

# Machine Intelligence:: Deep Learning

## Week 2

*Beate Sick, Elvis Murina, Oliver Dürr*

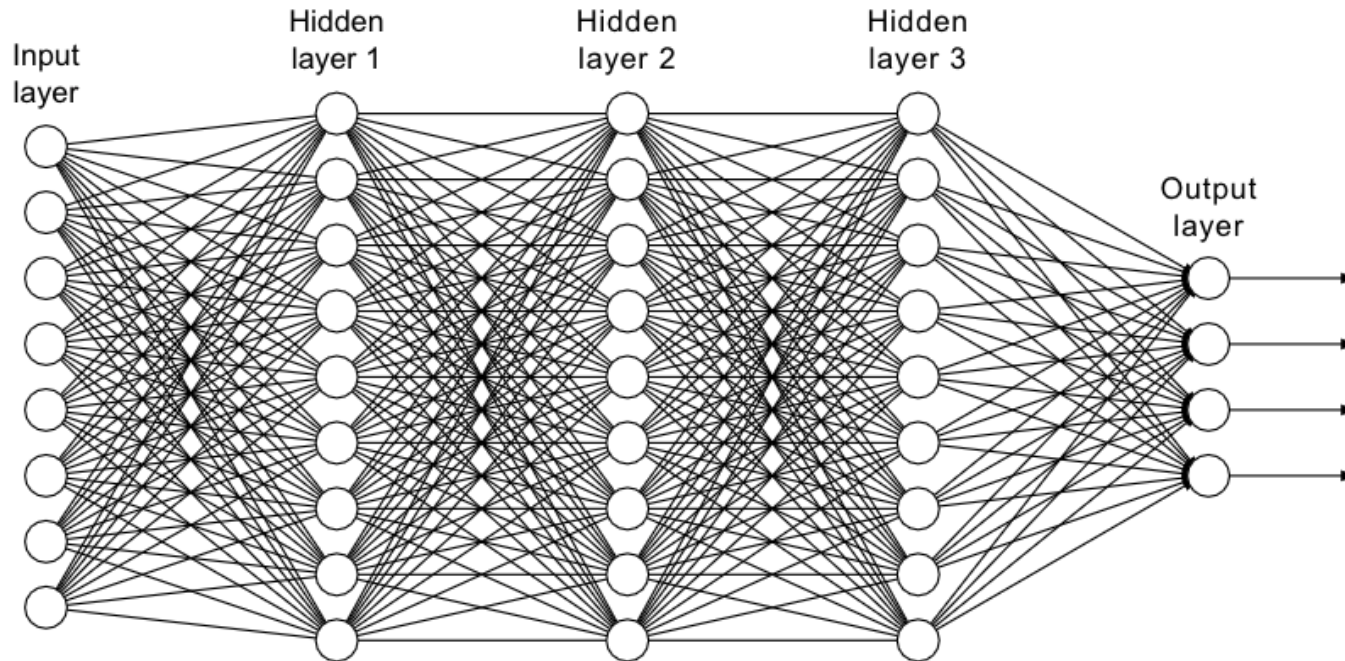
Institut für Datenanalyse und Prozessdesign  
Zürcher Hochschule für Angewandte Wissenschaften

This slide deck is work in progress and might change until Tuesday.

# 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



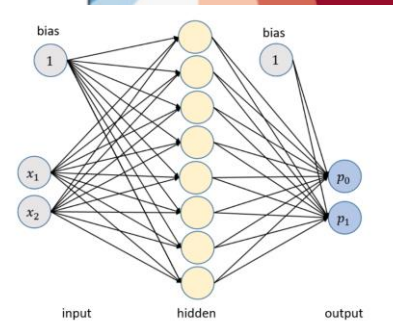
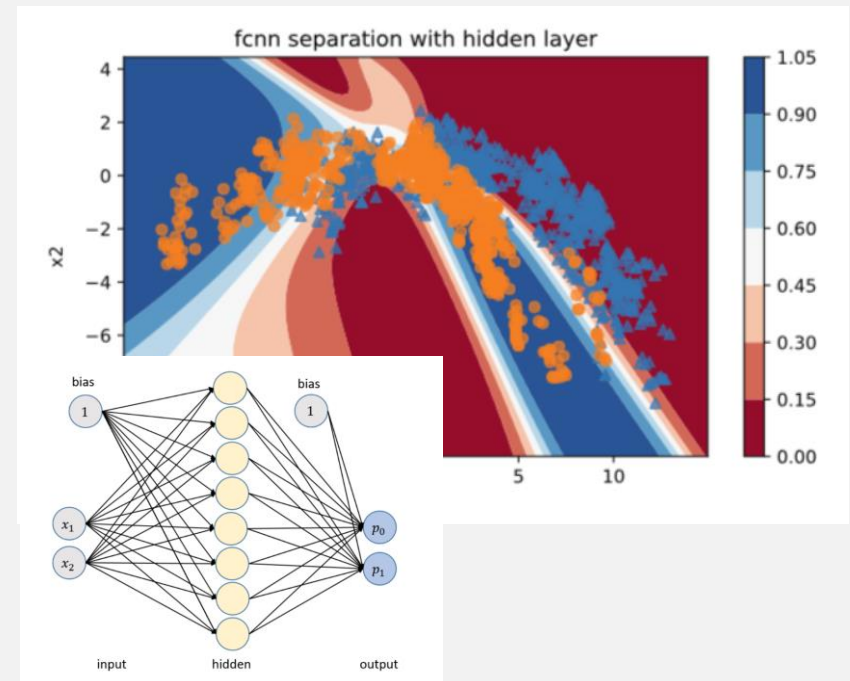
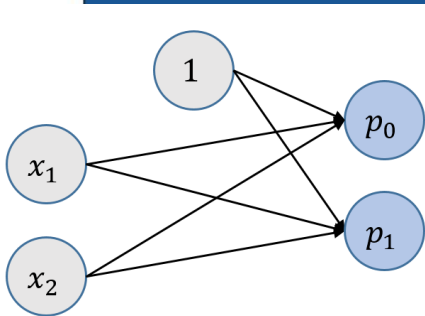
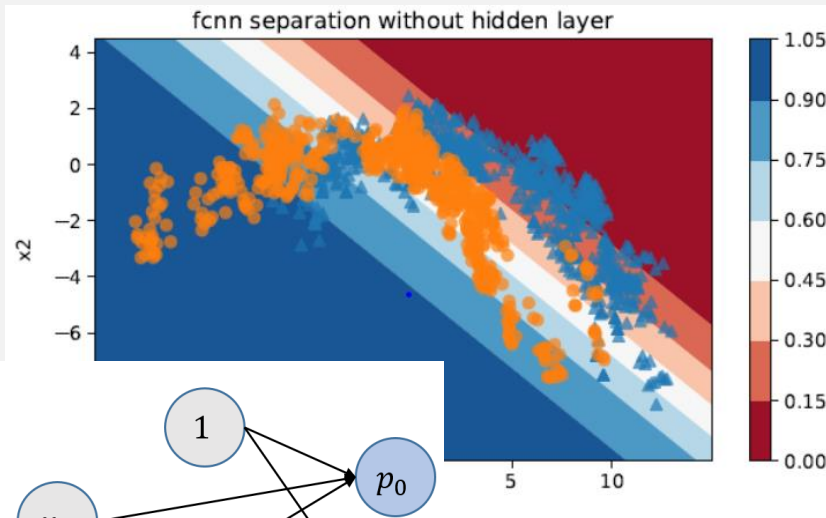
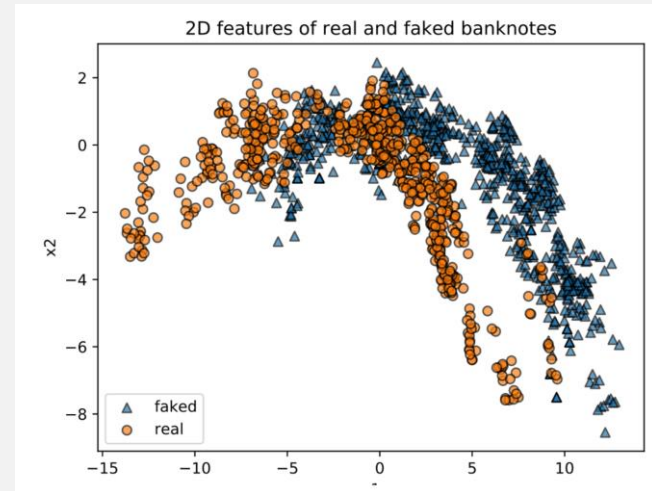
Each neuron in a fcNN gets as input a weighted sum of all neuron activation from one layer below. Different neurons in the same layer have different weights in this weighted sum, which are learned during training.

# Homework

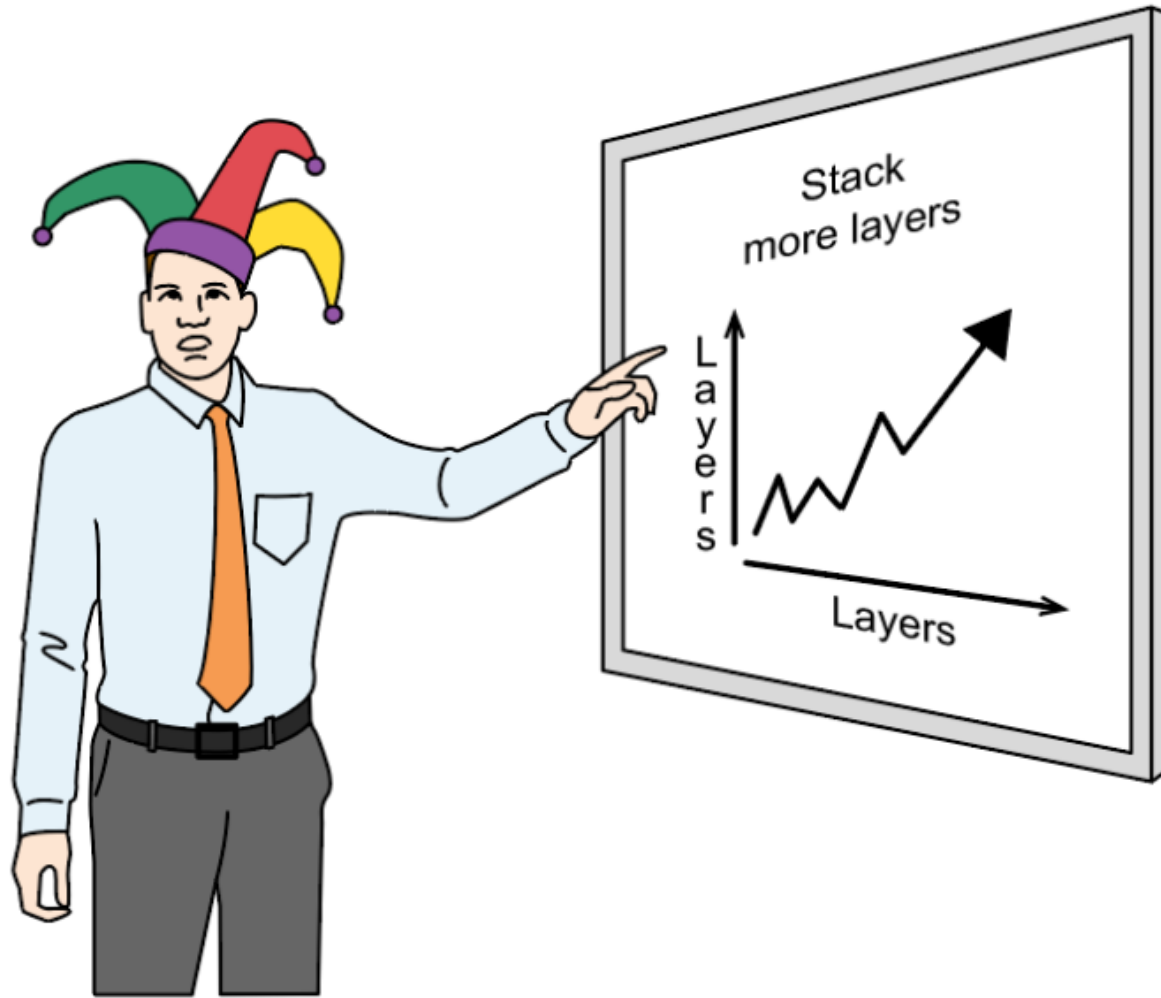
Do exercise NB 02: Classify

banknotes based on 2 features ( $x_1, x_2$ )

[https://github.com/tensorchiefs/dl\\_course\\_2020/blob/master/notebooks/02\\_fcnn\\_with\\_banknote.ipynb](https://github.com/tensorchiefs/dl_course_2020/blob/master/notebooks/02_fcnn_with_banknote.ipynb)



# A deep learning expert at work



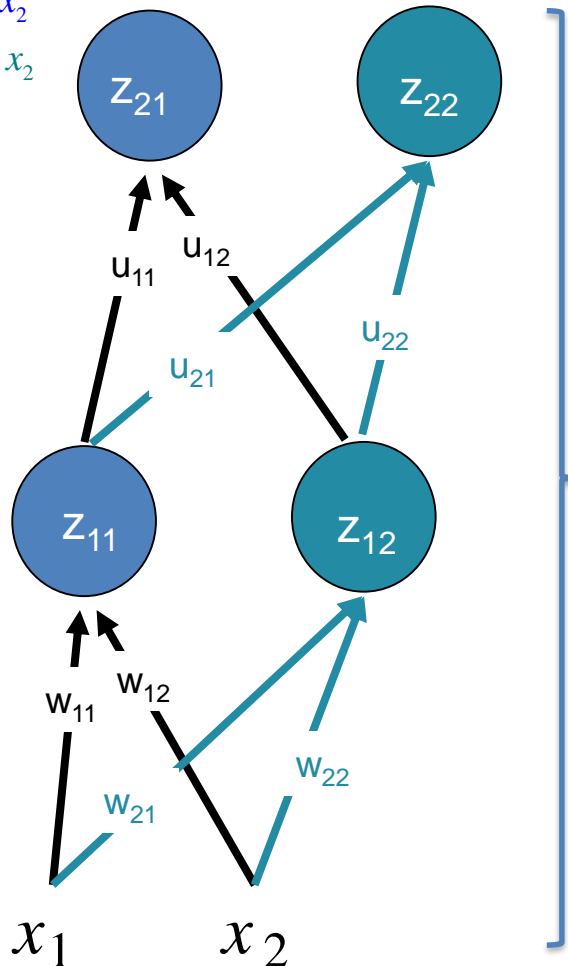
# To go deep non-linear activation functions are needed

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

$$\begin{aligned} z_{21} &= z_{11} \cdot u_{11} + z_{12} \cdot u_{12} = (w_{11} \cdot x_1 + w_{12} \cdot x_2) \cdot u_{11} + (w_{21} \cdot x_1 + w_{22} \cdot x_2) \cdot u_{12} \\ &= x_1 \cdot (w_{11} \cdot u_{11} + w_{21} \cdot u_{12}) + x_2 \cdot (w_{12} \cdot u_{11} + w_{22} \cdot u_{12}) \end{aligned}$$

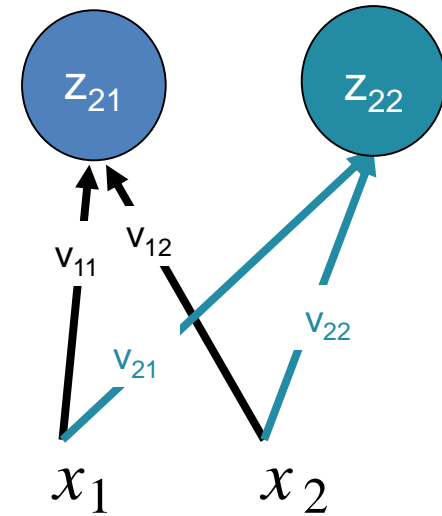
$$z_{11} = w_{11} \cdot x_1 + w_{12} \cdot x_2$$

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



=

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



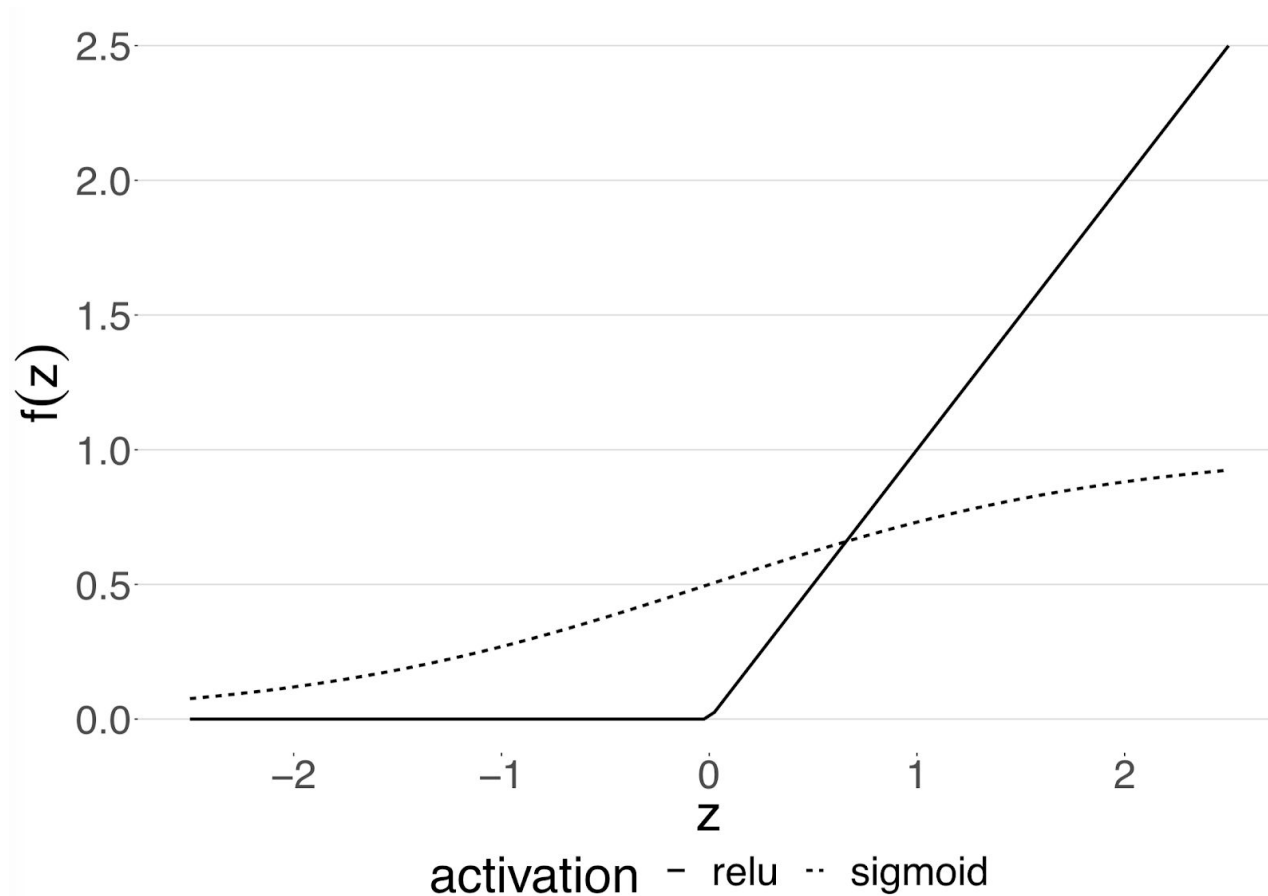
$$v_{11} = w_{11} \cdot u_{11} + w_{21} \cdot u_{12}$$

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

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

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

# Comon non-linear activation function



The sigmoid has small gradients for values far away from zero.  
ReLU clips values below zero and let values  $> 0$  pass unchanged.

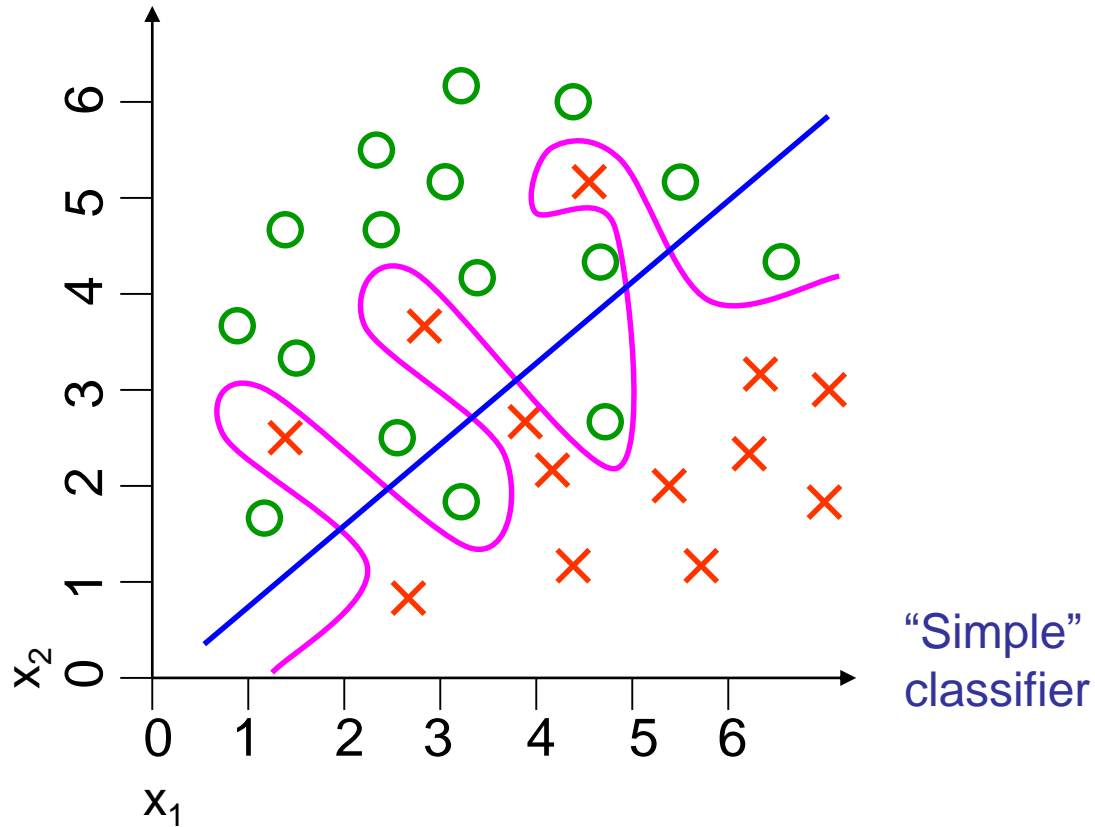
# Model development

## Overfitting and underfitting



# "Perfect" Vs. "Simple" classifier

"Perfect"  
classifier



Which is better?

Check on a test-set (cross validation).

# Cross validation of the “simple” classifier

Training set:

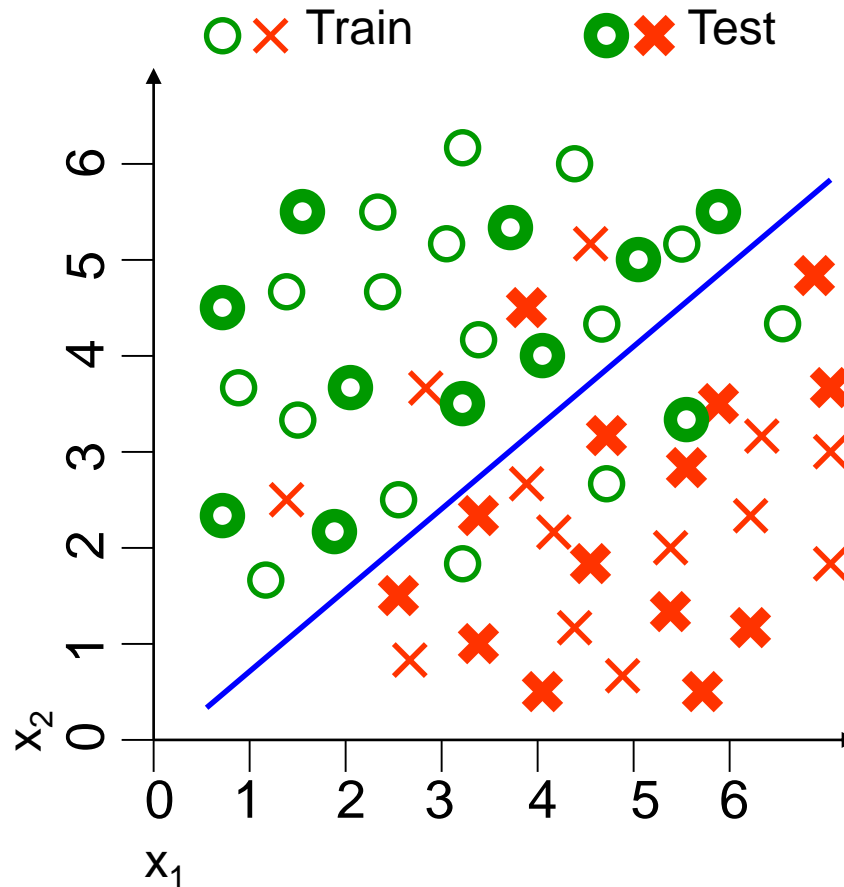
6/29=20%

misclassification

Test set:

2/25=8%

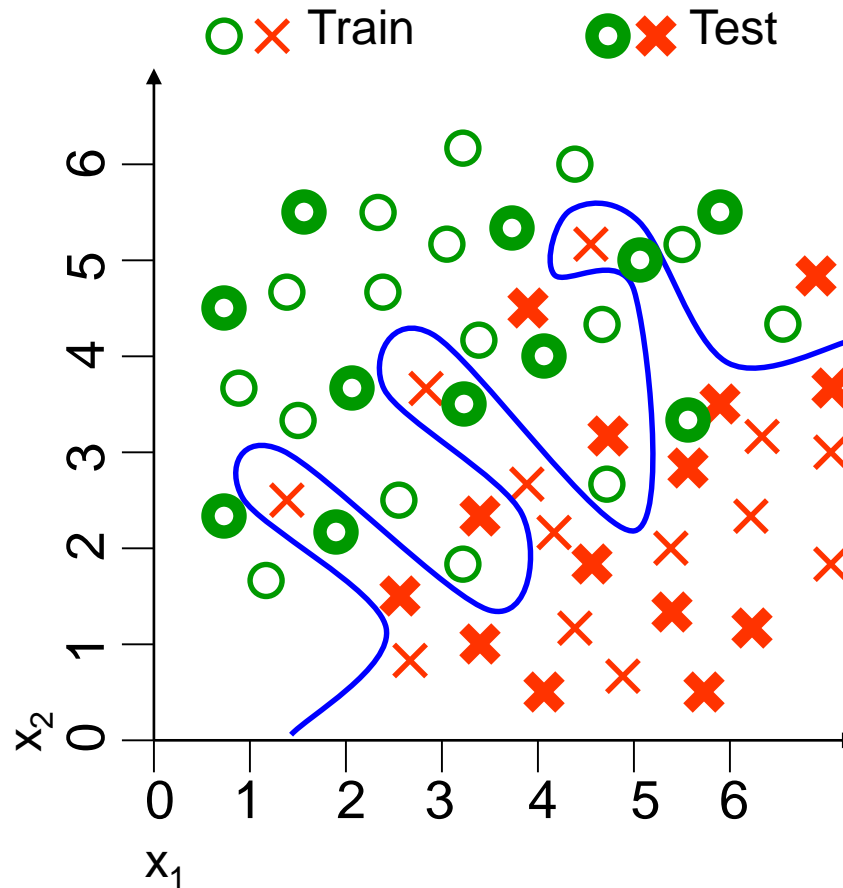
misclassification



# Cross validation of the “Perfect” classifier

Training set:  
0% misclassification

Test set:  
8/25=24%  
misclassification



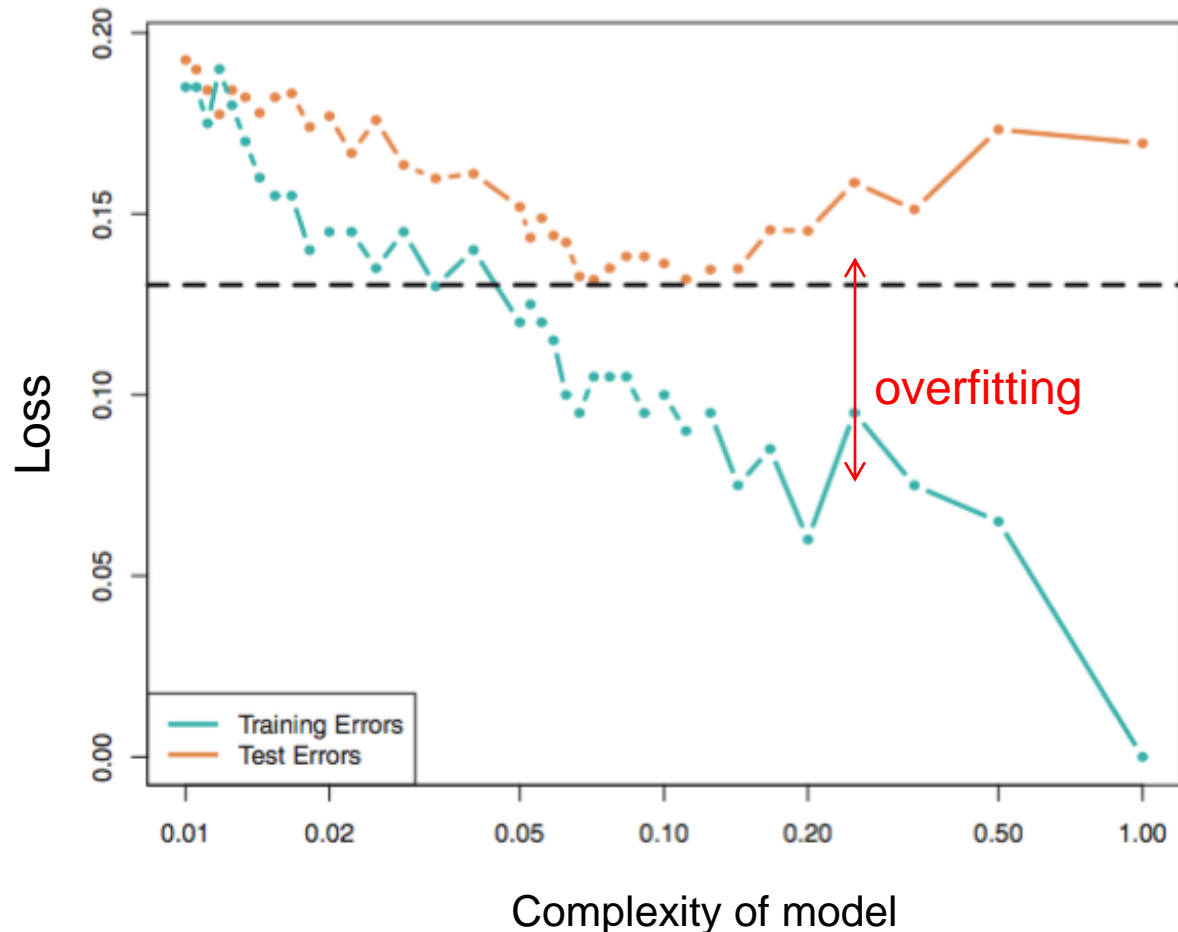
# Types of Errors

- **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:**

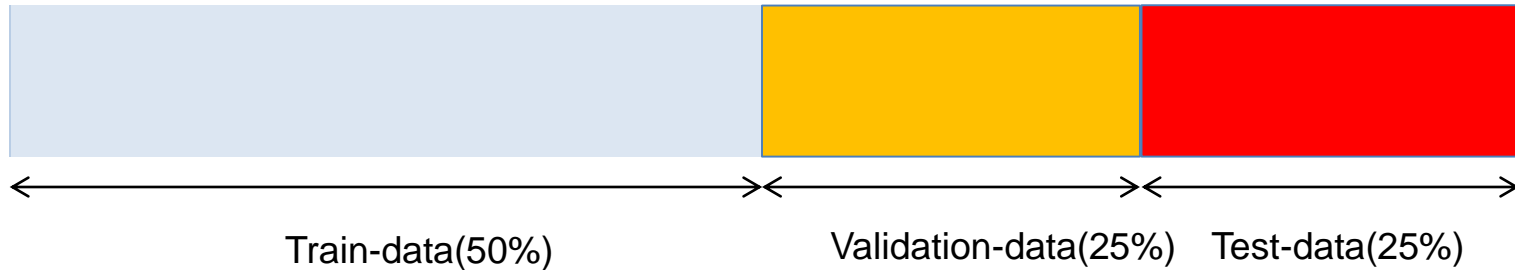
- Model fits the training data well (small training error) but shows high generalization error

# 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.

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



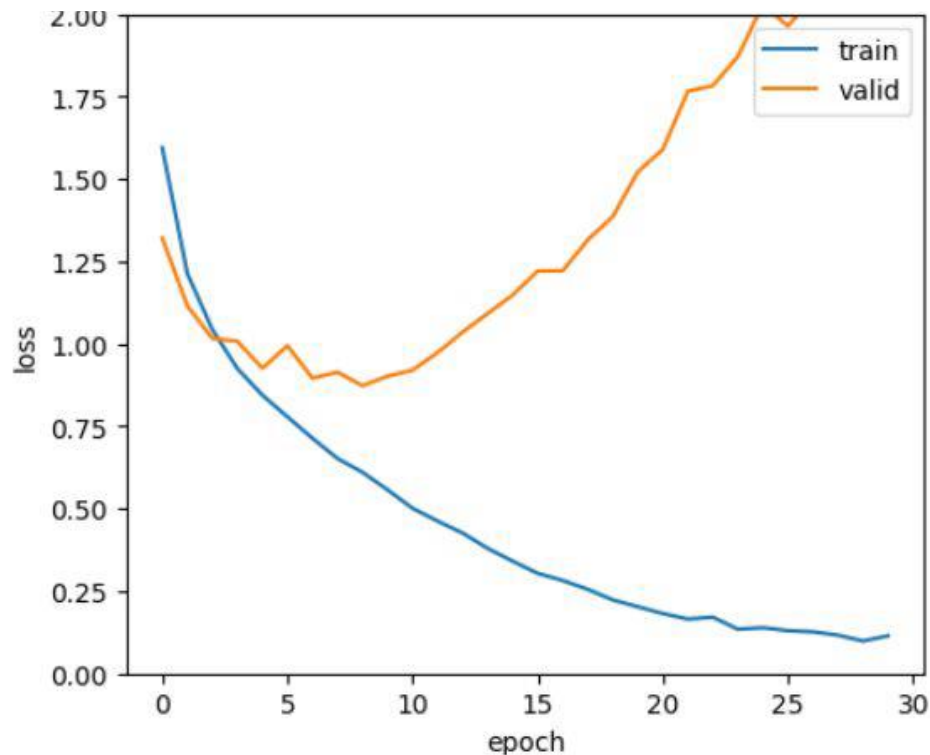
Best practice: Lock an extra **test data set** away, and use it only at the very end, to evaluate the chosen model, that performed best on your validation set.

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?

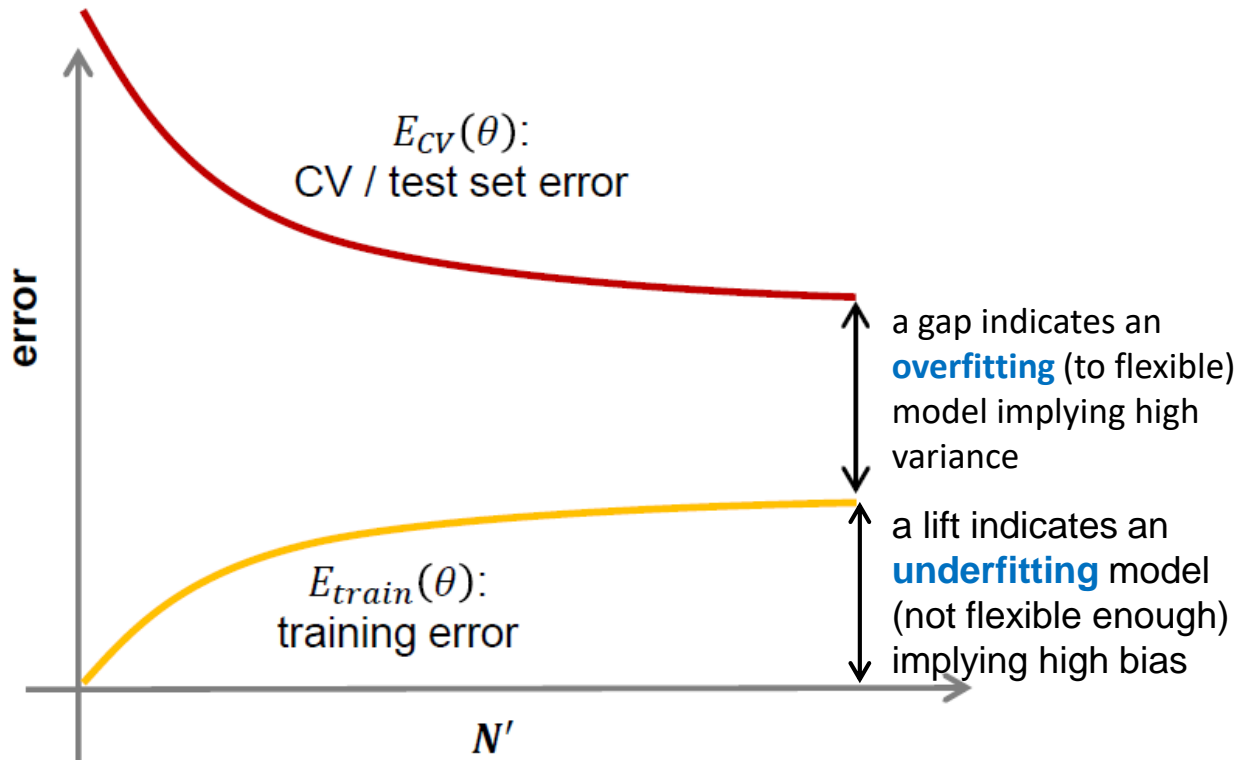
Very common check: Plot loss in train and validation data vs epoch of training.



- If training loss does not go down to zero: model is not flexible enough
- In case of overfitting (validation loss  $\gg$  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:

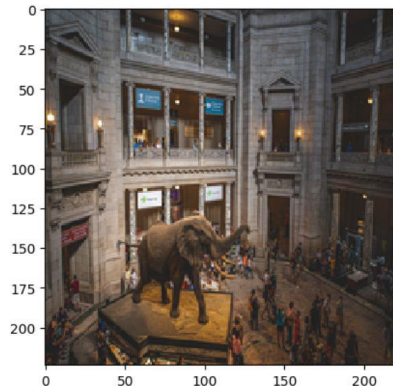
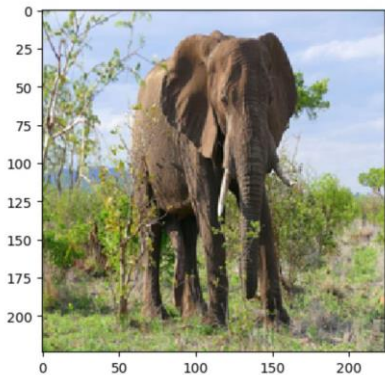
- Use more complex model (less regularization)
- Use other model structure
- boosting



# Model training process relies on “the big lie”

$P(\text{train}) = P(\text{test}) = P(\text{test})$ : The “big lie”

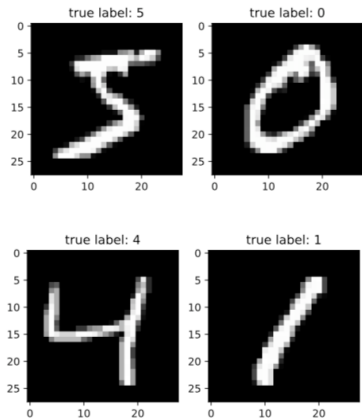
- In traditional machine learning and DL we usually assume that the train and test data do come from the same distribution.
- If this assumption is not true, a seemingly high performant model breaks down and cannot see the elephant in the room.
- We need models that can flag uncertain predictions or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



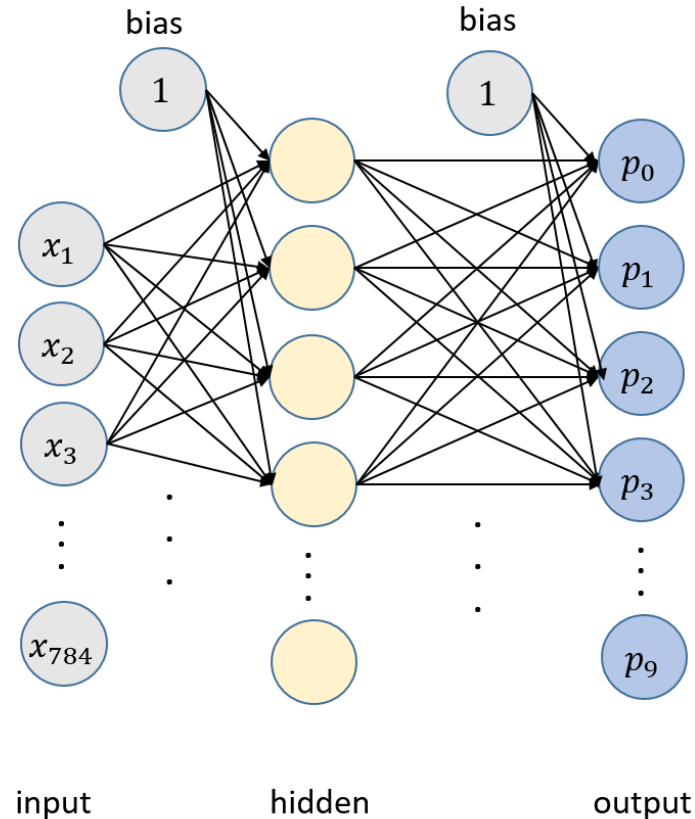
Remark: we will discuss this in more detail in later lectures and also learn about some cures.

Fully connected NN for image data  
Why not?

# A fcNN for MNIST data



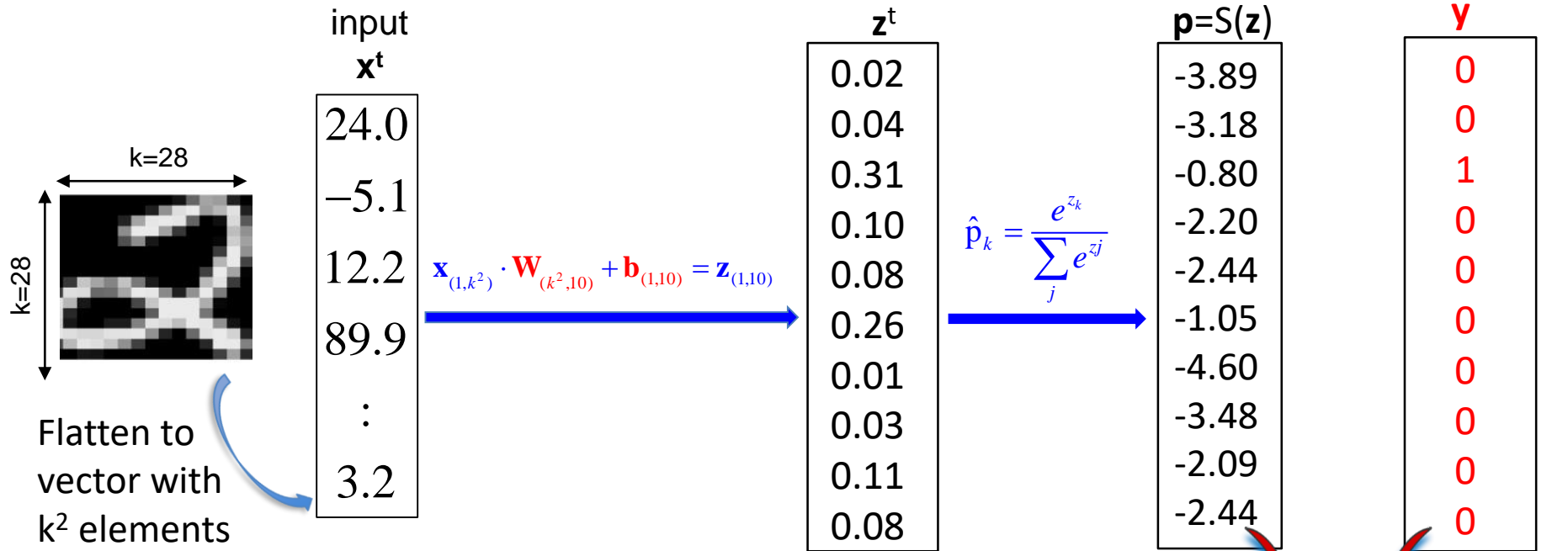
The first four digits of the MNIST data set - each image consisting of  $28 \times 28 = 784$  pixels



A fully connected NN with 2 hidden layers.

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

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

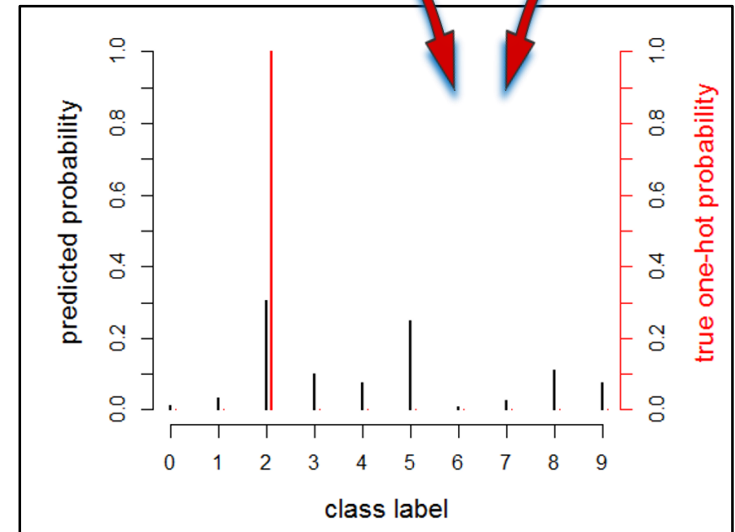


Cost C or Loss = cross-entropy averaged over all images in mini-batch

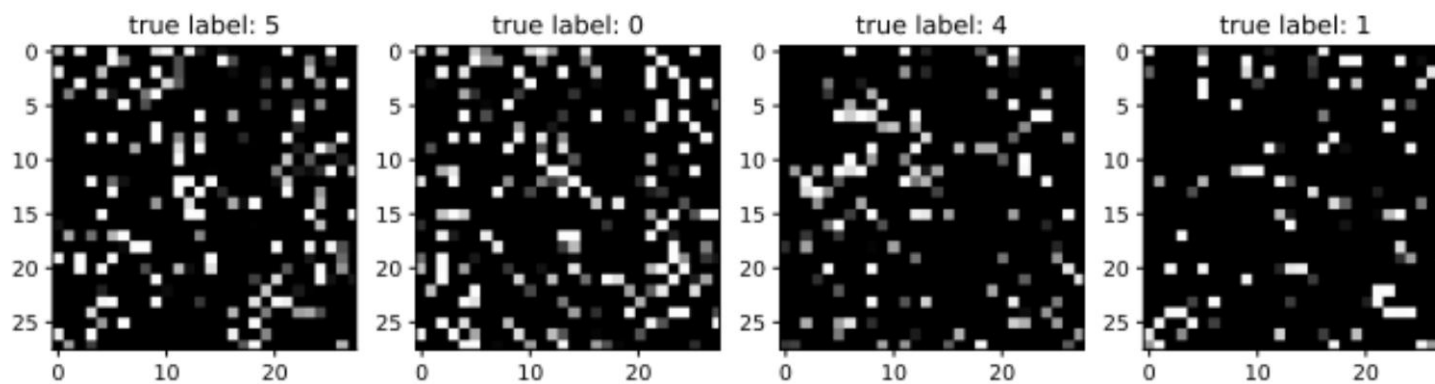
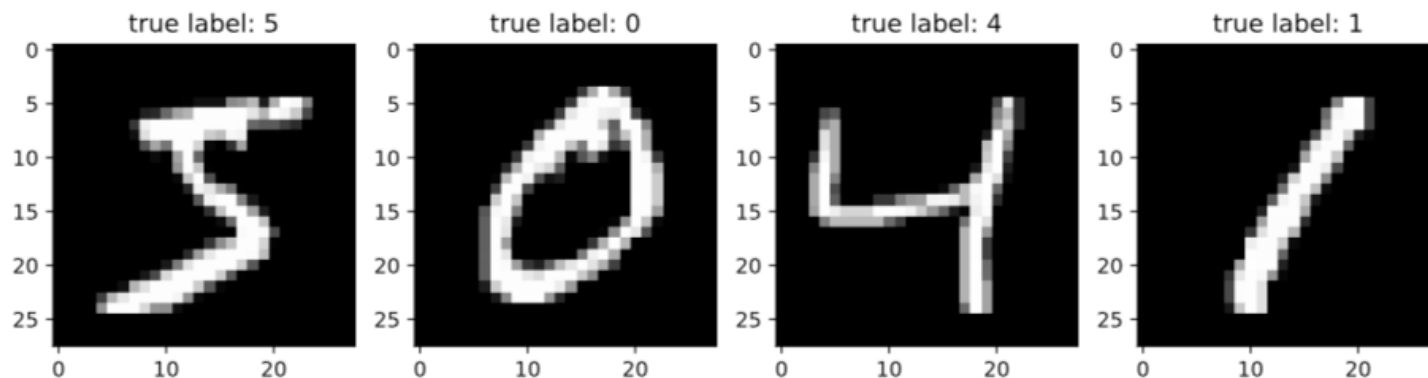
$$C = \frac{1}{N} \sum_i D(\mathbf{p}_i, \mathbf{y}_i)$$

Cross-Entropy

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



# Exercise: Does shuffling disturb a fcNN?



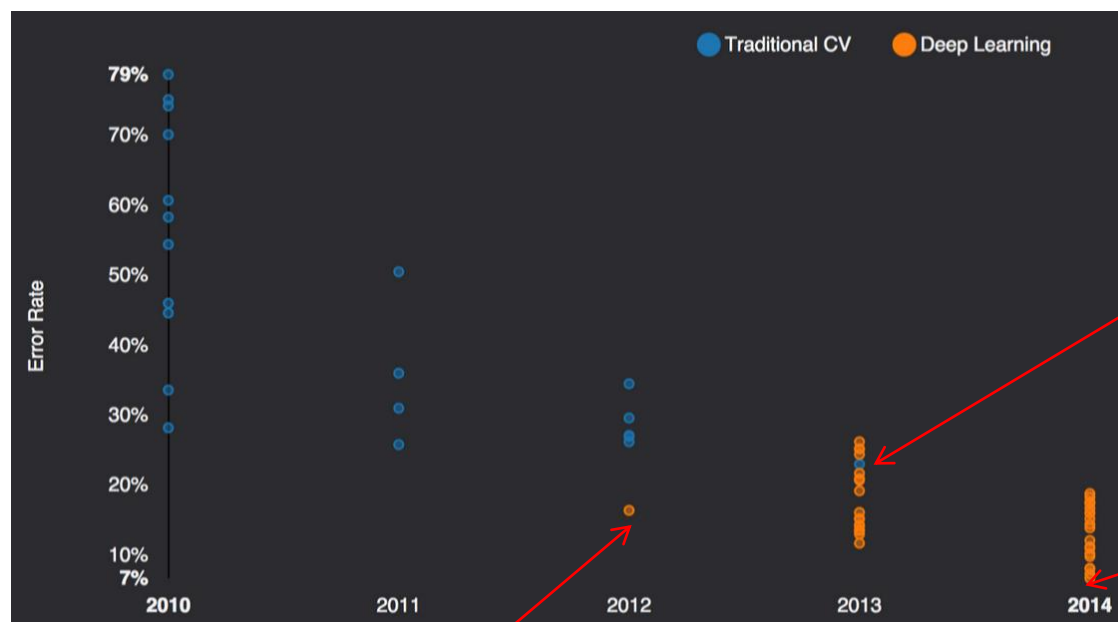
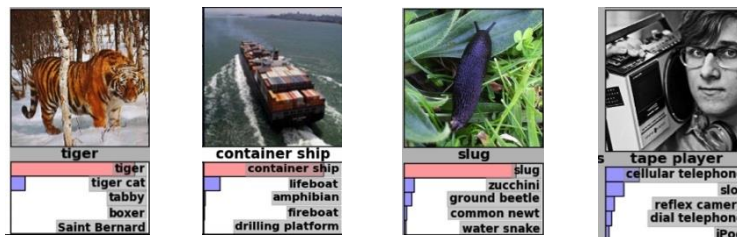
Open NB in:

# 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%)

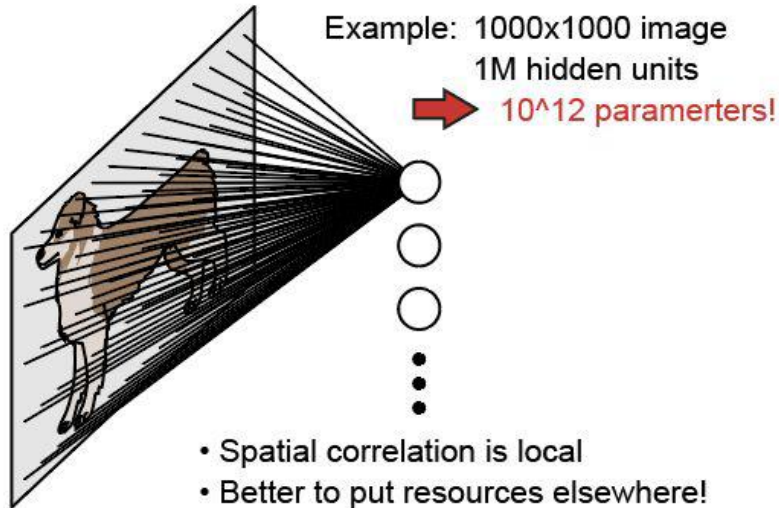
4.8% Google ([Feb 11](#)) -> further improved to 3.6 (Dec)?

4.58% Baidu (May 11 [banned due too many submissions](#))

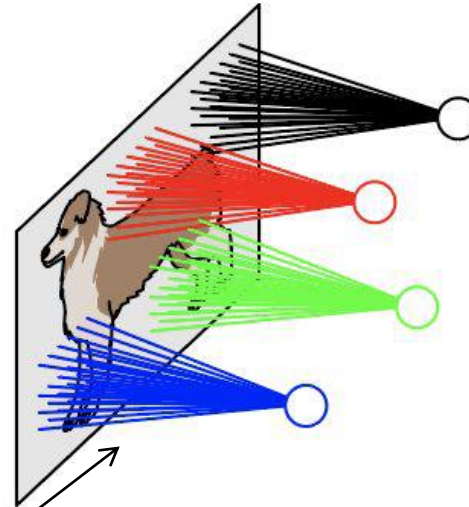
3.57% Microsoft (Resnet winner 2015)

# Convolution extracts local information using few weights

Fully connected neural net



Locally connected neural net

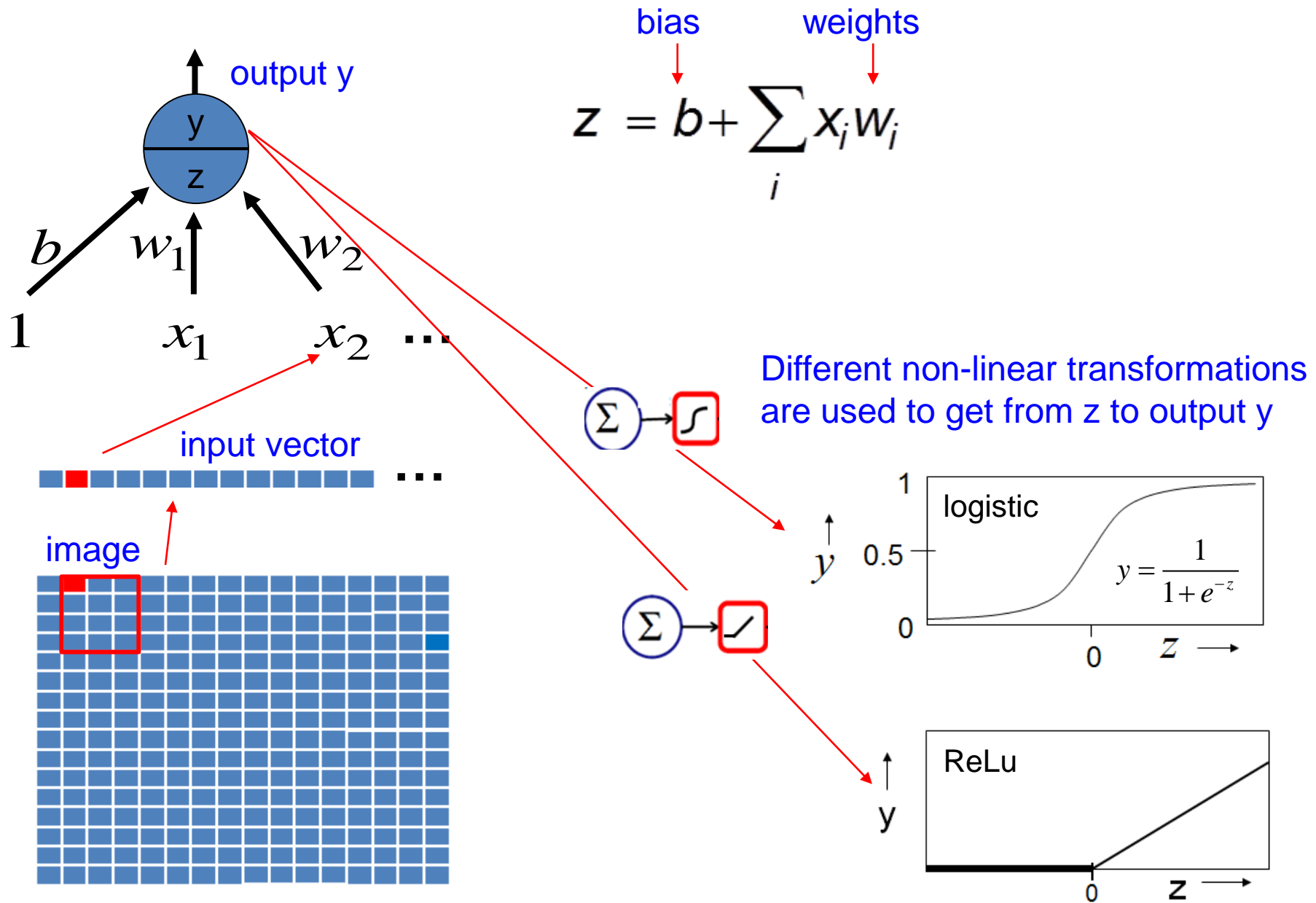


## Shared weights:

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 an edge.



# An artificial neuron



# 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 multiplied with weights of a small filter/kernel:

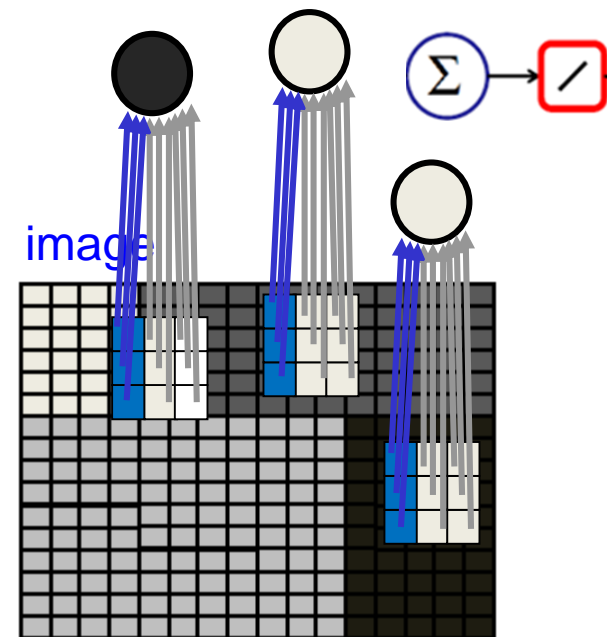
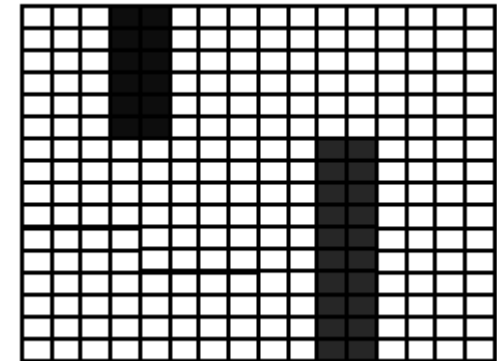
$w_1$	$w_2$	$w_3$
$w_4$	$w_5$	$w_6$
$w_7$	$w_8$	$w_9$

0.9	0.1	0.1
0.9	0.1	0.1
0.9	0.1	0.1

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



# Convolutional neural networks

Input image 6x6x1

255	220	150	200	110	100
240	50	35	45	200	130
0	20	245	250	230	120
170	180	235	145	170	255
190	185	170	165	130	120
255	255	245	190	200	175

-0.7	0.2	0.1
0.3	0.5	0.4
-0.2	-0.4	0.2

3x3 filter

$$z_j = \sum_i (x_i \cdot w_{ij})$$

## Exercise: Do one convolution step by hand

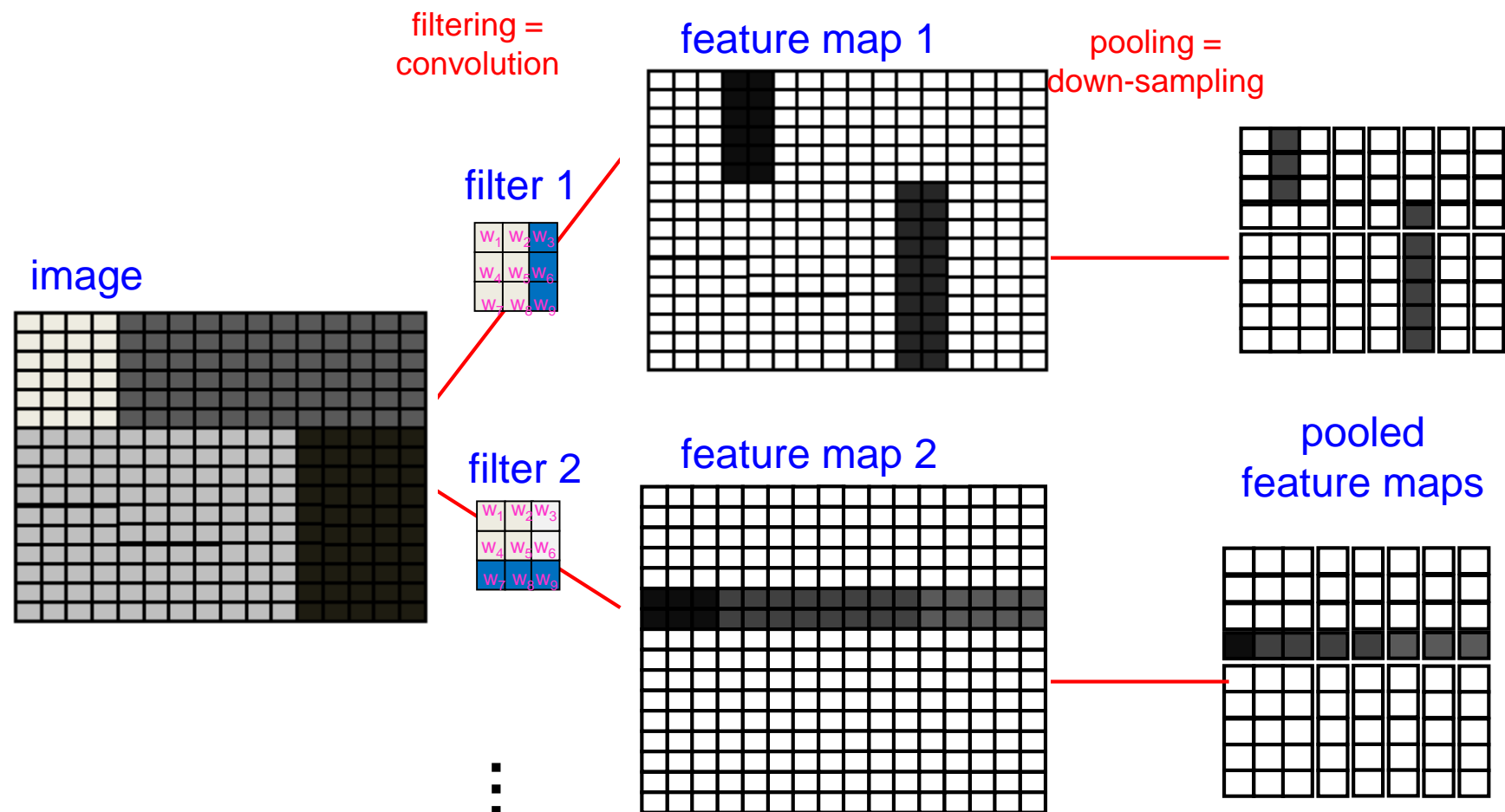


The kernel is 3x3 and is applied at each valid position  
– how large is the resulting activation map?

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

# Convolutional networks use neighborhood information and replicated local feature extraction



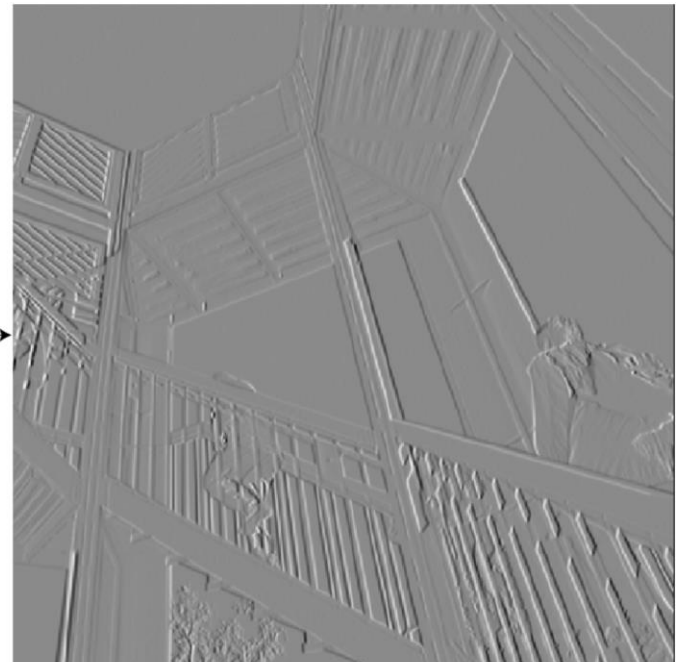
The weights of each filter are randomly initiated and then adapted during the training.

# Example of designed Kernel / Filter



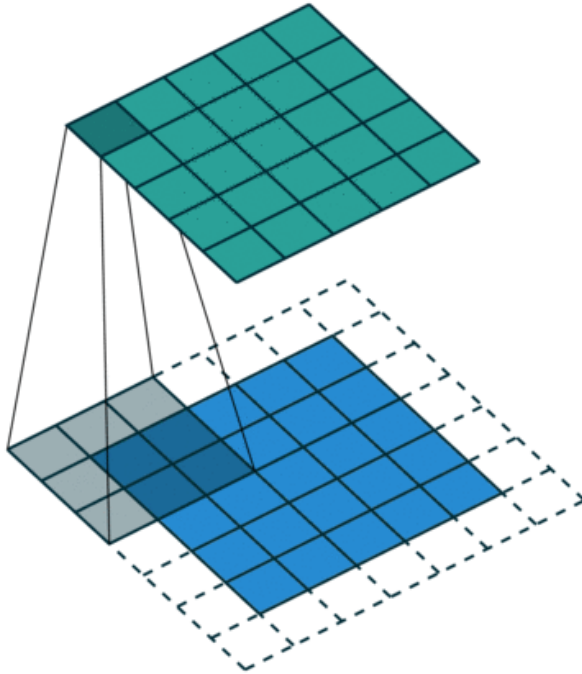
$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

Horizontal Sobel kernel

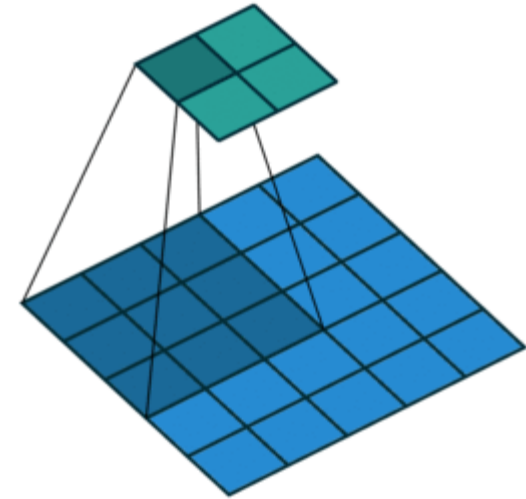


Applying a vertical edge detector kernel

# CNN Ingredient I: Convolution



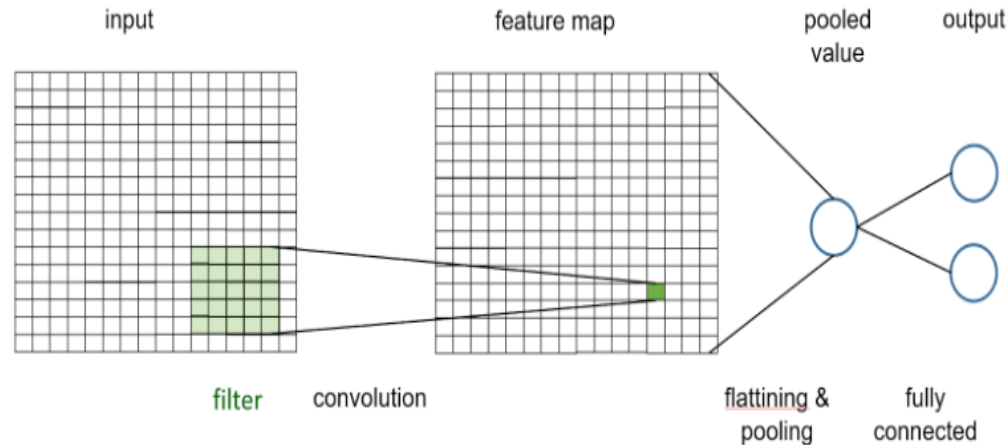
Zero-padding to achieve  
**same** size of feature and input



no padding to only use  
**valid** input information

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

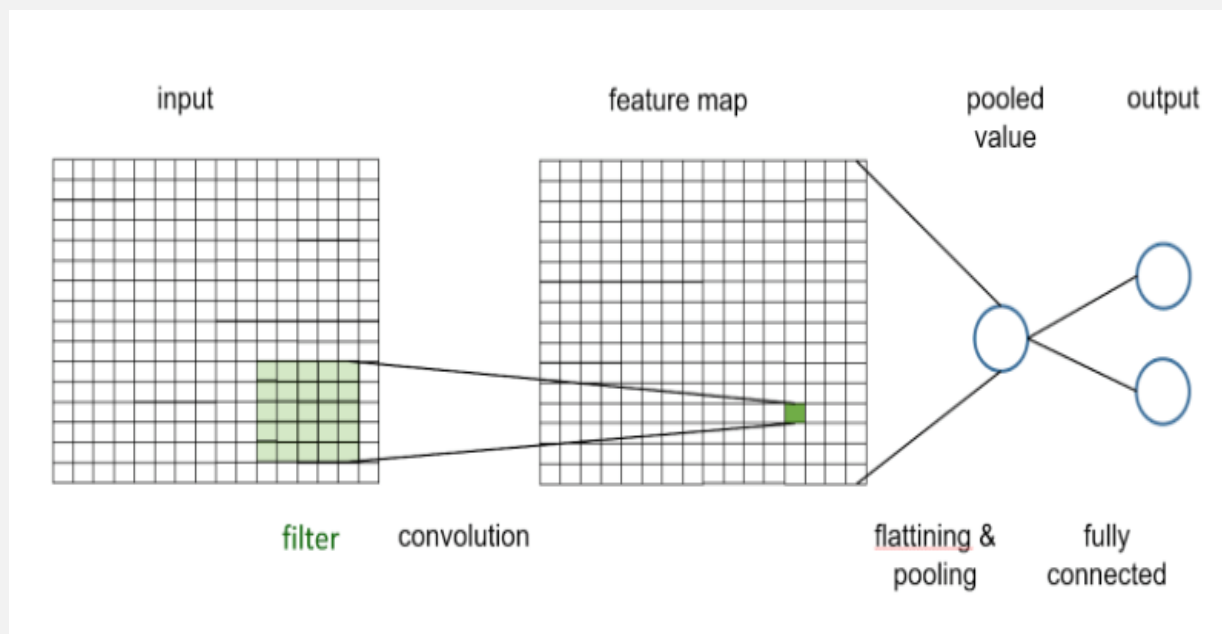
# Building a very simple CNN with keras



```
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'))
```

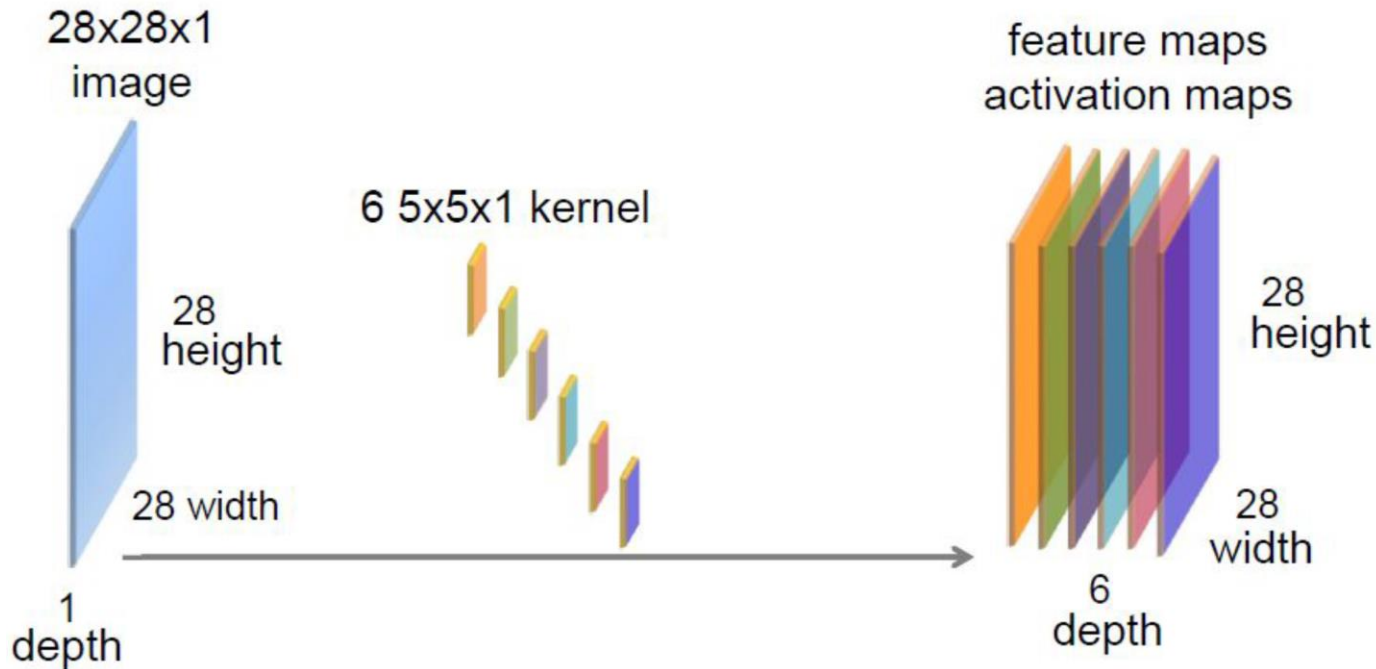


# Exercise: Artstyle Lover



Open NB in: [https://github.com/tensorchiefs/dl\\_book/blob/master/chapter\\_02/nb\\_ch02\\_03.ipynb](https://github.com/tensorchiefs/dl_book/blob/master/chapter_02/nb_ch02_03.ipynb)

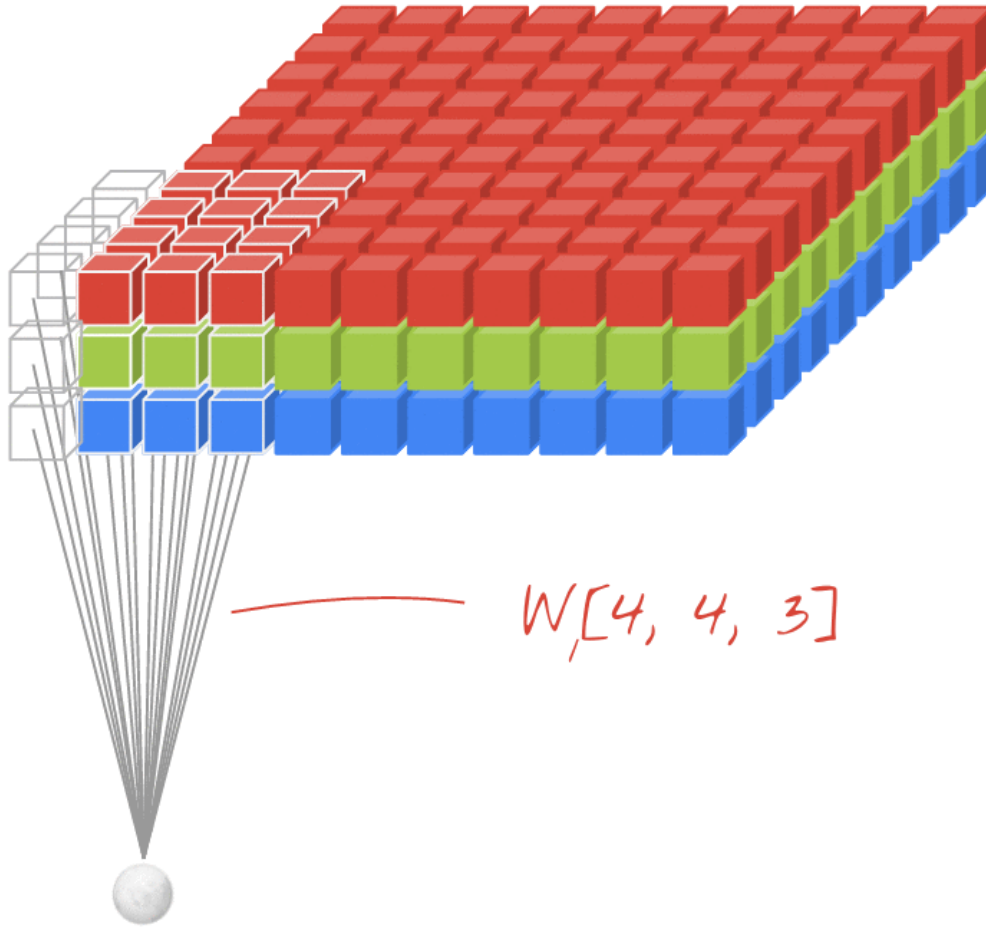
## Convolution layer with a 1-channel input and 6 kernels



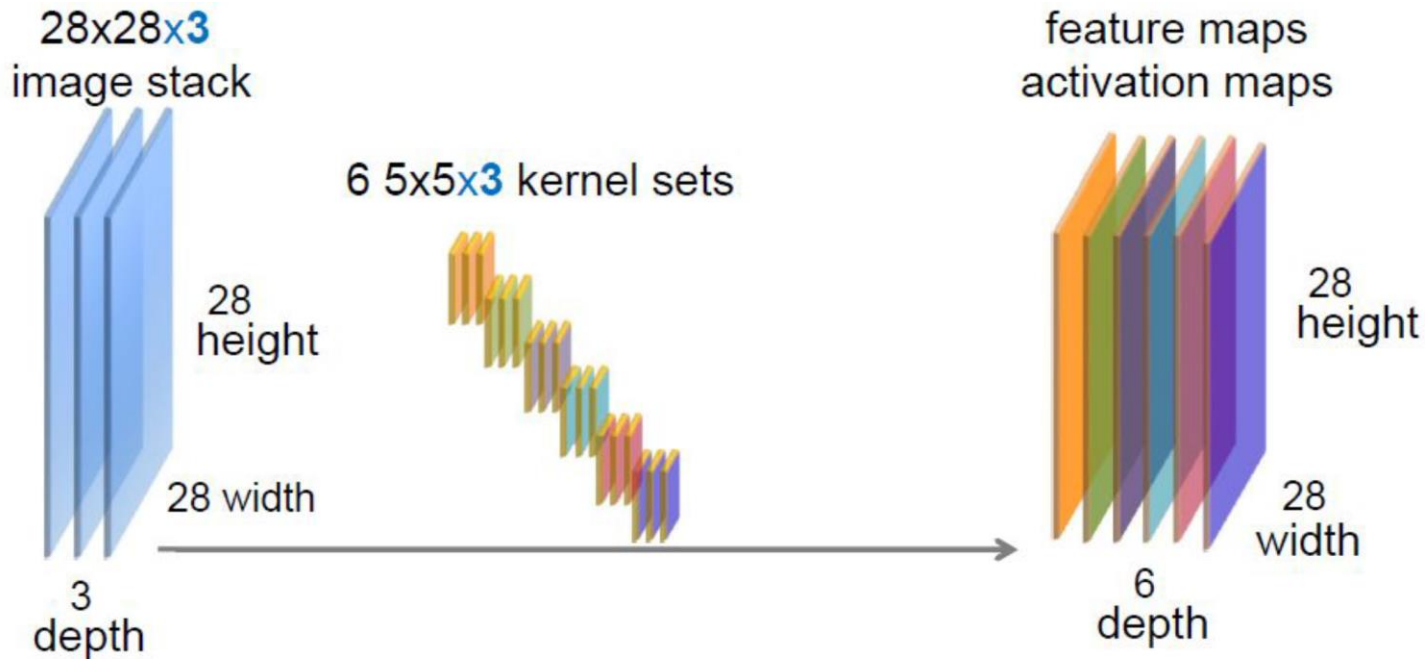
Convolution of the input image with 6 different kernels results in 6 activation maps.  
If the input image has only one channel, then each kernel has also only one channel.

# Animated convolution with 3 input channels

3 color channel input image

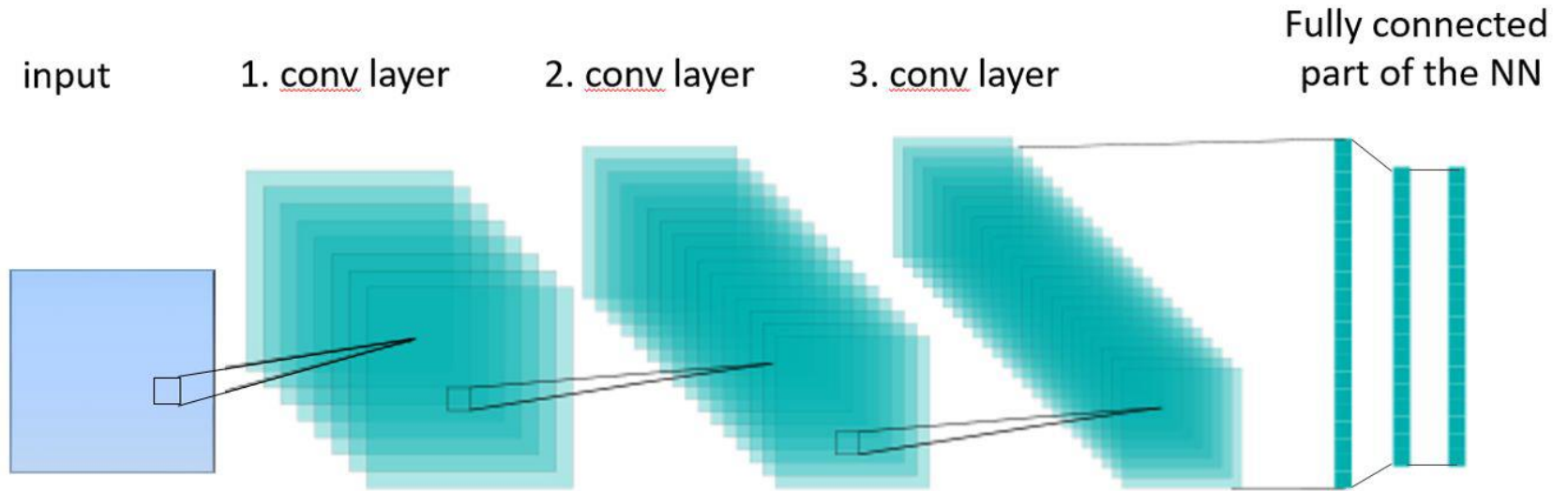


## Convolution layer with a 3-channel input and 6 kernels

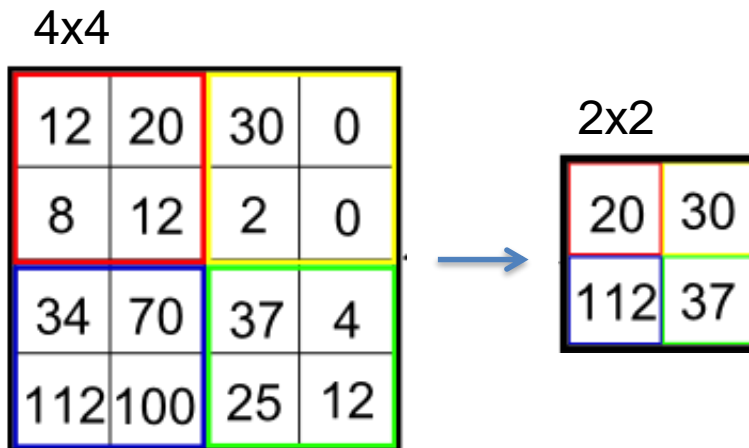
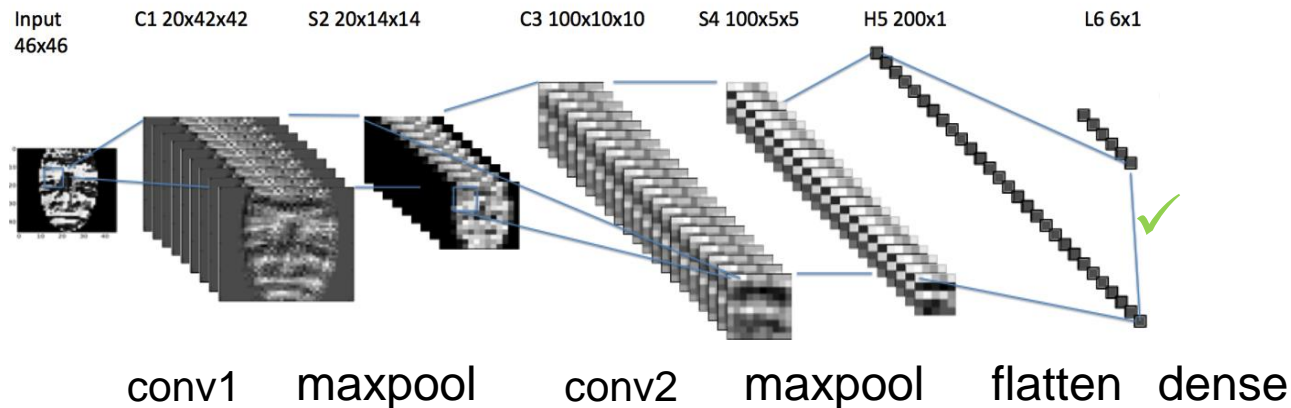


Convolution of the input image with 6 different kernels results in 6 activation maps.  
If the input image has 3 channels, then each filter has also 3 channels.

# A CNN with 3 convolution layers



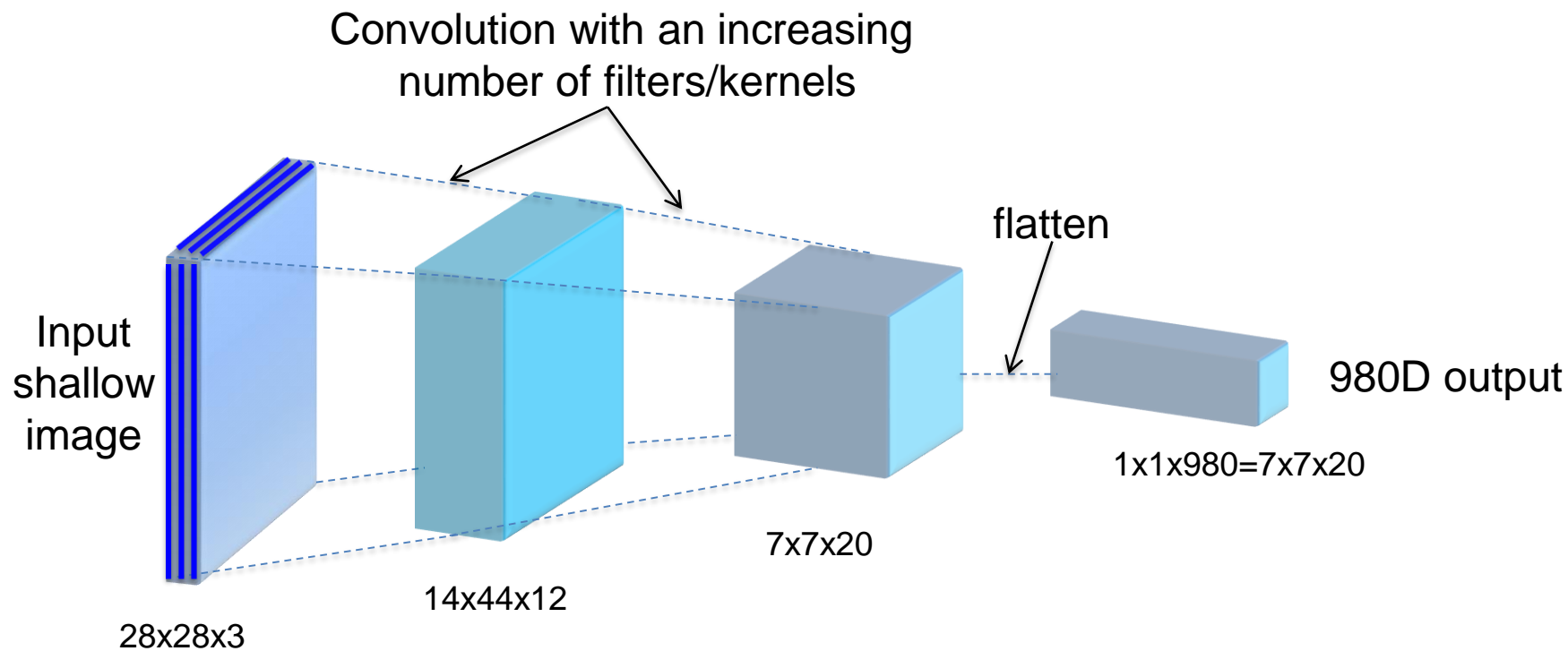
# CNN ingredient II: Maxpooling Building Blocks reduce size



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

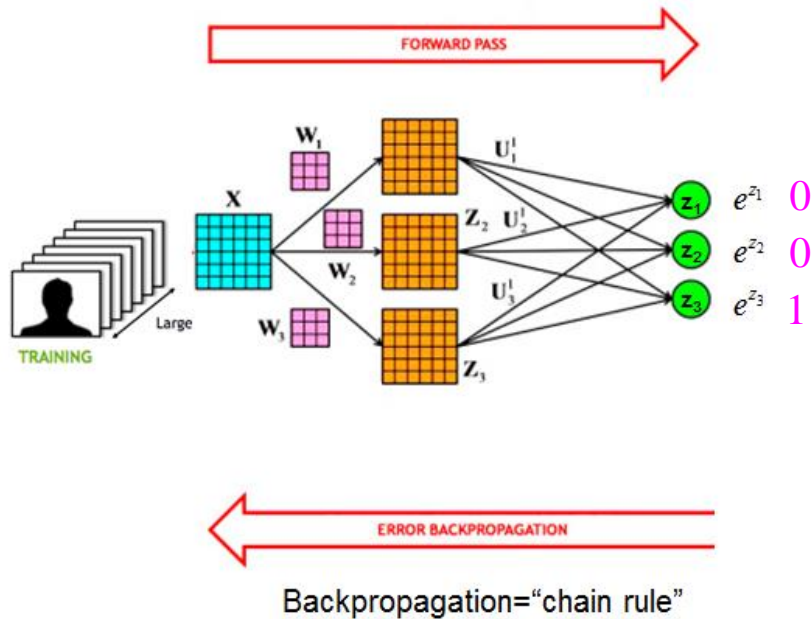
Hinton: „The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster“

# 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



For the training we need the **true 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:

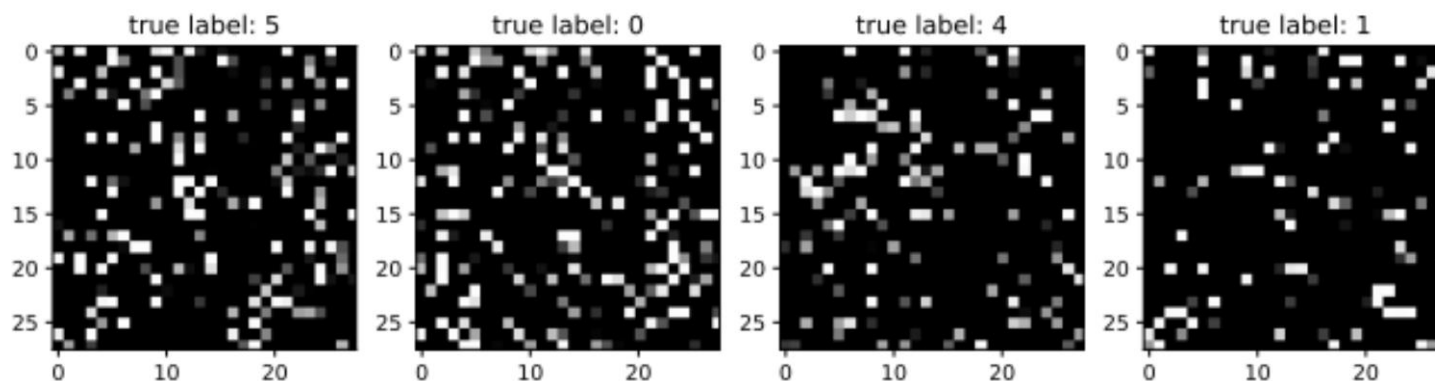
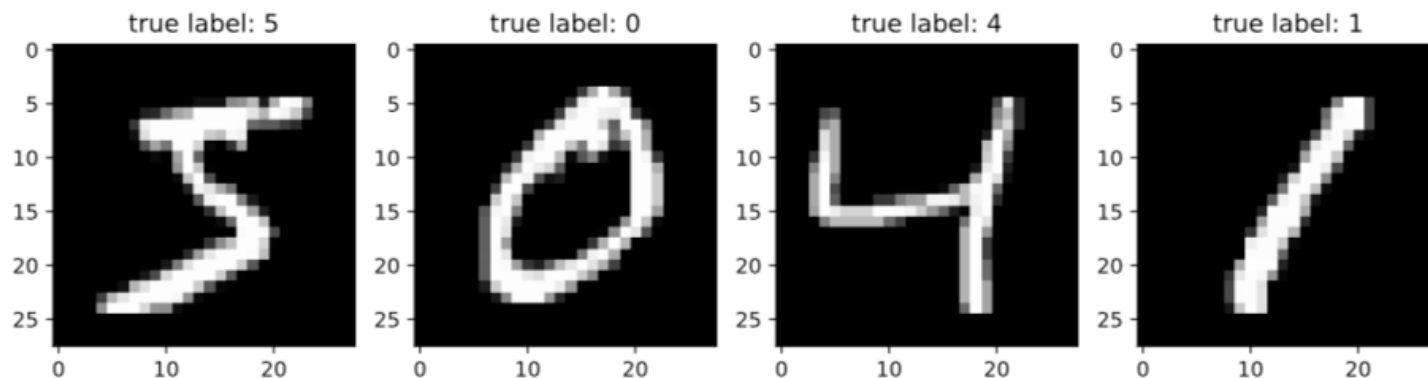
**L=distance(truth, output(w))**

Learning is done by weight updating:

$$w_i^{(t)} = w_i^{(t-1)} - \underset{\substack{\uparrow \\ \text{learning rate}}}{l^{(t)}} \frac{\partial L(w)}{\partial w_i} \bigg|_{w_i = w_i^{(t-1)}}$$



# Exercise: Does shuffling disturb a CNN?



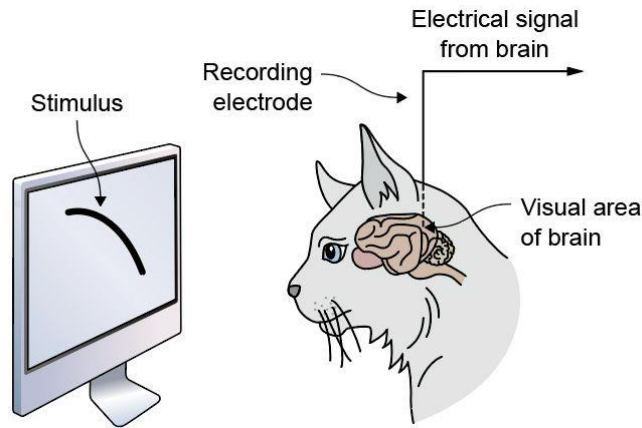
Open NB in:

# fcNN versus CNNs – some aspects

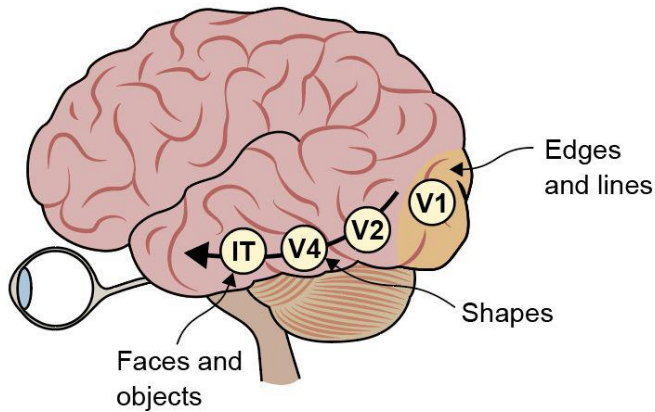
- A fcNN is good for tabular data, CNNs are good for ordered data (eg images).
- In a fcNN the order of the input does not matter, in CNN shuffling matters.
- A fcNN has no model bias, a CNN has the model bias that neighborhood matters.
- A node in one layer of a fcNN corresponds to one feature map in 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

# Biological Inspiration of CNNs

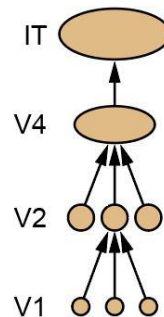
# How does the brain respond to visually recieved stimuli?



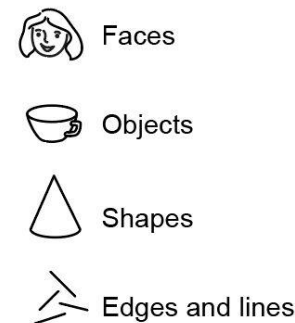
Setup of the experiment of Hubel and Wiesel in late 1950s in which they discovered **neurons** in the visual cortex that **responded** when moving **edges** were shown to the cat.



Receptive fields size

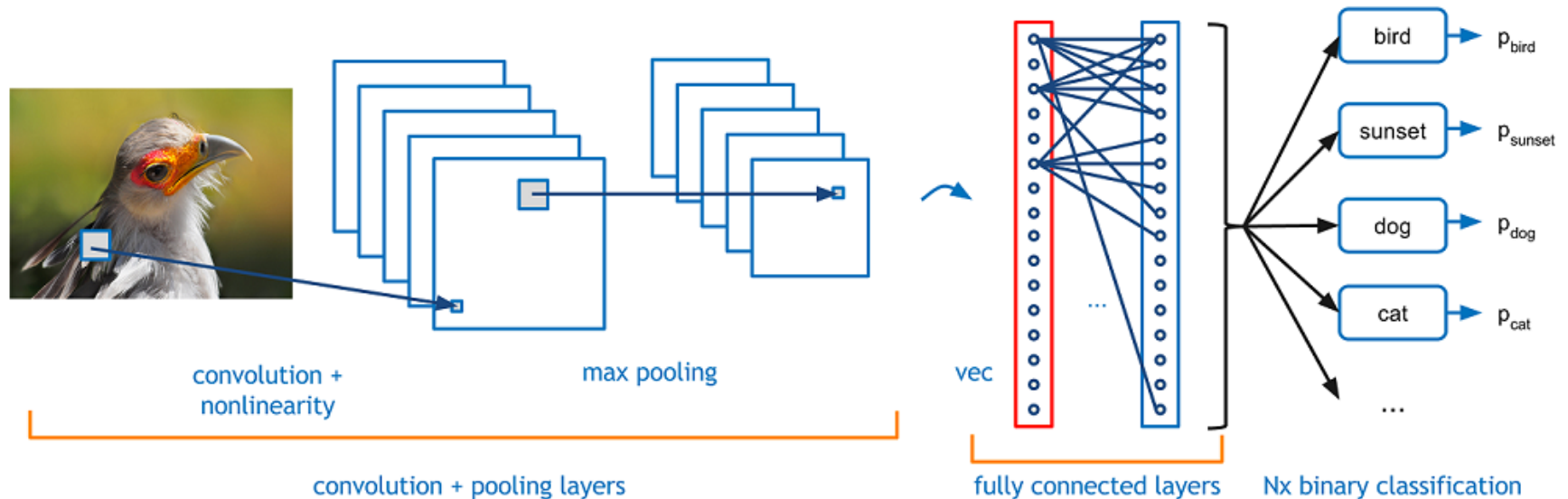


Features



Organization of the visual cortex in a brain, where neurons in different regions respond to more and more complex stimuli

# Convolutional part in CNN as representation learning



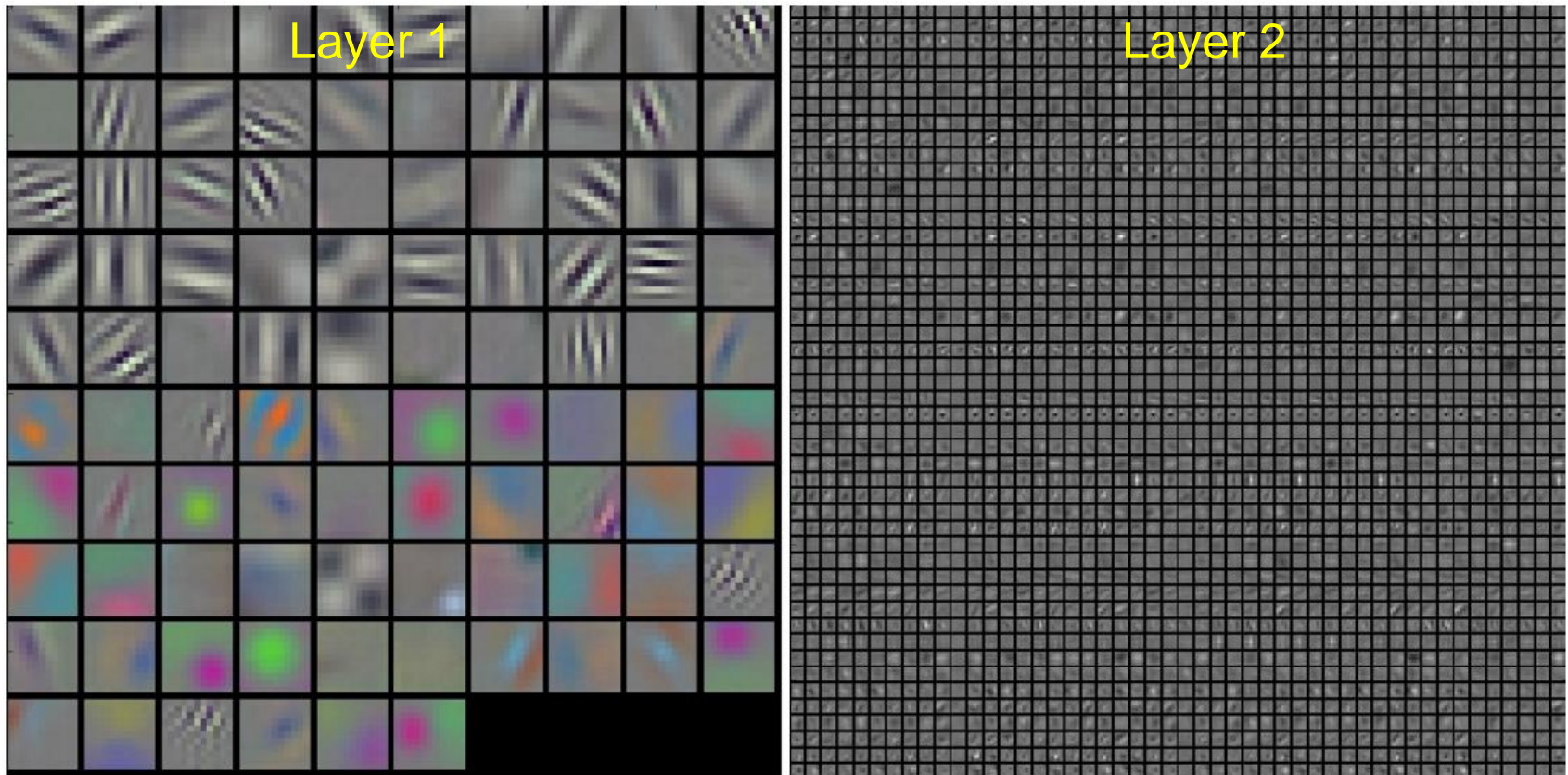
In a classical CNN we start with convolution layers and end with fc layers.

The task of the convolutional layers is to extract useful features from the image which might be appear at arbitrary positions in the image.

The task of the fc layer is to use these extracted features for classification.

# Visualize the weights used in filters

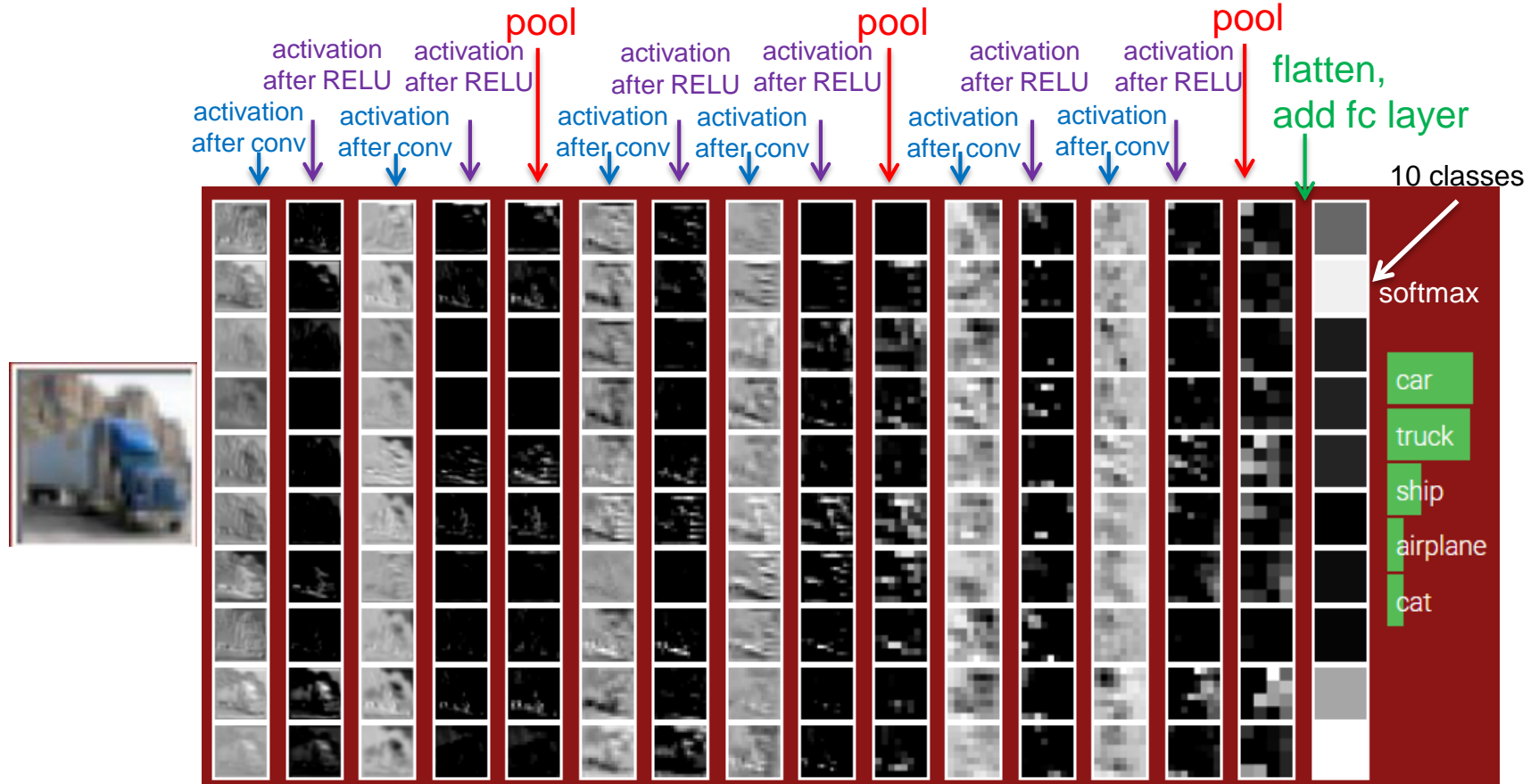
Filter weights from a trained Alex Net



Only in layer 1 the filter pattern correspond to extracted patterns in the image.

In higher layers we can only check if patterns look noisy, which would indicate that the network that hasn't been trained for long enough, or possibly with a too low regularization strength that may have led to overfitting.

# Activations in output-close layers can be used as new features



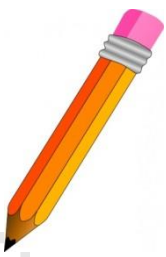
Activation maps give insight on the spatial positions where the filter pattern was found in the input **one layer below** (in higher layers activation maps have no easy interpretation) -> only the activation maps in the first hidden layer correspond directly to features of the input image.



# Check out a simple live CNN & fill the gaps

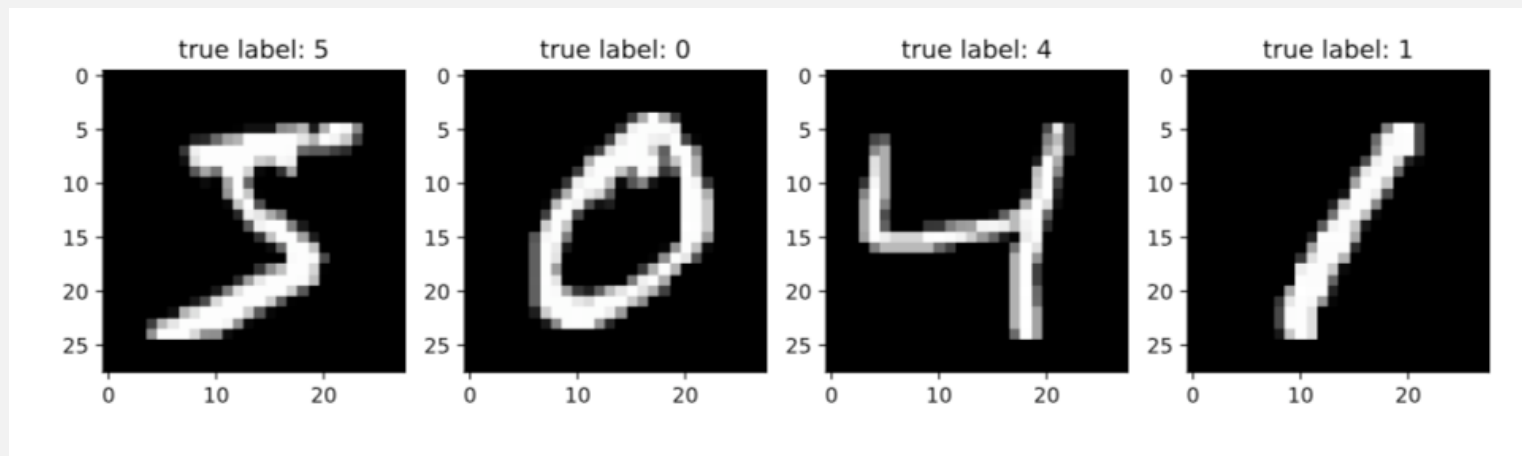
Follow the first link of day 5 on our course-webpage ["live cnn in browser"](https://transcranial.github.io/keras-js/#/mnist-cnn)  
<https://transcranial.github.io/keras-js/#/mnist-cnn>

```
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'))
```





# Homework: Tune the MNIST model



Investigate the impact of standardizing the input data in fcNN and CNN.

Compare your best fcNN and best CNN with respect to

- Accuracy
- Overfitting
- Number of weights