HANSER

Leseprobe

zu

„Maschinelles Lernen“

von Jörg Frochte

ISBN (Buch): 978-3-446-46144-4
ISBN (E-Book): 978-3-446-46355-4

Weitere Informationen und Bestellungen unter

https://www.hanser-fachbuch.de/978344646144-4

sowie im Buchhandel

Inhalt

1 Einleitung

Manche sagen, maschinelles Lernen sei ein Teilgebiet der künstlichen Intelligenz, andere ein Hilfsmittel in Disziplinen wie Data Mining oder Information Retrieval. Es hängt von der Sichtweise ab. Für mich ist es das Gebiet, das die interessantesten Aspekte zusammenbringt, nämlich Informatik, Mathematik und Anwendungen, die sehr vielfältig sind. Man kann hier mit Menschen zusammenarbeiten, denen es um autonomes Fahren oder um lernende Roboter in Industrie und Haushalt geht. Andere Anwendungsfelder sind beispielsweise das Verhalten von Menschen in Online-Shops oder Prognosen über Kreditwürdigkeit.
Das Feld ist aber deutlich breiter: Ein Kollege von mir setzt zum Beispiel in einem Projekt maschinelles Lernen ein, um Stimmen bedrohter Vögel in Neuseeland – *http://avianz.massey.ac.nz/* bzw. [PMC18] – zu erkennen und auch um zu ermitteln, wie viele Vögel wirklich auf einer Aufnahme zu hören sind. Das ist nicht leicht, denn es könnte dreimal derselbe Vogel sein, der ruft, oder eben drei verschiedene. Das ist maschinelles Lernen in der Ökologie. Andere Kollegen arbeiten mit Satellitendaten unterschiedlicher Qualität und versuchen so, Aussagen zu treffen, um die ausgebrachte Wasser- und Düngermenge zu optimieren. Das sind zwar alles sehr ernsthafte Fälle. Man darf aber auch einfach Spaß haben und versuchen, die KI in einem Computerspiel besser und unterhaltsamer zu machen. Das maschinelle Lernen ist also eine Art Schweizer Taschenmesser, welches man in den unterschiedlichsten Situationen sinnvoll und unterhaltsam einsetzen kann.

Ich versuche in diesem Buch, die meiner Ansicht nach wichtigsten Techniken und Ansätze darzustellen. Es enthält aber zunächst nur eines von diesen mitteldicken Schweizer Taschenmessern. Wenn Sie das Buch durchgearbeitet haben, schauen Sie mal nach folgenden Begriffen, die es nicht in die dritte Auflage geschafft haben: Outlier Detection, Radiale-Basisfunktionen-Netze, Self Organizing Maps, Recurrent Neural Networks …

Es gibt viele interessante Themen in diesem Feld. Am Ende des Buches haben Sie sich bestimmt die Grundlagen – und auch etwas mehr – angeeignet, um sich schnell in neue Bereiche einarbeiten zu können. Um die ganzen Werkzeuge in diesem Schweizer Taschenmesser sinnvoll nutzen zu können, sollte man einen breiten Überblick über das Gebiet haben und sich nicht auf eine Technik beschränken. Aktuell sind z. B. Convolutional Neural Networks sehr in Mode und natürlich kann man auch diese irgendwie benutzen, ohne verstanden zu haben, wie neuronale Netze überhaupt funktionieren. Ein Keras-Tutorial dazu kann man schnell abtippen und sehen, wie der eigene Computer Ziffern mit hoher Genauigkeit erkennt. Man muss zwar nicht immer alles durchdringen, aber wenn man sich dafür interessiert, will man dort nicht stehen bleiben, oder?
Ich verstehe zum Beispiel nicht viel von meinem Auto, weil es mich nicht interessiert. Meine Ignoranz funktioniert hervorragend, weil das Produkt recht gut entworfen ist und ich nicht den Wunsch habe, an ihm herumzuschrauben. Als reiner User fahre ich bei Problemen in eine Werkstatt. Ich gehe aber davon aus, dass Sie mit dem maschinellen Lernen mehr machen wollen als sich hineinzusetzen und loszufahren. Sie sind Designer von Lösungen, kreativer Kopf hinter neuen Anwendungen oder die Werkstatt, die sich um die rote Warnlampe kümmert.

Wer hier nur ein Werkzeug kennt, läuft in eine Falle, die als *Law of the instrument* oder auch **Maslows Hammer** nach dem Ausspruch *If all you have is a hammer, everything looks like a nail* bekannt ist. Um hier für unterschiedliche Probleme mit großen und kleinen Datenmengen ebenso wie für unterschiedliche Ressourcenlagen Lösungen anbieten zu können, versuchen wir es mit vielen Werkzeugen. Das Ziel ist es, beim Kennenlernen dieser Werkzeuge zwischen den beiden Monstern Skylla (Theorielastigkeit) und Charybdis (flache Unbedeutendheit) möglichst unbeschadet hindurch zu kommen. Das bedeutet, ich möchte, dass Sie am Ende des Buches die mathematischen Hintergründe dieser Technik im Wesentlichen kennen, aber nicht notwendigerweise in der trockenen Form aus Definition, Satz und Beweis. Ich bin überzeugt, dass ein guter Mittelweg darin besteht, möglichst viele Algorithmen aus dem Bereich einmal selbst umzusetzen. Benutzt man nur fertige Bibliotheken, ist es etwa so, als würde jemand glauben, vom Zusehen schwimmen gelernt zu haben. Ich möchte also mit Ihnen gemeinsam wirklich *schwimmen* und Algorithmen basierend auf Verständnis und Theorie umsetzen. Dabei ist es in Ordnung, wenn unsere Umsetzungen nicht das Rennen bzgl. der Performance gewinnen. Es geht darum, die Prinzipien und theoretischen Grundlagen einmal ausprobiert zu haben. Allerdings soll das kein fundamentalistisches Dogma sein. Wer glaubt, er habe den Ansatz gefunden, der immer und für jeden funktioniert, ist im besten Fall eine Gefahr für sich und im schlimmsten für andere Menschen.
Es gibt ein paar Stellen, an denen wir mit der Idee, die Dinge from-scratch umzusetzen, nicht weiterkommen würden: tiefe neuronale Netze (Deep Learning) und Support Vector Machines. Beide benötigen mehr Wissen und Software rund um Optimierung und Co. als in dieses Buch passt. Gleichzeitig sind sie sehr spannend, sodass man sie nicht einfach weglassen sollte, nur weil die Umsetzung nicht gut auf ein paar Buchseiten passt. Wir setzen daher die neuronalen Netze in ihrer klassischen Form zu Fuß um und gehen dann für die tiefen dazu über, Keras als Bibliothek zu verwenden. Im Zusammenhang mit klassischen Netzen handeln wir fast alle Fallen und Begriffe ab und gehen dann mit Keras zu Anwendungen über, die mehr Leistung oder bessere Optimierung brauchen. Dasselbe gilt in gewisser Weise für die Support Vector Machines, die ebenfalls ein Modul aus der quadratischen Optimierung benötigen; nur weglassen sollte man sie nicht. Hier schauen wir uns erst etwas theoretischer die Grundlagen an und greifen dann zum Ausprobieren zur fertigen Umsetzung aus scikit-learn.

Weil wir abseits dieser beiden Fälle tendenziell über Algorithmen gehen, versuchen wir vorzugsweise, einen eher algorithmischen statt einen statistischen Zugang zu nehmen. Das liegt auch daran, für welche Zielgruppe ich gewöhnlich maschinelles Lernen aufbereite. Ich selbst unterrichte das Fach für Ingenieure oder angewandte bzw. technische Informatiker. Die Ingenieure in der Praxis oder an der Hochschule sind oft mit MATLAB vertraut und haben dort ggf. bereits etwas mit maschinellem Lernen versucht. Die Informatiker in den Studiengängen hatten Java, C und ggf. MATLAB. Das meiner Meinung nach beste und kostengünstigste Ökosystem zum maschinellen Lernen findet man jedoch bei Python oder R. Letzteres ist gerade bei Statistikern sehr beliebt. Für Ingenieure ziehe ich Python R vor; u. a. wegen dessen besserer Einbindung, auch in Robotik-Projekten, und wegen des leichten Umstiegs. Der Sprung von MATLAB zu Python ist gering, muss aber immer gemacht werden. Ziemlich genau diese Sprunghöhe, die jemand schaffen muss, der von MATLAB bzw. GNU/Octave zu Python wechseln möchte, ist auch im Buch eingebaut. Es funktioniert aber auch, wenn jemand von Java oder C++ kommt.

Daneben sind Ingenieure oder Ingenieurinformatiker oft sehr gut ausgebildet in linearer Algebra und Analysis, dafür fehlt die Statistik in der Ausbildung. Daher habe ich auch die Statistik

in der Darstellung des maschinellen Lernens möglichst knapp gehalten und den wirklich notwendigen Teil der Mathematik ins Buch integriert.

Die Analysis und lineare Algebra – in Kapitel 5 – sind in weiten Teilen eingebettet, wenn auch teilweise auf dem Niveau einer Erinnerung.

Im Buch baue ich die Quelltexte immer (fast) vollständig ein. Sie können die Quellen natürlich auch von meiner Seite *https://joerg.frochte.de* downloaden, aber mir ist es wichtig, dass der Python-Code wirklich ins Buch eingebunden ist. Wenn man einen algorithmischen Zugang versucht, sind die Algorithmen eben wesentlich. Außerdem möchte ich gerne, dass man das Buch sowohl vor dem Computer benutzen kann – direkt alles mitmachen und ausprobieren – als auch in einem Park sitzend und nur lesend. Ich selbst lese gerne auch gedruckte Fachbücher als Entspannung, wenn ich gerade keinen Bildschirm sehen will … und ich vermute, ich bin damit nicht allein. Man kann Algorithmen in Python tatsächlich sehr kompakt notieren und auf die wesentlichen Ideen beschränken, was Python ebenfalls für den Abdruck interessant macht. Bezüglich des Codes wird jemandem, der Python schon länger benutzt, etwas auffallen: Ich benutze kein snake_case, was in Python ungewöhnlich ist. Gründe sind sicherlich, dass meine Kursteilnehmer von C, MATLAB oder Java kommen, ich selbst auch oft zwischen diesen Programmiersprachen wechsle und mir dabei eine Art Crossover-Stil angewöhnt habe. Ich hoffe, es stört niemanden, der an sauberes PEP 8 gewöhnt ist, und bitte falls doch um Nachsicht. Der Stil für die Variablennamen ist hier aber nicht so wichtig. Wir haben es die meiste Zeit mit sehr kurzen Variablen zu tun, wie X für die Trainingsmenge und Y für die Menge der Ziele usw. Da wölbt sich keine Schlange und macht kein Kamel einen Höcker.

In den Python-Codes, die einen wichtigen Teil des Buches ausmachen, müssen wir ohne Pfeile oder Fettdruck für Vektoren und Matrizen auskommen. Entsprechend lassen wir dies auch für die Formeln weg und bemühen uns, so zu klären, was was ist. Formeln haben immer dann eine Nummer, wenn auf diese später noch einmal verwiesen wird. Gleichheitszeichen im Pseudocode sind i. d. R. als *gleichgesetzt*, also als Zuweisung, zu lesen.

Im Buch sind drei Arten von „Boxen“ eingebaut:

Diese hat etwas damit zu tun, wo Sie in **scikit-learn** den Algorithmus, den wir gerade umgesetzt haben, finden können. Es ist lediglich ein Verweis auf professionelle Umsetzungen. **Keras** und scikit-learn sind für mich zwei sehr wichtige Stützpfeiler, um schnell und gut etwas in Python umsetzen zu können. Ich habe die Verweise auf diese beiden Bibliotheken beschränkt.

Die zweite Textbox ist ein allgemeines *Achtung*. Ich setze diese Box nicht so oft ein, weil irgendwie alles oder nichts wichtig ist. Aber manche Dinge sind doch ärgerlicher als andere und kosten sehr viel Zeit, wenn man sie überliest. Wenn ich einen dieser Fälle bereits erlebt habe, taucht diese Box auf.

Die wichtigste Gruppe von Boxen erkennt man an dem Schraubenschlüssel. Hier gibt es Anregungen, was Sie selbst anschließend ausprobieren könnten. Keine von diesen ist nötig, um dem Rest des Buches zu folgen. Es sind einfach Vorschläge, da-

mit Sie selbst etwas mit dem neu Gelernten anfangen können, ohne dass die Lösung sofort danebensteht. Oft gibt es nicht DIE Lösung, sondern eben sehr viele Ansätze.

Wenn ich eine Variable aus einem Quellcode im Fließtext verwende, wird diese als `Schreibmaschinenschrift` gesetzt. Dinge, die fettgedruckt sind, tauchen im Index hinten auf. Der Sinn liegt darin, dass Sie etwas hinten im Index suchen und dann auf der Seite sofort sehen, wo es steht. In der Regel wird etwas nur beim ersten Auftauchen in den Index aufgenommen. Ansonsten bedeutet ein kursiver Druck etwas Ähnliches wie *Eigenname* oder *In-Anführungszeichen*. Der Einstieg in Python ist in Kapitel 3 konzentriert. Wer Python zusammen mit NumPy & Co. schon beherrscht, kann das Kapitel überspringen. Alle anderen erhalten in Kapitel 3 einen schnellen praktischen Einstieg, wie mit Matrizen und Arrays in NumPy gearbeitet wird. Im Unterschied zu Python als Grundlage habe ich beim Aufbau des Buches versucht, auch die Einarbeitung der mathematischen Grundlagen über mehrere Kapitel zu verteilen und nicht in einem Einstiegskapitel oder Anhang zu bündeln; einfach damit es nicht einen langen trockenen Teil gibt und dann viele Passagen, die quasi primär aus Pseudo- bzw. Quellcode bestehen. In Kapitel 4 folgt zusammen mit dem ersten Klassifikator ein guter Teil der minimalisierten statistischen Grundlagen, die wir brauchen. In Kapitel 5 gehen wir noch einmal tiefer auf Vektorräume, Normen und Metriken ein. Diese sind u. a. notwendig, wenn man hinterher, wie in Kapitel 13, versucht, Grade von Ähnlichkeiten zwischen Objekten zu ändern, und zwar durch die Art, wie Abstände definiert werden. In den Kapiteln 7 und 8 ist wiederum etwas Optimierung eingebaut, welche nötig ist, um neuronale Netze zu trainieren.

Die Auswahl von Merkmalen und ihre Reduktion sind das Thema des Kapitels 9. Hier brauchen wir noch einen Nachschlag bzgl. der Statistik und dazu Grundlagen zu Eigenwerten und Eigenvektoren. Diese sind nötig, um die Principal Component Analysis richtig einzuordnen. Vielleicht etwas ungewöhnlich ist die Lage des Kapitels 9 innerhalb des Buches. Man könnte eigentlich annehmen, dass die Diskussion über die Daten weiter vorne kommen sollte. Jedoch wollte ich für einige Demonstrationen schon Lernverfahren zur Verfügung haben, wozu neuronale Netze gehören, da diese z. B. andere Ansprüche haben als ein Entscheidungsbaum. Neu in der dritten Auflage ist, dass ich versuche, in dieses Kapitel Pandas als wohl wichtigste Bibliothek im Umgang mit strukturierten Daten einzubinden.

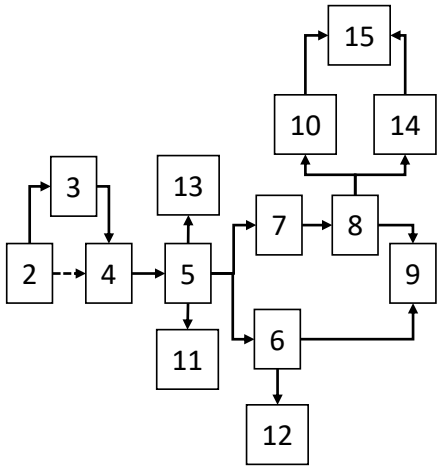

Abbildung 1.1 Abhängigkeiten zwischen den Kapitel des Buches

Wer das Buch in einer Vorlesung einsetzen will und kürzen möchte bzw. muss, für den habe ich die Abbildung 1.1 eingebaut. Sie gibt die wesentlichen Abhängigkeiten wieder. Es kann immer mal irgendwo ein Verweis auch außerhalb dieser Abhängigkeiten auftauchen. Daneben gibt es in jedem Kapitel viele Möglichkeiten zu kürzen, abhängig von den Vorkenntnissen der Teilnehmerinnen und Teilnehmer oder den Zielen zu kürzen. Wer z. B. eigentlich bestärkendes Lernen mit neuronalen Netzen machen möchte, der kann sich im Kapitel 8, auf die ersten zwei beiden Abschnitte konzentrieren. Generell kann man dann vieles auf dem Weg anpassen oder weglassen; dafür sind Dozentinnen und Dozenten eben wichtig.

Jetzt sollten wir aber anfangen, nachdem ich mich bei einigen Menschen bedankt habe. Das sind im Vergleich zur letzten Ausgabe einige mehr geworden. Ich möchte mich wirklich bei Lesern bedanken, die mir Anregungen schicken oder Stellen nennen, bei denen die Erklärungen im Buch nicht rundlaufen. Diese haben mich – ebenso wie die Arbeit zusammen mit Herrn Kaufmann am Projekt Weiterbildung AI (*https://we-ai.de*) – sehr weitergebracht bzgl. der Frage wo man noch was ändern oder ergänzen müsste. Neben den Leserinnen und Lesern, die mir konstruktive Rückmeldungen per E-Mail geben, haben mir auch viele Kolleginnen und Kollegen geholfen. Ich vergesse bestimmt jemanden und möchte mich dafür entschuldigen, jedoch unter anderem möchte ich mich bedanken bei Sabine Weidauer, Benno Stein, Matthias Rottmann, Peter Gerwinski, Herbert Schmidt, Michael Knorrenschild, Peter Beater, Christof Kaufmann, Henrik Blunck, Marco Schmidt, Stefan Müller-Schneiders, René Schoesau, Stefan Bader, Roland Schroth, Annabel Lindner, Julia Sudhoff und Gernot Kucera.

Wenn man ein Buch schreibt, kostet das immer viel zusätzliche Zeit neben dem normalen Beruf. Daher vielen Dank, dass ich mir diese nehmen durfte, an meinen Sohn Laurin und meine Frau Barbara. Letzterer genau wie den Korrektor und Lektorin des Hanser-Verlags vielen Dank für die Anregungen und Überarbeitungen.

Auch für die Zukunft freue ich mich auf jede Rückmeldung an joerg@frochte.de – und nun los!

wenn Sie das Buch lesen, schon nicht mehr. Durch die Anwendung von `argmax` stellen wir fest, welche Klasse die höchste Wahrscheinlichkeit hat und entsprechend als Vorhersage gilt.

```
testGenerator.reset()
yP = CNN.predict_generator(testGenerator, steps=len(testGenerator), verbose=True)
yPClass = np.argmax(yP,axis=1)
```

`testGenerator.classes` enthält die Labels gemäß den Verzeichnissen. Hiermit finden wir heraus viele Hunde und Katzen denn genau in unserer Testmenge sind und abschließend schauen wir danach, wie oft es mit der Klassifikation nicht funktioniert hat.

```
cats = np.sum(testGenerator.classes == 0)
dogs = np.sum(testGenerator.classes == 1)
catsAsDogs = np.sum( np.abs(yPClass[testGenerator.classes == 0] -0) )
dogsAsCats = np.sum( np.abs(yPClass[testGenerator.classes == 1] -1) )
confMatrix = np.array([[(cats-catsAsDogs)/cats, catsAsDogs/cats],
                       [dogsAsCats/dogs, (dogs-dogsAsCats)/dogs]])
print(confMatrix)
```

Als Konfusionsmatrix erhalte ich:

	y_P Cat	y_P Dog
y_T Cat	85.08	14.92
y_T Dog	06.07	93.93

Es kann sein, dass Sie wegen der zufälligen Initialisierung eine andere Verteilung bekommen. Fall Sie genau mit einem Netz weiterarbeiten wollen, laden Sie sich das ZIP-File von einer Webseite. Das hd5 dieses Netzes ist ebenfalls enthalten.

Der Klassifikator ist also wesentlich besser, wenn es um Hunde geht als um Katzen. Oder um genauer zu sein: Im Zweifelsfall tendiert er dazu etwas als Hund zu klassifizieren. Im nächsten Abschnitt versuchen wir unseren Klassifikator noch etwas besser zu verstehen.

11.4 Class Activation Maps und Grad-CAM

Wenn man ein CNN verwendet, um zum Beispiel eine Klassifizierung vorzunehmen, geht schon mal was schief. Manchmal sind es Dinge, die man algorithmisch lösen kann. Also zum Beispiel, indem man die Architektur des Netzes verändert oder die Optimierer. Nicht selten sind es aber einfach die Daten, auf die man aufsetzt. Ein schönes Beispiel stammt aus dem Paper [RSG16]. Hier sollten u. a. Wölfe und Huskys unterschieden werden. Die Bilder der Wölfe hatten jedoch immer Schnee im Hintergrund, während die Bilder der Huskys keinen Schnee im Hintergrund hatten. Auf einem Testset, welches nach dem gleichen Prinzip aufgebaut ist, funktioniert die Klassifikation hervorragend. Wird jedoch ein Husky mit Schnee im Hintergrund gezeigt, so wird dieser, sogar mit einer hohen durch das Netz angegebenen Genauigkeit, als Wolf klassifiziert. Das Netz hat durch einen Bias in den Daten zwar hohe Genauigkeiten, ist aber für viele Einsätze sinnlos; ein Dackel im Schnee ist weitab von einem Wolf.

Während es sich bei [RSG16] um ein bewusstes Experiment handelte, kommt das Problem auch in wesentlich kritischeren Anwendungen vor. Das Problem wird klarer, wenn man sich

die Arbeiten um das Paper [EKN+17] vor Augen führt. Hier geht es darum, Flecken auf der Haut als Hautkrebs – sogar verschiedene Arten – und harmlose Artefakte zu identifizieren. Das Paper hat einen sehr großen Eindruck gemacht und wurde tausendfach zitiert. Die Aussage war vereinfacht, dass die KI bzw. AI besser war als die meisten Ärzte und sich bald vermutlich jeder einfach sein Smartphone schnappen könnte, um damit schnell herauszufinden, ob es ein harmloses Muttermal oder Hautkrebs ist. Die Begeisterung macht es schwer, im Netz bzw. den Medien die eher kritischen Stimmen ausfindig zu machen. Der Datensatz weist nämlich einige Probleme auf, wie u. a. in [NKS+18] thematisiert wird. Die Bilder des Trainingssets enthalten teilweise Markierungen, um die Größen der Abbildung besser einschätzen zu können. Das Problem, auf das auch in [NKS+18] hingewiesen wird, ist, dass in dem Trainingsdatensatz Bilder mit Größenmarkierungen eher bösartige Hautveränderungen beinhalten, sodass es nah liegt, dass der Algorithmus unbeabsichtigterweise lernt, dass die Markierungen für bösartige Veränderungen stehen.

Häufig können Artefakte in der Datenerfassung wie der Schnee oder die Markierungen auf medizinischen Bildern unerwünschte Korrelationen hervorrufen, die die Klassifikatoren während des Trainings aufgreifen und nicht im Sinne der späteren Anwendung nutzen.

Diese Probleme können allein durch die Betrachtung der Rohdaten und Vorhersagen sehr schwer zu identifizieren sein. Das gilt besonders, weil man es hier mit unstrukturierten Daten zu tun hat, bei denen der Klassifikator selber auch die Merkmale generiert. Die Forschung auf dem Gebiet ist sehr aktiv und es gibt vielversprechende Ansätze. Ich möchte hier ein Beispiel für eine Methode präsentieren, die in ihrer Leistungsfähigkeit zwar begrenzt ist, aber einen ersten Einblick liefert. Außerdem erlaubt uns die Beschäftigung noch einmal mehr über CNN zu lernen. Man geht bei CNN davon aus, dass die erzeugten Merkmale an Abstraktion zunehmen. Was bedeutet das? Man weiß, dass der erste Layer im Wesentlichen recht einfache Muster auf der Basis von Kanten oder Gradienten erkennt. Es sind eher einfache Filter wie in Abbildung 11.11 auf Seite 367, welche die Voraussetzung für das weitere Lernen abstrakter Dinge schaffen. Die Hoffnung ist, dass die nächsten Layer sich dann von den Pixeln wegbewegen und eben ein Konzept wie z. B. Auge oder Schwanz erkennen. Optimalerweise gibt es am Ende nach unserer letzten Faltung dann Merkmale, die quasi für *Katzenaugen* oder *Pfote ohne Krallen* stehen. Es ist aber keineswegs sichergestellt, dass diese abstrakten Konzepte irgendetwas mit dem zu tun haben, was wir Menschen als ein abstraktes Konzept für ein Tier oder ein Ding verstehen. Sie sind mit Sicherheit in dem Sinne abstrakter als sie nicht mehr direkt auf der Pixelebene basieren.

Nimmt man an, dass der eine Filter am Schluss primär für einen Hunde- oder Katzenschwanz bzw. Katzenaugen oder Pfoten ohne Klauen steht, dann kommt man schnell hinter die Grundidee aus dem Paper [ZKL+16]. In diesem Paper wird ein Ansatz mit dem Namen **Global Average Pooling**, kurz **GAP**, propagiert. Der Name ist Programm und der bedeutet nichts anderes als Mittelwertbildung jeder Feature-Map. Statt eines Übergangs mittels Flatten in ein nachgelagertes dichtes Netz wird ein **GAP-Layer** verwendet; eben der Ansatz, den wir schon als Alternative zum Max-Pooling kennengelernt haben. Nimmt man an, dass ein Netz 14 Feature Maps hat, besteht der GAP-Layer aus deren Mittelwerten. Das bedeutet aber auch, dass man, um diese Technik anwenden zu können, eine spezielle Netzarchitektur benötigt. Hinter der Merkmalsgenerierung durch Faltung und Pooling darf nur noch ein GAP-Layer und ein vollverbundener Output-Layer folgen. Es erfolgt kein Flattening. Das Netz muss mit der in Abbildung 11.17 dargestellten Architektur trainiert werden, bevor man diese nutzen kann, um zu visualisieren, welche Aspekte im ursprünglichen Bild am meisten zu einer Klassifizierung bei-

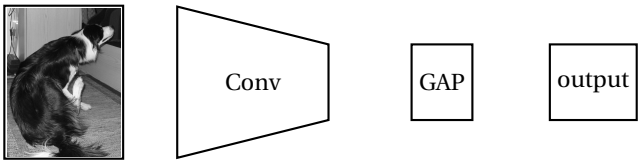

Abbildung 11.17 Netzwerkarchitektur mit GAP-Layer

getragen haben. Diese Darstellung, die wir erhalten wollen, nennt man **Class Activation Maps**. Es ist eine skalare Bitmap mit Werten, die den Grad der Aktivierung angeben.

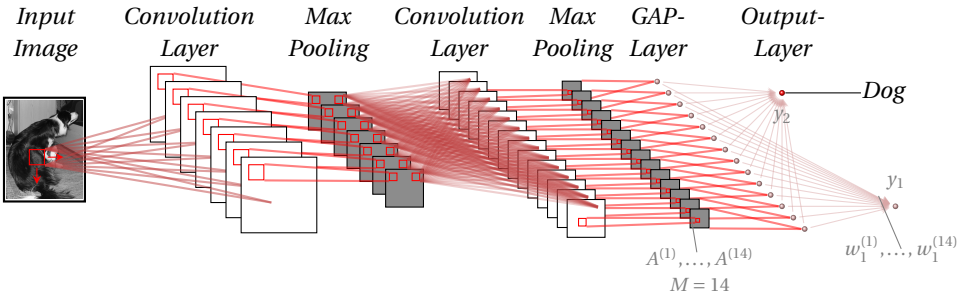

Abbildung 11.18 Detail-Netzwerkarchitektur mit GAP-Layer

Der erste Schritt ist das Bilden des Mittelwertes über eine Feature Map $A^{(m)}$:

$$\text{Mittelwert} = \sum_{i,j}^{I,J} A_{i,j}^{(m)} \tag{11.9}$$

Anschließend werden diese mit den Gewichten $w_c^{(m)}$ multipliziert. c ist dabei der Index für die Verbundene Klasse, während m für die Nummer der Feature Map steht. Die Entscheidung für eine Klasse geschieht dann gemäß der Gleichung (11.10).

$$y_c = \sum_{m=1}^{M} \left(\frac{1}{I \cdot J} \sum_{i,j}^{I,J} A_{i,j}^{(m)} \right) \cdot w_c^{(m)} \tag{11.10}$$

Das ist bis jetzt nur eine etwas andere Art, die Klassifikation durchzuführen. Wie hängt dies nun mit der Visualisierung der Aktivierung zusammen. Dazu stellen wir die Gleichung (11.10) ein wenig um.

$$y_c = \frac{1}{I \cdot J} \sum_{i,j}^{I,J} \underbrace{\left(\sum_{m=1}^{M} A_{i,j}^{(m)} \cdot w_c^{(m)} \right)}_{\text{CAM}} = \frac{1}{I \cdot J} \sum_{i,j}^{I,J} \underbrace{\left(\sum_{m=1}^{M} A^{(m)} \cdot w_c^{(m)} \right)}_{\text{CAM}} \frac{1}{I \cdot J} \sum_{i,j}^{I,J} \text{CAM}_{i,j} \tag{11.11}$$

Hierbei vertauschen wir die Summen über die Dimensionen I und J der Feature Map und die Summe über alle Feature Maps. In der inneren Klammer von Gleichung (11.11) erhalten wir nun eine Matrix, welche an den Stellen – über alle Feature Maps hinweg gemittelt – große Einträge hat, wo eine starke Aktivierung für die Entscheidung *Klasse c* vorliegt. Der Ausdruck *gemittelt* ergibt sich wegen der Division durch $I \cdot J$, also der Größe der Matrix. Die Darstellung

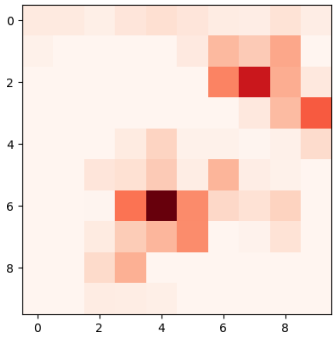

Abbildung 11.19 Beispielhafte Darstellung einer Class Activation Map (CAM)

$A^{(m)}$ soll deutlich machen, dass es eben Matrizen sind, die mit $w_c^{(m)}$ gewichtet und anschließend durch die Summation überlagert werden.

Diese CAM-Matrix können wir nun verwenden, um im ursprünglichen Bild diese Stellen für einen Menschen sichtbar zu machen. Dazu muss man sich aber vor Augen führen, dass die Feature Map durch die Pooling-Schritte und den Verschnitt am Rand – es sei denn, es wurde durchgängig ein Padding angewendet – deutlich kleiner geworden ist. Daher ist es üblich, diese eine Class Activation Map, wie sie beispielhaft in Abbildung 11.19 gezeigt wird, hochzuskalieren und diese skalierten Werte mit dem ursprünglichen Bild zu überlagern.

Das Problem ist, dass man mit dem GAP-Ansatz in der Regel schlechtere Ergebnisse erzielt als mit einem dichten Netz. Wir würden also bei gleich komplexen Modellen mehr Transparenz gegen weniger Genauigkeit tauschen. Zum Glück wurde ein Jahr nach dem GAP-Ansatz noch ein weiteres Paper publiziert. In [SCD+17] wird ein Ansatz vorgestellt, mit dessen Hilfe man versucht, die alten Architekturen weiterverwenden zu können und gleichzeitig doch mehr Transparenz zu erhalten. Das Ergebnis ist insofern sehr angenehm, als es auf viele Netze angewendet werden kann. Die Zuordnung zu den Aktivierungen für die Klassenentscheidung ist aber weniger gradlinig und an einigen Stellen eher heuristisch motiviert.

Als Erstes konzentrieren wir uns auf den Term $A^{(m)} \cdot w_c^{(m)}$ in Gleichung (11.11). Die Feature Maps existieren natürlich auch bei der klassischen CNN-Architektur, aber statt $w_c^{(m)}$ haben wir einen wesentlich komplexeren Zusammenhang, der durch ein neuronales Netz gegeben ist. Hier nimmt man eine Anleihe bei der Backpropagation, bei der man auch für die Gewichtsänderung versucht, die Einflüsse durch die Gradienten zu motivieren. Allerdings ist in diesem Fall nicht die ganze Feature Map mit dem Output-Neuron y_c verknüpft, sondern einzeln Einträge der Feature Map, welche in einen Vektor durch Flattening umformatiert werden.

$$A^{(m)} \cdot w_c^{(m)} \rightsquigarrow A^{(m)} \cdot \underbrace{\left(\sum_{i,j}^{I,J} \frac{\partial y_c}{\partial A_{i,j}^{(m)}} \right)}_{\text{Summierter Gradient}}$$

Nimmt man an, dass ein Netz vorliegt, in dem die Zusammenhänge linear sind und zuvor ein Average Pooling durchgeführt wurde, ist damit der Ansatz über einen GAP-Layer ein Spezialfall dieser Art, Gewichte zu berechnen.

Die gesamte Formel aus [SCD+17] lautet:

$$\text{ReLU}\left(\sum_{m=1}^{M}\left(\underbrace{\frac{1}{I\cdot J}\sum_{i,j}^{I,J}\frac{\partial y_c}{\partial A_{i,j}^{(m)}}}_{\text{Gemittelter Gradient}}\right)\cdot A^{(m)}\right)=\text{ReLU}\left(\sum_{m=1}^{M} w_c^{(m)}\cdot A^{(m)}\right) \tag{11.12}$$

In (11.12) tritt der gemittelte Gradient an die Stelle des Gewichtes für die Feature Map beim GAP-Ansatz. Anschließend verhindert die ReLU-Funktion, dass negative Werte auftreten können. Das ist einer der Aspekte, den ich mit *heuristisch* vorab meinte. Es liegt die Idee zugrunde, dass negative Gewichte beim ursprünglichen GAP-Verfahren eher ausdrücken wollen; *gehört nicht zu der Klasse* und positive *gehört zu der Klasse*. Daneben haben die Autoren des Grad-CAM rumprobiert – und was positiv ist diese Tests auch in die Veröffentlichung aufgenommen – und bei Ihnen klappte die Darstellung mit ReLU besser. Das ist aber kein mathematischer Beweis, nicht einmal für einen Spezialfall.

Diese Formel werden wir nun einmal versuchen umzusetzen. Dazu müssen wir Gradienten berechnen, und das wollen wir gerne automatisch berechnen lassen. Hierzu bietet TensorFlow – nicht Keras – eine geeignete Funktionalität. Entsprechend müssen wir TensorFlow selbst einbinden. Darüber hinaus soll der Gradient nicht über das ganze Modell berechnet werden, sondern nur von den Feature Maps bis zu den Output-Neuronen. Hierzu müssen wir quasi ein Teilmodell bilden. Um dieses Teilmodell zu erzeugen, nutzen wir ausnahmsweise einen Aspekt der **Model Class API** von Keras statt der Sequential. Entsprechend importieren wir die Klasse **Model**.

```
80
81  import tensorflow as tf
82  from tensorflow.keras.models import Model
83  def heatmap(img, model):
84      for l in model.layers:
85          if isinstance(l, Conv2D): lastConvLayer = l
86      calcFeaturesAndPred = Model([model.input], [lastConvLayer.output, model.output])
```

Als Erstes Suchen wir in Zeile 85 den letzten Layer vom Typ `Conv2D`, also die letzte Faltung. Liegt dahinter noch ein Pooling, wird dies in die Gradientenbildung einbezogen. In Zeile 86 bauen wir uns mit der Model-API ein neues Modell auf der Basis unseres bereits trainierten Modells. Der Unterschied ist hier lediglich, dass wir zwei Outputs festlegen, nämlich die bekannten Output des Softmax und zusätzlich den Output des letzten Faltungslayers `lastConvLayer.output`.

Nun benötigen wir nach langer Zeit das in Abschnitt 3.5 erwähnte `with`-Statement. Es ist die in TensorFlow vorgesehene Methode, um bei der parallelen Ausführung mit Ressourcenkonflikten und der Bereinigung umzugehen. Mit diesem `with`-Statement verknüpft ist hier `tf.GradientTape`. Es ist Teil der TensorFlow API zur automatischen Differenzierung, also der Berechung des Gradienten einer Funktion in Bezug auf die Eingabevariablen. Die Bezeichnung `tape` kommt daher, dass alle Operationen, die im Kontext eines `tf.GradientTape` ausgeführt werden quasi auf ein *Band* (eng. Tape) aufzeichnet. TensorFlow verwendet anschließend dieses Band bzw. Tape, um die Gradienten, die mit jeder aufgezeichneten Operation verbunden sind zu berechnen. Das funktioniert aber lediglich mit TensorFlow-Variablen. Entsprechend wandeln wir in Zeile 87 unser Bild in eine solche Variable um. Mit diesem Typ kann man fast genauso arbeiten wie mit NumPy-Arrays. In Zeile 89 müssen wir wieder eine leere Dimension hinzufügen, da wir eben nur ein Bild durch die Verarbeitung verfolgen wollen. Die Rückga-

bewerte unserer konstruierten Funktion sind jetzt wie gewünscht Feature Maps und die Vorhersagen. Wir sind nur daran interessiert, warum sich das Netz für die vorhergesagte Klasse entschiedenen hat, daher ermitteln wir deren Index (Zeile 90) und nutzen diesen anschließend, um den zugehörigen Outputwert zu ermitteln. Die Feature Maps haben vier Achsen. Die erste ist unsere künstliche Dimension, die wir hinzugefügt haben, da wir nur ein Bild haben. Die nächsten beiden stehen für die Größe der Map und die letzte für die Anzahl der Feature Maps.

```
87      img = tf.Variable(img);
88      with tf.GradientTape() as tape:
89          featureMaps, predictions = calcFeaturesAndPred(img[np.newaxis,...])
90          maxActivation = np.argmax(predictions[0])
91          predictedClassEntry = predictions[:, maxActivation]
```

Nachdem die Berechnungen ausgeführt sind, nutzen wir nun in Zeile 93 die auf dem Tape aufgezeichneten Daten, um die Gradienten aus Gleichung (11.12) zu berechnen. Die Syntax ist so, dass das erste Argument die Variable ist, die differenziert werden soll, und das zweite di,e nach der differenziert wird. In Zeile 94 bilden wir den Mittelwert für jede Feature Map. Hierbei ist es hilfreich, das `axis`-Argument gleich einem Tuple zu setzen. Die Werte $(0,1,2)$ bedeuten, dass der Mittelwert über eben diese Achsen gebildet wird. Der Gradient hat die Dimensionen: *(Anzahl von Datensätzen, x-Auflösung der Feature Map, y-Auflösung der Feature Map, Anzahl der Feature Maps)*. Die erste Dimension ist bekanntlich unsere künstliche, da wir nur ein Bild haben. Wird also über die ersten drei Achsen gemittelt, erhalten wir so viele Werte, wie wir Feature Maps haben, bzw. so viele Werte, wie die letzte Dimension angibt.

```
92
93      grads = tape.gradient(predictedClassEntry, featureMaps)
94      pooledGrads = np.mean(grads, axis=(0, 1, 2))
```

Nun multiplizieren wir gemäß Gleichung (11.12) die Feature Maps mit den Gewichten in Zeile 95 und erhalten $w_c^{(m)} \cdot A^{(m)}$. Anschließend bilden wir in Zeile 96 die Summe über alle Feature Maps.

```
95      weightedFeatures = featureMaps * pooledGrads
96      heatmapImg = np.sum(weightedFeatures, axis=-1).squeeze()
```

Am Schluss wenden wir ReLU an, was faktisch durch `np.maximum` geschieht. In den letzten zwei Zeilen der Funktionen geben wir die Werte zurück. Da wir aber die Heatmap als Bild weiterverarbeiten wollen, normieren wir diese zuvor vor. Sinnvolle Ansätze sind da auf den Bereich 0 bis 1 oder bis 255. Durch den Einsatz von Tape ist `predictions` eine TensorFlow-Variable. Mit der Methode `numpy` wandeln wir diese wieder um.

```
98      heatmapImg = np.maximum(heatmapImg, 0)
99      if np.max(heatmapImg) > 0 : heatmapImg /= np.max(heatmapImg)
100     return heatmapImg, predictions.numpy()
```

Jetzt haben wir eine Class Activation Map, müssen diese aber noch über unser ursprüngliches Bild legen. Hierzu müssen wir das Bild skalieren und ein paar weitere Anpassungen vornehmen. Da dies jetzt eher technisch ist, versuche ich kurz alles zusammenzufassen. `cm.Reds` wandelt als Klasse skalare Daten in eine RGBA-Darstellung gemäß der Colormap um. Rottöne passen hier schön zu dem Namen Heatmap, aber natürlich geht alles, was sequenziell ist,

und Ihren Geschmack trifft. Den Alpha-Kanal brauchen wir nicht, deshalb unterdrücken wir diesen mithilfe von `[...,:3]`. In PIL gibt es eine optisch schöne Methode zum Skalieren. Um diese nutzen zu können konvertieren wir etwas hin und her. Die Methoden dazu kommen aus der **image**-Klasse von Keras.

```
from matplotlib import cm
from PIL import Image as PILImage
from tensorflow.keras.preprocessing import image

def overlayHeatmap(img, heatmapImg):
    heatmapImg = cm.Reds(heatmapImg)[..., :3]
    heatmapImg = image.array_to_img(heatmapImg)
    heatmapImg = heatmapImg.resize(img.shape[:-1], resample=PILImage.BICUBIC)
    heatmapImg = image.img_to_array(heatmapImg)
```

Die nächsten drei Zeilen sind nur da, falls Sie die gleiche Ausgabe haben wollen wie im Buch. Eigentlich ist es in bunt natürlich etwas hübscher.

```

    imGray = 0.2989*img[:,:,0] + 0.5870*img[:,:,1] + 0.1140*img[:,:,2]
    imGray = cm.gray(imGray)[..., :3]
```

Nun geht es daran, die Heat bzw. Class Activation Map mit dem Bild zu überlagern. Das passiert einfach per Addition. Die Gewichte sorgen dafür, dass die Class Activation Map dominant ist und das Bild nur leicht im Hintergrund sichtbar ist. Wenn Sie es gerne anders haben wollen, ändern Sie es halt. Die beiden Zeilen bewirken, dass beide Bilder gleich skaliert sind, auch wenn sich das Verhalten der Konvertierungsmethoden mal ändert oder nicht gleich ist.

```

    if np.max(imGray) <= 1: imGray = 255*imGray
    if np.max(heatmapImg) <= 1 : heatmapImg = 255*heatmapImg
    superimposedImg = np.minimum(heatmapImg * 0.6 + 0.2*imGray, 255).astype(np.uint8)
    return superimposedImg
```

Nun wollen wir das alles auch mal ausprobieren. Dazu greifen wir auf ein paar Bilder aus meinem privaten Fundus zurück. Diese sind im ZIP-File auf der Webseite enthalten. Sie können aber auch Bilder aus der Testmenge auswählen. Diese enthält aber noch eine kleine Gemeinheit, die ich mit Ihnen besprechen möchte. Ich nehme nämlich auch drei Bilder, die weder Katzen noch Hunde enthalten. Darüber hinaus noch eine freigestellte Katze, auf die wir später kommen.

```

import matplotlib.pyplot as plt
plt.rcParams.update({'figure.max_open_warning': 0})
imageList = ['hund1.jpg', 'hund2.jpg','hund3.jpg', 'hund4.jpg',
             'katze1.jpg', 'katze2.jpg', 'katze3.jpg', 'katze4.jpg',
             'ratte.jpg', 'luchs.jpg', 'wolf.jpg', 'katze1freigestellt.jpg']
```

Zunächst laden wir in einer Schleife alle Bilder und skalieren diese direkt beim Laden auf die Größe, die das CNN verarbeiten kann. Unser Netz ist trainiert, Voraussagen für Daten zu treffen, die auf den Bereich 0 bis 1 normiert wurden, entsprechend normieren wir das Bild bzgl. der Wert für die weiteren Schritte.

Die Vorhersage muss immer als Pipeline gedacht werden. Daten werden geladen, vorverarbeitet und dann die Prognose durchgeführt. Wenn die Quelle wechselt ist das eine ganz unangenehme Fehlerquelle. Vielleicht waren vorher Daten immer schon auf [0,1] normiert und es erfolgte daher keine Normierung im Code. Wechselt die Quelle, bekommt man auf einmal z. B. Daten im Bereich [0,255]. Da die Bilddimensionen stimmen, gibt es keine Fehlermeldung, aber völlig sinnlose Vorhersagen. Solche Fehler zu finden, kostet manchmal Stunden!

```
125  for imageFile in imageList:
126      imgSize = CNN.input_shape[1:-1]
127      img = image.load_img(imageFile, target_size=imgSize)
128      img = image.img_to_array(img)/255.0
129      plt.figure(); plt.imshow(img)
```

Nun erstellen wir mit unserer Funktion die Heatmap und lassen uns gleichzeitig über die Vorhersage informieren.

```
130      hm, predictions = heatmap(img, CNN)
131      plt.figure(); plt.imshow(hm, cmap=cm.Reds)
132      if np.argmax(predictions[0]) == 0: classstring = 'cat'
133      else: classstring = 'dog'
134      print(imageFile,' predicted as ',classstring, ' with ', predictions)
```

Anschließend überlagern wir die Heatmap mit dem Original und speichern das Ergebnis als PNG, um es z. B. in einem Buch abdrucken zu können.

```
135      fusion = overlayHeatmap(img, hm)
136      plt.figure(); plt.imshow(fusion)
137      name = imageFile.split('.')[0]
138      plt.title(name+str(predictions))
139      name = 'heat'+name+'.png'
140      image.array_to_img(fusion).save(name, 'PNG')
```

Den Code oben kann man schlecht wiederverwenden. Kopieren Sie die beiden Funktionen `heatmap` und `overlayHeatmap` inklusive aller benötigten Einbindungen einmal in eine Datei `gradCAM.py`. Dann können wir diese später noch benutzen.

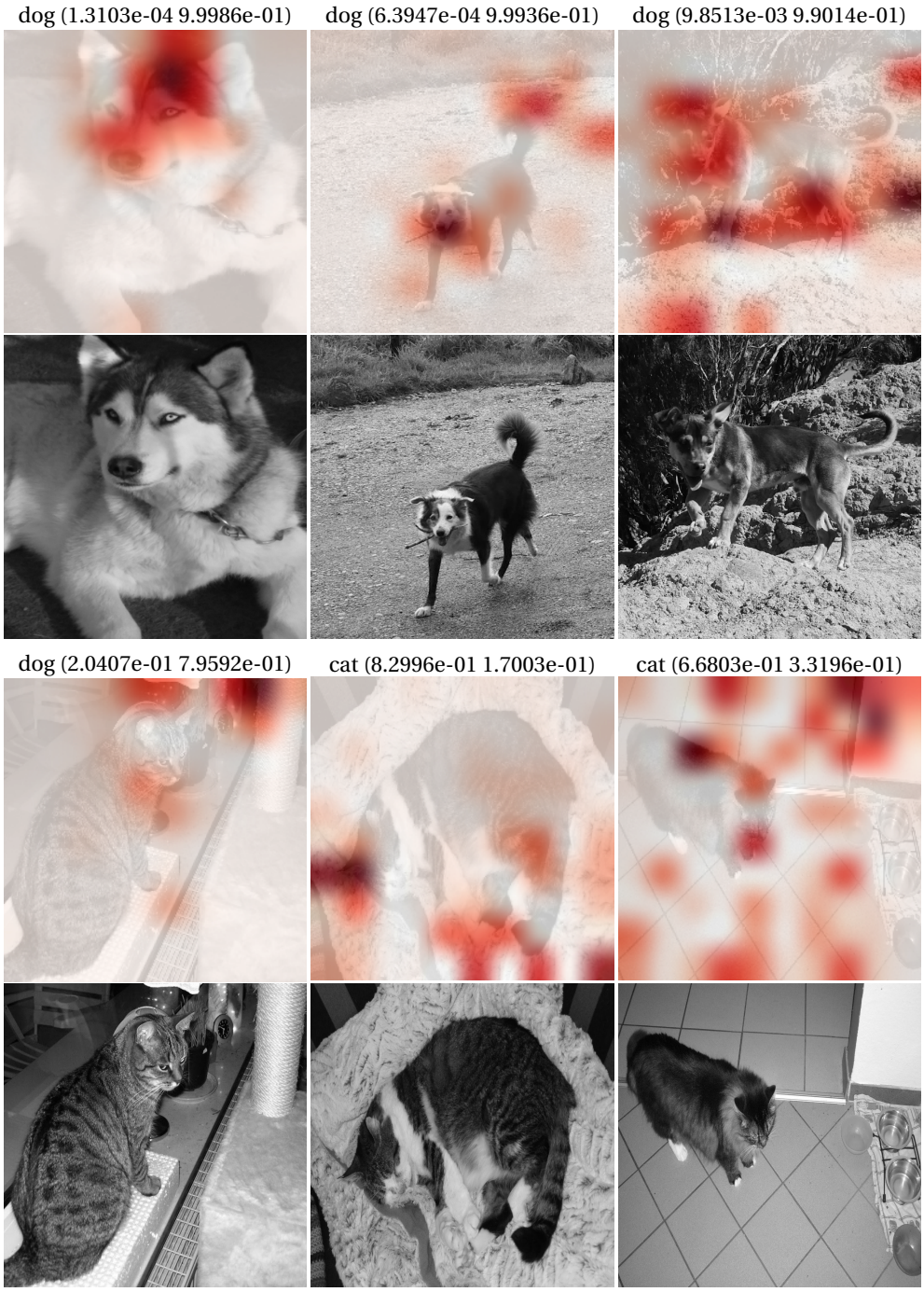

Abbildung 11.20 Heatmaps und Vorhersagen für Bilder mit Hunden und Katzen

Nun schauen wir mal, was wir rausbekommen haben. Wir wussten aus Abschnitt 11.3 schon, dass unser Netz ein Hundefreund ist und im Zweifel eher dazu tendiert, eine Katze zum Hund zu machen. Abbildung 11.20 zeigt die Prognosen und die Heatmaps für drei Hundebilder. Alle drei sind richtig als Hund klassifiziert und das auch mit einer hohen Sicherheit von Seiten des CNN. Man sieht, dass sich das Netz auf Schnauzen und Schwänze zu konzentrieren scheint. Das dritte Bild ist aber verdächtig. Es wurden viele Bereiche im Bild berücksichtigt, auf denen kein Hund zu sehen ist. Entweder sind mehr Hundebilder in freier Wildbahn im Archiv oder das Netz lässt sich von Strukturen im Hintergrund leicht ablenken. Solchen Klassifizierungen sollten wir mit etwas Vorsicht begegnen. Mal sehen, wie es bei den Katzen weitergeht.

Wie man aus Abschnitt 11.3 vermuten konnte, treten die Fehler eher bei den Katzen auf. Die erste Katze wurde mit beinahe 80% als Hund klassifiziert. Wir sehen aber auch, dass ähnlich wie bei der dritten Katze für die Klassifizierung eher der Hintergrund den Ausschlag gegeben hat.

Abbildung 11.21 Heatmap & Vorhersage für freigestellte Katze

Das kann legitim sein, weil man z. B. ein Kamel eher nicht in der Antarktis vermutet, aber hier erscheint es weniger verständlich. Der Hintergrund lenkt scheinbar ab. Wie wäre es wohl ausgegangen, wenn die erste Katze ohne Hintergrund hätte eingeordnet werden sollen? Für den Test habe ich diese einmal unfachmännisch freigestellt und erneut klassifizieren lassen. Abbildung 11.21 zeigt eindrucksvoll, dass sich das Netz nun wieder auf Ohren und Fellaspekte konzentriert. Die Katze wird nun auch zu 90% als Katze erkannt.

Wie sieht es mit den eingebrachten Bildern einer Farbratte, eines Luchses und eines Wolfes aus? Biologisch wäre es plausibel, wenn sich das Netz beim Luchs für Katze, beim Wolf für Hund und bzgl. der Ratte sich das Netz als knappes Ergebnis für Katze oder Hund entscheiden würde. Wie man in Abbildung 11.22 deutlich sieht, geht unser Netz aber nicht nach der Biologie, sondern nach Mustern. Beim Wolf ist noch alles, wie wir es erwarten. Es schaut auf Gesicht und Körper und sagt *Hund*. Beim Luchs lässt es sich wieder vom Hintergrund ablenken und klassifiziert ihn ebenfalls als *Hund*. Entgegen der generellen Tendenz unseres Netzes wird die Farbratte mit großer Sicherheit zur Katze. Vermutlich weil der Spielplatz der beiden Haustiere sich ähnelt. Die Ratte jedenfalls kommt in der Heatmap kaum vor.

Machen Sie sich doch mal einen Spaß daraus und schicken Sie Bilder von Tieren mit und ohne Hintergrund durch das Netz und sehen Sie sich die Ausgaben genau an. Vielleicht bekommen Sie so ein besseres Gefühl für das Verhalten des CNN.

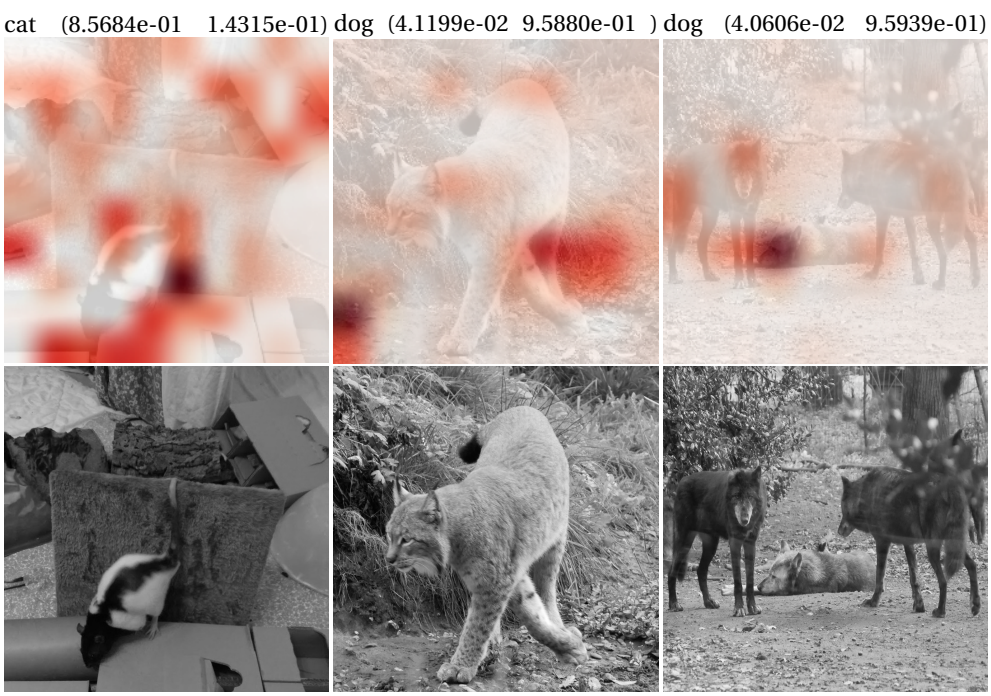

Abbildung 11.22 Heatmaps und Vorhersagen für Bilder mit nicht berücksichtigten Tieren

Nachdem die Idee einmal in der Welt war, wurde natürlich an unterschiedlichen Orten geschaut, ob man da nicht noch mehr rausholen kann. Ein Problem ist, dass die Größe der Feature Maps am Schluss automatisch die Genauigkeit der Lokalisation beeinflusst. Hätten wir noch ein paar mehr Poolings eingebaut, wäre noch weniger über gewesen, um herauszufinden, was interessant war. Ein anderes ist, dass wir händisch vorgehen müssen. Man muss Bilder als Mensch stichpunktartig analysieren und schauen, ob es einem plausibel vorkommt. Varianten wie Grad-CAM++, siehe [CSHB18], aus dem Jahr 2018 lösen diese Probleme noch nicht. Es geht eher um punktuelle Verbesserungen, um den Preis von mehr Heuristik bzw. weniger leicht nachvollziehbaren Formeln. Ich hoffe, dass Sie durch das Grad-CAM- bzw. GAP-Verfahren einen Einblick in die Möglichkeiten und Grenzen bekommen haben. Nun werden wir im nächsten Kapitel sehen, ob wir bzgl. unserer Hunde und Katzen nicht noch besser werden können und ggf. auch die starke Sensitivität für den Hintergrund etwas verringern können.

11.5 Transfer Learning

Wir haben im Abschnitt 11.3 unseren Datenbestand durch Data Augmentation aufgewertet. Dies kann auch ein wenig helfen, wenn man nur auf vergleichsweise wenig Daten zum Training zurückgreifen kann bzw. wenn es darum geht, die Erkennung unempfindlicher gegenüber Rotationen zu machen. Wie wir schon diskutiert haben, kann man hiermit jedoch nur beschränkt neue Informationen dem Netz zuführen. Ein anderer Ansatz, um ggf. mit vergleichsweise ge-

Index