Hochschule
für Technik
Stuttgart

Bachelor-Arbeit

vorgelegt von
Halil Kocak
Bachelor-Studiengang Informatik
Fakultät Vermessung, Informatik und Mathematik
Wintersemester 2021/22

Erkennung von Kleiderstücken anhand CRISP für Data Mining

Erstprüfer: Dr. Prof. Jörg Homberger
Zweitprüfer: Dr. Prof. Gero Lückemeyer

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Es war schon immer die Frage ob Maschinen jemals selbst, anhand vorhandenen Informationen, Entscheidungen treffen oder wie ein Mensch Regeln aus bestimmten Prozessen extrahieren können [Cho18](K.1.1.1). Diese Fragestellung ist vor allem in der heutigen Zeit, in der das Thema Digitalisierung voran getrieben wird, von großer Bedeutung. Auch setzen Unternehmen immer mehr auf Techniken aus dem Bereich des Maschinellen Lernens (ML) und verwenden diese um ihre Prozesse zu digitalisieren und somit zu optimieren. Es existieren verschiedene ML-Techniken, wie zum Beispiel die Klassifikation, Regression oder Deep Learning. Letzteres steht im Mittelpunkt vieler Anwendungsszenarien. Anwendungsfelder für Deep Learning sind vor allem, Bild- und Videoanalyse, sowie die Sprachverarbeitung. Diese werden in verschiedenen Branchen, wie zum Beispiel der Medizin, der Automobilindustrie oder im Kundendienst eingesetzt [FG18]. Auch im Bereich Online-Retail finden ML-Techniken einen relevanten Einsatz. Im Rahmen dieser Bachelor-Arbeit liegt der Fokus vor allem in diesem Bereich und wie die Bildanalyse hier unterstützen kann. Im Bereich Online-Retail können zum Beispiel von Kunden bestellte Artikel zurück gesendet werden. Diese Artikel müssen dann wieder richtig gekennzeichnet und neu einsortiert werden. Dieser manueller Prozess kann durch den Einsatz von einer Bildanalyse bzw. Bilderkennung optimiert und automatisiert werden. Dadurch können zum Beispiel wieder eingegangene Waren automatisch erkannt, entsprechend gelabelt und für den erneuten Verkauf vorbereitet werden. Diese Problemstellung wird in einem späteren Abschnitt der Arbeit, im Rahmen des Business Understanding des CRISP-DM Standards (Cross Industry Standard Process for Data Mining), näher definiert.

Diese Bachelor-Arbeit beinhaltet die Erkennung von Objekten auf Bildern durch den Einsatz von Künstlicher Intelligenz. Dabei wird die Disziplin Deep Learning unter Einsatz von Convutional Neural Network verwendet. Um diese zu verstehen werden zuerst allgemeine Informationen zur Künstlichen Intelligenz und dessen Bestandteile und Algorithmen benötigt. Zu Beginn ist es wichtig zu wissen, welche Probleme die Künstliche Intelligenz lösen kann und welche Bestandteile für diese Arbeit benötigt werden. In diesem Zusammenhang wird das Verfahren Convutional Neural Network behandelt, welches zu den Deep Learning Methoden gehört und ei-

ne Teilmenge der Maschinellen Lernalgorithmen bildet. Außerdem wird im Rahmen dieser Arbeit das CRIPS-DM Modell vorgestellt und verwendet, welches als Standard für Data Mining Modelle definiert wurde.

Der Aufbau der Arbeit beginnt mit der Einführung in das Maschinelle Lernen und allen wichtigen Komponenten, wie zum Beispiel die in der Arbeit eingesetzten Verfahren und wichtige Evaluationsmetriken für die Bewertung der Experimente. Danach erfolgt ein Einblick in das CRISP-DM Modell, welches für das bearbeitete Data Mining Experiment als Vorgehensmuster eingesetzt wird. Im Anschluss folgen die Experimente. In diesem Abschnitt wird das CRISP-DM Modell angewendet und die einzelnen Phasen des Modells bearbeitet. Hier wird unter anderem die Problemstellung detailliert erläutert, der verwendete Datensatz beschrieben und die implementierten Modelle erklärt und evaluiert. Als letztes folgt eine Zusammenfassung der gesamten Arbeit und ein möglicher Ausblick für die Fortsetzung dessen.

2 Maschinelles Lernen

Maschinelles Lernen stellt ein Teilgebiet der Künstlichen Intelligenz dar. Dabei werden durch den Einsatz von Maschinen ein Verhalten aus den Daten extrahiert, wodurch die Maschine selbst Lernen kann. Dies Geschieht unter Verwendung verschiedener Techniken und Algorithmen. Während der Lernphase versucht die Maschine, Regelmäßigkeiten aus den Daten zu erkennen und diese zu verallgemeinern. Nachdem die Lernphase beendet ist, kann die Maschine eine Vorhersage aus den aktuellen Daten treffen [Fro21](S.2-3).

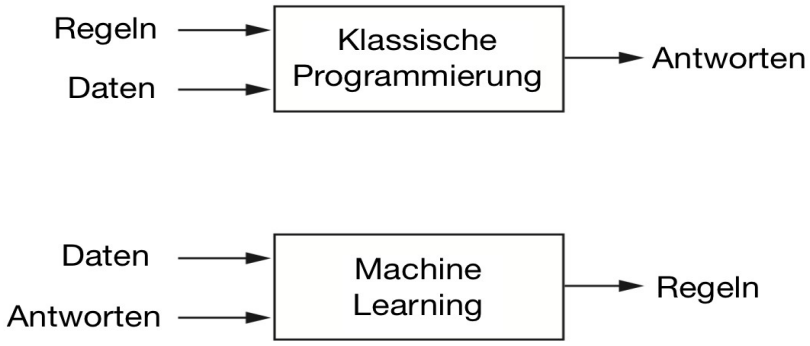


Abbildung 2.1: Maschinelles Lernen vs. Klassische Programmierung [Cho18](K.1.1.2

In der Abbildung 2.1, ist der Unterschied zwischen der Klassischen Programmierung und den Maschinellen Lernalgorithmen zu sehen. Bei der Klassischen Programmierung werden Regeln definiert und implementiert. Anhand dieser Regeln werden die Daten verarbeitet, die dann zu einer Antwort führen. Anders ist es aber bei den Maschinellen Lernalgorithmen. Hier werden Daten und dessen Antworten als Eingabedaten betrachtet, wodurch die Maschine für sich die Regeln definiert [Cho18](Kapitel 1). Dazu haben alle Maschinelle Lernverfahren zwei Phasen. Die erste Phase ist die Lernphase, in der es lernt, aus den Eingabedaten ein Muster zu extrahieren. Nachdem diese Phase beendet ist, wird die Anwendungsphase gestartet. Diese Phase wird sowohl für die Evaluation des Künstliche Intelligenz als auch für die Anwendung genutzt. Es ist zu beachten das die Künstliche Intelligenz auch in der Anwendungsphase lernen kann [Cho18](Kapitel 2.1).

Um Entscheidungen zu treffen, werden Eingabedaten benötigt, die der Maschine die nötigen Informationen liefern. Die Eingabedaten beinhalten auch die Lösung des Problems und somit auch den Wert, der von der Maschine vorhergesagt werden soll. Damit die Maschine nach Effizienz bewertet werden kann, ist eine Messung der Algorithmen erforderlich [Cho18](Kapitel 1.1.3).

Für die Lösungen der Probleme werden verschiedentliche Maschinelle Lernalgorithmen verwenden. Die Entscheidung welches benutzt werden soll hängt von den Eingabedaten und dem Ziel ab. Im Rahmen dieser Arbeit liegt der Fokus auf dem Convutional Neural Network Verfahren, welches zur Kategorie des überwachten Lernens gehört [Lub19].

2.1 Überwachtes Lernen

Das überwachte Lernen gehört zu den Lernverfahren des maschinellen Lernens und umfasst die Klassifikation und Regression. Bei einem überwachten Lernen werden Eingabedaten, die Beispiele beinhalten, einem bekannten Zielwert zugeordnet. Dadurch sind die Zielwerte bekannt und somit können die Fehler, bei einer falschen Klassifikation, während dem Lernprozess entdeckt werden [Cho18](K.4.1.2).

2.2 Klassifizierung

In diesem Abschnitt wird das Ziel der Klassifizierung beschrieben. Dabei handelt es sich um die richtig vorhergesagten Klassen durch gegebene Datenpunkte. Datensätze die zu einer Klasse gehören, nennt man klassifizierte Beispiele. Diese Beispiele werden je nach Verwendung für das Trainieren, Validieren oder das Testen der Künstliche Intelligenz genutzt. Während des Trainingsvorganges erkennt ein Modell vorhandene Muster aus den Daten und kann daraus später neue Daten klassifizieren. Der Klassifizierungsvorgang ist in zwei Schritten unterteilt. In dem ersten Schritt findet, wie bereits erwähnt, die Trainingsphase statt. Durch vorhandene Beobachtungen, die bereits einer Klasse zugeordnet sind, lernt der Klassifizierer und erkennt Muster. Damit die Leistung des Trainingsvorganges bewerten werden kann, findet parallel eine Validierung statt. Dafür werden dann die Validierungsdaten eingesetzt. Im zweiten Schritt findet die Testphase statt, in die Beobachtungen einfließen, bei denen es sich um nicht klassifizierte Beispiele handelt. Diesen bisher unklassifizierten Beobachtungen werden in diesem Schritt die Klassen zugeordnet. Hier findet also die eigentliche Klassifizierung statt. Sind nur zwei Klassen für die zuordnung vorhanden, handelt es sich um eine binäre Klassifizierung. Sind mehr als zwei Klassen vorhanden, ist

das Klassifizierungsproblem eine Multi-Class Klassifizierung. Im Rahmen dieser Arbeit wird eine Multi-Class Klassifizierung (Mehrklassen Klassifizierung) behandelt [Fro21] (S.21) [Asi18] [Sch19] (K.2).

2.3 Überanpassung (Overfitting)

Im Rahmen des maschinellen Lernens kann es durchaus passieren, dass sich ein Modell während seiner Lern- bzw. Trainingsphase zu sehr an die gegebenen Trainingsdaten orientiert. In diesem Fall spricht man von einer Überanpassung, auch bekannt als Overfitting. Der zweite Fall, der auftreten kann ist, dass ein Modell zu wenig Zusammenhänge erkennt bzw. wiedergibt. Hier spricht man von Underfitting. In dieser Arbeit wird das Problem des Overfittings näher betrachtet, da dies auch im Rahmen der Experimente von Bedeutung sein wird.

Bei der Darstellung komplexer Zusammenhänge, kann es von Vorteil sein Neuronale Netze für die Modellierung zu verwenden. Diese haben in der Regel sehr viele freie Parameter, die auch bekannt sind unter dem Namen Gewichte. Der große vorhandene Parameterraum kann aber auch Gefahren mit sich bringen, unter anderem das Problem des Overfittings. Neuronale Netze lernen während der Trainingsphase nicht nur die Trainingsdaten perfekt abzubilden sondern lernen auch Schwankungen in den Daten, Rauschen oder Fehler auswendig. Kommen dann neue Daten ins Spiel, aus denen Prognosen abgeleitet werden sollen, kann es, aufgrund der gespeicherten störenden Einflüsse, zu Verschlechterungen in den Prognosen kommen. Damit findet eine Überanpassung des Modells an die Trainingsdaten statt. Dieses Problem kann auch bei einer polynomialen Regression auftreten. In der nächsten Abbildung sind drei Modellarten in Bezug auf eine Regression zu sehen.

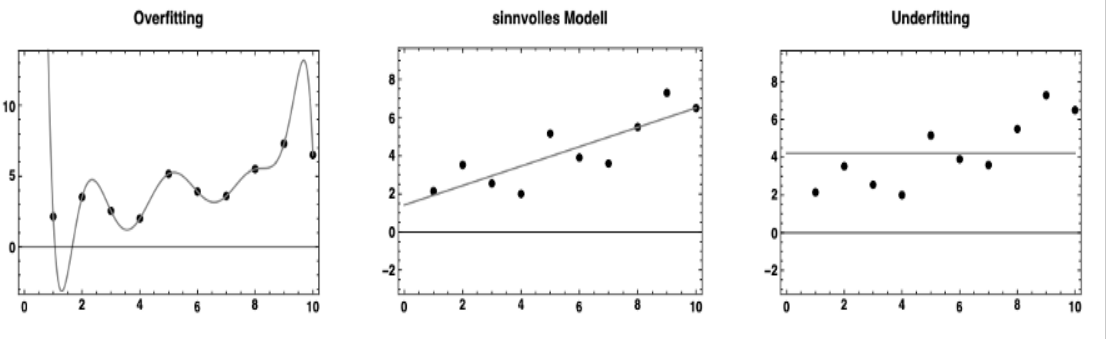


Abbildung 2.2: Darstellung des Overfitting [Vog21] (S.341)

Die Bilder stammen aus einem linearen Prozess und beinhalten Fehler. Das linke

Bild (Overfitting) stellt dar, dass das Modell auch jede Abweichung perfekt erkennt. Das Bild rechts zeigt, dass das Modell die Zusammenhänge zwischen den Inputdaten und Zielvariablen überhaupt nicht erkennt. In dem mittleren Bild ist ein Modell zu sehen, das sich gut auf neue Daten übertragen lässt [Vog21] (S.341).

Die einfachste Lösung des Problems ist, dass die vorhandenen Daten in drei Datensätze aufgeteilt werden. Somit kann man beobachten, wie sich das Modell auf unbekannte Daten übertragen lässt und konzentriert sich nicht nur auf das Lernverhalten des Modells. Der erste Datensatz wird dann zum Trainieren, der zweite Datensatz für die Validierung und der letzte Datensatz für das Testen verwendet. Diese Aufteilung wurde bereits näher im vorherigen Abschnitt erläutert. Nach der Aufteilung der Daten wird der Fehler auf den Trainingsdaten zusammen mit dem Fehler auf den Validierungsdaten beobachtet. Die Trainingsphase bzw. das Lernen wird gestoppt sobald der Validierungsfehler das Minimum erreicht. Dieser Vorgang wird Early Stopping genannt und wird in der nächsten Abbildung veranschaulicht.

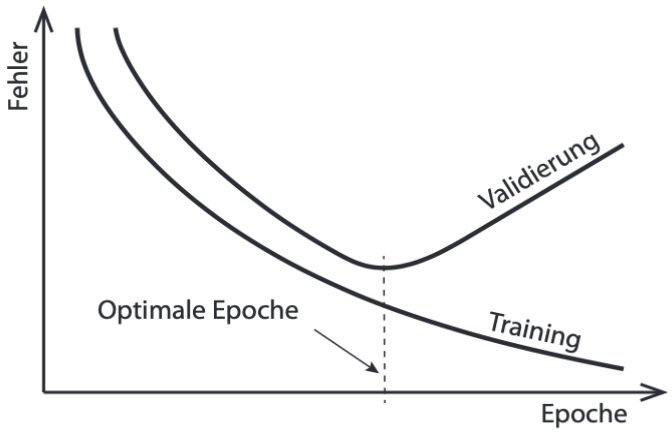


Abbildung 2.3: Early Stopping [Vog21] (S.342)

Eine weitere Möglichkeit ist die Verwendung eines Dropout-Layers. Diese Schicht veranlasst das Herausfallen von Neuronen in einem neuronalen Netz. Die Auswahl der eliminierten Neuronen passiert zufällig. Dabei werden auch die mit dem Neuron verbundenen eingehende und ausgehende Verbindungen vorübergehend entfernt. Dieser Vorgang ist in nachfolgender Abbildung dargestellt [Sal14]

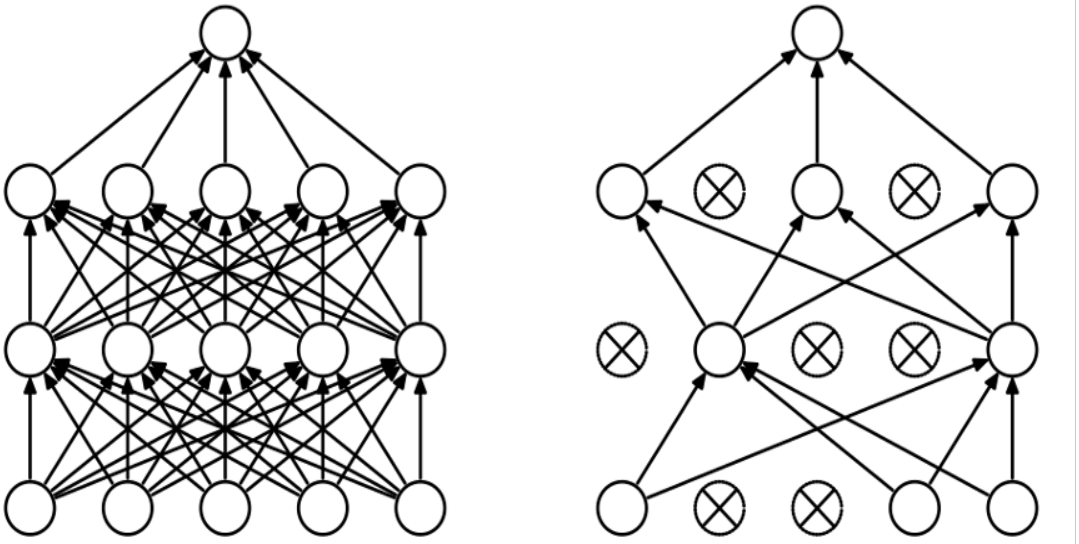


Abbildung 2.4: Neuronales Netz mit Dropout-Layer [Sal14]

Dabei ist auf dem linken Bild ein Neuronales Netz mit zwei Hidden Schichten zu sehen. Rechts ist ein Beispiel abgebildet, wie ein Netz mit eliminierten Neuronen durch die Anwendung des Dropout-Layers aussehen kann. Auch die entfernten Verbindungen werden hier deutlich. Neuronen, die während der Trainingsphase nicht entfernt werden, werden hochskaliert, damit die Summe über alle Eingaben unverändert bleibt. Im Rahmen der Experimente wird, um Overfitting zu vermeiden, ein Dropout-Layer eingesetzt [ten22].

2.4 Künstliche Neuronale Netze

Im folgenden Abschnitt werden die Bestandteile von künstlichen Neuronalen Netzen definiert und der Aufbau beschrieben.

2.4.1 Neuronales Netz

Auf der linken Seite der Abbildung 2.5, befinden sich die Eingabedaten die als $x_0, x_1, ..., x_n \in \vec{X}$ definiert sind, wobei $\vec{X}$ den Vektor $\mathbb{R}^n$ darstellt. Jede einzelnen Eingabe $x_0, x_1, ..., x_n$ wird mit seinem Gewicht $w_{n,j}$ multipliziert, wobei $j \in [0, AnzahlNeuronen]$ gilt. Nachdem multiplizieren mit den Gewichten werden alle berechneten Werte an die Übertragungsfunktion übergeben. Dieser Ablauf ist dabei an eine echte Nervenzelle angelehnt, indem Neuronen aus anderen Nervenzellen, dieses Neuron anregen und auf diese übertragen können. Die Übertragungsfunktion

summiert diese Werte zusammen und leitet sie dann an die Aktivierungsfunktion weiter. Die Aktivierungsfunktion beinhaltet genauso ein Schwellenpotenzial mit der wir auch Schwellenwerte prüfen können. Eine einfache Aktivierungsfunktion wäre hierbei das Alles-oder-nichts-Prinzip. Mit diesem Prinzip ist gemeint, dass bei Überschreitungen des Schwellwertes das Neuron aktiviert und unterhalb des Schwellwertes inaktiv gestellt wird [Fro21](S.169).

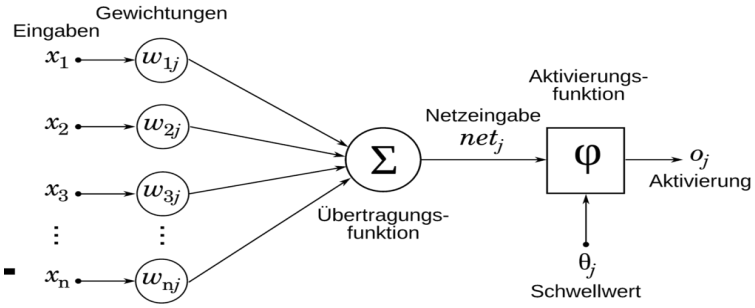


Abbildung 2.5: Modell eines Neurons [Abbe]

Zunächst ist wichtig wie die Vernetzungen und Verknüpfungen der Neuronen aussehen und funktionieren. In Abbildung 2.6 wird deutlich, dass ein einfaches Künstliches Neuronales Netz aus drei Schichten besteht. Die erste Schicht ist dabei die Eingabeschicht (engl. Inputlayer), die die Eingabe der Daten verarbeitet und wiederum mit jedem Neuron aus der verborgenen Schicht (engl.= Hiddenlayer) vernetzt ist. Durch die Ausgabeschicht werden Regeln, die aus der verborgenen Schicht extrahiert werden, bereitgestellt. Diese Neuronen werden zur Evaluation des Modells benutzt [Fro21](S.169).

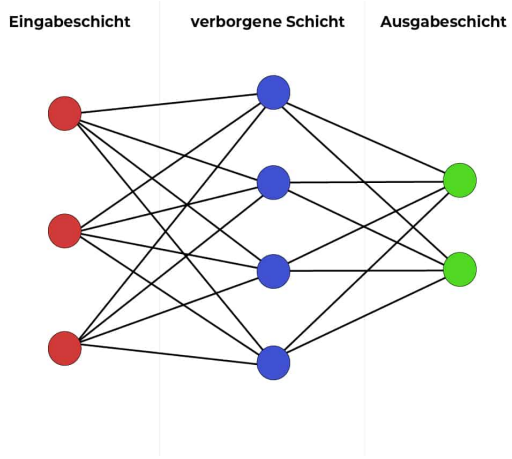


Abbildung 2.6: Schichten der Künstliche Neuronale Netze [Abbf]

2.4.2 Feed-Forward Propagierung

Das neuronale Netz durchläuft bei jedem Dateineintrag einmal alle Schichten. Verwendet wird dafür, dass im Abbild 2.5 vorgestellte Modell.

Mathematische Beschreibung eines Feed-Forward Netzes

In den vorherigen Abschnitten wurden die Bestandteile und deren Funktionen näher erklärt. An dieser Stelle wird das einfache neuronale Netz beschrieben. Dafür wird zuerst die Arbeitsweise zwischen den Schichten definiert:

$$f : \mathbb{R}^{D_0} \to \mathbb{R}^{D_l} \ , \ f(\overrightarrow{X}) = (f_l \circ f_{l-1} \circ ... \circ f_1)(u),$$

wobei f_l die Schichten sind und für jedes L $\in \{ \ 0, 1, 2, ..., l \ \}$ die Funktion,

$$f : \mathbb{R}^{D_{L-1}} \to \mathbb{R}^{D_0} \ , \ f_l(\overrightarrow{X}) = \phi(w^l * \overrightarrow{X} + b^l), \text{ beschreibt.}$$

Wobei $\overrightarrow{X}$ der Vektor für die Eingabedaten (engl. Inputdata), $w^l \in \mathbb{R}^{(D_l \times D_{l-1})}$ die Gewichte, welche die Verbindungen (Abbild 2.6) von einem Neuron zu anderen

Neuronen gewichten und b^l der Verzerrungsvektor (engl. Bias) der Schicht l ist. Des weiteren ist ϕ die Aktivierungsfunktion der Schicht l [Pla21] (S.232 - 234).

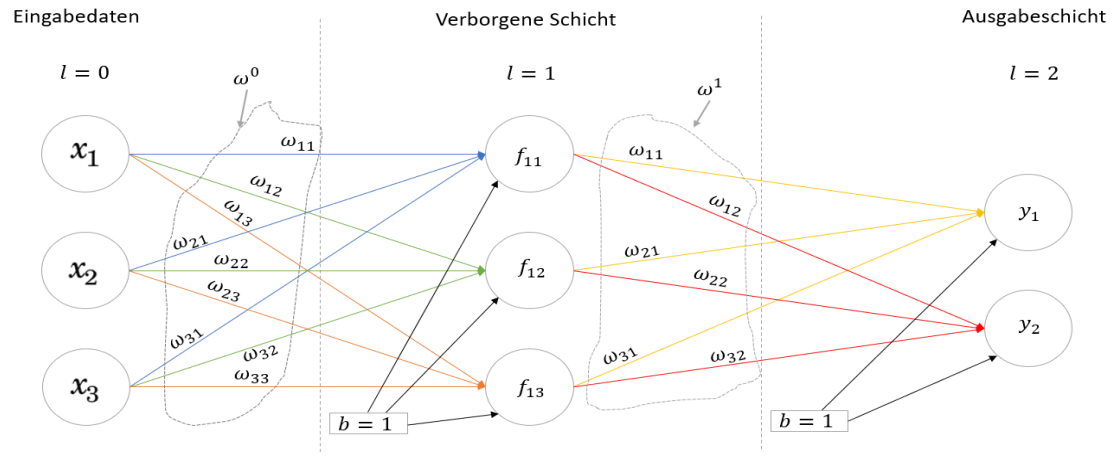


Abbildung 2.7: Feed-Forward Netz [Hal22]

Die Gewichte werden zunächst zufällig vergeben und mit dem Lernverfahren, das während der Rückpropagierung stattfindet, optimiert. Damit das Neuronale Netz Abweichungen erkennen kann, werden Bias verwendet, die diese Abweichung darstellen. Bevor mit der Trainingsphase begonnen wird, sollten folgende Elemente festgelegt werden. Zunächst ist die Verlustfunktion zu definieren, welche die Leistung der Trainingsdaten beschreibt. Durch die Verlustfunktion kann das Neuronale Netz die Gewichte optimieren. Als nächstes sollte die Menge an Trainingsdaten und Testdaten festgelegt werden [Cho18] (K.2.1 Listing 2.2). An dieser Stelle werden beispielhaft die Schritte der Feed-Forward Propagation in Abbild 2.7 betrachtet. Zunächst wird die Eingabeschicht mit den Eingabedaten gefüttert, wobei diese normalisierte Werte beinhalten sollten. Die zufällig generierten Gewichte stellen die Gewichtungen für die Neuronen der nächsten Schicht dar. Nun wird das, in Abbild 2.7 definierte, Modell für jedes Neuron aus der nächsten Schicht angewendet. Somit hat das erste Neuron aus der Schicht $l = 1$ folgende Berechnung.

$$f_{11} = \phi[(x_1 * w_{11}) + (x_2 * w_{21}) + (x_3 * w_{31})] + b^1$$

Allgemein lässt sich die Feed-Forward Propagierung wie folgt beschreiben:

$$f = \sum_{n=0}^{n}[\phi(x_n * w_{nj} + b^j)]$$

, wobei n die Anzahl an Neuronen in der vorherigen Schicht sind und j die Anzahl an Neuronen in der aktuellen Schicht. Somit kann das Neuronale Netz mit den bestmöglichen Gewichten Prognosen, anhand der Eingabedaten, liefern. Die Optimierung der Gewichte werden im Abschnitt 2.4.4 Lernverfahren näher beschrieben.

2.4.3 Eingabedaten

Die Eingabedaten $x_1, ..., x_n$ stellen die Merkmale (oder Attribute) dar, welche die Eigenschaften beinhalten, aus denen das Neuronale Netz lernen kann. Zu beachten sind dabei die Datentypen der Eingabedaten und die Normierung dieser Daten. Dabei gibt es unterschiedliche Datentypen. Im Rahmen dieser Arbeit ist der Fokus auf Vektor- und Bilddaten gerichtet [Cho18] (K.2.2.8).

Vektordaten

Die Vektordaten werden mathematisch als $\overrightarrow{x} \in \mathbb{R}^2$ definiert und beinhalten in der ersten Achse, die Datenwerte, wie zum Beispiel x_1. Die zweite Achse ist die Merkmalsachse, welche die Merkmale darstellt die zur Verfügung stehen [Cho18] (K.2.2.9).

Bilddaten

Bei den Bilddaten handelt es sich generell um drei dimensionale Matrizen, jedoch unterscheiden wir die Graustufenbilder, für die zwei dimensionale Matrizen ausreichen. Hier bekommt jeweils eine Achse die Höhe, die zweite Achse die Breite und die letzte Achse die Farbtiefe des Bildes. Da die Graustufenbilder nur einen Farbkanal beinhalten, ist die zwei dimensionale Matrix ausreichend [Cho18] (K.2.2.11).

2.4.4 Lernverfahren

Bisher wurde der Durchlauf eines Neuronalen Netzes definiert, jedoch nicht wie es aus den Daten lernen kann. Damit das Neuronale Netz lernen kann, muss er auch Fehler erkennen können. Je nach Größe oder Art der Ziel-Klassen wird eine Aktivierungsfunktion ausgewählt. Ein Beispiel dazu ist die Mehrklassen Klassifizierung, bei der die Aktivierungsfunktion Softmax ausgewählt wird. Die Softmax-Funktion, die auch als Softmax-Regression bekannt ist, stellt anhand der Eingabewerte eine Wahrscheinlichkeitsverteilung zur Verfügung, deren Summe 1 ergibt. Somit werden den Ziel-Klassen Werte vergeben die zwischen 0 und 1 liegen [Mah18]. Durch die Vereinheitlichung der Neuronen mit der Aktivierungsfunktion, können Fehlerwerte kalkuliert werden. Dies geschieht durch die Annahme, dass die Ausgabeschicht den Wert $x_0 = 0.75$ beinhaltet, wobei der echte erwartete Wert $y = 1$ ist. Durch die Anwendung eines der Verlustfunktionen, stellt das Neuronale Netz fest, wie groß die Abweichung des Ergebnisses von dem tatsächlichen Ergebnis ist. Hierzu werden unterschiedliche Verfahren genutzt [Mat21](S.124-126). Eines dieser Verfahren ist die mittlere quadratische Abweichung (engl. mean squared error, MSE).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

, wobei n Anzahl der Datenpunkte, Y_n das erwartete Ergebnis und $\hat{Y}_i$ die prognostizierten Werte sind.

Mit den vorher angenommenen Werten wird der Fehler wie folgt berechnet.

$$MSE = \frac{1}{1}\sum_{i=1}^{1}(Y_i - \hat{Y}_i)^2$$
$$MSE = 1 * (1 - 0.75)^2 = 0.0625$$

Somit kennt das Neuronale Netz die Abweichung der Prognose, womit es den Lernprozess starten kann. Es stellt sich die Frage wie das Neuronale Netz sich optimieren bzw. verbessern lässt. [Pla21] (S.194). Es ist bekannt das ein Neuronales Netz mehrere Ausgaben liefern kann und für jeder dieser Ausgaben ist es möglich eine Verlustfunktion zuzuweisen. [Cho18].

Verlustfunktion

Die Verlustfunktion (engl. loss-, costfunktion) wird für die Optimierung während der Trainingsphase genutzt. Hierzu gibt es verschiedentliche Funktionen wie zum Beispiel die oben genannte mittlere quadratische Abweichung. Ein anderes Beispiel ist die Cross-Entropy-Loss Funktion, welche für die Optimierung von Klassifizierungs-Modellen geeignet ist. Die Cross-Entropy-Loss Funktion ist ähnlich wie die Softmax-Funktion aufgebaut. Sie misst den Abstand zwischen dem prognostizierten Wert und dem Wahrheitswert, um den Fehler zu erkennen und die Gewichte so anzupassen, dass das Modell bessere Ergebnisse erzielt. Während der Trainingsphase werden die Gewichte iterativ und entsprechend mit dem Ziel, den Cross-Entropy-Loss zu minimieren, angepasst [Mah18]. Im Rahmen dieser Arbeit wird die Cross-Entropy-Loss Funktion angewendet, da es sich um eine Mehrklassen Klassifizierung handelt.

Rückpropagierung

Die Anpassung der entsprechenden Gewichte wird durch die Rückpropagierung ermöglicht. Damit die Gewichte angepasst werden können, ist es notwendig, dass die Verlustfunktion des Modells ein Minimum annimmt. Es ist bekannt, dass mit dem Gradienten der Funktion das Minimum erreichbar ist, welches die partielle Ableitung aller Vektoren darstellt. Dieser Vektor wird auch $\delta(f)$ genannt. Er zeigt an jedem Punkt der Funktion, die Richtung des steilsten Anstiegs (bzw. Maximums). $-\delta(f)$ zeigt die Richtung des steilsten Abstiegs (bzw. Minimums) der Funktion [Vog21] (S.337) an. Allgemein wird das Gradientenverfahren folgendermaßen dargestellt:

$$x^{i+1} = x^i - \eta * \delta(f(x^i))$$

, wobei x^0 der Startpunkt ist und η die Schrittweite, welche auch als die Lernrate beschrieben wird. Dieses Verfahren lässt sich wie folgt bei den Gewichten integrieren.

$$w_{neu} = w_{alt} - \eta * \delta(E(W_{alt}))$$

, wobei w das Gewicht darstellt und η die Schrittweite.

Somit kann das Neuronale Netz seine Gewichte anpassen und eine Optimierung ermöglichen. Die partielle Ableitung der Fehlerfunktion E, wird durch die Verwendung der Kettenregel bestimmt.

$$\frac{\delta E(W)}{\delta w_{ij}} = \sum_{k=1}^{n} [\frac{\delta}{\delta w_{ij}} (\frac{1}{2}(Y_i - \hat{Y_i})^2)]$$

, wobei Y_i der prognostizierter Wert ist und $\hat{Y_i}$ der erwartete Zielwert. Dabei muss beachtet werden, dass die Y_i Werte von Anfang an bekannt sind. Somit berechnet das Modell den Fehler mit der Fehlerfunktion und wendet die partielle Ableitung auf diese an. Dadurch werden die Gewichte mit der oben genannten Formel angepasst. [Vog21] (S.337-338).

Ablauf der Fehlerrückführung

Damit dieser Ablauf genauer beschrieben wird, müssen die Schritte zur Optimierung des Modells definiert werden. Zunächst muss die Feed-Forward Propagierung stattfinden, damit die Gewichte W^l und Bias b^{lj} (siehe Abbild 2.7), wobei l den Index von der Schicht darstellt, initialisiert werden. Als nächstes wird der Ausgabefehler berechnet, welches die Verlustfunktion darstellt. Durch diese Funktion wird die Größe des Fehlers berechnet. Im darauffolgendem Schritt, findet die Fehlerrückführung statt, welche durch die Rückpropagierung ermöglicht wird. Bei der Fehlerrückführung wird für jede Schicht $l \in L-1, L-2, ..., 1$ folgende Berechnung durchgeführt.

$$\delta^l = (D\phi_l(x^l))^T * (w^{l+1})^T * \phi_{l+1}$$

Nachdem diese Schritte vollständig bearbeitet wurden, werden die Gewichte angepasst, wodurch das Modell optimiert wird [Pla21] (S.243).

2.5 Deep Learning

Bei Deep Learning handelt es sich um eine Teilmenge der Maschinellen Lernalgorithmen, mit der komplexere Aufgaben gelöst und automatisiert werden können. Der Unterschied zu einem Künstlichen Neuronalen Netz ist, dass Deep Learning mehrere verborgene Schichten beinhaltet (siehe Abbildung 2.8). Hierzu ist die Aktivierungsfunktion als die Sigmoid-Funktion vorgesehen, die wie folgt definiert ist:

$$sig(t) = \frac{1}{(1+e^{-}t)}$$

,wobei $sig(t) \in [0, 1]$ und $t \in \mathbb{R}$ gilt.

Dies bedeutet, dass die Neuronen immer eine Zahl übertragen bekommen, die zwischen 0 und 1 liegt. Somit können durch Implementierungen Schwellenwerte definiert und dementsprechend Neuronen aktiviert oder deaktiviert werden [Fro21](S.178).

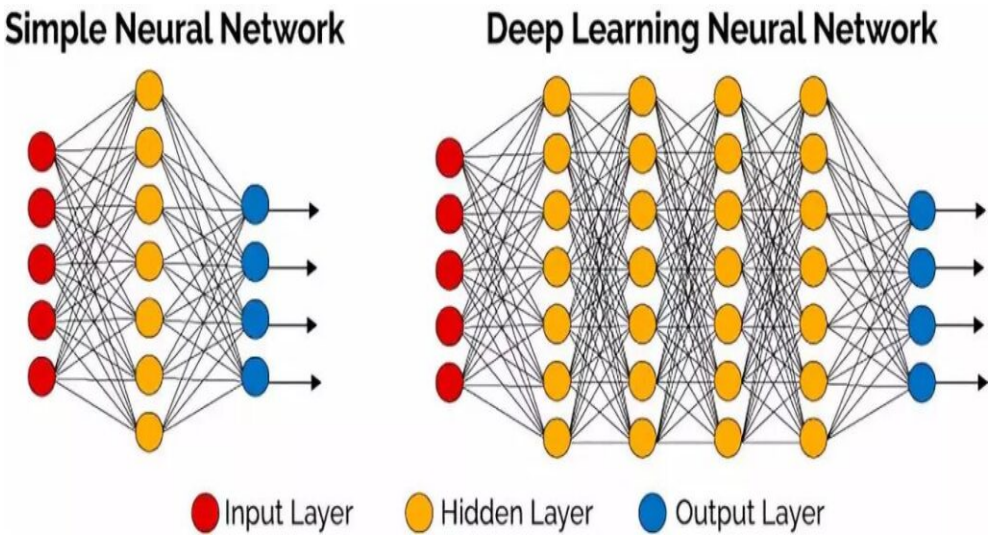


Abbildung 2.8: Künstliches Neuronales Netz vs. Deep Learning [Abbd]

2.5.1 Convolutional Neural Network

Bei einem CNN, im Deutschen "gefaltetes Neuronales Netzwerk", handelt es sich von einer Sonderform des künstlichen neuronalen Netzes, das speziell für Bild- oder Audiobearbeitung entwickelt wurde. Diese Form ist teils an den biologischen Ablauf angelehnt.[Lub19] Der Aufbau eines CNNs ist in Abbildung 2.9 dargestellt.

In Abbildung 2.9 sind mehrere Schichten (Layer) zu sehen. Dabei ist die Eingabeschicht (Inputlayer) und die Ausgabeschicht (Outputlayer) bekannt. Wie im vorherigen Abschnitt auch definiert, hat ein CNN mehrere verborgene Schichten,

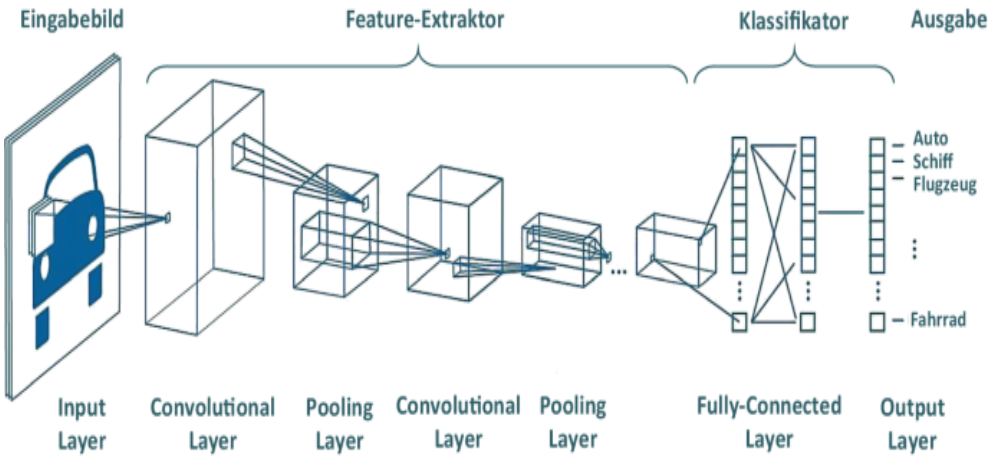


Abbildung 2.9: Aufbau eines CNNs [Abba]

darunter den **Convolutional Layer**, der einzelne Merkmale aus den Eingabedaten ortsunabhängig erkennt und daraufhin diese extrahiert. Diese Merkmale könnten bei einer Bildverarbeitung den Kanten, Linienformen, Farbtupfer und Faltung entsprechen, die eine Struktur darstellen. Ein Beispiel dazu wird im folgenden Abbild dargestellt.

Diese Merkmale werden in der **Pooling-Schicht**, auch als Subsampling bekannt, bearbeitet, indem die Auflösung der Merkmale reduziert und verdichtet wird. Die Pooling-Schicht enthält keine Modellparameter, weshalb das Modell durch diese Schicht nicht wirklich lernt. Somit werden überflüssige Informationen verworfen, wodurch die Datenmenge reduziert wird. Dadurch wird die Berechnungsgeschwindigkeit erhöht. Diese zwei Schichten können so oft wie möglich verwendet werden, da sie nur Merkmale aus den Bilder extrahieren und optimierend zu Verfügung stehen. Die eigentliche Klassifikation entsteht durch den **Fully-Connected Layer**, im Deutschen "vollständig verknüpfte Schicht", der sich der wiederholten Abfolge der Convolutional- und Pooling-Schichten anschließt, indem alle Merkmale der vorgelagerten Schichten mit jedem Ausgabemerkmal verknüpft sind. Somit werden mögliche Kombinationen aus den Strukturen entworfen, die komplexere Strukturen identifizieren lassen. Wichtig ist hierbei, dass die Anzahl an Neuronen in der vollständig vernetzten Schicht abhängig von den Klassen oder Objekten sind [Lub19].

2.6 Evalutionsmetriken

Für die Bewertung eines Klassifizierungsvorgangs werden Performance-Metriken benötigt. Diese Metriken lassen sich aus den relevanten Häufigkeiten der Fehler des Klassifikators welcher er während der Zuordnung der Objekte macht, herleiten. Metriken, die für bei einer Mehrklassen-Klassifizierung verwendet werden können, sind

die gleichen Metriken wie für Binäre-Klassifikationsprobleme. Die Metriken werden dabei für jede einzelne Klasse berechnet. Ein beliebtes Verfahren für die Bewertung ist dabei die Wahrheitsmatrix (Konfusionsmatrix). Hier werden die relevanten Häufigkeiten in einer Tabelle dargestellt. Aus dieser Tabelle können dann die Performance-Metriken abgeleitet bzw. berechnet werden. In der nächsten Abbildung ist eine Konfusionsmatrix zu sehen, die ein Beispiel zur Spam-Erkennung beinhaltet [Bur19] (K.5.6.1).

	Spam (vorhergesagt)	Kein_Spam (vorhergesagt)
Spam (tatsächlich)	23 (RP)	1 (FN)
Kein_Spam (tatsächlich)	12 (FP)	556 (RN)

Abbildung 2.10: Konfusionsmatrix [Bur19](K.5.6.1)

Die Achsen der Matrix geben die tatsächlichen und vorhergesagten Kennzeichnungen an. Bei einem binären Klassifikationsproblem gibt es zwei Klassen. In diesem Beispiel sind das „Spam“ und „Kein_Spam“. Aus der Matrix kann in diesem Fall abgelesen werden, dass von 24 Beispielen, die tatsächlich Spam waren, 23 vom Modell auch richtig als Spam klassifiziert wurden. Es wurden also 23 richtig positiv (RP) klassifiziert. Ein Beispiel wurde fälschlicherweise als „Kein_Spam“ klassifiziert, obwohl es sich dabei um Spam handelte. Dieses Beispiel wurde also falsch negativ (FN) klassifiziert. Außerdem wurden von 568 Beispiel, die eigentlich Spam waren, 556 richtig und 12 wurden falsch klassifiziert. Dies bedeutet, es sind 556 richtig negative (RN) und 12 falsch positive (FP) Klassifizierungen vorhanden. Die positive Klasse ist dabei „Spam“ und die negative Klasse „Kein_Spam“ [Bur19] (K.5.6.1).

Möchte man diese Art von Bewertung für ein Mehrklassen-Klassifikationsproblem (Multi-Class-Klassifikationsproblem) anwenden, muss die Anzahl der Zeilen und Spalten der Matrix, der Anzahl der verschiedenen vorhandenen Klassen entsprechen. Die Wahrheitsmatrix wird zum Beispiel für die Berechnung von Präzision (Precision), Trefferquote (Recall) oder Korrektklassifikationsrate (Accuracy) verwendet. Es gibt noch weitere Metriken, die aus der Matrix abgelesen werden können. Jedoch werden nur die drei genannten Metriken im Rahmen dieser Arbeit und damit für die Bewertung der Experimente verwendet [Bur19] (K.5.6.1).

2.6.1 Präzision (Precision) und Trefferquote (Recall)

Die am häufigsten verwendeten Metriken sind Precision und Recall. Diese Größen werden auch für die Evaluierung der Modelle verwendet. Mit diesen Metriken kann eine Leistungsbeurteilung eines Modells erfolgen.

Die Präzision eines Modells gibt das Verhältnis der richtigen positiven Vorhersagen zu den positiven Vorhersagen insgesamt an:

$$\text{Präzision} = \frac{RP}{RP+FP}$$

Die Trefferquote eines Modells gibt das Verhältnis der richtigen positiven Vorhersagen zu der Gesamtheit der tatsächlich positiven Beobachtungen an.

$$\text{Trefferquote} = \frac{RP}{RP+FN}$$

Diese Metriken finden vor allem bei binären Klassifikationsproblemen Anwendung, können jedoch auch zur Bewertung von Multi-Class Klassifikationsproblemen verwendet werden. Man wählt dafür zunächst eine Klasse aus, für die man diese Metriken ermitteln möchte und ordnet alle zu dieser Klasse gehörenden Beobachtungen den Positiven zu. Alle anderen Beispiele der restlichen Klassen werden dann den Negativen zugeordnet. Somit kann diese Metrik für jede Klasse einzeln berechnet werden und am Ende zum Beispiel durch die Betrachtung des Mittelwerts auch im Gesamten bewertet werden [Bur19] (K.5.6.2).

2.6.2 Korrektklassifizierungsrate (Accuracy)

Eine weitere Metrik, die aus der Wahrheitsmatrix berechnet werden kann, ist die Korrektklassifizierungsrate, auch bekannt als Accuracy. Diese Metrik gibt die Anzahl der richtig klassifizieren Beispiele im Verhältnis zur Gesamtzahl aller klassifizierten Beispiele bzw. vorhandener Daten an. Die Metrik ist gegeben durch:

$$\text{Korrektklassifikationsrate} = \frac{RP+RN}{RP+RN+FP+FN}$$

Diese Metrik ist vor allem dann eine sinnvolle Größe, wenn die vorhandenen Klassen im Datensatz gleich wichtig sind. Bei der Auswahl von Metriken für einen Klassifikationsvorgang ist vor allem das Ziel bzw. die Wichtigkeit der einzelnen Metriken zu beachten. In dem Beispiel der Spam Erkennung wird man falsch negative Klassifizierungen eher in Kauf nehmen als falsch positive. Den eine falsch positive Erkennung von Spam-Mails würde bedeuten, dass das Modell die Mail fälschlicherweise als Spam klassifiziert und die E-Mail dadurch im Posteingang nicht erscheint und untergeht. Bei einer falsch negativen Klassifizierung würde die E-Mail fälschlicherweise als kein Spam klassifiziert werden und sie wäre im Posteingang sichtbar [Bur19] (K.5.6.3).

3 CRISP-DM Modell

Das CRISP-DM Modell "Cross-Industry Standard Process for Data Mining"beschreibt den Standard für die Strukturierung der Maschinellen Lernalgorithmen. Das Ziel ist es, einen einheitlichen Prozess für Data Mining Projekte zu schaffen und ist eine Anleitung für Data Mining, die in 6 Schritten aufgeteilt ist [Wut19]. Der Ablauf des Modells ist in Abbildung 3.1 dargestellt.

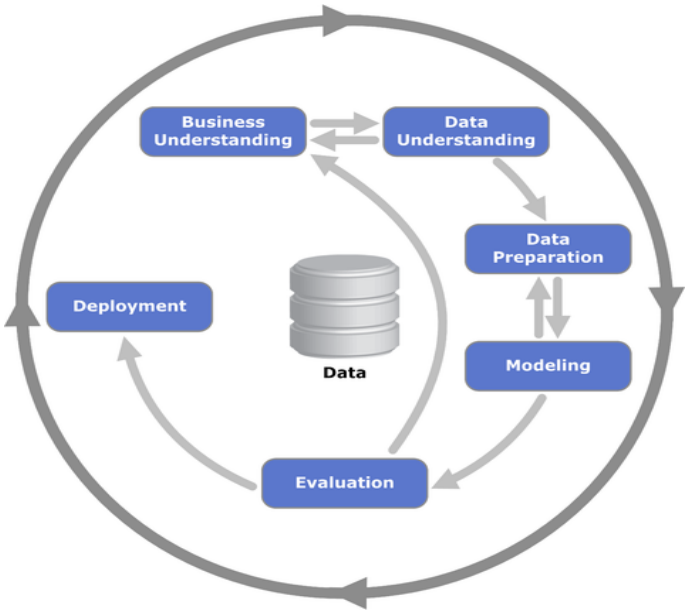


Abbildung 3.1: Ablauf CRIPS-DM Modell [Abbb]

Business Understanding

Der erste Schritt der 6 Schritte des CRISP-DM Modells, ist das Business Understanding, welches sich mit den betriebswirtschaftlichen Problemen befasst. Zunächst werden betriebswirtschaftliche Anforderungen extrahiert und analysiert, wodurch die Grundlagen für weitere Schritte und Entscheidungen gebildet werden. Damit ein Verständnis über die betriebswirtschaftlichen Fragestellungen übermittelt wird, ist es wichtig die Anwender in die Data Mining Prozesse mit einzubeziehen.

Das Business Understanding beinhaltet 4 Aspekte, die in Betracht gezogen werden sollen. Hierzu gehört wie zuvor auch erwähnt, die Bestimmung der betriebs-

wirtschaftlichen Problemstellungen und dessen Zielkriterien. Ein weiterer Aspekt, ist die Situationsbewertung, welches die zur Verfügung gestellten Ressourcen für das Data Mining Projekt beschreibt. Dies bedeutet zum Beispiel die Software- und Personalressourcen. Zusätzlich werden Risiken, die, während dem Data Mining Projekt entstehen können, identifiziert. Auch die Bestimmung analytischer Ziele ist ein Aspekt des Business Understanding, indem eine Analyse der zuvor bestimmten betriebswirtschaftlichen Probleme durchgeführt wird. Auch Erfolgskriterien für das Data Mining Projekt werden bestimmt, wie zum Beispiel eine Optimierung des Geschäftsprozesses um 5% mit weniger Bedarf an Ressourcen. Der letzte Aspekt beinhaltet die Erstellung des Projektplans, welches die beabsichtigten Ziele des Projektes beschreibt, indem die Auflistung der einzelnen Schritte mit jeweiliger Zeitspanne definiert werden. Hinzu kommen die möglichen Risken oder Ursachen für das Scheitern des Projektes, die aufgefasst werden [Bur19] (K.5.6.3).

Data Understanding

In diesem Schritt werden relevante Datenbestände, die zu einer Hypothese führen können, aufgestellt. Zum Beispiel werden bei einer Wettervorhersage die aktuellen Zustände der Atmosphäre betrachtet. In diesem Fall ist der Datenbestand die Zustände der Atmosphäre. Zunächst werden Daten gesammelt, die benötigt werden und zu einer bereits bestehenden Datenmenge integriert. Als nächstes werden diese Datenbestände beschrieben. Hierzu werden Eigenschaften wie Quantität, Formateigenschaften, Anzahl der Einträge und Felder der Daten erfasst. Der letzte Schritt des Data Understanding ist die Untersuchung und die Bewertung der Daten, in dem eine Analyse mit den Daten betrieben wird, um erste Hypothesen und Erkenntnisse zu treffen und diese dann zu dokumentieren. Bei der Bewertung der Daten wird sichergestellt, dass die Daten verwendbar sind und sie keine fehlende Attributwerte enthalten [Wut19].

Data Preparation

Um die finale Menge der Datenauswahl zu erstellen, wird der Schritt Data Preparation durchgeführt. An erster Stelle werden Daten ausgewählt, die für das Ziel des Projektes relevante Informationen hergeben. Wichtig hierbei ist zu betrachten, dass die Daten in dem richtigen Datenformat gespeichert werden [Wut19].

Modeling

Um eine vorliegende Problemstellung aus dem Bereich Maschinelles Lernen lösen zu können, stehen verschiedene Modelle beziehungsweise Algorithmen zur Verfügung.

Aus diesen Modellen wird dann das Modell ausgesucht, welches im Bezug zum definierten Ziel das optimale Ergebnis liefert. Manche Modelle benötigen hierzu vorbestimmte Datenstrukturen, die dazu führen können, einen Schritt zurück zu gehen und sich noch einmal mit der Datenvorbereitung zu befassen. Anschließend wird das Modell trainiert und getestet um die Qualität und Genauigkeit des Modells zu überprüfen [Wut19].

Evaluation

Vor der Finalisierung des Data Mining Projektes, ist es noch notwendig zu prüfen, ob die Qualität des Modelles für das Ziel des Projektes ausreicht. Ist das Ergebnis noch nicht gut genug oder werden vorher definierte Ziele nicht erreicht, müssen die Gründe für das Scheitern analysiert und entsprechende Änderungen im Modell vorgenommen werden. Dies hat zufolge, dass die vorherigen Schritte nochmals durchlaufen werden müssen [Wut19].

Deployment

Als letztes werden die aus dem erstellten Modell gewonnen Erkenntnisse für die Nutzer vorbereitet und ihm als Anwendung zur Verfügung gestellt. Damit kann ein Anwender zum Beispiel eigene Daten in das Modell hochladen und erhält, je nach der vorliegenden Problematik, ein Ergebnis für seine verwendeten Daten [Wut19].

4 Experimente

4.1 Business Understanding

Online-Versandhändler haben oft eine hohe Rückläuferquote. Diese Rückläuferquote kommt zum Beispiel durch den Kauf von Artikeln in verschiedenen Größen zustande. Die nicht passenden Größen werden dann vom Käufer wieder zurückgesendet. Unter diese Versandhändler fällt, mit einer Rückläuferquote von bis zu 50%, auch das Unternehmen Zalando. Dabei müssen 97% der zurückgesendeten Produkte wieder eingelagert und verkauft werden [C.18]. Für den erneuten Verkauf werden die Produkte, in diesem Betrachtungsfall Klamotten, identifiziert, gelabelt und bei Bedarf wieder eingelagert. Das Unternehmen Zalando verschickte im Jahr 2019 rund 186 Millionen Bestellungen [Raber]. Wenn man davon ausgeht, dass eine Bestellung 4 Artikel beinhaltet und 50% der Bestellungen wieder zurückgesendet werden, ergibt das 372 Millionen Artikel, die wieder neu gelabelt und eingelagert werden müssen.

Die Unterstützung und Optimierung der Rückläuferbearbeitung, kann zum Beispiel durch eine AI gestützte Klassifikation gewährleistet werden. Mit diesem Ansatz kann dann die zu einem Artikel zugehörige Kategorie, auf Basis von Bildern, bestimmt werden. Damit ergibt sich die Frage- bzw. Problemstellung, inwiefern können Machine Learning Algorithmen bei der Klassifizierung von Bildern einzelner Kleidungsstücke unterstützen?

4.2 Data Understanding

Der für diese Arbeit gewählte Datensatz heißt Fashion MNIST und wurde im Jahr 2017 von dem Unternehmen Zalando veröffentlicht. Der Datensatz steht in einer GitHub Repository zur Verfügung und ist frei zugänglich [SE17]. Er beinhaltet Zalando-Artikelbilder, die in zwei Datensätze aufgeteilt sind. Der erste Satz enthält 60.000 Beispiele, die für das Training verwendet werden und der zweite Satz 10.000 Beispiele, die für das Testen verwendet werden. Jedes Beispiel, welches jeweils ein Bild darstellt, ist ein 28x28 Graustufenbild. Jedem Bild wird dabei noch ein Label zugeordnet. Ein Label repräsentiert eins der 10 Klassen, wobei jede Klasse eine

Artikelkategorie darstellt. In der folgenden Abbildung 4.1 ist eine Übersicht des Datensatzes bzw. der Bilder zu sehen. Dabei werden pro Klasse drei Zeilen beansprucht.

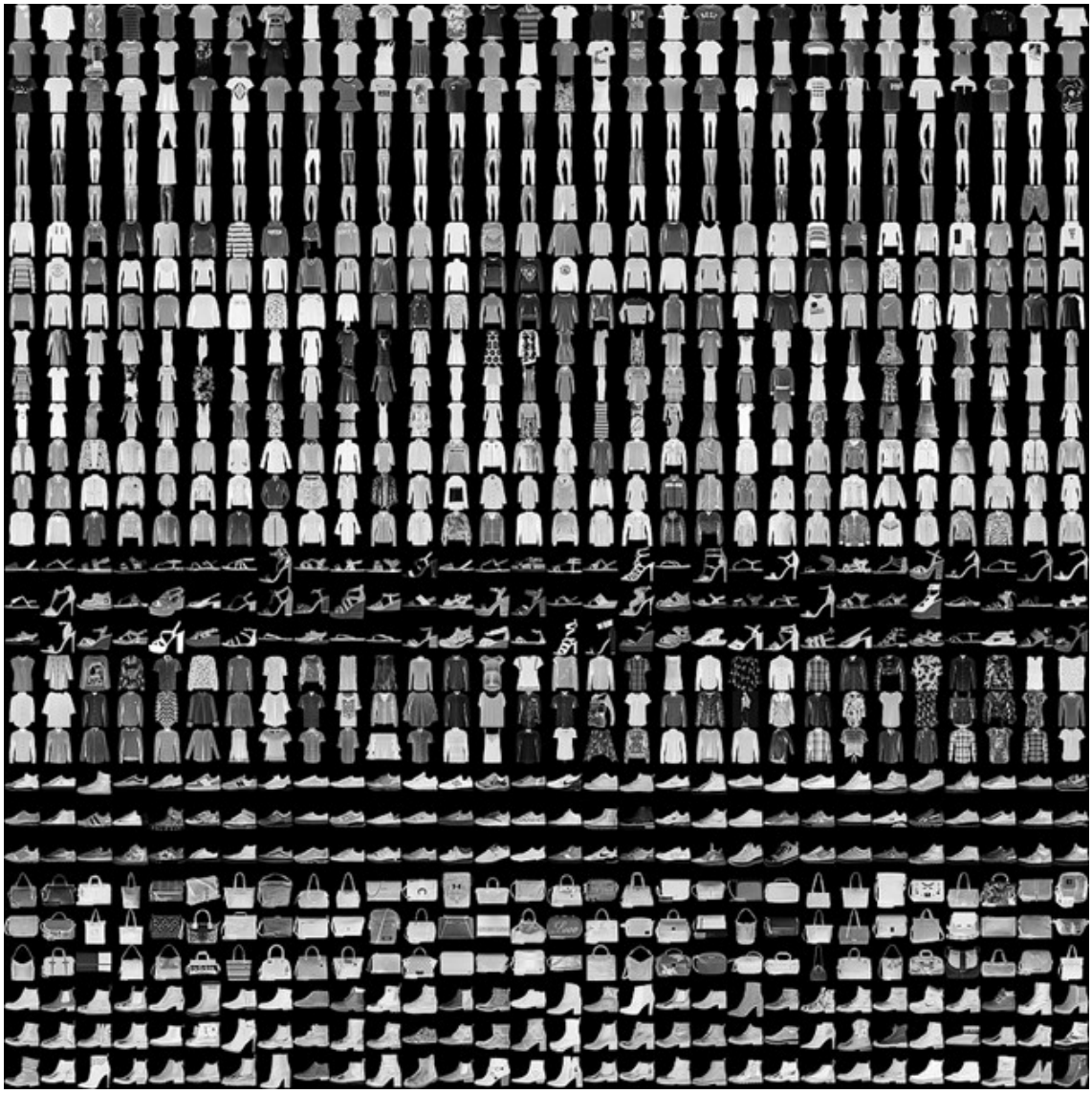


Abbildung 4.1: Der gesamte Datensatz: FASHION-MNIST [Abbc]

Der ursprüngliche MNIST Datensatz, der durch Fashion-MNIST ersetzt wurde, ist sehr beliebt und wird oft verwendet, um Algorithmen zu validieren. Es heißt auch „If it doesn't work on MNIST, it won't work at all". Jedoch wurde der MNIST Datensatz aufgrund seiner Einfachheit ausgetauscht [SE17].

Auch die Labels bestehen aus 28x28 Pixeln, wobei jeder einzelne Pixel einen Wert zwischen 0 und 255 annehmen kann. Der aktuelle Datensatz Fashion-MNIST enthält folgende 10 Klassen bzw. Labels:

Label	Artikel
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Tabelle 4.1: Labels des Datensatzes

Der Trainings- als auch der Testdatensatz ist ziemlich ausgeglichen. In jedem Satz ist jede Kategorie mit ca. 10% vertreten, was in den nächsten Abbildungen zu sehen ist.

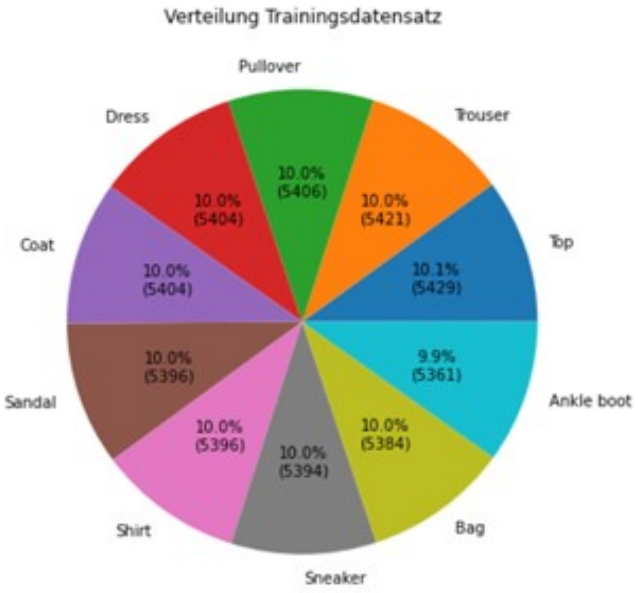


Abbildung 4.2: Verteilung Trainingsdatensatz [Abbh]

Abbildung 4.3: Verteilung Testdatensatz [Abbg]

Aufgrund der Ausgeglichenheit der Daten müssen im Rahmen der Data Preparation keine großen Änderungen oder Vorbereitungen vorgenommen werden.

4.3 Data Preparation

Um die vorliegenden Daten für die modellierten ML-Verfahren verwenden zu können, müssen diese entsprechend vorbereitet werden.

4.3.1 Test- und Trainingsdaten

Für die Vorbereitung werden die Trainingsdaten nochmal aufgeteilt. Dabei werden aus den vorhandenen 60000 Trainingssätzen, 6000 für die Validierung extrahiert. Dadurch bleiben für die Trainingsphase 54000 Datensätze übrig. Die Testdatensätze bleiben mit der ursprünglichen Anzahl von 10000 gleich.

4.3.2 Skalierung der Merkmale

Die Skalierung von Merkmalen dient zur Normalisierung des Bereichs der erklärenden bzw. unabhängigen Variablen. Das Verfahren ist auch als Datennormalisierung bekannt und wurde im Rahmen der Datenvorbereitung auf den Daten angewendet. Die Eingabedaten liegen als Graustufenbilder vor und besitzen Werte zwischen 0 und 255. Um die Normalisierung dieser Daten vorzunehmen, werden die Werte durch 255 geteilt um somit Ausprägungen zwischen 0 und 1 zu erreichen.

4.3.3 Datentransformation

In dem Ausgangsdatensatz stellt jede Zeile mit seinen 784 Spalten, die wiederum als 28x28 Pixel zu verstehen sind, jeweils ein Bild dar. Für die spätere Verwendung der Daten, müssen diese aber umgeformt werden. Die Umformung der Graustufenbilder muss so erfolgen, dass ein Bild durch 28 Zeilen und 28 Spalten dargestellt wird und so die Pixel wiedergibt. Dadurch kann das Neuronale Netz eine genaue Position der Pixel identifizieren. Die dafür verwendete Methode kann dem beigefügten Notebook entnommen werden. Im Rahmen der Datentransformation werden außerdem die Zielvariablen in die vorhandenen 10 Kategorien unterteilt.

4.4 Modellierung

In diesem Schritt wird ein Maschinelles Lernalgorithmus erstellt und trainiert. Die Optimierung dieses Modells findet in einer bestimmte Reihenfolge statt. Zunächst wird ein Modell erstellt, welches Trainiert und mit den Validierungsdaten bewertet wird. Als nächstes wird das Modell im Schritt 4.5 Evaluation evaluiert. Durch das Testen wird es möglich die neu implementierten Modell mit dem alten Modell zu vergleichen. Ist man mit den Ergebnissen nicht zufrieden oder das Zielergebniss wurde nicht erreicht, so springt man zur Phase 4.4 Modelliereung zurück und versucht das Modell zu optimieren. Sollten die Zielergebnisse erreicht werden, so geht es mit dem schritt 4.6 Deployment weiter.

Insgesamt werden drei verschiedene implementierte Modelle vorgestellt und im nächsten Abschnitt evaluiert. Jedes Modell bzw. Experiment beinhaltet dabei ein Convolutional Neuronal Network. Die Unterschiede der drei Modelle liegen dabei in den verwendeten Schichten und deren Zusammenspiel.

4.4.1 Experiment 1

Das erste implementierte CNN-Modell beinhaltet folgende drei Layer, die in vorherigen Abschnitten bereits erläutert wurden:

- Convolutional Layer (Conv2D)
- Pooling Layer (MaxPooling2D)
- Flatten Layer (Flatten)
- Fully-Connected Layer (Dense)

Der Conv2D und MaxPooling2D Layer gehören, wie in Kapitel 2 beschrieben, zu dem Feature-Extraktor. Die beiden Schichten werden zwei mal nacheinander in das Modell integriert. Im Anschluss werden dem Modell zwei Dense Layer hinzugefügt. Der erste Conv2D Layer bekommt die Eingabedaten im 28x28 Pixel Format, daher erhält diese Schicht einen Input-Parameter, der die Dimension der Eingabedaten beschreibt. Im MaxPooling2D werden die aus den Conv2D extrahierten Merkmale in eine 14x14 dimensionale Matrix zusammengefasst. Der zweite Conv2D Layer erhält nach dem MaxPooling2D Layer Daten in der Form 14x14 Pixeln, aus dem der Layer dann wiederum Merkmale extrahiert. Der anschließende MaxPooling2D Schicht fasst diese Merkmale dann nochmals zusammen. Als nächstes formt der Flatten Layer die Merkmale in eine eindimensionale Matrix um. Zuletzt folgen dann die beiden Dense Layer. Die erste Dense Schicht ist der Fully-Connected Layer und die zweite eingesetzte Dense Schicht der Output Layer (Ausgabeschicht). Der ersten Dense Schicht wird dabei die Aktivierungsfunktion Recitifier übergeben. Die Recitifer (relu) wird wie folgt definiert.

$$f(x) = max(0, x), \text{ wobei } x \text{ der Eingangswert ist.}$$

Die Funktion gibt den maximalen Wert zurück, der entweder x oder Null ist. Für die Ausgabeschicht wird die Aktivierungsfunktion Softmax eingestellt. Damit wird das erste Modell finalisiert und kann trainiert werden.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        320

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 14, 14, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 32)         0
 2D)

 flatten (Flatten)           (None, 1568)              0

 dense (Dense)               (None, 28)                43932

 dense_1 (Dense)             (None, 10)                290

=================================================================
Total params: 53,790
Trainable params: 53,790
Non-trainable params: 0
_________________________________________________________________
```

Abbildung 4.4: Modellaufbau [Koc21] (Experiment 1)

Insgesamt kann das Modell 53,790 Parameter trainieren. Bevor das Training gestartet werden kann, müssen noch einige Variablen definiert werden. Zunächst braucht das Modell ein optimizer der ein tensorflow Objekt darstellt. Dieses Objekt beschreibt den Optimierungsvorgang für das Modell. In diesem Fall wird der optimizer als "tf.keras.optimizers.Adam()"definiert, der das stochastische Gradientenabstiegsverfahren beschreibt. Der nächste Parameter definiert die Verlustfunktion des Modells, welche wir als Cross-Entropy-Loss Funktion angeben. Als Bewertungsmetrik für das Modell, wird die Genauigkeit (engl. accuracy),die Präzision (engl. precision) und die Trefferquote (engl. recall) ausgewählt. Diese Metriken geben prozentual an, wie gut ein Modell prognostiziert. Zuletzt werden Epochen-Anzahl und Batch-Größen angegeben.

Trainingsphase

Mit der Übergabe der Trainings- und Validierungsdaten wird die Trainingsphase gestartet. Nachdem die Trainingsphase beendet ist werden die Bewertungsmetriken genutzt, um eine Darstellung der Leistung des Modells zu erhalten.

```
Epoch 1/10
844/844 [==============================] - 17s 20ms/step - loss: 0.5194 - accuracy: 0.8144 - val_loss: 0.3874 - val_accuracy: 0.8656
Epoch 2/10
844/844 [==============================] - 17s 20ms/step - loss: 0.3344 - accuracy: 0.8802 - val_loss: 0.3154 - val_accuracy: 0.8813
Epoch 3/10
844/844 [==============================] - 17s 20ms/step - loss: 0.2968 - accuracy: 0.8934 - val_loss: 0.3070 - val_accuracy: 0.8797
Epoch 4/10
844/844 [==============================] - 17s 20ms/step - loss: 0.2709 - accuracy: 0.9025 - val_loss: 0.3085 - val_accuracy: 0.8906
Epoch 5/10
844/844 [==============================] - 17s 20ms/step - loss: 0.2520 - accuracy: 0.9087 - val_loss: 0.2713 - val_accuracy: 0.9031
Epoch 6/10
844/844 [==============================] - 17s 21ms/step - loss: 0.2310 - accuracy: 0.9156 - val_loss: 0.2803 - val_accuracy: 0.9031
Epoch 7/10
844/844 [==============================] - 17s 20ms/step - loss: 0.2159 - accuracy: 0.9211 - val_loss: 0.3079 - val_accuracy: 0.8828
Epoch 8/10
844/844 [==============================] - 17s 20ms/step - loss: 0.2009 - accuracy: 0.9269 - val_loss: 0.2420 - val_accuracy: 0.9141
Epoch 9/10
844/844 [==============================] - 17s 20ms/step - loss: 0.1882 - accuracy: 0.9310 - val_loss: 0.2347 - val_accuracy: 0.9187
Epoch 10/10
844/844 [==============================] - 17s 20ms/step - loss: 0.1766 - accuracy: 0.9343 - val_loss: 0.2497 - val_accuracy: 0.9141
```

Abbildung 4.5: Bewertung Trainingsphase [Koc21](Experiment 1)

Die Ergebnisse zeigen, dass die erste Epoche eine Genauigkeit von $86,56\%$ erreicht und diese sich bei den nächsten Epochen erhöht. Damit hat das Modell am Ende der Trainingsphase eine Genauigkeit von 91%. Sichtbar ist die Differenz zwischen *accuracy* $(93,34\%)$ und *val_accuracy* $(91,14\%)$. Die accuracy (auch Genauigkeit

genannt) beschreibt in diesem Fall, die Genauigkeit der Trainingsdaten, diese wird jedoch nicht für die Bewertung genutzt. Die *val_accuracy*, welche die Genauigkeit der Validierungsdaten anzeigt, wird hier stattdessen in Betracht gezogen. Die Daten, die für die Validierung ausgewählt wurden, werden komplett abgekapselt von den Trainingsdaten übergeben. Der nächste Schritt für die Optimierung des Modells, sollte die Anpassung sein, wodurch der Abstand der Genauigkeiten verringert wird. Es ist auch ersichtlich, dass die loss-Werte sich gegen null nähern. Dies zeigt, dass das Modell sich verbessert bzw. optimiert. Die Leistung kann auch anhand eines Graphen dargestellt werden, wobei die Ergebnisse in Epochen aufgeteilt und geplottet werden. Dieser Graph ist in der nächsten Abbildung zu sehen.

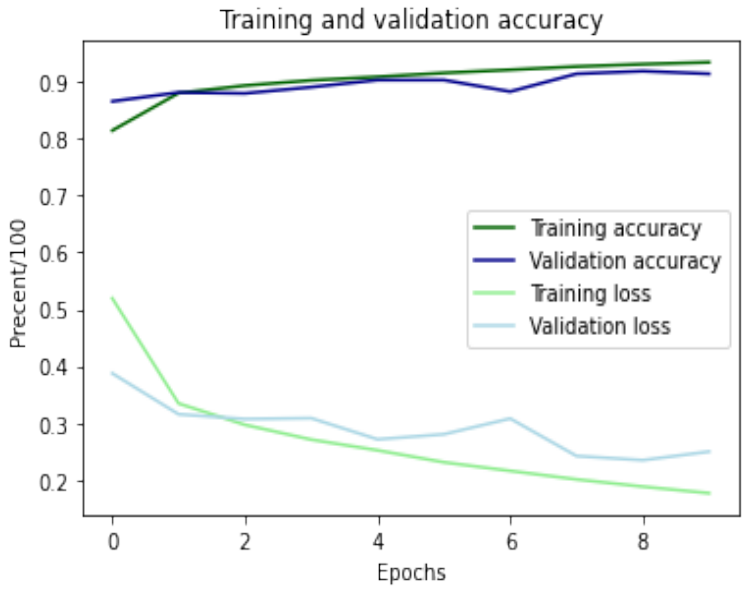


Abbildung 4.6: Bewertung der Trainingsphase anhand eines Graphen [Koc21](Experiment 1)

In diesem Abbild wird die Verbesserung des Modells durchaus ersichtlicher. Es ist deutlich, dass sich die loss-Werte gegen null nähern, währenddessen die Genauigkeit sich erhöht. Somit wird das Modell immer weiter optimiert bis zur letzten Epoche. Deutlich zu sehen sind die Sprünge der Linien bei den Validierungsmetriken. Eine Erklärung diesbezüglich ist die Überanpassung der Trainingsdaten im Modell, welches im nächsten Modellierungsschritt behandelt werden muss. Zunächst Evaluieren wir das aktuelle Modell im Kapitel 4.5.1, damit ein Vergleich mit dem neuen Modell möglich ist.

4.4.2 Experiment 2

In diesem Experiment wird das Modell aus dem ersten Experiment übernommen und weiter verbessert. Das Ziel ist zunächst die Reduzierung der Differenz von *val_accuracy* und *accuracy*, damit das Modell genauere Ergebnisse liefern kann. Dafür wird die Dropout Schicht verwendet, welche die Daten eliminiert, die zu einer Überanpassung des Modells führen. In diesem Layer werden Datensatze die mit einer Häufigkeit von rate auftreten, auf 0 gesetzt. Dabei ist rate der Parameter von dem Dropout Layer [ten22].

- Convolutional Layer (Conv2D)
- Pooling Layer (MaxPooling2D)
- Dropout Layer (Dropout)
- Flatten Layer (Flatten)
- Fully-Connected Layer (Dense)

Trainingsphase

```
Epoch 1/10
844/844 [==============================] - 18s 21ms/step - loss: 0.5884 - accuracy: 0.7867 - val_loss: 0.3718 - val_accuracy: 0.8656
Epoch 2/10
844/844 [==============================] - 17s 20ms/step - loss: 0.3927 - accuracy: 0.8580 - val_loss: 0.3423 - val_accuracy: 0.8797
Epoch 3/10
844/844 [==============================] - 17s 21ms/step - loss: 0.3504 - accuracy: 0.8738 - val_loss: 0.3059 - val_accuracy: 0.8875
Epoch 4/10
844/844 [==============================] - 17s 20ms/step - loss: 0.3209 - accuracy: 0.8843 - val_loss: 0.2815 - val_accuracy: 0.8859
Epoch 5/10
844/844 [==============================] - 17s 20ms/step - loss: 0.3049 - accuracy: 0.8896 - val_loss: 0.2831 - val_accuracy: 0.8906
Epoch 6/10
844/844 [==============================] - 18s 21ms/step - loss: 0.2875 - accuracy: 0.8945 - val_loss: 0.2579 - val_accuracy: 0.9094
Epoch 7/10
844/844 [==============================] - 17s 21ms/step - loss: 0.2774 - accuracy: 0.8993 - val_loss: 0.2722 - val_accuracy: 0.9000
Epoch 8/10
844/844 [==============================] - 17s 20ms/step - loss: 0.2634 - accuracy: 0.9033 - val_loss: 0.2405 - val_accuracy: 0.9172
Epoch 9/10
844/844 [==============================] - 17s 20ms/step - loss: 0.2557 - accuracy: 0.9058 - val_loss: 0.2291 - val_accuracy: 0.9094
Epoch 10/10
844/844 [==============================] - 17s 20ms/step - loss: 0.2438 - accuracy: 0.9102 - val_loss: 0.2324 - val_accuracy: 0.9141
```

Abbildung 4.7: Bewertung Trainingsphase [Koc21](Experiment 2)

Es wird deutlich, dass die Genauigkeit der Trainingsdaten, mit der Genauigkeit der Validierungsdaten fast übereinstimmt. Durch das Hinzufügen der Dropout Schicht wird also eine Überanpassung an die Trainingsdaten verhindert, womit das Modell auf unbekannte Daten besser vorbereitet ist. Die Differenz der Genauigkeit *accuracy* und *val_accuracy* wurde von 2% auf 0.38% reduziert.

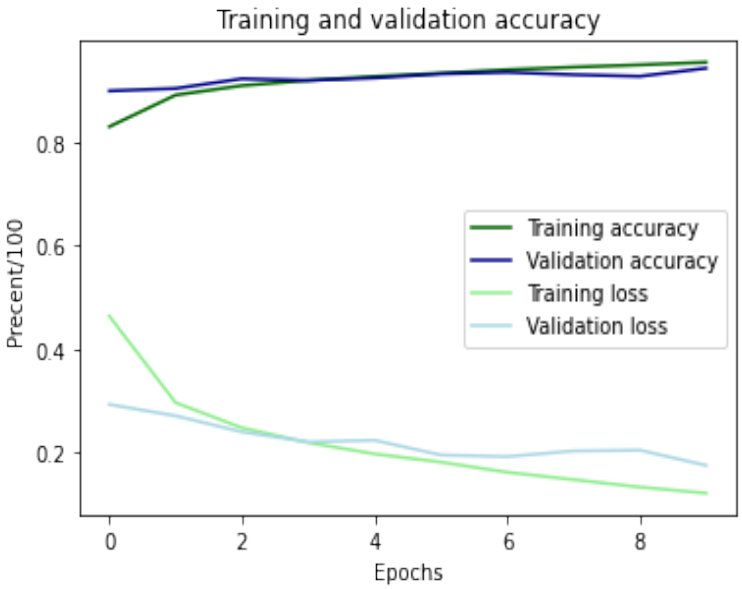
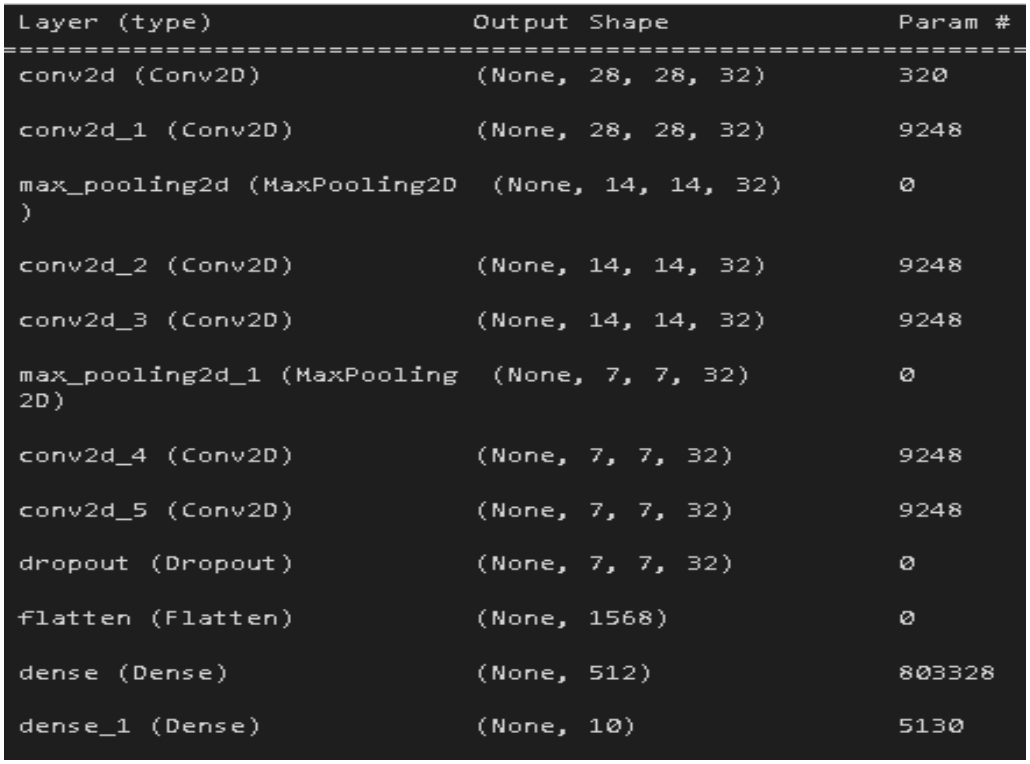


Abbildung 4.8: Bewertung Trainingsphase [Koc21](Experiment 2)

Im diesem Schritt wurde das Modell so optimiert, dass die Überanpassung verhindert werden kann. Durch den Graphen im Abbild 4.8 stellen wir fest, dass die Genauigkeiten der Validierungs- und Trainingsdaten konvergieren. Dir großen Sprünge im Graphen wurden genauso beseitigt. Dazu ist die Differenz der Genauigkeiten von 2% auf 0.38% gesunken. Als nächstes wird das Modell mit dem Testdaten getestet, welches im 4.5.2 Evaluation des zweiten Modells erarbeitet wurde.

4.4.3 Experiment 3

In diesem Modellierungsschritt wird der Fokus auf die Leistung der Prognosen gelegt. Es wurde bisher zwei Modelle erstellt, die bezüglich der Leistung fast identisch sind. In diesem Schritt ist das Ziel die Leistung des Modells zu erhöhen. Dabei werden mehrere Schichten hinzugefügt oder geändert. Zunächst werden alle Conv2D Schichten verdoppelt, um mehrere Merkmale extrahieren zu können. Damit die verdoppelten Conv2D Schichten nicht Merkmale verdoppelt an die nächste Schritt übergeben wird der Dropout Parameter (rate) reduziert. Da unsere Conv2D Schichte verdoppelt wurden, muss zunächst die Fully-Connected-Layer vergrößert werden. Als Parameter wird in dieser Schicht die Neuronen-Anzahl drastisch erhöht. Im folgenden Abbild 4.9 ist die Darstellung des geänderten Modell.



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	320
conv2d_1 (Conv2D)	(None, 28, 28, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_2 (Conv2D)	(None, 14, 14, 32)	9248
conv2d_3 (Conv2D)	(None, 14, 14, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 32)	0
conv2d_4 (Conv2D)	(None, 7, 7, 32)	9248
conv2d_5 (Conv2D)	(None, 7, 7, 32)	9248
dropout (Dropout)	(None, 7, 7, 32)	0
flatten (Flatten)	(None, 1568)	0
dense (Dense)	(None, 512)	803328
dense_1 (Dense)	(None, 10)	5130

Abbildung 4.9: Bewertung Trainingsphase [Koc21](Experiment 3)

Trainingsphase

4.5 Evaluation

Die definierten Modelle, werden nun im Evaluationsschritt getestet. Für das Testen wird hier der Testdaten eingesetzt.

4.5.1 Evaluation Experiment 1

Das Modell zur Experiment 1 wurde bisher trainiert und sollte als nächstes getestet werden. Hierzu übergeben wir die Testdaten dem Modell und beginnen mit dem Testen. Uns ist aus den Trainingsphase bekannt das, dass Modell eine Genauigkeit von 91% hat und während der Testphase dies liefern sollte. Bei der Evaluation wird für jede Klasse die Präzision und Trefferquote berechnet. Somit kann für jede Klasse eine Bewertung stattfinden.

Testphase

Aus den Daten im Abbild 4.11 kann die durchschnittliche Genauigkeit und Verlust berechnet werden. Das aktuelle Modell bring eine Leistung von 91.96% durchschnittliche Genauigkeit und 23.17% durchschnittliche Verlust über alle Klassen.

```
Epoch 1/10
844/844 [==============================] - 57s 67ms/step - loss: 0.4642 - accuracy: 0.8293
val_loss: 0.2937 - val_accuracy: 0.8984
Epoch 2/10
844/844 [==============================] - 56s 67ms/step - loss: 0.2972 - accuracy: 0.8904
val_loss: 0.2718 - val_accuracy: 0.9031
Epoch 3/10
844/844 [==============================] - 57s 68ms/step - loss: 0.2486 - accuracy: 0.9086
val_loss: 0.2407 - val_accuracy: 0.9219
Epoch 4/10
844/844 [==============================] - 58s 69ms/step - loss: 0.2201 - accuracy: 0.9187
val_loss: 0.2214 - val_accuracy: 0.9187
Epoch 5/10
844/844 [==============================] - 57s 67ms/step - loss: 0.1983 - accuracy: 0.9263
val_loss: 0.2246 - val_accuracy: 0.9234
Epoch 6/10
844/844 [==============================] - 57s 67ms/step - loss: 0.1824 - accuracy: 0.9326
val_loss: 0.1961 - val_accuracy: 0.9312
Epoch 7/10
844/844 [==============================] - 57s 68ms/step - loss: 0.1629 - accuracy: 0.9393
val_loss: 0.1931 - val_accuracy: 0.9344
Epoch 8/10
844/844 [==============================] - 57s 68ms/step - loss: 0.1487 - accuracy: 0.9443
val_loss: 0.2039 - val_accuracy: 0.9297
Epoch 9/10
844/844 [==============================] - 57s 67ms/step - loss: 0.1343 - accuracy: 0.9487
val_loss: 0.2055 - val_accuracy: 0.9266
Epoch 10/10
844/844 [==============================] - 57s 68ms/step - loss: 0.1225 - accuracy: 0.9536
val_loss: 0.1761 - val_accuracy: 0.9422
```

Abbildung 4.10: Bewertung Trainingsphase [Koc21](Experiment 3)

	precision	recall
Top	0.85	0.87
Trouser	0.99	0.98
Pullover	0.92	0.84
Dress	0.93	0.92
Coat	0.86	0.92
Sandal	0.98	0.98
Shirt	0.83	0.69
Sneaker	0.95	0.97
Bag	0.99	0.98
Ankle boot	0.98	0.96

Abbildung 4.11: Bewertung Testphase [Koc21](Experiment 3)

Das Modell erzielt bei jeder Klasse unterschiedliche Präzision und Trefferquote, dabei werden die Klassen, wie zum Beispiel die Bag, Sandal, Ankle Boot, Dress, Sneaker und Trouser mit Präzision von 95% bis 99% prognostiziert. Die Trefferquo-

te zu diesen Klassen sind im Bereich von 92% bis 98%. Somit stellt das Modell für diese Klassen eine gute Leistung dar. Als nächstes werden die Klassen ausgewählt die schlechte Leistungen in dem aktuellen Modell erbringen. Die Klassen Shirt, Coat und Top haben eine Präzision von 83% bis 86% und deren Trefferquote liegt zwischen 69% bis 92%, dabei hat die Klassen Coat eine höhere Trefferquote (92%) als seine Präzision (86%). Aufgrund der Überanpassung im Modell, welches durch die Trainingsphase entdeckt wurde, muss zunächst das Modell im nächsten Schritt 4.4.2 Experiment 2 optimiert werden.

4.5.2 Evaluation Experiment 2

Bisher wurde das Experiment 2 erstellt und es sollten keine Überanpassungen mehr stattfinden. Mit diesem Schritt Testen wir das optimierte Modell neu, um Änderungen bei den Metriken zu erkennen.

Testphase

Experiment 1 / Experiment 2	precision		recall	
Top		0.85		0.87
Trouser		0.99	0.98	0.99
Pullover	0.92	0.91	0.84	0.85
Dress	0.93	0.95	0.92	0.90
Coat	0.86	0.89	0.92	0.84
Sandal	0.98	0.99	0.98	0.97
Shirt	0.83	0.81	0.69	0.70
Sneaker	0.95	0.96	0.97	0.96
Bag		0.99		0.98
Ankle boot	0.98	0.95	0.96	0.97

Abbildung 4.12: Bewertung Testphase mit Vergleichswerten aus dem Experiment 1 [Koc21](Experiment 3)

Im Abbild 4.12 sind die neue Lösungen des optimierten Modells mit den Werten von dem zuvor erstelltem Modell zu sehen. Die Ergebnisse aus dem letzten Modell wurde nur dann aufgeschrieben, wenn es auch eine Änderung gegeben hat. So ist es ersichtlicher was im Modell wirklich optimiert wurde und was nicht. Eine Änderung der Leistungen des Modell wird deutlich durch die Abbildung 4.12. Die zuvor im Evaluationsschritt ausgewählten Klassen die gute Leistung erbracht haben, werden zunächst im aktuellen Modell betrachten und verglichen. Um diese einfacher zu beschreiben listen wird die Differenzen der Leistungen der Klassen.

Klasse	Differenz Präzision	Differenz Trefferquote
Top	0	0
Trouser	0	+1%
Pullover	-1%	+1%
Dress	+2%	-2%
Coat	+3%	-8%
Sandal	+1%	-1%
Shirt	-2%	+1%
Sneaker	+1%	-1%
Bag	0	0
Ankle boot	-3%	+1%

Tabelle 4.2: Differenz neue und alte Ergebnisse

Durch die Tabelle 4.2 werden die Ergebnisse deutlicher beschrieben. Durch das Verhindern der Überanpassung wurde die Klassifikation im Modell für die Klassen angepasst. Da dieses Modell nur die Überanpassung bearbeitet hat ist auch demnach keine Verbesserung des Modell deutlich. Die Trefferquoten wurden bei den Klassen Trouser, Shirt und Pullover um 1% verbessert. Die Klassen Coat und Dress haben sich im gegenzug verschlechtert. Leider wurde das Modell nicht gut optimiert, daher wäre die Empfehlung in diesem Fall, den sprung zur Modellierung 4.4.3 zurück zu machen.

4.5.3 Evaluation Experiment 3

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        320

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 14, 14, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 32)         0
 2D)

 flatten (Flatten)           (None, 1568)              0

 dense (Dense)               (None, 28)                43932

 dense_1 (Dense)             (None, 10)                290

=================================================================
Total params: 53,790
Trainable params: 53,790
Non-trainable params: 0
_________________________________________________________________
```

Abbildung 4.13: Bewertung Testphase [Koc21](Experiment 3)

TODO: Welches Ergebnisse werden geliefert wie gut ist das modell!

4.6 Deployment

4.7 Zusammenfassung und Ausblick

Literaturverzeichnis

[Abba] *Aufbau eines CNN.* https://www.researchgate.net/figure/Allgemeiner-Aufbau-eines-CNN-In-Anlehnung-an-MathWorks-2017_fig4_342750457, Zuletzt aufgerufen am 12.11.2021.

[Abbb] *CRISP-DM Modell.* https://statistik-dresden.de/wp-content/uploads/2012/04/CRISP-DM_Process_Diagram1.png, Zuletzt aufgerufen am 12.11.2021.

[Abbc] *FASHION-MNIST.* https://github.com/zalandoresearch/fashion-mnist/blob/master/doc/img/fashion-mnist-sprite.png, Zuletzt aufgerufen am 12.11.2021.

[Abbd] *KNN vs. DL.* https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0192-5, Zuletzt aufgerufen am 12.11.2021.

[Abbe] *Modell eines Neurons.* https://datasolut.com/wp-content/uploads/2019/11/neuronale_netze_aktivierung-1024x485.png, Zuletzt aufgerufen am 12.11.2021.

[Abbf] *Schichten der KNN.* https://datasolut.com/wp-content/uploads/2019/10/k%C3%9Cnstliche-neuronale-Netze.jpg, Zuletzt aufgerufen am 12.11.2021.

[Abbg] *Verteilung Testdatensatz.* https://github.com/zalandoresearch/fashion-mnist, Zuletzt aufgerufen am 12.11.2021.

[Abbh] *Verteilung Trainingsdatesatz.* https://github.com/zalandoresearch/fashion-mnist, Zuletzt aufgerufen am 12.11.2021.

[Asi18] Sidath Asiri. Machine learning classifiers. 2018. https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623, Zuletzt aufgerufen am 27.02.2022.

[Bur19] Andriy Burkov. *Machine Learning Kompakt.* mitp Verlag, 2019.

[C.18] Hoefer C. Wie amazon, zalando und otto mit retouren umgehen. *manager magazin*, 2018. https://www.manager-magazin.de/unternehmen/handel/retouren-wie-amazon-zalando-und-otto-mit-ruecksendungen-umgehen-a-1213.html, Zuletzt aufgerufen am 13.11.2021.

[Cho18] François Chollet. *Deep Learning mit Python und Keras : Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek*. [Frechen] : mitp Verlags, 2018.

[FG18] Fraunhofer-Gesellschaft. Maschinelles lernen eine analyse zu kompetenzen,forschung und anwendung. *Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V*, 2018. https://www.bigdata-ai.fraunhofer.de/content/dam/bigdata/de/documents/Publikationen/Fraunhofer_Studie_ML_201809.pdf, Zuletzt aufgerufen am 22.02.2022.

[Fro21] Jörg Frochte. *Maschinelles Lernen : Grundlagen und Algorithmen in Python*. Hanser eBook, 2021.

[Hal22] Halil. *Feed Forward Netz*, 2022. https://github.com/halil-kocak/Abschlussarbeit-ErkennungVonKleiderstuecken/blob/main/Dokumentation/FF_Netz.PNG, Zuletzt aufgerufen am 21.02.2022.

[Koc21] Halil Kocak. *Jupyter Notebook: Erkennung von Kleiderstücken*, 2021. https://github.com/halil-kocak/Abschlussarbeit-ErkennungVonKleiderstuecken/blob/main/Code/defaultCNN/default_cnn.ipynb, Zuletzt aufgerufen am 25.02.2021.

[Lub19] Dipl.-Ing. Stefan Luber. Was ist ein convolutional neural network? *BIGDATA INSIDER*, 2019. https://www.bigdata-insider.de/was-ist-ein-convolutional-neural-network-a-801246/, Zuletzt aufgerufen am 12.11.2021.

[Mah18] Hamza Mahmood. The softmax function, simplified. *Towards Data Science*, 2018. https://towardsdatascience.com/softmax-function-simplified-714068bf8156, Zuletzt aufgerufen am 22.02.2022.

[Mat21] Stephan Matzka. *Künstliche Intelligenz in den Ingenieurwissenschaften*. Springer Fachmedien Wiesbaden, 2021.

[Pla21] Matthias Plaue. *Data Science*. SpringerSpektrum, 2021.

[Raber] L. Rabe. Jährliche anzahl der bestellungen bei zalando bis 2020. *Statista*, 2021, 10. November. https://de.statista.com/statistik/daten/studie/325862/umfrage/anzahl-aller-bestellungen-bei-zalando-pro-jahr/, Zuletzt aufgerufen am 12.11.2021.

[Sal14] Nitish Srivastava; Geoffrey Hinton; Alex Krizhevsky; Ilya Sutskever; Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks fromoverfitting. *Journal of Machine Learning Research 15*, 2014. https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf, Zuletzt aufgerufen am 27.02.2022.

[Sch19] Sarah Schönbrodt. *Maschinelle Lernmethoden für Klassifizierung Probleme*. Springer Spektrum, 2019.

[SE17] Zalando SE. Fashion-mnist [datensatz]. 2017. https://github.com/zalandoresearch/fashion-mnist, Zuletzt aufgerufen am 10.11.2021.

[ten22] tensorflow. *TensorFlow Doku: Dropout*, 2022. https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout, Zuletzt aufgerufen am 27.02.2022.

[Vog21] Sandro Scheid; Stefanie Vogl. *Data Science*. Carl Hanser Verlag GmbH und Co. KG, 2021.

[Wut19] Laurenz Wuttke. Crisp-dm: Grundlagen, Ziele und die 6 Phasen des Data Mining Prozess. *datasolut*, 2019. https://datasolut.com/crisp-dm-standard/, Zuletzt aufgerufen am 10.11.2021.