# CS 4644-DL / 7643-A: LECTURE 15
# DANFEI XU

Deep Learning Application to Computer Vision

- Semantic Segmentation
- Object Detection
- Instance Segmentation

# **Image Classification**: A core task in Computer Vision

This image by Nikita is licensed under CC-BY 2.0

(assume given a set of possible labels)
{dog, cat, truck, plane, ...}

→ cat

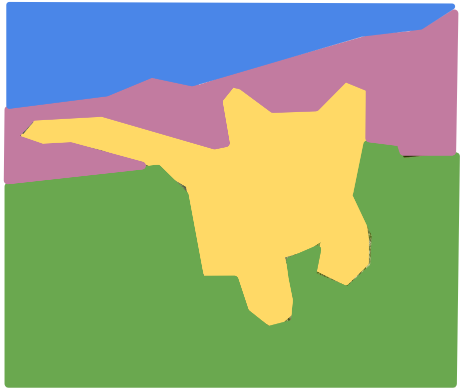
# Computer Vision Tasks

## Classification

**CAT**

No spatial extent

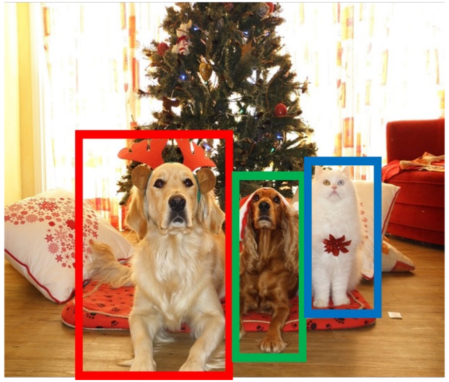
## Semantic Segmentation



**GRASS, CAT, TREE, SKY**

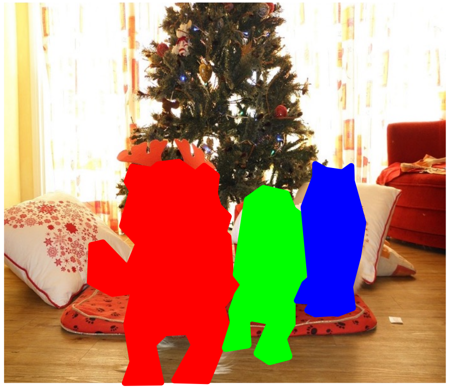
No objects, just pixels

## Object Detection



**DOG, DOG, CAT**

## Instance Segmentation



**DOG, DOG, CAT**

Multiple Object

# Semantic Segmentation

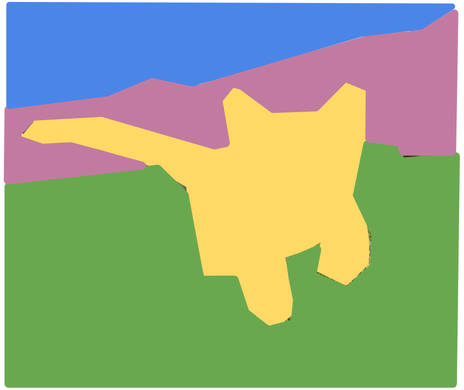
Classification



CAT

No spatial extent

**Semantic Segmentation**



GRASS, CAT, TREE, SKY

No objects, just pixels

Object Detection



DOG, DOG, CAT

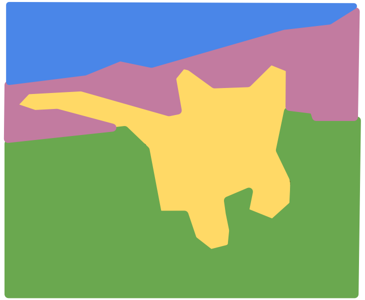
Instance Segmentation



DOG, DOG, CAT

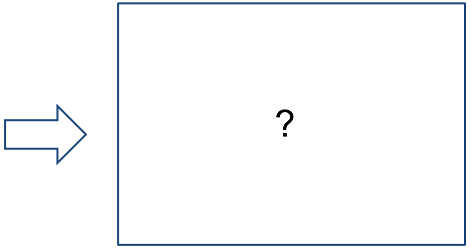
Multiple Object

# Semantic Segmentation: The Problem



GRASS, CAT,
TREE, SKY, ...

Paired training data: for each training image, each pixel is labeled with a semantic category.




At test time, classify each pixel of a new image.

# Semantic Segmentation Idea: Sliding Window

Full image

# Semantic Segmentation Idea: Sliding Window

Full image

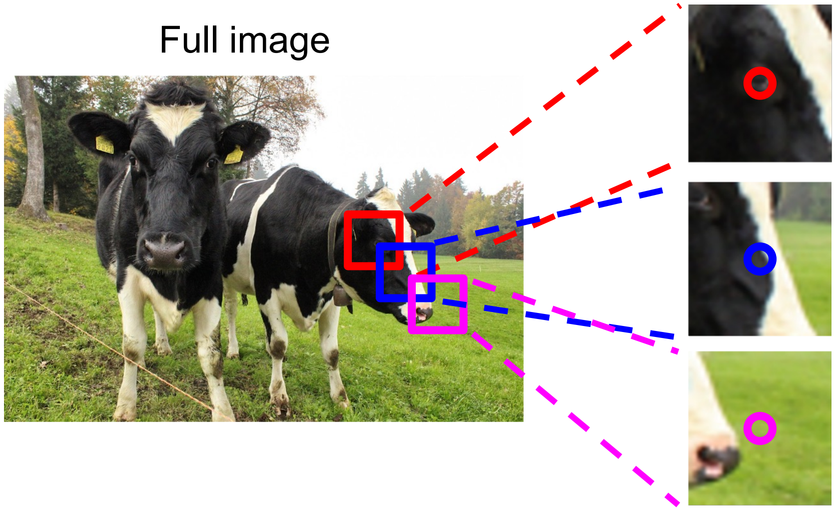
?

Impossible to classify without context

Q: how do we include context?

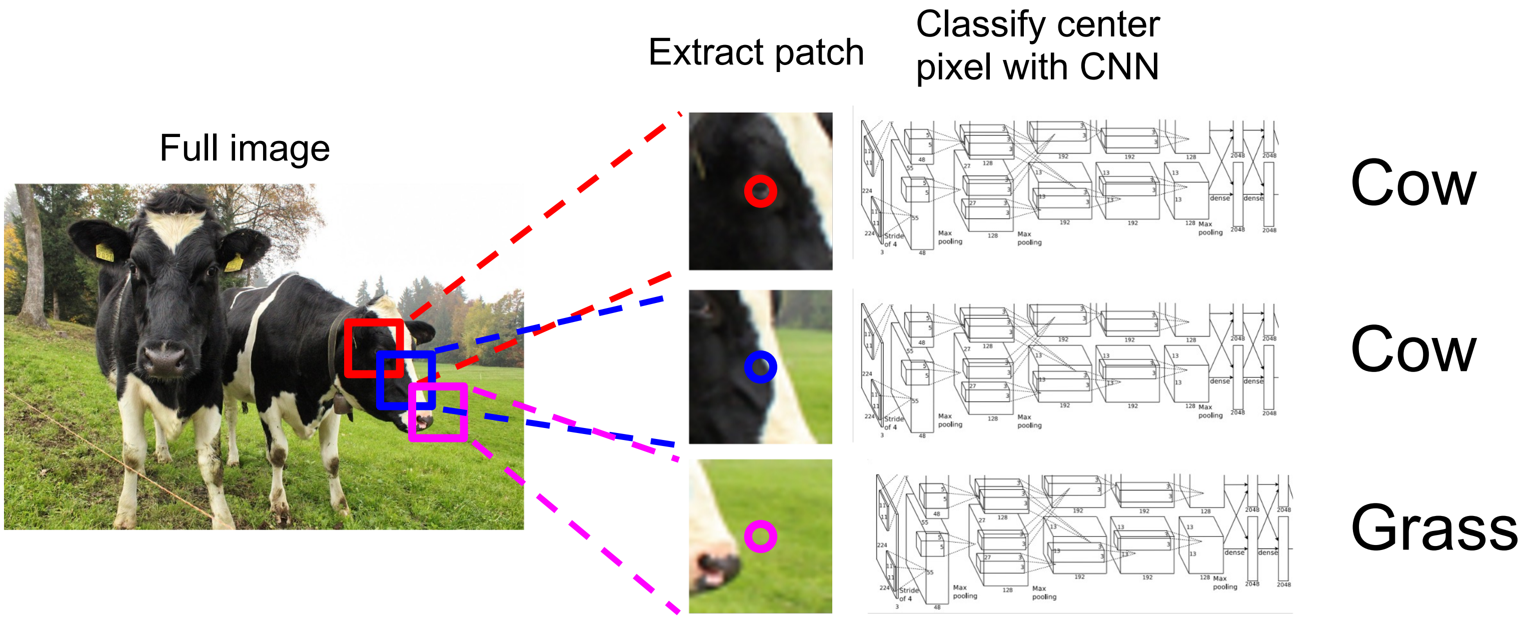
# Semantic Segmentation Idea: Sliding Window

Full image



Q: how do we model this?

# Semantic Segmentation Idea: Sliding Window




The “sliding window” approach

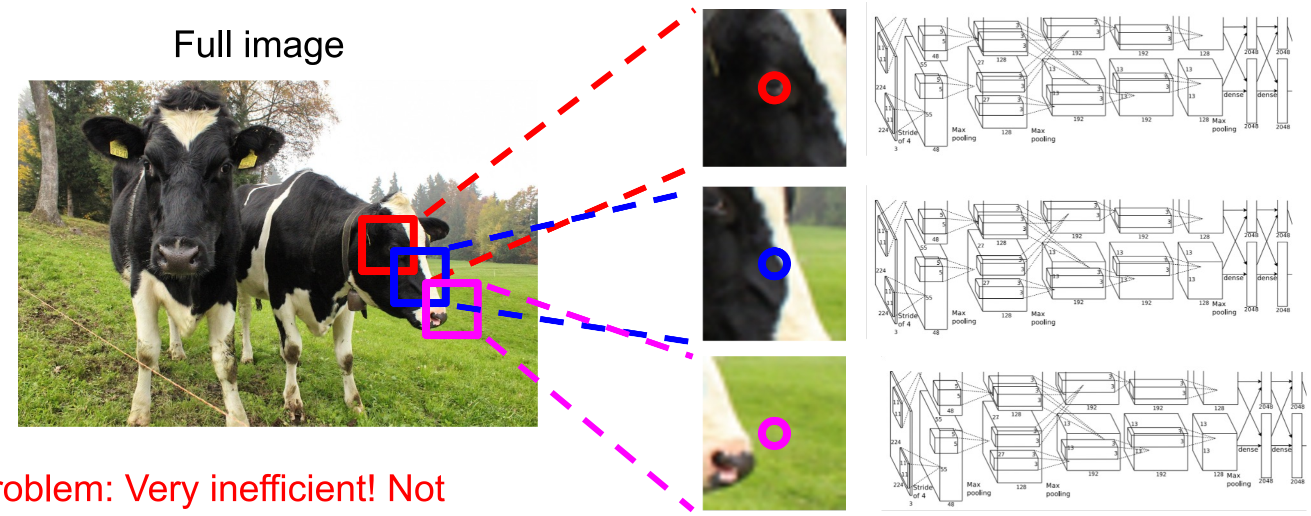
Farabet et al, “Learning Hierarchical Features for Scene Labeling,” TPAMI 2013
Pinheiro and Collobert, “Recurrent Convolutional Neural Networks for Scene Labeling”, ICML 2014

# Semantic Segmentation Idea: Sliding Window

Full image

Extract patch

Classify center pixel with CNN



Cow

Cow

Grass

Problem: Very inefficient! Not reusing shared features between overlapping patches

The “sliding window” approach

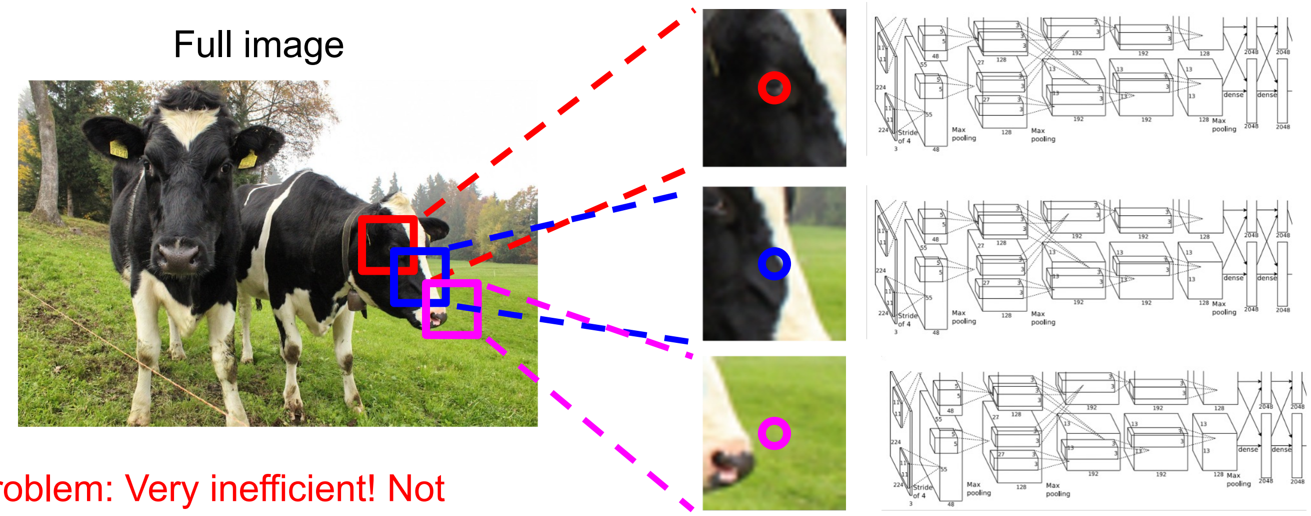
Farabet et al, “Learning Hierarchical Features for Scene Labeling,” TPAMI 2013
Pinheiro and Collobert, “Recurrent Convolutional Neural Networks for Scene Labeling”, ICML 2014

# Semantic Segmentation Idea: Sliding Window

Extract patch

Classify center pixel with CNN

Full image



Cow

Cow

Grass

Problem: Very inefficient! Not reusing shared features between overlapping patches

Observation: lots of duplicate computation in nearby pixels

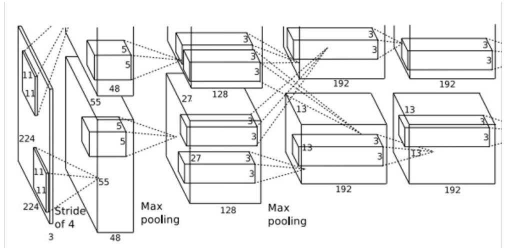
The "sliding window" approach

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013

Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

# Semantic Segmentation Idea: Convolution

Full image

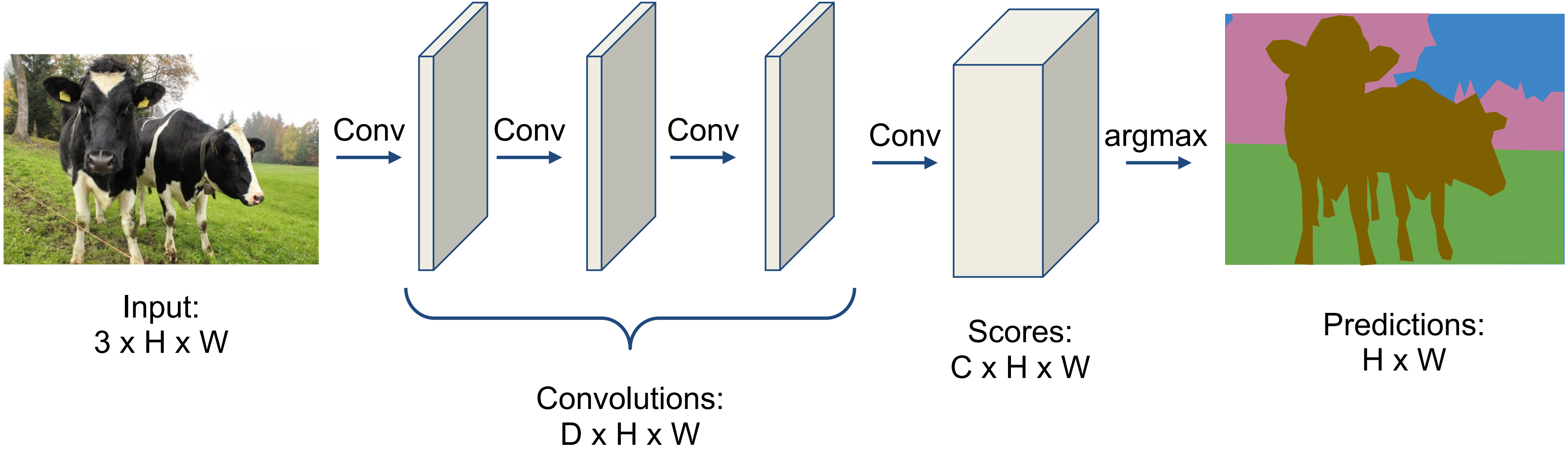



An intuitive idea: encode the entire image with conv net, and do semantic segmentation on top.

Problem: classification architectures often reduce feature spatial sizes to go deeper, but semantic segmentation requires the output size to be the same as input size.

# Semantic Segmentation Idea: Fully Convolutional

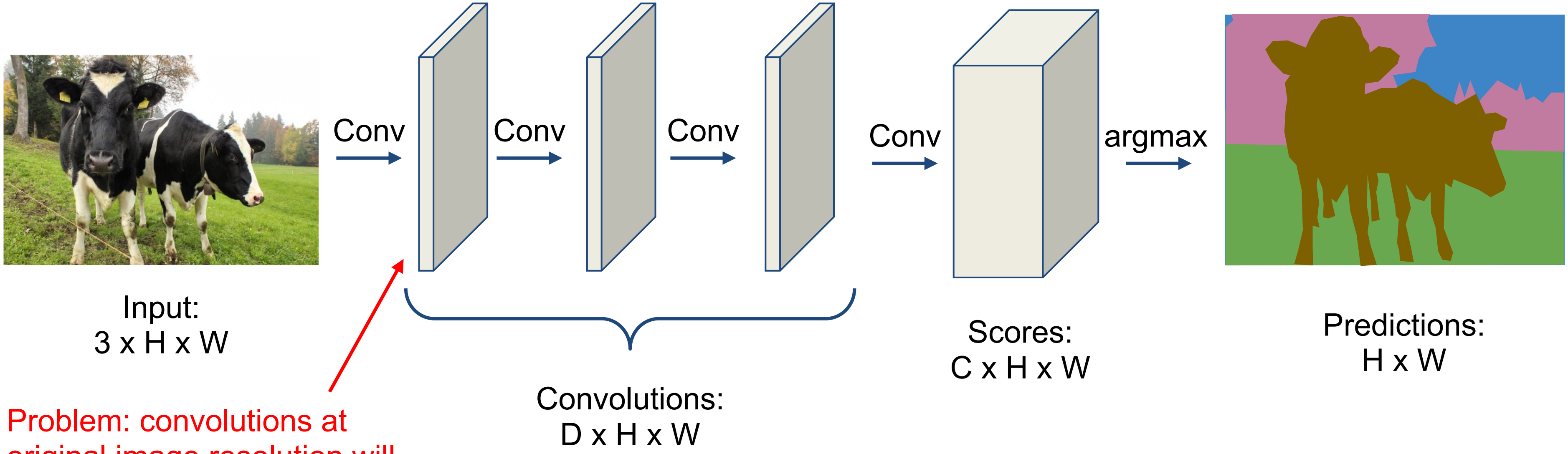
Design a network with only convolutional layers without downsampling operators to make predictions for pixels all at once!




Loss: Pixel-wise cross entropy!

# Semantic Segmentation Idea: Fully Convolutional

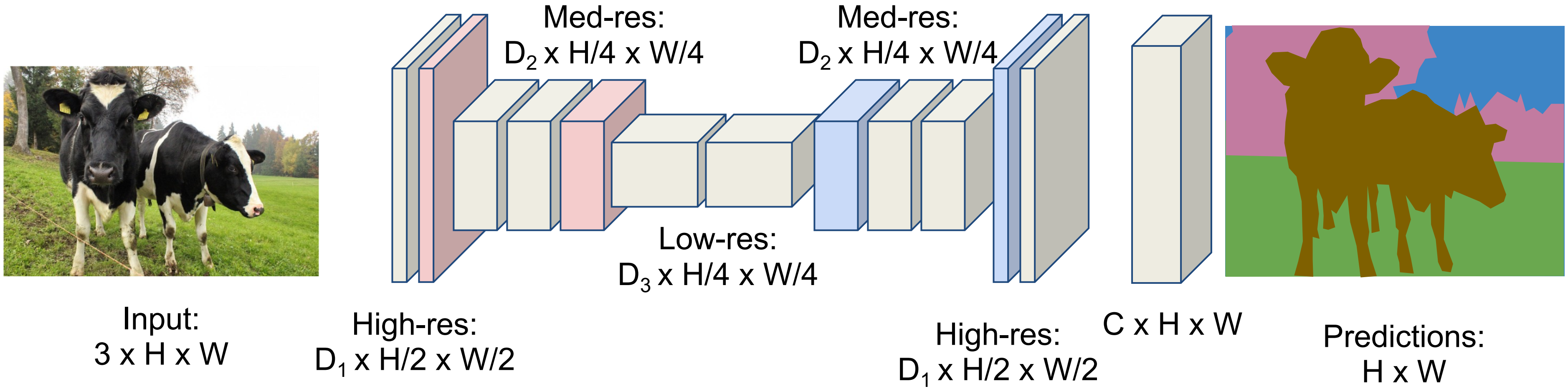
Design a network with only convolutional layers without downsampling operators to make predictions for pixels all at once!




Problem: convolutions at original image resolution will be very expensive ...

# Semantic Segmentation Idea: Fully Convolutional

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!





Long, Shelhamer, and Darrell, “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015
Noh et al, “Learning Deconvolution Network for Semantic Segmentation”, ICCV 2015

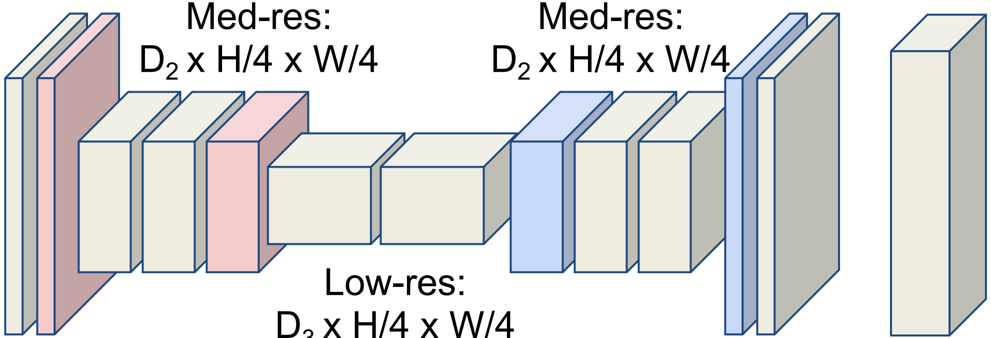
# Semantic Segmentation Idea: Fully Convolutional

**Downsampling**:
Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

**Upsampling**:
???

Input:
$3 \times H \times W$




Predictions:
$H \times W$

Long, Shelhamer, and Darrell, “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015
Noh et al, “Learning Deconvolution Network for Semantic Segmentation”, ICCV 2015

# In-Network upsampling: “Unpooling”

**Nearest Neighbor**

| | |
|---|---|
| 1 | 2 |
| 3 | 4 |

Input: 2 x 2

| | | | |
|---|---|---|---|
| 1 | 1 | 2 | 2 |
| 1 | 1 | 2 | 2 |
| 3 | 3 | 4 | 4 |
| 3 | 3 | 4 | 4 |

Output: 4 x 4

**“Bed of Nails”**

| | |
|---|---|
| 1 | 2 |
| 3 | 4 |

Input: 2 x 2

| | | | |
|---|---|---|---|
| 1 | 0 | 2 | 0 |
| 0 | 0 | 0 | 0 |
| 3 | 0 | 4 | 0 |
| 0 | 0 | 0 | 0 |

Output: 4 x 4

# In-Network upsampling: “Max Unpooling”

**Max Pooling**
Remember which element was max!

| | | | |
|---|---|---|---|
| 1 | 2 | 6 | 3 |
| 3 | 5 | 2 | 1 |
| 1 | 2 | 2 | 1 |
| 7 | 3 | 4 | 8 |

Input: 4 x 4

| | |
|---|---|
| 5 | 6 |
| 7 | 8 |

Output: 2 x 2

. . .

Rest of the network

**Max Unpooling**
Use positions from pooling layer

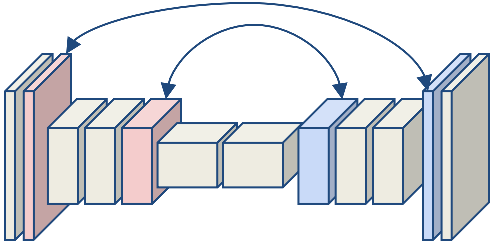
| | |
|---|---|
| 1 | 2 |
| 3 | 4 |

Input: 2 x 2

| | | | |
|---|---|---|---|
| 0 | 0 | 2 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 4 |

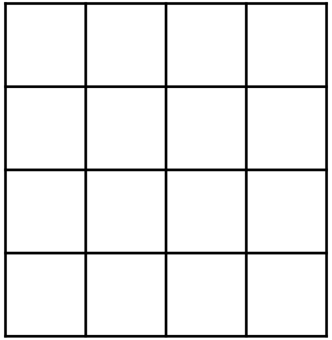
Output: 4 x 4

Corresponding pairs of downsampling and upsampling layers

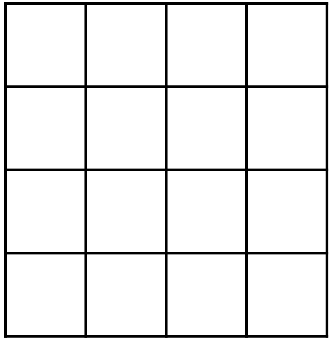


# Learnable Upsampling

**Recall:** Normal 3 x 3 convolution, stride 1 pad 1



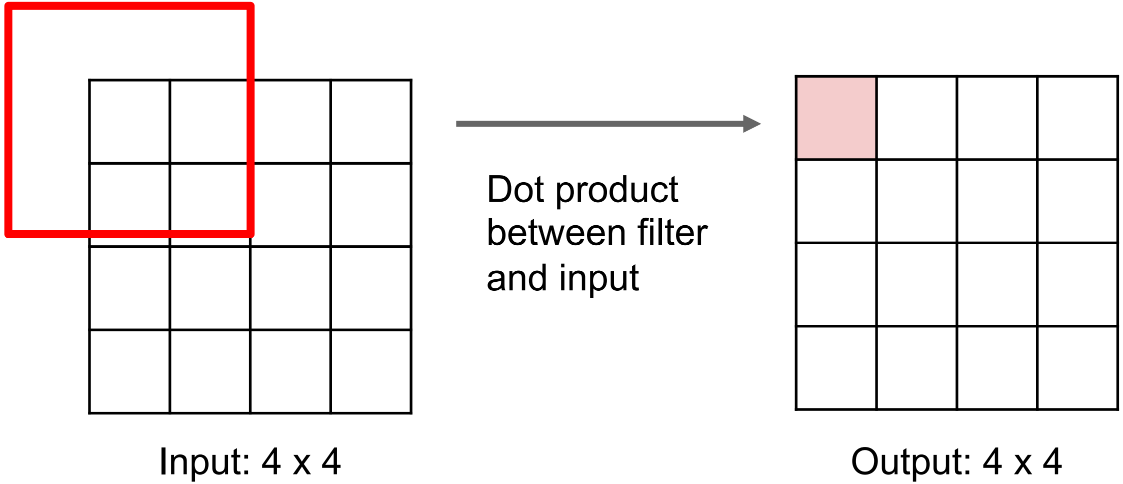
Input: 4 x 4



Output: 4 x 4

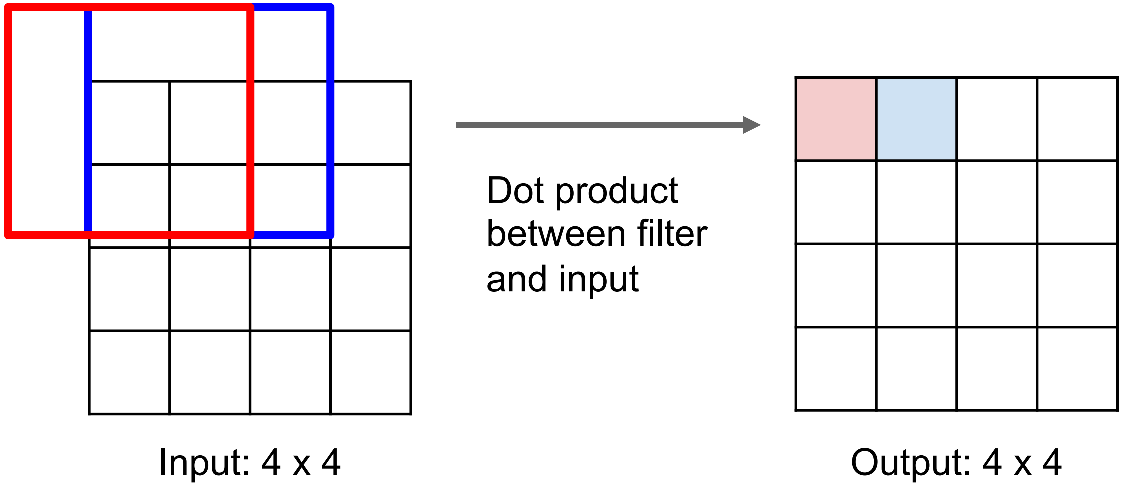
# Learnable Upsampling

**Recall:** Normal 3 x 3 convolution, stride 1 pad 1



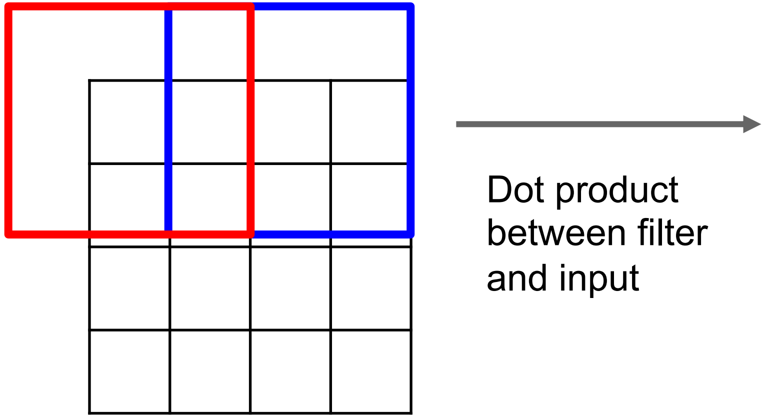
# Learnable Upsampling

**Recall:** Normal 3 x 3 convolution, stride 1 pad 1

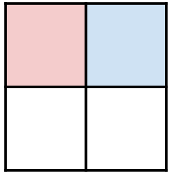


# Learnable Upsampling

**Recall:** Normal 3 x 3 convolution, stride 2 pad 1



Dot product between filter and input



Input: 4 x 4

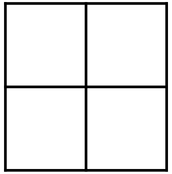
Output: 2 x 2

Filter moves 2 pixels in the input for every one pixel in the output

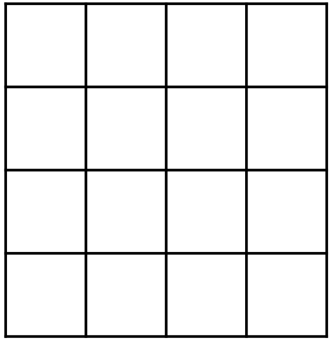
Stride gives ratio between movement in input and output

# Learnable Upsampling: Transposed Convolution

3 x 3 **transpose** convolution, stride 2 pad 1



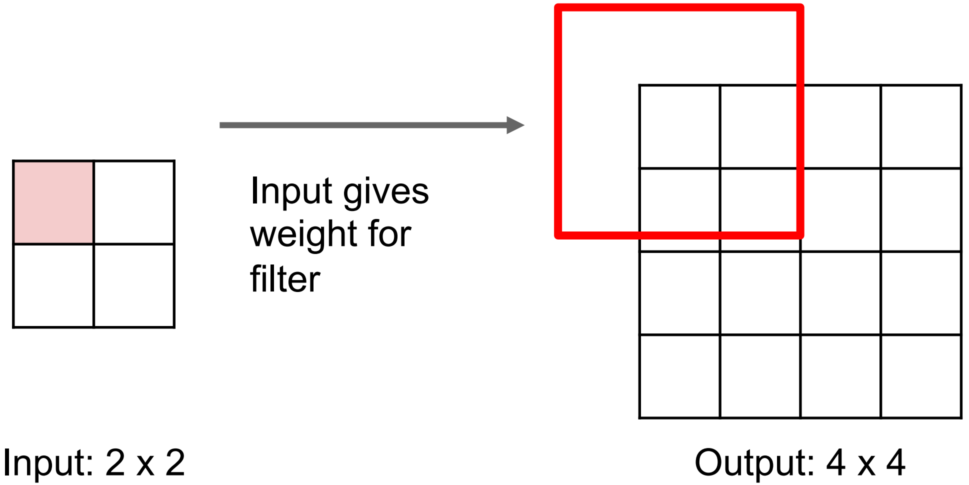
Input: 2 x 2



Output: 4 x 4

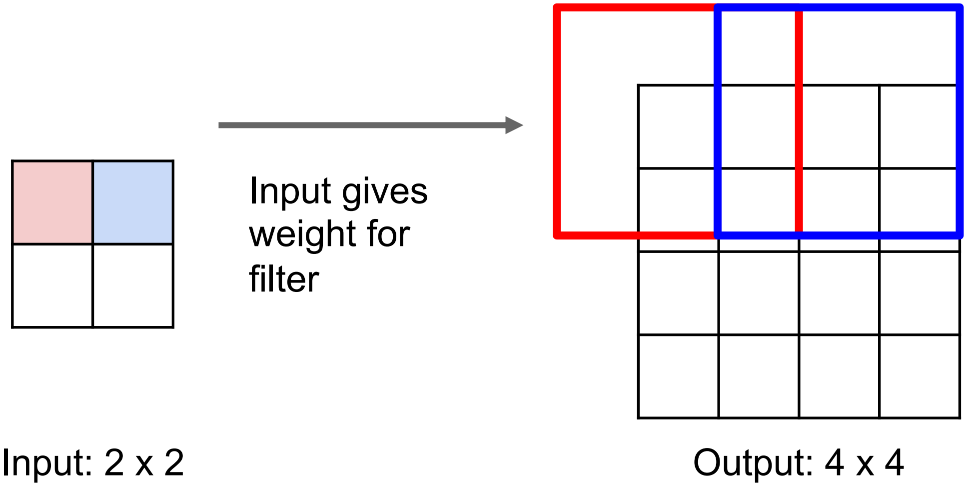
# Learnable Upsampling: Transposed Convolution

3 x 3 **transpose** convolution, stride 2 pad 1



# Learnable Upsampling: Transposed Convolution

3 x 3 **transpose** convolution, stride 2 pad 1



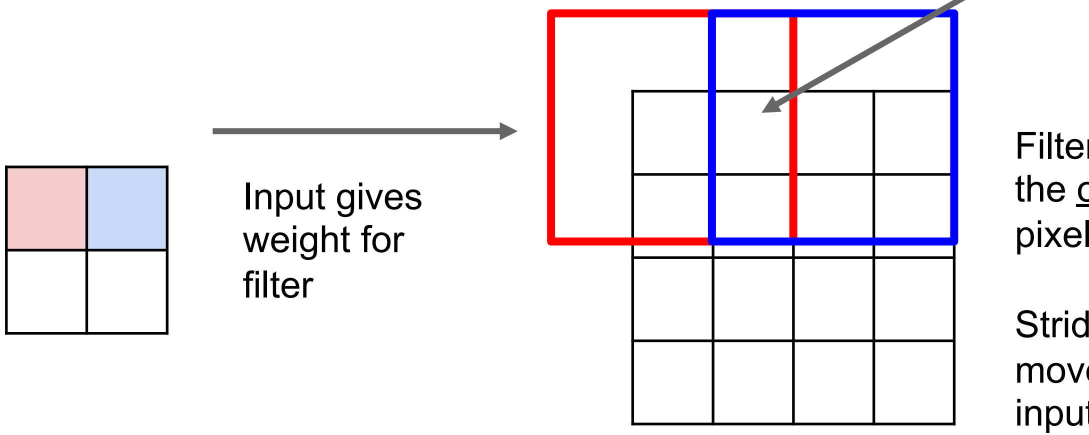

Filter moves 2 pixels in the <u>output</u> for every one pixel in the <u>input</u>

Stride gives ratio between movement in output and input

# Learnable Upsampling: Transposed Convolution

3 x 3 **transpose** convolution, stride 2 pad 1

Sum where output overlaps




Filter moves 2 pixels in the <u>output</u> for every one pixel in the <u>input</u>

Stride gives ratio between movement in output and input

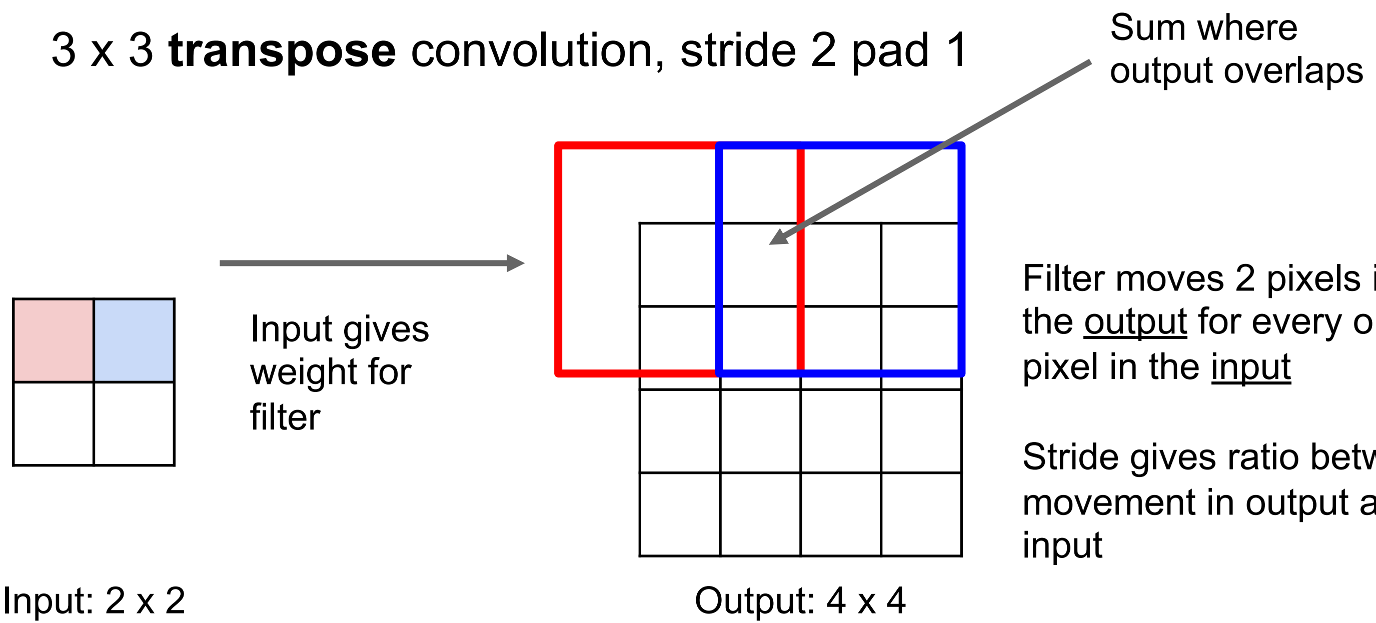
Input: 2 x 2

Output: 4 x 4

# Learnable Upsampling: Transposed Convolution

Q: Why is it called transpose convolution?

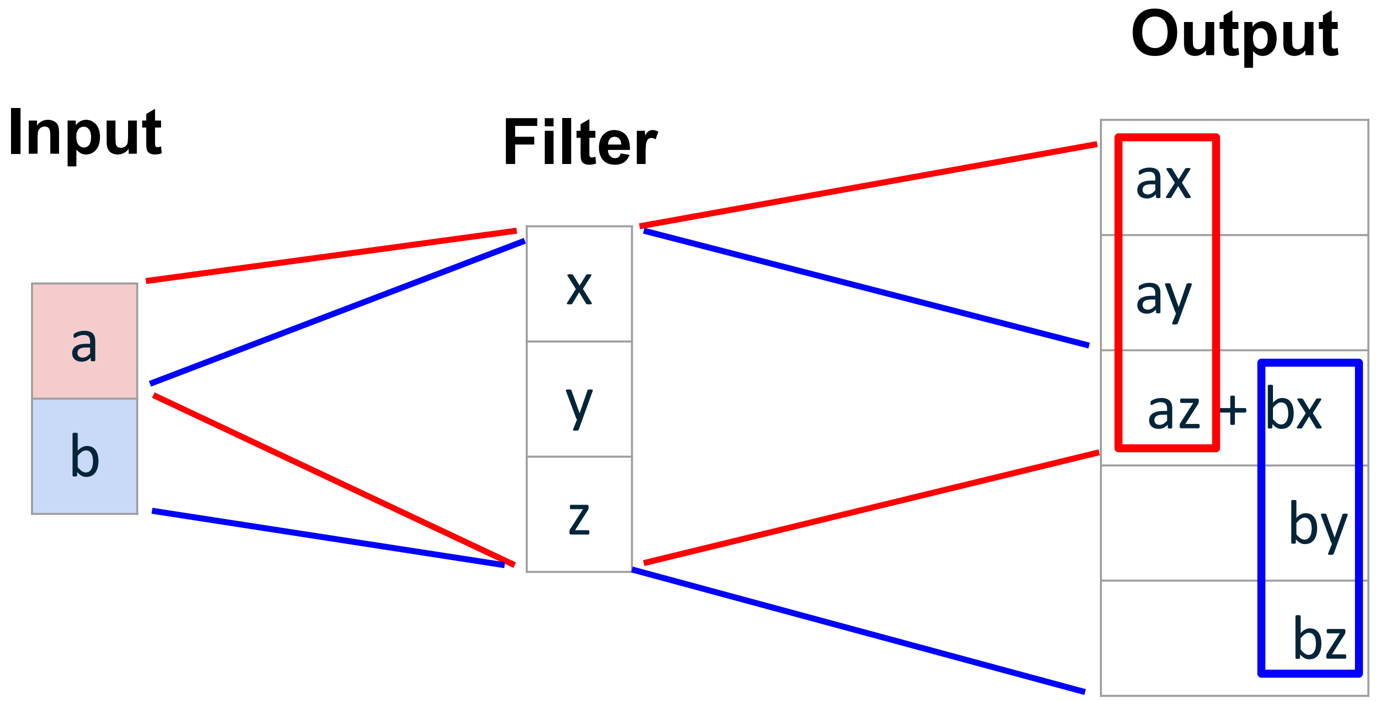
3 x 3 **transpose** convolution, stride 2 pad 1




Filter moves 2 pixels in the <u>output</u> for every one pixel in the <u>input</u>

Stride gives ratio between movement in output and input

# Learnable Upsampling: 1D Example




Output contains copies of the filter weighted by the input, summing at where at overlaps in the output

# Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, <u>stride=2</u>, padding=1

# Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, <u>stride=2</u>, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

Example: 1D transpose conv, kernel size=3, <u>stride=2</u>, padding=0

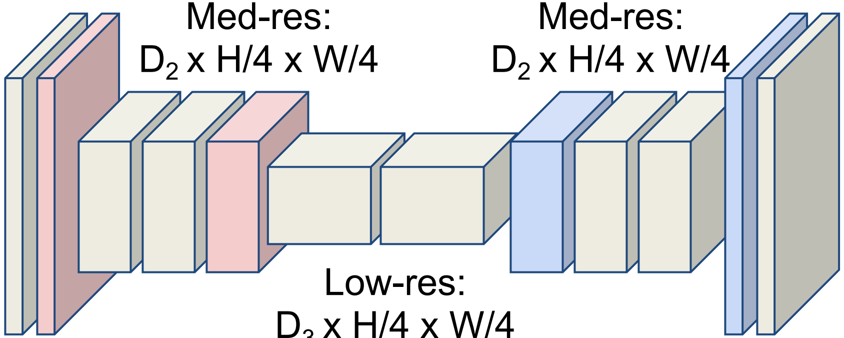
# Semantic Segmentation Idea: Fully Convolutional

**Downsampling**: Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

**Upsampling**: Unpooling or strided transpose convolution

Input:
3 x H x W



High-res:
$D_1$ x H/2 x W/2

Med-res:
$D_2$ x H/4 x W/4

Low-res:
$D_3$ x H/4 x W/4

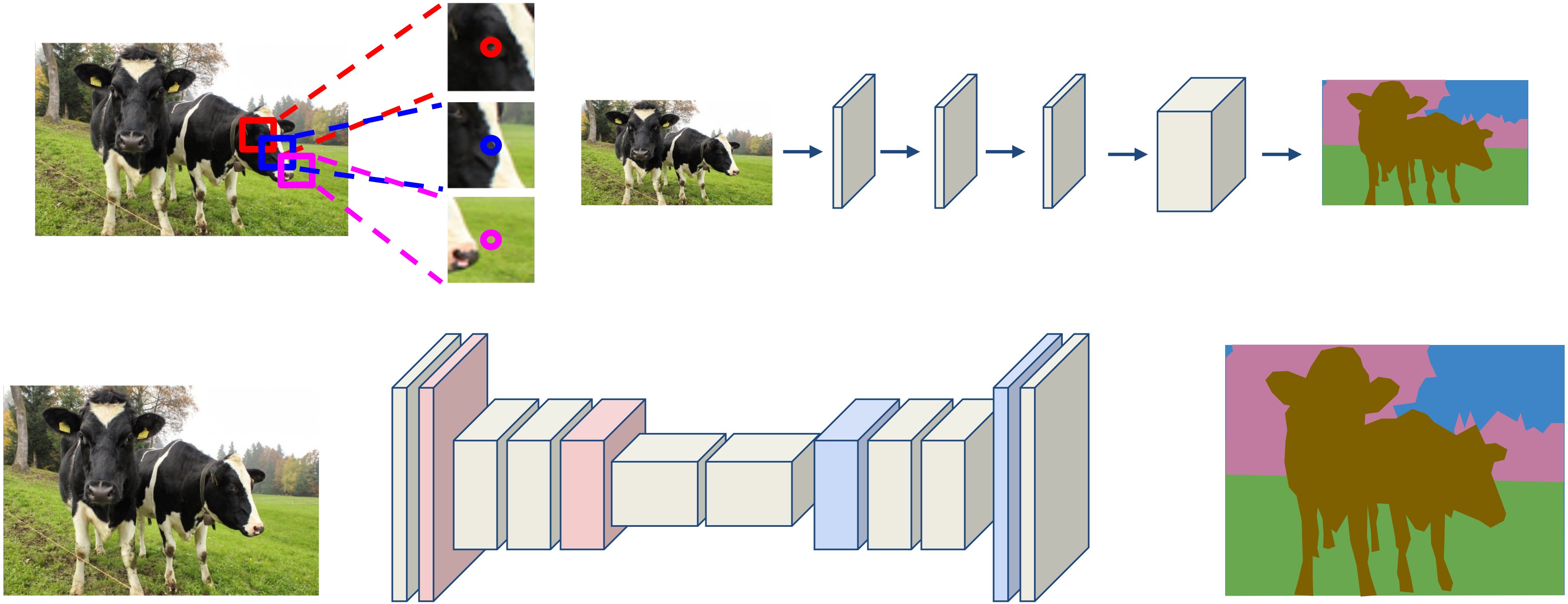
Med-res:
$D_2$ x H/4 x W/4

High-res:
$D_1$ x H/2 x W/2

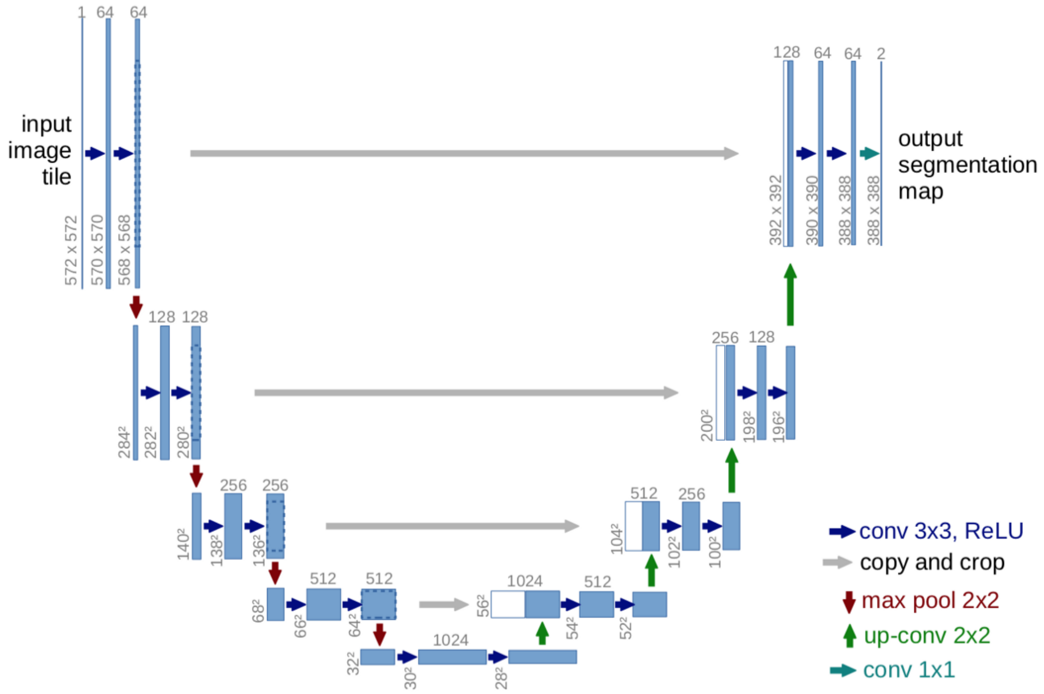
Predictions:
H x W

Long, Shelhamer, and Darrell, “Fully Convolutional Networks for Semantic Segmentation”, CVPR 2015
Noh et al, “Learning Deconvolution Network for Semantic Segmentation”, ICCV 2015

# Semantic Segmentation: Summary



# Semantic Segmentation: U-Net




Idea: Concatenate feature maps from the downsampling stage with the features in the upsampling stage.

Very commonly used today!

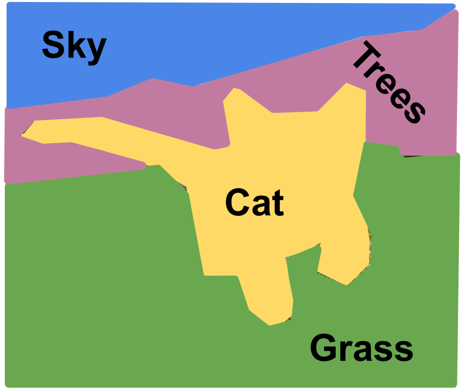
Ronneberger O, Fischer P, Brox T, 2015

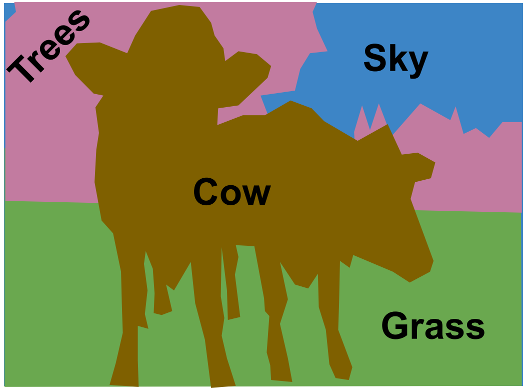
# Semantic Segmentation

Label each pixel in the image with a category label

Don't differentiate instances, only care about pixels





# Object Detection

**Classification**



**CAT**

**Semantic Segmentation**



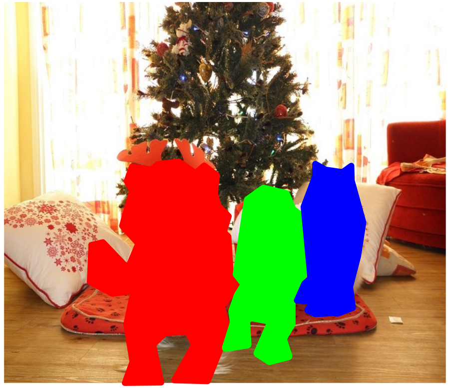
**GRASS, CAT, TREE, SKY**

**Object Detection**



**DOG, DOG, CAT**

**Instance Segmentation**



**DOG, DOG, CAT**

No spatial extent

No objects, just pixels

Multiple Object

# Object Detection

**Classification**



CAT

No spatial extent

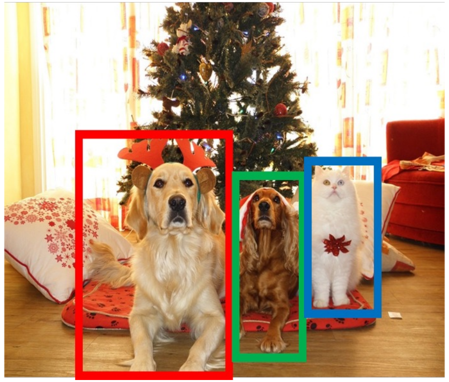
**Semantic Segmentation**



GRASS, CAT, TREE, SKY

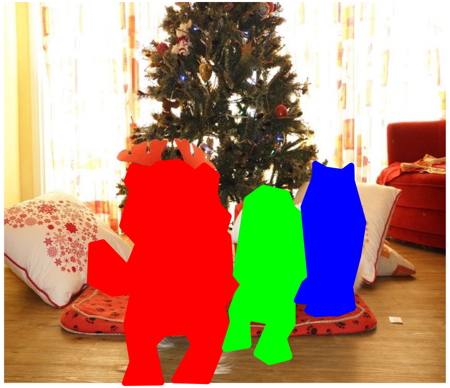
No objects, just pixels

**Object Detection**



DOG, DOG, CAT

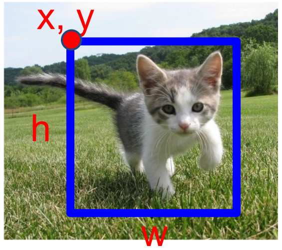
**Instance Segmentation**



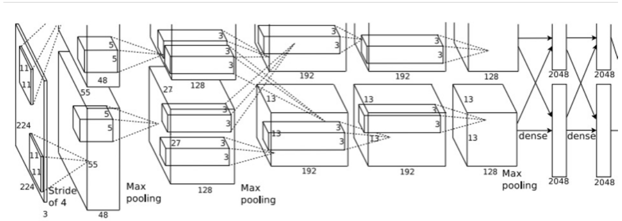
DOG, DOG, CAT

Multiple Object

# Object Detection: Single Object
## (Classification + Localization)




This image is CC0 public domain




**Vector:** 4096

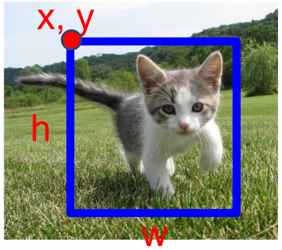
**Fully Connected:** 4096 to 1000

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

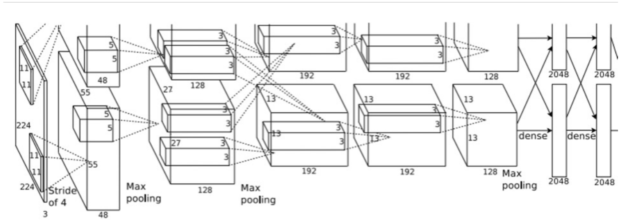
**Fully Connected:** 4096 to 4

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

# Object Detection: Single Object
(Classification + Localization)




This image is CC0 public domain




**Vector:** 4096

**Fully Connected:** 4096 to 1000

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

**Softmax Loss**

**Correct label:**
Cat

**Fully Connected:** 4096 to 4

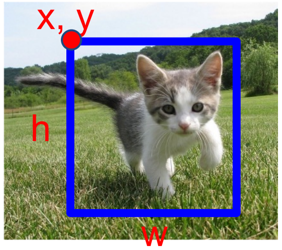
**Box Coordinates**
(x, y, w, h)

**L2 Loss**

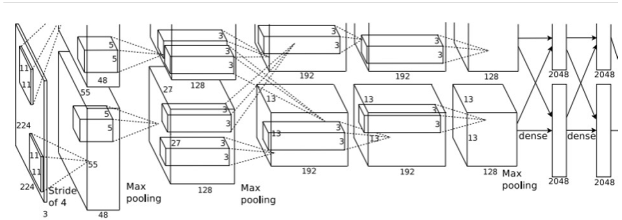
**Correct box:**
(x’, y’, w’, h’)

Treat localization as a regression problem!

# Object Detection: Single Object
(Classification + Localization)




This image is CC0 public domain




**Vector:** 4096

**Fully Connected:** 4096 to 1000

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

**Correct label:**
Cat

**Softmax Loss**

Multitask Loss

+ → **Loss**

**Fully Connected:** 4096 to 4

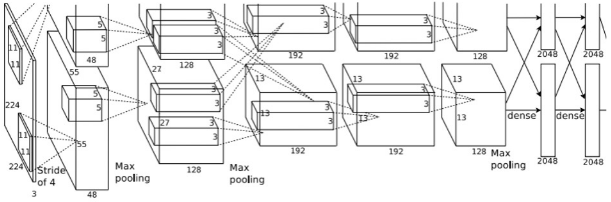
**Box Coordinates**
(x, y, w, h)

**L2 Loss**

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

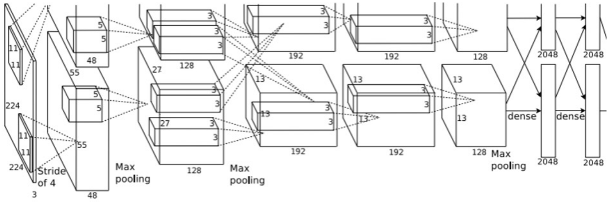
Treat localization as a regression problem!

# Object Detection: Multiple Objects



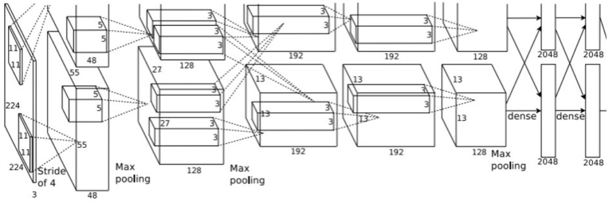


CAT: (x, y, w, h)





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

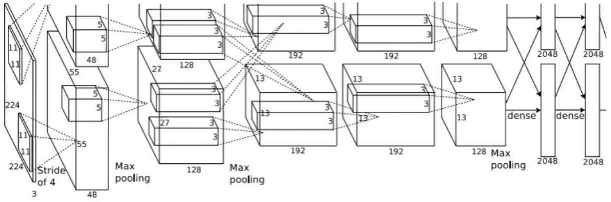




DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
….

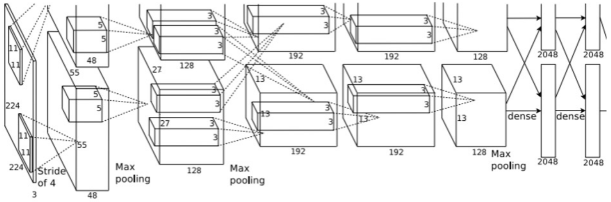
# Object Detection: Multiple Objects

Each image needs a different number of outputs!

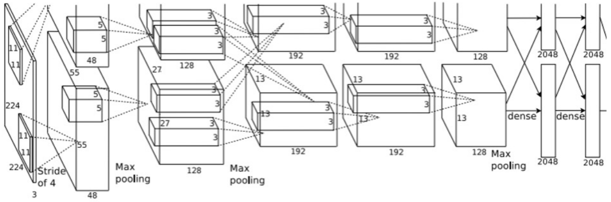

CAT: (x, y, w, h)

4 numbers


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

12 numbers

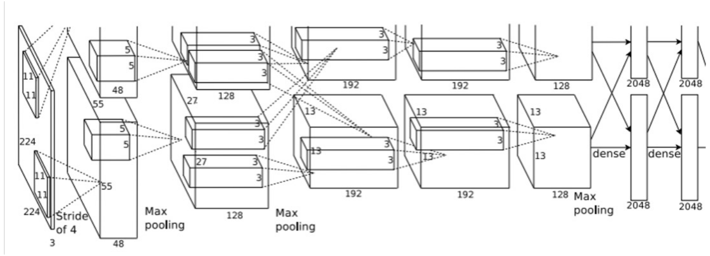




DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
….

Many numbers!

# Object Detection: Multiple Objects

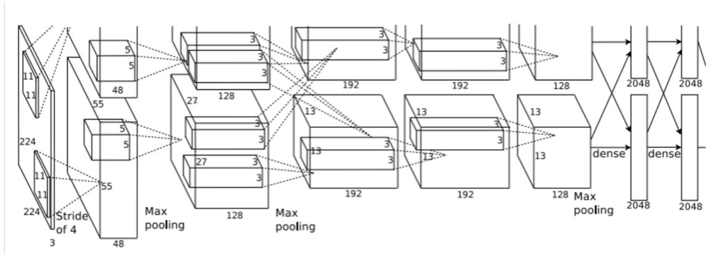
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background




Dog? NO
Cat? NO
Background? YES

# Object Detection: Multiple Objects

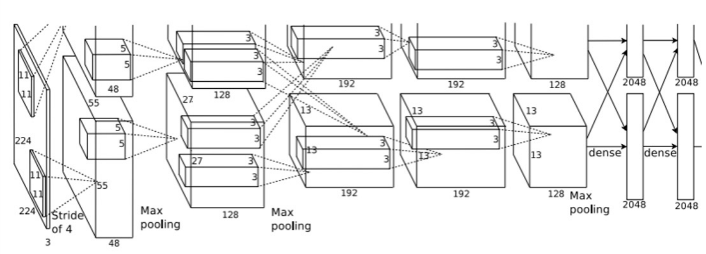
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background




Dog? YES
Cat? NO
Background? NO

# Object Detection: Multiple Objects

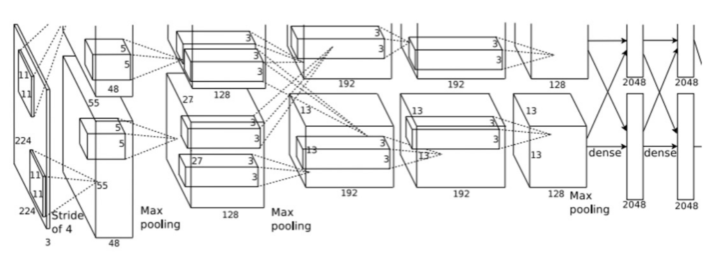
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background




Dog? YES
Cat? NO
Background? NO

# Object Detection: Multiple Objects

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

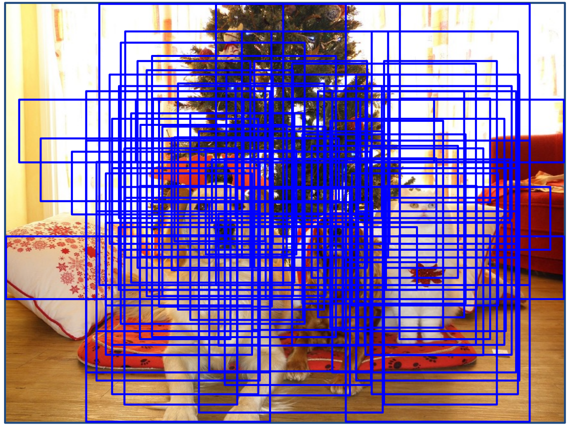


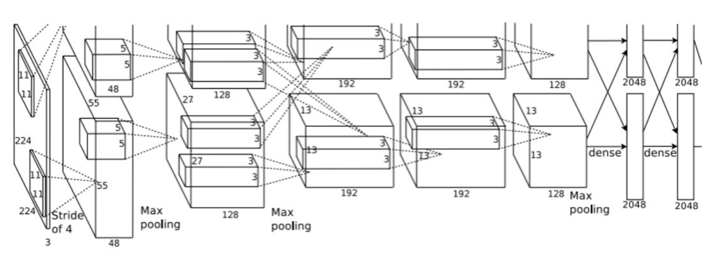

Dog? NO
Cat? YES
Background? NO

Q: What's the problem with this approach?

# Object Detection: Multiple Objects

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





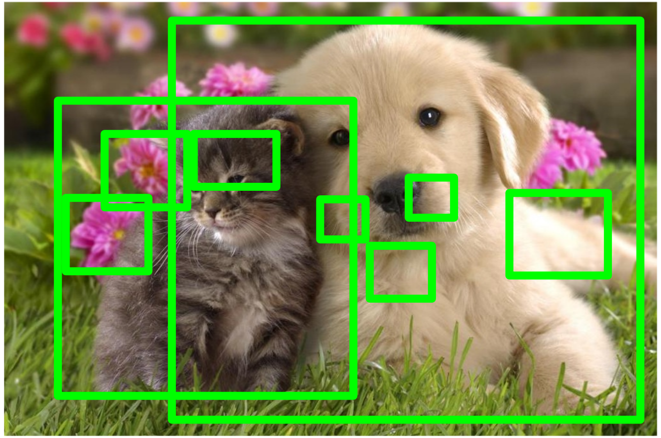
Dog? NO
Cat? YES
Background? NO

Problem: Need to apply CNN to huge number of locations, scales, and aspect ratios, very computationally expensive!

Need to find **promising regions**

# Region Proposals: Selective Search

- Find "blobby" image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 2000 region proposals in a few seconds on CPU



Alexe et al, "Measuring the objectness of image windows", TPAMI 2012
Uijlings et al, "Selective Search for Object Recognition", IJCV 2013
Cheng et al, "BING: Binarized normed gradients for objectness estimation at 300fps", CVPR 2014
Zitnick and Dollar, "Edge boxes: Locating object proposals from edges", ECCV 2014

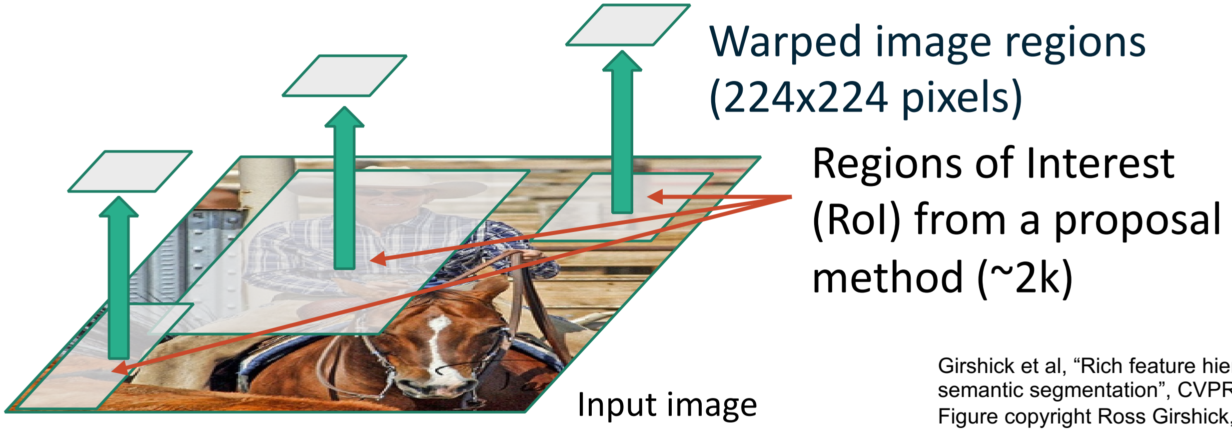
# R-CNN

Input image

# R-CNN

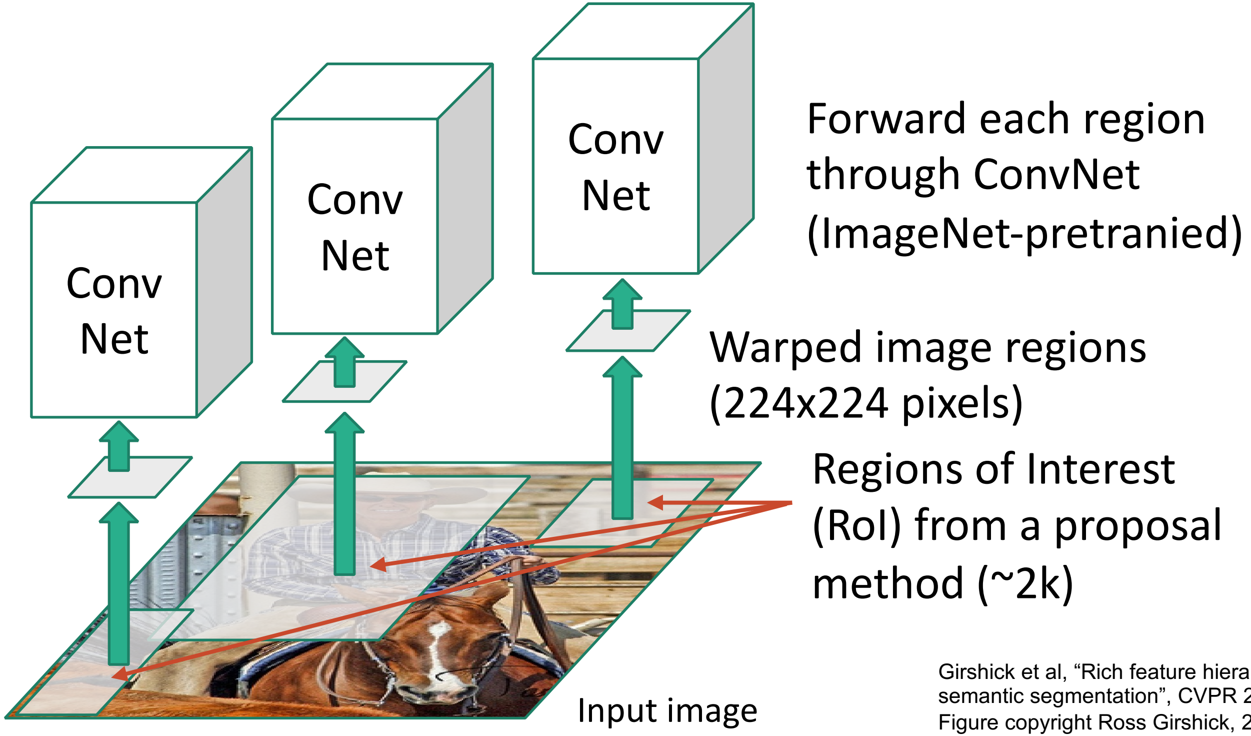

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

# R-CNN



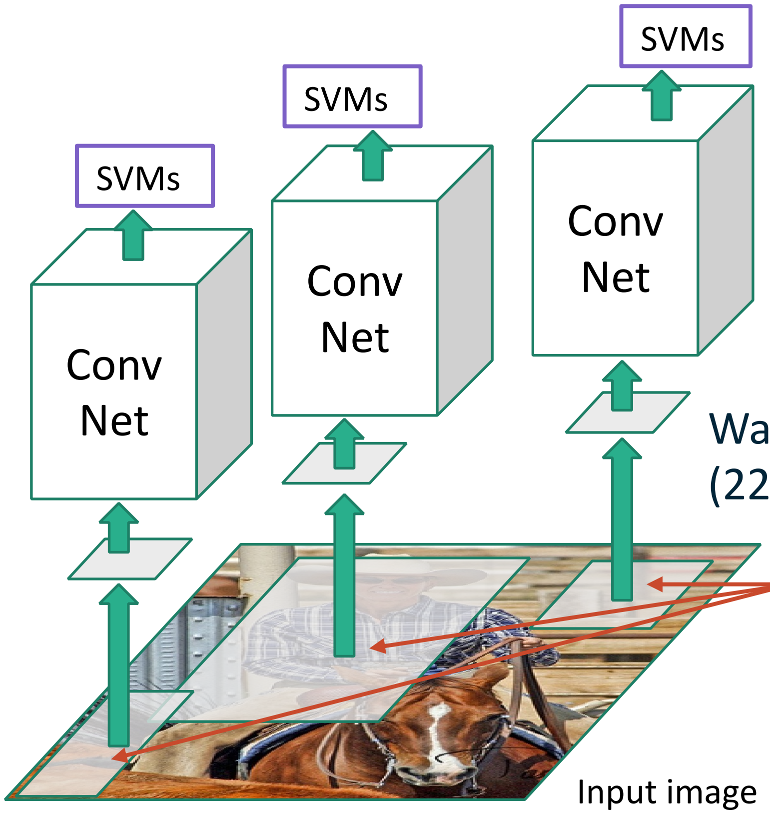

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

# R-CNN




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

# R-CNN




Classify regions with SVMs

Forward each region through ConvNet (ImageNet-pretranied)

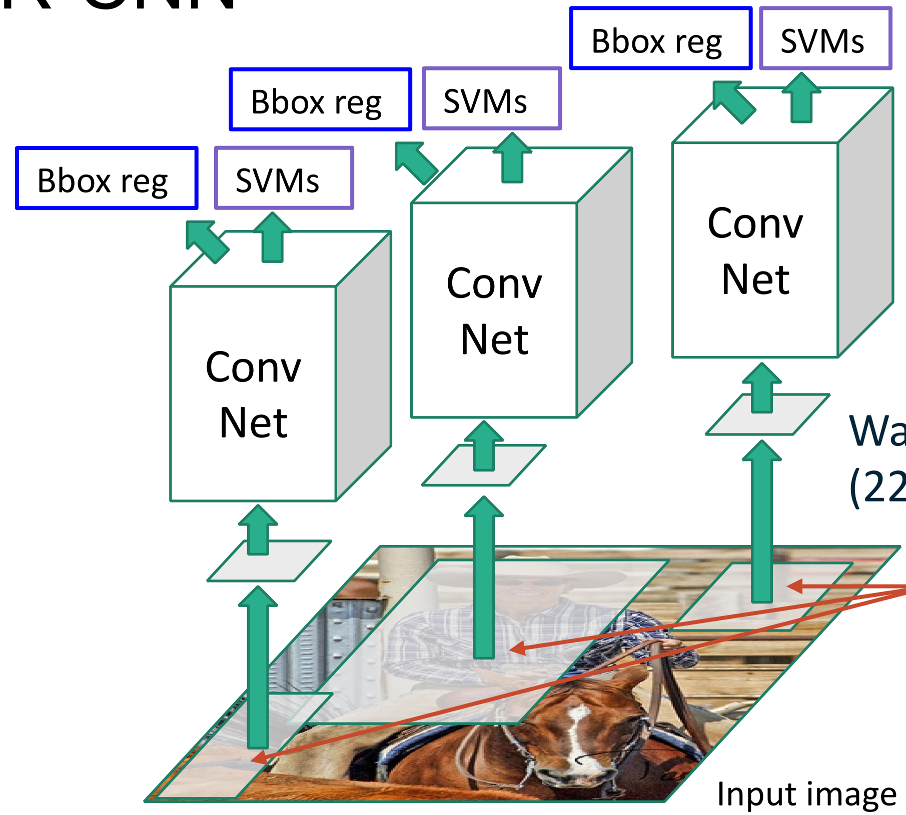
Warped image regions (224x224 pixels)

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

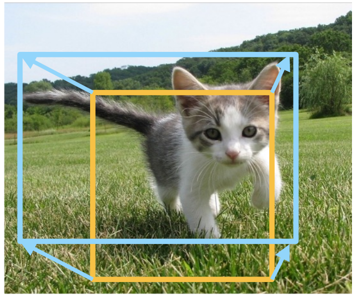
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

# R-CNN

Predict "corrections" to the RoI: 4 numbers: (dx, dy, dw, dh)




Classify regions with SVMs



Forward each region through ConvNet (ImageNet-pretranied)

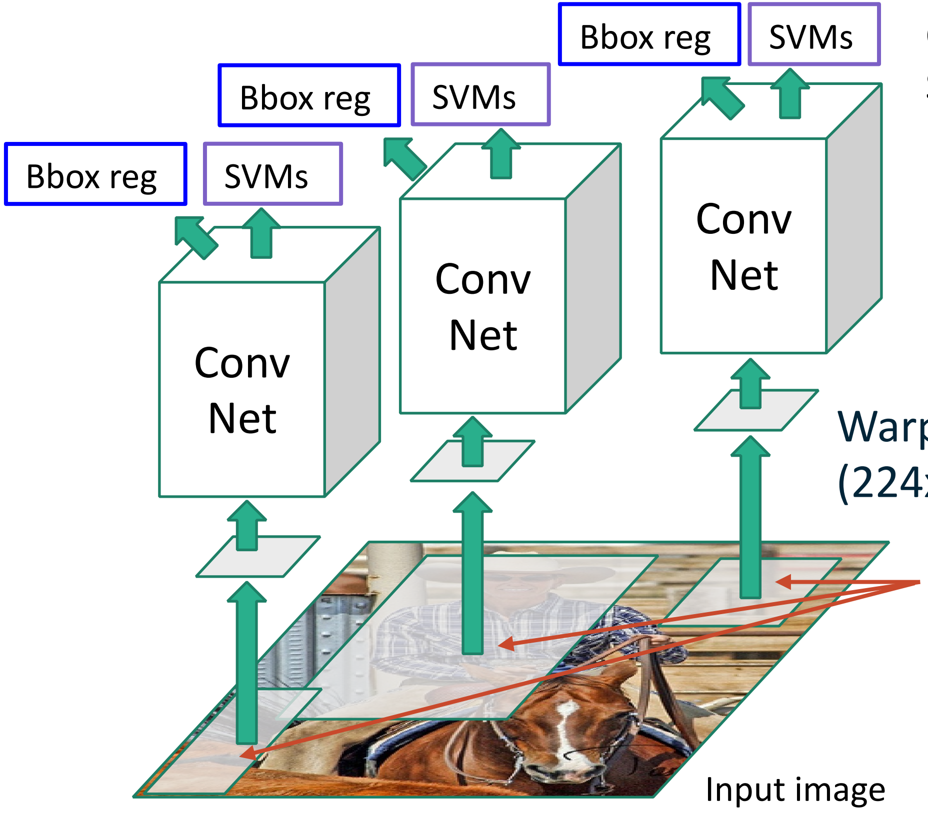
Warped image regions (224x224 pixels)

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

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

# R-CNN

Predict “corrections” to the RoI: 4 numbers: (dx, dy, dw, dh)




Classify regions with SVMs

Forward each region through ConvNet

Warped image regions (224x224 pixels)

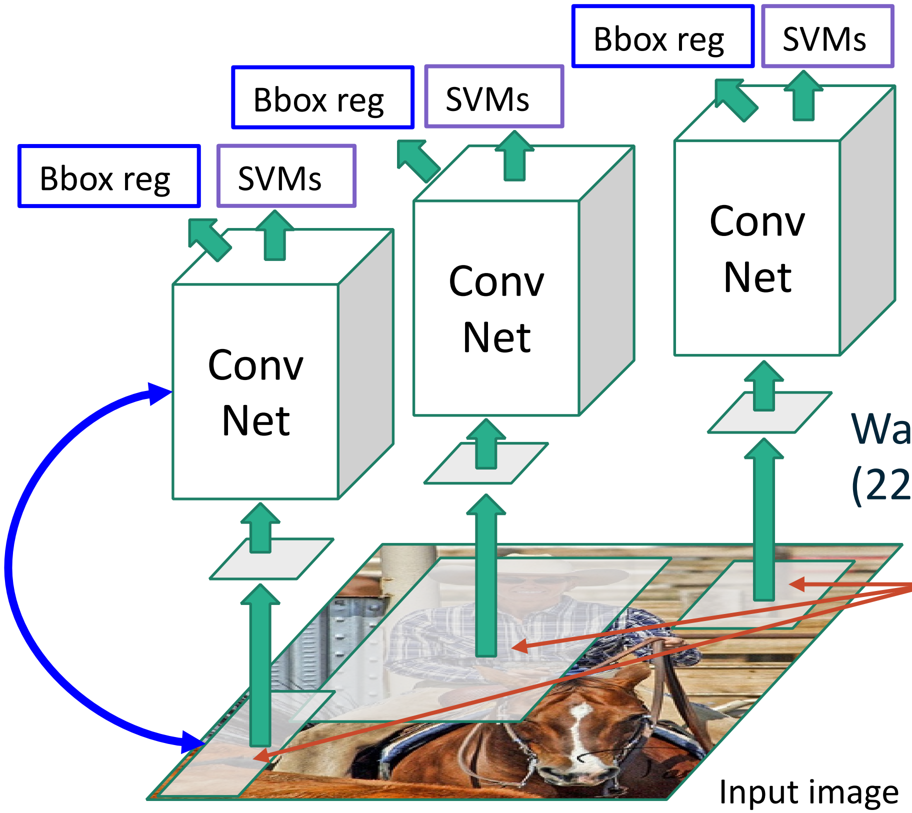
Regions of Interest (RoI) from a proposal method (~2k)

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

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

# "Slow" R-CNN

Predict "corrections" to the RoI: 4 numbers: (dx, dy, dw, dh)




Classify regions with SVMs

Forward each region through ConvNet

Warped image regions (224x224 pixels)

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

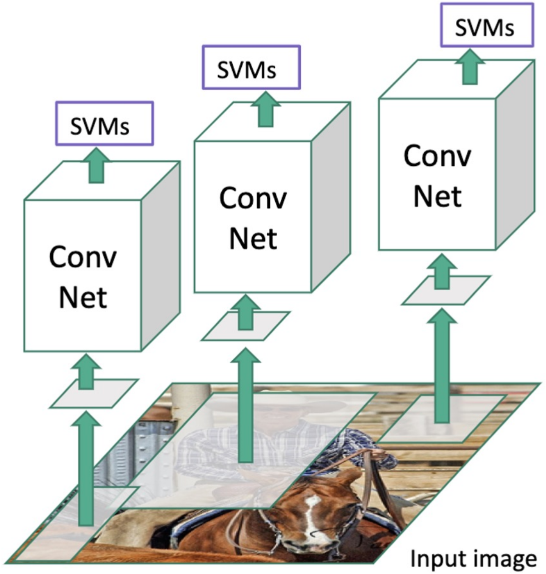
**Problem**: Very slow! Need to do ~2k independent forward passes for each image!

**Idea:** Pass the image through convnet before cropping! Crop the conv feature instead!

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

# Fast R-CNN

"Slow" R-CNN

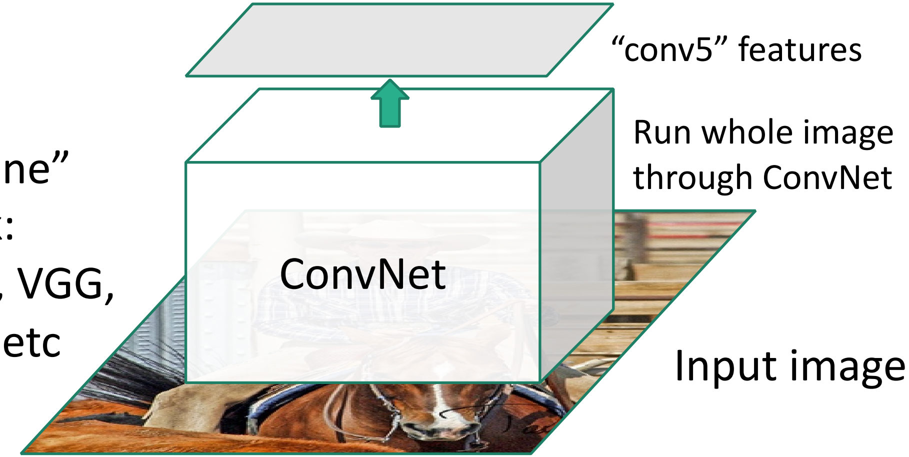


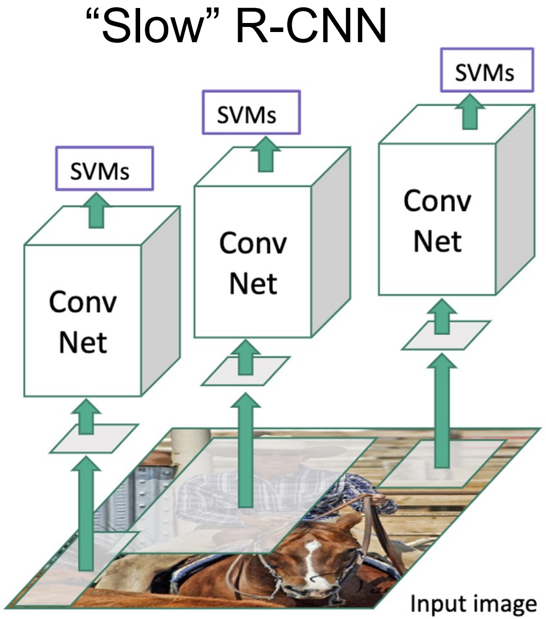

Input image

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

# Fast R-CNN

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



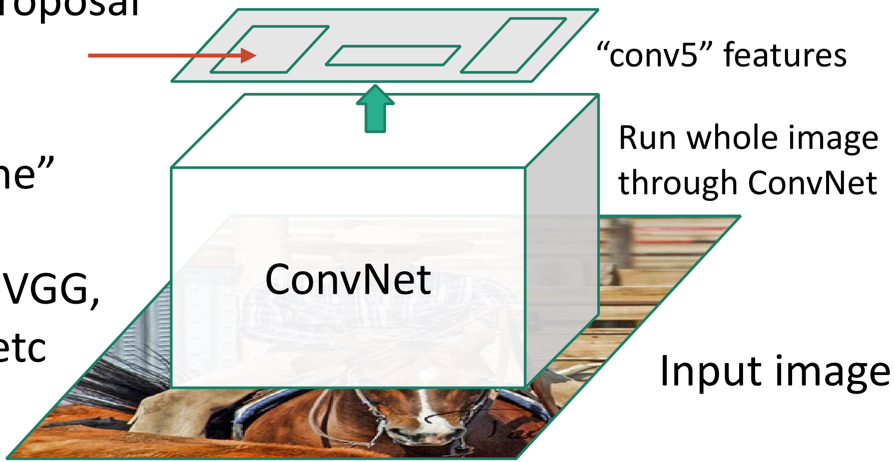




Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

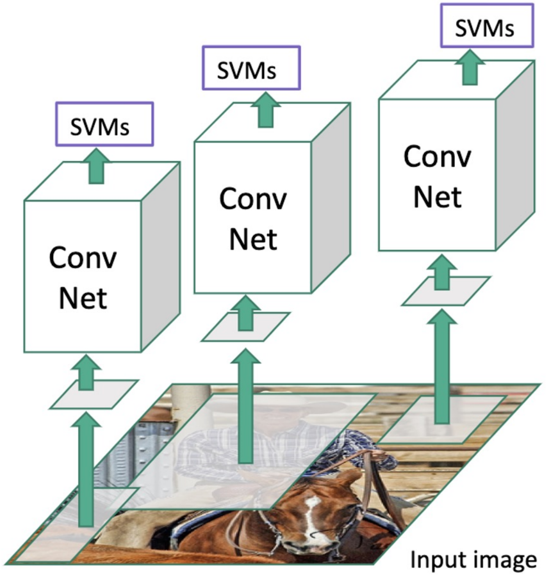
# Fast R-CNN

Regions of Interest (RoIs) from a proposal method

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



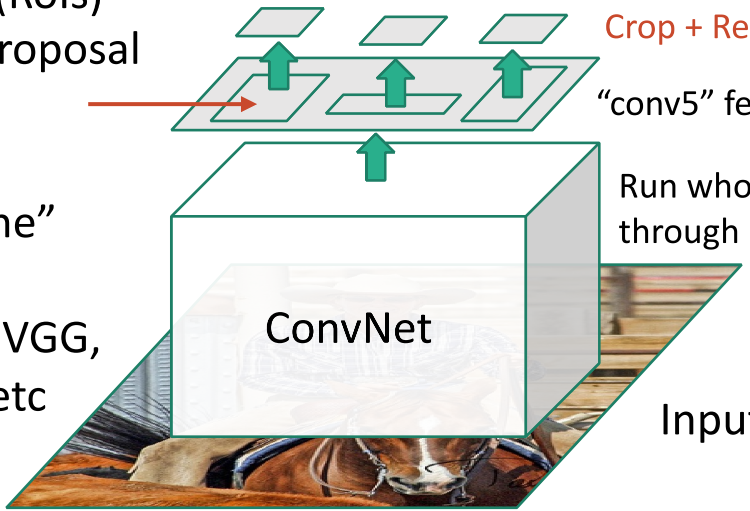

"Slow" R-CNN



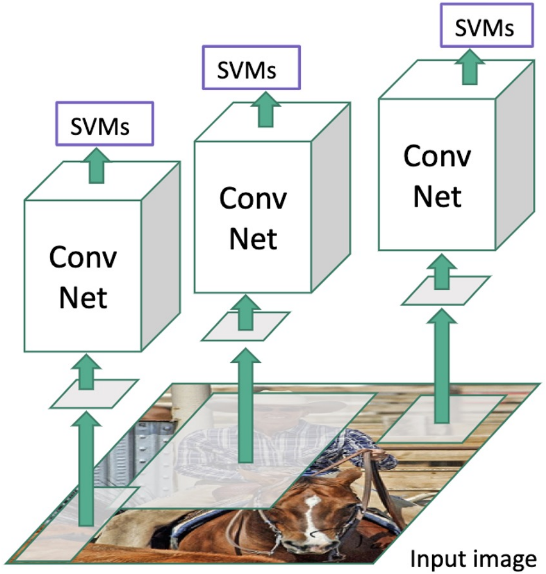
# Fast R-CNN

Regions of Interest (RoIs) from a proposal method

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

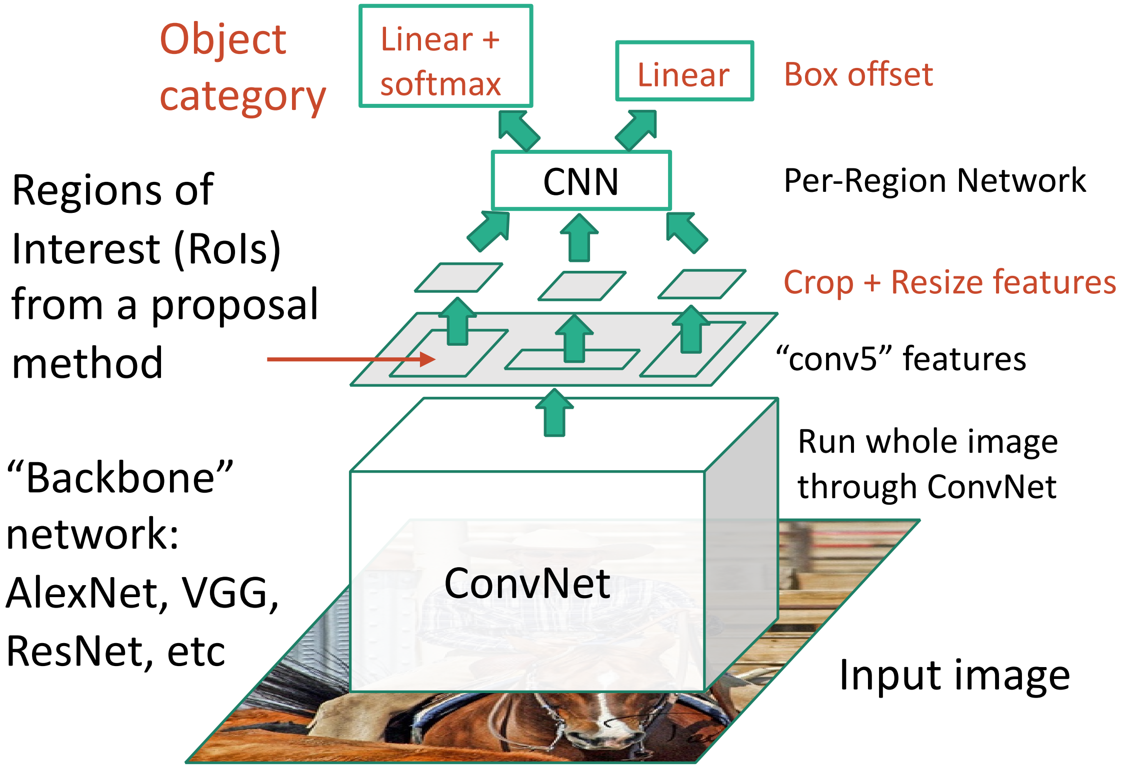


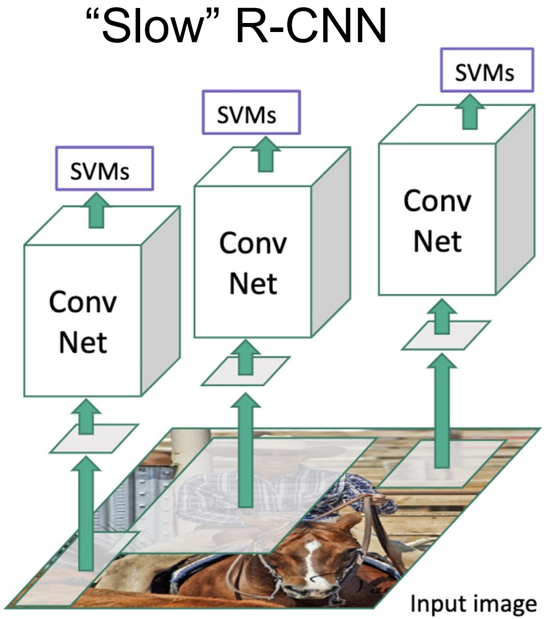

"Slow" R-CNN



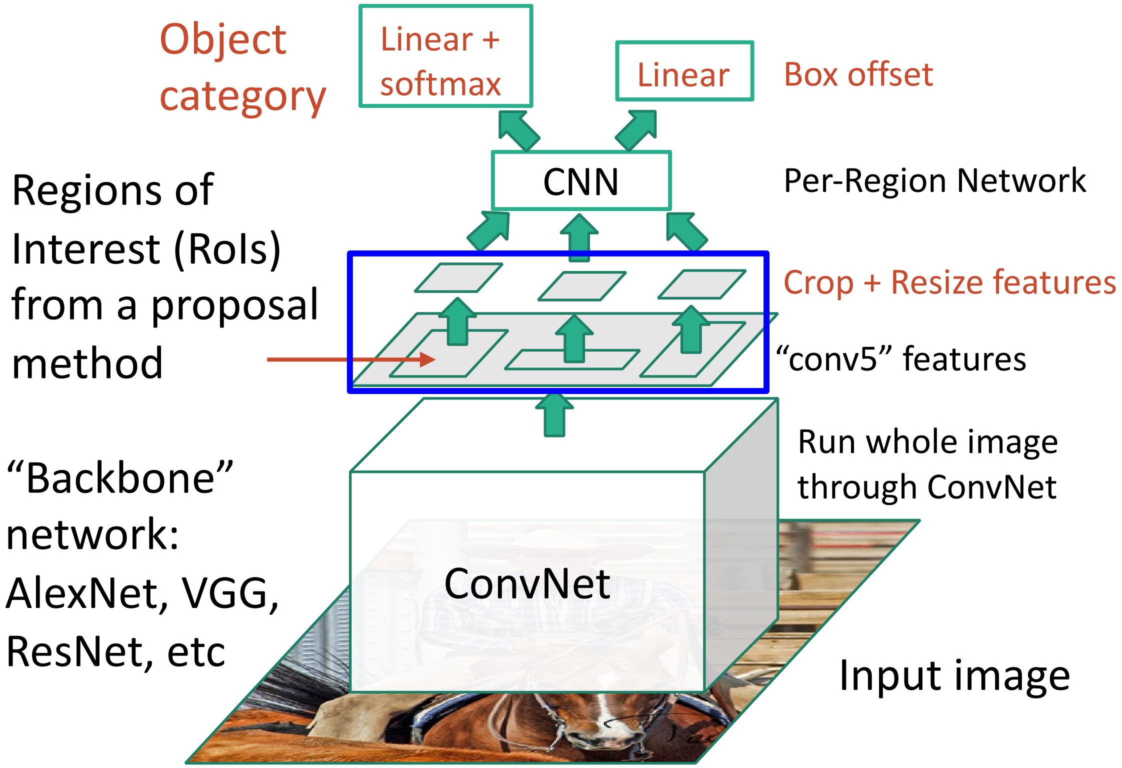

Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

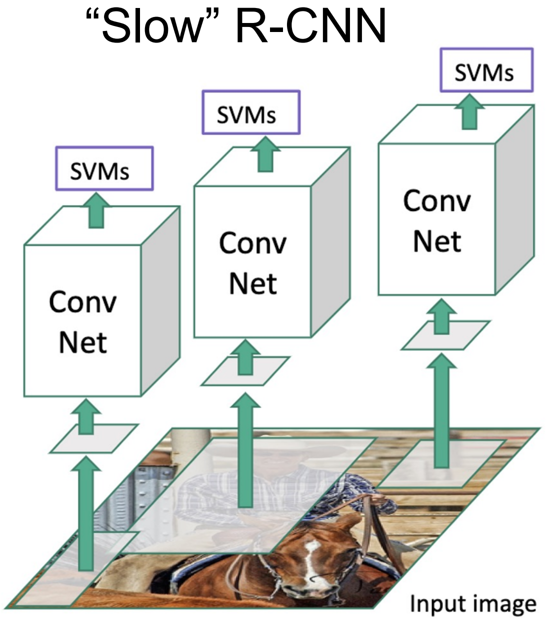
# Fast R-CNN


Object category
Linear + softmax
Linear
Box offset
CNN
Per-Region Network
Regions of Interest (RoIs) from a proposal method
Crop + Resize features
"conv5" features
Run whole image through ConvNet
"Backbone" network: AlexNet, VGG, ResNet, etc
ConvNet
Input image


"Slow" R-CNN
SVMs
SVMs
SVMs
Conv Net
Conv Net
Conv Net
Input image

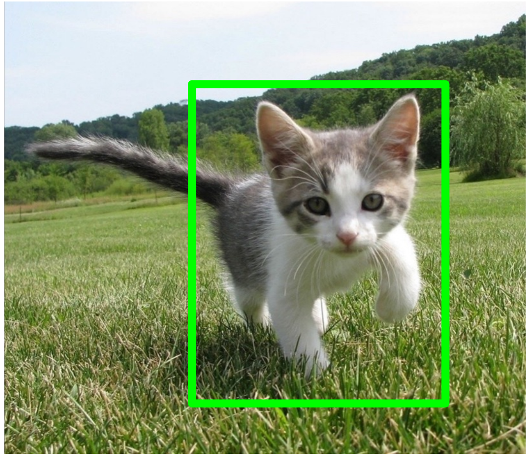
# Fast R-CNN




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

# Cropping Features: RoI Pool



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

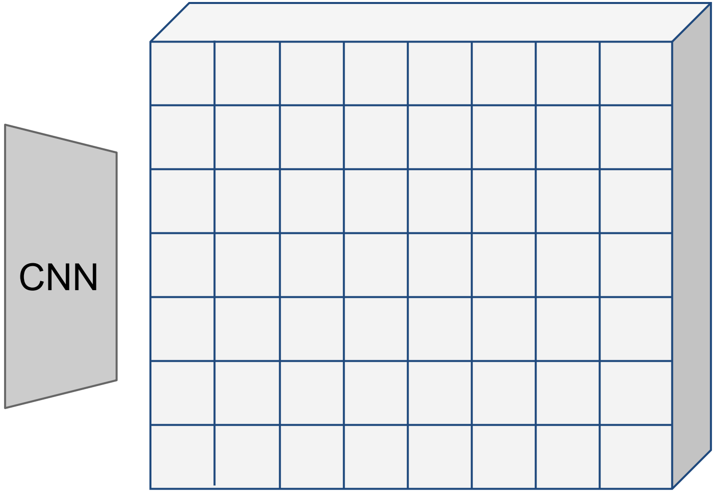
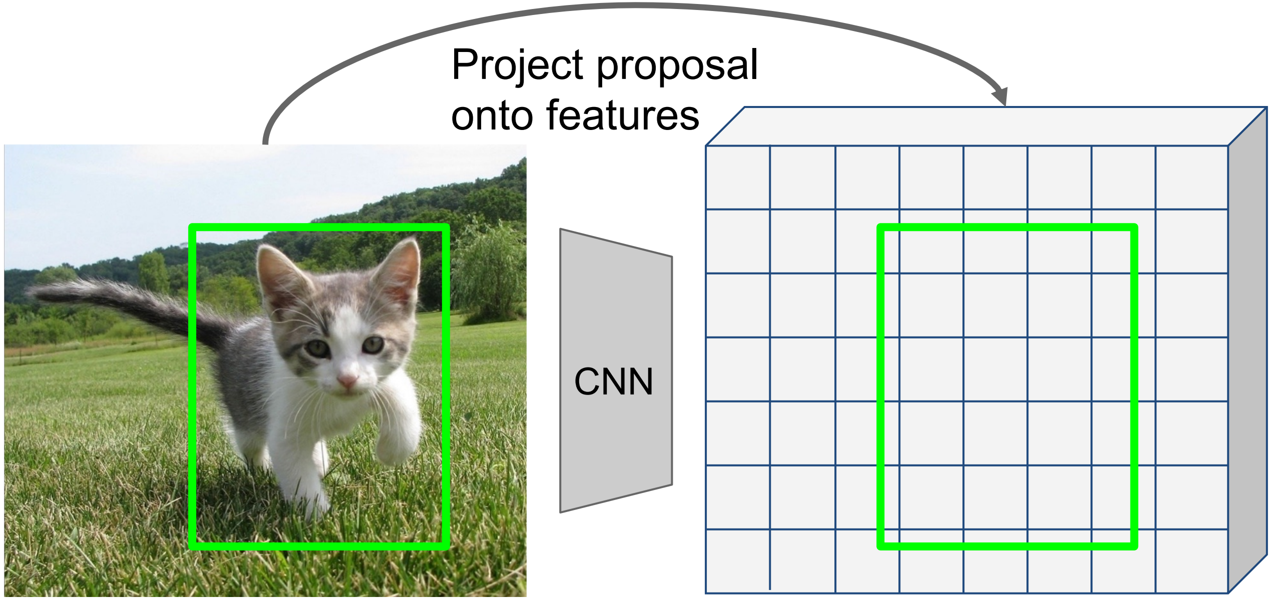



Image features: C x H x W
(e.g. 512 x 20 x 15)

Girshick, “Fast R-CNN”, ICCV 2015.

Girshick, “Fast R-CNN”, ICCV 2015.

# Cropping Features: RoI Pool



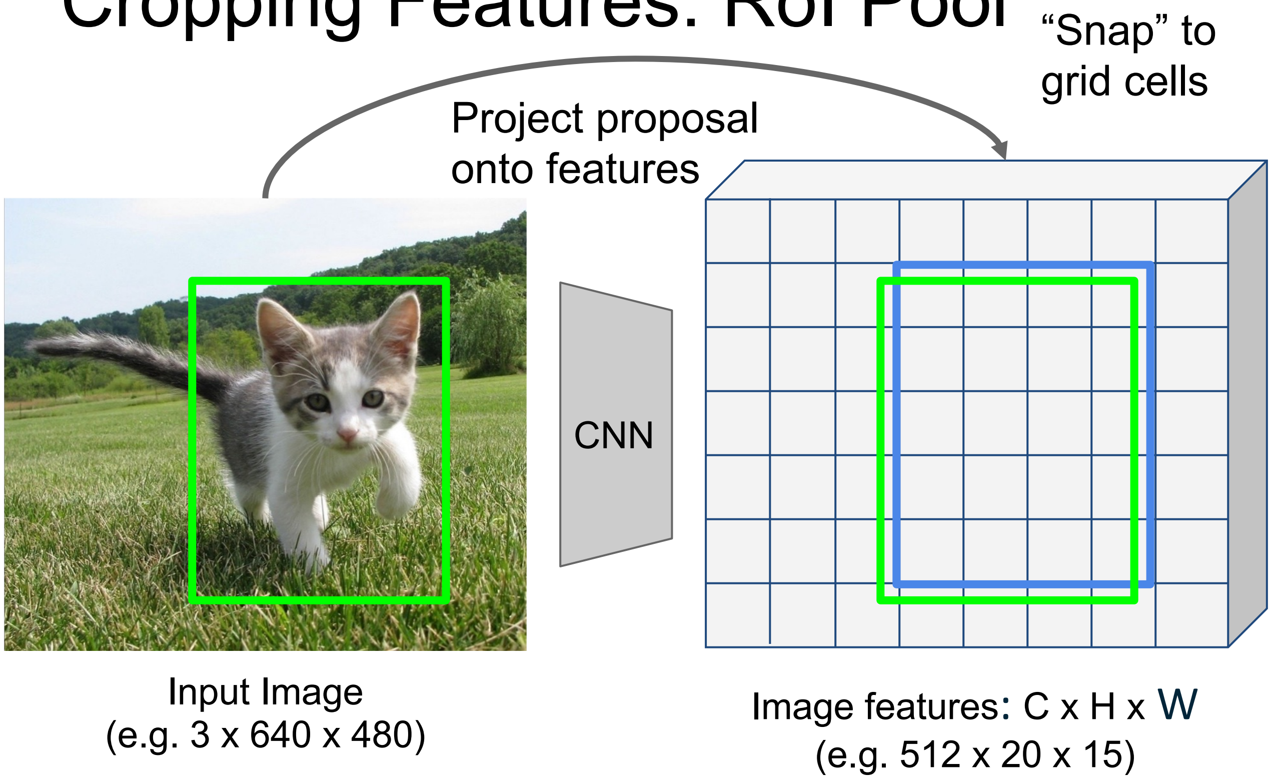

Input Image
(e.g. 3 x 640 x 480)

Image features: C x H x W
(e.g. 512 x 20 x 15)

Girshick, “Fast R-CNN”, ICCV 2015.

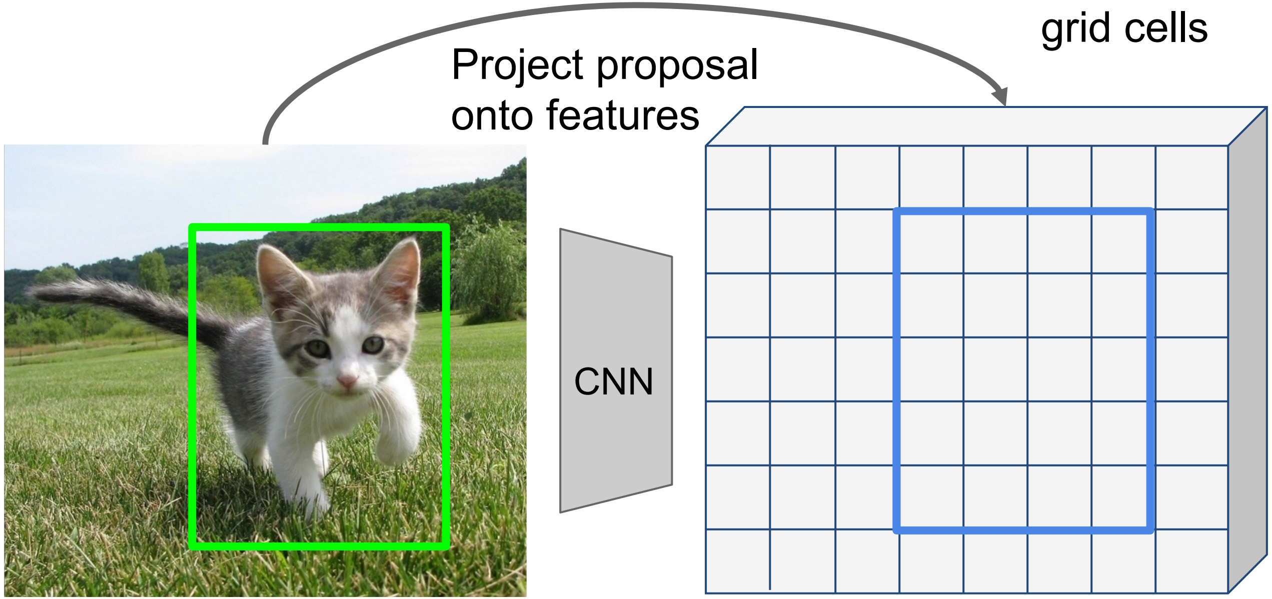
Girshick, “Fast R-CNN”, ICCV 2015.

# Cropping Features: RoI Pool




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

# Cropping Features: RoI Pool



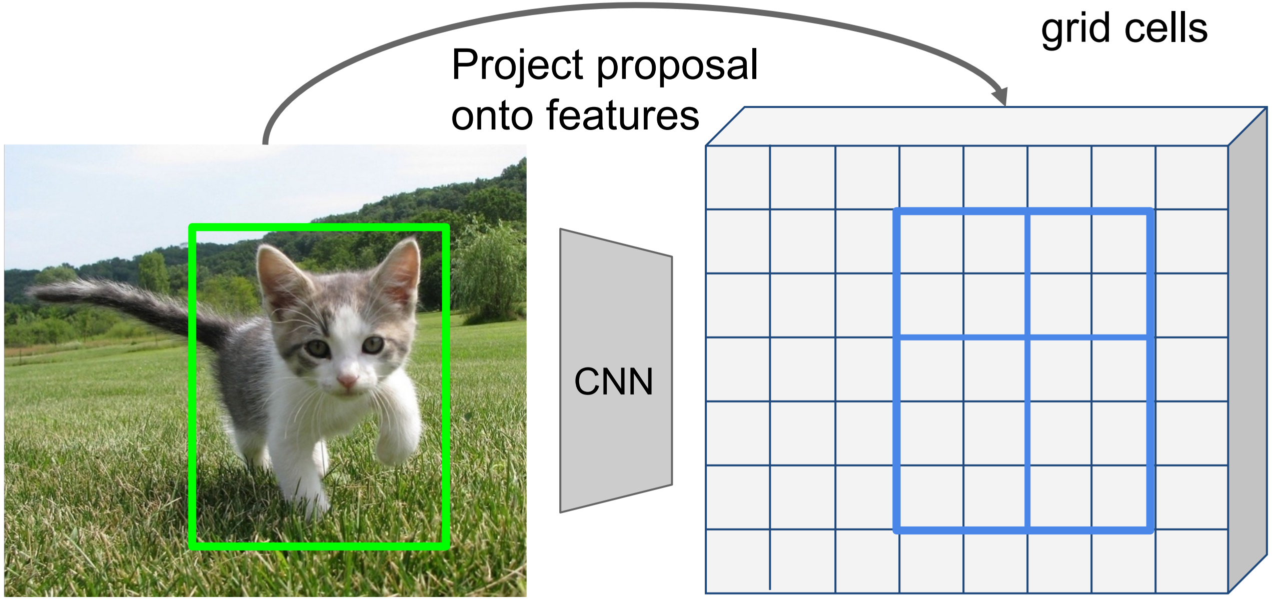

Input Image
(e.g. 3 x 640 x 480)

Image features: C x H x W
(e.g. 512 x 20 x 15)

Q: how do we resize the 512 x 20 x 15 region to, e.g., a 512 x 2 x 2 tensor?.

Girshick, “Fast R-CNN”, ICCV 2015.

# Cropping Features: RoI Pool




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

Image features: C x H x W
(e.g. 512 x 20 x 15)

Divide into 2x2 grid of (roughly) equal subregions

Q: how do we resize the 512 x 20 x 15 region to, e.g., a 512 x 2 x 2 tensor?.

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

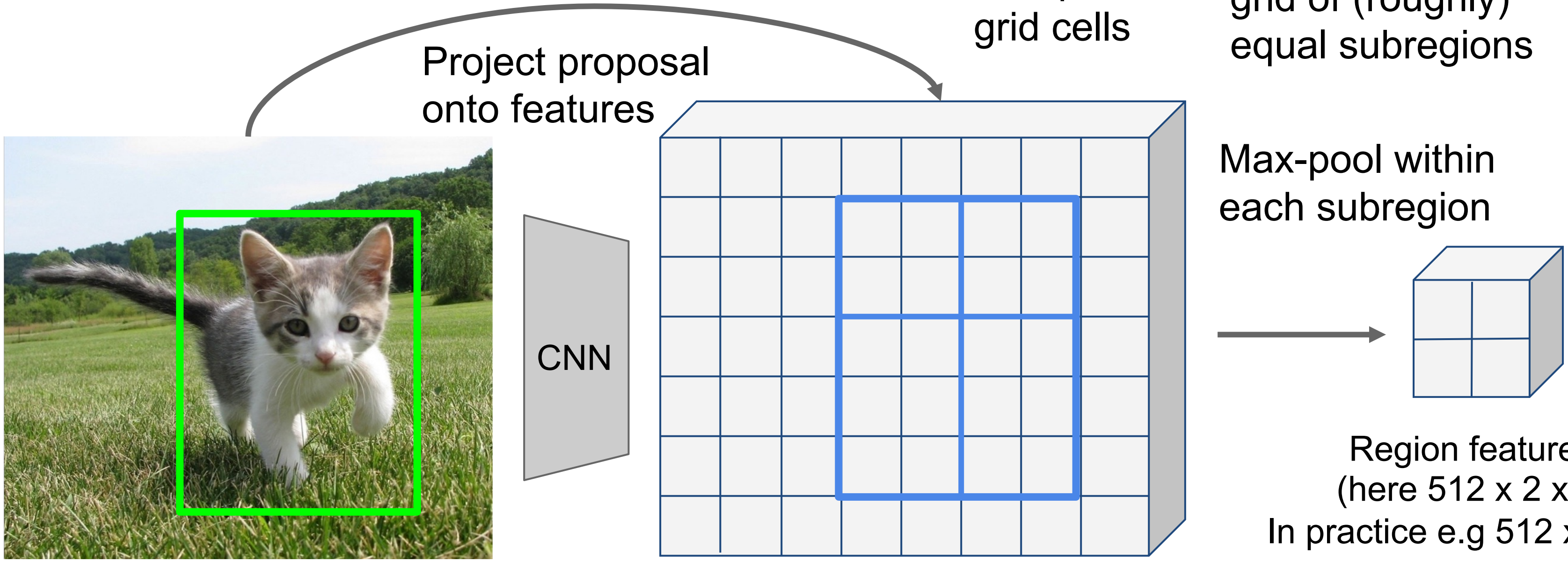
# Cropping Features: RoI Pool

"Snap" to grid cells

Divide into 2x2 grid of (roughly) equal subregions

Project proposal onto features

Max-pool within each subregion




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

Image features: C x H x W
(e.g. 512 x 20 x 15)

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

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

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

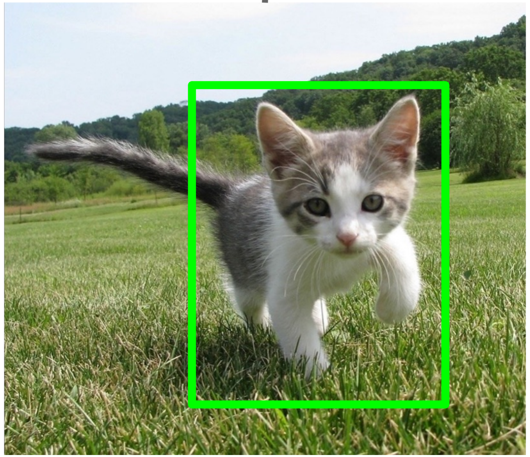
# Cropping Features: RoI Pool

Project proposal onto features

"Snap" to grid cells

Divide into 2x2 grid of (roughly) equal subregions

Max-pool within each subregion



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

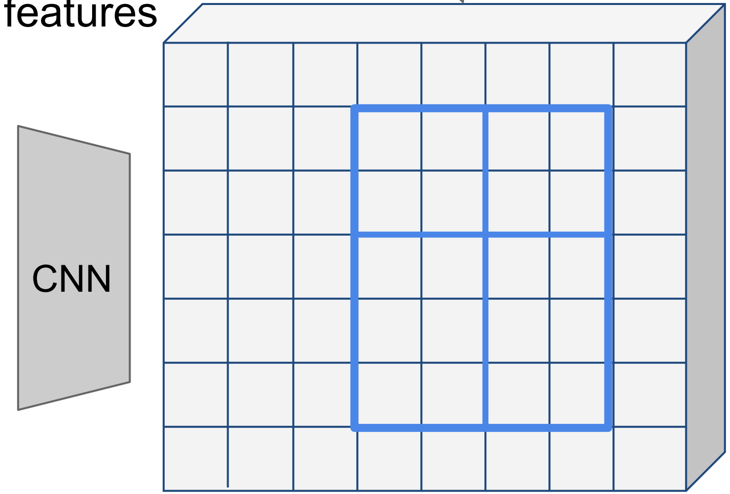
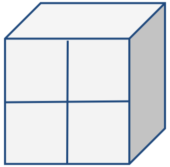



Image features: C x H x W
(e.g. 512 x 20 x 15)



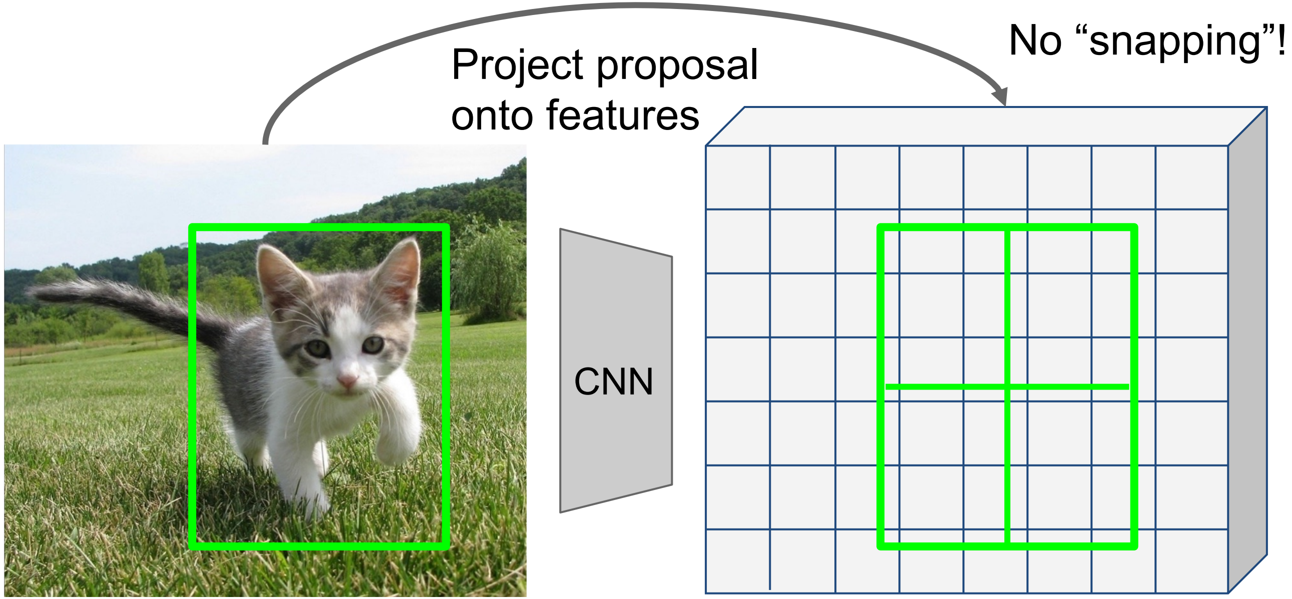
Region features
(here 512 x 2 x 2;
In practice e.g 512 x 7 x 7)

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

**Problem**: Region features slightly misaligned

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

# Cropping Features: RoI Align



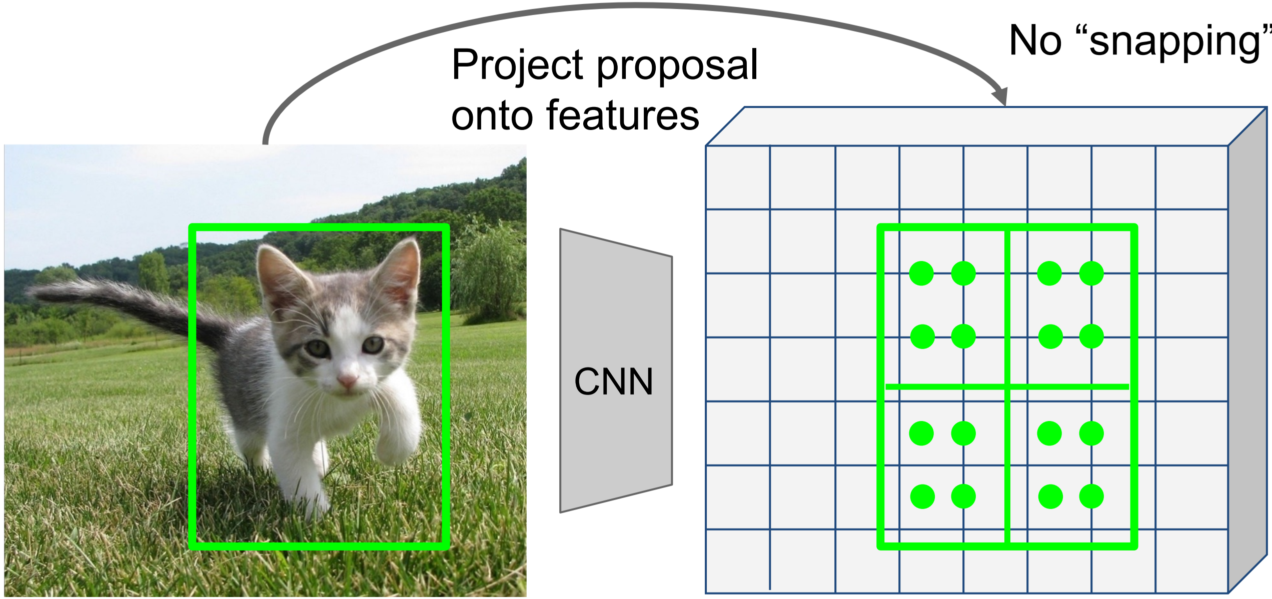

Input Image
(e.g. 3 x 640 x 480)

Image features: C x H x W
(e.g. 512 x 20 x 15)

He et al, “Mask R-CNN”, ICCV 2017

# Cropping Features: RoI Align

Sample at regular points in each subregion using bilinear interpolation




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

Image features: C x H x W
(e.g. 512 x 20 x 15)

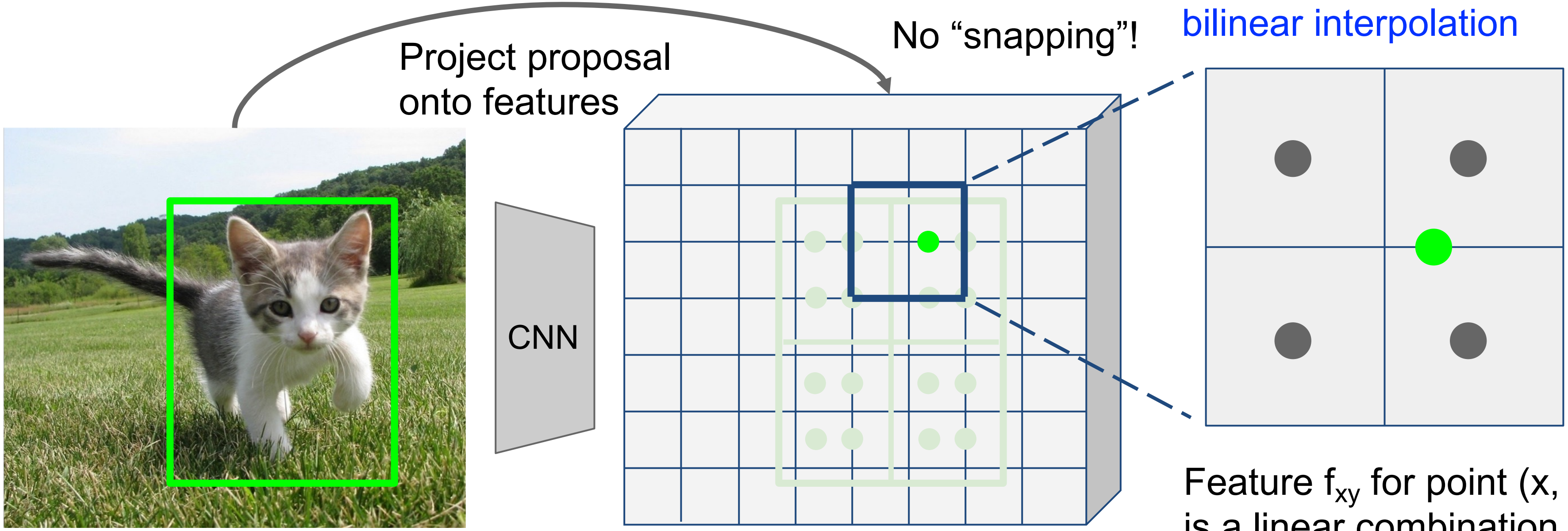
He et al, “Mask R-CNN”, ICCV 2017

# Cropping Features: RoI Align

Sample at regular points in each subregion using bilinear interpolation

No "snapping"!

Project proposal onto features




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

Image features: C x H x W
(e.g. 512 x 20 x 15)

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

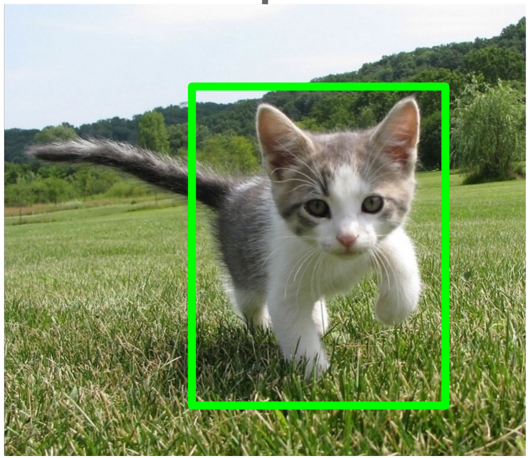
He et al, "Mask R-CNN", ICCV 2017

# Cropping Features: RoI Align

Sample at regular points in each subregion using bilinear interpolation

No "snapping"!

Project proposal onto features



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

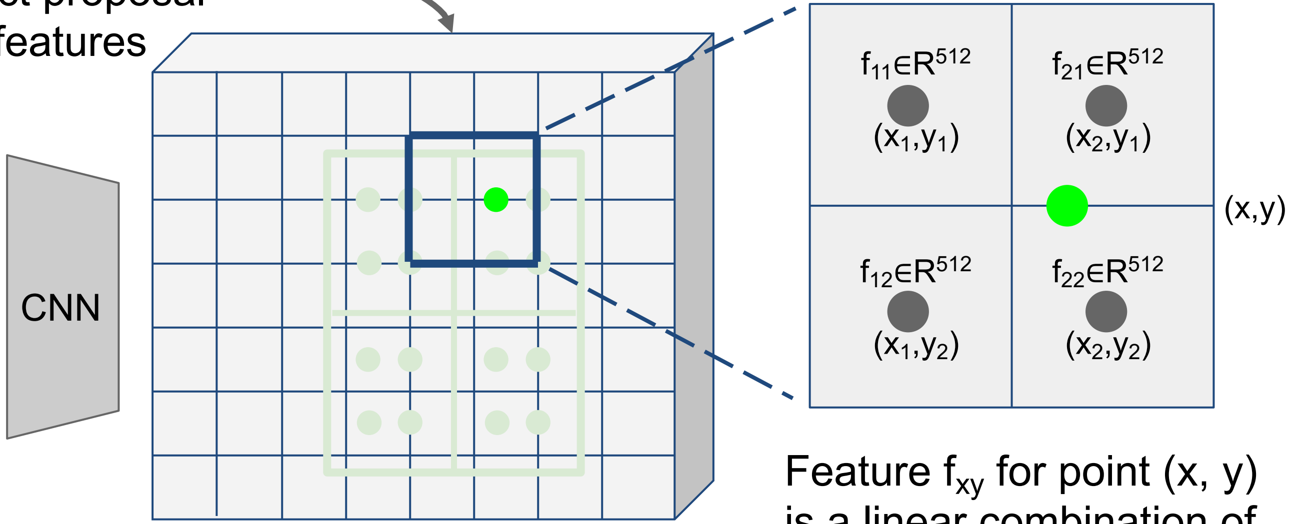



Image features: C x H x W
(e.g. 512 x 20 x 15)

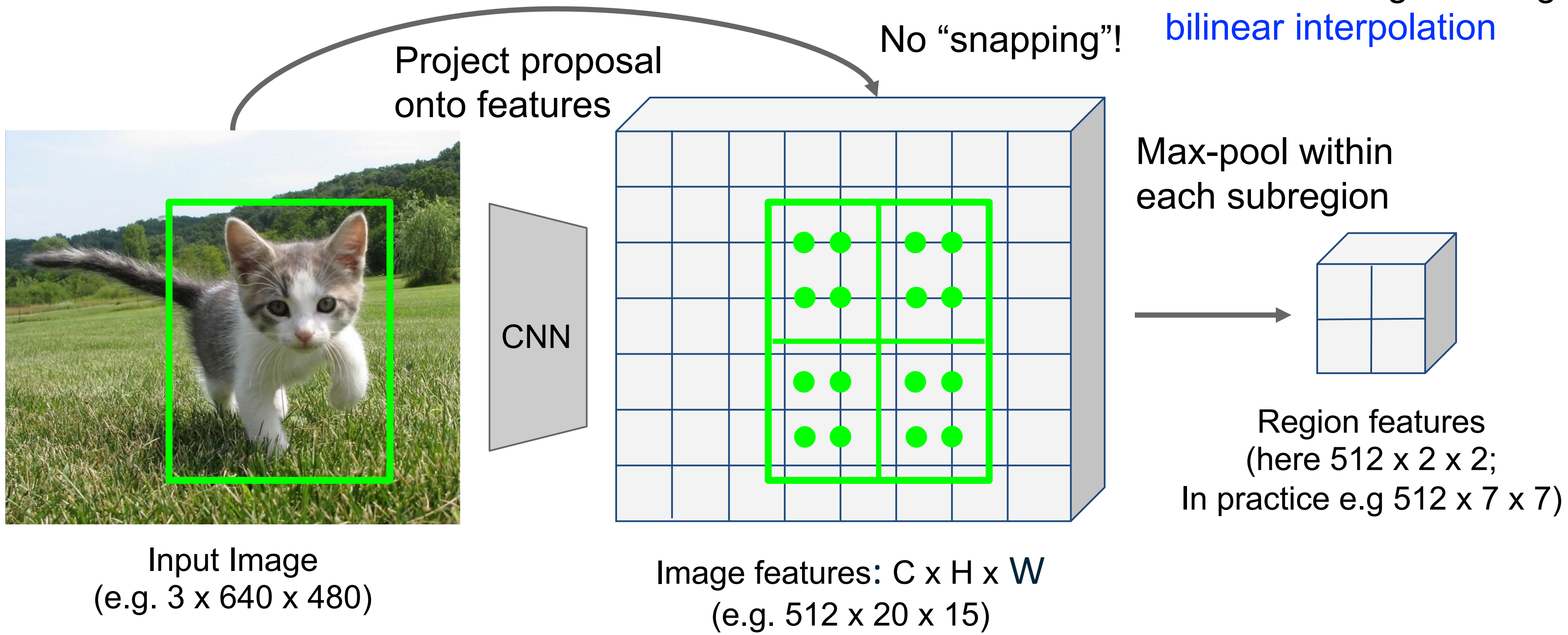
Feature $f_{xy}$ for point (x, y) is a linear combination of features at its four neighboring grid cells:

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

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

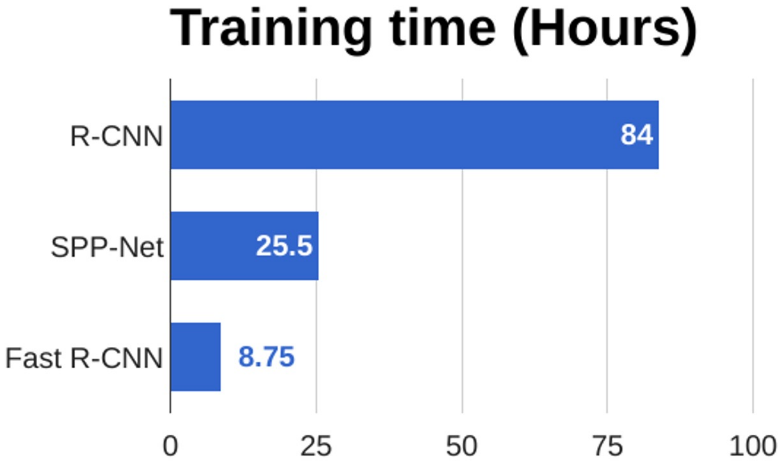
# Cropping Features: RoI Align

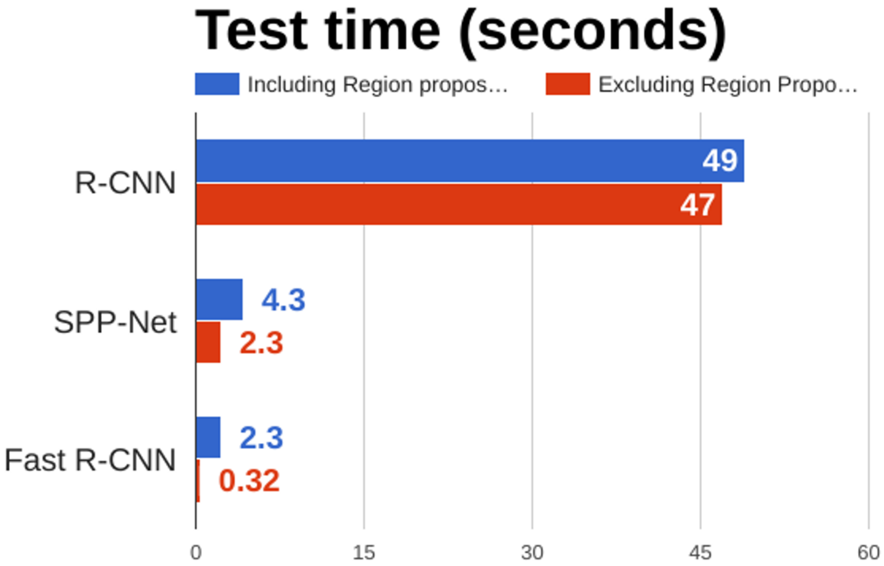
Sample at regular points in each subregion using bilinear interpolation




He et al, “Mask R-CNN”, ICCV 2017

# R-CNN vs Fast R-CNN

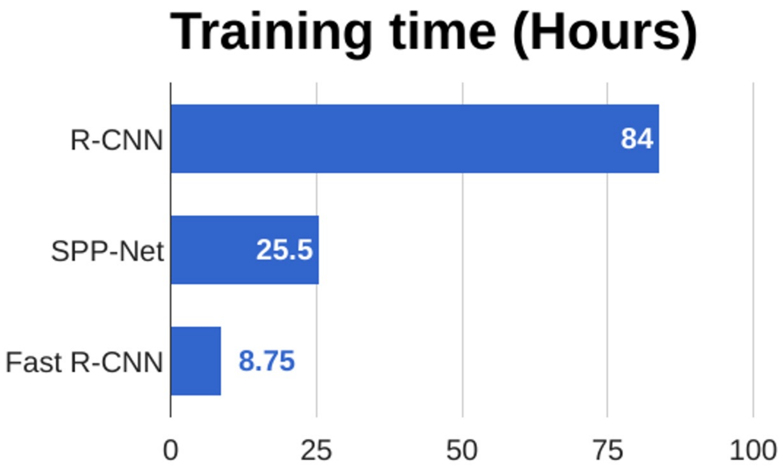


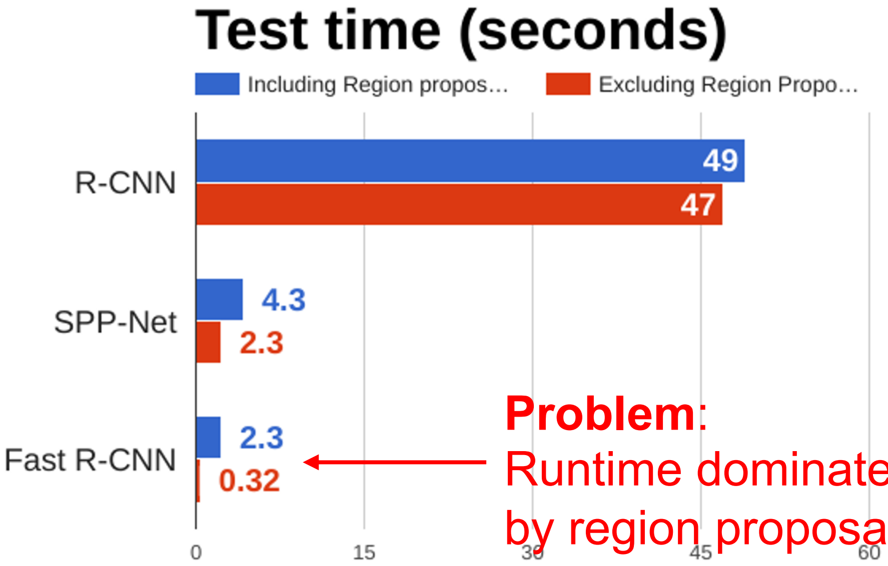





Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
He et al, “Spatial pyramid pooling in deep convolutional networks for visual recognition”, ECCV 2014
Girshick, “Fast R-CNN”, ICCV 2015

# R-CNN vs Fast R-CNN




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

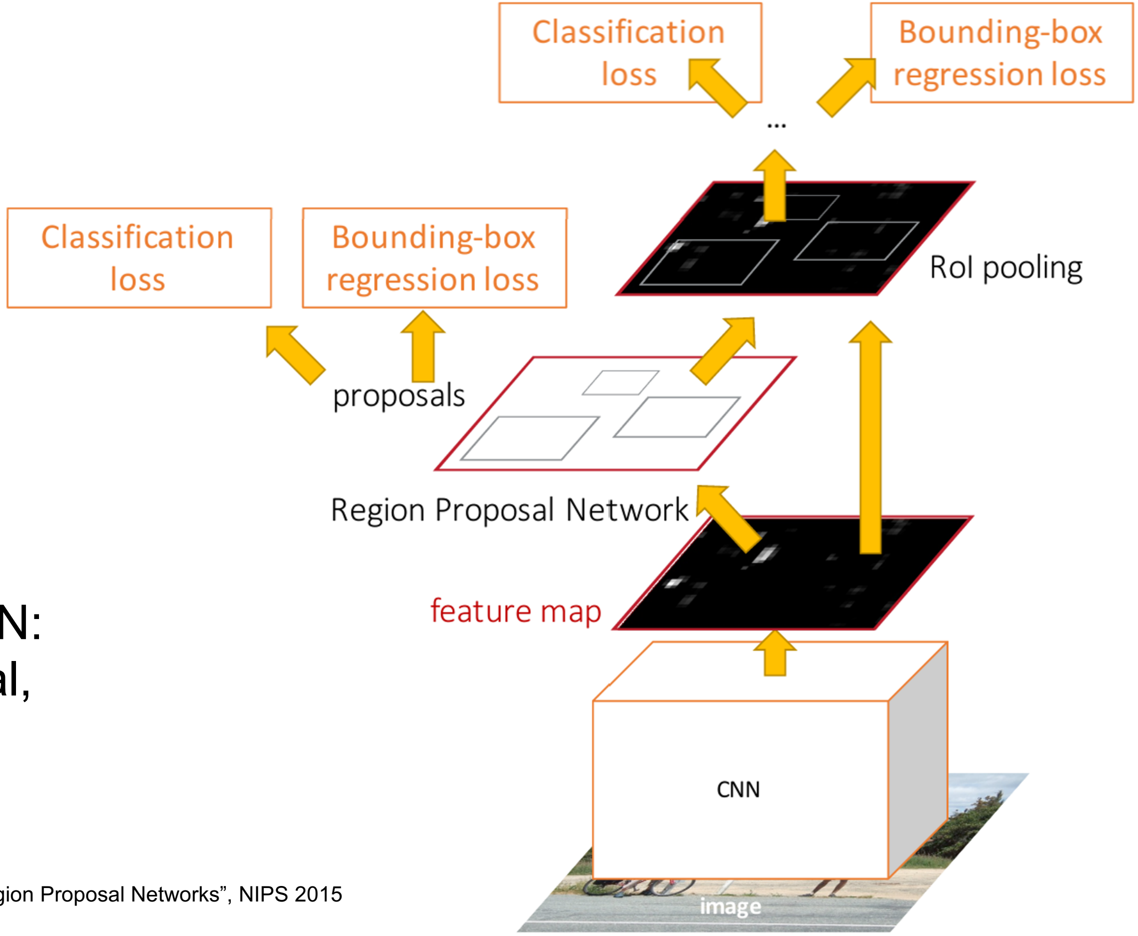

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
He et al, "Spatial pyramid pooling in deep convolutional networks for visual recognition", ECCV 2014
Girshick, "Fast R-CNN", ICCV 2015

# Fas**t**<u>**er**</u> R-CNN:

Make CNN do proposals!

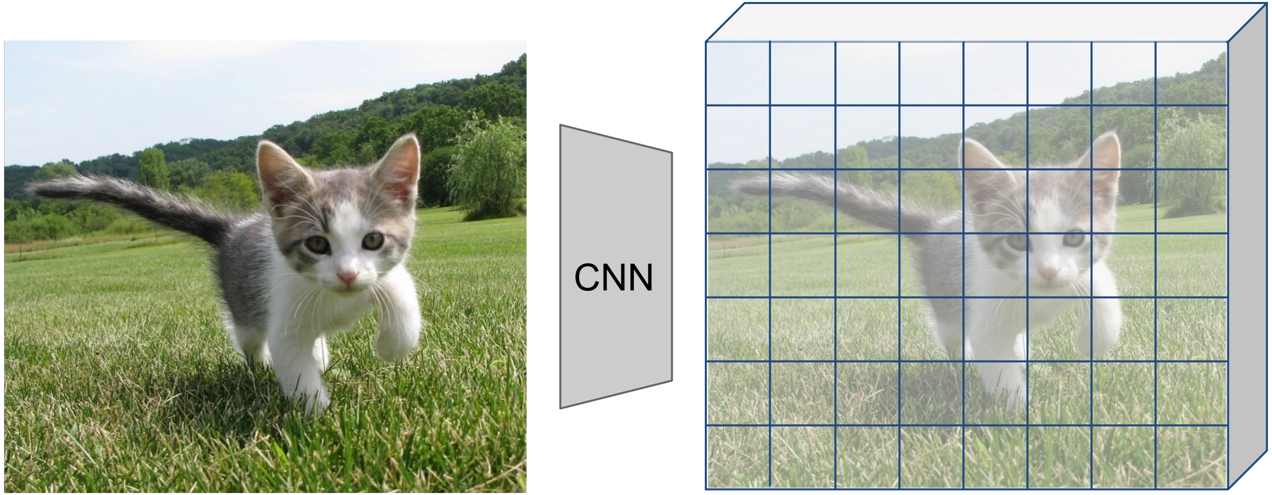
Insert **Region Proposal Network (RPN)** to predict proposals from features

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




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

# Region Proposal Network

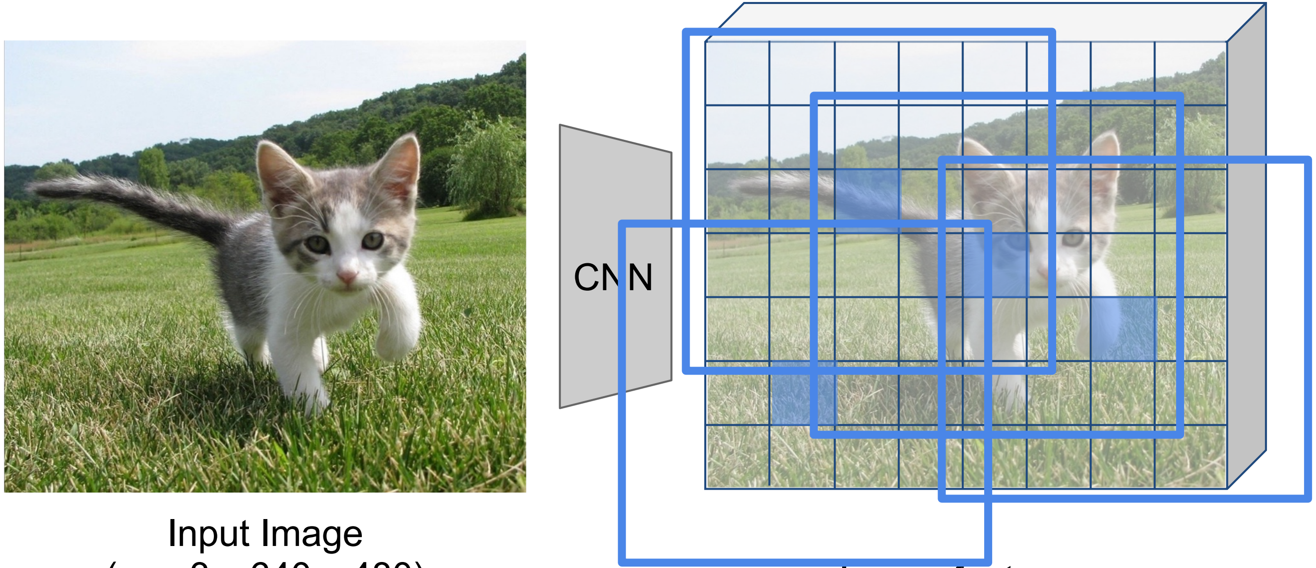



Input Image
(e.g. 3 x 640 x 480)

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

# Region Proposal Network

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

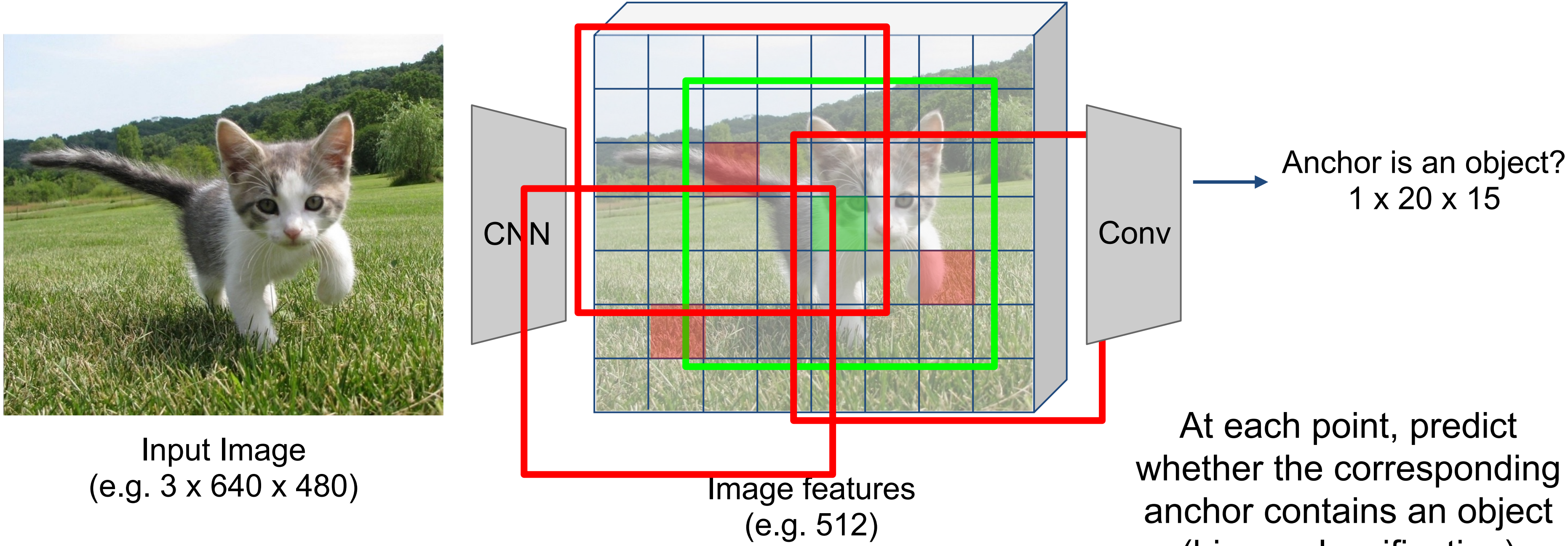



Input Image
(e.g. 3 x 640 x 480)

Image features
(e.g. 512)

# Region Proposal Network

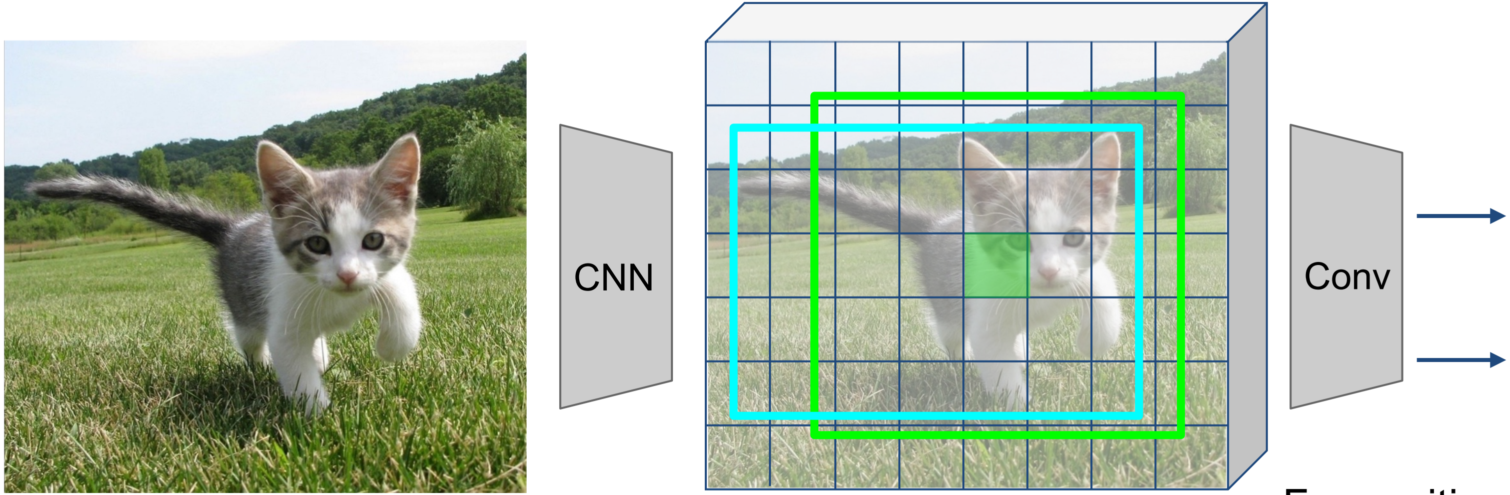
Example: 20 x 15 **anchor box** uniformly sampled on the feature map




At each point, predict whether the corresponding anchor contains an object (binary classification)

# Region Proposal Network

Example: 20 x 15 **anchor box** uniformly sampled on the feature map




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

Image features
(e.g. 512)

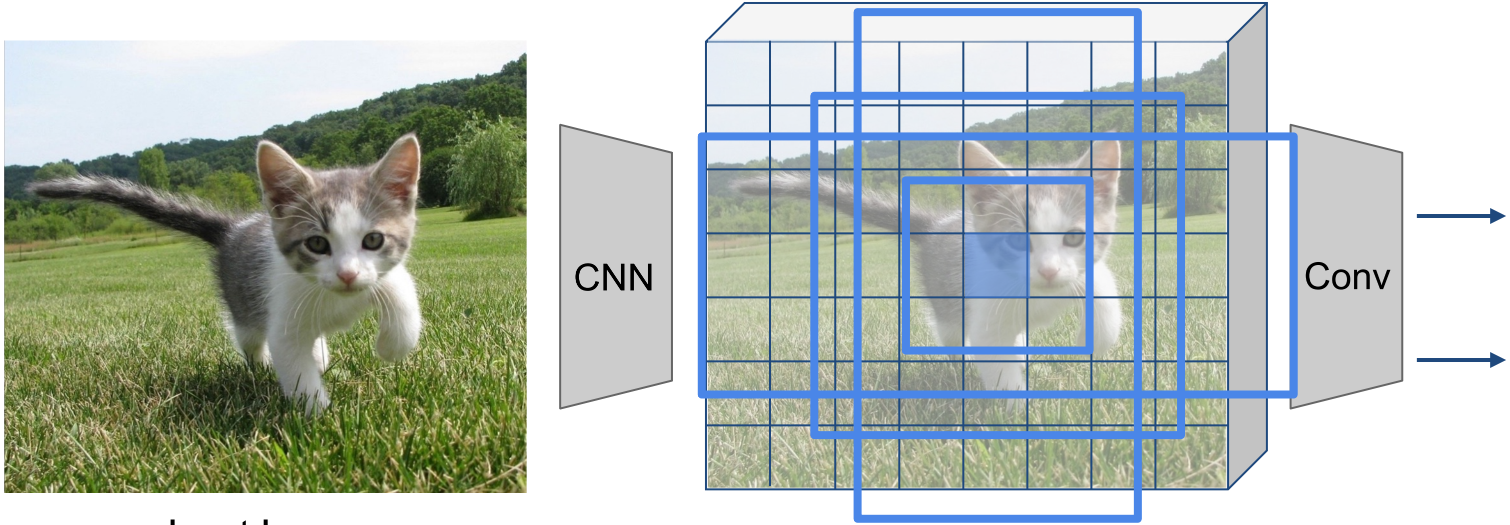
Anchor is an object?
1 x 20 x 15

Box corrections
4 x 20 x 15

For positive boxes, also predict a corrections from the anchor to the ground-truth box (regress 4 numbers per pixel)

# Region Proposal Network

In practice use K different anchor boxes of different size / scale at each point




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

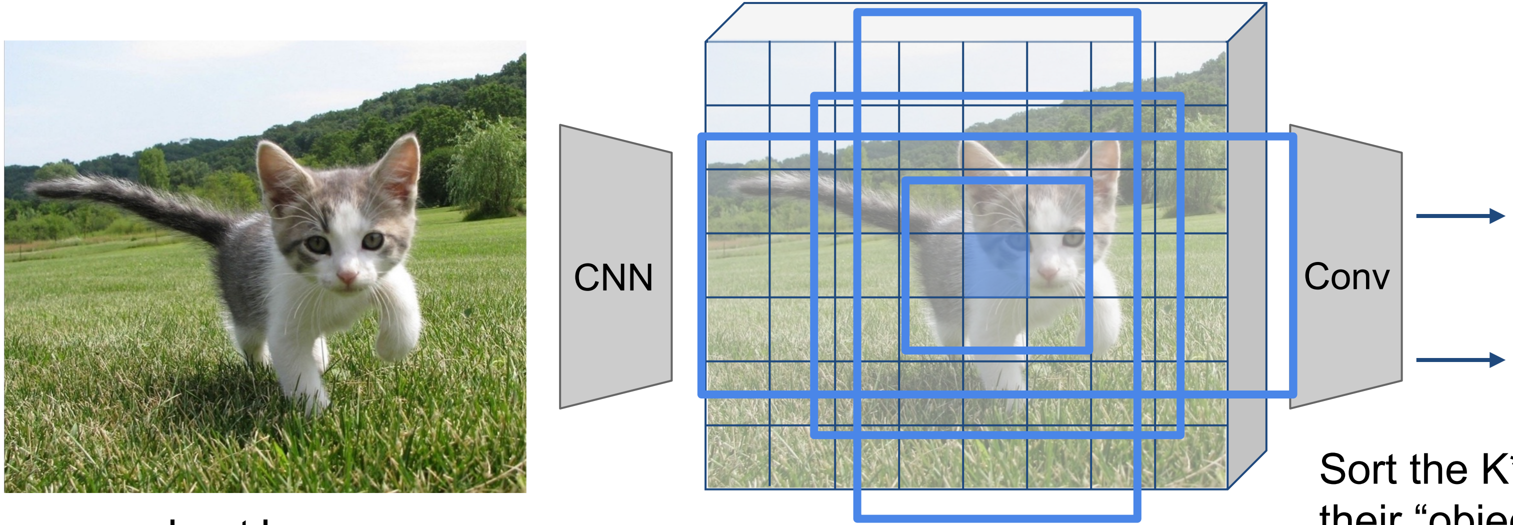
Image features
(e.g. 512)

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

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

# Region Proposal Network

In practice use K different anchor boxes of different size / scale at each point




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

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

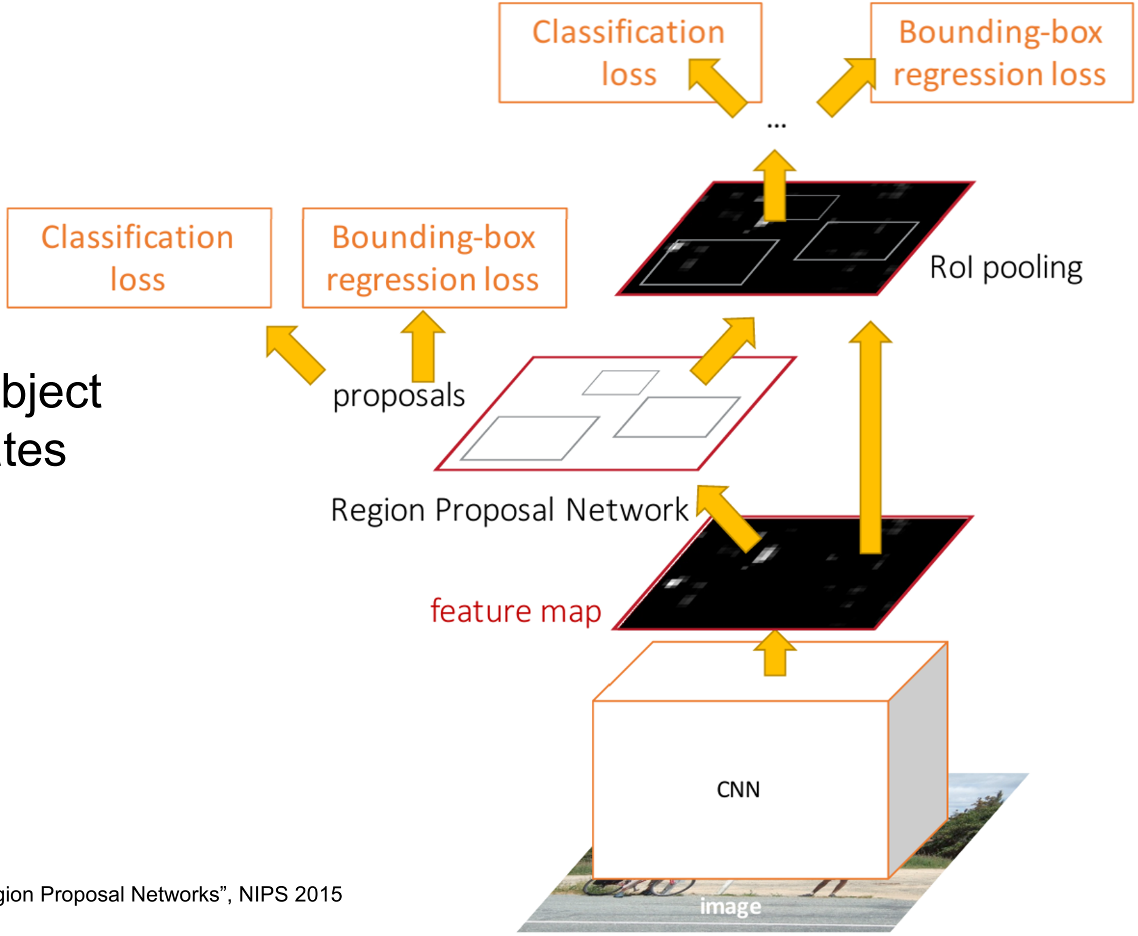
Input Image
(e.g. 3 x 640 x 480)

Image features
(e.g. 512)

Sort the K*20*15 boxes by their “objectness” score, take top ~300 as our proposals

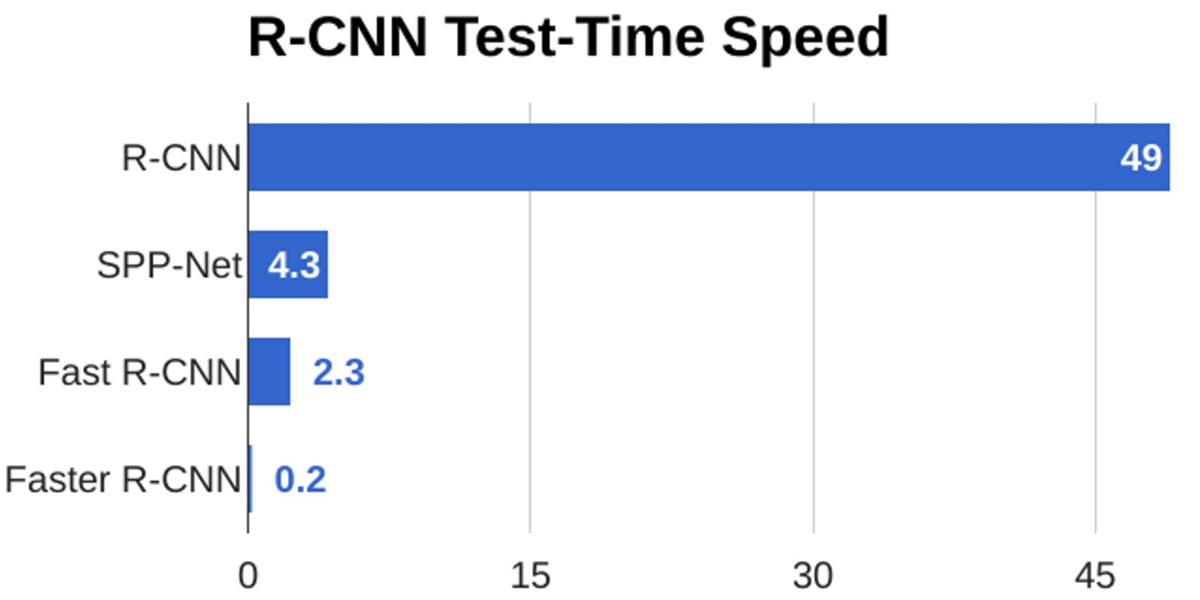
# Faster R-CNN:
Make CNN do proposals!

Jointly train with 4 losses:
1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates



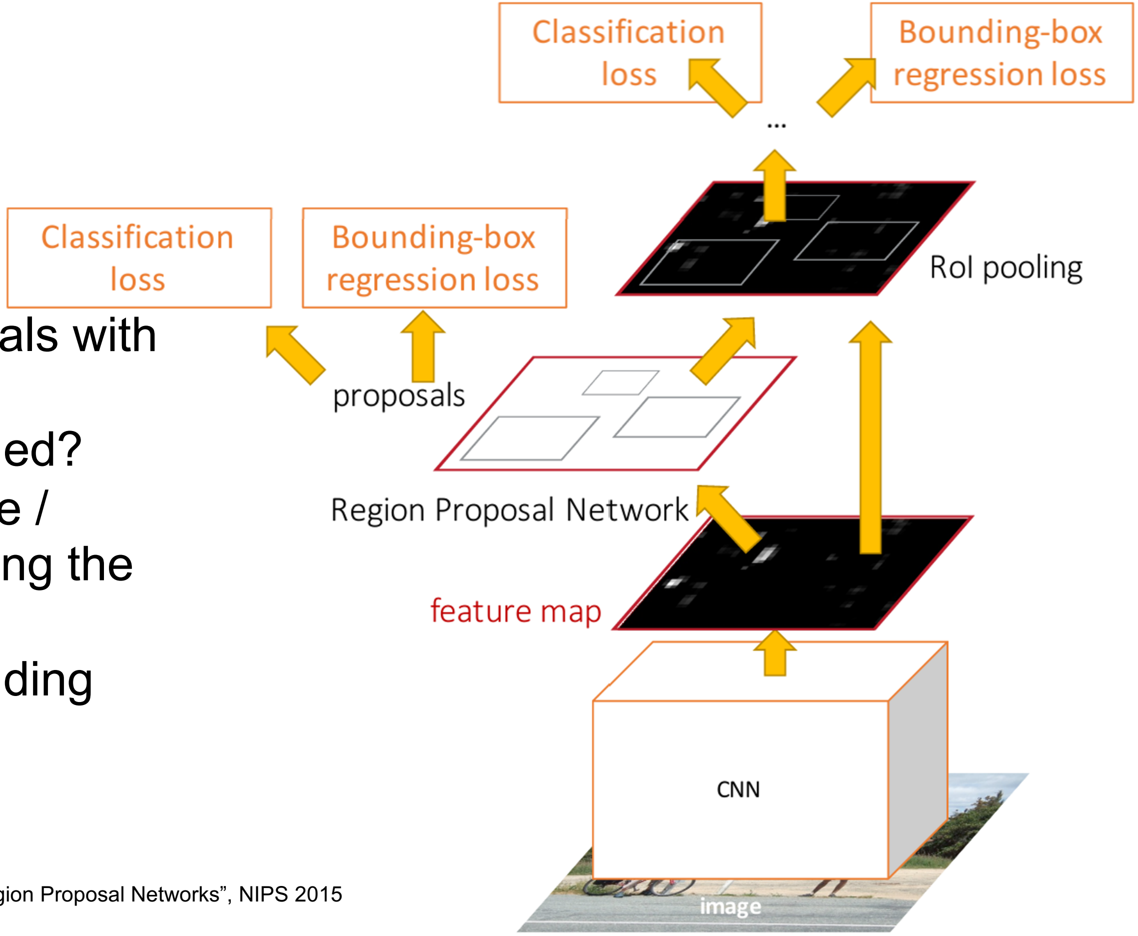

Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015

# Faster R-CNN:
Make CNN do proposals!



# Fast<u>er</u> R-CNN:
Make CNN do proposals!

Glossing over many details:
- Ignore overlapping proposals with **non-max suppression**
- How are anchors determined?
- How do we sample positive / negative samples for training the RPN?
- How to parameterize bounding box regression?




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

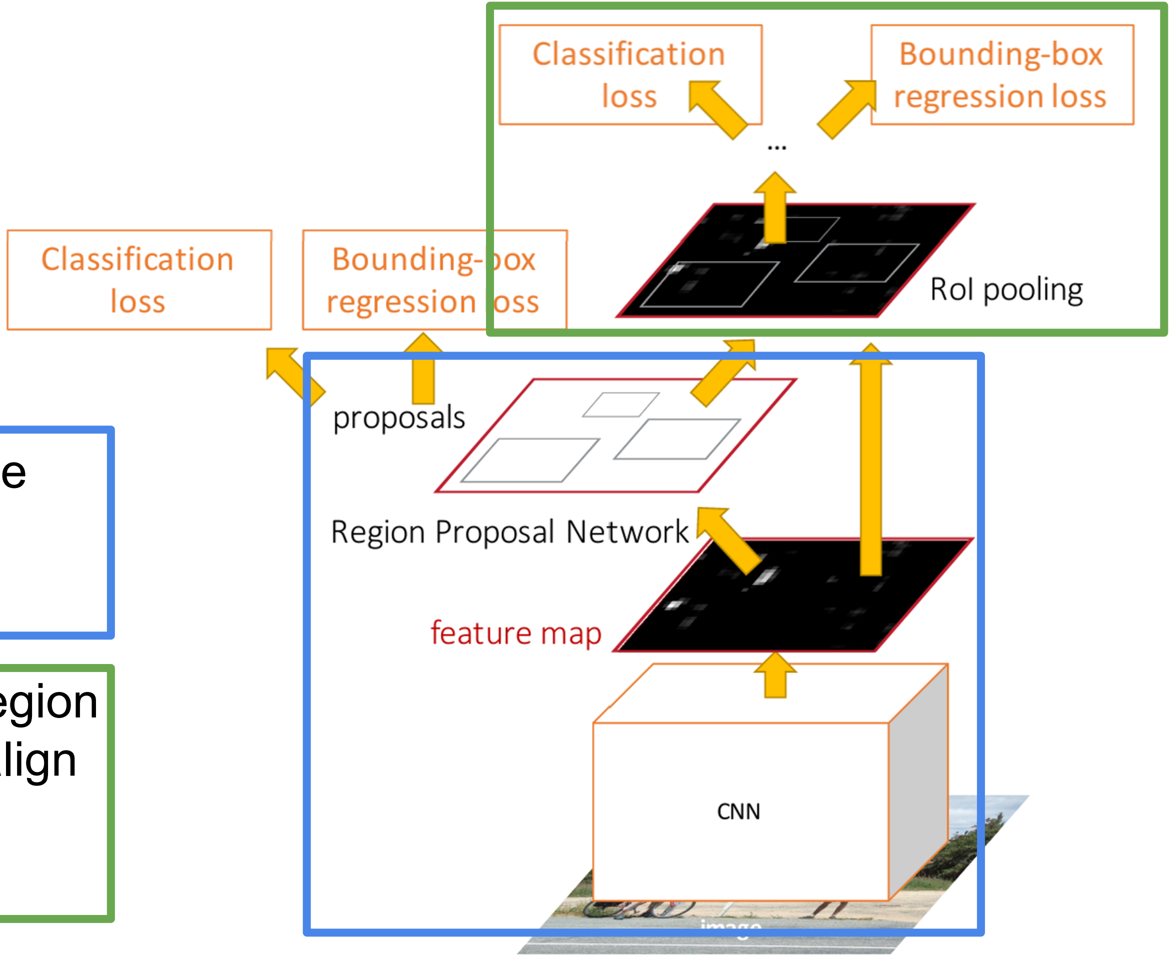
# Faster R-CNN:

Make CNN do 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:

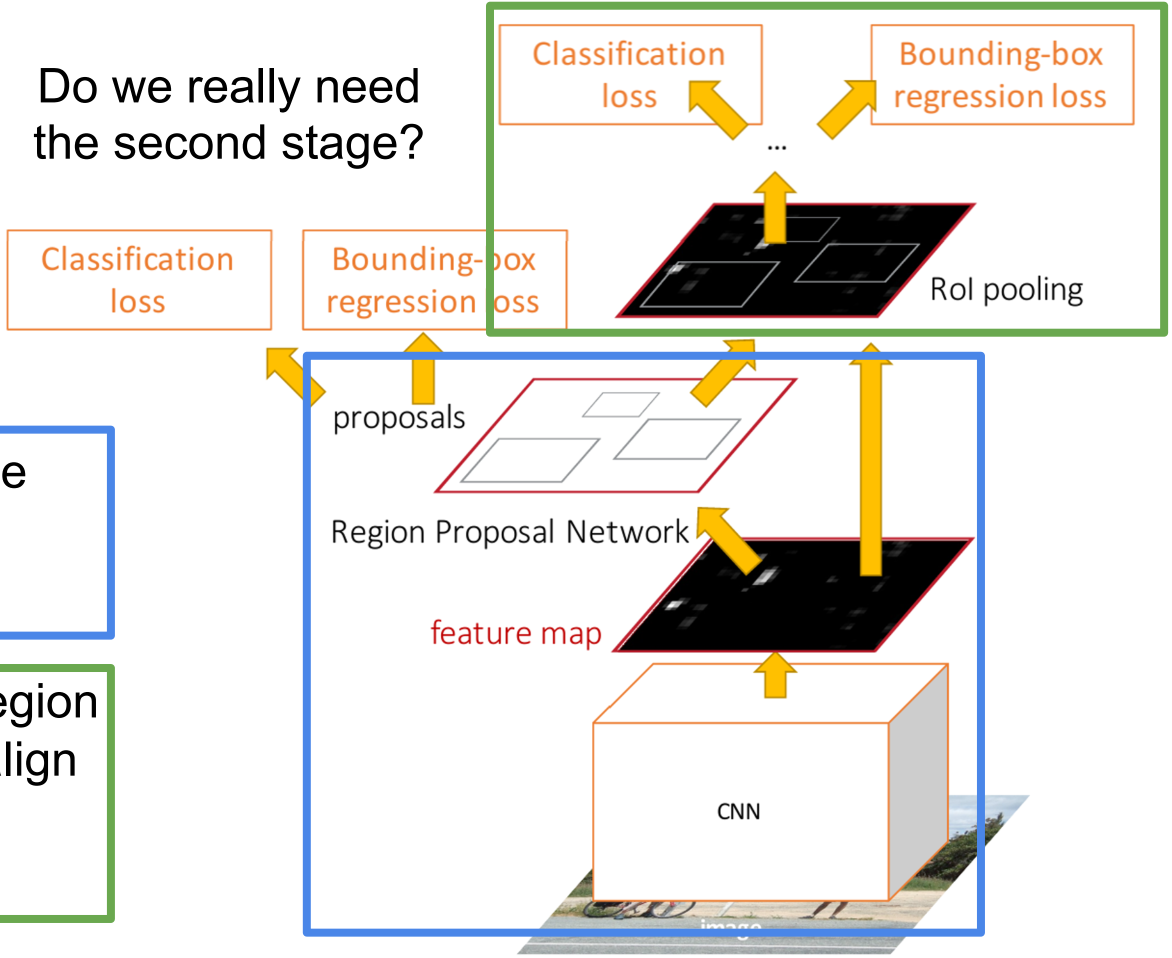
Make CNN do proposals!

Do we really need the second stage?

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

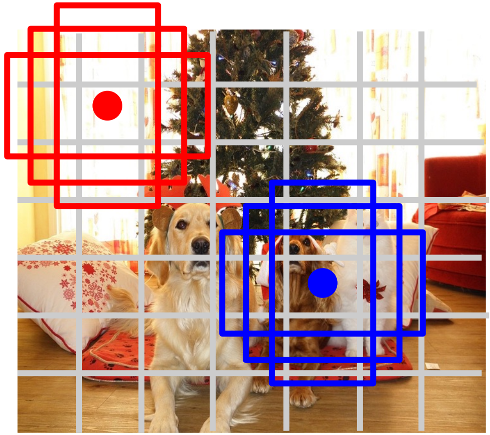
First stage: Run once per image
- Backbone network
- Region proposal network

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



# Single-Stage Object Detectors: YOLO / SSD / RetinaNet

Input image
3 x H x W



Divide image into grid
7 x 7

Image a set of **base boxes** centered at each grid cell
Here B = 3

Within each grid cell:
- Regress from each of the B base boxes to a final box with 5 numbers:
(dx, dy, dh, dw, confidence)
- Predict scores for each of C classes (including background as a class)
- Looks a lot like RPN, but category-specific!

Output:
7 x 7 x (5 * B + C)

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

# Object Detection: Lots of variables ...

**Backbone Network**
VGG16
ResNet-101
Inception V2
Inception V3
Inception ResNet
MobileNet

**“Meta-Architecture”**
Two-stage: Faster R-CNN
Single-stage: YOLO / SSD
Hybrid: R-FCN

**Image Size**
**# Region Proposals**
**…**

**Takeaways**
Faster R-CNN is slower but more accurate

SSD is much faster but not as accurate

Bigger / Deeper backbones work better

Huang et al, “Speed/accuracy trade-offs for modern convolutional object detectors”, CVPR 2017
Zou et al, “Object Detection in 20 Years: A Survey”, arXiv 2019

R-FCN: Dai et al, “R-FCN: Object Detection via Region-based Fully Convolutional Networks”, NIPS 2016
Inception-V2: Ioffe and Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, ICML 2015
Inception V3: Szegedy et al, “Rethinking the Inception Architecture for Computer Vision”, arXiv 2016
Inception ResNet: Szegedy et al, “Inception-V4, Inception-ResNet and the Impact of Residual Connections on Learning”, arXiv 2016
MobileNet: Howard et al, “Efficient Convolutional Neural Networks for Mobile Vision Applications”, arXiv 2017

# Instance Segmentation

**Classification**



**CAT**

No spatial extent

**Semantic Segmentation**



**GRASS, CAT, TREE, SKY**

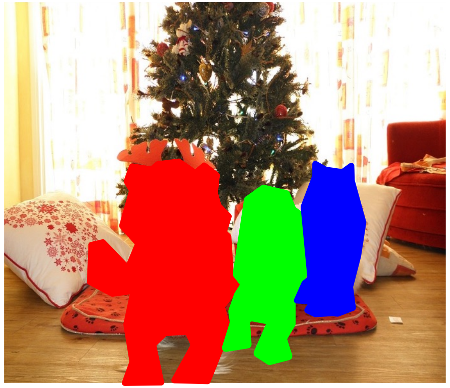
No objects, just pixels

**Object Detection**



**DOG, DOG, CAT**

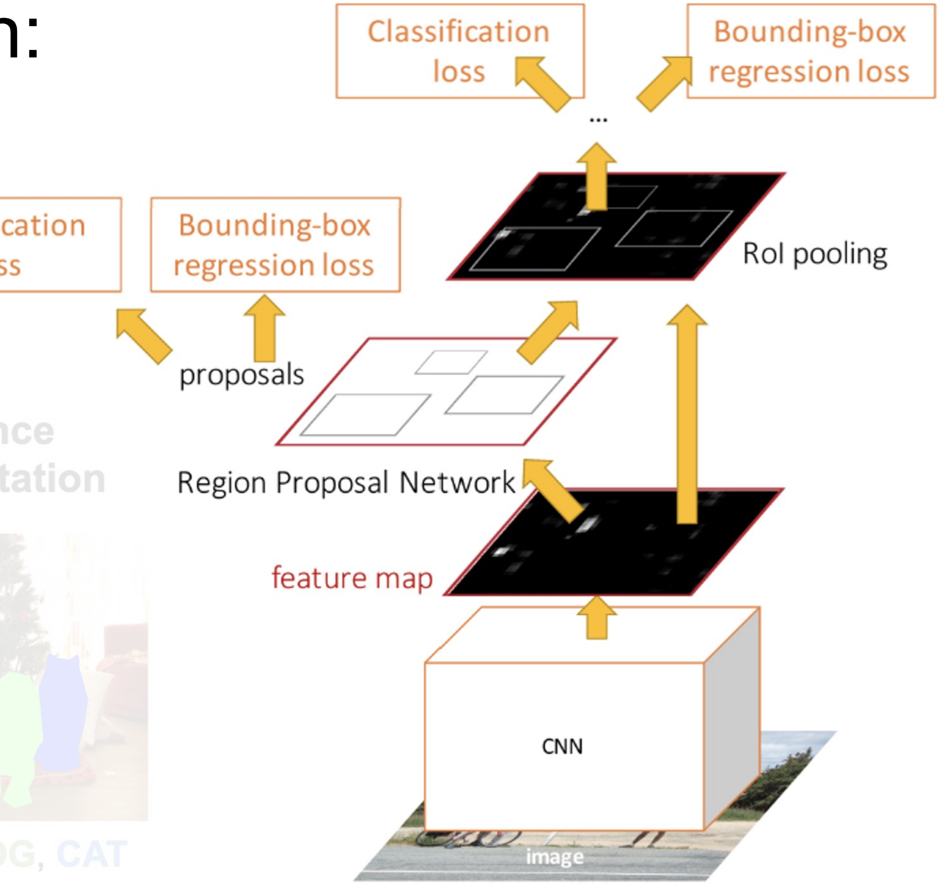
**Instance Segmentation**



**DOG, DOG, CAT**

Multiple Object

# Object Detection: Faster R-CNN




**Object Detection**

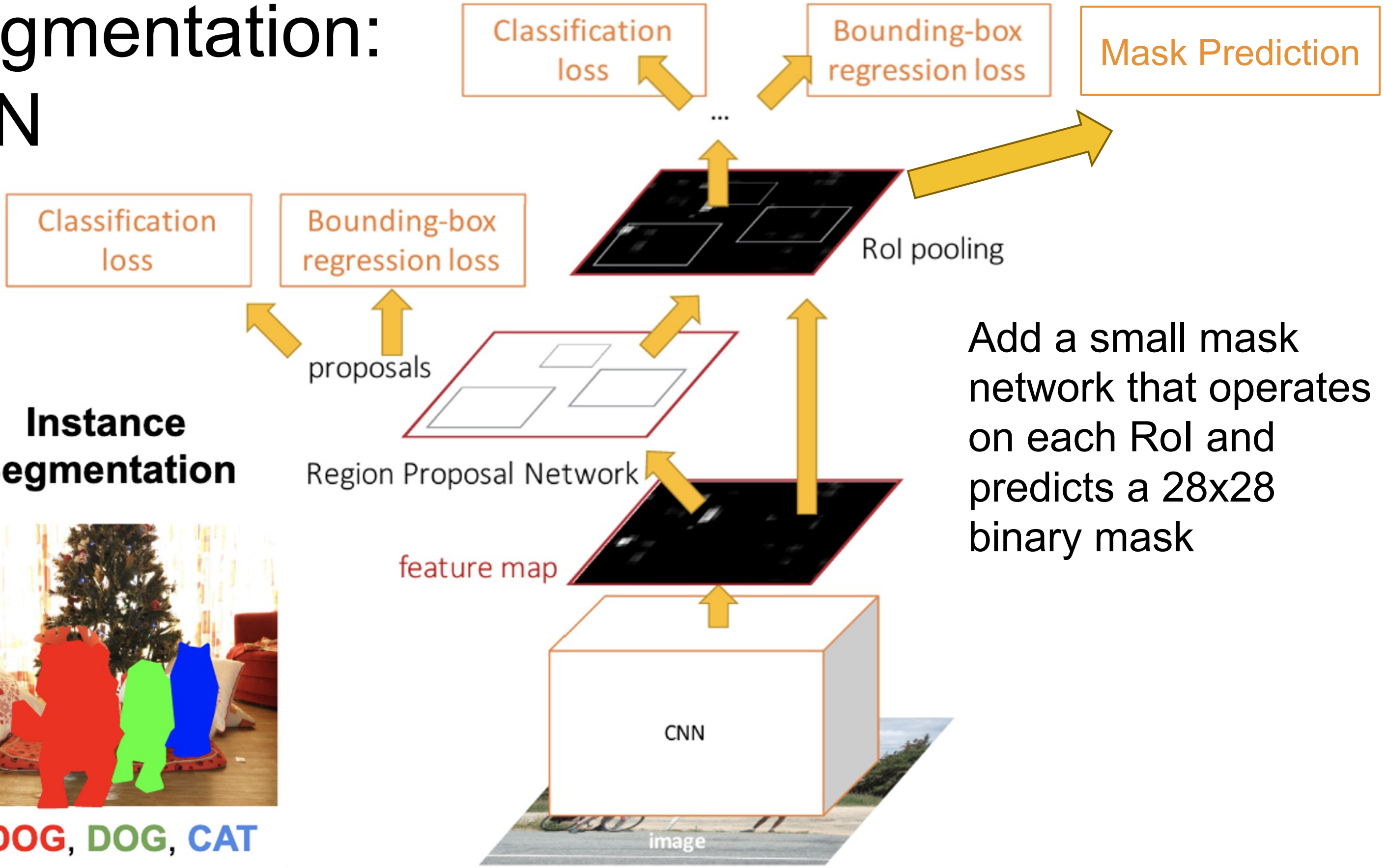
DOG, DOG, CAT

Instance Segmentation



DOG, DOG, CAT

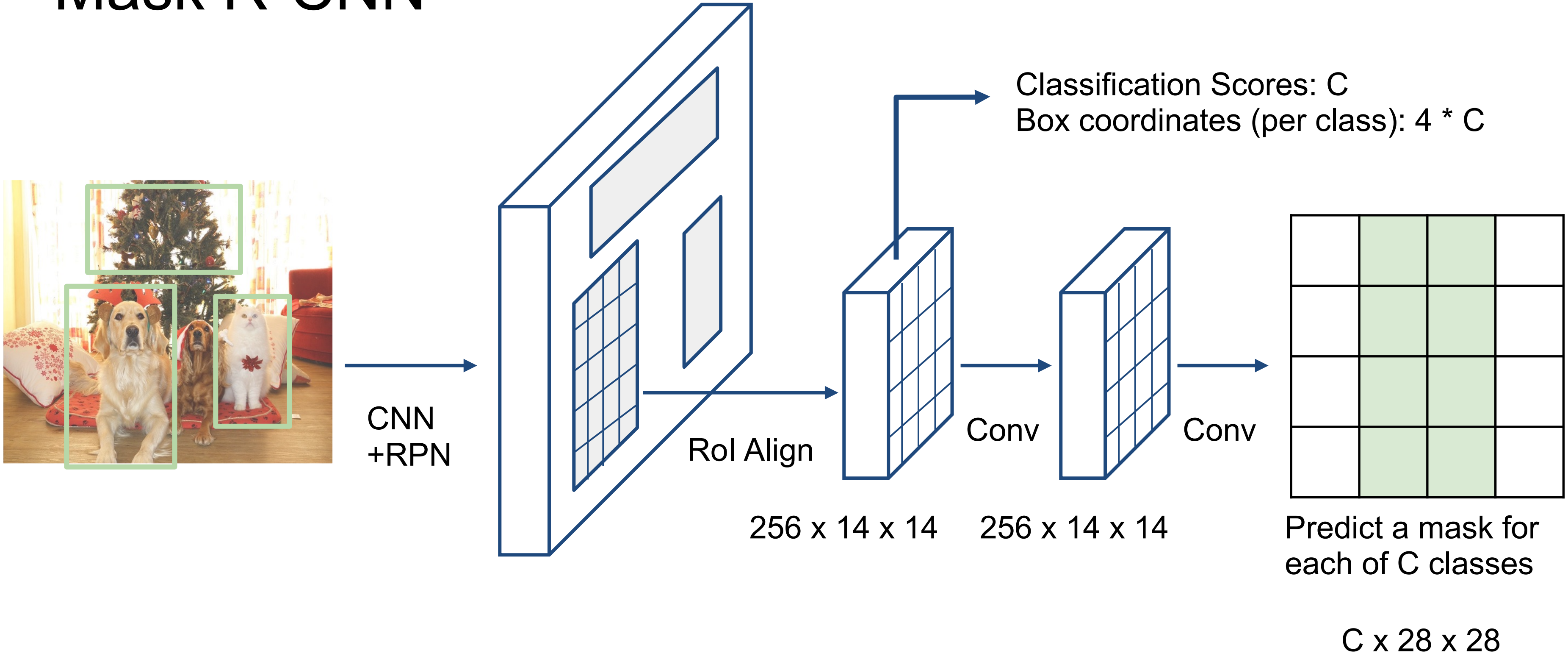
# Instance Segmentation: Mask R-CNN


Add a small mask network that operates on each RoI and predicts a 28x28 binary mask

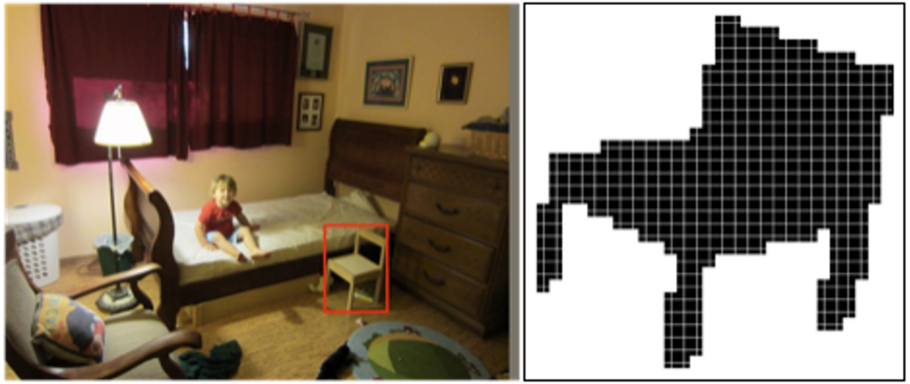
He et al, "Mask R-CNN", ICCV 2017

# Mask R-CNN

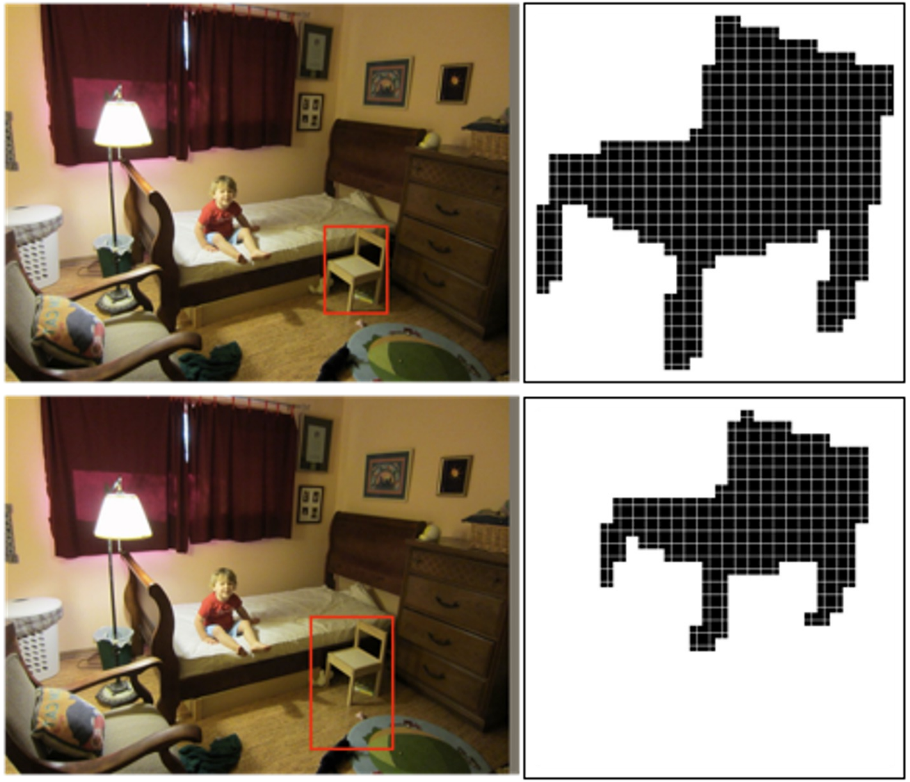



He et al, “Mask R-CNN”, arXiv 2017

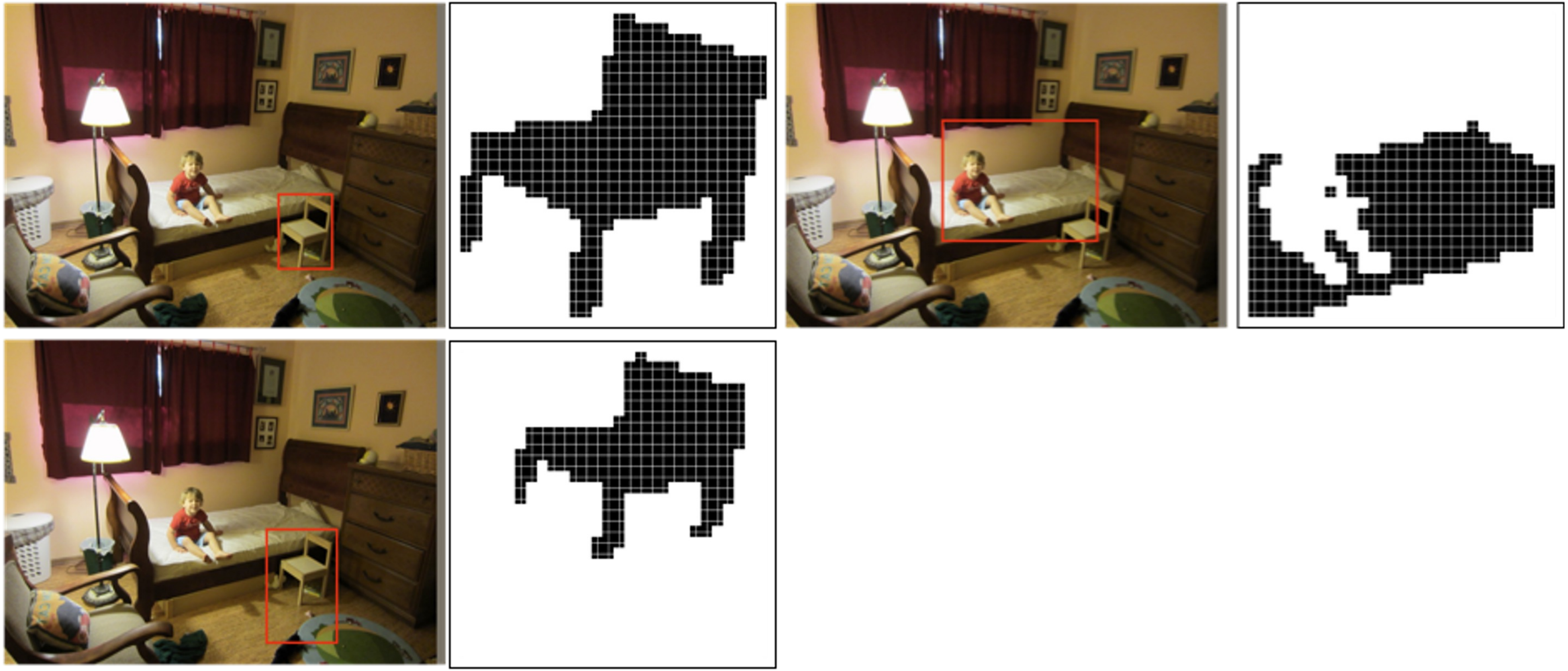
# Mask R-CNN: Example Mask Training Targets



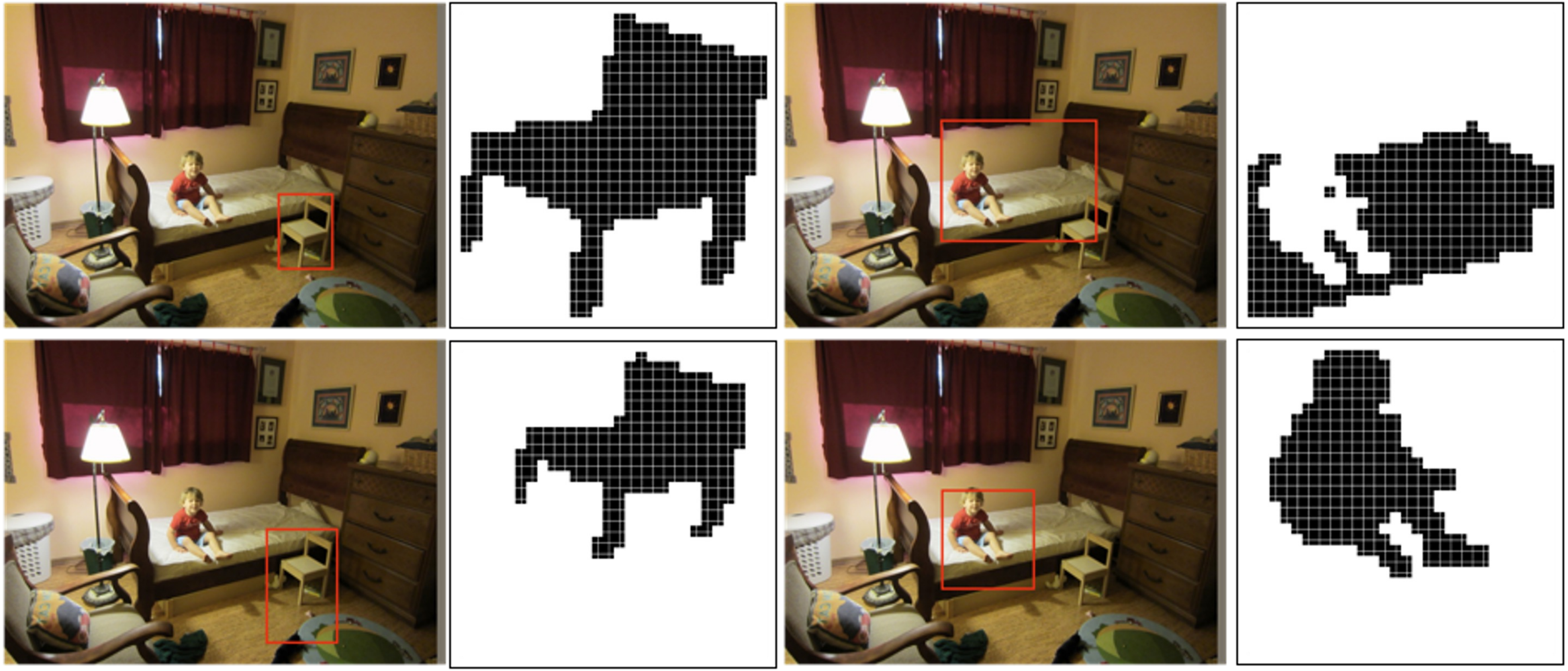
# Mask R-CNN: Example Mask Training Targets



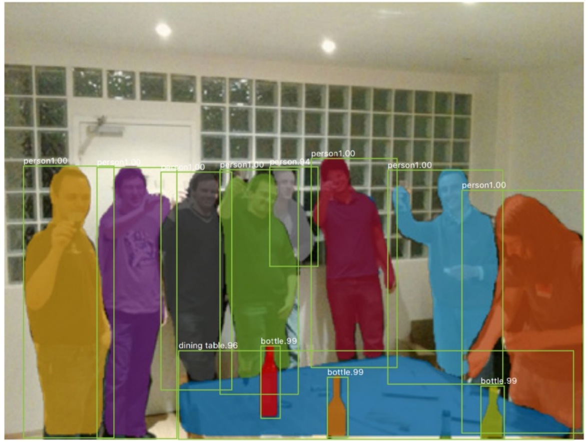
# Mask R-CNN: Example Mask Training Targets



# Mask R-CNN: Example Mask Training Targets



# Mask R-CNN: Very Good Results!



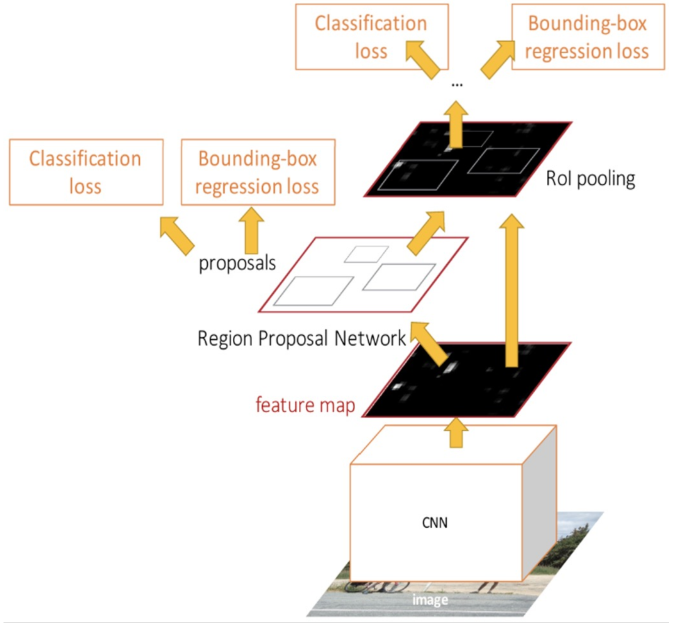
He et al, “Mask R-CNN”, ICCV 2017

# Mask R-CNN
# Also does pose


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

# RCNN Series

- **R-CNN**: Per-region detection, hand-crafted region proposal
- **Fast R-CNN**: Shared feature extraction, RoI Pooling, Anchors
- **Faster R-CNN**: Region Proposal Networks, RoI Align
- **Mask R-CNN**: Instance Segmentation




Detectors are becoming more complex!
Many hyperparameters to tune for each components …
Can we simplify it?

# End-to-End Object Detection with Transformers

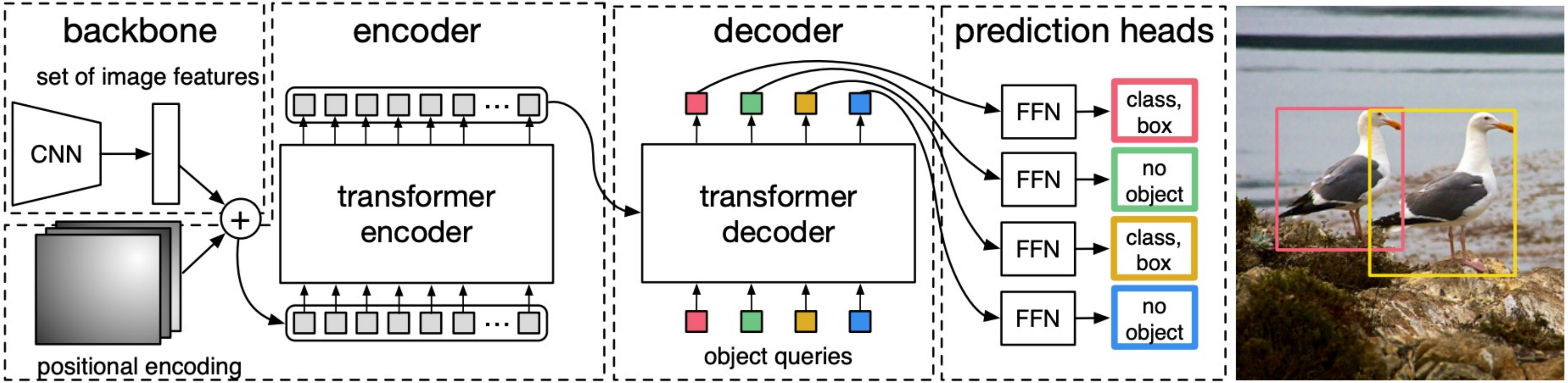
Nicolas Carion*, Francisco Massa*, Gabriel Synnaeve, Nicolas Usunier,
Alexander Kirillov, and Sergey Zagoruyko

Facebook AI

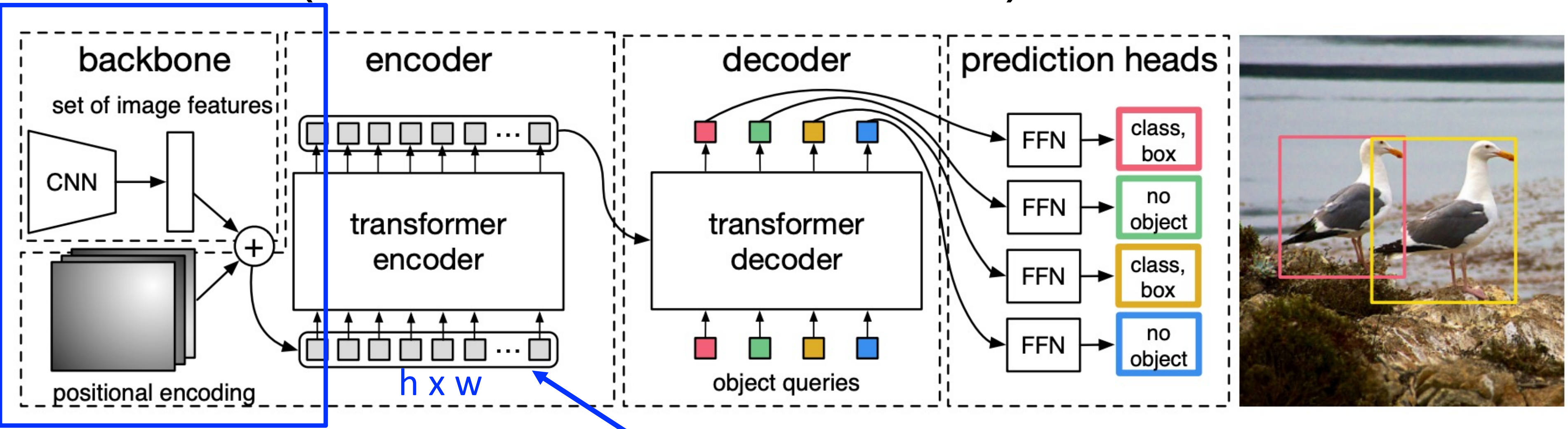
**Key ideas:**

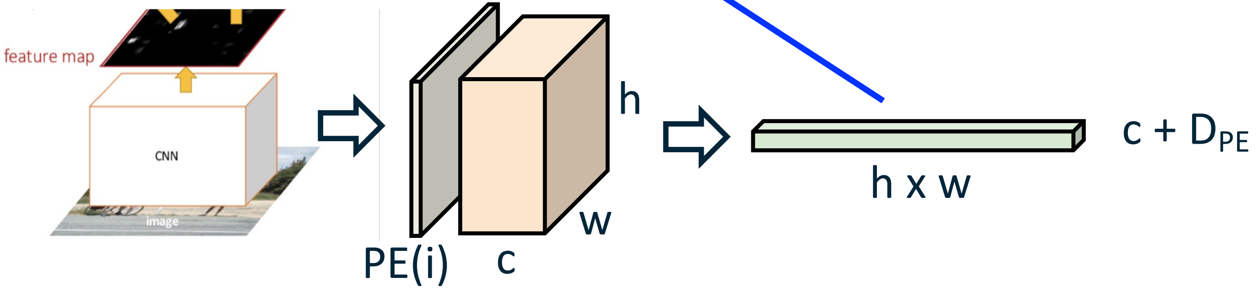
- Detection as a **set-to-set prediction** problem
- Use Transformer to model the detection problem

# DETR (DEtection TRansformer)

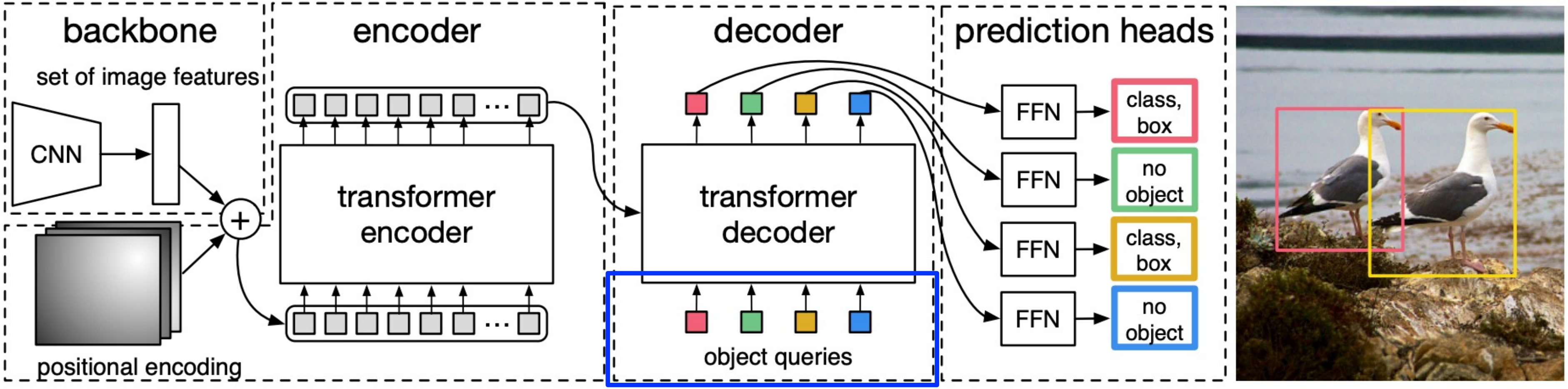


# DETR (DEtection TRansformer)



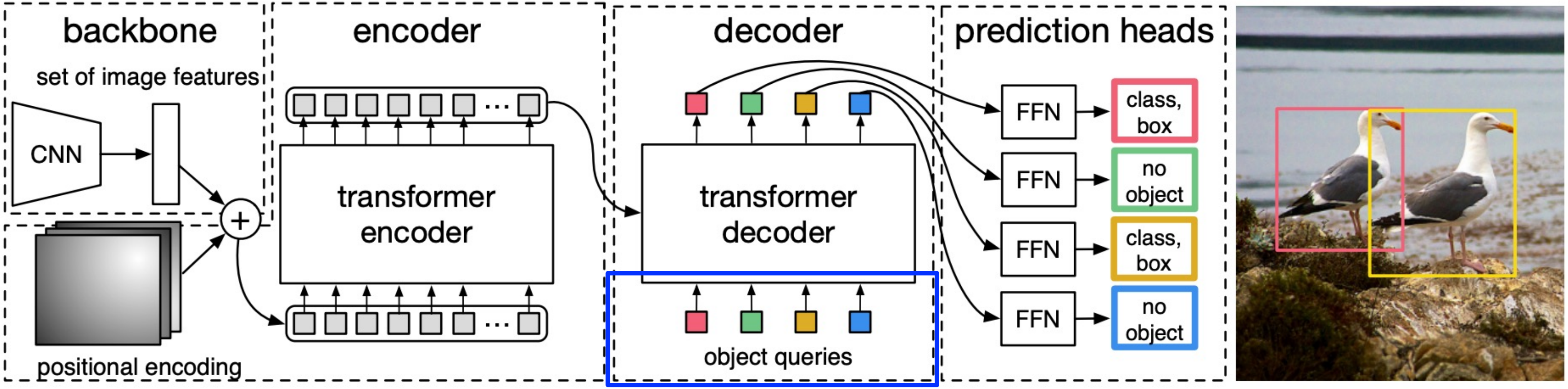



# DETR (DEtection TRansformer)




A fixed set of learnable embeddings,
e.g., 300 size-N vectors
Q: Why?

# DETR (DEtection TRansformer)

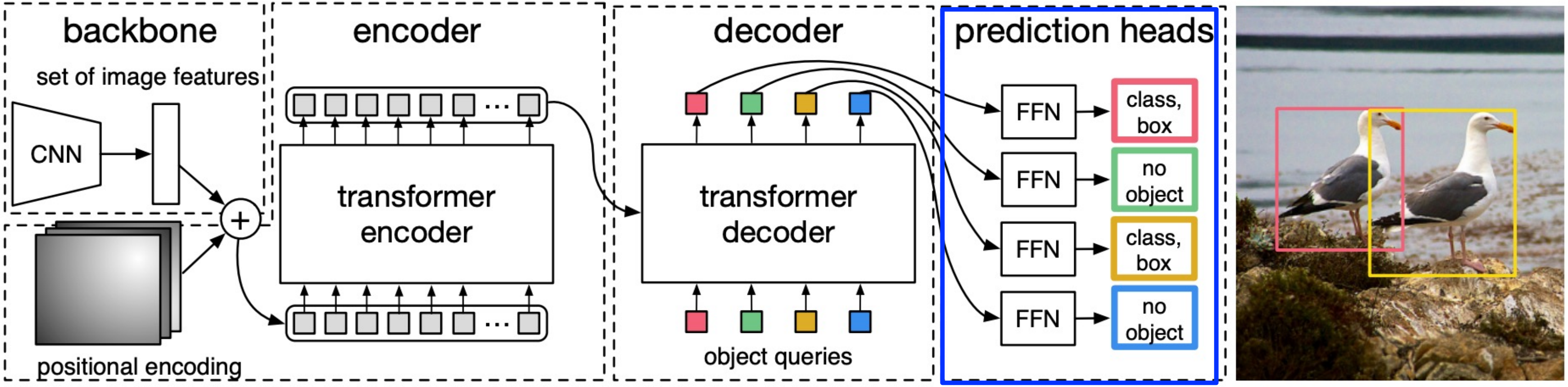



A fixed set of learnable embeddings, e.g., 300 size-N vectors

Q: Why?

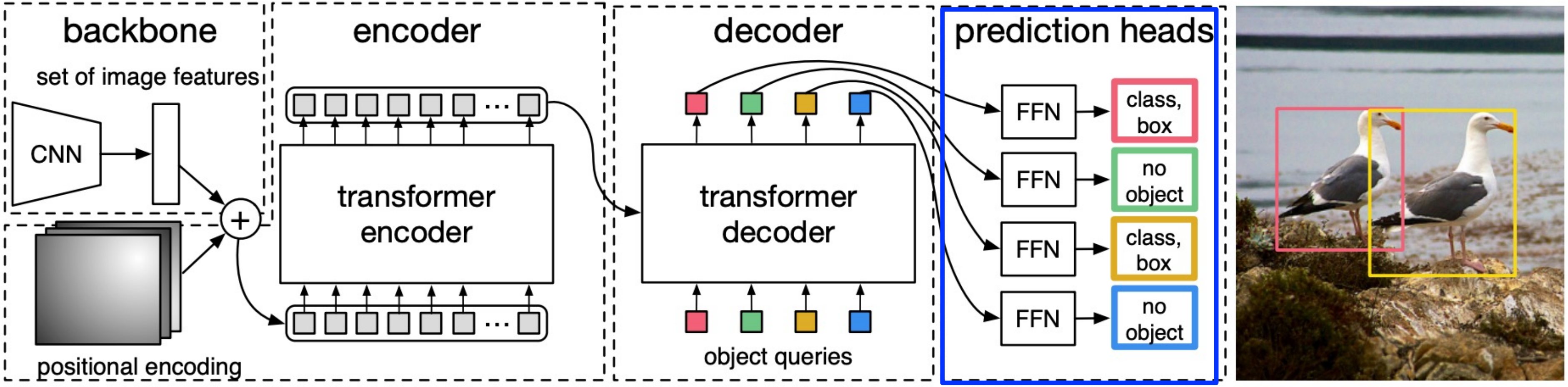
A: Break the symmetry of predictions, so that each prediction is different. Analogous to anchors in *R-CNN, but no spatial location

# DETR (DEtection TRansformer)



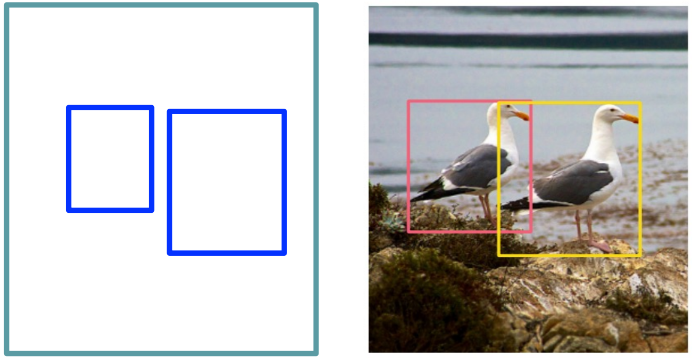

**Problem**: We don’t know which query corresponds to which ground truth during training! We can’t predetermine a fixed order like in sequence modeling.

# DETR (DEtection TRansformer)

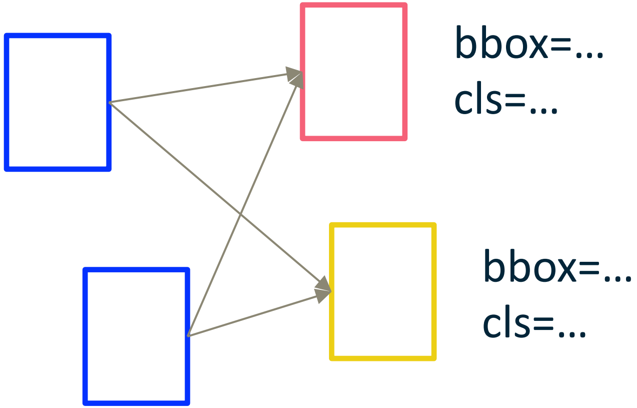



**Problem**: We don't know which query corresponds to which ground truth during training! We can't predetermine a fixed order like in sequence modeling.
**Solution:** Set matching loss --- train your model to generate a set of predictions that matches ground truth regardless of its order.

# Hangarian Loss (Set Matching Loss)



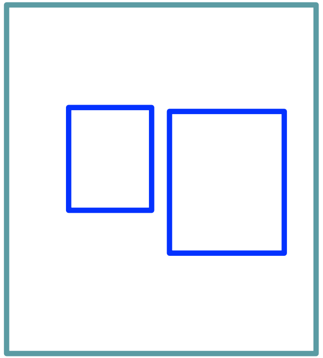
Prediction    Ground Truth



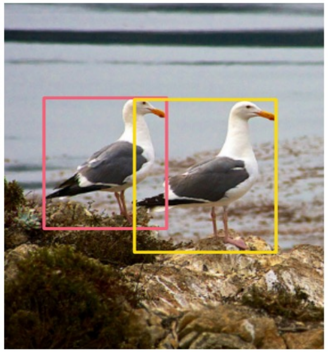

Goal: minimize bipartite distance

**Problem**: each prediction should only be trained to match one ground truth. We don’t know the matching beforehand!

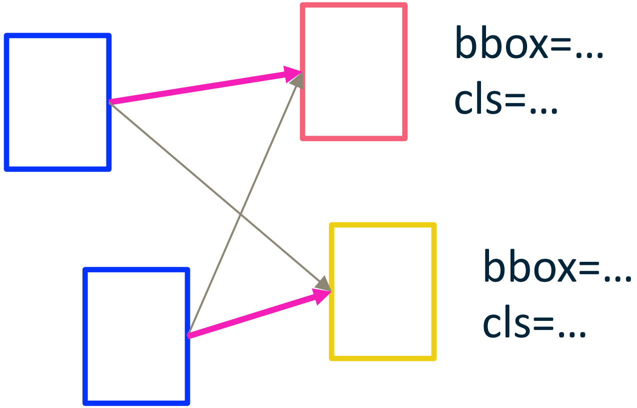
# Hangarian Loss (Set Matching Loss)



Prediction



Ground Truth




Goal: minimize bipartite distance

1. **Hungarian matching:** find the minimum-loss bipartite matching between prediction and ground truth given the current prediction.
2. **Minimize matched loss:** Given the matched prediction and ground truth, minimize the detection loss (bounding box distance and classification CE loss)

# Comparison with FasterRCNN

| Model | GFLOPS/FPS | #params | AP | $AP_{50}$ | $AP_{75}$ | $AP_S$ | $AP_M$ | $AP_L$ |
|---|---|---|---|---|---|---|---|---|
| Faster RCNN-DC5 | 320/16 | 166M | 39.0 | 60.5 | 42.3 | 21.4 | 43.5 | 52.5 |
| Faster RCNN-FPN | 180/26 | 42M | 40.2 | 61.0 | 43.8 | 24.2 | 43.5 | 52.0 |
| Faster RCNN-R101-FPN | 246/20 | 60M | 42.0 | 62.5 | 45.9 | 25.2 | 45.6 | 54.6 |
| Faster RCNN-DC5+ | 320/16 | 166M | 41.1 | 61.4 | 44.3 | 22.9 | 45.9 | 55.0 |
| Faster RCNN-FPN+ | 180/26 | 42M | 42.0 | 62.1 | 45.5 | 26.6 | 45.4 | 53.4 |
| Faster RCNN-R101-FPN+ | 246/20 | 60M | 44.0 | 63.9 | **47.8** | **27.2** | 48.1 | 56.0 |
| DETR | 86/28 | 41M | 42.0 | 62.4 | 44.2 | 20.5 | 45.8 | 61.1 |
| DETR-DC5 | 187/12 | 41M | 43.3 | 63.1 | 45.9 | 22.5 | 47.3 | 61.1 |
| DETR-R101 | 152/20 | 60M | 43.5 | 63.8 | 46.4 | 21.9 | 48.0 | 61.8 |
| DETR-DC5-R101 | 253/10 | 60M | **44.9** | **64.7** | 47.7 | 23.7 | **49.5** | **62.3** |

Similar size, simpler, and (mostly) better!

Can we make this even more general …

# Segment Anything

Alexander Kirillov[1,2,4] Eric Mintun[2] Nikhila Ravi[1,2] Hanzi Mao[2] Chloe Rolland[3] Laura Gustafson[3]
Tete Xiao[3] Spencer Whitehead Alexander C. Berg Wan-Yen Lo Piotr Dollár[4] Ross Girshick[4]

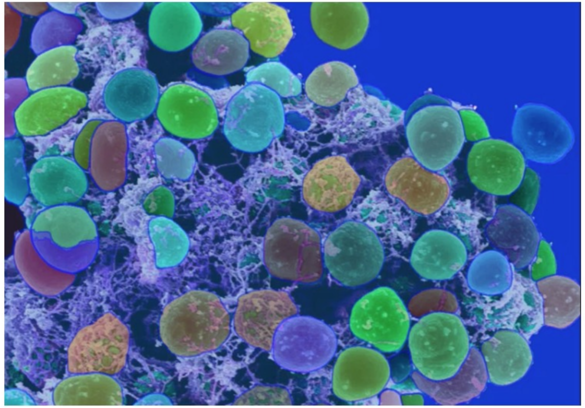
[1]project lead [2]joint first author [3]equal contribution [4]directional lead

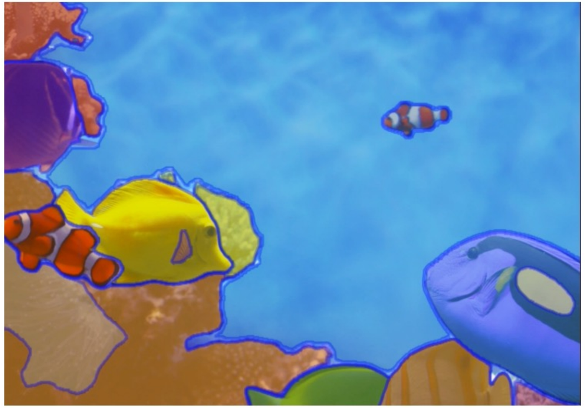
Meta AI Research, FAIR

**Key ideas:**

- Query-based prediction instead of fixed set-to-set prediction
- Large-scale training data with auto-labeling

# Foundation Image Segmentation Models





SegmentAnything (Meta AI, 2023)

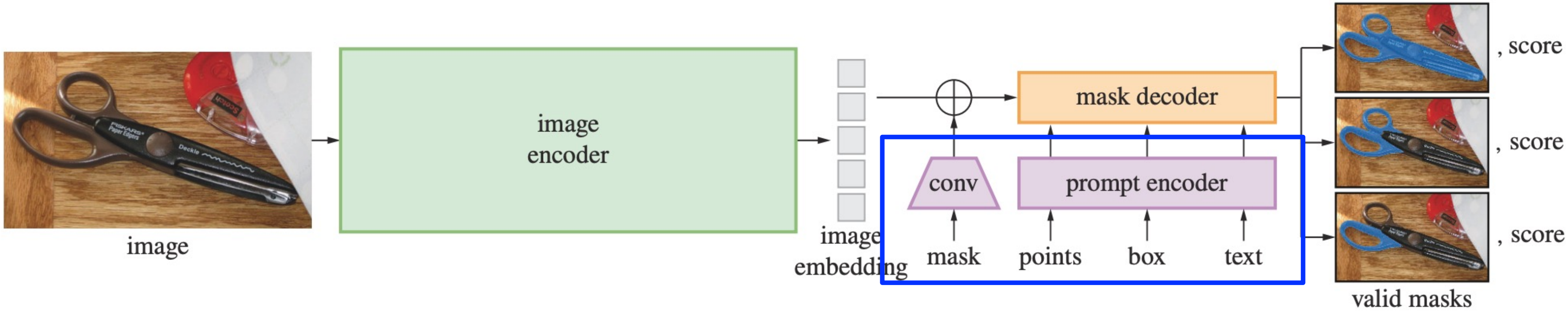
Try it yourself! https://segment-anything.com/demo#

# Foundation Image Segmentation Models

SegmentAnything (Meta AI, 2023)

Try it yourself! https://segment-anything.com/demo#

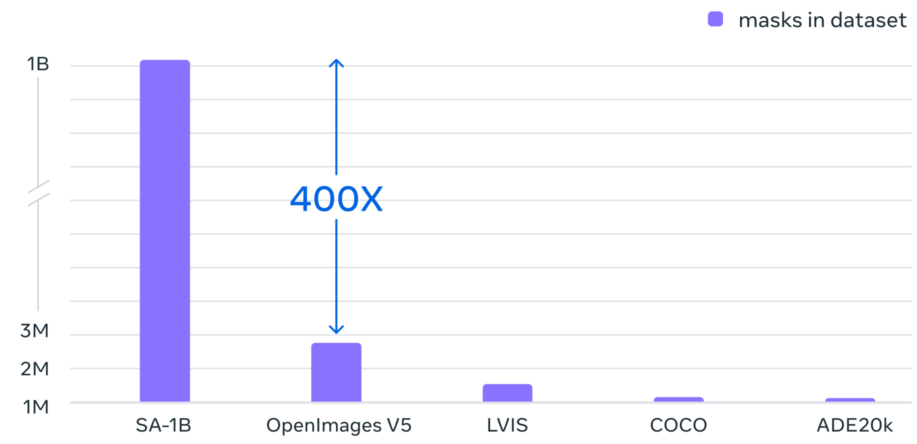
# Foundation Image Segmentation Models

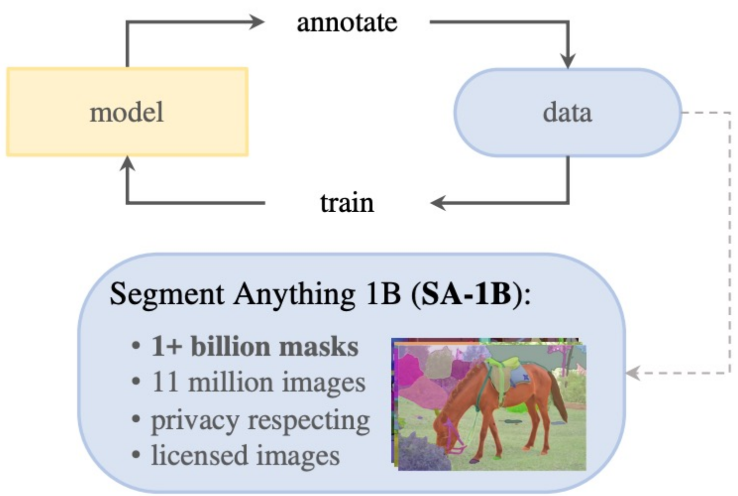



No more learned embeddings. Query anything you want!

SegmentAnything (Meta AI, 2023)

# Foundation Image Segmentation Models

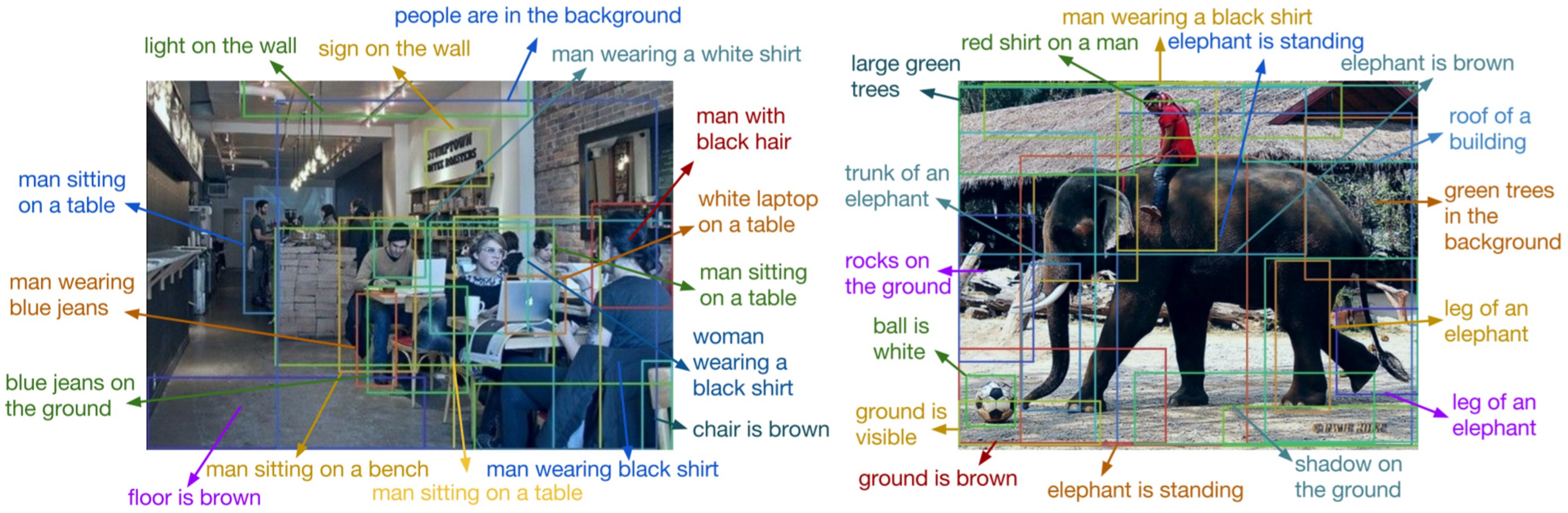


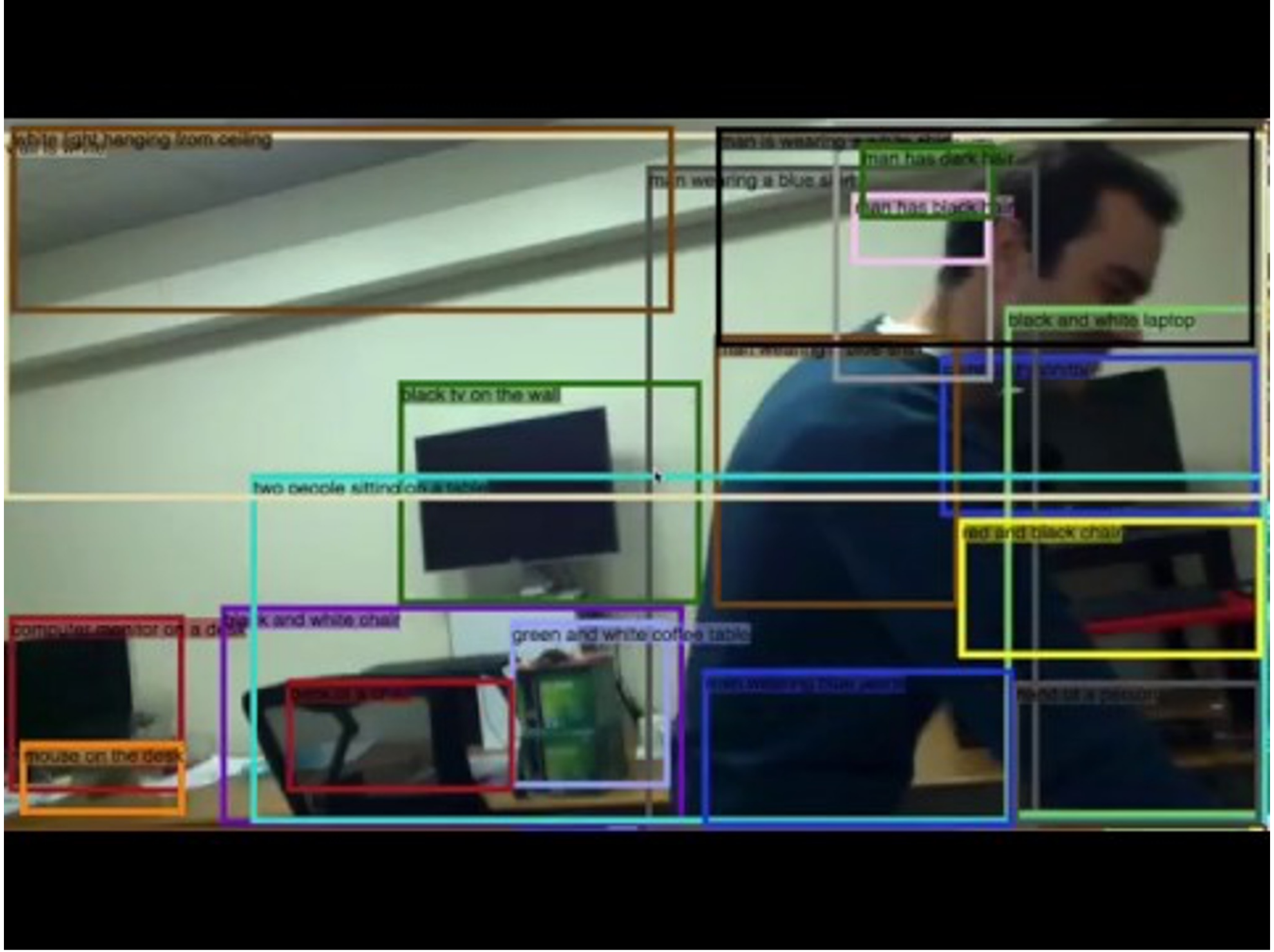

SegmentAnything (Meta AI, 2023)

# Beyond 2D Object Detection...

# Object Detection + Captioning = Dense Captioning

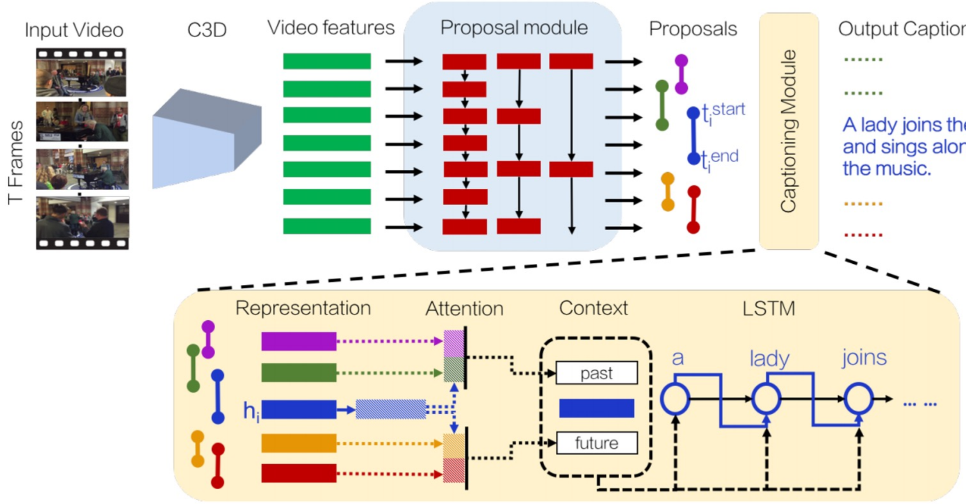


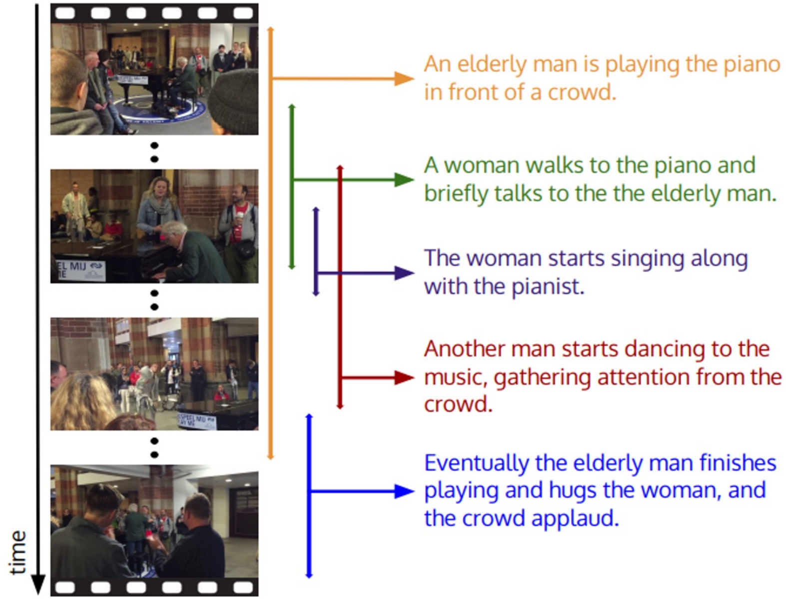

Johnson, Karpathy, and Fei-Fei, “DenseCap: Fully Convolutional Localization Networks for Dense Captioning”, CVPR 2016



man has dark hair
black and white laptop
black tv on the wall
red and black chair
black and white chair
green and white coffee table
mouse on the desk

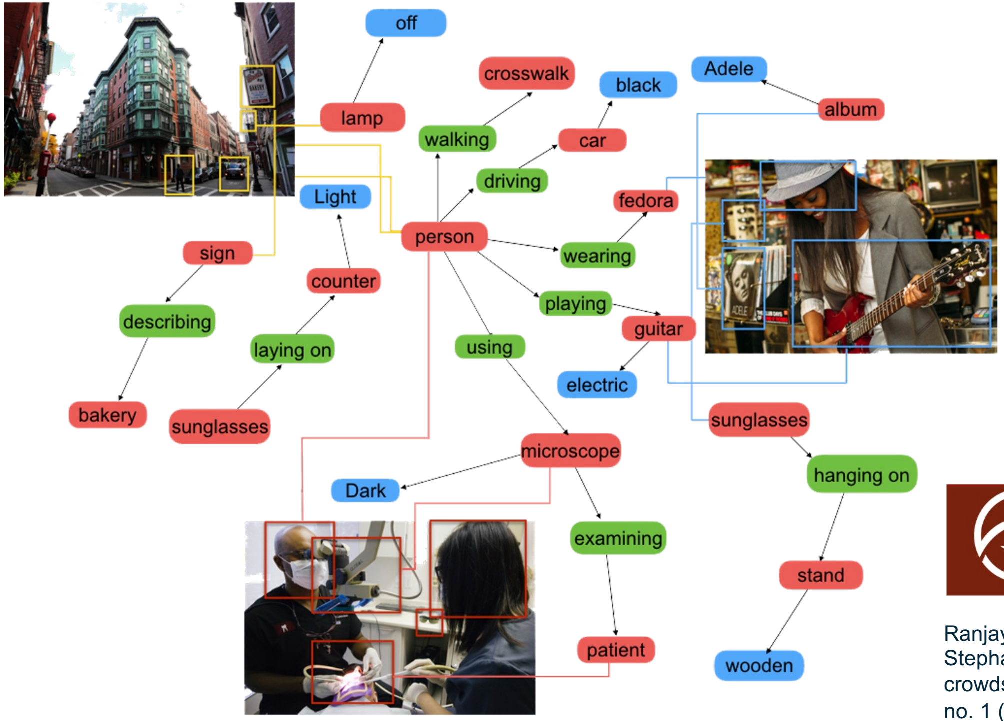
# Dense Video Captioning




Ranjay Krishna et al., “Dense-Captioning Events in Videos”, ICCV 2017

# Objects + Relationships = Scene Graphs

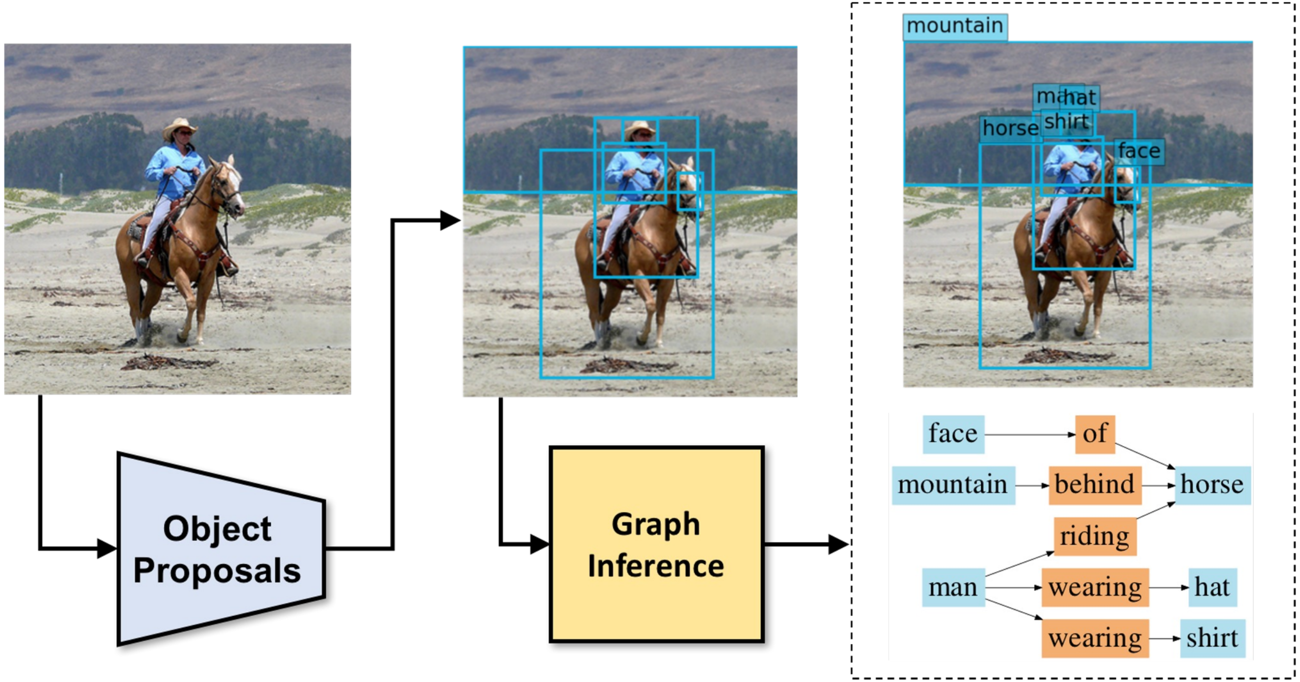



108,077 Images
5.4 Million Region Descriptions
1.7 Million Visual Question Answers
3.8 Million Object Instances
2.8 Million Attributes
2.3 Million Relationships
Everything Mapped to Wordnet Synsets

VISUALGENOME

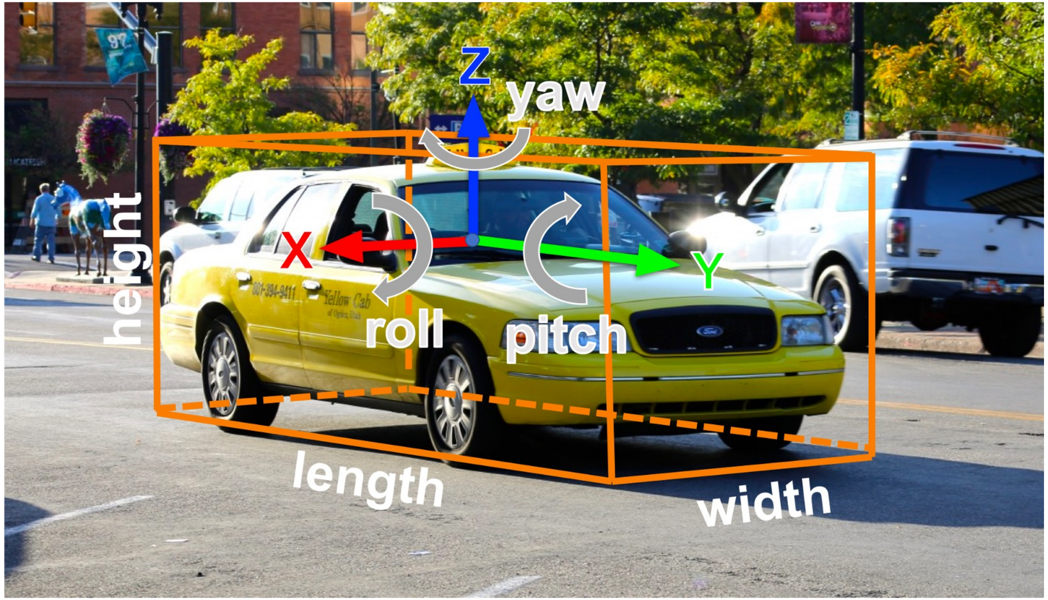
Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen et al. "Visual genome: Connecting language and vision using crowdsourced dense image annotations." International Journal of Computer Vision 123, no. 1 (2017): 32-73.

# Scene Graph Prediction




Xu, Zhu, Choy, and Fei-Fei, “Scene Graph Generation by Iterative Message Passing”, CVPR 2017

# 3D Object Detection



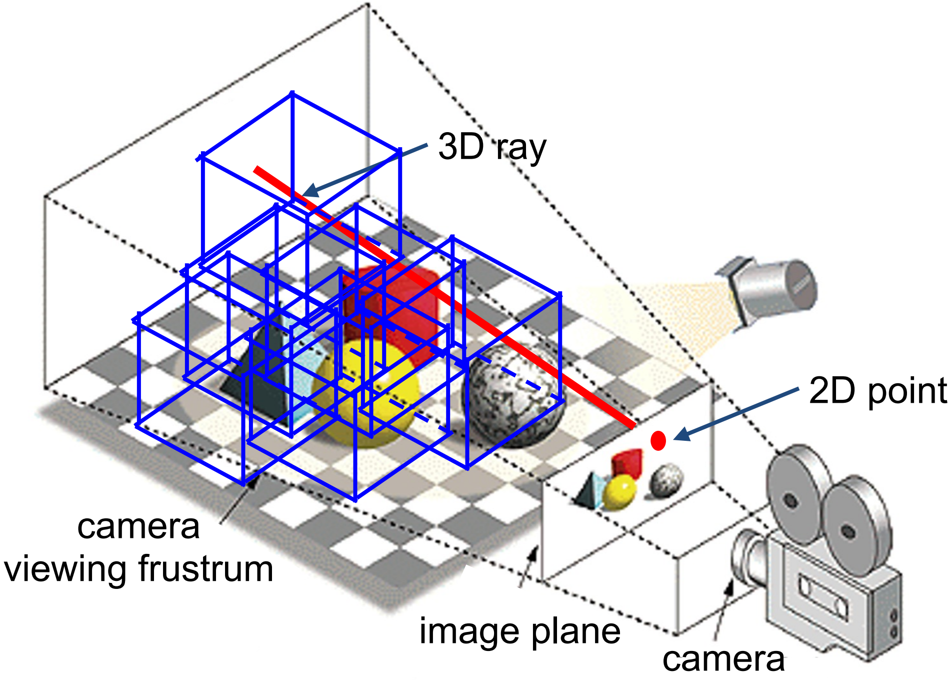

2D Object Detection:
2D bounding box
(x, y, w, h)

3D Object Detection:
3D oriented bounding box
(x, y, z, w, h, l, r, p, y)

Simplified bbox: no roll & pitch

Much harder problem than 2D object detection!

# 3D Object Detection: Simple Camera Model



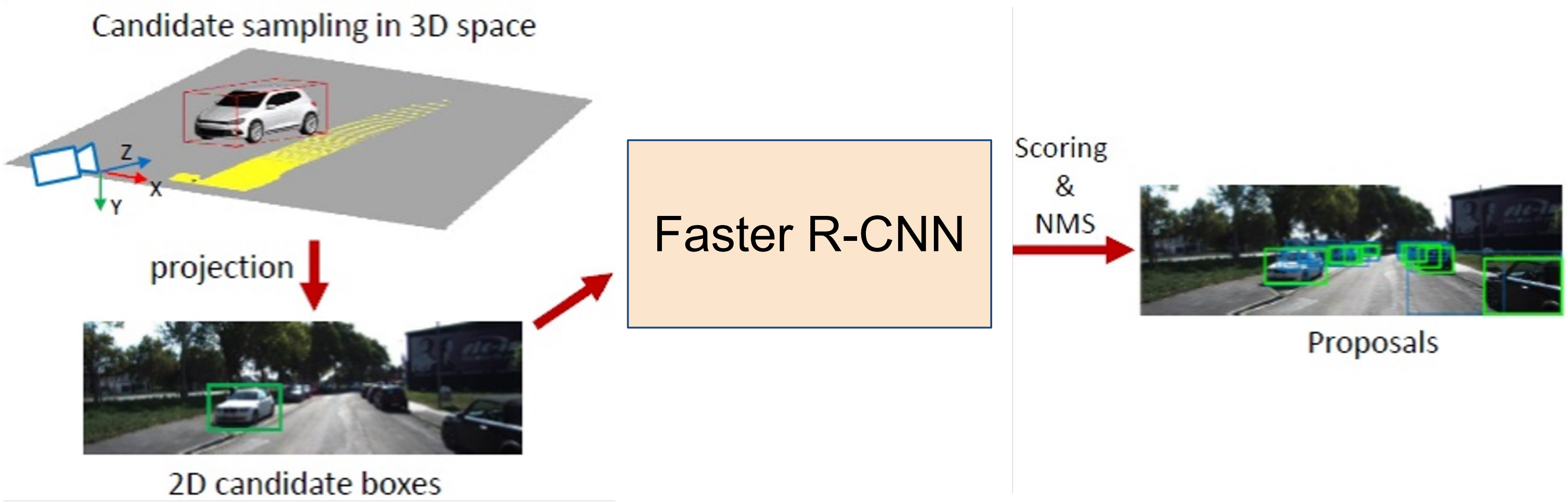

A point on the image plane corresponds to a **ray** in the 3D space

A 2D bounding box on an image is a **frustrum** in the 3D space

Localize an object in 3D:
The object can be anywhere in the **camera viewing frustrum**!

Image source: https://www.pcmag.com/encyclopedia_images/_FRUSTUM.GIF

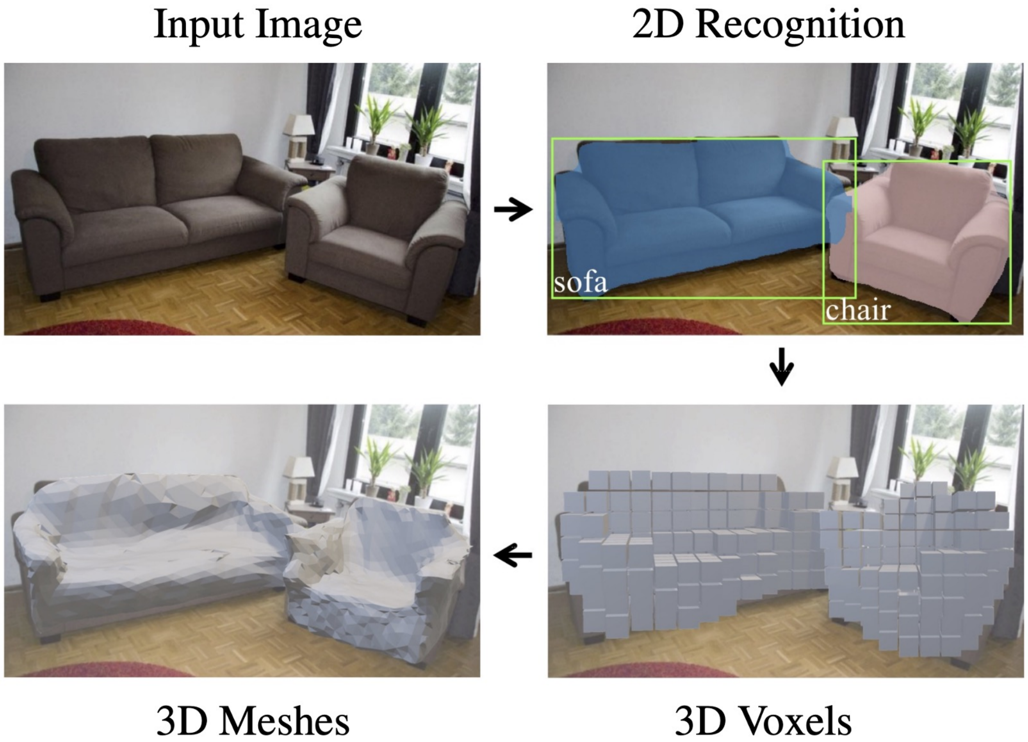
# 3D Object Detection: Monocular Camera




- Same idea as Faster RCNN, but proposals are in 3D
- 3D bounding box proposal, regress 3D box parameters + class score

Chen, Xiaozhi, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. "Monocular 3d object detection for autonomous driving." CVPR 2016.

# 3D Shape Prediction: Mesh R-CNN




Gkioxari et al., Mesh RCNN, ICCV 2019