

# Convolutional Neural Networks

Dozent: Dr. Zoran Nikolić

Machine Learning Seminar



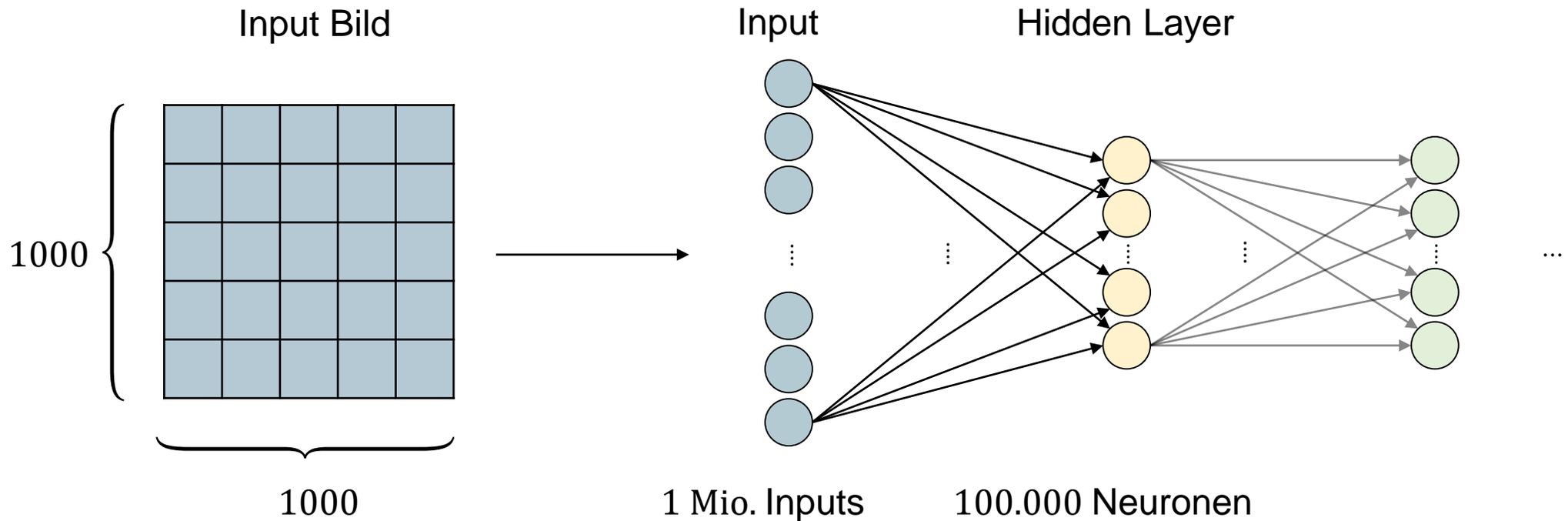
# Gliederung

- Der Convolution Operator
- Eigenschaften eines Convolutional Neural Network (CNN)
- Aufbau eines CNN
- Pooling-Layer
- Anwendung in Python

# Anwendung

- Bildererkennung
  - Autonomes Fahren (Klassifizierung der Verkehrszeichen)
  - Gesichts- und Objekterkennung
- Spracherkennung
  - Klassifizierung und Modellierung von Sätzen
  - Maschinelles Übersetzen

# Motivation

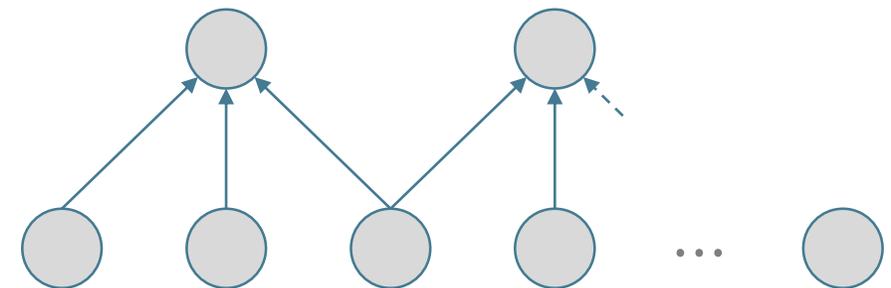
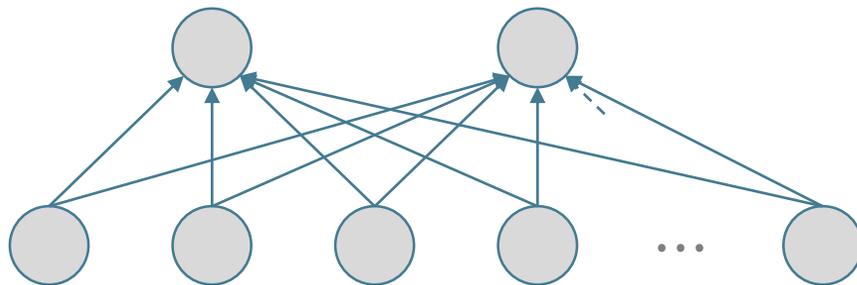


- $10^6 * 10^5 = 10^{11} = 100$  Milliarden Kanten/Gewichte
- Deep Learning: Vielfache
- Hoher Trainingsaufwand

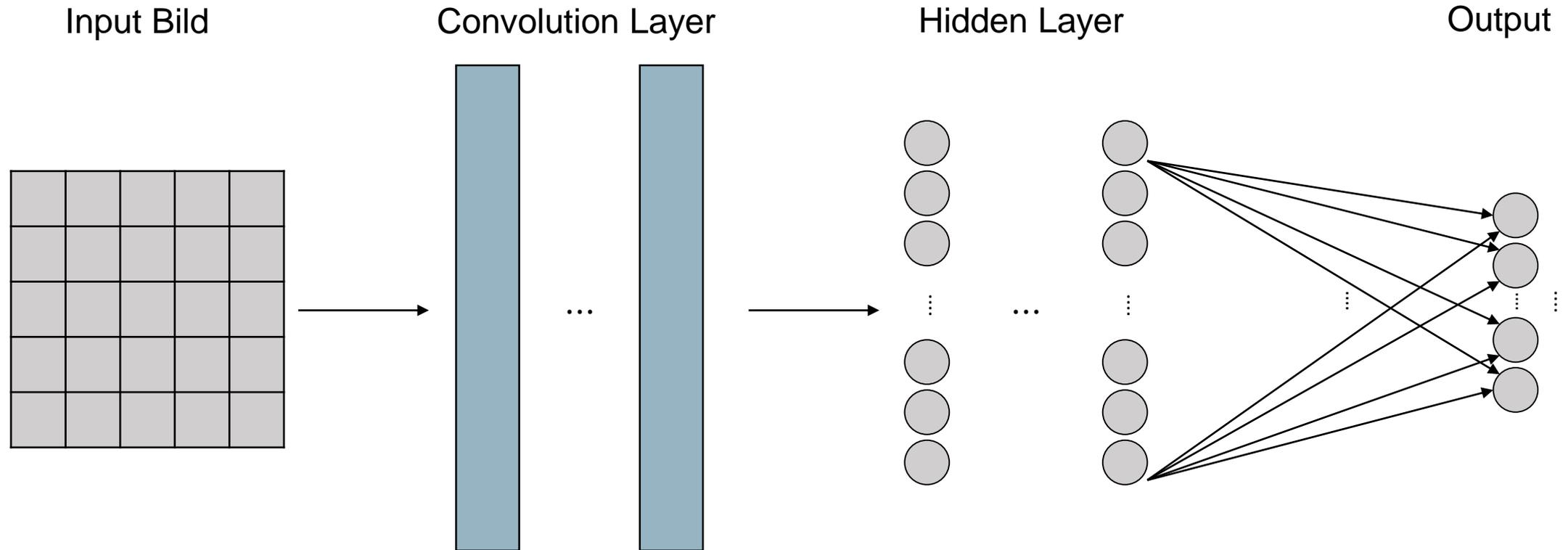
# Convolution Network

## Definition

- Convolutional Neuronal Network (CNN)
- Spezielle Form von neuronalen Netzen für Daten mit Gittertopologie
- Lokal verbundenes Netz
- Convolution (Faltung) statt Matrixmultiplikation



# Grober Aufbau



# Convolution Network

## Der Convolution Operator

- Faltung auf zwei reellen Funktionen

$$s(t) = (x * w)(t) = \int x(a)w(z - a) da$$

- $x$  = Input,  $w$  = Kern,  $s$  = Output = Feature Map
- In Praxis ist die Zeit meistens diskretisiert

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(z - a)$$

# Convolution Network

## Der Convolution Operator

- In Anwendungen mehrdimensionale Arrays als Input und Kern

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n)$$

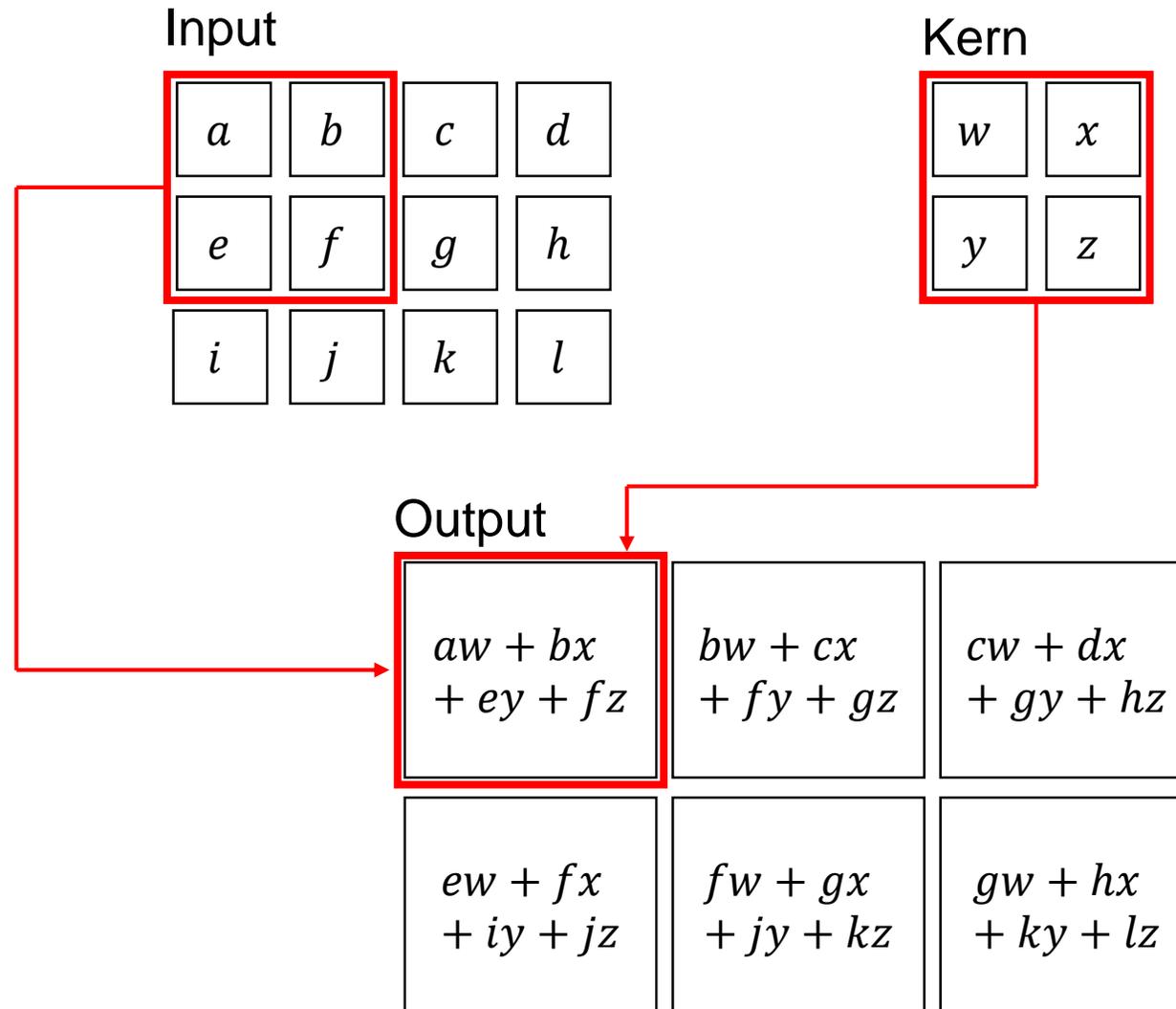
- Convolution Operator ist kommutativ

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n)$$

- Cross-Correlation

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n)$$

# Funktionsweise der Convolution



$$I \in \mathbb{K}^{m \times n}, K \in \mathbb{K}^{k \times k}$$

$$\Rightarrow K * I \in \mathbb{K}^{(m-k+1) \times (n-k+1)}$$

$$I \in \mathbb{K}^{3 \times 4}, K \in \mathbb{K}^{2 \times 2}$$

$$\Rightarrow K * I \in \mathbb{K}^{2 \times 3}$$

Input  $I$

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

Kern  $K$

1	0	-1
1	0	-1
1	0	-1

$K * I$

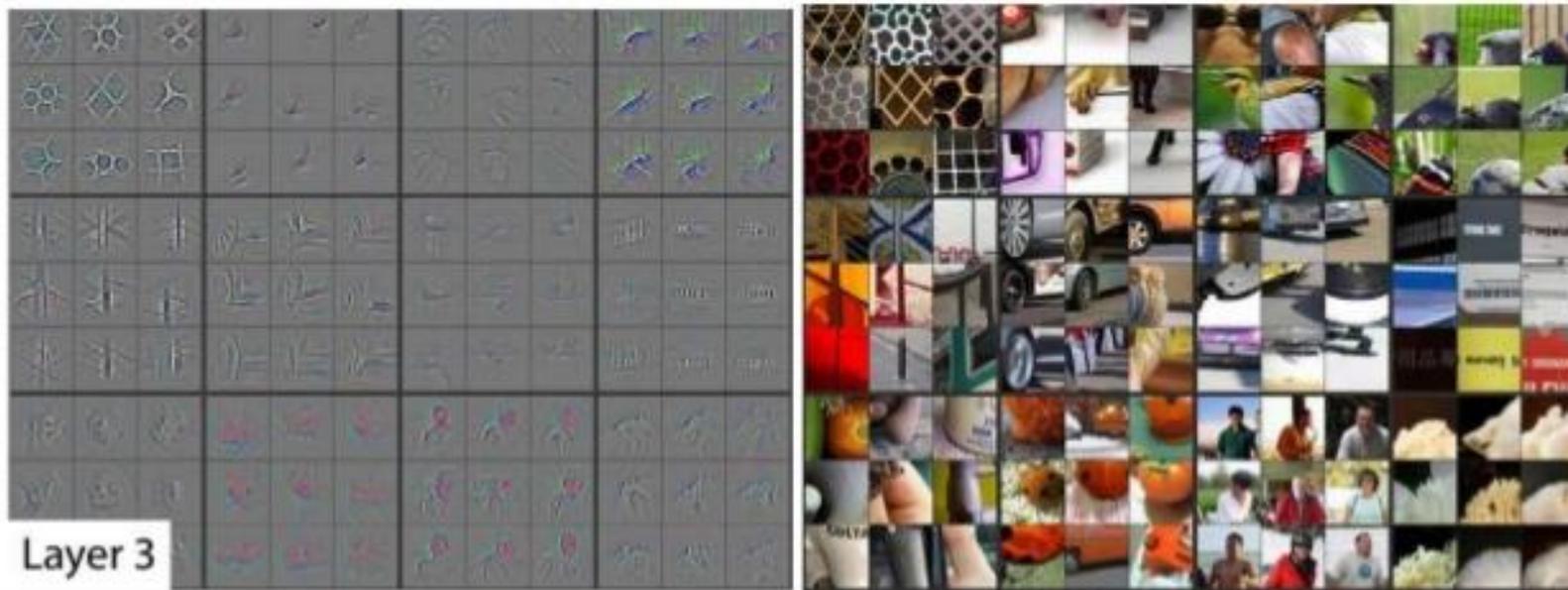
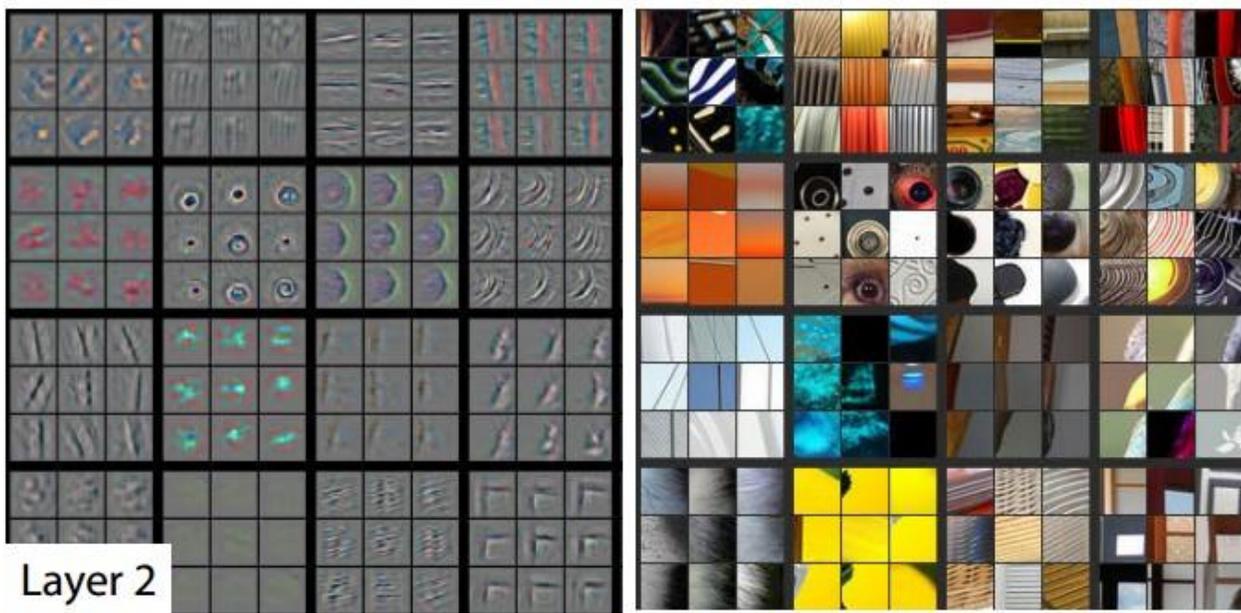
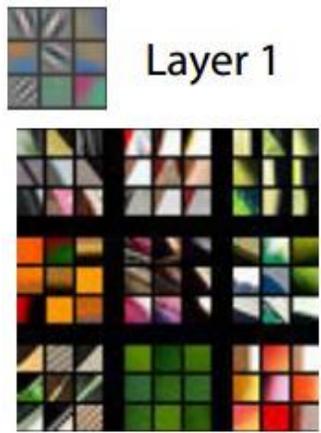
Output  $S$

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

$$S(1,1) = \sum_{m=-1}^1 \sum_{n=-1}^1 I(1+m, 1+n)K(m, n)$$

$$= 10 + 10 + 10 + 0 + 0 + 0 - 10 - 10 - 10 = 0$$

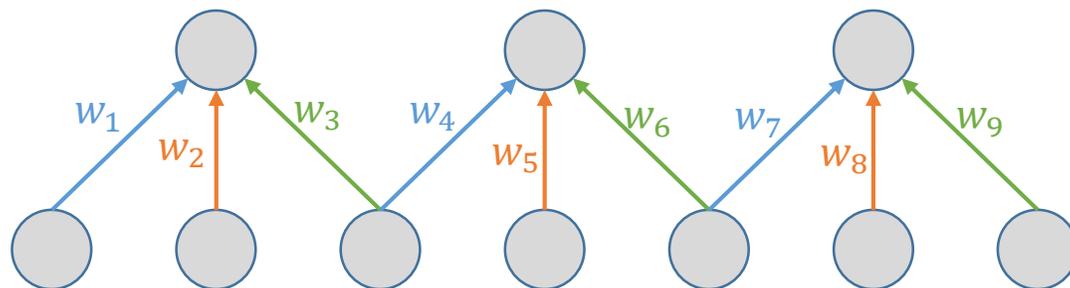
$\Rightarrow$  Kern hat Filterfunktion



# Ermittlung der Gewichte

## Backpropagation

- Kern Matrix ist nicht vorgegeben → muss gelernt werden
- Weniger Gewichte zu lernen als in einem normalen NN
- Mittels Backpropagation mit Nebenbedingung



$$w_1 = w_4 = w_7$$

$$w_2 = w_5 = w_8$$

$$w_3 = w_6 = w_9$$

# Erweiterungen

## Padding:

- Hinzufügen eines Randes aus Nullen  
→ Beibehalten der Größe

0	0	0	0	0
0				0
0				0
0				0
0	0	0	0	0

## Strided Convolution:

- Größere Schritte des Kerns über die Input Matrix  
→ Kleinerer Output
- $s$  = Größe der Schritte


# Eigenschaften eines CNN

## Sparse Interactions

NN:

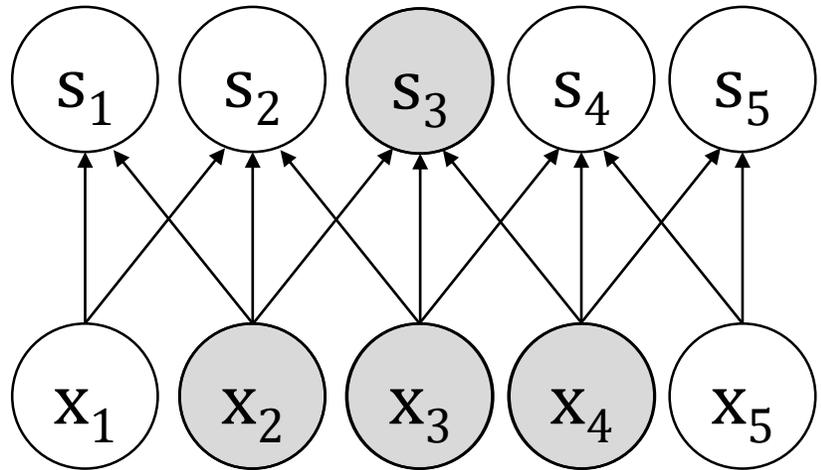
- Matrixmultiplikation
- Gewicht beschreibt Interaktion zwischen Output und Input
- Jeder Output interagiert mit jeder Input Einheit

CNN:

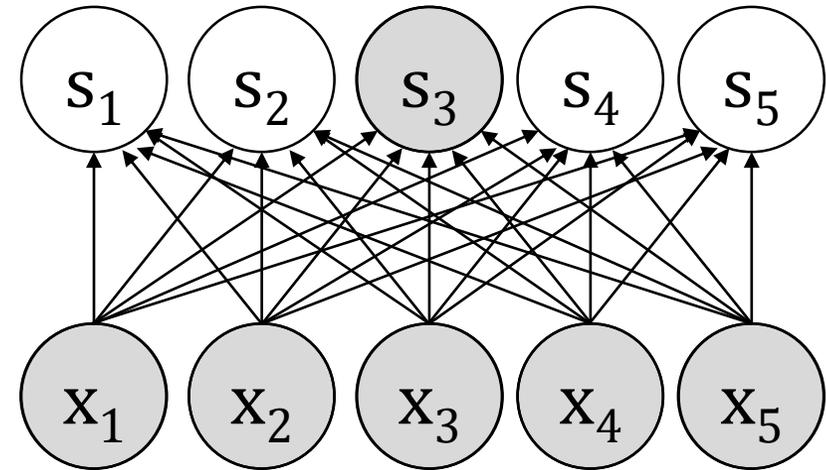
- Kern kleiner als Input
- Weniger Operationen zur Berechnung des Outputs
- Reduziert Speicherbedarf
- Verbesserungen der Effizienz

# Sparse Interactions

CNN vs. NN



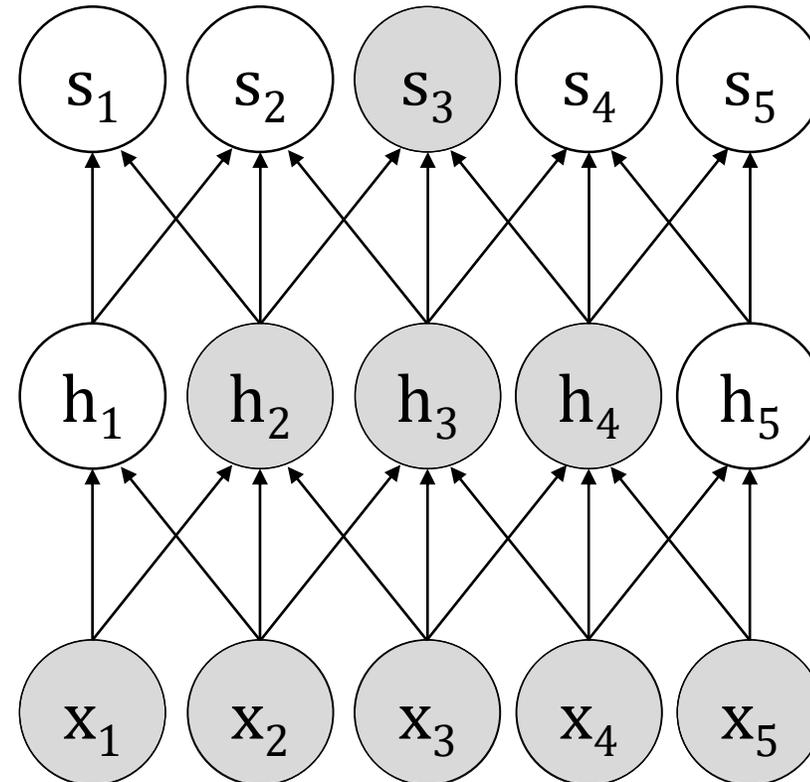
Output



Input

# Sparse Interactions

## Deeper Layer



- Tiefere Schichten können indirekt mit fast dem ganzen Input Bild verbunden sein

# Eigenschaften eines CNN

## Gewichts Sharing

NN:

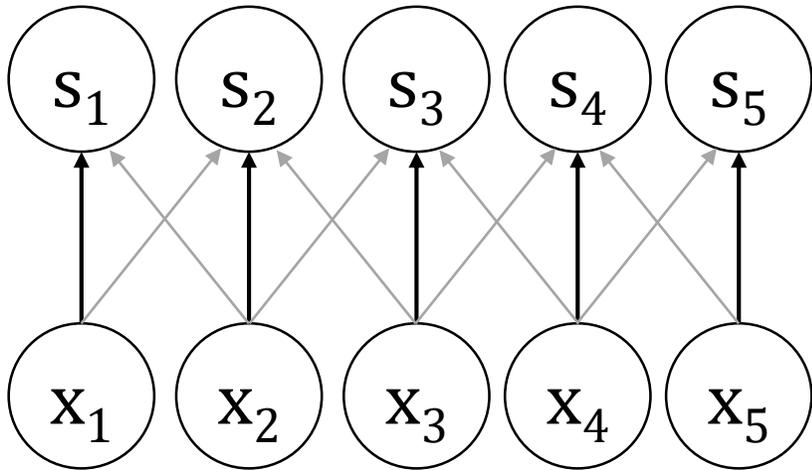
- Jedes Element der Gewichtsmatrix wird genau einmal benutzt um den Output der Schicht zu berechnen
- Multipliziert ein Element aus dem Input
- Danach nie wieder benutzt

CNN:

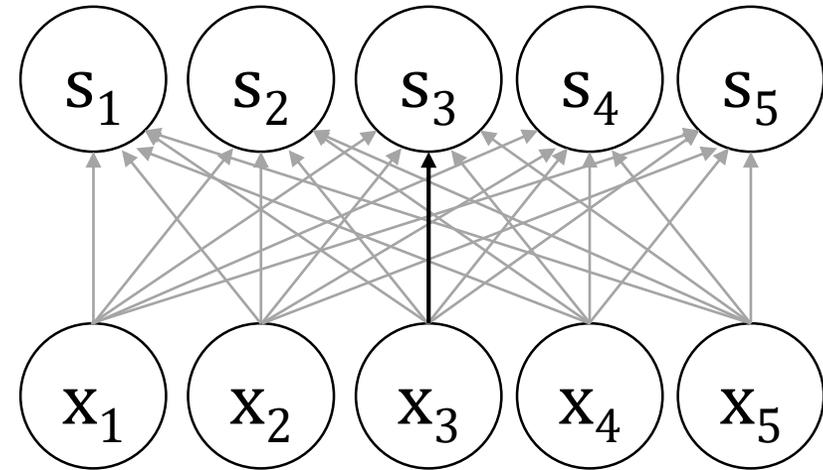
- Jeder Eintrag des Kerns wird an jeder Position des Inputs benutzt (außer an den Randpixeln)
- Ein Set Gewichte lernen statt ein separates für jede Stelle

# Gewichts Sharing

CNN vs. NN

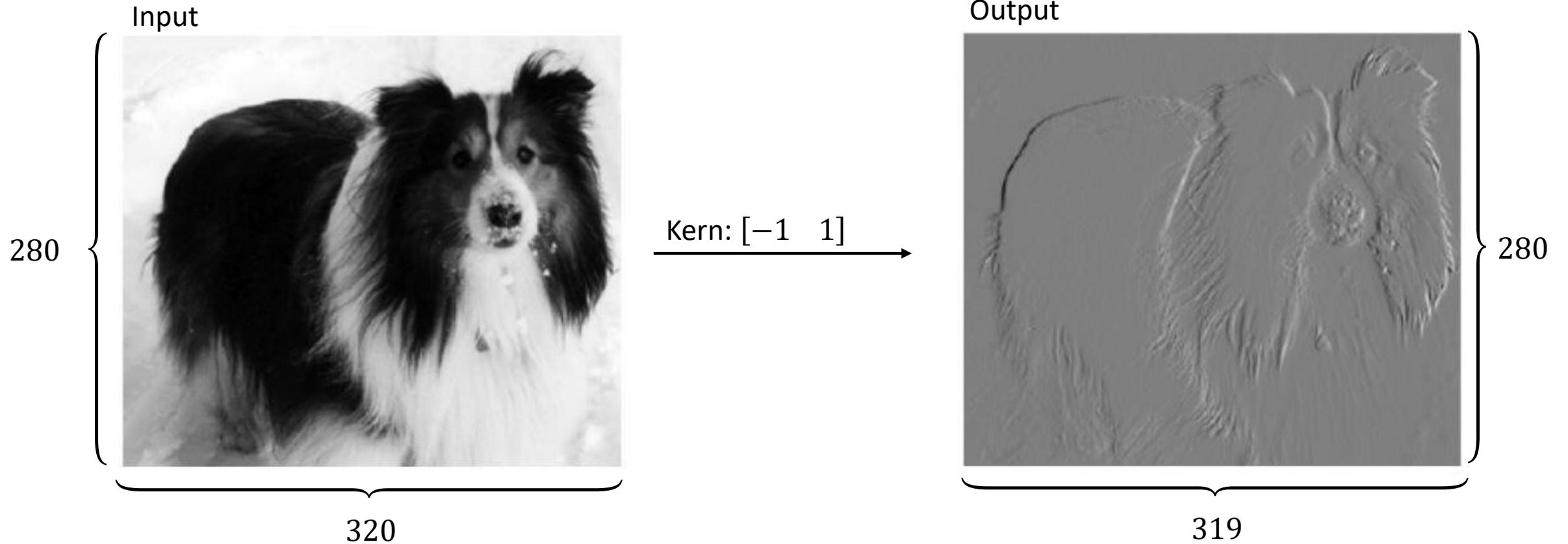


Output



Input

# Effizienz eines CNN



- $280 \cdot 320 \cdot 3 \approx 300.000$  Operationen

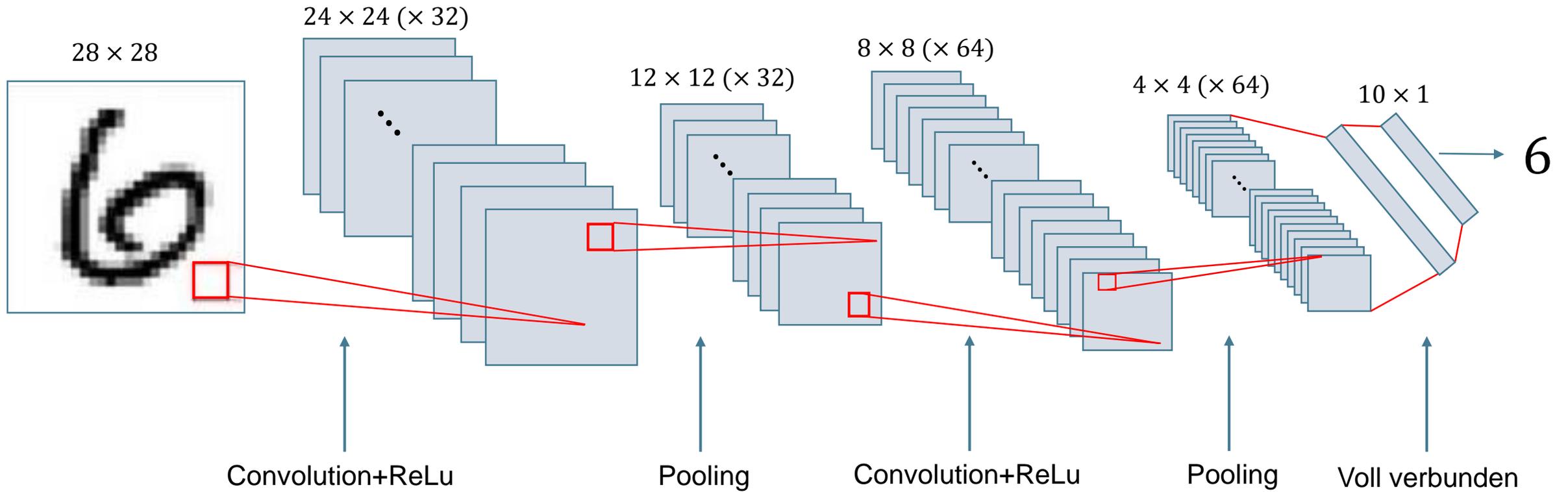
- $280 \cdot 320 \cdot 280 \cdot 319 \approx 8$  Bio. Operationen

# Aufbau eines CNN

3 Layer:

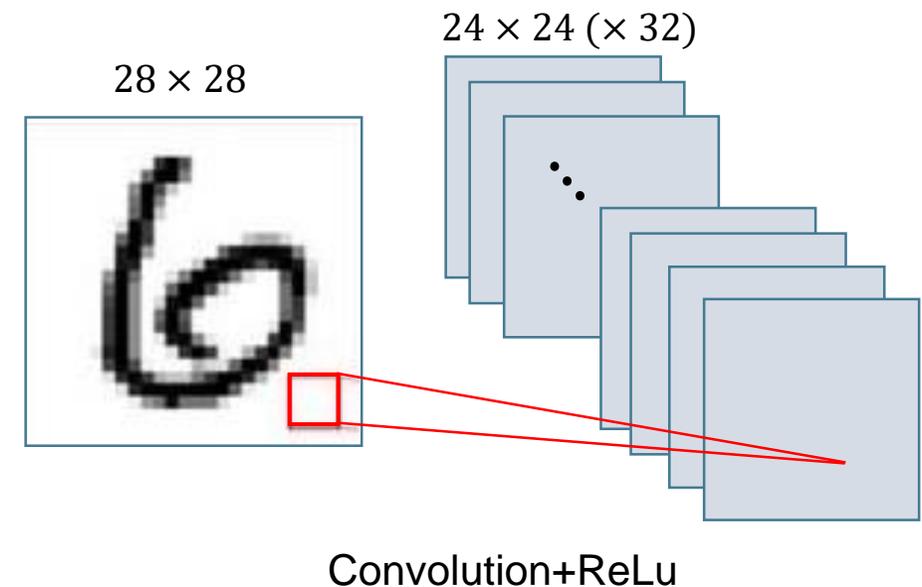
- **Convolution Layer** : Mehrere Convolution-Operatoren um eine Reihe von linearen Aktivierungen zu erzeugen
- **Detector Layer**: Jede lineare Aktivierung geht durch eine nichtlineare Aktivierungs Funktion, zB: ReLu
- **Pooling Layer**

# Aufbau

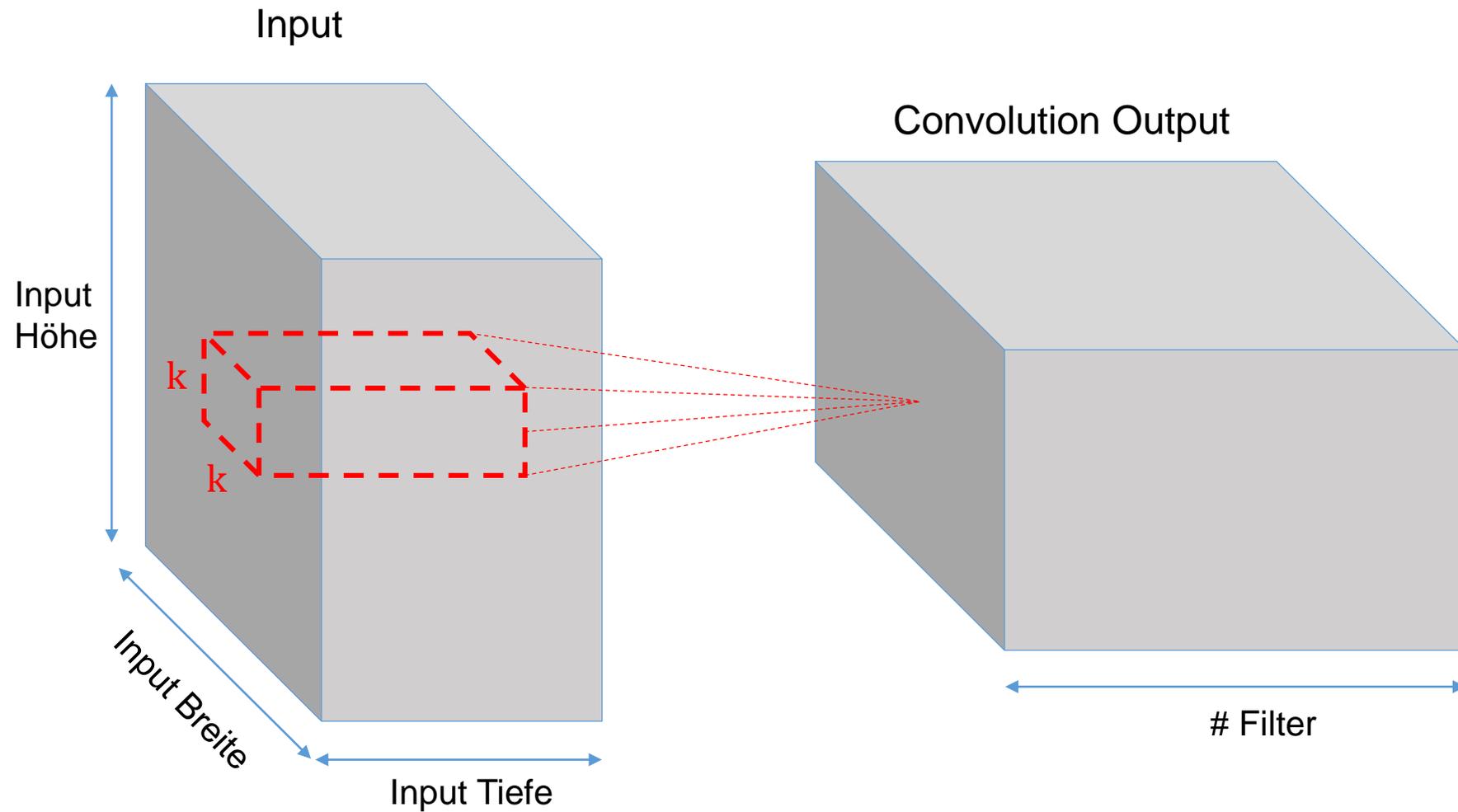


# Convolution Layer

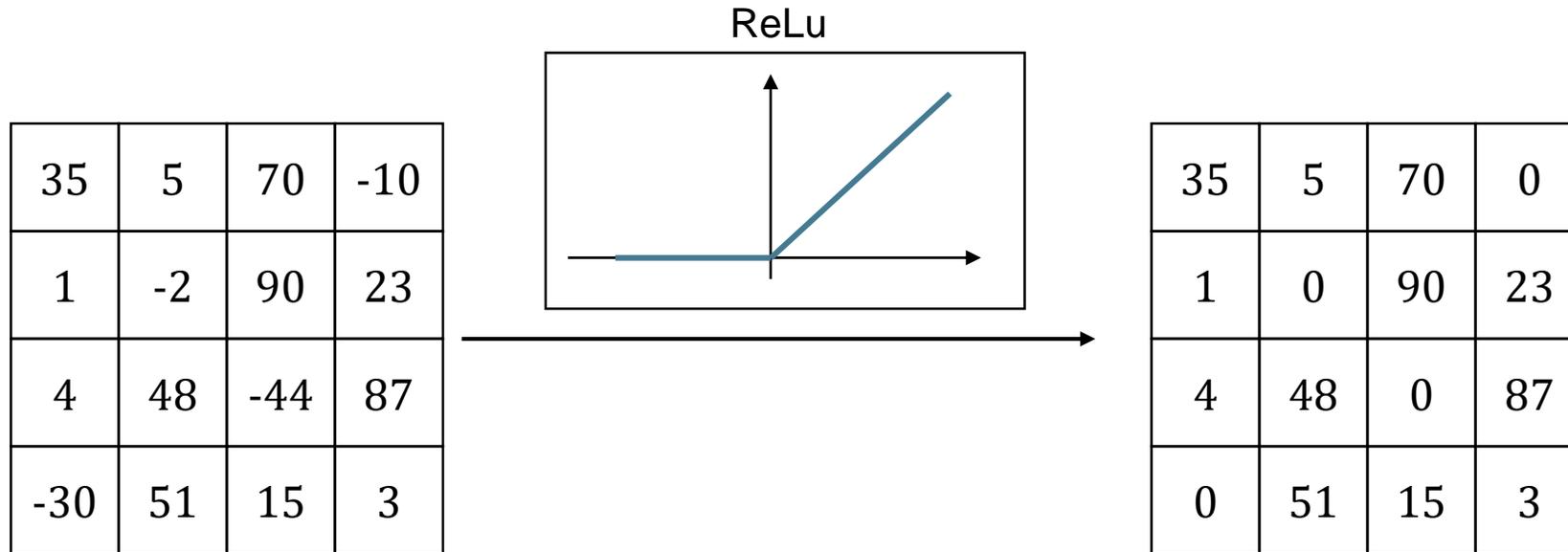
- Anwendung des Convolution Operators
- Mehrere Convolution Operatoren um eine Reihe von linearen Aktivierungen zu erzeugen
- Output dreidimensional



# Mehrdimensionale Convolution



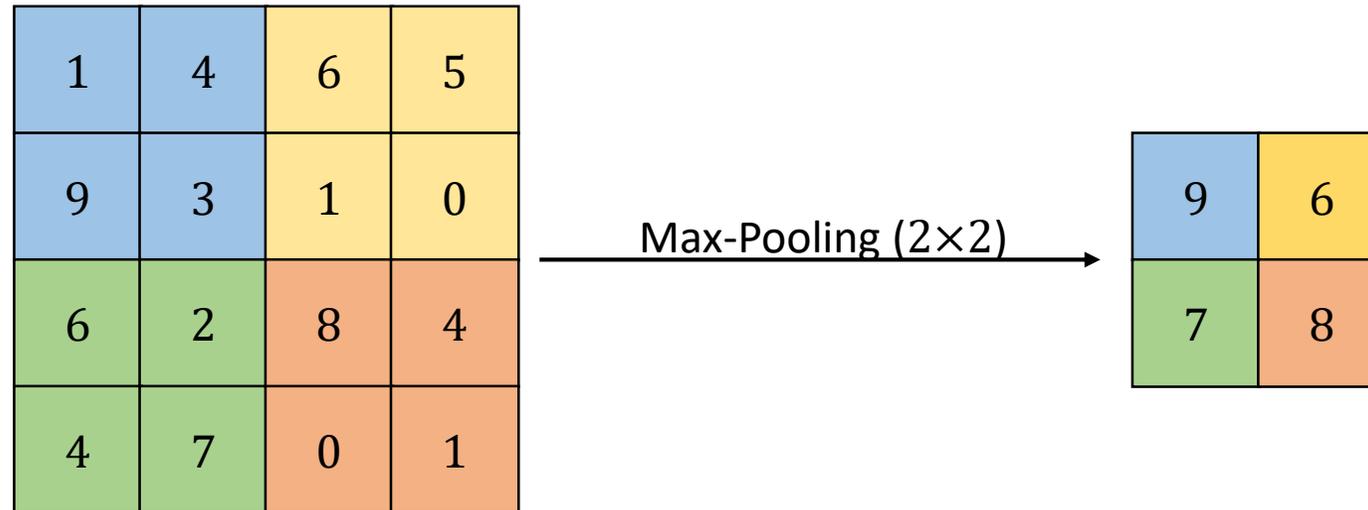
# Detector Layer



- Verwendet als Aktivierungsfunktion häufig ReLu mit  $f(x) = \max(0, x)$

# Pooling Layer

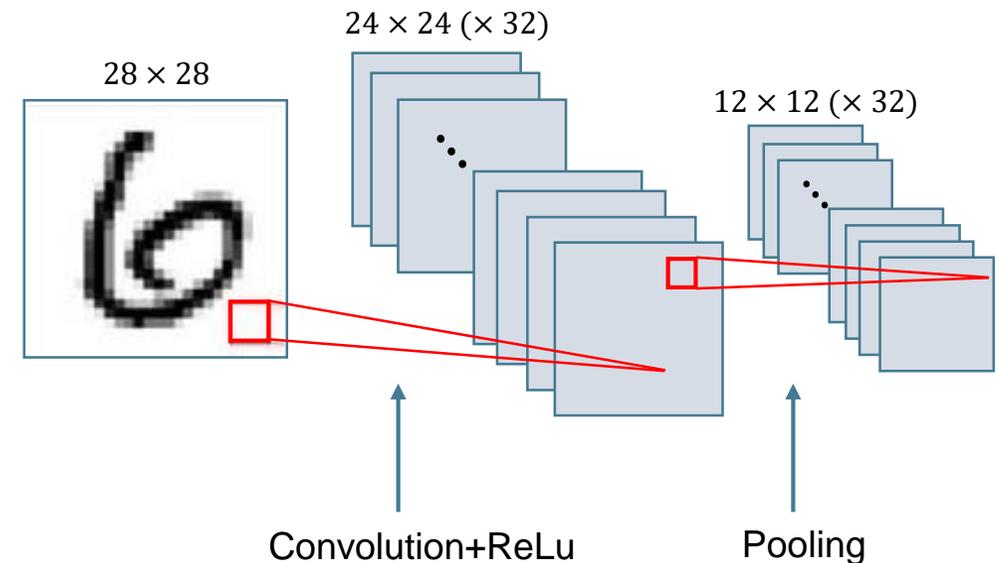
- Häufig: Max Pooling (Maximum in Umgebung),  $L_2$  Norm



- Ersetzt den Output des Netzes an einer bestimmten Stelle durch eine zusammenfassende Statistik der benachbarten Outputs
- Reduziert die Dimension
- Reduziert auf wesentliche Informationen

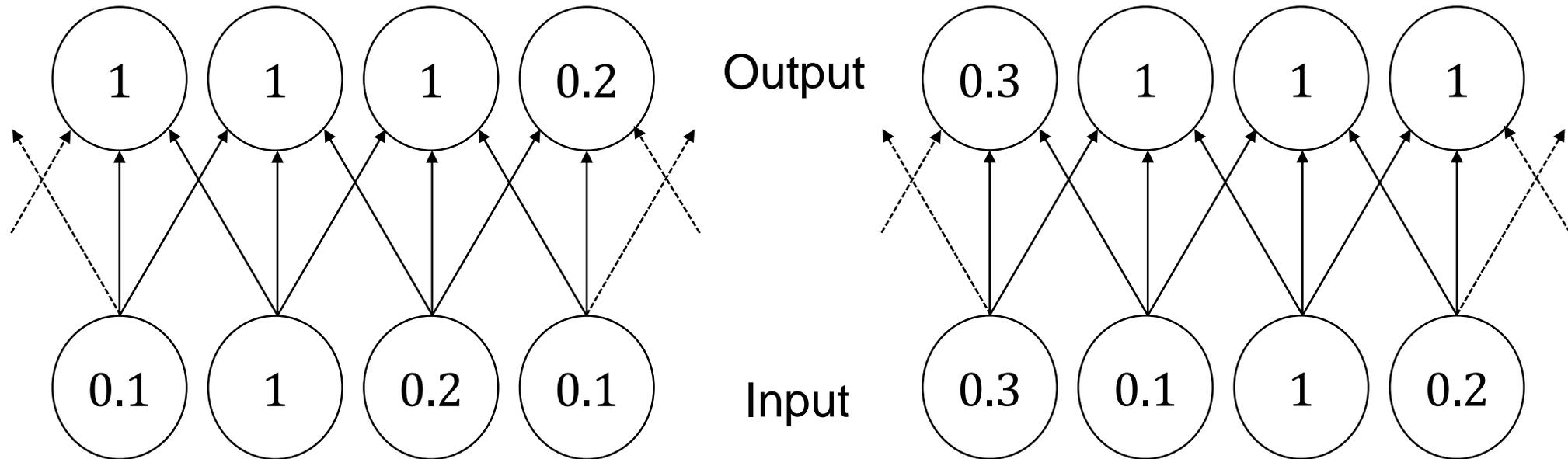
# Pooling Layer

- Erhöht Rechenleistung
- Meistens:  $2 \times 2$  Pooling ohne Überlappung → Bei größerem Pooling gehen häufig zu viele Informationen verloren
- Es müssen keine Gewichte gelernt werden
- Kontrolle von Overfitting



# Invarianz der Pooling Layer

Am Beispiel Max Pooling



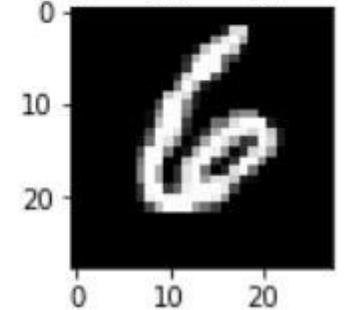
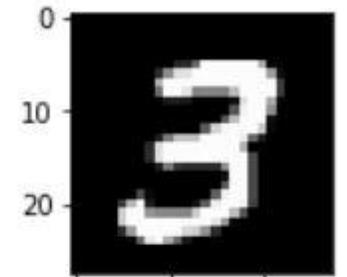
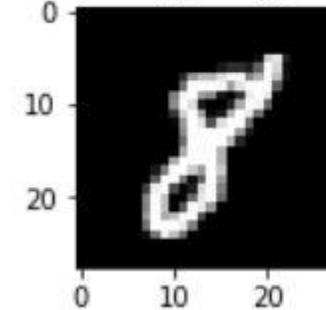
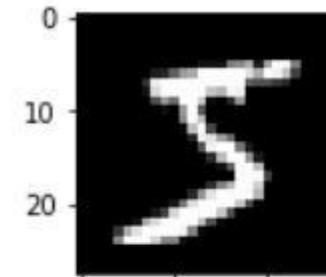
- Invariant gegenüber kleinen Verschiebungen im Input

# Aufbau Programm

## CNN in Python

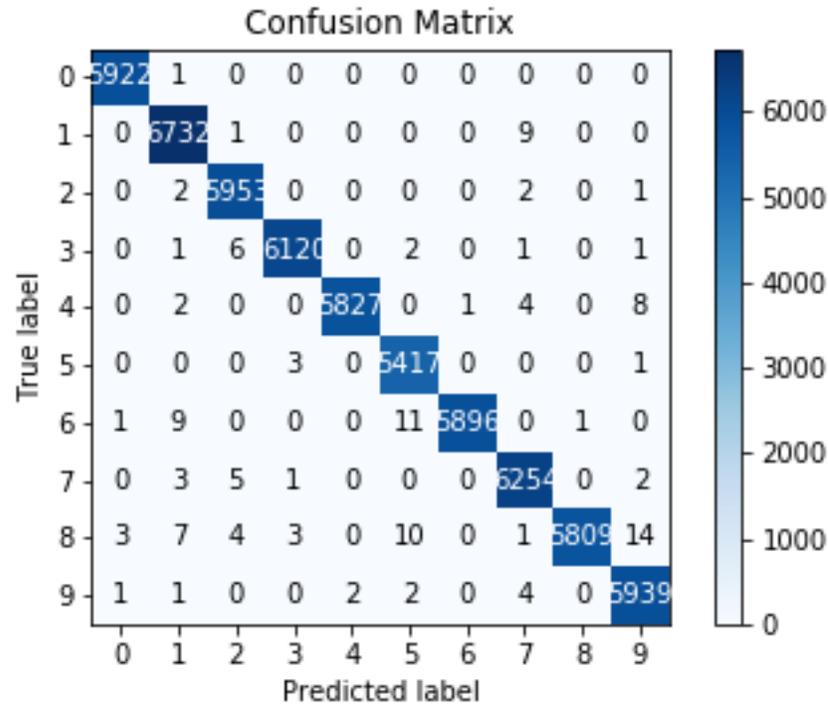
- MNIST Datensatz
- Keras in Python

```
# define the larger model
def larger_model():
    # create model
    model = Sequential()
    model.add(Conv2D(30, (5, 5), input_shape=(1, 28, 28), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(15, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(15, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

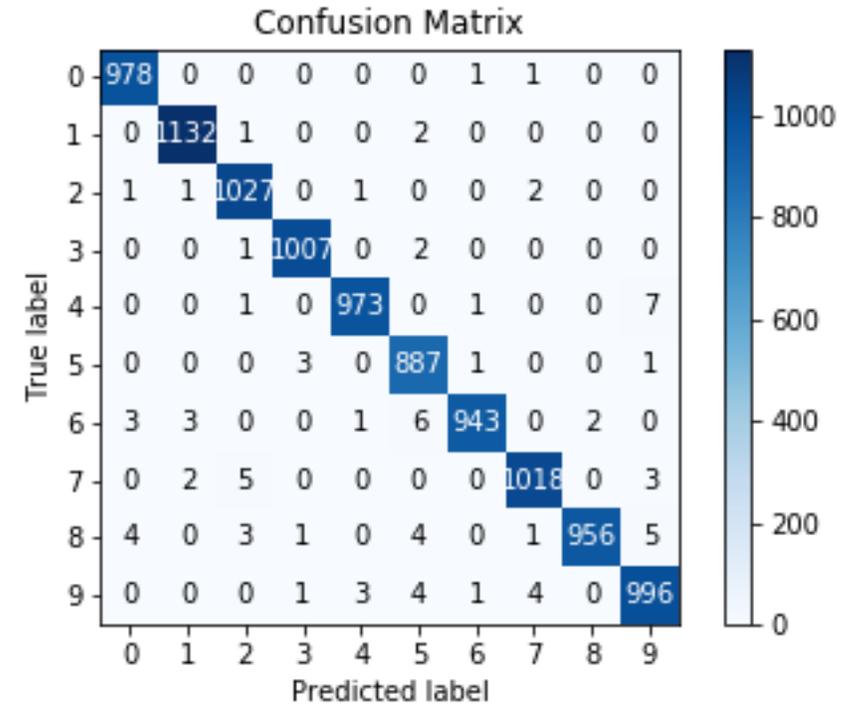


# Auswertung Programm

## CNN in Python



Trainingsgenauigkeit: 99,78%



Testgenauigkeit: 99,18%

# Auswertung Programm

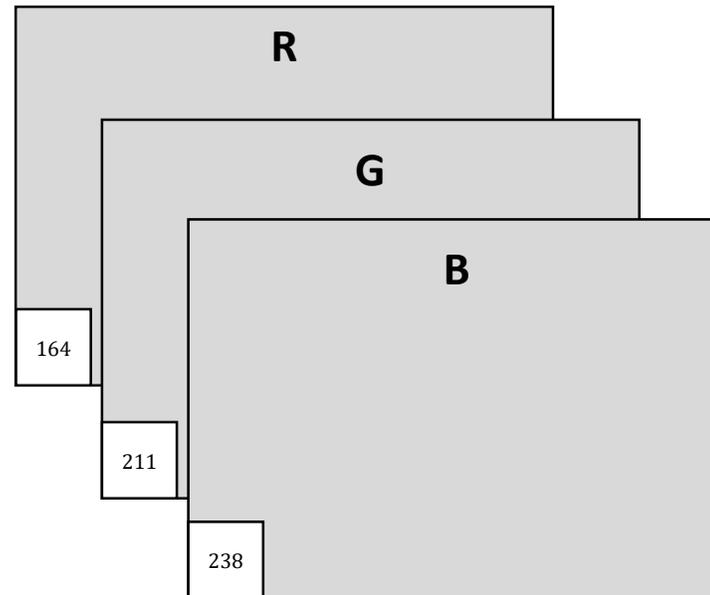
## CNN in Python

Modell	Trainingsgenauigkeit	Testgenauigkeit
<b>CNN:</b> 1 CNN Layer	99,92%	98,98%
<b>CNN:</b> 2 CNN Layer	99,78%	99,18%
<b>CNN:</b> 3 CNN Layer	99,71%	97,61%
<b>NN:</b> 1 Hidden Layer	99,72%	98,18%
<b>NN:</b> 2 Hidden Layer	99,23%	97,35%
<b>NN:</b> 6 Hidden Layer	99,41%	97,75%

# Zusatz: RGB Bild

R	G	B	Farb-Beispiel
255	165	000	
164	211	238	
154	205	050	

- Bisher: Input Bild der Größe Höhenpixel  $\times$  Breitenpixel
- Damit können nur Schwarz weiß Bilder dargestellt werden
- Buntes Bild hat pro Pixel 3 Werte:



$\Rightarrow$  Größe Input = Höhe  $\times$  Breite  $\times$  3

$\Rightarrow$  Kern hat Dimension  $k \times k \times 3$

# Fazit

- State-of-the-art in Bild- und Audioverarbeitung
- Fehlerraten: Teilweise besser als Mensch
- Mit Grafikprozessoren lassen sich CNN sehr effizient trainieren
- Vorteile CNN:
  - Robustheit
  - Weniger Speicherplatzbedarf
  - Einfacheres und besseres Training

# Literatur

- [AH] H. Aghdam, E. Heravi. Guide to Convolutional Neural Networks, Springer, 2017
- [GBC] I. Goodfellow, Y. Bengio, A. Courville. Deep Learning, MIT Press, 2016, <http://www.deeplearningbook.org>
- [QV] Quoc V Le et al. A tutorial on deep learning part 2: Autoencoder, convolutional neural networks and recurrent neural networks, Google Brain, 2015, <http://https://cs.stanford.edu/~quocle/tutorial2.pdf>