Hochschule Konstanz
Fakultät Informatik

Oliver Dürr

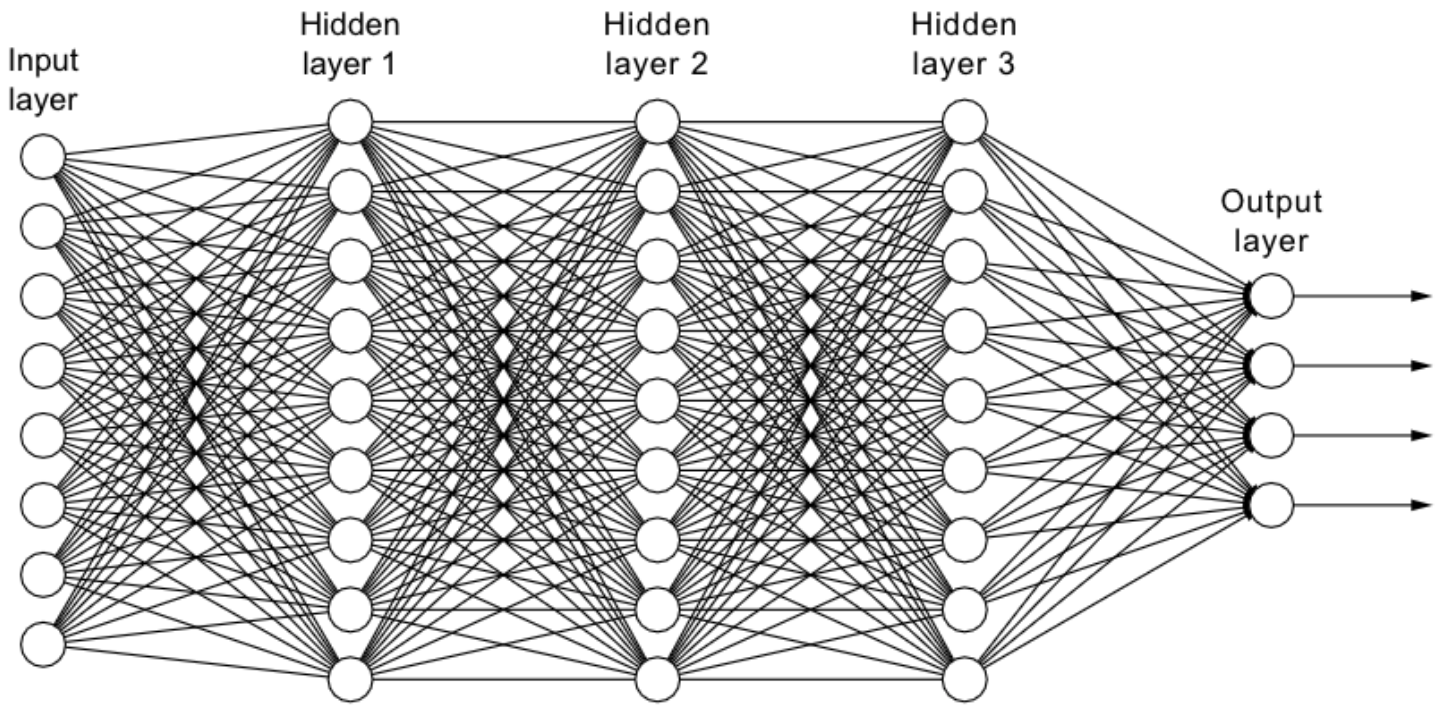
Short course on deep learning
Block 2

This course is a short version of dl_course_2022 I created with Beate Sick.

Topics of today

- 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
  - Biological inspiration of CNNs
  - Building CNNs

Architecture of a fully connected NN

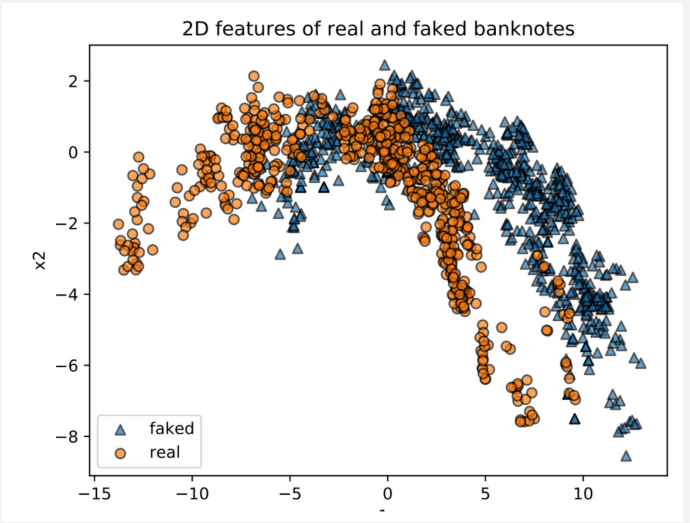


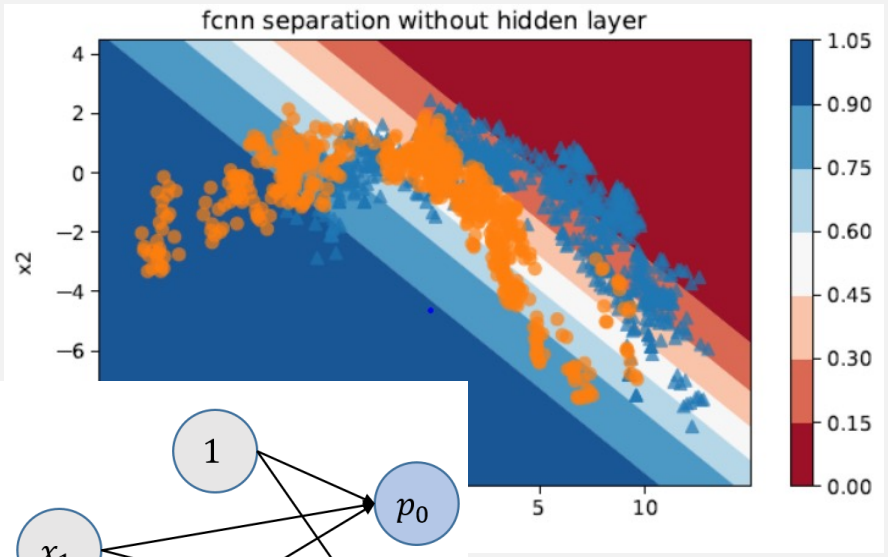
Each neuron in a fcNN gets as input a weighted sum of all neuron activation from one layer below. Different neurons in the same layer have different weights in this weighted sum, which are learned during training.

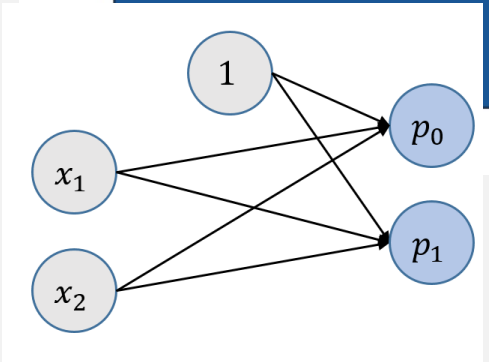
Discussion of Homework

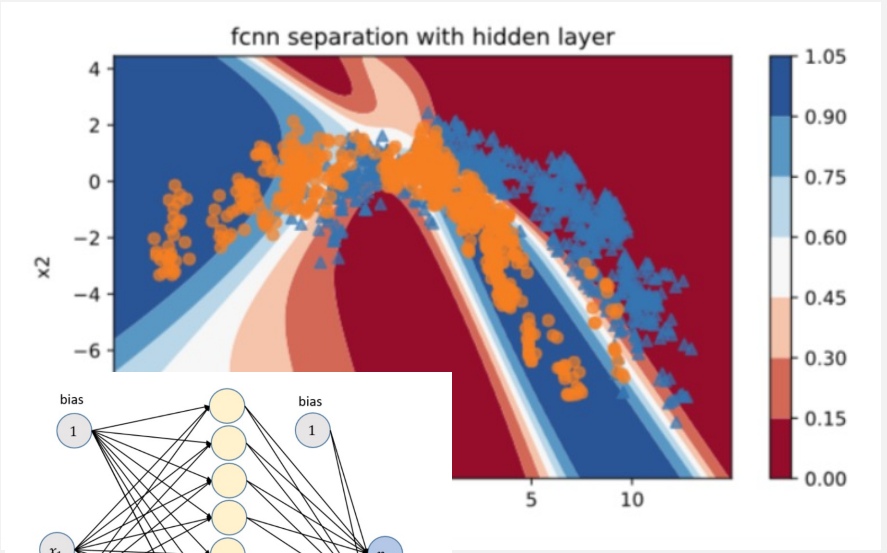
NB 02: Classify banknotesbased on 2 features (x1,x2)

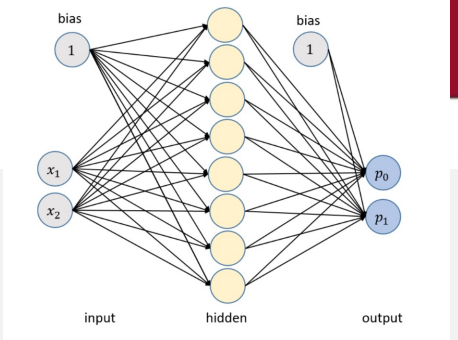
https://github.com/tensorchiefs/dl_course_2022/blob/master/notebooks/02_fcnn_with_banknote.ipynb



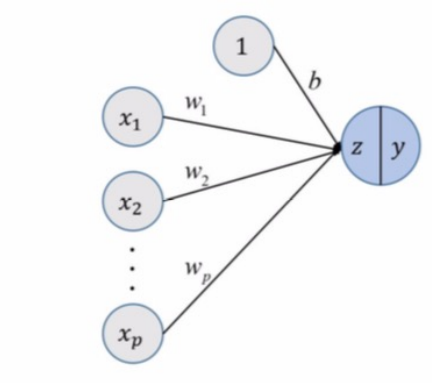




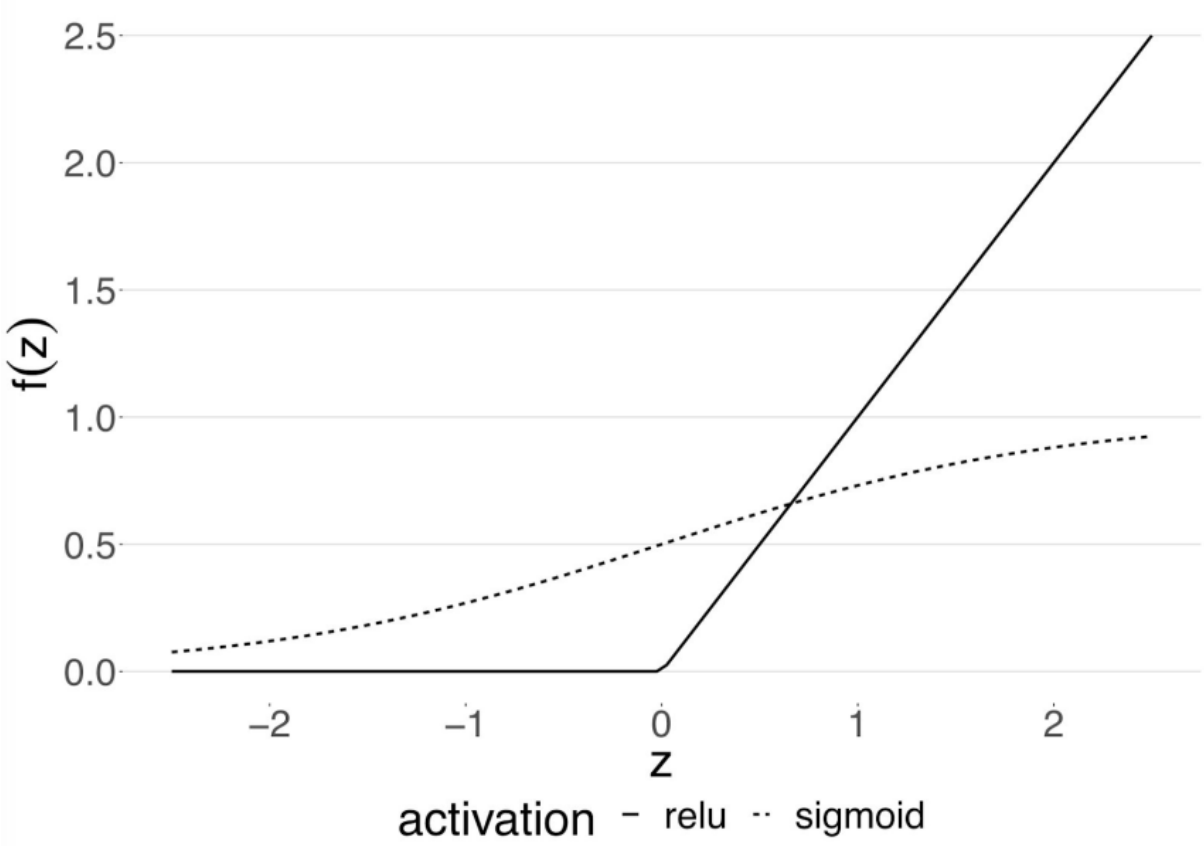




Common non-linear activation function

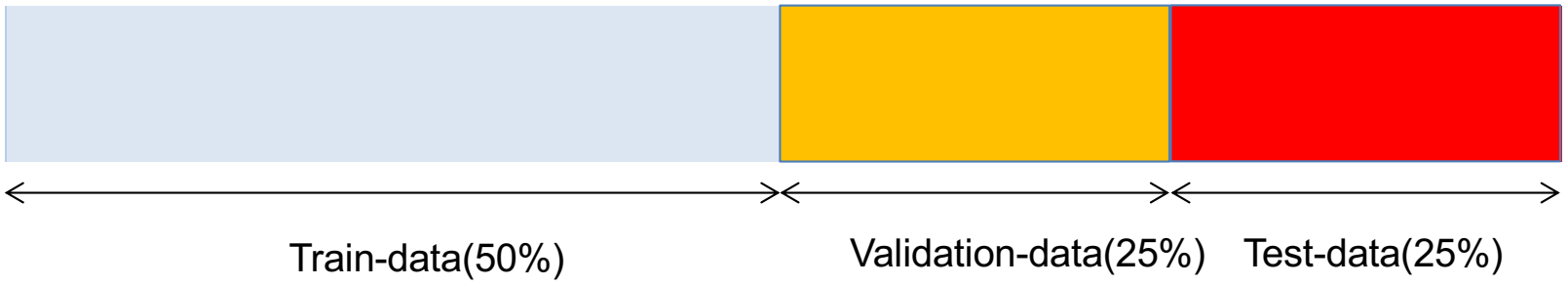


$$y = f(z) = f(b + \sum x_i \cdot w_i)$$



ReLU leads to faster training,.

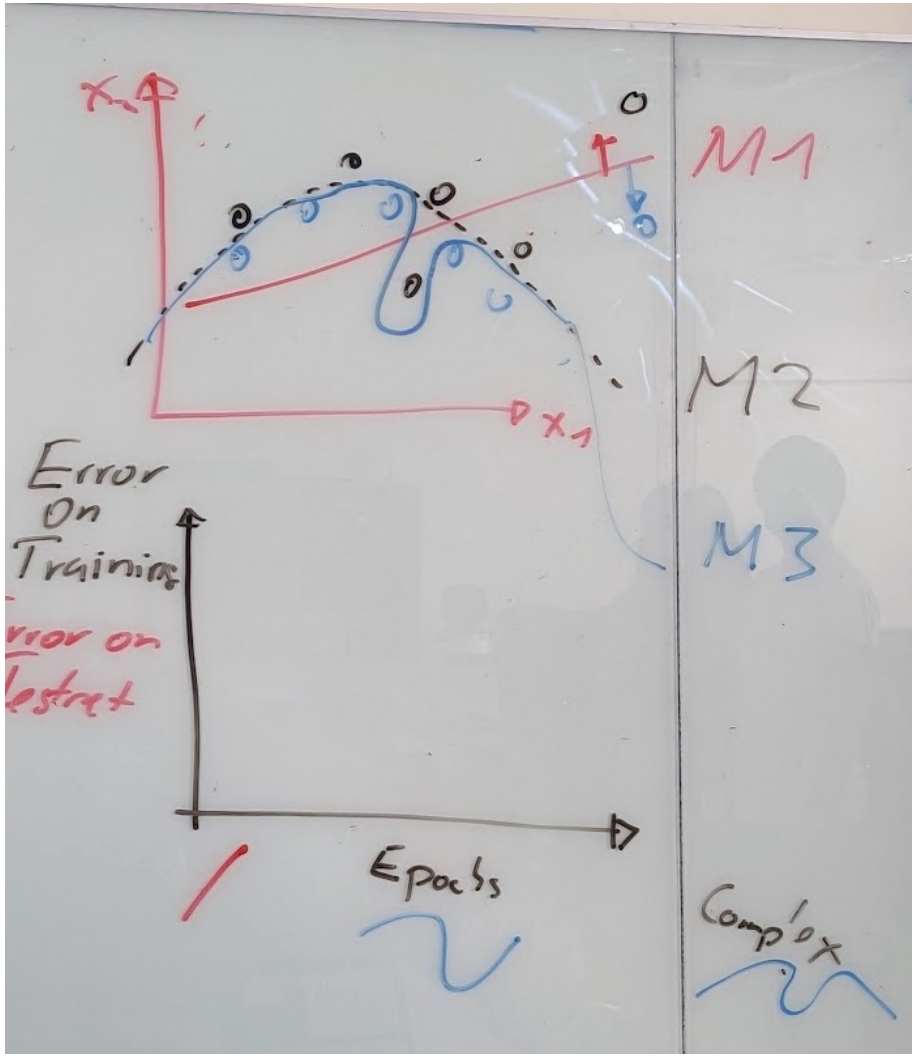
Best practice: Split in Train, Validation, and Test Set



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**.

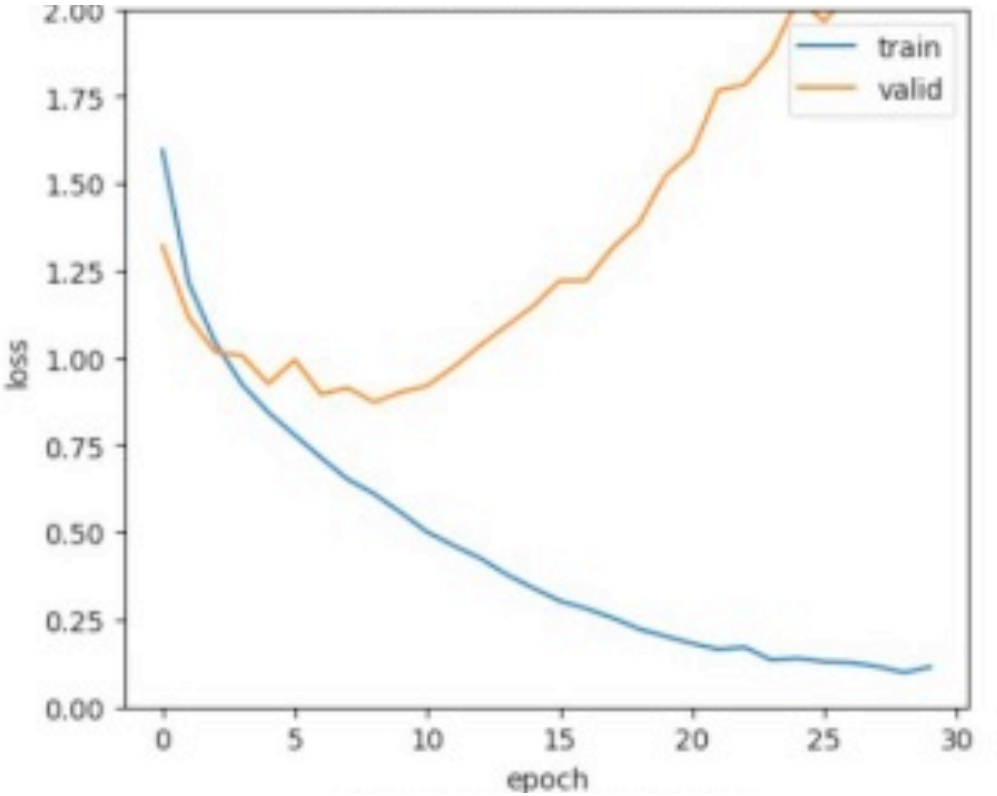
Determine performance metrics, such as MSE, to evaluate the predictions **on new validation or test data**

Overfitting Blackboard



What can loss curves tell us?

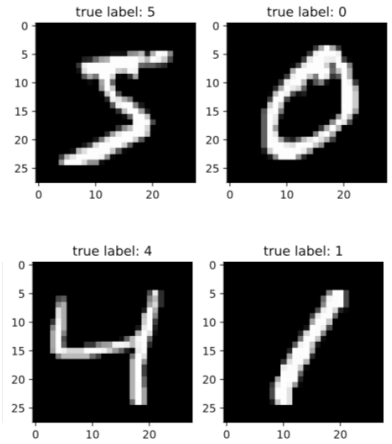
Very common check: Plot loss in train and validation data vs epoch of training.



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

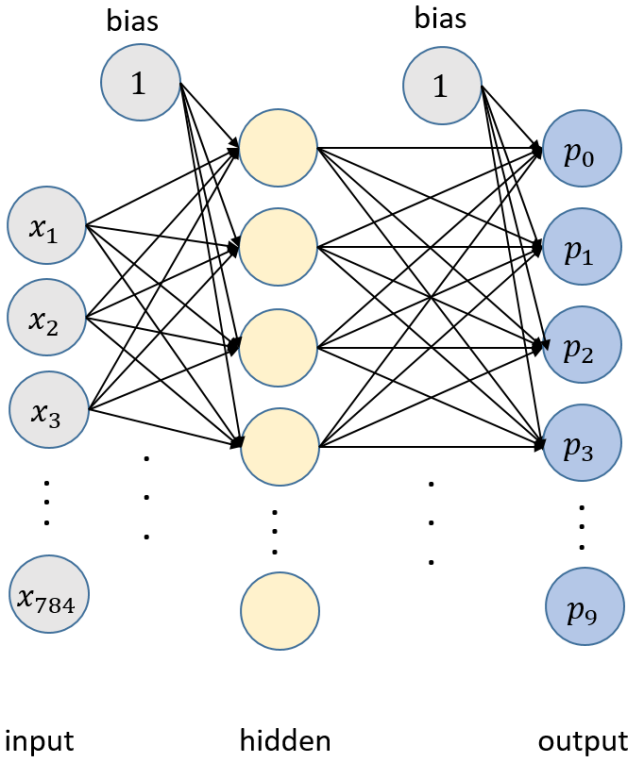
Analysis of Image data

A fcNN for MNIST data



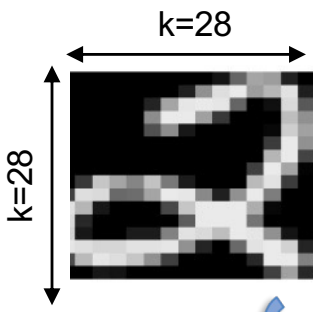
The first four digits of the MNIST data set - each image consisting of 28x28=784 pixels

A fully connected NN with 1 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 (probabilities)

What is going on in a (no hidden layer)



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

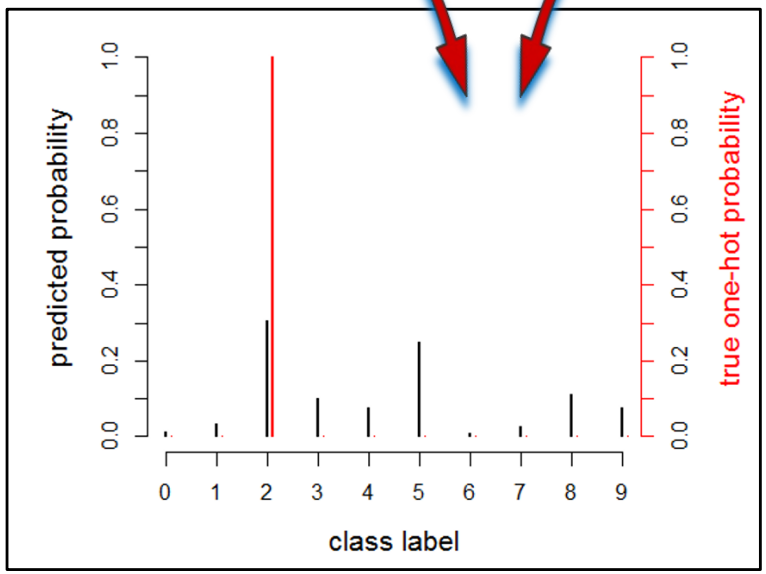
$$\hat{p}_k = \frac{e^{z_k}}{\sum_j e^{zj}}$$

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

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



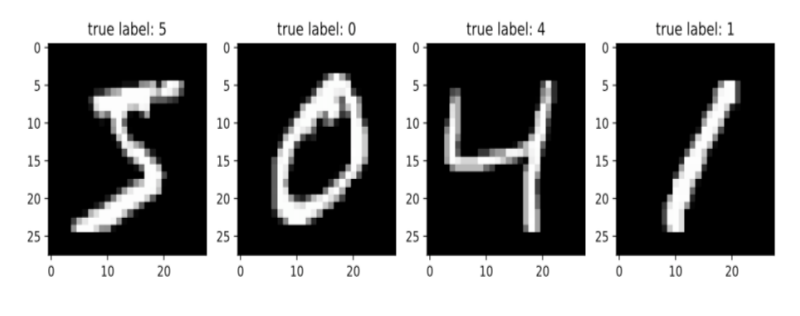
Cost C or Loss = cross-entropy averaged over all images in mini-batch

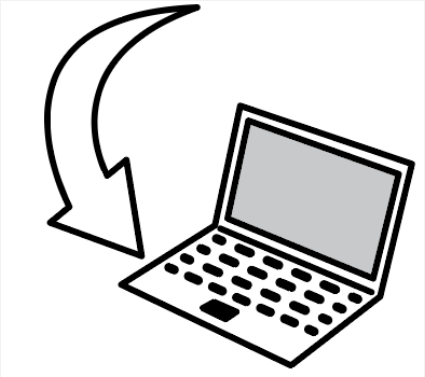
$$C = \frac{1}{N}\sum_i D(\mathbf{p}_i, \mathbf{y}_i)$$

Cross-Entropy

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

Exercise: FCNN [1 hour]





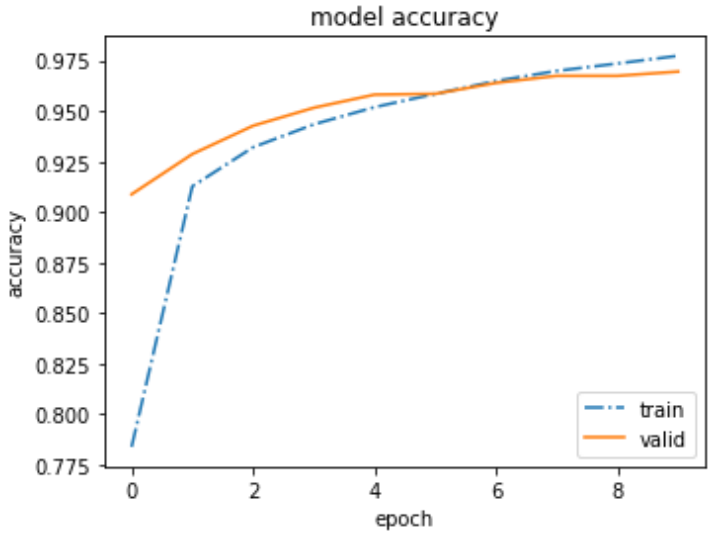
	Accuracy on Testset	Loss on Testset
M1 (no hidden)		
M2 (2 hidden)		
M3 (2 hidden ReLU)		

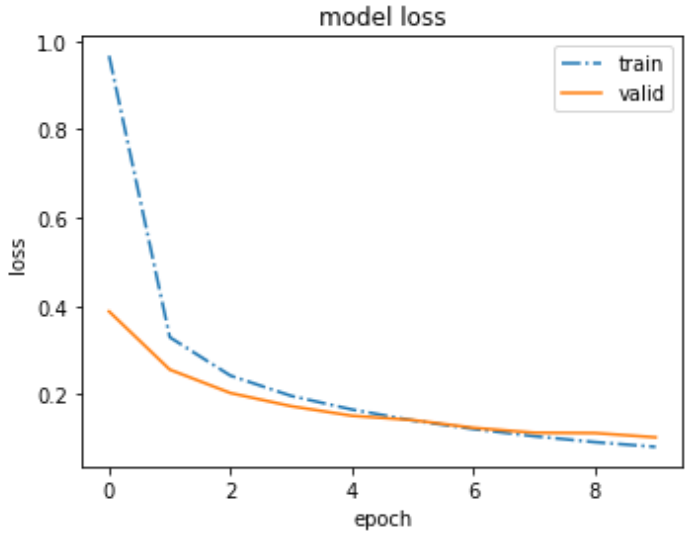
Exercise: fcNN for MNIST **03_fcnn_mnist**:
Learning Objective:

- Effect of ReLU
- Overfitting and Loss-Curves
- Comparing of Models

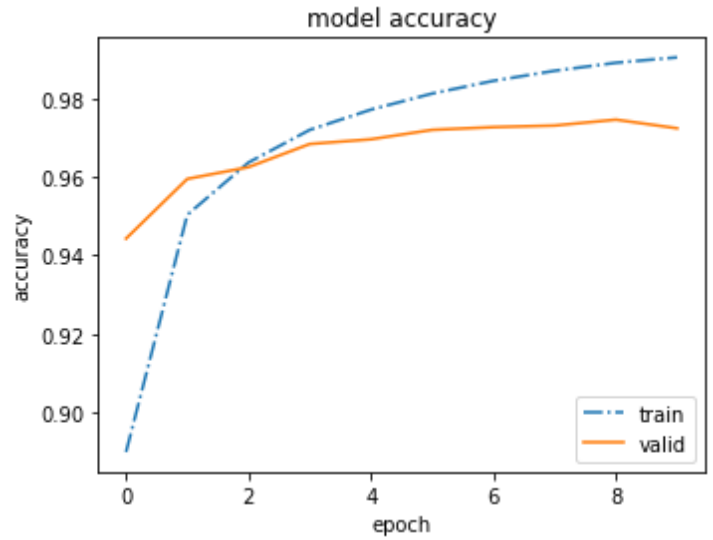
Result 03_fcNN with ReLU

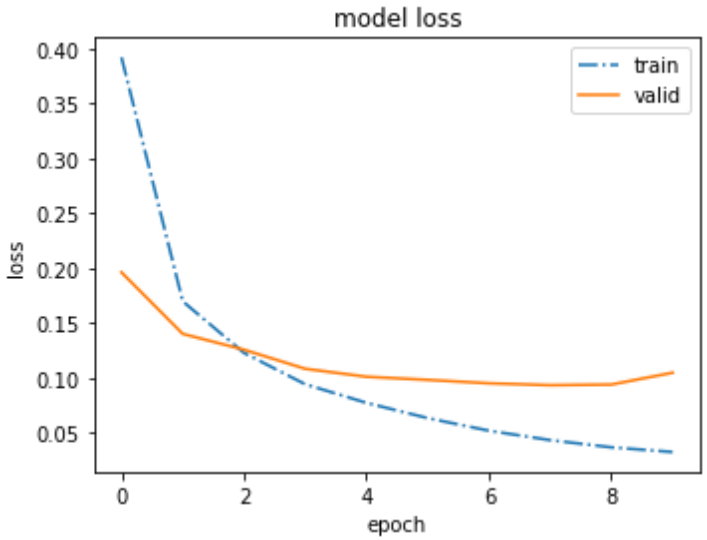
Sigmoid Acc = 0.9687





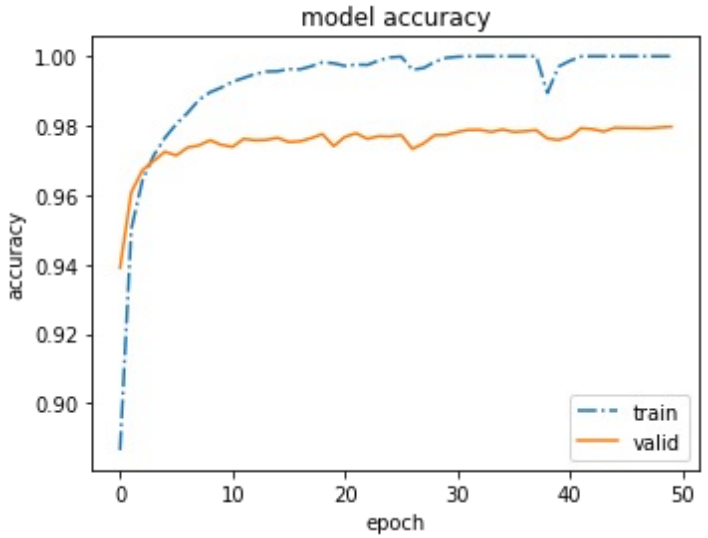
Relu / Acc = 0.9735

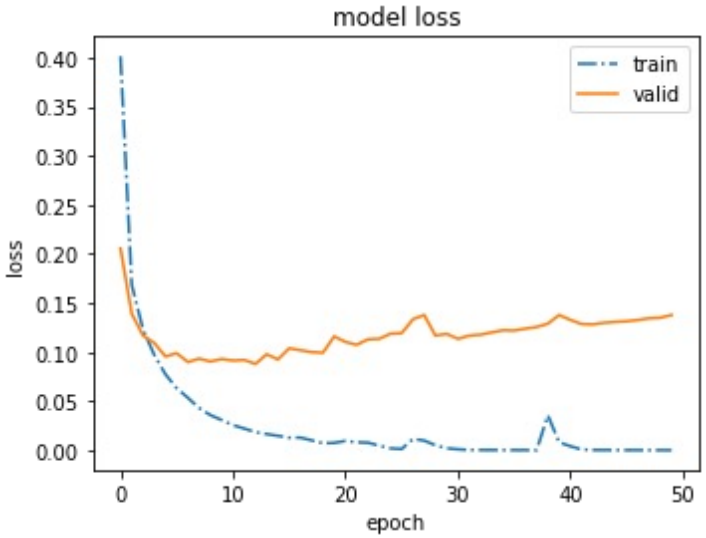




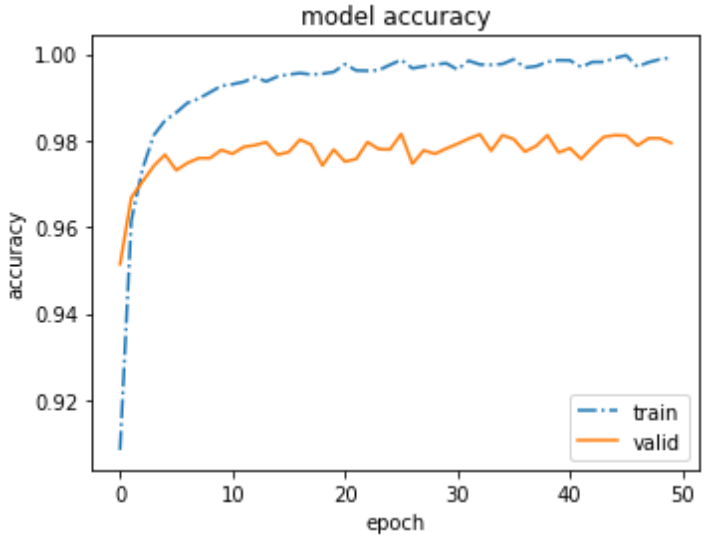
Result 03_fcNN with ReLU longer training

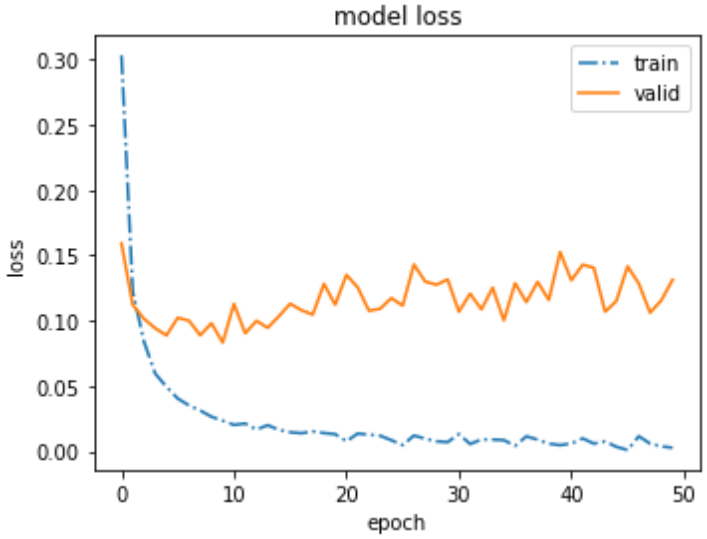
Sigmoid Acc = 0.9789





Relu / Acc = 0.9799





Model comparison

- Model 1 (two hidden Sigmoid)
  - Acc (test-set): 0.9239
- Model 2 (two hidden Sigmoid)
  - Acc (test-set): 0.9687
- Model 3 (two hidden ReLU)
  - Acc (test-set): 0.9735
- Model 4 (two hidden ReLU, long training)
  - Acc (test-set): 0.9799

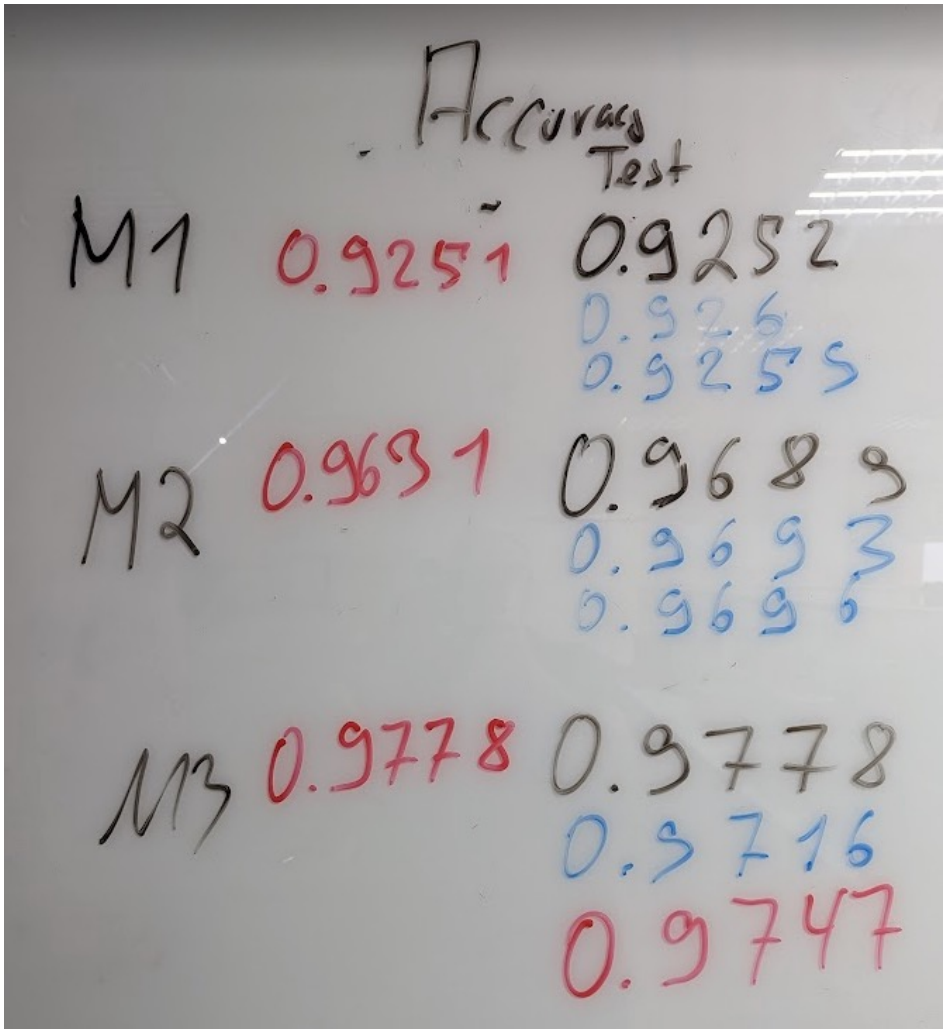
Which model is the best model?

Statistical considerations

It's save to say that model 2 is better than model 1. M1<M2

M3 better than M2, prob. but not 100% sure. Better to compare several runs.

Results from the students



Report mean and stddev

Typical Paper

METHODS	ACCURACY
SVM	95.50%
DBN	97.20%
CNN + GAUSSIAN	97.70%
CNN + GABOR	98.30%
CNN + GAUSSIAN + DROPOUT	98.64%
CNN + GABOR + DROPOUT	**98.78%**

Only reported the mean value.
Better include std.

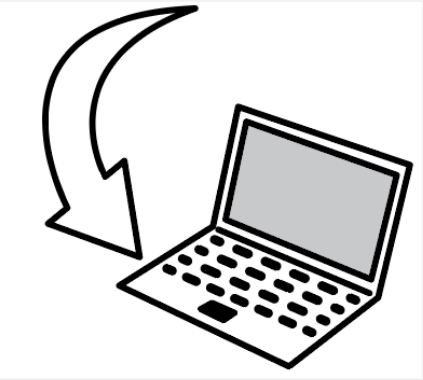
Other messures

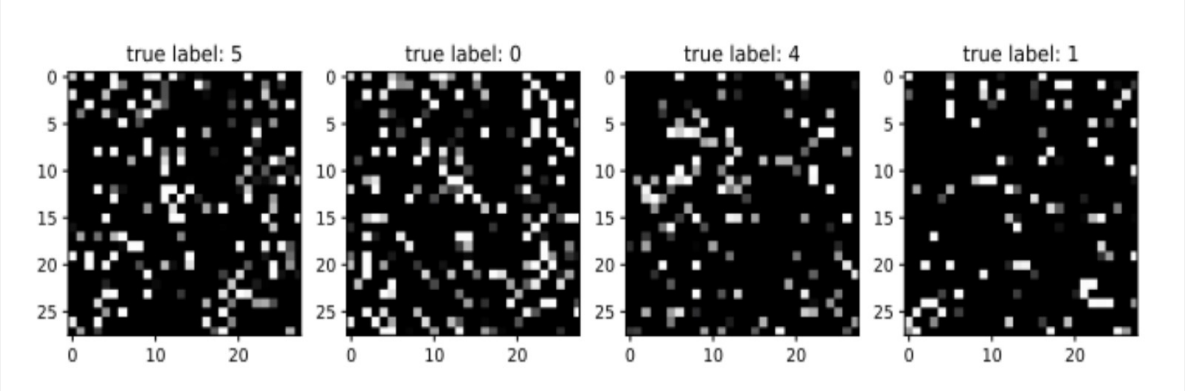
```
pred=model.predict(X_test_flat)
print(confusion_matrix(np.argmax(Y_test,axis=1),np.argmax(pred,axis=1)))
acc_fc = np.sum(np.argmax(Y_test,axis=1)==np.argmax(pred,axis=1))/len(pred)
print("Acc = " , acc_fc)
```

```
[[ 967    0    2    2    0    3    2    1    1    2]
 [   0 1121    4    0    0    1    4    1    4    0]
 [   6    1  998    3    4    0    5    7    7    1]
 [   1    0    6  981    0    3    0    7    9    3]
 [   2    0    1    0  955    0    5    1    2   16]
 [   6    1    0   17    1  844    8    2    7    6]
 [   8    3    0    1    6    6  928    0    6    0]
 [   1    7   14    8    1    1    0  989    0    7]
 [   4    1    2    9    3    4    2    4  942    3]
 [   4    4    1    8   17    8    0    7    7  953]]
Acc =  0.9678
```

Besides Accuracy there exists tons of other measures (some can be derived from confusion matrix)

Exercise: FCNN and Shuffling

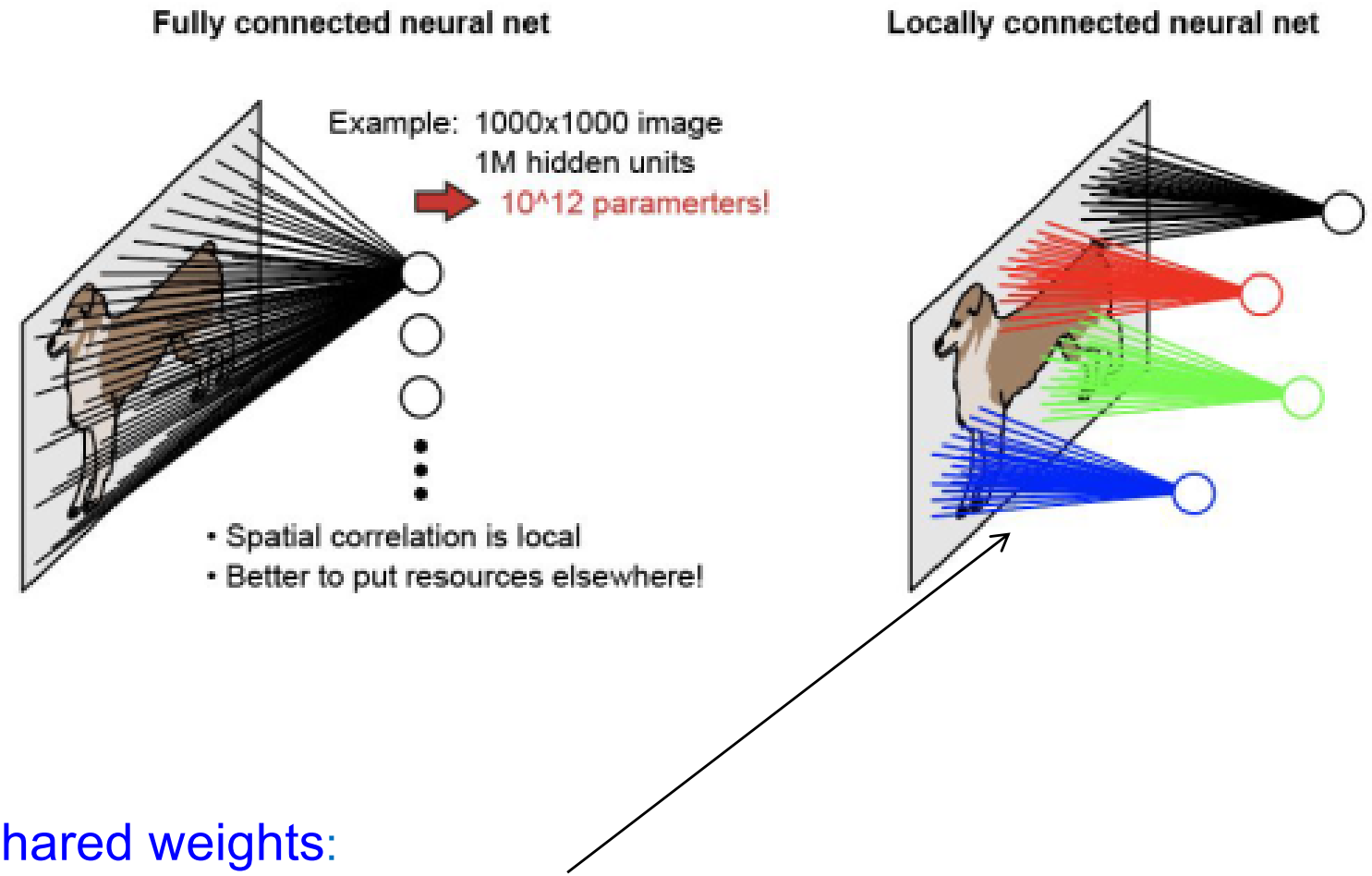




Exercise : fcNN for MNIST **04_fcnn_mnist**
Learning Objective:
Investigate if shuffling disturbs the performance fcNN for MNIST

Convolutional Neural Networks
For image data

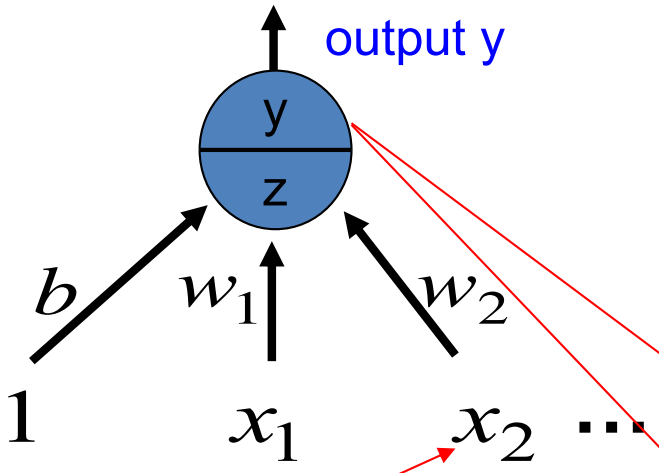
Convolution extracts local information using few weights



Shared weights:

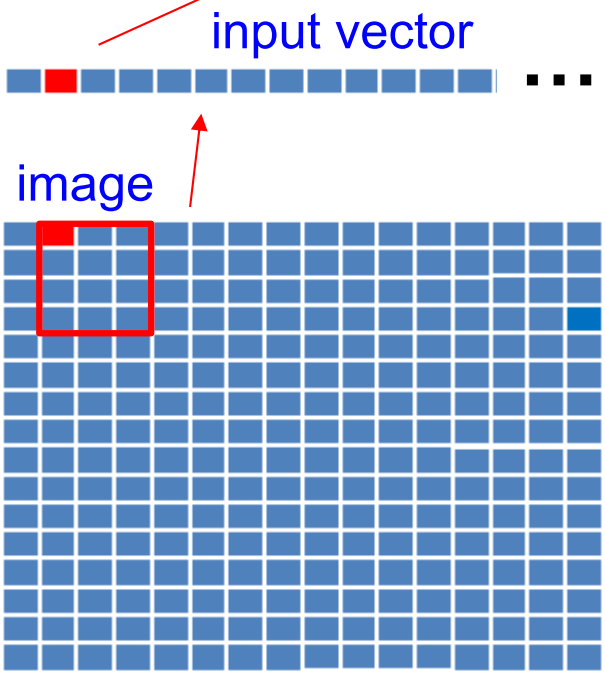
by using the same weights for each patch of the image we need much less parameters than in the fully connected NN and get from each patch the same kind of local feature information such as the presence of an edge.

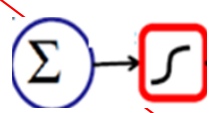
An artificial neuron

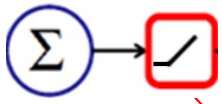


bias weights

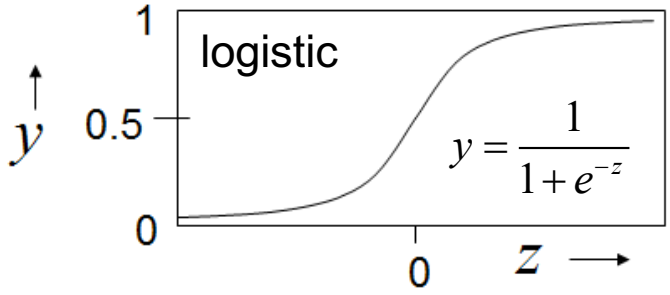
$$z = b + \sum_i x_i w_i$$

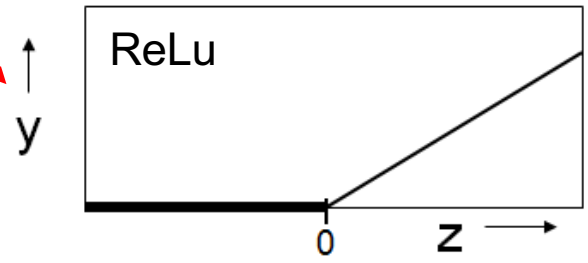




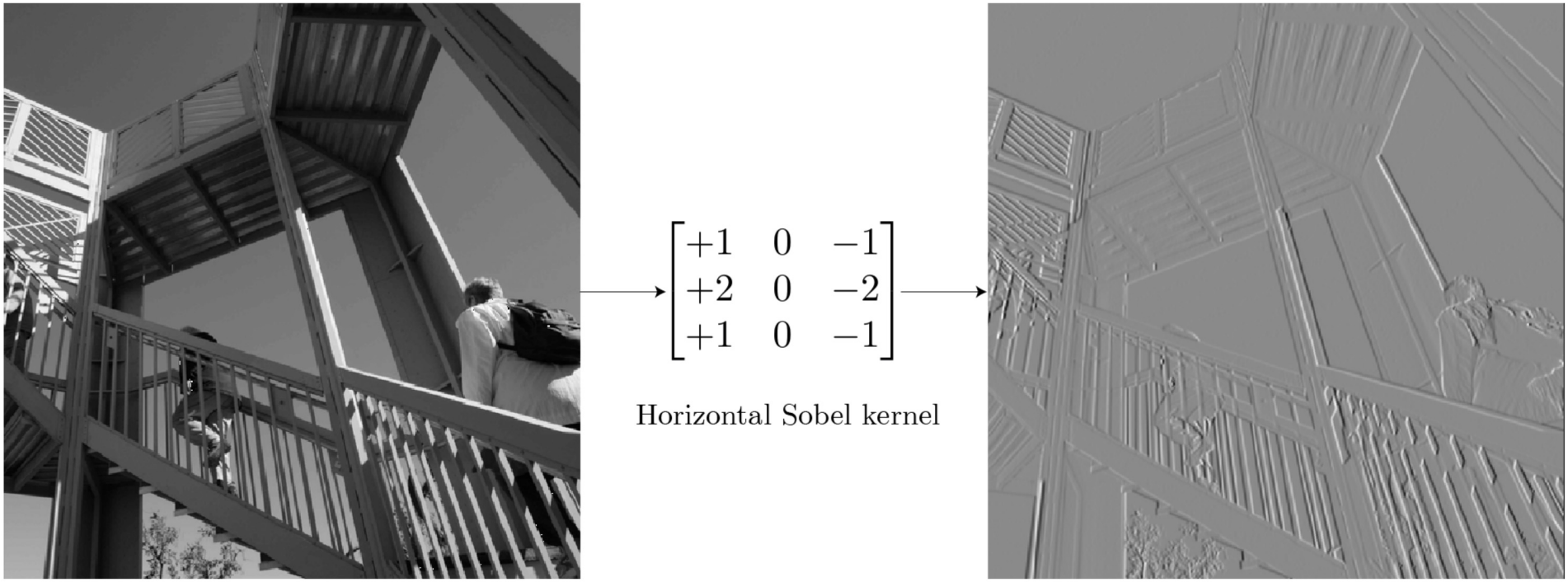


Different non-linear transformations are used to get from z to output y





Example of designed Kernel / Filter



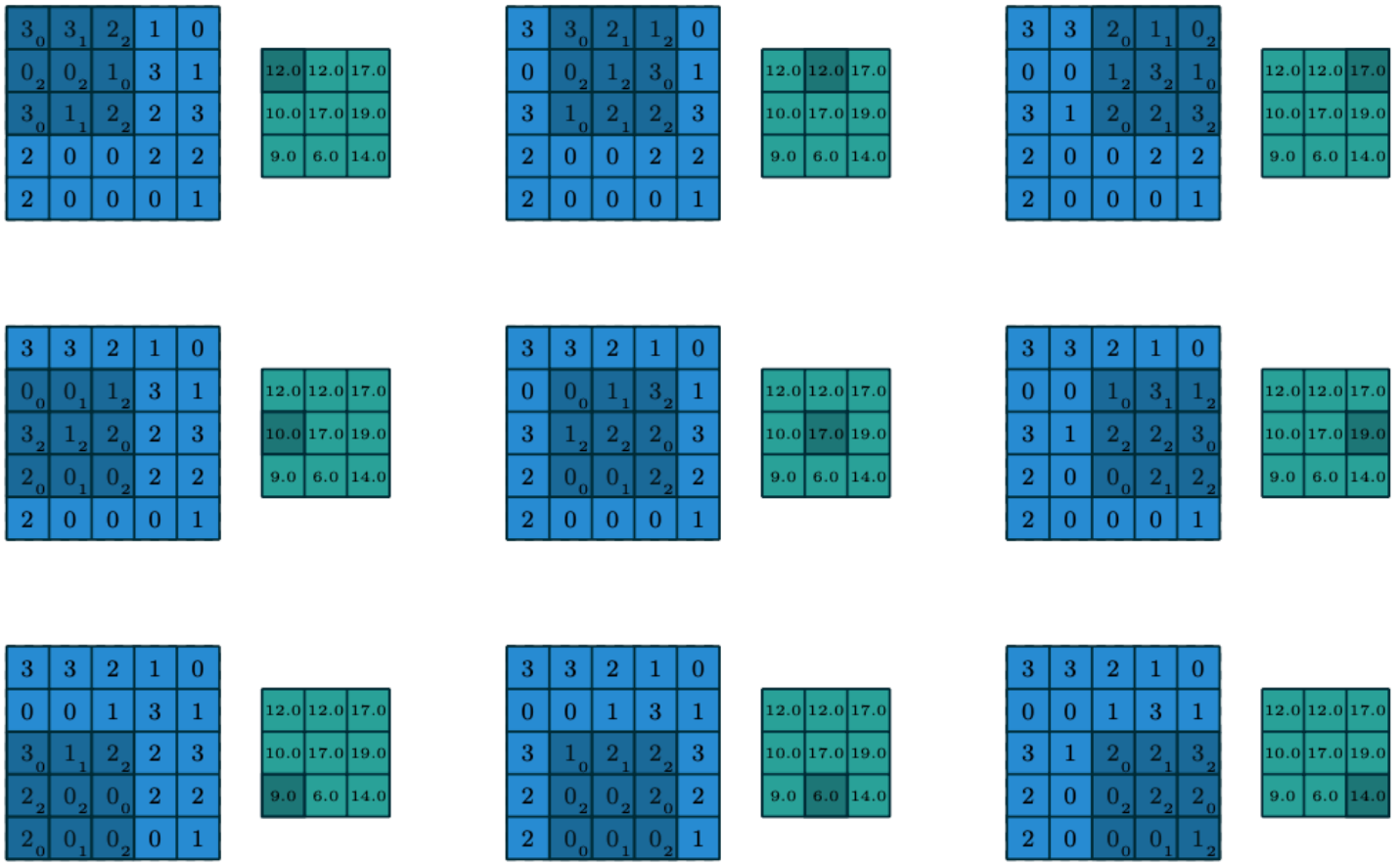
Applying a vertical edge detector kernel

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

The basic idea of convolution (blackboard)

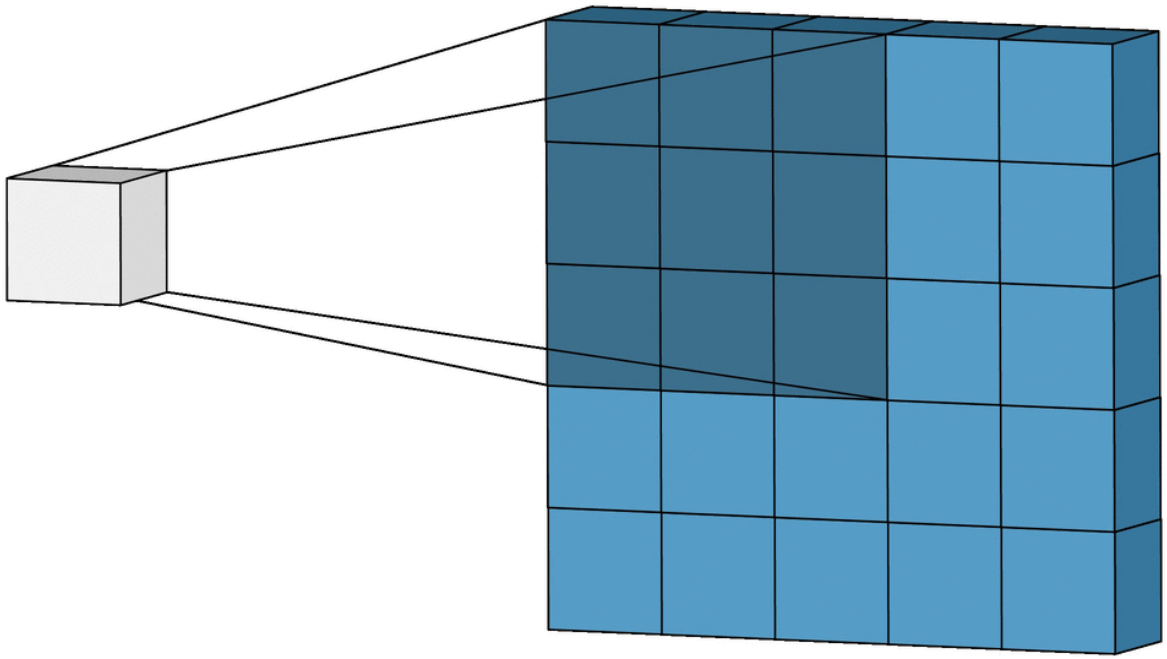
- One dimensional
  - 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1
- Kernel on a piece of paper and slide it true the image

2 Dimensional generalization



The value of the middle pixel is the sum weight times original values

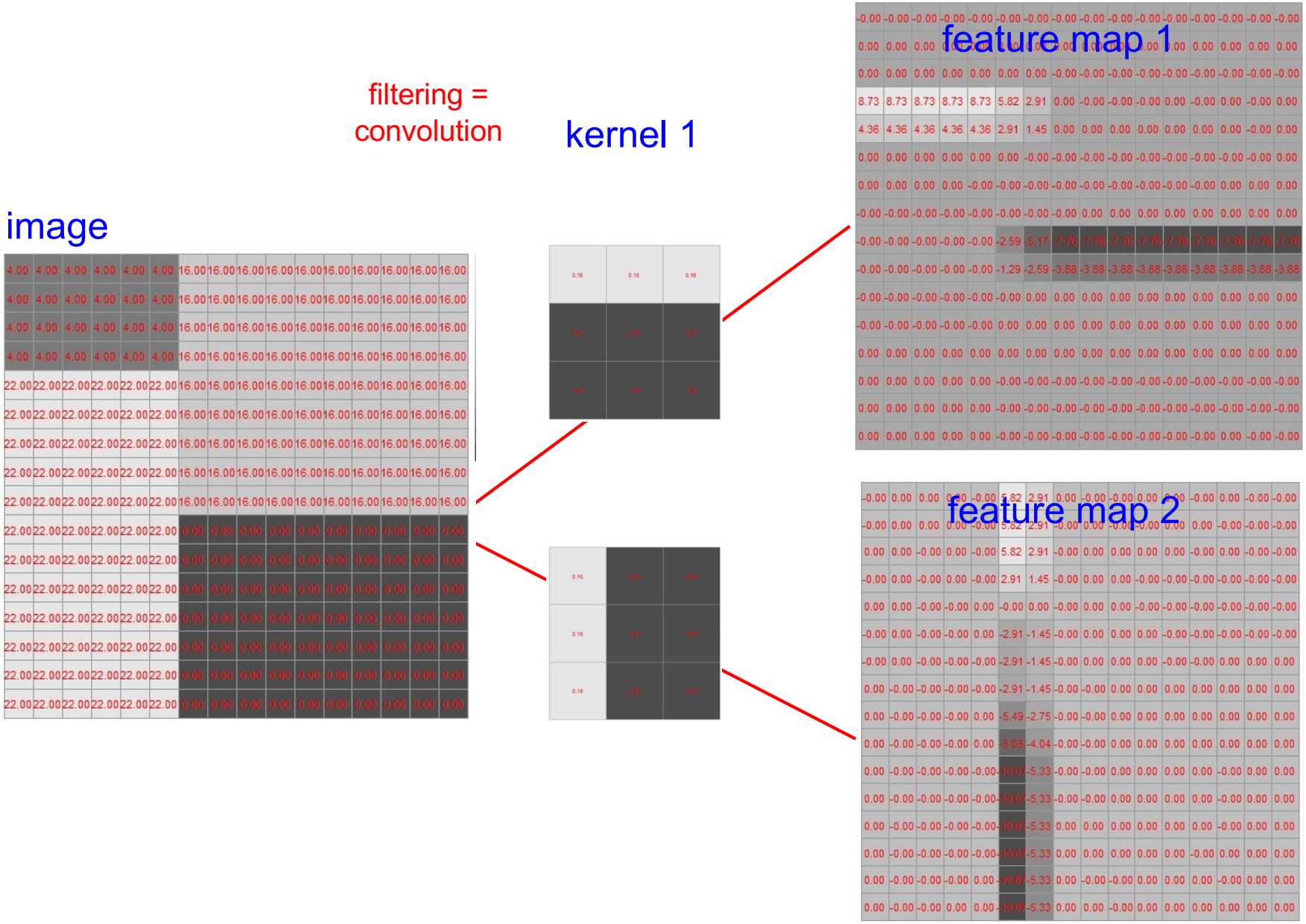
Applying the same 3x3 kernel at each image position



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.

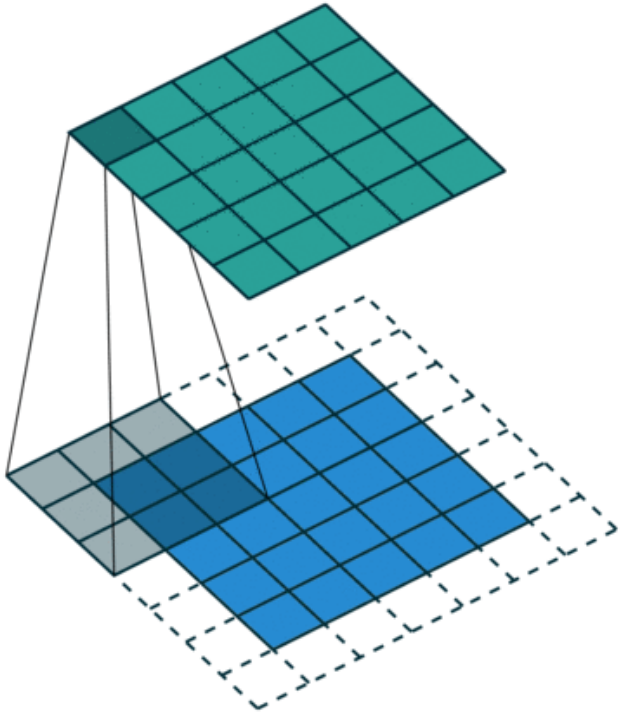
Exercise: Do one convolution step by hand

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

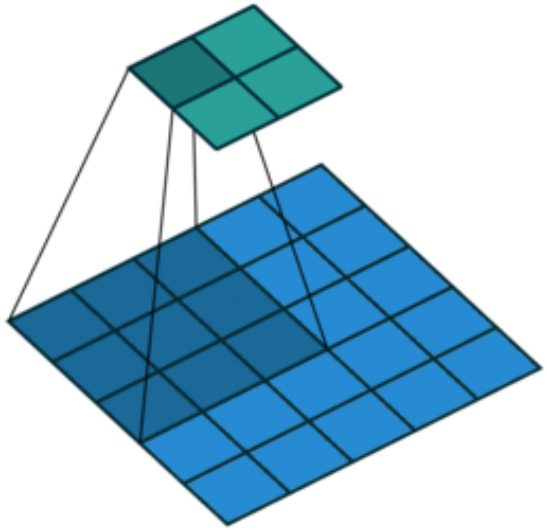
2. The small numbers in the shaded region are the kernel weights.
Determine the position and the value within the resulting activation map.

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

Padding



Zero-padding to achieve
same size of feature and input

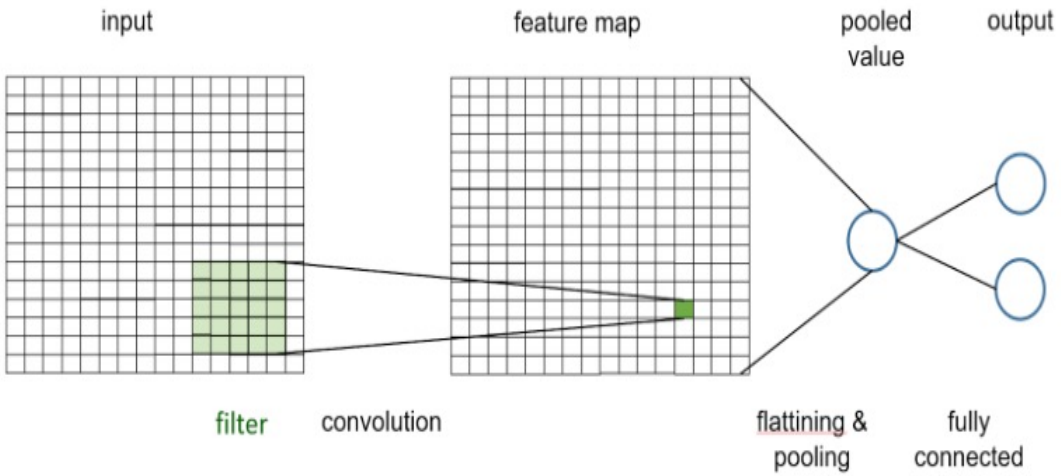


no padding to only use
valid input information

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

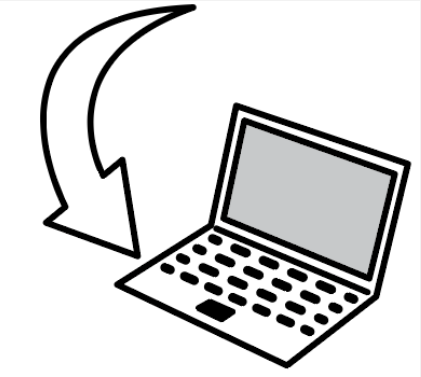
Building a very simple CNN with keras

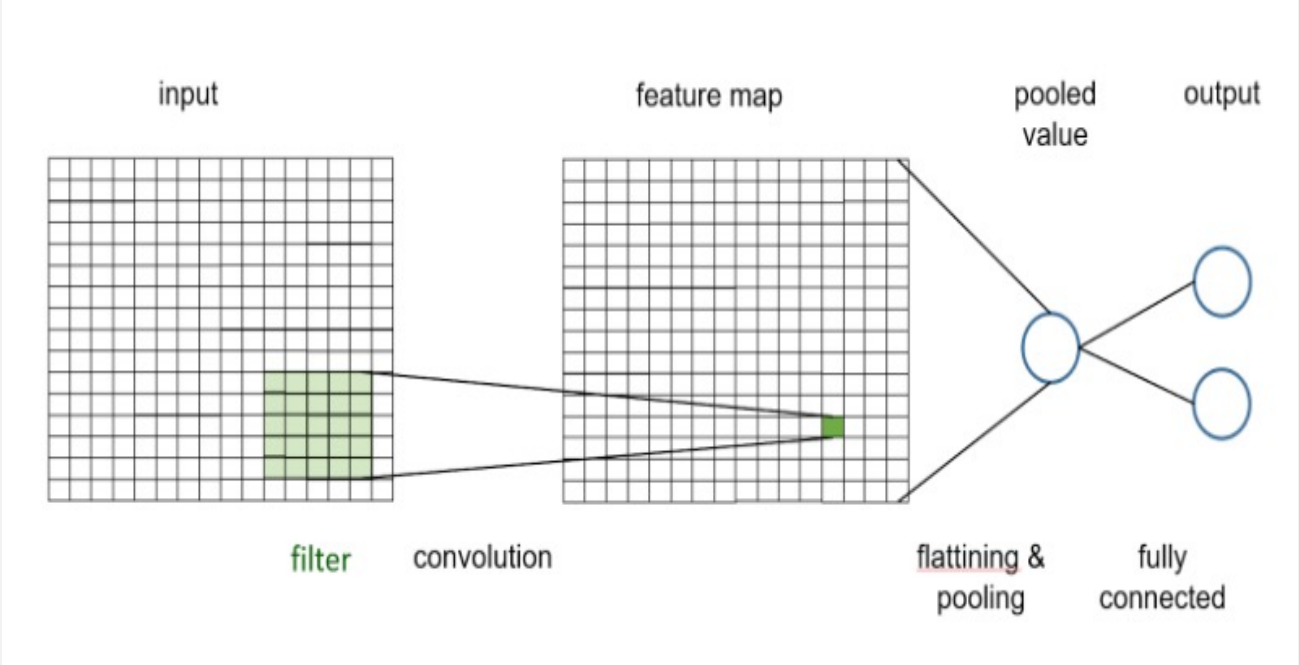



```
model = Sequential()
model.add(Convolution2D(1,(5,5), # one 5x5 kernel
        padding='same',          # zero-padding to preserve size
        input_shape=(pixel,pixel,1)))
model.add(Activation('linear'))
# take the max over all values in the activation map
model.add(MaxPooling2D(pool_size=(pixel,pixel)))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

Max-pooling simply take the largest value

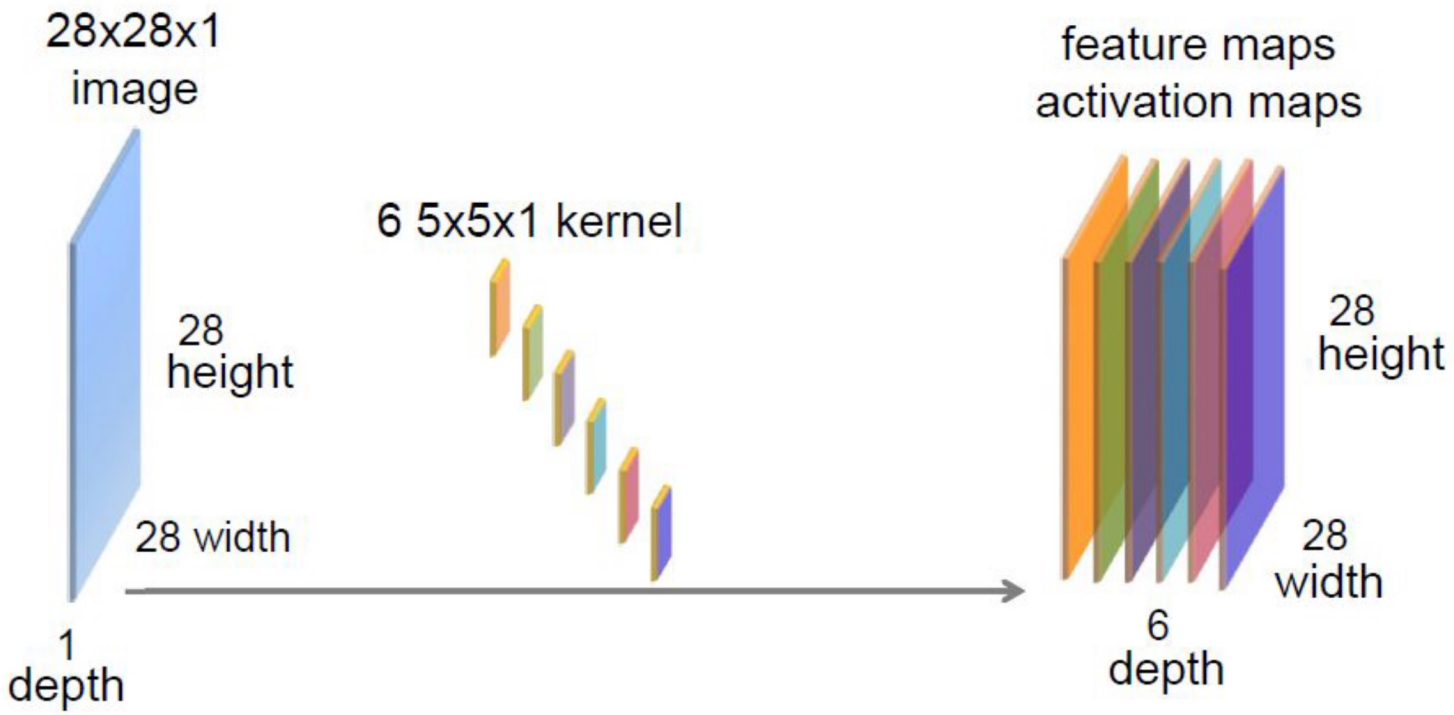
Exercise: Artstyle Lover





Open NB 05_cnn_edge_lover.ipynb

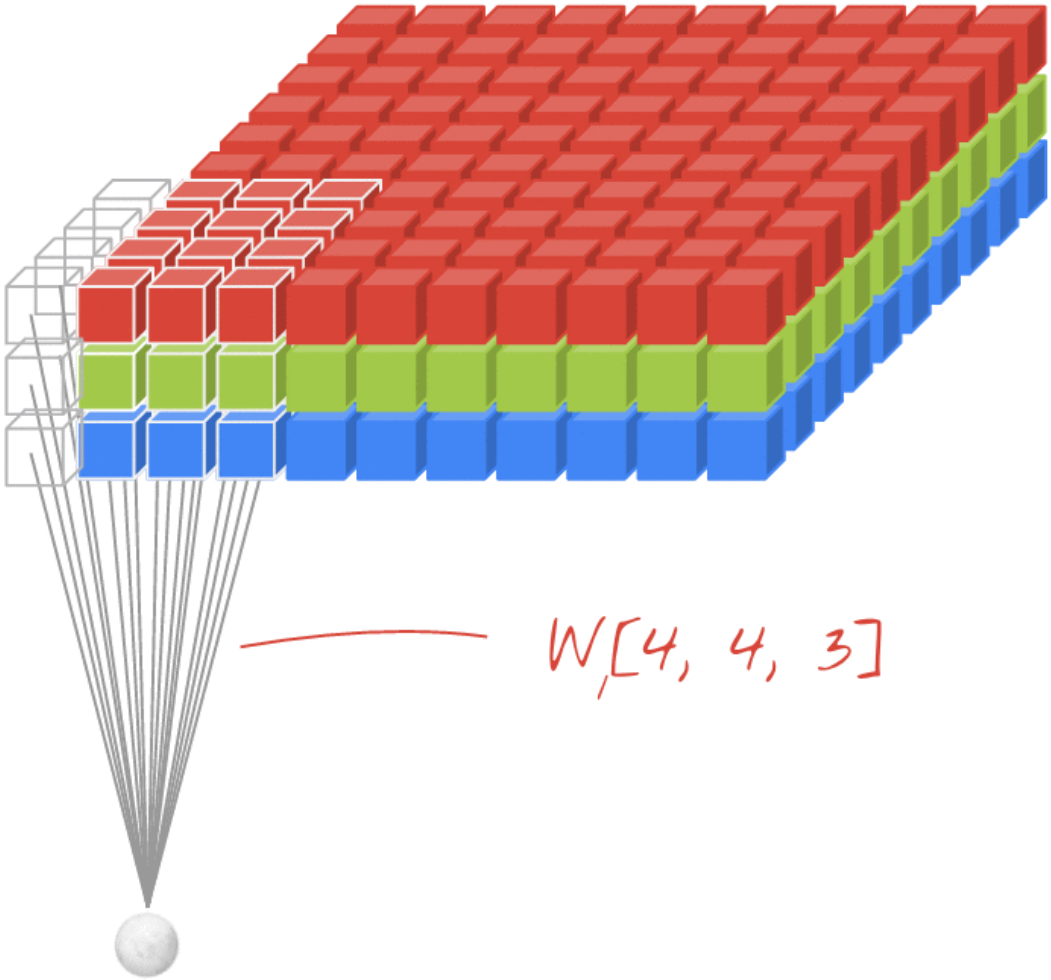
More than one kernel
Convolution layer with a 1-chanel input and 6 kernels



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 and two kernels



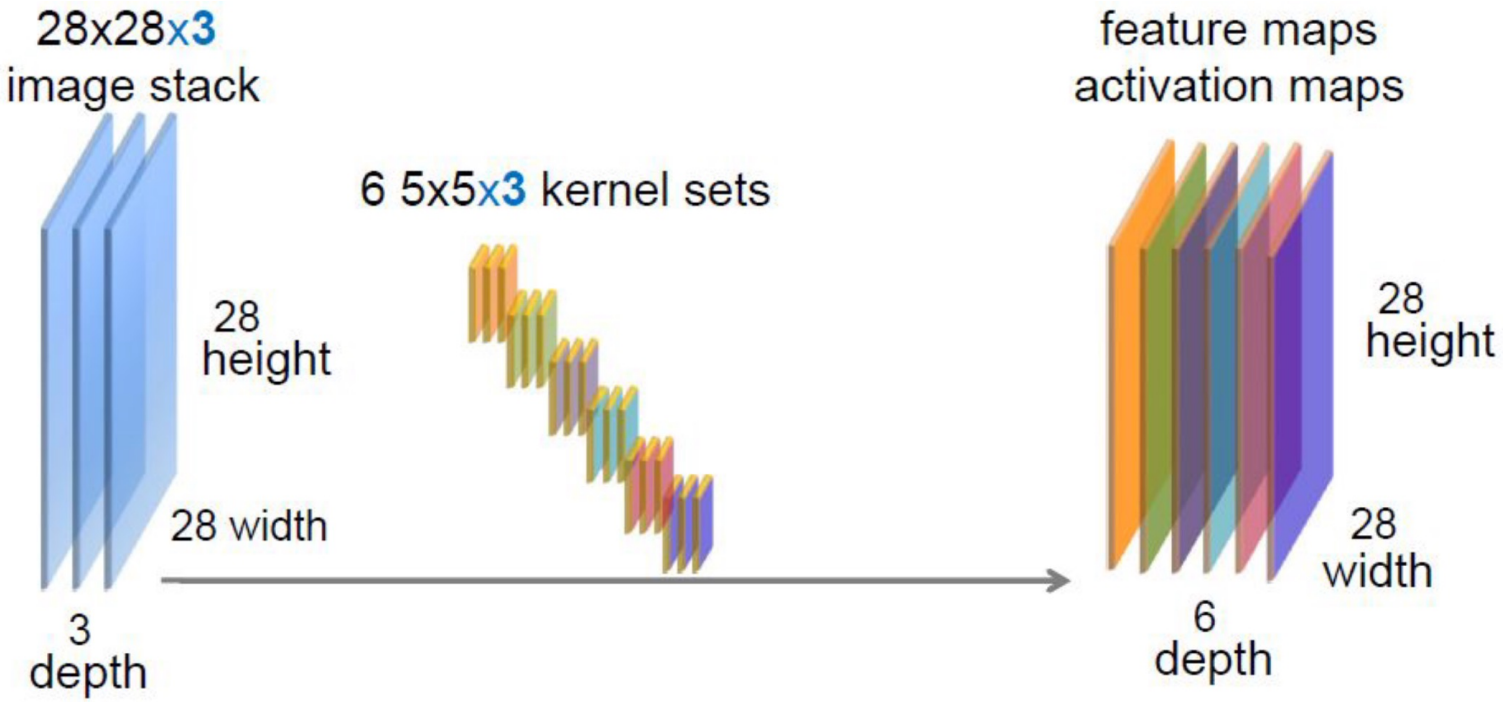
3 color channel input image

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

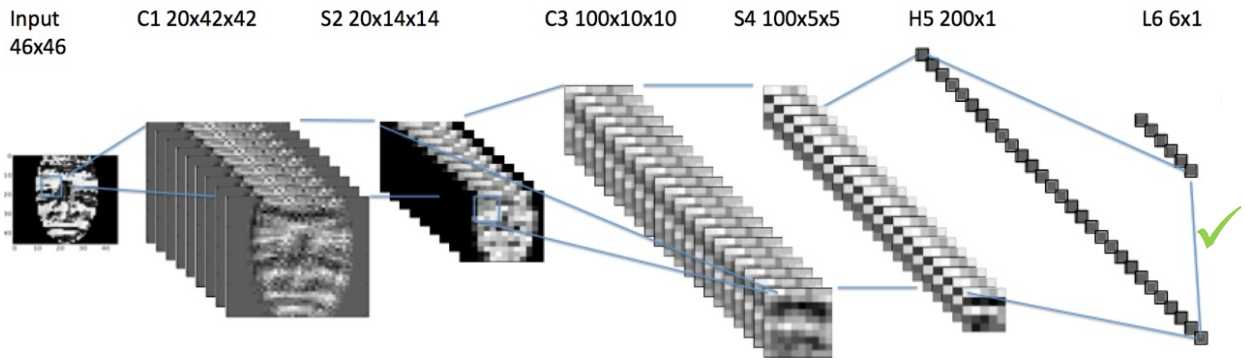
Convolution layer with a 3-chanel input and 6 kernels



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

→

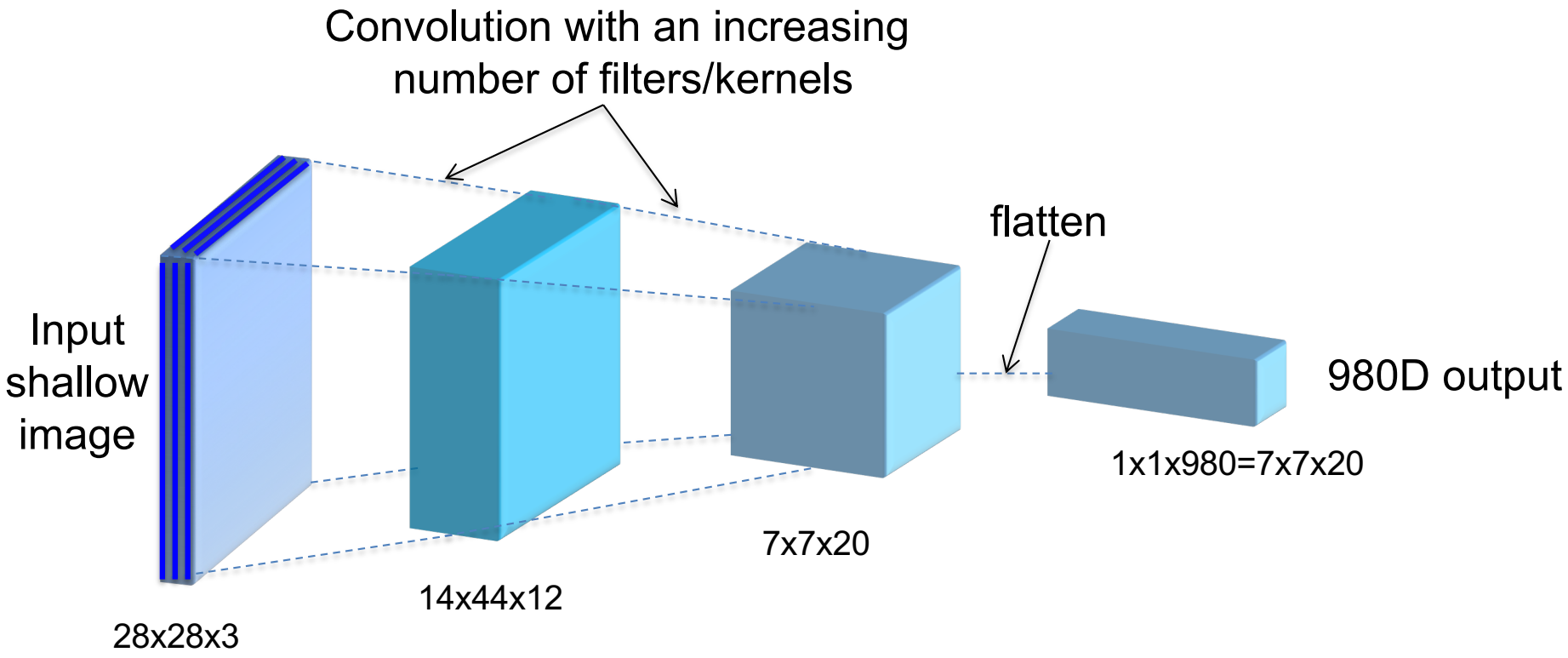
2x2

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

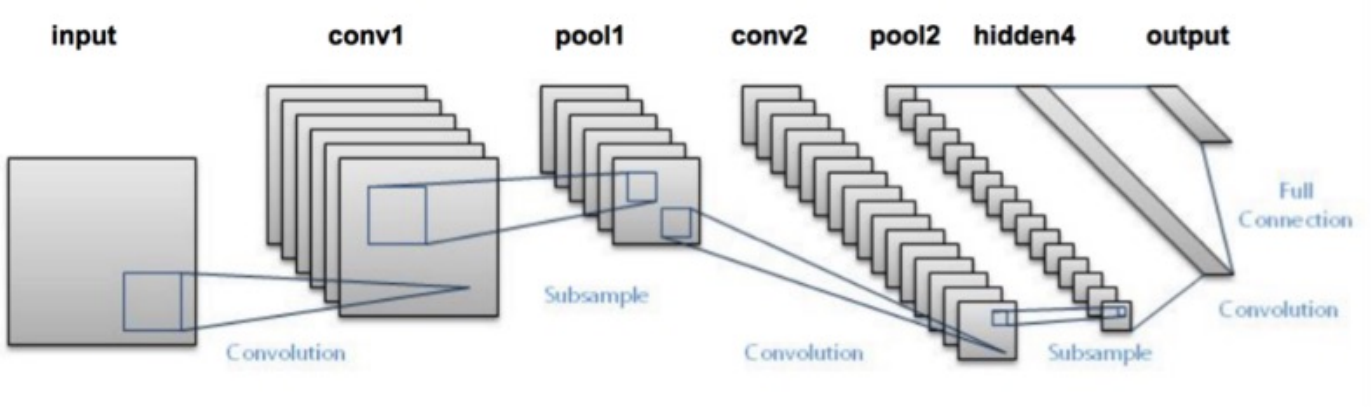
Hinton: „The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster“

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.

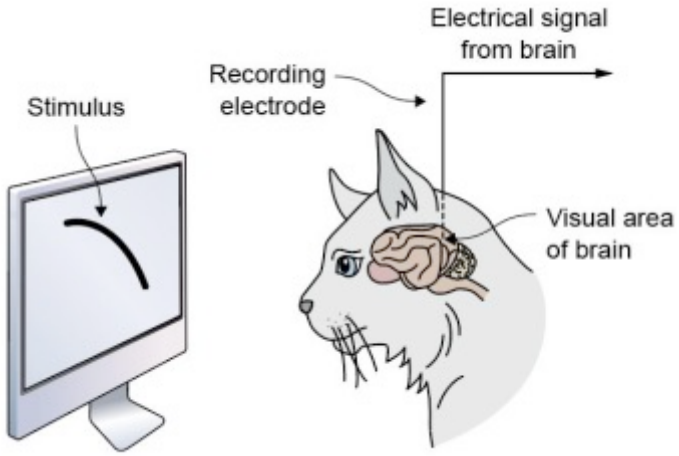
CNN for MNIST



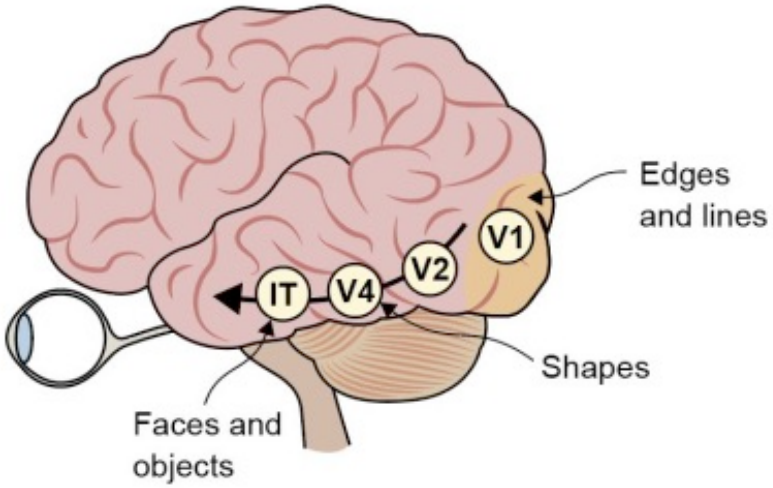

```
num_classes = 10              # 10 classes: 0, 1,…,9
input_shape = (28, 28, 1)  # 28x28 pixels, 1 channel (grey value)

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'))
```

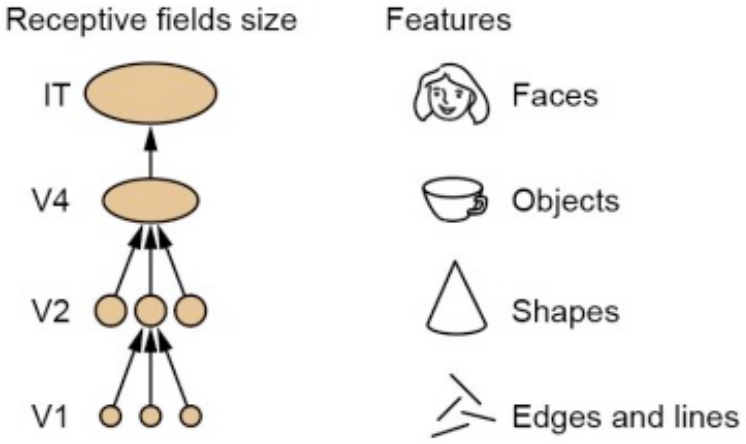
Biological Inspiration of CNNs

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.

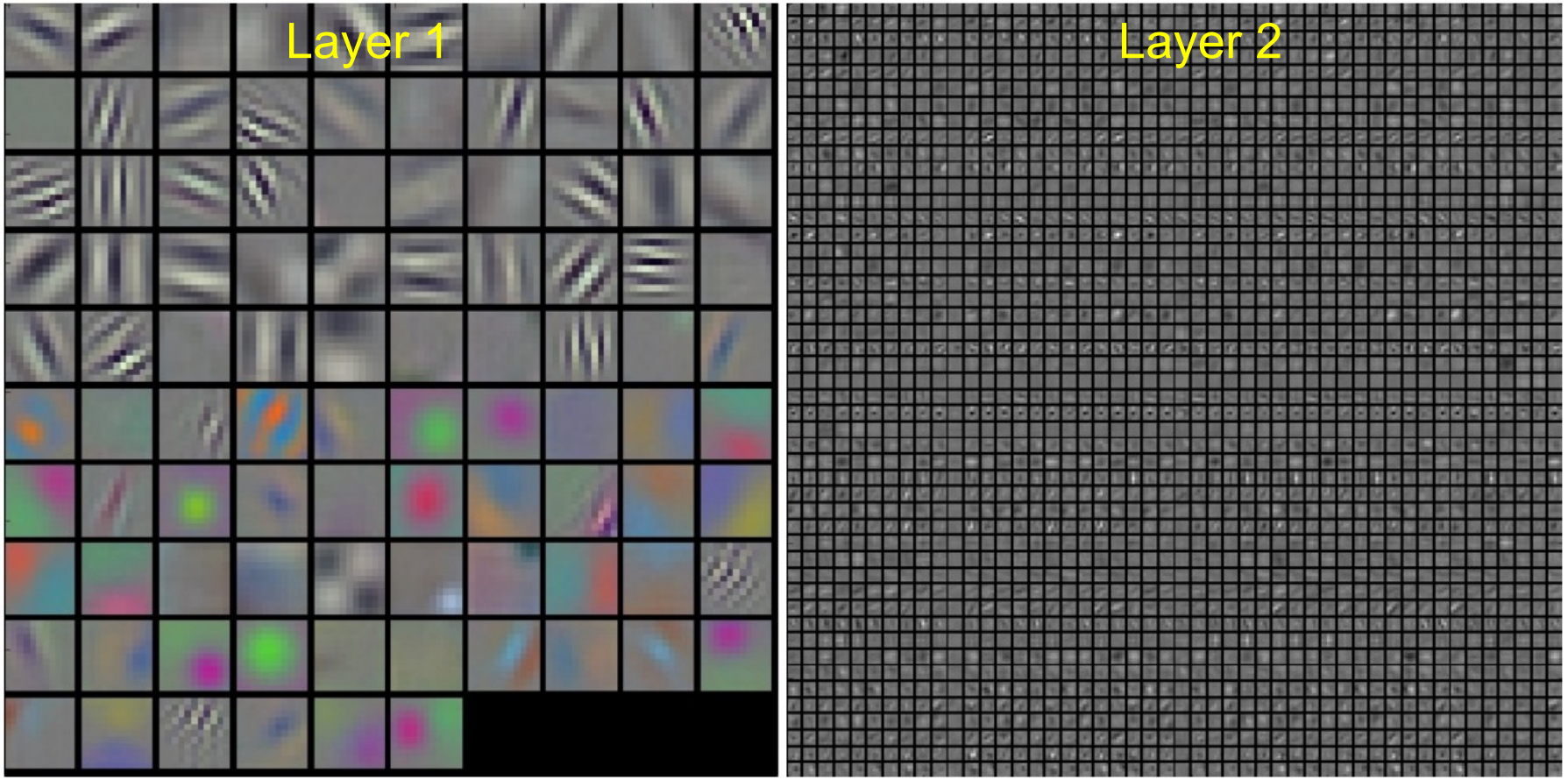




Organization of the visual cortex in a brain, where neurons in different regions respond to more and more complex stimuli

Visualize the weights used in filters

Filter weights from a trained Alex Net



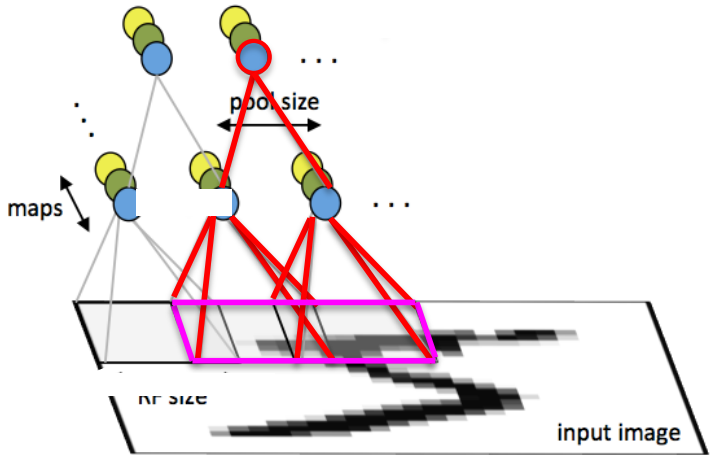
Only in layer 1 the filter pattern correspond to extracted patterns in the image.

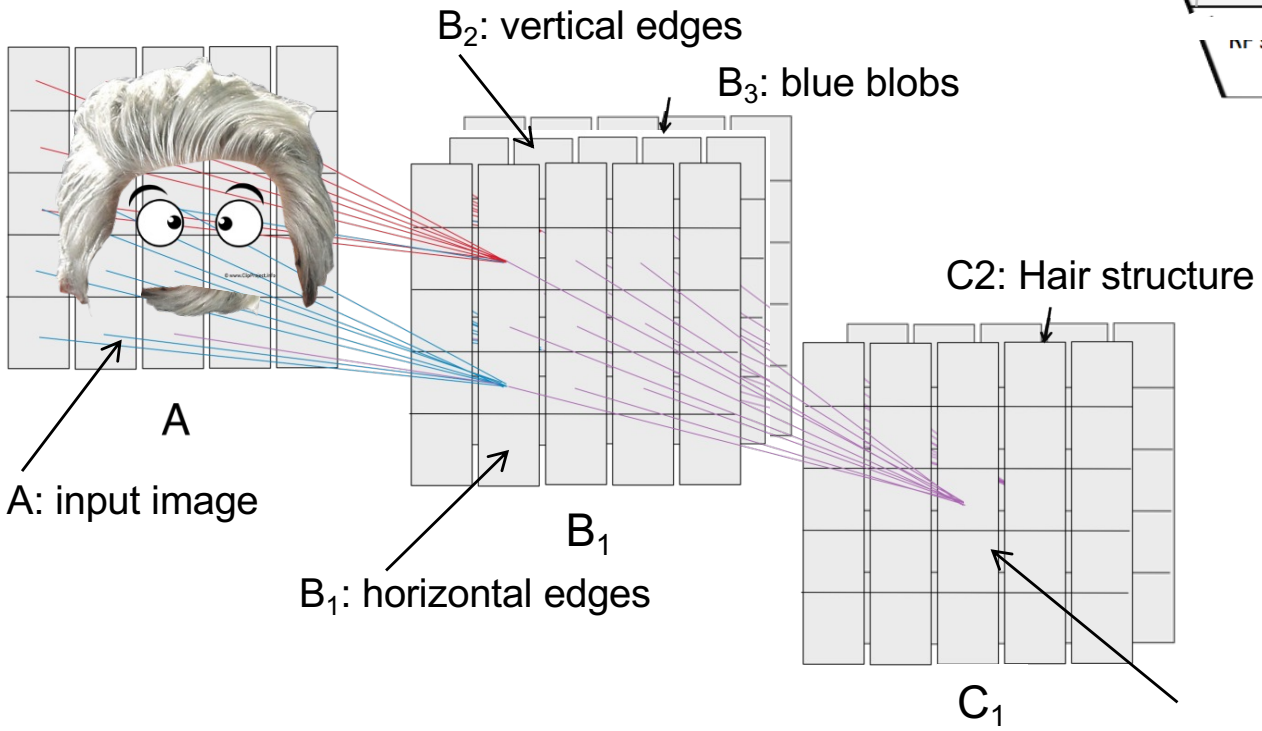
In higher layers we can only check if patterns look noisy, which would indicate that the network that hasn't been trained for long enough, or possibly with a too low regularization strength that may have led to overfitting.

Image credits: http://cs231n.github.io/understanding-cnn/

The receptive field

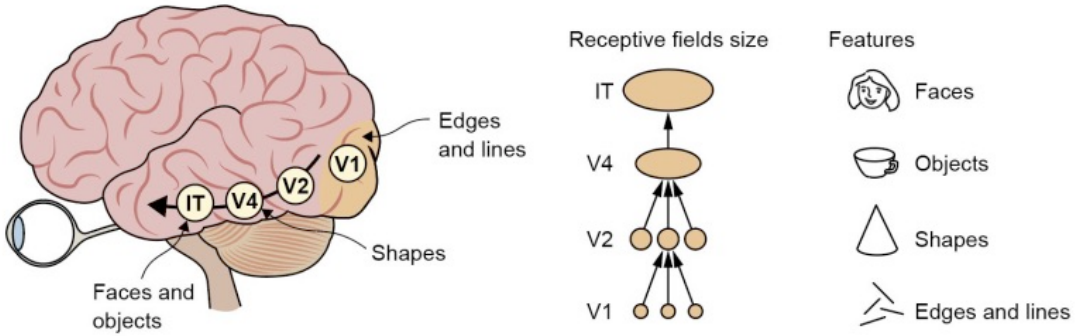
For each pixel of a feature map we can determine the connected area in the input image – this area in the input image is called receptive field.

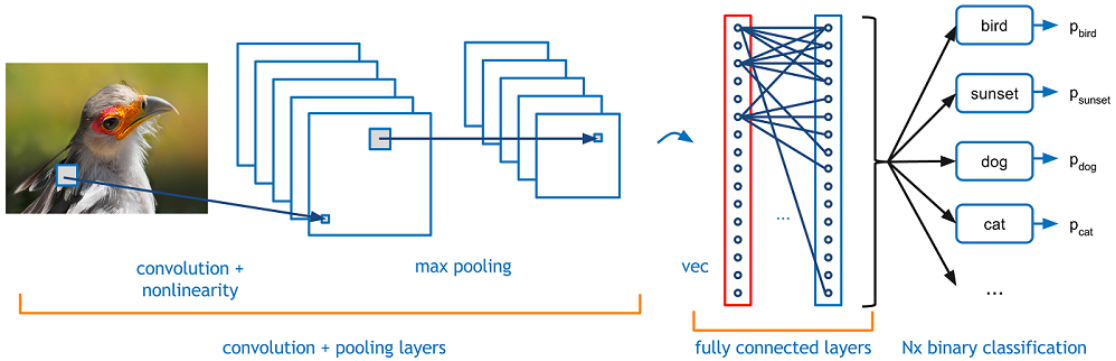


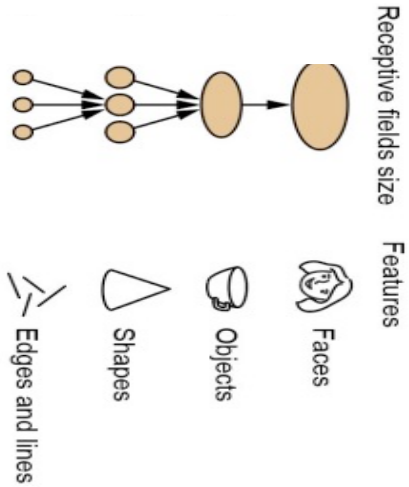


A feature map gets activated by a certain structure of the feature maps one layer below, which by itself depends on the input of a preceding layer etc and finally on the input image. Activation maps close to the input image are activated by simple structures in the image, higher maps by more complex image structures.

Weak analogies between brain and CNNs architecture



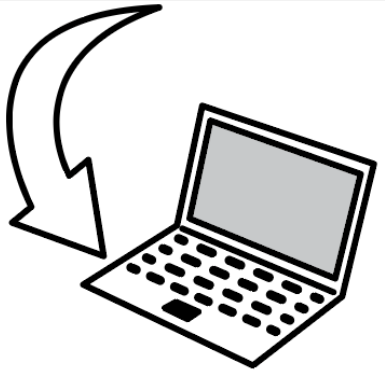


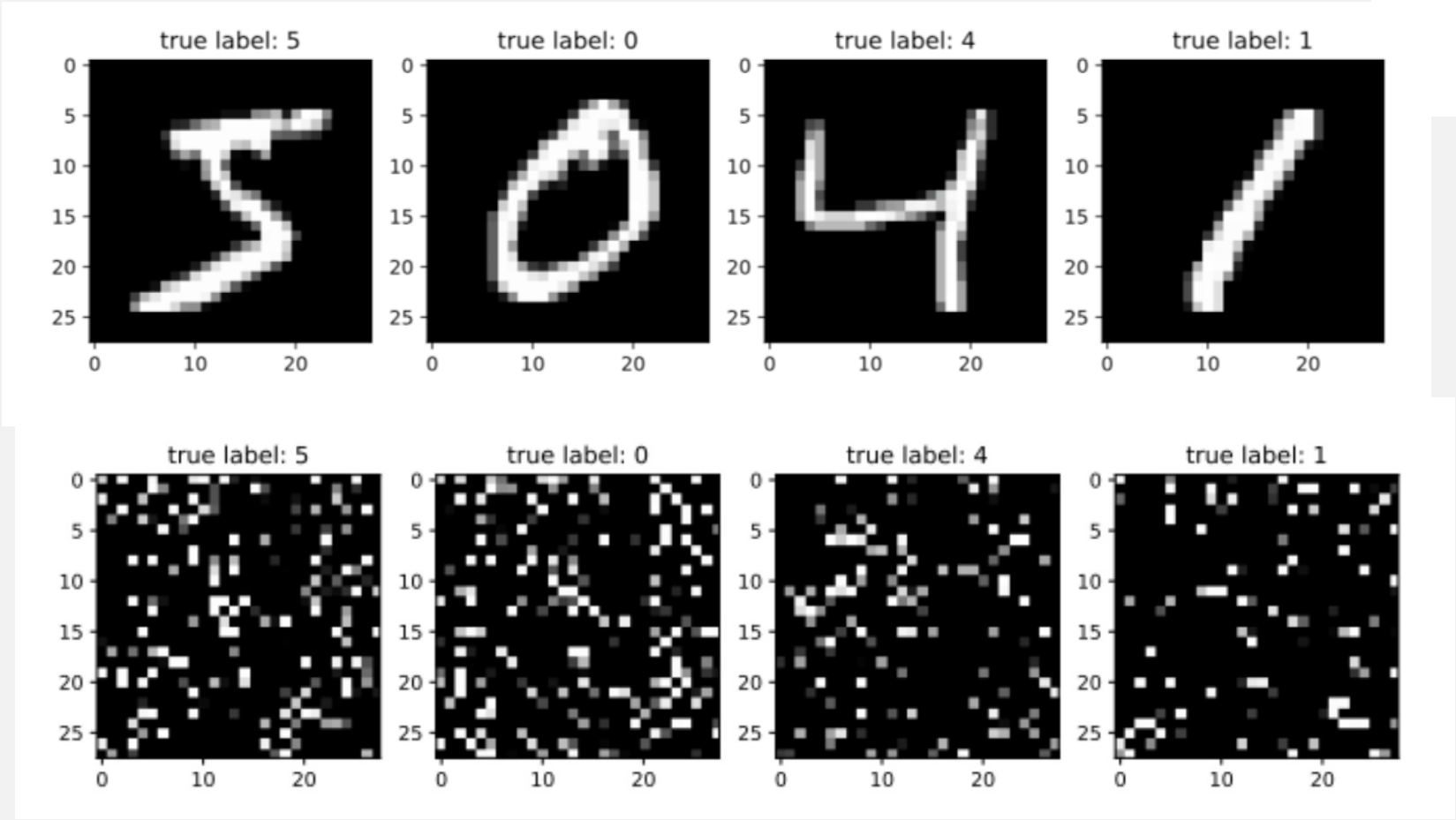


fcNN versus CNNs – some aspects

- 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

Does shuffling disturb a CNN?





Inverstigate if shuffling disturbs the CNN for MNIST **06_cnn_mnist_shuffled**

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 and fcNN
- NNs are loosely inspired by the structure of the brain.
  - When going deep the receptive field increases (~layer 5 sees whole input)
  - Deeper layer respond to more complex feature in the input