Image Segmentation

Advanced Computer Vision

Niels Landwehr

Overview

- Introduction: Computer Vision
- Data, Models, Optimization
- Neural Networks and Automatic Differentiation
- Convolutional Architectures For Image Classification
- Visualization and Transfer Learning
- Metric Learning
- Image Segmentation

Problem Setting Segmentation

- Problem setting image segmentation
  - Input: image as $m \text{ x } l \text{ x } d$ tensor (typically, d=3).
  - Output: for each of the pixels, a class label from a predefined set of classes $\{c_1, \ldots, c_k\}$ (semantic category).
  - Typically, output is an $m \text{ x } l \text{ x } k$ tensor of class scores.

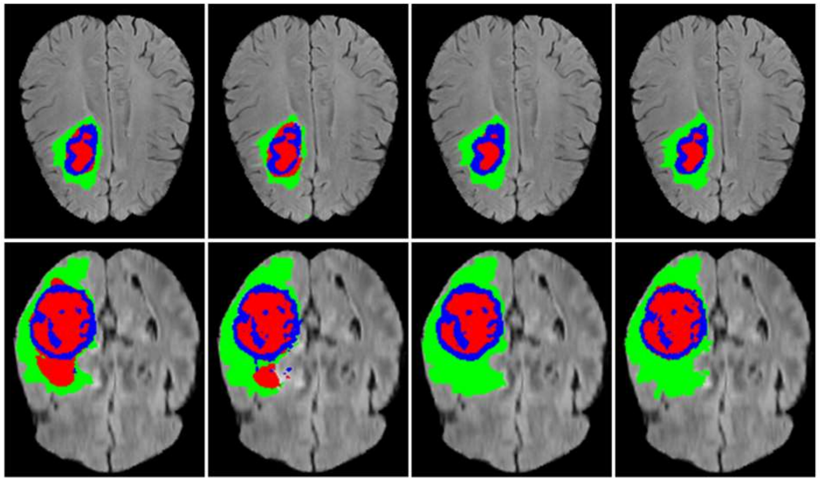
{Trees, Sky, Cow, Grass, ...}

- More fine-grained analysis compared to classification or object detection.
- **To arrive at segmentation, model needs to (implicitly) identify and classify objects in image and find their exact contours**

Problem Setting Segmentation

- Many applications: autonomous driving, medical domains, ...

Cordts, Marius, et al. "The cityscapes dataset." *CVPR Workshop on the Future of Datasets in Vision*. Vol. 2. 2015.

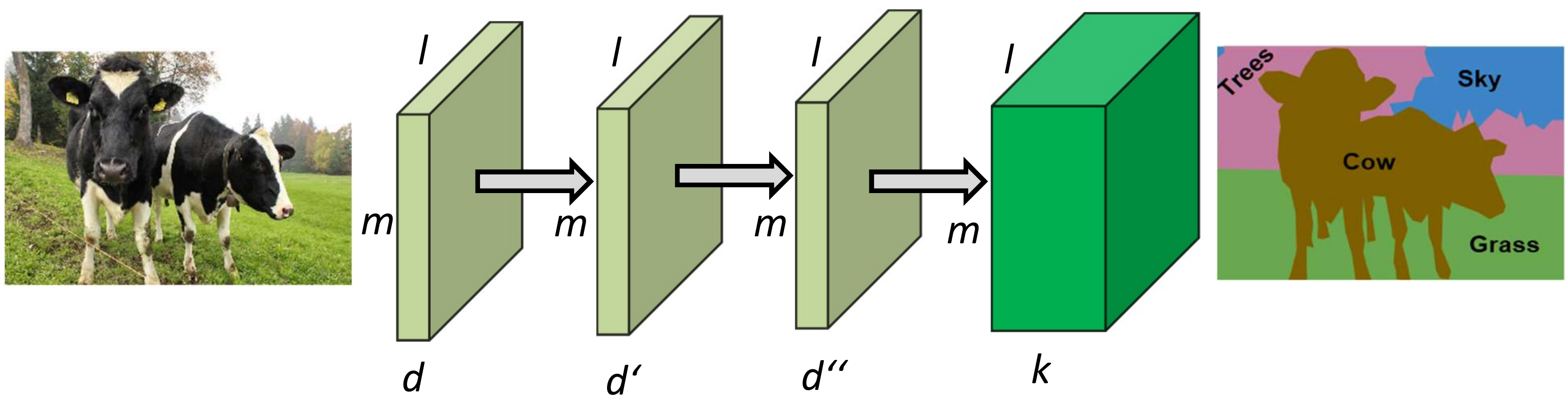


Brain tumor segmentation in MRI images

Chen, Chen, et al. "3D dilated multi-fiber network for real-time brain tumor segmentation in MRI." *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, Cham, 2019.

Network Architectures for Segmentation?

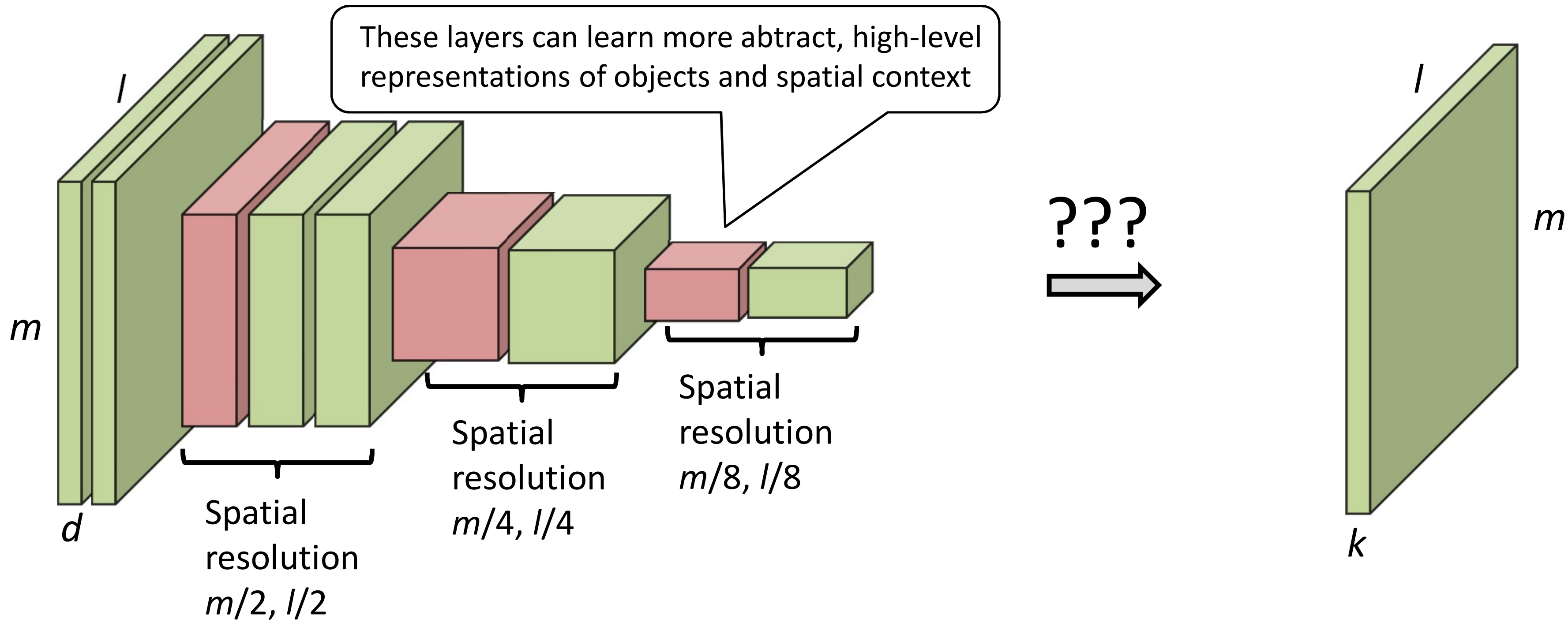
- How can we design a network architecture that maps m x l x d tensors to m x l x k tensors (and implicitly reasons about spatial objects)?
- First idea: use convolution layers with spatial dimension m x l throughout



- Problem: without spatial pooling, difficult to learn about larger-scale objects in images. Without large-scale context cannot perform good segmentation.
- Problem: convolutions at full spatial resolutions are expensive.

Spatial Downsampling

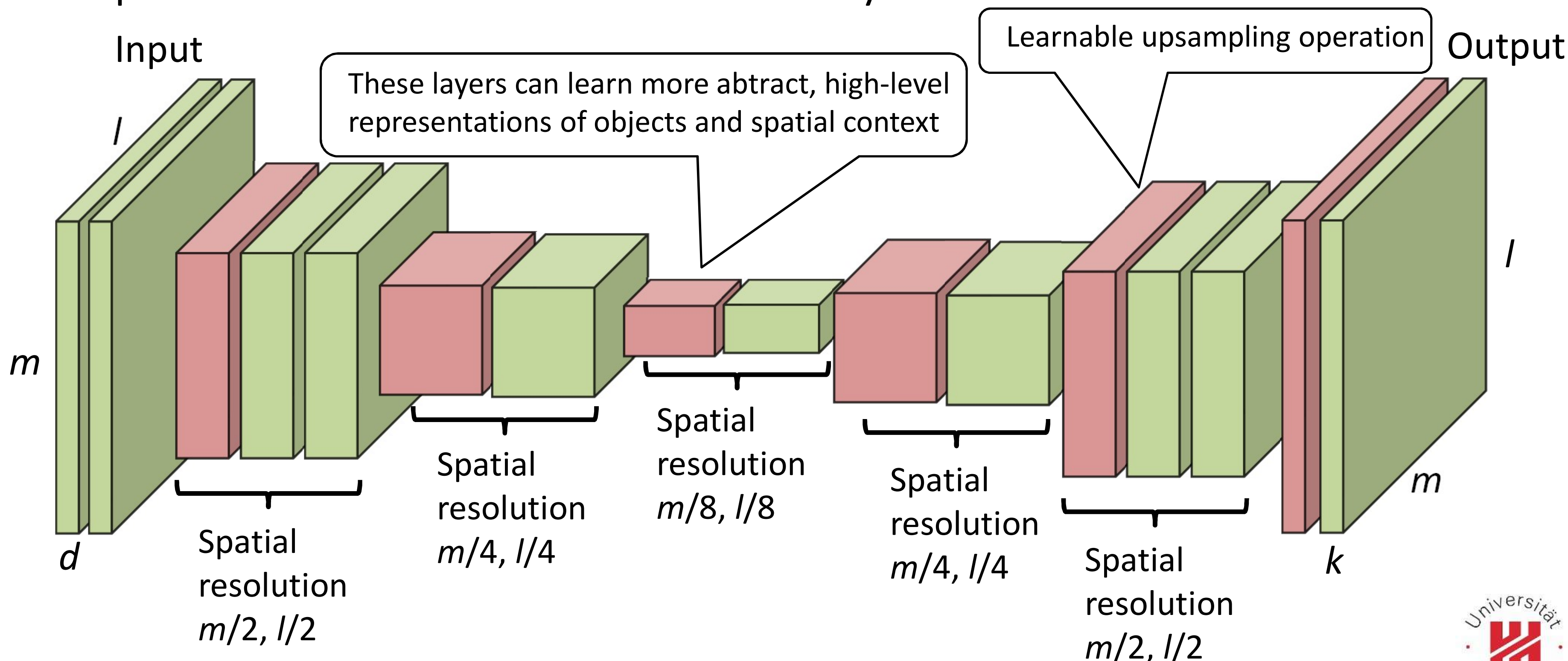
- To reason about large-scale objects (and reduce computational cost), should use pyramid-shaped architecture as in classification networks
- Pooling or strided convolution layers to reduce spatial dimension



- However, need high resolution in output. How do we go back?

Spatial Downsampling and Upsampling

- **Idea**: first downsample by pooling and strided convolutions, then upsample again to the final resolution
- Learnable upsampling operation that again constructs high-resolution features and finally class labels based on the more abstract low-resolution representations learned at intermediate layers



Recap: Convolution Layers

- **Transposed convolution layers:** learnable upsampling operation
- **Recap**: normal convolution operation
  - Learnable k x k kernel matrix
  - Move kernel matrix across input to generate single element in output
  - Multiply kernel elements with input elements and sum up
  - Stride controls movement of kernel in input, stride > 1 reduces output size

Standard convolution: 6 x 6 input, 3 x 3 kernel, stride one

Input

0	0	0	0	0	0
0	-1	2	2	1	0
0	1	-1	1	2	0
0	-1	0	2	1	0
0	1	1	0	2	0
0	0	0	0	0	0

Kernel

1	0	-1
2	1	0
0	1	2

+0

Output

-2			

Recap: Convolution Layers

- **Transposed convolution layers:** learnable upsampling operation
- **Recap**: normal convolution operation
  - Learnable k x k kernel matrix
  - Move kernel matrix across input to generate single element in output
  - Multiply kernel elements with input elements and sum up
  - Stride controls movement of kernel in input, stride > 1 reduces output size

Standard convolution: 6 x 6 input, 3 x 3 kernel, stride one

Input

0	0	0	0	0	0
0	-1	2	2	1	0
0	1	-1	1	2	0
0	-1	0	2	1	0
0	1	1	0	2	0
0	0	0	0	0	0

Kernel

1	0	-1
2	1	0
0	1	2

+0

Output

-2	1		

Transposed Convolution Operation

- **Transposed convolution layers:** learnable upsampling operation
  - Learnable k x k kernel matrix
  - Move across single elements in input, multiply kernel matrix with input element and copy the "weighted" kernel matrix to the output
  - Overlapping outputs accumulate
  - Stride controls movement in output space, stride > 1 increases output size

Transposed convolution: 4 x 4 input, 3 x 3 kernel, stride one

Input

1	2	1	0
-1	1	0	1
0	2	-2	1
1	1	2	-1

Kernel

1	0	-1
2	1	0
0	1	2

Weighted kernel

1	0	-1
2	1	0
0	1	2

copy & accumulate

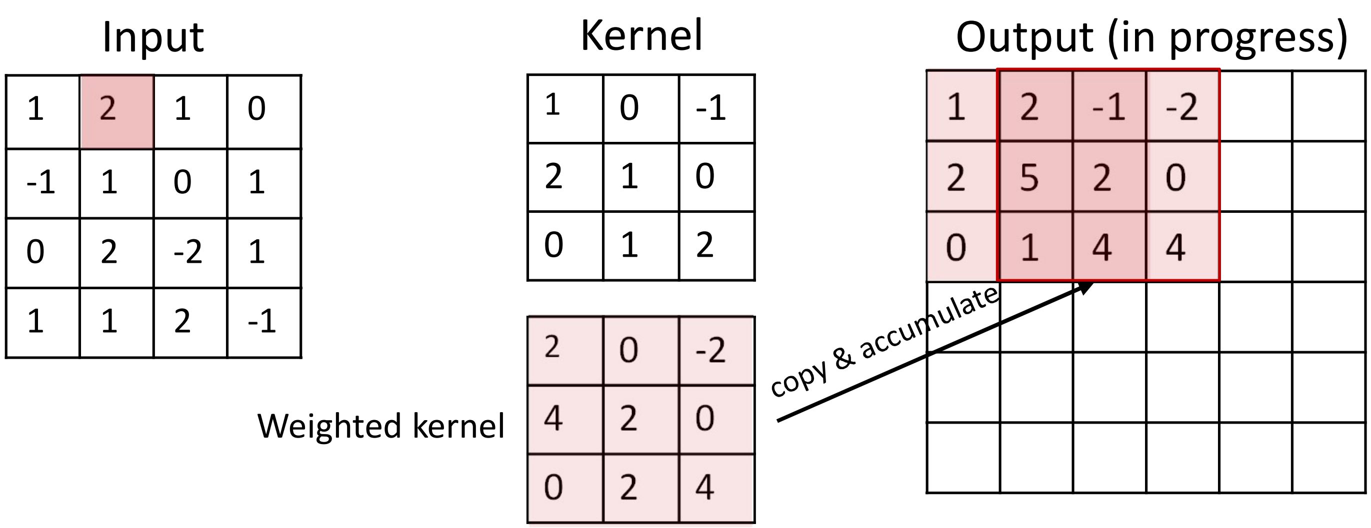
Output (in progress)

1	0	-1			
2	1	0			
0	1	2			

Transposed Convolution Operation

- **Transposed convolution layers:** learnable upsampling operation
  - Learnable k x k kernel matrix
  - Move across single elements in input, multiply kernel matrix with input element and copy the "weighted" kernel matrix to the output
  - Overlapping outputs accumulate
  - Stride controls movement in output space, stride > 1 increases output size

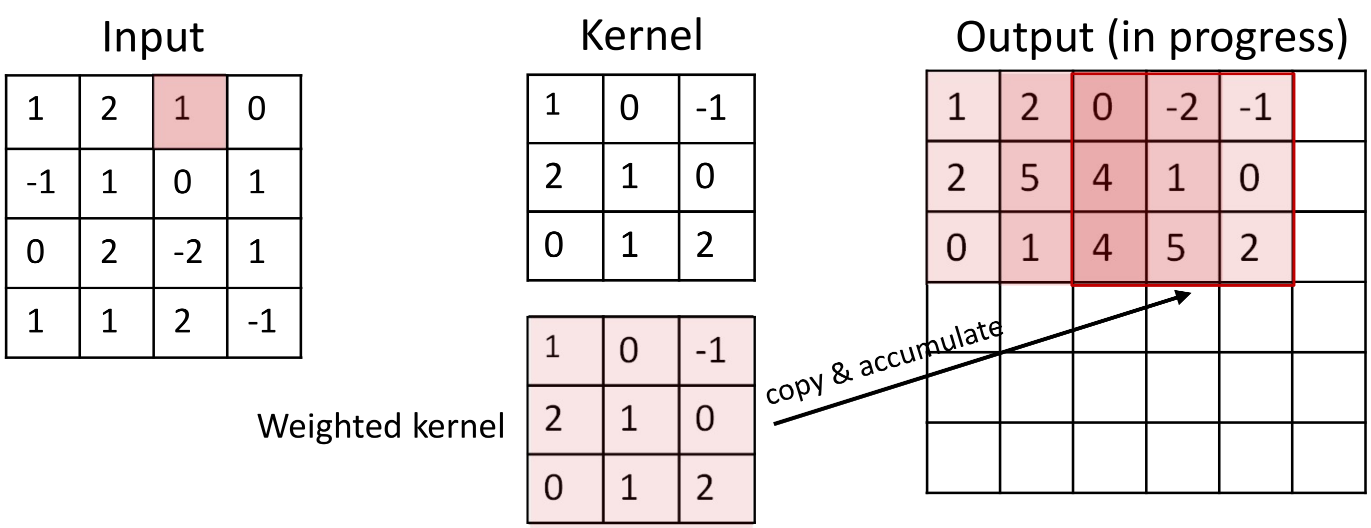
Transposed convolution: 4 x 4 input, 3 x 3 kernel, stride one



Transposed Convolution Operation

- **Transposed convolution layers:** learnable upsampling operation
  - Learnable k x k kernel matrix
  - Move across single elements in input, multiply kernel matrix with input element and copy the "weighted" kernel matrix to the output
  - Overlapping outputs accumulate
  - Stride controls movement in output space, stride > 1 increases output size

Transposed convolution: 4 x 4 input, 3 x 3 kernel, stride one



Transposed Convolution Operation

- **Transposed convolution layers:** learnable upsampling operation
  - Learnable k x k kernel matrix
  - Move across single elements in input, multiply kernel matrix with input element and copy the "weighted" kernel matrix to the output
  - Overlapping outputs accumulate
  - Stride controls movement in output space, stride > 1 increases output size

Transposed convolution: 4 x 4 input, 3 x 3 kernel, stride one

Input

1	2	1	0
-1	1	0	1
0	2	-2	1
1	1	2	-1

Kernel

1	0	-1
2	1	0
0	1	2

Weighted kernel

0	0	0
0	0	0
0	0	0

copy & accumulate

Output (in progress)

1	2	0	-2	-1	0
2	5	4	1	0	0
0	1	4	5	2	0

Transposed Convolution Operation

- **Transposed convolution layers:** learnable upsampling operation
  - Learnable k x k kernel matrix
  - Move across single elements in input, multiply kernel matrix with input element and copy the "weighted" kernel matrix to the output
  - Overlapping outputs accumulate
  - Stride controls movement in output space, stride > 1 increases output size

Transposed convolution: 4 x 4 input, 3 x 3 kernel, stride one

Input

1	2	1	0
-1	1	0	1
0	2	-2	1
1	1	2	-1

Kernel

1	0	-1
2	1	0
0	1	2

Weighted kernel

-1	0	1
-2	-1	0
0	-1	-2

copy & accumulate

Output (in progress)

1	2	0	-2	-1	0
1	5	5	1	0	0
-2	0	4	5	2	0
0	-1	-2			

Transposed Convolution Operation

- **Transposed convolution layers:** learnable upsampling operation
  - Learnable k x k kernel matrix
  - Move across single elements in input, multiply kernel matrix with input element and copy the "weighted" kernel matrix to the output
  - Overlapping outputs accumulate
  - Stride controls movement in output space, stride > 1 increases output size

Transposed convolution: 4 x 4 input, 3 x 3 kernel, stride one

Input

1	2	1	0
-1	1	0	1
0	2	-2	1
1	1	2	-1

Kernel

1	0	-1
2	1	0
0	1	2

Weighted kernel

1	0	-1
2	1	0
0	1	2

copy & accumulate

Output (in progress)

1	2	0	-2	-1	0
1	6	5	0	0	0
-2	2	5	5	2	0
0	-1	-1	2		

Transposed Convolution Operation: Padding

- **Padding for transposed convolution operation**: preserve size at stride=1
- For standard convolution, we pad the input with zeros to preserve size

Padded Input

0	0	0	0	0	0
0	-1	2	2	1	0
0	1	-1	1	2	0
0	-1	0	2	1	0
0	1	1	0	2	0
0	0	0	0	0	0

Kernel

1	0	-1
2	1	0
0	1	2

+0

Output

-2			

- For transposed convolutions, we crop the output (also called padding)

Input

1	2	1	0
-1	1	0	1
0	2	-2	1
1	1	2	-1

Kernel

1	0	-1
2	1	0
0	1	2

Output (in progress)

1	2	0	-2	-1	0
1	6	5	0	0	0
-2	2	5	5	2	0
0	-1	-1	2		

Transposed Convolution Operation: Striding

- **Striding for transposed convolution operation**: actual upsampling
  - Striding s moves kernel in output space by s elements each time
  - If $s > 1$, this increases output size and performs upsampling

Example: Stride 2

Input

1	2	1	0
-1	1	0	1
0	2	-2	1
1	1	2	-1

Kernel

1	0	-1
2	1	0
0	1	2

Weighted kernel

1	0	-1
2	1	0
0	1	2

Output (in progress)

1	0	-1						
2	1	0						
0	1	2						

Transposed Convolution Operation: Striding

- **Striding for transposed convolution operation**: actual upsampling
  - Striding s moves kernel in output space by s elements each time
  - If $s > 1$, this increases output size and performs upsampling

Example: Stride 2

Input

1	2	1	0
-1	1	0	1
0	2	-2	1
1	1	2	-1

Kernel

1	0	-1
2	1	0
0	1	2

Weighted kernel

2	0	-2
4	2	0
0	2	4

Output (in progress)

1	0	1	0	-2				
2	1	4	2	0				
0	1	2	2	4				

Transposed Convolution Operation: Striding

- **Striding for transposed convolution operation**: actual upsampling
  - Striding s moves kernel in output space by s elements each time
  - If $s > 1$, this increases output size and performs upsampling

Example: Stride 2

Input

1	2	1	0
-1	1	0	1
0	2	-2	1
1	1	2	-1

Kernel

1	0	-1
2	1	0
0	1	2

Weighted kernel

1	0	-1
2	1	0
0	1	2

Output (in progress)

1	0	1	0	-1	0	-1		
2	1	4	2	2	1	0		
0	1	2	2	4	1	2		

Transposed Convolution Operation: Striding

- **Striding for transposed convolution operation**: actual upsampling
  - Striding s moves kernel in output space by s elements each time
  - If $s > 1$, this increases output size and performs upsampling

Example: Stride 2

Input

1	2	1	0
-1	1	0	1
0	2	-2	1
1	1	2	-1

Kernel

1	0	-1
2	1	0
0	1	2

Weighted kernel

0	0	0
0	0	0
0	0	0

Output (in progress)

1	0	1	0	-1	0	-1	0	0
2	1	4	2	2	1	0	0	0
0	1	2	2	4	1	2	0	0

Transposed Convolution Operation: Striding

- **Striding for transposed convolution operation**: actual upsampling
  - Striding s moves kernel in output space by s elements each time
  - If $s > 1$, this increases output size and performs upsampling

Example: Stride 2

Input

1	2	1	0
-1	1	0	1
0	2	-2	1
1	1	2	-1

Kernel

1	0	-1
2	1	0
0	1	2

Weighted kernel

-1	0	1
-2	-1	0
0	-1	-2

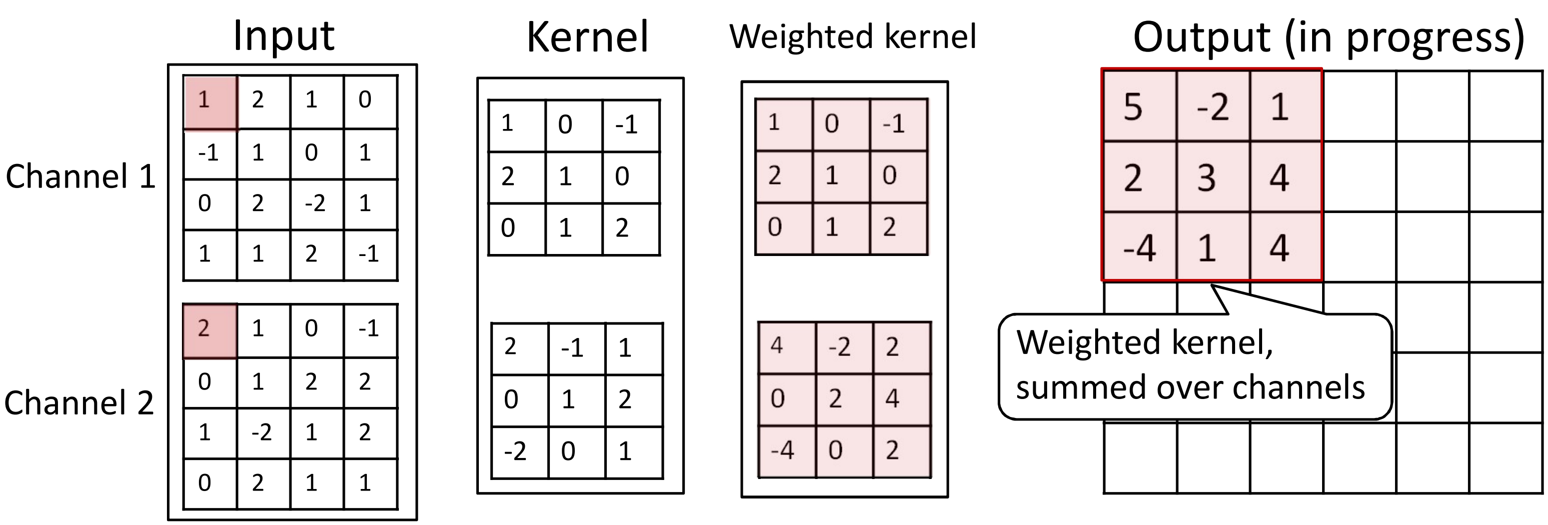
Output (in progress)

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

Transposed Convolutions: Input Channels

- As for normal convolution layers, we usually have multiple input channels for a transposed convolution layer
- As for normal convolution layers, in this case there is a further dimension in the kernel that corresponds to the different input channels
- Each channel in the kernel is multiplied with the element from corresponding channel in input. Results in output are accumulated over the channels

Example: Two input channels, stride 1



Transposed Convolutions: Input Channels

- As for normal convolution layers, we usually have multiple input channels for a transposed convolution layer
- As for normal convolution layers, in this case there is a further dimension in the kernel that corresponds to the different input channels
- Each channel in the kernel is multiplied with the element from corresponding channel in input. Results in output are accumulated over the channels

Example: Two input channels, stride 1

Input

Channel 1

1	2	1	0
-1	1	0	1
0	2	-2	1
1	1	2	-1

Channel 2

2	1	0	-1
0	1	2	2
1	-2	1	2
0	2	1	1

Kernel

1	0	-1
2	1	0
0	1	2

2	-1	1
0	1	2
-2	0	1

Weighted kernel

2	0	-2
4	2	0
0	2	4

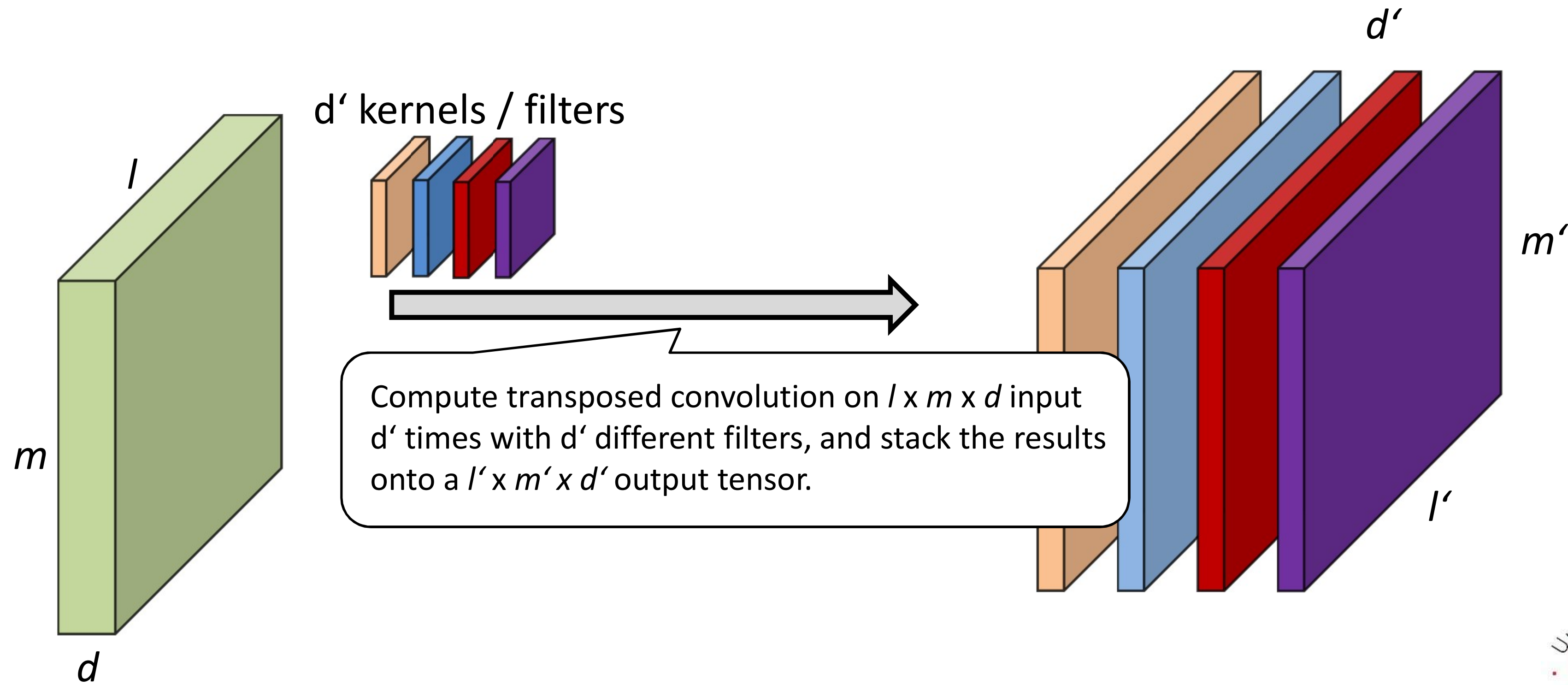
2	-1	1
0	1	2
-2	0	1

Output (in progress)

5	2	0	-1		
2	7	7	2		
-4	-1	6	5		

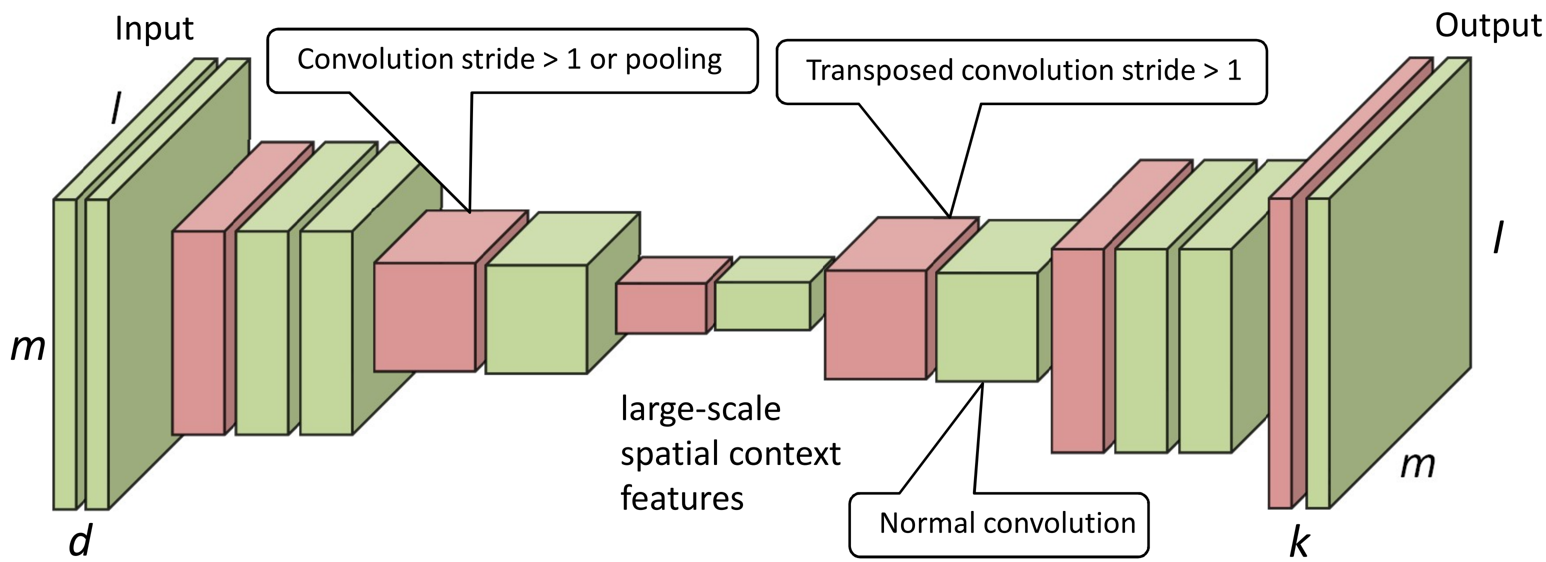
Transposed Convolutions: Output Channels

- As for normal convolution layers, we usually have multiple output channels for a transposed convolution layer
- As for normal convolution layers, each output channel is given by applying a different kernel to the same input in the same way. Resulting output channels are then stacked as usual.



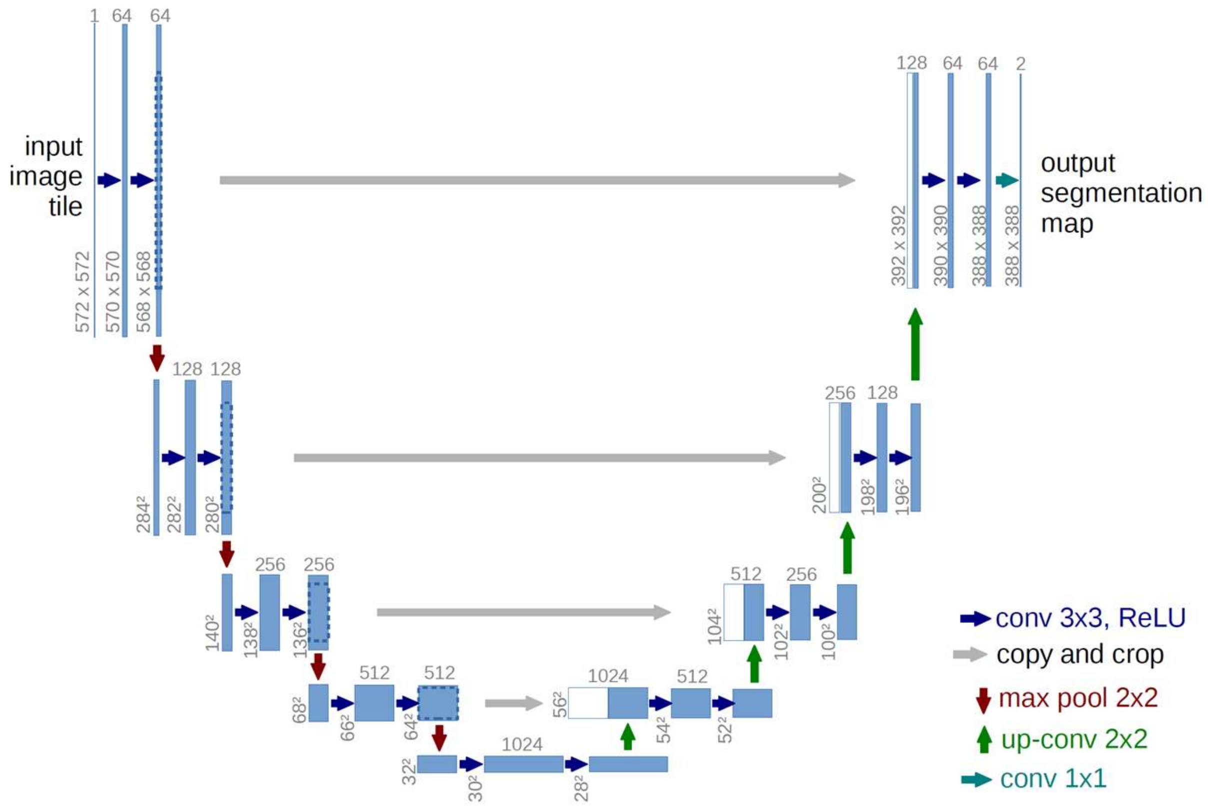
Problem Setting Segmentation

- **Back to full architecture**: Transposed convolution can be used for spatial upsampling
- Also called „deconvolution“, „upconvolution“, „fractionally strided convolution“
- Yields hourglass-shaped architecture with downsampling for learning about large-scale spatial context followed by upsampling for higher spatial resolution features learned based on the intermediate layers
- Upsampling step has learnable parameters, learns spatially high-resolution features that are helpful for the final segmentation



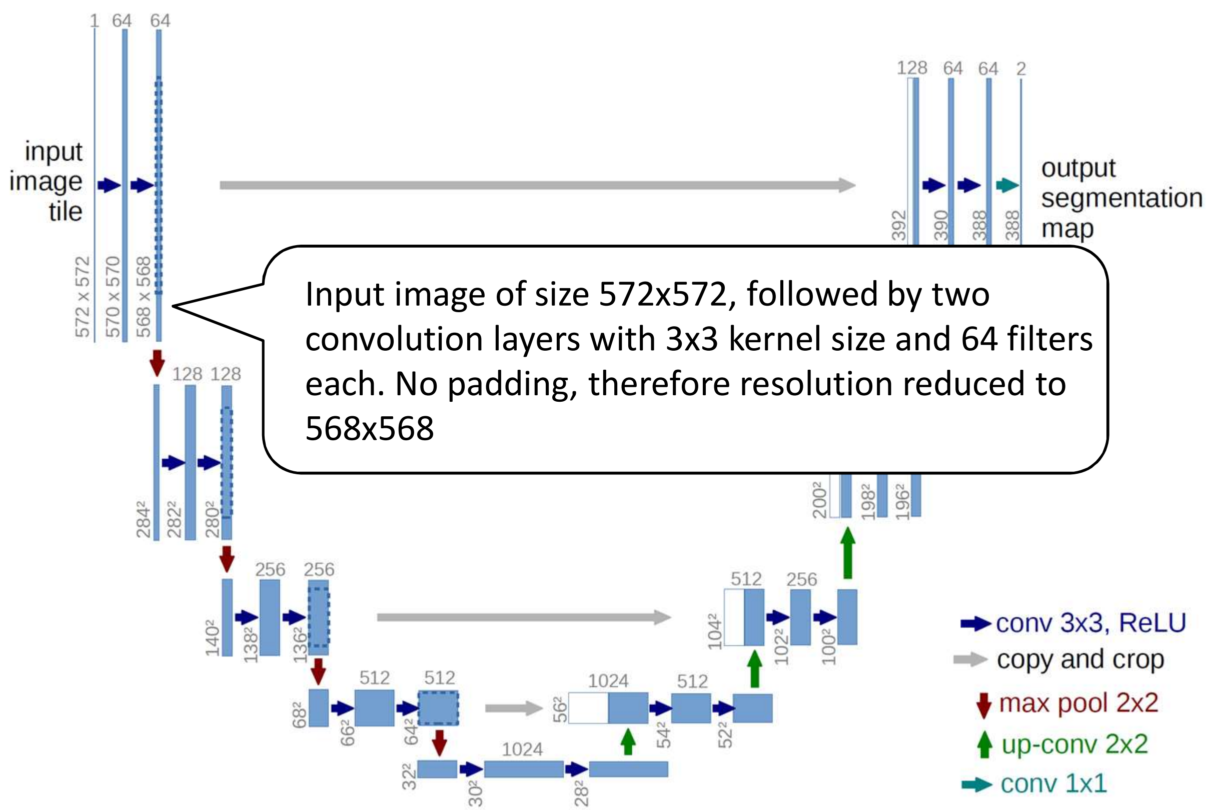
The U-Net Architecture for Segmentation

- U-Net architecture for segmentation [Ronneberger et al., 2015]: downsampling, upsampling, and direct connections between lower and higher layers



The U-Net Architecture for Segmentation

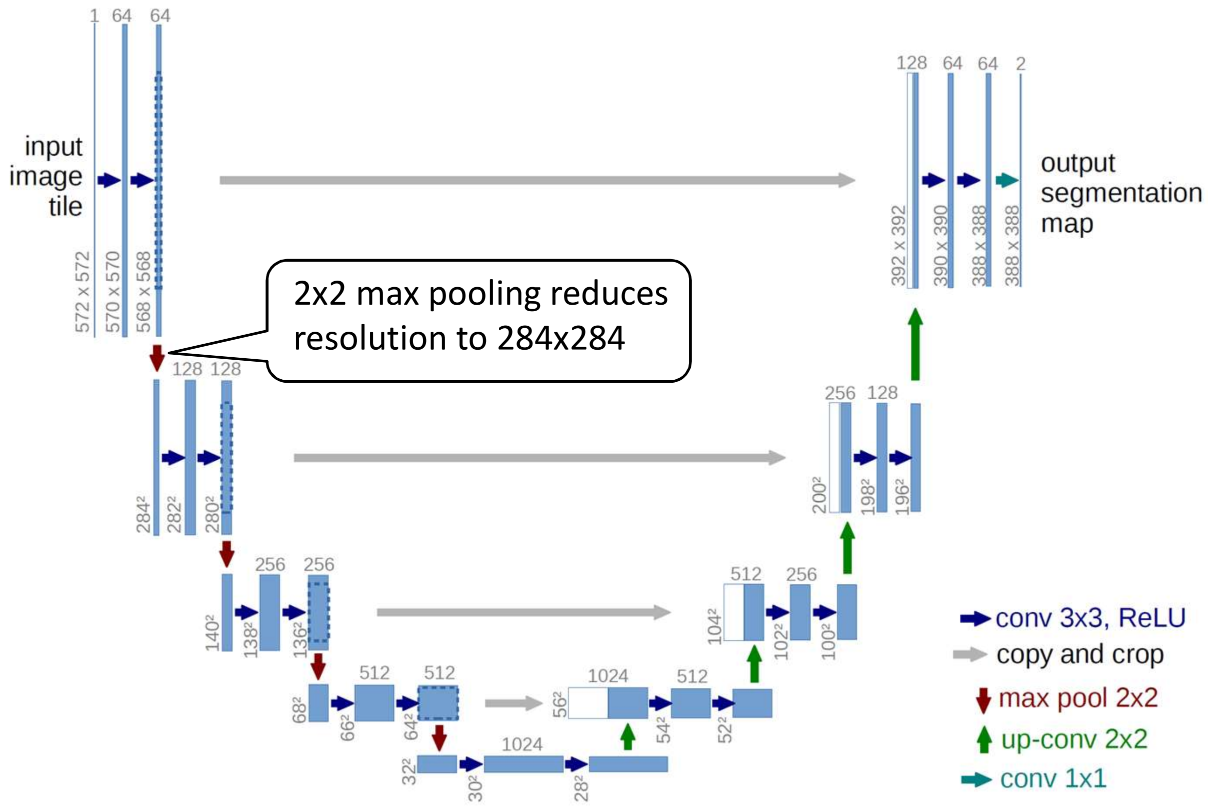
- U-Net architecture for segmentation [Ronneberger et al., 2015]: downsampling, upsampling, and direct connections between lower and higher layers



Input image of size 572x572, followed by two convolution layers with 3x3 kernel size and 64 filters each. No padding, therefore resolution reduced to 568x568

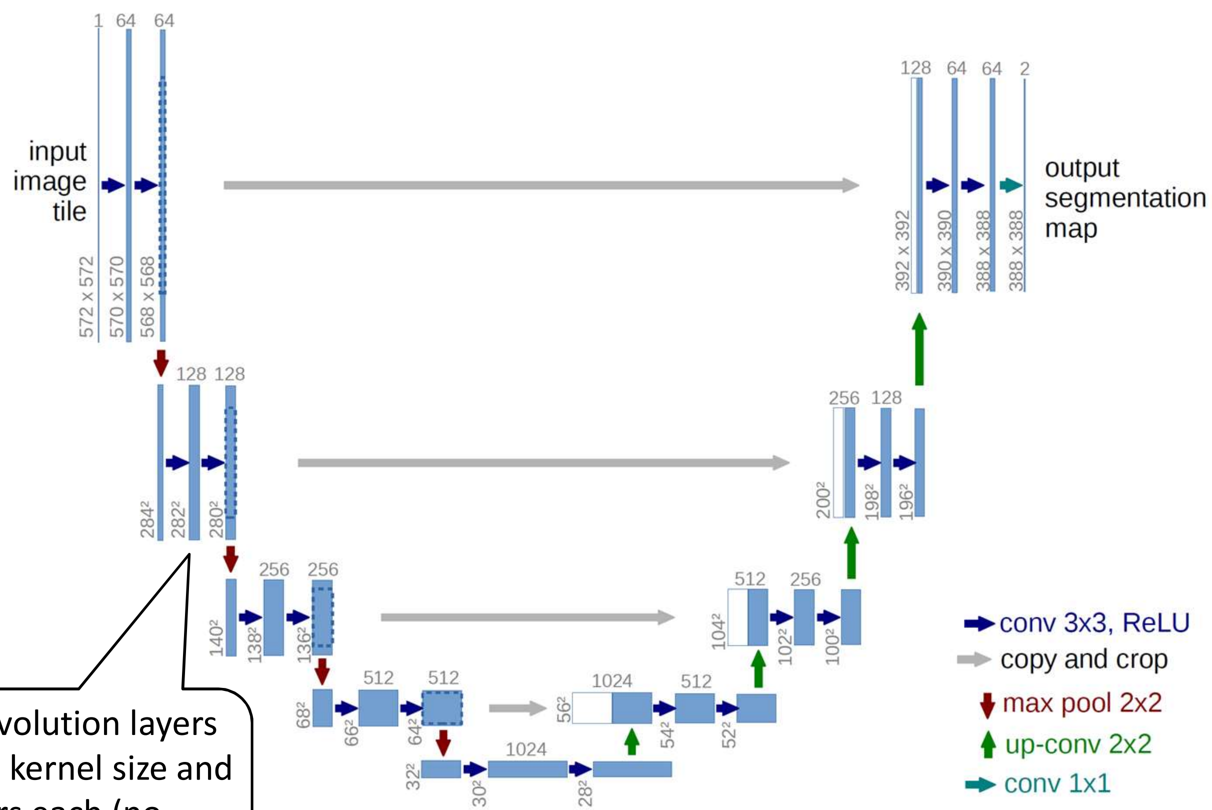
The U-Net Architecture for Segmentation

- U-Net architecture for segmentation [Ronneberger et al., 2015]: downsampling, upsampling, and direct connections between lower and higher layers



The U-Net Architecture for Segmentation

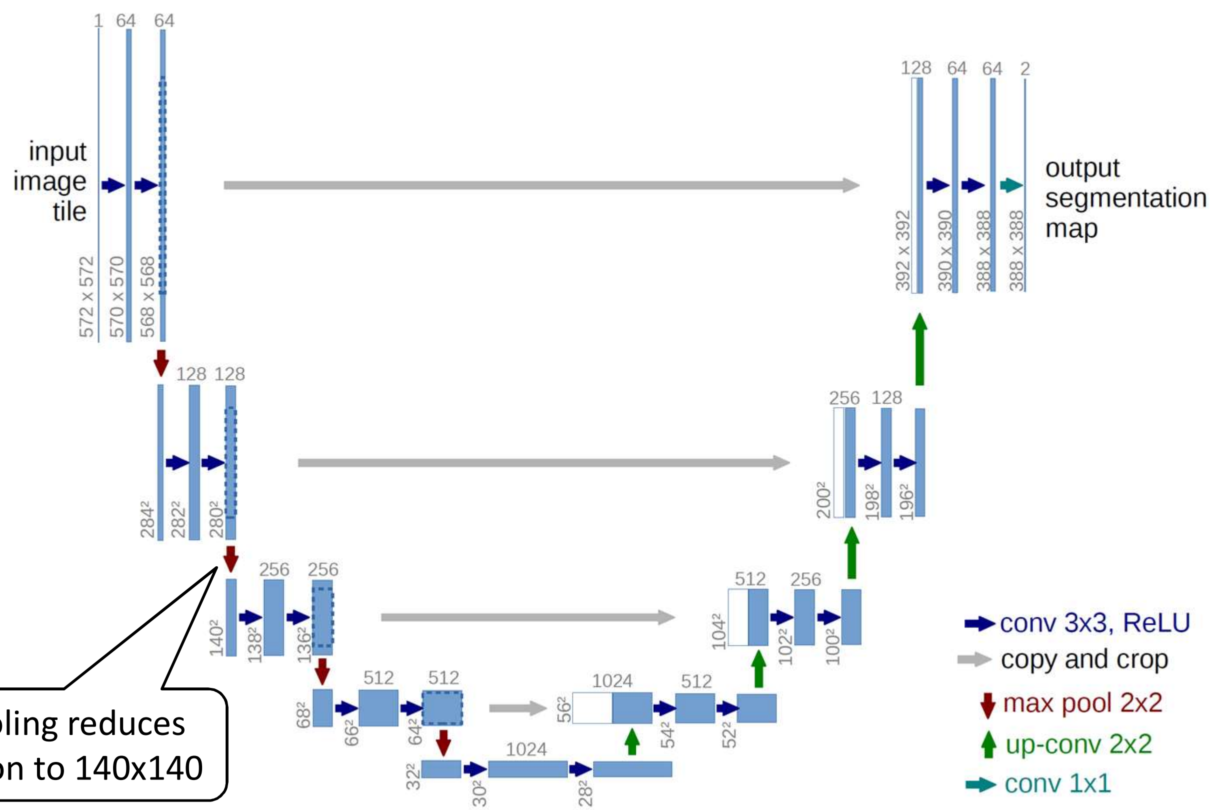
- U-Net architecture for segmentation [Ronneberger et al., 2015]: downsampling, upsampling, and direct connections between lower and higher layers



Two convolution layers with 3x3 kernel size and 128 filters each (no padding)

The U-Net Architecture for Segmentation

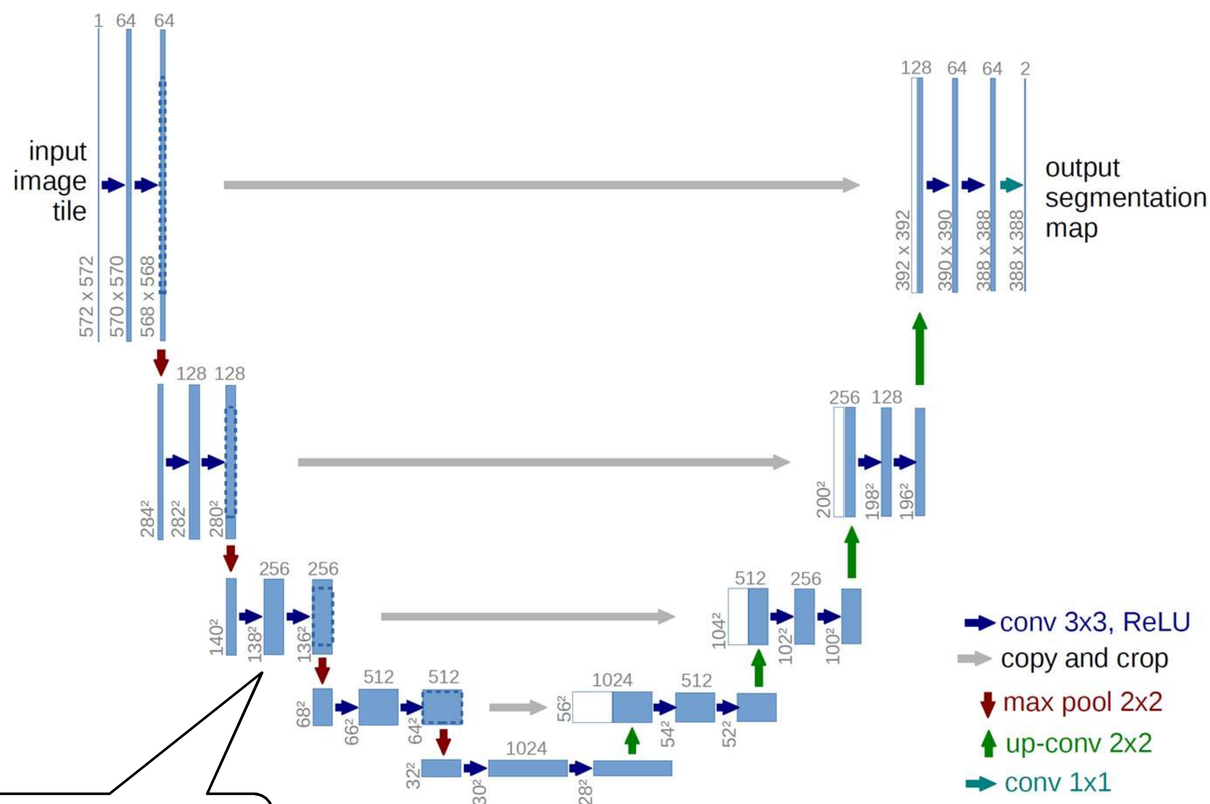
- U-Net architecture for segmentation [Ronneberger et al., 2015]: downsampling, upsampling, and direct connections between lower and higher layers



Max pooling reduces resolution to 140x140

The U-Net Architecture for Segmentation

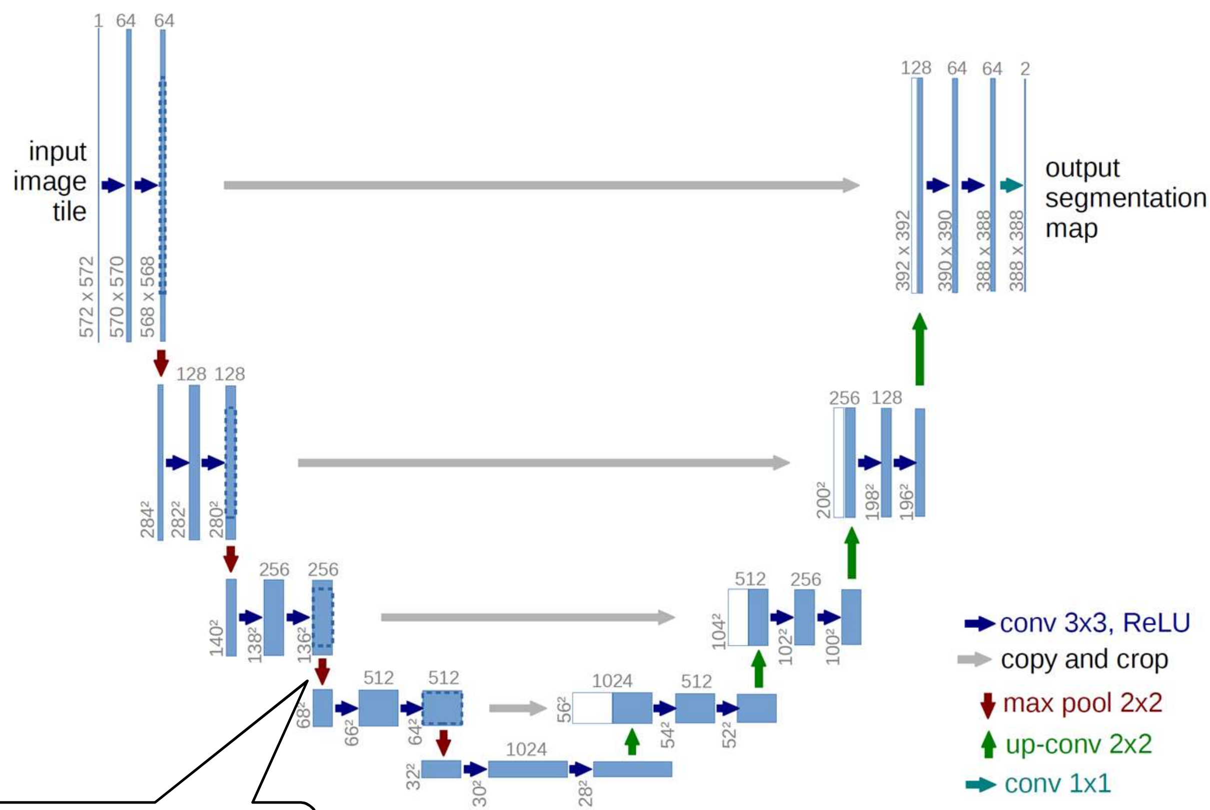
- U-Net architecture for segmentation [Ronneberger et al., 2015]: downsampling, upsampling, and direct connections between lower and higher layers



Two convolution layers with 256 filters each

The U-Net Architecture for Segmentation

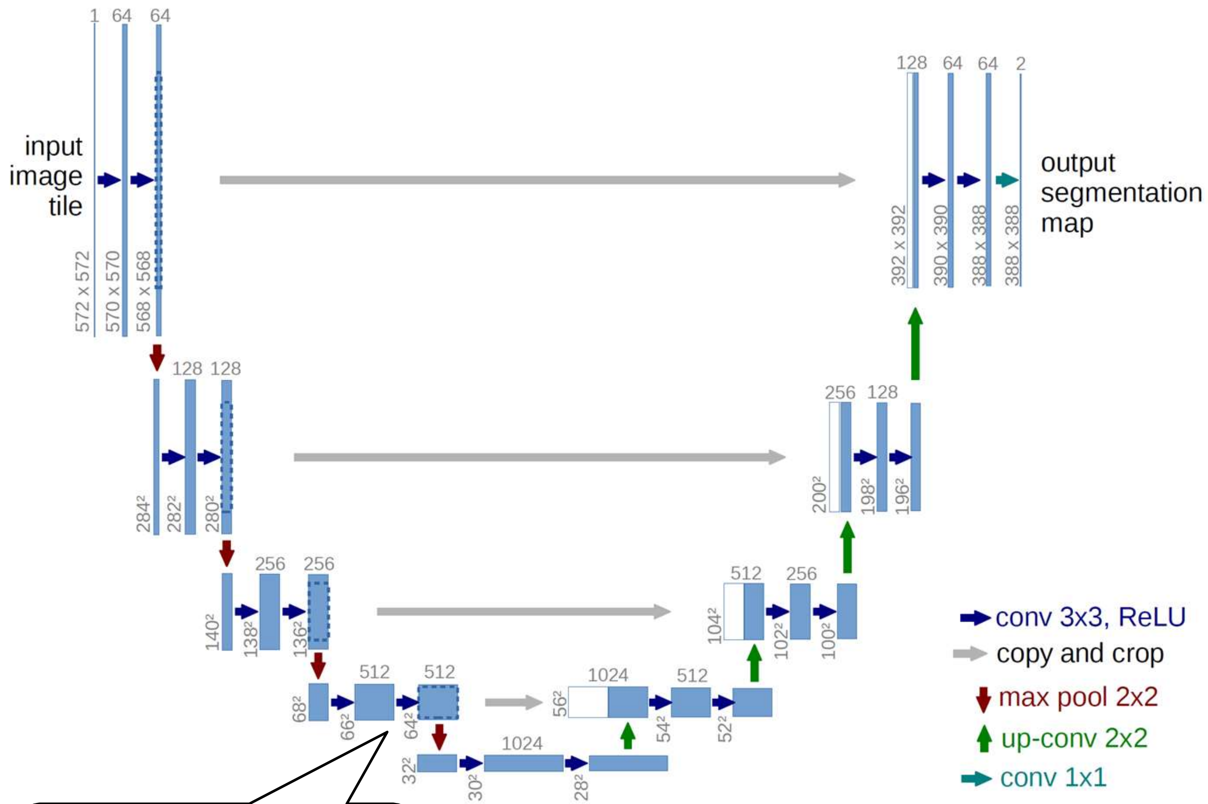
- U-Net architecture for segmentation [Ronneberger et al., 2015]: downsampling, upsampling, and direct connections between lower and higher layers



Max pooling reduces resolution to 68x68

The U-Net Architecture for Segmentation

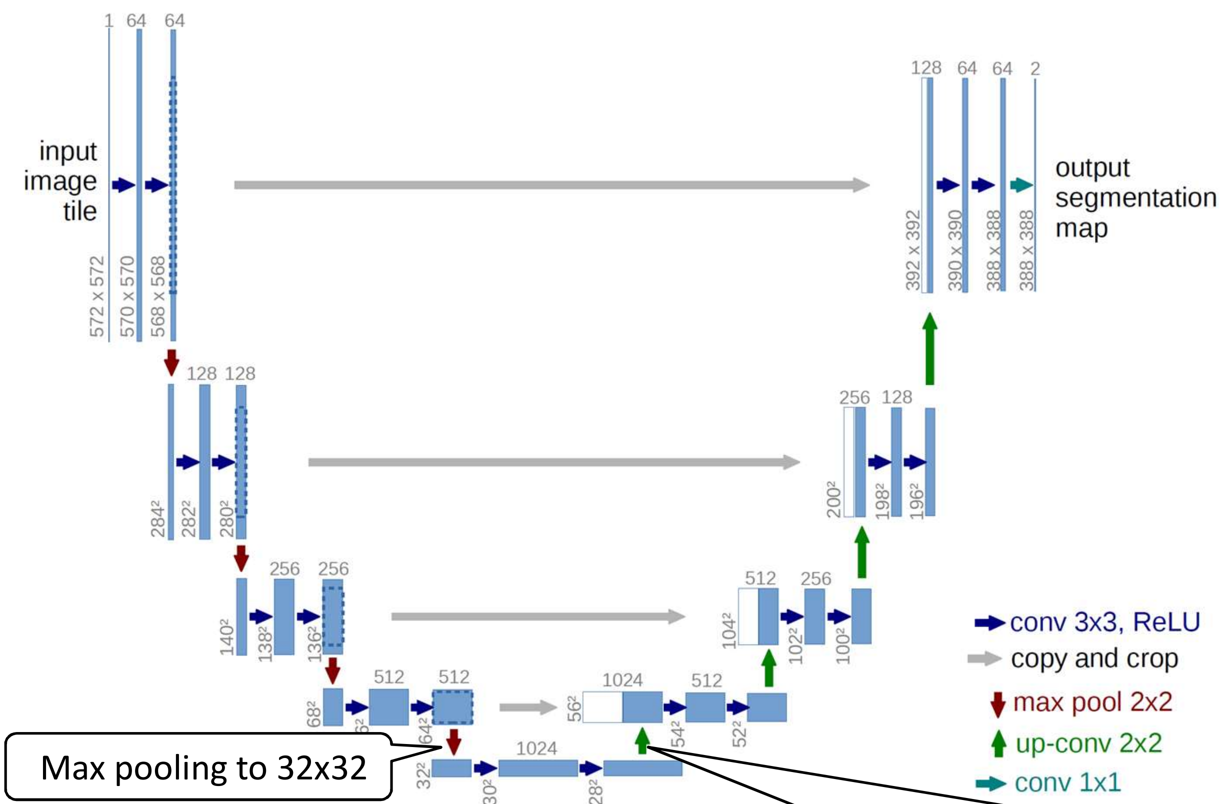
- U-Net architecture for segmentation [Ronneberger et al., 2015]: downsampling, upsampling, and direct connections between lower and higher layers



Two convolution layers
with 512 filters each

The U-Net Architecture for Segmentation

- U-Net architecture for segmentation [Ronneberger et al., 2015]: downsampling, upsampling, and direct connections between lower and higher layers



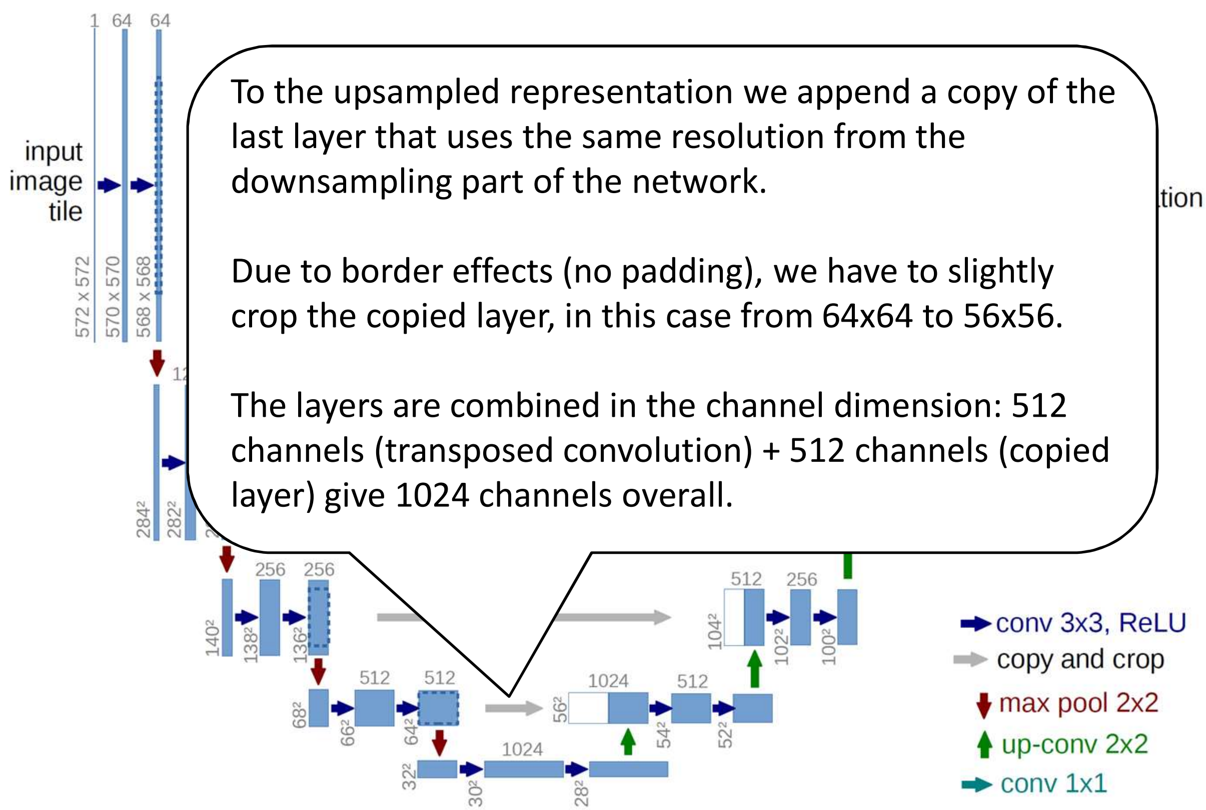
Max pooling to 32x32

Two convolution layers with 1024 filters each

Transposed convolution with 2x2 kernel, stride 2x2, 512 filters, and padding such that resolution is exactly doubled

The U-Net Architecture for Segmentation

- U-Net architecture for segmentation [Ronneberger et al., 2015]: downsampling, upsampling, and direct connections between lower and higher layers



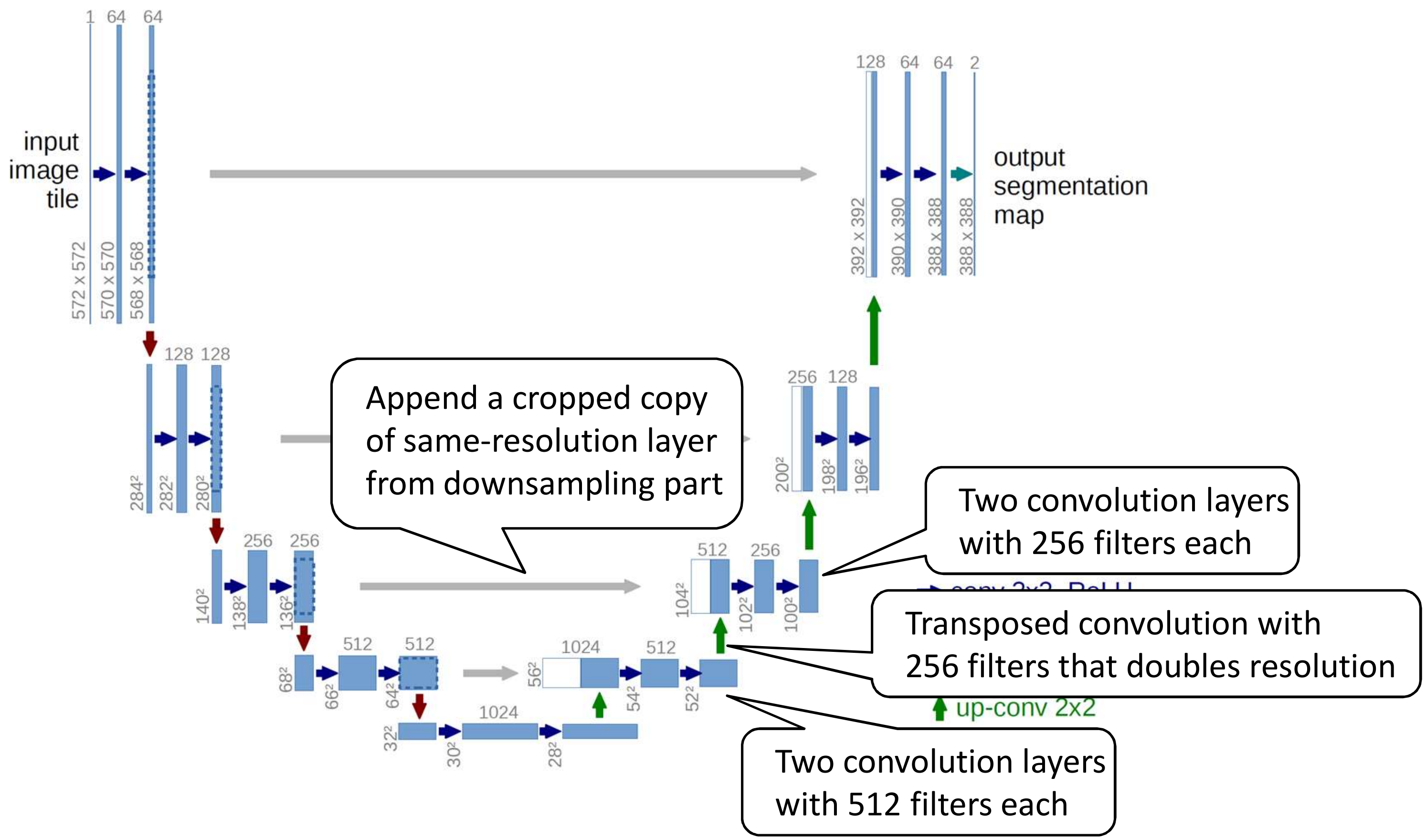
To the upsampled representation we append a copy of the last layer that uses the same resolution from the downsampling part of the network.

Due to border effects (no padding), we have to slightly crop the copied layer, in this case from 64x64 to 56x56.

The layers are combined in the channel dimension: 512 channels (transposed convolution) + 512 channels (copied layer) give 1024 channels overall.

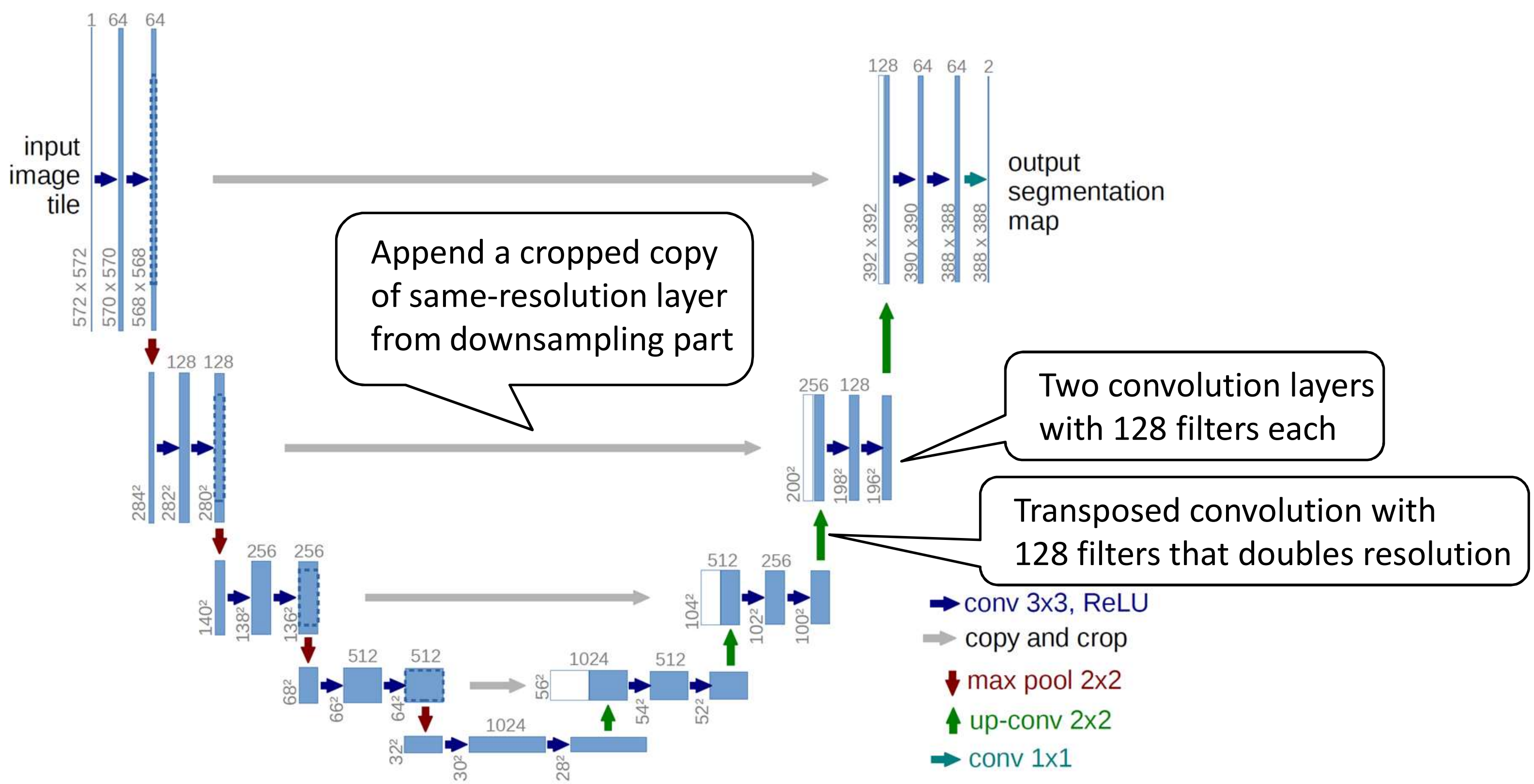
The U-Net Architecture for Segmentation

- U-Net architecture for segmentation [Ronneberger et al., 2015]: downsampling, upsampling, and direct connections between lower and higher layers



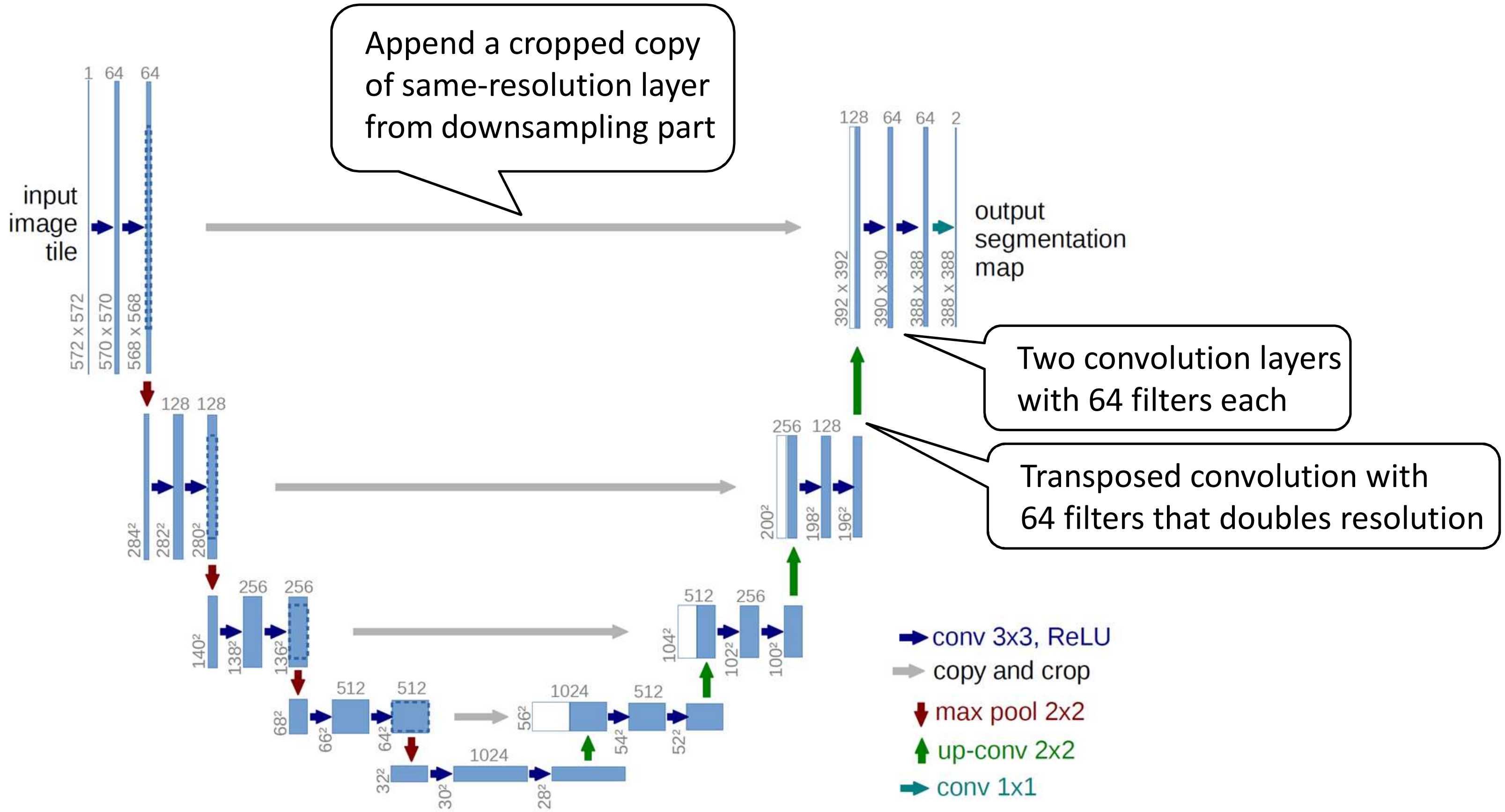
The U-Net Architecture for Segmentation

- U-Net architecture for segmentation [Ronneberger et al., 2015]: downsampling, upsampling, and direct connections between lower and higher layers



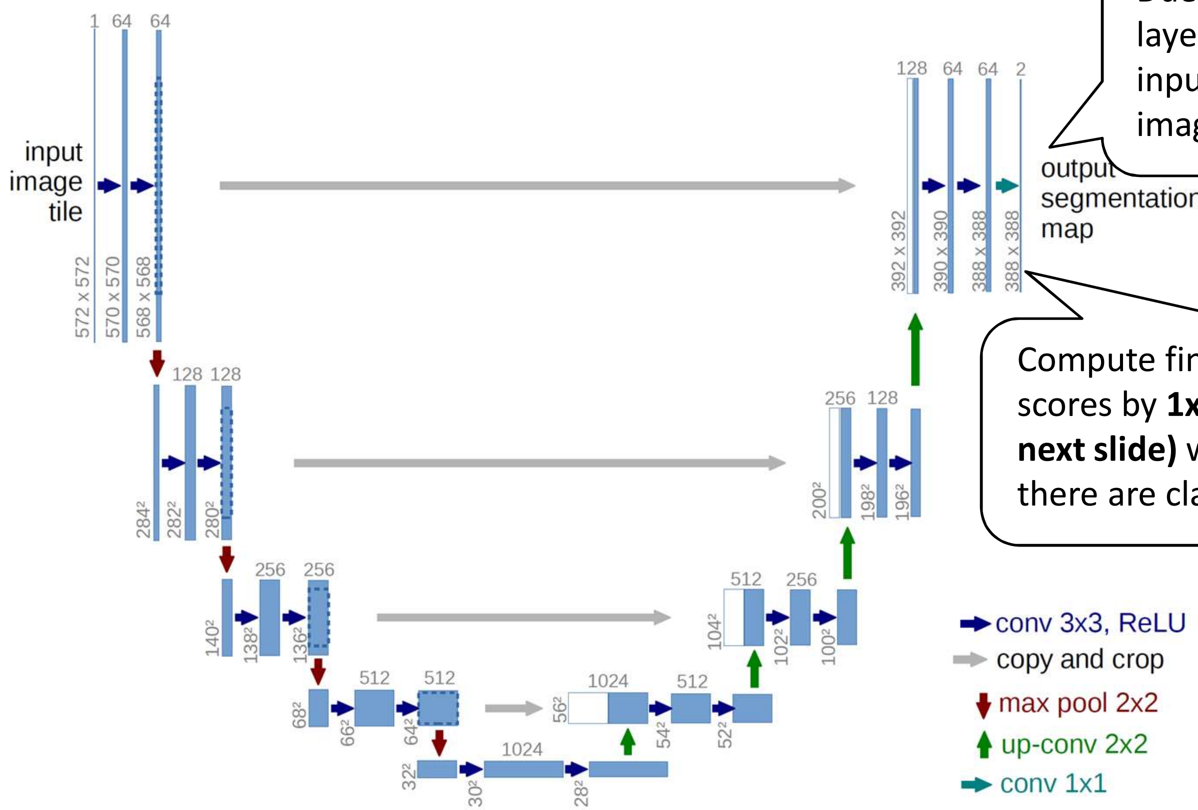
The U-Net Architecture for Segmentation

- U-Net architecture for segmentation [Ronneberger et al., 2015]: downsampling, upsampling, and direct connections between lower and higher layers



The U-Net Architecture for Segmentation

- U-Net architecture for segmentation [Ronneberger et al., 2015]: downsampling, upsampling, and direct connections between lower and higher layers



Due to non-padded convolution layers, output is smaller than input. Only this inner part of image is annotated.

Compute final output map of class scores by **1x1 convolution layer (see next slide)** with as many filters as there are classes (two in this case)

Special Case 1x1 Convolution

- Reminder: A frequently used special case of the standard convolution operation is a convolution with a 1x1 kernel size (and stride 1)

Example with three input channels

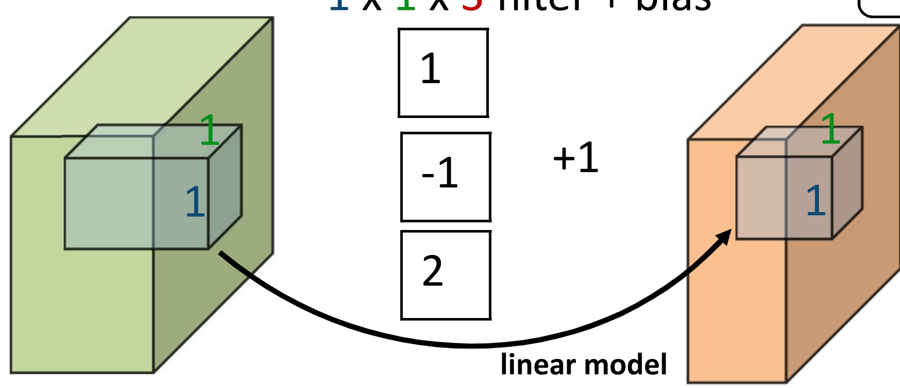
1	-2	2
-1	0	1
0	2	1

2	-1	1
1	1	0
-1	0	1

-2	1	0
0	1	2
-1	-1	1

3 x 3 x 3 input

1 x 1 x 3 filter + bias



3 x 3 x 1 output (one channel)

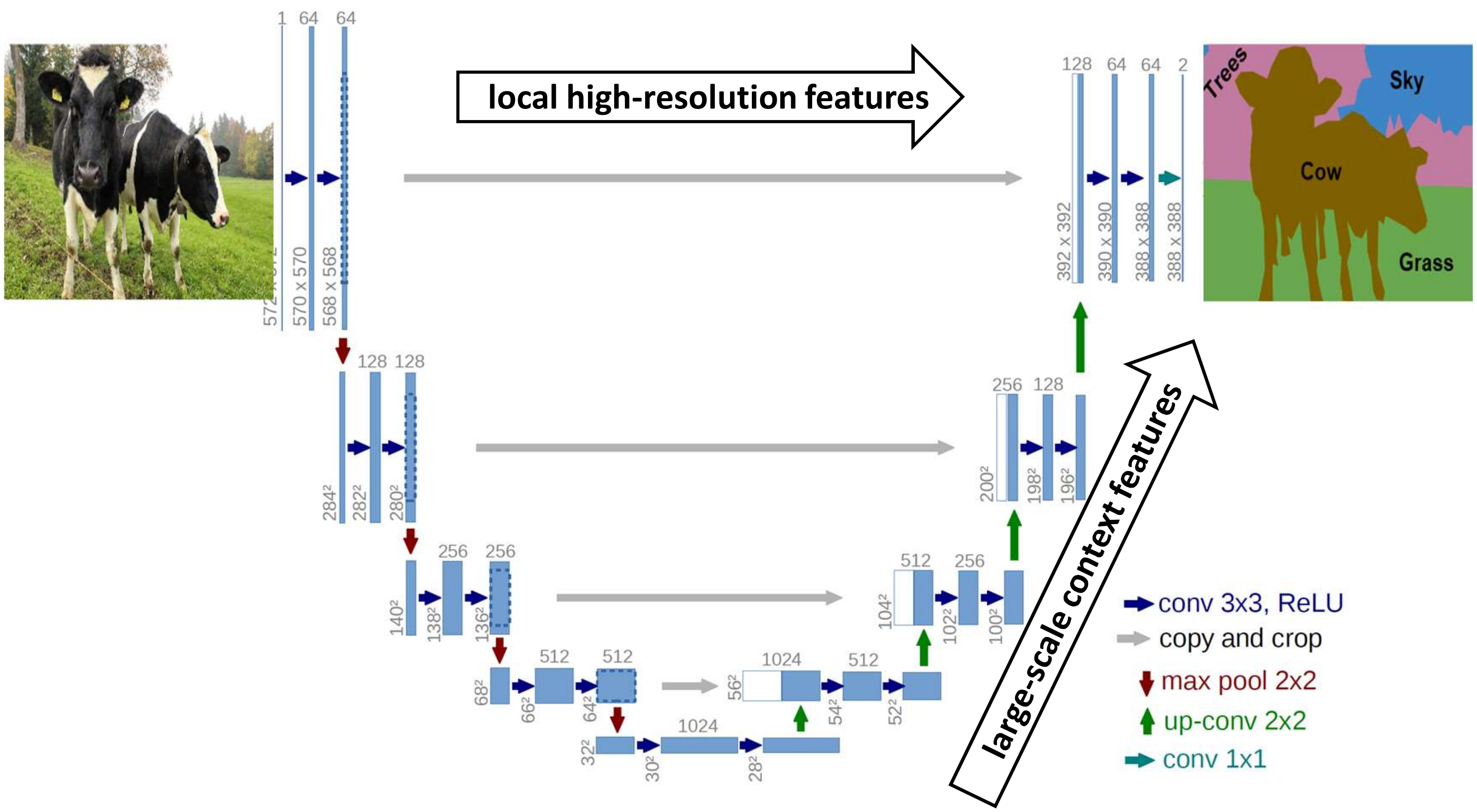
$$1 \cdot 1 + (-1) \cdot 2 + 2 \cdot (-2) + 1 = -4$$

-4	2	2
-1	2	6
0	1	3

Spatial size of output is unchanged, number of channels can be increased or reduced.
Acts like a linear model in the channel dimension, applied independently to every x/y position

The U-Net Architecture for Segmentation

- The „copy“-pathes in UNet combine the information from the higher-resolution representations in the original image and the early layers with the more large-scale context from the intermediate layers



U-Net Loss Function for Segmentation

- Model is trained on annotated images, that is, pairs of images and class maps: $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}, \mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$ with $\mathbf{x}_i \in \mathbb{R}^{m \times l \times d}$, $\mathbf{y}_i \in \mathbb{R}^{m \times l}$

- Let $\mathbf{s}_i \in \mathbb{R}^{m \times l \times k}$ denote the final output tensor of the model (class scores for all pixels in output) for input $\mathbf{x}_i$.
- For each position x, y: compute a predicted class distribution by

Probability for class j according to model

$$p(\mathbf{y}[x,y] = j \mid \mathbf{x}_i, \boldsymbol{\theta}) = \frac{\exp(\mathbf{s}_i[x,y,j])}{\sum_{j'=1}^{k} \exp(\mathbf{s}_i[x,y,j'])}$$

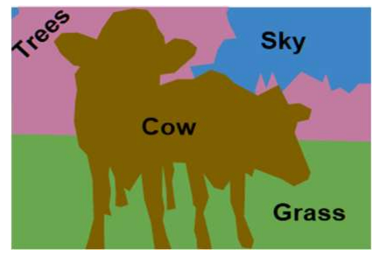
All model parameters

- Loss for an image is cross entropy summed over positions in output

$$\ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i) = -\sum_{x,y} \log p(\mathbf{y}[x,y] = \mathbf{y}_i[x,y] \mid \mathbf{x}_i, \boldsymbol{\theta})$$

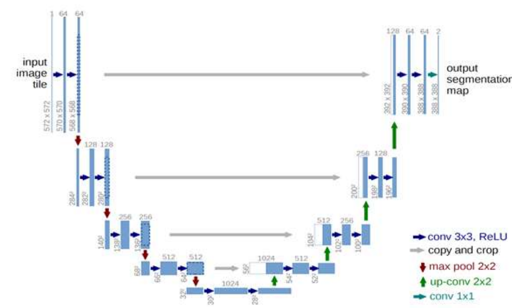
Summary: Segmentation, U-Net Architecture

- Segmentation problem:



{Trees, Sky, Cow, Grass, ...}

- U-Net Architecture: downsampling for large-scale context, upsampling for final segmentation map, „copy“-connections to reuse local features



- As usual, gradient of entire model can be derived via automatic differentiation, and model can be trained end-to-end by stochastic gradient descent