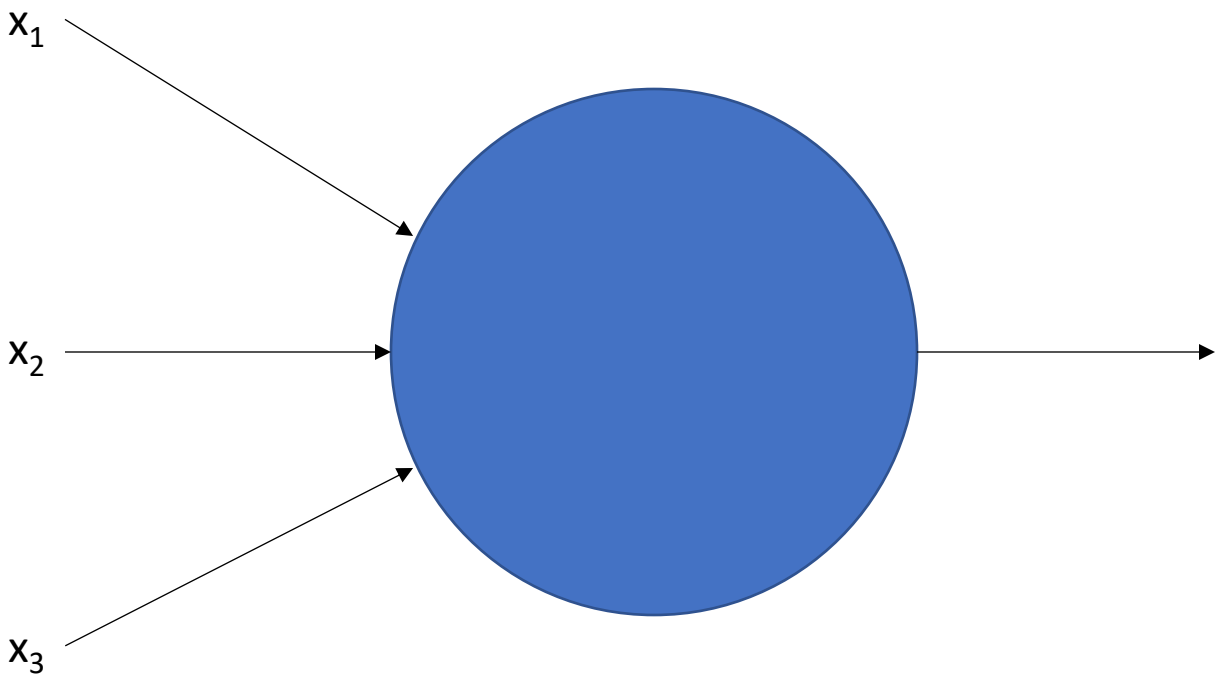
Data Science Meetup Kassel
Convolutional Neural Networks
27.02.2018

Nicolas Kuhaupt
Data Scientist Fraunhofer IEE

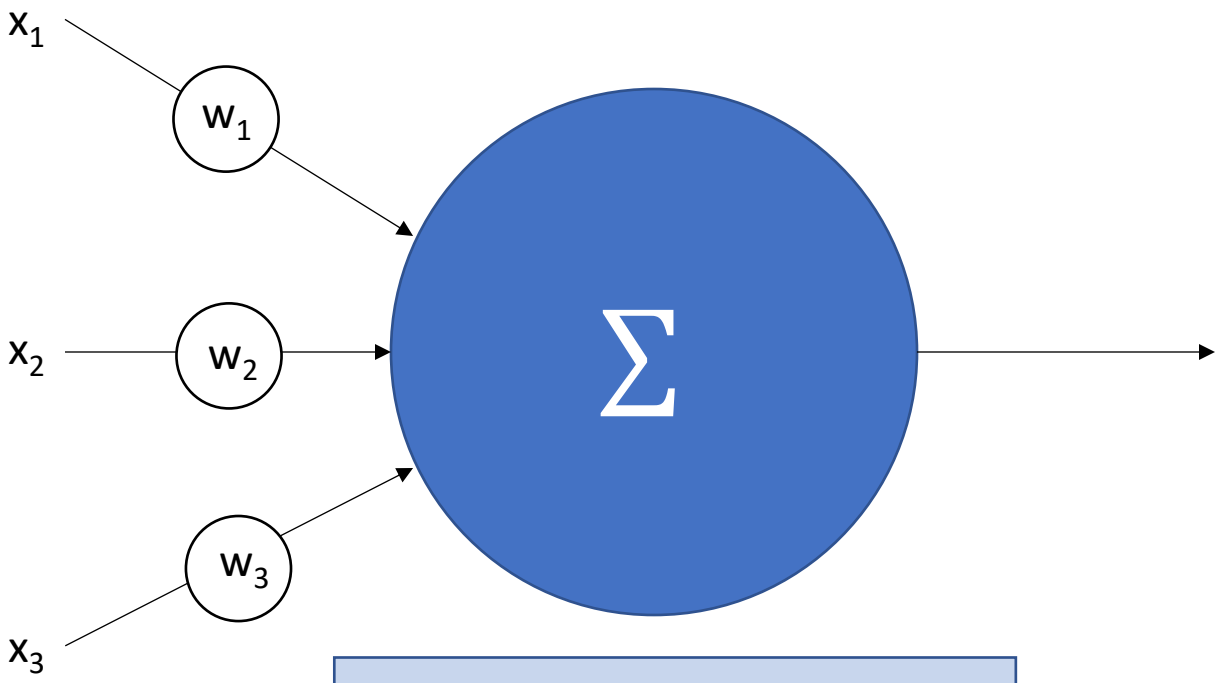
1. Neuronen
2. Neuronale Netzwerke
3. Convolutional Neural Network (CNN)
4. Anwendungen von CNN

1. **Neuronen**
2. Neuronale Netzwerke
3. Convolutional Neural Network (CNN)
4. Anwendungen von CNN

Neuron

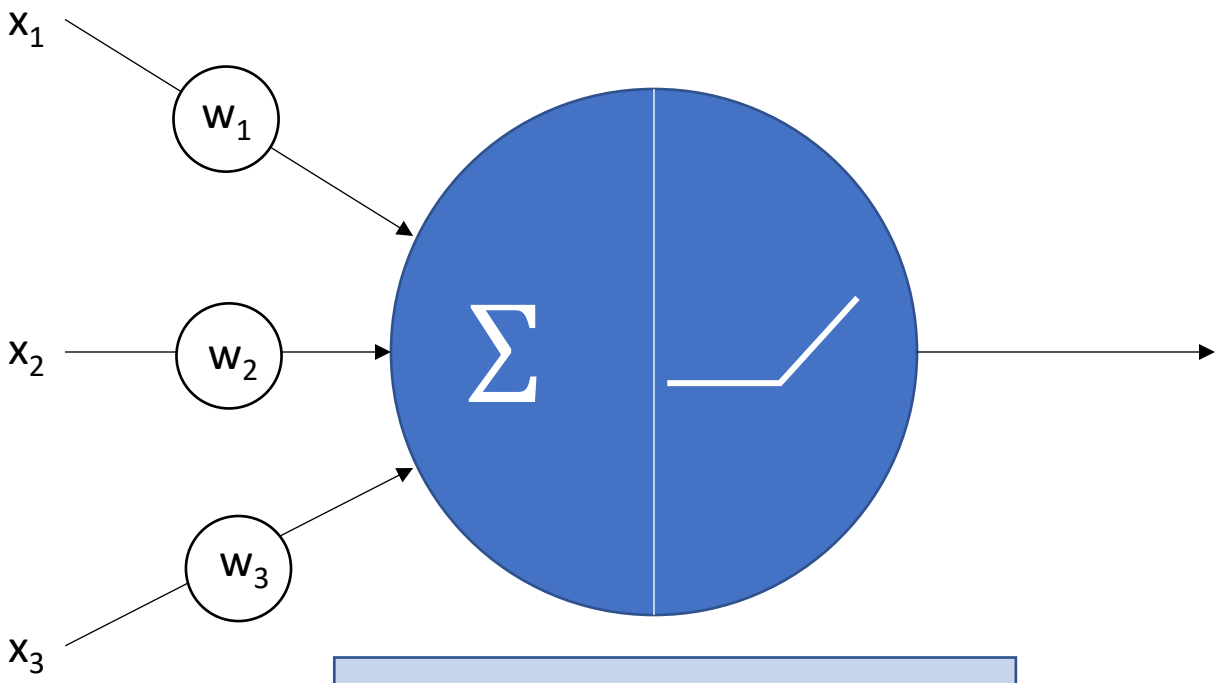


Neuron



$$z = w_1x_1 + w_2x_2 + w_3x_3 + b$$

Neuron - Aktivierungsfunktion

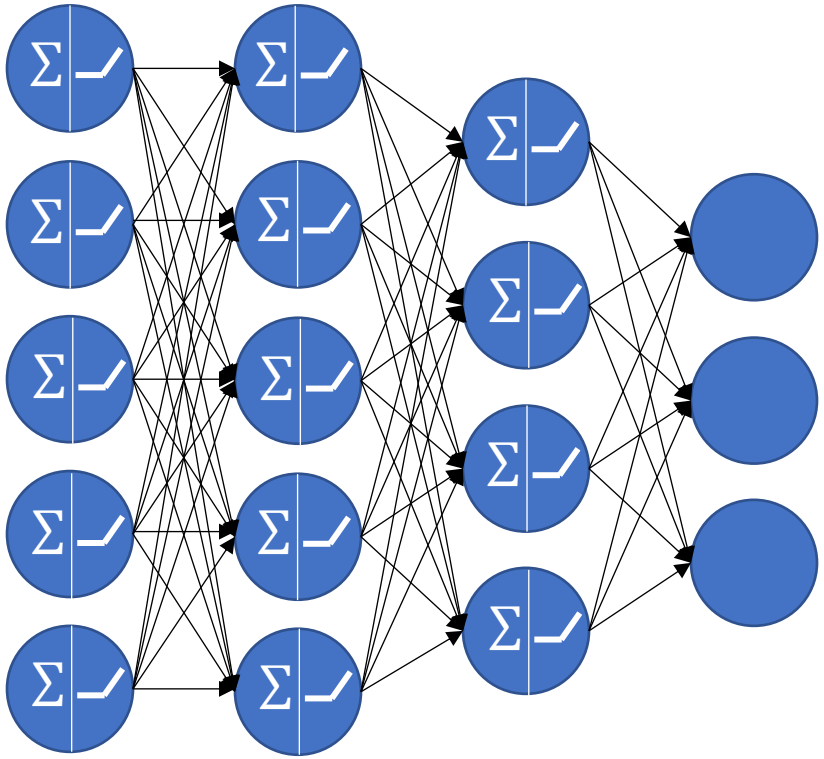


$$z = w_1x_1 + w_2x_2 + w_3x_3 + b$$

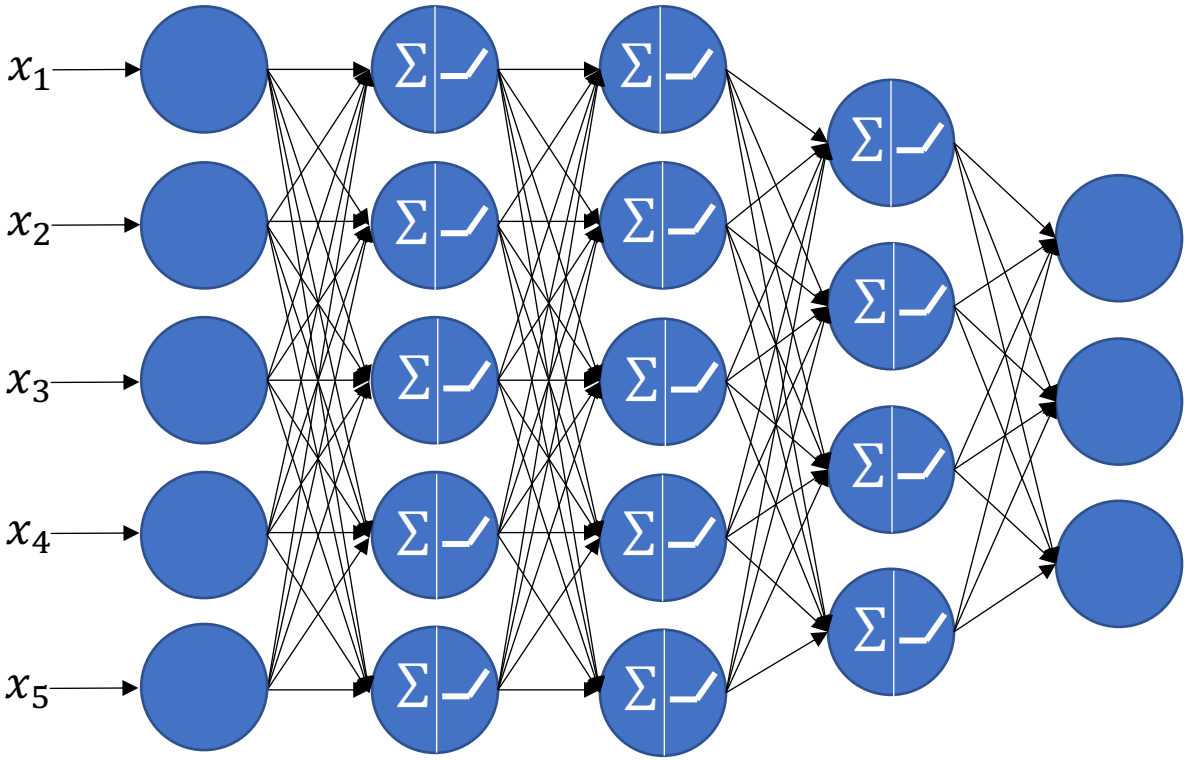
ReLu: $h(z) = \max(0, z)$

1. Neuronen
2. **Neuronale Netzwerke**
3. Convolutional Neural Network (CNN)
4. Anwendungen von CNN

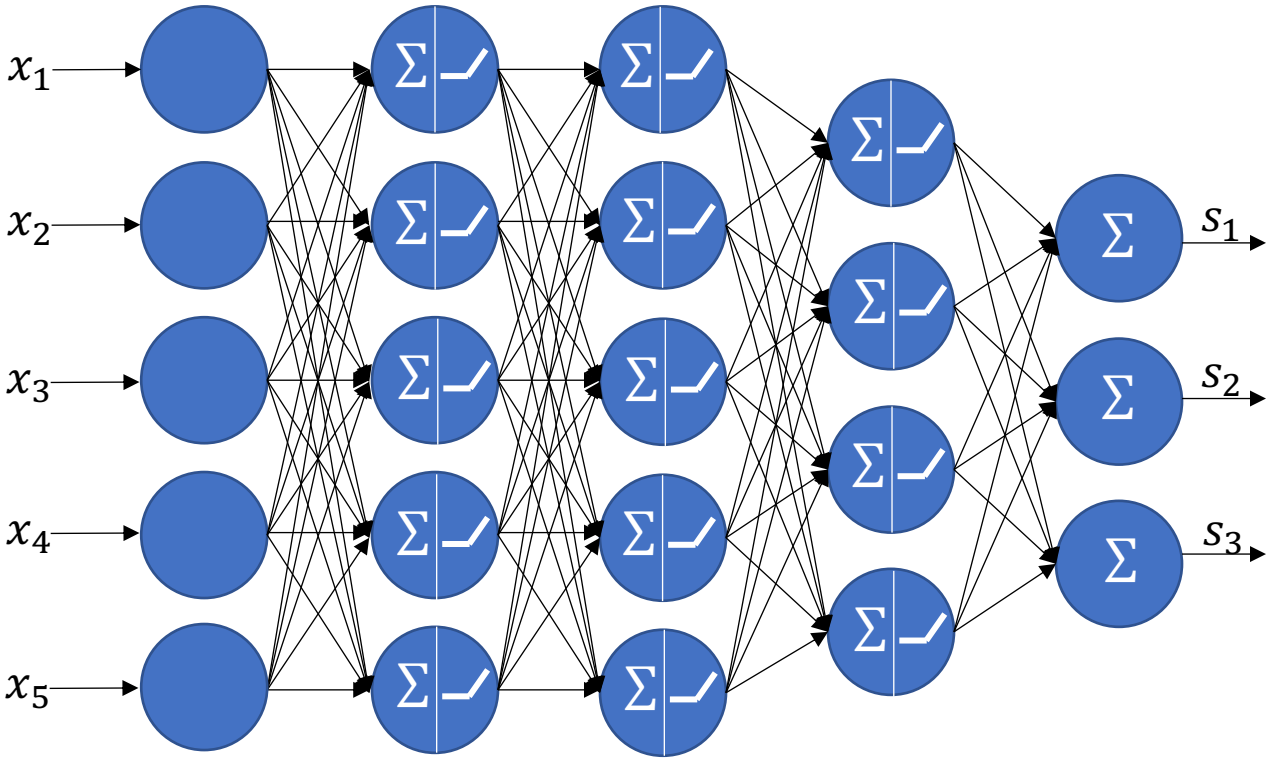
Neuronale Netze



Neuronale Netze – Input Layer



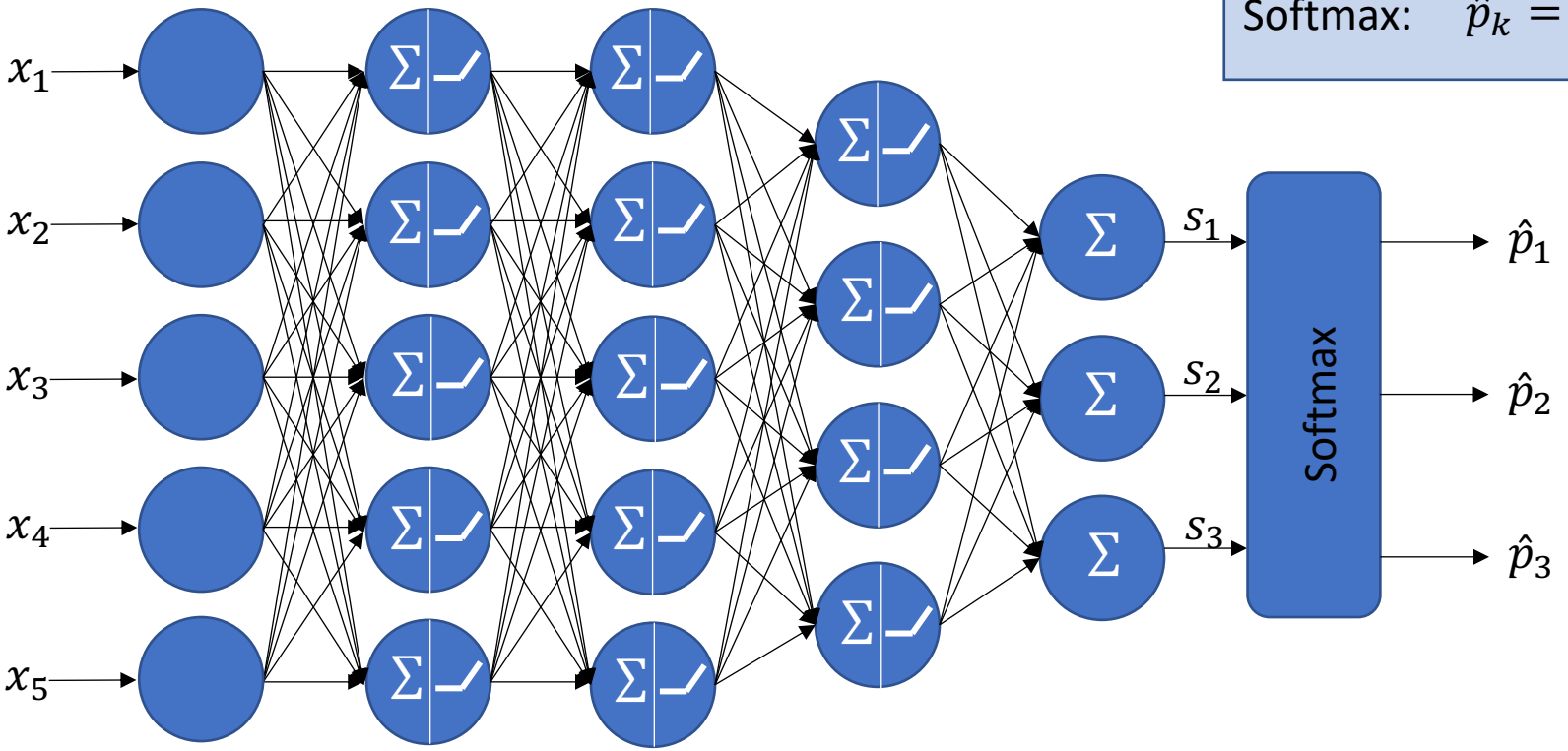
Neuronale Netze – Output Layer (I)



Neuronale Netze – Output Layer (II)

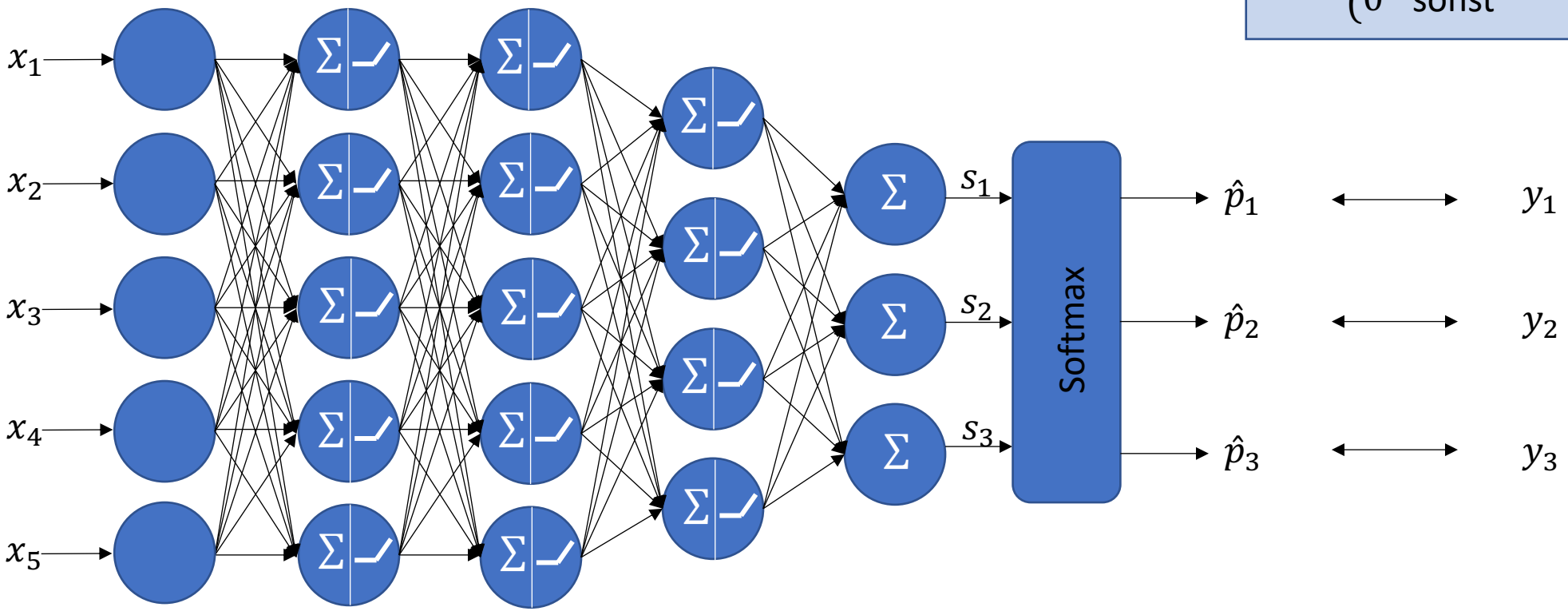
Von „Scores“ zu Wahrscheinlichkeiten:

Softmax: $\hat{p}_k = \frac{\exp(s_k)}{\sum \exp(s_i)}$



Neuronale Netze – Wie lernt ein NN?

$$y_i = \begin{cases} 1 & \text{wenn Klasse i} \\ 0 & \text{sonst} \end{cases}$$



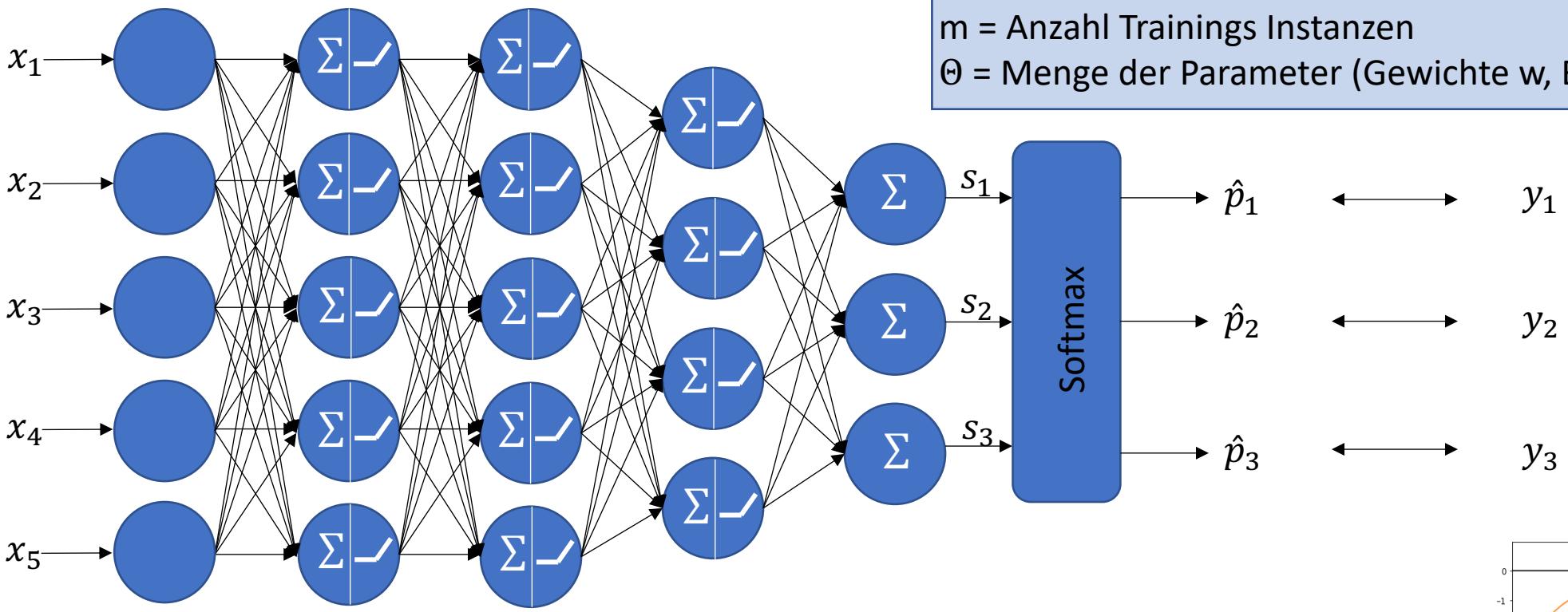
Neuronale Netze – Kostenfunktion

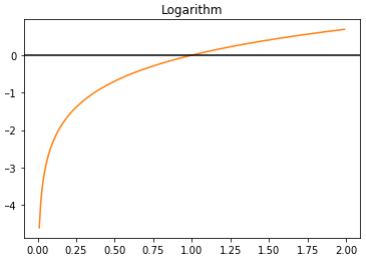
Cross Entropy: $C(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(\hat{p}_k^{(i)})$

K = Anzahl Klassen

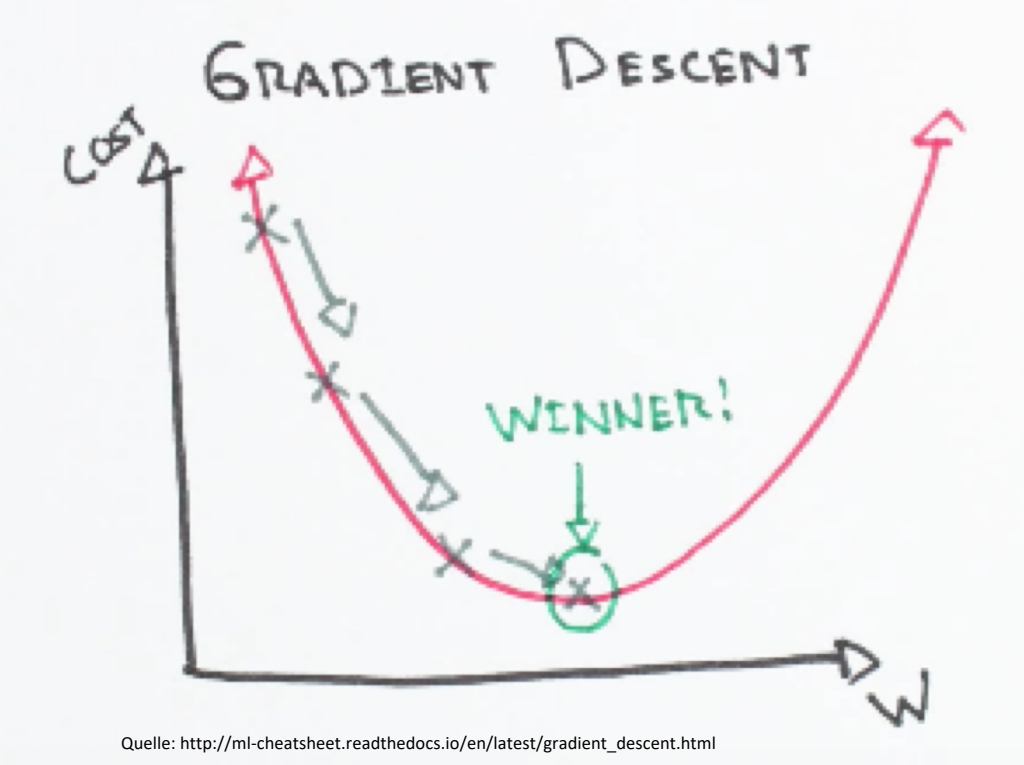
m = Anzahl Trainings Instanzen

Θ = Menge der Parameter (Gewichte w, Bias b)





Neuronale Netze – Gradient Descent (I)



Quelle: http://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html

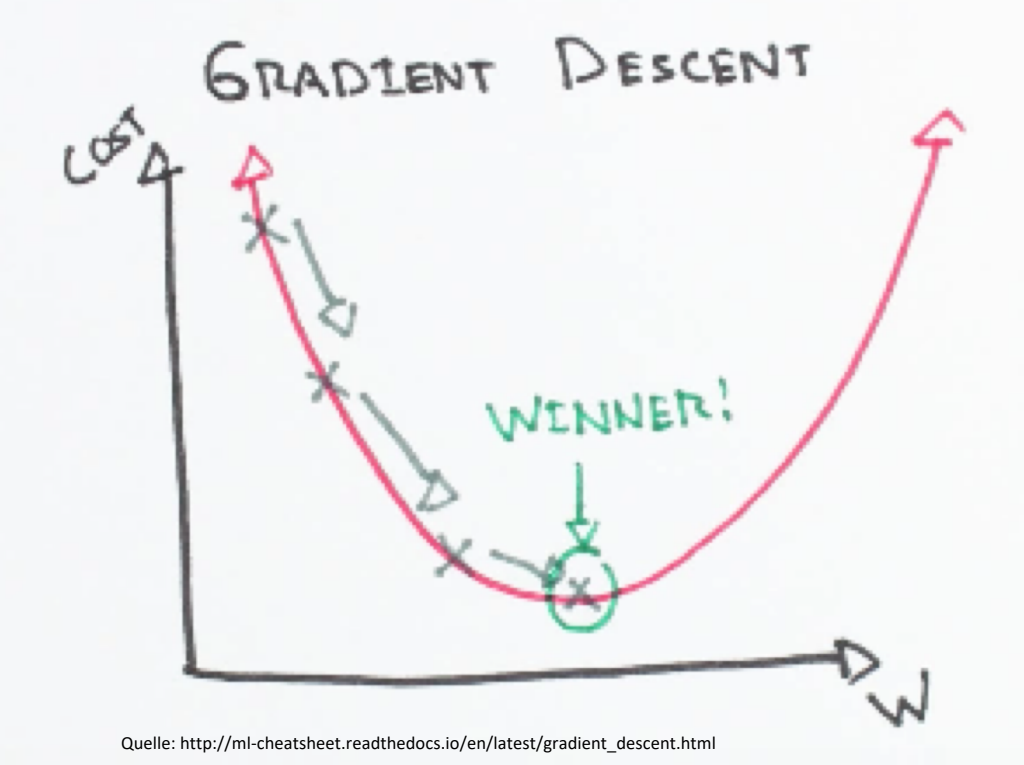
Cross Entropy: $C(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(\hat{p}_k^{(i)})$

K = Anzahl Klassen

m = Anzahl Trainings Instanzen

Θ = Menge der Parameter (Gewichte w, Bias b)

Neuronale Netze – Gradient Descent (II)



Quelle: http://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html

Cross Entropy: $C(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(\hat{p}_k^{(i)})$

K = Anzahl Klassen

m = Anzahl Trainings Instanzen

Θ = Menge der Parameter (Gewichte w, Bias b)

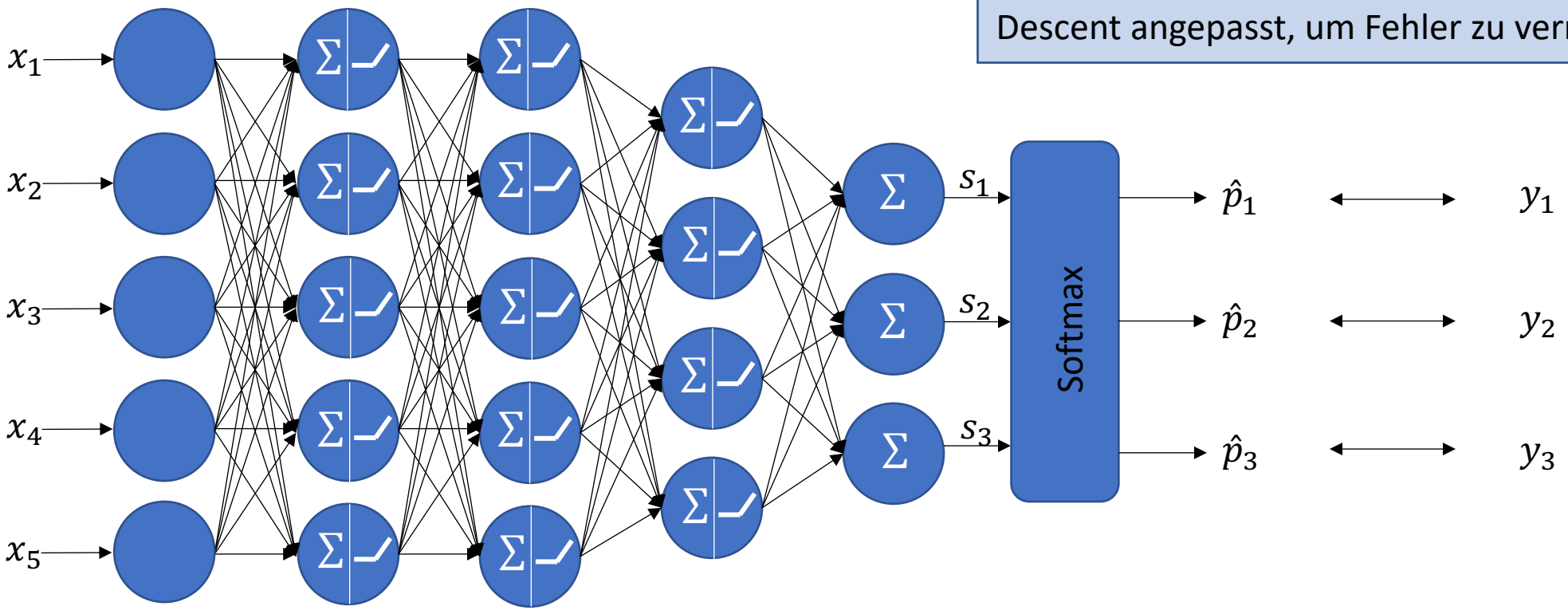
Stochastic Gradient Descent: m = 1

Mini Batch Gradient Descent: z.B. m = 50

Batch Gradient Descent: m = n

Neuronale Netze – Backpropagation

Fehler wird rückwärts durch Netzwerk propagiert (back propagation) und Gewichte mit Gradient Descent angepasst, um Fehler zu verringern



Tensorflow Intermezzo

- Deep Learning API in Python
- In C++ implementiert
- Multiplattform
- Visualisierung (Tensorboard)
- High Level API on top (Keras)
- Graph und Session

MNIST Datensatz

- Zahlen 0 – 9
- 28 x 28 Pixel
- 60000 Daten
- Jeder Pixel entspricht einem Grau-Wert

Quelle: https://en.wikipedia.org/wiki/MNIST_database#/media/File:MnistExamples.png

Tensorflow – NN Implementation: Graph

```python
import tensorflow as tf

n_inputs = 28*28
n_hidden1 = 300
n_hidden2 = 100
n_outputs = 10
```

```python
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int64, shape=(None), name="y")
```

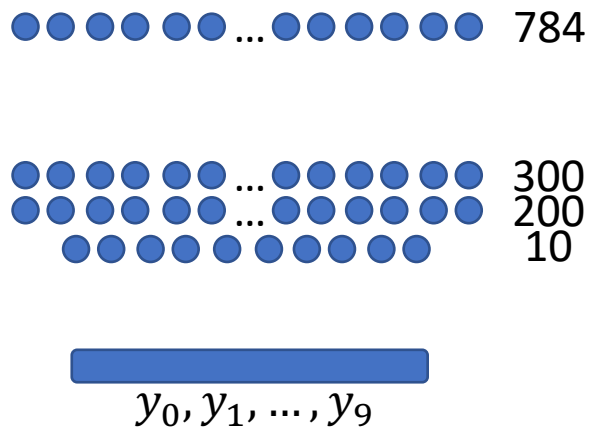
```python
with tf.name_scope("dnn"):
    hidden1 = tf.layers.dense(X, n_hidden1, name="hidden1", activation=tf.nn.relu)
    hidden2 = tf.layers.dense(hidden1, n_hidden2, name="hidden2", activation=tf.nn.relu)
    logits = tf.layers.dense(hidden2, n_outputs, name="outputs")
```

```python
with tf.name_scope("loss"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y, logits=logits, name="xentropy")
    loss = tf.reduce_mean(xentropy, name="loss")
```

```python
learning_rate = 0.01

with tf.name_scope("train"):
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    training_op = optimizer.minimize(loss)
```

```python
with tf.name_scope("eval"):
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
```



Tensorflow – NN Implementation: Session

```
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("/tmp/data/")
```

```
init = tf.global_variables_initializer()
```

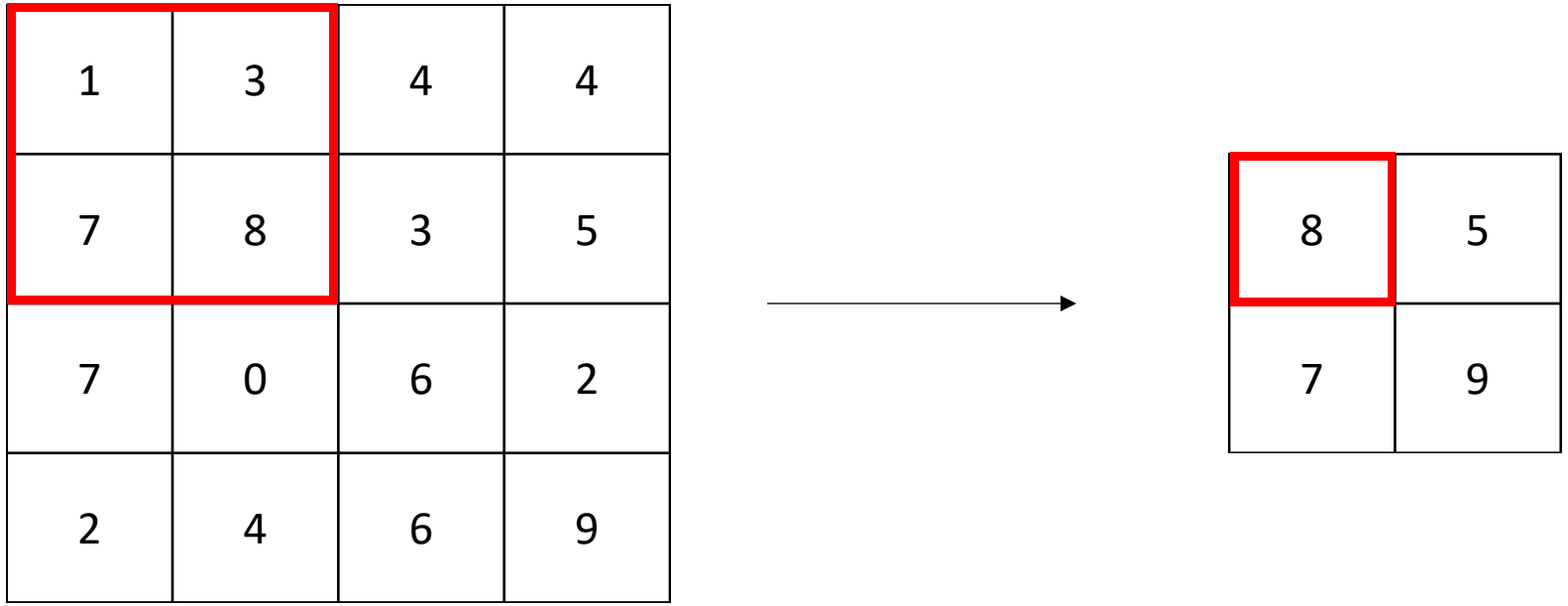
```
n_epochs = 20
batch_size = 50
```

```
with tf.Session() as sess:
    init.run()
    for epoch in range(n_epochs):
        for iteration in range(mnist.train.num_examples // batch_size):
            X_batch, y_batch = mnist.train.next_batch(batch_size)
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
        acc_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
        acc_test = accuracy.eval(feed_dict={X: mnist.test.images, y: mnist.test.labels})
        print(epoch, "Train accuracy:", acc_train, "Test accuracy:", acc_test)
```

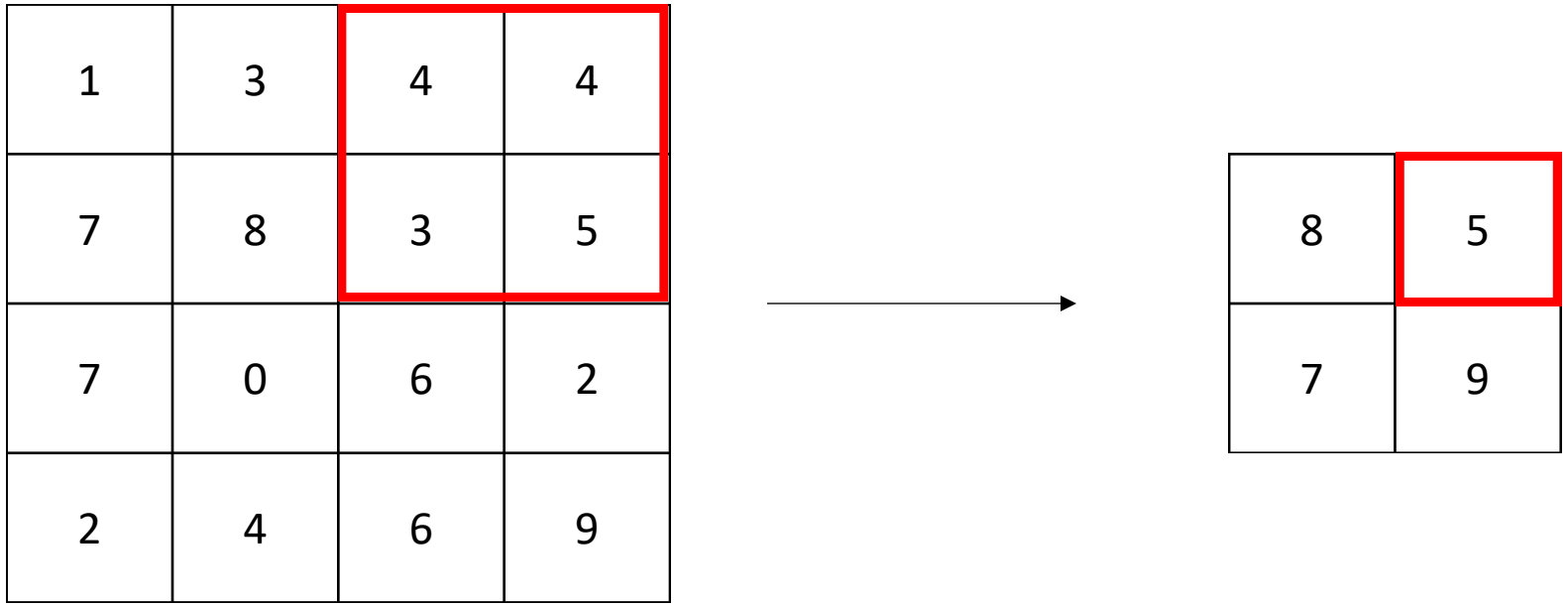
```
0 Train accuracy: 0.92 Test accuracy: 0.9061
1 Train accuracy: 0.98 Test accuracy: 0.9219
2 Train accuracy: 0.94 Test accuracy: 0.9326
3 Train accuracy: 0.82 Test accuracy: 0.938
4 Train accuracy: 0.9 Test accuracy: 0.9432
5 Train accuracy: 0.96 Test accuracy: 0.9468
6 Train accuracy: 0.98 Test accuracy: 0.9503
7 Train accuracy: 0.94 Test accuracy: 0.9539
8 Train accuracy: 0.98 Test accuracy: 0.9569
9 Train accuracy: 1.0 Test accuracy: 0.9584
10 Train accuracy: 0.98 Test accuracy: 0.9592
```

1. Neuronen
2. Neuronale Netzwerke
3. Convolutional Neural Network (CNN)
4. Anwendungen von CNN

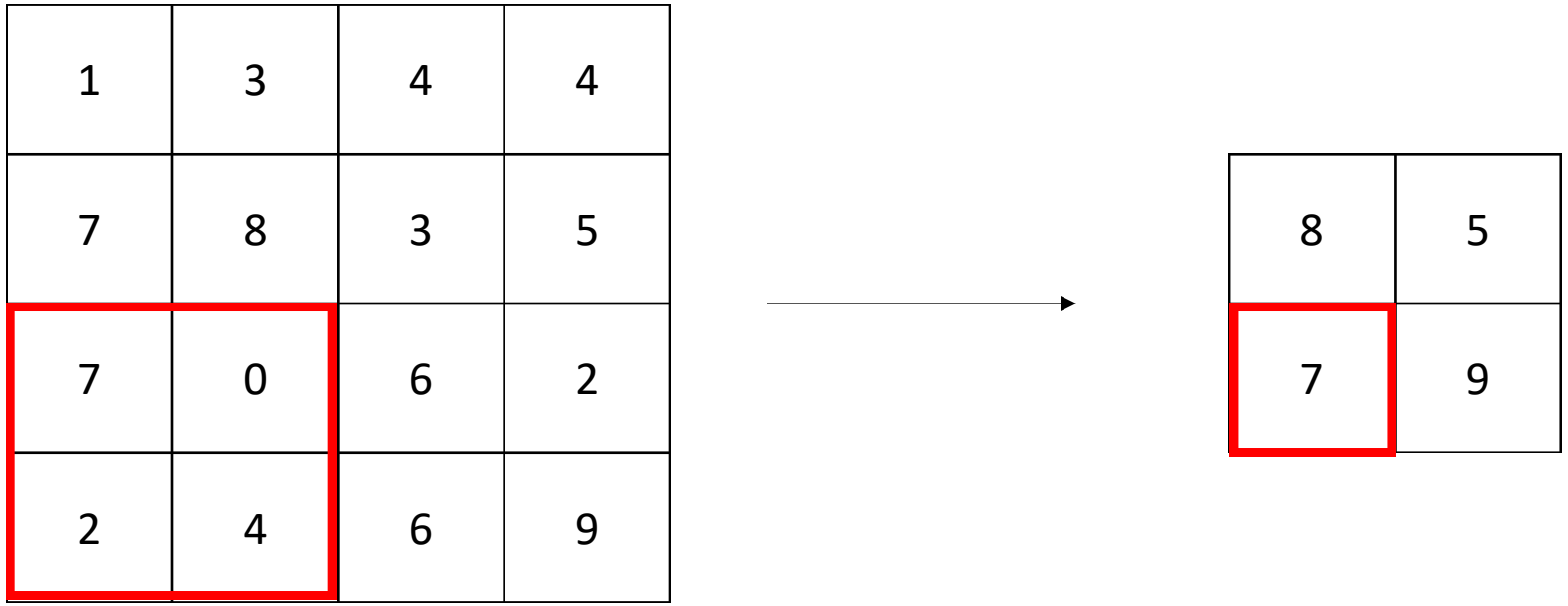
Convolutional Neural Network – Pooling Layer



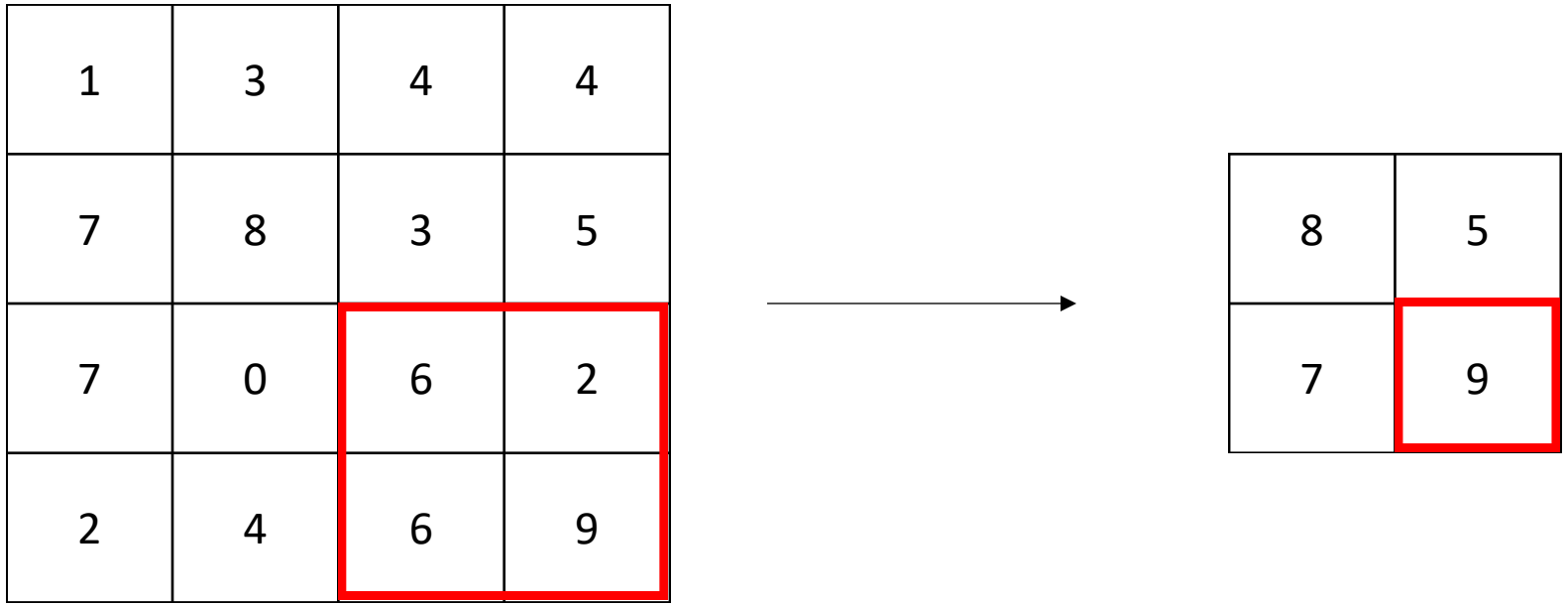
Convolutional Neural Network – Pooling Layer



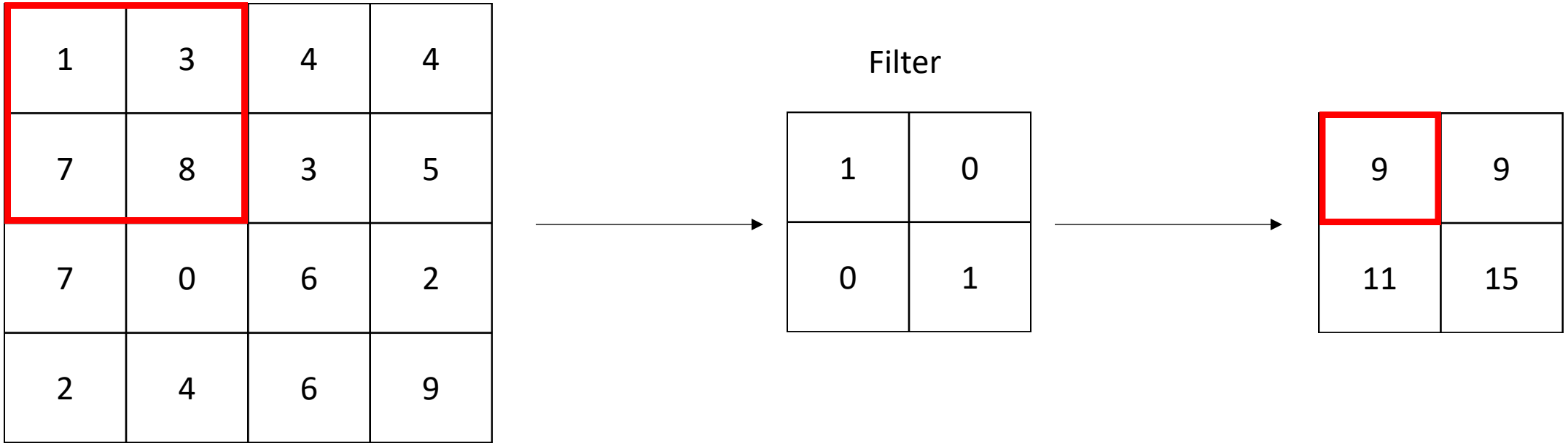
Convolutional Neural Network – Pooling Layer



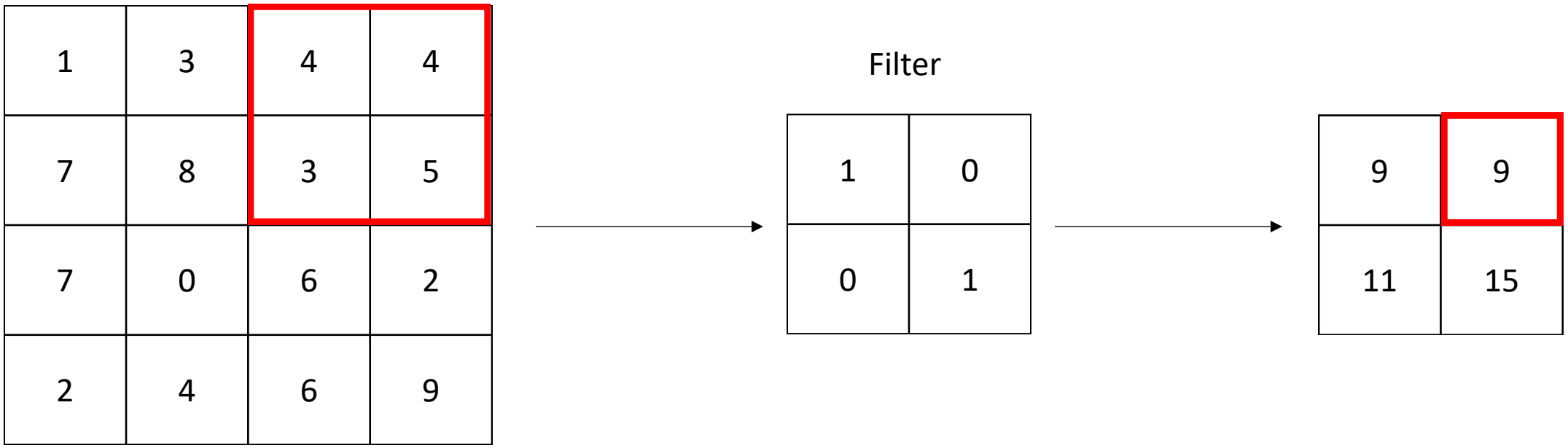
Convolutional Neural Network – Pooling Layer



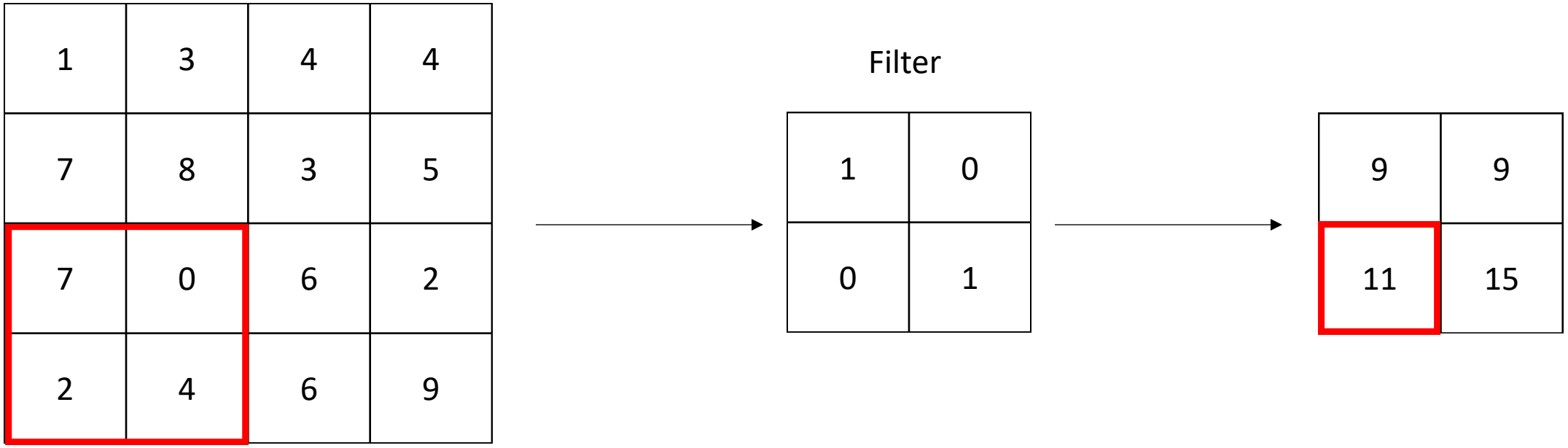
Convolutional Neural Network – Convolution Layer



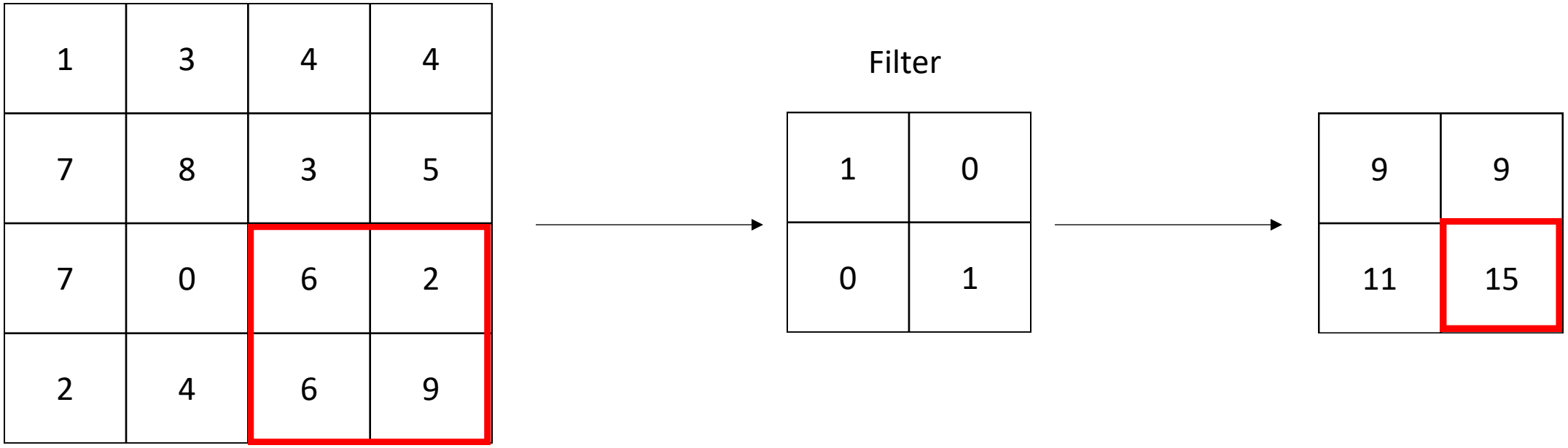
Convolutional Neural Network – Convolution Layer



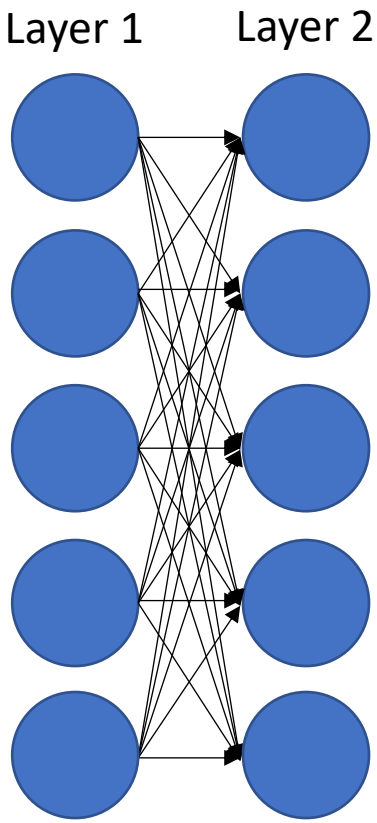
Convolutional Neural Network – Convolution Layer

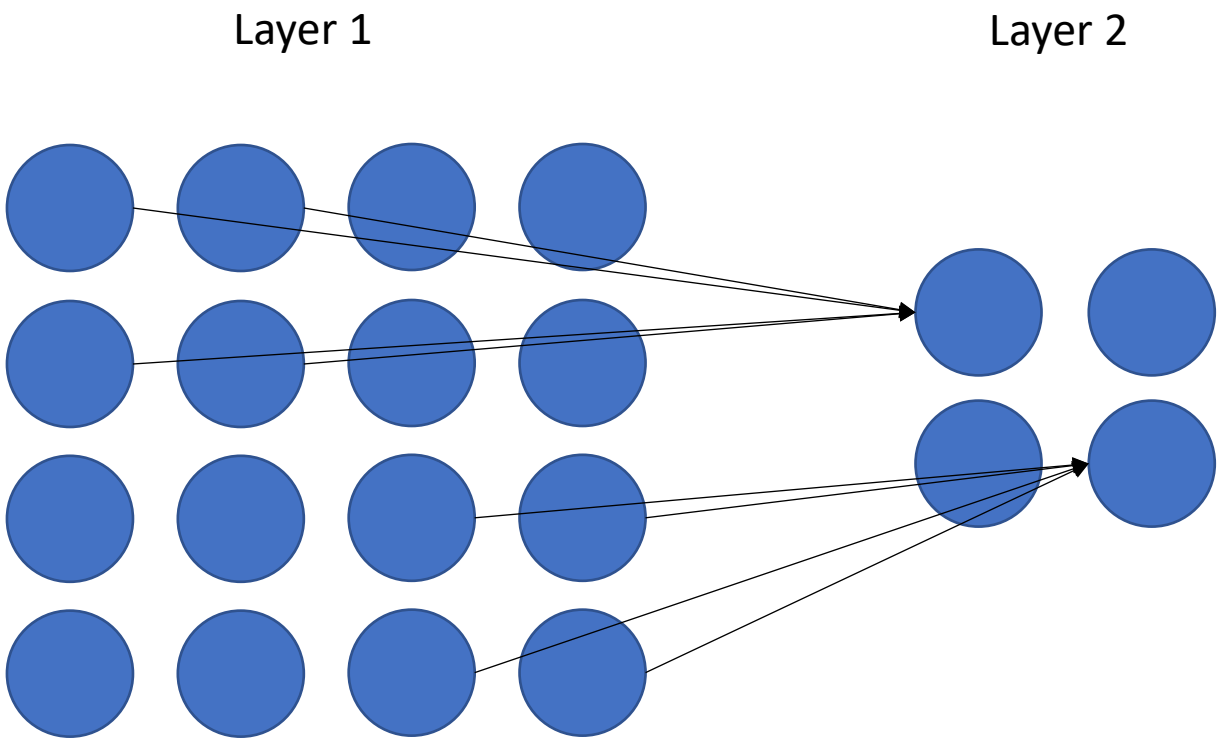


Convolutional Neural Network – Convolution Layer

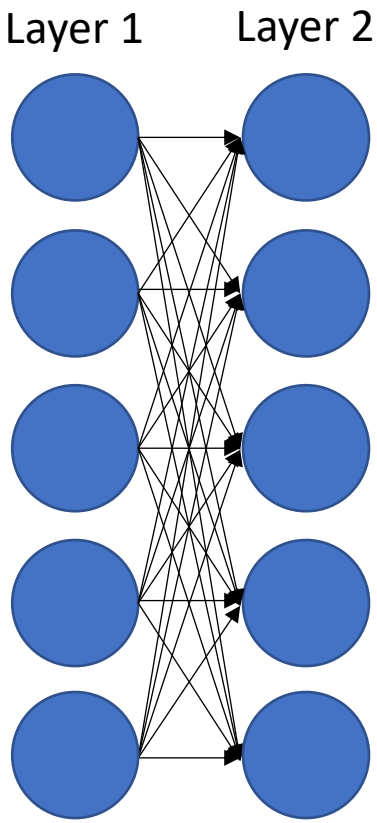


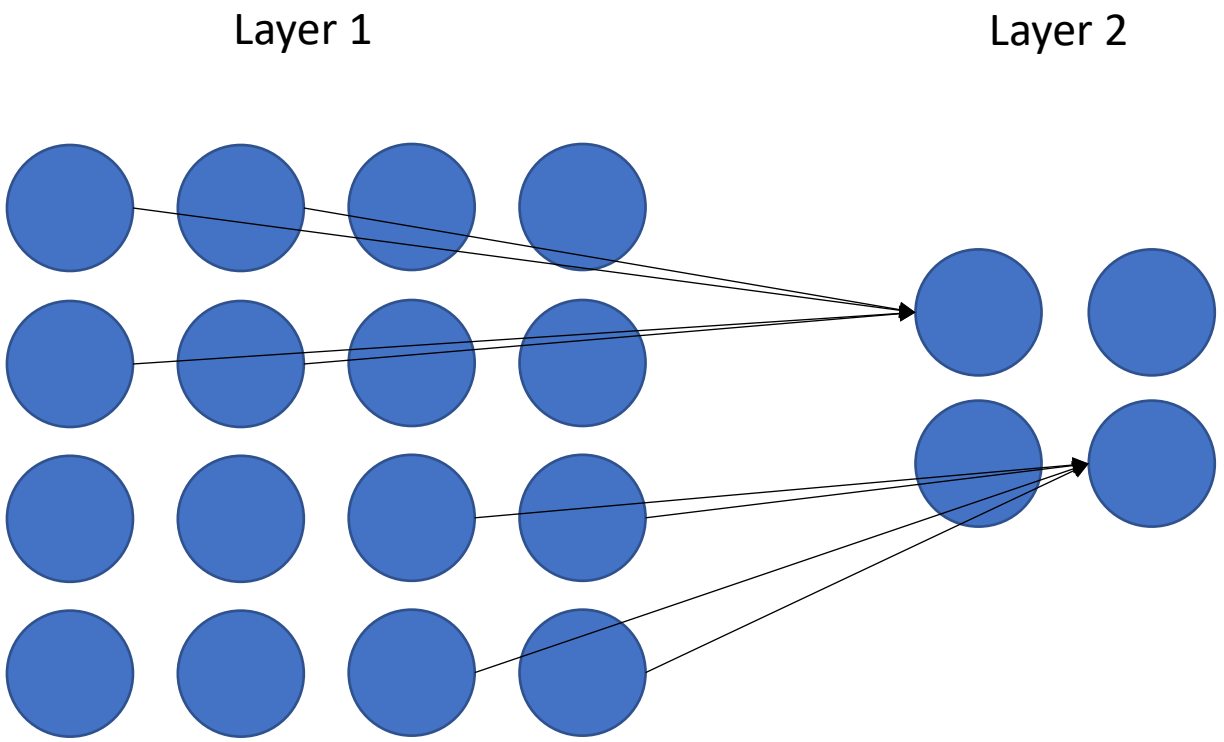
Convolutional Neural Network





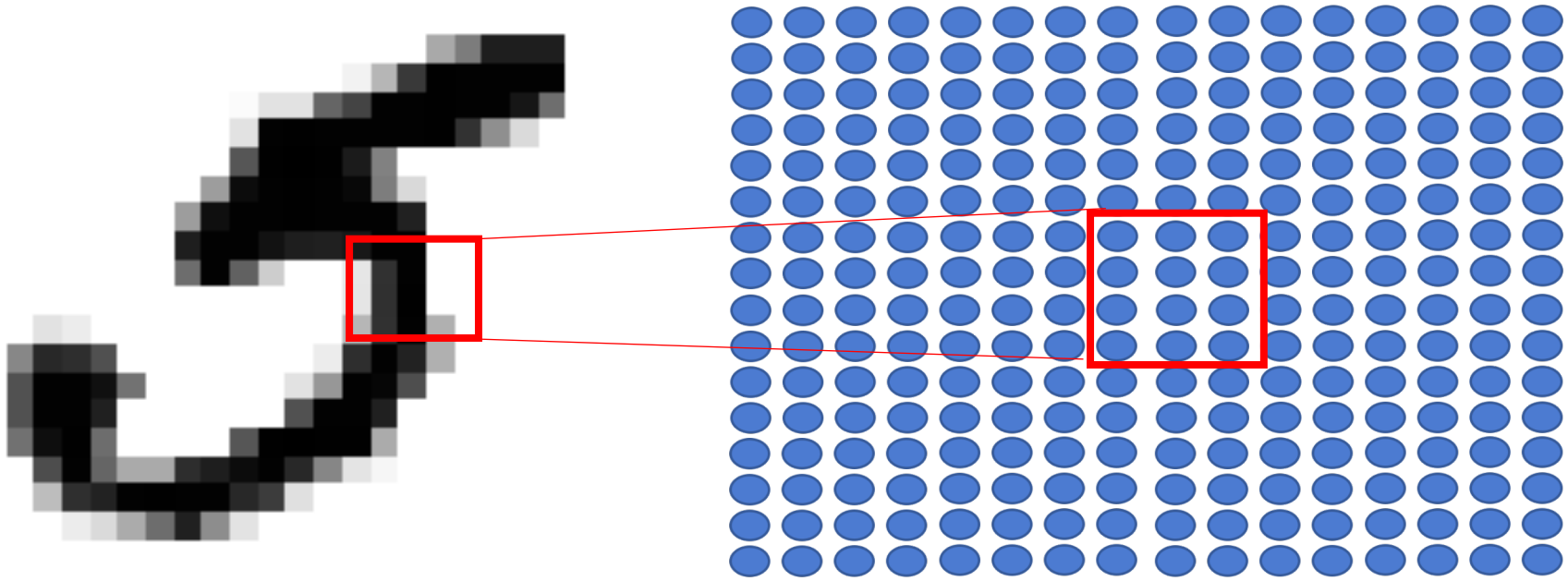
Convolutional Neural Network





Die Gewichte entsprechen hier dem Filter und werden somit gelernt.

Convolutional Neural Network – Intuition (I)

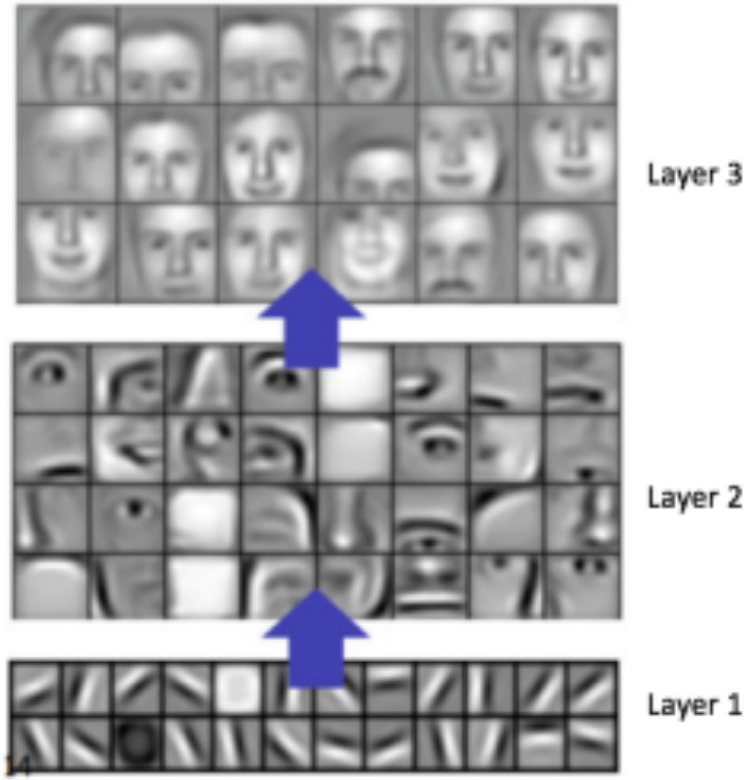


Convolutional Neural Network – Intuition (II)

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

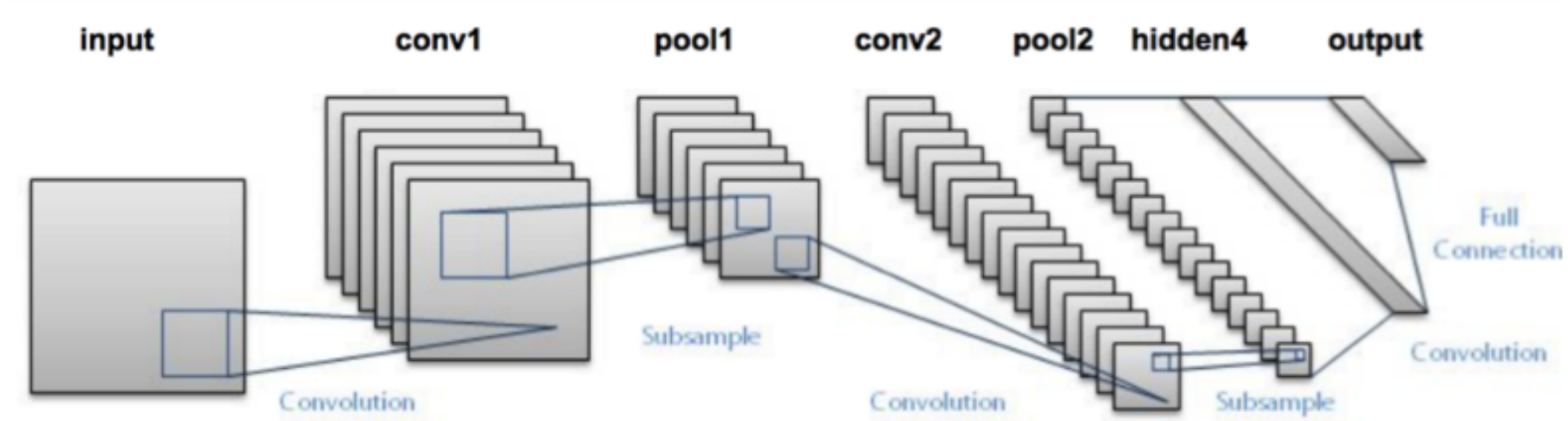
Quelle: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Convolutional Neural Network – Intuition (III)



Quelle: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Architekturen



1. Neuronen
2. Neuronale Netzwerke
3. Convolutional Neural Network (CNN)
4. Anwendungen von CNN

Bild Klassifizierung - Tensorflow Implementation – Graph

```python
import tensorflow as tf

with tf.name_scope("inputs"):
    X = tf.placeholder(tf.float32, shape=[None, 784], name="X")
    X_reshaped = tf.reshape(X, shape=[-1, 28, 28, 1])
    y = tf.placeholder(tf.int32, shape=[None], name="y")

conv1 = tf.layers.conv2d(X_reshaped, filters=32, kernel_size=3,
                         strides=1, padding='SAME',
                         activation=tf.nn.relu, name="conv1")

conv2 = tf.layers.conv2d(conv1, filters=64, kernel_size=3,
                         strides=2, padding='SAME',
                         activation=tf.nn.relu, name="conv2")

with tf.name_scope("pool3"):
    pool3 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding="VALID")
    pool3_flat = tf.reshape(pool3, shape=[-1, 64 * 7 * 7])

with tf.name_scope("fc1"):
    fc1 = tf.layers.dense(pool3_flat, 64, activation=tf.nn.relu, name="fc1")

with tf.name_scope("output"):
    logits = tf.layers.dense(fc1, 10, name="output")

with tf.name_scope("train"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=logits, labels=y)
    loss = tf.reduce_mean(xentropy)
    optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
    training_op = optimizer.minimize(loss)
```

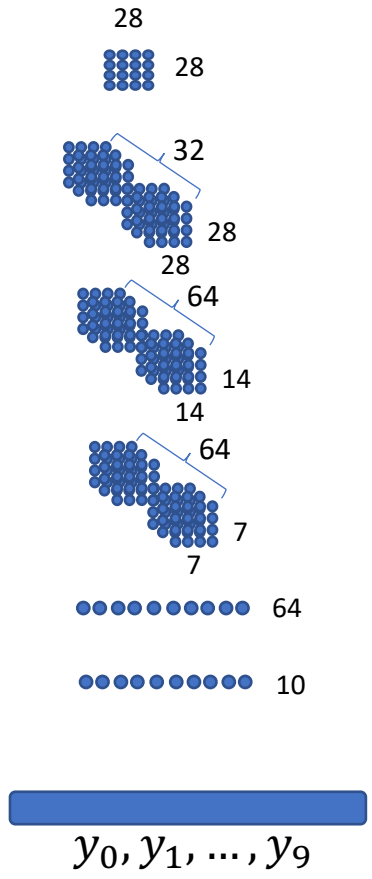


Bild Klassifizierung - Tensorflow Implementation - Session

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/")
```

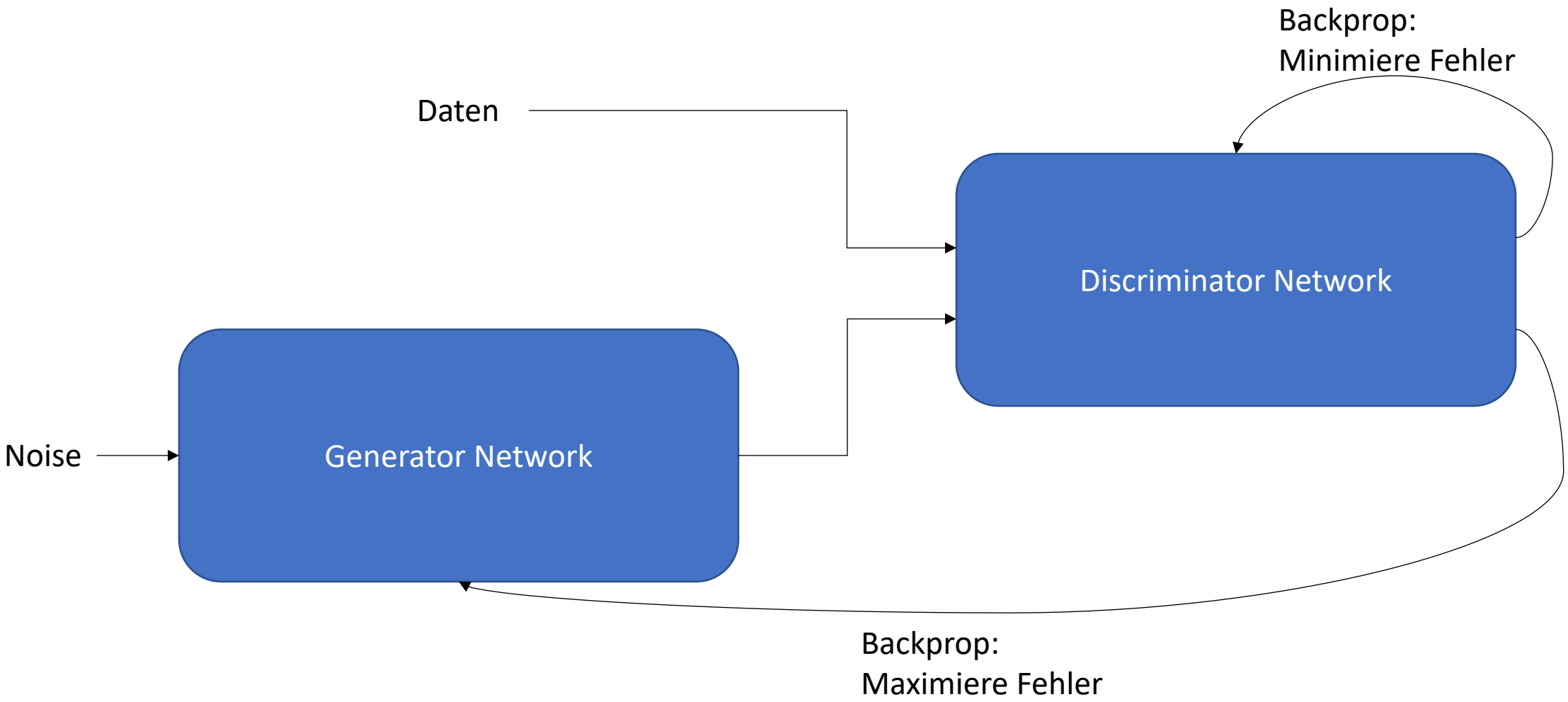
```
init = tf.global_variables_initializer()

n_epochs = 10
batch_size = 100

with tf.Session() as sess:
    init.run()
    for epoch in range(n_epochs):
        for iteration in range(mnist.train.num_examples // batch_size):
            X_batch, y_batch = mnist.train.next_batch(batch_size)
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
        acc_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
        acc_test = accuracy.eval(feed_dict={X: mnist.test.images, y: mnist.test.labels})
        print(epoch, "Train accuracy:", acc_train, "Test accuracy:", acc_test)
```

```
0 Train accuracy: 0.97 Test accuracy: 0.9789
1 Train accuracy: 0.99 Test accuracy: 0.9844
2 Train accuracy: 0.99 Test accuracy: 0.9867
3 Train accuracy: 0.99 Test accuracy: 0.9872
4 Train accuracy: 0.99 Test accuracy: 0.9879
5 Train accuracy: 0.99 Test accuracy: 0.9899
6 Train accuracy: 0.99 Test accuracy: 0.9885
7 Train accuracy: 1.0 Test accuracy: 0.9876
8 Train accuracy: 1.0 Test accuracy: 0.9891
9 Train accuracy: 1.0 Test accuracy: 0.9834
```

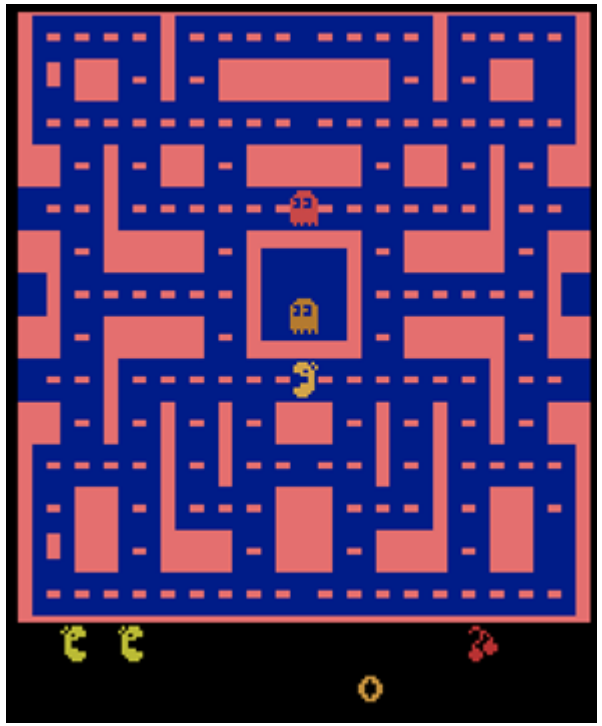
Generative Adversarial Networks



Generative Adversarial Networks

Quelle: https://towardsdatascience.com/mangagan-8362f06b9625

Reinforcement Learning



Convolutional Neural Network

Entscheidung für Bewegung von Pacman

Kostenfunktion anhand von Punkten im Spiel

What else?

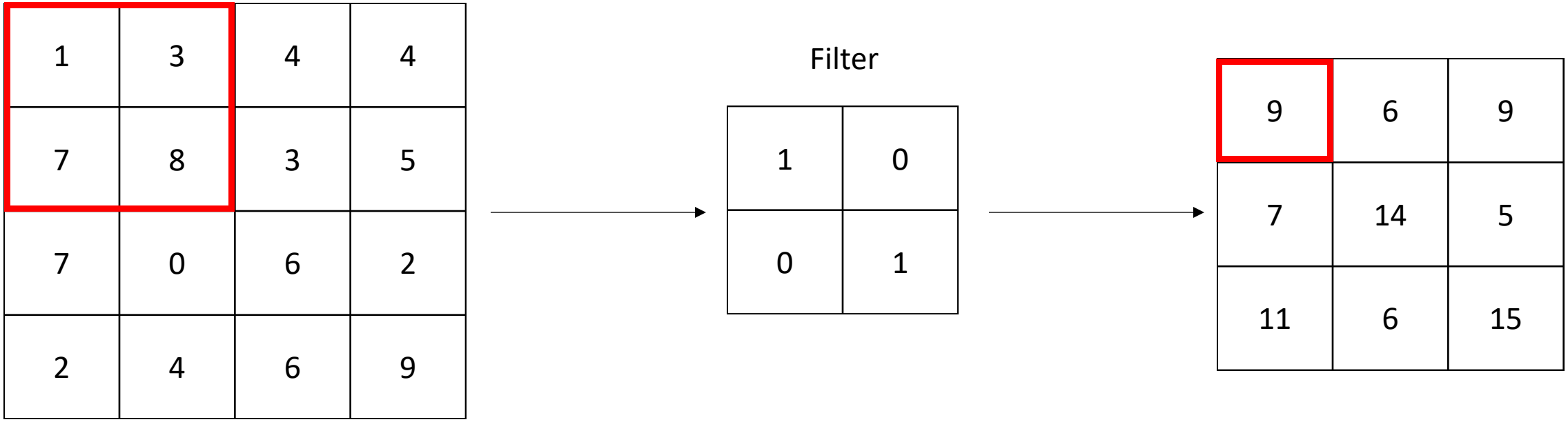
- Aktivierungsfunktion
- Batch Normalization
- Dropout
- Vortraining
- Optimierungsalgorithmen
- …

Vielen Dank für die Aufmerksamkeit

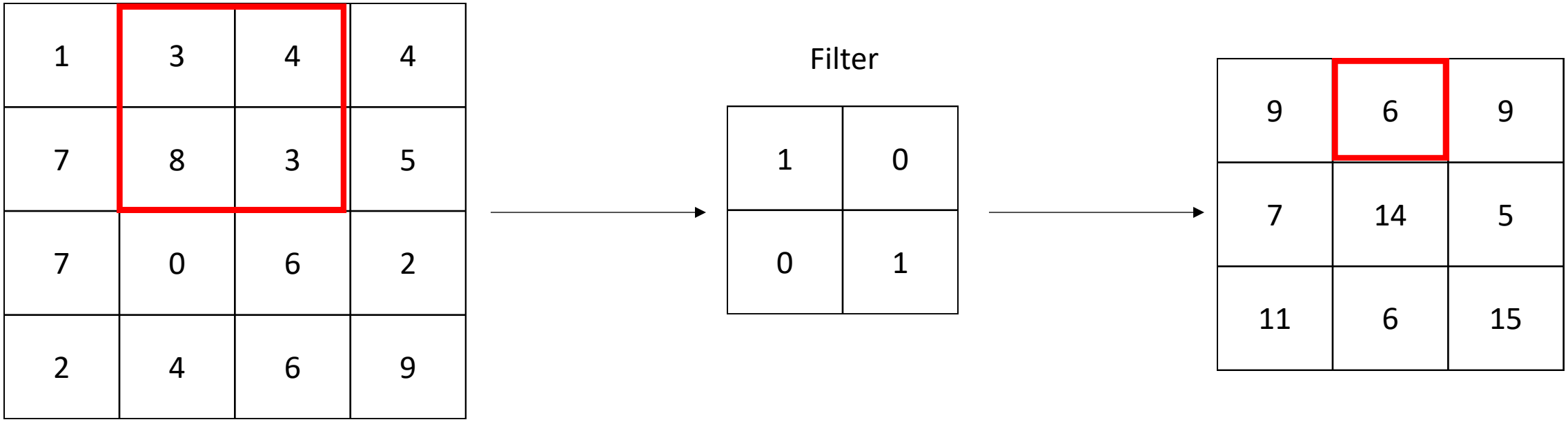
Nicolas Kuhaupt
Nicolaskuhaupt@aol.com
Data Scientist Fraunhofer IEE

Backup

Convolutional Neural Network – Stride



Convolutional Neural Network – Stride



Convolutional Neural Network – Stride

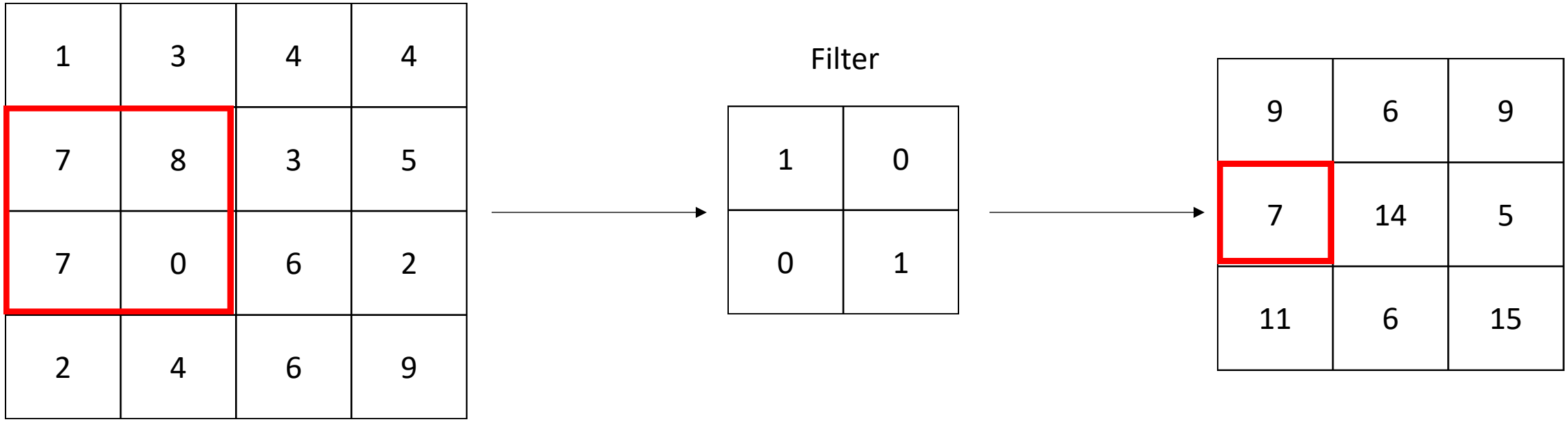
1	3	4	4
7	8	3	5
7	0	6	2
2	4	6	9

Filter

1	0
0	1

9	6	9
7	14	5
11	6	15

Convolutional Neural Network – Stride



Convolutional Neural Network – Padding

0	**0**	**0**	**0**	**0**	**0**
0	1	3	4	4	**0**
0	7	8	3	5	**0**
0	7	0	6	2	**0**
0	2	4	6	9	**0**
0	**0**	**0**	**0**	**0**	**0**

Filter

1	0	0
0	0	0
0	0	1

8	3	5	0
0	7	5	4
4	13	17	3
0	7	0	6

Convolutional Neural Network – Padding

0	**0**	**0**	**0**	**0**	**0**
0	1	3	4	4	**0**
0	7	8	3	5	**0**
0	7	0	6	2	**0**
0	2	4	6	9	**0**
0	**0**	**0**	**0**	**0**	**0**

Filter

1	0	0
0	0	0
0	0	1

8	3	5	0
0	7	5	4
4	13	17	3
0	7	0	6