Introduction to Computer Vision: Object detection (part 2)

Natalia Neverova

Last lecture's recap

- What is the object detection task?
- How is this task formalized?
- How is this task evaluated?
- Why is the task difficult?
- What are important datasets to know about?

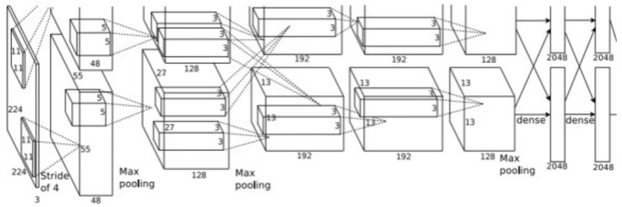
Today's focus

- Principles of multi-object detection
- Region proposals
- Bounding box regression
- R-CNN training and inference
- Non-Max suppression
- Modern object detectors

Recap: object detection task

What objects are in the image and **where** are they?

Detecting a single object



vector: 4096

fully connected:
4096 to 1000

"what?"

class scores:
cat: 0.9
dog: 0.05
car: 0.01
...

correct label:
cat

softmax loss

weighted sum

Loss

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

"where?"

fully connected:
4096 to 4

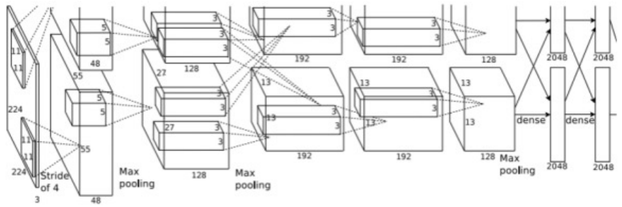
box coordinates:
(x,y,w,h)

L2 loss

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

treat localization as a regression problem!

Detecting a single object



vector: 4096

fully connected:
4096 to 1000

"what?"

class scores:
cat: 0.9
dog: 0.05
car: 0.01
...

correct label:
cat

softmax loss

weighted sum

Loss

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

"where?"

fully connected:
4096 to 4

box coordinates:
(x,y,w,h)

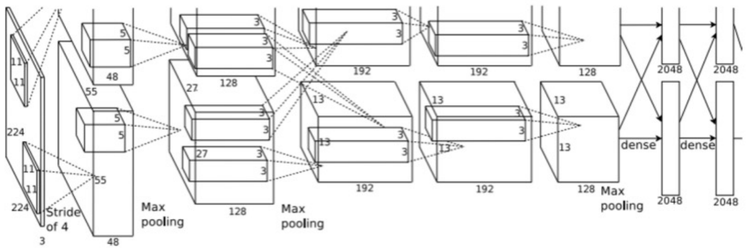
L2 loss

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

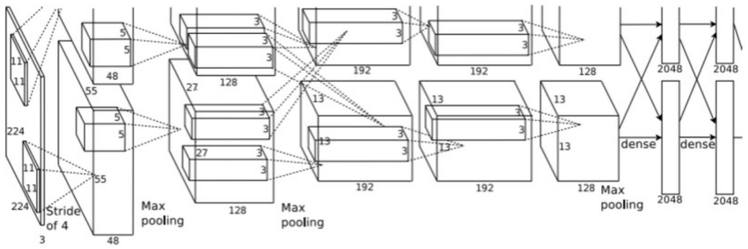
treat localization as a regression problem!

things get much more complicated with multiple objects! (today!)

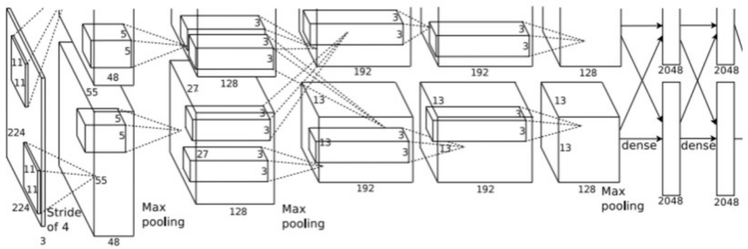
Detecting multiple objects



CAT: (x, y, w, h)



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



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

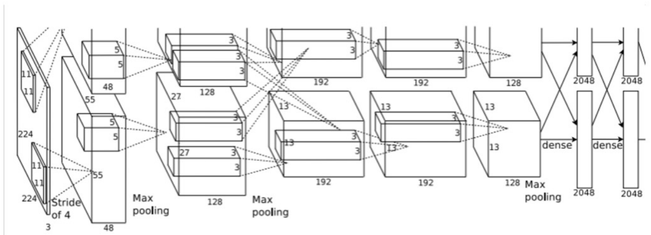
Multiple object detection as ML problem

Detection := the classification of boxes

Intuition: train a classifier to learn to classify every possible box
(box's class is 0 if it's background)

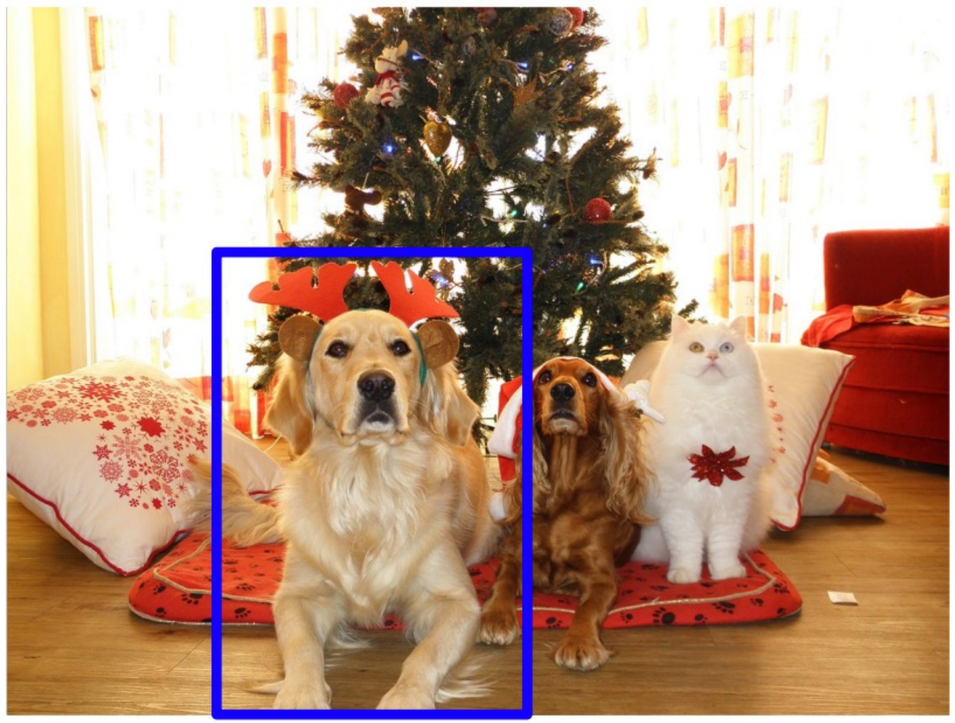
Foundational idea: sliding window

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

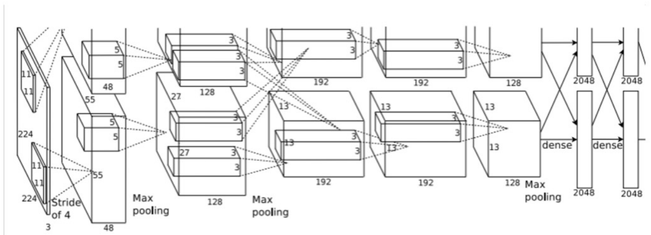


Dog? No
Cat? No
Background? Yes

Foundational idea: sliding window



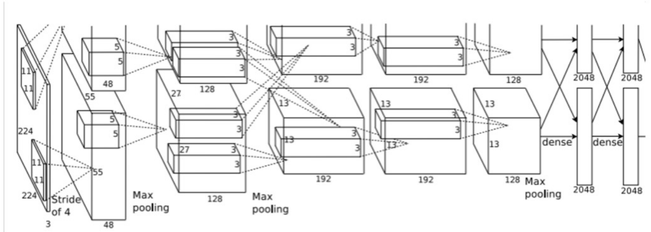
Intuition: apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? Yes
Cat? No
Background? No

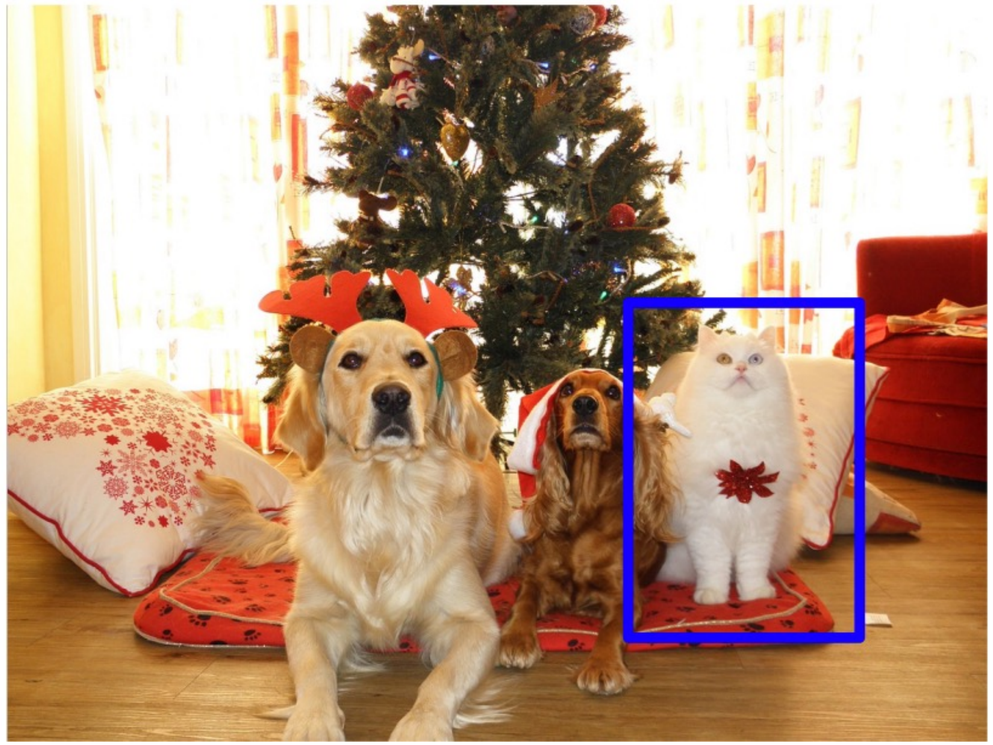
Foundational idea: sliding window

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

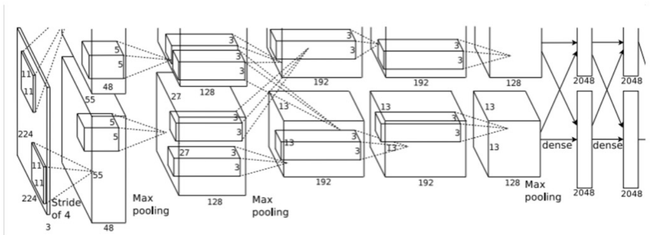


Dog? Yes
Cat? No
Background? No

Foundational idea: sliding window

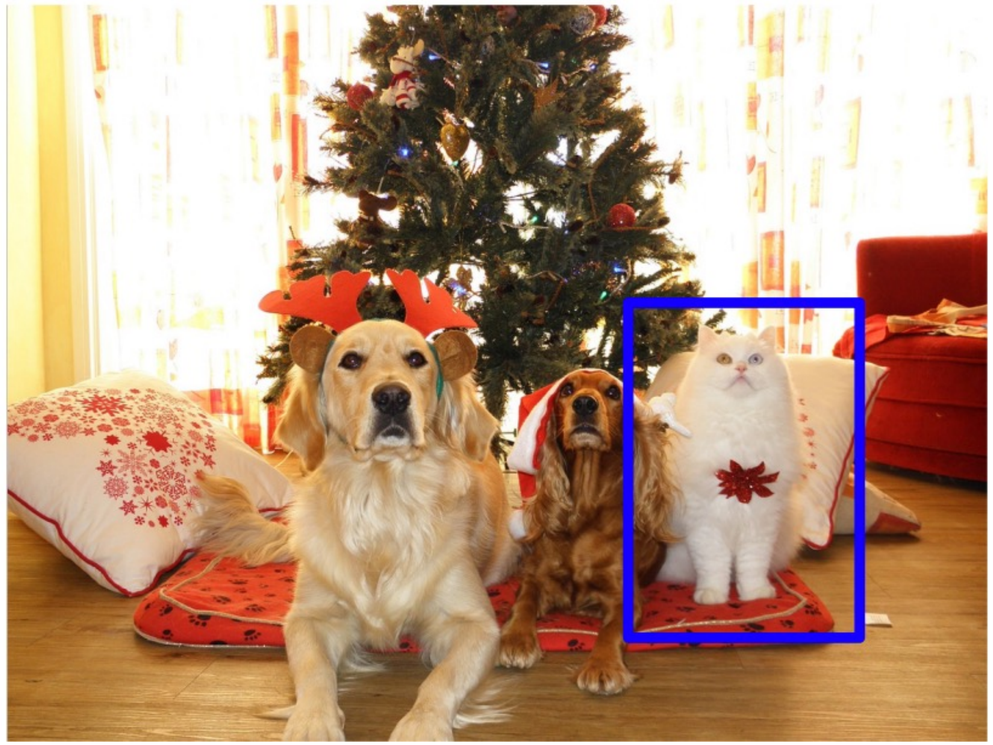


Intuition: apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? No
Cat? Yes
Background? No

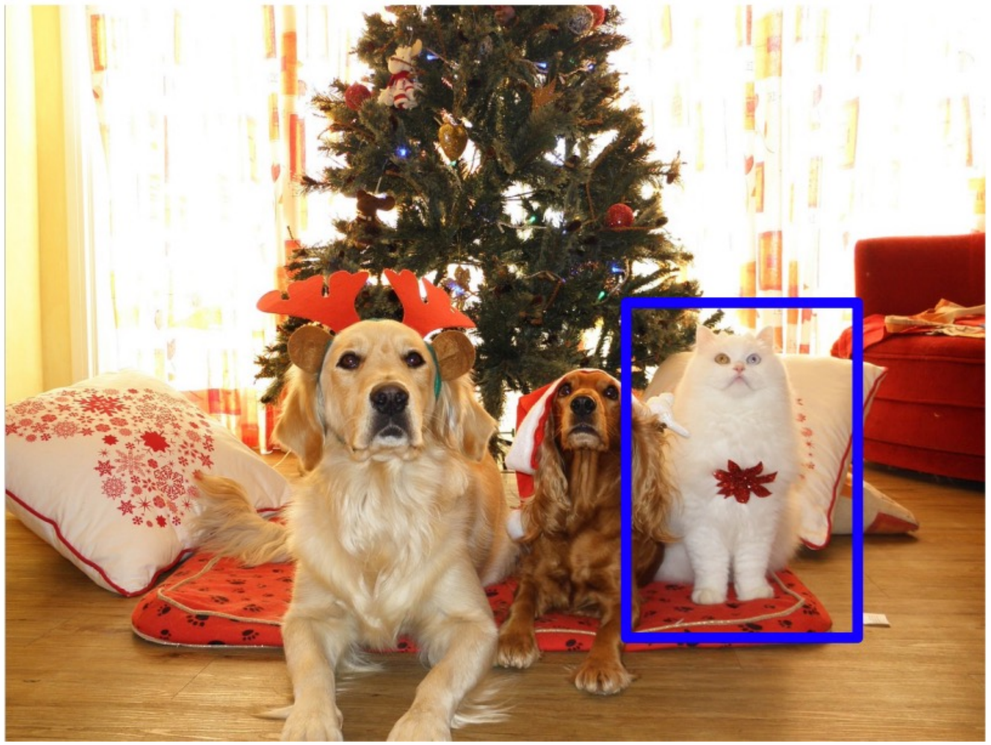
Foundational idea: sliding window



But: How many possible boxes are there in an image 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)

Foundational idea: sliding window

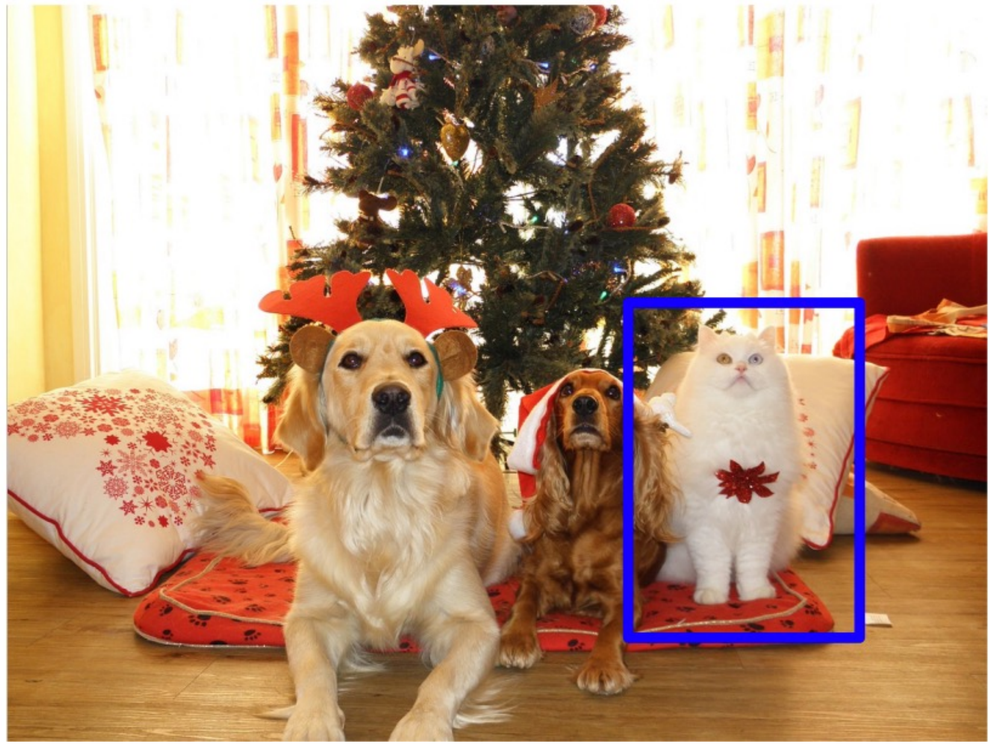


But: How many possible boxes are there in an image size H x W?

Total possible boxes:

$$\sum_{h=1}^{H}\sum_{w=1}^{W}(W-w+1)(H-h+1) = \frac{H(H+1)}{2}\frac{W(W+1)}{2}$$

Foundational idea: sliding window



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

Total possible boxes:

$$\sum_{h=1}^{H}\sum_{w=1}^{W}(W-w+1)(H-h+1) = \frac{H(H+1)}{2}\frac{W(W+1)}{2}$$

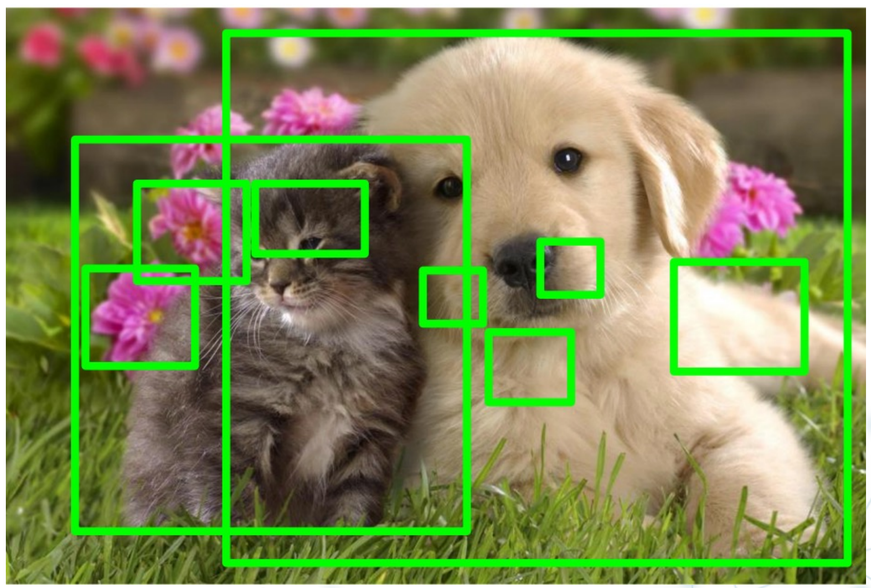
800x600 image has ~58M boxes!
No way we can evaluate them all

Region proposals

Idea: 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 [Uijings et al., 2013] gives 2000 regions proposals in a few seconds on CPU



R-CNN: Region-based CNN

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

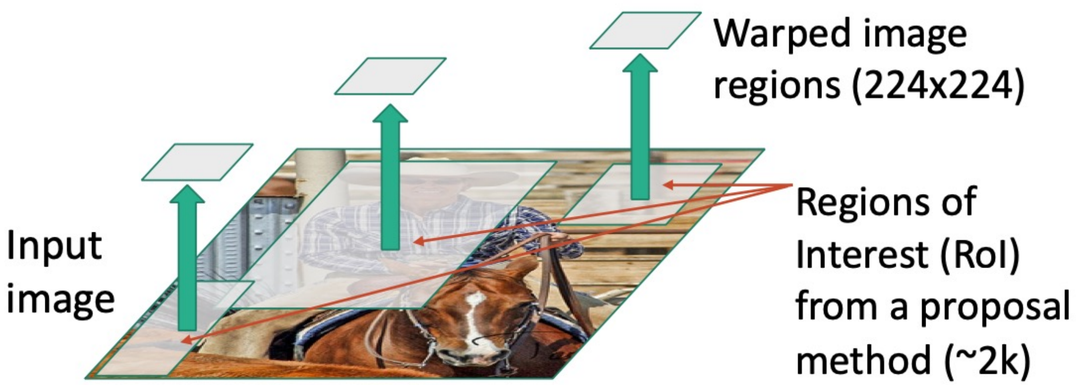
R-CNN: Region-based CNN

(1) select region proposals

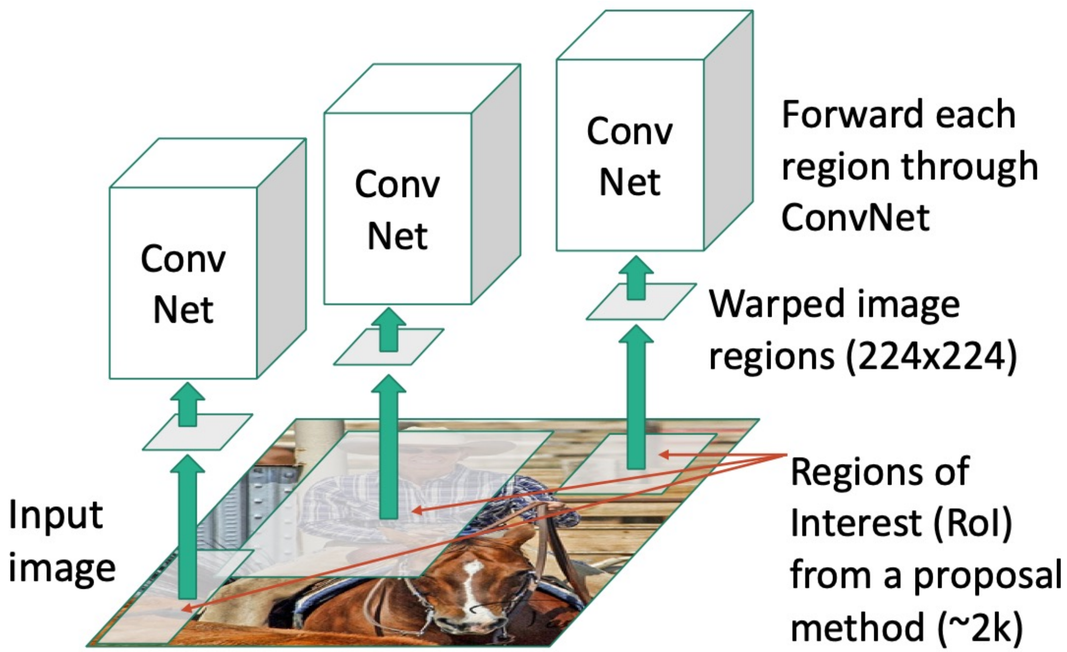
R-CNN: Region-based CNN

(1) select region proposals

(2) normalize in size



R-CNN: Region-based CNN

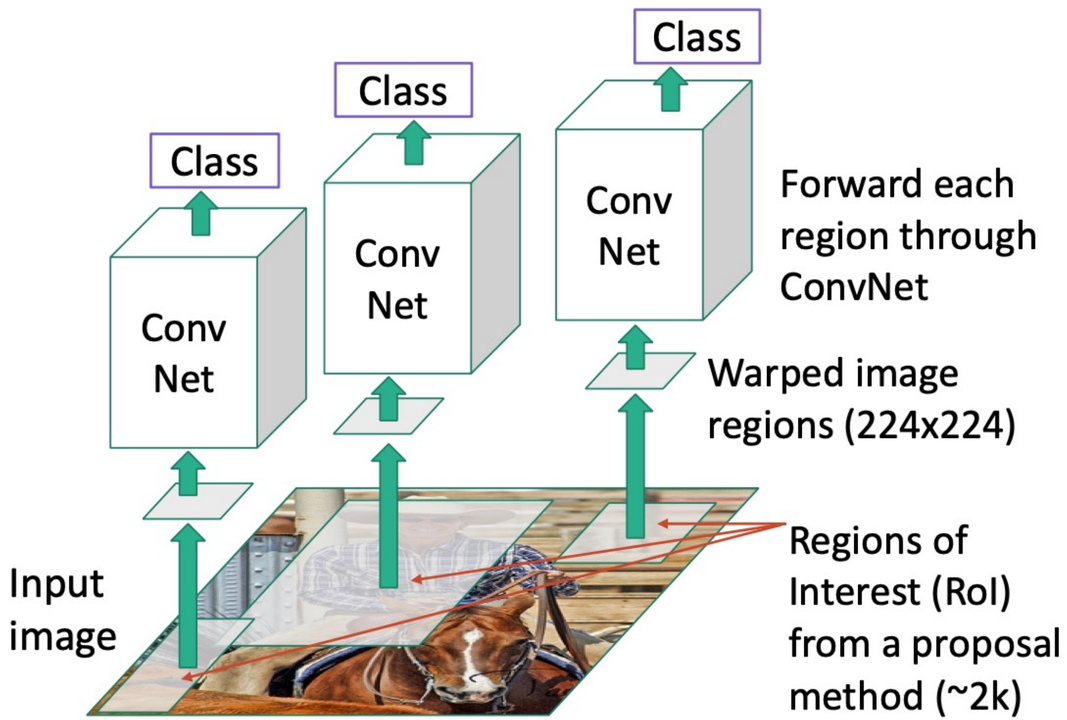


(1) select region proposals

(2) normalize in size

(3) process each region

R-CNN: Region-based CNN



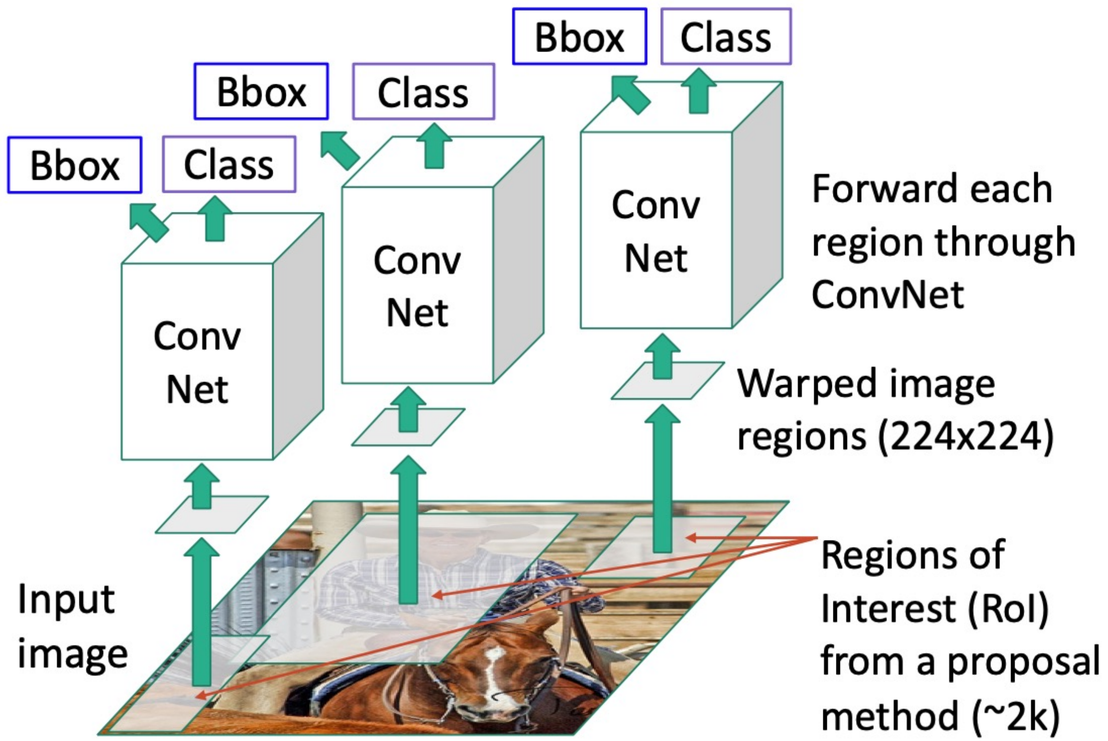
(1) select region proposals

(2) normalize in size

(3) process each region

(4) classify each region

R-CNN: Region-based CNN



(1) select region proposals

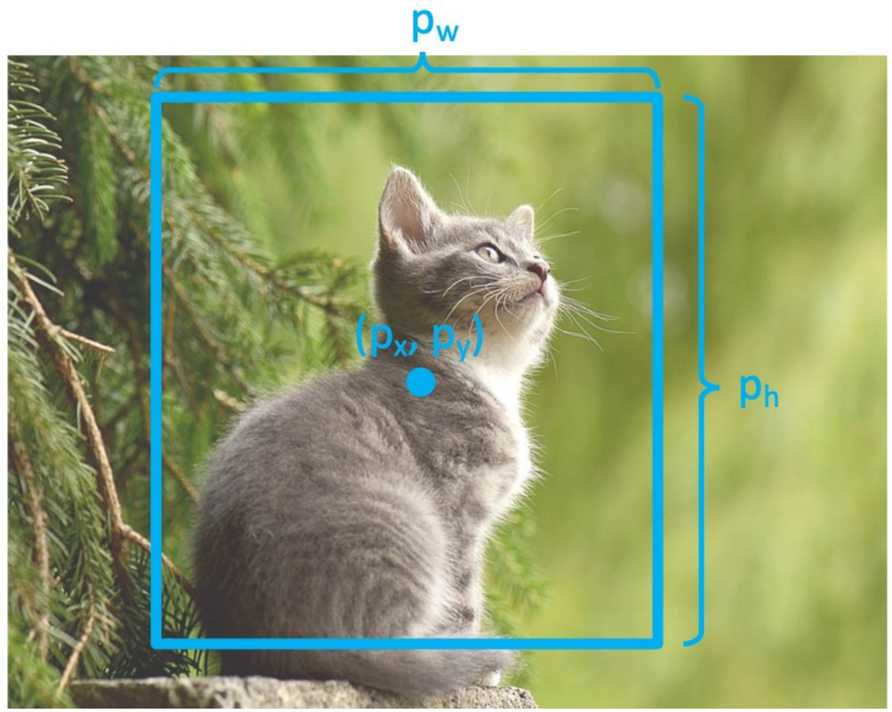
(2) normalize in size

(3) process each region

(4) classify each region

(5) bounding box regression: predict "transform" to correct the ROI: (t_x,t_y,t_h,t_w)

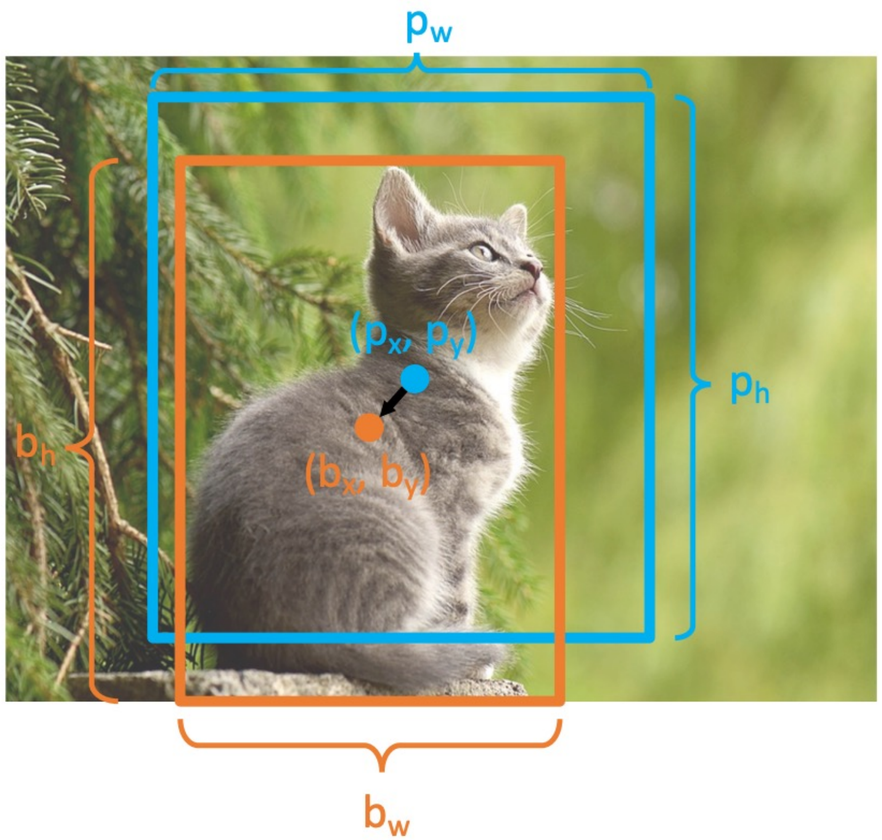
R-CNN: Bounding box regression



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: Bounding 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:

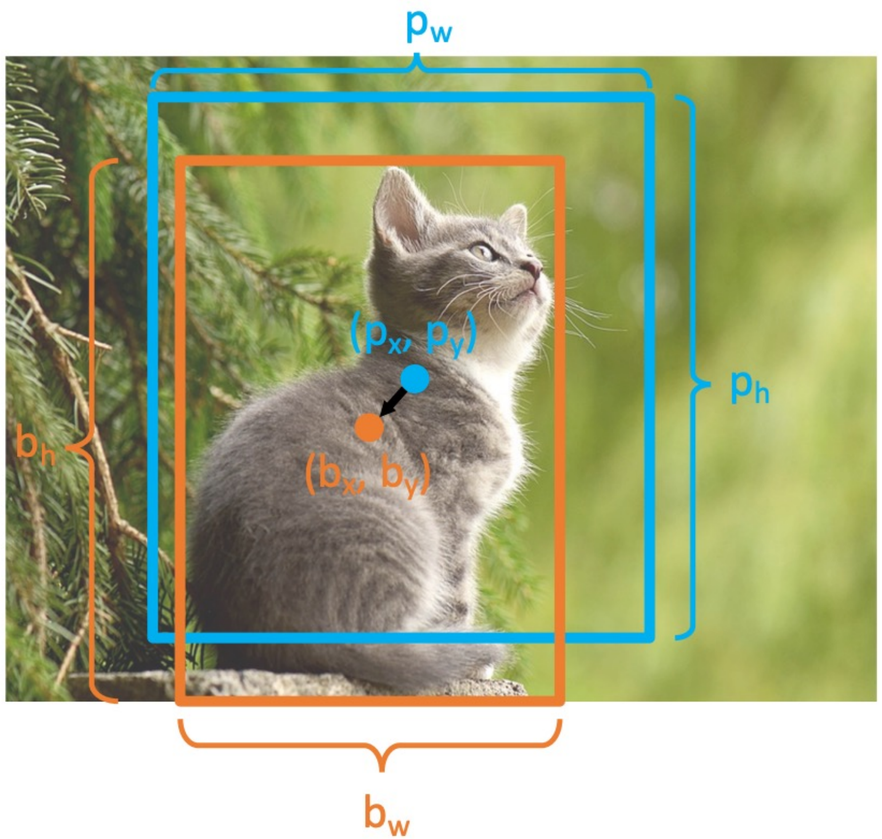
$$b_x = p_x + p_w t_x$$
shift center by amount relative to proposal size

$$b_y = p_y + p_h t_y$$

$$b_w = p_w \exp(t_w)$$
scale proposal; exp ensures that scaling factor is >0

$$b_h = p_h \exp(t_h)$$

R-CNN: Bounding 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

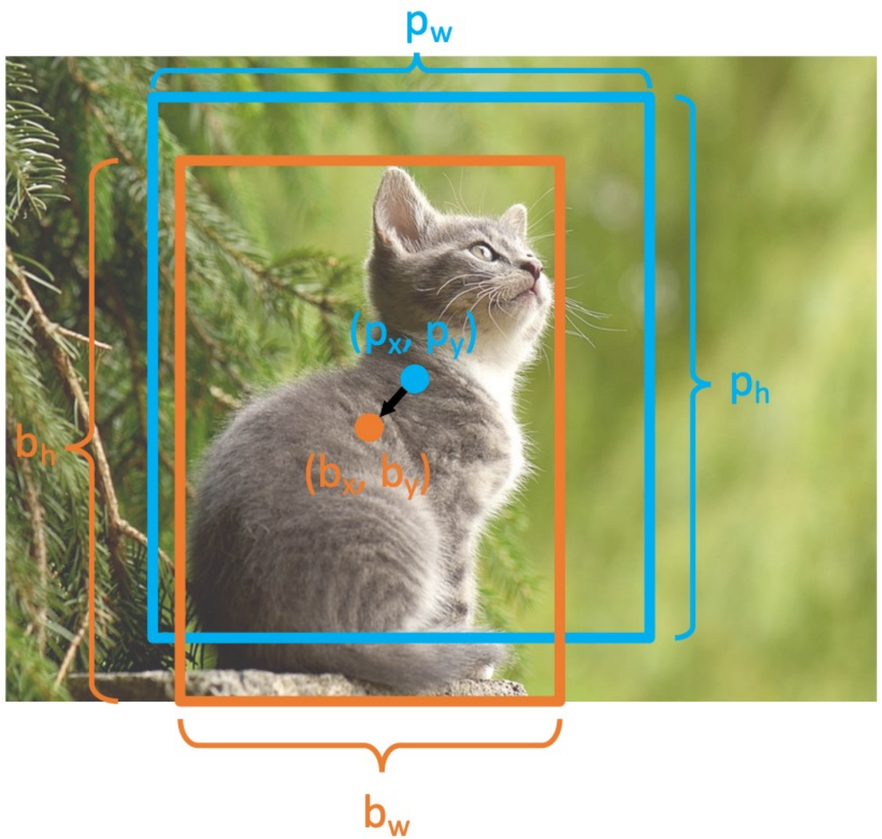
The output box is defined by:

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

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: Bounding box regression



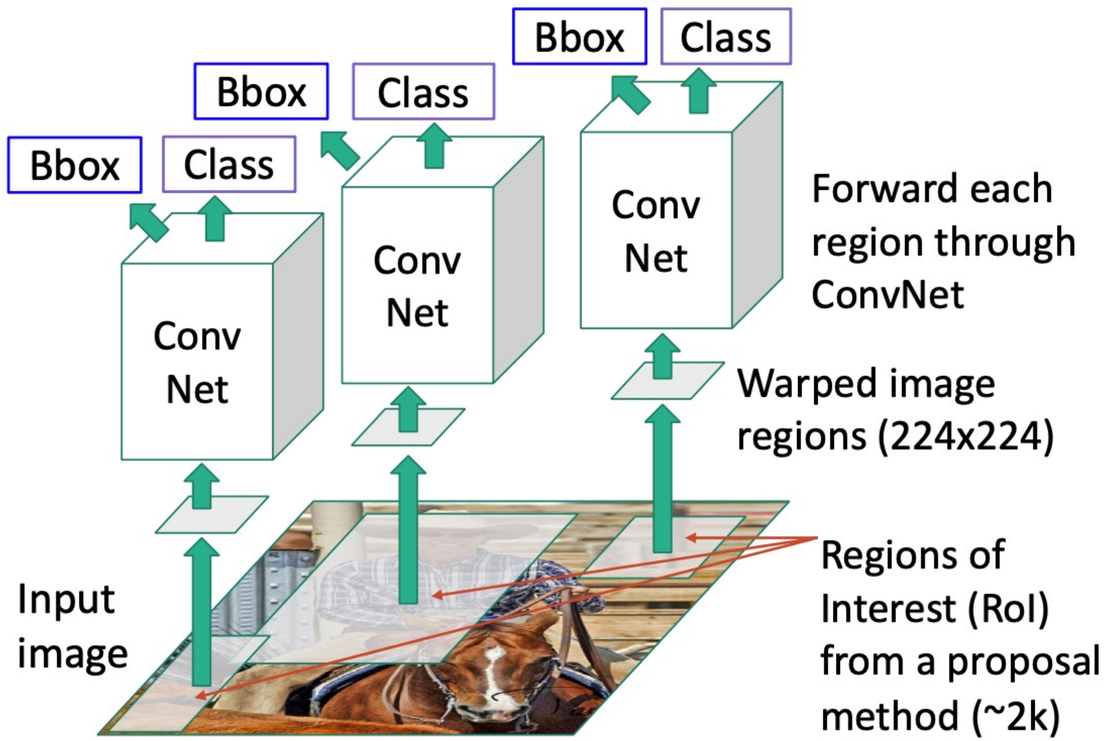
The output box is defined by:

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

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: Region-based CNN



(1) select region proposals

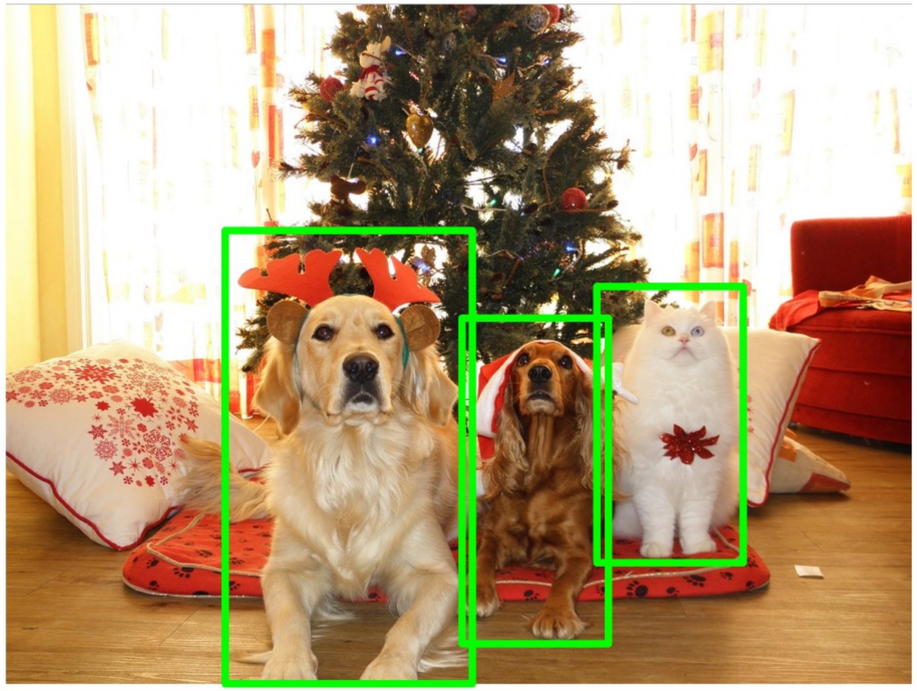
(2) normalize in size

(3) process each region

(4) classify each region

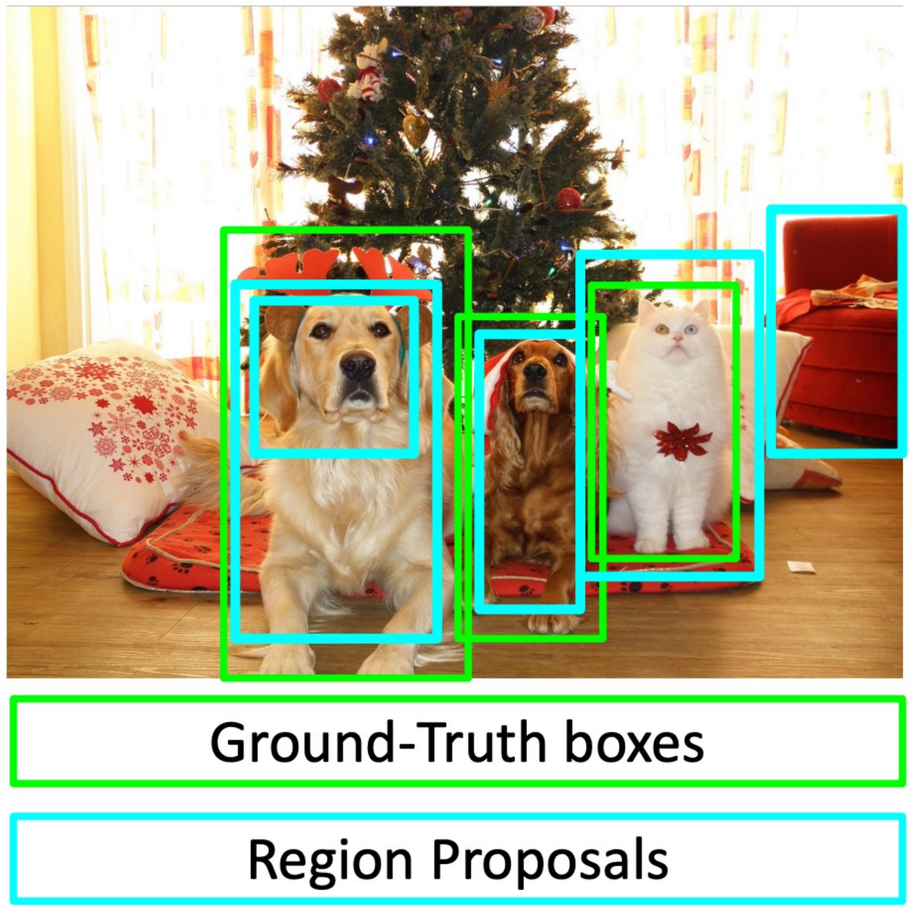
(5) bounding box regression: predict "transform" to correct the ROI: (t_x,t_y,t_h,t_w)

R-CNN training

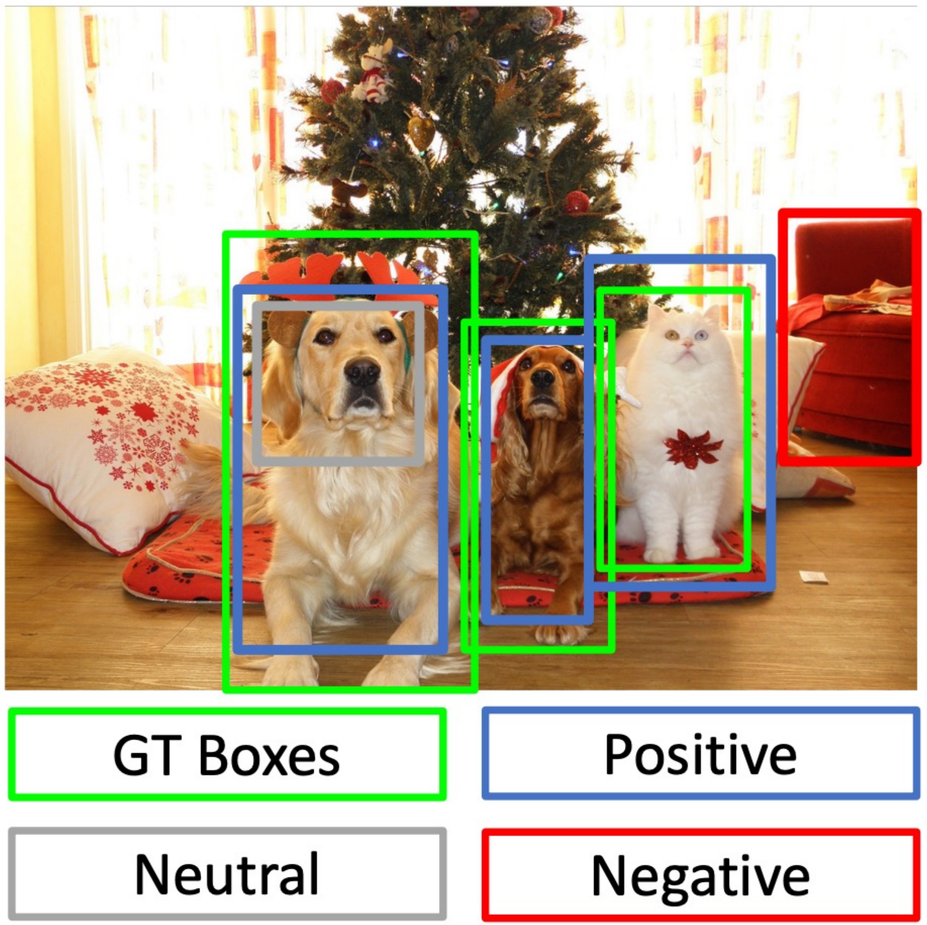


Ground-Truth boxes

R-CNN training



R-CNN training



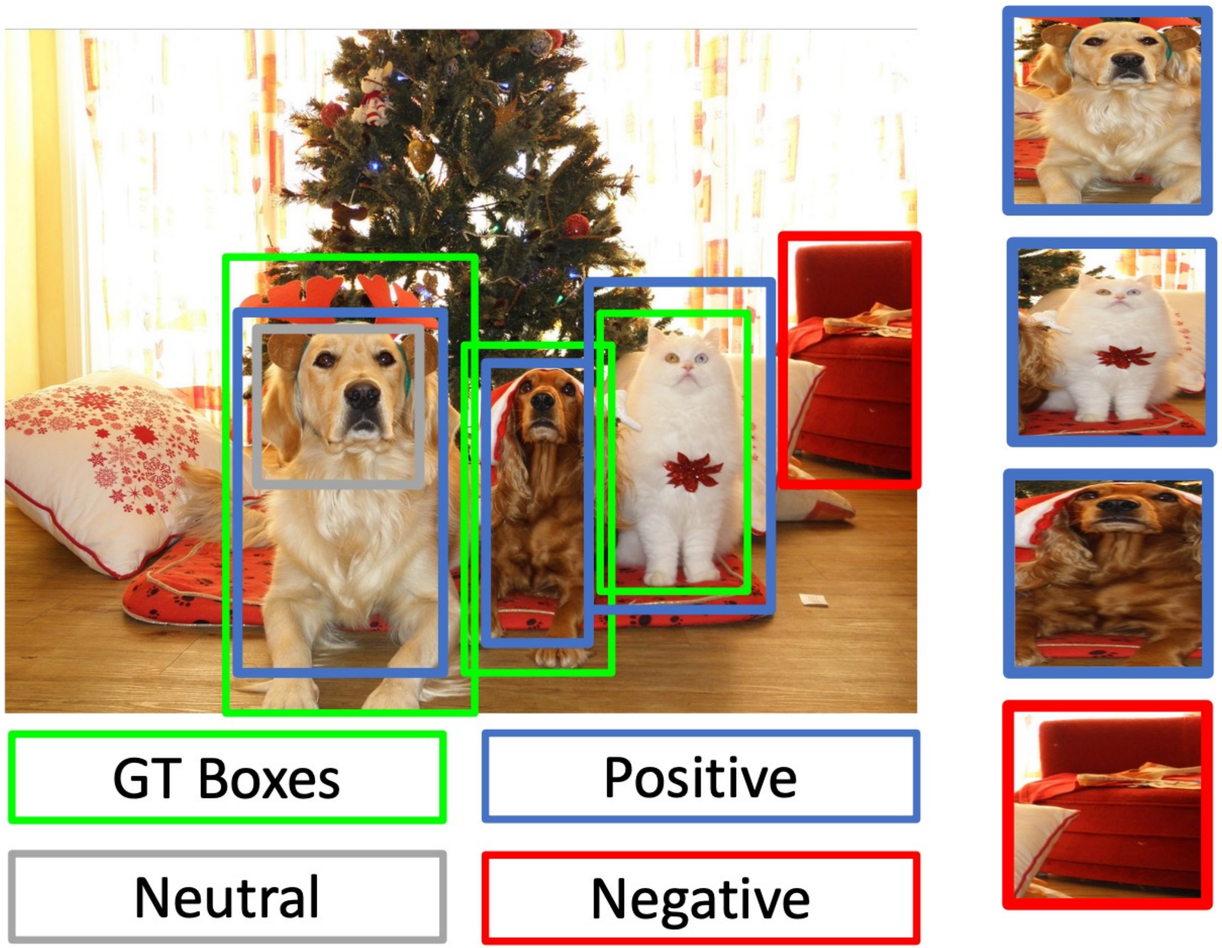
Categorize each region proposal as positive, negative, or neutral based on overlap with ground-truth boxes:

positive: > 0.5 IoU with GT box

negative: < 0.3 IoU with all GT boxes

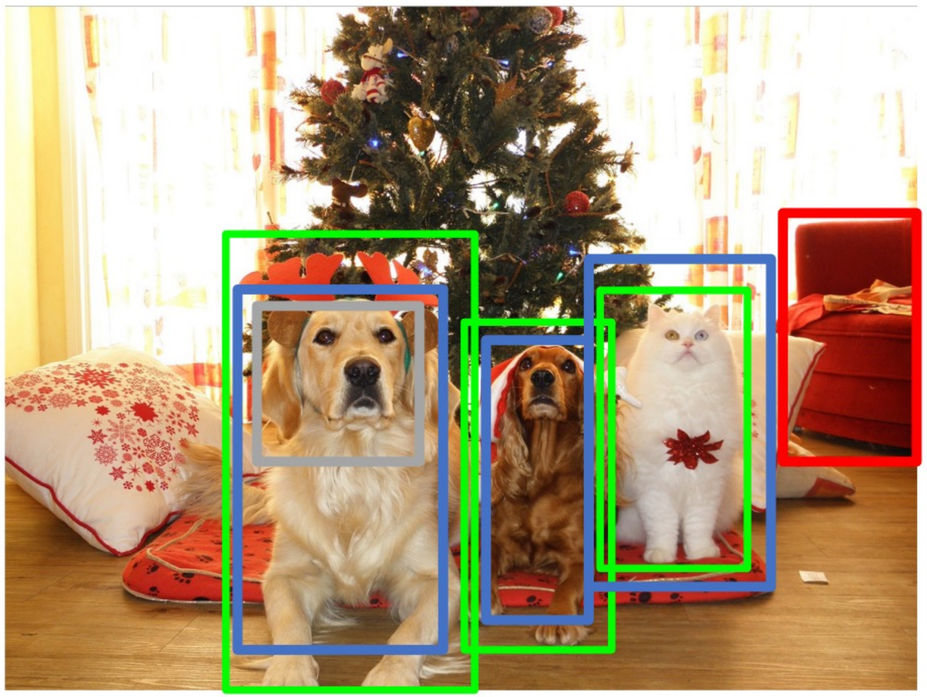
neutral: between 0.3 and 0.5 IoU with GT boxes

R-CNN training



Crop pixels from each positive and negative proposal, resize to 224x224

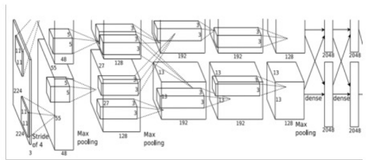
R-CNN training



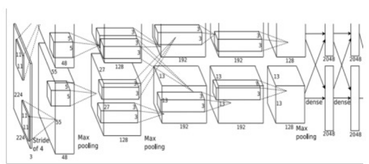
GT Boxes | Positive

Neutral | Negative

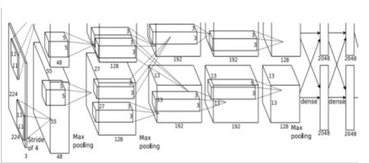
Run each region through CNN
Positive regions: predict class and transform
Negative regions: just predict class



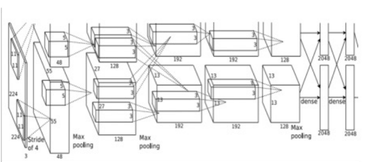
class target: dog
box target:



class target: cat
box target:

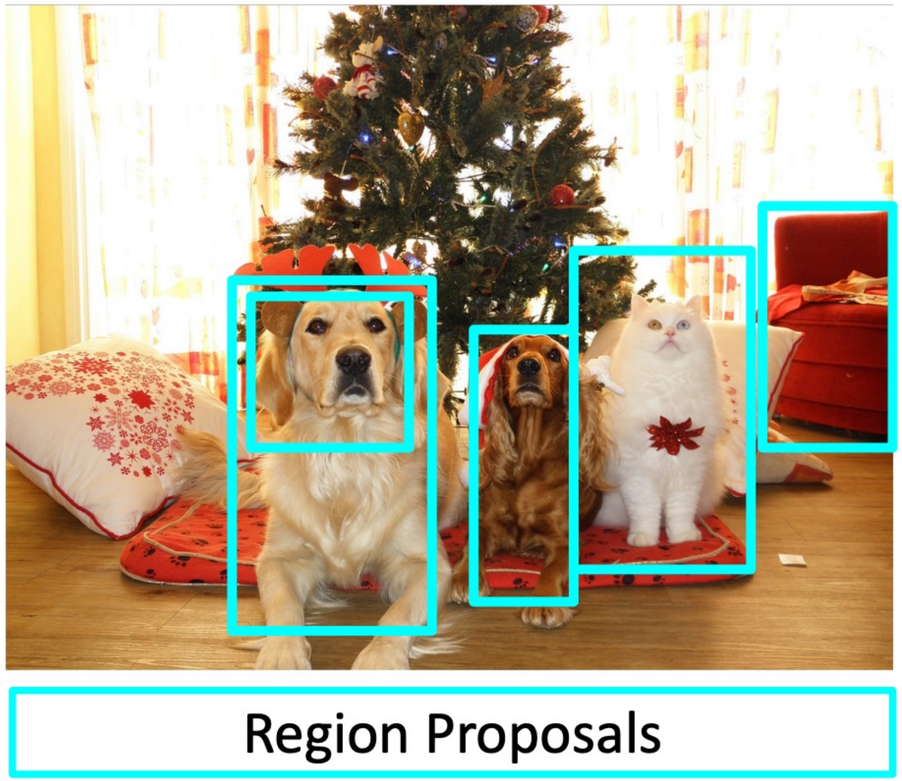


class target: dog
box target:



class target: background
box target: none

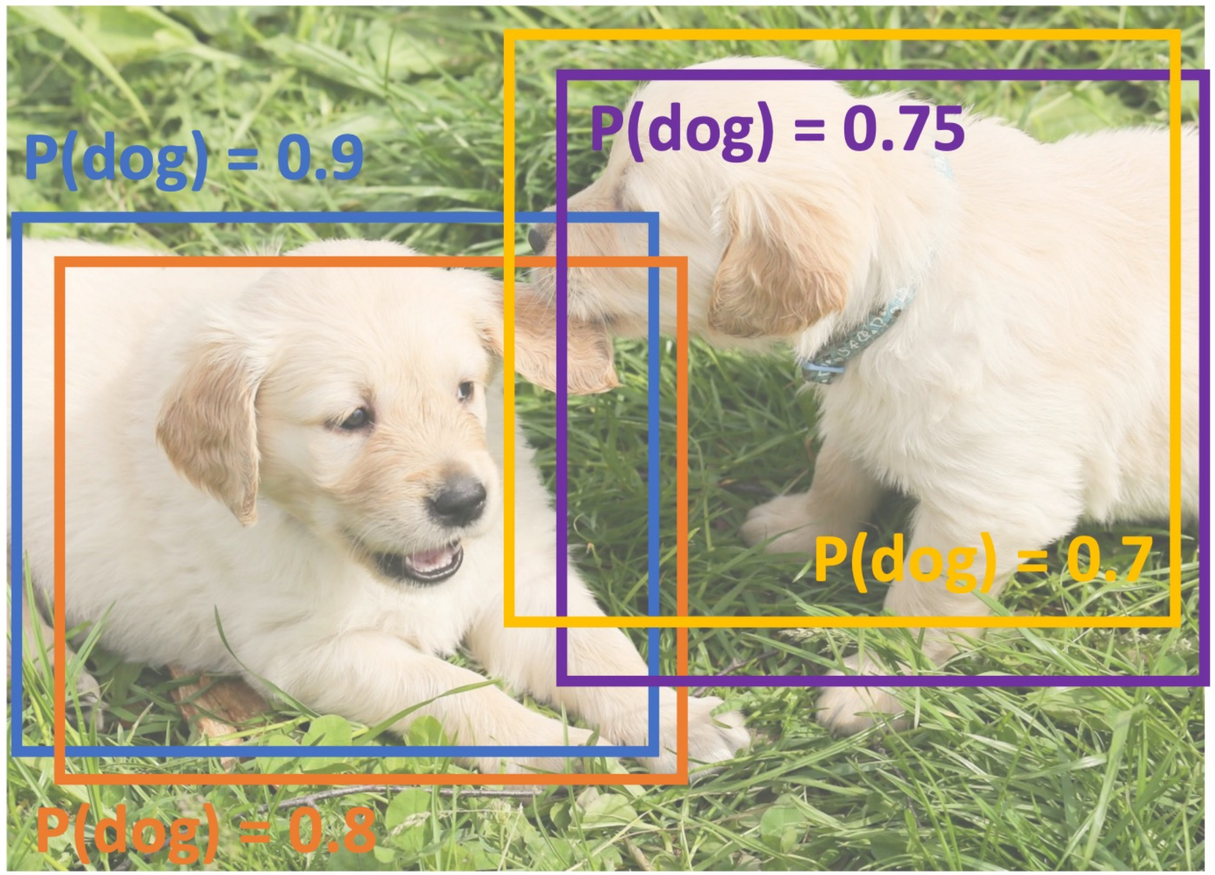
R-CNN testing



(1) Run proposal method
(2) Run CNN for each proposal to get class scores, transforms
(3) Threshold class scores to get a set of detections

2 problems:
- CNN often outputs overlapping boxes
- How to set thresholds?

Overlapping boxes



Problem: Object detectors often output many overlapping detections

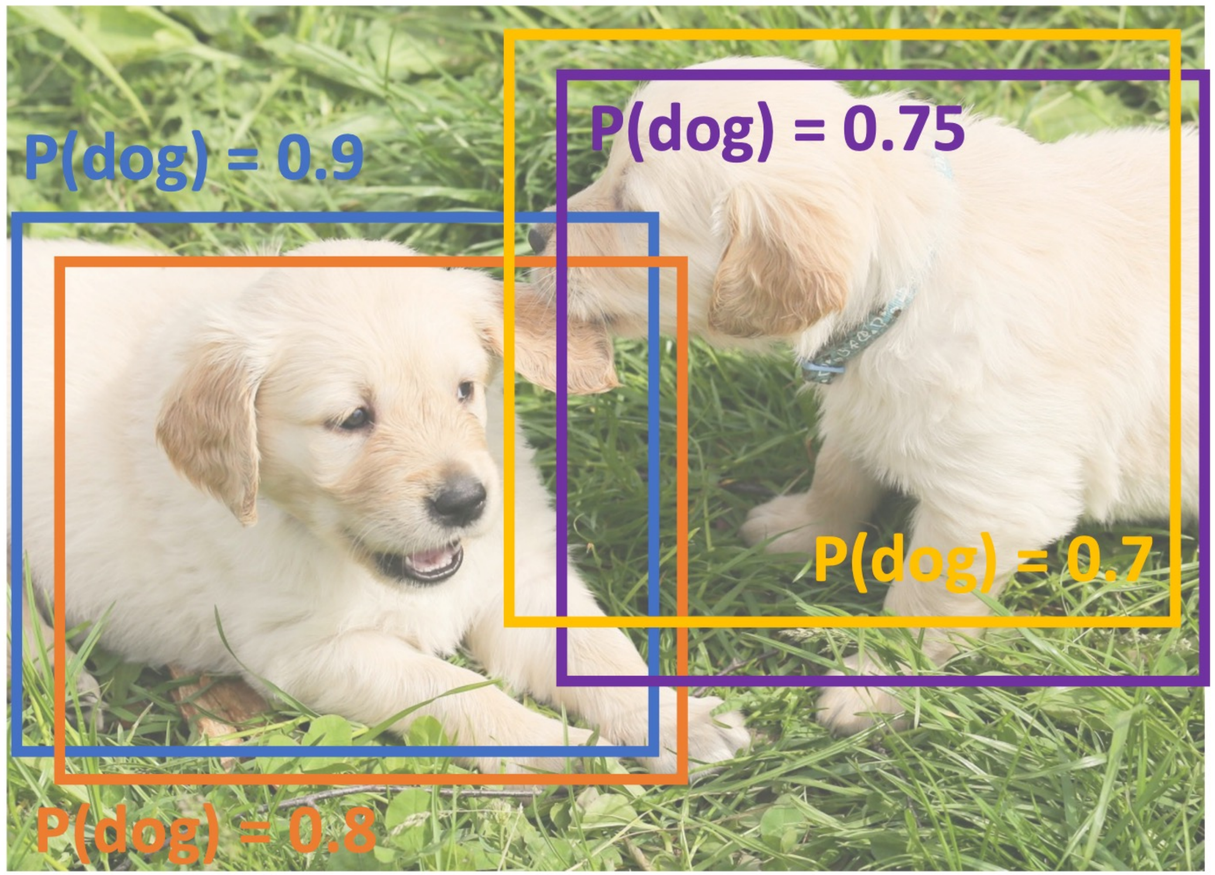
Solution: post-process raw detections using **Non-Max Suppression (NMS)**

(1) select highest-scoring box

(2) eliminate lower-scoring boxes with IoU > threshold (e.g. 0.7)

(3) If any boxes remain, GOTO 1

Overlapping boxes



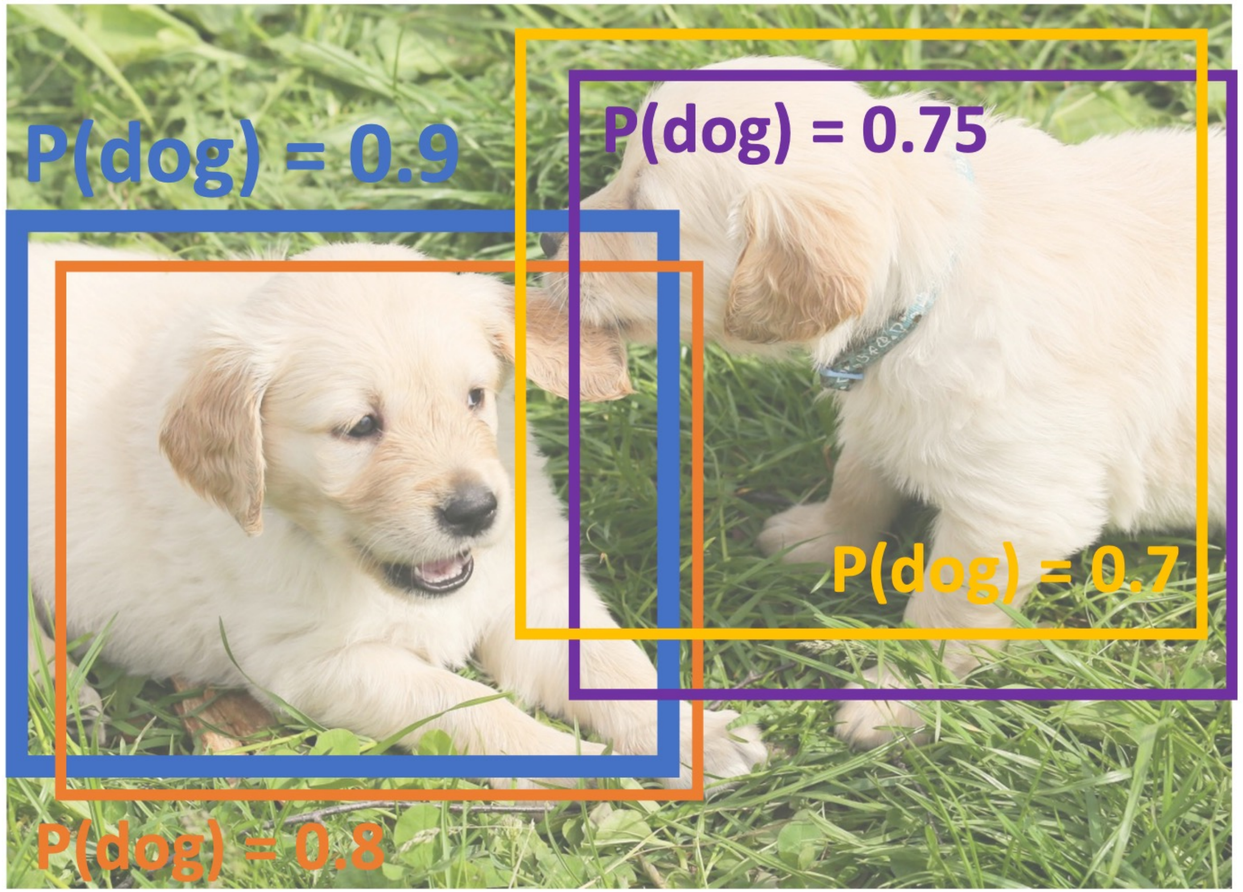
(1) select highest-scoring box

(2) eliminate lower-scoring boxes with IoU > threshold (e.g. 0.7)

(3) If any boxes remain, GOTO 1

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

Overlapping boxes



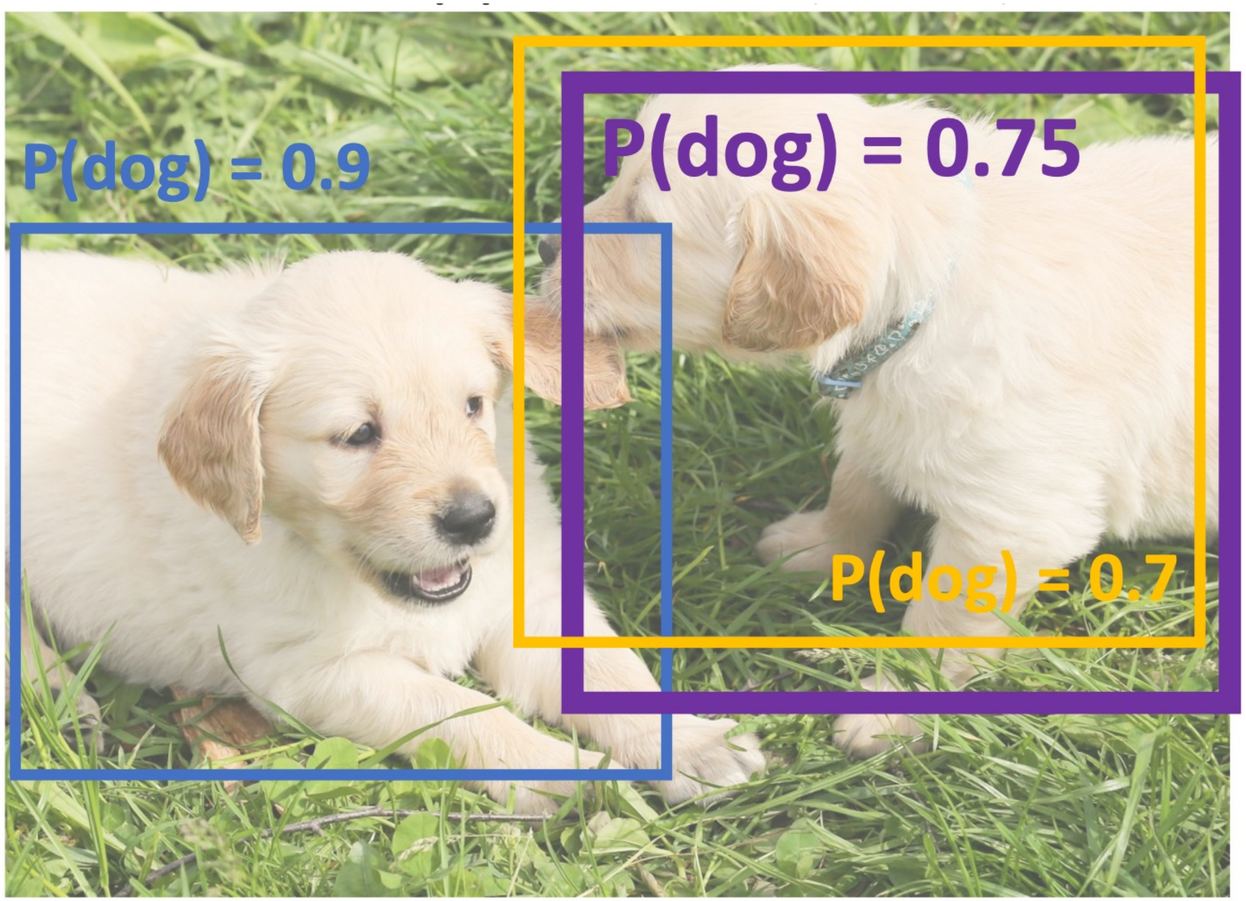
(1) select highest-scoring box

(2) eliminate lower-scoring boxes with IoU > threshold (e.g. 0.7)

(3) If any boxes remain, GOTO 1

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

Overlapping boxes



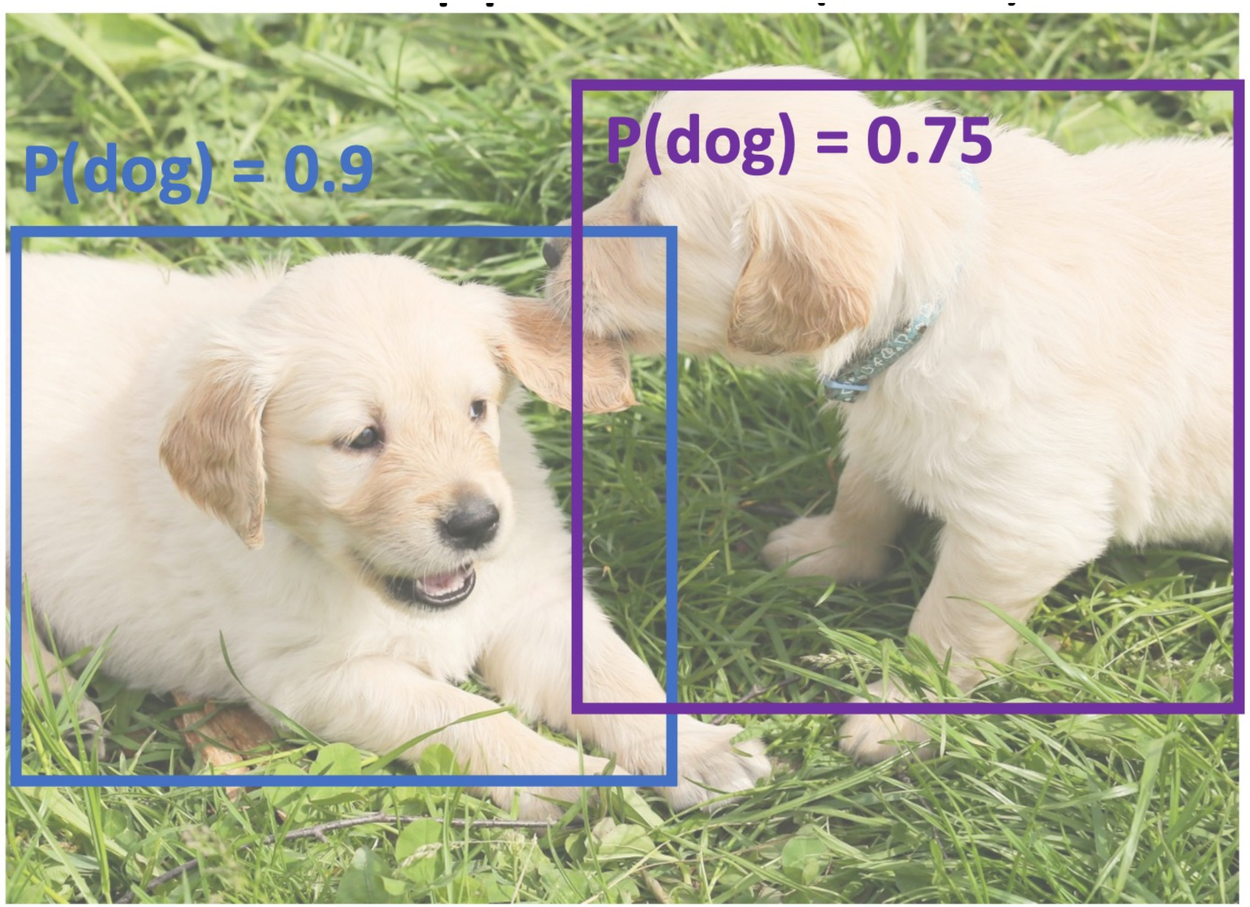
(1) select highest-scoring box

(2) eliminate lower-scoring boxes with IoU > threshold (e.g. 0.7)

(3) If any boxes remain, GOTO 1

IoU(■, ■) = **0.74**

Overlapping boxes



(1) select highest-scoring box

(2) eliminate lower-scoring boxes with IoU > threshold (e.g. 0.7)

(3) If any boxes remain, GOTO 1

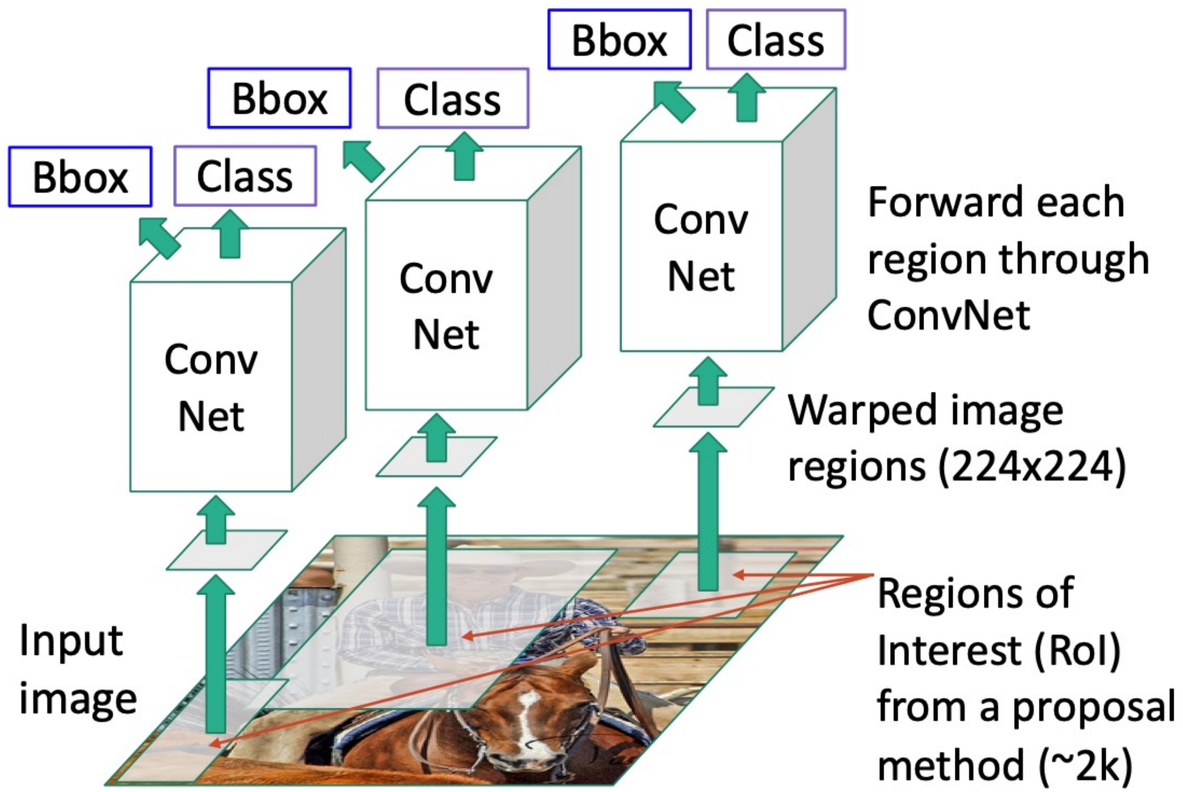
Overlapping boxes

Remaining problem:
NMS may eliminate "good" boxes when objects are highly overlapping – there is still no good solution!

Today's focus

- Principles of multi-object detection
- Region proposals
- Bounding box regression
- R-CNN training and inference
- Non-Max suppression
- Modern object detectors

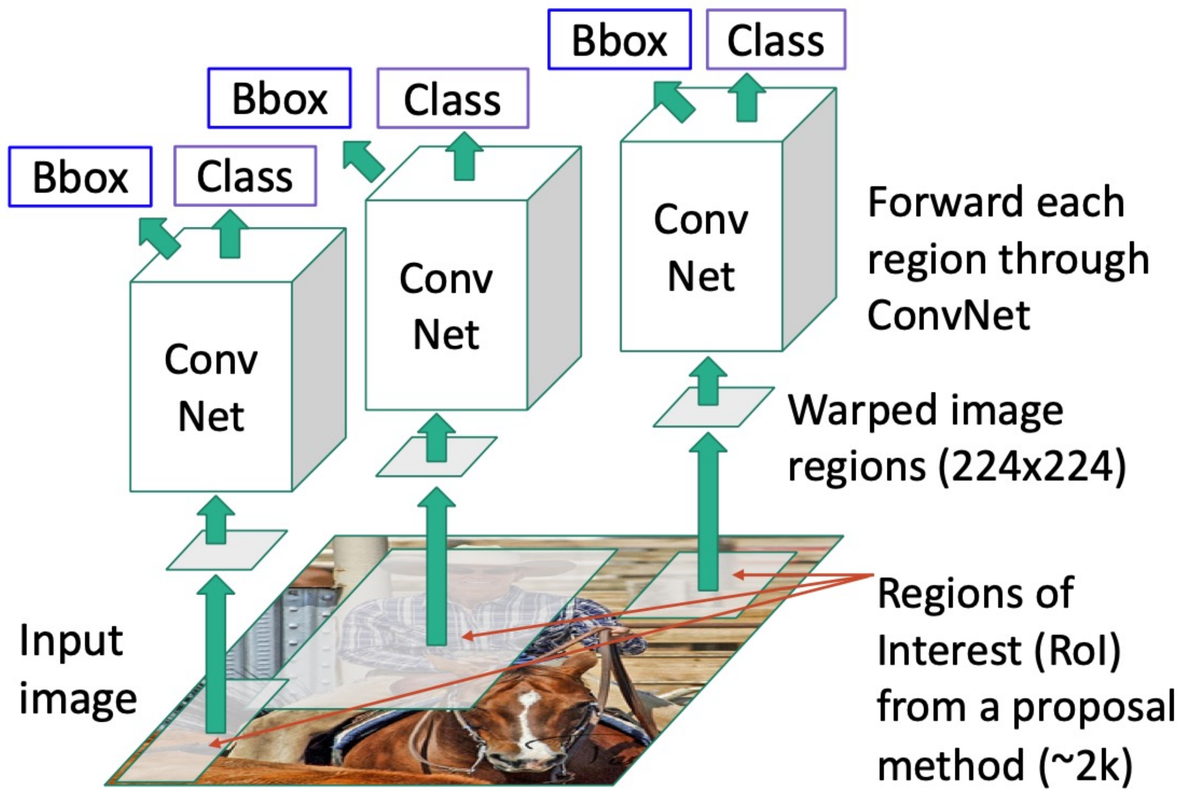
Recap: R-CNN



Problem: Very slow!

Need to do 2000 forward passes through CNN per image

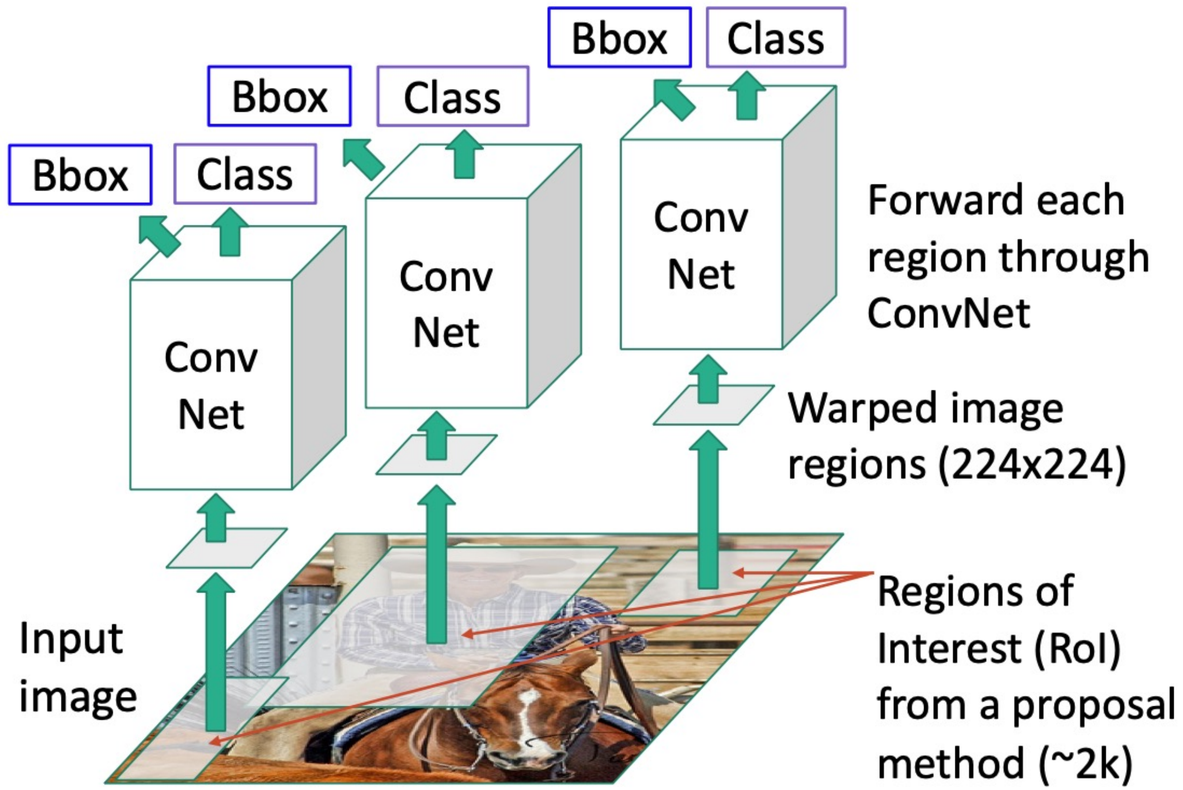
Recap: R-CNN



Problem: Very slow!

Need to do 2000 forward passes through CNN per image

Recap: R-CNN

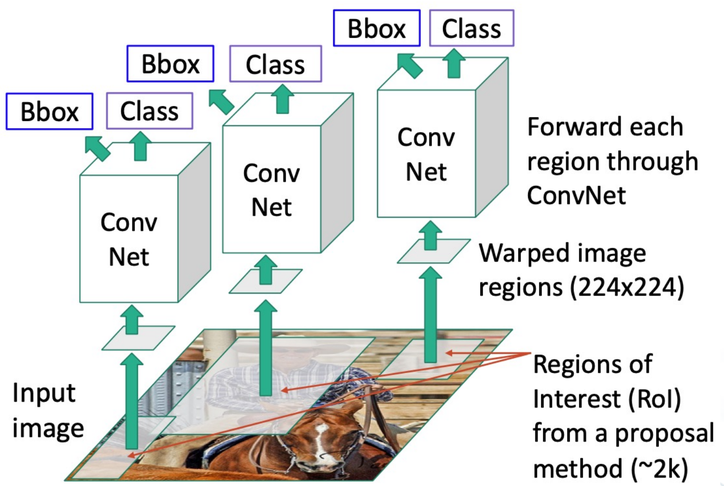


Problem: Very slow!

Need to do 2000 forward passes through CNN per image

Idea: Overlapping proposals cause a lot of repeated work – same pixels processed many times. Can we avoid this?

Fast R-CNN



R. Girshick, “Fast R-CNN”, ICCV 2015

Fast R-CNN

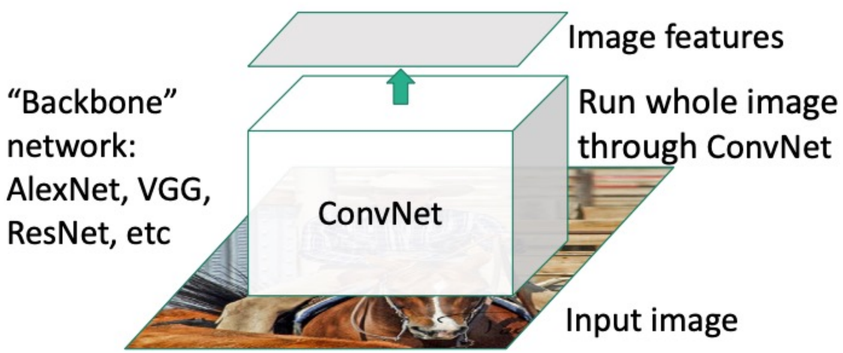
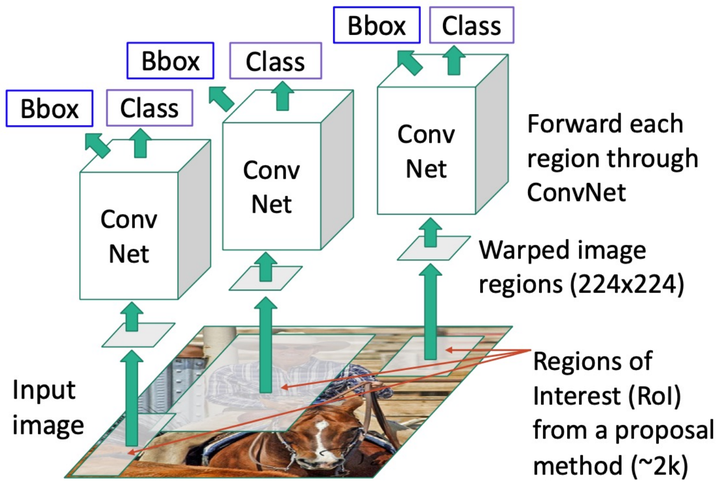
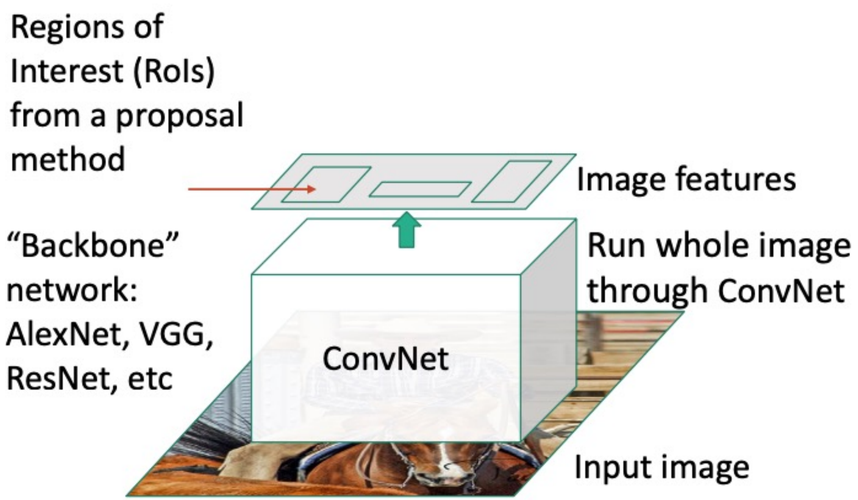
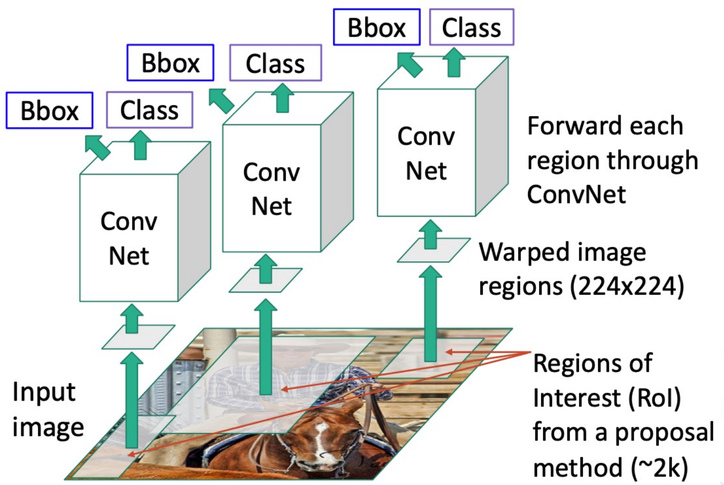


Image features
"Backbone"
network:
AlexNet, VGG,
ResNet, etc
Run whole image
through ConvNet
ConvNet
Input image

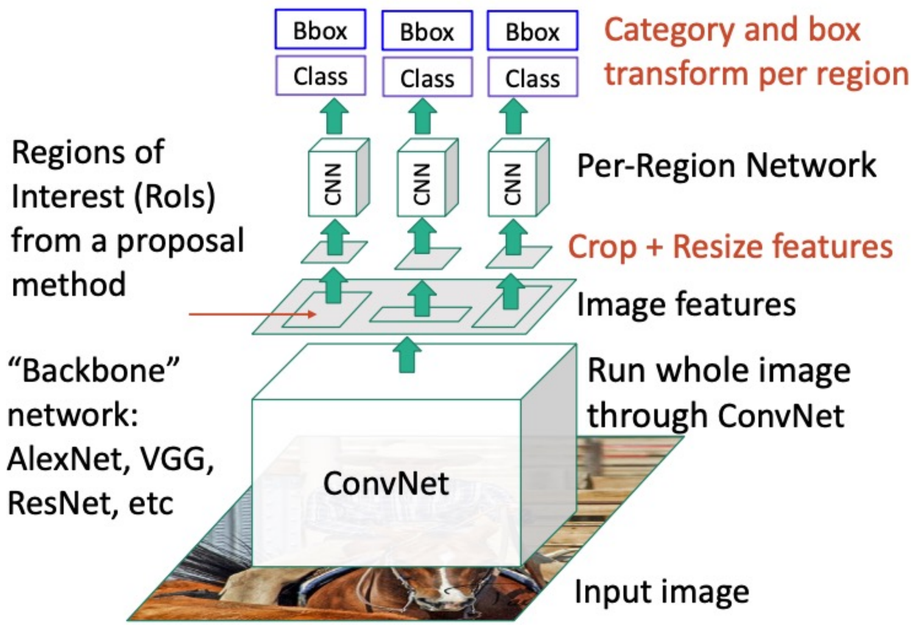

Bbox
Class
Bbox
Class
Bbox
Class
Conv
Net
Conv
Net
Conv
Net
Forward each
region through
ConvNet
Warped image
regions (224x224)
Input
image
Regions of
Interest (RoI)
from a proposal
method (~2k)

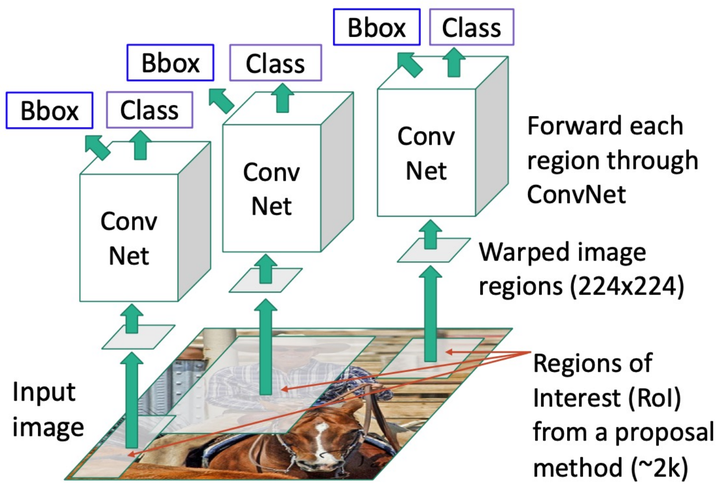
Fast R-CNN


Regions of Interest (RoIs) from a proposal method
Image features
"Backbone" network: AlexNet, VGG, ResNet, etc
Run whole image through ConvNet
ConvNet
Input image

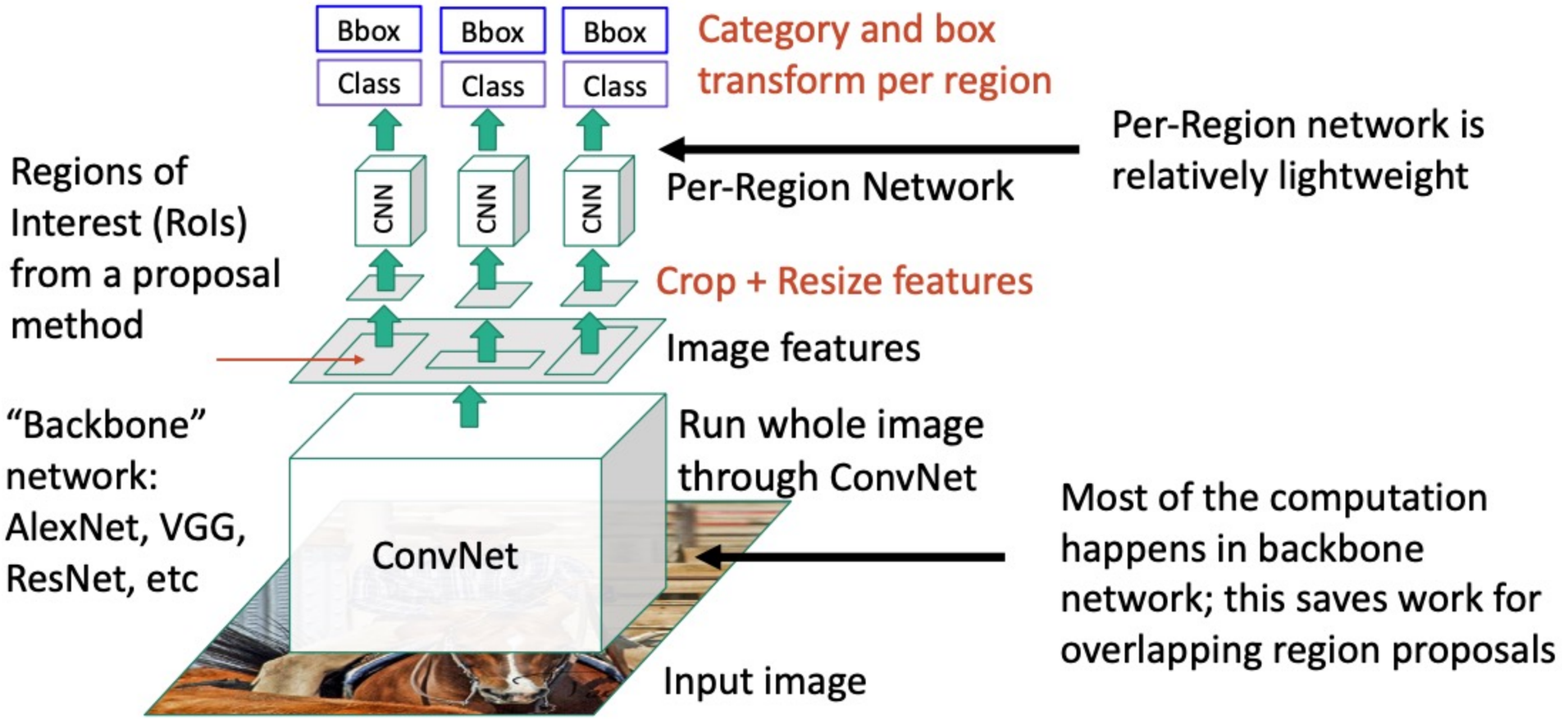

Bbox
Class
Conv Net
Forward each region through ConvNet
Warped image regions (224x224)
Input image
Regions of Interest (RoI) from a proposal method (~2k)

Fast R-CNN

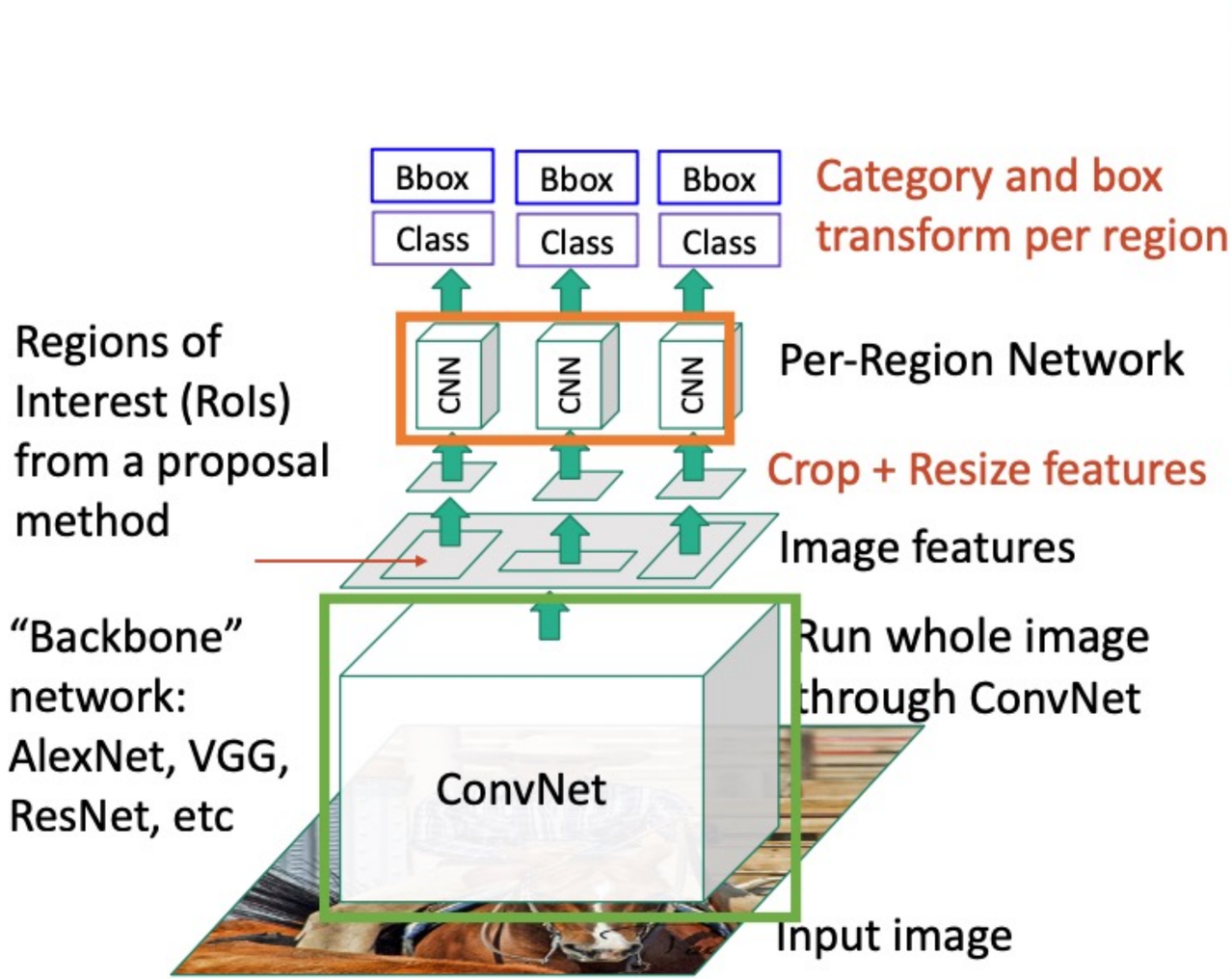

Bbox
Class
Bbox
Class
Bbox
Class
Category and box transform per region
Regions of Interest (RoIs) from a proposal method
CNN
CNN
CNN
Per-Region Network
Crop + Resize features
Image features
"Backbone" network: AlexNet, VGG, ResNet, etc
ConvNet
Run whole image through ConvNet
Input image

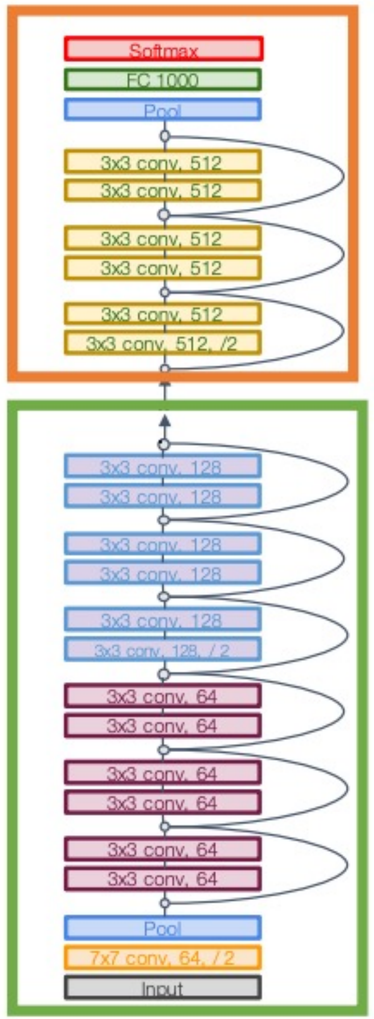

Bbox
Class
Bbox
Class
Bbox
Class
Conv Net
Conv Net
Conv Net
Forward each region through ConvNet
Warped image regions (224x224)
Input image
Regions of Interest (RoI) from a proposal method (~2k)

Fast R-CNN


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

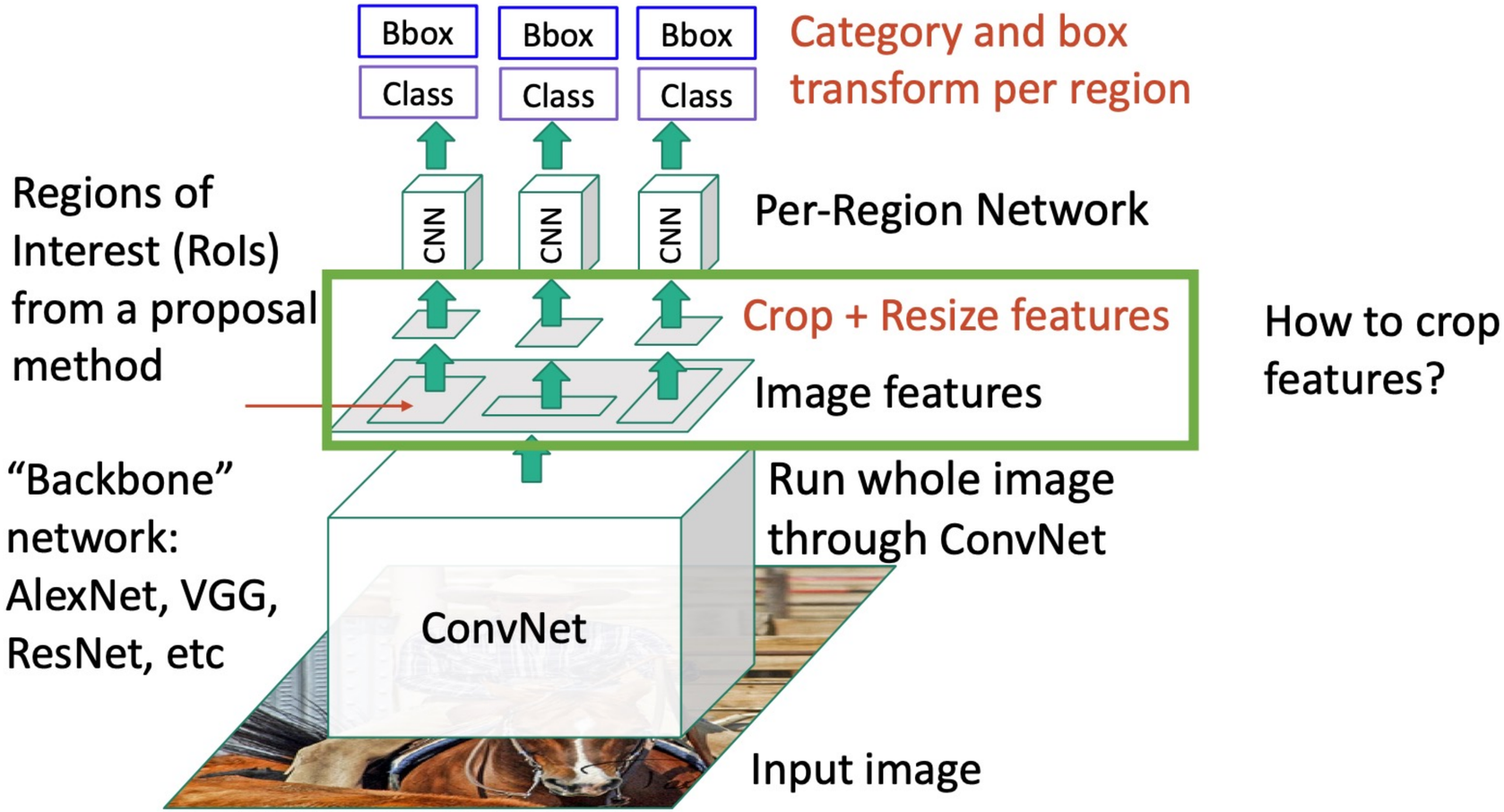
Fast R-CNN



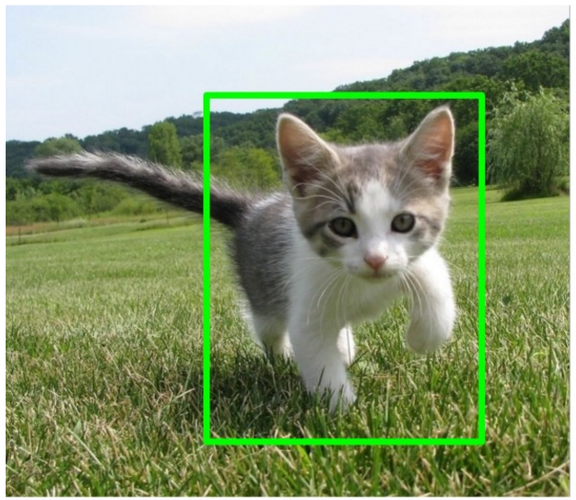


Example:
For ResNet, last stage is used as per-region network; the rest of the network is used as backbone

Fast R-CNN

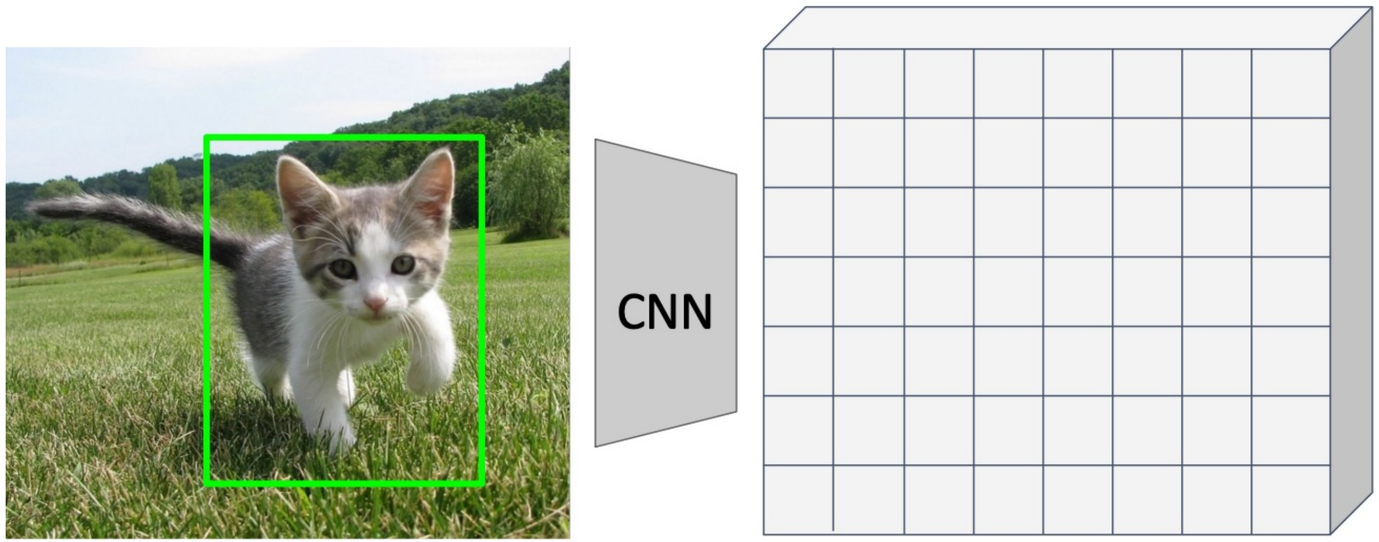

Bbox
Bbox
Bbox
Class
Class
Class
Category and box transform per region
Regions of Interest (RoIs) from a proposal method
CNN
CNN
CNN
Per-Region Network
Crop + Resize features
Image features
How to crop features?
"Backbone" network: AlexNet, VGG, ResNet, etc
Run whole image through ConvNet
ConvNet
Input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

Cropping features: ROI Pool



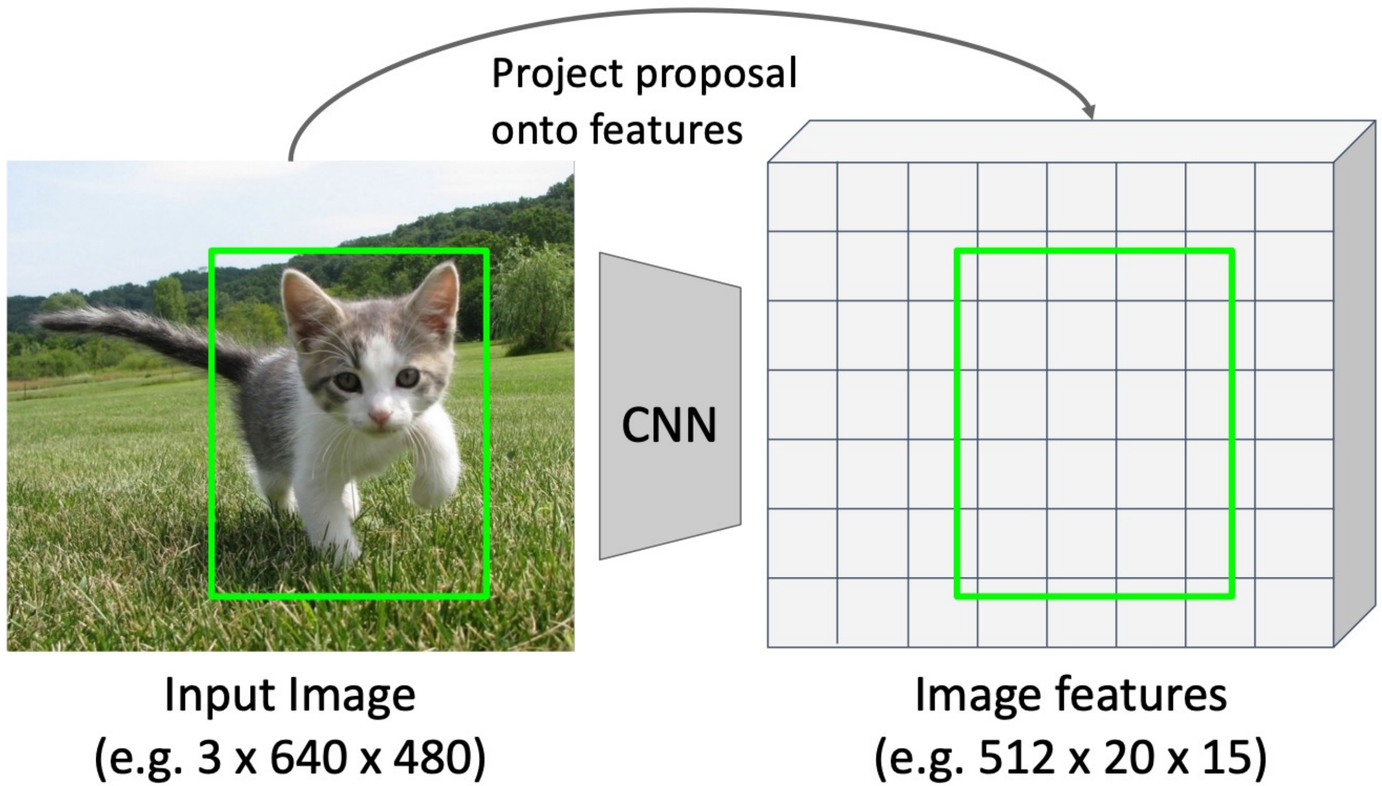
Input Image
(e.g. 3 x 640 x 480)

Cropping features: ROI Pool



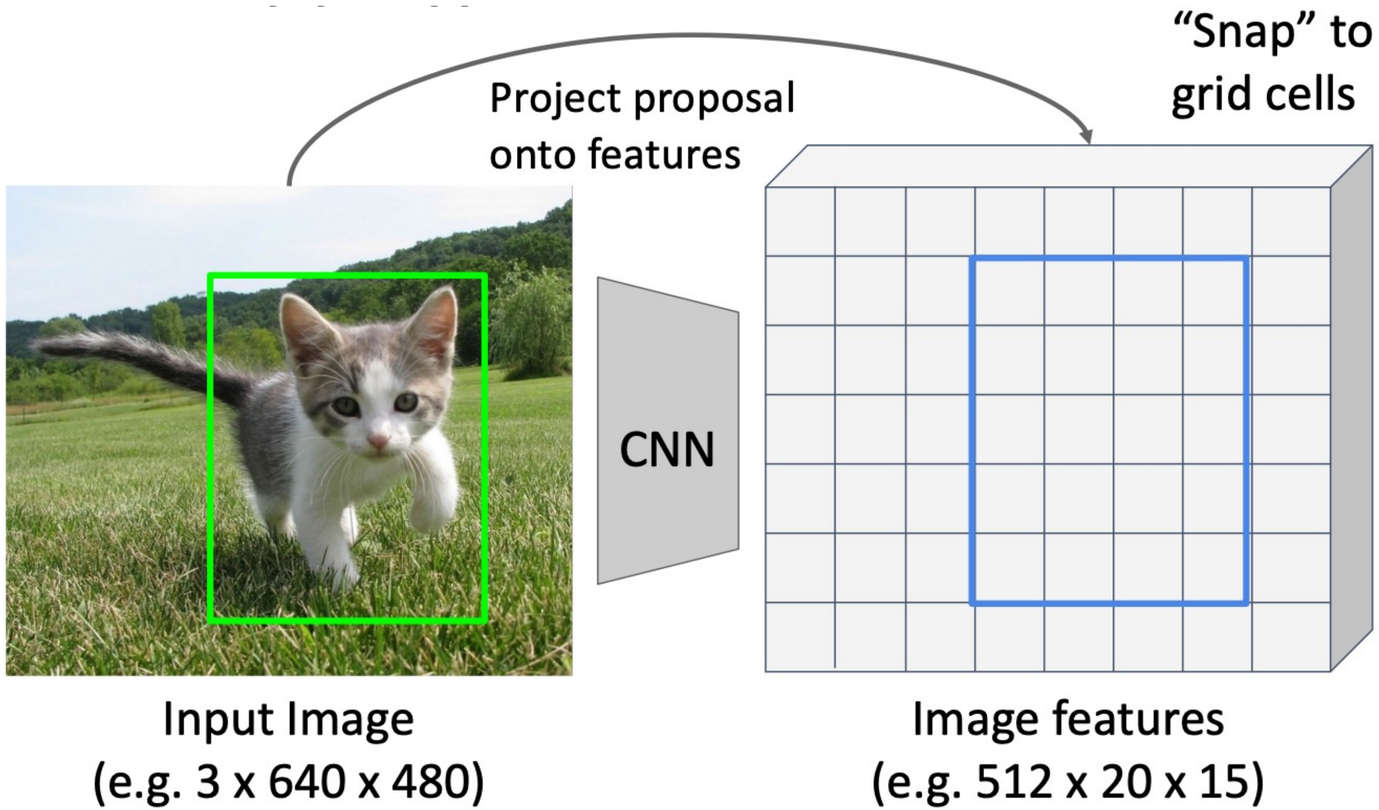
Want features for the box of a fixed size (2x2 in this example, but 7x7 or 14x14 in practice)

Cropping features: ROI Pool



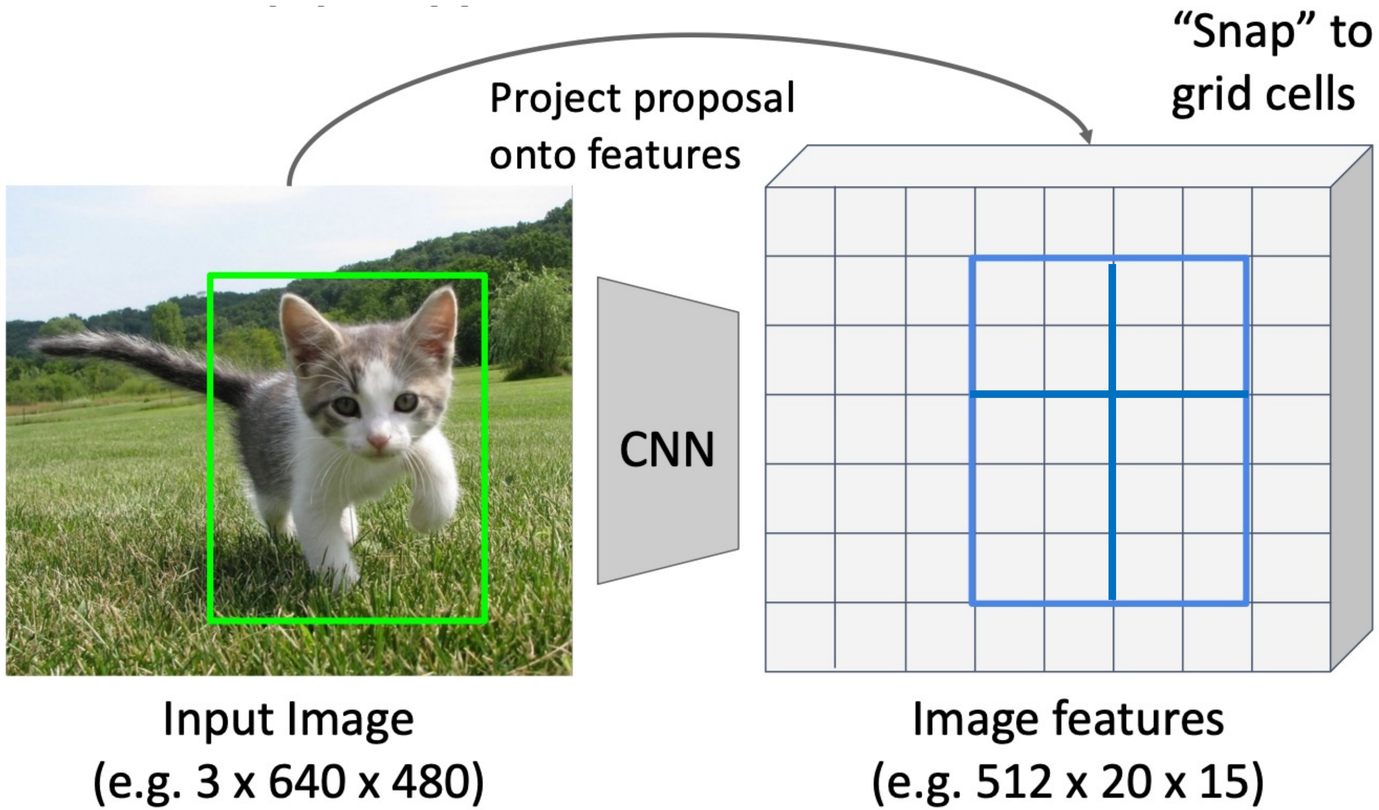
Want features for the box of a fixed size (2x2 in this example, but 7x7 or 14x14 in practice)

Cropping features: ROI Pool



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

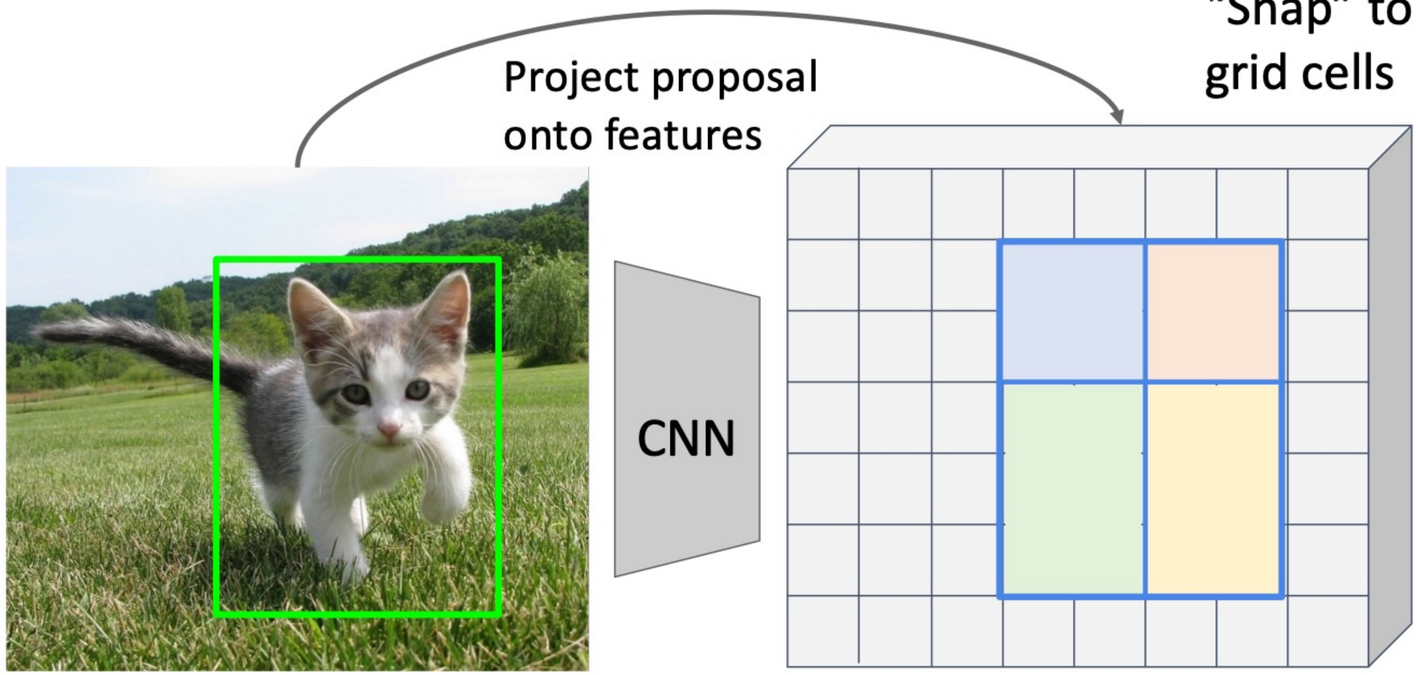
Cropping features: ROI Pool



divide into 2x2 grid of (roughly) equal subregions

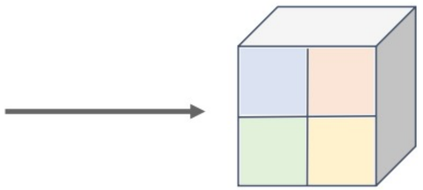
Want features for the box of a fixed size (2x2 in this example, but 7x7 or 14x14 in practice)

Cropping features: ROI Pool



divide into 2x2 grid of (roughly) equal subregions

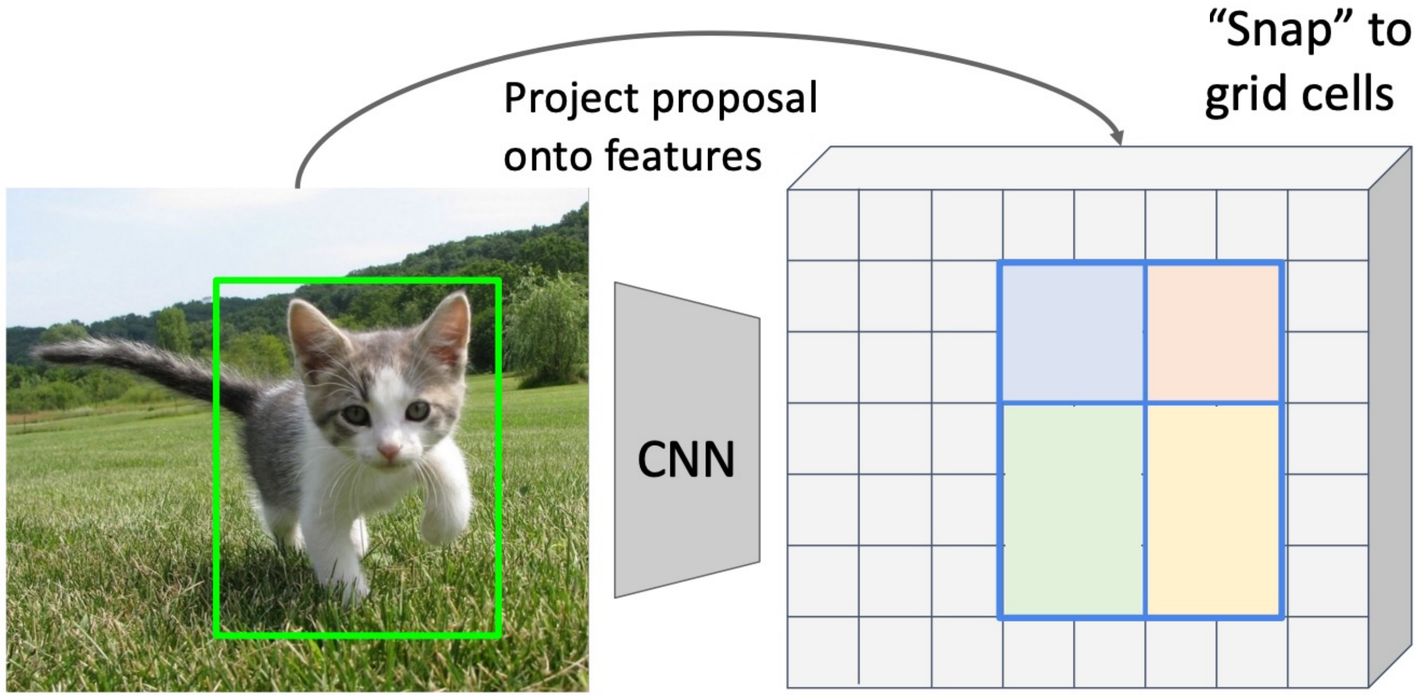
max-pool within each subregion



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

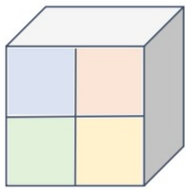
Region features always have the same size even if input regions of different sizes

Cropping features: ROI Pool



divide into 2x2 grid of (roughly) equal subregions

max-pool within each subregion

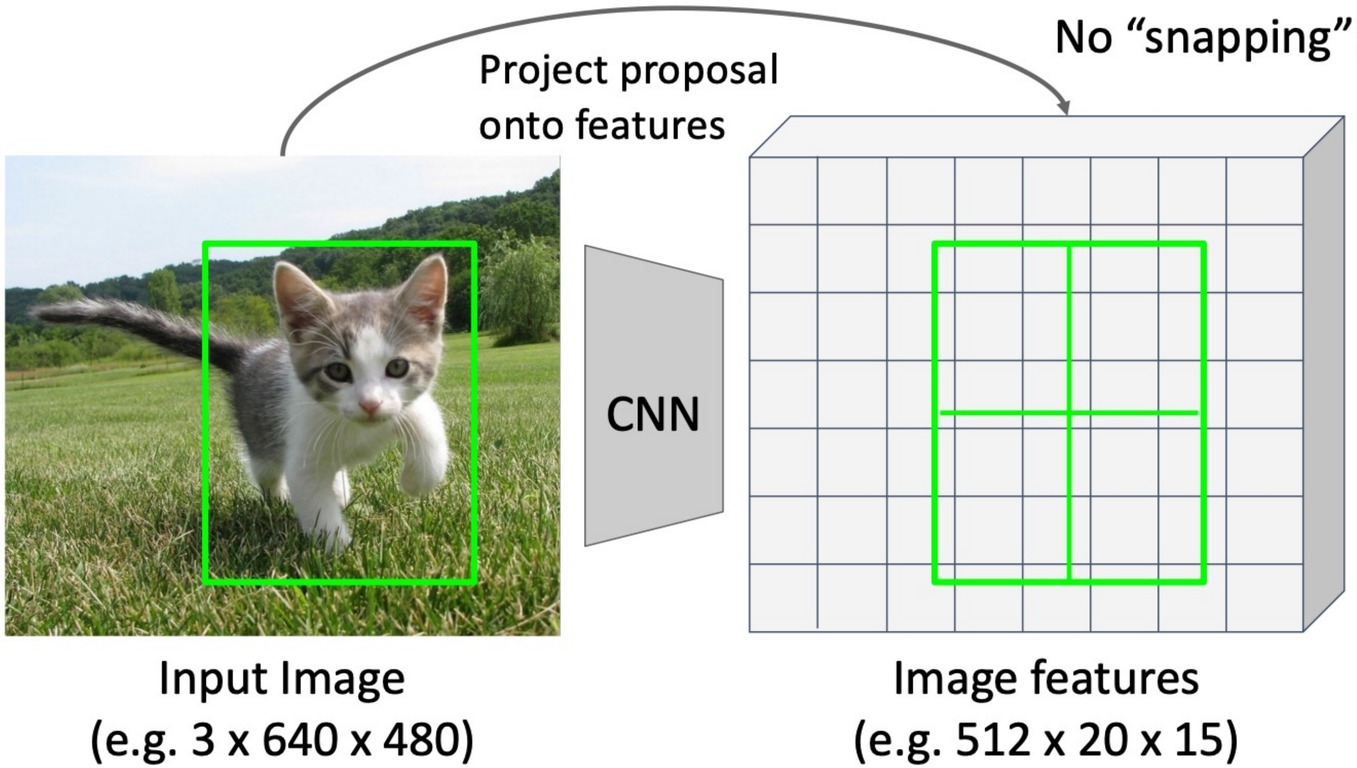


Region features
(here 512 x 2 x 2;
In practice 512x7x7)

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

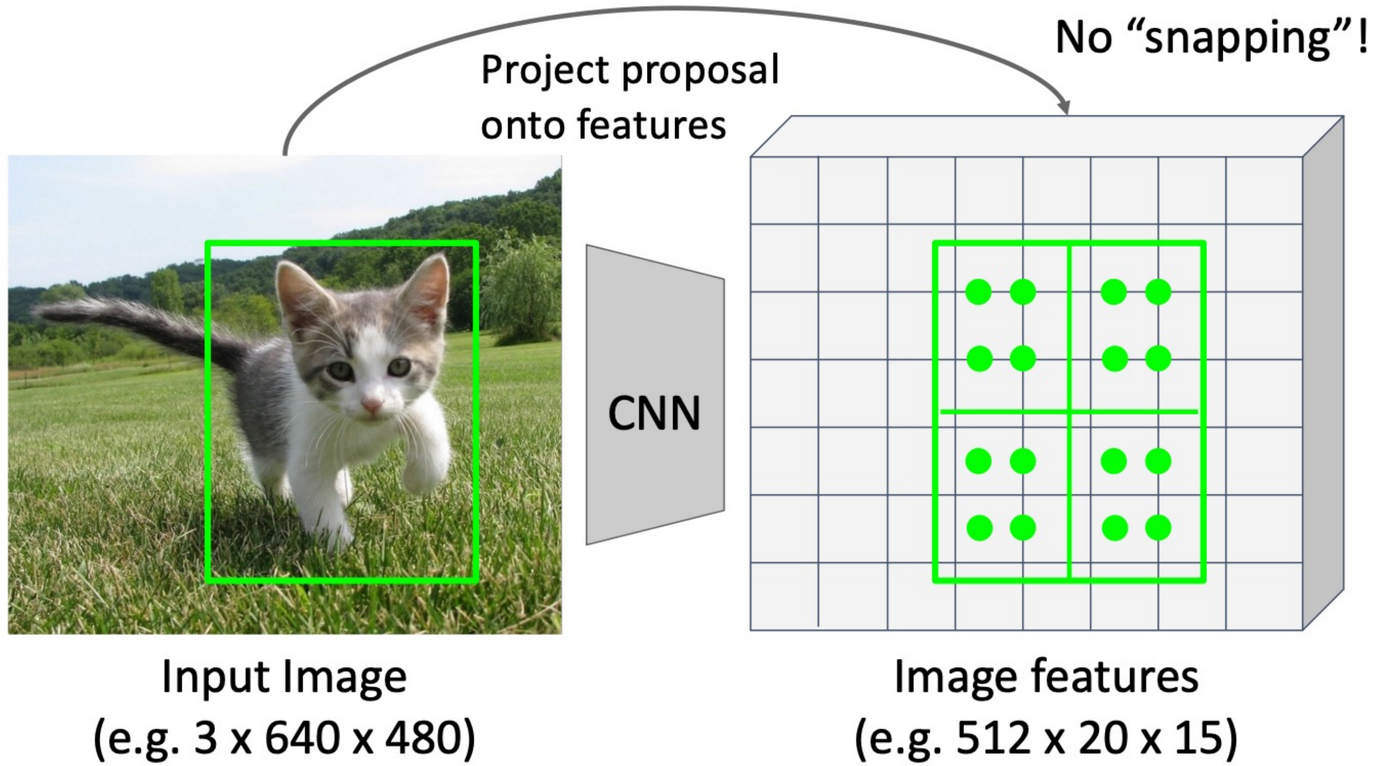
Problem: lots of approximation and misalignment

Cropping features: ROI Align



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

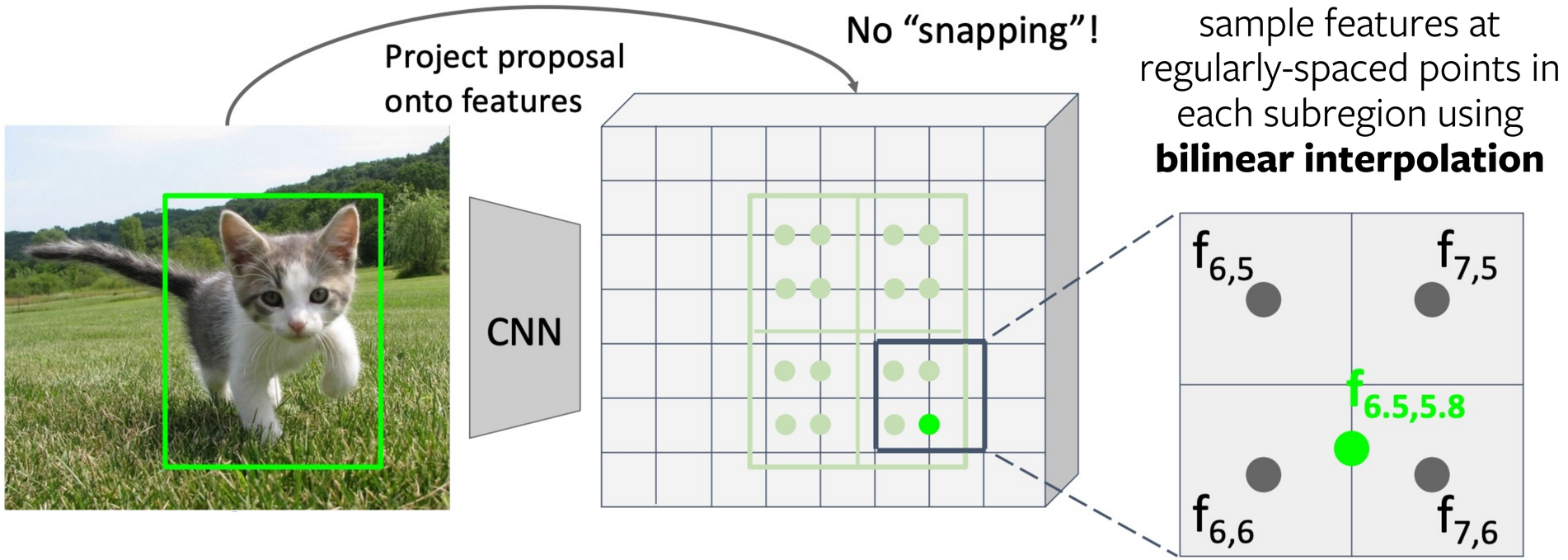
Cropping features: ROI Align



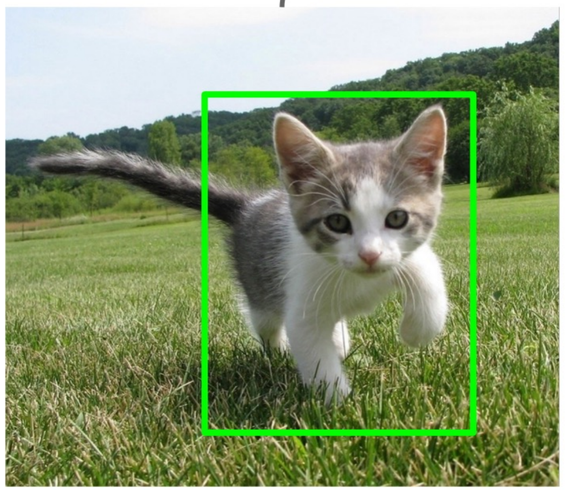
divide into equal-sized subregions (may not be aligned to grid!)

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

Cropping features: ROI Align

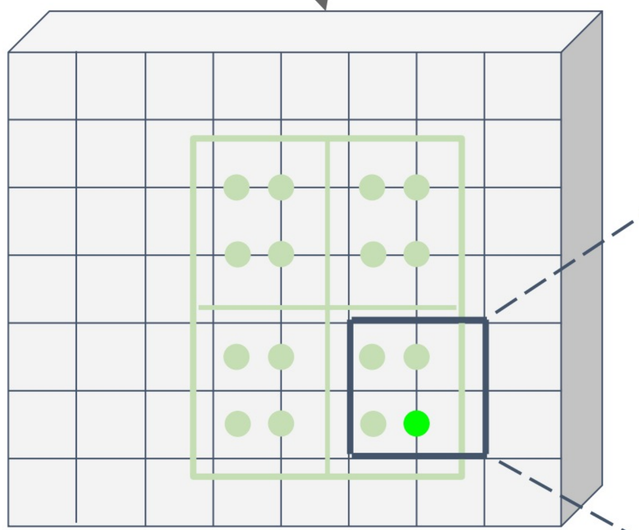


Cropping features: ROI Align

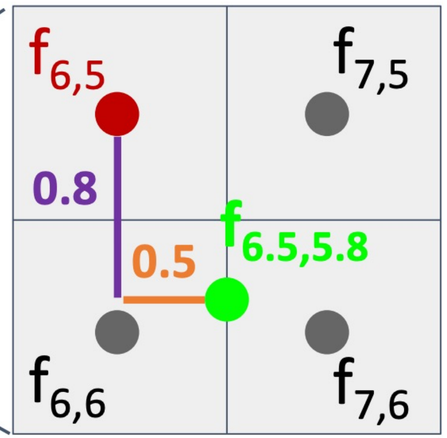


Project proposal onto features

No "snapping"



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

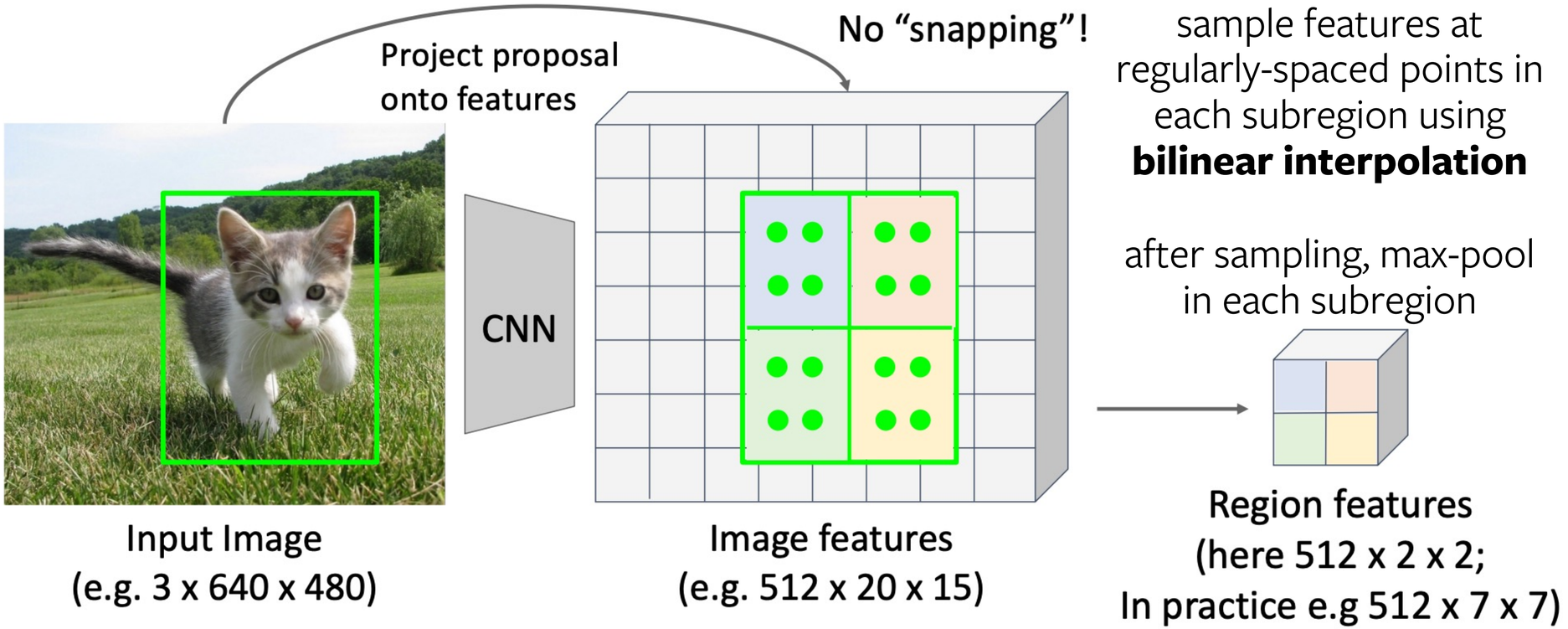


$$f_{xy} = \sum_{i,j} f_{i,j} \max(0, 1 - |x - x_i|) \max(0, 1 - |y - y_i|)$$

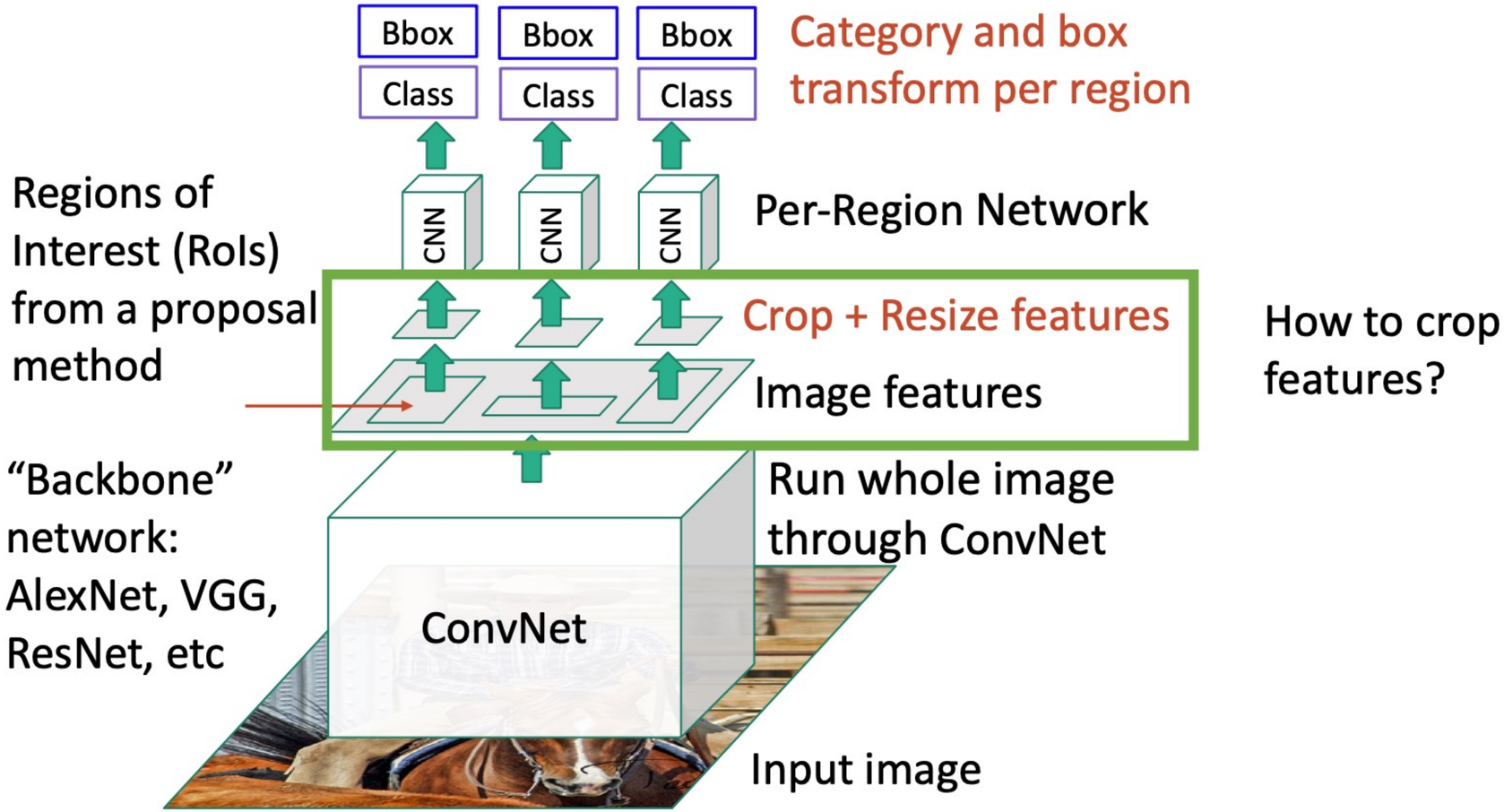
$f_{6.5,5.8} = (f_{6,5} * 0.5 * 0.2) + (f_{7,5} * 0.5 * 0.2)$
$+ (f_{6,6} * 0.5 * 0.8) + (f_{7,6} * 0.5 * 0.8)$

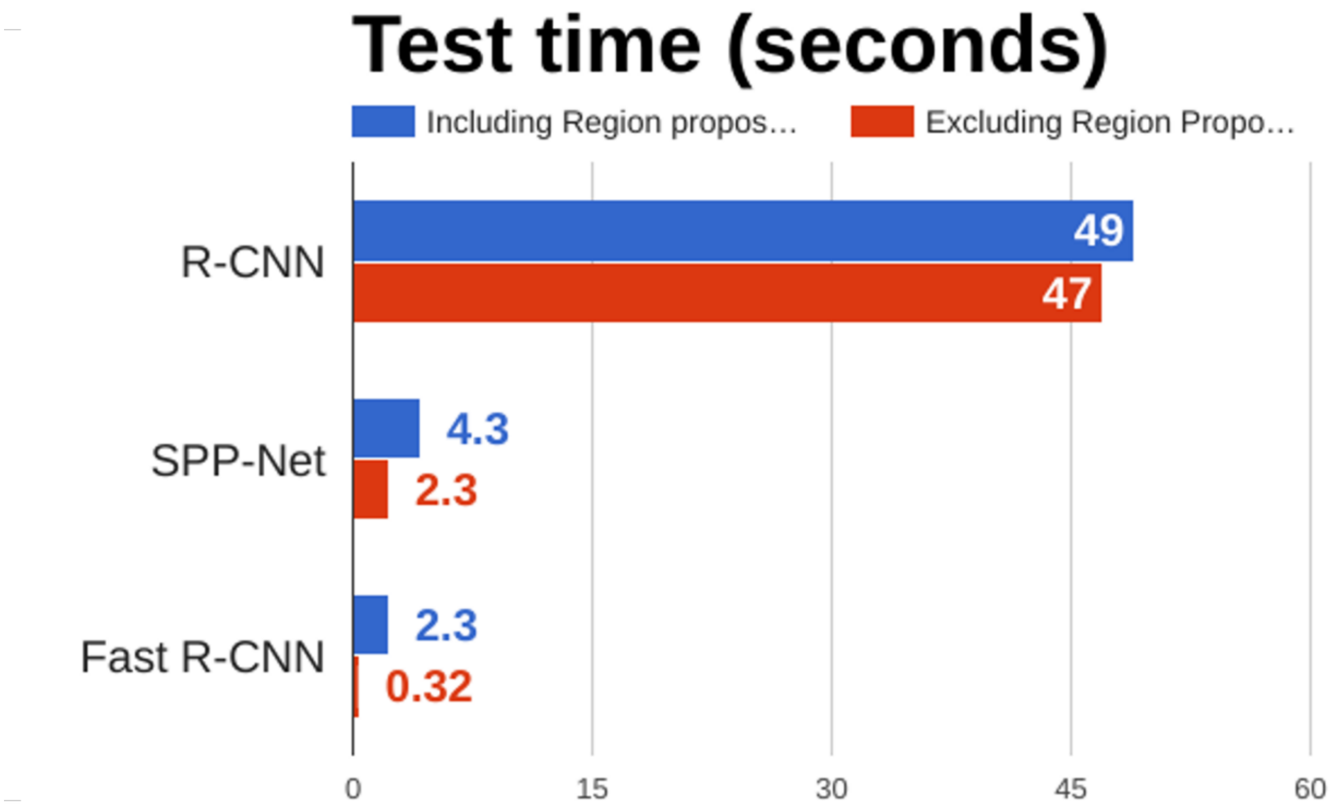
Feature f_{xy} for point (x, y) is a linear combination of features at its four neighboring grid cells:

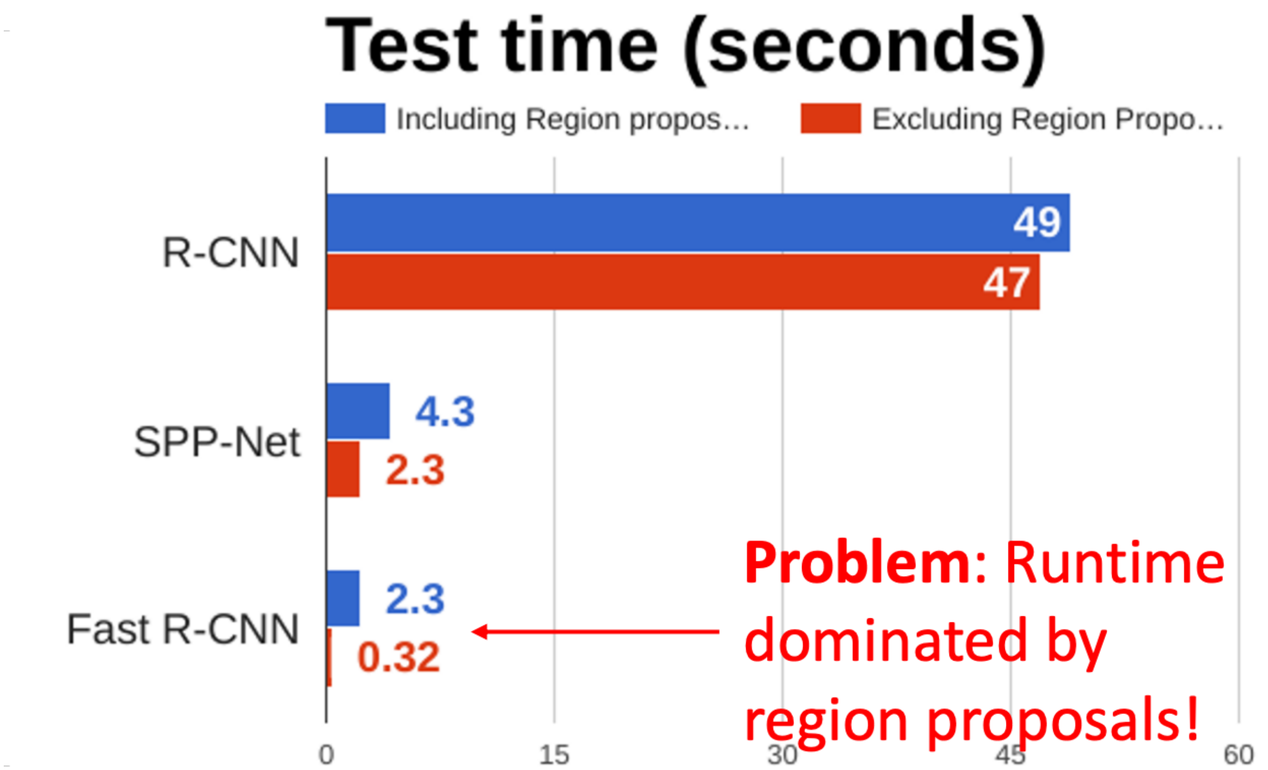
Cropping features: ROI Align



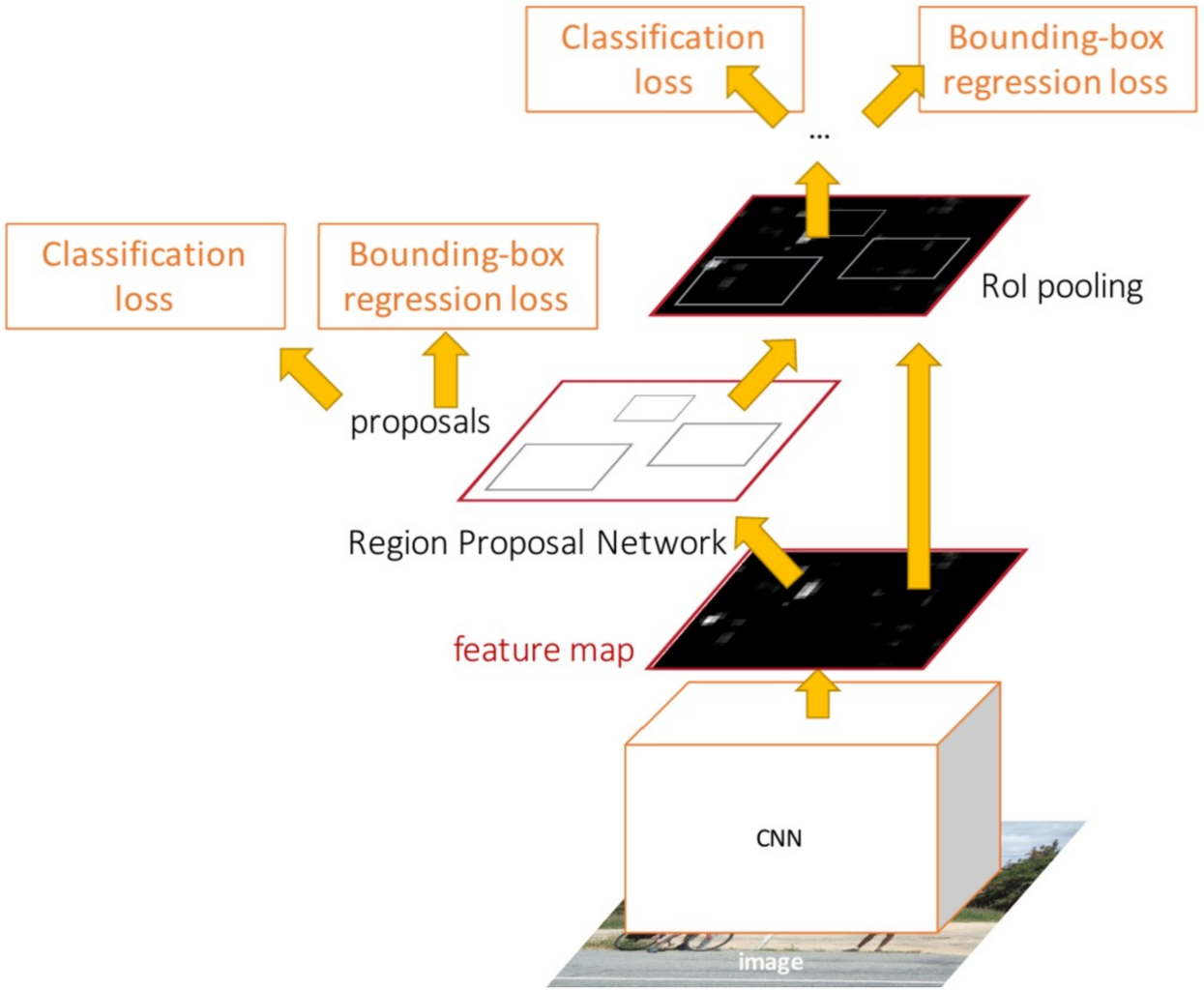
Fast R-CNN


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


Test time (seconds)
Including Region propos…
Excluding Region Propo…
R-CNN
49
47
SPP-Net
4.3
2.3
Fast R-CNN
2.3
0.32
0
15
30
45
60


Test time (seconds)
Including Region propos…
Excluding Region Propo…
R-CNN
49
47
SPP-Net
4.3
2.3
Fast R-CNN
2.3
0.32
0
15
30
45
60
Problem: Runtime dominated by region proposals!

Faster R-CNN: Learnable Region Proposals



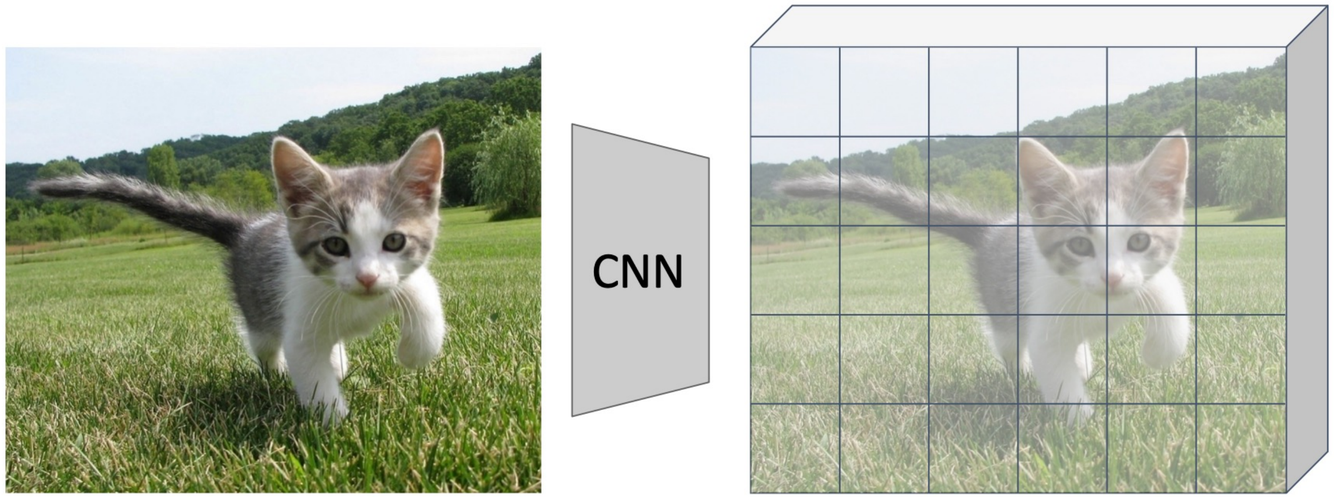
Idea: Calculating region proposals separately is slow – let's learn them jointly instead!

Implementation: insert Region Proposal Network (RPN) to predict proposals from features

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

Region proposal network (RPN)

Run backbone CNN to get
features aligned to input image



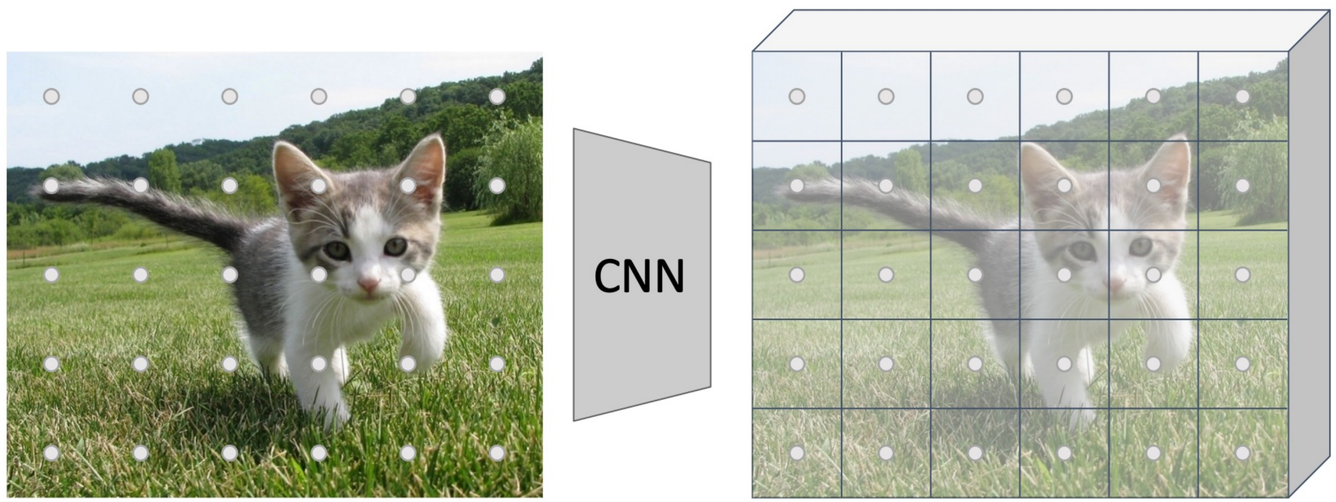
Input Image
(e.g. 3 x 640 x 480)

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

Region proposal network (RPN)

Run backbone CNN to get features aligned to input image

Each feature corresponds to point in the input



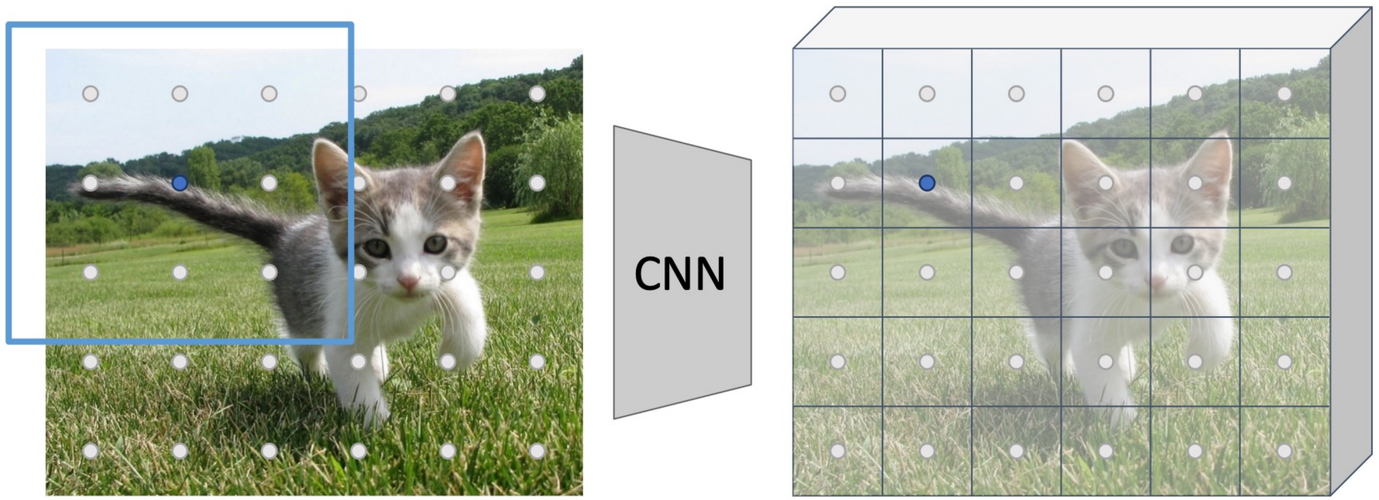
Input Image
(e.g. 3 x 640 x 480)

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

Region proposal network (RPN)

Run backbone CNN to get features aligned to input image

Each feature corresponds to point in the input



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

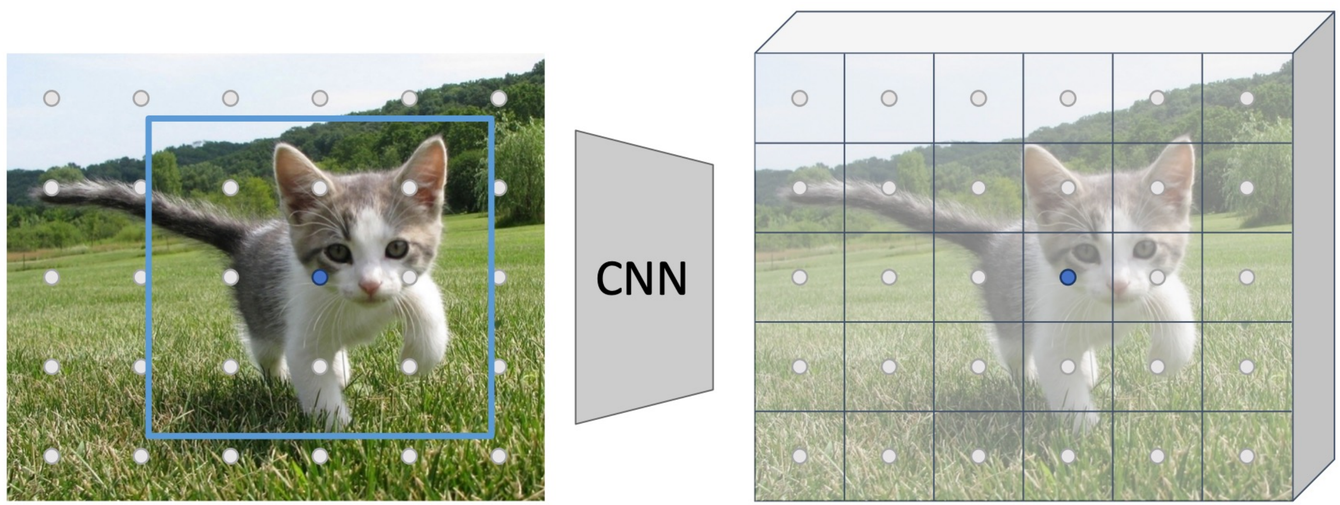
Image features
(e.g. 512 x 5 x 6)

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

Region proposal network (RPN)

Run backbone CNN to get features aligned to input image

Each feature corresponds to point in the input



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

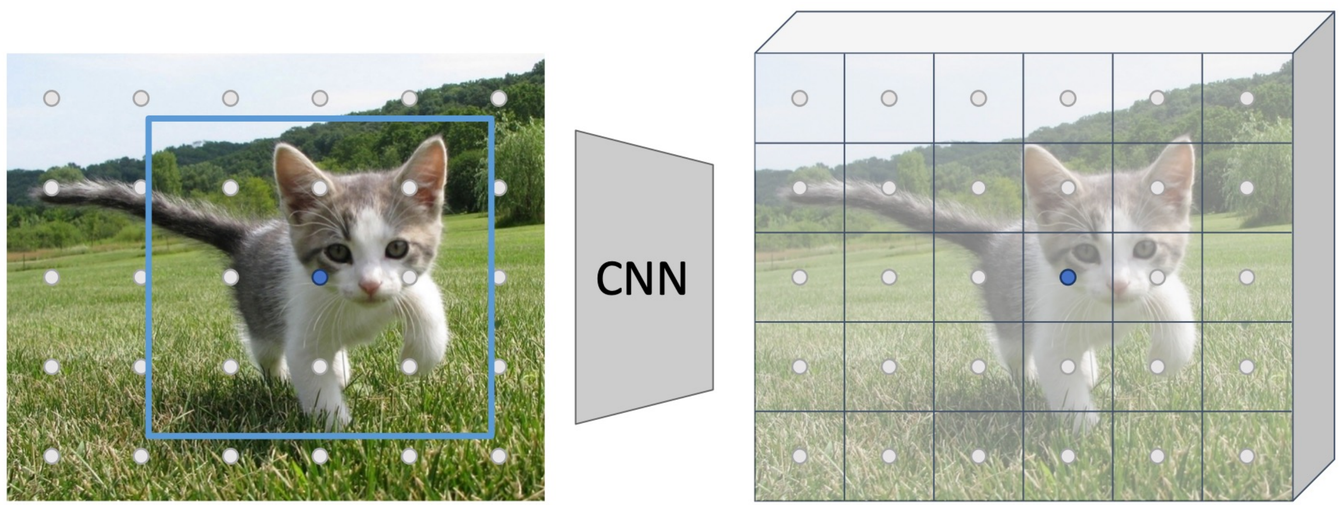
Image features
(e.g. 512 x 5 x 6)

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

Region proposal network (RPN)

Run backbone CNN to get features aligned to input image

Each feature corresponds to point in the input



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

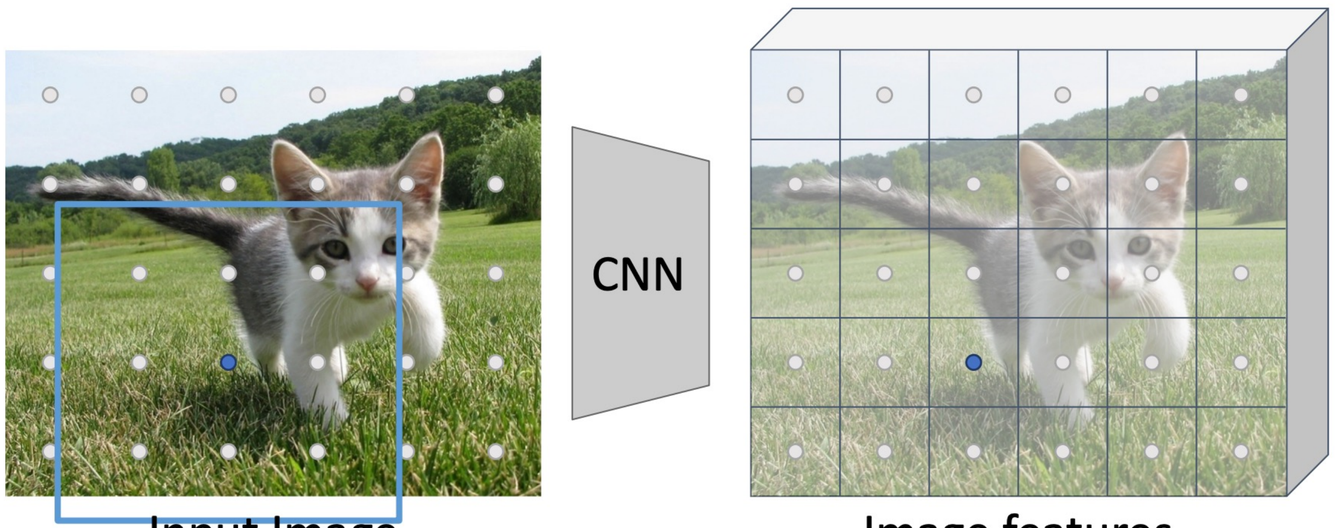
Image features
(e.g. 512 x 5 x 6)

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

Region proposal network (RPN)

Run backbone CNN to get features aligned to input image

Each feature corresponds to point in the input



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

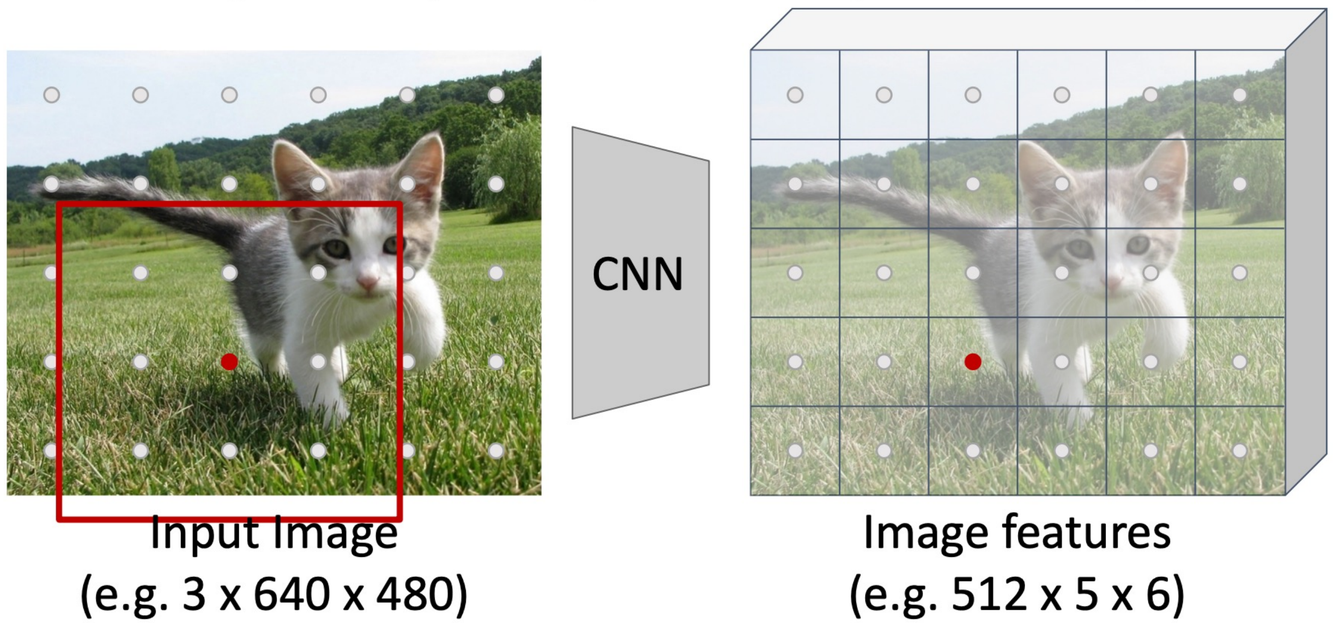
Image features
(e.g. 512 x 5 x 6)

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

Region proposal network (RPN)

Run backbone CNN to get features aligned to input image

Each feature corresponds to point in the input



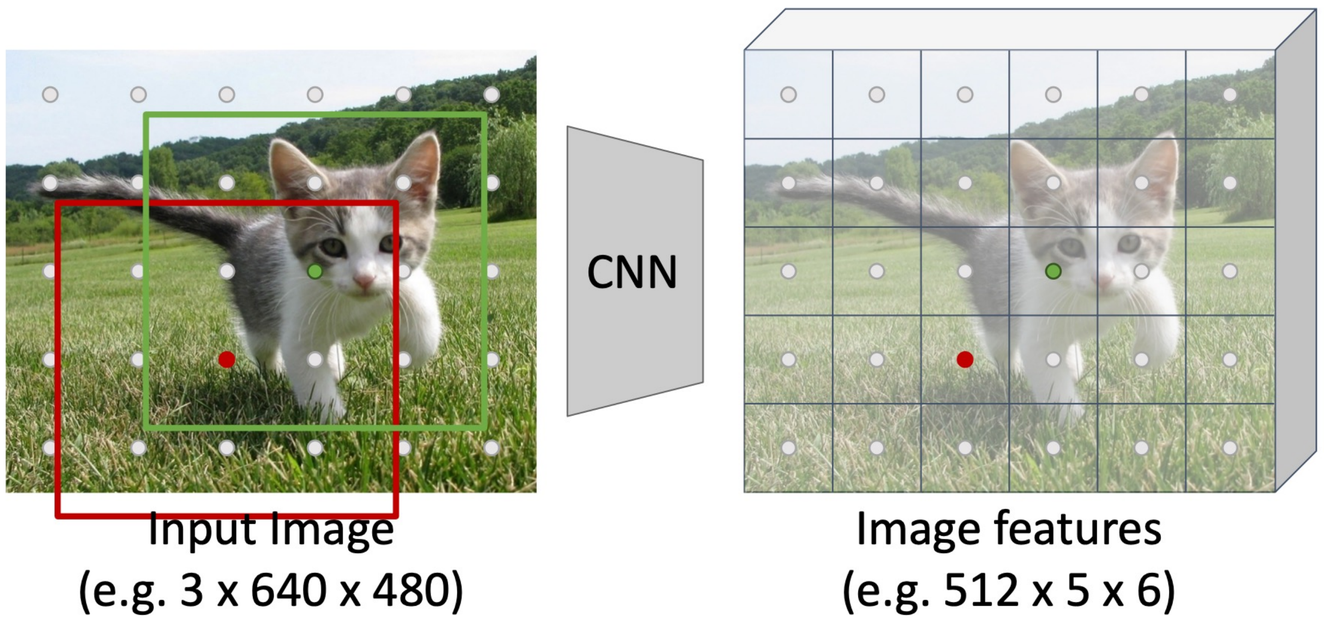
imagine an **anchor box** of fixed size at each point in the feature map

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 point in the input



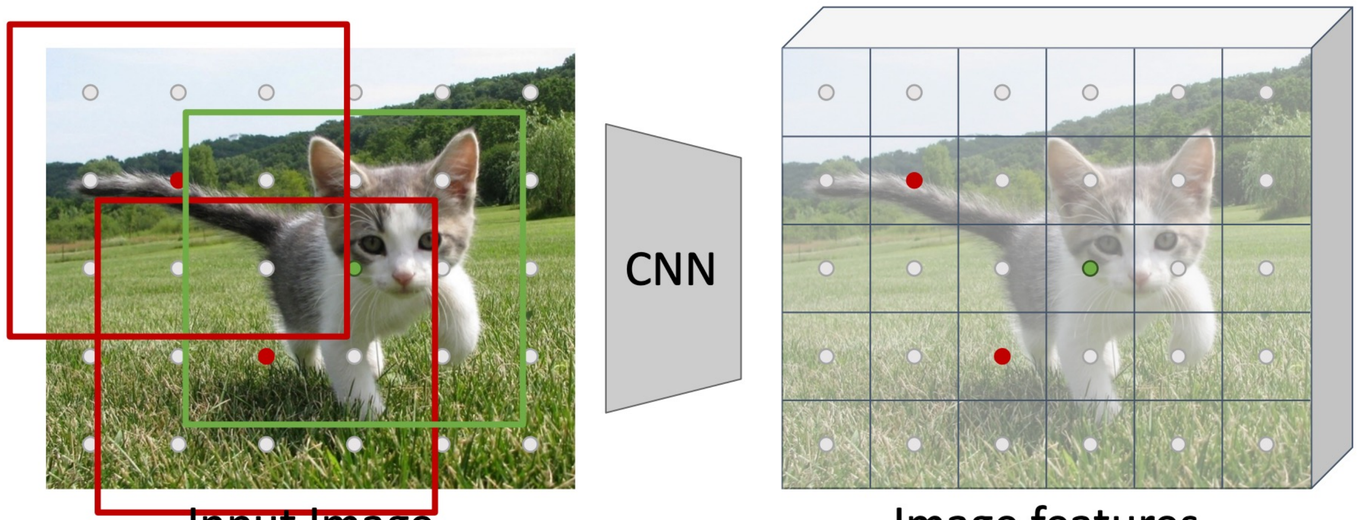
imagine an **anchor box** of fixed size at each point in the feature map

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 point in the input



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

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

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

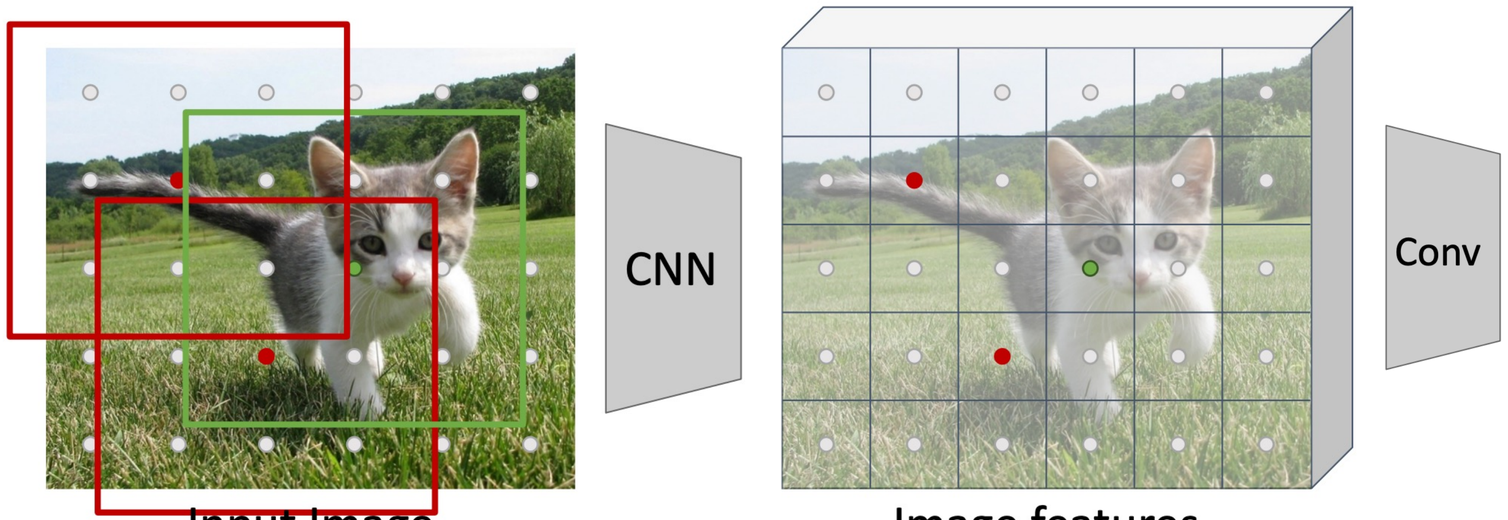
classify each anchor as **positive (object)** or **negative (no object)**

Region proposal network (RPN)

Run backbone CNN to get features aligned to input image

Each feature corresponds to point in the input

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



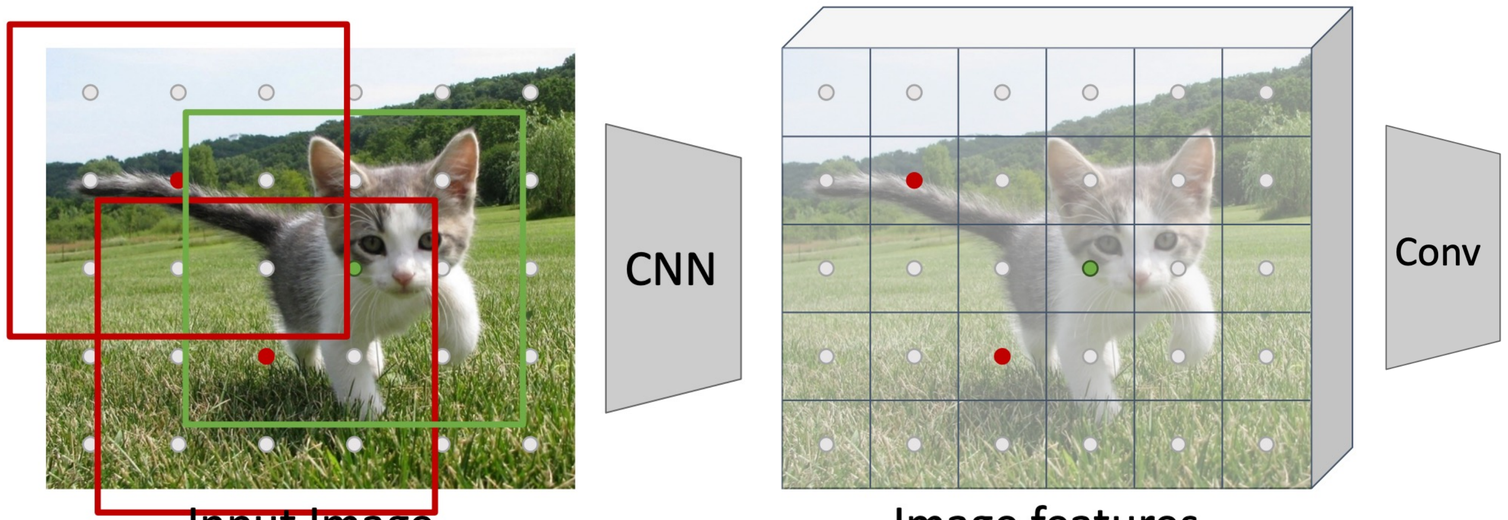
classify each anchor as **positive (object)** or **negative (no object)**

Region proposal network (RPN)

Run backbone CNN to get features aligned to input image

Each feature corresponds to point in the input

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



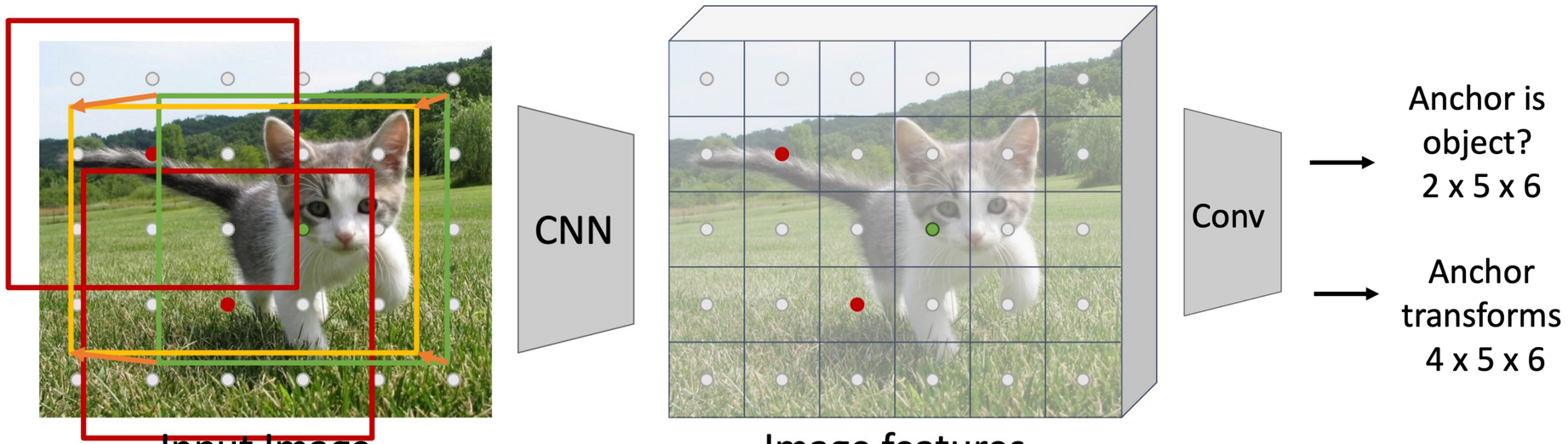
classify each anchor as **positive (object)** or **negative (no object)**

Region proposal network (RPN)

Run backbone CNN to get features aligned to input image

Each feature corresponds to point in the input

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



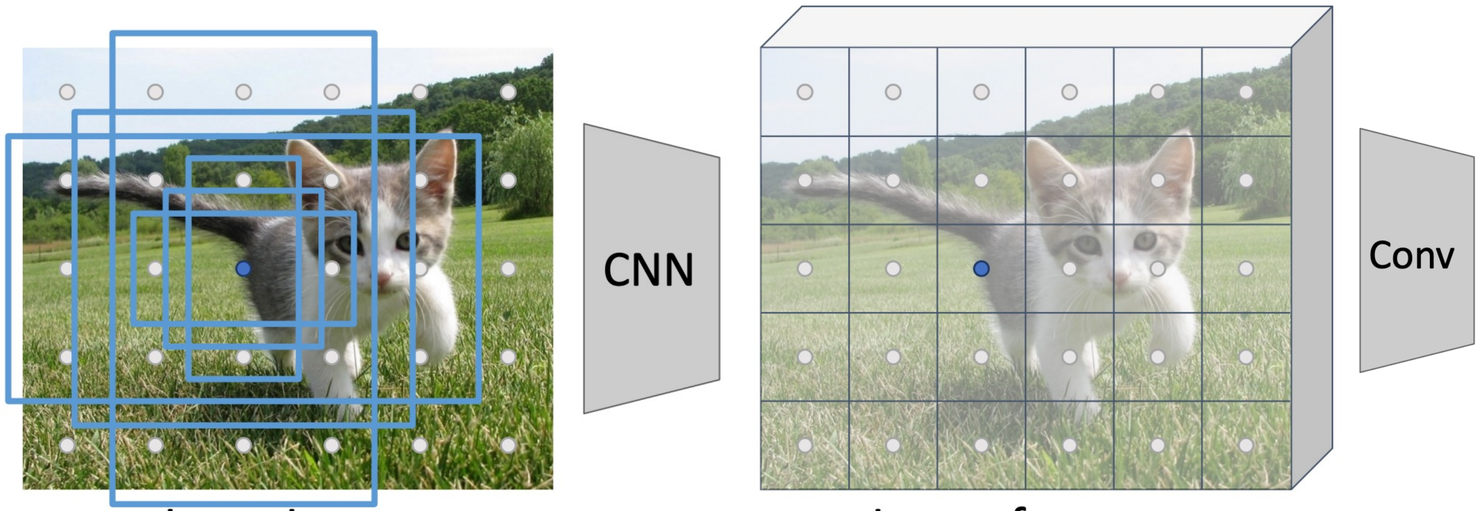
classify each anchor as **positive (object)** or **negative (no object)**

Region proposal network (RPN)

Run backbone CNN to get features aligned to input image

Each feature corresponds to point in the input

in practice: rather then 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)

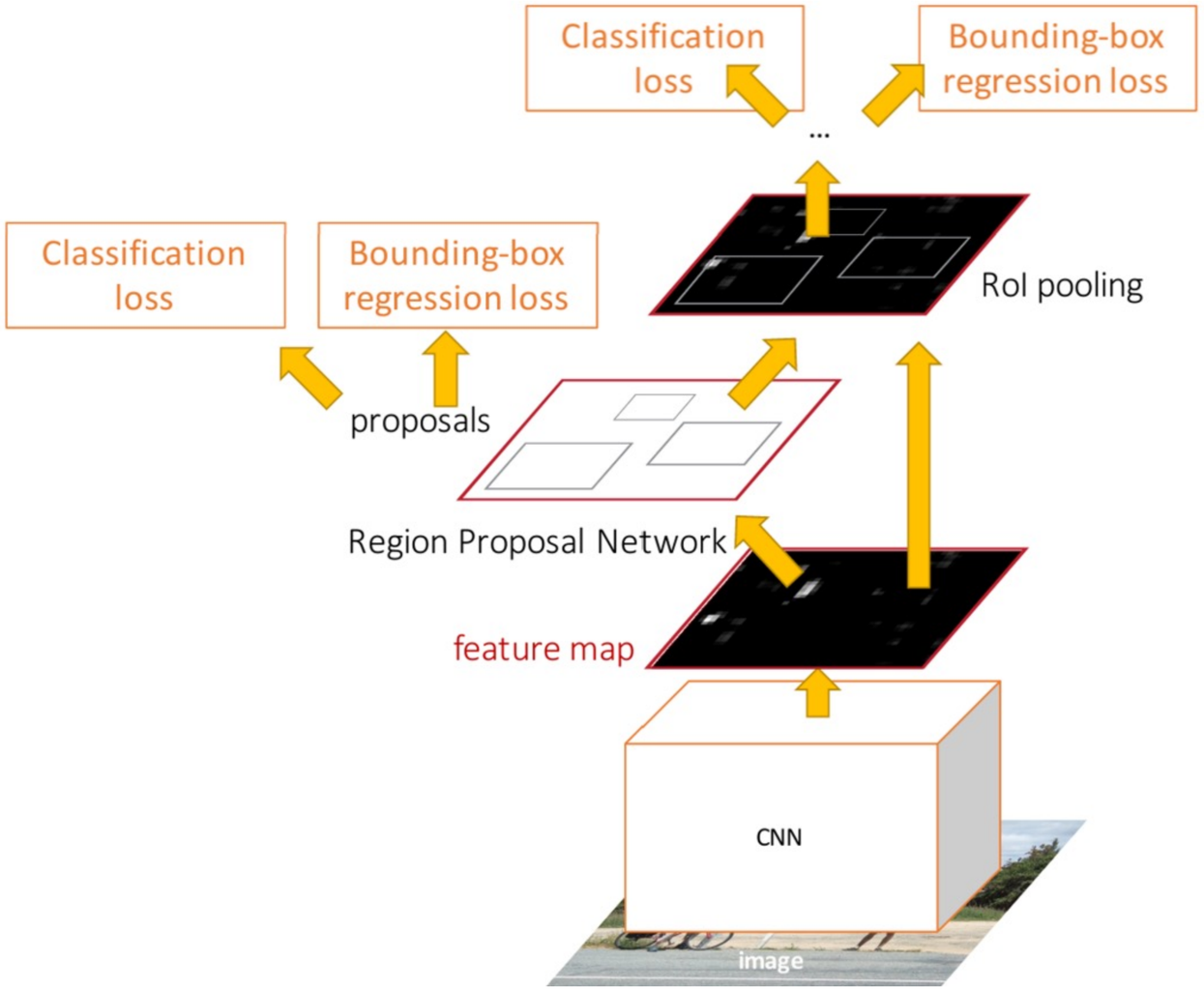
during training supervise positive / negative anchors and box transforms like R-CNN

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

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

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

Faster R-CNN: Learnable Region Proposals



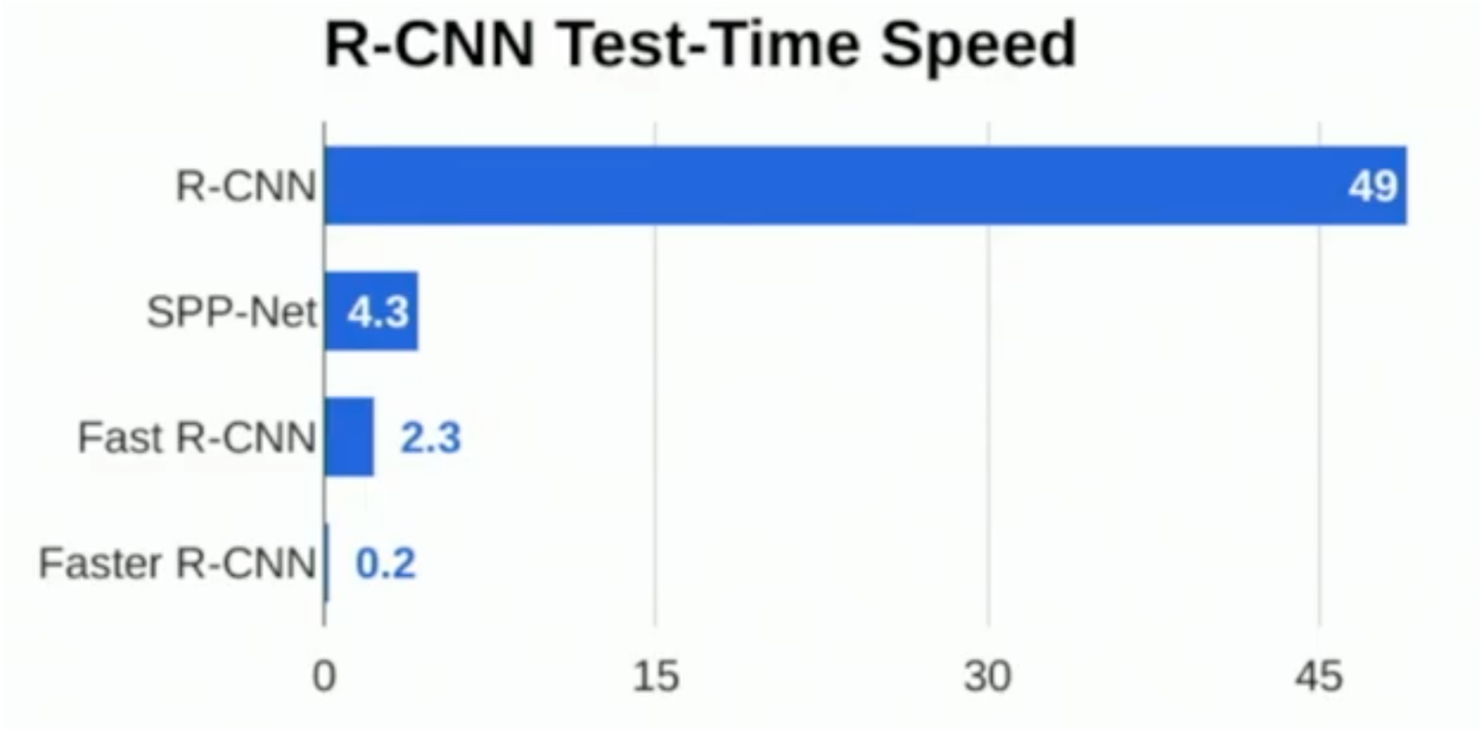
Jointly train with 4 losses:

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

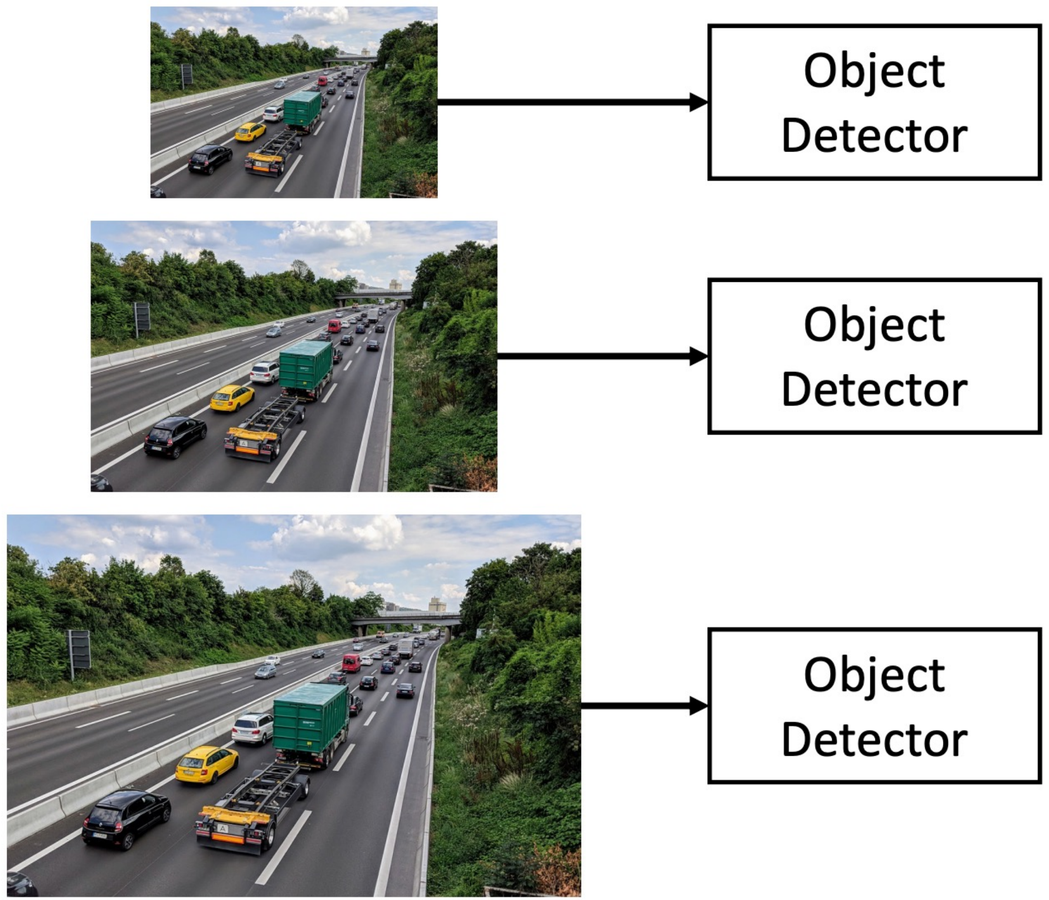
4. **Object regression**: predict transform from proposal box to object box


R-CNN Test-Time Speed
R-CNN
SPP-Net
Fast R-CNN
Faster R-CNN
49
4.3
2.3
0.2
0
15
30
45

Dealing with scale

We need to detect objects of many different scales. How to improve scale invariance of the detector?

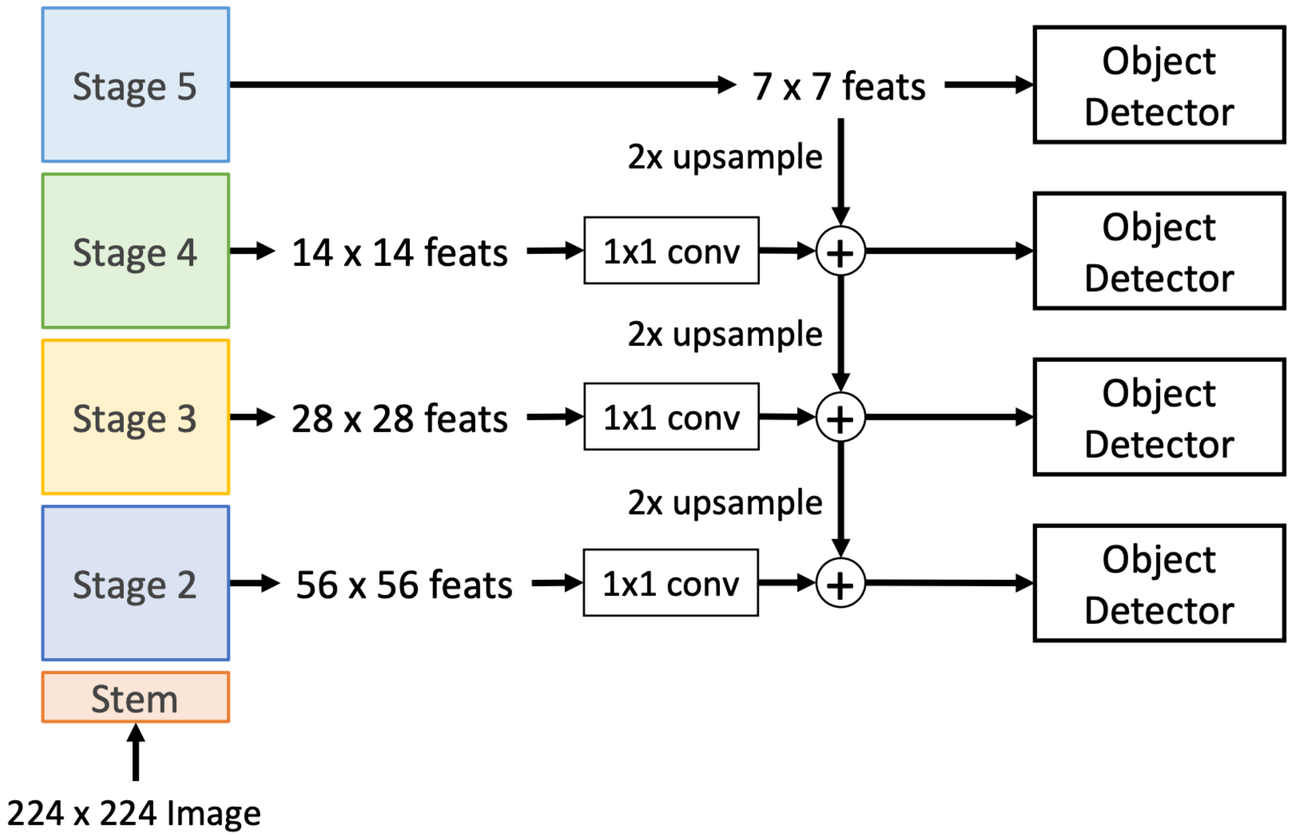
Classical idea: image pyramid



Build an image pyramid by resizing the image to different scales, then process each image scale independently.

Problem: expensive!

Feature Pyramid Network (FPN)



Idea: add top down connections that feed information from high level features back down to lower level features

Efficient multiscale features where all levels benefit from the whole backbone! Widely used in practice

Lin et al, "Feature Pyramid Networks for Object Detection", ICCV 2017

Pre-training (transfer learning)

Neural networks are typically initialized randomly.

Pre-training – an alternative to random initialization:
(1) train a neural network for some time (e.g., classification on ImageNet, detection on COCO)
(2) use this pre-trained network as an initialization for you task

This is often called **transfer learning**: information from the pre-training task is transferred to your downstream task.

Backbones

Architecture:
- CNNs such as AlexNet, VGG16, VGG19 etc
- ResNets such as ResNet34, ResNet50 etc
- Transformers such as ViT

Pretraining:
- On ImageNet for image classification
- Self-supervised learning, e.g. MAE

Modifications:
- Remove final global pooling layer and MLP (the part of the network that is designed for classification)

→ output spatial dims are proportional to input spatial dims

What we didn't talk about:

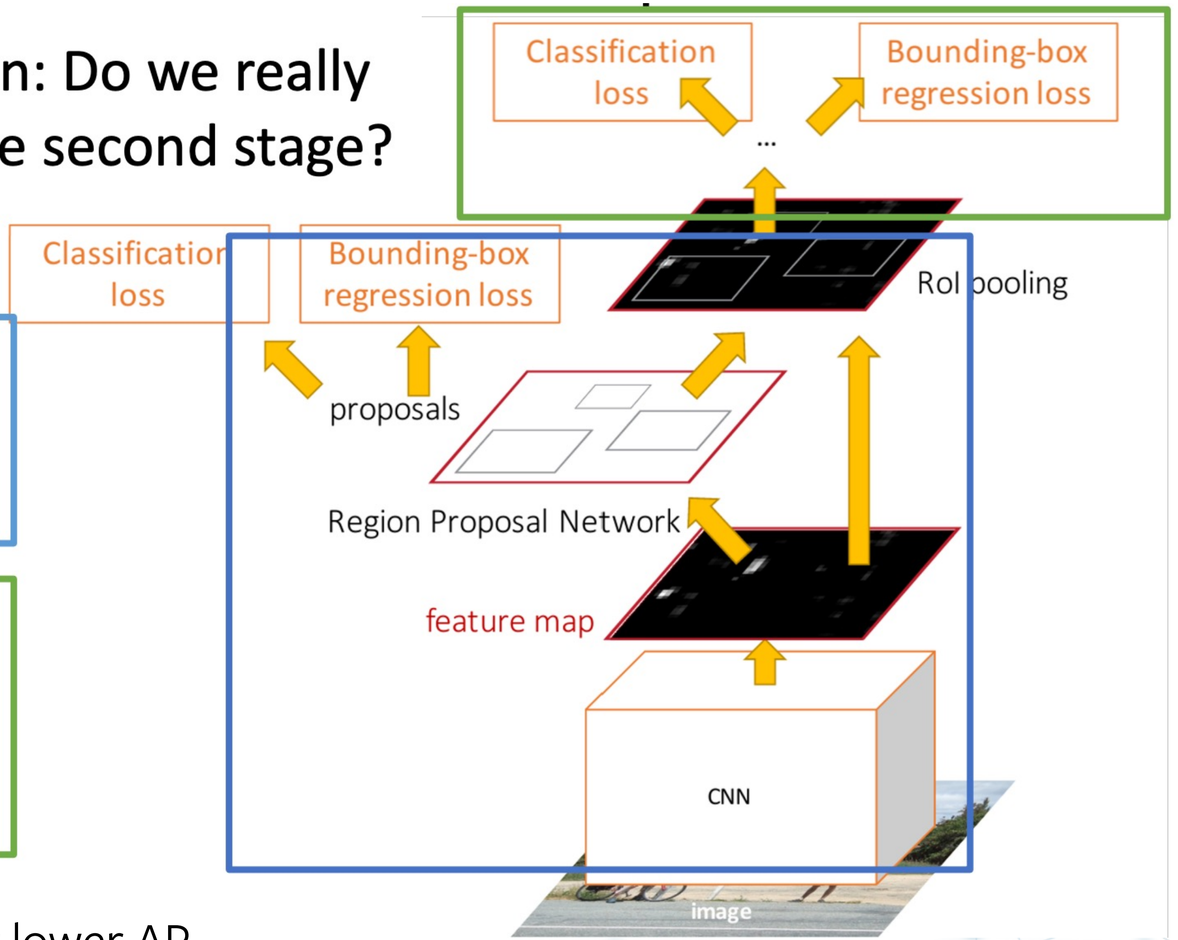
Two-stage vs single-stage detectors

Question: Do we really need the second stage?

Faster R-CNN is a
Two-stage object detector

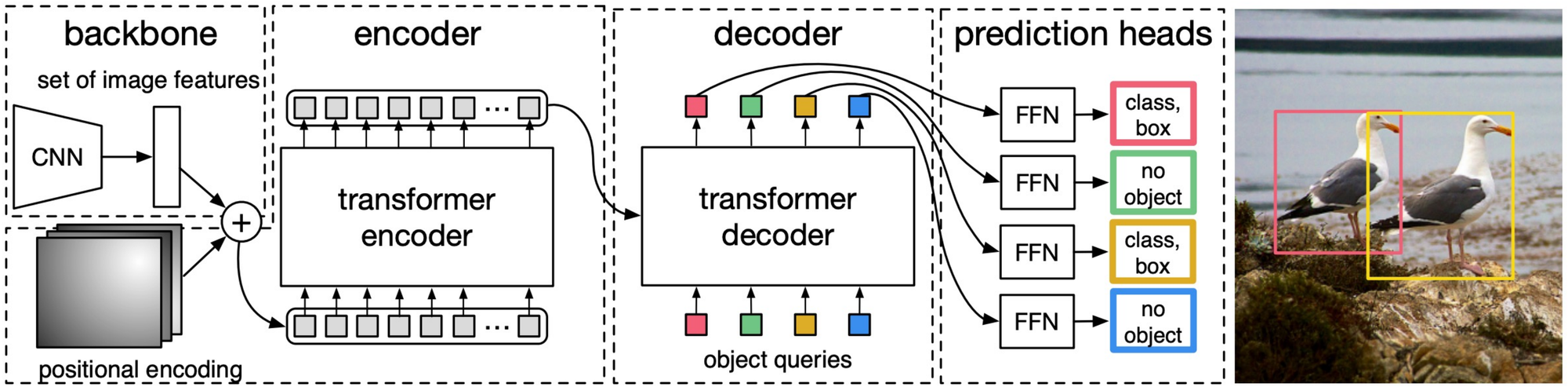
First stage: Run once per image
- Backbone network
- Region proposal network

Second stage: Run once per region
- Crop features: RoI pool / align
- Predict object class
- Prediction bbox offset



Single-stage: RetinaNet, SSD, FCOS – faster but lower AP

What we didn't talk about:
detection with transformers



Carion, Massa et al. End-to-End Object Detection with Transformers

Today's focus

- Principles of multi-object detection
- Region proposals
- Bounding box regression
- R-CNN training and inference
- Non-Max suppression
- Modern object detectors

Questions?