

Machine Intelligence:: Deep Learning

Week 2

Beate Sick, Oliver Dürr

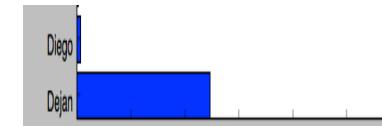
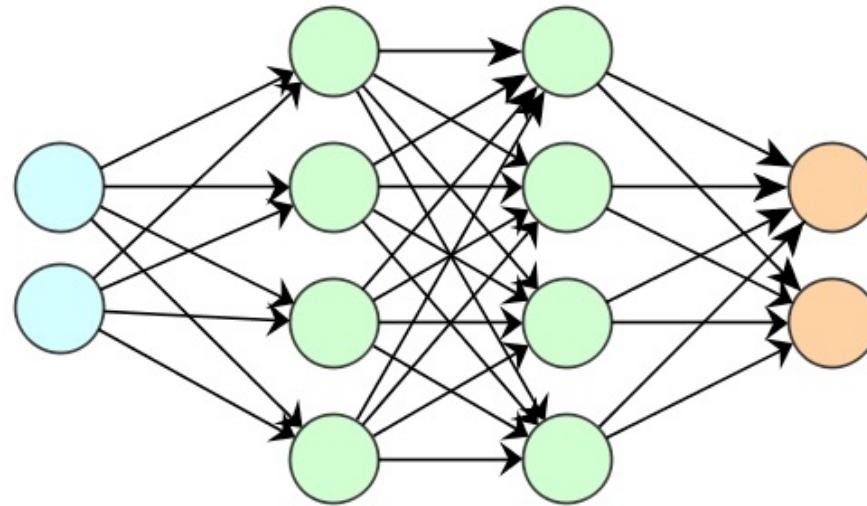
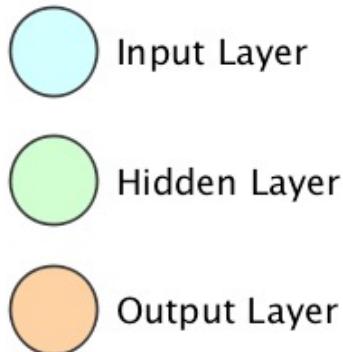
Institut für Datenanalyse und Prozessdesign
Zürcher Hochschule für Angewandte Wissenschaften

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
 - Biological inspiration of CNNs
 - 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 steroids (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

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

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

$\mathbf{f} \mapsto \mathbf{f} \cdot R = (\mathbf{e}_i R_1^i, \dots, \mathbf{e}_i R_n^i)$

Just kidding...

then the multidimensional array obeys the transformation law

$$T_{j'_1 \dots j'_q}^{i'_1 \dots i'_p} [\mathbf{f} \cdot R] = (R^{-1})_{i_1}^{i'_1} \dots (R^{-1})_{i_p}^{i'_p} T_{j_1, \dots, j_q}^{i_1, \dots, i_p} [\mathbf{f}] R_{j'_1}^{j_1} \dots R_{j'_q}^{j_q}.$$

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
```

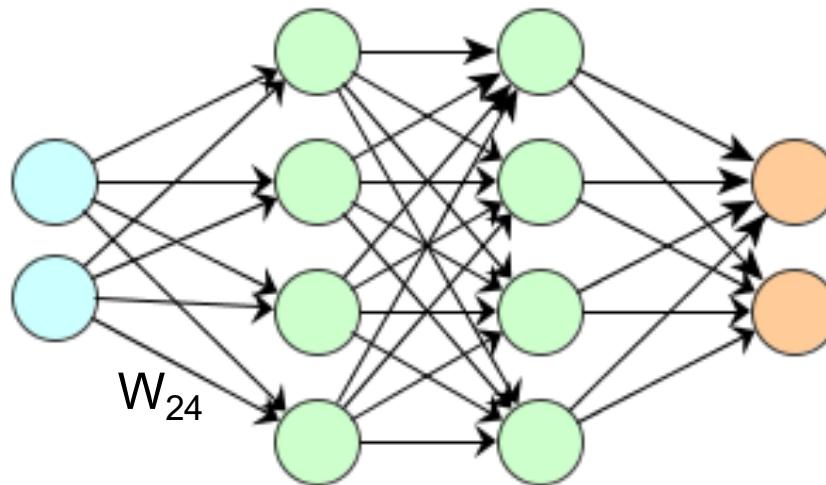
```
Out[3]: array([[1, 2, 3],  
               [4, 5, 6]])
```

```
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

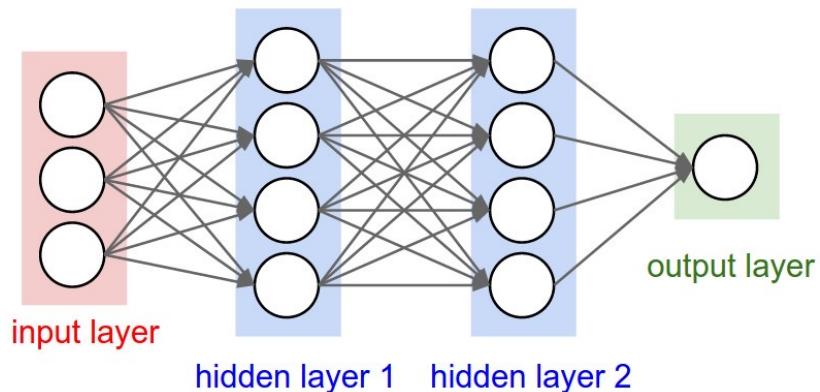


- The input can be understood as a vector
- A mini-batch of size 64 of input vectors can be understood as tensor of order 2
 - (index in batch, x_j)
- The weights going from e.g. Layer L_1 to Layer L_2 can be written as a matrix (often called W)
- A mini-batch of size 64 images with 256,256 pixels and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

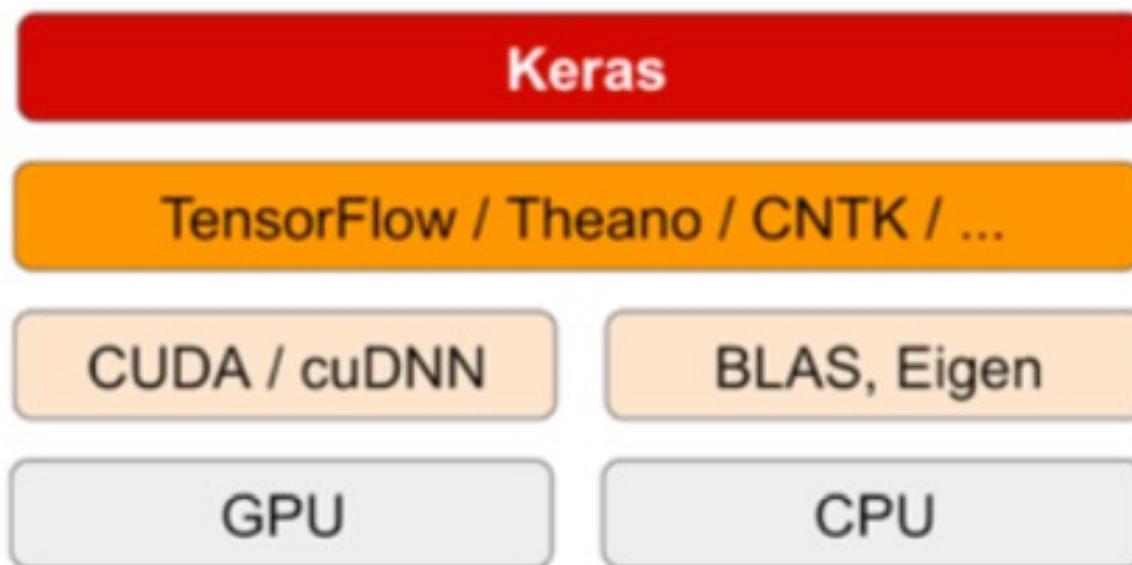


Figure: From Deep Learning with Python, Francois Chollett

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.

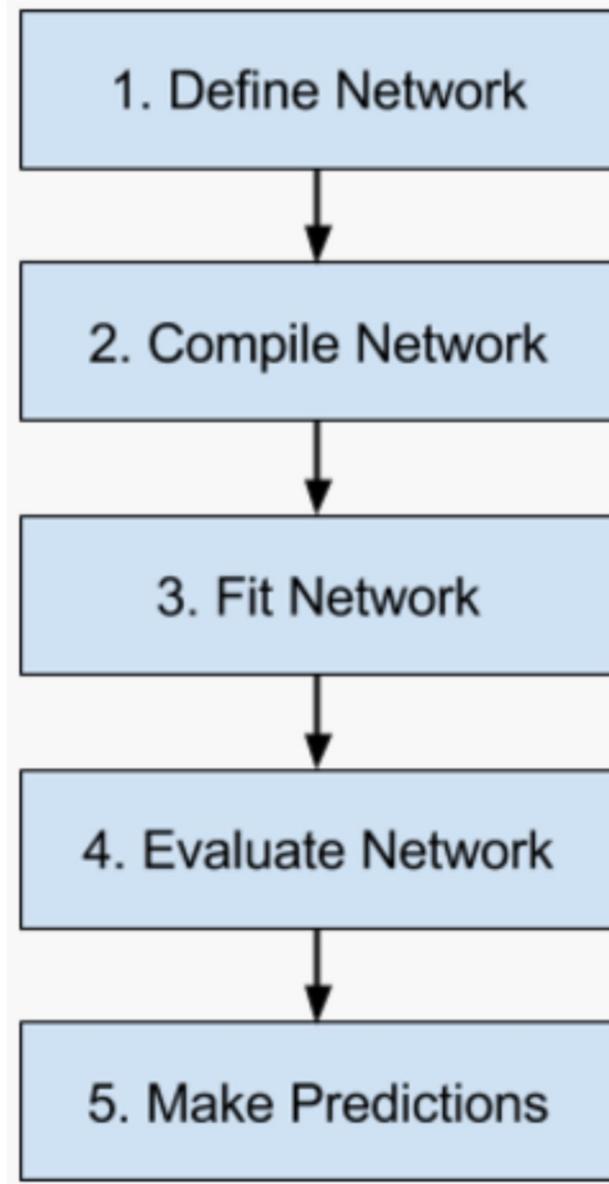
This makes Keras easy to learn and easy to use. As a Keras user, you are more productive, allowing you to try more ideas than your competition, faster -- which in turn helps you win machine learning competitions.

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.

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...

Keras Workflow



Define the network (layerwise)

Add loss and optimization method

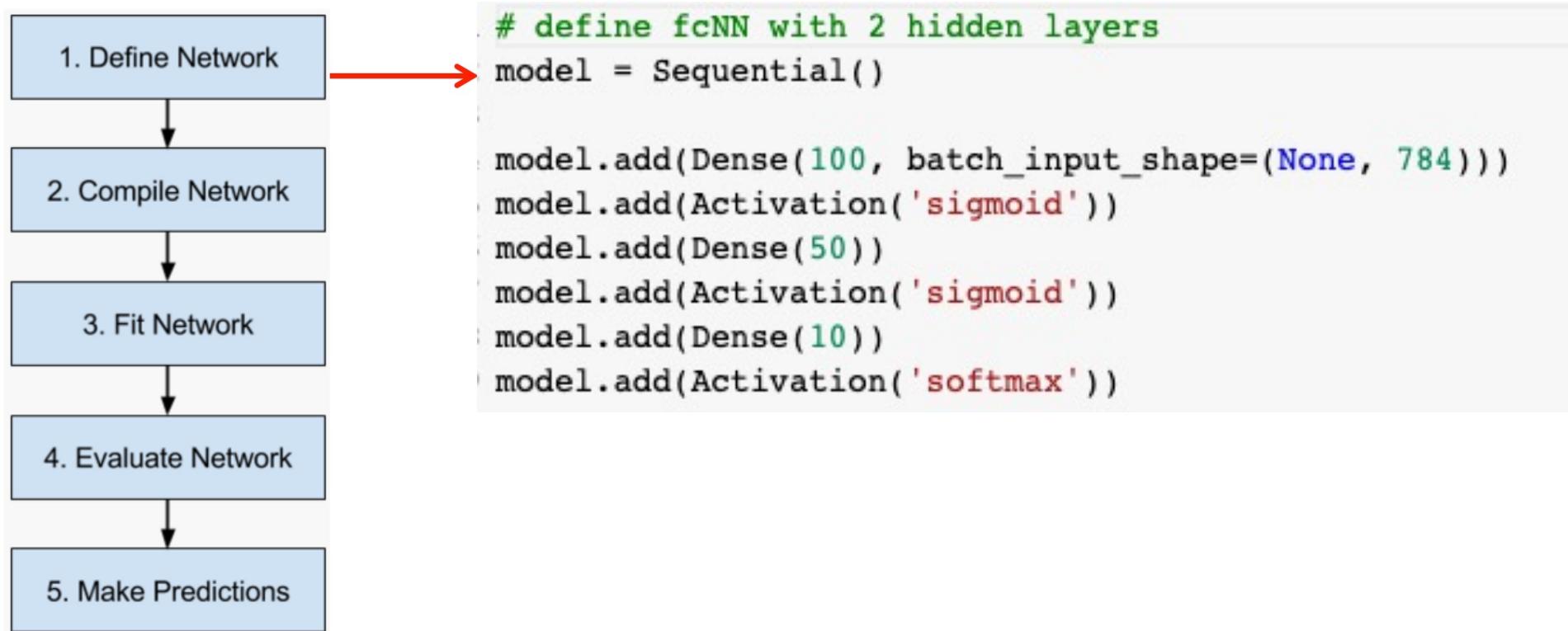
Fit network to training data

Evaluate network on test data

Use in production

A first run through

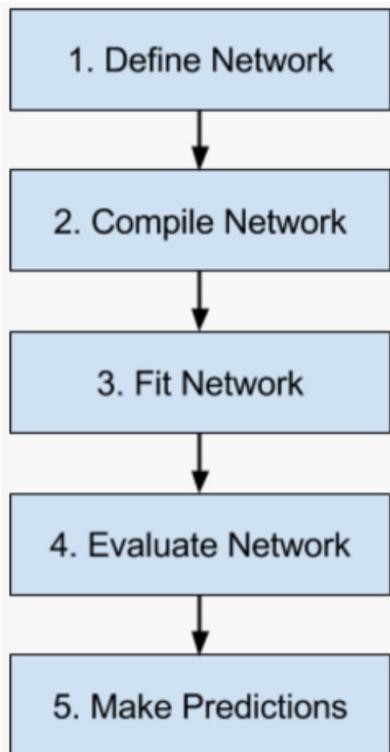
Define the network



Input shape needs to be defined only at the beginning.

Alternative: `input_dim=784`

Compile the network



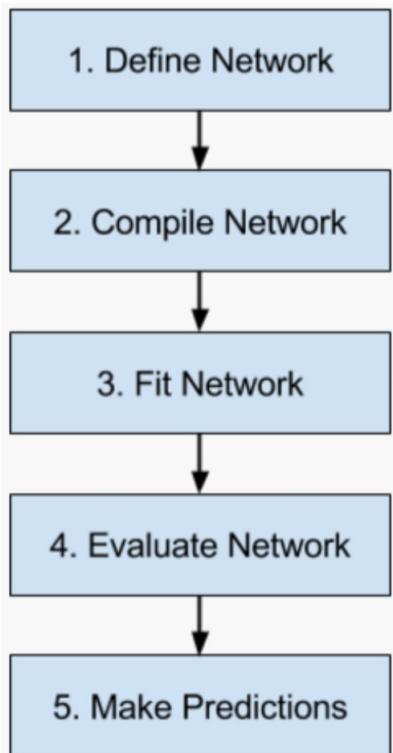
```
model.compile(loss='categorical_crossentropy',  
              optimizer='adadelta',  
              metrics=['accuracy'])
```

loss function that will be minimized

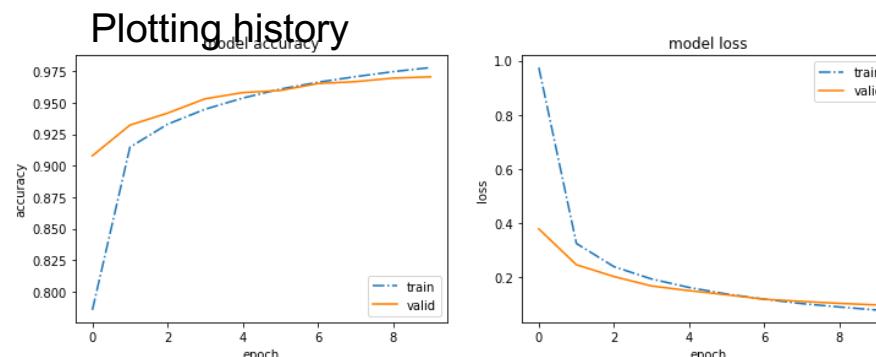
easiest optimizer is SGD
(stochastic gradient descent)

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

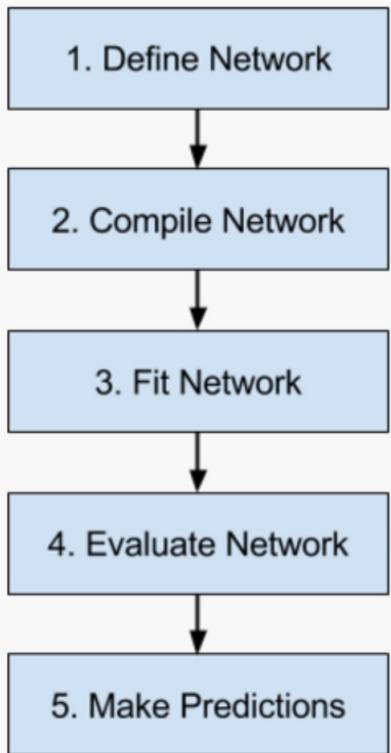
Fit the network



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



Evaluate the network



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

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

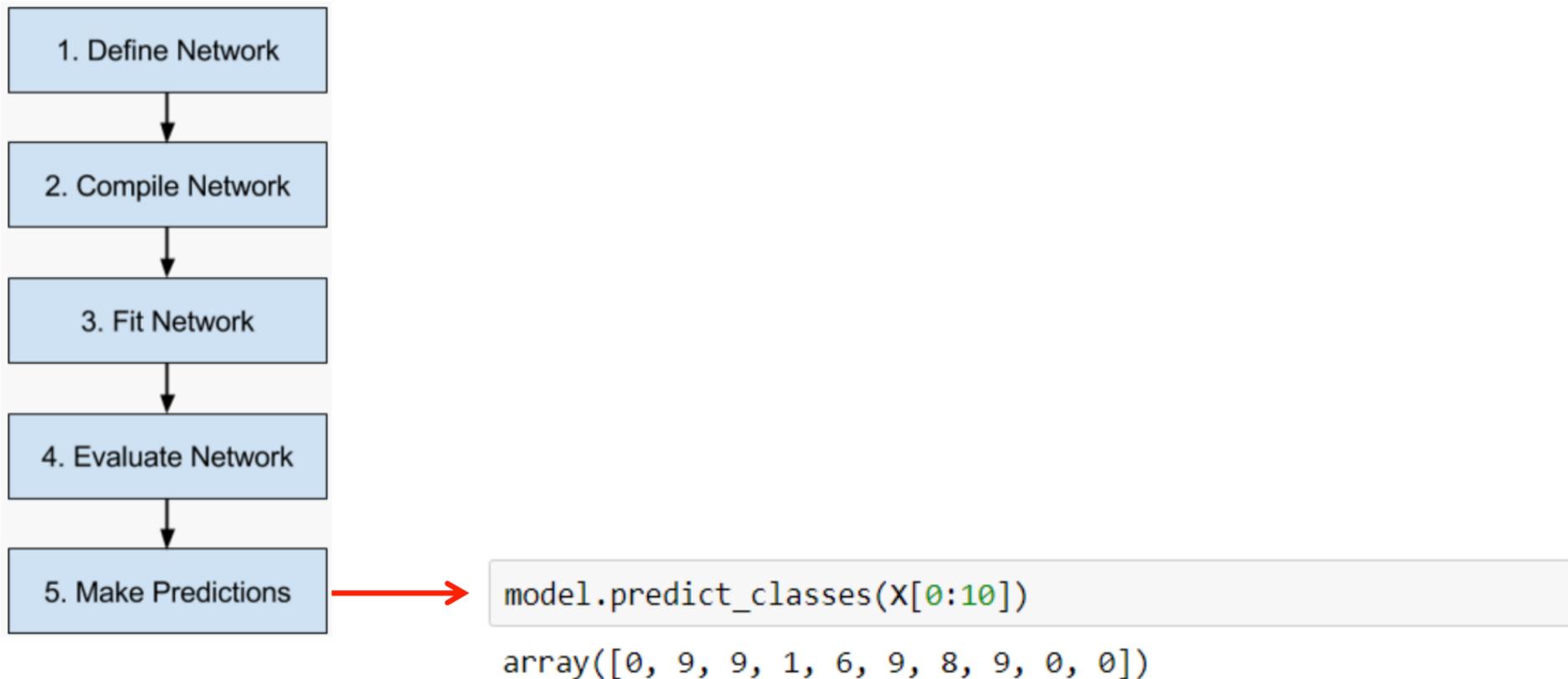
```
[2.5549037284851073, 0.1085]
```

↑
loss

←
metrics: accuracy

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

Make Predictions



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

Building a network (API Styles)

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 completeness]

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

Exercise

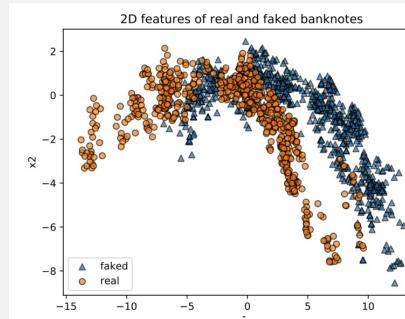


1. Finish NB 01_simple_forward_pass.ipynb have a look at the Keras implementation

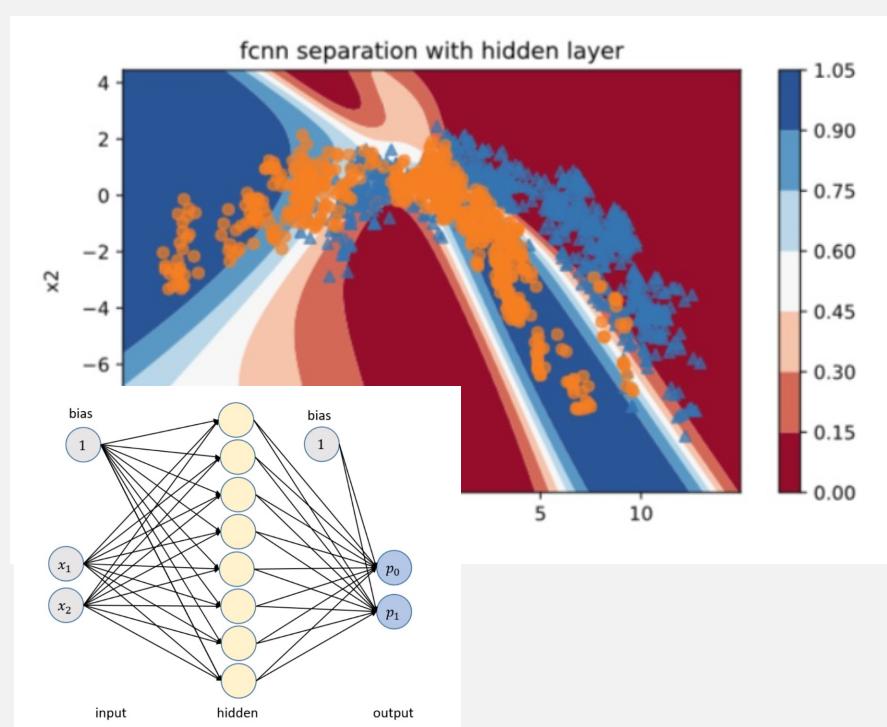
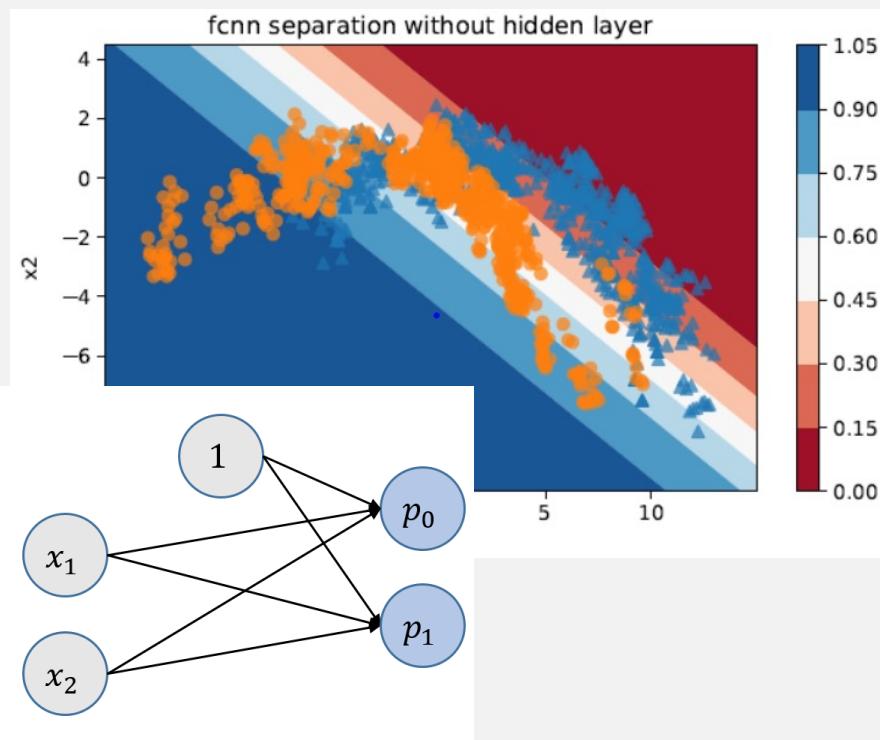
Homework

Do exercise NB 02:

Use Keras for modeling the banknotes data with binary outcome and 2 continuous input features (x_1, x_2)

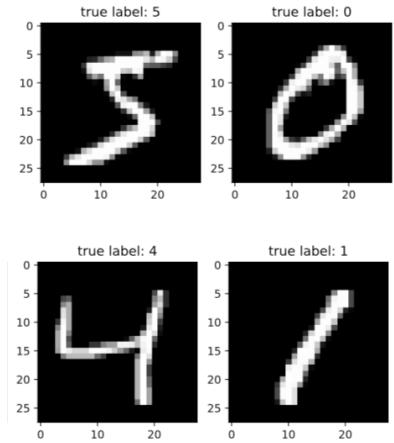


Fit the a fcNN w/o and with hidden layer to discriminate fake bank notes from real bank notes: NB02

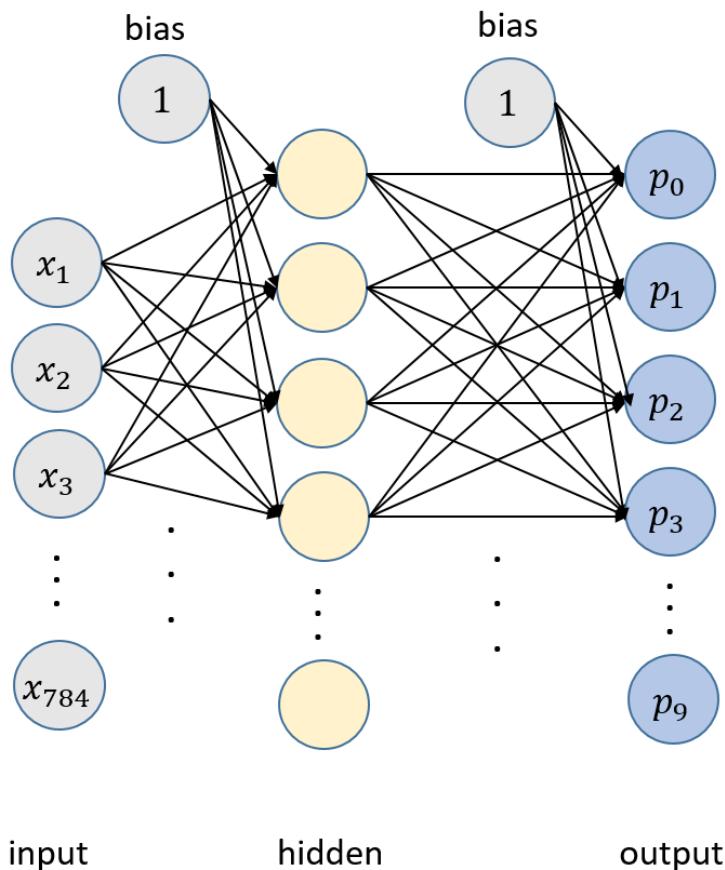


Fully connected NN for image data
Good Idea?

A fcNN for MNIST data



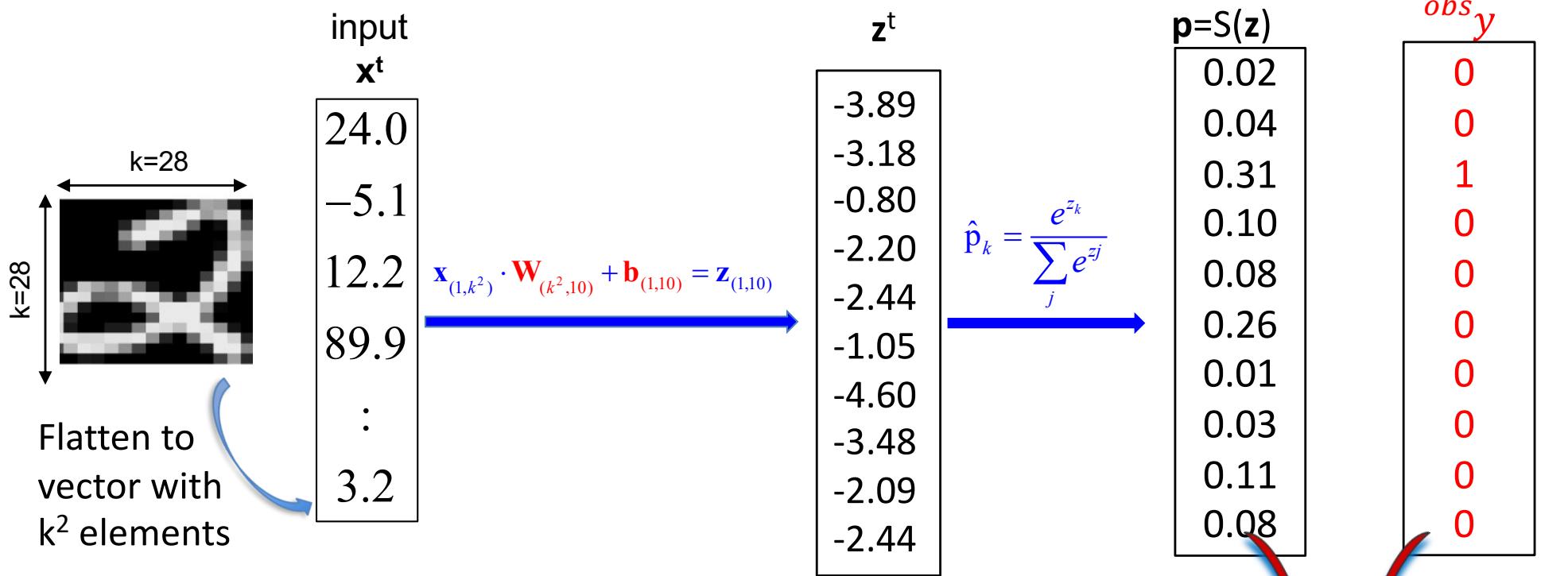
The first four digits of the MNIST data set - each image consisting of $28 \times 28 = 784$ pixels



A fully connected NN with 2 hidden layers.

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

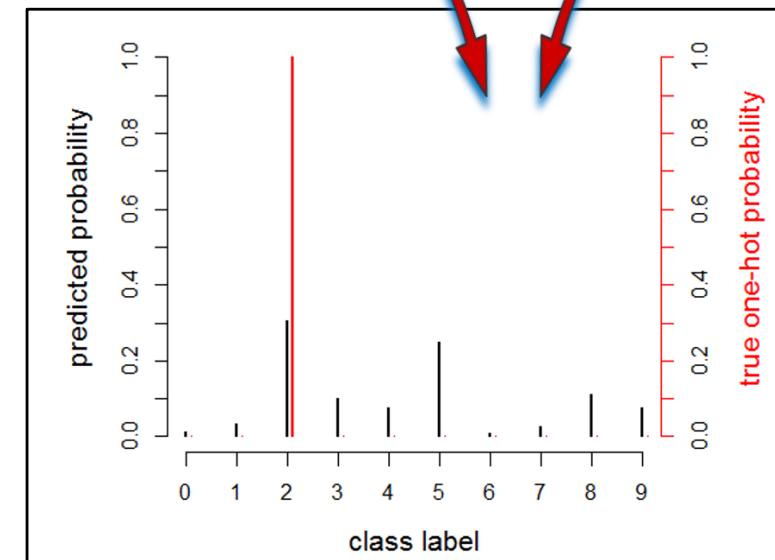
What is going on in a 1 layer fully connected NN?



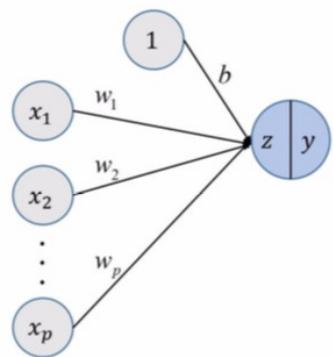
Cost C or Loss = cross-entropy averaged over all images in mini-batch

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

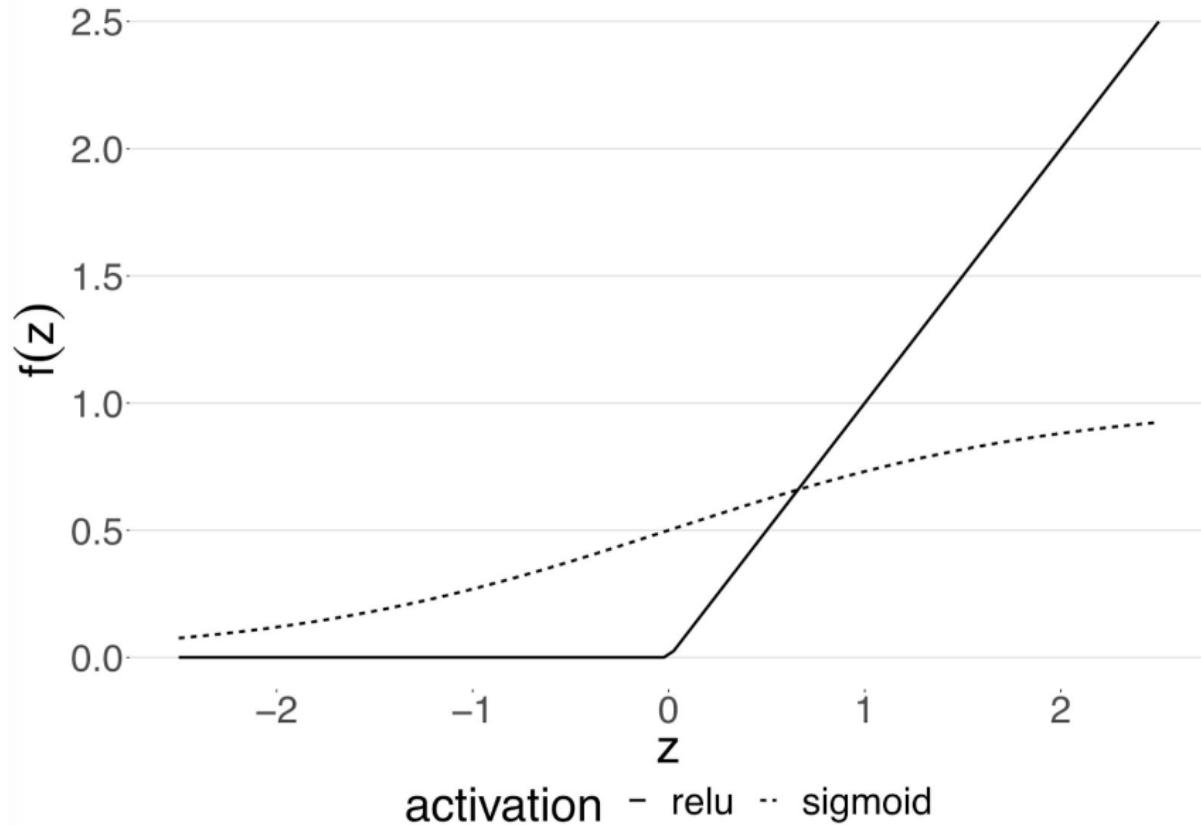
$$D(\mathbf{p}, \mathbf{y}) = -\sum_{k=1}^{10} {}^{obs} y_k \cdot \log(p_k)$$



Common non-linear activation function

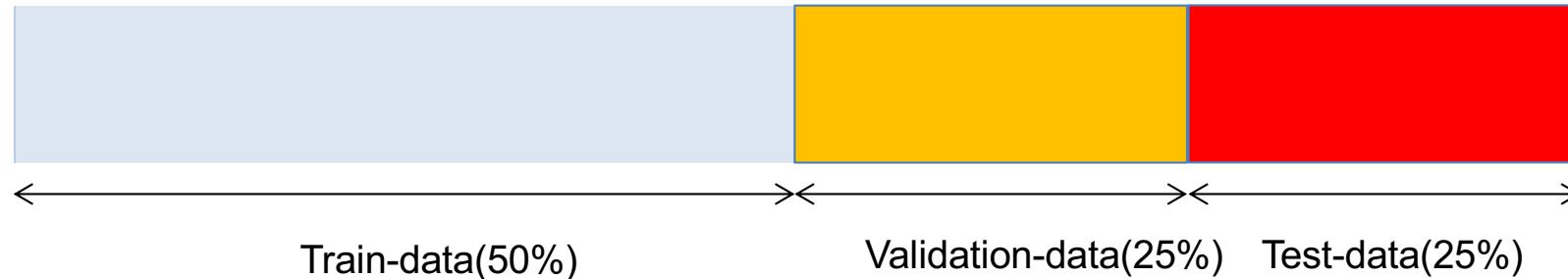


$$y = f(z) = f(b + \sum x_i \cdot w_i)$$



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

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



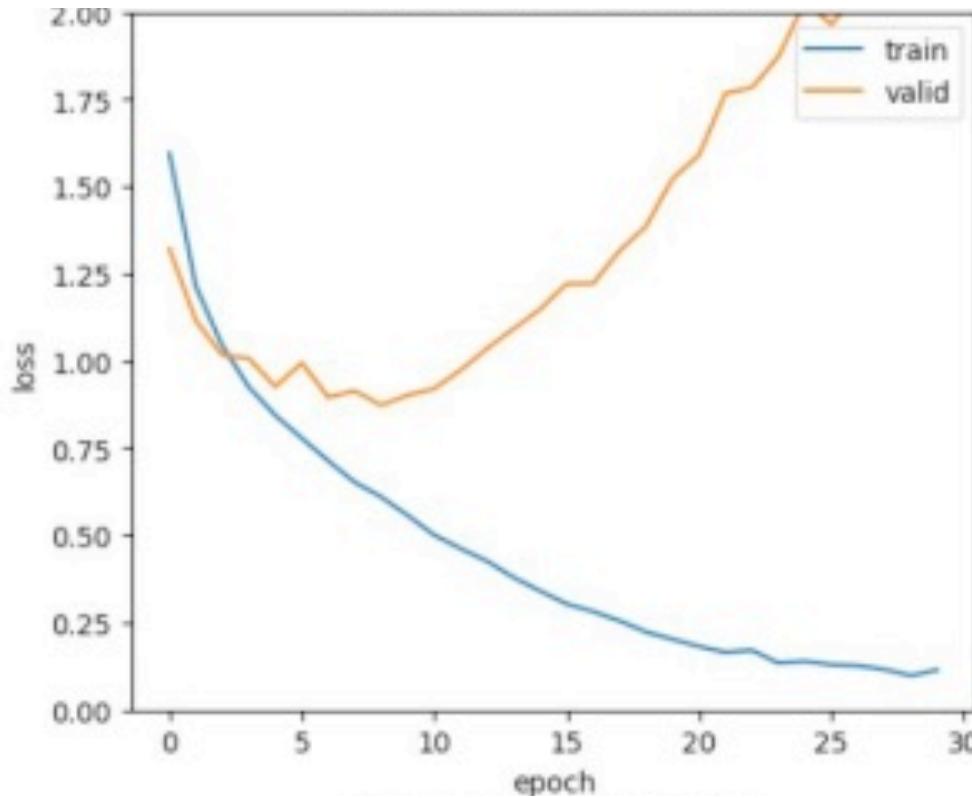
Best practice: Lock an extra **test data set** away, and use it only at the very end, to evaluate the chosen model, that performed best on your validation set.

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**

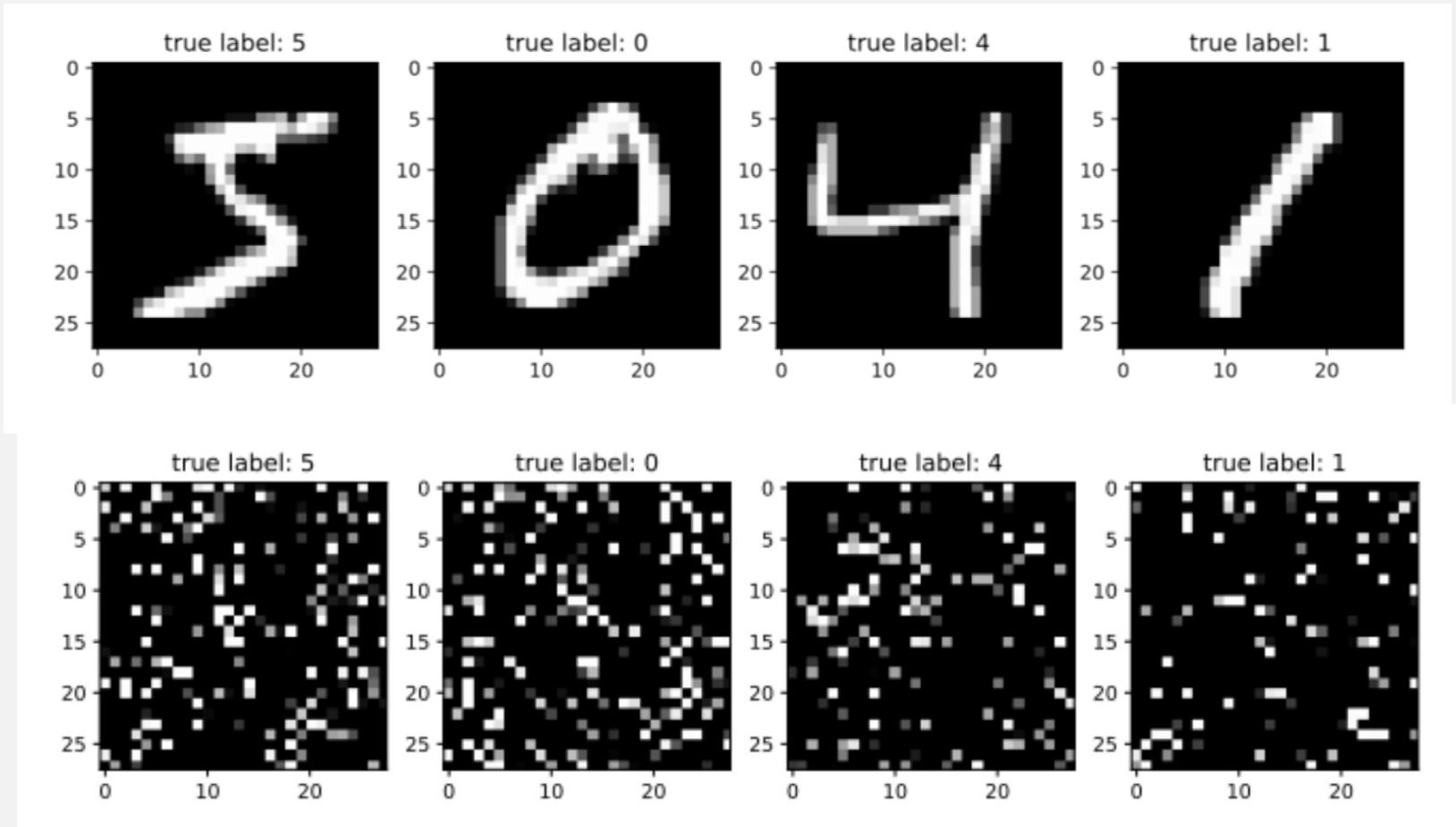
What can loss curves tell us?

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

Exercise: Does shuffling disturb a fcNN?



Use fcNN for MNIST:

https://github.com/tensorchiefs/dl_course_2023/blob/master/notebooks/03_fcnn_mnist.ipynb

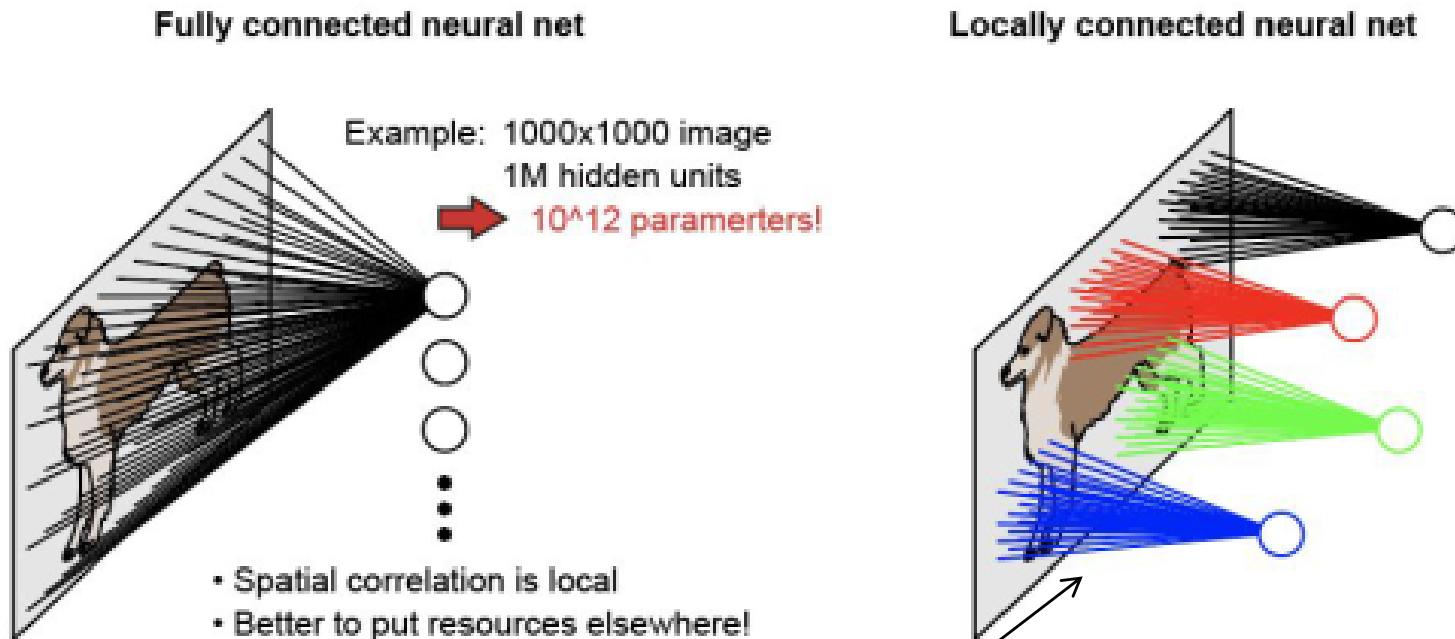
Inverstigate if shuffling disturbs the fcNN for MNIST:

https://github.com/tensorchiefs/dl_course_2023/blob/master/notebooks/04_fcnn_mnist_shuffled.ipynb

Convolutional Neural Networks

SoA for image data

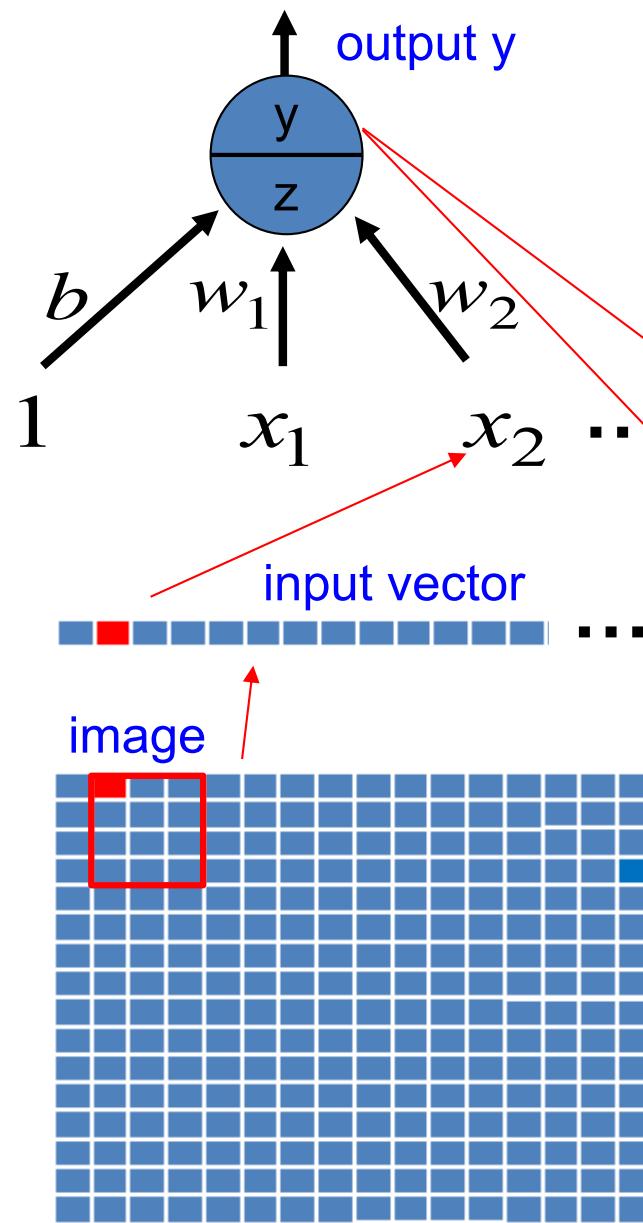
Convolution extracts local information using few weights



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

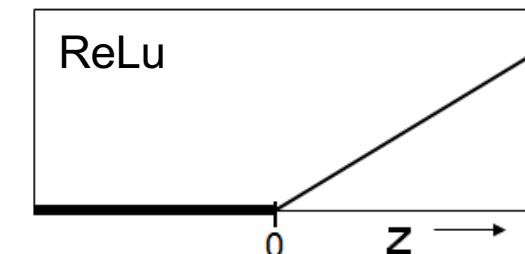
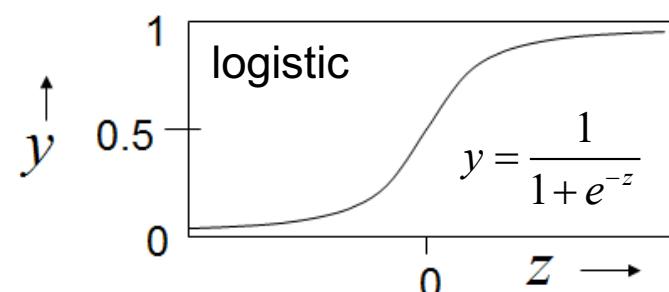


$$z = b + \sum_i x_i w_i$$

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

$$\Sigma \rightarrow \sigma$$

$$\Sigma \rightarrow \text{ReLU}$$

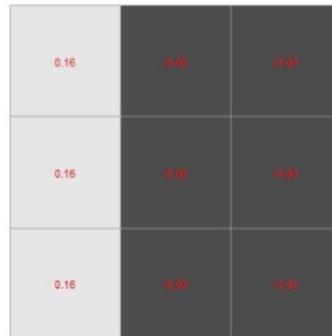


Convolutional networks use neighborhood information and replicated local feature extraction

In a locally connected network the calculation rule

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

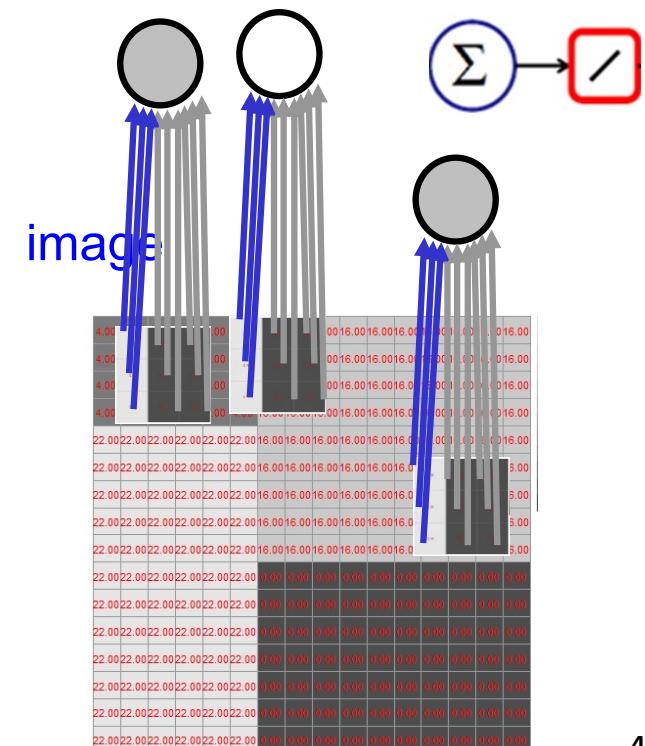
Pixel values in a small image patch are element-wise multiplied with weights of a small filter/kernel:



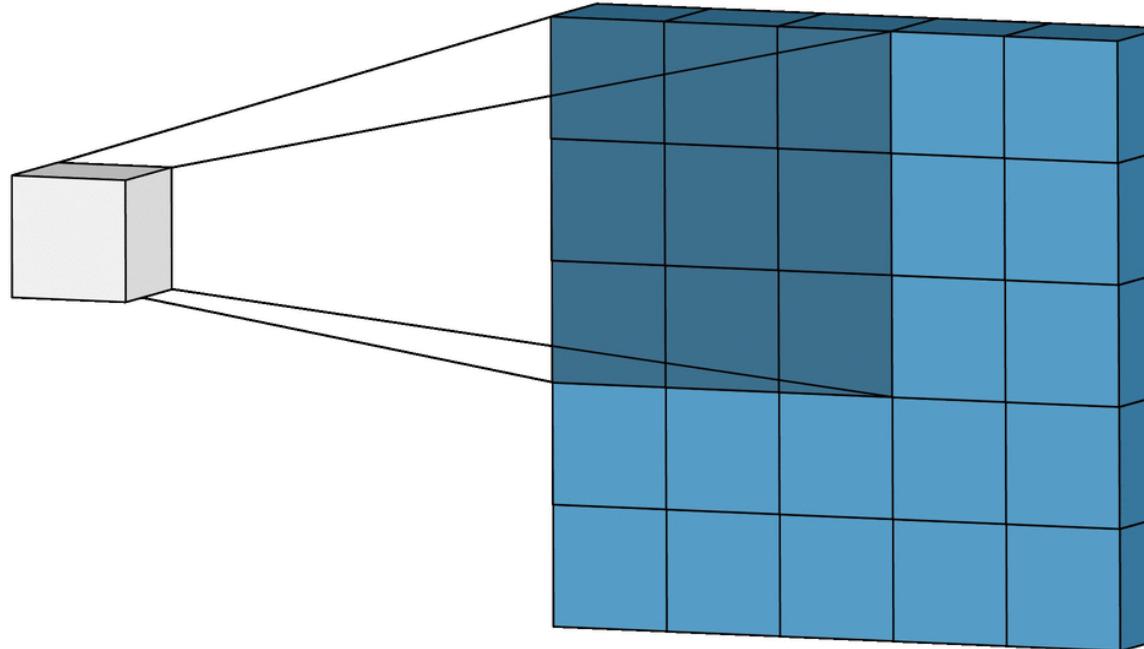
The filter is applied at each position of the image and it can be shown that the result is maximal if the image pattern corresponds to the weight pattern.

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

feature/activation map



Applying the same 3×3 kernel at each image position



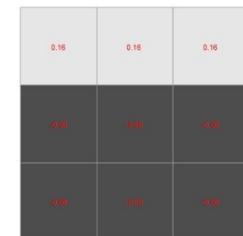
Applying the 3×3 kernel on a certain position of the image yields one pixel within the activation map where the position corresponds to the center of the image patch on which the kernel is applied.

Convolutional networks use neighborhood information and replicated local feature extraction

filtering = convolution

kernel 1

image



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 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.

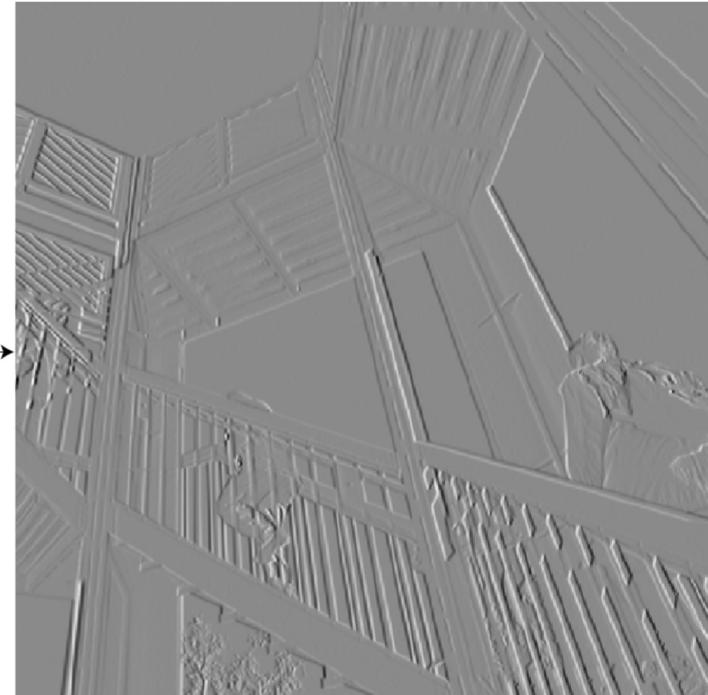
3	3	2	1	0
0 ₀	0 ₁	1 ₂	3	1
3 ₂	1 ₂	2 ₀	2	3
2 ₀	0 ₁	0 ₂	2	2
2	0	0	0	1

Example of designed Kernel / Filter



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

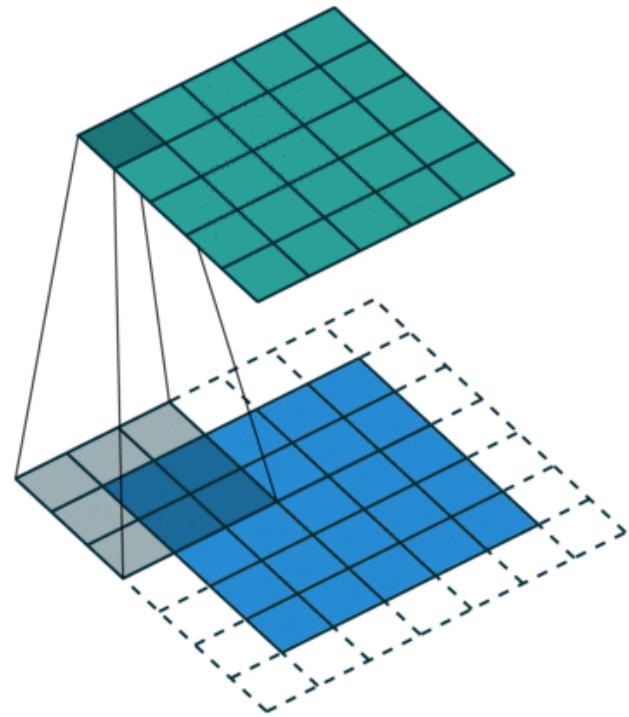
Horizontal Sobel kernel



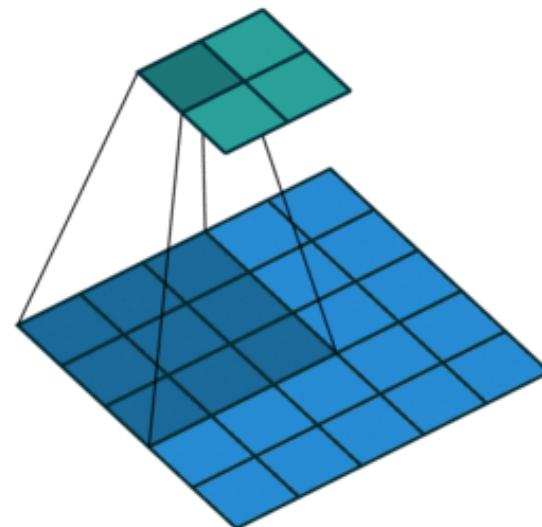
Applying a vertical edge detector kernel

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

CNN Ingredient I: Convolution



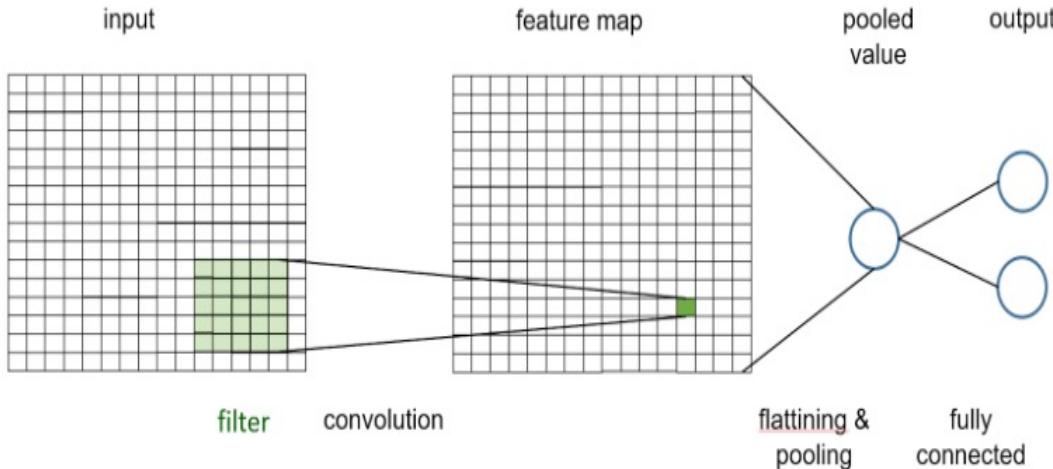
Zero-padding to achieve
same size of feature and input



no padding to only use
valid input information

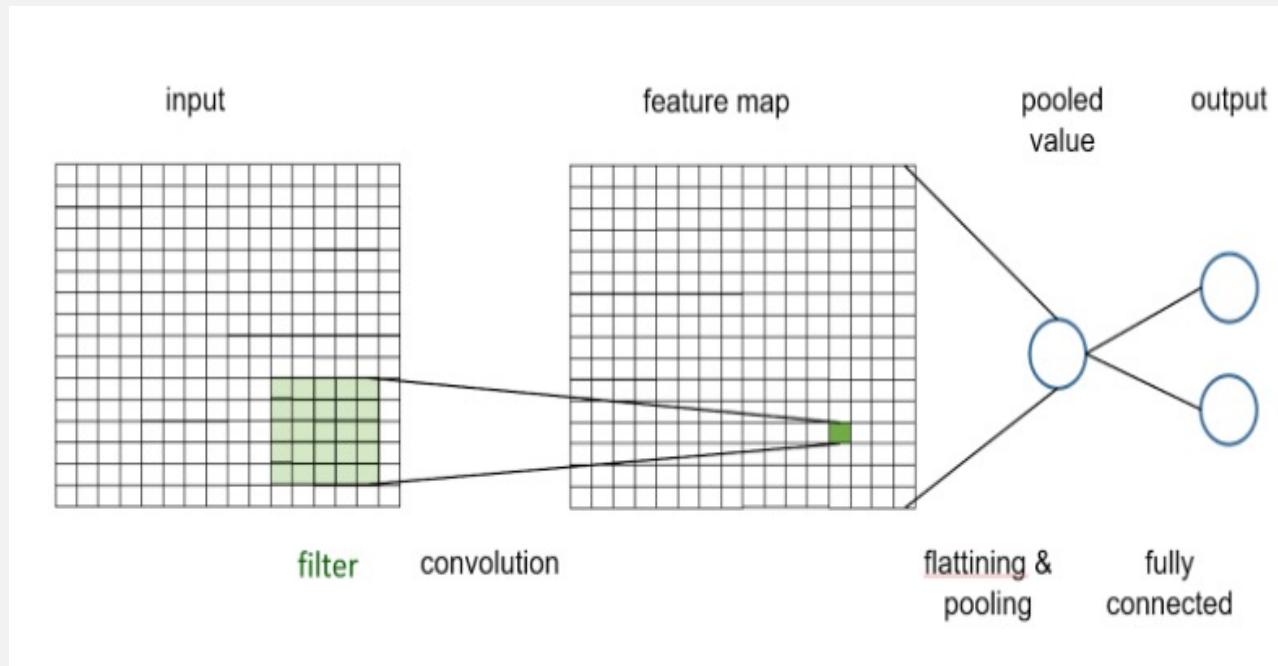
The *same* weights are used at each position of the input image.

Building a very simple CNN with keras



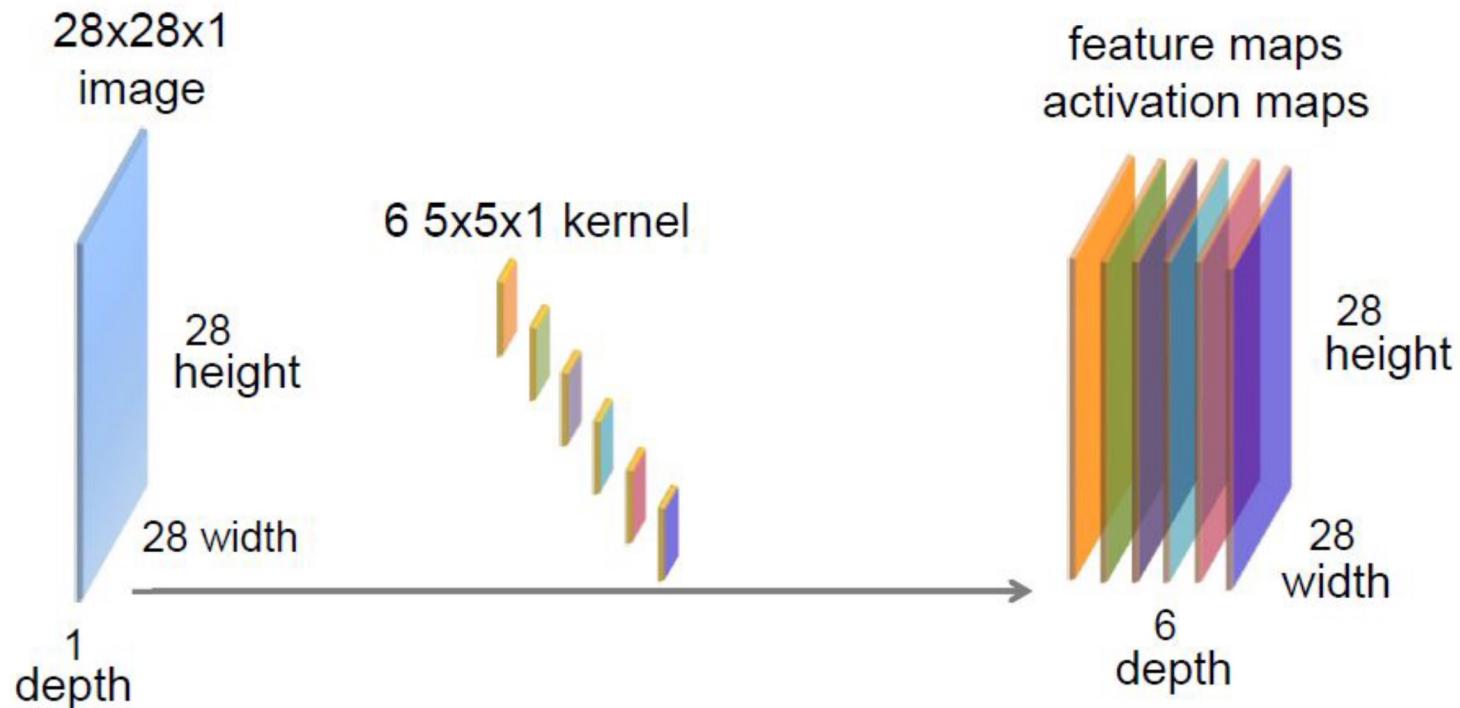
```
model = Sequential()
model.add(Convolution2D(1, (5,5), # one 5x5 kernel
                      padding='same',           # zero-padding to preserve size
                      input_shape=(pixel,pixel,1)))
model.add(Activation('linear'))
# take the max over all values in the activation map
model.add(MaxPooling2D(pool_size=(pixel,pixel)))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

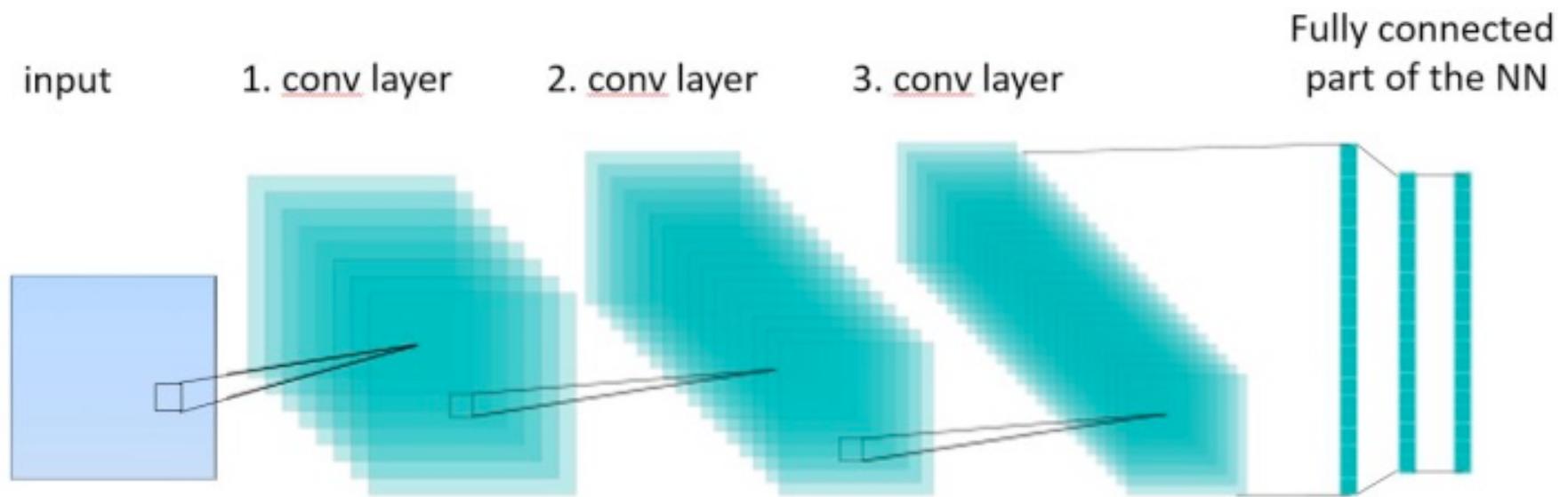
Convolution layer with a 1-channel input and 6 kernels



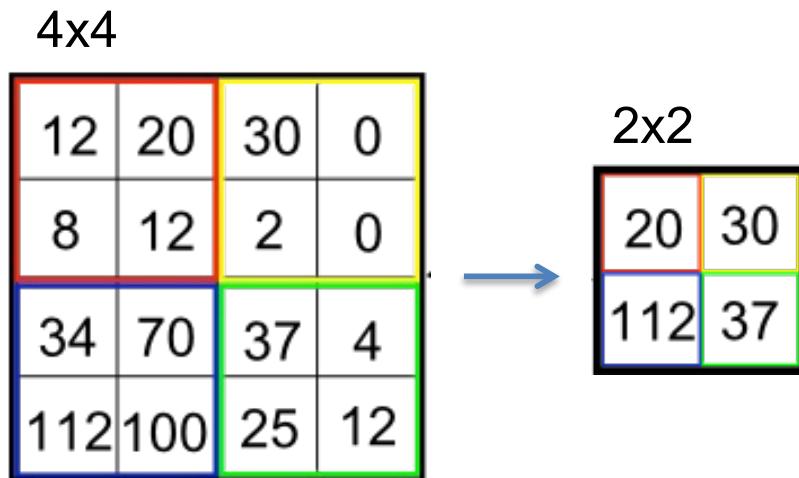
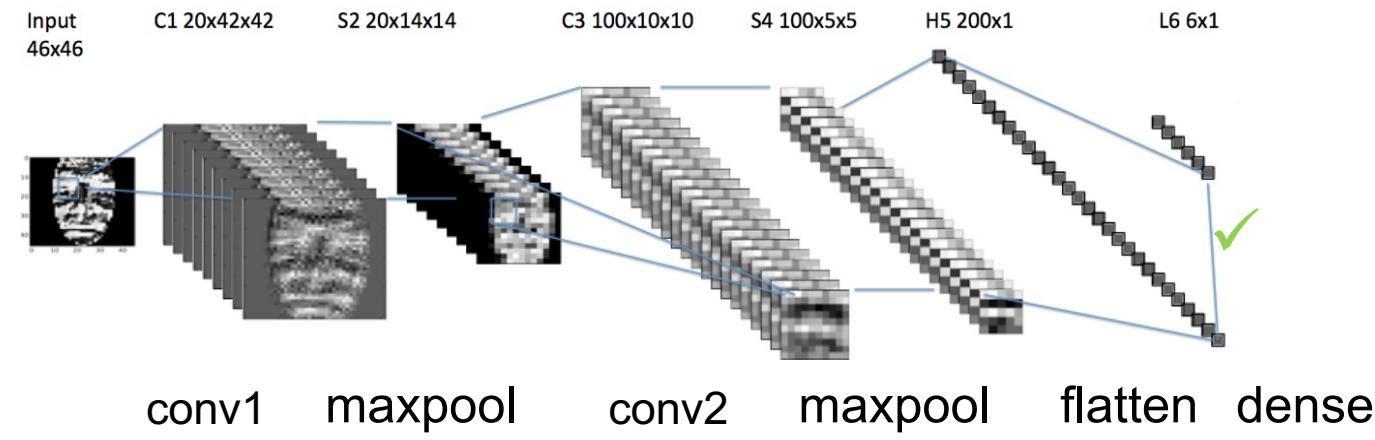
Convolution of the input image with 6 different kernels results in 6 activation maps.

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

A CNN with 3 convolution layers



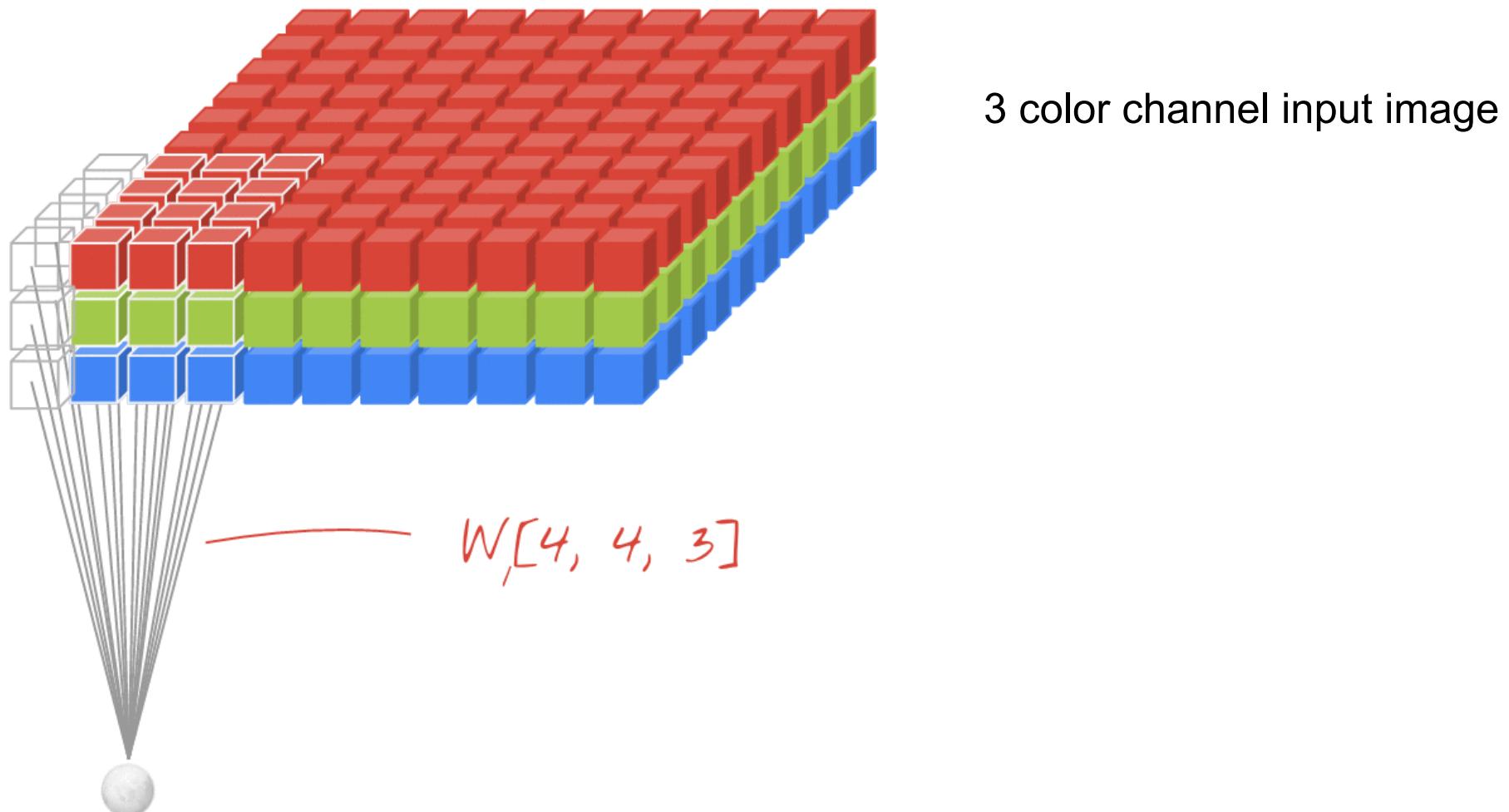
CNN ingredient II: Maxpooling Building Blocks reduce size



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“

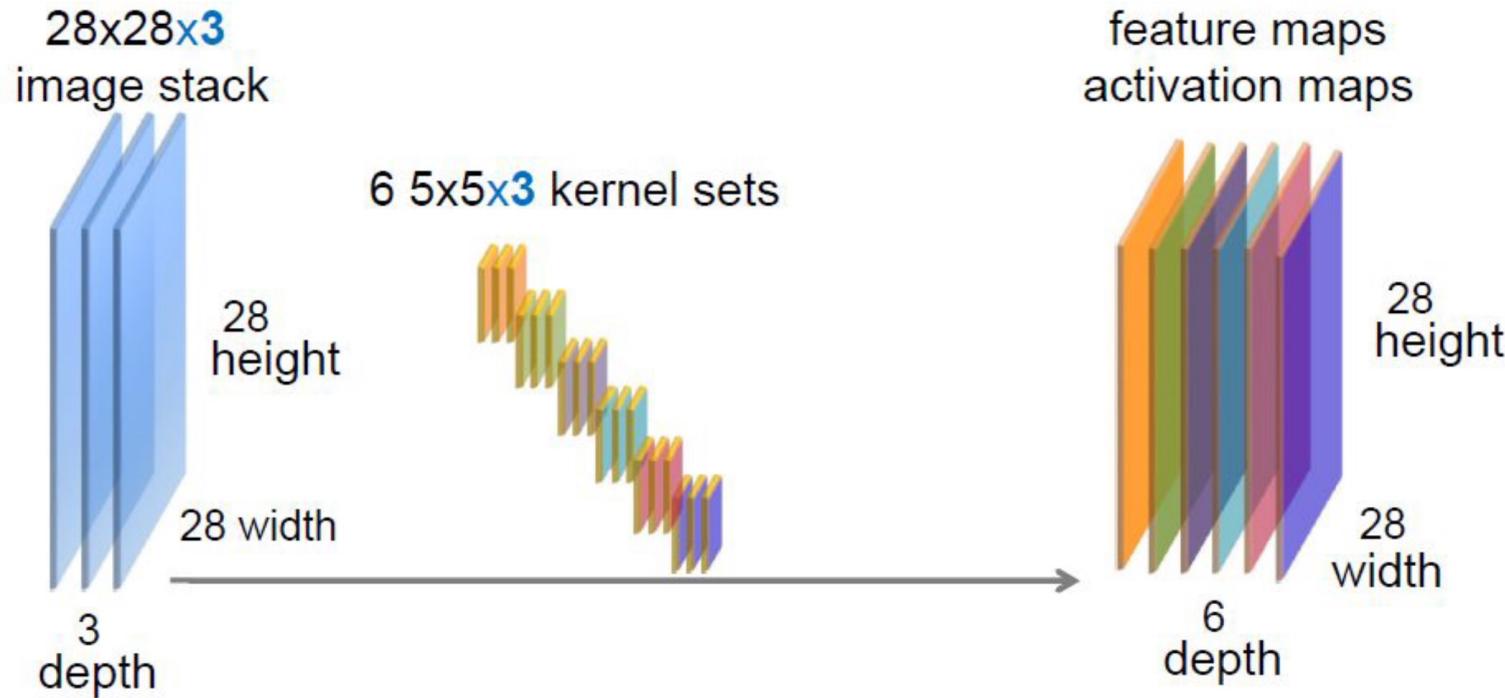
Animated convolution with 3 input channels



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 :

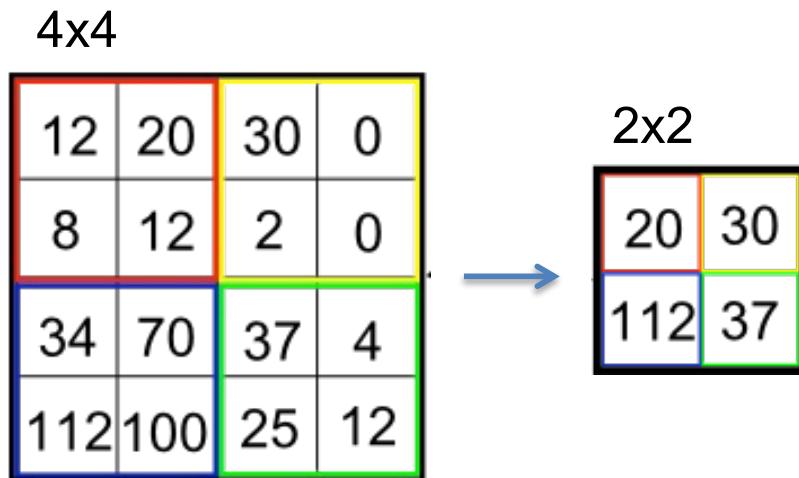
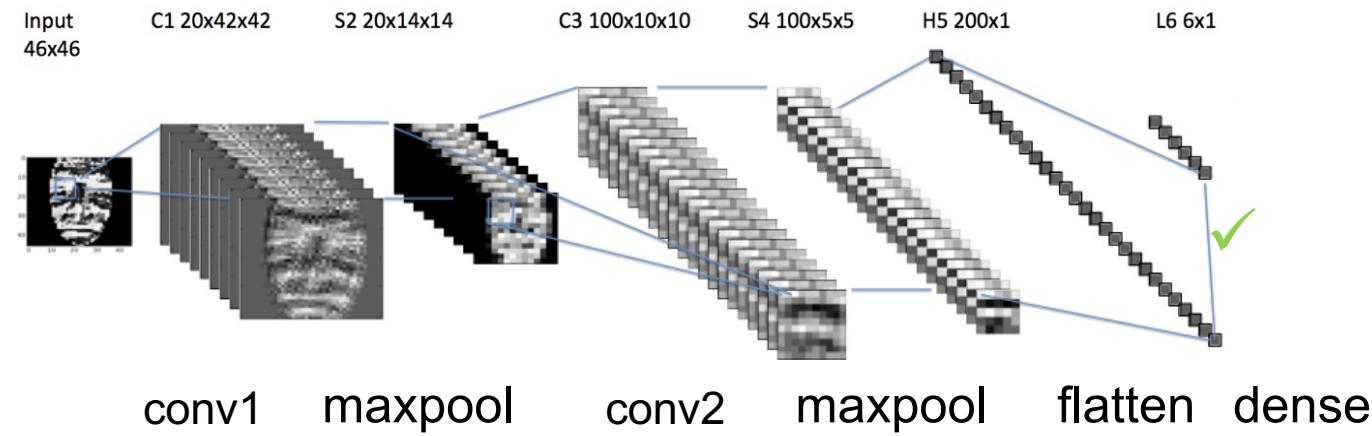
$$y_{jF_k} = f(z_{jF_k}) = f(b_k + \sum x_{ji} \cdot w_{ki})$$

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.

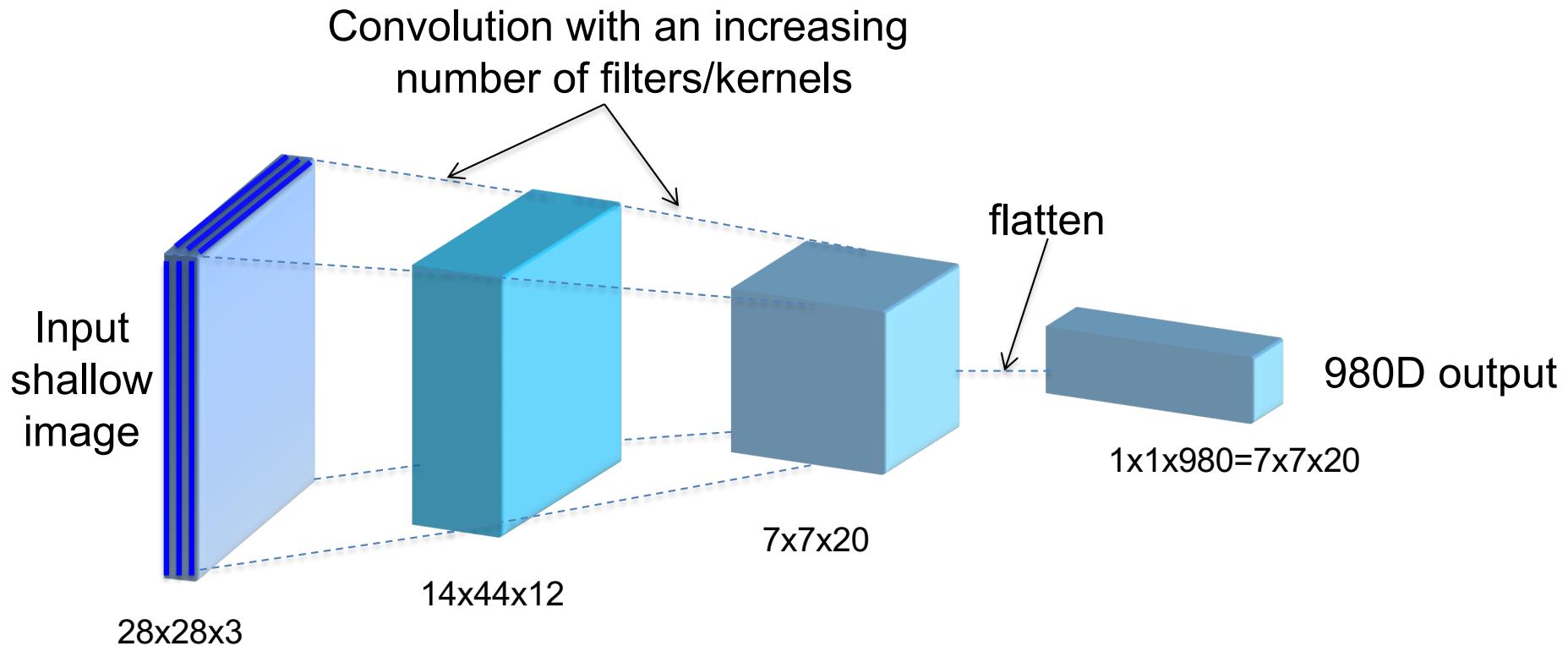
CNN ingredient II: Maxpooling Building Blocks reduce size



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

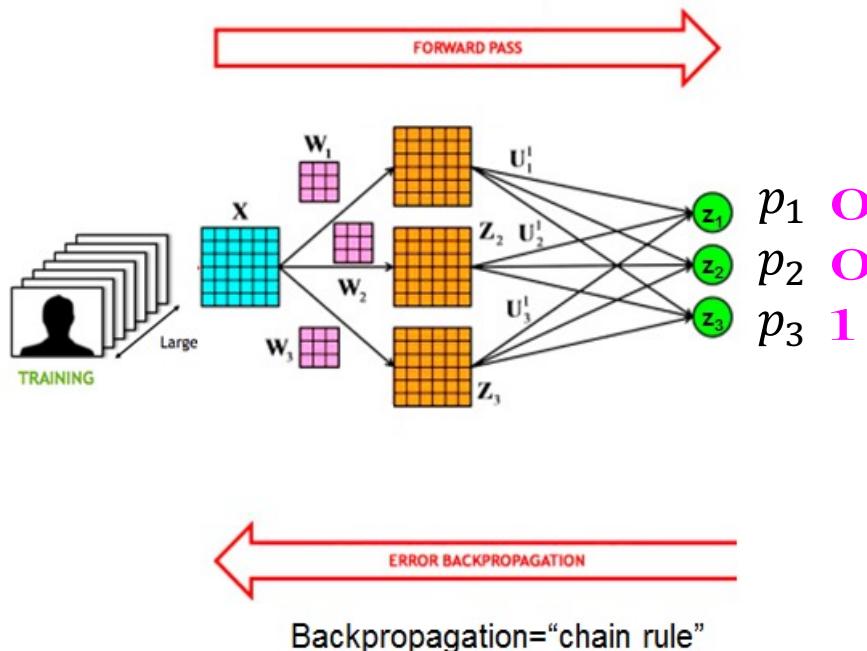
Hinton: „The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster“

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



Learning is done by weight updating:

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.

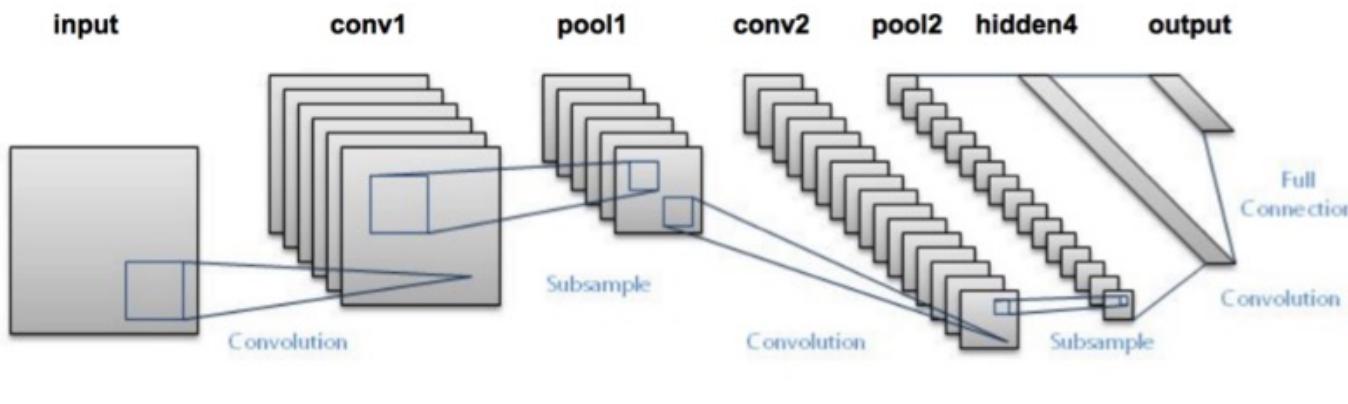
Minimize Loss-function:

$$L = \text{distance}(\text{observed}, \text{output}(w))$$

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

learning rate

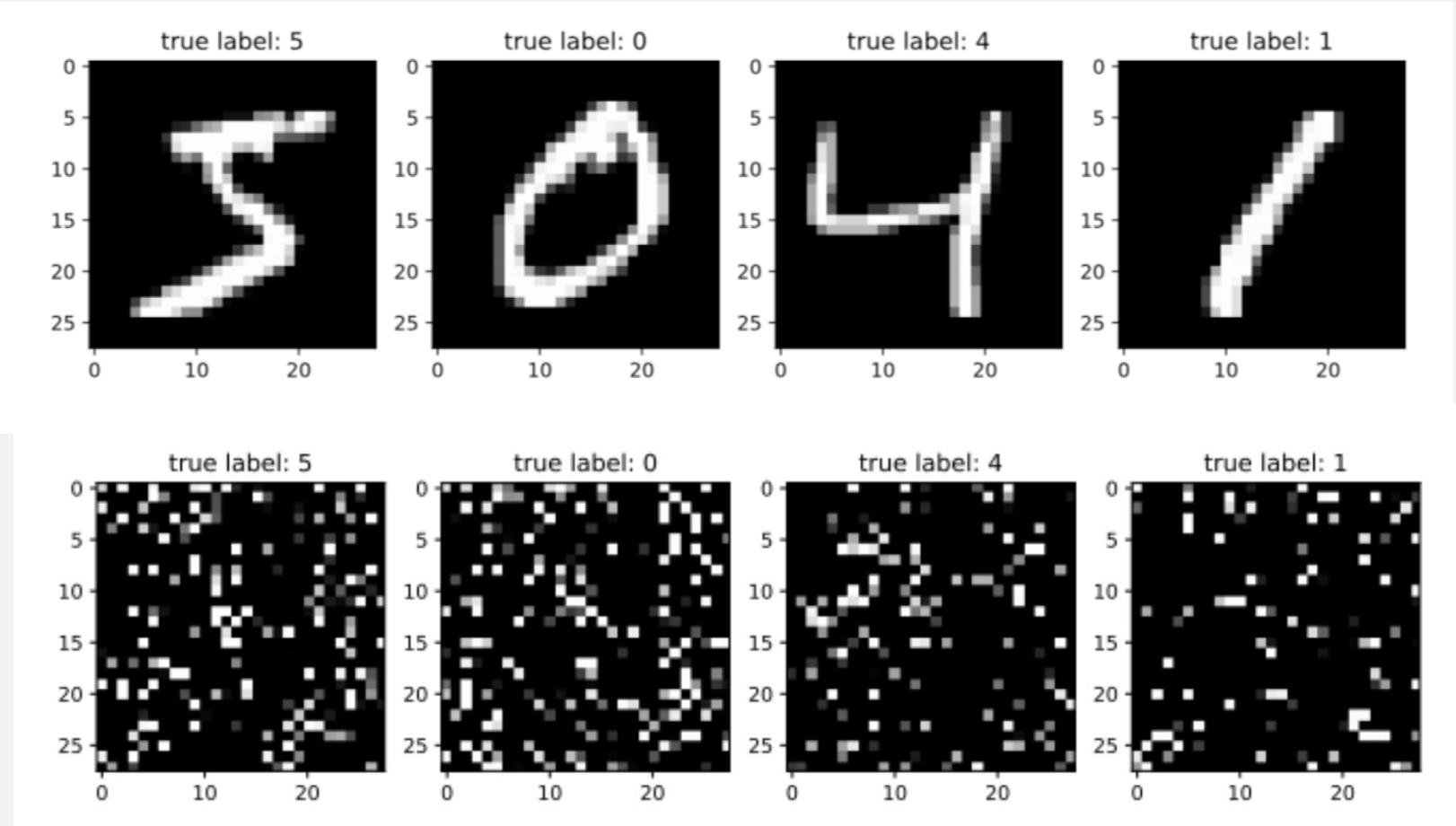
CNN for MNIST



```
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?



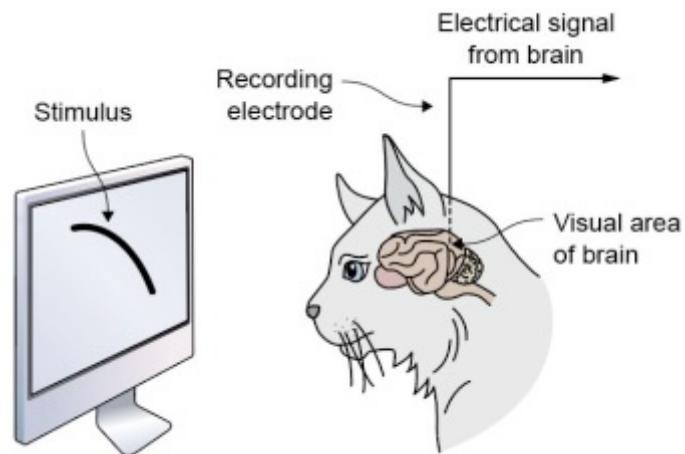
Open NB in: https://github.com/tensorchiefs/dl_course_2022/blob/master/notebooks/06_cnn_mnist_shuffled.ipynb

fcNN versus CNNs – some aspects

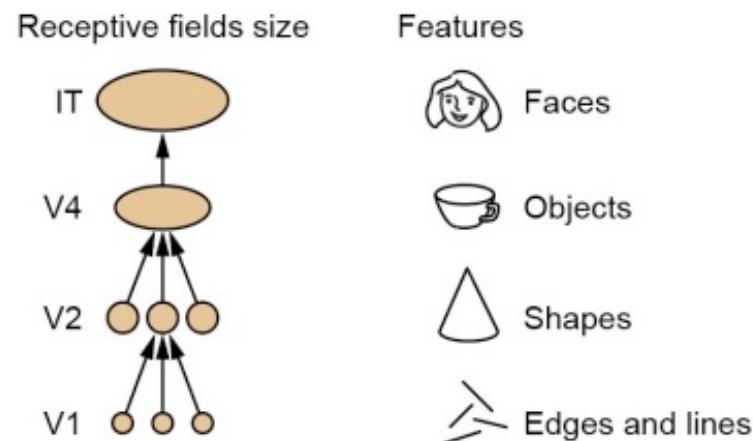
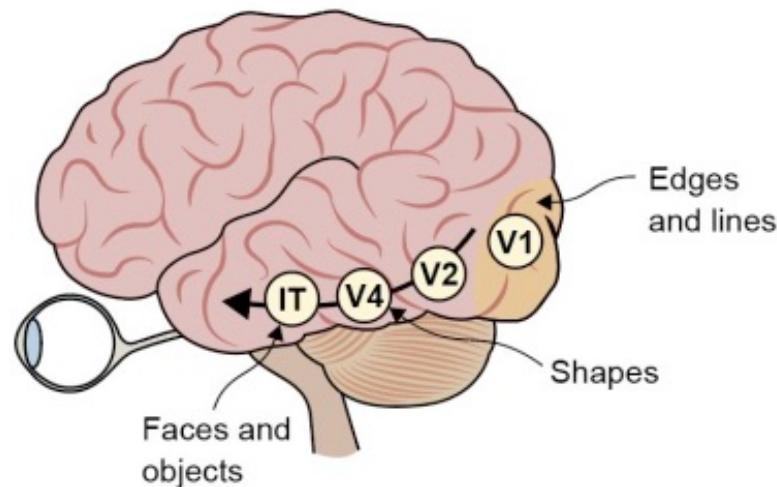
- A fcNN is good for tabular data, CNNs are good for ordered data (eg images).
- In a fcNN the order of the input does not matter, in CNN shuffling matters.
- A fcNN has no model bias, a CNN has the model bias that neighborhood matters.
- A node in one layer of a fcNN corresponds to one feature map in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

Biological Inspiration of CNNs

How does the brain respond to visually received stimuli?

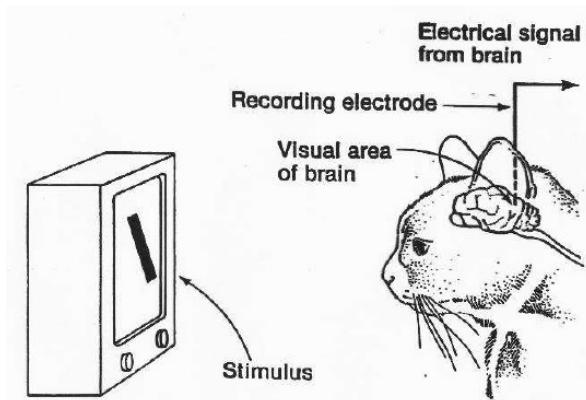


Setup of the experiment of Hubel and Wiesel in late 1950s in which they discovered **neurons** in the visual cortex that **responded** when moving **edges** were shown to the cat.

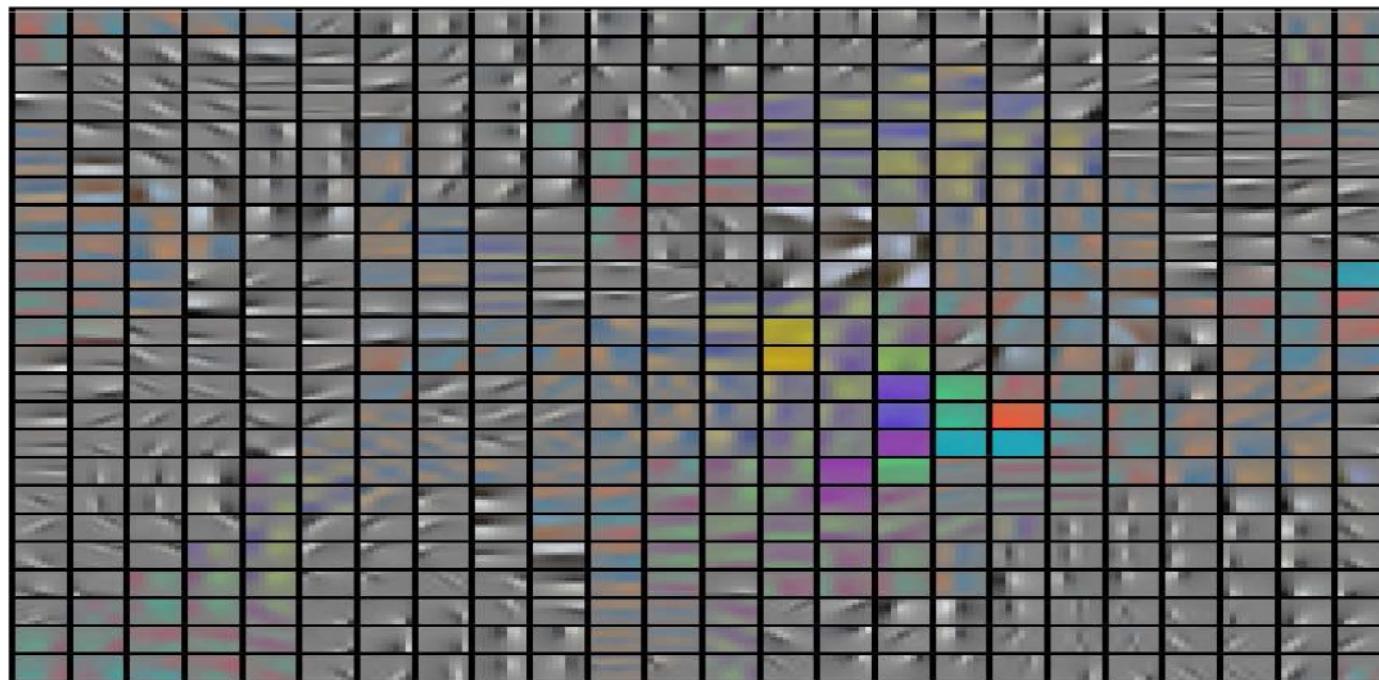
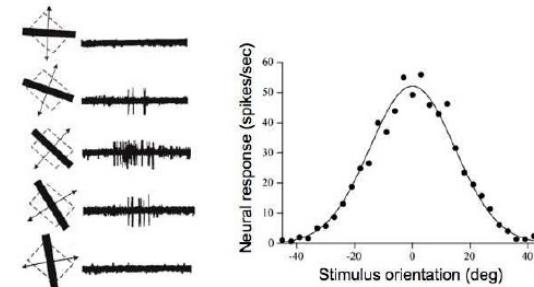


Organization of the visual cortex in a brain, where neurons in different regions respond to more and more complex stimuli

Compare neurons in brain region V1 in first layer of a CNN



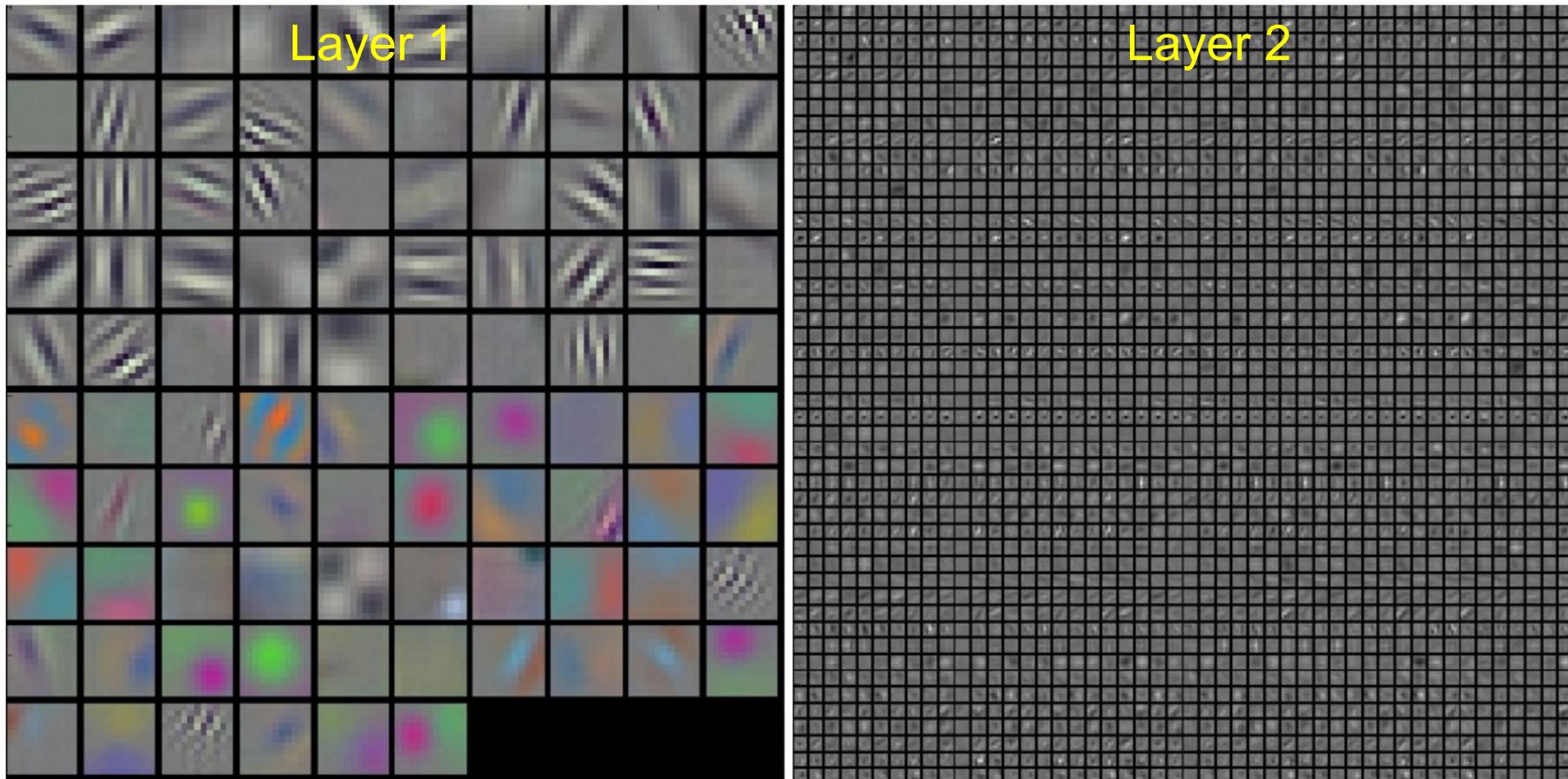
V1 physiology: orientation selectivity



Neurons in brain region V1 and neurons in 1. layer of a CNN respond to similar patterns

Visualize the weights used in filters

Filter weights from a trained Alex Net

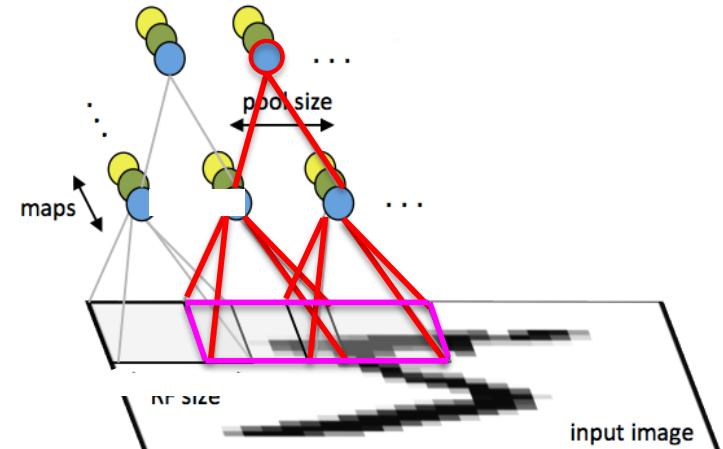
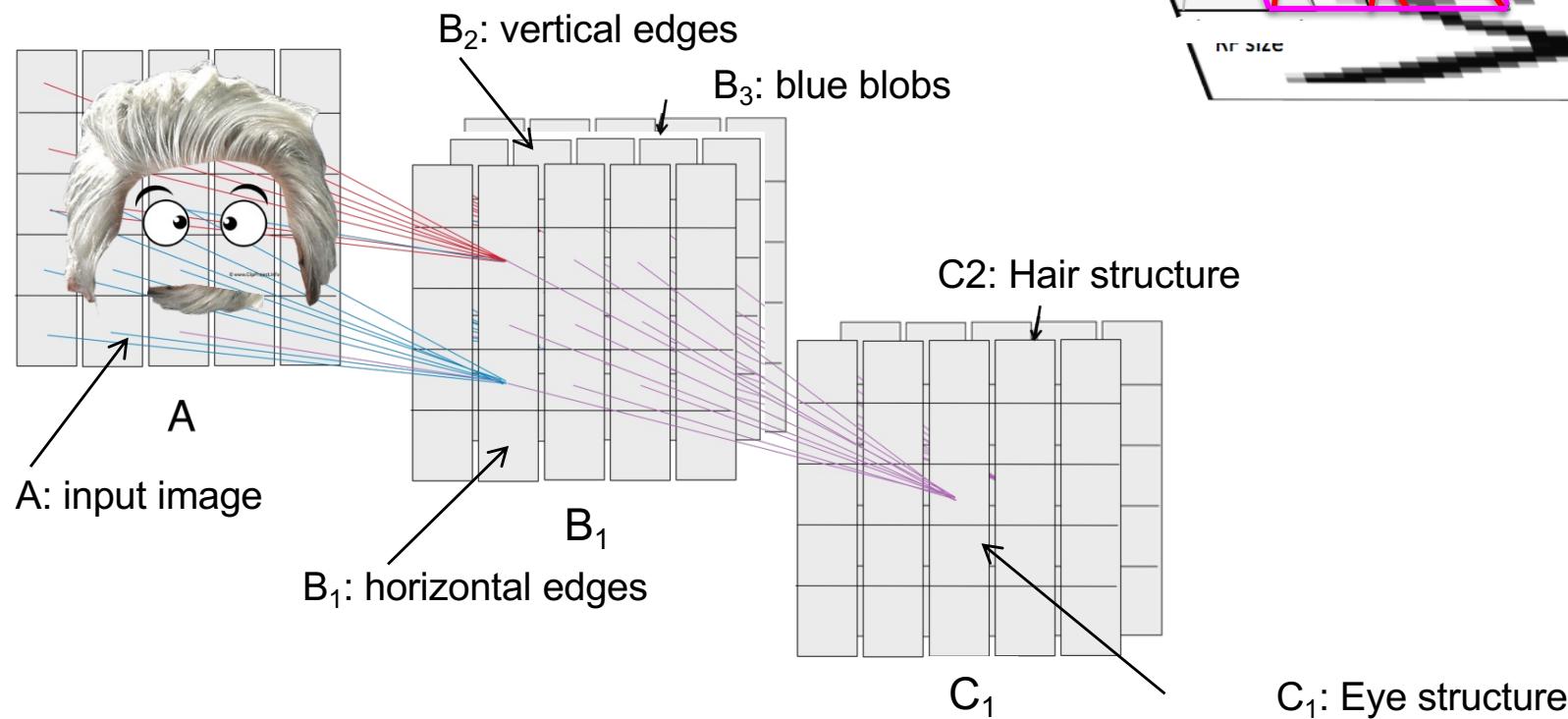


Only in layer 1 the filter pattern correspond to extracted patterns in the image.

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.

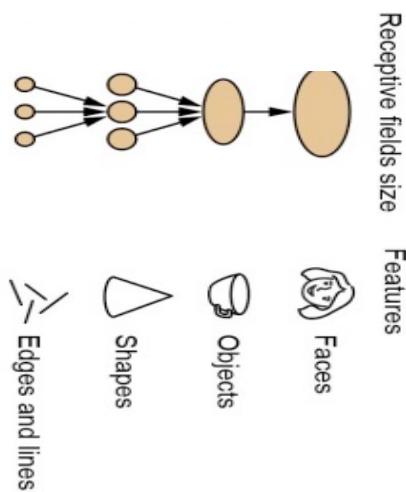
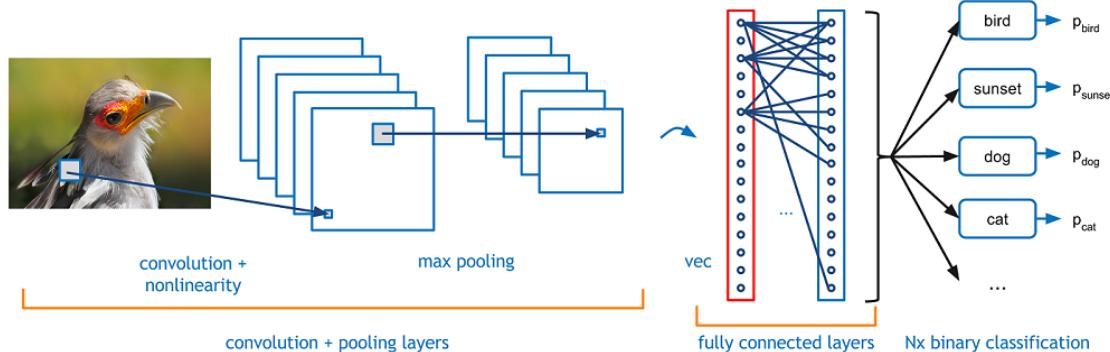
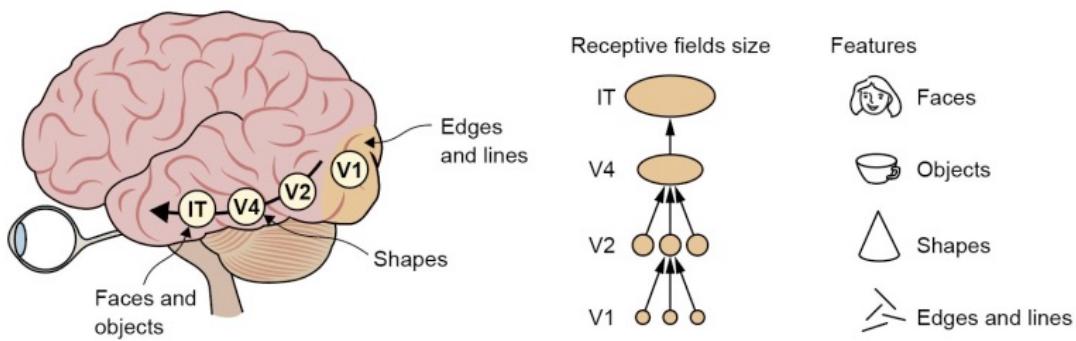
The receptive field

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. 71

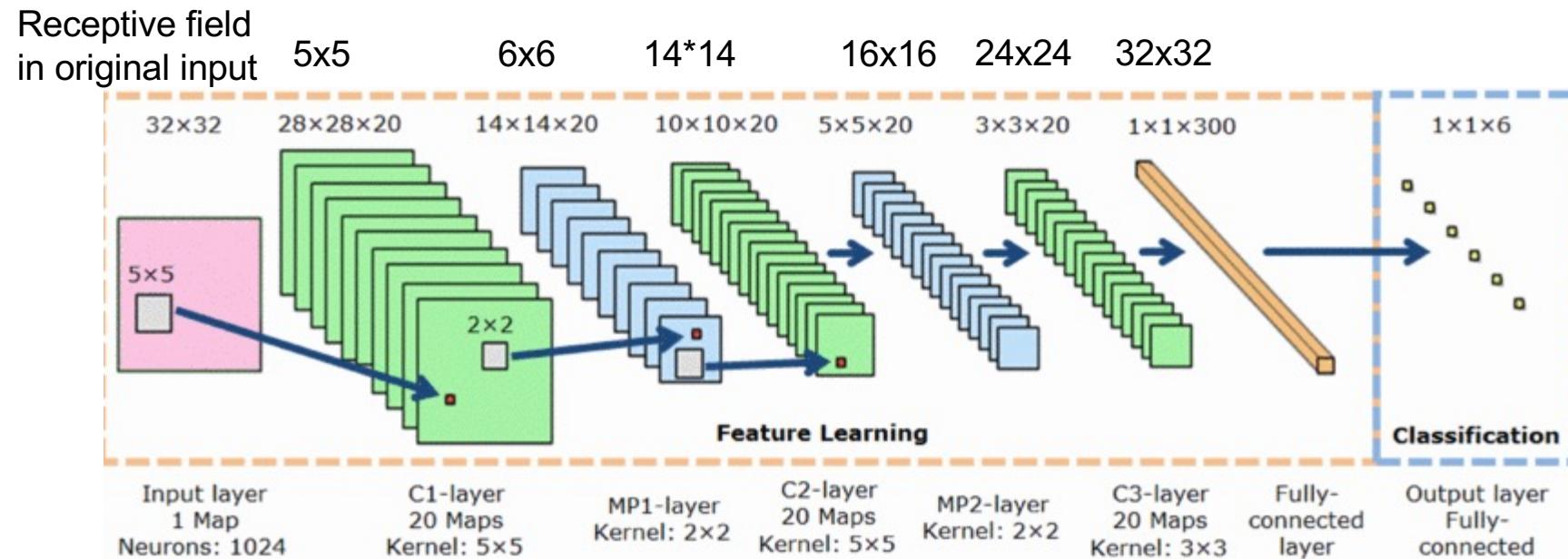
Weak analogies between brain and CNNs architecture



What input does activate a feature map in the CNN part or a neuron in the last layer?

The receptive field is growing from layer to layer

The receptive field of a neuron is the area in the original input image that impact the value of this neuron – “that can be seen by this neuron”.



Neurons from feature maps in higher layers have a larger receptive field than neurons sitting in feature maps closer to the input.

Code to determine size of receptive field: <http://stackoverflow.com/questions/35582521/how-to-calculate-receptive-field-size>

Visualize patches yielding high values in activation maps

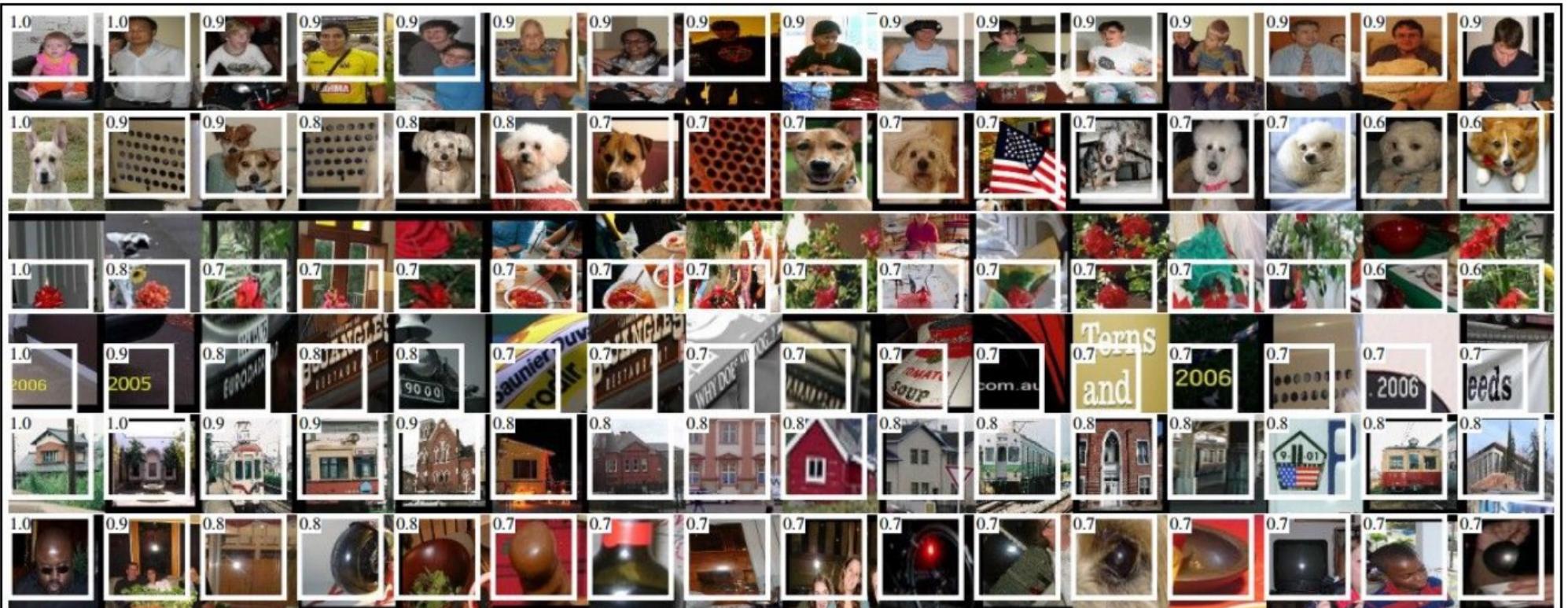
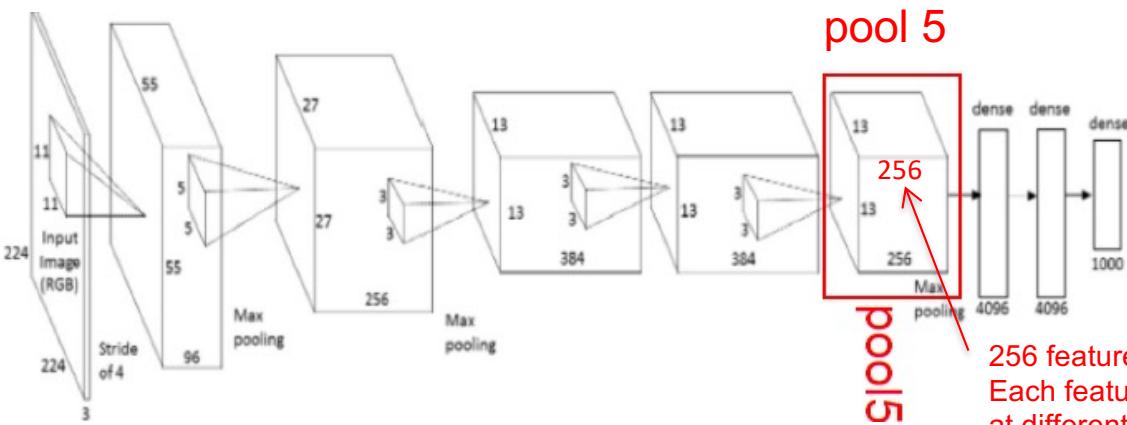


Figure 4: Top regions for six pool₅ units. Receptive fields and activation values are drawn in white. Some units are aligned to concepts, such as people (row 1) or text (4). Other units capture texture and material properties, such as dot arrays (2) and specular reflections (6).



<http://cs231n.github.io/understanding-cnn/>

Here we show image patches that activate maps in layer 5 most.

256 feature maps generated by 256 different 3x3 filters. Each feature map consists of equivalent neurons looking at different positions of the input volume.

What kind of image (patches) excites a certain neuron corresponding to a large activation in a feature map?

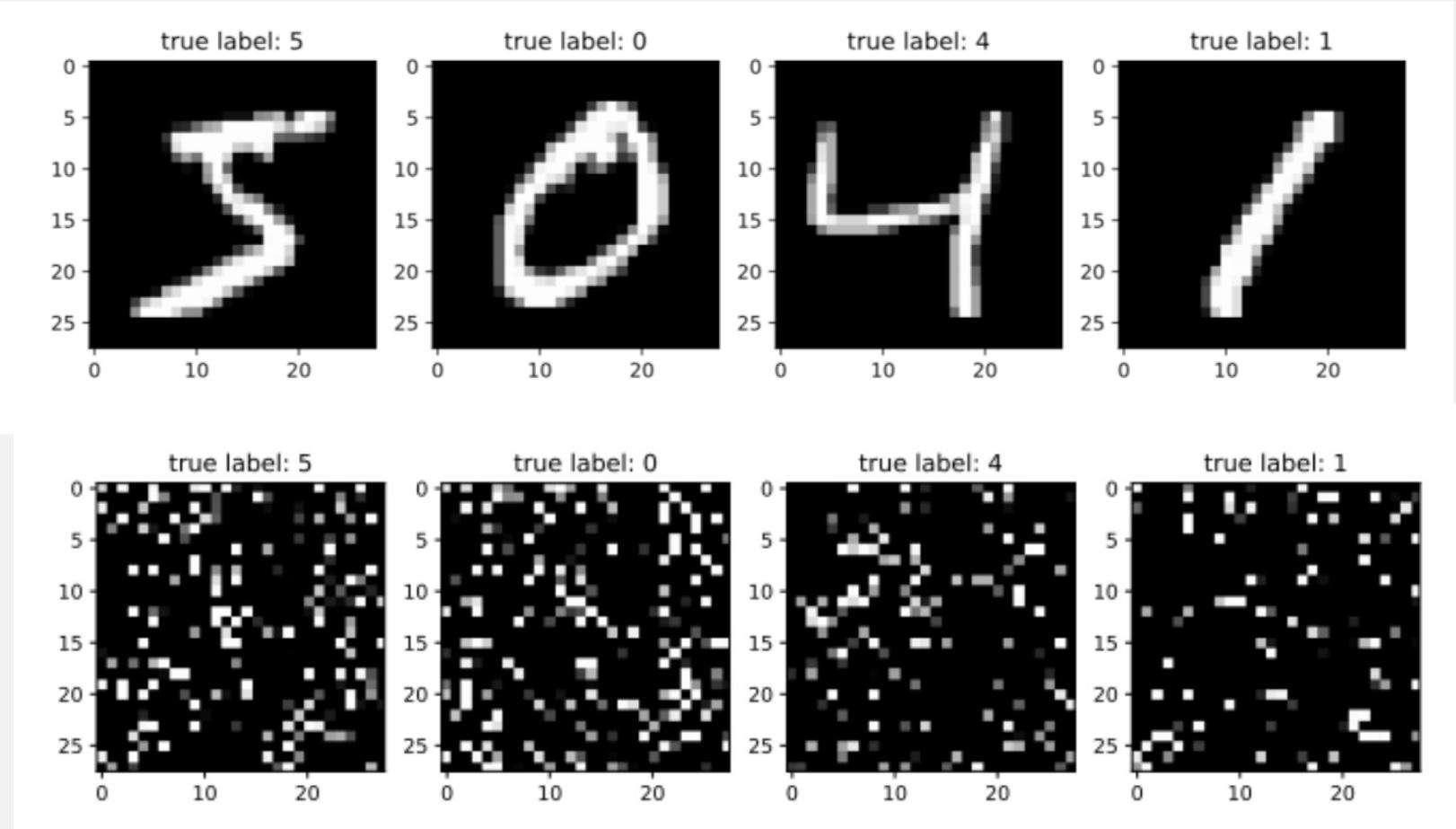
10 images from data set leading to high signals 6 feature maps of **conv6**



10 images from data set leading to high signals 6 feature maps of **conv9**



Ufzgi: Does shuffling disturb a fcNN or CNN?



Inverstigate if shuffling disturbs the fcNN for MNIST:

https://github.com/tensorchiefs/dl_course_2022/blob/master/notebooks/04_fcnn_mnist_shuffled.ipynb

Inverstigate if shuffling disturbs the CNN for MNIST

https://github.com/tensorchiefs/dl_course_2022/blob/master/notebooks/06_cnn_mnist_shuffled.ipynb

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.
- 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