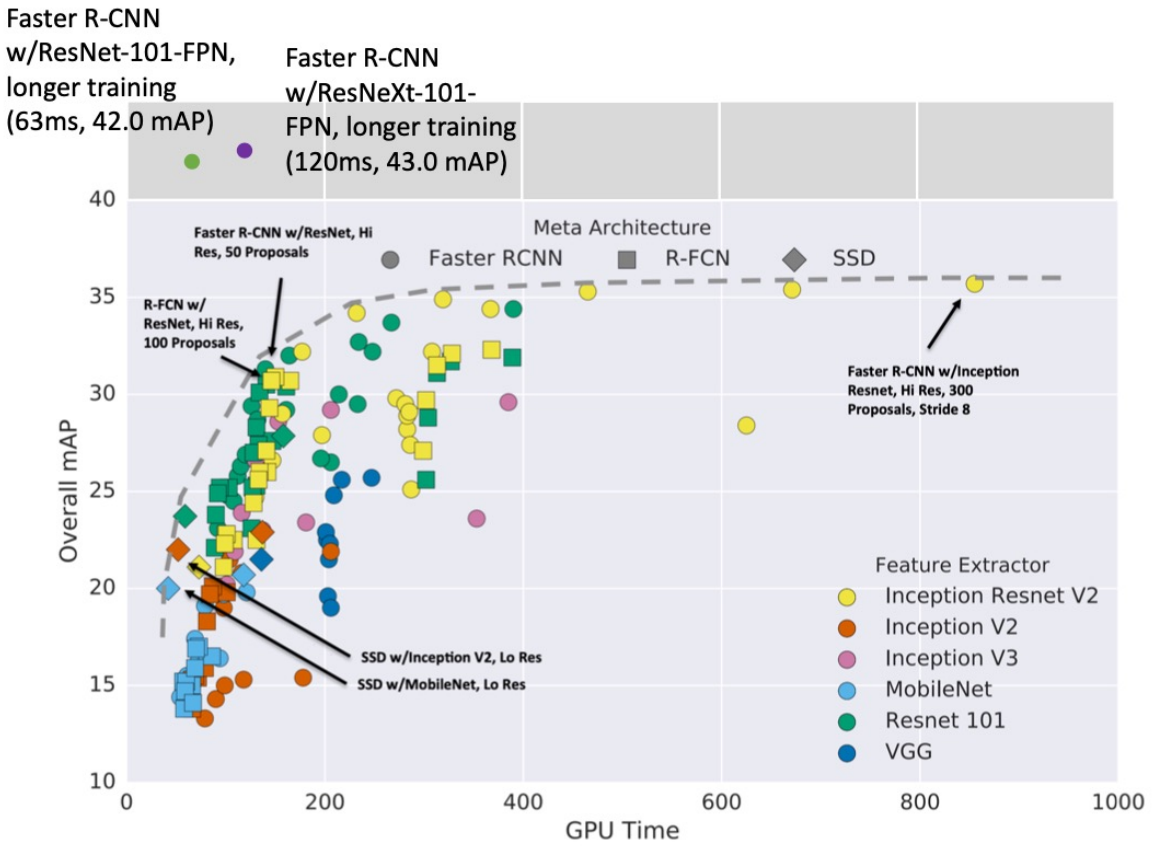


These results are a few years old ... since then GPUs have gotten faster, and we've improved performance with many tricks:

- Train longer!
- Multiscale backbone: Feature Pyramid Networks

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

Object detection: lots of variables!

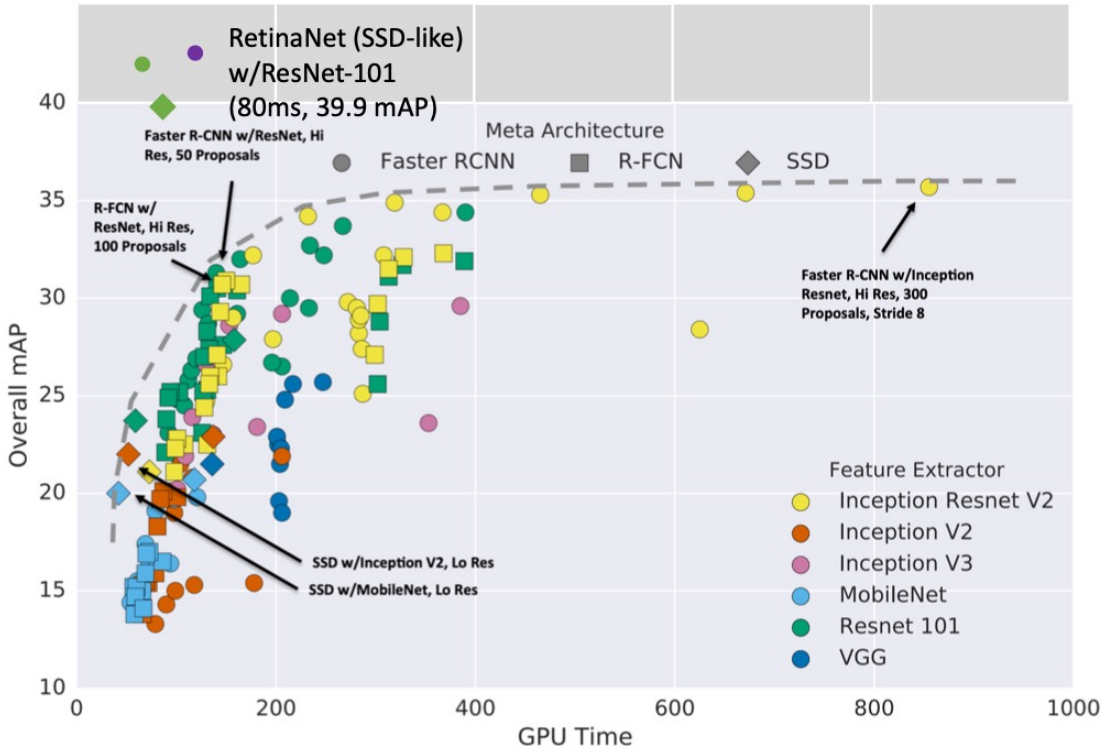


These results are a few years old ... since then GPUs have gotten faster, and we've improved performance with many tricks:

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

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

Object detection: lots of variables!

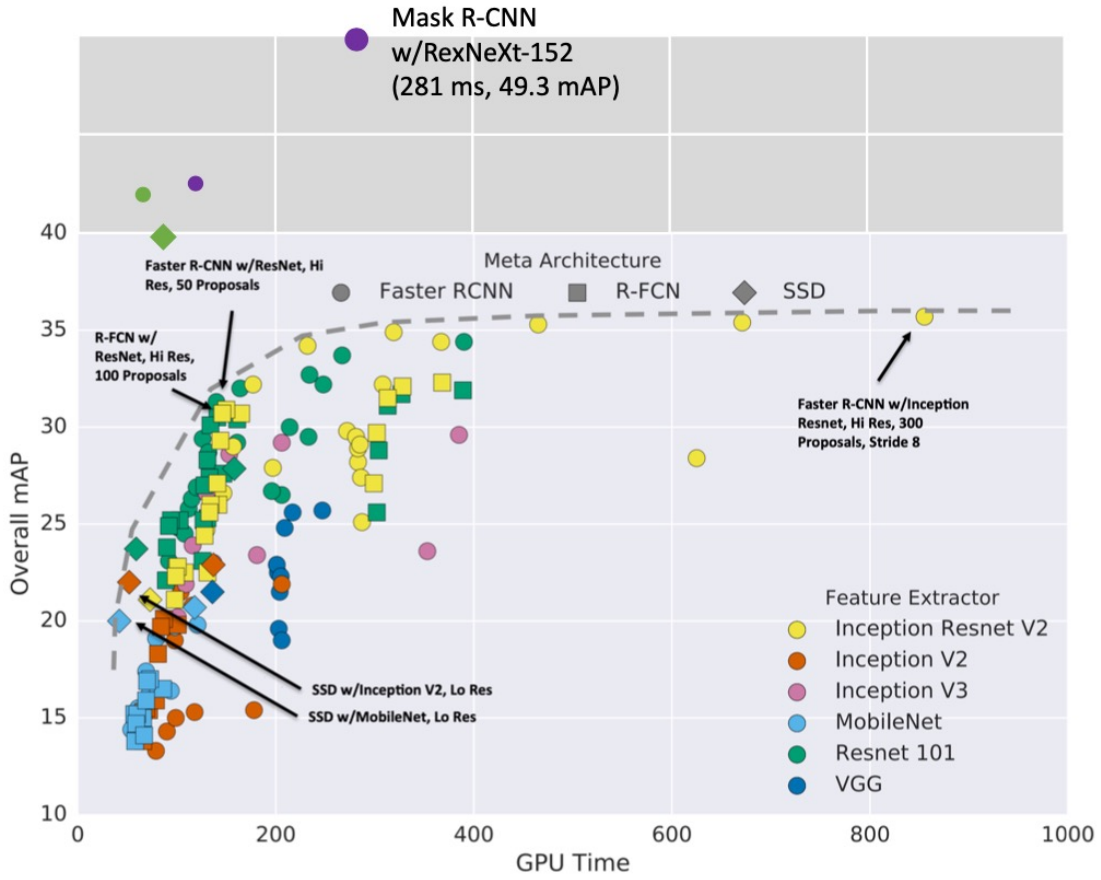


These results are a few years old ... since then GPUs have gotten faster, and we've improved performance with many tricks:

- 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

Object detection: lots of variables!

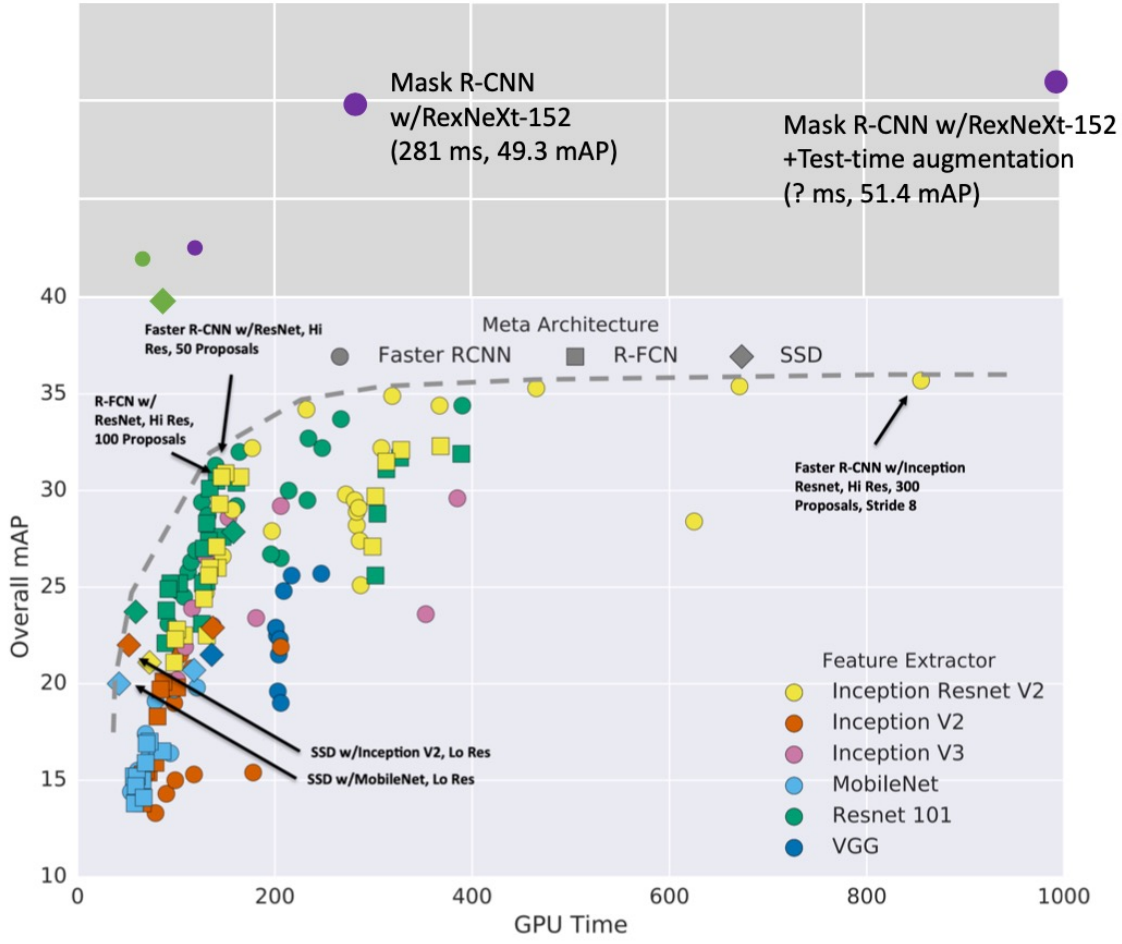


These results are a few years old ... since then GPUs have gotten faster, and we've improved performance with many tricks:

- 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

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

Object detection: open source codes

Object detection is hard! Don’t implement it yourself

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

Object detection: open source codes

Object detection is hard! Don’t implement it yourself

OpenMMLab MMDetection

https://github.com/open-mmlab/mmdetection

Tutorial 2: Faster R-CNN using MMDetection

See *tutorial_02_object_detection.ipynb*

Supported methods:

- [x] RPN
- [x] Fast R-CNN
- [x] Faster R-CNN
- [x] Mask R-CNN
- [x] Cascade R-CNN
- [x] Cascade Mask R-CNN
- [x] SSD
- [x] RetinaNet
- [x] GHM
- [x] Mask Scoring R-CNN
- [x] Double-Head R-CNN
- [x] Hybrid Task Cascade
- [x] Libra R-CNN
- [x] Guided Anchoring
- [x] FCOS
- [x] RepPoints
- [x] Foveabox
- [x] FreeAnchor
- [x] NAS-FPN
- [x] ATSS
- [x] FSAF
- [x] PAFPN
- [x] Dynamic R-CNN
- [x] PointRend
- [x] CARAFE
- [x] DCNv2
- [x] Group Normalization
- [x] Weight Standardization
- [x] OHEM
- [x] Soft-NMS
- [x] Generalized Attention
- [x] GCNet
- [x] Mixed Precision (FP16) Training
- [x] InstaBoost
- [x] GRoIE
- [x] DetectoRS
- [x] Generalized Focal Loss
- [x] CornerNet