Male eine Ziffer — dein erstes CNN

Trainiere in deinem Browser ein eigenes Convolutional Neural Network auf 65.000 echten Handschrift-Ziffern — und lass es danach die Ziffern erkennen, die du selbst malst.

Kapitel online

Interaktiv · Kapitel 44 von 50 · ca. 15 Min. · 300 XP

Einfach gesagt: Du trainierst in deinem Browser ein Netz mit tausenden fremden Handschriften — dann malst du selbst eine Ziffer, und es rät, welche es ist.

Jede Paket-Sortieranlage der Welt liest Postleitzahlen. Handgeschrieben. Schief. Mal Kugelschreiber, mal Filzstift, mal quer über die Klebekante. Zehntausende Sendungen pro Stunde, und irgendwo dazwischen deine krakelige 7. Dass das funktioniert, ist keine Magie — es ist ein kleines Netz aus ein paar tausend Zahlen. Und genau so eines baust du dir heute selbst: in diesem Browser-Tab, mit ungefähr einer Minute Rechenzeit.

Übungsmaterial brauchst du dafür — und da gibt es *den* Klassiker: **MNIST**. Ende der 1980er kritzelten Mitarbeiter des US-Volkszählungsamts und Highschool-Schüler Zehntausende Ziffern auf Formulare. Die wurden ausgeschnitten, auf 28×28 Pixel normiert, mittig ausgerichtet — und sind seit 1998 das „Hallo Welt“ der Bilderkennung: Jedes neue Verfahren muss zuerst hier antreten. **65.000 dieser Bilder liegen für dich bereit**, echte fremde Handschriften, kein einziges davon erfunden.

Roter Faden: Anwendung → runter bis auf die einzelne Zahl → wieder hoch. Am Ende hältst du beides in der Hand: die gemalte Ziffer und die zehn Zahlen, die das Netz darüber sagt.

Was du hier lernst

- Wie aus einem Bild ein **Tensor** wird — du siehst die Zahlen selbst
- Was ein **Conv-Filter** abtastet und warum er einer Dense-Schicht bei Bildern überlegen ist
- Die komplette Form-Reise durchs Netz: [28,28,1] → … → [10]
- Ein echtes Training mit Loss und Genauigkeit, live mitzulesen
- Deine eigene Handschrift gegen dein eigenes Netz — samt der Fälle, in denen es danebenliegt

Für den Unterricht

Einstieg (3 Min): Kapitel gemeinsam lesen, Alltagsbeispiel aufgreifen.

Erkunden am Gerät (9 Min): die interaktiven Module auf herrki.de/420_ziffern_erkennen ausprobieren.

Sichern mit Quiz (3 Min): „Kurzer Check" besprechen, „Das nimmst du mit" festhalten.

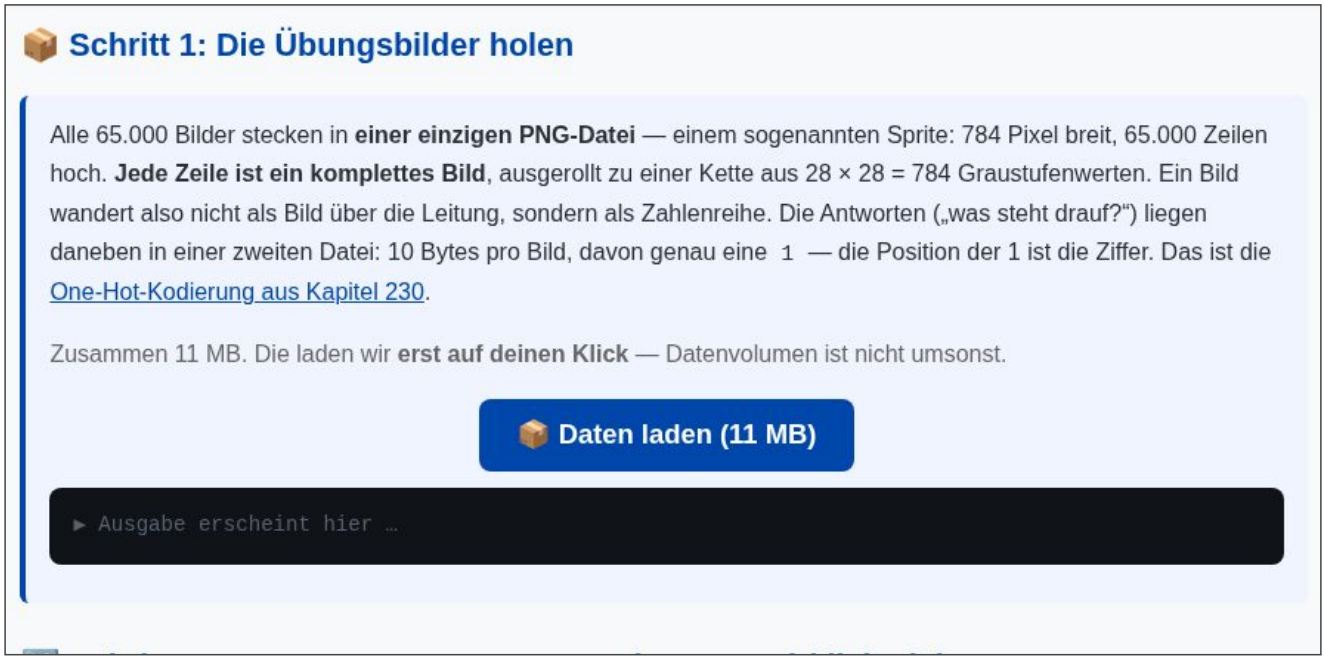
Schritt 1: Die Übungsbilder holen

Alle 65.000 Bilder stecken in **einer einzigen PNG-Datei** — einem sogenannten Sprite: 784 Pixel breit, 65.000 Zeilen hoch. **Jede Zeile ist ein komplettes Bild**, ausgerollt zu einer Kette aus 28 × 28 = 784 Graustufenwerten. Ein Bild wandert also nicht als Bild über die Leitung, sondern als Zahlenreihe. Die Antworten („was steht drauf?“) liegen daneben in einer zweiten Datei: 10 Bytes pro Bild, davon genau eine `1` — die Position der 1 ist die Ziffer. Das ist die One-Hot-Kodierung aus Kapitel 230.

Zusammen 11 MB. Die laden wir **erst auf deinen Klick** — Datenvolumen ist nicht umsonst.

Am Gerät ausprobieren: herrki.de/420_ziffern_erkennen#abschnitt-2

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Zwölf zufällige Bilder aus dem Stapel

Frisch aus dem Sprite gezogen, hochskaliert (deshalb die groben Pixel) — darunter steht jeweils, welche Ziffer laut Datensatz gemeint ist. Schau dir an, wie unterschiedlich Menschen eine 4 oder eine 9 schreiben. Genau diese Streuung muss dein Netz gleich verkraften.

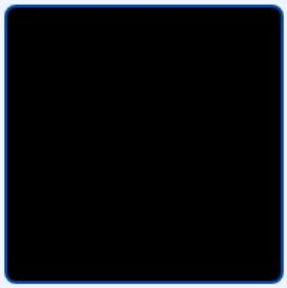
Schritt 2: Runter zum Tensor — was das Netz wirklich sieht

Für das Netz ist ein Bild keine Zeichnung, sondern eine **Tabelle**: 28 Zeilen, 28 Spalten, in jeder Zelle eine Zahl zwischen `0.00` (schwarz) und `1.00` (weiß). Mehr ist da nicht. Links siehst du ein Beispiel als Bild, rechts einen 10×10-Ausschnitt aus seiner Mitte als das, was es in Wahrheit ist — **jede Zeile rechts ist genau eine Zeile aus dem Bild links**. Wenn du die Augen ein wenig zusammenkneifst, erkennst du die Ziffer im Zahlenrelief.

Am Gerät ausprobieren: herrki.de/420_ziffern_erkennen#abschnitt-3

Direkt zum Modul

Für das Netz ist ein Bild keine Zeichnung, sondern eine **Tabelle**: 28 Zeilen, 28 Spalten, in jeder Zelle eine Zahl zwischen `0.00` (schwarz) und `1.00` (weiß). Mehr ist da nicht. Links siehst du ein Beispiel als Bild, rechts einen 10×10-Ausschnitt aus seiner Mitte als das, was es in Wahrheit ist — **jede Zeile rechts ist genau eine Zeile aus dem Bild links**. Wenn du die Augen ein wenig zusammenkneifst, erkennst du die Ziffer im Zahlenrelief.



Noch keine Daten geladen — drück oben den Knopf.

`[28, 28, 1]`

Zeilen 10–19, Spalten 10–19 — `0.00` = schwarz, `1.00` = weiß:

```
Erscheint hier, sobald die Daten geladen sind.
```

Und jetzt die Reise dieser Tabelle durch das Netz, das du gleich baust. Lies sie von oben nach unten — rechts steht immer, welche **Form** (Shape) der Tensor nach dem Schritt hat:

```
Eingabe                 [28, 28,  1]   ein Graustufenbild
 Conv2D   8 Filter 5x5 → [24, 24,  8]   8 Merkmalskarten
 MaxPool  2x2          → [12, 12,  8]   halbe Auflösung
 Conv2D  16 Filter 5x5 → [ 8,  8, 16]   Muster aus Mustern
 MaxPool  2x2          → [ 4,  4, 16]
 Flatten               → [256]          alles in eine Reihe
 Dense + Softmax       → [10]           zehn Wahrscheinlichkeiten
```

Von `[28,28,1]` — 784 Zahlen — bis `[10]` — zehn Zahlen. Zwei Dinge fallen auf: Die *Fläche* schrumpft (28 → 24 → 12 → 8 → 4), die *Tiefe* wächst (1 → 8 → 16). Das Netz gibt Ortsauflösung auf und kauft dafür Bedeutung ein. Dass aus 28 nach dem ersten Conv nur 24 wird, ist übrigens kein Trick: Eine 5×5-Lupe passt am Rand nicht mehr vollständig aufs Bild, also fallen links und rechts je 2 Pixel weg.

🔍 Was tastet so ein Filter eigentlich ab?

Stell dir eine **5×5-Lupe** vor, die Pixel für Pixel über das ganze Bild wandert und an jeder Stelle eine einzige Frage stellt: „Passt hier mein Muster?“ Der eine Filter schlägt bei senkrechten Kanten an, der nächste bei einer Rundung oben links, der dritte bei einem Strich-Ende. **Diese Muster gibt niemand vor — sie entstehen im Training**, so wie die Gewichte in „Was ist ein Neuronales Netz?“. Der entscheidende Kniff: Es ist überall *dieselbe* Lupe. Ein Muster einmal gelernt, im ganzen Bild erkannt — egal ob die Kante links oben oder rechts unten sitzt.

Das ist der ganze Unterschied zu einer normalen Dense-Schicht: Die bekäme 784 lose Zahlen serviert, ohne jede Ahnung, dass Pixel 100 und Pixel 101 *nebeneinander* liegen. Ein Conv-Filter weiß das per Bauart. (Wie so ein Bild-Tensor überhaupt aufgebaut ist, steht ausführlich in „Bilder als Tensoren“.)

Interaktives Modul – am Gerät ausprobieren

Noch keine Daten geladen — drück oben den Knopf.

[28, 28, 1]

Zeilen 10–19, Spalten 10–19 — `0.00` = schwarz, `1.00` = weiß:

```
Erscheint hier, sobald die Daten geladen sind.
```

Und jetzt die Reise dieser Tabelle durch das Netz, das du gleich baust. Lies sie von oben nach unten — rechts steht immer, welche **Form** (Shape) der Tensor nach dem Schritt hat:

```
Eingabe [28, 28, 1] ein Graustufenbild
 Conv2D 8 Filter 5x5 → [24, 24, 8] 8 Merkmalskarten
 MaxPool 2x2 → [12, 12, 8] halbe Auflösung
 Conv2D 16 Filter 5x5 → [ 8, 8, 16] Muster aus Mustern
 MaxPool 2x2 → [ 4, 4, 16]
 Flatten → [256] alles in eine Reihe
 Dense + Softmax → [10] zehn Wahrscheinlichkeiten
```

Von [28,28,1] — 784 Zahlen — bis [10] — zehn Zahlen. Zwei Dinge fallen auf: Die *Fläche* schrumpft (28 → 24 → 12 → 8 → 4), die *Tiefe* wächst (1 → 8 → 16). Das Netz gibt Ortsauflösung auf und kauft dafür Bedeutung ein. Dass aus 28 nach dem ersten Conv nur 24 wird, ist übrigens kein Trick: Eine 5×5-Lupe passt am Rand nicht mehr vollständig aufs Bild, also fallen links und rechts je 2 Pixel weg.

Was tastet so ein Filter eigentlich ab?

Stell dir eine **5×5-Lupe** vor, die Pixel für Pixel über das ganze Bild wandert und an jeder Stelle eine einzige Frage stellt: „Passt hier mein Muster?“ Der eine Filter schlägt bei senkrechten Kanten an, der nächste bei einer Rundung oben links, der dritte bei einem Strich-Ende. **Diese Muster gibt niemand vor — sie entstehen im Training**, so wie die Gewichte in „Was ist ein Neuronales Netz?“. Der entscheidende Kniff: Es ist überall *dieselbe* Lupe. Ein Muster einmal gelernt, im ganzen Bild erkannt — egal ob die Kante links oben oder rechts unten sitzt.

Das ist der ganze Unterschied zu einer normalen Dense-Schicht: Die bekäme 784 lose Zahlen serviert, ohne jede Ahnung, dass Pixel 100 und Pixel 101 *nebeneinander* liegen. Ein Conv-Filter weiß das per Bauart. (Wie so ein Bild-Tensor überhaupt aufgebaut ist, steht

ausführlich in „Bilder als Tensoren“.)

Schritt 3: Trainieren

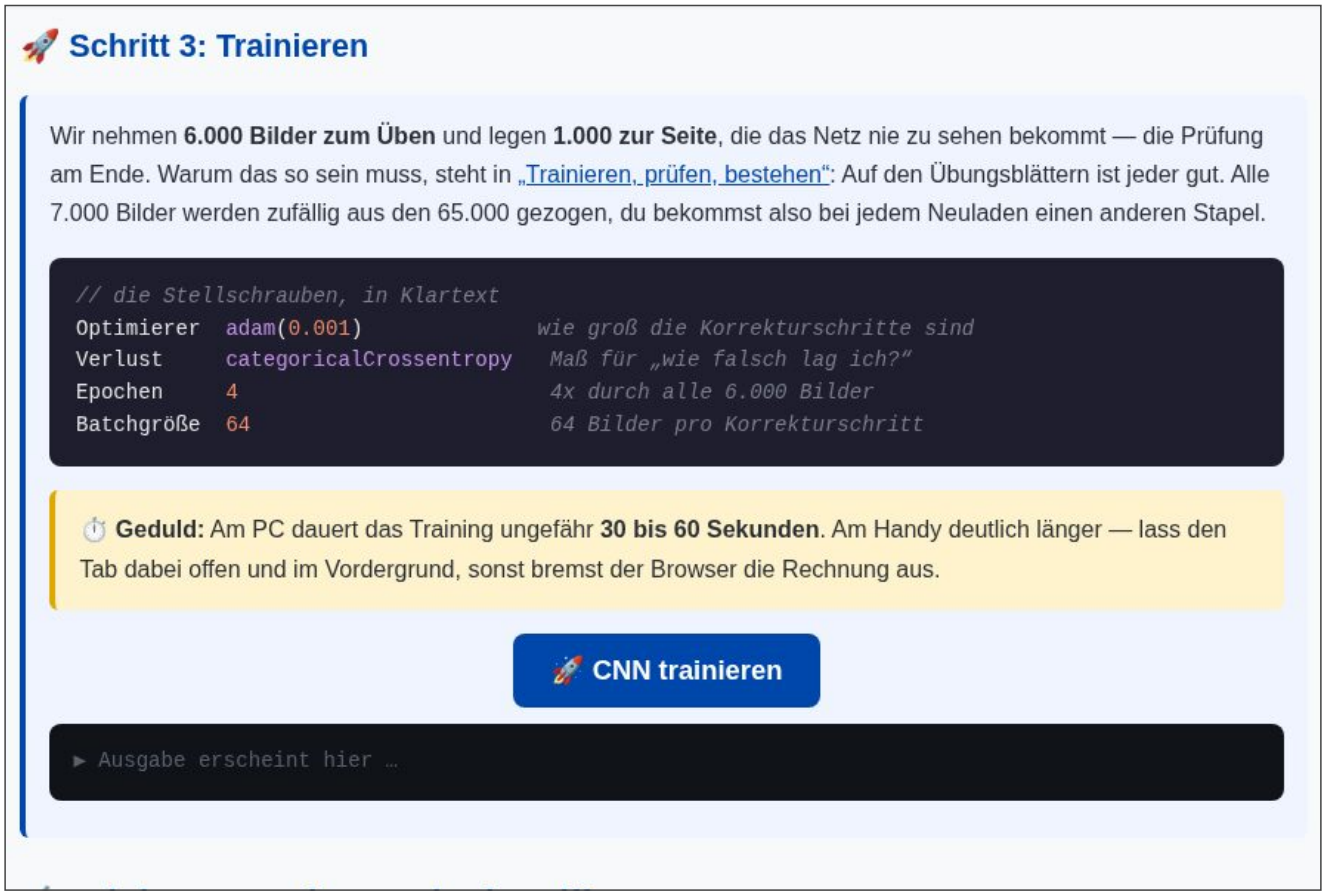
Wir nehmen **6.000 Bilder zum Üben** und legen **1.000 zur Seite**, die das Netz nie zu sehen bekommt — die Prüfung am Ende. Warum das so sein muss, steht in „Trainieren, prüfen, bestehen“: Auf den Übungsblättern ist jeder gut. Alle 7.000 Bilder werden zufällig aus den 65.000 gezogen, du bekommst also bei jedem Neuladen einen anderen Stapel.

```
// die Stellschrauben, in Klartext
Optimierer adam(0.001) wie groß die Korrekturschritte sind
Verlust categoricalCrossentropy Maß für „wie falsch lag ich?“
Epochen 4 4x durch alle 6.000 Bilder
Batchgröße 64 64 Bilder pro Korrekturschritt
```

Geduld: Am PC dauert das Training ungefähr **30 bis 60 Sekunden**. Am Handy deutlich länger — lass den Tab dabei offen und im Vordergrund, sonst bremst der Browser die Rechnung aus.

Am Gerät ausprobieren: herrki.de/420_ziffern_erkennen#abschnitt-4

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Die Prüfung: 1.000 nie gesehene Bilder

–

Acht davon im Einzelnen — ehrlich, inklusive der Patzer

Oben die Vorhersage deines Netzes, darunter die richtige Antwort. **Grün** = getroffen, **rot** = danebengelegen. Bei rund 97 % Genauigkeit ist im Schnitt jedes 33. Bild ein Fehler — mit etwas Pech siehst du hier also einen.

Schritt 4: Jetzt du — male eine Ziffer

Am Gerät ausprobieren: herrki.de/420_ziffern_erkennen#abschnitt-5

Direkt zum Modul

Male mit der Maus oder dem Finger eine Ziffer von 0 bis 9 — **groß und mittig**, so wie die Trainingsbilder aussehen. Sobald du den Strich absetzt, rechnet dein Netz. Der Weg deiner Zeichnung: 280×280 Pixel → auf **28×28 heruntergemittelt** (je 10×10 Pixel werden zu einem) → als Tensor `[1, 28, 28, 1]` ins Netz → raus kommen zehn Zahlen.



🧽 Löschen | 🔎 Erkennen

Erst trainieren — dann malen.

Das Netz sagt:

?

0	0.0 %
1	0.0 %
2	0.0 %
3	0.0 %
4	0.0 %
5	0.0 %
6	0.0 %
7	0.0 %
8	0.0 %
9	0.0 %

🔬 Und das ist ALLES, was das Netz sagt — zehn Zahlen

Kein Wort, kein Bild, keine Erklärung: der rohe Ausgabe-Tensor der Softmax-Schicht. Die zehn Werte addieren sich immer zu 1,00 — die Position des größten ist die erkannte Ziffer. Alles, was du oben an Balken und Prozenten siehst, ist nur eine hübschere Darstellung dieser Zeile.

```
Ausgabe-Tensor, Shape [1, 10]

Noch nichts gemalt.
```

Interaktives Modul – am Gerät ausprobieren

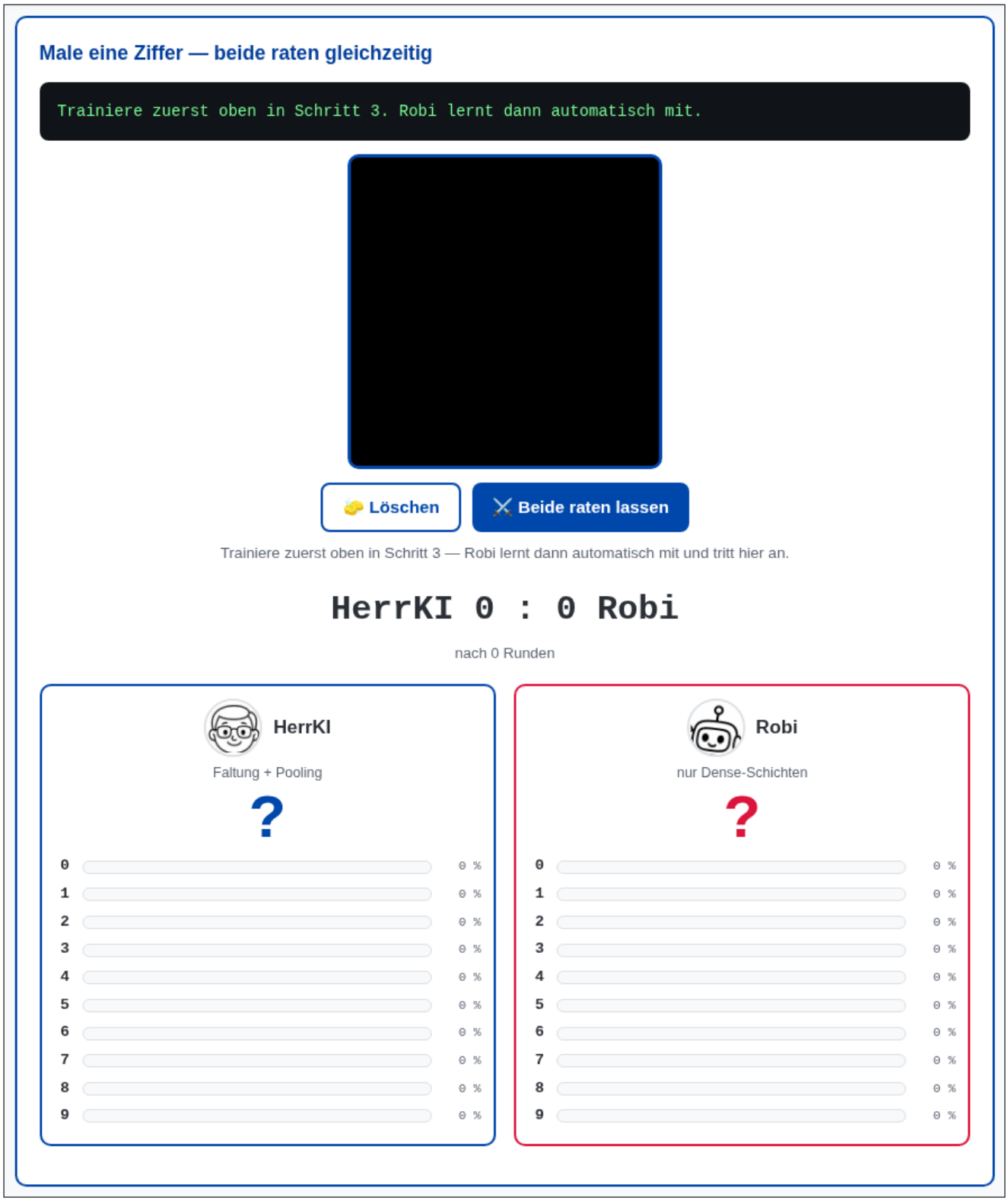
Netz-Wettkampf: HerrKI gegen Robi — wer erkennt deine Ziffer?

Dein Netz von eben hat einen Herausforderer bekommen. Während oben trainiert wurde, hat auf denselben 6.000 Bildern ein zweites Netz mitgelernt: **Robi**, der kleine Roboter. Er ist absichtlich schlichter gebaut — **keine Faltung**, kein Pooling, nur zwei vollverbundene Schichten (784 → 32 → 10). Das Bild kommt bei ihm als flache Reihe von 784 losen Zahlen an; welche Zahl neben welcher lag, weiß er nicht.

Der Witz daran: **Robi hat viel mehr Gewichte als HerrKI** — und trifft trotzdem seltener. Genau das ist der Punkt dieses Kapitels.

Am Gerät ausprobieren: herrki.de/420_ziffern_erkennen#abschnitt-6

Direkt zum Modul



Interaktives Modul – am Gerät ausprobieren

Mehr Schichten ≠ automatisch besser

Robi hat rund **25.000** Gewichte, HerrKI knapp **6.000** — Robi ist also das „größere“ Netz und verliert trotzdem meistens. Der Unterschied liegt nicht in der Menge, sondern in der **Bauform**:

- **Robi** bekommt 784 einzelne Zahlen. Für ihn ist Pixel 300 kein Nachbar von Pixel 301 — er müsste sich jede Strichform an jeder Stelle des Bildes einzeln merken.
- **HerrKI** schiebt 5 × 5 große Lupen über das Bild. Ein Filter, der eine Ecke erkennt, erkennt sie überall — deshalb kommt er mit einem Viertel der Gewichte aus. Wie diese Lupen genau arbeiten, steht in „Faltung: die Lupe, die über das Bild wandert“.

Fair ist der Vergleich nicht ganz, und das sagen wir offen: Damit dein Browser nicht spürbar länger rechnet, bekommt Robi nur **2 Trainingsrunden** statt der 4 von HerrKI (er ist so schlicht, dass ihn das nur wenige Sekunden kostet — die Zahl steht oben im Kasten). Mit 4 Runden käme er ein Stück näher heran, aber die Faltung bekäme er dadurch nicht.

Und die Umkehrung gilt auch: Bei Tabellendaten — Wohnungspreise, Messwerte, Kundenzahlen — gibt es keine Nachbarschaft zwischen den Spalten. Dort gewinnt Robis schlichte Bauform, und ein Faltungsnetz wäre schlicht die falsche Wahl. „Welche Architektur?“ heißt immer: „Welche Struktur steckt in den Daten?“

Ehrlich bleiben: Warum hält es deine 7 manchmal für eine 1?

97 % auf den Testbildern — und dann erkennt es deine Ziffer nicht. Kein Widerspruch, sondern der wichtigste Effekt dieses Kapitels: **Verteilungs-Verschiebung** (englisch *distribution shift*). Das Netz hat nie „Ziffern“ gelernt. Es hat gelernt, *diese* 65.000 Bilder zu unterscheiden — und die kommen alle aus derselben Fabrik:

- **Anderer Stift:** MNIST-Striche sind bei 28×28 rund zwei Pixel breit. Deine Fingerspur ist relativ dazu vielleicht dreimal so dick oder halb so dünn — für die 5×5-Lupen ein völlig anderes Muster.
- **Andere Zentrierung:** Jede MNIST-Ziffer wurde per Schwerpunkt *mittig* in ein 20×20-Feld gesetzt. Malst du klein in die Ecke, sitzt deine Ziffer schlicht an einer Stelle, an der das Netz nie eine gesehen hat.
- **Andere Schreibkultur:** MNIST ist US-amerikanisch. Die deutsche 7 mit Querstrich und die 1 mit dickem Aufstrich kommen dort kaum vor — die beiden landen deshalb gern in derselben Schublade.

Gegenmittel für hier: groß malen, mittig, mit ruhigem Strich — und die 7 ruhig mal ohne Querstrich. **Gegenmittel in der echten Welt:** mit Daten trainieren, die aussehen wie die späteren Einsatzdaten, plus künstliche Variation (drehen, verschieben, verzerren). Genau deshalb trainiert die Post nicht auf MNIST, sondern auf Fotos ihrer eigenen Pakete.

Merksatz: Eine Genauigkeit gilt immer nur für Daten, die aussehen wie die Testdaten. „97 %“ ohne die Frage „auf welchen Daten?“ ist eine leere Zahl.

Zurück zur Anwendung

Zurück zur Sortieranlage vom Anfang: Die liest handgeschriebene Postleitzahlen im Sekundentakt — und genau so ein Netz hast du eben selbst gebaut, trainiert und geprüft. Du weißt jetzt, was dort passiert: Aus der ausgeschnittenen Ziffer wird ein [28,28,1]-Tensor, die Conv-Filter tasten ihn Nachbarschaft für Nachbarschaft ab, und am Ende stehen zehn Wahrscheinlichkeiten, von denen wir die größte zur Antwort erklären. Und du weißt auch, warum die Post trotzdem nicht auf MNIST trainiert, sondern auf Fotos ihrer eigenen Pakete — weil eine Genauigkeit immer nur für Daten gilt, die den Testdaten ähneln.

Als Nächstes wird aus dem „*was* ist das?“ ein „*was* ist *wo*?“: In „Was sieht die KI? (Objekterkennung)“ bekommt derselbe Filter-Turm einen zweiten Ausgang und malt beschriftete Rahmen auf deine eigenen Fotos.

Zum Weiterlesen

- LeCun u. a., *Die MNIST-Datenbank* — yann.lecun.com/exdb/mnist — die Originalseite zu genau dem Bilderstapel, mit dem du eben trainiert hast.
- LeCun u. a. 1998, *„Gradient-Based Learning Applied to Document Recognition“* (LeNet) — PDF (yann.lecun.com/exdb/publis/pdf/lecun-98.pdf) — die Arbeit, in der das Ziffern-CNN erfunden wurde, inklusive der Scheck-Leser aus der Praxis.
- TensorFlow.js Tutorial, *Handwritten digit recognition with CNNs* — tensorflow.org (tensorflow.org/js/tutorials/training/handwritten_digit_cnn) — derselbe Aufbau als Schritt-für-Schritt-Anleitung zum Nachbauen im eigenen Projekt.
- Stanford CS231n, *Convolutional Networks* — cs231n.github.io (cs231n.github.io/convolutional-networks) — wenn du wissen willst, wie Filtergröße, Schrittweite und Pooling die Form-Reise bestimmen.

Kurzer Check – mit Lösung

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

Frage 1: Warum benutzt man für Bilder Conv-Filter statt einfach einer großen Dense-Schicht auf allen 784 Pixeln?

- Weil Dense-Schichten keine Bilder als Eingabe annehmen können.
- **✓ Ein Filter arbeitet auf Nachbarschaften und ist überall im Bild derselbe — ein einmal gelerntes Muster wird an jeder Stelle wiedererkannt.**
- Weil Conv-Filter immer weniger Rechenzeit brauchen als jede andere Schicht.

Eine Dense-Schicht bekommt 784 lose Zahlen und weiß nicht, dass Pixel 100 und 101 nebeneinander liegen — sie müsste eine verschobene 3 komplett neu lernen. Der Conv-Filter tastet 5×5-Nachbarschaften ab und wandert mit denselben Gewichten über das ganze Bild: Nachbarschaft plus Verschiebungs-Toleranz, beides gratis per Bauart.

Frage 2: Am Ende steht ein Tensor der Form [10]. Was steht da drin?

- Die zehn Pixel, die dem Netz am wichtigsten waren.
- Die Ziffer selbst, zehnmal zur Sicherheit wiederholt.
- **✓ Zehn Wahrscheinlichkeiten — je eine pro Ziffer 0 bis 9, zusammen 1,00. Die Position des größten Wertes ist die Antwort.**

Das Netz sagt nie „das ist eine 3“. Es liefert zehn Zahlen, und erst wir erklären die größte zur Antwort. Deshalb ist auch immer eine Unsicherheit ablesbar: 0,98 für die 3 ist etwas ganz anderes als 0,34 für die 3 bei 0,31 für die 8. Das ist die Softmax-Ausgabe aus Kapitel 230.

Frage 3: Dein Netz schafft 97 % auf den Testbildern, patzt aber bei deiner gemalten 7. Was ist da los?

- Das Netz hat die Testbilder auswendig gelernt — klassisches Overfitting, sonst nichts.
- **✓ Deine Ziffer kommt aus einer anderen „Fabrik“ als die Trainingsdaten: anderer Strich, andere Zentrierung, andere Schreibweise — Verteilungs-Verschiebung.**
- Der Ausgabe-Tensor ist zu klein — mit 20 statt 10 Werten würde es klappen.

Die Testbilder stammen aus demselben Topf wie die Trainingsbilder: gleiche Stiftdicke, gleiche Schwerpunkt-Zentrierung, gleiche Schreibkultur. Deine Maus- oder Fingerzeichnung nicht. Eine Genauigkeit gilt immer nur für Daten, die den Testdaten ähneln — deshalb sagt „97 %“ ohne die Frage „auf welchen Daten?“ genau gar nichts.

Das nimmst du mit

- **Ein Bild ist ein Tensor.** 28×28 Graustufen-Pixel, ein Kanal — [28,28,1], und jede dieser Zahlen konntest du dir selbst anschauen.
- **Der Conv-Filter schlägt bei Bildern die Dense-Schicht.** Er tastet 5×5-Nachbarschaften ab und wandert mit denselben Gewichten über das ganze Bild: Nachbarschaft und Verschiebungs-Toleranz gibt es gratis per Bauart.
- **Am Ende steht ein [10]-Tensor.** Zehn Wahrscheinlichkeiten, zusammen 1,00 — das Netz sagt nie „das ist eine 3“, wir erklären nur die größte Zahl zur Antwort.
- **MNIST ist eine Fabrik mit festen Regeln.** Rund zwei Pixel Strichbreite, per Schwerpunkt mittig in ein 20×20-Feld gesetzt, US-amerikanische Schreibkultur.
- **Verteilungs-Verschiebung erklärt deine 7.** „97 %“ gilt nur für Daten, die aussehen wie die Testdaten — anderer Stift, andere Zentrierung, andere Schreibweise sind für das Netz ein anderes Problem.

Geschafft — und wo geht es weiter? Du hast ein CNN von Hand gebaut, auf echten Daten trainiert und gegen deine eigene Handschrift antreten lassen. Die Bausteine dahinter kennst du längst einzeln: die verborgenen Schichten aus „Was ist ein Neuronales Netz?“, Softmax und One-Hot aus „Multi-Class Klassifikation“, die Trennung von Übung und Prüfung aus „Trainieren, prüfen, bestehen“ — und die Frage, wann ein Netz auswendig lernt statt zu verstehen, aus „Overfitting & Regularisierung“.

Als Nächstes in der Werkzeug-Reihe: **430 — Objekterkennung**. Dort geht es nicht mehr um *was* ist auf dem Bild, sondern um *wo genau* — und du wirfst ein fertiges Netz auf deine eigenen Fotos. Dieselben Filter, nur ein paar Größenordnungen mehr davon.

So zitierst du dieses Kapitel

HerrKI: Male eine Ziffer — dein erstes CNN. herrki.de/420_ziffern_erkennen, ein privates Hobbyprojekt von Terragon.de, abgerufen am 09.09.2026.