GYMNASIUM DER STADT MESCHEDE

FACHARBEIT

Künstliche neuronale Netze am Beispiel der Klassifizierung von Scandaten

Autor:
Lukas MERTENS

Betreuer:
C. LÖSER

Konzeption und Implementierung künstlicher neuronaler Netze zur Entwicklung lernfähiger Algorithmen

Informatik
Abitur 2019

10. Juni 2018

GYMNASIUM DER STADT MESCHEDE

Zusammenfassung

Informatik
Abitur 2019

Schüler

Künstliche neuronale Netze am Beispiel der Klassifizierung von Scandaten

von Lukas MERTENS

Diese Facharbeit beinhaltet eine Einführung in den Aufbau und die grundlegende Implementierung künstlicher neuronaler Netze zur Entwicklung lernfähiger Algorithmen. Das Wissen wird anschließend zur Entwicklung einer lernfähigen Anwendung zur Erkennung von unbedruckten Seiten bei Dokumentenscans genutzt...

Inhaltsverzeichnis

Abbildungsverzeichnis

Herkunft der Abbildungen

Alle verwendeten Abbildungen wurden vom Autor dieser Arbeit zum Zwecke der besseren Verständlichkeit und Lesbarkeit in Eigenleistung erstellt.

Tabellenverzeichnis

Abkürzungsverzeichnis

KNN	**k**ünstliches **n**euronales **N**etz
DNN	**D**eep **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
MLP	**M**ulti**l**ayer **P**erceptron
KI	**k**ünstliche **I**ntelligenz
CI	**C**ontinuous **I**ntegration
TDD	**T**est **D**riven **D**evelopment

Kapitel 1

Einleitung

Künstliche Intelligenz ist schon lange ein Thema, sowohl in Science-Fiction Literatur als auch in Filmen. Auch in der Philosophie spielt das Thema eine große Rolle. Während universelle künstliche Intelligenz noch auf sich warten lässt, konnten in den letzten Jahren große Durchbrüche mit aufgabenspezifischer künstlicher Intelligenz erzielt werden. Besonders große Fortschritte konnten bei der Bilderkennung und Spracherkennung erzielt werden. Aber auch kreative Aufgaben, wie das Komponieren von Musik oder das Generieren von Gemälden können durch künstliche neuronale Netze immer weitgehender übernommen werden. Künstliche neuronale Netze und speziell Deep Learning haben das Potenzial ganze Branchen zu revolutionieren, sowohl positiv als auch negativ.

Diese Arbeit soll einen Einstieg in das umfangreiche Feld des maschinellen Lernens geben. Der Fokus liegt dabei auf künstlichen neuronalen Netzen. Die Arbeit hat nicht den Anspruch einen kompletten Überblick über das weite Feld der künstlichen neuronalen Netze zu geben. Stattdessen wird eine Technik beispielhaft detailliert erklärt, damit der Leser die Funktionsweise eines einfachen KNNs vollständig versteht. Das Ziel dieser Arbeit ist erreicht, wenn ein einfaches KNN für den Leser keine „Blackbox“ mehr ist, sondern als einfaches mathematisches Modell verstanden wird.

Im Anschluss an die Einleitung in künstliche neuronale Netze wird der Entwicklungsprozess einer Onlineplattform zur Klassifizierung von Scandaten dokumentiert. Die Onlineplattform verwendet maschinelles Lernen, um gescannte Seiten in bedruckte bzw. beschriebene und leere Seiten zu klassifizeren.

Diese Arbeit beschäftigt sich nur mit den Möglichkeiten künstlicher neuronaler Netze. Dem Leser wird geraten sich, mit den ethischen und politischen Problemen künstlicher Intelligenz mithilfe von anderer Literatur auseinanderzusetzen.

Kapitel 2

Künstliche vs. Natürliche neuronale Netze

2.1 Natürliche neuronale Netze

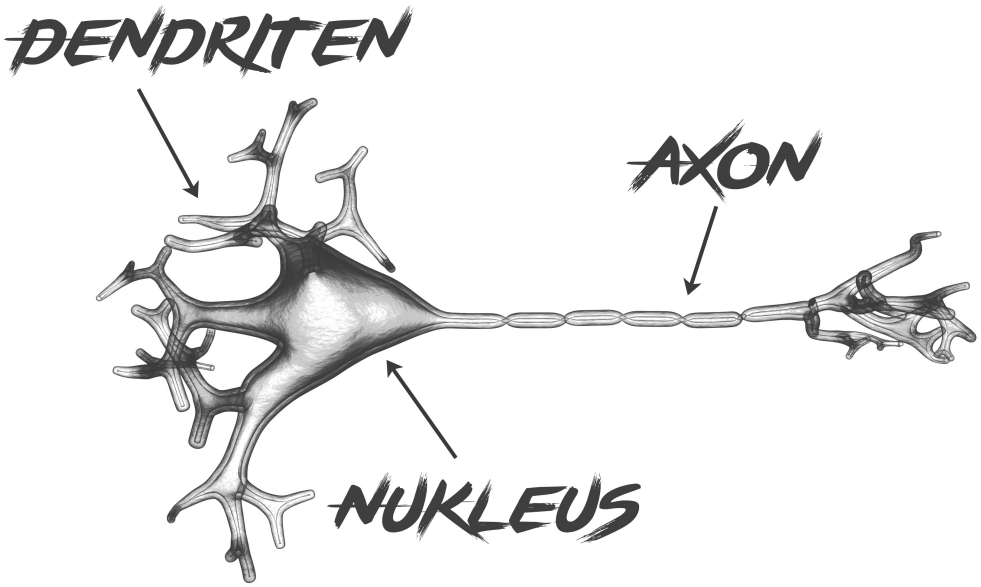

ABBILDUNG 2.1: Stark vereinfachter Aufbau der natürlichen Nervenzelle

Natürliche neuronale Netze sind aus der Biologie bekannt. Das menschliche Gehirn besteht aus ca. 86,6 Milliarden Neuronen (vgl. Azevedo u. a. 2009). Die Nuklei der Nervenzellen sind durch Axone und Dendriten miteinander stark vernetzt (siehe Abb. 2.1) (vgl. Soares und Souza 2016). Über die Dendriten empfangen die Nervenzellen Informationen von anderen Nervenzellen, die anschließend im Nukleus verarbeitet und über das Axon an weitere Nervenzellen weitergeleitet werden (vgl. National Institute on Alcohol Abuse and Alcoholism 1997). Die Informationen werden in Form von chemischen und elektrischen Signalen übertragen, und das natürliche neuronale Netz lernt durch Ausbildung neuer Verbindungen zwischen den Nervenzellen.

2.2 Künstliche neuronale Netze

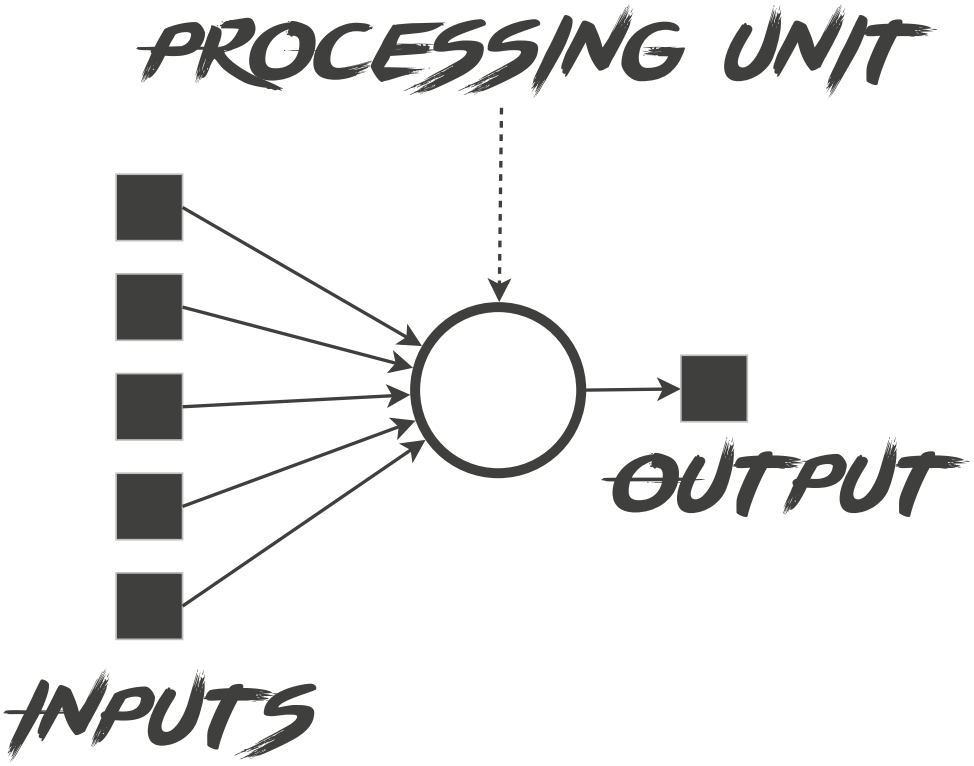

ABBILDUNG 2.2: Vereinfachter Aufbau des künstlichen Neurons

Künstliche neuronale Netze haben viele Ähnlichkeiten zu natürlichen neuronalen Netzen. Auch im künstlichen neuronalen Netz nimmt jedes Neuron unterschiedliche Informationen auf und gibt anschließend ein Signal als Output aus (siehe Abb. 2.2) (vgl. Soares und Souza 2016). Die Anordnung der Neuronen ist allerdings typischerweise in Ebenen aufgebaut. Die Verbindungen zwischen den Neuronen liegen in Form von Gewichten vor (siehe Abb. 3.1). Während des Trainings werden diese Gewichte angepasst, bis das KNN die gewünschte Funktionsweise aufweist.

Kapitel 3

Einführung in die Funktionsweise eines KNNs

3.1 Multilayer Perceptron

Um einen ersten, tieferen Einblick in die Funktionsweise eines KNNs zu erhalten, werden im Folgenden Multilayer Perceptrons genauer beschrieben. Es handelt sich dabei nur um eine von sehr vielen Klassen künstlicher neuronaler Netze, viele Techniken der Multilayer Perceptrons sind allerdings auch Kernbestandteil anderer Klassen künstlicher neuronaler Netze.

3.1.1 Input- und Outputlayer

Mit die einfachste Form eines KNNs besteht aus nur zwei Layern, einem Input- und einem Output-Layer (siehe Abb. 3.1) (vgl. Soares und Souza 2016). Die einzelnen Layer sind durch Gewichte verbunden. Jedes Neuron ist mit jedem Neuron des vorangehenden Layers durch jeweils ein Gewicht verbunden.

Im Input-Layer wird für jedes Neuron eine Zahl übergeben. Soll beispielsweise ein Graustufen-Bild mit 28×28 Pixeln durch ein KNN untersucht werden, benötigt das KNN $28 \times 28 = 784$ Neuronen im Input-Layer. Im Output-Layer steht nach Ausführung des KNNs in jedem Neuron eine Zahl. Diese muss interpretiert werden: In einem zur Lösung eines Klassifizierungsproblems genutzten KNN wird typischerweise ein Ausgabeneuron pro Kategorie verwendet. Das Neuron mit dem höchsten Wert stellt die Kategorie dar, in die das KNN die Eingabe klassifiziert hat.

3.1.2 Hidden-Layer

Bei komplexeren Problemen sind mehr Layer erforderlich, da die Entropie des Netzes nicht ausreicht, um für jede Eingabe eine sinnvolle Ausgabe zu erzielen. Layer zwischen Eingabe- und Ausgabe-Layer werden Hidden-Layer genannt, da Zugriff auf die Layer von außen weder sinnvoll noch möglich ist. Die zusätzlichen Layer ermöglichen dem Netz eine größere Anzahl an Mustern in mehreren Stufen in den Trainingsdaten zu erkennen, die später eine genauere Klassifizierung ermöglichen.

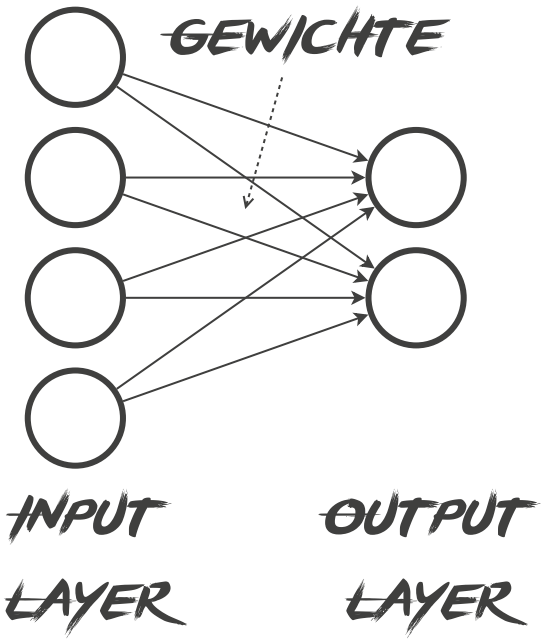

ABBILDUNG 3.1: Aufbau eines einfachen Feedforward-Netzes

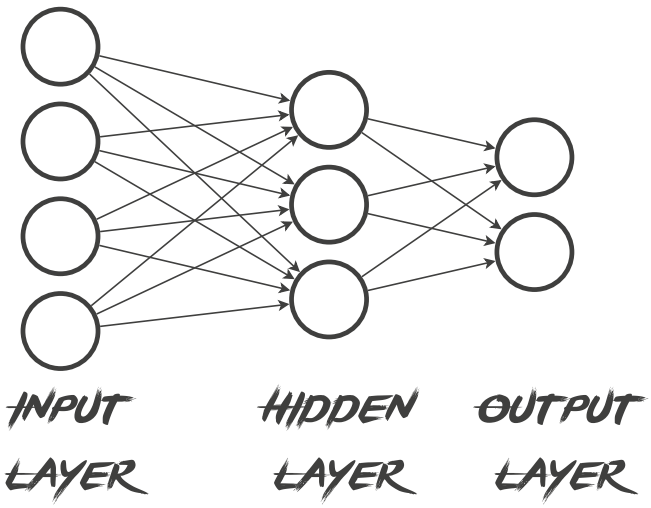

ABBILDUNG 3.2: Aufbau eines einfachen Feedforward-Netzes mit Hidden-Layer

3.1.3 Gewichte

Jedes Gewicht w_i ist letztlich nur eine reelle Zahl, die mit dem Output-Wert a_i des entsprechenden Vorgängerneurons multipliziert wird. Jedes Neuron bildet also eine gewichtete Summe aller Vorgängerneuronen:

$$A = w_0 * a_0 + w_1 * a_1 + \cdots + w_n * a_n = \sum_{i=0}^{n} w_i * a_i$$

3.1.4 Aktivierungsfunktionen

Das Ergebnis dieser Rechnung kann beliebig groß werden, was bei einem gewollten Output zwischen 0 und 1 im Output-Layer eines Klassifizierungs-KNNs problematisch ist (vgl. Soares und Souza 2016). Deshalb wird eine sogenannte Aktivierungsfunktion verwendet, die A in vielen Anwendungsfällen in den Bereich $0 \leq A \leq 1$ bzw. $-1 \leq A \leq 1$ bringt. Aber Aktivierungsfunktionen erfüllen noch weitere Aufgaben: Durch Aktivierungsfunktionen werden irrelevante Informationen, die keinen maßgeblichen Einfluss auf das Endergebnis haben, also nur Rauschen sind, aussortiert (vgl. Gupta 2017). Das funktioniert, weil die einzelnen Neuronen erst ab gewissen Grenzwerten nennenswert feuern. Außerdem sind Aktivierungsfunktionen nötig, um ein KNN nichtlinear zu machen.

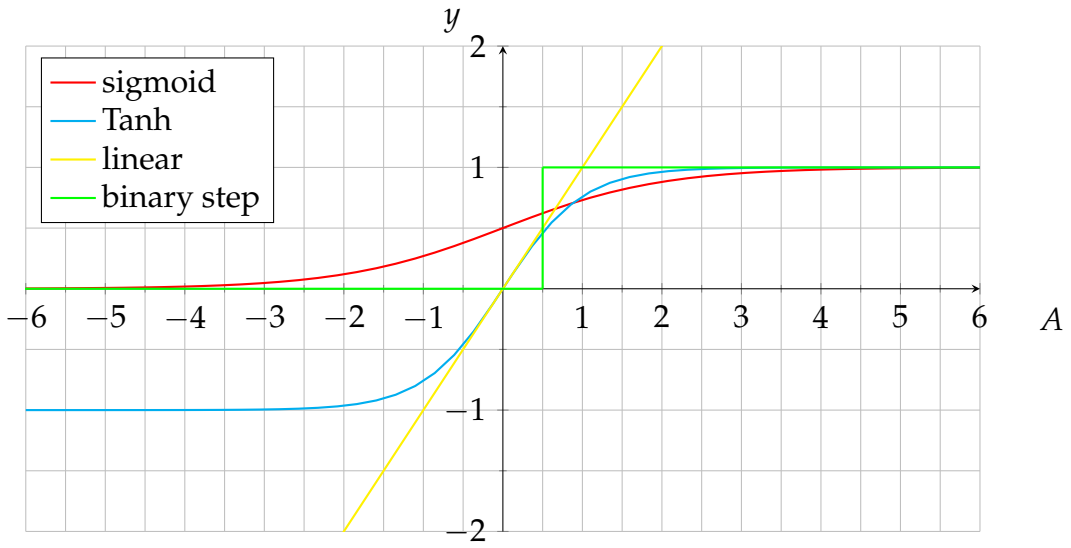

ABBILDUNG 3.3: Graphen der wichtigsten Aktivierungsfunktionen [1/2]

Neben dem Aussortieren von unnötigen Informationen sind Aktivierungsfunktionen auch notwendig, um Nichtlinearität in das KNN zu bringen. Da ohne Aktivierungsfunktion nur die Gewichte mit den Outputs des Vorgängerneurons multipliziert werden, kann das KNN ohne Aktivierungsfunktion nur lineare Operationen durchführen.

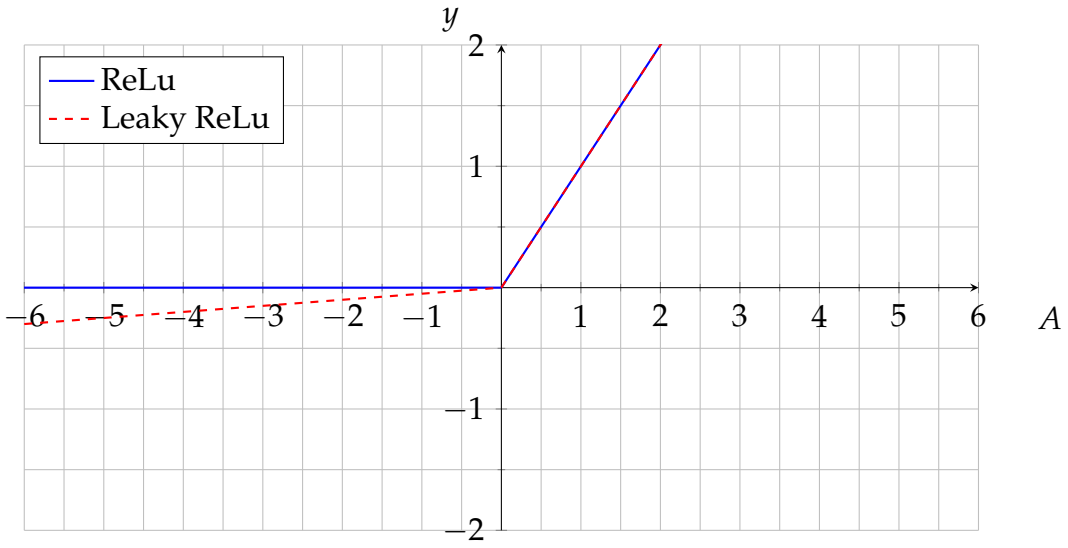

ABBILDUNG 3.4: Graphen der wichtigsten Aktivierungsfunktionen [2/2]

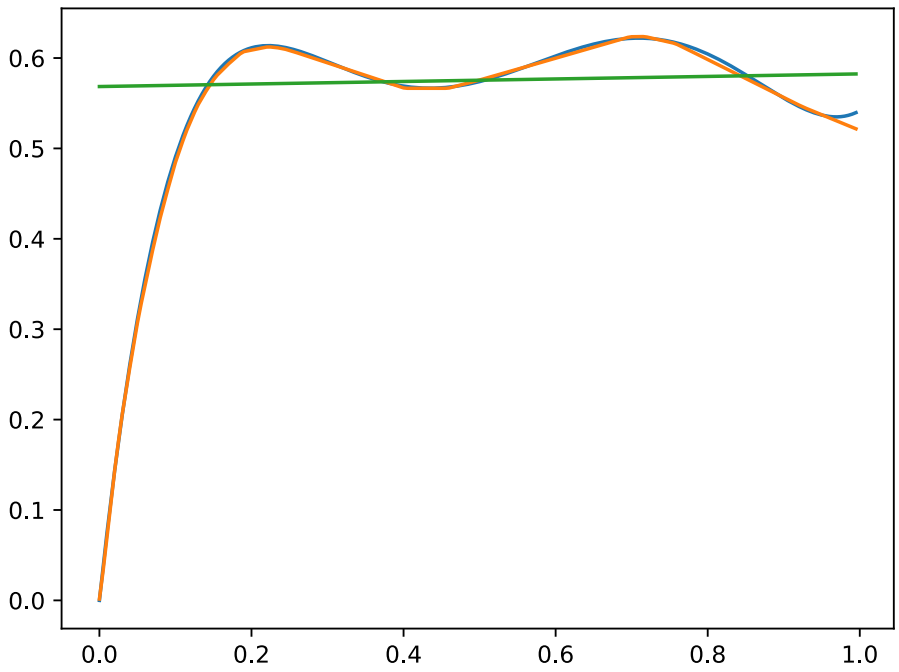

ABBILDUNG 3.5: Ausschließlich ReLu (orange) vs. ausschließlich lineare Aktivierungsfunktion (grün) in den Hidden-Layern

Aktivierungsfunktion	Funktion
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$
Tanh	$f(x) = 2 * \texttt{sigmoid}(2x) - 1 = \frac{2}{1+e^{-2x}} - 1$
ReLu	$f(x) = \texttt{max}(0, x)$
Leaky ReLu	$f(x) = \begin{cases} ax & x < 0 \\ x & x \geq 0 \end{cases}$
Linear	$f_a(x) = a * x$
Binary Step	$f(x) = \begin{cases} 1 & x \geq t \\ 0 & x < t \end{cases}$
Softmax	$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$

TABELLE 3.1: Überblick über die wichtigsten Aktivierungsfunktionen ($t \hat{=}$threshold)

Binary Step

Die binary step-Aktivierungsfunktion ist sehr einfach: Ab einem gewissen Schwellenwert wird die Ausgabe des Neurons 1, sonst 0. Die Aktivierungsfunktion kann beispielsweise für einen binären Klassifizierer genutzt werden, da die Neuronen nur binäre Ausgaben erzeugen. Ein Nachteil der Funktion ist, dass ihr Gradient immer Null ist. Das führt dazu, dass während der Backpropagation-Phase keine nennenswerten Verbesserungen an dem Netz zustande kommen (siehe Abschnitt 3.2.2).

Linear

Eine lineare Aktivierungsfunktion ist vollständig differenzierbar. Der Gradient von $f_a(x)$ ist allerdings konstant:

$$f_a'(x) = a$$

Backpropagation ist also auch bei einer linearen Aktivierungsfunktion problematisch.

Eine lineare Aktivierungsfunktion hat allerdings durchaus einen sinnvollen Einsatzzweck: Im Gegensatz zu den anderen vorgestellten Aktivierungsfunktionen ist die lineare Aktivierungsfunktion nicht auf einen bestimmten Wertebereich beschränkt. Soll das KNN also zur Regression benutzt werden, ist eine lineare Aktivierungsfunktion im Output-Layer sinnvoll, da der Output des KNNs so nicht auf einen bestimmten Wertebereich beschränkt ist.

Sigmoid

Ein großer Vorteil der Sigmoid-Aktivierungsfunktion ist die Nichtlinearität der Funktion. Bei einem KNN mit einigen Neuronen, die Sigmoid als Aktivierungsfunktion verwenden, ist die Ausgabe des KNNs automatisch auch nichtlinear. Außerdem ist die Funktion vollständig differenzierbar, kann also durch Backpropagation optimiert werden.

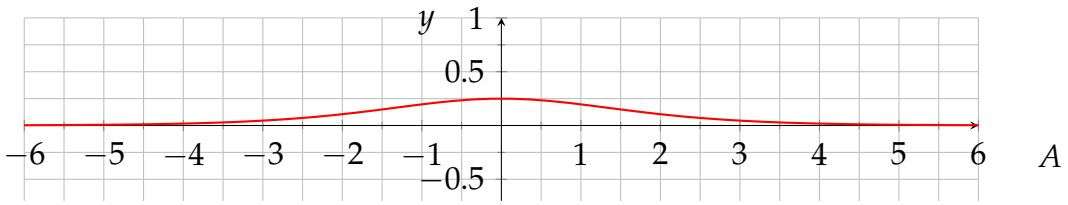

ABBILDUNG 3.6: Gradient der Sigmoid-Funktion

Der Gradient der Sigmoid-Funktion wird außerhalb von $-3 \leq x \leq 3$ sehr flach (siehe Abb. 3.6), also annähernd konstant, sodass das Netz bei sehr kleinen bzw. großen Werten kaum lernt.
Ein weiteres Problem der Sigmoidfunktion ist, dass der Output immer positiv ist. Das ist im Output-Layer eines Klassifizierungs-KNNs zwar sinnvoll,

in den Hidden-Layern aber nicht, da ein einzelnes Neuron mit Sigmoid-Aktivierungsfunktion so keinen negativen Einfluss auf die gewichtete Summe eines Neurons des Folgelayers haben kann.

Tanh

Tanh ist im Prinzip nur eine skalierte und verschobene Sigmoid-Funktion. Somit ist die Aktivierungsfunktion auch vollständig differenzierbar, bringt aber auch das Problem mit dem gegen Null konvergierenden Gradienten mit. Die Funktion ist allerdings punktsymmetrisch zum Ursprung und hat den Wertebereich $-1 < f(x) < 1$, ermöglicht also auch negative Ausgaben.

ReLu

ReLu ist nichtlinear und für Backpropagation gut nutzbar. Ein großer Vorteil der Aktivierungsfunktion ReLu ist, dass nicht alle Neuronen gleichzeitig aktiv sind. Da alle negativen Werte durch ReLu zu Null werden, werden diese Neuronen deaktiviert, was Geschwindigkeitsvorteile bringt.

Aus diesen Gründen ist ReLu die heutzutage meistgenutzte Aktivierungsfunktion. Allerdings gibt es auch bei ReLu ein Problem: Bei negativen Werten ist der Gradient Null. Dadurch können „tote“ Neuronen entstehen, die nie aktiviert werden.

Leaky ReLu

Leaky ReLu ist eine verbesserte Version von ReLu, die das Problem mit den „toten“ Neuronen löst. Statt dem konstanten Nullwert für negative Werte, wird eine Gerade verwendet. Ist die Steigung a dieser Gerade ein durch das KNN trainierbarer Parameter, wird die Aktivierungsfunktion auch *Parameterised ReLu* genannt.

Softmax

Die Softmax-Aktivierungsfunktion kommt im Normallfall nur im Output-Layer zum Einsatz. Sie wird bei Klassifizierungs-KNNs verwendet, um die Wahrscheinlichkeit, dass sich ein Objekt in einer bestimmten Klasse befindet, zu bestimmen. Sie bringt dafür die Summe aller Output-Neuronen auf 1:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Zusammenfassung

Der Output eines Neurons mit beispielsweise Sigmoidfunktion als Aktivierungsfunktion würde also wie folgt berechnet:

$$\sigma\left(\sum_{i=0}^{n} w_i * a_i\right)$$

Die Formel zur Berechnung der gewichteten Summe wurde nur um die Aktivierungsfunktion erweitert.

3.1.5 Bias

In dieser Form wird ein Neuron aktiviert, sobald die gewichtete Summe einen bestimmten Wert erreicht. Ab welchem Wert das der Fall ist, ist aber bei jedem Neuron mit der gleichen Aktivierungsfunktion gleich. Damit genauso wie im menschlichen Gehirn immer nur die nötigen Neuronen aktiv sind, wird ein Bias-Wert zu der gewichteten Summe hinzugerechnet bzw. abgezogen (vgl. Goodfellow, Bengio und Courville 2016).

$$\sigma\left(b + \sum_{i=0}^{n} w_i * a_i\right)$$

Dieser Bias ist neben den Gewichten ein weiterer Wert, der während der Trainingsphase optimiert wird, um zu einem bestmöglichen Ergebnis zu kommen.

3.1.6 Kompakte Notation der Berechnungen in einem Layer

Die Berechnung der Ausgabe eines Layers kann allerdings noch weiter vereinfacht werden, indem man die Ausgaben des vorherigen Layers, die Gewichte und den Bias jedes Neurons als Matrizen bzw. Vektoren notiert (vgl. Sanderson 2017; Sharlene Katz 2004).

$$a^{(1)} = \sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & \dots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \dots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_k \end{bmatrix}\right) = \sigma\left(Wa^0 + b\right)$$

Diese kurze Notation macht es sehr einfach in Libraries wie beispielsweise *TensorFlow* (Erläuterung siehe Abschnitt 5.2.2) eine Netzarchitektur zu entwerfen:

```
x = tf.placeholder(tf.float32, shape=[None, 784])
y = tf.placeholder(tf.float32, shape=[None, 10])

w = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

sess.run(tf.global_variables_initializer())

y_ = tf.matmul(x,w) + b
```

LISTING 3.1: Auszug aus einem TensorFlow-Programm zur Klassifizierung des MNIST-Datensatzes

Hier ist x eine Matrix mit den Trainingsdaten des MNIST-Datensatzes (Erläuterung siehe Abschnitt 5.2.1) und y eine Matrix mit der Klassifizierung

des KNNs. Es handelt sich bei diesen Variablen um `placeholder`. Das bedeutet, dass die Variablen von TensorFlow nicht trainiert werden, sondern als Schnittstelle des KNNs zum restlichen Programm fungieren.

Die Variablen w und b sind Matrizen, die während dem Training optimiert werden. Bei w handelt es sich um die Gewichte, die mit den Ausgaben des Vorgängerneurons multipliziert werden (siehe Abschnitt 3.1.3). Da es hier nur eine Eingabe- und eine Ausgabeschicht gibt, werden die Gewichte hier mit der Eingabeschicht, also mit x multipliziert (siehe Listing 3.1, Zeile 9). Die Variable b speichert den Bias-Wert jedes Neurons der Ausgabeschicht (Erläuterung siehe Abschnitt 3.1.5) und wird nach der Matrixmultiplikation mit den Gewichten auf das Ergebnis addiert.

Die meisten Libraries bringen viele Optimierungen zur Steigerung der Trainingsgeschwindigkeit mit. Die rechenintensiven Matrix-Multiplikationen werden beispielsweise auf der Grafikkarte ausgeführt, um die einzelnen Teiloperationen zu parallelisieren. (vgl. Goodfellow, Bengio und Courville 2016; TensorFlow 2017a).

3.1.7 Allgemeinheit von KNNs

Das KNN bildet letztlich nur eine Funktion zwischen Eingabe- und Ausgabe-Layer (vgl. Nielsen 2015). Das lässt sich sehr einfach durch ein einfaches KNN repräsentieren, das auf eine Funktion $f(x)$ trainiert wird und anschließend die den x-Werten entsprechenden y-Werte berechnet. In diesem Beispiel wurde die Funktion $f(x) = 0,7864035 * x - 0,3636947 * x^2 + 0,07456 * x^3 - 0,006923075 * x^4 + 0,0002371795 * x^5$ verwendet.

Die Funktion liefert für jeden x-Wert genau einen y-Wert. Das Netz benötigt also ein Eingabeneuron und ein Ausgabeneuron. Zwischen der Eingabe- und Ausgabeschicht befinden sich beliebig viele Hidden-Layer mit beliebig vielen Neuronen.

Das Beispiel wurde hier in Keras, einer High-Level Machine Learning Library für Python implementiert (siehe Abschnitt 5.3.1)

Im ersten Schritt werden die NumPy-Arrays X und Y deklariert, die später einige Beispielwerte für x-Werte und dazugehörige y-Werte enthalten sollen.

```
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
import matplotlib.pyplot as plt

# declare and init arrays for training-data
X = np.arange(0.0, 1.0, 0.005)
Y = np.empty(shape=0, dtype=float)
```

LISTING 3.2: Module importieren und NumPy-Arrays deklarieren

Im nächsten Schritt werden die Y-Werte zu den entsprechenden X-Werten berechnet.

```
# Calculate Y-Values
for x in np.arange(0.0, 10.0, 0.05):
```

```
    Y = np.append(Y, float(0.7864035*x - 0.3636947*x**2 + 0.07456*x
    **3 - 0.006923075*x**4 + 0.0002371795*x**5))
```

LISTING 3.3: Module importieren und NumPy-Arrays deklarieren

Nach der Erzeugung der Trainingsdaten wird die Netzarchitektur bestimmt

```
# model architecture
model = Sequential()
model.add(Dense(1, input_shape=(1,)))
model.add(Dense(1000, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(1, activation='linear'))
```

LISTING 3.4: Netzarchitektur

und das Model kompiliert, um eine schnelle Ausführung auf der Grafikkarte zu ermöglichen.

```
# compile model
model.compile(loss='mean_absolute_error', optimizer='adam')
```

LISTING 3.5: Model kompilieren

Im Anschluss wird das Netz auf die Beispielwerte der Funktion trainiert.

```
# train model
model.fit(X, Y, epochs=500, batch_size=50)
```

LISTING 3.6: Training des Netzes

Nach dem Training wird für jeden x-Wert ein y-Wert durch das KNN bestimmt und das Ergebnis mithilfe von Matplotlib grafisch dargestellt.

```
# declare and init arrays for prediction
YPredict = np.empty(shape=0, dtype=float)

# Predict Y
YPredict = model.predict(X)

# plot training-data and prediction
plt.plot(X, Y, 'C0')
plt.plot(X, YPredict, 'C1')

# show graph
plt.show()
```

LISTING 3.7: Model kompilieren

Trainiert man das Netz nun mit unterschiedlichen Netzarchitekturen auf die Funktion, sieht man, dass die Genauigkeit, die das Netz erreichen kann, von der Anzahl der Neuronen abhängig ist (siehe Abb. 3.7 Abb. 3.8).
An den Graphen lässt sich erkennen, dass das Netz mit der ersten Netzarchitektur größere Probleme hatte, Gewichte für die Hidden-Layer zu finden bei denen das Netz genaue y-Werte voraussagen kann (siehe Abb. 3.7). Mit mehr Neuronen in den Hidden-Layern, konnte das neuronale Netz eine bessere Genauigkeit erzielen (siehe Abb. 3.8).

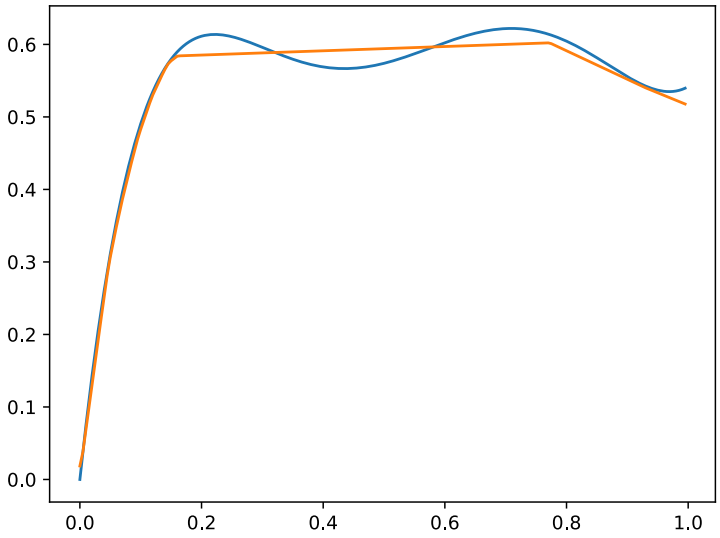

ABBILDUNG 3.7: Trainingsergebnis eines Netzes nach 150 Epochen mit zwei Hidden-Layern: 1. 100 Neuronen, 2. 20 Neuronen - Bei einer geringen Neuronenanzahl kann das KNN nur eine geringe Genauigkeit erreichen: Zwischen etwa $x = 0,2$ und $x = 0,8$ bildet das KNN nur einen Mittelwert der Hoch- und Tiefpunkte.

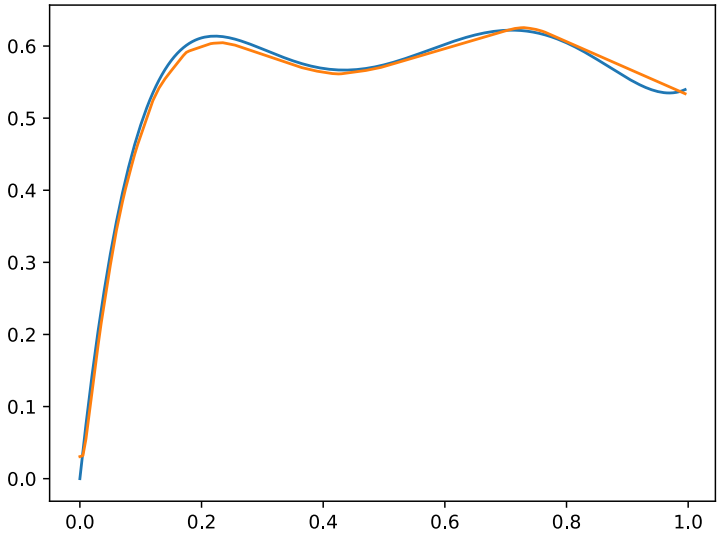

ABBILDUNG 3.8: Trainingsergebnis eines Netzes nach 150 Epochen mit zwei Hidden-Layern: 1. 1000 Neuronen, 2. 100 Neuronen - Bei einer höheren Neuronenanzahl reicht die Entropie schließlich aus, um die Krümmungswechsel des Graphen abzubilden.

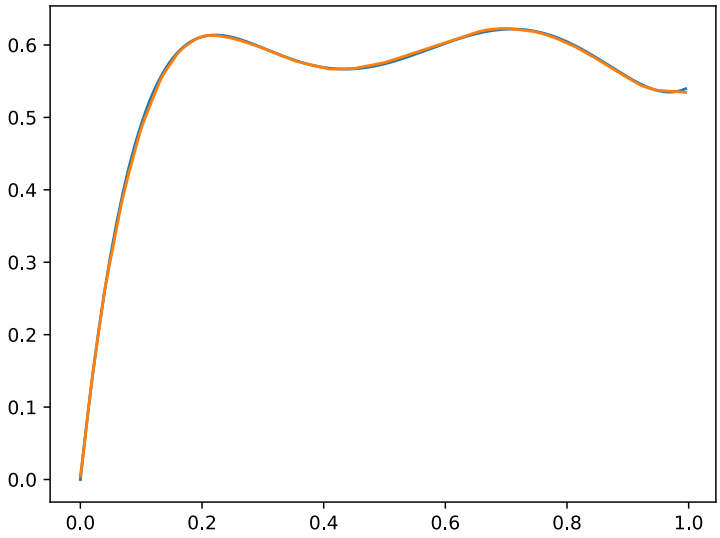

ABBILDUNG 3.9: Trainingsergebnis eines Netzes nach 1500 Epochen mit zwei Hidden-Layern: 1. 1000 Neuronen, 2. 100 Neuronen - Bei einer höheren Epochenanzahl kann der Loss-Wert weiter minimiert werden: Das neuronale Netz ist besser trainiert.

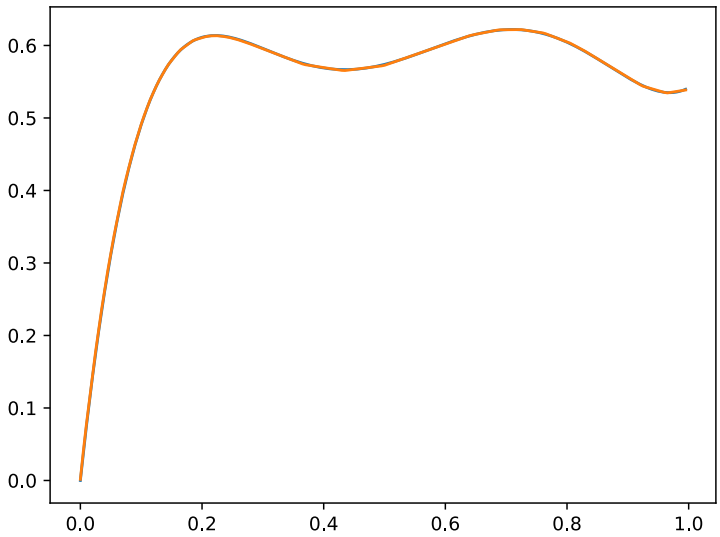

ABBILDUNG 3.10: Nach 50 000 Epochen kann auch mit wenigen Neuronen ein sehr gutes Ergebnis erzielt werden. Hier wurden 100 Neuronen im ersten und 20 Neuronen im zweiten Hidden-Layer verwendet. Ein Training in so vielen Epochen kommt nur aufgrund der geringen Datenmenge und der kleinen Netzarchitektur infrage.

Neben der Netzarchitektur beeinflusst auch die Trainingsdauer das Ergebnis: Durchschnittlich verbessert sich pro Epoche die Leistung des Netzes, bis das Netz gegen ein Leistungsmaximum konvergiert. Ab diesem Leistungsmaximum kann eine Leistungssteigerung nur durch Überarbeitung der Trainingsdaten und/oder durch Veränderung der Netzarchitektur erzielt werden (siehe Abb. 3.9).

Neben dem gezeigten Beispiel kann ein KNN auch auf beliebige andere Funktionen trainiert werden. Meistens ist die Ausgabe auch nicht nur von einem Eingabewert, sondern von vielen hunderten abhängig. Da derartige Beispiele allerdings schwer zu visualisieren sind, wird sich hier auf ein einfaches Beispiel beschränkt. Bei der Verfügbarkeit unendlich vieler Neuronen kann ein KNN also in der Theorie jede beliebige Funktion zwischen Eingabe- und Ausgabelayer darstellen, wie das Experiment veranschaulichen soll.

3.2 Optimierungsalgorithmen

3.2.1 Allgemein

Optimierungsalgorithmen sind einer der Kernbestandteile des Trainingsprozesses. Der Optimierungsalgorithmus passt die Gewichte des KNNs so an, dass die Fehler minimal werden.

3.2.2 Gradient Descent

Ein KNN ist mathematisch gesehen nur eine Funktion zwischen dem Input-Layer und dem Output-Layer (siehe Abschnitt 3.1.7). Die Funktion hat sehr viele Eingabeparameter und einige Rückgabewerte: Eingabeparameter sind die Gewichte, der Bias jedes Neurons und die Eingaben, die das Netz verarbeiten soll (vgl. Goodfellow, Bengio und Courville 2016). Die Rückgabewerte sind die Ausgaben des Output-Layers. Diese Rückgabewerte werden nie dem exakten erwarteten Wert entsprechen (siehe Abb. 3.11), was aber auch nicht nötig ist. Soll das KNN zur Klassifizierung in n Klassen verwendet werden, besteht der Output-Layer aus n Neuronen, die jeweils angeben, wie sicher das KNN den Datensatz in die entsprechende Klasse einordnet. Um nun zu bestimmen, wie gut das KNN die Klassifizierung vorgenommen hat, benötigt man einen einzelnen Wert, der eine Aussage darüber trifft, wie weit die Vorhersage des KNNs von dem gewünschten Ergebnis abweicht.
Eine einfache Möglichkeit, einen solchen Wert für ein Klassifizierungs-KNN zu berechnen (hier am Beispiel aus Abb. 3.11), ist die Summe der quadrierten Differenzen von realen und erwünschten Ausgaben zu berechnen (vgl. Ng 2018):

$$\text{Fehler eines Trainingsbeispiels} \begin{cases} (0,00 - 0,35)^2 + \\ (0,00 - 0,57)^2 + \\ (1,00 - 0,96)^2 + \\ (0,00 - 0,61)^2 + \\ (0,00 - 0,35)^2 \end{cases}$$

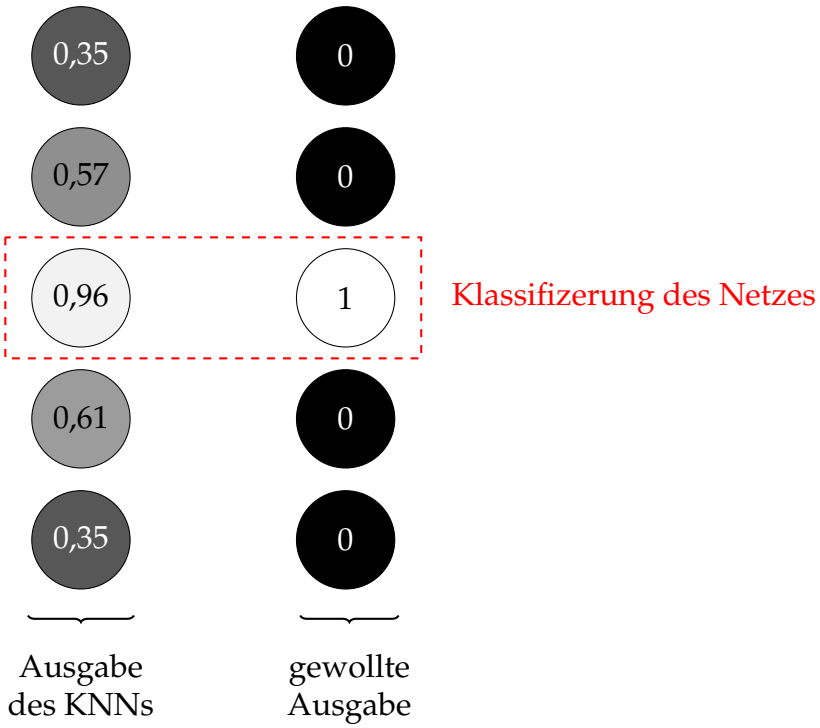

ABBILDUNG 3.11: reale und erwünschte Ausgabe eines Klassifizierungs-KNNs

Allgemein:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

Kombiniert man diese Funktion nun mit der Funktion des gesamten KNNs, erhält man eine Funktion mit sehr vielen Eingabeparametern, die aus den Gewichten, dem Bias jedes Neurons und den Eingaben des Netzes besteht, und einem einzigen Ausgabewert, dem Loss-Wert (vgl. Nedrich 2014). Diese Funktion wird Fehlerfunktion/Loss-Function genannt, da sie abhängig von dem Trainingszustand und den Eingabedaten des KNNs die Stärke der Abweichung vom gewünschten Ergebnis zurückgibt. Bei den Eingabeparametern, die zum globalen Minimum dieser Funktion führen, liegt automatisch das KNN vor, das die optimale Leistung bei gegebenen Eingabewerten liefert.

Lineare Regression

Da eine Funktion mit einigen hunderttausend Eingabeparametern schwer zu visualisieren ist, ist es sinnvoll, zuerst eine einfache Fehlerfunktion zu betrachten: Die der linearen Regression (siehe Abb. 3.12).

Die Fehlerfunktion der linearen Regression hat nur die zwei Eingabeparameter m und b und kann deshalb einfach grafisch dargestellt werden (siehe Abb. 3.13).

$$loss(m, b) = (\frac{1}{N} \sum_{i=1}^{N} (y_i - (m * x_i + b))^2$$

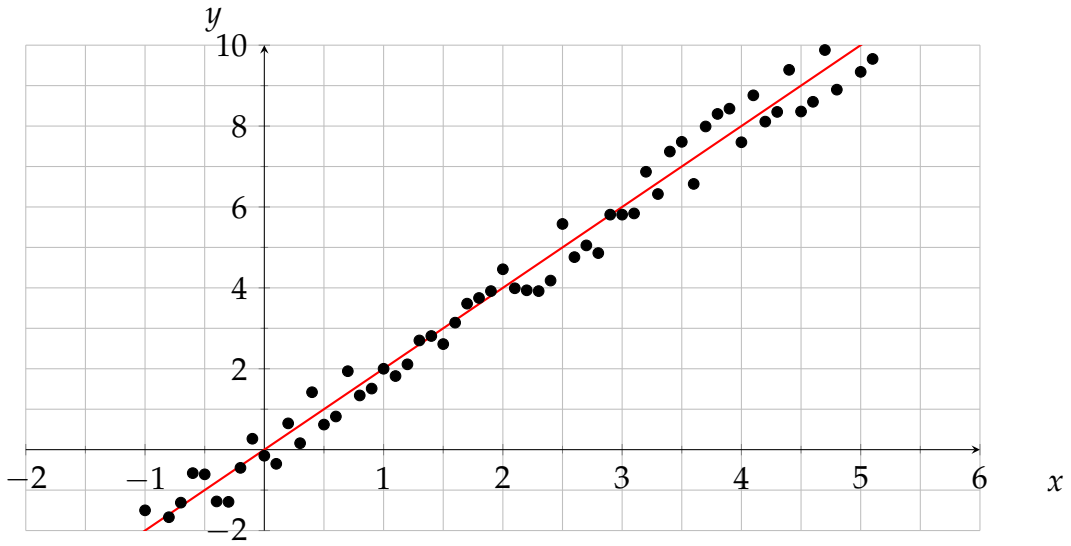

ABBILDUNG 3.12: lineare Regression

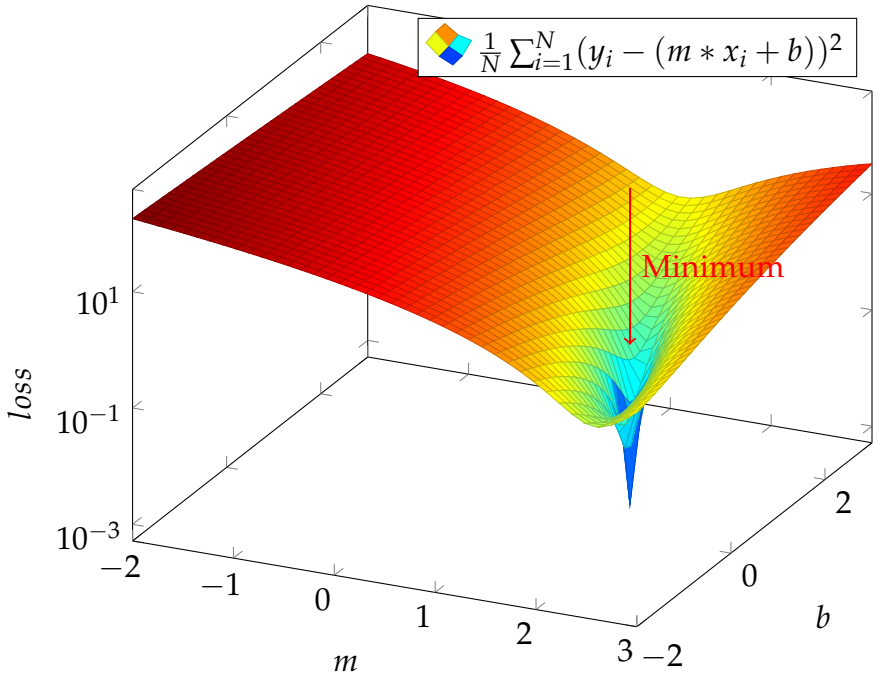

ABBILDUNG 3.13: Fehlerfunktion der linearen Regression

Betrachtet man nun die grafische Darstellung, sieht man, dass der tiefste Punkt bei $m = 2$ und $b = 0$ liegt. Das sind die gleichen Werte wie aus Abb. 3.12.

Beim Bestimmen dieses Punktes, bei dem ein lokales Minimum vorliegt, kommt Gradient Descent zum Einsatz (vgl. Nedrich 2014). Der Gradient eines Punktes zeigt immer in die Richtung mit dem stärksten Anstieg. Möchte man also das nächste lokale Minimum finden, muss man nur in die Richtung des negativen Gradienten gehen und die Schrittgröße verringern, sobald man sich dem Minimum nähert, um das Minimum nicht zu überspringen. Macht man

die Schrittgröße proportional zur Größe des Gradienten, wird die Schrittgröße kleiner, sobald man sich dem lokalen Minimum nähert.

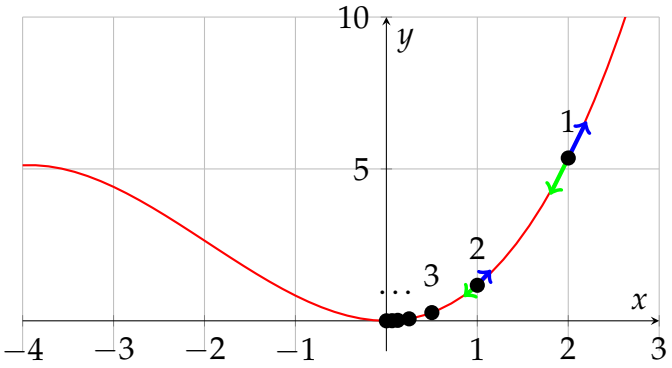

ABBILDUNG 3.14: Gradient Descent an der einfachen Funktion $f(x) = 0,17x^3 + x^2$

Diese Idee lässt sich nun in einen einfachen Algorithmus fassen (vgl. Ng 2018):

while *not converged* **do**

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$$

end

Algorithmus 1 : Pseudocode Gradient Descent, $\alpha \hat{=}$Lernrate

3.2.3 Lernrate

Die Lernrate beeinflusst die Schrittgröße des Gradient Descent Algorithmus (vgl. Ng 2018). Bei Gradient Descent ist die Schrittgröße zusätzlich noch von dem Gradienten abhängig, weshalb es nicht nötig ist, die Lernrate während des Trainings zu verändern: Nähert sich der Algorithmus einem lokalen Minimum, nimmt der Gradient ab, und die Schrittgröße wird kleiner. Trotzdem ist die Wahl einer geeigneten Lernrate wichtig. Bei einer sehr kleinen Lernrate ist die Schrittgröße des Algorithmus sehr klein, und der Algorithmus arbeitet sehr langsam. Ist die Lernrate zu hoch, kann es passieren, dass Gradient Descent das lokale Minimum überspringt, im schlimmsten Fall sogar so weit, dass sich der Algorithmus mit jedem Schritt vom lokalen Minimum weiter entfernt.

3.3 Häufig verwechselte Begriffe für künstliche Intelligenz

Häufig werden die Begriffe maschinelles Lernen, künstliche neuronale Netze und Deep Learning fälschlicherweise synonym verwendet.

3.3.1 künstliche Intelligenz

Künstliche Intelligenz ist schon lange ein Thema in Literatur und Film (vgl. Copeland 2016). Die in Science-Fiction Werken häufig thematisierte Form künstlicher Intelligenz ist universelle künstliche Intelligenz, also künstliche Intelligenz, die ähnlich wie ein Mensch, nicht nur auf eine Aufgabe spezialisiert ist. Diese Form der künstlichen Intelligenz konnte bisher noch nicht entwickelt werden. Bei aktuell eingesetzter künstlicher Intelligenz handelt es sich fast immer um eine KI, die nur eine spezielle Aufgabe erfüllen kann.

Künstliche Intelligenz ist definiert als eine intellektuelle Operation, die simuliert werden kann (vgl. Poole, Mackworth und Goebel 1998). Künstliche Intelligenz ist also ein weites Feld, das aus vielen Unterfeldern besteht, da auch beispielsweise Spracherkennung oder Schachcomputer in die Kategorie fallen.

3.3.2 maschinelles Lernen

Maschinelles Lernen ist das wahrscheinlich vielversprechendste Unterfeld der Forschung an künstlicher Intelligenz (vgl. Renelle 2017). Unter maschinelles Lernen fallen alle Algorithmen, die Daten analysieren, aus den Daten lernen und basierend auf dem Gelernten Vorhersagen treffen (vgl. Copeland 2016). Lange Zeit konnte maschinelles Lernen dem Menschen nicht zur Konkurrenz werden, weil geeignete Algorithmen und die nötige Rechenleistung fehlten. Das änderte sich aber mit der Weiterentwicklung von künstlichen neuronalen Netzen, Deep Learning und der Entwicklung hochperformanter Grafikkarten, die die nötige Rechenleistung bieten.

3.3.3 künstliches neuronales Netz

Künstliche neuronale Netze, die in dieser Arbeit behandelt werden, sind eine Unterkategorie von Machine-Learning Algorithmen (vgl. Renelle 2017). Ein großer Vorteil gegenüber anderen Machine-Learning Algorithmen ist die Fähigkeit von KNNs, selbständig die für eine Verarbeitung eines Datensatzes nötigen Features zu bestimmen. Mit *Features* sind Eigenschaften eines Datensatzes gemeint, auf Basis derer ein KNN Schlüsse auf andere Eigenschaften des Datensatzes ziehen kann. Ein Beispiel für ein solches Feature ist die Quadratmeteranzahl eines Hauses, die ausschlaggebend für den Kaufpreis ist. Auf Basis der gelernten Features kann das KNN später eine Vorhersage treffen. Andere Algorithmen des maschinellen Lernens benötigen in vielen Fällen beispielsweise eine manuell entwickelte Kantenerkennung, die die Außenkanten von Objekten in Fotos erkennt. Ein künstliches neuronales Netz findet hingegen selbstständig die optimale Möglichkeit, Kanten in den Trainingsdaten zu erkennen.

3.3.4 Deep Learning

Deep Learning ist eine spezielle Art künstlicher neuronaler Netze (vgl. Goodfellow, Bengio und Courville 2016). Besteht ein KNN aus vielen Hidden-Layern, handelt es sich um Deep Learning. Deep Learning kann andere Technologien im Bereich der künstlichen Intelligenz häufig bei Weitem schlagen. Vorraussetzung für den erfolgreichen Einsatz von Deep Learning ist allerdings eine große Datenmenge (häufig sind hunderttausende bis mehrere Millionen Trainingsbeispiele nötig). Aufgrund der großen Anzahl an Werten, die bei Deep Learning wegen der großen Neuronenanzahl optimiert werden müssen und durch die Notwendigkeit großer Trainingsdatenbestände, ist Deep Learning erst im Zeitalter hochperformanter Grafikkarten und Big Data möglich geworden.

3.4 Supervised Learning, Unsupervised Learning und Reinforcement Learning

Künstliche neuronale Netze können auf drei grundlegende Arten trainiert werden: Supervised Learning, Unsupervised Learning und Reinforcement Learning. Diese Arbeit beschäftigt sich primär mit Supervised Learning. Ein grober Überblick, auch über die anderen Arten, ist aber zum besseren Verständnis sinnvoll.

3.4.1 Supervised Learning

Supervised Learning ist die Kategorie künstlicher neuronaler Netze, die in dieser Arbeit behandelt werden. Beim Supervised Learning wird dem KNN nach jedem Trainingsschritt Feedback gegeben. Die Trainingsdaten werden in die den Output-Neuronen entsprechenden Kategorien eingeteilt (vgl. Ray 2017). Die Fehlerfunktion wird aus dem Unterschied zwischen der Kategorisierung des KNNs und der Kategorisierung der Trainingsdaten erstellt. Nach der Optimierung der Fehlerfunktion kann die tatsächliche Leistung des KNNs mithilfe von Validierungsdaten ermittelt werden, um das KNN auf overfitting (siehe Abschnitt 4.3.1) zu prüfen. Dieser Lerntyp wird Supervised Learning genannt, weil dem KNN regelmäßig Feedback über die Qualität der Voraussagen gegeben wird. Ein Beispiel für Supervised Learning ist die Voraussage von Daten anhand bestehender Daten. Beispielsweise kann der Preis eines Hauses abhängig von Wohnfläche, Anzahl der Schlafzimmer, Anzahl der Badezimmer, Lage usw. mit einem KNN vorausgesagt werden, wenn eine gewisse Anzahl an Beispielen vorliegt.

3.4.2 Unsupervised Learning

Unsupervised Learning geschieht komplett ohne Vorkategorisierung der Trainingsdaten (vgl. Ray 2017). Das KNN sucht ohne Feedback von außen nach Mustern in den Daten. Unsupervised Learning ist besonders interessant, weil

ein KNN häufig Muster in Daten erkennt, die ein Mensch nicht erkennen würde. Ein weiterer Vorteil ist, dass keine vorkategorisierten Datenbeständen vorhanden sein müssen. Unsupervised Learning wird beispielsweise verwendet, um das Kaufverhalten von Kunden zu analysieren. So können Onlineshops ihren Kunden beispielsweise Produkte anbieten, die in das Muster ihrer zuvor gekauften Produkte passen.

3.4.3 Reinforcement Learning

Wird ein künstliches neuronales Netz mit Reinforcement Learning trainiert, erhält es nach jedem Schritt entweder eine Belohnung oder eine Bestrafung, abhängig davon, ob die ausgeführte Aktion gut oder schlecht war (vgl. Ray 2017). Ein großer Vorteil dieser Methode ist, dass die Qualität jedes einzelnen Schrittes des KNNs nicht bewertet werden muss. Stattdessen wird sich ein KNN so verhalten, dass es über einen längeren Zeitraum mehr Belohnungen als Strafen bekommt. Soll ein KNN beispielsweise darauf trainiert werden Pacman zu spielen, ist eine Bewertung jedes einzelnen Spielzuges nur schwer möglich, da der einzelne Spielzug im Verlauf des Spiels eine ganz andere Bedeutung haben kann. Stattdessen kann man für jeden eingesammelten Punkt eine Belohnung geben und für jeden gegangenen Schritt Strafpunkte vergeben. Somit wird das KNN lernen mit möglichst wenigen Spielzügen möglichst viele Punkte zu sammeln. [1]

[1] Beispiel eines künstlichen neuronalen Netzes, das Pacman durch Reinforcement Learning lernt: https://www.youtube.com/watch?v=t5--kLRI4UE

Kapitel 4

Stärken und Schwächen von maschinellem Lernen gegenüber traditionellen Algorithmen

4.1 Stärken von künstlichen neuronalen Netzen

Künstliche neuronale Netze sind besonders gut in der Mustererkennung (vgl. Ray 2017). In Fällen bei denen es schwer ist konkrete Regeln für beispielsweise eine Klassifizierung festzulegen, übernimmt das Definieren der konkreten Regeln das KNN. Das Problem konkrete Bedingungen für ein Problem zu finden liegt vor allem bei Problemen vor, die viele Variablen haben, die das Endergebnis beeinflussen. Bilderkennung ist ein klassisches Beispiel hierfür: Soll beispielsweise erkannt werden, ob ein Bild einen Hund oder eine Katze zeigt, ist dies durch klassische, nicht lernende Algorithmen schwer, da konkrete Regeln aufgestellt werden müssen, wann ein Bild eine Katze und wann einen Hund zeigt. Da Bilder von Hunden und Katzen meist aus unterschiedlichen Perspektiven, mit unterschiedlichen Lichtbedingungen und Kameras aufgenommen wurden und die gezeigten Tiere unterschiedliche Rassen haben und somit stark unterschiedlich sind, reichen einfache Bedingungen wie beispielsweise die Form der Ohren nicht aus. Außerdem soll der Algorithmus im Optimalfall Rassen erkennen, an die der Entwickler nie gedacht hat. Ein gut trainiertes künstliches neuronales Netz erreicht dieses Ziel, indem es allgemeine Kriterien findet, die eine Unterscheidung von Katze und Hund ermöglicht.

Die Ausführungszeit von künstlichen neuronalen Netzen ist verhältnismäßig gering, was mithilfe einiger Optimierungen den Einsatz von KNNs in Realtime-Anwendungen ermöglicht.

4.2 Schwächen von künstlichen neuronalen Netzen

Ein großer Nachteil künstlicher neuronaler Netze ist, dass es bei Netzen mit vielen Layern und Neuronen schwer wird den Überblick über die Funktionsweise des KNNs zu behalten (vgl. Ray 2017). Es ist somit nicht immer leicht

herauszufinden, warum sich ein KNN nicht so verhält wie es soll. Die Entscheidung zu einer bestimmten Architektur für das KNN kann also nicht nach festen Regeln getroffen werden, sondern erfordert meist das stumpfe Ausprobieren unterschiedlicher Architekturen, basierend auf den Erfahrungen und der Intuition des Entwicklers. Außerdem ist nicht immer sicher, ob man sich auf die Entscheidung eines künstlichen neuronalen Netzes verlassen kann, weil der Entscheidungsprozess schwer nachvollziehbar ist. Besonders wichtig wird das beispielsweise, wenn KNNs verwendet werden, um Patienten auf Krebs zu untersuchen[1].

Das Training künstlicher neuronaler Netze ist sehr rechenaufwendig. Aufgrund der zunehmenden Anzahl zu optimierender Werte bei Deep Learning, ist Deep Learning erst seit der Entwicklung hochperformanter Grafikkarten in der Praxis einsetzbar, da diese aufgrund der guten Parallelisierbarkeit der Trainingsoperationen die Trainingsgeschwindigkeit vervielfachen.

Neben des rechenintensiven Trainings, ist auch die Notwendigkeit großer Trainingsdatenmengen problematisch. Besonders bei Deep Learning sind sehr große Datenmengen erforderlich, da die hohe Entropie des KNNs häufig zu overfitting führt.

4.3 Typische Probleme

4.3.1 Overfitting

Beim Training eines KNNs wird versucht die Funktion zwischen Input- und Outputlayer zu optimieren, also den Loss-Wert zu minimieren (vgl. Goodfellow, Bengio und Courville 2016). Wenn ein KNN ausreichend Neuronen hat, tendiert das KNN dazu die Trainingsdaten einfach „auswendig zu lernen“. Besonders wenn nur wenige Trainingsdaten vorliegen, ist ein „auswendig lernen“ einfacher möglich als allgemeingültige Muster zur Klassifizierung zu finden. Folge dieses Problems ist, dass das KNN zwar eine sehr hohe Genauigkeit bei den Trainingsdaten, aber eine deutlich geringere Genauigkeit bei den Validierungsdaten erzielt.

[1] Interessante Studie zum Thema: Javed Khan u. a. (2001). „Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks“. In: *Nature medicine* 7.6, S. 673. URL: https://www.nature.com/articles/nm0601_673 (besucht am 23. 05. 2018).

Kapitel 5

Entwicklung eines KNNs zur Klassifizierung von Scandaten

5.1 Die Idee

5.1.1 Überblick

Beim Einscannen von großen Dokumentensammlungen kann es immer wieder vorkommen, dass sich leere Seiten zwischen den Blättern befinden. Besonders beim Einscannen von doppelseitigen Blättern werden ungewollt häufig unbeschriebene Seiten gescannt. Das Aussortieren per Hand ist bei großen Dokumentensammlungen sehr aufwendig und soll deshalb automatisiert werden.

5.1.2 Warum ist ein künstliches neuronales Netz sinnvoll?

Solange man Dokumentensammlungen scannt, die auf Blankopapier gedruckt wurden, ist eine Unterscheidung zwischen beschriebenen und unbeschriebenen Seiten nicht schwer: Wenn man die durchschnittle Helligkeit aller Pixel berechnet, lassen sich beschriebene und unbeschriebene Seiten schon sehr gut differenzieren. Alternativ kann man den Benutzer des Programms auch einen Grenzwert einstellen lassen, ab dem Seiten als unbeschrieben gewertet werden sollen. Bei linierten und karierten Seiten ist das nicht möglich. Außerdem muss das Programm auch gelochte Seiten, Artefakte und Verschmutzungen auf dem Papier korrekt erkennen und klassifizieren können. Da künstliche neuronale Netze sehr gut in der Klassifikation von Bildern sind (vgl. Ray 2017), sind künstliche neuronale Netze auch sehr gut in der Klassifikation von leeren und beschriebenen Seiten, so die Hypothese.

5.2 Erste Versuche mit der MNIST-Datenbank

5.2.1 Entscheidung zur MNIST-Datenbank

Vor der Klassifikation von eigenen Daten ist es sinnvoll, sich an einem Standarddatensatz zu bedienen, um erste Vergleichsergebnisse mit der verwendeten Software zu bekommen. Besonders gut geeignet ist hier die MNIST-Datenbank. Die Datenbank besteht aus 60 000 Trainingsbildern und 10 000

Validierungsbildern handschriftlich geschriebener Ziffern (vgl. LeCun, Cortes und Burges 2018). Die MNIST-Datenbank ist ein kleinerer Auszug von Bilder aus einer Datenbank vom National Institute of Standards and Technology[1]. Für erste Tests eignet sich die Datenbank sehr gut, weil die Daten bereits vorverarbeitet sind, d.h. alle Zahlen sind zentriert, die Bilder gleich groß usw.

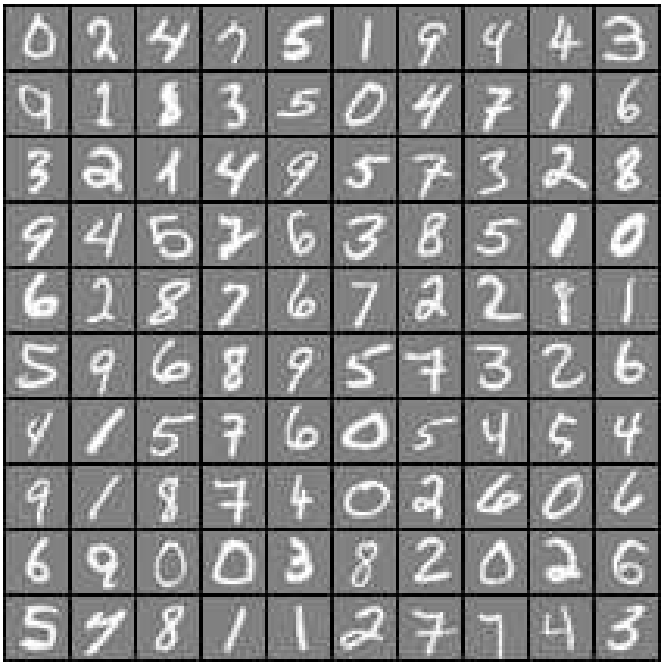

ABBILDUNG 5.1: Beispieldatensätze der MNIST-Datenbank

5.2.2 TensorFlow

Entscheidung zur Nutzung eines Frameworks

Aufgrund der Komplexität von maschinellem Lernen und der nötigen Optimierung der Trainingsalgorithmen, ist es sinnvoll fertige Frameworks zur Umsetzung zu verwenden (vgl. Ray 2017). Zwar ist eine grundlegende Implementierung ohne fertige Frameworks noch recht einfach möglich, die Auslagerung und Optimierung der rechenintensiven Operationen auf die GPU ist jedoch aufwendig. Außerdem erlaubt einem die Nutzung eines fertigen Frameworks schnelle Experimente mit anderen Trainingsalgorithmen.

Entscheidung zur Nutzung von TensorFlow

TensorFlow ist ein Machine Learning Framework von Google (vgl. TensorFlow 2017a). TensorFlow bringt APIs für Python, C++, Java und Go mit, wird

[1] https://www.nist.gov/sites/default/files/documents/srd/nistsd19.pdf

aber meist in Verbindung mit Python benutzt (vgl. TensorFlow 2018a). Da Python eine Interpretersprache ist, ist es sinnvoll, die rechenintensiven Operationen in einer effizienteren Sprache durchzuführen (vgl. TensorFlow 2018b). TensorFlow verwendet hier vor allem C++, CUDA und cuDNN und lagert die Operationen optional auf die GPU aus. In Python wird nur der grundlegende Programmablauf entwickelt, der keine rechenintensiven Operationen beinhaltet. Das TensorFlow-Python-Modul bietet ein Interface zwischen dem TensorFlow-Backend und Python. Neben TensorFlow gibt es auch viele andere Machine Learning Frameworks für Python, wie beispielsweise Scikit-Learn, Brainstorm oder Microsofts CNTK. Die Wahl fiel auf TensorFlow aufgrund der guten Dokumentation, der großen Community um TensorFlow und die Flexibilität und Performance, die TensorFlow bietet.

5.2.3 KNN zur MNIST-Klassifizierung mit TensorFlow

Um schnell erste Ergebnisse zu erhalten, blieb die Netzarchitektur zur Klassifizierung der MNIST-Datenbank mit TensorFlow anfangs relativ klein. Da die ersten Tests primär dazu dienen, die korrekte Funktionsweise des Testsystem sicherzustellen, ist es sinnvoll nach Anleitungen der TensorFlow Dokumentation vorzugehen (vgl. TensorFlow 2017b):

Zuerst werden die nötigen Module importiert und die MNIST-Datenbank geladen. Anschließend wird eine interaktive TensorFlow-Session gestartet. In der Session wird später der Dataflow-Graph ausgeführt. Sie dient als Interface zwischen dem Tensorflow-Backend und Python.

```
import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
print()
sess = tf.InteractiveSession()
```

LISTING 5.1: Nötige Module importieren; TensorFlow Session erstellen

Im nächsten Schritt werden die Platzhalter erstellt, die später mit den Labeln und Eingabedaten der MNIST-Datenbank gefüllt werden. Außerdem werden die zu trainierenden Parameter deklariert:

```
x = tf.placeholder(tf.float32, shape=[None, 784])
y = tf.placeholder(tf.float32, shape=[None, 10])

w = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
```

LISTING 5.2: Platzhalter für Trainingsdaten und Label erstellen; zu trainierende Parameter erstellen

Im nächsten Schritt werden alle Variablen mit den Startwerten initialisiert. Diese Operation wird durch `tf.global_variables_initializer()` parallelisiert auf der Grafikkarte für alle Neuronen gleichzeitig ausgeführt.

```
sess.run(tf.global_variables_initializer())
```

LISTING 5.3: Zu trainierende Parameter initialisieren

Anschließend wird der TensorFlow-Graph definiert. Hier werden die Trainingsdaten durch eine Matrixmultiplikation mit den Gewichten multipliziert und anschließend der Bias-Wert hinzugerechnet (siehe Abb. 5.2).

```
y_ = tf.matmul(x,w) + b
```

LISTING 5.4: Erste Node definieren

Im nächsten Schritt wird die Loss-Function definiert. Hier wird Cross-Entropy mit den Labeln der MNIST-Datenbank und dem Output des Netzes verwendet. Diese Loss-Function wird verwendet, um im nächsten Schritt festzulegen, was bei jedem Trainingsschritt gemacht werden soll. Gradient Descent versucht bei jedem Trainingsschritt die Ausgabe der Loss-Function zu minimieren:

```
cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(labels=y, logits=y_)
)

train_step = tf.train.GradientDescentOptimizer(0.5).minimize(
    cross_entropy)
```

LISTING 5.5: Loss-Function und Optimierungsalgorithmus definieren

Anschließend wird das KNN mit 1000 Trainingsschritten trainiert. Vor jedem Trainingsschritt wird ein neuer Batch Trainingsdaten geladen.

```
for _ in range (1000):
    batch = mnist.train.next_batch(100)
    train_step.run(feed_dict={x: batch[0], y: batch[1]})
```

LISTING 5.6: KNN trainieren

Im letzten Schritt wird noch die Performance des KNNs gemessen und anschließend ausgegeben.

```
correct_prediction = tf.equal(tf.argmax(y_, 1), tf.argmax(y, 1))

accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

print(accuracy.eval(feed_dict={x:mnist.test.images, y:mnist.test.
    labels}))
```

LISTING 5.7: Accuracy berechnen und Ausgeben

5.2.4 Ergebnisse

Das KNN erreicht eine Genauigkeit von ca. 92%. Der Wert zeigt, dass das Testsystem funktioniert. Der Wert ist aber noch weit entfernt vom State of the Art. Andere KNNs erreichen Werte von bis zu 99, 8% (vgl. Benenson 2016). Die geringe Performance ist darauf zurückzuführen, dass das verwendete KNN sehr klein ist und viele aktuelle Technologien nicht einsetzt (vgl. TensorFlow 2017b).

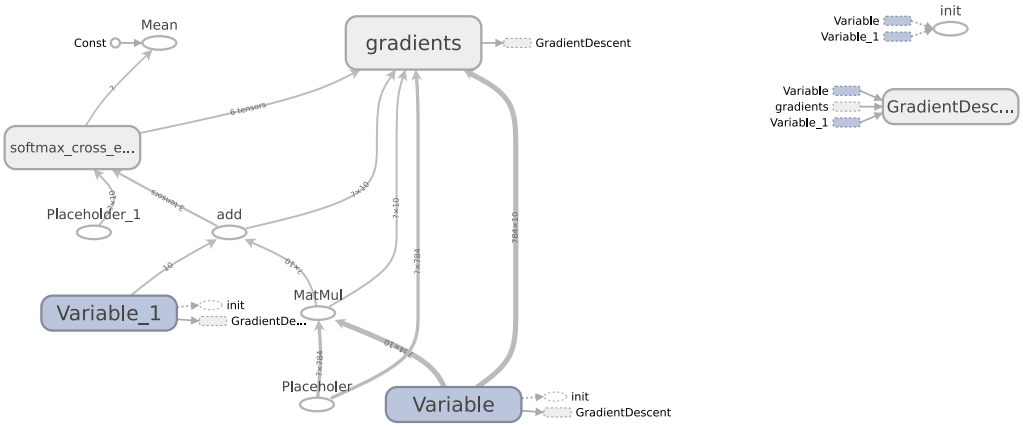

ABBILDUNG 5.2: TensorFlow-Graph des KNNs zur Klassifizierung der MNIST-Datenbank | Erstellt mit TensorBoard

5.3 Convolutional Neural Network mit Keras - Versuch 2

5.3.1 Warum Keras?

Keras ist eine High-Level-API. Keras unterstützt als Backend unterschiedliche Machine Learning-Frameworks, unter anderem auch TensorFlow. Aufgrund der Möglichkeit sehr leicht mit Netzarchitektur, Aktivierungsfunktionen usw. zu experimentieren, ist Keras gut für Prototypen geeignet. Keras schafft einen weiteren Abstraktionslayer zwischen den Machine Learning Backends, wie z.B. TensorFlow, und dem Benutzer.

Diese Abstraktion erlaubt es, mit nur wenigen Zeilen ein leistungsfähiges KNN zur Klassifizierung von Bildern zu entwickeln, das erste Ergebnisse auf die Problemstellung liefert. Da das KNN zu einem TensorFlow-Graph kompiliert wird, bringt Keras trotzdem die Flexibilität von TensorFlow mit.

5.3.2 Netzarchitektur

Aufgrund der hohen Erfolge von Deep-Learning in ähnlichen Versuchen ist eine Multilayer-Netzarchitektur sinnvoll. Um besonders gut Muster in den Bildern erkennen zu können, bietet sich die Verwendung eines Convolutional Neural Networks, kurz CNN an.

Der erste Teil des CNNs besteht aus drei Convolutional-Layern und jeweils drei dazugehörigen MaxPooling- und Activation-Layern. Ein typisches Convolutional Neural Network besteht aus 4 grundlegenden Schritten:

1. Convolution
2. Pooling
3. Flattening

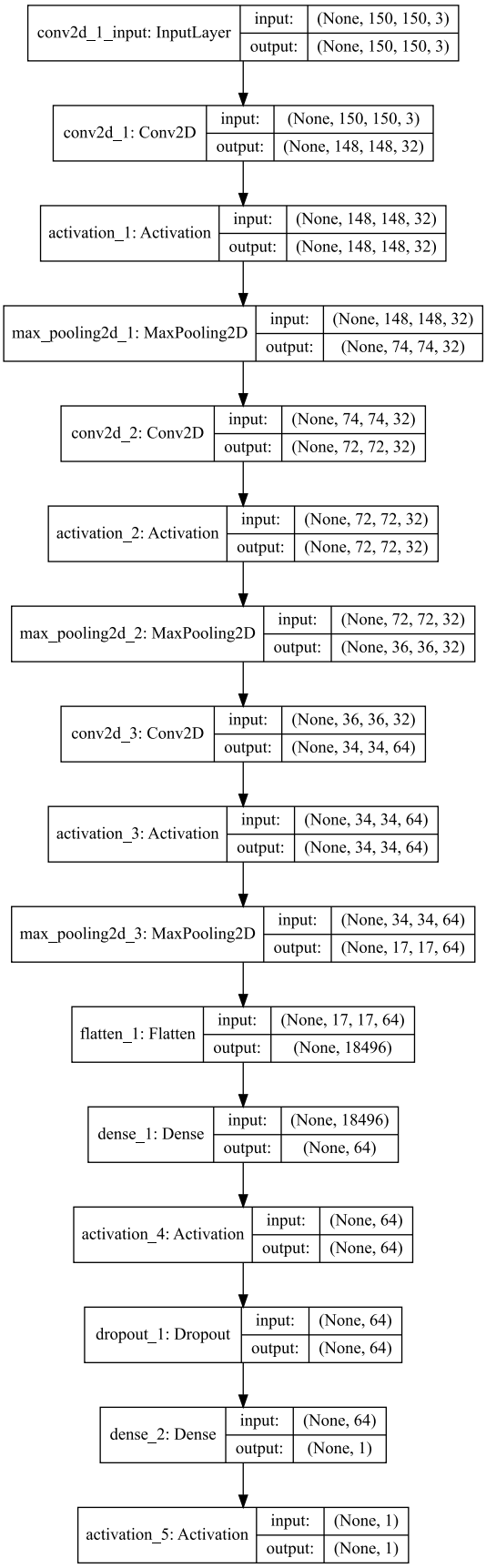

ABBILDUNG 5.3: Netzarchitektur von Versuch 2

4. Full connection

Convolutional Layer

Convolutional Layer sind Layer, die besonders gut Muster in Bildern erkennen können (vgl. Goodfellow, Bengio und Courville 2016). In einem Convolutional Layer werden unterschiedliche Filter über die Pixelmatrix des Bildes *geschoben*, sodass für jeden Filter eine neue Matrix entsteht, die aufzeigt an welchen Stellen der Filter besonders gut auf das Bild passt. Entsprechend feuern bestimmte Neuronen besonders stark, wenn ein Filter an einer Stelle des Bildes ein Muster erkennt.

Im Beispiel (siehe Abb. 5.4) wird eine 5×4 große Eingabematrix mit einem Beispielfilter zur Eckenerkennung in eine 3×2 große Ausgabematrix transformiert. Dazu wird das Skalarprodukt einer $n \times m$ großen Filtermatrix (hier 3×3) und einer ebenfalls $n \times m$ großen Teilmatrix der Eingabematrix gebildet. Das Ergebnis der Operation wird zum ersten Element der Ausgabematrix. Anschließend wird der Filter verschoben, also eine andere Teilmatrix der Eingabematrix verwendet und die Operation wiederholt. Dadurch wird jeder Filterposition ein Wert zugewiesen, der angibt wie deckungsgleich der Filter mit der entsprechenden Position auf der Eingabematrix ist.

Im Beispiel 1 liegt eine Ecke vor, was zu einem hohen Wert in der Ausgabematrix führt. In Beispiel 2 liegt keine Ecke vor, der Wert der Ausgabematrix ist entsprechend gering. In der Ausgabematrix lässt sich erkennen, dass oben links im Eingabebild eine Ecke vorliegt. Diese Information kann in weiteren Convolutional Layern zur Erkennung komplexerer Muster genutzt werden.

Die Filter werden nicht wie in dem Beispiel händisch gewählt, sondern während dem Trainingsprozess automatisch durch das CNN gebildet bzw. verändert. Visualisiert man die Filter nach dem Trainingsprozess, so sieht man welche Muster das CNN erkannt und gelernt hat.

Activation Layer

Ein Activation Layer ist nur eine andere Darstellungsweise oder Implementierung der Aktivierungsfunktion. In diesem Beispiel wurde ReLu als Aktivierungsfunktion eingesetzt.

Max Pooling Layer

Der Max Pooling Layer nach einem Convolutional Layer verwirft einige der Informationen, sodass eine kleinere Ausgabematrix entsteht. Dazu werden sogenannte *Pools* gebildet, also $n \times m$ große Teilmatrizen der Eingabematrix, aus denen jeweils das Element mit dem maximalen Wert zu einem Element der Ausgabematrix wird (siehe Abb. 5.5).

Flattening

Vor dem letzten Schritt muss die Ausgabe des letzten Max Pooling Layers wieder in einen Vektor ausgerollt werden. Der letzte Teil eines CNNs ist

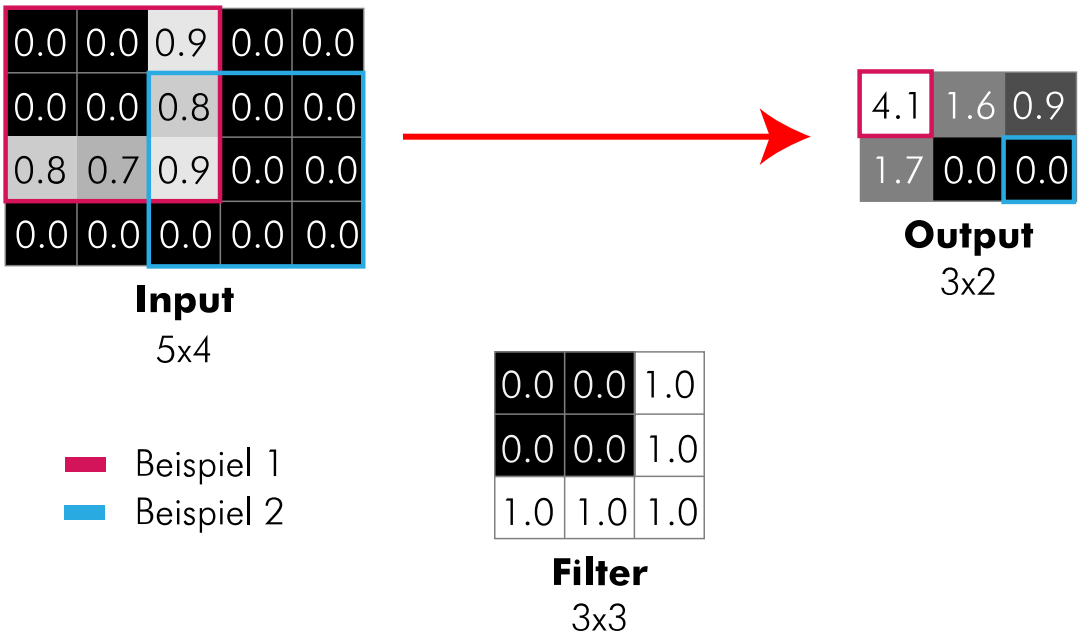

ABBILDUNG 5.4: Visualisierung Convolutional Layer

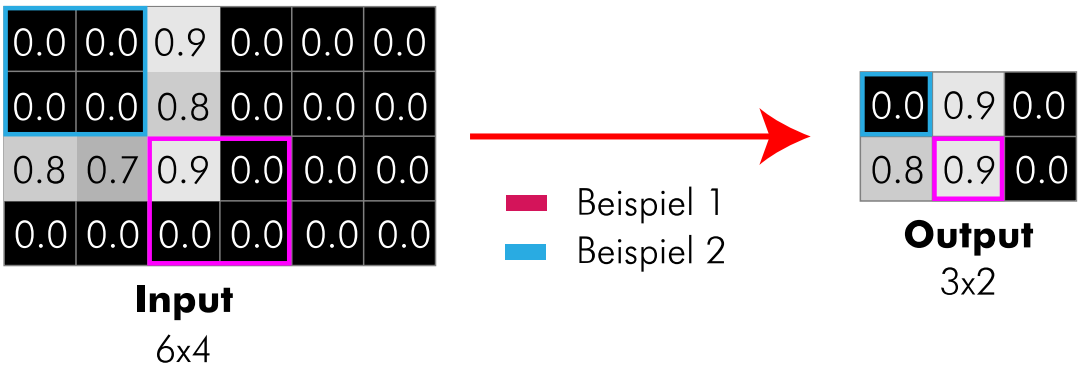

ABBILDUNG 5.5: Visualisierung MaxPooling-Layer

ein normales MLP, es benötigt als Eingabe also wieder einen Feature-Vektor. Dieser wird durch den Flatten Layer erzeugt.

Dense Layer

Ein Dense Layer ist ein typischer Layer eines MLPs. Die Dense Layer bilden gemeinsam mit den Dropout Layern den letzten Teil eines CNNs.

Dropout Layer

In einem Dropout Layer werden zufällige Neuronen während dem Trainingsprozess kurzzeitig ausgeschaltet, sodass deren Ergebnis in nachfolgenden Berechnungen nicht berücksichtigt wird. Das ist eine effektive Methode um Overfitting zu vermeiden. Das KNN wird dazu gezwungen Muster in den Eingabedaten zu erkennen und somit zu verallgemeinern.

5.3.3 Vorverarbeitung der Daten

Um dafür zu sorgen, dass das KNN möglichst gut verallgemeinert, sind sehr viele Trainingsdaten nötig. Besonders bei Deep Learning ist eine große Datenmenge erforderlich, um Overfitting zu vermeiden (vgl. Chollet 2016). Aufgrund des großen Aufwands Scandaten per Hand zu sortieren oder neue Scandaten zu produzieren, ist die für das KNN zur Verfügung stehende Datengrundlage sehr klein. Praktischerweise bringt Keras für dieses Problem das Modul `ImageDataGenerator` mit, das Variationen der Trainingsdaten generiert, indem das Quellbild skaliert, rotiert, verschoben usw. wird.

```
2 class ImagePreProcessor:

4         def generate_training_data(self, input_image, output_dir,
    variations):
            print("Started generating " + str(variations) + "
    variations for " + input_image)
6
            datagen = ImageDataGenerator(
8                   rotation_range=40,
                    width_shift_range=0.2,
10                  height_shift_range=0.2,
                    shear_range=0.2,
12                  zoom_range=0.2,
                    horizontal_flip=True,
14                  rescale=1./255,
                    fill_mode='nearest')
16
            img = load_img(path=input_image, target_size=cfg.
    img_resolution)
18          x = img_to_array(img)
            x = x.reshape((1,) + x.shape)
20
            i = 0
22          for batch in datagen.flow(x, batch_size=1,
```

Trainingsbilder	90
Validierungsbilder	40
Variationen	2
Epochen	50
Auflösung	150×150
Accuracy	99,8%
Probleme	linierte Seiten, Seiten mit wenig Text

TABELLE 5.1: Daten Versuch 2, Version 1

```
                                        save_to_dir=output_dir,
    save_prefix='', save_format='jpeg'):
24                  i += 1
                    if i > variations:
26                          break
```

LISTING 5.8: Variationen der Trainingsdaten generieren

Die Bilder werden schon bei der Trainingsdatengenerierung auf die benötigten Abmessungen skaliert. So können beim späteren Training lange Ladezeiten verhindert werden. Ausgabe des Generators sind 150x150 Pixel große Variationen der Quellbilder im JPEG-Format (siehe Abb. 5.6).

ABBILDUNG 5.6: Ausgabe des Datengenerators

5.3.4 Ergebnisse - Version 1

Der erste Versuch mit diesem Netz, wurde mit insgesamt 130 Bildern (90 Trainingsbilder, 40 Validierungsbilder) durchgeführt (siehe Tabelle 5.1). Nach 50 Epochen konnte eine *Accuracy* von 99,8% erreicht werden. Aufgrund der geringen Anzahl der Validierungsdaten ist dieses Ergebnis allerings nur bedingt repräsentiv. Der Test mit großen PDF-Dateien lieferte jedoch schon sehr gute Ergebnisse, wenn die Seiten komplett unbedruckt waren. Probleme traten vor allem bei linierten Seiten und Seiten mit sehr wenig Text auf. Letztere wurden häufig als unbedruckt klassifiziert.

Da die Trainingsdaten in Version 1 hauptsächlich aus komplett bedruckten bzw. komplett unbedruckten Seiten bestanden, hat das KNN gelernt,

Trainingsbilder	296
Validierungsbilder	188
Variationen	2
Epochen	50
Auflösung	150×150
Accuracy	90%
Probleme	niedrige Accuracy

TABELLE 5.2: Daten Versuch 2, Version 2

bedruckte Seiten anhand der vielen Schwarzwerte zu erkennen. Eine zumindest grundlegende Mustererkennung war aber bereits erkennbar: Komplett schwarze oder unbedruckte farbige Seiten wurden als unbedruckt erkannt.

5.3.5 Ergebnisse - Version 2

In Version 2 wurde ausschließlich die Trainingsdatenbank erweitert: Um für eine bessere Klassifizierung von leicht bedruckten und linierten Blättern zu sorgen, brauchte das KNN mehr Beispiele dieser Seiten.

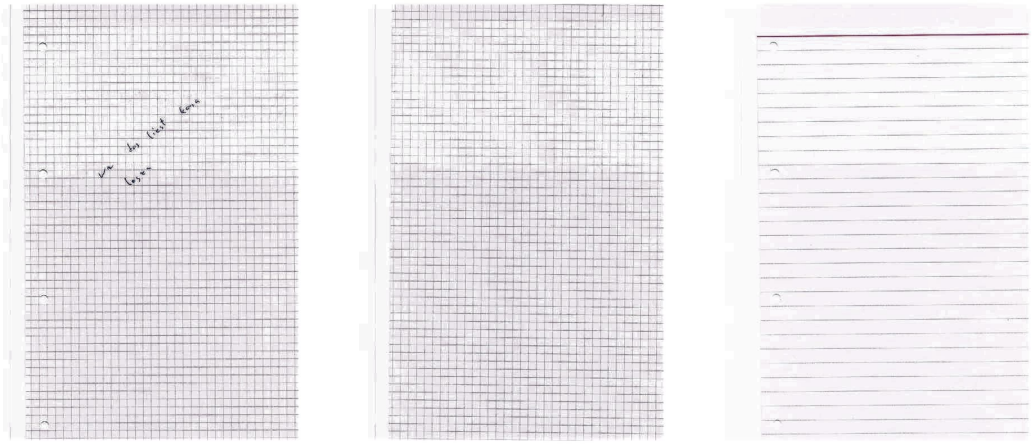

ABBILDUNG 5.7: Beispiele hinzugefügter Datensätze in Version 2 (Kontrast zur besseren Darstellung erhöht)

Die Accuracy ist durch das neue Trainingsmaterial stark gesunken, was zu erwarten war: Die neuen Daten waren einerseits deutlich schwieriger zu klassifizieren, die größere Datenmenge verhinderte aber auch overfitting bei ähnlichen Datensätzen.

5.3.6 Ergebnisse - Version 3

Da zwei Variationen des Potential das `ImageDataGenerator`-Moduls von Keras bei weitem nicht ausschöpfen, war ein Test mit mehr Variationen sinnvoll (siehe Tabelle 5.3). Da mehr Varianten aber zu keinen nennenswerten Vorteilen, aber deutlich höheren Trainingszeiten führten, schien ein Erhöhen der Variationsanzahl bei gegebenem Datensatz kontraproduktiv.

Trainingsbilder	740
Validierungsbilder	470
Variationen	5
Epochen	50
Auflösung	150×150
Accuracy	90%
Probleme	niedrige Accuracy

TABELLE 5.3: Daten Versuch 2, Version 3

	50×50	150×150	300×300	400×400
Accuracy	81,9%	92,7%	89,5%	93%
Accuracy Validierungsdaten	80,3%	90,1%	91,9%	87,8%
Trainingsdauer	30,35s	236,93s	895,5s	1577,04s

TABELLE 5.4: Daten Versuch 2, Version 4 | Vergleich verschiedener Auflösungen über 150 Epochen

5.3.7 Ergebnisse - Version 4

Durch das Skalieren der Trainingsbilder auf 150×150 Pixel gehen viele Informationen verloren. Ein Test mit einer höheren Auflösung ist also sicherlich sinnvoll, um auszuschließen, dass das KNN zu wenig Informationen aus den Trainingsbildern entnehmen kann.

Wie aus den Trainingsergebnissen erkennbar ist, hat die Bildauflösung einen Einfluss auf die Genauigkeit (siehe Tabelle 5.4). Ab einer gewissen Bildauflösung steigt die Genauigkeit allerdings nicht mehr nennenswert an. Die Trainingsdauer erhöht sich aber durch die größere Anzahl der Gewichte und die höheren Bildladezeiten deutlich. Somit ist eine Auflösung von 150×150 Pixel für weitere Tests ein sinnvoller Wert.

Neben der optimalen Auflösung ist auch die optimale Anzahl der Epochen wichtig für das KNN. Wird das KNN in zu vielen Epochen trainiert, besteht das Risiko von Overfitting. Bei zu wenigen Epochen wird nicht die optimale Leistungsfähigkeit des KNNs erreicht (siehe Abb. 5.9). Die optimale Anzahl der Epochen kann während des Trainings automatisch ermittelt werden, indem das Training bei sehr geringen Veränderungen der Validation-Accuracy abgebrochen wird. Da die Ergebnisse aber leicht vergleichbar bleiben sollen, ist eine feste Epochenzahl während der Entwicklung sinnvoll.

Der Trainingsverlauf über 150 Epochen ist bei allen Auflösungen relativ ähnlich (siehe Abb. 5.8, Abb. 5.9, Abb. 5.10, Abb. 5.11). Nach der ersten Epoche erreichen alle Netze eine Accuracy von ca. 70%. Der Anstieg der Accuracy verläuft ab diesem Punkt aber deutlich stärker bei den Netzen, die Bilder der Auflösungen $150px$ und größer verarbeiten.

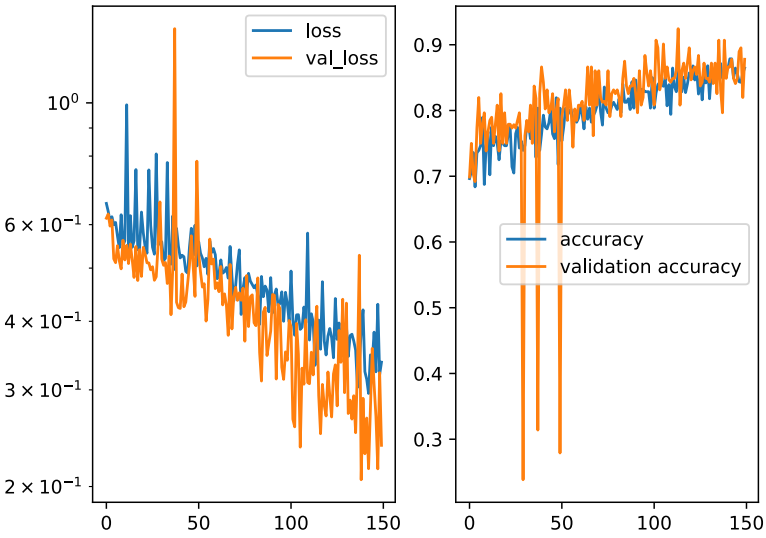

ABBILDUNG 5.8: Trainingsverlauf für eine Auflösung von 50 × 50 Pixel über 150 Epochen

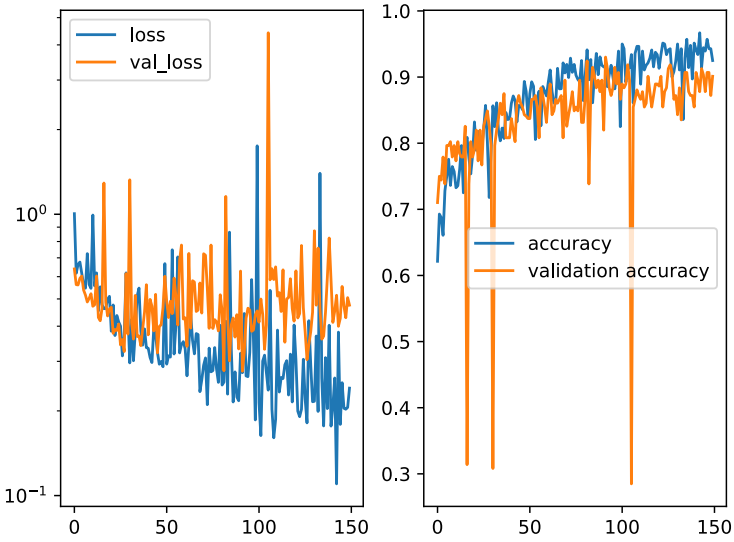

ABBILDUNG 5.9: Trainingsverlauf für eine Auflösung von 150 × 150 Pixel über 150 Epochen

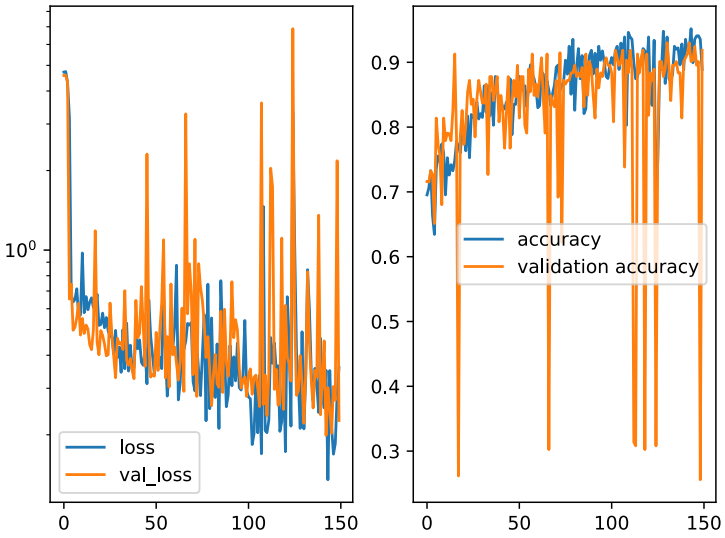

ABBILDUNG 5.10: Trainingsverlauf für eine Auflösung von 300×300 Pixel über 150 Epochen

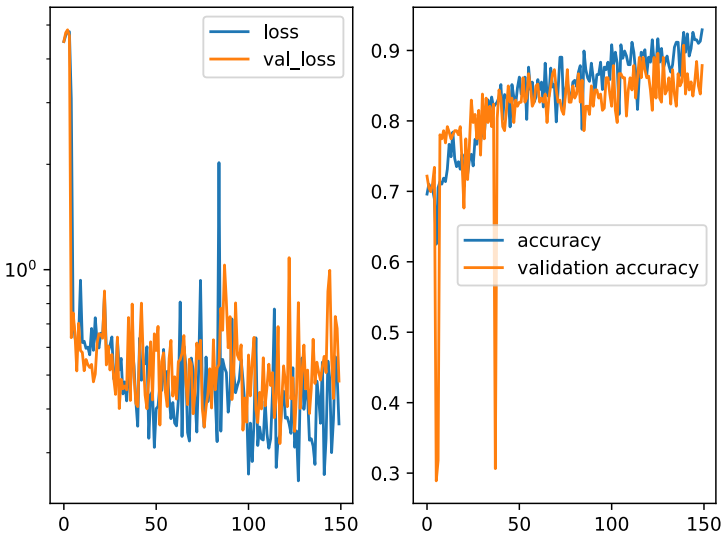

ABBILDUNG 5.11: Trainingsverlauf für eine Auflösung von 400×400 Pixel über 150 Epochen

Analyse einzelner Layer

Um das Verhalten des neuronalen Netzes besser zu verstehen, ist es sinnvoll, einzelne Layer und Neuronen zu visualisieren. Ein Tool, das diese Aufgabe erleichtert, ist das `Keras Visualization Toolkit`, kurz `Keras-Vis` (vgl. Kotikalapudi und Mitwirkende 2017).

```
model, graph = netloader.init()


# index of layer to analyze
layer_idx = -1

# Swap softmax with linear
model.layers[layer_idx].activation = activations.linear
model = utils.apply_modifications(model)

# matplotlib settings
plt.rcParams['figure.dpi'] = 240
plt.interactive(False)

images = []
for path in (['./sourceData/printed/1.png', './sourceData/printed/2.png', './sourceData/printed/3.png']):
    images.append(utils.load_img(path, target_size=cfg.img_resolution))

for modifier in ['relu']:
    f, ax = plt.subplots(1, 3)
    f.dpi = 160
    plt.suptitle(modifier)
    for i, img in enumerate(images):
        img = np.reshape(img, (1, cfg.img_resolution[0], cfg.img_resolution[1], 1))
        img = np.repeat(img, 3, 3)
        grads = visualize_saliency(model, layer_idx, filter_indices=0,
                                   seed_input=img, backprop_modifier=modifier)
        ax[i].imshow(grads, cmap="jet")

plt.show()
# plt.savefig(str(uuid.uuid4()) + '.png', bbox_inches='tight', format='png', dpi=240)
```

LISTING 5.9: Visualisieren des letzten Layers mit Keras-Vis

Um zu untersuchen welche Neuronen eines KNNs besonders großen Einfluss auf das Endergebnis haben, kann eine sogenannte Attention-Map generiert werden (vgl. Kotikalapudi und Mitwirkende 2017). Hellere Stellen der Attention-Map stehen für Neuronen, die einen größeren Einfluss auf das Endergebnis haben, als dunklere Stellen. Rote Stellen haben besonders großen Einfluss. Die Attention-Map von drei Beispielscans zeigt, dass der Text die

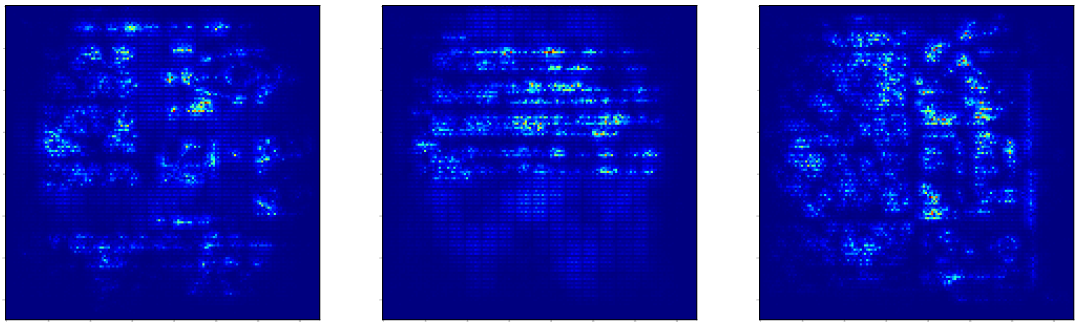

ABBILDUNG 5.12: Attention-Map des letzten Layers

stärksten Auswirkungen auf das Endergebnis hat. Die Linien werden größtenteils ignoriert. Das zeigt, dass das KNN grundlegende Muster in den Daten erkennt und Text und Linien unterscheiden kann.

5.3.8 Ergebnisse - Version 5

Nach genauerer Untersuchung der Scandaten wurde schnell klar, dass zumindest ein Teil der schlechten Accuracy auf schlechte Einstellungen für den `ImageDataGenerator` zurückzuführen ist: Einige Transformationen, die auf die Quelldaten angewendet wurden, sorgten dafür, dass eine Unterscheidung zwischen bedruckten und unbedruckten Seiten teilweise unmöglich wird (siehe Abb. 5.13). Im Beispiel ist auf dem linken Bild nicht erkennbar, ob es sich um Text oder um ein kariertes Blatt handelt. Auch der Mensch kann nicht mehr zwischen bedruckt und unbedruckt entscheiden, was ein deutliches Zeichen dafür ist, dass zu viele Informationen verworfen wurden. Deshalb wurde die Vorverarbeitung der Daten (siehe Abschnitt 5.3.3) wieder auf eine Normalisierung der Daten reduziert:

```
datagen = ImageDataGenerator(
    rescale=1./255)
```

LISTING 5.10: Die Daten werden in der Vorverarbeitung nur noch normalisiert

Trainingsbilder	148
Validierungsbilder	94
Variationen	1
Epochen	50
Auflösung	150×150
Accuracy	95%
Probleme	niedrige Accuracy

TABELLE 5.5: Daten Versuch 2, Version 5

Das Reduzieren der Vorverarbeitung auf eine Normalisierung der Trainingsdaten hat zu einem deutlichen Anstieg der Accuracy auf 95% geführt (siehe

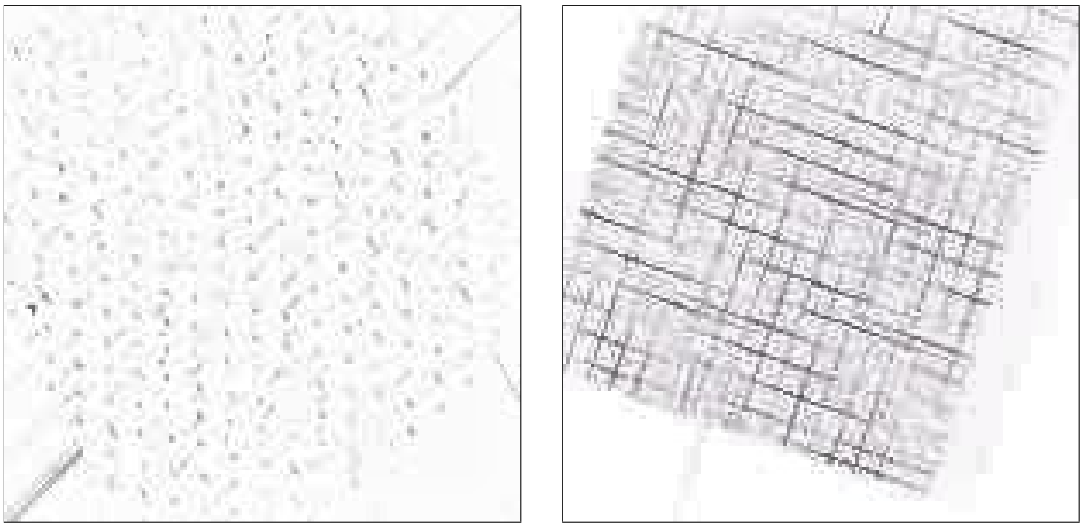

ABBILDUNG 5.13: Beispiel schlechter Trainingsdaten: links bedruckt, rechts unbedruckt (Kontrast zur besseren Darstellung erhöht)

Tabelle 5.5). Das zeigt, dass die Vorverarbeitung kontraproduktiv war. Vermutlich liegt das daran, dass die Trainingsdaten zu stark verändert wurden. Da in Version 6 eine Vorverarbeitung zwecklos wird, ist eine weitere Anpassung der Vorverarbeitung auch nicht nötig.

5.3.9 Ergebnisse - Version 6

Eine Verbesserung der Trainingsdaten konnte in vorherigen Versionen bereits zu einer höheren Genauigkeit führen. Das größte Problem ist bis Version 5, dass zu wenige Trainingsdaten vorliegen. Eine Möglichkeit dieses Problem zu lösen, wäre mehr sortierte Blätter zu scannen, um die Trainingsdatenbank zu vergrößern. Um große Erfolge mit Deep Learning zu erzielen, sind allerdings tausende vorkategorisierte Beispiele erforderlich. Diese Datenmenge kann während der Nutzung des Tools gesammelt werden, indem man Benutzern die Möglichkeit gibt, ihre Daten nach der Klassifizierung zur Verbesserung des Systems einzureichen. Nennenswerte Datenmengen kommen allerdings erst zustande, wenn das Tool sich etabliert hat und frequentiert benutzt wird. In der Übergangszeit muss eine Lösung gefunden werden, die die Genauigkeit des KNNs soweit erhöht, dass das Tool in der Praxis einen nennenswerten Vorteil gegenüber dem händischen Sortieren der Seiten bietet.

Als alternative Möglichkeit an viele Trainingsdaten zu kommen, kann man auch ein Skript schreiben, das zufällige Trainingsdaten generiert. Die `Python Imaging Library`, kurz `PIL` bringt alle nötigen Tools dafür mit.

Zuerst werden die nötigen Module importiert:

```
1 from PIL import Image, ImageFilter, ImageDraw, ImageFont
  import random
3 from pathlib import Path
  import string
5 import config as cfg
  import os
```

```
import sys
import math
```

LISTING 5.11: Nötige Module importieren

Bevor neue Trainingsdaten generiert werden, sollen die alten Trainingsdaten gelöscht werden. Zum Leeren von Ordnern auf dem Dateisystem kann eine einfache Methode angelegt werden.

```
def clear_folder(folder):
    for the_file in os.listdir(folder):
        file_path = os.path.join(folder, the_file)
        try:
            if os.path.isfile(file_path):
                os.unlink(file_path)
        except Exception as e:
            print(e)
```

LISTING 5.12: clear_folder()-Methode

Das Generieren der Bilder wird ebenfalls in eine eigene Methode ausgelagert. Ob das Bild beschrieben sein und welche Maße es haben soll, wird per Parameter übergeben.

```
def generateImage(imgWidth, imgHeight, empty=True):
```

LISTING 5.13: generateImage()-Methode

In der Methode zum Generieren des Bildes wird zuerst ein leeres Bild mit einer zufälligen Hintergrundfarbe erstellt.

```
def generateImage(imgWidth, imgHeight, empty=True):
    bgColor = (random.randrange(200, 255),) * 3
    img = Image.new("RGB", (imgWidth,imgHeight), bgColor)
```

LISTING 5.14: Hintergrund erstellen

Anschließend werden die Variablen der restlichen Eigenschaften der Seite mit Zufallswerten initialisiert.

```
    verticalLineSpacing = random.randrange(5, 15)
    horizontalLineSpacing = random.randrange(5, 15)
    papertype = random.randrange(0, 4)
    lineColor = (random.randrange(0, 50), random.randrange(0, 50),
    random.randrange(0, 50))
    textAmount = random.randrange(1, 10)
    textLengthRange = (2, 200)
    pageMargin = random.randrange(40, 80)
    lineShift = random.randrange(0, 20)
    generateHoles = bool(random.getrandbits(1))
```

LISTING 5.15: Variablen mit Zufallswerten initialisieren

Um das Blatt zu linieren wird jeder Pixel betrachtet und eingefärbt, falls die Koordinate des Pixels durch den vertikalen bzw. horizontalen Linienabstand teilbar ist. Außerdem wird geprüft, ob der Papiertyp eine entsprechende Linie zulässt.

```
    for x in range(imgWidth):
        for y in range(imgHeight):
            if (((y + lineShift) % verticalLineSpacing == 0 and
papertype != 0) or ((x + lineShift) % horizontalLineSpacing == 0
 and papertype == 2)) or (papertype == 3 and (x == pageMargin or
 x == imgWidth-pageMargin)):
                img.putpixel((x,y),lineColor)
```

LISTING 5.16: Linien zeichnen

Anschließend wird, falls das Blatt nicht leer sein soll, zufälliger Text auf das Blatt geschrieben. Position und Schriftart wird dabei zufällig gewählt.

```
    draw = ImageDraw.Draw(img)
    if (not(empty)):
        for i in range(textAmount):
            text = ''.join(random.choice(string.ascii_uppercase +
string.digits) for _ in range(random.randrange(textLengthRange
[0], textLengthRange[1])))
            w, h = draw.textsize(text)
            draw.text((random.randrange(0, imgWidth)-w/2, random.
randrange(0, imgHeight)-h/2), text, font=random.choice(fonts),
                      fill=(random.randrange(0, 120), random.
randrange(0, 120), random.randrange(0, 120), random.randrange
(200, 255)))
```

LISTING 5.17: Zufallstext schreiben

Die Liste der Schriftarten wird bei Start des Skriptes einmalig mit Schriftarten gefüllt, die aus einem Ordner geladen werden:

```
fonts = []
for path in get_files_only("./dataGeneration/fonts"):
    fonts.append(ImageFont.truetype(str(path), 40))
```

LISTING 5.18: Schriftarten laden

Die hier verwendete Methode `get_files_only()` ist eine kurze Anweisung, die die Pfade zu allen Dateien in einem Ordner in eine Liste schreibt und Unterordner ignoriert.

```
def get_files_only(path):
    return [x for x in Path(str(path)).iterdir() if x.is_file()]
```

LISTING 5.19: get_files_only()-Methode

Zum Schluss wird das Blatt noch um einem Zufallswert gedreht, um schiefe Scans und verschobene Blätter zu simulieren.

```
    img = img.rotate(rotationAmount, expand=1)
```

LISTING 5.20: Bild drehen

Die Bilder werden vor dem Speichern direkt auf die Zielauflösung von 150 × 150 Pixeln skaliert.

```
imagesToGenerate = 100
for i in range(imagesToGenerate):
```

```
3     generateImage(470, 640, empty=True).resize(cfg.img_resolution,
    resample=Image.BILINEAR).save("./trainingData/empty/" + str(i) +
    ".jpeg")
    generateImage(470, 640, empty=False).resize(cfg.img_resolution,
    resample=Image.BILINEAR).save("./trainingData/printed/" + str(i
    ) + ".jpeg")
```

LISTING 5.21: Bilder skalieren und speichern

Aufgrund der hohen Anzahl an Trainingsbildern, stieg die Trainingszeit für 120 Epochen auf ca. 12*h* an. Es konnten allerdings erneut deutliche Verbesserungen in der Genauigkeit erzielt werden: Die Fehleranzahl bei den Validierungsdaten halbierte sich fast (siehe Tabelle 5.6).

Da das KNN in der Version nur mit den generierten Trainingsdaten trainiert wurde, können die für die vorherigen Versionen verwendeten Daten als Validierungsdaten verwendet werden.

Probleme hat das KNN vor allem mit Blättern, bei denen Ränder, Knicke oder ähnliche Verunreinigungen deutlich sichtbar sind. Bedenkt man, dass das Netz auf einem von Verunreinigungen freien Datenbestand trainiert wurde, ist das Problem ersichtlich. Mit sehr wenig Text kommt das KNN schon verhältnismäßig gut klar, in Einzelfällen werden sie aber auch falsch eingeordnet.

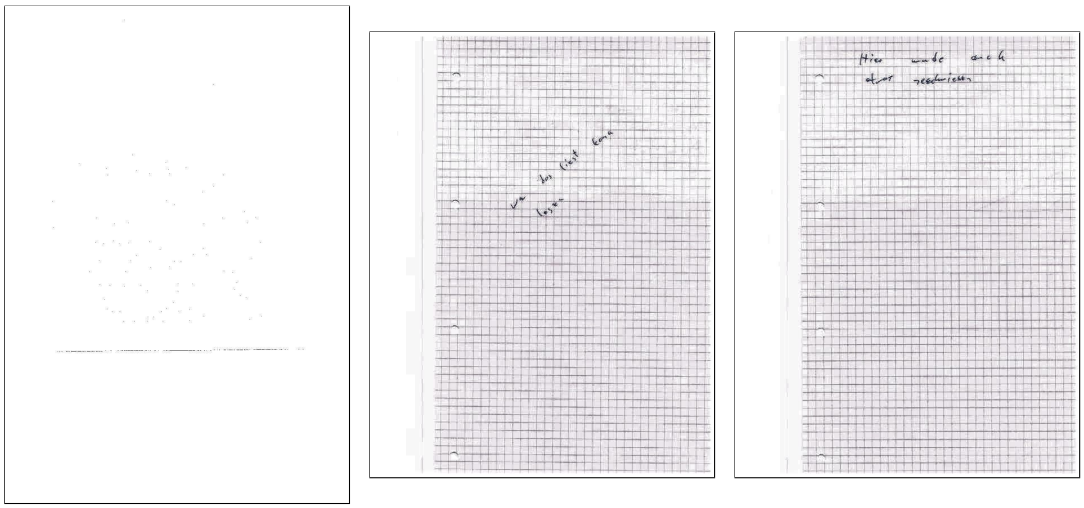

ABBILDUNG 5.14: Beispiele falsch klassifizierter Seiten (Kontrast zur besseren Darstellung erhöht)

ABBILDUNG 5.15: Beispiele generierter Trainingsbilder in Version 6

Trainingsbilder	10000
Validierungsbilder	240
Variationen	1
Epochen	120
Auflösung	150×150
Accuracy	97%
Probleme	Fehler bei erkennbaren eingescannten Rändern und bei Seiten mit wenig Text

TABELLE 5.6: Daten Versuch 2, Version 6

5.4 Onlineplattform

Damit das Tool zur Klassifizierung komfortabel verwendet werden kann, wird eine grafische Oberfläche benötigt. Die für den Benutzer angenehmste Lösung ist eine Onlineversion des Tools, da das Tool ohne Installation sofort verwendet werden kann. Außerdem muss ein Onlinetool nicht auf mehrere Betriebssysteme portiert werden. Die einzige Voraussetzung ist, dass der Benutzer einen aktuellen Webbrowser installiert hat. Eine Eigenentwicklung bietet die größte Flexibilität und das größte Skalierungspotenzial. Aus diesen Gründen wurde eine Online-Plattform basierend auf Flask und Bootstrap entwickelt. Das fertige Tool ist unter https://malscc.de zu finden.

5.4.1 Architektur

Da das KNN in Python implementiert ist, ist es am einfachsten, Python auch zur Bereitstellung der Webplattform zu verwenden. Flask ist ein minimales Webframework für Python, das diese Aufgabe übernehmen kann. Flask bringt alles was zum Betrieb eines Webservers notwendig ist mit, ist aber nur für eine Entwicklungsumgebung ausgelegt. Im Vergleich zu vollwertigen Webservern wie NGINX ist Flask sehr langsam. Außerdem kann Flask immer nur eine Anfrage gleichzeitig verarbeiten, was zu großen Problemen bei der Skalierung der Plattform führt. Für erste Testzwecke ist Flask aber völlig ausreichend.

Bei Eingang einer HTTP GET-Anfrage auf Port 80 führt Flask abhängig vom aufgerufenen Pfad unterschiedliche Methoden aus. Beim Aufrufen des Pfades / gibt die Anwendung den statischen Content der Seite aus. Das HTML wird dazu aus dem Ordner `templates` geladen (siehe Abb. 5.16). In dem `templates`-Ordner können `Jinja2`-Templates abgelegt werden, in diesem Fall reicht aber eine statische HTML-Datei aus. Die restlichen Abhängigkeiten liegen im Ordner `static` und werden bei Aufruf der Seite nachgeladen.

```
/
├── static
│   ├── css
│   ├── img
│   ├── js
│   ├── scss
│   ├── uploads
│   │   ├── 03b0d5f1-8f68-459c-a2d2-7a9f36b1e232
│   │   │   ├── downloads
│   │   │   │   └── 1d439f55-0930-4edd-b7fe-85f7.zip
│   │   │   ├── pdf-files
│   │   │   │   └── 51757767-5832-43b4-a484-e3af76f2788b.pdf
│   │   │   ├── preview
│   │   │   │   ├── 51757767-5832-43b4-a484-e3af76f2788b$001.jpeg
│   │   │   │   ├── 51757767-5832-43b4-a484-e3af76f2788b$002.jpeg
│   │   │   │   └── 51757767-5832-43b4-a484-e3af76f2788b$003.jpeg
│   │   │   ├── 51757767-5832-43b4-a484-e3af76f2788b$001.jpeg
│   │   │   ├── 51757767-5832-43b4-a484-e3af76f2788b$002.jpeg
│   │   │   ├── 51757767-5832-43b4-a484-e3af76f2788b$003.jpeg
│   │   │   └── processed
│   │   │       ├── 51757767-5832-43b4-a484-e3af76f2788b$001.jpeg+_0_5038.jpeg
│   │   │       ├── 51757767-5832-43b4-a484-e3af76f2788b$002.jpeg+_0_9198.jpeg
│   │   │       └── 51757767-5832-43b4-a484-e3af76f2788b$003.jpeg+_0_9414.jpeg
│   │   └── f4c6b2e6-bd2b-4f3f-9495-50078fb305e3
│   └── vendor
└── templates
    └── index.html
```

ABBILDUNG 5.16: Ordnerstruktur der Webplattform (vereinfacht)

5.4.2 Benutzersessions

Um die hochgeladenen Dateien den einzelnen Benutzern zuordnern zu können, wird für jeden Benutzer eine Session erstellt, die so lange aktiv ist, wie der Nutzer die Seite geöffnet hat. Damit der Server sicher erkennen kann, dass ein Benutzer die Seite verlassen hat, sendet der Client per AJAX in regelmäßigen Abständen eine Anfrage an den Pfad `/alive`. Sobald der Server über einen längeren Zeitraum keine Nachricht des Clients erhalten hat, wird die Session

als ungültig markiert und alle hochgeladenen Dateien der Session gelöscht. Jede Session hat eine eindeutige UUID, die auf dem Dateisystem verwendet wird, um die hochgeladenen Dateien zu ordnen.

5.4.3 Dateiverwaltung

Für jede Session wird ein Ordner erstellt, in den der Benutzer Dateien hochladen kann. Als Name wird die UUID der Session verwendet. Lädt der Benutzer Dateien hoch, werden diese per AJAX als POST-Request an den Pfad `/file-upload` geleitet und vom Server in dem für die Session erstellten Ordner abgelegt. Jede Datei bekommt dabei eine UUID als Dateiname, um eine Zuordnung zu ermöglichen und Exploits über den Dateinamen zu verhindern. PDF-Dateien werden automatisch in den Unterordner `pdf-files` abgelegt. Nach der Vorverarbeitung werden die Bilder in den Ordner `processed` gespeichert. Damit Bilddateien den ursprünglichen PDF-Dateien zugeordnet werden können, wird die UUID der PDF-Datei Teil des Dateinamens der dazugehörigen Bilddateien.

5.4.4 Klassifizierung

Betätigt der Nutzer den Button zum Starten der Verarbeitung, wird per AJAX eine Anfrage an den Pfad `/start-processing` gesendet. Der Server konvertiert dann alle hochgeladenen PDF-Dateien aus dem Ordner mit Ghostscript zu jpeg-Dateien. Anschließend werden alle Bilder normalisiert, auf 150×150 Pixel skaliert und im Ordner `processed` abgelegt. Die abgelegten Bilder werden anschließend von dem KNN klasifiziert. Nach der Klassifizierung gibt der Webserver als Antwort auf die ursprüngliche AJAX-Anfrage die URLs zu den Vorschaubildern gemeinsam mit der jeweiligen Klasse zurück. Sobald der Client die Antwort auf die Anfrage erhalten hat, werden per Javascript die Vorschaubilder nachgeladen und aufbereitet angezeigt.

5.4.5 Download der verarbeiteten Dateien

Der Nutzer hat die Möglichkeit Fehler, die das KNN gemacht hat, per Knopfdruck zu korrigieren. Betätigt der Nutzer den Button eines Bildes, um das Bilder in die andere Kategorie einzuordnen, werden die CSS-Klassen des Moduls getauscht, sodass das Bild in der anderen Liste erscheint. Die Sortierknöpfe sorgen dafür, dass nur Elemente mit bestimmten CSS-Klassen eingeblendet werden. Die Änderungen an den CSS-Klassen müssen dem Server mitgeteilt werden. Bei der Anfrage zum Herunterladen der neu sortierten Daten, wird dem Server dazu die Liste der korrekten Bilder mitgeteilt. Der Server erstellt anschließend die sortierten PDF-Dateien bzw. Bilder. Wurden mehrere Dateien hochgeladen, werden diese in ein ZIP-Archiv verpackt. Der Download-Link wird als Antwort auf die Anfrage zurückgegeben, sodass der Client per Javascript den Download initiieren kann.

Kapitel 6

Zwischenfazit der bisherigen Ergebnisse

6.1 Wurden alle zum Entwicklungsbeginn gesetzten Ziele erfüllt?

Der Versuch, ein künstliches neuronales Netz zur Klassifizerung von Scandaten zu entwickeln ist gelungen. Die Validierungsgenauigkeit konnte durch Optimieren der Trainingsdaten von ca. 80% auf 97% erhöht werden. Das entwickelte künstliche neuronale Netz konnte erfolgreich in eine Onlineplattform integriert werden, sodass die Benutzung des Tools ohne weitere Kenntnisse und sehr komfortabel möglich ist.

6.2 Optimierungspotenzial

6.2.1 Verarbeitung der hochgeladenen Daten

Die hochgeladenen Dateien werden während der Verarbeitung mehrfach in unterschiedliche Formate umgewandelt. Für jedes dieser Formate wird eine Datei auf dem Dateisystem erstellt. Dadurch ist die Anzahl der IOPS, besonders bei vielen Benutzern, sehr hoch. Außerdem müssen die Daten mehrfach vom Datenträger geladen werden. Optimaler wäre, die Daten komplett im Arbeitsspeicher zu verarbeiten, solange die Ressourcen des Servers dies erlauben. Dadurch würden auch lange Ladezeiten zwischen den einzelnen Verarbeitungsschritten verhindert.

6.2.2 Sammeln von Trainingsdaten von Nutzern

Da die Nutzer falsch sortierte Seiten manuell umsortieren können, entsteht im produktiven Einsatz eine große Datenmenge manuell kontrollierter, sortierter Seiten. Erlaubt man den Benutzern ihre Daten zum besseren Training des KNNs einzureichen, kann man das KNN aufgrund der besseren Trainingsdaten regelmäßig neu trainieren und verbessern.

6.2.3 Sortieren der Ausgabe nach Eindeutigkeit der Klassifizierung

Statt ausschließlich die Klasse zum Client zu übertragen, könnte zusätzlich übertragen werden, wie sicher sich das KNN bei der Klasse ist, in die das KNN die Seite eingeordnet hat. Dann können die Seiten im Browser so sortiert werden, dass die Seiten, die am wahrscheinlichsten falsch sortiert wurden, am Anfang stehen. Das würde dem Benutzer ein schnelleres Korrigieren der Fehler ermöglichen.

6.2.4 Bessere Skalierung

In der aktuellen Version ist es schwer das System auf größere Nutzerzahlen zu skalieren. Neben den anderen Performance-Optimierungen muss das System für größere Nutzerzahlen modular aufgebaut werden. Eine Möglichkeit wäre, die einzelnen Verarbeitungsschritte in einzelne Programme auszulagern, sodass jeder Verarbeitungsschritt von einem auf den Schritt spezialisierten Server ausgeführt werden kann. Außerdem muss ein Load-Balancing-Server vorgeschaltet werden, der die Benutzer auf parallel arbeitende Server verteilt. Ob es sinnvoller ist pro Server nur einen Verarbeitungsschritt oder alle Verarbeitungsschritte auszuführen, muss durch weitere Studien überprüft werden.

6.2.5 Weboberfläche

Für das Nachladen der Vorschaubilder erstellt der Client für jedes Bild eine eigene HTTP-GET Anfrage. Bei großen Nutzerzahlen könnte die Anzahl der Anfragen problematisch für den Server werden. Es sollten für das Nachladen der Vorschaubilder also mehrere Anfragen gruppiert werden. Außerdem sollten nicht alle Vorschaubilder gleichzeitig geladen werden. Die Vorschaubilder sollten während der Benutzer nach unten scrollt dynamisch nachgeladen werden.

6.2.6 API

Da das System für die Verarbeitung großer Scanarchive verwendet werden soll, muss es eine Alternative zur grafischen Benutzeroberfläche geben. Die grafische Benutzeroberfläche ist für große Datenmengen nicht ausgelegt. Das Nachladen der Vorschaubilder beispielsweise braucht für große Datenmengen zu viele Ressourcen. Eine einfache Rest-API würde dieses Problem lösen und anderen Entwicklern ermöglichen, das Tool in ihre eigenen Programme zu integrieren.

6.2.7 Sicherheit

Die aktuelle Version beinhaltet nur minimale Sicherheitsvorkehrungen. Die hochgeladenen Dateien müssen noch auf viele weitere Exploits geprüft werden, bevor sie vom System verarbeitet werden. Außerdem ist Flask sowohl aus Performancegründen als auch aufgrund einiger Sicherheitslücken auf Produktivsystemen nicht für den produktiven Einsatz geeignet. Da das System aber schon aus Performancegründen auf einen dedizierten Webserver umgebaut werden muss, stellt das nachdem die Performanceoptimierungen vorgenommen wurden kein Problem mehr da.

Außerdem muss eine absichtliche oder unabsichtliche Überlastung des Systems verhindert werden. Es müssen Dateigrößenlimits und Seitenzahllimits festgelegt werden, damit es nicht möglich ist den Server durch Hochladen übergroßer PDF-Dateien zu überlasten.

6.2.8 Finanzierung

Um den Betrieb der Server zu finanzieren, müssen Möglichkeiten gefunden werden das Tool zu monetarisieren. Die fairste Möglichkeit ist eine Einschränkung der Ressourcen, sodass die Server von den Nutzern, die die Server stark belasten, finanziert werden. Im Optimalfall wird das kostenlose Nutzungskontingent dynamisch an die aktuelle Benutzerzahl angepasst. So wird sichergestellt, dass der Server immer voll ausgelastet ist und den zahlenden Nutzern immer ausreichend Ressourcen zugewiesen werden können.

Um zahlende Nutzer zu verwalten, können API-Keys verwendet werden, die beim Erwerb von Nutzungskontingent an die zahlenden Nutzer verteilt werden. Die einfachste Möglichkeit den Zahlungsverkehr zu verwalten, wäre eine Möglichkeit im Webinterface zu bieten sich einen API-Key zu generieren. Dem API-Key kann standardmäßig ein kostenloses Nutzungskontingent zugewiesen werden. Neben dem API-Key wird eine Zahlungsadresse einer beliebigen Kryptowährung generiert. Der API-Key wird in einer Datenbank mit der Zahlungsadresse der Kryptowährung verknüpft und eingehende Zahlungen auf das Wallet überwacht. Anhand der verwendeten Zahlungsadresse der eingehenden Transaktion kann dem API-Key Nutzungskontingent gutgeschrieben werden. Neben der Verwaltung des Zahlungsverkehrs durch Kryptowährungen können auch Dienstleister zur Verwaltung der Transaktionen verwendet werden, aber Kryptowährungen bieten den Vorteil der schnellen Verarbeitung und geringen bzw. nicht vorhandenen Transaktionsgebühren.

Kapitel 7

Portierung der Online-Plattform

7.1 Gründe für die Portierung der Onlineplattform in JavaScript

Kurz nachdem die Entwicklung der auf Flask basierenden Onlineplattform abgeschlossen war, wurde TensorFlow.js[1] veröffentlicht. TensorFlow.js ermöglicht die Ausführung existierender TensorFlow-Graphen mithilfe von JavaScript und WebGL im Webbrowser (vgl. *TensorFlow.js* 2018). Zusätzlich können auch neue Graphen entwickelt und trainiert werden. Für die Verarbeitung der Nutzerdaten muss also kein Server mehr bereitgestellt werden. Stattdessen kann die rechenintensive Ausführung des TensorFlow-Graphen komplett dem Client überlassen werden. Schafft man es auch alle anderen Aufgaben, die zuvor vom Server übernommen wurden, vom Client ausführen zu lassen, bietet die Portierung auch große Datenschutz-Vorteile: Die Scandaten, die möglicherweise sensible Daten enthalten, werden nicht mehr über das Internet übertragen, sondern verbleiben komplett auf dem Computer des Nutzers. Das bietet zusätzlich Sicherheitsvorteile für Client- und Server, da der Server keine Nutzerdaten mehr verarbeiten muss, sondern nur noch eine statische Webseite ausliefern muss.

7.2 Überblick über die Funktionsweise der Applikation

7.2.1 Grundgerüst der Applikation

Die Webanwendung wurde auf Basis von Node.js entwickelt. Node.js ermöglicht die Verwendung von JavaScript in der Backendentwicklung. Da die Anwendung aber ohne Backend als statische Webseite funktionieren soll, waren nicht die Möglichkeiten in der Backend-Entwicklung für eine Verwendung der Plattform ausschlaggebend. Node.js bietet aufgrund der Paketverwaltung *npm* eine angenehme Entwicklungsumgebung, da Libraries komfortabel heruntergeladen und installiert werden können. Die Vielzahl der Module für die Plattform ist ein besonders großer Vorteil der Plattform. Als minimalistisches Webframework wurde Express[2] verwendet.

[1] https://js.tensorflow.org/
[2] https://expressjs.com/

Um Fehler zu vermeiden, die Entwicklung für verschiedene Webbrowser zu vereinfachen und Redundanz zu vermeiden, empfiehlt sich der Einsatz von Präprozessoren und Template-Engines.

Jade

Jade ist eine Template-Engine, die für eine strikte Trennung von Front- und Backend verwendet werden kann. Hier wird sie aber aufgrund des im Gegensatz zu HTML deutlich angenehmeren Syntax und aufgrund der dadurch gesteigerten Entwicklungsgeschwindigkeit verwendet.

Sass

Die Stylesheet-Sprache Sass, kurz für *Syntacticly Awesome Stylesheets*, ist eine Erweiterung des CSS-Standards. Unter anderem erlaubt Sass die Verwendung von Variablen und verringert somit die Redundanz des Quellcodes. Der Sass-Präprozessor wandelt den Code in normales CSS um, das mit allen Webbrowsern kompatibel ist.

Babel

Babel ist ein Präprozessor für JavaScript. Das Modul erlaubt die Verwendung von modernen JavaScript-Standards, die in vielen Webbrowsern noch gar nicht unterstützt werden. Hier wird Babel verwendet um Code nach dem ES2017-Standard verwenden zu können. Mit älteren Browsern nicht kompatible Befehle wie `await` werden mithilfe von Babel Polyfill durch kompatiblen Code ersetzt.

Browserify

Browserify erlaubt die Verwendung der meisten Node.js-Module durch den Befehl `require` im Frontend. Während des Build-Vorgangs erstellt Browserify eine große bundle.js-Datei, die alle Node-Module inklusive Abhängigkeiten und das Frontend-JavaScript beinhaltet. Somit muss trotz übersichtlicher Einteilung des Programmcodes in verschiedene Dateien nur eine durch Browserify generierte JavaScript-Datei eingebunden werden.

7.3 Funktionsweise der Plattform im Detail

Dieser Abschnitt soll einen detaillierten Einblick in die Funktionsweise der Plattform bieten. Der Abschnitt ist wichtig, wenn man auch Details der Funktionsweise nachvollziehen möchte, kann aber für einen groben Überblick auch übersprungen werden.

7.3.1 Dateien von der Festplatte laden

Da die Plattform komplett lokal arbeiten soll, ist ein klassischer Dateiupload an einen Server nicht nötig. Trotzdem soll die Plattform dem Nutzer eine gewohnte Benutzererfahrung mit einem Drag&Drop-Bereich und einem Knopf zum Hochladen von Dateien bieten. Die `FileReader` API von HTML5 bietet eine Möglichkeit Dateien von der Festplatte des Nutzers zu laden (vgl. Bidelman 2010).

Sobald das Dokument fertig geladen ist, wird geprüft ob der Webbrowser die benötigten APIs unterstützt. Anschließend werden die benötigten Event-Listener registriert, die auf Drag&Drop-Events und hinzugefügte Dateien reagieren. Da jQuery standardmäßig Events ohne `dataTransfer`-Attribut erstellt, ist der Einsatz klassischer JavaScript EventListener für die Drag&Drop Events sinnvoller.

```
1     if (window.File && window.FileReader && window.FileList &&
    window.Blob) {
          var dropZone = document.getElementById("dropZone");
3         dropZone.addEventListener("dragover", handleDragOver);
          dropZone.addEventListener("drop", handleFileDropped);
5         $("#browseFiles").bind("change", handleFileBrowse);
      } else {
7         alert("The File APIs are not fully supported in this
    browser.");
      }
```

LISTING 7.1: EventListener hinzufügen

Der `dragover`-EventListener ruft nur eine Methode auf, die die Darstellung des Cursors verändert, um dem Nutzer zu verdeutlichen, dass er die Datei in dem Bereich ablegen kann.

```
1 function handleDragOver(evt) {
      evt.stopPropagation();
3     evt.preventDefault();
      evt.dataTransfer.dropEffect = "copy"; // Explicitly show this
    is a copy.
5 }
```

LISTING 7.2: dragover-Event

Wird eine Datei in dem dafür vorgesehenen Bereich fallengelassen, wird eine Methode aufgerufen, die eine asynchrone Methode zur Verarbeitung der hochgeladenen Dateien aufruft.

```
function handleFileDropped(evt) {
2     evt.stopPropagation();
      evt.preventDefault();
4     handleFileUpload(evt.dataTransfer.files); // FileList object
}
```

LISTING 7.3: drop-Event

Die Methode wird auch aufgerufen wenn sich die ausgewählten Dateien des `input type="file"` Formularelements ändern, hier allerdings mit einer anderen Dateiquelle.

```
function handleFileBrowse(evt) {
    evt.stopPropagation();
    evt.preventDefault();
    handleFileUpload(evt.target.files); // FileList object
}
```

LISTING 7.4: change-Event

Die Methode zur Verarbeitung der hochgeladenen Dateien arbeitet asynchron. Da die Verarbeitung abhängig von der Geschwindigkeit des Computers eine Weile dauern kann, würde sonst der Webbrowser abstürzen oder einfrieren. Durch die Implementierung einer asynchronen Sub-Methode, können alle Dateien parallel verarbeitet werden.

Abhängig vom Dateityp werden die Dateien unterschiedlich verarbeitet. Bilddateien werden durch die `loadImage()`-Methode in Base64 umgewandelt. Für jedes Bild wird ein neues Objekt erzeugt und zu einem globalen Array `images` hinzugefügt. Das Objekt speichert Metadaten über jedes Bild, unter anderem eine eindeutige Identifikationsnummer, damit weitere Verarbeitungsschritte programmiersprachenunabhängig auf das Bild referenzieren können.

PDF-Dateien werden als `ArrayBuffer` eingelesen und mit einigen Metadaten in ein weiteres Array gespeichert. Anschließend wird die asynchrone Methode `generatePDFThumbnails()` aufgerufen, die die Generierung von Vorschaubildern einzelner Seiten initiiert (genaueres in Abschnitt 7.3.4).

```
async function handleFileUpload(files) {
    waitingDialog.show("processing files...");
    await sleep(300);

    await Promise.all(Array.from(files).map(async (f) => {
        var data;
        if (f.type.match("image.*")) {
            // process image-files
            data = await loadImage(f);
            images.push(
        {
        "data" : data,
        "attribute" : "unprocessed",
        "filename" : f.name,
        "pdfThumbnail" : false,
        "pdf_uuid" : "",
        "pdfPage" : 0,
        "uuid" : uuidv4()
        }
      );
        }
        else if (f.type.match("application/pdf")) {
            // process pdf files
            data = await loadPDF(f);
            var uuid = uuidv4(); // unique identifier for pdf
            pdfFiles.push(
        {
        "data" : data,
        "filename" : f.name,
```

```
30          "uuid" : uuid
            }
32        );
              await generatePDFThumbnails(data, f, uuid);
34          }
            else {
36              return;
            }
38          startFileAnimation();
      }));
40    await updateImageFilterCache();
      await updateShownImages();
42    waitingDialog.hide();
}
```

LISTING 7.5: Dateien verarbeiten

```
async function loadImage(file) {
2     return new Promise((resolve, reject) => {
          var reader = new FileReader();
4
          reader.onload = function(event) {
6             var data = event.target.result;
              resolve(data);
8         };

10        // Read in the image file as a data URL.
          reader.readAsDataURL(file);
12    });
}
```

LISTING 7.6: Bilder als Base64-String laden

```
async function loadPDF(file) {
2     return new Promise((resolve, reject) => {
          var reader = new FileReader();
4
          reader.onload = function(event) {
6             var data = event.target.result;
              resolve(data);
8         };

10        // Read in the pdf-file as an ArrayBuffer.
          reader.readAsArrayBuffer(file);
12    });
}
```

LISTING 7.7: PDF-Dateien als ArrayBuffer laden

7.3.2 Galerie

Bei vielen hochgeladenen Bildern bzw. PDF-Dateien ist die Galerie in mehrere Seiten aufgeteilt. Um das Wechseln der Seite zu beschleunigen, werden Referenzen auf alle Bilder, die den aktuellen Filterkriterien entsprechen, in ein Array zwischengespeichert. Somit müssen bei einem Seitenwechsel nicht

alle Bilder neu durchsucht werden. Ändern sich die Filterkriterien, wird die Cache-Variable durch die Methode `updateImageFilterCache()` aktualisiert. Die Methode blendet außerdem die Status-Icons in der Galerie ein bzw. aus.

```
function updateImageFilterCache() {
    if(currentFilter === "all") {
        currentFilterImages = images;
        $(".status-label").show();
    }
    else {
        currentFilterImages = getImagesWithAttribute(currentFilter)
    ;
        $(".status-label").hide();
    }
    updateShownImages();
}
```

LISTING 7.8: Aktualisierung des Filter-Caches

Wechselt der Nutzer die Seite, wird die Variable `page` angepasst und die Methode `updateShownImages()` aufgerufen, die die aktuell angezeigten Bilder aktualisiert.

```
    // previous page
    $(".page-switcher.left").click(function() {
        if(page>0) {
            page--;
            updateShownImages();
        }
    });

    // next page
    $(".page-switcher.right").click(function() {
        if((page+1)*imagesPerPage<currentFilterImages.length-1) {
            page++;
            updateShownImages();
        }
    });
```

LISTING 7.9: Wechsel der Galerie-Seite

Die Methode `updateShownImages()` berechnet welche Bilder derzeit angezeigt werden müssen und gibt diese an die Methode `showImagesInGallery()` weiter.

```
function updateShownImages() {
    var lowerBound = Math.min(page*imagesPerPage, images.length-1);
    showImagesInGallery(currentFilterImages.slice(lowerBound,
    lowerBound+imagesPerPage));
}
```

LISTING 7.10: Aktualisierung der angezeigten Bilder

Beim Aufruf der Webseite werden 24 Galerie-Elemente hinzugefügt und anschließend versteckt.

```
async function prepareDocument() {
    for(var i=0; i<imagesPerPage; i++) {
        $("#output-images-container .row").append(‘
```

```
            <div class="galleryElement col-lg-4 col-md-4 col-sm-4
    col-xs-6 filter unprocessed">
                <img class="img-responsive-gallery px-2">
                <div class="btn-switch-category-container">
                    <button class="btn-switch-category"><i class="
    fa fa-exchange animated"></i></button>
                </div>
                <div class="status-label">
                    <i class="fa fa-asterisk"></i>
                    <i class="fa fa-file-o"></i>
                    <i class="fa fa-file-text"></i>
                </div>
            </div>
        `);
    }
    showImagesInGallery([]);
}
```

LISTING 7.11: Vorbereitung der Webseite

Diese vorgefertigten Blöcke werden von der Methode `showImagesInGallery()` verwendet, um die Bilder anzuzeigen. Jedes Galerie-Element speichert die eindeutige Identifikationsnummer in einem `data-attribute`, sodass bei einem Klick des Knopfes für das Wechseln der Kategorie eine Verbindung zu dem Objekt im `images`-Array hergestellt werden kann. Alle nicht verwendeten Galerie-Elemente werden versteckt.

```
function showImagesInGallery(images) {
    var galleryElements = $(".galleryElement").toArray(); //
    available gallery-elements
    images.forEach(image => {
        $(galleryElements[0]).find("img").attr("src", image.data);
        $(galleryElements[0]).data("uuid", image.uuid);
        $(galleryElements[0]).removeClass("unprocessed printed
    empty");
        $(galleryElements[0]).addClass(image.attribute);
        $(galleryElements[0]).show(500);
        galleryElements.splice(0, 1); // remove from available
    gallery-elements
    });
    galleryElements.forEach(element => {
        $(element).hide(500);
    });
}
```

LISTING 7.12: Bilder in der Galerie anzeigen

7.3.3 Manuelle Klassifikation einzelner Seiten

Da das KNN in Einzelfällen eventuell falsch klassifizert, soll dem Benutzer die Möglichkeit gegeben werden einzelne Bilder manuell zu klassifizieren. Dafür hat jedes Galerie-Element einen Knopf. Betätigt der Nutzer den Knopf, wird das zugehörige Bild aus dem `images`-Array anhand der eindeutigen Identifikationsnummer gesucht und der aktuelle Status des Bildes geändert.

```
    // switch category
    $("#output-images-container .row").on("click", ".btn-switch-
    category", function () {
        var self = $(this).closest(".galleryElement");
        var imageObject = images.find(function(image) {
            return image.uuid === self.data("uuid");
        });
        if(imageObject !== undefined) {
            if (self.hasClass("empty")) {
                imageObject.attribute = "printed";
            }
            else if (self.hasClass("printed")) {
                imageObject.attribute = "empty";
            }
        } else {
            console.error("Image not found in DB!");
        }
        updateImageFilterCache();
    });
```

LISTING 7.13: Klassifizerung manuell korrigieren

7.3.4 Vorschaubilder der PDF-Dateien generieren

PDF-Dateien müssen für die Verarbeitung mit TensorFlow.js zuerst in Bilddateien umgewandelt werden. Außerdem sollen die Seiten der PDF-Dateien in der Galerie am Seitenende angezeigt werden. Das Rendern von PDF-Dateien im Browser, ausschließlich unter Verwendung von HTML, JavaScript und CSS, wird durch die Library PDF.js von Mozilla ermöglicht. Die Library soll eine Alternative zu sonst nötigen Browser-Plugins zum Lesen von PDF-Dateien bieten. Sie bietet aber auch Funktionen, um Vorschaubilder einzelner Seiten zu erstellen.

Die `generatePDFThumbnails()`-Methode ermittelt die Seitenzahl der PDF und ruft die ebenfalls asynchrone Methode `thumbnailFromPage()` auf. Die Methode erzeugt zuerst ein Canvas-Objekt mit den Abmessungen der PDF-Seite. Anschließend wird die Seite mithilfe von PDF.js auf das Canvas-Objekt gerendert. Im nächsten Schritt wird der Inhalt des Canvas-Objektes in Base64 kodiert und mit den Metadaten dem `images`-Array hinzugefügt.

```
async function generatePDFThumbnails(data, f, pdf_uuid) {
    var pdf = await pdfjsLib.getDocument(data);
    const totalPages = pdf.numPages;
    var promises = [];
    for(var i=1; i<=totalPages; i++) {
        promises.push(thumbnailFromPage(pdf, i, f, pdf_uuid));
    }
    await Promise.all(promises);
}

async function thumbnailFromPage(pdf, pageNumber, file, pdf_uuid) {
    var page = await pdf.getPage(pageNumber);
    var viewport = page.getViewport(1); //scale 1
    var canvas = document.createElement("canvas");
```

```
15      var context = canvas.getContext("2d");
        canvas.height = viewport.height;
17      canvas.width = viewport.width;
        var renderContext = {
19          canvasContext: context,
            viewport: viewport
21      };
        await page.render(renderContext);
23      var imageData = await canvas.toDataURL("image/jpeg");
        await images.push(
25      {
        "data" : imageData,
27      "attribute" : "unprocessed",
        "filename" : file.name + "_" + pageNumber,
29      "pdfThumbnail" : true,
        "pdf_uuid" : pdf_uuid,
31      "pdfPage" : pageNumber,
        "uuid" : uuidv4()
33      }
    );
35  }
```

LISTING 7.14: Vorschaubilder der PDF-Dateien erzeugen

7.3.5 Keras-Model ausführen

Wenn Dateien hochgeladen wurden, kann der Benutzer den Analysevorgang starten. Ein jQuery-EventListener reagiert auf das Betätigen eines Knopfes der die Verarbeitung startet mit einem Aufruf der `predictImages()`-Methode.

```
1       $(".startProcessing").click(function (e) {
            e.preventDefault();
3           predictImages();
        });
```

LISTING 7.15: EventListener Analysebeginn

Bevor TensorFlow.js Seiten klassifizieren kann, muss das in Kapitel 5 entwickelte Keras-Model in TensorFlow.js importiert werden. TensorFlow.js bietet hierfür ein Python-Modul, das die von TensorFlow.js benötigten Dateien exportiert:

```
    import tensorflowjs as tfjs
2   [...]
    tfjs.converters.save_keras_model(model, tfjs_target_dir)
```

LISTING 7.16: Export von Keras nach TensorFlow.js

Die exportierten Dateien werden, sobald der Nutzer die Seite aufruft, im Hintergrund asynchron heruntergeladen.

```
1   async function loadModel() {
        model = await tf.loadModel("model-data/model.json");
3   }
```

LISTING 7.17: Model asynchron herunterladen

Die Methode `predictImages()` ruft asynchron die Methode `predictImage()` auf, die dafür zuständig ist ein einzelnes Bild zu klassifizieren. Auch die Analyse der Bilder muss asynchron erfolgen, damit der Webbrowser nicht einfriert.

```
async function predictImages() {
    await waitingDialog.show("We are doing magic!");
    await sleep(500);
    await Promise.all(images.map(async (image) => {
        await predictImage(image)
    }));
    page = 0;
    await updateImageFilterCache();
    await waitingDialog.hide();
    await sleep(500);
}
```

LISTING 7.18: predictImages-Methode

Die Methode `predictImage()` ruft, bevor die Analyse beginnen kann, die Methode `prepareImageForPrediction()` auf, die das Bild vorverarbeit. Bevor das Bild als Input des KNNs verwendet werden kann, muss es auf 150×150 Pixel skaliert werden. Da das Bild nicht verzerrt werden soll, wird es auf einen schwarzen Hintergrund platziert.

```
function drawImageScaled(img, ctx) {
    var canvas = ctx.canvas;
    var hRatio = canvas.width  / img.width;
    var vRatio =  canvas.height / img.height;
    var ratio  = Math.min ( hRatio, vRatio );
    var centerShift_x = ( canvas.width - img.width*ratio ) / 2;
    var centerShift_y = ( canvas.height - img.height*ratio ) / 2;
    ctx.drawImage(img, 0,0, img.width, img.height,
                       centerShift_x,centerShift_y,img.width*ratio,
    img.height*ratio);
}
```

LISTING 7.19: Skalierung der Bilder

Das Ergebnis wird aus dem Canvas-Objekt als `ImageData` wieder ausgelesen.

```
async function prepareImageForPrediction(image) {
    var canvas = document.createElement("canvas");
    canvas.width = 150;
    canvas.height = 150;
    var ctx = canvas.getContext("2d");
    ctx.clearRect(0, 0, canvas.width, canvas.height)
    ctx.beginPath();
    ctx.rect(0, 0, canvas.width, canvas.height);
    ctx.fillStyle = "black";
    ctx.fill();
    var imgObject = await drawImageScaled(await base64ToImage(image
    .data), ctx);
    return await ctx.getImageData(0, 0, canvas.width, canvas.height
    );
}
```

LISTING 7.20: Vorbereitung der Bilder für TensorFlow.js

Nachdem die Vorverarbeitung abgeschlossen ist, wird mit TensorFlow.js ein Tensor aus den Pixeln erzeugt und auf die nötigen Dimensionen expandiert. TensorFlow.js führt anschließend das zuvor importierte Model mit dem Tensor als Eingabe aus. Je nach Ergebnis werden die Metadaten des Bildes in der Datenbank geändert.

```
async function predictImage(image) {
    var img = tf.fromPixels(await prepareImageForPrediction(image),
    3);
    img = tf.cast(tf.expandDims(img), "float32"); // expand to rank
    4 and cast to float
    const prediction = model.predict(img).dataSync()[0];
    if (prediction === 0) {
        image.attribute = "empty";
    } else {
        image.attribute = "printed";
    }
```

LISTING 7.21: predictImage-Methode

7.3.6 ZIP-Datei generieren und Dateien zum Download anbieten

Nach der Klassifizierung der Scandaten, soll der Nutzer die verarbeiteten Daten wieder herunterladen können. Da meist mehrere Dateien hochgeladen werden, sollen die Dateien zuerst in einem ZIP-Archiv zusammengefasst werden, bevor anschließend der Download gestartet wird. Zuerst wird geprüft, ob noch unverarbeitete Seiten vorliegen. Sollten noch unverarbeite Seiten vorliegen, werden diese vor dem nächsten Schritt erst analysiert. Um das ZIP-Archiv per JavaScript zu generieren, bietet sich die Library JSZip[3] an. Mithilfe der Library wird zuerst ein Objekt erzeugt das die spätere ZIP-Datei repräsentiert. Anschließend werden die PDF-Dateien generiert und gemeinsam mit den hochgeladenen Bildern dem Archiv hinzugefügt. Auch diese Schritte werden asynchron ausgeführt, um Performanceprobleme zu vermeiden. Abschließend wird aus der zuvor festgelegten Struktur eine PDF-Datei generiert, die dem Nutzer zum Download angeboten wird. Technisch gesehen handelt es sich natürlich nicht um einen klassischen Download, sondern vielmehr um ein Speichern der Dateien aus dem Arbeitsspeicher auf die Festplatte.

```
async function generateDownload(attribute) {
    // images to download
    if(getImagesWithAttribute("unprocessed").length>0) {
        await predictImages();
    }

    //get images, exclude pdf-thumbnails
    var imgs = getImagesWithAttribute(attribute, true);

    waitingDialog.show("We are preparing your download!");
    await sleep(300);
    var zip = new JSZip();
```

[3] https://stuk.github.io/jszip/

```
14      // generate & add pdfs
        var pdfs = await generatePDFs(attribute);
16      await Promise.all(pdfs.map(async (pdf) => {
            zip.file(pdf.name, await loadPDF(pdf), {binary: true})
18      }));

20      // add images
        await Promise.all(imgs.map(async (img) => {
22          zip.file(img.filename, img.data.split(",")[1], {base64:
    true});
        }));
24
        // start download
26      zip.generateAsync({type:"blob"})
        .then(function (blob) {
28          saveAs(blob, attribute + ".zip");
        }).then(function() {
30          waitingDialog.hide();
        });
32  }
```

LISTING 7.22: ZIP-Dateien generieren

7.3.7 PDF-Seiten entnehmen

Bevor die verarbeiteten Dateien heruntergeladen werden können, müssen allen PDF-Dateien die leeren bzw. beschriebenen Seiten entnommen werden. Um den Prozess zu vereinfachen, wurde die Library PDF Assembler[4] verwendet. Die Library extrahiert die logische Struktur der PDF-Dateien und wandelt sie in JavaScript-Objekte um. Diese Objekte lassen sich leicht bearbeiten. Anschließend setzt die Library die PDF-Datei wieder aus der veränderten Struktur zusammen. Da sich die Library noch in der frühen Entwicklungsphase befindet, können komplexe PDF-Features eventuell bei der Verarbeitung verloren gehen. PDF-Dateien von Scandaten sind aber meist sehr simpel aufgebaut, weshalb das Problem vernachlässigbar ist.

```
1   async function generatePDFs(attribute) {
        var out_pdfs = [];
3       await Promise.all(pdfFiles.map(async (f) => {
            var pdfAssemblerObj = new PDFAssembler(f.data);
5           var pdf = await pdfAssemblerObj.pdfObject;
            const thumbnails = getThumbnailsOfPDF(f, attribute);
7           var pages = pdf["/Root"]["/Pages"]["/Kids"];

9       // keep only pages of category
            pages = getSubArrayFromIndices(pages, thumbnails.map(a => a
    .pdfPage-1));
11          pdf["/Root"]["/Pages"]["/Kids"] = pages;

13          await pdfAssemblerObj.removeRootEntries();
            out_pdfs.push(await pdfAssemblerObj.assemblePdf(f.filename)
    );
```

[4] https://github.com/DevelopingMagic/pdfassembler

```
15      }));
        return out_pdfs;
17  }
```

LISTING 7.23: PDF-Dateien bearbeiten

Aufgrund des Umfangs des Programmtextes wurden einige Methoden hier nicht erläutert. Bei weiterem Interesse ist der komplette Quellcode jedoch unter https://github.com/lukas-mertens/malscc einsehbar.

Kapitel 8

Zusammenfassung der Ergebnisse

8.1 Zusammenfassung

Im Zuge dieser Arbeit wurde ein künstliches neuronales Netz entwickelt, das Scandaten in leere und beschriebene Seiten klassifizieren kann. Für das Training des KNNs wurde ein Programm entwickelt, das Scandaten prozedural generiert. Das KNN erreicht eine Genauigkeit von ca. 97% und kann somit in Verbindung mit einer manuellen Kontrolle produktiv eingesetzt werden. Zur benutzerfreundlichen Verwendung des KNNs wurden zwei Online-Plattformen für unterschiedliche Anwendungszwecke und Zielgruppen entwickelt.

8.2 Ergebnisse

Alle vor Projektbeginn gesetzten Ziele konnten erfüllt werden. Das Ergebnis der Entwicklungen ist eine Online-Plattform, die benutzerfreundlich Scandaten klassifizieren kann. Die Online-Plattform verarbeitet alle Daten direkt im Webbrowser durch Verwendung moderner Webtechnologien wie JavaScript und WebGL. Dadurch ist die Online-Plattform auch mit einer langsamen Internetverbindung nutzbar und der Schutz sensibler Scandaten ist gewährleistet. Gleichzeitig ist die Bereitstellung der Applikation sehr kostengünstig, da alle rechenintensiven Operationen auf dem Client-Computer ausgeführt werden.

Die auf Flask basierende Online-Plattform kann genutzt werden, wenn der Client-Computer zu wenige Ressourcen für eine Verarbeitung der Daten hat. Die Plattform kann beispielsweise für Mobilgeräte oder ältere Computer verwendet werden. Auch wenn der Nutzer die Klassifizierung größerer Dokumentensammlungen outsourcen möchte, ist die Python-Implementierung, in Verbindung mit einer API, die richtige Wahl.

8.3 Optimierungspotenzial

Neben dem Optimierungspotenzial der Python-Implementierung der Online-Plattform auf Basis von Flask (siehe Kapitel 6), hat auch die Neuimplementierung der Online-Plattform Optimierungspotenzial. Das KNN kann an vielen Stellen ebenfalls weiter optimiert werden.

8.3.1 hoher Arbeitsspeicherbedarf

Die JavaScript-Implementierung der Plattform hat im aktuellen Zustand einen sehr hohen Arbeitsspeicherbedarf. Da alle Bilder in Base64 konvertiert und als Strings gespeichert werden, ist der Speicherverbrauch größer als die Dateigröße der Bilder auf der Festplatte. PDF-Dateien werden sogar doppelt, einmal als Base64-String der PDF-Datei und einmal als Base64-Strings der einzelnen Seiten gespeichert. Somit können größere Dokumentensammlungen nicht durch die Online-Plattform verarbeitet werden.

Lösungsansatz

Das Problem kann behoben werden, indem alle Dateien nach dem Hochladen direkt verarbeitet werden. Sobald eine Datei klassifiziert wurde, wird sie automatisch wieder heruntergeladen, sodass sie aus dem Speicher entfernt werden kann. Das funktioniert aber nur, wenn es sich um viele kleine PDF-Dateien handelt. Eine sehr große PDF-Datei könnte trotzdem nicht verarbeitet werden, da diese vorerst vollständig in den Arbeitsspeicher geladen werden müsste.

Da der browserbasierte Ansatz keine Verarbeitung sehr großer Dateien erlaubt, könnte alternativ eine Desktopanwendung entwickelt werden, die Zugriff auf den lokalen Speicher hat und Daten zwischenspeichern kann. Das führt dazu, dass die Applikation auch deutlich größere PDF-Dateien verarbeiten kann.

8.3.2 langsamer Seitenwechsel

Wenn sich auf einer Seite der Galerie große Bilder befinden, kann der Seitenwechsel eine Weile dauern. Da das Wechseln der Seite nicht vollständig asynchron stattfindet, kann es dazu kommen, dass der Browser kurz einfriert. Da man zur Kontrolle der Analyseergebnisse schnell durch die Galerie blättert, ist das ein großes Problem.

Lösungsansatz

Das Problem könnte durch eine Generierung von kleineren Vorschaubildern der einzelnen Seiten gelöst werden. Somit würde die Dateigröße großer hochgeladener Bild- und PDF-Dateien verringert, was in einer kürzeren Ladezeit resultiert. Speichert man ausschließlich die verkleinerte Version des Bildes im Arbeitsspeicher, hat diese Taktik auch positive Auswirkungen auf die Arbeitsspeichernutzung. Ein Nachteil ist aber der zusätzliche Verarbeitungsschritt und die dadurch ansteigende Verarbeitungszeit beim Hochladen neuer Dateien.

8.3.3 Vorverarbeitung vor der Klassifizierung

Die Vorverarbeitung der Bilder ist beim Training und beim Vorhersagen etwas unterschiedlich. Dadurch entsteht ein Genauigkeitsverlust, der durch

eine überarbeitete Vorverarbeitung in der Online-Plattform behoben werden könnte.

8.3.4 Besseres Training des KNNs

Das KNN wurde bisher nur sehr rudimentär untersucht und optimiert. Durch eine genauere Untersuchung des KNNs beispielsweise mithilfe von Learning Curves, Attention-Maps oder durch Visualisierung der Filter der Convolutional Layer, würden sich sehr wahrscheinlich weitere Optimierungsmöglichkeiten aufzeigen. Aufgrund des begrenzten Zeitrahmens, war eine detaillierte Analyse allerdings nicht mehr möglich.

8.3.5 Transfer Learning

Wenn ein Nutzer die Klassifizierung einer Seite manuell korrigiert, können diese Daten für eine Verbesserung des KNNs eingesetzt werden. Da TensorFlow.js nicht nur die Ausführung, sondern auch das Training eines KNNs im Browser unterstützt, könnte ohne Übertragung der sensiblen Scandaten ein Training des KNNs stattfinden.

8.4 Ausblick

Die Applikation hat noch sehr viel Optimierungs- und Weiterentwicklungspotenzial. Ein erster wichtiger Schritt für die Weiterentwicklung war die Veröffentlichung des Quellcodes unter einer OpenSource-Lizenz. Ein weiterer wichtiger Schritt wird die Einrichtung einer besseren Entwicklungsumgebung mit CI-Build, TDD usw. sein.

Anknüpfend an die Forschungsfrage, ob es möglich ist leere von beschriebenen Seiten mithilfe von Machine Learning zu unterscheiden, kann eine Klassifizierung der beschriebenen Seiten in verschiedene Kategorien untersucht werden. Interessant wäre beispielsweise eine automatische Unterscheidung von Rechnungen, Angeboten, Werbung etc.

Literatur

Azevedo, Frederico AC u. a. (2009). „Equal numbers of neuronal and non-neuronal cells make the human brain an isometrically scaled-up primate brain“. In: *Journal of Comparative Neurology* 513.5, S. 532–541. URL: https://www.ncbi.nlm.nih.gov/pubmed/19226510.

Benenson, Rodrigo (2016). *What is the class of this image? - Discover the current state of the art in objects classification.* URL: https://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html.

Bidelman, Eric (2010). *Reading files in JavaScript using the File APIs*. URL: https://www.html5rocks.com/en/tutorials/file/dndfiles/ (besucht am 04. 06. 2018).

Chollet, Francois (2016). *Building powerful image classification models using very little data*. URL: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html (besucht am 16. 01. 2018).

Copeland, Michael (2016). *The Difference Between AI, Machine Learning, and Deep Learning? | NVIDIA Blog*. URL: https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/ (besucht am 21. 02. 2018).

Goodfellow, Ian, Yoshua Bengio und Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Gupta, Dishashree (2017). *Fundamentals of Deep Learning - Activation Functions and When to Use Them?* URL: https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/ (besucht am 18. 02. 2018).

Khan, Javed u. a. (2001). „Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks“. In: *Nature medicine* 7.6, S. 673. URL: https://www.nature.com/articles/nm0601_673 (besucht am 23. 05. 2018).

Kotikalapudi, Raghavendra und Mitwirkende (2017). *keras-vis*. https://github.com/raghakot/keras-vis.

LeCun, Yann, Corinna Cortes und Christopher J.C: Burges (2018). *THE MNIST DATABASE of handwritten digits*. URL: http://yann.lecun.com/exdb/mnist/ (besucht am 17. 02. 2018).

National Institute on Alcohol Abuse and Alcoholism (1997). *THE PRINCIPLES OF NERVE CELL COMMUNICATION*. URL: https://pubs.niaaa.nih.gov/publications/arh21-2/107.pdf (besucht am 27. 01. 2018).

Nedrich, Matt (2014). *An Introduction to Gradient Descent and Linear Regression*. URL: https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression/ (besucht am 25. 02. 2018).

Ng, Andrew (2018). *Machine Learning*. Hrsg. von Stanford University. URL: https://www.coursera.org/learn/machine-learning (besucht am 04. 03. 2018).

Nielsen, Michael A. (2015). *Neural Networks and Deep Learning*. URL: http://neuralnetworksanddeeplearning.com/ (besucht am 28. 01. 2018).

Poole, David, Alan Mackworth und Randy Goebel (1998). *Computational intelligence: a logical approach*. Oxford University Press.

Ray, Sunil (2017). *Essentials of Machine Learning Algorithms (with Python and R Codes)*. English. URL: https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/ (besucht am 15. 01. 2018).

Renelle, Tyler (2017). *Machine Learning Guide*. URL: http://ocdevel.com/podcasts/machine-learning/2.

Sanderson, Grant (2017). *But what *is* a Neural Network? | Chapter 1, deep learning*. URL: https://www.youtube.com/watch?v=aircAruvnKk (besucht am 25. 02. 2018).

Sharlene Katz Ph.D, P.E. (2004). *Introduction to Neural Networks for Senior Design*. Department of Electrical und Computer Engineering - California State University, Northridge. URL: https://www.csun.edu/~skatz/nn_proj/intro_nn.pdf (besucht am 25. 02. 2018).

Soares, Fabio M. und Alan M. F. Souza (2016). *Neural Network Programming with Java* -. Birmingham: Packt Publishing Ltd. ISBN: 978-1-787-12297-0.

TensorFlow (2017a). *Getting Started With TensorFlow*. URL: https://www.tensorflow.org/get_started/get_started (besucht am 16. 01. 2018).

— (2017b). *MNIST For ML Beginners*. URL: https://www.tensorflow.org/get_started/mnist/beginners (besucht am 16. 01. 2018).

— (2018a). *API Documentation*. URL: https://www.tensorflow.org/api_docs/ (besucht am 03. 03. 2018).

— (2018b). *TensorFlow Architecture*. URL: https://www.tensorflow.org/extend/architecture (besucht am 03. 03. 2018).

TensorFlow.js (2018). *A JavaScript library for training and deploying ML models in the browser and on Node.js*. URL: https://js.tensorflow.org/ (besucht am 04. 06. 2018).

Zitierweise

Alle Zitate und Belege wurden nach dem APA-Zitierstandard verfasst. Ein Großteil der Quellen ist als Beleg für Aussagen aus dem Allgemeinwissen zu verstehen. Wenn sich längere Abschnitte auf eine Quelle beziehen, wird immer nur im ersten Satz auf die Quelle verwiesen.

Selbstständigkeitserklärung

Ich erkläre, dass ich die Facharbeit ohne fremde Hilfe angefertigt und nur die im Literatur und Quellenverzeichnis angeführten Quellen und Hilfsmittel benutzt habe. Direkte und indirekte Übernahmen aus der (Sekundär-) Literatur habe ich im Literaturverzeichnis vollständig ausgewiesen.

Meschede, 10. Juni 2018

Ort, Datum

Lukas MERTENS