

# Segmentation

Vladislav Goncharenko  
Spring 2019  
MIPT, Moscow

# Outline

- Segmentation task
- Simple solutions
- Upsampling methods
  - Unpooling
  - Transposed convolution
- FCN
- DeconvNet
- SegNet
- U-Net
- Mask R-CNN

# Semantic Segmentation



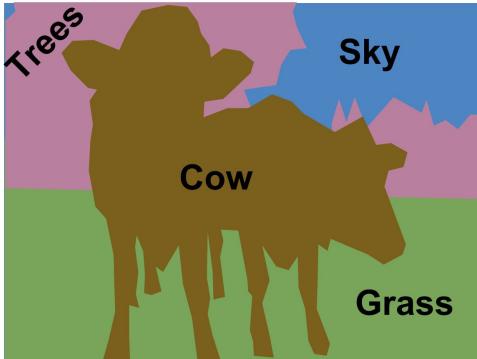
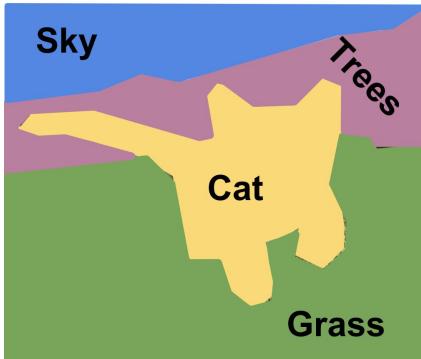
Input image ( $C \times H \times W$ )

Output

mask of classes ( $1 \times H \times W$ )

Assumption

just class label for each pixel



# Instance Segmentation



**Input** image ( $C \times H \times W$ )

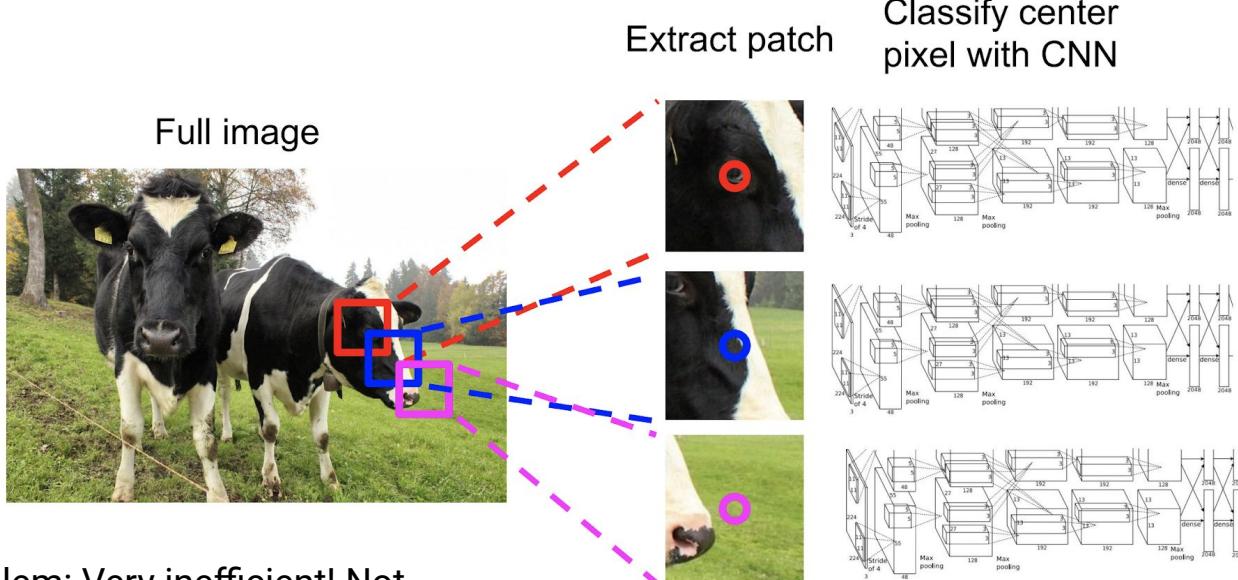
**Output**

mask of class instances (detection + 1  $\times H \times W$ )

**Assumption**

separated masks for each distinct object

# Naive solution

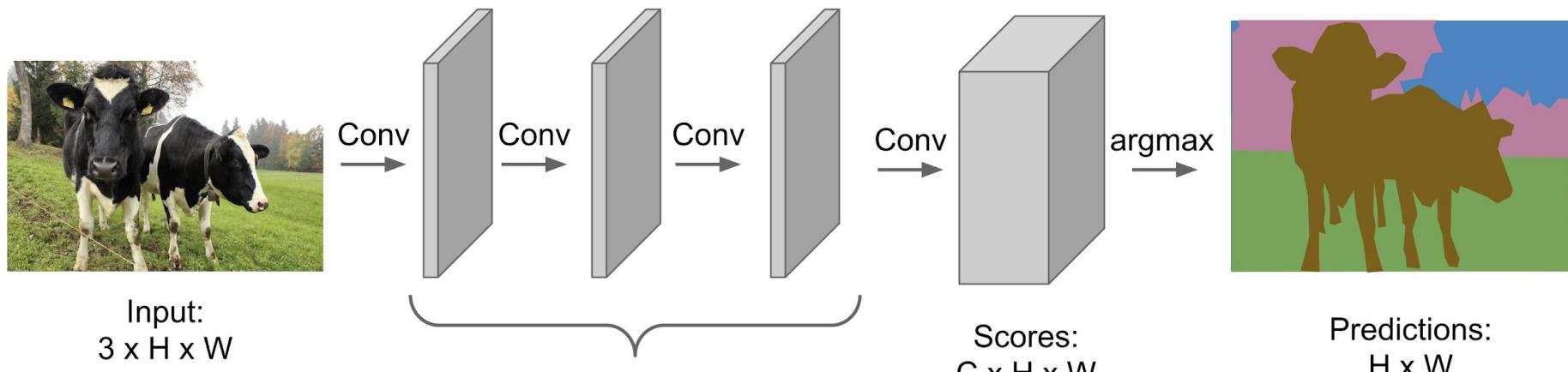


Problem: Very inefficient! Not reusing shared features between overlapping patches (same as R-CNN)

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

# Fully Convolutional

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



Problem: convolutions at original image resolution will be very expensive  
(remember VGG architecture)

Convolutions:  
 $D \times H \times W$

Scores:  
 $C \times H \times W$

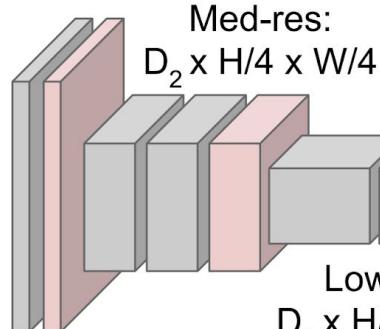
Predictions:  
 $H \times W$

# Downsampling and Upsampling

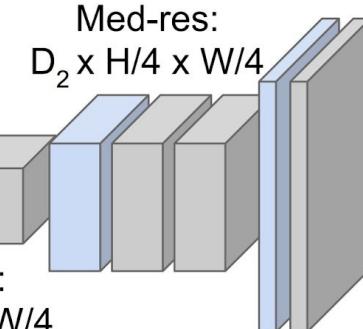
Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!



Input:  
 $3 \times H \times W$



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



Low-res:  
 $D_3 \times H/4 \times W/4$

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



Predictions:  
 $H \times W$

# Simple unpooling

**Nearest Neighbor**

1	2
3	4



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

Input: 2 x 2

Output: 4 x 4

**“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

# 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

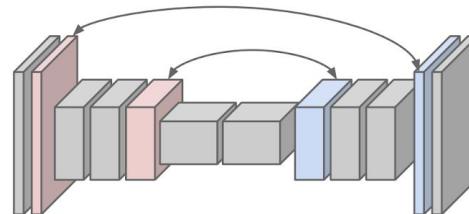
1	2
3	4

Input: 2 x 2

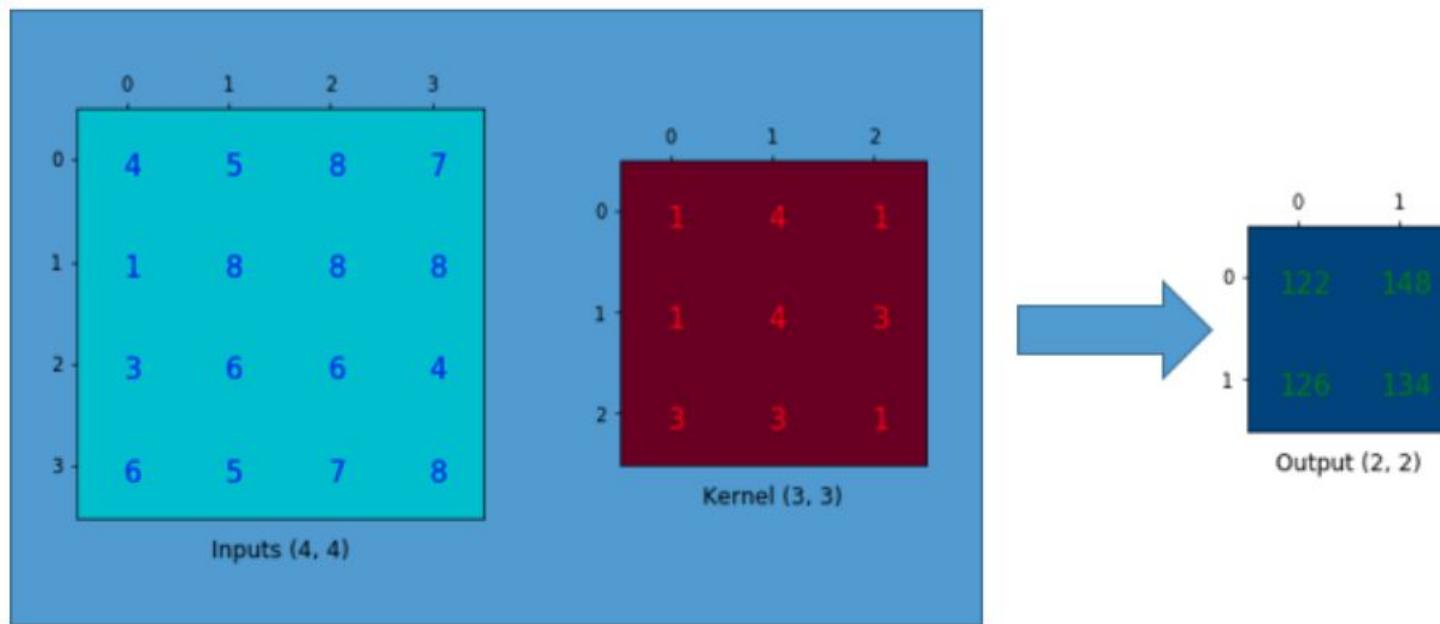
0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

Output: 4 x 4

Corresponding pairs of  
downsampling and  
upsampling layers

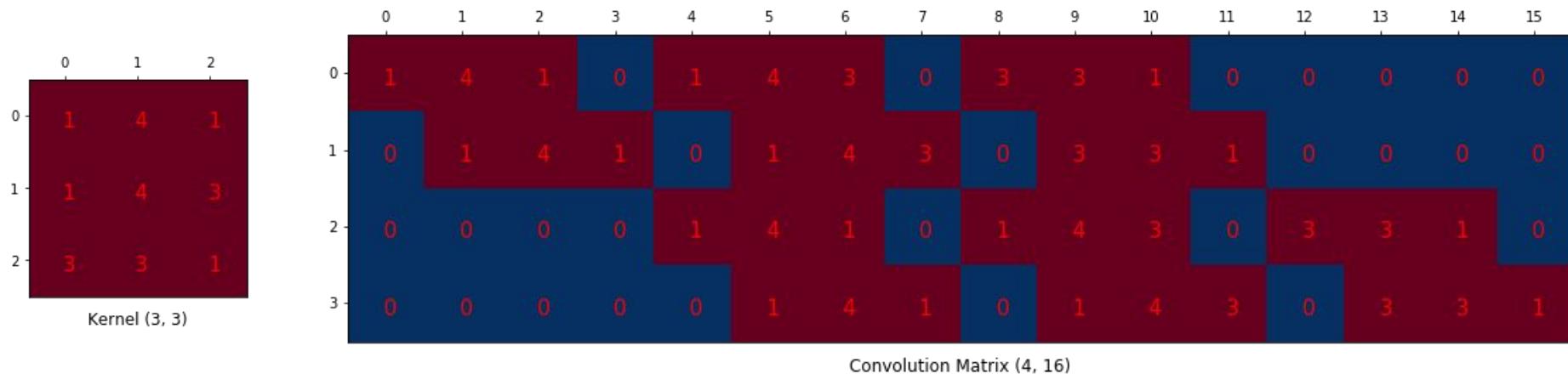
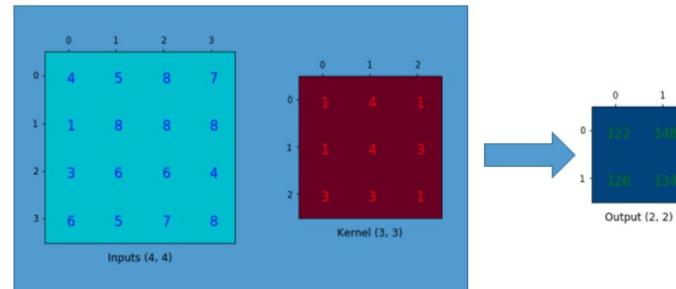


# Regular convolution

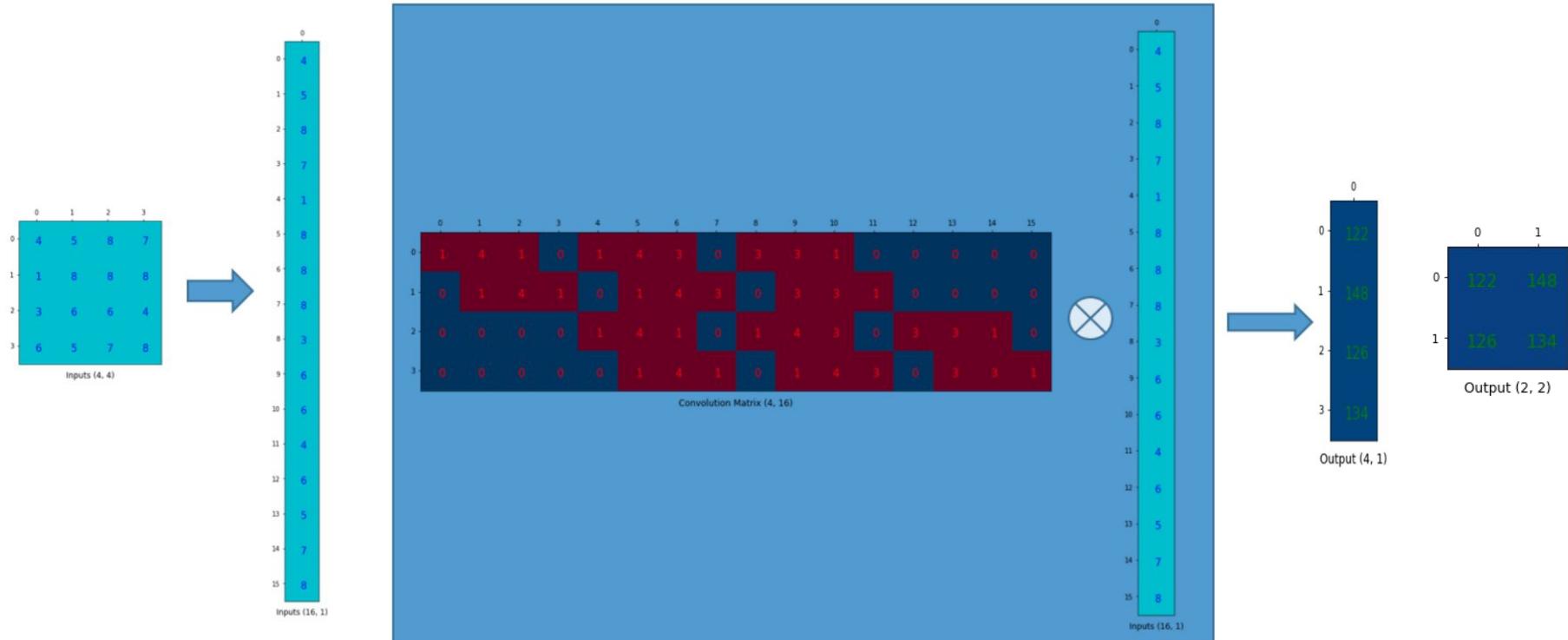


[Images credentials](#)

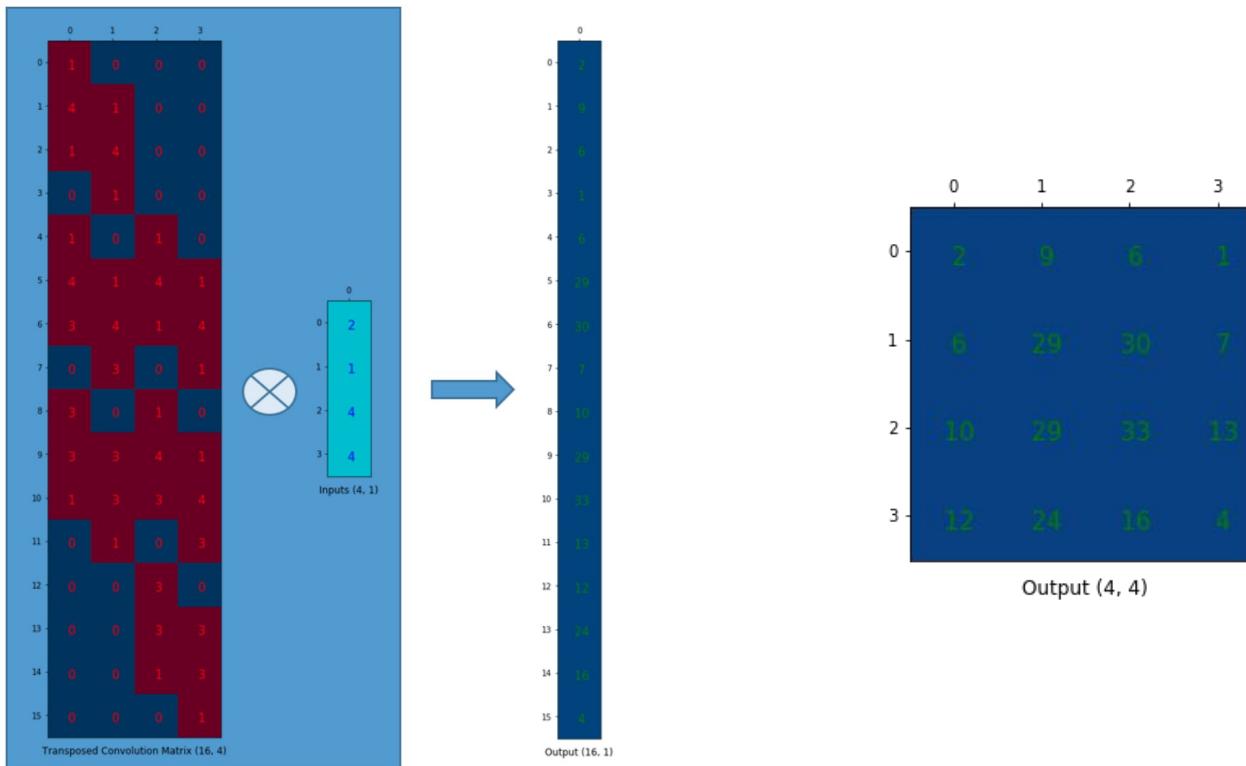
# Regular convolution: matrix form



# Regular convolution: matrix form



# Transposed convolution matrix



# Convolution as Matrix Multiplication (1D)

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)

# Terms

Some researchers denote Transposed convolution with term “Deconvolution”. However:

**deconvolution** is an [algorithm](#)-based process used to reverse the effects of [convolution](#) on recorded data [\[Wikipedia\]](#)

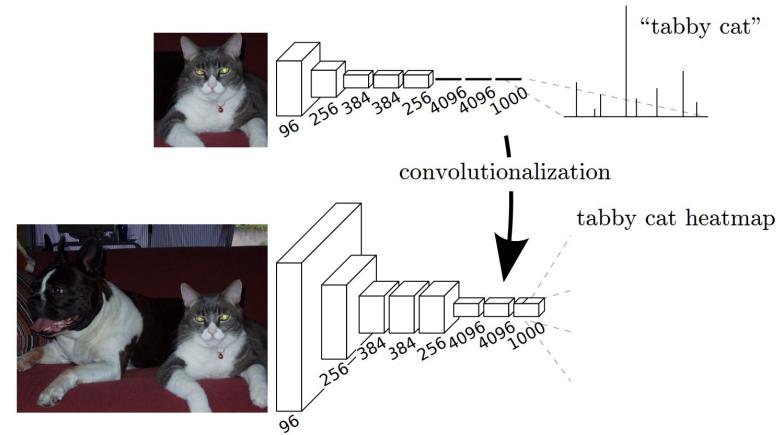
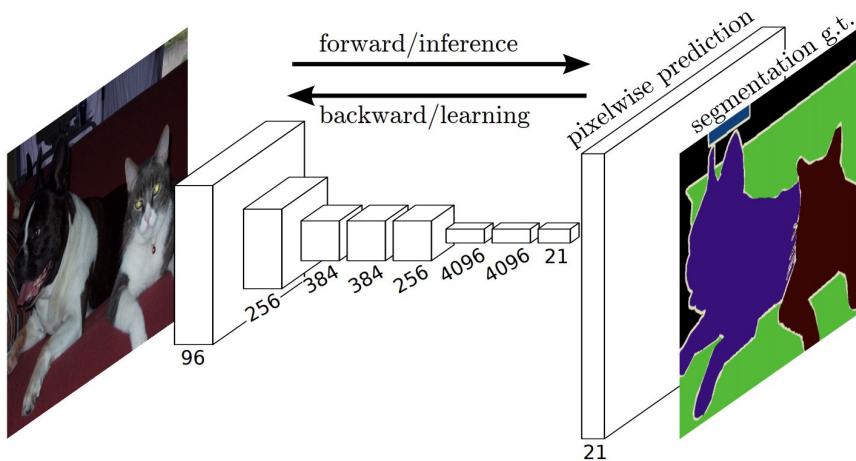
What we do is not a deconvolution.

Please be precise in terms

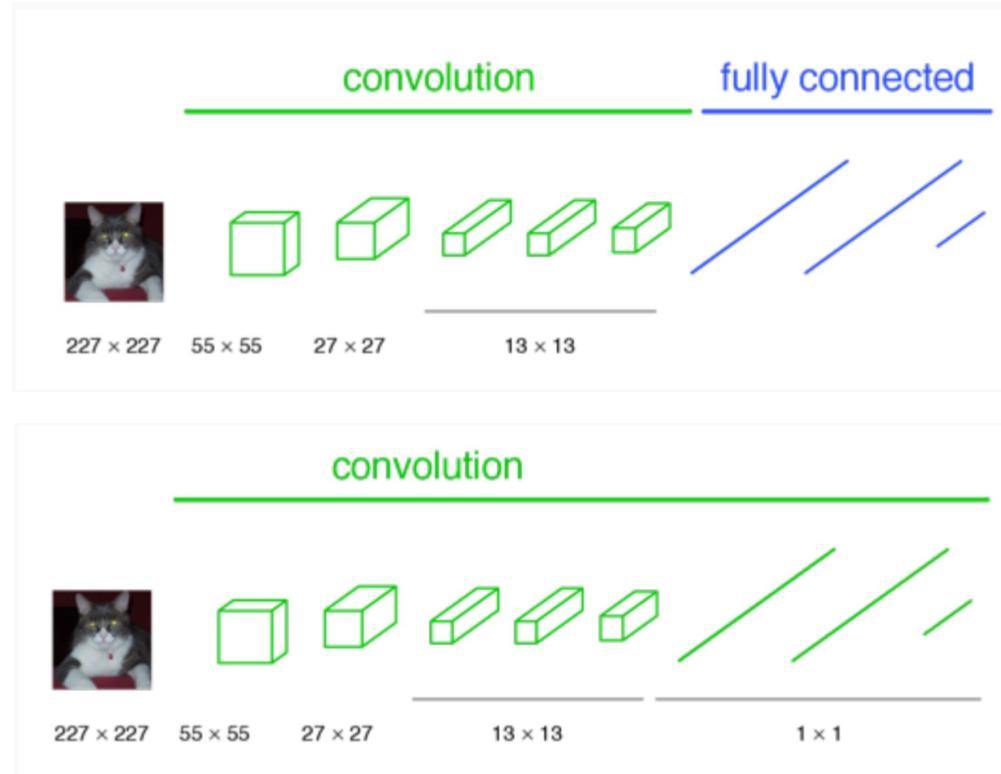
Right terms:

- Transposed convolution
- Upconvolution

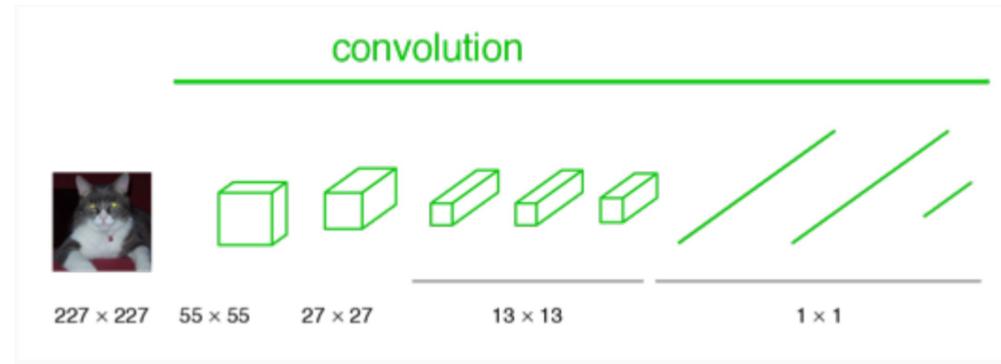
# FCN - Fully Convolutional Network



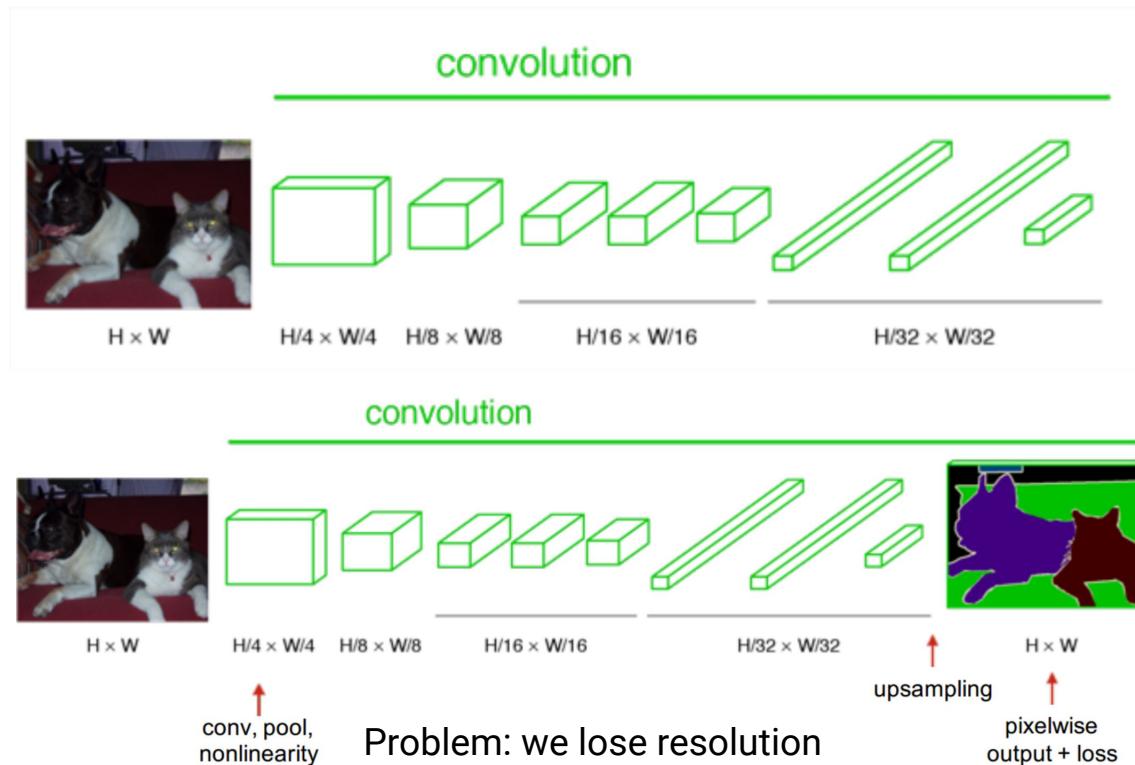
# Regular CNN to FCN



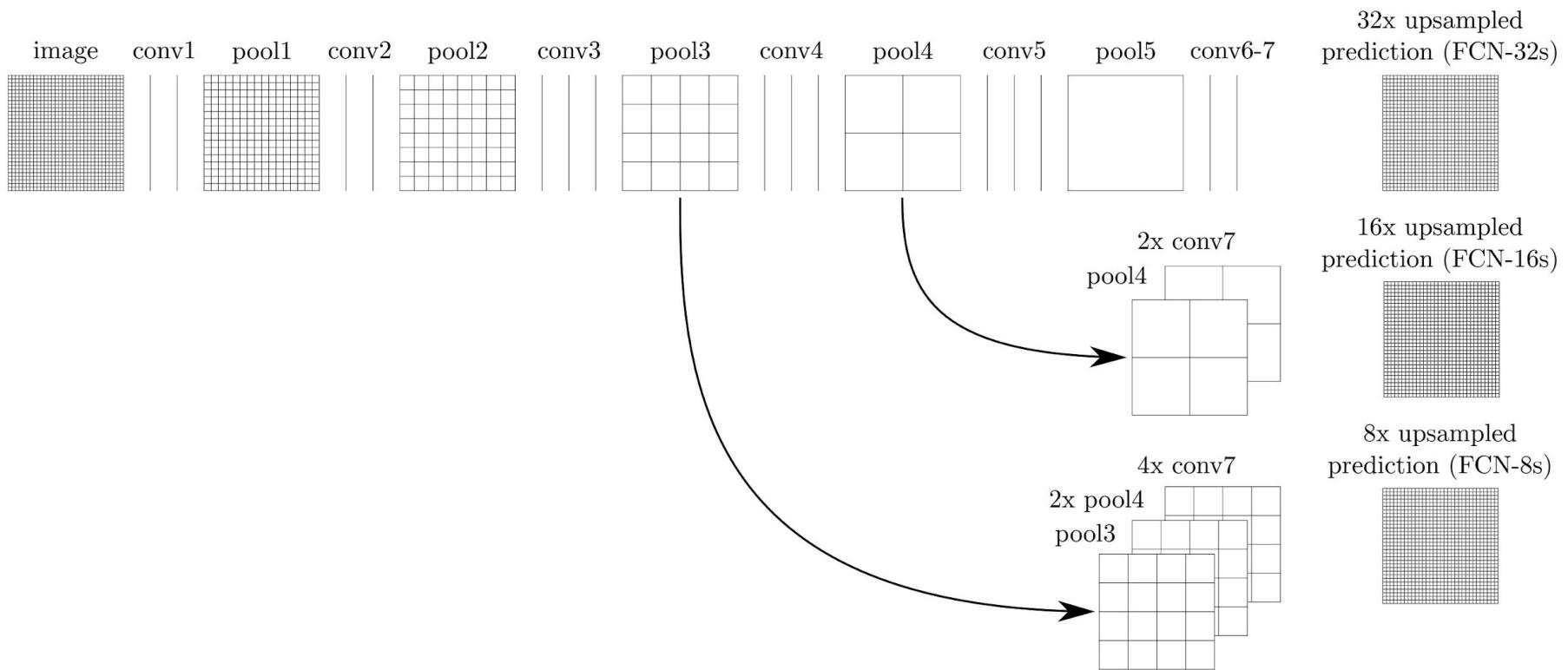
# Regular CNN to FCN



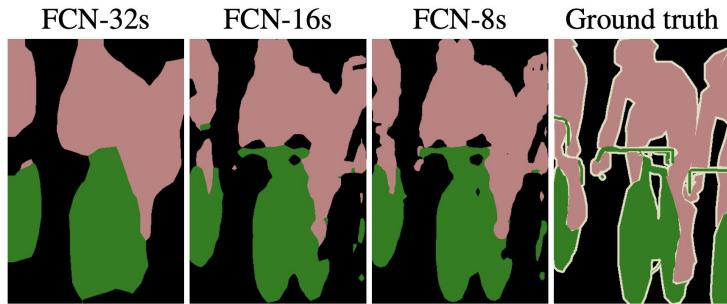
# Regular CNN to FCN



# FCN: Introduce skip connections

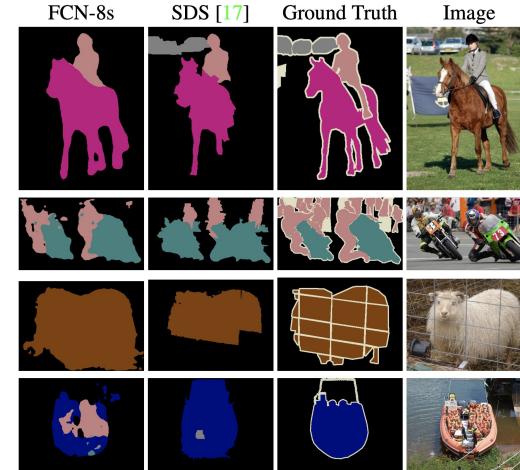


# FCN: Results



	FCN-AlexNet	FCN-VGG16	FCN-GoogLeNet <sup>4</sup>
mean IU	39.8	<b>56.0</b>	42.5
forward time	50 ms	210 ms	59 ms
conv. layers	8	16	22
parameters	57M	134M	6M
rf size	355	404	907
max stride	32	32	32

	pixel acc.	mean acc.	mean IU	f.w. IU
FCN-32s-fixed	83.0	59.7	45.4	72.0
FCN-32s	89.1	73.3	59.4	81.4
FCN-16s	90.0	75.7	62.4	83.0
FCN-8s	<b>90.3</b>	<b>75.9</b>	<b>62.7</b>	<b>83.2</b>



# DeconvNet: use Max Unpooling

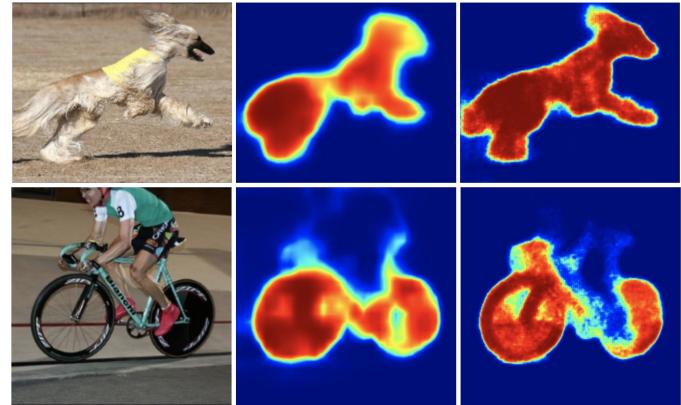
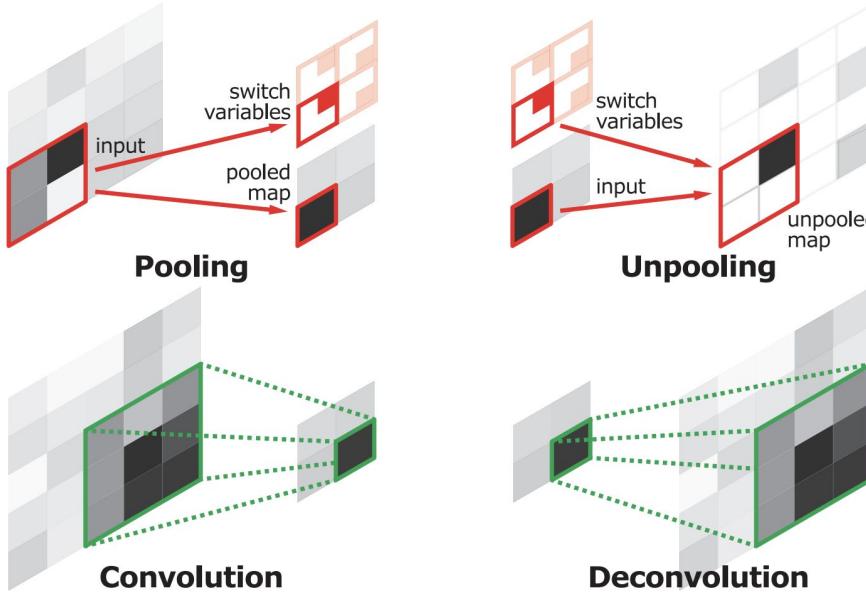


Figure 5. Comparison of class conditional probability maps from FCN and our network (top: dog, bottom: bicycle).

# DeconvNet: use Max Unpooling

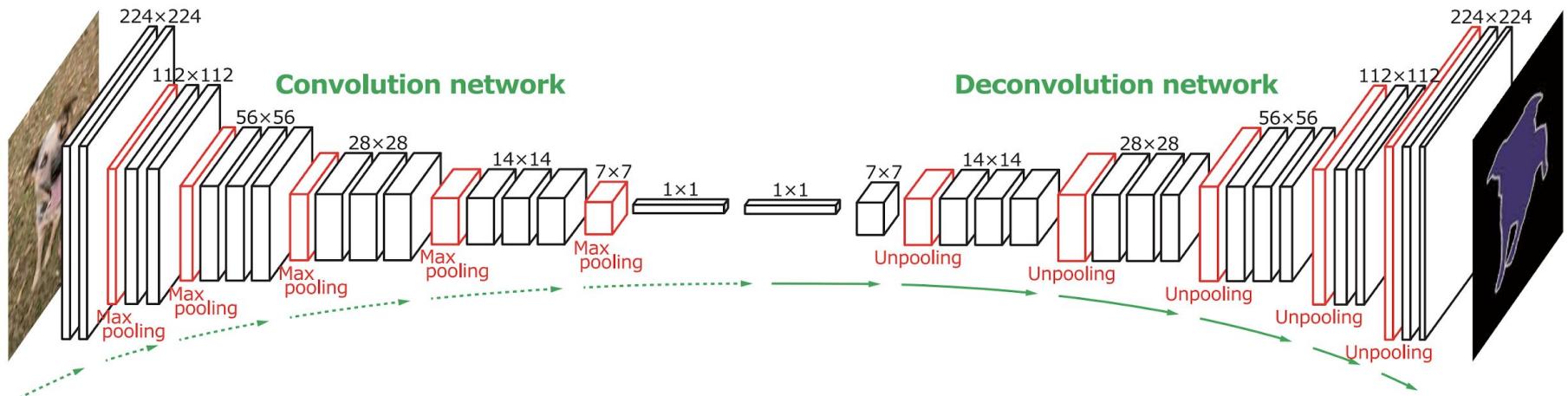
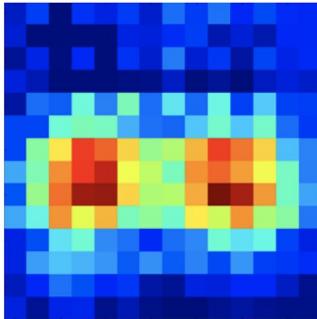


Figure 2. Overall architecture of the proposed network. On top of the convolution network based on VGG 16-layer net, we put a multi-layer deconvolution network to generate the accurate segmentation map of an input proposal. Given a feature representation obtained from the convolution network, dense pixel-wise class prediction map is constructed through multiple series of unpooling, deconvolution and rectification operations.

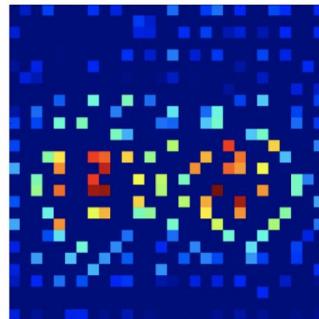
# DeconvNet: use Max Unpooling



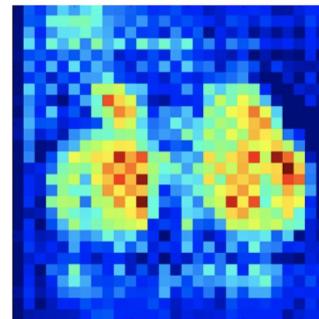
(a)



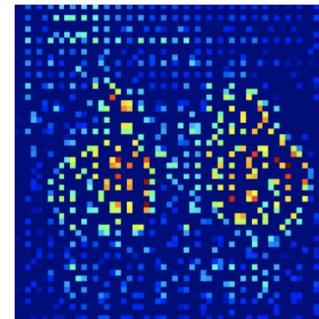
(b)



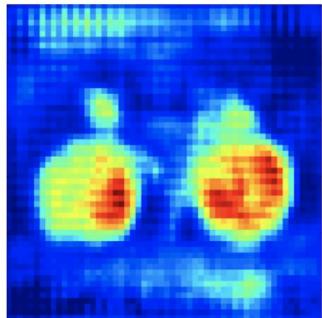
(c)



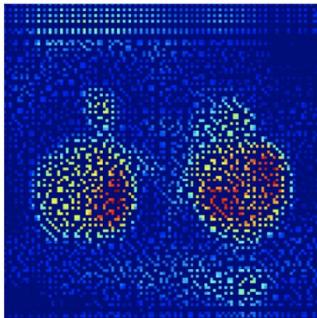
(d)



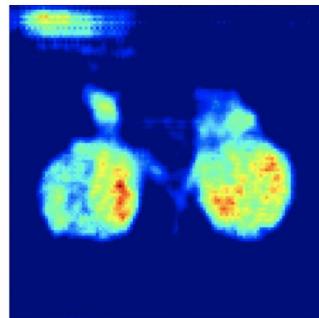
(e)



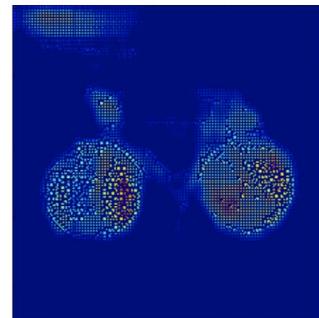
(f)



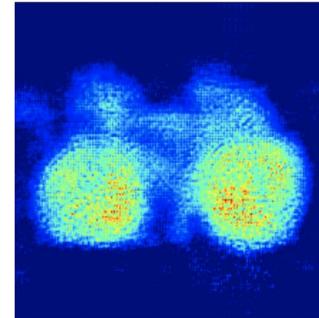
(g)



(h)



(i)



(j)

# SegNet: Further development

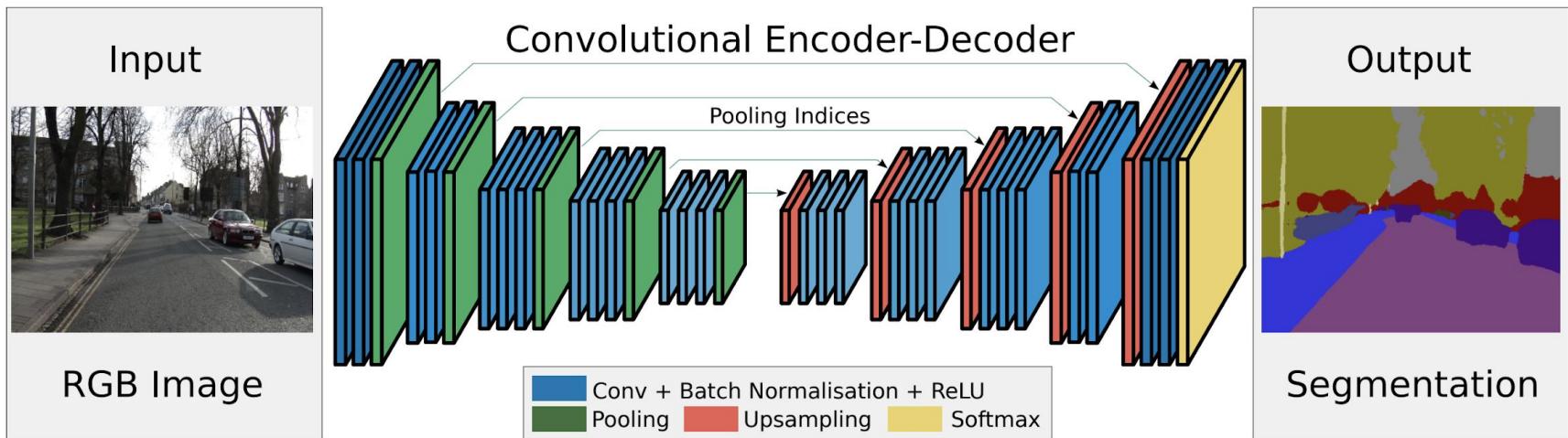
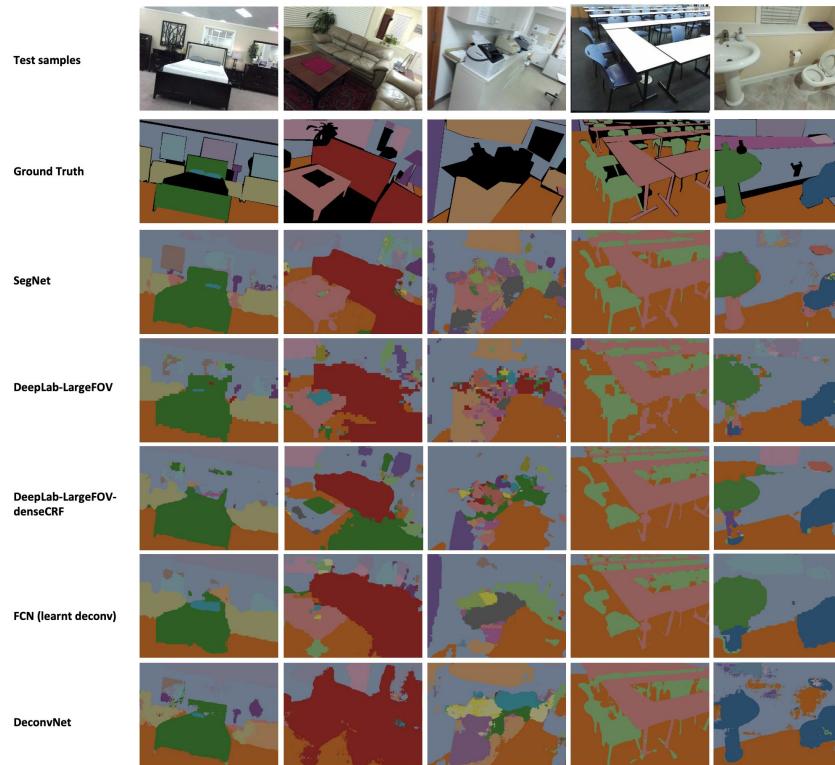
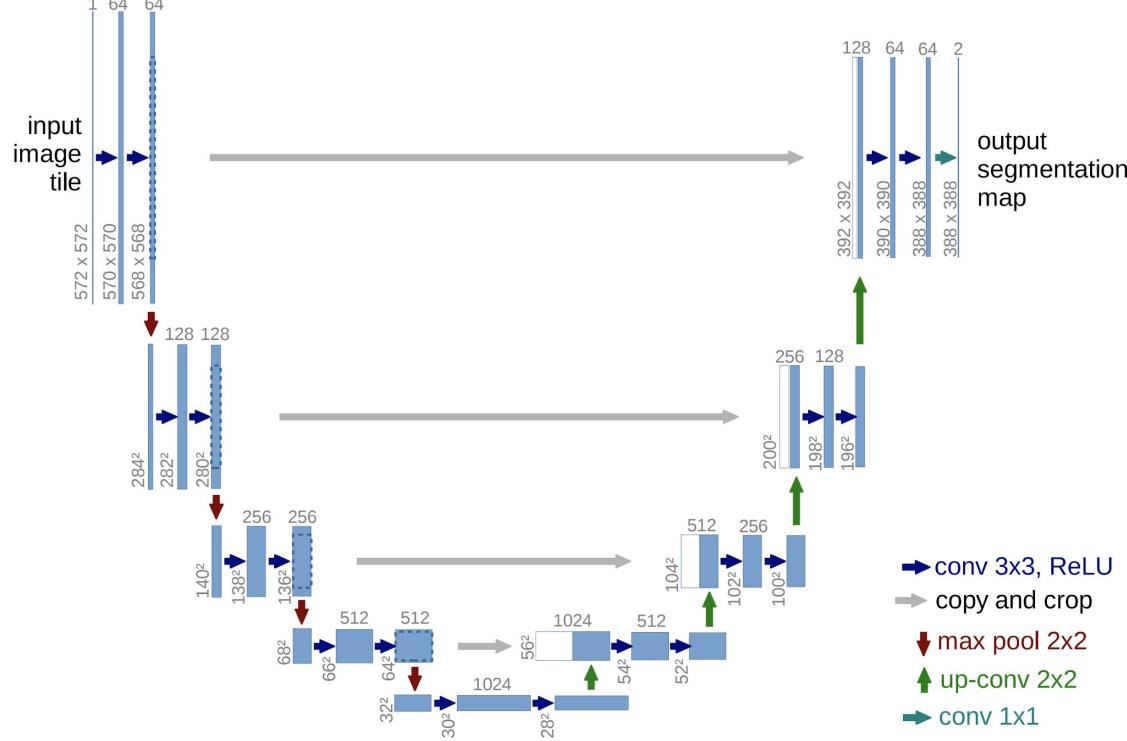


Fig. 2. An illustration of the SegNet architecture. There are no fully connected layers and hence it is only convolutional. A decoder upsamples its input using the transferred pool indices from its encoder to produce a sparse feature map(s). It then performs convolution with a trainable filter bank to densify the feature map. The final decoder output feature maps are fed to a soft-max classifier for pixel-wise classification.

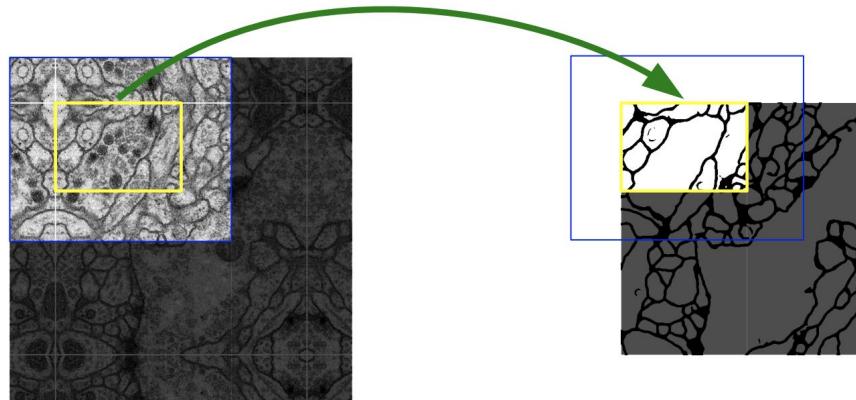
# SegNet: Further development



# U-Net

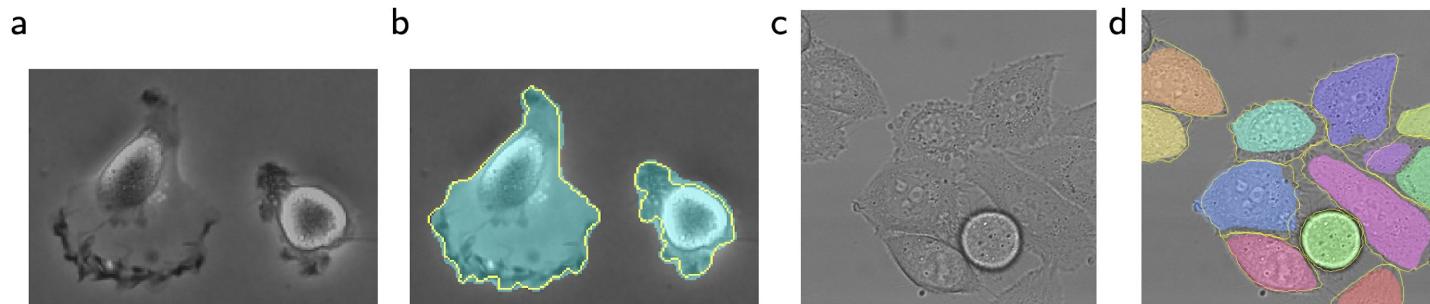
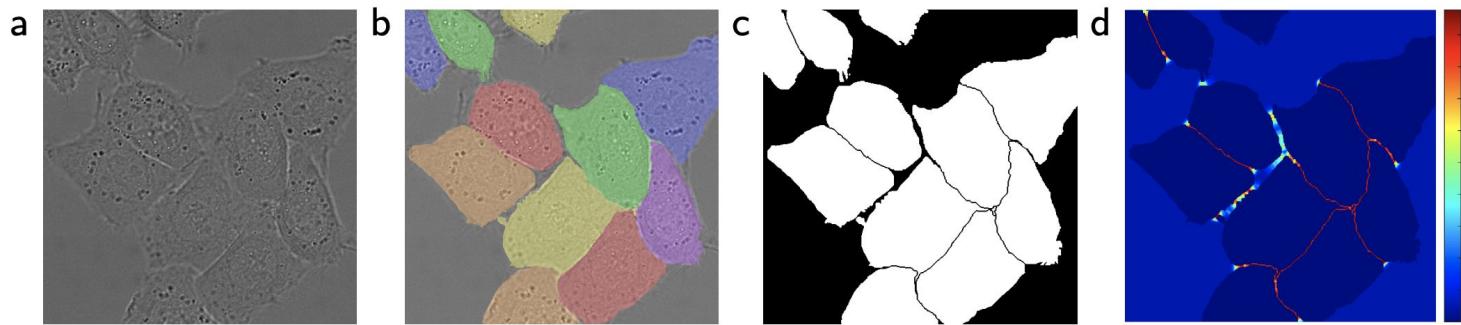


# U-Net



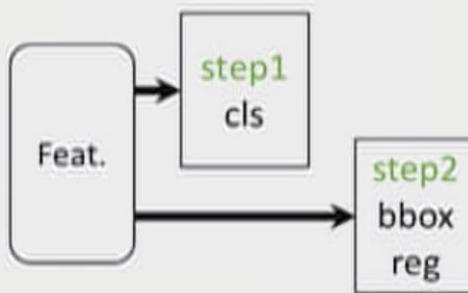
**Fig. 2.** Overlap-tile strategy for seamless segmentation of arbitrary large images (here segmentation of neuronal structures in EM stacks). Prediction of the segmentation in the yellow area, requires image data within the blue area as input. Missing input data is extrapolated by mirroring

# U-Net

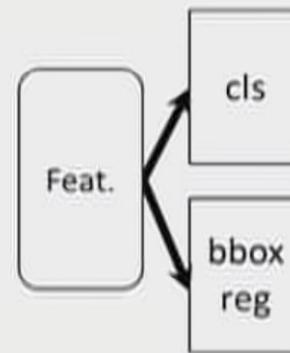


# Mask R-CNN: Instance segmentation

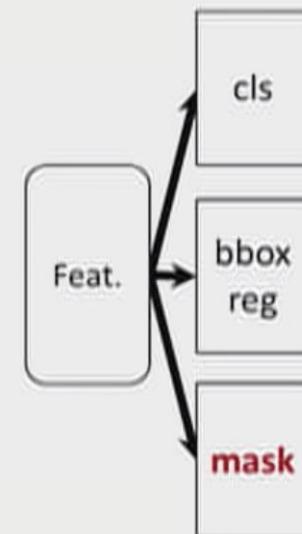
- Easy, fast to implement and use



(slow) R-CNN

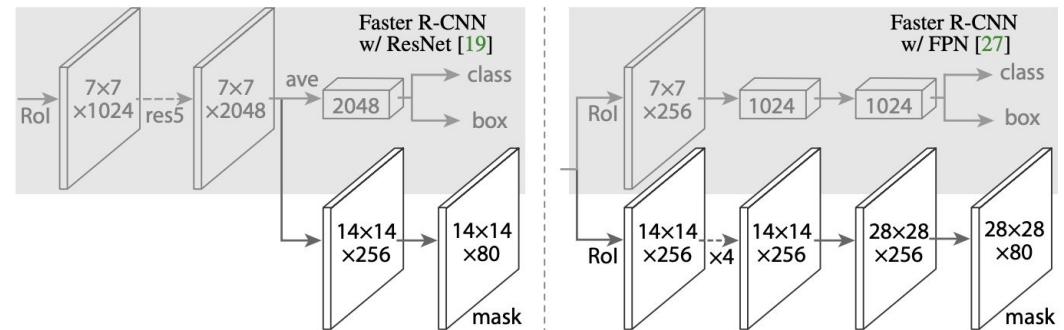
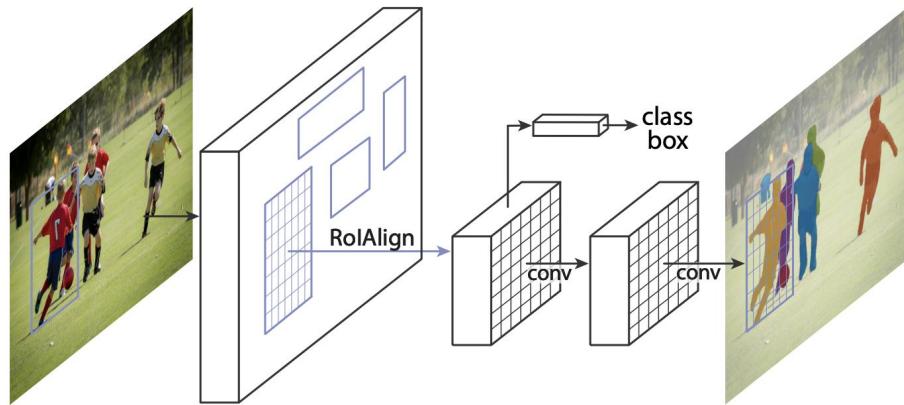


Fast(er) R-CNN

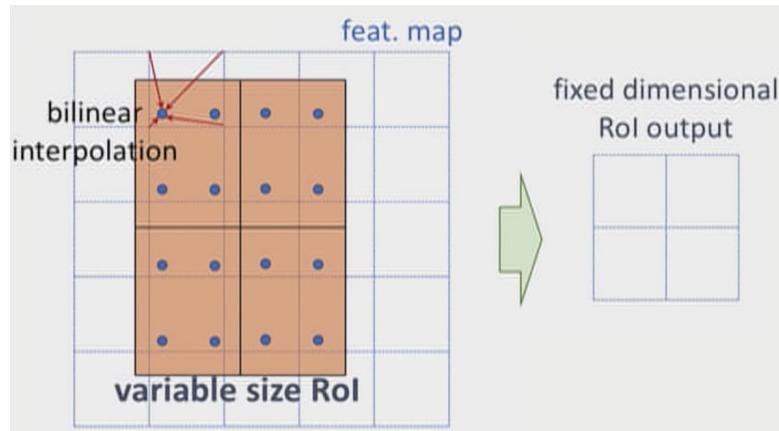


Mask R-CNN

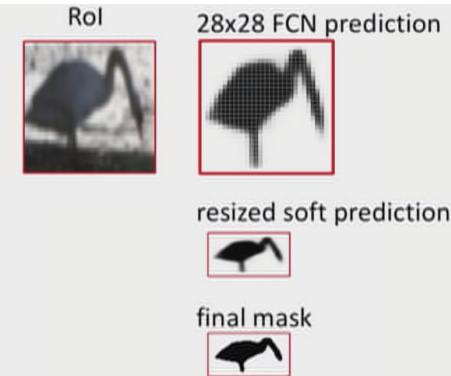
# Mask R-CNN



# Mask R-CNN: RoI Align



- Pixel-to-pixel aligned

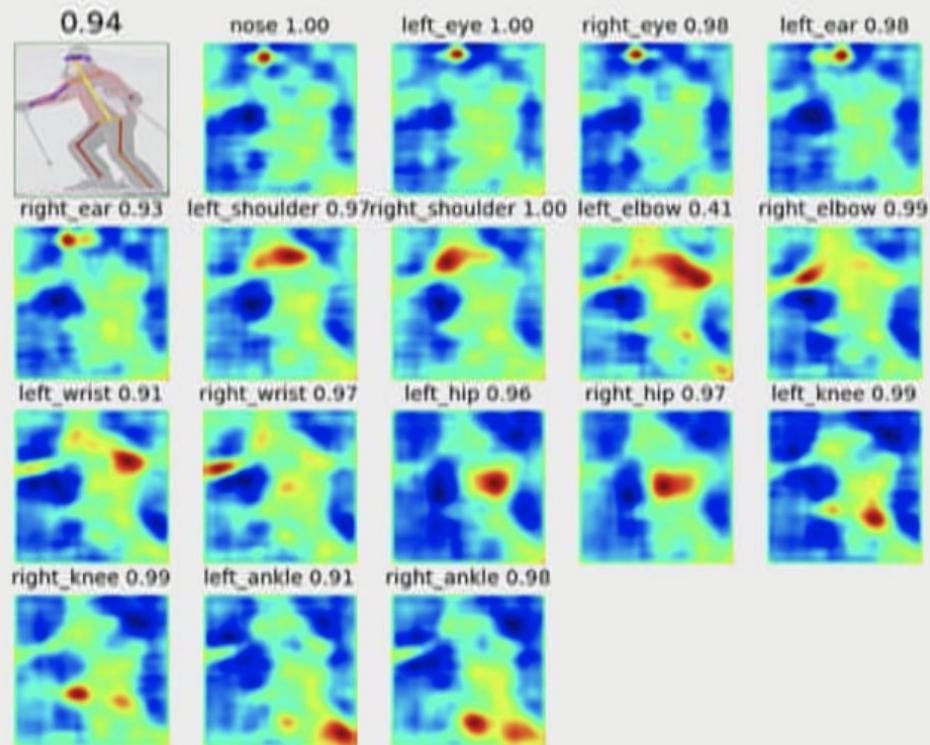


# Mask R-CNN: Results



# Bonus: Pose estimation

- keypoint = 1-hot mask
- human pose = 17 masks
- One framework for
  - ✓ bbox
  - ✓ mask
  - ✓ keypoint



# Bonus: Pose estimation



# Revise

- Segmentation task
- Simple solutions
- Upsampling methods
  - Unpooling
  - Transposed convolution
- FCN
- DeconvNet
- SegNet
- U-Net
- Mask R-CNN

# Next time

- Variational AutoEncoders
- GANs

[Interactive playground](#) for image generation

edges2shoes

