Kapitel 2
Machine Learning und Deep Learning

»Wahrlich es ist nicht das Wissen, sondern das Lernen, nicht das Besitzen, sondern das Erwerben, nicht das Da-Seyn, sondern das Hinkommen, was den grössten Genuss gewährt.«
– Carl Friedrich Gauß

Das Ziel der Forschung und Entwicklung auf dem Gebiet der Künstlichen Intelligenz ist es, Systeme zu erforschen und zu entwickeln, die in einigen Bereichen mindestens die gleiche Performanz wie ein menschlicher Experte erreichen. Diese Systeme bearbeiten Aufgaben, für deren Lösung der Mensch seinen Verstand und sein Wissen benötigt, damit die so erreichte Leistung als intelligent empfunden werden kann.

2.1 Einführung

Im Gegensatz zur klassischen Vorgehensweise in der Programmierung, bei der die Verarbeitung durch Menschen komplett programmiert werden muss, beruht dieses Paradigma darauf, dass das Wissen des Menschen computergerecht aufbereitet in den Computer übertragen wird und dass der Computer eigene Schlussfolgerungen aus diesem Wissen ziehen kann (siehe Abbildung 2.1). Dieses Prinzip nennt man in der KI *wissensbasierte Verarbeitung*. Man spricht auch von einem symbolischen Ansatz. Während bei der klassischen Systementwicklung der Programmierer der »Flaschenhals« ist, weil viel Entwicklungsbedarf und eine aufwendige Adaption an neue Gegebenheiten entstehen können, zeichnen sich *wissensbasierte Systeme* durch hohen Entwicklungsbedarf und Pflegeaufwand, aber auch prinzipiell durch eine gute Erklärungsfähigkeit aus. Dabei kommt es auf eine klare Trennung an – zwischen der Repräsentation des Wissens, das für die betrachtete Anwendungsdomäne spezifisch ist (Wissensbasis), und der »Maschinerie« zur Verarbeitung dieses Wissens (Wissensverarbeitungskomponente) im Hinblick auf die Problemlösung. Wir präsentieren hierzu kurz zwei prominente Ansätze, nämlich Systeme zu regelbasiertem und zu erfahrungsbasiertem Problemlösen:

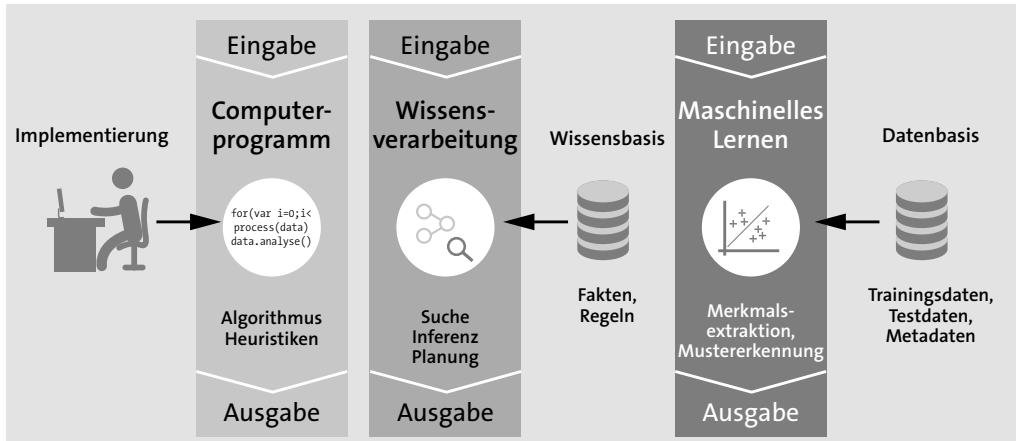

Abbildung 2.1 Computerprogramme, Wissensverarbeitung und maschinelles Lernen (Bildquelle: Grafisch adaptiert, Vortrag von Professor Wahlster – https://www.dfki.de/wwdata/Gutenberg_Stiftungsprofessur_Mainz_2017/Lernende_Maschinen.pdf)

- Bei *regelbasierten Systemen* steht die konkrete Repräsentation von generischem Wissen und dessen Verarbeitung im Vordergrund. Zudem charakterisieren sie sich durch eine klare Trennung zwischen Repräsentation und Verarbeitung von Wissen. Ein regelbasiertes System besteht aus einer Wissensbasis und einer *Inferenzmaschine* (engl. *inference engine*). Während die Wissensbasis die Datenbasis bzw. Faktenbasis und die Regelbasis beinhaltet, steuert die Inferenzmaschine durch Anwendung der Regeln (Schlussfolgerung) die Ableitung neuer Fakten für die Wissensbasis. Regeln bestehen aus *Prämissen* (Wenn-Teil) und aus *Konklusionen* (Dann-Teil).

  Ein möglicher Flaschenhals hierbei ist einerseits die Definition der Regeln, die ein tiefes Domänenwissen erfordert, und andererseits die Entwicklung von Metaregeln für den Fall, dass mehrere Regeln anwendbar sind oder sich gar widersprechen. Die in den 1980er-Jahren populär gewordenen Expertensysteme sind prominente Beispiele für diesen rein wissensbasierten KI-Ansatz, die durchaus kommerzielle Erfolge aufweisen können. Dennoch wurden die Beschränkungen auch schnell deutlich: Der manuelle Aufbau und die Pflege der Wissensbasen stellen einen limitierenden und kostentreibenden Faktor dar.
- Bei *fallbasierten Systemen* verwendet man *Case-Based Reasoning* (fallbasiertes Schließen), ein Problemlösungsparadigma, das auf der Wiederverwendung von Erfahrungswissen zur Lösung neuer Probleme basiert. Dieses Wissen bezieht sich auf konkrete Beispiele bzw. Fälle, die die gemachten Erfahrungen widerspiegeln.

Die Methodik baut auf Analogien auf und beruht auf der Annahme, dass ähnliche Probleme ähnliche Lösungen haben. Denken Sie an medizinische Anwendungen: Hier werden ständig neue Bilder der Patienten analysiert und von Ärzten annotiert. Nach einer gewissen Zeit kann das System erste Anomalien automatisch erkennen. Das *Case-based Reasoning* (CBR) folgt diesem Paradigma:

- *Retrieve* – Als Erstes wird nachgeschaut, ob ein Problem schon mit einer vorhandenen Lösung gelöst werden kann.
- *Reuse* – Die Lösungsansätze oder Algorithmen, die zu einer Lösung geführt haben, werden wiederverwendet.
- *Revise* – Gefundene Lösungen werden angepasst und verbessert.
- *Retain* – Die beste Lösung wird gespeichert.

Folglich können bereits bekannte Lösungen zu bestimmten Problemen wiederverwendet werden. Das Erfahrungswissen wird in Form von sogenannten *Fällen* (engl. *cases*) in einem Problemlösungsgedächtnis, der *Fallbasis*, abgelegt, in die jede neu gemachte Erfahrung als zusätzlich gewonnenes Wissen integriert wird. Mithilfe dieser Wissensbasis werden Inferenzen aus den vorhandenen Erfahrungen abgeleitet.

Ein weit verbreiteter Ansatz ist das *Maschinelle Lernen* (ML), ein Gebiet der Künstlichen Intelligenz, das in den letzten Jahren nicht zuletzt durch die Erfolge des Deep Learning sehr populär geworden ist. Die Grundidee des ML ist: Aus Beispielen werden z. B. Regelmäßigkeiten, Muster oder Modelle extrahiert (sprich: gelernt), mit deren Hilfe man neue Daten klassifizieren oder künftige Werte vorhersagen kann. KI-Systeme, die auf ML basieren, werden empirisch mithilfe von Beispielen bzw. Daten trainiert (siehe Abbildung 2.1). Man spricht auch von einem subsymbolischen Ansatz. Nach Beendigung dieser Lernphase entstehen *Modelle*, die die bisher gesehenen Beispiele durch Erkennung von Mustern, Beziehungen und Regelmäßigkeiten in den vorliegenden Daten verallgemeinern, sodass unbekannte Daten korrekt verarbeitet werden können. Dadurch werden neue Problemlösungen eigenständig gefunden.

Bei diesem Ansatz des Lernens durch Beispiele werden aus einer Menge von Beispielen (den sogenannten *Trainingsinstanzen*) Modelle induziert, die möglichst alle Beispiele abdecken. Hierzu werden mathematische und statistische Verfahren verwendet, um aus den Datasets zu lernen. Anwendungsbeispiele hierfür sind: Spracherkennung, Sprachübersetzung, automatische Erkennung und Sortierung von Adressen bei handbeschrifteten Briefumschlägen, Fraud Detection, automatisierte Diagnose z. B. aus CT-Aufnahmen etc.

Lazy vs. Eager Learning

CBR (*Case-based Reasoning*) wird dem Gebiet des Wissensmanagements (Knowledge Management) zugeordnet, kann aber auch als instanzbasiertes Lernen (*Instance-based Learning*) klassifiziert werden. Im Gegensatz zu Machine- bzw. Deep-Learning-Ansätzen, die zunächst in einer Offline-Phase (Trainingsphase) von den vorhandenen Trainingsdaten generalisieren bzw. lernen, bevor eine neue Instanz bearbeitet wird, verschiebt CBR die Lernaufgabe zur Anwendungsphase. Erst dann wird gegebenenfalls generalisiert und der neue Fall in die Fallbasis übernommen. Bei CBR wird kein explizites Modell gelernt. Man spricht daher im Falle von CBR von *Lazy* oder *Memory-based Learning* im Gegensatz zu *Eager Learning*. Ein weiteres Beispiel für Lazy Learner ist der k-NN-Algorithmus (*k-nächste-Nachbarn-Algorithmus*, engl. *k-nearest neighbors algorithm*) zur Mustererkennung. Man spricht auch von *parametrischen* bzw. *nicht-parametrischen Lernverfahren* (z. B. CBR, k-NN).

Wie Sie später in diesem Buch sehen werden, zeichnen sich ML-Systeme aktuell durch einen relativ geringen Entwicklungsaufwand und eine leichte Anpassbarkeit aus. Ihre Nachteile sind jedoch ein großer Datenbedarf sowie die Tatsache, dass sich schlecht erklären lässt, wie und warum das System welche Lösungen gefunden hat.

Es existieren viele ML-Verfahren (Entscheidungsbäume, Random Forest, Support Vector Machines (kurz: SVM), Boosting-Verfahren, k-Means, bayessche und genetische Algorithmen etc.). In diesem Buch legen wir unser Augenmerk auf das Deep Learning, eine Spezialform des maschinellen Lernens, die wiederum ein Teilgebiet der Künstlichen Intelligenz ist. *Deep Learning* (wörtlich »tiefes Lernen«) basiert auf sogenannten künstlichen neuronalen Netzen, die im nächsten Kapitel ausführlich vorgestellt werden, und auf der Verfügbarkeit großer Datasets sowie hoher Rechenleistung zum Trainieren komplexer neuronaler Netze. (Die Rechenleistung wird durch den zunehmenden Einsatz von GPUs zur Verfügung gestellt oder sogar von den neuerdings durch Google entwickelten TPUs, *Tensor Processing Units*. Das sind Hardware-Chips, die eine massive Parallelverarbeitung ermöglichen.) Mehr zu diesem Thema erfahren Sie in Kapitel 3, »Neuronale Netze«, und in Kapitel 5, »TensorFlow: Installation und Grundlagen«.

Das Adjektiv »deep« ergibt sich daraus, dass der zugrunde liegende Lernprozess mithilfe von tiefen (sprich: mehrschichtigen) neuronalen Netzen erfolgt. Bei den heutigen Systemen kann durchaus mit Netzen mit mehr als 100 Schichten gearbeitet werden. Durch diesen Ansatz sind in den letzten Jahren große Fortschritte erzielt worden (siehe Kapitel 1, »Einführung«), unter anderem in den Bereichen Spracherkennung,

maschinelle Übersetzung, autonomes Fahren, Computer Vision (maschinelles Sehen), radiologische Diagnostik, Industrie 4.0, Brett- und Kartenspiele etc. Sie erinnern sich sicherlich an die Schlagzeilen, als AlphaGo, ein System der Google-Tochterfirma DeepMind, die Go-Großmeister Lee Sedol 2016 und Ke Jie 2017 bei medienwirksamen Events geschlagen hat!

Die Vorgehensweise bei allen Machine- und Deep-Learning-Projekten ist im Prinzip gleich, wie prototypisch im Workflow der Abbildung 2.2 dargestellt ist:

❶ Als Erstes soll sichergestellt werden, dass Daten zur Verfügung stehen und dass diese so weit vorverarbeitet sind, dass sie für eine Modellbildung genutzt werden können. Später werden Sie sehen, wie eine Normalisierung der Daten durchgeführt werden kann und wie mit fehlenden Merkmalen in einigen Datasets umzugehen ist.

❷ Sollte das Dataset nicht in einem verwertbaren Format vorliegen, müssen eventuelle Konvertierungsschritte vorgenommen werden, z. B. die Umwandlung der Daten in ein CSV- oder JSON-Format.

❸ Stehen nicht genügend Daten zur Verfügung, ist zu klären, ob ein alternatives Dataset benutzt werden kann.

❹ Alternativ können die bestehenden Daten eventuell angereichert werden, etwa mit synthetischen Daten.

❺ Der Datenvorbereitungsprozess schließt mit einer finalen Analyse der Daten, etwa auf Integrität.

❻ Anschließend wird überprüft, ob bereits existierende Modelle für die definierte Aufgabe herangezogen werden können. Sollte dies der Fall sein, kann das Modell direkt angewendet und evaluiert werden.

❼ Wenn kein vorgefertigtes Modell frei verfügbar ist, muss auf Basis der Datenanalyse ein erstes Modell erstellt, ...

❽ ... trainiert, ...

❾ ... und evaluiert werden.

❿ Dann wird überprüft, ob das Modell hinreichend verwertbar ist.

⓫ Wenn dies nicht der Fall ist, wird das Modell in einem iterativen Prozess angepasst bzw. ersetzt.

⓬ Hat man ein akzeptables Modell generiert, wird es angewandt und für künftige ähnliche Situationen herangezogen.

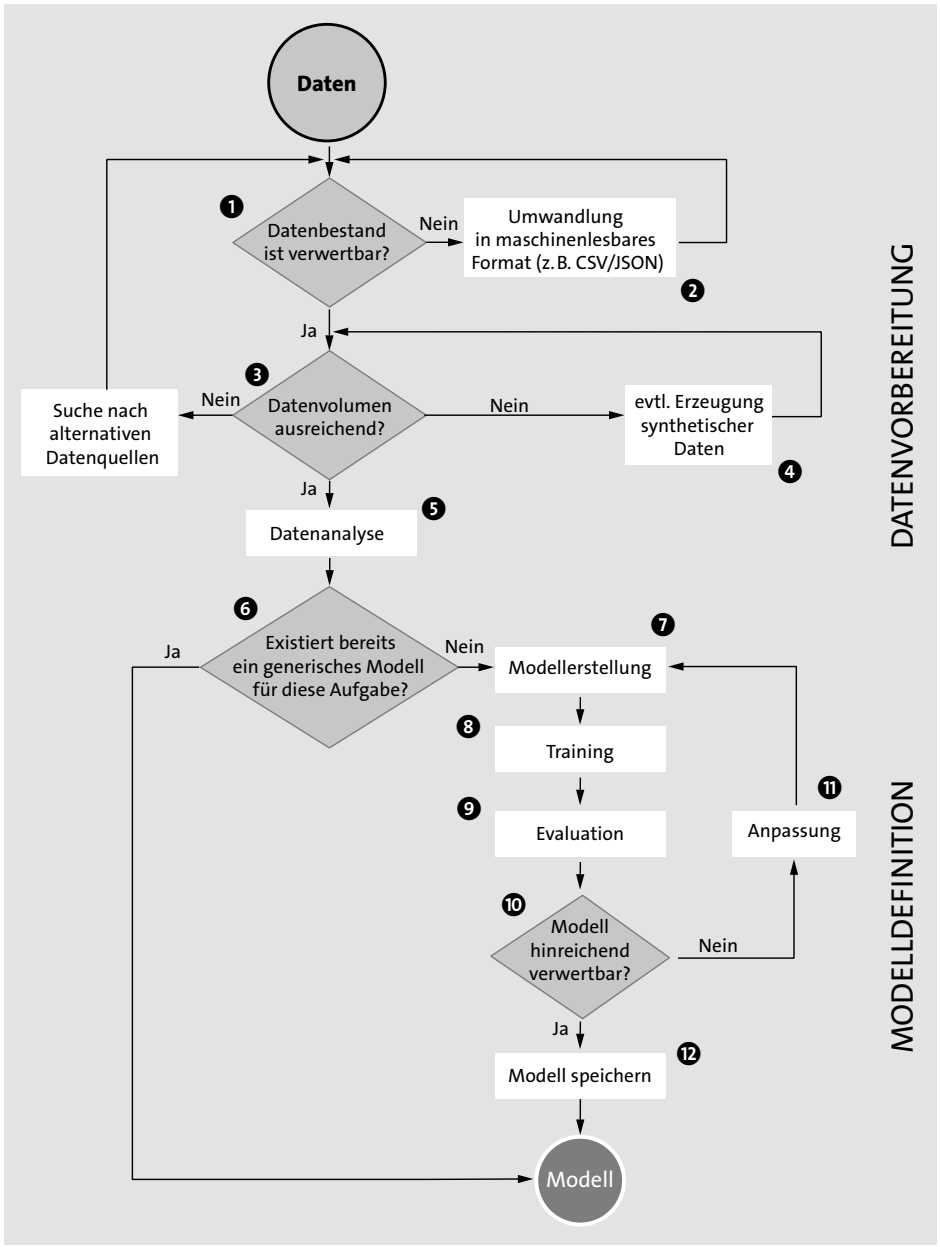

Abbildung 2.2 Ablaufdiagramm eines typischen Machine-Learning-Projekts

2.2 Lernansätze beim Machine Learning

Die Lernverfahren zur Erstellung von Machine-Learning-Modellen können entlang von zwei Dimensionen eingeordnet werden:

- **Lernmodus** – Mit welchem Ansatz soll das Modell trainiert werden? Hierbei existieren fünf Kategorien von Lernmodi, die das Lernen durch Beispiele als Grundlage unterstützen:
  - Überwachtes Lernen (Supervised Learning)
  - Unüberwachtes Lernen (Unsupervised Learning)
  - Teilüberwachtes Lernen (Semi-supervised Learning)
  - Bestärktes Lernen (Reinforcement Learning)
  - Aktives Lernen (Active Learning)
- **Typ des zu lösenden Problems**
  - Regression (Vorhersage von kontinuierlichen Werten, z. B. Preisentwicklung, Aktienkurse)
  - Klassifikation (Einordnung von Daten nach vorgegebenen Gruppenbezeichnungen [Labels], z. B. für die Personen- und Objekterkennung)
  - Clustering (Gruppierung von Daten nach nicht vorgegebenen Bezeichnungen)

Wir präsentieren in diesem Abschnitt die verschiedenen Begriffe und zeigen, welche Aufgaben mit welchem Lernansatz vorzugsweise bearbeitet werden können.

2.2.1 Supervised Learning

Beim *Supervised Learning* (dt. *überwachten Lernen*) wird einem System vorgeben, was es lernen soll, z. B. dass es einen bestimmten Gegenstand oder Sachverhalt in Bildern oder Videos erkennen soll. So kann etwa ein System lernen, Krebszellen von gesunden Zellen zu unterscheiden, Hunde von Katzen zu unterscheiden oder Verkehrsschilder zu erkennen.

Charakteristisch bei diesem Lernmodus ist, dass die präsentierten Beispiele annotiert (gelabelt) sind, also als Trainingspaare durch Eingabe mit korrespondierender Ausgabe dargestellt sind. Konkret heißt das, dass für jede Eingabe die richtige Antwort vorliegt. Ausgehend von diesen annotierten Beispielen soll nun ein Modell gelernt werden, das automatisch für jede neue präsentierte Eingabe die möglichst korrekte Ausgabe liefert. Es wird also zurückgemeldet, ob die Eingabe korrekt oder falsch bearbeitet wurde. Darüber hinaus wird auch gegebenenfalls quantifiziert, wie weit man sozusagen danebenliegt, und dies wird dem System zurückgemeldet, damit es sich korrigieren bzw. anpassen kann. Dieser Anpassungsprozess läuft so lange, bis die Beispiele in akzeptabler Weise korrekt bearbeitet werden oder bis keine Veränderung mehr zu erwarten ist. Diese schrittweise Anpassung ist das eigentliche Lernen. Dieses Lernen bezieht sich auf die Arbeitsweise des gesamten künstlichen neuronalen Netzes.

In Abhängigkeit von den betrachteten Typen der Ein-/Ausgabepaare ergeben sich drei Aufgabentypen:

- *Regression*
  Bei Regressionsaufgaben werden kontinuierliche Werte vorhergesagt, im Gegensatz zu den diskreten Werten der Klassifikationsaufgaben, die die zu erkennenden Klassen repräsentieren. Hierzu gibt es verschiedene statistische Methoden (lineare Regression, polynomiale Regression, multivariate Regression etc.), die alle darauf abzielen, die Beziehung zwischen einer (vorherzusagenden) Variablen und einer oder mehreren anderen Variablen zu analysieren. Im Kern geht es darum, eine Abbildung von unabhängigen Variablen (Einflussgrößen, auch Features genannt) auf eine abhängige Variable (Zielgröße) zu ermitteln.

  Beispiele hierfür sind z. B.:

  – die Prognose von Absatzzahlen in Abhängigkeit von wirtschaftlichen Indikatoren, saisonalen Aspekten und historischen Daten
  – die Vorhersage von Gebrauchtwagenpreisen z. B. in Abhängigkeit von Marke, Modell, Getriebeart, Kilometerstand, Beliebtheit, Datum der Erstzulassung etc.
  – die Prognose von verbrannten Flächen bei Waldbränden in Abhängigkeit von Einflussgrößen wie z. B. Windgeschwindigkeit, Jahreszeit, Temperatur und Luftfeuchtigkeit
  – die Vorhersage der elektrischen Leistung, die von Photovoltaik- oder Windkraftanlagen erzeugt wird, in Abhängigkeit von meteorologischen Daten wie Temperatur, Sonneneinstrahlung, Windgeschwindigkeit und -richtung
  – die Vorhersage des elektrischen Energieverbrauchs von privaten Haushalten oder gewerblichen Gebäuden in Abhängigkeit von Wetterdaten und gesellschaftlichen Ereignissen

  Einen Spezialfall stellt die logistische Regression dar, die auch für (binäre) Klassifikationsaufgaben genutzt werden kann. Sie ermöglicht es z. B., anhand diverser Merkmale die Wahrscheinlichkeit zu ermitteln, dass jemand Träger einer bestimmten Krankheit ist oder wird. Das ist letztlich auch eine Regression mit dem Ergebnis 0 oder 1.

- *Klassifikation*
  Hier besteht die Aufgabe darin, die Klassenzugehörigkeiten von Beispielen vorherzusagen, also die Abbildung von Merkmalen in einen diskreten Wertebereich, der die Klassen bzw. Labels repräsentiert. Die besagten Klassen sind schon bei der Trainingsphase bekannt. Beispiele von Klassifikationsaufgaben sind:

  – die Kategorisierung einer E-Mail durch den Spam-Filter als Spam oder Nicht-Spam
  – die Erkennung von Gegenständen oder Personen auf Videos

  – die Vorhersage einer potenziellen Kündigung eines Vertragsverhältnisses durch einen Kunden (*Churn*)
  – die Kreditausfallvorhersage

- *Zeitreihen*
  Bei Zeitreihen werden künftige Werte vorhergesagt, und zwar auf Basis von vergangenen Werten. Eine Zeitreihe ist eine zeitlich geordnete Reihe von Werten, z. B. Aktienkursen oder Preisen auf dem Energiemarkt. Da Zeitreihen nicht zufällig entstehen, sondern die Historie aus den vergangenen Prozessen reflektieren, geht man davon aus, dass ihnen bestimmte Strukturen inhärent sind, die etwa für die Wettervorhersage oder Aktien-, Preis- und Trendvorhersagen genutzt werden können. Die Zeitreihenanalyse ist im Grunde eine Form der oben beschriebenen Regression.

2.2.2 Unsupervised Learning

Beim *Unsupervised Learning* (dt. *unüberwachten Lernen*) geht es darum, die vorhandenen Daten zu explorieren und die inhärenten strukturellen Eigenschaften bzw. Ähnlichkeiten zu identifizieren. Dabei sind die Beispiele nicht vorannotiert. Dies ist oft der Fall bei großen, unstrukturierten Datenmengen. Das System muss dann ohne vorher definierte Ausgabewerte die präsentierten Beispiele strukturieren und die vorhandenen Muster erkennen, um sie beispielsweise in homogene Gruppen bzw. Klassen einzuordnen. Dabei soll möglichst die Ähnlichkeit innerhalb der jeweiligen Gruppen maximiert und die Ähnlichkeit zwischen den gefundenen Gruppen minimiert werden. Diese Technik zum Auffinden von Ähnlichkeitsgruppen in Daten – den Clustern – nennt man *Clustering*.

Im Gegensatz zu Klassifikationsaufgaben, bei denen die Kategorien im Vorfeld bekannt sind, werden bei Clusteringaufgaben neue, nicht a priori bekannte Gruppen aus den Daten entdeckt. Ein Beispiel hierfür ist die Kunden- oder Marktsegmentierung, um marketingrelevante Zielgruppen zu identifizieren.

Gut zu wissen: Unsupervised Learning und Clustering

Oft werden Unsupervised Learning und Clustering als Synonyme betrachtet. Es gibt aber durchaus andere Verfahren, die ebenfalls dem Gebiet Unsupervised Learning zugeordnet werden können – etwa aus dem Bereich Data Mining die Assoziationsanalyse (engl. *association rule mining*), eine Methode zur Suche nach Regeln, die Korrelationen zwischen Objekten beschreiben. Beispielsweise könnte aus der Analyse von Einkaufszetteln in den Supermärkten eine Regel folgender Art aufgedeckt werden: »Wer eine große Packung Eier kauft, wird mit einer Wahrscheinlichkeit von 85 % auch (pasteurisierte) Milch kaufen.«

2.2.3 Semi-supervised Learning

Semi-supervised Learning (dt. *semi-überwachtes Lernen*) ist eine Mischform von überwachtem und unüberwachtem Lernen. Es zeichnet sich durch die gemischte Verwendung von gelabelten und nicht annotierten Datasets aus. Typischerweise liegen weniger gelabelte als ungelabelte Daten vor, da das Annotieren von Daten relativ kostspielig ist. Der Ansatz läuft meist in zwei Schritten ab: Man trainiert zuerst mit den annotierten Daten und setzt dann die nicht annotierten ein, um die Lernperformanz zu verbessern.

2.2.4 Reinforcement Learning

Beim *Reinforcement Learning* (dt. *bestärkenden Lernen*) werden die richtigen Aktionen bzw. Antworten belohnt (engl. *rewarded*) und die falschen Antworten bzw. schlechten Aktionen »bestraft«. DeepMinds faszinierende automatisierte Atari-Spiele oder Googles AlphaGo verwenden Reinforcement Learning.

Generell eignet sich Reinforcement Learning bei Systemen, deren Performanz evaluiert werden kann, ohne ihnen die richtige Antwort vorzugeben: Bei Spielen z. B. weiß man, ob jemand gewonnen hat oder nicht, auch wenn man ausgehend von bestimmten Situationen nicht weiß, welcher Spielzug zum Erfolg führt. Bei jedem Spielzug lernt das System: »Nächstes Mal in dieser Konstellation mache ich es genauso, da dieser Zug zum Ziel führt.« Der Nachteil von Reinforcement Learning ist, dass es sehr ressourcenaufwendig ist, da eine sehr große Anzahl von Versuchen (Trial-and-Error Prinzip) notwendig ist. Diese Vorgehensweise eignet sich eher in Bereichen, in denen Wiederholungen nicht kritisch sind, etwa bei Computerspielen, aber eher nicht beim autonomen Fahren.

Sehr beliebt ist das Reinforcement Learning bei Wissenschaftler/innen, um Videospiele automatisch anzusteuern und somit die höchste Punkteanzahl zu erreichen. Während des Spielens werden Tastenkombinationen ausgelöst und je nachdem, wie sich der Spielcharakter (*Avatar*) bewegt oder auf Gegenstände bzw. Gegner stößt, wird der Score ausgewertet (*Monitor*). Somit lernt das Programm (*Agent*), welche Tastenkombination in welchem Moment ausgelöst werden muss (*Policy*). Dabei entsteht ein präzises Modell des Spiels. Ein bekanntes Beispiel von Reinforcement Learning in Zusammenhang mit Videospielen ist das Projekt Mario-AI[1], das Sie in Abbildung 2.3 sehen.

AlphaGo ist ebenfalls ein prominenter Repräsentant des Reinforcement Learning. Go (siehe Abbildung 2.4) ist ein Strategiespiel, das viel komplexer als Schach ist. Bei Go sind die Kombinationsmöglichkeiten astronomisch hoch: Es existieren nämlich mehr, als es Atome im Universum gibt. In der neuesten Version *AlphaGo Zero* lernt

1 *https://github.com/aleju/mario-ai*

das Programm »from scratch« mithilfe von TensorFlow und 64 GPUs. Das initiale System, das nur die Go-Spielregeln kannte, wurde durch Reinforcement Learning einzig bei Spielen gegen sich selbst performanter! Es wurden dabei fürs Training keine Partien mit Menschen herangezogen.[2] *AlphaZero* ist die generische Weiterentwicklung von AlphaGo Zero, die weitere Brettspiele anhand der Spielregeln und durch Reinforcement Learning erlernt.

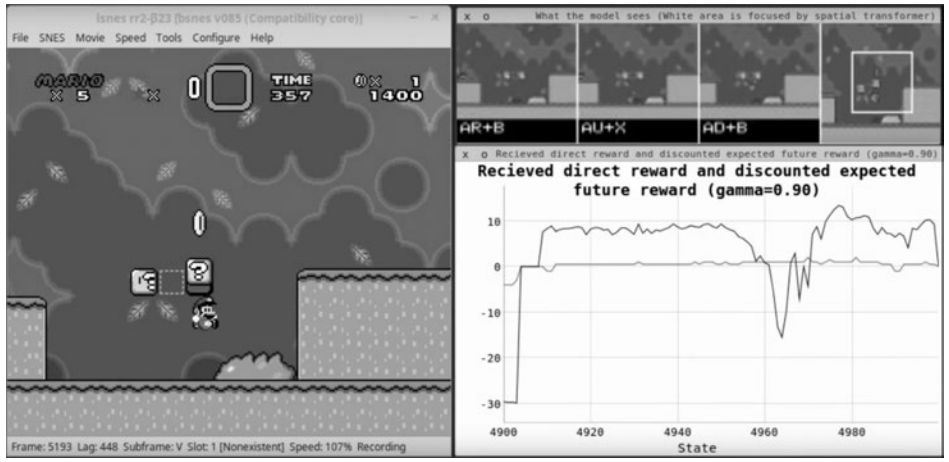

Abbildung 2.3 Bildschirmauszug von Mario-AI: Die einzelnen Pixelbereiche werden während des Spiels analysiert, und aus dieser Analyse wird ein Modell erstellt.

Abbildung 2.4 Googles AlphaGo (Bildquelle: https://www.technologyreview.com/s/604273/finding-solace-in-defeat-by-artificial-intelligence)

Inzwischen gibt es mehrere (Re-)Implementierungen von AlphaGo, so z. B. das *ELF OpenGo* von Facebook (*https://ai.facebook.com/blog/open-sourcing-new-elf-opengo-*

2 Siehe *https://www.nature.com/articles/nature24270*

bot-and-go-research sowie *https://arxiv.org/pdf/1902.04522*). Die *ELF* (Extensive, Lightweight, Flexible) *Game Research Platform* adressiert über Go hinaus eine ganze Palette von Computerspielen.

Gut zu wissen: Reinforcement Learning

Falls Sie mehr Interesse an Reinforcement Learning im Gaming-Bereich haben, sollten Sie sich auf die Seite von OpenAI begeben (*https://gym.openai.com*). Dort werden ein Framework und ein Emulator angeboten, die das Lernern vieler ATARI-Spieleklassiker ermöglichen.

2.2.5 Aktives Lernen

Der Vollständigkeit halber besprechen wir hier kurz das *Aktive Lernen*, das als interaktives semi-überwachtes Lernen bezeichnet werden kann. Da Datenannotierung kostspielig ist, können menschliche Benutzer eingebunden werden, um große Datenmengen zu klassifizieren.[3]

Weiterführende Literatur zum Machine Learning

In diesem Buch wird der Schwerpunkt auf Deep Learning als Teilgebiet von Machine Learning gesetzt. Wenn Sie sich mit Machine Learning beschäftigen möchten, verweisen wir gerne auf folgende Ressourcen:

- Joachim Steinwendner, Roland Schwaiger: *Neuronale Netze programmieren mit Python*. Rheinwerk Verlag, Bonn 2020 (*https://www.rheinwerk-verlag.de/neuronale-netze-programmieren-mit-python_5046/*).
- Christopher M. Bishop: *Pattern Recognition and Machine Learning. Information Science and Statistics*. Springer-Verlag, Berlin 2008.
- Richard O. Duda, Peter E. Hart, David G. Stork: *Pattern Classification.* Wiley, New York 2001.
- Nils J. Nilsson: *Introduction to Machine Learning* (*http://robotics.stanford.edu/~nilsson/mlbook.html*).

2.3 Deep-Learning-Frameworks

Zur Umsetzung der vorgestellten Lernansätze in Deep Learning werden viele implementatorische und mathematische Hilfsmittel benötigt. Aber keine Angst, Sie müssen nicht das Rad neu erfinden! Es gibt inzwischen zahlreiche Programmbibliothe-

3 Ausführliche Informationen hierzu finden Sie unter *http://burrsettles.com/pub/settles.activelearning.pdf*.

ken, die das Arbeiten mit Deep-Learning-Methoden unterstützen. Die meisten von ihnen sind Open Source und kostenlos erhältlich. Diese Bibliotheken bieten alle notwendigen Werkzeuge, um den in Abbildung 2.2 dargestellten Prozess zu unterstützen. Im Folgenden listen wir einige dieser Programmbibliotheken mit kurzen Erklärungen auf:

- **TensorFlow** (*https://www.tensorflow.org*) ist die von Google unterstützte plattformunabhängige Programmbibliothek für Machine- und Deep-Learning-Aufgaben. Wir werden TensorFlow ausführlicher in Kapitel 5 vorstellen.
- **Keras** (*https://keras.io*) baut auf TensorFlow auf und bietet eine High-Level-API, die einen leichteren Zugang ermöglicht. Inzwischen wird Keras von Google sowohl standalone betrieben als auch als Teil von TensorFlow. Keras kann ebenfalls die unten beschriebene Bibliothek Theano sowie CNTK kapseln. Keras wird ausführlich in Kapitel 6 behandelt.
- **TensorFlow.js** (*https://js.tensorflow.org*) ist eine relativ junge JavaScript-Bibliothek für das Machine Learning, die die Grundkonzepte von TensorFlow und von Keras übernimmt. Sie ermöglicht es durch eine JavaScript-basierte API, Modelle innerhalb des Webbrowsers anzutrainieren bzw. zu benutzen. Dieses Framework wird ausführlich in Kapitel 8 präsentiert.
- **Scikit-learn** (*http://scikit-learn.org*) fungiert nicht speziell als Deep-Learning-Bibliothek, sondern eher als freies Machine-Learning-Framework für die Sprache Python. Es wird als Python-Bibliothek integriert und bietet zahlreiche ML-Algorithmen wie Regression, Klassifikation und Clustering. Darüber hinaus bietet Scikit-learn Funktionalitäten für Datenvorverarbeitung, Dimensionalitätsreduktion und Modellauswahl.
- **Microsoft Cognitive Toolkit** (CNTK, *https://github.com/Microsoft/CNTK*) ist das Deep-Learning-Framework von Microsoft. Es kann mit den Programmiersprachen Python, C#, Java oder BrainScript benutzt werden. CNTK kann auch als Keras-Backend betrieben werden.
- **Caffe** (*http://caffe.berkeleyvision.org*) ist ein in C++ geschriebenes Open-Source-Deep-Learning-Framework, das sowohl mit CPUs als auch mit GPUs arbeitet. Es wurde am *Vision and Learning Center* der University of California, Berkeley, entwickelt. Caffe wird eher in der Forschungswelt eingesetzt.
- **Torch** (*http://torch.ch*) ist eine in der Skriptsprache Lua geschriebene Open-Source-Bibliothek, die unter anderem von Facebook und IBM eingesetzt wurde.[4] Darauf aufbauend wurde die Python-Library *PyTorch* (*https://pytorch.org*) von der Facebook AI Research Group entwickelt. Beide Frameworks können mit GPU-Unterstützung arbeiten.

4 Torch wird seit der Version 7 von 2017 nicht mehr aktiv weiterentwickelt.

- **Theano** (*http://www.deeplearning.net/software/theano*) ist ein von der Universität Montreal entwickeltes Open-Source-Deep-Learning-Framework. Es wurde in Python und CUDA entwickelt und hat eine GPU-Unterstützung.

Eine detaillierte Liste der zurzeit verfügbaren Deep-Learning-Frameworks finden Sie unter anderem unter:

https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software

2.4 Datenbeschaffung

Wir haben es schon erwähnt: Im Bereich Machine Learning und Deep Learning spielen Daten die zentrale Rolle.[5] Ohne Daten können keine Modelle trainiert und evaluiert werden: Somit bilden die Datasets die Grundlage des gesamten Machine-Learning-Ökosystems. Leider ist es aber oft so, dass wertvolle Daten nicht gern weitergegeben oder dem breiten Publikum zur Verfügung gestellt werden, denn es hängen sehr viele kommerzielle Interessen daran. Auf Programmierer/innen, Deep-Learning-Expert/innen und solche, die es werden wollen, wirkt das am Anfang sehr frustrierend. In den folgenden Abschnitten werden Sie erfahren, welche Möglichkeiten Sie haben, um an interessante Daten für das Training Ihrer Modelle heranzukommen.

2.4.1 Vorgefertigte Datasets

Vorgefertigte Datensets bzw. Daten (engl. *datasets*) werden frei verfügbar von Firmen, Staaten, Kommunen, Institutionen oder Communitys im Internet zum Herunterladen bereitgestellt. Meistens wurden diese Daten schon strukturiert, auf fehlende Einträge überprüft und bereinigt. Diese können dann 1:1 übernommen und für das Training der neuronalen Netze eingesetzt werden.

Trotzdem müssen Sie jedoch beachten, welche Methodik bei diesem Datensammeln benutzt wurde. Das heißt, Sie müssen die Gültigkeit der Daten, die Quelle und die Werte überprüfen. Wenn Sie versuchen, Daten aus verschiedenen Quellen zu benutzen, beachten Sie, dass diese wohlmöglich auch andere Maßeinheiten (Fahrenheit statt Celsius) und andere Zeitstempel (UTC statt GMT) besitzen können. Aus diesem Grund müssen Sie immer auf die Skala und die Normierung der Daten achten!

Überprüfen Sie ebenfalls, welche Formate zum Herunterladen angeboten werden. Formate wie CSV, JSON oder XLS können problemlos von Python-Bibliotheken geladen bzw. geparst werden (siehe Kapitel 4, »Python und Machine-Learning-Bibliotheken«).

5 Im Jahre 2016 bezeichnete Angela Merkel auf der IT-Messe CeBIT Daten als »die Rohstoffe des 21. Jahrhunderts« und andere Journalisten bezeichnen Daten sogar als »neues Öl«.

Ein weiterer Punkt, der oft vergessen wird: Wenn Sie eine bestimmte Domäne, wie z. B. Sport oder Wetterdaten, trainieren wollen, dürfen Sie nicht vergessen, Ihre Daten zu aktualisieren! Genau wie die Außenwelt sich verändert, müssen auch die Daten aktualisiert werden, ansonsten laufen Sie Gefahr, dass das trainierte Modell nicht mehr korrekt funktioniert bzw. ungenaue Ergebnisse liefert!

2.4.2 Eigene Datasets

Wie der Name schon sagt, werden diese Datasets von Ihnen bzw. Ihren intelligenten elektronischen Geräten produziert und gesammelt. Schauen Sie sich mal um, Sie werden bestimmt auch Quellen finden, die für Sie interessante Daten generieren können, z. B.:

- Stromzähler (mit einem Smartmeter oder Raspberry Pi)
- Stromproduktion (z. B. von Ihrer Photovoltaikanlage)
- Wetterdaten (z. B. von einer heimischen Wetterstation)
- GPS-Daten (z. B. von Ihrem Handy)
- MP3-Bibliothek (von Ihrer iTunes-Bibliothek)
- annotiertes Fotoalbum (z. B. von Adobe Bridge oder iPhoto)
- Jogging- und Herzfrequenz-Daten (z. B. von Ihrer Smartwatch)
- Aktienkurse

Im Business-Bereich bzw. in Ihrem Unternehmen werden auch täglich Tausende von Datasets generiert, z. B.:

- Kundendaten (z. B. Churn-Bewertung)
- Produktionsdaten
- Workflow- und Business-Process-Daten (z. B. aus einem SAP-System)
- OBD-2-Daten (z. B. aus einem Kfz-Flottenmanagementsystem)
- Daten aus Buchungssystemen
- Parkplatzdaten (z. B. um Parkplätze vorzureservieren)
- Finanzdaten
- Gesundheitsdaten

Diese kleine Auflistung zeigt, dass Daten in allen Anwendungsgebieten vorhanden sind und in unterschiedlichster Form benutzt werden können. Doch ähnlich wie die Daten, die von einer Community oder über Open Data zur Verfügung gestellt werden, müssen diese im Vorfeld jeder Analyse auf Integrität, Korrektheit und Aktualität

überprüft werden. Unvollständige gesammelte Datasets können dazu führen, dass die darauf trainierten Modelle fehlerhaft sind oder eine zu große Ungenauigkeit aufweisen. Zusätzlich muss geklärt werden, ob die Daten lizenzrechtlich und datenschutzrechtlich (Stichwort: Datenschutz-Grundverordnung) benutzt werden dürfen.

2.5 Datasets

Bevor Sie aufwendig Ihre eigenen Daten sammeln, mühsam annotieren und ein eigenes Datenformat entwerfen, werfen Sie mal einen Blick auf Open Data oder durch die Community erzeugte Daten. In diesem Abschnitt werden wir ein paar Datasets genauer betrachten, die im Internet frei verfügbar sind.

2.5.1 Kaggle

Wenn es eine Website gibt, die man als Data Scientist auf keinen Fall verpassen sollte, dann ist es Kaggle (*www.kaggle.com*). Das selbst ernannte *Home Of Data Science and Machine Learning* bietet über 10.000 frei verfügbare Datasets (mit Creative-Commons-Lizenz) in den Formaten (CSV oder JSON) an. Abbildung 2.5 zeigt einige von ihnen. Teilweise werden zusätzlich Quellcode bzw. Modelle für Keras oder TensorFlow angeboten.

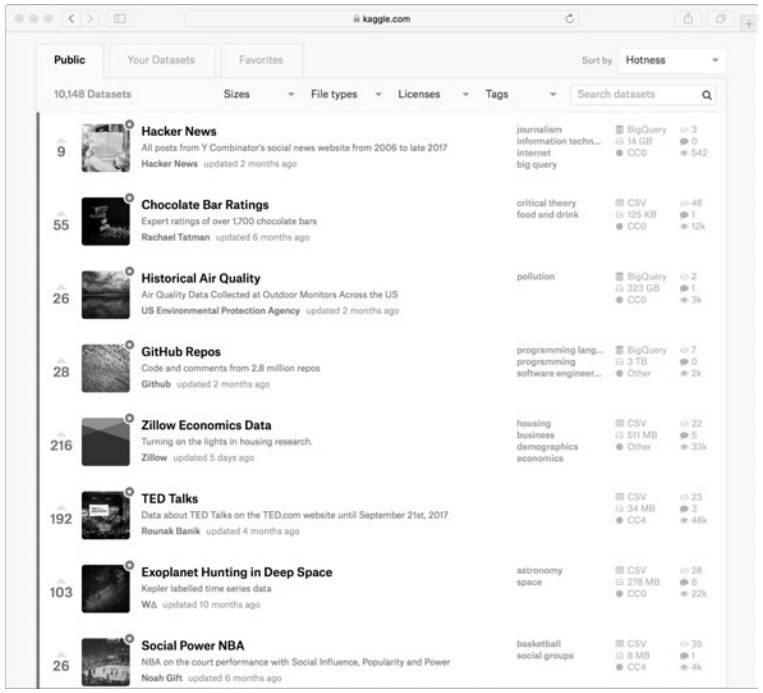

Abbildung 2.5 Liste von Datasets bei Kaggle

Kaggle als Inspirationsquelle für das Machine Learning

Bei einem Blick auf Kaggle wird der Ausdruck »Daten sind das neue Öl« verständlicher. Die auf Kaggle angebotenen Datasets können Ihnen auch als neue Inspirationsquelle für zukünftige Apps oder Machine-Learning-Aufgaben dienen. Als Beispiel listen wir ein paar spannende Fragenstellungen auf, die sich mit Kaggle-Datasets und Machine Learning bestimmt beantworten lassen:

- Welche Pilze kann ich essen?
- Wie hoch wird die Miete für ein 200 m^2 großes Haus in einem bestimmten Stadtteil sein?
- Zur welcher botanischen Familie gehört eine bestimmte Blume?
- Welche Stethoskop-Audioaufnahmen zeigen Herzschlaganomalien an?
- Apfel oder Orange? Auch wenn diese Aufgabe für den Menschen ziemlich trivial klingt, bietet Kaggle mit dem Dataset *Fruits 360* (*https://www.kaggle.com/moltean/fruits/data*) die Möglichkeit, aus einer Früchtebilderdatenbank seine eigenen Modelle zu trainieren und somit diese besonders komplexe Frage zu beantworten!

Viele Ergebnisse im Bereich der Medizin, z. B. die Bewertung, ob eine Person auf HIV-Behandlungen anspricht, wurden dank der frei verfügbaren Datasets von Kaggle berechnet (Quelle: *http://science.sciencemag.org/content/331/6018/698.full*). Und falls Sie nun Lust bekommen haben, diese Daten zu erforschen und daraus neue Ergebnisse zu gewinnen, können Sie – nachdem Sie dieses Buch gelesen haben – auch an den zahlreichen und spannenden Kaggle Challenges teilnehmen!

2.5.2 Google Dataset Search

Google bietet seit September 2018 die Suchmaschine *Dataset Search*, die es ermöglicht, frei zugängliche Datasets von Behörden, Universitäten und Firmen zu finden (siehe Abbildung 2.6). Sie stellt neben Kaggle einen guten Startpunkt dar, um relevante Datenquellen ausfindig zu machen.

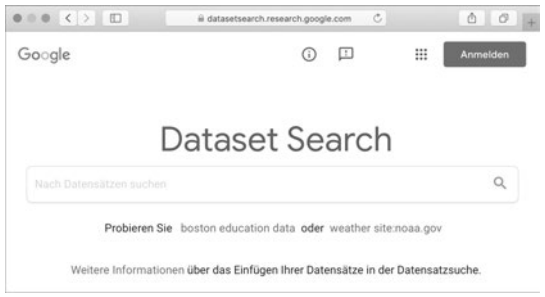

Abbildung 2.6 Google Dataset Search (https://datasetsearch.research.google.com)

2.5.3 Chars74K

Die Aufgabe von OCR (*Optical Character Recognition*) besteht darin, eine Texterkennung von Zeichen in Dokumenten (z. B. in Bildern, Texten oder Formularen) zu ermöglichen, die durch ein optisches Eingabegerät erzeugt worden sind, z. B. durch einen Scanner oder eine Digitalkamera. Diese Aufgabe gelingt besonders gut bei Textdokumenten, deren Schriftarten und Hintergrund gut erkennbar sind. Doch nicht jedes Bild stammt aus einem gut gescannten Formular, insbesondere wenn dieses Bild mit einer Handykamera aufgenommen wurde.

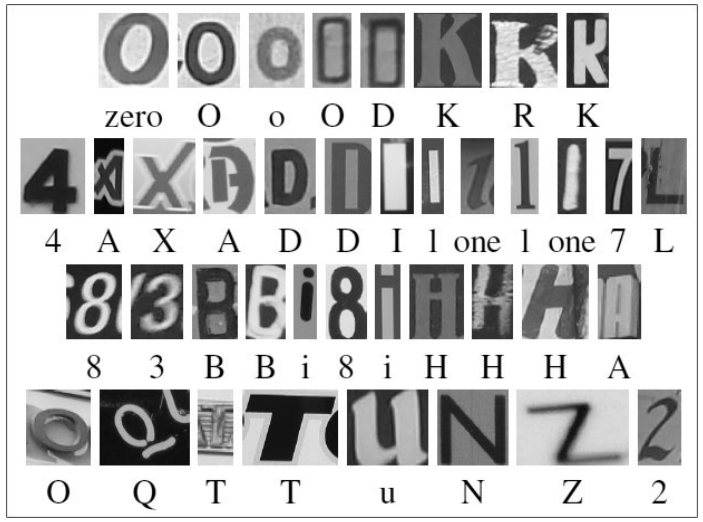

Abbildung 2.7 Auszug aus dem »Chars47K dataset« (Bildquelle: http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/)

Um trotz allem OCR auf solche Bilder durchführen zu können, hat Chars74K einzelne Buchstaben aus unterschiedlichen Bildern (z. B. den Buchstaben *C* von einer Coca-Cola-Dose oder ein *X* aus einer Twix-Verpackung) gesammelt und annotiert (siehe Abbildung 2.7). Weitere Quellen, wie die Kannada-Schrift[6] oder Handschriften können Sie optional herunterladen und somit die Erkennung verbessern.

2.5.4 ImageNET

Zu den wohl bekanntesten und von Forschern und Entwicklern meistbenutzten Datasets gehört definitiv ImageNET.[7] Mit aktuell 14 Millionen annotierten Bildern dient diese Bilddatenbank als Grundlage für zahlreiche Bilderkennungswettbewerbe. Das bedeutet, wenn Sie Ihre generierten Modelle testen und evaluieren wollen, können

6 *https://de.wikipedia.org/wiki/Kannada-Schrift*
7 *http://www.image-net.org*

Sie diese Bilder als Evaluierungsbilder benutzen. Doch die Besonderheit von ImageNET besteht nicht nur in der Anzahl der Bilder, sondern auch darin, dass jedes Bild zu einer Kategorie bzw. Hierarchie des WordNet-Datasets gehört. Was bedeutet das? WordNet[8] ist eine sogenannte lexikalische Datenbank; darin werden Verben, Namen und sogar Adjektive in Gruppen bzw. Kategorien zusammengefasst. Diese Gruppen werden nach dem Prinzip der Synonyme gebildet, was zum englischen Begriff *Synset* führt.

Bei ImageNET werden dann diese vorgegebenen WordNet-Kategorien benutzt, um den Inhalt eines Bildes zu beschreiben. Nehmen wir folgendes Beispiel aus ImageNET: Wenn Sie nach »dog« suchen, werden Sie viele Bilder von unterschiedlichen Hunden diverser Rassen finden (siehe Abbildung 2.8). Auf der linken Seite finden Sie Kategorien. Wenn Sie auf die Begriffe klicken, klappen neue Kategorien auf (z. B. »spaniel«, »terrier«, »poodle«, siehe Abbildung 2.9). Diese Kategorien stammen vom WordNet-Dataset.

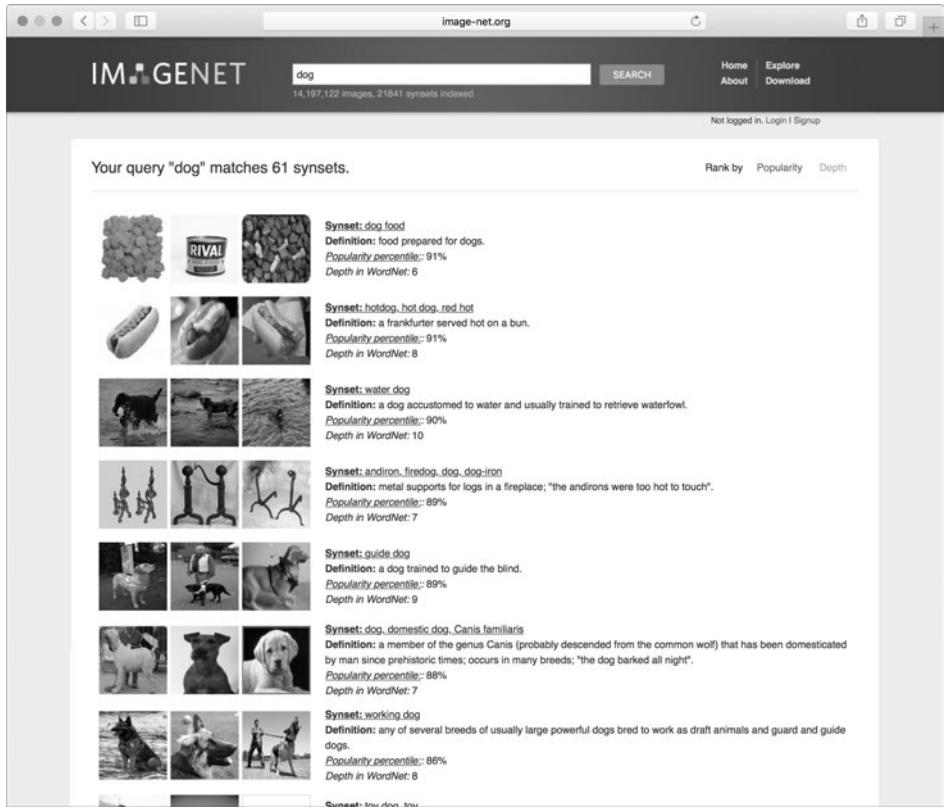

Abbildung 2.8 Ergebnisse einer Suche bei ImageNET

8 *https://wordnet.princeton.edu*

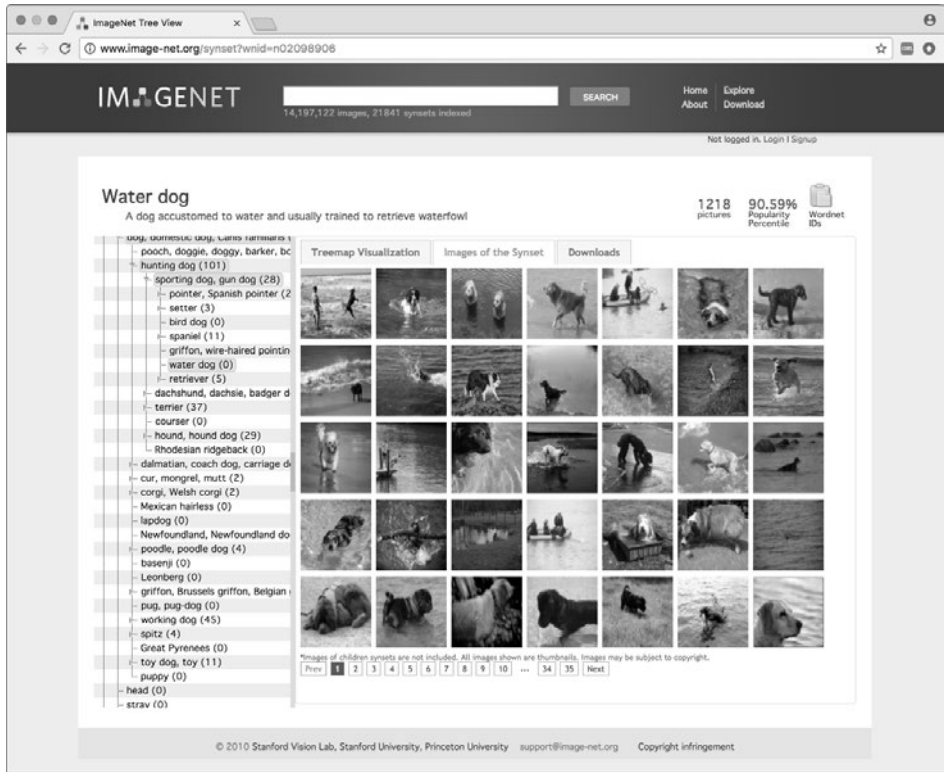

Abbildung 2.9 Auf der linken Seite können Sie durch die Synonyme und Kategorien von »dog« navigieren.

Gut zu wissen: VGG-Modelle

Bekannte Modelle, wie VGG-16 oder VGG-19 (siehe Kapitel 3, »Neuronale Netze«), wurden auf ImageNet trainiert.

Gut zu wissen: Natural Adversarial Examples

Modelle werden auf Basis von Datensätzen wie ImageNet trainiert und liefern auch in der Regel gute Ergebnisse. Es gibt daneben jedoch noch Daten, die für Deep-Learning-Modelle eine große Herausforderung darstellen, da sie systematisch missklassifiziert werden. Solche Bilder stellen ImageNet-A und ImageNet-O (*https://arxiv.org/pdf/1907.07174.pdf*) bereit. ImageNet-A enthält beispielsweise 7.500 Bilder, die erkannt werden sollten, aber dennoch nicht korrekt klassifiziert werden; ImageNet-O hingegen beinhaltet Bilder ungesehener Klassen mit Anomalien, die zu einem niedrigen Konfidenzwert der Klassifikation führen sollten. Abbildung 2.10 zeigt Beispiele missklassifizierter Bilder. So wird z. B. (oben rechts in der Abbildung) ein Bild mit einer Libelle als Bild eines Gullydeckels klassifiziert.

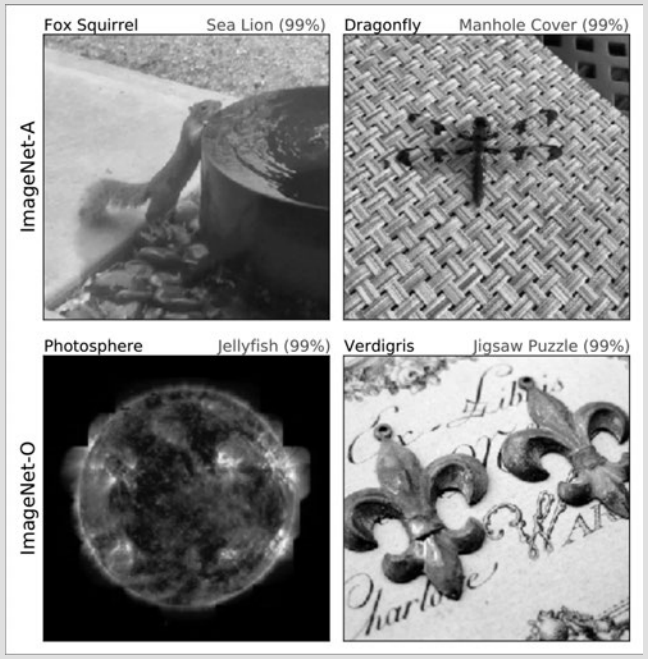

Abbildung 2.10 Beispiele missklassifizierter Bilder (Bildquelle: https://arxiv.org/pdf/1907.07174.pdf)

Beide Datensätze können unter der Adresse *https://people.eecs.berkeley.edu/~hendrycks/imagenet-a.tar* bzw. *https://people.eecs.berkeley.edu/~hendrycks/imagenet-o.tar* heruntergeladen werden.

2.5.5 ImageClef

Eine weitere sehr beliebte Bilddatenbank zum Evaluieren und Trainieren von Bilderkennungsalgorithmen ist ImageClef. Hier werden mehrere Bild-Datasets angeboten.[9] Die Bilder stammen meistens aus freien Quellen oder aus Wikipedia und wurden in verschiedensten Granularitäten annotiert und beschrieben. Die Annotationen reichen von einem globalen textuellen Inhalt des Bildes bis zu einer genauen Segmentierung des Bildes. Das heißt, die einzelnen Teilelemente des Bildes (*Baum, Strand, Himmel*) werden bei der Beschreibung ebenfalls berücksichtigt (siehe Abbildung 2.11).

Wenn Sie Ihre Bilderkennung mithilfe des Datasets von ImageClef trainieren, können Sie zum Beispiel herausfinden, ob im Bild eine Person am Strand liegt oder ein Hund über die Straße läuft!

9 *http://www.imageclef.org/datasets*

Abbildung 2.11 Beispiel einer Segmentierung mit dem Dataset von »ImageClef« (Bildquelle: http://www.imageclef.org)

2.5.6 VGG

Die Universität Oxford bietet zahlreiche Datasets an, die Sie frei für das Trainieren Ihrer Modelle benutzen können. Diese Datasets sind sehr unterschiedlich und umfassen annotierte Bildern von Hunden und Katzen (Oxford IIT Pet Dataset[10], siehe Abbildung 2.12) sowie botanische Eigenschaften von Blumen (Oxford Flowers Dataset[11]) bis hin zu Skulpturen[12] von Auguste Rodin und Henry Moore und einer ausführlichen Auswahl von Gebäuden in Paris (siehe Abbildung 2.13).

[+]

Tipp: Brauchen Sie Inspiration?

Falls Sie neue Ideen für eine App oder Software im Bereich Bild- bzw. Videoerkennung suchen, sollten Sie sich die Demo-Webseite der Forscher der *Oxford Visual Geometry Group* anschauen.[13] Zahlreiche Onlinedemos zeigen auf Basis von frei verfügbaren Datasets, welche spannenden Szenarien im Multimediabereich mit Machine Learning erzielt werden können!

10 *http://www.robots.ox.ac.uk/~vgg/data/pets/*
11 *http://www.robots.ox.ac.uk/~vgg/data/flowers/*
12 *http://www.robots.ox.ac.uk/~vgg/data/sculptures6k/*
13 *http://www.robots.ox.ac.uk/~vgg/demo/*

Abbildung 2.12 Bildschirmauszug des »Oxford IIT Pet Dataset«

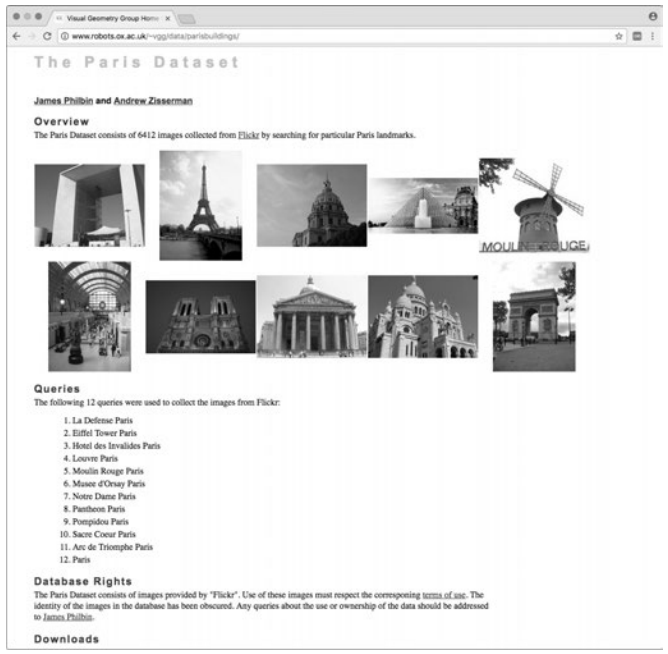

Abbildung 2.13 »The Paris Dataset« beinhaltet Bilder von Gebäuden in Paris, die über Flickr gesammelt und annotiert wurden.

2.5.7 YFCC100M

Über 100 Millionen Bilder und Videos, die von Flickr stammen und durch die jeweiligen Autoren mit der Creative-Commons-Lizenz versehen wurden, bilden die sehr umfangreiche Bild- und Videodatenbank namens YFCC100M.[14] Nachdem Sie ein Onlineformular ausgefüllt haben, können Sie sie herunterladen.

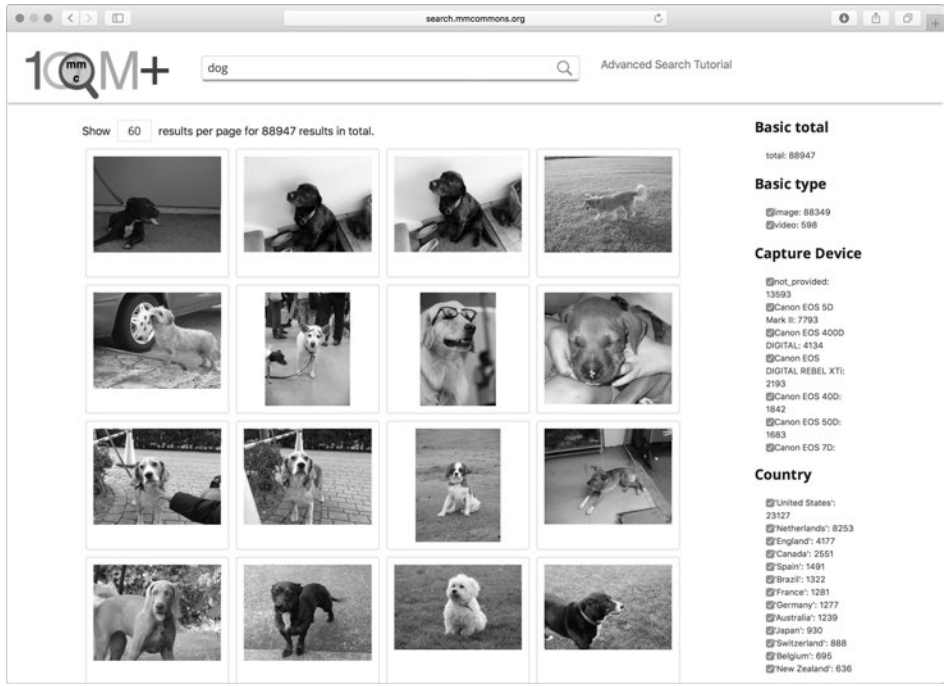

Abbildung 2.14 Auszug des Datasets »YFCC100M«

Falls Sie jedoch ohne Herunterladen einen Einblick in die gigantische Datenbank bekommen möchten, können Sie die Website der am Projekt YFCC100M beteiligten Forscher besuchen: *http://search.mmcommons.org* (siehe Abbildung 2.14). Über eine Suche können Sie durch alle Medien (Bilder und Videos) stöbern und über entsprechende Tags die Suchergebnisse verfeinern.

2.5.8 YouTube-8M

Das YouTube-8M Dataset[15] beruht, wie der Name schon verrät, nur auf YouTube-Videos. Das Dataset beinhaltet über 450.000 Stunden Videos, die anhand der Labels annotiert wurden, die von Benutzern angegeben worden sind. Diese Videos wurden

14 *https://webscope.sandbox.yahoo.com/catalog.php?datatype=i&did=67*
15 *https://research.google.com/youtube8m/*

dann in 4.716 unterschiedliche Klassen sortiert und mit Knowledge-Graph[16]-Entitäten semantisch verknüpft (siehe Abbildung 2.15).

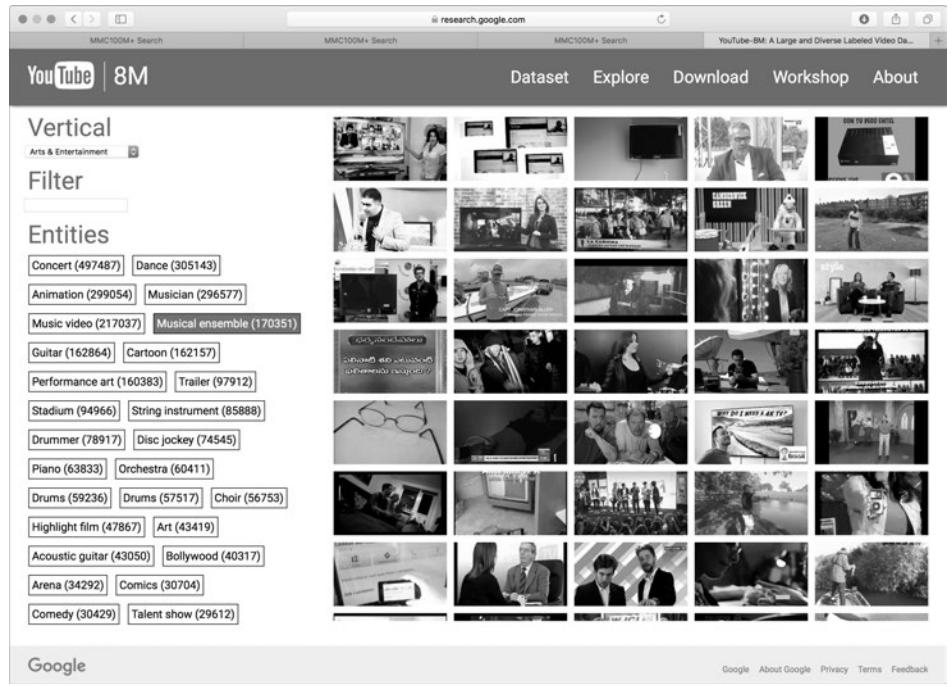

Abbildung 2.15 Auszug des »YouTube-8M Dataset«

2.5.9 MS-Celeb-1M

Microsoft bot bis vor Kurzem mit seiner Bilddatenbank MS-Celeb-1M[17] eine Sammlung von einer Million Promi-Bildern an (siehe Abbildung 2.16). Diese konnten benutzt werden, um z. B. in Bildern Prominente automatisch zu erkennen. Kritisiert wurde aber, dass neben den Promi-Bildern im Netz frei zugängliche Bilder integriert wurden. Das Dataset enthielt 10 Millionen Bilder von mehr als 100.000 unterschiedlichen Personen.

Hinweis: Nutzungsbedingungen von Datasets

MS-Celeb-1M wurde bis vor einiger Zeit zum Zwecke der Gesichtserkennung benutzt, musste aber nach diverser Kritik wegen missbräuchlicher Nutzung des Datasets durch Dritte von Microsoft deaktiviert werden. Bei Datasets mit Promi-Bildern ist oft

16 *https://developers.google.com/knowledge-graph/*
17 *https://www.researchgate.net/publication/295074418_MS-Celeb-1M_Challenge_of_Recognizing_One_Million_Celebrities_in_the_Real_World*

die Benutzung der Daten nur für Forschungszwecke und nichtkommerzielle Anwendungen erlaubt. Lesen Sie bitte immer vorher sorgfältig die Nutzungsbedingungen!

Abbildung 2.16 Auszug des »MS-Celeb1M Datasets« für die Künstlerin »Lady Gaga«

2.5.10 CelebA

Das Dataset CelebA[18] beinhaltet 40 Gesichtsattribute (Klassen, wie z. B. Eyeglasses, Mustache, Wavy_Hair)[19] für mehr als 200.000 Promi-Bilder und kann unter anderem benutzt werden, um Modelle zu entwickeln, die Posen oder Mimik-Variationen bei Menschen erkennen (siehe Abbildung 2.17).

Abbildung 2.17 Beispielbilder vom »CelebA«-Dataset

18 *http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html*
19 Für alle Attribute, siehe auch: *https://www.tensorflow.org/datasets/catalog/celeb_a*

2.5.11 VoxCeleb2

Mit dem Dataset VoxCeleb2[20] (siehe Abbildung 2.18) bleiben wir in der Welt der Promis. Dieses von der University of Oxford bereitgestellte Dataset enthält über eine Million aus YouTube-Videos extrahierte Äußerungen von ca. 6.000 prominenten Personen. Auf dieser Grundlage lassen sich Sprecherklassifikationen erstellen. Das Dataset kann unter *http://www.robots.ox.ac.uk/~vgg/data/voxceleb/vox2.html* heruntergeladen werden.

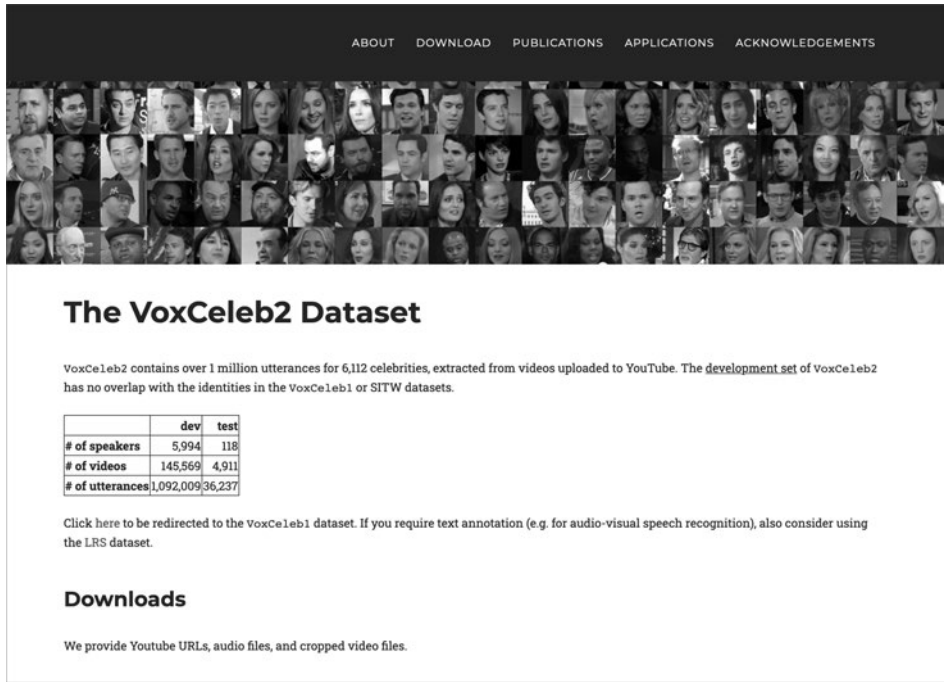

Abbildung 2.18 Die Website von »VoxCeleb2«

2.5.12 Microsoft COCO

Eine von Microsoft initiierte Datenbank für Objekterkennung ist die Microsoft-COCO-Datenbank, die oft auch nur COCO[21] genannt wird (nach dem englischen Akronym für *Common Objects in Context*). Bei dieser Datenbank wurden 330.000 Bilder samt Segmentierung und 1,5 Millionen Objektinstanzen annotiert und klassifiziert. Zusätzlich wurden auch die Kontexte der Bilder erfasst. Das heißt, bei jedem Bild wird eine genaue textuelle Beschreibung des Bildinhalts zurückgegeben. Über eine COCO-

20 Siehe den Artikel: J. S. Chung, A. Nagrani, A. Zisserman: VoxCeleb2: Deep Speaker Recognition, INTERSPEECH, 2018 unter *http://www.robots.ox.ac.uk/~vgg/publications/2018/Chung18a/chung18a.pdf*
21 *http://cocodataset.org*

API können Sie leichter auf die Annotationen und die Bilder zugreifen. In den letzten Monaten wurden zahlreiche performante Modelle zur Erkennung von Personen und Objekten auf Basis des COCO-Datasets trainiert.

Über die interaktive *COCO Explorer*-Suchseite[22] können Sie erfahren, welche Datasets innerhalb des COCO-Datasets vorhanden sind, und sich somit einen schnellen Überblick über die Segmentierung, Bilder und Annotationen verschaffen.

Abbildung 2.19 zeigt einen Auszug von einer Recherche mit dem Stichwort »car«. Hier sehen Sie die Segmentierungen und die Objekte, die als Bereich gekennzeichnet werden. Wenn Sie auf das Listen-Icon klicken, können Sie die textuelle Beschreibung des Bildes einsehen.

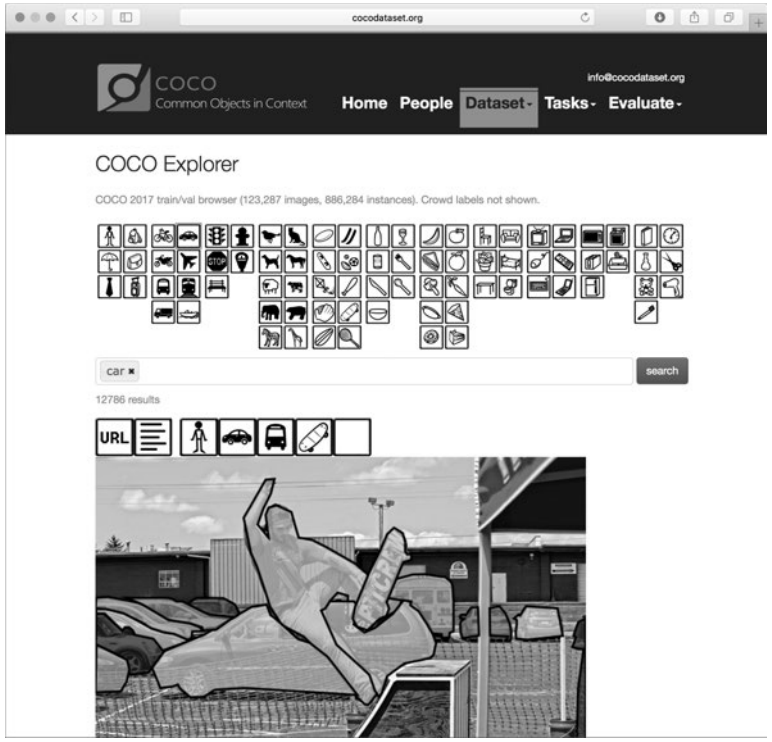

Abbildung 2.19 Auszug der Suchseite des »COCO Explorer«

2.5.13 MNIST und Fashion-MNIST

Die Datenbank MNIST[23] (*Modified National Institute of Standards and Technology*) enthält über 70.000 handgeschriebene Schwarz-Weiß-Bilder von Zahlen (siehe Abbildung 2.20) und gehört zu den wohl bekanntesten Datasets im Bereich Machine

22 *http://cocodataset.org/#explore*
23 *http://yann.lecun.com/exdb/mnist/index.html*

Learning. In der Tat wird dieses Dataset sehr oft als Beispiel für Klassifizierungsmodelle und Benchmarks benutzt.

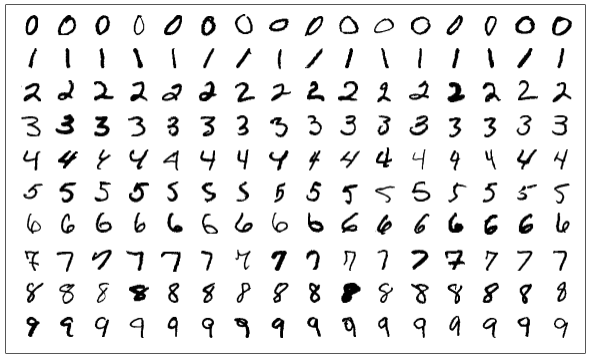

Abbildung 2.20 Auszug aus dem »MNIST-Dataset«

Experten sind jedoch der Meinung, dass die Benutzung der MNIST-Datenbank als Benchmark nicht mehr zeitgemäß sei: Die Klassifizierung der Bilder sei zu einfach und die Modelle würden mit einer Präzision von 99,7 % den Inhalt der Bilder erkennen.[24] Aus diesem Grund werden wir in diesem Buch oft die neuere Fashion-MNIST-Datenbank[25] benutzen. Diese wurde von dem Online-Versandhändler Zalando zur Verfügung gestellt und beinhaltet statt handgeschriebener Schwarz-Weiß-Bilder von Zahlen Bilder von T-Shirts, Schuhen, Taschen und Kleidern (siehe Abbildung 2.21). Hätten Sie jemals gedacht, dass Ihre erste Begegnung mit Machine Learning etwas mit Fashion und Mode zu tun haben würde?

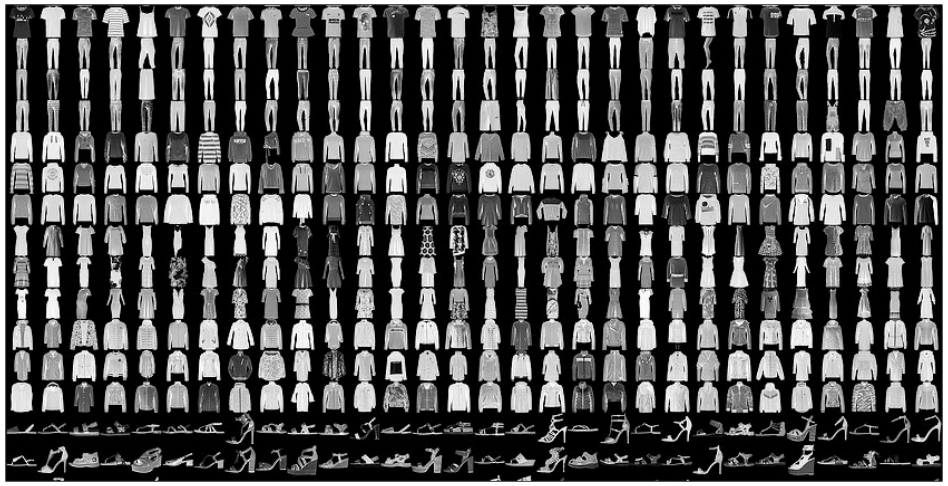

Abbildung 2.21 Auszug aus dem Zalando »Fashion-MNIST«-Dataset

24 *http://fashion-mnist.s3-website.eu-central-1.amazonaws.com*
25 *https://github.com/zalandoresearch/fashion-mnist*

2.5.14 UCI-Datasets

Das *Center for Machine Learning and Intelligent Systems* der University of California in Irvine stellt seit Mitte der 80er-Jahre eine Sammlung verschiedener frei zugänglicher Datasets bereit (siehe *https://archive.ics.uci.edu/ml/datasets.php*). Aufgelistet sind inzwischen 488 Datensätze für unterschiedliche Aufgaben (Klassifikation, Regression, Clustering) zu diversen Bereichen (Medizin, Wirtschaft, Umwelt, Soziologie, soziale Medien etc.; siehe Abbildung 2.22). Ein Manko ist aber, dass einige Datasets der Sammlung teilweise nicht groß genug sind.

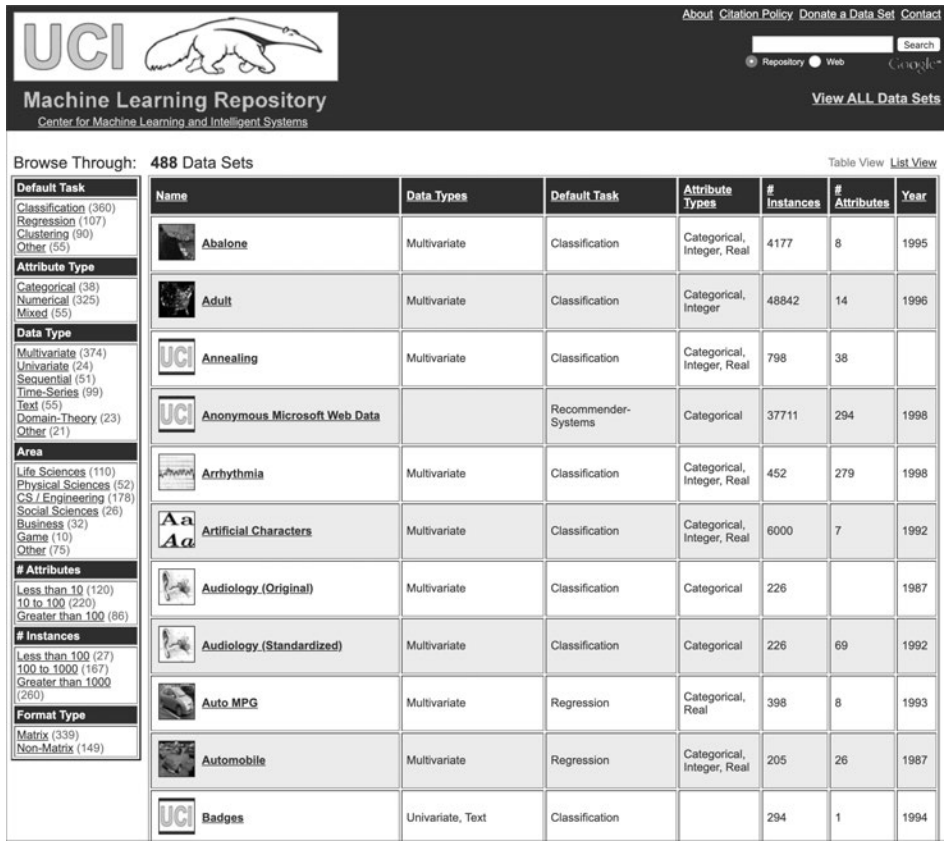

Abbildung 2.22 Liste von Datasets auf der »UCI Machine Learning Repository«-Website

2.5.15 Uber-Datasets

Die Firma Uber stellt auch frei zugängliche Datasets bereit, so z. B. *Uber Movement* (*https://movement.uber.com/*), das Verkehrsdaten für diverse Städte liefert. Im Kontext von Smart City sind Daten dieser Art nützliche Mittel zur Städte- und Verkehrsplanung. Abbildung 2.23 zeigt einen Auszug der Daten für Berlin.

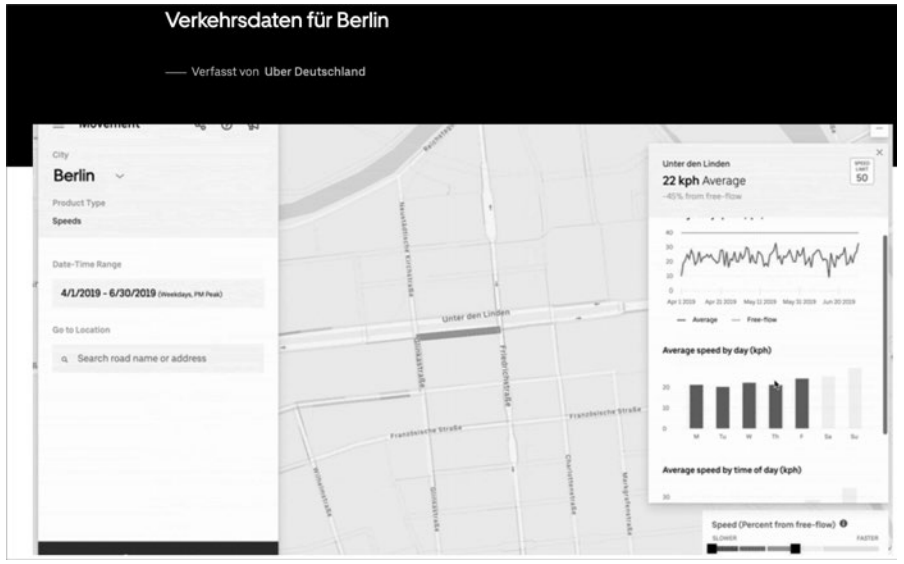

Abbildung 2.23 Auszug aus »Uber Movement« zu Berlin

2.5.16 CLEVR-Dataset

Zur Entwicklung von KI-Systemen, die in der Lage sind, visuelle Daten zu »verstehen« und Fragen zu den Bildern zu beantworten, bietet sich das CLEVR-(*Compositional Language and Elementary Visual Reasoning*-)Dataset[26] der Stanford University an. Das Dataset beinhaltet ca. 100.000 Bilder und über 850.000 Fragen. Die Datensätze können von der Seite *https://cs.stanford.edu/people/jcjohns/clevr* heruntergeladen werden. Abbildung 2.24 zeigt Beispielfragen zu einem Bild aus dem CLEVR-Dataset.

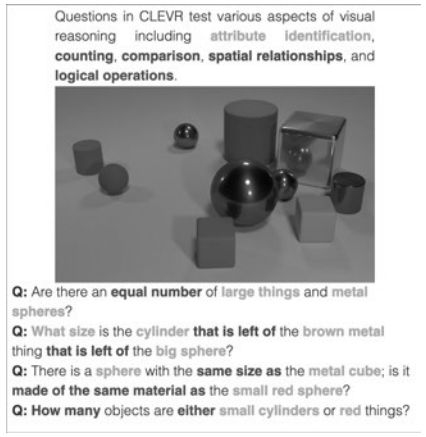

Abbildung 2.24 Beispielfragen zu einem Bild aus dem »CLEVR«-Dataset

26 *https://arxiv.org/pdf/1612.06890.pdf*

2.5.17 Weitere Datasets

Die oben erwähnten Datasets bilden natürlich nur eine Auswahl von Daten, die für das Training von neuronalen Netzen benutzt werden können. Es gibt weitere Datasets, die jedoch domänenspezifischer sind (z. B. für den Finanzsektor oder für den medizinischen Bereich) bzw. nur für bestimmte Aufgaben angewandt werden können (Wetterprognose, Emotionserkennung in Texten, Früherkennung von Defekten elektronischer Komponenten).

Aus dem Grund empfehlen wir Ihnen, noch einen Blick auf folgende Links zu Datasets zu werfen:

- *https://en.wikipedia.org/wiki/List_of_datasets_for_machine_learning_research*
- *https://scikit-learn.org/stable/datasets*
- *https://www.visualdata.io/*
- *https://www.kairos.com/blog/60-facial-recognition-databases*
- *https://data-flair.training/blogs/machine-learning-datasets*

Wenn Sie nach bestimmten Daten im Netz suchen, können Sie auch die zahlreichen Open-Data-Initiativen besuchen, z. B. die deutsche Website *Govdata*[27], die Rohdaten aus Bund, Ländern und Kommunen in verschiedenen Kategorien anbietet, oder die Website *Statista.de*, die Statistiken und Fakten für mehrere Domänen bzw. Branchen zum Herunterladen anbietet. Welche Daten von welchen Quellen Sie benutzen sollen, hängt ganz davon ab, welche Domäne Sie analysieren möchten!

Mehr über Open Data

Falls Sie mehr über die Potenziale, Prinzipien, Fakten und rechtliche Fragen rund um Open Data erfahren möchten, bietet die Konrad-Adenauer-Stiftung Ihnen unter folgender URL eine ausführliche Broschüre zum Herunterladen an:

https://www.kas.de/wf/de/33.44530

2.5.18 Checkliste zu Datasets

Sie haben eine interessante Datenquelle gefunden und möchten diese direkt in Ihr Machine- bzw. Deep-Learning-Projekt einbeziehen? Nicht zu schnell! Zuerst müssen Sie folgende wichtige Punkte überprüfen, bevor Sie diese Daten für einen kommerziellen Gebrauch benutzen und ein Modell daraus extrahieren können.

27 *https://www.govdata.de*

Gehen Sie die folgende kleine Checkliste durch, und überprüfen Sie die einzelnen Punkte:

- Datum der Erfassung der Daten (Sind die Daten noch aktuell?)
- Herausgeber (Quelle, Internetadresse?)
- Verfügbarkeit (online, offline, auf Nachfrage?)
- Lizenzmodell (Creative Commons, Share-a-like, freie Nutzung oder Public Domain?)
- Nutzungsbedingungen (In welchem Rahmen dürfen diese Daten benutzt werden? Können diese auch kommerziell verwertet werden?)
- Ist ein Copyright oder Zitat notwendig? Oft ist es so, dass bei Datasets, die von Universitäten oder Forschergruppen zur Verfügung gestellt werden, verlangt wird, dass ein Zitat erfolgt oder ein Verweis auf eine Publikation angegeben wird.
- Formate (CVS, JSON, XLS, MATLAB?)
- Medientypen (RGB-Bilder, Texte, Audio?)
- Volumen der Daten
- Granularität der Annotation der Daten
- weitere Verwertung der Daten und des erlernten Modells

2.6 Zusammenfassung

In diesem Kapitel haben Sie erfahren, was Machine und Deep Learning sind, welche Rolle Datasets spielen und wo Sie welche finden können. Bevor wir ausführlich auf die Frameworks TensorFlow, Keras und TensorFlow.js eingehen, lernen Sie im nächsten Kapitel die Grundlagen von Deep Learning kennen, nämlich die neuronalen Netze.

Kapitel 6
Keras

»I get very excited when we discover a way of making neural networks better – and when that's closely related to how the brain works.«
– Geoffrey Hinton

TensorFlow ist sehr ein mächtiges Framework, das viele Möglichkeiten anbietet, seine Modelle zu trainieren. Aber wegen seiner Komplexität haben es bis TensorFlow 2 auch Entwickler nicht immer als sehr benutzerfreundlich betrachtet: Ein Teil von ihnen empfand TensorFlow als ungeeignet für schnelles Prototyping. Entwickler benutzten lieber TensorFlow als Backend mit dem von François Chollet entwickelten Framework *Keras*.

6.1 Von Keras zu tf.keras

Keras wurde zunächst im Rahmen des Projekts ONEIROS (*Open-ended Neuro-Electronic Intelligent Robot Operating System*) entwickelt – mit dem Hintergedanken, ein generisches unabhängiges Framework für das Deep Learning zu implementieren. In der Tat unterstützt Keras verschiedene sogenannte Backends bzw. Deep-Learning-Frameworks mit einer einheitlichen API, z. B. *Microsoft Cognitive Toolkit* (CNTK), *Theano* und *TensorFlow*. Konkreter: Sie können ein neuronales Netz in Keras definieren und dann entscheiden, ob Sie es eher mit dem CNTK- oder dem TensorFlow-Framework trainieren lassen wollen. Wenn Sie eine (oder mehrere) GPUs besitzen, wird Keras den Code automatisch auf dieser ausführen.

Genau Letzteres machte Keras bei den TensorFlow-Benutzern so populär und motivierte zugleich die Entwickler von TensorFlow bei Google, Keras mit seinen Modulen direkt in TensorFlow zu integrieren. Ab 2017 wurde Keras in TensorFlow allmählich integriert und gehörte offiziell zu den sogenannten TensorFlow-High-Level-APIs. Keras ist als Modul *tf.keras* in TensorFlow 2 integriert und bietet vorgefertigte Funktionen, z. B. um ein Modell zu trainieren, ein Dataset zu laden und um natürlich mit wenig Aufwand Schichten und Aktivierungsfunktionen zu definieren.

6.1.1 K wie Keras oder Konfusion

Bevor Sie mit Ihrer Suchmaschine nach Keras suchen und wahrscheinlich den ersten Treffer anklicken, müssen wir Ihnen leider mitteilen, dass es nicht nur eine Keras-Version gibt, sondern eigentlich zwei! In der Tat herrscht durch die Integration von Keras in TensorFlow zurzeit ein wenig Konfusion beim Import der Python-Module.

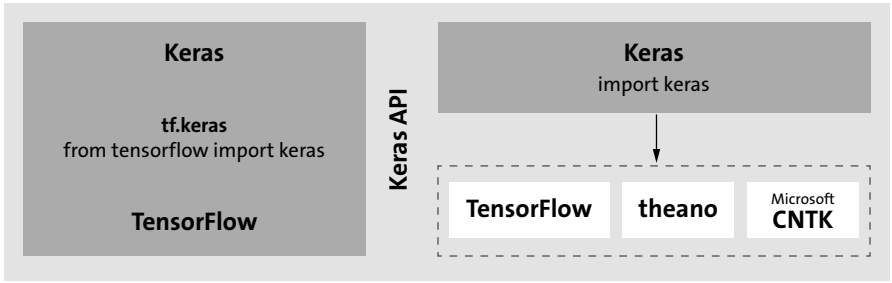

Abbildung 6.1 Die zwei Keras-Versionen: die »Keras API« integriert in TensorFlow (links) und Keras als Frontend für Backends (rechts)

Die erste Version ist diejenige Keras-Version, die unabhängig von einem bestimmten Deep-Learning-Framework (auch als Multi-Backend bezeichnet) installiert werden kann (Sie sehen sie rechts in Abbildung 6.1). Das heißt, Sie können einfach festlegen, mit welchem Framework im Hintergrund (TensorFlow, CNTK oder Theano) Keras ausgeführt werden soll. Auf der offiziellen Keras-Website (*https://keras.io*, siehe Abbildung 6.2) werden Sie zahlreiche Beispiele und eine sehr gute Dokumentation zu den Funktionen finden.

Die Installation erfolgt durch folgendes Kommando:

```
pip install keras
```

Die zweite Version von Keras (links in Abbildung 6.1) wird von TensorFlow 2 mitgeliefert (siehe Abbildung 6.3) und wird in Python durch folgende Zeile importiert:

```
import tensorflow as tf
from tensorflow import keras
```

Falls Sie schon Keras (z. B. von früheren Projekten) auf Ihrem Rechner haben und nicht mehr wissen, welche Version tatsächlich benutzt wird, können Sie dies mit folgendem Code überprüfen:

```
# Benutzung von Keras (in TensorFlow 2)
try:
    import tensorflow as tf
    from tensorflow import keras
    print("Keras TensorFlow version: {}".format(keras.__version__))
except:
    print("Keras TensorFlow Version nicht installiert")

# Wenn Sie zusätzlich Keras über keras.io installiert haben, werden Sie
# folgende Zeilen ausführen können.
# Wenn es nicht der Fall sein sollte,
# werden Sie folgende Fehlermeldung erhalten:
# ModuleNotFoundError: No module named 'keras'
try:
    import keras
    print("Keras Version: {}".format(keras.__version__))
except:
    print("Keras Version von keras.io nicht installiert")
```

Listing 6.1 Überprüfung der installierten Keras-Version (Auszug von »chap_6/keras_version_check.py«)

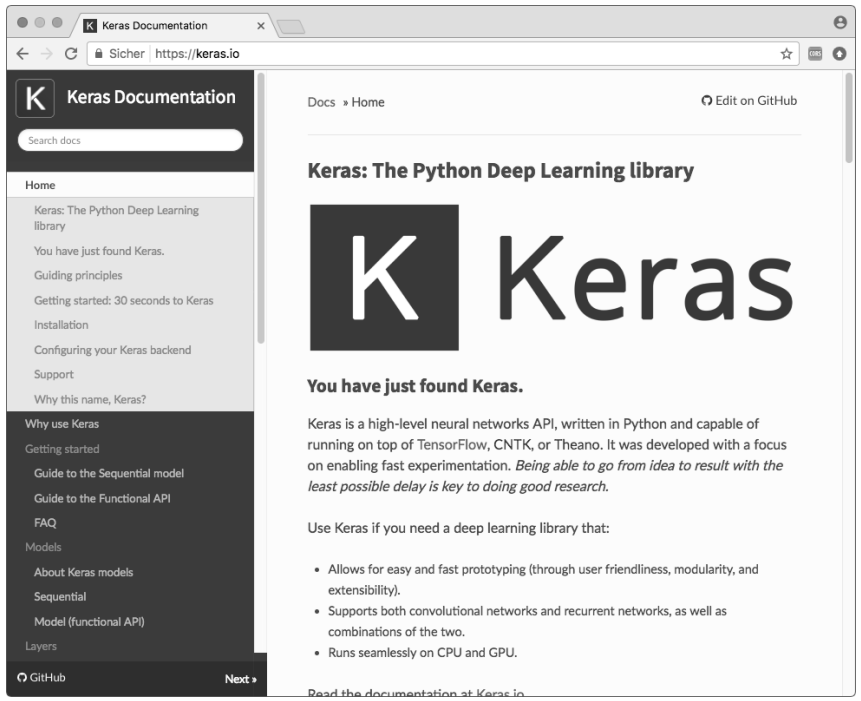

Abbildung 6.2 Die offizielle Website von Keras

Sie werden jedoch feststellen, dass sich die Versionsnummern der beiden (*Keras.io* und *tf.keras*) möglicherweise unterscheiden. In der Tat hat die Keras-Version von *Keras.io* eine höhere Versionsnummer als die von *tf.keras*. Das bringt uns zum nächsten Abschnitt ...

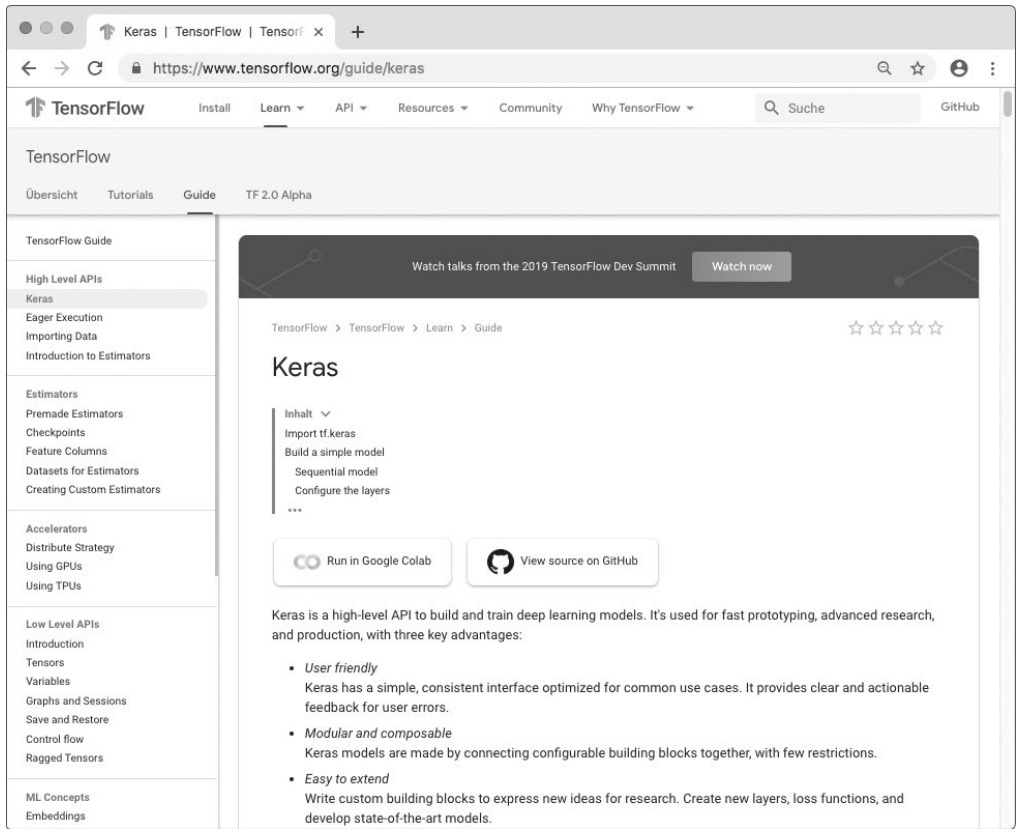

Abbildung 6.3 Keras auf der offiziellen TensorFlow-Website

6.1.2 Die Zukunft von Keras

Die Tatsache, dass zwei Versionen – und somit zwei Entwicklungsstränge – von Keras koexistieren, brachte den Gründer François Chollet Ende 2019 dazu, auf der GitHub-Seite von Keras[1] kurz den Stand der Versionen zu erklären: *Keras 2.3.0* von *Keras.io* ist ihm zufolge die erste und zugleich letzte Version des Multi-Backend-Keras, die TensorFlow 2.0 benutzen kann. Die API wurde an die API von `tf.keras` angeglichen. Keras ist auch noch zu Theano, CNTK, TensorFlow 1.14, 1.13 und 2.0 kompatibel, aber leider unterstützen Letztere z. B. die Eager Execution nicht! Zusätzlich wird explizit dazu geraten, auf `tf.keras` umzusteigen, da Support und Bugfixes von Keras.io in den nächsten Monaten bald nicht mehr gewährleistet werden.

Zusammengefasst: Solange Sie nicht beabsichtigen, Keras mit einem weiteren Backend wie Theano oder CNTK zu benutzen, raten wir Ihnen, nur die in TensorFlow integrierte Version von Keras (`tf.keras`) zu verwenden.

1 *https://github.com/keras-team/keras*

Wichtig!

Fast alle Beispiele in diesem Buch stützen sich auf die tf.keras-Version von TensorFlow 2. Keras.io (2.3.0) werden wir nur bei bestimmten Beispielen in Kapitel 7 und Kapitel 9 einsetzen. Wir werden Sie zum gegebenen Zeitpunkt darauf aufmerksam machen.

6.2 Erster Kontakt

In diesem Abschnitt werden wir uns kleineren Beispielen widmen, damit Sie Schritt für Schritt lernen, wie Keras benutzt werden kann, unter anderem um eine schnelle Zusammensetzung neuronaler Netze zu erzielen. Keras bietet eine für TensorFlow-Einsteiger sehr vereinfachte API an: Modelle werden in ein paar Zeilen definiert und durch das programmgesteuerte Zusammenschließen von Schichten realisiert. Das Training und die etwas komplexeren hinterliegenden Graph- bzw. Tensoroperationen werden im Hintergrund von TensorFlow ausgeführt. Keras kapselt so die Komplexität der Aufrufe von TensorFlow-Operation ab: ziemlich praktisch und programmierfreundlich!

Keras bietet zwei Möglichkeiten, ein Modell zur realisieren (*Sequential* und *Functional*), die wir in den nächsten Abschnitten präsentieren werden.

6.2.1 Sequential Model

Wie der Name es schon verrät, werden bei diesem Verfahren die Schichten dem Modell sequenziell hinzugefügt. Bei der Erstellung der Modelle werden die Instanzen der verschiedenen Schichten – diese werden als `Layers` bezeichnet – entweder mit der Funktion `model.add()` (Variante 1) oder mithilfe eines Arrays (Variante 2) angelegt. Folgendes Beispiel (*chap_6/keras_xor_sequential.py*) zeigt, wie ein sehr einfaches Netz eine XOR-Funktion (Exklusiv-Oder-Gatter) lernen kann:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.layers import Dense, Activation, Input

# 1. Variante
xor_model = Sequential()
xor_model.add(Dense(1024, input_dim=2))
xor_model.add(Activation('relu'))
xor_model.add(Dense(1))
```

```
xor_model.add(Activation('sigmoid'))

# 2. Variante
xor_model = Sequential([
    Dense(1024, input_dim=2),
    Activation('relu'),
    Dense(1),
    Activation('sigmoid')
])
```

Listing 6.2 Einfaches Keras-Modell

Die Reihenfolge, in der die Schichten hinzugefügt werden, gibt die Struktur des Modells vor. In unserem Beispiel haben wir zwei `Dense`-Schichten hinzugefügt mit jeweils 1.024 Neuronen und 1 Neuron. Die Eingabe-Dimension muss nur bei der ersten Schicht angegeben werden (hier mit dem Parameter `input_dim = 2`).

[+]

Tipp: Aktivierungsfunktionen

Sie können auch die Aktivierungsfunktion für jede Schicht mit dem Parameter `activation` angeben. Dadurch wird der Quellcode kompakter, wie im folgenden Beispiel:

```
xor_model = Sequential()
xor_model.add(Dense(1024, input_dim=2, activation="relu"))
xor_model.add(Dense(1, activation="sigmoid"))
```

6.2.2 Functional API

Mit dieser Form der Modellerstellung können Modelle mit mehreren Eingängen und mehreren Ausgängen, sogenannte azyklische Graphen, geschaffen werden, was beim Sequential-Modell nicht möglich ist. Folgendes Beispiel (aus *chap_6/keras_xor_functional.py*) zeigt, wie unser sehr einfaches Modell mit der Functional API von Keras aufgebaut werden kann. Die Variable `x` wird als Eingabe für die jeweilige nächste Schicht wiederverwendet:

```
inputs = Input(shape=(2,), dtype="float32")
x = Dense(1024, name="First_Layer")(inputs)
x = Activation('relu', name="ReLU ")(x)
x = Dense(1, name="Dense_Layer")(x)
predictions = Activation('sigmoid', name="Sigmoid ")(x)
```

Bei komplexeren Modellen wie im Projekt 2 von Kapitel 9, »Praxisbeispiele« (dort geht es um intelligente Spurerkennung), wird die Functional API zur Modellinitialisierung benutzt, um z. B. Abzweigungen (diese werden als *Skip Connections* bezeichnet) im Graphen zu definieren, die dann parallele Operationen durchführen.

Ausgabe der Modellstruktur

Mit der Keras-Funktion `model.summary()` können Sie jederzeit alle Schichten Ihres Modells in Textform ausgeben lassen, wie z. B. hier mit unserem Modell `xor_model`:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
First_Layer (Dense)          (None, 1024)              3072
_________________________________________________________________
Relu_Activation (Activation) (None, 1024)              0
_________________________________________________________________
Dense_Layer (Dense)          (None, 1)                 1025
_________________________________________________________________
Sigmoid_Activation (Activati (None, 1)                 0
=================================================================
Total params: 4,097
Trainable params: 4,097
Non-trainable params: 0
_________________________________________________________________
```

[+]

6.2.3 Keras-Layers

Keras bietet eine Reihe von Schichten (`tf.keras.layers`) an, die miteinander kombiniert werden können, um ein Modell zu generieren. Dazu zählen:

- Fully Connected Layers (Dense)
- Convolutional und Pooling Layers für CNNs (Conv1D, Conv2D, MaxPooling)
- Recurrent Layers (für die Definition von RNN und LSTM)
- Activation Layers (für Aktivierungsfunktionen)
- Normalization Layers (z. B. Batch Normalization)

Sie werden diese einzelnen `tf.keras.layers` Schritt für Schritt im Rahmen der Beispiele entdecken und einsetzen. Nun werden Sie Ihr erstes Modell mit Keras trainieren!

6.3 Modelle trainieren

Der Aufbau und die Generierung von Modellen laufen bei Keras nach dem Prinzip aus Abbildung 6.4 ab:

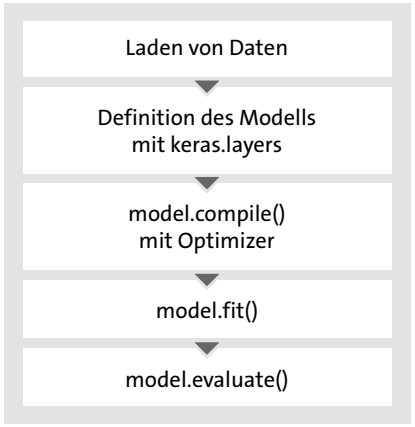

Abbildung 6.4 Typischer Prozess zur Generierung eines Modells mit Keras

Verschiedene Optimizer[2] können für das Training des Modells eingesetzt werden. Der `SGD-Optimizer` (*Stochastic Gradient Descent*-Optimizer) wird wie folgt mit einer Lernrate (`lr`) von 0.01 instanziiert:

```
sgd = SGD(lr=0.01)
```

Keras bietet weitere Optimizer, z. B. `Adadelta` , `Adagrad`, `Adamax` oder den `RMSprop` an.

Nachdem die einzelnen Schichten des Modells angelegt und instanziiert wurden, muss das Training noch mit einer Lernrate, einer Kostenfunktion und Metriken definiert werden. Dies passiert vor dem Training mit der Funktion `model.compile()`, die drei Eingabeparameter akzeptiert:

```
model.compile(loss="mean_squared_error", optimizer=sgd, metrics= [
"mae", "acc"])
```

Eine Kosten- bzw. Loss-Funktion muss ebenfalls angegeben werden. Hier kann direkt der Name der Loss-Funktion als String angegeben werden, z. B. `categorical_crossentropy`. Metriken können ebenfalls definiert werden. Diese werden als Parameter `metrics` innerhalb der `model.compile()`-Funktion angegeben, entweder als String (`mse`, `mae`, `accuracy`, `mean_squared_error`, `mean_absolute_error`) oder als Konstante (z. B. `metrics.categorical_accuracy` aus `tf.keras.metrics`).

[»]

Gut zu wissen: Welche Kostenfunktion für welche Aufgabe?

Für Multiklassen-Klassifizierungsprobleme à la »*Was ist auf dem Bild zu sehen?*« wird `categorical_crossentropy` benutzt:

```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Für binäre Klassifizierungsprobleme à la »*Hund oder Katze?*« wird `binary_crossentropy` benutzt:

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Für ein Regressionsproblem kann z. B. `mse` benutzt werden:

```
model.compile(optimizer='rmsprop', loss='mse', metrics=['mse'])
```

2 *https://www.tensorflow.org/api_docs/python/tf/keras/optimizers*

Der erste Schritt unseres Modelltrainings ist bereits definiert. Nun kommen wir zur Trainingsschleife, die innerhalb der Funktion `model.fit()` implizit aufgebaut wird. In der Tat können Sie hier durch die Eingabe der Parameter `x`, `y`, `batch_size` und `epochs` jeweils die Eingabedaten, Ausgabedaten, die Anzahl der Batches und Epochen bestimmen:

```
model.fit(x=input_data, y=output_data, batch_size=96, epochs=100)
```

Ziemlich einfach, oder? Des Weiteren können Sie mit dem Parameter `verbose=1` den Fortschritt des Trainings in der Konsole ausgeben lassen. Wie Sie sehen, ist die Definition der Trainingsschleife mit Keras ein Kinderspiel!

6.4 Modelle evaluieren

Die Evaluation eines Modells kann in drei Varianten durchgeführt werden. Bei der ersten Variante geben Sie innerhalb von `model.fit()` einen sogenannten `validation_split` an:

```
model.fit(x=input_data, y=output_data, batch_size=96, epochs=100,
          validation_split=0.3)
```

Hierbei werden die Eingabedaten (hier `x`) in zwei Gruppen aufgeteilt: 70 % der Eingabedaten dienen dem Training und 30 % der Validierung.

[!]

Achtung: validation_split

Bei der Aufteilung mit `validation_split` werden die Daten nicht durchgemischt. Beachten Sie also, dass Sie die Daten zunächst z. B. mit dem Aufruf von `np.random.shuffle()` mischen.

Die zweite Variante besteht darin, wiederum innerhalb von `model.fit()` den Parameter `validation_data` mit den Validationsdaten-Tupeln (`input_validation_data` und `output_validation_data`) anzugeben, die Sie vorher selbst angelegt haben:

```
model.fit(x=input_data, y=output_data, batch_size=96, epochs=100,
          validation_data=(input_validation_data, output_validation_data))
```

Die dritte Variante besteht darin, die Funktion `model.evaluate()` aufzurufen:

```
evaluation_results = model.evaluate(input_test_data, output_test_data)
print("Loss: {}".format(evaluation_results[0]))
print("Accuracy: {}".format(evaluation_results[1]))
```

Die Evaluation wird erst nach dem Training durchgeführt und sollte immer mit Daten erfolgen, die das Modell noch nicht kennt. Denken Sie hier z. B. an eine Bildklassifizierung, die man mit neuen Bildern testen muss, um zu sehen, ob das Modell etwas Richtiges erkennen kann. Als Ergebnis von `model.evaluate()` werden die Loss- und die Accuracy-Metriken ausgegeben.

6.5 Modelle laden und exportieren

Keras-Modelle können für eine weitere Verwendung exportiert werden. Die Struktur eines Modells kann entweder als JSON- oder als YAML-Datei exportiert werden. Öffnen Sie dazu die Datei *chap_6/keras_load_save.py*.

6.5.1 Speichern des Modells als h5-Datei

Um ein Keras-Modell zu speichern, wird die Funktion `model.save()` benutzt. Diese exportiert die gesamte Struktur und die Parameter des neuronalen Netzes als **.h5*-Datei[3]. Unter *chap_6/addition_model.h5* finden Sie ein Modell aus einem sehr einfachen Beispiel. Dieses wurde mit folgender Zeile gespeichert:

```
addition_model.save("addition_model.h5")
```

Um ein Modell wiederzuherstellen bzw. von einer **.h5*-Datei zu laden, wird die `model.load_model()`-Funktion verwendet:

```
model = load_model('addition_model.h5')
```

Wenn das Modell geladen ist, können Sie weiter mit ihm arbeiten und z. B. die `predict()`-Funktion aufrufen.

```
# Modell wird neu geladen (von *.h5-Datei)
model = load_model('addition_model.h5')
```

3 Hierarchical Data Format, siehe auch *https://www.hdfgroup.org/solutions/hdf5/*


```
result = model.predict([[5, 5]])
# Das Ergebnis müsste ungefähr bei 10 liegen.
print("Ergebnis: {}".format(result))
```

Listing 6.3 Laden eines Keras-Modells im .h5-Format und Benutzung der »predict()«-Funktion

6.5.2 Speichern als SavedModel-Format

Alternativ kann das Modell in das SavedModel[4]-Format exportiert werden:

```
tf.saved_model.save(addition_model,"addition_model")
```

Hierbei wird die Modelldefinition in das angegebene Unterverzeichnis (hier *addition_model*) exportiert. In diesem Verzeichnis finden Sie neben den Unterverzeichnissen *variables* und *assets* eine Datei mit dem Namen *saved_model.pb* (siehe Abbildung 6.5). Diese Datei wird mit dem von Google entwickelten *Protocol Buffer Format*[5] definiert und beinhaltet die Definition des Graphen.

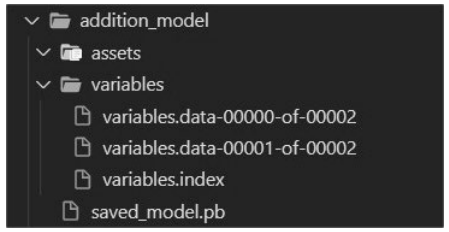

Abbildung 6.5 Struktur des exportierten Modells als SavedModel

Falls Sie neugierig sind und den Inhalt der *saved_model.pb* sichten möchten, können Sie dies mit der Applikation *Netron*[6] (siehe auch Abschnitt 7.8.1) realisieren. Sie werden den von TensorFlow erzeugten Graph für das Modell visualisiert bekommen wie in Abbildung 6.6.

Das SavedModel-Format ermöglicht nun die Benutzung des Modells in TensorFlow Lite, TensorFlow Serving, TensorFlow Hub oder sogar in TensorFlow.js, nachdem es konvertiert wurde (siehe auch *https://www.tensorflow.org/js/tutorials/conversion/import_saved_model*).

Mehr über das SavedModel-Format und die Export- bzw. Import-Möglichkeiten innerhalb von TensorFlow und Keras finden Sie unter *https://www.tensorflow.org/guide/saved_model*.

4 *https://www.tensorflow.org/guide/saved_model*
5 *https://developers.google.com/protocol-buffers/?hl=en*
6 *https://github.com/lutzroeder/netron*

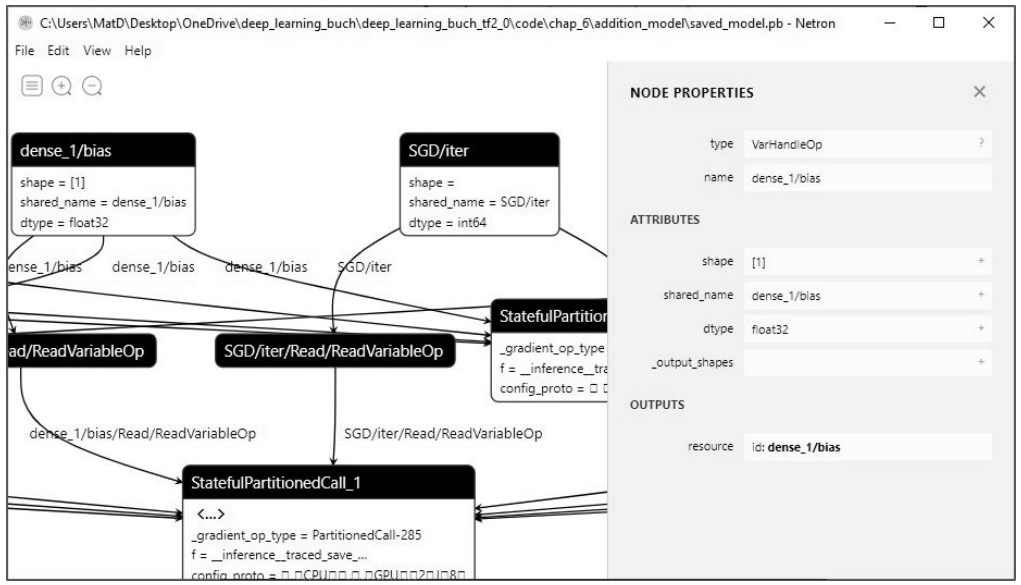

Abbildung 6.6 Ausszug des generierten Graphen in Netron

6.5.3 Separates Speichern der Struktur und Parameter des Modells

Es gibt Fälle, in denen Sie die Struktur eines Modells separat von seinen Parametern exportieren möchten, z. B. wenn Sie eine Visualisierung der Modellstruktur erzeugen wollen: Da würden die Gewichtungen und die Bias wenig nützlich sein. Aus diesem Grund bietet Keras ein separates Speichern von Struktur und Parametern an.

Um die Struktur als JSON- oder YAML-Datei zu exportieren, können Sie die Funktionen `model.to_json()` bzw. `model.to_yaml()` benutzen und die Struktur im jeweiligen Datei-Format speichern. Die Datei *chap_6/keras_load_save.py* gibt ein kompaktes Beispiel, wie ein Modell gespeichert und geladen werden kann.

```
# Weights werden gespeichert
addition_model.save_weights("addition_weights.h5")

# Die Struktur des Modells wird als JSON gespeichert
json_str = addition_model.to_json()

with open("addition_model.json", "w") as json_file:
    json_file.write(json_str)
```

Listing 6.4 Die Parameter des Keras-Modells werden in »addition_weights.h5« gespeichert und die Struktur des Modells wird in »addition_model.json« abgelegt.

Die Gewichtungen des trainierten Netzes werden mit der Funktion `model.save_weights()` als **.h5*-Datei gespeichert.

Nach dem Speichern können Sie folgenden Quellcode-Abschnitt aufrufen, um die Struktur und die Parameter des Modells zu laden:

```
# Das Modell wird mit der Kombination JSON und Weights geladen.
with open('addition_model.json', "r") as f:
    json_file_content = f.read()
model = model_from_json(json_file_content)
model.load_weights('addition_weights.h5')
# Das Ergebnis müsste ungefähr bei 5 liegen.
result = model.predict([[1, 4]])
print("Ergebnis: {}".format(result))
```

Listing 6.5 Laden eines Keras-Modells mithilfe der Dateien »addition_model.json« und »addition_weights.h5«

Alternativ können Sie mit der Funktion `model_from_yaml(yaml_file_content)` das Modell im YAML-Format laden.

Gut zu wissen: *.h5-Datei

In Kapitel 9, »Praxisbeispiele«, werden Sie sehen, dass Sie Keras-Modelle im **.h5*-Format mit folgender Zeile für TensorFlow.js exportieren lassen können – vorausgesetzt, Sie haben vorher `pip install tensorflowjs` ausgeführt:

```
tfjs.converters.save_keras_model(addition_model, "./addition_model")
```

6.6 Keras Applications

Damit Sie nicht jedes Mal bekannte Netze neu anlegen und konfigurieren müssen, bietet TensorFlow sogenannte *Keras Applications*. Das sind vorgefertigte und konfigurierte Deep-Learning-Modelle, die eine Modellarchitektur und vortrainierte Gewichtungen zur Verfügung stellen. Diese Modelle können direkt für die Vorhersage, Merkmalsextraktion und Feinabstimmung verwendet werden. Die einzelnen Gewichtungen und Parameter werden ebenfalls bei der Instanziierung des Modells automatisch heruntergeladen.

In Tabelle 6.1 wurden diese direkt benutzbaren Modelle aufgelistet (samt Genauigkeit und Gesamtanzahl der benutzten Layer). In der Datei *chap_6/keras_applications_list.py* haben wir ein paar dieser Keras Applications instanziiert und ihre Strukturen mit `model.summary()` ausgeben lassen.

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
InceptionV3	92 MB	0.788	0.944	23,851,784	159
InceptionResNetV2	215 MB	0.804	0.953	55,873,736	572
MobileNet	17 MB	0.665	0.871	4,253,864	88
DenseNet121	33 MB	0.745	0.918	8,062,504	121
DenseNet169	57 MB	0.759	0.928	14,307,880	169
DenseNet201	80 MB	0.770	0.933	20,242,984	201

Tabelle 6.1 Auszug aus einer Liste der Modelle, die direkt mit Keras benutzt werden können (Quelle: https://keras.io/applications bzw. https://www.tensorflow.org/api_docs/python/tf/keras/applications)

Hinweis: Vorgefertigte Modelle

Die vorgefertigten Modelle sind eigentlich keine direkten Bestandteile von TensorFlow, sondern werden nachträglich als **.h5*-Datei heruntergeladen. Das einmalige Herunterladen wird während der Ausführung der Python-Datei angezeigt und kann je nach Modellgröße und Geschwindigkeit Ihres Internetanschlusses etwas länger dauern.

6.7 Keras Callbacks

Es ist während eines Trainings immer wichtig, die Metriken im Auge zu behalten. Hierbei werden Ihnen die *Keras Callbacks* behilflich sein, denn durch diese können Funktionen während der Ausführung des Trainings getriggert werden und somit wichtige Informationen z. B. am Anfang einer Epoche oder am Ende eines Batchs sammeln und weitergeben. Keras enthält eine Reihe von Callbacks – unter anderem:

- *ProgbarLogger* – gibt die Metriken auf der Standardausgabe bzw. der Konsole aus.
- *ModelCheckpoint* – speichert das Modell nach jeder Epoche bzw. jedem Durchgang und kann dazu benutzt werden, Zwischenversionen der Modelle zu speichern, wenn komplexere Modelle inkrementell antrainiert werden sollen.
- *TensorBoard* – generiert eine Logdatei, die nachträglich über TensorBoard aufgerufen und visualisiert werden kann.
- *CSVLogger* – speichert bei jeder Epoche die Werte der Metriken in eine CSV-Datei.
- *RemoteMonitor* – überträgt am Ende einer Epoche mithilfe einer HTTP-POST-Methode Werte an einen Server.
- *LambdaCallback* – ermöglicht die Erstellung eigener Callbacks.
- *EarlyStopping* – das Training wird gestoppt, wenn sich eine angegebene Metrik (z. B. `loss` oder `val_loss`) während des Trainings für eine gewisse Anzahl von Epochen nicht mehr verbessert.

Es existieren noch weitere Callbacks (siehe *https://www.tensorflow.org/api_docs/python/tf/keras/callbacks*), deren Funktionsweisen noch deutlicher auf der offiziellen Keras-Website vorgestellt (siehe *https://keras.io/callbacks*) werden.

Callbacks werden innerhalb der Funktion `model.fit()` als Parameter (`callbacks` mit als Array von Callback-Instanzen) angegeben:

```
# CSVLogger-Callback
csv_logger_cb = CSVLogger("xor_log.csv", separator=',', append=False)

# Eigenes Callback
my_cb = MyCallback()

xor_model.fit(input_data, output_data, batch_size=1, epochs=6000, verbose=1, ↩
              callbacks=[csv_logger_cb, my_cb])
```

Listing 6.6 Beispiele einer Benutzung von Callbacks

Im ersten Fall wird das `CSVLogger ()`-Callback benutzt, das für jede Epoche die Metriken in eine CSV-Datei speichert, sowie ein selbst definiertes Callback, `MyCallback()`, das die Dauer eines Trainings misst.

Ein eigenes Callback, abgeleitet von der Klasse `Callbacks` (hier `MyCallback`), kann wie folgt definiert werden, indem die Funktionen `on_train_begin()` und `on_train_end()` angelegt werden:

```
class MyCallback(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        print("Anfang vom Training")
        self.begin_time = time.time()
```

```
    def on_train_end(self, logs={}):
        print("Ende vom Training")
        self.training_duration = time.time() - self.begin_time
        print("Dauer des Trainings: {} sec.".format(self.training_duration))
```

Listing 6.7 Definition eines eigenen Callbacks

Mit dem Callback `EarlyStopping` kann ein Abbruchkriterium des Trainings definiert werden (und somit Rechenzeit gewonnen werden), z. B. wenn sich die Loss-Metriken über die Epochen nur noch sehr minimal verändern. Im folgenden Beispiel wird während 20 Epochen (gesetzt durch den Parameter `patience=20`) überprüft, ob sich der Wert von `loss` nur noch minimal um 0,0001 (Parameter `min_delta`) verändert hat:

```
early_stopping_cb = EarlyStopping(monitor="loss",min_delta=0.0001,patience=20)
```

[!]

Abbruchkriterien gut auswählen

Seien Sie jedoch vorsichtig mit der Benutzung von `EarlyStopping`, und überprüfen Sie sorgfältig immer die Abbruchkriterien. Ansonsten kann Ihr Modell unvollständig trainiert sein.

Sie werden mehr über die Benutzung weiterer Callbacks (insbesondere von `LambdaCallback` und `RemoteMonitor`) in Abschnitt 7.5 erfahren, in dem es um die Visualisierung der Keras-Metriken geht.

6.8 Projekt 1: Iris-Klassifikation mit Keras

In Kapitel 4, »Python und Machine-Learning-Bibliotheken«, haben wir gesehen, wie man mit der Python-Bibliothek Scikit-learn anhand von vier botanischen Eigenschaften Schwertlilien (*Iris*) in drei Kategorien klassifizieren kann. In diesem Abschnitt werden Sie lernen, wie Sie diese Aufgabe in ein paar Schritten mit Keras lösen können.

[»]

Quellcode

Den Quellcode zu diesem Projekt finden Sie in den Dateien *chap_6/keras_iris_classification.py* und *chap_6/keras_iris_classification_with_evaluation.py*.

6.8.1 Dataset laden

Das Dataset befindet sich in der Datei *chap_6/data/iris.csv*. Diese Daten werden mit der `np.loadtxt`-Funktion von NumPy als Array geladen. Auch hier werden die drei

Kategorien `Iris-setosa`, `Iris-versicolor` und `Iris-virginica` auf die Werte `0`, `1` und `2` indiziert:

```
import numpy as np
# Laden der iris.csv-Datei
iris_data = np.loadtxt("data/iris.csv",delimiter=",",dtype="str",skiprows=1)
# Wir erstellen die Kategorien:
iris_label_array = ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]
label_index = 0
for label in iris_label_array :
    print(label)
    # Wenn eines der Label innerhalb von iris_data gefunden wird,
    # wird dieses durch den label_index ersetzt,
    # z. B. werden alle Einträge von Iris-versicolor durch den Wert 1
    # ersetzt.
    iris_data[np.where(iris_data[:,4]==label), 4] = label_index
    label_index = label_index + 1

iris_data = iris_data.astype("float32")
```

Nun sind alle Daten im Array `iris_data` enthalten. Als Eingabe für unser Netz sind nur die 4 ersten Spalten wichtig (d. h. nur die Spalten *sepal-length, sepal-width, petal-length* und *petal-width*).

```
input_data = iris_data[:,0:4] # Spalten 0 bis 4 werden extrahiert
```

Dem gleichen Prinzip folgend, wird für die Ausgabe unseres Netzes die letzte Spalte benutzt:

```
output_data = iris_data[:,4].reshape(-1, 1)
# Die 4. Spalte wird extrahiert output_data = output_data.reshape(-1, 1)
# Array von 1D-Array
output_data = tf.keras.utils.to_categorical(output_data,3)
```

Da wir es hier mit einer Klassifikationsaufgabe zu tun haben, werden wir die Keras-Funktion `to_categorical()` benutzen, die die Zahlen 0, 1 und 2 in Arrays der Form [1. 0. 0.], [0. 1. 0.] und [0. 0. 1.] umwandelt. Dies bezeichnet man im Machine Learning auch als *One Hot Encoding*. Das ist sehr wichtig: Würde man willkürlich festlegen, dass z. B. *Iris-setosa = 1* und z. B. *Iris-virginica = 2* ist, könnte unser Modell denken, dass *Iris-setosa < Iris-virginica,* was eigentlich überhaupt keinen Sinn macht – und das Training wäre somit komplett falsch! Mit dem One Hot Encoding wird dieses Problem gelöst.

6.8.2 Modell erstellen

Das Modell für die Klassifikation ist sehr schlicht und übersichtlich:

```
import tensorflow as tf
import numpy as np
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.layers import Dense, Activation
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from pprint import pprint
import numpy as np

[…]
# Aufbau des Modells mit Keras
iris_model = Sequential()
iris_model.add(Dense(5, input_shape=(4,), activation="tanh"))
iris_model.add(Dense(24, activation="relu"))
iris_model.add(Dense(3, activation="softmax"))
[…]
```

Listing 6.8 Auszug der Datei »chap_6/keras_iris_classification.py«

Wir haben hier drei `Dense`-Schichten mit jeweils 5, 24 und 3 Neuronen definiert. Eine `Dense`-Schicht ist als eine vollständig verbundene (*fully connected*) Schicht definiert. Die Eingabeschicht erwartet eine Struktur mit 4 Elementen. Die benutzte Aktivierungsfunktion ist `ReLu`. Die letzte Schicht wird uns 3 Werte zurückliefern bzw. Wahrscheinlichkeiten, dass eine bestimmte Klasse mit den Eingabewerten zugeordnet wurde.

6.8.3 Modell trainieren

Das Training des Modells wird mit folgendem Code gestartet:

```
sgd = SGD(lr=0.001)
iris_model.compile(loss="categorical_crossentropy", optimizer=sgd, ➲
                   metrics=["accuracy"])
iris_model.fit(x=input_data, y=output_data, batch_size=10, ➲
               epochs=500, verbose=1)
```

6.8.4 Modell evaluieren

Um unser Modell evaluieren zu können, müssen wir eine leichte Veränderung an den Eingabedaten durchführen. Wie Sie der vorbereiteten Datei *chap_6/keras_iris_*

classification_with_evaluation.py entnehmen können, werden nun die Eingabedaten in zwei »Gruppen« aufgeteilt: in eine für das Training und in eine für das Testen bzw. Evaluieren. Hierfür benutzen wir die Scikit-learn-Bibliothek mit der Funktion `train_test_split()` mit dem Wert `0.20` für den Parameter `test_size`:

```
iris_train_input, iris_test_input, iris_train_output, iris_test_output = ➲
  train_test_split(input_data, output_data, test_size=0.20)
```

Das Training wird mit den Daten von `iris_train_input` und `iris_train_output` wie folgt gestartet:

```
iris_model.compile(loss="categorical_crossentropy", ➲
                   optimizer=sgd, metrics=["accuracy", metrics.mae])
iris_model.fit(x=iris_train_input, y=iris_train_output, batch_size=10, ➲
               epochs=500, verbose=1)
```

Nach dem Training kann die Funktion `model.evaluate()` diesmal mit den Testdaten (`iris_test_input` und `iris_test_output`) benutzt werden:

```
# Evaluation auf Testdaten
evaluation_results = iris_model.evaluate(iris_test_input, iris_test_output)

print("Loss: {}".format(evaluation_results[0]))
print("Accuracy: {}".format(evaluation_results[1]))
print("Mean Absolute Error: {}".format(evaluation_results[2]))

# Ausgabe
# Loss: 0.12635588645935059
# Accuracy: 0.9666666388511658
# Mean Absolute Error: 0.07339978963136673
```

Die Variable `evaluation_results` beinhaltet die berechneten Metriken für den Loss, die Accuracy und den Mean Absolute Error (diese Metrik wurde in `model.compile()` angegeben). Das Ergebnis sieht sehr gut aus!

6.8.5 Modell benutzen

Nun ist die Zeit gekommen, um zu sehen, ob das Modell auch mit unbekannten Eingabedaten die richtige Iris-Klasse bestimmen kann. Ähnlich wie in Kapitel 4, »Python und Machine-Learning-Bibliotheken«, werden wir ein Testdatenset auswählen, das in `test_data` definiert ist:

```
# Test
test_data = np.array([[5.1,3.5,1.4,0.2], [5.9,3.,5.1,1.8], ➲
           [4.9,3.,1.4,0.2], [5.8,2.7,4.1,1.]])
predictions = iris_model.predict(test_data)
```

```
index_max_predictions = np.argmax(predictions, axis=1)

for i in index_max_predictions:
    print("Iris mit den Eigenschaften {} gehört zur Klasse: {}".format(
       test_data[i], iris_label_array[i]))
# Ausgabe
# Iris mit den Eigenschaften [5.1 3.5 1.4 0.2] gehört zur Klasse: Iris-setosa
# Iris mit den Eigenschaften [4.9 3.  1.4 0.2] gehört zur Klasse: Iris-virginica
# Iris mit den Eigenschaften [5.1 3.5 1.4 0.2] gehört zur Klasse: Iris-setosa
# Iris mit den Eigenschaften [5.9 3.  5.1 1.8] gehört zur Klasse: Iris-versicolor
```

Die Funktion `iris_model.predict()` liefert ein Array von Wahrscheinlichkeiten zurück. Um zu wissen, zur welchen tatsächlichen Klasse die Eingabedaten gehören, suchen wir mit `np.argmax()` den Index mit der größten Wahrscheinlichkeit. Dieser Index wird dann die passende Klasse sein, die wir in einem letzten Schritt textuell ausgeben. Den kompletten Quellcode für diese Aufgabe finden Sie in *chap_6/keras_iris_classification.py*.