

# Faltende Neuronen<sup>1</sup>

## Vorlesung 8, Artificial Intelligence

Dozenten:

Prof. Dr. M. O. Franz, Prof. Dr. O. Bittel, Prof. Dr. O. Dürr

HTWG Konstanz, Fakultät für Informatik

---

<sup>1</sup>Folien basieren z.T. auf B. Sick, O. Dürr, DL Course.

# Übersicht

- 1 Voll verbundene Netzwerkarchitekturen bei Bildern
- 2 Faltungsschichten

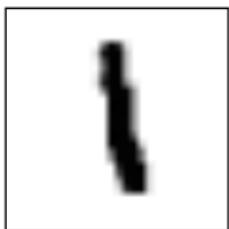
# Übersicht

1 Voll verbundene Netzwerkarchitekturen bei Bildern

2 Faltungsschichten

# Beispiel: Voll verbundenes Netzwerk trainiert auf MNIST

MNIST Handwritten Digits



$\approx$

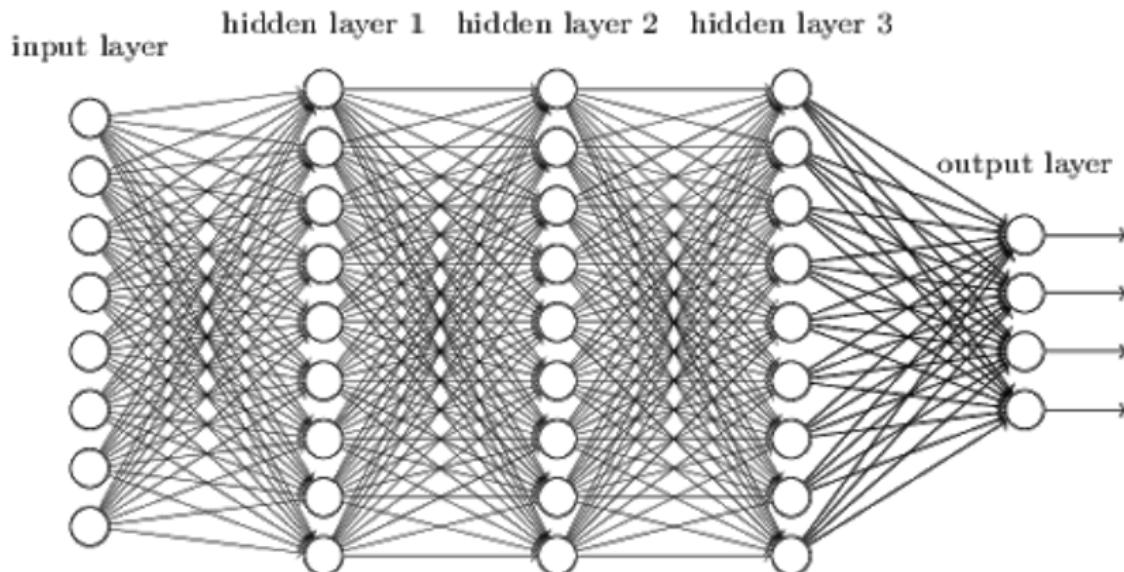
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$28 \times 28 = 784$  features

Row Names	Digit	Pixel_1	Pixel_2	Pixel_3	Pixel_4	Pixel_5	Pixel_256
Sample16	0	0	0	0	0	1	0
Sample78	8	0	1	1	1	1	0
Sample79	3	1	1	1	1	1	...
Sample80	2	0	0	0	1	1	1
Sample81	1	0	0	0	0	0	0
Sample82	2	0	0	0	0	1	1
Sample83	4	0	0	0	0	1	1

60'000  
cases

# Voll verbundene Netzwerkarchitektur



[Nielsen, Neural Networks and Deep Learning, 2015]

Jedes Neuron ist mit allen Neuronen seiner Eingangs- und Ausgangsschicht verbunden. In unserem Beispiel haben wir 784 Eingangsneuronen, 2 verdeckte Schichten mit 100 bzw. 50 Neuronen und 10 Ausgangsneuronen (eins für jede Klasse).

## Kreuzentropie

Bei Klassifikationsproblemen wird bei neuronalen Netzen für die Labels gern **One-Hot-Coding** verwendet: bei  $K$  Klassen besteht der Output des Netzes aus  $K$  Neuronen. Die zugehörigen Labels sind ebenfalls  $K$ -dimensionale Vektoren, wobei alle Einträge 0 sind, nur der Eintrag der korrekten Klasse ist 1.

Eine Kostenfunktion für diesen Fall ist die **Kreuzentropie** (zunächst für ein einzelnes Ausgangsneuron mit Output  $\hat{y}_j(x_i)$  für Input  $x_i$ ,  $i = 1 \dots n$ ):

$$C_j = -\frac{1}{n} \sum_{i=1}^n [y_i \ln \hat{y}_j(x_i) + (1 - y_i) \ln(1 - \hat{y}_j(x_i))].$$

$C$  ist immer positiv, und wenn  $\hat{y}(x_i)$  nahe  $y_i$  liegt, wird  $C$  minimal.

Erweiterung der Kreuzentropie auf mehrere Outputneuronen  $a_j$ :

$$C = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^K [y_{j,i} \ln \hat{y}_j(x_i) + (1 - y_{j,i}) \ln(1 - \hat{y}_j(x_i))].$$

# Softmax-Nichtlinearität

Statt der Sigmoidfunktion wird häufig die **Softmax-Nichtlinearität** als Aktivierungsfunktion in der letzten Schicht eingesetzt:

$$\hat{y}_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{für } j = 1 \dots K \quad (\text{Anzahl der Klassen})$$

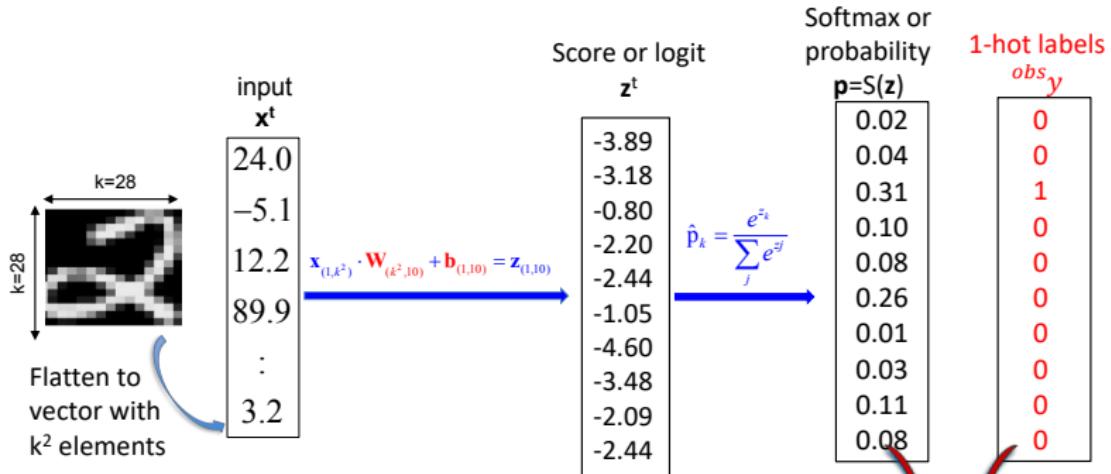
Alle Aktivierungen sind immer positiv und summieren sich zu

$$\sum_{j=1}^K \hat{y}_j = \sum_{j=1}^K \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} = \frac{\sum_{j=1}^K e^{z_j}}{\sum_{k=1}^K e^{z_k}} = 1,$$

d.h. der Output der Softmax-Schicht ist eine **Wahrscheinlichkeit**.

Damit kann die Aktivierung des  $j$ -ten Neurons als die vom Netzwerk geschätzte Wahrscheinlichkeit dafür interpretiert werden, dass die  $j$ -te Klasse vorliegt.

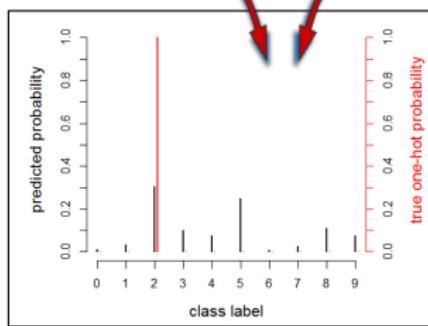
# Was passiert beim Training des Netzwerks?



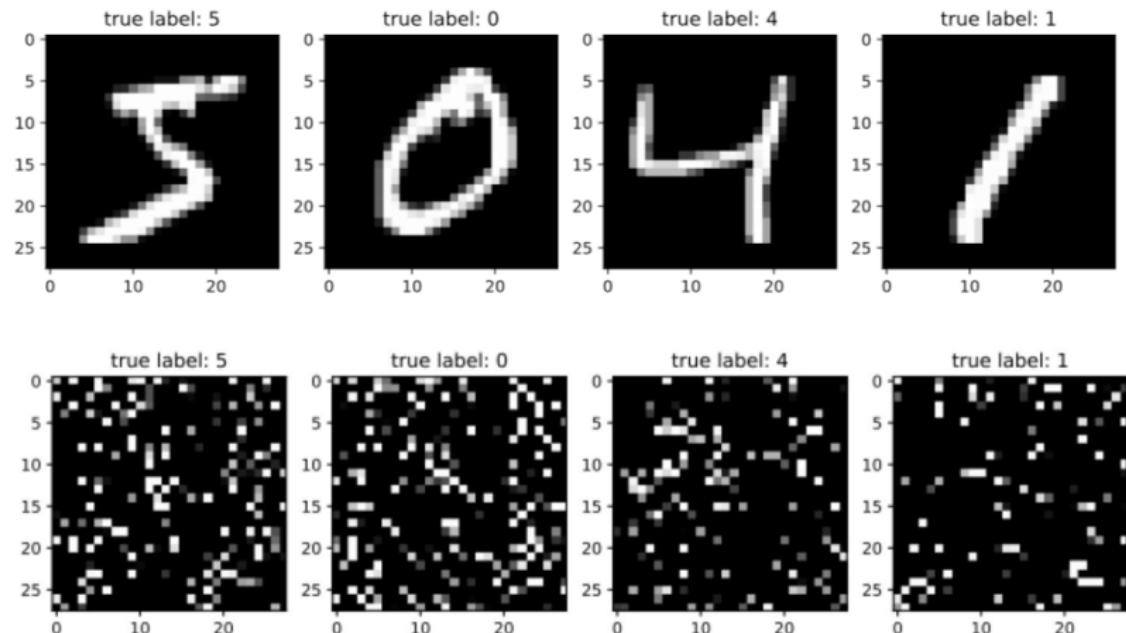
Cost C or Loss = cross-entropy averaged over all images in mini-batch

$$C = \frac{1}{N} \sum_i D(\mathbf{p}_i, \mathbf{y}_i) \quad D(\mathbf{p}, \mathbf{y}) = -\sum_{k=1}^{10} y_k \cdot \log(p_k)$$

Cross-Entropy



# Experiment: Stören Pixelpermutationen das Training?



[Demo]

# Übersicht

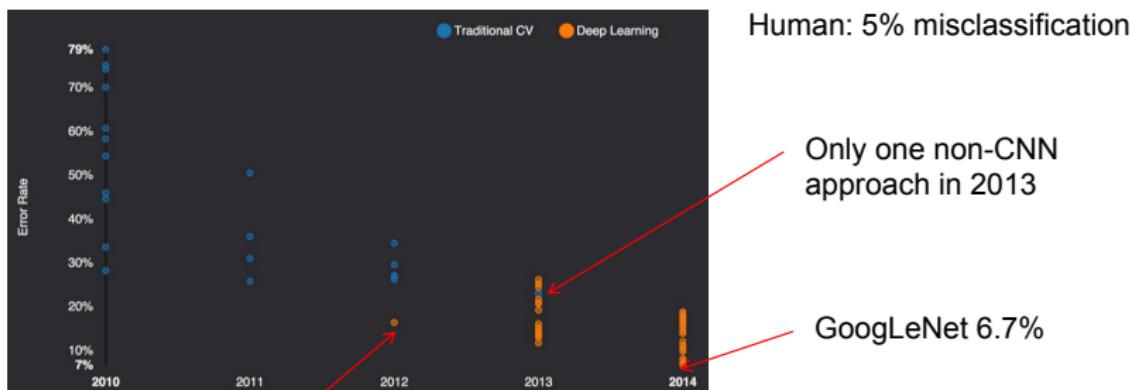
- 1 Voll verbundene Netzwerkarchitekturen bei Bildern
- 2 Faltungsschichten

# Motivation: Imagenet Challenge

1000 classes  
1 Mio samples



...



Human: 5% misclassification

Only one non-CNN approach in 2013

GoogLeNet 6.7%

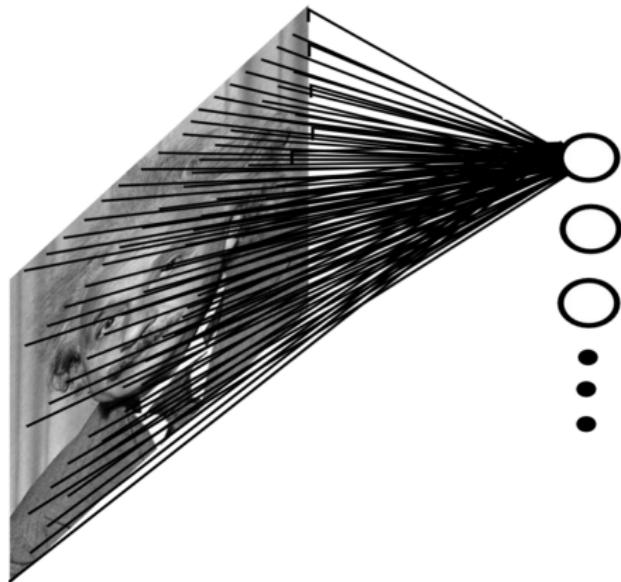
A. Krizhevsky  
first CNN in 2012

2015: It gets tougher

- 4.95% Microsoft ([Feb 6](#) surpassing human performance 5.1%)
- 4.8% Google ([Feb 11](#)) -> further improved to 3.6 (Dec)?
- 4.58% Baidu (May 11 [banned due to many submissions](#))
- 3.57% Microsoft (Resnet winner 2015)

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

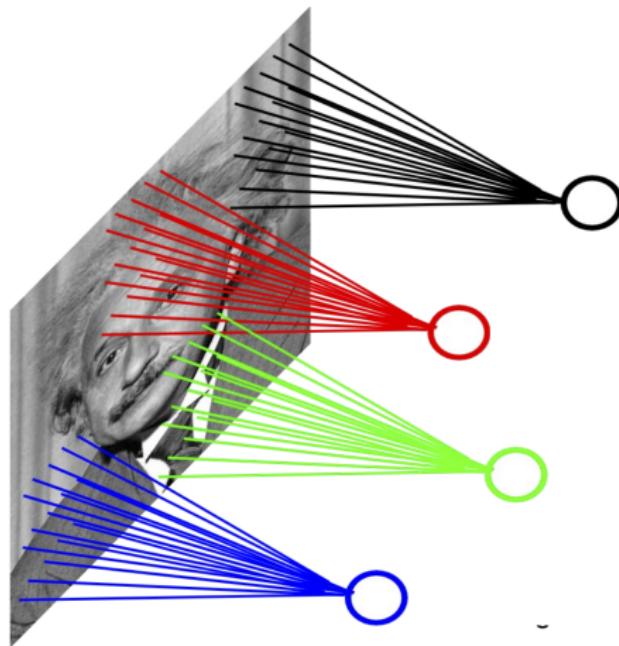
# Gefahr des Overfittings in MLPs



- In einem traditionellen MLP ist jedes Neuron mit allen Neuronen der vorangehenden Schicht verbunden.
- Ein  $200 \times 200$  Eingangsbild mit 40.000 Neuronen in der verdeckten Schicht bräuchte 2 Mrd. Gewichte!
- Es gibt selten genug Trainingsbeispiele, um so viele Parameter festzulegen  $\Rightarrow$  Overfitting.
- Korrelationen im Bild sind i. W. nur auf die direkte Nachbarschaft beschränkt.

[Ranzato, 2014]

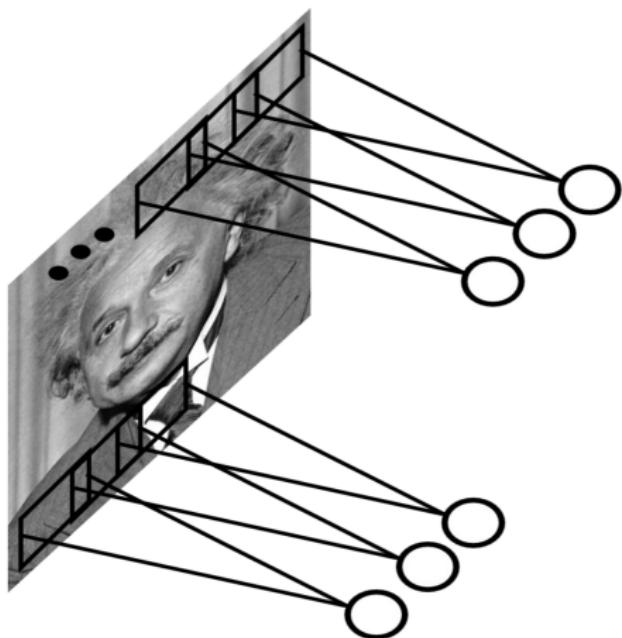
# Möglicher Ansatz: lokal verknüpfte Neuronen



- Ein  $200 \times 200$  Eingangsbild mit 40.000 Neuronen in der verdeckten Schicht und  $10 \times 10$  rezeptivem Feld braucht nur noch 4 Mio. Parameter.
- Funktioniert gut, wenn alle Bilder genau registriert sind.
- Die lokale Statistik in verschiedenen Bildausschnitten ist oft sehr ähnlich. Reicht vielleicht ein einheitlicher Gewichtssatz?

[Ranzato, 2014]

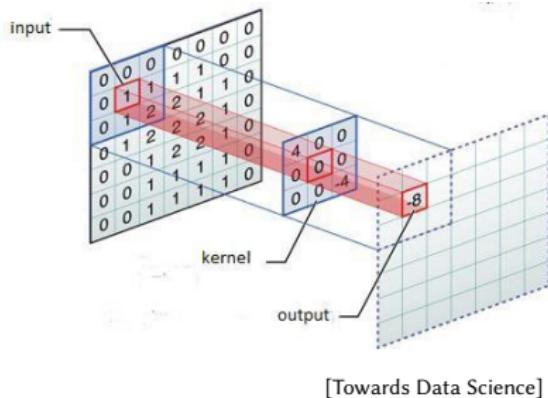
# Faltung durch Neuronen (Faltungsschicht, Convolutional Layer)



[Ranzato, 2014]

- Jedes Neuron der verdeckten Schicht teilt sich seine Gewichte mit allen anderen: nur noch 100 Gewichte!
- Die rezeptiven Felder überlappen sich.
- Entspricht einer Korrelation mit einer **gelernten** Filtermaske.
- Je nach Randbehandlung und Überlappungsgrad der Fenster sind die Faltungsschichten gleich groß oder kleiner als die Eingangsschicht.

# Faltendes Neuron (Convolutional Neuron)



Berechnung wie Standardperzepron, allerdings beschränkt sich der Input auf ein  $n \times m$ -Fenster:

$$z_{ij} = \sum_{i'=\lfloor -n/2 \rfloor}^{\lfloor n/2 \rfloor} \sum_{j'=\lfloor -m/2 \rfloor}^{\lfloor m/2 \rfloor} w_{i', j'} x_{i+i', j+j'} + b$$

$$a_{ij} = \sigma(z_{ij})$$

$\sigma()$ : Aktivierungsfunktion (z.B. ReLU)

Statt einer Schicht identischer Neuronen kann man sich auch vorstellen, dass das faltende Neuron sein Eingabefenster über das Eingangsbild Zeile für Zeile scannt. Für jedes Fenster werden die Gewichte des Filtermaske punktweise mit den Pixelwerten multipliziert. Der Output ist ebenfalls ein Bild (Merkmals- oder Aktivierungskarte, **feature map**). Die Position in der Aktivierungskarte entspricht der Position des Zentrums des Fensters im Eingangsbild.

## Aufgabe: Berechnen Sie einen Faltungsschritt von Hand!

Berechnen Sie das Ergebnis der Faltung für die eingezeichnete Fensterposition. Die Gewichte der Filtermaske sind die kleiner gedruckten Werte im Fenster. Wie groß ist die gesamte Aktivierungskarte? An welche Position in der Karte wird der neuronale Output geschrieben?

3	3	2	1	0
0 <sub>0</sub>	0 <sub>1</sub>	1 <sub>2</sub>	3	1
3 <sub>2</sub>	1 <sub>2</sub>	2 <sub>0</sub>	2	3
2 <sub>0</sub>	0 <sub>1</sub>	0 <sub>2</sub>	2	2
2	0	0	0	1

# Ein Filter sucht nach einem spezifischen lokalen Merkmal



image patch						
0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

filter/kernel: curve detector

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

$$= 6600$$

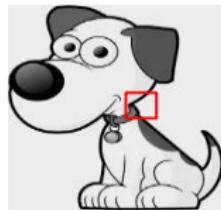


image patch						
0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

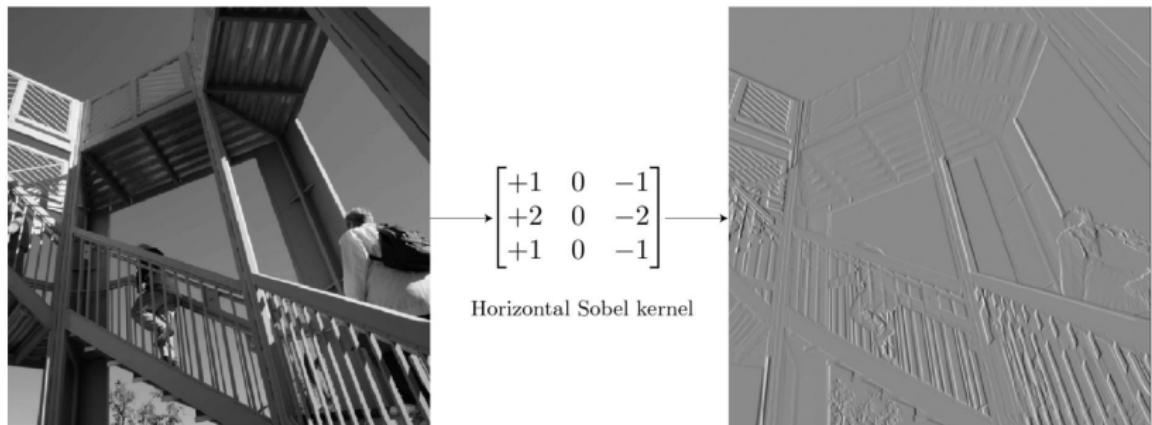
filter/kernel: curve detector

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

$$= 0$$

We get a large resulting value if the filter resembles the pattern in the image patch on which the filter was applied.

## Beispiel: Sobelfilter als Merkmalsdetektor für vertikale Kanten



Die Aktivierungskarte gibt an, wie ähnlich der lokale Bildausschnitt zu der Filtermaske ist, genauer: wie stark Bildausschnitt und Filtermaske miteinander **korreliert** sind.

## Kreuzkorrelation und Faltung

Trotz ihres irreführenden Namens führen Faltungsneuronen eigentlich keine Faltung des Bildes  $I$  durch, sondern eine **Kreuzkorrelation** mit einer  $n \times m$ -Filtermaske  $w$ :

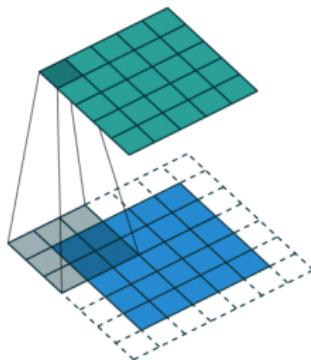
$$a(x, y) = w \otimes I = \sum_{x'=0}^{n-1} \sum_{y'=0}^{m-1} w(x', y') I(x + x', y + y'),$$

d.h. die Maske wird wie ein Fenster über das Eingangsbild  $I$  gelegt und punktweise multipliziert. Vgl. hierzu die **Faltung**:

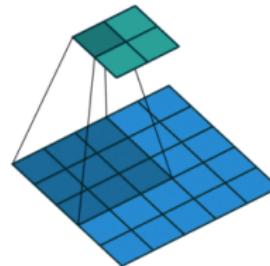
$$\begin{aligned} a(x, y) &= w * I = \sum_{x'=0}^{n-1} \sum_{y'=0}^{m-1} w(x', y') I(x - x', y - y') \\ &= \sum_{x'=0}^{n-1} \sum_{y'=0}^{m-1} w(-x', -y') I(x + x', y + y'), \end{aligned}$$

d.h. die Maske wird bei der Faltung zuerst gespiegelt und erst dann korreliert!

# Randbehandlung und Strided Convolution



Zero-padding to achieve  
same size of feature and input



no padding to only use  
valid input information

Bei "valid"-Padding wird das Ausgangsbild kleiner als das Eingangsbild, bei Zero-Padding wird eine virtuelle Randregion mit Nullen gefüllt, so dass die Bildgröße erhalten bleibt. Allerdings wird dadurch die Randregion abgedunkelt (v.a. bei mehreren Faltungsschichten hintereinander).

Das Ausgangsbild kann auch durch Auslassen von Zeilen und Spalten bei der Faltung verkleinert werden (**Strided Convolution**).

# Beispiel Faltungsschicht mit Strided Convolution und Zero-Padding

0	0	0	0	0	0	0	0
0	2	3	2	1	0	0	0
0	0	1	-0.2	1	3	1	0
0	3	1	2	2	3	0	0
0	2	0	0	2	2	0	0
0	2	0	0	0	1	0	0
0	0	0	0	0	0	0	0



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



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



0	0	0	0	0	0	0	0
0	3	3	2	1	0	0	0
0	0	1	0.2	1	3	1	0
0	2	3	2	1	0	0	0
0	0	2	1	2	2	3	0
0	2	1	0.2	0	2	2	0
0	2	0	0	0	1	0	0
0	0	0	0	0	0	0	0



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



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



0	0	0	0	0	0	0	0
0	3	3	2	1	0	0	0
0	0	0	1	3	1	0	0
0	3	1	2	2	3	0	0
0	2	1	0.2	0	2	2	0
0	2	0	0	0	1	0	0
0	0	1	0.2	0	0	0	0



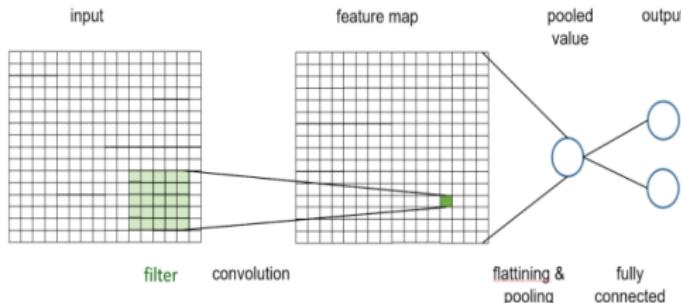
0	0	0	0	0	0	0	0
0	3	3	2	1	0	0	0
0	0	0	1	3	1	0	0
0	3	1	2	2	3	0	0
0	2	0	0	1	2	2	0
0	2	0	0	0	1	0	0
0	0	0	0	0	0	0	0



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



# Beispiel Edge Lover: Ein einschichtiges Faltungsnetz



```
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'))
```

[Demo]