

# Deep Learning

Lecture 13

# Object detection and segmentation

Part 1

# So far: Image Classification



This image is CC0 public domain

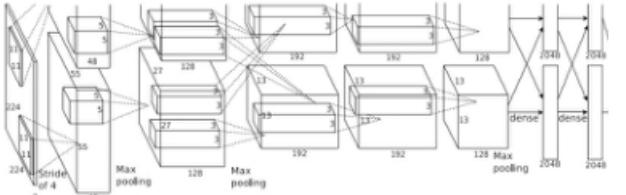


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

**Vector:**  
4096

**Fully-Connected :**  
4096 to 1000



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

# Other Computer Vision Tasks

## Semantic Segmentation



GRASS , CAT ,  
TREE , SKY

No objects, just pixels

## Classification + Localization



CAT

Single Object

## Object Detection



DOG , DOG , CAT

Multiple Object

## Instance Segmentation



DOG , DOG , CAT

[This image is CC0 public domain](#)

# Other Computer Vision Tasks

## Semantic Segmentation



**GRASS , CAT ,  
TREE , SKY**

No objects, just pixels

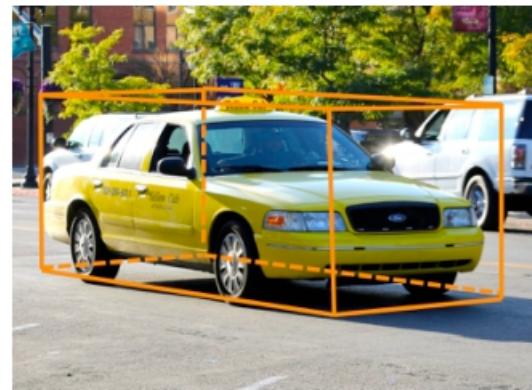
## 2D Object Detection



**DOG , DOG , CAT**

Object categories +  
2D bounding boxes

## 3D Object Detection



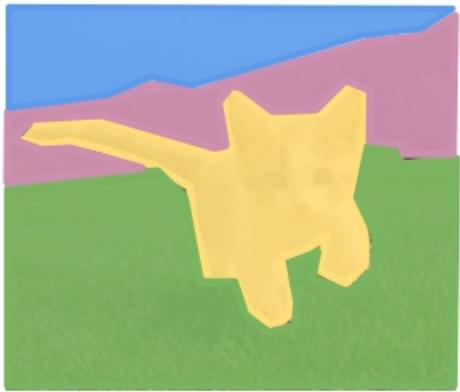
**Car**

Object categories +  
3D bounding boxes

This image is CC0 public domain

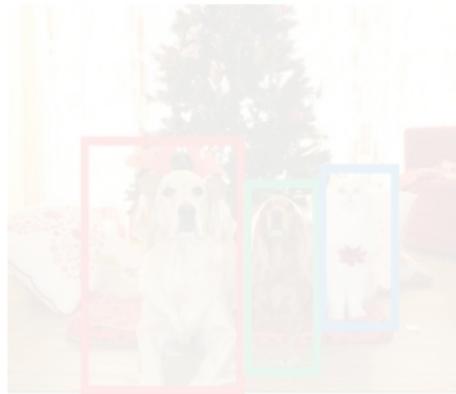
# Semantic Segmentation

## Semantic Segmentation



**GRASS , CAT ,  
TREE , SKY**

No objects, just pixels



This image is CC0 public domain

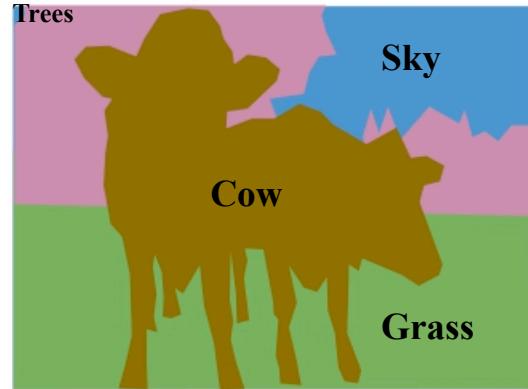
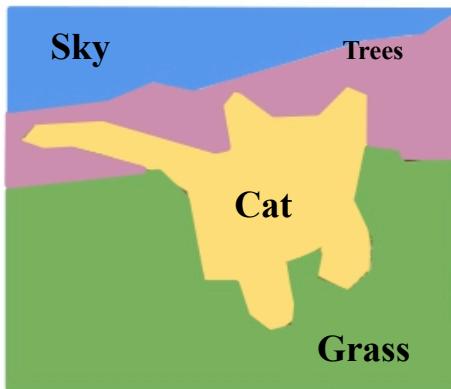
# Semantic Segmentation

Label each pixel in the image with a category label

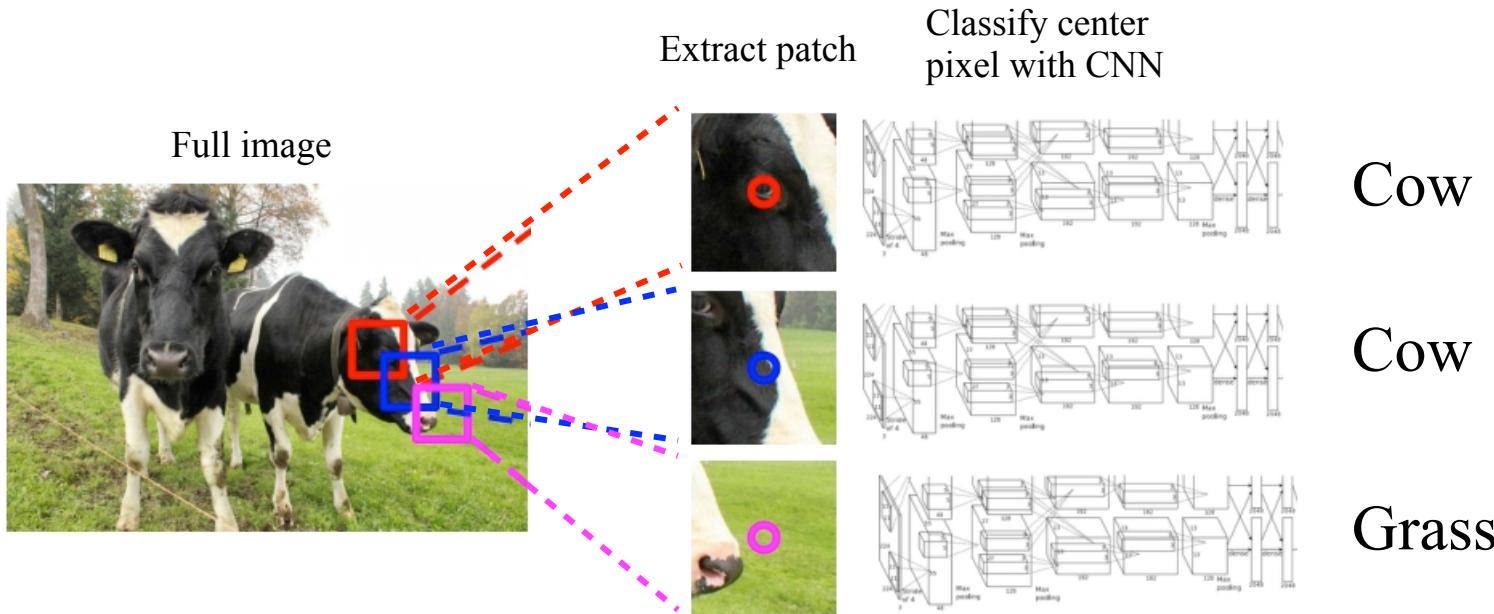
Don't differentiate instances, only care about pixels



[This image is CC0 public domain](#)



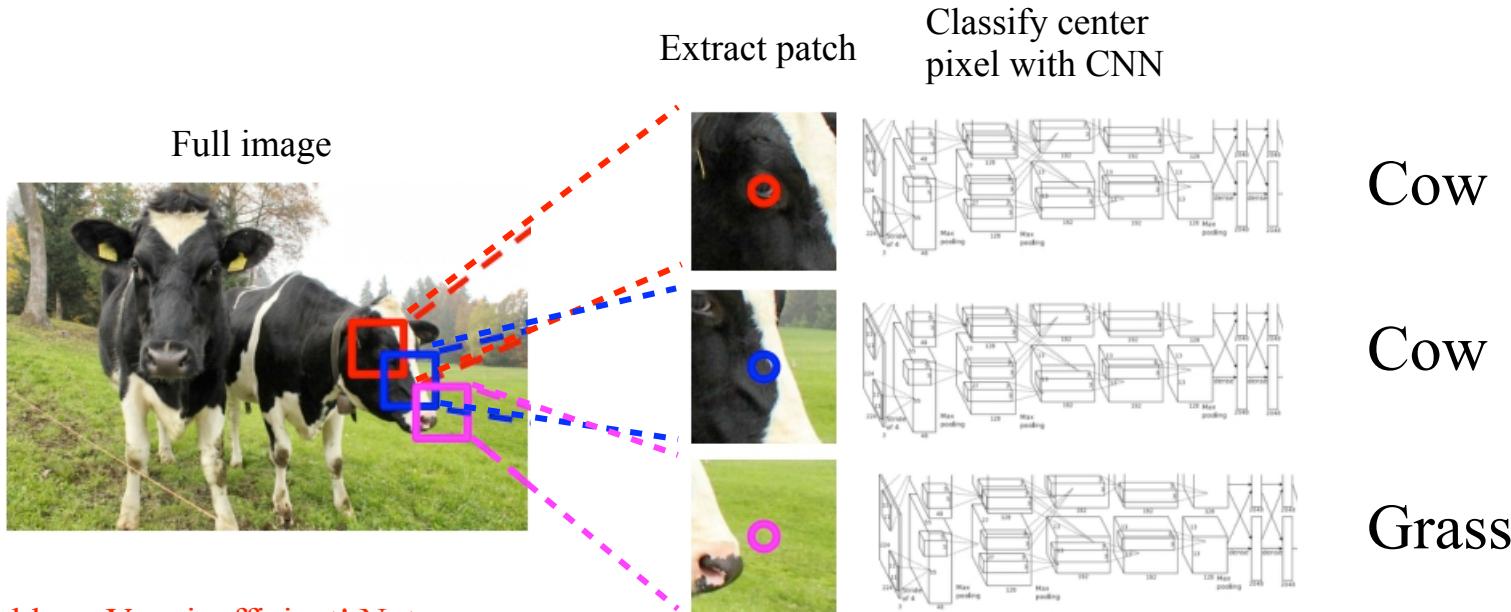
# Semantic Segmentation Idea: Sliding Window



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

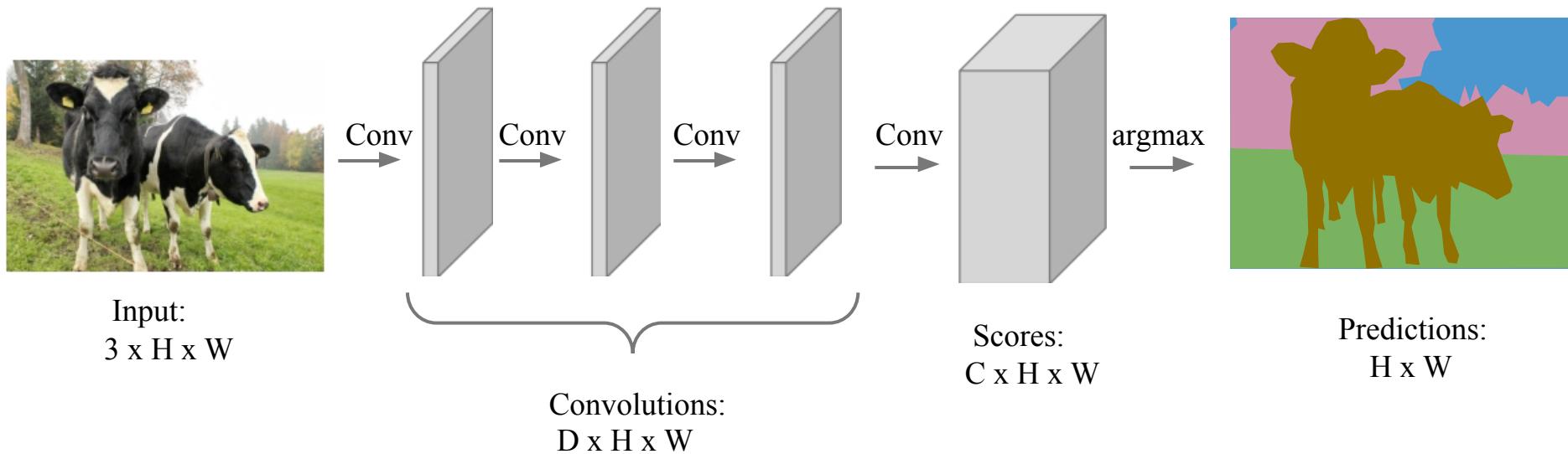


Problem: Very inefficient! Not reusing shared features between overlapping patches

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

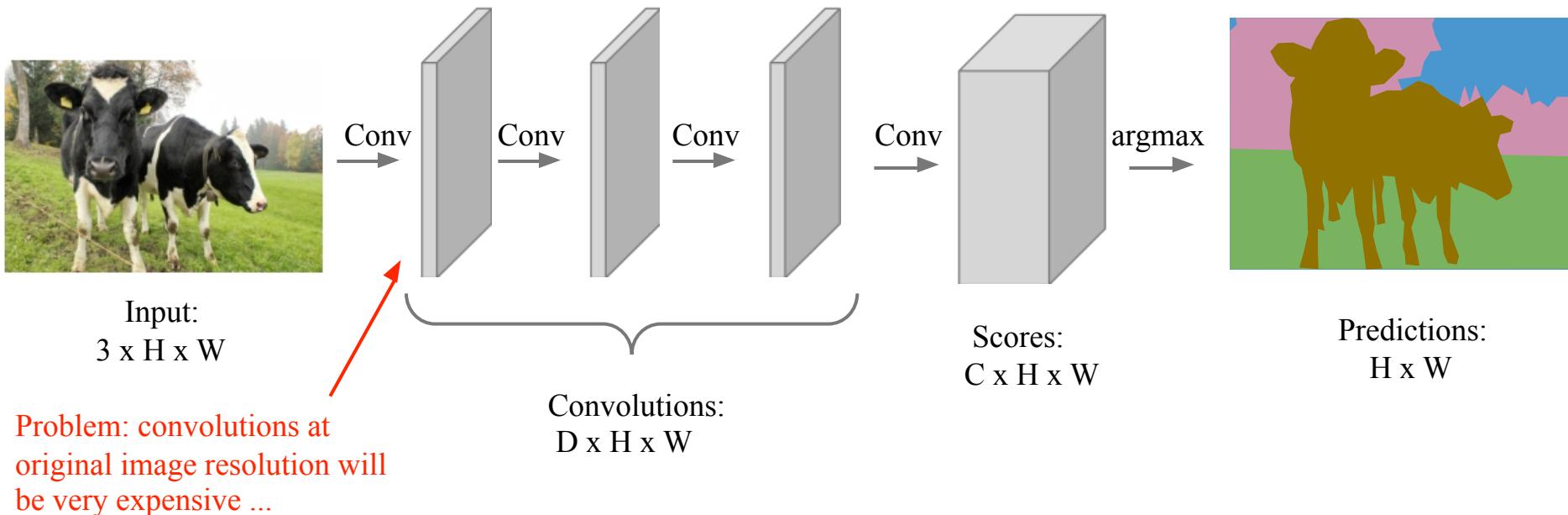
# Semantic Segmentation Idea: Fully Convolutional

Design a network as a bunch of convolutional layers  
to make predictions for pixels all at once!



# Semantic Segmentation Idea: Fully Convolutional

Design a network as a bunch of convolutional layers  
to make predictions for pixels all at once!



# Semantic Segmentation Idea: Fully Convolutional

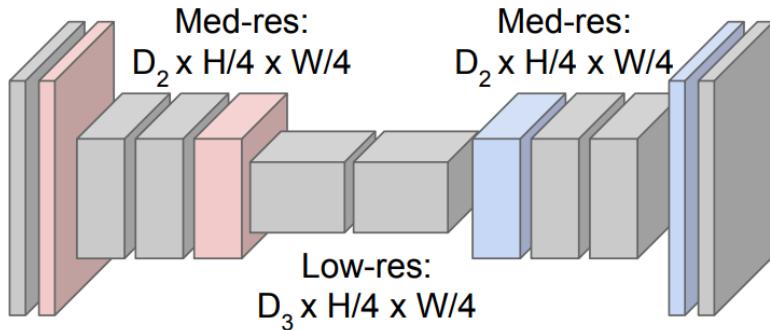
**Downsampling :**  
Pooling, strided convolution



Input:  
 $3 \times H \times W$

High-res:  
 $D_1 \times H/2 \times W/2$

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



**Upsampling :**  
???



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



1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Output: 4 x 4

Input: 2 x 2

**“Bed of Nails”**

1	2
3	4



1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

Output: 4 x 4

Input: 2 x 2

# 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

## Max Unpooling

Use positions from pooling layer

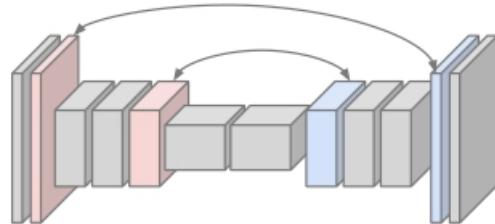
1	2
3	4

Rest of the network

0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

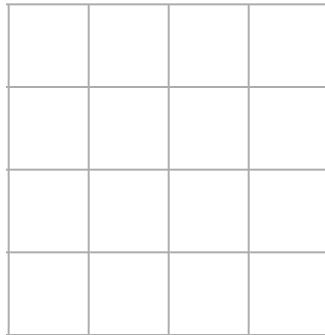
Output: 4 x 4

Corresponding pairs of  
downsampling and  
upsampling layers

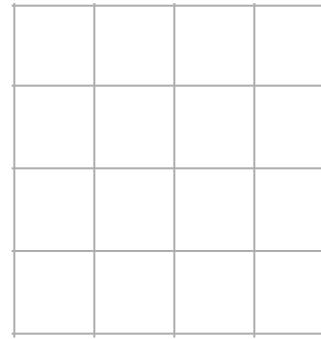


# Learnable Upsampling: Transpose Convolution

**Recall:** Typical  $3 \times 3$  convolution, stride 1 pad 1



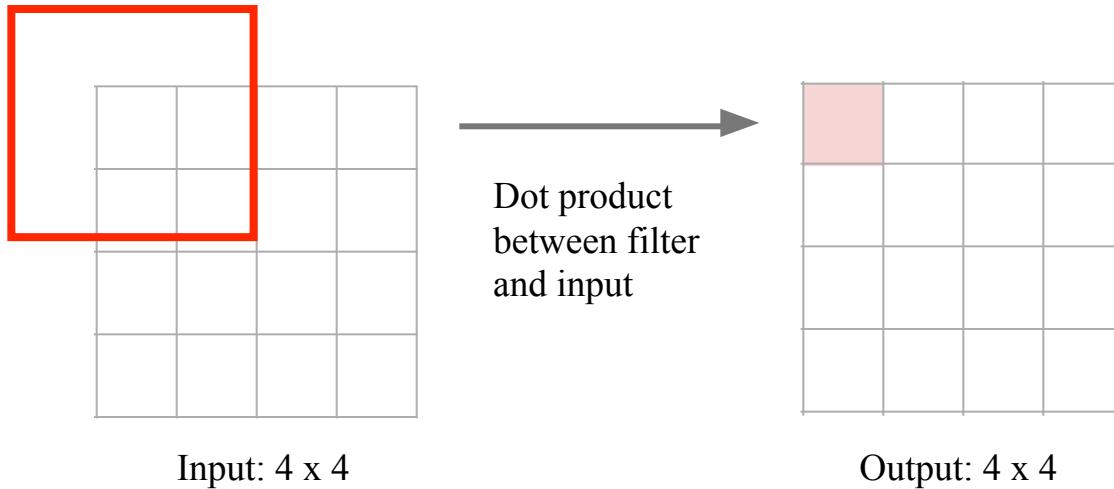
Input:  $4 \times 4$



Output:  $4 \times 4$

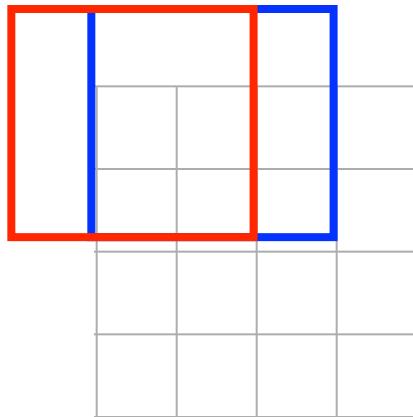
# Learnable Upsampling: Transpose Convolution

**Recall:** Normal  $3 \times 3$  convolution, stride 1 pad 1



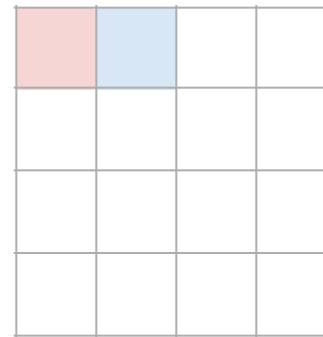
# Learnable Upsampling: Transpose Convolution

**Recall:** Normal  $3 \times 3$  convolution, stride 1 pad 1



Input:  $4 \times 4$

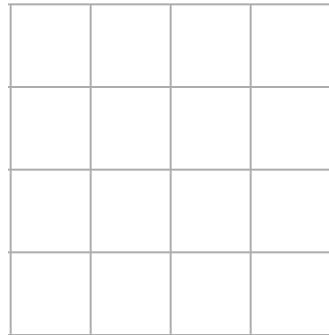
Dot product  
between filter  
and input



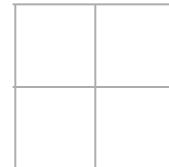
Output:  $4 \times 4$

# Learnable Upsampling: Transpose Convolution

**Recall:** Normal  $3 \times 3$  convolution, stride 2 pad 1



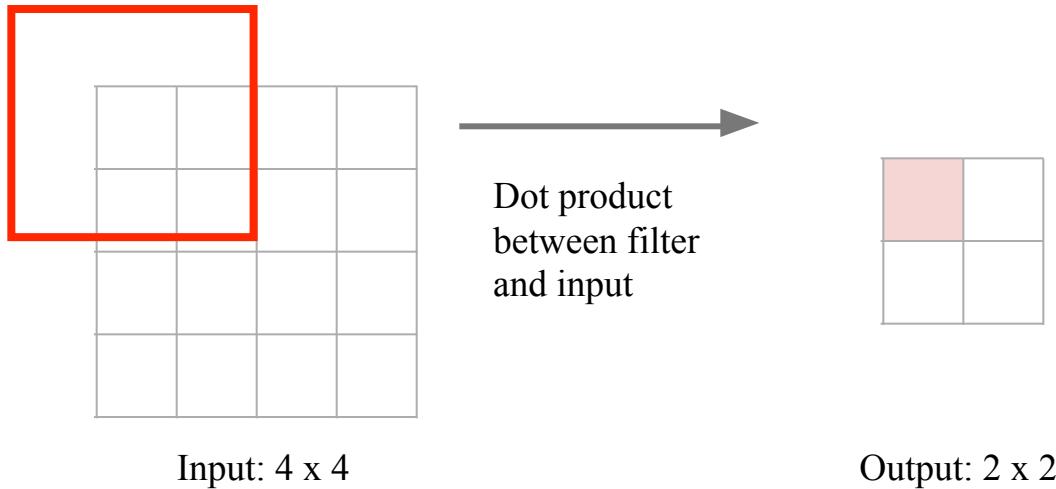
Input:  $4 \times 4$



Output:  $2 \times 2$

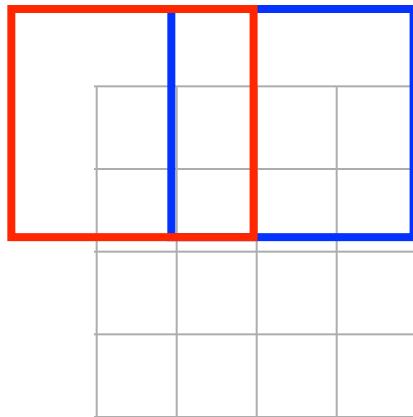
# Learnable Upsampling: Transpose Convolution

**Recall:** Normal  $3 \times 3$  convolution, stride 2 pad 1



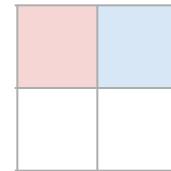
# Learnable Upsampling: Transpose Convolution

**Recall:** Normal  $3 \times 3$  convolution, stride 2 pad 1



Input:  $4 \times 4$

Dot product  
between filter  
and input



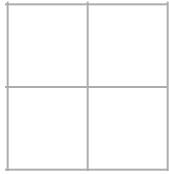
Output:  $2 \times 2$

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

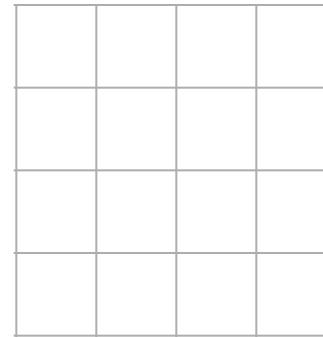
Stride gives ratio between  
movement in input and  
output

# Learnable Upsampling: Transpose Convolution

$3 \times 3$  **transpose** convolution, stride 2 pad 1



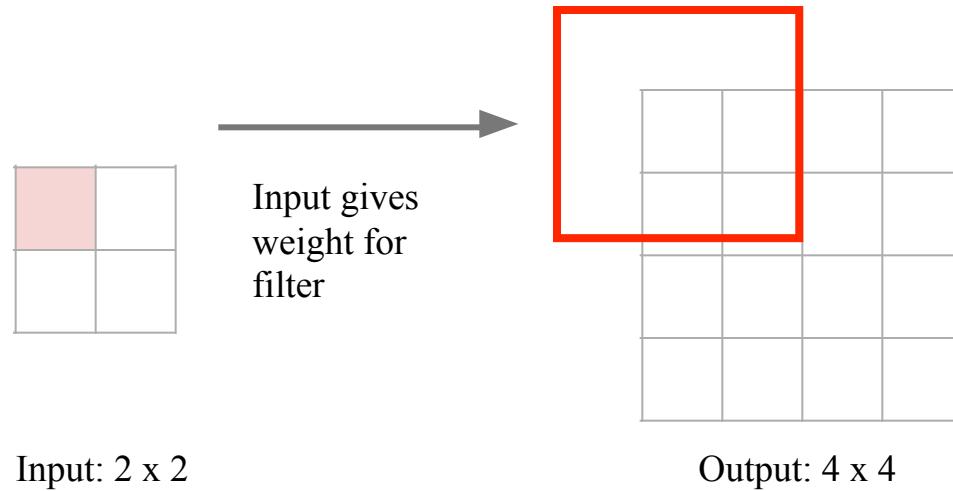
Input:  $2 \times 2$



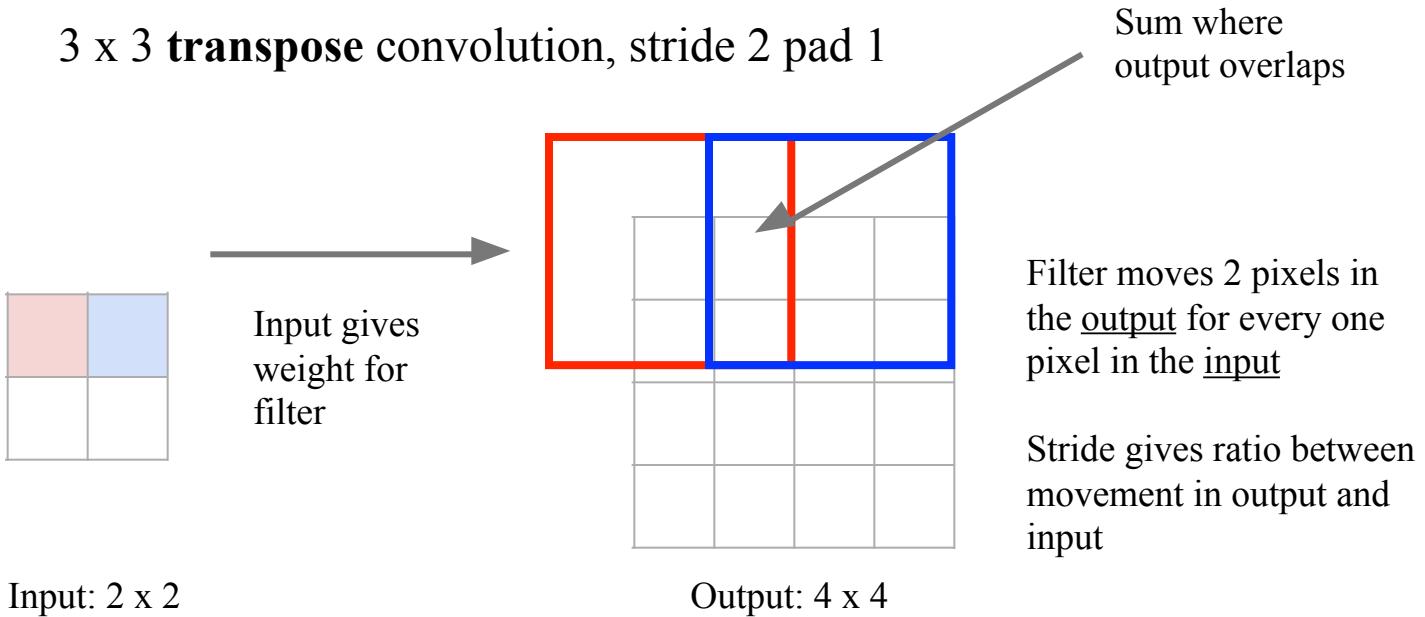
Output:  $4 \times 4$

# Learnable Upsampling: Transpose Convolution

3 x 3 transpose convolution, stride 2 pad 1



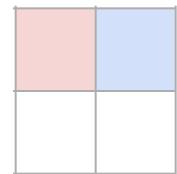
# Learnable Upsampling: Transpose Convolution



# Learnable Upsampling: Transpose Convolution

**Other names:**

- Deconvolution (bad)
- Upconvolution
- Fractionally strided convolution
- Backward strided convolution

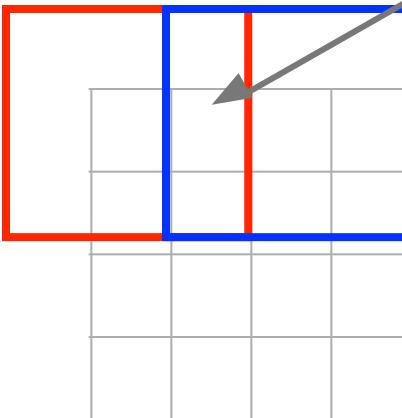


Input: 2 x 2

3 x 3 transpose convolution, stride 2 pad 1



Input gives weight for filter



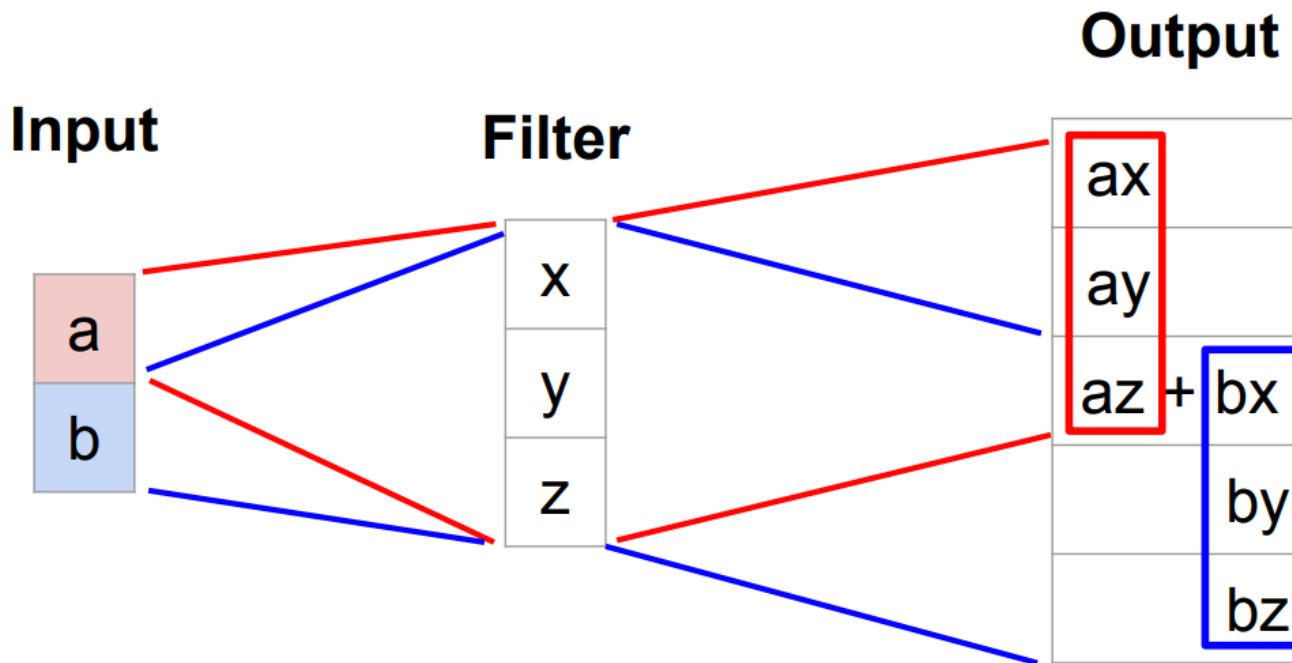
Output: 4 x 4

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

Stride gives ratio between movement in output and input

Sum where output overlaps

# Learnable Upsampling: 1D Example



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

Need to crop one pixel from output to make output exactly 2x input

# Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

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

$$\begin{bmatrix} x & y & x & 0 & 0 & 0 \\ 0 & x & y & x & 0 & 0 \\ 0 & 0 & x & y & x & 0 \\ 0 & 0 & 0 & x & y & x \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel  
size=3, stride=1, padding=1

# Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

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

$$\begin{bmatrix} x & y & x & 0 & 0 & 0 \\ 0 & x & y & x & 0 & 0 \\ 0 & 0 & x & y & x & 0 \\ 0 & 0 & 0 & x & y & x \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

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

$$\begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix}$$

When stride=1, convolution transpose is just a regular convolution (with different padding rules)

# Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

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

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

Example: 1D conv, kernel  
size=3, stride=2, 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, stride=2, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

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

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

When stride>1, convolution transpose is no longer a normal convolution!

# Semantic Segmentation Idea: Fully Convolutional

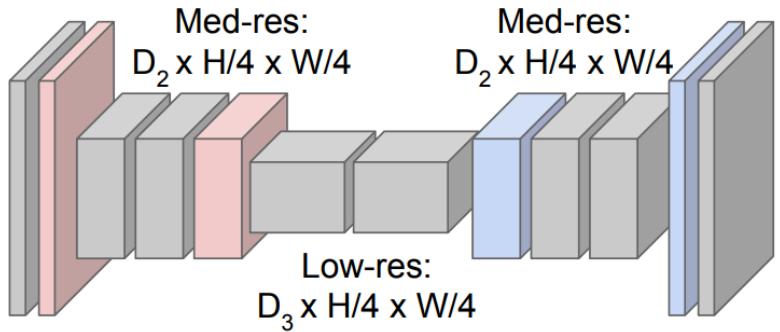
**Downsampling :**  
Pooling, strided  
convolution



Input:  
 $3 \times H \times W$

High-res:  
 $D_1 \times H/2 \times W/2$

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



**Upsampling :**  
Unpooling or strided  
transpose convolution



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

# 2D Object Detection

## 2D Object Detection



**DOG , DOG , CAT**

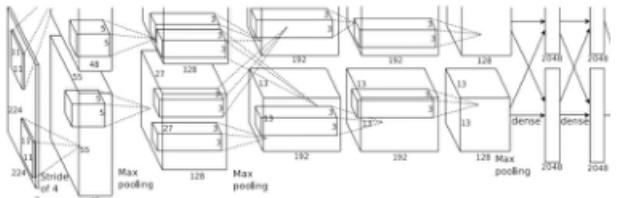
Object categories +  
2D bounding boxes

This image is CC0 public domain

# 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

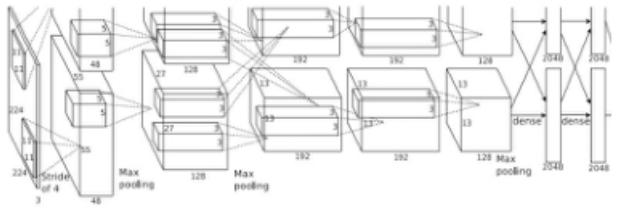
**Box  
Coordinates**  
(x, y, w, h)

Treat localization as a  
regression problem!

# Classification + Localization



This image is CC0 public domain.



Fully  
Connected :  
4096 to 1000

Vector:  
4096

Fully  
Connected :  
4096 to 4

Treat localization as a  
regression problem!

Class Scores

Cat: 0.9  
Dog: 0.05  
Car: 0.01

...

Correct label:  
Cat

Softmax  
Loss

Box

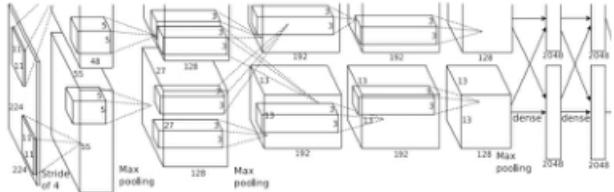
Coordinates → L2 Loss  
( $x, y, w, h$ )

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

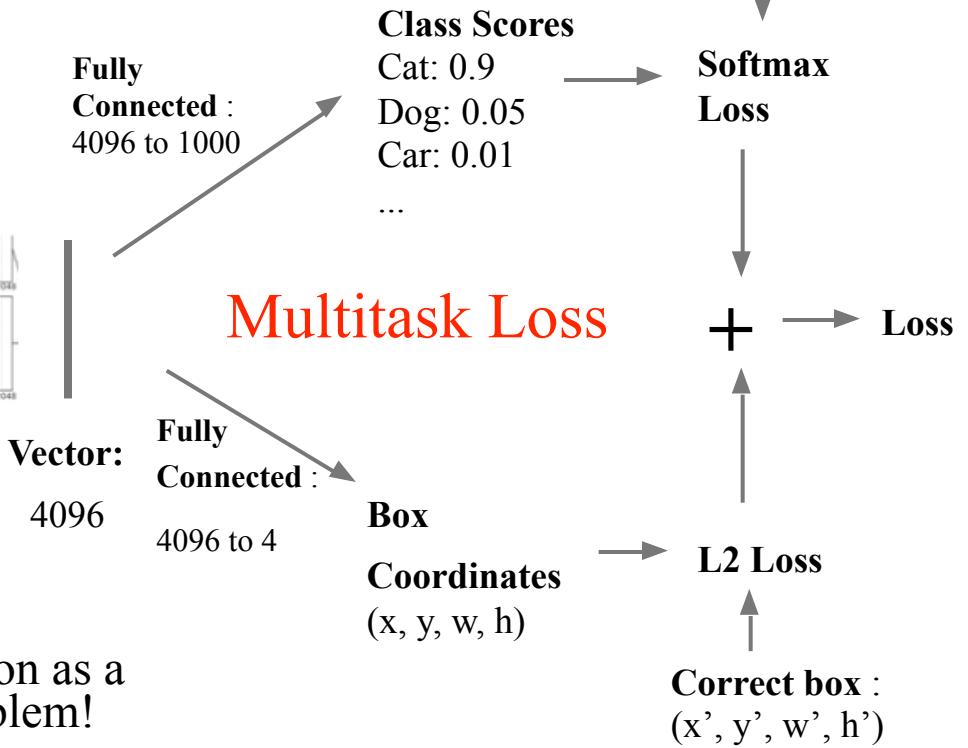
# Classification + Localization



This image is CC0 public domain.



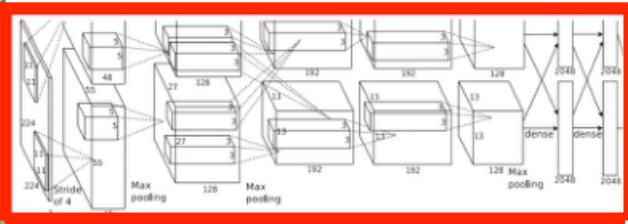
Treat localization as a  
regression problem!



# Classification + Localization



This image is CC0 public domain.



Often pretrained on ImageNet  
(Transfer learning)

Treat localization as a  
regression problem!

Vector:  
4096

Fully  
Connected :  
4096 to 1000

Class Scores

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

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

Correct label:  
Cat

Softmax  
Loss

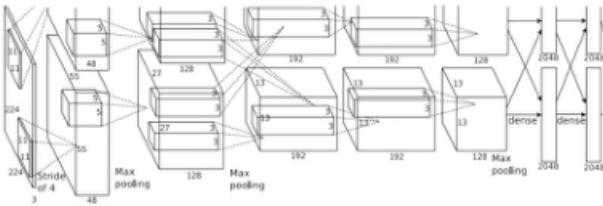
+

Loss

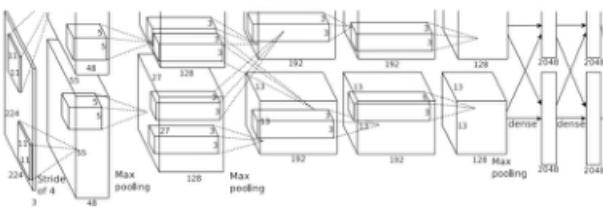
Correct box :  
( $x', y', w', h'$ )

L2 Loss

# Object Detection as Regression?



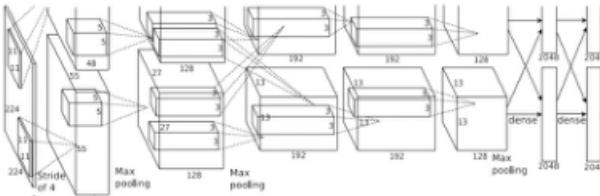
CAT: (x, y, w, h)



DOG: (x, y, w, h)

DOG: (x, y, w, h)

CAT: (x, y, w, h)



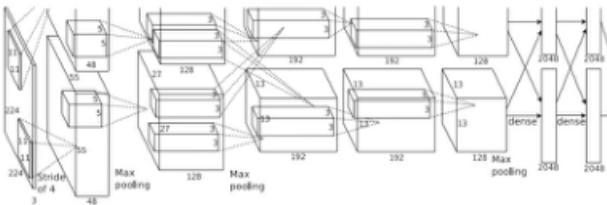
DUCK: (x, y, w, h)

DUCK: (x, y, w, h)

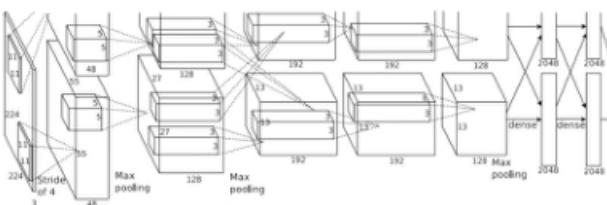
....

# Object Detection as Regression?

Each image needs a different number of outputs!



CAT: (x, y, w, h) **4 numbers**

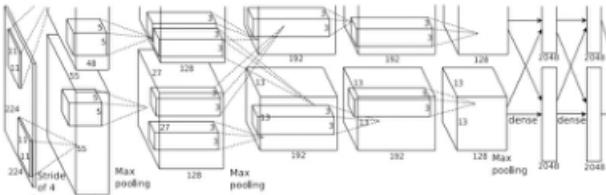


DOG: (x, y, w, h)

**16 numbers**

DOG: (x, y, w, h)

CAT: (x, y, w, h)



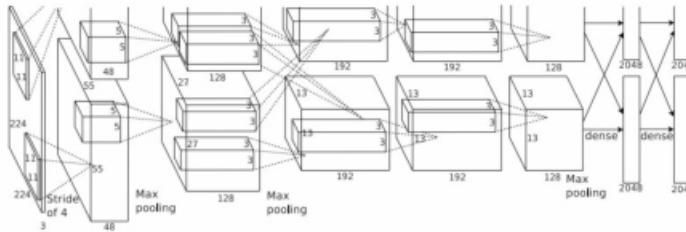
DUCK: (x, y, w, h) **Many**

DUCK: (x, y, w, h) **numbers!**

....

# Object Detection as Classification: Sliding Window

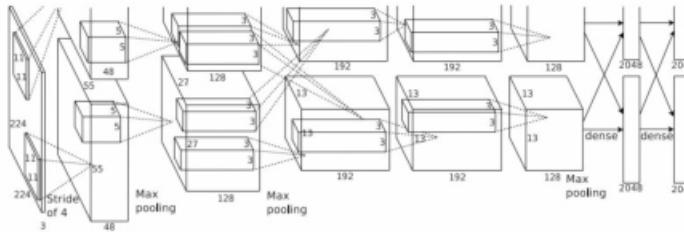
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? NO  
Cat? NO  
Background? YES

# Object Detection as Classification: Sliding Window

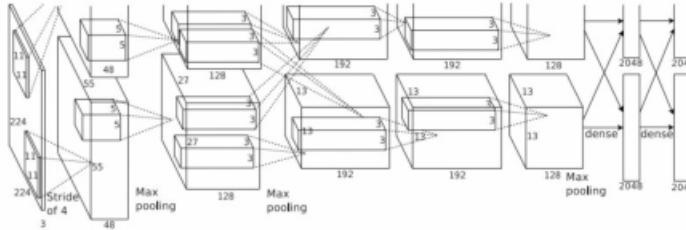
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? YES  
Cat? NO  
Background? NO

# Object Detection as Classification: Sliding Window

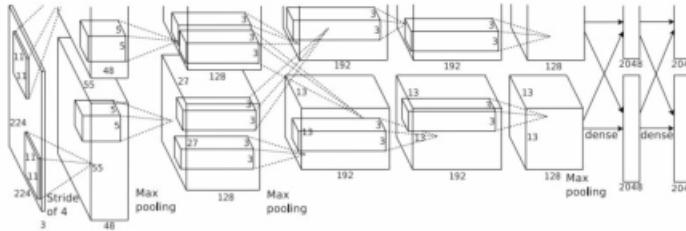
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? YES  
Cat? NO  
Background? NO

# Object Detection as Classification: Sliding Window

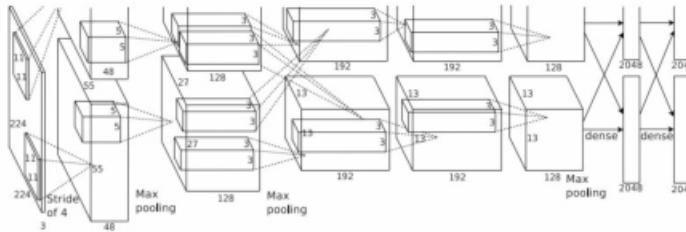
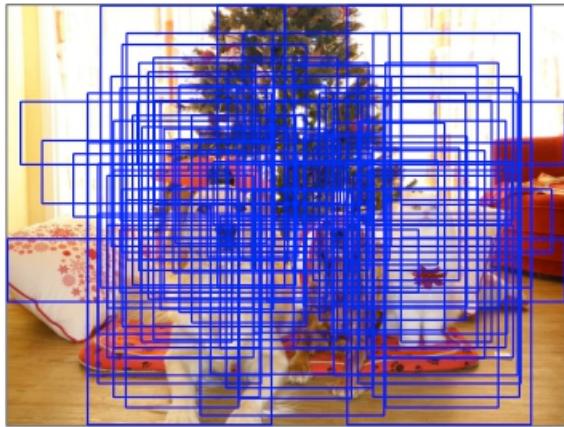
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? NO  
Cat? YES  
Background? NO

# Object Detection as Classification: Sliding Window

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

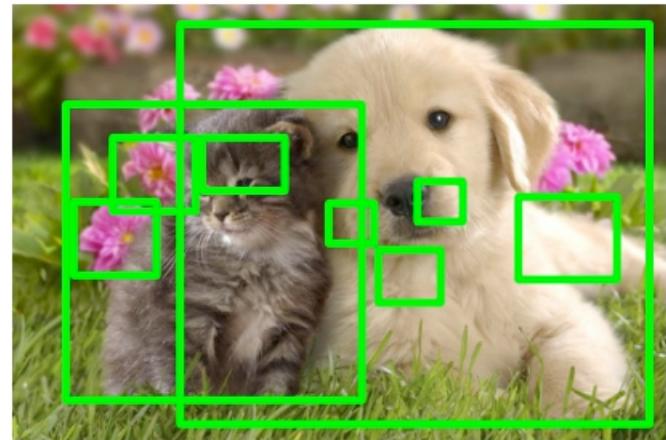


Dog? NO  
Cat? YES  
Background? NO

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

# 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

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.  
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

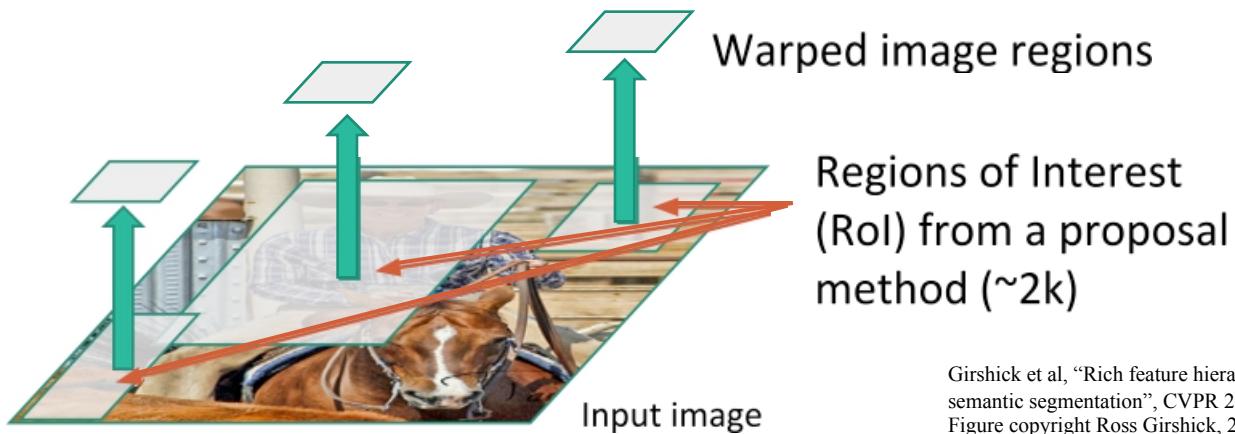
# R-CNN



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

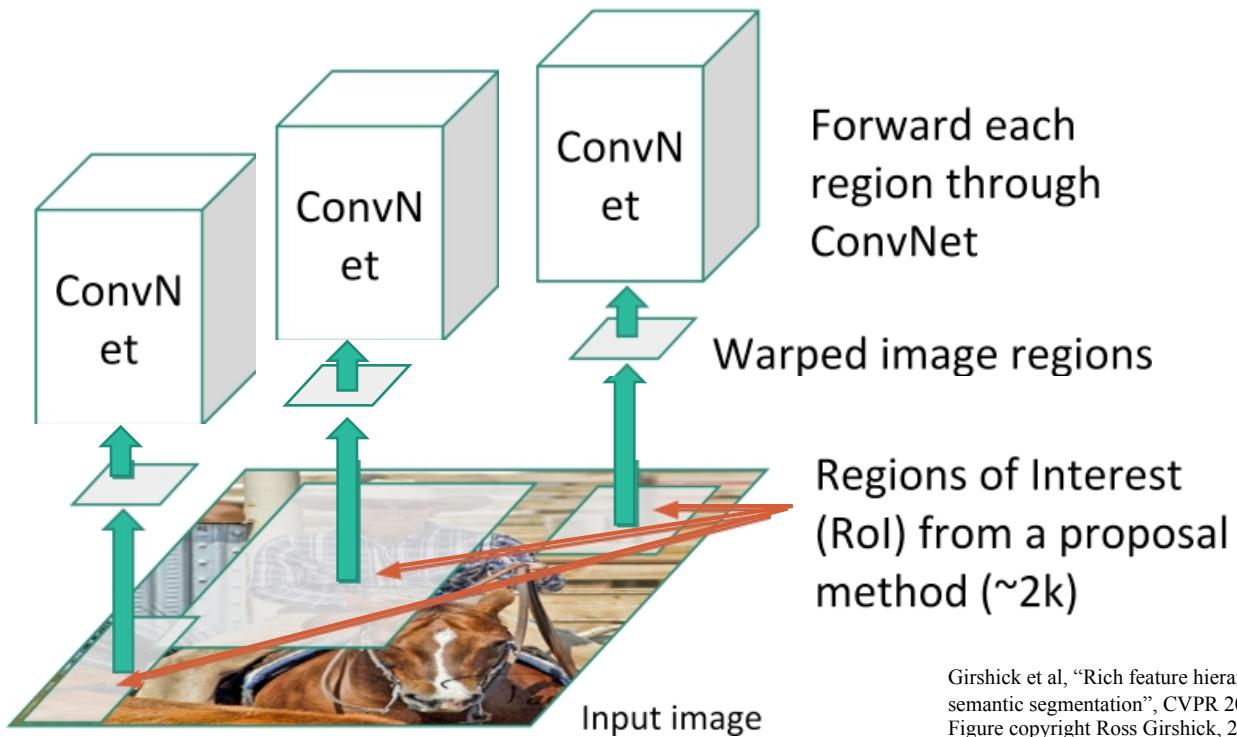
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.  
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

# R-CNN



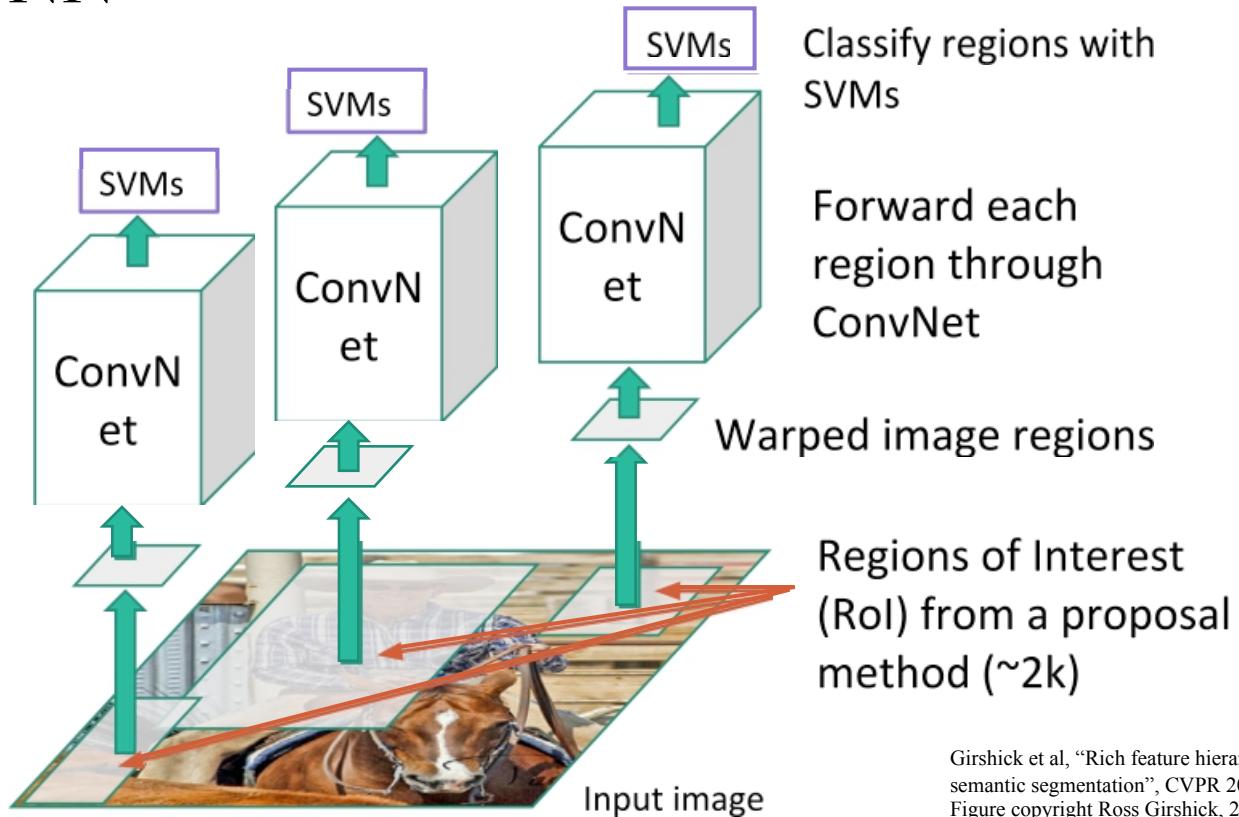
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.  
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

# R-CNN



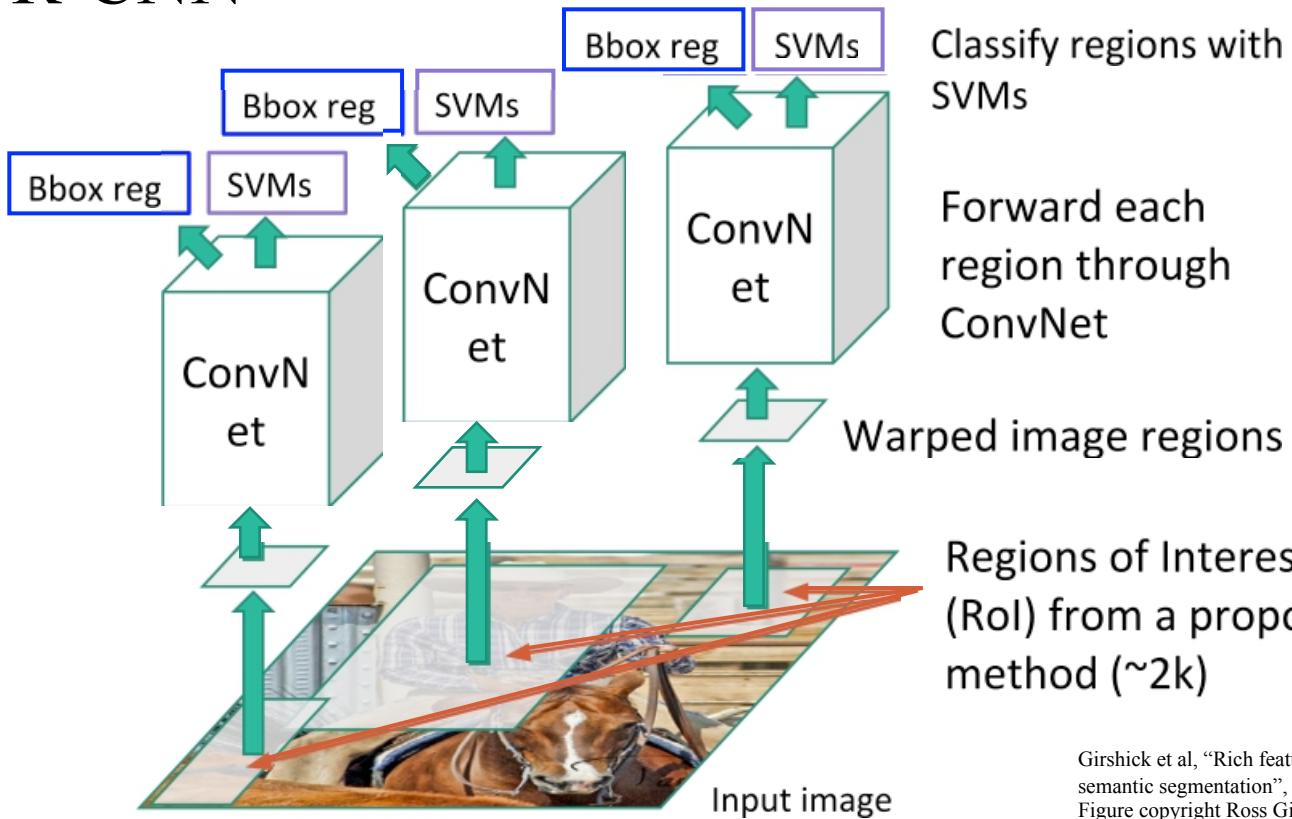
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.  
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

# R-CNN



Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.  
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

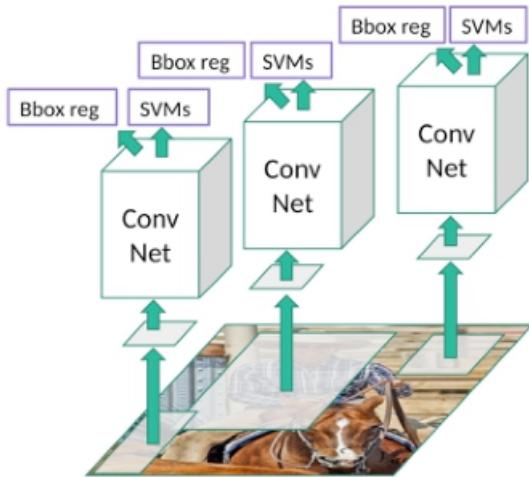
# R-CNN



Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.  
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

# R-CNN: Problems

- Ad hoc training objectives
  - Fine-tune network with softmax classifier (log loss)
  - Train post-hoc linear SVMs (hinge loss)
  - Train post-hoc bounding-box regressions (least squares)
- Training is slow (84h), takes a lot of disk space
- Inference (detection) is slow
  - 47s / image with VGG16 [Simonyan & Zisserman. ICLR15]
  - Fixed by SPP-net [He et al. ECCV14]



Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.  
Slide copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

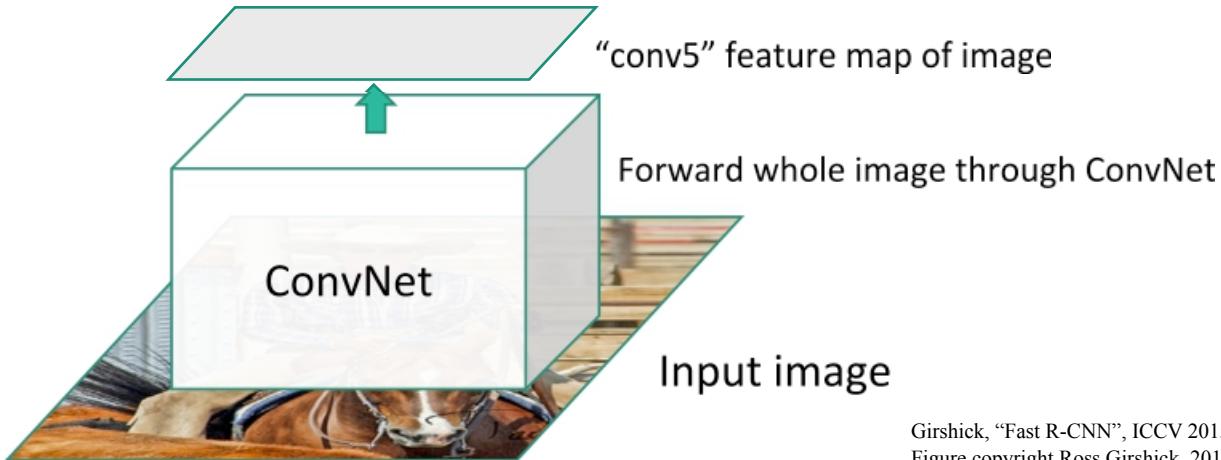
# Fast R-CNN



Input image

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

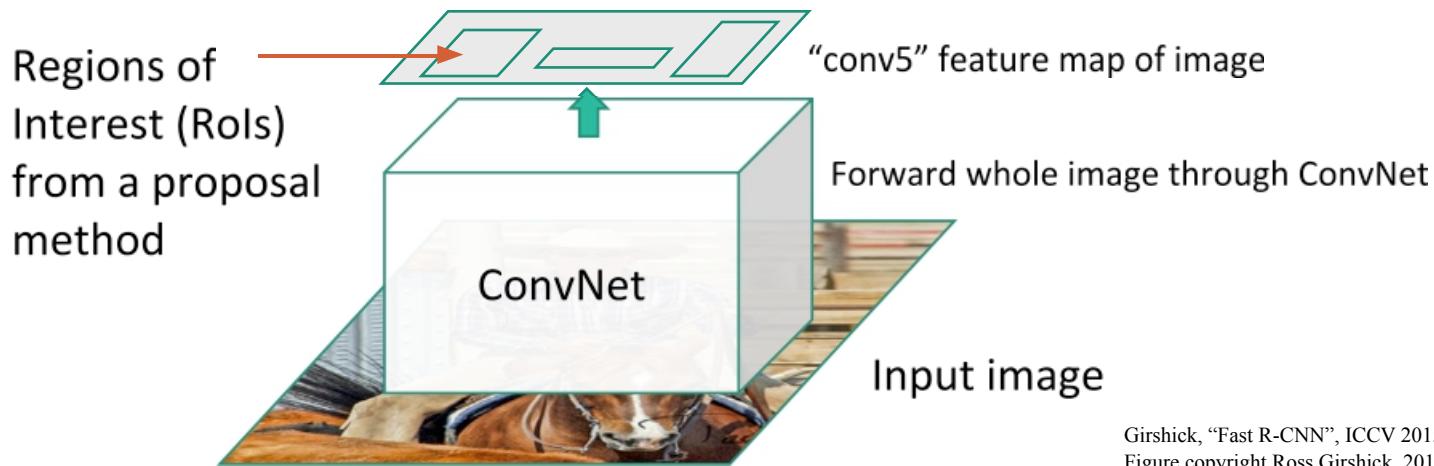
# Fast R-CNN



Girshick, “Fast R-CNN”, ICCV 2015.

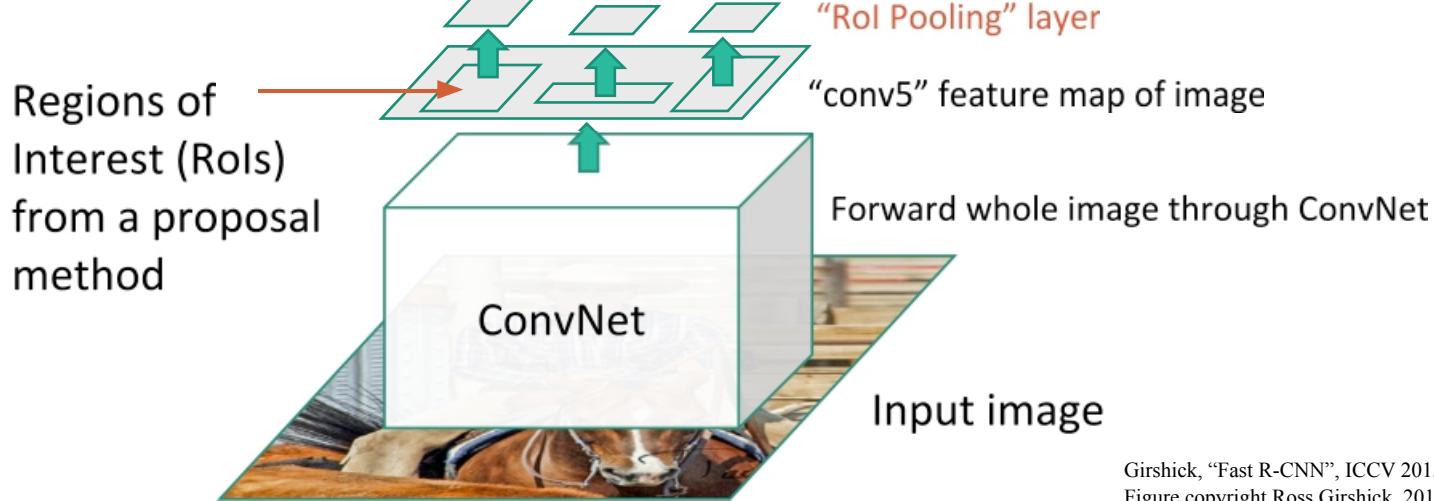
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

# Fast R-CNN



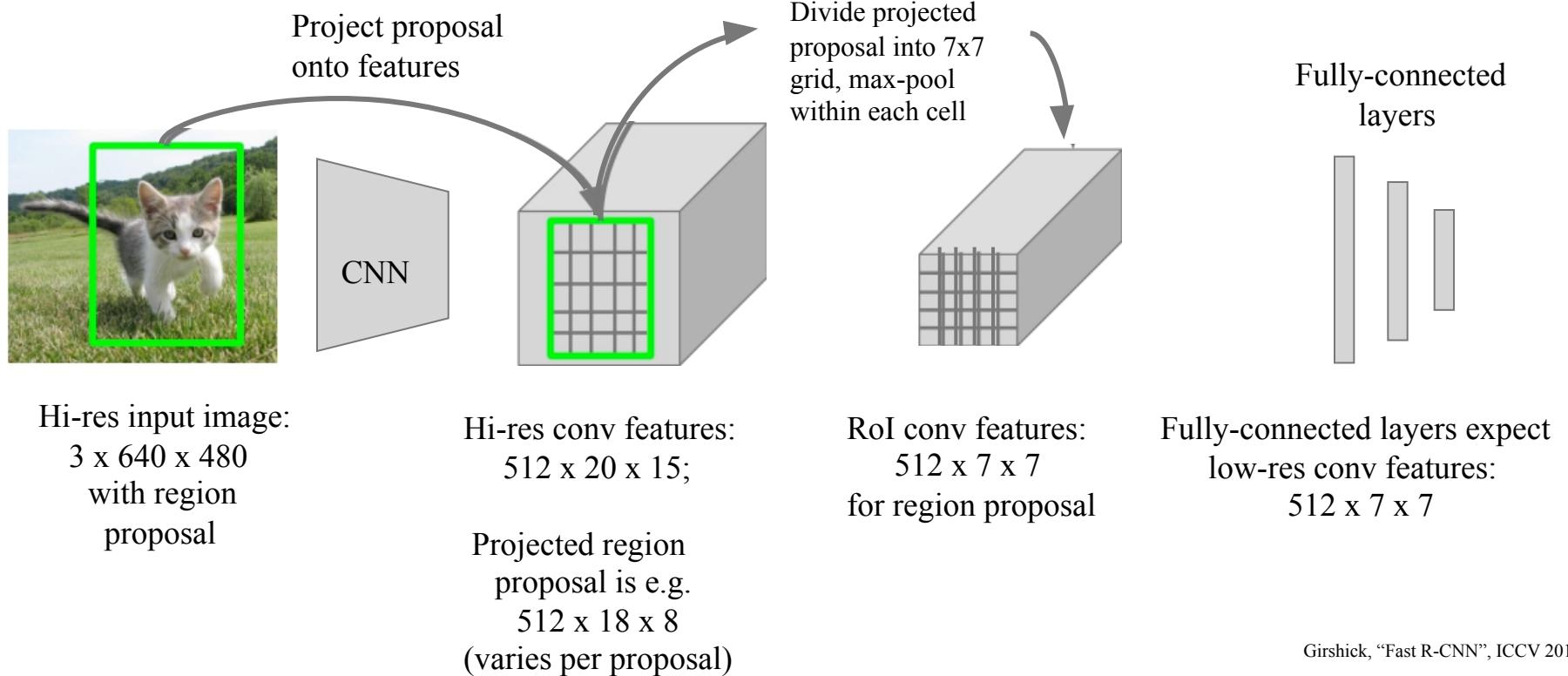
Girshick, “Fast R-CNN”, ICCV 2015.  
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

# Fast R-CNN

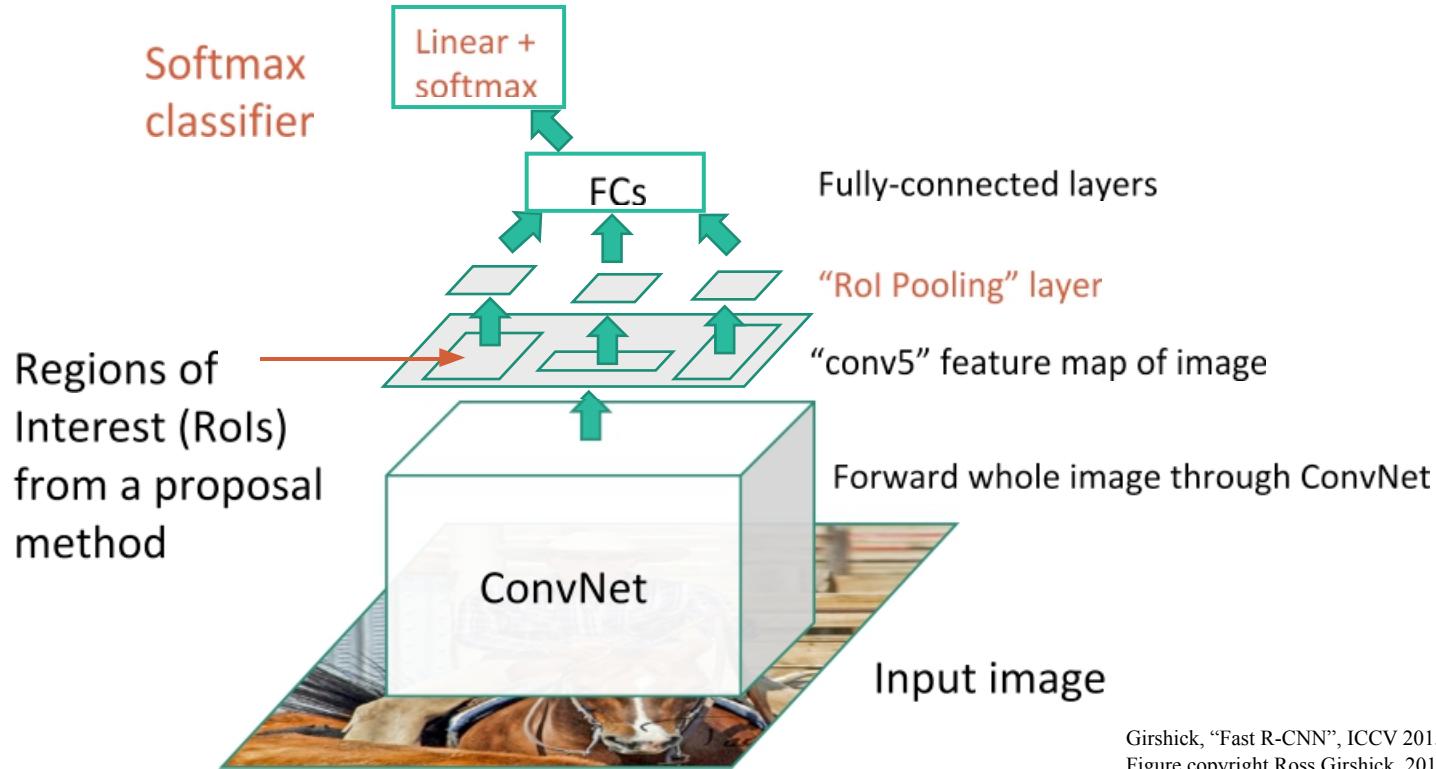


Girshick, "Fast R-CNN", ICCV 2015.  
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

# Fast R-CNN: RoI Pooling

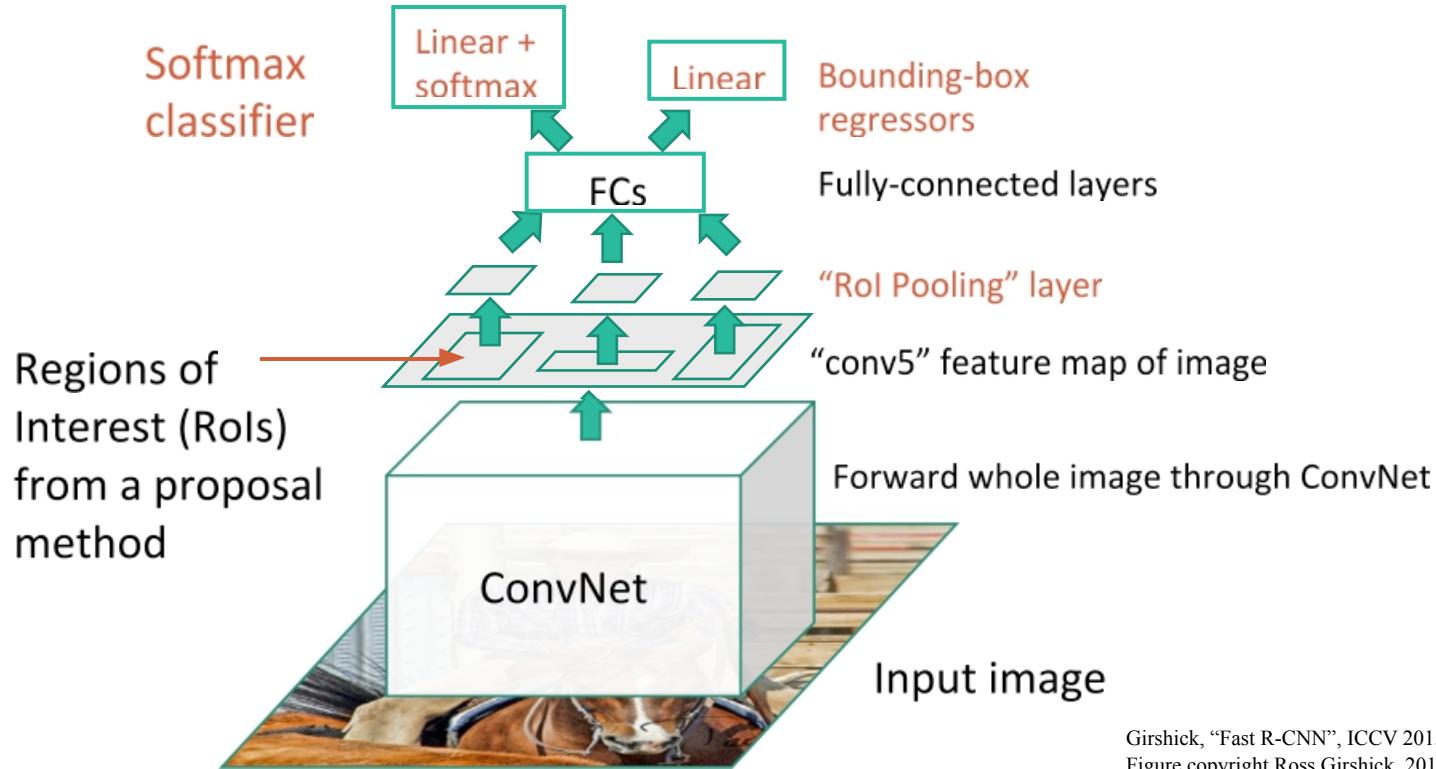


# Fast R-CNN



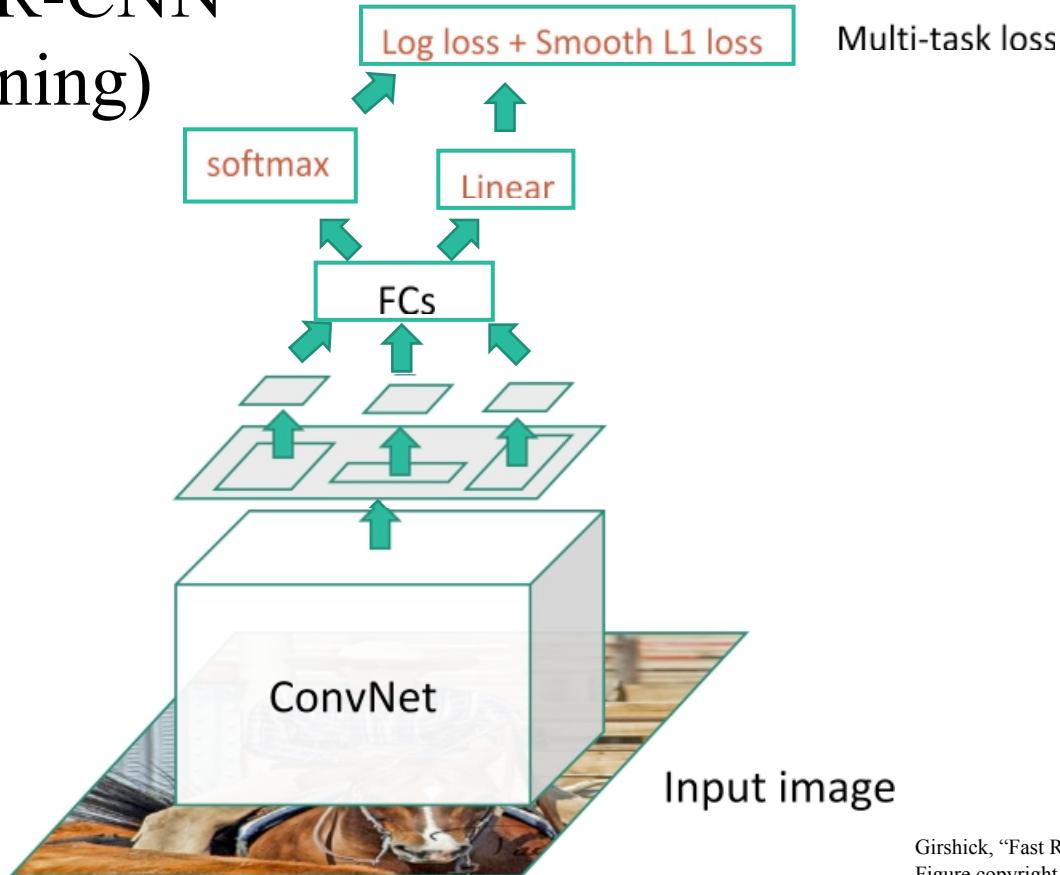
Girshick, "Fast R-CNN", ICCV 2015.  
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

# Fast R-CNN



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

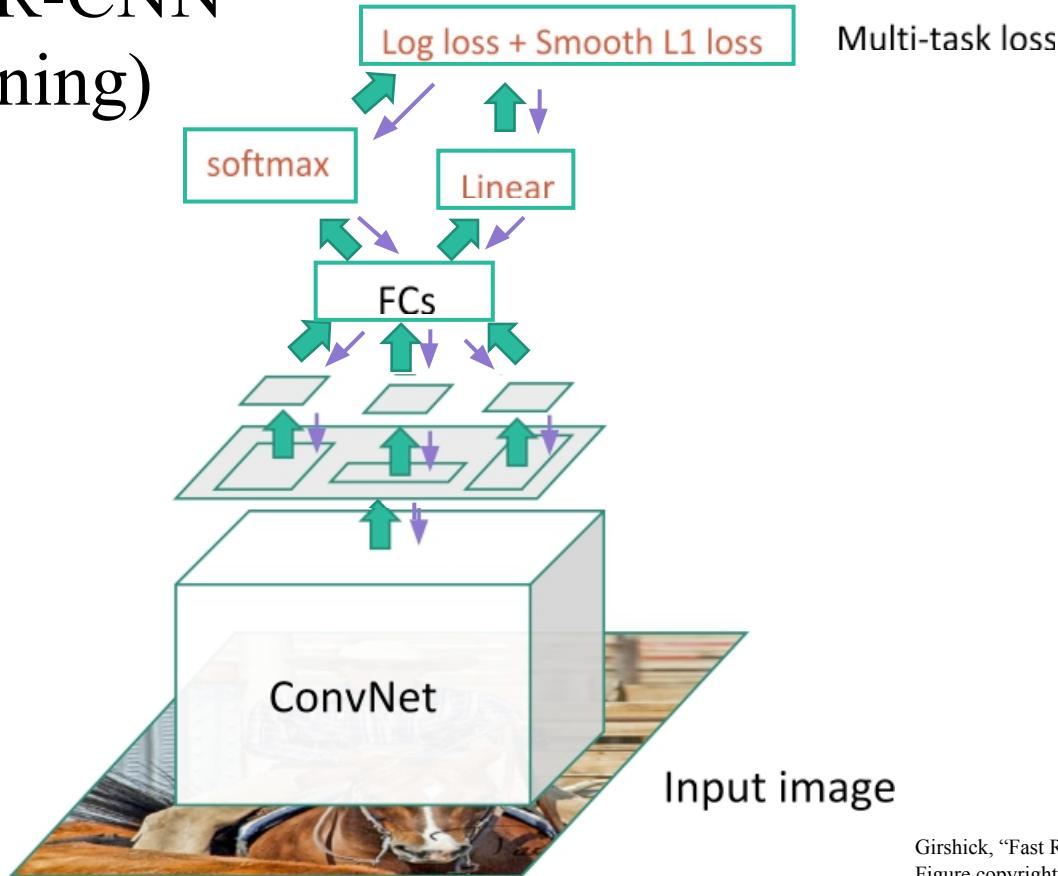
# Fast R-CNN (Training)



Girshick, “Fast R-CNN”, ICCV 2015.

Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

# Fast R-CNN (Training)

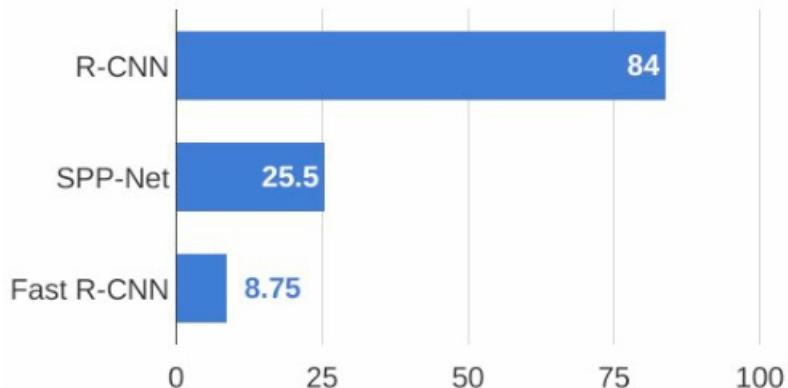


Girshick, “Fast R-CNN”, ICCV 2015.

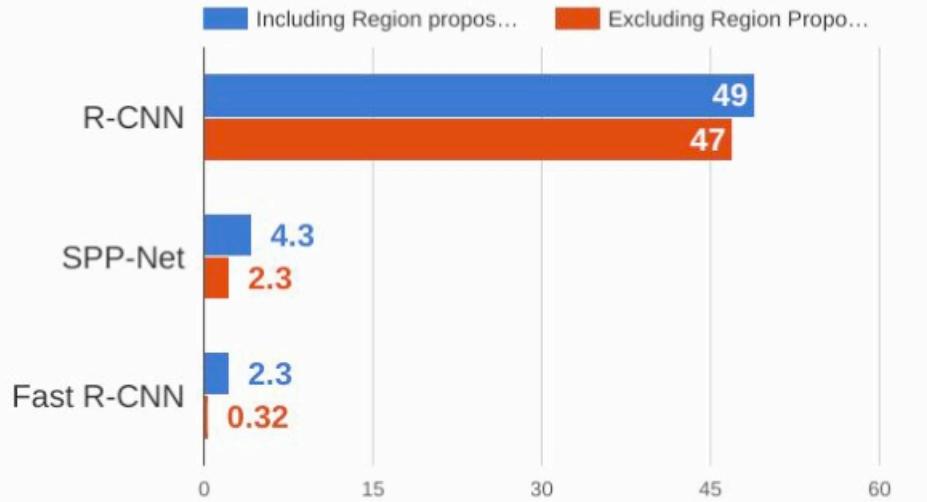
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

# R-CNN vs SPP vs Fast R-CNN

**Training time (Hours)**



**Test time (seconds)**

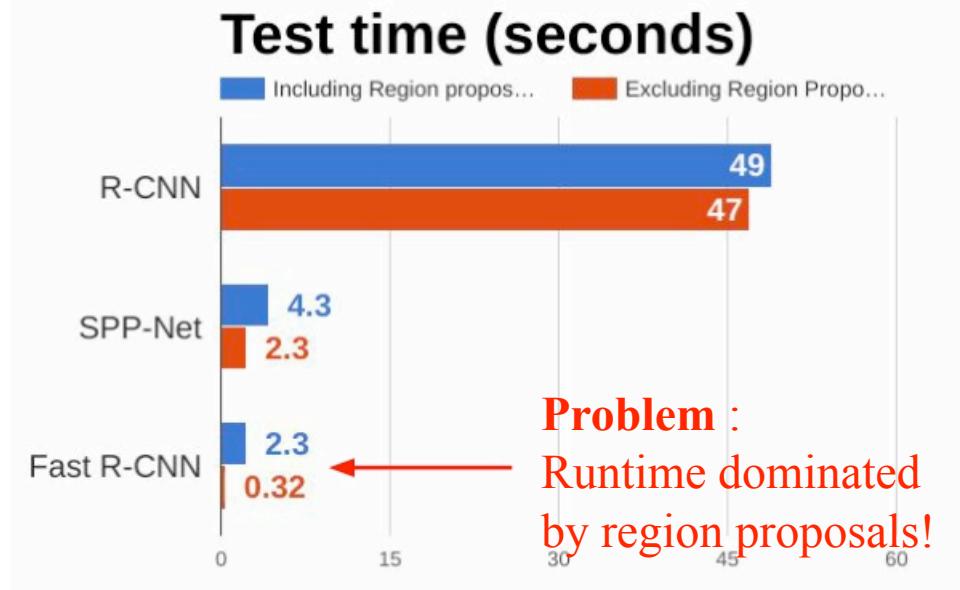
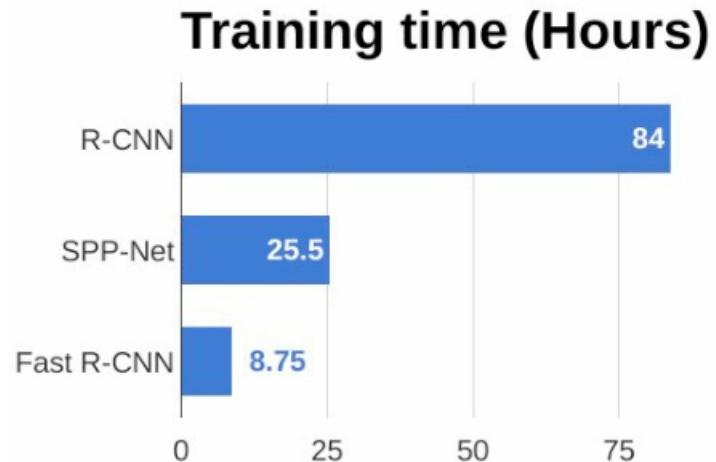


Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

He et al, “Spatial pyramid pooling in deep convolutional networks for visual recognition”, ECCV 2014

Girshick, “Fast R-CNN”, ICCV 2015

# R-CNN vs SPP 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

# Faster R-CNN:

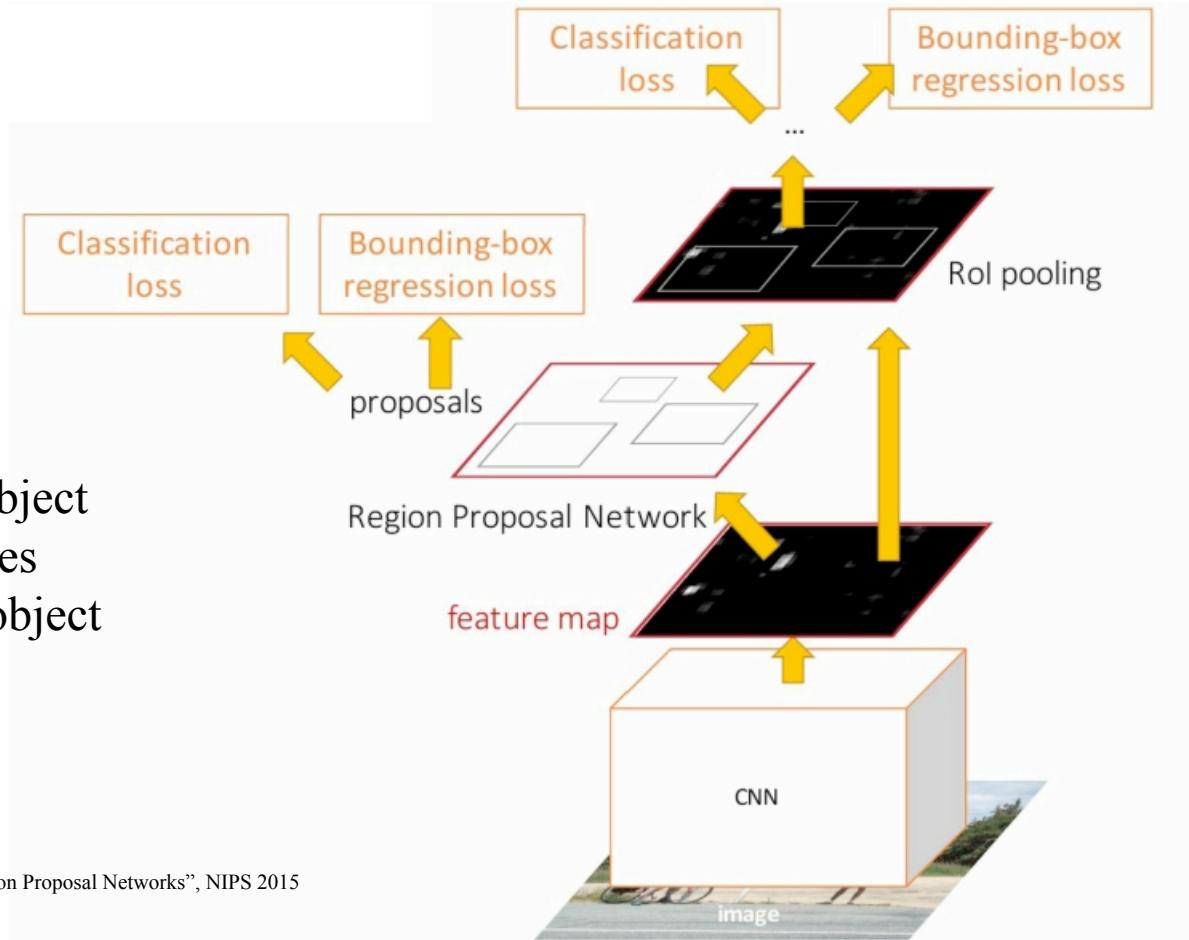
Make CNN do proposals!

Insert **Region Proposal**

**Network (RPN)** to predict  
proposals from features

Jointly train with 4 losses:

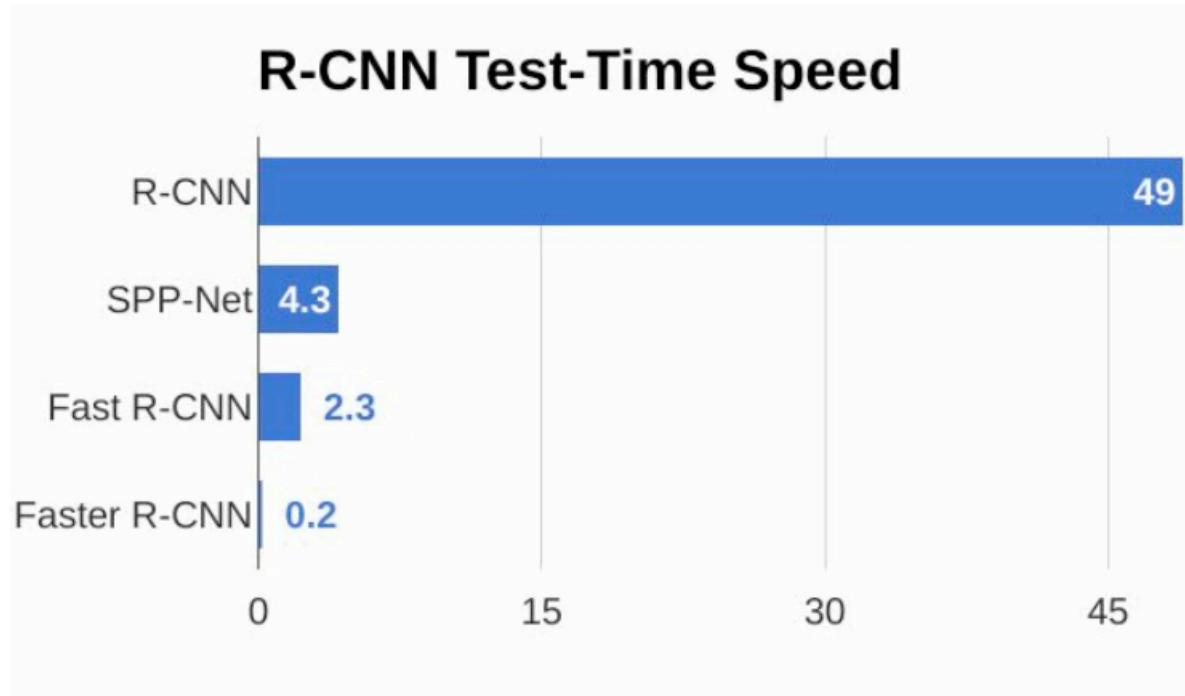
1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates



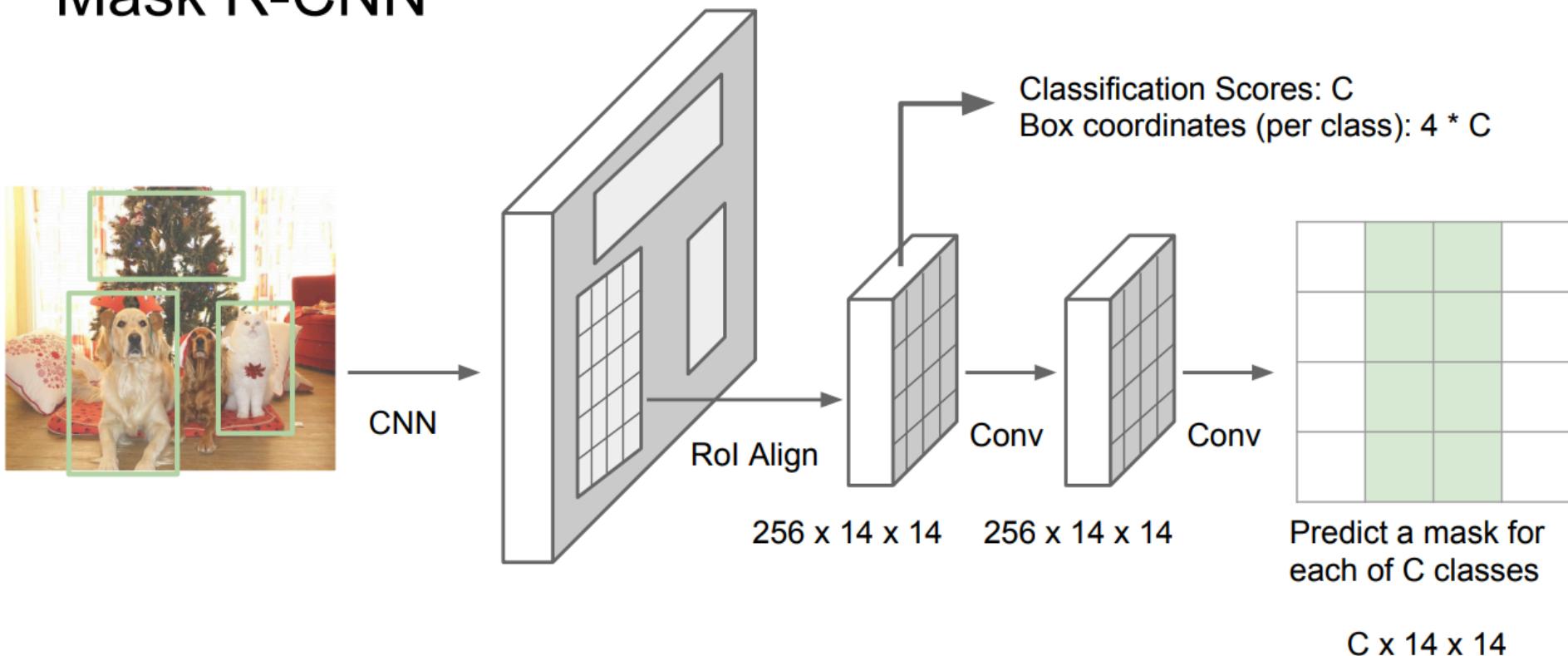
Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015  
Figure copyright 2015, Ross Girshick; reproduced with permission

# Fast er R-CNN:

Make CNN do proposals!

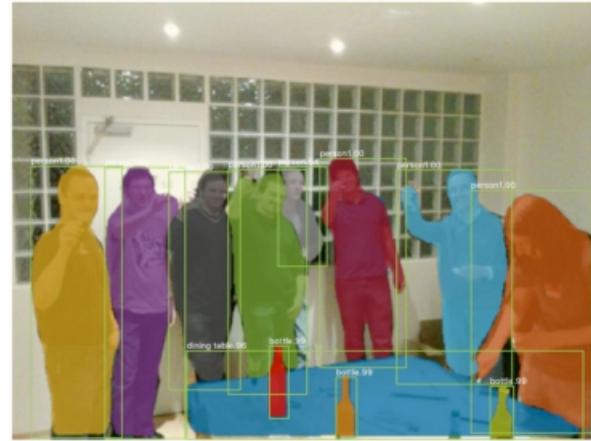
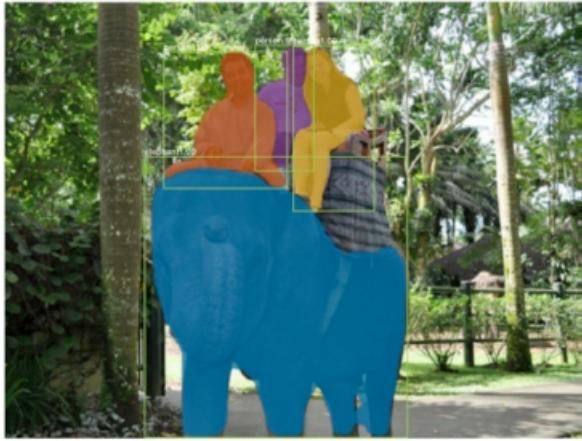


# Mask R-CNN



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

# Mask R-CNN: Very Good Results!



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

Figures copyright Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick, 2017.  
Reproduced with permission.

# Mask R-CNN

## Also does pose

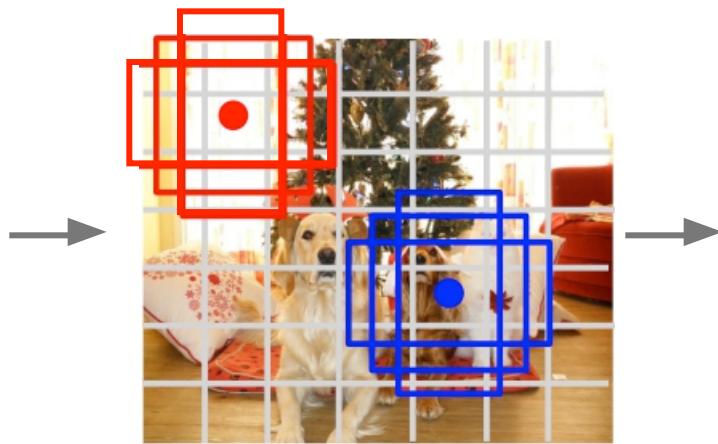


He et al, "Mask R-CNN", arXiv 2017  
Figures copyright Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick, 2017.  
Reproduced with permission.

# Detection without Proposals: YOLO / SSD



Input image  
 $3 \times H \times W$



Divide image into grid  
 $7 \times 7$

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)

Output:  
 $7 \times 7 \times (5 * B + C)$

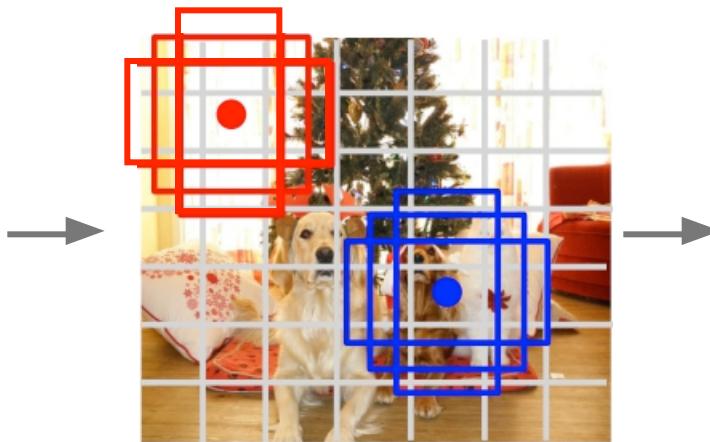
Redmon et al, "You Only Look Once:  
Unified, Real-Time Object Detection", CVPR 2016  
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016

# Detection without Proposals: YOLO / SSD

Go from input image to tensor of scores with one big convolutional network!



Input image  
 $3 \times H \times W$



Divide image into grid  
 $7 \times 7$

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)

Output:  
 $7 \times 7 \times (5 * B + C)$

Redmon et al, "You Only Look Once:  
Unified, Real-Time Object Detection", CVPR 2016  
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016

# Object Detection: Lots of variables ...

## Base Network

VGG16

ResNet-101

Inception V2

Inception V3

Inception

ResNet

MobileNet

## Object Detection architecture

Faster R-CNN

R-FCN

SSD

## Image Size # Region Proposals

...

## Takeaways

Faster R-CNN is slower but more accurate

SSD is much faster but not as accurate

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

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

# Object Detection: Impact of Deep Learning

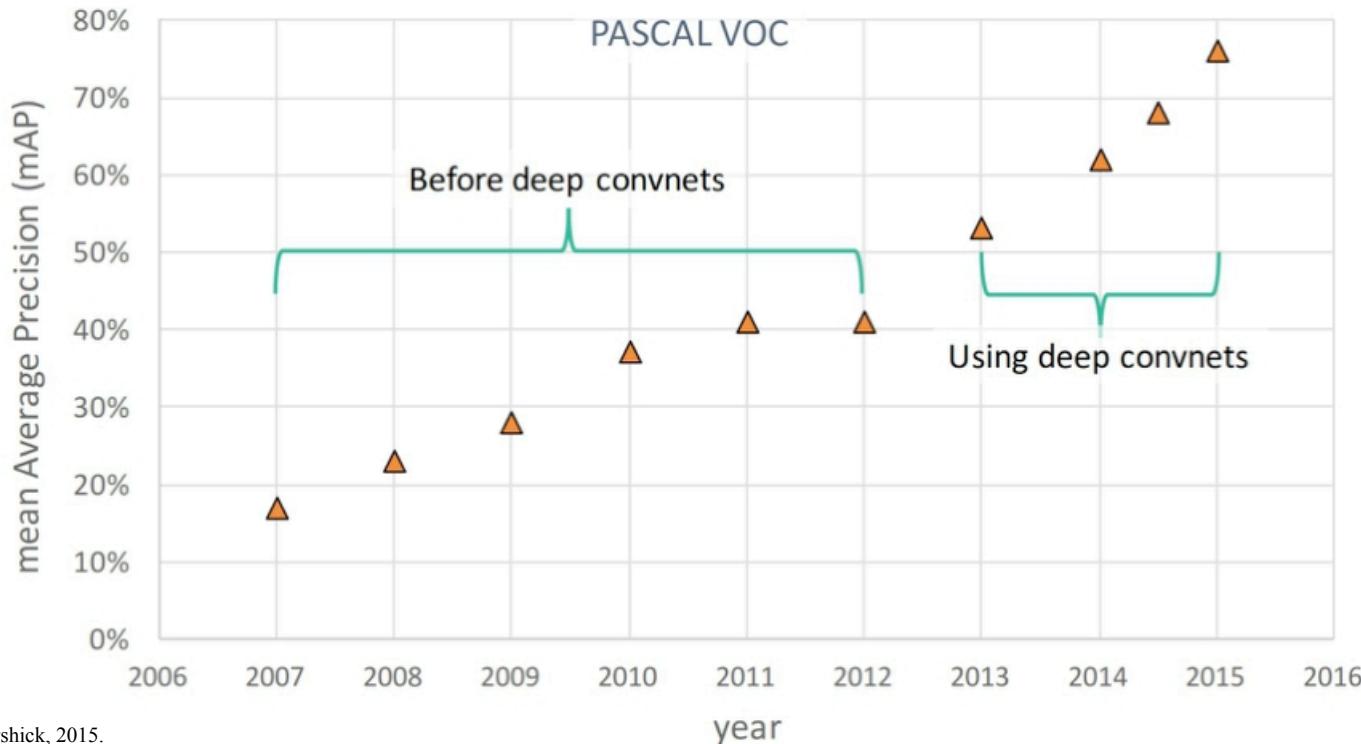


Figure copyright Ross Girshick, 2015.  
Reproduced with permission.

# Open Source Frameworks

Lots of good implementations on GitHub!

TensorFlow Detection API:

[https://github.com/tensorflow/models/tree/master/research/object\\_detection](https://github.com/tensorflow/models/tree/master/research/object_detection)

Faster RCNN, SSD, RFCN, Mask R-CNN

Caffe2 Detectron:

<https://github.com/facebookresearch/Detectron>

Mask R-CNN, RetinaNet, Faster R-CNN, RPN, Fast R-CNN, R-FCN

Finetune on your own dataset with pre-trained models