

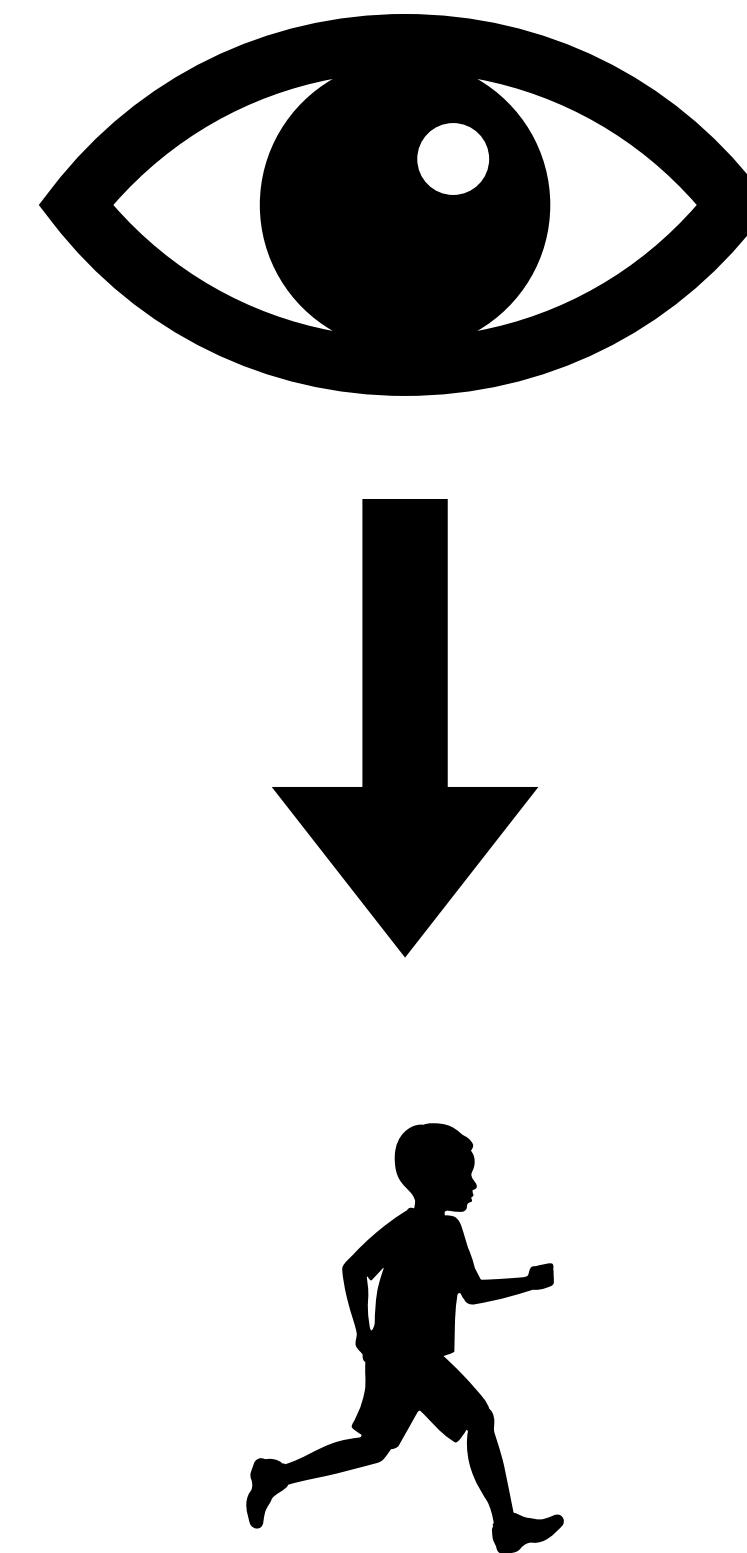
Grundlagen des maschinellen Lernens DeepLearning (CNNs)

Ole Meyer

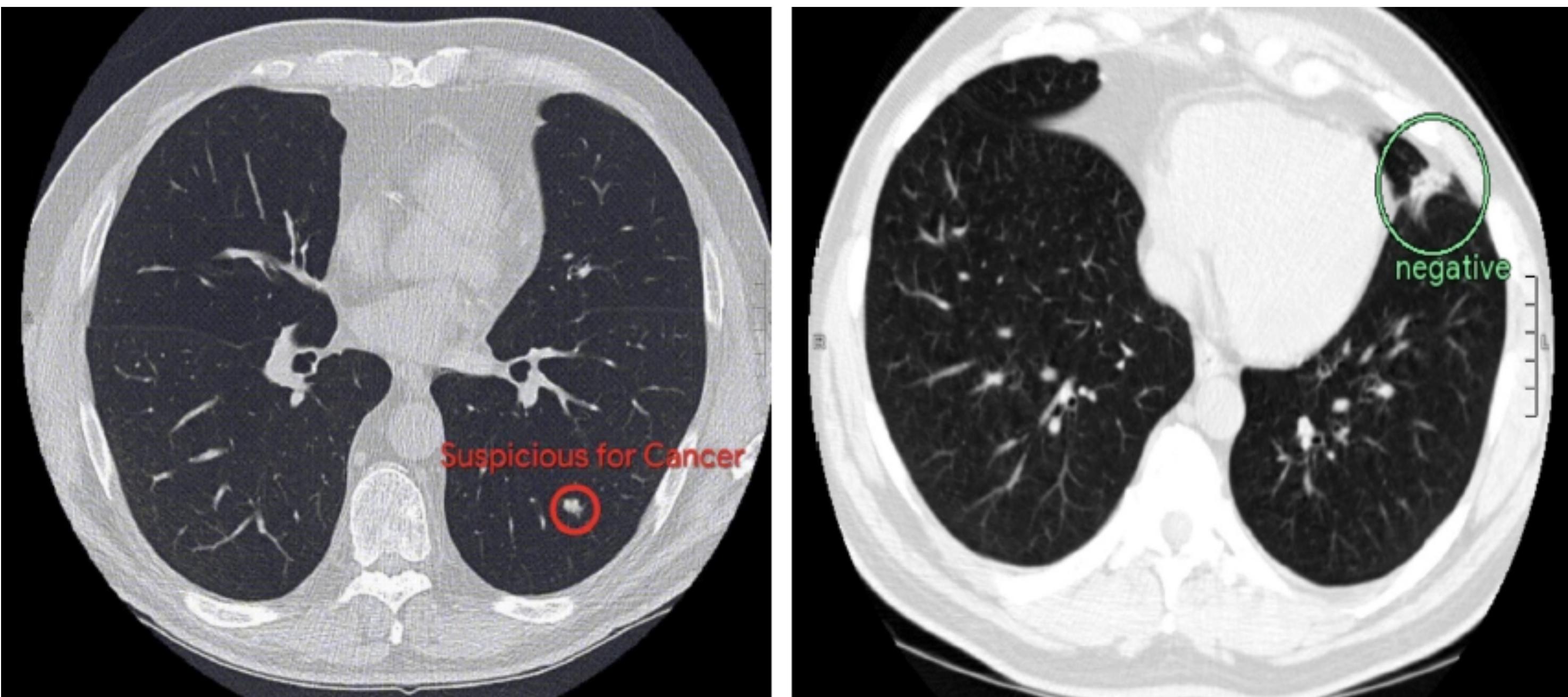
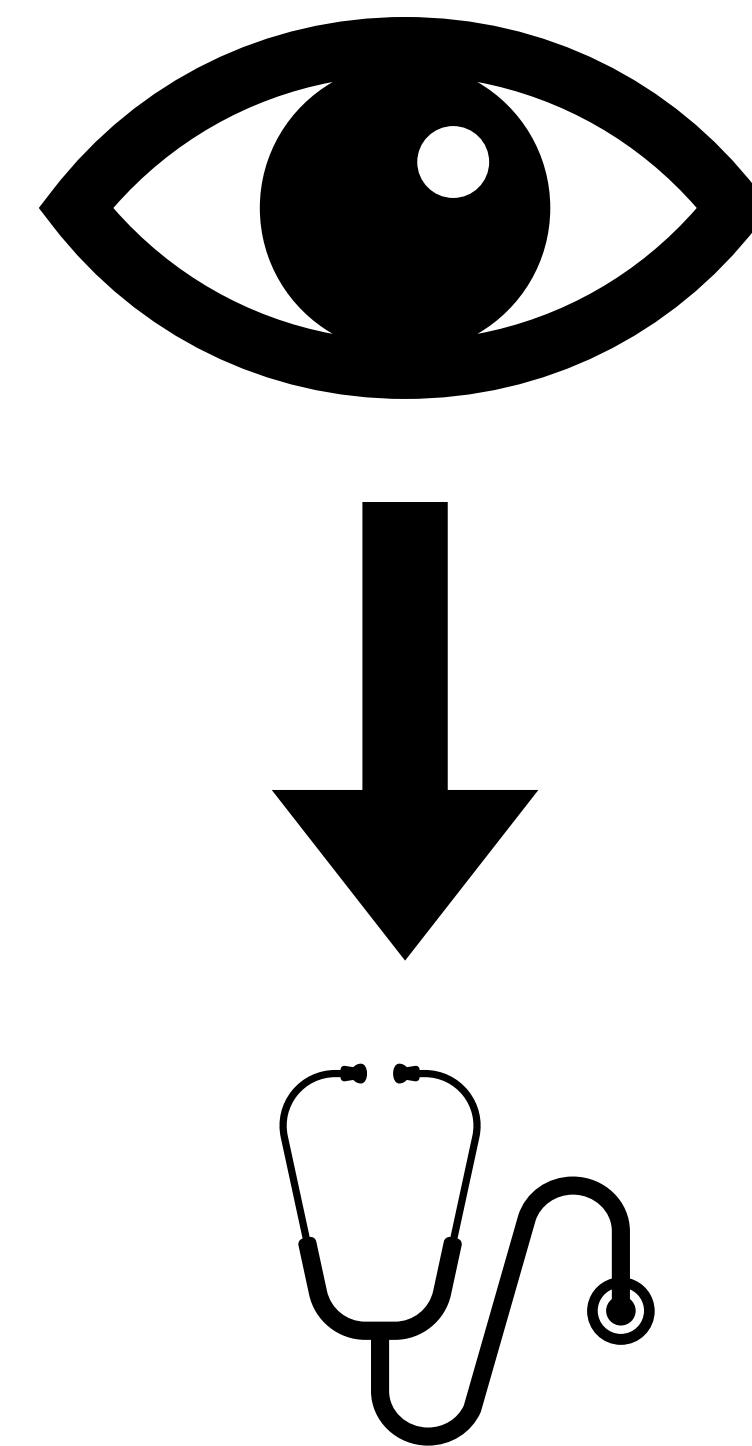


Bildverarbeitung - Bedeutung

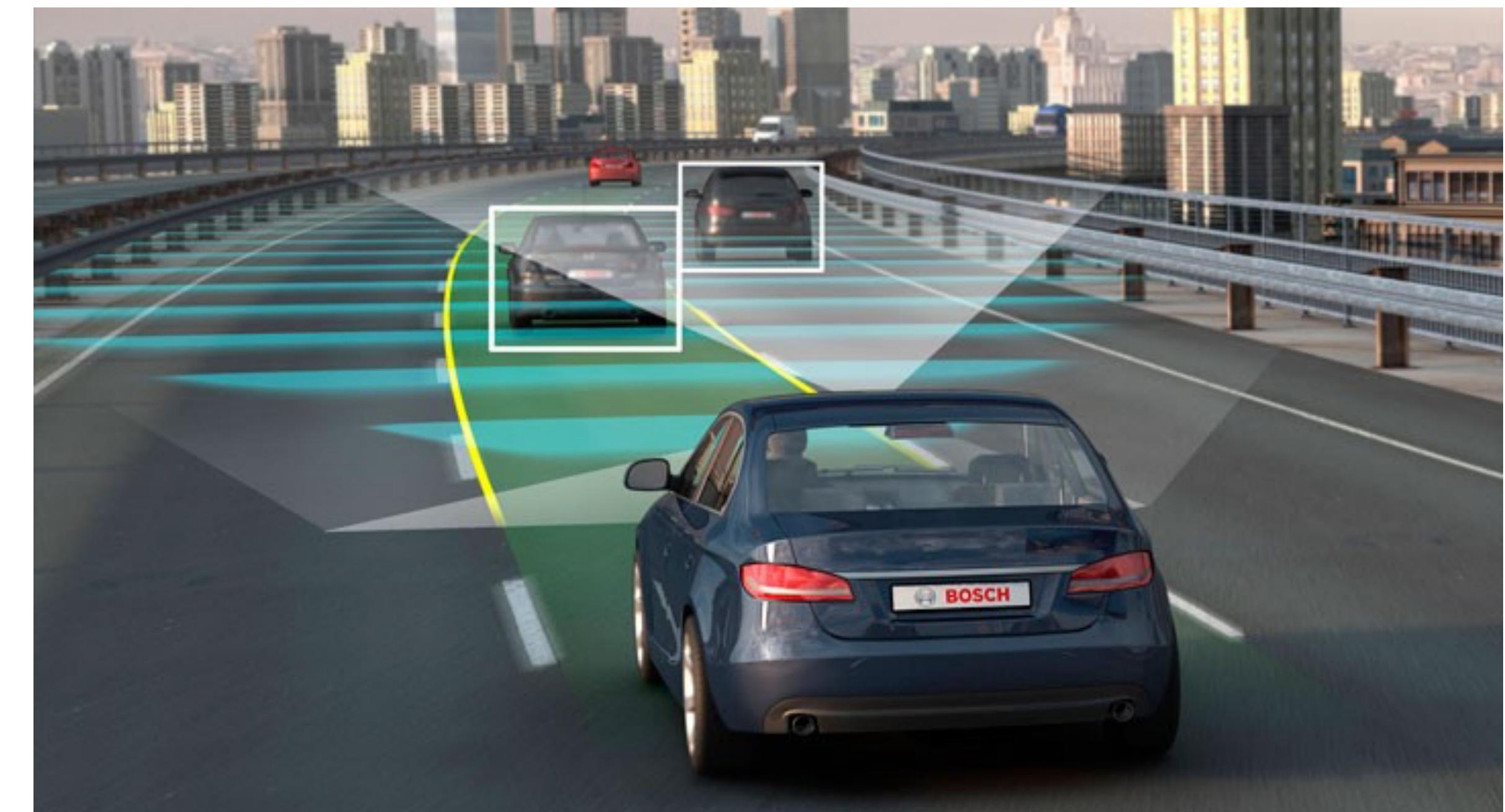
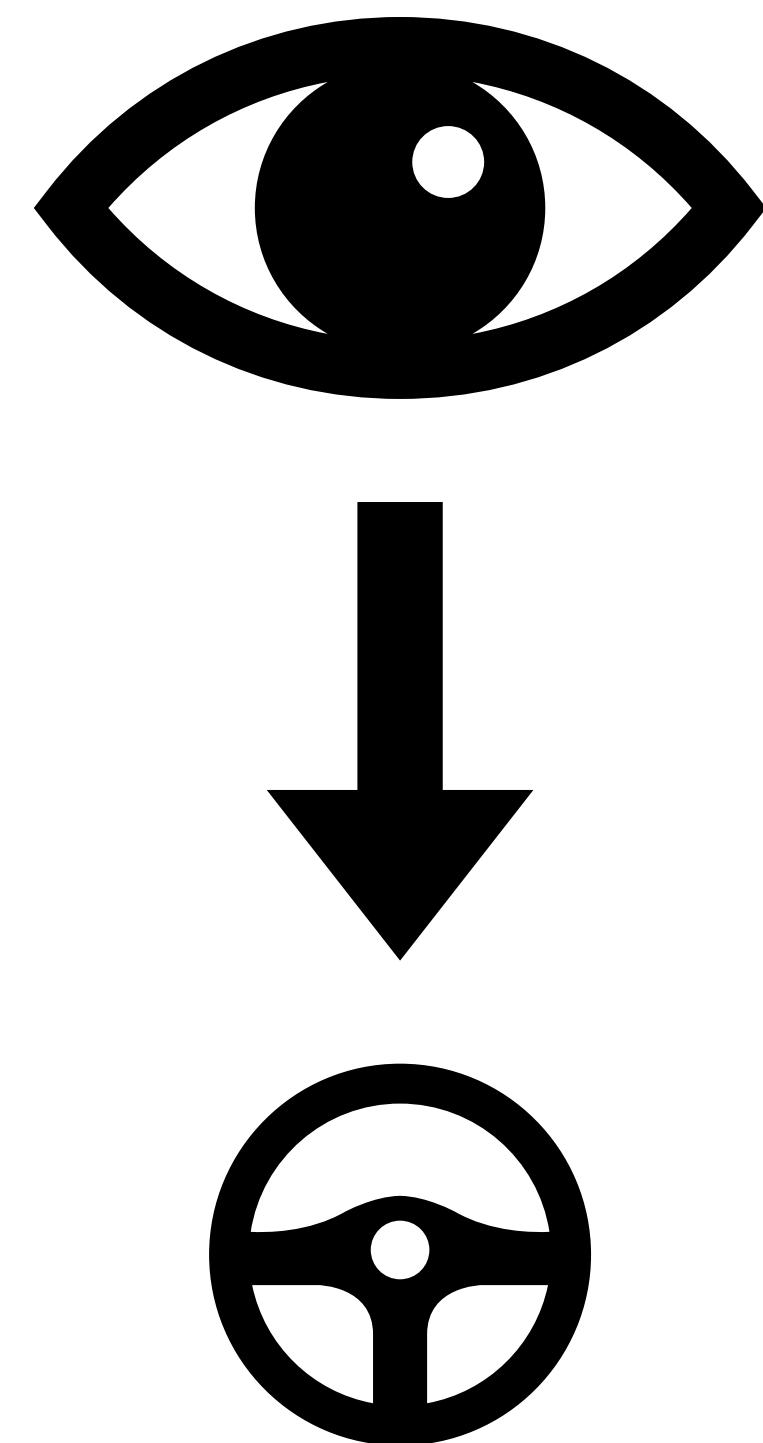
Personenerkennung & Extraktion von Bewegungen



Erkennung von Lungenkrebs

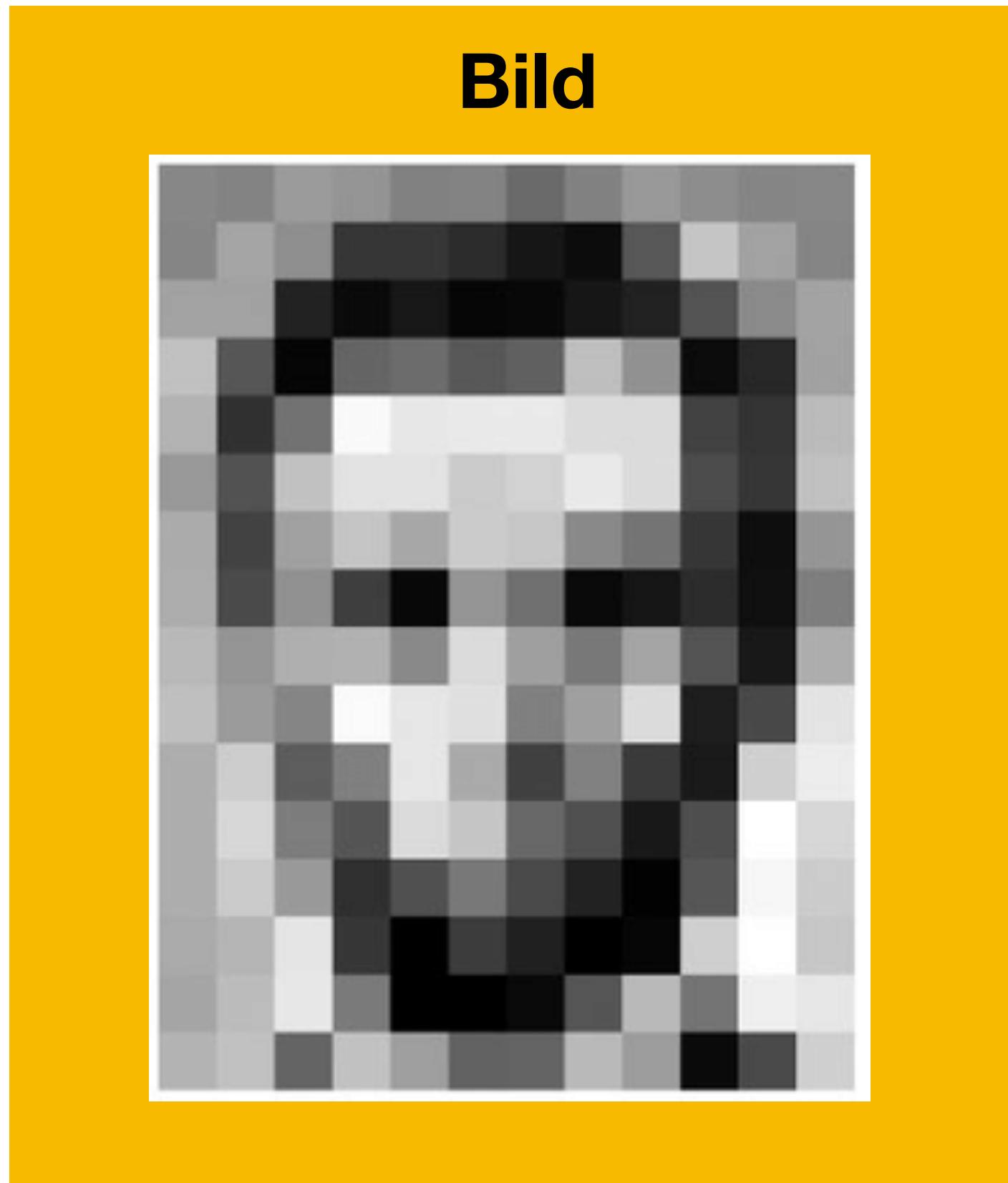


Autonomes Fahren



Was sieht ein Computer?

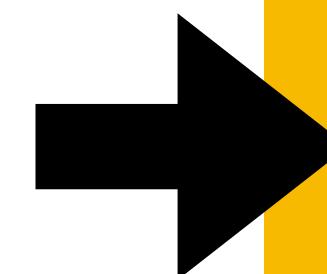
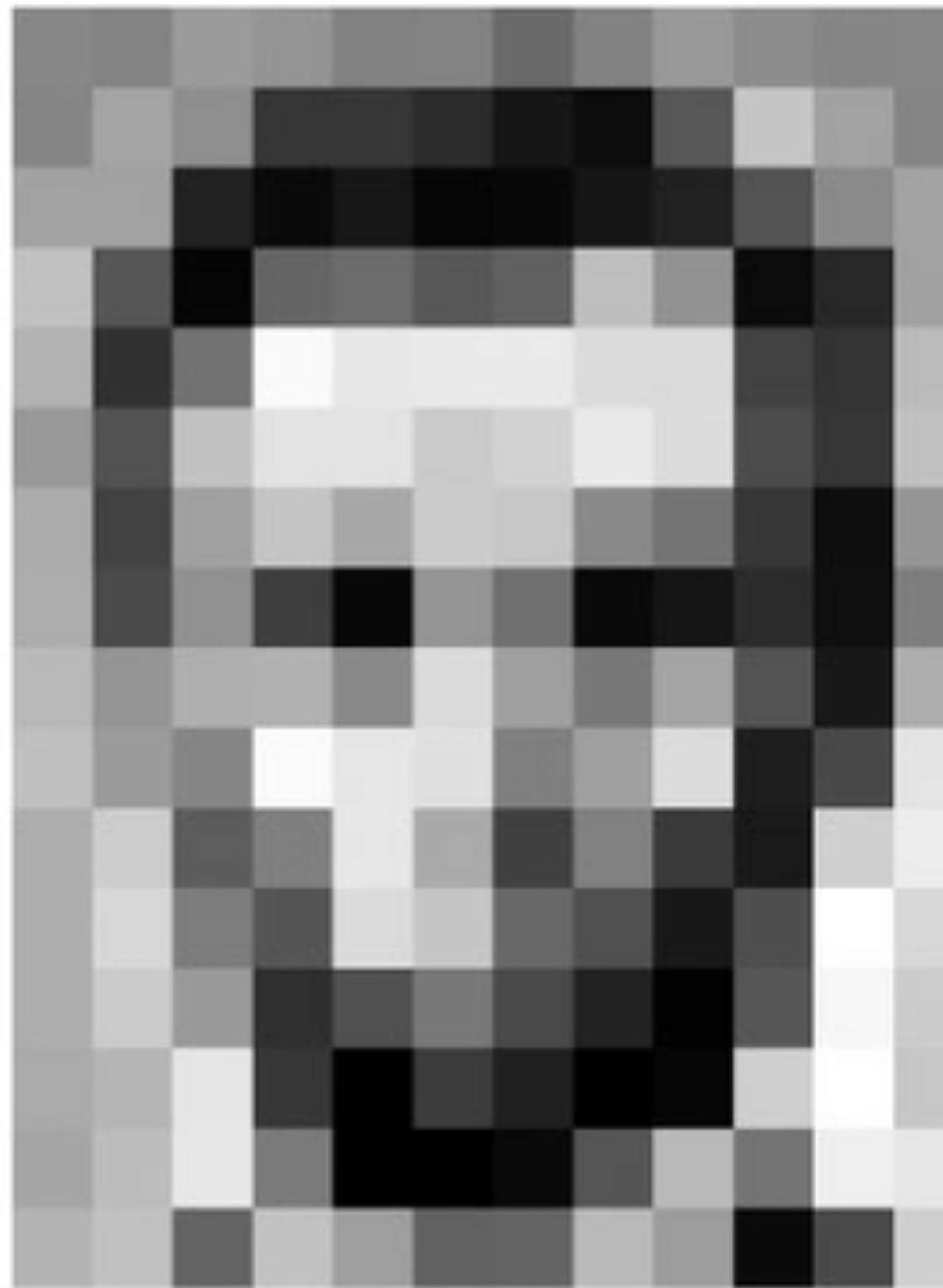
Datenformat von Bildern



Menschliche Wahrnehmung

Datenformat von Bildern

Bild

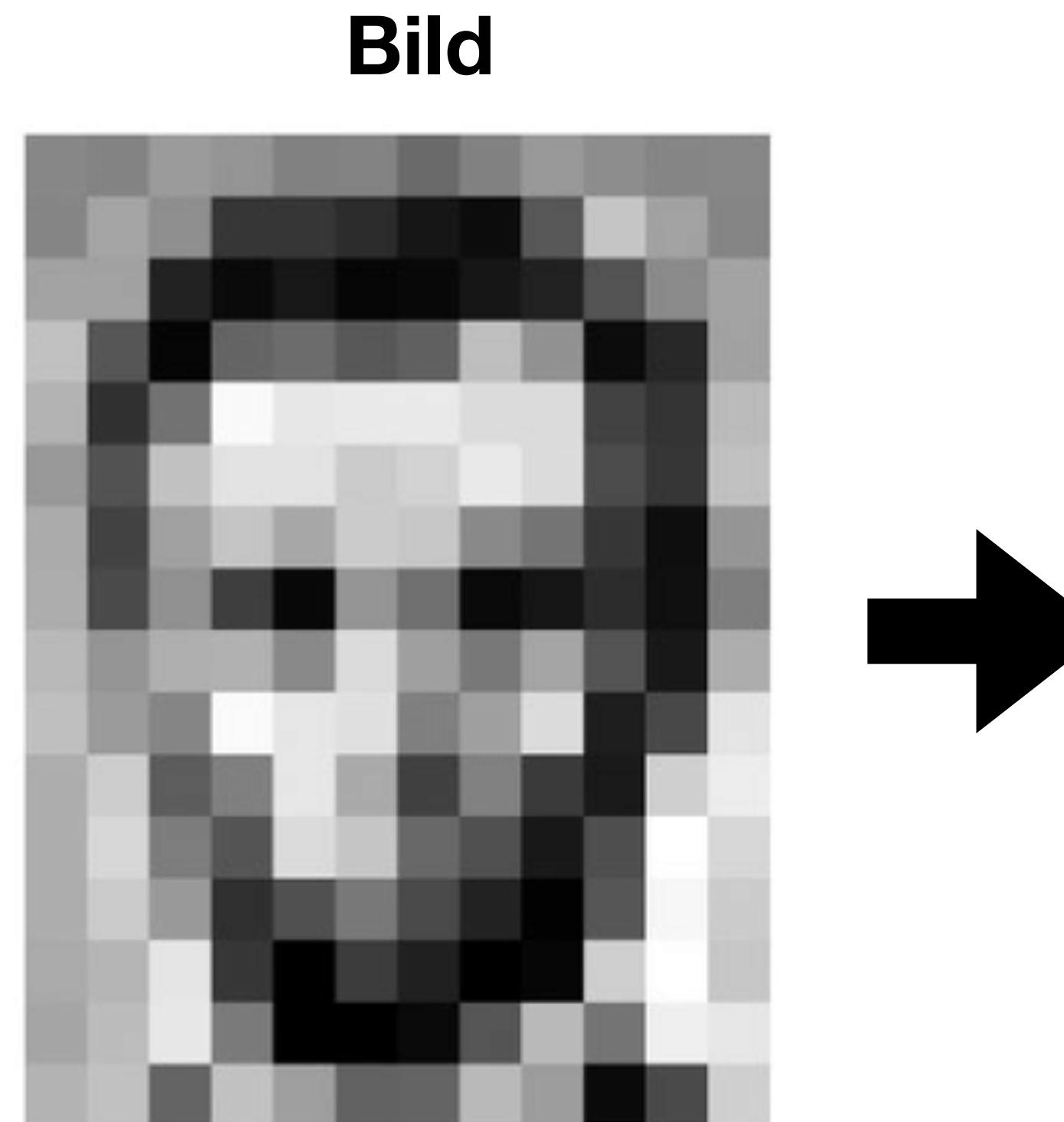


Graustufenwerte

157	153	174	168	150	162	129	151	172	161	155	166
155	182	169	74	75	62	93	17	110	210	180	154
180	180	50	14	54	6	10	93	49	105	159	181
206	109	5	124	191	111	120	204	166	15	56	180
194	68	197	251	237	299	239	228	227	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	86	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	105	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	85	150	79	38	218	241
190	224	147	108	227	210	127	102	35	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Repräsentation von Farbwerten

Datenformat von Bildern



Graustufenwerte

157	153	174	168	150	152	129	151	172	161	155	166
155	182	163	74	75	62	93	17	110	210	180	154
180	180	50	14	34	6	10	33	48	105	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	299	239	228	227	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	105	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	85	150	79	38	218	241
190	224	147	108	227	210	127	102	35	101	255	224
190	214	173	66	103	143	98	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

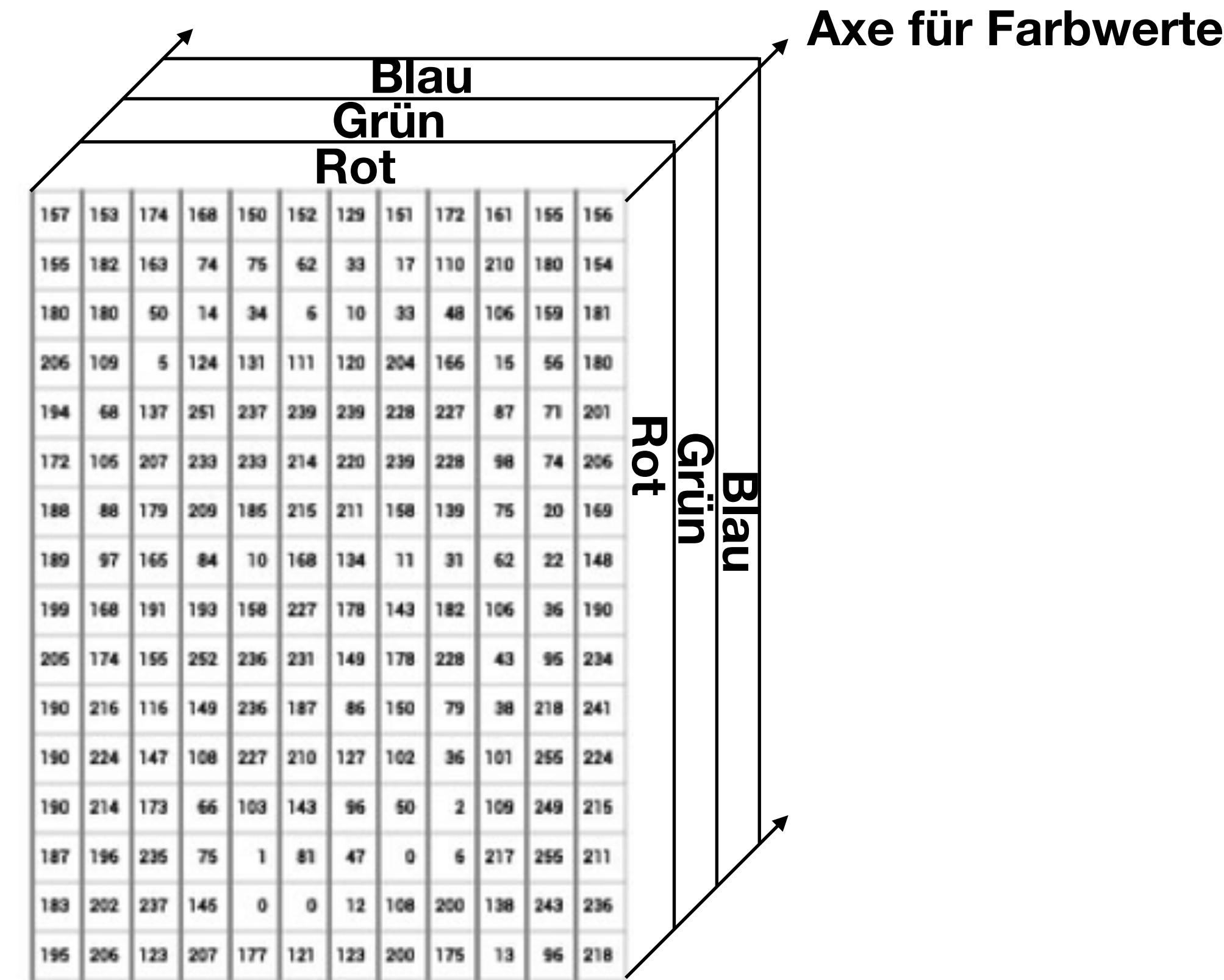
Zahlendarstellung

157	153	174	168	150	152	129	151	172	161	155	166
155	182	163	74	75	62	93	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	299	239	228	227	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	105	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	85	150	79	38	218	241
190	224	147	108	227	210	127	102	35	101	255	224
190	214	173	66	103	143	98	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Das sieht ein Computer!

Schwarz-Weiß-Bilder werden durch eine Matrix der Farbwerte jedes Pixels dargestellt.
Farbwerte bewegen sich immer zwischen 0 und 255 (Integer!).

Datenformat von Bildern



Bei Farbbildern können die einzelnen Farbwerte (z.B Rot/Gelb/Blau bei RGB-Bildern) in einer zusätzlichen Dimension gespeichert werden.

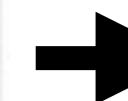
Beispiel für eine Klassifikation



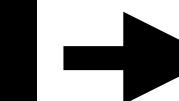
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	195	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	162	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Bild

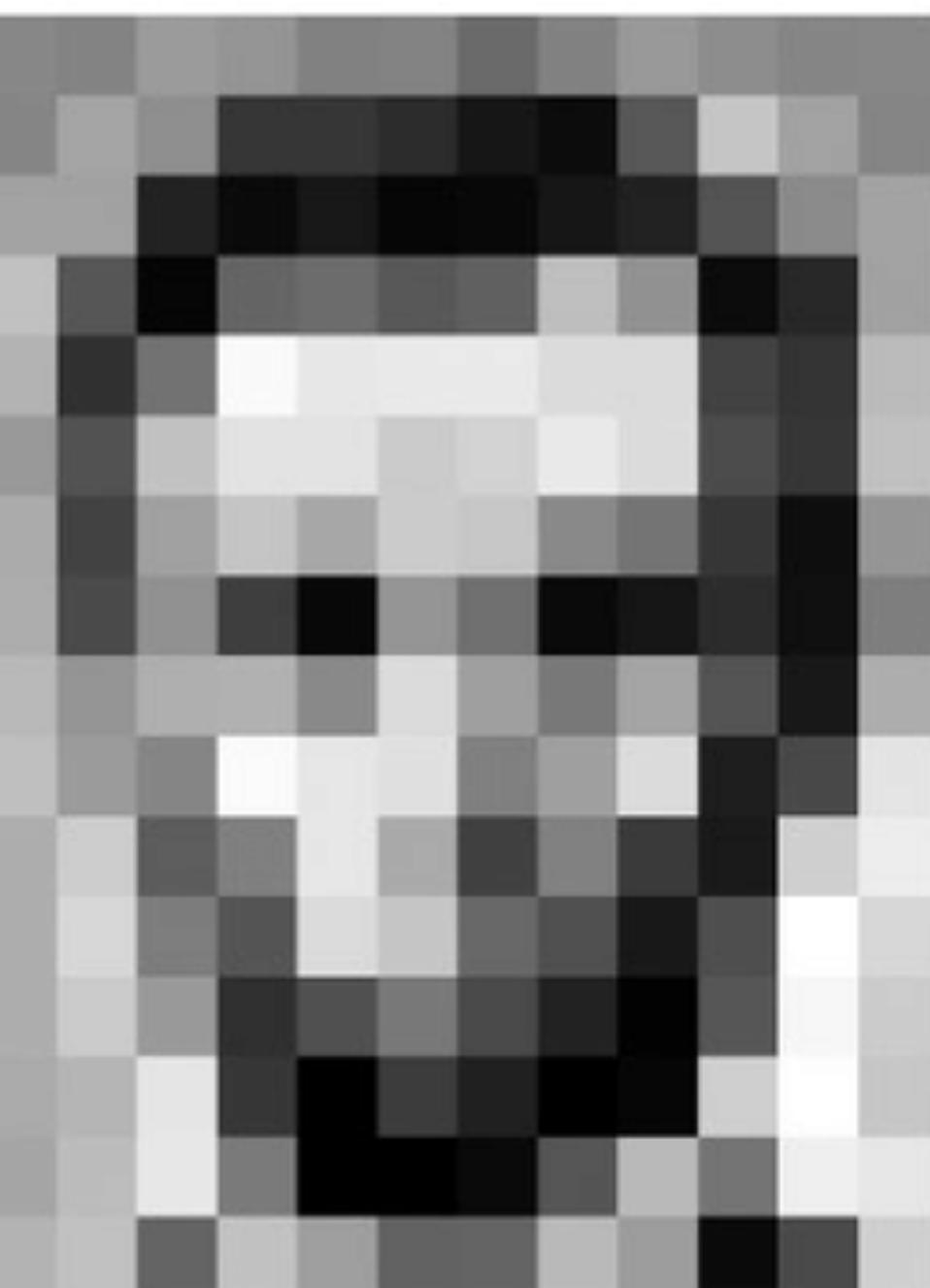
Pixelrepräsentation



Klassifikation



$$\begin{bmatrix} 0.95 \\ 0.02 \\ 0.03 \end{bmatrix} \begin{matrix} \text{Lincoln} \\ \text{Trump} \\ \text{Obama} \end{matrix}$$

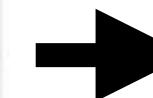


Bild



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	195	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	162	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Pixelrepräsentation



Regression

→ 54 Jahre

Highlevel-Features helfen uns (Menschen) Objekte zu erkennen und zu beschreiben



Das ist ein Auto wegen: Nummernschild, Reifen, Scheinwerfern,...

Highlevel-Features helfen uns (Menschen) Objekte zu erkennen und zu beschreiben



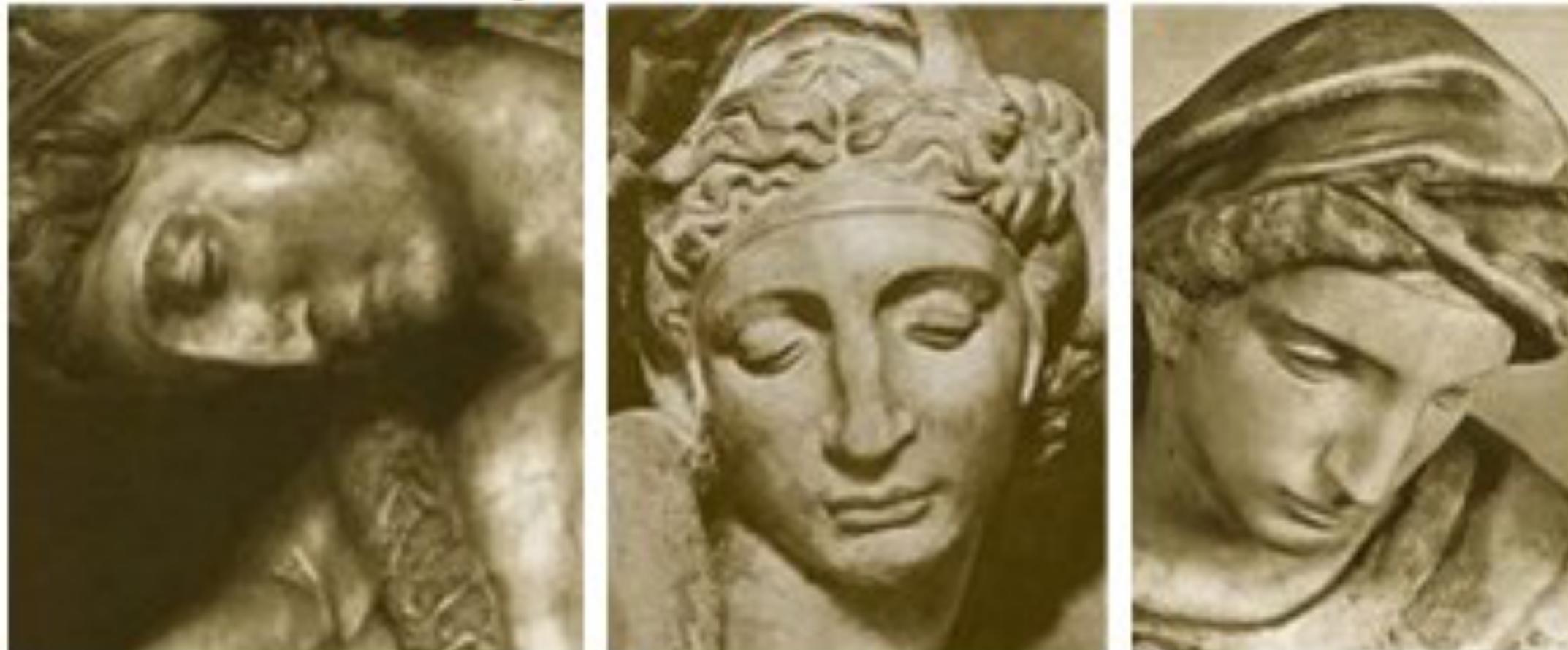
Das ist ein Haus wegen: Tür, Fenster, Dach,...

Highlevel-Features helfen uns (Menschen) Objekte zu erkennen und zu beschreiben



Das ist eine Person wegen: Augen, Nase, Mund, Haare, ...

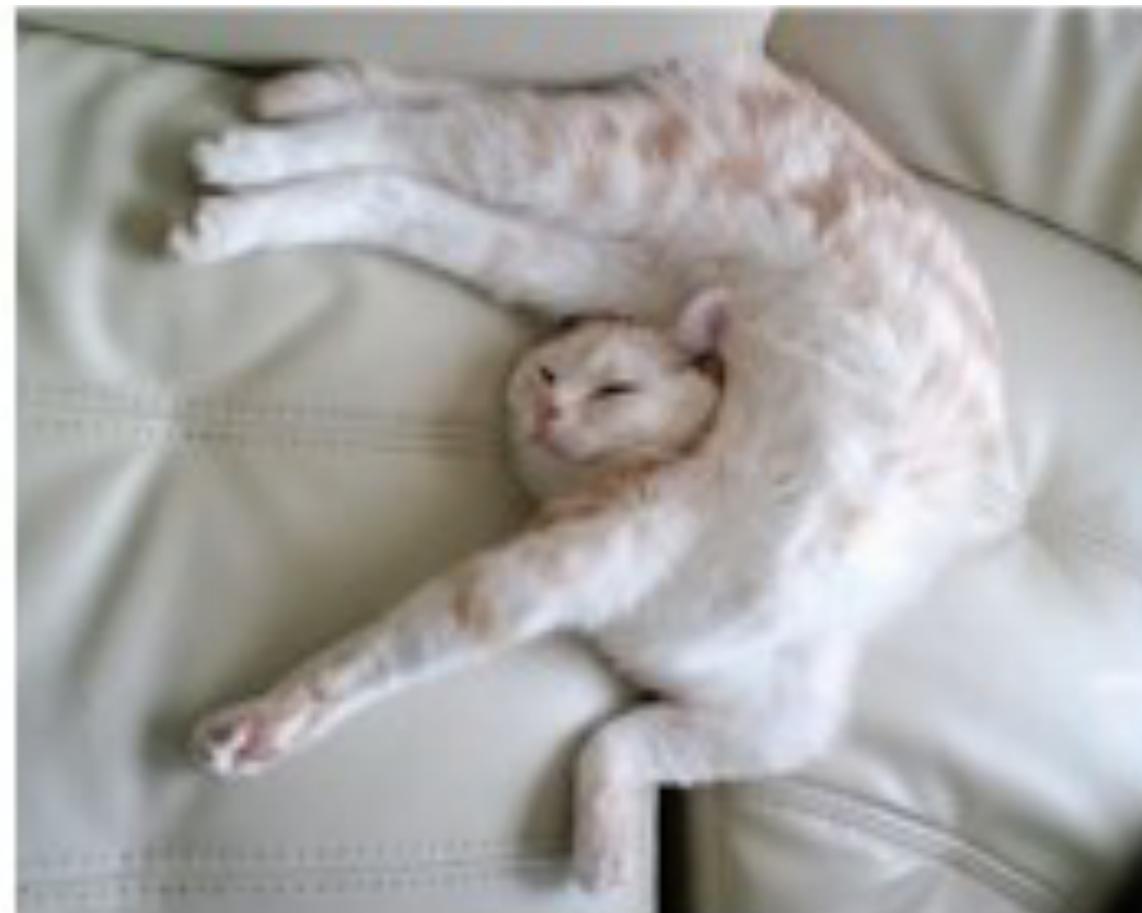
Variationen des Point of View



Variationen in der Beleuchtung



Deformation & Occlusion



Hintergrund & Intra-Class Variation

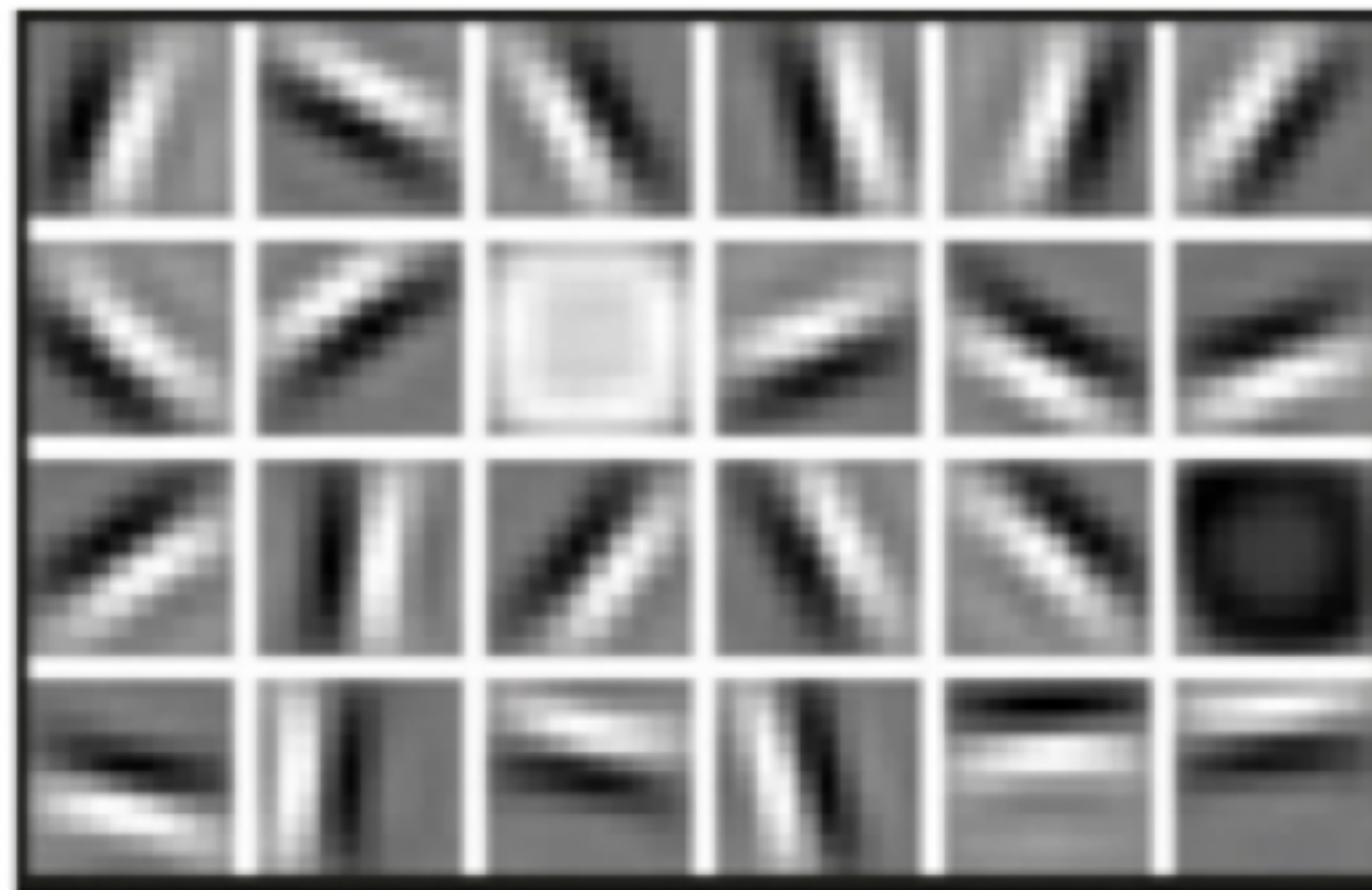


Variationen im Größenverhältnis



Gibt es die Möglichkeit Features automatisch durch ein Neuronales Netzwerk lernen zu lassen?

Low-Level



Kanten & Linien

Mid-Level



Ohren & Augen & ...

High-Level

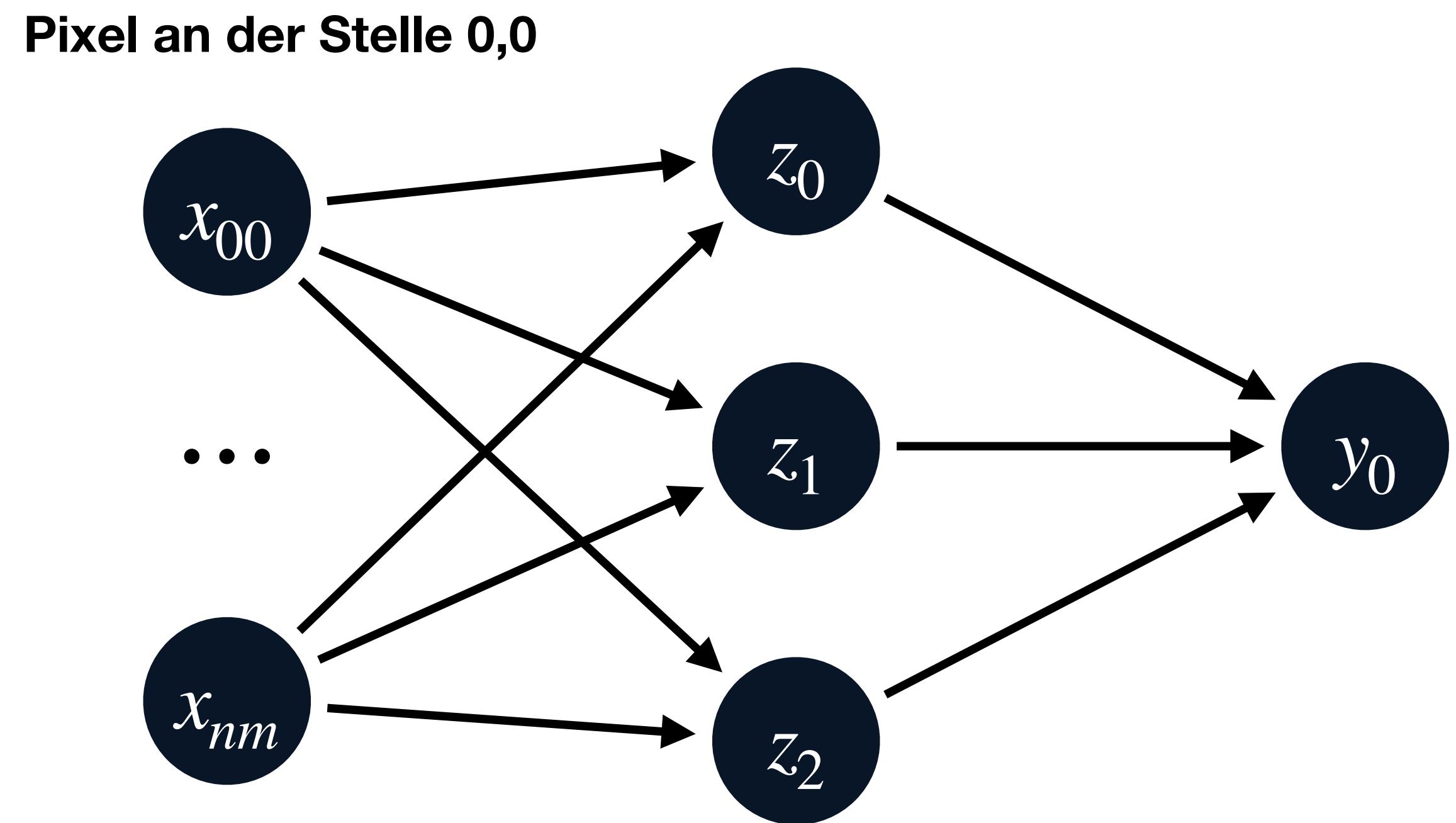


Gesichter

Convolutional Neural Networks (CNN)

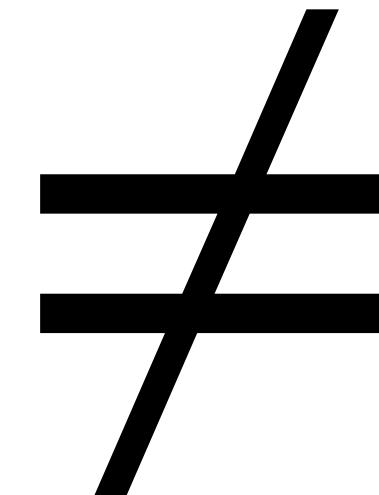
Bausteine

- Vollvermaschtes Netz
 - DenseLayer/LinearLayer
- Jeder Pixelwert ist eine Eingabedimension
- Räumliche Beziehungen werden nicht abgebildet
- Extrem viele Parameter
 - Bei einem 500x500 großen Bild gibt es alleine 250.000 Eingangsneuronen (Schwarz-Weiß)



Bilder sind selten gleich. Das Netzwerk muss daher robust sein gegenüber Verschiebungen, Verformungen und anderen Variationen.

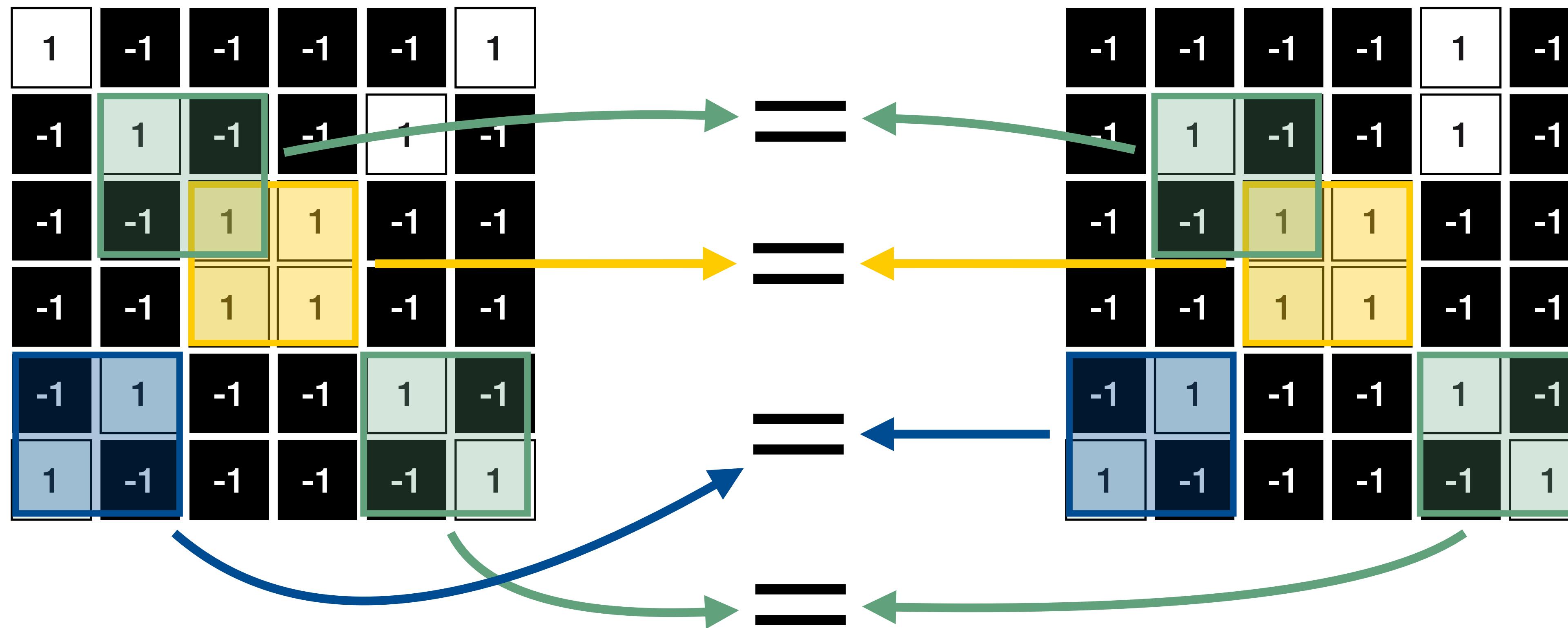
1	-1	-1	-1	-1	1
-1	1	-1	-1	1	-1
-1	-1	1	1	-1	-1
-1	-1	1	1	-1	-1
-1	1	-1	-1	1	-1
1	-1	-1	-1	-1	1



-1	-1	-1	-1	1	-1
-1	1	-1	-1	1	-1
-1	-1	1	1	-1	-1
-1	-1	1	1	-1	-1
-1	1	-1	-1	1	-1
1	-1	-1	-1	-1	1

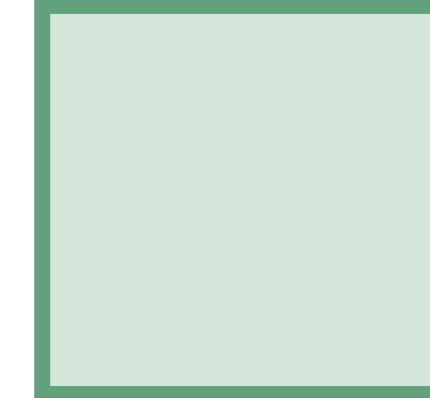
Grundideen: Filter

Auch wenn das Bild sich grundsätzlich unterscheidet, können Teile (Patches) die gleichen Muster aufweisen.



Filter = Elementweise multiplizierte Matrix / Tensor

1	-1	-1	-1	-1	1
-1	1	-1	-1	1	-1
-1	-1	1	1	-1	-1
-1	-1	1	1	-1	-1
-1	1	-1	-1	1	-1
1	-1	-1	-1	-1	1



$$= \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

Elementweise Multiplikation

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \odot \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \Rightarrow 4$$

Summe des Ergebnisses

=

Wir stark spricht ein Filter an?

Filter = Elementweise multiplizierte Matrix / Tensor

1	-1	-1	-1	-1	1
-1	1	-1	-1	1	-1
-1	-1	1	1	-1	-1
-1	-1	1	1	-1	-1
-1	1	-1	-1	1	-1
1	-1	-1	-1	-1	1



$$= \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$$

Elementweise Multiplikation

$$\begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \odot \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \Rightarrow 4$$

Summe des Ergebnisses

=

Wir stark spricht ein Filter an?

Filter = Elementweise multiplizierte Matrix / Tensor

1	-1	-1	-1	-1	1
-1	1	-1	-1	1	-1
-1	-1	1	1	-1	-1
-1	-1	1	1	-1	-1
-1	1	-1	-1	1	-1
1	-1	-1	-1	-1	1



$$= \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Elementweise Multiplikation

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \odot \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \Rightarrow 4$$

Summe des Ergebnisses

=

Wir stark spricht ein Filter an?

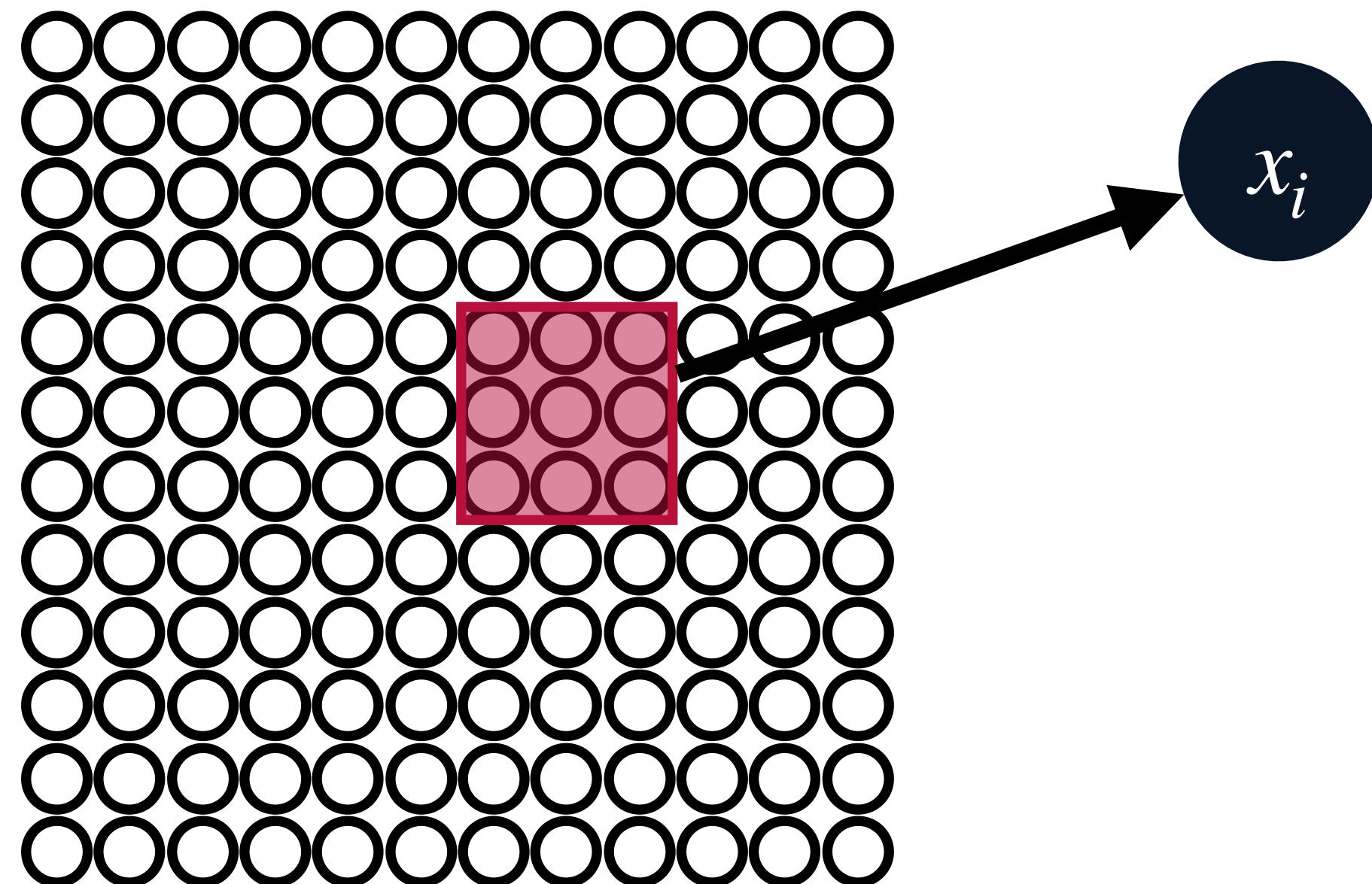
- Filter können eine beliebige Größe besitzen von 1×1 bis $m \times n$
- Größere Filter extrahieren komplexerer Features aus dem Bild
- Kleinere Filter lernen einfachere, allgemeinere Features

1	-1	-1	-1	-1	-1	1
-1	1	-1	-1	1	-1	
-1	-1	1	1	-1	-1	
-1	-1	1	1	-1	-1	
-1	1	-1	-1	1	-1	
1	-1	-1	-1	-1	1	

Einzelneuronen werden nur zu kleineren Ausschnitten des gesamten Bildes verbunden und lernen einen **Filter**.

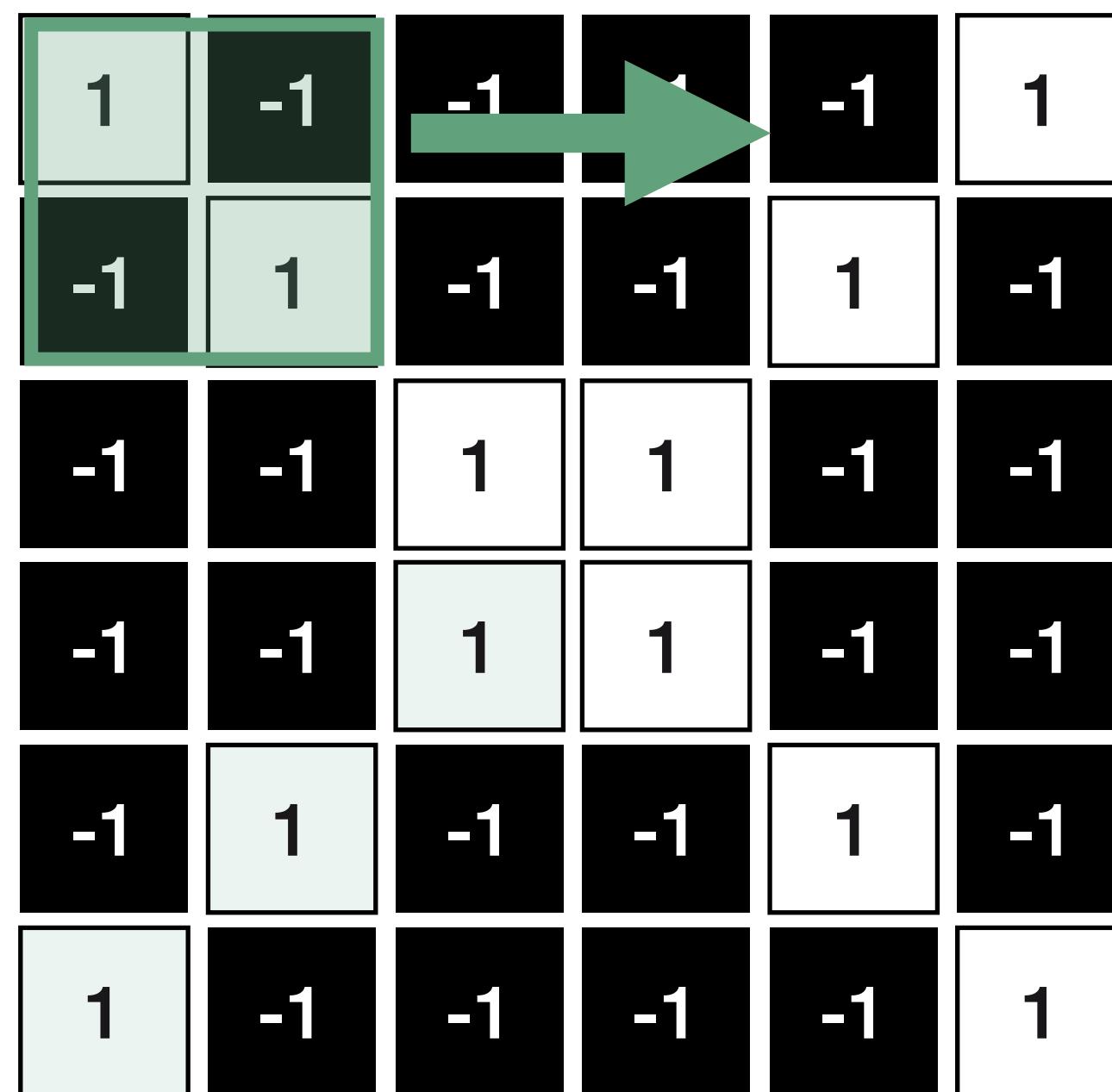
Ein Filter erkennt eine lokale Eigenschaft, z.B. eine bestimmte Kante.

Anwendung von Filter 1 auf einem Patch



Die Convolution kann berechnet werden, indem der Filter Stück für Stück über jeden Patch der Eingabe geschoben wird.

Das Ergebnis ist eine neue Repräsentation, welche die Relevanz des jeweiligen Filters im Ursprungsbild beschreibt.



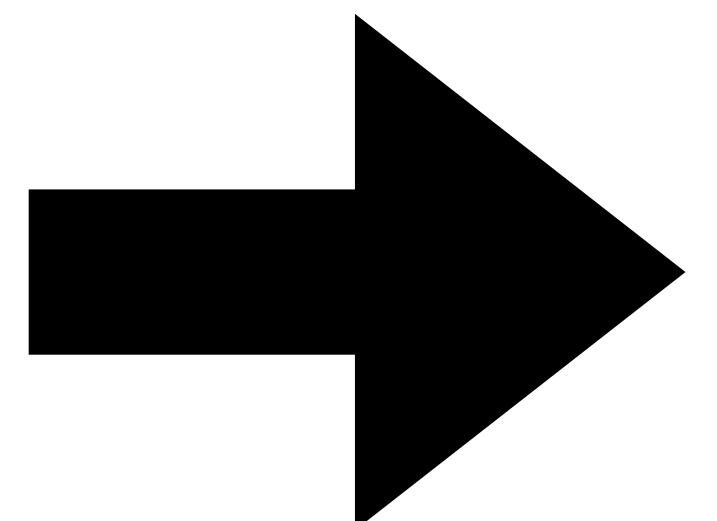
$$\otimes \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 4 & -2 & 0 & 2 & -4 \\ -2 & 4 & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

Grundideen: Convolution

Convolution ist ein in der Bildverarbeitung oft genutztes Verfahren, z.B. um Bilder zu schärfen oder Kanten zu erkennen.



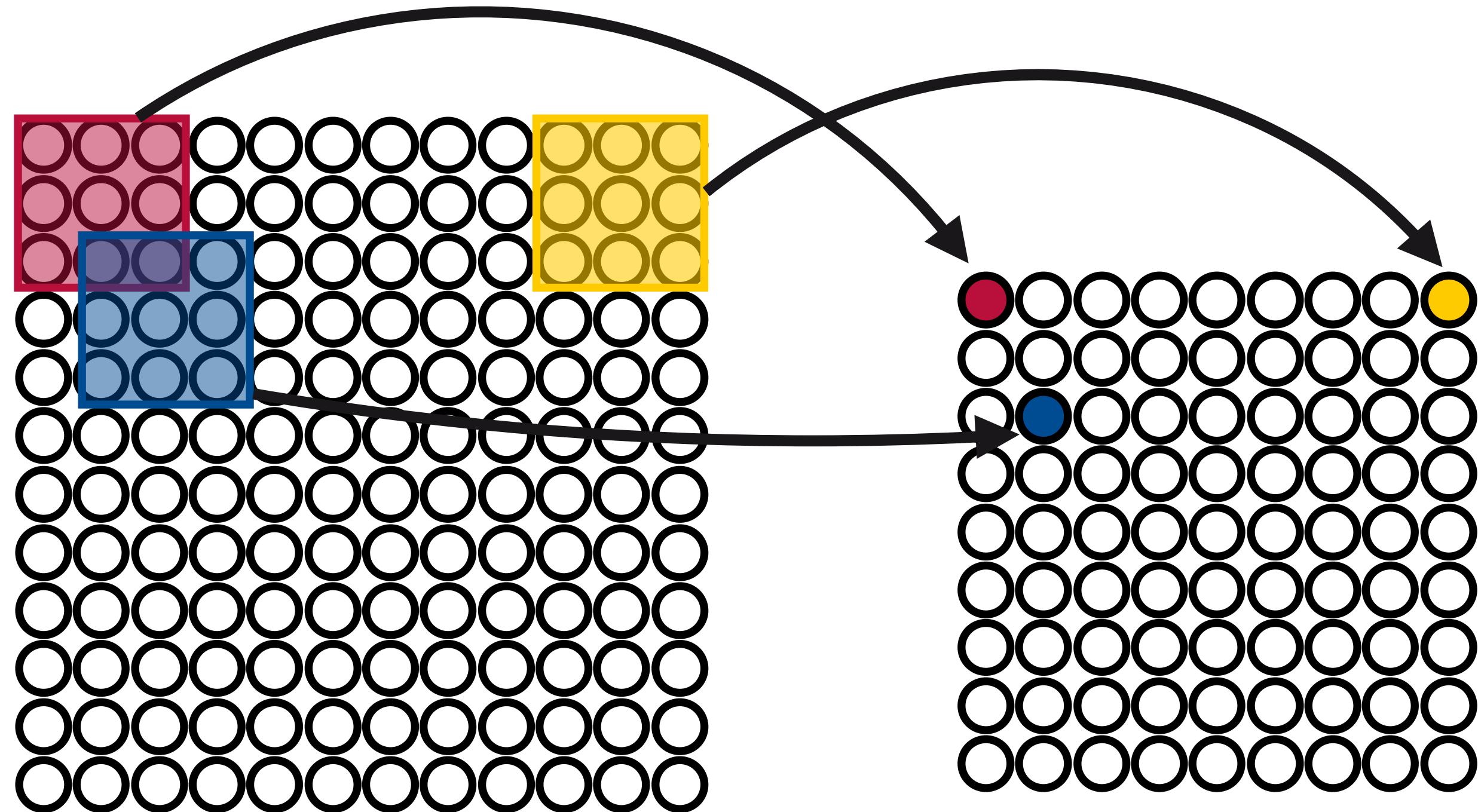
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 6 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



Anwendung von Filter 1 auf verschiedenen Patches

Räumliche Strukturen können erhalten bleiben, indem Neuronen entsprechend der verarbeitenden Bereiche geordnet bleiben.

Der Filter wird über ein Sliding-Window-Verfahren auf jeden verfügbaren Patch angewendet. Wenn Filter 1 in diesem Beispiel eine bestimmte Kante erkennt, dann ist das Ergebnis ein Verzeichnis darüber, wo diese spezielle Kante aufgetreten ist.

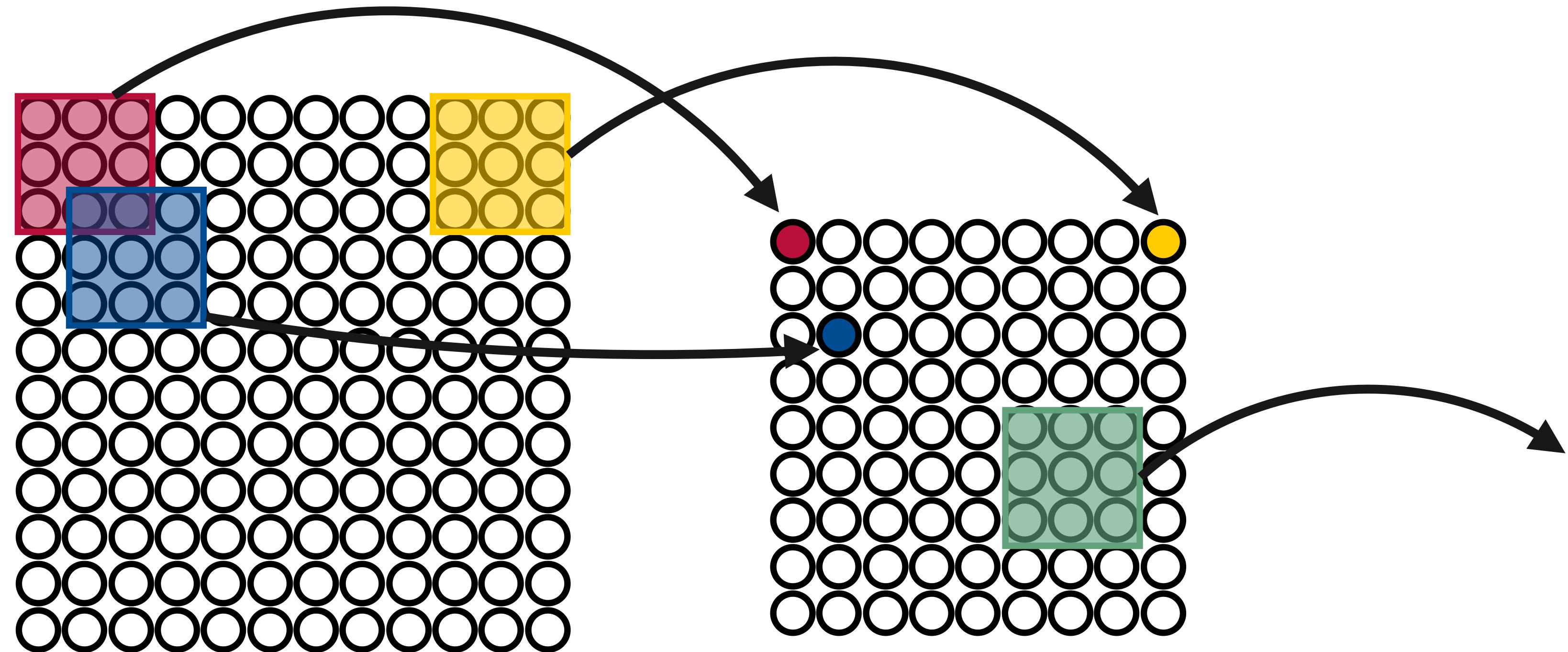


Mehrschichtige Convolution

Die durch die Neuronen abgebildeten Features/Filter, können nun auf die gleiche Art und Weise wieder verarbeitet werden.

So entstehen mehrere Schichten.

Der nächste Filter erkennt somit Kombinationen aus Formen des ersten Layers.

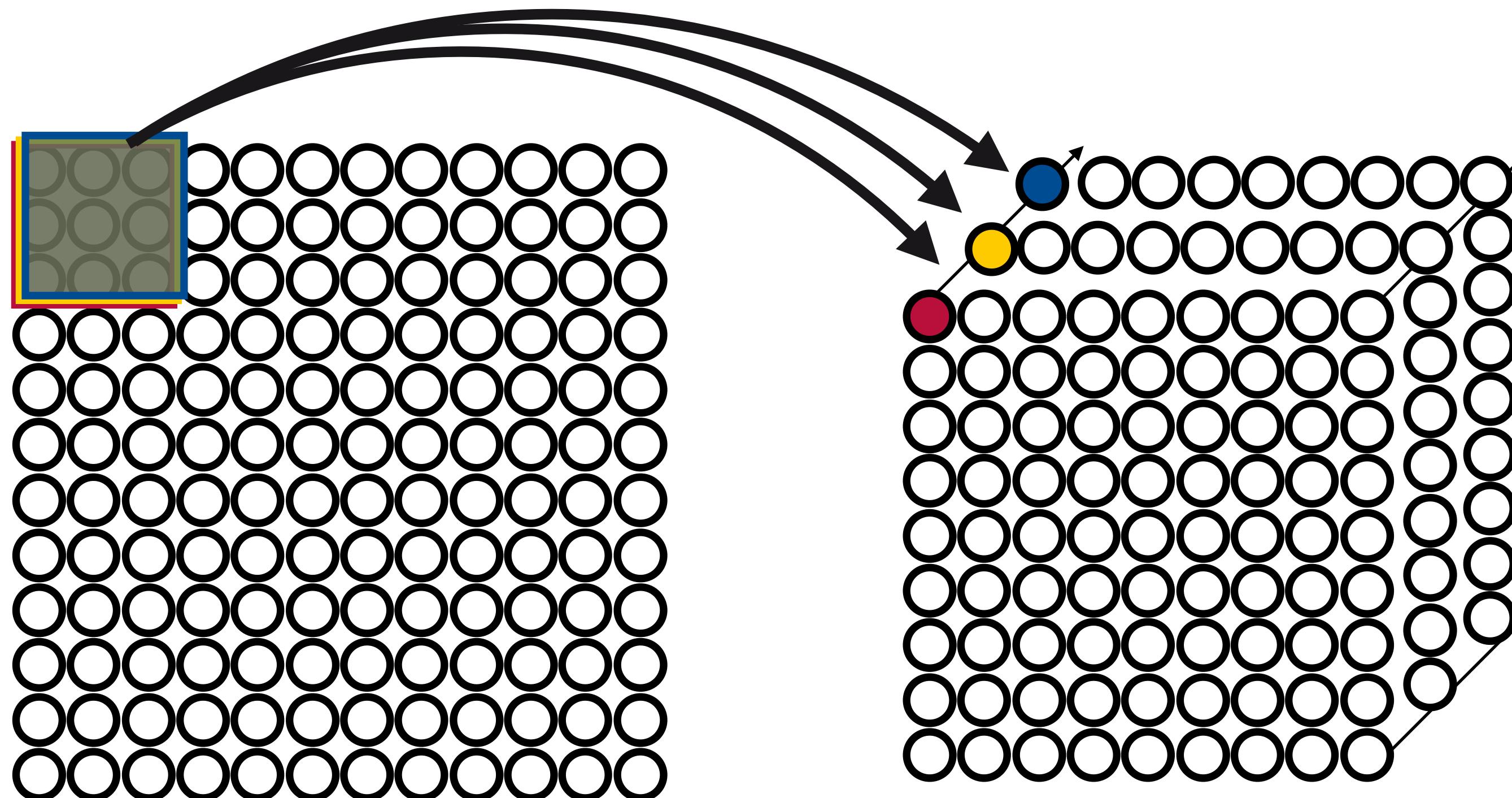


Anwendung von Filter 1, Filter 2 und Filter 3 auf einem Patch

I.d.R. reicht ein Filter pro Layer nicht aus.

Es können parallel mehrere Filter gelernt werden.

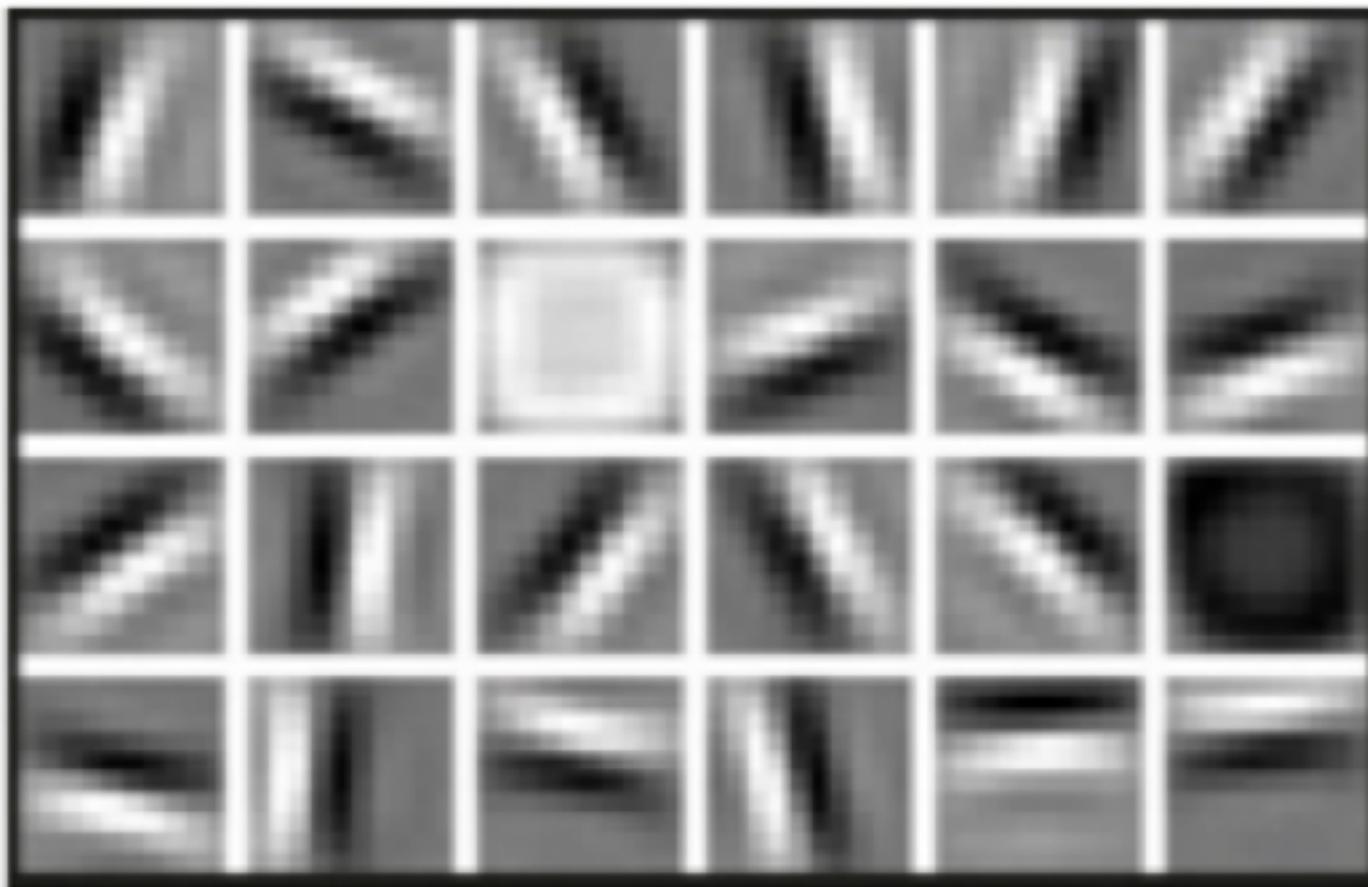
Jeder Filter erzeugt eine neue Dimension in der Eingabe des nächsten Layers.



Gibt es die Möglichkeit Features automatisch durch ein Neuronales Netzwerk lernen zu lassen?

Durch mehrschichtige Convolution können zunächst einfache Features und darauf aufbauend immer komplexere Konzepte erlernt werden.

Low-Level



Kanten & Linien

Mid-Level



Ohren & Augen

High-Level

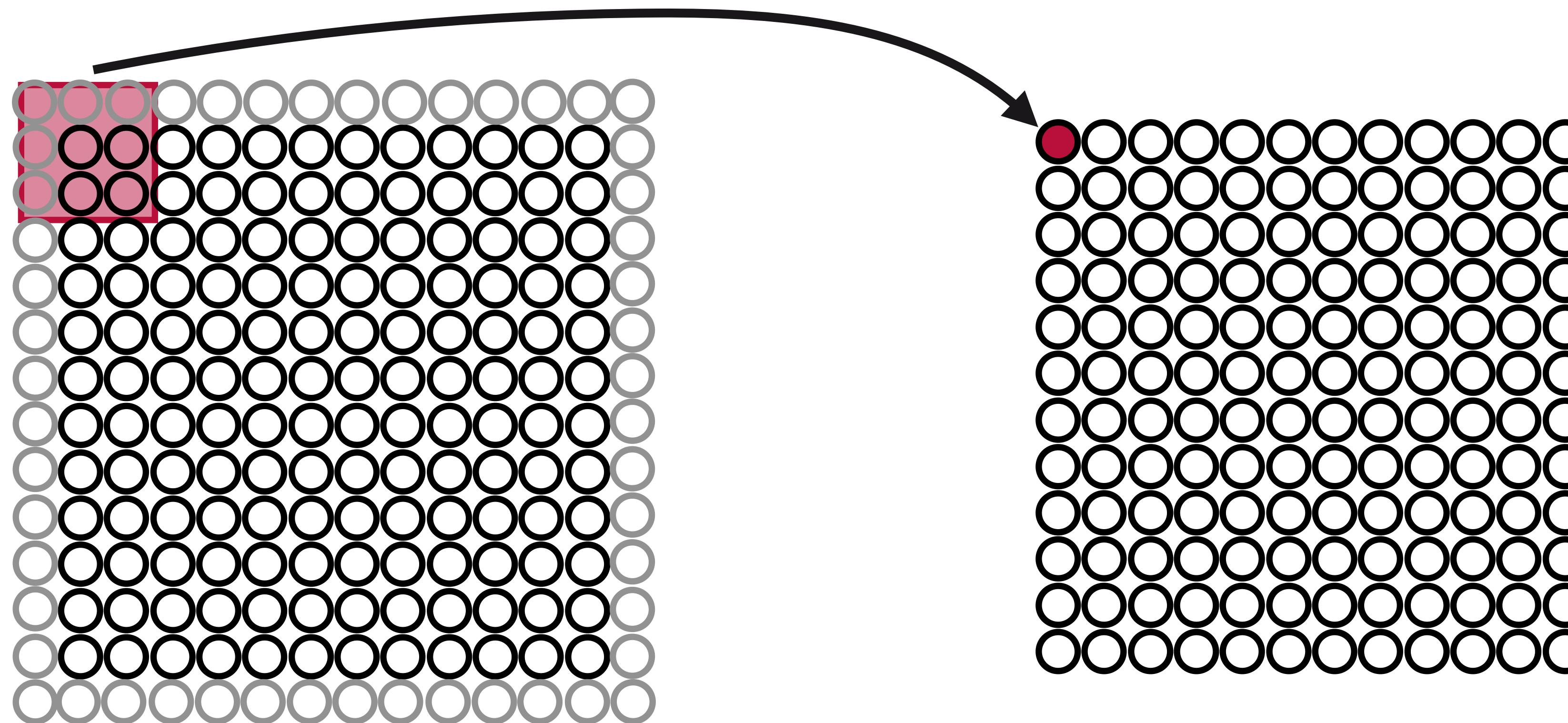


Gesichter

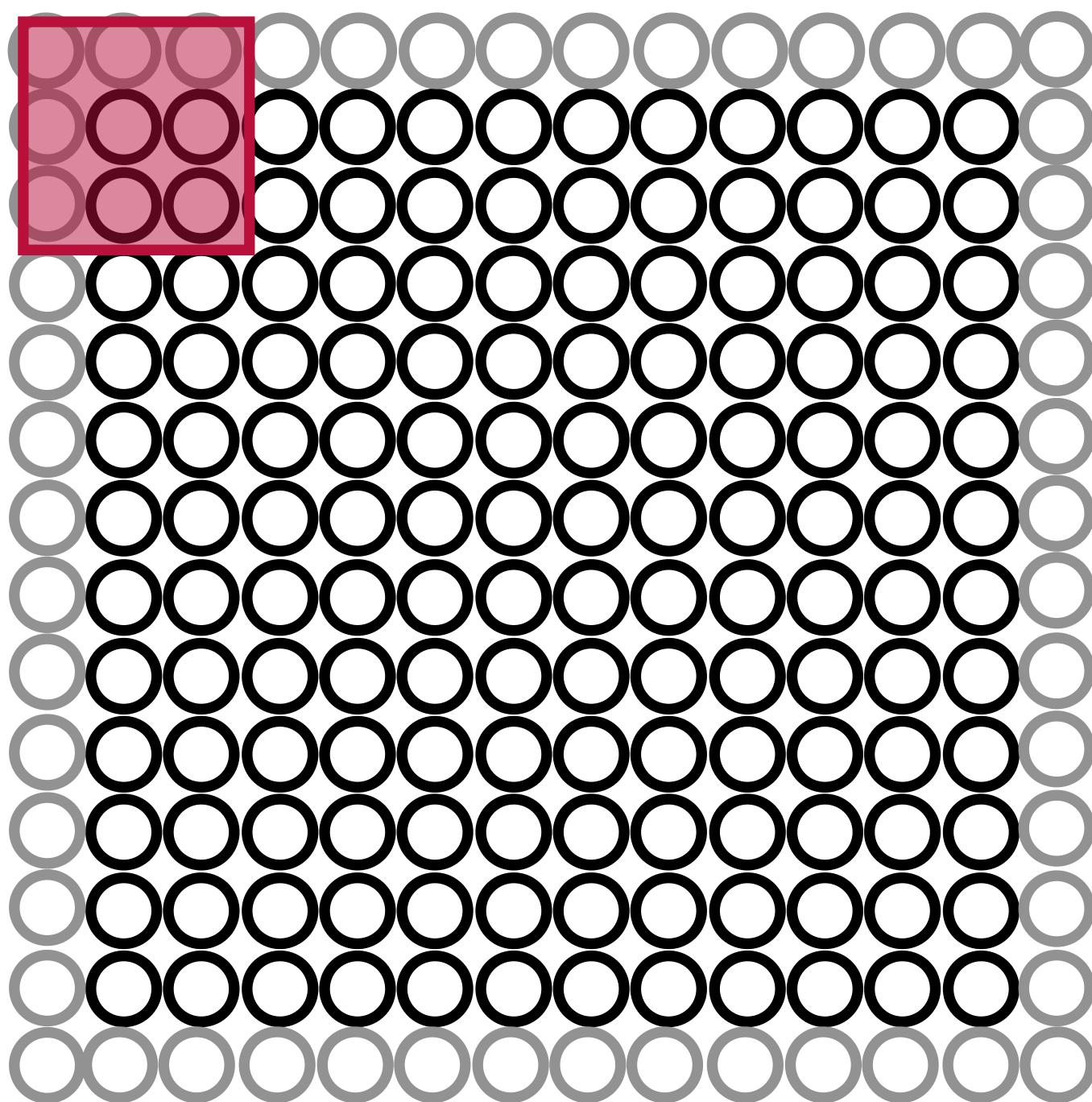
Convolutional Neural Networks (CNN)

Praxis

**Convolution mit einem Filter größer als 1×1 ändert die Größe. Äußere Pixel werden z.T. nicht erreicht.
Lösung: Padding. Die Eingabe wird am Rand erweitert.**

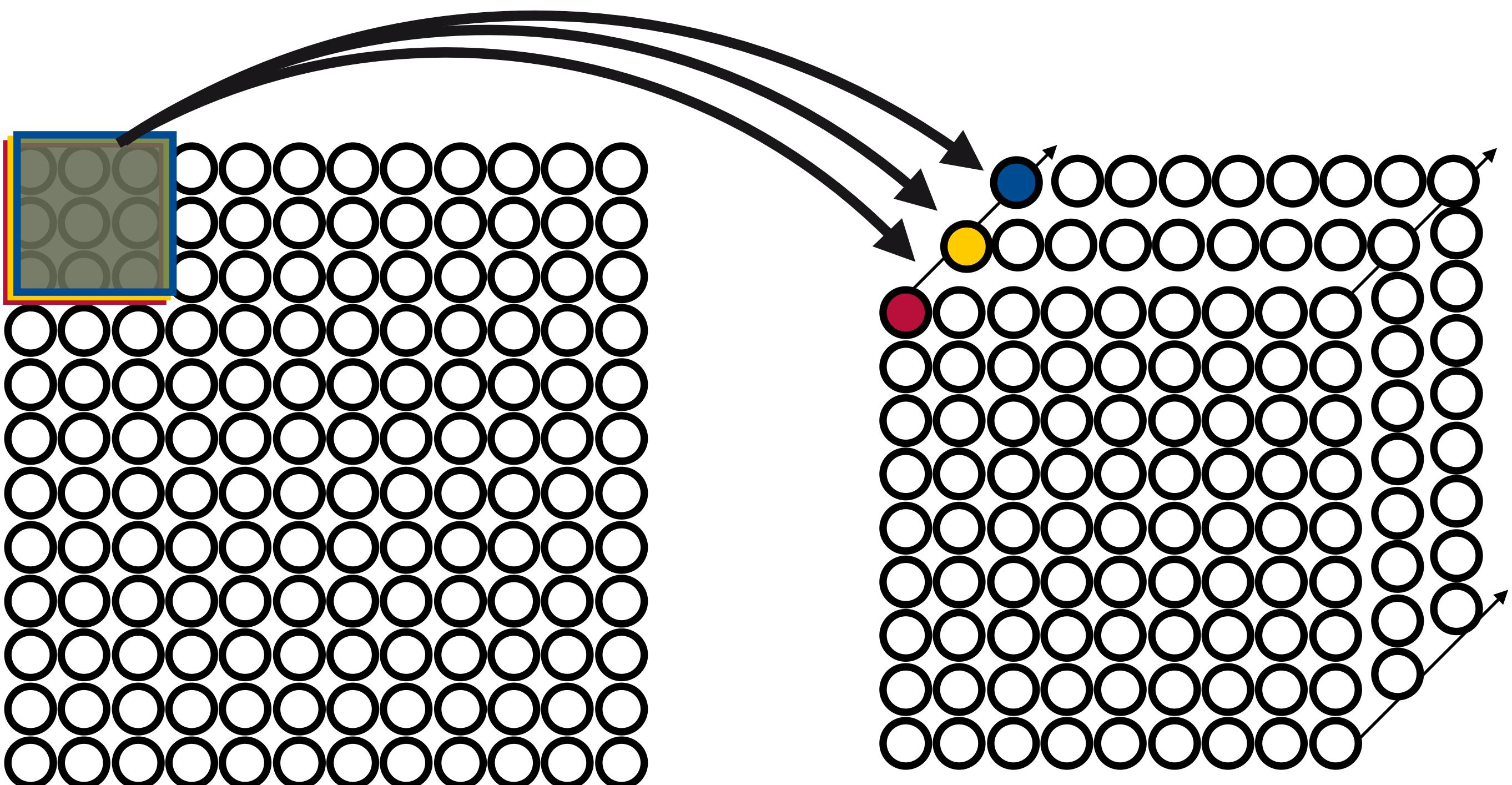


- Möglichkeiten für das Padding:
 - 0 als Defaultwert
 - Kopieren des letzten Pixels
 - Kopieren des Pixels auf der anderen Seite des Bildes
 -
- Achtung: Je nach fachlicher Aufgabe können unterschiedliche Formen des Paddings einen unterschiedlichen Effekt haben!



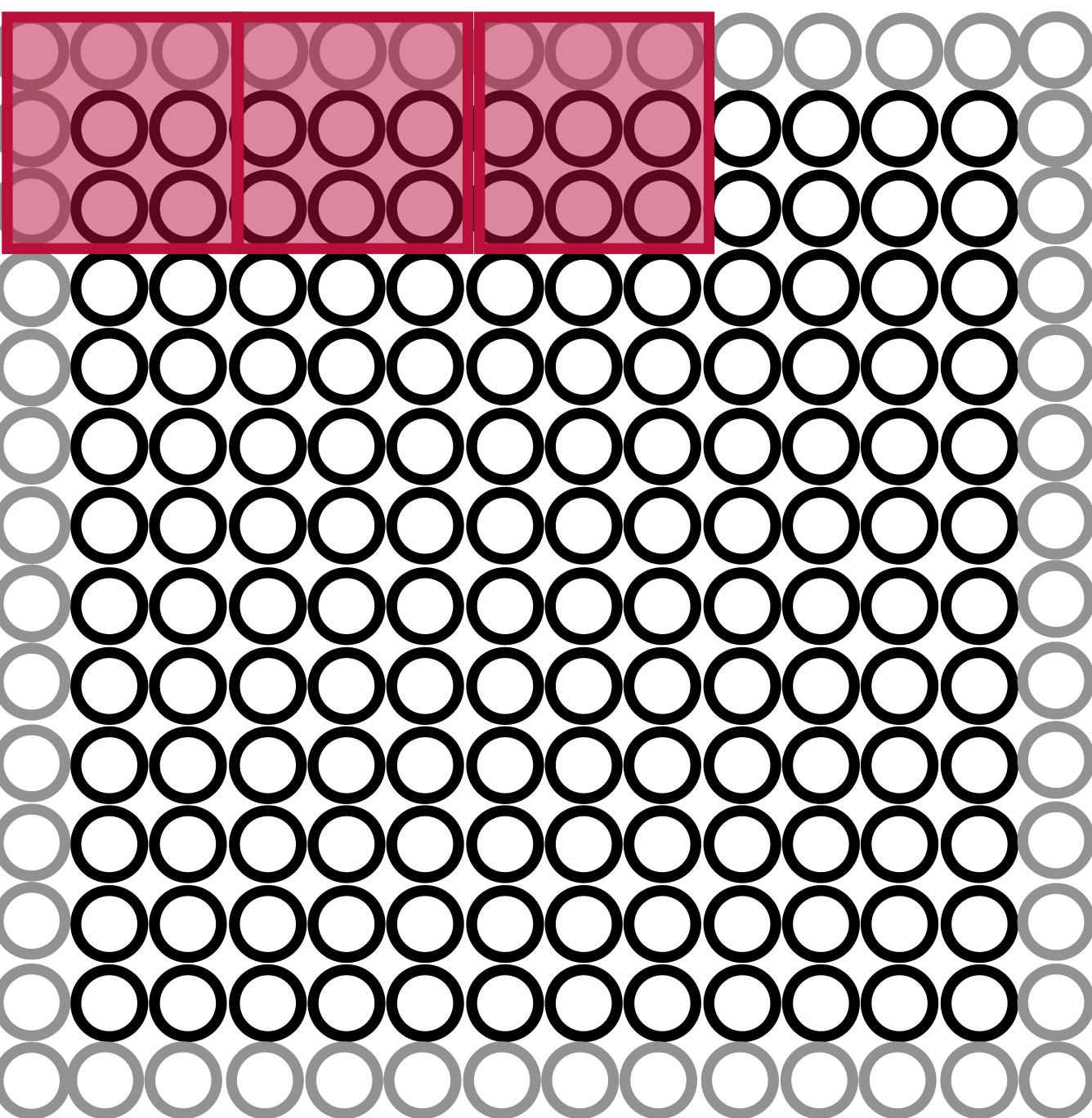
Problem: Größe des Datenflusses

- Bei komplexen Problemen kann eine große Anzahl von Filtern notwendig sein.
 - Die Größe der Ausgabe kann dadurch signifikant ansteigen (jeder Filter erzeugt eine eigene Featuremap)
 - Aus Performanzgründen kann es sinnvoll sein den Datenfluss möglichst klein zu halten.



- Der Hyperparameter Stride beschreibt die Schrittgröße des Sliding-Windows.
- Die Größe der Ausgabe wird dadurch minimiert.
- Hochauflösende Bilder werden dadurch signifikant in ihrer Größe reduziert.
- Achtung: Sehr kleine Features/Details werden ggf. nicht mehr erfasst bzw. übersprungen.

Stride der Größe 3 (mit Padding)



Codebeispiel Convolution

- Ein Convolutional-Layer in PyTorch erfordert als Eingabe:
 - Anzahl der eingehenden Features / Dimensionen
 - Anzahl der zu lernenden Filter
 - Größe der Filter
 - Weitere Angaben z.B. zum Padding

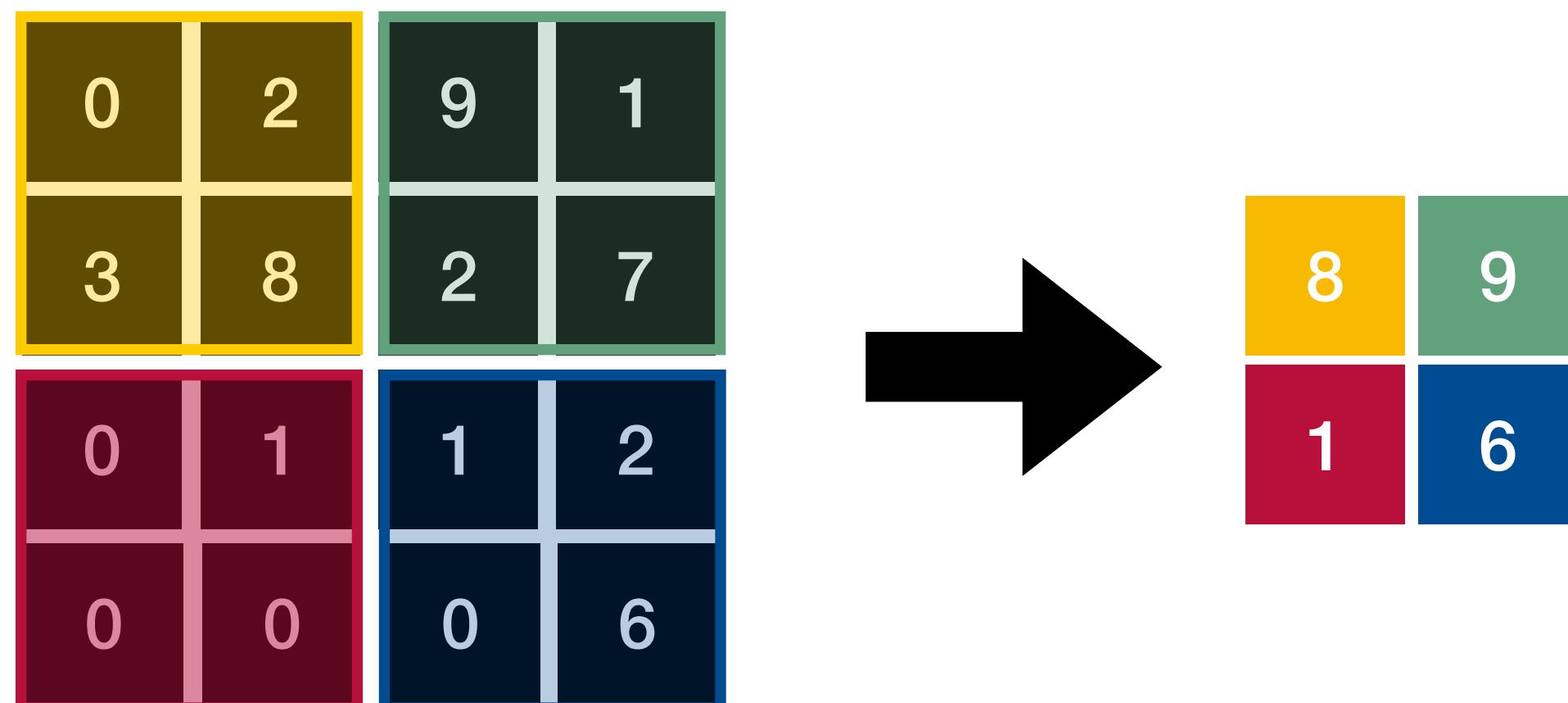
```
conv=nn.Conv2d(3,8,(3,3),stride=1, padding=1, padding_mode="zeros")  
  
x=torch.randn(4,3, 100,100)  
y=conv(x)  
  
y.shape  
-> torch.Size([4, 8, 100, 100])
```



MaxPooling

- MaxPooling wählt den maximalen Wert in einem Bereich, welcher durch einen Filter erzeugt wird.
- Die Ausgabe wird in der Größe verringert.
- Informationen über das Vorliegen eines Features in einem bestimmten Bereich wird bestmöglich erhalten.

MaxPooling mit der Größe 2×2 und Stride 2



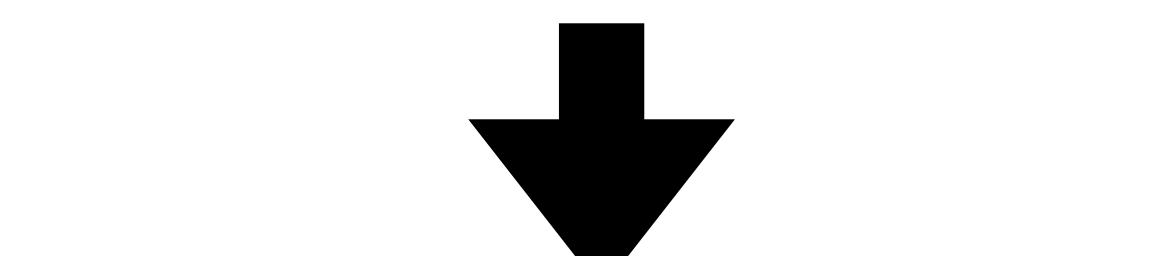
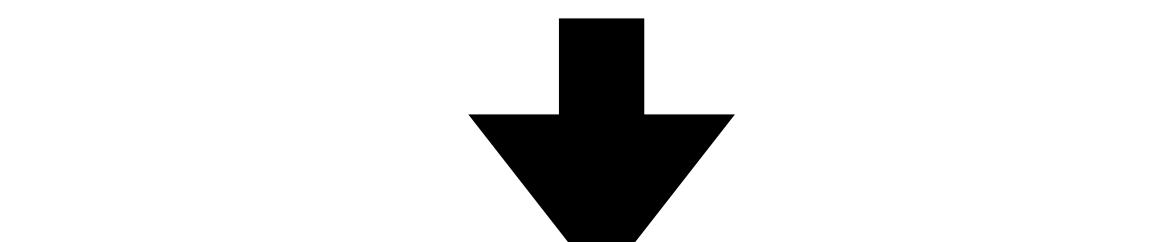
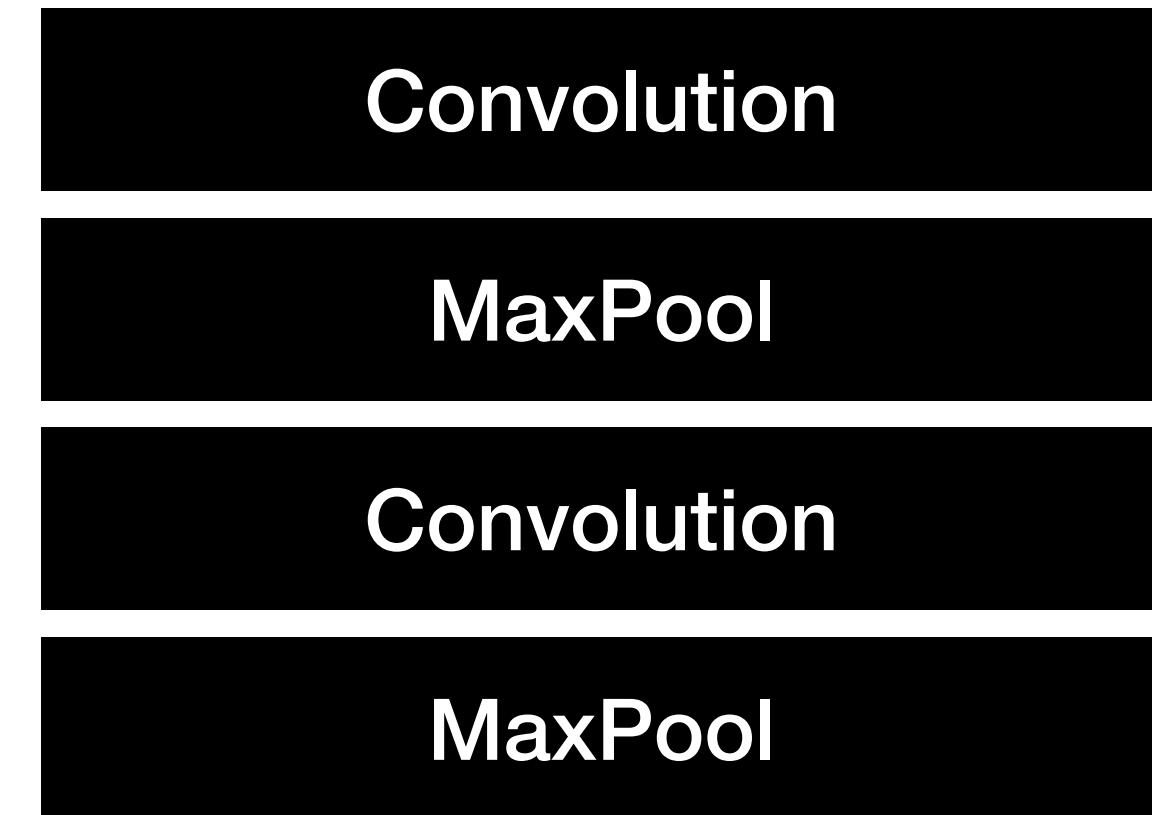
Codebeispiel MaxPooling

- Ein MaxPooling-Layer in PyTorch benötigt
 - Größe für den Kernel
 - Weitere optionale Parameter (z.B. Stride)

```
net=nn.Sequential(  
    nn.Conv2d(3,8,(3,3),stride=1, padding=1, padding_mode="zeros"),  
    nn.MaxPool2d((2,2),stride=2)  
)  
  
x=torch.randn(4,3, 100,100)  
y=net(x)  
  
y.shape  
  
-> torch.Size([4, 8, 50, 50])
```



- Die Ausgabe von Convolutional-Layers und MaxPooling-Layers ist (bedingt durch verschiedene Filter) i.d.R. mehrdimensional.
- Für den Übergang in ein „klassisches“ LinearLayer muss daher die Ausgabe einmal angepasst werden (Reshaping).
 - Aus $4 \times 4 \times 2$ wird z.B. ein 32-dimensionaler Vektor
 - Ab hier kann die Neuronale Netz wie bereits bekannt aufgebaut und verwendet werden!



Codebeispiel ConvNet

```
class MyConvNet(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv_net=nn.Sequential(
            self.block(3,16),
            self.block(16,8),
            self.block(8,4)
        )

        self.head=nn.Sequential(
            nn.Linear(400,100),
            nn.ReLU(),
            nn.Linear(100,3),
            nn.Sigmoid()
        )

    def block(self,dim_in,n_filter, conv_kernel_size=(3,3), pool_kernel_size=(2,2)):
        return nn.Sequential(
            nn.Conv2d(dim_in,n_filter, conv_kernel_size),
            nn.ReLU(),
            nn.MaxPool2d(pool_kernel_size)
        )

    def forward(self,x):
        y=self.conv_net(x)
        y=y.reshape(-1,400)
        return self.head(y)

net=MyConvNet()

x=torch.randn(4,3, 100,100)
net(x)
```



Codebeispiel ConvNet

```
class MyConvNet(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv_net=nn.Sequential(
            self.block(3,16),
            self.block(16,8),
            self.block(8,4)
        )
        self.head=nn.Sequential(
            nn.Linear(400,100),
            nn.ReLU(),
            nn.Linear(100,3),
            nn.Sigmoid()
        )

    def block(self,dim_in,n_filter, conv_kernel_size=(3,3), pool_kernel_size=(2,2)):
        return nn.Sequential(
            nn.Conv2d(dim_in,n_filter, conv_kernel_size),
            nn.ReLU(),
            nn.MaxPool2d(pool_kernel_size)
        )

    def forward(self,x):
        y=self.conv_net(x)
        y=y.reshape(-1,400)
        return self.head(y)

net=MyConvNet()

x=torch.randn(4,3, 100,100)
net(x)
```

```
MyConvNet(
    (conv_net): Sequential(
        (0): Sequential(
            (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
            (1): ReLU()
            (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
        )
        (1): Sequential(
            (0): Conv2d(16, 8, kernel_size=(3, 3), stride=(1, 1))
            (1): ReLU()
            (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
        )
        (2): Sequential(
            (0): Conv2d(8, 4, kernel_size=(3, 3), stride=(1, 1))
            (1): ReLU()
            (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
        )
    )
    (head): Sequential(
        (0): Linear(in_features=400, out_features=100, bias=True)
        (1): ReLU()
        (2): Linear(in_features=100, out_features=3, bias=True)
        (3): Sigmoid()
    )
)
```

