10 Maschinelles Sehen

Willkommen zu Teil III, liebe Leser! In Teil I gaben wir Ihnen einen Überblick über spezielle Anwendungen des Deep Learning. Mit der grundlegenden Theorie, die wir anschließend in Teil II behandelt haben, sind Sie nun bestens ausgestattet, um sich einer Reihe von Anwendungen aus verschiedenen Bereichen zu widmen. Dies wird vor allem in Form von Codebeispielen geschehen. In diesem Kapitel untersuchen Sie z. B. Convolutional Neural Networks und setzen diese für Aufgaben aus dem Gebiet des maschinellen Sehens (Machine Vision) ein. Außerdem behandeln wir im weiteren Verlauf von Teil III praktische Beispiele für:

- Recurrent Neural Networks zur Verarbeitung natürlicher Sprache (in Kapitel 11)
- Generative Adversarial Networks für eine visuelle Kreativität (in Kapitel 12)
- Deep Reinforcement Learning für eine sequenzielle Entscheidungsfindung in komplexen, veränderlichen Umgebungen (in Kapitel 13)

10.1 Convolutional Neural Networks

Ein *convolutional*, zu Deutsch »faltendes«, neuronales Netz – auch *Convolutional Neural Network*, ConvNet oder CNN genannt – ist ein künstliches neuronales Netz, das ein oder mehrere Konvolutionsschichten oder *Convolutional Layers* aufweist. Dieser Schichttyp erlaubt es einem Deep-Learning-Modell, effizient räumliche Muster zu verarbeiten. Wie Sie in diesem Kapitel aus erster Hand erfahren werden, eignen sich solche Netzwerke daher besonders gut für Anwendungen aus dem Bereich des maschinellen Sehens oder der Computer Vision.

10.1.1 Die zweidimensionale Struktur der visuellen Bilddarstellung

In unseren bisherigen Codebeispielen mit den handgeschriebenen MNIST-Ziffern wandelten wir die Bilddaten in ein eindimensionales Array aus Zahlen um, die wir dann in die vollständig verbundene verborgene Schicht eingaben. Genauer gesagt, begannen wir mit 28×28 Pixel großen Graustufenbildern und verwandelten diese in eindimensionale Arrays mit 784 Elementen.[1] Obwohl dieser Schritt im Kontext eines vollständig verbundenen Netzwerks notwendig war (wir mussten die 784 Pixelwerte egalisieren, damit sie jeweils in ein Neuron der ersten verborgenen Schicht eingegeben werden konnten), bringt die Reduzierung eines zweidimensionalen Bildes in eine Dimension einen beträchtlichen Strukturverlust mit sich. Wenn Sie mit einem Stift eine Ziffer auf Papier zeichnen, stellen Sie sich das Ganze nicht als eine kontinuierliche lineare Abfolge von Pixeln vor, die von oben links nach unten rechts verlaufen. Würden wir z.B. hier eine MNIST-Ziffer als 784 Pixel langen Strom aus Graustufen ausdrucken, dann wetten wir, dass Sie nicht in der Lage wären, die Ziffer zu erkennen. Menschen nehmen visuelle Informationen nämlich in einer zweidimensionalen Form wahr,[2] und unsere Fähigkeit, zu erkennen, was wir anschauen, ist von Natur aus mit den räumlichen Beziehungen zwischen den Formen und Farben verbunden, die wir wahrnehmen.

10.1.2 Berechnungskomplexität

Neben dem Verlust der zweidimensionalen Struktur beim Reduzieren eines Bildes müssen wir noch eine zweite Sache bedenken, wenn wir Bilder in ein vollständig verbundenes Netzwerk leiten: die Berechnungskomplexität. Die MNIST-Bilder sind sehr klein – 28×28 Pixel mit nur einem *Kanal*. (Es gibt nur einen Farb»kanal«, weil MNIST-Ziffern monochromatisch sind; um vollfarbige Bilder darzustellen, sind wenigstens drei Kanäle – üblicherweise Rot, Grün und Blau – erforderlich.) Wenn man MNIST-Bildinformationen in eine vollständig verbundene Schicht übergibt, entspricht dies 785 Parametern pro Neuron: 784 Gewichte für jedes der Pixel plus der Bias des Neurons. Würden wir jedoch ein mäßig großes Bild verarbeiten – sagen wir ein 200×200 Pixel großes vollfarbiges RGB-Bild[3] – nimmt die Anzahl der Parameter ganz drastisch zu. In diesem Fall hätten wir drei Farbkanäle, jeweils mit 40.000 Pixeln, was insgesamt 120.001 Parametern pro Neuron in einer vollständig verbundenen Schicht entspricht.[4] Bei einer bescheidenen

1. Erinnern Sie sich daran, dass die Pixelwerte durch 255 dividiert wurden, um alles in den Bereich [0 : 1] zu skalieren.
2. Na ja … dreidimensional, aber wir wollen die Tiefe bei dieser Diskussion einfach einmal ignorieren.
3. Die erforderlichen *Rot*-, *Grün*- und *Blau*-Kanäle für ein vollfarbiges Bild.
4. 200 Pixel × 200 Pixel × 3 Farbkanäle + 1 Bias = 120.001 Parameter.

Anzahl an Neuronen in der Schicht – sagen wir, 64 – wären das fast 8 Millionen Parameter allein für die erste verborgene Schicht unseres Netzwerks.[5] Außerdem hat das Bild nur 200×200 Pixel – das sind kaum 0,4 MP[6], während die meisten modernen Smartphones Kamerasensoren mit 12 MP oder mehr haben. Im Allgemeinen müssen Machine-Vision-Aufgaben nicht auf hochaufgelösten Bildern ausgeführt werden, um erfolgreich zu sein, aber der Punkt ist doch klar: Bilder können sehr viele Datenpunkte enthalten, und wenn man diese naiverweise in einer vollständig verbundenen Struktur verarbeiten möchte, explodieren die Anforderungen an die Rechenkapazitäten des neuronalen Netzes geradezu.

10.1.3 Konvolutionsschichten

Konvolutionsschichten (Convolutional Layers) bestehen aus Gruppen von *Kerneln*, die auch als *Filter* oder Filterkernel bezeichnet werden. Jeder dieser Kernel ist ein kleines Fenster (ein sogenanntes *Patch*), das das Bild von oben links nach unten rechts scannt (technisch ausgedrückt: Der Filter führt eine *Konvolution* (Faltung) durch). Abbildung 10–1 verdeutlicht diese *Konvolutionsoperation*.

Kernel bestehen aus Gewichten, die – wie in vollständig verbundenen Schichten – durch Backpropagation gelernt werden. Ihre Größe ist unterschiedlich, typischerweise sind sie 3×3 groß, und so werden wir das auch in den Beispielen in diesem Kapitel nutzen.[7] Für die monochromatischen MNIST-Ziffern würde dieses 3×3-Pixel-Fenster aus 3×3×1 Gewichten bestehen – d. h., aus neun Gewichten für insgesamt 10 Parameter. (Wie ein künstliches Neuron in einer vollständig verbundenen Schicht hat auch jeder Konvolutionsfilter einen Bias-Term b.) Zum Vergleich: Würden wir mit vollfarbigen RGB-Bildern arbeiten, dann hätte ein Kernel, der die gleiche Anzahl an Pixeln abdeckt, dreimal so viele Gewichte – 3×3×3 für insgesamt 27 Gewichte und 28 Parameter.

5. 64 Neuronen × 120.001 Parameter pro Neuron = 7.680.064 Parameter.
6. Megapixel.
7. Eine andere typische Größe ist 5×5, wobei Kernel, die größer sind als diese, nur selten verwendet werden.

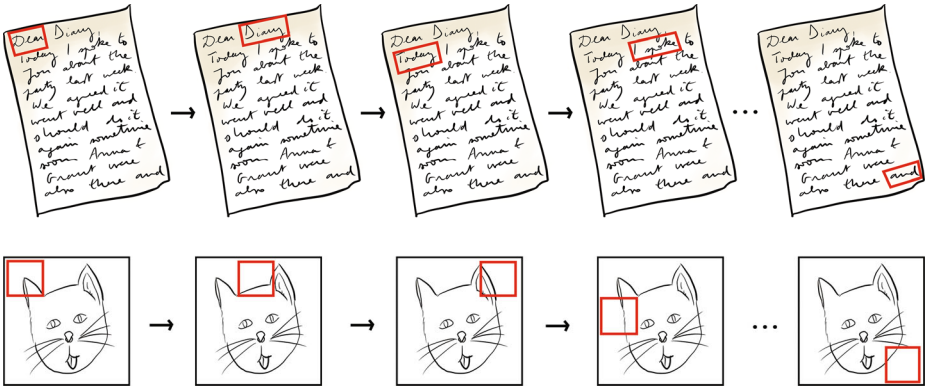

Abb. 10–1 *Wenn wir die Seite eines Buches lesen, das in Englisch (oder Deutsch) geschrieben ist, beginnen wir in der obersten linken Ecke und lesen nach rechts. Immer wenn wir das Ende einer Textzeile erreichen, machen wir in der nächsten Zeile weiter. Auf diese Weise erreichen wir irgendwann die untere rechte Ecke und haben damit alle Wörter auf der Seite gelesen. Analog beginnt der Kernel in einer Konvolutionsschicht in einem kleinen Fenster aus Pixeln in der oberen linken Ecke eines Bildes. Der Kernel scannt von der oberen Reihe nach unten, und zwar von links nach rechts, bis er schließlich die untere rechte Ecke erreicht hat. Damit hat er dann alle Pixel des Bildes gescannt.*

Wie in Abbildung 10–1 dargestellt ist, besetzt der Kernel während seiner Faltungsoperation diskrete Positionen entlang dem Bild. Bleiben wir für unsere Erklärung einmal bei der Kernel-Größe 3×3. Während der Forwardpropagation wird eine multidimensionale Variante der »wichtigsten Gleichung in diesem Buch« $w \cdot x + b$ (vorgestellt in Abbildung 6–7) an jeder Position berechnet, die der Kernel während der Konvolution entlang dem Bild einnimmt. Wenn wir das 3×3-Fenster aus Pixeln und den 3×3-Kernel in Abbildung 10–2 als Eingaben x bzw. Gewichte w annehmen, dann können wir die Berechnung der gewichteten Summe $w \cdot x$ demonstrieren, bei der Produkte elementweise berechnet werden. Die Berechnung basiert dabei auf der Ausrichtung der vertikalen und horizontalen Stellen. Es hilft, wenn man sich vorstellt, dass der Kernel über die Pixelwerte gelegt ist. Die Berechnung ist hier:

$$\begin{aligned} w \cdot x &= 0{,}01 \cdot 0{,}53 + 0{,}09 \cdot 0{,}34 + 0{,}22 \cdot 0{,}06 \\ &\quad + -1{,}36 \cdot 0{,}37 + 0{,}34 \cdot 0{,}82 + -1{,}59 \cdot 0{,}01 \\ &\quad + 0{,}13 \cdot 0{,}62 + 0{,}69 \cdot 0{,}91 + 1{,}02 \cdot 0{,}34 \\ &= -0{,}3917 \end{aligned}$$

Gleichung 10–1

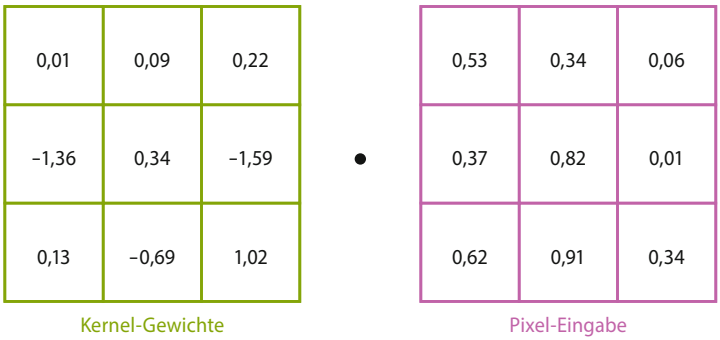

Abb. 10–2 *Ein 3×3-Kernel und ein 3×3-Pixelfenster*

Jetzt addieren wir entsprechend Gleichung 7–1 noch einen Bias b (sagen wir, 0,19), um z zu erhalten:

$$\begin{aligned} z &= w \cdot x + b \\ &= -0{,}39 + b \\ &= -0{,}39 + 0{,}20 \\ &= -0{,}19 \end{aligned}$$

Gleichung 10–2

Mit z können wir schließlich einen Aktivierungswert a berechnen, indem wir z an die Aktivierungsfunktion unserer Wahl übergeben, etwa an die Tanh-Funktion oder an die ReLU-Funktion.

Sie sehen, dass sich die grundlegenden Rechenoperationen im Vergleich zu den künstlichen Neuronen aus den Kapiteln 6 und 7 nicht geändert haben. Faltende Kernel haben Gewichte, Eingaben und einen Bias. Daraus wird mithilfe unserer wichtigsten Gleichung eine gewichtete Summe erzeugt und das daraus resultierende z wird an irgendeine nichtlineare Funktion übergeben, um eine Aktivierung zu erzeugen. Geändert hat sich, dass es nicht für *jede* Eingabe ein Gewicht gibt, sondern einen diskreten Kernel mit 3×3 Gewichten. Diese Gewichte ändern sich nicht, während der Kernel die Konvolution durchführt, sondern sie werden über *alle* Eingaben verwendet. Auf diese Weise kann eine Konvolutionsschicht um Größenordnungen weniger Gewichte haben als eine vollständig verbundene Schicht. Ein weiterer wichtiger Punkt ist, dass die Ausgaben dieses Kernels (alle Aktivierungen) genau wie die Eingaben in einem zweidimensionalen Array angeordnet sind. Wir werden uns das gleich genauer anschauen, aber zuvor …

10.1.4 Mehrere Filter

Typischerweise haben wir mehrere Filter in einer Konvolutionsschicht. Jeder Filter erlaubt es dem Netzwerk, eine Repräsentation der Daten in einer bestimmten Schicht auf einzigartige Weise zu lernen. Falls z. B. analog zu den einfachen Zellen von Hubel und Wiesel im biologischen visuellen System (Abbildung 1–5) die erste verborgene Schicht in unserem Netzwerk eine Konvolutionsschicht ist, dann könnte sie einen Kernel enthalten, der optimal auf vertikale Linien reagiert. Das bedeutet: Immer wenn er über eine vertikale Linie im Eingabebild gleitet, erzeugt er einen großen Aktivierungswert (*a*). Weitere Kernel in dieser Schicht können lernen, andere einfache räumliche Eigenschaften darzustellen, wie etwa horizontale Linien und Farbübergänge. (Schauen Sie sich z. B. das Bild unten links in Abbildung 1–17 an.) Auf diese Weise kamen die Kernel zu ihrer Bezeichnung *Filter*: Sie überfliegen das Bild und filtern den Ort bestimmter Features heraus. Dabei erzeugen sie hohe Aktivierungen, wenn sie einem Muster, einer Form und/oder einer Farbe begegnen, auf deren Erkennung sie spezialisiert sind. Man könnte sagen, dass sie als Marker fungieren, die ein zweidimensionales Array aus Aktivierungen erzeugen, die andeuten, *wo* das besondere Feature dieses Filters im Originalbild existiert. Aus diesem Grund wird die Ausgabe eines Kernels als *Aktivierungs-Map* bezeichnet.

Analog zu den hierarchischen Repräsentationen des biologischen visuellen Systems (Abbildung 1–6) erhalten nachfolgende Konvolutionsschichten diese Aktivierungs-Maps als Eingaben. Je tiefer das Netzwerk wird, umso komplexer werden die Kombinationen dieser simplen Features, auf die die Filter in diesen Schichten reagieren, d. h., umso abstrakter werden die räumlichen Muster, die repräsentiert werden. So baut sich eine Hierarchie aus einfachen Linien und Farben nach und nach zu komplexen Texturen und Formen auf (siehe die Einzelbilder unten in Abbildung 1–17). Auf diese Weise haben spätere Schichten innerhalb des Netzwerks die Fähigkeit, ganze Objekte zu erkennen oder sogar ein Bild einer Dänischen Dogge von dem eines Yorkshire Terriers zu unterscheiden.

Die Anzahl der Filter in der Schicht ist genauso wie die Anzahl der Neuronen in einer vollständig verbundenen Schicht ein Hyperparameter, den wir selbst konfigurieren können. Wie bei den anderen Hyperparametern, die wir bereits besprochen haben, gibt es auch hier einen Goldlöckchen-haften Mittelweg für die Anzahl der Filter. Mithilfe dieser Faustregeln können Sie die nötigen Filter für ein bestimmtes Problem festlegen:

- Eine größere Anzahl an Kerneln erlaubt die Identifizierung komplexerer Features. Sie sollten daher die Komplexität der Daten und des zu lösenden Problems in Betracht ziehen. Natürlich geht eine größere Anzahl an Kerneln zulasten der Berechnungseffizienz.
- Wenn ein Netzwerk mehrere Konvolutionsschichten besitzt, kann die optimale Anzahl der Kernel von Schicht zu Schicht stark schwanken. Denken Sie daran, dass frühere Schichten einfache Features erkennen, während spätere Schichten komplexe Neukombinationen dieser einfachen Features identifizieren. Berücksichtigen Sie dies beim Entwurf des Netzwerks. Wie wir bei den Codebeispielen für CNNs weiter hinten in diesem Kapitel sehen werden, besteht ein gern genommener Ansatz darin, in späteren Konvolutionsschichten im Verhältnis mehr Kernel zu verwenden als in früheren Schichten.
- Streben Sie wie immer danach, die Berechnungskomplexität zu minimieren: Versuchen Sie, die kleinstmögliche Anzahl an Kerneln zu benutzen, die niedrige Kosten für Ihre Validierungsdaten verursacht. Falls eine Verdoppelung der Anzahl an Kerneln (etwa von 32 auf 64 auf 128) in einer Schicht die Validierungskosten Ihres Modells merklich senkt, sollten Sie möglicherweise den höheren Wert nehmen. Erhöht eine Halbierung der Anzahl (etwa von 32 auf 16 auf 8) die Validierungskosten des Modells nicht, dann sollten Sie den niedrigeren Wert nutzen.

10.1.5 Ein Beispiel für Konvolutionsschichten

Konvolutionsschichten (Faltungsschichten) sind eine nichttriviale Abweichung von den einfacheren vollständig verbundenen Schichten aus Teil II. Damit Sie verstehen, wie die Pixelwerte und Gewichte gemeinsam Feature-Maps erzeugen, zeigen wir in den Abbildungen 10–3 bis 10–5 ein ausführliches Beispiel und erläutern auch die Berechnungen, die damit einhergehen. Stellen Sie sich zu Beginn vor, wir falten über ein einzelnes RGB-Bild von 3×3 Pixeln Größe. In Python werden diese Daten in einem `[3,3,3]`-Array gespeichert, wie oben in Abbildung 10–3 zu sehen ist.[8]

8. Wir geben zu, dass das RGB-Beispiel des Baumes deutlich mehr als nur neun Pixel hat, aber wir hatten Probleme, ein passendes Farbbild zu finden, das 3×3 Pixel groß ist.