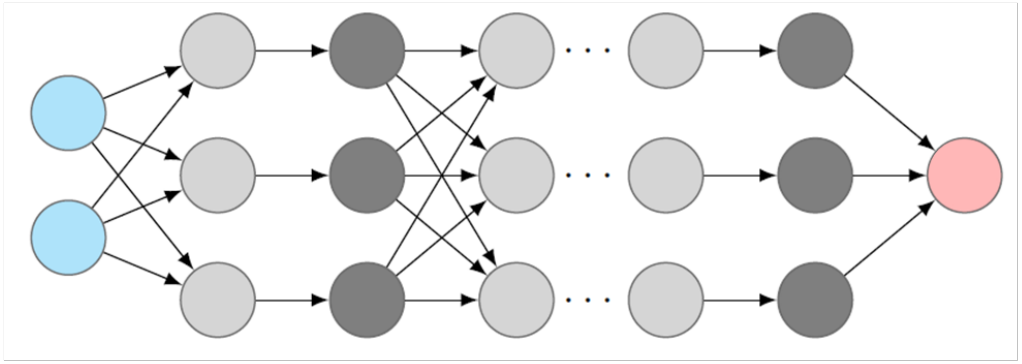
Convolutional Neural Networks

- Revisiting convolutions and their adjoint operators -

Lecturer: Michael Möller – michael.moeller@uni-siegen.de
Exercises: Hartmut Bauermeister – hartmut.bauermeister@uni-siegen.de



Remember Convolutions

By using different numbers of filters you can steer the number of output channel.

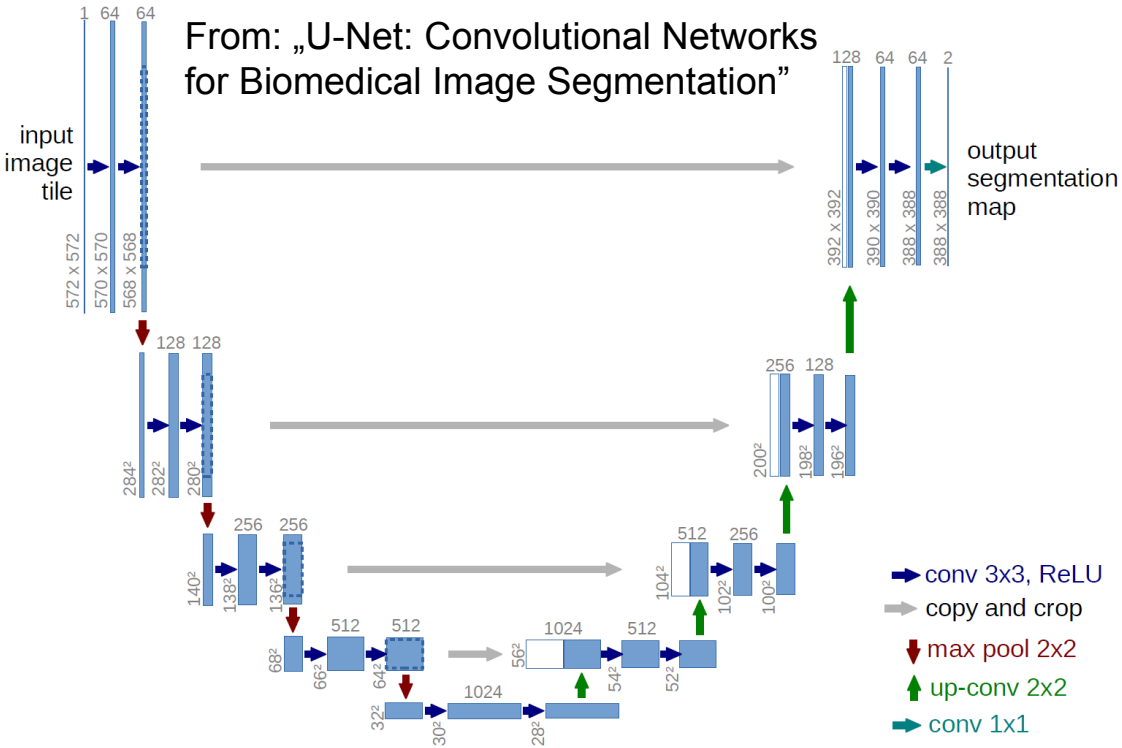
By using stride you can reduce the spatial resolution.

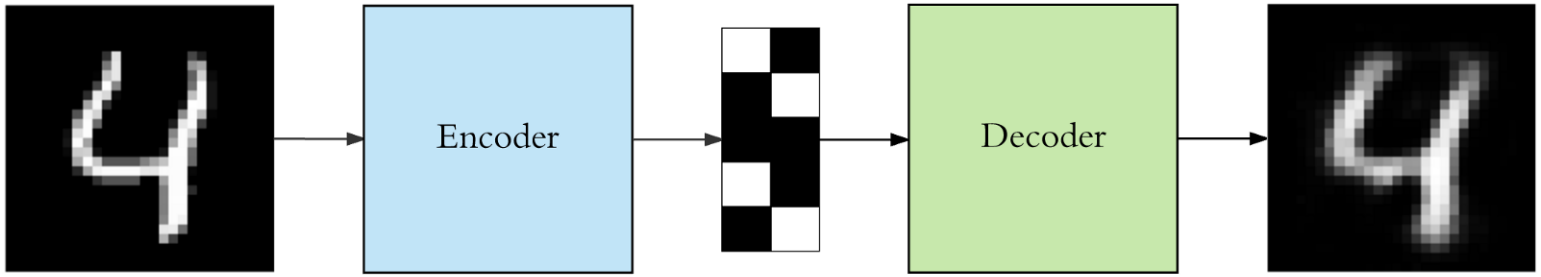
Animation taken from https://sites.google.com/site/nttrungmtwiki/home/it/data-science---python/tensorflow/tensorflow-and-deep-learning-part-3

Increasing the resolution?

But I have briefly shown you architectures that seem to increase the spatial resolution of images with suitable layers.

So far a fully connected layer is the only option for increasing the size that we have seen in the lecture.





From: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Some math

One can define an scalar product on arbitrary 2d or 3d images / tensors:

For $X \in \mathbb{R}^{n_y \times n_x \times n_c}$ and $Y \in \mathbb{R}^{n_y \times n_x \times n_c}$ one defines

$$\langle X, Y \rangle = \sum_{i=1}^{n_y} \sum_{j=1}^{n_x} \sum_{k=1}^{n_c} X_{i,j,k} Y_{i,j,k}$$

A convolution layer k (without bias) is a linear operator on such vector spaces, i.e.

$$k * (X + Y) = k * X + k * Y \qquad \text{and} \qquad k * (\alpha X) = \alpha k * X$$

and meets that for $X \in \mathbb{R}^{n_{y_1} \times n_{x_1} \times n_{c_1}}$, one obtains $k * X \in \mathbb{R}^{n_{y_2} \times n_{x_2} \times n_{c_2}}$

A famous result in math, the *Riesz-Representation Theorem*, now states that there exists an *adjoint* linear operator A^* such that

$$\langle k * X, Y \rangle = \langle X, A^*(Y) \rangle \quad \forall\ X,\ Y$$

Now for $Y \in \mathbb{R}^{n_{y_2} \times n_{x_2} \times n_{c_2}}$, one obtains $A^*Y \in \mathbb{R}^{n_{y_1} \times n_{x_1} \times n_{c_1}}$

Idea: We can use the *adjoint* of a convolution to increase the spatial resolution of our images! This operation is known as a *transpose convolution* or *deconvolution*.

… a little simpler …

A convolution (even with stride) can be written as a matrix vector multiplication

f

1	0	-2	3	1	2	-1

input signal

k

0.5	1	0.5

convolution kernel

-0.5	2.5	2

Valid convolution with stride 2

$$\begin{pmatrix} 0.5 & 1 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 1 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 1 & 0.5 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ -2 \\ 3 \\ 1 \\ 2 \\ -1 \end{pmatrix} = \begin{pmatrix} -0.5 \\ 2.5 \\ 2 \end{pmatrix}$$

The transpose of this matrix maps a vector with three entries to a vector with seven entries!

Higher dimensional again

The exact same thing can be done for convolutional layers that map 3d images to 3d images! By vectorising the images, the convolutional layer (without bias) can be written as a matrix multiplication!

The transpose of this matrix reverses the modifications of the image size, hence the name *transpose convolution* (although adjoint is the better word mathematically). Computationally, you of course never form the actual matrix.

Did you ever wonder how to compute the backpropagation through a convolution layer? The answer is applying the adjoint operator (similar to how we multiply with the transpose matrix in the case of fully connected layers)!