The Ancient Secrets of Computer Vision

Logistics:

- Homework 5 is due!
  - Probably still have some late days left
- Final Project Proposals were due on Tuesday
  - If you haven’t turned one in, please do
  - Kaggle competition on birds if you don’t have another idea
    - Link is on Google group, please do not share
- Start working on projects!

Previously On

the

Ancient Secrets of Computer Vision

Gradient explosion / vanishing

With very deep networks, the gradients flow through many layers of weights on their way back

With saturating activation functions like logistic or with small weights gradients can “vanish”

With non-saturating activations or large weights gradients can EXPLODE!

Learning doesn’t scale, what works at 2 levels doesn’t at 20

Batch normalization

One way to deal with gradient vanishing:

Normalize activations of filters spatially / over the mini-batch

During training, the distribution of network activations changes over time because the parameters (weights) change

Learning is more stable if this change (or *internal covariate shift*) as reduced

If output is 32 x 32 x 16 image, batch size of 64, normalize activations for each filter across all images in batch:

I.e. calculate 16 means and variances

Subtract mean, divide by variance both across spatial dimensions and images in the batch

https://arxiv.org/pdf/1502.03167.pdf

Batch normalization

Other benefits:

Output is normalized before activation, mean 0 var 1 means it’s in the “good” domain of most activation functions

Each image is seen relative to others in a batch, introduces a form of regularization because we don’t ever “see” same image twice

Stabilizes training so much larger learning rates can be used

https://arxiv.org/pdf/1502.03167.pdf

GoogleNetv2 or Inception Net?

Or something, GoogleNet + batch norm

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	double #3×3 reduce	double #3×3	Pool +proj
convolution*	7×7/2	112×112×64	1						
max pool	3×3/2	56×56×64	0						
convolution	3×3/1	56×56×192	1		64	192			
max pool	3×3/2	28×28×192	0						
inception (3a)		28×28×256	3	64	64	64	64	96	avg + 32
inception (3b)		28×28×320	3	64	64	96	64	96	avg + 64
inception (3c)	stride 2	28×28×576	3	0	128	160	64	96	max + pass through
inception (4a)		14×14×576	3	224	64	96	96	128	avg + 128
inception (4b)		14×14×576	3	192	96	128	96	128	avg + 128
inception (4c)		14×14×576	3	160	128	160	128	160	avg + 128
inception (4d)		14×14×576	3	96	128	192	160	192	avg + 128
inception (4e)	stride 2	14×14×1024	3	0	128	192	192	256	max + pass through
inception (5a)		7×7×1024	3	352	192	320	160	224	avg + 128
inception (5b)		7×7×1024	3	352	192	320	192	224	max + 128
avg pool	7×7/1	1×1×1024	0						

Figure 5: Inception architecture

https://arxiv.org/pdf/1502.03167.pdf

Residual connections

Normally, output of two layers is: f(w*f(vx))

Residual connections: f(w*f(vx) + x)

Learning how to modify x, add some transformed amount

Gives delta another path, less vanishing gradient

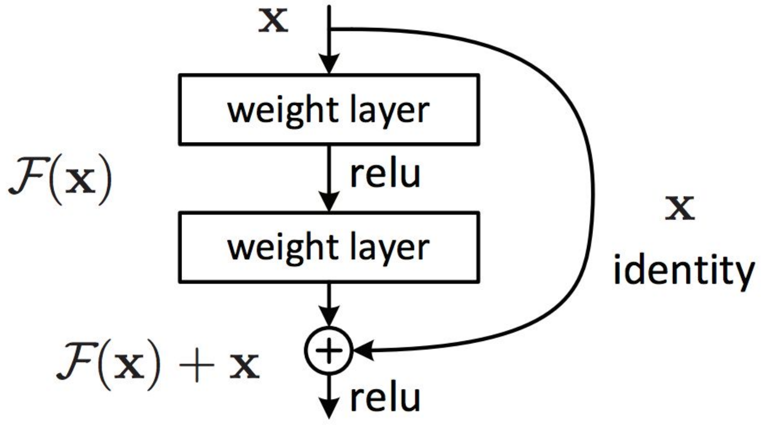
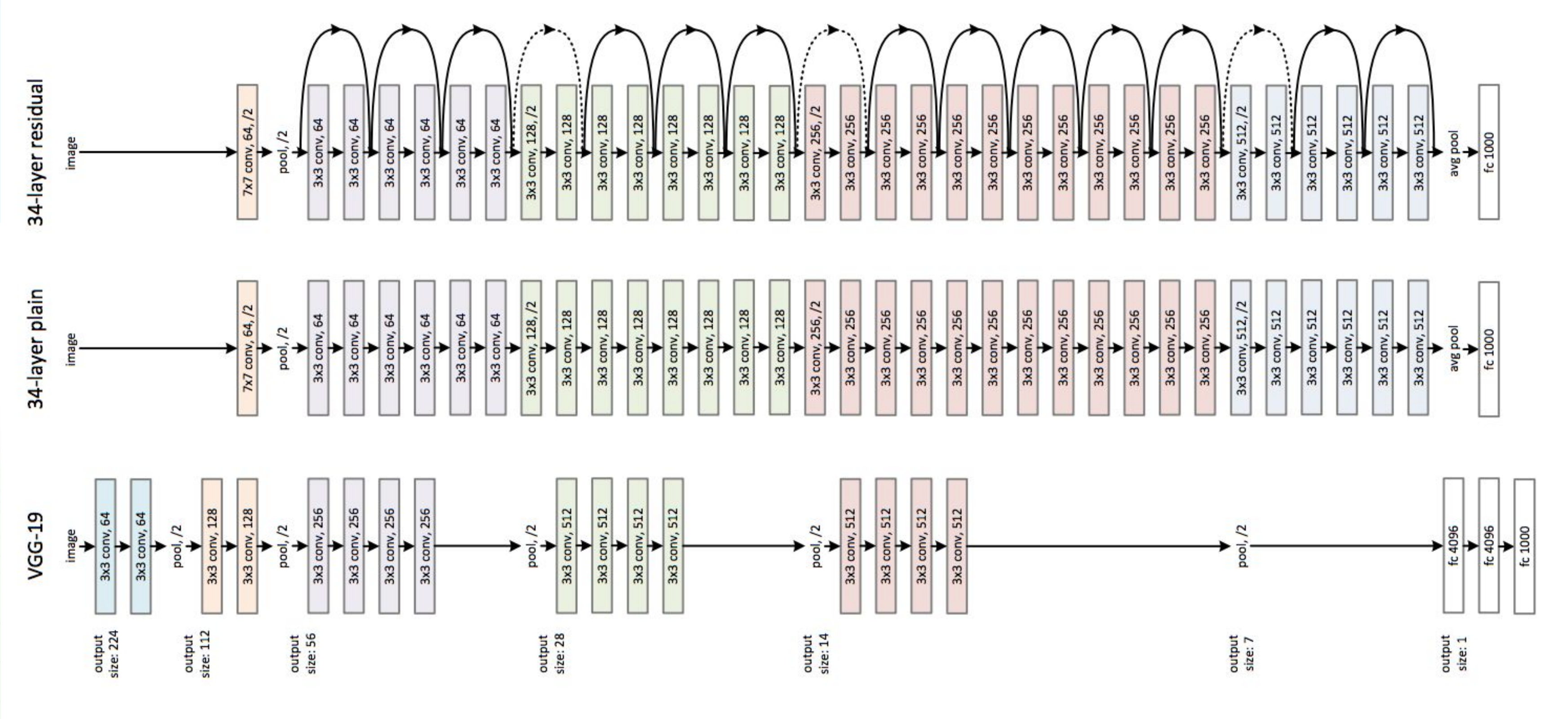


Figure 2. Residual learning: a building block.

ResNet



34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000
34-layer plain
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000
VGG-19
image
3x3 conv, 64
3x3 conv, 64
pool, /2
3x3 conv, 128
3x3 conv, 128
pool, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
fc 4096
fc 4096
fc 1000
output size: 224
output size: 112
output size: 56
output size: 28
output size: 14
output size: 7
output size: 1

ResNet

3x3 conv blocks or 3x3 and 1x1 conv blocks

Residual connections

VERY deep, 100+ layers

Grouped convolutions

Most filters look at every channel in input

- Very expensive
- Maybe not needed? Might only pull info from a few of them

Grouped convolutions:

- Split up input feature map intro groups
- Run convs on groups independently
- Recombine

Grouped convolutions

Grouped convolutions:

- Split up input feature map intro groups
- Run convs on groups independently
- Recombine

E.g. 3x3 conv layer 32 x 32 x 256 input, 128 filters, 32 groups:

- Split input into 32 different feature maps
- Each is 32 x 32 x 8
- Run 4 filters, 3x3x8 on each group
- Merge 4*32 channels back together, get 32 x 32 x 128 output

Input, output stays same dimensions, less computation

ResNeXt

Replace 3x3 blocks with larger grouped convs

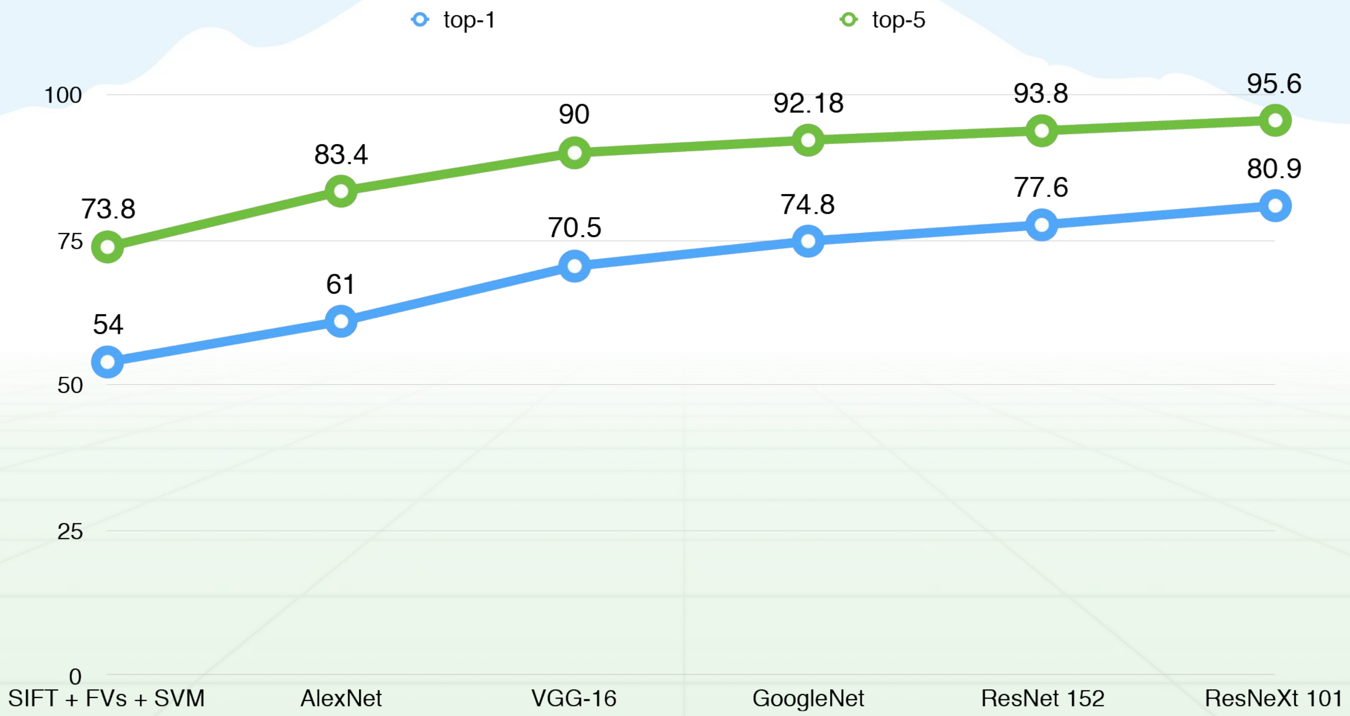
“Larger” network but same computational complexity

stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C{=}32 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C{=}32 \\ 1\times1, 512 \end{bmatrix} \times 4$
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C{=}32 \\ 1\times1, 1024 \end{bmatrix} \times 6$
conv5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, C{=}32 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		**25.5**$\times10^6$	**25.0**$\times10^6$
FLOPs		**4.1**$\times10^9$	**4.2**$\times10^9$

https://arxiv.org/pdf/1611.05431.pdf

What's NeXt?

Starting to saturate ImageNet, fighting over 1-2%



What's NeXt?

Starting to saturate ImageNet, fighting over 1-2%

But now vision really *works*, other tasks

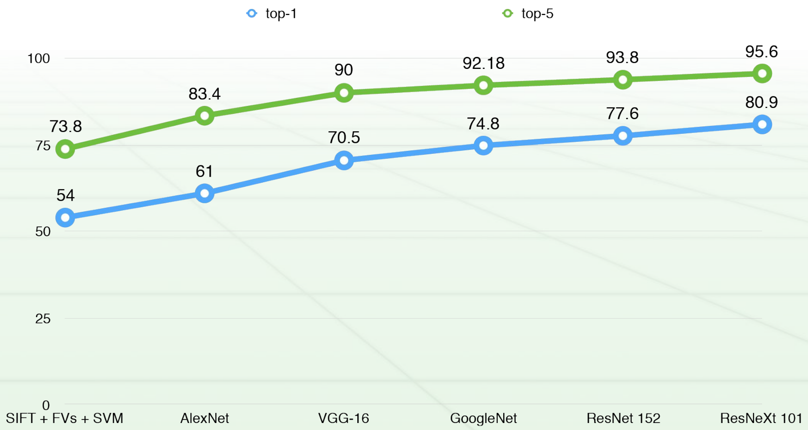
Segmentation

Object detection

Captioning

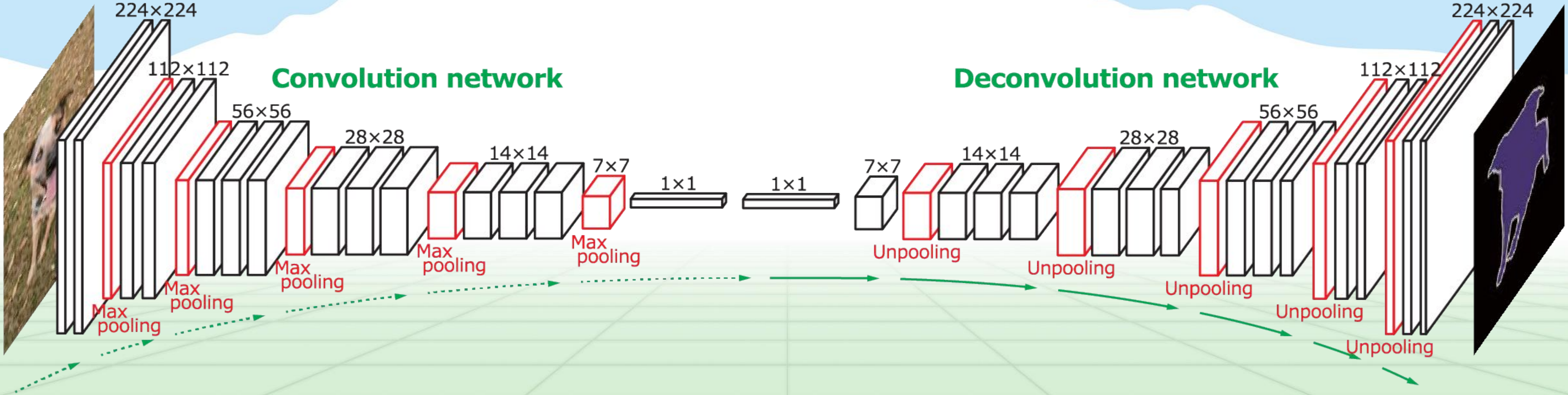
…

The rest of the class



Semantic Segmentation

Semantic Segmentation

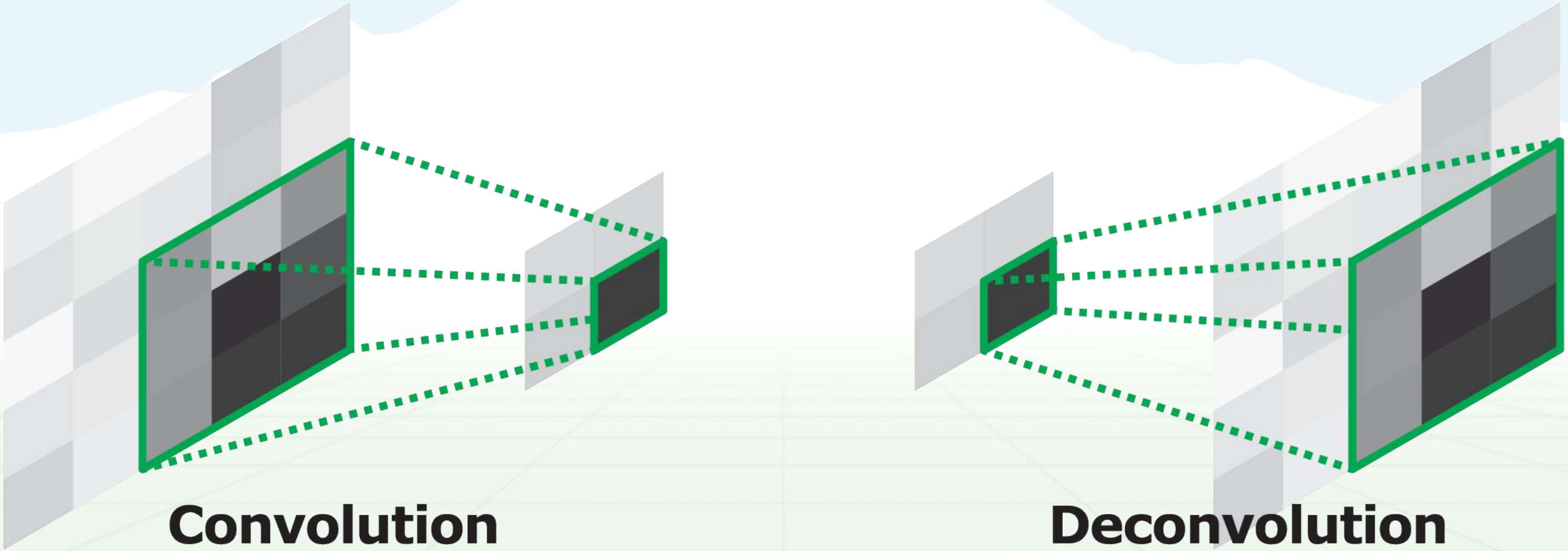


https://arxiv.org/pdf/1505.04366.pdf

Chapter Sixteen

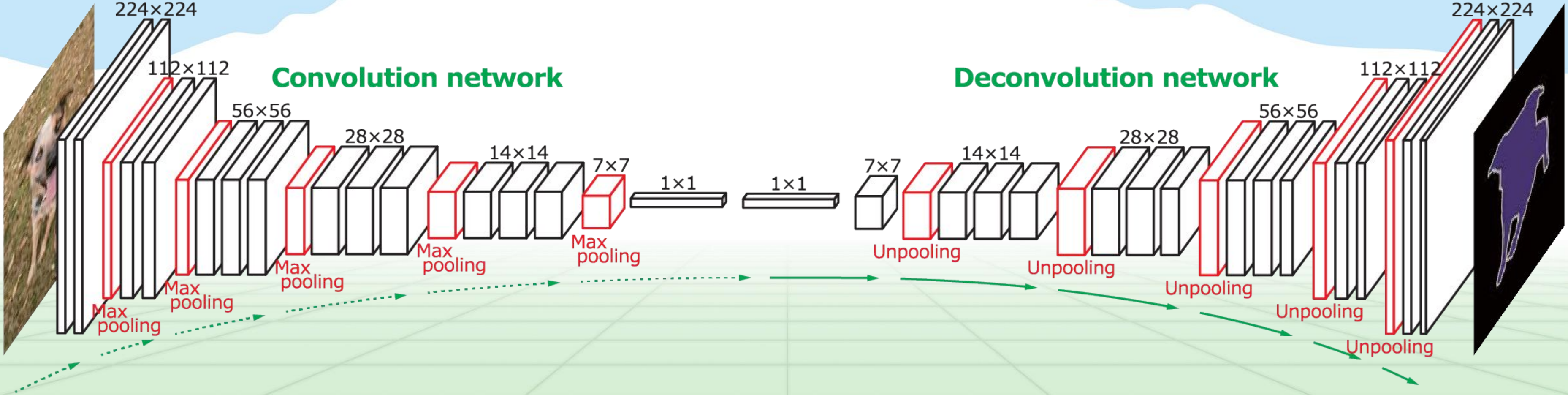
Object Detection

Semantic Segmentation



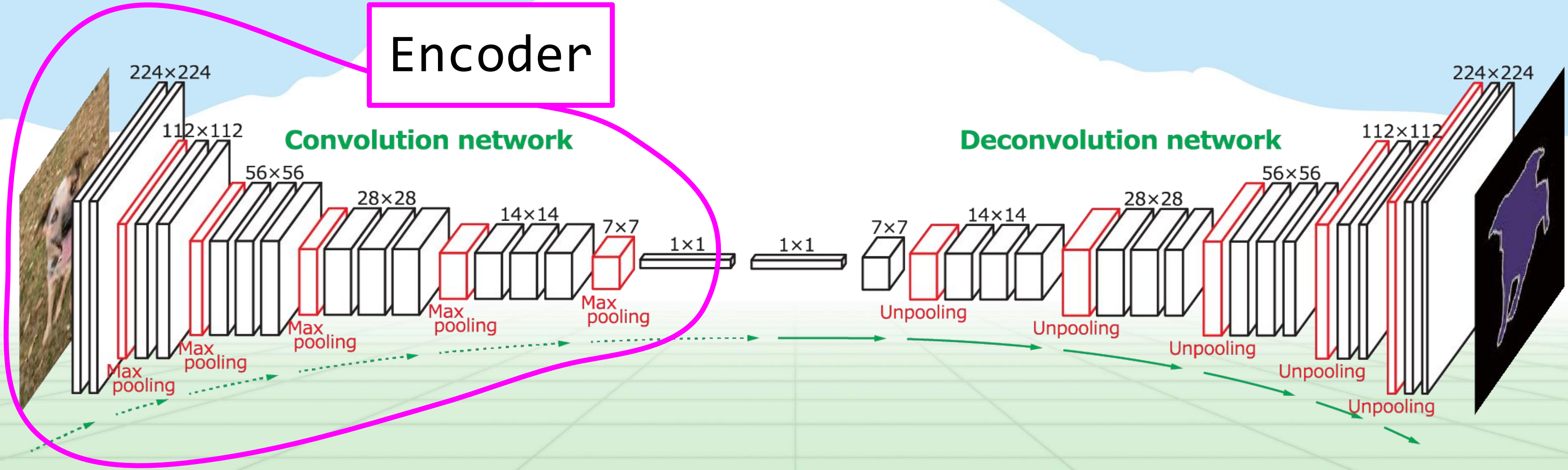
https://arxiv.org/pdf/1505.04366.pdf

Semantic Segmentation



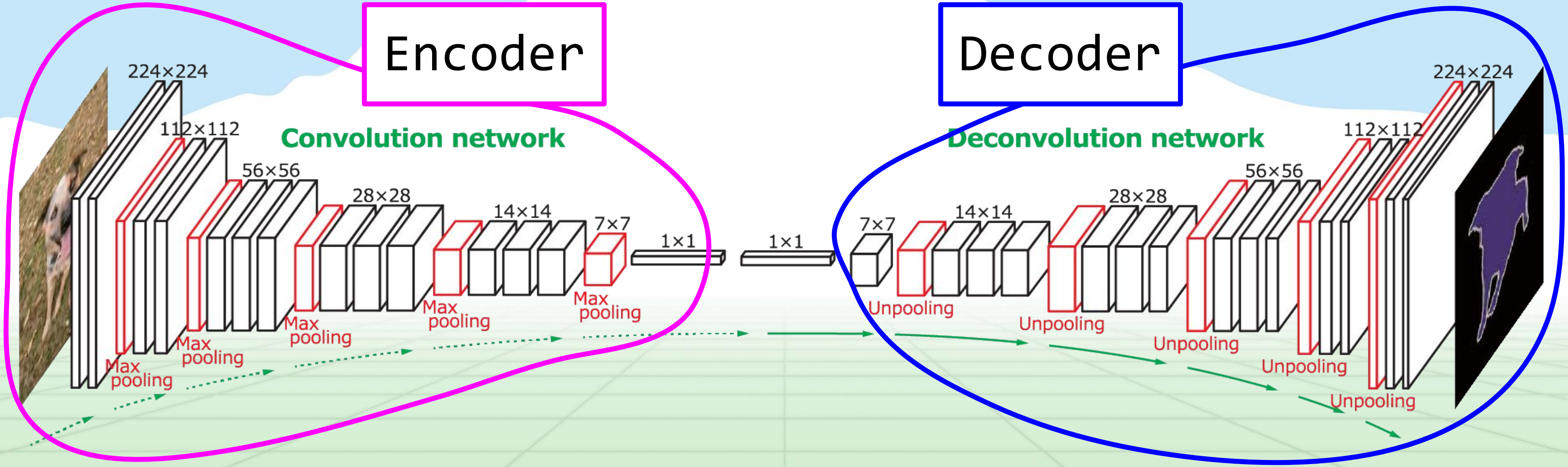
https://arxiv.org/pdf/1505.04366.pdf

Semantic Segmentation



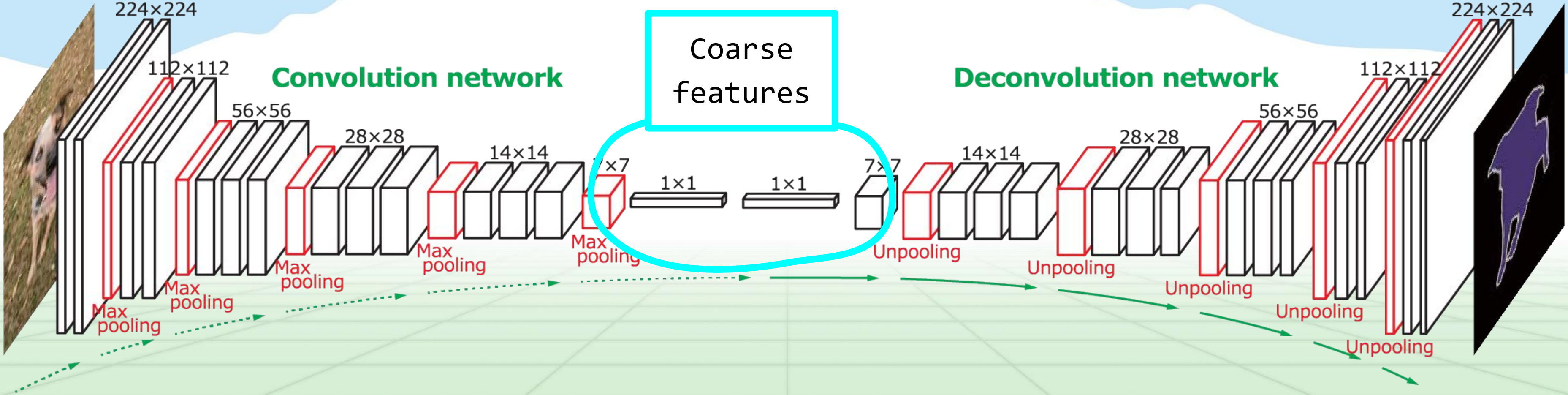
https://arxiv.org/pdf/1505.04366.pdf

Semantic Segmentation



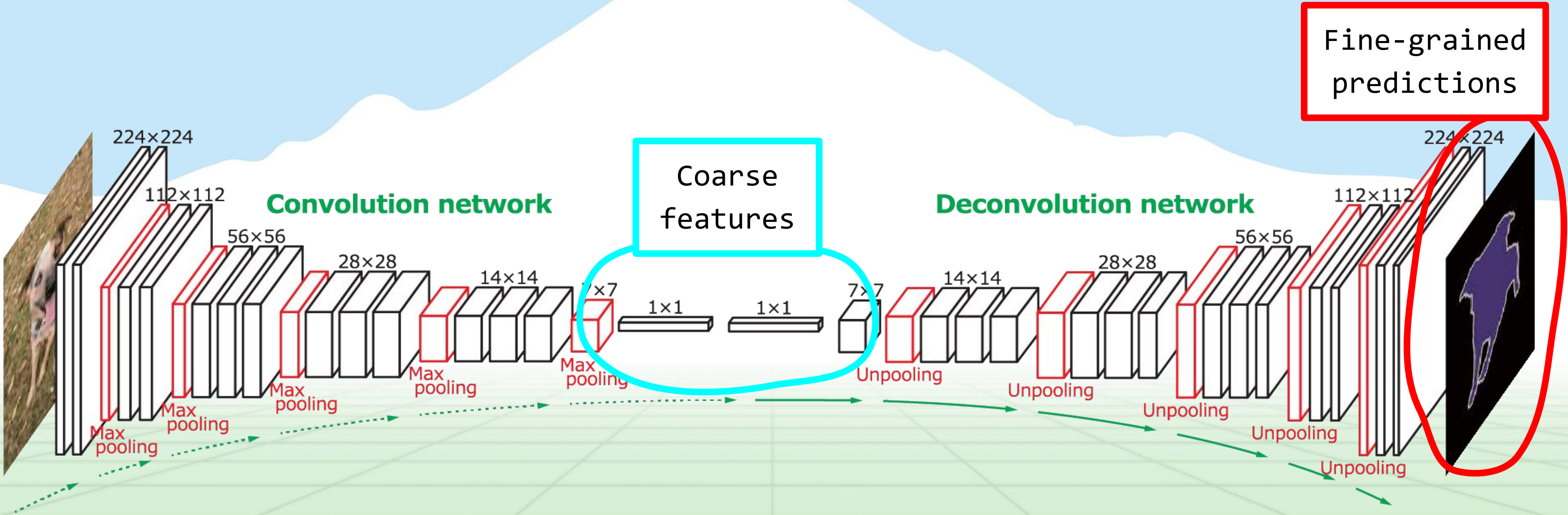
https://arxiv.org/pdf/1505.04366.pdf

Semantic Segmentation



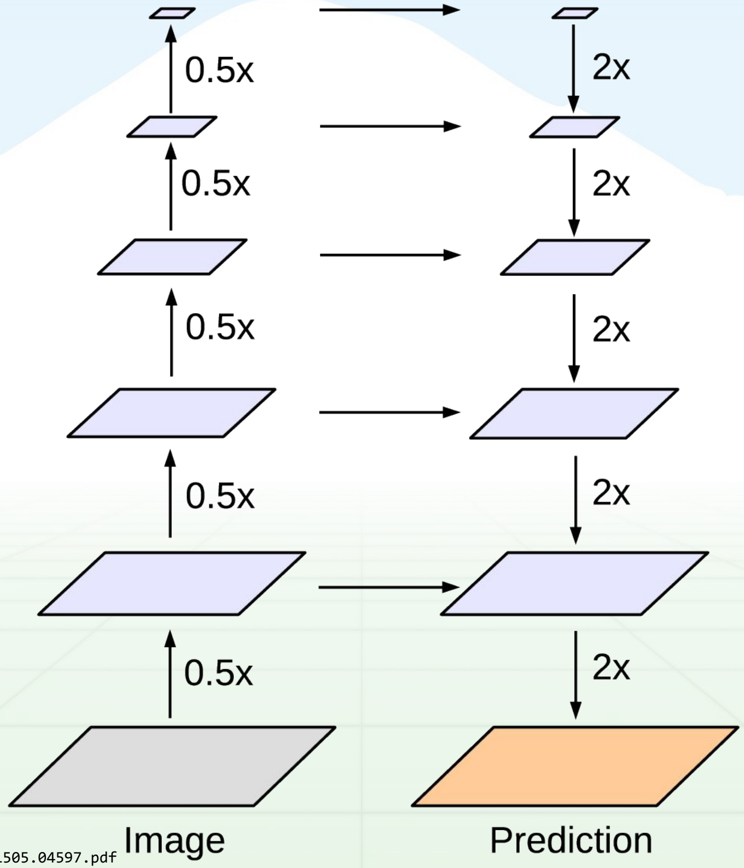
https://arxiv.org/pdf/1505.04366.pdf

Semantic Segmentation



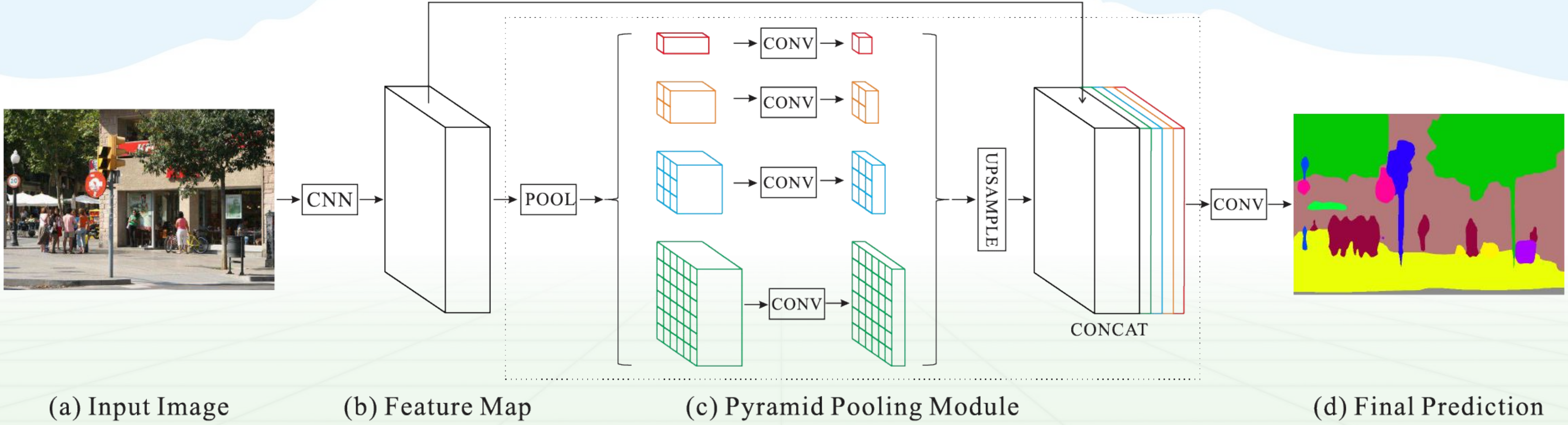
https://arxiv.org/pdf/1505.04366.pdf

U-net/Segnet



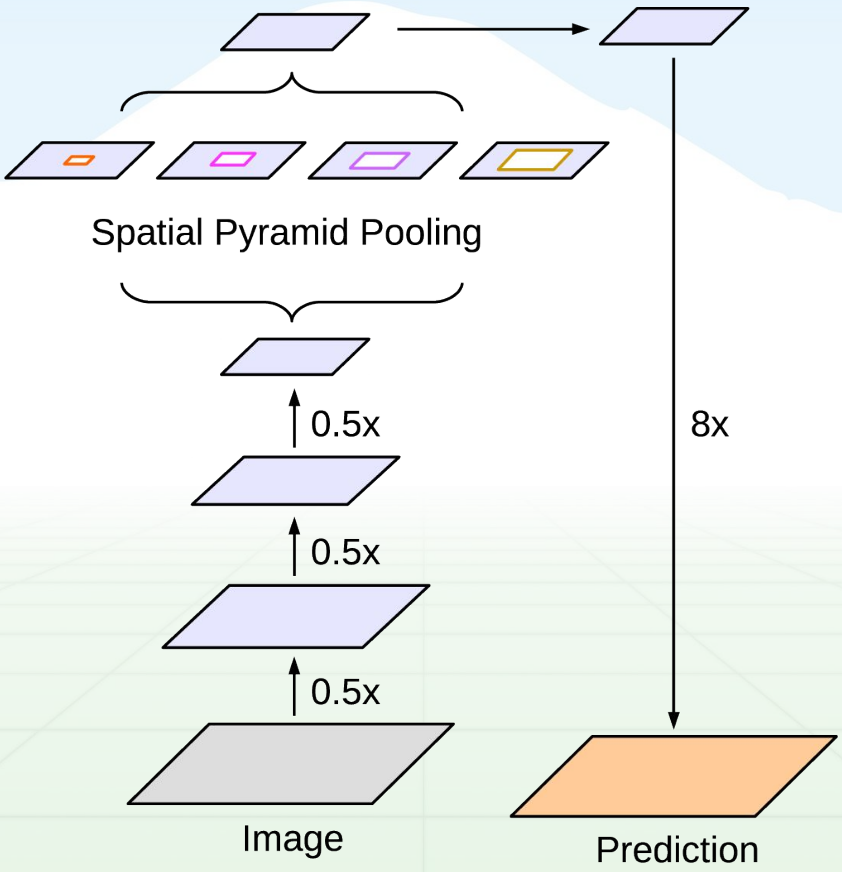
https://arxiv.org/pdf/1511.00561.pdf, https://arxiv.org/pdf/1505.04597.pdf

Spatial pyramid pooling



https://arxiv.org/pdf/1612.01105.pdf

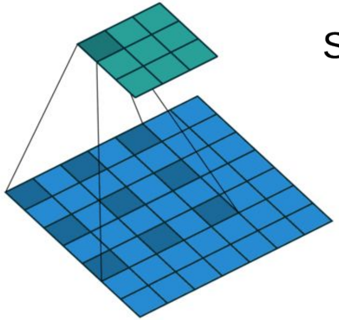
Spatial pyramid pooling

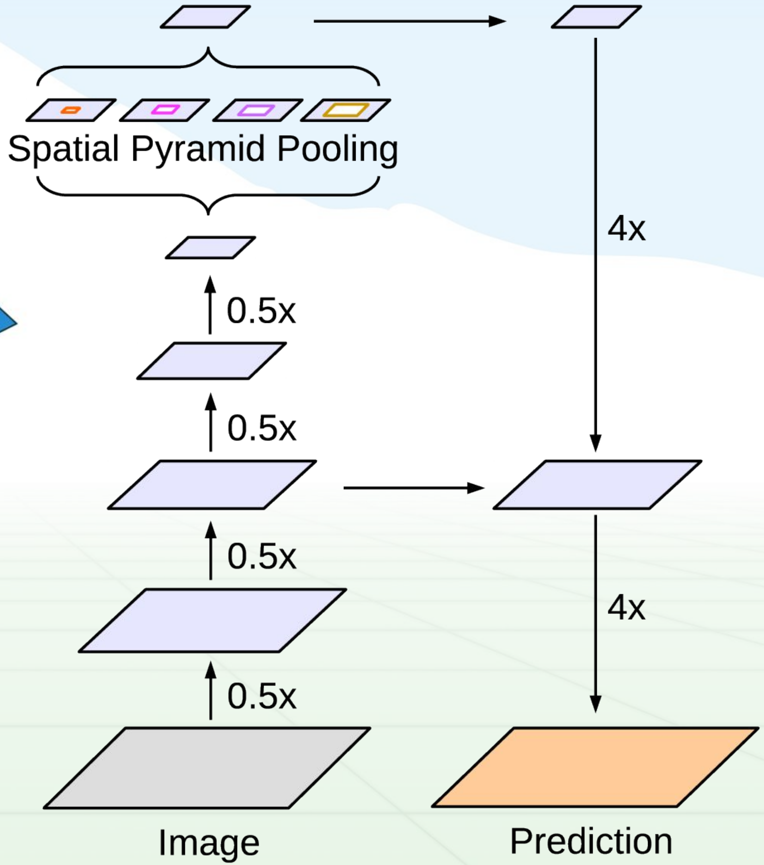


DeepLabv3+

Atrous convolutions

Spaced inputs



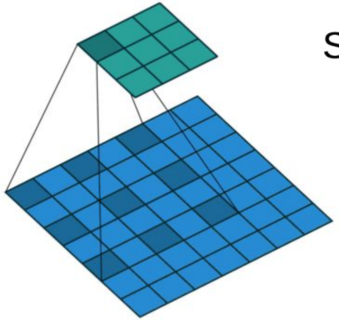


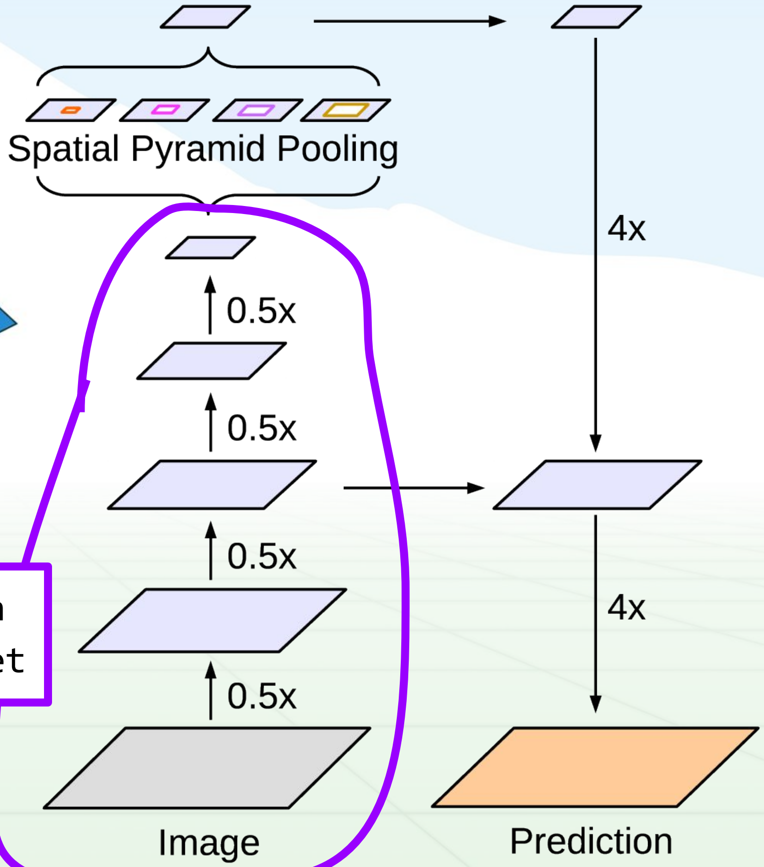
https://arxiv.org/pdf/1802.02611.pdf

DeepLabv3+

Atrous convolutions

Spaced inputs





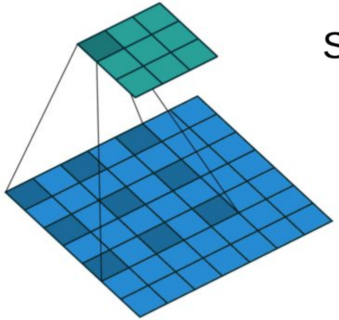
Pre-train on ImageNet

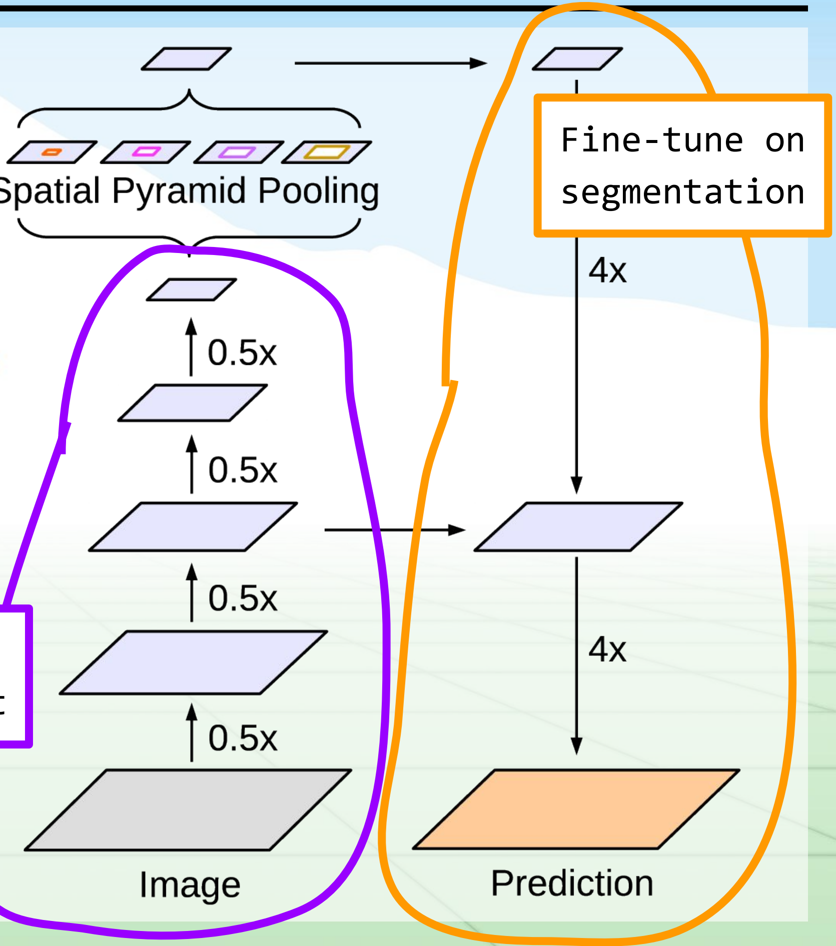
https://arxiv.org/pdf/1802.02611.pdf

DeepLabv3+

Atrous convolutions

Spaced inputs

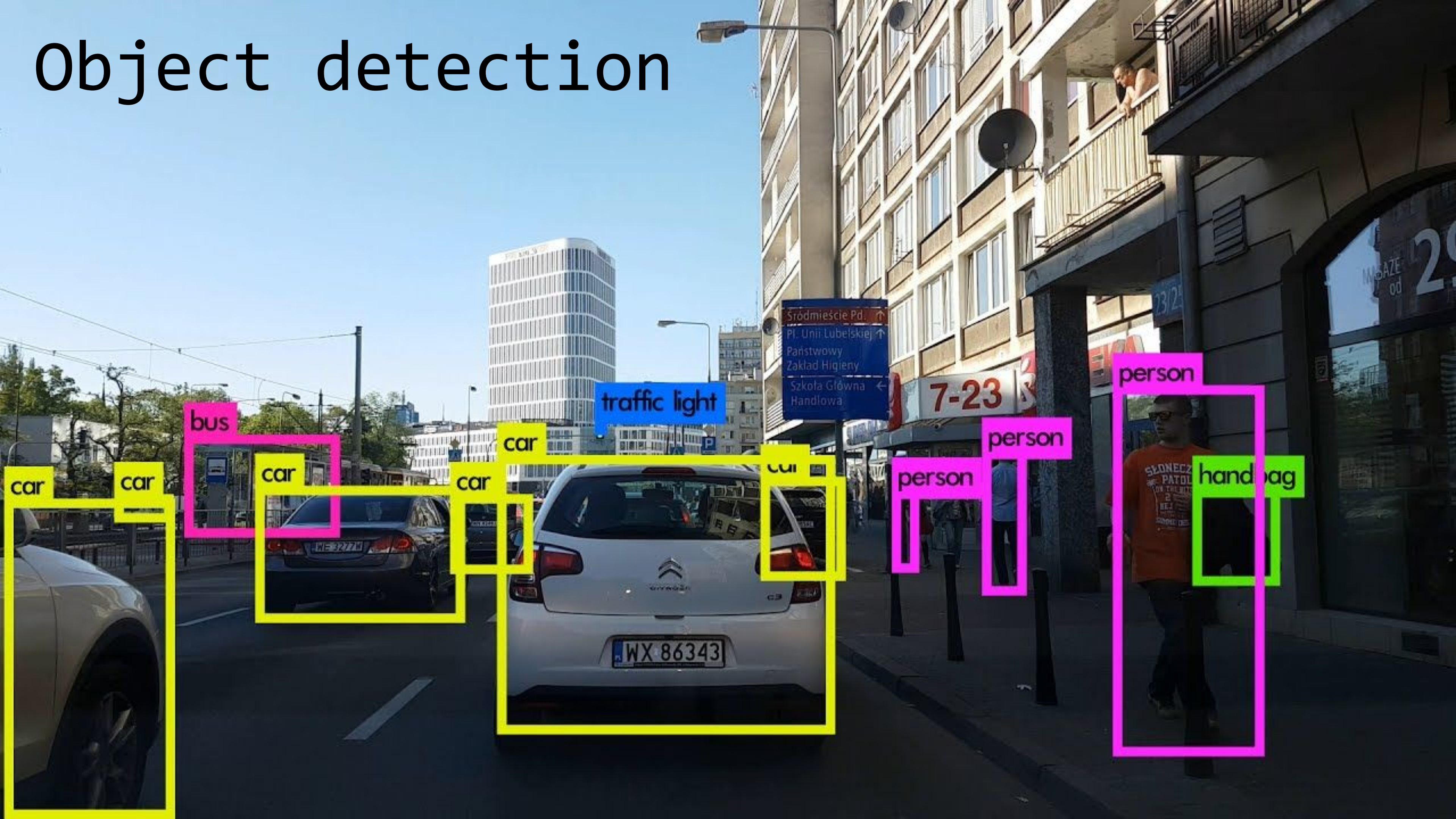




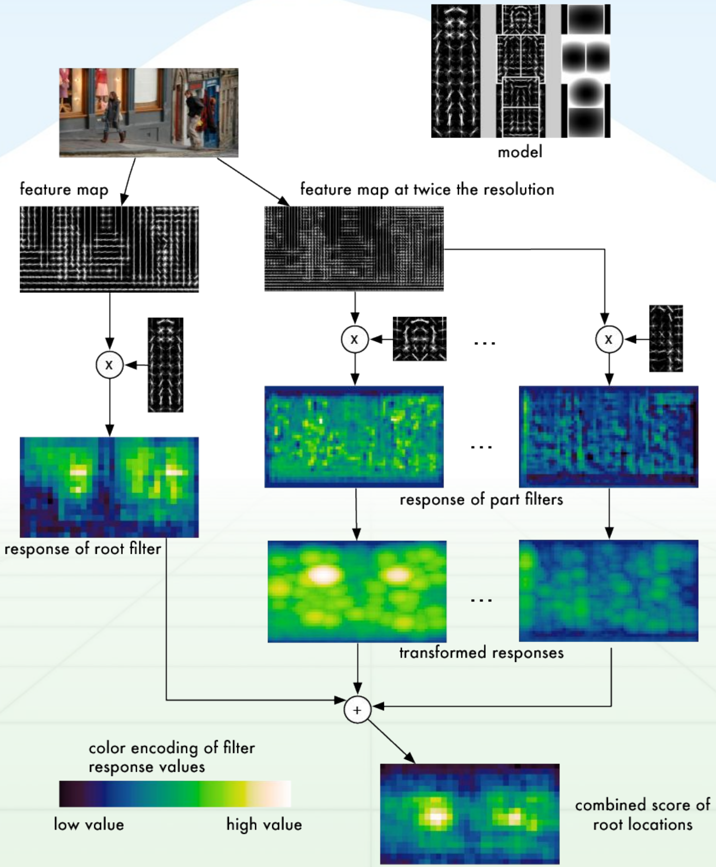
https://arxiv.org/pdf/1802.02611.pdf


C. Lothar Lenz
www.pferdefotoarchiv.de

Object detection



Deformable parts models



Scoring object detection

Multiple classes, multiple objects per images

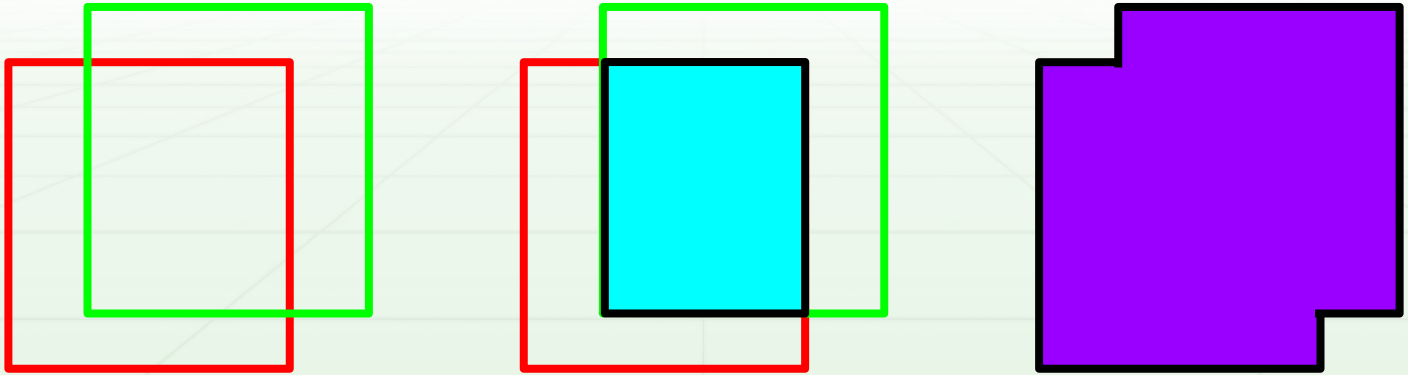
Can't just use accuracy

"Correct" bounding box:

    Intersection / Union > 0.5

Intersection: Ground truth ∩ prediction

Union: Ground truth ∪ prediction



Scoring object detection

“Correct” bounding box:

Intersection / Union > 0.5

Recall:

Correct bounding boxes / total ground-truth boxes

Precision:

Correct bounding boxes / total predicted boxes

Only the most confident predictions:	High precision, low recall
All the predictions:	Low precision, high recall

Scoring object detection

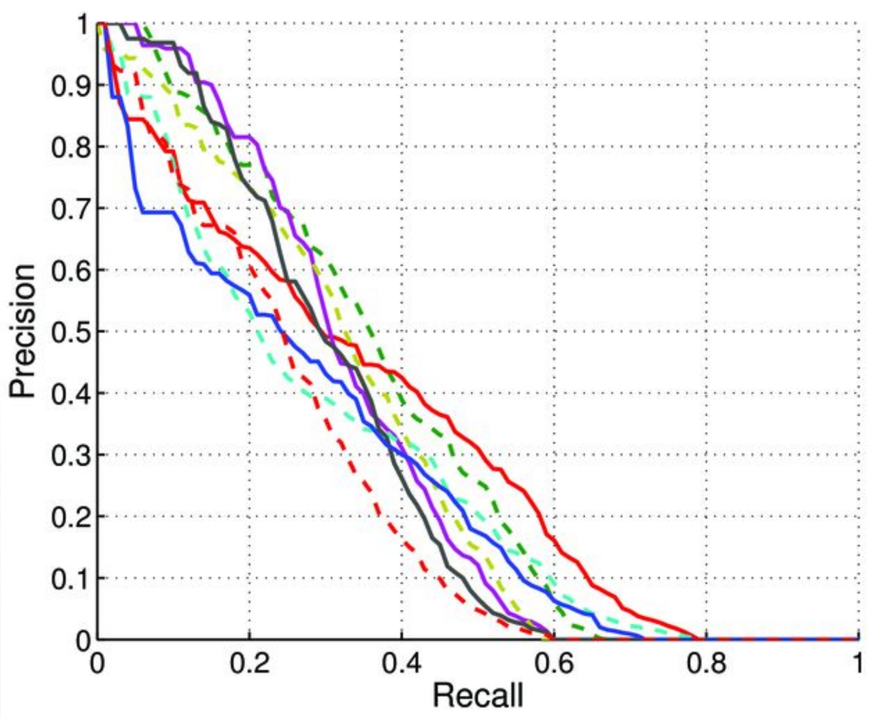
Precision-Recall curve: vary threshold, plot precision and recall

Average precision:

- Area under PR curve
- Only for a single class

Take mean of AP across classes:

- Mean AP (mAP)
- Standard detection metric
- Sometimes at particular IOU
  - I.e. mAP@.5 or mAP@.75



Pascal VOC

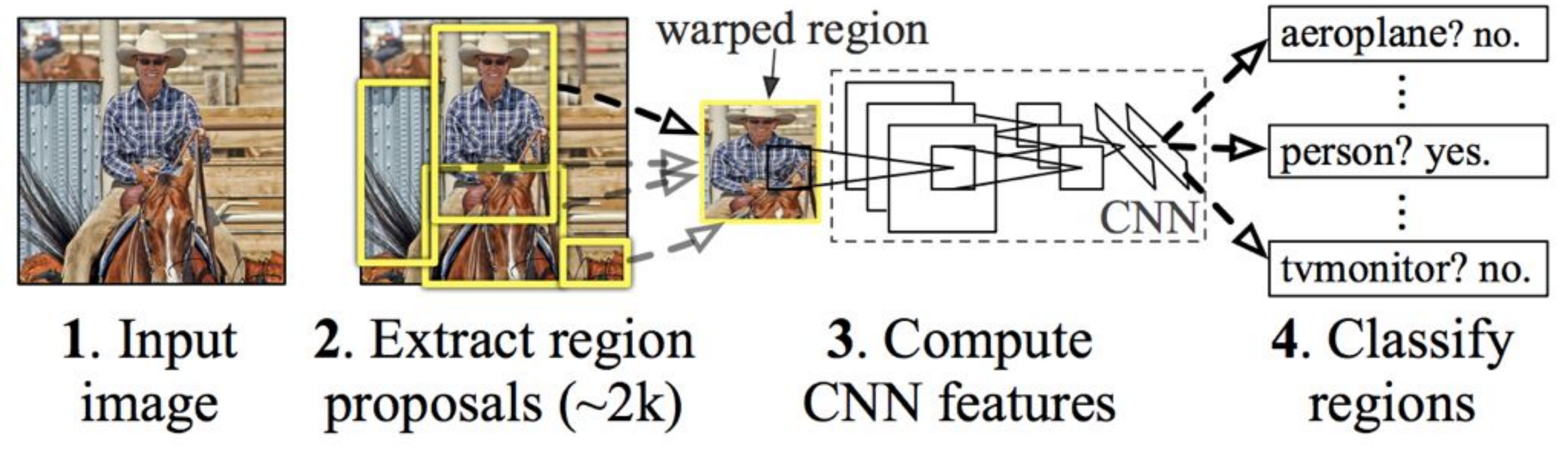
One of the first large detection datasets:

- 20 classes
- 11,530 training images
- 27,450 annotated objects

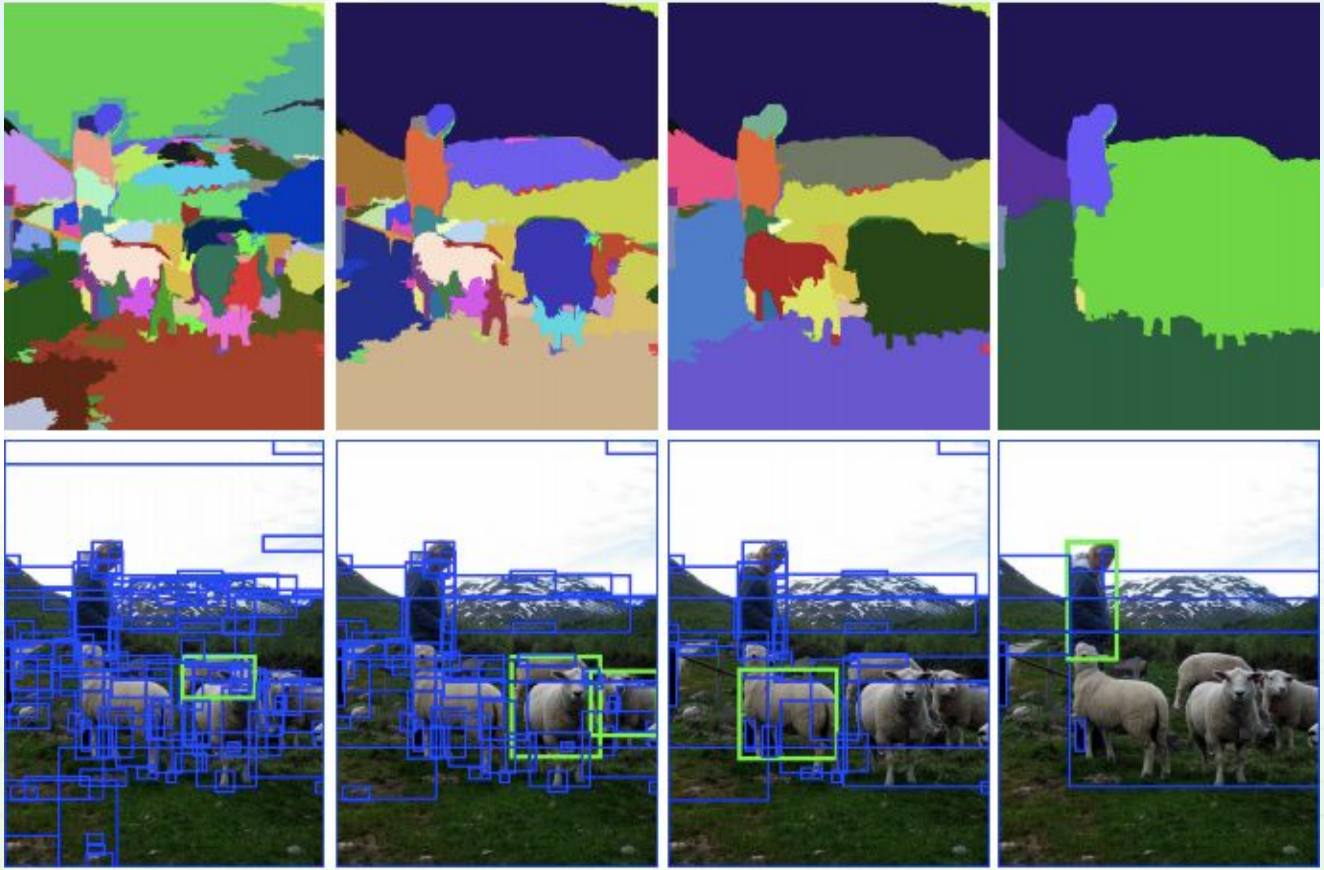
DPM: 33.6% mAP

DPM is pre-neural network, how do we use CNNs for detection?

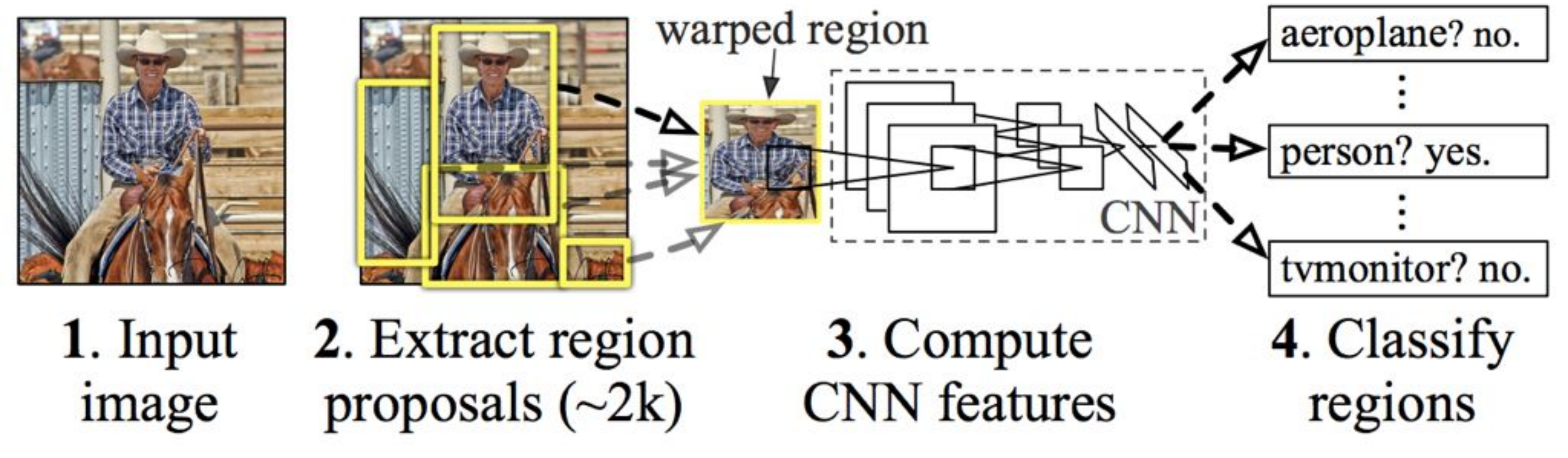
R-CNN: Regions with CNN features



Selective search: fewer proposals



R-CNN: Regions with CNN features



Lots of post processing, 20 sec/im

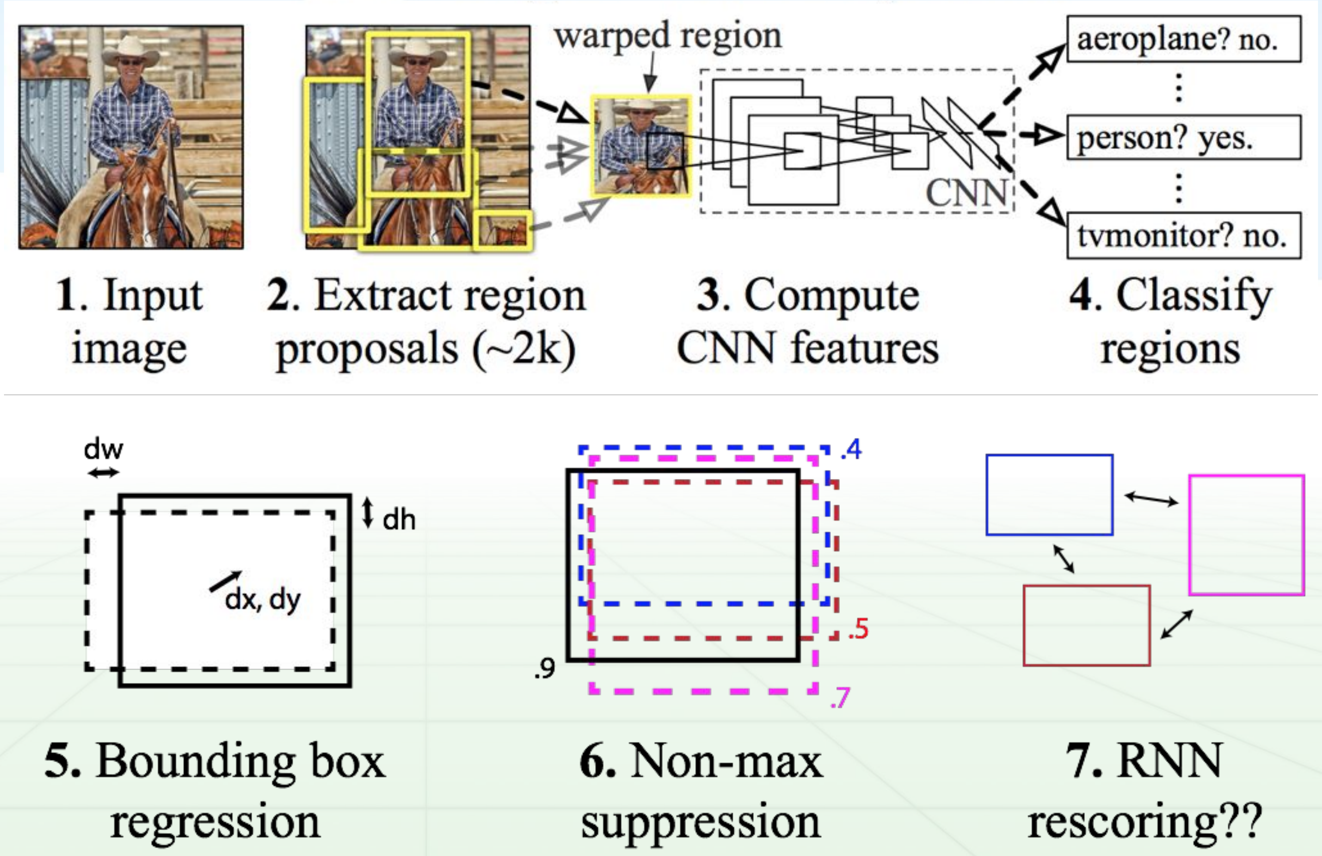
Pascal VOC:

AlexNet

53.3% mAP

VGG-16

62.4% mAP



YOLO