Sehen lernen: Faltung, Filter, Pooling

Kapitel online

Wie ein Netz in einem Zahlenwürfel Kanten, Ecken und schließlich Katzen findet — mit einem 3×3-Fenster, das über das Bild wandert. Mit Kernel-Werkstatt zum Selbstbauen, Pooling-Demo und Parameter-Zähler, der zeigt, warum Faltung tausendmal sparsamer ist als ein normales Netz.

Fortgeschritten · Kapitel 25 von 50 · ca. 25 Min. · 250 XP

Einfach gesagt: Für den Computer ist ein Foto nur eine riesige Zahlentabelle — ein kleines Fenster wandert darüber und findet so Kanten, dann Ecken, dann Katzen.

Dein Handy macht es beiläufig: Noch bevor du auslöst, sitzen Rahmen auf allen Gesichtern. Du tippst „Hund“ in die Galerie und bekommst Hunde. In Kapitel 430 lässt du so einen Erkenner live über dein Kamerabild laufen.

Für den Rechner ist ein Foto aber nur ein **Zahlenwürfel**: Höhe × Breite × 3 Kanäle, jede Zahl eine Helligkeit von 0 bis 255 — so, wie du es in Kapitel 020 auseinandergenommen hast. Nirgendwo darin steht „Katze“. Nirgendwo „Ohr“. Da stehen nur Zahlen.

Die Frage lautet deshalb: **Wie findet ein Netz in einem Haufen Zahlen eine Katze — egal, ob sie links oben oder rechts unten sitzt?** Die Antwort heißt **Faltung** (englisch *convolution*) und ist einfacher, als das Wort klingt: Ein kleines Fenster wandert über das Bild und rechnet an jeder Stelle *eine einzige Zahl* aus. Mehr passiert nicht.

HerrKI-Tipp: Ein Satz für dieses Kapitel — **„Neun Zahlen, die überall gelten.“** Der Rest ist Buchhaltung.

Was du hier lernst

- Warum ein **normales Netz an Bildern scheitert**
- Was eine Faltung rechnet: die **gewichtete Summe im Fenster**
- Dass die Filter **gelernt und nicht programmiert** werden
- Was **Pooling** macht — kleiner rechnen, Wesentliches behalten
- Wie aus Kanten über Ecken und Teile **Katzen** werden

Für den Unterricht

Einstieg (5 Min): Kapitel gemeinsam lesen, Alltagsbeispiel aufgreifen.

Erkunden am Gerät (15 Min): die interaktiven Module auf herrki.de/250_faltung

ausprobieren.

Sichern mit Quiz (5 Min): „Kurzer Check" besprechen, „Das nimmst du mit" festhalten.

Warum ein normales Netz an Bildern scheitert

Nimm das Netz aus Kapitel 110: eine **Dense-Schicht** („dicht“ = jede Eingabezahl mit jedem Neuron verbunden). Jedes Pixel bekommt zu jedem Neuron ein *eigenes* Gewicht. Rechnen wir das für 28 × 28 Pixel durch, grau — das Ziffernformat aus Kapitel 420:

```
// Dense-Schicht direkt auf den rohen Pixeln
Eingabe: 28 x 28 x 1 = 784 Zahlen // Graubild, flach
Neuronen: 128
Gewichte: 784 * 128 = 100 352
Bias: + 128
Parameter: 100 480 // fuer EINE Schicht
```

Hunderttausend Zahlen für eine Schicht und ein Briefmarkenbild. Ein Handyfoto hat zwölf Millionen Pixel — die Rechnung willst du nicht sehen.

Das Zahlenproblem ist dabei das kleinere. Schlimmer: Verschiebt sich die Katze um drei Pixel, landen alle Helligkeiten bei *anderen* Gewichten — für das Netz ein komplett neues Bild. Es müsste jede Position **einzeln lernen**. Kein Datensatz der Welt reicht dafür.

Wir brauchen also zwei Eigenschaften:

- **Lokal schauen.** Ein Ohr erkennt man an benachbarten Pixeln — nicht daran, was 400 Pixel weiter unten passiert.
- **Überall gleich rechnen.** Eine Kante sieht oben links aus wie unten rechts.

Genau das ist die Faltung: keine neue Mathematik, sondern eine **Einschränkung** des Dense-Layers — weniger Verbindungen, dafür wiederverwendete Gewichte.

Runter zum Tensor: Die Faltung ist eine gewichtete Summe im Fenster

Ein **Kernel** (auch **Filter**) ist eine kleine Matrix, meist 3 × 3 — also ganze **neun Zahlen**. Damit machst du Folgendes:

1. Lege den Kernel auf eine Stelle im Bild — er deckt 9 Pixel ab.
2. Multipliziere jede Kernel-Zahl mit dem Pixel darunter.
3. Addiere die neun Produkte — das ist **eine** Ausgabezahl.
4. Schiebe das Fenster ein Pixel weiter, und wieder. Über das ganze Bild.

Schritt 2 und 3 kennst du längst: das **Skalarprodukt** aus Kapitel 060, nur auf einem 3 × 3-Ausschnitt. „Faltung“ heißt also: *viele kleine Skalarprodukte, systematisch über das Bild geschoben.*

Einmal von Hand nachgerechnet

Wir vereinfachen die Grauwerte zum Mitrechnen auf **0 und 1** (im echten Bild sind es 0–255). Links dunkel, rechts hell — mitten durch läuft eine senkrechte Kante:

```
// Bildausschnitt 5 x 5 Kernel "Kante senkrecht"
  0 0 1 1 1 -1 0 1
  0 0 1 1 1 -1 0 1
  0 0 1 1 1 -1 0 1
  0 0 1 1 1
  0 0 1 1 1 // links minus rechts

// Fenster auf die erste Position (Mitte = Zeile 1, Spalte 1):
  0*(-1) + 0*0 + 1*1
+ 0*(-1) + 0*0 + 1*1
+ 0*(-1) + 0*0 + 1*1 = 3

// an allen 3 x 3 gueltigen Positionen -> die Feature Map:
  3 3 0
  3 3 0
  3 3 0
```

Lies das Ergebnis in Ruhe: Wo die Kante im Fenster liegt, steht eine **3**; wo alles gleich hell ist, eine **0**. Der Kernel hat keine Ahnung, was ein Bild ist — er hat links minus rechts gerechnet und trotzdem die Kante gefunden. Die 3 taucht zweimal auf, weil ein drei Pixel breites Fenster die Kante aus zwei Positionen berührt; und die Ausgabe ist **kleiner** als die Eingabe — dazu gleich mehr.

Das Ergebnis-Bild heißt **Feature Map** („Merkmalskarte“). Sie sagt für jede Stelle: *Wie stark ist hier das Muster, nach dem dieser Kernel sucht?* Hohe Zahl = gefunden, null = nichts, negativ = dasselbe Muster andersherum.

Der eigentliche Trick: Gewichts-Teilung

Es sind **immer dieselben neun Zahlen**, die über das Bild wandern — **Gewichts-Teilung** (*weight sharing*). Das löst beide Probleme auf einen Schlag: neun Gewichte statt hunderttausend, egal wie groß das Bild ist. Und **die Katze darf sitzen, wo sie will** — wandert sie nach rechts, wandert ihre Antwort einfach mit.

Und bei Farbbildern?

Bei drei Kanälen (R, G, B) ist der Kernel eben **3 × 3 × 3**: ein Fenster je Farbkanal, 27 Zahlen, alle Produkte addiert — heraus kommt trotzdem **eine** Zahl pro Position. Und weil ein Filter allein einseitig ist, hat eine Faltungsschicht **viele** Kernel nebeneinander; alle Feature Maps werden gestapelt: `[Höhe, Breite, Anzahl Filter]`.

Padding, Stride und die Ausgabegröße

- **Padding** („Auffüllen“): Am Rand ragt das Fenster ins Nichts, deshalb wird die Ausgabe pro Schicht 2 Pixel kleiner — nach zehn Schichten bleibt wenig übrig. Ein Rand aus Nullen hält die Größe.
- **Stride** („Schrittweite“): Stride 1 = jedes Pixel, Stride 2 = jedes zweite, also halbe Ausgabegröße und drei Viertel Rechenarbeit gespart.

```
// Ausgabegroesse pro Achse (abgerundet)
Ausgabe = (Eingabe + 2*Padding - Kernel) / Stride + 1

// 150er-Bild, 3x3-Kernel:
Padding 0, Stride 1: (150 + 0 - 3) / 1 + 1 = 148
Padding 1, Stride 1: (150 + 2 - 3) / 1 + 1 = 150
Padding 0, Stride 2: (150 + 0 - 3) / 2 + 1 = 74
```

Am Gerät ausprobieren: herrki.de/250_faltung#abschnitt-4

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Filter, die niemand programmiert hat

Bis hier hast du dir die neun Zahlen selbst ausgedacht. In einem echten **CNN** (Convolutional Neural Network) macht das *niemand*: Die Kernel-Zahlen sind **Gewichte wie alle anderen** — zufällig gestartet, dann vom Gradient Descent (118) so verschoben, dass der Fehler sinkt.

Niemand sagt dem Netz „such mal Kanten“. Es findet selbst heraus, dass Kanten- und Farbfilter in der ersten Schicht am nützlichsten sind. **Zeiler und Fergus** haben 2014 sichtbar gemacht, was in einem trainierten Netz steht: erst Kanten und Farbflecken, dann Ecken und Texturen, dann Teile, dann ganze Objekte. Hineingeschrieben hatte das niemand.

```
// Faltungsschicht, 8 Filter a 3x3, Graubild
Gewichte: 3 * 3 * 1 * 8 = 72 // hoehe * breite * Eingangskanaele * Filter
Bias: + 8 // einer je Filter
Parameter: 80

// Dense-Schicht von oben: 100 480 -> Faktor 1 256
```

80 gegen 100 480. Und die 80 funktionieren zusätzlich auf jedem Bildformat, während die 100 480 exakt an 28 × 28 kleben.

Nach der Faltung kommt ReLU. Eine Faltung ist linear — multiplizieren und addieren. Und wie du in Kapitel 145 gesehen hast, falten sich zehn lineare Schichten zu *einer einzigen* zusammen. Deshalb steht hinter jeder Faltung `ReLU`: negative Werte auf 0, positive bleiben. Erst dadurch lohnt sich Schicht zwei.

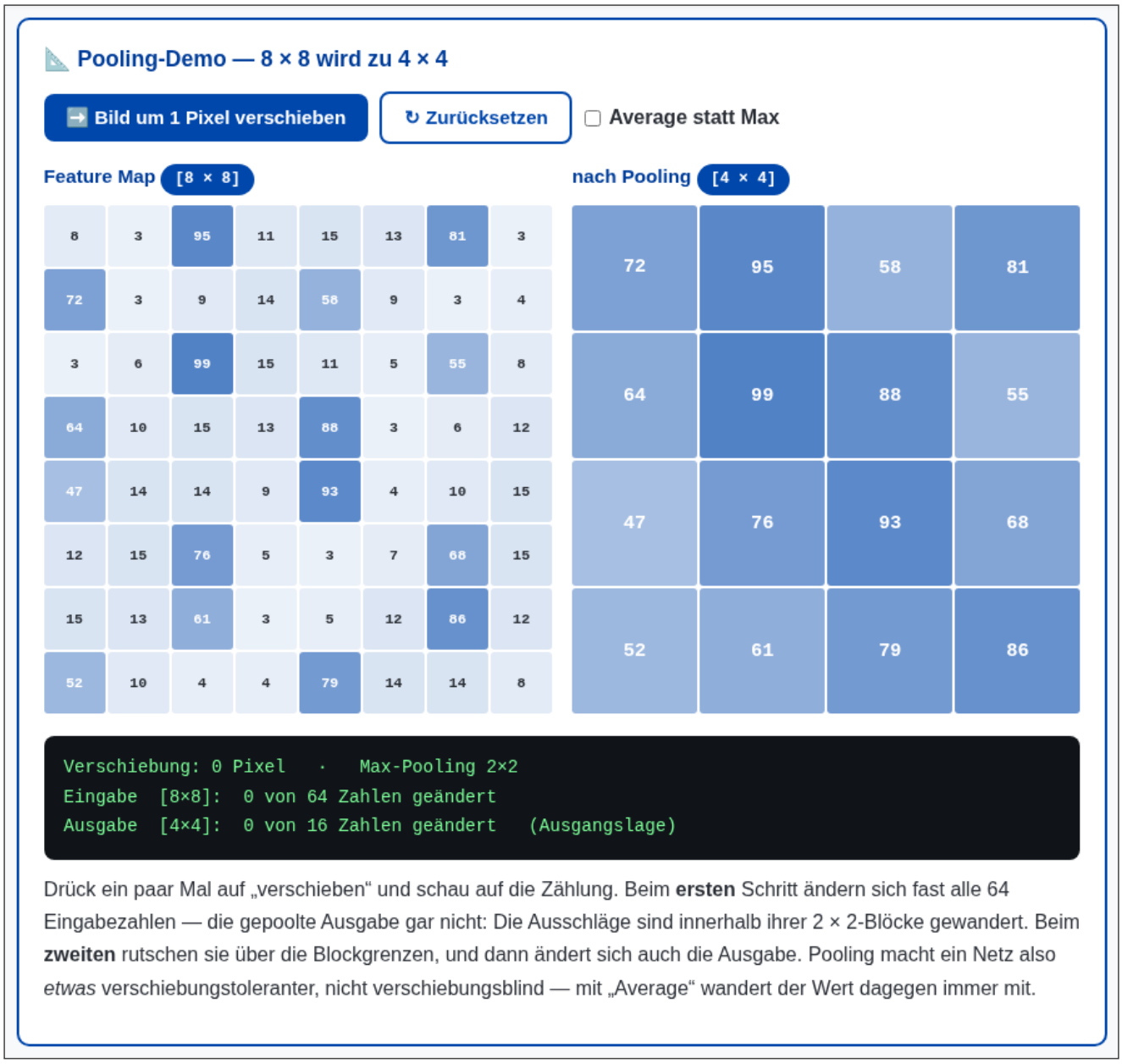
Pooling: kleiner machen, Wesentliches behalten

Nach ein paar Faltungen liegt ein Stapel Feature Maps in voller Bildgröße herum — viel zu viele Zahlen. **Max-Pooling 2 × 2** ist die einfachste Zusammenfassung: Nimm jeden 2 × 2-Block, behalte den **größten** Wert. Halbe Höhe, halbe Breite, **ein Viertel der Zahlen** — mit zwei Wirkungen:

- **Weniger Rechnen.** Die nächste Schicht arbeitet auf einem Viertel der Daten.
- **Kleine Verschiebungen verschwinden.** Wo im Block der stärkste Ausschlag sitzt, ist dem Maximum egal. Es zählt, *dass* ein Merkmal da ist, nicht *wo genau*.

Am Gerät ausprobieren: herrki.de/250_faltung#abschnitt-6

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Der Stapel: Von Kanten zu Katzen

Einzeln kann eine Faltung nur Kanten. Die Katze entsteht erst dadurch, dass der Block *Faltung → ReLU → Pooling* mehrfach läuft. So sieht ein vollständiges kleines CNN für Ziffern aus:

Bild
↓
[28, 28, 1]

→

Conv 8×3×3 + ReLU
↓
[28, 28, 8]
80

→

Max-Pool 2×2
↓
[14, 14, 8]
0

→

Conv 16×3×3 + ReLU
↓
[14, 14, 16]
1 168

→

Max-Pool 2×2
↓
[7, 7, 16]
0

→

Flatten
↓
[784]

→

Dense + Softmax
↓
[10]

- Die zweite Faltung hat **3 × 3 × 8 × 16 + 16 = 1 168 Parameter**. Die 8, weil die Eingabe acht Kanäle hat — jeder Kernel greift durch *alle* Feature Maps der Vorschicht und lernt so Kombinationen: „senkrechte UND waagerechte Kante an derselben Stelle“ = **Ecke**.
- **Pooling hat null Parameter.** Reine Buchhaltung, da wird nichts gelernt.
- Nach dem zweiten Pooling sind es 7 × 7 × 16 = **784 Zahlen** — zufällig so viele wie das Bild am Anfang, nur keine Helligkeiten mehr, sondern Merkmalsstärken. Damit kann eine Dense-Schicht plus Softmax (230) etwas anfangen.

Das rezeptive Feld: Warum Tiefe Bedeutung schafft

Ein Wert der ersten Feature Map hat 3 × 3 Pixel gesehen. Ein Wert der zweiten sieht 3 × 3 *Werte der ersten* — die je 3 × 3 Pixel gesehen haben; das Pooling verdoppelt den Ausschnitt nochmals. Dieser Ausschnitt heißt **rezeptives Feld**:

Schicht	Sieht ungefähr	Findet typischerweise
1. Faltung	3 × 3 Pixel	Kanten, Farbübergänge
2. Faltung	ein Dutzend Pixel	Ecken, Texturen
3.–4. Faltung	ein Bildviertel	Teile: Auge, Ohr, Rad
letzte Schichten	fast das ganze Bild	Objekte: Katze, Auto, Ziffer 7

Das ist die Antwort auf die Eingangsfrage: **Aus Kanten werden Ecken, aus Ecken Teile, aus Teilen Katzen** — nicht weil jemand einen Katzen-Detektor programmiert hat, sondern weil sich einfache Muster schichtweise zu komplizierten zusammensetzen.

Am Gerät ausprobieren: herrki.de/250_faltung#abschnitt-7

Direkt zum Modul

Parameter-Zähler — wie sparsam ist Faltung wirklich?

Bildgröße: 28

Filter (Conv): 8

Neuronen (Dense): 128

Dense auf Pixeln 100 480

Faltungsschicht 80

```
Dense:  28 × 28 × 128 + 128  =  100 480 Parameter
Conv:   3 × 3 × 1 × 8 + 8  =  80 Parameter
→ Die Faltungsschicht ist 1 256-mal sparsamer – und bleibt es auch, wenn du das Bild größer machst.
```

Schieb die Bildgröße ganz nach rechts: Der Dense-Layer explodiert quadratisch mit der Bildfläche — die Faltungsschicht rührt sich **gar nicht**, weil ihre Gewichte von der Bildgröße unabhängig sind. Genau deshalb wurde Bild-KI praktikabel.

Interaktives Modul – am Gerät ausprobieren

Zurück zur Anwendung

- 420 — Ziffern erkennen: Dort trainierst du genau dieses CNN im Browser.
- 430 — Objekterkennung: ein CNN, das zusätzlich **Kästen** vorhersagt — vier Zahlen pro Objekt für Position und Größe.
- 440 — Körper als Tensor: Pro Körperpunkt liefert das Netz eine Wärmekarte — eine Feature Map, deren hellste Stelle das Handgelenk ist.
- 460 — Eigene Bild-KI: **Transfer Learning** — die frühen Filter passen auf *jedes* Bildproblem; du wirfst nur das Ende eines fertigen Netzes weg und trainierst mit ein paar Dutzend eigenen Fotos neu.
- 480 — Style Transfer: „Inhalt“ steckt in den tiefen Aktivierungen, „Stil“ in den Zusammenhängen der frühen Kanäle.
- 380 — KI-Agenten: Eine Drohne rechnet auf ihrem Chip genau so ein Netz — klein genug für ein paar Watt.

⚠ **Ehrlich bleiben: Faltung ist nicht mehr das einzige Rezept.** Seit 2020 gibt es den **Vision Transformer** (ViT, Dosovitskiy u. a.): Er zerschneidet das Bild in Kacheln, behandelt jede wie ein Token und lässt Attention (320) darüber laufen — ohne Annahme über Nachbarschaft. Mit sehr vielen Trainingsbildern schlägt das CNNs; bei *wenigen* Daten und auf kleinen Geräten liegen CNNs vorn, weil die Faltung „Nachbarpixel gehören zusammen“ schon eingebaut hat.

Was heißt das für dich?

- **Bild-KI ist kein anderes Fach.** Dasselbe Netz wie in Kapitel 110, nur mit mehrfach benutzten Gewichten. „Computer Vision“ ist keine Zauberdisziplin, sondern gewichtete Summen im Fenster.
- **Du kannst Modellgrößen einschätzen.** „5 Millionen Parameter“ sagt dir jetzt etwas — und dass die meisten am Ende in den Dense-Schichten sitzen.
- **Du weißt, warum Trainingsdaten vielfältig sein müssen.** Faltung schenkt dir Verschiebung. Drehung, Beleuchtung und Maßstab *nicht* — die musst du in die Daten bringen (siehe Kapitel 240).
- **Du erkennst Grenzen.** Ein CNN sieht Texturen und lokale Muster gut, hat aber kein Konzept von „das ist ein Tier, also hat es vier Beine“ — ungewöhnliche Perspektiven oder gestörte Pixel bringen es leicht aus dem Tritt.

Zum Weiterlesen

- LeCun u. a. 1998, *„Gradient-Based Learning Applied to Document Recognition“* (LeNet) — PDF (yann.lecun.com/exdb/publis/pdf/lecun-98.pdf)
- Krizhevsky u. a. 2012, *„ImageNet Classification with Deep CNNs“* (AlexNet) — papers.nips.cc (papers.nips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html)
- Zeiler & Fergus 2014, *„Visualizing and Understanding Convolutional Networks“* — arxiv.org/abs/1311.2901
- Dosovitskiy u. a. 2020, *„An Image is Worth 16x16 Words“* (ViT) — arxiv.org/abs/2010.11929
- Stanford CS231n, *Convolutional Networks* — cs231n.github.io (cs231n.github.io/convolutional-networks)

Kurzer Check – mit Lösung

Lösungen für Lehrkräfte – für Lernende die Online-Version nutzen.

Frage 1: Was rechnet eine Faltung an einer Bildposition?

- Den Mittelwert des ganzen Bildes.
- ✔ **Die gewichtete Summe des 3 × 3-Fensters mit den neun Kernel-Zahlen — eine einzige Ausgabezahl.**
- Eine Multiplikation jedes Pixels mit seinem eigenen Gewicht.

Neun Produkte, addiert — ein Skalarprodukt im Fenster. Die dritte Antwort beschreibt die Dense-Schicht, also das, was die Faltung ersetzt.

Frage 2: Warum braucht eine Faltungsschicht so viel weniger Parameter als eine Dense-Schicht?

- Weil die Bilder vorher komprimiert werden.
- Weil nur jedes zweite Pixel angeschaut wird.
- ✔ **Weil dieselben Kernel-Gewichte an jeder Position wiederverwendet werden.**

Gewichts-Teilung: Neun Zahlen gelten für das ganze Bild — 80 statt 100 480 Parameter, und derselbe Filter passt auch auf größere Bilder.

Frage 3: Was macht Max-Pooling?

- ✔ **Aus jedem 2 × 2-Block bleibt der größte Wert — das Bild wird kleiner und unempfindlicher gegen kleine Verschiebungen.**
- Es verdoppelt die Auflösung der Feature Map.
- Es entfernt alle Werte unter null.

Halbe Höhe, halbe Breite, ein Viertel der Zahlen, null gelernte Parameter. „Werte unter null entfernen“ ist ReLU (Kapitel 145) — wird gern verwechselt.

Das nimmst du mit

- **Faltung ist eine gewichtete Summe im Fenster** — ein Skalarprodukt, über das Bild geschoben.
- **Gewichts-Teilung ist der Zauber.** Dieselben neun Zahlen überall: 80 statt 100 480 Parameter, und die Katze darf sitzen, wo sie will.
- **Die Filter werden gelernt.** Zufälliger Start, Gradient Descent macht Kanten- und Texturdetektoren daraus.
- **Pooling fasst zusammen.** Max-Pooling 2 × 2 viertelt die Zahlen, kostet keine Parameter, macht das Netz verschiebungstoleranter.
- **Tiefe schafft Bedeutung.** Kanten → Ecken → Teile → Objekte.

Weiter geht’s: Bisher hat das Netz gelernt, was WIR ihm sagen — jedes Trainingsbild hatte ein Etikett. Im nächsten Kapitel lernt es ganz ohne Etiketten — und erfindet dabei einen Raum.

So zitierst du dieses Kapitel

HerrKI: Sehen lernen: Faltung, Filter, Pooling. herrki.de/250_faltung, ein privates Hobbyprojekt von Terragon.de, abgerufen am 09.09.2026.