

# Lecture 14b: Segmentation and autoencoders

Machine Learning and Imaging

BME 590L  
Roarke Horstmeyer

# Other Computer Vision Tasks

## Semantic Segmentation



**GRASS, CAT,  
TREE, SKY**

No objects, just pixels

Balanced equation

## Classification + Localization



**CAT**

Single Object

Over-determined

## Object Detection



**DOG, DOG, CAT**

Multiple Object

Over-determined

## Instance Segmentation

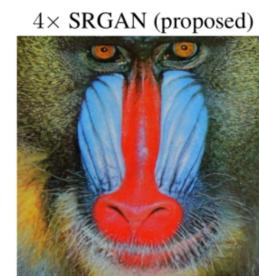


**DOG, DOG, CAT**

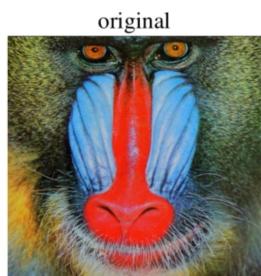
This image is CC0 public domain

Over-determined

## Super-resolution



4× SRGAN (proposed)



original

Figure 1: Super-resolved image (left) is almost indistinguishable from original (right). [4× upscaling]

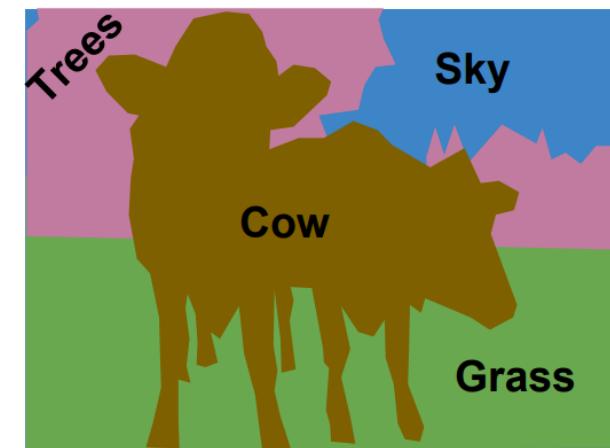
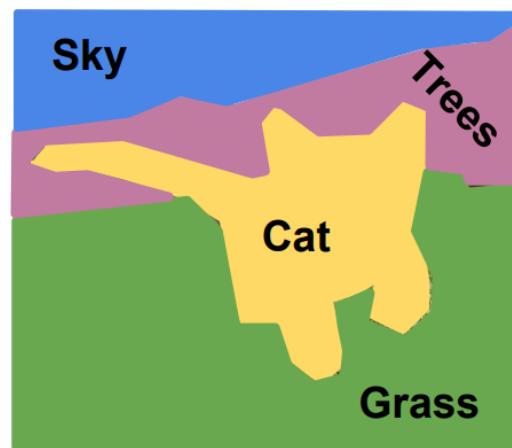
# Semantic Segmentation

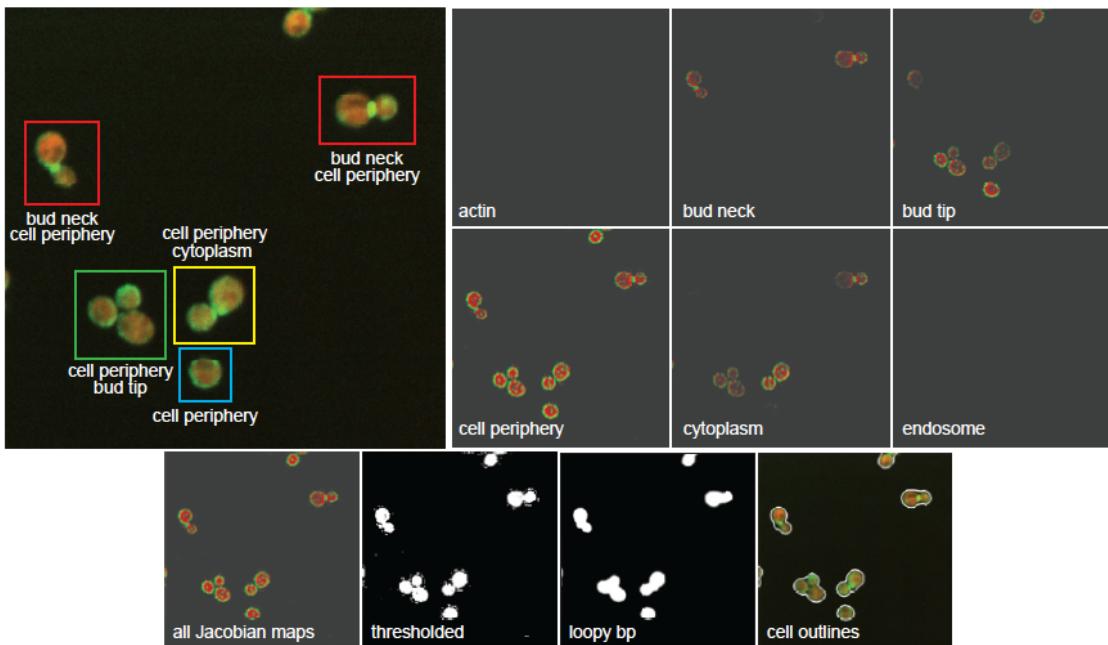
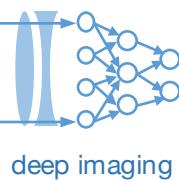
Label each pixel in the image with a category label

Don't differentiate instances, only care about pixels

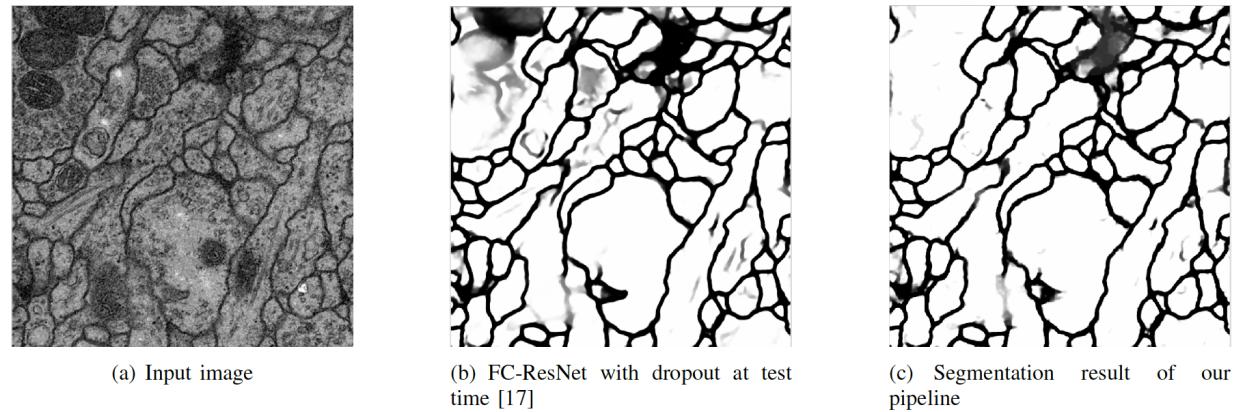


This image is CC0 public domain

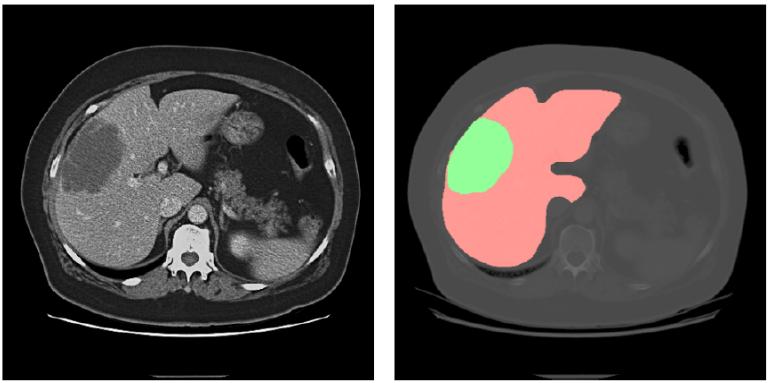
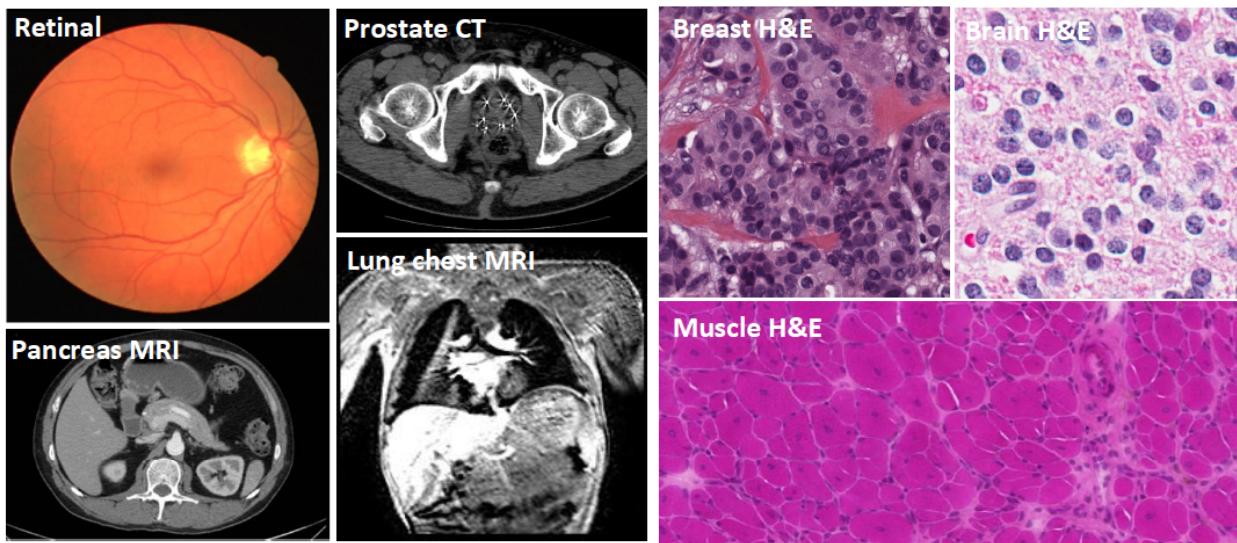




Oren Z. Kraus et al., "Classifying and Segmenting Microscopy Images Using Convolutional Multiple Instance Learning," arXiv 2015



Other possible examples:

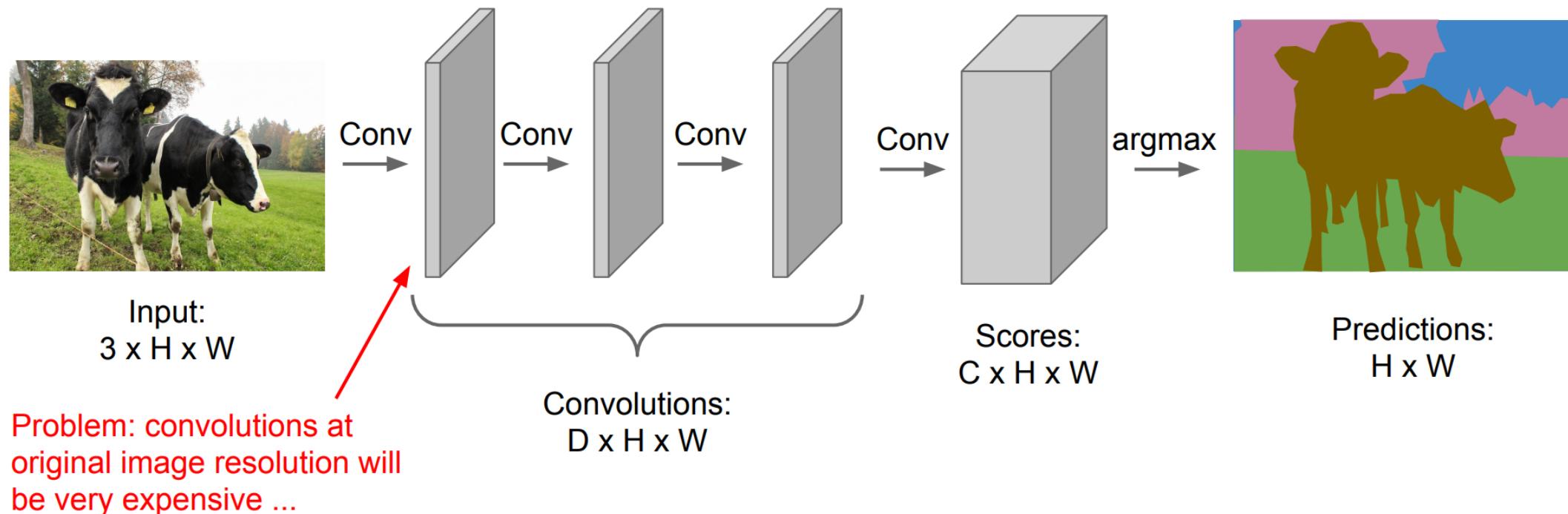


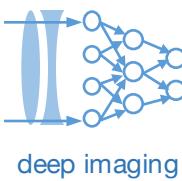
M. Drozdzal et al., Learning Normalized Inputs for Iterative Estimation in Medical Image Segmentation (2017)

Z. Zhang et al., Recent Advances in the Applications of Convolutional Neural Networks to Medical Image Contour Detection (2017)

# Semantic Segmentation Idea: Fully Convolutional ?

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

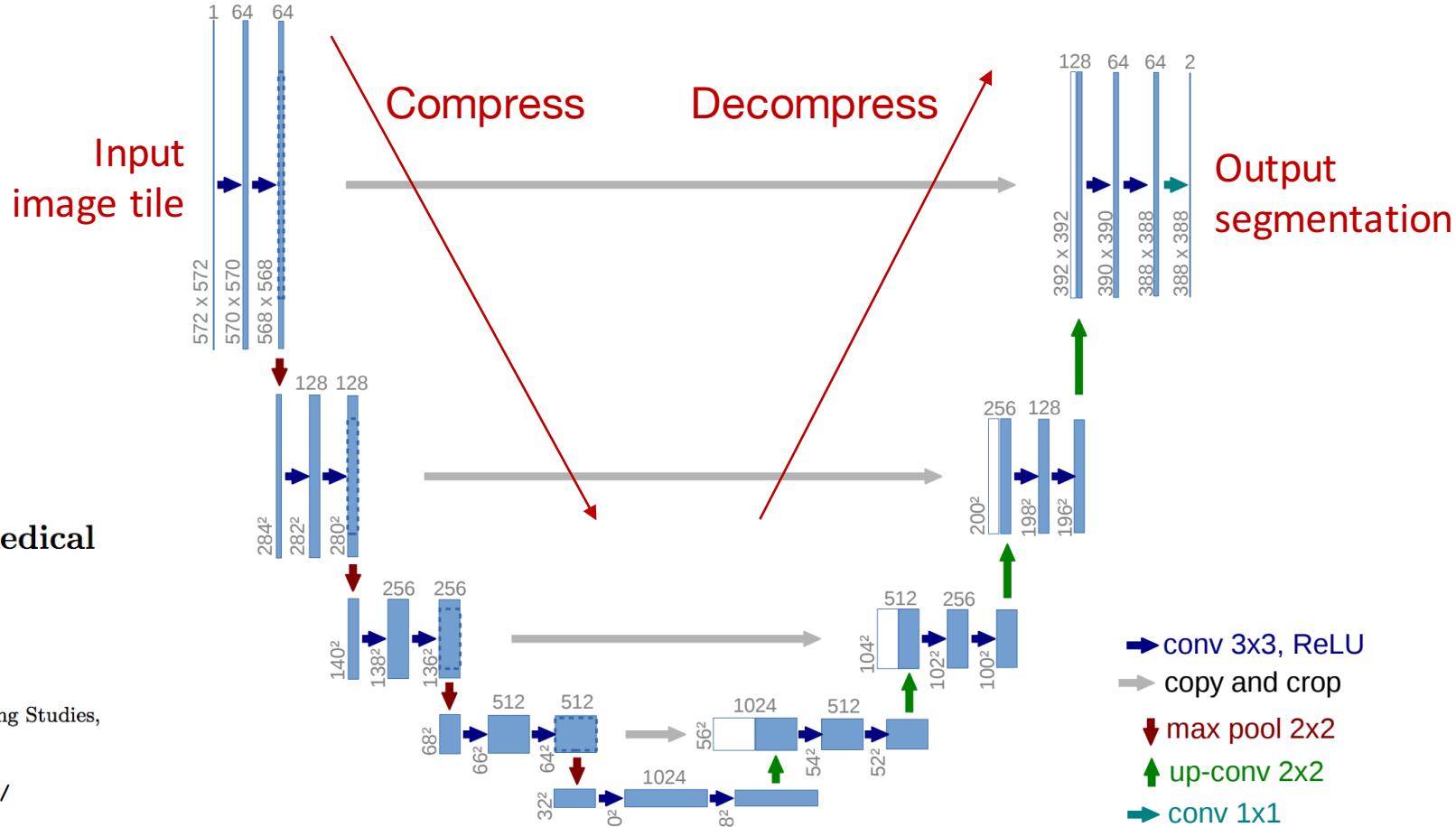




## Instead, compress x-y dimensions of input image

- Compress spatial features into learned filters
- Then, decompress learned filters back into same spatial dimensions

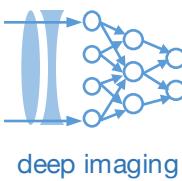
### U-Net Architecture



### U-Net: Convolutional Networks for Biomedical Image Segmentation

Olaf Ronneberger, Philipp Fischer, and Thomas Brox

Computer Science Department and BIOSS Centre for Biological Signalling Studies,  
University of Freiburg, Germany  
[ronneber@informatik.uni-freiburg.de](mailto:ronneber@informatik.uni-freiburg.de),  
WWW home page: <http://lmb.informatik.uni-freiburg.de/>



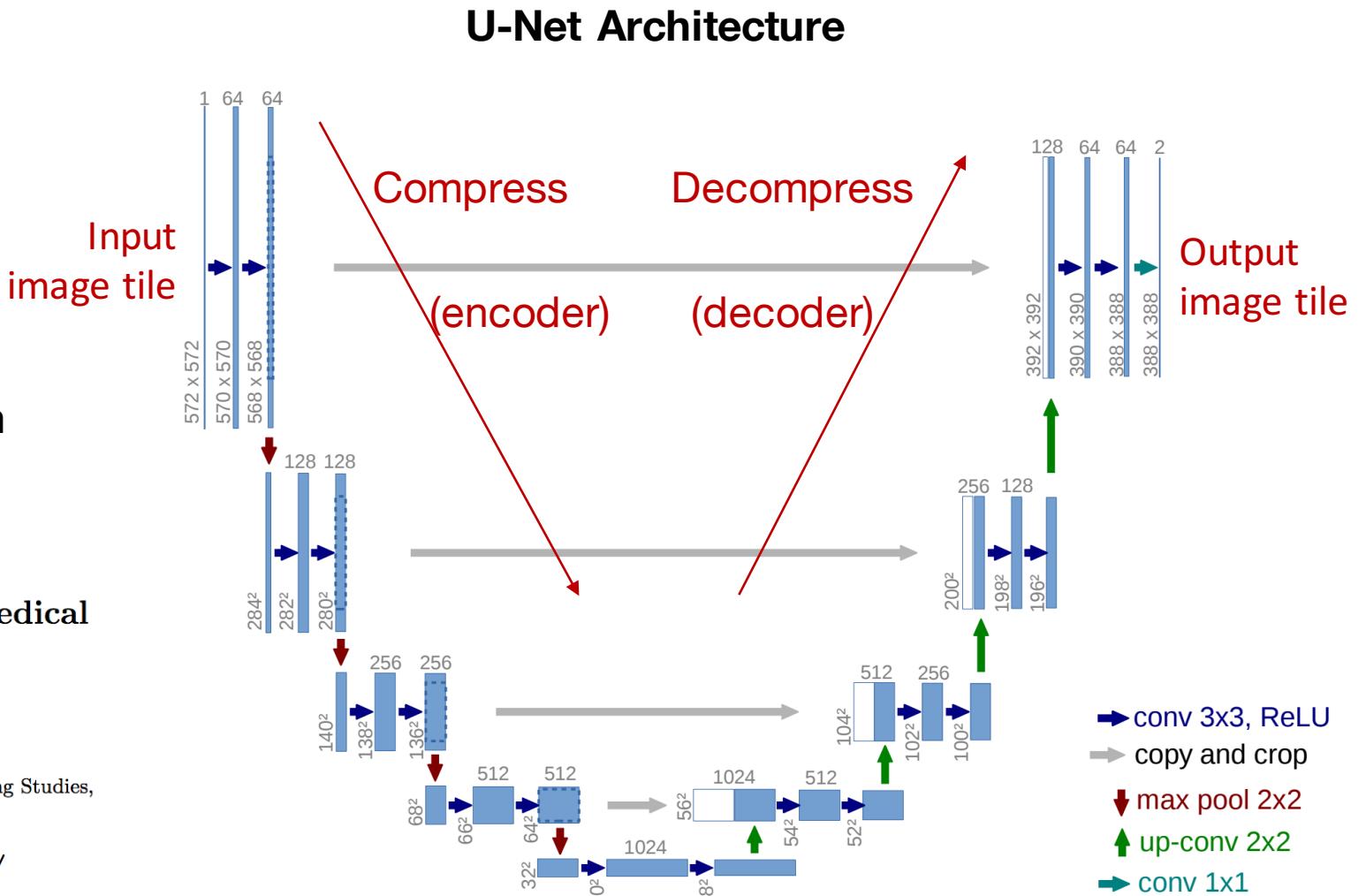
# Instead, compress x-y dimensions of input image

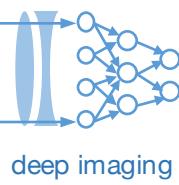
- Compress spatial features into learned filters
- Then, decompress learned filters back into same spatial dimensions
- **Can be an autoencoder**
- Analogous to image compression
- A very powerful idea...

## U-Net: Convolutional Networks for Biomedical Image Segmentation

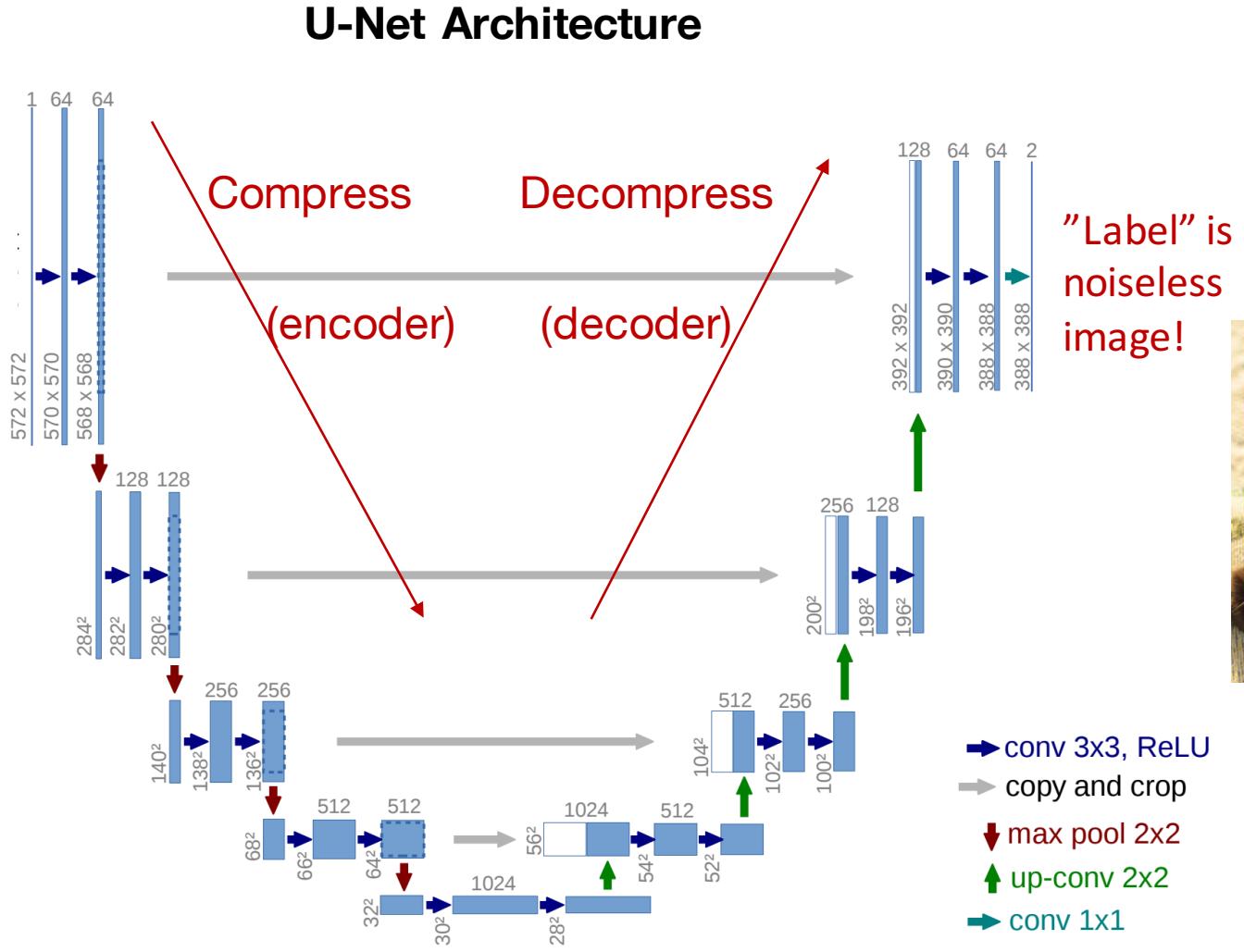
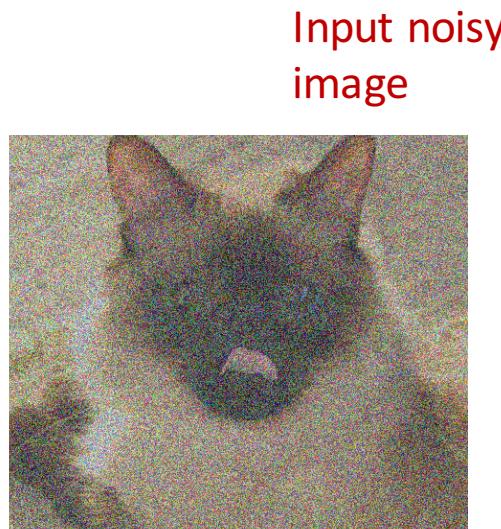
Olaf Ronneberger, Philipp Fischer, and Thomas Brox

Computer Science Department and BIOSS Centre for Biological Signalling Studies,  
University of Freiburg, Germany  
[ronneber@informatik.uni-freiburg.de](mailto:ronneber@informatik.uni-freiburg.de),  
WWW home page: <http://lmb.informatik.uni-freiburg.de/>

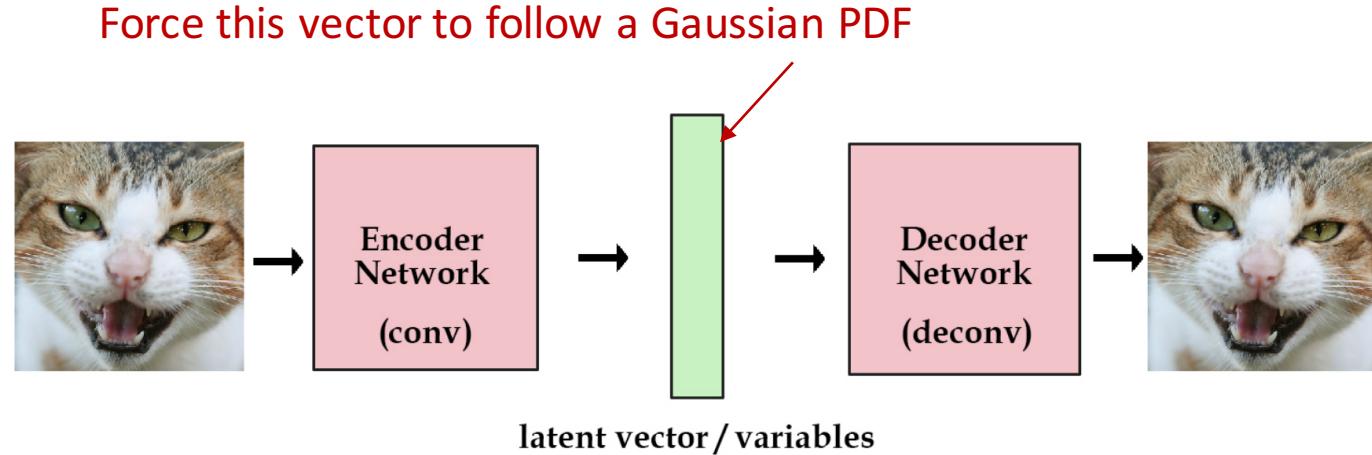




# Example: Denoising Autoencoder

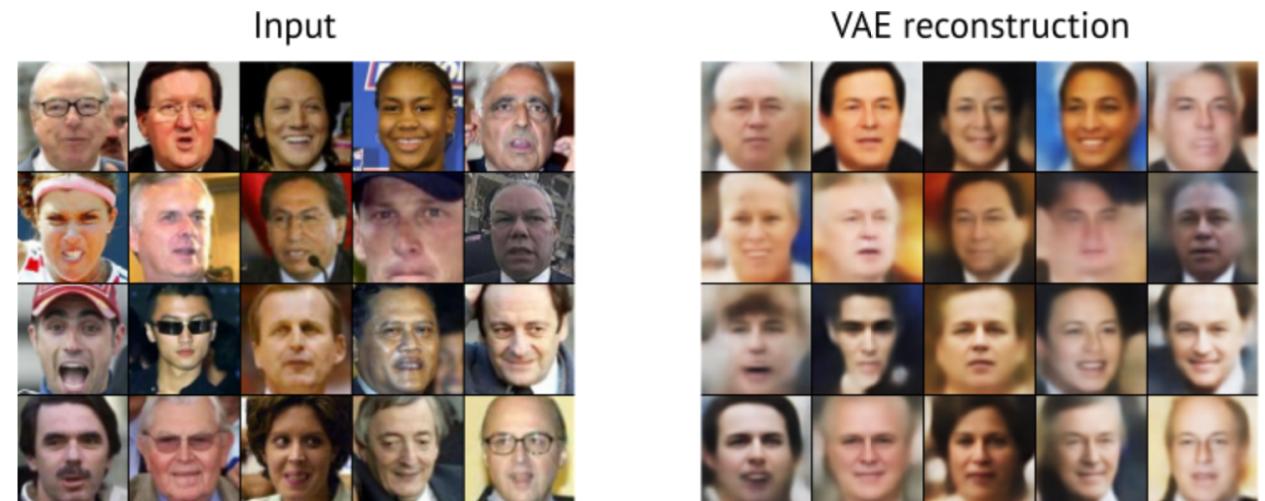


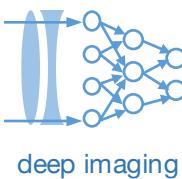
# Example: Variational Autoencoder (VAE)



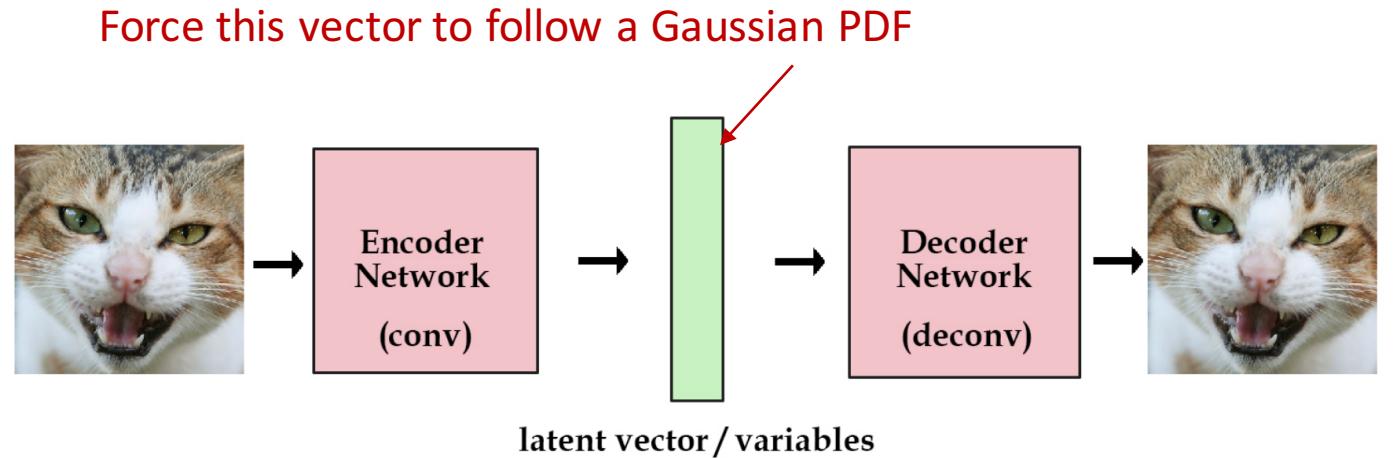
- Good *generative model*
- Have a clean probability distribution to select from to generate new examples

Minimize (KL) distance between latent vector and Gaussian normal



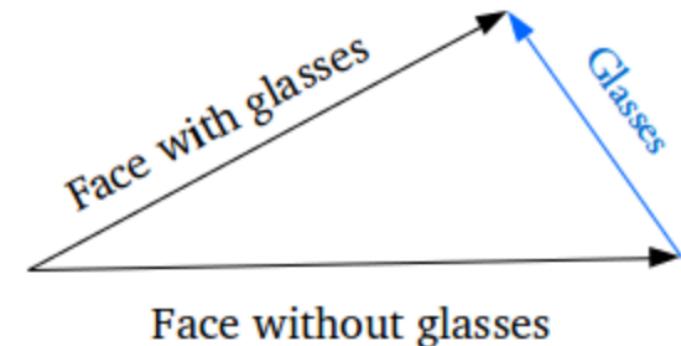


## Example: Variational Autoencoder (VAE)

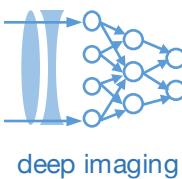


Minimize (KL) distance between latent  
vector and Gaussian normal

- With Gaussian PDF, can start to add/subtract latent vector in a normalized vector space

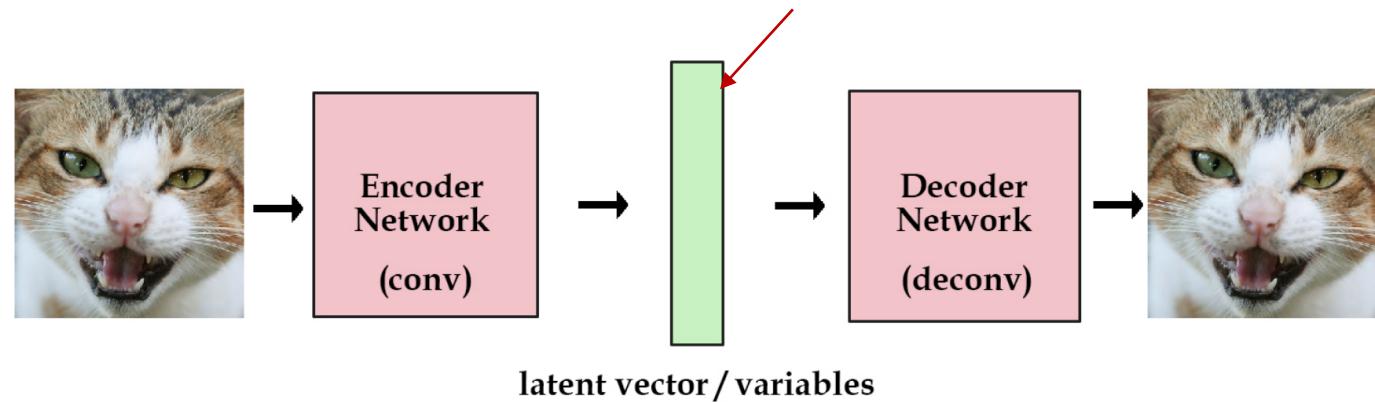


Adding new features to samples



# Example: Variational Autoencoder (VAE)

Force this vector to follow a Gaussian PDF

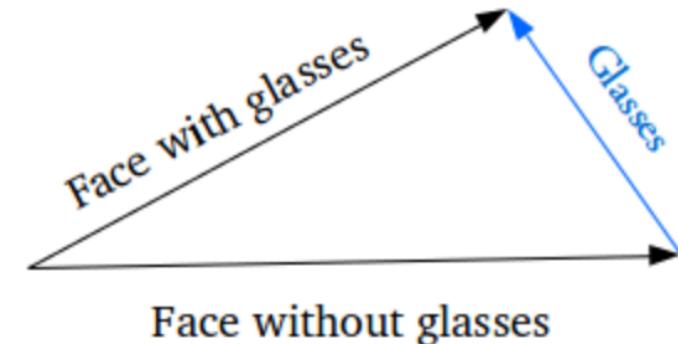


Minimize (KL) distance between latent vector and Gaussian normal

Generative Example (once trained):

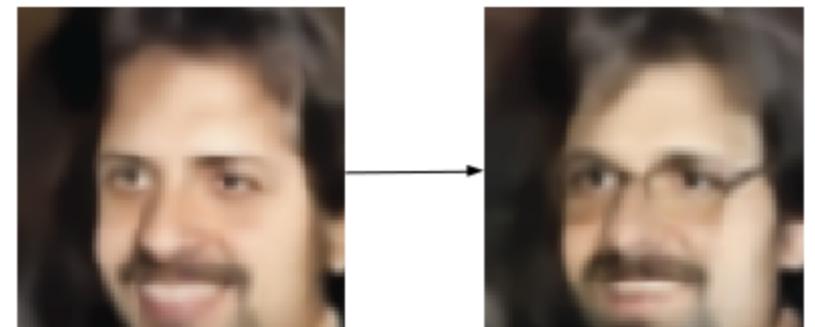
- Encode image with glasses, obtain latent vector PDF  $\mathbf{P}_g$
- Encode image without glasses, obtain PDF  $\mathbf{P}_{ng}$
- Compute  $\mathbf{diff} = \mathbf{P}_g - \mathbf{P}_{ng}$
- Encode new image to obtain  $\mathbf{P}_{new}$ , add in  $\mathbf{diff}$
- Decode  $\mathbf{P}_{new} + \mathbf{diff}$  to get guy with glasses!

- With Gaussian PDF, can start to add/subtract latent vector in a normalized vector space



Adding new features to samples

Glasses



Exploring a specific variation of input data[1]

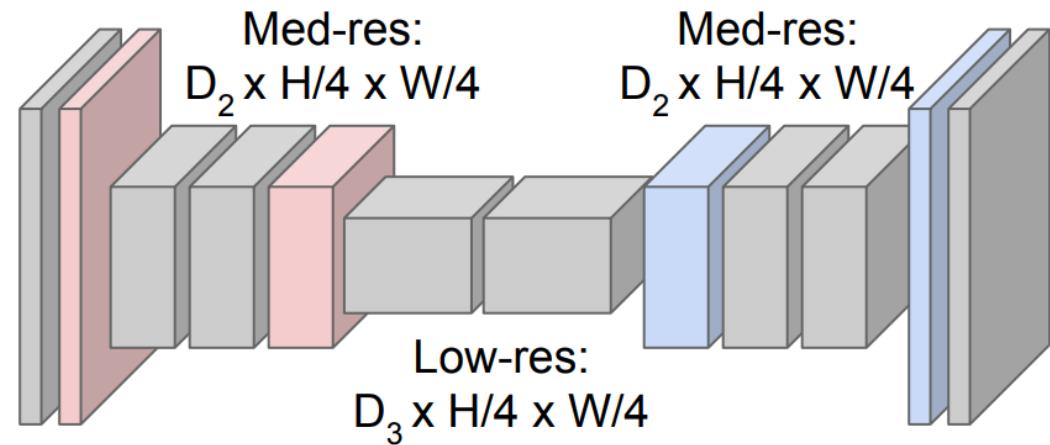
# Semantic Segmentation Idea: Fully Convolutional

**Downsampling:**  
Pooling, strided convolution

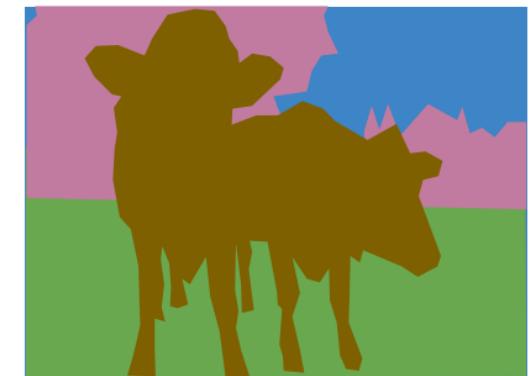


Input:  
 $3 \times H \times W$

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



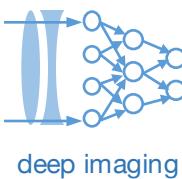
**Upsampling:**  
???



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

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

Input: 2 x 2

Output: 4 x 4

# 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

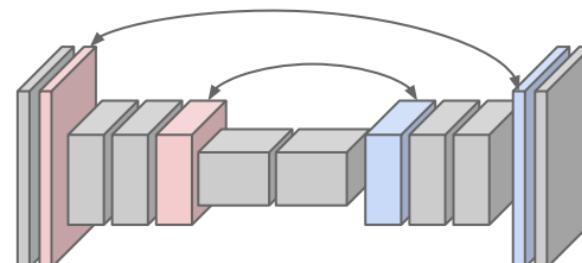
1	2
3	4

Rest of the network

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

Output: 4 x 4

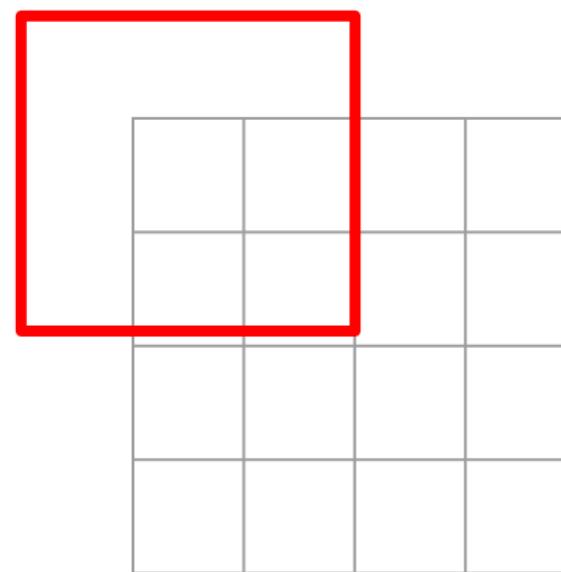
Corresponding pairs of  
downsampling and  
upsampling layers





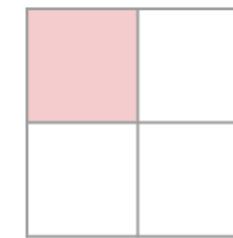
# Learnable Upsampling: Transpose Convolution

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

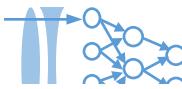


Input:  $4 \times 4$

Dot product  
between filter  
and input

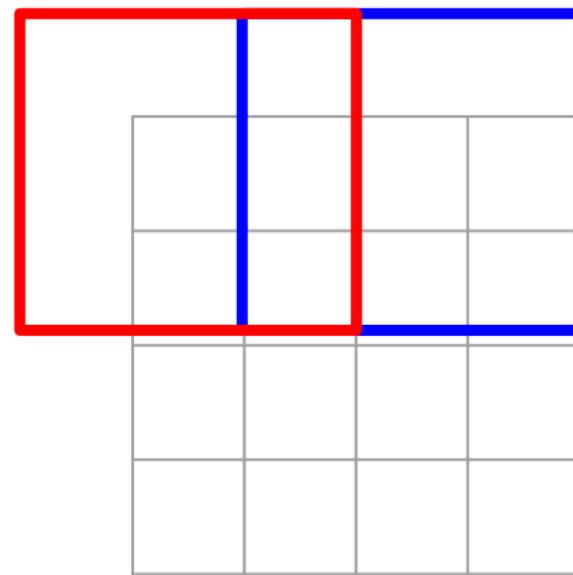


Output:  $2 \times 2$



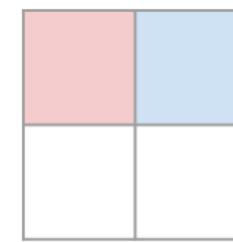
# Learnable Upsampling: Transpose Convolution

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



Input:  $4 \times 4$

Dot product  
between filter  
and input



Output:  $2 \times 2$

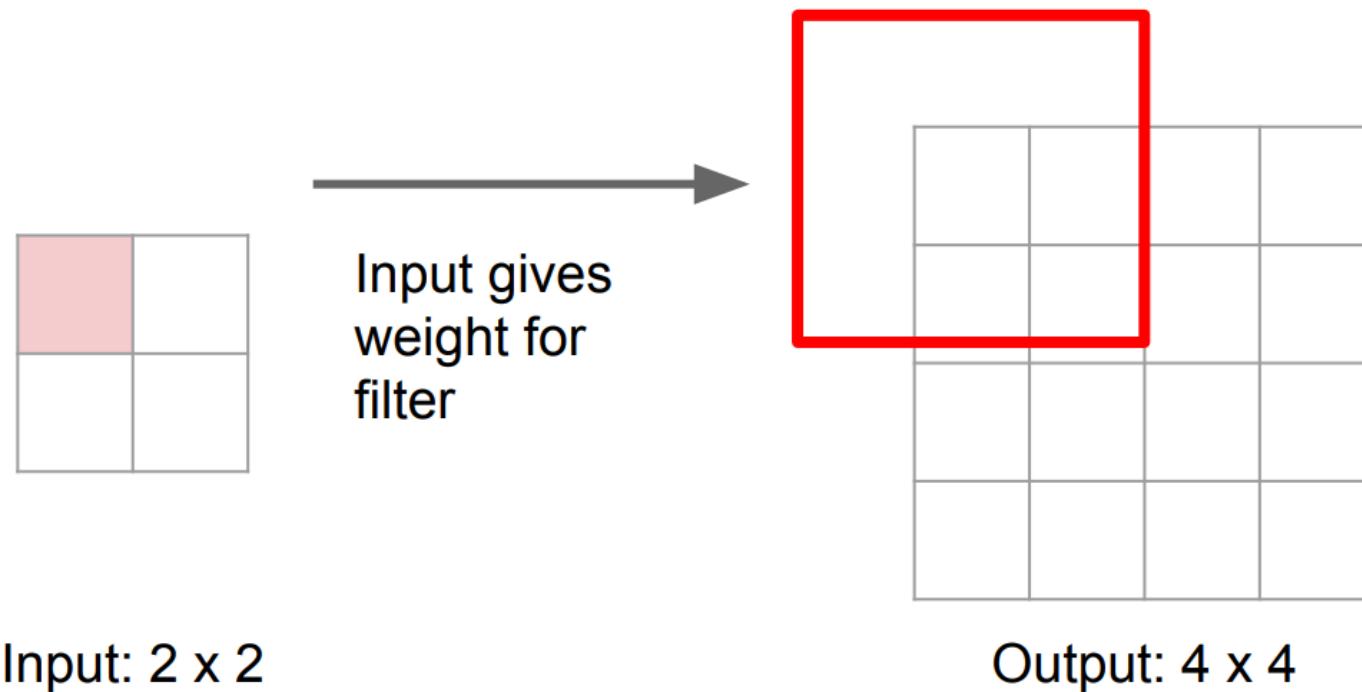
Filter moves 2 pixels in  
the input for every one  
pixel in the output

Stride gives ratio between  
movement in input and  
output

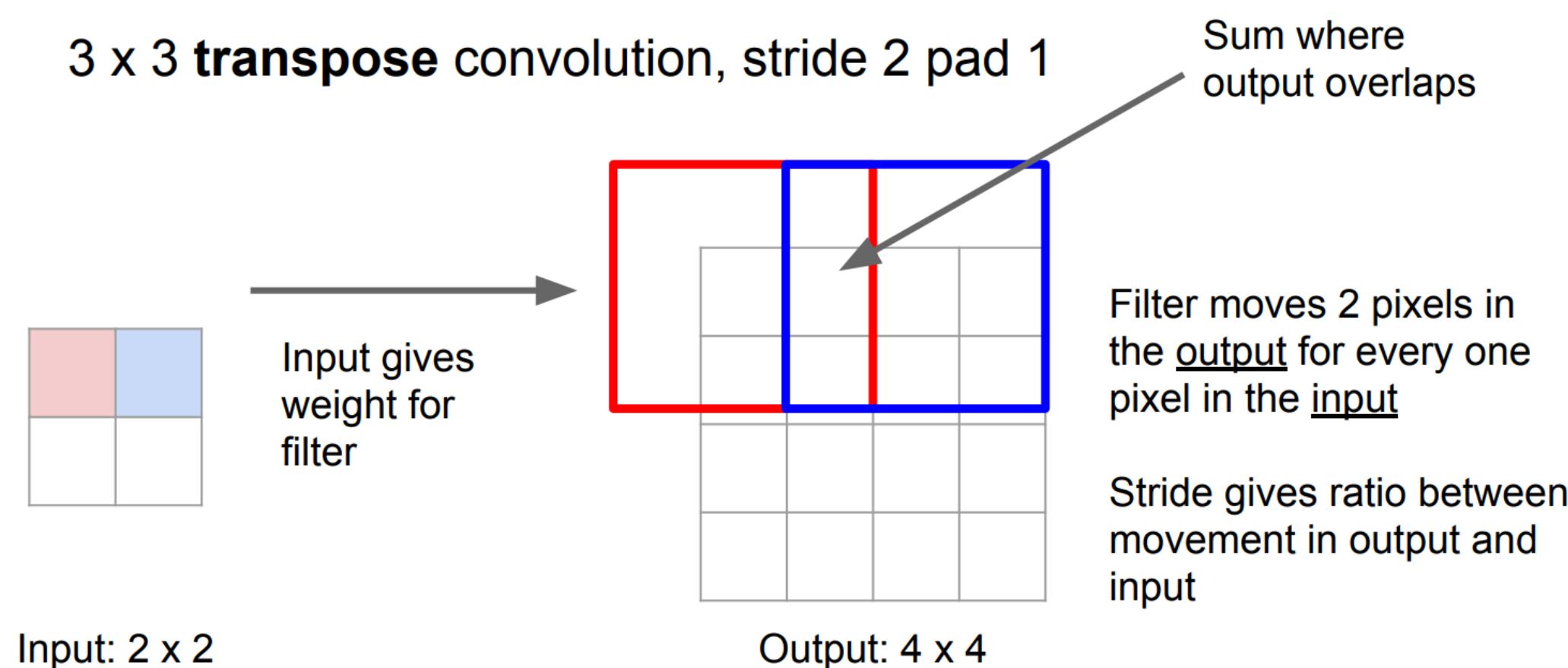


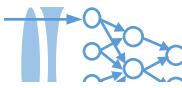
# Learnable Upsampling: Transpose Convolution

3 x 3 **transpose** convolution, stride 2 pad 1

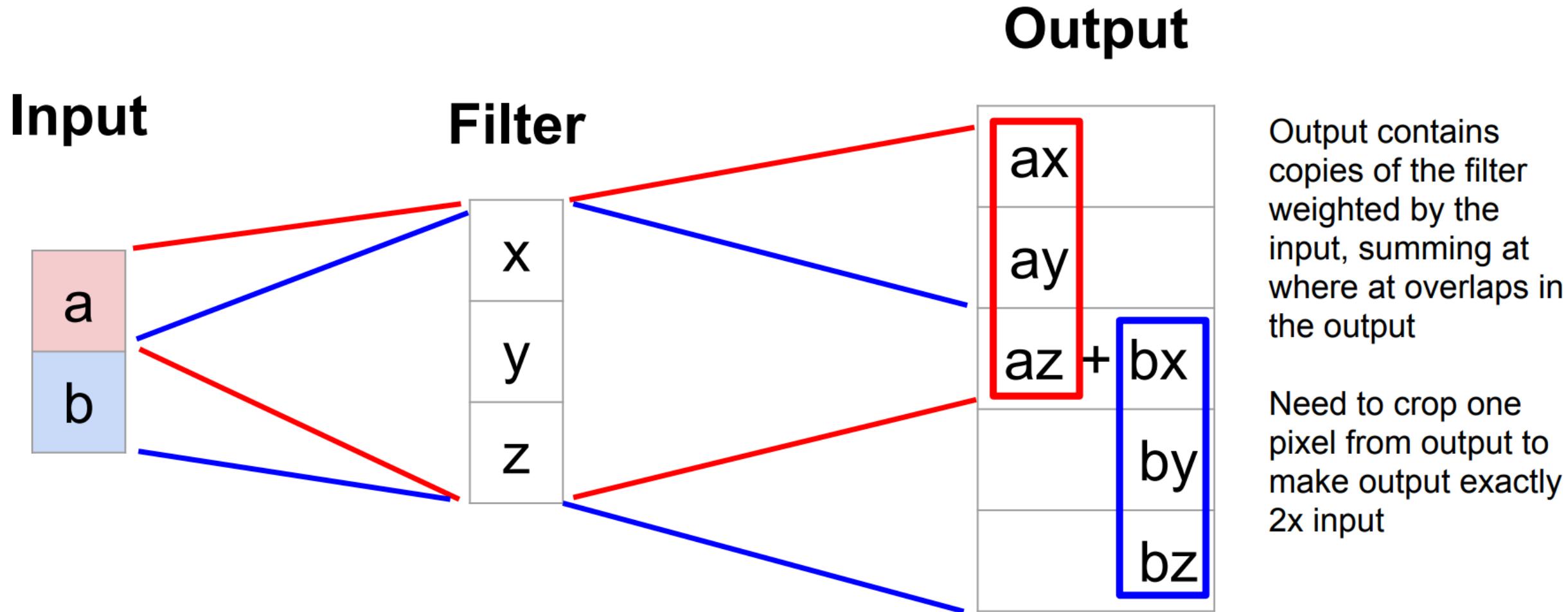


# Learnable Upsampling: Transpose Convolution





# Learnable Upsampling: 1D Example





# 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

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

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

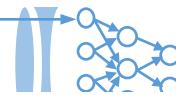
Example: 1D conv, kernel size=3, stride=1, padding=1

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:

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

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

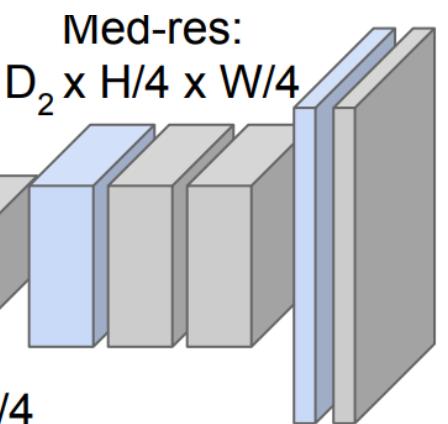
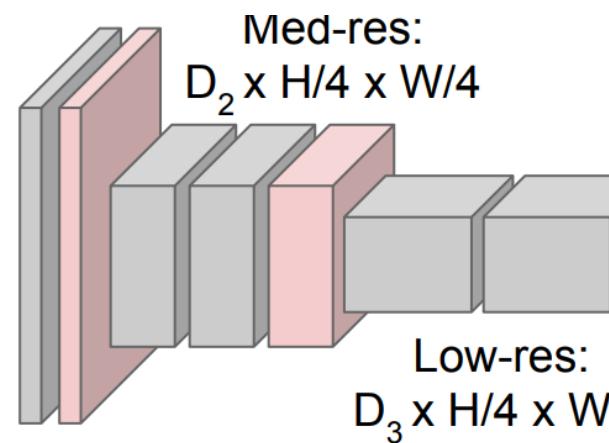
When stride>1, convolution transpose is no longer a normal convolution!

# Semantic Segmentation Idea: Fully Convolutional

- Train and test the same way as for classification, now just with labels that are  $H \times W$  maps!

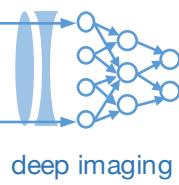


Input:  
 $3 \times H \times W$



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



Code review: See the following:

[Jupyter Notebook: A simple Autoencoder in Tensorflow/Keras](#)

<https://deepimaging.github.io/lectures/>