Machine Intelligence:: Deep Learning
Week 2

Beate Sick, Oliver Dürr

Institut für Datenanalyse und Prozessdesign

Zürcher Hochschule für Angewandte Wissenschaften

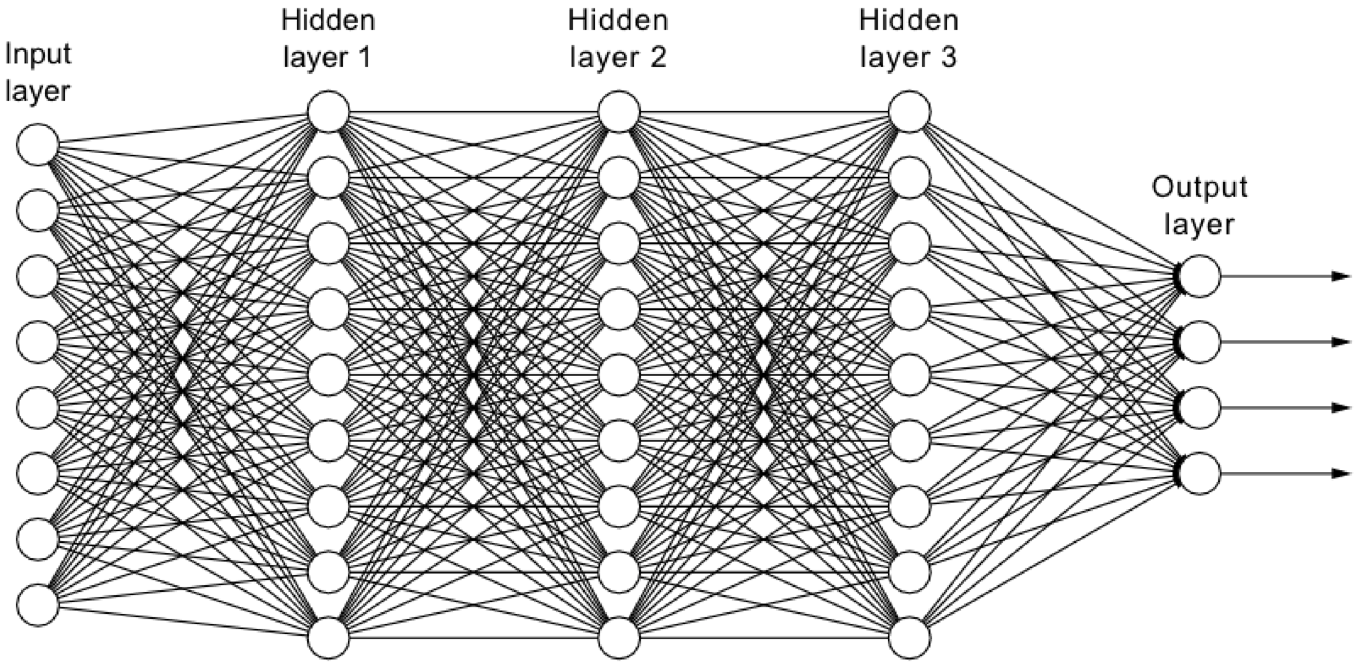
Topics of today

- A second look on fully connected Neural Networks (fcNN)

- Model development
  - What can we learn from loss curves
    - How to recognize overfitting and underfitting
    - How much data is needed?
    - Does my model learn the right things?

- Convolutional Neural Networks (CNN) for images
  - Motivation for switching from fcNN to CNNs
  - Introduction of convolution
  - ReLu and Maxpooling Layer
  - Biological inspiration of CNNs
  - Building CNNs

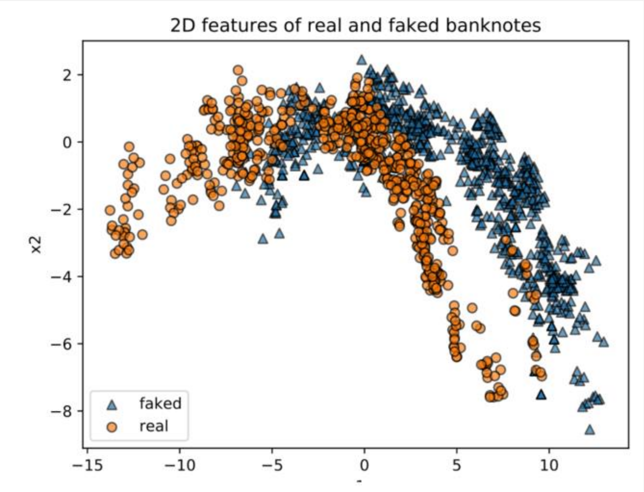
Architecture of a fully connected NN

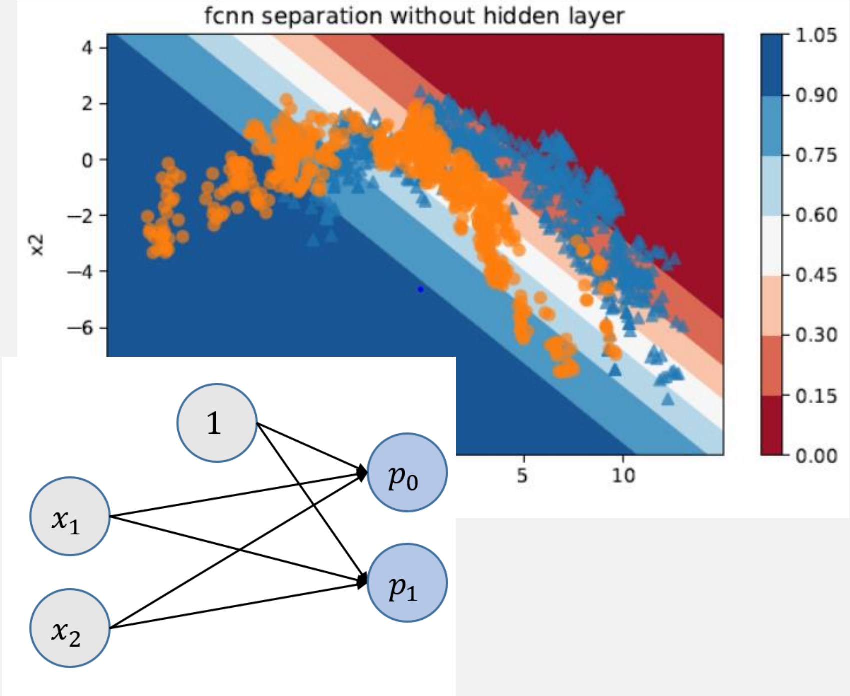


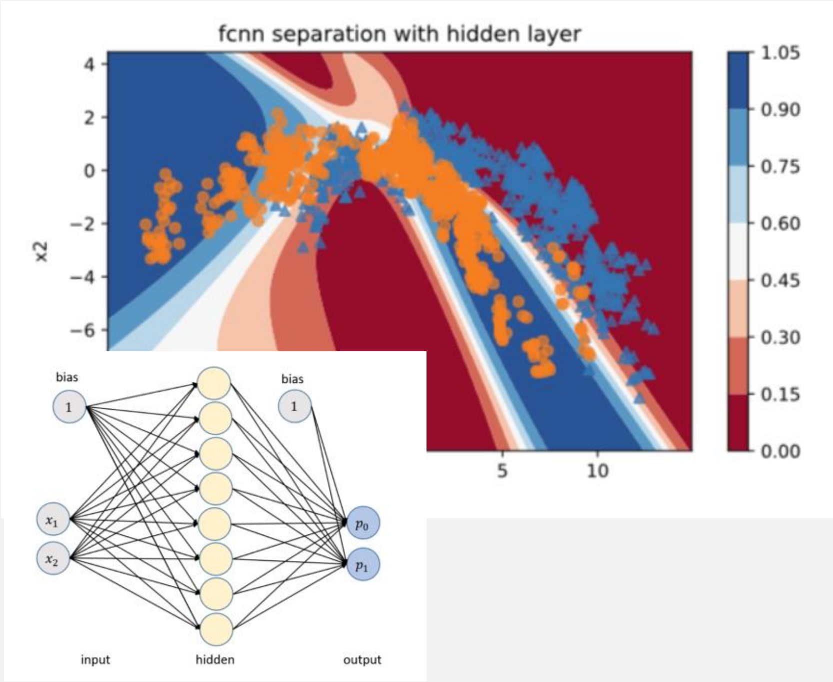
Each neuron in a fcNN gets as input a weighted sum of all neuron activation from one layer below. Different neurons in the same layer have different weights in this weighted sum, which are learned during training.

Homework

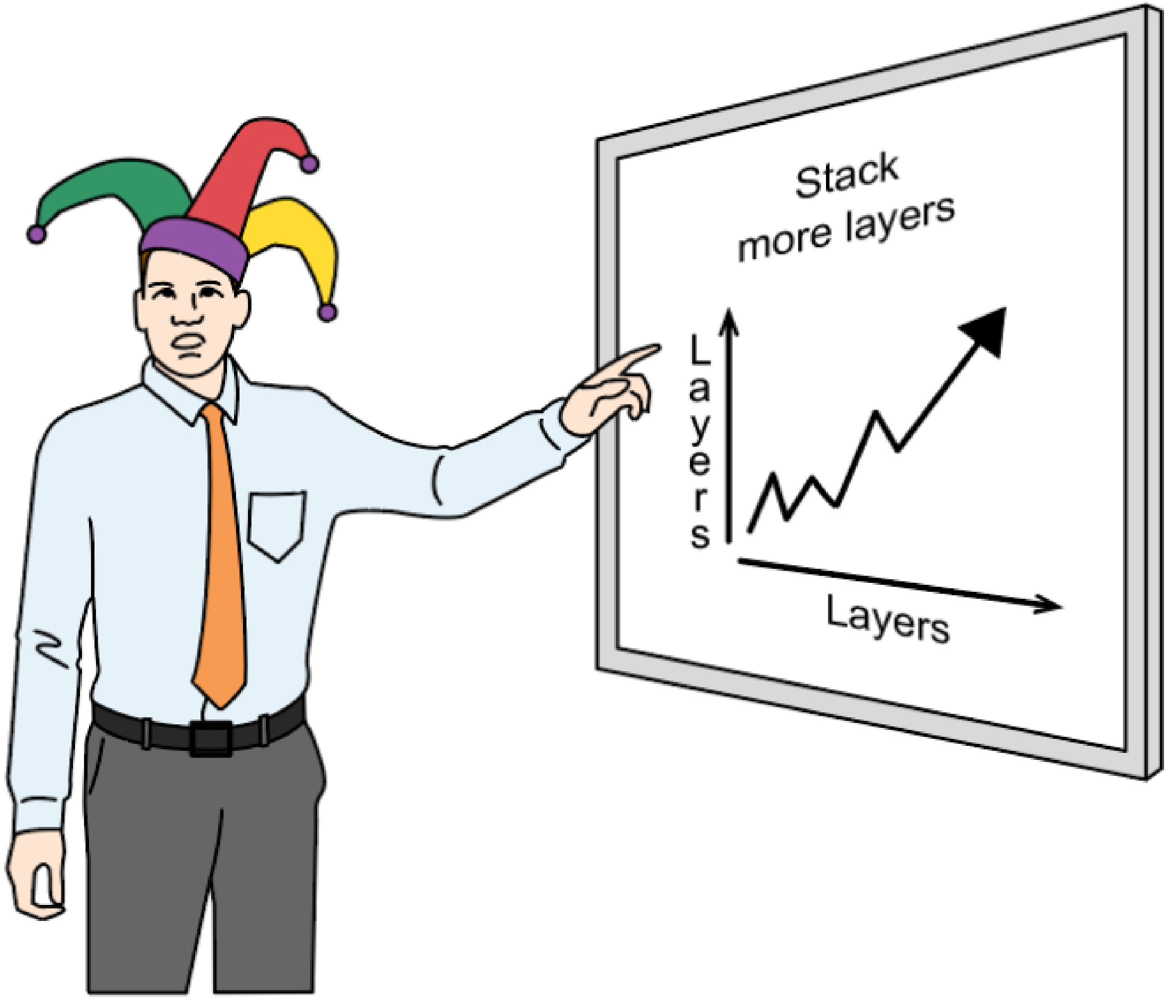
Do exercise NB 02: Classify
banknotesbased on 2 features (x1,x2)
https://github.com/tensorchiefs/dl_course_2021/blob/master/notebooks/02_fcnn_with_banknote.ipynb







A deep learning expert at work



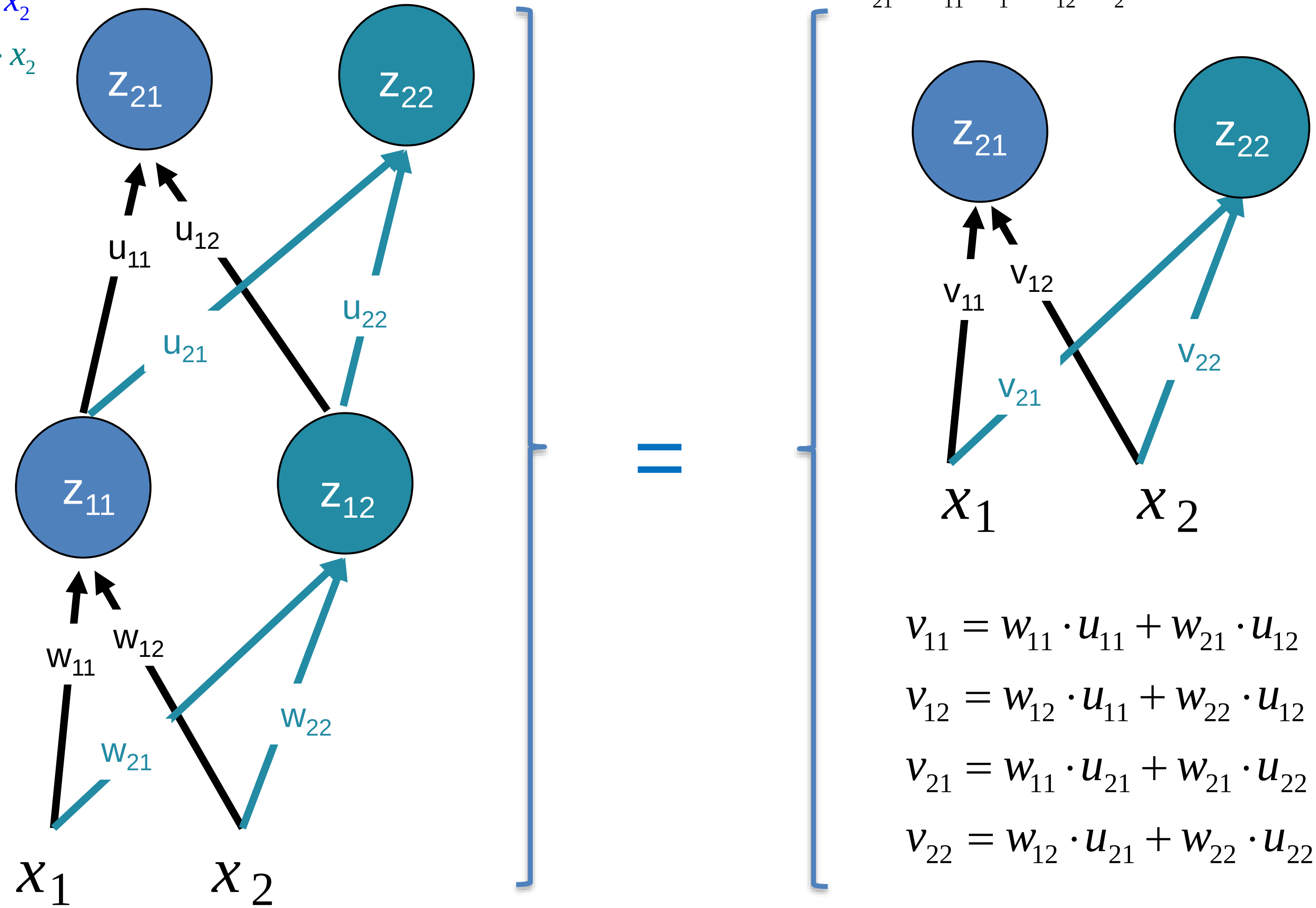
To go deep non-linear activation functions are needed

2 linear layers can be replaced by 1 linear layer -> can't go deep with linear layers!

$$z_{21} = z_{11} \cdot u_{11} + z_{12} \cdot u_{12} = (w_{11} \cdot x_1 + w_{12} \cdot x_2) \cdot u_{11} + (w_{21} \cdot x_1 + w_{22} \cdot x_2) \cdot u_{12}$$
$$= \mathrm{x}_1 \cdot (w_{11} \cdot u_{11} + w_{21} \cdot u_{12}) + \mathrm{x}_2 \cdot (w_{12} \cdot u_{11} + w_{22} \cdot u_{12})$$

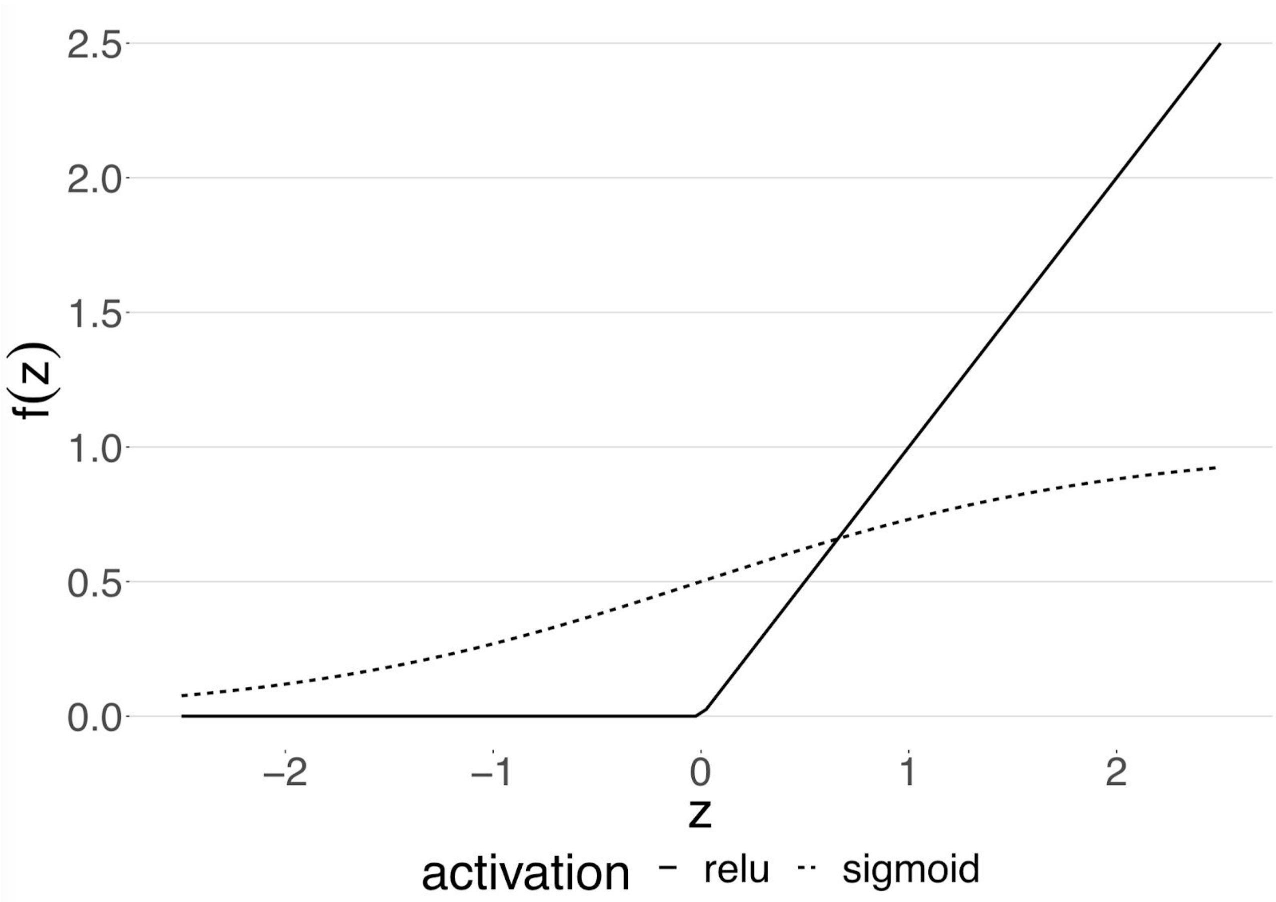
$$z_{11} = w_{11} \cdot x_1 + w_{12} \cdot x_2$$
$$z_{12} = w_{21} \cdot x_1 + w_{22} \cdot x_2$$

$$z_{21} = v_{11} \cdot x_1 + v_{12} \cdot x_2$$



$$v_{11} = w_{11} \cdot u_{11} + w_{21} \cdot u_{12}$$
$$v_{12} = w_{12} \cdot u_{11} + w_{22} \cdot u_{12}$$
$$v_{21} = w_{11} \cdot u_{21} + w_{21} \cdot u_{22}$$
$$v_{22} = w_{12} \cdot u_{21} + w_{22} \cdot u_{22}$$

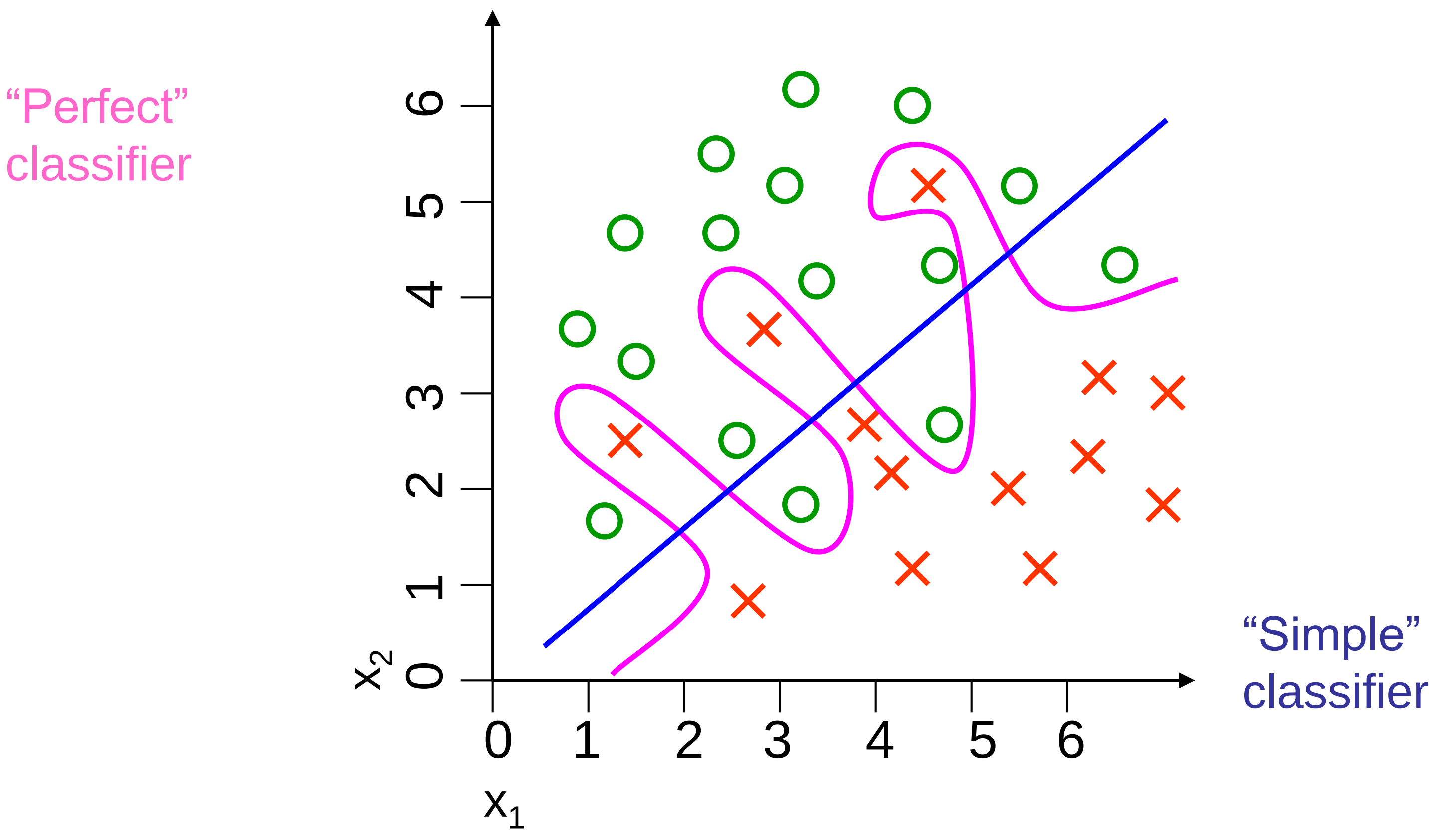
Comon non-linear activation function



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

Model development
Overfitting and underfitting

“Perfect” Vs. “Simple” classifier



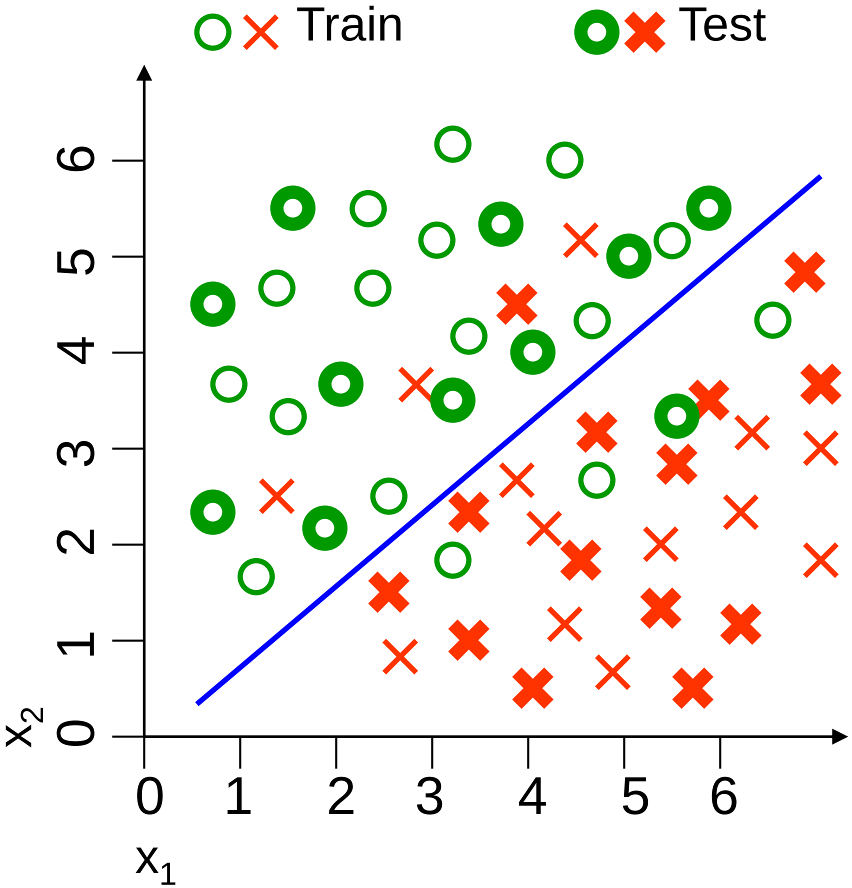
Which is better?

Check on a test-set (cross validation).

Cross validation of the "simple" classifier

Training set:
6/29=20%
misclassification

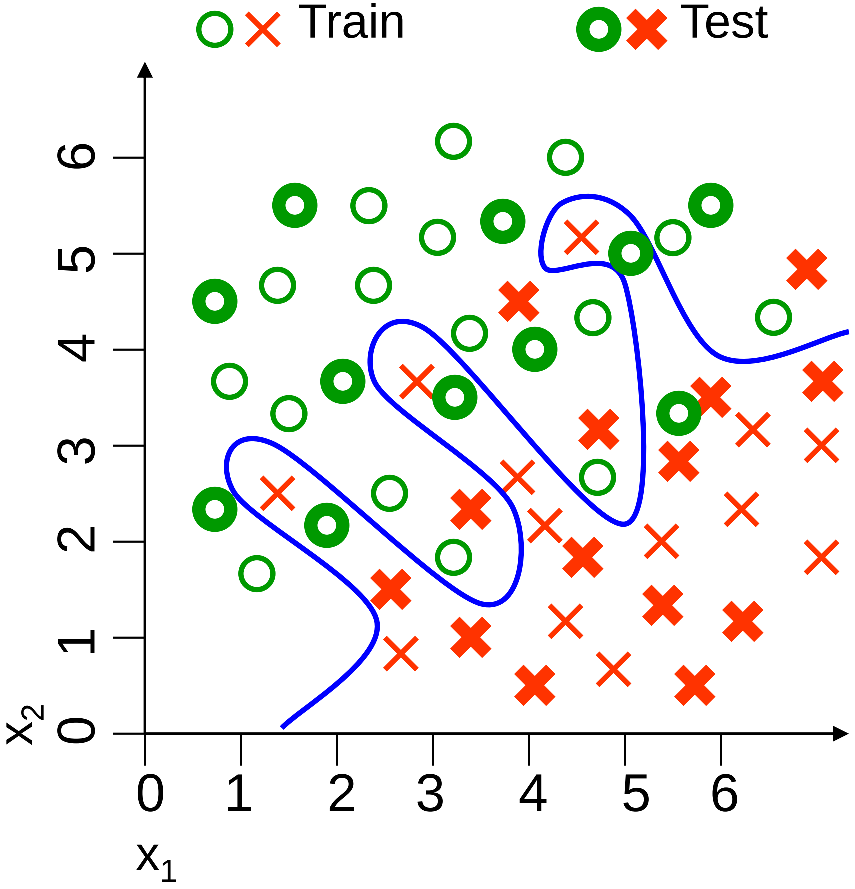
Test set:
2/25=8%
misclassification



Cross validation of the "Perfect" classifier

Training set:
0%misclassification

Test set:
8/25=24%
misclassification



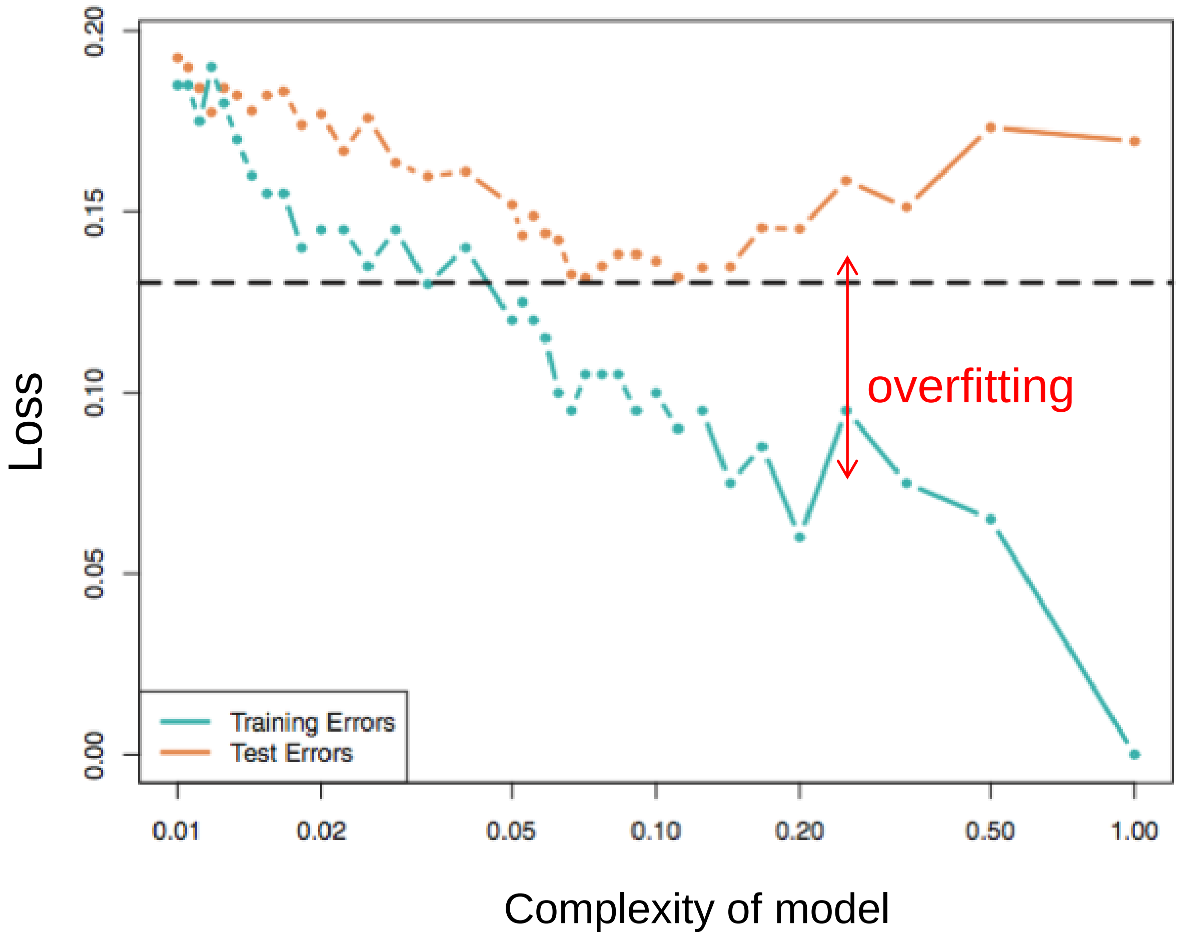
Types of Errors

- **Training error** **or** **naïve error** **or** **in-sample-error:**
  - Error on data that have been used to train the model
- **Generalization error** **or** **test error** **or** **out-of-sample-error:**
  - Error on previously unseen records (out of sample)

Over-fitting phenomenon:

- Model fits the training data well (small training error) but shows high generalization error

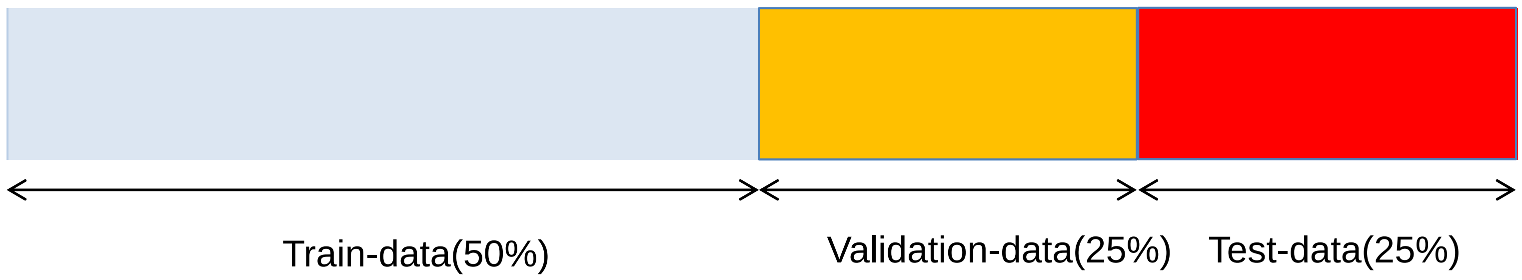
What is the right level of complexity



Remark: In DL the models are often very flexible and show overfitting when trained over many epochs – early stopping or regularization are needed.

Image credits: "Elements of statistical learning"

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

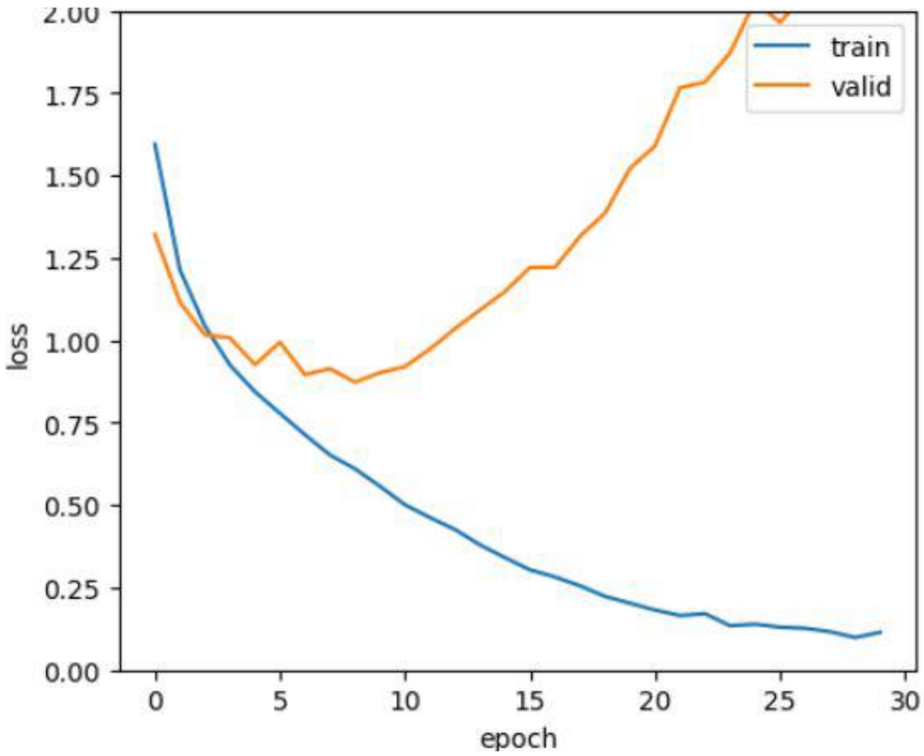


Best practice: Lock an extra test data set away, and use it only at the very end, to evaluate the chosen model, that performed best on your validation set.
Reason: **When trying many models, you probably overfit on the validation set**.

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

What can loss curves tell us?

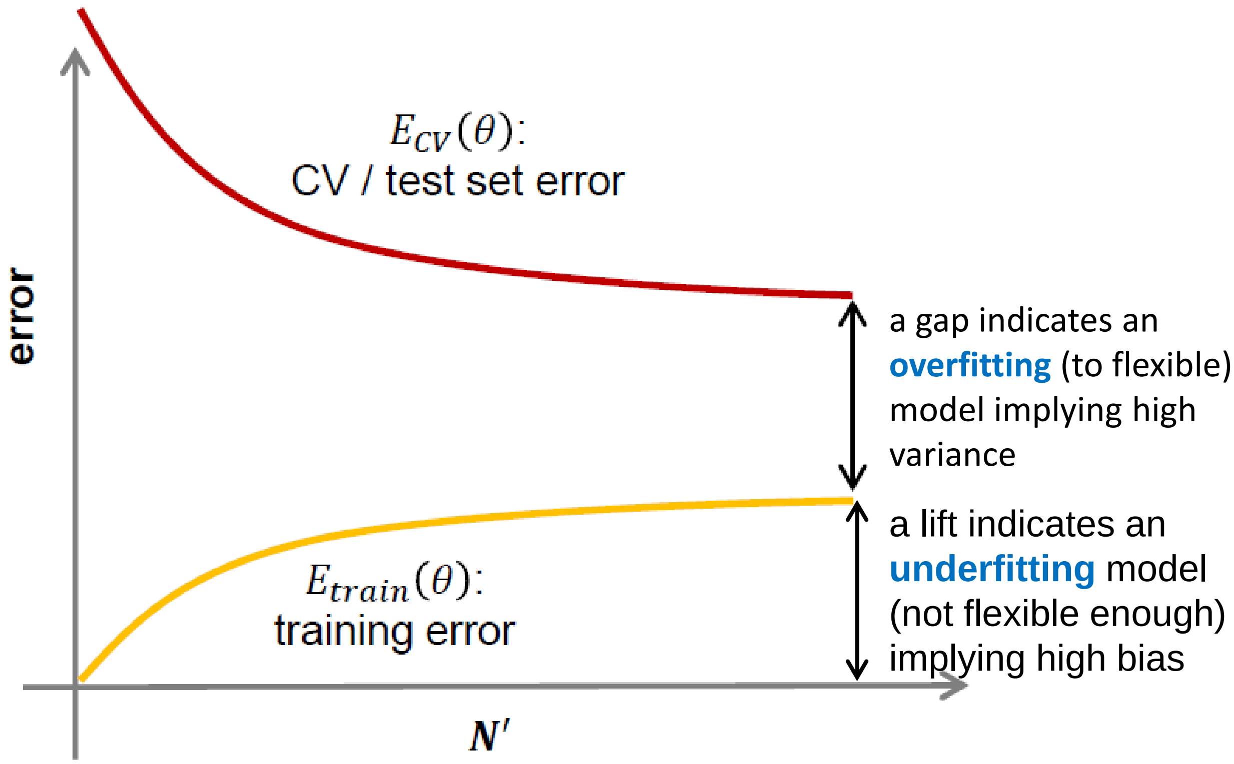
Very common check: Plot loss in train and validation data vs epoch of training.



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

What can loss curves tell us?

Less common check: Plot loss in train and validation data vs amount of train data.



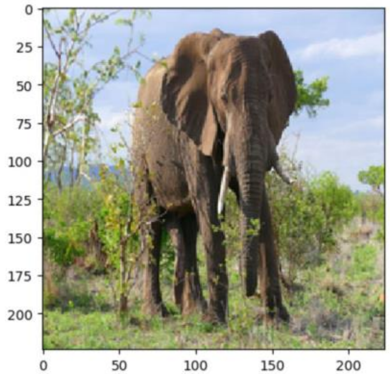
possible cures for overfitting:
- more data (if curves are still approaching)
- more representative data (if gap is constant)
- less complex model (regularization)
- bagging (and boosting)

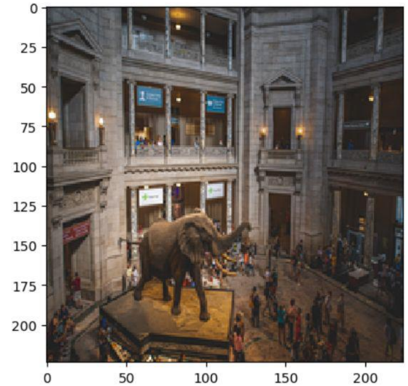
possible cures for underfitting:
- Use more complex model (less regularization)
- Use other model structure
- boosting

Model training process relies on "the big lie"

P(train) = P(test) = P(test): The "big lie"

- In traditional machine learning and DL we usually assume that the train and test data do come from the same distribution.
- If this assumption is not true, a seemingly high performant model breaks down and cannot see the elephant in the room.
- We need models that can flag uncertain predictions or even better learn relevant (causal) features that do not depend on distribution.

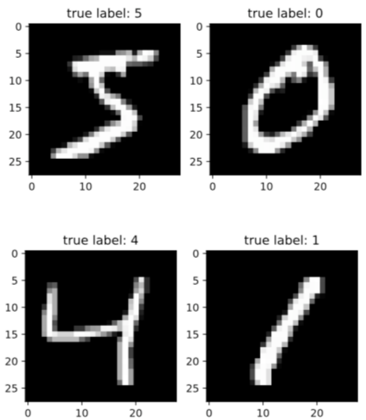




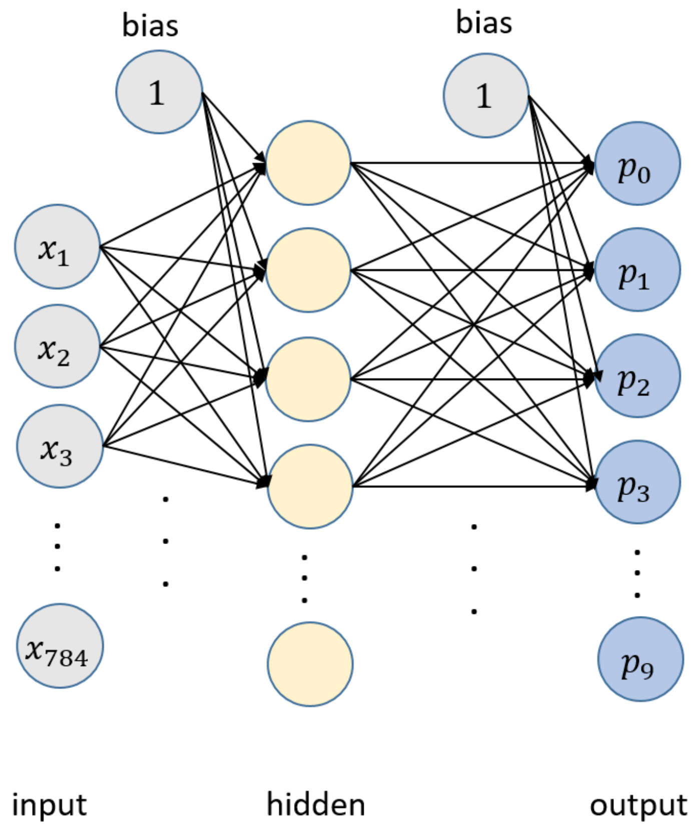
Remark: we will discuss this in more detail in later lectures and also learn about some cures.

Fully connected NN for image data
Why not?

A fcNN for MNIST data



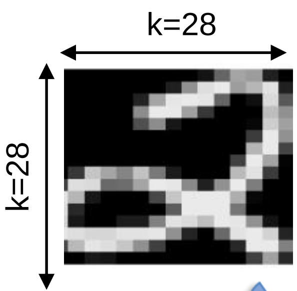
The first four digits of the MNIST data set - each image consisting of 28x28=784 pixels



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

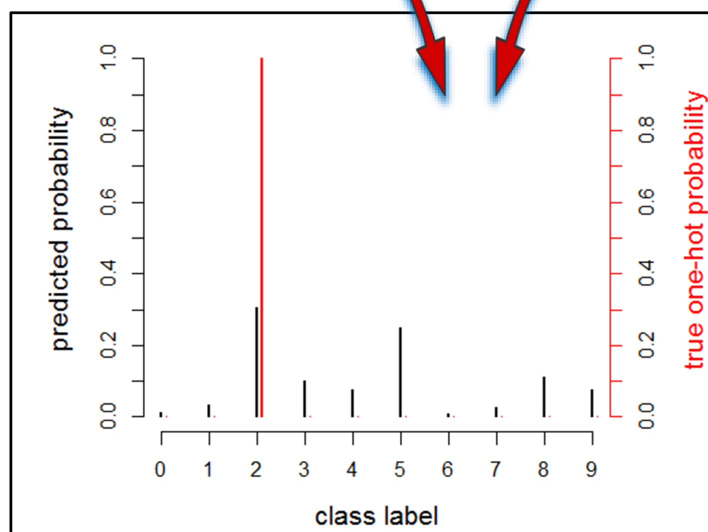
$$\hat{\mathrm{p}}_k = \frac{e^{z_k}}{\sum_j e^{zj}}$$

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



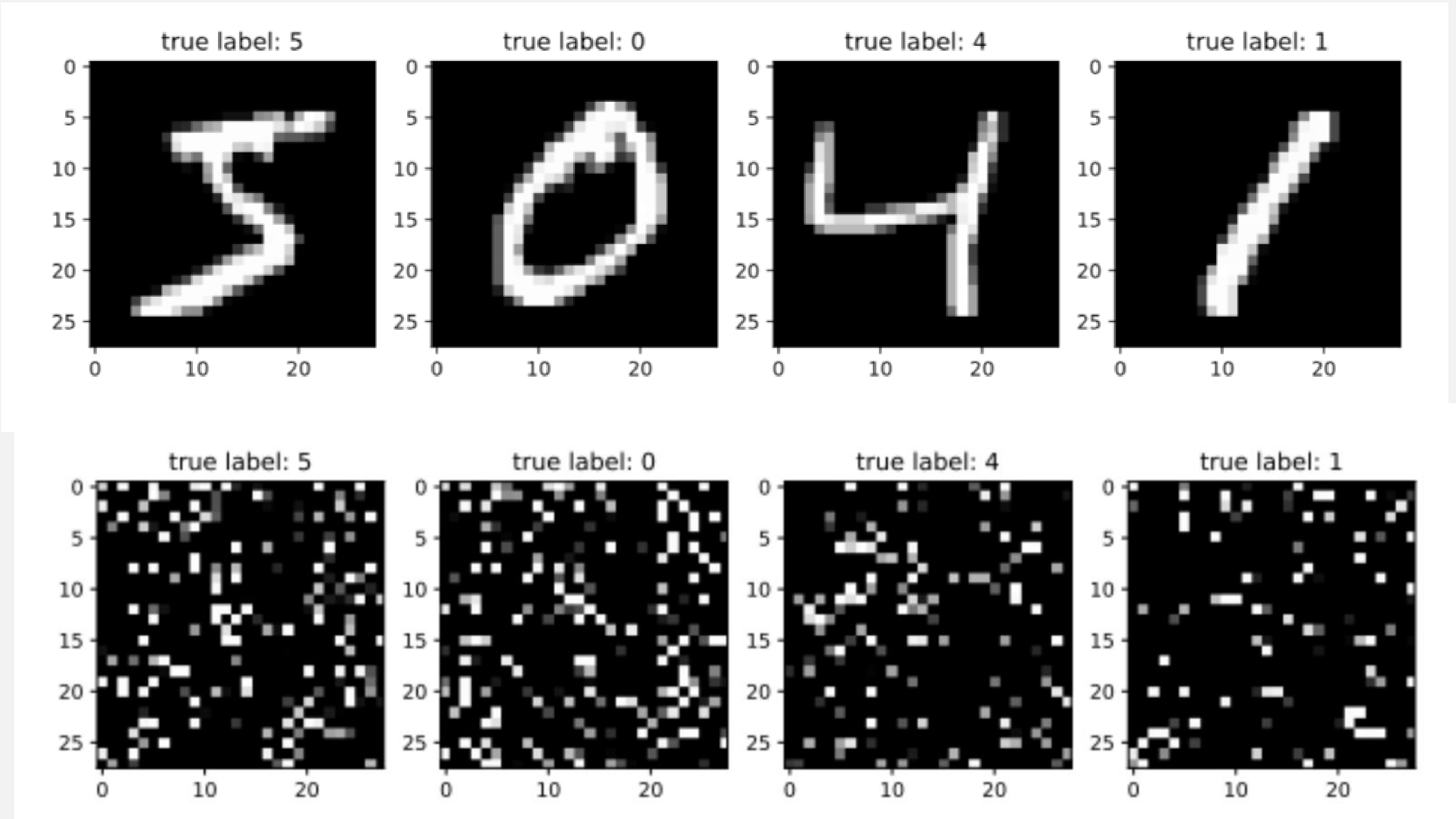
Cost C or Loss = cross-entropy averaged over all images in mini-batch

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

Cross-Entropy

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

Exercise: Does shuffling disturb a fcNN?



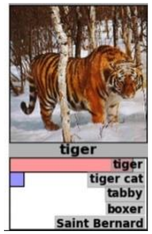
Use fcNN for MNIST:
https://github.com/tensorchiefs/dl_course_2020/blob/master/notebooks/03_fcnn_mnist.ipynb

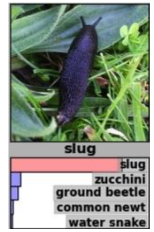
Inverstigate if shuffling disturbs the fcNN for MNIST:
https://github.com/tensorchiefs/dl_course_2020/blob/master/notebooks/04_fcnn_mnist_shuffled.ipynb

Convolutional Neural Networks SoA for image data

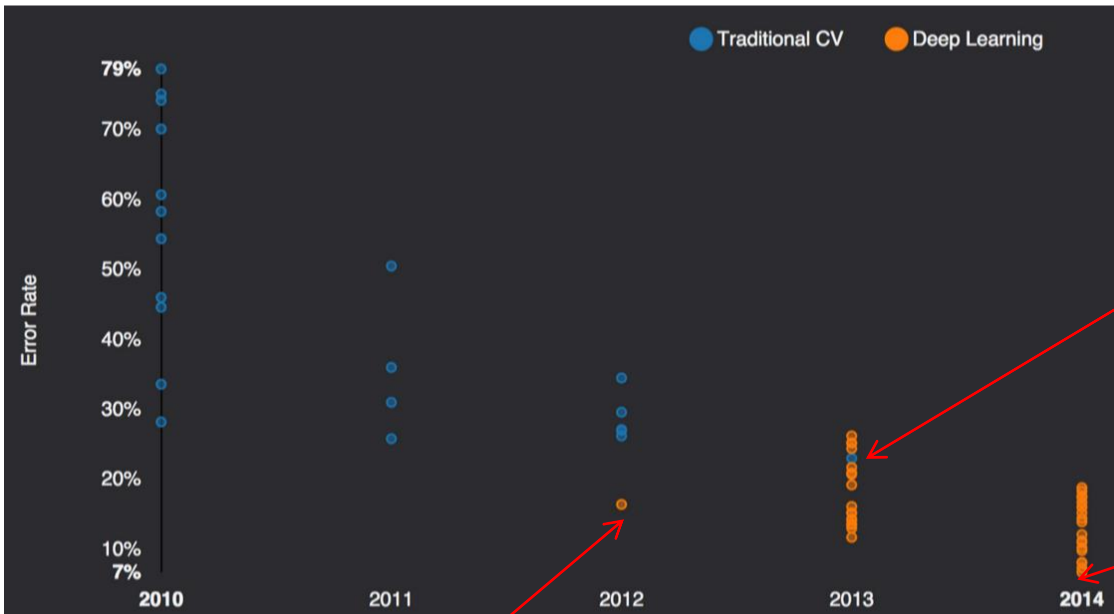
Recall: Imagenet challenge

1000 classes
1 Mio samples





…



Human: 5% misclassification

Only one non-CNN approach in 2013

GoogLeNet 6.7%

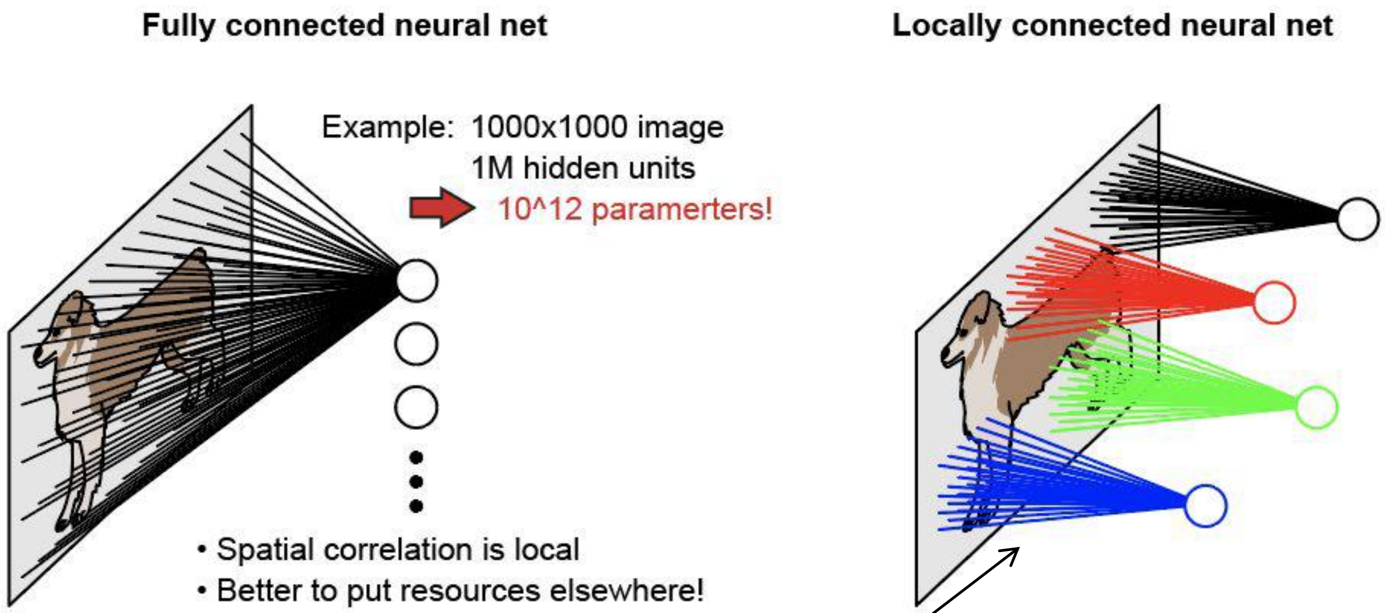
A. Krizhevsky
first CNN in 2012
Und es hat zoom gemacht

2015: It gets tougher

- 4.95% Microsoft (Feb 6 surpassing human performance 5.1%)
- 4.8% Google (Feb 11) -> further improved to 3.6 (Dec)?
- 4.58% Baidu (May 11 banned due too many submissions)
- 3.57% Microsoft (Resnet winner 2015)

Figure: https://medium.com/global-silicon-valley/machine-learning-yesterday-today-tomorrow-3d3023c7b519

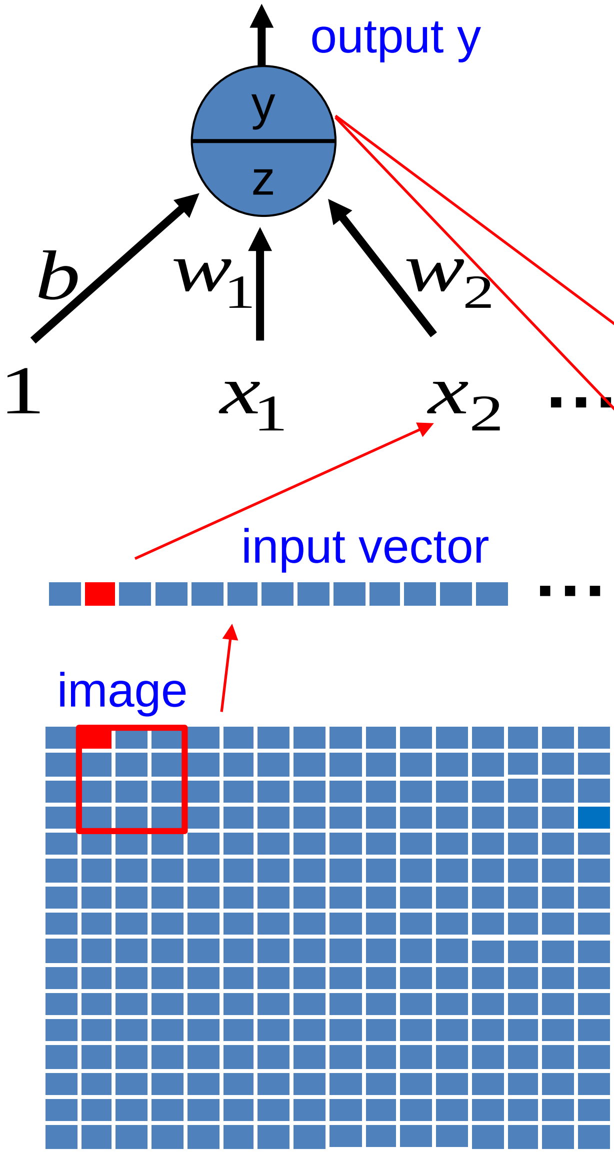
Convolution extracts local information using few weights



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

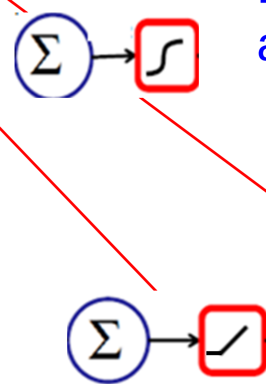


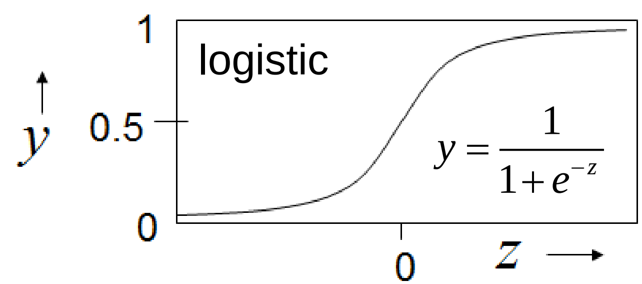
bias

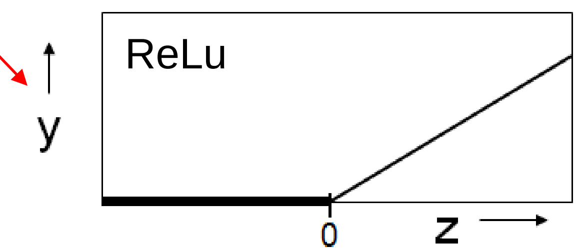
weights

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

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





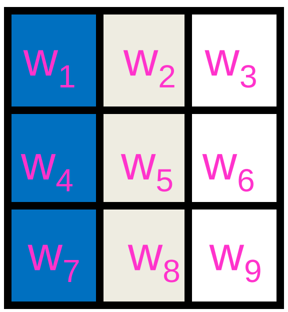


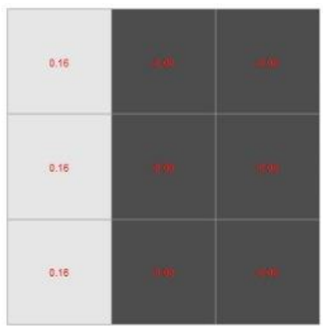
Convolutional networks use neighborhood information and replicated local feature extraction

In a locally connected network the calculation rule

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

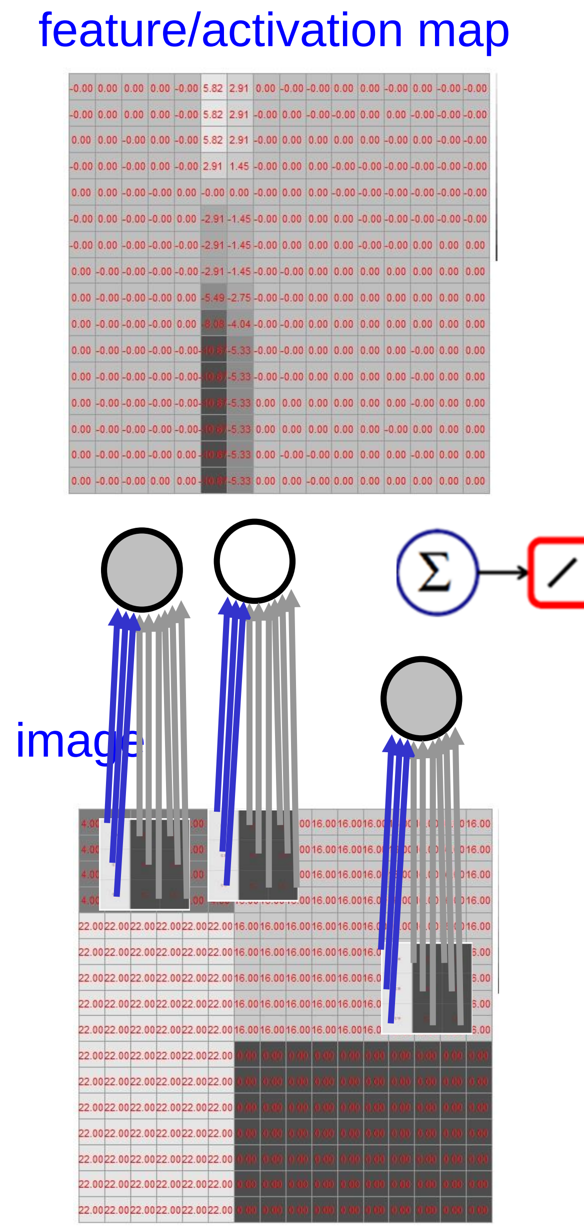
Pixel values in a small image patch are element-wise multilied with weights of a small filter/kernel:



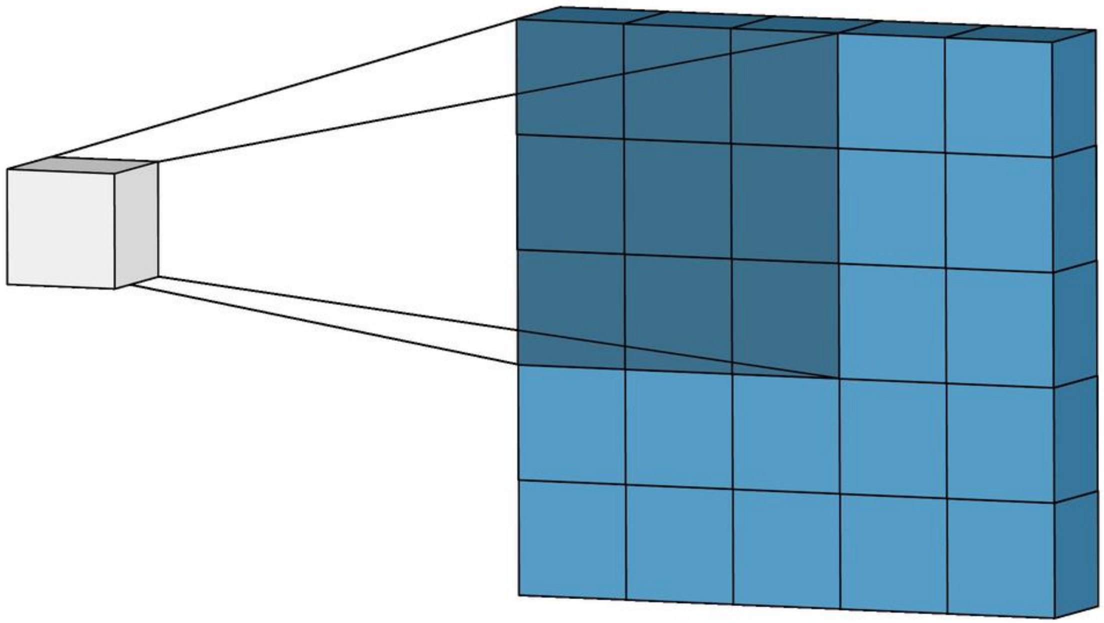


The filter is applied at each position of the image and it can be shown that the result is maximal if the image pattern corresponds to the weight pattern.

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



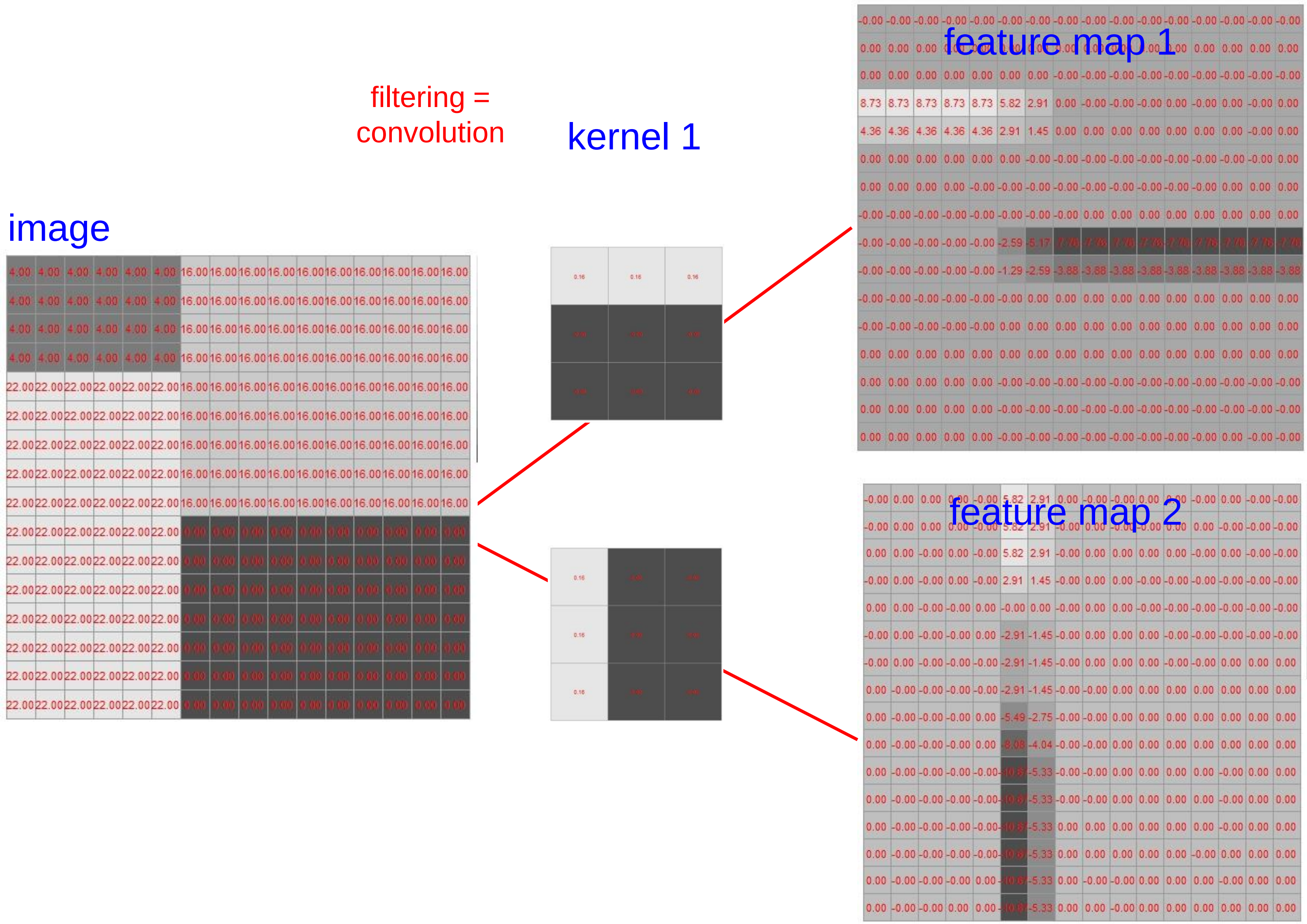
Applying the same 3x3 kernel at each image position



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

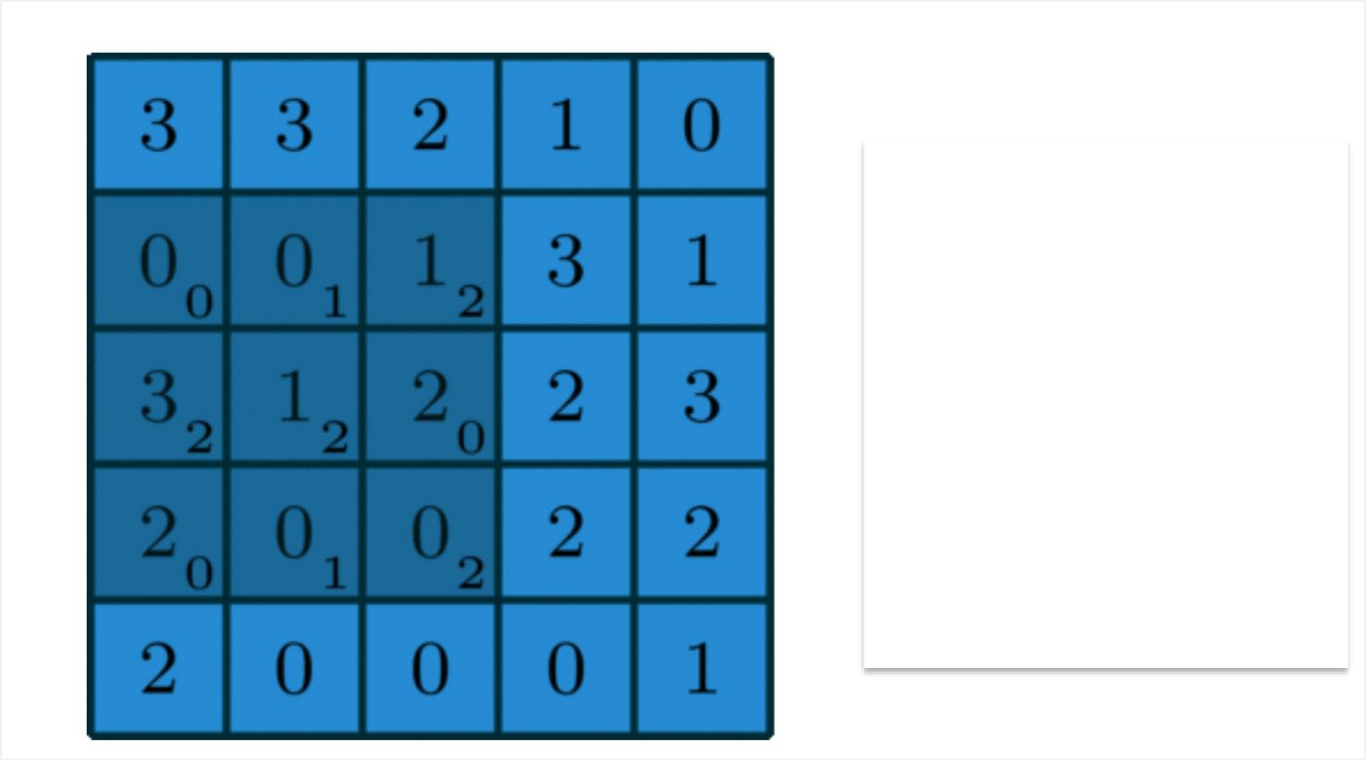


The weights of each filter are randomly initiated and then adapted during the training.

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?

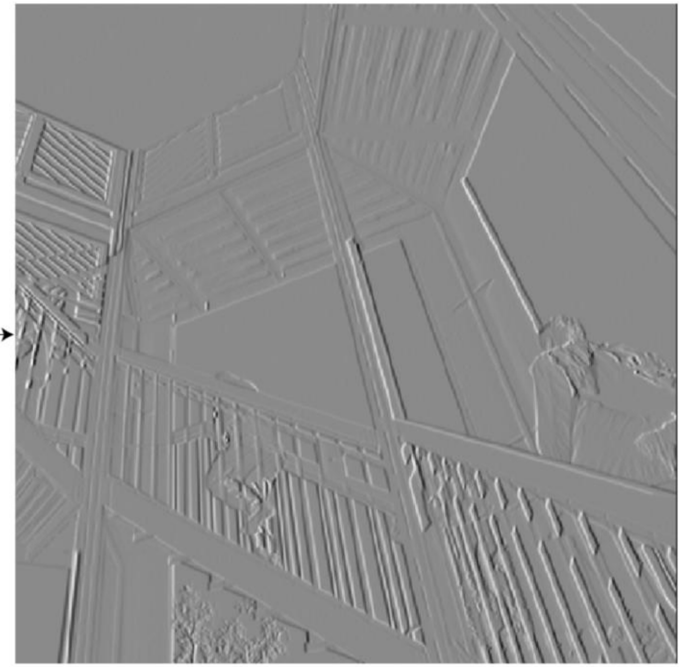
The small numbers in the shaded region are the kernel weights.
Determine the position and the value within the resulting activation map.



Example of designed Kernel / Filter

$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

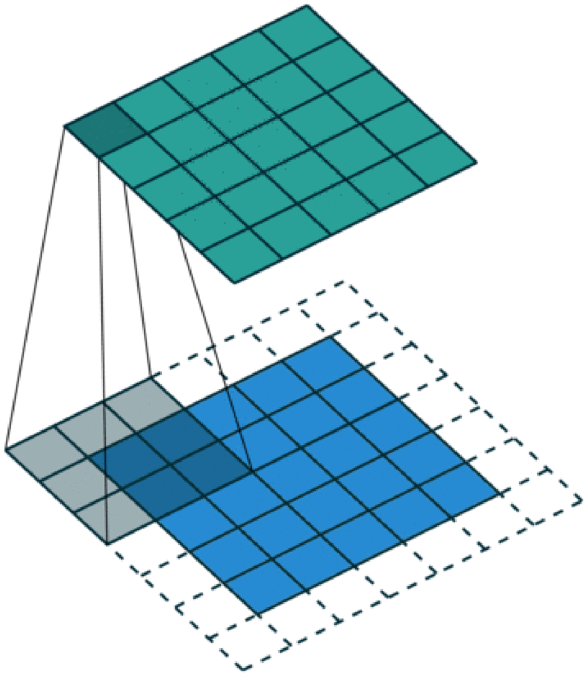
Horizontal Sobel kernel



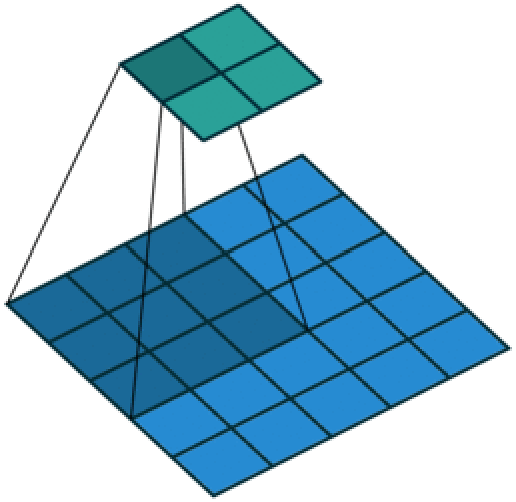
Applying a vertical edge detector kernel

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

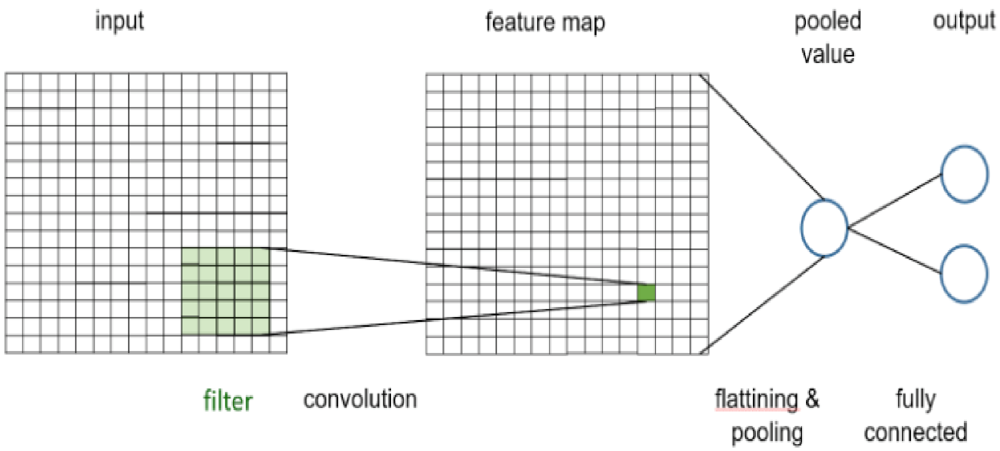


no padding to only use
valid input information

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

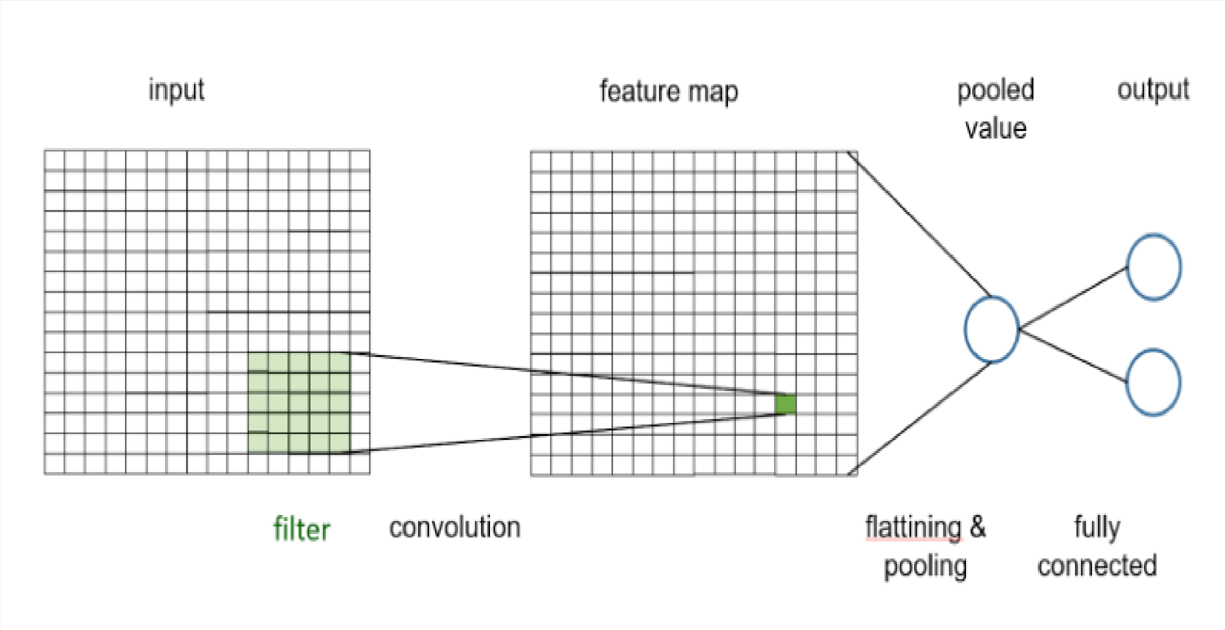
Taken from the great website on convolution: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

Building a very simple CNN with keras



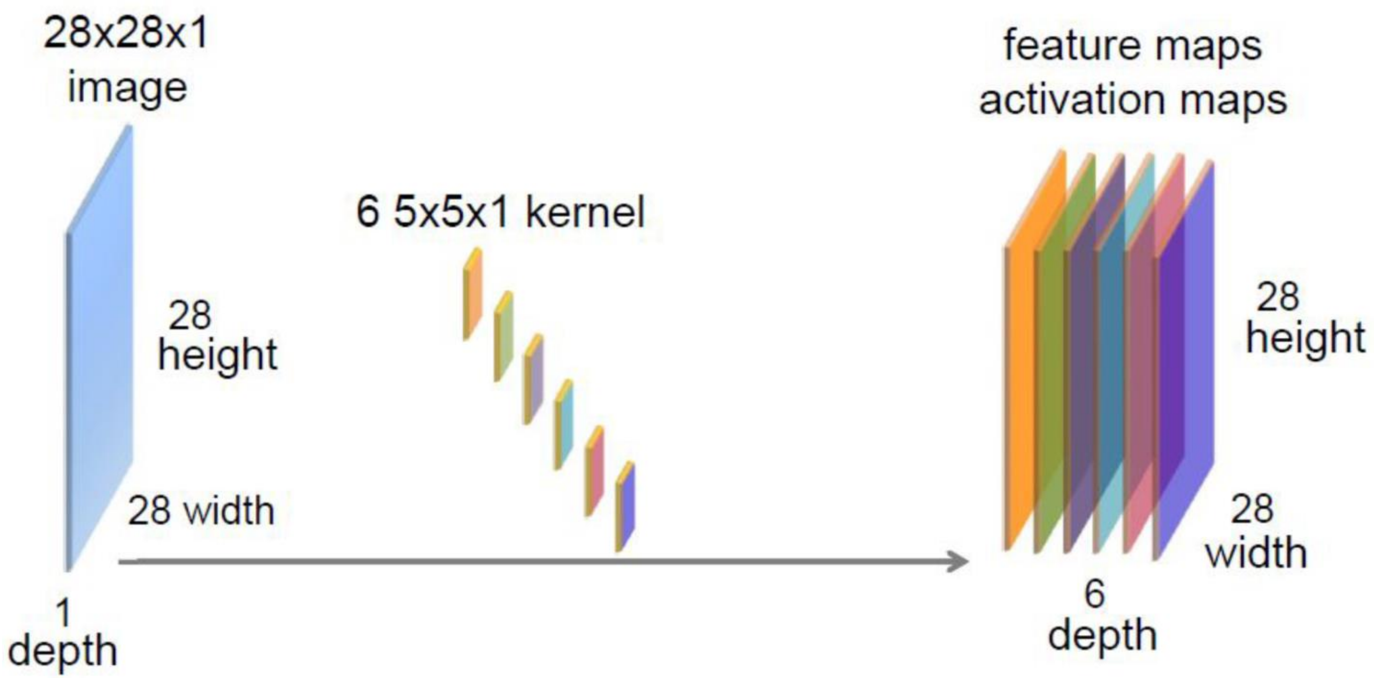

```
model = Sequential()
model.add(Convolution2D(1,(5,5), # one 5x5 kernel
         padding='same',           # zero-padding to preserve size
         input_shape=(pixel,pixel,1)))
model.add(Activation('linear'))
# take the max over all values in the activation map
model.add(MaxPooling2D(pool_size=(pixel,pixel)))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

Exercise: Artstyle Lover



Open NB in: https://github.com/tensorchiefs/dl_course_2020/blob/master/notebooks/05_cnn_edge_lover.ipynb

Convolution layer with a 1-chanel input and 6 kernels

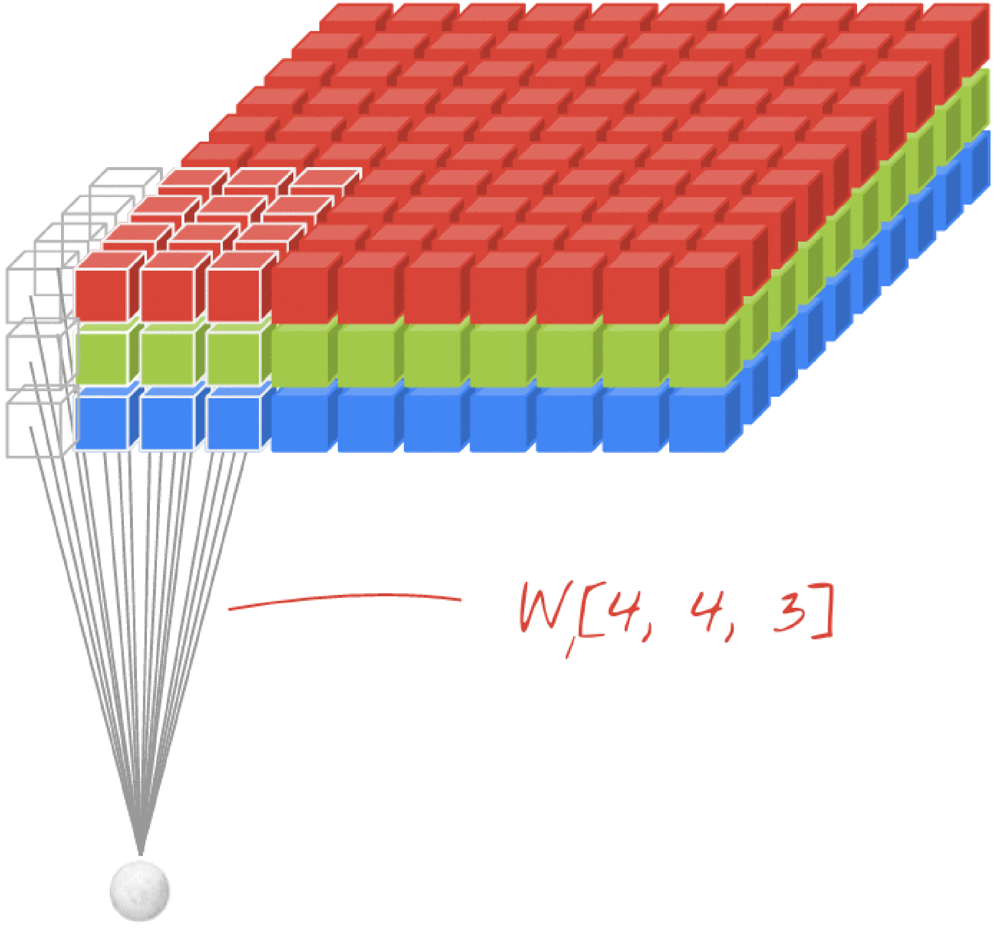


Convolution of the input image with 6 different kernels results in 6 activation maps.

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

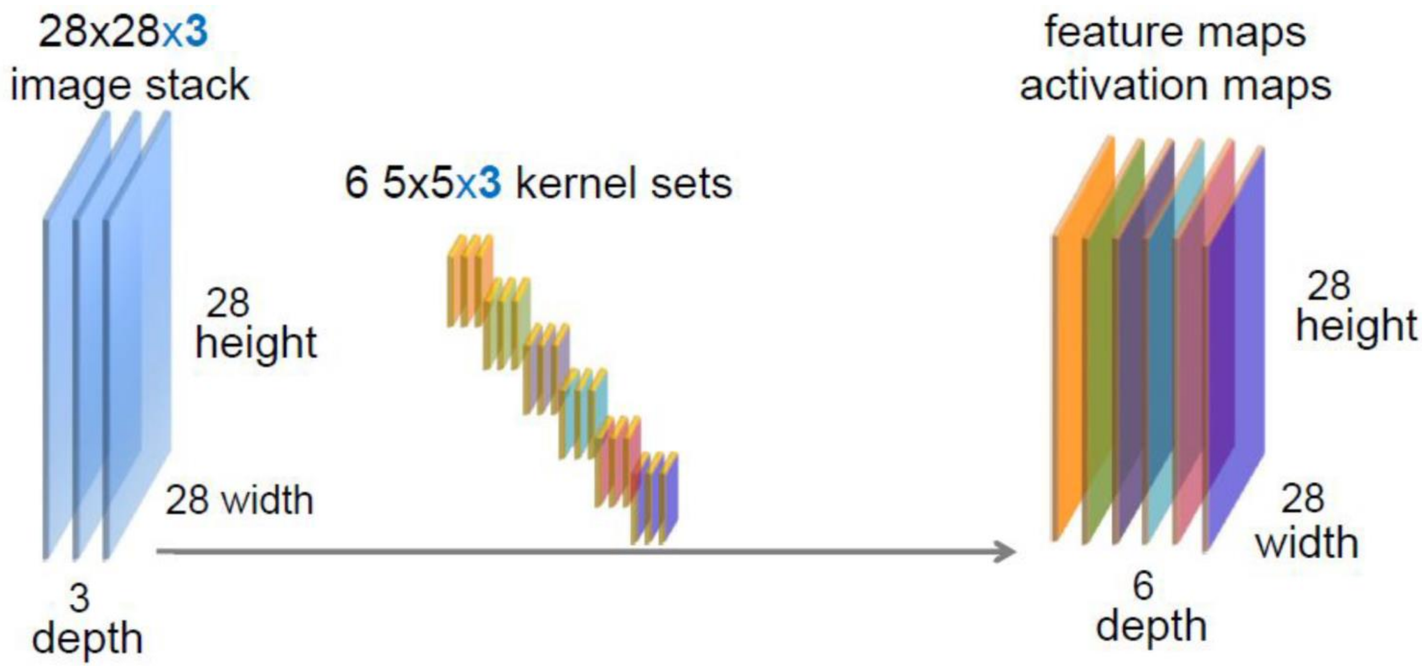
Animated convolution with 3 input channels

3 color channel input image



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

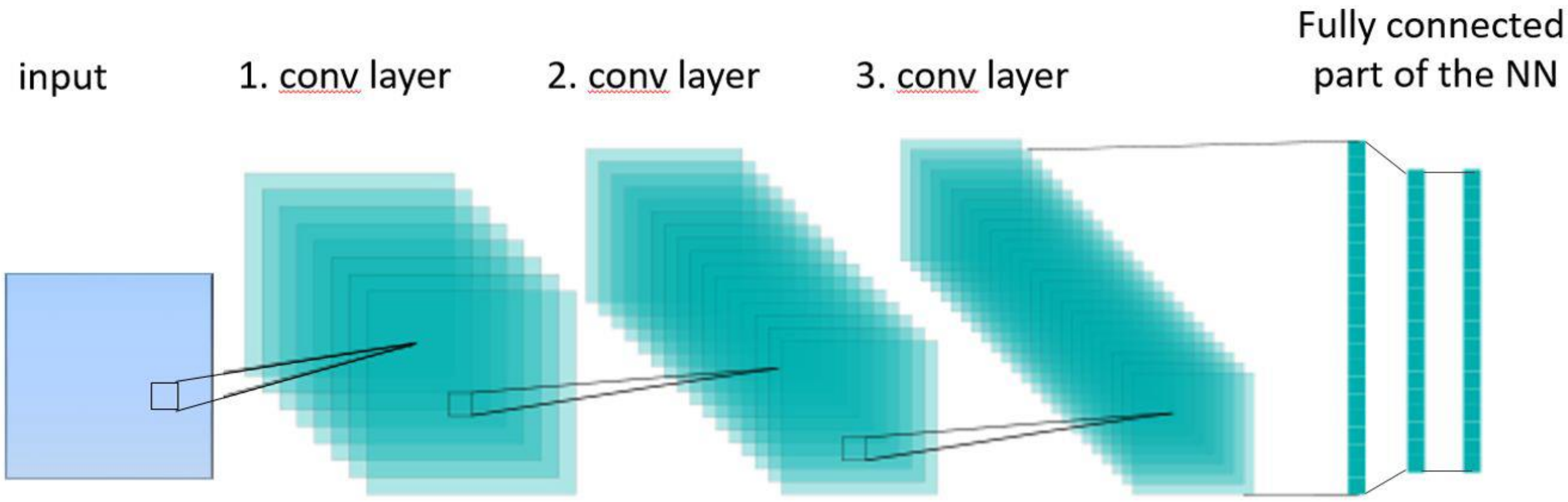
Convolution layer with a 3-chanel input and 6 kernels



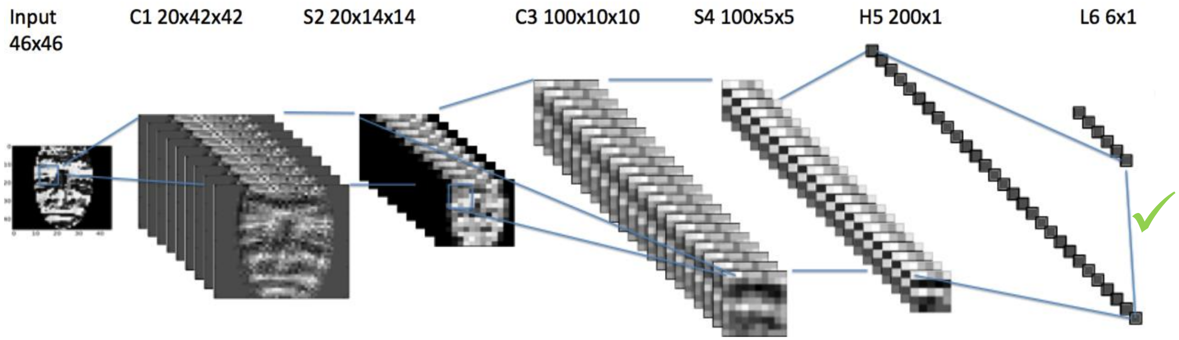
Convolution of the input image with 6 different kernels results in 6 activation maps.

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

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

→

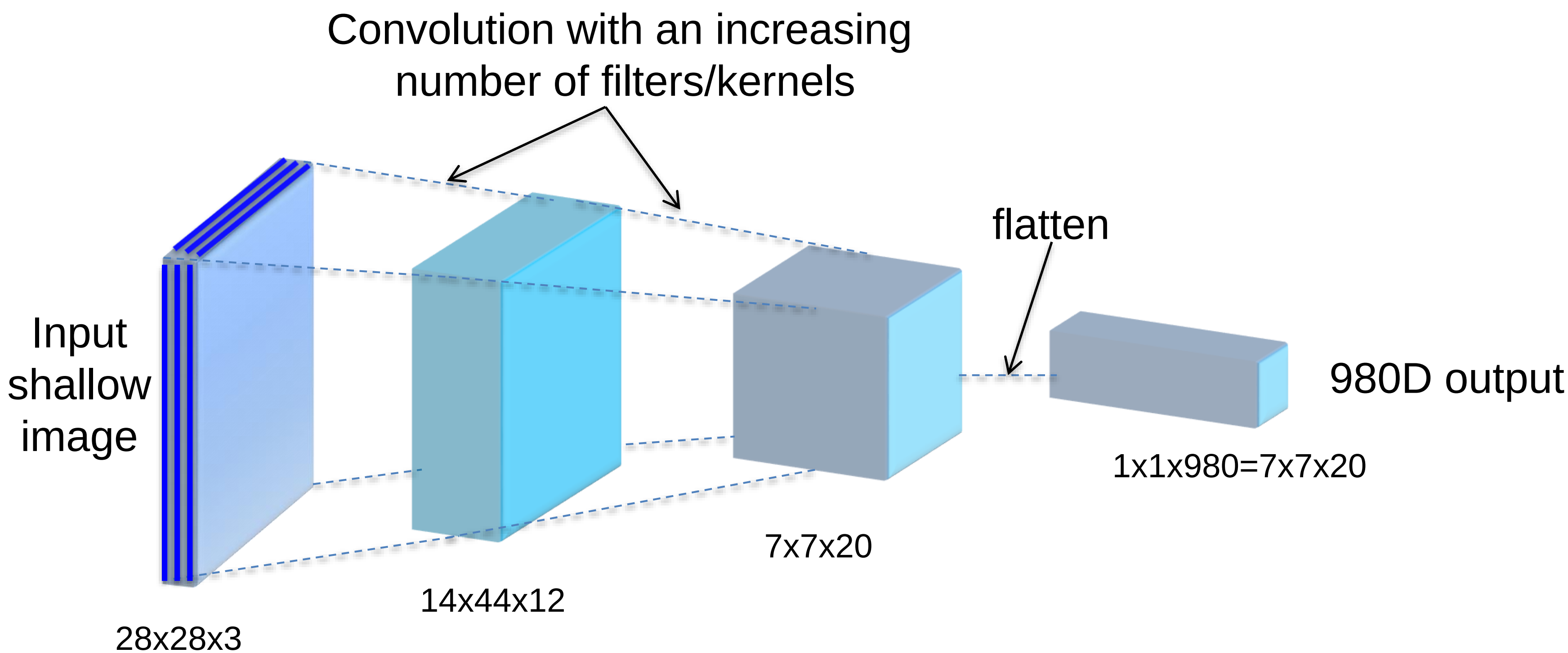
2x2

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

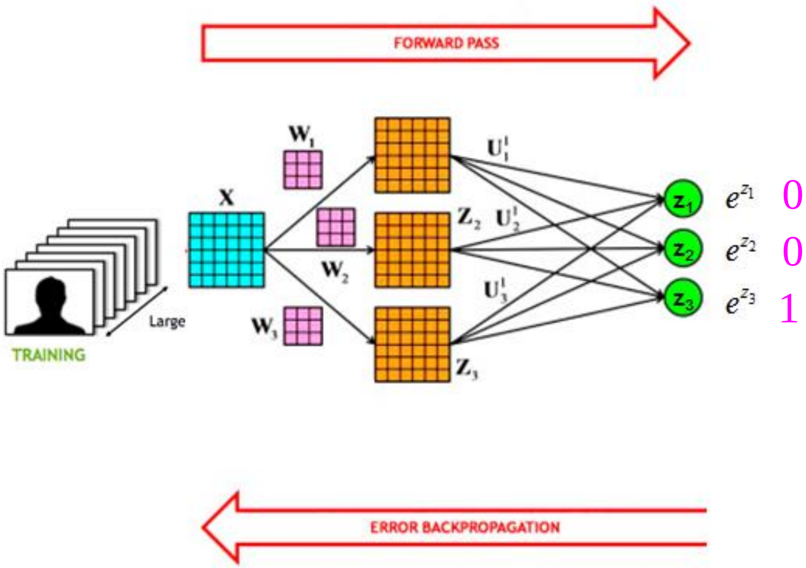
Hinton: „The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster“

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



Backpropagation="chain rule"

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)**)**

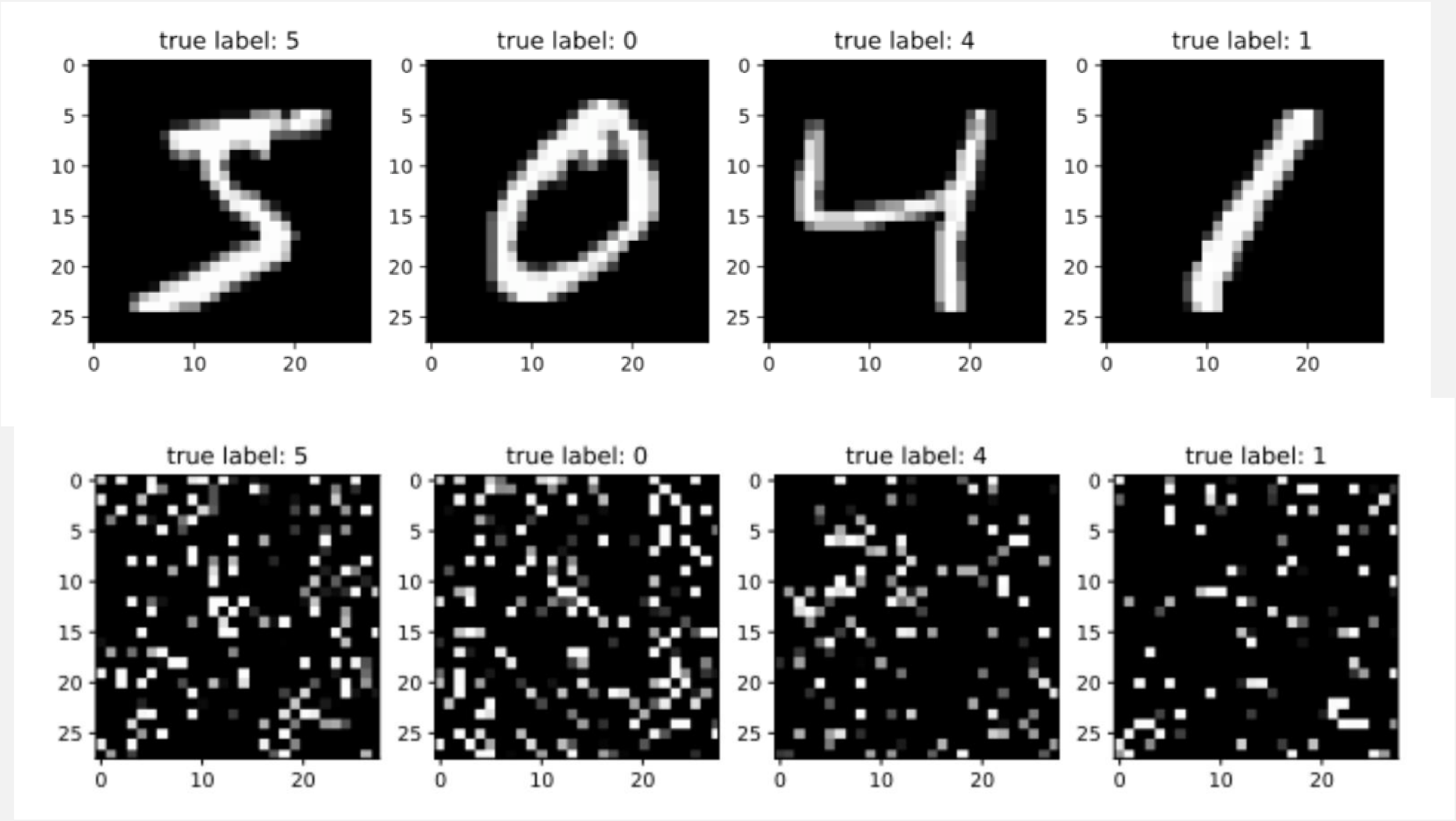
Learning is done by weight updating:

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

learning rate

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

Exercise: Does shuffling disturb a CNN?

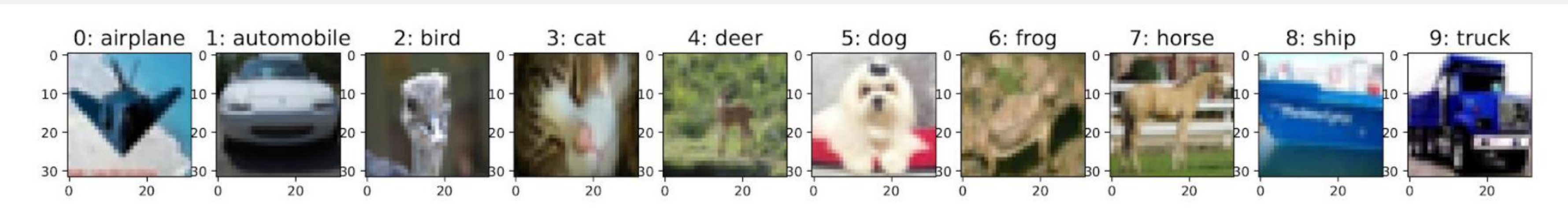


Open NB in: https://github.com/tensorchiefs/dl_course_2020/blob/master/notebooks/06_cnn_mnist_shuffled.ipynb

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 model bias, a CNN has the 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

Homework: Develop a CNN for cifar10 data



Develop a CNN to classify cifar10 images (we have 10 classes)

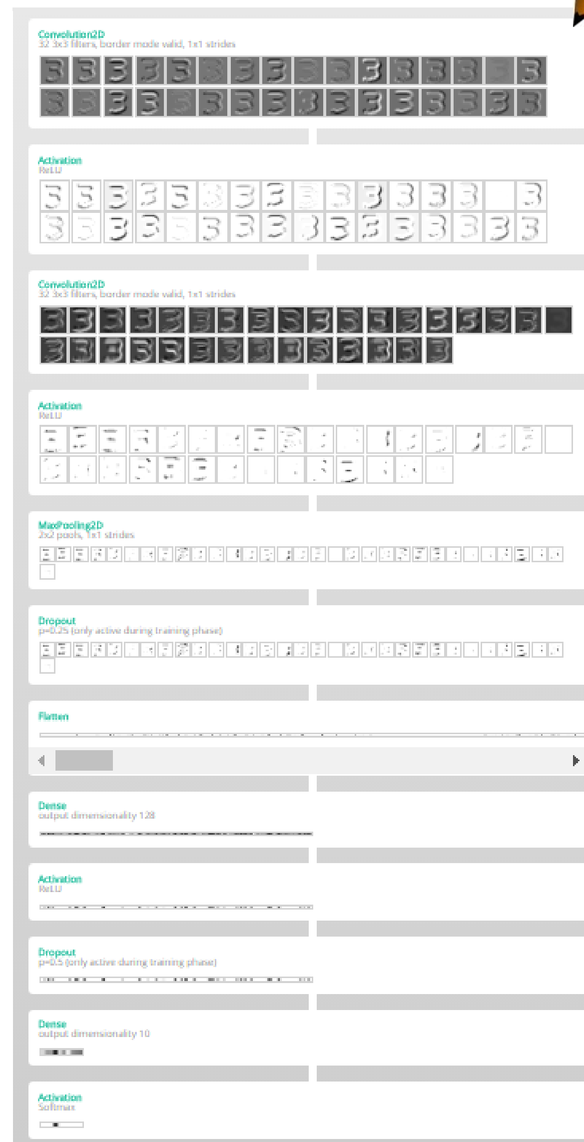
Investigate the impact of standardizing the data on the performance

Notebook for homework will soon be on the course webpage.

Homework: a simple live CNN: fill the gaps (optional)

Follow the first link [“live cnn in browser”](https://transcranial.github.io/keras-js/#/mnist-cnn)
https://transcranial.github.io/keras-js/#/mnist-cnn

```
model = Sequential()
model.add(Convolution2D(... , ..., ...,
                border_mode='valid',
                input_shape=(... ,... ),
                dim_ordering='tf'))
model.add(Activation('..............'))
model.add(Convolution2D(... , ..., ...,
                border_mode='valid',
                dim_ordering='tf'))
model.add(Activation('..............'))
model.add(MaxPooling2D(pool_size=(... , ...),
                border_mode='valid',
                dim_ordering='tf'))
model.add(Dropout(..............))
model.add(Flatten())
model.add(Dense(..............))
model.add(Activation('relu'))
model.add(Dropout(..............))
model.add(Dense(..............))
model.add(Activation('softmax'))
```



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.