

# A Primer on Deep Learning based Medical Image Analysis

**Tianyu Han**

Physics of Molecular Imaging Systems (PMI)

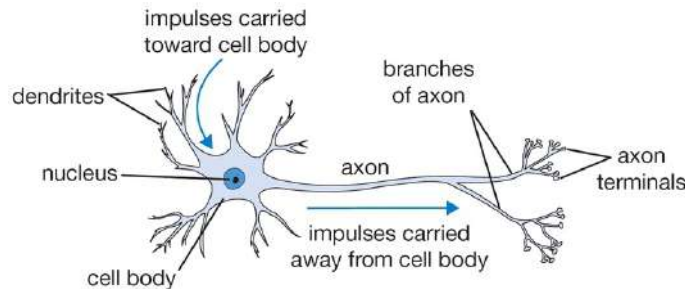
Experimental Molecular Imaging (ExMI)

RWTH Aachen University

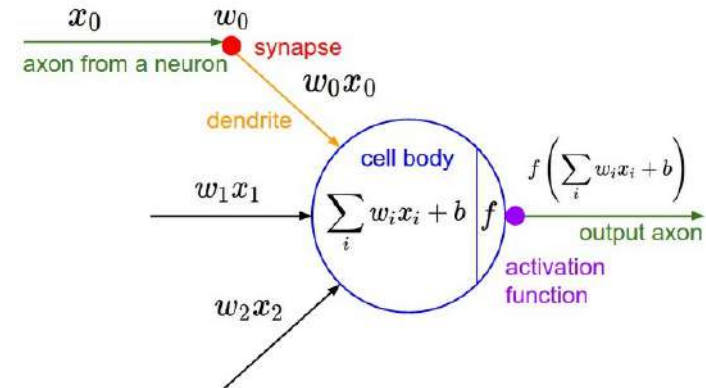
Uniklinik RWTH Aachen

- Deep learning basics
  - Supervised learning
  - Backpropagation to train multilayer architectures
- Image processing
  - Convolutional neural networks
- Medical image understanding with deep networks
  - Disease classification
  - Tumour detection and tissue segmentation

## a biological neuron



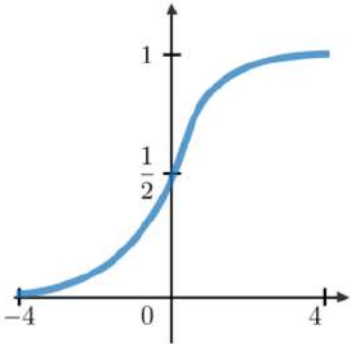
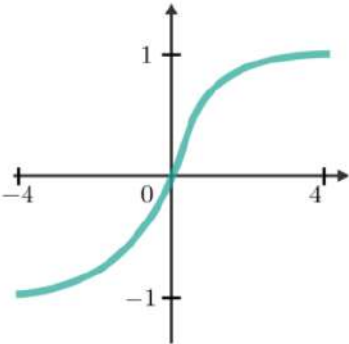
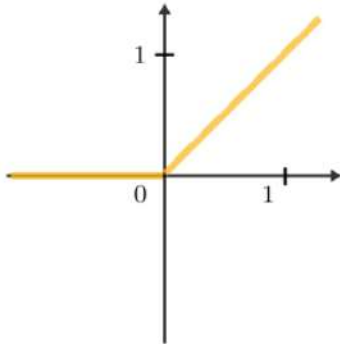
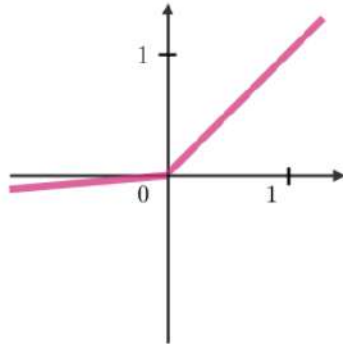
## mathematical model



- Synaptic strengths (the weights  $w_i$ ) are learnable
- the dendrites carry the signal to the cell body where they all get summed
  - If the final sum  $\geq$  certain threshold  $\rightarrow$  the neuron can *fire!*
- we model the firing rate of the neuron with an activation function  $f$

```
class Neuron(object):
    # ...
    def forward(self, inputs):
        """ assume inputs and weights are 1-D numpy arrays and bias is a number """
        cell_body_sum = np.sum(inputs * self.weights) + self.bias
        firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
        return firing_rate
```

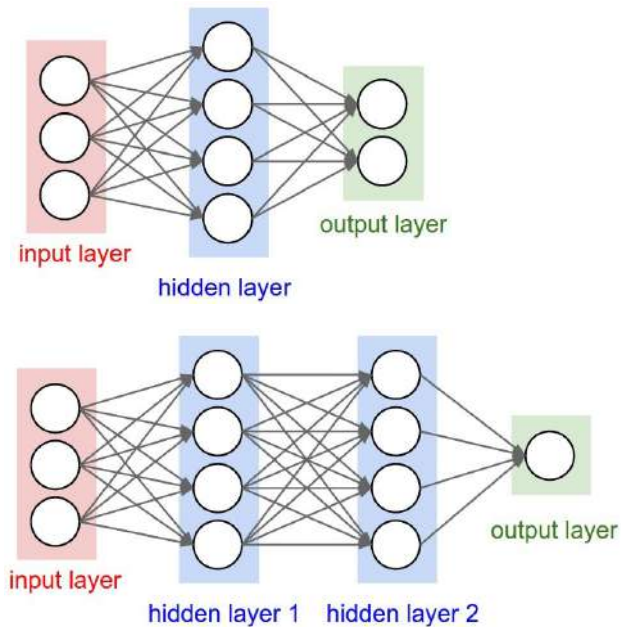
# Non-linear activation functions

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

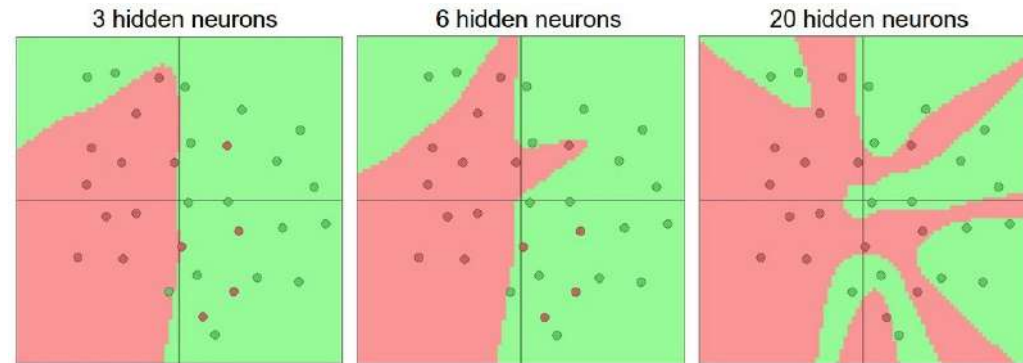
Now, we prefer ReLU activation over Sigmoid and Tanh:

- *Sigmoid* and *Tanh* saturate and kill gradients.
- *Sigmoid* outputs are not zero-centered.
  - Always,  $g'(z) = g(z)(1 - g(z)) > 0$

# Network architectures (MLP)



## Representational power



```
# forward-pass of a 3-Layer neural network:  
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)  
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)  
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)  
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)  
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

The forward pass of a fully-connected layer corresponds to one matrix multiplication followed by a bias offset and an activation function.

## MLP-Mixer: An all-MLP Architecture for Vision

Ilya Tolstikhin\*, Neil Houlsby\*, Alexander Kolesnikov\*, Lucas Beyer\*,  
Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner,  
Daniel Keysers, Jakob Uszkoreit, Mario Lucic, Alexey Dosovitskiy

\*equal contribution

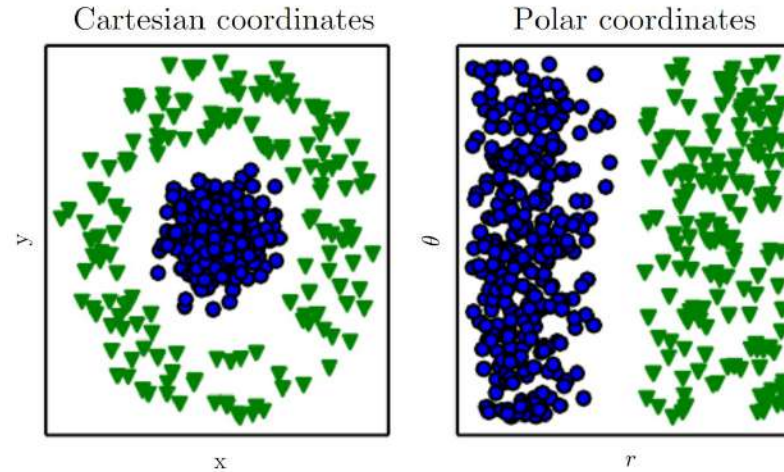
Google Research, Brain Team

{tolstikhin, neilhoulby, akolesnikov, lbeyer,  
xzhai, unterthiner, jessicayung, andstein,  
keysers, usz, lucic, adosovitskiy}@google.com

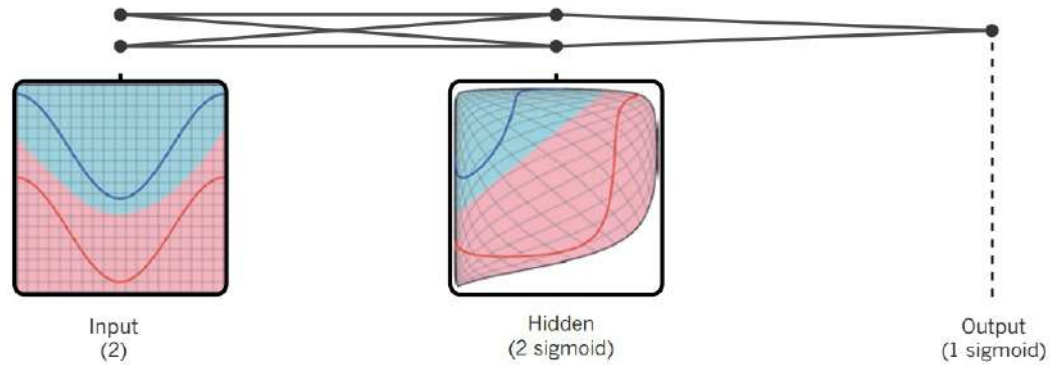
<sup>†</sup>work done during Google AI Residency

<https://arxiv.org/pdf/2105.01601.pdf>

# What is the purpose of the hidden layers?



<https://www.deeplearningbook.org/>



<https://www.nature.com/articles/nature14539.pdf>

## ■ Epoch:

- In the context of training a model, epoch is a term used to refer to one iteration where the model sees the whole training set to update its weights.

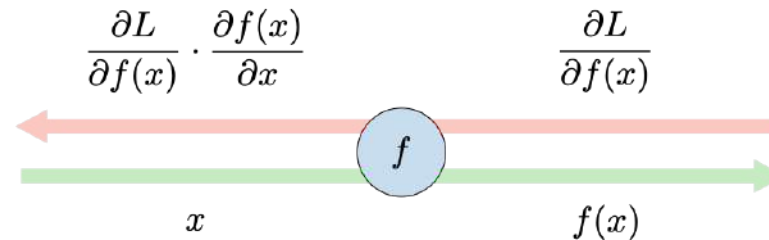
## ■ Mini-batch gradient descent:

- During the training phase, updating weights is usually **not** based on the whole training set at once due to computation complexities or one data point due to noise issues.
- Instead, the update step is done on mini-batches, where the number of data points in a batch is a hyperparameter that we can tune.

## ■ Loss function:

- In order to quantify how a given model performs, the loss function  $L$  is usually used to evaluate to what extent the actual outputs  $y$  are correctly predicted by the model outputs  $z$ .

# Finding optimal weights: backpropagation

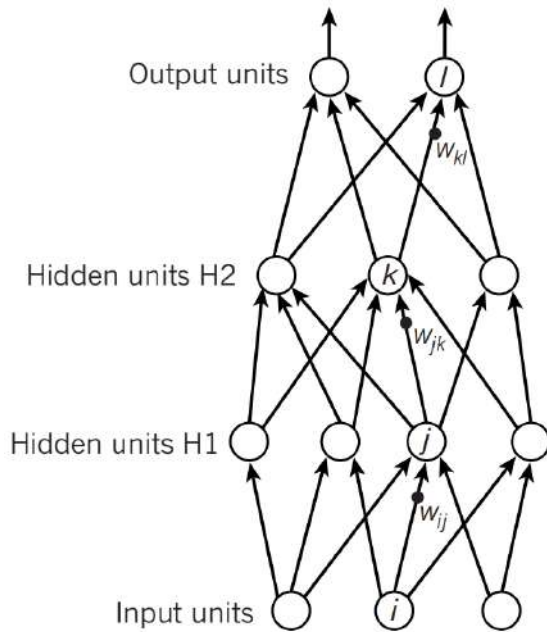


Updating weights - In a neural network, weights are updated as follows:

- Step 1: Take a batch of training data and perform forward propagation to compute the loss.
- Step 2: Backpropagate the loss to get the gradient of the loss with respect to each weight.
- Step 3: Use the gradients to update the weights of the network.



# Forward pass and backward pass



$$y_l = f(z_l)$$

$$z_l = \sum_{k \in H2} w_{kl} y_k$$

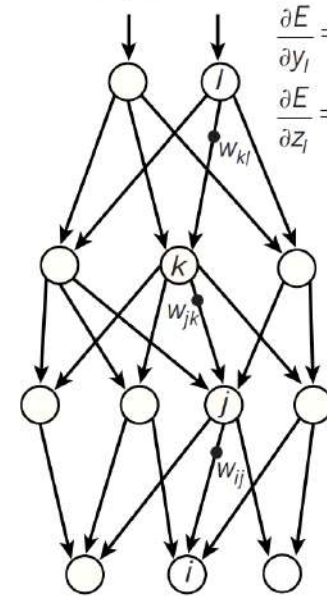
$$y_k = f(z_k)$$

$$z_k = \sum_{j \in H1} w_{jk} y_j$$

$$y_j = f(z_j)$$

$$z_j = \sum_{i \in \text{Input}} w_{ij} x_i$$

Compare outputs with correct answer to get error derivatives



$$\frac{\partial E}{\partial y_l} = y_l - t_l$$

$$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l} \frac{\partial y_l}{\partial z_l}$$

$$\frac{\partial E}{\partial y_k} = \sum_{l \in \text{out}} w_{kl} \frac{\partial E}{\partial z_l}$$

$$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial z_k}$$

$$\frac{\partial E}{\partial y_j} = \sum_{k \in H2} w_{jk} \frac{\partial E}{\partial z_k}$$

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_j}$$

# Parameter tuning and transfer learning

Training size	Illustration	Explanation
Small		Freezes all layers, trains weights on softmax
Medium		Freezes most layers, trains weights on last layers and softmax
Large		Trains weights on layers and softmax by initializing weights on pre-trained ones



<https://image-net.org/>

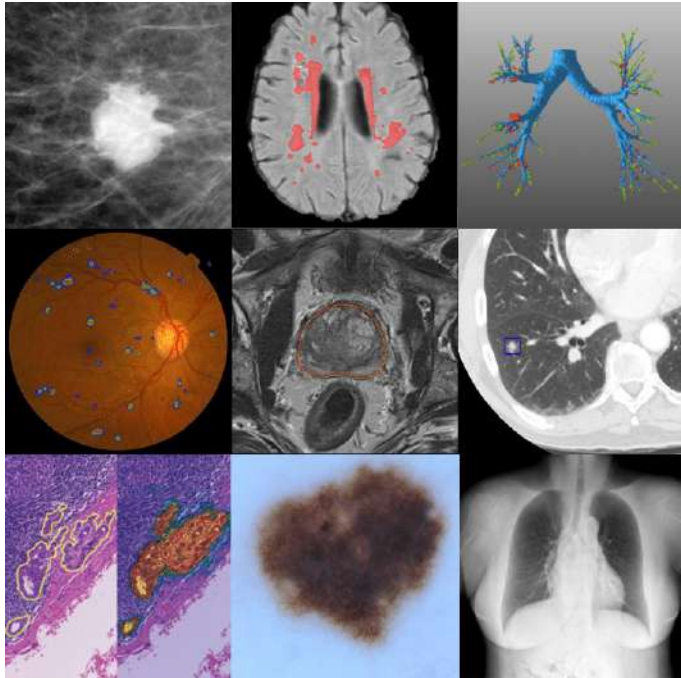


<https://cocodataset.org/#home>

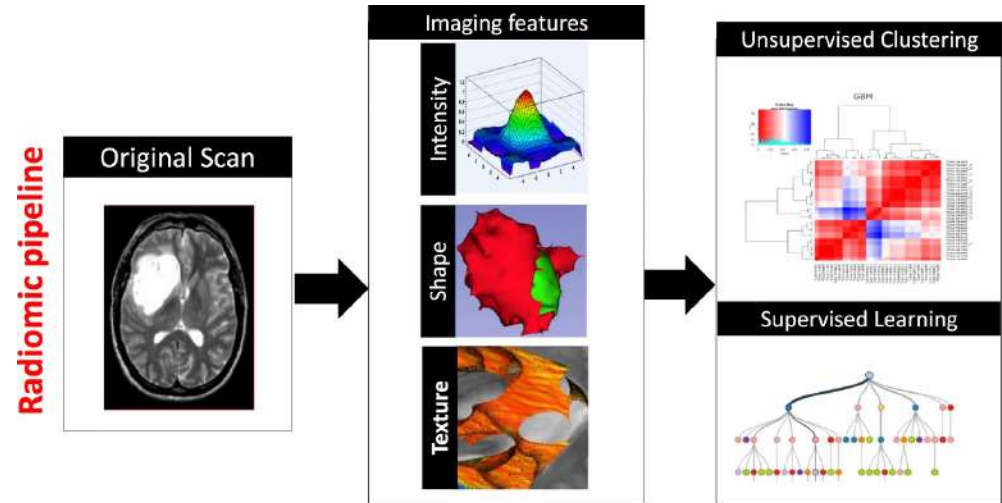


<https://research.google.com/youtube8m/>

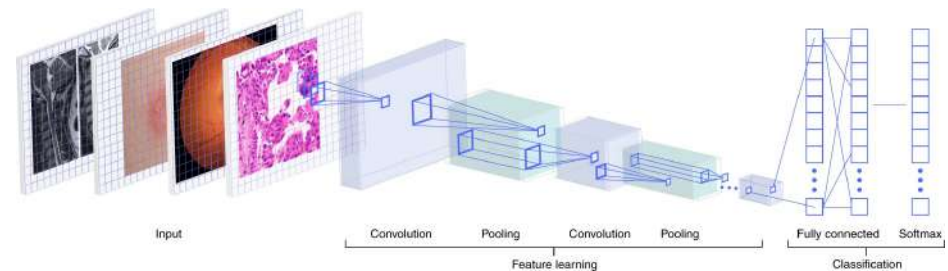
# Processing medical imaging data



<https://www.sciencedirect.com/science/article/abs/pii/S1361841517301135?via%3Dihub>

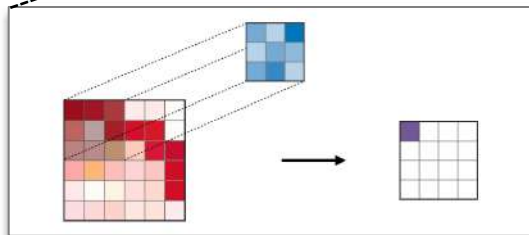
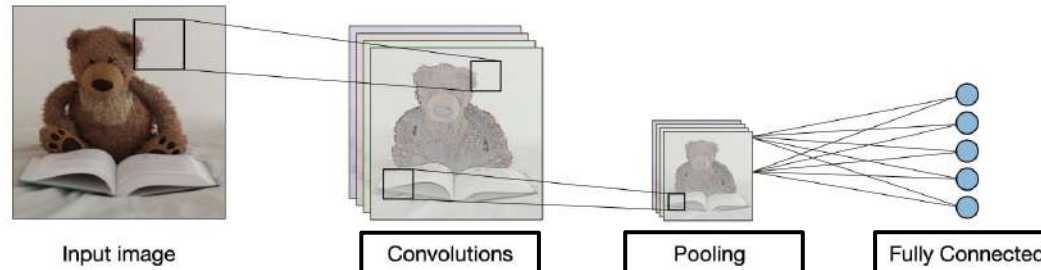


<https://www.nature.com/articles/s41416-021-01387-w>

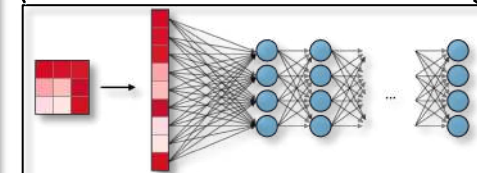


<https://www.nature.com/articles/s41591-018-0316-z>

# Layers in a convolutional neural network

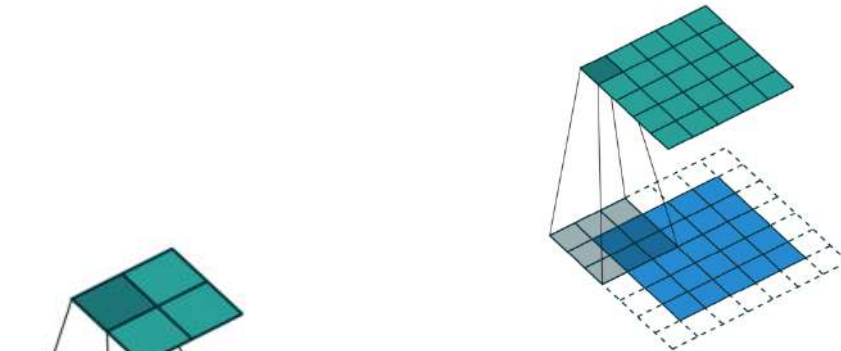


Type	Max pooling	Average pooling
<b>Purpose</b>	Each pooling operation selects the maximum value of the current view	Each pooling operation averages the values of the current view
<b>Illustration</b>		
<b>Comments</b>	<ul style="list-style-type: none"> <li>• Preserves detected features</li> <li>• Most commonly used</li> </ul>	<ul style="list-style-type: none"> <li>• Downsamples feature map</li> <li>• Used in LeNet</li> </ul>

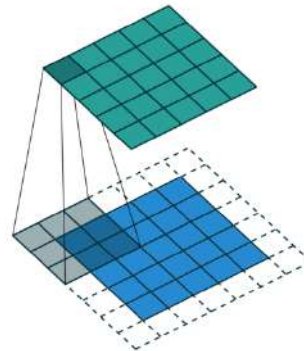




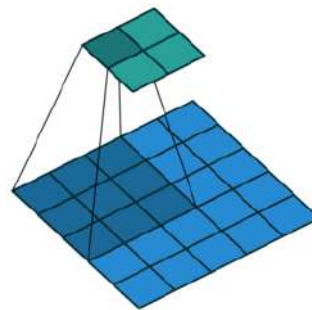
# Hyperparameters inside the convolutional layer



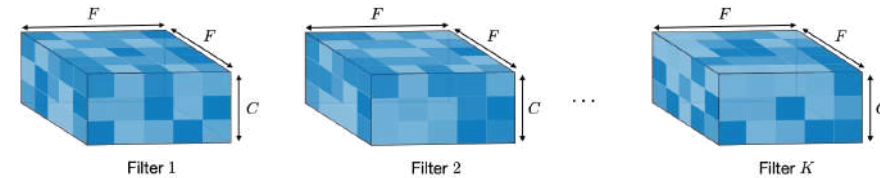
- 3x3 conv kernel
- Valid (**no**) padding
- Stride = 1



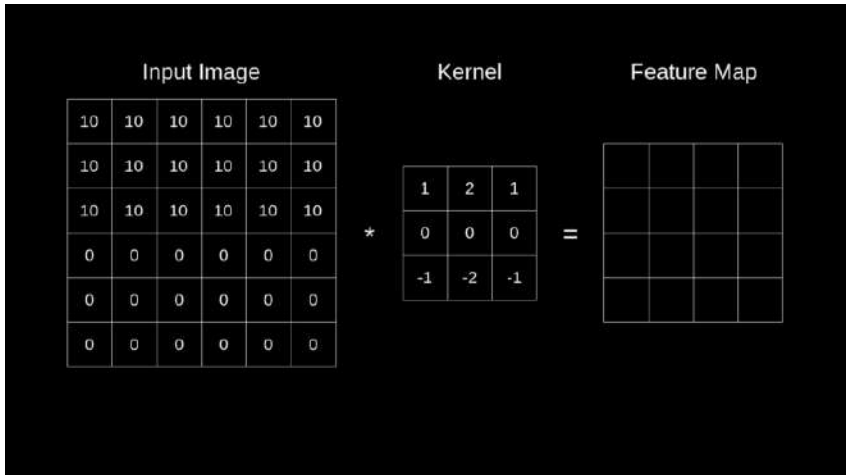
- 3x3 conv kernel
- **Same padding**
- Stride = 1



- 3x3 conv kernel
- Same padding
- **Stride = 2**



	CONV
Illustration	$\begin{matrix} & & F & & \\ & & \begin{matrix} \downarrow F \\ \uparrow F \end{matrix} & & \\ & & \otimes C & & \\ & & \times K & & \end{matrix}$
Input size	$I \times I \times C$
Output size	$O \times O \times K$
Number of parameters	$(F \times F \times C + 1) \cdot K$
Remarks	<ul style="list-style-type: none"> <li>• One bias parameter per filter</li> <li>• In most cases, <math>S &lt; F</math></li> <li>• A common choice for <math>K</math> is <math>2C</math></li> </ul>



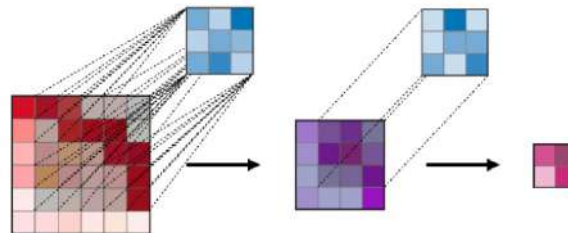
One step convolutional operation:

```
def convolution2d(image, kernel, bias):
    m, n = kernel.shape
    if m == n:
        y, x = image.shape
        y = y - m + 1
        x = x - m + 1
        new_image = np.zeros((y,x))
        for i in range(y):
            for j in range(x):
                new_image[i][j] = np.sum(image[i:i+m, j:j+m]*kernel) + bias
    return new_image
```

Receptive field:

$$R_k = 1 + \sum_{j=1}^k (F_j - 1) \prod_{i=0}^{j-1} S_i$$

In the example below, we have  $F_1 = F_2 = 3$  and  $S_1 = S_2 = 1$ , which gives  $R_2 = 1 + 2 \cdot 1 + 2 \cdot 1 = 5$ .



# Famous CNN model designs: residual connection

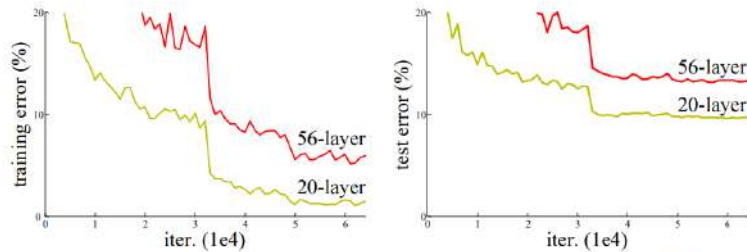
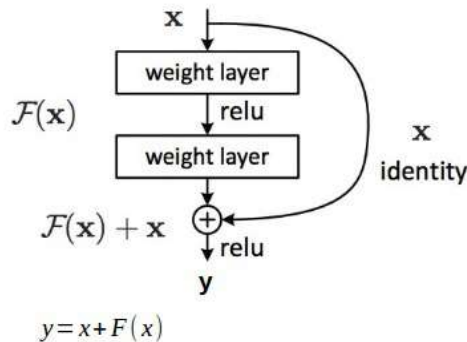
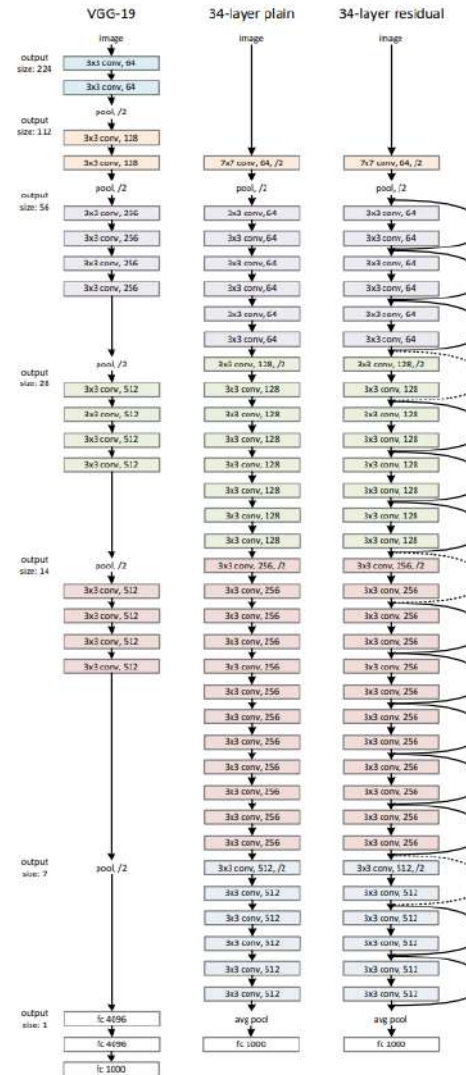



Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.



$$\frac{\delta E}{\delta x} = \frac{\delta E}{\delta y} * \frac{\delta y}{\delta x} = \frac{\delta E}{\delta y} * (1 + F'(x))$$

$$= \frac{\delta E}{\delta y} + \frac{\delta E}{\delta y} * F'(x)$$






**Input**  
Chest X-Ray Image

---

**CheXNet**  
121-layer CNN

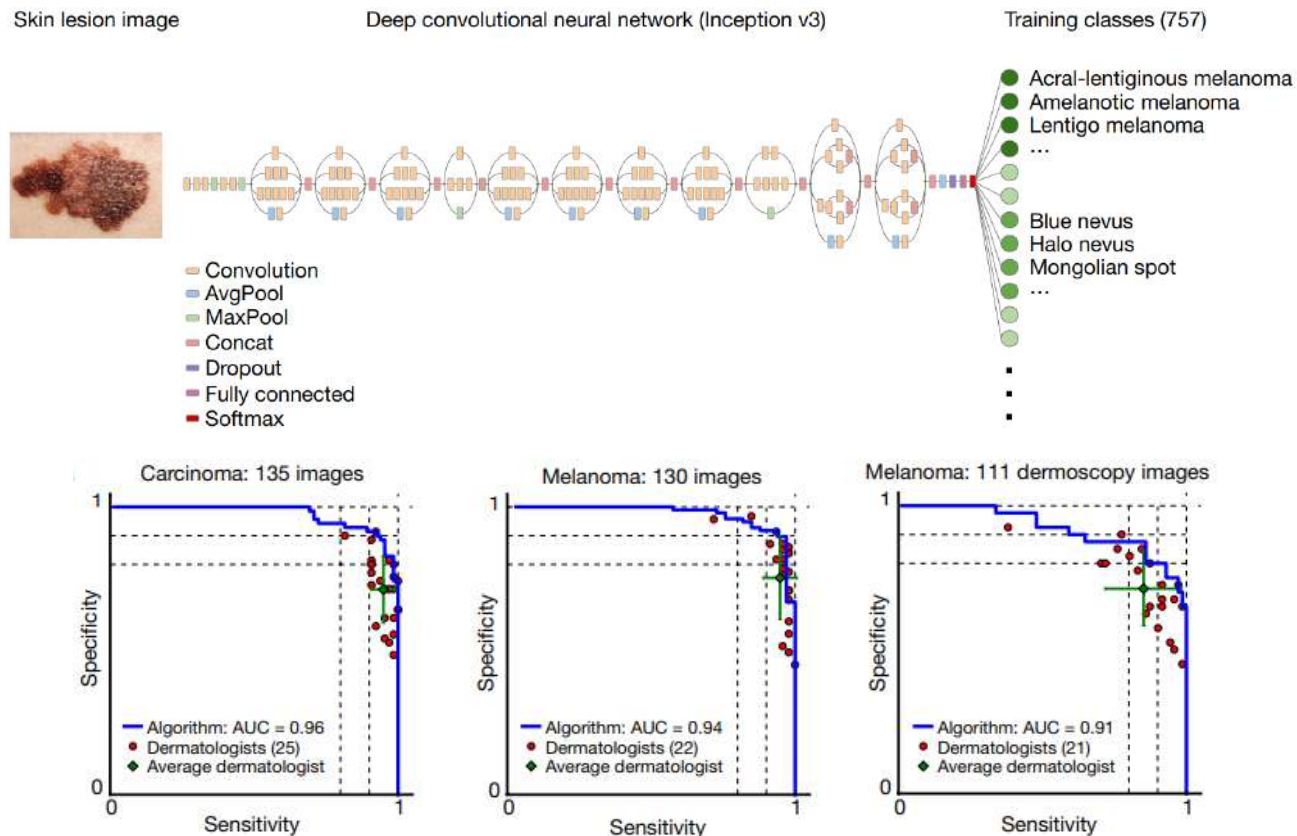
---

**Output**  
Pneumonia Positive (85%)



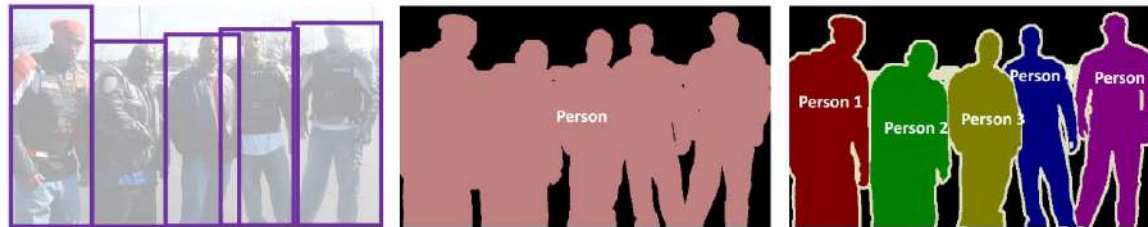
## Dermatologist-level classification of skin cancer with deep neural networks

Andre Esteva<sup>1\*</sup>, Brett Kuprel<sup>1\*</sup>, Roberto A. Novoa<sup>2,3</sup>, Justin Ko<sup>2</sup>, Susan M. Swetter<sup>2,4</sup>, Helen M. Blau<sup>5</sup> & Sebastian Thrun<sup>6</sup>





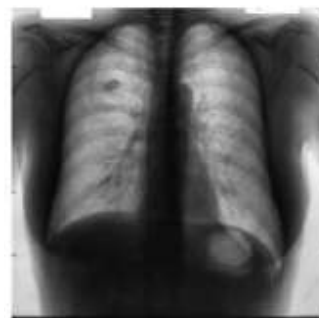
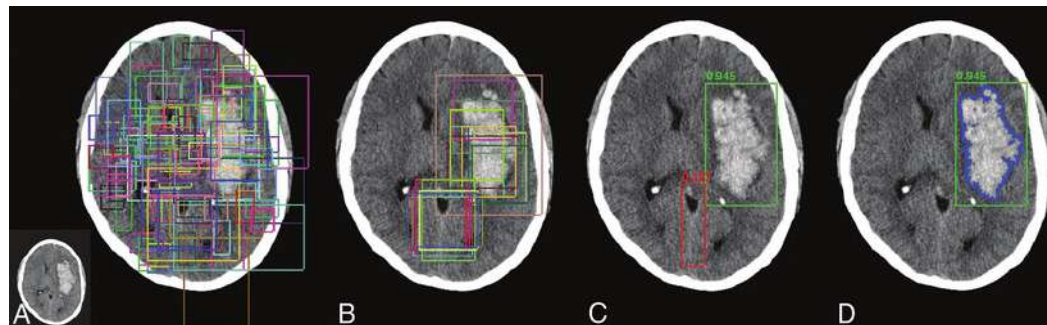
# Beyond image classification



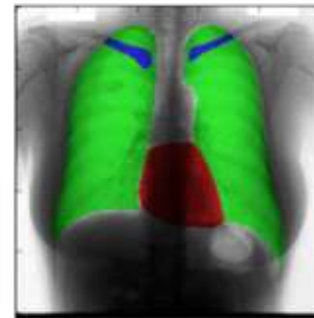
Object Detection

Semantic Segmentation

Instance Segmentation

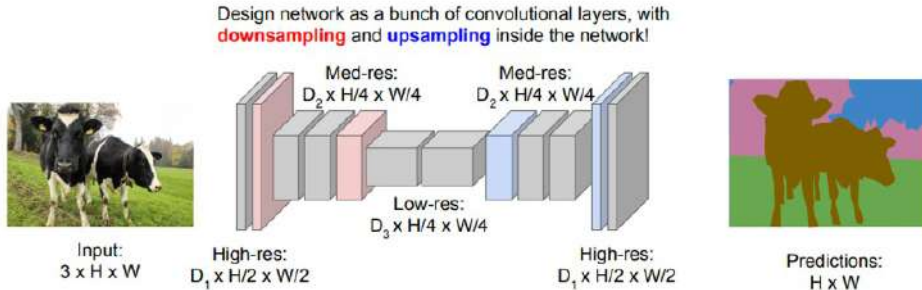


Input Image

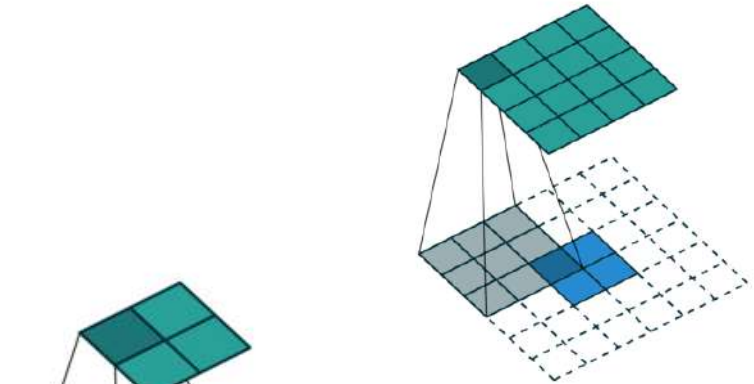
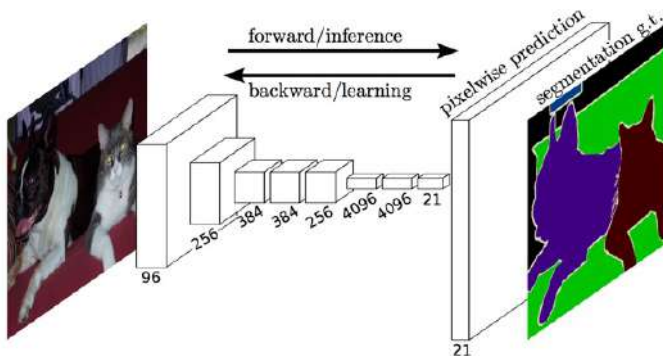


Segmented Image

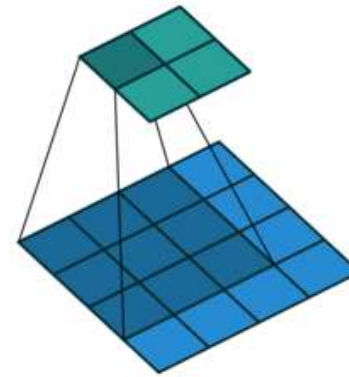
# Convolutions in segmentation



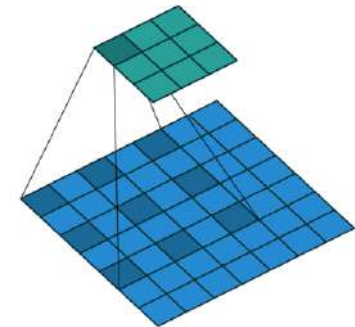
**Solution:** Make network deep and work at a lower spatial resolution for many of the layers.



Transpose convolution (upsampling)

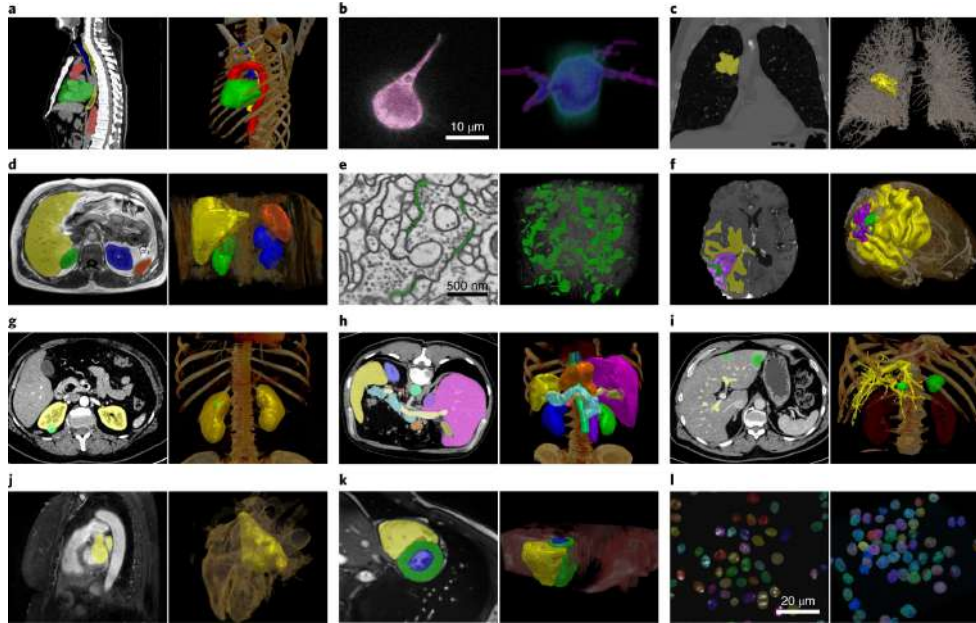


Convolution

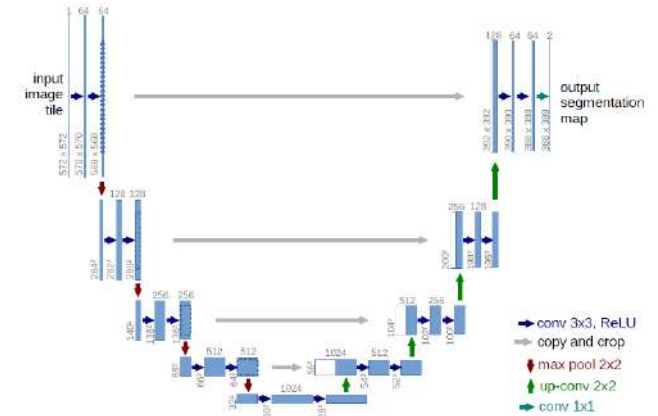


Dilated convolution

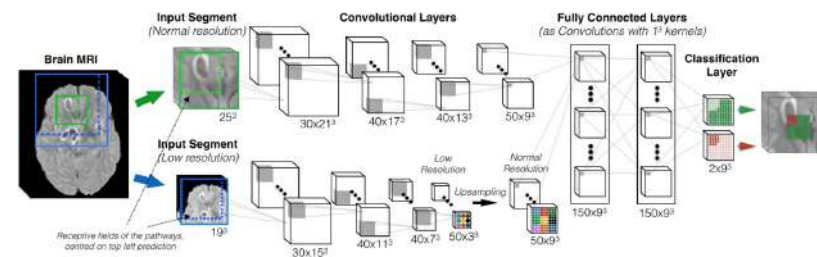
# Beyond classification: medical segmentation



<https://www.nature.com/articles/s41592-020-01008-z>



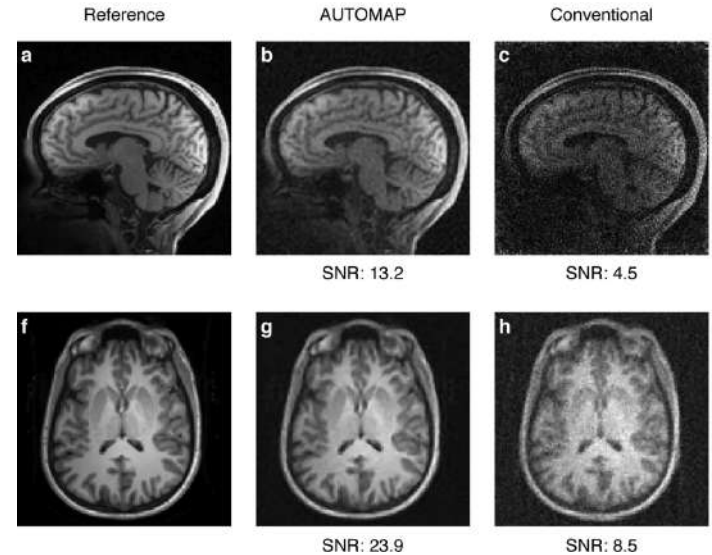
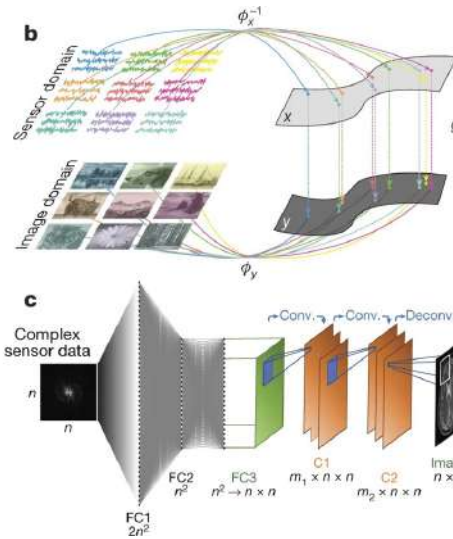
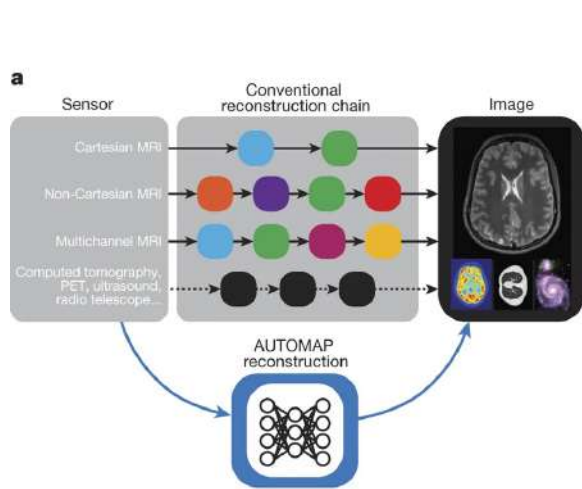
<https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/>



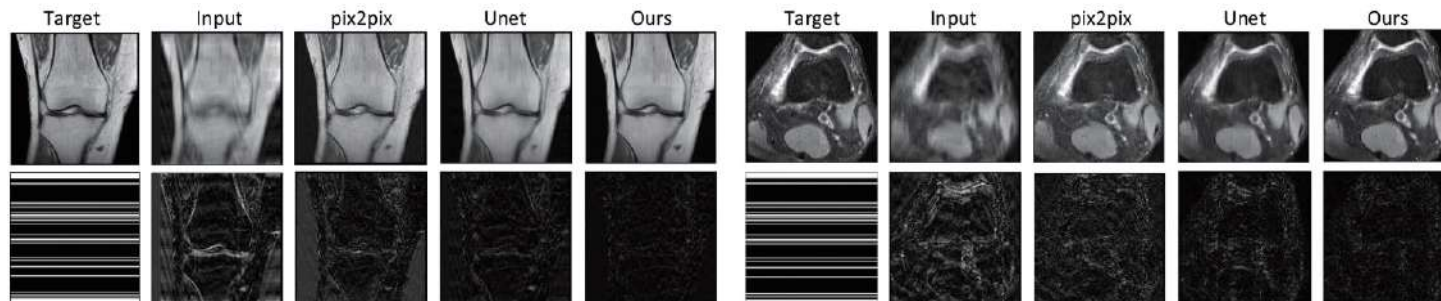
<https://www.sciencedirect.com/science/article/pii/S1361841516301839>



# Medical image reconstruction



<https://www.nature.com/articles/nature25988>



[https://openaccess.thecvf.com/content\\_CVPR\\_2019/papers/Zhang\\_Reducing\\_Uncertainty\\_in\\_Undersampled\\_MRI\\_Reconstruction\\_With\\_Active\\_Acquisition\\_CVPR\\_2019\\_paper.pdf](https://openaccess.thecvf.com/content_CVPR_2019/papers/Zhang_Reducing_Uncertainty_in_Undersampled_MRI_Reconstruction_With_Active_Acquisition_CVPR_2019_paper.pdf)

# Data driven, ennn..., where is our data?

## ■ Disease Classification

- CheXpert, ChestXray-14, MIMIC-CXR
- Osteoarthritis Initiative
- Diabetic Retinopathy Detection (Kaggle)

## ■ Tissue segmentation

- Multimodal Brain Tumour Segmentation Challenge (BraTS)
- Medical Segmentation Decathlon
- EchoNet Dynamic

## ■ Disease detection:

- National Lung Screening Trial (NLST)
- Lung Image Database Consortium image collection (LIDC-IDRI)

Thank you for listening!  
Questions?

# Optimization by Adam optimizer

Backpropagation

$$\theta = \theta - \eta \cdot \overbrace{\nabla_{\theta} J(\theta; x, y)}^{\text{Backpropagation}} \quad J(\theta; x, y): \text{loss function}$$

↓

$$\theta_{t+1} = \theta_t - \frac{\eta \cdot \hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$$

where

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

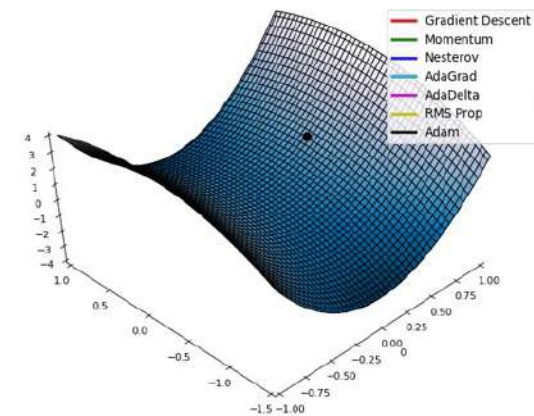
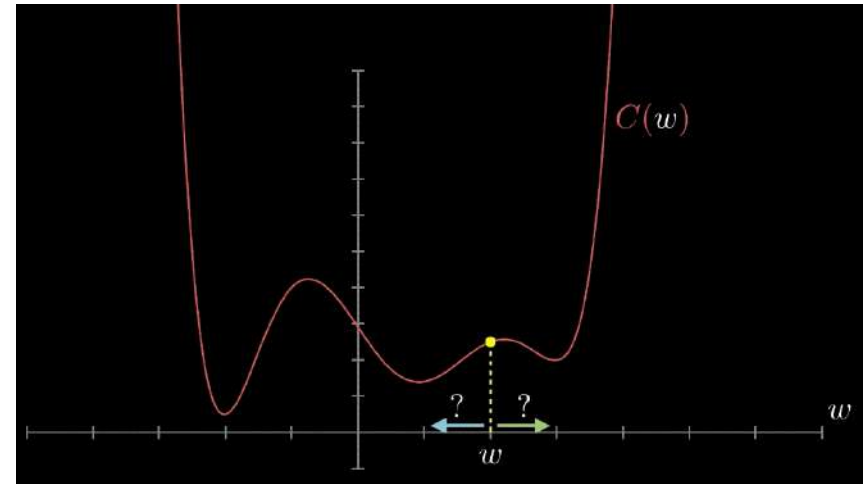
and where

$$m_t = (1 - \beta_1)g_t + \beta_1 m_{t-1}$$

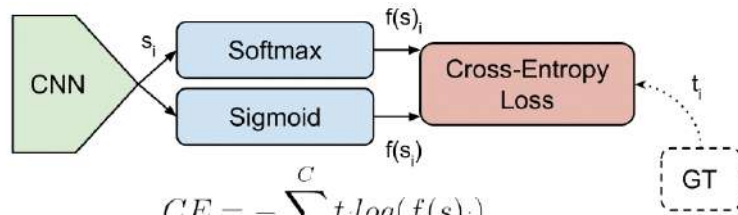
$$v_t = (1 - \beta_2)g_t^2 + \beta_2 v_{t-1}$$

(Moving) average

↖

$$g_t = \nabla_{\theta} J(\theta; x, y): \text{gradient}$$


# Classification loss function and evaluation

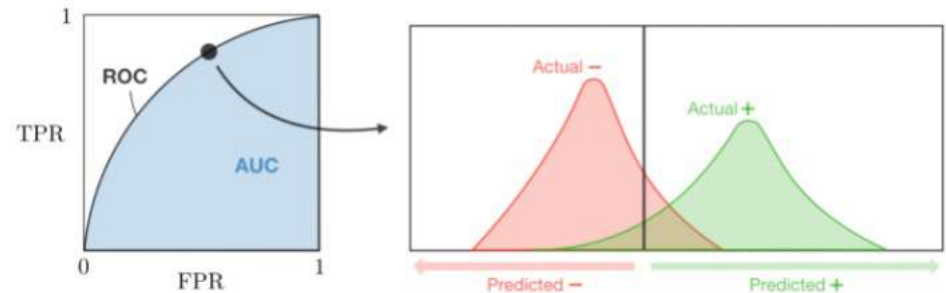


$$CE = - \sum_i^C t_i \log(f(s)_i)$$

$$CE = - \sum_{i=1}^{C'-2} t_i \log(f(s_i)) = -t_1 \log(f(s_1)) - (1 - t_1) \log(1 - f(s_1))$$

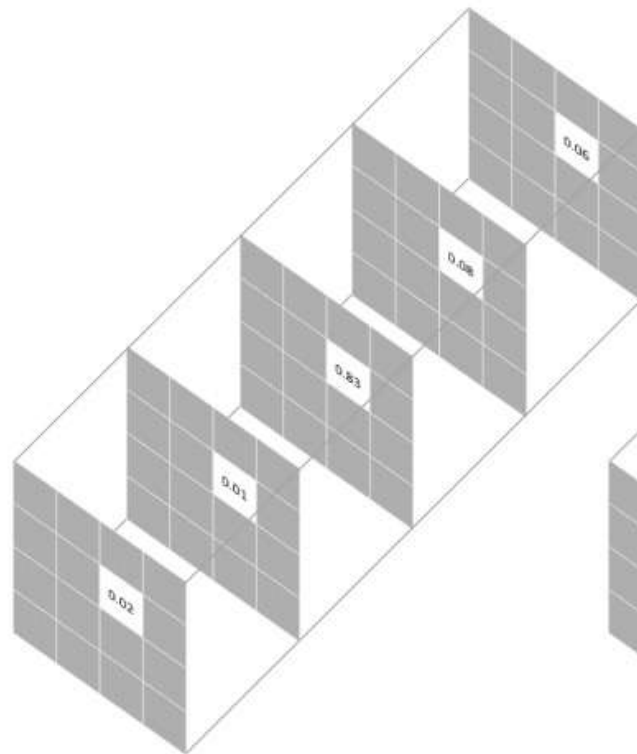
Metric	Formula	Equivalent
True Positive Rate TPR	$\frac{TP}{TP + FN}$	Recall, sensitivity
False Positive Rate FPR	$\frac{FP}{TN + FP}$	1-specificity

		Predicted class	
		+	-
Actual class	+	<b>TP</b> True Positives	<b>FN</b> False Negatives Type II error
	-	<b>FP</b> False Positives Type I error	<b>TN</b> True Negatives

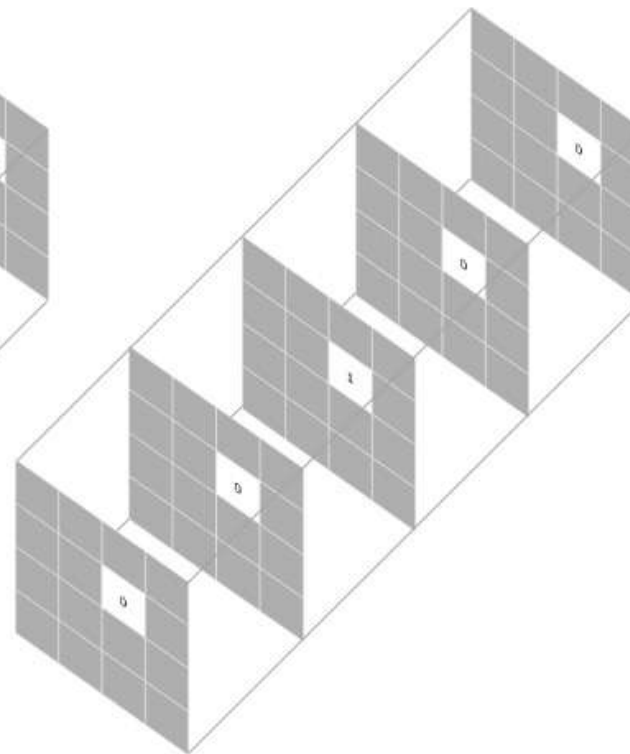




# Cross entropy loss functions in segmentation



Prediction for a selected pixel



Target for the corresponding pixel

Pixel-wise loss is calculated as the log loss, summed over all possible classes

$$-\sum_{classes} y_{true} \log(y_{pred})$$

This scoring is repeated over all **pixels** and averaged

# Dice loss in segmentation

$$Dice = \frac{2 |A \cap B|}{|A| + |B|}$$

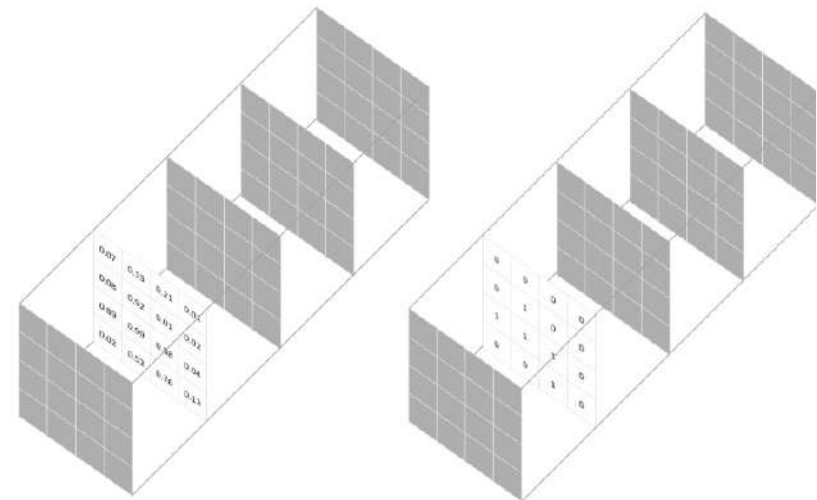
$$|A \cap B| = \begin{bmatrix} 0.01 & 0.03 & 0.02 & 0.02 \\ 0.05 & 0.12 & 0.09 & 0.07 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

prediction                      target

$$\xrightarrow{\text{element-wise multiply}} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix} \xrightarrow{\text{sum}} 7.41$$

$$|A| = \begin{bmatrix} 0.01 & 0.03 & 0.02 & 0.02 \\ 0.05 & 0.12 & 0.09 & 0.07 \\ 0.89 & 0.85 & 0.88 & 0.91 \\ 0.99 & 0.97 & 0.95 & 0.97 \end{bmatrix}^{2 \text{ (optional)}} \xrightarrow{\text{sum}} 7.82$$

$$|B| = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}^{2 \text{ (optional)}} \xrightarrow{\text{sum}} 8$$



Prediction for a selected class

Target for the corresponding class