Deep Learning

Grundlagen Grundbegriffe Grundübungen

Alfatraining Kurs 2025-08 Dozent: Karsten Flügge

Themen des Tages

Tag 6

Convolutional Neural Network (CNN) Bildklassifizierung

Reshaping-Schichten, Flatten, Pooling-Schichten, Global-Average-Pooling

Themen des Tages

Perceptron Synonyme:
Neuron ≈ Synapses
Linear Layer
Dense
Fully Connected layer

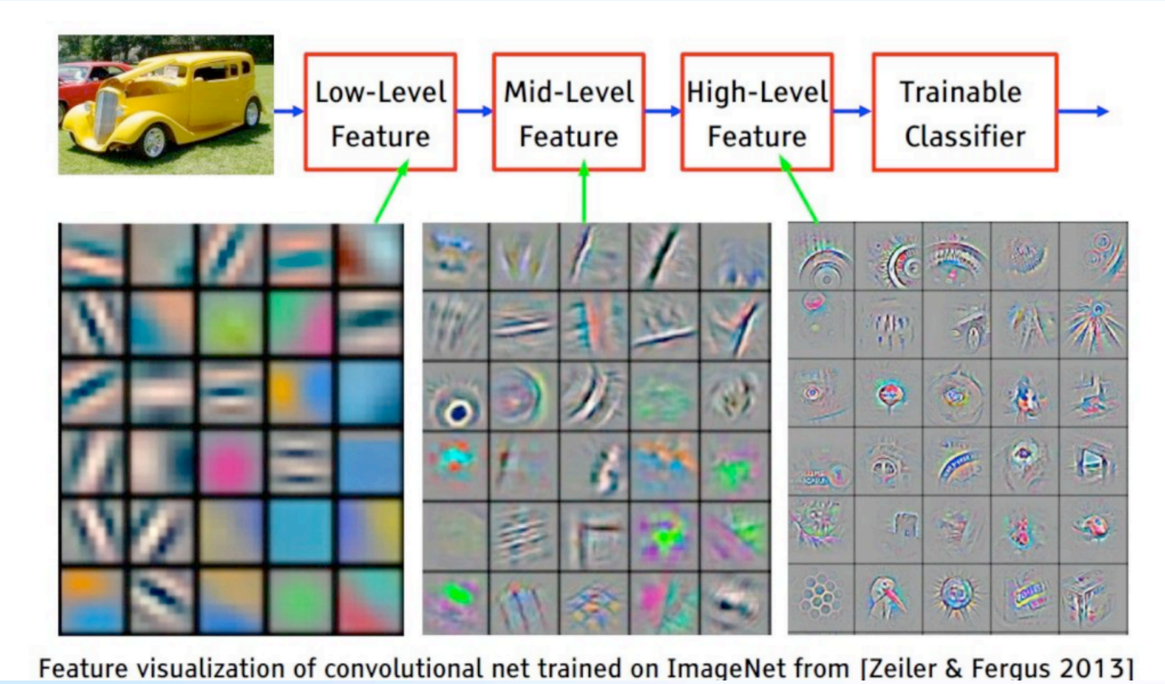
Vorteil: Alle Zusammenhänge werden gefunden
Nachteil: Teuer in der Berechnung

Convolutional Neural Network

Convolutional Neural Network (CNN)

Ein „**Faltendes neuronales Netzwerk**“ ist ein (**optisches**) Netzwerk in welchem die fordersten Schichten optischer Mustererkennung der Biologie nachempfunden wurde.

Konkret wird das Input Bild nach zunehmend komplexeren **Mustern** 'abgesucht'/'durchstriffen':



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

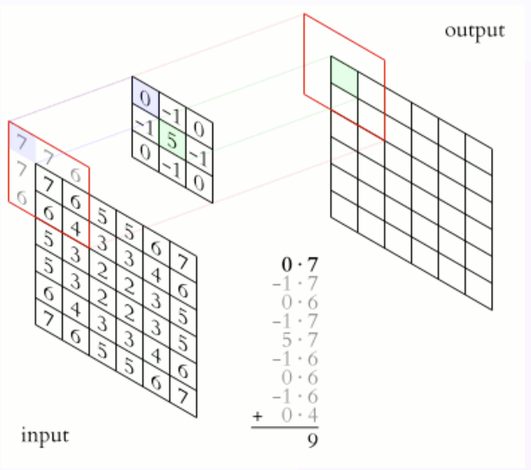
Convolutional Neural Network

Was heisst convolution „Faltung” in CNN

Idee: **Messe** die **Übereinstimmung** eines **Musters** mit einem Bildsegment.

Mathematisch: Multipliziere zwei Matrizen punktweise (skalar) und messe den 'Ausschlag'

bei uns:
Bild-Helligkeits-Matrix



Geschichte

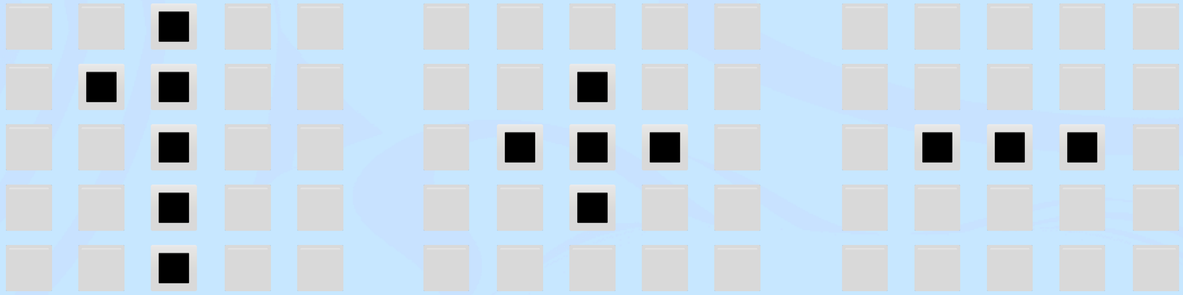
Convolution:

con- (together) + *volvere* “to roll”

Ursprünglich: Multipliziere Frequenz und messe die Kohärenz / **Resonanz**
mit der zu untersuchenden Funktion durch Integration, oder diskrete Summe.

Einfaches Beispiel

Unterscheide Zeichen 1, +, - als 5x5 Pixel-Font:



Längs-Balken Filter
[[0, 1, 0],
[0, 1, 0],
[0, 1, 0]]

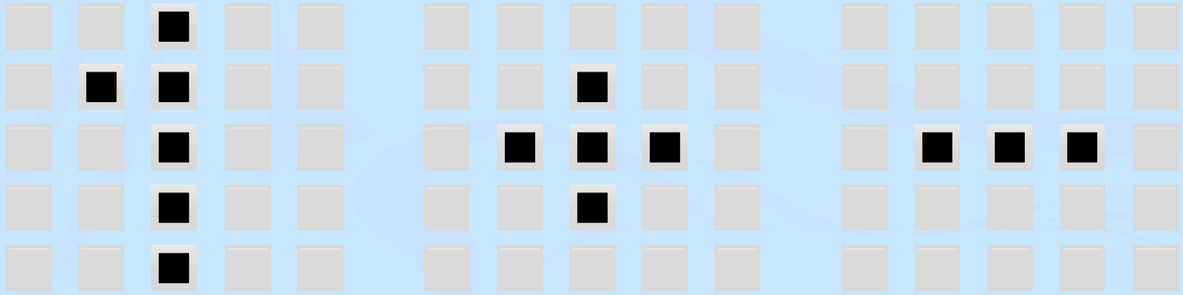
Quer-Balken Filter
[[0, 0, 0],
[1, 1, 1],
[0, 0, 0]]

entspricht 3x3 Schablonen

Zähle Punkte, an welchen Schablone und Bildpixel beide aktiv sind

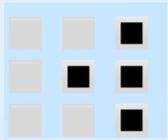
Einfaches Beispiel

Unterscheide Zeichen 1, +, - als 5x5 Pixel-Font:



Längs-Balken Filter
[[0, 1, 0],
[0, 1, 0],
[0, 1, 0]]

Quer-Balken Filter
[[0, 0, 0],
[1, 1, 1],
[0, 0, 0]]

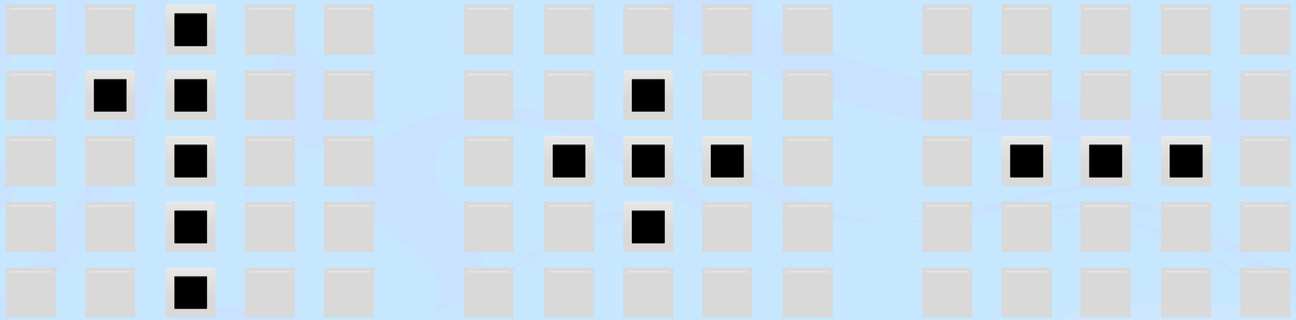


entspricht 3x3 Schablonen

Zähle Punkte, an welchen Schablone und Bildpixel beide aktiv sind

Aufgabe

Unterscheide Zeichen 1, +, - als 5x5 Pixel-Font:



Längs-Balken Filter : oben links

[[0, 1, 0],
[0, 1, 0], * = **1 (nur der in der Mitte trifft ■ auf ■)**
[0, 1, 0]]

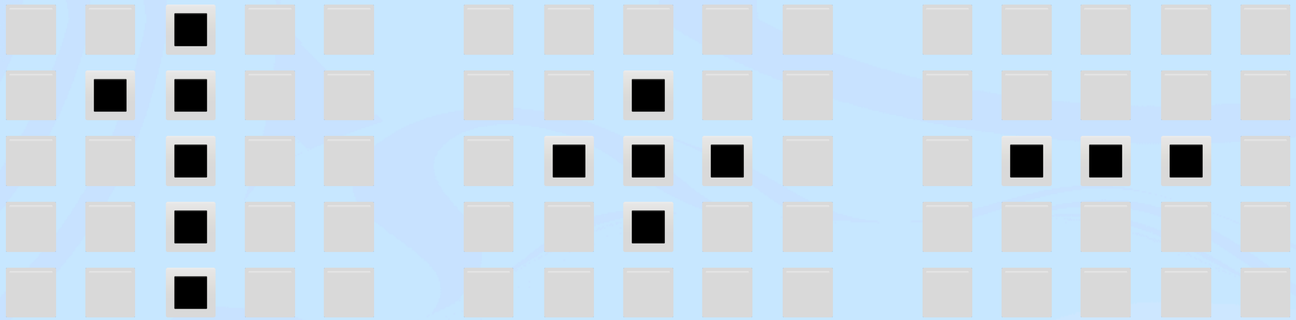
Feature Activation Map für "1":
Zähle Übereinstimmungen pro Position:

[[***1***,3,0],
[1,3,0],
[0,3,0]]

a) Berechne Quer-Balken Aktivierungen für "1" "+" und "-"
b) Berechne Längs-Balken Aktivierungen für "1" "+" und "-"

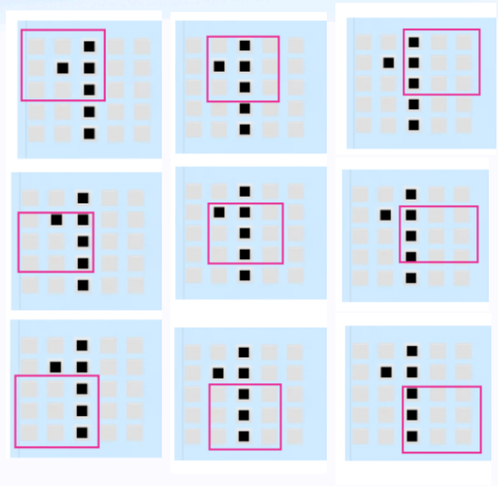
Aufgabe

Unterscheide Zeichen 1, +, - als 5x5 Pixel-Font:



Längs-Balken Filter : oben mitte

[[0, 1, 0],
[0, 1, 0], * **= 3 (alle mittleren ■ auf ■ treffen)**
[0, 1, 0]]

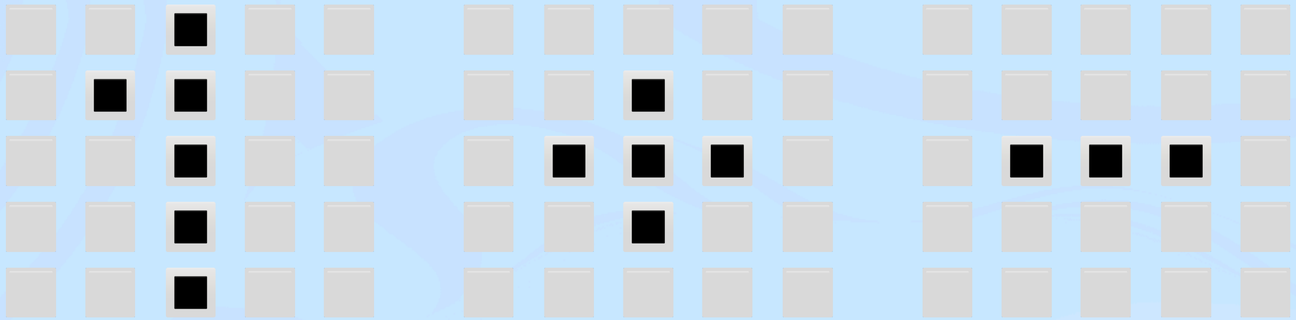


Feature Activation Map für "1":
Zähle Übereinstimmungen pro Position:
[[1,**3**,0],
[1,3,0],
[0,3,0]]

a) Berechne Quer-Balken Aktivierungen für "1" "+" und "-"
b) Berechne Längs-Balken Aktivierungen für "1" "+" und "-"

Aufgabe

Unterscheide Zeichen 1, +, - als 5x5 Pixel-Font:



Längs-Balken Filter : oben rechts

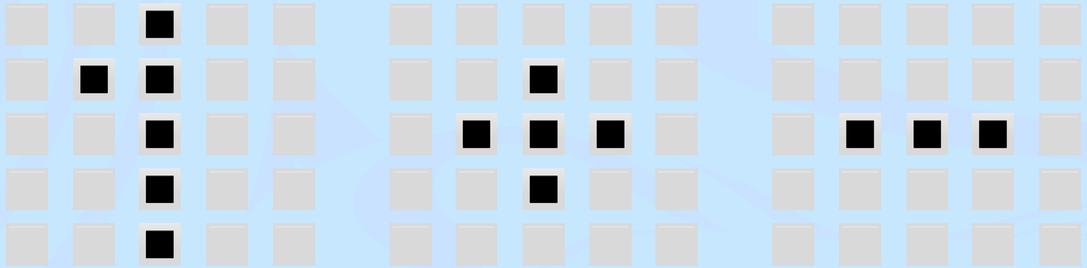
[[0, 1, 0],
[0, 1, 0], * = **0 (kein ■ trifft auf ■)**
[0, 1, 0]]

Feature Activation Map für "1":
Zähle Übereinstimmungen pro Position:
[[1,3,***0***],
[1,3,0],
[0,3,0]]

a) Berechne Quer-Balken Aktivierungen für "1" "+" und "-"
b) Berechne Längs-Balken Aktivierungen für "1" "+" und "-"

Einfaches Beispiel

Unterscheide Zeichen 1, +, - als 5x5 Pixel-Font:



Längs-Balken Filter

```
[[0, 1, 0],
 [0, 1, 0],
 [0, 1, 0]]
```

"1":

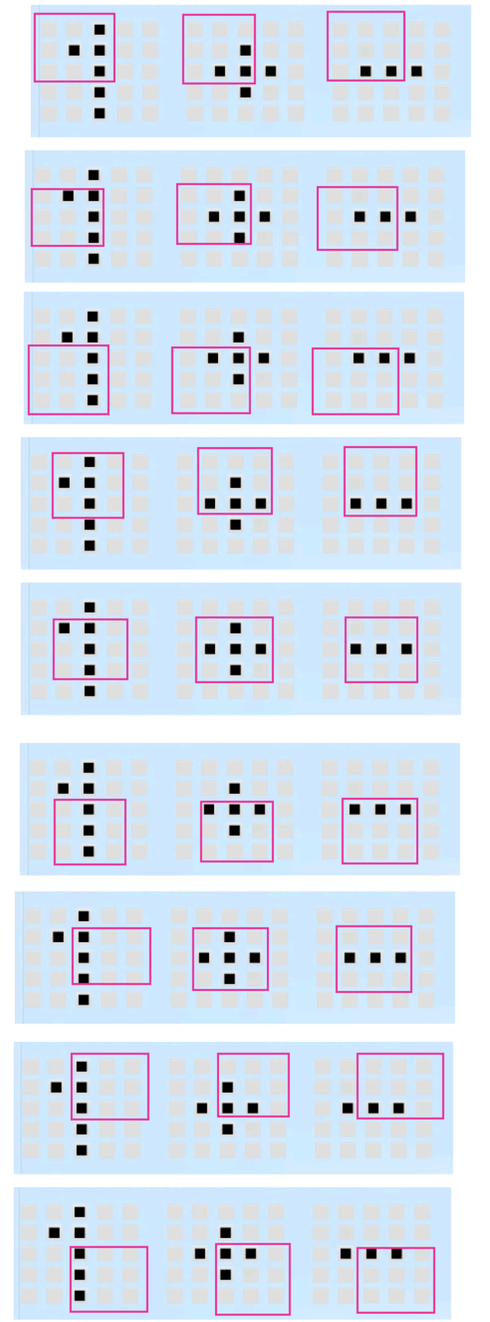
```
[[1,3,0],
 [1,3,0],
 [0,3,0]]
```

"+":

```
[[1,2,1],
 [1,3,1],
 [1,2,1]]
```

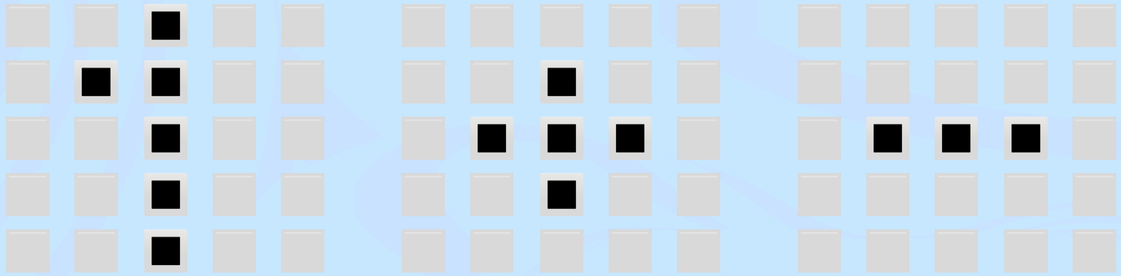
"-":

```
[[1,1,1],
 [1,1,1],
 [1,1,1]]
```



Einfaches Beispiel

Unterscheide Zeichen 1, +, - als 5x5 Pixel-Font:



Quer-Balken Filter
[[0, 0, 0],
[1, 1, 1],
[0, 0, 0]]

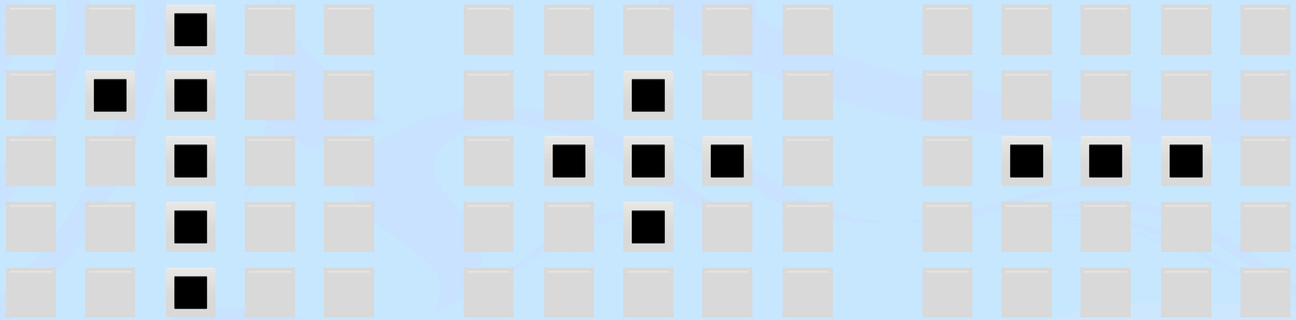
"1": Feature Map vom Quer-Balken Filter
[[2,2,1],
[1,1,1],
[1,1,1]]

"+":
[[1,1,1],
[2,3,2],
[1,1,1]]

"-":
[[0,0,0],
[2,3,2],
[0,0,0]]

Aufgabe

Unterscheide Zeichen 1, +, - als 5x5 Pixel-Font:



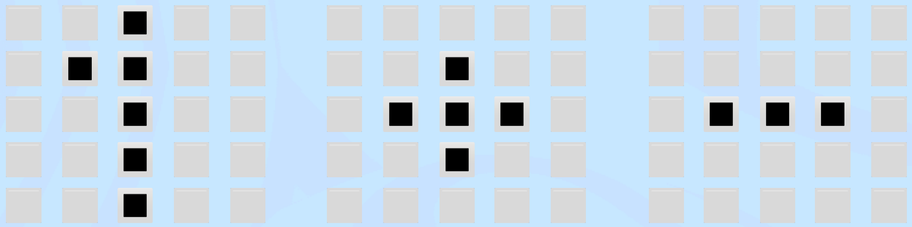
Längs-Balken Filter
[[0, 1, 0],
[0, 1, 0],
[0, 1, 0]]

Feature Activation Map für "1":
Zähle Übereinstimmungen pro Position:
[[1,3,0],
[1,3,0],
[0,3,0]]

a) Berechne Quer-Balken Aktivierungen für "1" "+" und "-"
b) Berechne Längs-Balken Aktivierungen für "1" "+" und "-"

Bessere Kanten Filter

Unterscheide Zeichen 1, +, - als 5x5 Pixel-Font:



<u>Längs</u>-Balken Filter mit -1

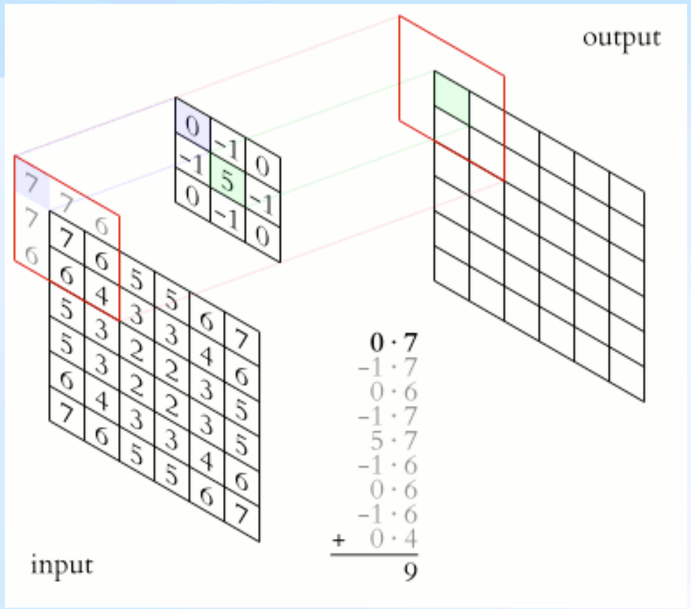
[[0, 1, **-1**],
 [0, 1, **-1**],
 [0, 1, **-1**]]

- **Kontrast**
- **Kein horizontales Signal bei vertikalen Linien!**
- **Abweicher von dem Muster werden bestraft!**

Muster

Messe die Übereinstimmung eines **Musters** mit einem Bildsegment.

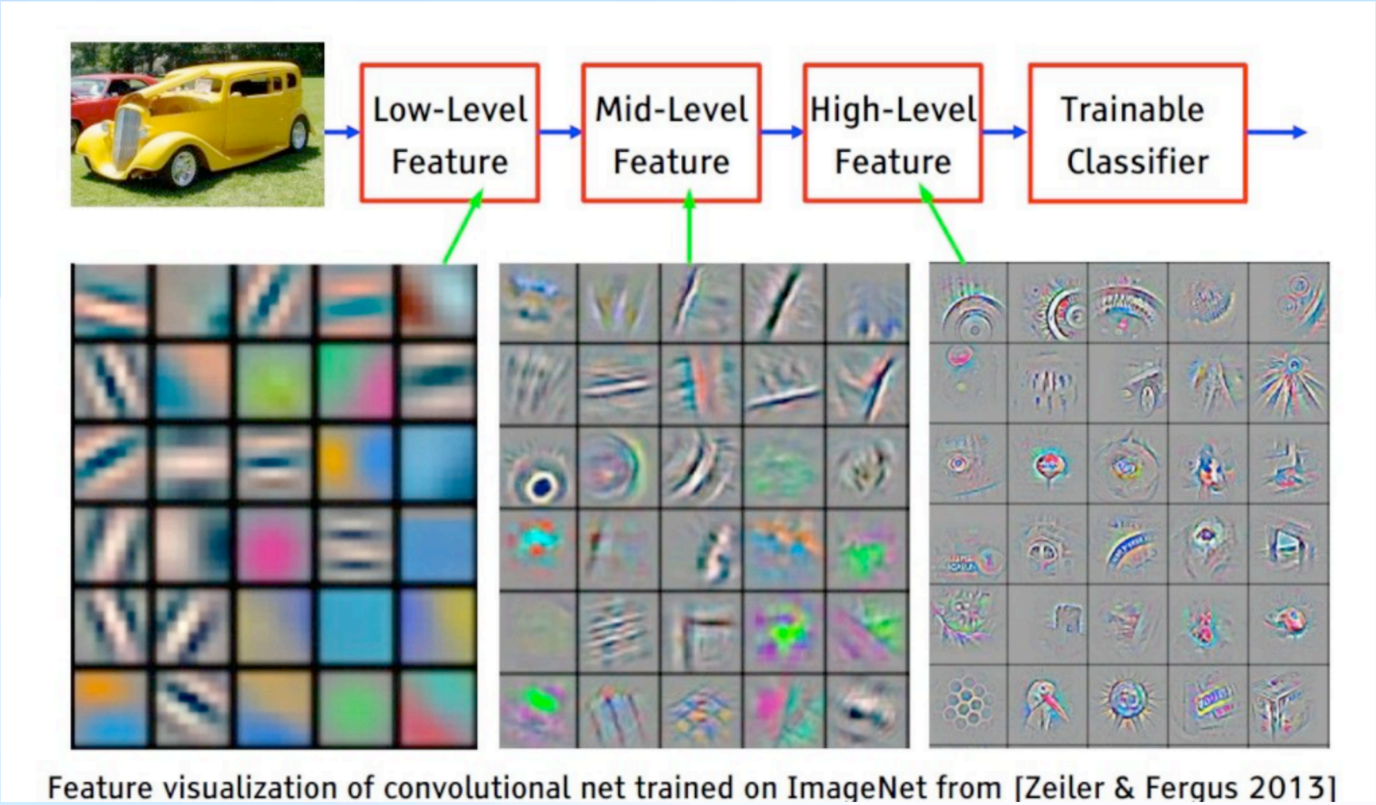
Ein Muster ist eine **kleine Matrix** mit welcher schrittweise ein Bild abgetastet wird. Synonyme sind **Schablone, Filter, *kernel*.**
Wir präferieren "**Filter**", weil der Begriff im Englischen und Deutschen passt.



Convolutional Neural Network

⚠️💡 Die Muster des CNN werden nicht manuell mitgegeben sondern bilden sich im Laufe des Lernprozesses automatisch aus!

Beginnend mit **Linien, Streifen** und **Farben** in der **untersten Schicht** nimmt die Komplexität der Muster zu. In mittleren Schichten finden sich **Kreise, Formen** und **Texturen.** Zuletzt finden sich **Segmente, Körperteile** und ganze **Objekte.** Letztere werden aber meist nicht mehr durch eine CNN Schicht abgebildet, sondern durch eine abstrakte **Kombination vorheriger Teile** mittels unserer bekannten Perceptron Schichten ("fully connected")

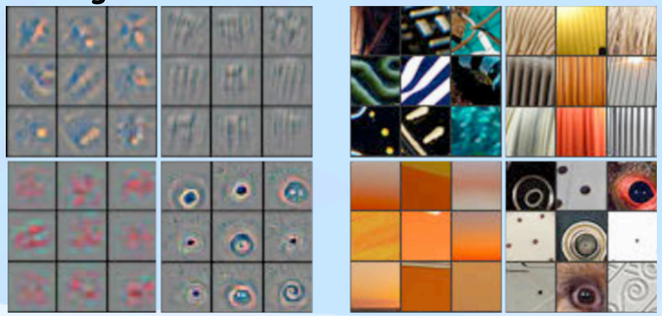


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Convolutional Neural Network

Das Input Bild wird nach zunehmend komplexeren Mustern 'abgesucht'/'durchstriffen':

- **Linien, Streifen** und **Farben** in der **untersten Schicht**
- **Kreise, Formen** und **Texturen**
- **Segmente, Körperteile**
- **Objekte** als abstrakte **Kombination vorheriger Teile**



Layer 2

Layer 5

⚠️💡 Die Muster des CNN werden nicht manuell mitgegeben sondern bilden sich im Laufe des Lernprozesses automatisch aus! Das war ein riesiger **Erfolg** nicht nur **algorithmisch** sondern auch **philosophisch.**

Eigenschaften

- *Lokale* Konnektivität
  - anders als bei "fully connected" Schichten begrenzt man sich auf Teilabschnitte
  - bei uns nur 3*3 input => 3*3 output *viel effizienter*!
- 2D- oder 3D-Anordnung der Neuronen
  - kleine **Faltungsmatrix** (Filterkernel, **filter, kernel**)
- **Geteilte Gewichte**: viel weniger Parameter
- **Bewegungsinvarianz** (globale Nutzung der selben Muster)
  - "Linien Segment ist an vielen stellen nützlich, ")
- Sehr **effizient** und parallelisierbar (GPU!)

⚠️💡 `3D heisst bei uns erstmal Matrix(Tensor) mit` **`Farbkanälen`**`, nicht räumliches abtasten. Gibt es aber auch.`

Biologische Plausibilität

Da Backpropagation kein biologisches Pendant hat, bezieht sich die Ähnlichkeit nur auf die Muster, nicht auf den Lernprozess.

"Andererseits konnte durch Untersuchungen mit fMRT gezeigt werden, dass Aktivierungsmuster einzelner Schichten eines CNNs mit den Neuronenaktivitäten in bestimmten Arealen des visuellen Cortex korrelieren, wenn sowohl das CNN als auch die menschlichen Testprobanden mit ähnlichen Aufgaben aus der Bildverarbeitung konfrontiert werden."

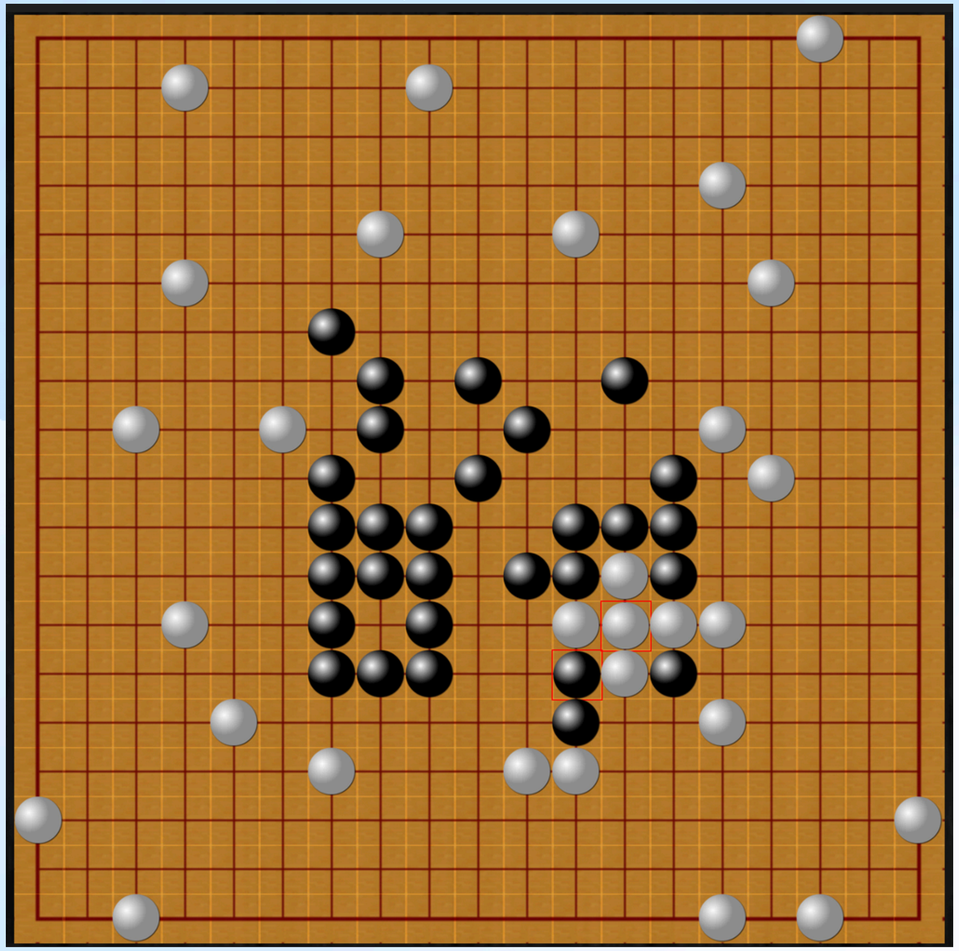
Nachweis von biologischen „**simple cells**“:
Erkennung von Kanten in bestimmten Orientierungen
direkt hinter den optischen Rezeptoren.

Eigenschaften

- **Anwendbar** nicht nur auf Bilder sondern auch **auf** 'abstrakte **Gitter**',
- z.B. **Schachbrett 8*8** (direkt ohne Bilderkennung)

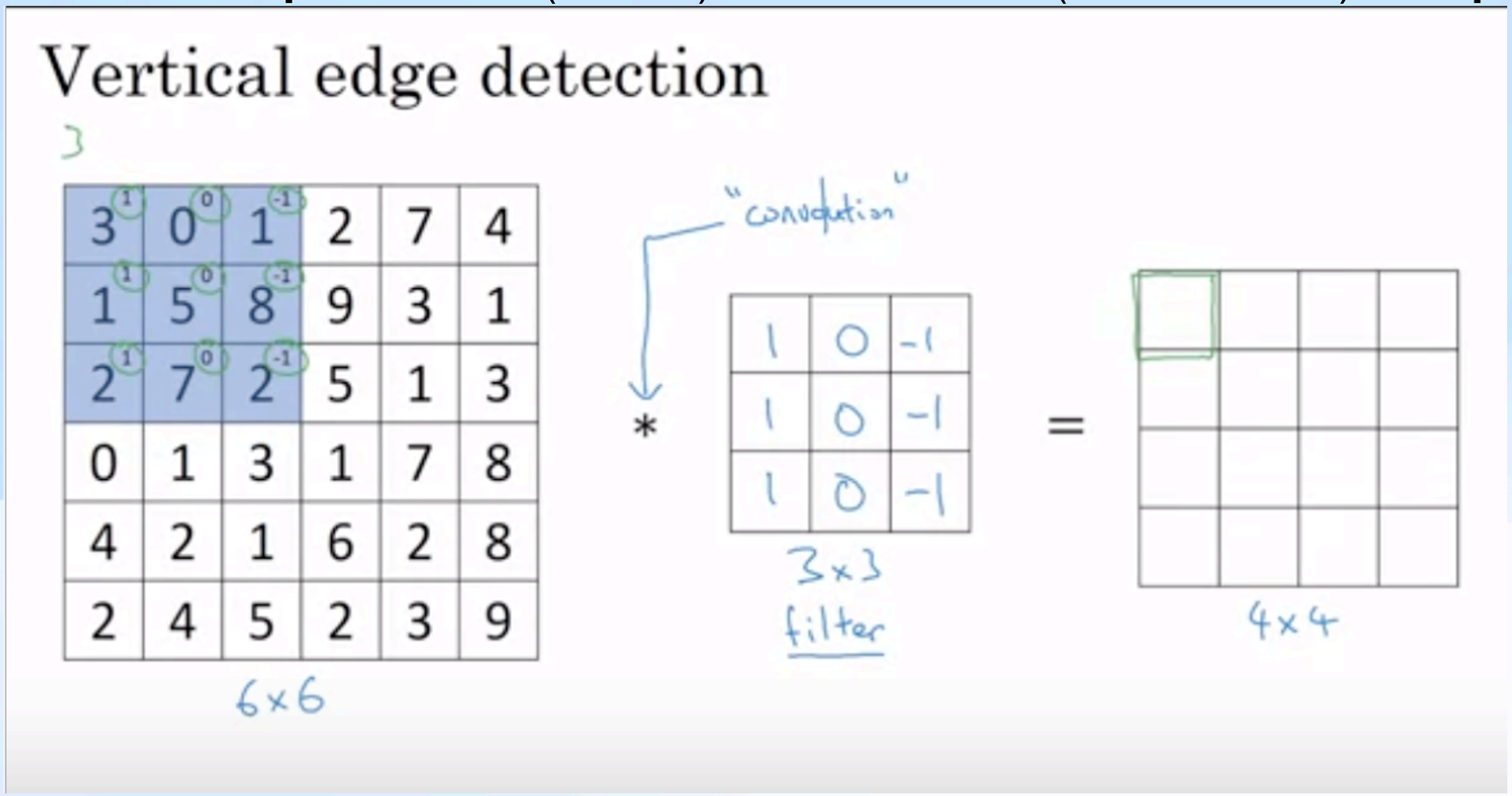
Eigenschaften

- **Anwendbar** nicht nur auf Bilder sondern auch **auf** 'abstrakte **Gitter**',
- z.B. **Go 8*8** (direkt ohne Bilderkennung) => AlphaGo Weltmeister!



Convolutional Layer

Das Bild wird **punktweise** ('skalar') mit dem Muster ('filter'/'kernel') **multipliziert**:



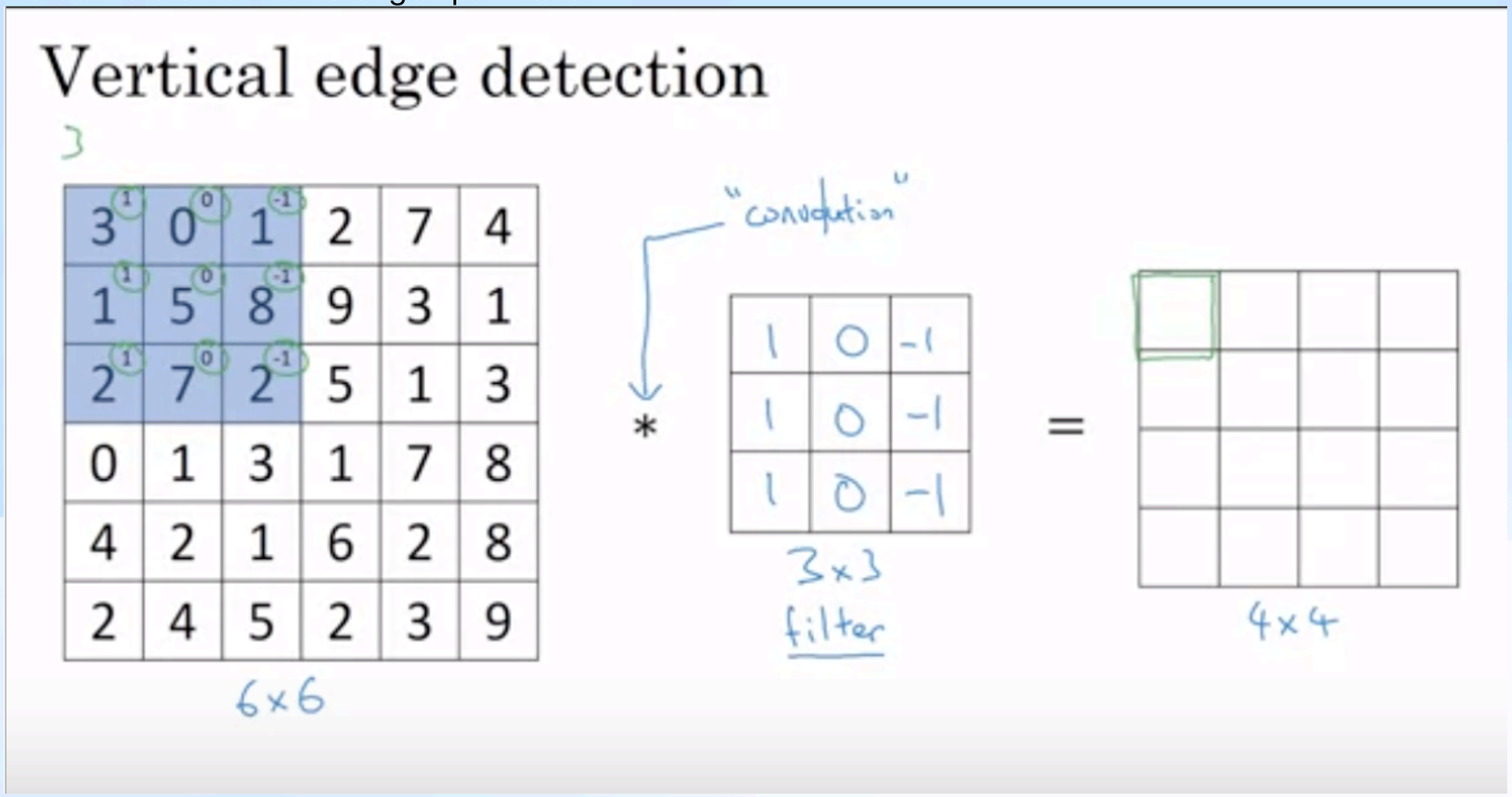
Die Summe ergibt die Aktivierung ∑∑m[i+j][i+y]*k[i]

Hier: 3*1+1*1+2*1+ 0 + 0 + 0 **- 1 - 8 - 2** = 6 - 11 = -5 … Aufgabe

Um zum Beispiel vertikale Kanten zu entdecken wird das obige Bild an 4*4 stellen 'abgestastet'.

Aufgabe

Berechne die 16 Aktivierungen per Hand:



Eintrag links oben: 3***1***+1***1***+2***1***+ 0 + 0 + 0 **- 1 - 8 - 2** = 6 - 11 = -5

2. links oben: 0*1 + 1*0 + 2*-1 + 5*1 + 8*0 + 9 *-1 + 7*1 + 0 + 5*-1 = -2+5-9+7-5=**-4**

-5 -4 0 8
-10 -2 2 3
0 -2 -4 -7

Deep Convolutional Net

Die Aktivierungen nach den Convolutional layern nennen sich
"feature vector" oder **"embedding"**
Sie enthalten die **kondensierte Information** des Ursprungs Bildes.

Mit diesen **embeddings lässt sich in verschiedener Weise weiter arbeiten:**

- **Klassifikation (Auto, Vogel, …)**
- **Cluster Analyse (ähnliche Bilder)**
- **Bild Beschreibung** ("Eine Frau hält ein Telefon")
- **Bildgenerierung und Style Transfer** ("wandle Bild in Van Gogh Gemälde um")

- Die **Klassifikation** erfolgt z.B. über unser bekanntes **Softmax**
  - (**One-Hot-encoding** mit verschiedenen 'Töpfen')

Geschichte

1968 paper by Hubel and Wiesel identified two basic visual cell types in the **brain**:
simple cells, whose output is maximized by straight edges having particular orientations within their receptive field
complex cells, which have larger **receptive fields**, whose output is insensitive to the exact position of the edges in the field.
cascading model of these two types of cells for use in pattern recognition tasks.

1980 … inspired **neocognitron** by Fukushima
"S-layer": a shared-weights receptive-field layer, later known as a convolutional layer
"C-layer": a downsampling layer that contain units whose receptive fields cover patches of previous convolutional layers. Later: **pooling**

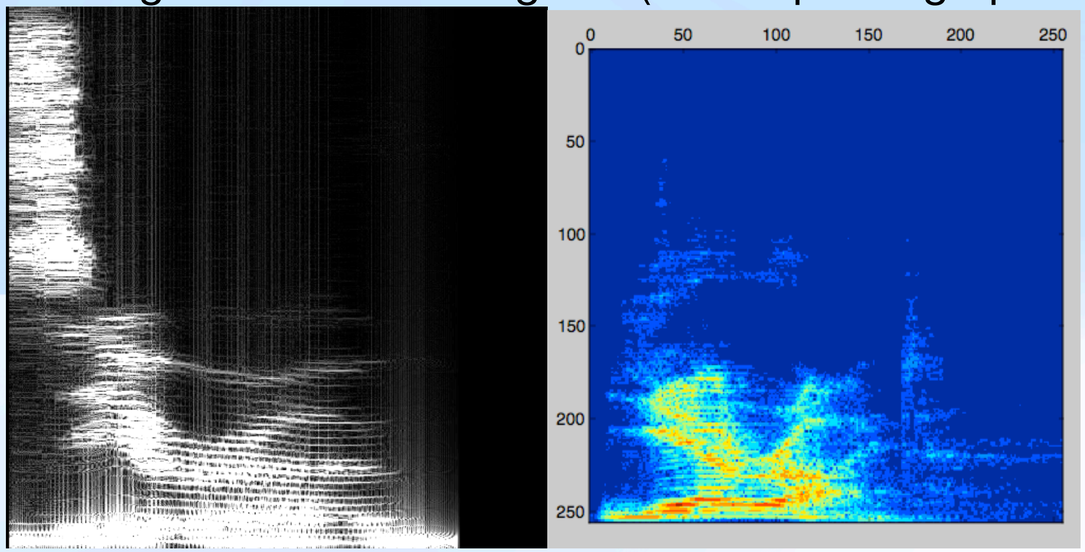
1989-1998 **LeNet** by Yann LeCun: Anwendung auf zip code recognition ≈ **MNIST**
2004, 2010 **GPU**s, **60x faster** MNIST

2012 AlexNet: Anwendung für **1000 Klassen** Bildererkennung

Geschichte

2012 AlexNet: ImageNet Rekord senkt Fehlerquote 25,8% auf 16,4%.
Seitdem nutzen alle vorne platzierten Algorithmen **CNN**-Strukturen.
Im Jahr 2016 wurde eine Fehlerquote **< 3 %** erreicht.[30]

Nutzung für akustische Signale (dank Spektrograph / Fourier Transformation)



Nutzung für andere 'artfremde' Aufgaben.

Update: **Transformer** Architektur von Text Modellen ist nun erfolgreichstes allgemeines Rechenschema, machen CNNs starke Konkurrenz.

Cifar-10

Better MNIST

The CIFAR-10 dataset contains 60,000 **32x32 color images** in 10 different classes

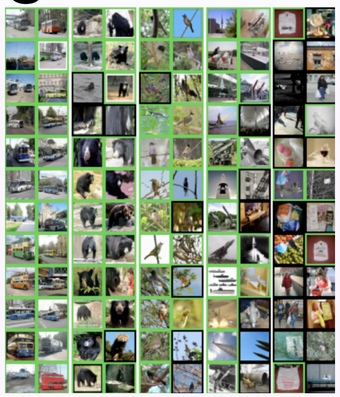
ImageNet

Komplizierteres MNIST: 1000 Klassen von Bildern: Auto, Flugzeug, ...

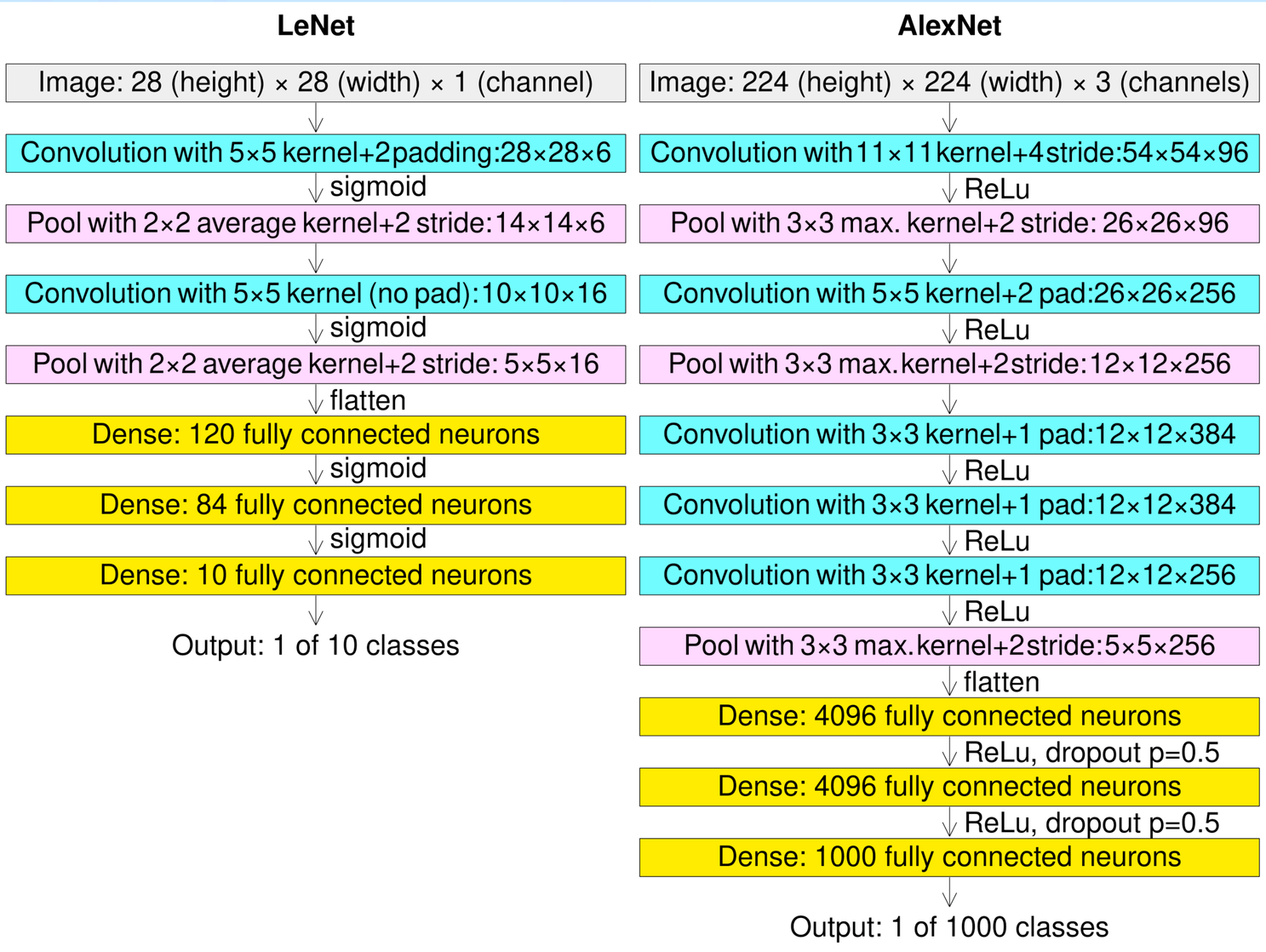
2012 **AlexNet**: ImageNet Rekord senkt Fehlerquote 25,8% auf **16,4%**
Seitdem nutzen alle vorne platzierten Algorithmen CNN-Strukturen.
Im Jahr 2016 wurde eine Fehlerquote < 3 % erreicht.[30]

Rekord 2025 ">90%" OmniVec 93%
Keine Convolution mehr

https://paperswithcode.com/sota/image-classification-on-imagenet



LeNet und AlexNet



Convolution Layer

Eingabe:
(**number** of inputs) × (input **height**) × (input **width**) × (input **channels**)

Ausgabe:
(**number** of inputs) × (**feature map height**) × (feature map **width**) × (feature map **channels**)

Klassifikation:

Linear/**Dense** layer erwartet einen Vektor, das heisst, wir müssen alle Kanäle zusammen stauchen "Flatten"

Dimension / Kanal

äussere Tensor **Dimension**:

```
Tensoren kennen wir:
0-d "Skalar" Zahl
1-d "Vektor" Liste von Zahlen
2-d "Matrizen" Liste von Liste von Zahlen
3-d "Tensor"   "Blob"
4-d …  "Tiefe der Schachtelung"
```

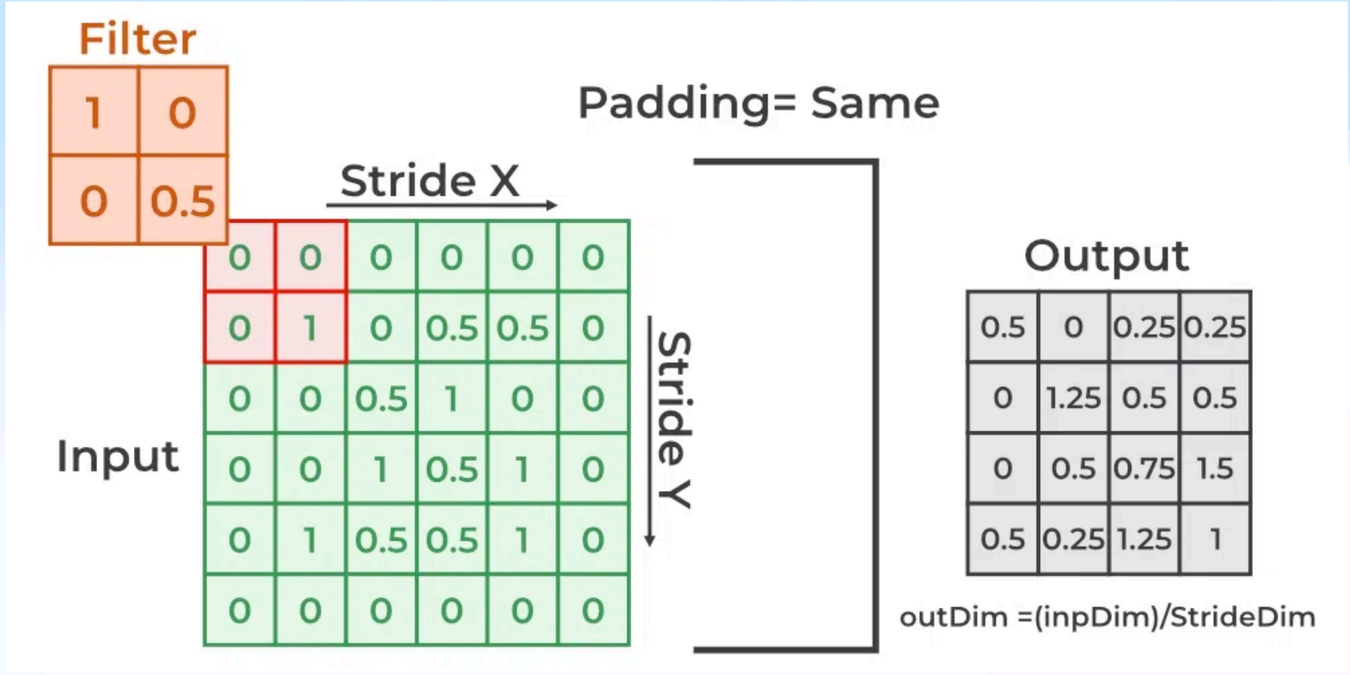
innere Dimension:
Breite "28 dimensionales Array" besser: "**Länge/Size**"
Breite * Höhe = 2D Bild oder 28*28-D ??

Farbe "3/4 dimensionales Array (mit alpha) RGB(**A)/HSV/HSL**"
hue*saturation*value/light 4-Kanal-Tensor (1d!) für Farben = Vektor mit 4 Einträgen

Kanal: a) Element im Array **b) Dimension im Array** (Doppelte Verwendung)
Z.B. "Der Farb Kanal" hat 3 (Teil) Kanäle = RGB channels "Red is a channel"

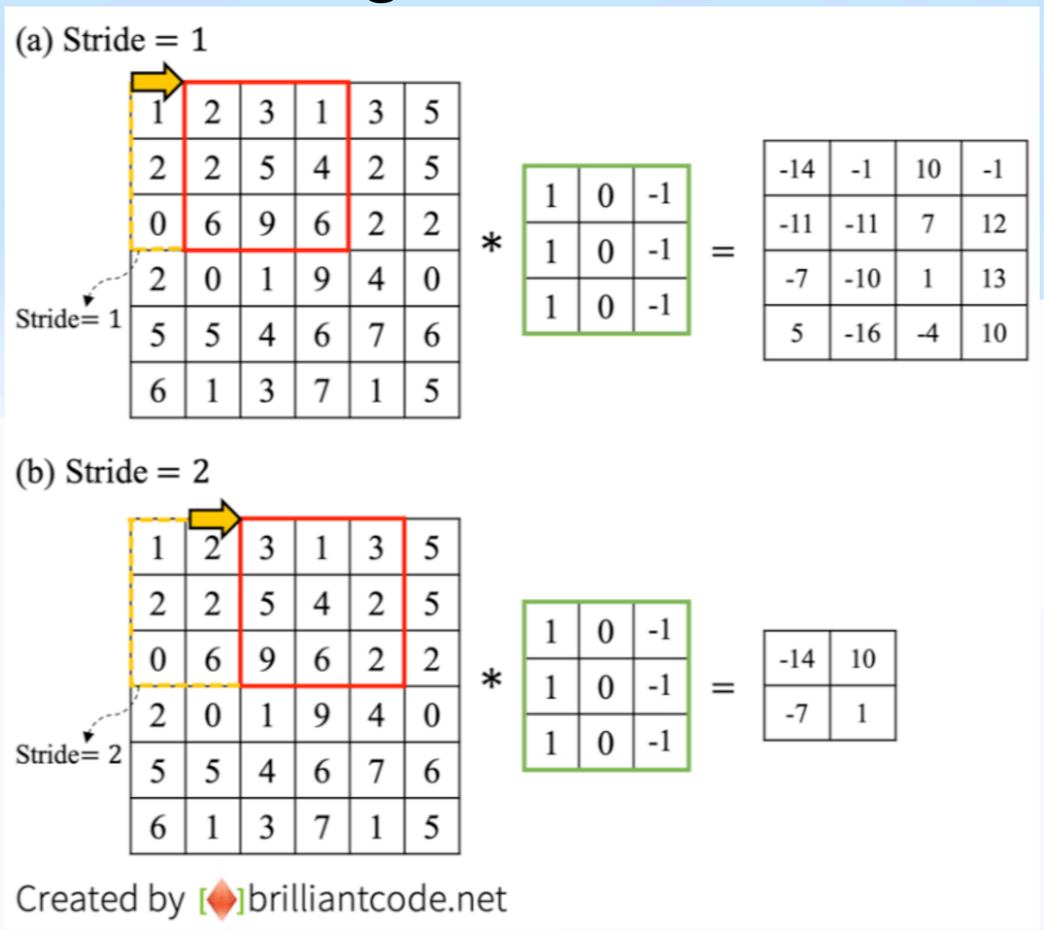
Convolution Parameter

kernel_size "**receptive field**" Höhe / Breite des Filters (meist 2,3,5,7,11 als Quadrat)
stride Schrittweite des Filters (meist 1)
padding Auffüllen der Ränder (meist **None** oder 1 "Same")



Stride

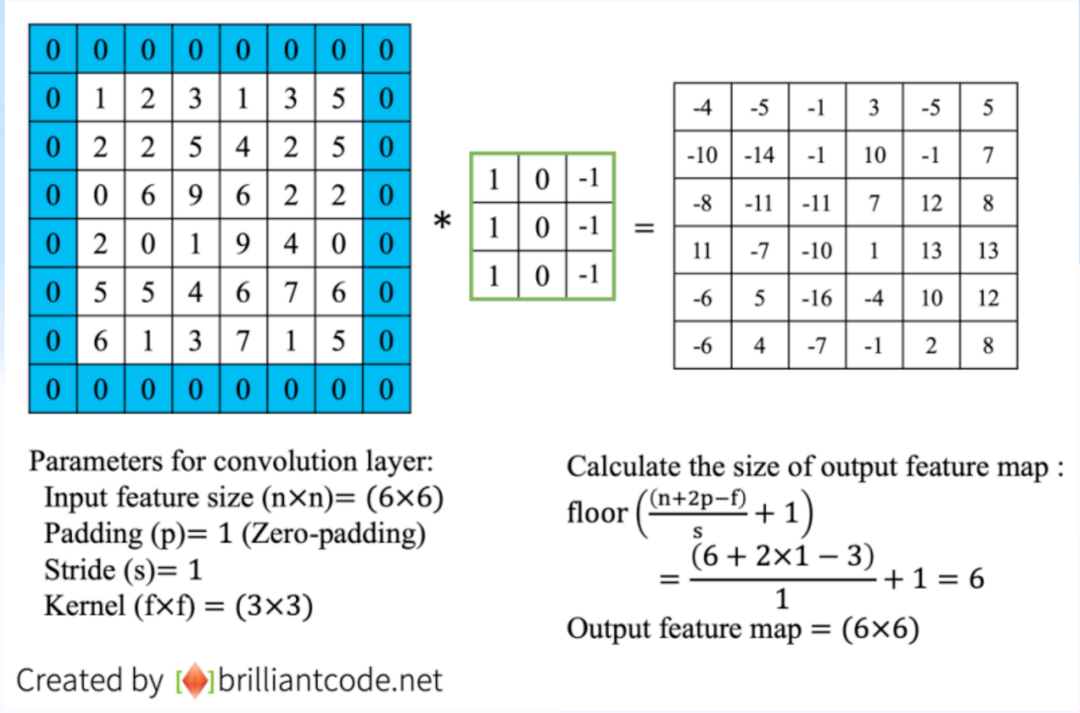
Stride = Schrittweite
Wie weit soll unser Filter in jedem Schritt weiter gerückt werden



Padding

Padding = extra Rand
Unser Filter kann über das Bild hinaus angewendet werden

padding "**same**" erzeugt selbe output Dimension abhängig von kernel size!



Padding

Hauptgründe für die Verwendung von Padding:

1. **Größenerhaltung**: Ohne Padding würde jede Faltung die räumliche Größe des Ausgabetensors verringern. Für ein Eingabebild der Größe $n * n$ und einen Filter der Größe $f * f$ mit einem Stride von 1, wäre die Größe des Ausgabebildes $(n - f + 1) * (n - f + 1)$. Durch die Anwendung von Padding kann die Ausgabegröße gleich der Eingabegröße gehalten werden, was besonders in tiefen Architekturen nützlich ist, um Informationsverlust über viele Schichten hinweg zu vermeiden.
2. **Informationsverlust vermeiden**: Ohne Padding werden die Randpixel des Eingabebildes seltener in den Faltungsoperationen verwendet als die inneren Pixel. Dies führt dazu, dass Informationen, die am Rand des Bildes liegen, weniger berücksichtigt werden. Padding fügt künstliche Ränder hinzu, sodass auch die echten Randpixel häufiger in den Berechnungen erscheinen und somit ihre Informationen besser genutzt werden.
3. **Anwendbarkeit von Filtern**: Padding ermöglicht die Anwendung eines Filters auf Bereiche, die ohne zusätzlichen **Rand** nicht **zugänglich** wären. Dies ist insbesondere bei größeren Filtern (z.B. $5 * 5$ oder $7 * 7$) relevant, wo ohne Padding die Anzahl der möglichen Faltungspositionen stark eingeschränkt wäre.

Padding

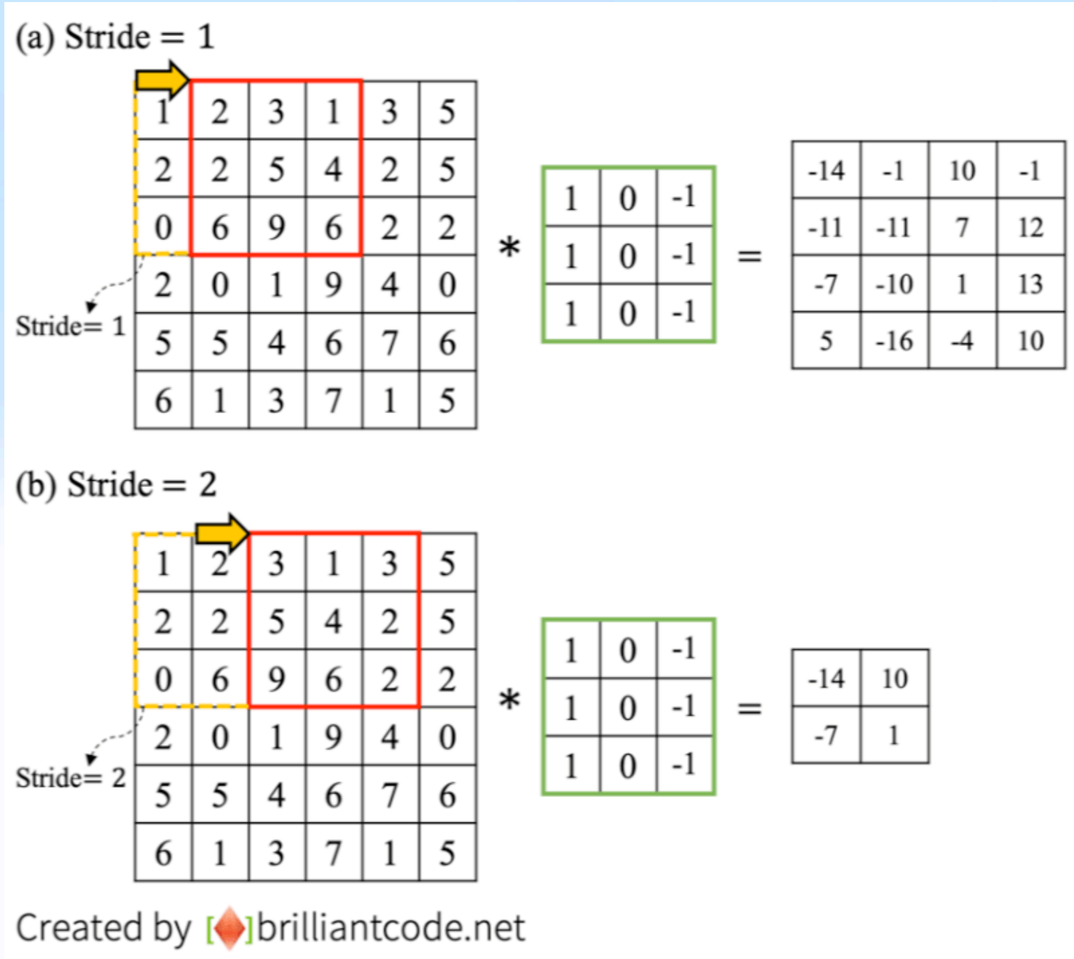
Typen von Padding

- **Zero Padding**: Dies ist die häufigste Form, bei der der **Rand des Bildes mit Nullen aufgefüllt** wird. Es hat den Vorteil, dass es die mathematische Handhabung vereinfacht, kann aber zu künstlichen Kanten im Bild führen, wenn der Filter auf die gepaddeten Bereiche angewendet wird.
- **Replicative (wiederholendes) Padding**: Die Randpixel des Originalbildes werden einfach wiederholt. Dies kann ebenfalls zu natürlicheren Ergebnissen führen als Zero Padding.
- **Reflective Padding**: Hierbei werden die Ränder des Bildes mit Spiegelungen der tatsächlichen Bildpixel aufgefüllt. Dies kann *natürlichere Übergänge erzeugen*, da es keine künstlichen Nullen einführt.

Formel

n = input width/height (hier 6)
k = kernel size (3)
s = stride 1
p = 0

Formel:
Wie groß ist unser output
n / s

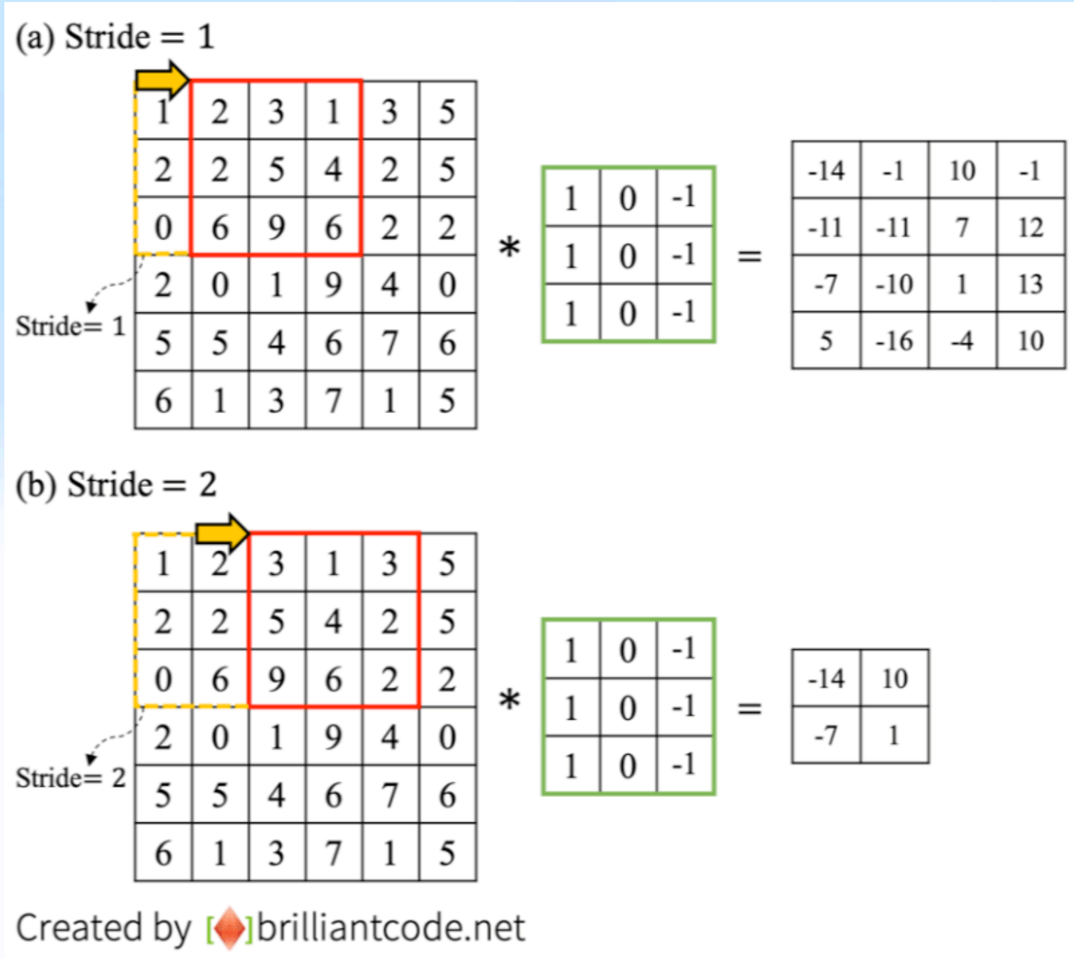


Formel

n = input width/height (hier 6)
k = kernel size (3)
s = stride 1
p = 0 (no padding)

Formel:
Wie groß ist unser output
(n + **2***p **- k**) / s + 1

mit padding p erweitern wir Bild



Bildklassifizierung

Cats vs Dogs / Waschbären
Einfaches Netzwerk mit 2 Klassen

MNIST
MNIST Bildklassifizierung kennen wir, man kann (auch) mit Conv2d die Genauigkeit erhöhen

CIFAR-10

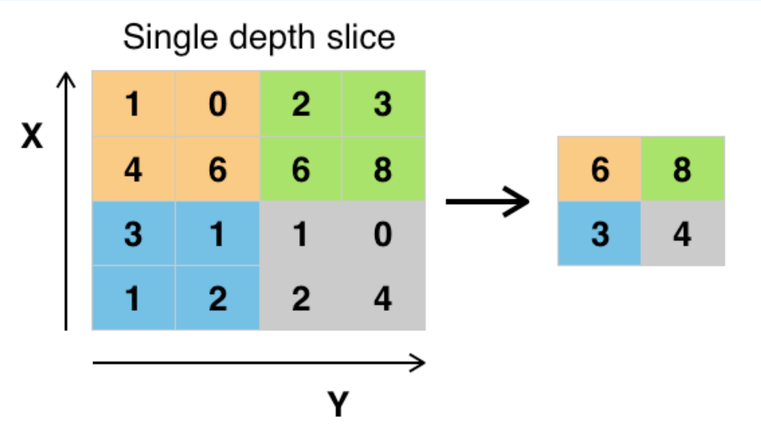
The CIFAR-10 dataset is a collection of images that are commonly used to train machine learning and computer vision algorithms. It is one of the most widely used datasets for machine learning research. The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes.

ImageNet
mit Dense Layers alleine kaum effiziente Berechnungen mehr möglich.
Braucht ConvNet / Transformers

Pooling Layer

Um die Berechenbarkeitskomplexität zu reduzieren,
werden Aktivierungen von Mustern zusammengefasst in sogenannten
"**Pooling Schichten**"

Oft ist die **exakte Position** eines Features (Kante, Tigerfell, ...) für die Klassifikation eines Objektes nicht mehr relevant.



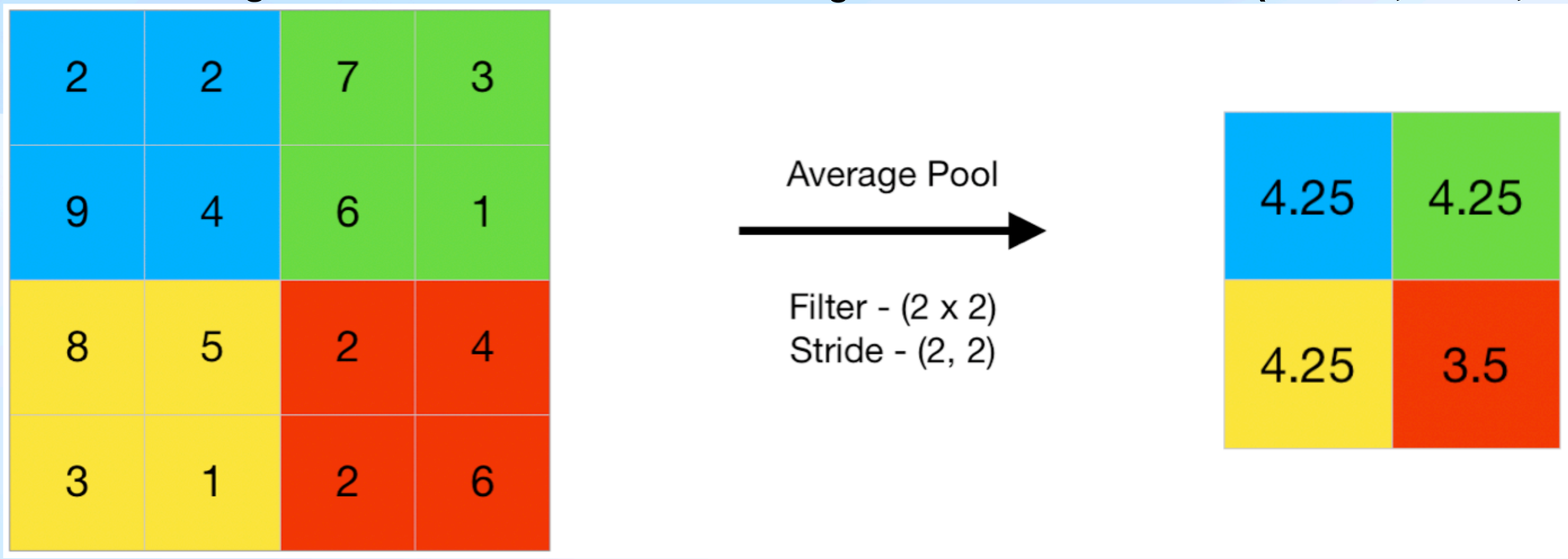
Max-Pooling: nimm die **größte Aktivierung** eines Features in einem Bereich
Argmax-Pooling: nimm **das Features** mit der größten Aktivierung eines in einem Bereich
Mean-Pooling: nimm das Mittel der Aktivierungen

Pooling Layer

Um die Berechenbarkeitskomplexität zu reduzieren,
werden Aktivierungen von Mustern zusammengefasst in sogenannten
"**Pooling Schichten**"

Oft ist die **exakte Position** eines Features (Kante, Tigerfell, ...) für die Klassifikation eines Objektes nicht mehr relevant.

Mean-Pooling: nimm das Mittel der Aktivierungen von einem **Feature (Muster, Form, Filter, kernel)**

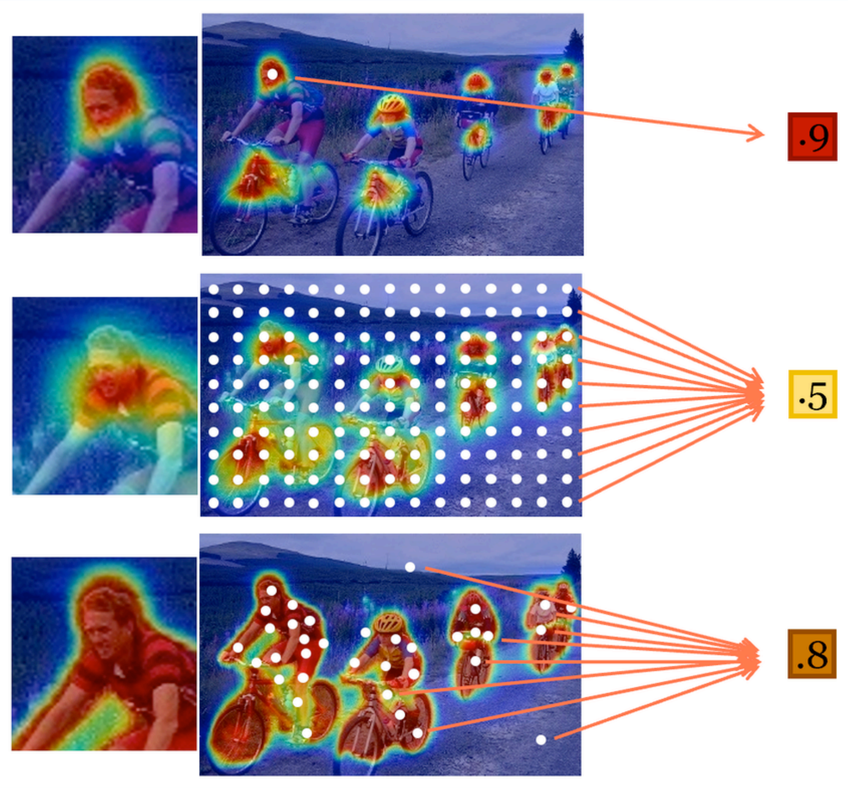


Global Average Pooling (GAP)

Global **Average** Pooling (GAP) wandelt **jedes Kanal-Feature-Map** auf eine **einzige Zahl um**

GAP reduziert eine 2D-Feature-Map direkt in einen **Skalar** pro Kanal (**durchschnittlicher Wert der gesamten Feature-Map**).

💡Das ist in der **letzten <u>Klassifizierungsschicht</u>** sinnvoll, wenn wir nur noch wissen wollen **was** ein Bild darstellt (nicht wo)!



Global Average Pooling (GAP)

- Wandelt jedes Kanal-Feature-Map auf eine einzige Zahl (durchschnittlicher Wert des gesamten Feature-Maps) um. GAP reduziert also eine 2D-Feature-Map direkt in einen Skalar pro Kanal.
- **Funktionsweise**: Statt eines festen Fensters über das Bild zu verschieben, wird
- **über das gesamte Feature-Map eines Kanals hinweg der Durchschnitt berechnet.**

- **Beispiel**:
  - Input: 7x7 Feature-Map pro Kanal
  - Global Average Pooling: Das Ergebnis ist eine einzelne Zahl pro Kanal, die den Durchschnitt aller Werte des Kanals darstellt.
- **Vorteile**:
  - Sehr kompakte Darstellung der Information; **reduziert drastisch die Anzahl der Parameter**
- **Nachteile**:
  - In Fällen, in denen die **räumliche Information** innerhalb der Feature-Maps besonders wichtig ist, kann GAP diese möglicherweise **zu stark verallgemeinern**.

Poolings Vergleich

- **Granularität**:
  - **Max/Mean-Pooling** reduziert die Dimensionen in **kleineren Schritten** (durch lokale Durchschnittswerte).
  - Global Average Pooling komprimiert ein **komplettes Feature-Map in eine einzige Zahl.**

- **Anwendungsfall**:
  - **Max/Mean-Pooling** wird oft **innerhalb** des Netzwerks verwendet, um die Dimensionen schrittweise zu reduzieren.
  - **GAP** wird in der Regel **am Ende** eines Netzwerks eingesetzt, insbesondere *bei Klassifizierungsaufgaben*.

Aspekt:
GAP kann als 'schlaues' Flatten betrachtet werden
batch*width*height*channels =>
batch*scalar*channels

Reshaping-Schichten, Flatten

Reshape: allgemeine Tensor Operation zum **ordnen von Daten** in neuen Gruppierungen, ohne Daten zu verwerfen.

- **squeeze** (**1**,3,**1**,5) => (3,5) (get rid of fake 1 'stacks') can be simulated with flatten(2,3).flatten(0,1)
- **flatten() == reshape(-1) (2,3,4) => (12) -1 = behalte die batch size bei**

Flatten-Schicht:

- Wandelt ein mehrdimensionales Feature-Map (z.B. von einem Convolution- oder Pooling Layer) in einen **eindimensionalen Vektor** um.

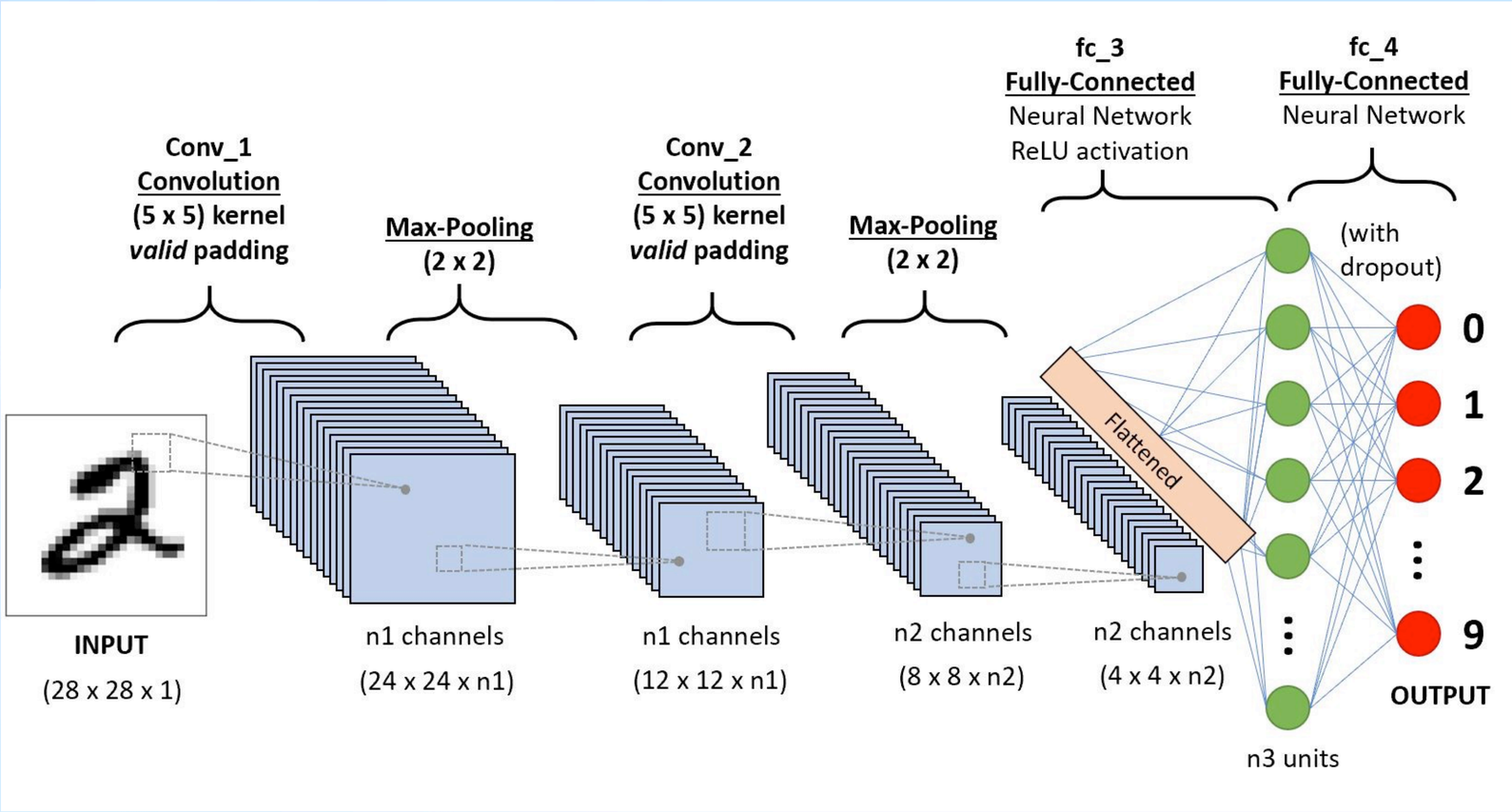
- **Kontext**: Nach mehreren **Faltungs- und Pooling**-Operationen haben die Daten meist eine **3D-Struktur (Breite x Höhe x Kanäle)**.
  - Diese Struktur ist für die vollständig verbundenen "**Dense**" Schichten ungeeignet, da diese einen Vektor als Eingabe erwarten.
  - Die **Flatten-Schicht** löst dieses Problem, indem sie die 3D-Daten in einen **1D-Vektor** umwandelt.

💡 das "**view**" welches wir gesehen haben ist **nur ein effizientes reshape**,
ohne die zugrunde liegenden Daten im Speicher wirklich neu anzuordnen (es wird nur die Sicht geändert)
Hierzu müssen die Daten aber **zusammenhängend** im Speicher abgelegt sein (kann man mit **tensor.contiguous()** erzwingen)

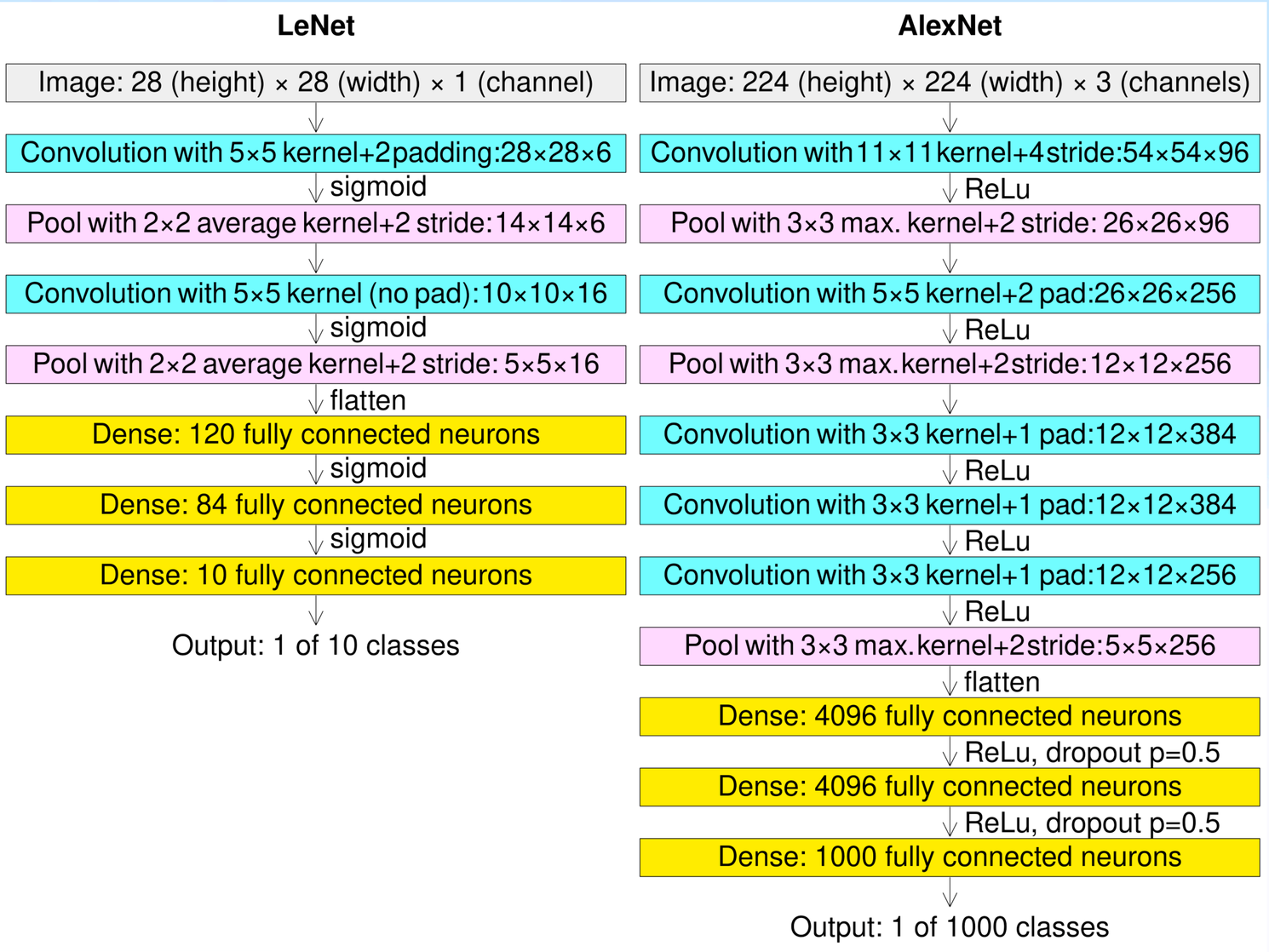
💡 **reorder / permute ≠ reshape**
transpose (per trick) **dimension swap**

Deep Convolutional Net

Im Deep Convolutional Net werden die Faltenden Schichten (oft abwechselnd mit Pooling) mehrfach gestapelt, darauf folgen eine oder mehrere Perzeptronen Schichten (oder andere Bausteine).



LeNet und AlexNet



Fragen

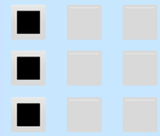
Frage: Was soll kernel size 1?

- effizientes downsampling
- bestimmte Farben selektieren
- Kanäle Reduzieren

```
model = nn.Sequential(
    nn.Conv2d(in_channels=256, out_channels=32, kernel_size=1),
    nn.ReLU(),
    nn.Conv2d(in_channels=32, out_channels=10, kernel_size=1)
)
```

Vokabeln

Filter Maske Muster Schablone Form **kernel**



`Anzahl der Filter: channels / feature size`

Feature Map Übereinstimmungen/**Aktivierung** der Filter

Convolution "abscannen eines Bildes mit Filtern"

Max-Pooling

Mean-Pooling

Aufgabe

a) Triviales ConvNet MNIST auschecken / implementieren
b) Hyperparameter Suche über die ConvNet Parameter
c) Zweite ConvNet Schicht einbauen
d) Extra: visualisiere MNIST CNN Muster! (siehe lenet_cifar_visualize.py)

Ziel: Komme über 72% bei Cifar!

Buch

Buch: