Machine Intelligence:: Deep Learning
Week 2

Beate Sick, Oliver Dürr

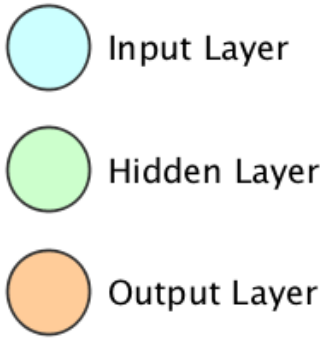
Institut für Datenanalyse und Prozessdesign
Zürcher Hochschule für Angewandte Wissenschaften

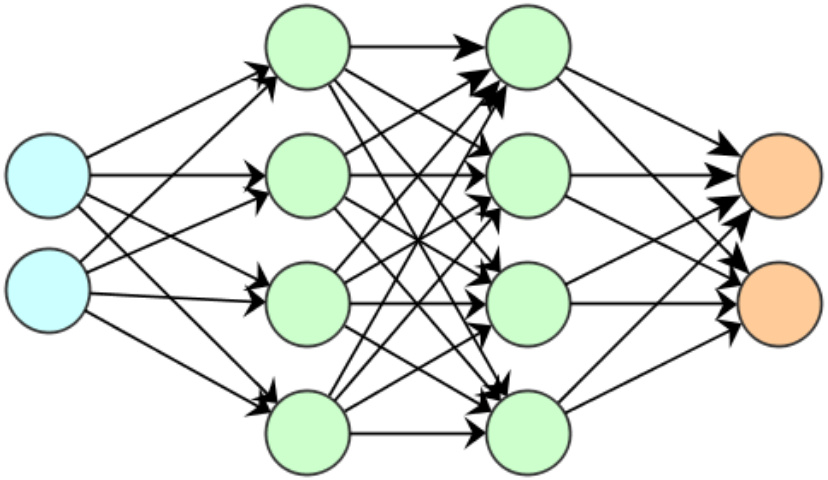
Topics of today

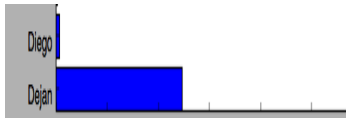
- Keras
- A second look on fully connected Neural Networks (fcNN)
- Convolutional Neural Networks (CNN) for images
  - Motivation for switching from fcNN to CNNs
  - Introduction of convolution
  - ReLu and Maxpooling Layer
  - Building CNNs

Deep Learning Frameworks

Recap: The first network







- The input: e.g. intensity values of pixels of an image
  - (Almost) no pre-processing
- Information is processed layer by layer from building blocks
- Output: probability that image belongs to certain class
- Arrows are weights (these need to be learned / training)

Deep Learning Frameworks (common)

- Computation needs to be done on GPU or specialized hardware (compute performance)
- On GPU: almost exclusively on NVIDIA using the cuda library, cudnn
- Data Structure are multidimensional arrays (*tensors*) which are manipulated
- Learning require to calculate derivatives of the network w.r.t parameters.

In this course: TensorFlow with Keras

Low Level Deep Learning Libraries for Tensor Manipulations

- TensorFlow
  - Open sourced by Google Dec 2015
  - TF 1.x static computational graphs
  - Since version 2.0 also dynamic computational graphs in eager mode
- Torch / pytorch
  - Facebook, quite flexible, lua (Jan 2017 also in python, **pytorch**)
  - Dynamic computational graph
- JAX
  - New kid on the block
  - Numpy on stereoids (GPU / TPU replacement)
- Chainer
  - Flexible, build graph on the fly
- MXNet
  - Can be used in many languages, build graph on the fly?
- Caffe
  - Inflexible, good of CV
  - Calculate gradients by hand
- Theano
  - Around since 2008,
  - Active development abandoned
  - Slow compiling of graph (due to optimization)

We will work (mainly) with high level libraries Keras ontop of TensorFlow

TensorFlow

What is TensorFlow

- It's API about ***tensors***, which ***flow*** in a ***computational graph***

https://www.tensorflow.org/

- What are **tensors**?

What is a tensor?

In this course we only need the simple and easy accessible definition of Ricci:

Definition. A tensor of type (*p*, *q*) is an assignment of a multidimensional array

$$T^{i_1 \dots i_p}_{j_1 \dots j_q}[\mathbf{f}]$$

to each basis $\mathbf{f} = (\mathbf{e}_1, \dots, \mathbf{e}_n)$ of a fixed *n*-dimensional vector space such that, if we apply the change of basis

$$\mathbf{f} \mapsto \mathbf{f} \cdot R = (\mathbf{e}_i R^i_1, \dots,$$

Just kidding...

then the multidimensional array obeys the transformation law

$$T^{i'_1 \dots i'_p}_{j'_1 \dots j'_q}[\mathbf{f} \cdot R] = (R^{-1})^{i'_1}_{i_1} \cdots (R^{-1})^{i'_p}_{i_p} \, T^{i_1, \dots, i_p}_{j_1, \dots, j_q}[\mathbf{f}] \, R^{j_1}_{j'_1} \cdots R^{j_q}_{j'_q}.$$

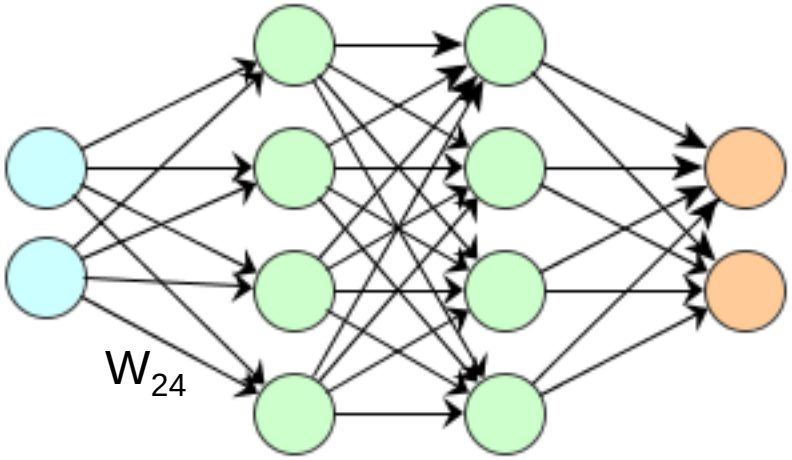
Sharpe, R. W. (1997). Differential Geometry: Cartan's Generalization of Klein's Erlangen Program. Berlin, New York: Springer-Verlag. p. 194. ISBN 978-0-387-94732-7.

What is a tensor?

For TensorFlow: A tensor is an array with several indices (like in numpy).
Order (a.k.a rank) are number of indices and shape is the range.

```
In [1]: import numpy as np

In [2]: T1 = np.asarray([1,2,3]) #Tensor of order 1 aka Vector
        T1

Out[2]: array([1, 2, 3])

In [3]: T2 = np.asarray([[1,2,3],[4,5,6]]) #Tensor of order 2 aka Matrix
        T2

Out[3]: array([[1, 2, 3],
               [4, 5, 6]])

In [4]: T3 = np.zeros((10,2,3)) #Tensor of order 3 (Volume like objects)

In [6]: print(T1.shape)
        print(T2.shape)
        print(T3.shape)

        (3,)
        (2, 3)
        (10, 2, 3)
```

Typical Tensors in Deep Learning

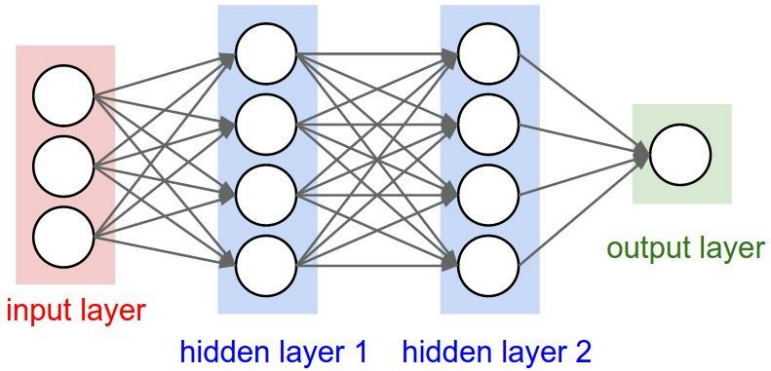


- The input can be understood as a vector
- A mini-batch of size 64 of input vectors can be understood as tensor of order 2
  - (index in batch, x_j)
- The weights going from e.g. Layer L_1 to Layer L_2 can be written as a matrix (often called W)
- A mini-batch of size 64 images with 256,256 pixels and 3 color-channels can be understood as a tensor of order 4.

Introduction to Keras

Keras as High-Level library to TensorFlow

- We use Keras as high-level library
- Libraries make use of the Lego like block structure of networks



High Level Libraries

- Keras
  - Keras is now part of TF core
  - https://keras.io/

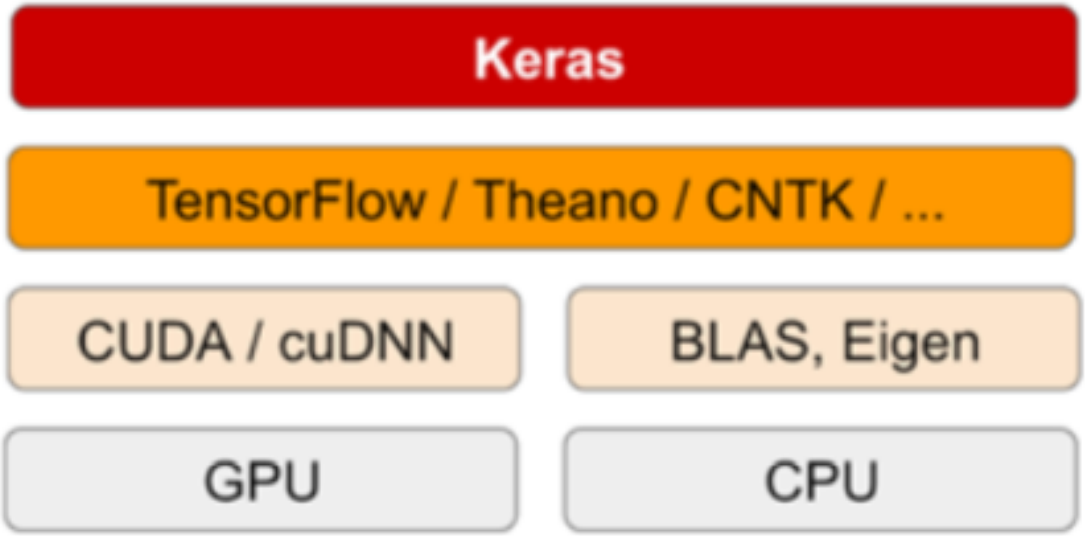


Figure: From Deep Learning with Python, Francois Chollett

The Keras user experience [marketing]

The Keras user experience

Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

This makes Keras easy to learn and easy to use. As a Keras user, you are more productive, allowing you to try more ideas than your competition, faster -- which in turn helps you win machine learning competitions.

This ease of use does not come at the cost of reduced flexibility: because Keras integrates with lower-level deep learning languages (in particular TensorFlow), it enables you to implement anything you could have built in the base language. In particular, as tf.keras, the Keras API integrates seamlessly with your TensorFlow workflows.

https://web.stanford.edu/class/cs20si/lectures/march9guestlecture.pdf

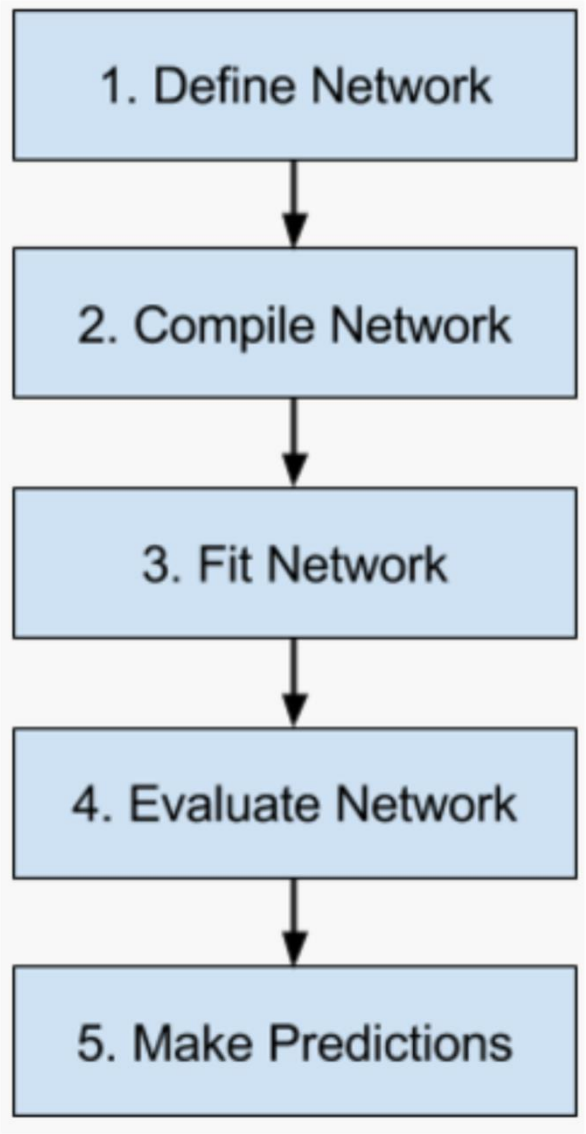
[marketing]

Keras is multi-backend, multi-platform

- Develop in Python, R
  - On Unix, Windows, OSX
- Run the same code with…
  - TensorFlow
  - CNTK
  - Theano
  - MXNet
  - PlaidML
  - ??
- CPU, NVIDIA GPU, AMD GPU, TPU…

https://web.stanford.edu/class/cs20si/lectures/march9guestlecture.pdf

Keras Workflow



Define the network (layerwise)

Add loss and optimization method

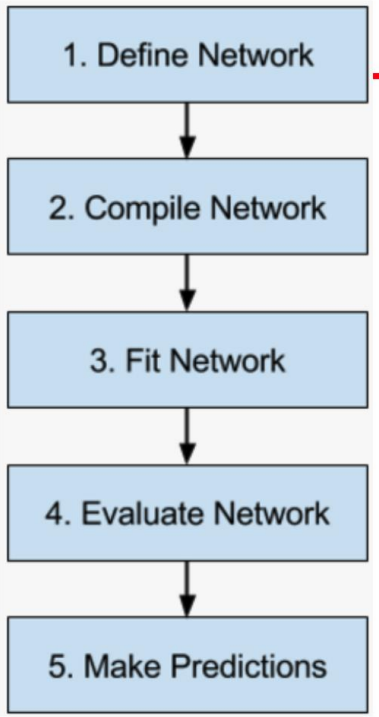
Fit network to training data

Evaluate network on test data

Use in production

A first run through

Define the network

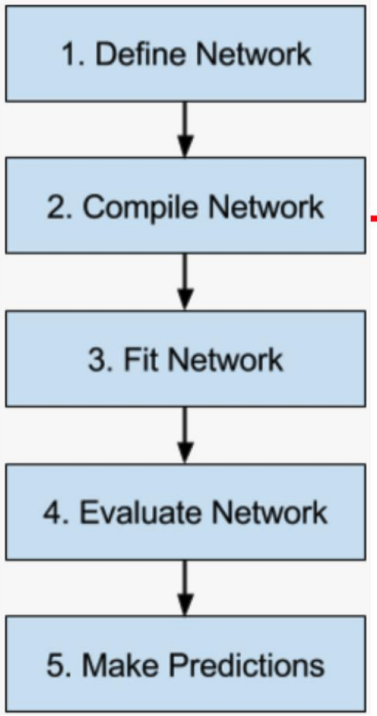



```
# define fcNN with 2 hidden layers
model = Sequential()

model.add(Dense(100, batch_input_shape=(None, 784)))
model.add(Activation('sigmoid'))
model.add(Dense(50))
model.add(Activation('sigmoid'))
model.add(Dense(10))
model.add(Activation('softmax'))
```

Input shape needs to be defined only at the beginning.

Alternative: `input_dim=784`

Compile the network



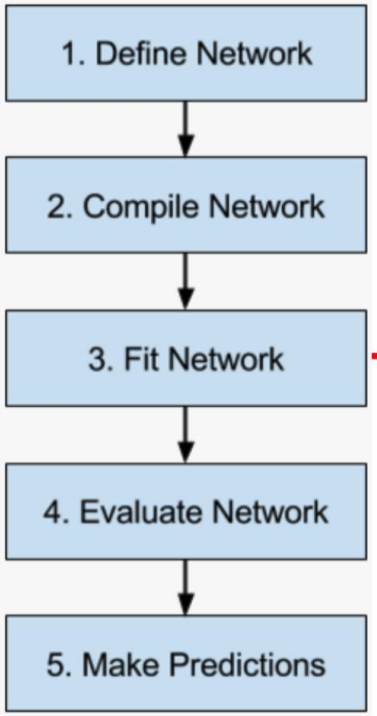
loss function that will be minimized

```
model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])
```

easiest optimizer is SGD
(stochastic gradient descent)

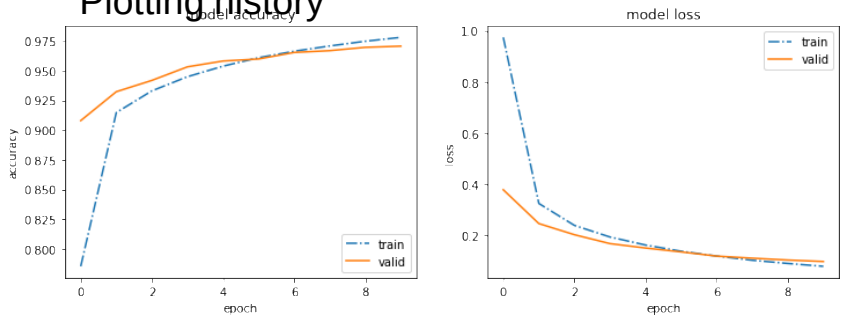
Which metrics besides 'loss' do we want to collect during training

Fit the network

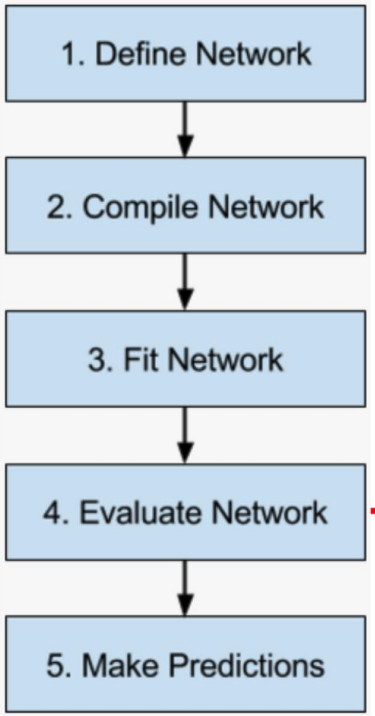



```
# Training of the network
history = model.fit(X, Y,
                    epochs=400,
                    batch_size=128,
                    verbose=0)
```

Plotting history



Evaluate the network




```
model.evaluate(X[0:2000], convertToOneHot(y[0:2000],10))
```

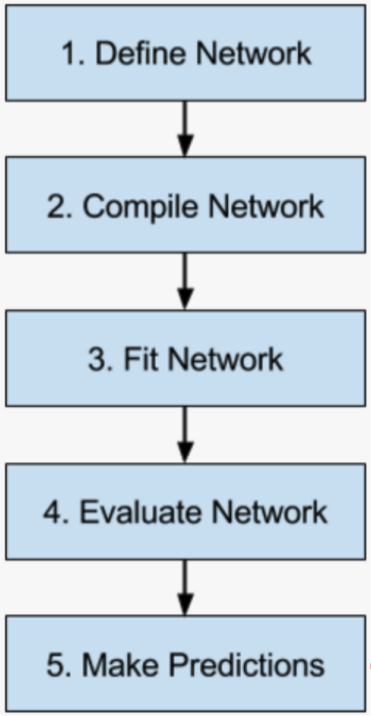
```
2000/2000 [==============================] - 0s 215us/step

[2.5549037284851073, 0.1085]
```

loss

metrics: accuracy

```
model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])
```

Make Predictions




```
model.predict_classes(X[0:10])
```

```
array([0, 9, 9, 1, 6, 9, 8, 9, 0, 0])
```

Demo Time Example in Keras [only if time permits]

Demo Time: https://github.com/tensorchiefs/dl_book/blob/master/chapter_02/nb_ch02_02a.ipynb

Colab https://colab.research.google.com/github/tensorchiefs/dl_book/blob/master/chapter_02/nb_ch02_02a.ipynb

Building NN (with keras)

- Lego Blocks (Layers)
- Way of stacking them together API Style

Three API styles

- The Sequential Model
  - Dead simple
  - Only for single-input, single-output, sequential layer stacks
  - Good for 70+% of use cases
- The functional API
  - Like playing with Lego bricks
  - Multi-input, multi-output, arbitrary static graph topologies
  - Good for 95% of use cases
- Model subclassing
  - Maximum flexibility
  - Larger potential error surface

Sequential API

The Sequential API

```
import keras
from keras import layers

model = keras.Sequential()
model.add(layers.Dense(20, activation='relu', input_shape=(10,)))
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.fit(x, y, epochs=10, batch_size=32)
```

Functional (do you spot the error?)

The functional API

```
import keras
from keras import layers

inputs = keras.Input(shape=(10,))
x = layers.Dense(20, activation='relu')(x)
x = layers.Dense(20, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)

model = keras.Model(inputs, outputs)
model.fit(x, y, epochs=10, batch_size=32)
```

Subclassing (do you spot the error?) [for completness]

Model subclassing

```
import keras
from keras import layers

class MyModel(keras.Model):

    def __init__(self):
        super(MyModel, self).__init__()
        self.dense1 = layers.Dense(20, activation='relu')
        self.dense2 = layers.Dense(20, activation='relu')
        self.dense3 = layers.Dense(10, activation='softmax')

    def call(self, inputs):
        x = self.dense1(x)
        x = self.dense2(x)
        return self.dense3(x)

model = MyModel()
model.fit(x, y, epochs=10, batch_size=32)
```

layers

Dense fully connected

```
keras.layers.core.Dense(
        output_dim,
        init='glorot_uniform',
        activation='linear',
        weights=None,
        W_regularizer=None,
        b_regularizer=None,
        activity_regularizer=None,
        W_constraint=None,
        b_constraint=None,
        input_dim=None)
```

https://keras.io/layers/

More layers

- Dropout
  - keras.layers.Dropout
- Convolutional (see lecture on CNN)
  - keras.layers.Conv2D
  - keras.layers.Conv1D
- Pooling (see lecture on CNN)
  - keras.layers.MaxPooling2D
- Recurrent (See Lecture on RNN)
  - keras.layers.SimpleRNNCell
  - keras.layers.GRU
  - keras.layers.LSTM

- Roll your own:
  - Implement keras.layers.Layer class
  - https://keras.io/layers/writing-your-own-keras-layers/

Activation

```
keras.layers.Activation(activation)
```

Applies an activation function to an output.

Arguments

e.g. `'relu'`, `'softmax'`, `'tanh'`,...

- **activation**: name of activation function to use (see: activations), or alternatively, a Theano or TensorFlow operation.

```
from keras.layers import Activation, Dense

model.add(Dense(64))
model.add(Activation('tanh'))
```

This is equivalent to:

```
model.add(Dense(64, activation='tanh'))
```

Note: Activations are also layers

Output Layer

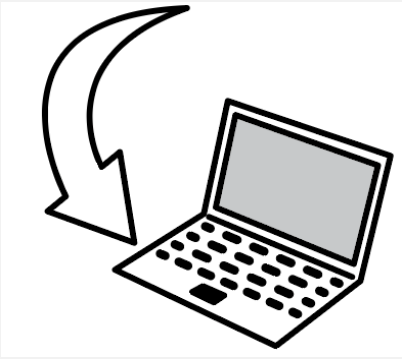
The last layer of the network is a bit special

- Classification
  - # of nodes = # of classes
    - For binary classification sometime only probability of class 1 is reported
  - Usually output is probability for class
    - Use softmax in that case

- Regression (simple distributions)
  - In the interpretation the output of a NN is a probability
  - #nodes = 1
  - Gaussian with fixed variance (the usual regression)
  - Poisson (count data) see later

- Regression (more complicated distributions)
  - #nodes = #number of parameters for distribution

Exercise

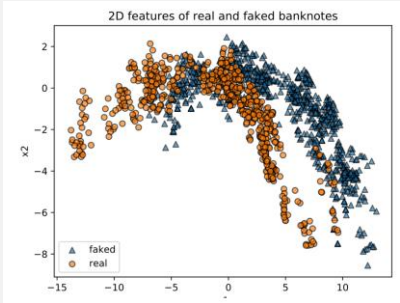


1. Finish NB 01_simple_forward_pass.ipynb have a look at the Keras implementation

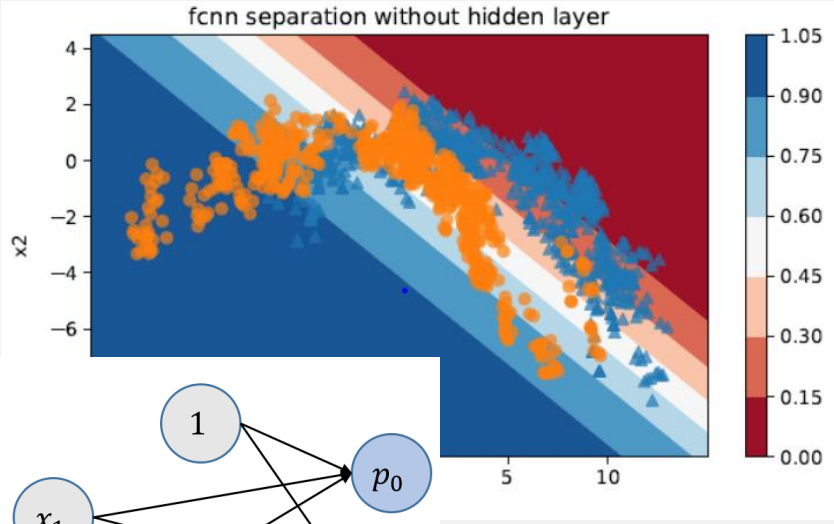
Homework

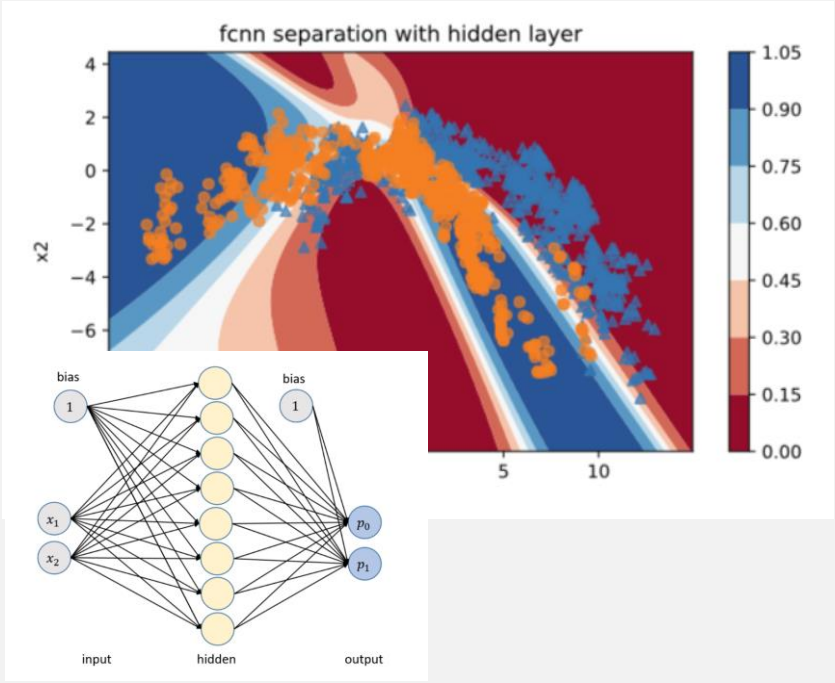
Do exercise NB 02:

Use Keras for modeling the banknotes data with binary outcome and 2 continuous input features (x1,x2)

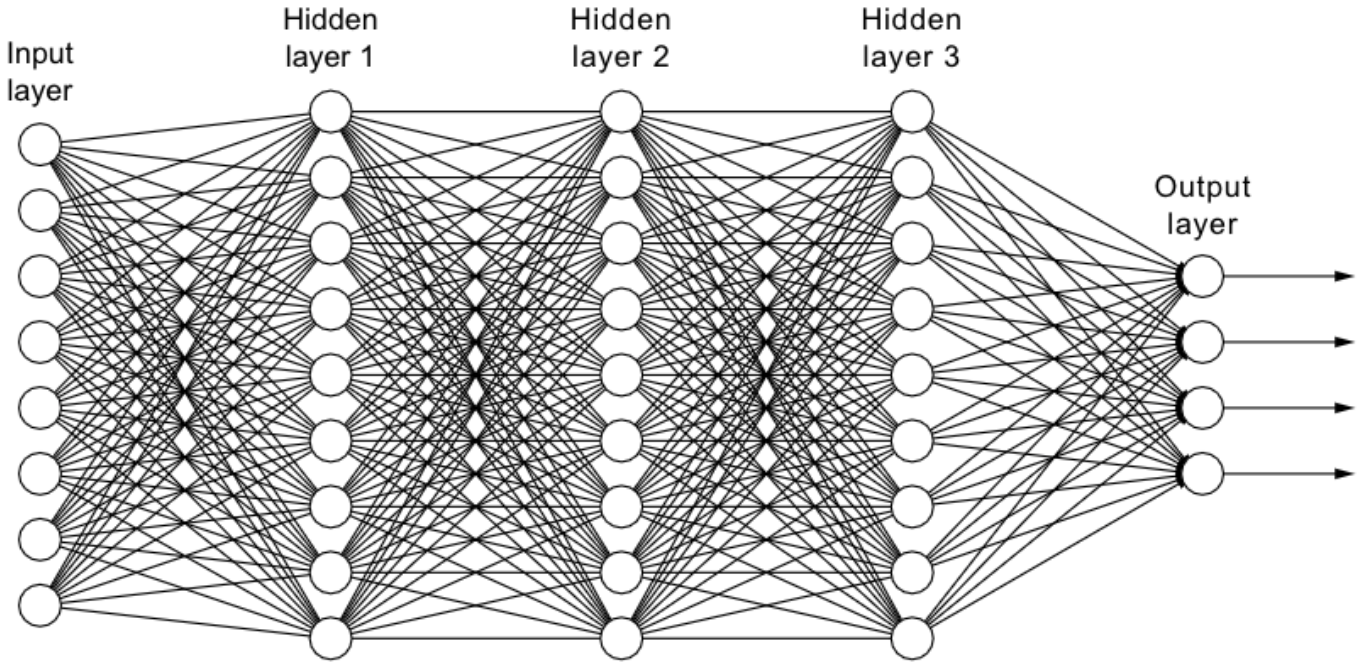


Fit the a fcNN w/o and with hidden layer to discriminate fake bank notes from real bank notes: NB02



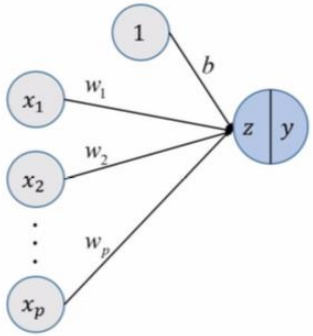


Architecture of a fully connected NN

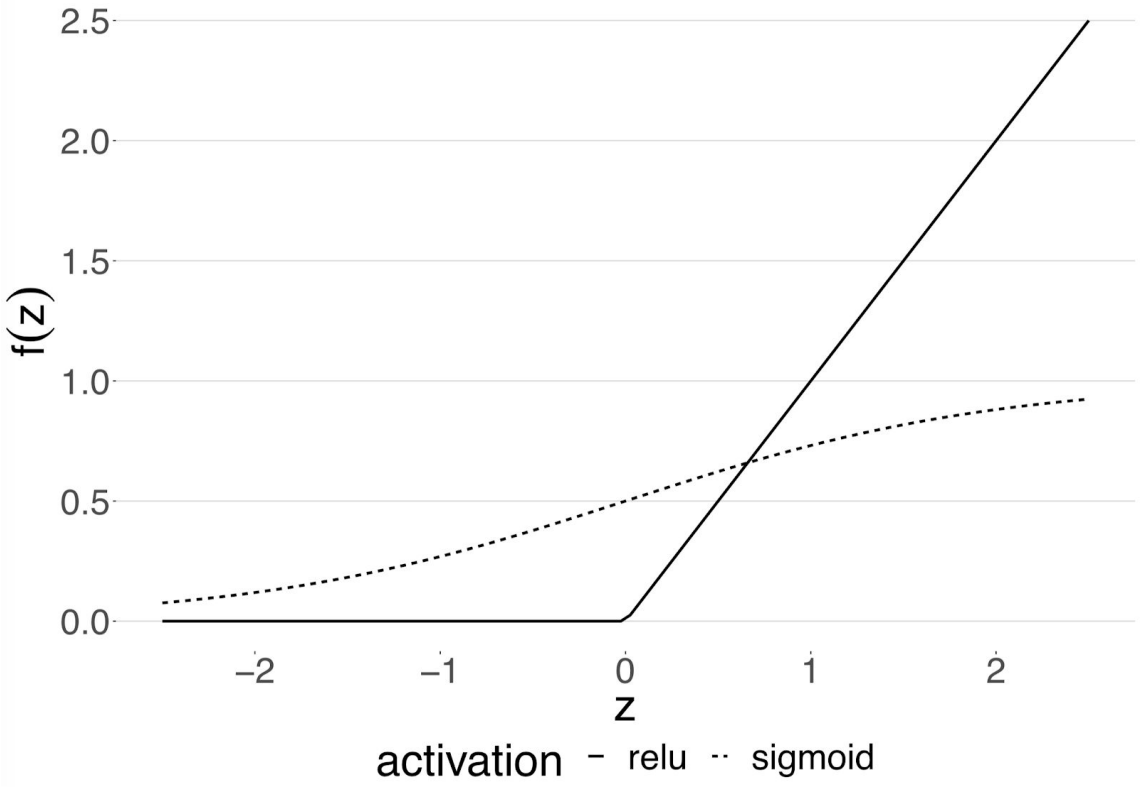


Each neuron in a fcNN gets as input a weighted sum of all neuron activation from one layer below. Different neurons in the same layer have different weights in this weighted sum, which are learned during training.

Comon non-linear activation function

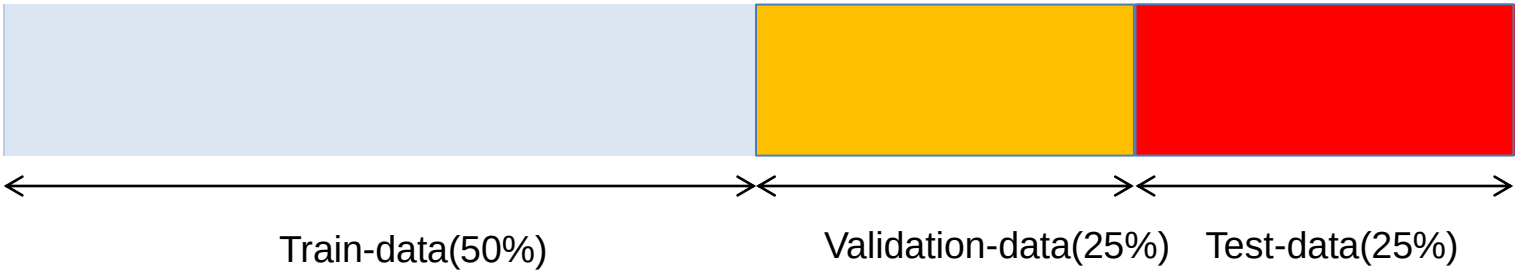


$$y = f(z) = f(b + \sum x_i \cdot w_i)$$



The sigmoid has small gradients for values far away from zero.
ReLu clipps values below zero and let values>0 pass unchanged.

Best practice: Split in Train, Validation, and Test Set

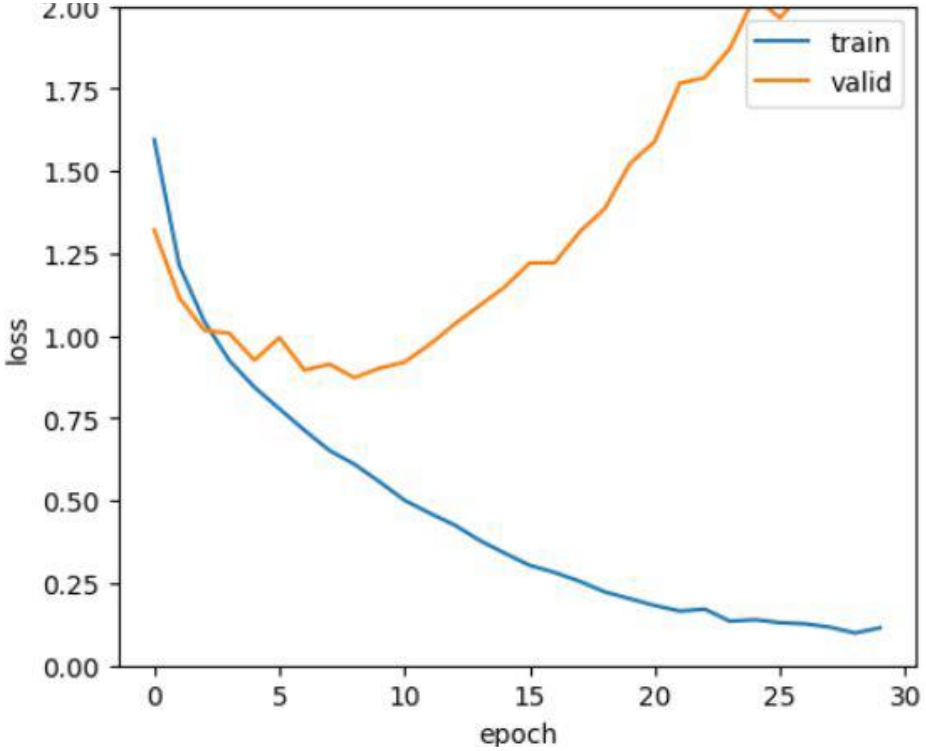


Best practice: Lock an extra test data set away, and use it only at the very end, to evaluate the chosen model, that performed best on your validation set.
Reason: **When trying many models, you probably overfit on the validation set**.

Determine performance metrics, such as MSE, to evaluate the predictions **on new validation or test data**

What can loss curves tell us?

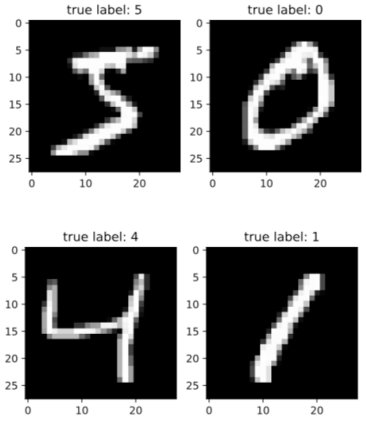
Very common check: Plot loss in train and validation data vs epoch of training.



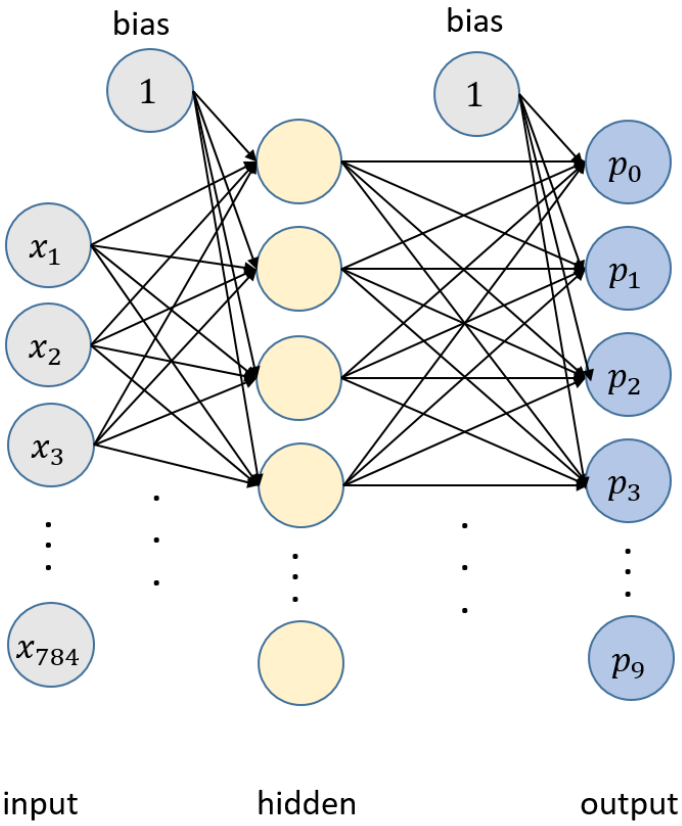
- If training loss does not go down to zero: model is not flexible enough
- In case of overfitting (validation loss >> train loss): regularize model

Fully connected NN for image data
Why not?

A fcNN for MNIST data



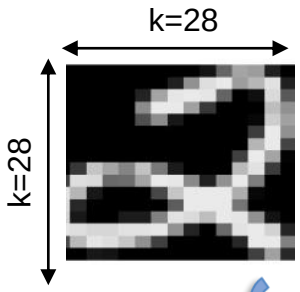
The first four digits of the MNIST data set - each image consisting of 28x28=784 pixels



A fully connected NN with 2 hidden layers.

For the MNIST example, the input layer has 784 values for the 28 x 28 pixels and the output layer has 10 nodes for the 10 classes.

What is going on in a 1 layer fully connected NN?



Flatten to vector with k^2 elements

input $\mathbf{x}^t$

24.0
−5.1
12.2
89.9
:
3.2

$$\mathbf{x}_{(1,k^2)} \cdot \mathbf{W}_{(k^2,10)} + \mathbf{b}_{(1,10)} = \mathbf{z}_{(1,10)}$$

Score or logit $\mathbf{z}^t$

-3.89
-3.18
-0.80
-2.20
-2.44
-1.05
-4.60
-3.48
-2.09
-2.44

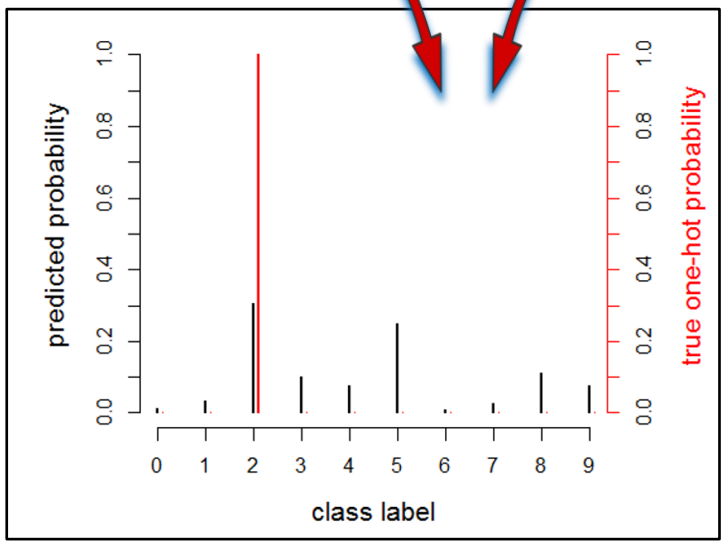
$$\hat{p}_k = \frac{e^{z_k}}{\sum_j e^{zj}}$$

Softmax or probability $\mathbf{p}=S(\mathbf{z})$

0.02
0.04
0.31
0.10
0.08
0.26
0.01
0.03
0.11
0.08

1-hot labels ^{obs}y

0
0
1
0
0
0
0
0
0
0



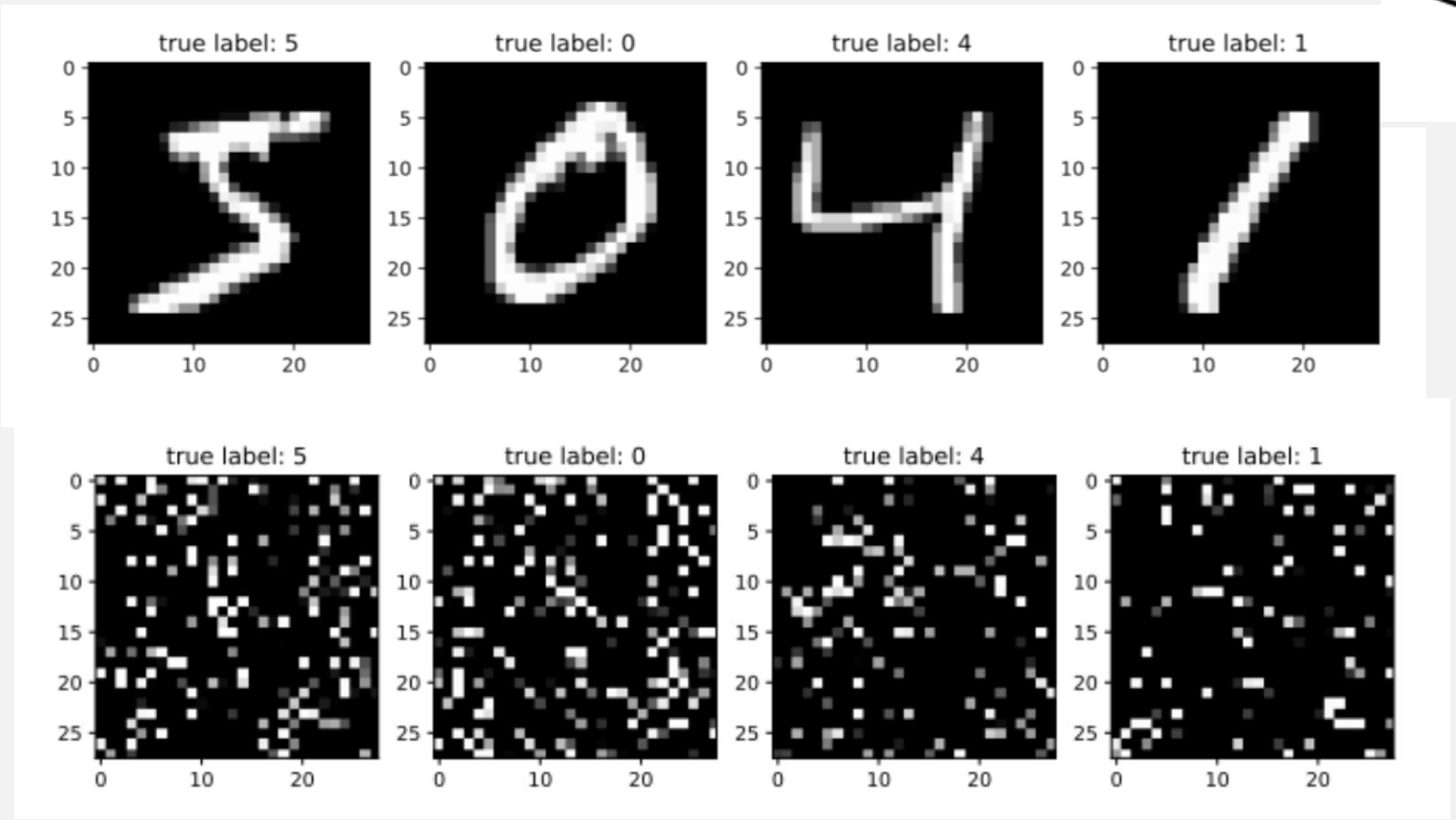
Cost C or Loss = cross-entropy averaged over all images in mini-batch

$$C = \frac{1}{N}\sum_i D(\mathbf{p}_i, \mathbf{y}_i)$$

Cross-Entropy

$$D(\mathbf{p},\mathbf{y}) = -\sum_{k=1}^{10} {}^{obs}y_k \cdot \log(p_k)$$

Exercise: Does shuffling disturb a fcNN?

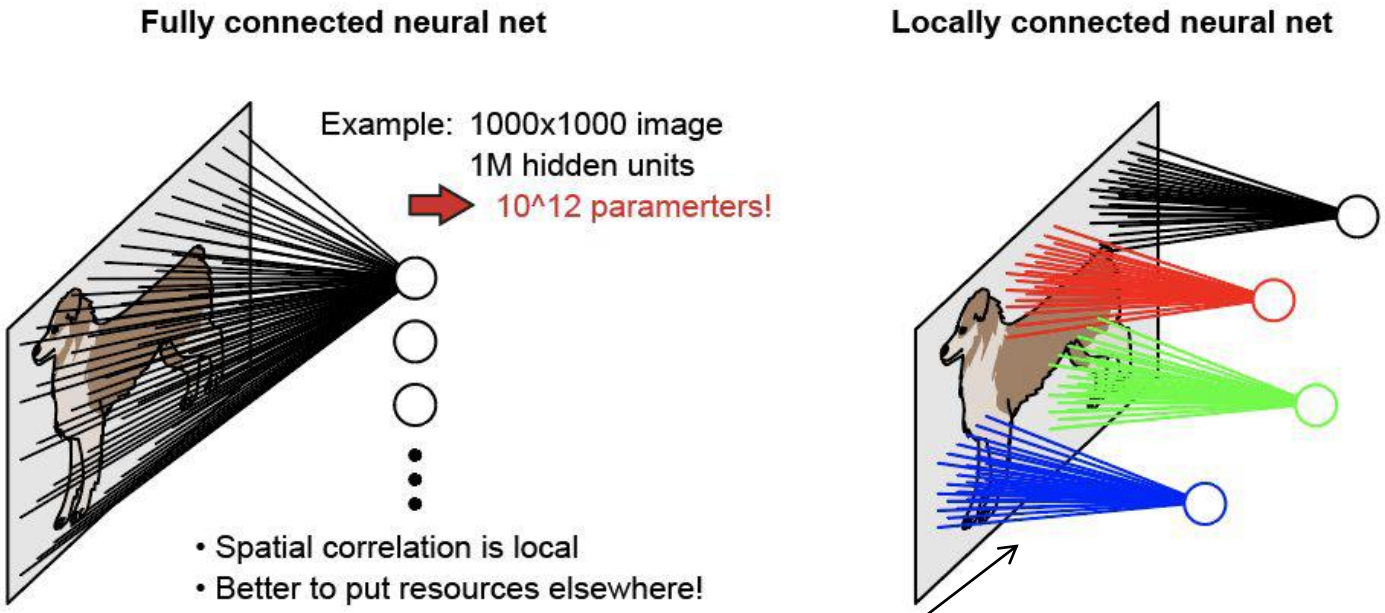


Use fcNN for MNIST:
https://github.com/tensorchiefs/dl_course_2023/blob/master/notebooks/03_fcnn_mnist.ipynb

Inverstigate if shuffling disturbs the fcNN for MNIST:
https://github.com/tensorchiefs/dl_course_2023/blob/master/notebooks/04_fcnn_mnist_shuffled.ipynb

Convolutional Neural Networks
SoA for image data

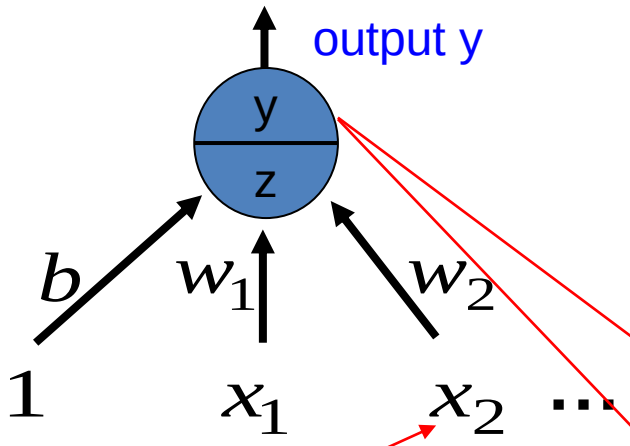
Convolution extracts local information using few weights



Shared weights:

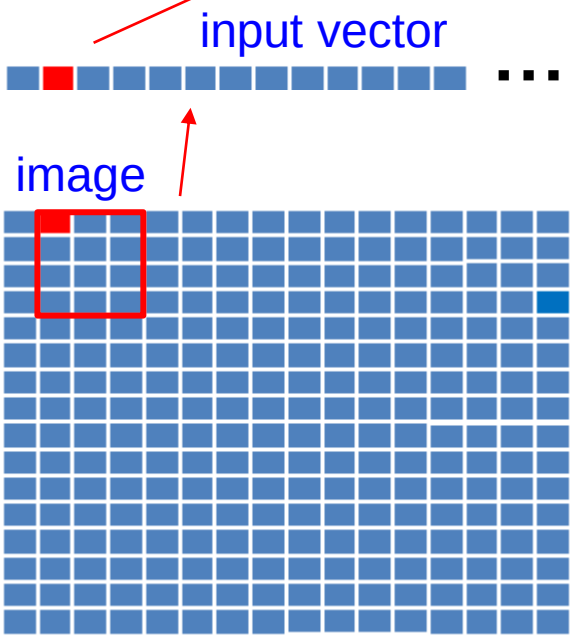
by using the same weights for each patch of the image we need much less parameters than in the fully connected NN and get from each patch the same kind of local feature information such as the presence of a edge.

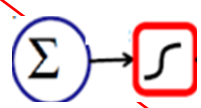
An artificial neuron

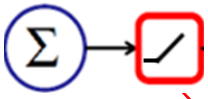


bias weights

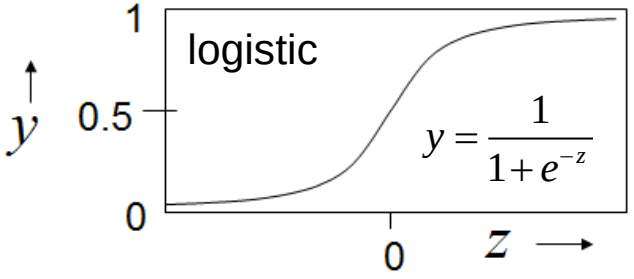
$$z = b + \sum_i x_i w_i$$

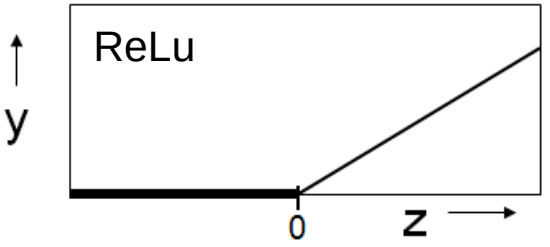






Different non-linear transformations are used to get from z to output y





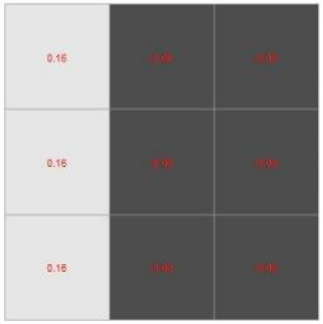
Convolutional networks use neighborhood information and replicated local feature extraction

In a locally connected network the calculation rule

$$z = b + \sum_i x_i w_i$$

Pixel values in a small image patch are element-wise multilied with weights of a small filter/kernel:

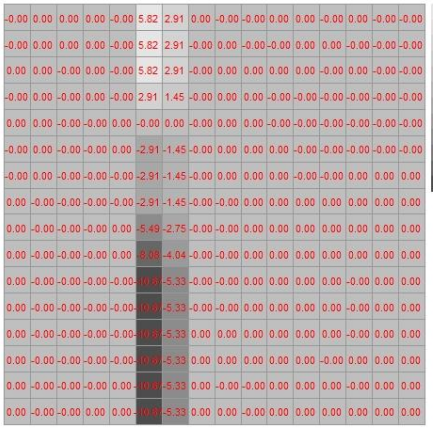
w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

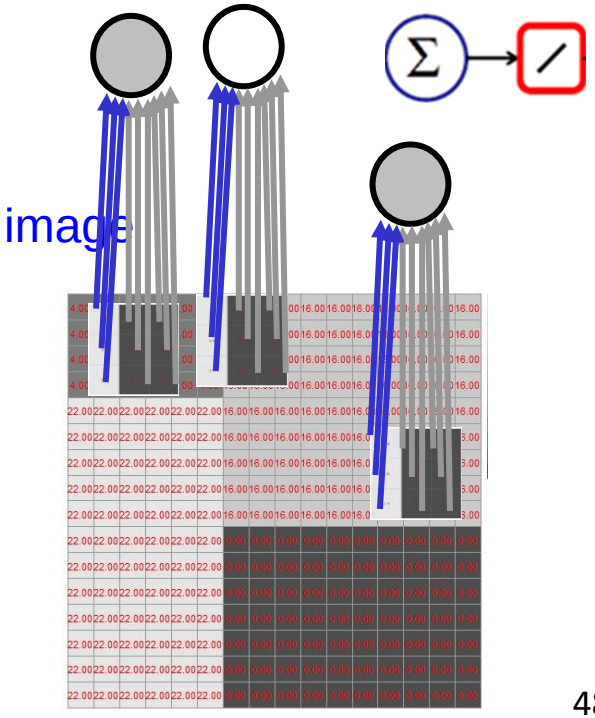


The filter is applied at each position of the image and it can be shown that the result is maximal if the image pattern corresponds to the weight pattern.

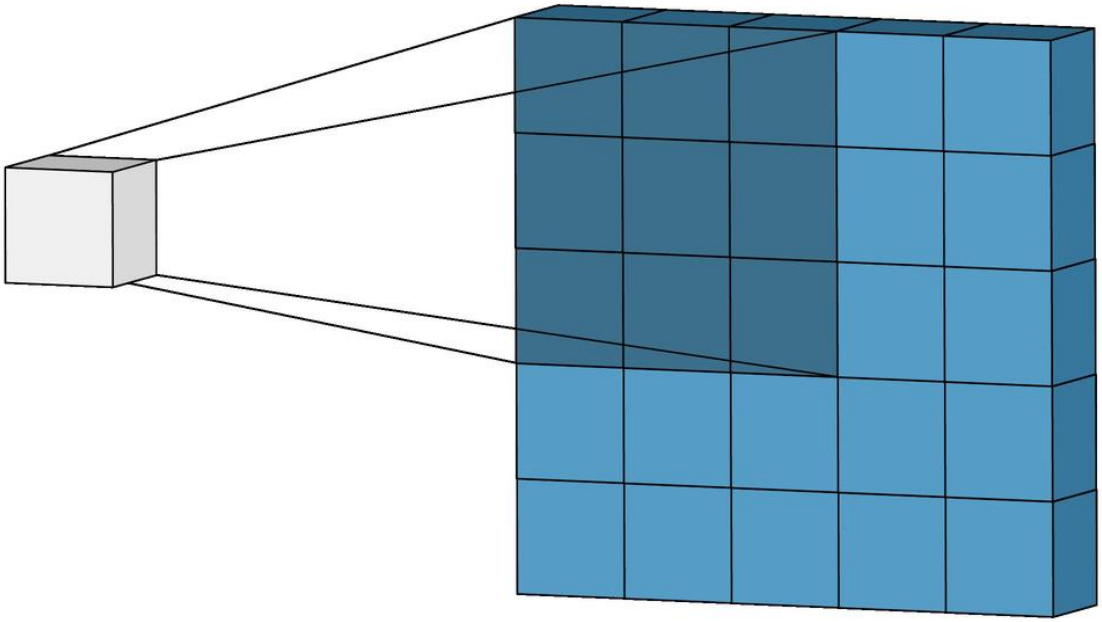
The results form again an image called feature map (=activation map) which shows at which position the feature is present.

feature/activation map





Applying the same 3x3 kernel at each image position



Applying the 3x3 kernel on a certain position of the image yields one pixel within the activation map where the position corresponds to the center of the image patch on which the kernel is applied.

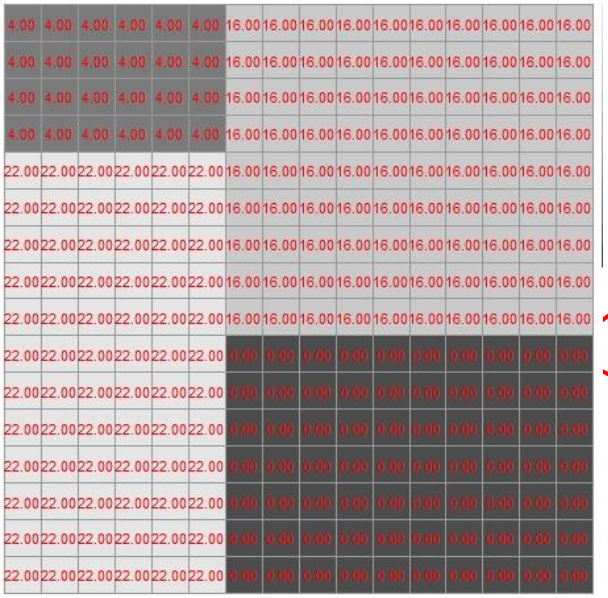
Taken from the great website on convolution: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

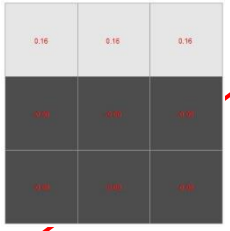
Convolutional networks use neighborhood information and replicated local feature extraction

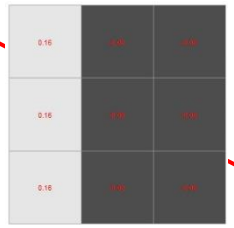
filtering = convolution

kernel 1

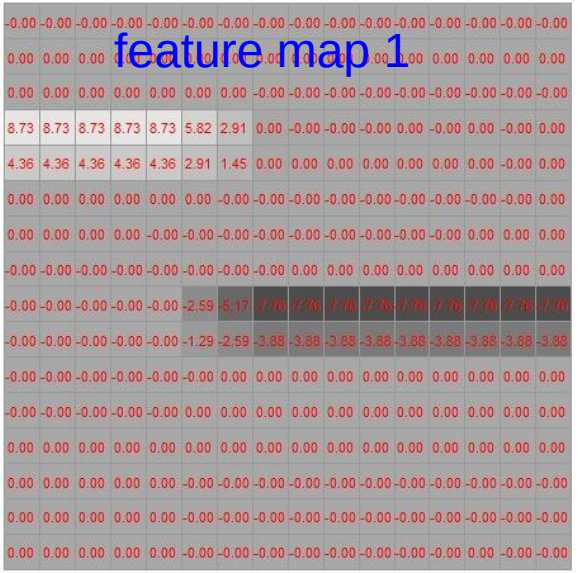
image







feature map 1



feature map 2

The weights of each filter are randomly initiated and then adapted during the training.

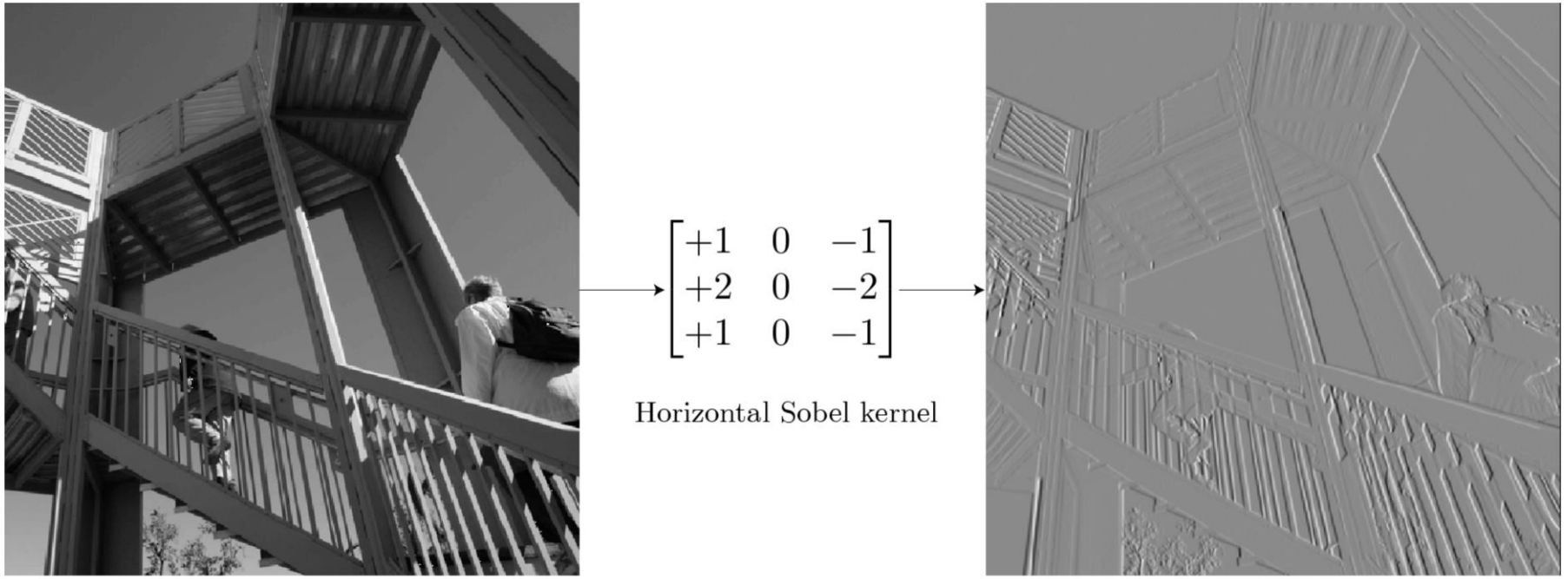
Exercise: Do one convolution step by hand

The kernel is 3x3 and is applied at each valid positon
– how large is the resulting activation map?

The small numbers in the shaded region are the kernel weights.
Determine the position and the value within the resulting activation map.

3	3	2	1	0
0_0	0_1	1_2	3	1
3_2	1_2	2_0	2	3
2_0	0_1	0_2	2	2
2	0	0	0	1

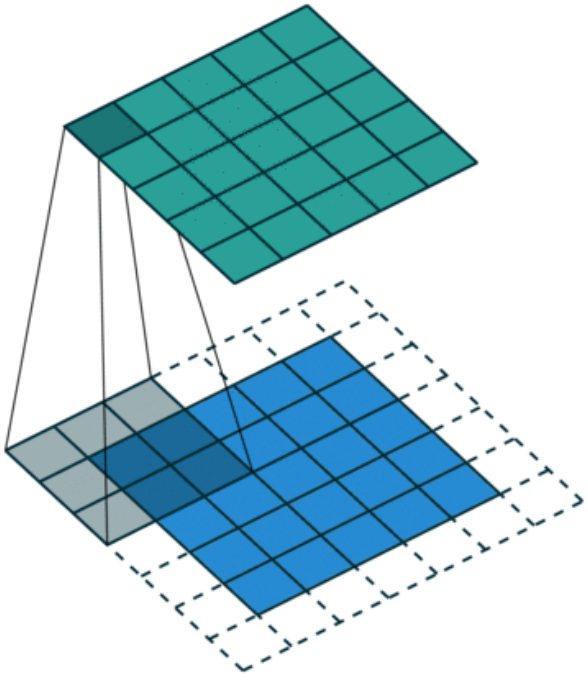
Example of designed Kernel / Filter



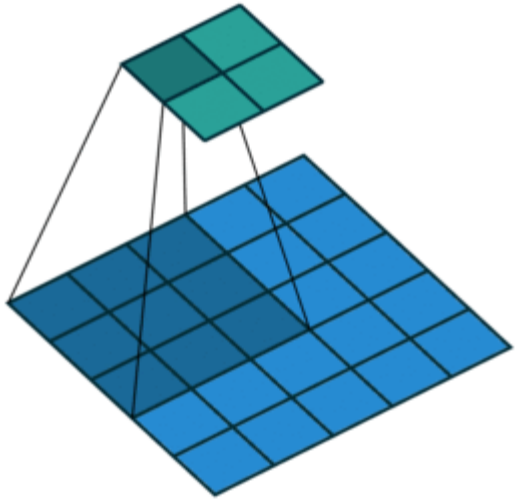
Applying a vertical edge detector kernel

Taken from the great website on convolution: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

CNN Ingredient I: Convolution



Zero-padding to achieve
same size of feature and input

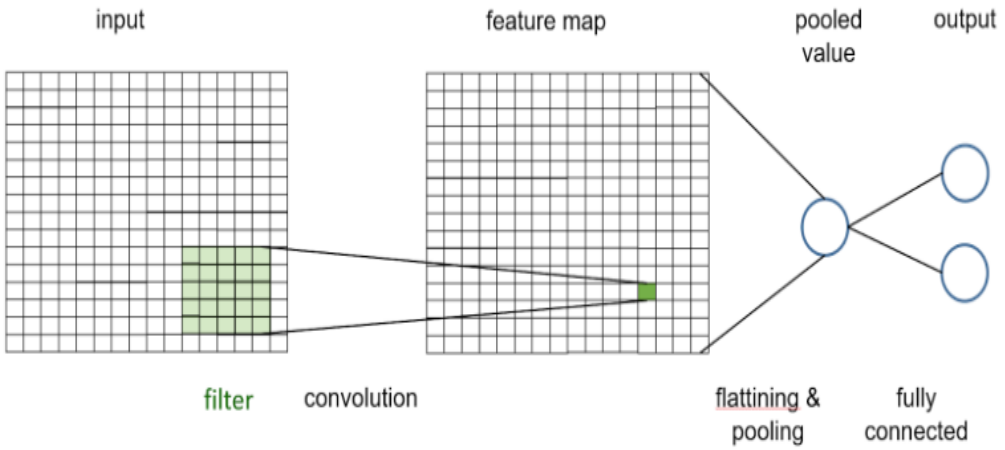


no padding to only use
valid input information

The *same* weights are used at each position of the input image.

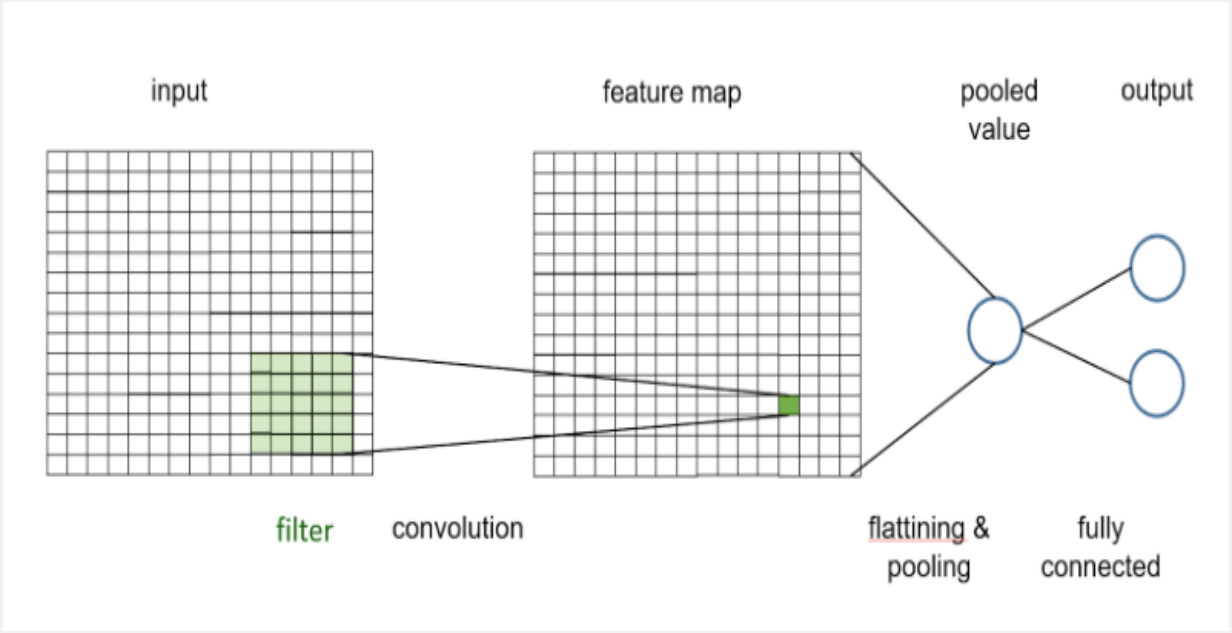
Taken from the great website on convolution: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

Building a very simple CNN with keras



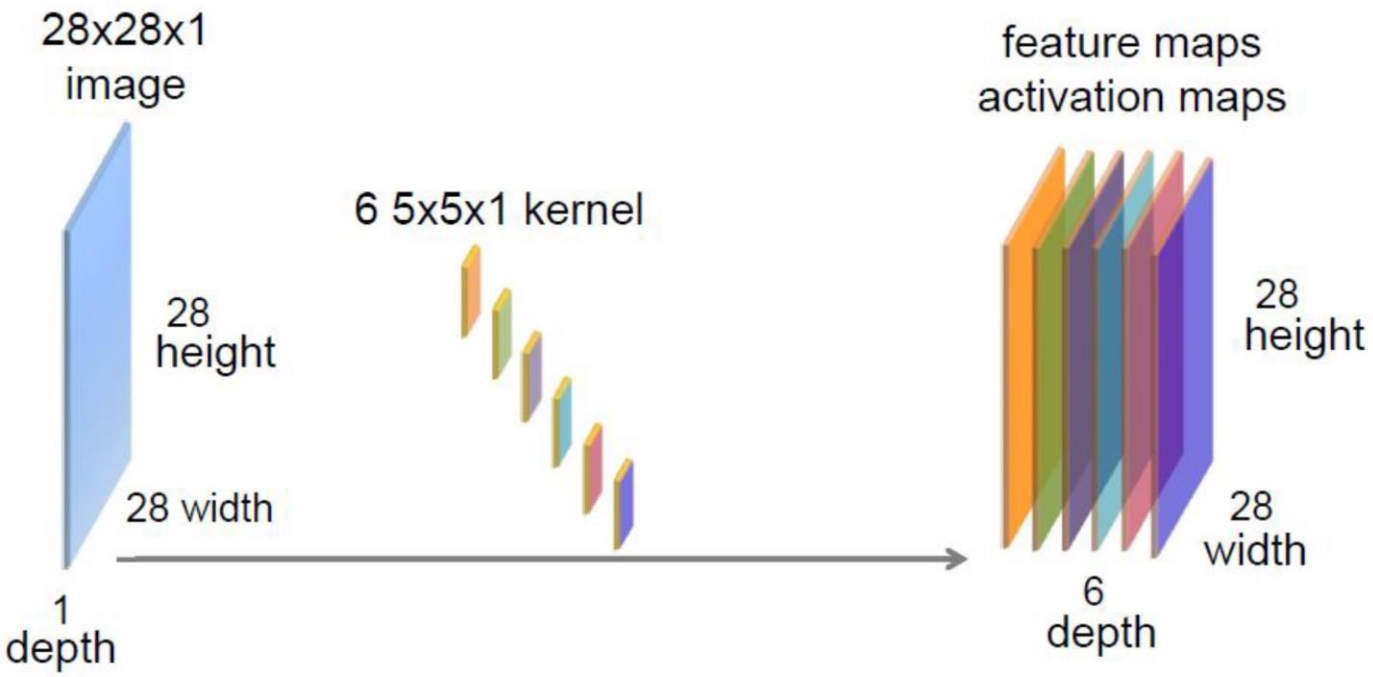

```
model = Sequential()
model.add(Convolution2D(1,(5,5), # one 5x5 kernel
        padding='same',          # zero-padding to preserve size
        input_shape=(pixel,pixel,1)))
model.add(Activation('linear'))
# take the max over all values in the activation map
model.add(MaxPooling2D(pool_size=(pixel,pixel)))
model.add(Flatten())
model.add(Dense(2))
model.add(Activation('softmax'))
```

Exercise: Artstyle Lover



Open NB in: https://github.com/tensorchiefs/dl_course_2022/blob/master/notebooks/05_cnn_edge_lover.ipynb

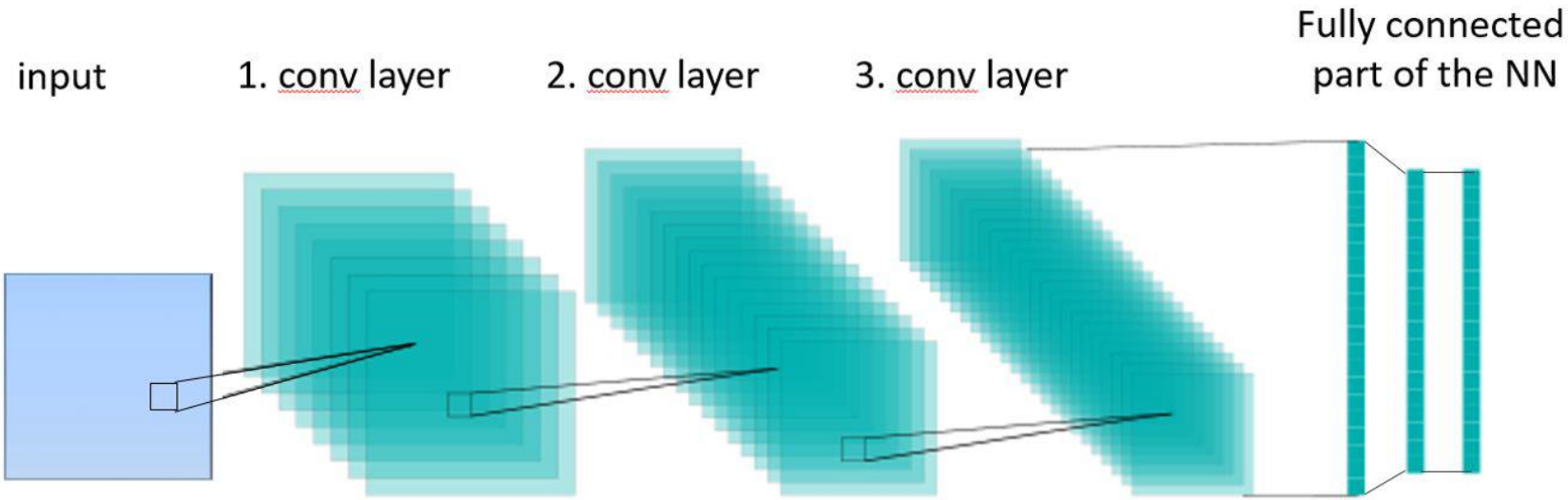
Convolution layer with a 1-chanel input and 6 kernels



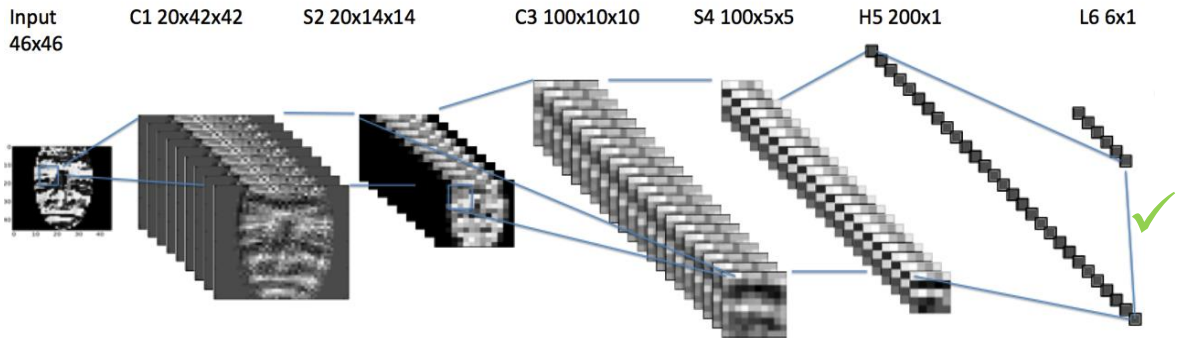
Convolution of the input image with 6 different kernels results in 6 activation maps.

If the input image has only one channel, then each kernel has also only one channel.

A CNN with 3 convolution layers



CNN ingredient II: Maxpooling Building Blocks reduce size



conv1 maxpool conv2 maxpool flatten dense

4x4

12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

→

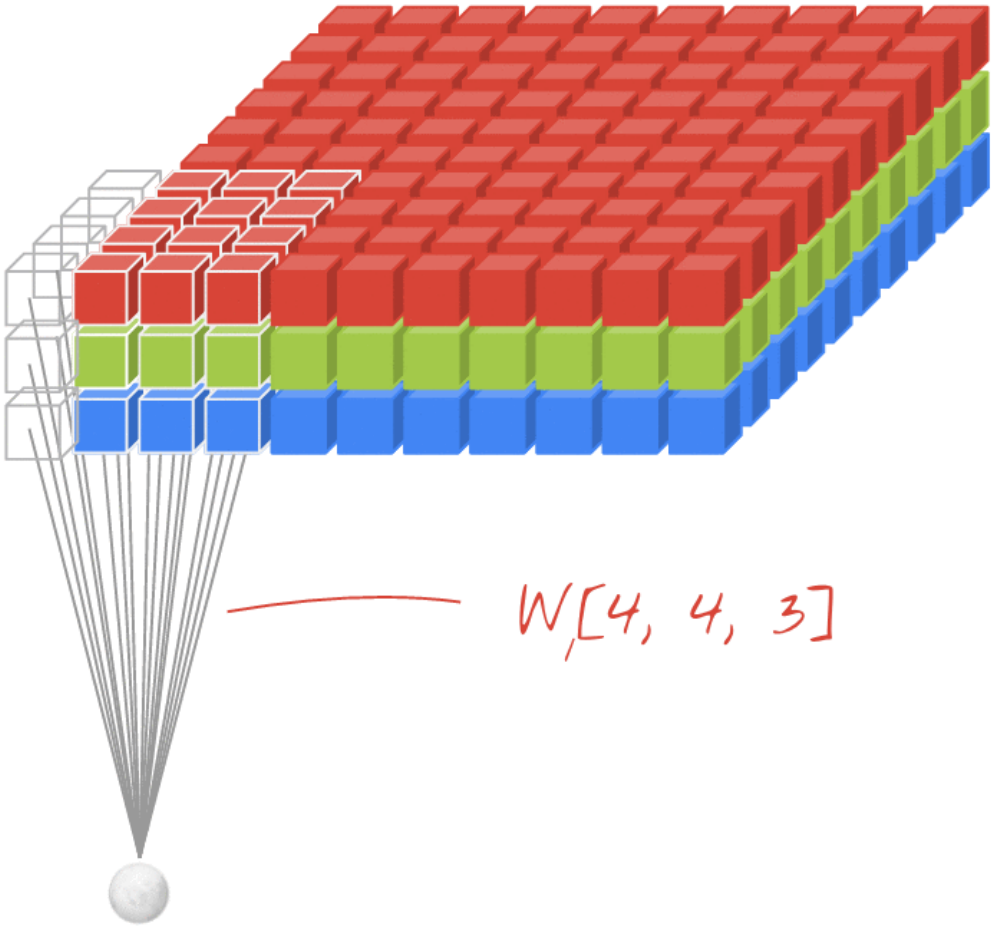
2x2

20	30
112	37

Simply join e.g. 2x2 adjacent pixels in one by taking the max.
→ less weights in model
→ Less train data needed
→ increased performance

Hinton: „The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster“

Animated convolution with 3 input channels



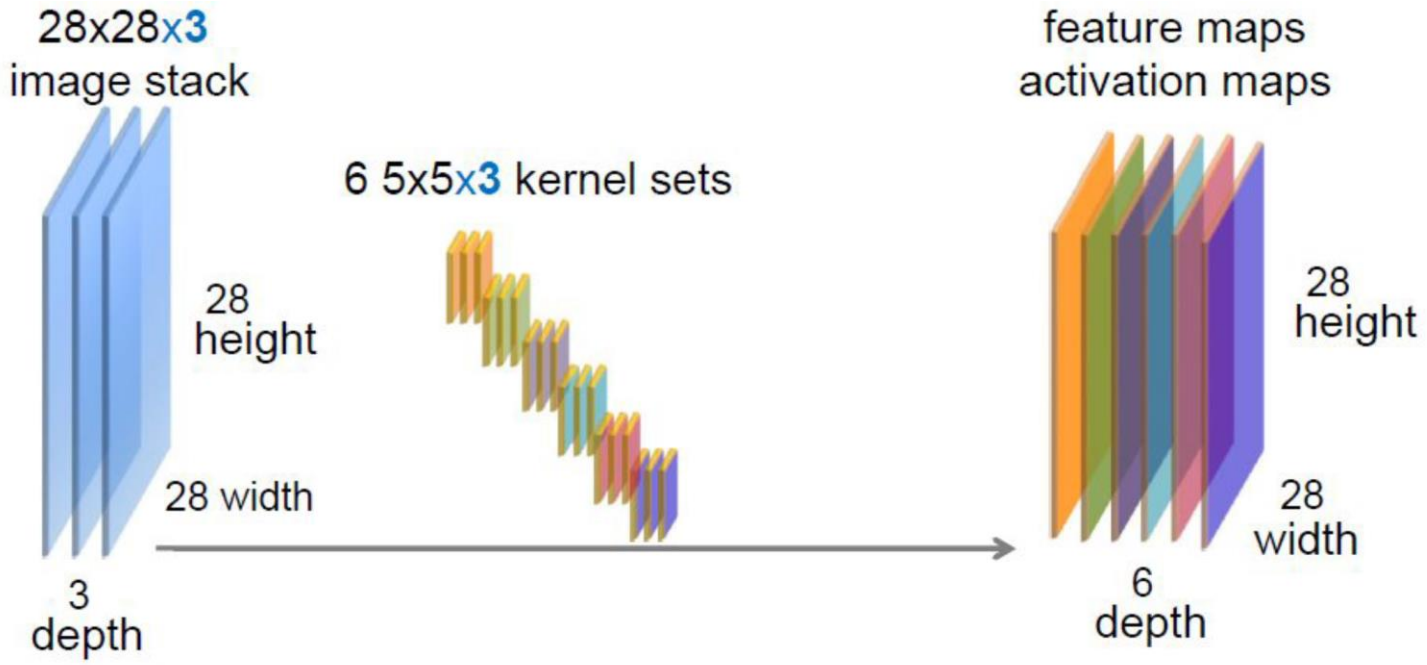
3 color channel input image

The value of neuron j in the k-th featuremap are computed from the weights in the k-th filter w_{ki} and the input values x_{ji} at the position j:

$$y_{jF_k} = f(z_{jF_k}) = f(b_k + \sum x_{ji} \cdot w_{ki})$$

Animation credits: M.Gorner, https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#10

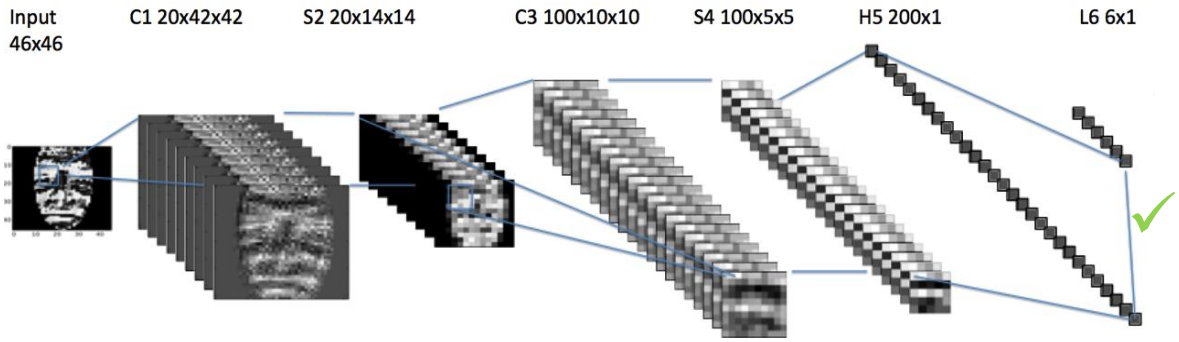
Convolution layer with a 3-chanel input and 6 kernels



Convolution of the input image with 6 different kernels results in 6 activation maps.

If the input image has 3 channels, then each filter has also 3 channels.

CNN ingredient II: Maxpooling Building Blocks reduce size



conv1 maxpool conv2 maxpool flatten dense

4x4

12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

→

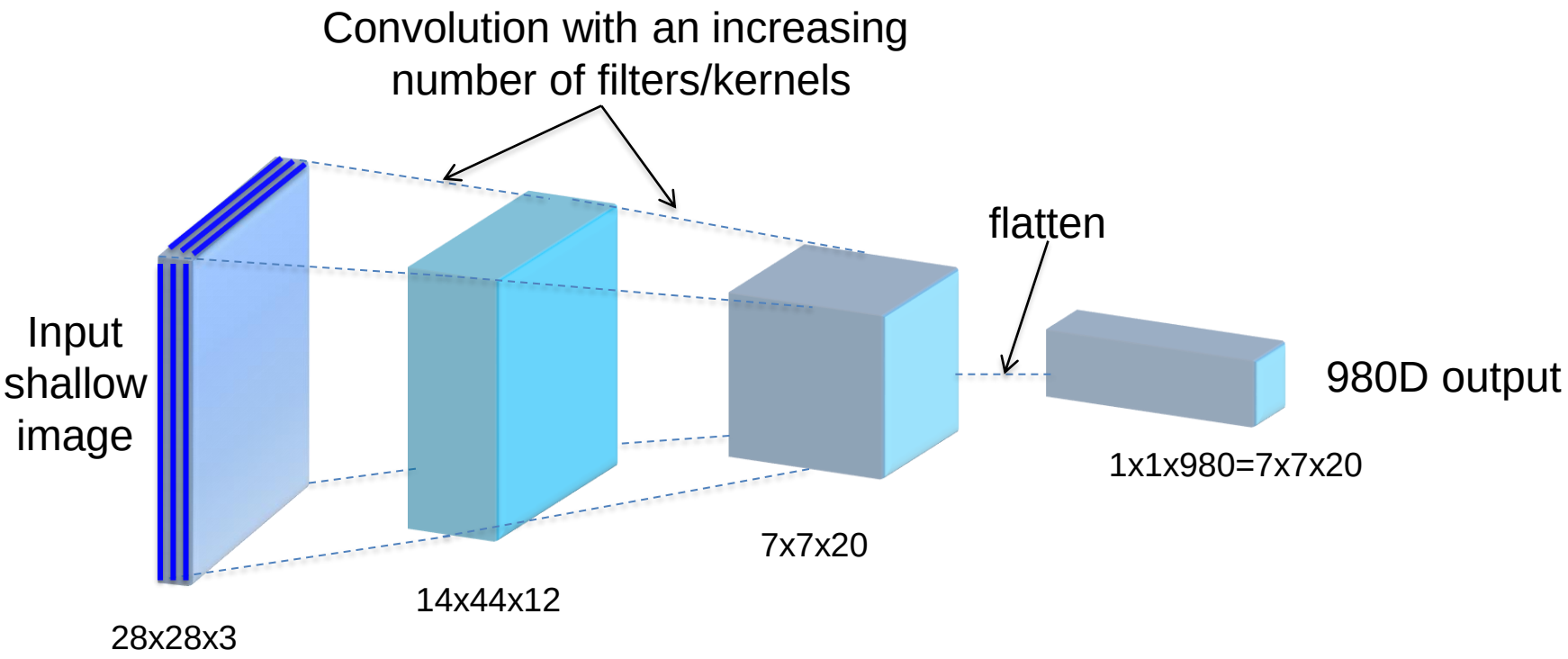
2x2

20	30
112	37

Simply join e.g. 2x2 adjacent pixels in one by taking the max.
→ less weights in model
→ Less train data needed
→ increased performance

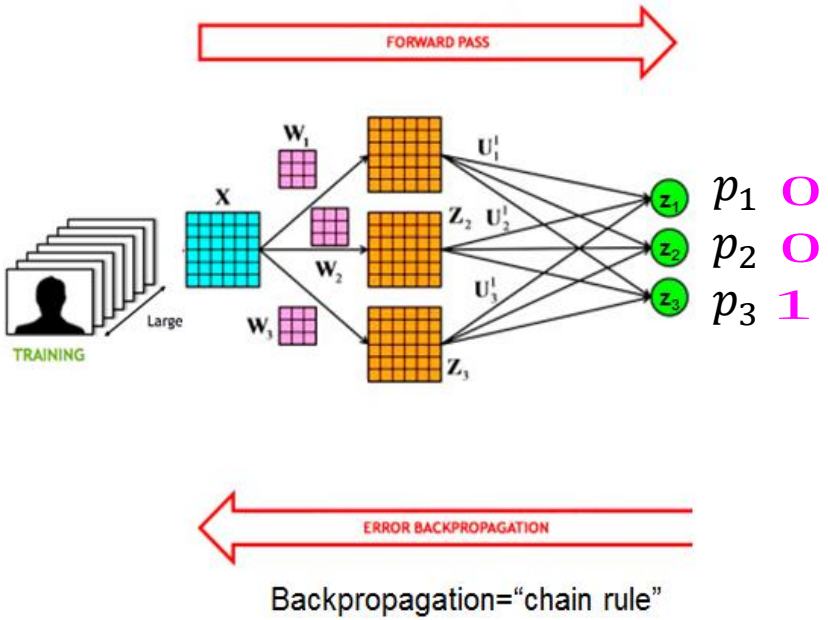
Hinton: „The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster“

Typical shape of a classical CNN



Spatial resolution is decreased e.g. via max-pooling while more abstract image features are detected in deeper layers.

Training of a CNN is based on gradient backpropagation



p_1 O
p_2 O
p_3 1

For the training we need the **observed label** for each image which we then compare with the **output** of the CNN.

We want to adjust the weights in a way so that difference between true label and output is minimal.

Minimize Loss-function:

L=distance(observed, output(w)**)**

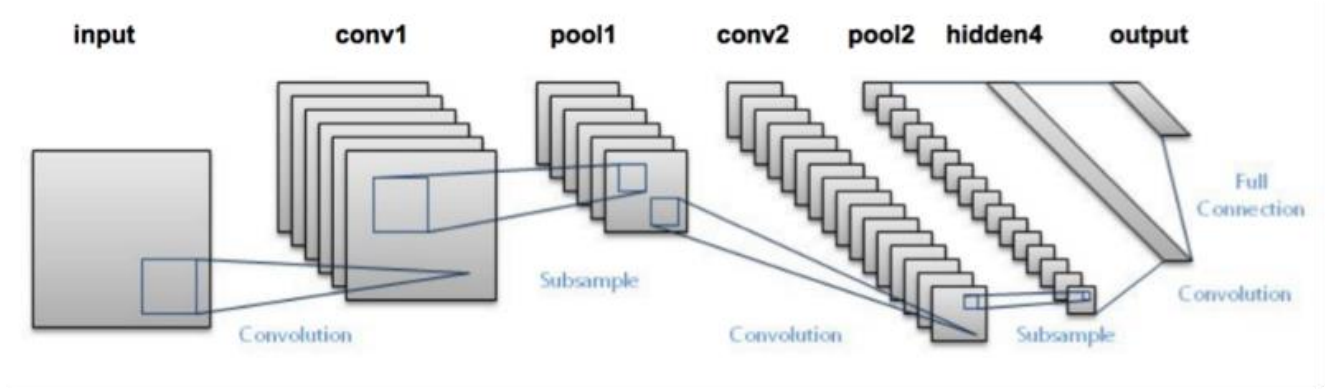
Learning is done by weight updating:

$$w_i^{(t)} = w_i^{(t-1)} - l^{(t)} \left. \frac{\partial L(w)}{\partial w_i} \right|_{w_i = w_i^{(t-1)}}$$

learning rate

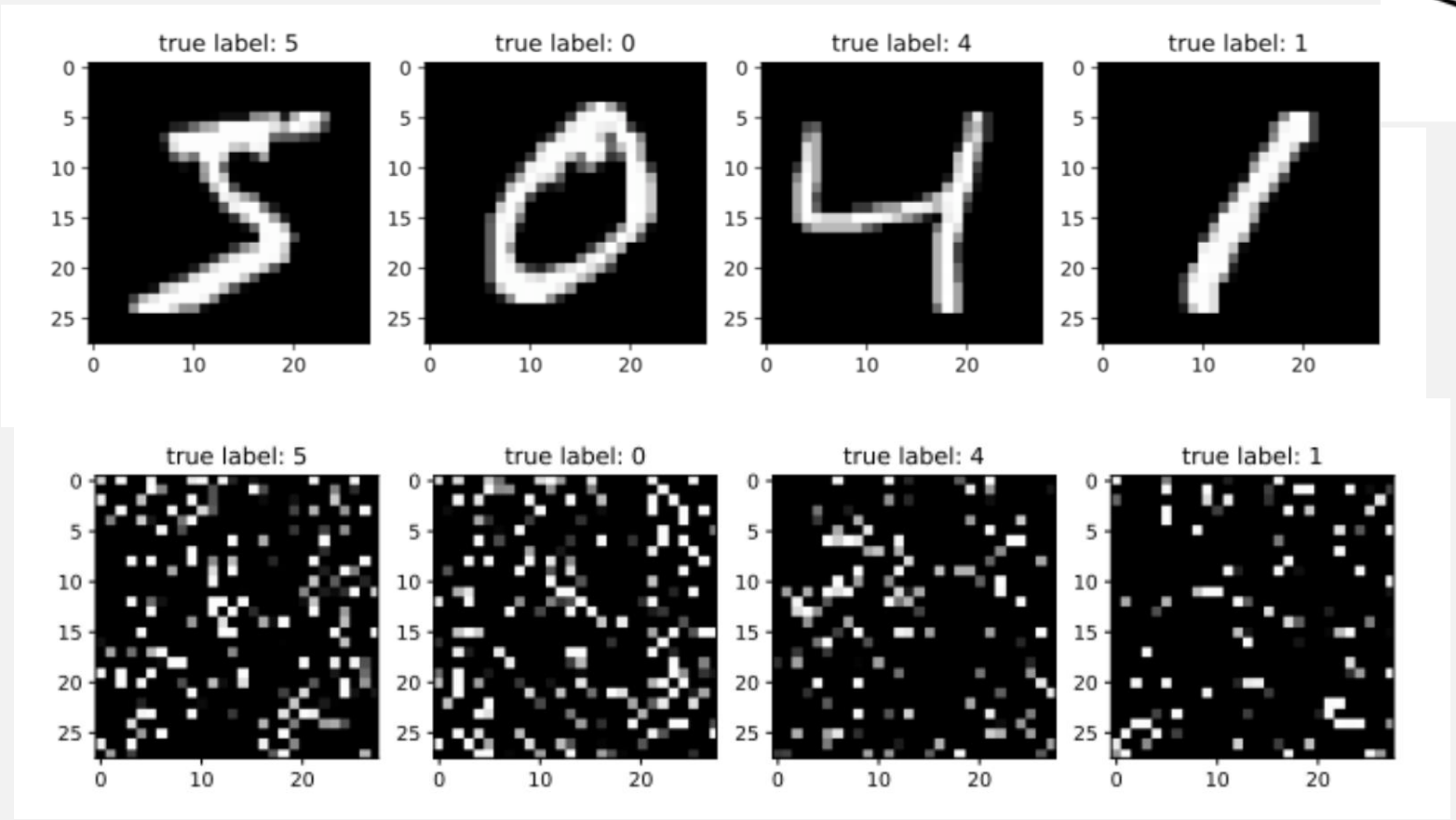
Image credits: Audhkhasi et al., Neural Networks 78 (2016) 15–23

CNN for MNIST




```
num_classes = 10              # 10 classes: 0, 1,…,9
input_shape = (28, 28, 1)   # 28x28 pixels, 1 channel (grey value)

model = Sequential()
model.add(Convolution2D(32, (3, 3), #32 filters of size 3x3
                  activation='relu',
                  input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Convolution2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(40, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

Exercise: Does shuffling disturb a CNN?

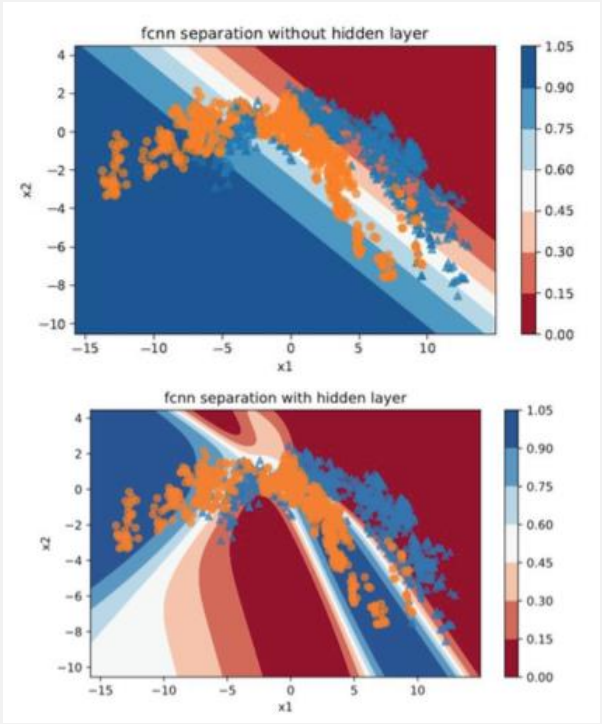


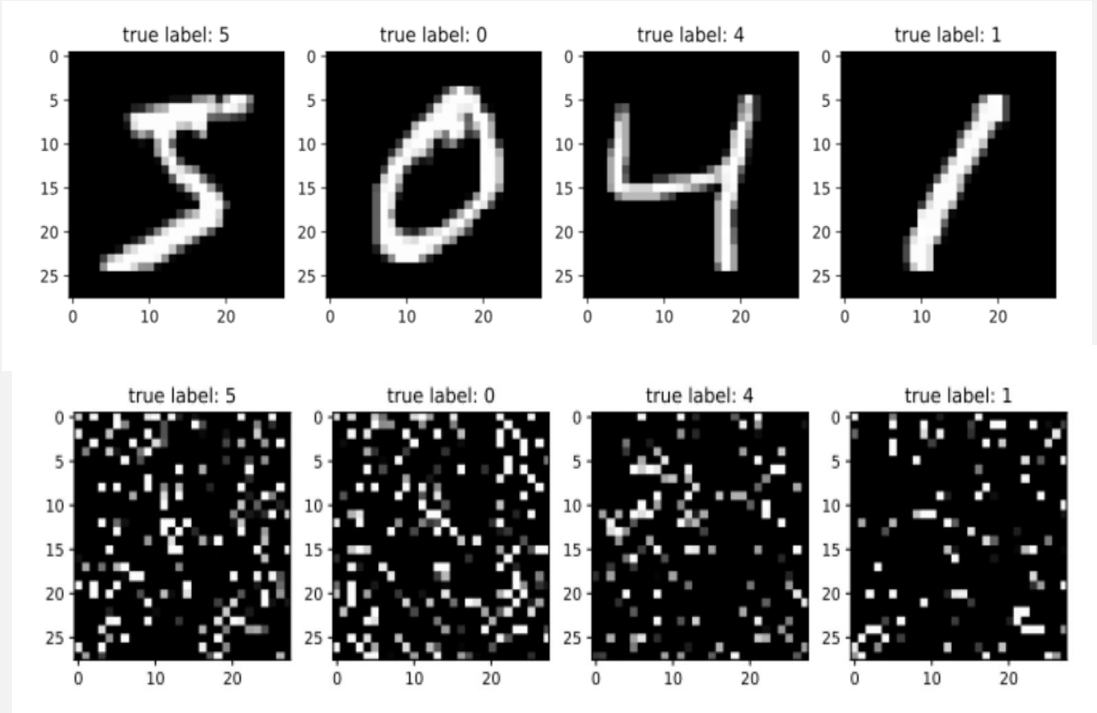
Open NB in: https://github.com/tensorchiefs/dl_course_2023/blob/master/notebooks/06_cnn_mnist_shuffled.ipynb

fcNN versus CNNs – some aspects

- A fcNN is good for tabular data, CNNs are good for ordered data (eg images).
- In a fcNN the order of the input does not matter, in CNN shuffling matters.
- A fcNN has no inductive model bias, while a CNN has the inductive model bias that neighborhood matters.
- A node in one layer of a fcNN corresponds to one feature map in a convolution layer:
- In each layer of a fcNN connecting p to q nodes, we learn q linear combinations of the incoming p signals, in each layer of a CNN connecting p channels with q channels we learn q filters (each having p channels) yielding q feature maps

Ufzgi: Finish banknote & shuffling experiment





Fit the a fcNN w/o and with hidden layer to discriminate fake bank notes from real bank notes: NB02

Inverstigate if shuffling disturbs the fcNN and/or CNN for MNIST: NB04 & NB06

Summary

- Use loss curves to detect overfitting or underfitting problems
- NNs work best when respecting the underlying structure of the data.
  - Use fully connected NN for tabular data
  - Use convolutional NN for data with local order such as images
- CNNs exploit the local structure of images by local connections and shared weight (same kernel is applied at each position of the image).
- Use the `relu` activation function for hidden layers in CNNs.