Machine Intelligence:: Deep Learning

Beate Sick, Lilach Goren, Pascal Bühler

Institut für Datenanalyse und Prozessdesign

Zürcher Hochschule für Angewandte Wissenschaften

Most of the slides were developed by Oliver Dürr & Beate Sick

Outline of the DL Module (tentative)

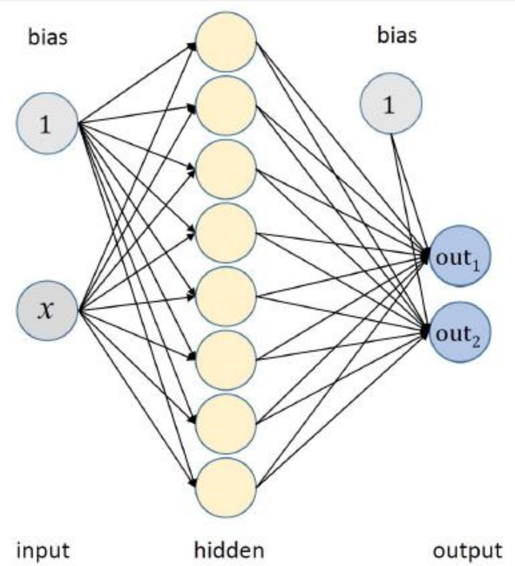
The course is split in 8 sessions, each 4 lectures long. Topics might be adapted during the course

Day	Date	Time	Topic
1	15.04.2025	09:00-12:30	Introduction to Deep Learning & Keras, first NNs
-	21.04.2025	-	FRÜHLINGS-FERIEN
-	28.04.2025	-	FRÜHLINGS-FERIEN
2	06.05.2025	09:00-12:30	Loss, Optimization, Regression, Classification
3	13.05.2025	09:00-12:30	Computer vision, CNN-archictecture
4	20.05.2025	09:00-12:30	DL in practice, pretrained (foundation) models
5	27.05.2025	09:00-12:30	Model evaluation, baselines, xAI, troubleshooting
6	03.06.2025	09:00-12:30	Generative Models, Transformer-architecture
7	10.06.2025	09:00-12:30	Vision Transformer
8	17.06.2025	09:00-12:30	Projects, deep Ensembling

Look back on homework:
define and use a custom loss function

Notebook: [02_custom_loss.ipynb](https://github.com/tensorchiefs/dl_course_2025/blob/main/notebooks/02_custom_loss.ipynb)

https://github.com/tensorchiefs/dl_course_2025/blob/main/notebooks/02_custom_loss.ipynb



$$(Y|x) \sim N(\mu(x), \sigma(x))$$

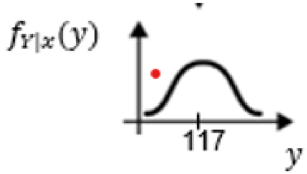
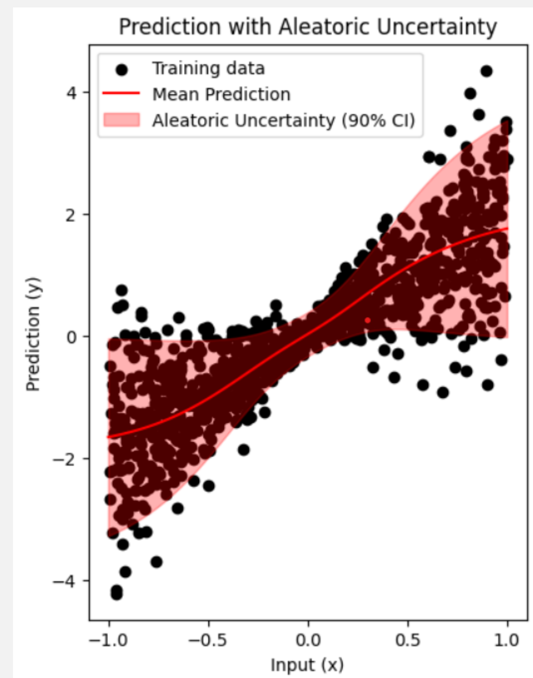
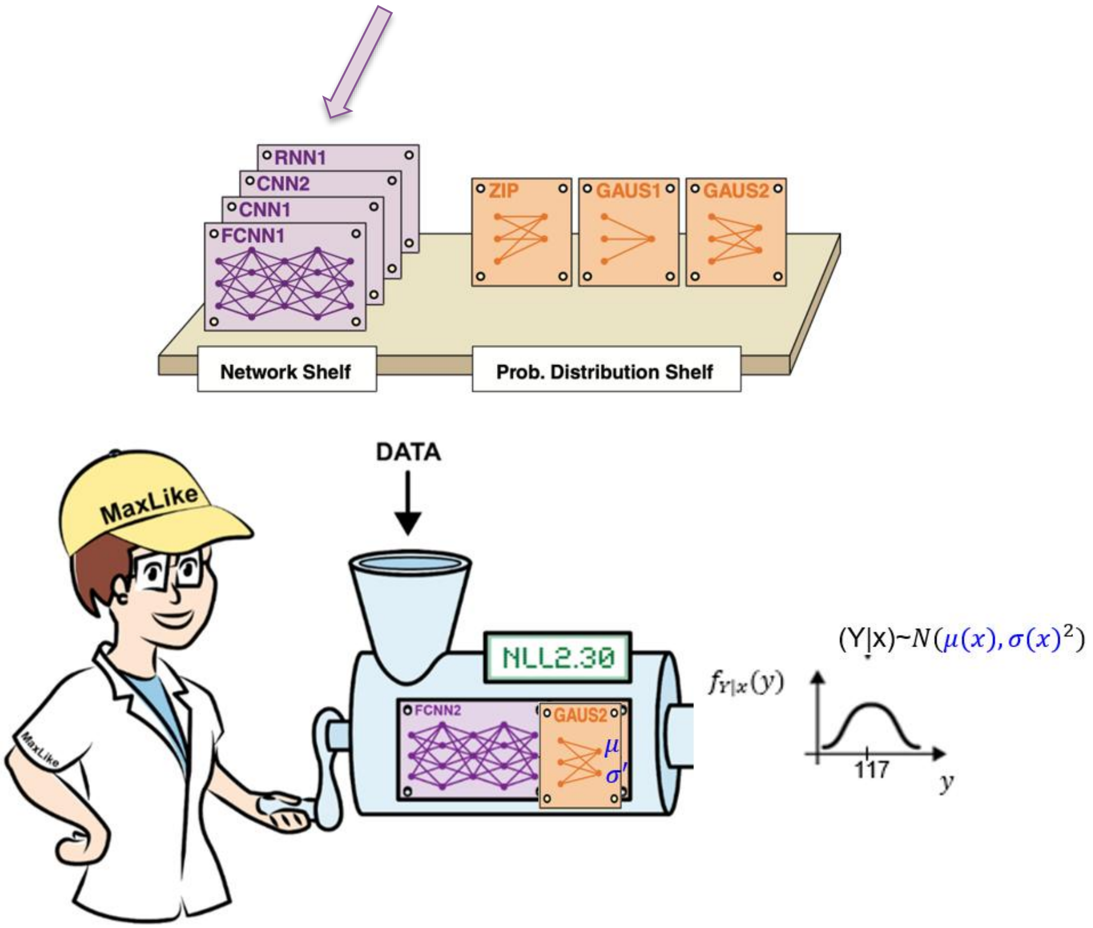


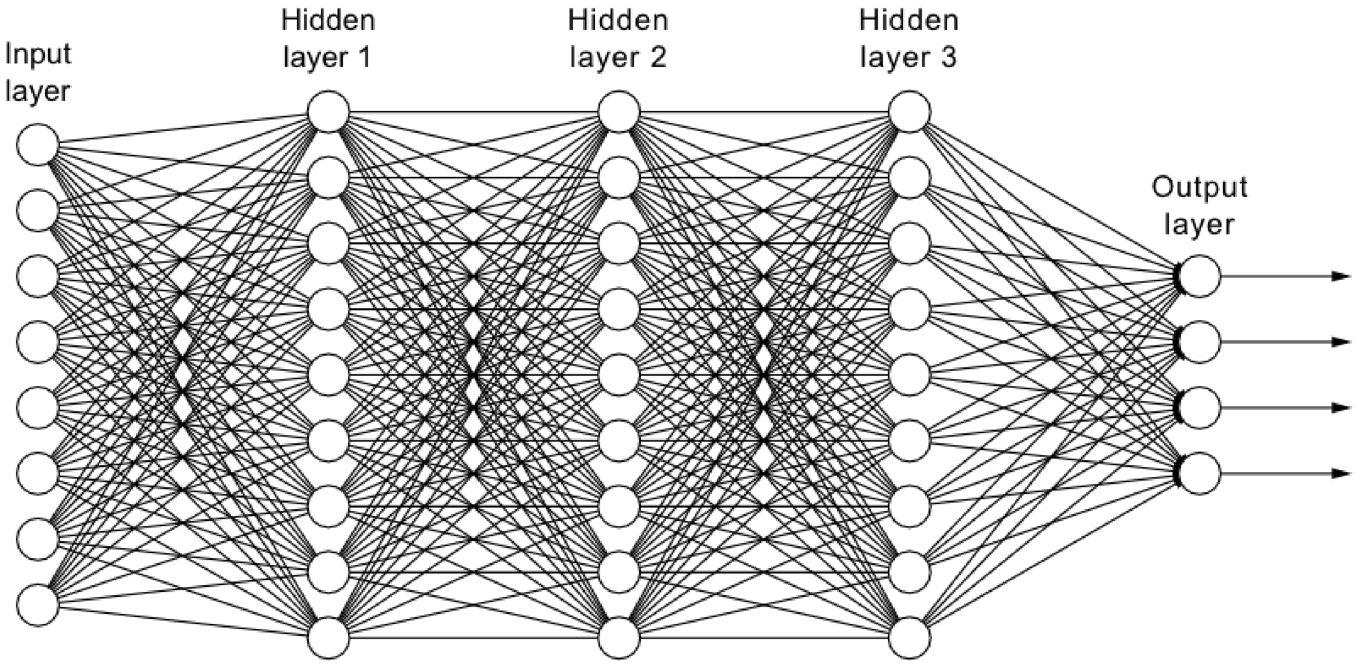
Figure 4.19 A NN with two output nodes can be used to control the parameter μ_x and σ_x of the conditional outcome distribution $N(\mu_x, \sigma_x)$ for regression tasks with non-constant variance



NN architectures

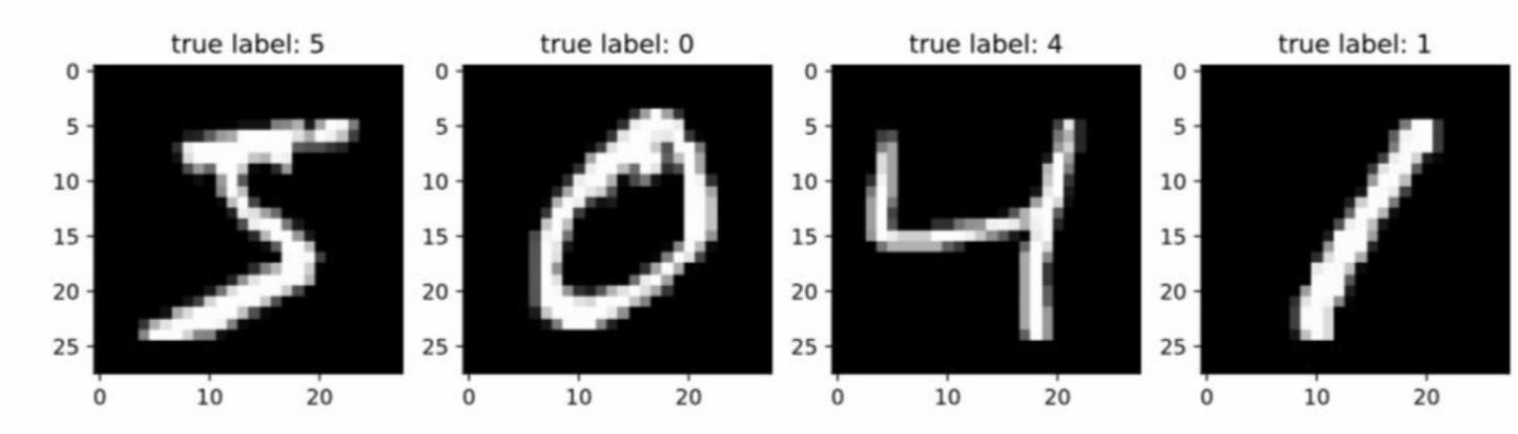


A fully connected NN

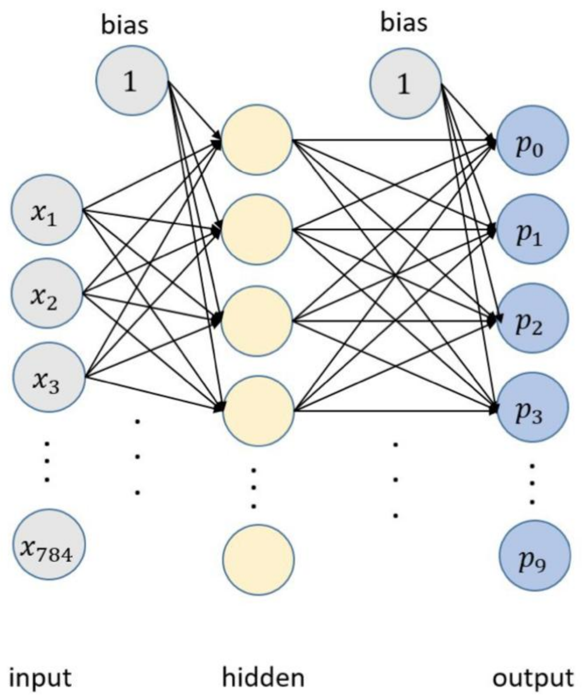


Probabilistic classification

- Usually in DL the model predicts a probability for each possible class
- Example:
  - Banknote from exercise – classes: “real” or “fake”
  - Typical example Number from hand-written digit – classes: 0, 1, …, 9



Classification: Softmax Activation



$p_o, p_1 \ldots p_9$ are probabilities for the classes 0 to 9.

Activation of last layer z_i incomming

$$p_i = \frac{e^{z_i}}{\sum_{j=0}^{9} e^{z_j}}$$

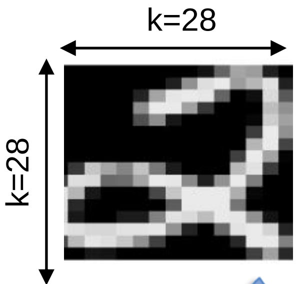
Makes outcome positive

Ensures that pi's sum up to one

This activation is called softmax

Figure 2.12: A fully connected NN with 2 hidden layers. For the MNIST example, the input layer has 784 values for the 28 x 28 pixels and the output layer out of 10 nodes for the 10 classes.

NLL for multi-class = cross-entropy

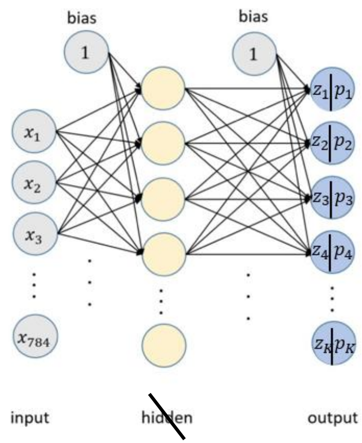


Flatten image$_i$

Input $\boldsymbol{x}_i$

24.0
–5.1
12.2
89.9
:
3.2

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

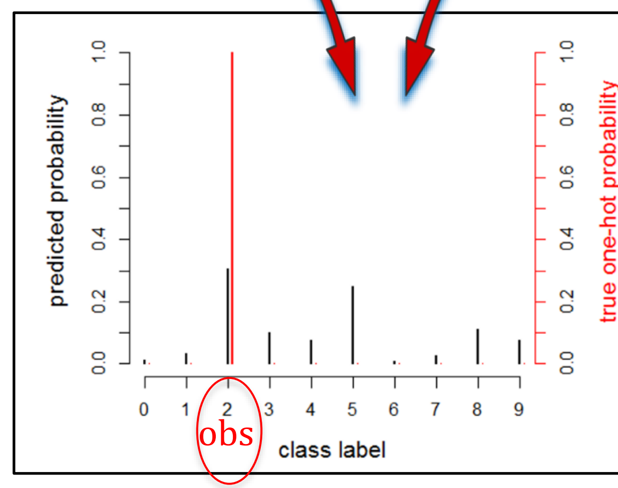


Score or logit $\boldsymbol{z}_i$

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

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

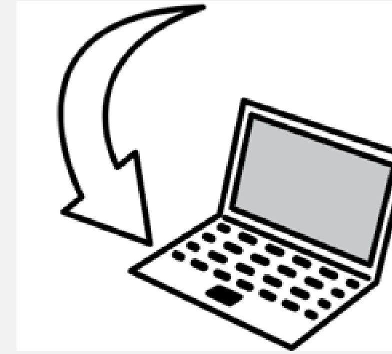
probability $\boldsymbol{p}_i$	1-hot labels $\boldsymbol{y}_i$	index
0.02	0	0
0.04	0	1
0.31	1	2
0.10	0	3
0.08	0	4
0.26	0	5
0.01	0	6
0.03	0	7
0.11	0	8
0.08	0	9

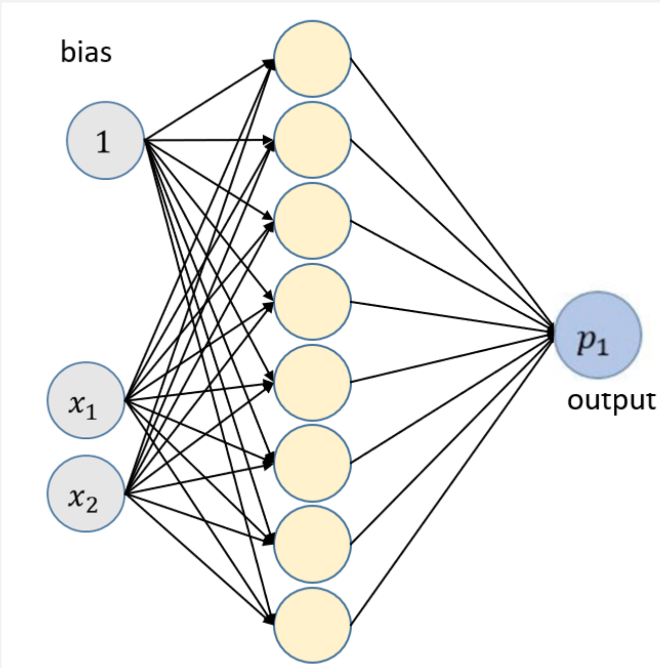


$$\text{loss} = \text{NLL} = \frac{1}{N_{\text{batch}}} \sum_{i=1}^{N_{\text{batch}}} l_i \quad \text{, with} \quad l_i = -\sum_{k=1}^{10} {}^{obs}y_{ik} \cdot \log(p_{ik})$$

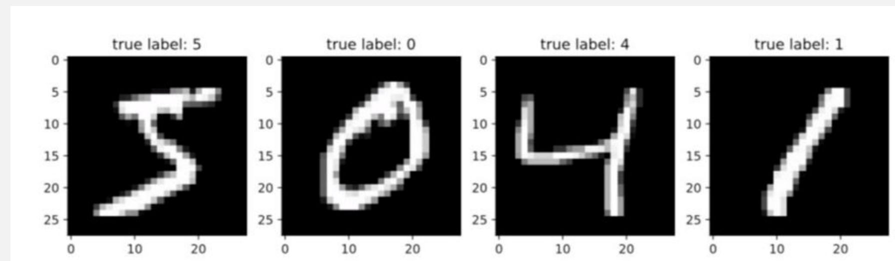
Loss = NLL= cross-entropy ($-\sum p_i^{\text{obs}} \log p_i^{\text{pred}}$) averaged over all images in mini-batch

Exercise:





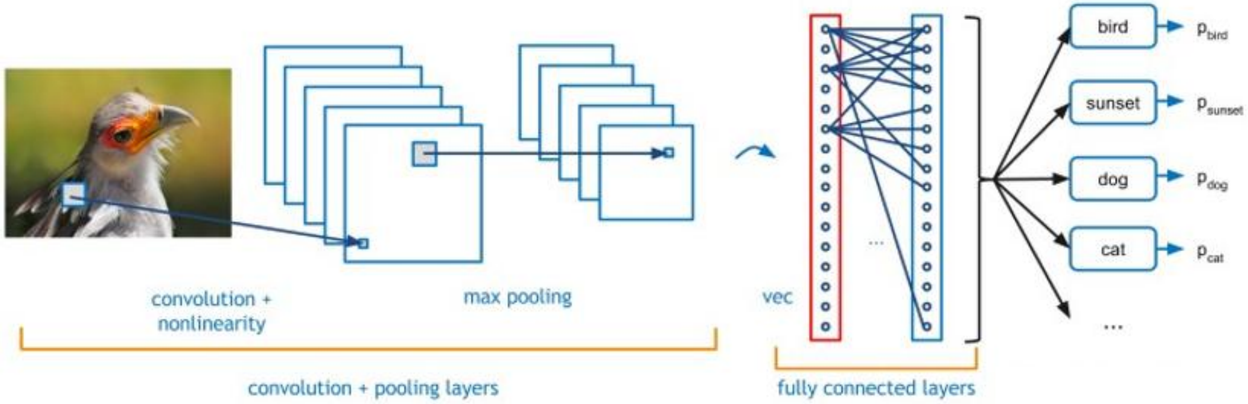
input hidden



Use a fully connected NN to classify MNIST data to 10 possible classes

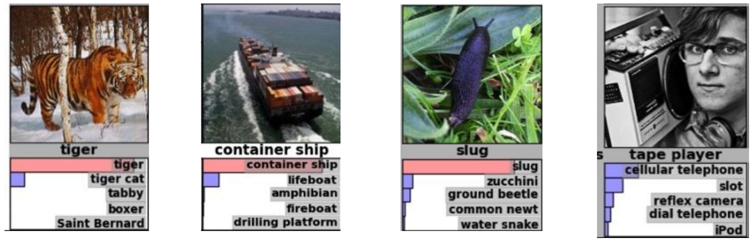
[03_fcnn_mnist_keras_torch.ipynb]()

Convolutional Neural Networks for image data

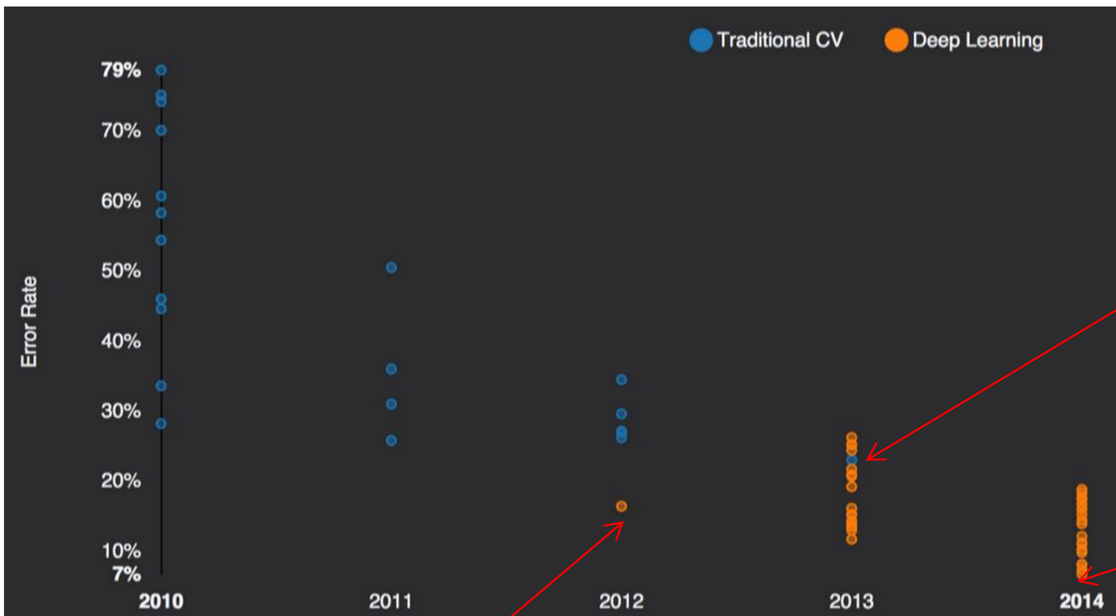


The first DL breackthrough: Imagenet challenge

1000 classes
1 Mio samples



...



Human: 5% misclassification

Only one non-CNN approach in 2013

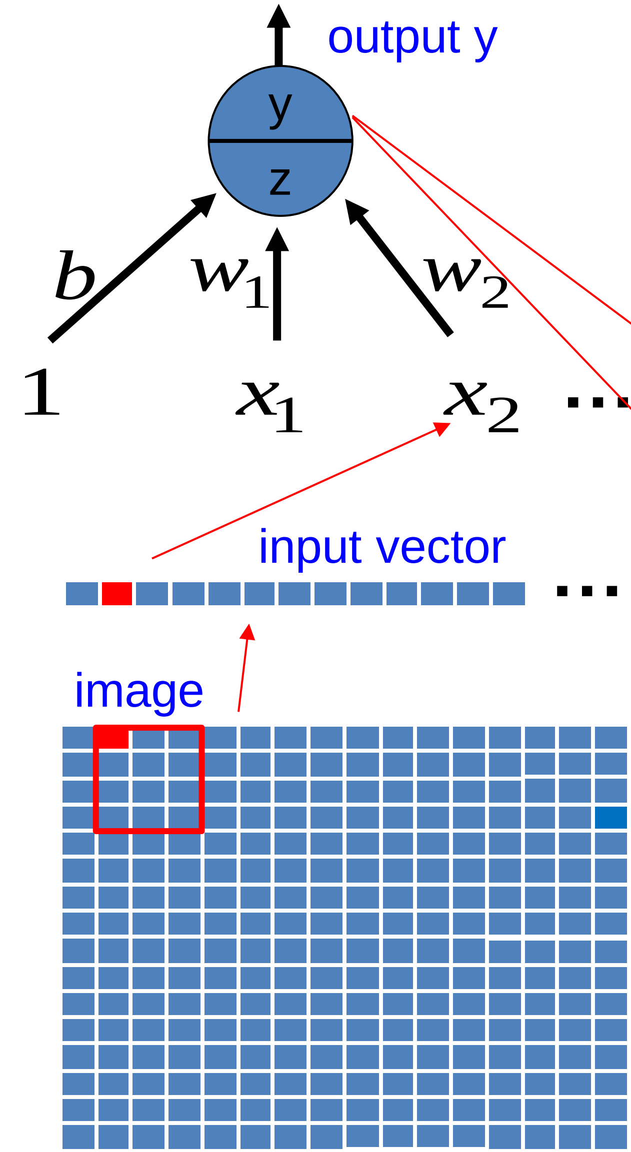
GoogLeNet 6.7%

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

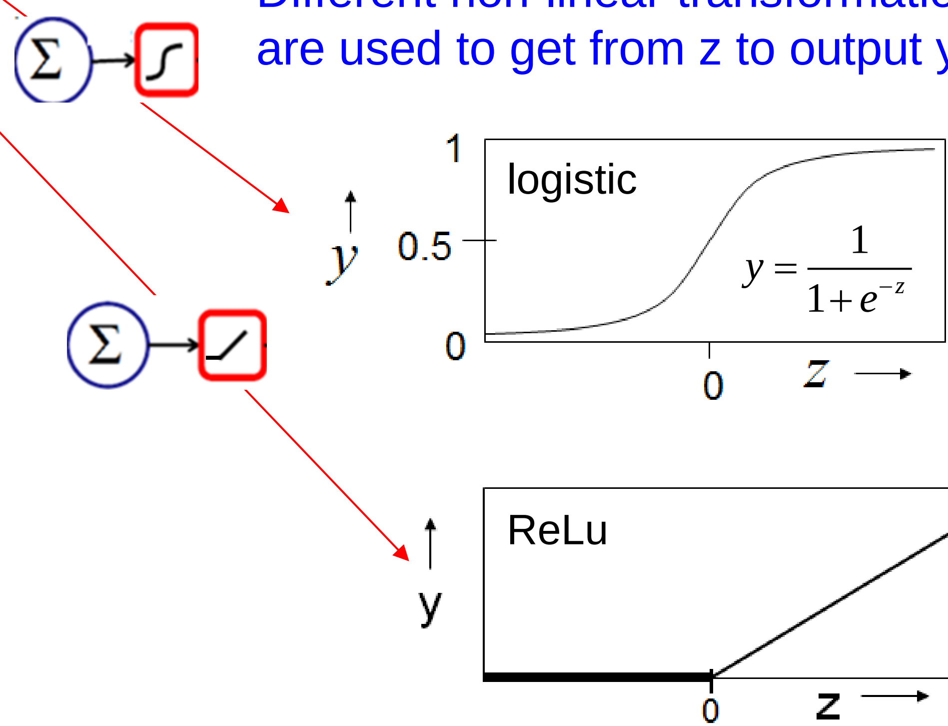
Figure: https://medium.com/global-silicon-valley/machine-learning-yesterday-today-tomorrow-3d3023c7b519

An artificial neuron

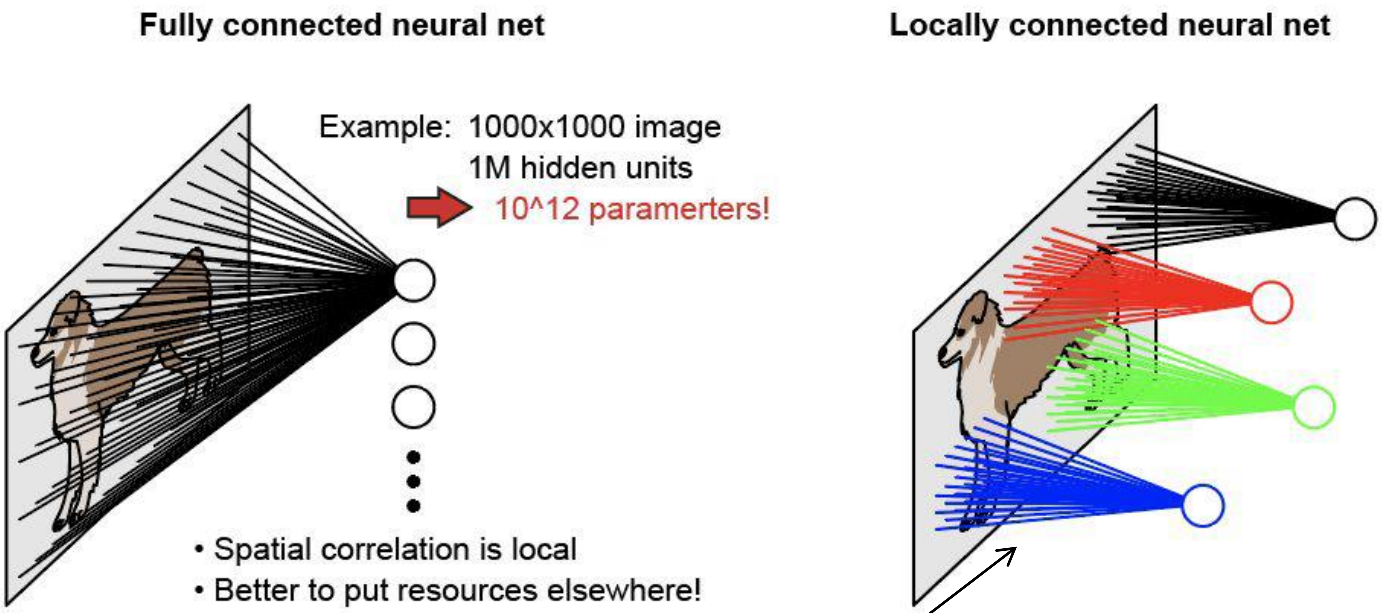
$$z = \overset{\text{bias}}{b} + \sum_i x_i \overset{\text{weights}}{w_i}$$



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



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.

Example of designed Kernel / Filter

Input

Identity

0	0	0
0	1	0
0	0	0

Right Sobel

-1	0	1
-2	0	2
-1	0	1

Left Sobel

-1	0	1
-2	0	2
-1	0	1

https://setosa.io/ev/image-kernels/ upload your own image

Convolution

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input X

1	0	1
0	1	0
1	0	1

Kernel W

4	3	4
2	4	3
2	3	4

Result Z

Convolution (let's ignore bias b):

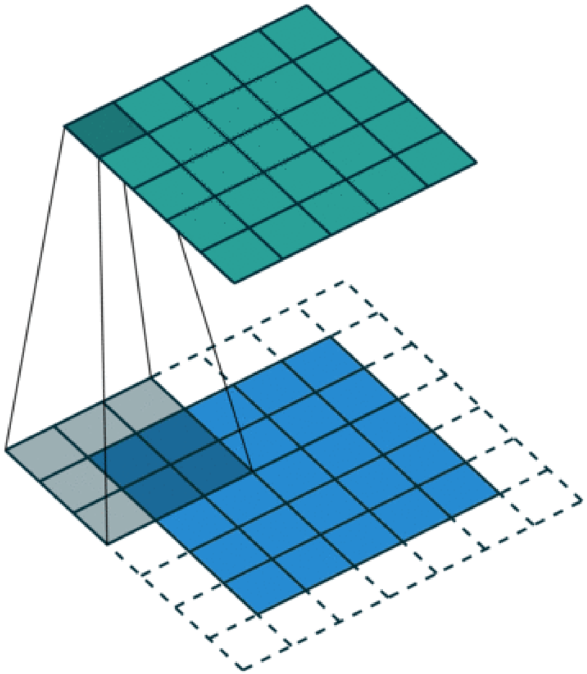
$$z = b + \sum_i x_i w_i$$

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

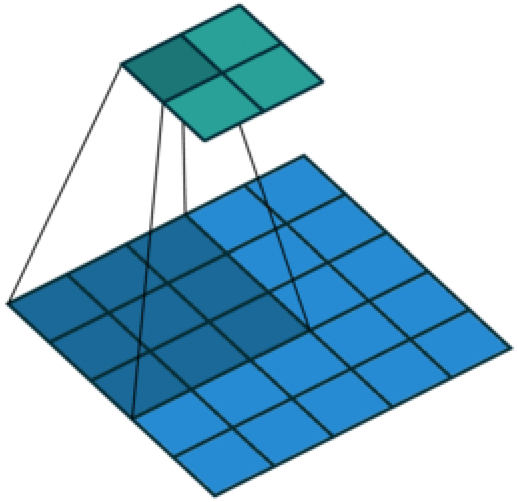
4		

https://medium.com/data-science/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

CNN Ingredient I: Convolution



Zero-padding to achieve
same size of feature and input



no padding to only use
valid input information

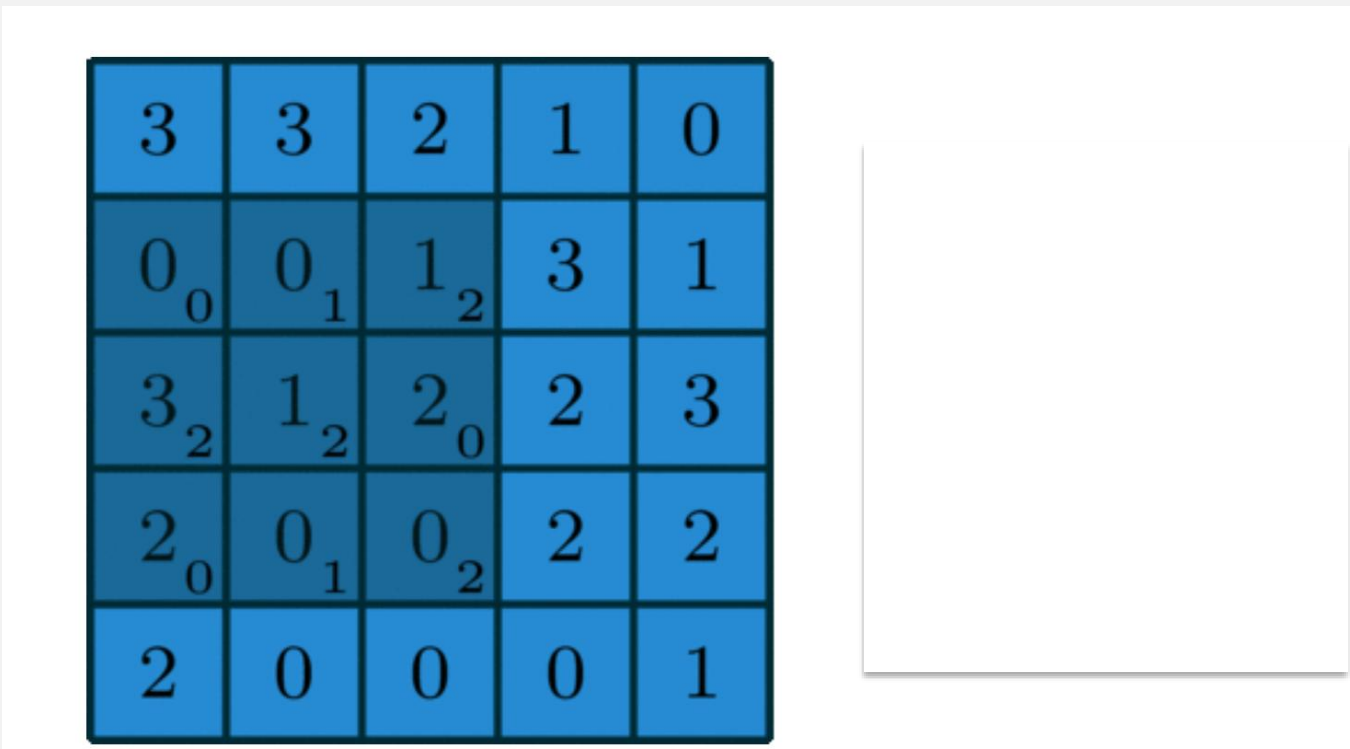
The *same* weights are used at each position of the input image.

Image from https://arxiv.org/pdf/1603.07285

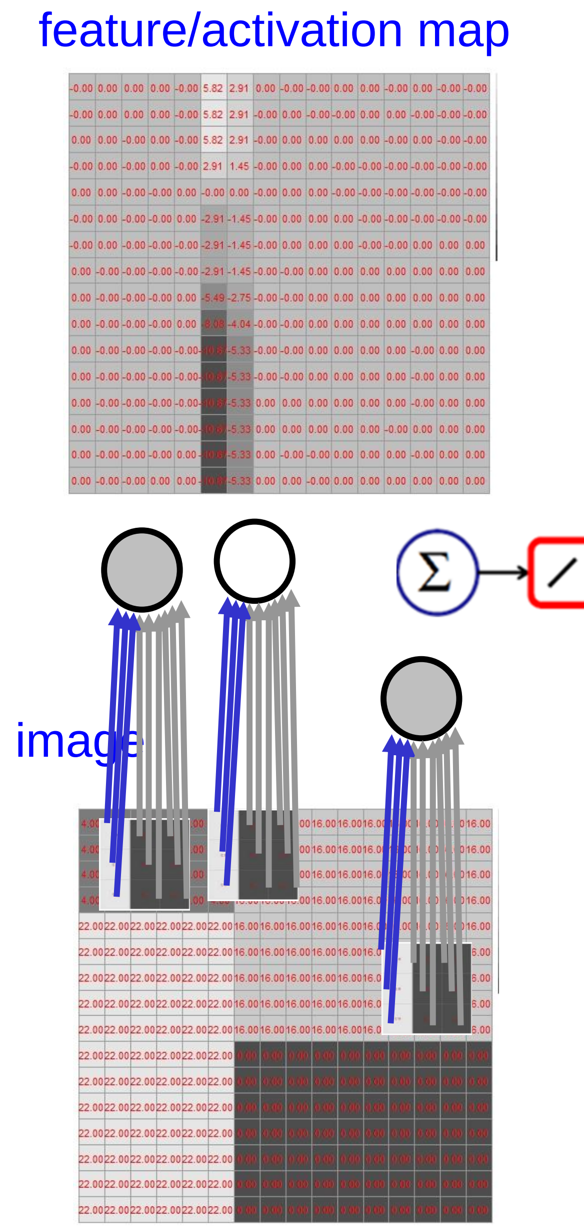
Exercise: Do one convolution step by hand

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

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



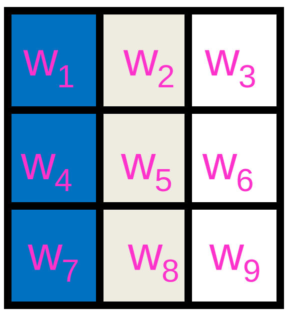
Convolutional networks use neighborhood information and replicated local feature extraction

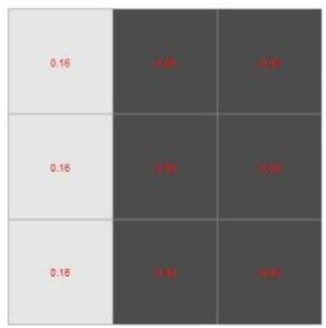


In a locally connected network the calculation rule

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

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

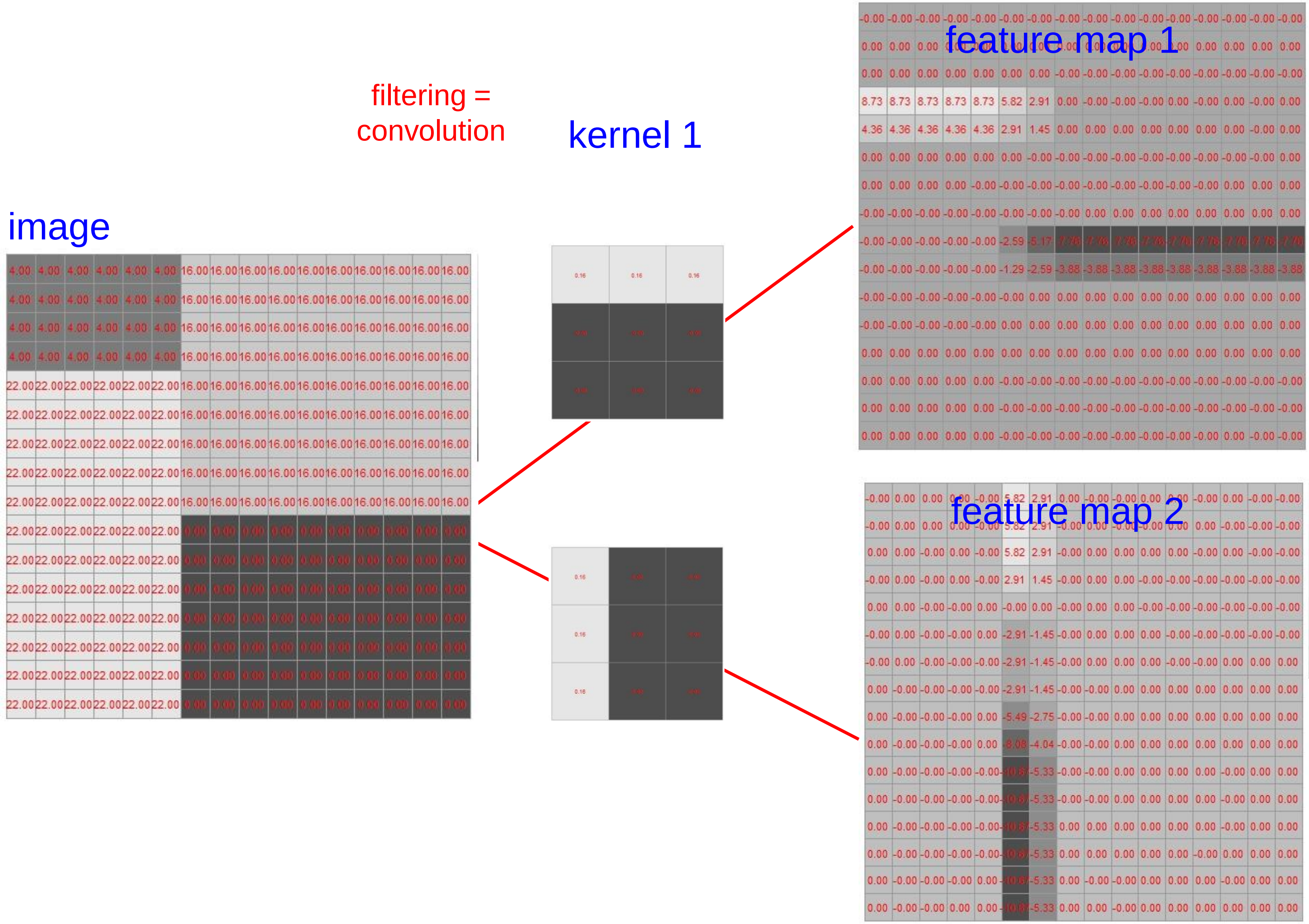




The filter is applied at each position of the image and it can be shown that the result is maximal if the image pattern corresponds to the weight pattern.

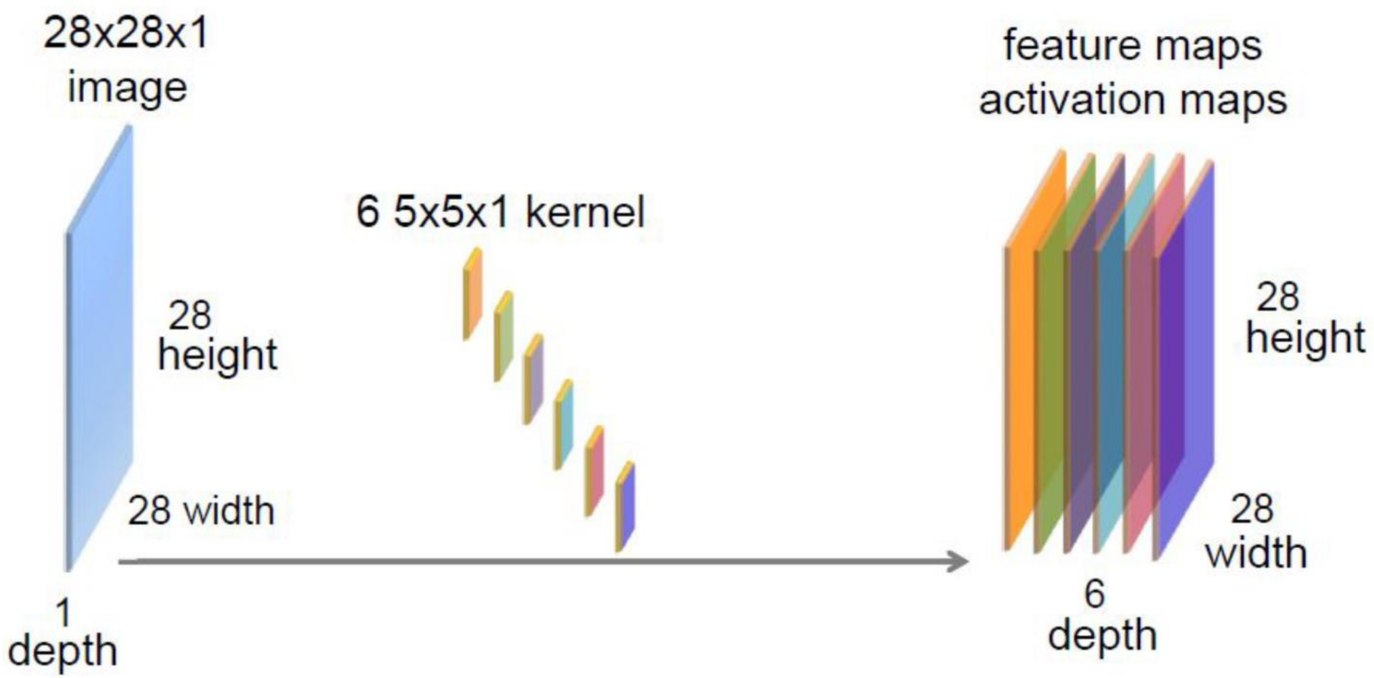
The results form again an image called feature map (=activation map) which shows at which position the feature is present.

Convolutional networks use neighborhood information and replicated local feature extraction



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

Convolution layer with a 1-chanel input and 6 kernels

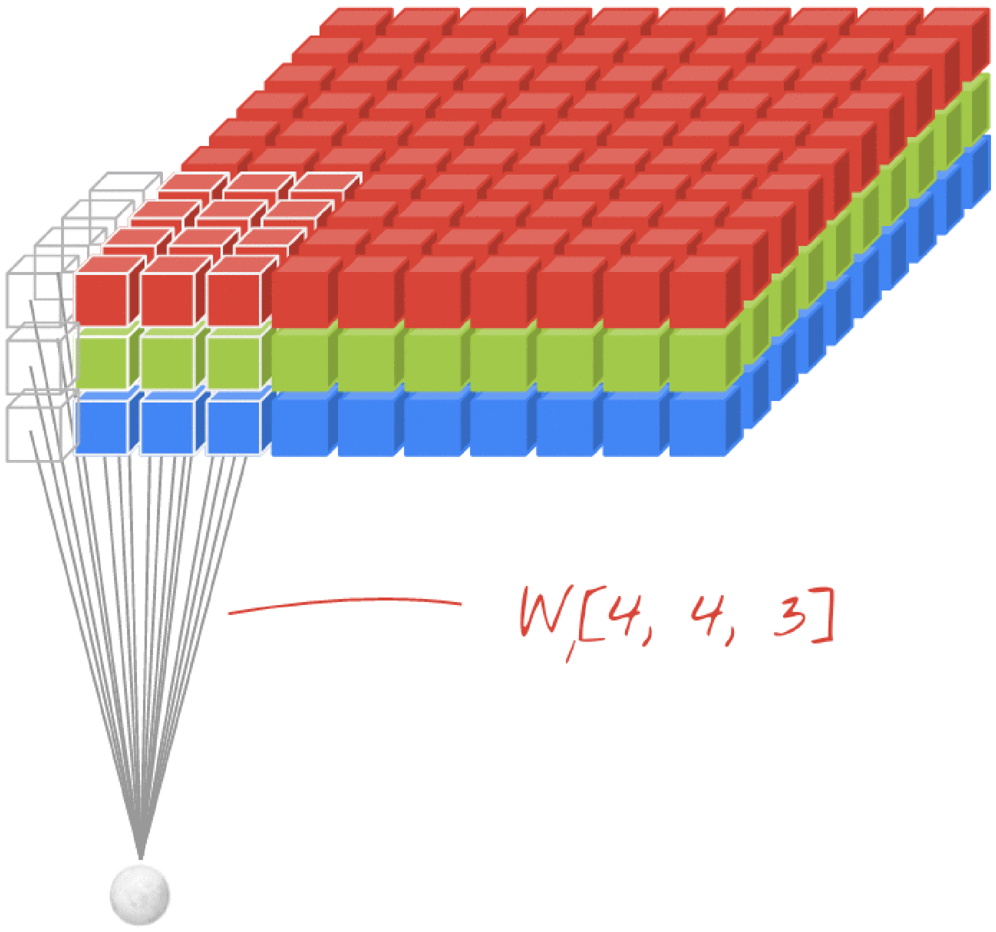


Convolution of the input image with 6 different kernels results in 6 activation maps.

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

Animated convolution with 3 input channels

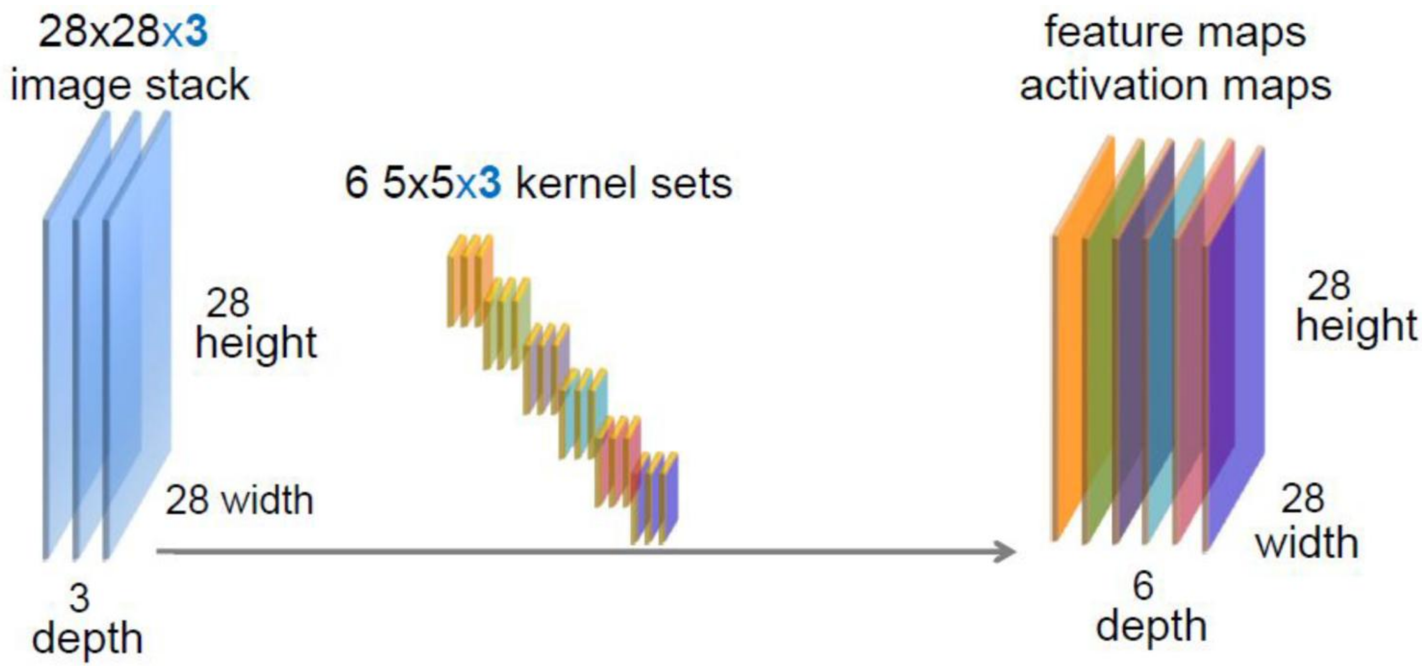
3 color channel input image



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

Animation credits: M.Gorner, https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#10
For an example with number see convolution demo in: https://cs231n.github.io/convolutional-networks/

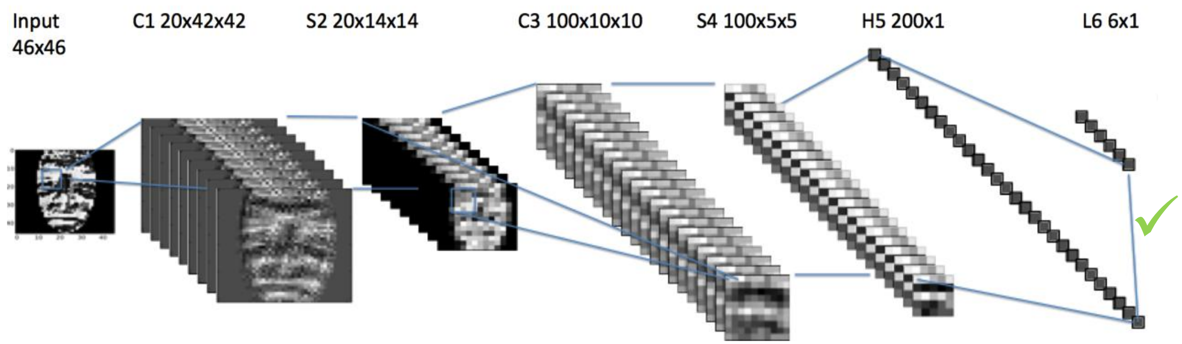
Convolution layer with a 3-chanel input and 6 kernels



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

→

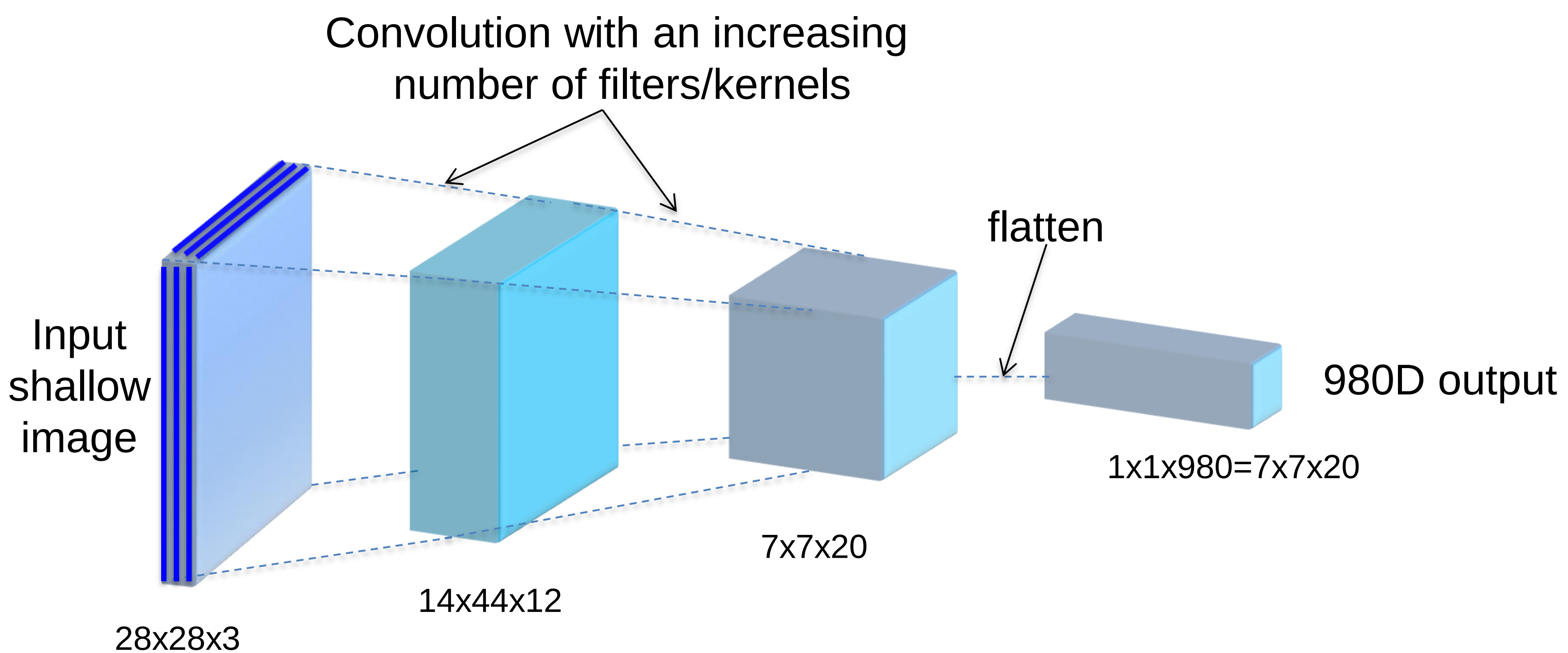
2x2

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

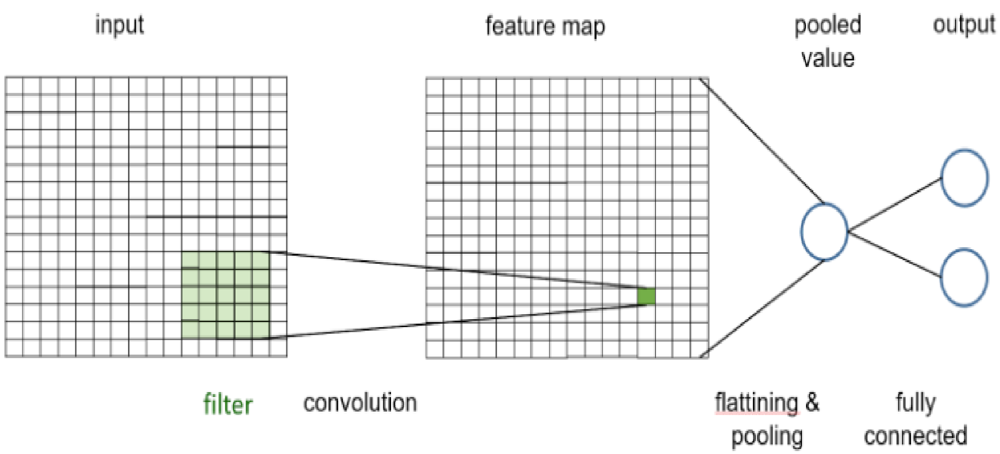
Hinton: „The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster“

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.

Building a very simple CNN with keras

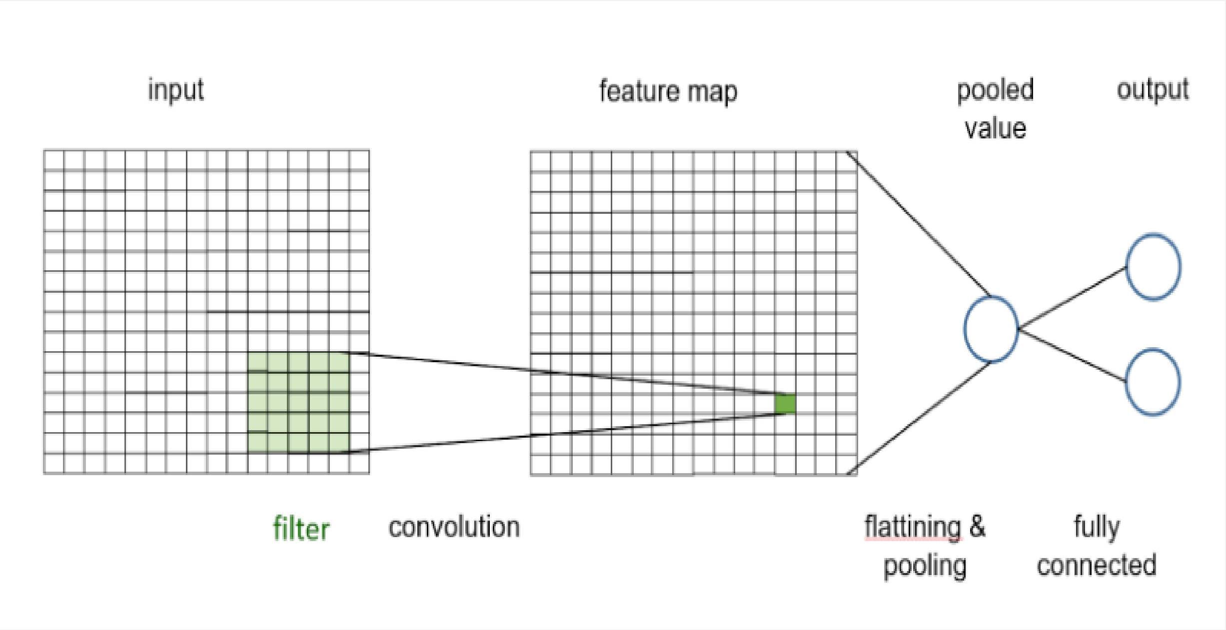



```
model <- keras_model_sequential()
model %>%
  layer_conv_2d(filters=,
                kernel_size = c(5,5),
                padding = 'same',
…                input_shape = …,
                activation = 'linear') %>%
  # take the max over all values in the activation map
  layer_max_pooling_2d(pool_size = …) %>%
  layer_flatten() %>%
  layer_dense(units = 2,activation = 'softmax')
```

Fill the gaps!

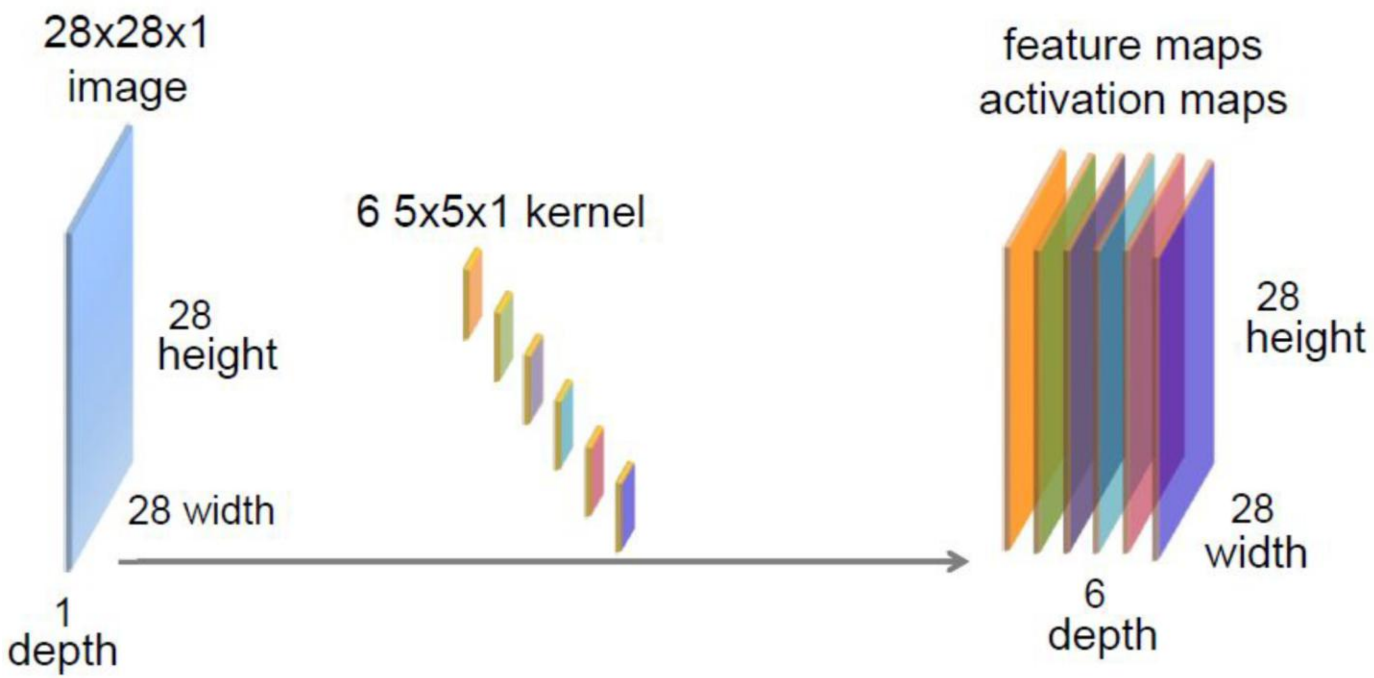
Exercise: Artstyle Lover

Train a CNN with only one filter in one conv-layer, that can predict, if the presented image shows vertical or horizontal lines.



Open NB in: https://github.com/tensorchiefs/dl_course_2025/blob/master/notebooks/05_cnn_edge_lover_keras_torch.

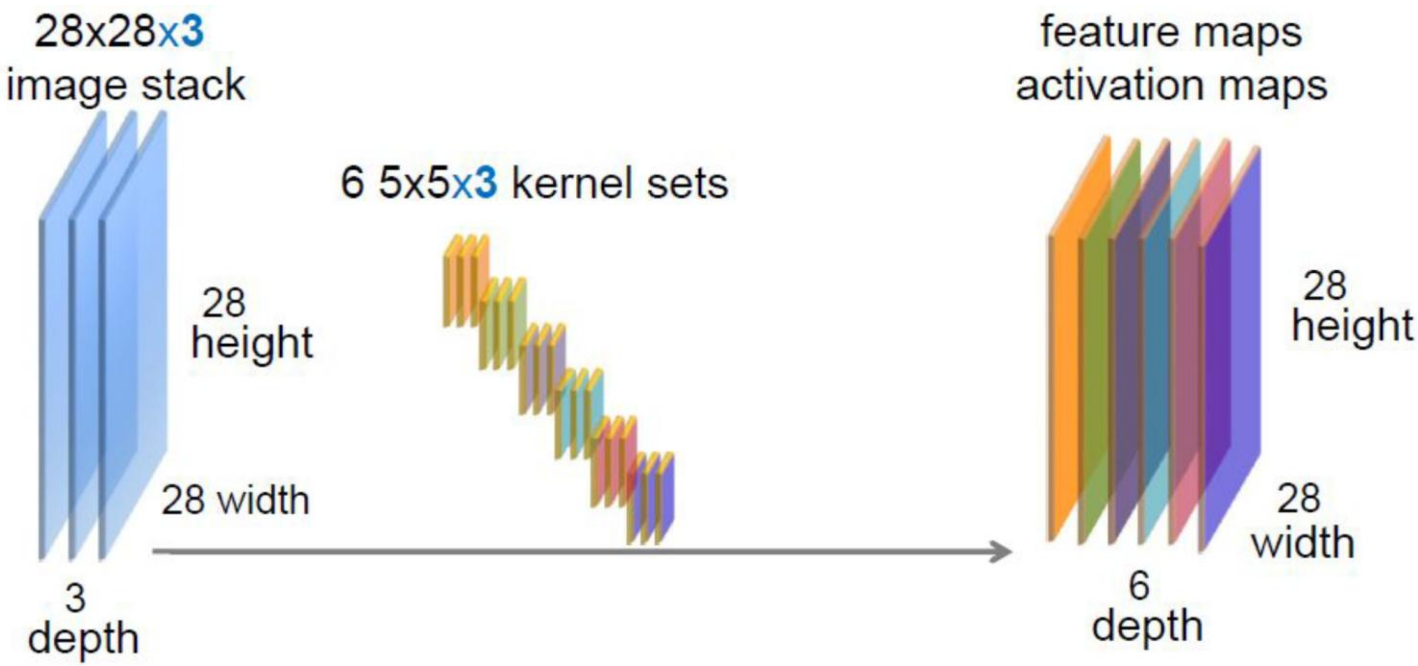
Convolution layer with a 1-chanel input and 6 kernels



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.

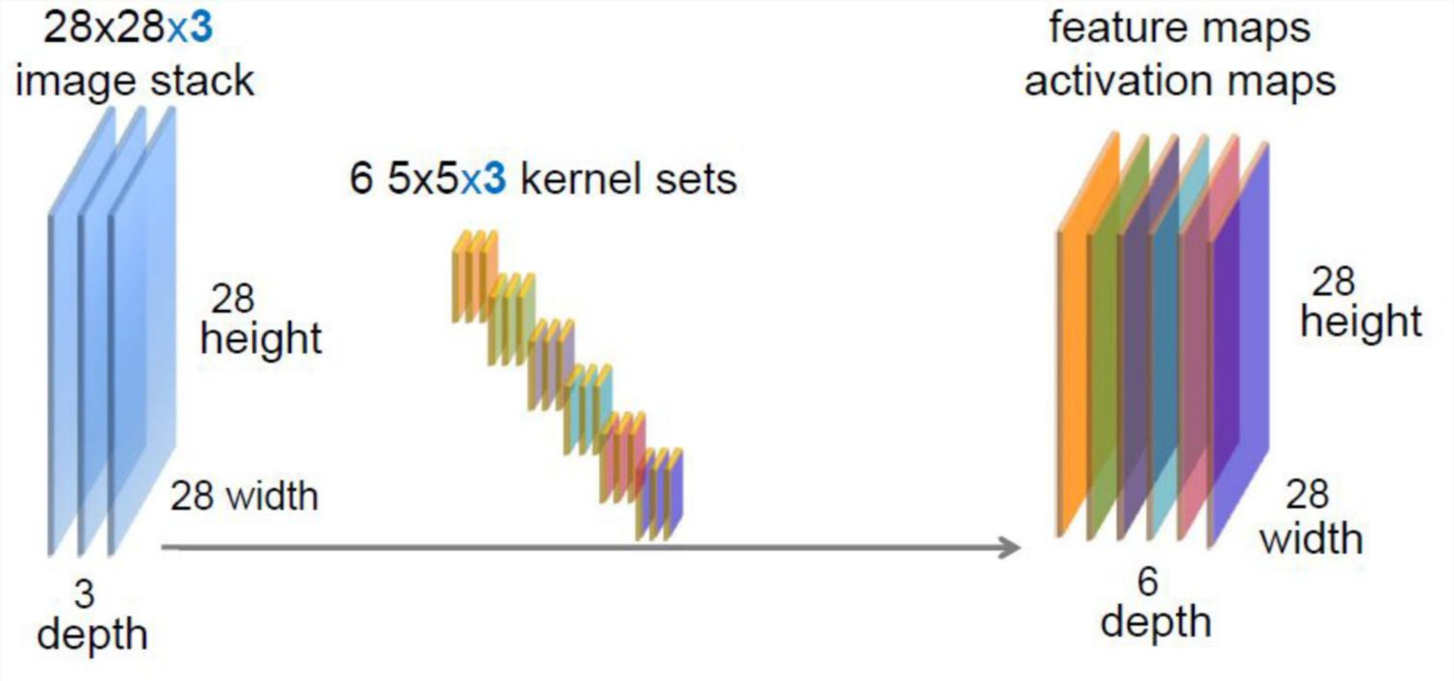
Convolution layer with a 3-chanel input and 6 kernels



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

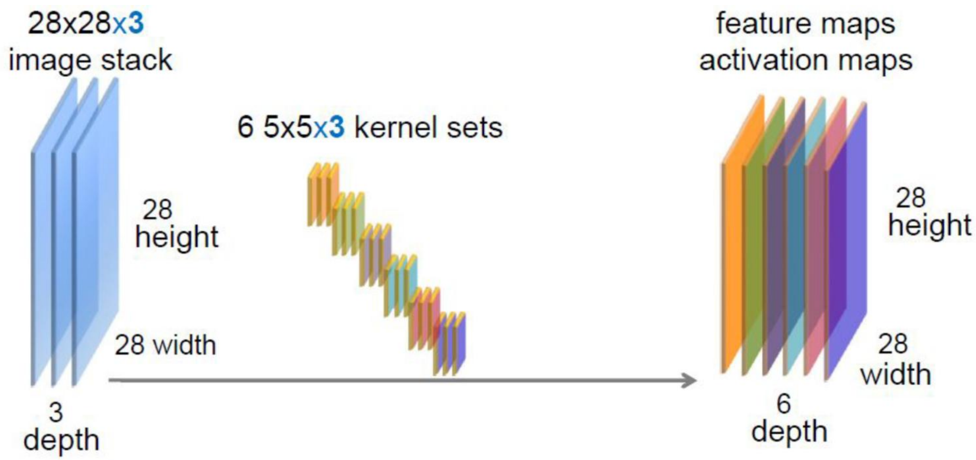
If the input image has 3 channels, then each filter has also 3 channels.

Convolution layer with a 3-chanel input and 6 kernels



How many weights?

Solution



$$6*5*5*3 + 6 = 456$$

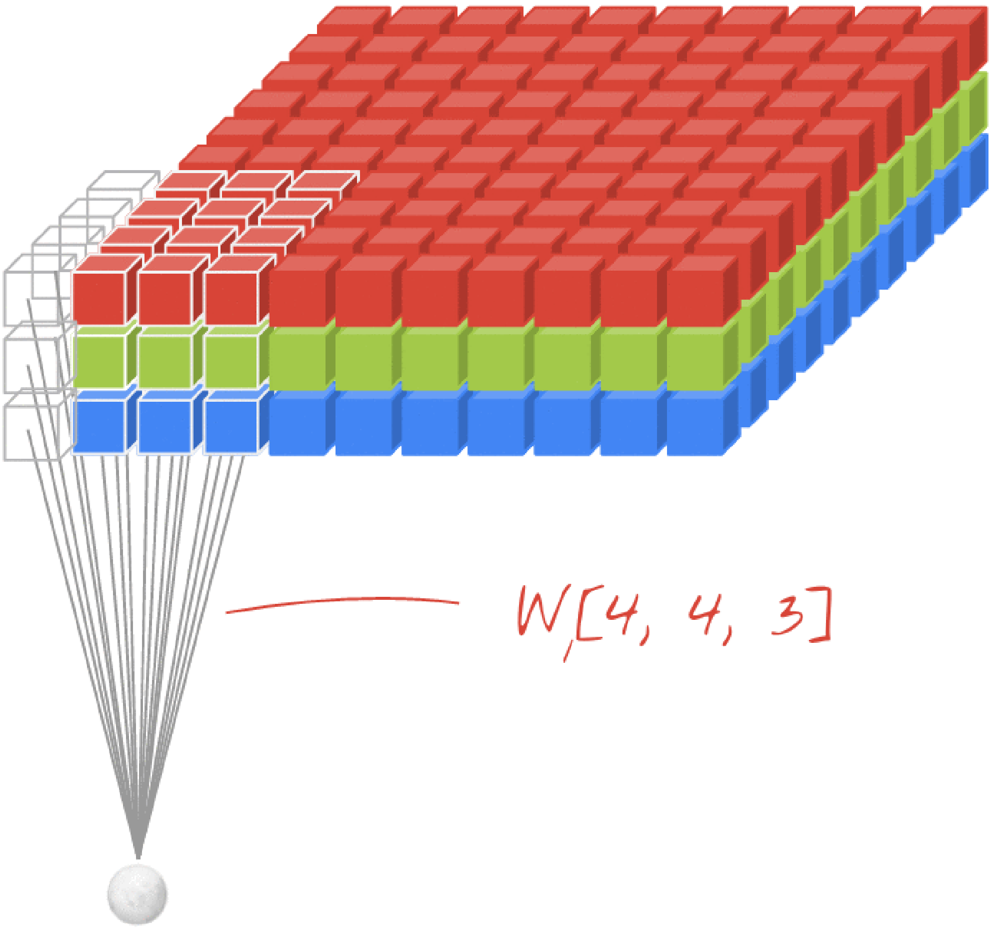
```
# Number of weights in Convolution
model = Sequential()
model.add(Convolution2D(6,kernel_size=(5,5),padding='same',input_shape=(28,28,3)))
model.summary()
```

Model: "sequential_11"

Layer (type)	Output Shape	Param #
conv2d_44 (Conv2D)	(None, 28, 28, 6)	456

Total params: 456 (1.78 KB)
Trainable params: 456 (1.78 KB)
Non-trainable params: 0 (0.00 B)

Animated convolution with 3 input channels



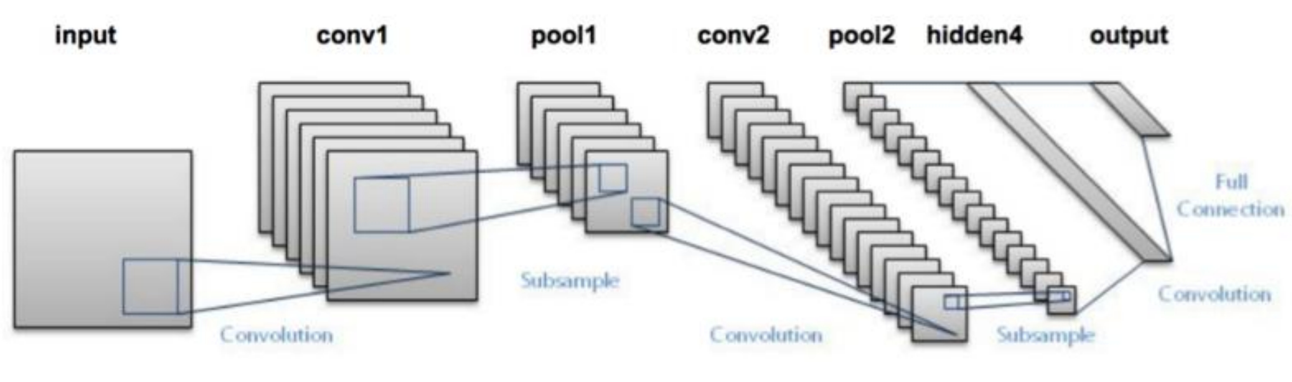
3 color channel input image

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

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

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

CNN for MNIST

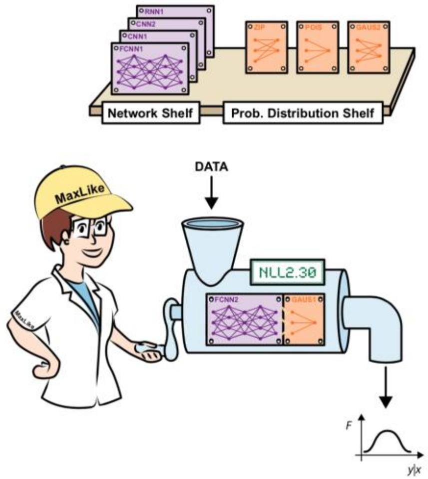



```
model = Sequential()

model.add(Convolution2D(filters=8,kernel_size=(3,3),
                        padding='same',input_shape=(28,28,1)))
model.add(Activation('relu'))
model.add(Convolution2D(16, (3, 3),padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(40))
model.add(Activation('relu'))
model.add(Dense(10))
model.add(Activation('softmax'))

# compile model and intitialize weights
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Summary



- The NN architecture choice depends on the structure of the data.
  - Fully connected NNs work best for tabular data
  - CNNs work best for images and other data with local order

- CNNs exploit the local structure of images by local connections and shared weight (same kernel is applied at each position of the image).

- In each convolutional layer of a CNN we define how many filters/kernel (usually 3x3) are learned resulting in the same number of feature maps aka activation maps, which are the input to the next convolutional layer.

- After the convolutional part of the CNN we flatten the output of the last convolutional layer to a vector of tabular “features” that are used as input to one or several denslayer, where the last layer has as many nodes as we have parameters in the predicted outcome distribution using the corresponding activation function and NLL loss function.