Deep Learning Part 1

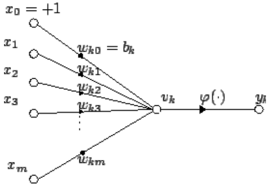
Introduction

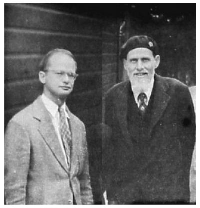
Artificial Neural Networks - Timeline

A Logical Calculus of Ideas Immanent in Nervous Activity

1943

McCulloch–Pitts neuron



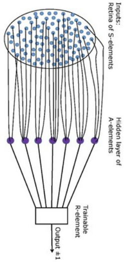


Warren McCulloch and Walter Pitts

The perceptron :
A probalistic model for information storage and organization in the brain

1958

perceptrons

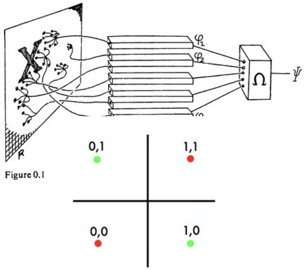


Frank Rosenblatt

Perceptions
An Introduction to Computational Geometry

1969

limits of specific perceptrons

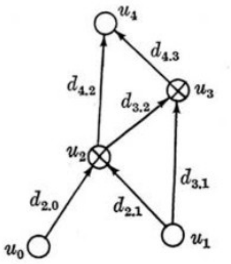


XOR

Marvin Minsky and Seymour Papert

The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors

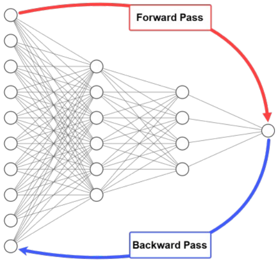
1970



Seppo Linnainmaa

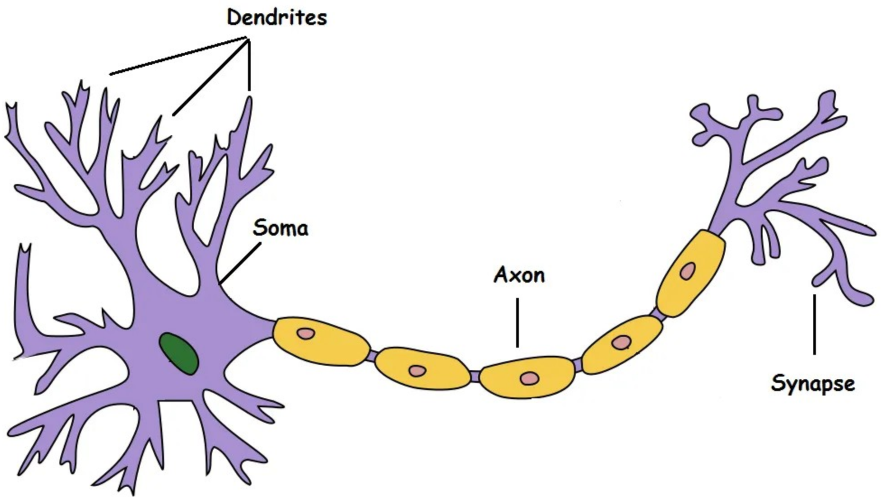
Learning representations by back-propagating errors

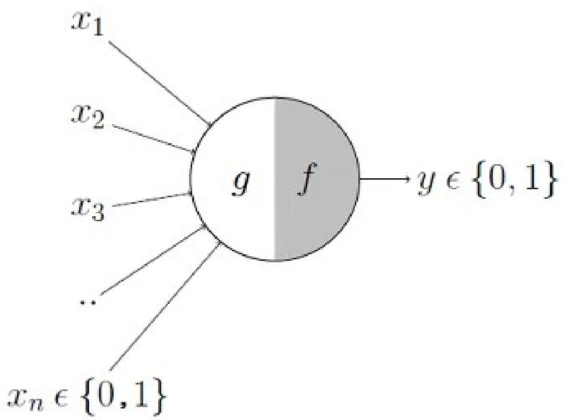
1986



David Rumelhart, Geoffrey Hinton, Ronald Williams

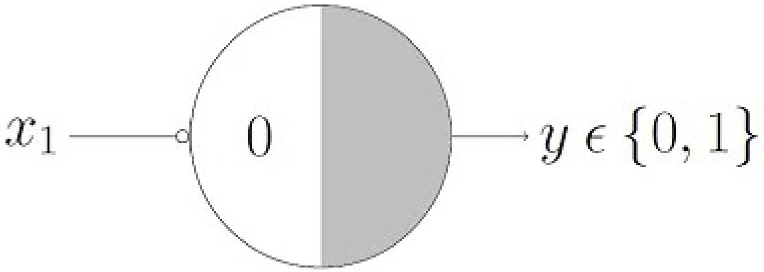
McCulloch-Pitts Neuron



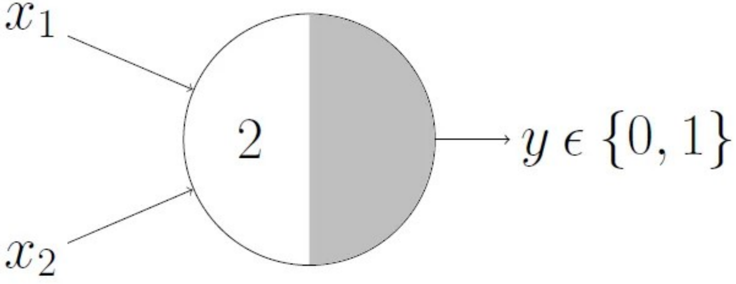


- The output of a neuron is all or nothing (0 or 1)
- Input synapses are exciting or inhibiting
- If one inhibiting synapse is active the output is 0
- Otherwise
  - The output is 1 if more than a fixed number of exciting synapses are active
  - And zero otherwise

Networks of McCulloch Pitts neurons can calculate any logical functions

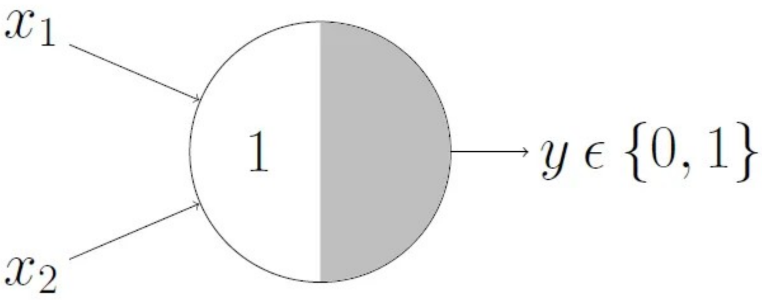


NOT(x1)



$AND\ function$

$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 2$$

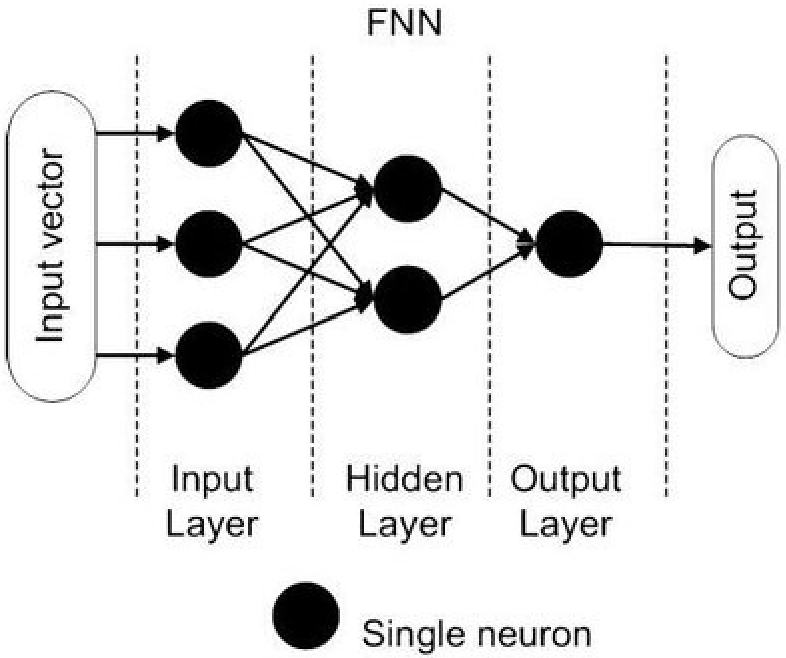


$OR\ function$

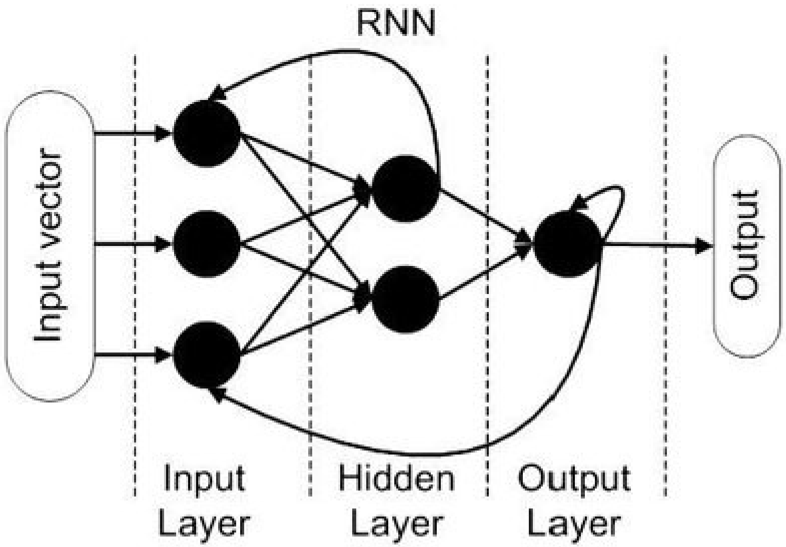
$$x_1 + x_2 = \sum_{i=1}^{2} x_i \geq 1$$

Artifical Neural Networks (ANNs)

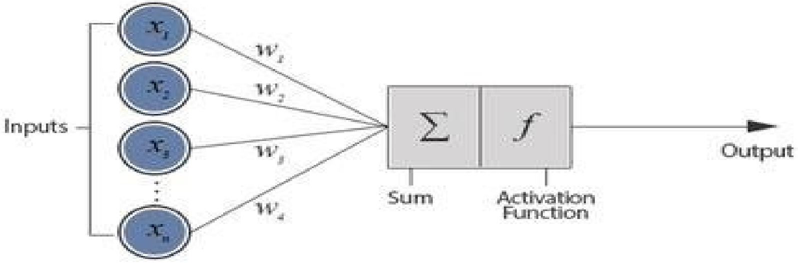
Feed-Forward Neural Network

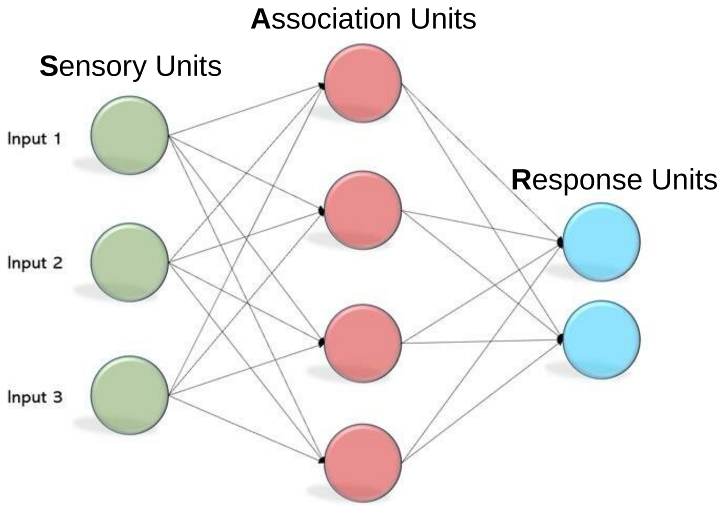


Recurrent Neural Network



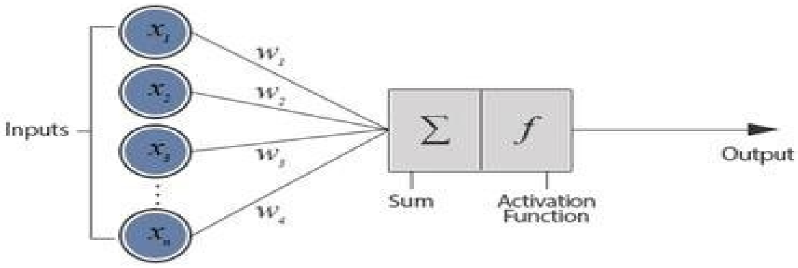
Perceptrons (Rosenblatt)

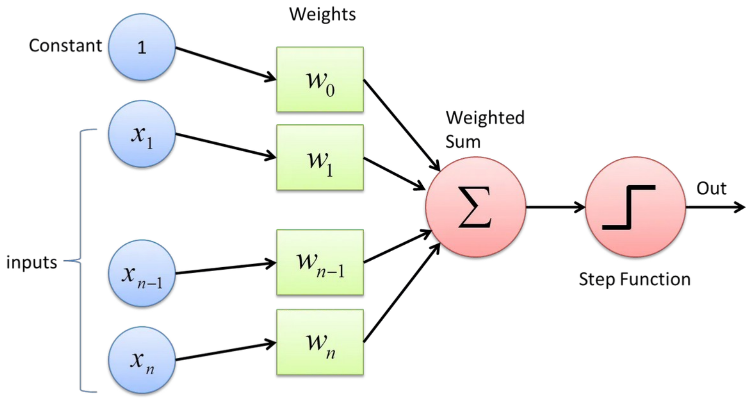


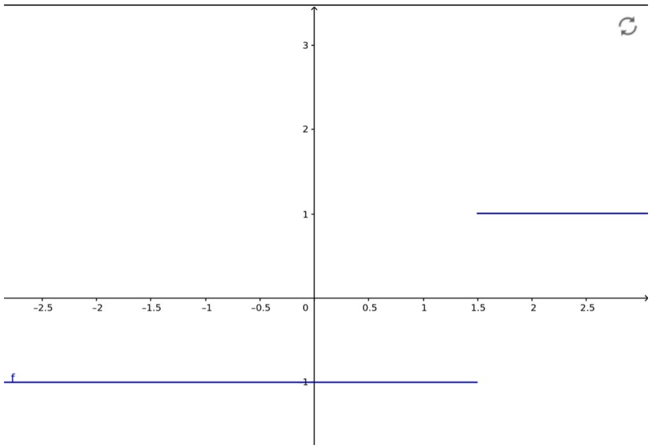


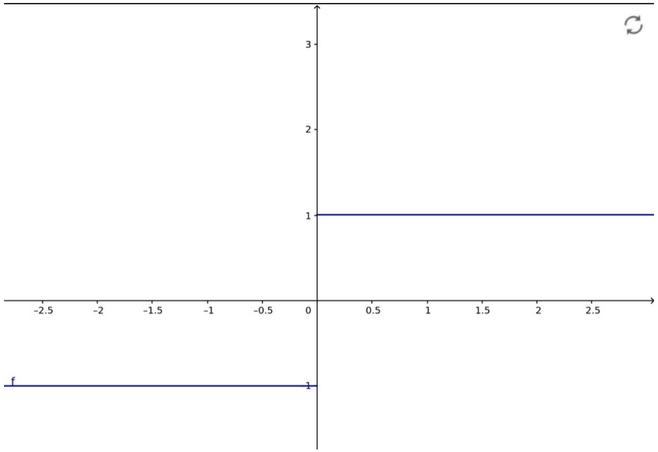
- A neuron calculates a weighted sum S of the input
- The output of a neuron is 1 if S is above a threshold value and -1 otherwise.
- It learns through reinforcement, by changing the weights of the connections and the threshold values

Bias instead of Threshold value

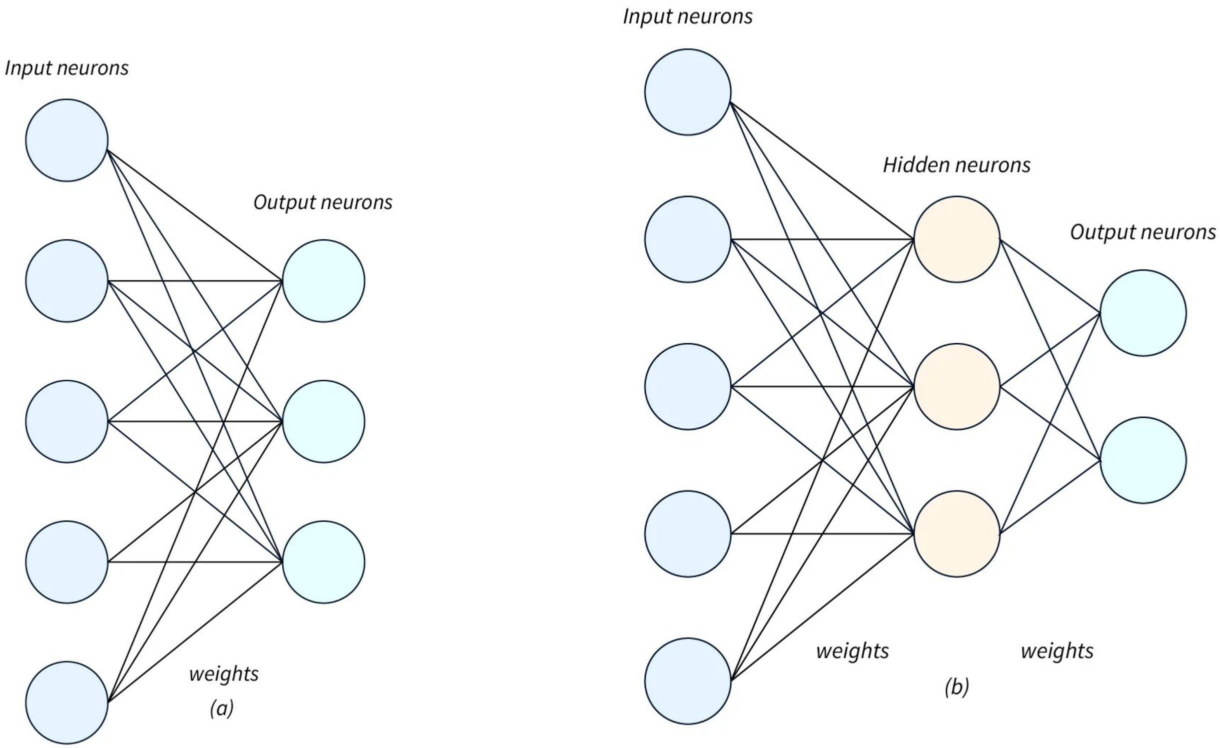






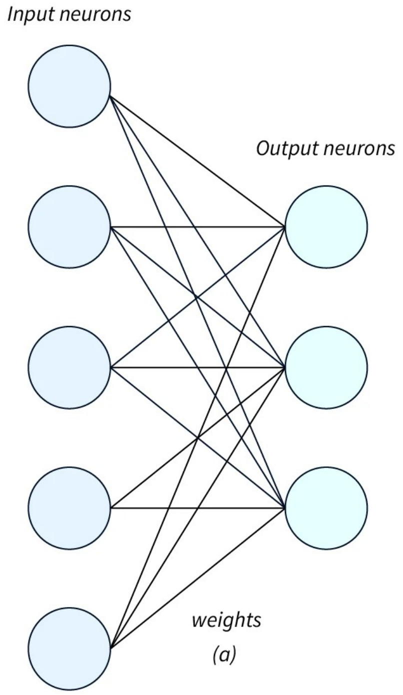


Single-layer / multi-layer



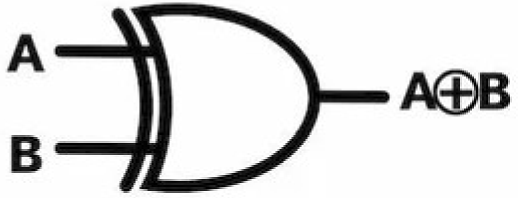
Training

- N times repeat (N number of epochs) :
  - For all pairs of input-ground-truth data:
    - Forward pass
      - Calculate output
      - Calculate the difference between the output and gt for each output neuron
    - Adjust weights
      - Add the difference * the input * the learning rate to each weight



Restrictions of perceptrons

- Single layer perceptron does only linear classification
- Can't calculate the logical xor function

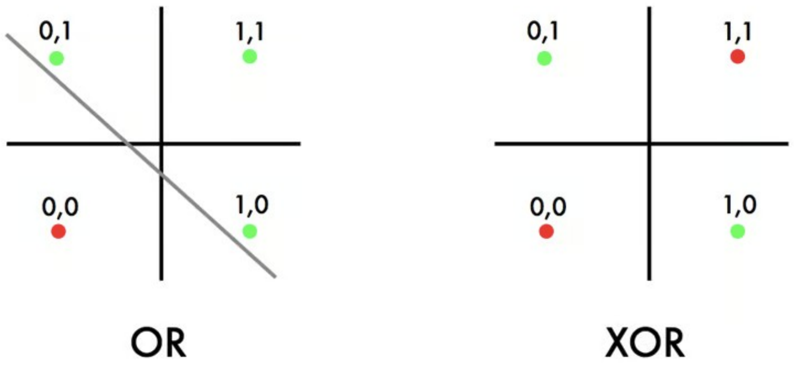


*eXclusive OR

2 input XOR gate

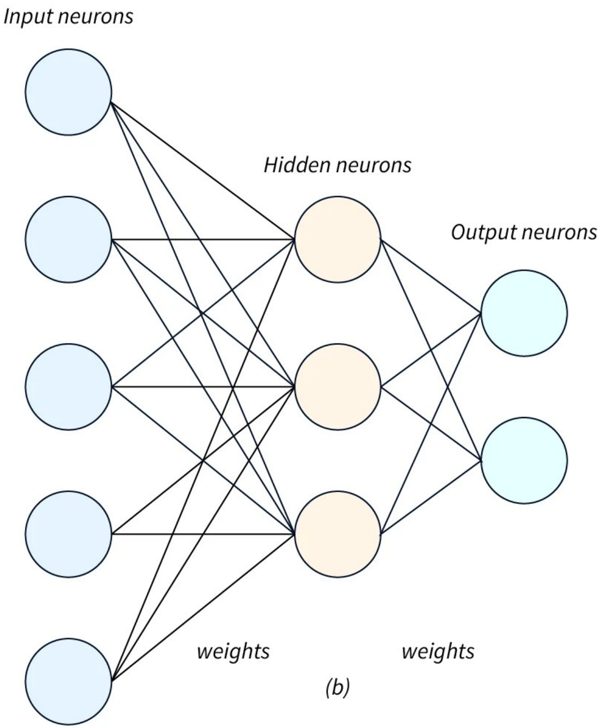
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

The XOR problem



Training of multilayer perceptrons

- Multi-layer perceptron can calculate any logical function
- But how to train it?
- The output of a response neuron depends on all the inner neurons
- Solution:
  - Use gradient descent on the error-function and calculate the gradients via backpropagation



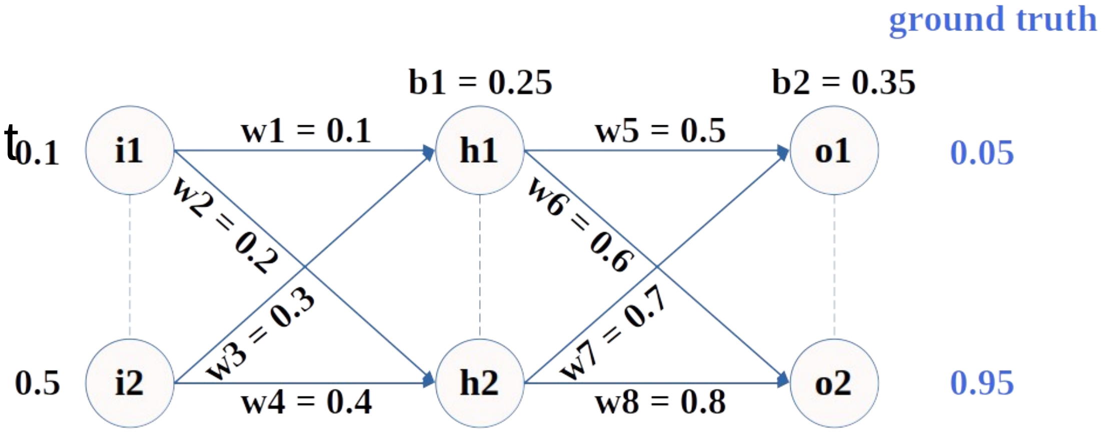
Chain rule

- We need to calculate the gradiant of the error function for each weight
- The derivative of a function, which is a composition of functions

  h(x) = f(g(x))

  is

  f'(g(x)) * g'(x)

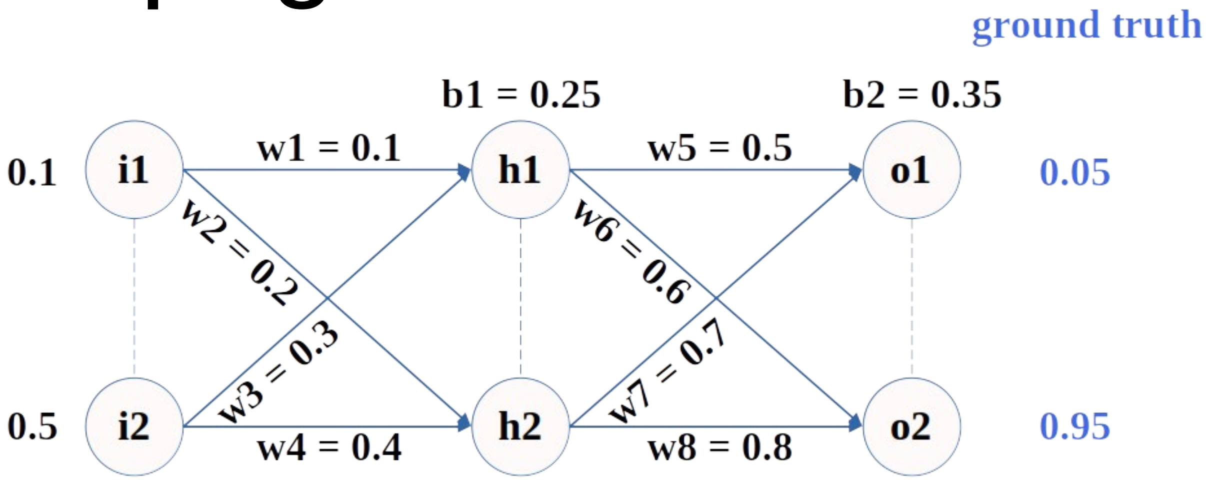


$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial a_j^k} \frac{\partial a_j^k}{\partial w_{ij}^k}$$

Backpropagation

- The loss is a function of the parameters of the network (w_i, b_i)
- The loss is the average of the loss for all pairs of input and ground truth
- The loss for one input/GT pair is the accumulation of the losses of all output-neurons
- The loss of one output neuron is a function of the difference between the output and the ground truth

$$E = \frac{1}{2}\sum_c \sum_j (y_{j,c} - d_{j,c})^2$$

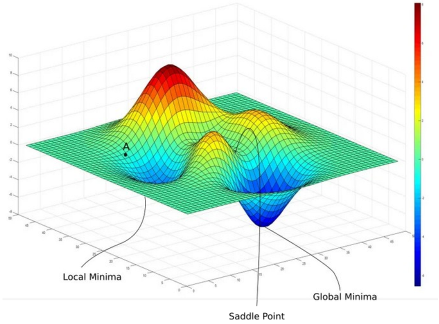


- gradiant at the output neuron

$$\partial E / \partial y_j = y_j - d_j$$

- Calculate gradiants for all weights and biases from output to previous layer using the chain chain rule

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial a_j^k} \frac{\partial a_j^k}{\partial w_{ij}^k}$$

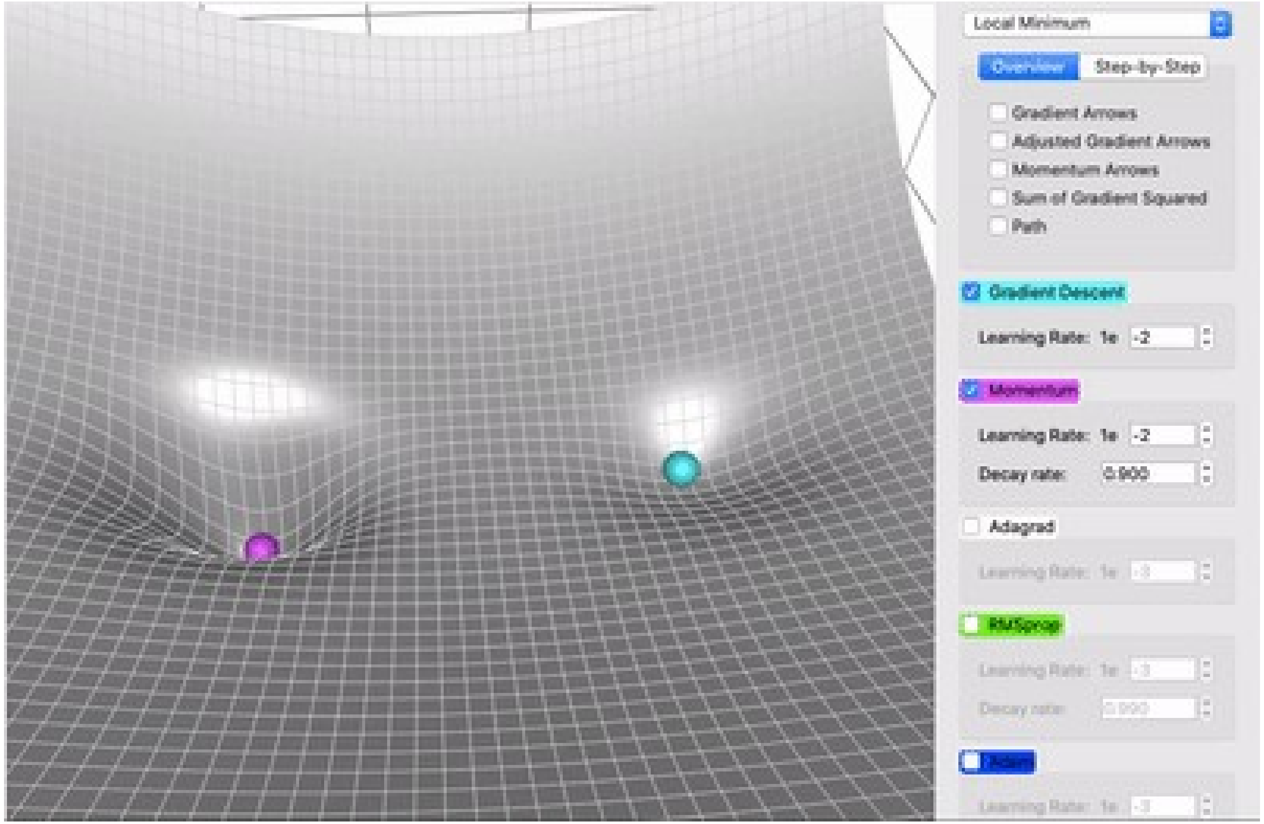


Optimizers

- Backpropagation calculates the gradients of the error function
- Optimizer decides how to update the weights

- Stochatical gradient descent (SGD)
  w = w - learning_rate * g
- SGD with momentum
  - Calculate a running exponential average of gradients from previous steps
    - More weight given to closer values
    - Parameter gamma controls the impact of the momentum (often 0.9)

SGD with and without momentum

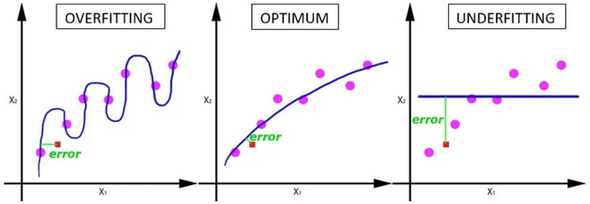


Adaptive Moment Estimation (Adam)

- Use first order and second order momentum, i.e. average and variation of previous gradients
- Adapt the learning rate for each parameter based on the previous gradients and squared gradients

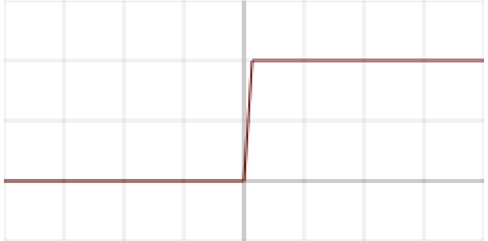
- Meta-parameters
  - Learning rate
  - $Beta_1$: decay rate for the average (0.9)
  - $Beta_2$: decay rate for the variance (0.999)

Mini batches

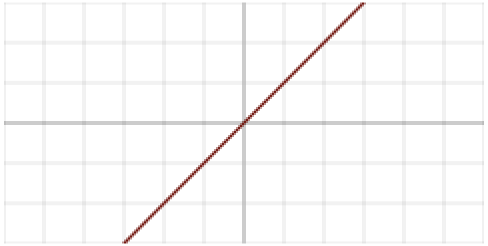


- Calculation of gradients and update of weights in each epoch, can be done using:
  - online learning
    - For each input, ground truth pair
  - Mini batches
    - For a given number of input, ground truth pairs
  - Full Batch
    - For all input, ground truth pairs in the training set
    - Full batch is gradient descent instead of stochastic gradient descent
- Mini batches
  - Less memory needed
  - Adds noise which can help to
    - Generalize
    - Not get stuck in local minima
  - Use smaller learning rate with larger batch-size
  - For historical reasons the batch size is often a power of 2 : 32, 64, 128, 256, …
- The order of the input, ground truth pairs is often randomized for each epoch

Activation Functions



- Threshold (step function) of perceptron not good for gradient descent
- The derivative is undefined at 0 and 0 anywhere else



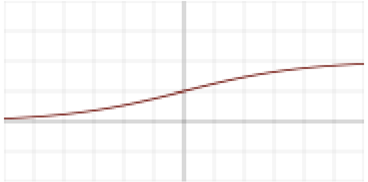
- With linear functions we can only get linear results, independent of the number of layers

- We need activation functions that are
  - Continuously differentiable
  - Nonlinear

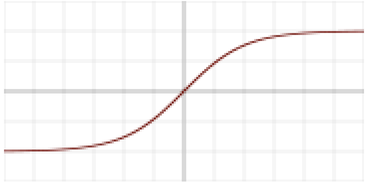
Activation Functions

- Saturating
  - vanishing gradient problem

Sigmoid

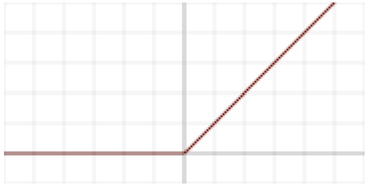


tanh



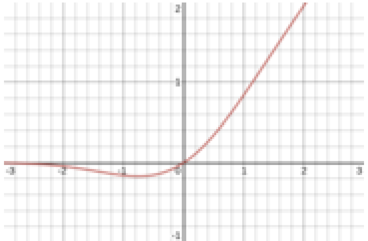
- Non-saturating
  - Exploding gradients
    - Batch Normalization

Rectified Linear Unit (ReLU)

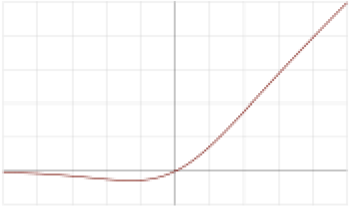


dead neurons

Gaussian Error Linear Unit (GELU)

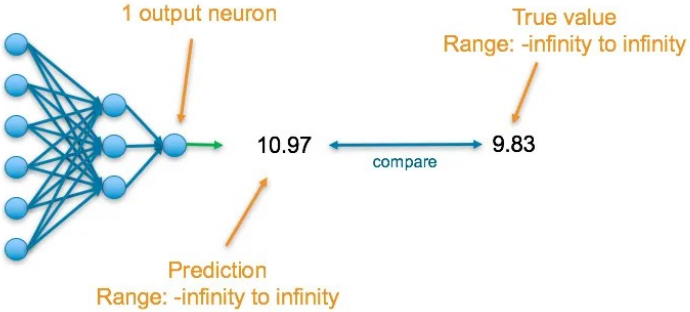


Sigmoid Linear Unit (SiLU)

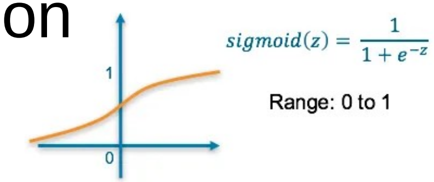


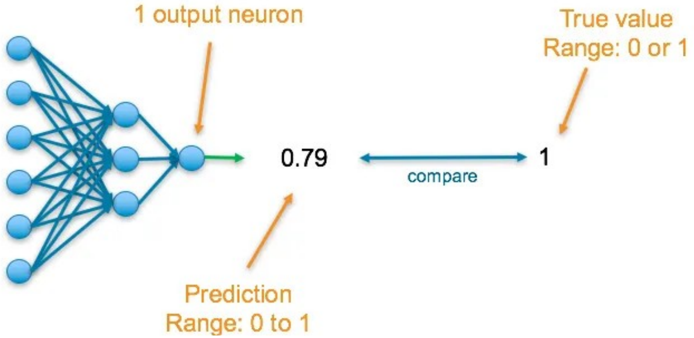
Loss Functions and Activation Functions for output layers

- Regression
  - Linear or ReLU
  - MSE



- Binary classification
  - Sigmoid
  - Binary cross entropy



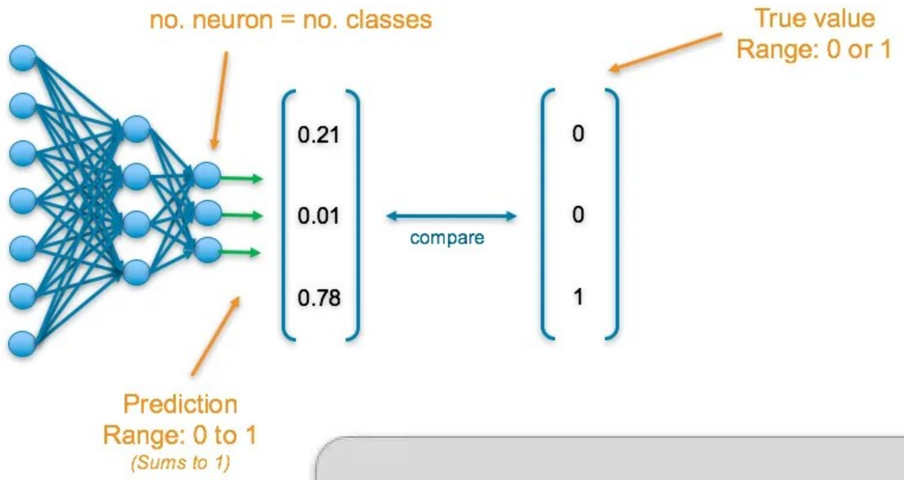


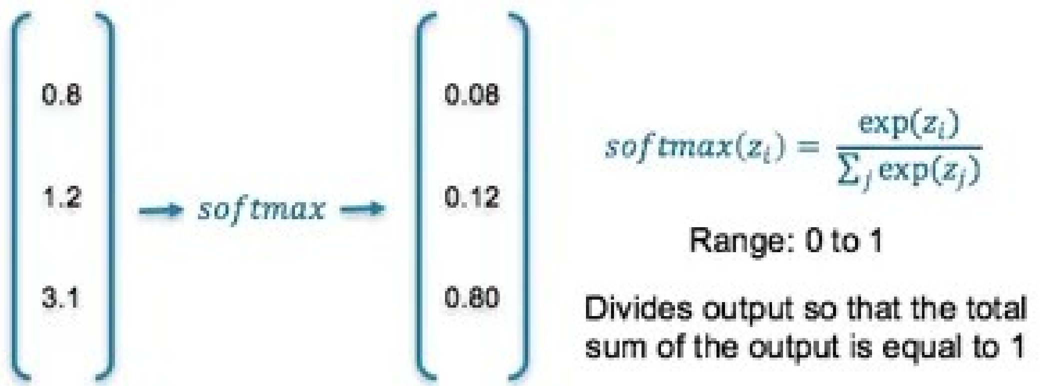
Binary cross entropy = $-(y\log(\hat{y}) + (1-y)\log(1-\hat{y}))$

Where $\hat{y}$ is the predicted value and y is the true value

Loss Functions and Activation Functions for output layers

- Categorial classification
  - Each input belongs to exactly one class



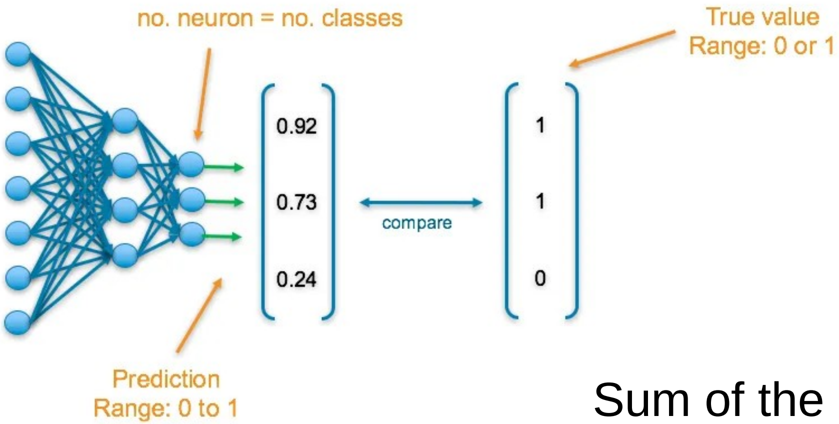


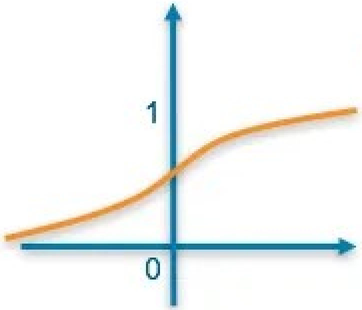
Cross entropy = $-\sum_i^M y_i \log(\hat{y}_i)$

Where $\hat{y}$ is the predicted value, y is the true value and M is the number of classes

Loss Functions and Activation Functions for output layers

- Categorial classification
  - Each input can belong to multiple classes





$$sigmoid(z) = \frac{1}{1 + e^{-z}}$$

Range: 0 to 1

Sum of the binary cross entropies of the output neurons

Binary cross entropy = $-\sum_i^M (y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i))$

Where $\hat{y}$ is the predicted value and y is the true value

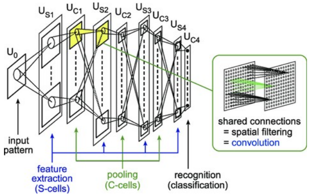
Part 2 – Neural Networks for bioimage analysis

Convolutional Neural Networks - Timeline

Neural network model for a mechanism of pattern recognition unaffected by shift in position — Neocognitron —

1979

Neocognitron

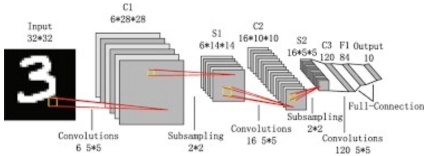


Fukushima, Kunihiko

"Gradient-based learning applied to document recognition"

1998

LeNet

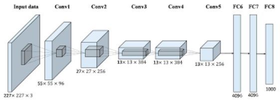


Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P.

AlexNet does well in ImageNet Large Scale Visual Recognition Challenge

2012

AlexNet

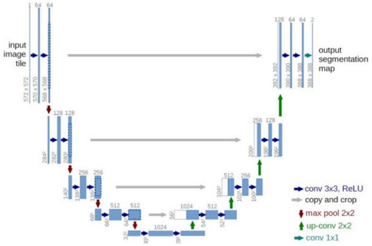


Alex Krizhevsky

U-Net: Convolutional Networks for Biomedical Image Segmentation

2015

Unet



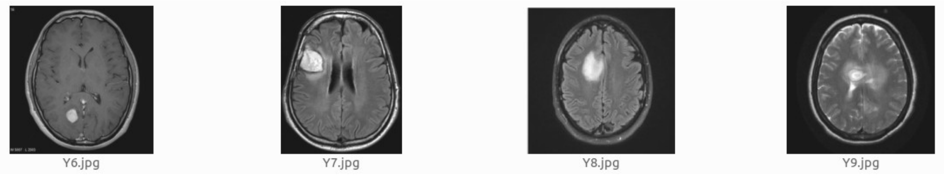
Olaf Ronneberger, Philipp Fischer, and Thomas Brox

ANNs in Image Analysis

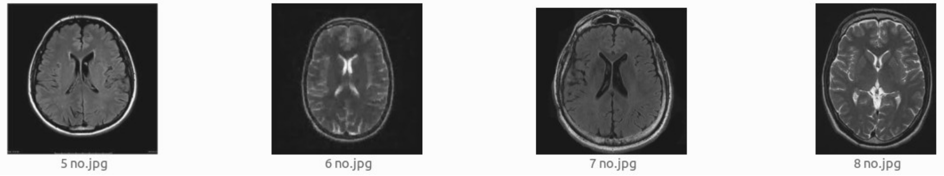
- image classification
- detection + tracking + object classification
- semantic segmentation
- instance segmentation
- image transformation

- **Image classificaton**
  - Classify the image as a whole (cat, dog, …)
  - Input: image
  - Output: Label / Probability for class

Tumor



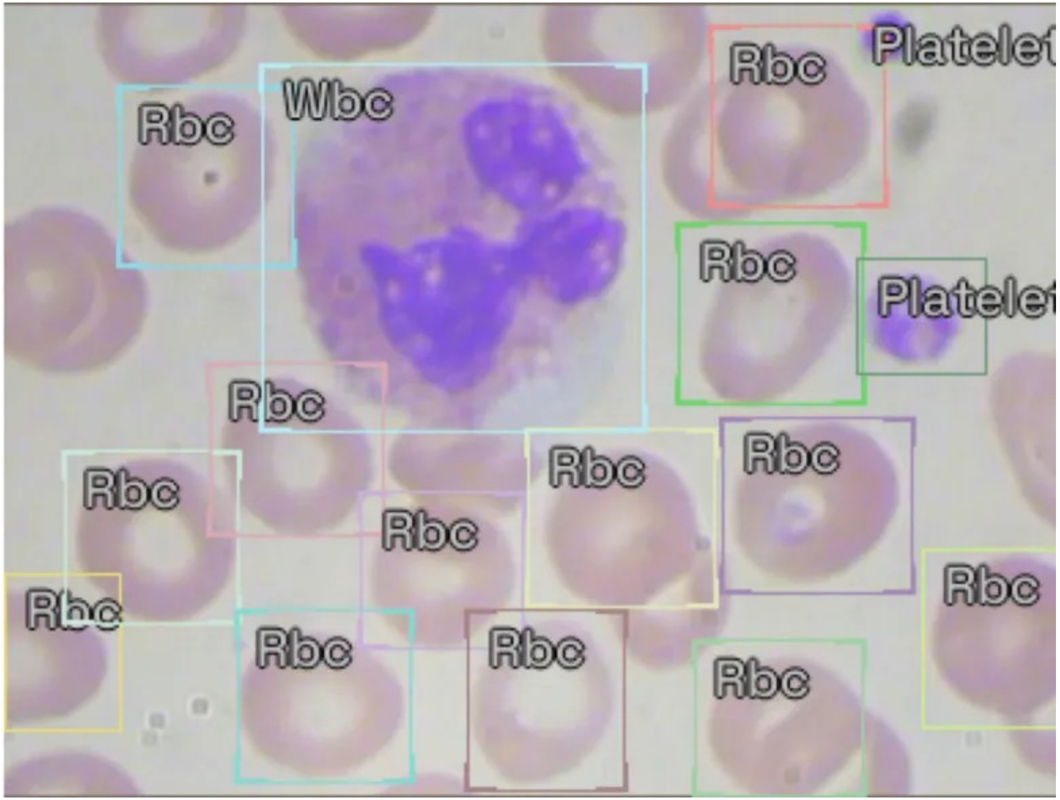
No Tumor



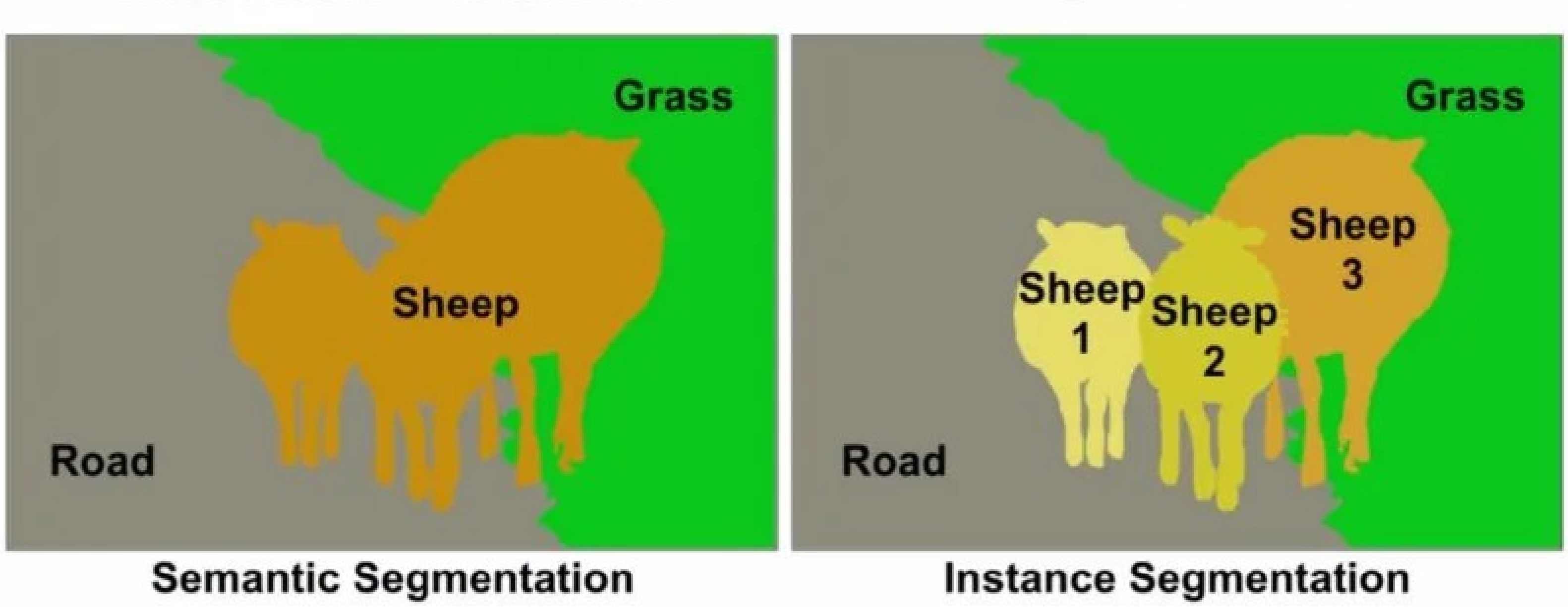
Object Detection and classification

- Find bounding boxes of objects and classify objects
- Input: Image
- Output:
  bounding boxes and labels

Detection and classification of blood cells

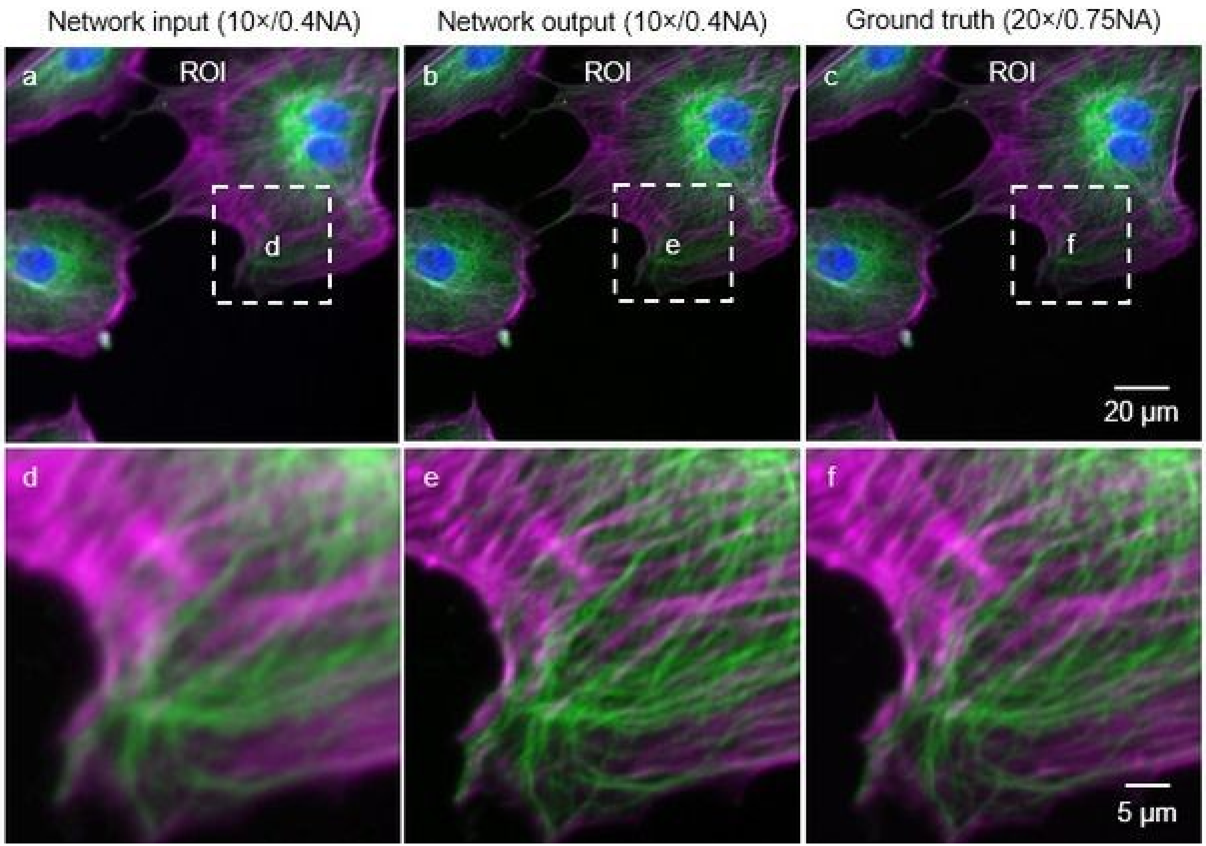


Segmentation



- Input: image
- Output: mask or index mask or probability maps for each class

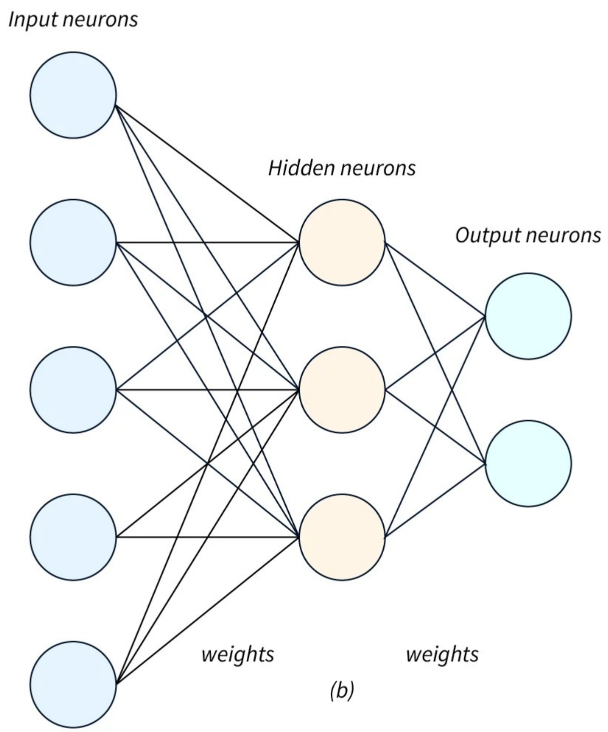
Image transformation



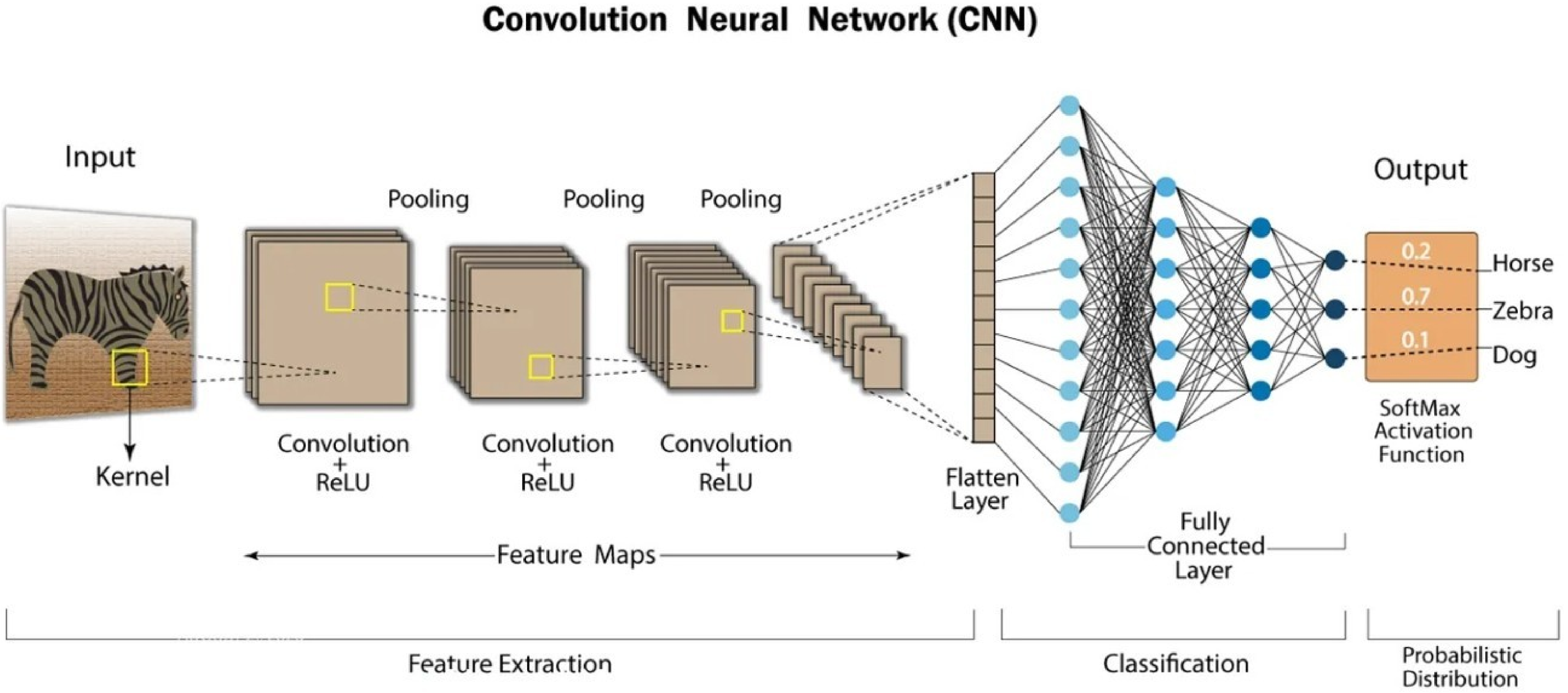
- Input: Image
- Output : Image
  - usually of the same type as the input image
  - the content is transformed, not the image type

MLPs or Fully Connected Neural Networks for image analysis

- Problems:
  - Images can be big
    - A lot of input neurons
    - A lot of connections
      - A lot of parameters
  - The spatial relations of the pixels/voxels are lost
  - The networks must spontaneously learn to extract useful features at the right scales
- Solution :
  - Convolutional Neural Networks
    - Add convolutional layers and pooling layers before the fully connected part

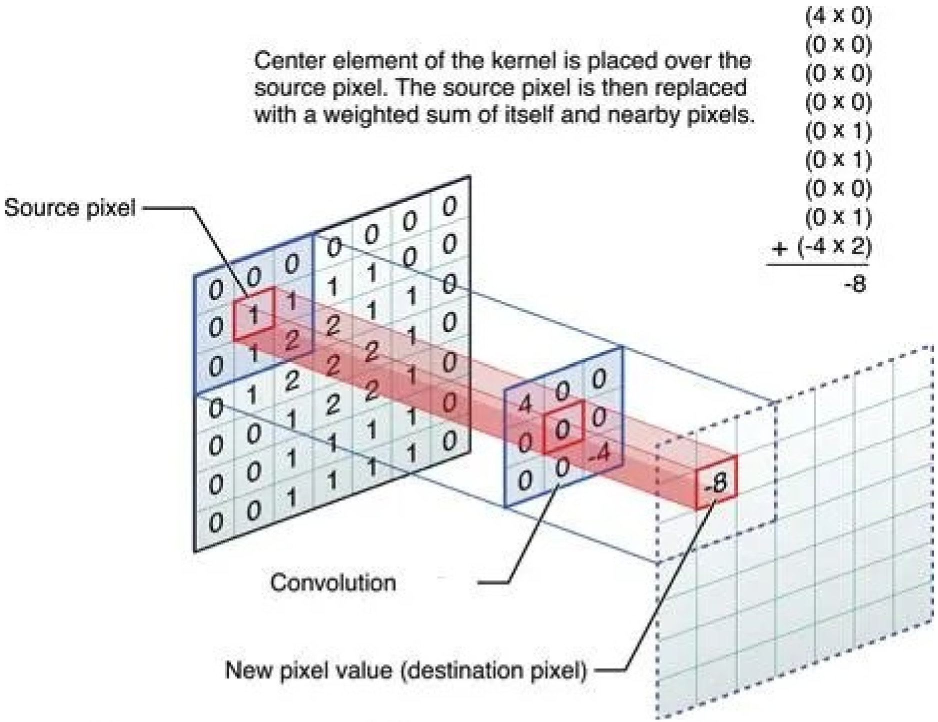


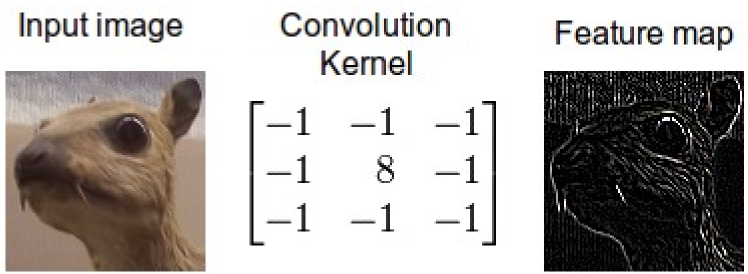
CNNs



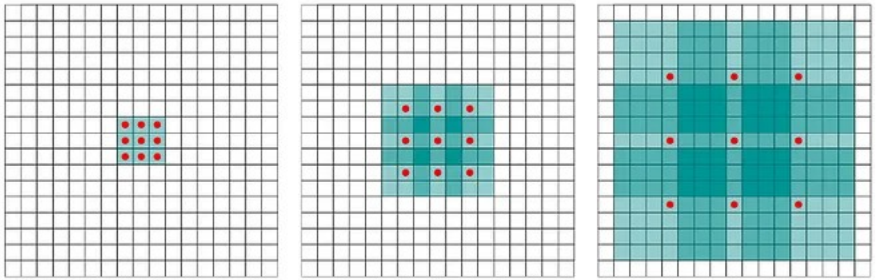
Convolution Neural Network (CNN)
Input
Kernel
Pooling
Pooling
Pooling
Convolution + ReLU
Convolution + ReLU
Convolution + ReLU
Feature Maps
Flatten Layer
Fully Connected Layer
Output
0.2
0.7
0.1
Horse
Zebra
Dog
SoftMax Activation Function
Feature Extraction
Classification
Probabilistic Distribution

Convolution layer



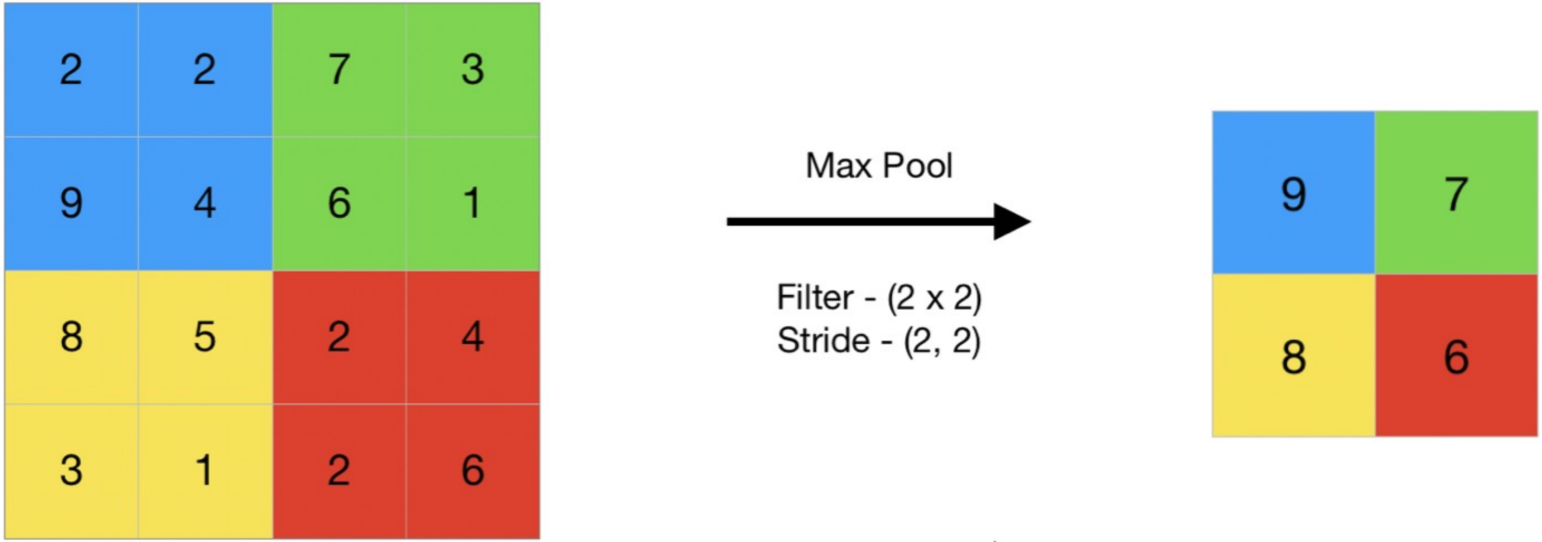


- Values at the borders are missing
  - Padding
  - Shrink result image
- Hyperparameters
  - nr. of filters (feature maps, convolutions)
  - kernel_size (nxm)
  - Strides (pxq)
    - The distance the kernel moves in each step
  - Padding
  - Dilation



Dilated convolution

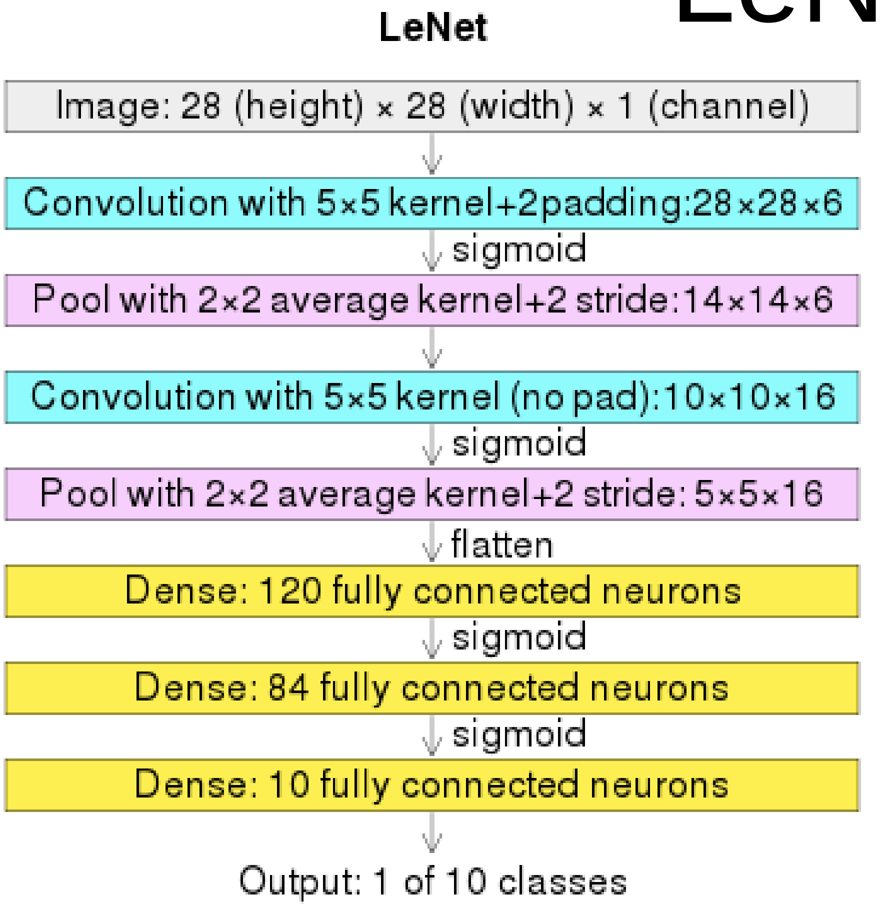
Pooling Layers



- Max pooling / Average pooling
  - Reduce size
  - Local shift invariance
  - Keep most significant info

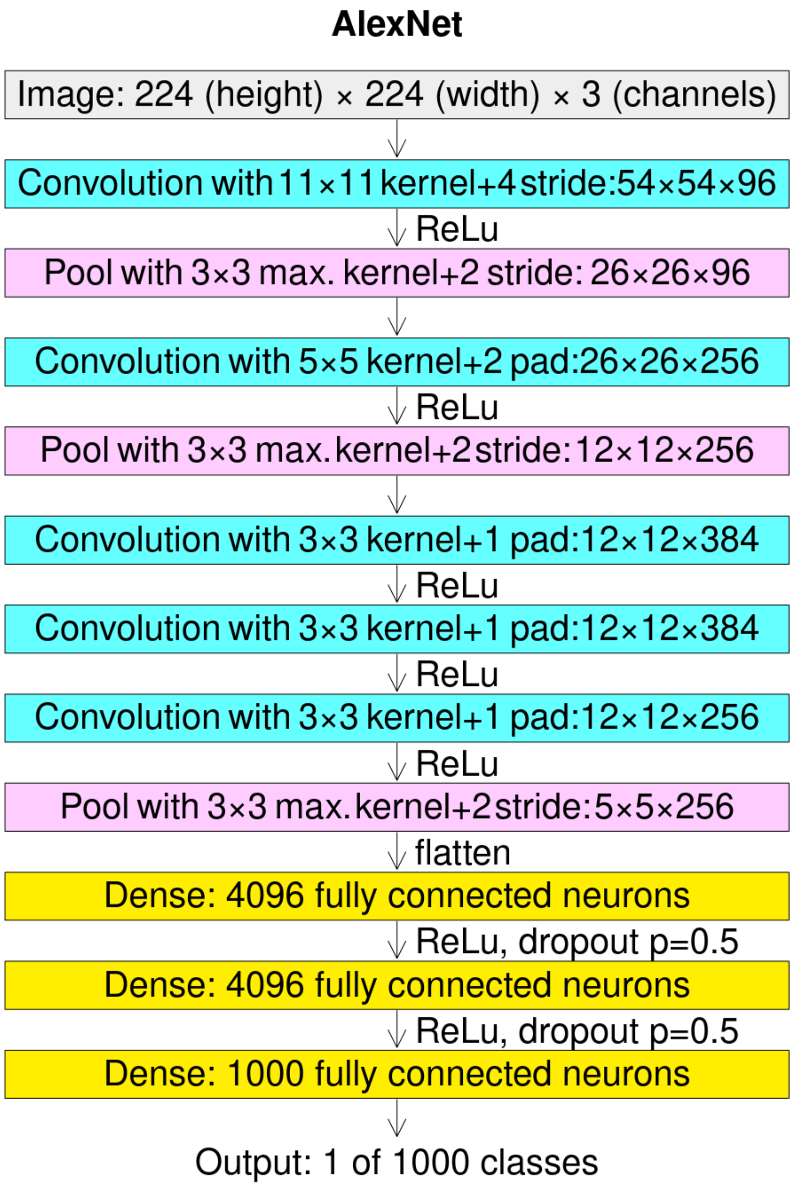
- Hyperparameters
  - Pool size (nxm)
  - Stride
    - Often equal to pool size

CNNs examples
LeNet - 1989

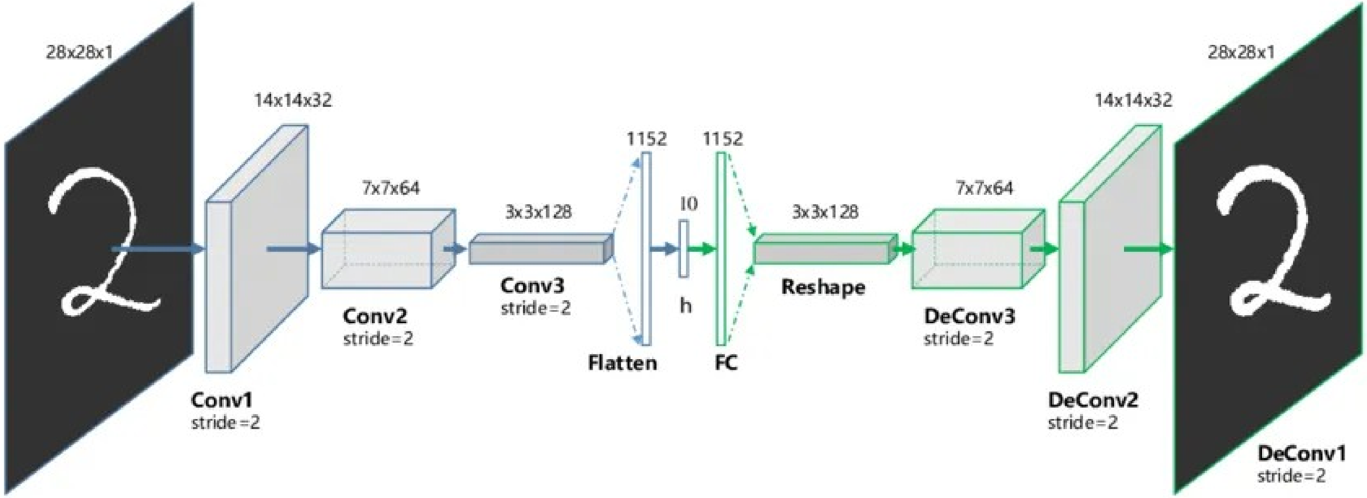


- Yann LeCun
- Recognition of handwritten digits

AlexNet 2012, ImageNet

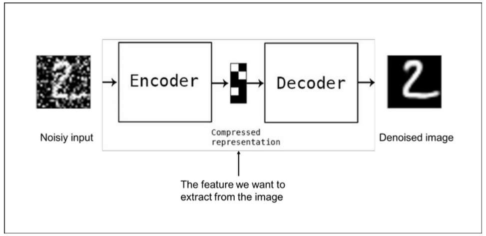


Autoencoders

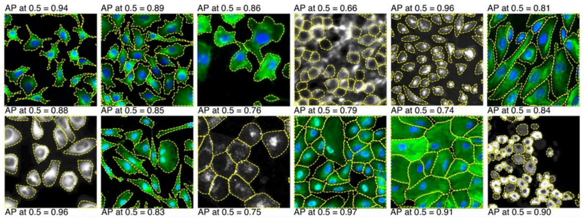


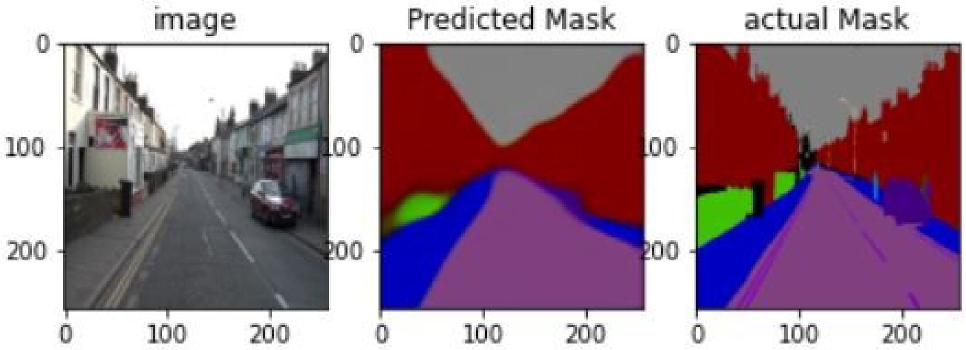
- What is learned was originally the compressed version of the input h
- However autoencoders are also used for :
  - Finding feature sets
  - Principal component analysis
  - De-noising of images
  - ...

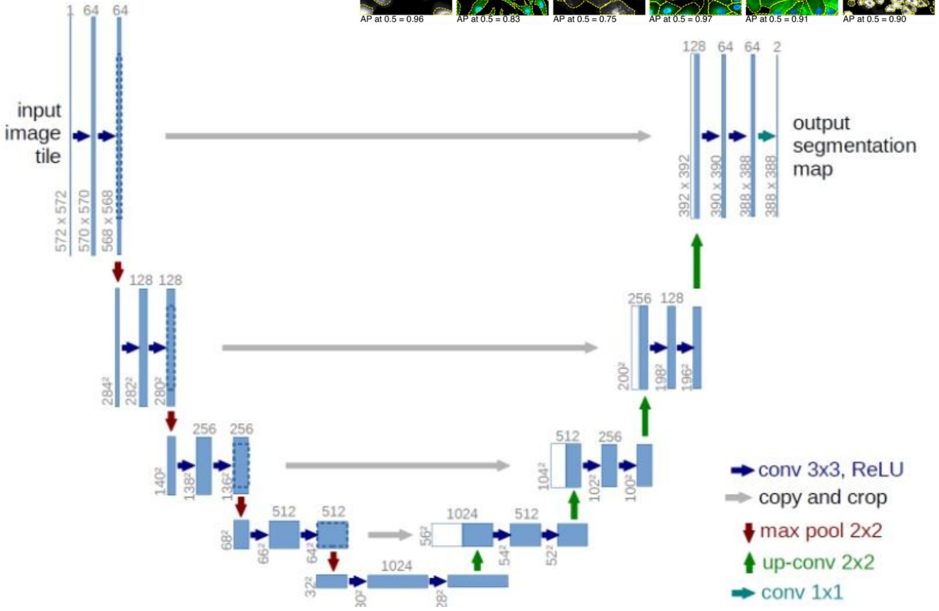
- Unsupervised
  - Encoder creates a compressed version h of the input
  - Decoder reconstructs h to create the output
  - Error is calculated between the input and its reconstructed version



Unet







- Problems for semantic segmentation in CNNs
  - The scale information is lost, everything is based on the smallest feature maps
- Unet
  - Supervised
  - Autoencoder architecture with interconnections between encoder and decoder layers
  - Fully convolutional neural network

- Unet can directly be used for semantic segmentation
- Unet is also the basis of instance segmentation networks, like
  - stardist
  - cellpose
  - ...